Eurographics Symposium on Rendering 2018
T. Hachisuka and W. Jakob
(Guest Editors)

Volume 37 (2018), Number 4

Progressive Multi-Jittered Sample Sequences

Per Christensen Andrew Kensler Charlie Kilpatrick

Pixar Animation Studios

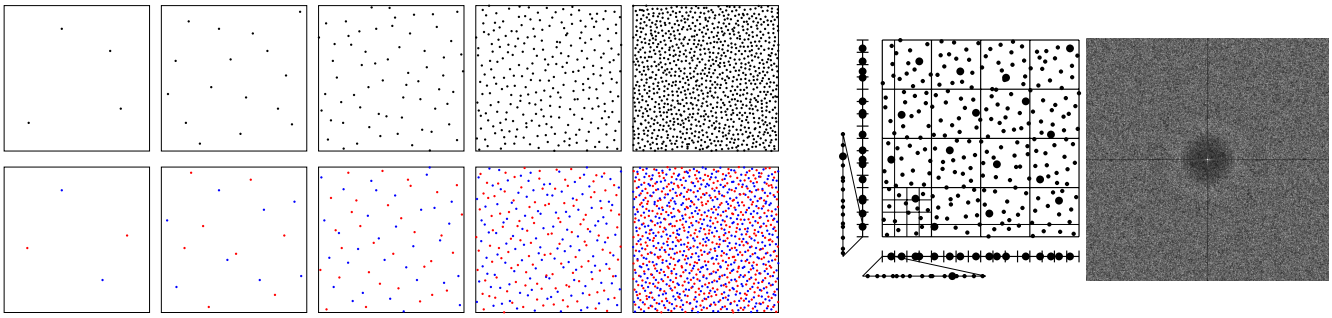

Figure 1: *Left, top row: the first 4, 16, 64, 256, and 1024 samples of a progressive multi-jittered sequence with blue noise properties. Left, bottom row: the first 4, 16, 64, 256, and 1024 samples from a progressive multi-jittered (0,2) sequence divided into two interleaved classes (blue and red dots) on the fly. Right: stratification and Fourier spectrum of the progressive multi-jittered sequence with blue noise properties.*

Abstract

We introduce three new families of stochastic algorithms to generate progressive 2D sample point sequences. This opens a general framework that researchers and practitioners may find useful when developing future sample sequences. Our best sequences have the same low sampling error as the best known sequence (a particular randomization of the Sobol' (0,2) sequence). The sample points are generated using a simple, diagonally alternating strategy that progressively fills in holes in increasingly fine stratifications. The sequences are progressive (hierarchical): any prefix is well distributed, making them suitable for incremental rendering and adaptive sampling. The first sample family is only jittered in 2D; we call it progressive jittered*. It is nearly identical to existing sample sequences. The second family is multi-jittered: the samples are stratified in both 1D and 2D; we call it* progressive multi-jittered*. The third family is stratified in all elementary intervals in base 2, hence we call it* progressive multi-jittered (0,2)*. We compare sampling error and convergence of our sequences with uniform random, best candidates, randomized quasi-random sequences (Halton and Sobol'), Ahmed's ART sequences, and Perrier's LDBN sequences. We test the sequences on function integration and in two settings that are typical for computer graphics: pixel sampling and area light sampling. Within this new framework we present variations that generate visually pleasing samples with blue noise spectra, and well-stratified interleaved multi-class samples; we also suggest possible future variations.*

1. Introduction

Sampling is used widely in rendering, particularly in Monte Carlo simulation and integration. A common goal is to optimize speed by rendering accurate images with as few samples as possible. In this paper we focus on progressive sequences for incremental rendering and adaptive sampling. Improvements in error or convergence can lead to significant speedups. Here we only consider 2D sample domains, which are common sub-domains in path tracing, for example for pixel sampling and area light sampling.

For efficient sampling we typically want an even distribution of the samples (no dense clumps of samples and no large regions without samples), but also no regular patterns. A common way to ensure even distribution is jittering, that is, dividing the sampling domain into regular, even-sized strata and placing one sample in each. Multi-jittering takes this one step further by ensuring stratification in both 2D and 1D. Another popular way of obtaining even distributions is to use quasi-random (qmc) patterns.

Sample patterns can be divided into two categories: finite, unordered sample *sets*, and infinite, ordered sample *sequences*. A *progressive (a.k.a. hierarchical or extensible) sample sequence* is a sequence where any prefix of the full sequence is well-distributed.

Using (finite) sample sets requires a-priori knowledge of how many samples will be taken, and yields high error if only a subset of those samples are used. This is fine for rendering final images with a fixed number of samples per pixel. But in several common settings – including adaptive sampling – we do not know in advance how many samples will be taken, or we are using incremental results during computation as in interactive rendering and off-line rendering writing check-point images. In these cases we need (infinite) progressive sample sequences.

Figure 2 shows a close-up of a penumbra region in a path-traced image. The image shows an incremental render after 100 (out of 400) samples per pixel have been taken. The image on the left was rendered with a non-progressive stratified sample set and has much more noise than the image on the right which was rendered in the same time with a progressive sample sequence. If a non-progressive set is used for incremental rendering, the image will remain excessively noisy until the last iterations of the rendering. Noise reduction from progressive sequences makes it easier to form an opinion about the final image sooner during interactive look development, and also gives faster termination of adaptive sampling.

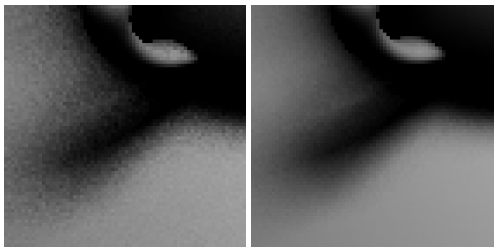

Figure 2: *Penumbra region with 100 samples per pixel. Left: non-progressive sample set. Right: progressive sample sequence.*

Many popular sample patterns are unfortunately non-progressive sets: jittered [Coo86, Mit96], Latin hypercube (also known as N-rooks or independent sampling) [Shi91], multi-jittered [CSW94], correlated multi-jittered [Ken13], Gaussian jittered [Coo86, SK12], uniform jittered [RAMN12], antithetic jittered [SK12], Hammersley [Ham60], Larcher-Pillichshammer [LP01], and golden ratio sets [SKD12]. Samples based on Penrose or polyhex tiles [ODJ04, WPC*14] need correction vectors that depend on the number of samples, so they are not strictly speaking progressive either. On the other hand, uniform random, best candidates [Mit91], Halton [Hal64], Sobol' [Sob67], adaptive regular tiling [ANHD17], and low-discrepancy blue-noise [PCX*18] sequences are progressive and hence well suited for incremental rendering and adaptive sampling.

In this paper we assess existing sample sequences and introduce new algorithms that generate progressive sample sequences with the following properties:

1. The samples are stratified in 2D squares only, in 2D squares and 1D rows and columns, or in all 2D elementary intervals.
2. The samples can be generated with improved nearest-neighbor distances (blue noise spectrum).
3. The samples give image noise that matches the best existing progressive sequences – including remarkably fast convergence for smooth pixels and illumination.

We call the sample sequences *progressive jittered*, *progressive multi-jittered*, and *progressive multi-jittered (0,2)* or shorter *pj*, *pmj*, and *pmj02*. Pj and pmj samples can be generated on the fly as needed, while pmj02 samples take longer to generate and should be tabulated before use.

When evaluating and comparing sample patterns, it is common to use the star discrepancy D^* as a measure of quality [Zar68]. However, like several authors [Shi91, Mit92, DEM96], we have found that D^* is a poor indicator of image quality. Instead we measure sampling error for integrating a few simple functions, and in settings more typical for computer graphics, and return to the topic of discrepancy in the supplemental material. Briefly, we find that an arbitrary-edge discrepancy is a more accurate error predictor than star discrepancy, but even the arbitrary-edge discrepancy does not measure the fact that some sequences perform much better than others when sampling smooth functions.

We evaluate the sample sequences by using them for pixel sampling and area light sampling, and comparing image quality and convergence rates. Our analyses of pixel sampling error have similarities to Mitchell's analysis of stratified sampling [Mit96]. He noted that there are three types of error convergence with stratified sample sets: complex pixels with many edges converge as $O(N^{-0.5})$, pixels with a few edges converge roughly as $O(N^{-0.75})$, and smooth pixels converge roughly as $O(N^{-1})$. We demonstrate that a few sample sequences have even faster convergence for smooth pixels, roughly $O(N^{-1.5})$. (We believe this fact is not yet fully appreciated and utilized in computer graphics.) Our analyses of light sampling strategies for square, rectangular, and disk lights are inspired by Ramamoorthi et al. [RAMN12]. Mitchell and Ramamoorthi used sample sets, while we focus on sample sequences.

2. Related work

The most relevant previous work falls mainly in two categories: multi-jittered sample sets and progressive sample sequences.

2.1. Multi-jittered sample sets

Our sample sequences build upon jittered [Coo86] and multi-jittered [CSW94] sample sets. Jittered samples are stratified in 2D. Multi-jittered samples are stratified in 2D like jittered samples and also in 1D like Latin hypercube samples. Multi-jittered samples can be generated by first placing samples in a canonical arrangement that is stratified in both 2D and 1D. Then the x coordinates of each column of 2D strata are shuffled, and the y coordinates of each row are shuffled. Kensler [Ken13] improved multi-jittering by applying the same shuffle to the x coordinates in each column and the same shuffle to the y coordinates in each row. This reduces the clumpiness of the sample points. (It also deals elegantly with non-square sample counts.)

Figure 3 shows 500 sample points from a regular grid, jittered, multi-jittered and Kensler's correlated multi-jittered sets. A regular grid has excellent distance between sample points (at least for square or nearly-square sample counts), but entire columns and rows of sample points are aligned and project onto the same points on the x and y axes; this is suboptimal when sampling a horizontal

or vertical object or shadow edge, and for elongated sample domains such as long skinny light sources. There is no difference in 2D distribution quality between jittered and multi-jittered samples, but the 1D projection is more uniform for multi-jittered samples.

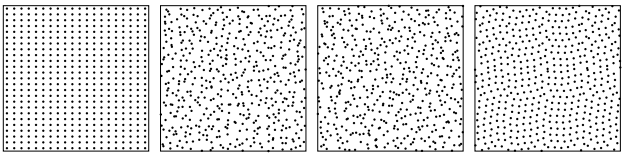

Figure 3: *500 samples from four non-progressive sample sets: regular grid, jittered, multi-jittered, and correlated multi-jittered.*

In order to generate samples from these four sets one needs to know the number of samples a priori; our progressive multi-jittered samples do not have this restriction.

2.2. Progressive sample sequences

Random. Uniform random 2D samples are generated by pairing two statistically independent pseudo-random numbers. They are typically generated with a linear congruential function [Knu98] such as drand48(), or with the Mersenne twister [MN98]. Random samples exhibit clumping and give high error and poor convergence.

Best candidates. Best-candidate ("blue noise") samples are an approximation to Poisson disk samples [Mit91, MF92]. Each new sample is found by generating a set of random candidate points, and picking the candidate point with the largest distance to the closest existing point. Best-candidate samples have nice spacing between points but uneven distribution overall [Shi91]. Although their absolute error is lower than for uniform random, their convergence rate is the same. Mitchell [Mit91] and Reinert et al. [RRSG15] used blue noise sets with improved projections.

Halton. The Halton sequence [Hal64] is a quasi-random sequence based on radical inverses in prime bases. Two-dimensional Halton sample points are constructed by combining numbers with different bases. Halton sequences typically have some clumping.

Sobol'. Sobol' [Sob67, PTVF92] introduced a family of quasi-random sample sequences. One particular 2D Sobol' sequence is a (0,2) sequence, i.e., it is stratified in all *elementary intervals* in base 2: $1 \times N$, $2 \times (N/2)$, ..., $N \times 1$. Elegant code to generate Sobol' (0,2) sequences can be found in Kollig and Keller [KK02], and on the web page of Grünschloß [Grü12]. (This is the sequence we mean when we refer to Sobol' in the remainder of this paper.) Despite its many merits, it unfortunately has systematic patterns, stripes of points aligned along diagonals, and many pairs and triplets of clumped points. Our progressive multi-jittered (0,2) sample sequences have the same elementary interval stratification, but do not have patterns, diagonals, or systematic clumping.

Blue noise and stratification. Recently, work has been done to combine blue noise and stratification. Ahmed et al. [ANHD17] introduced adaptive regular tiling (ART), an elegant and flexible tiling method to generate blue-noise sequences that also have some stratification (they are jittered due to the generation on a regular lattice, but not multi-jittered). The stratification reduces the unevenness and reduces sampling error, making their error comparable to most randomized quasi-random sequences. Some of our sequences are similar to ART, but have better stratification. Perrier et al. [PCX*18] modified the Sobol' sequence to get a blue noise spectrum; their LDBN sequences are stratified in all elementary intervals when the sample count is a power of 16, but for in-between sample counts their samples are not particularly evenly distributed.

Figure 4 shows 500 samples from these six sample sequences.

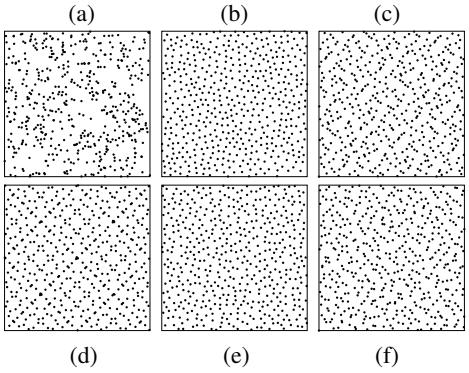

Figure 4: *First 500 samples from six progressive sequences: (a) uniform random; (b) best candidates; (c) Halton with bases 2 and 3; (d) Sobol' (0,2); (e) Ahmed ART; (f) Perrier LDBN.*

When quasi-random sequences are used in practice, they are often randomized to avoid aliasing [Kel12, Owe03]. One common randomization is toroidal shifts a.k.a. Cranley-Patterson rotations [CP76]. An other common randomization is random digit scrambling, which for the Sobol' sequence can be done with bit-wise xor [KK02] or Owen scrambling [Owe97]. Figure 5 shows 500 samples from rotated and scrambled Halton and Sobol' sequences. We do Owen scrambling efficiently with hashing – similar in spirit to Laine and Karras [LK11], but with a better hash function provided by Brent Burley.

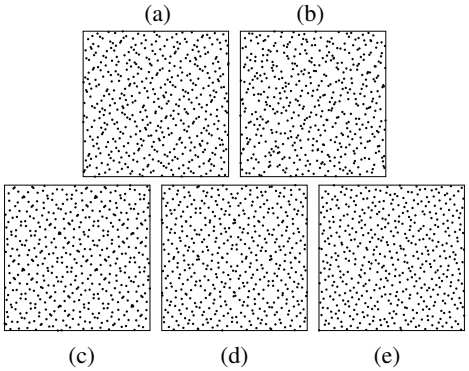

Figure 5: *First 500 samples from five randomized quasi-random sequences: (a) rotated Halton; (b) scrambled Halton; (c) rotated Sobol' (0,2); (d) xor-scrambled Sobol' (0,2); (e) Owen-scrambled Sobol' (0,2).*

3. Initial comparisons of common sample sequences

Comparing sample sequences with each other is a complex task with many fine nuances. Which sequence is best depends on the characteristics of the function being sampled and the error metric chosen, and sometimes there are no clear answers. In this section we attempt to provide an objective evaluation of the most popular sample sequences. We sample five simple test functions: disk, triangle, step, Gaussian, and bilinear. The sample domain is the unit square, and the correct values of the integral of these functions are known analytically. Although these are very simple functions, they are representative of sampling, for example, pixels with sharp edges or smooth textures, and area light sources with partial occlusion or smooth intensity change due to distance and cosine terms.

We sample each function with uniform random, best candidates, Ahmed ART, Perrier LDBN, Halton, and Sobol' sequences. The Halton sequence is randomized with rotations and random digit scrambling; the Sobol' sequence is randomized with rotations, bitwise xor, and Owen scrambling. The plots in the following tests show sampling error as a function of the number of samples; each curve is the average of 10000 trials.

3.1. Discontinuous functions

Disk. Our first test is a disk function: $f(x,y) = 1$ if $x^2 + y^2 < 2/\pi, 0$ otherwise. (The reference value is 0.5.) This function has no dominant direction; it does not favor stratification at any particular angle. The plot in Figure 6 (top) shows that for this discontinuous function, the error for uniform random and best candidates converges as $O(N^{-0.5})$, while the error for other sequences converges faster at roughly $O(N^{-0.75})$.

Triangle. The plot in Figure 6 (middle) shows sampling error for a triangle function (1 if $y > x$; reference value 0.5). The Sobol' sequence with rotations and xor scrambling has very high error; this is caused by the diagonal patterns evident in Figure 5(c) and (d). Perrier LDBN, Ahmed ART, both Halton sequences, and the Owen-scrambled Sobol' sequence converge nicely at roughly $O(N^{-0.75})$.

Step. Our next test function is a step function (1 if $x < 1/\pi$; reference value $1/\pi$). This is a test of projection of 2D samples to 1D: only the samples' x dimension matters. (Practical examples are sampling a light-aligned shadow edge, sampling long, skinny area lights, and using one dimension of 2D samples to select a light source [SWZ96].) The plot in Figure 6 (bottom) shows that Ahmed's ART sequence has high error (it is only jittered in 2D, not in 1D), and Perrier's LDBN sequence has high error except at 16, 256, and 4096 samples. The quasi-random sequences converge at roughly $O(N^{-1})$. The x component of the Halton and Sobol' sequences are identical, so those error curves overlap.

3.2. Smooth functions

Gaussian. For the first test of a smooth function we sample a 2D Gaussian function, $f(x,y) = e^{-x^2-y^2}$. The reference value is $\frac{\pi}{4}\text{erf}^2(1)$. We see three distinct convergence rates in Figure 7 (top): Random and best candidates still converge as $O(N^{-0.5})$. Ahmed ART sequences and nearly all combinations of quasi-random sequence and randomization converge faster at roughly $O(N^{-1})$. The

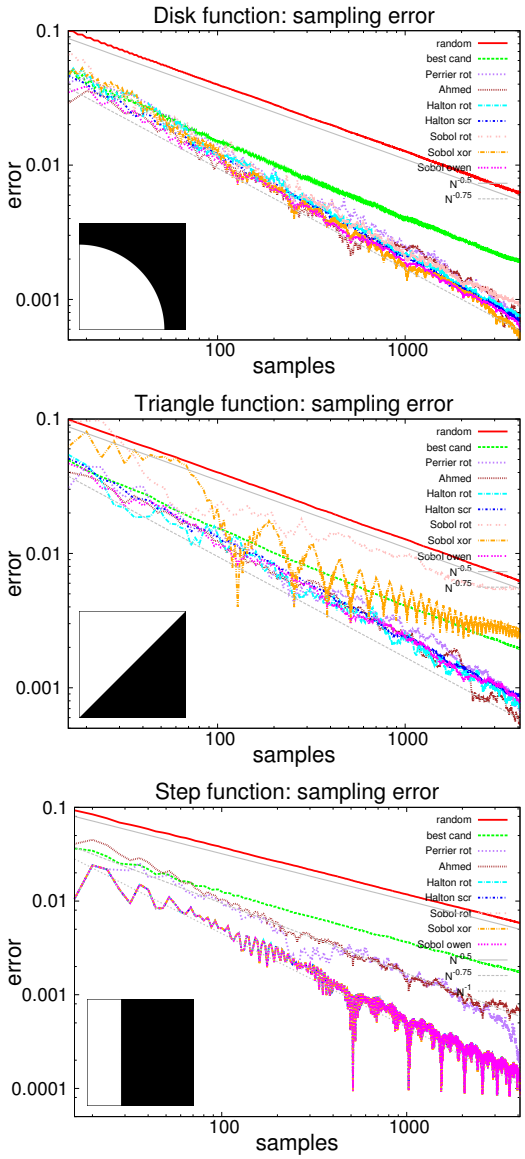

Figure 6: *Error for sampling disk, triangle, and step functions with 16–4096 samples using various sample sequences. Note the very high error for rotated and xor-scrambled Sobol' for the triangle function, and the high error for Ahmed's ART sequence for the step function.*

Sobol' sequence with Owen scrambling has even faster convergence: roughly $O(N^{-1.5})$ for power-of-two numbers of samples. This is because Owen-scrambled samples are randomly jittered within their strata [Owe97, Owe03] whereas rotated and xor scrambled quasi-random samples are at fixed equidistant grid positions in 1D. Owen-scrambled Sobol's fast convergence rate is remarkable: for example, for 1024 samples, random samples give an average error of 0.005388, xor-scrambled Sobol' gives error 0.000154, while Owen-scrambled Sobol' only gives error 0.000008 – a factor of 19.

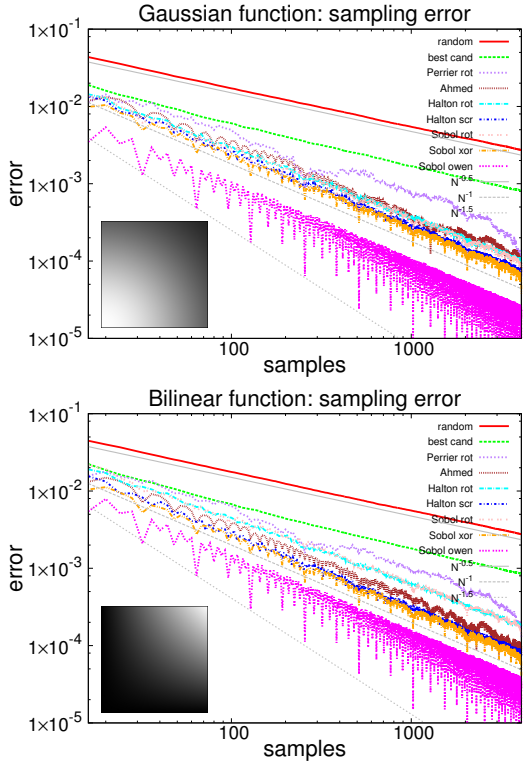

Figure 7: *Error for sampling 2D Gaussian and bilinear functions with various sample sequences. Note the deep "error valleys" for Owen-scrambled Sobol' when the number of samples is a power of two.*

Bilinear. For the bilinear function $f(x,y) = xy$ (reference value 0.25) we see results similar to the Gaussian function, as shown in Figure 7 (bottom). Again, Owen-scrambled Sobol' clearly has the fastest convergence. (Also, rotated Halton and Sobol' sequences have noticeably higher error than the other randomizations of these sequences.)

3.3. Summary

Our conclusion from these tests is that the Owen-scrambled Sobol' sequence is the best of the tested sequences: it does not have pathological behavior for sampling discontinuous functions at certain angles, and it has extraordinarily fast convergence for sampling of smooth functions. In Section 7 we will revisit function sampling.

4. Progressive jittered sequences

In order to describe our algorithm for generating progressive multi-jittered samples, it is instructive to first look at the slightly simpler case of progressive jittered (pj) samples without multi-jittering. Figure 8 shows an example of 4 samples stratified into 2×2 square cells, 8 samples, and 16 samples stratified into 4×4 square cells.

The 2D strata (cells) are iteratively subdivided and new samples are placed in a diagonally alternating order, thereby avoiding strata with previous samples while at the same time maintaining a

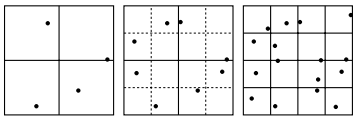

Figure 8: *4, 8, and 16 progressive jittered samples shown with their 2D strata.*

balanced distribution. In detail: (a) The first sample is placed completely at random in the unit square. (b) The unit square is divided into 4 quadrants and the second sample is placed in the diagonally opposite quadrant of the first. (c) The third sample is placed in one of the two remaining empty quadrants. (d) The fourth sample is placed in the last empty quadrant. (e) The unit square is divided into 16 sub-quadrants, and for each of the first four sample points a new sample point is placed in the diagonally opposite sub-quadrant in the same quadrant. (f) For each of the first four samples a new sample point is placed in one of the two empty sub-quadrants in the same quadrant. (g) Samples are placed in the last four empty sub-quadrants. And so on. These steps are illustrated in Figure 9. Pseudo-code in the supplemental material elaborates the algorithm.

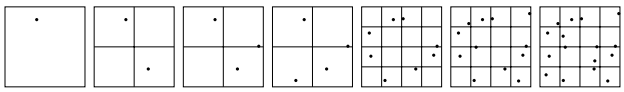

Figure 9: *The first 1, 2, 3, 4, 8, 12, and 16 sample points from a progressive jittered sequence.*

Each sample sequence with 1, 4, 16, 64, ... samples is stratified (jittered) as if we had simply generated that number of stratified samples, and sample sequences in-between are balanced as well: the number of samples in one quadrant of the unit square is at most 1 off from the number of samples in any other quadrant.

Apart from our preference of diagonal sample placement, this is very similar to the progressive jittered samples described by Dippé and Wold [DW85] and by Kajiya [Kaj86]. This progressive stratification scheme is so simple that it probably has been re-invented many times since Dippé's and Kajiya's papers.

We can generate 40 million progressive jittered samples per second using double-precision floats and drand48() for pseudo-random numbers. With a table of 1K pseudo-random numbers the speed goes up to 170 million samples per second. For comparison, we can generate 73 million pseudo-random samples per second using drand48(). These performance numbers are for a C++ implementation running on a single core of a 2.5 GHz Intel Xeon E5-2680 processor.

5. Progressive multi-jittered sequences

The multi-jittering extension of the algorithm is inspired by Chiu et al. [CSW94] and Kensler [Ken13]. Figure 10 shows 4, 8, and 16 samples that are stratified into square cells and row/column strips. As mentioned previously, 1D stratification is particularly important in cases where mainly one dimension of the samples is

used, for example for sampling a long skinny light source or if one dimension is used to choose between light sources [SWZ96].

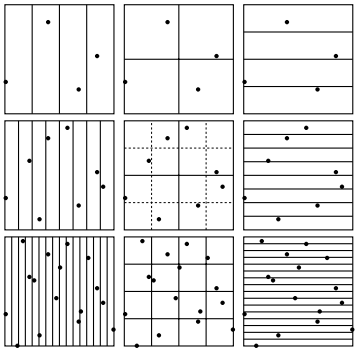

Figure 10: *4, 8, and 16 progressive multi-jittered samples with 1D and 2D strata.*

Progressive *multi*-jittering (pmj) is only slightly more complicated than progressive jittering – we just have to also keep track of which 1D strips (rows and columns) have been occupied by previous sample points and only place new sample points in unoccupied 1D strips. The 1D strips are subdivided on the fly as more samples are generated, just like the 2D cells. In detail: (a) The first sample is placed at random in the unit square. (b) The unit square is divided into 4 quadrants and the second sample is placed in the diagonally opposite quadrant of the first. (c) The square is divided into four rows and four columns. One of the two empty quadrants is chosen, and a point that is not in an occupied 1D strip is generated in it. (d) The fourth sample is placed in the last empty quadrant, again choosing a position that is not in one of the occupied rows or columns. (e) The unit square is divided into 16 sub-quadrants and 8 rows and columns. For each of the first four sample points a new sample point is placed in the diagonally opposite sub-quadrant in the same quadrant without conflicting with occupied rows or columns. (f) For each of the first four samples a new sample point is placed in one of the two empty sub-quadrants in the same quadrant (again avoiding occupied 1D strips). (g) Samples are placed in each of the last four empty sub-quadrants (while avoiding occupied 1D strips). And so on. These steps are illustrated in Figure 11. Pseudo-code in the supplemental material explains more details.

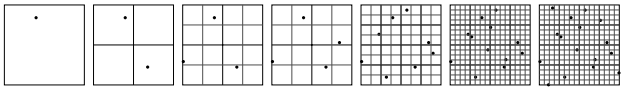

Figure 11: *The first 1, 2, 3, 4, 8, 12, and 16 sample points from a progressive multi-jittered sequence.*

For sequences of length 2^N, the pmj samples have the same full 2D and 1D stratification as Chiu et al.'s multi-jittered sets and Kensler's correlated multi-jittered sets.

An improvement of the sample distribution between powers of two can be achieved by replacing the random choice of one of the two empty subquadrants in step (f) with more well-balanced choices ensuring that the selected subquadrants are at most off by 1. Please refer to the pseudo-code in the supplemental material for details. This adds only a few percent to the sample generation time.

We can generate more than 11 million progressive multi-jittered samples per second (using a table of pseudo-random numbers rather than calling drand48()). For comparison, we can generate 38 million "raw", rotated, or xor-scrambled Sobol' samples per second, and around 7 million Owen-scrambled Sobol' samples per second.

6. Progressive multi-jittered (0,2) sequences

The final family of sample sequences generalizes the strata to all base 2 elementary intervals, as shown in Figure 12.

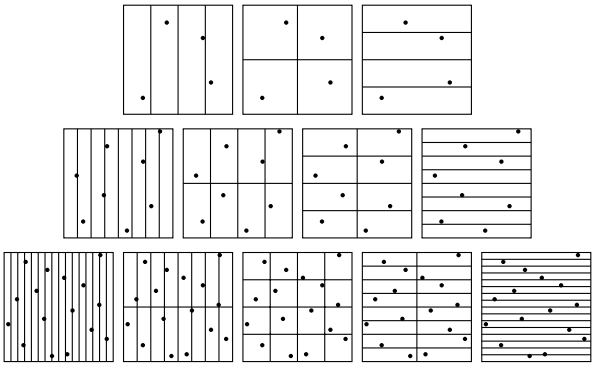

Figure 12: *4, 8, and 16 progressive multi-jittered (0,2) samples and their strata (base 2 elementary intervals).*

We have chosen to call this family of sequences "progressive multi-jittered (0,2)", or shorter: pmj02. (This name is admittedly rather redundant since any (0,2) sequence is inherently progressive and multi-jittered.)

The algorithm for generating pmj02 sequences is very similar to the algorithm for pmj sequences in the previous section, but once a candidate sample has been generated we check whether it falls into any 2D elementary interval that is already occupied by a sample. If so, a new candidate sample is stochastically generated and tested; we keep generating new candidates until one is found that is not in any previously occupied 2D elementary interval. This ensures that all power-of-two prefixes of the sequence are (0,m,2) nets and the full sequence is a (0,2) sequence. Pseudo-code can be found in the supplemental material.

Our pmj02 sequences have the same stratification as the Sobol' sequence. As we will see in the following sections, they also have the same advantageous stochastic jittering as Owen scrambling. Our algorithm also opens up the possibility of generating samples with a blue noise spectrum, as we'll show in Section 10.

We can generate around 21,000 progressive multi-jittered (0,2) samples per second using this simple brute-force trial-and-error approach. With a more optimized version that keeps track of unoccupied squares within each subquadrant we can generate 39,000 samples per second. Both approaches are too slow to generate samples

during rendering, so we tabulate these sequences. (We believe that further optimizations can increase the speed significantly.)

The samples generated by the algorithm described above have the same extremely low error for smooth functions as Owen-scrambled Sobol' at powers of two (full octaves). However, between odd and even powers of two (where we have the arbitrary choice of which empty sub-quadrant to choose first), the error is higher. A useful improvement of the pmj02 samples between octaves can be obtained by a simple change: choose sub-quadrants such that the new samples are themselves (0,2) sequences. For example, the samples 32 ... 47 and 48 ... 63 should themselves be (0,2) sequences. This yields sequences that fully match Owen-scrambled Sobol's error for all sample counts.

7. Results: revisiting sampling of simple functions

Here we revisit the five simple functions from Section 3 to see how the pj, pmj, and pmj02 sequences compare with the best of the tested sequences, Owen-scrambled Sobol'. We also show results for scrambled Halton for comparison.

7.1. Discontinuous functions

Disk. A new error plot for sampling the disk function is shown in Figure 13. All five sequences have the same convergence rate, roughly $O(N^{-0.75})$. The error curves for Owen-scrambled Sobol' and pmj02 mostly overlap; they are both slightly better than Halton, pj, and pmj.

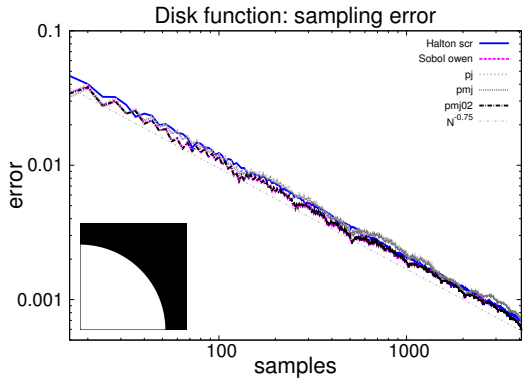

Figure 13: *Error for sampling of disk function. (The curves for Owen-scrambled Sobol' and pmj02 overlap.)*

Triangle. For the triangle function, pj, pmj, and pmj02 perform the same on average; there is no advantage in multi-jittering over regular jittering since the discontinuity in the function lies along the diagonal $y = x$. Hence the curves for Owen-scrambled Sobol', pj, pmj, and pmj02 all overlap. (Error plot omitted here.)

Step. For the step function, the pj sequence is just as bad as Ahmed's ART sequence (as shown in Figure 6 (bottom)), pmj is better, and pmj02 has the same low error as Owen-scrambled Sobol'. (Plot omitted.)

7.2. Smooth functions

Gaussian. An updated error plot for the Gaussian function is shown in Figure 14. The pj sequence is slightly better than Halton; both converge at $O(N^{-1})$. Owen-scrambled Sobol' and pmj02 have exactly the same error, including the $O(N^{-1.5})$ fast convergence at powers of two. Pmj also has fast convergence, but cannot quite compete with Owen-scrambled Sobol' and pmj02.

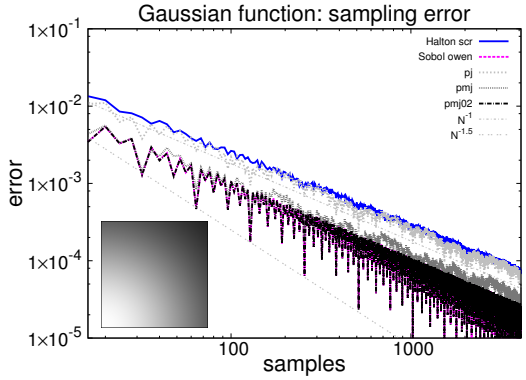

Figure 14: *Error for sampling the 2D Gaussian function. (The curves for Owen-scrambled Sobol' and pmj02 overlap.)*

Bilinear. Results for sampling the bilinear function are very similar to the results for the Gaussian function; hence not shown here.

8. Results: pixel sampling

In this section we test the sample sequences on pixel sampling of various discontinuous and smooth images. All images are rendered at resolution 400×400 or 400×300. Reference images are computed with $500^2 = 250000$ jittered samples per pixel. Each error curve is computed as the average of 100 sequences. In these tests we use a 1×1 box pixel filter, but in the supplemental material we repeat two of the tests using a Gaussian pixel filter, and find only a slight reduction in convergence rates.

8.1. Zone plate images

The zone plate function is often used to evaluate sample distributions and image sampling algorithms. The insets in Figure 15 show binary and smooth zone plate images. The curves show root-mean-square (rms) error for 25–2500 samples per pixel. The binary image pixels contain discontinuities at many different angles. For the binary zone plate function, the errors shown in Figure 15 (top) converge as $O(N^{-0.5})$ and $O(N^{-0.75})$, with rotated Sobol' having slightly higher error than other quasi-random sequences and pmj02.

The error curves in Figure 15 (bottom) show that, similar to the smooth simple functions, pmj02 and Owen-scrambled Sobol' have much lower error than other sequences: their rms error converges as roughly $O(N^{-1.5})$ at powers of two. The error for Ahmed's ART sequence is slightly higher than for the quasi-random sequences.

Rms error is the typical measure for image comparisons, but maximum error might be more meaningful when trying to eliminate fireflies. For completeness, we have repeated these tests using maximum error in the supplemental material.

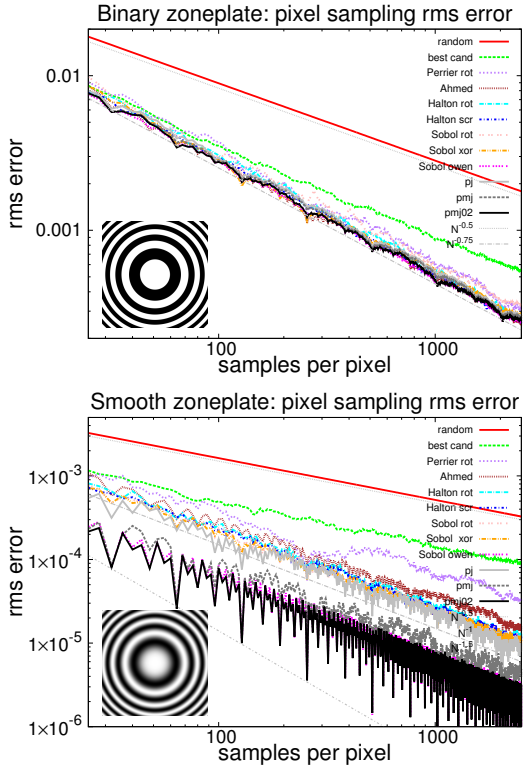

Figure 15: *rms error for pixel sampling of binary and smooth zone plate images. (25–2500 samples per pixel.)*

8.2. Checkered teapots

Figure 16 shows two checkered teapots on a checkered ground plane. This image has many pixels with discontinuities at various angles, similar to the binary zone plate image. Figure 17 shows rms error for the different progressive sequences. Similarly to what we found for the binary zone plate, most sequences have nearly the same error, except rotated Sobol' which has higher error.

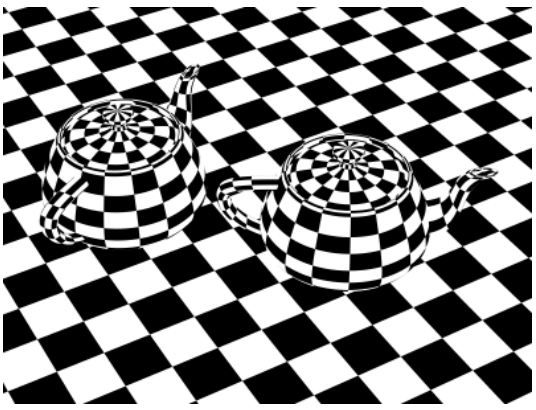

Figure 16: *Checkered teapots on a checkered ground plane.*

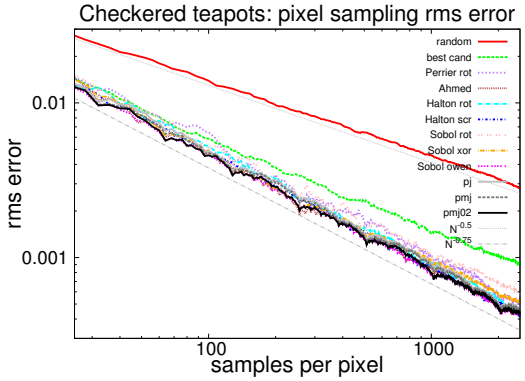

Figure 17: *rms error for pixel sampling of checkered teapots image.*

8.3. Textured teapots

In real production renderers, texture lookups at ray hit points are always filtered. This turns discrete texel colors into continuous functions. Figure 18 shows a version of the teapot scene with a grayscale texture map with a (truncated) Gaussian texture filter. The image is mostly smooth, but with discontinuities along silhouettes of the teapots.

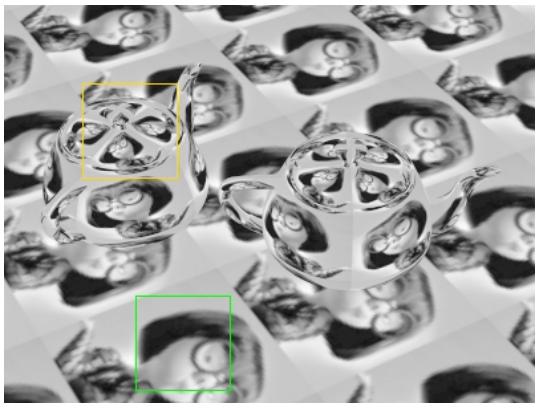

Figure 18: *Textured teapots on a textured ground plane. The yellow outline marks a 70×70 pixel region containing sharp object edges; the green outline marks a region with only smooth texture variation.*

Figure 19 (top) shows rms error for pixels in a region with sharp discontinuities due to object edges. The error is dominated by the pixels with discontinuities. All sequences have the expected $O(N^{-0.5})$ and $O(N^{-0.75})$ convergence rates. Figure 19 (bottom) shows error for pixels on the textured ground plane. The pixels in this region are all smooth and hence have very fast convergence. Most sequences converge at $O(N^{-1})$, while pmj02 and Owen-scrambled Sobol' converge faster at roughly $O(N^{-1.5})$. (A triangle texture filter makes the image function in each pixel less smooth and gives a $O(N^{-1.25})$ convergence rate. With a 1×1 box texture filter, the texel edges become pixel discontinuities, and the convergence rate drops to $O(N^{-0.75})$.)

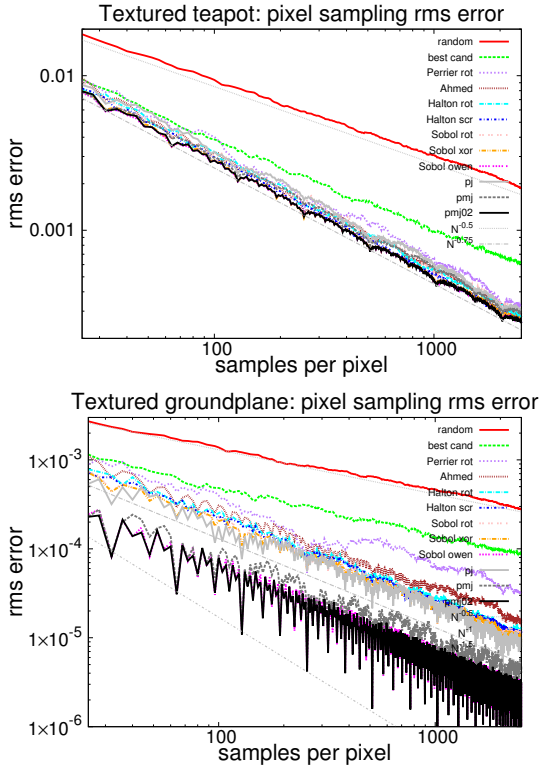

Figure 19: *rms error for pixel sampling of textured teapots and ground plane. Top: sharp edge region. Bottom: smooth texture region.*

9. Results: square area light source sampling

In this section we use the sequences for area light sampling. Figure 20 shows the same two teapots and ground plane, but now illuminated by a square area light source creating soft shadows and interesting penumbra regions. The light source is sampled with shadow rays shot from the surface point at the center of each pixel.

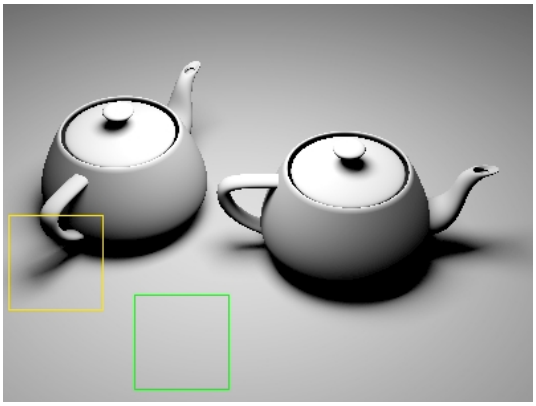

Figure 20: *Teapots on a ground plane illuminated by a square area light source. The yellow outline marks a penumbra region; the green outline marks a fully illuminated region. (No pixel sampling.)*

Surface points in the penumbra region (marked with a yellow outline and also shown in Figure 2) are partially in shadow and partially lit with smoothly varying intensity. (Surface points completely in shadow have black illumination, and all sample sequences get zero error there.) Figure 21 (top) shows the familiar $O(N^{-0.5})$ and $O(N^{-0.75})$ convergence rates. Surface points in the fully illuminated region, marked with a green outline, receive smooth illumination, only modulated by distance and cosine terms. Figure 21 (bottom) shows that the convergence is roughly $O(N^{-1.5})$ for pmj02 and Owen-scrambled Sobol'.

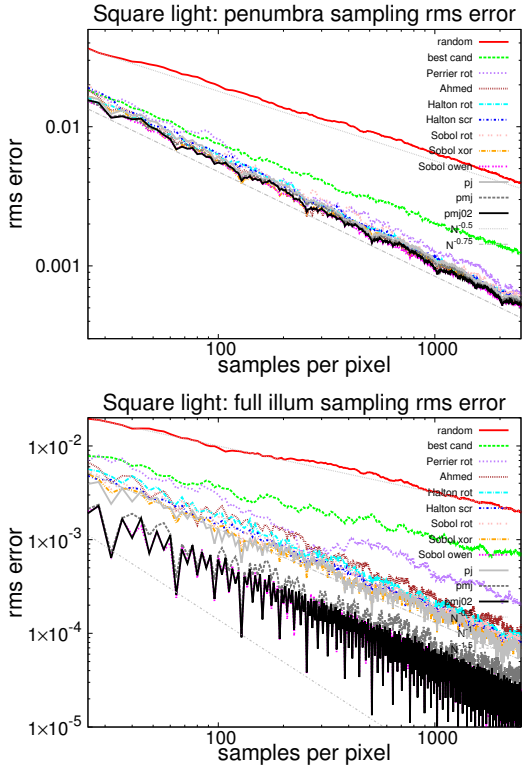

Figure 21: *rms error for 25–2500 samples per pixel of a square light source. Top: penumbra region. Bottom: fully illuminated region.*

We repeat these experiments with a rectangular area light source in the supplemental material, and investigate sampling strategies for disk light sources in a separate technical report [Chr18]. The results are mostly the same as for the square light, but disk lights require a special sampling strategy to obtain low error and fast convergence – roughly $O(N^{-1.4})$ – in smooth, fully illuminated areas.

10. Variations and extensions

There are several useful variations of progressive multi-jittered sample sequences.

10.1. Progressive multi-jittered samples with blue noise properties

Up until this point, we have only shown that our best stochastic sample sequences have the same low error as the best combina-

tion of quasi-random sequence and randomization. In this section we add another desirable property (which does not exist for the standard quasi-random sequences): larger distance between nearest neighbor points, which gives a blue noise spectrum. The only difference from the algorithm for simple pmj is that after we select a 2D cell to place a sample in, we generate a few candidate points (in empty 1D strips) within that 2D cell and pick the one with the largest distance to the nearest previous points. The pseudo-code in the supplemental material shows the details.

We call this version of the algorithm *progressive multi-jittering with blue noise* or *pmjbn*. Figure 22 shows 500 pmj samples without and with the blue noise property. The left top row of Figure 1 shows 4, 16, 64, 256, and 1024 samples from a pmjbn sequence.

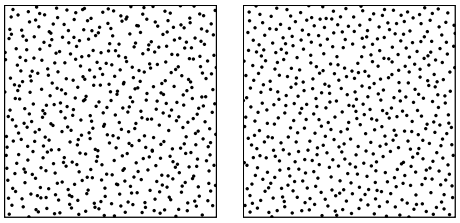

Figure 22: *First 500 samples from a standard pmj sequence and from a pmj sequence with blue noise property (improved nearest-neighbor distances).*

Table 1 shows the average and minimum 2D nearest-neighbor distance between 25 and 500 sample points for various progressive sequences. Distances are toroidal. Bigger nearest-neighbor distances are better as they indicate less clumping. As can be seen by comparing the entries for 500 pmj and pmjbn samples, the addition of best candidates within each sample cell improves the 2D average distance from 0.0287 to 0.0336, and 2D minimum distance from 0.0055 to 0.0105. Pjbn and pmjbn samples have significantly larger average distances than any Halton or Sobol' sequence, indicating a more uniform distribution. Only the best-candidates sequence has larger distances, but that is because its sample points are constructed to maximize this measure. (As mentioned earlier, best candidates are too uneven overall, and converge slowly.) Note that using the best-candidate algorithm for pj and pmj gives much improved nearest-neighbor distances, but doing the same for pmj02 does not improve it much. It seems that the strict stratification of pmj02 samples does not leave much "wiggle room" to increase the nearest-neighbor distances – at least with our current algorithm. See Perrier et al. [PCX*18] for a different trade-off between stratification and blue noise.

Figure 23 shows Fourier spectra of various sample sequences, including the blue-noise versions of our pj, pmj, and pmj02 sequences.

We have repeated our tests in Sections 7 through 9 with pjbn and pmjbn sequences. The results are mixed; for example, for the binary zoneplate the rms and maximum error using pmjbn is a few percent lower than using pmj, but for the smooth zoneplate it is the other way around. Anyway, we speculate that pmjbn might be useful in situations where its spectral properties (blue noise) are desirable for a more visually pleasing pattern.

Table 1: *Average and minimum nearest-neighbor distances between 25 and 500 sample points from various sample sequences. (Computed as average of 10000 sequences, except the Perrier and canonical Halton and Sobol' sequences marked with '*'.)*

	25 samples		500 samples	
sequence	avg dist	min dist	avg dist	min dist
random	0.100	0.028	0.0224	0.0014
best cand	0.169	0.133	0.0380	0.0289
Perrier *	0.122	0.060	0.0271	0.0088
Ahmed	0.143	0.093	0.0340	0.0179
Halton *	0.141	0.115	0.0278	0.0111
Halton scr	0.129	0.068	0.0279	0.0082
Sobol' *	0.119	0.088	0.0283	0.0055
Sobol' xor	0.119	0.066	0.0291	0.0041
Sobol' owen	0.128	0.065	0.0290	0.0067
pj	0.126	0.059	0.0287	0.0051
pjbn	0.156	0.120	0.0354	0.0217
pmj	0.128	0.064	0.0287	0.0055
pmjbn	0.153	0.103	0.0336	0.0105
pmj02	0.128	0.065	0.0290	0.0067
pmj02bn	0.139	0.082	0.0296	0.0077

We have not yet been able to develop a version of pmj02 that gives significantly higher nearest-neighbor distances at high sample counts. Simply choosing the best out of some candidates that fulfill the (0,2) property only improves nearest-neighbor distances by a small amount. We leave this as future work. As it stands, one will have to choose between the improved spectral properties of pmjbn or the better convergence of pmj02 for smooth functions.

10.2. Interleaved multi-class samples

Inspired by multi-class blue noise samples [Wei10], we wondered whether our sample sequences can be divided into multiple classes on the fly. It turns out this is very simple to do.

10.2.1. Two classes

Pj, pmj, and pmj02 samples can be divided on the fly into two classes during sample generation. When we generate 3 new samples based on a previous sample (as described in Sections 4–6), the first new sample (the one in the diagonally opposite subquadrant) is assigned the same class as the previous sample; the other two samples are assigned the other class. This gives the following order of sample classes: *AABBAABBBBAABBAA*... for all pj, pmj, and pmj02 sequences. The left bottom row of Figure 1 shows a pmj02 sequence where the samples are divided into two interleaved classes this way. This classification is trivial to do on the fly, and does not increase the complexity of sample generation.

Figure 24 shows error plots for sampling the disk and Gaussian functions with the original pmj02 sequences and their two sample classes. Note that the samples in the two classes have the same convergence rate as the full pmj02 sequences. They are in fact (1,2) sequences. (This is easy to check: at each octave there are exactly two samples in each elementary interval.)

The two interleaved sample classes are useful for calculating two independent estimates of a result while sampling with a full pmj02

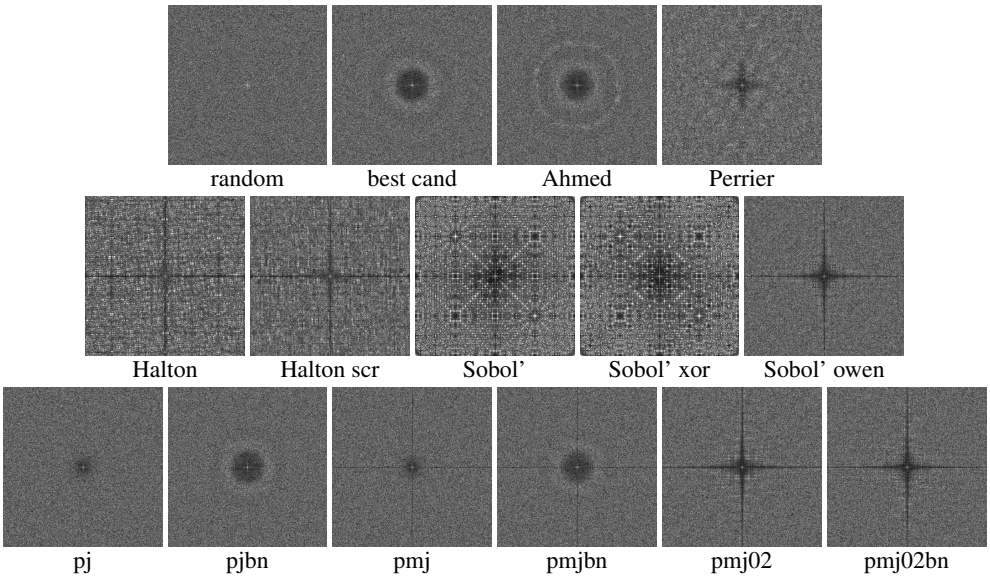

Figure 23: *Fourier spectra of sample sequences.*

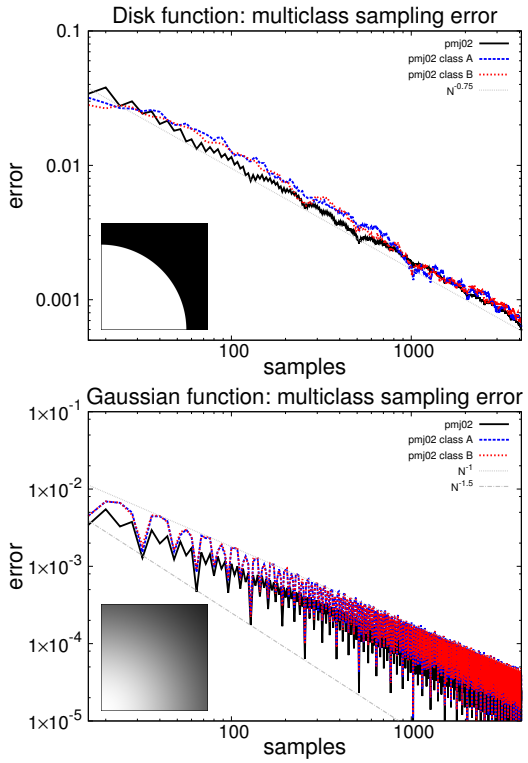

Figure 24: *Error for sampling the disk and Gaussian functions with pmj02 sequences and their two classes.*

sequence. The difference between the two estimates is an indicator of sampling error and can be used to drive adaptive sampling by detecting pixels with high error and focusing the sampling there.

Note that this interleaved generation is very different from the standard multi-classes of Halton, Sobol', and Ahmed ART sequences: if one knows a priori the total number of samples to be taken, the samples can be divided into the first and second half, with each half being well distributed. Our sequences have this property as well, but it is obviously not suitable for adaptive sampling.

It should be mentioned that 2D Halton and Sobol' sequences can also be divided into interleaved classes; this can be done by generating 3D samples and using the third dimension to partition the classes.

10.2.2. Four classes

Alternatively, we can divide the samples into *four* classes on the fly. Denote the classes as A,B,C,D. When we generate 3 new samples based on a previous sample, the class of the first new sample (the one in the diagonally opposite subquadrant) is assigned with the mapping $A,B,C,D \mapsto B,A,D,C$ from the previous sample, the other two samples are assigned class with mapping $A,B,C,D \mapsto C,D,A,B$ and $A,B,C,D \mapsto D,C,B,A$, respectively. These rules give the following order: *ABCDBADCCDABDCBA*... Figure 25 shows a pmj02 sequence where the samples are divided into four classes this way. Each class is a pj sequence.

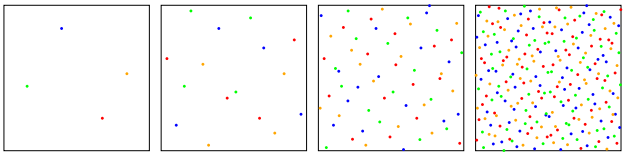

Figure 25: *4, 16, 64, and 256 samples from a pmj02 sequence divided into four classes (denoted by blue, red, green, and orange dots).*

11. Discussion

As demonstrated in Sections 7 through 9, sampling with progressive multi-jittered (0,2) sequences matches the error of the best quasi-random sequences we know of, including extremely fast convergence for smooth pixels and unshadowed area light illumination. The quasi-random sequences have the advantage that any sample can be generated without first having to generate the previous samples in the sequence, and their implementations can be written in just a handful of lines of C code. On the other hand, our pmj sequences open up new avenues for stochastic generation of high-quality samples, for example targeting blue noise spectra. Table 2 summarizes which sequences are jittered, multi-jittered, or (0,2) sequences, and/or have blue noise spectra.

Table 2: *Table summarizing which progressive sample sequences have jitter, multi-jitter, (0,2), and blue-noise properties.*

sequence	jittered	multi-jittered	(0,2)	blue noise
random	-	-	-	-
best cand	-	-	-	+
Perrier	+	+	-	+
Ahmed	+	-	-	+
Halton	+	+	-	-
Sobol'	+	+	+	-
pj	+	-	-	-
pjbn	+	-	-	+
pmj	+	+	-	-
pmjbn	+	+	-	+
pmj02	+	+	+	-
pmj02bn	+	+	+	(-)

The pj, pmj, and pmj02 sample sequences are generated for square domains. In the supplemental material we show that they are also suitable for warped sample domains such as Gaussian pixel filters and rectangular light sources.

Although the pmj02 sequences are theoretically infinite, in the RenderMan renderer, we truncate them at 4096 samples in pre-generated tables (and start over in a different table in the rare case that more samples are required per pixel per dimension). We have a few hundred different tables, and each pixel index and ray depth is hashed to a table number.

It is fairly easy to generalize the pmj sample generation algorithms to three dimensions. For four dimensions or higher, we combine 2D pmj02 sequences by locally shuffling the samples in pairs of dimensions. Typically such shuffling is done just to decorrelate the dimensions [Coo86], but since we use tables we can do better: shuffle such that the samples are stratified in 4D (while maintaining the full (0,2) property in each pair of dimensions). This shuffling is implemented as a simple "greedy" algorithm: for all pairs of 2D points within segments of a (truncated) sequence, swap the points if swapping them reduces the number of 4D strata with more than one sample. This results in a sequence with full 4D stratification (for example $4\times4\times2\times2$ strata for the first 64 samples) and takes only a few minutes for two 2D sequences with 4096 samples.

Figure 26 shows a close-up of a scene where samples are being used for pixel positions, brdf reflection directions for multiple bounces of diffuse reflection, and sampling of an environment light source. The left image was rendered with random samples; the right image was rendered with pmj02 samples and is less noisy.

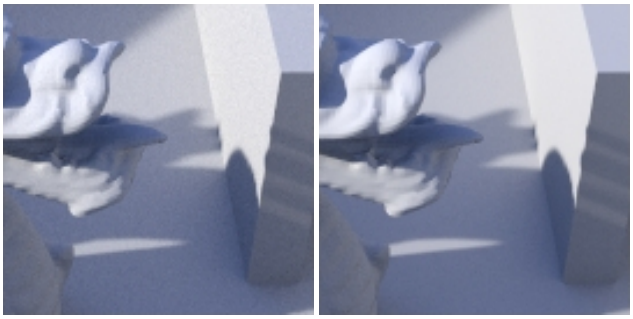

Figure 26: *Diffuse scene illuminated by an environment light, rendered with 1024 samples per pixel. Left: noisy image rendered with random samples. Right: cleaner image rendered with pmj02 samples. Dragon model courtesy of the Stanford 3D Scanning Repository. Scene by Philippe Leprince.*

12. Conclusion and future work

The contributions of this paper are two-fold: a fresh assessment of existing progressive sample sequences, and new stochastic alternatives to randomized quasi-random sample sequences. Our new sample sequences yield sampling errors that are on par with the best known sequence: Owen-scrambled Sobol' (0,2). But with our stochastic generation, we can also target blue noise and other properties.

We pointed out faster convergence in smooth pixels and unshadowed area lights; this can be utilized in adaptive sampling, particularly when combined with multi-class samples for error estimation.

Possible future work consists of further improvement of the pmj and pmj02 samples, and optimization of the sample generation time. Higher quality could possibly be obtained by better merging the stratification of pmj02 with the minimum distance property of blue noise (if possible). Faster sample generation could perhaps be done with only integer arithmetic (although our experiments with this have failed so far); certainly, multiple tables could be generated independently in parallel.

It would also be interesting to test our pmj and pmj02 sequences with the stratum shearing approach suggested by Singh and Jarosz [SJ17].

In addition to the blue-noise and interleaved multi-class samples generated within our general pmj framework, we also envision such variations as targeting other spectral profiles (for example pink or purple noise), running simulated annealing for each new round of sample points (while restricting each sample to stay within its strata), or backtracking placement of samples when the remaining empty strata forces points to be too close.

It is our hope that the pmj framework will inspire future development of progressive sample sequences – hopefully sequences that are even more optimal for sampling in general and computer graphics in particular.

Acknowledgements

Many thanks to our colleagues in Pixar's RenderMan team for all their help in developing and testing pmj sampling, especially the "Noise task force" including Cliff Ramshaw, Jonathan Shade, Philippe Leprince, Marc Bannister, and Max Liani. Very special thanks to Brent Burley at Disney for many sampling discussions and suggestions, and for sharing his efficient code for progressive Owen scrambling. Max Liani independently constructed progressive multi-jittered sequences while working at Animal Logic. Many thanks to Matt Pharr and Alexander Keller at Nvidia, Christophe Hery and Ryusuke Villemin at Pixar's lighting tools group, and Emmanuel Turquin and André Mazzone at ILM for nuanced discussions about sample sequences and sampling in general. Also many thanks to Victor Ostromoukhov for explaining his tiling-based sample sets and for discussions about discrepancy and tests, and feed-back on drafts of this paper. And finally thanks to Per Skafte Hansen for early discussions about math and random numbers in general.

References

[ANHD17] AHMED A., NIESE T., HUANG H., DEUSSEN O.: An adaptive point sampler on a regular lattice. *ACM Transactions on Graphics (Proc. SIGGRAPH) 36*, 4 (2017). 2, 3

[Chr18] CHRISTENSEN P.: *Progressive sampling strategies for disk light sources*. Tech. Rep. 18-02, Pixar Animation Studios, 2018. 9

[Coo86] COOK R.: Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1 (1986), 51–72. 2, 12

[CP76] CRANLEY R., PATTERSON T.: Randomization of number theoretic methods for multiple integration. *SIAM Journal on Numerical Analysis 13*, 6 (1976), 904–914. 3

[CSW94] CHIU K., SHIRLEY P., WANG C.: Multi-jittered sampling. In *Graphics Gems IV*. Academic Press, 1994, pp. 370–374. 2, 5

[DEM96] DOBKIN D., EPPSTEIN D., MITCHELL D.: Computing the discrepancy with applications to supersampling patterns. *ACM Transactions on Graphics 15*, 4 (1996), 354–376. 2

[DW85] DIPPÉ M., WOLD E.: Antialiasing through stochastic sampling. *Computer Graphics (Proc. SIGGRAPH) 19*, 3 (1985), 69–78. 5

[Grü12] GRÜNSCHLOSS L.: QMC sampling source code, 2012. http://gruenschloss.org. 3

[Hal64] HALTON J.: Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM 7*, 12 (1964), 701–702. 2, 3

[Ham60] HAMMERSLEY J.: Monte Carlo methods for solving multivariable problems. *Annals of the New York Academy of Sciences 86* (1960), 844–874. 2

[Kaj86] KAJIYA J.: The rendering equation. *Computer Graphics (Proc. SIGGRAPH) 20*, 4 (1986), 143–150. 5

[Kel12] KELLER A.: Quasi-Monte Carlo image synthesis in a nutshell. In *Proc. Monte Carlo and Quasi-Monte Carlo Methods* (2012), pp. 213–249. 3

[Ken13] KENSLER A.: *Correlated multi-jittered sampling*. Tech. Rep. 13-01, Pixar Animation Studios, 2013. 2, 5

[KK02] KOLLIG T., KELLER A.: Efficient multidimensional sampling. *Computer Graphics Forum (Proc. Eurographics) 21*, 3 (2002), 557–563. 3

[Knu98] KNUTH D.: *The Art of Computer Programming*, 3rd ed., vol. 2. Addison Wesley, 1998. 3

[LK11] LAINE S., KARRAS T.: Stratified sampling for stochastic transparency. *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering) 30*, 4 (2011). 3

[LP01] LARCHER G., PILLICHSHAMMER F.: Walsh series analysis of the L_2-discrepancy of symmetrisized point sets. *Monatshefte für Mathematik 132* (April 2001), 1–18. 2

[MF92] MCCOOL M., FIUME E.: Hierarchical Poisson disk sampling distributions. In *Proc. Graphics Interface* (1992), pp. 94–105. 3

[Mit91] MITCHELL D.: Spectrally optimal sampling for distribution ray tracing. *Computer Graphics (Proc. SIGGRAPH) 25*, 4 (1991), 157–164. 2, 3

[Mit92] MITCHELL D.: Ray tracing and irregularities in distribution. *Proc. Eurographics Workshop on Rendering* (1992), 61–69. 2

[Mit96] MITCHELL D.: Consequences of stratified sampling in graphics. *Computer Graphics (Proc. SIGGRAPH) 30*, 4 (1996), 277–280. 2

[MN98] MATSUMOTO M., NISHIMURA T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation 8*, 1 (1998), 3–30. 3

[ODJ04] OSTROMOUKHOV V., DONOHUE C., JODOIN P.-M.: Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics (Proc. SIGGRAPH) 23*, 3 (2004), 488–495. 2

[Owe97] OWEN A.: Monte Carlo variance of scrambled net quadrature. *SIAM Journal on Numerical Analysis 34*, 5 (1997), 1884–1910. 3, 4

[Owe03] OWEN A.: Quasi-Monte Carlo sampling. In *SIGGRAPH Monte Carlo Ray Tracing Course Notes*. ACM, 2003. 3, 4

[PCX*18] PERRIER H., COEURJOLLY D., XIE F., PHARR M., HANRAHAN P., OSTROMOUKHOV V.: Sequences with low-discrepancy blue-noise 2-D projections. *Computer Graphics Forum (Proc. Eurographics) 37*, 2 (2018), 339–353. 2, 3, 10

[PTVF92] PRESS W., TEUKOLSKY S., VETTERLING W., FLANNERY B.: *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. Cambridge University Press, 1992. 3

[RAMN12] RAMAMOORTHI R., ANDERSON J., MEYER M., NOWROUZEZAHRAI D.: A theory of Monte Carlo visibility sampling. *ACM Transactions on Graphics 31*, 5 (2012). 2

[RRSG15] REINERT B., RITSCHEL T., SEIDEL H.-P., GEORGIEV I.: Projective blue-noise sampling. *Computer Graphics Forum 35*, 1 (2015), 285–295. 3

[Shi91] SHIRLEY P.: Discrepancy as a quality measure for sample distributions. *Proc. Eurographics* (1991), 183–193. 2, 3

[SJ17] SINGH G., JAROSZ W.: Convergence analysis for anisotropic Monte Carlo sampling. *ACM Transactions on Graphics (Proc. SIGGRAPH) 36*, 4 (2017). 12

[SK12] SUBR K., KAUTZ J.: Fourier analysis of stochastic sampling strategies for assessing bias and variance in integration. *ACM Transactions on Graphics (Proc. SIGGRAPH) 31*, 4 (2012). 2

[SKD12] SCHRETTER C., KOBBELT L., DEHAYE P.-O.: Golden ratio sequences for low-discrepancy sampling. *Journal of Graphics Tools 16*, 2 (2012), 95–104. 2

[Sob67] SOBOL' I.: On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics 7*, 4 (1967), 86–112. 2, 3

[SWZ96] SHIRLEY P., WANG C., ZIMMERMAN A.: Monte Carlo methods for direct lighting calculations. *ACM Transactions on Graphics 15*, 1 (1996), 1–36. 4, 6

[Wei10] WEI L.-Y.: Multi-class blue noise sampling. *ACM Transactions on Graphics (Proc. SIGGRAPH) 29*, 4 (2010). 10

[WPC*14] WACHTEL F., PILLEBOUE A., COEURJOLLY D., BREEDEN K., SINGH G., CATHELIN G., DE GOES F., DESBRUN M., OSTROMOUKHOV V.: Fast tile-based adaptive sampling with user-specified Fourier spectra. *ACM Transactions on Graphics (Proc. SIGGRAPH) 33*, 3 (2014). 2

[Zar68] ZAREMBA S.: The mathematical basis of Monte Carlo and quasi-Monte Carlo methods. *SIAM Review 10*, 3 (1968), 303–314. 2