Efficient Generation of Points that Satisfy Two-Dimensional Elementary Intervals

Matt Pharr
NVIDIA Research

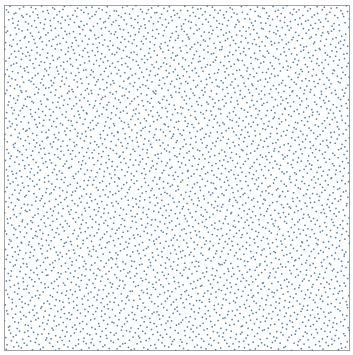

Figure 1. 4,096 pmj02bn samples generated using our technique. These points are progressive, satisfy all applicable power-of-two elementary intervals, and have optimized spectral properties. They were generated in 5.19 ms on a single core of a 4 GHz CPU.

Abstract

Precomputing high-quality sample points has been shown to be a useful technique for Monte Carlo integration in rendering; doing so allows optimizing properties of the points without the performance constraints of generating samples during rendering. A particularly useful property to incorporate is stratification across elementary intervals, which has been shown to reduce error in Monte Carlo integration. This is a key property of the recently-introduced progressive multi-jittered, pmj02 and pmj02bn points [Christensen et al. 2018].

For generating such sets of sample points, it is important to be able to efficiently choose new samples that are not in elementary intervals occupied by existing samples. Random search, while easy to implement, quickly becomes infeasible after a few thousand points. We describe an algorithm that efficiently generates 2D sample points that are stratified with respect to sets of elementary intervals. If a total of n sample points are being generated, then for each sample, our algorithm uses $O(\sqrt{n})$ time to build a data structure that represents the regions where a next sample may be placed. Given this data structure, valid samples can be generated in $O(1)$ time. We demonstrate the utility of our method by generating much larger sets of pmj02bn points than were feasible previously.

 ISSN 2331-7418

1. Introduction

Sampling is at the heart of modern rendering; care in choosing good $[0,1)^n$ sample points for use in (quasi)-Monte Carlo integration has been shown to have significant benefits for image quality. Good samples effectively provide a performance improvement by making it possible to render images with fewer samples (and thus, fewer rays traced) than if inferior distributions of sample points are used. See, for example, Keller's survey [2013] or Chapter 7 of Pharr et al. [2016] for background and more information.

In a Monte Carlo path tracer, each ray path consumes tens or even hundreds of dimensions of such a sample point, and in high performance ray tracers, millions or even billions of rays may be traced in a second. Thus, it is important to be able to generate samples efficiently; too much time spent generating them eventually would be better used in just tracing more rays.

Two main approaches have been taken to the sample generation problem: direct generation of samples with known good properties at rendering-time, or optimization of explicitly enumerated samples before rendering in the pursuit of specific criteria. Examples of the first approach include both stratified sampling and Kollig and Keller's application of randomized instances of the first two dimensions of the Sobol' sequence to rendering [Kollig and Keller 2002]. The second is exemplified by Mitchell's work on multidimensional sampling [Mitchell 1991].[1]

While the first approach makes it possible to generate unlimited numbers of sample points at rendering time, it is constrained because not much computation is possible for each one. Further, to fulfill particular criteria requires finding an appropriate mathematical construction and proving that it fulfills the criteria. The second approach generally trades generating a fixed number of samples for the opportunity to apply much more computation to generate them, thus being able to satisfy a wider range of criteria.

1.1. Stratification and Elementary Intervals

It has long been understood that stratified sampling patterns are effective for low-dimensional integration problems in rendering (e.g., direct illumination) [Dippé and Wold 1985; Cook 1986]. With stratified sampling in 2D, the $[0,1)^2$ integration domain is divided into $n_x \times n_y$ non-overlapping *strata* defined by

$$\left[\frac{x}{n_x}, \frac{x+1}{n_x}\right) \times \left[\frac{y}{n_y}, \frac{y+1}{n_y}\right),$$

where x and y are integers with $0 \le x < n_x$ and $0 \le y < n_y$. In the following, we will sometimes label the (x,y)-stratum in an $n_x \times n_y$ stratification as $(x,y)_{n_x \times n_y}$.

[1] There are, of course, also hybrid approaches that mix both of these approaches, for example Grünschloß et al. [2008] and Perrier et al. [2018].

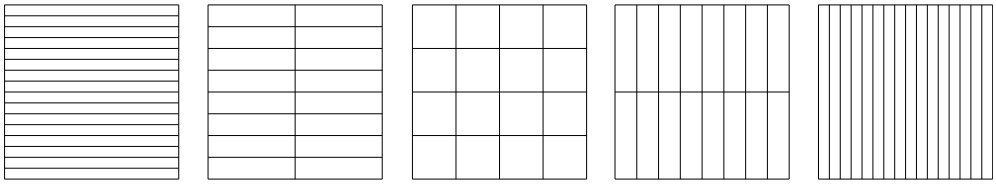

Figure 2. The five 2D power-of-two sets of elementary intervals that apply given $n = 16$ sample points. From left to right, there are stratifications of 1×16, 2×8, 4×4, 8×2, and 16×1.

If a single sample point placed is within each stratum, then the sample points cannot cluster together as much as they could with independent samples, which usually reduces integration error.

A more general form of stratification, which can be used to define multiple stratifications, can be defined using elementary intervals.[2] A single 2D base-2 elementary interval is a stratum with power-of-two side lengths

$$\left[\frac{x}{2^{l_x}}, \frac{x+1}{2^{l_x}}\right) \times \left[\frac{y}{2^{l_y}}, \frac{y+1}{2^{l_y}}\right),$$

where l_x and l_y are integers that are greater than or equal to zero, and x and y are integers where $0 \leq x < 2^{l_x}$ and $0 \leq y < 2^{l_y}$. For fixed l_x and l_y, the set of intervals given by all valid x- and y-values are a $2^{l_x} \times 2^{l_y}$ stratification over $[0, 1)^2$.

Given a power-of-2 number of sample points n, we will define the set of all applicable elementary intervals as those that satisfy the additional constraint $l_x + l_y = \log_2 n$. For an example, see Figure 2, which shows the five elementary intervals that apply given $n = 16$ sample points. For the remainder of this paper, all of the sets of elementary intervals will be of this form.

We will say that a set of sample points where there is exactly one sample point in every elementary interval in the entire set is stratified with respect to those elementary intervals. This form of stratification has been shown to be important for point sets used in rendering because it helps preserve the distribution when samples are warped during importance sampling.

It can be shown that for any power-of-two number of samples, it is possible to generate a point set that is stratified with respect to all applicable elementary intervals. Furthermore, such sample sets can be constructed in a greedy manner—we can always be sure there is a free spot left, even for the last sample.

[2] See Dick and Pillichshammer's book [2010] for more information on elementary intervals and low-discrepancy sampling.

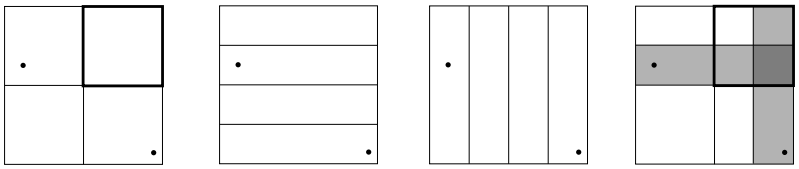

Figure 3. When placing a new sample point in the $(1,1)_{2\times2}$ stratum (left, shown with a bold outline), we would like to make sure that the point doesn't violate the constraint of there being only a single point in each elementary interval. If we consider the two existing sample points here with respect to the 1×4 and 4×1 elementary intervals that they are in (middle two illustrations), we can see that only the upper left quarter of the $(1,1)_{2\times2}$ stratum may be chosen for the new point (unshaded region inside bold outline, right).

2. Generating Samples Subject to Elementary Intervals

It is easy to represent the occupancy of elementary intervals: if one is generating n sample points, there are a total of $n(1 + \log_2 n)$ elementary intervals and the occupancy of each can be represented with a Boolean value. Because every point in $[0,1)^2$ is inside $O(\log n)$ elementary intervals, both querying the admissibility of a candidate point and updating the representation of occupied intervals for a new point require $O(\log n)$ operations.

It is less obvious how to efficiently generate new samples that are valid with respect to the already-occupied elementary intervals.

Figure 3 illustrates the elementary interval constraint in the context of choosing the location for a new sample in a point set. There, we have two sample points already placed and are trying to place a third in the square stratum shown with a bold outline. If we consider the two occupied elementary intervals from the 1×4 and 4×1 stratifications that overlap the square stratum, then we can see that only one-quarter of it can be used. Placing a point in the rest (shaded region) would leave two sample points in one elementary interval. As the number of sample points increases (and thus, the number of elementary intervals that must be satisfied increases), finding admissible sample points becomes increasingly challenging.

Christensen et al. [2018] used random search to find valid points, first choosing a square interval and then generating random points inside it until they find one that satisfies all of the other overlapping elementary intervals. Figure 4 illustrates the difficulty with this approach. It shows one of the square elementary intervals encountered when generating 16,384 samples; invalid regions are shaded gray. It's evident that random sampling over the entire square interval is an inefficient way to find valid samples.

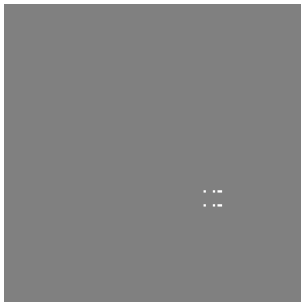

Figure 4. Visualization of a randomly-chosen stratum from the 128×128 stratification of $[0, 1)^2$ that was considered when generating a set of 16,384 points that satisfy the elementary interval constraint. The invalid area is shaded in gray and the valid region is white.

2.1. Algorithm

Our approach builds on a single key observation: given an elementary interval $e = (x, y)_{n_x \times n_y}$ in which we'd like to to generate a new sample, all of the elementary intervals from other stratifications that overlap e cover it entirely in one of the two dimensions—the one where the other elementary interval's length is longer than e's. This observation is illustrated in Figure 5. This property follows from the fact that the power-of-two elementary intervals all have inverse power-of-two side lengths, start at offsets that are integer multiples of their side lengths, and all have the same area.

Therefore, in order to represent the 2D subset of e that is invalid due to overlapping occupied elementary intervals, it suffices to represent the 1D x- and y-ranges within e that are invalid. Overlapping strata that have a narrower x-extent than e contribute to the set of invalid x-ranges, and those that are narrower in y contribute to the invalid y-ranges.

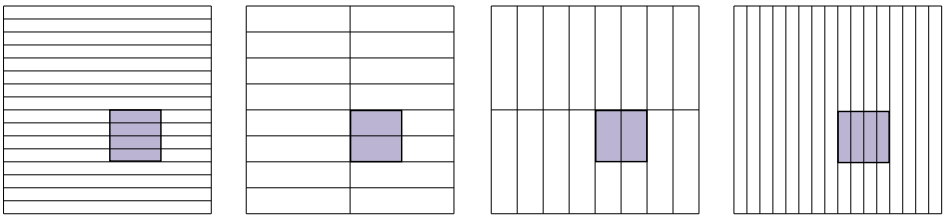

Figure 5. All of the elementary intervals from different stratifications that overlap an elementary interval span its entire extent in one of the two dimensions. Here, we consider the $(2, 1)_{4\times4}$ interval (shaded square) and its overlap with all of the other applicable stratifications. For example, in the third figure, we can see that two of the 8×2 intervals overlap—$(4, 0)_{8\times2}$ and $(5, 0)_{8\times2}$. Both cover all of the square interval's extent in y and then extend below it. Thanks to this property, we only have to maintain two one-dimensional representations of the regions of an interval that are invalid for new samples, which in turn is the foundation of our efficient sample generation algorithm.

Note that it is easy to enumerate the strata from other stratifications that overlap e. In the x-dimension, the elementary intervals that cover part of e's x-range and all of its y-range have stratifications $2n_x \times n_y/2$, $4n_x \times n_y/4$, ..., $n \times 1$. The coordinates of the overlapping intervals are also easily enumerated. Again considering x, the intervals $(2x, \lfloor y/2 \rfloor)_{2n_x \times n_y/2}$ and $(2x+1, \lfloor y/2 \rfloor)_{2n_x \times n_y/2}$ each overlap half of the x-extent of e's extent in x and overlap all of its y-extent. Similarly for $(4x+i, \lfloor y/4 \rfloor)_{4n_x \times n_y/4}$ for $i = 0, 1, 2, 3$, and so forth.

Our algorithm uses binary trees to encode a hierarchical representation of invalid ranges. If a total of $n = n_x n_y$ samples are being generated, then given an elementary interval $e = (x, y)_{n_x \times n_y}$ in which to generate a sample, we construct two binary trees, one in x and one in y, where there are n/n_x and n/n_y leaves, respectively, that encode

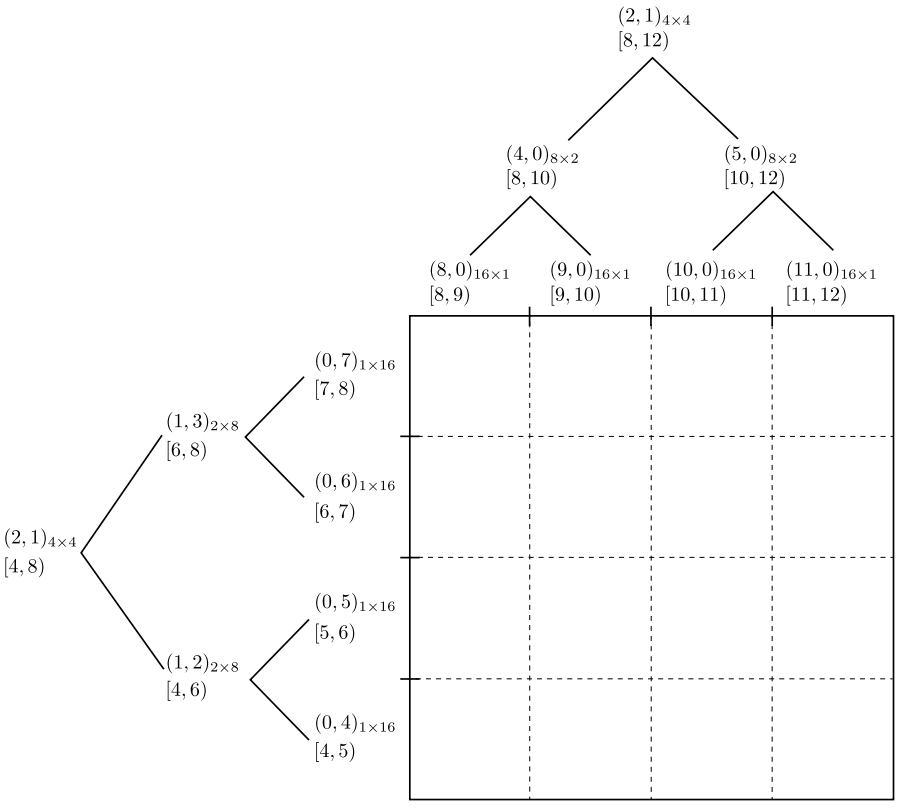

Figure 6. The two binary trees used to represent invalid regions for samples in the elementary interval $e = (2, 1)_{4\times4}$. (This is the same setting as the case illustrated in Figure 5.) Each non-root node of each tree corresponds to a range of the extent of e in its dimension and stores a Boolean value that's true if that range is completely covered by already-occupied elementary intervals. Each non-root node corresponds to the extent of overlap of a single elementary interval, shown here. Under each interval's coordinates is the range of the domain of either the 16×1 (x's tree) or 1×16 (y's tree) stratification that it represents.

whether the corresponding range in their dimension is invalid. Interior nodes store a Boolean value that is true only if the entire range that its children cover is invalid.

If we consider the overlapping elementary intervals, we can see that each node of each tree corresponds to exactly one of them. This idea is shown in Figure 6. Note there that in x, for example, the stratum $(2x, \lfloor y/2 \rfloor)_{2n_x \times n_y/2}$ corresponds to the x-range represented by the first node one level down from the root. For each occupied elementary interval, we can mark the corresponding node as occupied and don't need to consider any nodes (and corresponding elementary intervals) beneath it.

The algorithm to construct the tree for the x-dimension is shown in Figure 7. In our implementation, we use bit vectors to represent the trees, with one bit representing each node's occupancy. After the trees have been initialized, a second traversal gives arrays of 1D coordinates of acceptable sub-strata within the selected stratum e. That algorithm is given in Figure 8.

Given these arrays, generating a valid sample is a matter of randomly selecting elements x' and y' from them. Because the sub-strata indices are defined with respect to a full $n \times n$ stratification of the unit square, any point within the square interval

$$\left[\frac{x'}{n}, \frac{x'+1}{n}\right) \times \left[\frac{y'}{n}, \frac{y'+1}{n}\right)$$

is both a valid sample and inside e.

```
function InitializeXTree(node, x, y, n_x, n_y)
    if the (x, y)_{n_x × n_y} elementary interval is filled then
        node.occupied ← true                          ▷ Tree traversal stops
    else
        node.occupied ← false
        if ¬node.leaf then
            InitializeXTree(node.left, 2x, ⌊y/2⌋, 2n_x, n_y/2)
            InitializeXTree(node.right, 2x + 1, ⌊y/2⌋, 2n_x, n_y/2)
            node.occupied ← node.left.occupied ∧ node.right.occupied
```

Figure 7. Our algorithm initializes two binary trees corresponding to the binary subdivision of the x- and y-extent of an elementary interval. The algorithm for the x-case is shown here. Traversal starts with the elementary interval selected to have a sample placed in it, $e = (x, y)_{n_x \times n_y}$, at the root. Each node stores a Boolean value that indicates whether the corresponding range of the extent is fully covered by filled elementary intervals and thus inadmissible for a new sample point. Note that the coordinates and dimensions of the corresponding elementary interval at each node are easily computed during traversal. The algorithm for y is analogous, just with the elementary interval indexing modified appropriately.

```
function GetValidXOffsets(node, x, y, n_x, n_y, out offsets[])
    if ¬node.occupied then
        if node.isLeaf then
            offsets.append(x)
        else
            GetValidXOffsets(node.left, 2x, ⌊y/2⌋, 2n_x, n_y/2, offsets)
            GetValidXOffsets(node.right, 2x + 1, ⌊y/2⌋, 2n_x, n_y/2, offsets)
```

Figure 8. After the invalid interval trees have been initialized, they can be traversed to extract valid stratum indices with respect to the $n \times 1$ stratification in x (and $1 \times n$ in y.) The algorithm for the x-dimension is shown here.

2.2. Summary

Our algorithm can be used to generate a set of well-distributed points via the following steps:

1. Select an empty elementary interval for the next sample point. The interval may be chosen so that the point set is progressive—that prefixes of it are well-distributed, as was done by Christensen et al. [2018], or, for example, by considering square elementary intervals in scanline order.

2. Construct two binary trees that encode the regions of the selected elementary interval that are invalid due to overlapping occupied elementary intervals.

3. Generate a new sample inside the subset of the selected elementary interval that is admissible, for example by choosing a random point in the valid area.

4. Update data structures that represent occupied elementary intervals, given the new point.

2.3. Running-time Analysis

Given an empty elementary interval in which we'd like to generate a sample, our algorithm has three phases: initializing the binary trees, extracting the valid sub-strata in each dimension, and generating samples.

We have $O(n/n_x) = O(n_y)$ and $O(n/n_y) = O(n_x)$ nodes to visit in each tree and thus $O(n_y + n_x)$ elementary intervals to consider. The total number of tree nodes is minimized if we choose a square stratum as the initial one, in which case $n_x = n_y = \sqrt{n}$ and there are $O(\sqrt{n})$ nodes in both trees. Note that not all nodes will always be visited—once we mark an interior node as occupied, we don't need to consider the elementary intervals that correspond to its children. We don't model that factor in the analysis here, though as more of the elementary intervals are filled, this characteristic significantly reduces the number of nodes visited. (And thus, our algorithm becomes more efficient in that respect as more points are added to the sample set.)

Thus, with two tree traversals and a square stratum as a starting point, we're left with $O(\sqrt{n})$ computation to create the trees and find valid sub-strata indices, since each node is visited no more than once for each of those phases.

Then, given the x- and y-arrays of sub-strata indices from the algorithm in Figure 8, it requires $O(1)$ time to pick a random element of each one and generate a sample.

Finally, because $1+\log_2 n$ elementary intervals overlap each point, $O(\log n)$ time is needed to update the data structure representing occupied elementary intervals after we place a sample, but that's dominated by the $O(\sqrt{n})$ time for the tree traversals.

3. Results

In order to evaluate our algorithm, we applied it to the generation of pmj02 and pmj02bn points [Christensen et al. 2018]. We used the same approach that Christensen et al. did to select elementary intervals in which to generate samples; see their paper for details. Performance was measured with a single-threaded implementation on a 4 GHz Intel CPU.

The pmj02 points are progressive and stratified across elementary intervals. There are no further constraints on the points: any valid one will do. Table 1 compares the performance of our algorithm to using random search to generate these points. Our algorithm is always faster, and its advantage increases as more points are generated. Our approach is over $750\times$ faster for a 1024^2 point set, generating the points in just over three seconds, while using random search requires over 40 minutes.

The pmj02bn points are further optimized to have blue noise properties: multiple candidate points are generated, and the one farthest away from the existing points is kept. Our algorithm is particularly beneficial for that case, thanks to its $O(1)$ execution time to generate valid points after the binary trees have been created and the valid stratum indices extracted. Table 2 shows the result when 10 candidates are consid-

# samples	Random	Our algorithm	Ratio
256	0.415 ms	0.202 ms	2.05 : 1
1,024	5.13 ms	0.632 ms	8.11 : 1
4,096	63.5 ms	3.19 ms	19.9 : 1
16,384	857 ms	19.0 ms	45.1 : 1
65,536	12,199 ms	103 ms	118 : 1
262,144	175,703 ms	543 ms	323 : 1
1,048,576	2,509,131 ms	3,343 ms	751 : 1

Table 1. Comparison of execution time to generate various numbers of pmj02 sample points using random search versus using our algorithm to find samples that satisfy the elementary intervals.

# samples	Random	Our algorithm	Ratio
256	3.38 ms	0.280 ms	12.1 : 1
1,024	42.7 ms	1.08 ms	39.5 : 1
4,096	548.3 ms	5.19 ms	106 : 1
16,384	7,398 ms	26.2 ms	282 : 1
65,536	103,540 ms	130.1 ms	796 : 1
262,144	1,452,286 ms	687.8 ms	2,111 : 1
1,048,576	21,287,063 ms	3,695 ms	5,761 : 1

Table 2. Comparison of execution time to generate various numbers of pmj02bn sample points using random search and our algorithm to find valid candidate samples. For each sample generated, ten candidates were considered in order to improve the blue noise characteristics of the points.

ered. For very low sample counts (e.g., 256 sample points), our approach is over $12\times$ faster than random search. For higher sample counts, the relative performance benefits of our algorithm grow quickly: it is over $5000\times$ faster when generating 1024^2 sample points.[3] Note that the total runtime with random search increases by roughly a factor of 12 with each quadrupling of the number of samples, which suggests an $O(n^2)$ complexity.

To measure the impact of our algorithm's ability to generate candidates in $O(1)$ time, we also measured the performance of generating pmj02bn samples using one thousand candidates. Table 3 shows the results. We found that our algorithm was

# samples	10 candidates	1000 candidates	Ratio
256	0.280 ms	9.43 ms	1 : 33.7
1,024	1.08 ms	42.0 ms	1 : 38.8
4,096	5.19 ms	171.2 ms	1 : 33.0
16,384	26.2 ms	706.8 ms	1 : 27.0
65,536	130.1 ms	2,961 ms	1 : 22.8
262,144	687.8 ms	11,967 ms	1 : 17.4
1,048,576	3,695 ms	49,337 ms	1 : 13.4

Table 3. Comparison of execution time to generate various of pmj02bn sample points using ten candidate samples and one thousand candidate samples. Thanks to our algorithm's $O(1)$ runtime to generate a candidate sample after the invalid interval tree has been generated and the valid sub-stratum offsets extracted from it, we observe much less than the $100\times$ increase in processing time that random search exhibits.

[3] In the full pmj02bn point-generation algorithm, the time to choose the stratum for the next progressive point is just a $O(\log n)$ walk down a quadtree for each new point. Then, the time to compute the distances to the nearest neighbors is constant time if the points are stored in a $\sqrt{n} \times \sqrt{n}$ grid, since only the nearest eight neighbors need be considered. Thus, both asymptotic running time and the vast majority of measured runtime is determined by the time for candidate point generation.

# samples	Avg. # candidates until valid sample
4	3.8
16	12.6
64	45.8
256	128.2
1,024	430.2
4,096	1,577
16,384	5,416
65,536	19,300
262,144	69,550
1,048,576	253,560

Table 4. Average number of candidates considered before finding a single valid sample when random search is used, for various total numbers of samples generated.

between $13\times$ and $39\times$ slower to generate one hundred times more candidate samples, with the relative slow-down decreasing as more samples were generated. Note that random search would be $100\times$ slower in this case, due to requiring one hundred times more random searches.

Table 4 indicates why random search becomes intractable; it shows the average number of samples considered before a valid sample was found when using random search. As the total number of sample points becomes large, we can see that random search quickly becomes very inefficient.

4. Conclusion

We have introduced an efficient new algorithm for incrementally generating sample points that are stratified across the elementary intervals and have shown its use for generating pmj02 and pmj02bn point sets. Performance is over $5000\times$ faster than random search for million-point pmj02bn sets.

With our algorithm, it is now reasonable for a renderer to generate large tables of pmj02bn points at start-up rather than precomputing them, and it is possible to generate much larger sets of pmj02bn points than before, which allows using optimized point sets that span large blocks of pixels, for example. For sample counts up to a few thousand (as would be the case when generating independent sample sets at each pixel), our algorithm is efficient enough to allow on-demand generation of samples.

The extension to higher-dimensional elementary intervals is not just a matter of using more binary trees: the challenge is that one elementary interval that overlaps another will only necessarily cover all of the other's extent in a single dimension. As such, in three dimensions for example, three quadtrees would be required to maintain the invalid region information.

Our algorithm is not amenable to high-performance GPU implementation, as it requires a fair amount of per-thread state, including data structures to represent elementary interval occupancy. If "blue noise" points are to be generated, then all of the samples that have previously been generated must be stored as well. The fact that each sample must be placed before the next can be considered also inhibits parallelism. We plan on investigating these shortcomings in future work.

Approaches based on optimizing sample patterns to achieve specific goals in terms of distribution and quality have repeatedly been shown to be effective for rendering; all modern examples have included satisfying elementary intervals in their criteria. We hope that by having demonstrated an efficient approach for generating points that satisfy the elementary interval constraints, this work will facilitate more progress in the area of sample pattern optimization.

Acknowledgments

Thanks to Aaron Lefohn and NVIDIA Research for supporting this work.

References

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, doi:10.1111/cgf.13472. 56, 59, 63, 64

COOK, R. L. 1986. Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5*, 1 (Jan.), 51–72. URL: http://doi.acm.org/10.1145/7529.8927, doi:10.1145/7529.8927. 57

DICK, J., AND PILLICHSHAMMER, F. 2010. *Digital Nets and Sequences: Discrepancy Theory and Quasi-Monte Carlo Integration*. Cambridge University Press, New York, NY. 58

DIPPÉ, M. A. Z., AND WOLD, E. H. 1985. Antialiasing through stochastic sampling. *SIGGRAPH Computer Graphics Proceedings 19*, 3 (July), 69–78. URL: http://doi.acm.org/10.1145/325165.325182, doi:10.1145/325165.325182. 57

GRÜNSCHLOSS, L., HANIKA, J., SCHWEDE, R., AND KELLER, A. 2008. (t, m, s)-nets and maximized minimum distance. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*, Springer, Berlin - Heidelberg, Germany, A. Keller, S. Heinrich, and H. Niederreiter, Eds., 397–412. 57

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. 57

KOLLIG, T., AND KELLER, A. 2002. Efficient multidimensional sampling. *Computer Graphics Forum 21*, 3, 557–563. doi:10.1111/1467-8659.00706. 57

MITCHELL, D. P. 1991. Spectrally optimal sampling for distribution ray tracing. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1991, Providence, RI, April 27-30, 1991*, ACM, New York, NY, 157–164. URL: https://doi.org/10.1145/122718.122736, doi:10.1145/122718.122736. 57

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. 57

PHARR, M., JAKOB, W., AND HUMPHREYS, G. 2016. *Physically Based Rendering: From Theory To Implementation*. Elsevier, Cambridge, MA. URL: http://www.pbr-book.org/3ed-2018. 57

Author Contact Information

Matt Pharr
NVIDIA, Inc.
2788 San Tomas Expressway
Santa Clara, CA 95051
matt@pharr.org
http://pharr.org/matt

Matt Pharr, Generation of Points that Satisfy Two-Dimensional Elementary Intervals, *Journal of Computer Graphics Techniques (JCGT)*, vol. 8, no. 1, 56–68, 2019
http://jcgt.org/published/0008/01/04/

Received:	2018-12-17		
Recommended:	2019-02-01	Corresponding Editor:	Marc Stamminger
Published:	2019-02-27	Editor-in-Chief:	Marc Olano