An Adaptive Point Sampler on a Regular Lattice

ABDALLA G. M. AHMED, University of Konstanz
TILL NIESE, University of Konstanz
HUI HUANG, Shenzhen University
OLIVER DEUSSEN, University of Konstanz and SIAT Shenzhen

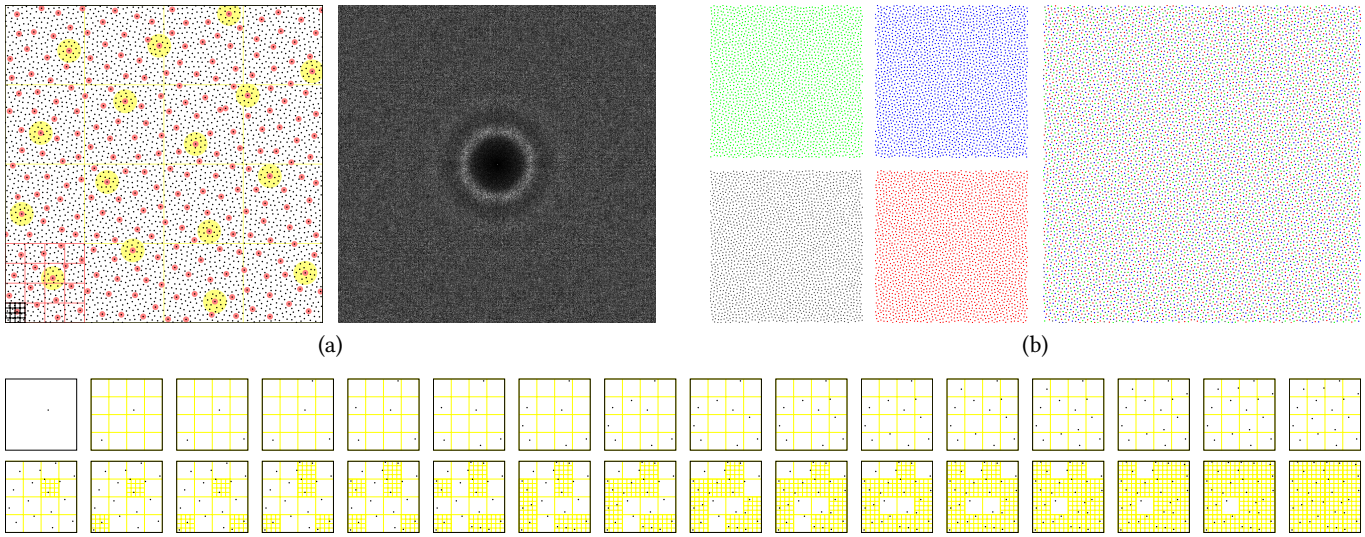

Fig. 1. (a) The proposed ART (**A**daptive **R**egular **T**iles) sampler uses a self-similar regular tile set with 1 sample per tile to supply sequences of blue-noise samples. (b) Tiled multi-class sets can be used to partition a tiled blue noise set into separate blue-noise sets. The two bottom lines show the filling order of our recursive tile in (a). First, sample points are filled in that are shared by one of the respective child tiles. The parent tile then visits the remaining children (in an optimized order) and instructs them to add their samples. For each subsequent 16 (number of children) samples, control is passed recursively to the children – in the same order – to add more samples.

We present a framework to distribute point samples with controlled spectral properties using a regular lattice of tiles with a single sample per tile. We employ a word-based identification scheme to identify individual tiles in the lattice. Our scheme is recursive, permitting tiles to be subdivided into smaller tiles that use the same set of IDs. The corresponding framework offers a very simple setup for optimization towards different spectral properties. Small lookup tables are sufficient to store all the information needed to produce different point sets. For blue noise with varying densities, we employ the bit-reversal principle to recursively traverse sub-tiles. Our framework is also capable of delivering multi-class blue noise samples. It is well-suited for different sampling scenarios in rendering, including area-light sampling (uniform and adaptive), and importance sampling. Other applications include stippling and distributing objects.

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

Additional Key Words and Phrases: Blue noise, tiling, sampling, Monte Carlo, quasi-Monte Carlo, self-similarity, multi-class blue noise, Thue-Morse word

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

1 INTRODUCTION

Point sets are ubiquitous in computer graphics. Of special interest are point sets with a Poisson-disk property, that is, a constrained minimal spacing between neighboring points [Cook 1986; Dippé and Wold 1985], and blue-noise spectra, that is, attenuated energy in low frequencies and flat spectrum in high frequencies [Ulichney 1988]. Such point sets are primarily needed for anti-aliased sampling [Glassner 1995], but are also used in other applications including Monte Carlo integration [Pilleboue et al. 2015], stippling [Secord 2002], meshing [Yan et al. 2009], and distributing different kinds of

We thank the anonymous reviewers for their detailed feedback to improve the paper. Thanks to Cengiz Öztireli for sharing the grid test scene. Thanks to Carla Avolio for the voice over of the supporting video clip. Corresponding author is Abdalla G. M. Ahmed, abdalla_gafar@hotmail.com. This work was partially funded by Deutsche Forschungsgemeinschaft Grant (DE-620/22-1), the National Foreign 1000 Talent Plan (WQ201344000169), Leading Talents of Guangdong Program (00201509), NSFC (61522213, 61379090, 61232011), Guangdong Science and Technology Program (2015A030312015), and Shenzhen Innovation Program (JCYJ20151015151249564).

0730-0301/2017/7-ART138 $15.00
DOI: http://dx.doi.org/10.1145/3072959.3073588

objects such as people in crowds or trees in forests [Bradbury et al. 2015].

Blue-noise sets are usually obtained by applying complex refinements on a random point process. There are many algorithms for this purpose, e.g. [Ahmed et al. 2016; Balzer et al. 2009; Cook 1986; de Goes et al. 2012; Fattal 2011; Jiang et al. 2015; McCool and Fiume 1992; Schlömer et al. 2011]. In all these algorithms the production cost (time and memory) is high, which leads to the idea of tabulating the blue-noise sets for subsequent reuse, replacing on-the-fly generation of sample points by a lookup framework. Alternatively, real-time techniques such as jittered sampling [Dippé and Wold 1985] and low-discrepancy sampling [Keller 2012] are commonly used to approximate blue noise.

A common technique for lookup sampling is to distribute the sample points over a set of tiles. The basic idea is as follows: A finite set of distinct tiles is designed, each tile has a unique ID, and is populated with one or more sample points at fixed locations. At run time, copies of these tiles are assembled into an arbitrarily large *tiling*, in accordance with a predefined "matching rule" that constrains which tiles in the set can be placed next to each other.

There are various desirable features in a lookup sampler, depending on the target application; for example:

- **Granularity:** to be able to control the generated number of samples. A single sample per tile would be ideal. Alternatively, the samples should be ordered (ranked [Kopf et al. 2006]) within a multi-sample tile.
- **Progressiveness:** incrementally add more samples depending on the outcome of the previously taken samples. The sampler should retain similar distribution properties over different numbers of samples.
- **Localization:** some applications, e.g. image reconstruction [Dippé and Wold 1985], require the ability to easily locate samples nearby a given location.
- **Adaptivity:** Localization is also sometimes combined with progressiveness; that is, more samples are generated in a specific location, possibly guided by a density map. Example applications include stippling and importance sampling.
- **Optimizability:** it should be possible to apply different optimization techniques to control the placement of the sample points. Any generated tiling should retain the optimal quality of the point distribution.
- **Efficiency:** a look-up sampler has to maintain reasonable costs in terms of memory, processing time, and coding complexity.

The matching rule is arguably the most crucial part for building a successful tile-based sampler. The essence of a matching rule is that it makes a tile of a given ID aware of the IDs of potential adjacent tiles, so that the placement of the sample points on each tile is coordinated with the placement of the sample points on all possible adjacent tiles. Consider, for example, the case of using the tiles to distribute a Poisson-disk point set. Unless the number of possible distinct adjacent tiles is limited, it would be quite difficult to place sample points near tile edges or corners, since this would inhibit placing points near the corresponding edges or corners of all potential adjacent tiles.

The classic approach to sampling with regular (square) tiles is based on Wang tiles [Cohen et al. 2003]. The four edges of each distinct tile are color-coded, and the matching rule is that two tiles can be adjacent only if they bear the same color on the shared edge. This gives each tile an idea about the IDs of potential neighbors on the four sides (North, East, South, West), but there is no explicit information about, say, the North-Eastern neighbor tile. Thus, too many degrees of freedom appear in Wang tilings, which drastically limits optimizability. To address this issue, Lagae and Dutreé [Lagae and Dutré 2006] proposed color-coding the tile corners instead of their edges, offering each tile information about all the possible eight neighbors. While this offers a tangible improvement [Lagae and Dutré 2008], the degree of freedom is still too much, since each corner is set independently. Thus, corner tiles are still unable to sustain a granularity of one sample per tile, neither do they offer a good environment for optimization across tile boundaries.

In this paper we develop a lookup sampler that meets all the listed requirements. It builds on a set of regular tiles, with as few as one sample per tile. At the heart of our solution is a novel string-based matching rule. Instead of matching individual tiles, our key idea is to match whole rows or columns. This way, a tile is not informed *individually* about adjacent tiles on each side or corner, but collectively about potential *combinations* of adjacent tiles.

2 RELATED WORK

Since the early days when Dippé and Wold [1985] proposed using Poisson-disk point sets for anti-aliasing, they also suggested to store such sets on toroidal tiles to avoid the high computational cost of their creation. In a different context, halftoning, Ulichney [1988] studied the frequency properties of a similar distribution, and coined the name "blue noise" to describe its spectral profile. He subsequently devised the "void-and-cluster" method to distribute such point sets also over torodial tiles [1993].

With the advancement of computer graphics, optimization techniques were developed to produce higher quality noise sets [McCool and Fiume 1992], and a single torodial tiling was no longer able to cope with the improved quality [Glassner 1995]. Cohen et al. [2003] introduced Wang tiles to create non-periodic tilings for sampling. These are square tiles with color-coded edges and a matching rule that can be used to create a stochastic tiling. Unfortunately, Wang tiles create visible seams near the edges, because each tile may match many combinations of neighbor tiles, making it difficult to place points near tile edges. Lagae and Dutré [2006] suggested color-coding the corners instead of edges, and also proposed a more systematic approach for placing the points on the tiles. Kopf et al. [2006] presented recursive Wang tiles, a quite sophisticated tiling structure that enables seamless tiling; side-by-side as well as recursively, which makes the method suitable for adaptive sampling. In addition, points on each tile are ordered (ranked), which enables progressive sampling.

Lagae and Dutré [2008] report that the quality of conveyable noise sets is low for the Wang tiles family of lookup samplers, and the granularity is poor, starting from 256 samples per tile for a decent quality blue noise (2048 for recursive Wang tiles), which is not suitable for some applications, e.g. when it is desirable to locate nearby samples. Furthermore, optimizing the placement of samples

across tile boundaries is rather difficult, which limits Wang-tile based point sets to distributions based on dart-throwing [McCool and Fiume 1992].

An alternative approach was introduced by Ostromoukhov et al. [2004], using Penrose tiles. The key principle is to use a complex recursive tiling structure that itself creates a spectrum that resembles blue noise. This allows to place only a single sample per tile, which makes optimization substantially easier. This approach is well-suited for adaptive sampling, and aims at substantially higher spectral qualities than Wang tiles; but this comes at a considerable cost in memory, since the whole tiling structure has to be stored. Through two subsequent steps of development [Ostromoukhov 2007; Wachtel et al. 2014], this approach reached a quality that enables almost full control over the spectral properties of the conveyed point sets, using sophisticated optimization techniques [Heck et al. 2013; Öztireli and Gross 2012; Zhou et al. 2012] that were developed concurrently. Unfortunately, to that end the required memory footprint grows beyond the practical limits of many applications: gigabyte-sized lookup tables for a single spectral profile. Another drawback of this family of lookup techniques is that such complex tilings are difficult to map to a unit square, a desirable feature for some applications such as quasi-Monte Carlo integration [Keller 2012].

In all the aforementioned tile-based approaches, locating nearby samples is difficult, especially at locations near the edges of two or more tiles. A different approach that overcomes this localization problem was presented by Ahmed et al. [2015], using so-called AA Patterns. These are analytical grid-based point sets that can be morphed into a desired distribution. The approach is very efficient in terms of memory usage and speed. It is also quite flexible for applying different kinds of optimization, enabling spectral control, while all necessary operations are facilitated over a single toroidal point set. It is, however, non-adaptive, and can therefore only be used for applications that require a uniform density of sample points.

Our proposed framework inherits features from most of the aforementioned sampling approaches. In addition, the way we scan our tiles is closely related to ordered dither matrices [Ulichney 1987], as well as the implicit scanning order of some low-discrepancy sequences [Grünschloß et al. 2012].

3 STRING-BASED IDENTIFICATION

Our goal is to build a finite set of regular tiles that would be assembled into a tiling at run time to place the sample points carried on every tile. To enable progressive and adaptive sampling we also make our tiles recursive, as illustrated in Fig. 2, so that the same tile-set is able to tile a multi-resolution lattice.

In this section we design a matching rule that guides the placement of the tiles on the lattice, as well as on other tiles, and in the following section we describe how to optimize the placement of the points on the tiles.

As mentioned in the introduction, our key idea is to match whole rows and columns rather than individual tiles. Thus, our first step is to use an orthogonal identification for rows and columns, such that the ID of any tile is an ordered pair of the IDs of the distinct column and row in which the tile may fit. We will talk about columns, but the same discussion holds for rows.

Assume that we have already assigned tile IDs, built a matching rule, and used it to make a valid tiling. The distinct IDs can be represented by symbols from a finite set:

$$\mathbf{X} = \{A, B, \cdots\}, \tag{1}$$

As we traverse through any row, we read a string of symbols of column IDs:

$$\mathbf{S} = S_0 S_1 S_2 \cdots \tag{2}$$

where each S_i is a symbol from $\mathbf{X}$. Different tilings are described by different strings that use the same set of symbols. That is, a row or a column with a certain ID, say A, is identical in a given tiling, but might be different from a column with the same ID in a different tiling. In the following discussion, however, we use the symbol $\mathbf{S}$ to stand for what we read in any tiling that adheres to the matching rule.

The set $\mathbf{X}$ and the set of strings $\mathbf{S}$ comprise all the information in our tile identification scheme, and we will now tailor properties of these variables to impose certain desired features on this scheme.

The first property we may set is the size of $\mathbf{X}$, which is the number of IDs per dimension. This is directly reflected in the size of the final lookup-tables. Thus, it is desirable to keep $|\mathbf{X}|$ as small as possible.

Once the number of distinct IDs is decided, we need to decide on the number of potential distinct neighbor columns for a given column ID. From an optimization perspective, we want to keep this number low, as we discussed in the introduction, so we allow at most two distinct neighbor columns for any column. In our symbolic representation, this is equivalent to saying that any symbol from $\mathbf{X}$ may be followed (or preceded) by only one or two distinct symbols in $\mathbf{S}$. This is our equivalent of a matching rule, and dramatically reduces the string $\mathbf{S}$ to an equivalent of a binary string $\mathbf{W}$. Given the initial symbol, S_0, we may traverse $\mathbf{S}$, symbol by symbol, and write '0' or '1' if the current symbol is followed by its first or second allowable successor. If, however, the following symbol is not allowed, then the tiling does not respect the matching rule.

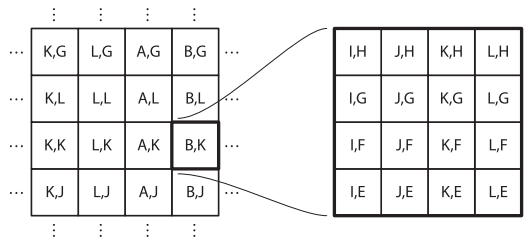

Fig. 2. On each cell on a regular lattice we place a copy of one of our tiles, in accordance with a matching rule. Each tile is itself a tiling of smaller copies of tiles from the same set, similar to recursive Wang tiles [Kopf et al. 2006].

So far we have been analyzing a fictitious matching rule that meets our needs. To actually build the matching rule we may work the other way around; starting from a binary string, $\mathbf{W}$, that represents a valid tiling. The ID, I, of any column X, is defined as the binary string "$W_{X-n+1} \cdots W_{X-1} W_X$" comprising the n entries of $\mathbf{W}$ preceding and including the Xth entry; where n is a design parameter that controls the number of IDs. The first $n-1$ IDs have to be

assigned manually. Alternatively, the string **W** has to be defined periodically.

The properties of the string **W** are directly reflected on the identification scheme. For example:

- A periodic string generates a periodic sequence of IDs.
- The *complexity* [Lothaire 2002] of **W**, that is, the number of *factors* (distinct substrings) of length n, determines the number of distinct IDs.
- The number of factors of length $n+2$ determines the number of combinations of immediate neighbors, or more generally:
- The number of factors of length $n + 2r$ determines the number of combinations of neighbors in a range r on each side.
- A *morphic* string, that is, a string that is the stable point of a *homomorphism* (replacement rule), defines a recursive identification scheme.
- If the ith entry of the string can be computed from i, we obtain a randomly accessible identification scheme.

A binary string that meets our typical needs is the Thue-Morse word [Allouche and Shallit 1999; Lothaire 2002]:

$$T = 01101001100101101001011 0 \cdots \tag{3}$$

There is more than a single way to describe this string. For example, it is obtained by starting with 0 and successively appending the Boolean complement of the sequence obtained thus far. The Thue-Morse word is a fixed point of the homomorphism:

$$\mu : \begin{array}{l} 0 \mapsto 01 \\ 1 \mapsto 10 \end{array} . \tag{4}$$

It is a low-complexity string, which greatly simplifies optimization, since only a few IDs have more than one neighbor ID. By applying Eq (4) i times, each symbol is replaced by $s = 2^i$ symbols. This specifies the subdivision ratio of the tiles. The morphed string then identifies columns and rows in a proportionally higher resolution lattice.

The Thue-Morse word is non-periodic. For most practical applications, however, we would not have to deal with T; it is sufficient to work with a periodic sequence of IDs generated from a periodic sub-string of T.

Let us give a small example of how tile IDs are created, and how they are transformed using the homomorphism. Suppose that we set the length n of ID-strings to 5. The string T has 12 factors of this length, all of them are found (with periodic wrapping) in the first 24 bits shown in Eq (3). Thus, we have 12 IDs. Since T is a fixed point of the homomorphism in Eq (4), these IDs would not change by applying this homomorphism. Table 1 displays the relation between these IDs as we traverse T sequentially or recursively. For example, on the right of a column bearing an ID A we can only find an ID B, because the string "011011" never exists in T. Similar to A, most IDs in Table 1 have only one successor, only $\{D, F, H, L\}$ have two.

To give an example of the recursive traversal of IDs, we start with an ID C, our symbolic name for the bit string "10100". Applying the homomorphism of Eq (4), we get "1001100101". We need 5 symbols to define an ID, so only the rightmost 6 bits of the morphed string define the next-generation IDs; namely, an I ("10010") followed by a J ("00101"). Thus, only the 3 rightmost bits of the ID-string of the parent tile are relevant in determining the IDs of child tiles, which means that even though there are 12 distinct IDs, there are only 6 distinct children structures upon subdivision. Note that the sub-tiles of C are always IJ irrespective of the neighboring tiles. Subsequent application of Eq (4) give us:

$$C \to IJ \to GHEF \to IJKLABCD \to \cdots .$$

Fig. 3 illustrates how this subdivision and mapping of IDs works in 2D. Please note that since the whole string T is generated recursively from one bit, any ID would eventually have descendants of all the other IDs.

ID-string	ID	on 0	on 1	Morphed ID-string	Children IDs
01101	A	B	-	0110100110	EF
11010	B	C	-	1010011001	GH
10100	C	-	D	1001100101	IJ
01001	D	I	E	0110010110	KL
10011	E	F	-	1001011010	AB
00110	F	G	A	0101101001	CD
01100	G	-	H	0110100101	IJ
11001	H	I	E	1010010110	KL
10010	I	-	J	1001011001	GH
00101	J	-	K	0101100110	EF
01011	K	L	-	0110011010	AB
10110	L	G	A	1001101001	CD

Table 1. An example set of string-based IDs using the 12 distinct 5-bit factors of the Thue-Morse word T, and a single application of the homomorphism in Eq (4) to generate IDs of sub-tiles when the resolution of the underlying grid is doubled. "On 0/1" columns show the following ID when a 0/1 is encountered in the subsequent entry of T. The dimmed portions in the fifth column indicates an irrelevant portion of the morphed ID string.

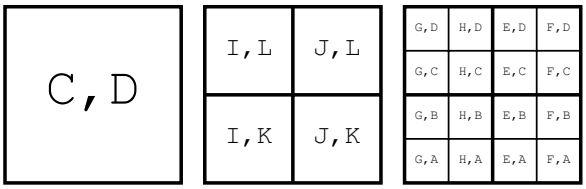

Fig. 3. Illustration of how a 2D tile ID is translated in two steps of subdivision. The column and row IDs are translated independently according to Table 1, and IDs of the children tiles are obtained as a Cartesian product.

Discussion: It is worth mentioning that the identification scheme of AA Patterns [Ahmed et al. 2015] is equivalent to a binary identification that uses so-called Sturmian words [Lothaire 2002], but it further adds more complexity by splitting the string between evenly and oddly indexed rows and columns, and by taking only a subset of rows and columns. Periodic identification schemes, such as that of LDBN [Ahmed et al. 2016], are also equivalent to binary strings: for a $t \times t$ periodic set, the corresponding string is a t-digits periodic binary string containing only one digit set to 1.

4 OPTIMIZATION

With our string-based recursive identification scheme we now proceed to build a sampler by optimally placing sample points on the tiles.

4.1 Optimizing Point Positions

Optimal placement of sample points means that any tiling that observes the matching rule should produce an optimal distribution of the conveyed sample points. Thus, the locations of the sample point(s) on each distinct tile should be coordinated with the locations of the sample points on all possible neighbor tiles. The size of relevant neighborhood depends on the target distribution. For Poisson-disk distributions, for example, only immediate neighbors matter.

To optimize the locations of the points we make a tiling that captures all the possible combinations of neighbors allowed by the matching rule. This is done by assigning tile IDs from a periodic sub-string of the Thue-Morse word that is long enough to comprise all the factors of length $n + 2r$, where n is the length of ID strings, e.g. 5 in Table 1, and r is the required neighborhood size, e.g 1 for common blue noise distributions. Such a tiling would furnish a toroidal domain for optimizing the point set. To give an example, a periodic string of the first 24 bits of T shown in Eq (3) contains all the 7-bits-long factors of T, and as such encompasses all the possible combinations of distinct columns and their immediate neighbors for the 5-bits-long set of IDs shown in Table 1. Thus, the string "GHEFABCDEFGHIJKLABCDIJKL", more specifically, a 2D Cartesian product of it, defines a toroidal optimization environment for this tile set for placing blue noise samples.

Once the optimization environment is set up, we proceed to optimize the point locations, starting with a random location for each distinct tile, and then iteratively adjust the locations of the points towards the optimal distribution. We focus on blue noise, but the process is similar for general noise. It is an advantage of our framework that numerous optimizations techniques could be used. We prefer serial algorithms that work on a point-by-point basis, since such methods work much better under the constraints of tile boundaries and multiple combinations of neighbors.

Typical optimization algorithms include Lloyd's algorithm [Lloyd 1982; McCool and Fiume 1992], a localized version of Farthest Point Optimization (FPO) [Schlömer et al. 2011], and the more recent Push-Pull Optimization (PPO) [Ahmed et al. 2016]. Since all the mentioned algorithms are based on a Delaunay triangulation of the point set, they can easily be combined, leading to a sequence of optimizations applied to each visited point. Algorithm 1 summarizes the framework we used for optimization. The acceptance criterion varies between algorithms; for example, PPO automatically terminates once the prescribed spatial measures are satisfied. Please note that restricting the points to stay within tile boundaries is not absolutely necessary, but is highly desirable for some applications such as Monte Carlo integration. It is also a pre-requisite for building the recursive tiles in subsequent sections.

Combating the Grid Structure: Optimizing under a regular lattice constraint is challenging [Ahmed et al. 2016], but we experimentally found an effective solution by intertwining Lloyd's algorithm and FPO. For each visited point, we first apply a step of Lloyd's relaxation, moving the point to the centroid of its Voronoi cell, followed by an FPO step that moves the point farthest away from its (new) neighbors. FPO has a good performance in avoiding regular structures, as observed by Schloemer et al. [2011], while Lloyd's method improves coverage and fills the holes commonly found in FPO profile. On top of that we found it helpful to occasionally Latinize [Saka et al. 2007] the point set to get rid of any remaining grid structure. It takes only 10 iterations to reach the quality in Fig. 10. Once the point set is sufficiently isotropic, other optimizations may be applied.

ALGORITHM 1: Optimizing the placement of points using a representative toroidal tiling.

```
foreach ID do
    assign a random point location;
Populate the tiles with the points according to their IDs;
repeat
    foreach point, in a random scanning order do
        foreach entry in the optimization sequence do
            find the optimum location of the point and/or the neighbors
            according to the designated optimization;
            foreach suggested optimum location do
                if the point would stay within the tile then
                    move the point;
                    update the locations of all the points in same-ID tiles;
until the distribution is acceptable;
Store the final point locations (per tile ID) in a lookup table.
```

Multi-Class Blue Noise. Our framework can also produce multi-class blue noise – a blue noise point set that is a composite of many blue noise point sets [Wei 2010]. For this we distribute multiple samples per tile, one sample for each class. Algorithm 1 can easily be adapted for multi-class optimization by splitting each iteration into two passes: we first optimize the points within each class, then optimize globally over the whole set. Displacements of the global pass are substantially smaller than in the class-wise pass, hence the optimization converges. The two passes do not have to use the same optimization algorithm. For instance, the point set in Fig. 1(b) uses Lloyds-FPO for global optimization and conflict-coverage PPO for per-class optimization; note the difference in the spectra in Fig. 11

Spectral Control. Target matching algorithms [Heck et al. 2013; Öztireli and Gross 2012; Zhou et al. 2012] can easily be adapted to our framework thanks to the toroidal domain optimization environment we have. The only change needed is to average the displacements of the points holding the same ID, as advocated by all of [Ahmed et al. 2015; Ostromoukhov 2007; Ostromoukhov et al. 2004; Wachtel et al. 2014]. Confining the sample points to the tile boundary is not mandatory, though it is highly desirable. Please note that the radial distribution function of some noise profiles, e.g. pink noise [Zhou et al. 2012], makes them unrealizable under the regular lattice constraint, and we have to allow the sample points to leave the tile boundaries, or use more samples per tile.

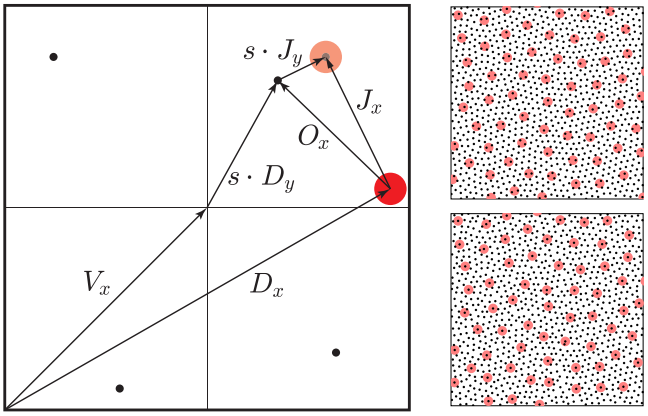

Fig. 4. (a) Snapping a point (red) to the nearest point in a child tile. The displacement vector D comes from initial optimization, s is the self-similarity scale of the tiling, $1/2$ in this illustration. (b) The distribution of a point set (top) before and (bottom) after snapping, demonstrating that the net effect of snapping is a slight jitter. This distribution uses a $s = 1/4$; the jitter would be substantially larger using $s = 1/2$.

4.2 Snapping

So far the tiling can distribute sample points in a linear tiling order, one point per tile, as well as recursively by subdividing the tile into sub-tiles, and putting sample points into these tiles. Recursive generation, however, works only with full octaves, that is, by putting a sample point on each of a complete set of child tiles.

This is why we will now optimize the placement of the samples across octaves by ensuring that the sample point in any tile fits well within the sample points of its sub-tiles. The only way to cope with an arbitrary depth of subdivision is to ensure that the sample point on a tile coincides with the sample point on one of the child tiles, cf. [Kopf et al. 2006]. Thus, we overlay parent and children tiles, and snap the point of the parent to the closest point of one of the children. Instead of the heuristic treatment in [Kopf et al. 2006], however, we provide a closed-form solution.

As illustrated in Fig. 4, we want to snap the (red) point of a parent tile with ID x, to coincide with the (black) point in a child tile that holds an ID y. The target point also moves, since it has to be snapped to its own target in a grand-child tile. Please note, however, that the target, being at a smaller scale, is moving slower than the chasing point of the parent tile, which guarantees convergence. Our goal is now to compute the final value of the displacement D_x when all the points of all IDs coincide with their targets. We also want to estimate the amount of jitter, J, experienced by each point after this process.

We start by factoring D_x into a sum of two vectors:

$$D_x = V_x + sD_y\,, \tag{5}$$

where s is the scaling factor. The vector V_x is the origin of the designated child tile relative to the parent tile, and is invariant. The other component, D_y, is a variable that can be computed by applying Eq. (5) recursively to the tile ID y. This leads to the following formulation:

$$D_x = \sum_{i=0}^{\infty} s^i V_{h^i(x)}\,, \tag{6}$$

where $h(x)$ is the ID of the nominated target child for a tile of ID x (that is y in Fig. 4), $h^i(x) = h\left(h^{i-1}(x)\right)$, and $h^0(x) = x$.

Since the number of ID's is finite, by design, we will eventually have $h^j(x) = x$ for some $j \le N$, the number of IDs; hence we obtain the following closed-form formulation:

$$D_x = \frac{\sum_{i=0}^{j-1} s^i V_{h^i(x)}}{1 - s^j}\,. \tag{7}$$

In practice, the computation is limited by the numeric precision, hence it is sufficient to take a large enough j, irrespective of whether $h^j(x) = x$ or not.

The error vector J_x can be computed in a similar way by factoring it into two components; the only difference is that we now compute it relative to the displaced positions rather than the original positions of the points. We end up with the following equation:

$$J_x = \frac{\sum_{i=0}^{j-1} s^i O_{h^i(x)}}{1 - s^j}\,. \tag{8}$$

The first thing to notice is that the optimization data is only used to nominate the target child; the initial displacement vectors D never appear in Eq. (7). In some sense, this represents a quantization process, and the subdivision scale s plays an important role for the final quality. The example in Fig. 1(a) uses a scaling factor of $1/4$, obtained by applying Eq. (4) twice.

Besides the scaling factor, the quality of the initial optimization directly affects the final results. The coverage radius determines how far the nearest point would be to each point (the offset vector O), hence it is directly reflected in the expected jitter J. The conflict radius has no direct influence on the jitter, but on the impact of these jitters, as they would bring some points closer to each other.

This analysis is generic and works for different tiling schemes, e.g. [Kopf et al. 2006; Ostromoukhov 2007; Wachtel et al. 2014], but it simplifies greatly for our case that uses orthogonal coordinates. Each ID is assigned $\log_2(1/s)$ fractional bits, on each axis, to indicate the row and column of the nominated target child; then Eq (7) reduces to concatenation of bits.

Thus, the overall effect of snapping the points is a random jitter sized by the coverage radius and the scaling factor s. We want to have a large conflict radius and a small coverage radius. This directly suggests FPO and Lloyd's algorithm, respectively, which is another reason to favor this combination. We may further assist these algorithms by applying coverage and conflict optimization using PPO. It is worth noting that the conflict and coverage radius are usually measured relative to the packing radius of a triangular grid, and to convert them in relation to our regular grid spacing we need to divide by $\sqrt{2/\sqrt{3}}$.

Finally, we found that higher quality sets can be obtained by applying cycles of optimizing (Algorithm 1) and snapping, so we integrated the snapping process in the optimization code. We ended up applying the snapping step after each iteration of the optimization algorithm.

4.3 Ranking

After snapping, the tiles can deliver coherent sets of samples progressively in full octaves. The next step is to optimize them to deliver also subsets of octaves. For this goal we follow the procedure suggested by Ostromoukhov [Ostromoukhov 2007] for Polyominos: we assign ranks to the children of each tile to work with different densities. Like Ostromoukhov we use an adaptation of Ulichney's void-and-cluster method [1993], which is also reflected in methods for ordered dithering [Bayer 1973]. In our case the first rank is always assigned to the child whose sample point coincides with the parent's.

Ranking is the final necessary step in our construction, which we call Adaptive Regular Tiles (ART) sampler. Afterward we store all the optimization information in a single lookup table. The record for each tile ID contains the coordinates of the sample point(s), plus the IDs and ranks of the child tiles. This table actually represents an infinite non-periodic set of sample points, and each sample holds a unique sequence number in the top-level tile, obtained by left-concatenating the ranks of its ancestry, then "bit-reversing". Algorithm 2 describes how to retrieve the ith sample in a tile, and the idea is visually illustrated in Fig. 1. Please note that in contrast to Polyhexes [Wachtel et al. 2014] and Polyominos [Ostromoukhov 2007], we do not need to store any geometric rules to define the tiles.

ALGORITHM 2: Retrieving the ith sample in a tile given the top-level tile ID. The variable s is the subdivision ratio (per axis) of the tiling.

```
1 while i > 0 do
2     childNo ← tile[id].order[i % s²];
3     id ← tile[id].childID[childNo];
4     translate to (childNo % s, childNo/s);
5     scale by 1/s;
6     i = i/s²;
7 return tile[id].point.
```

4.4 Further Optimization

We may optionally implement further application-specific optimizations. For example, since our point sets are stratified by construction, they are ripe for enforcing the low-discrepancy property on the generated point sets by applying Algorithm 2 of [Ahmed et al. 2016].

One interesting post-optimization, due to Ostromoukhov [2007], is to optimize the distribution at each rank (density) to achieve good qualities between octaves. That is, the distribution is post-optimized for each of the densities in the bottom on Fig. 1. Our framework offers a 2×2 subdivision ratio, with only three ranks, which substantially reduces the size of the involved lookup tables. A 4×4 subdivision gives a smoother transition of the sample locations with density changes, but at the cost of substantially larger lookup tables. This step would improve the visual appearance of the point set, and suits applications like stippling, but the point set would no longer be progressive; that is, no more samples can be inserted without disturbing the distribution quality. We describe our own implementation of Ostromoukhov's idea in the supplementary material.

5 APPLICATIONS

Quasi-Monte Carlo integration is arguably the most important application of point samplers in computer graphics, so we proceed here by highlighting two example application scenarios where our design will be advantageous, then we briefly discuss other applications.

5.1 Area-Light Sampling

Theoretical analysis on Monte Carlo integration error/variance, by different groups [Durand 2011; Öztireli 2016; Pilleboue et al. 2015; Ramamoorthi et al. 2012; Subr and Kautz 2013], leads to the intuition that a sampling pattern with a blue noise spectrum is a good choice for area-light sampling. Yet, low-discrepancy sequences continue to prevail as the favorite sampler for this purpose [Pharr and Humphreys 2010]. Notably, low discrepancy sequences are not only much easier to produce, but they also tend to give better results; which is quite surprising and not intuitive at first sight.

One possible explanation relates to the way how shadow samples are eventually aggregated to decide the per-pixel radiance. For a super-sampled image [Glassner 1995], many samples are taken around each pixel. For each pixel-sample a set of shadow-samples is generated, for each light source, and these sets are aggregated for the final pixel radiance. Most of the theoretical analysis consider shadow samples as a single sample set and neglects that many sample sets are actually being averaged. On the practical side, common blue noise samplers can not do better than generating the shadow sample sets independently for each pixel-sample. In contrast, a low-discrepancy sequence can produce negatively-correlated sets of samples to reduce variance, by generating the whole set of shadow samples as a single sequence, and assigning sub-sequences of it to individual pixel-samples, as described by Pharr and Humphreys [2010]. This grants a low discrepancy sequence of samples for each pixel sample, as well as for the whole set.

Our framework offers more than one way to emulate the partitioning capability of low-discrepancy sequences. The simplest configuration uses a randomly chosen tile of our set to produce a sequence of blue-noise samples; the same way as a low-discrepancy sequence does. A more sophisticated setup combines the side-by-side and recursive tiling order of our tile set. To produce i^2 light samples for each of p pixel samples, we construct an $i \times i$ tiling, and sequentially add p samples to each tile. The first sample on each tile is assigned to the first pixel-sample, the second to the second, and so on. This assigns a stratified sample sub-set to each pixel-sample, and a blue-noise super-set to the whole pixel. Yet another possibility is to use the multi-class concept described by Wei [2010], using one class for each pixel-sample. It assigns a blue noise set to the individual pixels-samples as well as the whole pixel.

Fig. 5 illustrates these ideas for partitioning the point set, which are still at an early stage of development. The initial results are promising; at least as good as the PBRT low discrepancy sampler, as demonstrated in the rendering results of Fig. 9.

5.2 Importance and Adaptive Sampling

The recursive nature of our tiling makes it suitable for importance sampling, where a map is provided to control the density of samples. As already pointed out by McCool and Fiume [1992], importance

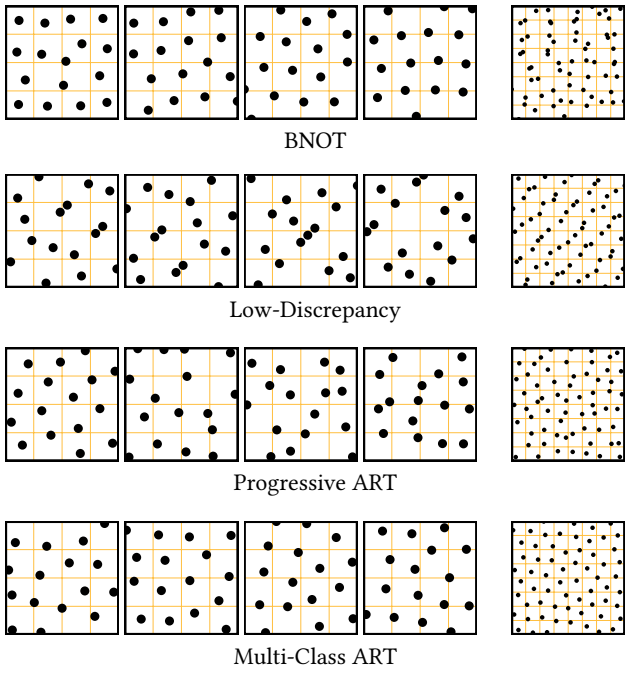

Fig. 5. Four individual shadow sample sets and the combined set for different samplers. A typical blue noise sampler like BNOT offers excellent individual sets, but the combined set is poor. In contrast, low-discrepancy sampling is capable of maintaining the same quality for the sub-sets and the combined set. Our framework tries to emulate this capability.

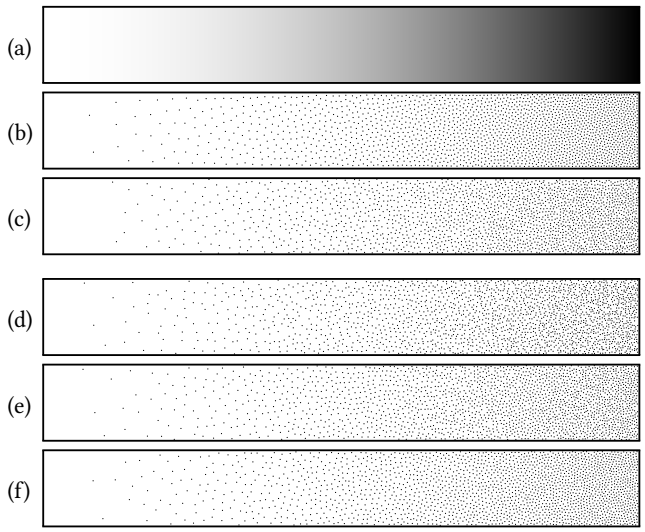

Fig. 6. Sampling of (a) a quadratic density map using (b) an optimization-based sampler (BNOT), (c) Polyhexes, the current state-of-the-art, and (d)-(f) our method: (d) 2×2 progressive ART, (e) 2×2 post-optimized ART using the initial table of (d), and (f) 4×4 post-optimized ART. Notice how the points are relocated between (d) and (e). Even at 2×2 subdivision ratio our post-optimized sampler compares with the state-of-the-art in adaptive lookup sampling. At 4×4 we are evidently better.

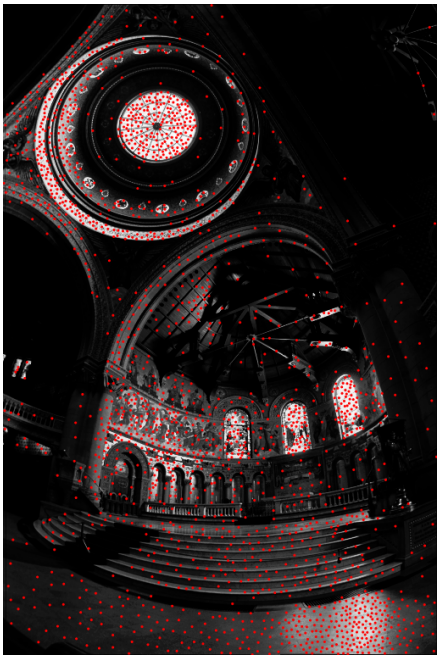

Fig. 7. Example importance sampling. Our method is able to distribute samples over a large density variation.

sampling is equivalent to the well-studied problem of ordered dithering [Ulichney 1987].

Each of our tiles, with sequentially ordered sub-tiles (and sample points), represents a matrix for ordered dither with an infinitude of thresholds, optimized to recursively deliver a blue noise distribution. This is the setup developed by Ostromoukhov [2007] for Polyominos. We were inspired by his work, but we use a much simpler structure for the tiles. The square tiles of our design have an important advantage over Polyominos: For most applications, tiles have to be clipped to sample a rectangular domain. For the complex shapes of a G-Hexomino, for example, 33% of the samples would fall outside the sampled domain, and the corresponding thresholds would be lost. In contrast, our tiles match the sampled domain, so we retain a smooth linear gradient of thresholds. Fig. 6 demonstrates how our samples can faithfully reproduce a density map, and Fig. 7 shows an example importance sampling with our sampler.

For some adaptive sampling scenarios the required number of samples is not known apriori but is determined from the outcome of previously taken samples [Glassner 1995]. Our progressive design is especially advantageous for these scenarios, since the new samples fit well with the already taken ones. It is worth noting that Grünschloß et al. [2012] studied ways to add similar functionality to low-discrepancy sequences, which are defined only globally. In contrast, localization functionality is enabled by design in our framework. Fig 8 illustrates how our tiles adaptively subdivide to retrieve more samples.

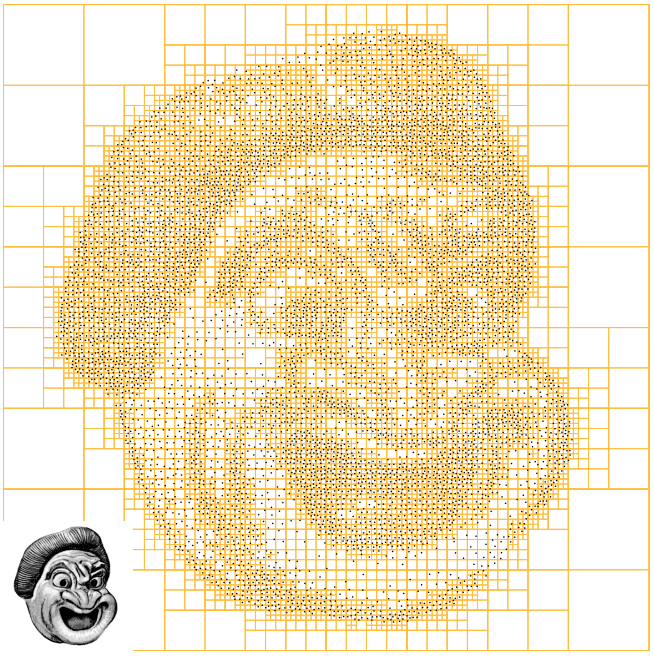

Fig. 8. Adaptive localized subdivision of the tiles to add more samples.

5.3 Other Applications

Our framework combines features of all major lookup samplers, and can therefore serve as a substitute for existing implementations of Penrose tiles, Wang tiles, Polyominoes, Polyhexes, or AA Patterns, in a number of applications ranging from stippling, to various systems for distributing objects in films and games (crowds), biology (trees and other plants) or texturing.

One unique advantage of our framework is that it readily scales to three or four dimensions. This could be useful for distributing particles in 3D, or using Monte Carlo integration to estimate volumes.

6 RESULTS

Table 2 shows a qualitative comparison of our method with some of the existing samplers. In the rest of this section we discuss some practical aspects of our design and the way it delivers the samples.

6.1 Design Parameters

A few parameters have to be set to define our tile system. The first important parameter is the number of IDs. For our self-similar construction during all our tests we considered a 4k (64×64) set of IDs, as advocated in [Ahmed et al. 2015]. This corresponds to identification strings of 21 bits from the Thue-Morse word T. If we use more than one sample per tile, we may consider fewer IDs, since we have another degree of freedom then. In our experiments with multi-class blue noise sets, for example, we were satisfied with the results created by using only 44×44 sets of IDs. Fig. 1 and Fig. 5 show the quality of these sets.

Once the number of IDs (and the length of the ID bit strings) is determined, we need to choose a sub-string of T to define the optimization environment. As discussed in Section 4, the length of this sub-string depends on the neighborhood we wish to consider during optimization. For blue noise, only one or two levels of neighbors need to be considered, but the framework is flexible and allows us to define an arbitrary wide neighborhood. It is a property of T that any prefix of length $3 \cdot 2^k$ toroidally wraps the first few elements. The first 96 elements of this string can be wrapped periodically to contain all the 21-bit ID strings, and all combinations of six neighbors on each size. Thus, our optimization environment was a single 96×96 toroidal set of tiles for the self-similar construction, and 48×48 tiles for the multi-class configuration.

Finally, we need to define a subdivision ratio for the self-similar tiling. The subdivision ratio needs to be a power of 2 per axis[1]. This is a trade-off between many aspects, including the quality of the distribution at full octaves, the quality between octaves, the granularity of subdivision, the size of tables for the post-optimized variant, and the uniformity of distribution. We observed that a too-optimal distribution at full octaves tends to produce poorer distributions between octaves. For many applications we find 4×4 to be a good compromise.

6.2 Performance

All our experiments were computed on a Core i7 desktop with 16 GB memory. The first phase of optimization, placing the points, takes between a few seconds using classic blue noise optimization techniques (Lloyd, FPO, etc), and a few minutes using the target-matching algorithm of [Heck et al. 2013]. Snapping does not take any time, since it comprises only a few bit operations. Ranking takes around two minutes for 16 children. Thus, our complete lookup tables can be created within a few minutes. It might be desirable, though, to feed the tables back through a few cycles of optimization-snapping, which takes more time.

At run time, our framework delivers more than 270M samples per second for uniform sampling if the Thue-Morse word has to be evaluated, reaching 390M when using a periodic sub-string, which is faster than any lookup sampler we are aware of. In recursive mode our sampler is an order of magnitude slower, delivering 13M samples per second, but is still faster than the speeds reported for common adaptive samplers [Wachtel et al. 2014]. Thus, our sampler meets the speed requirements of time-sensitive applications such are real-time rendering. Indeed, we tested our sampler in PBRT, and did not notice any speed difference from the default low-discrepancy sampler.

6.3 Quality

Fig. 10 shows some of the results we obtained from optimization. Overall, the quality is good, though it can be seen that for all the spectra there is some fixed level of anisotropy. This is inevitable in lookup samplers and comes from the fact that some parts of the distribution have to be repeated elsewhere in the point set. The shown plots use 96×96 points (our optimization environment) but

[1] In theory the concept supports other subdivision ratios using different homomorphisms than Eq (4), but in practice we would prefer powers of 2 to take advantage of the binary nature of computers.

	ART (ours)	**Polyhexes** [Wachtel et al. 2014]	**Recursive Wang Tiles** [Kopf et al. 2006]	**AA Patterns** [Ahmed et al. 2015]
Granularity	1 sample per tile	1 sample per tile	2048, but ranked	not applicable
Progressive	Yes	No	Yes	No
Adaptive	Yes	Yes	Yes	No
Localization	easy	difficult	difficult	easy
Spectral control	Yes	Yes	No	Yes
Optimization domain	single, toroidal	multiple, non-toroidal	multiple, non-toroidal	single, toroidal
Memory footprint	Kilobytes to Megabytes	Gigabytes	Megabytes	Kilobytes

Table 2. Qualitative comparison of our method to the major categories of lookup samplers. All the four are ultra fast.

only 64×64 degrees of freedom (the number of IDs), which means that, on average, each local neighborhood is found twice.

This anisotropy can be concealed by using larger tables, but would immediately appear when the number of points is substantially larger than the size of the table. Compared to other lookup frameworks, our construction has the advantage that the lookup tables can be scaled freely to any desirable/affordable size. Many practical applications, such as area-light sampling, distributing objects, or stippling, would not be too sensitive to repetitions beyond 4k points, which justifies our choice of this table size.

The more important comment on the spectra is that even though the spectrum might look visually perfect, it might contain some *spikes* that could be harmful for some applications such as rendering. Anisotropy is a good tool to highlight such artifacts. We observed that enforcing self-similarity has the tendency to emphasize harmonic frequencies of the tiling. For our regular tiling we know precisely where these harmonics are located, and we can get rid of them by using Latinization. On the down side, Latinization implies some regularity, and might lead to banding in shadow rendering, as is the case with low-discrepancy sequences. A possibly better treatment is to weaken these harmonics by interleaving Latinization with optimization, which explains the cross in some of our spectra. The jitter implied during snapping (Section 4.2) then removes this negative effect of Latinization, and we obtain a good distribution. This is our current plan, but we do not claim any optimality, and we encourage further research on this aspect. Finally, we observed that the multi-class setup is less susceptible to this problem, which is another reason to prefer it for area-light sampling. This is a natural consequence of having more than one sample per tile, but the self-similar setup has its own merits. Fig. 9 shows some rendering results that highlight these insights.

7 CONCLUSION

We have presented a framework for producing a variety of point sets with applications ranging from sampling to distributing objects. To our knowledge, it is the first adaptive method that uses a simple lattice structure with only a single point per tile and is able to produce point sets with blue noise or other spectral properties. This makes our method suitable for importance sampling, area-light sampling, stippling and many other applications that need point sets with varying densities, and where sampling takes place in a rectangular domain.

We have demonstrated how to optimize point sets using combinations of existing methods. While we were able to produce point sets with good quality, we do not claim optimality, and we are almost sure that better qualities are achievable using other optimization methods and combinations. The main advantage of our framework is that such optimizations can easily be integrated and future works can produce even better point sets. We provide the code of our framework as well as the point sets for further comparison.

While our look-up tables are very compact for the usual application in two dimensions, they grow exponentially with the number of dimensions, thus the method gets impractical for more than four dimensions.

The focus in our paper is on presenting the framework itself, and we think that it will encourage further research on many aspects. We are especially interested in the ranking step. Current ranking algorithms are devoted to blue noise, and it would be great to see ranking techniques that enable varying densities of arbitrary noise spectra.

REFERENCES

A. G. M. Ahmed, J. Guo, D. M. Yan, J. Y. Franceschi, X. Zhang, and O. Deussen. 2016. A Simple Push-Pull Algorithm for Blue-Noise Sampling. *IEEE Transactions on Visualization and Computer Graphics* (2016). preprint.

Abdalla G. M. Ahmed, Hui Huang, and Oliver Deussen. 2015. AA Patterns for Point Sets with Controlled Spectral Properties. *ACM Trans. Graph.* 34, 6, Article 212 (2015), 8 pages. DOI: https://doi.org/10.1145/2816795.2818139

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

Jean-Paul Allouche and Jeffrey Shallit. 1999. The ubiquitous prouhet-thue-morse sequence. In *Sequences and their applications*. Springer, 1–16.

Michael Balzer, Thomas Schlömer, and Oliver Deussen. 2009. Capacity-Constrained Point Distributions: A Variant of Lloyd's Method. *ACM Trans. Graph.* 28, 3 (2009), 86:1–8.

B. Bayer. 1973. An optimum method for two-level rendition of continuous-tone pictures. In *IEEE International Conference on Communications*, Vol. 1. 11–15.

Gwyneth A. Bradbury, Kartic Subr, Charalampos Koniaris, Kenny Mitchell, and Tim Weyrich. 2015. Guided Ecological Simulation for Artistic Editing of Plant Distributions in Natural Scenes. *Journal of Computer Graphics Techniques (JCGT)* 4, 4 (19 Nov. 2015), 28–53.

Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. 2003. Wang Tiles for Image and Texture Generation. In *ACM SIGGRAPH Conference proceedings*. 287–294.

Robert L. Cook. 1986. Stochastic sampling in computer graphics. *ACM Trans. Graph.* 5, 1 (1986), 51–72.

Fernando de Goes, Katherine Breeden, Victor Ostromoukhov, and Mathieu Desbrun. 2012. Blue Noise Through Optimal Transport. *ACM Trans. Graph.* 31, 6, Article 171 (2012), 11 pages. DOI: https://doi.org/10.1145/2366145.2366190

Mark A. Z. Dippé and Erling Henry Wold. 1985. Antialiasing Through Stochastic Sampling. *SIGGRAPH Comput. Graph.* 19, 3 (July 1985), 69–78.

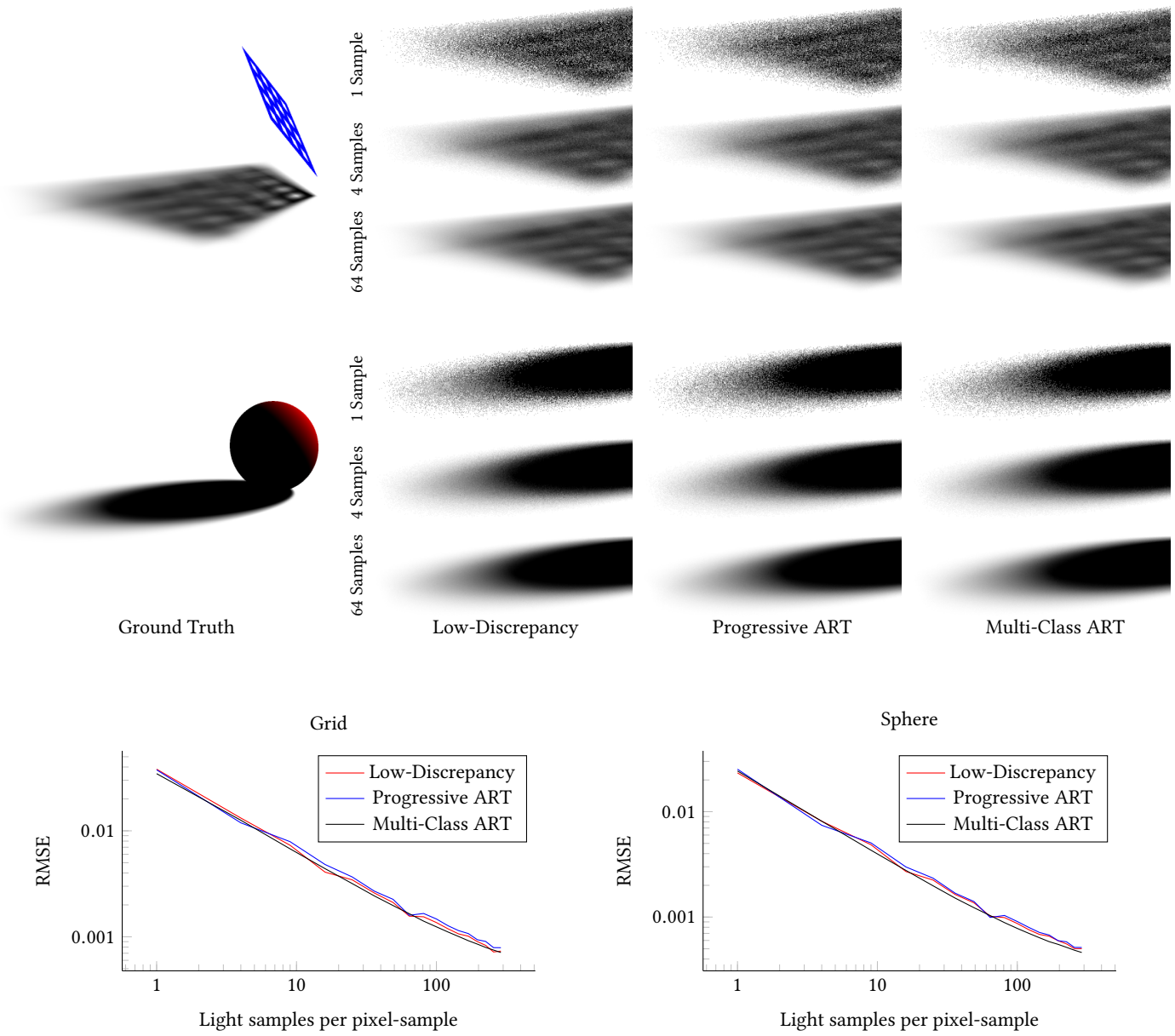

Fig. 9. Rendering results for soft shadows. The ground truth images use four samples per pixel and 1024 light samples. Detail images of different methods use four samples per pixel and a varying number of light samples per pixel sample. "Low-discrepancy" refers to the scrambled low-discrepancy sequences of PBRT. Please note the banding in low-discrepancy towards the end of shadow in the sphere scene. Our samplers, in contrast, provide the typical noise-aliasing trading of blue noise [Glassner 1995]. The complete set of full-resolution images is available in the supplementary materials.

Fredo Durand. 2011. A Frequency Analysis of Monte Carlo and Other Numerical Integration Schemes. *MIT CSAIL Tech. rep. TR-2011-052* (2011).

Raanan Fattal. 2011. Blue-noise point sampling using kernel density model. *ACM Trans. Graph.* 30, 3 (2011), 48:1–48:12.

Andrew S Glassner. 1995. *Principles of digital image synthesis.* Vol. 1. Elsevier.

Leonhard Grünschloß, Matthias Raab, and Alexander Keller. 2012. *Monte Carlo and Quasi-Monte Carlo Methods 2010.* Springer Berlin Heidelberg, Chapter Enumerating Quasi-Monte Carlo Point Sequences in Elementary Intervals, 399–408.

Daniel Heck, Thomas Schlömer, and Oliver Deussen. 2013. Blue Noise Sampling with Controlled Aliasing. *ACM Trans. Graph.* 32, 3, Article 25 (July 2013), 12 pages. DOI: https://doi.org/10.1145/2487228.2487233

Min Jiang, Yahan Zhou, Rui Wang, Richard Southern, and Jian Jun Zhang. 2015. Blue Noise Sampling Using an SPH-based Method. *ACM Trans. Graph.* 34, 6, Article 211 (2015), 11 pages. DOI: https://doi.org/10.1145/2816795.2818102

Alexander Keller. 2012. Quasi-Monte Carlo Image Synthesis in a Nutshell. *Monte Carlo and Quasi-Monte Carlo Methods* (2012), 213–252.

Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. 2006. Recursive Wang Tiles for Real-time Blue Noise. *ACM Trans. Graph.* 25, 3 (2006), 509–518. DOI: https://doi.org/10.1145/1141911.1141916

Ares Lagae and Philip Dutré. 2006. An Alternative for Wang Tiles: Colored Edges Versus Colored Corners. *ACM Trans. Graph.* 25, 4 (2006), 1442–1459. DOI: https://doi.org/10.1145/1183287.1183296

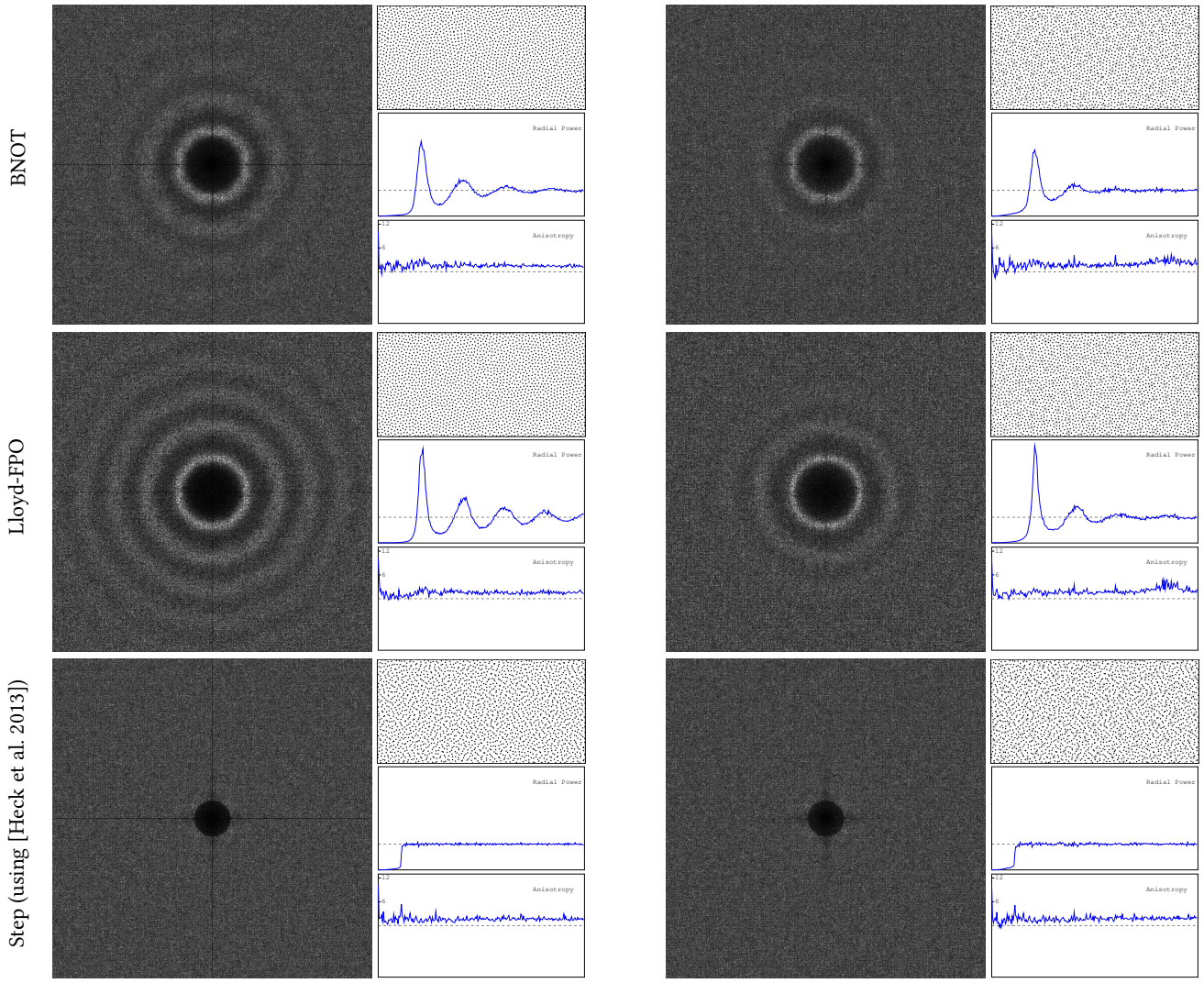

Fig. 10. Resulting point set, spectrum, radial power, and anisotropy for different optimization schemes. On the left we show the tiled point set without enforcing self-similarity, and on the right is the one snapped to achieve self-similarity.

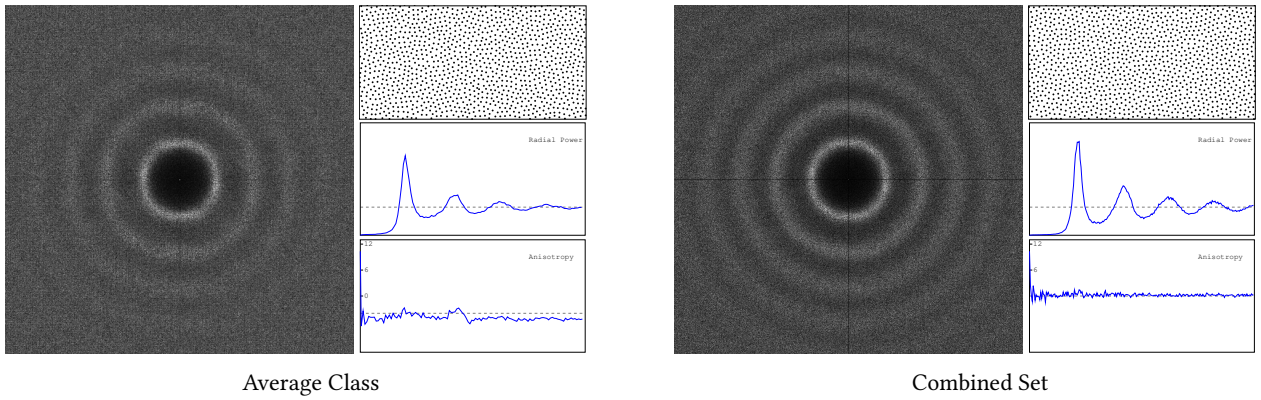

Fig. 11. Spectral properties of four-class blue noise sets, cf. Fig. 1(b) and Fig. 5

Ares Lagae and Philip Dutré. 2008. A Comparison of Methods for Generating Poisson Disk Distributions. *Computer Graphics Forum* 27, 1 (2008), 114–129.

S. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137.

M. Lothaire. 2002. *Algebraic Combinatorics on Words*. Cambridge University Press. https://books.google.de/books?id=JcJniBQ2dD4C

Michael McCool and Eugene Fiume. 1992. Hierarchical Poisson Disk Sampling Distributions. In *Proceedings of the Conference on Graphics Interface '92*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 94–105. http://dl.acm.org/citation.cfm?id=155294.155306

Victor Ostromoukhov. 2007. Sampling with Polyominoes. *ACM Trans. Graph.* 26, 3 (2007), 78:1–78:6.

Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. 2004. Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph.* 23, 3 (2004), 488–495.

A. Cengiz Öztireli. 2016. Integration with Stochastic Point Processes. *ACM Trans. Graph.* 35, 5, Article 160 (Aug. 2016), 16 pages. DOI: https://doi.org/10.1145/2932186

A. Cengiz Öztireli and Markus Gross. 2012. Analysis and Synthesis of Point Distributions Based on Pair Correlation. *ACM Trans. Graph.* 31, 6, Article 170 (2012), 10 pages. DOI: https://doi.org/10.1145/2366145.2366189

Matt Pharr and Greg Humphreys. 2010. *Physically Based Rendering, Second Edition: From Theory To Implementation* (2nd ed.). Morgan Kaufmann Publishers Inc.

Adrien Pilleboue, Gurprit Singh, David Coeurjolly, Michael Kazhdan, and Victor Ostromoukhov. 2015. Variance Analysis for Monte Carlo Integration. *ACM Trans. Graph.* 34, 4, Article 124 (July 2015), 14 pages. DOI: https://doi.org/10.1145/2766930

Ravi Ramamoorthi, John Anderson, Mark Meyer, and Derek Nowrouzezahrai. 2012. A Theory of Monte Carlo Visibility Sampling. *ACM Trans. Graph.* 31, 5, Article 121 (2012), 16 pages. DOI: https://doi.org/10.1145/2231816.2231819

Yuki Saka, Max Gunzburger, and John Burkardt. 2007. Latinized, improved LHS, and CVT point sets in hypercubes. *International Journal of Numerical Analysis and Modeling* 4, 3-4 (2007), 729–743.

Thomas Schlömer, Daniel Heck, and Oliver Deussen. 2011. Farthest-point Optimized Point Sets with Maximized Minimum Distance. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*. 135–142.

Adrian Secord. 2002. Weighted Voronoi Stippling. In *Proceedings of the 2Nd International Symposium on Non-photorealistic Animation and Rendering (NPAR '02)*. 37–43.

Kartic Subr and Jan Kautz. 2013. Fourier Analysis of Stochastic Sampling Strategies for Assessing Bias and Variance in Integration. *ACM Trans. Graph.* 32, 4, Article 128 (2013), 12 pages. DOI: https://doi.org/10.1145/2461912.2462013

Robert Ulichney. 1987. *Digital Halftoning*. MIT Press, Cambridge, MA, USA.

R.A. Ulichney. 1988. Dithering with Blue Noise. *Proc. IEEE* 76, 1 (1988), 56–79. DOI: https://doi.org/10.1109/5.3288

Robert A Ulichney. 1993. Void-and-Cluster Method for Dither Array Generation. In *IS&T/SPIE's Symposium on Electronic Imaging: Science and Technology*. International Society for Optics and Photonics, 332–343.

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

Li-Yi Wei. 2010. Multi-class Blue Noise Sampling. *ACM Trans. Graph.* 29, 4 (2010), 79:1–79:8.

Dong-Ming Yan, Bruno Lévy, Yang Liu, Feng Sun, and Wenping Wang. 2009. Isotropic Remeshing with Fast and Exact Computation of Restricted Voronoi Diagram. *Computer Graphics Forum* 28, 5 (2009), 1445–1454.

Yahan Zhou, Haibin Huang, Li-Yi Wei, and Rui Wang. 2012. Point Sampling with General Noise Spectrum. *ACM Trans. Graph.* 31, 4, Article 76 (2012), 11 pages. DOI: https://doi.org/10.1145/2185520.2185572

Received January 2017; final version April 2017; accepted May 2017