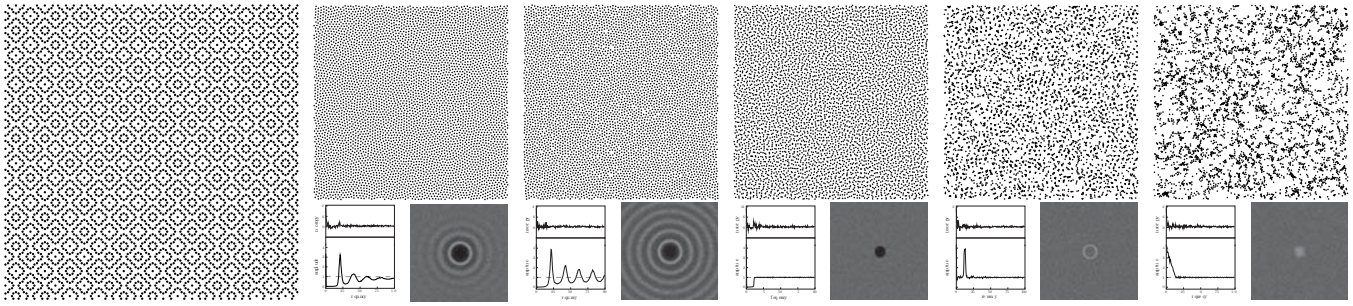
Erschienen in: ACM Transactions on Graphics (TOG) ; 34 (2015), 6. - 212
https://dx.doi.org/10.1145/2816795.2818139

AA Patterns for Point Sets with Controlled Spectral Properties

Abdalla G. M. Ahmed[1] Hui Huang[2] Oliver Deussen[1,2]
[1]University of Konstanz, Germany
[2] Shenzhen VisuCA Key Lab / SIAT, China



(a) (b) (c) (d) (e) (f)

Figure 1: *(a) AA$(183/112)$; converted into b) BNOT blue noise profile [de Goes et al. 2012], (c) FPO-like profile ($\delta_{min} = 0.925$) [Schlomer et al. 2011], (d) step blue noise [Heck et al. 2013], (e) green noise (using [Heck et al. 2013]), and (f) pink noise (using [Heck et al. 2013]).*

Abstract

We describe a novel technique for the fast production of large point sets with different spectral properties. In contrast to tile-based methods we use so-called AA Patterns: ornamental point sets obtained from quantization errors. These patterns have a discrete and structured number-theoretic nature, can be produced at very low costs, and possess an inherent structural indexing mechanism equivalent to those used in recursive tiling techniques. This allows us to generate, manipulate and store point sets very efficiently. The technique outperforms existing methods in speed, memory footprint, quality, and flexibility. This is demonstrated by a number of measurements and comparisons to existing point generation algorithms.

CR Categories: I.4.1 [Image Processing and Computer Vision]: Digitization and Image Capture—Sampling.

Keywords: Sampling Methods, Blue Noise, Tiling, Spectral Analysis.

1 Introduction

The creation of well-distributed point sets is a fundamental problem in computer graphics, since they appear in many applications including sampling, stippling and half-toning, Monte Carlo integration, and distributing objects.

Research on producing point sets with desired properties has been active for more than two decades. In recent years, optimization methods allow synthesizing point sets which match user-specified features (e.g. a target power spectrum or a reference point set). Since the production costs of such sets are typically high, point sets are produced on tiles that are distributed. Such tiling methods usually suffer from a number of inherent quality problems [Lagae and Dutré 2008] such as repetition artifacts and noticeable seams in the point sets, and spurious spikes in their frequency spectra that manifest the frequencies of the underlying tiling structure.

In this paper we propose a new approach for generating arbitrarily large point sets with controlled spectra, at very high speeds and very low memory footprint. Our technique starts from what is called an AA Pattern, a well-selected low-cost point set that is adapted to match a given spectrum. A small number of displacement vectors (around 4k) is enough to define infinitely large point sets without noticeable artifacts.

1.1 Related Work

For a long time research on well-distributed point-sets was dominated by searching for isotropic point sets that exhibit blue-noise spectral properties, as suggested by Ulichney [1988]. Direct production algorithms to generate such point sets include stratified jittering, dart throwing [Dippé and Wold 1985; Cook 1986; Mitchell 1987], and their variants. There are also iterative optimization techniques to modify a given point set, including Lloyd's relaxation algorithm [McCool and Fiume 1992] and its variants [Balzer et al. 2009; Xu et al. 2011; Chen et al. 2012; de Goes et al. 2012], other iterative methods [Schmaltz et al. 2010; Fattal 2011; Schlömer et al. 2011], and the recently invented target-matching algorithms [Zhou et al. 2012; Öztireli and Gross 2012; Heck et al. 2013].

As mentioned above, tile-based techniques were used to produce large point sets from small sets with optimized properties [Lagae and Dutré 2008]. Multiple tiles can be used to avoid the repetition artifacts of a single toroidal tile, but give rise to a new problem: optimal placement of points on each tile needs a way to identify and limit the number of possible adjacent tiles.

Two approaches were proposed to solve this problem: Color-based methods use a plain lattice of square tiles, and use colorized edges

[Cohen et al. 2003; Kopf et al. 2006] or corners [Lagae and Dutré 2006] to identify adjacent tiles. Geometry-based methods employ recursive tilings where the tessellation process enforces having only a few possible adjacent tiles around each tile [Ostromoukhov et al. 2004; Ostromoukhov 2007; Wachtel et al. 2014].

Recursive methods employ a set of so-called 'structural indices': numbers used to identify distinct configurations (shape and orientation) of a tile and its adjacent tiles. Even though the number of structural indices is finite (thanks to self-similarity of the tilings), it grows exponentially with the number of adjacent tiles that have to be identified [Wachtel et al. 2014, supplementary material]. Recursive methods also keep a set of so-called 'production rules' to describe how geometry and structural indices transform upon subdivision.

1.2 Motivation

We have shown that most methods for creating large point sets with controlled spectral behavior are tiling methods. However, such methods have a number of problems that we highlight by identifying requirements for tilings and their lookup tables. This will also motivate our new approach which replaces tiles with a direct index-based mechanism:

1. Even distribution of points. Most tiling techniques achieve this by using tiles of equal area, putting equal number of points inside each tile.

2. An indexing mechanism to assemble tilings without violating the given spectral properties, and to efficiently identify actual or potential locations of nearby points across tile boundaries. Tiling methods do this by matching the shape or color of tile edges or corners.

3. A randomization mechanism for distributing tiles. A proper randomization: *a*) should not repeat periodically, *b*) should avoid local repetition of the same tile or group of tiles, and *c*) should provide random access to individual tiles.

   Early tiling methods based on colors [Cohen et al. 2003] employ a stochastic process to lay tiles, but this could not avoid local repetition, and does not enable random access. Quasi-random sequences were also proposed subsequently for randomization [Schlömer and Deussen 2010]. These are deterministic point sets which sacrifice favorable spectral features of random point sets to gain more control over spatial properties (e.g. uniformity or ability to identify a neighborhood of a point). They form the basis for quasi-Monte Carlo methods [Niederreiter 1992].

   Recursive methods, on the other hand, rely solely on the structure of the tiling for randomization. This poses substantial limitations is terms of flexibility because, to the best of our knowledge, there is no systematic method for designing such tiles; as reflected in the need to store production rules.

4. Enabling optimization across tile borders. A common problem with color-based tiling is that points close to tile edges are treated quite differently by the optimization process than points in the middle, which usually leads to visible seams in the created point sets. An extended set of colors for edges/-corners, and more points per tile, help to conceal this problem [Lagae and Dutré 2008] but are not a general solution. Recursive methods do not suffer from noticeable seams, but it is still difficult to synchronize the optimization process across all possible neighbors of each tile [Ostromoukhov 2007].

5. Using compact look-up tables.

6. Providing a facility to control the density of points adaptively, while maintaining the same lookup tables. Only recursive methods feature this capability.

Unfortunately, all tiling methods fall short in some of the listed requirements. With modern optimization techniques [Zhou et al. 2012; Öztireli and Gross 2012; Heck et al. 2013] a much wider neighborhood becomes involved in determining the optimal placement of individual points. This would exacerbate the seams problem of color-based approaches as well as the memory usage problem of geometry-based approaches. Tables would grow up to the order of gigabytes, as noted by Wachtel *et al.* [2014].

We propose an alternative to tiling techniques which is more flexible and more efficient in computation time and space, and meets most of the listed requirements. Our observation is that *the tiles themselves are not essential*; they are only an auxiliary element to guide the placement of points. Thus, instead of employing a quasi-random process for the layout of tiles, we propose using such a process to directly lay out granular points. These basic point sets are then adapted using a small set of displacement vectors to shape their spatial and/or spectral properties. We demonstrate this using so-called AA Patterns: a family of quasi-random point sets on a regular grid which exhibit desirable properties.

2 AA Patterns

AA Patterns are ornamental point sets which result from quantization errors in 2D forward linear texture mapping [Ahmed 2011c]. They emerge from the interaction of a grid of points $\{(x, y)\}$ (originally: pixels of a source image) with a background array of cells $\{(X, Y)\}$ (originally: pixels of a target viewport), where a linear transformation:

$$\begin{bmatrix} X \\ Y \end{bmatrix} = \frac{1}{2}\begin{bmatrix} \alpha & -1 \\ 1 & \alpha \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} \quad \alpha \in (1, 2) \tag{1}$$

maps points to cells. A point is *set* (included in the point set) iff it hits a specific region of a cell that we will refer to as the "set-region" in this paper; see Figure 2.

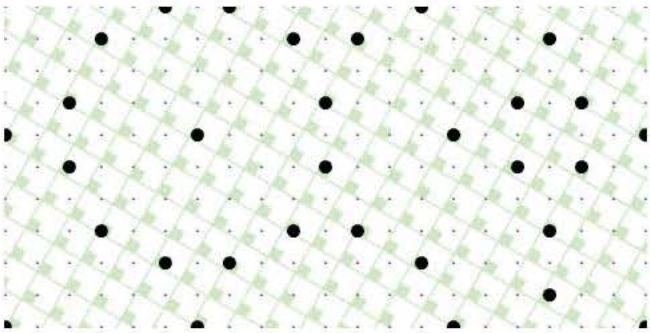


Figure 2: *An AA Pattern is obtained when an appropriately transformed array of cells is used to filter points in the integer grid. A point is retained iff it hits the set (green) region of a cell.*

This description of AA Patterns can be formulated algebraically as:

$$\mathrm{AA}(\alpha) = \{(x, y) : x, y \in \mathbb{Z}; dX < t; dY < t\}, \tag{2}$$

where

$$\begin{aligned} dX &= \mathrm{dX}(x, y) \triangleq \{(\alpha x + y)/2\} \\ dY &= \mathrm{dY}(x, y) \triangleq \{(\alpha y + x)/2\} \end{aligned}, \tag{3}$$

and

$$t = (\alpha - 1)/2\,. \tag{4}$$

The notation $\{.\}$ in Eq. (3) denotes the fractional part of a real number:

$$\{x\} = x - \lfloor x \rfloor$$

Eq. (3) can be seen as a modular hashing function that maps points in the integer grid into 'indices' in the unit torus, while Eq. (2) defines a filtering map inside the unit torus. When α is irrational the index is unique for each point, and the pattern is aperiodic[1]. On the other hand, for a rational parameter $\alpha = p/q$, indices repeat over a $2q$ period, and the pattern is therefore periodic over that period. In that case it is more convenient to write t, dX, and dY as integers, with an implicit $2q$ denominator:

$$t = p - q\,, \tag{5}$$
$$dX = (px + qy)\,\%\,2q\,, \tag{6}$$
$$dY = (qx + py)\,\%\,2q\,. \tag{7}$$

Each period of $\mathrm{AA}(p/q)$ contains t^2 points.

Figure 3 shows an example AA Pattern. Despite their complex appearance, AA Patterns are made up of only a few distinct "clusters" of varying sizes [Ahmed 2011c]. These clusters are interspersed in a hierarchical, self-similar manner [2011a; 2011b; 2012] for example, the distribution of each cluster resembles an AA Pattern (not necessarily the parent one). The density of points is even, which follows from the fact that (dX, dY) make *well distributed sequences modulo 1* [Kuipers and Niederreiter 1974].

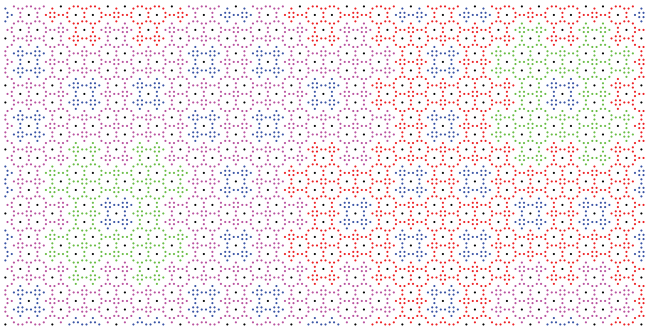


Figure 3: *A part of $AA(\sqrt{3})$, colored to highlight distinct clusters of points (only the smallest four clusters appear in this part).*

The structure of AA Patterns is reflected in index-space: there exists an algorithm [2011c, Algorithm 5] for coloring the distinct clusters using maps in the set-region. These coloring maps identify clusters, which vary in size and are asymmetric around points, and the maps themselves are fractal in nature [2012], so they do not readily suit our purpose. Nevertheless, they indicate a neighborhood identification capability of AA Patterns through indices. Further investigation revealed intrinsic multi-resolution index-space maps that identify symmetric neighbourhoods of arbitrary fixed sizes around points, as will be shown subsequently.

3 Neighborhood Maps in AA Patterns

As implied in Eq. (2), an AA Pattern is separable into two bitmaps, one for each index, combined by a bitwise AND operation (see Figure 4 and [Ahmed 2012][2]). These bitmaps are transpositions of each other. The dX-bitmap is made up of only two repeating rows, and the two rows themselves are interdependent; that is, each row implies the other. This structure offers the convenience of investigating only a one-dimensional profile – a single row of the dX-bitmap – that implies the whole pattern. Let this be the row through $y = 0$, and for sake of brevity we will drop the y parameter of dX:

$$\mathrm{dX}(x) \triangleq \mathrm{dX}(x, 0) = \{\alpha x/2\}\ . \tag{8}$$

Since the row can be represented as a bit-sequence we will use the words "bit" and "point" interchangeably in our discussion.

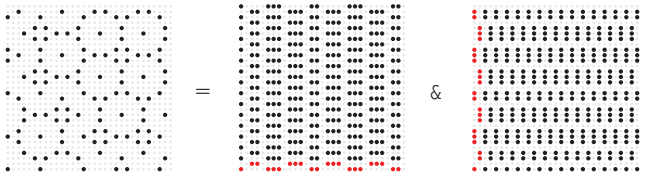


Figure 4: *(a) A part of $\mathrm{AA}(\sqrt{3})$, and its constituent (b) dX-bitmap and (c) dY-bitmap; highlighting the constituent pair of (b) rows and (c) columns.*

Zooming into the row, consider a set point c at distance x_c from the origin:

$$\mathrm{dX}(x_c) < t\,.$$

Now we examine the neighbor point at distance x from c. If the point at $(x_c + x)$ is set, it means that

$$\mathrm{dX}(x_c + x) < t\,,$$

and since

$$\mathrm{dX}(x_c + x) = \{\mathrm{dX}(x_c) + \mathrm{dX}(x)\} \tag{9}$$

we conclude that:

$$\begin{cases} \mathrm{dX}(x_c) < t - \mathrm{dX}(x) & \text{if } \mathrm{dX}(x) < t \\ \mathrm{dX}(x_c) >= 1 - \mathrm{dX}(x) & \text{if } \mathrm{dX}(x) > 1 - t \\ \text{impossible} & \text{if } t \le \mathrm{dX}(x) \le 1 - t \end{cases}, \tag{10}$$

c stands for any point in the row, and the values $(t - \mathrm{dX}(x))$ or $(1 - \mathrm{dX}(x))$ mark thresholds between indices of points which have and those which do not have the xth neighbor set. $(t - \mathrm{dX}(x))$ are right delimiters, and $(1 - \mathrm{dX}(x))$ are left delimiters; the xth neighbor of c is set iff the index of c falls on the delimited side of the threshold. Each offset x which does not return "impossible" in Eq. (10) sets a new threshold.

Now we examine all offsets in a window of width W centered at c. If an offsets returns "impossible" it means that this location is always unset. All other offsets generate thresholds that partition the interval $[0, t)$ into sub-intervals; see Figure 5.

Indices in the same interval mean identical neighborhoods of indexed points, because these indices fall on the same side of every threshold. Conversely, each interval between consecutive thresholds correspond to a unique bit-sequence in the window around c, which can be reproduced by iterating through all thresholds, testing if the interval falls on the delimited side of the threshold, and setting/unsetting the respective bit accordingly. For example, the third (red) interval in Figure 5 falls in the delimited side of the thresholds corresponding to the offsets $\{-7, -2, -1, 6, 7\}$, so it corresponds to a window containing the bit-sequence 1000011 1 0000011; and

[1] This is easier to see in terms of the points-cells interaction: the whole overlap of points and cells is aperiodic for irrational α, and there would always be discrepancy in the filtered point set around any two points.

[2] Since our analysis is based on [Ahmed 2012], we will narrow down our discussion to the constraint $\alpha \in (1.5, 2)$ considered thereat.

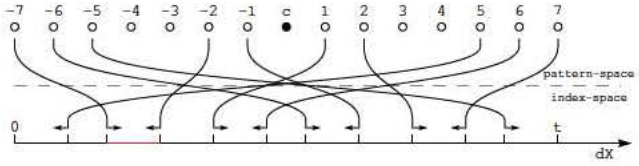


Figure 5: *In a row of the dX-bitmap of $AA(\sqrt{3})$, examining a window of width 15 generates 10 thresholds that partition the indices into 11 intervals, corresponding to 11 distinct pattern-space layouts of neighbor points. We make it clear that '15' is an arbitrarily chosen width, and does not have a special meaning; any other width gives analogous results.*

nearby points can be found 7, 2, and 1 step to the left, as well as 6 and 7 steps to the right. Recall that we are still inside a single row of the dX-bitmap.

Since the union of all sub-intervals spans the whole range of indices of set points, the union of the corresponding bit sequences comprises all distinct bit-sequences in the row around a set bit; that is, all distinct neighborhoods around a point. The number of sub-intervals is equal to the number of thresholds, which is no larger than W. Thus: *a*) the number of distinct layouts is no larger than W, and *b*) the indices of points at the center of each layout fall in a contiguous interval.

When α is irrational dX is a well distributed sequence modulo 1, which means that the average number of thresholds is proportional to W; but in all cases W is a supremum. Also note in Eq. (8) that $\mathrm{dX}(x) = 0$ only when $x = 0$ (otherwise α will be rational); applying this to Eq. (9) reveals that indices are unique. Since thresholds make a well distributed sequence, there would eventually be a threshold (in fact an infinitude of thresholds) between any two indices, indicating a different neighborhood of some size. This proves that the pattern is aperiodic when α is irrational.

IDs: It is easy to convert Eq. (10) into an algorithm for finding the thresholds, as summarized in pseudo-code in Algorithm 1. We sequentially assign numerical IDs to intervals of indices to label them. All points with a given ID have identical neighborhoods. The ID of a point can be retrieved by searching for the interval in which its index falls, which can be implemented efficiently as a binary search. We call this conversion "thresholding", because it effectively searches for the threshold immediately preceding the index. Thus, we have three levels of addressing in AA Patterns:

$$\underbrace{(x,y)}_{\text{coordinates}} \xrightarrow{(3)} \underbrace{(dX,dY)}_{\text{index}} \xrightarrow{\text{thresholding}} \underbrace{(\text{col},\text{row})}_{\text{ID}}. \quad (11)$$

Algorithm 1 Finding thresholds of indices a for a designated neighborhood window width W or a prescribed number of IDs N.

```
i = 0; id_count = 1;
repeat:
    ++i;
    for (x = i) and (x = -i)
        dX = frac(alpha * x / 2);
        if (dX < t)
            addThreshold(t - dX); ++id_count;
        if (dX > 1 - t)
            addThreshold(1 - dX); ++id_count;
until (i == W/2) or (id_count == N);
sort the table of thresholds;
```

Contexts: For a given ID, a context refers to a distinct configuration of IDs of close neighbors around a point with that ID. Whereas points with the same ID have identical locations of neighbors, these neighbors might have different IDs. For the same ID we might therefore have different contexts. To give an example, we inspect the immediate neighbors outside the window of Figure 5. These two locations produce two more thresholds that split two intervals; see Figure 6. IDs of these two intervals can occur in two contexts; that is, points with these IDs have two possibilities for IDs of their immediate neighbors. All other intervals have single contexts: the state of the eighth neighbor is fixed, hence the whole ID window of immediate neighbors, hence their IDs. We may also consider wider contexts than the immediate neighbors, but the number of dual-context IDs grows no larger than the considered width.

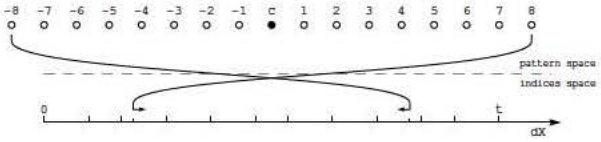


Figure 6: *Inspecting the immediate neighbor location outside the window of Figure 5 reveals two dual-context IDs.*

Close parameters ⇒ similar patterns: Algorithm 1 outputs a similar sequence of thresholds for $\alpha' \approx \alpha$, until α' *fails* to approximate α. It is the order of thresholds that matter in identifying neighborhoods, not their exact values, therefore $\mathrm{AA}(\alpha')$ and $\mathrm{AA}(\alpha)$ share all distinct layouts of points up to the width where the sequence of thresholds of $\mathrm{AA}(\alpha')$ goes out of order. The two patterns are indistinguishable in a window smaller than that width. Note that if the indistinguishable window is larger than a chosen ID window, the two patterns share the same set of IDs; if it is larger by $2n$ they share all n-steps contexts of all IDs.

3.1 Two-Dimensional Neighbourhood Maps

The single row we analyzed implies the whole dX-bitmap, so our conclusions extend to whole vertical slices in the dX-bitmap. Analogous arguments hold for dY, and upon combining the two bitmaps we get a Cartesian product of indices and IDs. Pattern-space neighborhood windows become squares, and index-space intervals become two-dimensional blocks; as illustrated in Figure 7.

Following the distribution of indices, IDs are distributed evenly and quasi-randomly when α is irrational, as illustrated in Figure 8.

As we will show subsequently, IDs can play a similar role to colors or structural indices in tiling methods, with three major advantages:

- IDs can be extracted directly from point coordinates: no need to store production rules.
- The distribution of IDs is inherently quasi-random, eliminating the need to use an external randomization process.
- The number of IDs grows quadratically (linearly in each direction) with the width of identified neighborhood, compared to exponential growth in tiling techniques (cf. [Wachtel et al. 2014]).

Dual-context IDs in the one-dimensional profile produce dual- and quad-context IDs in the two-dimensional Cartesian product. This brings an interesting analogy to color-based tiling methods: In tiling there is a majority of interior (single-context) points, a few edge points (multiple-context), and very few corner points (too many contexts). In AA-Patterns the majority of IDs have single-contexts, only a few have dual-contexts, and very few have quad-

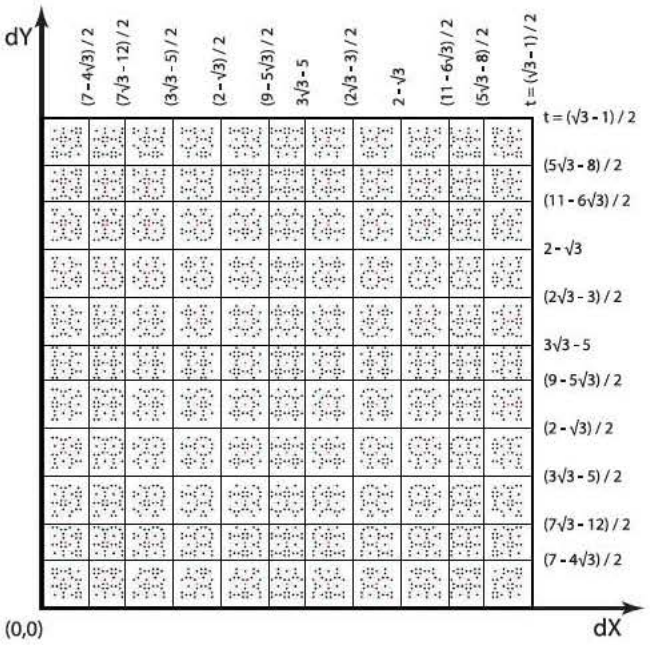


Figure 7: *A Cartesian product of two of the one-dimensional profile in Figure 5 produces a checkered map that fills the set region in index-space. The actual layouts of points are shown inside the corresponding intervals.*

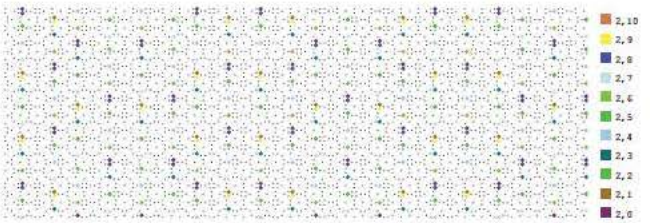


Figure 8: *Distribution of the third column of IDs of Figure 7 in a part of AA* $(\sqrt{3})$.

contexts. Thus, in AA Patterns we logically (not spatially) have "interior points", "edge points", and "corner points", with the important difference that the three classes are interspersed, which reduces seams. This is one reason why this approach surpasses tiling.

4 Optimizing AA Patterns

The above analysis of the basic structure of AA Patterns revealed all the elements needed to build an efficient sampling framework. Now we optimize the point locations for achieving various spectral properties. AA Patterns offer a quite convenient facility for optimization via "similar patterns" discussed in Section 3: choosing a rational parameter p/q close enough to the target irrational α produces a periodic pattern that shares all contexts as $\mathrm{AA}(\alpha)$ in the desired neighborhood width. This setup is an advantage of AA Patterns, as it circumvents the overhead of synchronizing the optimization process across many patches. It also eliminates the difficulty pointed out by Ostromoukhov [2007] of handling non-toroidal domains.

First, we explain the concept using Lloyd's relaxation, then we move on to other optimizations. We applied Lloyd's algorithm to a few periodic AA Patterns, and visualized the incurred displacements in index-space; see Figure 9. A block structure emerges in index-space plots, coinciding with the neighborhood maps described in Section 3, and it is also multi-resolution, revealing more sub-blocks of the map as more iterations are applied; that is, as wider neighborhoods becomes involved in determining displacements. The behaviour of points with the same ID is identical, both within the same pattern and across similar patterns.

Note that this is the natural response of AA Patterns to Lloyd's algorithm: we are not enforcing any behavior. Its implication is that:

1. A small table, one vector per ID, is sufficient to store information about an optimized AA Pattern.
2. The structure of the table is simple, enabling easy retrieval of stored information.
3. Optimization information can be generated/stored to any desirable resolution.
4. Information obtained from a small-period pattern can be reused for a similar long-period (or aperiodic) pattern.

4.1 Optimizing with Other Methods

A general optimizer may optimize points sequentially, one at a time (e.g. CCDT [Xu et al. 2011]), and/or may incorporate a random behavior (e.g. jittering to break local symmetries[3]). Consequently, points with the same ID might be displaced differently. Thus, we have to enforce identical displacements, which we do by averaging as advocated in [Ostromoukhov 2007; Wachtel et al. 2014].

Optimizing a periodic AA Pattern is very similar to optimizing a point set in a toroidal domain; the only constraint is to ensure that points with the same ID make identical displacements. Algorithm 2 summarizes the procedure. The result is stored in a compact table which we call a "displacement map".

In our examples we used $\mathrm{AA}(183/112)$ during optimization, approximating $\alpha = (5 - \sqrt{3})/2$, with ID window width of 99 steps (49 on each side), and 4-steps contexts. This produced 55 single-context and 8 dual-context IDs in each of (dX, dY). Thus, the majority of IDs have single contexts: averaging is sparingly applied, and it has little effect on the optimization process. Figure 1 shows some of the profiles we managed to obtained under the mentioned constraints: save a little anisotropy, the results are almost indistinguishable from the unconstrained optimization. For FPO [Schlömer et al. 2011] we used a very local variant which moves each point to the farthest point within the convex hull of its immediate neighbors (hence preserving contexts).

4.2 Retrieving Optimized Point Sets from Tables

Once a displacement map is available, it can be used with any AA Pattern that shares the same set of IDs and contexts. The local statistics (e.g. nearest neighbor distance, coverage radius, and bond orientation order [Heck et al. 2013]) are maintained; as long as points do not make large displacements beyond the minimum context width. Algorithm 3 lists the steps. It is simple, fast, compact, self-contained, and therefore lends itself well for system-level implementation in any device.

5 Discussion

We now discuss various aspects of our technique, and compare to tiling methods.

[3] Jittering, if needed, should be sized such that it does not take a point "out of context"; that is, a point should not jump behind immediate neighbors.

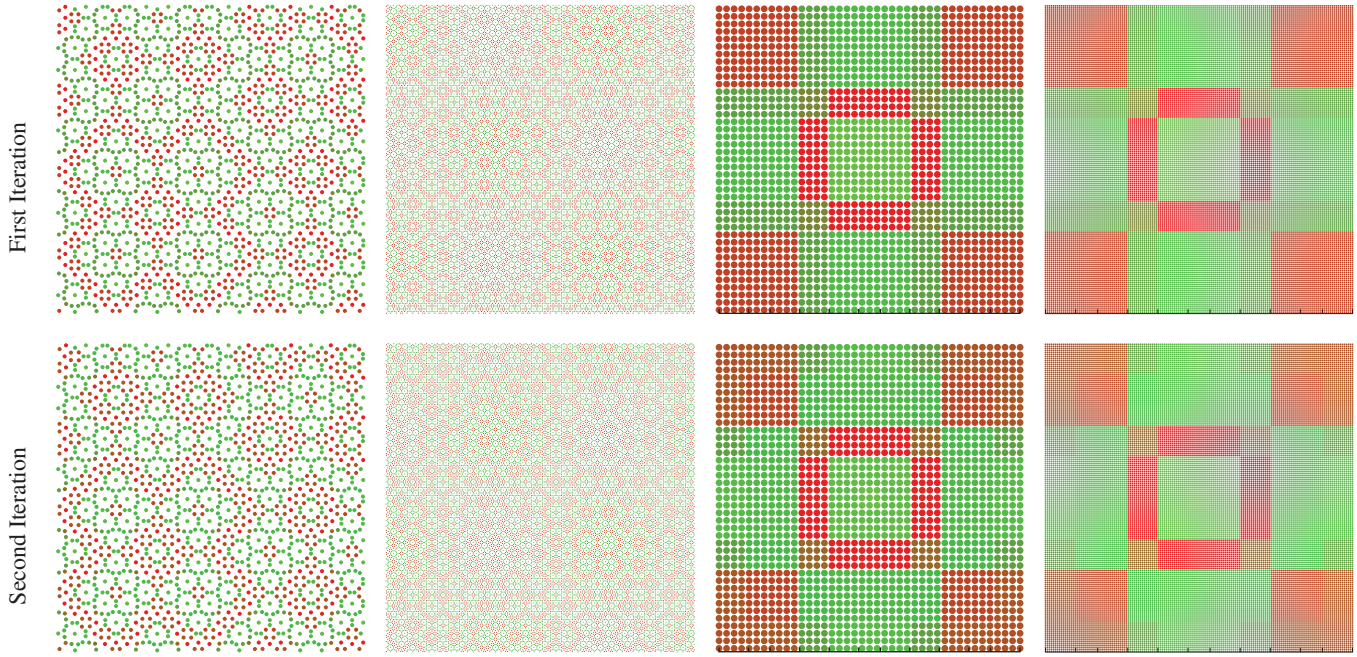


Figure 9: *Visualization of two iterations of Lloyd's algorithm applied to a single period of AA*$(97/56)$ *and AA*$(362/209)$*, where* $97/56 \approx 362/209 \approx \sqrt{3}$*. (a, b), respectively, show the actual displaced point set, where each point is colored according to the size of incurred displacement; in a smooth scale from green (0) to red (largest displacement). In (c, d), respectively, we rearranged the points by order of their indices; in other words, we plotted the sizes of displacements in index-space. For comparison, the ticks below each index-space plot show the thresholds of Figure 5 at the right scale.*

Algorithm 2 Generating an optimized displacement map for a periodic pattern AA(p/q).

Input:
1. A $t \times t$ map assigning a set of $N \times N$ IDs to points.
2. An iterative optimizer of a local nature.

Steps:
1. Use Eq. (2) to generate a single period of $AA(p/q)$;
2. Repeat:
   - use the optimizer to find the necessary displacement for each point;
   - average the displacements across points sharing the same ID;
   - apply the average displacements to points;

   until the point set has the desired properties;
3. Record the displacements in an $N \times N$ map, one vector per ID

Algorithm 3 Generating an optimized point set from a displacement map using a parameter α.

Input:
1. A table **T** of N thresholds generated by Algorithm 1.
2. An $N \times N$ displacement map **D** generated by Algorithm 2.

Pseudo Code:

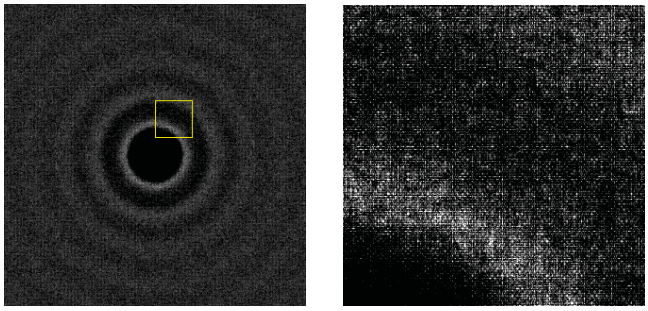
```
For each grid point (x, y)
    dX = frac((alpha * x + y)/2)
    dY = frac((alpha * y + x)/2)
    if (dX < t) and (dY < t) {
        row = binarySearch(T, dY)
        col = binarySearch(T, dX)
        output (x, y) + D[row, col]
    }
```

Speed: In a 2.50GHz CORE i5 CPU our production algorithm produces more than 100M points per second; this is almost 20 times faster than [Wachtel et al. 2014], and is faster than any tiling method we are aware of. Our optimization algorithm is also quite fast thanks to the periodic approximation patterns. We used a point set of 5041 points during optimization, and it takes from a few seconds for Lloyd's relaxation, up to 3 minutes using the method of Heck et al. [2013]. The latter can be considered an upper limit, since this algorithm is able to match most common profiles.

Memory: Due to the limited size of the displacement map (a few kilobytes), the memory footprint of our method is orders of magnitude smaller than Wachtel et al. [2014] or other tile-based methods. Note that the speed and memory usage of our production algorithm are independent of the profile of the point set. We also need a small memory footprint during optimization, since our method optimizes a single point set over a toroidal domain, eliminating the overheads of synchronizing across many patches.

Quality: Compared to tiling methods, our approach puts less constraints on the optimization algorithms, and is able to consider arbitrarily large neighborhoods and contexts. To the best of our knowledge, none of the tiling methods can index beyond the second ring of neighbors and stay within the practical limits of resources; cf. [Wachtel et al. 2014, supplementary material]. Our optimization is therefore more accurate.

The spectral quality of all known lookup methods deteriorates gradually as the size of the point set grows [Lagae and Dutré 2008]. Specifically, a grid-like structure appears in the power spectrum, reflecting the fact that this spectrum is only a quantized approxi-



(a) (b)

Figure 10: *(a) Power spectrum of a 64k blue-noise point-set generated with Algorithm 3 using a 5k table. (b) A closeup showing the quantized nature of the spectrum.*

mation of the target spectrum; see Figure 10. The intuition is that lookup methods (including ours) displace points from a deterministic structure using a finite set of vectors, and the latter has a limited capacity to conceal the underlying structure as the point set grows. Considering this, our approach offers two important advantages over tiling methods: *a*) it can generate very faithful displacement maps (as discussed above) which retain their quality for higher ratios of point set size to map size, and *b*) thanks to the large selection set of α, it enables generating tables of exactly any desirable size, to match the available budget of memory and/or support a specific target neighborhood size.

Flexibility: The fact that we do not need much resources makes our approach suitable for many implementation scenarios unattainable by tiling methods. For example: *a*) it is ready for client-server implementation, where a server can keep thousands of reusable displacement tables of different noise profiles, *b*) it enables implementation of on-the-fly target-matching at interactive speeds, and *c*) it enables storing extra information in the neighborhood tables, tabular or functional.

6 Conclusion

In this paper we presented a new technique for producing large point sets that are optimized to match a wide range of spacial and/or spectral profiles.

The method runs at high speed and uses very little resources. It is free of the seams problem found in the color-based tiling approaches, and it solves the problem of large memory consumption for geometry-based methods by generating all structural information from coordinates of points in a grid.

Our technique is based on AA Patterns, ornamental patterns originally introduced for artistic purposes. We reintroduce these patterns as quasi-random point sets with powerful neighborhood identification mechanism through their maps of indices.

The adaptation of AA Patterns for sampling suggests two directions for future research: one is to investigate other properties of AA Patterns as quasi-random point sets (e.g. their incremental quality), and the other is to inspect optimizing other quasi-random point sets (e.g. Halton sequences).

Even though our approach offers some advantages, it is currently limited to fixed densities of points. In principle, AA Patterns are hierarchical and self-similar [Ahmed 2012], so there is a good potential for extending the concept to adaptive sampling. However, we so far faced two obstacles: their self-similarity is topological, not geometrical, and details appear upon zooming out, not upon zooming in, thus we currently do not have an infinite possibility for zooming-in. We leave this for further research. Other directions for future research include extending the concept to multi-class sampling and higher dimensions.

Acknowledgements

We thank the anonymous reviewers for their great help in shaping this paper. Thanks to Mohamed Sayed for his discussions during an early stage of the idea. This work was supported in part by the Deutsche Forschungsgemeinschaft Grant DE-620/22-1, Foreign 1000 Talent Plan (WQ201344000169), NSFC (61522213, 61379090), 973 Program (2014CB360503), Guangdong Science and Technology Program (2015A030312015, 2014B050502009), Shenzhen VisuCA Key Lab (CXB201104220029A).

References

AHMED, A. G. M. 2011. AA Patterns. In *EG 2011 - Posters*, Eurographics Association, Llandudno, UK, 35–36. Accessible online at http://diglib.eg.org/EG/DL/conf/EG2011/posters/035-036.pdf.

AHMED, A. G. M. 2011. Mathematical Hints for Parameter Selection for AA Patterns. In *Bridges Coimbra*, The Bridges Organization, 271–278.

AHMED, A. G. M. 2011. Pixel Patterns from Quantization Artifacts of Forward Affine Mapping. *Journal of Graphics, GPU, and Game Tools 15*, 2, 73–94.

AHMED, A. G. M. 2012. On the Fractal Behaviour of AA Patterns. In *Theory and Practice of Computer Graphics*, Eurographics Association, Rutherford, United Kingdom, Hamish Carr and Silvester Czanner, Ed., 93–97.

BALZER, M., SCHLÖMER, T., AND DEUSSEN, O. 2009. Capacity-constrained point distributions: A variant of lloyd's method. In *ACM SIGGRAPH 2009 Papers*, ACM, New York, NY, USA, SIGGRAPH '09, 86:1–86:8.

CHEN, Z., YUAN, Z., CHOI, Y.-K., LIU, L., AND WANG, W. 2012. Variational blue noise sampling. *IEEE Transactions on Visualization and Computer Graphics 18*, 10 (Oct.), 1784–1796.

COHEN, M. F., SHADE, J., HILLER, S., AND DEUSSEN, O. 2003. Wang tiles for image and texture generation. In *ACM SIGGRAPH 2003 Papers*, ACM, New York, NY, USA, SIGGRAPH '03, 287–294.

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

DE GOES, F., BREEDEN, K., OSTROMOUKHOV, V., AND DESBRUN, M. 2012. Blue noise through optimal transport. *ACM Trans. Graph. 31*, 6 (Nov.), 171:1–171:11.

DIPPÉ, M. A. Z., AND WOLD, E. H. 1985. Antialiasing through stochastic sampling. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '85, 69–78.

FATTAL, R. 2011. Blue-noise point sampling using kernel density model. In *ACM SIGGRAPH 2011 Papers*, ACM, New York, NY, USA, SIGGRAPH '11, 48:1–48:12.

HECK, D., SCHLÖMER, T., AND DEUSSEN, O. 2013. Blue noise sampling with controlled aliasing. *ACM Trans. Graph. 32*, 3 (July), 25:1–25:12.

KOPF, J., COHEN-OR, D., DEUSSEN, O., AND LISCHINSKI, D. 2006. Recursive wang tiles for real-time blue noise. *ACM Trans. Graph. 25*, 3 (July), 509–518.

KUIPERS, L., AND NIEDERREITER, H. 1974. *Uniform distribution of sequences*. Pure and applied mathematics. John Wiley & Sons, New York. A Wiley-Interscience publication.

LAGAE, A., AND DUTRÉ, P. 2006. An alternative for wang tiles: Colored edges versus colored corners. *ACM Trans. Graph. 25*, 4 (Oct.), 1442–1459.

LAGAE, A., AND DUTRÉ, P. 2008. A comparison of methods for generating poisson disk distributions. In *Computer Graphics Forum*, vol. 27, Wiley Online Library, 114–129.

MCCOOL, M., AND FIUME, E. 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.

MITCHELL, D. P. 1987. Generating antialiased images at low sampling densities. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '87, 65–72.

NIEDERREITER, H. 1992. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM.

OSTROMOUKHOV, V., DONOHUE, C., AND JODOIN, P.-M. 2004. Fast hierarchical importance sampling with blue noise properties. In *ACM SIGGRAPH 2004 Papers*, ACM, New York, NY, USA, SIGGRAPH '04, 488–495.

OSTROMOUKHOV, V. 2007. Sampling with polyominoes. *ACM Trans. Graph. 26*, 3 (July).

ÖZTIRELI, A. C., AND GROSS, M. 2012. Analysis and synthesis of point distributions based on pair correlation. *ACM Trans. Graph. 31*, 6 (Nov.), 170:1–170:10.

SCHLÖMER, T., AND DEUSSEN, O. 2010. Semi-stochastic tilings for example-based texture synthesis. In *Proceedings of the 21st Eurographics Conference on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, EGSR'10, 1431–1439.

SCHLÖMER, T., HECK, D., AND DEUSSEN, O. 2011. Farthest-point optimized point sets with maximized minimum distance. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, ACM, New York, NY, USA, HPG '11, 135–142.

SCHMALTZ, C., GWOSDEK, P., BRUHN, A., AND WEICKERT, J. 2010. Electrostatic halftoning. *Computer Graphics Forum 29*, 8, 2313–2327.

ULICHNEY, R. 1988. Dithering with blue noise. *Proceedings of the IEEE 76*, 1 (Jan), 56–79.

WACHTEL, F., PILLEBOUE, A., COEURJOLLY, D., BREEDEN, K., SINGH, G., CATHELIN, G., DE GOES, F., DESBRUN, M., AND OSTROMOUKHOV, V. 2014. Fast tile-based adaptive sampling with user-specified fourier spectra. *ACM Trans. Graph. 33*, 4 (July), 56:1–56:11.

XU, Y., LIU, L., GOTSMAN, C., AND GORTLER, S. J. 2011. Capacity-constrained delaunay triangulation for point distributions. *Computers & Graphics 35*, 3, 510 – 516. Shape Modeling International (SMI) Conference 2011.

ZHOU, Y., HUANG, H., WEI, L.-Y., AND WANG, R. 2012. Point sampling with general noise spectrum. *ACM Trans. Graph. 31*, 4 (July), 76:1–76:11.

A Glossary

Here we summarize the terms that appeared in our discussion.

Cluster: The visually distinguishable grouping of points in an AA Pattern. We do not use clusters directly in our proposed method.

Neighborhood: Locations of other points near a given point.

Window: A square area (or a range in 1D) of a given size centered at a point.

Layout: Locations of points in a window.

Neighborhood Window: A layout with a specific point at the center.

Index: A pair of fractional hash keys obtained from the coordinates of a point using Eq. (3) (or Eq. (8) in one dimension).

Set-Region: The interval of indices of points included in an AA Pattern.

ID: A label to identify a subset of points which share the same neighborhood window. Indices of such points fall in a contiguous rectangular block.

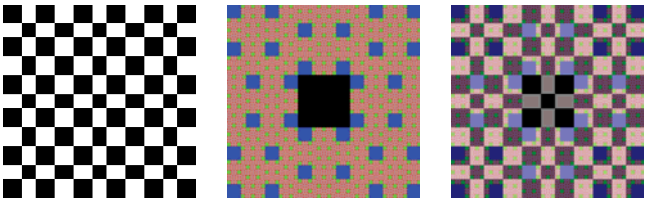
Threshold: An index value marking the edge between blocks of different IDs.

Thresholding: Converting an index to an ID (e.g. using binary search).

Neighborhood Map: The union of all ID blocks in index-space, which exactly fills the set-region.

B The Big Picture

Figure 8 shows the quasi-random local distribution of IDs. Global distribution can be visualized in index-space by comparing neighborhood maps to coloring maps; each block in the latter comprises indices of a specific point in a distinct cluster. From Figure 11(c) we see that Each ID is held by either a subset of a specific point in a distinct cluster, or a union of specific points from different clusters. The distribution of each cluster resembles an AA Pattern, but subsetting and merging obscures this ornamental distribution. Thus, the global distribution of IDs is visually quasi-random, in addition to being quasi-random in a mathematical sense.



(a) (b) (c)

Figure 11: *Set-region maps of $AA(\sqrt{3})$: (a) The neighborhood map from Figure 7. (b) The coloring map for Figure 3(b). (c) The superposition of the two maps: the neighborhood map tends to align with the blue blocks because our chosen neighborhood window is the same order of size as the blue clusters in Figure 3(a).*