Non-Periodic Corner Tilings in Computer Graphics

Dissertation

zur Erlangung des akademischen Grades des

Doktors der Naturwissenschaften (Dr. rer. nat.)

vorgelegt von

THOMAS SCHLÖMER

Universität Konstanz

Mathematisch-Naturwissenschaftliche Sektion

Fachbereich Informatik und Informationswissenschaft

Tag der mündlichen Prüfung: 16.11.2012

Referent: Prof. Dr. Oliver Deussen

Referent: Prof. Dr. Ulrik Brandes

Advisor
Prof. Dr. Oliver Deussen, University of Konstanz

Reviewers
Prof. Dr. Oliver Deussen, University of Konstanz
Prof. Dr. Ulrik Brandes, University of Konstanz

Date of Submission
July 12th, 2012

Date of Defense
November 16th, 2012

Abstract

Rendering computer-generated images is both memory and runtime intensive. This is particularly true in realtime computer graphics where large amounts of content have to be produced very quickly and from limited data. Tile-based methods offer a solution to this problem by generating large portions of a specific content out of a much smaller data set of tiles.

This dissertation investigates the use of corner tiles for this purpose—unit square tiles with color-coded corners. They tile the plane by placing them without gaps or overlaps such that tiles have matching corner colors. We present efficient algorithms to perform such a tiling that are both more flexible and less prone to artifacts than existing algorithms. We also present solutions to combinatorial problems that arise when using corner tiles, and introduce high-quality methods to perform the tile-based generation of two fundamental components of any rendering system: textures and two-dimensional sample point sets.

The results of this dissertation are advantageous for both realtime and offline rendering systems where they improve state-of-the-art results in texture synthesis, image plane sampling, and lighting computations based on numerical integration.

Zusammenfassung

Die Berechnung computergenerierter Bilder ist sehr speicher- und laufzeitintensiv. Besonders kritisch ist dies in der Echtzeit-Computergraphik, in der große Inhaltsmengen sehr schnell und auf einer limitieren Datenbasis generiert werden müssen. Kachelbasierte Methoden bieten für diese Herausforderung eine mögliche Lösung an: Ein bestimmter Inhalt wird dynamisch anhand einer kleinen Menge von speziell präparierten Kacheln generiert.

Diese Dissertation untersucht in diesem Kontext die Verwendung von gefärbten Eckkacheln. Eckkacheln sind quadratische Kacheln, die mit farbkodierten Ecken versehen sind. Sie erlauben eine Parkettierung der Ebene, indem man sie so anordnet, dass benachbarte Kacheln übereinstimmende Eckfarben haben. In dieser Arbeit präsentieren wir Algorithmen, die eine solche Parkettierung effizient ermöglichen und dabei gleichzeitig flexibler sind und zu weniger Artefakten führen als existierende Algorithmen. Wir geben darüber hinaus Lösungen für kombinatorische Probleme an, die bei der Verwendung von Eckkacheln entstehen. Von praktischer Konsequenz sind außerdem Methoden, die zwei der wesentlichen Komponenten eines jeden Renderingsystems in hoher Qualität generieren können: Texturen und zweidimensionale Punktmengen.

Die Ergebnisse dieser Dissertation können sowohl für die Echtzeit-Computergraphik als auch für Offline-Systeme von Vorteil sein: In der Textursynthese, der Abtastung der Bildebene und der Beleuchtungsberechnung anhand numerischer Integration verbessern die hier vorgestellten Verfahren bisherige Methoden.

Acknowledgements

This thesis was made possible by several people whom I wish to thank very dearly.

First, I like to thank my advisor, Prof. Dr. Oliver Deussen, for inviting me to his group and giving me the opportunity to pursue this thesis under favorable working conditions. I also thank my second advisor, Prof. Dr. Ulrik Brandes, for reviewing this thesis and for providing valuable comments.

Many thanks go to my colleagues at the Konstanz computer graphics group, in particular Daniel Heck, Michael Balzer, Boris Neubert, and Sören Pirk who were as interested in discussing graphics topics as they were in discussing StarCraft II build orders. I also like to thank my external collaborators: Steffen Frey, Sebastian Grottel, and Carsten Dachsbacher. Special thanks go to Alexander Keller and the former mental images GmbH for providing early access to some of their research results. This thesis was also supported by the German Research Foundation, DFG, via the project DE 620/14-1 and the PhD Program "Explorative Analysis and Visualization of Large Information Spaces".

Very personal thanks go to my family and to Helena Stehle. Their support remains invisible in this thesis, yet still is the most important.

Contents

1

Introduction

Creating rich and complex content is one of the major problems in computer graphics, especially in interactive applications where large amounts of content have to be produced very quickly and from limited data. Textures, for example, play a significant role in the creation of believable images but must typically be provided in the form of memory intensive images due to their visual complexity. Other fundamental tasks, such as the generation of sample points, can be computationally expensive so that the best possible points may not become beneficial unless they can be generated more quickly.

Tile-based methods offer a solution to both memory and runtime problems by synthesizing large amounts of content out of a much smaller data set of tiles. If the tiles can be aptly filled with portions of the desired signal in a preprocessing stage, only an arrangement of these tiles—a tiling—has to be generated online. Compared to the traditional direct approach, this may improve either the generation speed or the amount of necessary memory by several orders of magnitude. Figure 1.1 illustrates the principal idea of tile-based methods using the example of texture synthesis.

There are two main aspects that make such a tile-based approach challenging. First, the tiling must usually be non-periodic in order to hide the unavoidable repetition of content: If we would just repeat the same tile over and over, the tiled result will likely show unsatisfactory artifacts. Second, the tile interiors must be constructed very carefully because their content must seamlessly fit between adjacent tiles for every possible tiling and without reducing the quality of the original signal.

1.1 History of Tilings

The history of tilings dates back to early examples in decorative art and ornamental design. In the islamic culture, for instance, small tile-like pieces where used to decorate large parts of important monuments and mosques (see Figure 1.2(a)). In the occident, tilings became prominent in mid-20th century thanks

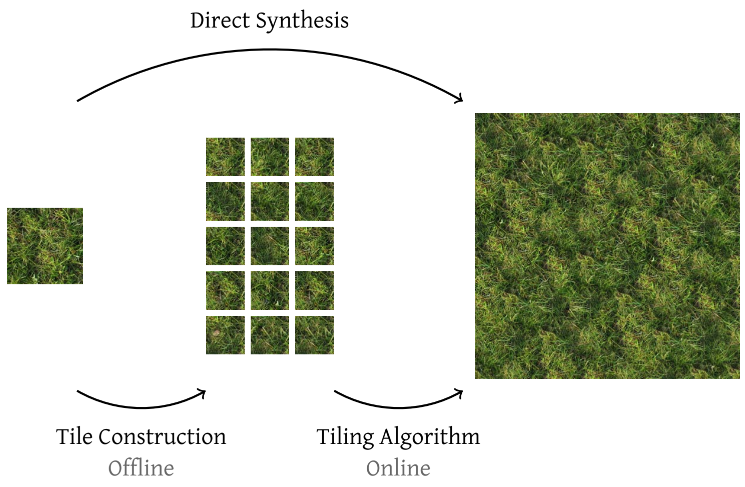


Figure 1.1: Principle of tile-based methods using the example of textures synthesis. A set of tiles is constructed in a preprocessing step such that only a valid arrangement of these tiles has to be generated online.

to the graphic work of Maurits C. Escher whose tilings are often interesting because they contain variations of a single, complex tile only (see Figure 1.2(b)). Similar tiling phenomena also appear in nature: The basalt columns at the Giant's Causeway, for example, appear in the form of hexagons that cover large parts of this coast in Northern Ireland (see Figure 1.2(c)).

The idea of tilings in a strict mathematical sense can be traced back to Johannes Kepler who documented regular tilings of the plane in the early 17th century [Kep19]. But the in-depth mathematical study of tilings did not really start until the late 19th century with the works of Yevgraf Fyodorov [Fyo91] and Arthur Schoenflies [Sch91], who were the first to fully enumerate all distinct symmetry groups in 2D and 3D. In the 1960's and 1970's, tiles that can never induce periodic tilings received increasing attention due to the works of Wáng Hào and Roger Penrose, both of which came up with sets of tiles that now bear their names [Wán61, Pen74]. These tiles and the corresponding tilings are called aperiodic and are of mathematical interest till this day [Cul96]. We will take a closer look at these tilings and their theoretical properties in the following chapter. Overall, however, this thesis will only briefly touch tiling theory because of its limited applicability to computer graphics problems. A good introduction into these aspects is the book by Grünbaum and Shephard [GS86].

1.2 Scope of the Thesis

This thesis concentrates on a specific class of tiles called *corner tiles*. Corner tiles are unit square tiles with colored-coded corners and were developed concurrently by Ng et al. [NWT*05] and Lagae and Dutré [LD06a]. They tile the plane by

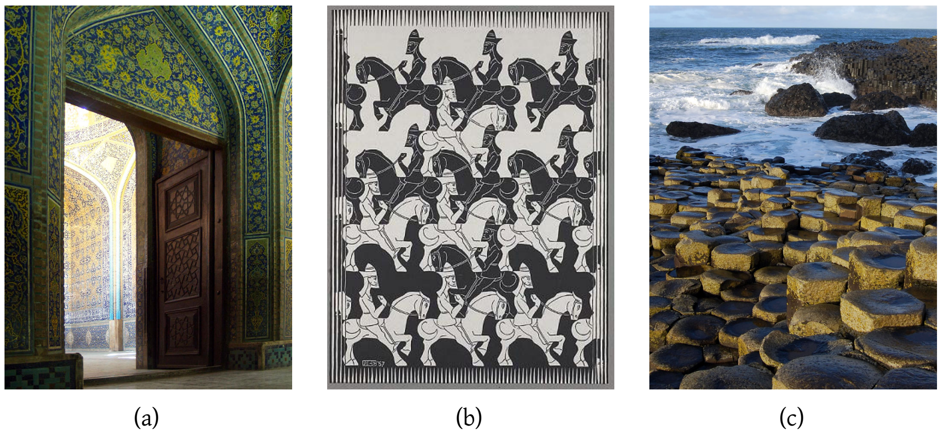


Figure 1.2: Tilings appearing in art and nature. (a) Sheikh Lotf Allah Mosque, Isfahan, Iran, 1603–1618. [Man08] (b) Regular Division of the Plane III, M. C. Escher, 1957–1958. [Esc58] (c) Basalt columns at the Giant's Causeway in Northern Ireland. [Bro10]

placing them without gaps or overlaps such that neighboring tiles have matching corner colors. The tiles are not allowed to be rotated or reflected. Due to their simplicity and flexibility, they have quickly proven to be among the most useful tiles for two-dimensional computer graphics problems [LKF*08].

This thesis presents advances with respect to several aspects of tilings based on corner tiles. The first aspect are tiling algorithms, i.e., algorithms to distribute corner tiles in a way that is both efficient and less prone to artifacts than current algorithms. In particular, we present a novel deterministic tiling algorithm that yields a more uniform distribution of corner colors than current stochastic algorithms. We demonstrate that the new algorithm is particularly useful for tile-based texture synthesis.

The second aspect is the tile construction process. Here we concentrate on two-dimensional sample point sets, which have numerous fields of application within graphics such as sampling, numerical integration, or object distribution. We present two optimization techniques for such point sets, each tailored for one important characteristic of the points. The first technique distributes points such that the volume of their Voronoi cells are of equal size which is advantageous for low-dimensional numerical integration problems. The second technique distributes points such that their mutual distances are maximized which is advantageous for sampling problems such as image plane sampling. Both techniques improve the state of the art in their respective fields and become efficient in a corner tile setting.

The last aspect are theoretical insights into corner tilings. We present solutions to the so-called tile packing problem which seeks a toroidal arrangement of all tiles in a given tile set where each tile is used exactly once. We also introduce a new class of tile sets which contain only those tiles that are needed for a specific range of tilings. In an application scenario like tile-based texture synthesis, these tile sets may save a significant amount of memory.

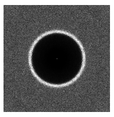
The insights and results provided by this thesis continue to improve the efficiency and applicability of corner tile-based methods in computer graphics and help in the ongoing effort to deal with and further increase the complexity of computer generated images. Figure 1.3 gives a sneak peek of some of the results of this thesis.

1.3 Summary of Contributions

The main contributions of this thesis are:

- A deterministic tiling algorithm for fast and improved online corner tilings. This is the first algorithm to consider a non-random distribution of corner colors and yields improvements in tile-based texture synthesis.
- A generalization of stochastic tilings where the probability for each corner color can be locally adjusted. The resulting semi-stochastic tilings extend tile-based texture synthesis to globally varying textures.
- A method to prune tile sets based on neighbor information of corner colors and a closed-form expression for the size of these tile sets. This allows the construction of minimal tile sets tailored to a specific range of tilings.
- Solutions to the tile packing problem for compact corner tile sets over three and four colors. So far, only a solution over two colors was known.
- A tile construction algorithm for spatially uniform point distributions based on capacity-constrained Voronoi tessellations. The resulting point sets yield improvements in several numerical integration problems occurring in physically-based rendering.
- A tile construction algorithm for spatially uniform point distributions using a new distance-based optimization method. The resulting point sets show significantly higher mutual distances than previous methods and yield improvements in image plane sampling.
- Three algorithms that add certain properties to any point set as a post process. We utilize these algorithms during practical applications of our tiled point sets.

These contributions have partly been published in the following publications:



D. Heck, T. Schlömer, O. Deussen. Blue Noise Sampling with Controlled Aliasing. Conditionally accepted to *ACM Transactions on Graphics*, 2012. [HSD12b]

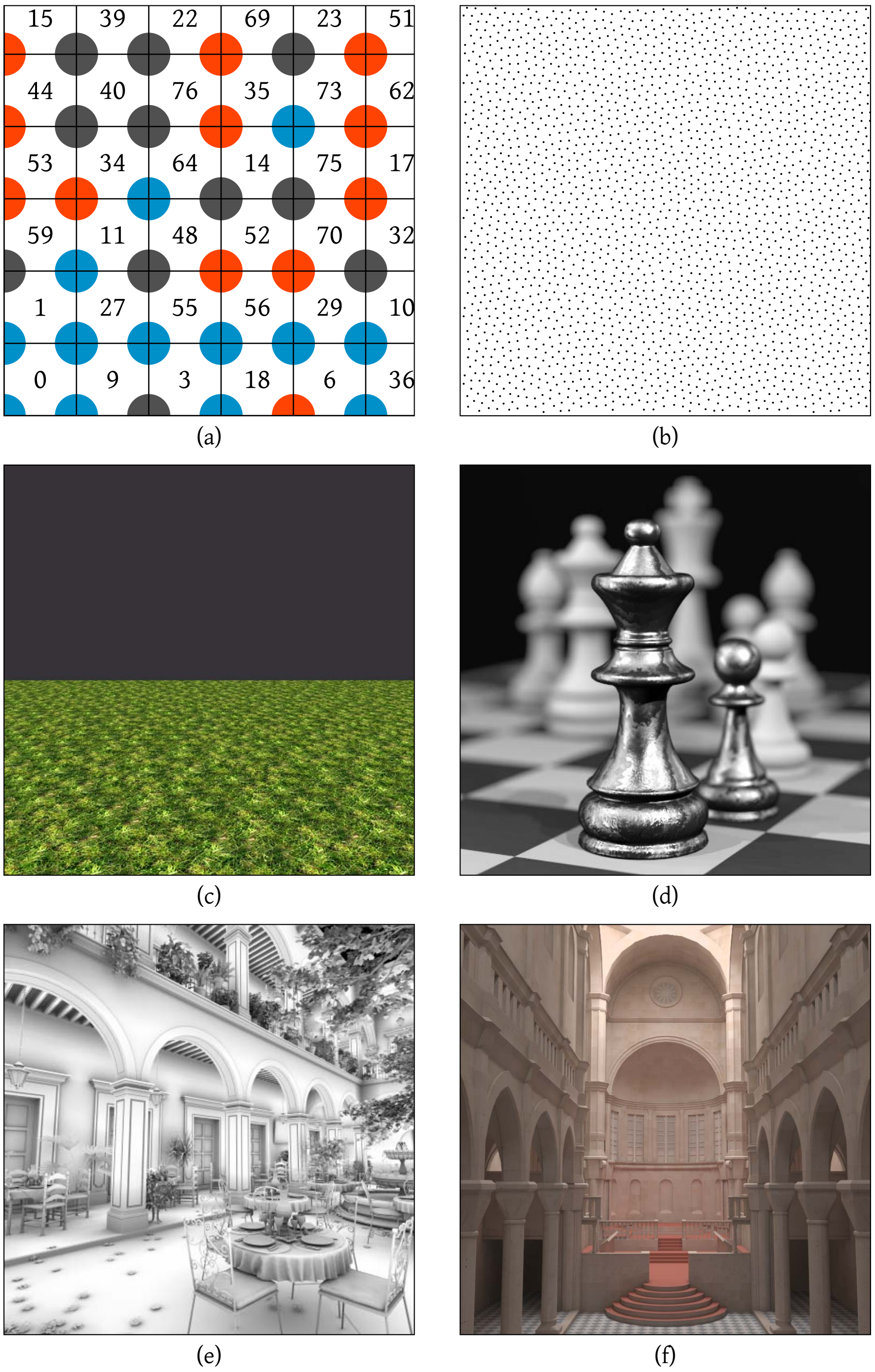
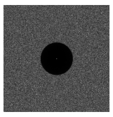
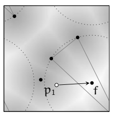


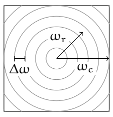
Figure 1.3: Some of the results presented in this thesis. (a) Packing solutions for compact corner tile sets. Corner tile-based generation of optimized point sets (b) and large non-periodic textures (c). Applications for the point sets include depth-of-field (d), ambient occlusion (e), and global illumination (f).



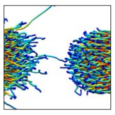
D. Heck, T. Schlömer, O. Deussen. Aliasing-Free Blue Noise Sampling. Technical Report, University of Konstanz, 2012. [HSD12a]



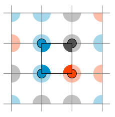
T. Schlömer, D. Heck, O. Deussen. Farthest-Point Optimized Point Sets with Maximized Minimum Distance. *Proc. High Performance Graphics*, 135–142, 2011. [SHD11]



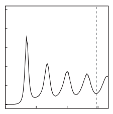
T. Schlömer, O. Deussen. Accurate Spectral Analysis of Two-Dimensional Point Sets. *Journal of Graphics, GPU, and Game Tools*, 15(3):152–160, 2011. [SD11]



S. Frey, T. Schlömer, S. Grottel, C. Dachsbacher, O. Deussen, T. Ertl. Loose Capacity-Constrained Representatives for the Qualitative Visual Analysis in Molecular Dynamics. *Proc. IEEE Pacific Visualization Symposium*, 51–58, 2011. [FSG*11]



T. Schlömer, O. Deussen. Semi-Stochastic Tilings for Example-Based Texture Synthesis. *Computer Graphics Forum (Proc. Eurographics Symposium on Rendering)*, 29(4):1431–1439, 2010. [SD10a]



T. Schlömer, O. Deussen. Towards a Standardized Spectral Analysis of Point Sets with Applications in Graphics. *Technical Report, University of Konstanz*, 2010. [SD10b]

M. Balzer, T. Schlömer, O. Deussen. Capacity-Constrained Point Distributions: A Variant of Lloyd's Method. *ACM Transactions on Graphics (Proc. SIGGRAPH 2009)*, 28(3):86:1–8, 2009. [BSD09]

1.4 Structure of the Thesis

This thesis is organized as follows. In the next chapter we define non-periodic tilings and corner tiles in particular. We view them in regard to other tilings in computer graphics and present algorithms that efficiently tile the plane using corner tiles. The following chapters are then dedicated to the construction of the tile interiors: textures in Chapter 3 and sample point sets in Chapter 4. We evaluate each of the methods presented and demonstrate how they improve prominent applications of corner tilings in the fields of texture synthesis, image plane sampling, and numerical integration. For tiled point sets, we demonstrate this in a dedicated Chapter 5. The thesis concludes with a summary and outline for future work in Chapter 6.

2

Non-Periodic Corner Tilings

We are interested in corner tilings of the plane that do not repeat themselves periodically. Before we specify corner tiles in detail, we briefly define basic nomenclature and recapitulate other types of tiles that have found application in computer graphics. This allows us to view corner tiles in context and will give a clearer picture of their advantages. We then discuss several algorithms that efficiently generate tilings of the plane using corner tiles.

2.1 Definitions

In the following, we define the most important terms used in this thesis.

1. A *tiling* is an arrangement of plane figures that fills the plane without gaps or overlaps.
2. Each plane figure is called a *tile*.
3. The set of of plane figures that may be used in the tiling is called the *tile set*.
4. *To tile* means to cover the plane with copies of tiles in the tile set, i.e., to generate a tiling.
5. A tiling is called *periodic* if a translation exists that maps the tiling to itself. Otherwise it is called *non-periodic*.
6. If a tile set does not admit periodic tilings, both the tile set and a tiling using this tile set are called *aperiodic*.

The mathematical literature sometimes prefers the term *tessellation* over *tiling*. But since *tessellation* usually describes the discretization of a free-form surface in a computer graphics context (e.g., by generating a triangle mesh), this thesis exclusively uses the term *tiling*.

The notion of periodicity implies infinite tilings. The tilings used in graphics, however, are always finite. Nevertheless, we are still interested in the question if there ever can or cannot be a systematic repetition of a portion of the tiling.

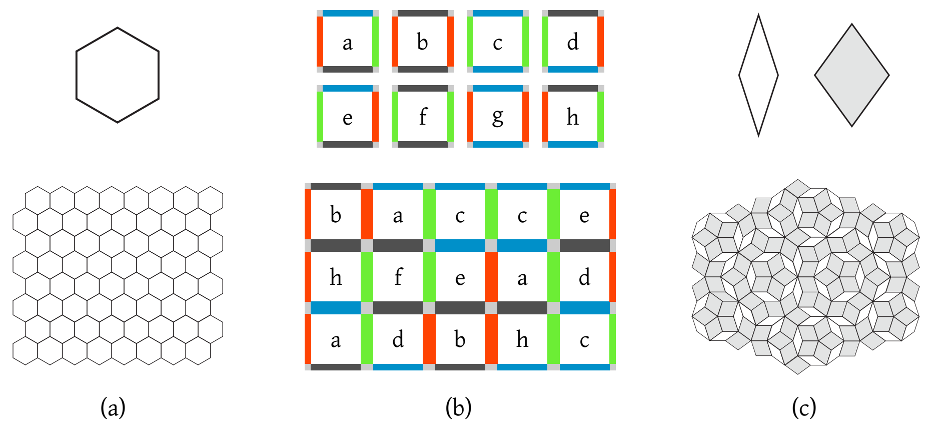


Figure 2.1: Tilings in logic and mathematics. (a) Periodic hexagonal tiling. (b) Non-periodic Wang tiling. [Wán61] (c) Aperiodic Penrose tiling. [Pen78]

Also note the difference between the terms *non-periodic* and *aperiodic*. A given tile set may not be aperiodic (i.e. it may lead to periodic tilings) but—with a suitable tiling algorithm—may nevertheless only yield non-periodic tilings. This is connected to the observation that, in graphics, the visual appearance of a tiling is sometimes more important than its theoretical properties.

Figure 2.1 shows an example for each type of periodicity. Figure 2.1(a) depicts a periodic tiling based on a single hexagonal tile that periodically tiles the plane. In Figure 2.1(b), on the other hand, we see a cutout from a tiling based on a set of eight Wang tiles that is non-periodic. The tile set itself, however, is not an aperiodic tile set because it also admits periodic tilings (for example, by exclusively using tile b). The two rhombs by Roger Penrose in Figure 2.1(c) are a prominent example for an aperiodic tile set: No matter how we tile the plane using these tiles, strict periodicity will never occur.

2.2 Tilings in Computer Graphics

In computer graphics, methods based on tilings date back to early days where tiles appeared in the form of sprites in hardware-accelerated 2D graphics [LB09]. Although strictly periodic, they already fulfilled an intent similar to the non-periodic corner tilings we are interested in, i.e., to save memory and give the appearance of being a much larger portion of the same content.

Non-periodic tilings were not introduced to computer graphics until 1997 when Jos Stam [Sta97] utilized a set of Wang tiles [Wán61] for texture synthesis. Wang tiles are square tiles with color-coded edges that must have matching edge colors in order to form a valid tiling (see Figure 2.1(b)). They can be considered the predecessor of corner tiles and will be discussed in more detail when we introduce corner tiles.

Other non-periodic tilings used in graphics include Penrose rhombs [Pen78] and polyominoes [Gol65]. They were mainly used for generating spatially uniform point sets [ODJ04, Ost07], which has also become a prominent application scenario for Wang tiles [SCM00, HDK01, CSHD03, KCODL06]. We will take a closer look at these methods when we discuss our own approach in Chapter 4. On the other side of the spectrum Craig S. Kaplan investigated the computer-aided generation of tilings for geometric art and ornament [Kap02]. A comprehensive overview over tile-based methods in graphics is given by Lagae et al. [LKF*08].

Some of these tiling concepts extend naturally to higher dimensions. Lu et al. [LEQ*07] and Peytavie et al. [PGGM09], for instance, use three dimensional Wang or corner cubes for the synthesis of complex volume data while Lagae and Dutré [LD06c] use corner cubes for the synthesis of Poisson-sphere distributions. Most results of this thesis do generalize to higher dimensions, too, but overall the scope of applications for three or higher dimensional tilings is limited. Tile sets typically grow exponentially with dimension and constructing tile interiors in e.g. three dimensions is a lot more difficult than in two dimensions. This thesis solely concentrates on the 2D case where the application of a tile-based method is most beneficial [LKF*08].

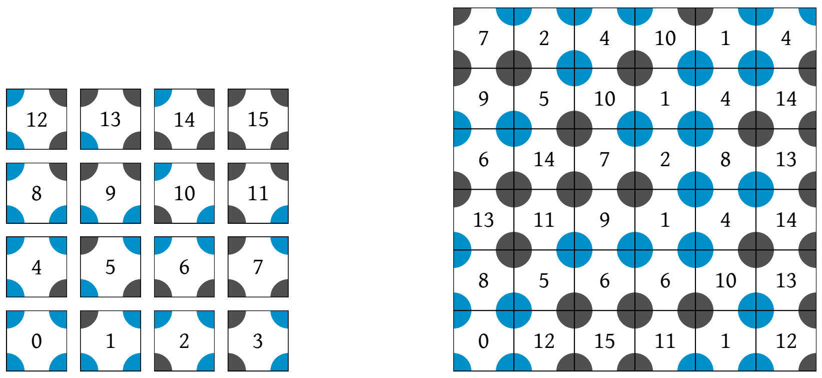
2.3 Corner Tiles

Corner tiles are unit-square tiles with color-coded corners. They tile the plane by placing them without gaps or overlaps such that neighboring tiles have matching corner colors. If we allow C colors for the corners, there can be at most C^4 distinct tiles because the tiles are not allowed to be rotated or reflected. We will see in Chapter 3 that this is a desirable property for typical computer graphics problems.

Corner tiles were developed concurrently by Ng et al. [NWT*05] and Lagae and Dutré [LD06a] as an alternative to Wang tiles. Corner tiles address the corner problem of Wang tiles: Due to their edge coloring, Wang tiles only guarantee continuity of a tile's content with respect to its horizontal and vertical but not its diagonal neighbors. This often leads to continuity artifacts and is the main reason why corner tiles are preferred over Wang tiles in computer graphics by now [Lag07, PGGM09, SD10a].

To define corner tiles more formally, let T be a finite set of corner tiles and let $\mathcal{C} = \{0, 1, \dots, C-1\}$ be the set of $C \geqslant 2$ different colors in T. As the tiles have four corners, T can contain at most C^4 tiles. These tiles can be uniquely identified by their corner color combination or by a tile index i, i.e., they can be represented by C-ary numbers with four digits $(c_j)_{j=0}^{3}$ or by the decimal integers $0, 1, \dots, C^4-1$. The two representations are connected by common radix conversion, i.e.

$$i = \sum_{j=0}^{3} c_j(i)C^j \quad \text{and} \quad c_j(i) = (i/C^j) \bmod C \tag{2.1}$$



(a) (b)

Figure 2.2: (a) The complete corner tile set over two colors. (b) A valid tiling using this set.

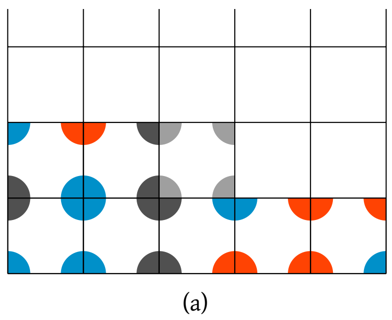
for $0 \leqslant j \leqslant 3$. In addition to the enumeration scheme, we sometimes denote the corners based on common compass directions c_{nw}, c_{sw}, c_{se}, and c_{ne}. Figure 2.2 shows all corner tiles over two colors and a valid tiling using this set. We call corner tile sets that contain all corner tiles over C colors *complete*.

2.4 Corner Tilings

How can we arrange corner tiles such that they form a valid tiling, i.e., a tiling where tiles have matching corner colors? There are several requirements for such a tiling algorithm:

1. The algorithm should be efficient.
2. The algorithm should generate tilings that are non-periodic.
3. The algorithm should generate tilings that do not show local repetitions.

Why are these aspects important? Efficiency is important because the tiling algorithm forms the part of any tile-based method that is performed online. (Recall Figure 1.1.) If the algorithm would not be efficient, the application of a tile-based method would probably not be advantageous in the first place. In particular, the algorithm should be compatible to the parallel nature of graphics hardware. The second requirement is important because we generally want to hide the tile-based nature of the generation process as much as possible. Generating tilings that are non-periodic ensures that we minimize artifacts stemming from global tile repetitions. The third requirement is more subtle and depends on the application. Many tile construction procedures associate each corner color with specific content such that repetitions of a corner color directly translate to artifacts in the synthesized results. Thus, we equally want to minimize artifacts



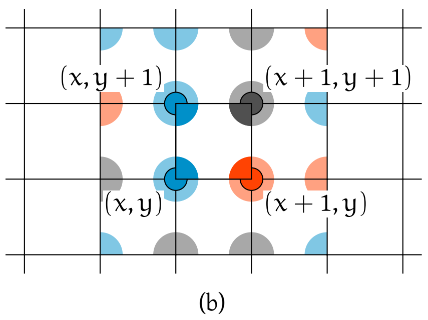


Figure 2.3: Two tiling approaches. (a) Tiling by placing corner tiles one by one in scanline order. (b) Tiling by assigning colors to an underlying integer lattice.

stemming from local color repetitions. We will examine each of the following tiling algorithms more closely with respect to these three requirements.

Algorithmic Approaches

Cohen et al. [CSHD03] proposed a straightforward tiling algorithm for Wang tiles that naturally translates to corners tiles. Tiles are placed in scanline order such that neighboring tiles have matching corner colors (see Figure 2.3). Each tile is chosen randomly among those tiles with a matching corner color combination. Such a scanline approach fulfills our second requirement since choosing tiles randomly prevents strictly periodic tilings. A general deficit of this approach, however, is that it does not allow local tile evaluations: A tiling has to be generated in its entirety in order to evaluate a single tile of interest. A better way is to align tile corners to the integer lattice points which have been assigned a color $c \in \mathcal{C}$. This way, tiles are implicitly defined by the resulting corner color combinations and can be evaluated locally [Wei04, LD06a]. This approach is captured by a function h that maps lattice points to colors, i.e.

$$h : \mathbb{N}_0^2 \to \mathcal{C}.$$

We call h the *color distribution function*. Tiling algorithms based on such a color distribution function are called *direct* tiling algorithms (in contrast to the aforementioned scanline algorithm). In the following, we consider different types of color distribution functions for such a direct tiling algorithm and analyze the types of tilings they produce.

Stochastic Tilings

An analogue way to produce non-periodic tilings using a color distribution function is to generate a random integer at each lattice point and then map this integer to the set of corner colors $\mathcal{C}$. This can be achieved by building h upon stochastic hash functions that ensure that corners shared by neighboring tiles obtain the

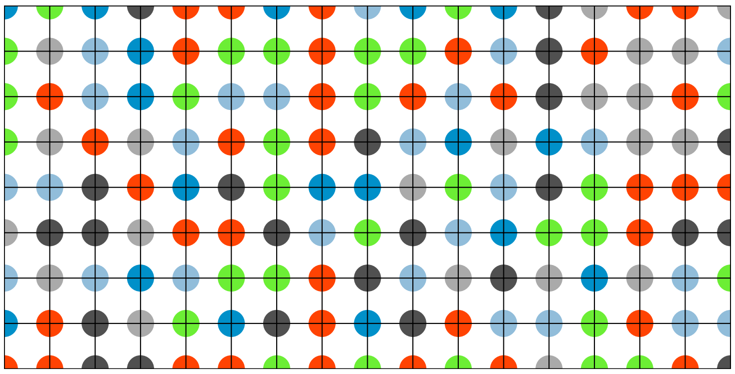


Figure 2.4: Stochastic tiling over 6 colors.

same colors despite being evaluated independently. An example would be a hash function of the type

$$h'(x, y) = P\big[(P[x \bmod n] + y) \bmod n\big]$$

where P is a permutation table of size n. The resulting color distribution function is then given by $h : (x, y) \mapsto h'(x, y) \bmod C$. Although, strictly speaking, a permutation table-based color distribution function behaves deterministic, we call these tilings *stochastic tilings*: They produce (pseudo-)random tilings in the same sense linear congruential generators produce (pseudo-)random numbers [LD06b]. Figure 2.4 shows a stochastic tiling of size 16×8 over $C = 6$ colors.

Tilings based on stochastic hash functions are efficient and produce non-periodic tilings since the random distribution of corner colors ensures a random distribution of tiles. Hence, they fulfill the first two of our stated requirements. The problem with stochastic tilings, however, is that this random distribution of corner colors doesn't impose any restrictions on the local distribution characteristic of each corner color. Distributing colors randomly means that there can be larger clusters of corners with the same color which results in unsatisfactory local repetitions. We will analyze these defects more carefully after the discussion of the following deterministic procedure.

Deterministic Tilings

We now turn to a novel color distribution function which does not involve randomness and is fully deterministic. In particular, we are interested in a more uniform distribution of colors in the following sense:

1. If a lattice point has a color c, it should be less likely that one of its neighbors has the same color c. This avoids local clusters of the same color.
2. The overall distribution of a color c_i should be similar to the overall distribution of any other color c_j. This ensures that no color is favored in any part of the tiling space.

3. The distribution of all colors should not be regular but maintain a pseudo-random appearance. This helps to avoid easily recognizable global patterns of repetition.

These requirements parallel the characteristics of some point sequences of low discrepancy [Nie92]. These sequences unfold incrementally in such way that not only the total sequence up to n points offers favorable uniformity properties but also the subsets of a consecutive partition of these points. Of particular interest are radical-inverse based sequences which are intrinsically stratified and can be utilized to pseudo-randomly enumerate the integer lattice.

In the following, we make use of such a low-discrepancy sequence to design a color distribution function that yields a more uniform distribution of corner colors than stochastic hash functions. We do this by mapping the enumeration indices of the low-discrepancy sequence to the set of corner colors. This way, we are able transform the uniformity properties of the considered sequence to a deterministic color distribution function h.

Discrepancy

First of all, we briefly specify in which sense the utilized point sequences are uniform. Let $X = x_0, \dots, x_{n-1}$ be a sequence of n points in the s-dimensional unit cube I^s. Let $B = \{[0, \nu_1] \times [0, \nu_2] \times \dots \times [0, \nu_s]\}$, $\nu_i \in [0, 1)$, be the family of box-shaped subsets of I^s anchored at the origin. Let χ_b be the characteristic function of a set $b \in B$ and $\lambda(b)$ its volume. Then the measure

$$D_n^*(B, X) := \sup_{b \in B} \left| \frac{1}{n} \sum_{i=0}^{n-1} \chi_b(x_i) - \lambda(b) \right| \tag{2.2}$$

is called the *star-discrepancy* [Nie92]. It can be interpreted as the worst error possible if the points of sequence X are utilized to approximate the volume of any box in B. A sequence X is said to be *low-discrepancy* if $D_n^*(X) \in \mathcal{O}(\log^s n/n)$.

Scaled Halton Sequence

Among many low-discrepancy sequences, we found the unscrambled Halton sequence [Hal60] to be a suitable choice for our purposes. It is based on the van der Corput radical inverse function [vdC35] which maps integers to the unit interval by mirroring the integer's b-adic expansion around the radix point. The radical inverse function is

$$\begin{aligned} \phi_b : \mathbb{N}_0 &\rightarrow \mathbb{Q} \cap [0, 1) \\ i = \sum_{k=0}^{\infty} a_k(i) b^k &\mapsto \sum_{k=0}^{\infty} a_k(i) b^{-k-1} \end{aligned} \tag{2.3}$$

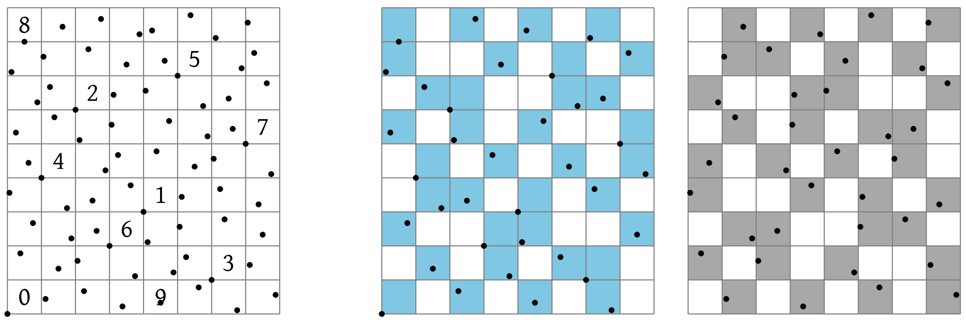


Figure 2.5: The first 72 points of the scaled Halton sequence induce a stratification grid of size 8×9. Dividing the point sequence on the basis of these indices yields a uniform subdivision of the grid cells.

where $a_k(i)$ denotes the $(k+1)$-st digit of the integer $i \in \mathbb{N}_0$ in base b. The two-dimensional *Halton sequence* is then defined as

$$x_i := \left(\phi_{b_1}(i),\ \phi_{b_2}(i)\right)$$

for relatively prime bases b_1 and b_2. A typical choice for the bases are the first s prime numbers for s dimensions, that is we choose $b_1 = 2$ and $b_2 = 3$.

Decomposing the integer into n least significant digits $l \in \{0, \ldots, b^n - 1\}$ and the remaining most significant digits h by $i = b^n h + l$ reveals the stratification property [Kel04]:

$$\begin{aligned} \phi_b(i) &= \phi_b(b^n h + l) = b^{-n}\phi_b(h) + \phi_b(l) \\ \Leftrightarrow\ b^n \phi_b(i) &= \underbrace{\phi_b(h)}_{\in[0,1)} + \underbrace{b^n \phi_b(l)}_{\in[0,b^n)}, \end{aligned}$$

i.e., the least significant digits select the stratum and the most significant digits determine the point inside that stratum. Thus, if we multiply the point coordinates of the Halton sequence by powers of their respective bases the *scaled Halton sequence*

$$x'_i := \left(b_1^{n_1}\phi_2(i),\ b_2^{n_2}\phi_3(i)\right)$$

induces a stratification grid of size $b_1^{n_1} \times b_2^{n_2}$ with exponents $n_1, n_2 \in \mathbb{N}_0$. Figure 2.5 (left) shows an example where the first 72 points induce a stratification grid of size $2^3 \times 3^2$. While the sequence unfolds, the grid cells get enumerated by indices $i' \in \{0, \ldots, b_1^{n_1} b_2^{n_2} - 1\}$ as shown for the first 10 indices.

Deterministic Color Distribution Function

One key observation is that the scaled Halton sequence enumerates the intrinsic stratification grid and hence delivers a pseudo-random permutation of the grid

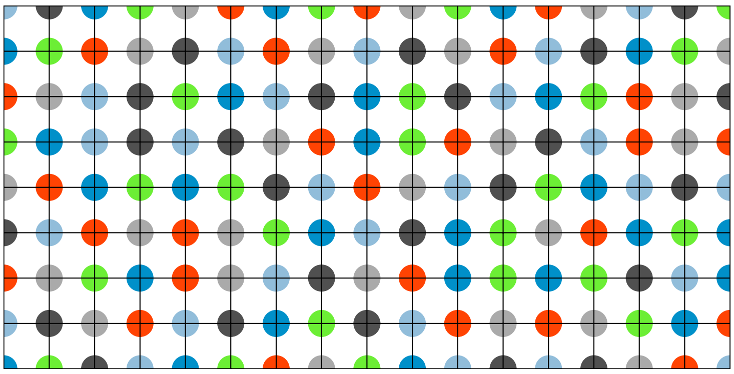


Figure 2.6: Deterministic tiling based on the Halton sequence over 6 colors.

coordinates. Another key observation is that if we partition the Halton points into k subsets of size m by

$$\{x_0, \dots, x_{b_1^{n_1} b_2^{n_2} - 1}\} = \bigcup_{i=0}^{k-1} \{x_{im}, \dots, x_{(i+1)m}\}, \quad k, m \geqslant 1,$$

every subset is of low-discrepancy, too [KG12]. Likewise, we can partition the total set of grid cells into k subsets on the basis of the Halton point indices i' and assume each subset to still be of good uniformity. Figure 2.5 (right) shows an example for $k = 2$ where the first $m = 36$ cells constitute the first subset (blue) and the second 36 cells the second subset (gray).

These observations let us construct a color distribution function for corner tilings by considering the mapping

$$h : (x, y) \mapsto \left\lfloor \frac{i'}{b_1^{n_1} b_2^{n_2}} C \right\rfloor .$$

Here, we choose the base exponents n_1 and n_2 such that $b_1^{n_1} \geqslant T_x + 1$ and $b_2^{n_2} \geqslant T_y + 1$ for a tiling of size $T_x \times T_y$. Since the Halton sequence is deterministic we call the resulting tilings *deterministic* as well. Figure 2.6 shows a deterministic tiling of size 16×8. Although it is subtle in this example, note the more uniform distribution of colors compared to Figure 2.4.

Implementation

Currently, the deterministic color distribution function is not efficient because the enumeration indices i' are derived in a “forward” manner from the original integers i and not directly from grid coordinates (x, y). We would have to compute every Halton point up to the first one that falls into cell (x, y). To solve this problem, we utilize a connection between the index of a Halton point and the coordinates of its grid cell discovered by Grünschloß et al. [GRK12]. It is based on the aforementioned decomposition of the index into least and most significant digits where the least significant digits identify the stratum of the corresponding Halton point. They can be derived from grid coordinates (x, y) by

$$l_1 = \phi_{b_1}^{-1}\left(\frac{x}{b_1^{n_1}}\right) \text{ and } l_2 = \phi_{b_2}^{-1}\left(\frac{y}{b_2^{n_2}}\right),$$

where ϕ_b^{-1} is the inverse of (2.3) and reverses the digits again to yield the enumeration index. With the least significant digits at our hand, the index i' can be reconstructed using the Chinese remainder theorem [CLRS09, GRK12] as

$$\begin{aligned} i' &= (l_1 p_2 m_1 + l_2 p_1 m_2) \bmod (p_1 p_2) && (2.4) \\ &= \big(p_2(l_1 m_1 \bmod p_1) + p_1(l_2 m_2 \bmod p_2)\big) \bmod (p_1 p_2). && (2.5) \end{aligned}$$

Here, the $p_k = b_k^{n_k}$ denote width and height of the stratification grid and the m_k their modular multiplicative inverses $(p_1^{-1} \bmod p_2)$ and $(p_2^{-1} \bmod p_1)$. To illustrate the full method, we include a GLSL example implementation as part of a tile-based texture mapping application in Appendix A.1.

Evaluation

Let us review this approach with respect to efficiency, periodicity, and local color repetitions.

An implementation based on Equation (2.5) makes our deterministic tilings as efficient as tilings based on stochastic hash functions with a time complexity that is constant per lattice point. This allows realtime applications for e.g. tile-based texture mapping at several hundreds frames per second.

Can these deterministic tilings be considered non-periodic? In theory, yes, because the radical inverse function is a bijection [Kel03]. Equation (2.5) illustrates that, in practice, the method is bounded by the largest intermediate result that fits into the available integer range. In fact, our transformation in (2.5) is motivated by fact that the products $l_1 p_2 m_1$ and $l_2 p_1 m_2$ in (2.4) quickly exceed a typical 32-bit integer range. Due to the transformation the largest subtotal becomes $\max(p_1^2, p_2^2, 2\,p_1 p_2)$ since $l_1, m_1 < p_1$ and $l_2, m_2 < p_2$. For example, for square tilings this prevents periodicity until sizes of approximately 46341^2.

To analyze local repetitions, we consider both a quantitative and a qualitative analysis. For the quantitative analysis, we look at the local 8-neighborhood of each corner and count the number N_I of identically colored neighbors. Since we can expect an average $\mu(N_I) = 8/C$ for stochastic tilings, smaller values indicate fewer local clusters:

Method $C =$	2	3	4	5	6	7	8
Stochastic	3.9970	2.6653	1.9981	1.5981	1.3328	1.1421	0.9993
Deterministic	3.4967	2.1548	1.5566	1.1536	0.8032	0.6968	0.6058

These results are averages over 100 tilings of random sizes up to 4096×4096.

For the qualitative analysis, consider Figure 2.7 which lists several stochastic and deterministic tilings of size 30×30. Tile borders have been removed to emphasize the color distribution across the underlying integer lattice. If we review these tilings with respect to our uniformity criteria (see page 12) we see that

1. Clusters of the same color are largely absent for the deterministic tilings while they can be quite prominent in the stochastic case, for example in the bottom center for $C = 3$. This behavior is a consequence of the low-discrepancy property of the underlying Halton sequence.

2. Each color shows comparable distribution characteristics across the tiling space as illustrated by the individual color charts below each tiling. This behavior is a consequence of the low-discrepancy property of the subsets of the partitioned Halton sequence.

3. The color distributions are irregular and maintain a pseudo-random appearance. This is a consequence of the radical inverse function which (a) is bijective and (b) is applied to two coprime bases which prevents apparent correlations between both dimensions.

One may observe a slight directional pattern in the deterministic distribution of colors. We will see in Chapter 3 that for an application like tile-based texture synthesis these artifacts are a lot less severe than the local color clusters of stochastic tilings.

Semi-Stochastic Tilings

As a last color distribution function, we consider a mixture between the presented stochastic and deterministic approaches. The idea is to locally vary the probability for each corner color and therefore allow a user-influenced distribution of corner colors. Because each color will be associated with specific content, this yields a direct way to influence the synthesized result. We will see in Chapter 3 that this allows tile-based texture synthesis to include globally varying textures.

We first define a random field on the unit square that provides the probability for each corner color at each point in the square, i.e., each point $x \in [0,1]^2$ is assigned a discrete random variable X_x that can take values from our set of colors $\mathcal{C}$. Let $p_c \equiv \Pr(\{X_x = c\}), c \in \mathcal{C}$, and let

$$P := \begin{pmatrix} 0 & 1 & \cdots & C-1 \\ p_0 & p_1 & \cdots & p_{C-1} \end{pmatrix}, \quad \sum_{c=0}^{C-1} p_c = 1$$

denote a discrete probability distribution (probability mass function) [Ros02]. With this definition the user may provide a *color probability function* $\rho : [0,1]^2 \to \mathcal{P}$

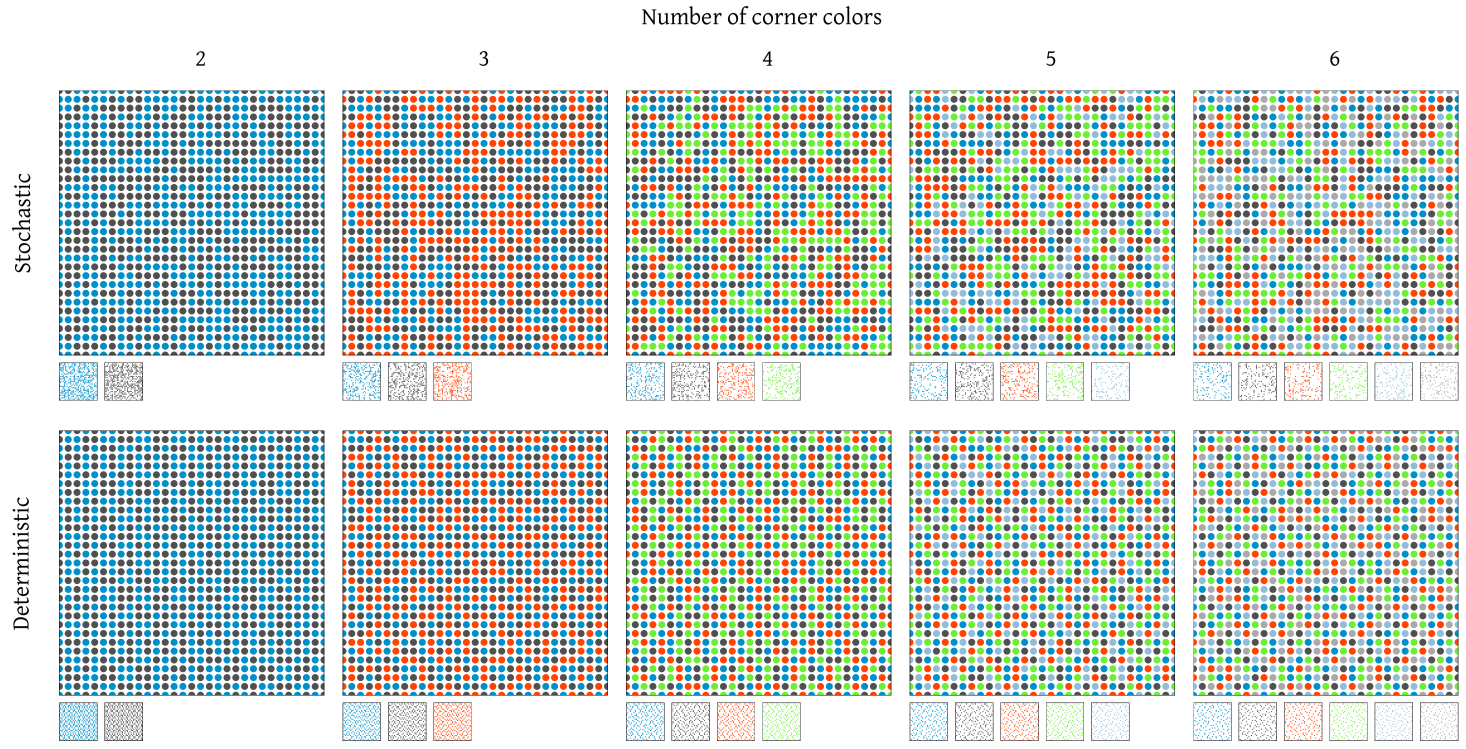


Figure 2.7: Corner color distribution of several stochastic and deterministic tilings of size 30×30.

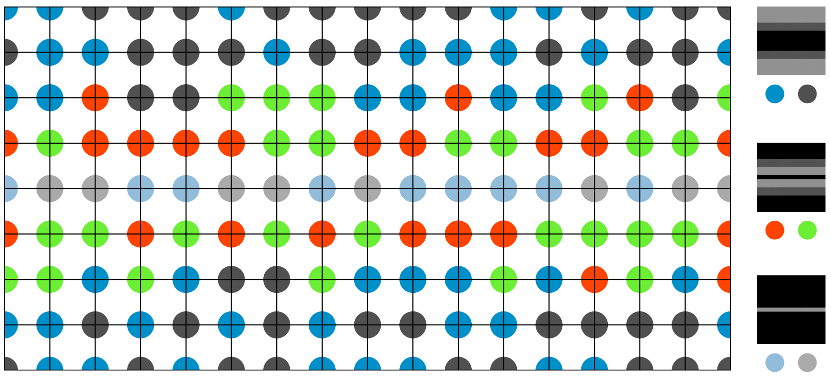


Figure 2.8: Semi-stochastic tiling over 6 colors.

where $\mathcal{P}$ denotes the function space of all probability mass functions, $\mathcal{P} = \{P : \mathcal{C} \to [0,1] \mid \sum_{c=0}^{C-1} p_c = 1\}$. Thus, ρ assigns each point x an individual discrete distribution P_x.

To determine a corner color at a given lattice point (x, y) we now sample ρ and the random variable at this lattice point. Hence, the corresponding color distribution function h is given by

$$h : (x, y) \mapsto X\left[\rho\left(\frac{x}{T_x+1}, \frac{y}{T_y+1}\right)\right]$$

where $T_x \times T_y$ denotes the size of the desired tiling. We call these tilings *semi-stochastic*.

Figure 2.8 shows a semi-stochastic tiling of size 16×8. The underlying color probability function ρ is depicted to the right where probabilities map to gray values. (Black corresponds to 0, white corresponds to 1.) In this example, pairs of corners share the same probability distribution which makes them equiprobable. Note that the probabilities sum up to one everywhere.

Let's take a closer look at semi-stochastic tilings in terms of the three requirements efficiency, periodicity, and local color repetitions.

In contrast to tilings based on a stochastic hash function, each corner color is not computed directly but by sampling the random variable at a given lattice point. Sampling a discrete random variable, however, can be done very efficiently in constant time using the method of "aliases" [Wal77, Knu97]. Obtaining consistent results for corners that are shared by neighboring tiles can be guaranteed by utilizing an (x, y)-dependent random number generator similar to the one underlying stochastic hash functions. This leaves us only with the additional burden of having to store the color probability function ρ which is often given in discrete form in practice. Overall, semi-stochastic tilings can be generated with similar efficiency to stochastic and deterministic tilings.

The problem of local or global repetitions—and thus periodicity—cannot be answered in general for semi-stochastic tilings because it largely depends on the given color probability function. For example, if a color probability function makes every corner color equiprobable everywhere, the tilings behave like stochastic tilings and share their properties in terms of global and local repetitions. If a color probability function is constant (i.e., yields the same single color everywhere), a tiling using a single tile is enforced, and repetition artifacts are severe. One could try to design a color probability function in such a way that repetitions become less likely but this is cumbersome as long as there is some degree of randomness involved.

Pruned Tile Sets

The color probability function in Figure 2.8 has been chosen to demonstrate that semi-stochastic tilings show an interesting behavior when looking at the necessary number of tiles. In this example, there are no tiles that contain both one of the outer corner colors (dark blue, dark gray) as well as one of the inner corner colors (light blue, light gray). This means we can reduce the size of the necessary tile set based on the range of tilings that can occur. Since we know that a corner color c_i will never be adjacent to a corner color c_j, we are able to omit all tiles with this pair of colors. We call such tile sets *pruned*. If an application only works with one or a few color probability functions it is likely that pruned tile sets will be sufficient for this application. This is an advantage of semi-stochastic over stochastic and deterministic tilings.

Identifying non-adjacent pairs of colors is difficult for continuous color probability functions but simple for discrete ones which can be quickly scanned for possible corner color combinations. Assume for now that we know that there are k pairs of corner colors that will never share a tile. If we remove all tiles with these k pairs, how many tiles do the pruned tile sets still contain?

Let k denote the number of such mutually-exclusive pairs of corner colors. From the inclusion-exclusion principle we know that the number of tiles without an s-element set $S \subseteq \mathcal{C}$ of corner colors equals

$$\begin{aligned} N_s &= \sum_{i=1}^{s} \binom{s}{i} (-1)^{i+1} (C-i)^4 \\ &= C^4 - \underbrace{\sum_{i=0}^{s} \binom{s}{i} (-1)^{i} (C-i)^4}_{=0 \text{ if } s>4} . \end{aligned} \tag{2.6}$$

We see that the second summand disappears for $s > 4$ since $(C-i)^4$ is a polynomial of degree 4 [GKP94]. This reflects the fact that there can be no tile with an $(s > 4)$-element set of corner colors when we have tiles with just four corners. Subsequently, let M_k denote the total number of tiles without k mutually exclusive pairs of corner colors. Again, the inclusion-exclusion principle tells us

k\C	2	3	4	5	6	7	8	9
0	16	81	256	625	1296	2401	4096	6561
1	2	31	146	431	994	1967	3506	5791
2	-	-	60	261	716	1557	2940	5045
3	-	-	-	-	462	1171	2398	4323
4	-	-	-	-	-	-	1880	3625

Table 2.1: Sizes of pruned corner tile sets when all tiles with k mutually-exclusive pairs of corner colors can be omitted from the corresponding complete corner tile sets over C colors.

that

$$M_k = \sum_{p=1}^{k} \binom{k}{p} (-1)^{p+1} N_{2p}.$$

Using (2.6), we can transform this to

$$\begin{aligned} M_k &= \binom{k}{1} N_2 - \binom{k}{2} N_4 + \sum_{p=3}^{k} \binom{k}{p} (-1)^{p+1} C^4 \\ &= C^4 - 2k(6C^2 - 12C - 6k + 13), \end{aligned}$$

for $0 \leqslant k \leqslant \lfloor C/2 \rfloor$. This means, pruned tile sets are of size $|T_k| = M_k$ if k pairs of corner colors can be omitted. Table 2.1 gives an impression of the sizes of several pruned tile sets. For example, for $k = 1$ pruned tile sets are 61.7%, 43.0%, 31.0% and 23.3% smaller than complete tile sets for $C = 3, 4, 5, 6$.

2.5 Conclusion

We defined corner tiles and showed that they can tile the plane by deriving a tiling from a colored integer lattice. Efficient tiling algorithms are then given by color distribution functions which map lattice coordinates to the set of corner colors. We presented two such color distribution functions as alternatives to the stochastic hash functions in the literature. The first alternative is a deterministic tiling algorithm based on the Halton low-discrepancy sequence. We showed that this deterministic method yields a more uniform distribution of corner colors across the tiling space and derived an efficient implementation. The second alternative are semi-stochastic tilings which generalize the concept of stochastic tilings and locally vary the probability for each corner color. We saw that these tilings can be controlled by a user-defined color probability function and that complete tile sets can be pruned by excluding tiles based on pairs of colors that will not be adjacent. In the next chapter, we will see that both of these methods are of value for tile-based texture synthesis.

3

Tile Construction for Textures

In the preceding chapter we saw that there are efficient algorithms to tile the plane using corner tiles. By filling the tiles with content that is otherwise expensive or difficult to generate, we can gain a significant increase in efficiency. This chapter is interested in filling corner tiles with texture images, i.e., images that mimic the surface properties of complex real-world objects. As surface properties are often difficult to describe mathematically, textures in the form of specifically prepared images (e.g. edited photos) are one of the most important tools to increase the realism in computer generated images. And because they are image-based, they are typically able to capture a broader range of surface properties than textures that can be generated procedurally [EMP*03].

Texture images, or simply *textures*, can be characterized as Markov random fields in order to distinguish them from general images [Li09, WLKT09]. From this perspective, they can be viewed as realizations of stochastic processes that are both local and stationary. An image is *local* if the properties of a certain pixel only depend on the properties of its neighboring pixels. An image is *stationary* if we observe it through a window of fixed size and find its appearance similar no matter where we place this window. We will be mostly interested in stationary textures in this thesis but will consider *globally varying* (non-stationary) textures as a possible application scenario for semi-stochastic tilings.

Generating texture images in a (semi-)automatic way is known as *texture synthesis*, and generating a large texture out of small (often real-world) examples is called *example-based texture synthesis.* Example-based texture synthesis in turn can be roughly categorized as *pixel-based* or *patch-based* [WLKT09]. Pixel-based methods generate the output texture pixel by pixel from the provided exemplar. Patch-based methods, on the other hand, copy larger portions of the exemplar to the output texture and then try to merge these patches without apparent seams.

In the broad field of example-based texture synthesis, tile-based approaches stand out as one of the fastest methods to synthesize textures. They form a special case of patch-based texture synthesis where the tiles form patches that seamlessly fit together. The construction of the tiles themselves, however, can take place pixel-based or patch-based, depending on the utilized method. In this thesis, we build upon existing work in patch-based texture synthesis to fill the interiors of our corner tiles. We present a variant of the tile construction process by Lagae and Dutré [LD06a] and use it to evaluate the influence of our tiling algorithm. We will see that tiling the plane using our deterministic tiling algorithm readily improves the final result.

What makes this form of *tile-based texture synthesis* so attractive is that most of the computational effort is shifted to a preprocessing phase during which the tiles are constructed. Only the underlying tiling process is performed online which makes the final synthesis very fast. Moreover, it is easy to integrate into a standard graphics pipeline for which we include an implementation that complements our tiling algorithm from the previous chapter. We will also see that our semi-stochastic tilings are compatible with this approach. This allows us to generate globally varying textures in a similar way.

3.1 Related Work

Tile-based texture synthesis was first considered by Stam [Sta97] and later extended to example-based texture synthesis using tiles by several researchers [CSHD03, NWT*05, LD06a]. Cohen et al. [CSHD03] merged different patches of an input texture by constructing Wang tiles in correspondence to their edge colors and then generated stochastic tilings. Wei [Wei04] improved this stochastic algorithm by allowing random access to tiles which is important for tile-based texture mapping [LN03, Lef08]. Lagae and Dutré [LD06a] translated the approach of Cohen et al. to corner tiles, and Fu and Leung [FL05] extended the tiling mechanism to arbitrary surfaces. All of these approaches only consider stochastic tilings.

Cohen et al. [CSHD03] were also the first to consider tile construction from multiple input textures to generate globally varying textures. The example-based synthesis of such textures was also considered by several pixel- and patch-based approaches [Ash01, MZD05] but their performance is not comparable to a tile-based approach. Neyret and collaborators [NC99, LN03] considered to control tilings based on a probability distribution for manually created textures and small patterns (textons), as did Lu et al. [LEQ*07] for volume illustrations but these techniques do not directly translate to example-based texture synthesis. Texture synthesis can also be viewed as an optimization problem where output pixels are chosen according to an energy function that respects neighborhood similarities [KEBK05] but such a method is not suited for realtime synthesis.

As mentioned, pixel- and patch-based approaches are often incorporated into tile-based texture synthesis during tile construction [NWT*05, LD06a, DZP07]. For example, we will use a variant of the "image quilting" technique by Efros and Freeman [EF01] to merge different patches inside our tiles. We will briefly explain this technique in the next section.

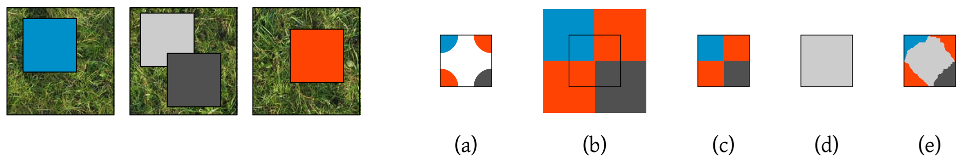


Figure 3.1: The corner tile construction process for textures. (Left) Selecting example patches from several source textures. (Right) The selected patches are arranged and merged to construct a specific corner tile.

3.2 Tile Assembly

We want to use corner tiles for example-based texture synthesis where the output texture should look like a larger portion of an exemplary input texture. In many cases, a single input texture is already sufficient for believable synthesis results but the results generally get better if more than one exemplar image is considered. If these exemplar images stem from the same source (e.g. a large grass texture), we want the increase in variety to be reflected in the resulting tile set. Similarly, if these exemplar images stem from related but different sources (e.g. two grass textures where one also shows flowers), we want these sources to be represented equally in the resulting tile set.

We extend the construction process by Lagae and Dutré [LD06a] to respect these requirements. Figure 3.1 (left) shows the first part: For a corner tile set over C colors, C patches are evenly selected from the given source texture(s). Most of the time, we have a one-to-one relationship between source textures and corner colors but if there are fewer or more than C source textures, the assignment is performed in a round-robin fashion. A *patch* is a square cutout of a given source texture at usually a quarter of the source's size. The patches' locations are chosen randomly within the boundary of their corresponding source textures. The gray patch, however, is a unique patch for each tile whose location is chosen during the second part of the construction process. Since we don't want to favor one of the source textures in the resulting tile set, the source texture for this gray patch may be obtained by interpreting a tile's corner color combination as a discrete probability distribution. When sampled, this distribution yields the value of the predominant corner color with a higher probability. For instance, in the example, it is twice as probable that the gray patch is chosen from the input texture corresponding to the red corner than from the other two.

One important aspect of this selection process is that it leads to a balanced representation of the source textures across the resulting tile set. Another aspect is that there is a direct connection between corner colors and synthesized output, i.e., distributing corner colors roughly corresponds to distributing texture content. For semi-stochastic tilings, this means that if we increase the probability of a desired corner color for a specific tiling region, then this leads to a proportional increase of the probability that the associated texture content will dominate this region after synthesis.

Figure 3.1 (right) shows the second part of the construction process where the selected patches are arranged and merged to construct a specific tile. It consists of five steps:

(a) Identify the tile's corner color combination.

(b) Arrange the associated patches according to this combination.

(c) Cut out the parts of the patches that reside inside the tile's boundary.

(d) Cover the tile with a unique gray patch.

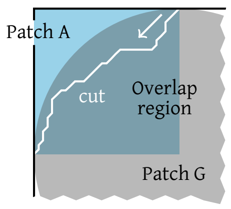
(e) Merge the gray patch with the other patches.

Note that because, in step (b), each corner patch is centered at the corresponding tile corner, tiles with matching corner colors will later show texture content that seamlessly fits together. We could omit the gray patch and directly try to merge the corner color patches but results are generally better when each tile offers parts of the input textures that is unique to this tile. This increases the overall variety of texture content over the whole tile set.

To merge the patches, we use a variant of the "image quilting" method by Efros and Freeman [EF01]. The method consists of two steps:

1. Find a cover patch (gray patch) that best fits the selected corner patches.

2. Find the minimum error cut across the overlapping region of each corner patch with the chosen cover patch.



Both steps consider the mean squared error (MSE) between grayscale versions of the relevant patches to estimate their matching quality. Since tile construction is performed offline and because computing the matching quality is reasonably fast, the first step can often be done by an exhaustive search that finds the cover patch with the globally minimal MSE. The second step requires a bit more care and is illustrated in the inset figure. For a corner patch A, the overlap region is defined as the intersection of A with the gray patch G, confined to the quadrant of the disk centered in the tile. (The restriction to the disk prevents the merging process to replace texture content close to the tile's corners.) Within this overlap region, the minimum error cut is computed by dynamic programming: start from one of the region corners and follow the minimum error in the local neighborhood of the current pixel. For the upper left corner in the example and a current pixel (x, y), this neighborhood is given by the pixel locations $(x-1, y)$, $(x-1, y-1)$, and $(x, y-1)$. The neighborhoods for the other corners follow analogously.

The following figure shows an example result of the whole tile construction process where a complete corner tile set over two colors has been automatically generated from the input texture to the left:

In this example, the input texture is of size 128×128 so the resulting tile textures are of size 64×64.

3.3 Tile Packings

We typically want to use such a texture tile set within a standard graphics pipeline where it should support all of the features of a conventional (non tile-based) texture, in particular mip-mapping and linear- or higher-order filtering [SM09]. Naively arranging all tiles on a single texture is problematic because this leads to noticeable artifacts along tile boundaries [Wei04]. Lefebvre [Lef08] presented a general solution to this problem using texture arrays where each tile gets separately mip-mapped and filtered. In our case, however, many tiles share the same texture information across their borders which allows a more elegant solution: arranging all tiles of the tile set according to a so-called packing and using this single packing texture as a conventional texture [Wei04].

A *packing* is an arrangement of tiles that is toroidal and uses each tile in a given tile set exactly once (compared to a regular tiling where we use multiple copies of a tile), and the problem of finding one is called the tile packing problem [Wei04, LD06d, LD07]. For complete sets of Wang tiles, the tile packing problem was solved by Wei [Wei04]. Here, solutions for higher dimensional packings can be derived from one-dimensional packings (using Wang dominoes). There is even a closed-form expression for the position of a specific tile in the packing. Finding a tile packing for corner tiles is a lot more difficult because of their stricter matching constraints. The fact that corner tiles also constrain their diagonal neighbors prevents a solution analogous to Wang tiles. For complete tile sets, solutions up to $C = 4$ were found by Lagae and Dutr'e [LD06d, LD07]. In this section, we present packing solutions up to $C = 4$ for so-called compact tile sets which are useful for tile-based texture mapping in offline renderers.

Compact Tile Sets

Figure 3.2 shows such a tile set over $C = 3$ colors. This tile set uses all possible combinations of colors for only three of the four corners. Then, for each of these C^3 *base tiles*, there are exactly two alternatives for the remaining corner. Such tile sets contain $2C^3$ tiles and are called *compact*. Compact corner tile sets are

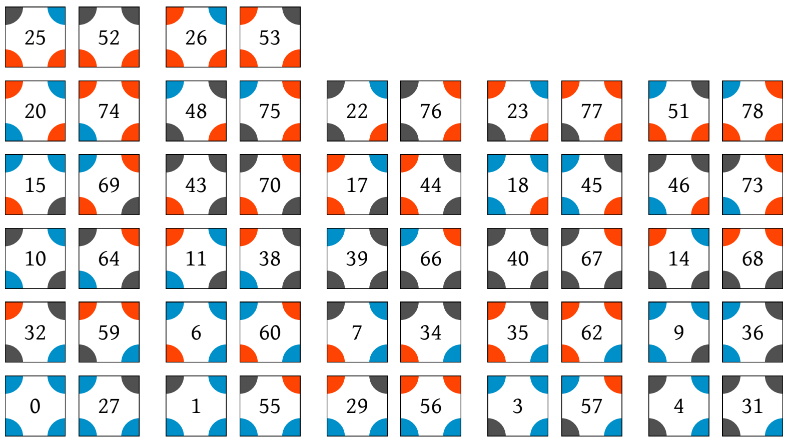


Figure 3.2: A compact corner set over three colors where each corner color appears equally often.

interesting because they are smaller than complete tile sets but still allow non-periodic tilings via a stochastic scanline algorithm (see Section 2.4).

While there is only one complete corner tile set for a given number of colors C, there are several compact tile sets. This is because there is no restriction on the choice of colors for the remaining corner. For example, it can be the same two colors for all tiles or a new choice of two colors for each tile individually. The only exception is the case $C = 2$ for which the compact tile set is identical to the complete one. We saw in the previous section that corner colors will eventually be associated with specific content. For this reason it is preferable to have compact tile sets where each corner color (and thus content) appears equally often.

We want to characterize this class of compact tile sets more precisely. Let j enumerate the C^3 base tiles and let without loss of generality c_{ne} be the corner that can only choose between two colors. Let S_j denote this subset of two colors for tile j, so for each tile $c_{ne} \in S_j$ but $c_{nw}, c_{sw}, c_{se} \in \mathcal{C}$. Over all subsets S_j, each corner color should then appear exactly $2C^3/C = 2C^2$ times:

$$\forall c \in \mathcal{C} : \sum_j \chi_{S_j}(c) = 2C^2, \tag{3.1}$$

where χ_S denotes the indicator function of a subset S. We call compact tile sets that fulfill this condition *balanced*. The compact tile set from Figure 3.2 actually was such a balanced tile set, sorted in pairs of base tiles.

Tile Packings for Compact Tile Sets

In general, the tile packing problem can be considered a constraint satisfaction problem which is solvable by combinatorial search methods [CLRS09]. Lagae and Dutré [LD06d] investigated the packing problem from this angle and employed a backtracking algorithm similar to the classic n-queens problem. This approach greatly reduces the giant search space of $C^4!$ possible arrangements and leads to solutions for complete corner tile sets over two, three, and four colors before excessive runtimes occur. Solutions for compact tile sets, however, were still missing. We partly fill this gap with our solutions for compact tile sets over three and four colors. (Recall that the compact tile set over two colors is identical to the complete one and so are their packing solutions.)

We saw that compact tile sets differ from complete ones in that they only contain all color combinations for three of the four corners. Each of these C^3 base tiles then possesses two alternatives for the remaining corner. As explained before, we are particularly interested in solutions for a compact tile set that is balanced. A desirable side effect is that this further restricts the search space.

A toroidal tiling for all C^4 tiles of a complete corner tile sets can be achieved by arranging them in a $C^2 \times C^2$ square [LD07]. Since compact corner tile sets contain only $2C^3$ tiles, a packing for compact tile sets must often be non-square. For texture mapping, however, it is still preferable to have a solution as close to a square as possible. We thus searched for packings of size $X \times Y = 2C^3$ where without loss of generality Y is the largest integer smaller or equal to $\lfloor \sqrt{2C^3} \rfloor$ that divides $2C^3$ without remainder.

We then employ a backtracking algorithm that places tiles in scanline order as long as some matching condition holds, and that falls back to the last valid arrangement otherwise [LD06d].

```
TILE-PACKING($T, i$)
1  if $|T| == 2C^3$
2      print $T$ // Solution found
3  else
4      foreach $j$ in $\{0, \ldots, C^4 - 1\} \setminus T$
5          if tile $j$ can be placed at position $i$
6              place tile $j$ at position $i$ and add $j$ to $T$
7              TILE-PACKING($T, i + 1$)
8              clear position $i$ and remove $j$ from $T$
```

The search starts by calling TILE-PACKING($\emptyset, 0$). The position i is to be interpreted in scanline order, i.e., tile coordinates are given by $(i \bmod X, i/Y)$. T is the set of tiles that have already been placed. Tiles that are not in T are placed at a position i only if the matching condition in line 5 holds. We set the matching condition to

1. Tile j and its neighbors at position i must have matching corner colors, and
2. Balance condition (3.1) holds.

With this condition, we automatically search for both a tile packing and a compatible balanced tile set.

Figure 3.3 shows the solutions we found. For $C = 3$ the packing is of size 9×6 and for $C = 4$ of size 16×8. Note that corner colors match toroidally and that each color appears exactly $2C^2$ times at a north east corner (or $8C^2$ in total), that is, the underlying compact set is balanced. (It is in fact the compact set from Figure 3.2.) With these solutions, textures based on compact tile sets can be filtered as efficiently as textures based on complete tile sets.

When we started the search, it was not clear if there is a packing solution for compact sets for $C > 2$, in particular with the balance condition. For $C = 2$ it takes only a few milliseconds to find all 32 solutions. (The solutions collapse to a single fundamental solution due to the torus condition.) However, the first solution for $C = 3$ already took a few seconds, and the search for $C = 4$ had to be performed in parallel and took roughly 144 days of CPU time. Once we had a solution, many others could be found through rotation and reflection but it was impossible to count all solutions using the backtracking approach.

3.4 Tile-Based Texture Mapping

Now that we have packing solutions for both compact and complete tile sets, incoming texture coordinates $(t, s) \in [0, 1]^2$ have to be mapped to coordinates $(u, v) \in [0, 1]^2$ in the packing texture. For a tiling of size $X \times Y$ and a packing solution of size $W \times H$, this can be done in several steps:

1. Compute the tile coordinates $(x, y) = (\lfloor tX \rfloor, \lfloor sY \rfloor)$ and the texture coordinates $(u', v') = (\{tX\}, \{sY\})$ with respect to this tile.

2. Retrieve the tile's corner colors and compute its index i by base converting the corner color combination via Equation (2.1).

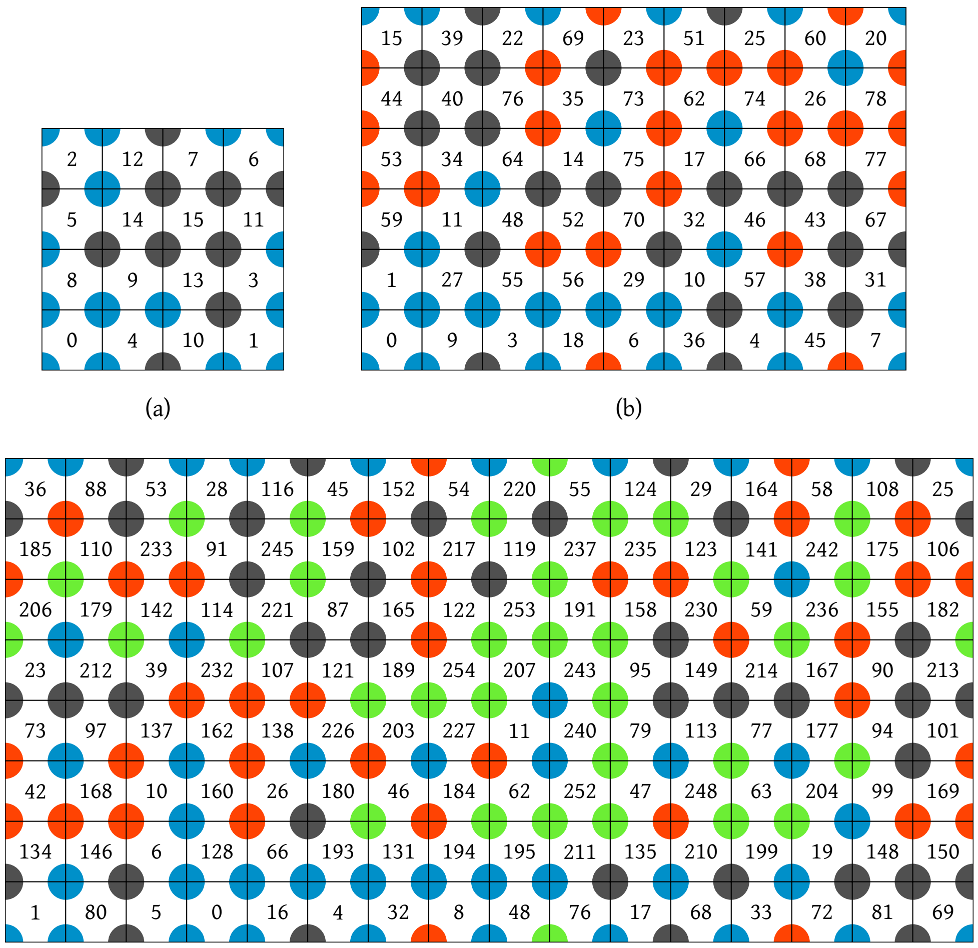
3. Look up the coordinates (x', y') of tile i in the packing solution.

4. Compute the final texture coordinates $(u, v) = \big((x'+u')/W, (y'+v')/H\big)$.

Here, $\{\cdot\}$ denotes the fractional part of a given scalar. To further illustrate the full method, we include a GLSL implementation in Appendix A.1 in combination with our deterministic color distribution function.

3.5 Evaluation of Tiled Textures

We now look at practical results when using our tiling algorithms from the previous chapter for example-based texture synthesis. We use the aforementioned construction process to generate texture tile sets over various numbers of corner colors and generate large non-periodic textures at runtime using tile-based texture mapping. In particular, we compare the results for stochastic tilings to our deterministic tilings using the same sets of tiles.



(c)

Figure 3.3: Tile packings for balanced compact corner tile sets over (a) two, (b) three, and (c) four colors. The solution over two colors was found by Lagae and Dutré [LD06d].

Figures 3.4 and 3.5 show several results when synthesizing stationary textures out of one or more example textures which are depicted below each pair of results. The results are 10×10 cutouts from much larger tilings dynamically generated on the GPU (for a screenshot, see Figure 1.3). The input textures have a resolution of 128×128 pixels, so the cutouts have a resolution of 640×640.

The results for stochastic tilings underline that large clusters of the same corner color can be harmful with respect to texture synthesis: They translate to local repetitions of texture content. Increasing the number of corner colors from two to three or four (and thus the number of tiles from 16 to 81 or 256) does not necessarily improve on this behavior as indicated by the results in Figure 3.5. In contrast, the deterministic tilings show a more even distribution of corner colors that is free of color clusters and makes repetition artifacts muss less detectable, even for $C = 2$ colors with only 16 tiles. Because the tilings are based on the same sets of constructed tiles, these improvements stem solely from the better arrangement of tiles using our method.

Figure 3.6 shows results based on semi-stochastic tilings that generate globally varying textures. In this case, the example textures were taken from different parts of the same underlying non-stationary texture function. This approach extends the range of current tile-based methods and yields a rich set of tiles which can be used to generate large amounts of the same globally varying texture. The corner colors of the underlying tilings have been distributed according to user-specified color probability functions. For example, the bottom result is based on the 6-color tiling from Figure 2.8.

Both of the tiling methods generate textures as fast as existing stochastic tiling methods and run at several hundreds frames per second on graphics hardware. When the utilized color probability functions allow the usage of pruned tile sets, semi-stochastic tilings also require significantly less texture memory (40% to 70% less in our examples). Using semi-stochastic tilings, tiles can also be interactively rearranged in correspondence to a changing color probability function. In this case, the underlying tile sets have to be complete.

3.6 Conclusion

We showed how corner tiles can be filled with texture content such that tiles with matching corner colors seamlessly fit together. We discussed a novel selection process that leads to a balanced representation of multiple example textures across the resulting tile set. We also described the tile packing problem and presented solutions for balanced compact tile sets up to four colors. Solutions for more than four colors remain an open problem, both for complete and compact tile sets.

We saw that for tile-based texture synthesis, our deterministic tiling method leads to less severe repetition artifacts than current stochastic approaches. As a consequence, synthesized textures are of better quality, even when using smaller

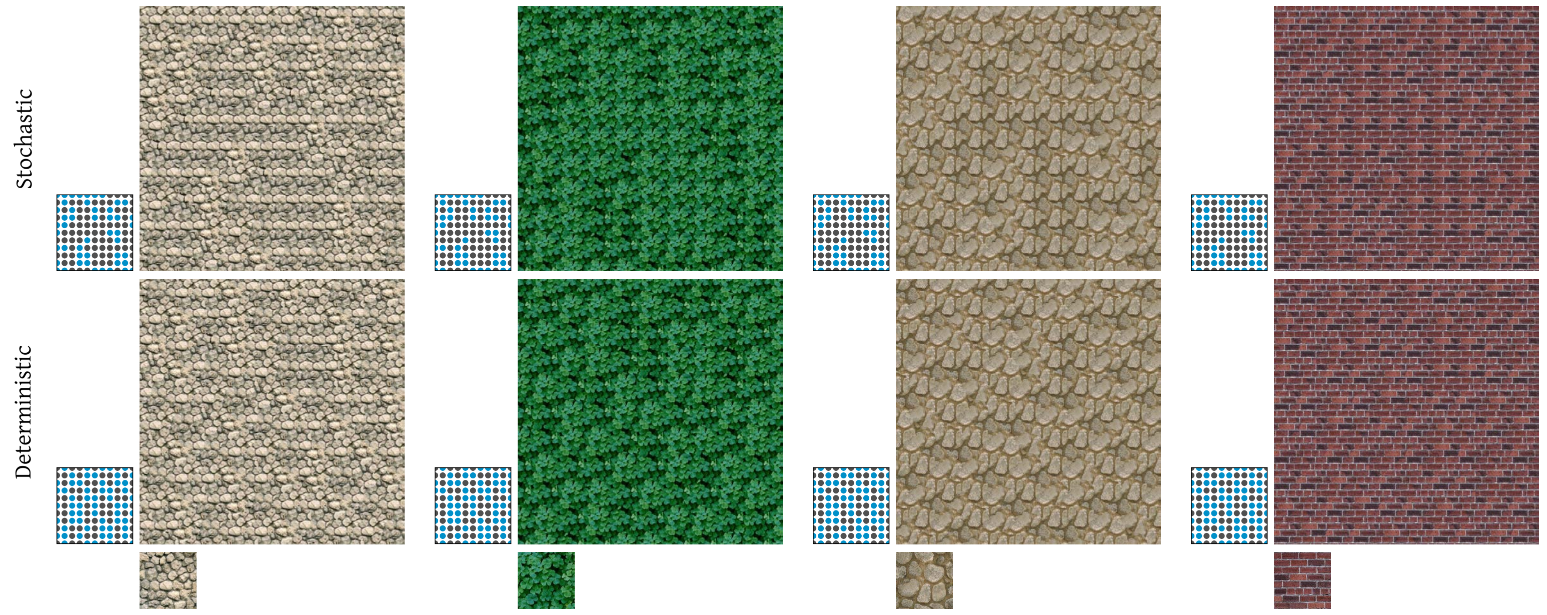


Figure 3.4: Texture synthesis results for corner tilings with two colors.

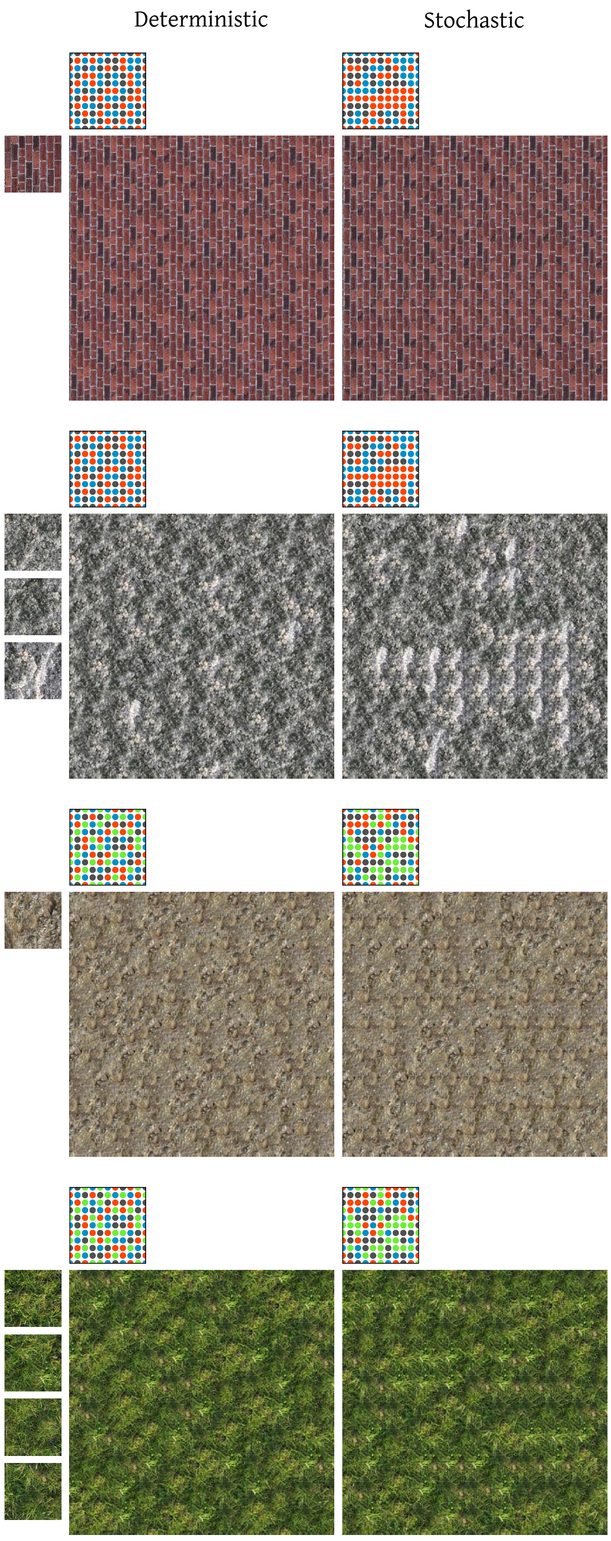


Figure 3.5: Texture synthesis results for corner tilings with three and four colors.

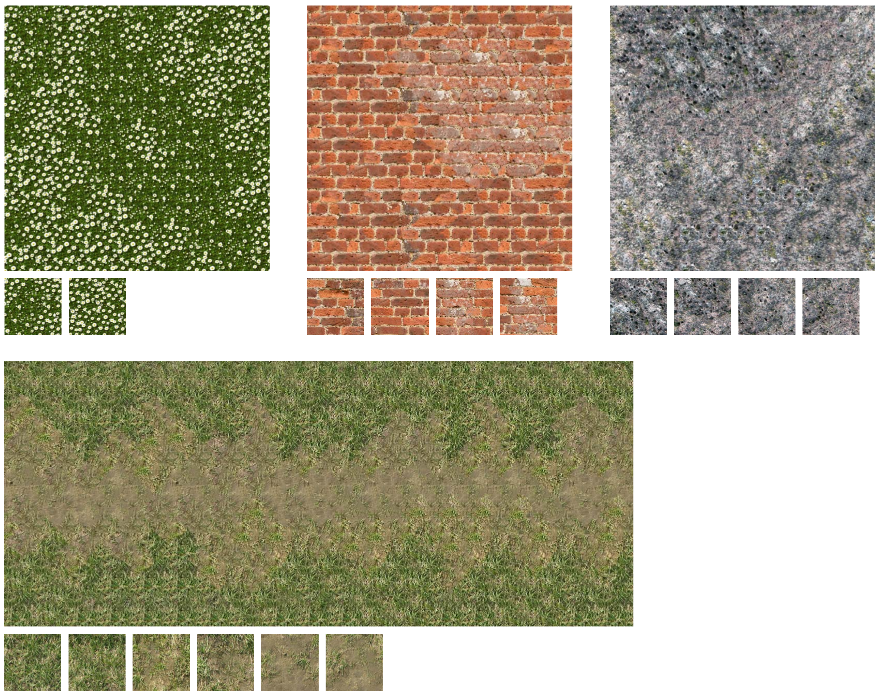


Figure 3.6: Texture synthesis results for semi-stochastic tilings that generate globally varying textures.

sets of tiles. Semi-stochastic tilings allow tile-based texture synthesis to be extended to globally varying textures if the utilized example textures stem from the same underlying texture function. Both methods can provide efficient texturing in connection with tile-based texture mapping.

Obviously, the final quality of the synthesized textures also relies on the pixel- or patch-based approach that is employed during tile construction. The basic algorithm by Efros and Freeman [EF01] was sufficient to generate good results and analyze the influence of the tilings alone. But we can expect even better results if we consider a more recent technique [WLKT09].

4

Tile Construction for Point Sets

We saw that corner tiles are very good at generating large portions of textures once the tiles have been carefully filled with the appropriate content. In this chapter, we are interested in filling corner tiles with another type of content: point sets, i.e., collections of coordinates in two-dimensional space. Point sets are one of the fundamental signals in computer graphics and are at the core of any rendering system. This is because photorealistic image synthesis consists of several sampling problems where the sample positions are either directly given by or derived from point coordinates. These sampling problems range from image plane sampling, light map sampling, and ray tracing to the evaluation of integrals by numerical approximation [PH10]. In addition, point sets appear in non-photorealistic rendering, geometry processing, and object distribution tasks.

In general, there is no universal point set that is the perfect match for every application scenario. For example, while lattice points are acceptable for numerical quadrature, they are generally unacceptable for image plane sampling because of emerging moiré patterns. And while random points with a guaranteed mutual distance are considered excellent for image plane sampling, they are typically inferior to carefully constructed deterministic points in an integration setting. Nevertheless, there are two properties that are favorable across several application scenarios: uniformity and irregularity. A point set is *uniform* when the average point density is approximately constant and when there are no "holes" or "clusters" of points. A point set is *irregular* when there are no symmetries or regular structures in the distribution that may lead to moiré patterns or other visually distracting artifacts. Satisfying both properties is the main challenge that sets the search for point sets in computer graphics apart from similar undertakings in packing theory or statistical mechanics. We will define both terms more precisely later in this chapter.

Generating point sets that are both uniform and irregular is hard if not impossible to do by construction. Deterministic methods based on number theoretic considerations (such as the Halton sequence from Chapter 2) come close but are still far from optimal. Optimization techniques, on the other hand, may be able to find favorable point arrangements but only after a significant investment in runtime. Using a tile-based approach is a solution to this problem: we can distribute points in an optimized way across several tiles and generate the full point set later by generating a tiling. If we can ensure that the points are arranged in such a way that they maintain their favorable properties across tile boundaries, we benefit from the qualities of an optimized distribution without having to sacrifice runtime.

This chapter introduces two optimization techniques that fulfill these requirements and produce highly uniform, irregular point sets that are tileable using corner tiles. The first method is based on capacity-constrained Voronoi diagrams where each Voronoi cell is of equal size. We will see that the resulting point sets are particularly suited for numerical integration. The second method optimizes a given set of points with respect to their mutual distances which guarantees that no two points are closer than a certain distance. We will see that these points are particularly good for image plane sampling due to their properties in the Fourier domain. Both methods yield arrangements that can be considered uniform and irregular and become efficient in a corner tile setting. Although both methods generalize to higher dimensions, we will solely concentrate on the two-dimensional case. The benefits of well-distributed points quickly diminish for higher dimensional integration problems and many problems such as image plane sampling are two-dimensional problems anyway.

In the following, we first define quantitative measures that allow us to categorize existing methods when discussing the background of point distributions in computer graphics in Section 4.2. Sections 4.3 and 4.4 present both of our optimization methods without yet considering the distribution across corner tiles. This is done in the following section which discusses how both methods can be incorporated into a corner tile construction procedure. We will evaluate the final point sets in the next chapter.

4.1 Quantitative Measures

We will be mostly interested in points distributed in the 2D unit torus $[0,1)^2$ (the unit square with periodic boundary conditions), in which the distance between two points $p = (x_1, x_2)$ and $q = (y_1, y_2)$ is measured using the toroidal norm

$$\|p - q\|_T := \sqrt{\sum_{i=1}^{2} (\min\{|x_i - y_i|, 1 - |x_i - y_i|\})^2}. \tag{4.1}$$

This norm is typically preferred over the common Euclidean norm because the spatial properties of point sets should be preserved when tiled across the domain of interest, e.g. from pixel to pixel during image plane sampling.

If X denotes a point set containing $n := |X|$ points, the geometrical relationship between those points can be analyzed in terms of their toroidal Delaunay triangulation $\mathcal{D}(X)$. Since the Euler characteristic of the torus is $\chi = 0$, this gives us fixed values for the number of vertices V, number of edges E, and number of faces F in $\mathcal{D}(X)$:

$$V = n, \quad E = 3n, \quad F = 2n, \tag{4.2}$$

because $\chi = V - E + F$ and $E = 3F/2$ for a toroidal triangulation.

Quantitative Uniformity Measures

We define the following three measures of uniformity for a set of points X:

$$\begin{aligned} d_x &:= \min_{y \in X \setminus \{x\}} \|x - y\|_T && \text{local mindist,} \\ d_X &:= \min_{x,y \in X, x \neq y} \|x - y\|_T = \min_{x \in X} d_x && \text{global mindist,} \\ \bar{d}_X &:= \frac{1}{n} \sum_{x \in X} d_x && \text{average mindist.} \end{aligned}$$

The *local mindist* d_x is the distance from a point x to its nearest neighbor in X, the *global mindist* d_X is the smallest separation between any two points in X, and the *average mindist* measures the overall spacing of the point set. The advantage of these measures is that they are general and not geared towards specific applications (e.g. numerical integration). The average mindist is also used in other research domains such as ecology where it characterizes populations [CE54].

A point set that is uniformly distributed has both a high global mindist and a high average mindist: a high d_X means that the points do not cluster anywhere and a high $\bar{d}_X$ means that the points are evenly spaced. The largest mindist is obtained if the points form a hexagonal lattice [Tót51]: $d_{\max} = (2/\sqrt{3}n)^{1/2}$. We generally report the distance measures relative to this maximum value

$$\delta_x := d_x/d_{\max}, \quad \delta_X := d_X/d_{\max}, \quad \bar{\delta}_X := \bar{d}_X/d_{\max}.$$

Note that this normalizes δ_X, i.e., $\delta_X \in [0, 1]$.

Quantitative Irregularity Measure

To quantify irregularity, we make use of a measure from packing theory and chemical physics called *bond-orientational order* [KTT00, TTD00]. It is defined as

$$\psi_X := \frac{1}{2E} \sum_{x \in X} \left| \sum_{y \in \mathcal{N}_x} e^{ik\theta_{x-y}} \right|,$$

where $\mathcal{N}_x$ is the set of neighbors of point x in $\mathcal{D}(X)$, θ_{x-y} is the angle of the edge ("bond") $x - y$ with respect to some arbitrary but fixed reference axis, i is the imaginary unit, and $k \in \mathbb{N}$. The normalization constant $1/2E$ is derived from the fact that each edge is visited twice during the summation.

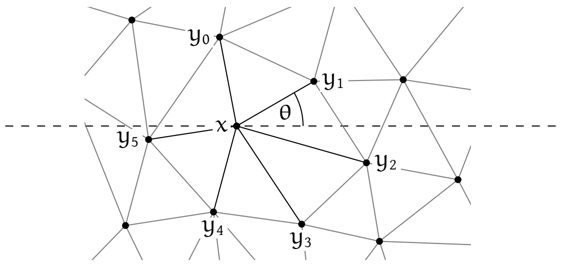


Figure 4.1: The computation of the bond-orientational order considers the geometrical neighborhood of a point x in the Delaunay sense.

Figure 4.1 illustrates how to interpret ψ_X: it measures the similarity of the polygon around a point x to a perfect k-gon and sums up these local measures for all points in X. A good choice for k is $k = 6$, measuring the similarity to a hexagon locally and the similarity to the hexagonal lattice globally. One important argument for this choice is that the hexagonal lattice is the global maximum/minimum for many optimization methods for point distributions, e.g. Lloyd's algorithm [Llo82]. In general, $\psi_X \in [0, 1]$ while $\psi_{\text{hex}} = 1$ for a perfect hexagonal lattice. In order for a point set to be irregular, ψ_X should not be too large; in our experience smaller than 0.6.

4.2 Background

Irregular sampling has been introduced to computer graphics in the form of stochastic sampling to solve the aliasing problem [DW85]. Using a Poisson distribution as sample points turns coherent aliasing such as moiré patterns into featureless noise. Cook [Coo86] observed that a Poisson distribution with the additional constraint that no two points are closer than a certain mindist reduces this noise while still preventing strong coherent aliasing. These distributions became known as *Poisson-disk* distributions and were conjectured to be the optimal distributions for image plane sampling [Mit91]. One supporting argument was that imaging problems in nature show similar solutions, e.g. the receptor distribution in the retina of primates [Yel83].

The reference algorithm to generate Poisson-disk distributions is the dart throwing algorithm [Coo86]. Sample positions are generated randomly but are accepted only if $\delta_x > t$ for all points and some threshold value $t > 0$. Since the algorithm is slow and not guaranteed to converge for a given number of points, many alternative methods have been proposed over the last 25 years. Cook himself proposed to roughly approximate the distribution by randomly shifting ("jittering") regular grid points by a small offset. Later methods aimed at accelerating the original dart throwing algorithm without compromising its properties, either by incorporating acceleration data structures such as a uniform grid [Jon06, DH06, WCE07, Wei08, GM09, EDP*11, JK11] or by opting for a

Generation Method	δ_X	$\bar{\delta}_X$	ψ_X	*Note*
Random	0.010	0.464	0.365	
Jittered Grid[a]	0.049	0.586	0.363	
Kernel Density[b] ($t = 1$)	0.426	0.864	0.436	I
Electrostatic Halftoning[c] ($\rho = 0.002$)	0.741	0.882	0.460	I
Dart Throwing[a] and variants[d]	0.750	0.805	0.367	
Best Candidate[e] and FPS[f]	0.751	0.839	0.396	
Capacity-Constrained	0.764	0.891	0.519	I
CVT Centroids[g] and methods using Lloyd's algorithm[h]	0.795	0.939	0.804	I, R
Electrostatic Halftoning[c]	0.826	0.952	0.877	I, R
Boundary Sampling[i]	0.829	0.862	0.400	
Low-Discrepancy[j]	0.903	0.920	0.663	D, R
Farthest-Point Optimization	0.930	0.932	0.426	I

[a] [Coo86] [b] [Fat11] [c] [SGBW10] [d] [LD08] [e] [Mit91] [f] [ELPZ97] [g] [DFG99] [h] [Llo82] [i] [DH06] [j] [GK09]

Table 4.1: Comparison of methods for uniform and irregular point sets with respect to the global mindist δ_X, the average mindist $\bar{\delta}_X$, and the bond-orientational order ψ_X. The notes mark (D)eterministic methods, (I)terative methods, and methods that converge towards (R)egular arrangements.

tile-based approach [SCM00, HDK01, ODJ04, LD06a, KCODL06, Ost07]. A survey by Lagae and Dutré reviews most methods existing up to 2008 [LD08].

We distinguish two main categories of algorithms: non-iterative algorithms that generate point sets in one pass and iterative algorithms that improve the arrangement of points in multiple passes. The relative mindist δ_X is a good measure to compare the uniformity of the methods while the bond-orientational order ψ_X is a good measure to compare their irregularity. Table 4.1 lists global mindist, average mindist, and bond-orientational order for existing methods that generate uniform but irregular point sets. The methods presented in this chapter are listed as well and are set in italics. Since most methods are non-deterministic, all measures are averages over an ensemble of 10 point sets containing 4096 points each. Figure 4.2 allows the visual inspection of some point sets.

Non-Iterative Algorithms

First, we observe that the classic dart throwing algorithm generates both uniform ($\delta_X > 0.75$) and irregular points ($\psi_X < 0.6$). The approximation by a jittered grid, however, is poor since points may get arbitrarily close together. The maximum mindist that is achievable by dart throwing is limited to $\delta_X \approx 0.77$. In general, this behavior does not change substantially for other methods that imitate the dart throwing procedure [LD08].

A non-iterative algorithm that is not based on a modified Poisson distribu-

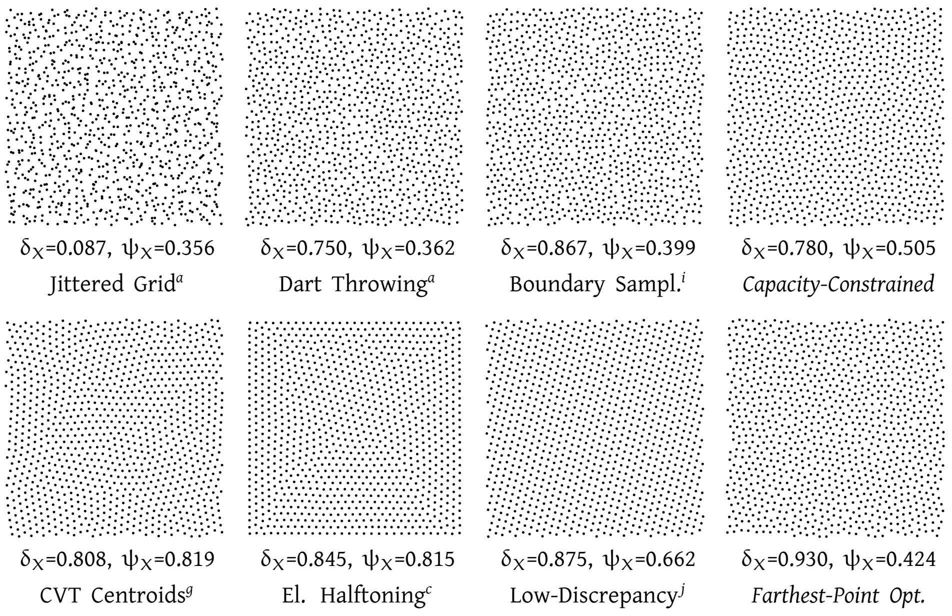


Figure 4.2: Gallery of example points sets along with their mindist δ_X and their bond-orientational order ψ_X. The number of points per set is 1024. The labeling follows Table 4.1. The electrostatic halftoning example was computed in a non-toroidal space.

tion was introduced by Eldar et al. [ELPZ97]. Given a few randomly distributed seed points, the algorithm deterministically adds points according to the "farthest point strategy" which chooses the location with maximum distance from all current points. Voronoi diagrams [OBSC00] can be used to obtain an $\mathcal{O}(n \log n)$ implementation of this algorithm. In general, the results are comparable to dart throwing with similar values of uniformity and irregularity. The algorithms by Mitchell [Mit91], Laine et al. [LSK*07], and Kanamori et al. [KSN11] can be considered equivalent to this approach.

Most non-iterative methods have difficulty achieving a mindist larger than approx. 0.75 because they cannot move points after they have been placed. This causes later points to be placed in suboptimal positions and is the main reason why iterative methods have been investigated in the first place. Exceptions are the boundary algorithm by Dunbar and Humphreys [DH06] with $\delta_X \approx 0.83$ and low-discrepancy point sets optimized for a high mindist [GHSK08, GK09] which can achieve mindists up to $\delta_X \approx 0.90$. In general, these point sets are very uniform but not irregular (also see Figure 4.2).

Iterative Algorithms

To improve the uniformity of Poisson-disk point sets, some methods employ an iterative algorithm by Lloyd [Llo82], commonly known as "Lloyd's method". The algorithm iteratively constructs the Voronoi diagram of the given point set and moves each point to the centroid of its Voronoi cell. Unfortunately, since the global minimum of the underlying energy function is a hexagonal lattice, an increase in uniformity ($\delta_X \approx 0.8$) is typically paid for with an increase in regularity ($\psi_X \approx 0.8$). Stopping the method before the arrangements become too regular is sometimes possible but in general unsatisfactory [LD08]. We will discuss Lloyd's method in more detail in the next section.

Two alternative iterative methods have been proposed in recent years. The method by Schmaltz et al. [SGBW10] models the points as charged particles with repelling forces. Simulating the movement of these particles results in uniform point distributions with a mindist comparable to Lloyd's method ($\delta_X \approx 0.83$), but it is also prone to producing locally hexagonal structures ($\psi_X \approx 0.88$). The same paper proposes a variant that breaks up these regularities by incorporating random movements, but this variant also decreases uniformity ($\delta_X \approx 0.74$).

The method by Fattal [Fat11] uses a similar approach where the point distribution is drawn from a Boltzmann-Gibbs statistical model of interacting particles. Controlling a temperature value allows to trade uniformity for irregularity and vice versa but achieving good values for both is difficult ($\delta_X \approx 0.43$ when $\psi_X \approx 0.44$).

4.3 Capacity-Constrained Points

The main challenge for iterative methods that generate uniform and irregular point sets is that they should reliably converge towards an equilibrium state that does not become regular again. Achieving such behavior in an automatic and reliable way is surprisingly difficult. The problem lies not so much in the requirement that the points should be uniform (most methods in Table 4.1 have at least a high average mindist) but in keeping them irregular. Our first method achieves this by introducing a constraint to the method by Lloyd [Llo82] that computes centroidal Voronoi tessellations.

Centroidal Voronoi Tessellations

First, we briefly define centroidal Voronoi tessellations for the space $\Omega := [0,1)^2$ with the toroidal norm (4.1). A *Voronoi tessellation* $\mathcal{V}(S)$ is a partition (tessellation) of the given space induced by a set of points $S \subset \Omega$, often called generator points or *sites*. The space is tessellated into $n := |S|$ distinct regions V_i, each corresponding to a site $s_i \in S$. Each region consists of all points x that are closer to the region's site s_i than to any other site $s_j \in S$, $i \neq j$. A Voronoi tessellation is called *centroidal* if each site s_i coincides with the centroid z_i of its Voronoi region. The centroids (centers of mass) are given by

$$z_i = \frac{\int_{V_i} x\rho(x)\,dx}{\int_{V_i} \rho(x)\,dx}, \tag{4.3}$$

where $\rho(x) > 0$ is a given density function defined on Ω. Centroidal Voronoi tessellations (CVTs) are related to the energy function

$$\mathcal{E}(S, \mathcal{V}) = \sum_{i=0}^{n-1} \int_{V_i} \rho(x) \|x - s_i\|_T^2 \, dx, \tag{4.4}$$

which is minimized only if $\mathcal{V}(S)$ is a CVT [DFG99, OBSC00].

For the current discussion, let us fix the density function to be constant, $\rho(x) := 1$. This provides a clearer picture of the centroid in the context of the first three moments of each Voronoi region V_i [SGB07]:

$$\lambda_i := \int_{V_i} dx \qquad \text{area (zeroth moment)}, \tag{4.5}$$

$$z_i := \frac{1}{\lambda_i} \int_{V_i} x \, dx \qquad \text{centroid (first moment)}, \tag{4.6}$$

$$I_i := \frac{1}{\lambda_i} \int_{V_i} x \otimes x \, dx \qquad \text{inertia (second moment)}, \tag{4.7}$$

where $u \otimes v$ denotes the outer product of two vectors $u, v \in \Omega$. The tensor I_i describes the inertia (second moment of area) of the Voronoi region V_i with respect to the origin. Using the parallel axis theorem [LL76], we can derive the inertia tensor with respect to the corresponding site s_i as

$$I_i' = \lambda_i \Big[I_i - z_i \otimes z_i + \|\Delta_i\|_T^2 E_2 - \Delta_i \otimes \Delta_i \Big].$$

Here, E_2 denotes the identity matrix and $\Delta_i := s_i - z_i$. This allows us to express the energy function $\mathcal{E}(S, \mathcal{V})$ as the sum of the Voronoi region's second moments with respect to their sites [SGB07]:

$$\mathcal{E}(S, \mathcal{V}) = \sum_{i=0}^{n-1} \int_{V_i} \|x - s_i\|_T^2 \, dx = \sum_{i=0}^{n-1} \operatorname{trace}(I_i'). \tag{4.8}$$

This relationship gives us a more intuitive understanding of why CVTs approximate the hexagonal lattice: The second moments become minimal for Voronoi regions that approximate circles and the densest arrangement of circles in two dimensions is given by the hexagonal lattice [Tót51].

Lloyd's Method

Lloyd's method [Llo82] generates CVTs from an input set of sites S by performing the following steps:

1. generate the Voronoi tessellation $\mathcal{V}(S)$ of S,

2. move each site s_i to the centroid z_i of its Voronoi region V_i,

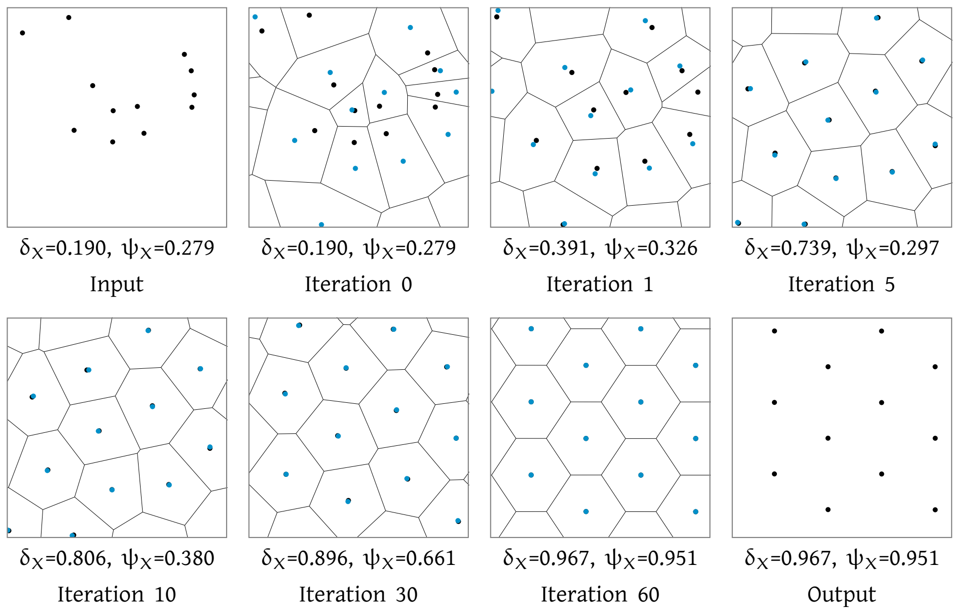

δ_X=0.190, ψ_X=0.279
Input
δ_X=0.190, ψ_X=0.279
Iteration 0
δ_X=0.391, ψ_X=0.326
Iteration 1
δ_X=0.739, ψ_X=0.297
Iteration 5
δ_X=0.806, ψ_X=0.380
Iteration 10
δ_X=0.896, ψ_X=0.661
Iteration 30
δ_X=0.967, ψ_X=0.951
Iteration 60
δ_X=0.967, ψ_X=0.951
Output

Figure 4.3: Evolution of a random point set with 12 points when optimized via Lloyd's method [Llo82]. The arrangement approaches a hexagonal lattice and thus becomes regular again. The black dots mark the original points, the blue dots mark the centroids of their Voronoi regions.

3. if the new sites in S meet some convergence criterion, terminate; otherwise return to step 1.

This algorithm can be considered a gradient descent minimization of the energy function $\mathcal{E}$ which has the partial derivative $\partial\mathcal{E}/\partial s_i = 2\lambda_i(s_i - z_i)$ [DFG99]. Thus, the common termination criterion is $\|\nabla\mathcal{E}(S, \mathcal{V})\| < \epsilon$ for a small $\epsilon > 0$. Each iteration has a time complexity of $\mathcal{O}(n \log n)$ [OBSC00].

Figure 4.3 illustrates Lloyd's method for a random set of 12 sites. It can be seen that the sites effectively repel each other and that the total distribution of sites gets more uniform. It is this implicit repelling force that is the main reason for the application of Lloyd's method for sampling tasks in computer graphics, not the fact that the method generates CVTs (there are faster methods, e.g. [LWL*09, YWLA11]). In particular, CVTs itself are undesirable since they are often not irregular (see the later iterations in Figure 4.3). In fact, for the bounded 2D case, Tóth [Tót01] proved the famous conjecture by Gersho [Ger79] that in globally optimal CVTs Voronoi regions indeed approach regular hexagons.

Capacity Constraint

Our algorithm is based on the method by Lloyd but constrains it such that it converges to an equilibrium state with an irregular distribution of sites. The constraint we use is built upon the concept of capacity [AHA98]. Again, let S denote a set of n sites that induce a Voronoi tessellation $\mathcal{V}(S)$ in the space Ω with the density function ρ. The *capacity* c_i of a site $s_i \in S$ with respect to its Voronoi region $V_i \in \mathcal{V}$ is defined as

$$c_i := \int_{V_i} \rho(x)\,dx.$$

We say that the distribution of sites adapts optimally to the density function ρ if the capacity of each site fulfills the *capacity constraint*

$$c_i = c^* := \frac{1}{n}\int_{\Omega} \rho(x)\,dx, \tag{4.9}$$

which intuitively gives each site the same "importance" in the distribution. For $\rho(x) := 1$ and the toroidal space $\Omega = [0,1)^2$, the constraint reduces to

$$c_i = \lambda_i = \frac{1}{n},$$

i.e., each Voronoi region must be of equal size $1/n$. We will see that this property is useful in a numerical integration setting.

Capacity-Constrained Voronoi Tessellations

An arbitrary distribution of sites S and its Voronoi tessellation $\mathcal{V}(S)$ usually do not fulfill a given capacity constraint for all sites. Instead, one has to compute a *capacity-constrained Voronoi tessellation* [Bal09]. A capacity-constrained Voronoi tessellation (CCVT) is a special Voronoi tessellation (power tessellation) that not only depends on a set of sites S but also on a corresponding set of non-negative capacities C. In our case (4.9), all elements $c_i \in C$ are assigned the same value c^* such that the CCVT fulfills the condition $\sum(\lambda_i - c_i)^2 = 0$.

To compute CCVTs, we utilize a discrete-space algorithm by Balzer and Heck [BH08] that is based on a discrete variant of the CVT energy function (4.4):

$$\mathcal{E}(P, A) = \sum_{i=0}^{m-1} \|p_i - A(p_i)\|_T^2,$$

where the density function ρ is represented by a finite set P of m sample points and $A : P \to S$ is a function which assigns each point in P to a site in S. The ordinary Voronoi tessellation would be formed by assigning each point in P to the closest site in S without any constraint; the capacity-constrained Voronoi tessellation is formed similarly but assigns each site s_i exactly $c_i \in C$ points, in our case $c^* = m/n$. Sites then iteratively swap assigned points as long as this minimizes $\mathcal{E}(P, A)$. Faster variants of this basic algorithm have been presented by Li et al. [LNW*10] and Frey et al. [FSG*11].

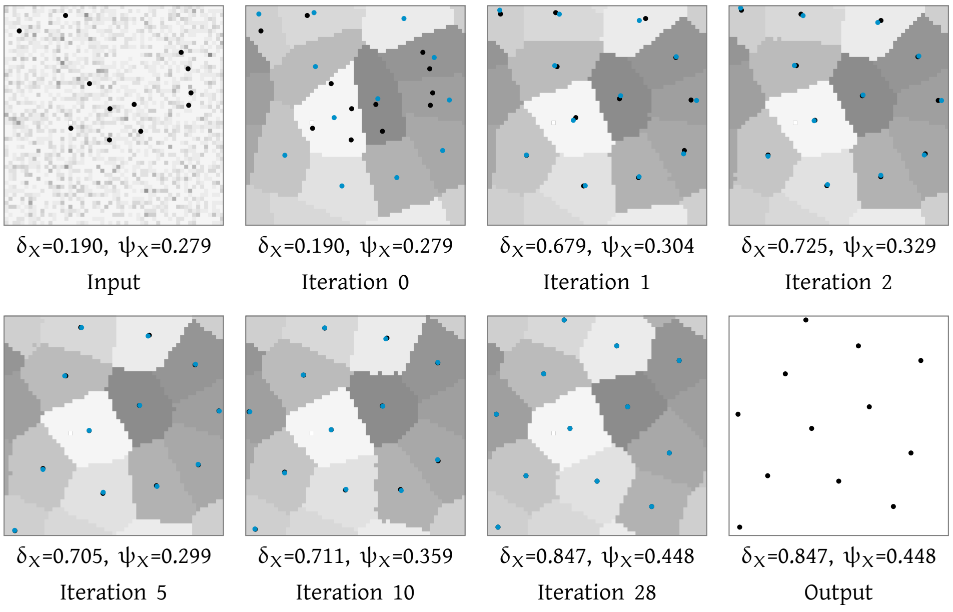

δ_X=0.190, ψ_X=0.279
Input
δ_X=0.190, ψ_X=0.279
Iteration 0
δ_X=0.679, ψ_X=0.304
Iteration 1
δ_X=0.725, ψ_X=0.329
Iteration 2
δ_X=0.705, ψ_X=0.299
Iteration 5
δ_X=0.711, ψ_X=0.359
Iteration 10
δ_X=0.847, ψ_X=0.448
Iteration 28
δ_X=0.847, ψ_X=0.448
Output

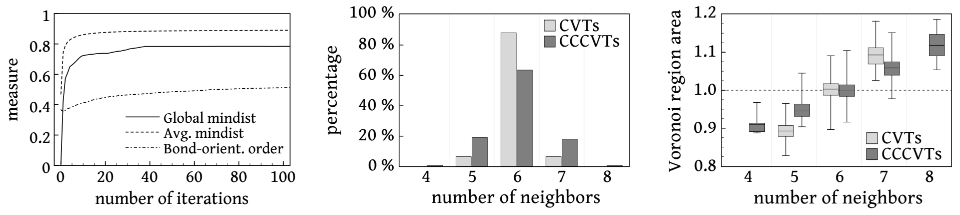
Figure 4.4: Evolution of a random point set with 12 points when optimized via our capacity-constrained method. The optimization converges to an equilibrium state with a distribution that is irregular. The resolution of the discrete space is exaggerated in this example for illustrative purposes.

Capacity-Constrained Method

Both Lloyd's method and CCVTs minimize the same energy function, but they do so in very different ways. Lloyd's method minimizes $\mathcal{E}$ by relocating the sites until the ordinary Voronoi tessellation yields a local energy minimum. CCVTs minimize $\mathcal{E}$ by optimizing the assignment A and keeping the sites fixed. We can simultaneously optimize the assignment A and the location of the sites S if we generate centroidal CCVTs by using a series of steps similar to Lloyd's method:

1. generate the capacity-constrained Voronoi tessellation $\mathcal{V}(S, C)$ of S using a set of capacities C conforming to our constraint (4.9),
2. move each site s_i to the center of mass of its points $p_i \in P$, $A(p_i) = s_i$,
3. if the new sites in S meet some convergence criterion, terminate; otherwise return to step 1.

These steps generate centroidal capacity-constrained Voronoi tessellations (CC-CVTs). Since the sites coincide with their centroids in this capacity-constrained CVT, we call the final distributions of sites *capacity-constrained* as well. The termination criterion we use is $\|\nabla\mathcal{E}(P, A)\| < \epsilon$ for a small $\epsilon > 0$. Each iteration has a time complexity of $\mathcal{O}(mn)$ [LNW*10].



(a) (b) (c)

Figure 4.5: (a) Development of the quantitative measures during the capacity-constrained optimization of 1024 random input points. (b) Histogram of the percentage of k-gons in the final Voronoi tessellations. (c) Normalized Voronoi region areas in the final Voronoi tessellations.

Comparison with Lloyd's method

Figure 4.4 gives an example using the same random set of sites that was optimized by Lloyd's method in Figure 4.3. First, it becomes apparent that the method converges very fast. For example, the sites are already evenly distributed after the first iteration because the capacity constraint immediately enforces Voronoi regions of equal size (δ_X = 0.679). The constraint also leads to a final arrangement of sites that can be considered irregular (ψ_X = 0.448) and that does not resemble the hexagonal arrangement generated by Lloyd's method. Figure 4.5(a) underlines this by depicting the development of the quantitative measures during the optimization of a random set with 1024 sites. To minimize side effects from a limited resolution m of the discrete space, we chose m very high (such that $m/n = 16384$) for most results in this thesis.

One explanation for the effectiveness of our constraint can be derived from the energy function (4.8) which we saw becomes minimal for Voronoi regions that approximate circles, i.e., when the sites form a hexagonal lattice. Since the hexagonal lattice does not fit into the unit torus for an arbitrary number of sites, CVTs approach the global minimum only locally by large patches of hexagons with a few non-hexagons in between (see Figure 4.6 left). Furthermore, a region that forms a k-gon is less optimal with respect to $\mathcal{E}$ than a region that forms a $(k + 1)$-gon with the same area. Since our capacity constraint enforces Voronoi regions of the same area, the energy can no longer benefit from area differences between k-gons and $(k + 1)$-gons and thus it becomes less likely that larger patches exclusively consist of hexagons (see Figure 4.6 right).

These observations are supported by the other two plots in Figure 4.5 for which we analyzed an ensemble of 10 distributions with 1024 sites each. Figure 4.5(b) shows a histogram of the percentage of k-gons present in the final (ordinary) Voronoi tessellations and Figure 4.5(c) shows a boxplot of their (normalized) areas. The CVTs consist of 87.8% hexagons, 6.1% pentagons, and 6.1% heptagons. The pentagons are significantly smaller than the hexagons which in turn

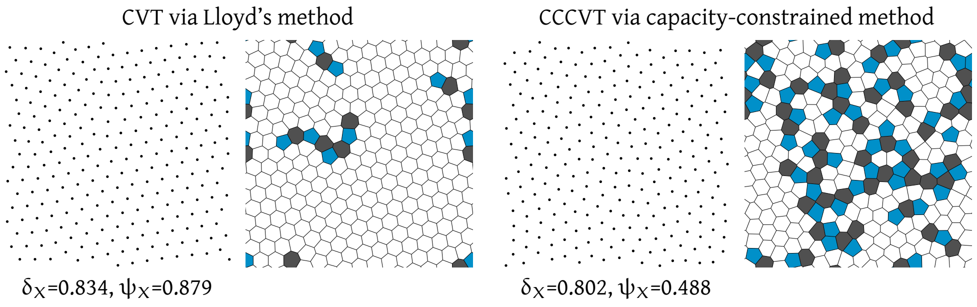
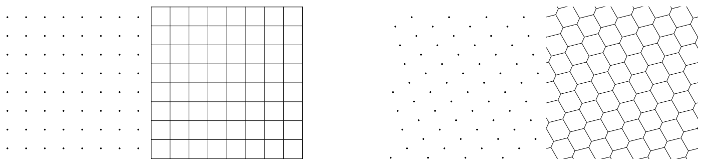

CVT via Lloyd's method
CCCVT via capacity-constrained method
δ_X=0.834, ψ_X=0.879
δ_X=0.802, ψ_X=0.488

Figure 4.6: Color-coding the Voronoi regions reveals the distribution of k-gons across the tessellation. Regions with $k < 6$ are filled blue, regions with $k > 6$ are filled black, and hexagonal regions are filled white.

are significantly smaller than the heptagons. The CCCVTs, on the other hand, show a distribution of k-gons that is more heterogenous with 69.6% hexagons, 15.5% pentagons, 14.7% heptagons, and a small fraction of quadrilaterals and octagons. Their areas are more uniform as well, with those of the pentagons and heptagons closer to 1. Note that these areas are measured with respect to the ordinary Voronoi tessellation of the given sites, not with respect to the capacity-constrained Voronoi tessellation (for which the areas are identical).

In the given examples, the initial distribution of sites was always irregular (random, to be exact) with $\psi_X = 0.365$. This irregularity is largely maintained during the optimization using the capacity-constrained method but almost fully disappears when Lloyd's method is used. We have found this to be case for all input point sets that are irregular. On the other hand, if the initial points are (even partly) regular, both methods tend to leave these areas undisturbed, often because the corresponding Voronoi regions are already beneficial with respect to the energy $\mathcal{E}$. In particular, some configurations are invariant to both methods because they already represent CVTs with Voronoi regions of equal size, e.g. points from the Cartesian grid or Rank-1 lattices [Dam09]:



Such configurations represent local minima of the energy function $\mathcal{E}$ but are typically not stable, i.e., they are unlikely to emerge from the optimization of irregular input sites [LWL*09]. We will further evaluate the final point sets in the next chapter.

4.4 Farthest-Point Optimization

Our capacity-constrained method generates irregular point distributions that are very uniform but their global mindist is not much higher than previous methods (recall Table 4.1). Methods explicitly based on repelling forces [SGBW10, Fat11] are able to achieve higher mindists but only for the price of (re-)emerging regularities. In fact, it was conjectured by Lagae and Dutré [LD08] that $\delta_X < 0.85$ in order to avoid regular configurations. The method presented in this sections shows that this conjecture is not correct.

Our new optimization procedure is explicitly designed to improve a point set's global and average mindist. It does so by iteratively enlarging the minimum distance between points monotonically until convergence. Since it can be interpreted as an iterative version of the farthest point strategy introduced by Eldar et al. [ELPZ97], we call the point sets *farthest-point optimized* and our method *farthest-point optimization* (FPO). We will see that the resulting point sets have excellent spectral properties and a significantly higher minimum distance than previous methods. And unlike other iterative methods, our procedure does not converge towards (locally) regular patterns. In addition, the close connection between farthest points and Delaunay triangulations permits a very efficient implementation that operates in continuous spaces and requires only $\mathcal{O}(n \log n)$ per full iteration. The principal is similar to the void-and-cluster method in halftoning [Uli93b, ABS99] where a binary mask is optimized in discrete space to generate uniform halftone dither arrays.

In the following, we first present the main algorithm and discuss its runtime complexity and convergence. We then discuss a variant of the main algorithm that runs in only $\mathcal{O}(n)$ per full iteration and gives results of the same quality.

Main Algorithm

The basic algorithm is very simple: Each step takes a single point from a given point set X and attempts to move it to a new position that is as far away from the remaining points as possible, i.e., the *farthest point*. One full iteration consists of moving each point in X once. As we will see, this iteration scheme converges, and each full iteration increases the average mindist $\bar{\delta}_X$.

In general, the farthest point f_Y for a set of points Y is the center of the largest circle that can be placed in the domain under consideration without covering any of the points in Y. This largest empty circle can be computed efficiently using the Delaunay triangulation of Y, $\mathcal{D}(Y)$, where it corresponds to the largest circumcircle of the triangles in $\mathcal{D}(Y)$. An equivalent formulation in terms of the Voronoi diagram of Y was used by Eldar et al. [ELPZ97].

In our case, to move a point x, we need to inspect the Delaunay triangulation (DT) of the remaining points $X\backslash\{x\}$. Instead of calculating the full DT for each point x, we build a full DT once and update it dynamically during the iteration: Before we move x, we remove it from the DT, inspect the remaining triangles to

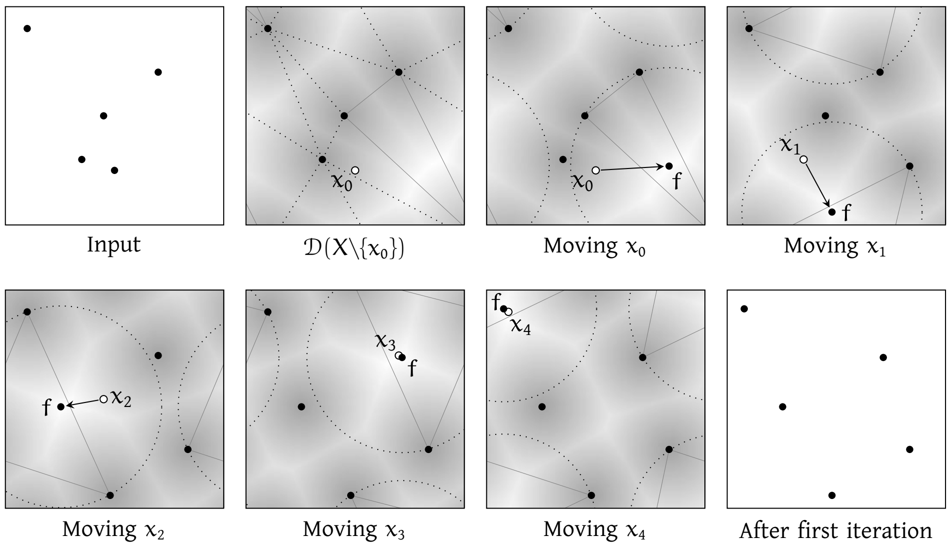


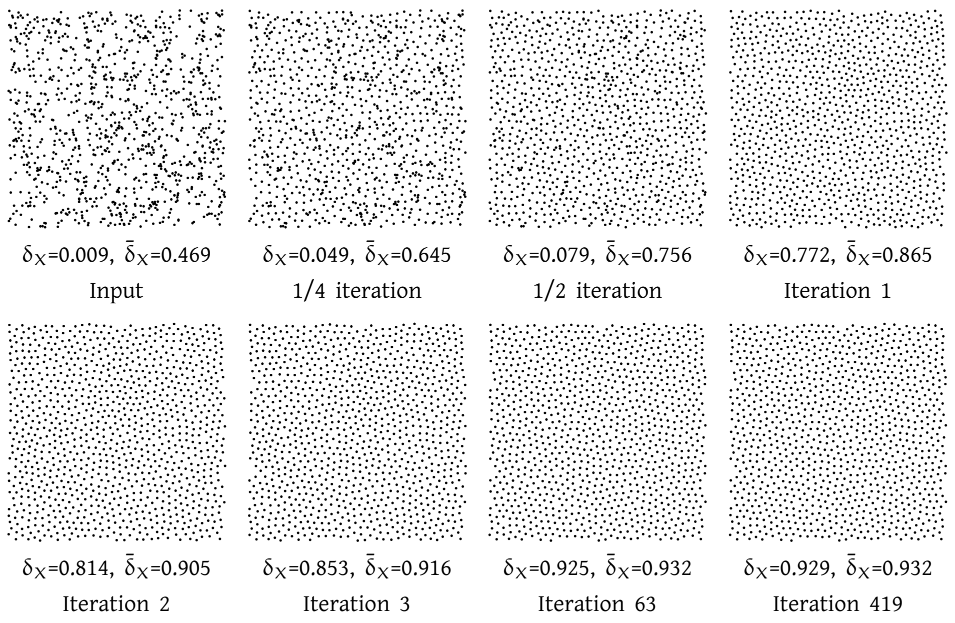
Figure 4.7: Illustration of one full iteration applied to 5 points in the unit torus. The grayscale image in the background represents the distance map of $X\backslash\{x_i\}$, the dotted circle is the largest empty circle, and the highlighted triangle the corresponding face in the Delaunay triangulation of $X\backslash\{x_i\}$.

find the farthest point f, and finally reinsert f as a new point into the DT. The full algorithm can be formulated as follows:

```
FARTHEST-POINT-OPTIMIZATION($X$)
 1  $\mathcal{D} = $ DELAUNAY($X$)
 2  **repeat**
 3      **foreach** vertex $x$ **in** $\mathcal{D}$
 4          $(f, r_{max}) = (x, d_x)$
 5          DELAUNAY-REMOVE($\mathcal{D}$, $x$)
 6          **foreach** $t$ **in** $\mathcal{D}$
 7              $(c, r) =$ center and radius of t's circumcircle
 8              **if** $r > r_{max}$
 9                  $(f, r_{max}) = (c, r)$
10          DELAUNAY-INSERT($\mathcal{D}$, $f$)
11  **until** converged
12  **return** vertices of $\mathcal{D}$
```

We make sure that a point is only moved to a new position if its new local mindist, namely r_{max}, would be larger than its old local mindist d_x; otherwise, we simply reinsert it at its old position.

Figure 4.7 illustrates how this procedure successively distributes five points $X = \{x_0, \dots, x_4\}$ in the unit torus. The first panels show how the target position for the first point x_0 is chosen: We search for the triangle in $\mathcal{D}(X\backslash\{x_0\})$ that has



δ_X=0.009, $\bar{\delta}_X$=0.469
Input
δ_X=0.049, $\bar{\delta}_X$=0.645
1/4 iteration
δ_X=0.079, $\bar{\delta}_X$=0.756
1/2 iteration
δ_X=0.772, $\bar{\delta}_X$=0.865
Iteration 1
δ_X=0.814, $\bar{\delta}_X$=0.905
Iteration 2
δ_X=0.853, $\bar{\delta}_X$=0.916
Iteration 3
δ_X=0.925, $\bar{\delta}_X$=0.932
Iteration 63
δ_X=0.929, $\bar{\delta}_X$=0.932
Iteration 419

Figure 4.8: Farthest-point optimization of a random point set with 1024 points. Both the global mindist δ_X and the average mindist $\bar{\delta}_X$ increase rapidly during the optimization. Still, the point set remains irregular ($\psi_X = 0.416$ after the last iteration).

the largest circumcircle and move x_0 to the circle's center. The distance map in the background indicates that this is indeed the farthest point. We proceed in the same way for $x_1, \ldots, x_4$ as shown in the remaining panels. The distribution has become much more uniform after only one full iteration.

It is easy to see that this farthest-point optimization always converges and yields arrangements with a high average mindist $\bar{\delta}_X$. The key observation is that moving a point x to the farthest point of $X \backslash \{x\}$ maximizes, by definition, its local mindist δ_x. In the worst case, no better position can be found and x remains at its old position. However, because $\bar{\delta}_X \propto \sum \delta_{x_i}$, the average mindist must increase during a full iteration, so the optimization can never return to a previous point distribution or get stuck in cyclic configurations. We stop the iteration as soon as the increase of $\bar{\delta}_X$ falls below a threshold ϵ, i.e., as soon as $\bar{\delta}_X^{\text{new}} - \bar{\delta}_X^{\text{old}} < \epsilon$. This must happen eventually since $\bar{\delta}_X$ is bounded for points in the unit torus. Convergence is fast enough that we can use the machine precision for ϵ.

For the global mindist we have $\delta_X = \min \delta_{x_i}$, so we are only guaranteed that it is non-decreasing. It is easy to construct point sets where δ_X remains constant for several iterations. But δ_X is strictly increasing as long as all points are still moving. For randomly distributed point sets we found this to be always the case.

In this case of random seed points, farthest-point optimization converges towards distributions with a mindist $\delta_X \approx 0.93$. A few intermediate steps during the optimization of 1024 points are shown in Figure 4.8. Since convergence becomes slower as we approach the maximum, we have found it useful to stop the iteration earlier. In our experience, a threshold of $\delta_X = 0.925$ is a good compromise between high-quality results and reasonable computation times. We will study the convergence empirically later on.

Even though most input point sets converge towards irregular arrangements, some stable configurations are regular, for example points from the hexagonal lattice or a honeycomb pattern. In these cases, no point can be moved to a position that is "farther" away from the remaining points. This is similar to the behavior of the Voronoi-based optimization techniques from the previous section but with one important difference: If there are defects in the regular arrangements, FPO quickly breaks up the regularity. One reason is that the method can move points over longer distances than the previous techniques since it operates globally, not locally. In this sense, FPO doesn't actively randomize its input, but it amplifies irregularities. This intuitively explains why the algorithm does not converge towards regular arrangements.

Runtime Complexity

We now consider the runtime complexity of the inner loop in our pseudo-code FARTHEST-POINT-OPTIMIZATION. If we denote the average *degree* of a point (i.e., its average number of neighbors in the triangulation) by g, the runtime of lines 4–10 can be broken down as follows:

4: $\mathcal{O}(g)$ since we have to inspect the neighbors of x to determine d_x.

5: between $\mathcal{O}(g)$ and $\mathcal{O}(g^2)$, depending on the algorithm used [Dev02].

6–9: $\mathcal{O}(n)$ since there are $2n$ triangles in $\mathcal{D}(X)$ as shown in Equation (4.2).

10: $\mathcal{O}(g)$ if we already know the triangle that contains the point. Otherwise, between $\mathcal{O}(\sqrt{n})$ and $\mathcal{O}(\log n)$, depending on the algorithm used to locate the triangle [DLM04].

We assume that $g = \mathcal{O}(1)$ which is the expected complexity for bounded 2D Delaunay triangulations [Dev02]. In this case, the overall runtime is $\mathcal{O}(n)$ for a single movement and $\mathcal{O}(n^2)$ for a full iteration. Two algorithmic improvements allow us to push this down to approximately $\mathcal{O}(n \log n)$ per full iteration.

First, we can speed up the process of inserting the farthest point f into the triangulation. In our experience, f almost always lies either inside the triangle t corresponding to the largest empty circle, or at least close to it. (This could already be seen in Figure 4.7.) Since we know t from lines 6–9, locating and inserting f can be done in approximately constant time.

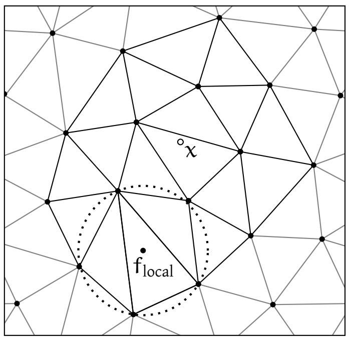
Second, we can speed up the search for the farthest point by using a binary search tree to keep track of the largest empty circle. This lets us find the farthest point in $\mathcal{O}(\log n)$, but increases the time required for lines 5 and 10 also to

$\mathcal{O}(\log n)$, since structural changes to the triangulation must be reflected in the tree. Taken together, this means that the running time is dominated by the tree operations, and the time required for a full iteration is $\mathcal{O}(n \log n)$.

Local Farthest-Point Optimization

This final $\mathcal{O}(n \log n)$ algorithm is efficient, but since the tree operations must be intertwined with the update operations of the Delaunay triangulation, its implementation is a little involved. As an alternative we can use the following variant that only requires $\mathcal{O}(n)$ per iteration but converges more slowly.

The idea behind this modified algorithm is to simplify the search for the farthest point. When moving a point x, we do not attempt to determine the *largest* empty circle but contend ourselves with a *large* empty circle in the neighborhood of x. In other words, instead of checking the circumcircle of all triangles in $\mathcal{D}(X\backslash\{x\})$, we restrict the search to a subset $T \subset \mathcal{D}(X\backslash\{x\})$ that is in some sense "close" to x. If the expected size of T is independent of n, each point can be moved in constant time.

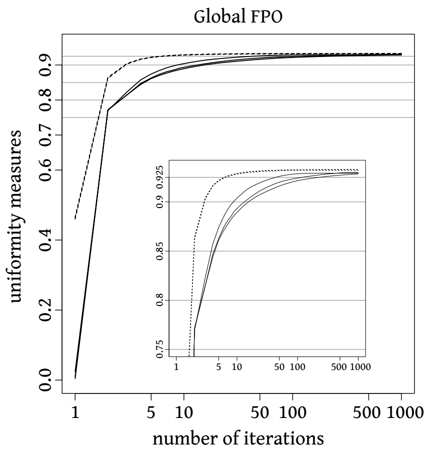


There are many strategies for choosing T. In our experience, the choice does not influence the quality of the resulting point sets, only the number of iterations required until convergence. Here, we discuss the one that has proven to be a good compromise between iteration and convergence speed: we include in T all triangles that are incident with the neighbors of x in $\mathcal{D}(X)$ (see embedded figure). Since there are $\mathcal{O}(g^2)$ such triangles, moving a single point can indeed be done in constant time. We will refer to this variant as *local FPO*, in contrast to the *global FPO* that constitutes the main algorithm.

Because our convergence argumentation only relies on the fact that the local mindist doesn't decrease, it remains valid in the case of the local FPO. However, since the local FPO moves points only locally, the mindist increases more slowly. Nevertheless, both methods converge towards point sets that are indistinguishable. In fact, once the points are sufficiently well distributed, local and global FPO are equivalent, since the farthest point of $X\backslash\{x\}$ is almost always located inside the hole that results from removing x. This suggests a hybrid algorithm that uses the global $\mathcal{O}(n \log n)$ algorithm for the first few iterations and then switches to the more efficient $\mathcal{O}(n)$ variant. In practice, this has turned out to be the fastest variant of farthest-point optimization.

Convergence

We studied the convergence speed of both global and local FPO empirically. The result is illustrated in Figure 4.9. Both δ_X and $\bar{\delta}_X$ increase rapidly at first and then



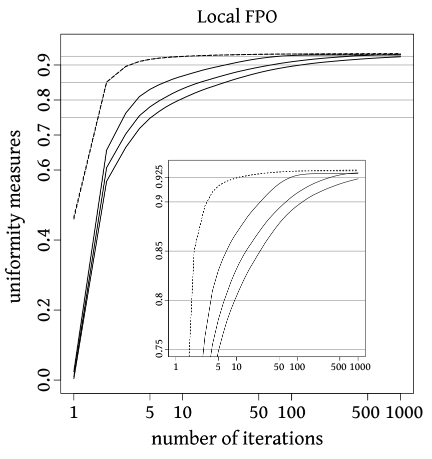


Figure 4.9: Average convergence of δ_X (solid lines) and $\bar{\delta}_X$ (dashed lines) for random sets of 512, 4096, and 32768 points, from left to right. The inset magnifies the region $0.75 \leqslant \delta_X \leqslant 0.95$.

converge more slowly towards a maximum around 0.932. The achieved maximum isn't the same for each set but consistently falls between 0.93 and 0.933. For both algorithms, the three curves for the average mindist (dashed lines) lie almost on top of each other. This means that convergence of $\bar{\delta}_X$ is mostly independent of the number of points, which underlines how effectively FPO distributes the points. The convergence of the global mindist (solid lines) depends more strongly on the input size, especially for the local variant.

In order to set these curves in context, the following table compares the number of iterations required to obtain well-distributed point sets with Lloyd's method and our capacity-constrained method from the previous section.

Method	$\delta_X =$ 0.75	0.775	0.8	0.825	0.85	0.875	0.9	0.925
[Llo82]	70	113	425*	-	-	-	-	-
CCCVT method	111	357*	-	-	-	-	-	-
Local FPO	3	4	6	8	14	27	64	352
Global FPO	1	2	2	3	4	6	13	118
	$\bar{\delta}_X =$ 0.75	0.775	0.8	0.825	0.85	0.875	0.9	0.925
[Llo82]	2	3	4	5	8	13	29	122
CCCVT method	2	2	2	4	10	50	414*	-
Local FPO	1	1	1	1	1	2	3	10
Global FPO	1	1	1	1	1	2	2	6

The results are averaged values from optimizing 10 sets of 4096 random points. The asterisk indicates that the mindist could not always be achieved. It becomes apparent that both FPO variants are far more effective than the other methods at spreading out the points: A handful of iterations are typically sufficient to obtain highly uniform point sets. The bond-orientational order of the final points of both local and global FPO is $\psi_X = 0.426$, i.e., they can be considered irregular.

4.5 Tile Assembly

Both our optimization methods are able to generate very uniform, irregular point sets but since they are iterative methods, they often have to be combined with a tile-based technique to make them beneficial. Corner tiles are a very good basis for this because of their simple shape and their ability to force continuity with all their neighbors (horizontal, vertical, and diagonal). There is one remaining challenge, however: How can we distribute the points across the corner tiles in such a way that they maintain their properties (capacity constraint, high mindist) even over tile boundaries? Furthermore, do the optimization methods still work without limitations despite possible boundary restrictions?

This section addresses these questions by presenting variants of both optimization methods that are compatible with a tile construction process for corner tiles. To put things in perspective, we briefly discuss previous tile-based methods for point distributions. We then sketch the construction process we use for our corner tiles. This is followed by the discussion of the variants of both optimization methods, the capacity-constrained approach and farthest-point optimization.

Background

Tiling a set of precomputed sample points across a given domain has already been considered in the early days of computer graphics by Dippé and Wold [DW85]. Although this approach is general and suits every point set in the unit torus, the strict periodicity is harmful for some applications where it yields artifacts such as moiré patterns. We will see this in more detail when we evaluate the spectral properties of the generated point sets in the next chapter.

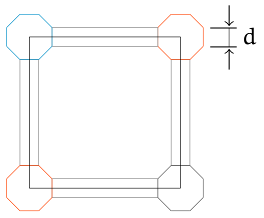
Shade et al. [SCM00] were the first to consider aperiodic tilings of point sets. They generate points across Wang tiles using an extended dart throwing algorithm that only accepts points if the Poisson-disk criterion is not violated across all possible neighbors. Hiller et al. [HDK01], Cohen et al. [CSHD03], and Kopf et al. [KCODL06] presented variants of this approach that employed Lloyd's method instead of dart throwing. The methods suffer from artifacts near tile boundaries, not least because of the multiple constraints along tile boundaries [LD08].

In 2004, the first method based on a type of tiles other than Wang tiles was presented by Ostromoukhov et al. [ODJ04]. They used a hierarchical tiling based on Penrose rhombs where point positions are given by the vertices of the tiling and used correction vectors based on Lloyd's method to hide regularities. Ostromoukhov later introduced an improved method based on polyominoes [Ost07] that shows less artifacts but that still relies on Lloyd's method.

Lagae and Dutré [LD05a, LD05b] were the first to consider explicit construction processes for distributing points across square tiles. They identified edge and corner regions on Wang tiles that should be partly shared among all tiles in the set, and interior regions that should be unique for each tile. This approach gave them better control over the quality of the final distributions, in particular in terms of artifacts near tile boundaries. An extension for corner tiles was presented in 2006 [LD06a] which we adopt and discuss in more detail in the following. Li et al. [LLLF08] presented a variant of this construction process that allowed rotation of corner and edge regions. Both Lagae and Dutré and Li et al. generate pure Poisson-disk distributions, optionally followed by Lloyd's method.

Construction Process

In order to fill corner tiles with well-distributed point sets that seamlessly tile the plane, one has to pay special attention to the tile boundaries. Points along these boundaries should be well-distributed with respect to all possible neighbors. This means in particular: points close to tile edges should fit all possible horizontal and vertical neighbors, and points close to tile corners should fit all possible horizontal, vertical, and diagonal neighbors. At the same time, each tile should contain as many points as possible that are unique to this tile as this lessens potential artifacts from repetitions. These observations are respected in the construction process of Lagae and Dutré [LD06a] which we briefly recapitulate here.



First, corner tiles are dissected into three types of regions: corner regions, edge regions, and interior regions. Corner regions and edge regions are shared among tiles while interior regions are unique to each tile. Each tile is constructed out of these regions according to its corner color combination (see embedded figure). The shape of each region is derived from the desired spatial properties of the final point sets. Here, a desired mindist d is a good criterion because it ensures that points in different edge regions don't influence each other. This leads to corner regions that are regular octagons with edge length d and edge regions that are rectangles of height d and width $1-(1+\sqrt{2})d$.

Since corner tiles have C corner colors, there will be C distinct corner octagons, and since corner tiles are not allowed to be rotated, there will be C^2 distinct edge rectangles for each orientation (horizontal and vertical), i.e., $2C^2$ edge rectangles in total. The number of interior regions equals the number of tiles in the tile set, e.g. C^4 for complete tile sets.

Figures 4.10 to 4.12 illustrate the full construction process for each tile. In this example, the final tiles should contain $n = 64$ points each.

First, the C corner octagons are constructed (see Figure 4.10). This involves generating a well-distributed point set of n points in the unit torus and simply cutting out the points that fall within the corner octagon.

With the corner octagons at hand, we can construct the edge rectangles by first placing the corresponding corner octagons (see Figure 4.11). The remainder of the domain is then filled with additional points such that the point density remains constant for the given number of points n. Only these additional points are then optimized; the corner points remain fixed. The additional points are also not allowed to enter the corner octagons. Again, we cut out those points that fall within the edge rectangle.

Finally, we can construct the full tile by first arranging corner octagons and edge rectangles according to the tile's corner color combination (see Figure 4.12). The tile interior is then filled with as much additional points as is needed to have exactly n points inside the tile's boundary. Again, only these additional points are optimized while not allowed to enter the corner and edge regions. The final tile is constituted by the points that fall within the tile's boundary (including those from the corner and edge regions). Since potential neighboring tiles share the "other" halves of the corresponding corner octagons and edge rectangles, their point sets fit seamlessly across their boundaries.

This construction process is simple enough to potentially work with various methods to distribute points across corner, edge, and interior regions. But while the construction of the corner octagons doesn't impose restrictions on the utilized generation method, the construction processes of the edge and interior regions are disallowing parts of the domain. In addition, some points have to be held fixed during the generation of the additional points while still influencing their placement. Generating points under these restrictions is comparatively easy with an extended dart throwing algorithm [LD06c, LLLF08] because the original algorithm is rejection-based anyway and as such can easily incorporate regions that are prohibited. In the following, we discuss how both our optimization methods are able to work under these restrictions, too, while not compromising their qualities in a substantial way.

Tiled Capacity-Constrained Points

Again, we want to optimize a given set of points X containing $n := |X|$ points in the unit torus. However, this time $m < n$ points in X are assumed to be *fixed* in the sense that they are not allowed to move from their initial positions. We assume without loss of generality that the fixed points are the first m points in X. In addition, let $D \subset [0,1)^2$ identify the region of the unit torus that the non-fixed points are not allowed to enter. We call this region the *disallowed region*. We are then interested in optimized points X under the restriction

$$x_i \in D, 0 \leqslant i < m \quad \text{and} \quad x_i \notin D, m \leqslant i < n. \tag{4.10}$$

We use the following steps to compute capacity-constrained points under this restriction.

1. generate the capacity-constrained Voronoi tessellation $\mathcal{V}(X, C)$ of X using a set of capacities C conforming to the capacity constraint (4.9),

2. for each movable point x_i, $m \leqslant i < n$, compute its center of mass z_i,

    (a) if $z_i \notin D$, move x_i to z_i,

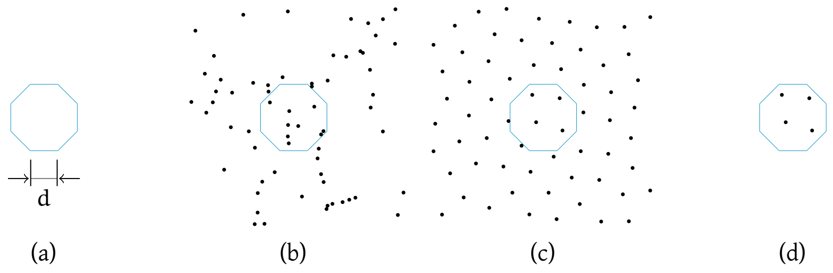


Figure 4.10: Constructing a corner octagon involves optimizing a point set over the unit torus and then cutting out the octagon with edge length d.

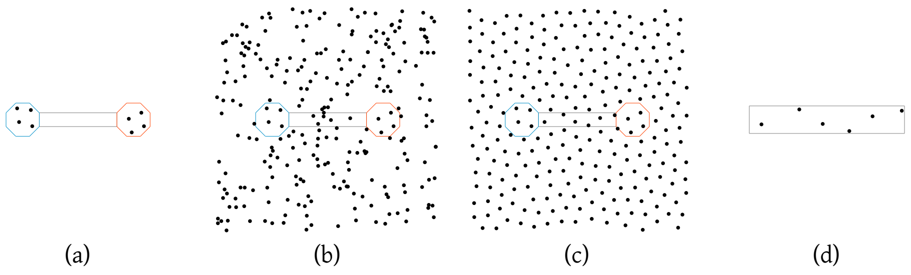


Figure 4.11: Constructing an edge rectangle involves (a) placing the two corresponding corners, (b) filling the unit torus with additional points, (c) optimizing those additional points, and (d) cutting out the rectangle.

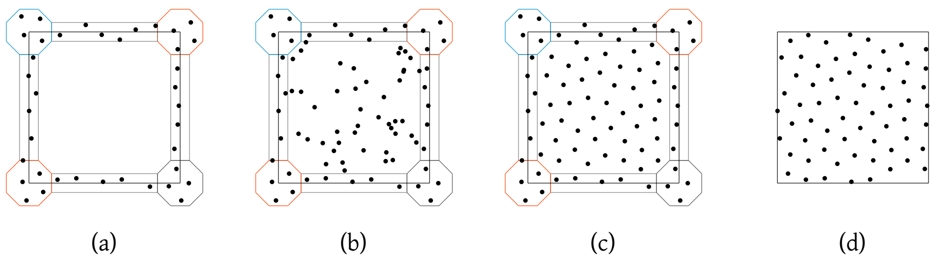


Figure 4.12: Constructing the full corner tile involves (a) arranging the corner octagons and edge rectangles according to the tile's corner color combination, (b) filling the tile interior with additional points, (c) optimizing those additional points, and (d) cutting out the final tile.

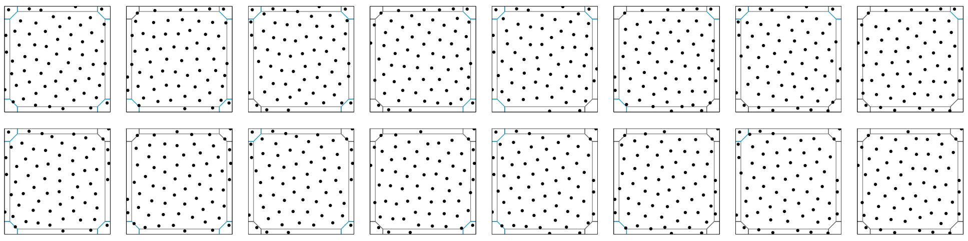


Figure 4.13: The complete corner tile set over two colors filled with $n_t = 64$ capacity-constrained points.

(b) otherwise, compute the point of intersection y_i of the line segment $\overline{x_i z_i}$ with D that is closest to z_i, and move x_i to y_i,

3. if the new points in X meet some convergence criterion, terminate; otherwise return to step 1.

These steps are a *restricted* variant of our capacity-constrained method from Section 4.3. We use the same termination criterion as for the unrestricted method.

Note that we compute the capacity-constrained Voronoi tessellation for all points in X but only move the non-fixed points with an index $m \leqslant i < n$. Also note that we only accept the centroid as the new position for a point if the centroid is outside the disallowed region D. Otherwise, we move the point to the intersection point of the line segment $\overline{x_i z_i}$ with D that is closest to the centroid. Such an intersection point must exist because in this case $x_i \notin D$ but $z_i \in D$. For complex disallowed regions (e.g. consisting of many non-connected components) there may be multiple intersection points in which case choosing the closest may lead to abrupt point movements across parts of D but for the "well-behaved" disallowed regions of the presented construction process, this is not an issue. We also didn't have a single case where the algorithm did not converge although we cannot guarantee that this holds for arbitrary choices of D.

Figure 4.13 shows an example set of corner tiles over two colors where each tile was filled with $n_t = 64$ capacity-constrained points using the aforementioned construction process. The desired mindist d was set to an average value for capacity-constrained points at a normalized value of 0.75 (recall Table 4.1). We will see that point sets resulting from tilings with such a set of corner tiles show similar properties to point sets directly generated by the original capacity-constrained method.

Tiled Farthest-Point Optimized Points

Similarly, the following variant of FARTHEST-POINT-OPTIMIZATION allows us to generate farthest-point optimized points under the restriction (4.10).

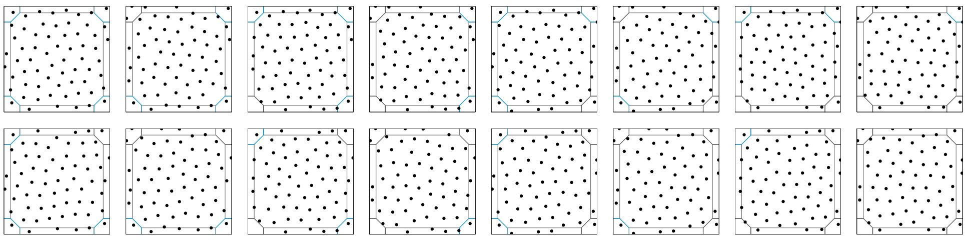


Figure 4.14: The complete corner tile set over two colors filled with $n_t = 64$ farthest-point optimized points.

```
RESTRICTED-FARTHEST-POINT-OPTIMIZATION($X, D, m$)
 1  $\mathcal{D} = $ DELAUNAY($X$)
 2  repeat
 3      foreach vertex $x_i$ in $\mathcal{D}$ with $i \geqslant m$
 4          $(f, r_{max}) = (x_i, d_{x_i})$
 5          DELAUNAY-REMOVE($\mathcal{D}, x_i$)
 6          foreach $t$ in $\mathcal{D}$
 7              $(c, r) = $ center and radius of $t$'s circumcircle
 8              if $r > r_{max}$ and $c \notin D$
 9                  $(f, r_{max}) = (c, r)$
10          DELAUNAY-INSERT($\mathcal{D}, f$)
11  until converged
12  return vertices of $\mathcal{D}$
```

Again, we use the same termination criterion as for the unrestricted method.

The main differences with respect to the unrestricted method are the adjusted loop in line 3 and the adjusted condition in line 8. The loop now only considers the vertices in the Delaunay triangulation which correspond to the non-fixed points in X (we assume the order of the points is preserved by $\mathcal{D}(X)$), and the condition now only accepts those circumcenters which do not fall into the disallowed region D. This means that we may not end up with the center of the largest empty circle as the new position for the current point, but only with the center of the largest empty circle with respect to the allowed domain. Note, however, that the method still converges because we still only accept circumcenters that improve a point's local mindist. In fact, because global FPO is able to move points not only locally, the convergence argument holds even for arbitrarily shaped disallowed regions D. We will see that this is also the main reason why FPO is a bit more robust in its restricted variant than the capacity-constrained method.

Figure 4.14 shows an example set of corner tiles over two colors where each tile was filled with $n_t = 64$ farthest-point optimized points using the construction process and RESTRICTED-FARTHEST-POINT-OPTIMIZATION. The mindist d was set to an average value for FPO points at a normalized value of 0.925.

Comparison with Non-Tiled Points

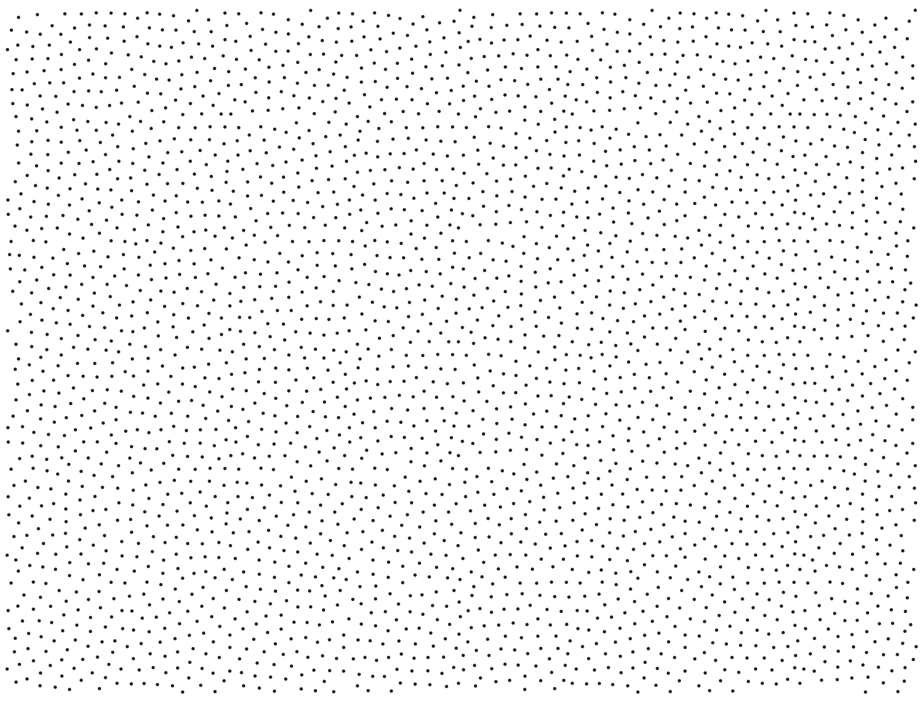
We now compare the unrestricted and restricted optimization methods when they are incorporated into the corner tile construction process. As a visual example, Figure 4.15 shows 8×6 stochastic tilings from complete corner tile sets filled with capacity-constrained points (top) and FPO points (bottom). Visually, the repeated corner octagons and edge rectangles are almost undetectable. Instead, the appearance is dominated by the tile interiors which repeat much less often and are unique to each tile. We will see, however, that the tile-based nature of the generated point sets cannot be hidden as easily from Fourier spectrum analysis which we perform in the next chapter.

For now, let us concentrate on the quantitative measures. Table 4.2 lists global mindist, average mindist, bond-orientational order, and CCVT energy for capacity-constrained point sets generated by complete corner tile sets with number of colors C, number of points per tile n_t, and total number of points n. By CCVT energy, we mean the normalized energy

$$\mathcal{E}'(X, \mathcal{V}) = n \sum_{i=0}^{n-1} \left(\lambda_i - \frac{1}{n} \right)^2,$$

where $\mathcal{V}(X)$ denotes the ordinary Voronoi tessellation of the point set X and λ_i the area of Voronoi region V_i as before. It is the energy being minimized by capacity-constrained Voronoi tessellations when the capacities conform to our constraint (4.9). All measures in Table 4.2 are averaged values over an ensemble of 10 tile sets generated from different seed points. We chose to use the packing solutions for complete corner tile sets as example tilings because this ensures (a) that the measures can be used unaltered (the packing solutions are toroidal), and (b) that each tile is used exactly once.

We see that the boundary restriction during tile construction affects both the global mindist and the CCVT energy but less the average mindist and the bond-orientational order. The global mindist suffers most because the restricted capacity-constrained optimization works only locally. For example, the method is not able to resolve situations where a point "wants" to move to its centroid inside the tile boundary but is not allowed to. This can lead to close-by positions of two points as is also detectable in Figure 4.15 (top). More important is the CCVT energy which is the main feature of capacity-constrained points and should be close to zero. We see that the energy decreases with the number of points per tile n_t but is largely independent of the number of colors (and hence the number of tiles). This is because the size of the boundary region is inversely proportional to the number of points per tile and thus its influence decreases when n_t increases. Still, for all examples the energy is of the same order of magnitude as the unrestricted (non-tiled) capacity-constrained points. Overall, a good compromise between tile set size and favorable CCVT energy is a tile set with $C = 2$ colors and $n_t = 256$ points per tile.



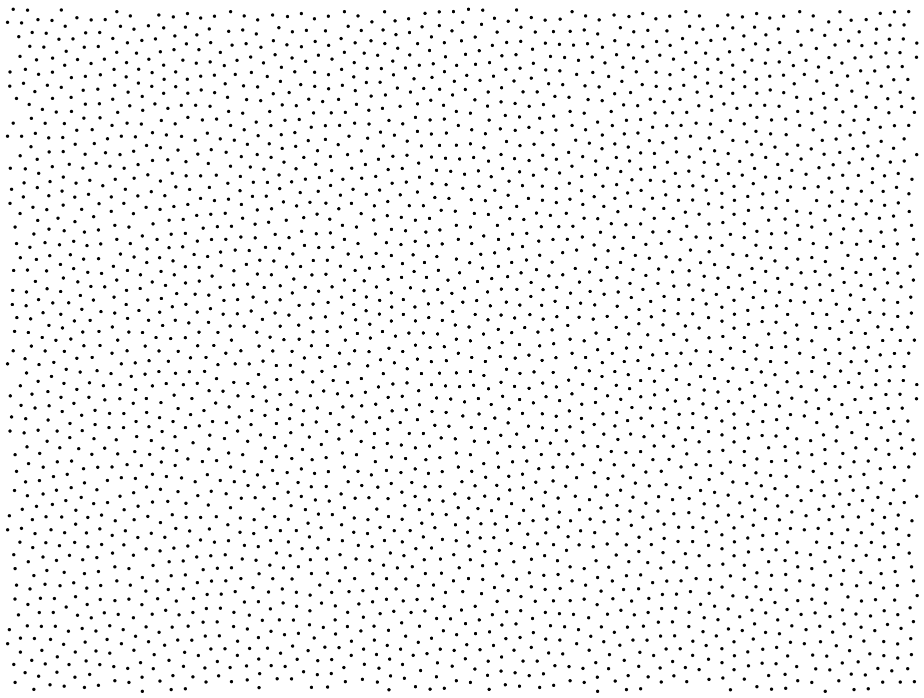


Figure 4.15: 8 × 6 stochastic tilings based on the tile sets from Figures 4.13 (top) and 4.14 (bottom). The repeated corner octagons and edge rectangles are almost undetectable.

Method	C	n_t	n	δ_X	$\bar{\delta}_X$	ψ_X	$\mathcal{E}'$ $(\times 10^{-3})$
Tiled	2	16	256	0.544	0.849	0.470	7.34
	3	16	1296	0.492	0.839	0.464	8.40
	4	16	4096	0.424	0.841	0.463	8.26
	2	64	1024	0.473	0.863	0.481	6.00
	3	64	5184	0.392	0.862	0.487	5.72
	4	64	16384	0.329	0.862	0.486	5.79
	2	256	4096	0.318	0.876	0.501	4.02
	3	256	20736	0.339	0.877	0.501	4.26
	4	256	65536	0.294	0.877	0.502	4.26
Non-Tiled			4096	0.764	0.891	0.519	1.49

Table 4.2: Quantitative measures and normalized CCVT energy for various capacity-constrained tilings based on complete corner tile sets with number of colors C, number of points per tile n_t, and total number of points n.

Table 4.3 shows the same measurements for point sets generated by complete corner tile sets filled with FPO points. Here, the boundary restriction mostly affects the global mindist and less the average mindist or the bond-orientational order. Generally, the global mindist gets larger as we allow more points per tile. This is for the same reason the CCVT energy became better in the case of the capacity-constrained points, namely that the influence of the boundary region with respect to the overall distribution decreases as n_t increases. At the same time, the global mindist gets smaller as we increase the number of corner colors. This is because more corner colors impose more restrictions on the overall distribution of points across all tiles, and hence increases the probability for situations where two points get relatively close. Still, the fact that even the restricted FPO is guaranteed to increase a point's local mindist leads to global minimum distances as high as $\delta_X = 0.91$. Overall, a good compromise between tile set size and global mindist is a tile set with $C = 2$ colors and $n_t = 64$ points per tile.

4.6 Conclusion

This chapter introduced two new optimization methods for uniform and irregular point distributions in the unit torus. Capacity-constrained point distributions cover the domain uniformly by having Voronoi cells of almost equal size and farthest-point optimized distributions are uniform in the sense of very high mutual distances. Both methods become efficient when combined with a corner-tile based approach without substantially limiting their favorable properties.

In this chapter, we also saw that the bond-orientational order is a good measure to characterize the irregularity of point distributions. This raises the ques-

Method	C	n_t	n	δ_X	$\bar{\delta}_X$	ψ_X
Tiled	2	16	256	0.847	0.906	0.410
	3	16	1296	0.818	0.903	0.398
	4	16	4096	0.795	0.903	0.409
	2	64	1024	0.892	0.919	0.409
	3	64	5184	0.868	0.919	0.411
	4	64	16384	0.844	0.919	0.408
	2	256	4096	0.910	0.926	0.417
	3	256	20736	0.891	0.926	0.417
	4	256	65536	0.873	0.926	0.416
Non-Tiled			4096	0.930	0.932	0.426

Table 4.3: Quantitative measures for various FPO tilings based on complete corner tile sets with number of colors C, number of points per tile n_t, and total number of points n.

tion if it would be a reliable termination criterion for Lloyd's method—a feature still missing in the computer graphics literature. Early tests indicate that it does work well to prevent the appearance of hexagonal arrangements but that this early termination also prevents good values for the global minimum distance.

On the other hand, the high minimum distance obtainable by farthest-point optimization raises another question for irregular point distributions: Are we able to increase the minimum distance any further without introducing regular structures? We cannot conclude that our points have indeed reached the limit for irregular distributions but results using other approaches [GHSK08, GK09, Dam09] at least indicate that (semi-)regular configurations are hard to avoid if one aims for an even higher minimum distance.

5

Evaluation of Tiled Point Sets

In the previous chapter, we presented two new optimization methods for point distributions, each of them utilized to distribute points across sets of corner tiles. We first evaluated the tiled point sets by considering quantitative measurements such as their mutual distances and their orientational order. In this chapter, we want to analyze and evaluate the point sets more thoroughly by considering both qualitative analysis and their behavior in practical applications such as image plane sampling and global illumination.

The qualitative analysis will be performed in two parts: in the spatial and in the frequency domain. In the spatial domain, we will consider a point set's radial distribution function which characterizes its spatial distribution by a 1D profile of pairwise point distances. As the radial distribution function has not yet found much application in computer graphics, we include a brief introduction in this thesis. In the frequency domain, we will analyze the point sets by their power spectrum which measures the power of each frequency component of a point set's Fourier transform. We will be interested in both a 2D visualization of the power spectrum and a radially averaged 1D profile. The importance of the 1D profile stems from the fact that it is directly related to the radial distribution function in the spatial domain.

To assess the quality of the point distributions in practical applications, we will consider two fundamental scenarios. The first is image plane sampling where the points are distributed across the image plane to yield images that are of low noise and free of coherent aliasing. In this scenario, the points' frequency characteristics are of particular importance. The second is physically-based rendering by ray tracing where the points determine ray directions to estimate the distribution of light in a virtual scene. This scenario involves the numerical approximation of the inherent lighting integrals and relies on the points' uniformity properties. In some examples of the latter scenario, we make use of one of three simple algorithms we included in the appendix. These allow us to further improve results in certain lighting situations.

5.1 Spatial Analysis

In order to characterize the spatial properties of a given point set X, we can ask the following question: What is the mean number of points in X that fall within a disk of radius r? Since our points are designed to be isotropic and statistically homogeneous [IPSS08], we can expect that, on average, we get the same answer no matter where we center this disk. This means that the answer should only depend on the disk radius and we could plot the answer as a 1D curve depending only on r.

Ripley's K Function

We are able to count the number of points in a disk D with radius r and its center being located at an arbitrary point c with the help of the indicator function by

$$n'(c, r) := \sum_{x \in X \setminus \{c\}} \chi_{D_{c,r}}(x). \tag{5.1}$$

Note that we do not count the point c should it happen to be part of the point set X. With this definition, an estimator for the mean number of points within a disk of radius r is given by

$$\bar{n}(r) := \frac{1}{n} \sum_{x \in X} n'(x, r). \tag{5.2}$$

That is we place a disk at each point in X and count the number of points within distance r. The following function is then known as *Ripley's K function* [Rip77]:

$$K(r) := E[\bar{n}(r)]/\rho, \quad r \geqslant 0,$$

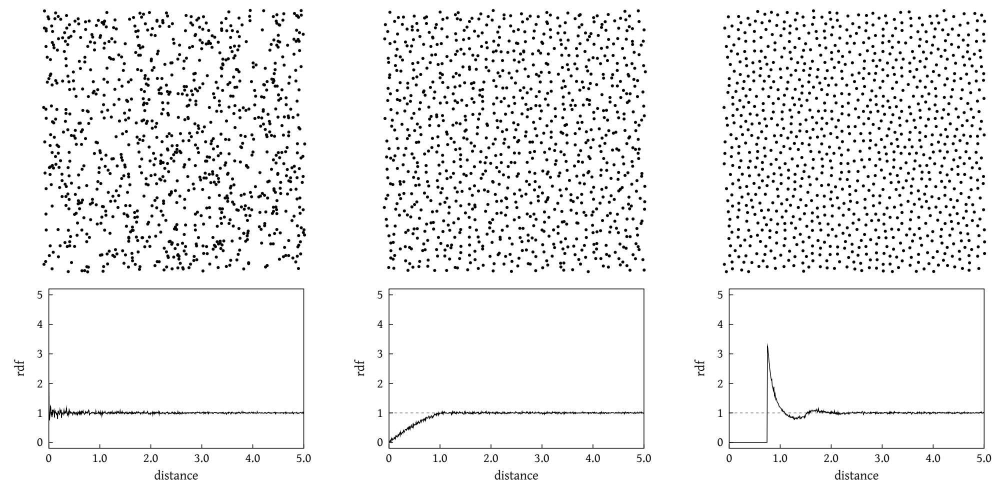
where E denotes the expected value and ρ the point density. In our cases of n points in the unit torus, it is simply $\rho = n$.

In order to estimate the behavior of a whole method (for example "dart throwing"), we can average K over k point sets generated by this generation process. This linearly reduces the variance of the estimator [Ros02]. For example, random point sets are generated by a Poisson process and it is easy to see that in this case $K(r) = \pi r^2$. This is analogous to having a Monte Carlo estimator for the area of a disk with radius r.

Radial Distribution Function

While Ripley's K function captures all the statistical information we are interested in, the resulting curves become easier to understand if we slightly transform it:

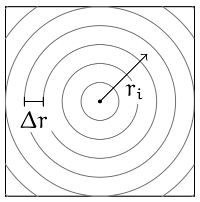
$$g(r) := \frac{K'(r)}{2\pi r}, \quad r \geqslant 0.$$



Random Jittered Grid Dart throwing

Figure 5.1: Example radial distribution functions for random, jittered grid, and dart throwing point sets.

This function is known as the *radial distribution function* (RDF) or *pair correlation function* in physics and spatial statistics [IPSS08]. It is proportional to the derivative of Ripley's K function but is typically easier to understand because it is not cumulative and approaches 1 as $r \to \infty$. In addition, for a Poisson process, we have $g(r) = 1$ which means we can easily observe the deviation from a set of points that are totally uncorrelated. The RDF also allows an intuitive probabilistic interpretation: The probability $\pi(r)$ of finding a point in distance r from another point is $\pi(r) = g(r)/\rho$.



With a few exceptions (such as the Poisson process), we usually do not have closed-form expression for the RDF of a specific generation process. Thus, to approximate g, we could numerically differentiate the estimate of Ripley's K function or, equivalently, partition the disks into concentric rings R_i of central radius r_i and width Δr. (See embedded figure.) We are then able to replace (5.1) and (5.2) with an estimator for the mean number of points within each ring:

$$n'(c, r_i) := \sum_{x \in X \setminus \{c\}} \chi_{R_{c,r_i}}(p) \quad \text{and} \quad \bar{n}(r_i) = \frac{1}{n} \sum_{x \in X} n'(x, r_i).$$

This is equivalent to setting up a histogram of the distances between all pairs of points and choosing a bin width equal to the width Δr of each ring. The choice of Δr is a delicate balance between smoothing and precision and recommended to be in the order of $\rho^{-1/2}$ [IPSS08].

It is interesting to note that a RDF does not uniquely identify a specific point generation process (i.e., two different processes can have the same RDFs) and

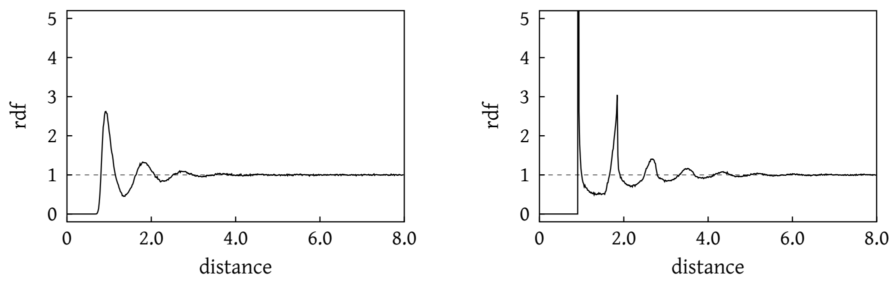


Figure 5.2: Radial distribution functions for untiled capacity-constrained and farthest-point optimized points.

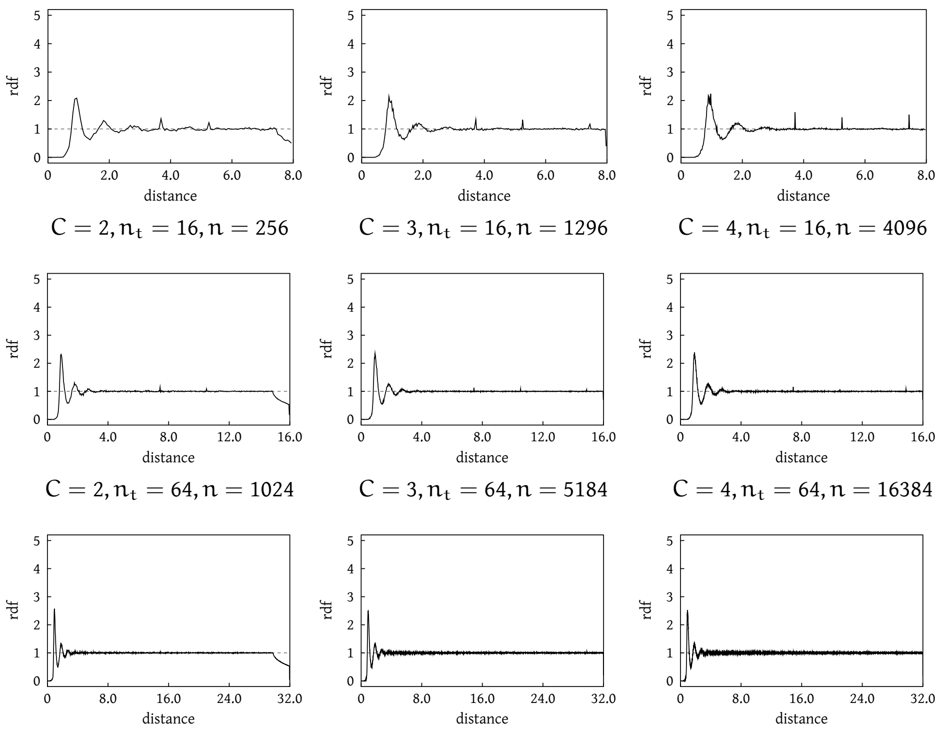
that it is easy to construct RDFs to which no corresponding point process exists [TS03, UST06]. A more complete introduction to the RDF and similar statistics can be found in the book by Illian et al. [IPSS08].

Figure 5.1 shows estimated RDFs over $k = 10$ sets of random, jittered grid, and dart throwing points with 4096 points each. For dart throwing, the minimum distance was set to a normalized value of $\delta_{\text{darts}} = 0.75$. We generally scale the distance axis by the same normalization factor of $(2/\sqrt{3}n)^{-1/2}$ so that we can directly read off this mindist. We see that the RDF for random points indeed approximates 1 while the other methods influence close distances, either only softly (jittered grid) or hard (dart throwing). We also see that all RDFs approximate 1 as $r \to \infty$ which implies that there is always a distance r_{corr} after which the points become uncorrelated.

Capacity-Constrained Points

Let's take a look at the RDF of our capacity-constrained points. Figure 5.2 (left) shows the RDF for plain (i.e., untiled) capacity-constrained points. We see that its profile is comparable to the dart throwing profile from Figure 5.1 in the sense that both are zero at first and then rapidly increase to a local maximum around $r = \bar{\delta}_X$. But while the transition for dart throwing point sets is sharp (stemming from the hard minimum distance constraint), the transition is smoother for the capacity-constrained points. This underlines the optimization characteristic of the capacity-constrained method in which points implicitly repel each other but without a hard distance constraint.

We also see that an increase of the probability that the points have a certain mindist leads to a decrease of the probability that they have a distance around 1.5 times their average mindist (at $r \approx 1.35$). This is a natural consequence of a high δ_X because points must roughly arrange at multiples of $\bar{\delta}_X$ if the mindist should be preserved. This correlation between points continues in an oscillating manner until $r_{\text{corr}} \approx 4$ after which $g(r) \to 1$.



$C = 2, n_t = 16, n = 256$ $C = 3, n_t = 16, n = 1296$ $C = 4, n_t = 16, n = 4096$

$C = 2, n_t = 64, n = 1024$ $C = 3, n_t = 64, n = 5184$ $C = 4, n_t = 64, n = 16384$

$C = 2, n_t = 256, n = 4096$ $C = 3, n_t = 256, n = 20736$ $C = 4, n_t = 256, n = 65536$

Figure 5.3: Radial distribution functions for tiled capacity-constrained points based on tile sets with C colors, n_t points per tile, and total number of points n.

Figure 5.3 shows the same function for tiled capacity-constrained points based on the complete corner tile sets from Chapter 4. The tilings are consistent with those used for the quantitative measures, i.e., they follow the packing arrangements with increasing number of colors C, number of points per tile n_t, and total number of points n. We see that the curves generally follow the profile of the non-tiled points but with one important difference: They show peaks at distances that correspond to the underlying tile grid, revealing the repetitions that occur at the tile boundaries. These peaks worsen with an increasing number of tiles (because boundary points are repeated more often) and better with an increasing number of points per tile (because boundary points are overwhelmed by interior points). For $n_t = 256$ they almost disappear, even for $C = 4$. Note that the distance ranges of the plots double from row to row in order to keep the peaks at the same position.

Farthest-Point Optimized Points

We now take a look at the RDF of farthest-point optimized point sets. Figure 5.2 (right) shows the RDF for plain (i.e., untiled) farthest-point optimized points. As

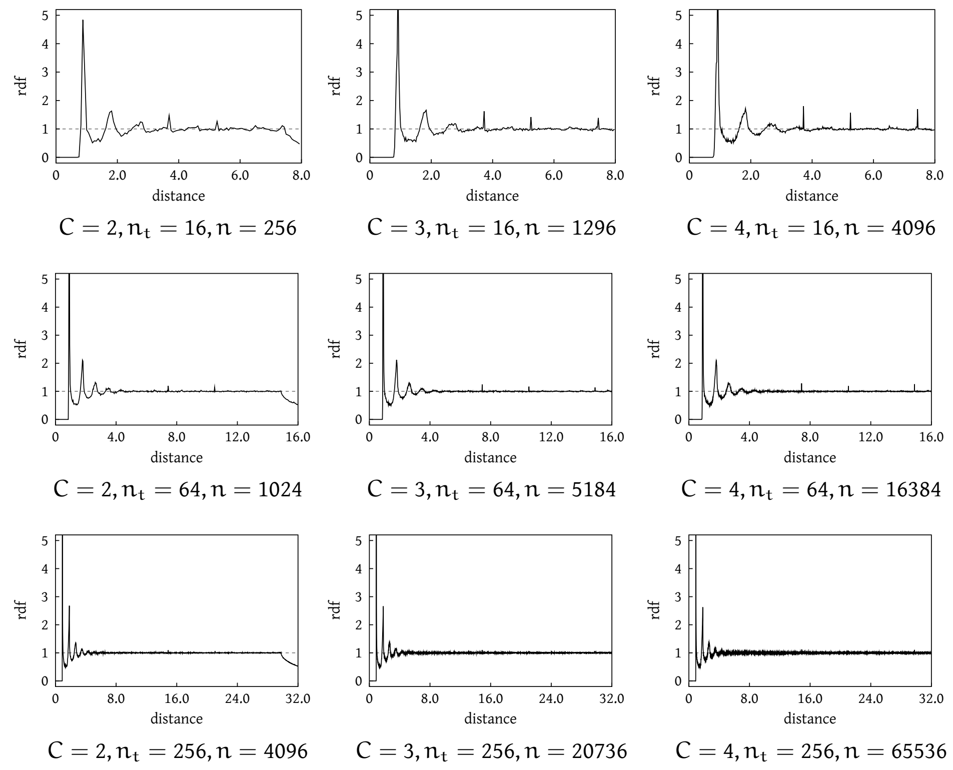


Figure 5.4: Radial distribution functions for tiled farthest-point optimized points based on tile sets with C colors, n_t points per tile, and total number of points n.

farthest-point optimization aims at optimizing a point's local mindist, the profile is also comparable to the dart throwing profile. In contrast to the RDF of capacity-constrained points, however, the transition at $r = \bar{\delta}_X$ is sharp and the local maximum significantly higher than for dart throwing or the capacity-constrained method ($g(\bar{\delta}_X) \approx 30.3$). This underlines the small variance of the points' local minimum distances, i.e., $\bar{\delta}_X \approx \delta_X$. We also see that g oscillates stronger than for the other methods and that $r_{\text{corr}} \approx 6$. This illustrates that not only the distance of each point to its direct neighbors is relatively constant but also the distance to its second and higher-order neighbors.

Figure 5.4 shows the radial distribution functions for tiled FPO points, generated by the corresponding corner tile sets as before. The behavior is very similar to the behavior of the tiled capacity-constrained points: Peaks at multiples of the tile grid show up until the ratio of the number of tiles to the number of points per tile is sufficiently small. Again, from the perspective of the RDF, a tile set with $n_t = 256$ points per tile is sufficient to hide the tile-based nature of the generation process for the considered tilings. Generally, the number of points per tile should surpass the number of tiles in the tiling at least by a factor of 16 for the peaks to disappear.

5.2 Spectral Analysis

Point sets with a uniform but irregular distribution have a characteristic energy distribution in the Fourier domain: If the points are widely spaced, the spectral energy is low in a circular disk around the origin, and if they are irregularly distributed, the energy varies smoothly outside this empty inner ring. Point sets with such a spectrum are known as *blue noise* patterns in computer graphics in an analogy to the corresponding color in the spectrum of visible light [Mit87, Uli88]. We will see that both methods presented in this thesis generate blue noise point sets.

In order to perform a proper Fourier analysis of a point set X, we can represent it as a superposition of Dirac δ-functions by $s(x) = \sum_X \delta(x - x_j)$. Its Fourier transform $\mathcal{F}\{s(x)\} = S(\omega)$ is then given by

$$\begin{aligned}\mathcal{F}\{s(x)\} &= \mathcal{F}\left\{ \sum_{x_j \in X} \delta(x - x_j) \right\} \\ &= \sum_{x_j \in X} \mathcal{F}\{\delta(x - x_j)\} \\ &= \sum_{x_j \in X} e^{-i\omega x_j} = S(\omega),\end{aligned}$$

where $\omega = 2\pi\nu$ denotes a real-valued spatial frequency. We also have to consider that many point generation methods are not deterministic (i.e. they yield different point sets for different seeds). From a signal processing point of view, we can interpret them as stationary stochastic processes of finite average power [Uli93a]. We can then characterize their spectral behavior by the corresponding power spectrum.

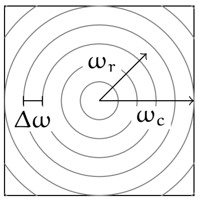
Power Spectrum

The *power spectrum* $\mathcal{P}$ of a stationary stochastic process is the Fourier transform of its autocorrelation function [PM96]. Since we typically don't know the autocorrelation function, we have to estimate the power spectrum, similar to estimating the radial distribution function during spatial analysis. We do this by averaging k *periodograms*. A periodogram is the squared magnitude of the Fourier transform. An estimator $\bar{\mathcal{P}}$ for the power spectrum is thus given by

$$\bar{\mathcal{P}}(\omega) := \frac{1}{kn} \sum_{i=0}^{k-1} \left|\mathcal{F}\{s_i(x)\}\right|^2,$$

where the s_i represent unique point sets generated by the process with n points each. In digital signal processing, this procedure of averaging periodograms is known as Bartlett's method [PM96]. In practice, we estimate the power spectrum from $k = 10$ periodograms and visualize it by rendering grayscale images using a logarithmic tone mapping. The tone mapping we use is $x \mapsto \log_2(1 + \alpha x)$ with $\alpha = 0.25$.

Radial Statistics



From the 2D power spectrum, we can derive two useful one-dimensional statistics, the *circularly averaged power spectrum* and the variance of this measurement, characterizing the *anisotropy*. For this purpose, we partition the spectrum domain into concentric rings R_{ω_r} of central frequency ω_r and width $\Delta\omega$, comparable to the partition during spatial analysis. ω_c denotes the critical frequency after which the radii exceed the power spectrum. Estimators for the 1D power spectrum P and the anisotropy A are then given by

$$P(\omega_r) := \frac{1}{|R_{\omega_r}|} \sum_{\omega_i \in R_{\omega_r}} \bar{\mathcal{P}}(\omega_i) \quad \text{and} \quad A(\omega_r) := \frac{V^2(\omega_r)}{P^2(\omega_r)},$$

where $|R_{\omega_r}|$ denotes the number of frequency samples within R_{ω_r} and $V^2(\omega_r)$ the squared variance of the spectrum samples, i.e.,

$$V^2(\omega_r) = \frac{1}{|R_{\omega_r}| - 1} \sum_{\omega_i \in R_{\omega_r}} \left(\bar{\mathcal{P}}(\omega_i) - P(\omega_i)\right)^2.$$

The anisotropy is usually plotted in decibels.

We see that radial distribution function and 1D power spectrum are measured in a similar fashion albeit in different domains. Both are obtained by circular averages in their respective domains and for a Poisson process, both take on the constant value 1. This is no coincidence. For isotropic point sets, both are connected by the equation [IPSS08]

$$P(\omega) = 1 + \rho\, \mathcal{H}\{g(r)\},$$

where $\mathcal{H}$ denotes the Hankel transform [Bra99] (Fourier transform of circularly symmetric functions). This relationship is also the reason why the power spectrum is used in other research domains where it is known as the *structure factor* [KTT00, TTD00, TS03]. Interestingly, these research domains are often interested in the opposite direction: to draw conclusions about the spatial properties of “points” (particles) by observations made in the frequency domain, for example through scattering experiments. Wei and Wang [WW11] recently derived this relationship independently in a two-dimensional variant.

To get an impression of these statistics, Figure 5.5 shows power spectrum, circularly averaged power spectrum, and anisotropy for the processes behind random, jittered grid, and dart throwing point sets. We see that random point

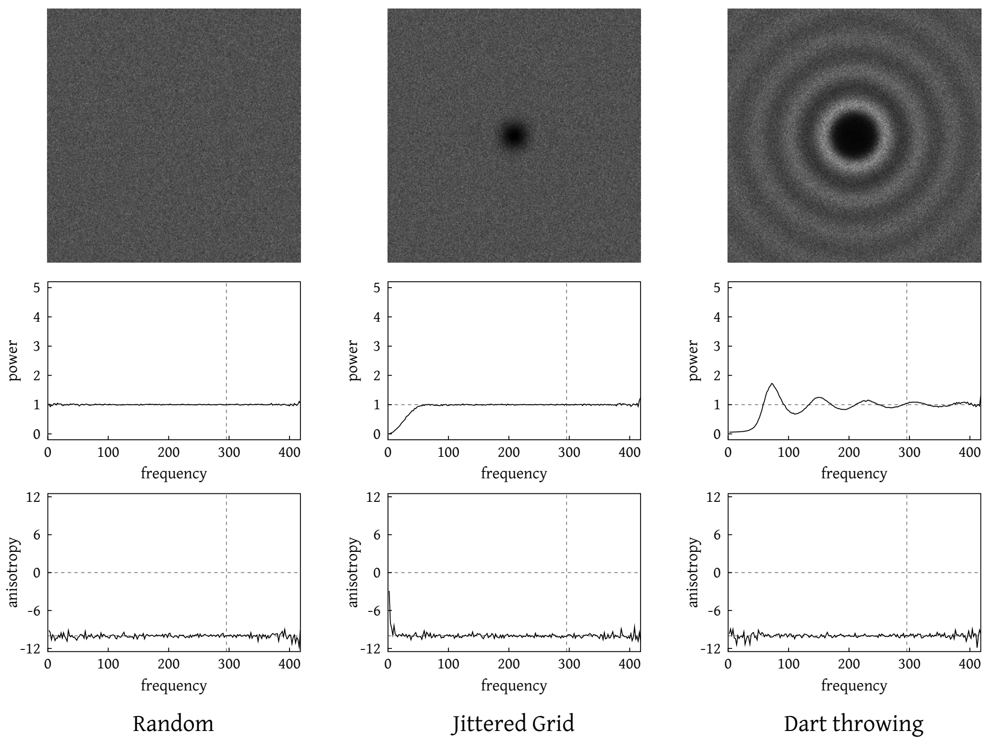


Figure 5.5: Example spectral analysis of random, jittered grid, and dart throwing point sets.

sets approximate a Poisson process with unit power, $P(\omega_r) \approx 1$, and no observable anisotropy, $A(\omega_r) \approx -k\,\text{dB}$. Generally, an anisotropy close to this level indicates good circular symmetry for the specific generation method. We also see that both jittered grid and dart throwing show the aforementioned blue noise characteristic to various degrees: low power for lower frequencies and a smooth (or no) oscillation around 1 for higher frequencies with no perceivable anisotropy.

In all of the examples in this thesis, the Fourier transform of each point set is sampled at the integer values in the range $[-w/d_{\max}, w/d_{\max}]$ where w indicates a window width. It is determined empirically at $w = 5$ as is the ring width at $\Delta\omega = 2$. Also note that the DC peak has been removed from all plots.

Capacity-Constrained Points

Figure 5.6 (top) shows the spectral results for plain capacity-constrained points. We see that the points exhibit the blue noise property with almost no energy in lower frequencies and anisotropy only around the DC peak. Compared to the power spectrum of a Poisson-disk process (“Dart throwing” in Figure 5.5), there is fewer energy in lower frequencies and the transition to the principal frequency

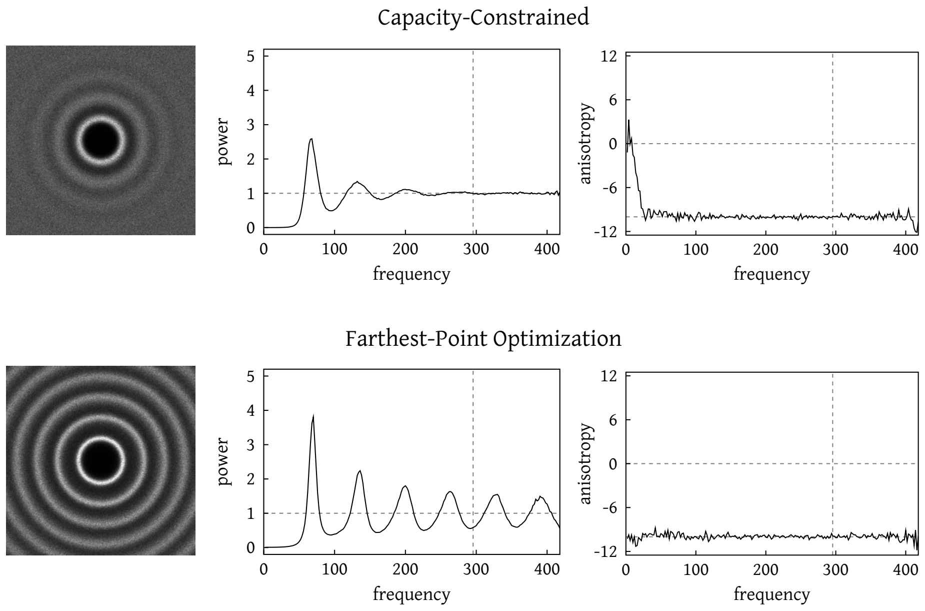


Figure 5.6: Spectral analysis of plain capacity-constrained and farthest-point optimized points.

(the first local maximum after the DC peak) is steeper. We will see that this is advantageous for an application like image plane sampling where the region below the principal frequency should be as close to zero as possible.

Figures 5.7 and 5.8 show the same analysis for tiled capacity-constrained points as already considered during the spatial analysis. We can observe that while the circularly averaged power spectra stay almost always intact, the 2D power spectra and the anisotropy curves reveal even subtle correlations in the final points sets. The underlying tile grid is all the more apparent in the power spectrum, the larger the ratio of the number of tiles (via C) to the number of points per tile n_t becomes. This is similar to what we observed with the RDF during spatial analysis but in this case the anisotropy curves reveal the correlations even for good ratios such as $C = 2$ and $n_t = 256$. We conclude that for high-quality results the ratio of n_t to the number of tiles $|T|$ in the tiling should be at least $n_t/|T| \geqslant 16$.

Farthest-Point Optimized Points

Figure 5.6 (bottom) shows the spectral results for plain farthest-point optimized points. We can see that the spectrum profiles can be considered "blue noise" as well, with virtually no energy in lower frequencies and a very steep transition to the principal frequency. This is a consequence of maximizing the minimum distance between points in the spatial domain. We also see that, after the principal frequency, the power continues to oscillate around 1. We suspect this to

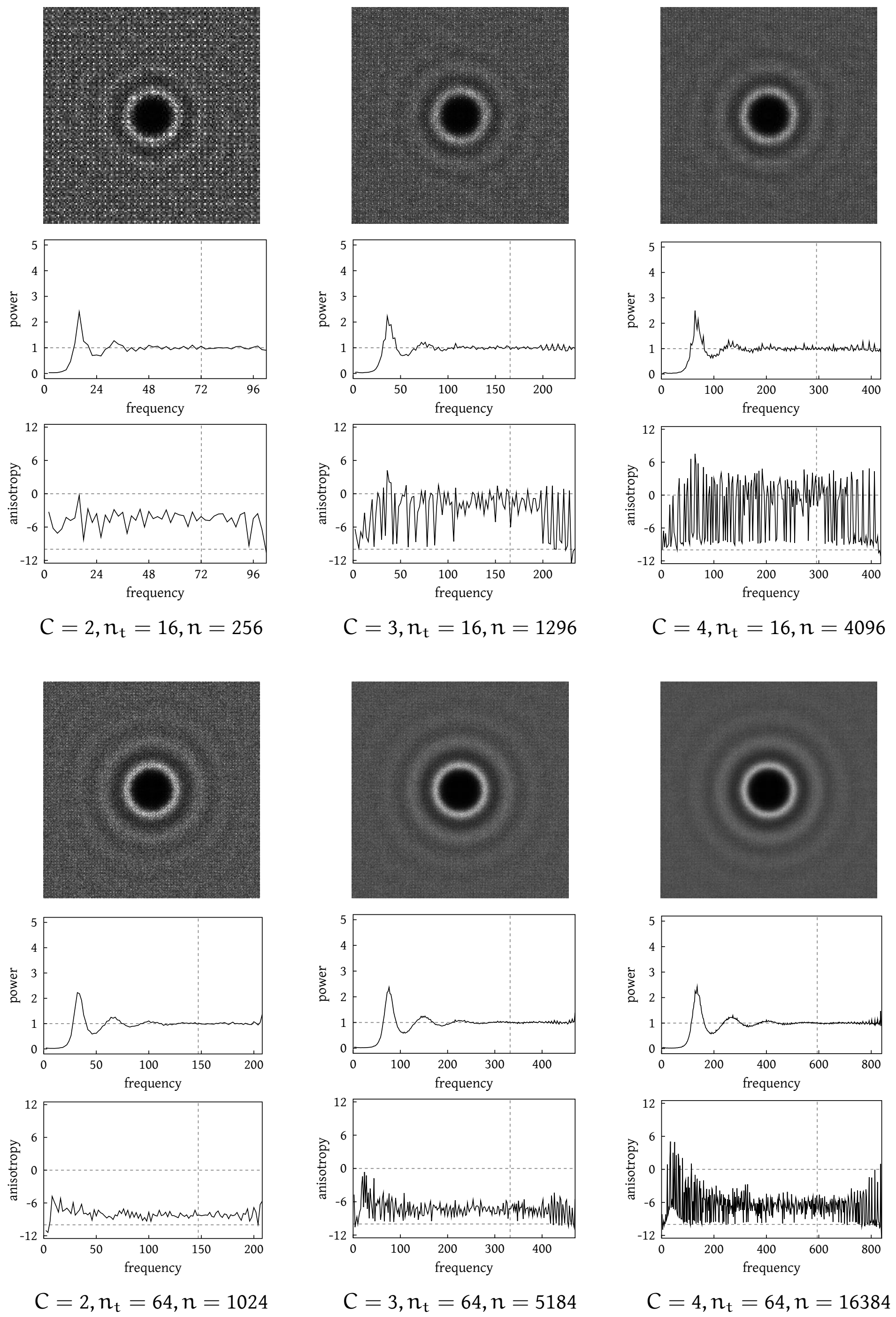
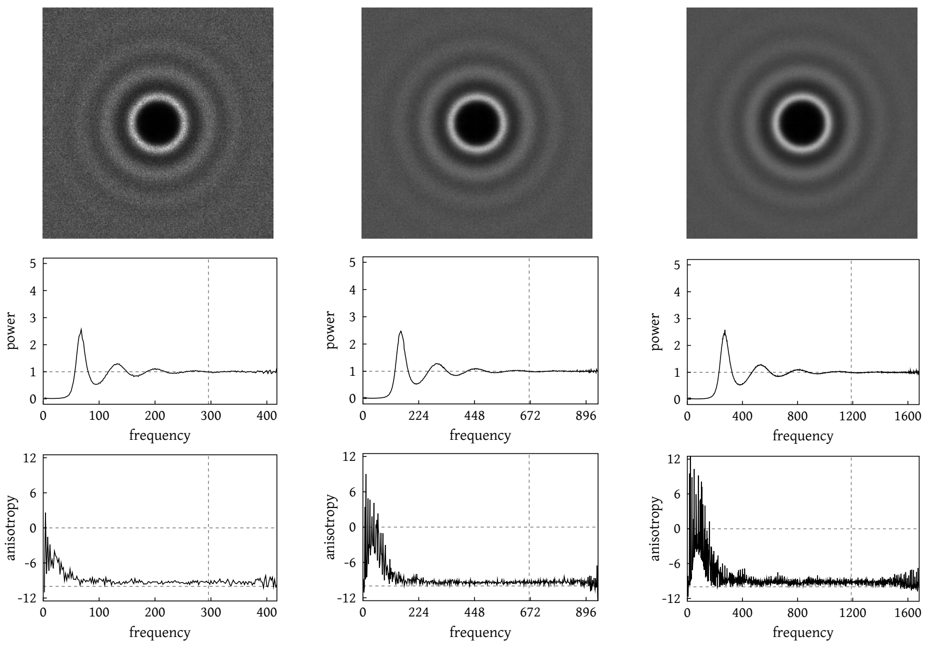
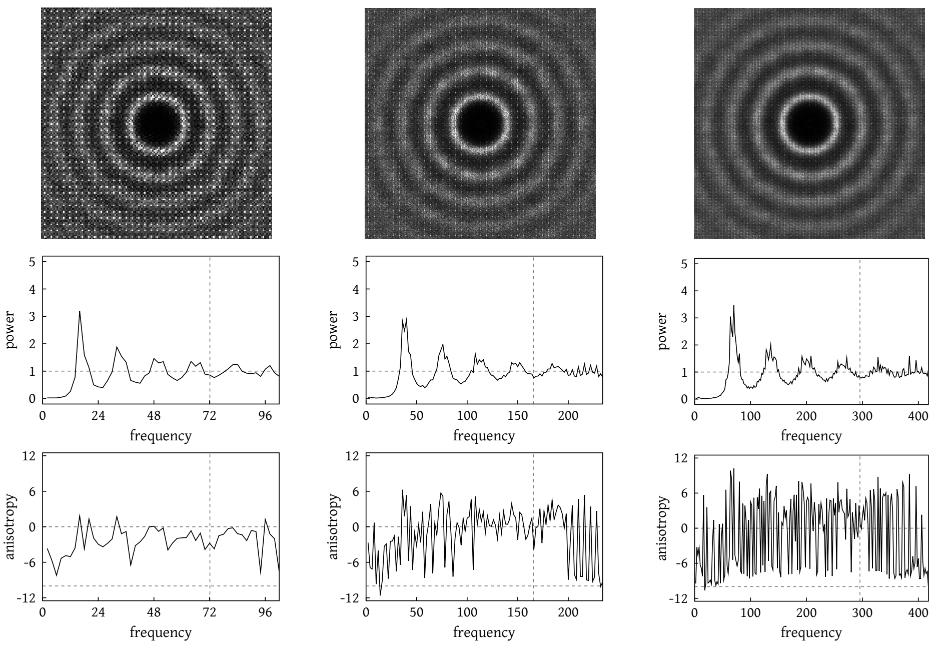


Figure 5.7: Spectral analysis of tiled capacity-constrained points based on corner tile sets with C colors, n_t points per tile, and total number of points n.



$C = 2, n_t = 256, n = 4096$ $C = 3, n_t = 256, n = 20736$ $C = 4, n_t = 256, n = 65536$

Figure 5.8: Spectral analysis of tiled capacity-constrained points based on corner tile sets with C colors, n_t points per tile, and total number of points n.



$C = 2, n_t = 16, n = 256$ $C = 3, n_t = 16, n = 1296$ $C = 4, n_t = 16, n = 4096$

Figure 5.9: Spectral analysis of tiled farthest-point optimized points based on corner tile sets with C colors, n_t points per tile, and total number of points n.

be related to the small variance of the point-to-point distances but have no hard evidence. In fact, we were unable to determine the end of the oscillation because the amplitude falls off so slowly. Nonetheless, the low-energy part below the principal frequency make the FPO points an even better candidate for image plane sampling than the capacity-constrained points.

Meanwhile, Figures 5.9 and 5.10 show analysis results for tiled FPO points. The results are similar to those of the capacity-constrained points when we compare each of them to their untiled counterparts: Artifacts from the tile grid become hardly detectable once we use enough points per tile. This underlines that the variants of both optimization techniques employed during tile construction are able to keep the favorable characteristics of the original methods and, e.g., do not introduce additional artifacts.

5.3 Image Plane Sampling

The main motivation for analyzing point distributions with respect to their spectral properties can be explained by considering the fundamental task of image plane sampling [Gla94] for which blue noise point sets have long been conjectured to be ideal [Mit91]. During image plane sampling, the points are distributed across the image plane and mark the positions at which the image function is being sampled. In computer graphics, we typically only have the information from these samples to reconstruct the original image function because the original function is often defined only implicitly, for example by a given scene description, material properties, camera setup, and more.

Sampling and Reconstruction

Let us briefly recapitulate the sampling and reconstruction process from the computer graphics point of view. A general introduction can be found in any textbook on signal processing [PM96] and a good introduction from a computer graphics angle in the books by Glassner [Gla94] or Pharr and Humphreys [PH10].

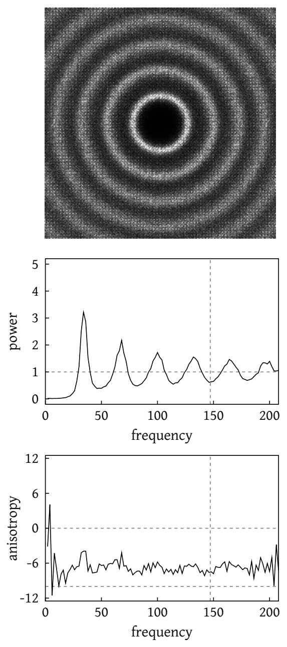
From a signal processing perspective, sampling a continuous image function f by a set of samples X represented by $s(x) = \sum_X \delta(x - x_j)$ amounts to the product

$$f_d(x) = f(x) \cdot s(x),$$

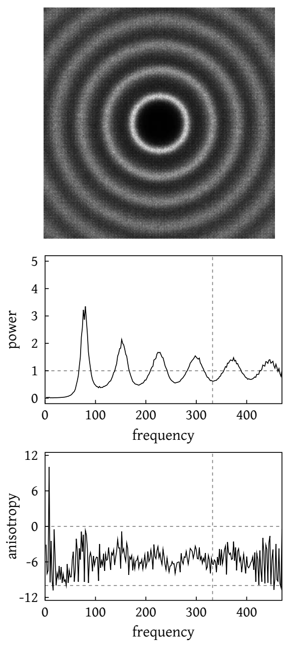
where f_d describes the discretized image function. In the frequency domain, this is equal to the convolution of the Fourier spectrum F of the image function with the Fourier spectrum S of the point set, i.e.,

$$F_d(\omega) = F(\omega) \star S(\omega).$$

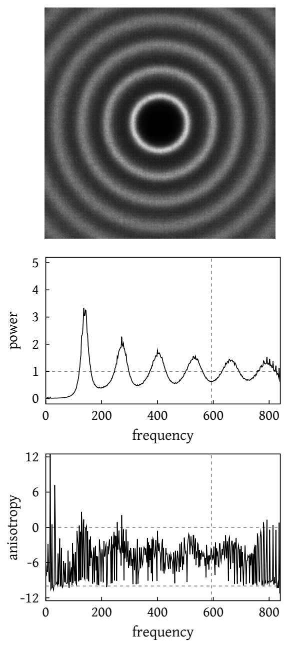
This convolution results in shifted copies of F across the Fourier domain where the displacement and scaling of each copy is determined by the spectral properties of the point set. This already hints at why point sets with no energy in the



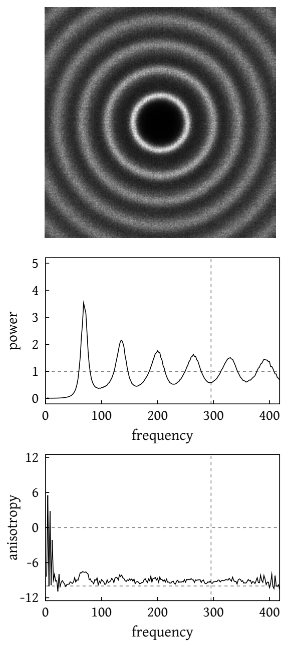
$C = 2, n_t = 64, n = 1024$



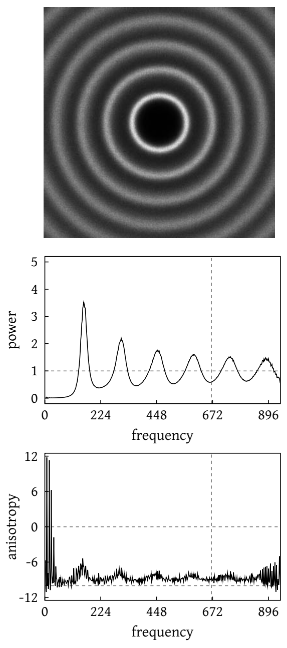
$C = 3, n_t = 64, n = 5184$



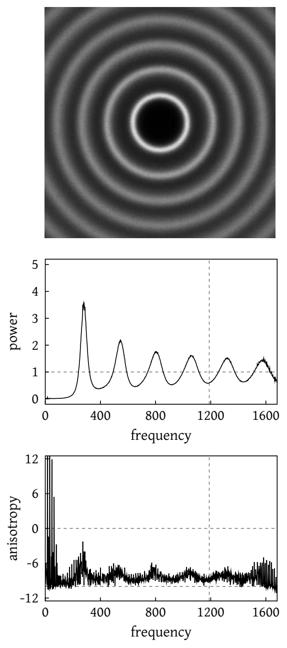
$C = 4, n_t = 64, n = 16384$



$C = 2, n_t = 256, n = 4096$



$C = 3, n_t = 256, n = 20736$



$C = 4, n_t = 256, n = 65536$

Figure 5.10: Spectral analysis of tiled farthest-point optimized points based on corner tile sets with C colors, n_t points per tile, and total number of points n.

$$\begin{tikzcd} f \arrow[r] & \boxed{\text{Supersampl.}} \arrow[r] & \boxed{\text{Filtering}} \arrow[r] & \boxed{\text{Resampling}} \arrow[r, "f_d"] & \boxed{\text{Reconstr.}} \arrow[r] & f' \end{tikzcd}$$

Figure 5.11: The full sampling and reconstruction pipeline.

lower frequencies are preferable for image plane sampling: Copies are placed far apart and the valuable low-frequency parts of the image spectrum remain intact.

So far, we did not make any assumptions about the image function (it only must be compatible with point sampling) nor about the utilized points. We know from classical sampling theory that the original function f can only be reconstructed exactly from f_d if two conditions hold: F must not contain frequencies above some finite frequency ω_f after which $F(\omega) = 0$ and, simultaneously, the point set X must consist of points arranged on a rectangular grid not farther than $1/2\omega_f$ apart. Otherwise, the shifted copies of F start to overlap and it becomes impossible to isolate an uncontaminated copy of F, a process known as *reconstruction*. The error due to a potential overlap is known as *aliasing* and the necessary sampling frequency ω_f is known as the *Nyquist frequency*. If a given image function has $\omega_f < \infty$, it is said to be *band-limited*.

In computer graphics, however, most of the interesting image functions are not band-limited as they contain discontinuities, so the uniform sampling theorem based on rectangular grid points does not fit. Thus, it is hard to avoid aliasing in the final images. For example, using the aforementioned rectangular grid samples results in a systematic overlap of the image spectra which yields aliasing that is *coherent* (e.g. moiré patterns). We will see this shortly when we sample a difficult test function. Luckily, several countermeasures allow us to attenuate such coherent aliasing by a substantial degree:

Prefiltering We remove frequencies in F higher than the Nyquist frequency of the final pixel grid. Because we do not know the image function explicitly, exact prefiltering is seldomly possible in practice.

Supersampling We increase the sampling frequency by sampling at a higher rate than the Nyquist frequency of the final pixel grid. We then have to resample to the output resolution.

Non-Uniform Sampling We do not align sample positions with the pixel grid and make the aliasing incoherent. Frequencies higher than the Nyquist frequency then appear as noise.

Often, none of these techniques alone is sufficient enough to yield satisfactory images which is why they are often combined. In particular, prefiltering is typically used in conjunction with supersampling: The image function is temporarily reconstructed from all "supersamples" and the frequencies beyond the Nyquist frequency of the pixel grid are removed by applying a digital filter. In fact, supersampling alone can sometimes be considered a (local) prefilter of the image function [HD11]. The final reconstruction of the image function from this pixel grid is then performed by the output device (e.g. display or printer). The full sampling and reconstruction pipeline is summarized in Figure 5.11.

In recent years, reconstructing functions using non-uniform sampling has found increasing attention [Mar01] but obtaining a general result comparable to the uniform case is a lot harder. For example, it is possible to reconstruct a function from a non-uniform sampling pattern if it is used periodically but the underlying functions must still be band-limited [ST06]. The contributions in this thesis touch both non-uniform sampling and supersampling. Both of our methods generate irregular (non-uniform) point sets that may conform to a higher sampling rate than the pixel grid and are typically distributed across the whole image plane, i.e., regardless of pixel boundaries. We will see that both methods are very effective at mapping aliasing to high frequency noise which generally yields better images than comparable methods at the same sampling rate.

Zone Plate Test Function

We can evaluate the image plane sampling qualities of our points by considering a case where we actually know the underlying image function. In particular, we would like to judge their quality in relation to the occurring frequencies. A common test function for such a case is a sine wave of the form $z : x \mapsto \frac{1}{2}[1+\cos(\lambda^2)]$ [Mit90] where $\lambda := \|x\|$ is the length of a two-dimensional sample point (vector) $x \in [0,1)^2$. In computer graphics, this type of function is often called *zone plate* in an analogy to test plates for optical systems [Goo04]. In signal processing, it is often called *chirp* or *sweep signal* because the frequency of the sinusoid increases with time (in our case with λ). This means that z must eventually produce frequencies that are beyond the Nyquist frequency of the pixel grid at $\omega_{\text{px}} = \frac{\pi}{4}m$ where $m \times m$ denotes the resolution of a quadratic output image. This way, aliasing is inevitable, either in coherent form or in the form of noise.

Following Heck and Deussen [HD11], we use a generalized form of this function which allows better control over the occurring frequencies:

$$\begin{aligned} z' : [0,1)^2 &\to [0,1] \\ x &\mapsto \frac{1}{2}\big[1+\cos\big(\omega(\alpha\lambda+\phi)\big)\big]. \end{aligned}$$

We choose the frequency ω, the parameter α, and the phase ϕ as:

$$\begin{aligned} \omega &:= \omega_{\text{px}}\omega_{\max} & \omega_{\max} &\geqslant 0, \\ \alpha &:= \min(\lambda/\lambda_{\max}, 1) & \lambda_{\max} &> 0, \\ \phi &:= \max(\lambda-\lambda_{\max}, 0). \end{aligned}$$

Here, $\omega_{\max}$ denotes the maximum frequency relative to the Nyquist frequency of the pixel grid, and $\lambda_{\max}$ denotes the desired distance of $\omega_{\max}$ to the origin. For

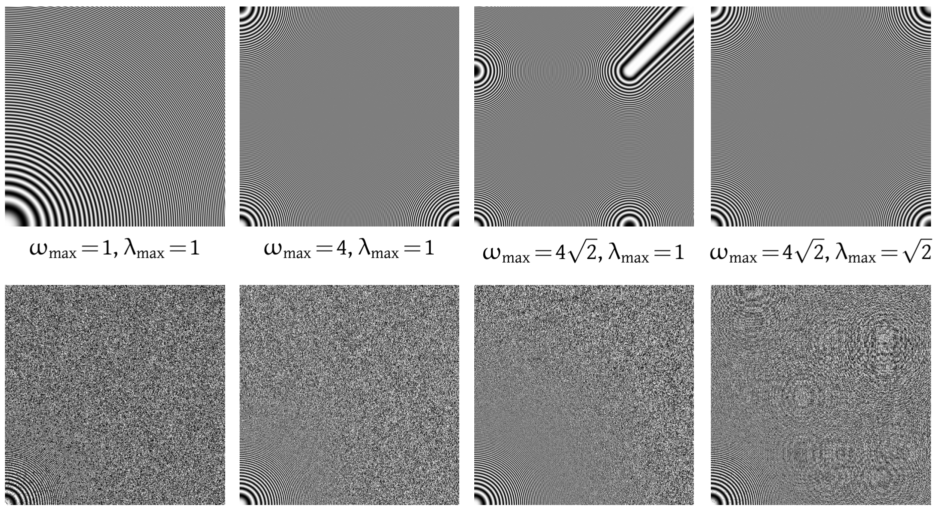

$\omega_{\max}=1$, $\lambda_{\max}=1$
$\omega_{\max}=4$, $\lambda_{\max}=1$
$\omega_{\max}=4\sqrt{2}$, $\lambda_{\max}=1$
$\omega_{\max}=4\sqrt{2}$, $\lambda_{\max}=\sqrt{2}$
Random
Jittered Grid
Dart throwing
[Hal60]

Figure 5.12: Examples when sampling the test function z' with a regular sampling pattern corresponding to the pixel grid (top) and several irregular sampling patterns for $\omega_{\max} = 4\sqrt{2}$ and $\lambda_{\max} = \sqrt{2}$ (bottom).

filtering, we use the common filter by Lanczos [Duc79] which is defined as

$$r(x) := \begin{cases} \operatorname{sinc}(x)\operatorname{sinc}(x/a) & \text{if } |x| < a, \\ 0 & \text{otherwise.} \end{cases}$$

The corresponding function in two dimensions is given by $r(x, y) := r(x)r(y)$. The parameter a defines the support of the filter. We use $a = 2$.

Figure 5.12 (top) shows some examples when sampling z' with a regular (uniform) sampling pattern of 512×512 points corresponding to the pixel grid at resolution $m = 512$.[1] Setting $\omega_{\max} \leqslant 1$ yields no aliasing because there are no frequencies larger than the Nyquist frequency of the grid. Setting $\omega_{\max} > 1$ yields aliasing if this frequency occurs within range $\lambda_{\max} < \sqrt{2}\omega_{\max}$. We can see this in the center examples where we get coherent aliasing in the form of moiré or ringing patterns, most notably around $\omega = 4$ where the output looks like the original signal around $\omega = 0$. The main configuration we use is shown to the right where $\omega_{\max} = 4\sqrt{2}$ and $\lambda_{\max} = \sqrt{2}$, i.e., at the outer corners the sine has a frequency four times the Nyquist frequency of the grid. Note that an ideal rendering of z' would take on a perfect gray value of 0.5 for frequencies above ω_{px}.

Figure 5.12 (bottom) demonstrates the effect when regular sampling is replaced with irregular sampling. Coherent aliasing is replaced by incoherent aliasing in the form of noise. We also see the relation between aliasing and the energy

[1]Note that a printer or PDF viewer most likely distorts these images due to resampling.

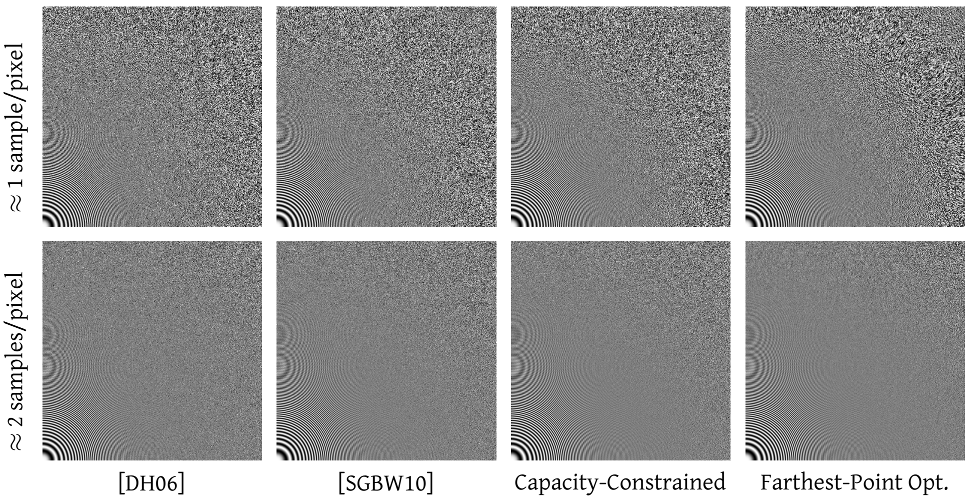


Figure 5.13: Sampling the test function z' with point sets from the methods by Dunbar and Humphreys [DH06], Schmaltz et al. [SGBW10], and our two methods using approx. one sample per pixel (top) and two samples per pixel (bottom).

distribution in a point set's power spectrum (recall Figure 5.5). The less energy in the lower frequency parts of a point set's power spectrum, the better we are able to cleanly capture the low frequency-part of the image function. Increasing the mutual distances between points (from random to jittered grid to dart throwing) increases the effective Nyquist frequency of the sample pattern while their irregularity keeps high frequency-aliasing mapped to noise. The example to the right shows what happens when a semi-regular point set such as points from the Halton sequence [Hal60] is used: We get a mixture of noise and moiré patterns.

Figure 5.13 shows the result of our two methods in comparison to the two state-of-the-art methods by Dunbar and Humphreys [DH06] and Schmaltz et al. [SGBW10]. For this example, we precomputed untiled sets of 512×512 and 2×512^2 sample points in order to judge the quality of both methods without influences from the tiling process. Both of our methods yield very good results. Low-frequency content is kept even clearer than for the state-of-the-art methods and aliasing is mapped to higher-frequency noise. The FPO points do a particularly good job in this case, offering a good compromise between moiré artifacts and noise. Their high minimum distance result both in a high effective Nyquist frequency (which lessens aliasing) and a high uniformity (which lessens noise). At approx. 2 samples/pixel the solution already approaches an ideal one. For these reasons, we will mostly concentrate on FPO points in the remaining image plane sampling scenarios.

Figure 5.14 shows what happens when we replace the 512×512 untiled FPO points by tiled FPO points generated from complete sets of corner tiles. This allows us to study the influence of the tiling process with varying number of corner colors C and number of points per tile n_t. For reference, the first column shows

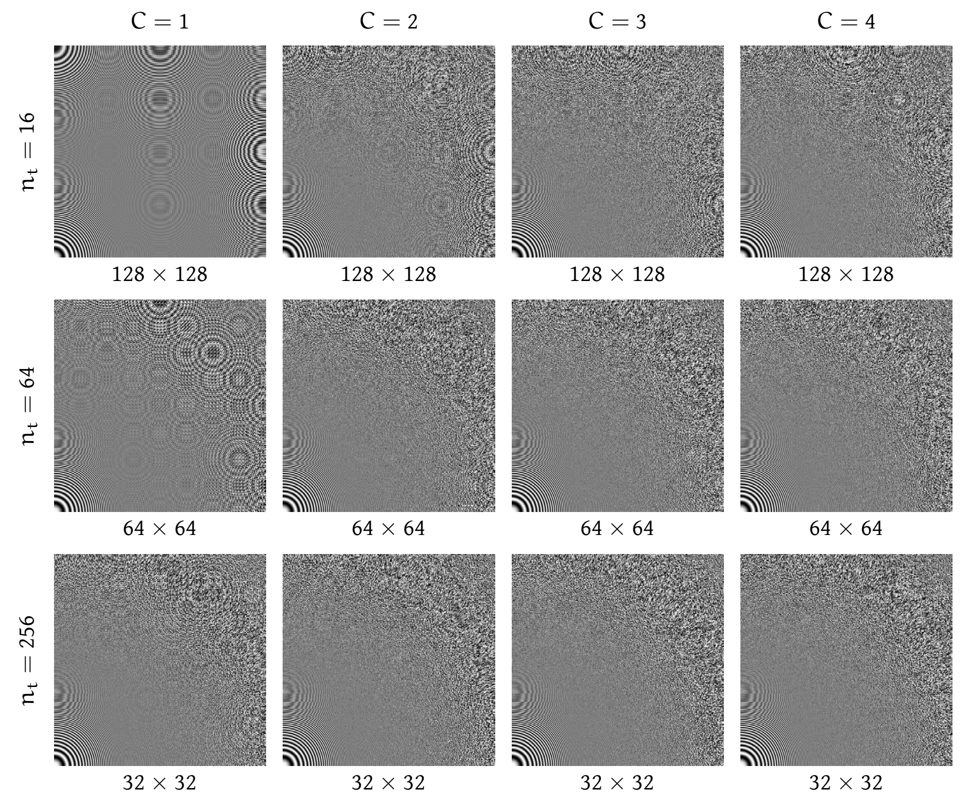


Figure 5.14: Sampling the test function z' with tiled FPO points based on complete corner tile sets with varying number of corner colors C and number of points per tile n_t. The size of the underlying tilings is given below each rendering and was chosen such that the total number of points equals 512×512.

a naive periodic tiling where a single tile is repeated over the whole domain. While the point's sampling behavior with respect to low-frequency content stays mostly untouched, coherent aliasing reappears above the Nyquist frequency if we use too few points per tile. Interestingly, this behavior cannot be remedied by simply using more tiles as is evident by very similar renderings from column to column. One explanation for this is that the coherent aliasing largely stems from the tiles' repeated boundary points whose influence on the total point set doesn't diminish until we use more points per tile. This is also supported by the points' anisotropy plots from Figures 5.9 and 5.10 which generally remained at the same level despite increasing C. For $n_t \geqslant 64$, however, the tiled results are on a par with those of the untiled FPO points and become hardly distinguishable.

The employed tiling algorithm for these results was a stochastic tiling algorithm because the arrangement of the tiles shouldn't introduce additional correlations to the final points (which could further amplify coherent aliasing). Our deterministic tiling algorithm, however, generated only slightly worse results. This underlines that the uniform distribution of corner colors is less important

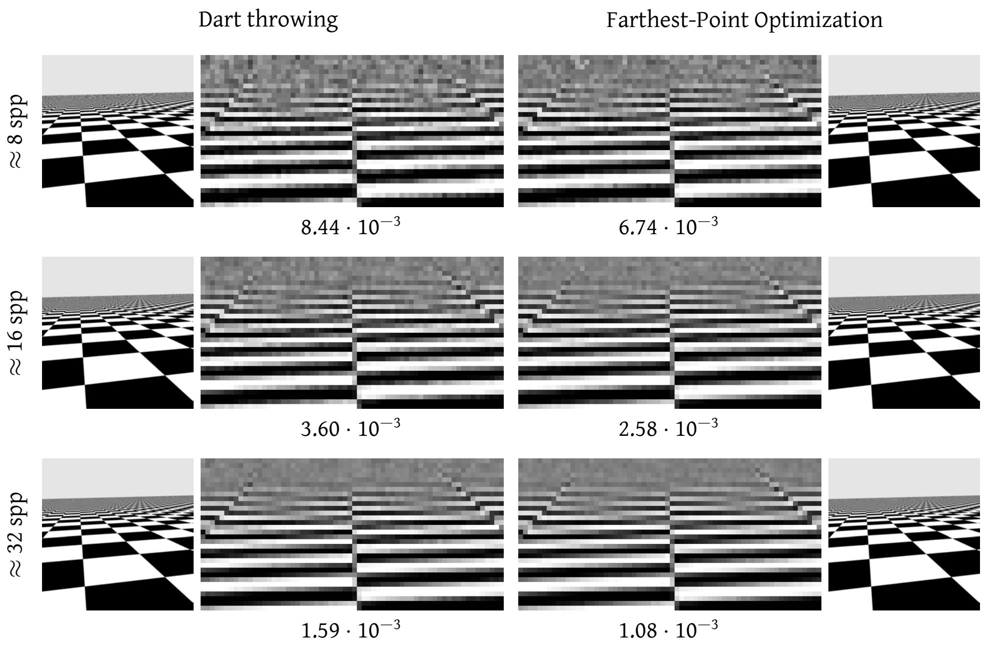


Figure 5.15: Sampling a checkerboard scene with dart throwing and FPO points.

for the generation of point sets than it is for the generation of textures.

Edge Anti-Aliasing

Another prominent image plane sampling-scenario is edge anti-aliasing. Edges are interesting because they constitute discontinuities in the underlying image function. This means that edges must suffer from aliasing almost always as the corresponding frequency in the Fourier domain goes to infinity. While this is a worst-case scenario from a signal processing point of view, from a perceptual perspective errors due to edge aliasing may actually be less critical than other forms of coherent aliasing [Nai98, LA06]. This is mainly because, at edge boundaries, moiré patterns can be harder to observe than e.g. traces of noise.

A consequence for the underlying sample point sets is that their uniformity properties become more important than their ability to cleanly capture low-frequency content. The more uniform a point set, the less noise can be observed at edges. This perceived quality often corresponds to a lower numerical error when comparing the results with reference images obtained by using many more samples per pixel.

Figures 5.15 and 5.16 demonstrate this when sampling dedicated edge anti-aliasing scenarios with points obtained by dart throwing and farthest-point optimization. We compare the resulting images by computing mean squared errors (MSE) in relation to reference solutions using 4096 random samples/pixel. In Figure 5.15 we render a slightly tilted, infinite checkerboard which yields

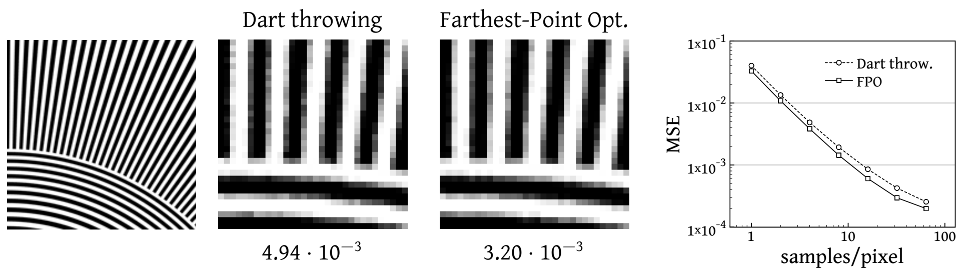


Figure 5.16: Sampling a series of edges with dart throwing and FPO points. The closeups use approx. 6 samples/pixel.

edges of many different orientations as well as frequency content above the Nyquist frequency. Although the visual differences are subtle, note the better edge anti-aliasing (bottom of the closeups) and reduced noise in regions beyond the Nyquist frequency (top of the closeups) for the FPO points. The MSE is also 20% to 33% lower. Figure 5.16 shows similar results when sampling a series of circles and lines. Increasing the uniformity of point sets by farthest-point optimization noticeably enhances edge anti-aliasing, both visually and numerically.

These observations remain true when using tiled FPO points. In fact, for these edge anti-aliasing scenarios, the results for tiled and untiled FPO points can be hardly distinguished, even for small sets of 16 tiles with $n_t = 16$ points per tile. MSEs also stay very close and often only differ after the fourth decimal place. We suspect this is a direct consequence of the low to medium frequency characteristic of edges which are insensitive to the high-frequency aliasing problems we could observe when sampling the zone plate function z'.

Lens Sampling

As a third application, we consider the problem of sampling a virtual camera lens to simulate optical effects such as depth of field. This can be considered an image plane sampling problem, too [PH10]: The n points on the image plane that fall within a certain pixel are randomly coupled with n well-distributed points on the lens to form the full camera rays. As before, we cover the image plane with one large set of FPO points, but across the camera lens, we reuse a second set of points and randomly shift this set on the unit torus for each pixel, i.e., $x_i(k) := (x_i(k) + \xi_i(k)) \bmod 1$ where ξ_i is a random vector chosen uniformly in $[0,1)^2$ and $x_i(k)$ selects the k-th coordinate of x_i. This technique is known as *Cranley-Patterson rotation* [CP76] and has the advantage of preserving the important minimum distance property of FPO points.

In Figure 5.17, we rendered a scene where a camera lens is configured such that only one object lies in the focal plane and the other objects are out of focus and appear blurred. Because blurred image content corresponds to low frequencies in the Fourier domain, we can expect FPO points to do a particularly good

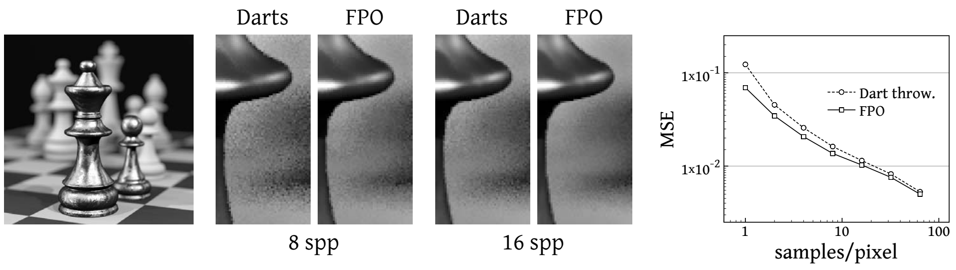


Figure 5.17: Lens sampling for depth of field with dart throwing and FPO points. The chess scene is courtesy of Wojciech Jarosz [HJW*08].

job in such a setting. (Recall their power spectrum from Figure 5.6 where we saw that they leave low-frequency content almost undisturbed.) The visual and numerical results in Figure 5.17 support this assumption. For example, for 8 FPO points the amount of noise remaining in out-of-focus image regions is almost as low as with twice the number of samples when using pure Poisson-disk points by dart throwing.

5.4 Numerical Integration

While image plane sampling is best to be considered a signal processing problem, many other sampling problems in graphics are best understood as numerical integration problems where (unknown) integrals are estimated by summing up contributions from point samples [PH10]. There are some results that connect sampling from a signal processing point of view and sampling from a numerical integration point of view [Dur11] but, so far, results are sparse and not understood in its entirety.

We are particularly interested in the integrals occurring in physically-based rendering where the central task is to estimate the value of the *light transport equation* (LTE) at any point x in a virtual scene

$$L(x, \omega) = L_e(x, \omega) + \int_{S^2} f_r(x, \omega, \omega_i) L(y, -\omega_i) |\cos\theta_i| \, d\omega_i,$$

where ω and ω_i denote outgoing and incoming light directions respectively, S^2 the unit sphere around point x, f_r the *bidirectional scattering distribution function* (BSDF) and θ_i the angle between the surface normal at point x and ω_i. The point $y := h(x, \omega_i)$ is the closest point from x in direction ω_i, determined by a *ray casting* function h. The LTE is formulated from the perspective of geometric optics where light is propagated in terms of *rays* between different surface points and where outgoing, emitting, and incoming light are measured in *radiance*. This modern form of the LTE was famously introduced as the *rendering equation* by Kajiya [Kaj86].

Solving the LTE in analytical form is only possible in few cases of simple scene complexity. It is possible to simplify the equation by limiting the BSDFs but this sacrifices generality and is usually undesirable in physically-based rendering. The classic approach to solve the LTE in its general form is by numerical integration where the integral of a function $f : I^s \mapsto \mathbb{R}$ is approximated by averaging weighted point samples at locations $\{x_0, \ldots, x_{n-1}\} \in I^s$ by

$$\int_{I^s} f(x)\,dx \approx \sum_{i=0}^{n-1} w_i f(x_i).$$

For the weights, we usually have $w_i \in [0,1]$, $\sum_i w_i = 1$. This approach is a good choice for a computer graphics setting where point sampling is possible by ray tracing.

Among many schemes for the choice of the corresponding sample locations and weights, the following three are the most popular within graphics: Monte Carlo integration, quasi-Monte Carlo integration, and volume-weighted quadrature. We briefly review these techniques and their error bounds in the following.

Monte Carlo Integration

Monte Carlo methods [Sob94] were among the first to robustly estimate graphics integrals such as the LTE [Kaj86, Gla94]. In its most basic form, point locations are realizations of independent and identically distributed random variables over I^s and the weights equal at $w_i := 1/n$. Since the method is probabilistic, the corresponding error bound can only be probabilistic as well. We have

$$\left| \int_{I^s} f(x)\,dx - \frac{1}{n} \sum_{i=0}^{n-1} f(x_i) \right| \leqslant \frac{3\sigma(f)}{\sqrt{n}}$$

with a probability of approx. 0.997. Here, $\sigma(f)$ denotes the standard deviation of the function f. The power of the method does not only stem from the fact that it is simple but also that the error bound is both independent of the dimension s and any smoothness properties of f. The big disadvantage is that the error falls off very slowly at a rate in the order of $\mathcal{O}(1/\sqrt{n})$.

There are two variants of this basic form of Monte Carlo integration, one that alters the point locations and one that alters the weights. In *importance sampling*, point locations are not chosen uniformly anymore but as proportional to f as possible while the weights are kept equal, $w_i := 1/n$. Importance sampling is only possible if we know something about the shape of f which, in graphics, is only sometimes the case. In *weighted Monte Carlo integration* [YKS78], on the other hand, point locations are chosen uniformly again but the weights conform to the volumes of the corresponding Voronoi regions, $w_i := \lambda_i$. (The definition of λ was given in Eq. (4.5).) In this case and under the assumption that f is twice-differentiable, the error is in the order of $\mathcal{O}(1/n^{2/s})$. This improves upon the convergence of pure Monte Carlo integration for dimensions $s < 4$ but we have to compute the full Voronoi diagram in order to determine the w_i. Generally, the dependence of the convergence rate on the dimension s is also a disadvantage and is known as the *curse of dimension(ality).*

Quasi-Monte Carlo Integration

Quasi-Monte Carlo methods [Nie92] are similar to Monte Carlo methods but replace random points with deterministic points of higher uniformity. These deterministic points often rely on number theoretic construction rules. For example, many are based on the radical-inverse function (such as the Halton sequence from Chapter 2) but some are based on lattice rules [Dam09], and others use construction matrices [GHSK08, GK09]. The common feature they share is that all of them are low-discrepancy, that is their star-discrepancy is in the order of $\mathcal{O}(\log^s n/n)$. (Recall the definition of star-discrepancy from Eq. (2.2).) A good introduction to quasi-Monte Carlo methods with applications to graphics can be found in the report by Keller [Kel03] or the thesis by Wächter [Wäc07].

The importance of a low star-discrepancy stems from the following deterministic error bound which is known as the *Koksma-Hlawka inequality*:

$$\left| \int_{I^s} f(x)\,dx - \frac{1}{n} \sum_{i=0}^{n-1} f(x_i) \right| \leqslant V(f)\, D_n^*(X),$$

where $V(f)$ denotes the variation of f in the sense of Hardy and Krause [Nie92]. Just as in plain Monte Carlo integration, we have equal weights $w_i := 1/n$.

A nice property of the Koksma-Hlawka inequality is that the error is divided into a property of the integrand and a property of the point set. Unfortunately, the variation V is seldomly finite in computer graphics because discontinuities occur so frequently. Thus, the error bound can only serve as an indicator for the favorable behavior of low-discrepancy points. In practice, however, low-discrepancy points often show a significantly faster convergence rate compared to random points [KK02].

Volume-Weighted Quadrature

Volume-weighted quadrature uses the idea of weighted Monte Carlo integration to choose the weights according to the volumes of the corresponding Voronoi regions and combines it with the idea of point locations that are well-distributed. More specifically, it connects the integration error to the energy of a point set's Voronoi tessellation by [Pag97, SGB07]

$$\left| \int_{I^s} f(x)\,dx - \sum_{i=0}^{n-1} \lambda_i f(x_i) \right| \leqslant \|f\|_2\, \mathcal{E}(X, \mathcal{V}),$$

where $\mathcal{V}(X)$ denotes the Voronoi tessellation of X, $\mathcal{E}$ the energy from Eq. (4.4), and $\|\cdot\|_2$ the L^2-norm.

Similar to the Koksma-Hlawka inequality, this error bound is separated into a property of the integrand and a property of the point set. And similar to weighted Monte Carlo integration, it can be shown that the error is in the order of $\mathcal{O}(1/n^{2/s})$ if $f \in C^2$. Again, the functions in computer graphics usually aren't C^2-continuous, so this error bound can also only serve as an indicator. In addition, we still need the full Voronoi tessellation to determine the correct weights.

Numerical Integration by Capacity-Constrained Points

The volume-weighted integration scheme is interesting because it suits the notion of our capacity-constrained points. Recall from Section 4.3 that the energy $\mathcal{E}$ is minimized only if the points X form a centroidal Voronoi tessellation. Since the same energy function is minimized by capacity-constrained Voronoi tessellations that are centroidal, we can expect a low integration error for capacity-constrained points. Using our capacity-constraint (4.9), we have

$$\sum_{i=0}^{n-1} \lambda_i f(x_i) \approx \frac{1}{n} \sum_{i=0}^{n-1} f(x_i) \quad \text{and} \quad \mathcal{E}(X, \mathcal{V}) \to \min.$$

This means we can use the same sampling-and-averaging quadrature rule that is used for (quasi-)Monte Carlo points for our capacity-constrained points and still benefit from a low energy $\mathcal{E}$, indicating a low integration error. The Voronoi region volumes are not strictly $1/n$ because $\mathcal{E}$ is defined with respect to the ordinary Voronoi tessellation and not with respect to the capacity-constrained Voronoi tessellation. In Table 4.2, however, we showed that the volumes' deviation from $1/n$ is typically very small, i.e., the CCVT energy $\mathcal{E}'$ is close to zero.

In the following, we demonstrate the benefits of capacity-constrained points in practical applications by estimating various integrals occurring in physically-based rendering. To this end, we integrated our tile-based capacity-constrained points as a sampler into the PBRT rendering system [PH10]. Due to the necessary (low) sampling rates, we mostly used a small tile set based on $C = 2$ corner colors with $n_t = 16$ points per tile and a stochastic tiling algorithm. If greater versatility in the generated points is necessary, it suffices to either use compact tile sets with $C > 2$ or complete tile sets with $C = 2$ but $n_t > 16$. Obviously, for a full coverage of the sample space, the total number of points must be dividable by the number of tiles and points per tile without remainder. This is similar to many deterministic point sequences which must often be power of two for a good coverage [Kel03]. In cases where still other sampling rates are required, we rank each tiles' points using the ranking algorithm from Appendix A.3.

We will compare our results with those from quasi-Monte Carlo integration using the $(0, 2)$-sequence by Sobol' [Sob67], a low-discrepancy sequence that has proven to be among the best and most versatile for integration problems in graphics [GRK12]. Although the LTE is a high-dimensional (theoretically infinite-dimensional) integral, it is usually more efficient to understand it as a combination of many one- and two-dimensional integrals because some dimensions are highly correlated (for example, material properties captured by a BSDF or multiple non-singular light sources). Both the low-discrepancy and our capacity-constrained points are thus integrated as 2D samplers into PBRT and will often be transformed from $[0, 1)^2$ to a domain of interest.

In order to keep the integral estimators unbiased, we cannot use the same point set over and over again but have to use unique point sets for every integral. For (randomized) quasi-Monte Carlo integration, these unique point sets are often determined by *scrambling* [KK02] which preserves low-discrepancy properties. For our tile-based capacity-constrained points, we could either use Cranley-Patterson rotations on the same point set or we could use different point sets obtained by different stochastic tilings. Both of these techniques preserve the capacity constraint, but only the former yields an unbiased estimator. We thus use Cranley-Patterson rotations in the following although the bias of the latter technique is usually not visible once the points are generated by many tiles.

Generally, numerical error corresponds to noise in the final images, so we can again judge results both visually and by considering the MSE to a reference solution using many more samples per pixel.

Direct Light Estimation

First, we will numerically approximate the *direct lighting integral* (DLI). The DLI computes the incident radiance at a point x that arrives directly from light sources. For efficiency, it is often separated from the full LTE and then combined with an explicit method for the remaining indirect light, for example path tracing [Kaj86] or Metropolis light transport [VG97]. It is defined as

$$L(x, \omega) = L_e(x, \omega) + \int_{S^2} f_r(x, \omega, \omega_i) L_d(x, \omega_i) |\cos\theta_i| \, d\omega_i,$$

where L_d denotes the incident radiance directly from light sources. When approximating the DLI, the sample points are used to determine the directions ω_i. For m light sources, this is typically done by distributing m unique point sets Y_j across their surfaces and connecting these points with the surface point x. Similarly, the evaluation of the BSDF at x uses its own point set Y' to sample f_r. The results are then combined using multiple importance sampling [VG95].

Figure 5.18 shows the result for a simple test scene illuminated by a single circular area light source and objects with only constant (diffuse) BSDFs. Using capacity-constrained points generally yields images with less noise and a lower MSE than using the $(0, 2)$-sequence: Soft shadows appear smoother and surfaces appear less noisy. Because the light source is circular, this indicates that the

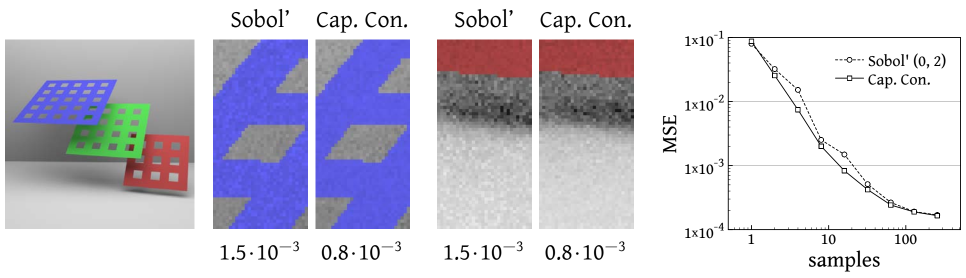


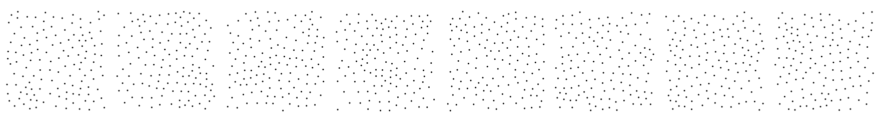
Figure 5.18: Estimating direct light in a simple test scene using points from the Sobol' $(0, 2)$-sequence and our capacity-constrained points. In the closeups 16 integrator samples were used.

properties of capacity-constrained points remain advantageous even after mapping the points from the unit torus to a disk [SC97].

Trajectory Splitting

This first example uses only one image sample per pixel but multiple *integrator samples* for the evaluation of the DLI. Usually, both multiple image samples as well as multiple integrator samples are desirable. In this case, it is advantageous to assign each image sample a set of integrator samples such that each set of integrator samples is well-distributed as well as their union. This way, the full sample space is covered much better than with sets that are well-distributed only with respect to a single image sample. Because a single primary ray is split into multiple secondary rays, this technique is known as *trajectory splitting* [KK02, PH10].

To support trajectory splitting using capacity-constrained points, we use the distance-based partition algorithm from Appendix A.2. It partitions any set of points into equally-sized subsets such that each subset is well-distributed in terms of the points' mutual distances while their union constitutes the original point set. The following figure shows what happens when we partition the 1024 capacity-constrained points from Figure 4.2 into eight subsets with 128 points each.



$\bar{\delta}_X =$ 0.810 0.808 0.807 0.807 0.817 0.813 0.809 0.808

The average mindist below each plot indicates good uniformity for each subset.

Figure 5.19 shows the result when estimating the DLI using trajectory splitting. The scene now involves two light sources, one circular area light source and one "infinite" light source where the samples are warped according to an HDR environment image. The scene also combines diffuse, glossy, and measured BSDFs using a microfacet distribution. In the closeups, we keep the product of image samples and integrator samples constant at 16 *combined samples* (cs). For

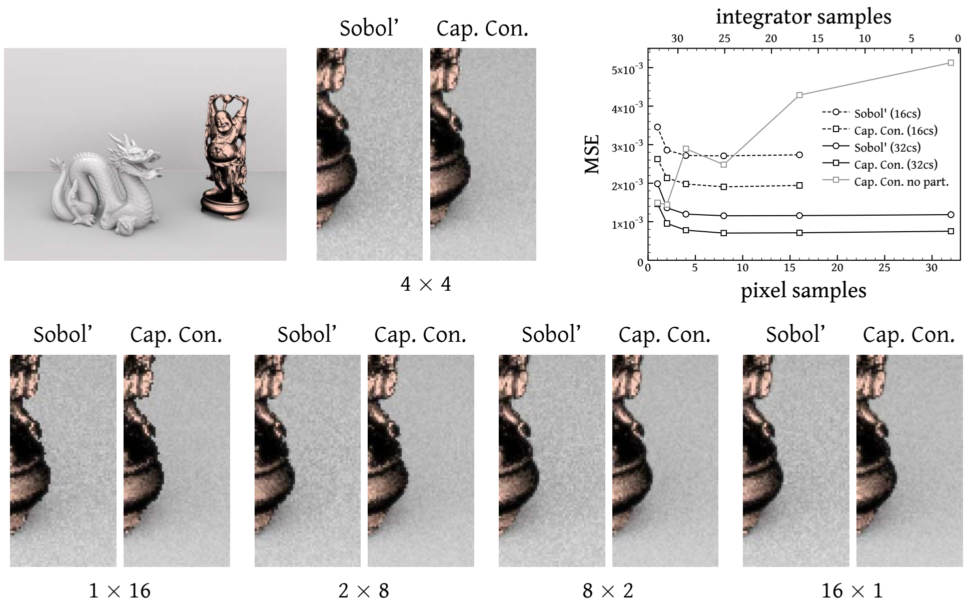


Figure 5.19: Estimating direct light by trajectory splitting points from the Sobol' $(0,2)$-sequence and our capacity-constrained points. In the closeups 16 combined samples (cs) were used.

performance reasons, we precompute the partition for the capacity-constrained points while the $(0,2)$-sequence can be partitioned by construction (each successive set of power of two points is a $(0,m,2)$-net of low-discrepancy [KG12]).

Using partitioned capacity-constrained points yields a lower error for every combination of pixel and integrator samples. In particular, the MSE remains roughly constant despite the varying numbers of pixel vs. integrator samples. The MSE plot also underlines the importance of the partition procedure. The naive variant ("Cap. Con. no part.") assigns each image sample a set of capacity-constrained integrator samples, but doesn't ensure that the union of the integrator samples is also well-distributed. This quickly becomes inferior to the partitioned results, even when using twice the number of combined samples.

Ambient Occlusion

As a third example, we consider the estimation of *ambient occlusion* [ZIK98]. Ambient occlusion determines how much of the hemisphere around a surface point x is occluded by other surfaces. It can be defined as the integral

$$A(x,n,r) = \frac{1}{\pi}\int_{H^2(n)} V(x,\omega_i,r)|\cos\theta_i|\,d\omega_i,$$

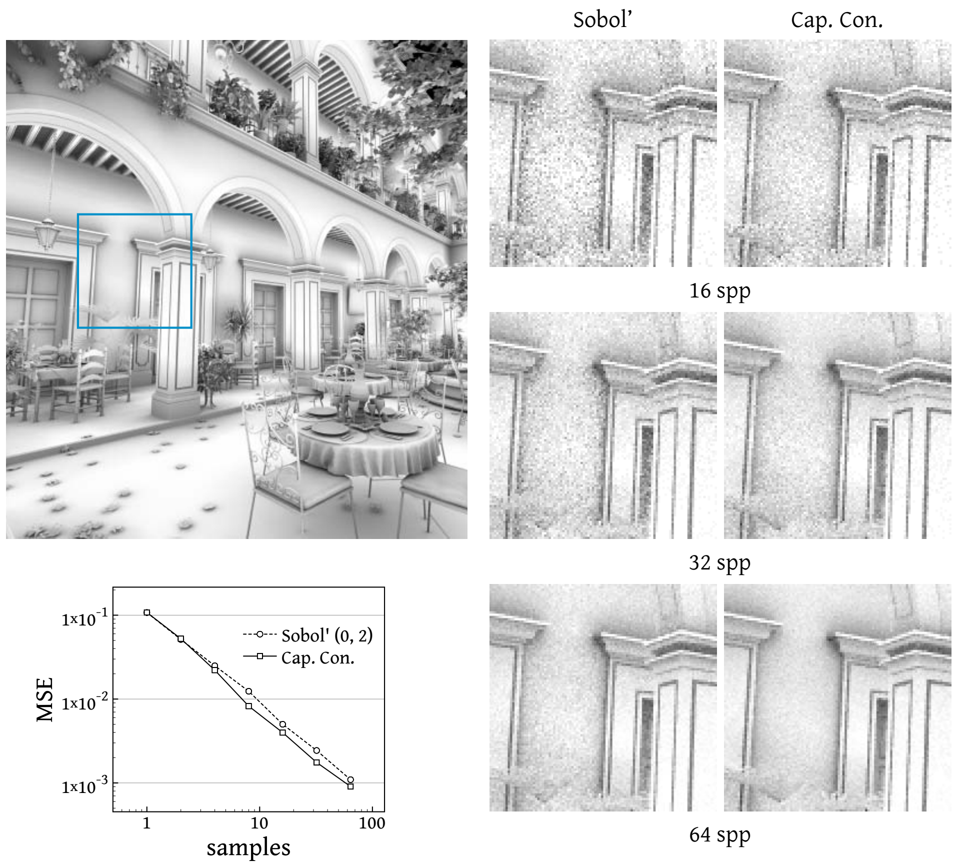


Figure 5.20: Estimating ambient occlusion using points from the Sobol’ $(0, 2)$-sequence and our capacity-constrained points. The San Miguel scene is courtesy of Guillermo M. Leal Llagun.

where

$$V(x, \omega, r) = \begin{cases} 1 & \text{if } \|h(x, \omega) - x\| \geqslant r, \\ 0 & \text{otherwise,} \end{cases}$$

is a variant of the *visibility function*, and $H^2(n)$ denotes the hemisphere around point x with normal n. (Recall that h is the ray casting function that returns the closest surface point from x in direction ω.) Estimating ambient occlusion is a good way of measuring how well the properties of a sample set translate to the unit hemisphere when determining the directions ω_i.

Figure 5.20 shows ambient occlusion results for a scene with complex geometry and a fixed parameter r. As from a number of samples greater than four, the capacity-constrained points clearly outperform the low-discrepancy points, consistently yielding a lower MSE. In some cases, visual results using the capacity-constrained points are practically as good as for the $(0, 2)$-sequence at only half the sampling rate. (Consider, for example, the results for 32 and 64 spp.)

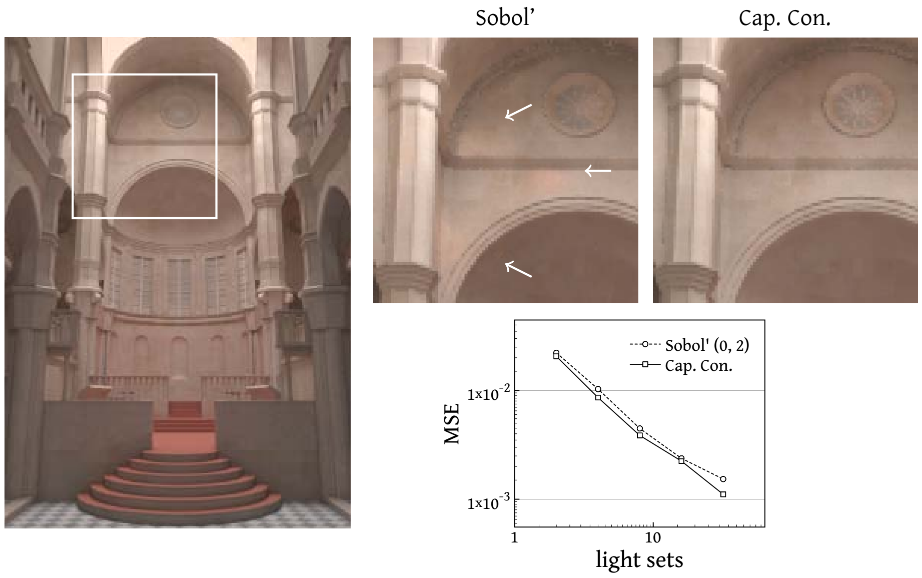


Figure 5.21: Distributing virtual point light sources for “instant global illumination” using the Sobol’ $(0, 2)$-sequence and our capacity-constrained points. The Šibenik Cathedral model is courtesy of Marko Dabrovic and Mihovil Odak.

Instant Global Illumination

As a last example, we consider the distribution of virtual point lights (VPLs) in an alternative approach to solve the LTE sometimes referred to as *instant global illumination* [WKB*02, PH10]. In this approach, sample point sets are used in a preprocessing step to mark the origins of a small but fixed number of light paths that are traced throughout the scene. VPLs are created along these paths at each point of intersection and which are then used during final rendering.

Although the point sets are not directly used for numerical integration, their distribution qualities are equally crucial for this approach. And similar to trajectory splitting, it is advantageous to use multiple sets of light paths that are well-distributed both with respect to a particular set as well as in the aggregate over all sets. For this approach to the LTE, a potential approximation error may not only show up as noise but also as a systematic error stemming from the finite set of VPLs: Parts of the image may appear too dark or too bright, depending on the distribution of the VPLs.

In the instant global illumination example in Figure 5.21, most of the light stems from a large sphere intersecting the ceiling of the utilized model. We compare capacity-constrained with low-discrepancy points when spawning a fixed number of 128 light paths at a varying number of light sets. Since we only want to assess the point sets’ quality with respect to the resulting distributions of VPLs, we used the same image sampling pattern for both experiments.

We see that the capacity-constrained points again yield lower MSEs of the final images than the Sobol' points. More importantly, the results show less artifacts due to an overall better distribution of VPLs. As highlighted in the closeups, approximation artifacts are less severe for the capacity-constrained points, with fewer noticeable spots stemming from an uneven distribution of VPLs.

5.5 Conclusion

In this chapter we thoroughly evaluated the spatial, spectral, and practical qualities of the tile-based point sets generated by our two optimization techniques from the previous chapter. We saw that both methods yield point sets that have blue noise spectral properties. In this regard, farthest-point optimized points stand out and improve upon the state of the art in the fundamental task of image plane sampling.

Capacity-constrained points, on the other hand, are favorable for numerical integration problems due to a related error bound in volume-weighted quadrature. Although this error bound doesn't strictly hold for many graphics problems, we saw that capacity-constrained points often improve upon state-of-the-art results using quasi-Monte Carlo integration for 2D integration problems that occur in physically-based rendering.

An interesting open question is if it is possible to construct point sets which minimize the energy $\mathcal{E}$ and, at the same time, have equally-sized Voronoi regions. A good candidate are Rank-1 lattices [Dam09] which strictly obey $\lambda_i = 1/n$ and show a very low energy $\mathcal{E}$. On the other hand, Rank-1 lattices are strictly regular and may constitute unstable CVTs [LWL*09]. We conjecture that it will be hard to further improve practical results using numerical integration by using only different sample locations. Even using FPO points instead of the capacity-constrained points for the integration problems already yields very similar results in practice.

6

Summary and Future Work

Efficiently generating complex content is one of the fundamental problems in computer graphics. The results of this thesis show that a tile-based method using corner tiles can be a feasible solution to this problem. Due to their square form, corner tiles are conceptually easy, allow straightforward tilings, and admit a broad range of tile construction methods. Once a set of corner tiles has been constructed, only a valid tiling using this set has to be generated online. Depending on the type of content, this may decrease the memory or runtime complexity for a specific generation process by several orders of magnitude.

We introduced two novel tiling algorithms for corner tiles, both of which are advantageous for the synthesis of non-periodic textures. Our deterministic tiling algorithm yields a more uniform distribution of texture content that is less prone to repetition artifacts than current stochastic algorithms. Our semi-stochastic tiling algorithm allows the synthesis of globally varying textures while still being compatible to tile-based texture mapping using graphics hardware.

Two-dimensional point distributions are another fundamental component of any rendering system. We introduced two new optimization methods that generate highly uniform, spatially homogeneous point distributions. Capacity-constrained point distributions are based on capacity-constrained Voronoi tessellations which enforce Voronoi regions of equal size. We demonstrated their advantages for numerical integration problems occurring in physically-based lighting computations. Farthest-point optimized point distributions are optimized with respect to their mutual distances and yield improvements in signal processing-based sampling tasks such as image plane sampling. Both methods improve the state of the art in their respective fields.

This thesis concentrates on two-dimensional corner tilings. A generalization to s-dimensional corner cubes is not difficult but the practical benefits of such corner cubes in a computer graphics setting remain to be seen. Complete sets of corner cubes would contain C^{2^s} cubes and constructing cube interiors is likely to be significantly more complicated than in two dimensions. Some research has already been done in this direction but, so far, the restriction on the type of content is quite strong [LEQ*07, PGGM09].

Some of the results of this thesis have already inspired further research in the last two years [LNW*10, Fat11, XLGG11, CYC*12, CG12, dGBOD12]. Still, there are several interesting open questions for future research:

Corner Tile Packings The compact tile packings presented in this thesis complement the packing solutions for complete tile sets in the literature for $C \leqslant 4$. Can we find solutions for $C > 4$? Is there a direct construction method for these solutions?

Tiling Algorithms Our semi-stochastic tiling algorithm often works with pruned tile sets, and our deterministic tiling algorithm works with complete tile sets. Is there a comparable tiling algorithm that also works with compact tile sets?

Tile Construction In this thesis, we only considered tile construction procedures for textures and sample point sets. Can we faithfully construct corner tiles for other types of content, e.g. complex geometry?

Texture Synthesis The presented texture tile construction method is quiet robust in the case of stationary textures but for globally varying textures, the results depend more strongly on the input textures. Is it possible to merge related input textures with as good results as stationary input textures? Is there a way to make this process recursive to support level-of-detail?

Image Plane Sampling For irregular point sets, farthest-point optimized points offer the largest minimum distance yet. Such point sets have long been conjectured to be ideal for image plane sampling [Mit91]. Recent results indicate large mutual distances may not be the most important aspect for such point sets, after all [HSD12a, HSD12b]. Is there a precise way to measure the noise/aliasing tradeoff for image plane sampling?

Numerical Integration Capacity-constrained points deliver good results for several integration problems in physically-based rendering but some properties (e.g. a good partition) have to be added as a post process. Is there a way to add these properties during the tile construction process? Can we make the process recursive akin to tile-based Poisson-disk point sets [KCODL06]?

To support the adoption of the methods presented in this thesis and simplify a comparison to future methods, we provide exemplary implementations for most of our algorithms.[1]

[1] `http://thomas-schloemer.org`

A

Appendix

In this appendix, we include an example implementation for a tile-based texture mapping application using our deterministic color distribution function from Chapter 2. We also discuss three algorithms that add different properties to a given point set. The first algorithm partitions a point set into equally-sized subsets such that each subset is well-distributed. The second algorithm determines a ranking for a point set such that it may unfold in a well-distributed manner. And the third algorithm adds the Latin hypercube property to a point set such that it becomes stratified in each dimension.

A.1 Deterministic Corner Tilings Implementation

To illustrate the computation of the deterministic color distribution function from Chapter 2, we give a full example implementation for a tile-based texture mapping application using GLSL [SA12]. A tile packing solution is expected to be encoded as a one-dimensional array in the variable `tp`.

```
uniform int nc;            // Number of corner colors
uniform ivec2 pows;        // Halton powers pk
uniform ivec2 exps;        // Halton exponents nk
uniform ivec2 minverses;   // Halton multiplicative inverses

uniform ivec2 tsize;       // Tiling width and height
uniform ivec2 tpsize;      // Tile packing width and height
uniform sampler1D tp;      // Tile packing
uniform sampler2D tpdecal; // Input texture in tile packing arrangement
in vec2 coords01;          // Texture coords in [0,1]^2
out vec4 texcolor;         // Final output color

int vdc_inverse(int inverse, int ndigits, const int base)
{
  int index, digit;
  index = 0;
  while (ndigits > 0) {
    digit   = inverse % base;
    inverse = inverse / base;
    index   = index * base + digit;
    ndigits--;
  }

  return index;
}

int corner_color(int x, int y)
{
  int l1 = vdc_inverse(x, exps[0], 2);
  int l2 = vdc_inverse(y, exps[1], 3);

  int prod = pows[0] * pows[1];

  int s1 = (pows[1] * ((l1 * minverses[0]) % pows[0])) % prod;
  int s2 = (pows[0] * ((l2 * minverses[1]) % pows[1])) % prod;

  int index = (s1 + s2) % prod;

  float ri = float(index) / float(prod);
  return int(ri * float(nc));
}

void main()
{
  // Tile coordinates
  vec2 coords = coords01 * vec2(tsize);
  int x = int(coords.x);
  int y = int(coords.y);

  // Corner colors
  int c_ne = corner_color(x+1, y+1);
  int c_se = corner_color(x+1, y  );
  int c_sw = corner_color(x  , y  );
  int c_nw = corner_color(x  , y+1);

  // Tile index via tile packing
  int ntiles = nc * nc * nc * nc;
  int tidx   = ((c_ne * nc + c_se) * nc + c_sw) * nc + c_nw;
  int tpidx  = texture(tp, float(tidx) / float(ntiles)).s;

  // Texture coordinates for tile packing texture
  vec2 uv_tile = fract(coords);
  vec2 uv_tp   = vec2(tpidx % tpsize.x, tpidx / tpsize.x);
  vec2 uv      = (uv_tp + uv_tile) / vec2(tpsize);

  // Fetch the texture color
  texcolor = textureGrad(tpdecal, uv, dFdx(coords) / float(tpsize.x),
                                      dFdy(coords) / float(tpsize.y));
}
```

A.2 Partition Algorithm

Inspired by the FPO algorithm from Section 4.4, the following algorithm partitions a given set of points $X \in [0,1)^2$ into m equally-sized subsets Y_i such that each subset is well-distributed in the sense of high average mindists $\bar{\delta}_{Y_i}$ (see Chapter 4). The algorithm is an iterative optimization method that guarantees the strictly monotonic increase of $\sum_i \bar{\delta}_{Y_i}$ until converging at a (local) maximum.

More formally, we are interested in partitioning X such that $\sum_i \bar{\delta}_{Y_i} \to \max$ under the conditions

$$\forall\, Y_i, Y_j : Y_i \cap Y_j = \emptyset, |Y_i| = |Y_j|, i \neq j \quad \text{and} \quad \bigcup_{i=1}^{m} Y_i = X. \tag{A.1}$$

We assume that the total number of points $|X|$ divides the desired number of subsets without remainder such that each subset contains $n := |X|/m$ points.

Our algorithm is based on the observation that if we swap a pair of points $y \in Y_i$ and $y' \in Y_j$ between two arbitrary but equally-sized subsets, then

1. $\bar{\delta}_{Y_i}$ must increase if $\bar{\delta}_{Y_i - y + y'} - \bar{\delta}_{Y_i} > 0$,
2. $\bar{\delta}_{Y_j}$ must increase if $\bar{\delta}_{Y_j - y' + y} - \bar{\delta}_{Y_j} > 0$,
3. $\sum_i \bar{\delta}_{Y_i}$ must increase if $(\bar{\delta}_{Y_i - y + y'} - \bar{\delta}_{Y_i}) + (\bar{\delta}_{Y_j - y' + y} - \bar{\delta}_{Y_j}) > 0$.

This means that swapping a pair of points between two subsets is beneficial for the sum $\sum_i \bar{\delta}_{Y_i}$ as long as the gain in average mindist for one of the subsets is bigger than the (potential) loss is for the other. This is captured by the following method which performs such a beneficial swap between two subsets and returns 1 on success and 0 otherwise.

```
BENEFICIAL-SWAP($Y_i, Y_j$)
$\bar{\delta}_{Y_i}, \bar{\delta}_{Y_j}$ = average mindist of $Y_i, Y_j$
**foreach** $y$ **in** $Y_i$
    **foreach** $y'$ **in** $Y_j$
        swap $Y_i[y]$ and $Y_j[y']$
        $\bar{\delta}'_{Y_i}, \bar{\delta}'_{Y_j}$ = average mindist of $Y_i, Y_j$
        $\Delta = (\bar{\delta}'_{Y_i} - \bar{\delta}_{Y_i}) + (\bar{\delta}'_{Y_j} - \bar{\delta}_{Y_j})$
        **if** $\Delta > 0$
            **return** 1
        swap $Y_i[y]$ and $Y_j[y']$
**return** 0
```

Note that the method swaps at most one pair of points between the two subsets. Thus, calling BENEFICIAL-SWAP for any two subsets can only increase $\sum_i \bar{\delta}_{Y_i}$ and never decrease it. Also note that if two subsets stem from a partition that already fulfills the conditions (A.1), these conditions remain valid after calling BENEFICIAL-SWAP for the two subsets.

The full algorithm now starts with a random partition that fulfills (A.1) and iteratively loops over all pairs of subsets, calling the swapping method once for each pair. This concludes one full iteration of the partition algorithm. The algorithm stops when there are no more beneficial swaps for all pairs:

```
OPTIMIZED-PARTITION($X$)
1  $\{Y_i\}$ = random partition of $X$ with $|Y_i| = n$
2  $n_s = |X|$
3  while $n_s > 0$
4      $n_s = 0$
5      foreach pair $Y_i, Y_j$
6          $n_s = n_s +$ BENEFICIAL-SWAP($Y_i, Y_j$)
7  return $\{Y_i\}$
```

Figure A.1 (left) shows the result when partitioning 1024 random points into eights subsets with 128 points each. Note the good distribution of each subset with average mindists around 0.75 even though the input point set itself is not well-distributed. For better input point sets such as capacity-constrained points, partition results get better, too. Consider, for instance, the example during the discussion of trajectory splitting in Section 5.4.

Since swapping a pair of points can only increase $\sum_i \bar{\delta}_{Y_i}$, the sum must increase strictly monotonically during one iteration of OPTIMIZED-PARTITION. (See right-hand side of Figure A.1.) Consequently, the algorithm converges when $\sum_i \bar{\delta}_{Y_i}{}^{\text{new}} - \sum_i \bar{\delta}_{Y_i}{}^{\text{old}} = 0$ or, equivalently, when $n_s = 0$.

Convergence, however, is rather slow and can be significantly speed up by a variant of BENEFICIAL-SWAP that searches the most beneficial swap (instead of just any) for the subsets under consideration:

```
MOST-BENEFICIAL-SWAP($Y_i, Y_j$)
 1  $\bar{\delta}_{Y_i}, \bar{\delta}_{Y_j}$ = average mindist of $Y_i, Y_j$
 2  $\Delta_{\max} = 0$
 3  foreach $y$ in $Y_i$
 4      foreach $y'$ in $Y_j$
 5          swap $Y_i[y]$ and $Y_j[y']$
 6          $\bar{\delta}'_{Y_i}, \bar{\delta}'_{Y_j}$ = average mindist of $Y_i, Y_j$
 7          $\Delta = (\bar{\delta}'_{Y_i} - \bar{\delta}_{Y_i}) + (\bar{\delta}'_{Y_j} - \bar{\delta}_{Y_j})$
 8          if $\Delta > \Delta_{\max}$
 9              $\Delta_{\max} = \Delta, c = y, c' = y'$
10          swap $Y_i[y]$ and $Y_j[y']$
11  if $\Delta_{\max} > 0$
12      swap $Y_i[c]$ and $Y_j[c']$
13      return 1
14  return 0
```

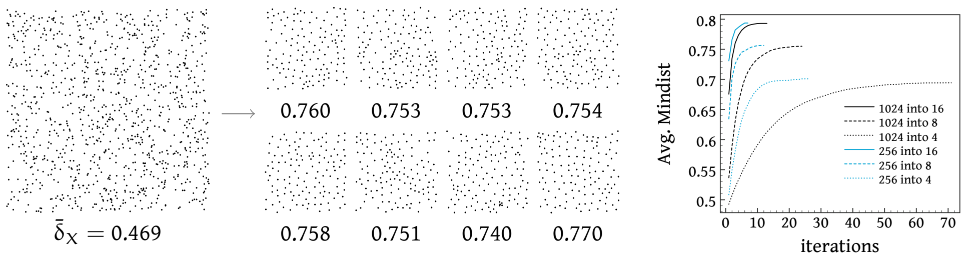


Figure A.1: (Left) 1024 random points partitioned into eight subsets. (Right) Convergence behavior based on MOST-BENEFICIAL-SWAP.

This is the method we use in practice. Compared to BENEFICIAL-SWAP, it trades runtime complexity for convergence speed which, overall, benefits the full algorithm. In addition, the method can be significantly speed up itself if we utilize Delaunay triangulations which particularly benefits the computation of the average mindists. Using DTs, it is not hard to see that the runtime complexity for one iteration of the full algorithm becomes $\mathcal{O}(m^2 n^2 \log n)$. In practice, it is near $\mathcal{O}(n^2 \log n)$ since we often have $n \gg m$ for our application scenarios.

A.3 Ranking Algorithm

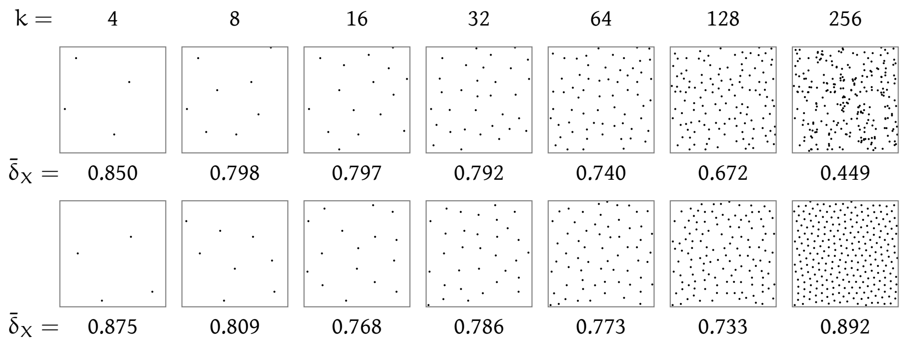
In cases where we only need parts of an optimized point set, it is sometimes useful to rank it such that it unfolds in a well-distributed manner. If we have a point set $X = \{x_0, \dots, x_{n-1}\}$, we want the subsets $\{x_0, \dots, x_k\}$, $0 \leqslant k < n$ to be well-distributed, too. This is similar to the way some radical-inverse based sequences unfold where subsequent points always “fall into the largest gap” [Kel04].

We can mimic this property by the following simple procedure which is applicable to any given point set X with at least two points.

```
RANK($X$)
1  $n = |X|$
2  for $i = 0$ to $1$
3      $r$ = random integer in $[i, n]$
4      swap $X[i]$ and $X[r]$
5  for $i = 2$ to $n - 1$
6      $f$ = index of the point in $X[i] \dots X[n-1]$ that is farthest from any point
           in $X[0] \dots X[i-1]$
7      swap $X[i]$ and $X[f]$
8  return $X$
```

This algorithm is essentially a discrete space version of the original farthest-point algorithm [ELPZ97] implemented as an in-place sorting algorithm. Starting with two randomly chosen points, it sorts the points in such a way that the “next” point is always farthest from the already chosen points. If only the first point is chosen randomly, the algorithm tends to produce subsets that resemble regular point arrangements, similar to the original FPS algorithm. Randomly choosing at least two points prevents this in practice.

If an input point set is irregular, the algorithm typically generates a ranking where the subsets $\{x_0, \dots, x_k\}$ are also irregular, with spatial and spectral characteristics similar to point sets obtained by dart throwing. The following figure shows example results for two ranked points sets, a random point set (top) and a capacity-constrained point set (bottom):



For both point sets, all intermediate steps can be considered reasonably well-distributed with most $\bar{\delta}_X > 0.7$. It is interesting to note that the average mindist decreases with increasing index k which is probably due to the increasing lack of freedom in choosing the next farthest point. It is also interesting to see that a well-distributed total set doesn't necessarily yield better subsets. The good distribution quality of the total set only comes into play at later steps.

A.4 Latin Hypercube Algorithm

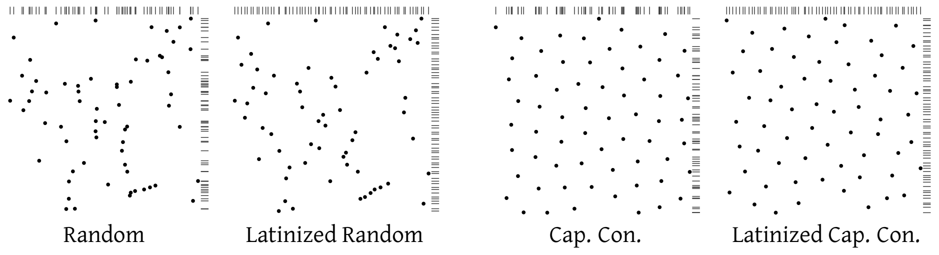
The Latin hypercube sampling property (LHS property) is a stratification property that can improve the efficiency of multi-dimensional sample sets when projected onto any of the sampling dimension's axes [Shi90]. It states that for a point set $X \in [0, 1)^s$ with n points, in each dimension there is exactly one point in every interval $\big[i/n, (i+1)/n\big)$, $0 \leqslant i < n$. In graphics, 1D projections of 2D point sets matter in situations where a 2D function mainly changes in one direction, e.g. during direct light estimation from long thin area light sources.

Saka et al. [SGB07] proposed an algorithm to add the LHS property to any given point set. If $\xi \in [0, 1)$ denotes a uniform random number, the simple algorithm can be formulated as follows.

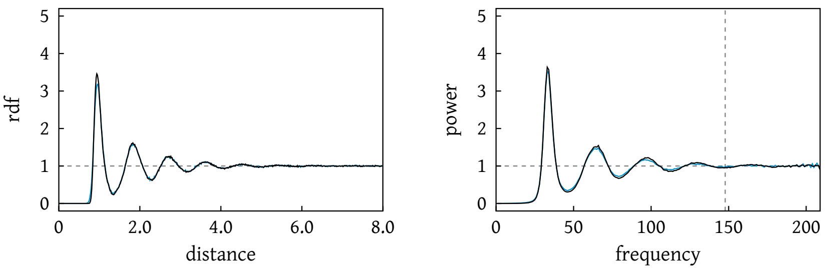
```
LATINIZE($X$)
1  $n = |X|$
2  for $k = 0$ to $s - 1$
3      sort the points in $X$ by their $k$-th coordinate in ascending order
4      for $i = 0$ to $n - 1$
5          if $X[i][k] \notin \big[i/n, (i+1)/n\big)$
6              $X[i][k] = (i + \xi)/n$
7  return $X$
```

The first []-operator denotes access to the i-th point in the array X, and the second []-operator denotes access to the k-th component of this point. It can be seen that after calling LATINIZE(X), the point set X indeed has the LHS property.

What is interesting about this algorithm is that although it significantly alters the 1D projections of a given 2D point set, it does only slightly influence the point set's 2D properties. The following figure shows an example using a random point set and a capacity-constrained point set where the lines visualize the 1D projections along each axis.



Both point sets show significantly improved 1D projections but because the potential shift is implicitly attenuated by the sort operation, their relative locations changed only slightly in two dimensions. For example, if we look at the RDF and 1D power spectrum of the capacity-constrained points, we see that they maintain their characteristic spatial and spectral properties:



The curves corresponding to the original capacity-constrained points are printed in black and the curves corresponding to the latinized variants are printed in blue. They are almost identical. Table A.1, on the other hand, shows the influence of the algorithm on our measures from Chapter 4. The most significant im-

Method	n	δ_X	$\bar{\delta}_X$	ψ_X	$\mathcal{E}'$ $(\times 10^{-3})$
Random	4096	0.010	0.465	0.365	-
Lat. Random	4096	0.018	0.469	0.364	-
FPO	4096	0.930	0.932	0.426	-
Lat. FPO	4096	0.772	0.895	0.423	-
Cap. Con.	4096	0.764	0.891	0.519	1.49
Lat. Cap. Con.	4096	0.692	0.879	0.511	2.10

Table A.1: Effect of LATINIZE on mindist, average mindist, bond-orientational order, and CCVT energy of several points.

pact can be observed for the mindist δ_X which generally decreases. The average mindist also gets slightly worse while the other variables remain stable.

Nevertheless, when capacity-constrained points are used as sample points in physically-based rendering, the improved 1D projections can be very useful in certain lighting situations.

Bibliography

[ABS99] Ancin H., Bhattacharjya A. K., Shu J.: New void-and-cluster method for improved halftone uniformity. *Journal of Electronic Imaging 8*, 104 (1999). (Cited on page 50.)

[AHA98] Aurenhammer F., Hoffmann F., Aronov B.: Minkowski-type theorems and least-squares clustering. *Algorithmica 20*, 1 (1998), 61–76. (Cited on page 46.)

[Ash01] Ashikhmin M.: Synthesizing natural textures. In *Proceedings of the 2001 Symposium on Interactive 3D graphics* (2001), ACM, pp. 217–226. (Cited on page 24.)

[Bal09] Balzer M.: *Capacity-Constrained Voronoi tessellations.* PhD thesis, University of Konstanz, 2009. (Cited on page 46.)

[BH08] Balzer M., Heck D.: Capacity-constrained Voronoi diagrams in finite spaces. In *Proceedings of the 5th International Symposium on Voronoi Diagrams in Science and Engineering* (2008), vol. 2, pp. 44–56. (Cited on page 46.)

[Bra99] Bracewell R. N.: *The Fourier transform and its applications*, 3rd ed. McGraw-Hill, 1999. (Cited on page 74.)

[Bro10] Brož P.: Hexagonal basalts at the giant's causeway, Northern Ireland. Creative Commons, 2010. (Cited on page 3.)

[BSD09] Balzer M., Schlömer T., Deussen O.: Capacity-constrained point distributions: A variant of Lloyd's method. *ACM Trans. Graph. 28* (2009), 86:1–86:8. (Cited on page 6.)

[CE54] Clark P. J., Evans F. C.: Distance to nearest neighbor as a measure of spatial relationships in populations. *Ecology 35*, 4 (1954), 445–453. (Cited on page 39.)

[CG12] Chen R., Gotsman C.: Parallel blue-noise sampling by constrained farthest point optimization. *Computer Graphics Forum 31*, 5 (2012). (Cited on page 100.)

[CLRS09] Cormen T. H., Leiserson C. E., Rivest R. L., Stein C.: *Introduction to Algorithms, 3rd Edition.* MIT Press, 2009. (Cited on pages 16 and 28.)

[Coo86] Cook R. L.: Stochastic sampling in computer graphics. In *Computer Graphics (Proc. of SIGGRAPH 86)* (1986), vol. 5, ACM, pp. 51–72. (Cited on pages 40 and 41.)

[CP76] Cranley R., Patterson T. N. L.: Randomization of number theoretic methods for multiple integration. *SIAM Journal on Numerical Analysis 13*, 6 (1976), 904–914. (Cited on page 87.)

[CSHD03] Cohen M. F., Shade J., Hiller S., Deussen O.: Wang tiles for image and texture generation. *ACM Trans. Graph. 22* (2003), 287–294. (Cited on pages 9, 11, 24, and 56.)

[Cul96] Culik II K.: An aperiodic set of 13 wang tiles. *Discrete Applied Mathematics 160* (1996), 245–251. (Cited on page 2.)

[CYC*12] Chen Z., Yuan Z., Choi Y.-K., Liu L., Wang W.: Variational blue noise sampling. *IEEE Transactions on Visualization and Computer Graphics 99* (2012). (Cited on page 100.)

[Dam09] Dammertz S.: *Rank-1 lattices in computer graphics.* PhD thesis, Ulm University, 2009. (Cited on pages 49, 65, 90, and 97.)

[Dev02] Devillers O.: On deletion in Delaunay triangulations. *Int. J. Comput. Geometry Appl. 12*, 3 (2002), 193–205. (Cited on page 53.)

[DFG99] Du Q., Faber V., Gunzburger M.: Centroidal Voronoi tessellations: Applications and algorithms. *SIAM Review 41*, 4 (1999), 637–676. (Cited on pages 41, 44, and 45.)

[dGBOD12] de Goes F., Breeden K., Ostromoukhov V., Desbrun M.: Blue noise through optimal transport. *ACM Trans. Graph. (SIGGRAPH Asia) 31* (2012). (Cited on page 100.)

[DH06] Dunbar D., Humphreys G.: A spatial data structure for fast Poisson-disk sample generation. *ACM Trans. Graph. 25* (2006), 503–508. (Cited on pages 40, 41, 42, and 84.)

[DLM04] Devroye L., Lemaire C., Moreau J.-M.: Expected time analysis for Delaunay point location. *Comput. Geom. 29*, 2 (2004), 61–89. (Cited on page 53.)

[Duc79] Duchon C. E.: Lanczos filtering in one and two dimensions. *Journal of Applied Meteorology 18* (1979), 1016–1022. (Cited on page 83.)

[Dur11] Durand F.: *A Frequency Analysis of Monte-Carlo and other Numerical Integration Schemes*. Tech. Rep. TR-2011-052, MIT CSAIL, 2011. (Cited on page 88.)

[DW85] Dippé M. A. Z., Wold E. H.: Antialiasing through stochastic sampling. In *Computer Graphics (Proc. of SIGGRAPH 85)* (1985), vol. 19, pp. 69–78. (Cited on pages 40 and 56.)

[DZP07] Dong W., Zhou N., Paul J.-C.: Optimized tile-based texture synthesis. In *Graphics Interface* (2007), pp. 249–256. (Cited on page 24.)

[EDP*11] Ebeida M. S., Davidson A. A., Patney A., Knupp P. M., Mitchell S. A., Owens J. D.: Efficient maximal Poisson-disk sampling. *ACM Trans. Graph. 30*, 4 (2011), 49:1–49:12. (Cited on page 40.)

[EF01] Efros A. A., Freeman W. T.: Image quilting for texture synthesis and transfer. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), SIGGRAPH '01, pp. 341–346. (Cited on pages 24, 26, and 35.)

[ELPZ97] Eldar Y., Lindenbaum M., Porat M., Zeevi Y. Y.: The farthest point strategy for progressive image sampling. *IEEE Trans. Image Process. 6*, 9 (1997), 1305–1315. (Cited on pages 41, 42, 50, and 105.)

[EMP*03] Ebert D. S., Musgrave F., Peachey D., Perlin K., Worley S.: *Texturing and Modeling: A Procedural Approach*, 3rd ed. Morgan Kaufmann, 2003. (Cited on page 23.)

[Esc58] Escher M. C.: Regular Division of the Plane III, Woodcut in black on wove paper, 1957–1958. (Cited on page 3.)

[Fat11] Fattal R.: Blue-noise point sampling using kernel density model. *ACM Trans. Graph. 28*, 3 (2011), 48:1–48:12. (Cited on pages 41, 43, 50, and 100.)

[FL05] Fu C.-W., Leung M.-K.: Texture tiling on arbitrary topological surfaces using Wang tiles. In *Rendering Techniques* (2005), Eurographics Association, pp. 99–104. (Cited on page 24.)

[FSG*11] Frey S., Schlömer T., Grottel S., Dachsbacher C., Deussen O., Ertl T.: Loose capacity-constrained representatives for the qualitative visual analysis in molecular dynamics. In *IEEE Pacific Visualization Symposium (PacificVis)* (2011), IEEE Computer Society, pp. 51–58. (Cited on pages 6 and 46.)

[Fyo91] Fyodorov Y.: Symmetry in the plane [in Russian]. *Zapiski Rus. Mineralog. Obščestva 28* (1891), 345–390. (Cited on page 2.)

[Ger79] Gersho A.: Asymptotically optimal block quantization. *IEEE Transactions on Information Theory 25*, 4 (1979), 373–380. (Cited on page 45.)

[GHSK08] Grünschloß L., Hanika J., Schwede R., Keller A.: (t, m, s)-nets and maximized minimum distance. *Monte Carlo and Quasi-Monte Carlo Methods 2006* (2008), 397–412. (Cited on pages 42, 65, and 90.)

[GK09] Grünschloß L., Keller A.: (t, m, s)-nets and maximized minimum distance, part II. *Monte Carlo and Quasi-Monte Carlo Methods 2008* (2009), 395–409. (Cited on pages 41, 42, 65, and 90.)

[GKP94] Graham R. L., Knuth D. E., Patashnik O.: *Concrete Mathematics.* Addison-Wesley, 1994. (Cited on page 20.)

[Gla94] Glassner A. S.: *Principles of Digital Image Synthesis.* Morgan Kaufmann, 1994. (Cited on pages 79 and 89.)

[GM09] Gamito M. N., Maddock S. C.: Accurate multidimensional Poisson-disk sampling. *ACM Trans. Graph. 29* (2009), 8:1–8:19. (Cited on page 40.)

[Gol65] Golomb S. W.: *Polyominoes.* Scribner, New York, 1965. (Cited on page 9.)

[Goo04] Goodman J. W.: *Introduction to Fourier Optics*, 3rd ed. Roberts and Company Publishers, 2004. (Cited on page 82.)

[GRK12] Grünschloß L., Raab M., Keller A.: Enumerating quasi-Monte Carlo point sequences in elementary intervals. In *Monte Carlo and Quasi-Monte Carlo Methods 2010* (2012), Springer. (Cited on pages 15, 16, and 92.)

[GS86] Grünbaum B., Shephard G. C.: *Tilings and patterns.* W. H. Freeman & Co., 1986. (Cited on page 2.)

[Hal60] Halton J. H.: On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik 2* (1960), 84–90. (Cited on pages 13 and 84.)

[HD11] Heck D., Deussen O.: *Weighted-Average Supersampling and its Relation to Filtering.* Tech. rep., University of Konstanz, April 2011. (Cited on page 82.)

[HDK01] Hiller S., Deussen O., Keller A.: Tiled blue noise samples. In *Vision, Modeling, and Visualization* (2001), Akademische Verlagsgesellschaft Aka GmbH, pp. 265–272. (Cited on pages 9, 41, and 56.)

[HJW*08] Hachisuka T., Jarosz W., Weistroffer R. P., Dale K., Humphreys G., Zwicker M., Jensen H. W.: Multidimensional adaptive sampling and reconstruction for ray tracing. *ACM Trans. Graph. 27*, 3 (2008), 33:1–33:10. (Cited on page 88.)

[HSD12a] Heck D., Schlömer T., Deussen O.: *Aliasing-Free Blue Noise Sampling*. Tech. rep., University of Konstanz, August 2012. (Cited on pages 6 and 100.)

[HSD12b] Heck D., Schlömer T., Deussen O.: Blue noise sampling with controlled aliasing. *Cond. accepted to ACM Trans. Graph.* (2012). (Cited on pages 4 and 100.)

[IPSS08] Illian J., Penttinen A., Stoyan H., Stoyan D.: *Statistical Analysis and Modelling of Spatial Point Patterns*. John Wiley and Sons Ltd., 2008. (Cited on pages 68, 69, 70, and 74.)

[JK11] Jones T. R., Karger D. R.: Linear-time poisson-disk patterns. *Journal of Graphics, GPU, and Game Tools 15* (2011), 177–182. (Cited on page 40.)

[Jon06] Jones T. R.: Efficient generation of Poisson-disk sampling patterns. *Journal of Graphics Tools 11*, 2 (2006), 27–36. (Cited on page 40.)

[Kaj86] Kajiya J. T.: The rendering equation. In *Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (1986), SIGGRAPH '86, ACM, pp. 143–150. (Cited on pages 88, 89, and 92.)

[Kap02] Kaplan C. S.: *Computer graphics and geometric ornamental design*. PhD thesis, University of Washington, 2002. (Cited on page 9.)

[KCODL06] Kopf J., Cohen-Or D., Deussen O., Lischinski D.: Recursive Wang tiles for real-time blue noise. *ACM Trans. Graph. 25* (2006), 509–518. (Cited on pages 9, 41, 56, and 100.)

[KEBK05] Kwatra V., Essa I., Bobick A., Kwatra N.: Texture optimization for example-based synthesis. *ACM Trans. Graph. 24*, 3 (2005), 795–802. (Cited on page 24.)

[Kel03] Keller A.: Strictly deterministic sampling methods in computer graphics. ACM SIGGRAPH 2003 Course Notes, 2003. (Cited on pages 16, 90, and 91.)

[Kel04] Keller A.: Myths of computer graphics. *Monte Carlo and Quasi-Monte Carlo Methods* (2004), 217–243. (Cited on pages 14 and 105.)

[Kep19] Keppleri I.: *Harmonices Mundi Libri V*. Lincii, 1619. (Cited on page 2.)

[KG12] Keller A., Grünschloß L.: Parallel quasi-Monte Carlo integration by partitioning low discrepancy sequences. In *Monte Carlo and Quasi-Monte Carlo Methods 2010* (2012), Springer. (Cited on pages 15 and 94.)

[KK02] Kollig T., Keller A.: Efficient multidimensional sampling. *Computer Graphics Forum 21*, 3 (2002), 557–563. (Cited on pages 90, 92, and 93.)

[Knu97] Knuth D. E.: *The Art of Computer Programming, Volume 2: Seminumerical Algorithms (3rd Edition)*. Addison-Wesley, 1997. (Cited on page 19.)

[KSN11] Kanamori Y., Szego Z., Nishita T.: Deterministic blue noise sampling by solving largest empty circle problems. *Journal of IIEEJ 40*, 210 (2011). (Cited on page 42.)

[KTT00] Kansal A. R., Truskett T. M., Torquato S.: Nonequilibrium hard-disk packings with controlled orientational order. *Journal of Chemical Physics 113*, 12 (2000), 4844. (Cited on pages 39 and 74.)

[LA06] Laine S., Aila T.: A weighted error metric and optimization method for antialiasing patterns. *Computer Graphics Forum 25*, 1 (2006), 83–94. (Cited on page 86.)

[Lag07] Lagae A.: *Tile-Based Methods in Computer Graphics*. PhD thesis, Departement Computerwetenschappen, Katholieke Universiteit Leuven, 2007. (Cited on page 9.)

[LB09] Loguidice B., Barton M.: *Vintage games: an insider look at the history of Grand Theft Auto, Super Mario, and the most influential games of all time*. Focal Press, 2009. (Cited on page 8.)

[LD05a] Lagae A., Dutré P.: A procedural object distribution function. *ACM Trans. Graph. 24*, 4 (October 2005), 1442–1461. (Cited on page 56.)

[LD05b] Lagae A., Dutré P.: *Template Poisson Disk Tiles*. Report CW 413, Departement Computerwetenschappen, Katholieke Universiteit Leuven, May 2005. (Cited on page 56.)

[LD06a] Lagae A., Dutré P.: An alternative for Wang tiles: Colored edges versus colored corners. *ACM Trans. Graph. 25* (2006), 1442–1459. (Cited on pages 2, 9, 11, 24, 25, 41, and 57.)

[LD06b] Lagae A., Dutré P.: Long-period hash functions for procedural texturing. In *Vision, Modeling, and Visualization* (2006), Akademische Verlagsgesellschaft Aka GmbH, pp. 225–228. (Cited on page 12.)

[LD06c] Lagae A., Dutré P.: Poisson sphere distributions. In *Vision, Modeling, and Visualization* (2006), Akademische Verlagsgesellschaft Aka GmbH, pp. 373–379. (Cited on pages 9 and 58.)

[LD06d] Lagae A., Dutré P.: *The Tile Packing Problem.* Report CW 461, Departement Computerwetenschappen, Katholieke Universiteit Leuven, Celestijnenlaan 200A, 3001 Heverlee, Belgium, August 2006. (Cited on pages 27, 28, 29, and 31.)

[LD07] Lagae A., Dutré P.: The tile packing problem. *Geombinatorics 17*, 1 (2007), 8–18. (Cited on pages 27 and 29.)

[LD08] Lagae A., Dutré P.: A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum 27*, 1 (2008), 114–129. (Cited on pages 41, 43, 50, and 56.)

[Lef08] Lefebvre S.: *Filtered Tilemaps.* Shader X6. Charles River Media, 2008. (Cited on pages 24 and 27.)

[LEQ*07] Lu A., Ebert D. S., Qiao W., Kraus M., Mora B.: Volume illustration using Wang cubes. *ACM Trans. Graph. 26* (2007). (Cited on pages 9, 24, and 100.)

[Li09] Li S. Z.: *Markov Random Field Modeling in Image Analysis*, 3rd ed. Springer, 2009. (Cited on page 23.)

[LKF*08] Lagae A., Kaplan C. S., Fu C.-W., Ostromoukhov V., Deussen O.: Tile-based methods for interactive applications. ACM SIGGRAPH 2008 Classes, 2008. (Cited on pages 3 and 9.)

[LL76] Landau L. D., Lifshitz E. M.: *Mechanics*, 3rd ed. Pergamon Press, 1976. (Cited on page 44.)

[LLLF08] Li H., Lo K.-Y., Leung M.-K., Fu C.-W.: Dual Poisson-disk tiling: An efficient method for distributing features on arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics 14* (2008), 982–998. (Cited on pages 57 and 58.)

[Llo82] Lloyd S. P.: Least square quantization in PCM. *IEEE Transactions on Information Theory 28*, 2 (1982), 129–137. (Cited on pages 40, 41, 42, 43, 44, 45, and 55.)

[LN03] Lefebvre S., Neyret F.: Pattern based procedural textures. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics* (2003), ACM, pp. 203–212. (Cited on page 24.)

[LNW*10] Li H., Nehab D., Wei L.-Y., Sander P. V., Fu C.-W.: Fast capacity constrained Voronoi tessellation. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games* (2010), p. 13. (Cited on pages 46, 47, and 100.)

[LSK*07] Laine S., Saransaari H., Kontkanen J., Lehtinen J., Aila T.: Incremental instant radiosity for real-time indirect illumination. In *Proceedings of Eurographics Symposium on Rendering 2007* (2007), Eurographics Association, pp. 277–286. (Cited on page 42.)

[LWL*09] Liu Y., Wang W., Lévy B., Sun F., Yan D.-M., Lu L., Yang C.: On centroidal Voronoi tessellation — energy smoothness and fast computation. *ACM Trans. Graph. 28*, 4 (2009), 101:1–101:17. (Cited on pages 45, 49, and 97.)

[Man08] Manske M.: Photograph of the Sheikh Lotf Allah mosque, Isfahan, Iran, 1602-1619. Creative Commons, 2008. (Cited on page 3.)

[Mar01] Marvasti F. A. (Ed.): *Nonuniform Sampling: Theory and Practice*. Kluwer Academic/Plenum Publishers, 2001. (Cited on page 82.)

[Mit87] Mitchell D. P.: Generating antialiased images at low sampling densities. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987), SIGGRAPH '87, pp. 65–72. (Cited on page 73.)

[Mit90] Mitchell D. P.: The anti-aliasing problem in ray tracing. Advanced Topics in Ray Tracing, ACM SIGGRAPH 1990 Course Notes, 1990. (Cited on page 82.)

[Mit91] Mitchell D. P.: Spectrally optimal sampling for distribution ray tracing. *Computer Graphics (Proc. of SIGGRAPH 91)* (1991), 157–164. (Cited on pages 40, 41, 42, 79, and 100.)

[MZD05] Matusik W., Zwicker M., Durand F.: Texture design using a simplicial complex of morphable textures. *ACM Trans. Graph. (Proceedings of SIGGRAPH) 24*, 3 (2005), 787–794. (Cited on page 24.)

[Nai98] Naiman A. C.: Jagged edges: when is filtering needed? *ACM Trans. Graph. 17*, 4 (1998), 238–258. (Cited on page 86.)

[NC99] Neyret F., Cani M.-P.: Pattern-based texturing revisited. In *ACM Trans. Graph. (Proceedings of SIGGRAPH)* (1999), pp. 235–242. (Cited on page 24.)

[Nie92] Niederreiter H.: *Random number generation and quasi-Monte Carlo methods*. SIAM, 1992. (Cited on pages 13 and 90.)

[NWT*05] Ng T.-Y., Wen C., Tan T.-S., Zhang X., Kim Y. J.: Generating an ω-tile set for texture synthesis. In *Proceedings of the Computer Graphics International* (2005), IEEE Computer Society, pp. 177–184. (Cited on pages 2, 9, and 24.)

[OBSC00] Okabe A., Boots B., Sugihara K., Chiu S. N.: *Spatial Tessellations: Concepts and Applications of Voronoi Diagrams*, 2nd ed. John Wiley and Sons Ltd., 2000. (Cited on pages 42, 44, and 45.)

[ODJ04] Ostromoukhov V., Donohue C., Jodoin P.-M.: Fast hierarchical importance sampling with blue noise properties. *ACM Trans. Graph. 23* (2004), 488–495. (Cited on pages 9, 41, and 56.)

[Ost07] Ostromoukhov V.: Sampling with polyominoes. *ACM Trans. Graph. 26* (2007), 78:1–78:6. (Cited on pages 9, 41, and 56.)

[Pag97] Pagés G.: A space quantization method for numerical integration. *J. Comput. Appl. Math. 89*, 1 (1997), 1–38. (Cited on page 90.)

[Pen74] Penrose R.: The role of aesthetics in pure and applied mathematical research. *Bulletin of the Institute of Mathematics and its Applications 10* (1974), 266–271. (Cited on page 2.)

[Pen78] Penrose R.: Pentaplexity: A class of nonperiodic tilings of the plane. *Eureka 39* (1978), 16–22. (Cited on pages 8 and 9.)

[PGGM09] Peytavie A., Galin E., Grosjean J., Merillou S.: Procedural generation of rock piles using aperiodic tiling. *Computer Graphics Forum 28*, 7 (2009), 1801–1809. (Cited on pages 9 and 100.)

[PH10] Pharr M., Humphreys G.: *Physically Based Rendering: From Theory To Implementation*, 2nd ed. Morgan Kaufmann Publishers Inc., 2010. (Cited on pages 37, 79, 87, 88, 91, 93, and 96.)

[PM96] Proakis J. G., Manolakis D. G.: *Digital Signal Processing*. Prentice-Hall, 1996. (Cited on pages 73 and 79.)

[Rip77] Ripley B. D.: Modelling spatial patterns. *Journal of the Royal Statistical Society. Series B (Methodological) 39*, 2 (1977), 172–212. (Cited on page 68.)

[Ros02] Ross S. M.: *Introduction to Probability Models, 10th Edition*. Academic Press, 2002. (Cited on pages 17 and 68.)

[SA12] Segal M., Akeley K.: *The OpenGL® Graphics System: A Specification*. The Khronos Group Inc., 2012. (Cited on page 101.)

[SC97] Shirley P., Chiu K.: A low distortion map between disk and square. *Journal of Graphics, GPU, and Game Tools 2*, 3 (1997), 45–52. (Cited on page 93.)

[Sch91] Schoenflies A.: *Kristallsysteme und Kristallstruktur*. Teubner, Leipzig, 1891. (Cited on page 2.)

[SCM00] Shade J., Cohen M. F., Mitchell D. P.: *Tiled Layered Depth Images*. Tech. rep., University of Washington, 2000. (Cited on pages 9, 41, and 56.)

[SD10a] Schlömer T., Deussen O.: Semi-stochastic tilings for example-based texture synthesis. *Computer Graphics Forum 29*, 4 (2010), 1431–1439. (Cited on pages 6 and 9.)

[SD10b] Schlömer T., Deussen O.: *Towards a Standardized Spectral Analysis of Point Sets with Applications in Graphics*. Tech. rep., University of Konstanz, May 2010. (Cited on page 6.)

[SD11] Schlömer T., Deussen O.: Accurate spectral analysis of two-dimensional point sets. *Journal of Graphics, GPU, and Game Tools 15*, 3 (2011), 152–160. (Cited on page 6.)

[SGB07] Saka Y., Gunzburger M., Burkardt J.: Latinized, improved LHS, and CVT point sets in hypercubes. *International Journal of Numerical Analysis and Modeling 4*, 3–4 (2007), 729–743. (Cited on pages 44, 90, and 106.)

[SGBW10] Schmaltz C., Gwosdek P., Bruhn A., Weickert J.: Electrostatic halftoning. *Computer Graphics Forum 29*, 8 (2010), 2313–2327. (Cited on pages 41, 43, 50, and 84.)

[SHD11] Schlömer T., Heck D., Deussen O.: Farthest-point optimized point sets with maximized minimum distance. In *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics* (2011), ACM, pp. 135–142. (Cited on page 6.)

[Shi90] Shirley P.: *Physically Based Lighting Calculations for Computer Graphics*. PhD thesis, University of Illinois, 1990. (Cited on page 106.)

[SM09] Shirley P., Marschner S.: *Fundamentals of Computer Graphics*, 3rd ed. A K Peters, 2009. (Cited on page 27.)

[Sob67] Sobol' I. M.: On the distribution of points in a cube and the approximate evaluation of integrals. *Zh. vychisl. Mat. mat. Fiz. 7*, 4 (1967), 784–802. (Cited on page 91.)

[Sob94] Sobol' I. M.: *A Primer for the Monte Carlo Method*. CRC-Press, 1994. (Cited on page 89.)

[ST06] Strohmer T., Tanner J.: Fast reconstruction methods for bandlimited functions from periodic nonuniform sampling. *J. Numer. Anal. 44*, 3 (2006), 1073–1094. (Cited on page 82.)

[Sta97] Stam J.: *Aperiodic Texture Mapping*. Tech. rep., ERCIM, 1997. (Cited on pages 8 and 24.)

[Tót51] Tóth L. F.: Über gesättigte Kreissysteme. *Mathematische Nachrichten 5*, 3–5 (1951), 253–258. (Cited on pages 39 and 44.)

[Tót01] Tóth G. F.: A stability criterion to the moment theorem. *Studia Scientiarum Mathematicarum Hungarica 38* (2001), 209–224. (Cited on page 45.)

[TS03] Torquato S., Stillinger F. H.: Local density fluctuations, hyperuniformity, and order metrics. *Phys. Rev. E 68* (2003), 041113. (Cited on pages 70 and 74.)

[TTD00] Truskett T. M., Torquato S., Debenedetti P. G.: Towards a quantification of disorder in materials: Distinguishing equilibrium and glassy sphere packings. *Physical Review E 62*, 1 (2000), 993. (Cited on pages 39 and 74.)

[Uli88] Ulichney R. A.: Dithering with blue noise. *Proc. IEEE 76*, 1 (jan 1988), 56–79. (Cited on page 73.)

[Uli93a] Ulichney R. A.: *Digital Halftoning*. MIT Press, 1993. (Cited on page 73.)

[Uli93b] Ulichney R. A.: Void-and-cluster method for dither array generation. *Proc. SPIE 1913* (1993), 332–343. (Cited on page 50.)

[UST06] Uche O. U., Stillinger F. H., Torquato S.: On the realizability of pair correlation functions. *Physica A 360*, 21 (2006), 21–36. (Cited on page 70.)

[vdC35] van der Corput J. G.: Verteilungsfunktionen. *Proc. Ned. Akad. v. Wet. 38* (1935), 813–821. (Cited on page 13.)

[VG95] Veach E., Guibas L. J.: Optimally combining sampling techniques for Monte Carlo rendering. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), SIGGRAPH '95, ACM, pp. 419–428. (Cited on page 92.)

[VG97] Veach E., Guibas L. J.: Metropolis light transport. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (1997), SIGGRAPH '97, ACM, pp. 65–76. (Cited on page 92.)

[Wäc07] Wächter C.: *Quasi-Monte Carlo Light Transport Simulation by Efficient Ray Tracing*. PhD thesis, Ulm University, 2007. (Cited on page 90.)

[Wal77] Walker A. J.: An efficient method for generating discrete random variables with general distributions. *ACM Transactions on Mathematical Software 3*, 3 (1977), 253–256. (Cited on page 19.)

[Wán61] Wáng H.: Proving theorems by pattern recognition II. *Bell Systems Technical Journal 40* (1961), 1–42. (Cited on pages 2 and 8.)

[WCE07] White K., Cline D., Egbert P.: Poisson disk point sets by hierarchical dart throwing. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing* (2007), pp. 129–132. (Cited on page 40.)

[Wei04] Wei L.-Y.: Tile-based texture mapping on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (2004), pp. 55–63. (Cited on pages 11, 24, and 27.)

[Wei08] Wei L.-Y.: Parallel Poisson disk sampling. *ACM Trans. Graph. 27* (2008), 20:1–20:9. (Cited on page 40.)

[WKB*02] Wald I., Kollig T., Benthin C., Keller A., Slusallek P.: Interactive global illumination using fast ray tracing. In *Proceedings of the 13th Eurographics workshop on Rendering* (2002), EGRW '02, Eurographics Association, pp. 15–24. (Cited on page 96.)

[WLKT09] Wei L.-Y., Lefebvre S., Kwatra V., Turk G.: State of the art in example-based texture synthesis. In *EG 2009 - State of the Art Reports* (2009), pp. 93–117. (Cited on pages 23 and 35.)

[WW11] Wei L.-Y., Wang R.: Differential domain analysis for non-uniform sampling. *ACM Trans. Graph. 30*, 4 (2011), 50:1–50:10. (Cited on page 74.)

[XLGG11] Xu Y., Liu L., Gotsman C., Gortler S. J.: Capacity-constrained Delaunay triangulation for point distributions. *Computers & Graphics 35*, 3 (2011), 510–516. (Cited on page 100.)

[Yel83] Yellott, Jr. J.: Spectral consequences of photoreceptor sampling in the rhesus retina. *Science 12*, 1 (1983), 382–385. (Cited on page 40.)

[YKS78] Yakowitz S., Krimmel J. E., Szidarovszky F.: Weighted Monte Carlo integration. *SIAM Journal on Numerical Analysis 15*, 6 (1978), 1289–1300. (Cited on page 89.)

[YWLA11] Yan D.-M., Wang K., Lévy B., Alonso L.: Computing 2d periodic centroidal Voronoi tessellation. In *Proceedings of the 8th International Symposium on Voronoi Diagrams in Science and Engineering* (2011), pp. 177–184. (Cited on page 45.)

[ZIK98] Zhukov S., Inoes A., Kronin G.: An ambient light illumination model. In *Rendering Techniques '98* (1998), Eurographics, pp. 45–56. (Cited on page 94.)