# Wang Tiles in Computer Graphics

# Synthesis Lectures on Computer Graphics and Animation

Wang Tiles in Computer Graphics

Ares Lagae

www.morganclaypool.com

ISBN: 9781598299656 paperback
ISBN: 9781598299663 ebook

DOI 10.2200/S000179ED1V01Y200903CGR009

A Publication in the Morgan & Claypool Publishers series
*SYNTHESIS LECTURES ON COMPUTER GRAPHICS AND ANIMATION*

Lecture #9
Series Editor: Brian A. Barsky, *University of California, Berkeley*

Series ISSN
Synthesis Lectures on Computer Graphics and Animation
Print 1933-8996 Electronic 1933-9003

# Wang Tiles in Computer Graphics

Ares Lagae
Katholieke Universiteit Leuven

*SYNTHESIS LECTURES ON COMPUTER GRAPHICS AND ANIMATION #9*

MORGAN & CLAYPOOL PUBLISHERS

## ABSTRACT

Many complex signals in computer graphics, such as point distributions and textures, cannot be efficiently synthesized and stored. This book presents tile-based methods based on Wang tiles and corner tiles to solve both these problems.

Instead of synthesizing a complex signal when needed, the signal is synthesized beforehand over a small set of Wang tiles or corner tiles. Arbitrary large amounts of that signal can then efficiently be generated when needed by generating a stochastic tiling, and storing only a small set of tiles reduces storage requirements.

A tile-based method for generating a complex signal consists of a method for synthesizing the signal over a set of Wang tiles or corner tiles, and a method for generating a stochastic tiling using the set of tiles. The method for generating a stochastic tiling using the set of tiles is independent of the signal. This book covers scanline stochastic tiling algorithms and direct stochastic tiling algorithms for Wang tiles and corner tiles. The method for synthesizing the signal over a set of tiles is dependent on the signal. This book covers tile-based methods for texture synthesis and for generating Poisson disk distributions. This book also explores several applications such as tile-based texture mapping and procedural modeling and texturing.

Although the methods for constructing a complex signal over a set of Wang tiles or corner tiles are dependent on the signal, the general idea behind these methods generalizes to other kinds of signals. The methods presented in this book therefore have the potential to make the generation and storage of almost any complex signal efficient.

## KEYWORDS

Wang tiles, corner tiles, scanline stochastic tiling, direct stochastic tiling, hash functions, tile-based texture synthesis, tile-based texture mapping, tile packing, Poisson disk distributions, sampling, object distribution, geometry instancing, procedural modeling, procedural texturing

# Contents

# Preface

This book is a spinoff project of the ACM SIGGRAPH 2008 class *Tile-Based Methods for Interactive Applications*, which is in turn a spinoff project of my PhD *Tile-Based Methods in Computer Graphics*. Many people contributed to my PhD, the SIGGRAPH 2008 class, and this book. Now the time has finally come to thank them.

I would like to thank the members of my PhD committee, especially my advisor Philip Dutré, my assessors Ronald Cools and Marc Van Barel, and Philippe Bekaert and Marc Stamminger. I would also like to thank Wang Yue, Tuen-Young Ng, and Tiow-Seng Tan.

I would like to thank my SIGGRAPH 2008 class collaborators Craig S. Kaplan, Chi-Wing Fu, Victor Ostromoukhov, Johannes Kopf, and Oliver Deussen.

I would like to thank the people from Morgan & Claypool Publishing, especially Michael Morgan and Brian Barsky, for encouraging me to write this book, Sara Kreisman for the copyediting, and C.L. Tondo for producing the final pages for this book. I am grateful to Oliver Deussen and Li-Yi Wei for comments on an early draft of this book.

I would like to thank my former and current colleagues: Frank Suykens, Vincent Masselus, Pieter Peers, Karl vom Berge, Frederik Anrys, Bart Adams, Olivier Dumont, Muath Sabha, Peter Vangorp, Toon Lenaerts, Jurgen Laurijssen, and Roeland Schoukens.

I am grateful for funding by the Katholieke Universiteit Leuven (2002–2003), funding by a PhD fellowship of the Research Foundation - Flanders (FWO) (2003–2007), and funding as a Postdoctoral Fellow of the Research Foundation - Flanders (FWO) (2007–).

Last but not least, I would like to thank my wife Celine and our family and close friends for their support throughout writing this book.

Ares Lagae
Leuven, Belgium

February 2009

CHAPTER 1

# Introduction

Computer graphics is a very diverse field of research with many applications, including film, and visual effects, advertising, car and flight simulators, architecture, scientific simulations and computer games. These applications are the driving force behind computer graphics and the continuous demand for more quality and complexity in digitally synthesized images.

A common problem in the field of computer graphics is the efficient synthesis and storage of complex signals, such as textures or point distributions. For several of these complex signals, no efficient synthesis algorithms are available, and storing large quantities of these signals is expensive. Tile-based methods based on Wang tiles and corner tiles provide a solution for both these problems.

As a simple example, consider the use of textures in interactive computer games. A commonly used technique to create the impression of a large texture is tiling a small square texture. This technique clearly avoids synthesizing and storing a large texture, but also introduces visually disturbing artifacts. The large texture is repeating and tile seams are visible. The challenge of tile-based methods is to generate a tiled complex signal as similar as possible to the original complex signal, without obvious repetition and tile seams.

This book presents high-quality tile-based methods based on Wang tiles and corner tiles. Wang tiles are unit square tiles with colored edges, and corner tiles are unit square tiles with colored corners. Wang tiles and corner tiles have a fixed orientation. A tiling is generated by placing the tiles next to each other, such that adjoining edges or corners have matching colors.

Rather than synthesizing a complex signal directly, the signal is synthesized over a small set of Wang tiles or corner tiles on forehand. Arbitrary large quantities of that signal can then efficiently be obtained when needed simply by generating a tiling. The complex signal is synthesized over a set of tiles consistently with the continuity constraints imposed by the colored edges or corners. This ensures that no tile seams are noticeable in the tiled complex signals. The tiled signals are generated using stochastic tilings. This ensures that no repetition is noticeable.

A tile-based method for generating a complex signal consists of a method for synthesizing the complex signal over a set of tiles, and a method for generating a stochastic tiling using the set of tiles. The method for synthesizing the complex signal over a set of tiles is dependent on the signal and is typically expensive. The method for generating a stochastic tiling using the set of tiles is independent of the signal and is typically inexpensive. Once the complex signal is synthesized over a set of Wang tiles or corner tiles, arbitrary large quantities of that signal can be generated very efficiently by generating a stochastic tiling. The tile sets are usually small and therefore reduce storage requirements.

Although tile-based methods are traditionally based on Wang tiles, this book promotes corner tiles as a better alternative for Wang tiles. The colored edges of Wang tiles only constrain the four

direct neighboring tiles, but not the diagonally neighboring tiles. This leads to unwanted artifacts in the tiled complex signals and complicates methods for constructing complex signals over a set of Wang tiles. Corner tiles are not subject to this problem.

This book covers efficient tiling algorithms for generating stochastic tilings using Wang tiles and corner tiles, and methods for synthesizing textures and constructing Poisson disk distributions over a set of Wang tiles and corner tiles.

Textures are ubiquitous in computer graphics, and methods for efficiently synthesizing textures are clearly of interest. Tile-based methods for texture synthesis are an interesting alternative for existing texture synthesis techniques, because the process of texture synthesis is broken up into two parts. In a first part, a texture is synthesized over a set of tiles. In a second part, an arbitrary large texture can be generated very efficiently simply by generating a tiling. Tile-based methods for texture synthesis enable fast synthesis of textures but also enable new applications such as tile-based texture mapping.

Poisson disk distributions are stochastic point distributions in which all points are separated by a minimum distance. Poisson disk distributions have several applications in computer graphics, such as sampling and object distribution. Tile-based methods for generating Poisson disk distributions are an efficient alternative to traditional methods for generating Poisson disk distributions. Tile-based methods for Poisson disk distributions enable fast generation of Poisson disk distribution but also enable new applications such as a procedural object distribution function.

Although the methods for constructing a complex signal over a set of Wang tiles or corner tiles are dependent on the signal, the general idea behind these methods generalizes to other kinds of signals. The methods presented in this book therefore have the potential to make the generation and storage of almost any complex signal efficient.

## CHAPTER OVERVIEW

This book is organized as follows. Chapter 2 introduces Wang tiles and corner tiles. Chapter 3 covers efficient algorithms for generating stochastic tilings with Wang tiles and corner tiles. This chapter surveys scanline stochastic tiling algorithms and direct stochastic tiling algorithms for Wang tiles and corner tiles, and discusses hash functions used in direct stochastic tiling algorithms. Chapter 4 covers tile-based methods for texture synthesis. This chapter shows how to synthesize a texture over a set of corner tiles, presents an efficient tile-based texture mapping algorithm running on graphics hardware, and discusses the tile packing problem. Chapter 5 covers tile-based methods for Poisson disk distributions. This chapter introduces Poisson disk distributions, shows how to generate a Poisson disk distribution over a set of corner tiles, and surveys other methods for generating Poisson disk distributions. Chapter 6 discusses several applications of Poisson disk distributions, enabled by efficient tile-based methods for generating Poisson disk distributions.

CHAPTER 2

# Wang Tiles and Corner Tiles

The tile-based methods presented in this book are based on Wang tiles, and corner tiles. This chapter introduces tilings, Wang tiles and corner tiles.

## 2.1 TILINGS

Tilings are in abundance all around us—not only man-made, but also occurring in nature. Some of the most famous examples of tilings can be seen in the Alhambra at Granada, Spain (Saladin, 1926), and in the work of the Dutch artist M. C. Escher (Escher and Locher, 1971).

A tiling is an arrangement of plane figures that fills the plane without gaps or overlaps, or its generalization to higher dimensions. Each plane figure is a tile. The set of plane figures used in the tiling is the tile set. To tile means to cover the plane with the tiles.

A tiling is periodic if a translation exists that maps the tiling to itself. If this is not the case, the tiling is non-periodic. An aperiodic tile set is a tile set that does not admit a periodic tiling. A tiling generated by an aperiodic tile set is an aperiodic tiling.

The classic work on tilings is *Tilings and Patterns* (Grünbaum and Shepard, 1986). A good introductory text on aperiodic tilings can be found in *Andrew Glassner's Notebook: Recreational Computer Graphics* (Glassner, 1999, chapter12).

## 2.2 TILINGS IN COMPUTER GRAPHICS

Most applications of tilings in computer graphics simulate tilings in the real world.

Kaplan and Salesin (2000) used isohedral tilings to provide a solution to the problem of *Escherization*: given a closed figure in the plane, find a new closed figure that is similar to the original and tiles the plane. Their system creates illustrations much like the ones by the Dutch artist M. C. Escher. Kaplan and Salesin (2004) also constructed a set of tools for exploring Islamic star patterns. For more information about computer-generated geometric art and ornament refer to Kaplan (2002). Ostromoukhov et al. presented methods for fast hierarchical importance sampling with blue-noise properties based on the Penrose tiles (Ostromoukhov et al., 2004) and polyominoes (Ostromoukhov, 2007). Hausner (2001) presented a system for generating decorative tile mosaics.

## 2.3 WANG TILES

The tiles this book focuses on are Wang tiles. A Wang tile set is a finite set of square tiles. The tiles are all the same size, and each edge of a tile has a fixed color. The colors are combined in several

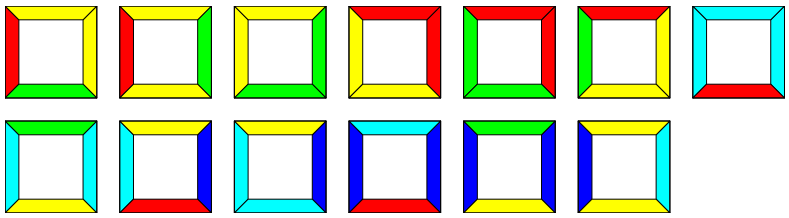


**Figure 2.1:** The smallest aperiodic Wang tile set currently known.

specified ways. The plane is tiled using arbitrary many copies of the tiles in the tile set, in such a way that adjoining edges have the same color.

Wang tiles were first proposed by Wang in 1961, and later popularized in an article in *Scientific American* (Wang, 1965). Wang presented an algorithm to decide whether a given set of Wang tiles could tile the plane. He relied on the conjecture that aperiodic tile sets, tile sets that do not admit periodic tilings, do not exist.

This conjecture was in 1966 refuted by Berger. He showed that any Turing machine can be translated into a Wang tile set, and that the Wang tile set tiles the plane if and only if the Turing machine will never halt. The halting problem is undecidable and thus so is Wang's original problem.

Berger constructed the first aperiodic tile set counting 20,426 tiles. This number was reduced repeatedly, often by well-known scientists, such as Knuth (1968). The smallest aperiodic set of Wang tiles consists of 13 tiles over 5 colors (Culik, 1996), and is shown in Figure 2.1.

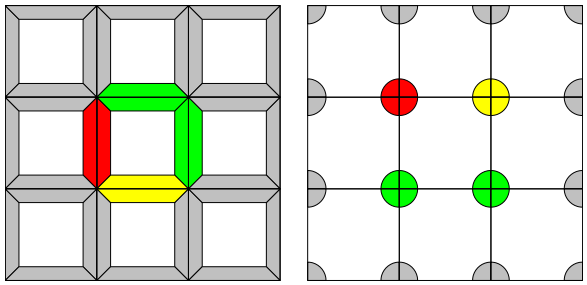
Not only Wang tiles allow the construction of aperiodic tile sets. In 1974, Penrose discovered his famous kite and dart, an aperiodic set of only two tiles. Whether a single aperiodic tile exists is still an open question.

## 2.4 WANG TILES IN COMPUTER GRAPHICS

Computer graphics is often concerned with the synthesis of complex signals. Wang tiles are an important tool to facilitate the generation of such signals. Instead of synthesizing a complex signal directly, the signal is constructed over a small set of Wang tiles, consistent with the continuity constraints imposed by the colored edges. This is usually more difficult than synthesizing the signal directly, but once the signal is synthesized over the tile set, arbitrary large quantities of this signal can be generated very efficiently by generating a tiling.

Wang tiles were introduced in the field of computer graphics by Stam (1997) who created non-repeating textures of arbitrary size using an aperiodic set of Wang tiles. Shade et al. (2000) and Hiller et al. (2001) used Wang tiles to generate Poisson disk distributions. Cohen et al. (2003) introduced a method for texture synthesis using Wang tiles, and popularized Wang tiles in the field of computer graphics. Since then, Wang tiles have been used mainly for texture synthesis and for efficient generation of Poisson disk distributions.

## 2.5 CORNER TILES AND THE CORNER PROBLEM



(a) (b)

**Figure 2.2:** The corner problem. (a) Wang tiles only enforce continuity with their four direct neighbors and do not constrain their diagonal neighbors. (b) Corner tiles enforce continuity with all their neighbors.

Wang tiles soon proved to be a valuable tool for constructing complex signals in real time. However, the colored edges of Wang tiles do not guarantee continuity of the signal near tile corners. Wang tiles do not constrain their diagonal neighbors. This is illustrated in Figure 2.2(a). Any two Wang tiles can be put diagonally to each other by adding two suitable tiles to complete the tiling. This problem, called the corner problem, complicates construction methods and causes unwanted artifacts in the generated signals.

In order to solve the corner problem, Lagae and Dutré (2006a) proposed corner tiles, square tiles with colored corners. Corner tiles are similar to Wang tiles, but their colored corners ensure continuity of the signal over both tile edges and tile corners, thus avoiding the corner problem. This is illustrated in Figure 2.2(b).

Cohen et al. (2003) first identified the corner problem. They superimpose corner markings on a Wang tile set in an attempt to solve the problem. Although this allowed them to synthesize textures with different densities, the corner problem remains: for a given corner marking, any two tiles can be put diagonally next to each other. They did not make the observation that the edge colors should be dropped altogether to adequately solve the corner problem.

Tiles with colored corners have been used previously in computer graphics by Ng et al. (2005) and Neyret and Cani (1999). Ng et al. presented a technique for synthesizing a texture over a set of corner tiles. Their technique is discussed in detail in Section 4.2. Neyret and Cani use triangular tiles with edge and corner colors to generate pattern-based textures over a triangle mesh, in the spirit of Stam (1997).

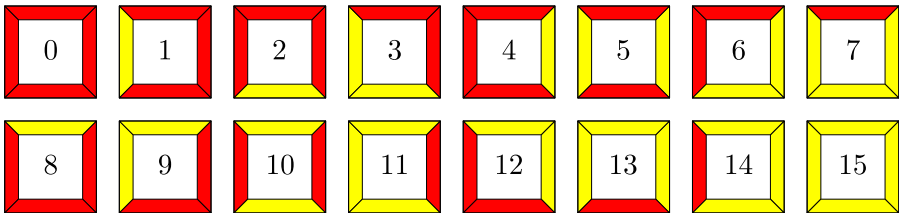



**Figure 2.3:** The complete Wang tile set over 2 colors.

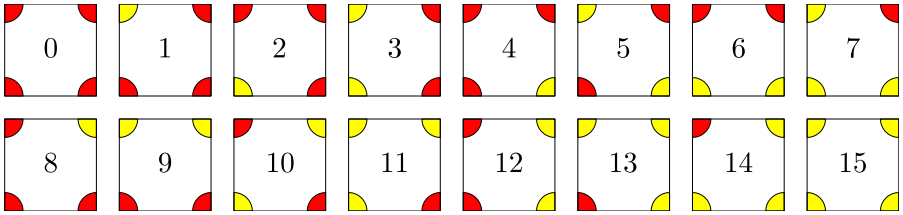



**Figure 2.4:** The complete corner tile set over 2 colors.

## 2.6 DEFINITIONS, CONVENTIONS, AND NOTATIONS

Wang tiles are unit square tiles with colored edges. The edges of a Wang tile are named after the compass headings north (N), east (E), south (S), and west (W). The colors of the edges are indicated by $c_N$, $c_E$, $c_S$, and $c_W$.

Corner tiles are unit square tiles with colored corners. The corners of a corner tile are named after the compass headings north-east (NE), south-east (SE), south-west (SW), and north-west (NW). The colors of the corners are indicated by $c_{NE}$, $c_{SE}$, $c_{SW}$, and $c_{NW}$.

A tile set is a finite set of tiles. The number of different colors used in the tile set is indicated by $C$. The $C$ colors are represented by the integers $0, 1, \ldots, C-1$. All illustrations in this book use the colors red, yellow, green, cyan, and blue for, respectively, 0, 1, 2, 3, and 4.

A complete tile set contains a tile for every possible combination of four edge or corner colors. A complete set of Wang tiles or corner tiles over $C$ colors therefore counts $C^4$ tiles. Figure 2.3 shows the complete Wang tile set over two colors, and Figure 2.4 shows the complete corner tile set over two colors. A complete set of Wang tiles or corner tiles over 2, 3, 4, 5, 6, 7, and 8 colors consist of 16, 81, 256, 625, 1,296, 2,401, and 4,096 tiles.

A tiling is constructed by placing the tiles next to each other such that adjoining edges or corners have matching colors. Each tile in the tile set can be used arbitrarily many times. The tiles are placed with their corners on the integer lattice points. By convention, the tile coordinates are the coordinates of the lower left corner of the tile. Figure 2.5 shows a tiling with the complete Wang tile set over three colors, and Figure 2.6 shows a tiling with the complete corner tile set over three colors.

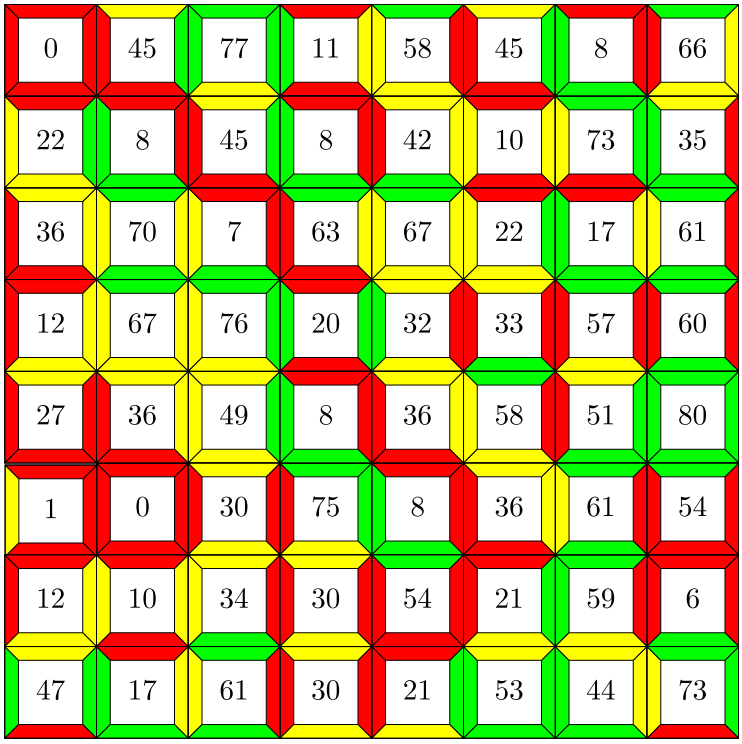



**Figure 2.5:** A tiling with the complete Wang tile set over 3 colors.

The horizontal edges and vertical edges of Wang tiles are independent. This allows Wang tile sets that use a different number of colors for horizontal and vertical edges. The number of colors used for horizontal edges is indicated by $C_h$, and the number of colors used for vertical edges is indicated by $C_v$. A complete Wang tile set over $C_h$ horizontal colors and $C_v$ vertical colors consist of $(C_h C_v)^2$ tiles. If $C_h$ colors are used for horizontal edges and $C_v$ colors for vertical edges, then the number of colors used in the tile set is $\max(C_h, C_v)$. In this book, $C_h$ is always a subset of $C_v$, or vice versa. However, not everyone follows this convention. For example, Cohen et al. (2003) use a tile set with red and green for horizontal edges and blue and yellow for vertical edges. Despite the fact that four different colors are used, this is a tile set over two colors.

## 2.7 ENUMERATING WANG TILE SETS AND CORNER TILE SETS

For efficiently manipulating Wang tiles and corner tiles, an enumeration of the tiles is needed. This book uses the following scheme.

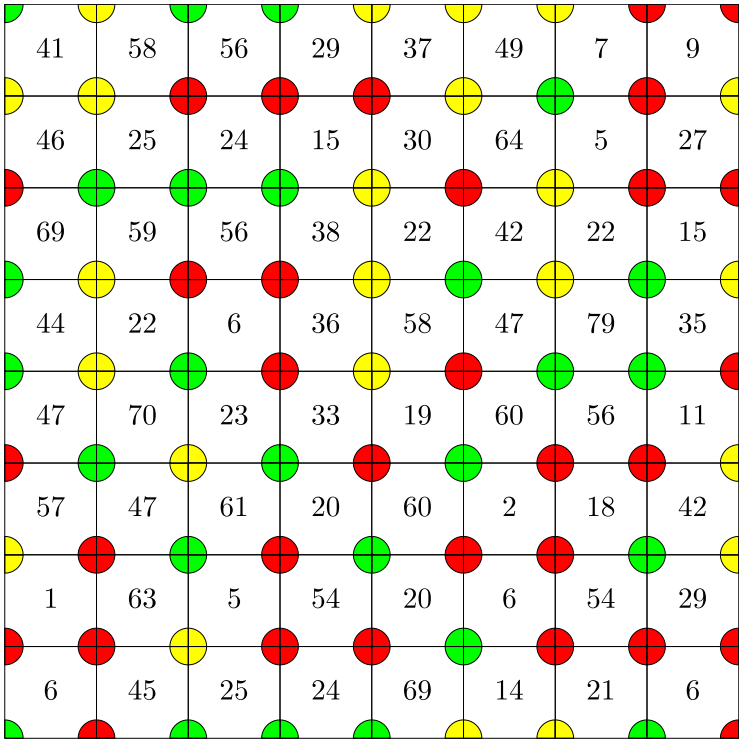



**Figure 2.6:** A tiling with the complete corner tile set over 3 colors.

Corner tiles are uniquely determined by their corner colors $c_{NE}, c_{SE}, c_{SW}$, and $c_{NW}$. Corner tiles can thus be represented as the 4-digit base-$C$ numbers $c_{NE}c_{SE}c_{SW}c_{NW}$, or as the decimal integers $0, 1, \ldots, C^4 - 1$. A base conversion switches between the corner colors and the tile index.

The tile index $i$ of the corner tile with corner colors $c_{NE}, c_{SE}, c_{SW}$, and $c_{NW}$ is given by

$$i = ((c_{NE}C + c_{SE})C + c_{SW})C + c_{NW}. \tag{2.1}$$

The corner colors $c_{NE}, c_{SE}, c_{SW}$, and $c_{NW}$ of the corner tile with tile index $i$ are given by

$$\begin{aligned} c_{NE} &= (i/C^3) \,\%\, C \\ c_{SE} &= (i/C^2) \,\%\, C \\ c_{SW} &= (i/C) \,\%\, C \\ c_{NW} &= i \,\%\, C \end{aligned} \tag{2.2}$$

where % is the modulo division, and / is the integer division. This conversion of base can be implemented using only three modulo divisions and three integer divisions

$$\begin{aligned} c_{NW} &\leftarrow i \,\%\, C \\ i &\leftarrow i / C \\ c_{SW} &\leftarrow i \,\%\, C \\ i &\leftarrow i / C \\ c_{SE} &\leftarrow i \,\%\, C \\ i &\leftarrow i / C \\ c_{NE} &\leftarrow i \end{aligned} \tag{2.3}$$

where $\leftarrow$ indicates assignment.

When the number of colors is a power of two, the base conversions can be implemented very efficiently using bitwise operators.

A similar scheme is used for Wang tiles. Wang tiles are uniquely determined by their edge colors $c_N$, $c_E$, $c_S$, and $c_W$. Wang tiles can thus be represented as the 4-digit base-$C$ numbers $c_N c_E c_S c_W$, or as the decimal integers $0, 1, \ldots, C^4 - 1$.

This enumeration scheme is illustrated in Figures 2.3, 2.4, 2.5, and 2.6.

## 2.8 CORNER TILES AS WANG TILES

Corner tiles are closely related to Wang tiles. In fact, every corner tile set can be transformed into an equivalent Wang tile set. This means that corner tiles actually are Wang tiles. However, in most situations it is easier to maintain the distinction between corner tiles and Wang tiles.

Transformed a corner tile set into a Wang tile set is done by encoding any combination of two corner colors into an edge color. This operation squares the number of colors. Figure 2.7 shows the Wang tile set equivalent to the complete corner tile set over two colors, shown in Figure 2.4. This is

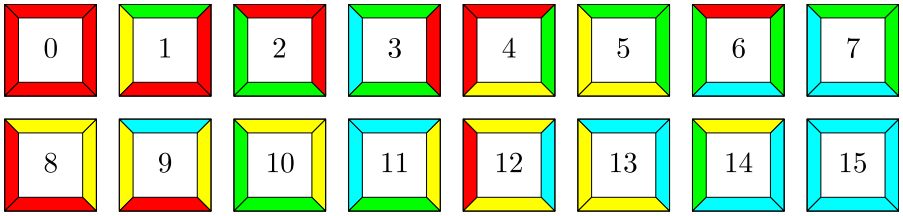



**Figure 2.7:** The Wang tile set equivalent to the corner tile set over 2 colors.

a Wang tile set of 16 tiles over 4 colors. This technique can be used to construct an aperiodic set of Wang tiles from an aperiodic set of corner tiles (Lagae et al., 2006).

In general, a Wang tile set cannot be transformed into an equivalent corner tile set, and a Wang tile set equivalent to a corner tile set is not subject to the corner problem. This shows that corner tiles are in some way more restrictive than Wang tiles.

## 2.9 DOMINOES, WANG CUBES, AND CORNER CUBES

Wang tiles and corner tiles easily generalize to arbitrary dimension. The one-dimensional and three-dimensional equivalents are especially useful.

In one dimension, Wang tiles and corner tiles are dominoes. These gaming pieces are well known and have been studied extensively in the field of recreational mathematics (Ball, 1926). Dominoes are used in Section 4.4 to solve the Wang tile packing problem.

In three dimensions, Wang tiles and corner tiles become Wang cubes and corner cubes. Wang cubes have received some attention in the field of discrete mathematics and in computer graphics. Culik and Kari (1995) showed that an aperiodic set of 21 Wang cubes exists. Lu and coworkers used Wang cubes for example-based volume illustrations (Lu and Ebert, 2005; Lu et al., 2007). Corner cubes can be used to efficiently generate Poisson sphere distributions, the 3D equivalent of Poisson disk distributions (Lagae and Dutré, 2006c).

Note that identifying tiles by their corner colors is more straightforward than identifying tiles by their edge or face colors, especially in higher dimension.

CHAPTER 3

# Tiling Algorithms for Wang Tiles and Corner Tiles

After synthesizing a signal over a set of Wang tiles or corner tiles, arbitrary large quantities of that signal can be generated very efficiently by generating a tiling. Because periodicity in the signal is visually disturbing, applications in computer graphics require random or stochastic tilings, such as the ones shown in Figures 2.5 and 2.6. Stochastic tilings are inherently non-periodic. The stronger mathematical guarantee of provable aperiodicity is not that useful in computer graphics. A mathematical proof of aperiodicity does not necessarily provide an algorithm for actually generating the aperiodic tiling, and even if it does, these algorithms are often very complex. Also, aperiodicity does not imply small scale non-periodicity. Aperiodic tilings are sometimes very structured. For these reasons, stochastic tilings are better suited for most applications in computer graphics.

This chapter is organized as follows. Section 3.1 presents scanline stochastic tiling algorithms, and Section 3.2 presents direct stochastic tiling algorithms, two classes of stochastic tiling algorithms. Section 3.3 discusses hash functions defined over the integer lattice, an essential ingredient of direct stochastic tiling algorithms.

## 3.1 SCANLINE STOCHASTIC TILING ALGORITHMS

Scanline stochastic tiling algorithms are stochastic tiling algorithms that generate a tiling by placing tiles in scanline order. This section discusses a scanline stochastic tiling algorithm for Wang tiles and a scanline stochastic tiling algorithm for corner tiles.

### 3.1.1 A SCANLINE STOCHASTIC TILING ALGORITHM FOR WANG TILES

In 2003, Cohen et al. presented a scanline stochastic tiling algorithm for Wang tiles.

The Wang tiles are placed in scanline order, from west to east, and from north to south. A random Wang tile is selected for the NW corner. The first row is completed by adding Wang tiles for which the color of the W edge corresponds to the color of the E edge of the Wang tile to the left. The leading Wang tile of each new row is selected so that its N edge matches the S edge of the Wang tile above. The row is completed by choosing Wang tiles for which the N and W edges match the S and E edges from the Wang tiles above and to the left. This is illustrated in Figure 3.1.

To ensure a non-periodic tiling, the Wang tile set is constructed such that there are two Wang tiles for each combination of N and W edge colors. Each time a Wang tile has to be selected, the choice is made at random. A Wang tile set over $C$ colors will contain $2C^2$ Wang tiles, since there are $C^2$ combinations of N and W edge colors. A Wang tile set obtained this way is called a compact

Wang tile set, because it is significantly smaller than a complete Wang tile set. Figure 3.2 shows a compact Wang tile set over two colors.

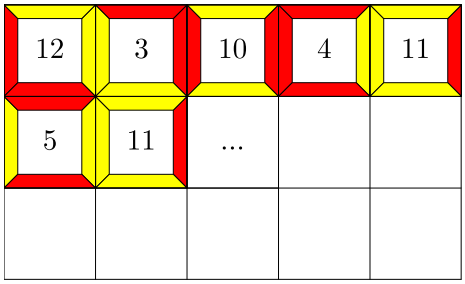



**Figure 3.1:** A scanline stochastic tiling algorithm for Wang tiles.

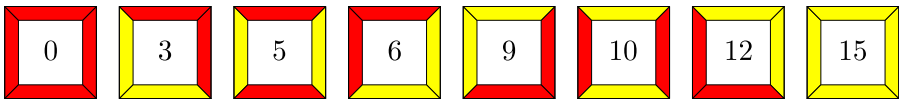



**Figure 3.2:** A compact Wang tile set over 2 colors.

Compact Wang tile sets are useful when the size of the Wang tile set should be minimized. A compact Wang tile set is quadratic in the number of colors, while a complete Wang tile set is quartic in the number of color. For 2, 3, 4, 5, 6, 7, and 8 colors, a compact Wang tile set counts 8, 18, 32, 50, 98, 128 Wang tiles while a complete Wang tile set consists of 16, 256, 625, 1,296, 2,401, and 4,096 Wang tiles.

### 3.1.2 A SCANLINE STOCHASTIC TILING ALGORITHM FOR CORNER TILES

In 2006, Lagae and Dutré (2006a) presented a scanline stochastic tiling algorithm for corner tiles. The scanline stochastic tiling algorithm for corner tiles is very similar to the scanline stochastic tiling algorithm for Wang tiles.

The corner tiles are placed in scanline order, from west to east, and from north to south. A random corner tile is selected for the NW corner. The first row is completed by adding corner tiles for which the color of the NW corner and the color of the SW corner corresponds to the color of the NE corner and the color of the SE corner of the corner tile to the left. The leading corner tile of each new row is selected so that its NW and NE corners match the SW and SE corners of the corner tile above. The row is completed by choosing corner tiles for which the NE, NW, and SW corners match the SE and SW corners from the corner tile above and the NE and SE corners from the corner tile to the left. This is illustrated in Figure 3.3.

To ensure a non-periodic tiling, the corner tile set is constructed such that there are two corner tiles for each combination of NE, NW, and SW corner colors. Each time a corner tile has to be

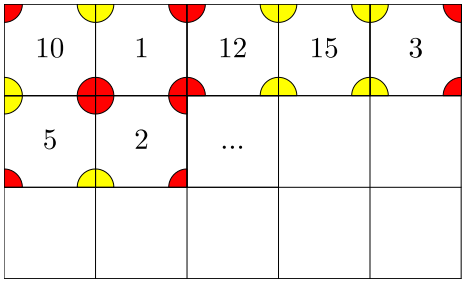



**Figure 3.3:** A scanline stochastic tiling algorithm for corner tiles.

selected, the choice is made at random. A corner tile set over $C$ colors will contain $2C^3$ corner tiles, since there are $C^3$ combinations of NE, NW, and SW corner colors. A corner tile set obtained this way is called a compact corner tile set, because it is significantly smaller than a complete corner tile set.

Compact corner tile sets are useful when the size of the corner tile set should be minimized. A compact corner tile set is cubic in the number of colors, while a complete corner tile set is quartic in the number of color. For 2, 3, 4, 5, 6, 7, and 8 colors, a compact corner tile set counts 16, 54, 128, 250, 432, 686, and 1,024 corner tiles while a complete corner tile set consists of 16, 256, 625, 1,296, 2,401, and 4,096 corner tiles.

Note that the complete corner tile set over two colors and the compact corner tile set over two colors are identical. Also note that compact Wang tile sets are smaller than compact corner tile sets.

## 3.2 DIRECT STOCHASTIC TILING ALGORITHMS

Several applications, such as tile-based texture mapping (see Section 4.3), and the procedural object distribution texture basis functions (see Section 6.5), require local evaluation of the tiling. In order to evaluate the tiling at a specific location, scanline stochastic tiling algorithms must construct and store the tiling up to that point. This is clearly not efficient. Direct stochastic tiling algorithms address this problem. Direct stochastic tiling algorithms are able to compute which tile is at a given location without explicitly constructing and storing the tiling up to that point.

Direct stochastic tiling algorithms are based on hash functions defined over the integer lattice. These hash functions associate a random color with each lattice point. A tiling is obtained by transforming the colored lattice. The hash functions that are used are efficient in time and space, and the transformation can be performed locally. This enables efficient direct stochastic tiling algorithms.

This section discusses several direct stochastic tiling algorithms for Wang tiles and a direct stochastic tiling algorithm for corner tiles. For clarity, the direct stochastic tiling algorithm for corner tiles is discussed first. The hash functions used in the direct stochastic tiling algorithms are discussed in Section 3.3.

### 3.2.1 A DIRECT STOCHASTIC TILING ALGORITHM FOR CORNER TILES

In 2006, Lagae and Dutré (2006a) proposed a direct stochastic tiling algorithm for corner tiles.

Corner tiles are placed with their corners on the integer lattice points. The coordinates of a corner tile are the coordinates of the integer lattice point corresponding to the lower left or SW corner. To generate a tiling with a corner tile set over $C$ colors, the direct stochastic tiling algorithm for corner tiles uses a hash function $h$ defined over the integer lattice. This hash function associates a random color $h(x, y) \in \{0, 1, \ldots, C-1\}$ with each location $(x, y)$. The corner colors $c_{NE}$, $c_{SE}$, $c_{SW}$, and $c_{NW}$ of the corner tile at tile coordinates $(x, y)$ are given by $h(x+1, y+1)$, $h(x+1, y)$, $h(x, y)$, and $h(x, y+1)$. The index of the corner tile can now be obtained using Equation 2.1. The direct stochastic tiling algorithm for corner tiles is illustrated in Figure 3.4.

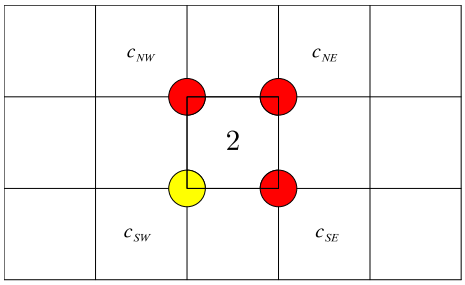



**Figure 3.4:** A direct stochastic tiling algorithm for corner tiles.

Because the color of each corner is chosen at random, the direct stochastic tiling algorithm for corner tiles results in a complete corner tile set over $C$ colors.

The direct stochastic tiling algorithm for corner tiles is very efficient. It requires only four hash function evaluations.

### 3.2.2 DIRECT STOCHASTIC TILING ALGORITHMS FOR WANG TILES

This subsection discusses a direct stochastic tiling algorithm for Wang tiles using two hash functions, a direct stochastic tiling algorithm for compact sets of Wang tiles, and a direct stochastic tiling algorithm for Wang tiles using a hash function defined at the tile edges.

#### 3.2.2.1 A Direct Stochastic Tiling Algorithm for Wang Tiles using Two Hash Functions

In 2005, Lagae and Dutré (2005a) proposed a direct stochastic tiling algorithm for Wang tiles using two hash functions.

Wang tiles are placed with their corners on the integer lattice points. The coordinates of a Wang tile are the coordinates of the integer lattice point corresponding to the lower left corner. To generate a tiling with a Wang tile set over $C$ colors, the direct stochastic tiling algorithm for Wang tiles uses two hash functions, $h_h$ and $h_v$, defined over the integer lattice. These hash functions associate a pair of random colors $(h_h(x, y), h_v(x, y)) \in \{0, 1, \ldots, C-1\}^2$ with each location $(x, y)$. The hash function $h_h$ is used to compute the color of the horizontal edges, and the hash function $h_v$ is used

to compute the color of the vertical edges. The edge colors of a Wang tile are computed as the sum modulo $C$ of the random colors associated with the corners of the Wang tile. If the pair of random colors associated with the NE, SE, SW, and NW corner of the Wang tile at tile coordinates $(x, y)$ are $(c^h_{NE}, c^v_{NE})$, $(c^h_{SE}, c^v_{SE})$, $(c^h_{SW}, c^v_{SW})$, and $(c^h_{NW}, c^v_{NW})$, then the edge colors $c_N, c_E, c_S$ and $c_W$ are given by $(c^h_{NW} + c^h_{NE})\%C$, $(c^v_{NE} + c^v_{SE})\%C$, $(c^h_{SE} + c^h_{SW})\%C$, $(c^c_{SW} + c^v_{NW})\%C$. The index of the Wang tile can now be obtained from the edge colors. The direct stochastic tiling algorithm for Wang tiles using two hash functions is illustrated in Figure 3.5.

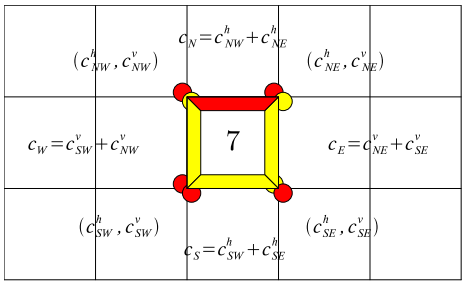



**Figure 3.5:** A direct stochastic tiling algorithm for Wang tiles using two hash functions. The first hash function is used for the horizontal edges, and the second hash function is used for the vertical edges.

Because the color of each edge is chosen at random, the direct stochastic tiling algorithm for Wang tiles using two hash functions results in a complete Wang tile set over $C$ colors.

The direct stochastic tiling algorithm for Wang tiles is more expensive than the direct stochastic tiling algorithm for corner tiles. It requires eight hash function evaluations, four integer additions, and four integer modulo divisions.

The direct stochastic tiling algorithm can easily be modified for tile sets over a different number of colors for horizontal edges $C_h$ and vertical edges $C_v$ by generating pairs of random colors $(h_h(x, y), h_v(x, y)) \in \{0, 1, \ldots, C_h - 1\} \times \{0, 1, \ldots, C_v - 1\}$ and performing edge computations for horizontal and vertical edges modulo $C_h$ and $C_v$. This modified direct stochastic tiling algorithm for Wang tiles also results in a complete Wang tile set.

An algorithm similar in spirit was proposed concurrently by Wei (2004).

#### 3.2.2.2 A Direct Stochastic Tiling Algorithm for Compact Sets of Wang Tiles

In 2005, Lagae and Dutré (2005a) also proposed a direct stochastic tiling algorithm for compact sets of Wang tiles.

The direct stochastic tiling algorithm for compact sets of Wang tiles is very similar to the direct stochastic tiling algorithm for Wang tiles using two hash functions. However, to generate a tiling with a compact Wang tile set over $C$ colors only a single hash function $h$ is used. This hash function associates a random color $h(x, y) \in \{0, 1, \ldots, C - 1\}$ with each location $(x, y)$ The edge colors of a Wang tile are computed as the sum modulo $C$ of the random colors associated with the corners of the Wang tile. If the random color associated with the NE, SE, SW, and NW corner of

the Wang tile at tile coordinates $(x, y)$ is $c_{NE}, c_{SE}, c_{SW}$, and $c_{NW}$, then the edge colors $c_N, c_E, c_S$ and $c_W$ are given by $(c_{NW} + c_{NE})\%C$, $(c_{NE} + c_{SE})\%C$, $(c_{SE} + c_{SW})\%C$, $(c_{SW} + c_{NW})\%C$. The direct stochastic tiling algorithm for compact sets of Wang tiles is illustrated in Figure 3.6.

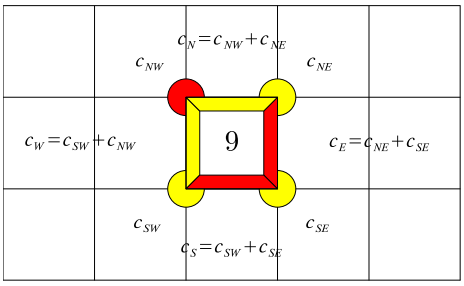



**Figure 3.6:** A direct stochastic tiling algorithm for compact sets of Wang tiles.

This algorithm results in a compact set of $C^3$ Wang tiles over $C$ colors. Note that, except for two colors, these compact Wang tile sets are different from the compact Wang tile sets produced by the scanline stochastic tiling algorithms for Wang tiles, that counted $2C^3$ tiles.

#### 3.2.2.3 A Direct Stochastic Tiling Algorithm for Wang Tiles using a Hash Function Defined at the Tile Edges

The direct stochastic tiling algorithm for Wang tiles using a hash function defined at the tile edges is a more efficient variant of the direct stochastic tiling algorithm for Wang tiles using two hash functions.

Compared with the direct stochastic tiling algorithm for corner tiles, direct stochastic tiling algorithm for Wang tiles are more complicated. This is because the lattice defined by the colored corners of corner tiles and the lattice over which the hash function is defined are both square lattices. In contrast, the lattice defined by the colored edges of Wang tiles is a diamond lattice (a square lattice rotated 45 degrees). The key observation of the direct stochastic tiling algorithm for Wang tiles using a hash function defined at the tile edges is that a diamond lattice can be obtained by discarding points in a square lattice.

Wang tiles are placed with their corners on the integer lattice points. The coordinates of a Wang tile are the coordinates of the integer lattice point corresponding to the lower left. To generate a tiling with a Wang tile set over $C$ colors, the direct stochastic tiling algorithm for Wang tiles uses a hash function $h$ defined over a square lattice twice as dense as the integer lattice. The edge colors $c_N, c_E, c_S$, and $c_W$ of the Wang tile at tile coordinates $(x, y)$ are given by $h(2x + 1, 2y + 2)$, $h(2x + 2, 2y + 1)$, $h(2x + 1, 2y)$, and $h(2x, 2y + 1)$. The remaining five random colors are ignored. The index of the Wang tile can now be obtained from the edge colors. The direct stochastic tiling algorithm for Wang tiles using a hash function defined at the tile edges is illustrated in Figure 3.7.

Because the color of each edge is chosen at random, the direct stochastic tiling algorithm for Wang tiles using a hash function defined at the tile edges results in a complete Wang tile set

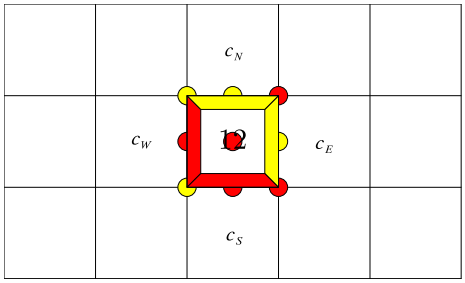



**Figure 3.7:** A direct stochastic tiling algorithm for Wang tiles using a hash function defined at the tile edges.

over $C$ colors. In contrast with the direct stochastic tiling algorithm for Wang tiles using two hash functions, the direct stochastic tiling algorithm for Wang tiles using a hash function defined at the tile edges only requires four hash function evaluations.

## 3.3 HASH FUNCTIONS

Hash functions defined over the integer lattice are an essential ingredient of the direct stochastic tiling algorithms discussed in Section 3.2. Hash functions are also used extensively in procedural modeling and texturing (Perlin, 1985; Ebert et al., 2002). This section discusses traditional hash functions based on permutation tables and proposes long-period hash functions based on permutation tables.

### 3.3.1 TRADITIONAL HASH FUNCTIONS BASED ON PERMUTATION TABLES

Hash functions used in procedural modeling and texturing are typically based on permutation tables. A permutation table $P$ of size $N$ contains a random permutation of the integers $\{0, 1, \ldots, N-1\}$. The permutation table is zero-based, the first element is $P[0]$.

A random permutation of the elements $\{0, 1, \ldots, N-1\}$ can be generated by starting with the permutation $\{0, 1, \ldots, N-1\}$, and then exchanging the $i$th element with an element randomly selected from the first $i$ elements, for $i \in 0, 1, \ldots, N-2$. Note that, for $i$ equal to $N-1$, this operation has no effect.

A one-dimensional hash function is defined as

$$h(x) = P[x\%N], \tag{3.1}$$

where $x$ is an integer, $P[i]$ is the $(i+1)$th element of $P$, and $\%$ is the modulo division. This hash function is a periodic function with period $N$. The range of this hash function is $N$.

A two-dimensional hash function can be defined using two permutation tables, $P_x$ and $P_y$, of size $N$

$$h(x, y) = (P_x[x\%N] + P_y[y\%N])\%N, \tag{3.2}$$

where $x$ and $y$ are integers. However, in order to avoid the storage of two permutation tables, the same permutation table is generally used twice

$$h(x, y) = P[(P[x\%N] + y)\%N]. \tag{3.3}$$

This hash function is a periodic function with period $(N, N)$. The range of this hash function is $N$.

A three-dimensional hash function is defined similarly as

$$h(x, y, z) = P[(P[(P[x\%N] + y)\%N]) + z)\%N], \tag{3.4}$$

where $x$, $y$, and $z$ are integers. This hash function is a periodic function with period $(N, N, N)$. The range of this hash function is $N$.

This family of hash functions is used extensively in procedural modeling and texturing, and is also used in the direct stochastic tiling algorithms. The hash functions are easy to implement and efficient to evaluate.

### 3.3.2 LONG-PERIOD HASH FUNCTIONS BASED ON PERMUTATION TABLES

The period of hash functions based on permutation tables is short. This causes unwanted repetition artifacts in procedural textures and tilings. Increasing the period is easy but also expensive, because the length of the period is equal to the size of the permutation table.

In 2006, Lagae and Dutré (2006b) proposed long-period hash functions based on permutation tables. The key observation for constructing long-period hash functions is that hash functions based on permutation tables are periodic functions, and that the addition of periodic functions yields a new periodic function with a larger period.

This subsection defines long-period hash functions, studies the period and range, distribution and efficiency of long-period hash functions, and formulates guidelines for designing long-period hash functions.

#### 3.3.2.1 Definition

A one-dimensional long-period hash function is defined as

$$h(x) = \left( \sum_{i=0}^{M-1} P_i[x\%N_i] \right) \%N_j, \tag{3.5}$$

where $x$ is an integer, $P_0, P_1, \ldots, P_{M-1}$ are $M$ permutation tables with size $N_0, N_1, \ldots, N_{M-1}$, and $N_j$ is one of these sizes.

A two-dimensional long-period hash function is defined as

$$h(x, y) = \left( \sum_{i=0}^{M-1} P_i[(P_i[x\%N_i] + y)\%N_i] \right) \%N_j, \tag{3.6}$$

where $x$ and $y$ are integers

A three-dimensional long-period hash function is defined similarly as

$$h(x, y, z) = \left( \sum_{i=0}^{M-1} P_i[(P_i[(P_i[x\%N_i] + y)\%N_i] + z)\%N_i] \right) \%N_j, \tag{3.7}$$

where $x$, $y$, and $z$ are integers. These long-period hash functions are also called combined hash functions.

The period of a combined hash function is the least common multiple of the periods of the combining hash functions. In order to maximize the period of the combined hash function, the periods of the combining hash functions should be relatively prime. The range of the combined hash function is determined by the final modulo divisor $N_j$, which is one of $N_0, N_1, \ldots, N_{M-1}$. Note that, in contrast with traditional hash functions, the range and period of the combined hash functions are different.

A similar technique was used in 1988 by L'Ecuyer to construct a long-period pseudo-random number generator by combining several shorter-period linear congruential generators. However, with the advent of recent pseudo-random number generators (Matsumoto and Nishimura, 1998), this technique has largely become obsolete in its original context.

#### 3.3.2.2 Distribution

Traditional hash functions produce uniformly distributed values over the integer lattice, and most applications of these hash functions rely on this property. The following theorem shows that combined hash functions will also produce uniformly distributed values.

**Theorem 3.1.** *If $X_0, X_1, \ldots, X_{N-1}$ are $N$ independent discrete random variables, such that $X_0$ is uniform between 0 and $d-1$, where $d$ is a positive integer, then*

$$X = \left( \sum_{i=0}^{N-1} X_i \right) \%d \tag{3.8}$$

*follows a discrete uniform probability law between 0 and $d-1$.*

Note that there are no requirements on the distribution of the random variables $X_1, X_2, \ldots, X_{N-1}$. This theorem was first hinted at by Wichmann and Hill (1982), and later proved by L'Ecuyer (1988).

For long-period hash functions, all combining hash functions are uniformly distributed. Therefore, the final modulo divisor $N_j$ can be any one of the sizes of the permutation tables of the combining hash functions $N_0, N_1, \ldots, N_{M-1}$. However, some care must be taken in selecting the appropriate permutation table sizes. Suppose a combined hash function is built from a small permutation table of size $N_s$ and a large permutation table of size $N_l$, with $N_s \ll N_l$. The period of the combined hash function is $N_s N_l$. However, if the final modulo divisor is $N_l$, then the period will contain $N_s$ almost

identical parts. This is because the range of the hash function using the permutation table of size $N_s$ is very small compared to the range of the hash function using the permutation table of size $N_l$. If the final modulo divisor is $N_s$, this will not be the case. Therefore, the sizes of the permutation tables should not differ too much.

#### 3.3.2.3 Efficiency

The time needed to evaluate a combined hash function is roughly proportional to the number of combining hash functions. This does not mean that an application that uses a long-period hash function consisting of $N$ combining hash functions will be $N$ times slower than the same application using a traditional hash function. In most applications, the evaluation of the hash function is only a small part of the total computation time. Also note that combined hash functions typically have a smaller memory footprint, which improves the cache efficiency of lookups in the permutation tables.

#### 3.3.2.4 Design

The design of a combined hash function is determined by several factors: the required range of the hash function, the period of the hash function, the memory footprint of the combined permutation tables, and the time needed to evaluate the hash function. The strategy outlined below is recommended to design a long-period hash function.

First, determine the required range of the hash function. This fixes the final modulo divisor and thus the size of one permutation table. If the range of the hash function is too small, or if the range should be adjustable, then use a multiple of the range of the hash function and apply an additional final modulo divisor.

Next, choose a number of permutation table sizes for the rest of the combining hash function. The sizes should not differ too much, in order to ensure a uniform distribution. The sizes should also be relatively prime in order to maximize the period. An easy way to choose the permutation table sizes is to take the primes closest to the range of the hash function. If the primes are not a factor of the range, then the period of the combined hash function will be the product of the permutation table sizes.

The number of permutation tables will determine the time needed to evaluate the hash function, and the joint size of the permutation tables will determine the memory footprint of the hash function. Period length can be traded for evaluation time and memory footprint.

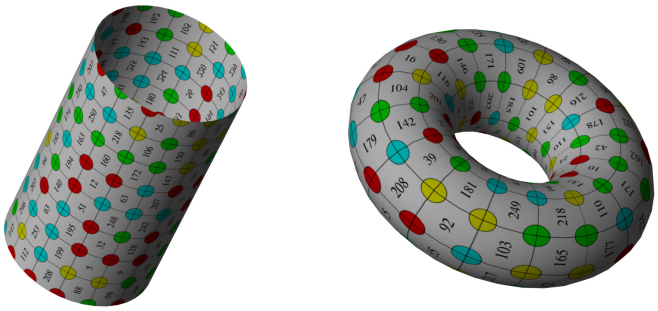
### 3.3.3 HASH FUNCTIONS FOR DIRECT STOCHASTIC TILING ALGORITHMS

Direct stochastic tiling algorithms use a hash function to associate a random color with each integer lattice point. This colored lattice is then transformed to a tiling. Properties of the direct stochastic tiling algorithms such as efficiency and periodicity are inherited from the hash function.

Traditional hash functions based on permutation tables are simple and efficient. They are often used when speed is crucial or when a long period is less important. Permutation table sizes of 256 are common.

When speed is less important or when a long period is crucial, long-period hash functions based on permutation tables can be used. A long-period hash function for direct stochastic tiling algorithms can be designed as follows. The number of colors used in most tilings is 2, 3, 4, 6, or 8. The range of the hash function is therefore set to 24, the least common multiple of these number of colors. By applying an additional modulo divisor, the range of the hash function can be adjusted to 2, 3, 4, 6, and 8. For the sizes of the other permutation tables, the primes 17, 19, 23, 29, 31, and 37 are selected. The period of the combined hash function is $17 \times 19 \times 23 \times 24 \times 29 \times 31 \times 37 = 5{,}930{,}659{,}848$. Note that this is more than the 32-bit integer range. The size of the combined permutation table is only $17 + 19 + 23 + 24 + 29 + 31 + 37 = 180$.

Note that periodicity is not necessarily a bad thing. Periodicity allows to correctly handle boundary conditions when tiling surfaces with cylindrical or toroidal topology. This is illustrated in Figure 3.8.



(a) (b)

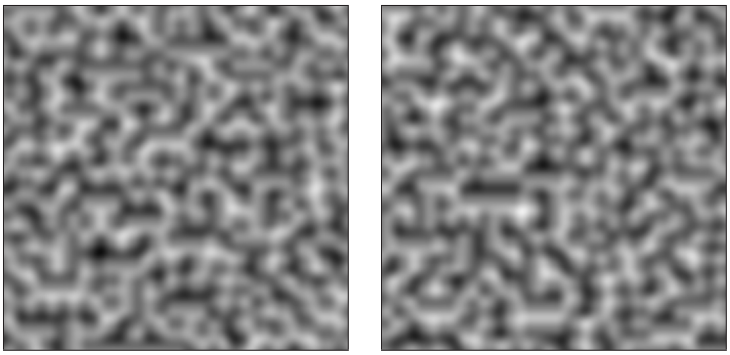
**Figure 3.8:** Tilings of surfaces with (a) cylindrical and (b) toroidal topology.

### 3.3.4 HASH FUNCTIONS FOR PROCEDURAL TEXTURING

The most famous texture basis function is Perlin's noise function (Perlin, 1985, 2002). Long-period hash functions based on permutation tables can be used to robustly implement this texture basis function.

A long-period hash function for Perlin's noise function can be designed as follows. Perlin's noise function uses the lower 4 bits of the hash function (16 values) to choose amongst one of 12 vectors at each integer lattice point. The required range of the hash function is therefore 16. For the sizes of the other permutation tables the primes 11, 13, 17, and 19 are selected. The period of the combined hash function is $11 \times 13 \times 16 \times 17 \times 19 = 739{,}024$. The size of the combined permutation table is $11 + 13 + 16 + 17 + 19 = 76$. Compared to Perlin's implementation, the period

is increased with a factor of almost 3,000, the memory footprint is decreased with a factor of almost 3.5, and the evaluation time is increased with a factor of about 5. The total evaluation time of the modified noise function is increased with a factor of about 2.5. Figure 3.9 shows the original and the modified implementation. As expected, the two images have the same appearance. The long-period hash function does not introduce unwanted artifacts and preserves the typical look of Perlin's noise function.



(a) (b)

**Figure 3.9:** Perlin noise generated (a) with the traditional hash function and (b) with the long-period hash function. The two images have the same appearance.

## 3.4 EXAMPLE CODE

Figure 3.10 shows example code for a simple tiling program. The tiling program generates a $10 \times 10$ tiling with a set of corner tiles over two colors. The tiling program uses the direct stochastic tiling algorithm for corner tiles and a traditional hash function based on a permutation table of size 256.

```cpp
#include <algorithm>
#include <cstdlib>
#include <ctime>
#include <iostream>

int permutation_table[256];

void initialize_permutation_table()
{
  for (int i = 0; i < 256; ++i) {
    permutation_table[i] = i;
  }
  for (int i = 0; i < 256 - 1; ++i) {
    std::swap(permutation_table[i], permutation_table[i + (std::rand() % (256 - i))]);
  }
}

int hash(int x, int y)
{
  return permutation_table[(permutation_table[x % 256] + y) % 256];
}

int tile(int c, int x, int y)
{
  int c_ne = hash(x + 1, y + 1) % c;
  int c_se = hash(x + 1, y    ) % c;
  int c_sw = hash(x    , y    ) % c;
  int c_nw = hash(x    , y + 1) % c;
  return (((((c_ne * c) + c_se) * c) + c_sw) * c) + c_nw;
}

int main(int argc, char* argv[])
{
  std::srand(std::time(0));
  initialize_permutation_table();

  int c = 2;

  for (int row = 0; row < 10; ++row) {
    for (int col = 0; col < 10; ++col) {
      std::cout << tile(c, col, row) << " ";
    }
    std::cout << "\n";
  }
}

```

**Figure 3.10:** Example code for a simple tiling program.

CHAPTER 4

# Tile-Based Methods for Texture Synthesis

In computer graphics, Wang tiles and corner tiles are used to facilitate the synthesis of complex signals. A texture is a complex signal that is difficult to synthesize efficiently. This chapter presents a method for synthesizing a texture over a set of Wang tiles or corner tiles, and a tile-based texture mapping algorithm for efficiently generating an arbitrary large non-repeating texture using a set of precomputed tiles.

This chapter is organized as follows. Section 4.1 introduces texture mapping and texture synthesis. Section 4.2 presents a method for synthesizing a texture over a set of Wang tiles or corner tiles. Section 4.3 presents a tile-based texture mapping algorithm. Section 4.4 discusses the tile packing problem.

This chapter only discusses methods for synthesizing textures over a set of Wang tiles or corner tiles. Efficient tiling algorithms for Wang tiles and corner tiles are presented in Chapter 3.

## 4.1 TEXTURE MAPPING AND TEXTURE SYNTHESIS

Texture mapping was introduced in 1974 by Catmull as a method for increasing the visual complexity of computer-generated images without adding geometric detail. A texture map, or simply a texture, is mapped onto the surface of a shape to add color or detail to the shape.

A texture can be acquired in several ways. Possibilities include painting a texture, taking a digital photograph of a texture, and generating a texture procedurally. Procedural texture synthesis is discussed in detail in Section 6.5. Texture synthesis is an alternative way to obtain textures. Texture synthesis creates from an example texture a new, usually larger texture that appears to be generated by the same underlying process.

Texture synthesis has become a popular area of research within computer graphics, and a complete survey of related work is beyond the scope of this book. For an overview, refer to Liu et al. (2004) and Kwatra et al. (2005). Most techniques for texture synthesis are region growing methods, such as pixel-based texture synthesis (Bonet, 1997; Efros and Leung, 1999; Wei and Levoy, 2000) and patch-based texture synthesis (Efros and Freeman, 2001; Liang et al., 2001; Cohen et al., 2003; Kwatra et al., 2003), or global methods (Heeger and Bergen, 1995; Kwatra et al., 2005). Tile-based texture synthesis can be classified as a patch-based method.

## 4.2 TILE-BASED TEXTURE SYNTHESIS

Stam (1997) was the first to consider Wang tiles in the context of texturing. Stam used Wang tiles to generate non-repeating procedural textures of arbitrary size. A method similar in spirit was presented by Neyret and Cani (1999). They used triangular tiles with edge and corner colors to generate pattern-based textures over a triangle mesh. Cohen et al. (2003) combined Wang tiles with texture synthesis, and presented a method for synthesizing an example texture over a set of Wang tiles. A variation on the technique of Cohen et al. was proposed by Burke. The method of Cohen et al. was also used by Wei (2004), who presented a texture mapping algorithm based on Wang tiles. Fu and Leung (2005) extended texture tiling to arbitrary topological surfaces. The method of Cohen et al. was extended to corner tiles by Ng et al. (2005). Leung et al. (2007) extended tile-based texture synthesis to bidirectional texture functions. Lo et al. (2007) performed reaction-diffusion on surface tiles.

This book uses the method of Ng et al. for synthesizing an example texture over a set of corner tiles. This is illustrated in Figure 4.1. For each corner color, a square patch is chosen at random from the example texture (see Figure 4.1(a)). Each tile of the tile set is constructed by combining the

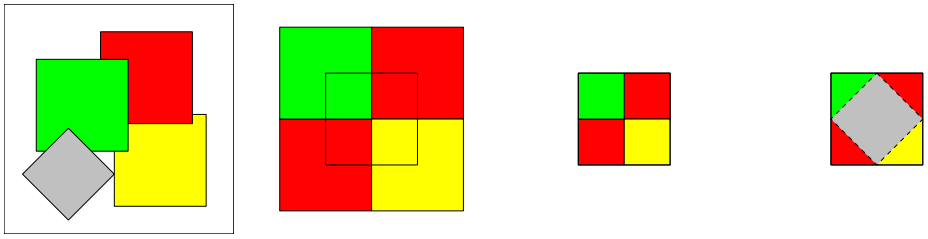
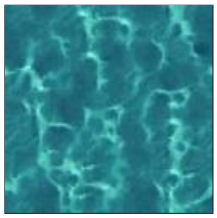
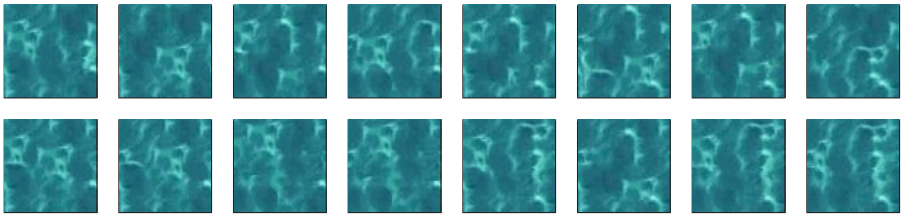



**Figure 4.1:** Construction of a texture tile set based on corner tiles from an example texture. (a) For each color, a square patch is chosen in the example texture (the red, green and yellow patch). (b) The patches are assembled according to the corner colors of the tile. (c) The tile is cut out. (d) The seam is covered with a new irregular patch from the example texture (the gray patch).

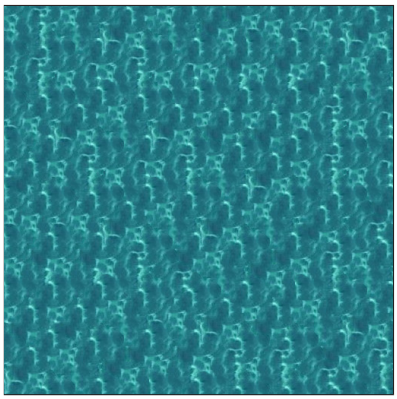
four patches corresponding to the corner colors (see Figure 4.1(b)), and cutting out the tile (see Figure 4.1(c)). This leaves a cross shaped seam that is covered with a new diamond-shaped irregular patch from the example texture (see Figure 4.1(d)). This patch is optimized using the graph cut method (Kwatra et al., 2003), and is restricted to lie in the circle inscribed in the tile. After an example texture is synthesized over a set of corner tiles, a new texture can be synthesized by generating a tiling. The process of tile-based texture synthesis is illustrated in Figure 4.2. The method of Ng et al. is simple and works well. Figure 4.3 shows several textures synthesized with corner-based texture tiles. The quality of the synthesized textures is similar to that of other patch-based techniques.



(a)



(b)

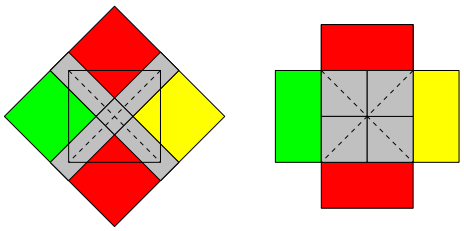


(c)

**Figure 4.2:** Texture synthesis with texture tiles based on corner tiles. (a) The example texture. (b) A set of texture tiles based on corner tiles constructed from the example texture. (c) A new texture synthesized with the texture tile set.

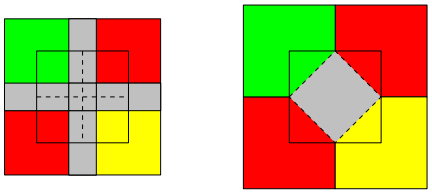
**Figure 4.3:** Textures synthesized with texture tiles based on corner tiles. These textures are synthesized by tiling 4 × 4 tiles from a complete texture tile set based on corner tiles over 2 or 3 colors.

Cohen et al. (2003) were the first to synthesize an example texture over a set of Wang tiles. Several variations on the method of Cohen et al. have been proposed, and the technique of Ng et al. for corner tiles is based on the method of Cohen et al. These methods only differ in how the patches are placed on the tile. Figures 4.4 and 4.5 show several patch combination strategies for Wang tiles and corner tiles. i



(a) (b)

**Figure 4.4:** Patch combination strategies for texture tiles based on Wang tiles. (a) The method of Cohen et al. (b) A variant introduced by Burke.



(a) (b)

**Figure 4.5:** Patch combination strategies for texture tiles based on corner tiles. (a) A straightforward extension of the method of Cohen et al. for Wang tiles to corner tiles. (b) The method of Ng et al. Note that this is the only method that adds a new texture patch to each tile.

The advantage of corner tiles over Wang tiles is less pronounced in texture tile construction. Unwanted artifacts in the synthesized textures are typically located where patches meet. This is at the corners for Wang tiles and in the middle of the edges for corner tiles. In this respect, Wang tiles and corner tiles are similar. However, textures synthesized with corner tiles are usually more similar to the example texture than textures synthesized with Wang tiles. This is because the center of each corner tile is covered with a new irregular patch from the example texture. Therefore,

each corner tile contains potentially unique texture samples from the example texture. Other patch combination strategies use for each tile only the patches corresponding to the corner or edge colors. Refer to Ng et al. (2005) for a more detailed comparison.

An important advantage of tile-based texture synthesis is that the process of texture synthesis is broken up into two parts. Once an example texture is synthesized over a set of tiles, arbitrary large textures can be synthesized very efficiently simply by generating stochastic tilings.

## 4.3 TILE-BASED TEXTURE MAPPING

Interactive applications in computer graphics harness the power of graphics hardware to guarantee interactive frame rates. Texture mapping is a fundamental feature for these applications. However, texture memory is a scarce resource on graphics hardware, and storing and managing large textures is challenging. All too often, tileable textures are used to save texture memory. However, this results in visually disturbing repetition. In 2004, Wei presented a texture mapping algorithm based on Wang tiles to overcome this problem. In 2006, Lagae and Dutré (2006b) presented a cleaner and more efficient variant version of this algorithm based on corner tiles. This section discusses these tile-based texture mapping algorithms in detail.

Tile-based texture mapping uses an example texture synthesized over a set of Wang tiles or corner tiles to simulate an arbitrary large non-periodic texture. This is more complicated than it seems at first sight, because graphics hardware is very specialized and the graphics processing unit (GPU) is a stream processing architecture.

The number of texture units on a GPU is typically small, and a GPU generally prefers square textures. Therefore, all tiles of the tile set must be packed into a single square texture. Tile-based texture mapping algorithms typically use complete tile sets. This is because the $C^4$ tiles of a complete tile set over $C$ colors can easily be arranged into a square texture using a $C^2 \times C^2$ configuration. However, in order to avoid unwanted discontinuity artifacts introduced by texture filtering, this configuration must also be a valid tiling. This is because texture sampling uses texels from adjacent tiles. Note that the borders of a texture are treated toroidally. For more details, refer to Wei (2004).

An arrangement of the $C^4$ tiles of a complete set of Wang tiles or corner tiles over $C$ colors into a $C^2 \times C^2$ toroidal configuration such that adjoining edges or corners have matching colors is called a tile packing. Tile packings are discussed in detail in Section 4.4.

The tile-based texture mapping algorithm runs as a fragment program on the GPU. This fragment program transforms texture coordinates in the arbitrary large non-periodic texture to texture coordinates in the texture containing the texture tiles. A tiling is imposed on the arbitrary large non-periodic texture. For each incoming fragment, the tile coordinates and the coordinates within the tile are computed. A direct stochastic tiling algorithm is used to determine the tile at these coordinates. The tile packing provides the location of that tile in the texture containing the texture tiles. This location is combined with the coordinates of the fragment within the tile, and the texture lookup is performed. For more implementation details refer to Wei (2004) and Lefebvre and Neyret (2003).

A Wang tile packing can be formulated as a closed-form expression (Wei, 2004). This expression is evaluated directly in the fragment program. However, this is not the case for a corner tile packing (see Section 4.4). Therefore, the corner tile packing is stored explicitely, as a constant array in the fragment program, or as an additional texture. The permutation table used by the hash function of the direct stochastic tiling algorithm is stored in the same way.

To avoid the corner problem, the tile-based texture mapping algorithm of Wei requires a second Wang tile packing that contains all possible corner configurations of the Wang tile set. This additional texture is used for texture lookups close to tile corners. Because corner tiles are not subject to the corner problem, only the texture containing the tile packing is needed. Compared to the original method of Wei, the tile-based texture mapping algorithm based on corner tiles reduces the required texture memory by a factor of two and saves one texture unit. This is an important advantage, as reducing texture memory usage is the main goal of tile-based texture mapping. The algorithm also runs faster, because the tiling algorithm for corner tiles is simpler and more efficient than equivalent algorithms for Wang tiles. Corner tiles reduce the cost of tile-based texture mapping almost to that of regular texture mapping. This is a significant saving for interactive applications.

The tile-based texture mapping algorithm based on corner tiles runs at several hundred frames per second on a NVidia GeForce 7800 GTX graphics card. Figure 4.6 shows several results.

## 4.4 THE TILE PACKING PROBLEM

A tile packing is an essential ingredient of the tile-based texture mapping algorithms discussed in Section 4.3. A tile packing is used to avoid unwanted texture filtering artifacts. This section discusses the problem of computing a tile packing of a complete set of Wang tiles or corner tiles, and shows that the tile packing problem is an interesting combinatorial problem.

The tile packing problem consists of arranging the $C^4$ tiles of a complete set of Wang tiles or corner tiles over $C$ colors into a $C^2 \times C^2$ toroidal configuration such that adjoining edges or corners have matching colors.

In 2006, Lagae and Dutré studied the corner tile packing problem in detail (Lagae and Dutré, 2006a,d, 2007).

### 4.4.1 THE ONE-DIMENSIONAL TILE PACKING PROBLEM

In one dimension, Wang tiles and corner tiles can be seen as dominoes. A tiling with one-dimensional Wang tiles can also be seen as a single row from a tiling with two-dimensional Wang tiles, where the colors of horizontal edges are ignored. A complete set of one-dimensional Wang tiles or corner tiles over $C$ colors counts $C^2$ tiles. Figure 4.7 shows the complete set of tiles over 2 and 3 colors. The one-dimensional tile packing problem consists of arranging a complete set of $C^2$ tiles into a single circular train. Domino problems like this one are well known in the field of recreational mathematics. A solution based on graph theory is given in the classic work *Mathematical Recreations and Essays* (Ball, 1926).

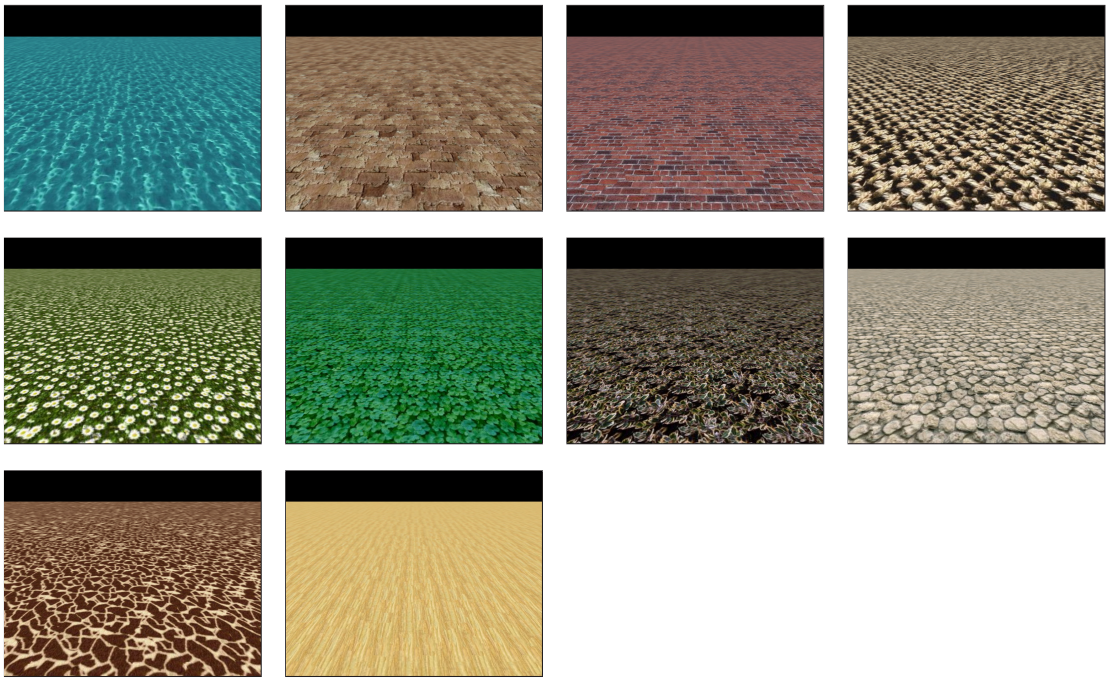


**Figure 4.6:** Tile-based texture mapping using corner tiles. Screenshots from our interactive tile-based texture mapping application based on corner tiles. Texture filtering does not introduce unwanted artifacts because a tile packing was used.

The tile set is represented as a directed graph with a vertex for each color, and an edge connecting two vertices for each tile in the tile set. Figure 4.8 shows the graphs for the complete tile sets over 2 and 3 colors. A solution for the tile packing problem is given by an Eulerian circuit, a graph cycle that uses each graph edge exactly once. A complete tile set results in a complete directed graph, which always has an Eulerian circuit. Figure 4.9 shows tile packings of the complete tile sets over 2 and 3 colors. A tile packing obtained with this method is called an Eulerian Wang tile packing.

Wei (2004) presented a closed-form expression that gives the position of a specific tile in a one-dimensional Eulerian tile packing.

### 4.4.2 THE WANG TILE PACKING PROBLEM

Wei (2004) observed that Wang tiles are separable and that a solution for the two-dimensional Wang tile packing problem is given by the outer product of two one-dimensional tile packings. This is illustrated in Figure 4.10.

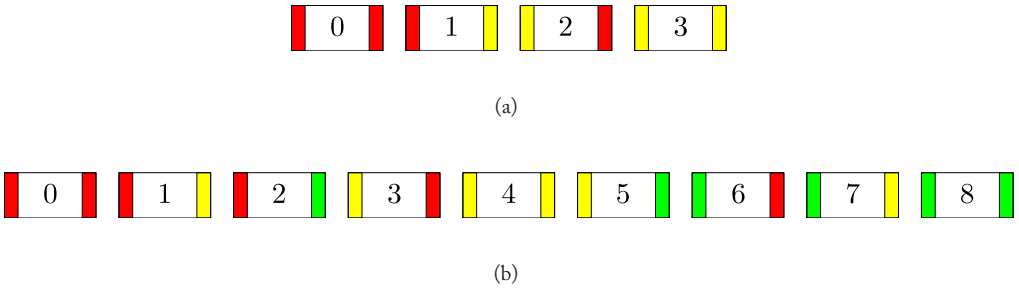



**Figure 4.7:** The complete 1D Wang or corner tile sets over (a) 2 and (b) 3 colors.

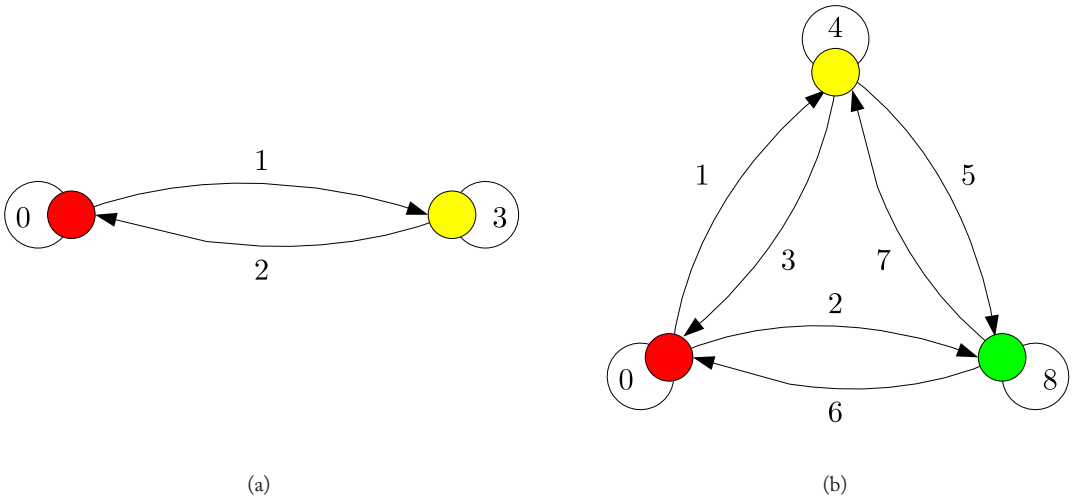



**Figure 4.8:** Graphs representing the complete 1D Wang or corner tile set over (a) 2 and (b) 3 colors.

A one-dimensional tile packing of a complete set over $C$ colors consist of $C^2$ tiles. The outer product of two such tile packings produces a matrix of $C^4$ tiles. Because adjoining edges have matching colors, and each tile occurs exactly once, this is a tile packing of the $C^4$ tiles of a complete Wang tile set over $C$ colors. This construction method generalizes to tile packings of Wang tiles in any dimension.

A closed-form expression that gives the position of a specific tile in an Eulerian Wang tile packing is obtained by applying the closed-form expression for the one-dimensional case for each dimension.

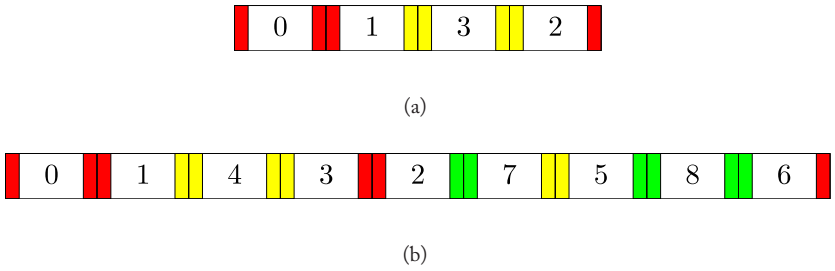



**Figure 4.9:** Tile packings of the complete 1D Wang or corner tile set over 2 and 3 colors.

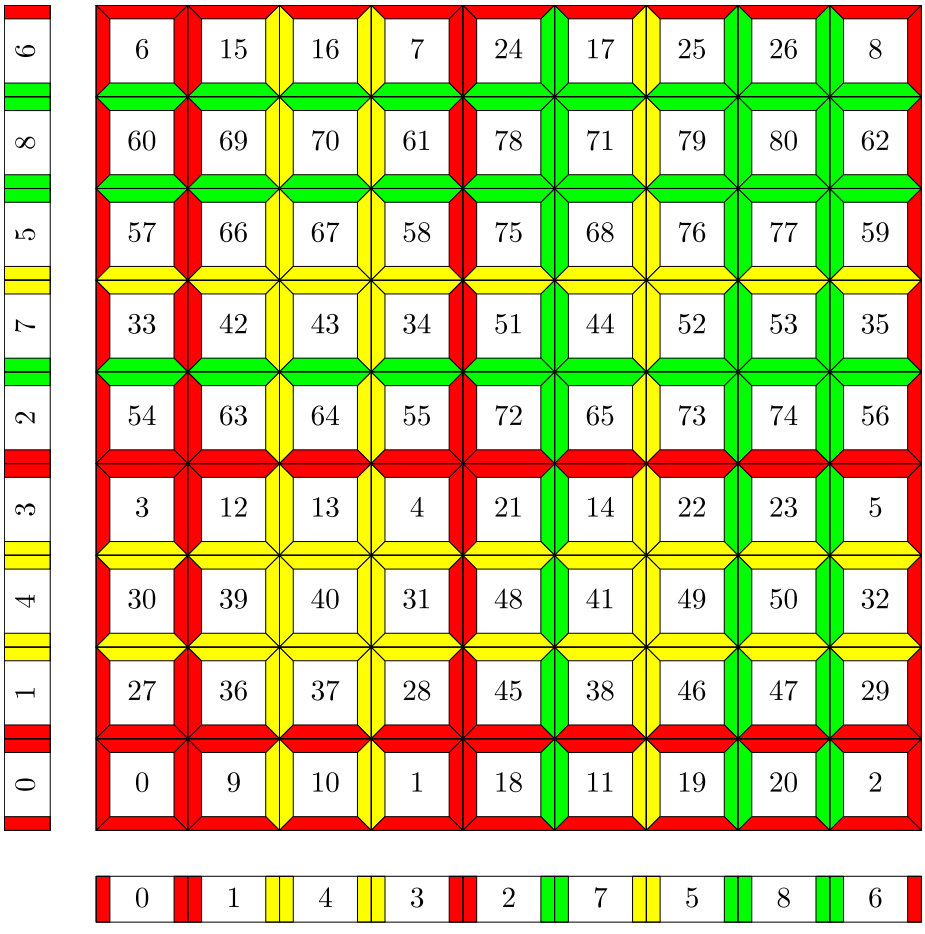



**Figure 4.10:** The construction of an Eulerian tile packing of the complete Wang tile set over 3 colors.

### 4.4.3 THE CORNER TILE PACKING PROBLEM

Although tiles with colored edges and problems similar to the Wang tile packing problem were studied before in the field of recreational mathematics (MacMahon, 1921), corner tiles and the corner tile packing problem have not been examined. The method for constructing a Wang tile packing also did not seem to extend to corner tiles. Because it was not clear whether the corner tile packing problem even had a solution, Lagae and Dutré (2006d) tackled the problem using combinatorial search methods.

A simple exhaustive search or generate-and-test method is not practical. For $C$ colors the tiles can be arranged in $C^4!$ ways. For 2, 3, and 4 colors, this equals approximately $2.09 \times 10^{13}$, $5.80 \times 10^{120}$, and $8.58 \times 10^{506}$. Instead, a backtracking method is used, that places one tile at a time until a dead end is reached, at which point previous steps are retraced. Backtracking greatly reduces the amount of work in an exhaustive search, and is often used to solve hard combinatorial problems such as the knights tour problem and the queens problem (Ball, 1926). The algorithm can also be used to search for solutions to the Wang tile packing problem.

Although backtracking is relatively fast compared to simple exhaustive search and generate-and-test methods, the time needed to solve the tile packing problem is still large. Therefore, our backtracking algorithm also supports parallelization, checkpointing and progress estimation. For implementation details, refer to (Lagae and Dutré, 2006d).

With the backtracking algorithm, Wang and corner tile packings for 2, 3, and 4 colors can be computed. For 2 colors, all solutions of the Wang and corner tile packing problem are obtained almost immediately on a regular desktop computer. The corner tile packing problem has 32 solutions and the Wang tile packing problem has 203,520 solutions. This supports the claim that in some way, corner tile packings are more difficult to construct than Wang tile packings. For 3 colors, the first solution of the Wang and corner tile packing problems is obtained almost immediately, but computing or counting all solutions seems to be hopeless. For 4 colors, computing a corner tile packing took 280 days of CPU time, and it took roughly 23 years of CPU time to find the first solution of the Wang tile packing problem. These last results were obtained using a parallel version of the backtracking algorithm, running on a cluster with almost 400 2.4 GHz CPU's.

A solution for $C$ colors can often be found faster by starting from a solution of $C - 1$ colors. That way, a recursive tile packing is obtained. Figures 4.12 and 4.11 show recursive Wang and corner tile packings for 4 colors.

The tile packing problem has many symmetries. New solutions can be obtained from existing ones using translation (the tile packing is toroidal), rotation, reflection, and permutation of the colors. The 32 solutions of the 2 color corner tile packing problem reduce to a single fundamental solution. There is still room for improving the backtracking algorithm, since the many symmetries of the tile packing problem are currently not exploited.

The tile packing problem is an interesting combinatorial puzzle. Although Wang and corner tile packings for up to 4 colors were obtained, several problems, such as counting the number of

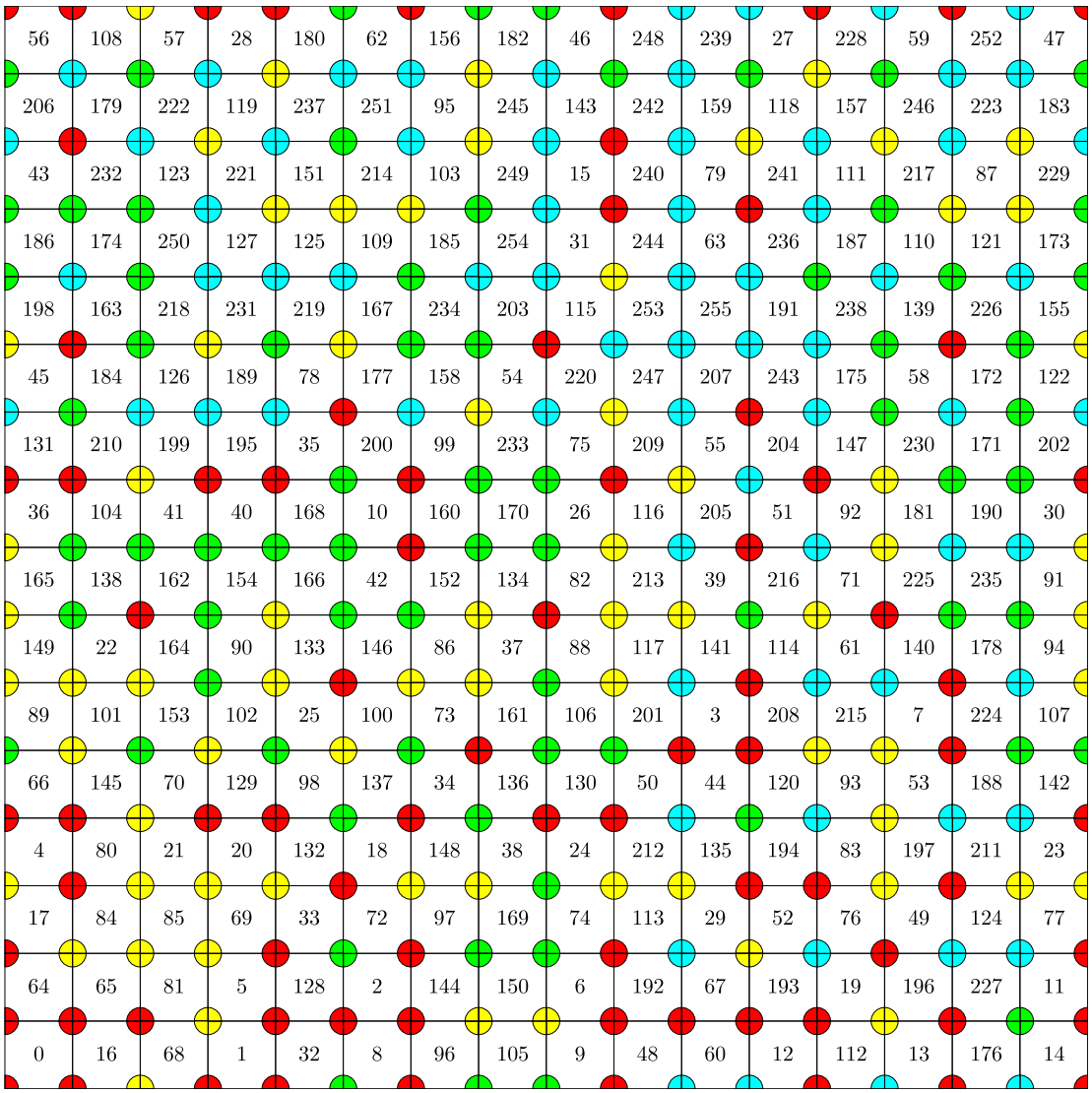



**Figure 4.11:** A recursive tile packing of the complete corner tile set over 4 colors.

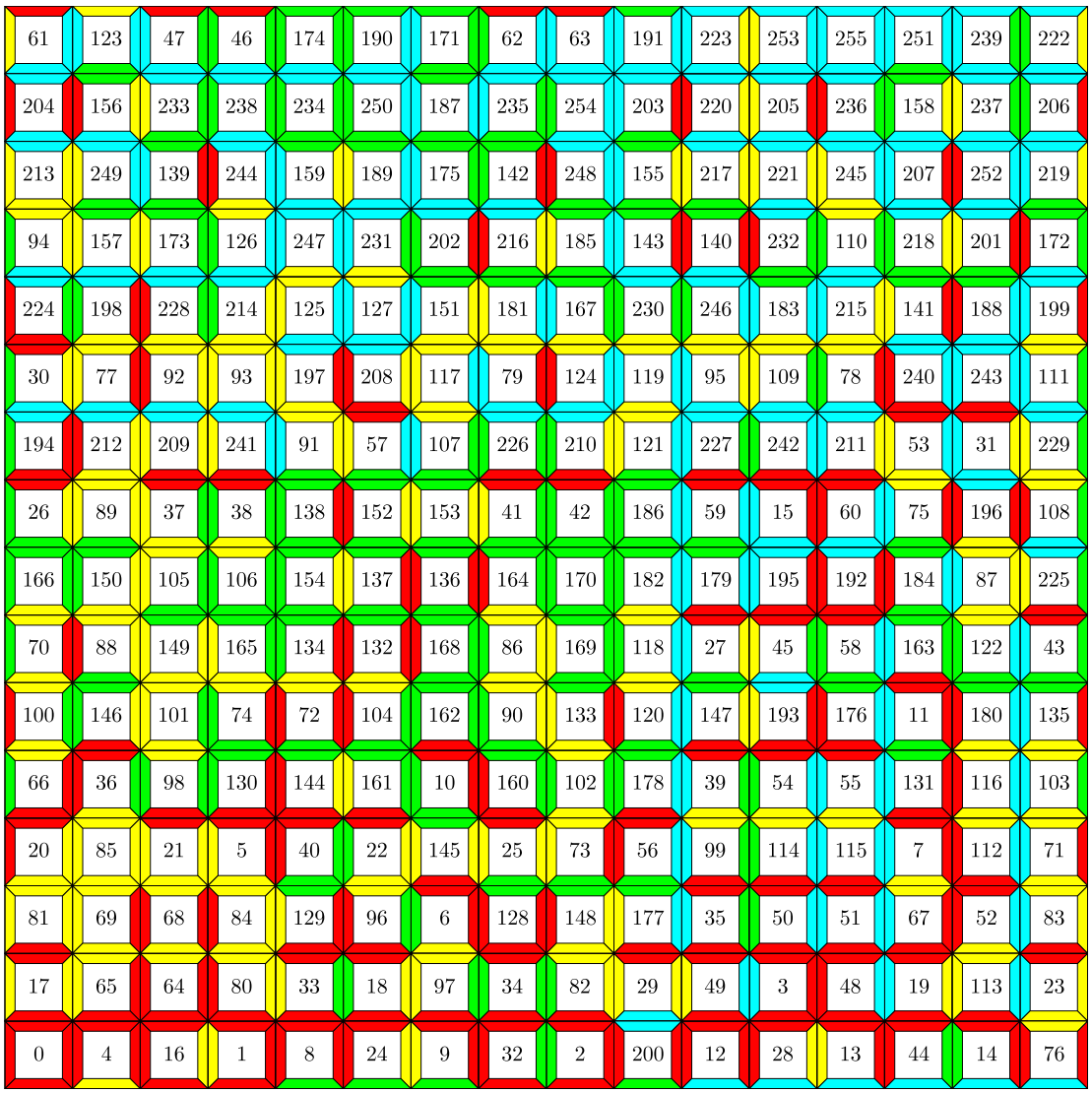


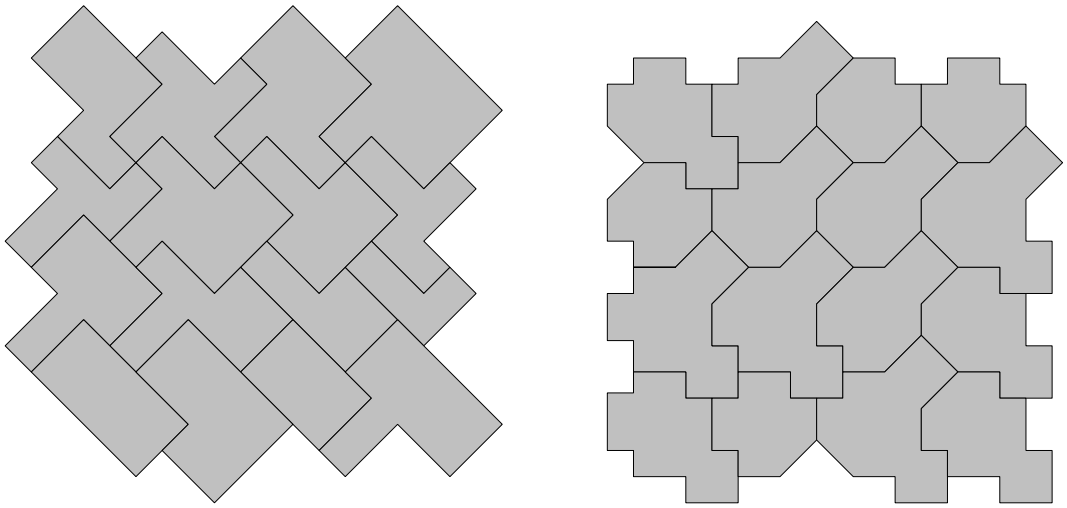

**Figure 4.12:** A recursive tile packing of the complete Wang tile set over 4 colors.

solutions of the tile packing problem, and finding a constructive method and a closed-form expression for the corner tile packing problem, remain unsolved.

### 4.4.4 PUZZLES DERIVED FROM THE TILE PACKING PROBLEM

The work most closely related to the tile packing problem in the field of recreational mathematics is that of MacMahon (1921). He describes sets of pieces of different geometrical forms (including equilateral triangles, squares, and pentagons) with colored edges that are tiled into another geometrical form. The profile of the adjoining edges is then altered to produce jigsaw puzzles. His work is unique in the fact that it details how the puzzles can be constructed and solved. In contrast with Wang and corner tiles, the pieces of MacMahon pieces may be rotated. MacMahon also does not consider pieces with colored corners.

Inspired by the work of MacMahon (1921), Lagae and Dutré (2007) created jigsaw puzzles from tile packings by altering the profile of the adjoining edges or corners. Figure 4.13 shows two examples. To create interesting puzzles, it is better to use tile packings constructed with the



(a) (b)

**Figure 4.13:** Jigsaw puzzles derived from tile packings of the complete (a) Wang and (b) corner tile set over 2 colors.

backtracking algorithm instead of Eulerian tile packings. It is already hard to construct a tile packing of the complete set of corner tiles over 2 colors by hand, so these puzzles should be challenging, especially puzzles based on tile packings of 3 or 4 colors. To prevent the tiles from being rotated, a picture could be printed on the puzzle.

CHAPTER 5

# Tile-Based Methods for Generating Poisson Disk Distributions

In computer graphics, Wang tiles and corner tiles are used to facilitate the synthesis of complex signals. Poisson disk distributions are complex point distributions that are difficult to generate in real time. This chapter presents a method for constructing a Poisson disk distribution over a set of corner tiles. With a single set of precomputed tiles, high-quality Poisson disk distributions of arbitrary size can be generated very efficient, simply by producing a stochastic tiling.

This chapter is organized as follows. Section 5.1 introduces Poisson disk distributions. Section 5.2 presents corner-based Poisson disk tiles, a tile-based methods for generating Poisson disk distributions. Section 5.3 briefly surveys other tile-based methods for generating Poisson disk distributions.

This chapter only discusses methods for constructing Poisson disk distributions over a set of Wang tiles or corner tiles. Applications of Poisson disk distributions are discussed in Chapter 6, and efficient tiling algorithms for Wang tiles and corner tiles are presented in Chapter 3.

## 5.1 POISSON DISK DISTRIBUTIONS

In this section Poisson disk distributions are defined, the history and background of Poisson disk distributions is sketched, an intuitive radius specification scheme for Poisson disk distributions is presented, and traditional methods for generating Poisson disk distributions are introduced.

### 5.1.1 DEFINITION

A Poisson distribution or random distribution is the simplest random point distribution. A Poisson distribution is a point distribution in which the points and the coordinates of the points have no relationship to each other. A Poisson distribution is obtained by generating uniformly distributed random numbers and using them as coordinates for the points. This distribution is called a Poisson distribution because the number of points in an area is distributed according to a Poisson probability distribution with mean equal to the area multiplied with the density.

A Poisson disk distribution is a two-dimensional Poisson distribution in which all points are separated from each other by a minimum distance. Half that distance is called the radius $r$ of the distribution. If a disk of that radius is placed at each point, then no two disks overlap. This explains

why this distribution is called a Poisson disk distribution. Figure 5.1 shows an example of a Poisson disk distribution.

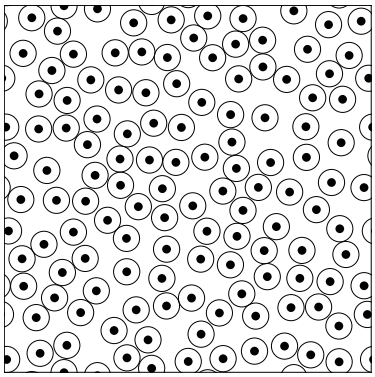


**Figure 5.1:** A Poisson disk distribution.

### 5.1.2 HISTORY AND BACKGROUND

Poisson disk distributions were introduced in the field of computer graphics to solve the aliasing problem. Aliasing is a major source of artifacts in digitally synthesized images. This problem was first identified by Crow (1977). Dippé and Wold (1985), Cook (1986) and Mitchell (1987) introduced nonuniform sampling to turn regular aliasing patterns into featureless noise, which is perceptually less objectable. The Poisson disk distribution was identified as one of the best sampling patterns. This work was based on studies by Yellot (1982, 1983), who found that the photoreceptors in the retina of the eye are distributed according to a Poisson disk distribution, an indication that this sampling pattern is effective for imaging.

Poisson disk distributions are traditionally generated using an expensive dart throwing algorithm (Cook, 1986). Fast methods that generate approximate Poisson disk distributions have been suggested by various authors (Dippé and Wold, 1985; Mitchell, 1987, 1991; Klassen, 2000). The algorithm mostly used nowadays is due to McCool and Fiume (1992). It generalizes over the dart throwing approach, and uses Lloyd's relaxation method (Lloyd, 1982) to optimize the generated distribution.

Because Poisson disk distributions are expensive to generate, Dippé and Wold (1985) suggested already in 1985 to replicate a precomputed tile with Poisson disk distributed points across the plane. Since then, several tile-based methods were proposed. Most of them use Wang tiles. A survey is given in Section 5.3. Tools to analyze the spectral properties of point sets were introduced by Ulichney (1987), in the context of dithering.

### 5.1.3 RADIUS SPECIFICATION

The radius of a Poisson disk distribution determines how well the points are distributed, and is therefore a measure of the quality of the Poisson disk distribution. The radius is typically expressed as an absolute number. However, this is not practical because the radius is dependent on the size of the domain of the point distribution and on the number of points in the distribution.

In 2005, Lagae and Dutré (2005a) proposed a more intuitive radius specification scheme. Instead of using the absolute radius $r$, the radius is expressed as a relative radius $\rho$. The relative radius $\rho$ is a fraction of the maximum radius $r_{max}$ that can be achieved.

The densest packing of disks in the plane is a hexagonal lattice. Therefore, the point configuration with maximum disk radius $r_{max}$ is a hexagonal lattice. The packing density $\eta$ of a hexagonal lattice is (Steinhaus, 1999)

$$\eta = \frac{\pi}{2\sqrt{3}} \approx 0.9069. \tag{5.1}$$

The packing density is defined as the fraction of the area filled by the disks.

The maximum disk area of a Poisson disk distribution counting $N$ points over the toroidal unit square is therefore $\eta/N$. The maximum possible disk radius $r_{max}$ of this Poisson disk distribution is thus given by

$$r_{max} = \sqrt{\frac{1}{2\sqrt{3}N}}. \tag{5.2}$$

The Poisson disk radius $r$ of a given point distribution is specified as a fraction $\rho$ of the maximum disk radius

$$r = \rho\, r_{max}, \tag{5.3}$$

with $\rho \in [0, 1]$.

In contrast with the absolute radius, the relative radius is independent of the number of points and the size of the domain of the point distribution. The relative radius is therefore a good measure of how well the points are distributed.

The relative radius of a Poisson distribution is 0, because a Poisson distribution does not enforce a minimum distance between points. The relative radius of a hexagonal lattice is 1, because a hexagonal lattice is the densest packing of disks in the plane. The relative radius of a Poisson disk distribution should be relatively large, in order to ensure a good distribution of points, but not too large, because point distributions with a very large relative radius are too close to the hexagonal lattice, and are therefore too regular. Practice shows that the radius of most Poisson disk distribution is somewhere in between 0.65 and 0.85.

### 5.1.4 GENERATION

Poisson disk distributions are traditionally generated with dart throwing, relaxation dart throwing, or Lloyd's relaxation method.

#### 5.1.4.1 Dart Throwing

The dart throwing algorithm of Cook (1986) was the first algorithm to generate Poisson disk distributions. The algorithm generates points distributed according to a Poisson distribution, and rejects points that do not satisfy the minimum separation with already generated points. This process continues until no more points can be added. To correctly handle boundary conditions, the distributions generated by the dart throwing algorithm are usually toroidal.

This algorithm is expensive, and difficult to control. Instead of specifying the number of points, the radius of the distribution has to be provided, the final number of points in the distribution is difficult to predict, and if the process is stopped too soon, the density of the points is not uniform.

#### 5.1.4.2 Relaxation Dart Throwing

McCool and Fiume (1992) proposed an improved version of the dart throwing algorithm, which is called relaxation dart throwing. Points are placed with a large radius initially, and once no more space has been found for a large number of attempts, the radius is reduced by some fraction.

This algorithm has several advantages compared to dart throwing. It is faster, it allows to specify the final size of the distributions rather than the radius, and termination is guaranteed.

#### 5.1.4.3 Lloyd's Relaxation Scheme

After a Poisson disk distribution is generated, McCool and Fiume (1992) apply Lloyd's relaxation method (Lloyd, 1982) to optimize the radius of the Poisson disk distribution. Lloyd's relaxation method is an iterative process. In each iteration, the Voronoi diagram of the point set is computed, and each point is moved to the centroid of its Voronoi cell. This process is illustrated in Figure 5.2.

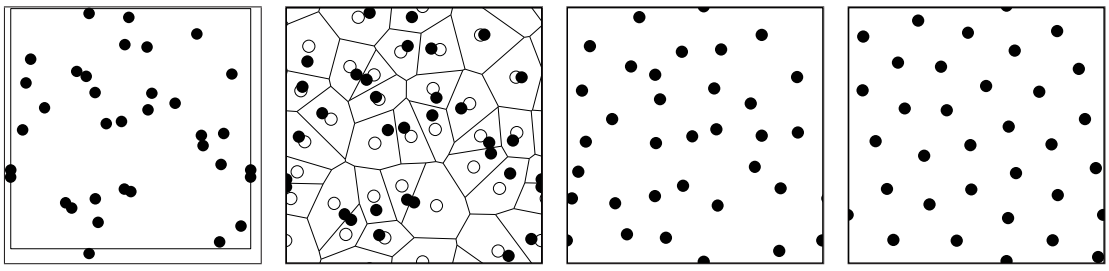



**Figure 5.2:** Lloyd's relaxation method. (a) The initial point set. (b) The Voronoi diagram of the initial point set. The centroids of the Voronoi cells are indicated by circles. (c) The points are moved to the centroid of their Voronoi cell. (d) This process is iterated. Note the increase in radius of the point set.

## 5.2 CORNER-BASED POISSON DISK TILES

In 2006, Lagae and Dutré (2006a) presented corner tiles and corner-based Poisson disk tiles, a method for constructing a Poisson disk distribution over a set of corner tiles.

Constructing a Poisson disk distribution over a set of corner tiles is challenging. The difficulty is to generate a Poisson disk distribution in each tile of the tile set, such that every tiling results in a valid Poisson disk distribution. The minimum distance criterion of a Poisson disk distribution imposes severe constraints on the point distributions in the corner tiles.

A point in a tile closer to a corner than the Poisson disk radius affects points in three neighboring tiles. A point in a tile closer to an edge than the Poisson disk radius affects points in one neighboring tile. A point in a tile, further away from the tile boundary than the Poisson disk radius does not affect points in neighboring tiles. The regions obtained this way are called the Poisson disk tile regions. The Poisson disk radius determines corner regions, edge regions and an interior region. This is illustrated in Figure 5.3(a).

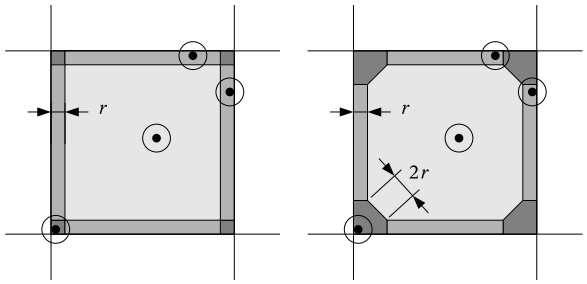



**Figure 5.3:** The Poisson disk tile regions and the modified Poisson disk tile regions. (a) The Poisson disk radius determines corner regions, edge regions and an interior region. (b) The corner regions are modified such that the distance between regions of the same type is at least $2r$.

To minimize the constraints between the different regions, the corner regions are enlarged such that the distance between edge regions is twice the Poisson disk radius. The regions obtained this way are called the modified Poisson disk tile regions. Within a single tile, points in edge regions now only affect points in corner regions, and not in other edge regions. This is illustrated in Figure 5.3(b).

By combining the modified Poisson disk tile regions with the complete corner tile set over $C$ colors, a new tiling is obtained. This is illustrated in Figure 5.4. This tiling uses three different kinds of tiles: corner tiles, horizontal and vertical edge tiles, and interior tiles. Corner tiles correspond to the union of four modified corner regions. There are $C$ corner tiles, one for each color. Edge tiles correspond to the union of two modified edge regions. There are $C^2$ horizontal and $C^2$ vertical

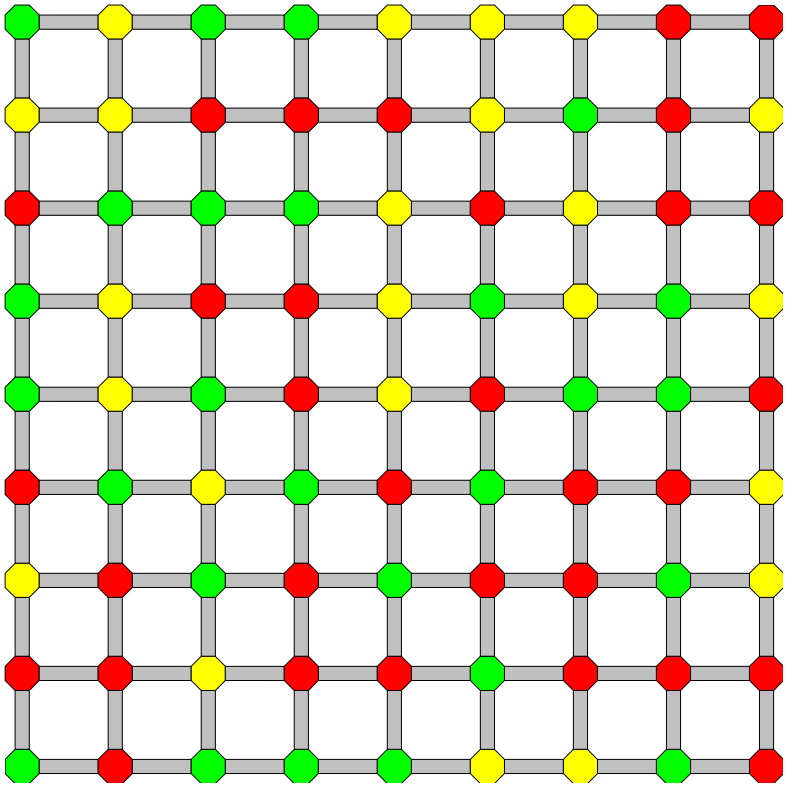


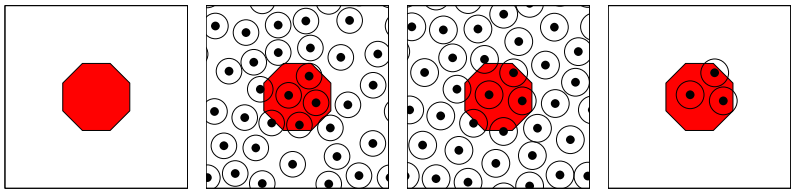
**Figure 5.4:** A tiling obtained by combining the modified Poisson disk tile regions with the complete corner tile set over 3 colors. This tiling was generated from the tiling shown in Figure 2.6.

edge tiles, one for each combination of two corner colors. Interior tiles correspond to the modified interior regions. There are $C^4$ interior tiles, one for each combination of four corner colors.

To construct a Poisson disk distribution over a set of corner tiles, the number of colors of the corner tile set $C$, the number of points per tile $N$, and the relative Poisson disk radius $\rho$ are chosen. The absolute Poisson disk radius determines the size of the modified Poisson disk regions.

First, a Poisson disk distribution is constructed over the corner tiles. This is illustrated in Figure 5.5. For each corner tile, a toroidal Poisson disk distribution of $N$ points is generated using dart throwing or relaxation dart throwing (see Figure 5.5(b)), optionally followed by Lloyd's relaxation method (see Figure 5.5(c)). The corner tile is then cut out of the Poisson disk distribution (see Figure 5.5(d)). If the desired Poisson disk radius is not reached, this process is repeated. Figure 5.6 shows Poisson disk distributions constructed over corner tiles.

Next, a Poisson disk distribution is constructed over the edge tiles. This is illustrated in Figure 5.7. Each edge tile is assembled with the corresponding corner tiles (see Figure 5.7(a)). A



(a) (b) (c) (d)

**Figure 5.5:** Construction of a Poisson disk distribution over a corner tile of a corner-based Poisson disk tile set. (a) The corner tile. (b) A toroidal Poisson disk distribution is generated. (c) The Poisson disk distribution is optimized using Lloyd's relaxation scheme. (d) The corner tile is cut out of the Poisson disk distribution.

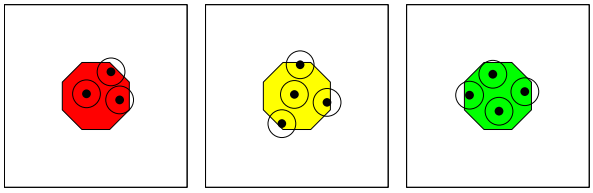
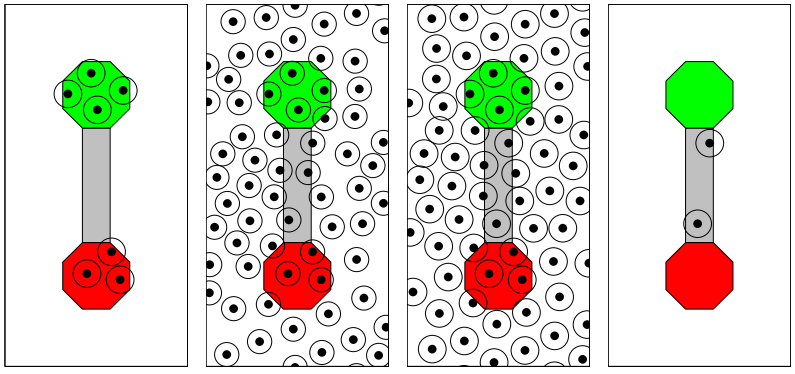


**Figure 5.6:** Poisson disk distributions constructed over corner tiles of a corner-based Poisson disk tile set.



(a) (b) (c) (d)

**Figure 5.7:** Construction of a Poisson disk distribution over a vertical edge tile of a corner-based Poisson disk tile set. (a) The edge tile is assembled with the corresponding corner tiles. (b) A toroidal Poisson disk distribution is generated. (c) The Poisson disk distribution is optimized using Lloyd's relaxation scheme. (d) The edge tile is cut out of the Poisson disk distribution.

toroidal Poisson disk distribution is generated using dart throwing or relaxation dart throwing (see Figure 5.7(b)), optionally followed by Lloyd's relaxation method (see Figure 5.7(c)). The edge tile is then cut out of the Poisson disk distribution (see Figure 5.7(d)). If the desired Poisson disk radius is not reached, this process is repeated. No new points are added to the corner tiles. During relaxation, the points in the corner tiles are fixed, and other points are prohibited to enter the corner tiles. This is done by clipping the displacement vectors of points that are about to enter the corner tiles. Figures 5.8 and 5.9 show Poisson disk distributions constructed over horizontal and vertical edge tiles.

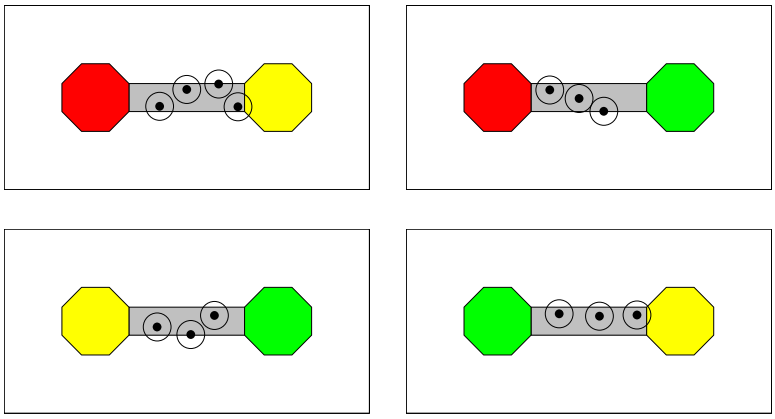


**Figure 5.8:** Poisson disk distributions constructed over horizontal edge tiles of a corner-based Poisson disk tile set.

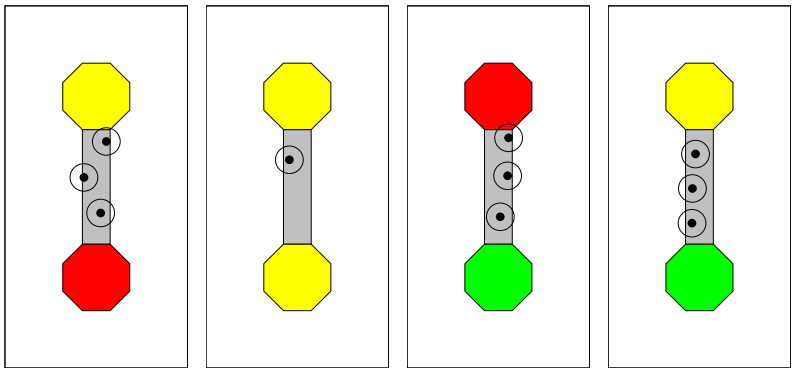


**Figure 5.9:** Poisson disk distributions constructed over vertical edge tiles of a corner-based Poisson disk tile set.

Finally, a Poisson disk distribution is constructed over the interior tiles. This is illustrated in Figure 5.10. Each interior tile is assembled with the corresponding corner tiles and edge tiles (see

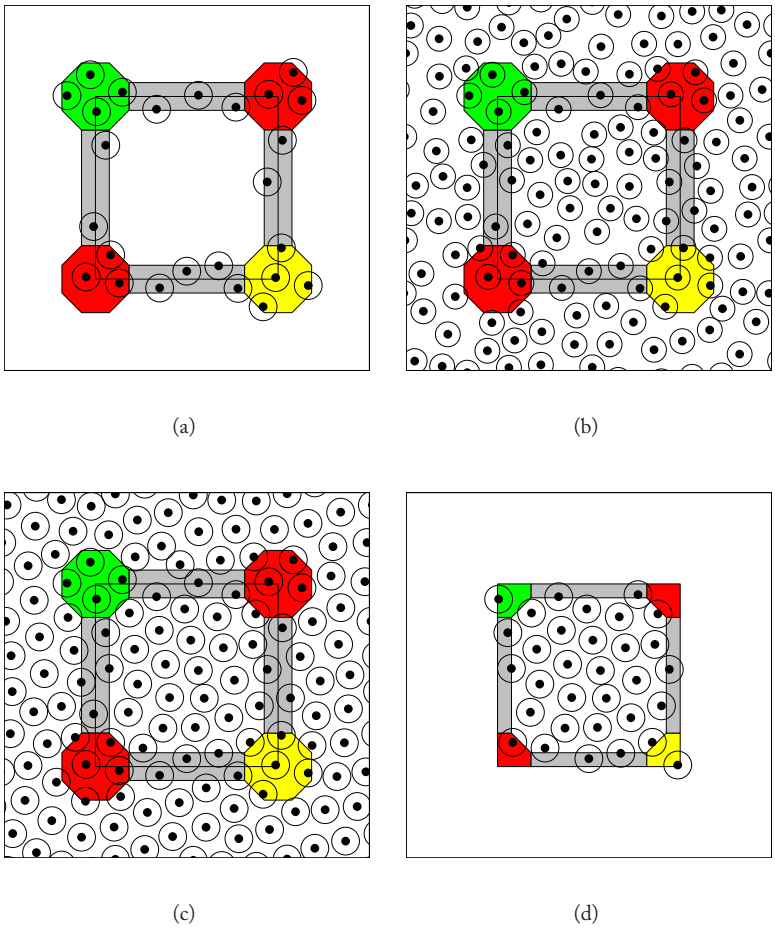



**Figure 5.10:** Construction of a Poisson disk distribution over a tile of a corner-based Poisson disk tile set. (a) The interior tile is assembled with the corresponding corner tiles and edge tiles. (b) A toroidal Poisson disk distribution is generated. (c) The Poisson disk distribution is optimized using Lloyd's relaxation scheme. (d) The tile is cut out of the Poisson disk distribution.

Figure 5.10(a)). A toroidal Poisson disk distribution that brings the number of points inside the tile to $N$ is generated using dart throwing or relaxation dart throwing (see Figure 5.10(b)), optionally followed by Lloyd's relaxation method (see Figure 5.10(c)). The tile is then cut out of the Poisson disk distribution (see Figure 5.10(d)). If the desired Poisson disk radius is not reached, this process

is repeated. No new points are added to the corner tiles and the edge tiles. During relaxation, the points in the corner tiles and the edge are fixed, and other points are prohibited to enter the corner tiles and the edge tiles. Figure 5.11 shows Poisson disk distributions constructed over corner tiles.

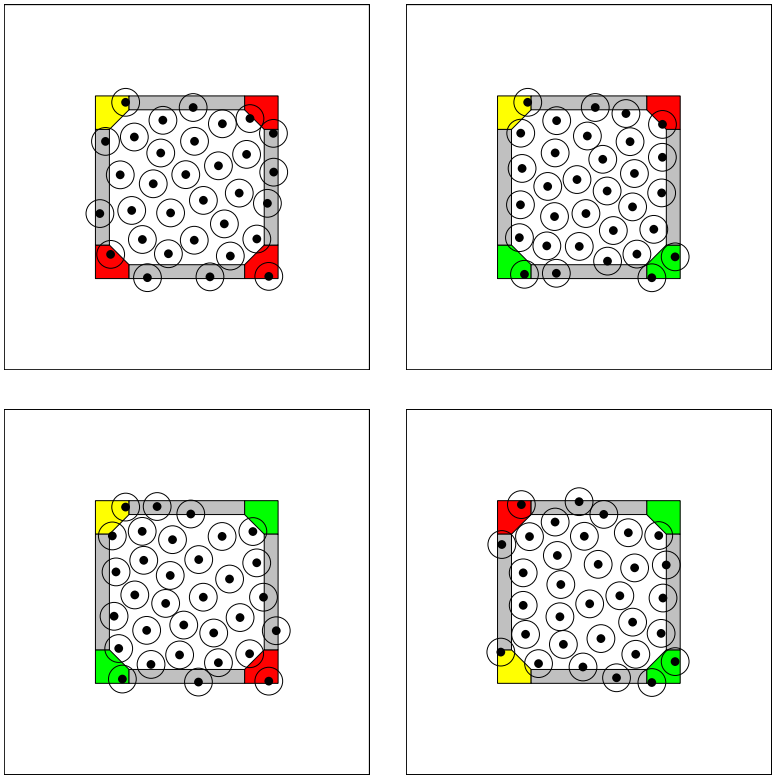


**Figure 5.11:** Poisson disk distributions constructed over tiles of a corner-based Poisson disk tile set.

A corner-based Poisson disk tile set based on a complete Wang tile set over $C$ colors consists of $C^4$ tiles. For 2, 3, 4, 5, 6, 7, and 8 colors, a set of corner-based Poisson disk tiles counts 16, 81, 256, 625, 1,296, 2,401, and 4,096.

Figure 5.12 shows a tiling with a set of corner-based Poisson disk tiles, and Figure 5.13 shows the resulting Poisson disk distribution.

The time needed to generate a Poisson disk tile set ranges from several minutes to several hours, depending on the parameters. However, the construction of a tile set has to be done only once. With a single set of tiles, an infinite number of Poisson disk distributions can be generated.

## 5.3 OTHER METHODS

Several other methods have been proposed to efficiently generate Poisson disk distributions.

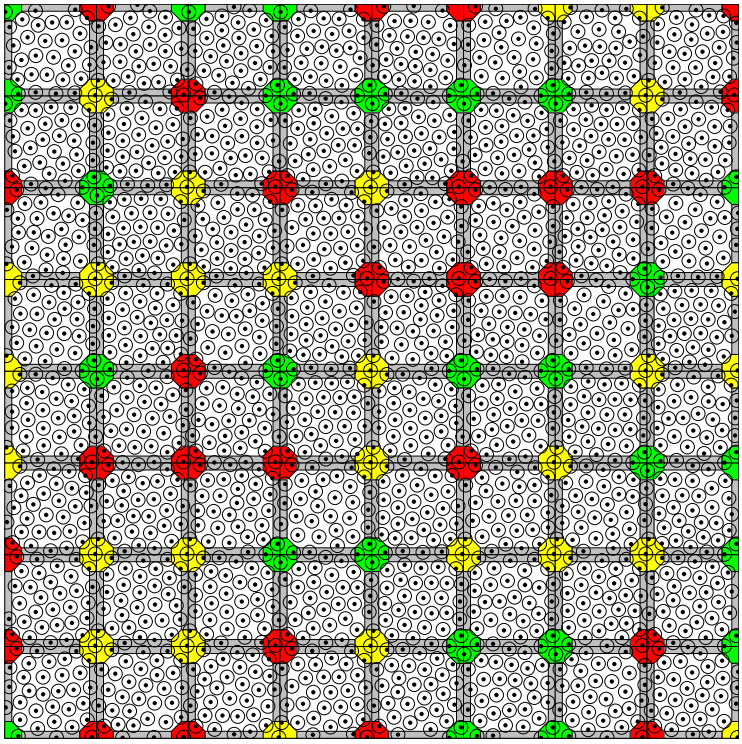


**Figure 5.12:** A tiling with a set of corner-based Poisson disk tiles.

Shade et al. (2000) presented the first tile-based approach for generating Poisson disk distributions, an extension of the dart throwing algorithm based on Wang tiles. Hiller et al. (2001) described an approach based on Lloyd's relaxation algorithm rather than dart throwing to construct a Poisson disk distribution over a set of Wang tiles. This method was later adopted by Cohen et al. (2003). Ostromoukhov et al. (2004) introduced an interesting technique for generating point distributions with blue noise properties over a given density based on the Penrose tiles. In 2005, Lagae and Dutré (2005a) presented edge-based Poisson disk tiles, a method for constructing a Poisson disk distribution over a set of Wang tiles, and template Poisson disk tiles (Lagae and Dutré, 2005b), a method that allowed us to investigate the effect of the size of the tile set on the quality of the resulting Poisson disk distributions. In 2006, Lagae and Dutré (2006a) proposed corner-based Poisson disk tiles, the method discussed in the previous section, and corner-based Poisson sphere tiles (Lagae and Dutré, 2006c), a tile-based method for efficiently generating Poisson sphere distributions, the 3D equivalent of Poisson disk distributions. Jones (2006) and Dunbar and Humphreys (2006) presented efficient implementations of the dart throwing algorithm. Kopf et al. (2006) introduced a method for efficiently generating nonuniform Poisson disk distributions based on recur-

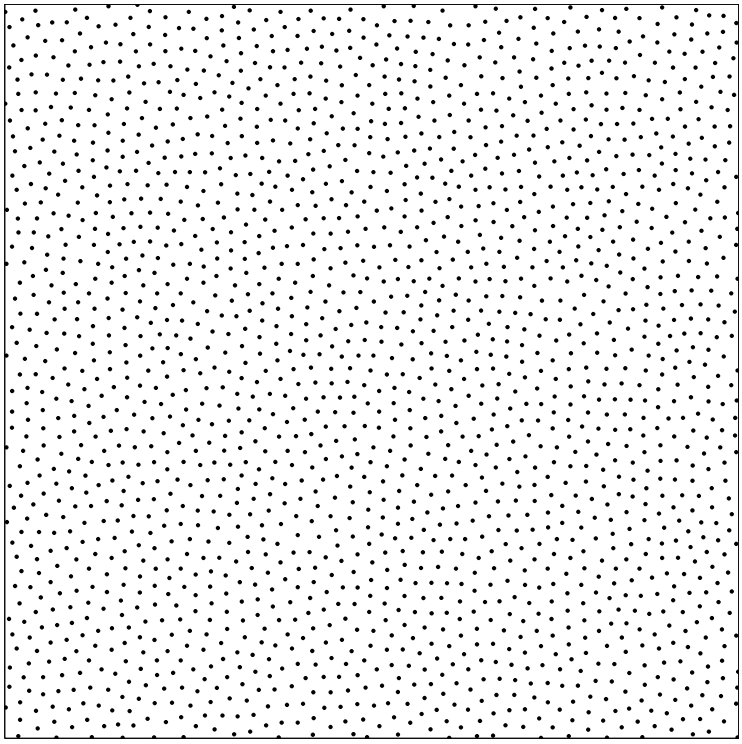


**Figure 5.13:** A Poisson disk distribution generated with a set of corner-based Poisson disk tiles. This Poisson disk distribution was generated from the tiling shown in Figure 5.12.

sive Wang tiles. Ostromoukhov et al. presented methods for fast hierarchical importance sampling with blue-noise properties based on the Penrose tiles (Ostromoukhov et al., 2004) and polyominoes (Ostromoukhov, 2007). Li et al. (????) extended tiled Poisson disk distributions to surfaces.

In 2008, Lagae and Dutré (2008) performed a detailed comparison of most of the methods mentioned in this section.

## 5.4 ANALYSIS

Figure 5.14 shows the spectral analysis of several methods for generating Poisson disk distributions. The power spectrum of Poisson disk distributions generated with corner-based Poisson disk tiles is very similar to that of distributions generated with dart throwing, which can be considered as the reference method. The tiling results in an increased anisotropy but this is not noticeable for most applications. For more details refer to (Lagae, 2007; Lagae and Dutré, 2008).

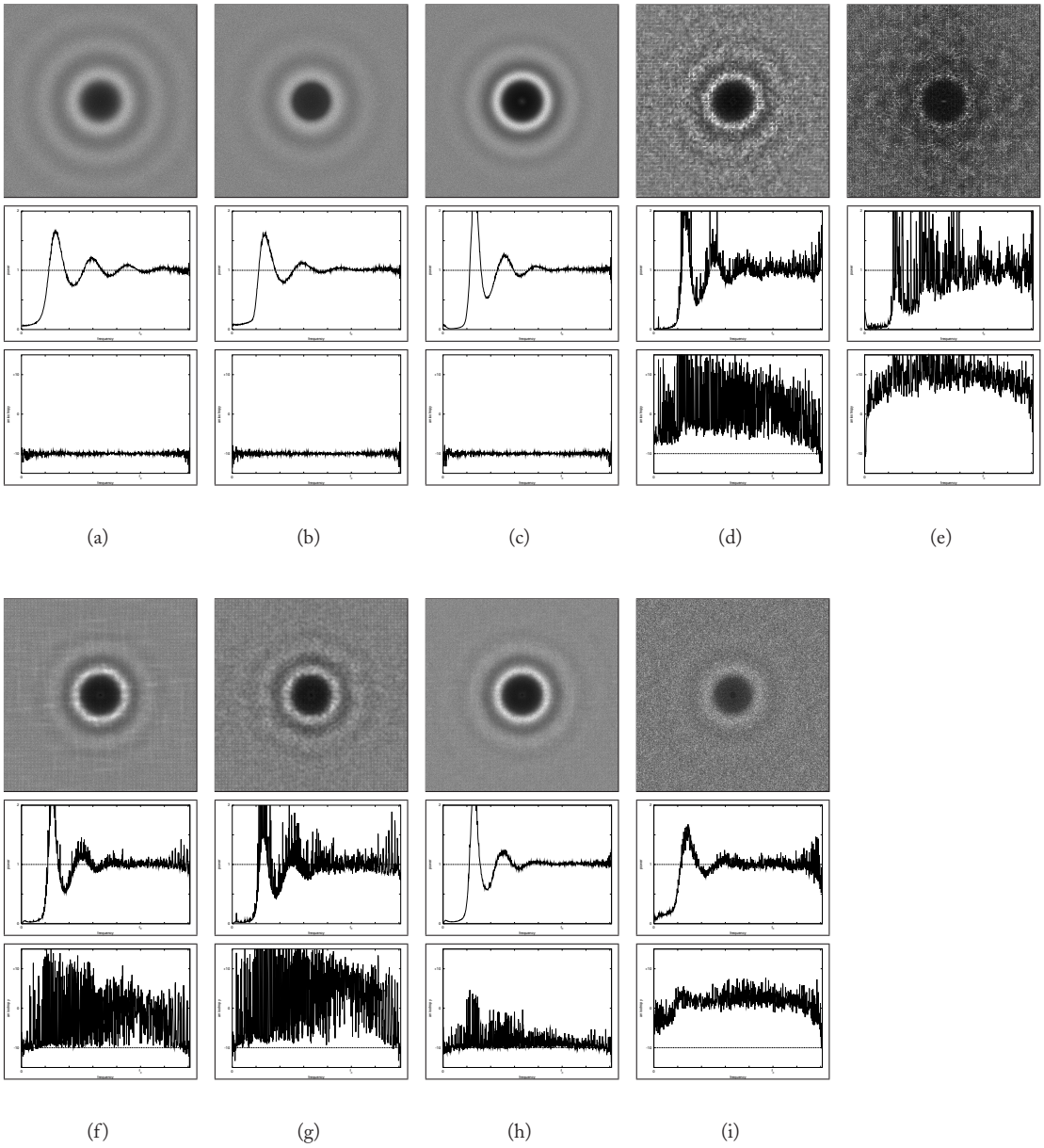



**Figure 5.14:** Spectral analysis of Poisson disk distributions generated with (a) dart throwing, (b) relaxation dart throwing, (c) Lloyd's relaxation scheme, (d) the method of Hiller et al., (e) the method of Ostromoukhov et al., (f) Edge-based Poisson disk tiles, (g) template Poisson disk tiles, (h) corner-based Poisson disk tiles, (i) the method of Kopf et al.

CHAPTER 6

# Applications of Poisson Disk Distributions

The previous chapter has introduced an efficient tile-based method for generating Poisson disk distributions. Generating Poisson disk distributions is of course not a goal in itself, Poisson disk distributions have several applications in computer graphics. Efficient methods for generating Poisson disk distributions enable efficient implementation of these applications but also enable new applications. In this chapter, several applications of Poisson disk distributions are discussed.

This chapter is organized as follows. Section 6.1 discusses sampling. Section 6.2 discusses applications in non-photorealistic rendering. Section 6.3 introduces applications in scientific visualization. Section 6.4 discusses procedural modeling, geometric object distribution and geometry instancing. Section 6.5 presents applications of Poisson disk distributions in procedural texturing.

## 6.1 SAMPLING

Poisson disk distributions were introduced in the field of computer graphics in the context of sampling. In 1977, Crow identified unwanted artifacts in digitally synthesized images, such as jaggies and moiré patterns, as instances of the aliasing problem from digital signal processing. In the mid-eighties, Dippé and Wold (1985), Cook (1986), and Mitchell (1987) introduced nonuniform sampling and the Poisson disk distribution to turn regular aliasing artifacts into perceptually less objectable stochastic noise. Their work was based on studies by Yellot (Yellot, 1982, 1983), who found that the photoreceptors in the retina of the eye are distributed according to a Poisson disk distribution, and presented theoretical evidence in favor of the Poisson disk distribution. It is now generally accepted that because of its blue noise power spectrum, the Poisson disk distribution is one of the best stochastic sampling patterns.

The Poisson disk sampling pattern allows a better reconstruction of a sampled function than other sampling patterns. This matters a lot for applications in computer graphics, which typically cannot compute enough samples to eliminate aliasing artifacts or stochastic noise. For example, the physically based rendering system of Pharr and Humphreys (2004) uses the best-candidate sampling pattern (Mitchell, 1991), an approximate Poisson disk distribution, to sample the image plane for generating primary rays. However, the Poisson disk sampling pattern is not commonly used for sampling, mainly because it is considered too difficult and too expensive to generate. The efficient methods for generating Poisson disk distributions discussed in the previous chapters enable the use of Poisson disk distributions for sampling, even for interactive and real-time applications.

Importance sampling is one of the most frequently used variance reduction techniques in global illumination and distribution ray tracing (Dutré et al., 2002; Pharr and Humphreys, 2004). Importance sampling requires nonuniform point distributions. Similar to Poisson disk distributions, nonuniform Poisson disk distributions have significant advantages over other point distributions. An example of importance sampling in the context of global illumination is sampling a high dynamic range environment map, representing an infinite area light source (Cohen and Debevec, 2001; Agarwal et al., 2003; Kollig and Keller, 2003). The environment map is replaced by a number of point light sources to speed up integration of the incoming illumination. This can be done by warping Poisson disk distributed points according to a probability density function derived from the environment map. Figure 6.1 shows environment maps sampled using warped Poisson disk distributions. However, this technique is not optimal. Warping can introduce clumping. A better method is to directly generate a non-uniform Poisson disk distribution (Ostromoukhov et al., 2004; Dunbar and Humphreys, 2006; Kopf et al., 2006).

## 6.2 NON-PHOTOREALISTIC RENDERING

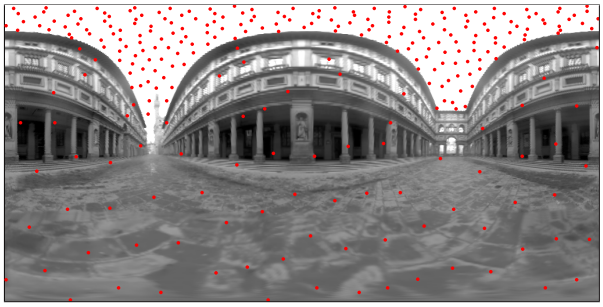
Non-photorealistic rendering (Gooch and Gooch, 2002) is an area of computer graphics that uses different rendering styles to communicate specific messages. Non-photorealistic rendering is used for artistic media simulation, user-assisted image creation and automatic image creation. Poisson disk distributions have several applications in non-photorealistic rendering.

A pen-and-ink illustration can be generated from a given image by placing a number of primitives, for example points or strokes, according to a density function derived from that image. It is widely accepted in stippling and halftoning that a Poisson disk distribution yields more visually pleasing results (Ulichney, 1987; Deussen et al., 2000; Secord et al., 2002). However, Poisson disk distributions are not frequently used because they are considered too expensive to generate. Pen-and-ink illustrations can efficiently be generated by warping or redistributing Poisson disk distributed points using the inverse cumulative of the density function. Figure 6.2 shows illustrations generated using warped Poisson disk distributions.

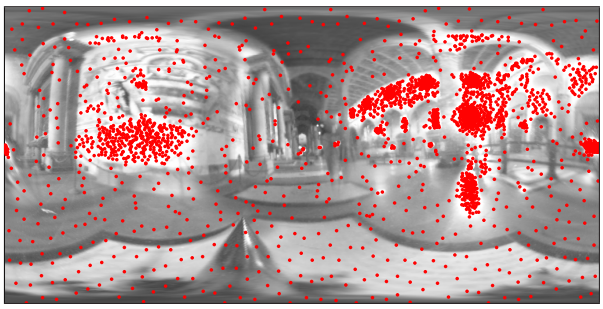
Non-photorealistic rendering also employs several other techniques introduced in previous chapters to simulate artistic styles. For example, Kaplan and Salesin (2000) used the theory of tiling to create images much like the ones by the Dutch artist M. C. Escher, and Hausner (2001) used Lloyd's relaxation method to simulate decorative mosaics.

## 6.3 SCIENTIFIC VISUALIZATION

Scientific visualization (Tufte, 1986) is a field of research that creates images, diagrams, or animations from complex scientific data. Like non-photorealistic rendering, the goal is to convey specific messages. Poisson disk distributions have several applications in scientific visualization. For example, a vector field can be visualized by sampling the vector field using icons (Tufte, 1986). The best results are obtained when the icons are placed according to a Poisson disk distribution. Scientific visual-



(a) The Uffizi Gallery, 288 points



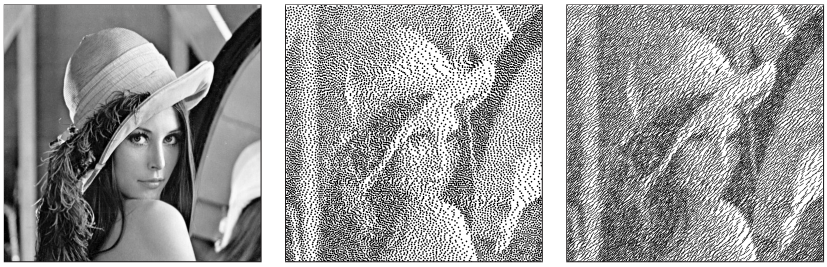
(b) Galileo's Tomb, 3, 200 points

**Figure 6.1:** Environment map sampling using warped Poisson disk distributed points. The (a) The Uffizi Gallery and (b) Galileo's Tomb environment maps were sampled with (a) 288 and (b) 3, 200 point light sources, by warping Poisson disk distributions generated with edge-based Poisson disk tiles. The sampling patterns were generated in approximately 150 ms. (The environment maps used in this figure are courtesy of Paul Debevec.)

ization also employs other techniques introduced in previous chapters to create illustrations. For example, Lu and Ebert (2005) used Wang cubes with point distributions to create example-based volume illustrations.

## 6.4 PROCEDURAL MODELING, GEOMETRIC OBJECT DISTRIBUTION, AND GEOMETRY INSTANCING

Modeling the real world is an important aspect of computer graphics. However, modeling complex environments such as plant ecosystems or cities by hand can be very time-consuming. Procedural



(a) (b) (c)

**Figure 6.2:** Primitive distribution for illustration using warped Poisson disk distributed points. (a) The Lena image. (b) Stippled and (c) hatched non-photorealistic renderings generated from the Lena image, by warping Poisson disk distributions generated with edge-based Poisson disk tiles. Approximately 13, 000 primitives were distributed.

modeling techniques assist the user to create complex environments, or create complex environments automatically. For example, Deussen et al. (1998) proposed a system for creating complex plant ecosystems and Parish and Müller (2001) presented a method for modeling cities.

Geometric object distribution is an important aspect of procedural modeling. Many man-made and natural distributions follow a pattern with a minimum distance criterion. For example, the trees in a forest and the individual hairs in fur. These distributions can easily be modeled using Poisson disk distributions. Figure 6.3 shows a beech forest in the winter. The trees were distributed according to a Poisson disk distribution. Figure 6.4 shows a planet with an asteroid belt. The asteroid belt was modeled by cutting out a ring of points from a Poisson sphere distribution.

Geometry instancing is frequently used to efficiently implement geometric object distribution. Instead of using a unique geometric model for each distributed object, only a limited set of geometric models is used, and each distributed object is an instance of one of these models. The instances may have differentiating parameters, such as orientation, size and color. This technique was also used in Figures 6.3 and 6.4.

However, for very large or complex environments, placing and storing all instances is still expensive. This problem can be relieved by using the tile-based methods for generating Poisson disk distributions introduced in Chapter 5. Because the direct stochastic tiling algorithm allows to efficiently evaluate a Poisson disk distribution locally, it enables on the fly instancing. This eliminates the cost of storing instancing information. This principle could also be used in real-time applications, such as flight simulators or games.

**Figure 6.3:** A beech forest in the winter. Over 2, 000 instances of 5 beeches were distributed using Poisson disk tiles to create this beech forest. Each beech consists of about 16, 000 triangles. (The beeches were generated with NatFX from Bionatics by Karl vom Berge. The environment map was created using the Utah sky model.)

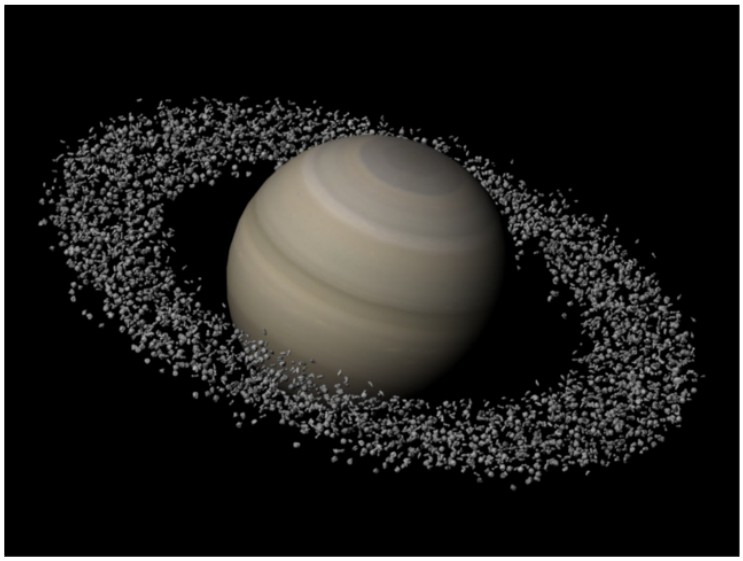


**Figure 6.4:** A planet with an asteroid belt. The asteroid belt was modeled by instancing several thousand asteroids using a Poisson sphere distribution. (The map of Saturn is courtesy of Björn Jónsson. The asteroid models are courtesy of Scott Hudson.)

## 6.5 PROCEDURAL TEXTURING

Texture mapping (Catmull, 1974) is commonly used to increasing the visual complexity of computer-generated images without adding geometric detail. A texture is mapped onto the surface of a shape to add color or detail to the shape. Traditional textures are raster graphics images. Raster graphics images have several disadvantages. Raster graphics images have a fixed resolution and size, and have large storage requirements.

Procedural textures are textures defined by a procedure or an algorithm rather than by a raster graphics image. Compared to traditional textures, procedural textures are compact, have no fixed resolution and size, and can be easily parameterized. Procedural texturing has become an invaluable tool for high-quality image synthesis. Procedural techniques are capable of generating a large variety of convincing textures, such as marble, wood and stone.

At the heart of procedural texturing are texture basis functions. They bootstrap the visual complexity which is present in the generated textures. The most famous texture basis function is Perlin's noise function (Perlin, 1985), or as Peachy states, "the function that launched a thousand textures" (Ebert et al., 2002). However, the use of texture basis functions is not limited to procedural texturing. Texture basis functions are also used in procedural modeling, shading, and animation. This large variety of applications is a motivation to find new texture basis functions and expand the range of textures that can be generated procedurally.

In this section, a procedural object distribution function is presented. This texture basis function distributes procedurally generated objects over a procedurally generated texture, which serves as background. Objects are placed uniformly over the texture, and are guaranteed not to overlap. The texture basis function allows intuitive control over the scale, size and orientation of the objects being distributed, and can be evaluated efficiently. The history and background of procedural texturing is discussed, and a two-dimensional as well as a tree-dimensional procedural object distribution function is presented.

### 6.5.1 HISTORY AND BACKGROUND

The introduction of solid texturing by Perlin (1985) and Peachy (1985) in the mid-eighties was a milestone in the field of procedural modeling.

The most popular three-dimensional texture basis function is Perlin's noise function (Perlin, 1985; Perlin and Hoffert, 1989; Perlin, 2002). The noise value at each point is determined by computing a pseudo-random gradient at each of the eight nearest vertices on the integer cubic lattice, followed by splined interpolation. Perlin's noise function has become the standard way to model natural materials such as marble, wood and stone, and natural phenomena such as smoke, water and fire. Although presented in 1985, the Perlin's texture basis function is still heavily used nowadays.

Another useful 3D texture basis function is the cellular texture basis function of Worley (1996). Random feature points are scattered throughout space, and the function returns the distance to the closest feature points. This process is accelerated using space subdivision: feature points are generated on the fly, in the cubes defined by the integer lattice. Worley's texture basis function is suited for

generating rocks, tiled areas, and a variety of organic patterns. Worley introduced his cellular texture basis function in 1996, although a simpler version of this texture basis function was already proposed in 1988 by Burchill.

To address a number of shortcomings of Perlin's noise function, Cook and DeRose (2005) presented wavelet noise, a band-limited version of Perlin's noise function. Their work was inspired by earlier work by Lewis (1989).

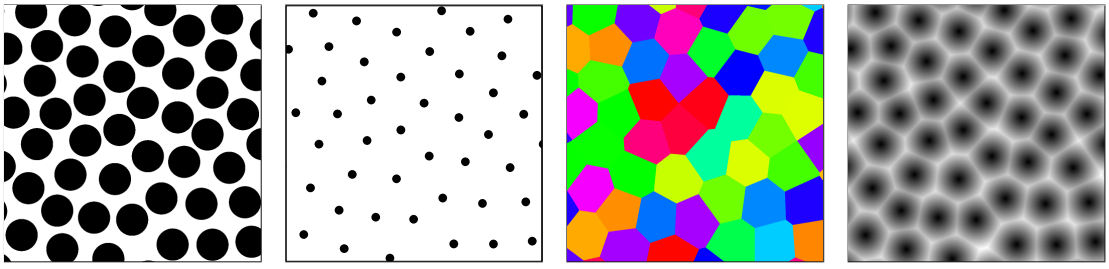
There are several other techniques to generate textures procedurally. For example, Turk (1991) presented a biologically inspired method, called reaction-diffusion, that generates interesting mammalian patterns. These methods, however, do not qualify as texture basis functions, because they do not have the semantics of a point evaluation, but require global operations to work.

For an excellent overview of the field of procedural texturing and modeling, refer to (Ebert et al., 2002).

### 6.5.2 A 2D PROCEDURAL OBJECT DISTRIBUTION FUNCTION

In 2006, Lagae and Dutré (2005a) presented a two-dimensional procedural object distribution function, a new texture basis function that distributes procedurally generated objects over a procedurally generated background.

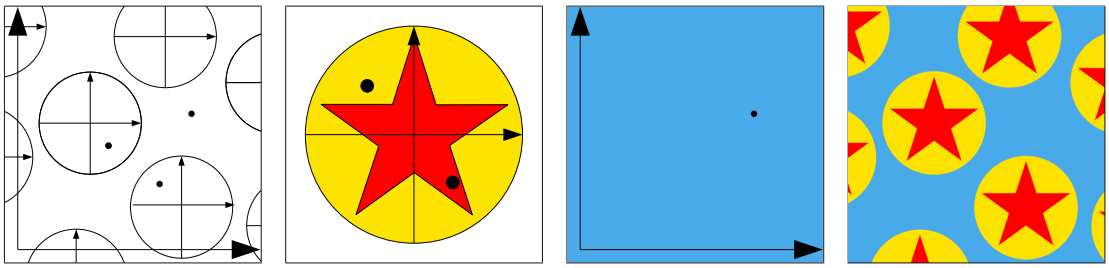
The texture basis function is defined over the infinite plane. When evaluated, it returns the point in a tiled Poisson disk distribution closest to the point of evaluation, and a unique identifier for this point. The function also returns the distance to the closest point, and a boolean value indicating whether the point of evaluation is within the Poisson disk of the closest point. This is illustrated in Figure 6.5.



(a) (b) (c) (d)

**Figure 6.5:** Evaluation of the 2D object distribution texture basis function. The texture basis function returns (a) a boolean value indicating whether the point of evaluation is within the Poisson disk of the closest feature point, (b) the coordinates of the closest feature point, (c) a unique ID identifying the closest feature point, and (d) the distance to the closest feature point.

To distribute procedural objects over a procedural background, the texture basis function is evaluated. If the point of evaluation lies within a Poisson disk, it is transformed to the local coordinate system of that disk, and a procedural object is evaluated. If the point of evaluation is not located inside a disk, a procedural texture which serves as background is evaluated. This process is illustrated in Figure 6.6.



(a) (b) (c) (d)

**Figure 6.6:** Procedural object distribution with the 2D object distribution texture basis function. (a) The texture basis function is evaluated. (b) If the point of evaluation lies within a Poisson disk, it is transformed to the local coordinate system of that disk, and a procedural object is evaluated. (c) If the point of evaluation is not located inside a Poisson disk, a procedural texture which serves as background is evaluated. (d) The resulting procedural texture.

The remainder of this subsection discusses how to evaluate the texture basis function efficiently, and how to control the placement of the distributed objects.

#### 6.5.2.1 Evaluation

Evaluation of the texture basis function is straightforward. The Poisson disk tile that contains the point of evaluation $(x, y)$ is located at tile coordinates $(\lfloor x \rfloor, \lfloor y \rfloor)$, and is provided by the direct stochastic tiling algorithm. The tile and its neighbors are then searched for the closest point. The unique identifier of the closest point is a combination of the hash value of the tile coordinates of the tile where the closest point was found, and the index of the closest point in that tile.

Only a single Poisson disk tile set is needed. Randomness is introduced by the direct stochastic tiling algorithm, randomizing the texture basis function is done by randomizing the permutation table used by the hash function of the tiling algorithm.

The texture basis function can be implemented using edge-based Poisson disk tiles, template Poisson disk tiles or corner-based Poisson disk tiles. Corner-based Poisson disk tiles are recommended because the tile sets are smaller and the tiling algorithms are more efficient.

Several optimizations are employed to evaluate the texture basis function efficiently. After constructing a Poisson disk tile set, the points in the tiles are sorted lexicographically. This speeds up the location of the closest point. Also note that if the distance to a candidate closest point is

less than the Poisson disk radius, it must be the closest point. The largest empty circle optimization limits the number of neighboring tiles that has to be searched while locating the closest point. During construction of the Poisson disk tile set, the radius of the largest empty circle $r_e$ is computed. Alternatively, $r_e$ can be bounded analytically. This radius determines different regions in the tile, much like the ones in Figure 5.3(a). If the point of evaluation $(x, y)$ is closer to a corner than $r_e$, three neighboring tiles have to be considered. Else, if $(x, y)$ is closer to an edge than $r_e$, one neighboring tile needs to be considered. In all other cases, the closest point must lie within the same tile as $(x, y)$. This optimization is very effective. For a Poisson disk tile set with $N = 32$ points per tile, and $\alpha = 0.75$, $r_e$ was approximately 0.16. For about 10% of the evaluations, four tiles had to be considered. Roughly 40% of the evaluations required two tiles, and for almost 50% of the evaluations, only a single tile was visited.

Due to these optimizations, the texture basis function can be evaluated very efficiently. In our implementation, one evaluation of the new texture basis function is as expensive as 5 evaluations of Perlin's two-dimensional noise function. This makes the texture basis function also suited for interactive and real-time applications.

#### 6.5.2.2 Parameters

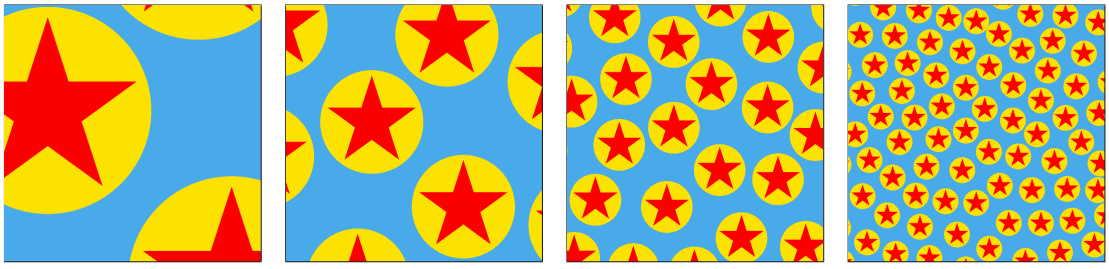
The placement of the distributed objects can be controlled by four parameters: the scale $s$, the size $r$, the orientation $\theta$, and the aspect ratio $a$.

To decouple the texture basis function as much as possible from the underlying tiled Poisson disk distribution, a scale parameter $s$ is introduced that controls the density of objects. A scale of $s$ corresponds to an object density of $s$ objects per unit square. Controlling the scale of the texture basis function is done by scaling the domain over which it is evaluated. To obtain an object density of $s$, the tiled Poisson disk distribution is scaled by a factor of $\sqrt{S/N}$, where $N$ is the number of points per tile. Figure 6.7 shows a procedural texture for different values of the scale parameter. Note that this scale parameter is different from the original scale parameter introduced in (Lagae and Dutré, 2005a). The new scale parameter is more intuitive and easier to use.

When the texture basis function is evaluated, and the point of evaluation lies within a disk, it is transformed to the local coordinate system of that disk. These coordinates are then used to evaluate the procedural object. Manipulating the size $r$ and orientation $\theta$ of the distributed objects is done by scaling the local coordinate system by a factor $r \in [0, 1]$, and rotating it by an angle $\theta \in [0, 2\pi]$, before evaluating the procedural object. Figure 6.8 shows a procedural texture for different values of the size and orientation parameters.

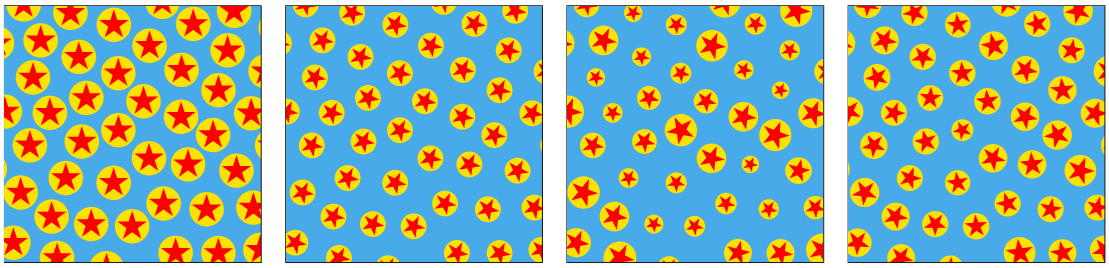
By introducing the aspect ratio $a$, a very general and flexible object distribution function is obtained. As Figure 6.9 shows, distributions of local coordinate systems can be generated procedurally using only four intuitive parameters. Arbitrary procedural content can be placed in these coordinate systems.

Object attributes, such as size, orientation, and aspect ratio, can be chosen at random on a per-object basis. However, some care must be taken. Although each object may have different attributes, all evaluations of the texture basis function involving the same object must produce the



(a) (b) (c) (d)

**Figure 6.7:** Manipulation of the scale $s$ of the 2D object distribution texture basis function. (a) $s = 1$. (b) $s = 4$. (c) $s = 16$. (d) $s = 64$. Note that each image is a closeup of the next one.



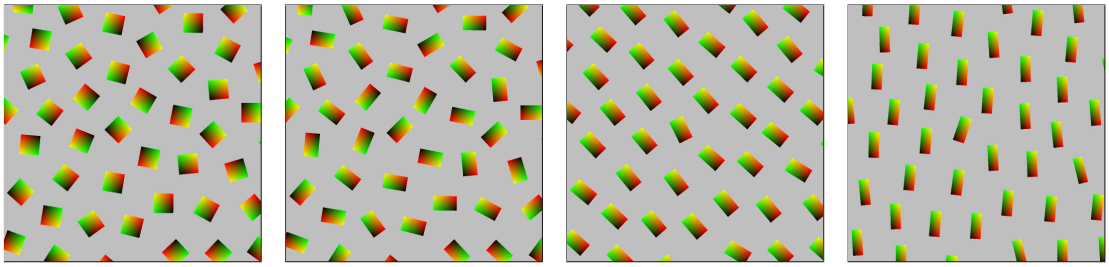
(a) (b) (c) (d)

**Figure 6.8:** Manipulation of the size $r$ and orientation $\theta$ of the 2D object distribution texture basis function. (a) $r = 1$, $\theta = 0$. (b) $r = 0.75$, $\theta = \pi/4$. (c) $r \sim U(0.5, 1)$, $\theta \sim N(\pi/4, \pi/32)$. (d) $r \sim N(0.8, 0.05)$, $\theta \sim U(0, 2\pi)$. The scale $s$ of all procedural textures is 36.

same random values for the attributes. This is why the texture basis function provides a unique identifier associated with each disk. When used to seed a random number generator, for example a fast linear congruential generator, random attributes can be generated correctly on a per-object basis. The unique identifier can also be used to generate additional object attributes.

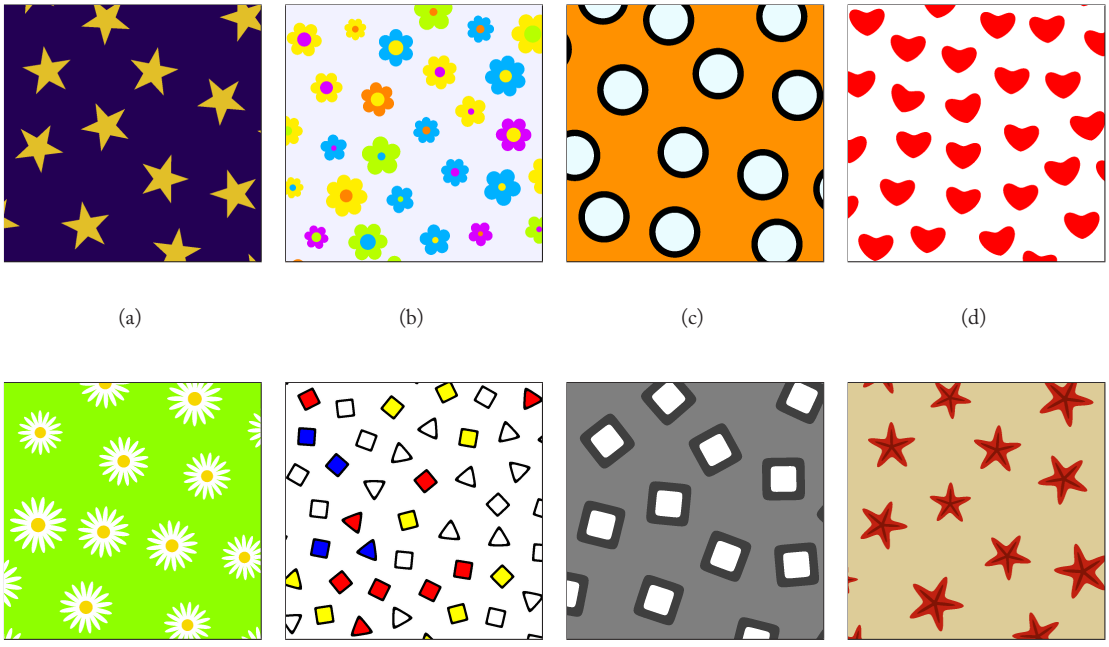
#### 6.5.2.3 Examples

The procedural object distribution function extends the range of textures that can be generated procedurally. Figures 6.10, 6.11, and 6.12 show several procedural textures generated with the new texture basis function. They demonstrate the procedural object distribution function for several settings of the scale, size and orientation parameters. Like all procedural textures, these textures have no fixed resolution and size, and can be easily parameterized.



(a) (b) (c) (d)

**Figure 6.9:** Manipulation of the aspect ratio $a$ of the 2D object distribution texture basis function. These procedural textures show a color encoding of the local coordinate systems. (a) $\theta \sim U(0, 2\pi)$, $a = 1$. (b) $\theta \sim U(0, 2\pi)$, $a = \phi$ (the golden ratio, $\phi \approx 1.6180$). (c) $\theta \sim N(\pi/4, \pi/32)$, $a = \phi$. (d) $\theta \sim N(0, \pi/32)$, $a \sim N(2.5, 0.1)$. The scale $s$ and size $r$ of all procedural textures is 36 and 0.80, respectively.



(a) (b) (c) (d)

(e) (f) (g) (h)

**Figure 6.10:** Textures generated with the 2D object distribution texture basis function. (a) Stars. (b) Flowers. (c) Polka dots. (d) Hearts. (e) Daisies. (f) Mondriaan shapes. (g) Abstract squares. (h) Starfish.

**Figure 6.11:** Dresses worn by the Venus model. The dresses are textured with the procedural textures shown in Figure 6.10.

A lot of interesting procedural objects can be generated with the so called superformula (Gielis, 2003; Gielis et al., 2003). The heart shape of Figure 6.10(d) is based on the polar equation $r(\theta) = \cos 5\theta - 5\cos\theta$. A single petal of a daisy of Figure 6.10(e) was created using an exponentiated cosine lobe. The texture of Figure 6.10(f) is inspired by Mondriaan's painting *Composition with red, yellow, and blue*. The parameters for the texture basis function are $r = 0.8$ and $\theta \sim U(0, 2\pi)$. The rounded triangle is a supershape with parameters $m = 3, n_1 = 6.7, n_2 = n_3 = 12$, and $a = b = 1$, and the rounded rectangle is a supershape with parameters $m = 4, n_1 = n_2 = n_3 = 12$, and $a = b = 1$. The rounded rectangle of Figure 6.10(g) is a supershape with parameters $m = 4$, $n_1 = n_2 = n_3 = 8$, and $a = b = 1$. The starfish of Figure 6.10(h) consists of two supershapes. The parameters for the outer one are $m = 5$, $n_1 = 2$, $n_2 = n_3 = 7$, and $a = b = 1$, and the parameters for the inner one are $m = 5$, $n_1 = 2$, $n_2 = n_3 = 13$, and $a = b = 1$. The particles in the granite of Figure 6.12 are

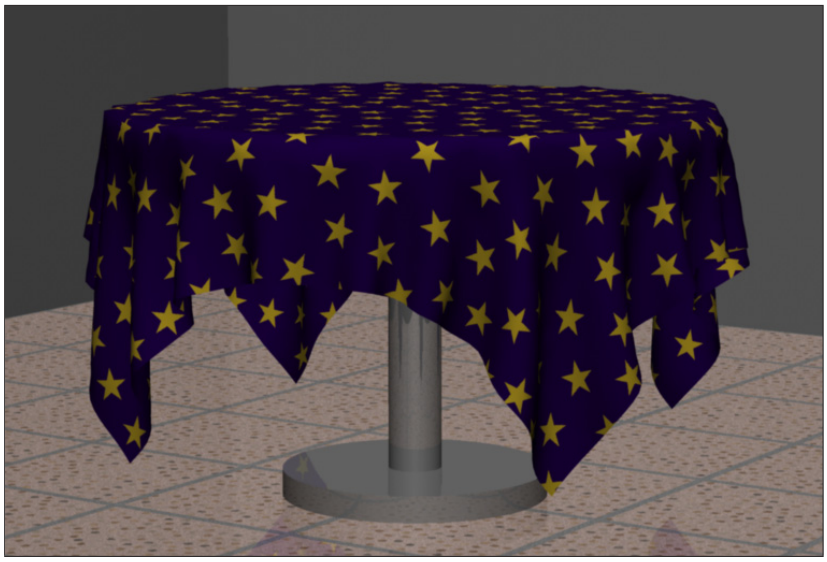


**Figure 6.12:** A table with a table cloth on a granite floor. The textures used in this scene were generated with the 2D procedural object distribution function.

random convex hexagons. The color of these particles, the color of the mortar and the base color were modulated with Perlin noise.

#### 6.5.2.4 Discussion

By modifying the hash function used in the direct stochastic tiling algorithm, seamless textures can be created. For example, evaluating the hash function with tile coordinates modulo $M$ results in a periodic tiling with period $M$, and can be used to produce a toroidally wrapping texture to cover a cylinder or texture.

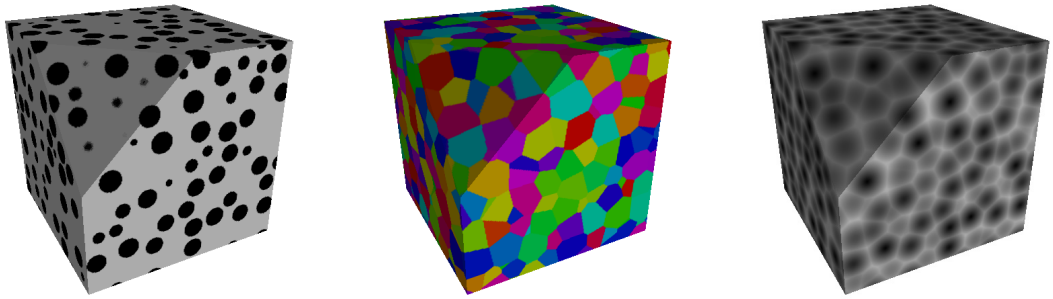
The procedural object distribution function is somewhat similar to the cellular texture basis function of Worley (1996). However, the cellular texture basis function of Worley uses feature points randomly scattered in space, and therefore cannot be used to distribute objects without overlap.

In general, most texture basis functions generate some kind of pseudo-random scalar value over their domain. From that perspective, the procedural object distribution function is not a typical texture basis function. However, the ultimate goal of all texture basis functions is the same: providing a solid basis for generating a large variety of textures. The procedural object distribution function does just that.

### 6.5.3 A 3D PROCEDURAL OBJECT DISTRIBUTION FUNCTION

Solid textures (Perlin, 1985; Peachy, 1985) are three-dimensional textures that simulate solid materials. When a solid texture is applied to the surface of an object, the object appears to be carved out of that material. Most texture basis functions are available in two as well as three dimensions.

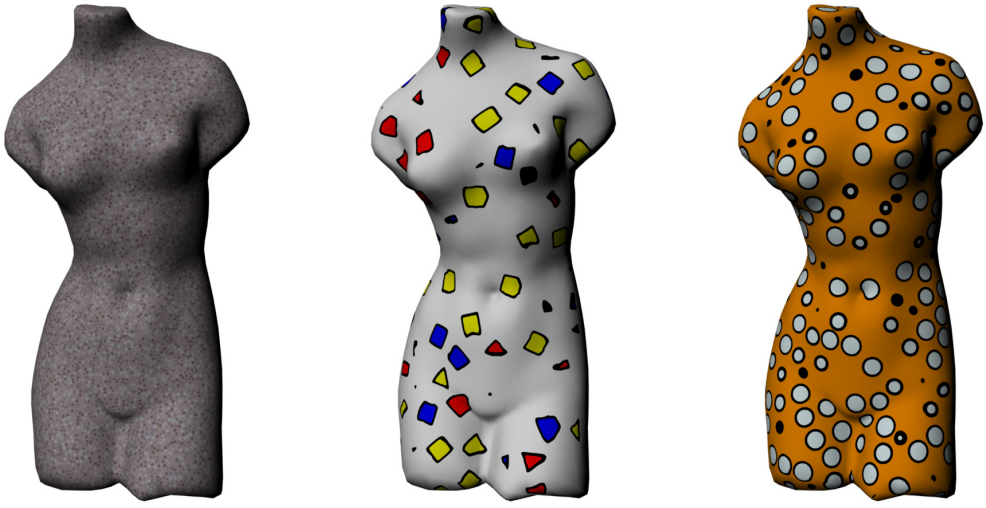
The two-dimensional procedural object distribution function easily extends to three dimensions using three-dimensional corner tiles and corner-based Poisson sphere tiles, the 3D equivalent of corner-based Poisson disk tiles (Lagae and Dutré, 2006c). Figure 6.13 shows the outputs of the three-dimensional texture basis function.



(a) (b) (c)

**Figure 6.13:** Evaluation of the 3D object distribution texture basis function. The texture basis function returns (a) a boolean value indicating whether the point of evaluation is within the Poisson disk of the closest feature point, (b) a unique ID identifying the closest feature point, and (c) the distance to the closest feature point. The coordinates of the closest feature point (not shown) are also returned.

The three-dimensional procedural object distribution function is good at modeling natural materials with particle distributions, such as granite, and abstract man-made patterns. Figure 6.14 shows several procedural solid textures generated with the texture basis function. The texture basis function has a small memory footprint and is quite efficient: one evaluation is about as expensive as 20 evaluations of Perlin's Noise function.



(a) (b) (c)

**Figure 6.14:** The Venus model carved from solid textures generated with the 3D object distribution texture basis function. (a) Granite. (b) Mondriaan shapes. (c) Polka dots.

CHAPTER 7

# Conclusion

The tile-based methods based on Wang tiles and corner tiles presented in this book enable the efficient synthesis and storage of complex signals, such as textures or point distributions.

Although the methods for constructing a complex signal over a set of Wang tiles or corner tiles are dependent on the signal, the general idea behind these methods generalizes to other kinds of signals. The methods presented in this book therefore have the potential to make the generation and storage of almost any complex signal efficient, even when no other efficient synthesis and storage methods are available.

We hope that this book enables new applications and inspires future tile-based methods.

# Bibliography


Agarwal, S., Ramamoorthi, R., Belongie, S., and Jensen, H. W. Structured importance sampling of environment maps. *ACM Transactions on Graphics*, 22(3):605–612, 2003. DOI: 10.1145/882262.882314

Ball, W. W. R and Coxeter, H.S. *Mathematical recreations and essays*. Dover Publications, 1987.

Berger, R. The undecidability of the domino problem. *Memoirs American Mathematical Society*, 66:1–72, 1966.

Bonet, J. S. D. Multiresolution sampling procedure for analysis and synthesis of texture images. In *Proceedings of ACM SIGGRAPH 1997*, pages 361–368. 1997. DOI: 10.1145/258734.258882

Burchill, L. Graphics goodies #2 - a simple, versatile procedural texture. *Computer Graphics*, 22(1):29–30, 1988. DOI: 10.1145/48155.48159

Catmull, E. E. *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Ph.D. thesis, Department of Computer Science, University of Utah, 1974.

Cohen, J. and Debevec, P. LightGen, HDRShop plugin. `http://gl.ict.usc.edu/HDRShop/lightgen/lightgen.html`, 2001.

Cohen, M. F., Shade, J., Hiller, S., and Deussen, O. Wang tiles for image and texture generation. *ACM Transactions on Graphics*, pages 287–294, 2003. DOI: 10.1145/882262.882265

Cook, R. L. Stochastic sampling in computer graphics. *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*, 5(1):51–72, 1986. DOI: 10.1145/7529.8927

Cook, R. L. and DeRose, T. Wavelet noise. *ACM Transactions on Graphics*, 24(3):803–811, 2005. DOI: 10.1145/1073204.1073264

Crow, F. C. The aliasing problem in computer-generated shaded images. *Communications of the ACM*, 20(11):799–805, 1977. DOI: 10.1145/359863.359869

Culik, II, K. An aperiodic set of 13 Wang tiles. *Discrete Mathematics*, 160(1-3):245–251, 1996. DOI: 10.1016/S0012-365X(96)00118-5

Culik, II, K. and Kari, J. An aperiodic set of Wang cubes. *Journal of Universal Computer Science*, 1(10), 1995.

Deussen, O., Hanrahan, P., Lintermann, B., Měch, R., Pharr, M., and Prusinkiewicz, P. Realistic modeling and rendering of plant ecosystems. In *Proceedings of ACM SIGGRAPH 1998*, pages 275–286. 1998. DOI: 10.1145/280814.280898

Deussen, O., Hiller, S., van Overveld, C., and Strothotte, T. Floating points: A method for computing stipple drawings. *Computer Graphics Forum*, 19(3):40–51, 2000. DOI: 10.1111/1467-8659.00396

Dippé, M. A. Z. and Wold, E. H. Antialiasing through stochastic sampling. *Computer Graphics (Proceedings of ACM SIGGRAPH 85)*, 19(3):69–78, 1985. DOI: 10.1145/325165.325182

Dunbar, D. and Humphreys, G. A spatial data structure for fast Poisson-disk sample generation. *ACM Transactions on Graphics*, 25(3):503–508, 2006. DOI: 10.1145/1141911.1141915

Dutré, P., Bala, K., and Bekaert, P. *Advanced Global Illumination*. A. K. Peters, Ltd., Natick, MA, 2002.

Ebert, D. S., Musgrave, F. K., Peachey, D., Perlin, K., and Worley, S. *Texturing and Modeling: A Procedural Approach*. Morgan Kaufmann Publishers, Inc., 2002.

Efros, A. A. and Freeman, W. T. Image quilting for texture synthesis and transfer. In *Proceedings of ACM SIGGRAPH 2001*, pages 341–346. 2001. DOI: 10.1145/383259.383296

Efros, A. A. and Leung, T. K. Texture synthesis by non-parametric sampling. In *International Conference on Computer Vision*, pages 1033–1038. 1999. DOI: 10.1109/ICCV.1999.790383

Escher, M. C. and Locher, J. C. *The World of M. C. Escher*. Abrams, New York, NY, 1971.

Fu, C.-W. and Leung, M.-K. Texture tiling on arbitrary topological surfaces using Wang tiles. In *Rendering Techniques 2005*, pages 99–104. 2005.

Gielis, J. A generic geometric transformation that unifies a wide range of natural and abstract shapes. *American Journal of Botany*, 90(3):333–338, 2003. DOI: 10.3732/ajb.90.3.333

Gielis, J., Beirinckx, B., and Bastiaens, E. Superquadrics with rational and irrational symmetry. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, pages 262–265. 2003. DOI: 10.1145/781606.781647

Glassner, A. *Andrew Glassner's notebook: recreational computer graphics*. Morgan Kaufmann Publishers, Inc., San Fransisco, CA, 1999.

Gooch, B. and Gooch, A. *Non-Photorealistic Rendering*. A. K. Peters, Ltd., Natick, MA, 2002.

Grünbaum, B. and Shepard, G. C. *Tilings and patterns*. W. H. Freeman and Company, 1986.

Hausner, A. Simulating decorative mosaics. In *Proceedings of ACM SIGGRAPH 2001*, pages 573–580. 2001. DOI: 10.1145/383259.383327

Heeger, D. J. and Bergen, J. R. Pyramid-based texture analysis/synthesis. In *Proceedings of ACM SIGGRAPH 1995*, pages 229–238. 1995. DOI: 10.1145/218380.218446

Hiller, S., Deussen, O., and Keller, A. Tiled blue noise samples. In *Vision, Modeling, and Visualization 2001*, pages 265–272. 2001.

Jones, T. R. Efficient generation of Poisson-disk sampling patterns. *Journal of Graphics Tools*, 11(2):27–36, 2006.

Kaplan, C. S. *Computer Graphics and Geometric Ornamental Design*. Ph.D. thesis, Department of Computer Science and Engineering, University of Washington, Seattle, 2002.

Kaplan, C. S. and Salesin, D. H. Escherization. In *Proceedings of ACM SIGGRAPH 2000*, pages 499–510. 2000. DOI: 10.1145/344779.345022

Kaplan, C. S. and Salesin, D. H. Islamic star patterns in absolute geometry. *ACM Transactions on Graphics*, 23(2):97–119, 2004. DOI: 10.1145/990002.990003

Klassen, R. V. Filtered jitter. *Computer Graphics Forum*, 19(4):223–230, 2000. DOI: 10.1111/1467-8659.00459

Knuth, D. E. *The art of computer programming*, volume 1. Addison-Wesley, Reading, MA, 1968.

Kollig, T. and Keller, A. Efficient illumination by high dynamic range images. In *Proceedings of the 14th Eurographics workshop on Rendering*, pages 45–50. 2003.

Kopf, J., Cohen-Or, D., Deussen, O., and Lischinski, D. Recursive Wang tiles for real-time blue noise. *ACM Transactions on Graphics*, 25(3):509–518, 2006. DOI: 10.1145/1141911.1141916

Kwatra, V., Essa, I., Bobick, A., and Kwatra, N. Texture optimization for example-based synthesis. *ACM Transactions on Graphics*, 24(3):795–802, 2005. DOI: 10.1145/1073204.1073263

Kwatra, V., Schödl, A., Essa, I., Turk, G., and Bobick, A. Graphcut textures: image and video synthesis using graph cuts. *ACM Transactions on Graphics*, 22(3):277–286, 2003. DOI: 10.1145/882262.882264

Lagae, A. *Tile-Based Methods in Computer Graphics*. Ph.D. thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2007.

Lagae, A. and Dutré, P. A procedural object distribution function. *ACM Transactions on Graphics*, 24(4):1442–1461, 2005a. DOI: 10.1145/1095878.1095888

Lagae, A. and Dutré, P. Template Poisson disk tiles. Report CW 413, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2005b.

Lagae, A. and Dutré, P. An alternative for Wang tiles: Colored edges versus colored corners. *ACM Transactions on Graphics*, 25(4):1442–1459, 2006a. DOI: 10.1145/1183287.1183296

Lagae, A. and Dutré, P. Long period hash functions for procedural texturing. In *Vision, Modeling, and Visualization 2006*, pages 225–228. Akademische Verlagsgesellschaft Aka GmbH, Berlin, 2006b.

Lagae, A. and Dutré, P. Poisson sphere distributions. In *Vision, Modeling, and Visualization 2006*, pages 373–379. Akademische Verlagsgesellschaft Aka GmbH, Berlin, 2006c.

Lagae, A. and Dutré, P. The tile packing problem. Report CW 461, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2006d.

Lagae, A. and Dutré, P. The tile packing problem. *Geombinatorics*, 17(1), 2007.

Lagae, A. and Dutré, P. A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum*, 21(1):114âŁ“129, 2008.

Lagae, A., Kari, J., and Dutré, P. Aperiodic sets of square tiles with colored corners. Report CW 460, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2006.

L'Ecuyer, P. Efficient and portable combined random number generators. *Communications of the ACM*, 31(6):742–749, 1988. DOI: 10.1145/62959.62969

Lefebvre, S. and Neyret, F. Pattern based procedural textures. In *Proceedings of the 2003 Symposium on Interactive 3D Graphics*, pages 203–212. 2003. DOI: 10.1145/641480.641518

Leung, M.-K., Pang, W.-M., Fu, C.-W., Wong, T.-T., and Heng, P.-A. Tileable BTF. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, 13(5):953–965, 2007. DOI: 10.1109/TVCG.2007.1034

Lewis, J. P. Algorithms for solid noise synthesis. *Computer Graphics (Proceedings of ACM SIGGRAPH 89)*, 23(3):263–270, 1989. DOI: 10.1145/74334.74360

Li, H., Lo, K.-Y., Leung, M.-K., and Fu, C.-W. Dual Poisson-disk tiling: An efficient method for distributing features on arbitrary surfaces. *Visualization and Computer Graphics, IEEE Transactions on*, 14(3):982–998, 2008. DOI: 10.1109/TVCG.2008.53

Liang, L., Liu, C., Xu, Y.-Q., Guo, B., and Shum, H.-Y. Real-time texture synthesis by patch-based sampling. *ACM Transactions on Graphics*, 20(3):127–150, 2001. DOI: 10.1145/501786.501787

Liu, Y., Lin, W.-C., and Hays, J. Near-regular texture analysis and manipulation. *ACM Transactions on Graphics*, 23(3):368–376, 2004. DOI: 10.1145/1015706.1015731

Lloyd, S. P. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982. DOI: 10.1109/TIT.1982.1056489

Lo, K.-Y., Li, H., Fu, C.-W., and Wong, T.-T. Interactive reaction-diffusion on surface tiles. In *Proceedings of Pacific Graphics 2007*, pages 65–74. 2007.

Lu, A. and Ebert, D. S. Example-based volume illustrations. In *Proceedings of IEEE Visualization*, pages 655–662. 2005. DOI: 10.1109/VISUAL.2005.1532854

Lu, A., Ebert, D. S., Qiao, W., Kraus, M., and Mora, B. Volume illustration using Wang cubes. 26(2), 2007. DOI: 10.1145/1243980.1243985

MacMahon, M. P. A. *New mathematical pastimes*. Cambridge University Press, Cambridge, 1921.

Matsumoto, M. and Nishimura, T. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998. DOI: 10.1145/272991.272995

McCool, M. and Fiume, E. Hierarchical Poisson disk sampling distributions. In *Proceedings of Graphics Interface '92*, pages 94–105. 1992.

Mitchell, D. P. Generating antialiased images at low sampling densities. *Computer Graphics (Proceedings of ACM SIGGRAPH 87)*, 21(4):65–72, 1987. DOI: 10.1145/37402.37410

Mitchell, D. P. Spectrally optimal sampling for distribution ray tracing. *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, 25(4):157–164, 1991. DOI: 10.1145/127719.122736

Neyret, F. and Cani, M.-P. Pattern-based texturing revisited. In *Proceedings of ACM SIGGRAPH 1999*, pages 235–242. 1999. DOI: 10.1145/311535.311561

Ng, T.-Y., Wen, C., Tan, T.-S., Zhang, X., and Kim, Y. J. Generating an $\omega$-tile set for texture synthesis. In *Proceedings of Computer Graphics International 2005*, pages 177–184. 2005. DOI: 10.1109/CGI.2005.1500411

Ostromoukhov, V. Sampling with polyominoes. *ACM Transactions on Graphics*, 26(3), 2007. DOI: 10.1145/1276377.1276475

Ostromoukhov, V., Donohue, C., and Jodoin, P.-M. Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics*, 23(3):488–495, 2004. DOI: 10.1145/1015706.1015750

Parish, Y. I. H. and Müller, P. Procedural modeling of cities. In *Proceedings of ACM SIGGRAPH 2001*, pages 301–308, 2001. DOI: 10.1145/383259.383292

Peachy, D. R. Solid texturing of complex surfaces. *Computer Graphics (Proceedings of ACM SIGGRAPH 85)*, 19(3):279–286, 1985. DOI: 10.1145/325165.325246

Penrose, R. The rôle of aesthetics in pure and applied mathematical research. *Bulletin of the Institute of Mathematics and its Applications*, 10:266–271, 1974.

Perlin, K. An image synthesizer. *Computer Graphics (Proceedings of ACM SIGGRAPH 85)*, 19(3):287–296, 1985. DOI: 10.1145/325165.325247

Perlin, K. Improving noise. *ACM Transactions on Graphics*, pages 681–682, 2002. DOI: 10.1145/566570.566636

Perlin, K. and Hoffert, E. M. Hypertexture. *Computer Graphics (Proceedings of ACM SIGGRAPH 89)*, 23(3):253–262, 1989. DOI: 10.1145/74334.74359

Pharr, M. and Humphreys, G. *Physically Based Rendering*. Morgan Kaufmann Publishers, Inc., San Fransisco, CA, 2004.

Saladin, H. *L'Alhambra de Grenade*. Morance, Paris, France, 1926.

Secord, A., Heidrich, W., and Streit, L. Fast primitive distribution for illustration. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 215–226. 2002.

Shade, J., Cohen, M. F., and Mitchell, D. P. Tiling layered depth images. Technical report, University of Washington, Department of Computer Science and Engineering, 2000.

Stam, J. Aperiodic texture mapping. Technical Report ERCIM-01/97-R046, European Research Consortium for Informatics and Mathematics (ECRIM), 1997.

Steinhaus, H. *Mathematical Snapshots*. Dover Publications, Inc., Mineaola, NY, 1999.

Tufte, E. R. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, CT, 1986.

Turk, G. Generating textures on arbitrary surfaces using reaction-diffusion. *Computer Graphics (Proceedings of ACM SIGGRAPH 91)*, 25(4):289–298, 1991. DOI: 10.1145/127719.122749

Ulichney, R. *Digital Halftoning*. The MIT Press, Cambridge, MA, 1987.

Wang, H. Proving theorems by pattern recognition - II. *Bell Systems Technical Journal*, 40:1–42, 1961.

Wang, H. Games, logic and computers. *Scientific American*, 213(5):98–106, 1965.

Wei, L.-Y. Tile-based texture mapping on graphics hardware. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 55–63. 2004. DOI: 10.1145/1058129.1058138

Wei, L.-Y. and Levoy, M. Fast texture synthesis using tree-structured vector quantization. In *Proceedings of ACM SIGGRAPH 2000*, pages 479–488. 2000. DOI: 10.1145/344779.345009

Wichmann, B. A. and Hill, I. D. An efficient and portable pseudo-random number generator. *Applied Statistics*, 31:188–190, 1982. DOI: 10.2307/2347988

Worley, S. A cellular texture basis function. In *Proceedings of ACM SIGGRAPH 1996*, pages 291–294. 1996. DOI: 10.1145/237170.237267

Yellot, Jr., J. I. Spectral analysis of spatial sampling by photoreceptors: Topological disorder prevents aliasing. *Vision Research*, 22:1205–1210, 1982. DOI: 10.1016/0042-6989(82)90086-4

Yellot, Jr., J. I. Spectral consequences of photoreceptor sampling in the rhesus retina. *Science*, 221:382–385, 1983. DOI: 10.1126/science.6867716

# Author Biography

**Ares Lagae** is a Postdoctoral Fellow of the Research Foundation - Flanders (FWO). He is doing research at the Computer Graphics Research Group of the Katholieke Universiteit Leuven in Belgium. His research interests include tile-based methods in computer graphics, ray-tracing, rendering, and computer graphics in general. He received a BS and MS degree in Informatics from the Katholieke Universiteit Leuven in 2000 and 2002. He received a PhD degree in Computer Science from the Katholieke Universiteit Leuven in 2007, funded by a PhD fellowship of the Research Foundation - Flanders (FWO).