Class Notes

Tile-Based Methods for Interactive Applications

Ares Lagae*
Katholieke Universiteit Leuven

Craig S. Kaplan†
University of Waterloo

Chi-Wing Fu‡
Nanyang Technological University

Victor Ostromoukhov§
University of Montreal

Johannes Kopf¶
Universität Konstanz

Oliver Deussen‖
Universität Konstanz

SIGGRAPH 2008

*e-mail: ares.lagae@cs.kuleuven.be
†e-mail: csk@cgl.uwaterloo.ca
‡e-mail: philip_cwfu@yahoo.com.hk
§e-mail: ostrom@iro.umontreal.ca
¶e-mail: johannes.kopf@uni-konstanz.de
‖e-mail: oliver.deussen@uni-konstanz.de

Class Description

Over the last years, several techniques have been demonstrated that rely on tile-based methods. A lot of interactive applications could potentially benefit from these techniques. However, the state-of-the-art is scattered over several publications, and survey works are not available. In this class we give a detailed overview of tile-based methods in computer graphics. The class consist of five parts, which are briefly covered in the following paragraphs.

Tile-Based Methods using Wang and Corner Tiles The first part of the class introduces tile-based methods in computer graphics based on Wang tiles and corner tiles. This part serves as a general introduction for the class, but also covers methods and applications based on Wang tiles and corner tiles. We introduce Wang tiles and corner tiles, and present several tiling algorithms. We discuss in detail tile-based texture mapping using graphics hardware, tile-based generation of Poisson disk distributions, and object distribution for procedural texturing. We briefly cover other applications such as sampling, non-photorealistic rendering, and geometric object distribution. The lecturer for the first part is Ares Lagae, who recently finished his PhD about tile-based methods in computer graphics [Lagae, 2007].

Periodic Tilings for Computer Graphics Applications The second part of the class introduces the mathematical and algorithmic aspects of decorative tilings such as those used by M. C. Escher. It focuses on the theory of isohedral tilings, tilings that cover the plane systematically with congruent copies of a single shape. The isohedral tilings are flexible enough to support a wide variety of applications in art and design, while admitting a compact and efficient implementation. We show how to store, manipulate and render isohedral tilings, and survey some recent applications. The lecturer for the second part is Craig Kaplan, an expert on the use of computer graphics in ornamental design Kaplan [2002].

Tile-Based Methods for Surface Modeling The third part of the class covers tile-based methods for surface modeling. Tiling is a practical and cost-effective method for

high-quality surface modeling and rendering. Rather than intensive data acquisition and synthesis, the generalized Wang tile set presented in this part of the talk allows us to seamlessly and non-periodically tile texture data on parameterized surfaces of arbitrary topology. Once we synthesize textures on tiles, we can reuse the same tile set on different surfaces and we can also instantaneously change the surface appearance by just switching the reference tile set. Further than color textures, we also extend surface tiling to include bump maps, geometry details, the BTF's, as well as Poisson disk tiling. The lecturer for the third part is Chi-Wing Fu, who wrote several papers on this topic [Fu and Leung, 2005].

Non-Periodic Tilings for Computer Graphics Applications The fourth part of the class covers an important class of non-periodic tilings and their benefits for computer graphics applications. First, the theory of Penrose tilings is presented. We show how the inherent self-similarity of Penrose tiling can be exploited in order to get efficient implementation of uniform distributions with blue-noise properties. Then, we present polyomino-based uniform distributions, and show their advantages. Finally, we explore other non-periodic tiling systems, potentially usable for computer graphics applications: dodecagonal tiling, Ammann tiling, etc. The lecturer for the fourth part is Victor Ostromoukhov who is an expert in this topic [Ostromoukhov et al., 2004; Ostromoukhov, 2007].

Tile-Based Methods for Non-Photorealistic Rendering and Landscape Modeling The fifth part of the class covers applications of tile-based methods in the fields of non-photorealistic rendering and landscape modeling [Cohen et al., 2003]. Using hierarchical tile sets one is able to create point sets with infinite density still showing Poisson disk characteristics [Kopf et al., 2006]. We will demonstrate this using a set of tiles that is recursively subdivided. This is possible because the set shows self similarity. The resulting points can be used to create stipple drawings and also distributions of plants that also show Poisson disk behavior. This will be demonstrated by an application that enables real-time modeling and rendering of complex landscapes. The lecturer for the fifth part is Johannes Kopf, who has considerable experience with tile-based design.

Intended Audience & Prerequisites

The intended audience for this class is both the experienced graphics researcher, interested in applying tile-based methods to his own research, and the graphics practitioner, interested in using the tile-based methods covered in this class.

Instructor Biographies

Ares Lagae (organizer, lecturer) 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).

Craig S. Kaplan (lecturer) Craig Kaplan is an assistant professor in the Computer Graphics Lab at the University of Waterloo. He is interested in the application of computer graphics to art and ornamental design, and in particular the use of geometry and tiling theory in graphics.

Chi-Wing Fu (lecturer) Chi-Wing Fu is an assistant professor in the School of Computer Engineering at the Nanyang Technological University in Singapore. His research interests include tile-based modeling and rendering methods, image-based rendering, texture synthesis, and large-scale astrophysical visualization. He received the BSc and MPhil degrees in computer science from the Chinese University of Hong Kong in 1997 and 1999, respectively, and the PhD degree in computer science from Indiana University at Bloomington in 2003.

Victor Ostromoukhov (lecturer) Victor Ostromoukhov is an associate professor at University of Montreal. Victor studied mathematics, physics and computer science at Moscow Phys-Tech (MIPT). After graduating in 1980, he spent several years with prominent European and American industrial companies (SG2, Paris; Olivetti, Paris and Milan; Canon Information Systems, Cupertino, CA) as a research scientist and/or computer engineer. He completed his Ph.D. in CS at Swiss Federal Institute of Technology (EPFL, Lausanne, 1995), where he continued to work as a lecturer and senior researcher. Invited professor at University of Washington, Seattle, in 1997, and at iMAGIS/INRIA, Grenoble, France,

in 2002. Research scientist at Massachusetts Institute of Technology, Cambridge, MA, in 1999-2000. His research interests are mainly in computer graphics, and more specifically in photorealistic and non-photorealistic rendering, sampling theory, color science, halftoning, and digital art. Victor is a co-author of eight SIGGRAPH papers and one SIGGRAPH course.

Johannes Kopf (lecturer) Johannes Kopf is a PhD student at the University of Konstanz in Germany. He has worked on a variety of topics ranging from tile-based methods to computational photography, texture synthesis, and mesh deformation. Johannes received his B.Sc. and diploma degrees in 2004 and 2005 from the University of Hamburg.

Oliver Deussen (contributor) Oliver Deussen is a full professor at the University of Konstanz (Germany). He works since several years on tile-based design and published some papers on this topic. He is interested in this area because many distribution problems (i.e. in non-photorealistic rendering or landscape modeling) can be solved efficiently using tile-based methods.

Contents

Part I

Tile-Based Methods using Wang Tiles and Corner Tiles

Chapter 1

Slides

Tile-Based Methods for Interactive Applications

Ares Lagae - Katholieke Universiteit Leuven
Craig S. Kaplan - University of Waterloo
Chi-Wing Fu (Philip) - Nanyang Technological University
Victor Ostromoukhov - Université de Montréal
Johannes Kopf - Universität Konstanz

Wednesday, 13 August

Goal

- A detailed overview of tile-based methods in computer graphics
- Focus on
  - Interactive techniques and applications
  - Mathematical concepts and tiling theory

2

Prerequisites

- Basic working knowledge in
  - Mathematics
  - Computer science
  - Computer graphics
- Target audience
  - Experienced graphics researchers
  - Graphics practitioner

3

Speakers

- In order of appearance
  - Ares Lagae — Katholieke Universiteit Leuven
  - Craig S. Kaplan — University of Waterloo
  - Chi-Wing Fu (Philip) — Nanyang Technological University
  - Victor Ostromoukhov — Université de Montréal
  - Johannes Kopf — Universität Konstanz

4

Schedule

08:30 - 08:35	Lagae	Introduction
08:35 - 09:15	Lagae	Tile-Based Methods using Wang and Corner Tiles
09:15 - 09:55	Kaplan	Periodic Tilings for Computer Graphics Applications
09:55 - 10:35	Fu	Tile-Based Methods for Surface Modeling
10:35 - 10:50		Break
10:50 - 11:30	Ostromoukhov	Non-Periodic Tilings for Computer Graphics Applications
11:30 - 12:10	Kopf	Tile-Based Methods for Non-Photorealistic Rendering and Landscape Modeling
12:10 - 12:15	Kopf	Conclusion

5

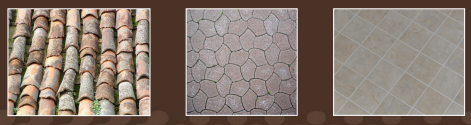
Part I

Tile-Based Methods using Wang Tiles and Corner Tiles

Ares Lagae
Katholieke Universiteit Leuven

Introduction

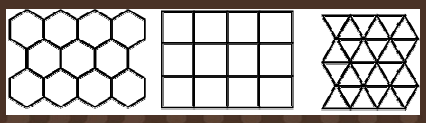
- Tiles & tilings
  - A tiling is an arrangement of plane figures that fills the plane without gaps or overlaps



7

Introduction

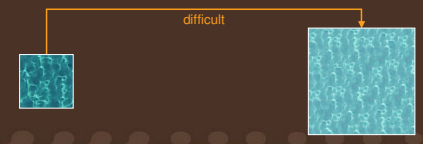
- Tiles & tilings
  - A tiling is an arrangement of plane figures that fills the plane without gaps or overlaps



8

Introduction

- Tile-based methods
  - The use of tilings to efficiently compute *signals* that are otherwise expensive to compute



9

Introduction

- Tile-based methods
  - The use of tilings to efficiently compute *signals* that are otherwise expensive to compute

10

Conclusion

- Tile-based methods for interactive applications

Which kind of tiles to use?

How to efficiently compute a tiling?

How to fill the tiles with a signal?

11

Overview

- Wang tiles and corner tiles
- Tiling algorithms
- Tile-based methods
  - for generating Poisson disk distributions
  - for texture synthesis
- Applications

12

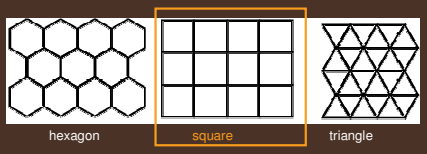
Overview

- Wang tiles and corner tiles
- Tiling algorithms
- Tile-based methods
  - for generating Poisson disk distributions
  - for texture synthesis
- Applications

13

Wang Tiles

- Regular tessellations
  - Regular tiling using a single regular polygonal tile



hexagon square triangle

14

Wang Tiles

- Wang tiles
  - Unit square tiles
  - Colored edges
  - Fixed orientation

the complete set of Wang tiles over 2 colors

15

Wang Tiles

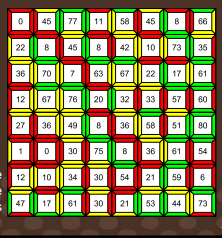
- Wang tile set
  - Finite set of Wang tiles
  - A complete set of Wang tiles over C colors counts C^4 tiles

the complete set of Wang tiles over 2 colors

16

Wang Tiles

- Tiling with Wang tiles
  - Using a Wang tile set
  - Adjoining edges must have matching colors
  - Generated using a tiling algorithm



a tiling with the complete Wang tile set over 3 colors

17

History of Wang Tiles

- Mathematics
  - Wang, 1961
    - Investigated the tiling problem; conjectured that aperiodic tile sets did not exist
  - Berger, 1966
    - Refuted the conjecture of Wang; Turing machines and the halting problem; Constructed the first aperiodic set of Wang tiles (20.426 tiles)
  - Berger, 1966 (104); Knuth, 1968 (92); Lauchli, 1966 (40); Robinson, 1967 (52); Penrose, 1977 (32), Robinson, 1977 (24), Kari, 1996 (14), Culik, 1996 (13)
  - Penrose, 1974
    - An aperiodic set of two tiles

18

History of Wang Tiles

- Computer graphics
  - Stam, 1997
    - Aperiodic texture
  - Hiller, Deussen & Keller, 2000
    - Poisson disk distributions
  - Cohen, Shade, Hiller & Deussen, 2003:
    - Wang tiles for image and texture generation
  - …

19

Corner Tiles

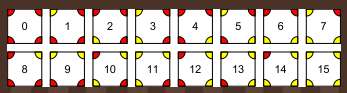
- Corner tiles
  - Unit square tiles
  - Colored corners
  - Fixed orientation

the complete set of corner tiles over 2 colors

20

Corner Tiles

- Corner tile set
  - Finite set of corner tiles
  - A complete set of corner tiles over C colors counts C^4 tiles

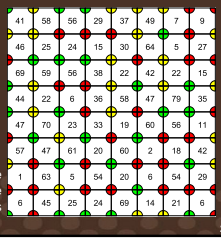


the complete set of corner tiles over 2 colors

21

Corner Tiles

- Tiling with corner tiles
  - Using a corner tiles set
  - Adjoining corners must have matching colors
  - Generated using a tiling algorithm

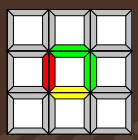


a tiling with the complete corner tile set over 3 colors

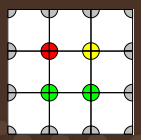
22

Wang Tiles vs. Corner Tiles

- *The corner problem*
  - Corner tiles constrain all their neighbors
  - Wang tiles don't



Wang tiles



corner tiles

23

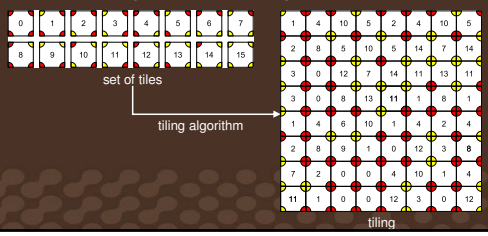
Overview

- Wang tiles and corner tiles
- Tiling algorithms
- Tile-based methods
  - for generating Poisson disk distributions
  - for texture synthesis
- Applications

24

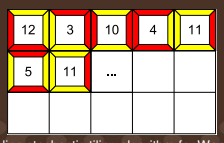
Tiling Algorithms

- Goal
  - Generating a stochastic tiling



Scanline Stochastic Tiling Algorithms

- For Wang tiles
  - Place tiles in scanline order (W→ E & N→S)
  - Choose a random tile with matching edge colors



a scanline stochastic tiling algorithm for Wang tiles

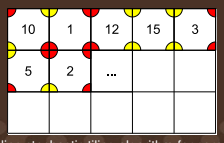
Scanline Stochastic Tiling Algorithms

- For Wang tiles
  - Place tiles in scanline order (W→ E & N→S)
  - Choose a random tile with matching edge colors
  - A compact set of Wang tiles over C colors counts $2C^2$ tiles

a compact Wang tile set over 2 colors

Scanline Stochastic Tiling Algorithms

- For corner tiles
  - Place tiles in scanline order (W→ E & N→S)
  - Choose a random tile with matching corner colors



a scanline stochastic tiling algorithm for corner tiles

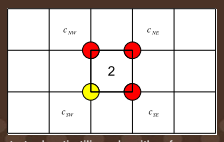
Direct Stochastic Tiling Algorithms

- *Which tile is at location (1000,1000)?*
- Scanline stochastic tiling algorithms
  - Compute the complete tiling at once

→ Direct stochastic tiling algorithms
  - Evaluate the stochastic tiling on the fly

Direct Stochastic Tiling Algorithms

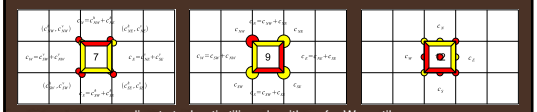
- For corner tiles
  - Based on a hash function that associates a random color with each integer lattice point



a direct stochastic tiling algorithm for corner tiles

Direct Stochastic Tiling Algorithms

- For Wang tiles
  - Slightly more complex
  - Transform from square lattice to diamond lattice



direct stochastic tiling algorithms for Wang tiles

31

Hash Functions

- 1D hash function
  - $hash(x) = P[x\%N]$
  - Permutation table P of size N
  - P contains a random permutation of {0, 1, …, N-1}

32

Hash Functions

- 2D hash function
  - $hash(x, y) = P_x[x\%N] + P_y[y\%N]$
  - $hash(x, y) = P[(P[x\%N] + y)\%N]$
  - $hash(x, y) = P[P_x[x\%N] \wedge P_y[y\%N]]$

33

Long-Period Hash Functions

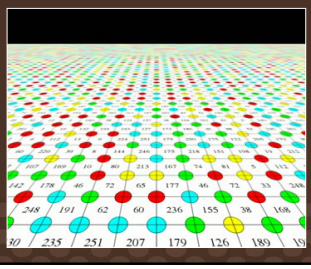
- Traditional hash functions
  - $hash(x) = P[x\%N]$
  - Period = N, size = N (256)
- Long-period hash functions
  - Combine multiple permutation tables (17,19,23,24,29,31,37)
  - Period = lcm (5.930.659.848), size = sum (180)

$$hash(x) = \left(\sum_{i=0}^{M-1} P_i[x\%N_i] \right) \%N$$

34

Long-Period Hash Functions

- Example



35

Overview

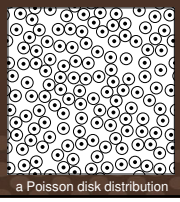
- Wang tiles and corner tiles
- Tiling algorithms
- Tile-based methods
  - for generating Poisson disk distributions
  - for texture synthesis
- Applications

36

Poisson Disk Distributions

- Definition
  - A 2D Poisson distribution in which all points are separated by a minimum distance 2r

minimum distance criterion

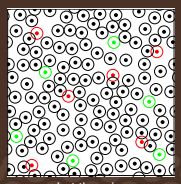


a Poisson disk distribution

37

Poisson Disk Distribution

- Generation
  - Dart throwing, relaxation dart throwing

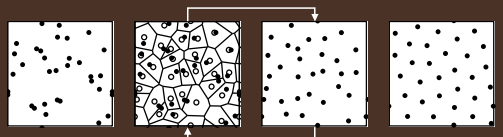


dart throwing

38

Poisson Disk Distribution

- Generation
  - Lloyd's relaxation scheme

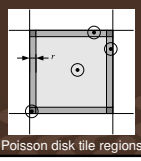


initial point distribution | Voronoi diagram centroids | improved point distribution | final point distribution

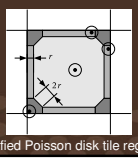
39

Corner-Based Poisson Disk Tiles

- Poisson disk tile regions
  - Determined by the Poisson disk radius r
- Modified Poisson disk tile regions
  - Slightly enlarge regions



Poisson disk tile regions

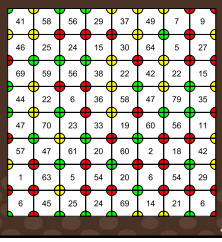


modified Poisson disk tile regions

40

Corner-Based Poisson Disk Tiles

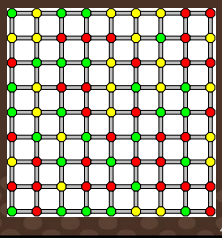
- Dual tiling
  - Combine the modified Poisson disk tile regions with a corner tile set



41

Corner-Based Poisson Disk Tiles

- Dual tiling
  - 3 kinds of tiles
    - C corner tiles
    - C^2 edge tiles
    - C^4 interior tiles



42

Corner-Based Poisson Disk Tiles

- Construct a Poisson disk distribution
  - Over the C corner tiles

43

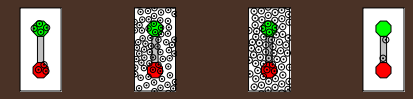
Corner-Based Poisson Disk Tiles

- Construct a Poisson disk distribution
  - Over the C corner tiles

44

Corner-Based Poisson Disk Tiles

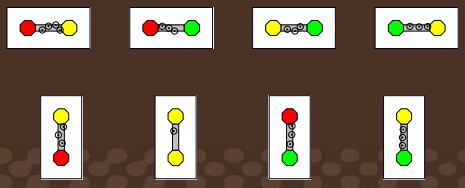
- Construct a Poisson disk distribution
  - Over the C^2 edge tiles



45

Corner-Based Poisson Disk Tiles

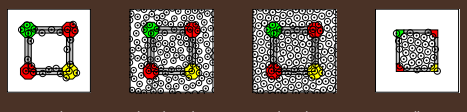
- Construct a Poisson disk distribution
  - Over the C^2 edge tiles



46

Corner-Based Poisson Disk Tiles

- Construct a Poisson disk distribution
  - Over the C^4 interior tiles



47

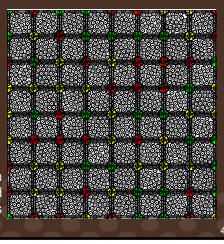
Corner-Based Poisson Disk Tiles

- Construct a Poisson disk distribution
  - Over the C^4 interior tiles

48

Corner-Based Poisson Disk Tiles

- Efficiently generating Poisson disk distributions
  - Construct corner-based Poisson disk tile set
  - Generate stochastic tiling

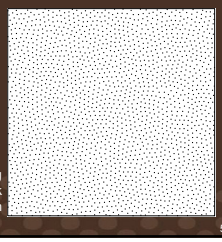


a tiling with corner-based Poisson disk tiles

49

Corner-Based Poisson Disk Tiles

- Efficiently generating Poisson disk distributions
  - Construct corner-based Poisson disk tile set
  - Generate stochastic tiling

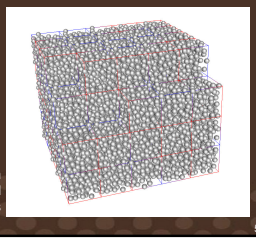


the resulting Poisson disk distribution

50

Corner-Based Poisson Sphere Tiles

- Tile-based method for Poisson sphere distributions



a tiling with corner-based Poisson sphere tiles

51

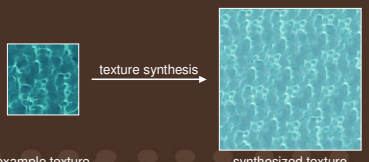
Overview

- Wang tiles and corner tiles
- Tiling algorithms
- Tile-based methods
  - for generating Poisson disk distributions
  - for texture synthesis
- Applications

52

Texture Synthesis

- Create from an example texture a larger similar texture



example texture

synthesized texture

53

Tile-Based Texture Synthesis

- Corner-based texture tiles
  - Tile-based method for efficiently synthesizing textures

original signal

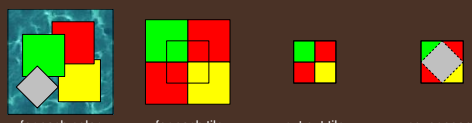
set of tiles

generated signal

54

Tile-Based Texture Synthesis

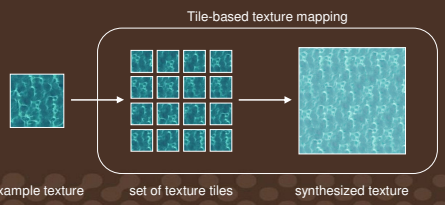
- Corner-based texture tiles



55

Tile-Based Texture Synthesis

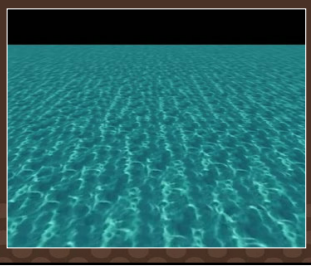
- Tile-based texture mapping



56

Tile-Based Texture Mapping

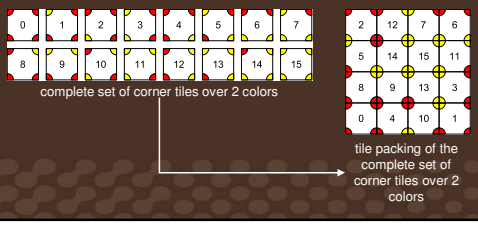
- Example



57

Tile-Based Texture Mapping

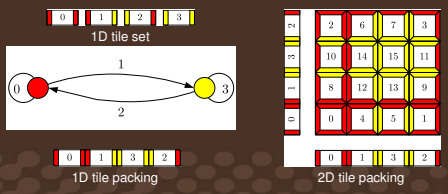
- Tile packing
  - Avoids unwanted artifacts



58

Tile-Based Texture Mapping

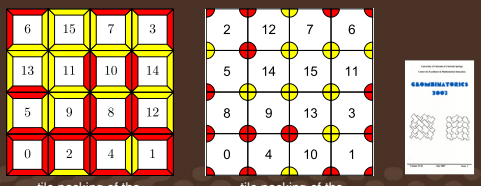
- Tile packing
  - Constructive method for Wang tiles



59

Tile-Based Texture Mapping

- Tile packing
  - Search method for corner tiles



60

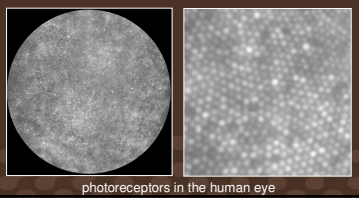
Overview

- Wang tiles and corner tiles
- Tiling algorithms
- Tile-based methods
  - for generating Poisson disk distributions
  - for texture synthesis
- Applications

61

Sampling

- The Poisson disk distribution is one of the best sampling distributions
  - Photoreceptors in the eye

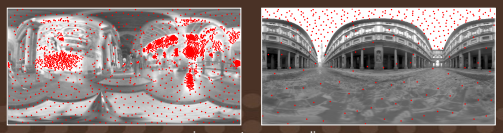


photoreceptors in the human eye

62

Sampling

- The Poisson disk distribution is one of the best sampling distributions
  - Photoreceptors in the eye, theoretical evidence
  - Primary rays, environment map sampling, ...



environment map sampling

63

Non-Photorealistic Rendering

- The Poisson disk distribution is one of the best primitive distributions
  - Pen-and-ink illustrations, stippling, halftoning, …

original image | stippled | hatched

64

Procedural Modeling

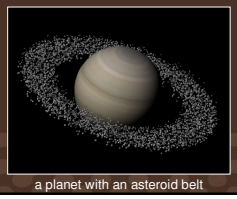
- Many distributions follow a pattern with a minimum-distance criterion
  - Trees in a forest, hairs in fur, people in a crowd, …

a beech forest in the winter

65

Procedural Modeling

- Many distributions follow a pattern with a minimum-distance criterion
  - Trees in a forest, hairs in fur, people in a crowd, …



a planet with an asteroid belt

66

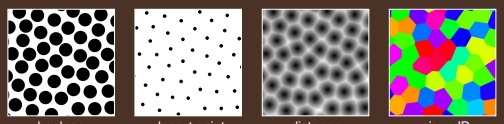
Procedural Texturing

- A 2D procedural object distribution function
  - Enables the procedural generation of a new class of textures

67

Procedural Texturing

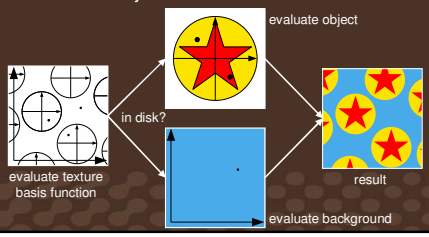
- A 2D procedural object distribution function
  - Evaluation of the texture basis function



68

Procedural Texturing

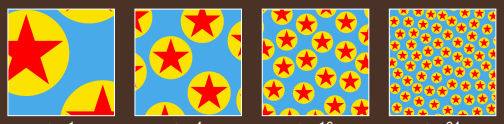
- A 2D procedural object distribution function
  - Procedural object distribution



69

Procedural Texturing

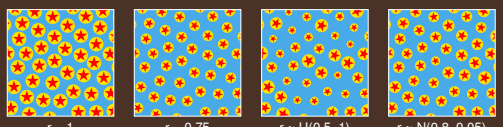
- A 2D procedural object distribution function
  - Parameters: scale s



70

Procedural Texturing

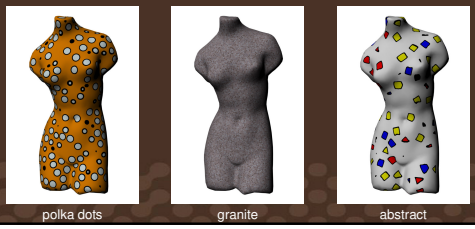
- A 2D procedural object distribution function
  - Parameters: size r, orientation θ



71

Procedural Texturing

- A 3D procedural object distribution function
  - Examples

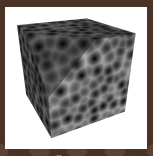


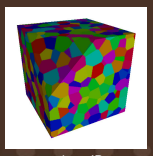
72

Procedural Texturing

- A 3D procedural object distribution function
  - Evaluation of the texture basis function


boolean


distance


unique ID

73

Overview

- Wang tiles and corner tiles
- Tiling algorithms
- Tile-based methods
  - for generating Poisson disk distributions
  - for texture synthesis
- Applications

74

Conclusion

- Tile-based methods for interactive applications

Which kind of tiles to use?
→ Corner tiles

How to efficiently compute a tiling?
→ Direct stochastic tiling algorithm

How to fill the tiles with a signal?
→ Two examples, general method

75

Thanks

- Questions?
- More information
  - Class notes
  - `http://www.cs.kuleuven.be/~ares/`

Acknowledgments

Travel fund of the Research Foundation – Flanders (FWO). Travel fund of ACM SIGGRAPH 2008.
Philip Dutré. Ph. D. fellowship of the Research Foundation - Flanders (FWO).
Ares Lagae is a Postdoctoral Fellow of the Research Foundation Flanders (FWO).

76

Chapter 2

Syllabus

Preface

These class notes are based on my dissertation *Tile-Based Methods in Computer Graphics*. The electronic version of my dissertation is available at `http://www.cs.kuleuven.be/~ares`.

Ares Lagae
Heverlee, May 2008

Abstract

Many complex signals, such as point distributions and textures, cannot efficiently be synthesized and stored. In this work we present tile-based methods to solve this problem. Instead of synthesizing a complex signal when needed, the signal is synthesized on forehand over a small set of tiles. Arbitrary large amounts of that signal can then efficiently be generated when needed by generating a stochastic tiling.

Tile-based methods are traditionally based on Wang tiles. The colored edges of Wang tiles only constrain the four direct neighboring tiles, but not the four diagonally neighboring tiles. This problem introduces unwanted artifacts in the tiled signals, and complicates methods for synthesizing signals over a set of Wang tiles. To solve this problem we present corner tiles. Corner tiles are unit square tiles with colored corners rather than colored edges. The colored corners of corner tiles constrain all neighboring tiles. We revisit the most important applications of Wang tiles, and we show that corner tiles have substantial advantages for each of these applications.

Stochastic tilings are traditionally generated using scanline stochastic tiling algorithms. However, these algorithms store the complete tiling and are therefore not efficient. To solve this problem, we present direct stochastic tiling algorithms for Wang tiles and corner tiles. These algorithms are capable of evaluating a stochastic tiling locally, without explicitely constructing and storing the tiling up to that point. We also introduce long-period hash functions to generate very large tilings.

Poisson disk distributions and textures are two examples of complex signals. We present tile-based methods for generating Poisson disk distributions and for synthesizing textures. Tile-based methods not only allow to efficiently generate Poisson disk distributions and synthesize textures, but also enable new applications such as tile-based texture synthesis and a procedural object distribution function. This new texture basis function allows to distribute procedural objects over a procedural background, using intuitive parameters such as the scale, size and orientation of the objects. We also present an overview of applications of tiled Poisson disk distributions.

The tile-based methods we present in this work are a valuable tool for computer graphics, and help to keep up with the continuously increasing demand for more complexity and realism in digitally synthesized images.

Contents

Chapter 1

Introduction

1.1 Tile-Based Methods in Computer Graphics

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 synthesis and storage of complex signals, such as point distributions or textures. For several of these complex signals, no efficient synthesis algorithms are available, and storing large quantities of these signals is expensive. Tile-based methods 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 work 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 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 consistently with the continuity constraints imposed by the colored edges. 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 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.

This work introduces 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 work introduces efficient tiling algorithms for generating stochastic tilings using Wang tiles and corner tiles, and methods for constructing Poisson disk distributions and synthesizing textures over a set of Wang tiles and corner tiles. Although the methods for constructing a complex signal over a set of tiles are dependent on the signal, the general idea behind the methods presented in this work should generalize to other kinds of signals.

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. However, no efficient algorithms are available for generating Poisson disk distributions. Constructing a Poisson disk distribution over a set of Wang tiles or corner tiles is challenging, because the minimum distance criterion should be respected over tile boundaries. This work also includes an overview of applications of tiled Poisson disk distributions, and a detailed comparison of several methods for generating Poisson disk distributions.

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.

The tile-based methods presented in this work enable efficient generation of Poisson disk distributions and rapid synthesis of textures, but also enable new applications, such as a procedural object distribution in the case of Poisson disk distributions, and tile-based texture mapping in the case of texture synthesis.

Corner tiles are also investigated in the context of the tiling problem and aperiodic tile sets, which is the original context of Wang tiles. Several new aperiodic sets of Wang tiles and corner tiles are introduced in this work.

The methods introduced in this work help to manage the continuous demand for more

quality and complexity in digitally synthesized images, and are a valuable tool for computer graphics.

1.2 Overview

This work is organized as follows.

Chapter 2 introduces tilings, Wang tiles and corner tiles, discusses previous applications of tilings in computer graphics, and introduces several useful definitions, conventions and notations.

Chapter 3 presents efficient algorithms for generating stochastic tilings with Wang tiles and corner tiles. This chapter presents scanline stochastic tiling algorithms and direct stochastic tiling algorithms for Wang tiles and corner tiles, and long-period hash functions defined over the integer lattice, used in direct stochastic tiling algorithms.

Chapter 4 introduces Poisson disk distributions and presents several tile-based methods for generating Poisson disk distributions.

Chapter 5 introduces tile-based methods for texture mapping and texture synthesis. This chapter shows how to synthesize a texture over a set of Wang tiles and corner tiles, presents an efficient tile-based texture mapping algorithm running on the GPU, and discusses the tile packing problem.

Chapter 6 discusses several applications of Poisson disk distributions. These applications include sampling, non-photorealistic rendering, scientific visualization, procedural modeling, and procedural texturing.

Chapter 2

Wang Tiles and Corner Tiles

2.1 Introduction

The tile-based methods presented in this work are based on Wang tiles and corner tiles. In this chapter we introduce Wang tiles and corner tiles. We briefly sketch their history, and introduce key concepts that will be used in later chapters.

Overview

This chapter is organized as follows. Section 2.2 introduces tilings. In section 2.3 we discuss previous applications of tilings in computer graphics. Section 2.4 introduces Wang tiles and briefly sketches their background. In section 2.5 we discuss applications of Wang tiles in computer graphics. Section 2.6 explains the corner problem and introduces corner tiles. In section 2.7 we introduce definitions, conventions and notations. Section 2.8 proposes a convenient scheme for enumerating Wang tile sets and corner tile sets. In section 2.9 we explain the close relationship between Wang tiles and corner tiles. Section 2.10 discusses generalizations of Wang tiles and corner tiles to arbitrary dimensions. In section 2.11 we conclude.

2.2 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.

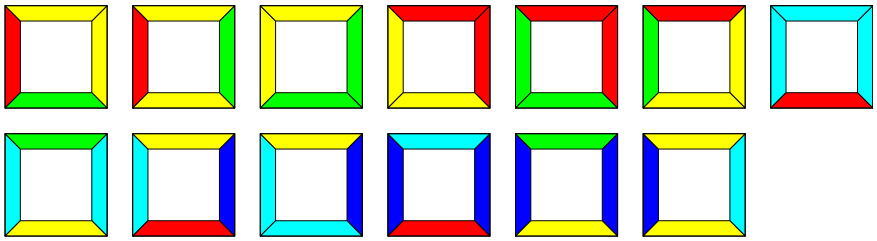


Figure 2.1: The smallest aperiodic Wang tile set currently known.

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, chapter 12].

2.3 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. Hausner [2001] presented a system for generating decorative tile mosaics. Ostromoukhov et al. [2004] used a hierarchically subdivided Penrose tiling to generate well-distributed point sets.

2.4 Wang Tiles

The tiles we focus on in this work 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 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 20426 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.

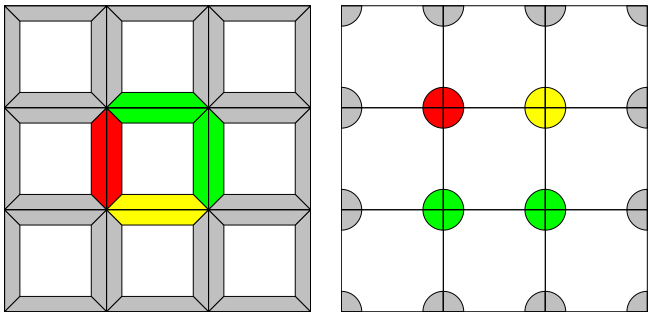
2.5 Wang Tiles in Computer Graphics

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. The latter approach was later adopted by Cohen et al. [2003], in a paper that popularized Wang tiles in the field of computer graphics. The same paper introduced a method for texture synthesis using Wang tiles. Wei [2004] proposed tile-based texture mapping on graphics hardware. Fu and Leung [2005] recently extended texture tiling to surfaces with arbitrary topology.

2.6 Corner Tiles and the Corner Problem

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.



(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.

In order to solve the corner problem, we proposed corner tiles [Lagae and Dutré, 2006a], 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.

To our knowledge, tiles with colored corners have not been used previously in computer graphics (or in other domains), except by Ng et al. [2005], who presented a technique for assembling a set of tiles similar to corner tiles from an input texture to synthesize larger textures. Their technique is discussed in detail in section 5.3. Neyret and Cani [1999] use triangular tiles with edge and corner colors to generate pattern-based textures over a triangle mesh, in the spirit of Stam [1997].

2.7 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

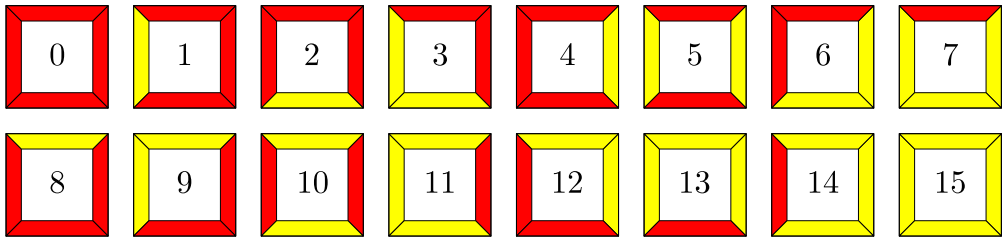


Figure 2.3: The complete Wang tile set over 2 colors.

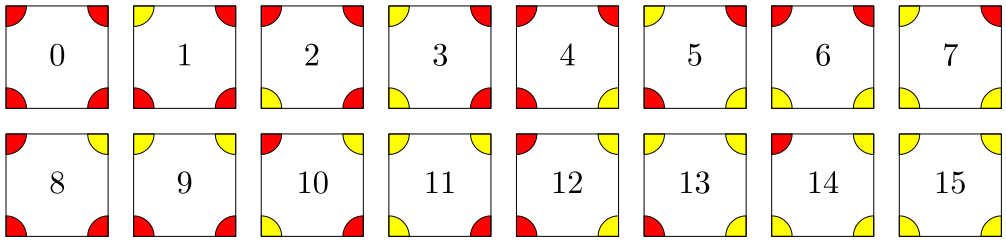


Figure 2.4: The complete corner tile set over 2 colors.

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 work 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.

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

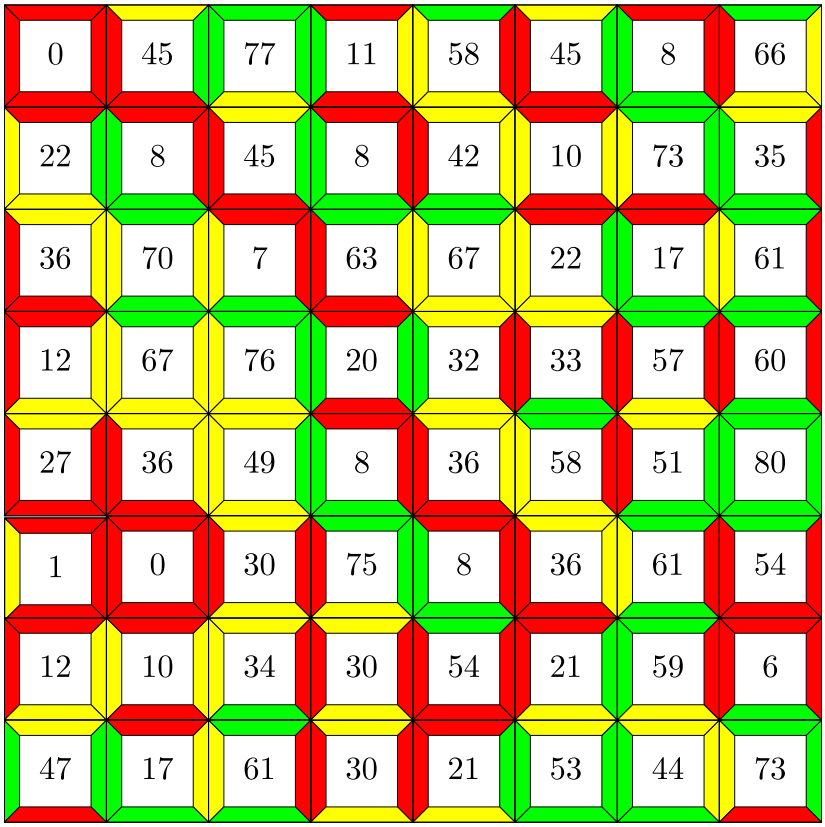


Figure 2.5: A tiling with the complete Wang tile set over 3 colors.

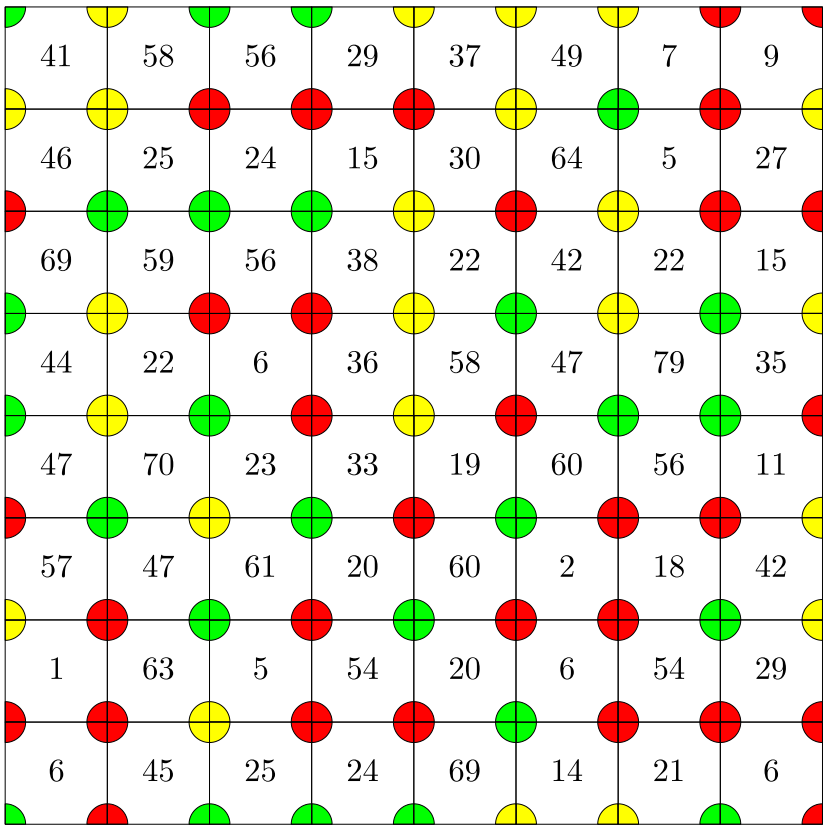


Figure 2.6: A tiling with the complete corner tile set over 3 colors.

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 work, 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.

2.8 Enumerating Wang Tile Sets and Corner Tile Sets

For efficiently manipulating Wang tiles and corner tiles, an enumeration of the tiles is needed. In this work we use the following scheme.

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 $0, 1, \ldots, C^4 - 1$. A base conversion switches between the corner colors and the tile index.

The tile index i of the Wang tile with edge colors c_N, c_E, c_S and c_W is given by

$$i = c_N C^3 + c_E C^2 + c_S C + c_W, \tag{2.1}$$

or, after factoring out powers of C using Horner's rule, by

$$i = ((c_N C + c_E)C + c_S)C + c_W. \tag{2.2}$$

This conversion of base only requires three integer multiplications and three integer additions.

The edge colors c_N, c_E, c_S and c_W of the Wang tile with tile index i are given by

$$\begin{aligned} c_N &= (i/C^3) \,\%\, C \\ c_E &= (i/C^2) \,\%\, C \\ c_S &= (i/C) \,\%\, C \\ c_W &= i \,\%\, C \end{aligned} \tag{2.3}$$

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

$$\begin{aligned} c_W &\leftarrow i \,\%\, C \\ i &\leftarrow i/C \\ c_S &\leftarrow i \,\%\, C \\ i &\leftarrow i/C \\ c_E &\leftarrow i \,\%\, C \\ i &\leftarrow i/C \\ c_N &\leftarrow i \end{aligned} \tag{2.4}$$

where $\leftarrow$ indicates assignment.

For corner tiles, we use a similar scheme. 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

$$i = ((c_{NE}C + c_{SE})C + c_{SW})C + c_{NW}. \tag{2.5}$$

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

$$\begin{aligned} c_{NE} &= (i/C^3)\,\%\,C \\ c_{SE} &= (i/C^2)\,\%\,C \\ c_{SW} &= (i/C)\,\%\,C \\ c_{NW} &= i\,\%\,C \end{aligned} \tag{2.6}$$

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

$$\begin{aligned} c_{NW} &\leftarrow i\,\%\,C \\ i &\leftarrow i/C \\ c_{SW} &\leftarrow i\,\%\,C \\ i &\leftarrow i/C \\ c_{SE} &\leftarrow i\,\%\,C \\ i &\leftarrow i/C \\ c_{NE} &\leftarrow i \end{aligned} \tag{2.7}$$

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.

2.9 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 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 a Wang tile set of 16 tiles over 4 colors.

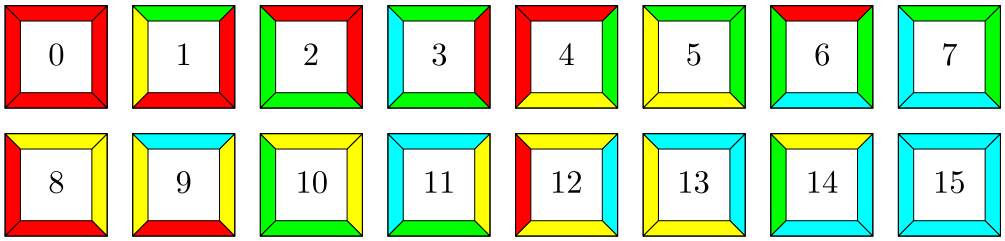


Figure 2.7: The Wang tile set equivalent to the corner tile set over 2 colors.

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.

2.10 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]. We will use dominoes in section 5.5 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 Ebert [2005] used Wang cubes for example-based volume illustrations. We will use corner cubes in section 4.4 to generate Poisson sphere distributions.

2.11 Conclusion

In this chapter we have introduced Wang tiles and corner tiles. We have discussed the history of Wang tiles and corner tiles, and we have introduced important concepts that will be used in later chapters.

Chapter 3

Tiling Algorithms for Wang Tiles and Corner Tiles

3.1 Introduction

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.

Overview

This chapter is organized as follows. In section 3.2 we discuss scanline stochastic tiling algorithms, and in section 3.3 we discuss direct stochastic tiling algorithms, two classes of stochastic tiling algorithms. Section 3.4 discusses hash functions defined over the integer lattice, an essential ingredient of direct stochastic tiling algorithms. In section 3.5 we conclude.

3.2 Scanline Stochastic Tiling Algorithms

Scanline stochastic tiling algorithms are stochastic tiling algorithms that generate a tiling by placing tiles in scanline order. In this section, we discuss a scanline tiling algorithm for Wang tiles and a scanline stochastic tiling algorithm for corner tiles.

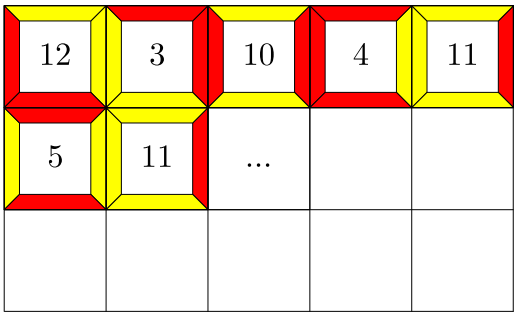


Figure 3.1: A scanline stochastic tiling algorithm for Wang tiles.

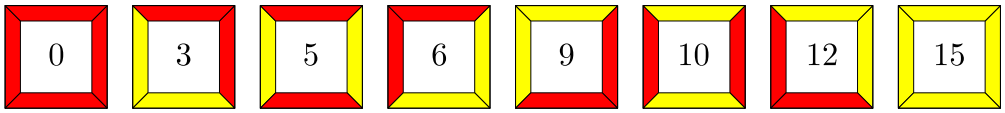


Figure 3.2: A compact Wang tile set over 2 colors.

3.2.1 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.

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.

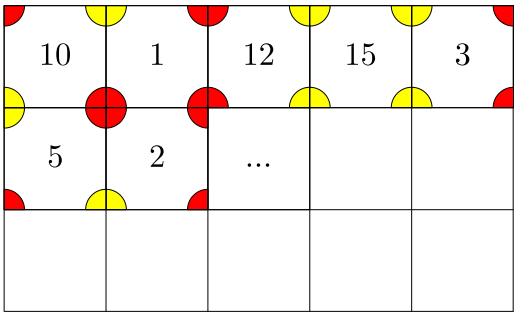


Figure 3.3: A scanline stochastic tiling algorithm for corner tiles.

3.2.2 A Scanline Stochastic Tiling Algorithm for Corner Tiles

In 2006, we presented a scanline stochastic tiling algorithm for corner tiles [Lagae and Dutré, 2006a]. 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 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.

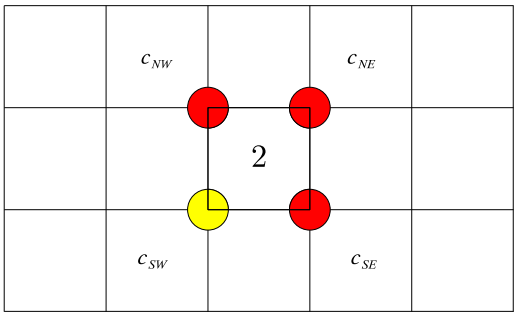


Figure 3.4: A direct stochastic tiling algorithm for corner tiles.

3.3 Direct Stochastic Tiling Algorithms

Several applications, such as tile-based texture mapping (see section 5.4), and the procedural object distribution texture basis functions (see section 6.6), 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. To address this problem, we propose direct stochastic tiling algorithms. 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.

The direct stochastic tiling algorithms we present 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 we use are efficient in time and space, and the transformation can be performed locally. This enables efficient direct stochastic tiling algorithms.

In this section, we discuss several direct stochastic tiling algorithms for Wang tiles and a direct stochastic tiling algorithm for corner tiles. For clarity, we will first discuss the direct stochastic tiling algorithm for corner tiles. The hash functions used in the direct stochastic tiling algorithms are discussed in section 3.4

3.3.1 A Direct Stochastic Tiling Algorithm for Corner Tiles

In 2006, we proposed a direct stochastic tiling algorithm for corner tiles [Lagae and Dutré, 2006a].

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

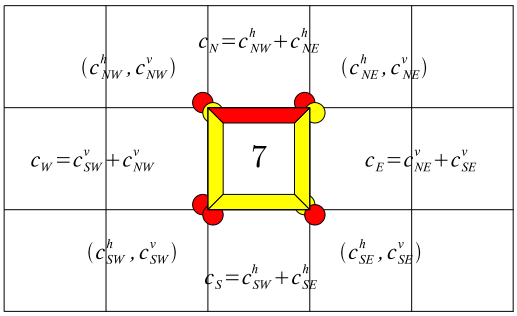


Figure 3.5: A direct stochastic tiling algorithm for Wang tiles using two hash functions. The first hash function is used for the horizontal edges, and the second hash function is used for the vertical edges.

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 2.5. The direct stochastic tiling algorithm for corner tiles is illustrated in figure 3.4.

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.

3.3.2 Direct Stochastic Tiling Algorithms for Wang Tiles

In this subsection, we discuss 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.

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

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

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

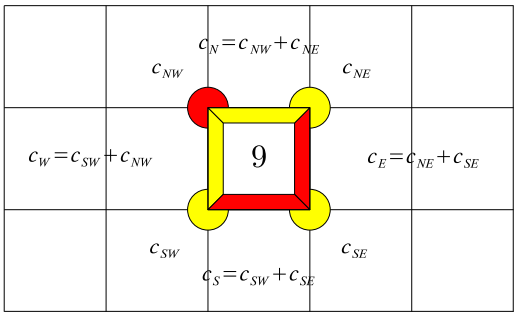


Figure 3.6: A direct stochastic tiling algorithm for compact sets of Wang tiles.

$(h_h(x,y), h_v(x,y)) \in \{0,1,\ldots,C-1\}^2$ 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 $\left(c^h_{NE}, c^v_{NE}\right)$, $\left(c^h_{SE}, c^v_{SE}\right)$, $\left(c^h_{SW}, c^v_{SW}\right)$ and $\left(c^h_{NW}, c^v_{NW}\right)$, 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 using equation 2.2. The direct stochastic tiling algorithm for Wang tiles using two hash functions is illustrated in figure 3.5.

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].

3.3.2.2 A Direct Stochastic Tiling Algorithm for Compact Sets of Wang Tiles

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

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

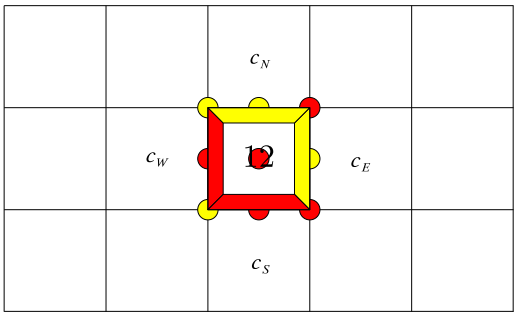


Figure 3.7: A direct stochastic tiling algorithm for Wang tiles using a hash function defined at the tile edges.

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.

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.

3.3.2.3 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 using equation 2.2. 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 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.

3.4 Hash Functions

Hash functions defined over the integer lattice are an essential ingredient of the direct stochastic tiling algorithms discussed in section 3.3. Hash functions are also used extensively in procedural modeling and texturing [Perlin, 1985; Ebert et al., 2002]. In this section, we discuss traditional hash functions based on permutation tables and we propose long-period hash functions based on permutation tables.

3.4.1 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 ith 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 we also use them in our direct stochastic tiling algorithms. The hash functions are easy to implement and efficient to evaluate.

3.4.2 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, we proposed long-period hash functions based on permutation tables [Lagae and Dutré, 2006b]. 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.

In this subsection we define long-period hash functions, we study the period and range, distribution and efficiency of long-period hash functions, and we formulate guidelines for designing long-period hash functions.

3.4.2.1 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.

3.4.2.2 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. *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.

3.4.2.3 Efficiency

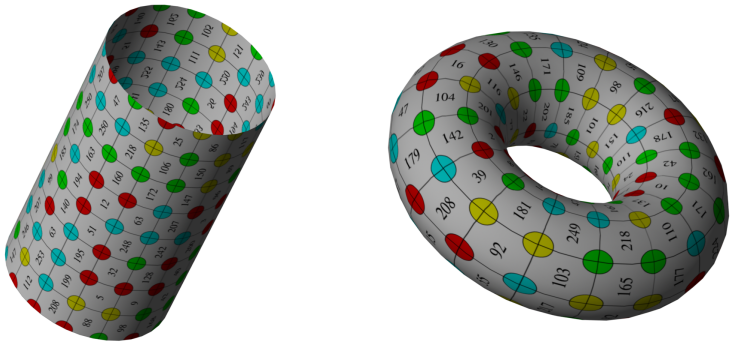
The time needed to evaluate a combined hash function is roughly proportional to the number of combining hash functions. This does not mean that an application that uses a long-period hash function consisting of N combining hash functions will be N times slower than the same application using a traditional 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.

3.4.2.4 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. We recommend the strategy outlined below 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



(a) (b)

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

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.

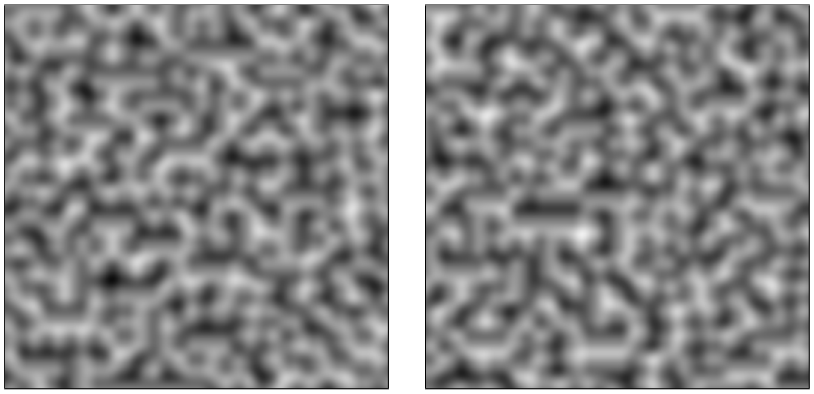
The number of permutation tables will determine the time needed to evaluate the hash function, and the joint size of the permutation tables will determine the memory footprint of the hash function. Period length can be traded for evaluation time and memory footprint.

3.4.3 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



(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.

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.

3.4.4 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.

3.5 Conclusion

Tile-based methods in computer graphics require efficient algorithms for generating stochastic tilings. In this chapter we have presented scanline stochastic tiling algorithms and direct stochastic tiling algorithms for Wang tiles and corner tiles. We have also studied the hash functions on which the direct stochastic tiling algorithms are based.

Direct stochastic tiling algorithms are more elegant and at least as efficient as scanline stochastic tiling algorithms. We therefore recommend to use direct stochastic tiling algorithms if possible, even for applications that do not require local evaluation of the tiling.

Stochastic tiling algorithms for corner tiles are usually more efficient and more elegant than stochastic tiling algorithms for Wang tiles. We therefore recommend to use corner tiles if possible.

One stochastic tiling algorithm that is still missing in this chapter is a direct stochastic tiling algorithm for compact sets of corner tiles. Although such an algorithm would be useful, we have not yet succeeded in developing one.

Direct stochastic tiling algorithms are based on hash functions. The period of the tiling is inherited from the period of the hash functions. Hash functions based on permutation tables allow to trade storage space and evaluation time for period length, and allow to construct hash functions with long periods. These hash functions are not only useful in tiling algorithms but also in procedural modeling and texturing techniques.

Chapter 4

Tile-Based Methods for Generating Poisson Disk Distributions

4.1 Introduction

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 several methods for constructing Poisson disk distributions over a set of Wang tiles or 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.

Overview

This chapter is organized as follows. In section 4.2 we introduce Poisson disk distributions. Section 4.3 presents corner-based Poisson disk tiles, a tile-based methods for generating Poisson disk distributions. In section 4.4 we investigate Poisson sphere distributions, the three-dimensional equivalent of Poisson disk distributions. In section 4.5 we conclude.

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.

4.2 Poisson Disk Distributions

In this section we define Poisson disk distributions, we sketch the history and background of Poisson disk distributions, we propose an intuitive radius specification scheme for Poisson disk distributions, and we introduce methods for generating Poisson disk distributions.

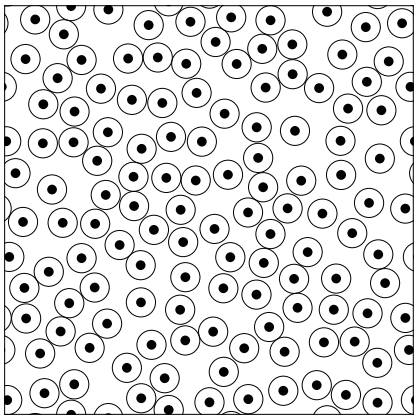


Figure 4.1: A Poisson disk distribution.

4.2.1 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 4.1 shows an example of a Poisson disk distribution.

4.2.2 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. The first tile based method, an extension of the dart throwing algorithm, was presented by Shade et al. [2000]. Hiller et al. [2001] used Lloyd's relaxation algorithm 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] presented an interesting technique to generate a distribution with blue noise properties over a given density, based on the Penrose tiles and Lloyd's relaxation method. Kopf et al. [2006] presented a method to generate Poisson disk distributions over a given density in real time, based on recursive Wang tiles that contain self-similar and progressive Poisson distributions.

Recently, Jones [2006] and Dunbar and Humphreys [2006] presented efficient implementations of the dart throwing algorithm.

Tools to analyze the spectral properties of point sets were introduced by Ulichney [1987], in the context of dithering.

4.2.3 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, we proposed a more intuitive radius specification scheme [Lagae and Dutré, 2005]. Instead of using the absolute radius r, the radius is expressed as a relative radius ρ. The relative radius ρ 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 η of a hexagonal lattice is [Steinhaus, 1999]

$$\eta = \frac{\pi}{2\sqrt{3}} \approx 0.9069. \tag{4.1}$$

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 η/N. The maximum possible disk radius r_{max} of this Poisson disk distribution is thus given by

$$r_{max} = \sqrt{\frac{1}{2\sqrt{3}N}}. \tag{4.2}$$

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

$$r = \rho\, r_{max}, \tag{4.3}$$

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.

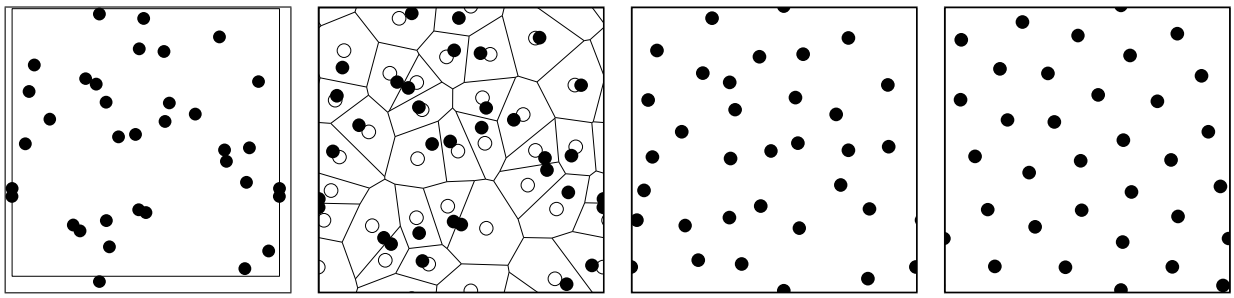
4.2.4 Generation

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

4.2.4.1 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



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

Figure 4.2: 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.

the distribution is difficult to predict, and if the process is stopped too soon, the density of the points is not uniform.

4.2.4.2 Relaxation Dart Throwing

McCool and Fiume [1992] proposed an improved version of the dart throwing algorithm, which we call 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.

4.2.4.3 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 4.2.

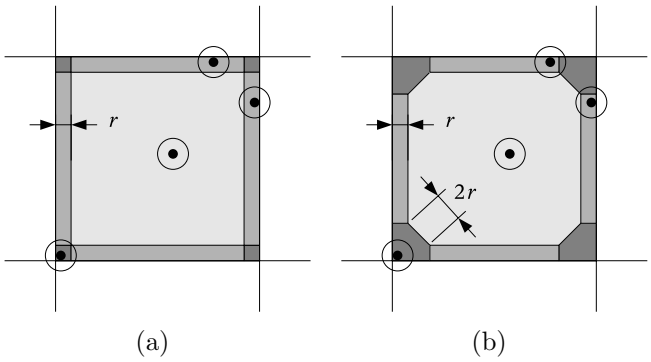


Figure 4.3: The Poisson disk tile regions and the modified Poisson disk tile regions. (a) The Poisson disk radius determines corner regions, edge regions and an interior region. (b) The corner regions are modified such that the distance between regions of the same type is at least $2r$.

4.3 Corner-Based Poisson Disk Tiles

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

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 4.3(a).

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 4.3(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 4.4. This tiling uses three different kinds of tiles. Corner tiles, horizontal and vertical edge tiles, and interior

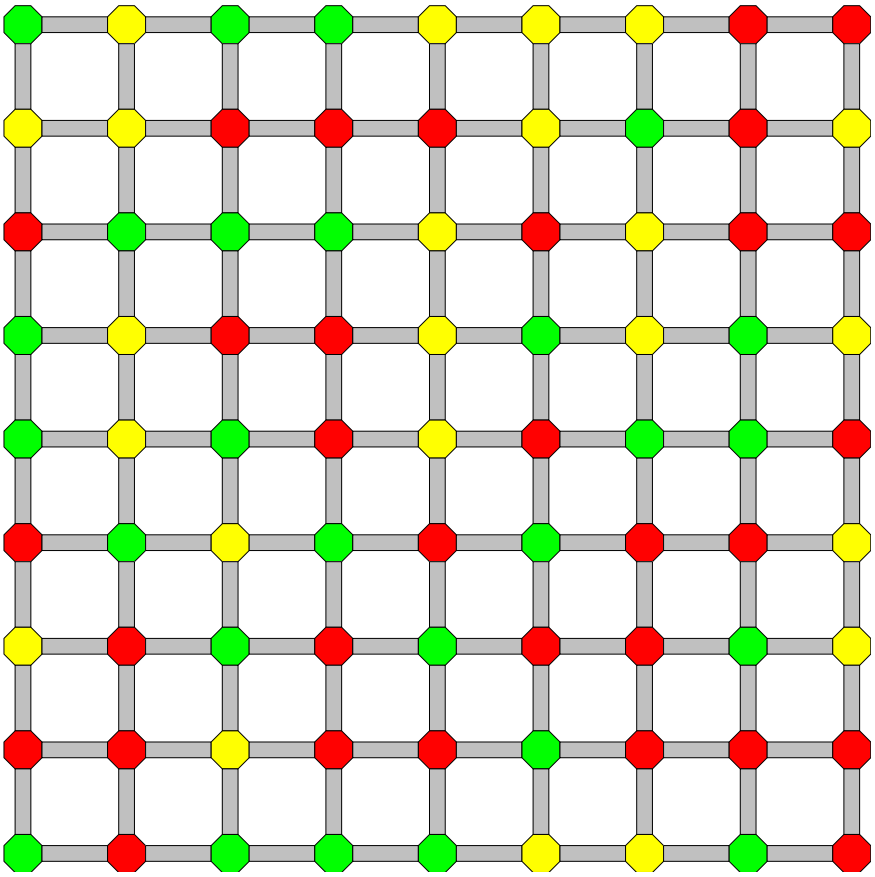


Figure 4.4: A tiling obtained by combining the modified Poisson disk tile regions with the complete corner tile set over 3 colors. This tiling was generated from the tiling shown in figure 2.6.

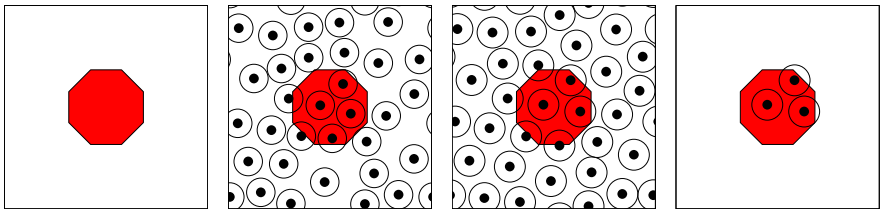


Figure 4.5: Construction of a Poisson disk distribution over a corner tile of a corner-based Poisson disk tile set. (a) The corner tile. (b) A toroidal Poisson disk distribution is generated. (c) The Poisson disk distribution is optimized using Lloyd's relaxation scheme. (d) The corner tile is cut out of the Poisson disk distribution.

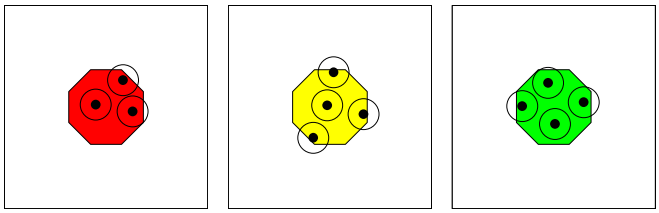


Figure 4.6: Poisson disk distributions constructed over corner tiles of a corner-based Poisson disk tile set.

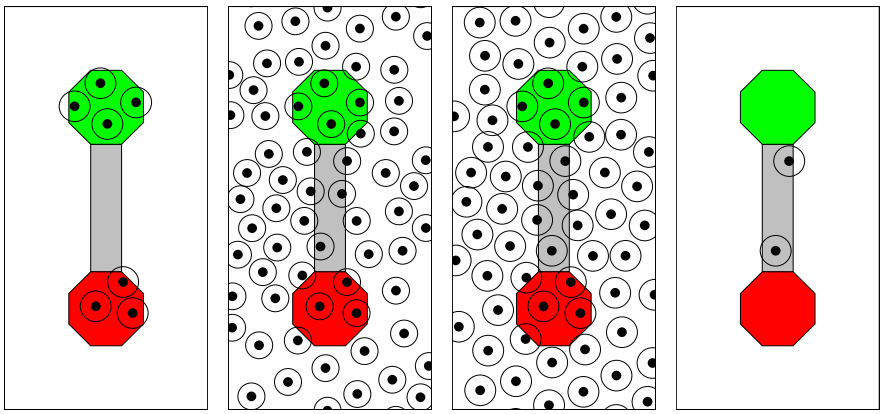


Figure 4.7: Construction of a Poisson disk distribution over a vertical edge tile of a corner-based Poisson disk tile set. (a) The edge tile is assembled with the corresponding corner tiles. (b) A toroidal Poisson disk distribution is generated. (c) The Poisson disk distribution is optimized using Lloyd's relaxation scheme. (d) The edge tile is cut out of the Poisson disk distribution.

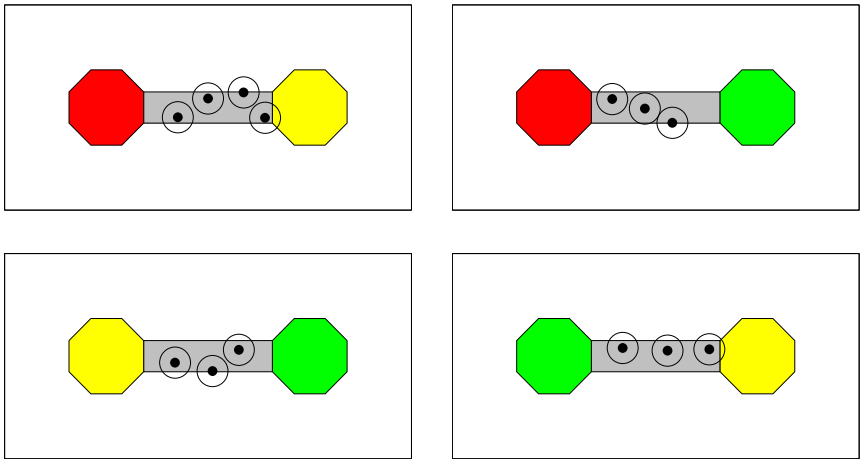


Figure 4.8: Poisson disk distributions constructed over horizontal edge tiles of a corner-based Poisson disk tile set.

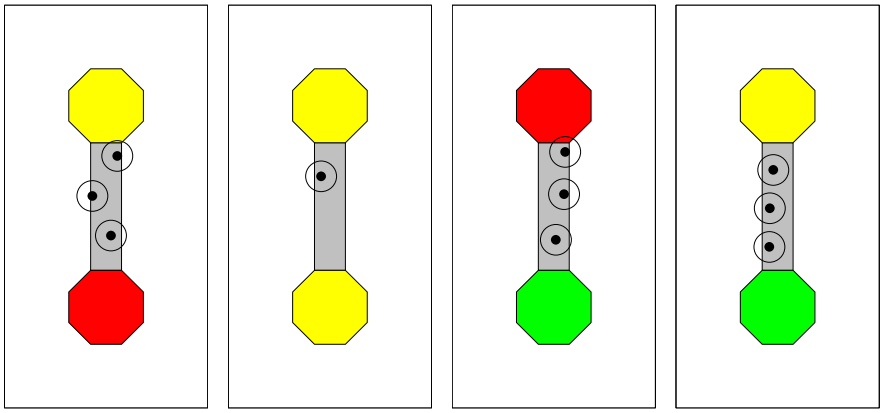


Figure 4.9: Poisson disk distributions constructed over vertical edge tiles of a corner-based Poisson disk tile set.

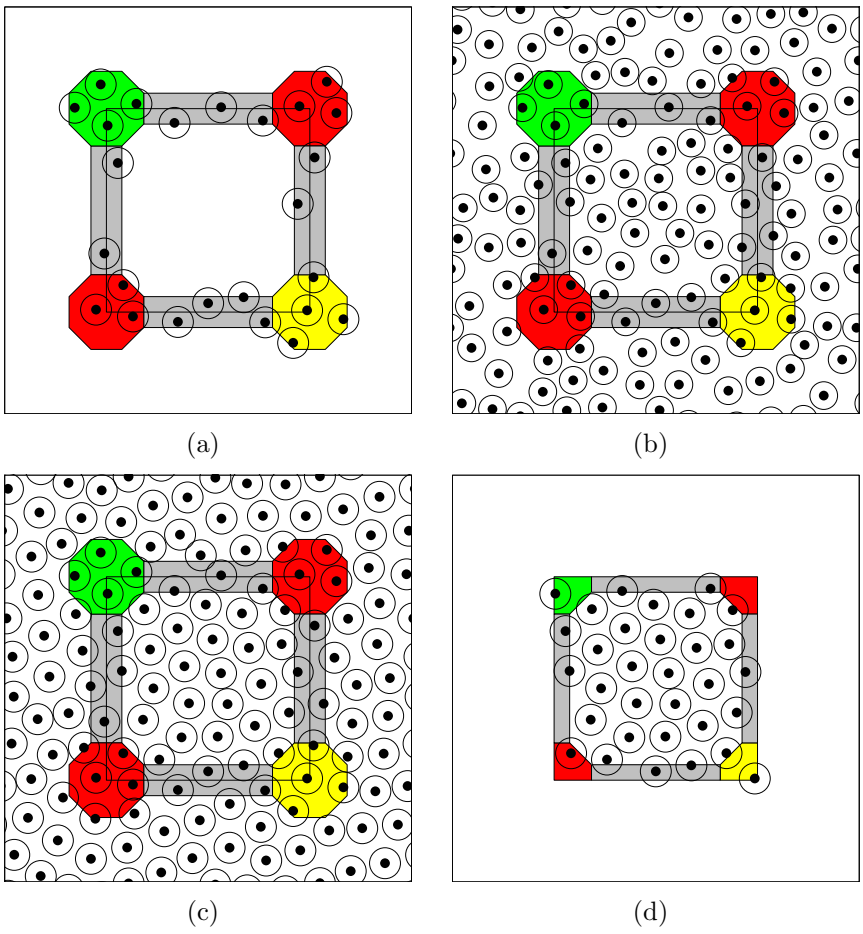


Figure 4.10: Construction of a Poisson disk distribution over a tile of a corner-based Poisson disk tile set. (a) The interior tile is assembled with the corresponding corner tiles and edge tiles. (b) A toroidal Poisson disk distribution is generated. (c) The Poisson disk distribution is optimized using Lloyd's relaxation scheme. (d) The tile is cut out of the Poisson disk distribution.

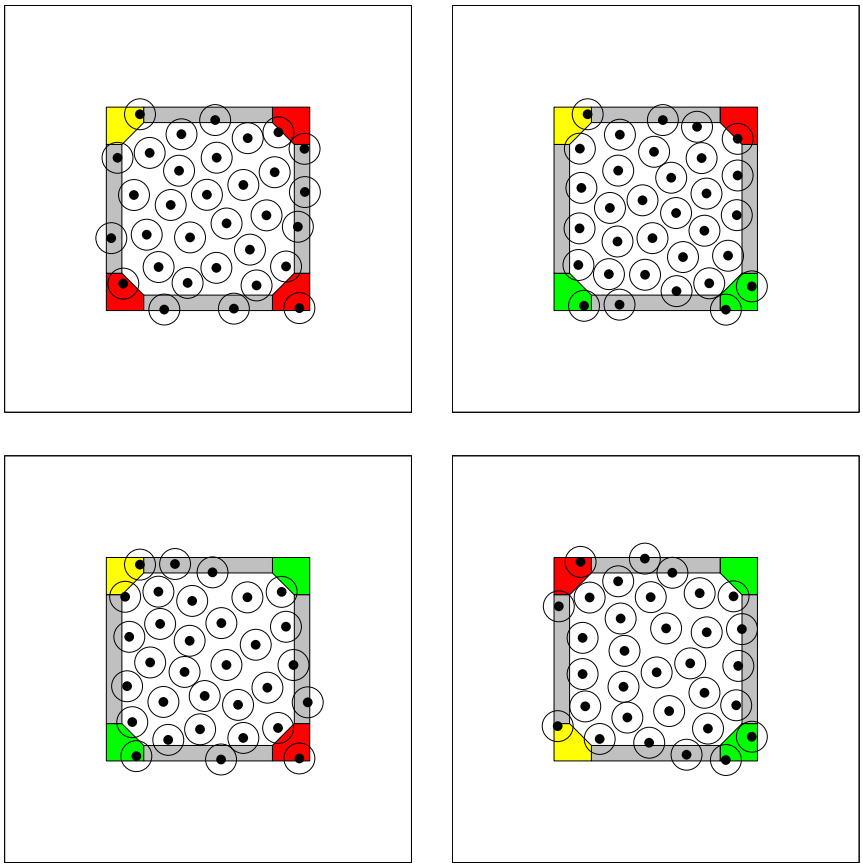


Figure 4.11: Poisson disk distributions constructed over tiles of a corner-based Poisson disk tile set.

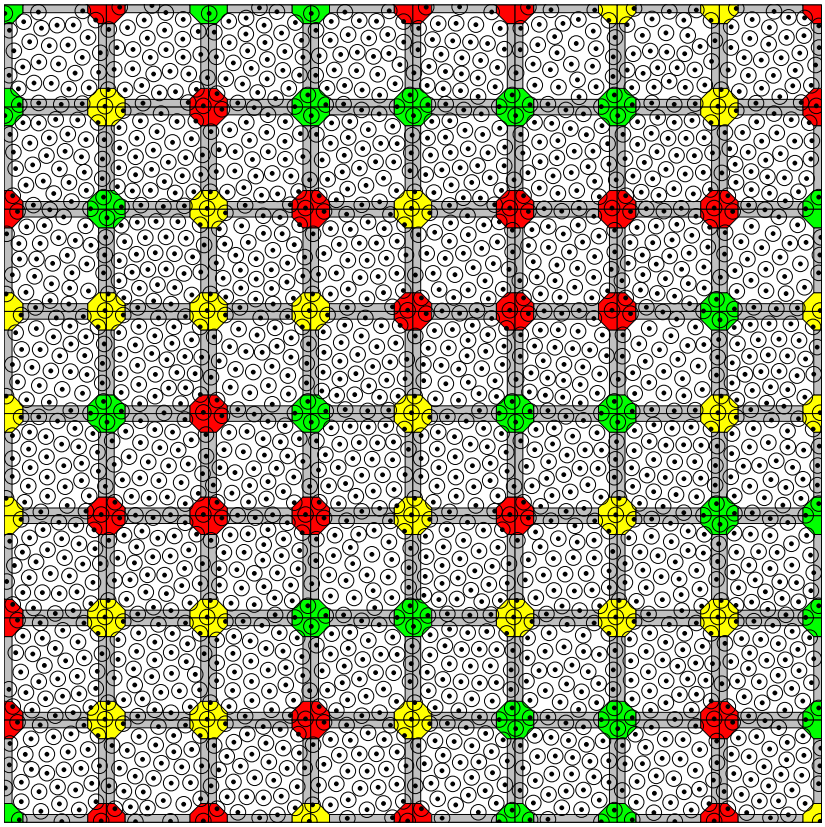


Figure 4.12: A tiling with a set of corner-based Poisson disk tiles.

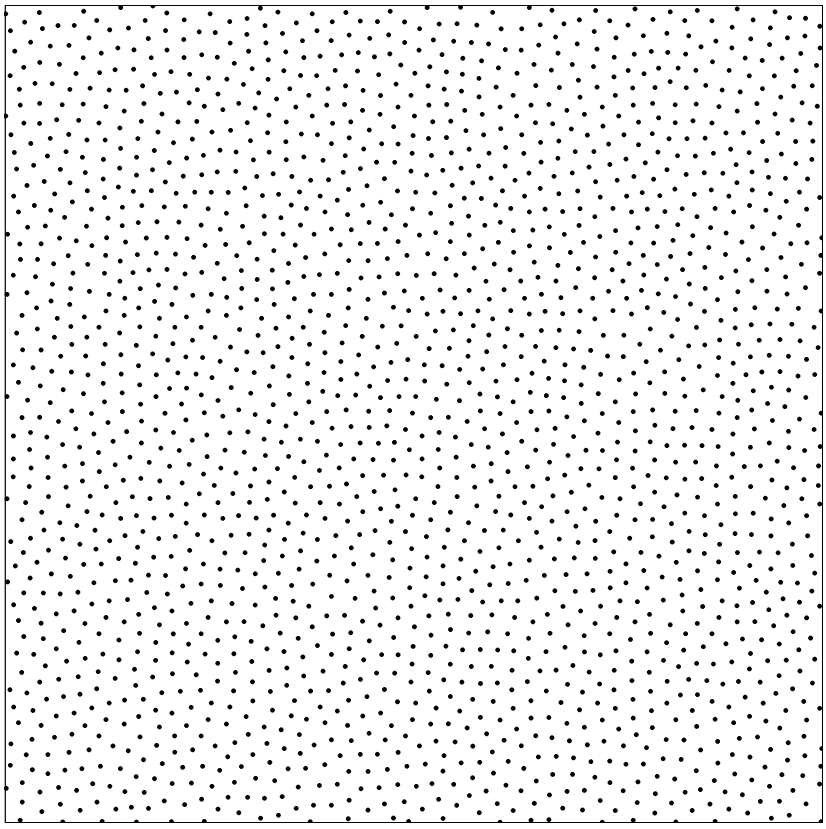


Figure 4.13: A Poisson disk distribution generated with a set of corner-based Poisson disk tiles. This Poisson disk distribution was generated from the tiling shown in figure 4.12.

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 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. Note that edge-based Poisson disk tiles use C^{12} rather than C^4 interior tiles.

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 ρ 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 4.5. For each corner tile, a toroidal Poisson disk distribution of N points is generated using dart throwing or relaxation dart throwing (see figure 4.5(b)), optionally followed by Lloyd's relaxation method (see figure 4.5(c)). The corner tile is then cut out of the Poisson disk distribution (see figure 4.5(d)). If the desired Poisson disk radius is not reached, this process is repeated. Figure 4.6 shows Poisson disk distributions constructed over corner tiles.

Next, a Poisson disk distribution is constructed over the edge tiles. This is illustrated in figure 4.7. Each edge tile is assembled with the corresponding corner tiles (see figure 4.7(a)). A toroidal Poisson disk distribution is generated using dart throwing or relaxation dart throwing (see figure 4.7(b)), optionally followed by Lloyd's relaxation method (see figure 4.7(c)). The edge tile is then cut out of the Poisson disk distribution (see figure 4.7(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 4.8 and 4.9 show Poisson disk distributions constructed over horizontal and vertical edge tiles.

Finally, a Poisson disk distribution is constructed over the interior tiles. This is illustrated in figure 4.10. Each interior tile is assembled with the corresponding corner tiles and edge tiles (see figure 4.10(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 4.10(b)), optionally followed by Lloyd's relaxation method (see figure 4.10(c)). The tile is then cut out of the Poisson disk distribution (see figure 4.10(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 4.11 shows Poisson disk distributions constructed over corner tiles.

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. Corner-based Poisson disk tile sets are significantly smaller than edge-based Poisson disk tile sets.

Figure 4.12 shows a tiling with a set of edge-based Poisson disk tiles, and figure 4.13 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.

4.4 A Tile-Based Method for Generating Poisson Sphere Distributions

Poisson disk distributions have many applications in computer graphics. Several of these applications, such as geometric object distribution (see section 6.5) and the two-dimensional procedural object distribution texture basis function (see section 6.6), have three-dimensional counterparts. These applications require Poisson sphere distributions, the three-dimensional equivalent of Poisson disk distributions.

In this section we present Poisson sphere distributions and three-dimensional corner tiles, and we introduce corner-based Poisson sphere tiles, a tile-based method for efficiently generating Poisson sphere distributions.

4.4.1 Poisson Sphere Distributions

Poisson sphere distributions are very similar to Poisson disk distributions, and most concepts of Poisson disk distributions easily generalize to Poisson sphere distribution. In this subsection we define Poisson sphere distributions, we extend the relative radius specification scheme to Poisson sphere distributions, and we discuss methods for generating Poisson sphere distributions.

4.4.1.1 Definition

A Poisson sphere distribution is a three-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 sphere of that radius is placed at each point, then no two spheres intersect.

4.4.1.2 Radius Specification

The relative radius specification scheme for Poisson disk distributions has a direct three-dimensional equivalent.

The packing density η of the densest packing of spheres is given by

$$\eta = \frac{\pi}{3\sqrt{2}} \approx 0.7405. \tag{4.4}$$

The packing density is defined as the fraction of the volume filled by the spheres.

The problem of finding the densest packing of spheres, also known as the Kepler Problem, was only solved recently [Cipra, 1998].

The maximum sphere volume of a Poisson sphere distribution counting N points over the toroidal unit cube is therefore η/N. The maximum possible sphere radius r_{max} of this Poisson sphere distribution is thus given by

$$r_{max} = \sqrt[3]{\frac{1}{4\sqrt{2}N}}. \tag{4.5}$$

The Poisson sphere radius r of a given point distribution is specified as a fraction ρ of the maximum disk radius

$$r = \rho\, r_{max}, \tag{4.6}$$

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

As in two dimensions, the relative radius is also a measure of how well the points are distributed. Good Poisson sphere distributions should have a relative radius that is relatively high.

4.4.1.3 Generation

Poisson disk distributions are traditionally generated with dart throwing, relaxation dart throwing or Lloyd's relaxation method. These algorithms easily generalize to three dimensions.

4.4.2 Three-Dimensional Corner Tiles

Three-dimensional corner tiles are a simple extension of two-dimensional corner tiles.

Three-dimensional corner tiles are unit cube tiles with colored corners. The corners of a three-dimensional corner tile are named after its coordinates. A complete tile set contains a tile for every possible combination of eight corner colors. A complete set of three-dimensional corner tiles over C colors counts C^8 tiles. Figure 4.14 shows several tiles

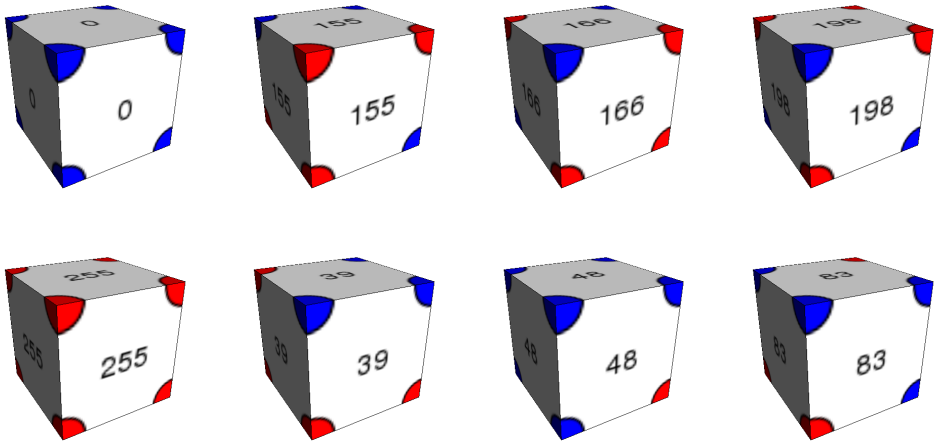


Figure 4.14: Several tiles of the complete 3D corner tile set over 2 colors.

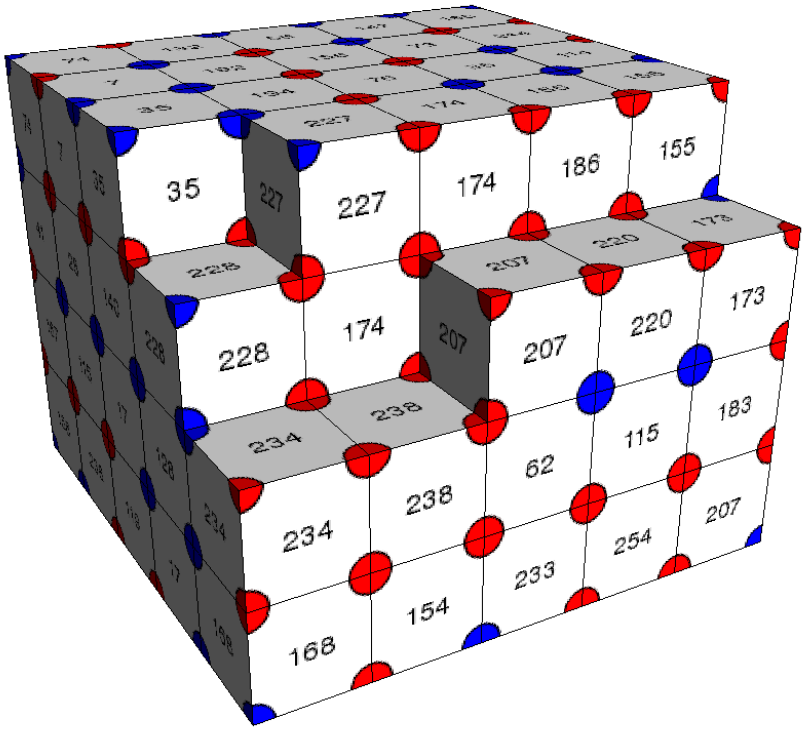


Figure 4.15: A tiling with the complete 3D corner tile set over 2 colors.

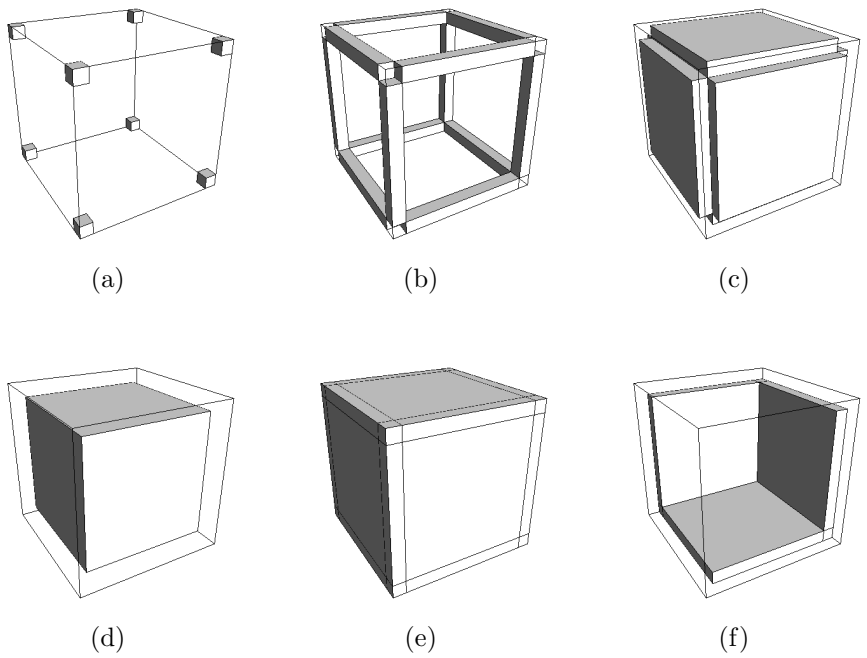


Figure 4.16: The Poisson sphere tile regions. (a) The corner regions. (b) The edge regions. (c) The face regions. (d) The interior region. (e) The assembled tile. (f) The partially assembled tile.

of the complete set of three-dimensional corner tiles over two colors. A complete set of three-dimensional corner tiles over 2, 3 and 4 colors consist of 256, 6,561 and 65,536 tiles.

A tiling is constructed by placing the tiles next to each other such that adjoining 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. Figure 4.15 shows a tiling with the complete set of three-dimensional corner tiles over two colors.

The enumeration scheme for corner tiles (see section 2.8) and the direct stochastic tiling algorithm for corner tiles (see section 3.3) easily generalize to three dimensions.

4.4.3 Corner-Based Poisson Sphere Tiles

In 2006, we presented corner-based Poisson sphere tiles, a tile-based method for efficiently generating Poisson sphere distributions [Lagae and Dutré, 2006c]. Corner-based Poisson sphere tiles are an extension of corner-based Poisson disk tiles (see section 4.3) to three dimensions.

A point in a tile closer to a corner than the Poisson sphere radius affects points in seven

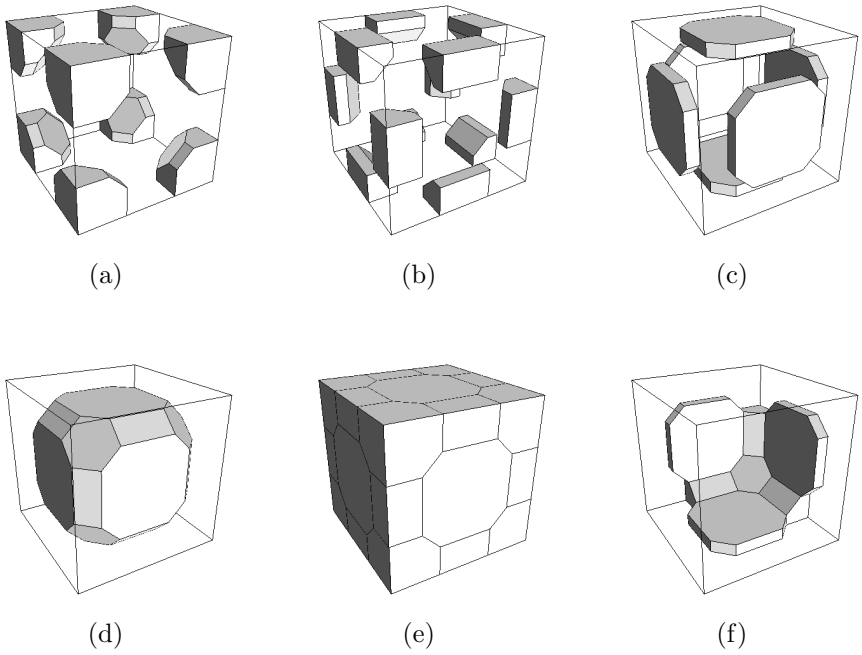


Figure 4.17: The modified Poisson sphere tile regions. (a) The modified corner regions. (b) The modified edge regions. (c) The modified face regions. (d) The modified interior region. (e) The assembled tile. (f) The partially assembled tile.

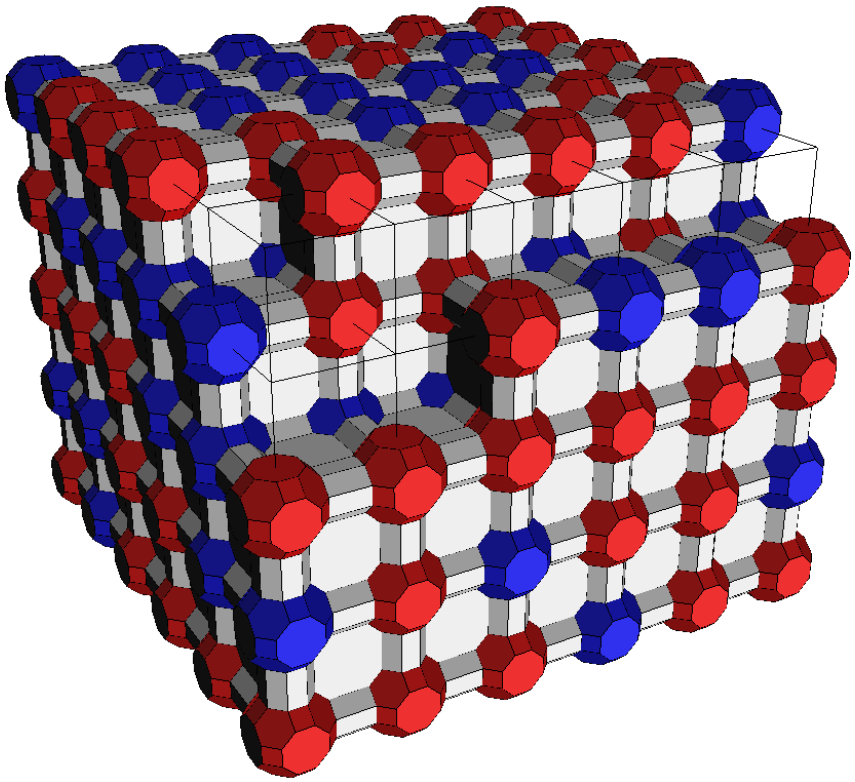
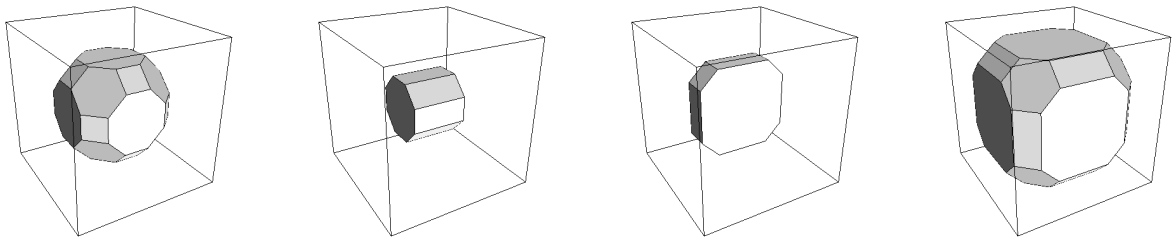


Figure 4.18: A tiling obtained by combining the modified Poisson sphere tile regions with the complete 3D corner tile set over 2 colors. This tiling was generated from the tiling shown in figure 4.15.

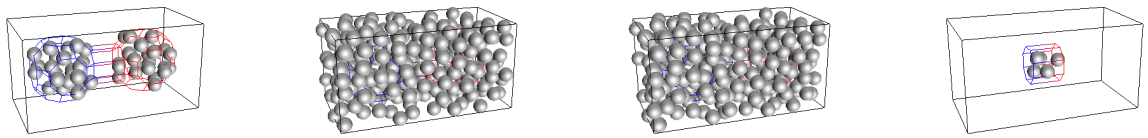


(a) (b) (c) (d)

Figure 4.19: The four kinds of tiles in the tiling obtained by combining the modified Poisson sphere tile regions with the complete 3D corner tile set over 2 colors. (a) Corner tiles. (b) Edge tiles. (c) Face tiles. (d) Interior tiles.

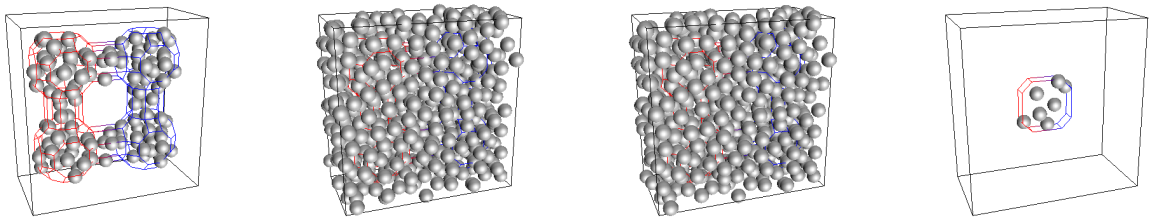
(a) (b) (c) (d)

Figure 4.20: Construction of a Poisson sphere distribution over a corner tile of a corner-based Poisson sphere tile set. (a) The corner tile. (b) A toroidal Poisson sphere distribution is generated. (c) The Poisson sphere distribution is optimized using Lloyd's relaxation scheme. (d) The corner tile is cut out of the Poisson disk distribution.



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

Figure 4.21: Construction of a Poisson sphere distribution over an edge tile of a corner-based Poisson sphere tile set. (a) The edge tile is assembled with the corresponding corner tiles. (b) A toroidal Poisson sphere distribution is generated. (c) The Poisson sphere distribution is optimized using Lloyd's relaxation scheme. (d) The edge tile is cut out of the Poisson sphere distribution.



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

Figure 4.22: Construction of a Poisson sphere distribution over a face tile of a corner-based Poisson sphere tile set. (a) The face tile is assembled with the corresponding corner tiles and edge tiles. (b) A toroidal Poisson sphere distribution is generated. (c) The Poisson sphere distribution is optimized using Lloyd's relaxation scheme. (d) The face tile is cut out of the Poisson sphere distribution.

neighboring tiles. A point in a tile closer to an edge than the Poisson sphere radius affects points in three neighboring tiles. A point in a tile closer to a face than the Poisson sphere radius affects points in one neighboring tile. A point in a tile, further away from the tile boundary than the Poisson sphere radius does not affect points in neighboring tiles. The regions obtained this way are called the Poisson sphere tile regions. The Poisson sphere radius determines corner regions, edge regions, face regions and an interior region. This is illustrated in figure 4.16.

To minimize the constraints between the different regions, the corner regions are enlarged such that the distance between edge regions is twice the Poisson sphere radius. Now points in different edge regions do not affect each other. The edge regions are enlarged such that the distance between face regions is twice the Poisson sphere radius. Now points in different face regions do not affect each other. The regions obtained this way are called the modified Poisson sphere tile regions. This is illustrated in figure 4.17.

By combining the modified Poisson sphere tile regions with the complete corner tile set over C colors, a new tiling is obtained. This is illustrated in figure 4.18. This tiling uses four different kinds of tiles. Corner tiles, edge tiles, face tiles and interior tiles. Corner tiles correspond to the union of eight modified corner regions. There are C corner tiles, one for each color. Edge tiles correspond to the union of four modified edge regions. There are C^2 edge tiles for each orientation, one for each combination of two corner colors. Face tiles correspond to the union of two modified face regions. There are C^4 face tiles for each orientation, one for each combination of four corner colors. Interior tiles correspond to the modified interior regions. There are C^8 interior tiles, one for each combination of eight corner colors. These four kinds of tiles are shown in figure 4.19. Note that the corner tile

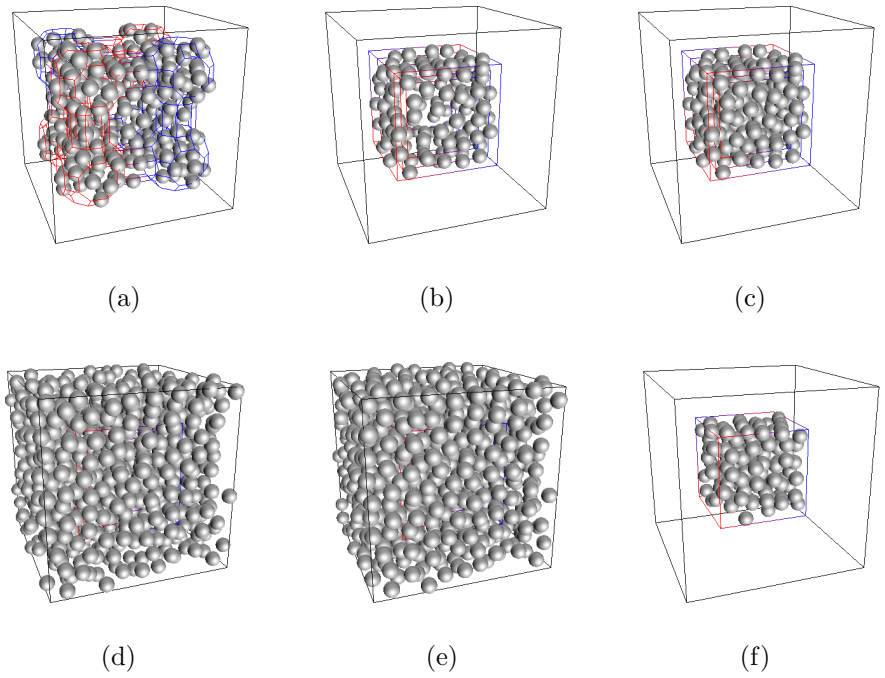


Figure 4.23: Construction of a Poisson sphere distribution over a tile of a corner-based Poisson sphere tile set. (a) The interior tile is assembled with the corresponding corner tiles, edge tiles and face tiles. (b) All points further from the tile than twice the Poisson sphere radius are discarded. (c) A toroidal Poisson sphere distribution is generated in the interior. (d) A toroidal Poisson sphere distribution is generated in the exterior. (e) The Poisson sphere distribution is optimized using Lloyd's relaxation scheme. (f) The tile is cut out of the Poisson sphere distribution.

is a great rhombicuboctahedron, an Archimedean solid.

To construct a Poisson sphere 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 sphere radius ρ are chosen. The absolute Poisson sphere radius determines the size of the modified Poisson sphere regions.

First, a Poisson sphere distribution is constructed over the corner tiles. This is illustrated in figure 4.20. For each corner tile, a toroidal Poisson sphere distribution of N points is generated using dart throwing or relaxation dart throwing (see figure 4.20(b)), optionally followed by Lloyd's relaxation method (see figure 4.20(c)). The corner tile is then cut out of the Poisson sphere distribution (see figure 4.20(d)). If the desired Poisson sphere radius is not reached, this process is repeated.

Next, a Poisson sphere distribution is constructed over the edge tiles. This is illustrated in figure 4.21. Each edge tile is assembled with the corresponding corner tiles (see figure 4.21(a)). A toroidal Poisson sphere distribution is generated using dart throwing or relaxation dart throwing (see figure 4.21(b)), optionally followed by Lloyd's relaxation method (see figure 4.21(c)). The edge tile is then cut out of the Poisson sphere distribution (see figure 4.21(d)). If the desired Poisson sphere 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.

Next, a Poisson sphere distribution is constructed over the face tiles in the same way. This is illustrated in figure 4.22.

Finally, a Poisson sphere distribution is constructed over the interior tiles. This is illustrated in figure 4.23. Each interior tile is assembled with the corresponding corner tiles, edge tiles and face tiles. (see figure 4.23(a)). All points further away from the tile boundary than the Poisson sphere radius are discarded (see figure 4.23(b)). A Poisson sphere distribution is generated using dart throwing or relaxation dart throwing, both in the inside (see figure 4.23(c)) and in the outside (see figure 4.23(d)), optionally followed by Lloyd's relaxation method (see figure 4.23(e)). The tile is then cut out of the Poisson sphere distribution (see figure 4.23(f)). If the desired Poisson sphere radius is not reached, this process is repeated.

A corner-based Poisson sphere tile set based on a complete corner tile set over C colors consists of C^8 tiles. The only practical choice for C is 2, which results in a corner-based Poisson sphere tile set counting 256 tiles.

The time needed to generate a Poisson sphere 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 sphere distributions can be generated.

4.5 Conclusion

In this chapter we have introduced Poisson disk distributions and we have proposed an intuitive scheme for specifying the radius of a Poisson disk distribution. We have presented edge-based Poisson disk tiles, template Poisson disk tiles, and corner-based Poisson disk tiles, three tile-based methods for generating Poisson disk distributions. We have introduced Poisson sphere distributions and three-dimensional corner tiles, and we have presented corner-based Poisson sphere tiles, a tile-based method for constructing Poisson sphere distributions. We have discussed nonuniform Poisson disk distributions and we have proposed a tile-based method for generating nonuniform well-distribution point distributions.

Chapter 5

Tile-Based Methods for Texture Synthesis

5.1 Introduction

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.

Overview

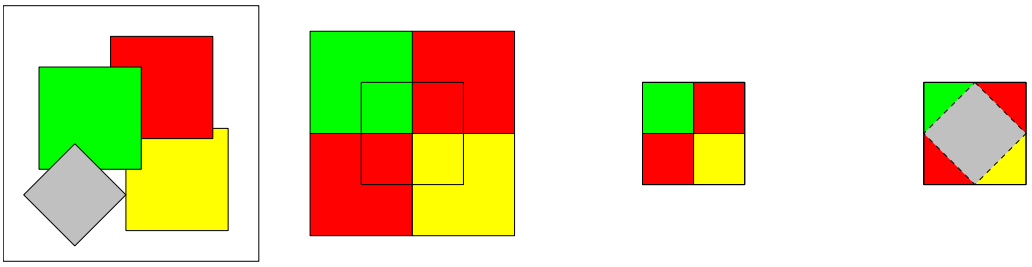
This chapter is organized as follows. In section 5.2 we introduce texture mapping and texture synthesis. Section 5.3 presents a method for synthesizing a texture over a set of Wang tiles or corner tiles. In section 5.4 we propose a tile-based texture mapping algorithm. Section 5.5 discusses the tile packing problem. In section 5.6 we conclude.

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.

5.2 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



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

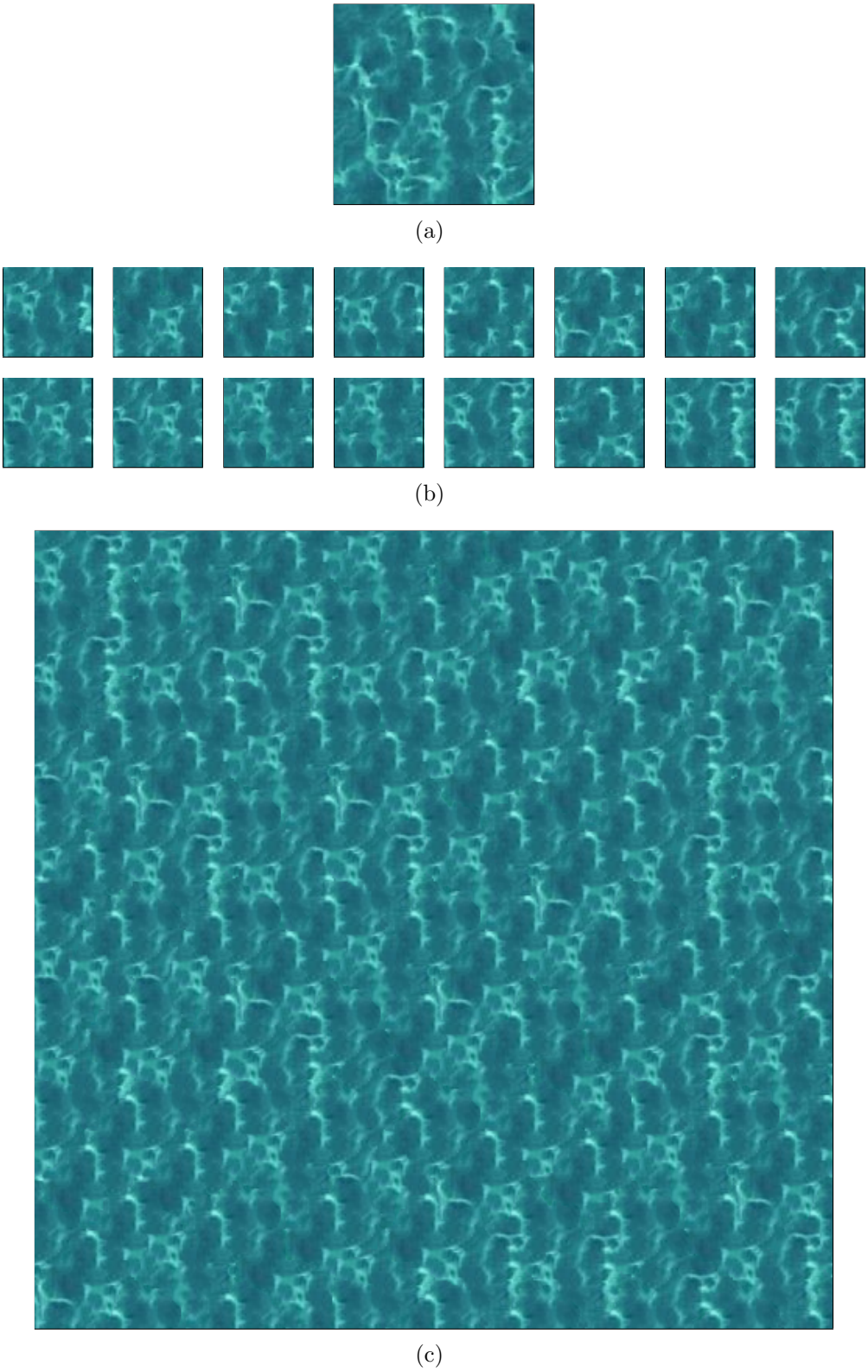
Figure 5.1: Construction of a texture tile set based on corner tiles from an example texture. (a) For each color, a square patch is chosen in the example texture (the red, green and yellow patch). (b) The patches are assembled according to the corner colors of the tile. (c) The tile is cut out. (d) The seam is covered with a new irregular patch from the example texture (the gray patch).

texture synthesis is discussed in detail in section 6.6. 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 work. For an overview, we 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.

5.3 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.



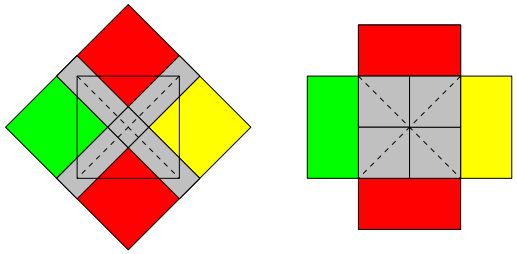
(a)

(b)

(c)

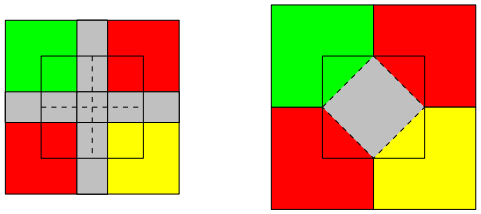
Figure 5.2: Texture synthesis with texture tiles based on corner tiles. (a) The example texture. (b) A set of texture tiles based on corner tiles constructed from the example texture. (c) A new texture synthesized with the texture tile set.

Figure 5.3: 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 two or three colors.



(a) (b)

Figure 5.4: 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 5.5: 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.

was extended to corner tiles by Ng et al. [2005].

We use the method of Ng et al. for synthesizing an example texture over a set of corner tiles. This is illustrated in figure 5.1. For each corner color, a square patch is chosen at random from the example texture (see figure 5.1(a)). Each tile of the tile set is constructed by combining the four patches corresponding to the corner colors (see figure 5.1(b)), and cutting out the tile (see figure 5.1(c)). This leaves a cross shaped seam that is covered with a new diamond-shaped irregular patch from the example texture (see figure 5.1(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 5.2. The method of Ng et al. is simple and works well. Figure 5.3 shows several textures synthesized with corner-based texture tiles. The quality of the synthesized textures is similar to that of other patch-based techniques.

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 5.4 and 5.5 show several patch combination strategies for Wang tiles and corner tiles.

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. We 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.

5.4 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

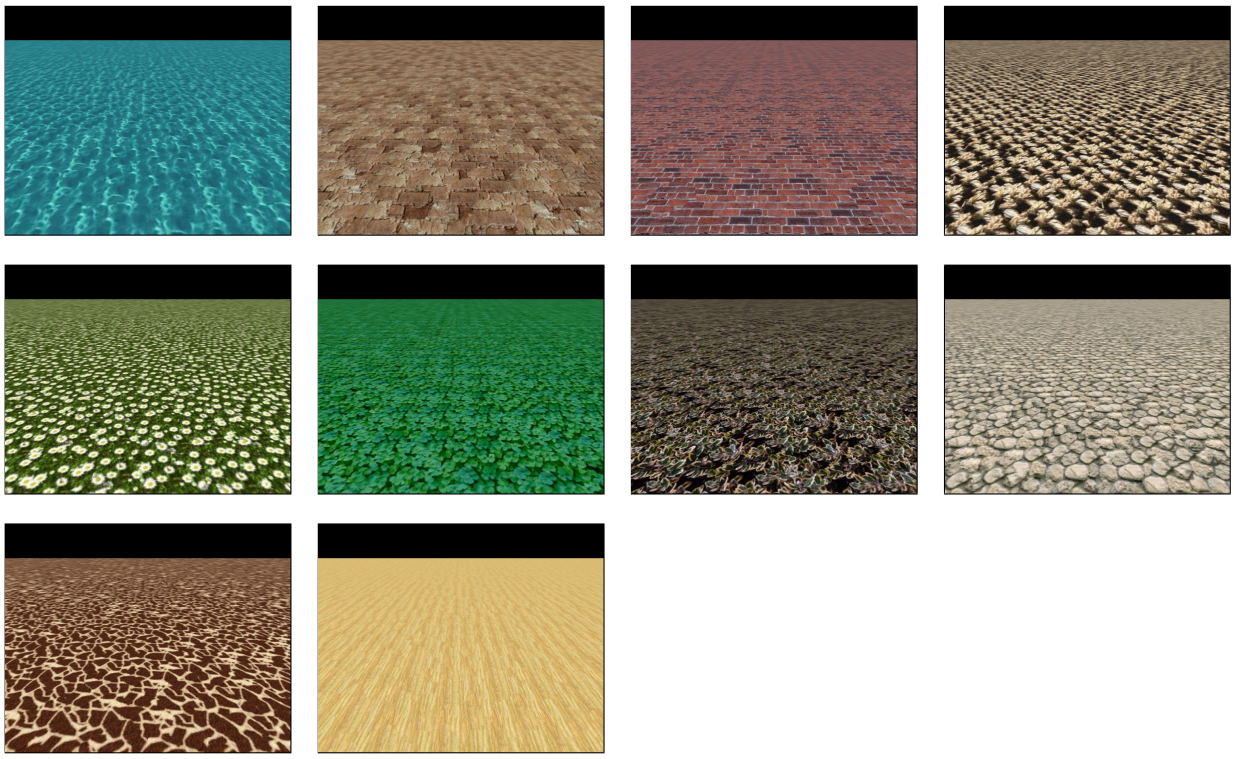


Figure 5.6: Tile-based texture mapping using corner tiles. Screenshots from our interactive tile-based texture mapping application based on corner tiles. Texture filtering does not introduce unwanted artifacts because a tile packing was used.

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, we presented a cleaner and more efficient variant version of this algorithm based on corner tiles [Lagae and Dutré, 2006a]. In this section we discuss 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, we 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 5.5.

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 we 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 5.5). 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

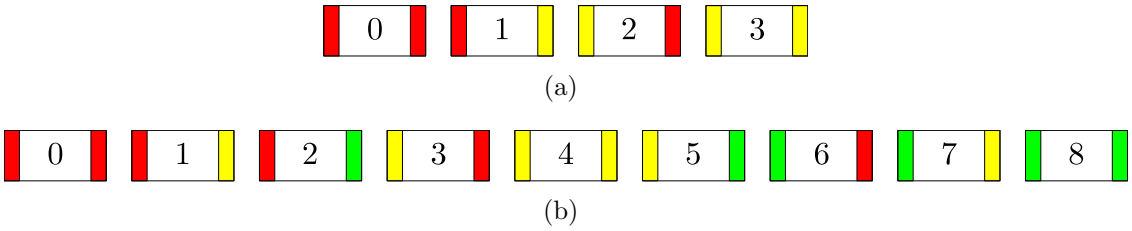


Figure 5.7: The complete 1D Wang or corner tile sets over (a) 2 and (b) 3 colors.

tiles are not subject to the corner problem, only the texture containing the tile packing is needed. Compared to the original method of Wei, our 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. Our 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.

Our 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 5.6 shows several results.

5.5 The Tile Packing Problem

A tile packing is an essential ingredient of the tile-based texture mapping algorithms discussed in section 5.4. A tile packing is used to avoid unwanted texture filtering artifacts. In this section we discuss the problem of computing a tile packing of a complete set of Wang tiles or corner tiles. We also show 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, we studied the corner tile packing problem in detail [Lagae and Dutré, 2006a,d].

5.5.1 The One-Dimensional Tile Packing Problem

In one dimension, Wang tiles and corner tiles can be seen as dominoes. A complete set of Wang tiles or corner tiles over C colors counts C^2 tiles. Figure 5.7 shows the complete set of tiles over 2 and 3 colors. The one-dimensional tile packing problem consists of arranging

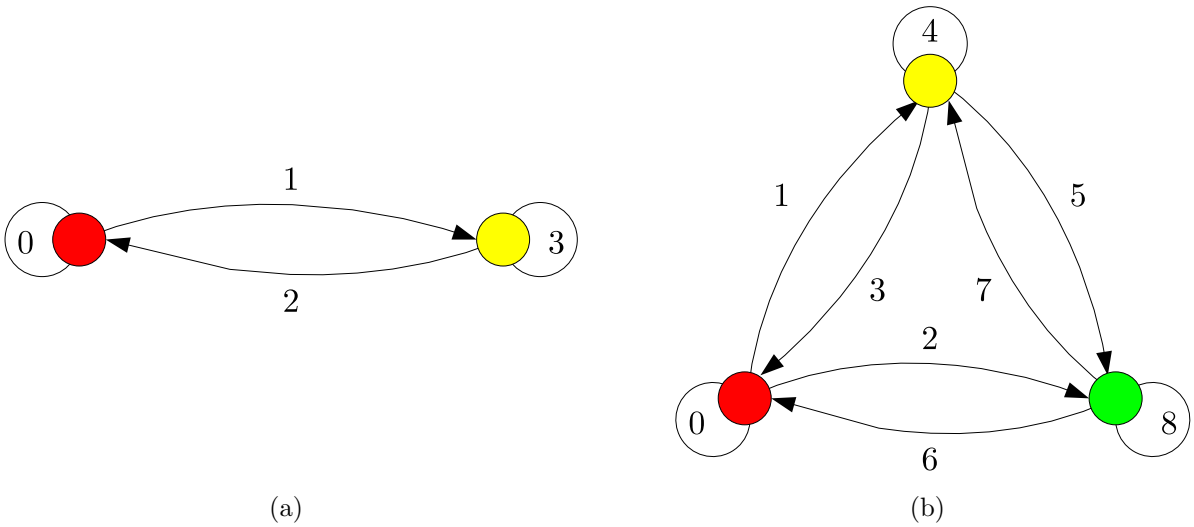


Figure 5.8: Graphs representing the complete 1D Wang or corner tile set over (a) 2 and (b) 3 colors.

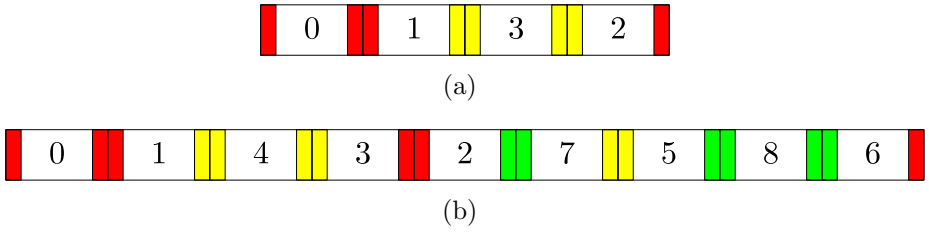


Figure 5.9: Tile packings of the complete 1D Wang or corner tile set over 2 and 3 colors.

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].

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 5.8 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 5.9 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.

5.5.2 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 5.10.

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.

5.5.3 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 does not seem to extend to corner tiles.

When we started exploring the corner tile packing problem, it resisted all attempts to solve it. It was not clear whether the corner tile packing problem even had a solution. We therefore decided to tackle 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 2.09×10^{13}, 5.80×10^{120} and 8.58×10^{506}. Instead we use a backtracking method, that

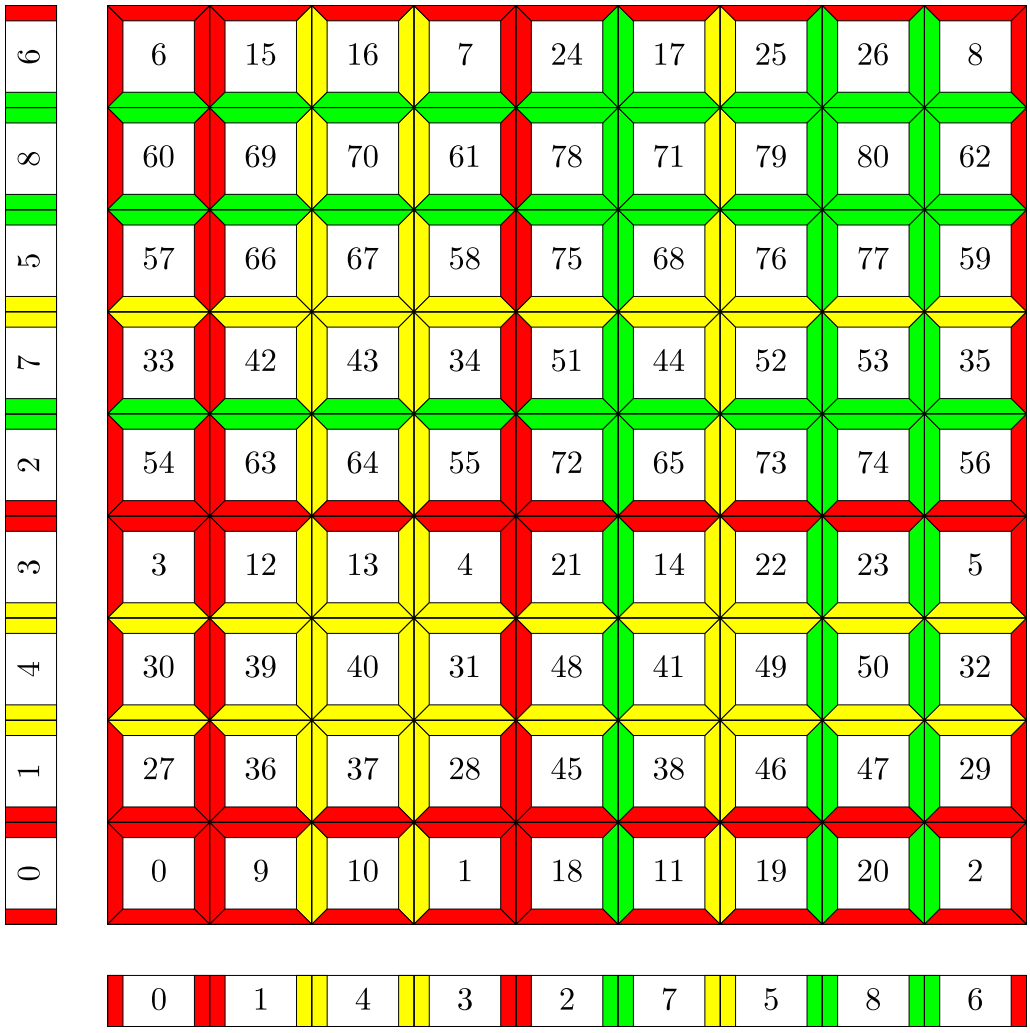


Figure 5.10: The construction of an Eulerian tile packing of the complete Wang tile set over 3 colors.

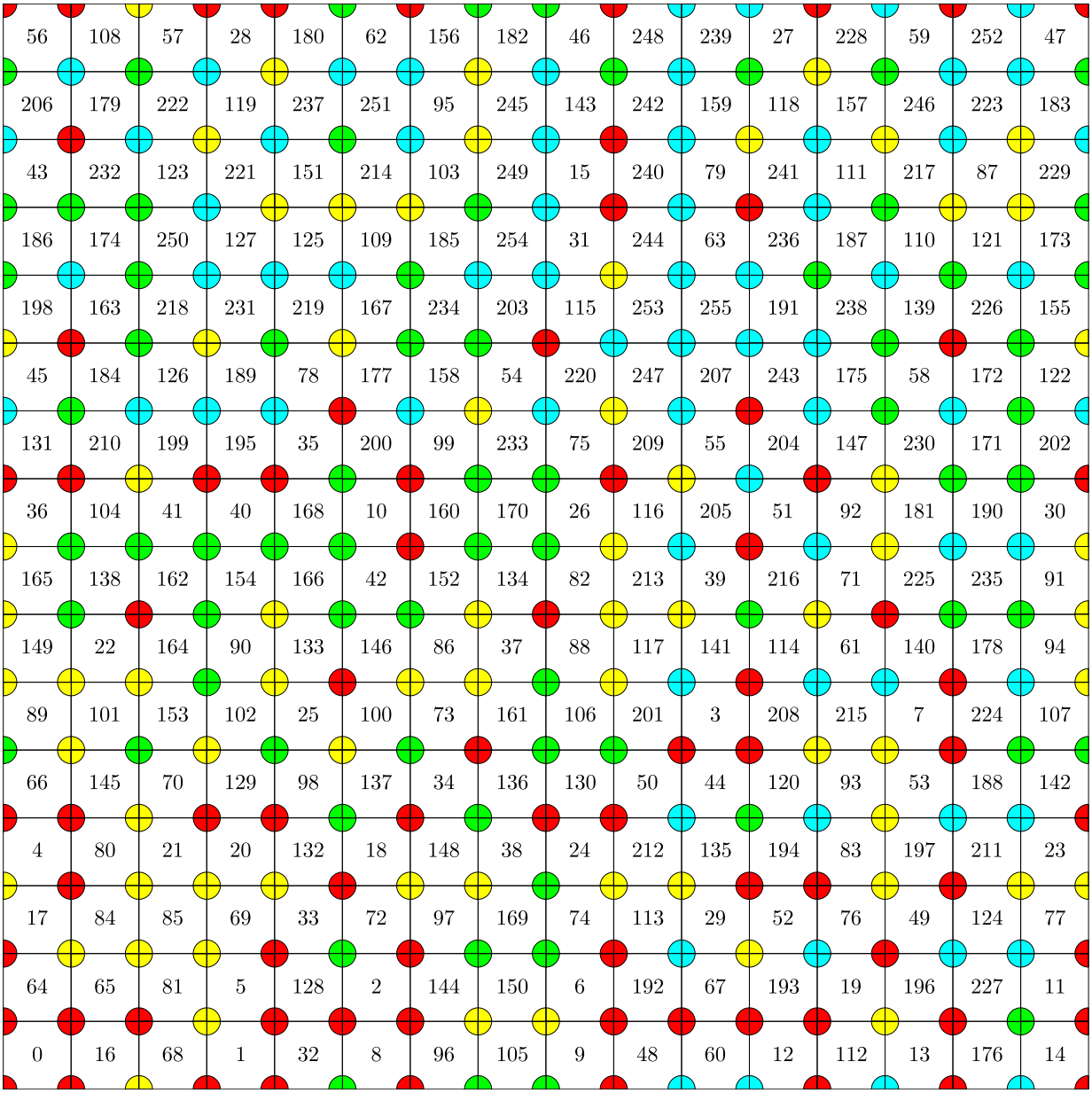


Figure 5.11: A recursive tile packing of the complete corner tile set over 4 colors.

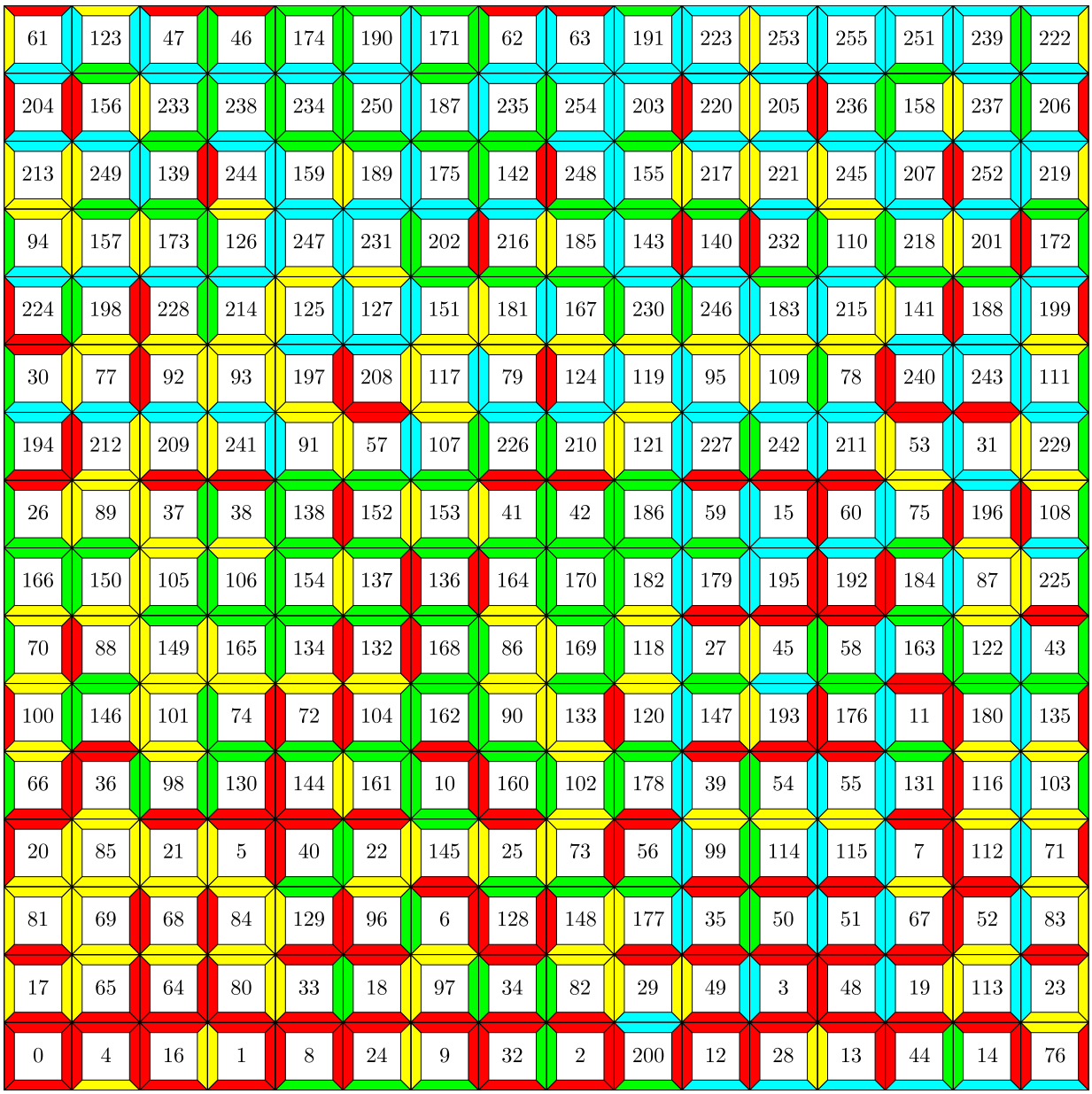


Figure 5.12: A recursive tile packing of the complete Wang tile set over 4 colors.

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, we refer to [Lagae and Dutré, 2006d].

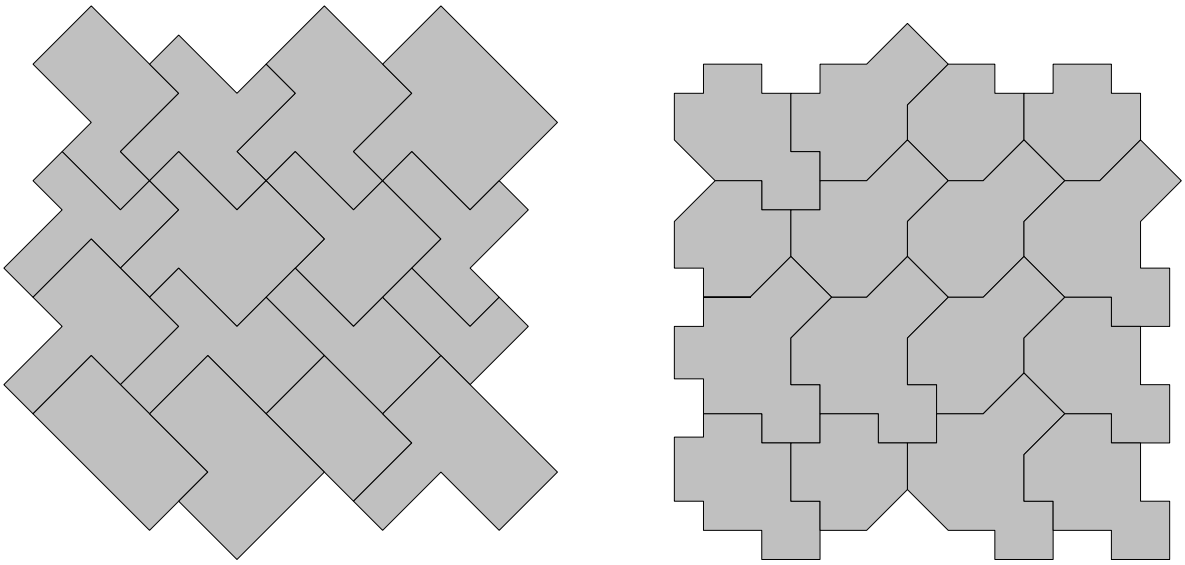
With the backtracking algorithm, we are able to compute Wang and corner tile packings for 2, 3 and 4 colors. For 2 colors, all solutions of the Wang and corner tile packing problem are obtained almost immediately on a regular desktop computer. The corner tile packing problem has 32 solutions and the 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 our 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 5.12 and 5.11 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. The many symmetries of the tile packing problem are currently not exploited. To solve the corner tile packing problem we have also experimented with a backtracking algorithm that places colored pegs rather than tiles. This algorithm seems to be faster.

The tile packing problem is an interesting combinatorial puzzle. We were able to obtain Wang and corner tile packings for up to 4 colors. However, several problems, such as counting the number of solutions of the tile packing problem, and finding a constructive method and a closed-form expression for the corner tile packing problem, remain unsolved.



(a) (b)

Figure 5.13: Jigsaw puzzles derived from tile packings of the complete (a) Wang and (b) corner tile set over 2 colors.

5.5.4 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, we have created jigsaw puzzles from tile packings by altering the profile of the adjoining edges or corners. Figure 5.13 shows two examples. To create interesting puzzles, it is better to use tile packings constructed with the backtracking algorithm instead of Eulerian tile packings. We found it 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.

5.6 Conclusion

In this chapter we have investigated the use of Wang tiles and corner tiles in texture mapping and texture synthesis. We have introduced texture mapping and texture synthesis. We have discussed a method for synthesizing a texture over a set of Wang tiles or corner tiles. We have presented a tile-based texture mapping algorithm, and we have discussed the tile packing problem. For tile-based texture mapping, corner tiles are clearly superior to Wang tiles. For tile-based texture synthesis corner tiles are also better than Wang tiles, but the difference is less pronounced. Tile packings are more difficult to compute for corner tiles than for Wang tiles. However, a tile packing has to be computed only once.

Chapter 6

Applications of Poisson Disk Distributions

6.1 Introduction

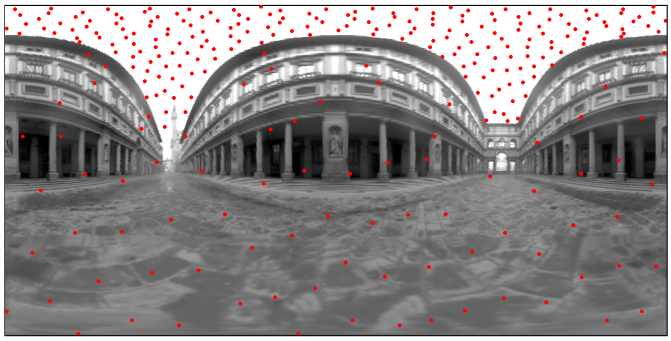
In the previous chapters we have introduced and analyzed efficient methods for generating Poisson disk distributions. Generating Poisson disk distributions is of course not a goal in itself, Poisson disk distributions have several applications in computer graphics. Efficient methods for generating Poisson disk distributions enable efficient implementation of these applications but also enable completely new applications. In this chapter we discuss several applications of Poisson disk distributions.

Overview

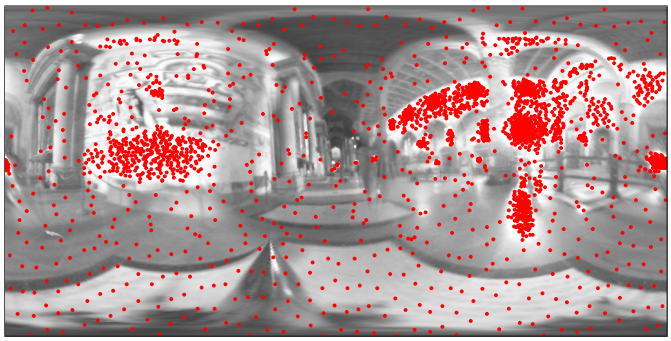
This chapter is organized as follows. Section 6.2 discusses sampling. In section 6.3 we discuss applications in non-photorealistic rendering. Section 6.4 introduces applications in scientific visualization. In section 6.5 we discuss procedural modeling, geometric object distribution and geometry instancing. Section 6.6 introduces a new application of Poisson disk distributions in procedural texturing. In section 6.7 we conclude.

6.2 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



(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.)

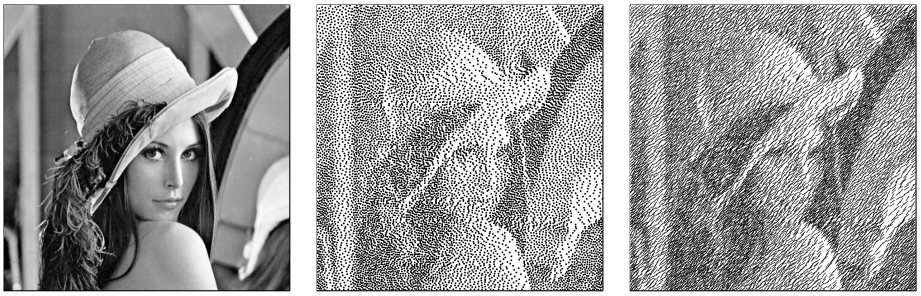
distributed according to a Poisson disk distribution, and presented theoretical evidence in favor of the Poisson disk distribution. It is now generally accepted that because of it's blue noise power spectrum, the Poisson disk distribution is one of the best stochastic sampling patterns.

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. Another solution is to directly generate a nonuniform Poisson disk distribution. However, both techniques are not optimal. Warping can introduce clumping, and using a single tile introduces periodicity. Efficiently generating sampling patterns with blue noise properties is still a very active area of research [Ostromoukhov et al., 2004; Dunbar and Humphreys, 2006; Kopf et al., 2006].

6.3 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) (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.

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. Another solution is to directly generate a nonuniform Poisson disk distribution.

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.

6.4 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 visualization also employs other

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.)

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.

6.5 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 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.

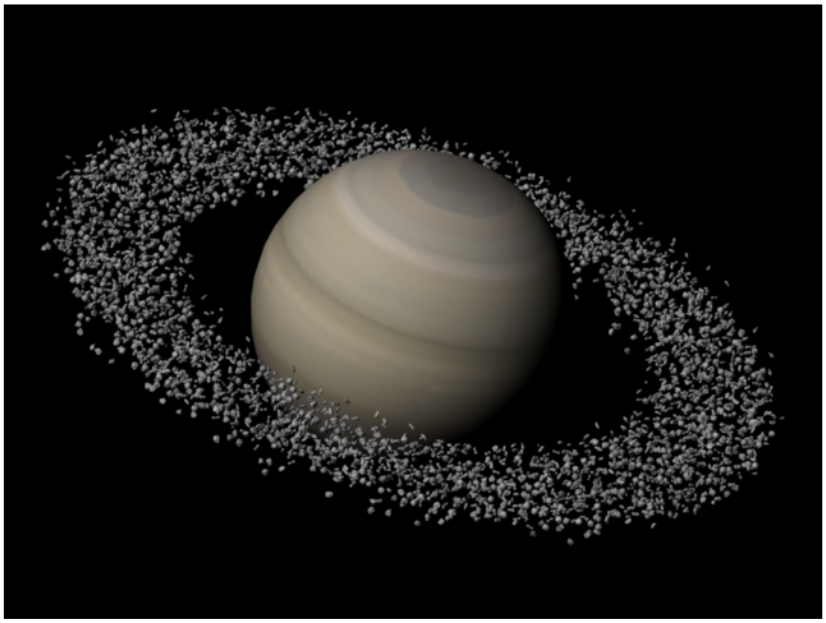


Figure 6.4: A planet with an asteroid belt. The asteroid belt was modeled by instancing several thousand asteroids using a Poisson sphere distribution. (The map of Saturn is courtesy of Björn Jónsson. The asteroid models are courtesy of Scott Hudson.)

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 4. 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.

6.6 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, we present a procedural object distribution function. This new 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. We discuss the history and background of procedural texturing, and present a two-dimensional as well as a tree-dimensional procedural object distribution function.

6.6.1 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.

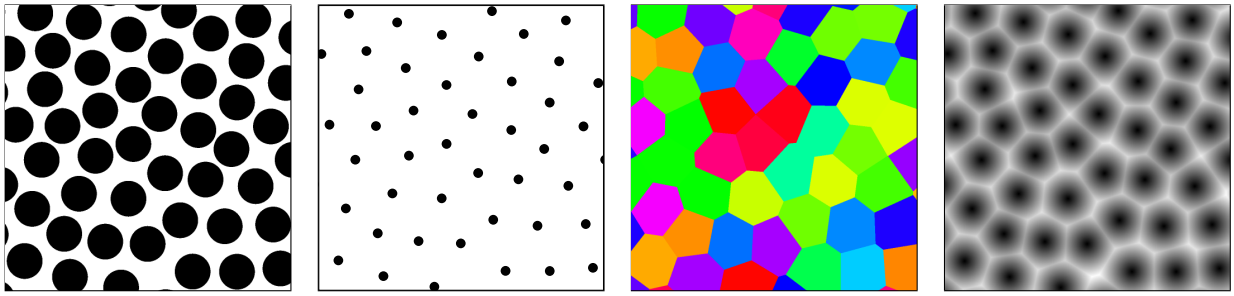
The most popular three-dimensional texture basis function is Perlin's noise function [Perlin, 1985; Perlin and Hoffert, 1989; Perlin, 2002]. The noise value at each point is determined by computing a pseudo-random gradient at each of the eight nearest vertices on the integer cubic lattice, followed by 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].

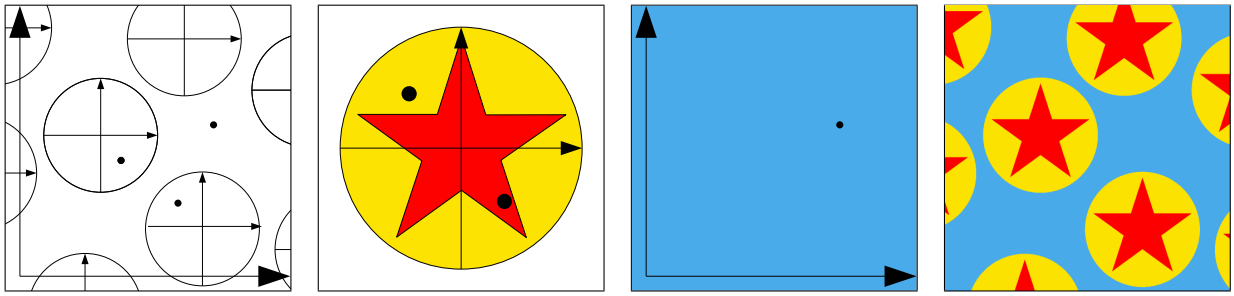
There are several other techniques to generate textures procedurally. For example, Turk [1991] presented a biologically inspired method, called reaction-diffusion, that generates interesting mammalian patterns. These methods, however, do not qualify as texture basis functions, because they do not have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

For an excellent overview of the field of procedural texturing and modeling, we refer to [Ebert et al., 2002].



(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.



(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.

6.6.2 A 2D Procedural Object Distribution Function

The two-dimensional procedural object distribution function is 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.

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.

In the remainder of this subsection, we discuss how to evaluate the texture basis function efficiently, and how to control the placement of the distributed objects. We also present several results and discuss some more advanced topics.

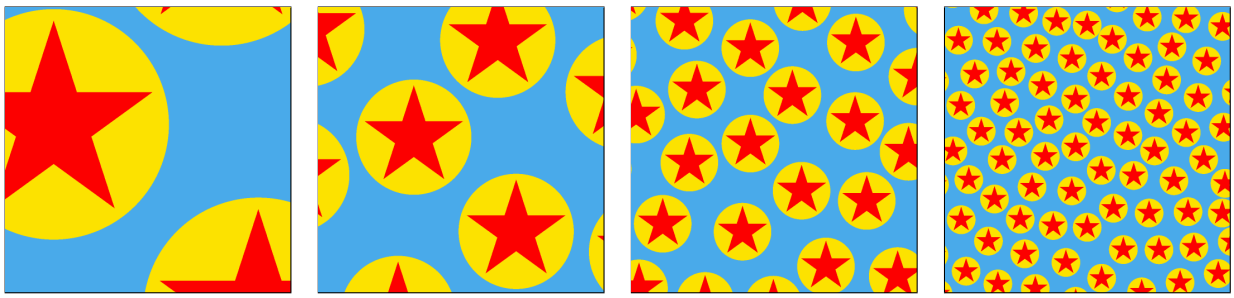
6.6.2.1 Evaluation

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



(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.

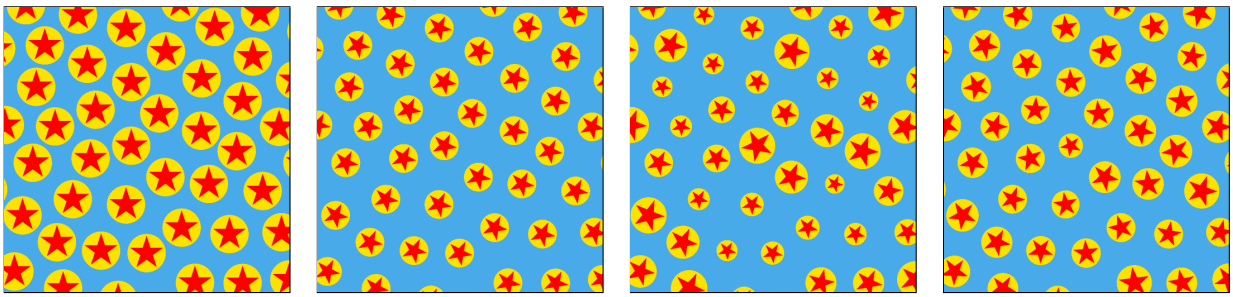
of the largest empty circle r_e is computed. Alternatively, r_e can be bounded analytically. This radius determines different regions in the tile, much like the ones in figure 4.3(a). If the point of evaluation (x, y) is closer to a corner than r_e, three neighboring tiles have to be considered. Else, 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 our texture basis function also suited for interactive and real-time applications.

6.6.2.2 Parameters

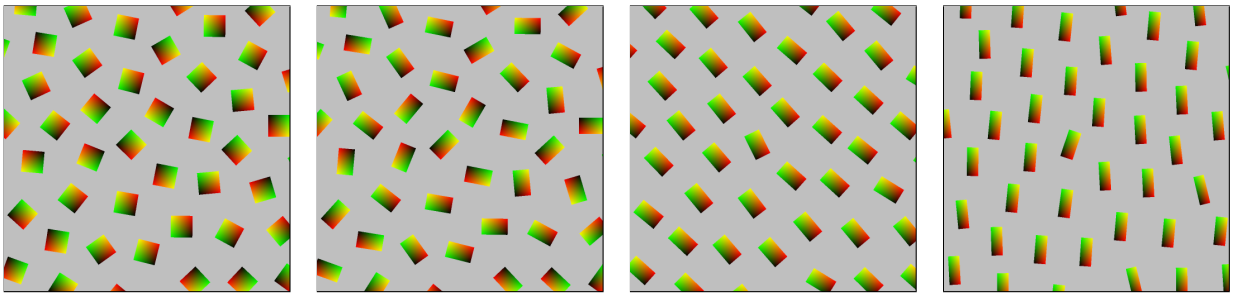
The placement of the distributed objects can be controlled by four parameters: the scale s, the size r, the orientation θ and the aspect ratio a.

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



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

Figure 6.8: Manipulation of the size r and orientation θ 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.



(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.

is different from the original scale parameter introduced in [Lagae and Dutré, 2005]. 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 θ 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 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.

6.6.2.3 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 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

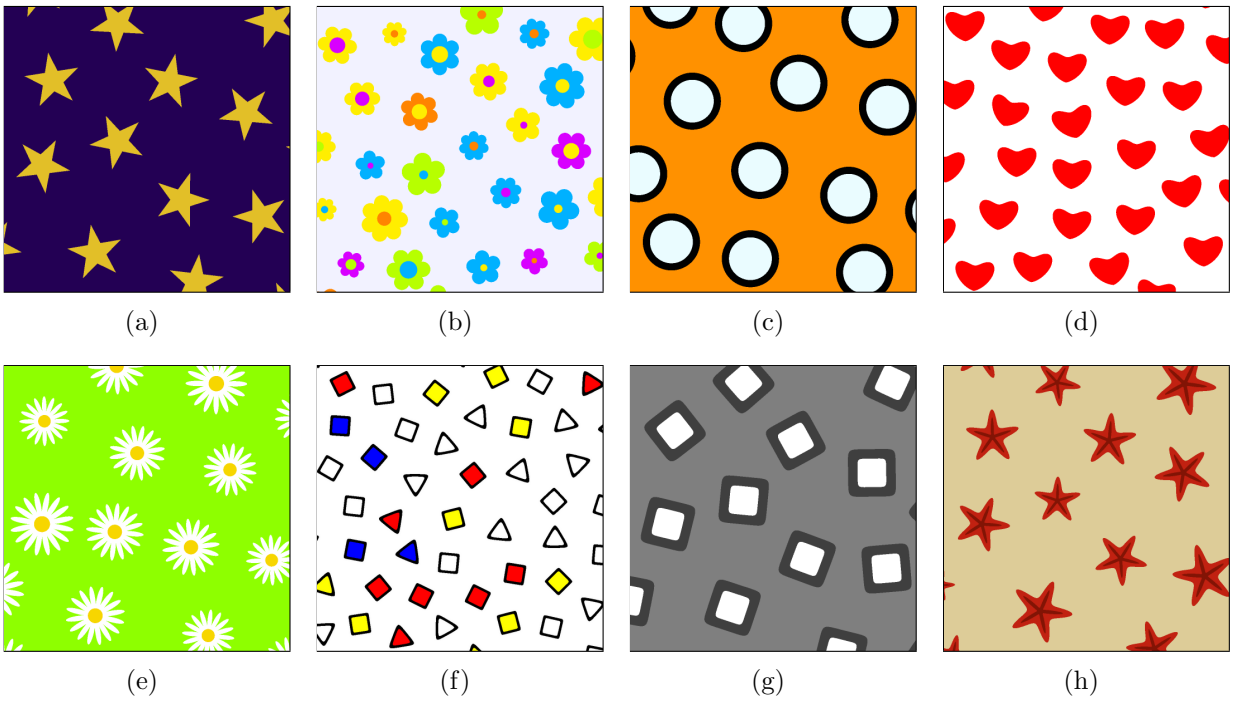


Figure 6.10: Textures generated with the 2D object distribution texture basis function. (a) Stars. (b) Flowers. (c) Polka dots. (d) Hearts. (e) Daisies. (f) Mondriaan shapes. (g) Abstract squares. (h) Starfish.

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

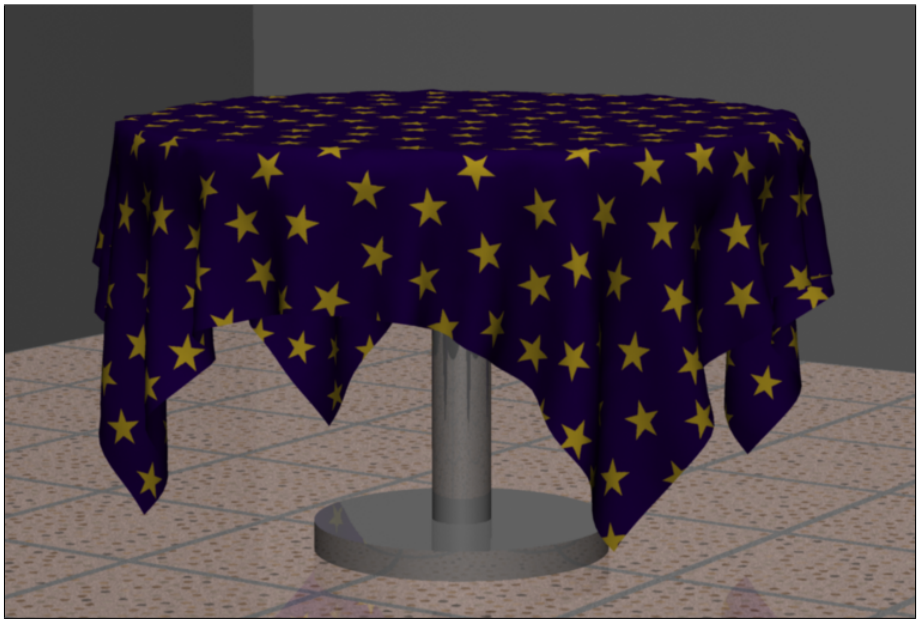


Figure 6.12: A table with a table cloth on a granite floor. The textures used in this scene were generated with the 2D procedural object distribution function.

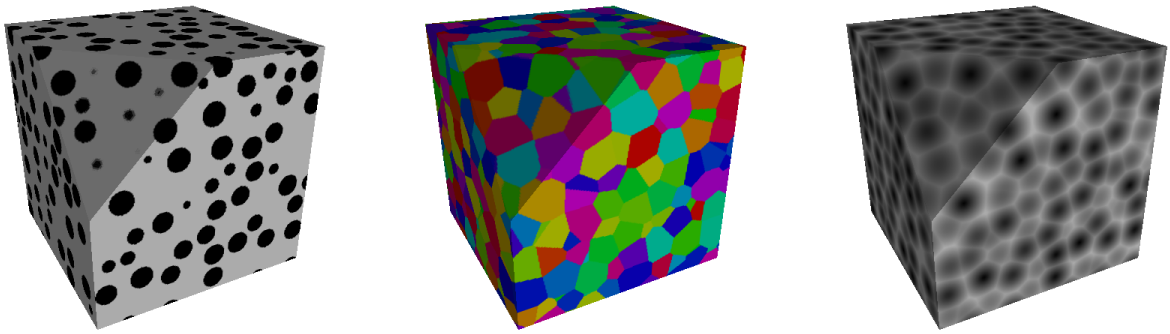
one are $m = 5$, $n_1 = 2$, $n_2 = n_3 = 7$ and $a = b = 1$, and the parameters for the inner one are $m = 5$, $n_1 = 2$, $n_2 = n_3 = 13$ and $a = b = 1$. The particles in the granite of figure 6.12 are random convex hexagons. The color of these particles, the color of the mortar and the base color were modulated with Perlin noise.

6.6.2.4 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.



(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.

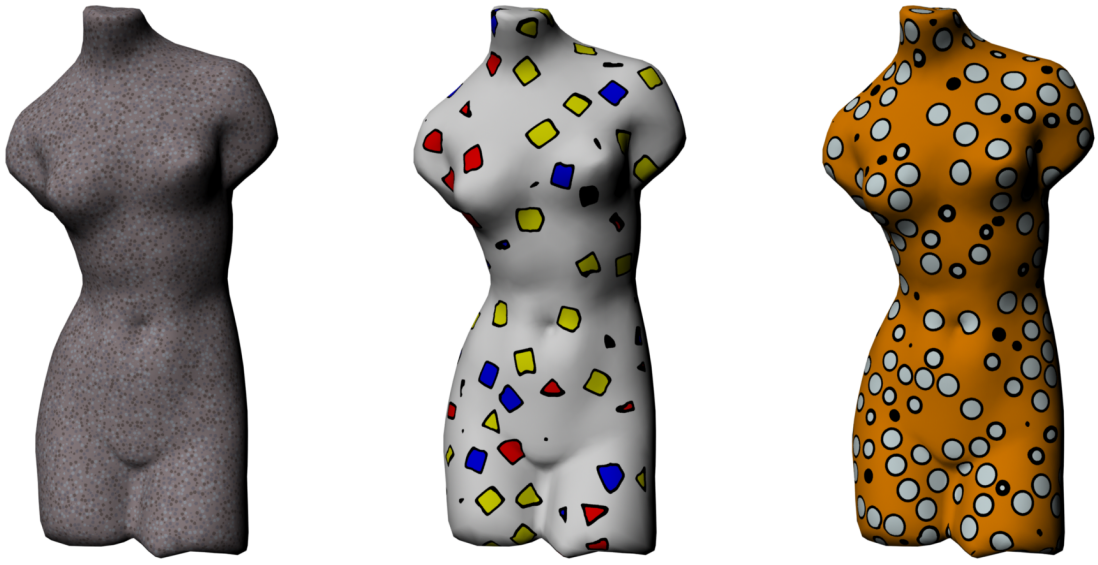
The procedural object distribution function does just that.

6.6.3 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 (see section 4.4). Figure 6.13 shows the outputs of the three-dimensional texture basis function.

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. Figure 6.15 shows how we integrated the procedural object distribution functions into a commercial rendering system.



(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.

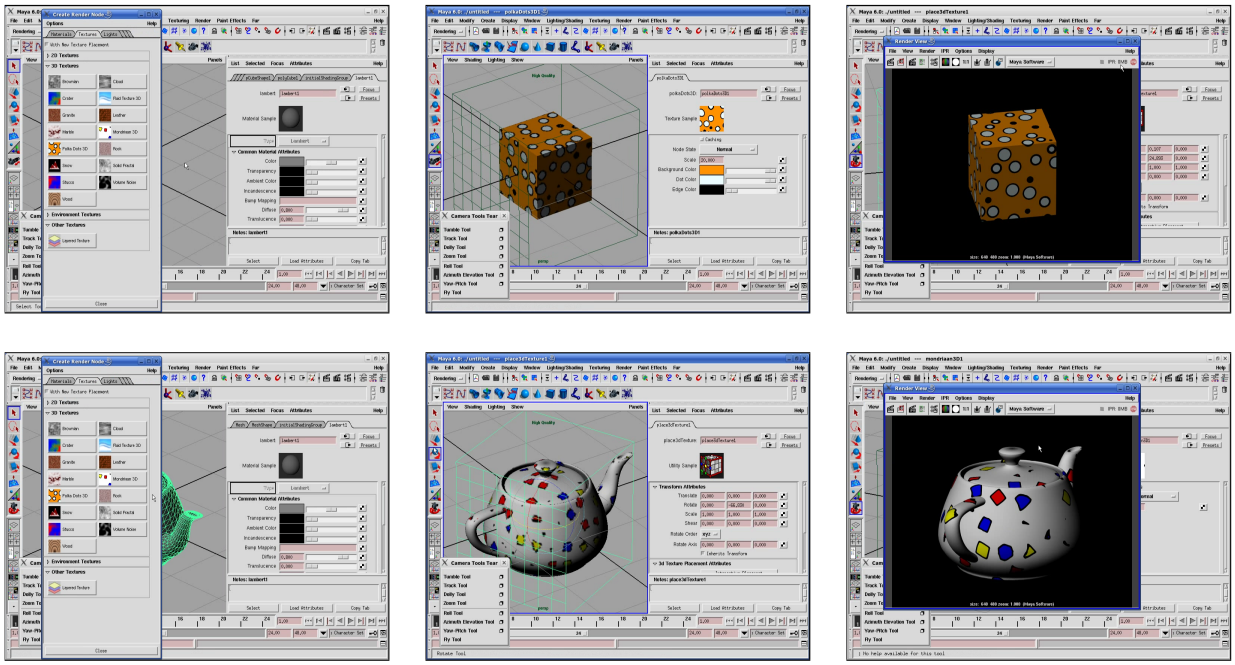


Figure 6.15: Integration of the object distribution texture basis function in a commercial rendering system.

6.7 Conclusion

In this chapter we have discussed applications of Poisson disk distributions in sampling, non-photorealistic rendering, scientific visualization and procedural modeling. We have also introduced a procedural object distribution function, a new texture basis function that extends the range of textures than can be generated procedurally. We have shown that Poisson disk distributions are a general tool in computer graphics, and that the tile-based methods for generating Poisson disk distributions introduced in the previous chapters can be used to improve existing applications but also enable new applications.

Chapter 7

Conclusion

7.1 Summary

A common problem in the field of computer graphics is the synthesis and storage of complex signals, such as point distributions or textures. In this work we have presented tile-based methods to solve this problem. Instead of synthesizing a complex signal directly, the signal is synthesized over a small set of tiles. Arbitrary large amounts of that signal can then be generated very efficiently simply by generating a stochastic tiling.

We have introduced corner tiles as an alternative for Wang tiles. In contrast with Wang tiles, corner tiles also constrain the four diagonally neighboring tiles, and are therefore not subject to the corner problem. We have revisited the most important applications of Wang tiles, and we have shown that corner tiles have substantial advantages for each of these applications.

We have presented direct stochastic tiling algorithms for Wang tiles and corner tiles. In contrast with scanline stochastic tiling algorithms, direct stochastic tiling algorithms are capable of evaluating a stochastic tiling locally, without explicitely constructing and storing the tiling up to that point. We have also presented long-period hash functions for direct stochastic tiling algorithms.

We have demonstrated tile-based methods for generating Poisson disk distributions and for synthesizing textures. Tile-based methods not only allow to efficiently generate Poisson disk distributions or synthesize textures, but also enable new applications such as tile-based texture synthesis and a procedural object distribution function. This new texture basis function allows to distribute procedural objects over a procedural background, using intuitive parameters such as the scale, size and orientation of the objects. We have also presented an overview of applications of tiled Poisson disk distributions.

The tile-based methods we have presented in this work are a valuable tool for computer graphics, and help to keep up with the continuously increasing demand for more complexity and realism in digitally synthesized images.

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.

Ball, W. W. R. *Mathematical recreations and essays*. MacMillan and Co., 1926.

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.

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

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

Cipra, B. Packing challenge mastered at last. *Science*, 281, 1998.

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.

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

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

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

Culik, II, K. An aperiodic set of 13 Wang tiles. *Discrete Mathematics*, 160(1-3):245–251, 1996.

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.

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.

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

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

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

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

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

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.

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

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

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

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. and Salesin, D. H. Escherization. In *Proceedings of ACM SIGGRAPH 2000*, pages 499–510. 2000.

Klassen, R. V. Filtered jitter. *Computer Graphics Forum*, 19(4):223–230, 2000.

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

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

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.

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

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

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.

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

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

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

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.

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

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

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

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

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.

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

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

Ng, T.-Y., Wen, C., Tan, T.-S., Zhang, X., and Kim, Y. J. Generating an ω-tile set for texture synthesis. In *Proceedings of Computer Graphics International 2005*, pages 177–184. 2005.

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.

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

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

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.

Perlin, K. Improving noise. *ACM Transactions on Graphics*, pages 681–682, 2002.

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

Pharr, M. and Humphreys, G. *Physically Based Rendering*. Morgan Kaufmann Publishers, Inc., San Fransisco, CA, USA, 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, USA, 1999.

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

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

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

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

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

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

Yellot, Jr., J. I. Spectral analysis of spatial sampling by photoreceptors: Topological disorder prevents aliasing. *Vision Research*, 22:1205–1210, 1982.

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

Part II

Periodic Tilings for Computer Graphics Applications

Chapter 3

Slides

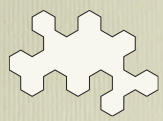
An introduction to tiling theory for computer graphics

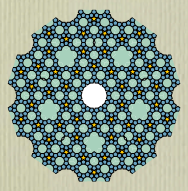
Craig S. Kaplan
University of Waterloo

SIGGRAPH 2008 course
13 August 2008

Tiling theory and graphics

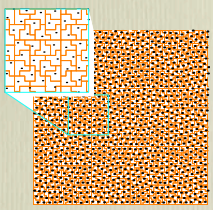
An elegant branch of mathematics





Art and design applications

Address traditional problems in computer graphics



Outline

Basic terminology

Periodic tilings

Isohedral tilings

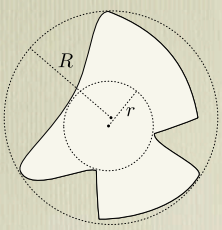
Nonperiodic tilings

Tiling

- Countable collection of closed topological disks called *tiles*
- Cover the plane with no gaps or overlaps
- Tiles are *uniformly bounded*
- Intersections are points or simple curves

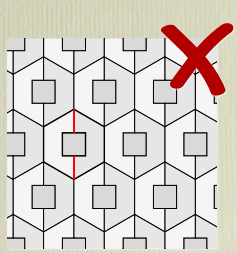
Tiling

- Countable collection of closed topological disks called *tiles*
- Cover the plane with no gaps or overlaps
- Tiles are *uniformly bounded*
- Intersections are points or simple curves



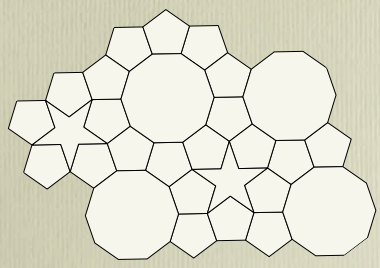
Tiling

- Countable collection of closed topological disks called *tiles*
- Cover the plane with no gaps or overlaps
- Tiles are *uniformly bounded*
- Intersections are points or simple curves



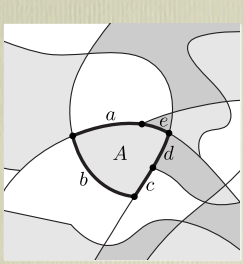
Patch

A finite set of tiles whose union is a topological disk



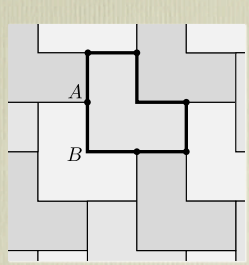
Features of a tiling

- *Tiling vertex*: location where three or more tiles intersect
- *Tiling edge*: arc connecting two tiling vertices



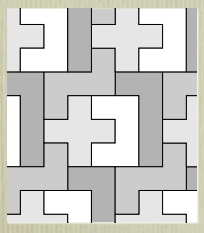
Features of a tiling

- Tiling features can be distinct from features of individual polygonal tiles



Congruence of tiles

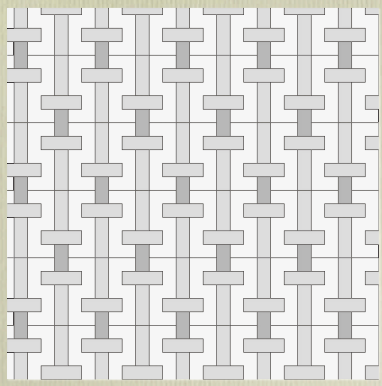
In a *k*-hedral tiling, every tile is congruent to one of *k* shapes called *prototiles*. (*monohedral*, *dihedral*)



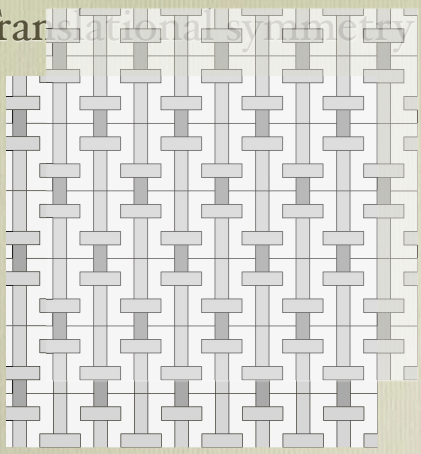
4-hedral tiling

Prototile set

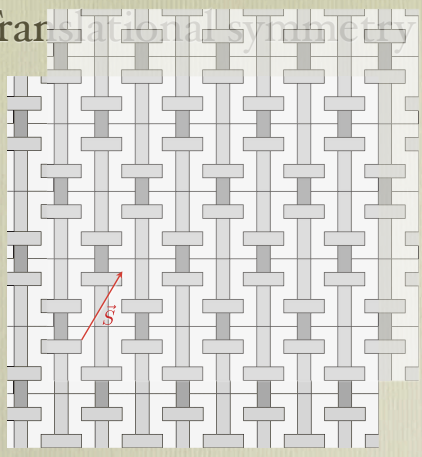
Translational symmetry



Translational symmetry

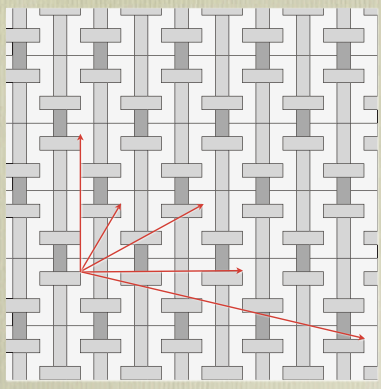


Translational symmetry



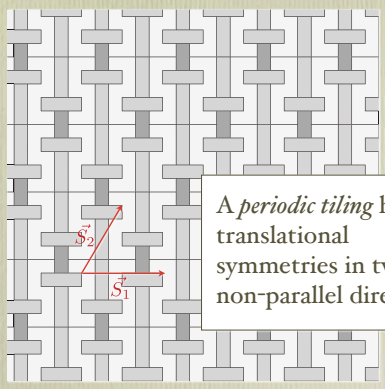
11

Translational symmetry



12

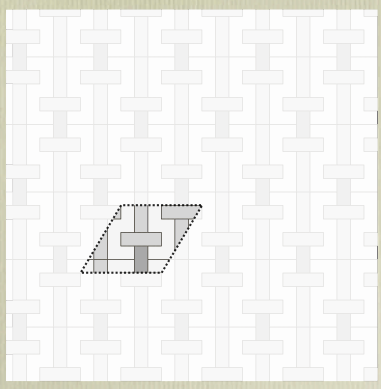
Periodic tiling



A *periodic tiling* has translational symmetries in two non-parallel directions.

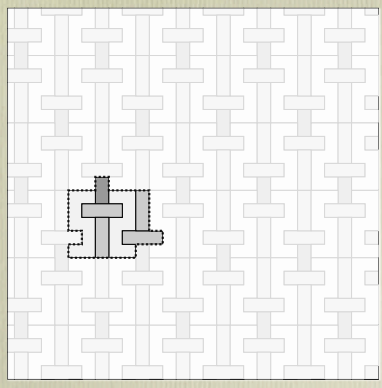
13

Translational unit



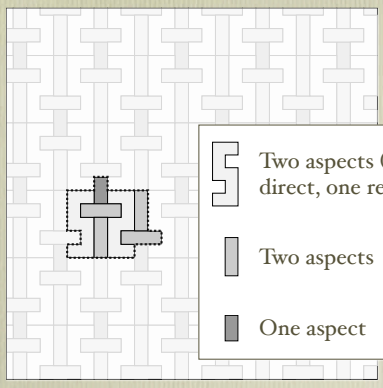
14

Translational unit



15

Aspects



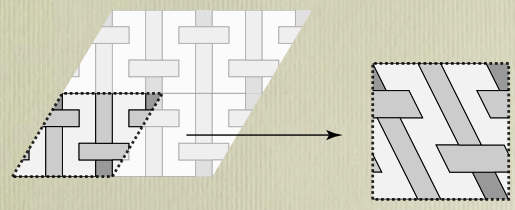
Two aspects (one direct, one reflected)

Two aspects

One aspect

16

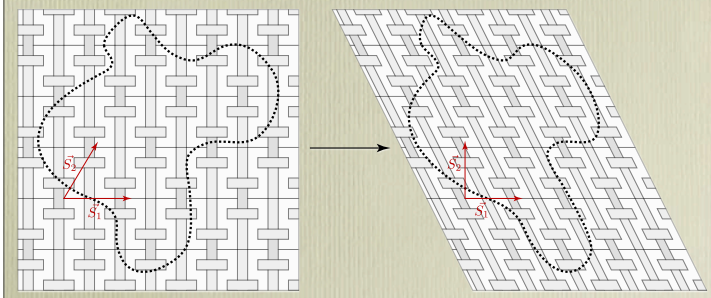
Replication



Replication as texture mapping

17

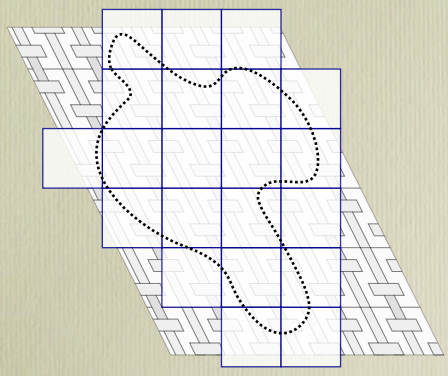
Replication



Replication by drawing tiles

18

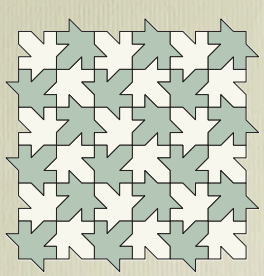
Replication



19

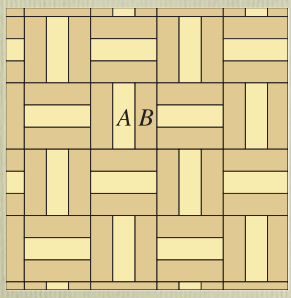
Isohedral tilings

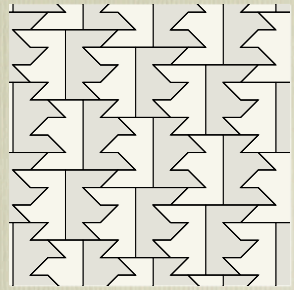
- Monohedral, periodic tiling with a lot of structure
- Convenient, efficient representation
- Flexible and expressive



20

Transitivity classes





21

Isohedral tiling

An *isohedral tiling* is one in which all tiles belong to a single transitivity class.

22

Isohedral tiling

An *isohedral tiling* is one in which all tiles belong to a single transitivity class.

...Any congruence between two tiles leaves the tiling as a whole invariant.

Isohedral tiling

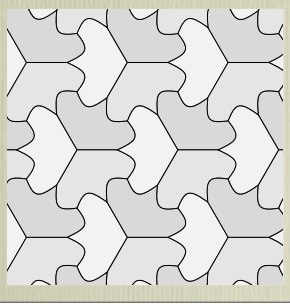
An *isohedral tiling* is one in which all tiles belong to a single transitivity class.

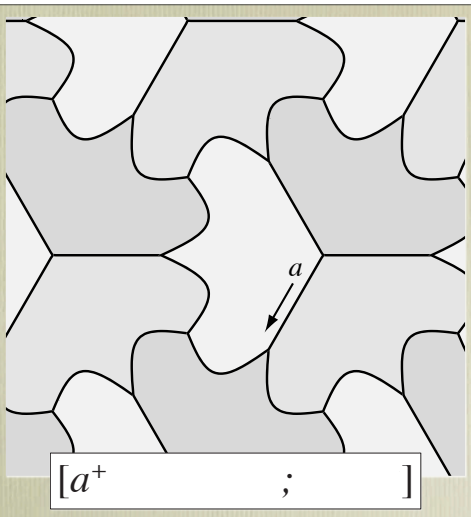
...Any congruence between two tiles leaves the tiling as a whole invariant.

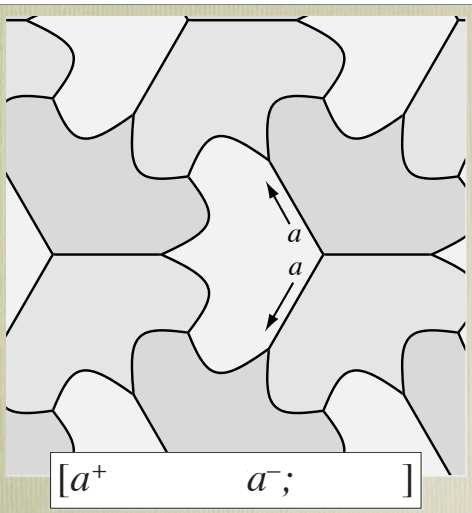
...The tiles are indistinguishable from each other (ignoring colour).

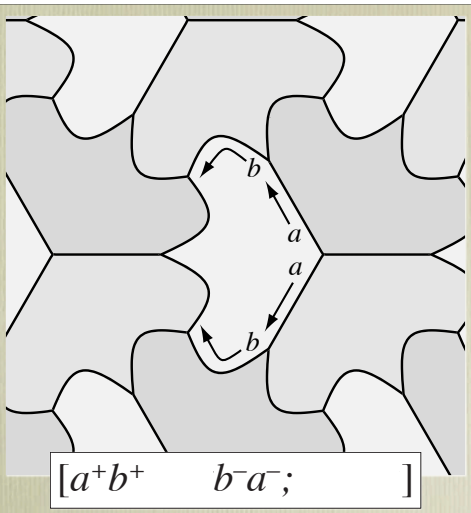
Incidence symbols

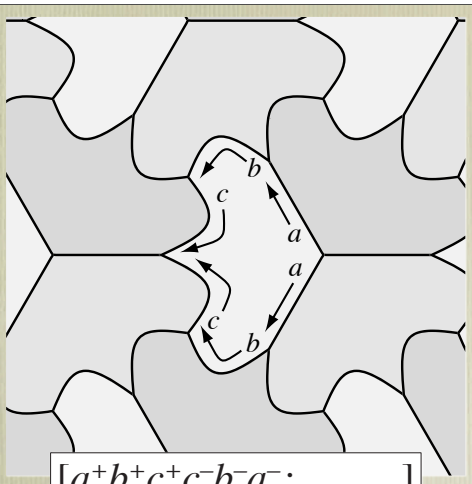
The structure of an isohedral tiling can be determined from one tile's relationship to its neighbours.



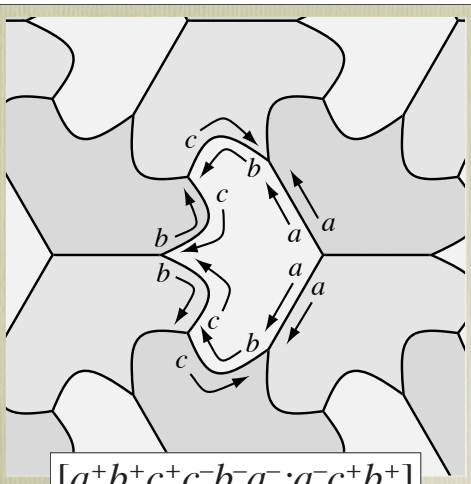








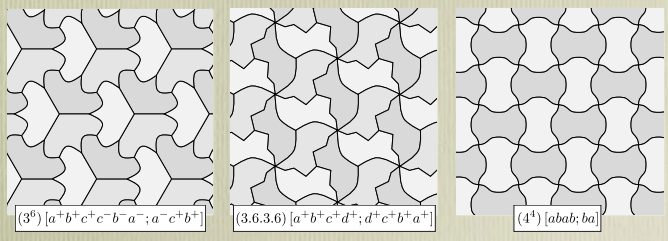
27



28

Isohedral types

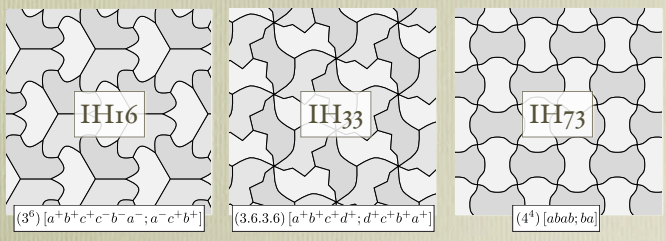
There are 93 isohedral tiling types



29

Isohedral types

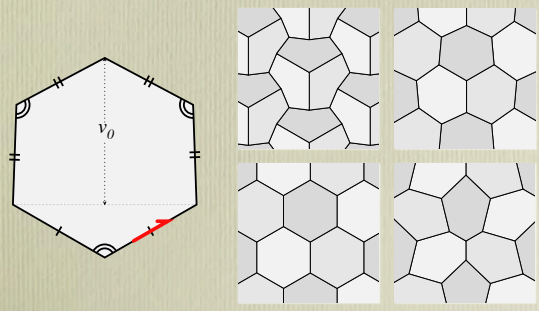
There are 93 isohedral tiling types



29

Parameterizing isohedral tilings

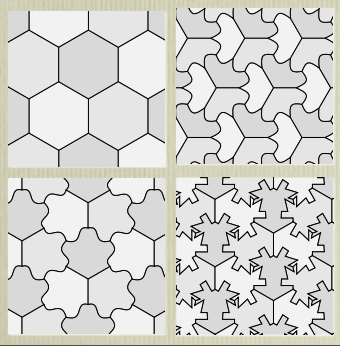
The shapes belonging to each isohedral type can be parameterized via tiling vertices and edge shapes.



30

Parameterizing isohedral tilings

The shapes belonging to each isohedral type can be parameterized via tiling vertices and edge shapes.



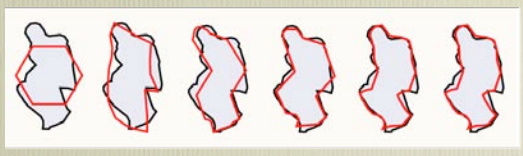
31

Escherization

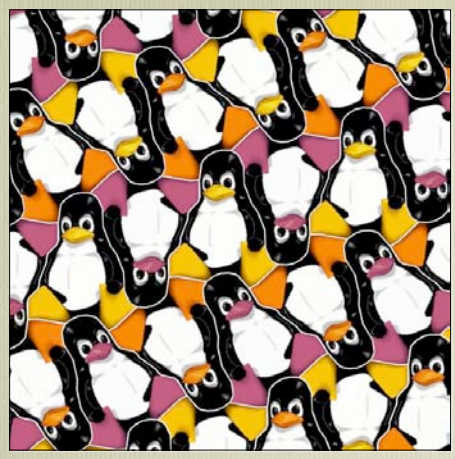
Given a user-supplied "goal shape" *S*, search the parameterized tiling types automatically for an isohedral prototile that resembles *S*.

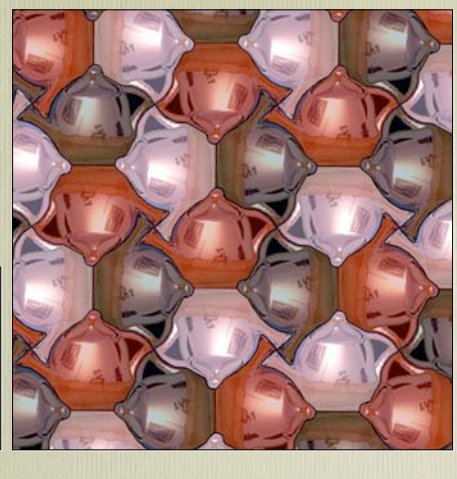
Goal shape

IH03



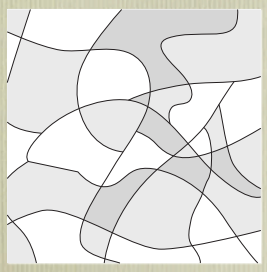
Escherization

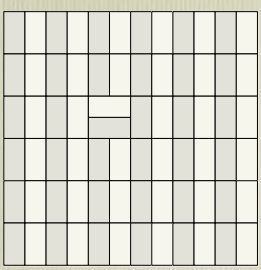




Nonperiodic tilings

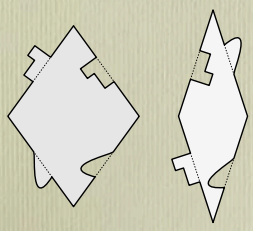
A *nonperiodic tiling* is one without translational symmetries.

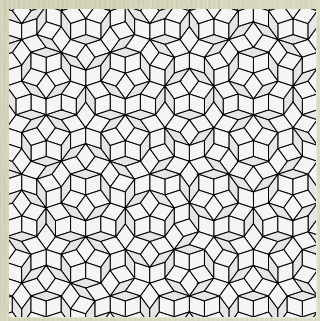




Nonperiodic tilings

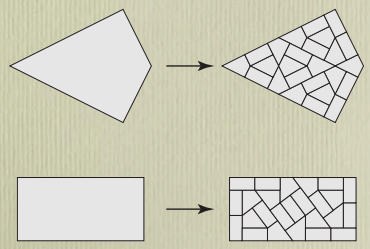
An *aperiodic prototile set* is one that admits only nonperiodic tilings.

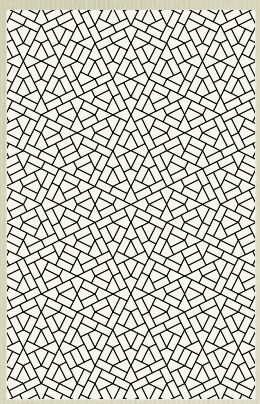




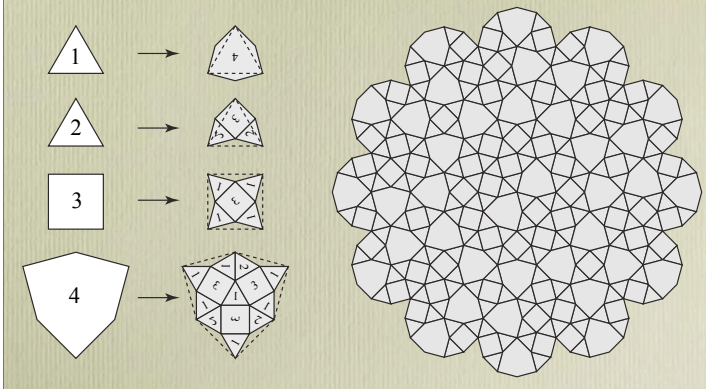
38

Substitution tilings

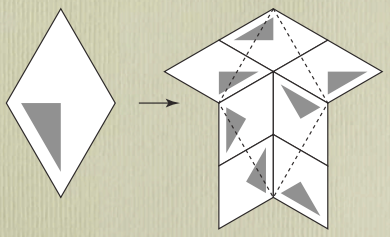


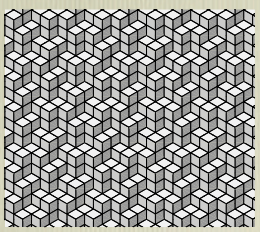


39



40

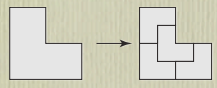


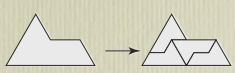


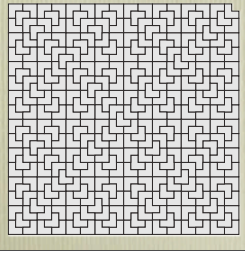
41

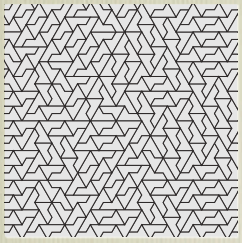
Rep-tiles

A *rep-tile* is a substitution system based on a single prototile.



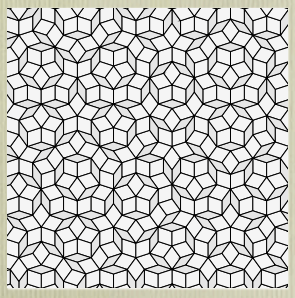


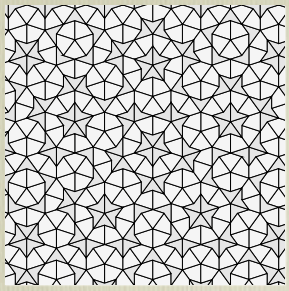




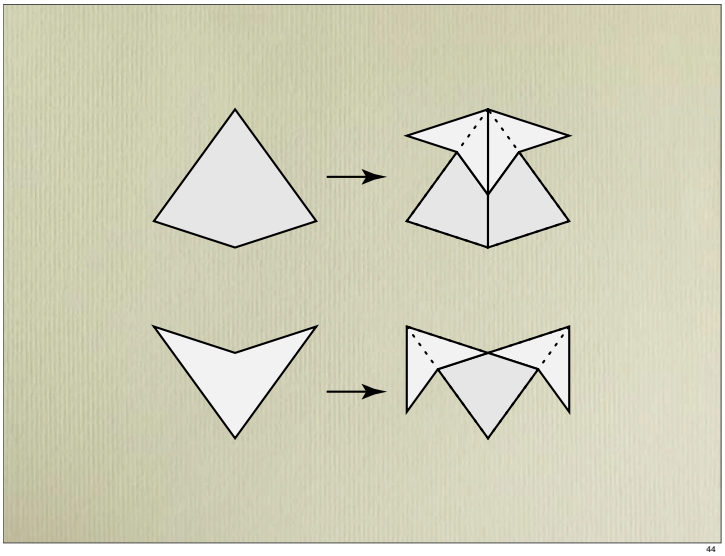
42

Penrose tilings

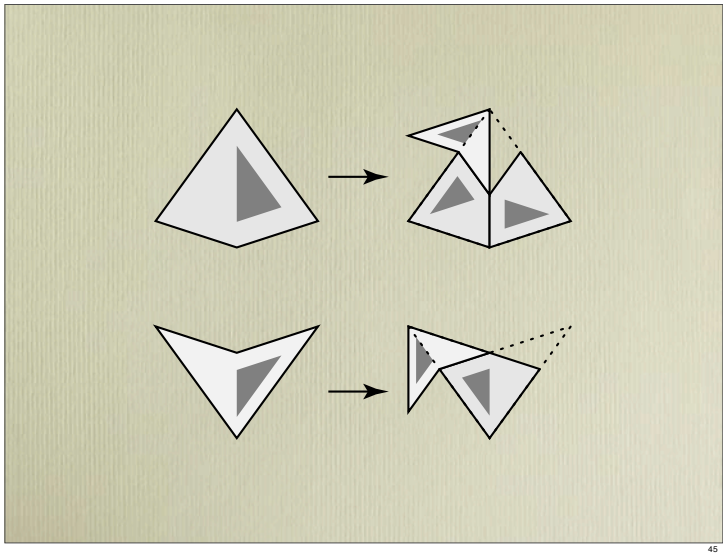




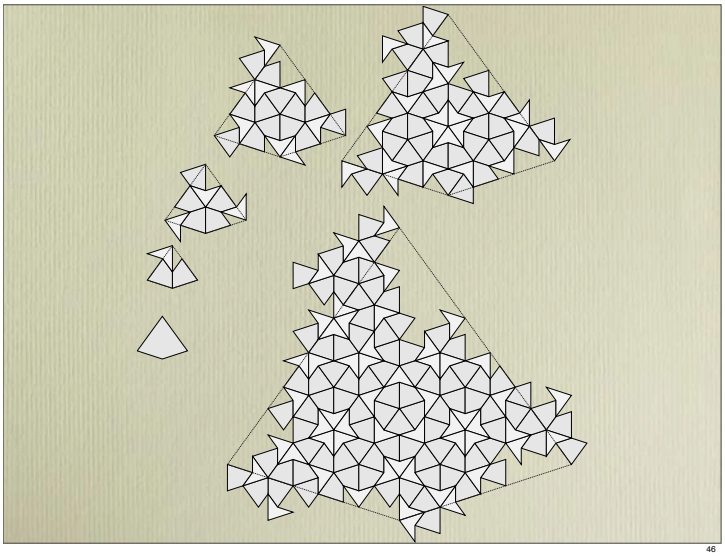
43



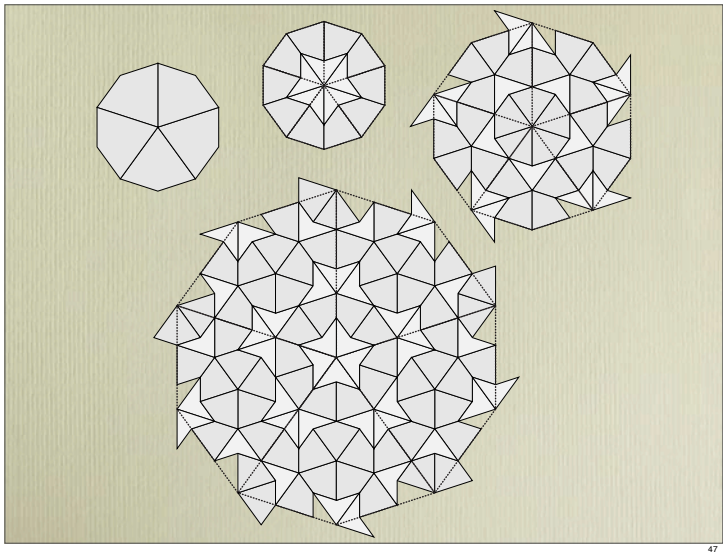
44



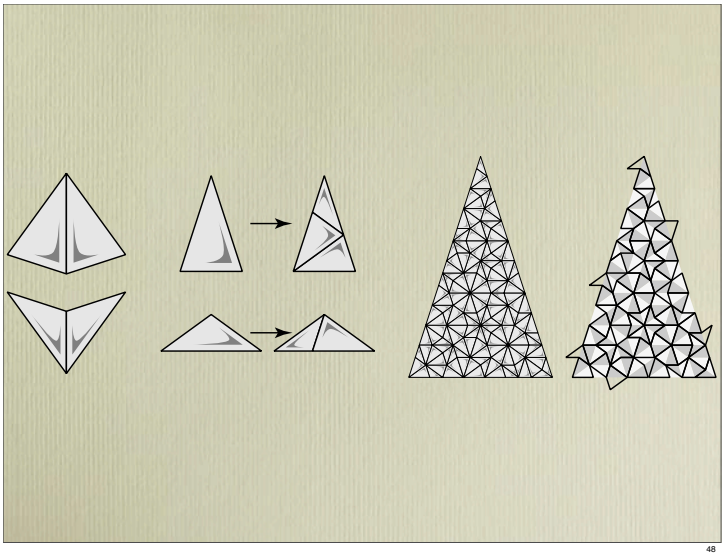
45



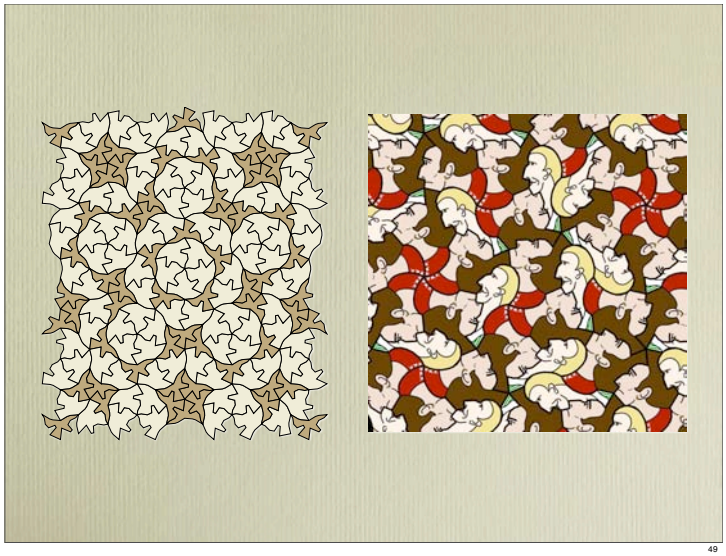
46



47



48



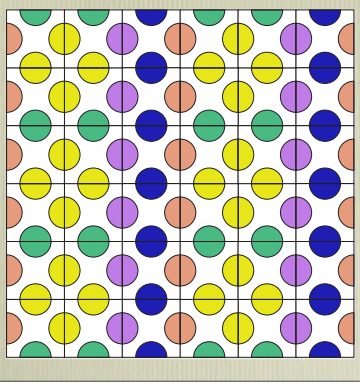
49

Wang tiles

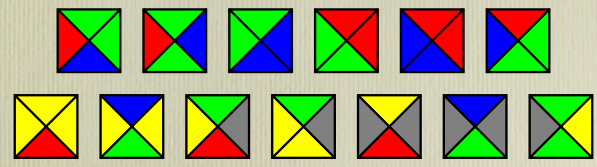
- *Wang tiles* are square tiles with marked edges.
- Tiles are placed by translation only.
- Markings must agree across tiling edges.

50

51

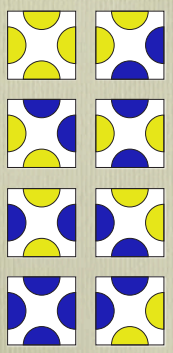


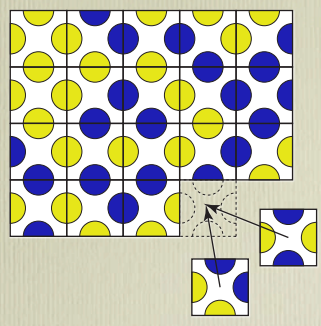
51



52

"Universal" Wang tiles





53

Thank you

54

Chapter 4

Syllabus

An Introduction to Tiling Theory for Computer Graphics

Craig S. Kaplan
David R. Cheriton School of Computer Science
University of Waterloo

May 28, 2008

1 Introduction

Tiling theory, the study of shapes that cover the plane with no gaps or overlaps, is an elegant and aesthetic branch of mathematics. Tilings themselves are an ancient artform; countless historical and contemporary examples motivate us to explain mathematically what was devised by intuition alone. The area is full of unsolved problems that are simple to state but complex and mysterious when considered carefully [7].

Elements of tiling theory have already found exciting applications in computer graphics, from art and ornamental design to sampling and texture synthesis. In computer graphics we might be able to benefit from many aspects of tiling theory, either by borrowing its theorems to solve problems, or by developing algorithms for generating novel attractive tilings. However, there is no reference that summarizes the fundamentals of tiling theory in a way suitable for computer graphics practitioners. The definitive reference on the subject, Grünbaum and Shephard's *Tilings and Patterns* [16], is wonderful but sadly out of print. Furthermore, its mathematical focus omits potentially important algorithmic details that would arise in any practical implementation.

The goal of this document is to present the basics of tiling theory in an accessible way, including additional details of interest to those seeking to use tilings in computer graphics applications. In some places, I have taken the license to include a few mathematical details where the math is especially worthwhile or to clear up common confusions.

This tutorial is necessarily just an introduction. Motivated readers should still consult *Tilings and Patterns* for most of the mathematical infrastructure of tiling theory. Readers familiar with tiling theory will note that I have deliberately omitted a formal discussion of symmetry theory. Although the two subjects frequently overlap, a considerable amount of tiling theory can be presented with only the rudiments of symmetry theory. I have also left out a discussion of related results in non-Euclidean geometry, which is a beautiful subject but arguably of interest only to a small minority in computer graphics.

Most of the text of this document is derived from my doctoral dissertation [20] (available online at `http://www.cgl.uwaterloo.ca/~csk/phd/`), particularly from Sections 2.3–2.5 and 4.2–4.6. Some additional material was taken from "Computer graphics, geometry, and ornamental design", a graduate course in Computer Science that I taught at the University of Waterloo in 2006 (`http://www.cgl.uwaterloo.ca/~csk/cs798/winter2006/`).

2 Tiling basics

In their indispensible book *Tilings and Patterns* [16], Grünbaum and Shephard develop an extensive theory of tilings of the Euclidean plane. They begin from first principles with a nearly universally inclusive definition of tilings, one that permits so many pathological cases that the resulting objects cannot be meaningfully studied. They then proceed to layer restrictions upon the basic definition, creating ever smaller families of tilings that yield to more and more precise analysis and classification.

Many of the early pathological cases discussed by Grünbaum and Shephard would not be amenable to practical applications in computer graphics. For our purposes, we can jump directly to a more restrictive definition, corresponding to their notion of "normal tiling" [16, Section 3.2].

Definition (Tiling) *A tiling is a countable collection $\mathcal{T}$ of tiles $\{T_1, T_2, \ldots\}$, such that:*

1. *Every tile is a closed topological disk.*
2. *Every point in the plane is contained in at least one tile.*
3. *The intersection of every two tiles is empty, a point, or a simple curve with endpoints.*
4. *The tiles are uniformly bounded; that is, there exist $u, U > 0$ such that every tile contains a closed ball of radius u and is contained in a closed ball of radius U.*

The most natural property associated with tilings, that they cover the plane with no gaps and no overlaps, is handled by Conditions 2 and 3. Conditions 1 and 4 ensure that the tiles are reasonably well behaved entities that do not have exotic topological properties or become pathological at infinity.

Observe that Condition 3 does more than prevent tiles from overlapping. It also forbids tilings like the one shown in Figure 1 from arising, where the boundary between two tiles is disconnected. However, in return we can identify well-defined *tiling vertices*, points that are the intersection of three or more tiles, and *tiling edges*, the paths that join tiling vertices. The set of tiling vertices and edges can be seen as capturing all the topological information about the tiling (information about a tile's neighbours, the tiles adjacent to a vertex, and so on). We may also use the term *tiling polygon* to refer to the (not necessarily simple) polygon joining the tiling vertices that lie on a single tile. These terms are illustrated in Figure 2.

Condition 3 might seem overly restrictive, in that a tiling like that of Figure 1 could arise naturally in computer graphics applications. Indeed, Grünbaum later indicated that the condition was included in the definition of normal tilings because he and Shephard felt they would have needed to assume it in many later parts of their theory (due to its simplifying effect on topological structure). An alternate development could leave Condition 3 out of normality and introduce it as needed.

In any practical application, we will necessarily operate on finite collections of tiles. A *patch* of tiles is a finite set of tiles whose union is a topological disk. That is, a patch is a contiguous block of tiles with no internal holes. Of course, no drawing of a tiling shows more than a finite patch; we make the the implicit assumption that the context will provide the means of extending the patch to cover the plane.

3 Tilings with congruent tiles

In many of the tilings we see every day on walls and streets, the tiles all have the same shape. If every tile in a tiling is congruent to some shape T (where the congruence may incorporate a reflection), we say that the tiling is *monohedral*, and that T is the *prototile* of the tiling. More generally, a *k-hedral* tiling is one in which every tile is congruent to one of k different prototiles. We also use the terms *dihedral*, *trihedral* and

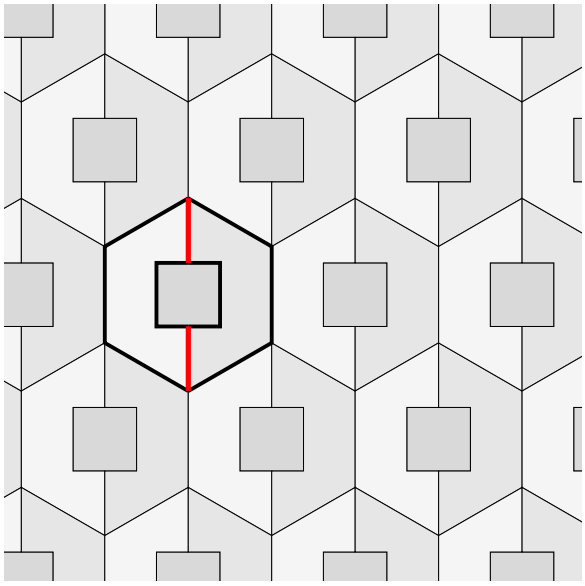


Figure 1: A set of shapes which do not constitute a tiling according to the definition set out in the text, because some tiles intersect in multiple disjoint curves. The intersection between the two tiles outlined in bold is shown in red; it consists of two disjoint line segments.

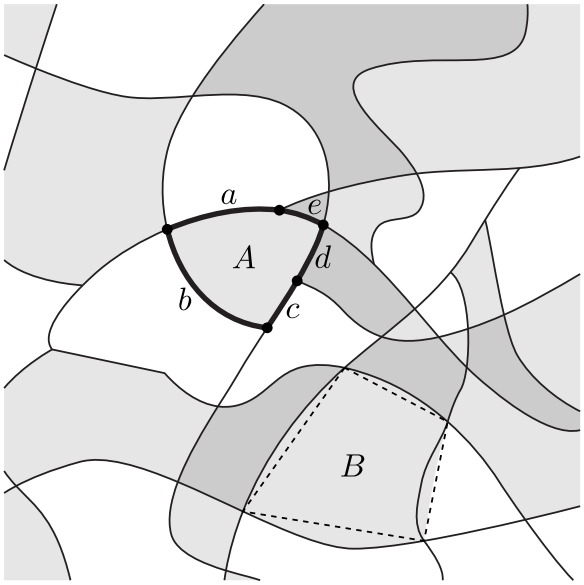


Figure 2: Basic topological features of a tiling. The tile labeled A is outlined in bold and its tiling vertices are marked with dots. Each of the tiling edges on its boundary is also labeled. Tile B's tiling polygon is shown with dashed lines.

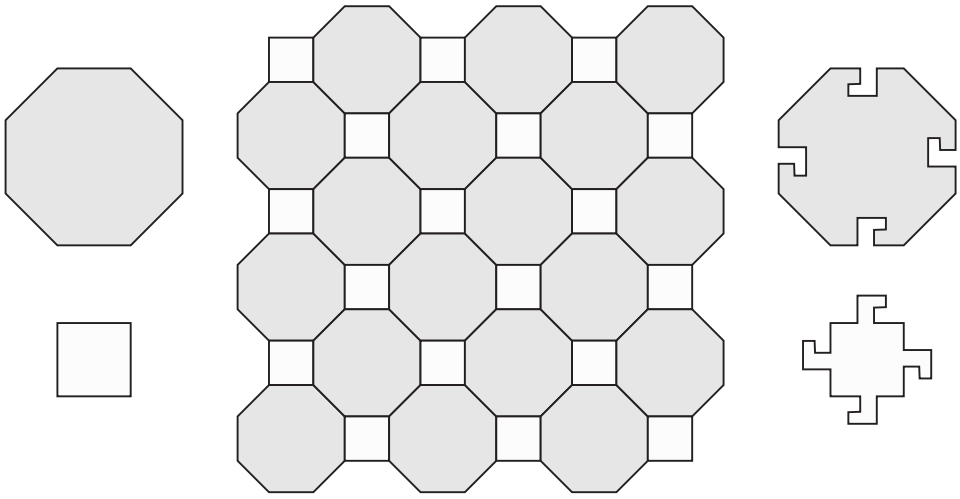


Figure 3: A demonstration of matching conditions. A prototile set consisting of a square and a regular octagon admits the tiling shown in the centre, but many other tilings as well, including the regular tiling by squares. The prototiles on the right are modified with simple geometric matching conditions. They admit only (the obvious analogue of) the square-octagon tiling.

multihedral for the cases $k = 2$, $k = 3$ and $k > 1$, respectively. If $\mathcal{P}$ is a set of prototiles, any tiling that can be formed exclusively from congruent copies of members of $\mathcal{P}$ is said to be *admitted* by $\mathcal{P}$.

Note that a tiling admitted by a set of prototiles need not use all of them (and thus k-hedrality is a property of a tiling, and not of a prototile set). In some cases where we wish to force every member of a prototile set to appear at least once in a tiling, the prototiles might require some modification. These modifications typically take the form of *matching conditions*: additional constraints placed on the vertices or edges of prototiles that restrict which tiles may be placed next to them. For example, prototile edges might be given symbolic labels, together with a rule indicating which pairs of labels are compatible. We then ask that in any tiling by these prototiles, two tiles that share an edge have compatible labels on that edge. Frequently, matching conditions of this sort can also be encoded geometrically as indentations and protusions on prototile edges. Once the matching conditions have been used to enforce a particular arrangement of tiles, they are often omitted from the final drawing of the tiling.

Given a finite set of shapes, we might wonder whether they are in fact a prototile set—that is, do the given shapes admit any tilings at all? In full generality this problem is known to be formally undecidable (see Section 8.2), and so we must speculate that it could be arbitrarily difficult to prove or disprove the fact for any given set of shapes. However, we do have one useful tool at our disposal [16, Section 3.8]:

> **Theorem (The Extension Theorem)** *Let $\mathcal{P}$ be a finite set of shapes, each a closed topological disk. If, for any $r > 0$, there exists a patch of tiles from $\mathcal{P}$ that contains a disk of radius r, then $\mathcal{P}$ admits a tiling of the plane.*

The Extension Theorem offers us a kind of generic limiting process: if we can construct arbirarily large finite patches of tiles, then we can go off to infinity in all directions. Naturally, any one prototile set might come equipped with a specialized argument that yields the tilings it admits; this theorem can take the

place of many of those arguments. For example, it can almost always be invoked to justify the construction of substitution tilings (see Section 8.1).

4 Symmetry

In the previous section, I used the term "congruence" to refer informally to a transformation in the plane that does not distort shape or size. In Euclidean geometry, we can define congruences as *rigid motions*: transformations of the plane that preserve distance. Every Euclidean rigid motion belongs to one of five categories: the identity (or "do nothing") transformation, translations, rotations, reflections, and glide reflections (a reflection followed by a translation parallel to the axis of reflection).

For every tiling $\mathcal{T}$ there exists a set $\Sigma(\mathcal{T})$ consisting of those rigid motions for which $\mathcal{T}$ is indistinguishable from its image under the rigid motion. If we think of the tiling as drawn on a piece of paper with another copy superimposed on a transparent overlay, any movement of the overlay that causes the two drawings to line up belongs in this set. The members of $\Sigma(\mathcal{T})$ are known as the *symmetries* of $\mathcal{T}$. Because a tiling's symmetries have a natural group structure under composition of rigid motions, $\Sigma(\mathcal{T})$ is known as the *symmetry group* of $\mathcal{T}$.

Symmetry theory is a rich and elegant branch of mathematics that overlaps significantly with tiling theory. However, for the purposes of these notes I will avoid an extensive discussion of the subject. For the most part, the topics discussed here can be understood with only basic, intuitive knowledge of symmetry theory. Interested readers should consult the introductory texts by Farmer [11] and Weyl [32], references on symmetry in art and ornament [29, 31], or the new treatment by Conway et al. [5].

5 Periodic tilings

We are particularly interested in the case where a tiling's symmetry group contains two non-parallel translations, which we will represent by vectors $\vec{S_1}$ and $\vec{S_2}$ (which are taken to be as short as possible). In such a case, any integer linear combination $n_1\vec{S_1} + n_2\vec{S_2}$ represents a translation that brings the tiling into coincidence with itself. We refer to such a tiling as *periodic*. Every periodic tiling belongs to one of 17 distinct symmetry groups known as *wallpaper groups*.

Periodic tilings are very well-behaved, and easy to represent and render in computer graphics applications. Let $\mathcal{T}$ be a periodic tiling with minimal translational symmetries $\vec{S_1}$ and $\vec{S_2}$. Because $\mathcal{T}$ repeats at every integer linear combination of the two translations, we can choose a maximal subset of the plane containing no redundant information, and draw the tiling by stamping out copies of this subset. A subset of this kind that is also a topological disk will be called a *translational unit* of the tiling. A periodic tiling can be represented using any translational unit, together with $\vec{S_1}$ and $\vec{S_2}$.

A periodic tiling has many possible translational units. The intersection of the tiling with any parallelogram with sides $\vec{S_1}$ and $\vec{S_2}$ is an example. Note that there is always an affine transformation that maps this parallelogram to a square. This transformation can be applied to the drawing inside the translational unit to allow it to fit comfortably inside a square texture in graphics memory. By applying the same transformation to texture coordinates during lookup, it becomes easy to render arbitrary periodic tilings via texture mapping.

It is also always possible to identify finite patches of tiles that act as translational units (meaning that every periodic tiling is k-hedral for some finite k). These patches will each have the same overall area as the parallelograms above. Furthermore, each prototile can occur in only finitely many orientations and reflected

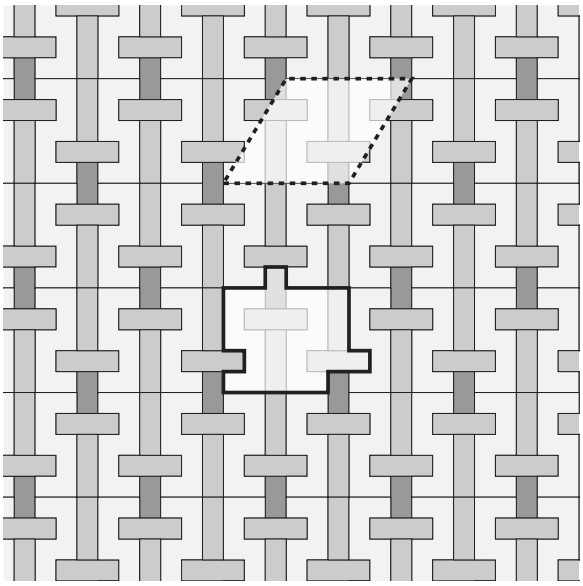


Figure 4: A trihedral periodic tiling. An example of a parallelogram-shaped translational unit is outlined with a dashed line. Beneath it, an alternate translational unit made from a patch of tiles is shown. The S-shaped tile occurs in one direct and one reflected aspect, the 1×3 rectangle occurs in two aspects, and the 1×2 rectangle occurs in a single aspect.

orientations. We refer to these orientations collectively as the prototile's *aspects*, and distinguish the *direct aspects* from the *reflected aspects* when necessary. Figure 4 shows a periodic tiling for which the various prototiles occur in different numbers of aspects.

The patch-based representation of a periodic tiling is useful when tiles must be drawn explicitly via a subroutine rather than sampled from a texture. However, we must then address the question of replication: given a region R of the plane, for which integer pairs (n_1, n_2) should we draw the tiles translated by $n_1\vec{S_1} + n_2\vec{S_2}$ in order to cover R completely with tiles? We can arrive at a suitable approximation of the solution by re-representing R in a basis formed by $\vec{S_1}$ and $\vec{S_2}$. In this basis translational units are unit squares and the units to draw are precisely those squares that intersect R. In other words, we need only rasterize the representation of R in this basis; the coordinates of the interior pixels are precisely the pairs (n_1, n_2) above. See Figure 5 for a visualization. This method is only truly correct when the patch is itself a parallelogram; in practice, it is usually necessary to add one or more rings of additional translational units around the rasterized pixels of R.

6 Tilings by polygons

In most cases we will be interested in tilings by polygons. All of the important features of tilings we will encounter in this document are adequately explained in terms of polygonal tilings. Even when we wish to render tiles with curves edges, those edges will likely be represented as piecewise-linear paths at some stage in the rendering pipeline.

In a polygonal tiling there can be some confusion between the tiling vertices and edges as described

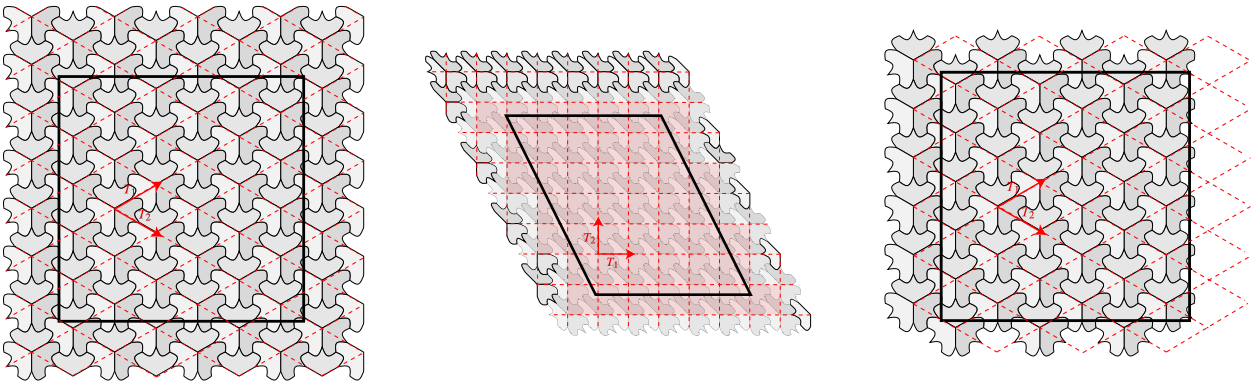


Figure 5: The replication algorithm for periodic tilings. The left image shows the tiling to be replicated, with a superimposed black square representing the desired viewing region. The red parallelograms delineate translational units of the tiling, based on vectors $\vec{S_1}$ and $\vec{S_2}$. In the middle image, the whole diagram is shown in a coordinate system where $\vec{S_1}$ and $\vec{S_2}$ are an orthonormal basis. In this coordinate system, translational units are lattice squares, and it is easy to choose the squares that overlap the viewing region. The chosen units are drawn in the untransformed image on the right. This algorithm can leave part of the viewing region unfilled (as seen on the right).

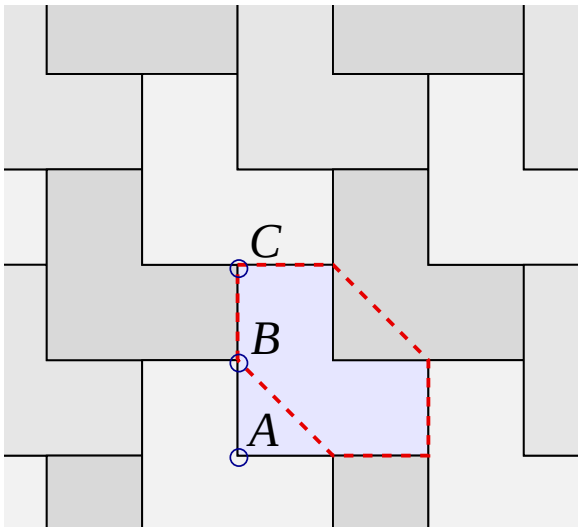


Figure 6: The features of a polygonal tiling. For the tile highlighted in blue, A is a shape vertex but not a tiling vertex, B is a tiling vertex but not a shape vertex, and C is both a tiling vertex and a shape vertex. Because the tiling vertices and shape vertices do not coincide, the tiling is not edge-to-edge. The tiling polygon is shown as a dashed red outline.

in Section 2 above and the vertices and edges of individual polygons. To avoid confusion, we refer to the latter features when necessary as *shape vertices* and *shape edges*. Shape vertices and edges are properties of tiles in isolation; tiling vertices and edges are topological properties of the assembled tiling. Although the features of the tiling occupy the same positions as the features of the tiles, they may break down differently, as shown in Figure 6. When the two sets of features coincide (that is, when the tiling vertices are precisely the shape vertices), the tiling is called *edge-to-edge*.

6.1 Regular and uniform tilings

A *regular tiling* is an edge-to-edge monohedral tiling of the plane by congruent regular polygons. In the Euclidean plane, an easy calculation shows that the only regular tilings are the familiar ones by squares, equilateral triangles, and regular hexagons.

We may then ask about edge-to-edge tilings of the plane using two or more different regular polygons as prototiles. In general, we can say very little about an arbitrary tiling of this type. To get more predictable behaviour, we further require that the vertices be indistinguishable, in the sense that any vertex can be mapped onto any other via a rigid motion that is also a symmetry of the tiling as a whole. Given such a restriction, we can describe the tiling using a *vertex symbol*. A vertex symbol is a sequence $p_1.p_2 \dots p_n$ that enumerates, in order, the regular polygons encountered around every vertex in the tiling. The tilings that can be described using vertex symbols in this way are called *uniform tilings*.

In the Euclidean plane, we can enumerate all legal vertex symbols (in the sense that the interior angles around the vertex sum to 360 degrees) and determine which of them can be extended to legal uniform tilings of the plane [16, Section 2.1]. The result is a set of eleven distinct periodic tilings, also known as the *Archimedean tilings*. We identify them by placing their vertex symbols in parentheses. Among the uniform tilings the regular tilings are the ones whose vertex symbols are of the form p^q: (4^4) for squares, (3^6) for equilateral triangles, and (6^3) for regular hexagons. (In a vertex symbol, we abbreviate blocks of repeated values using exponentiation.) Figure 7 shows the eleven Archimedean tilings. These tilings are useful for decorative and artistic purposes, and lead naturally to further exploration of tilings formed exclusively from regular or star-shaped polygons.

6.2 Laves tilings

Every Archimedean tiling has a well-defined geometric dual, obtained by replacing every n-sided tile by an n-valent vertex and vice versa. These dual tilings are monohedral and edge-to-edge, and have the property that every tiling vertex is regular: the edges leaving the vertex are evenly spaced around it. They are called the *Laves tilings*, and they are given labels analogous to their Archimedean duals. The Laves tilings are depicted in Figure 8. They will prove useful in the next section, where they serve as a set of "defaults" upon which to describe the more elaborate structure of the isohedral tilings.

7 Isohedral tilings

Let $\mathcal{T}$ be a monohedral tiling. For two tiles T_1 and T_2 in $\mathcal{T}$, there will be at least one rigid motion of the plane that maps T_1 to T_2. A special case occurs when the rigid motion is also a symmetry of the tiling. In this case, when T_1 and T_2 are brought into correspondence, the rest of the tiling will map onto itself as well. We then say that the two tiles are *transitively equivalent*.

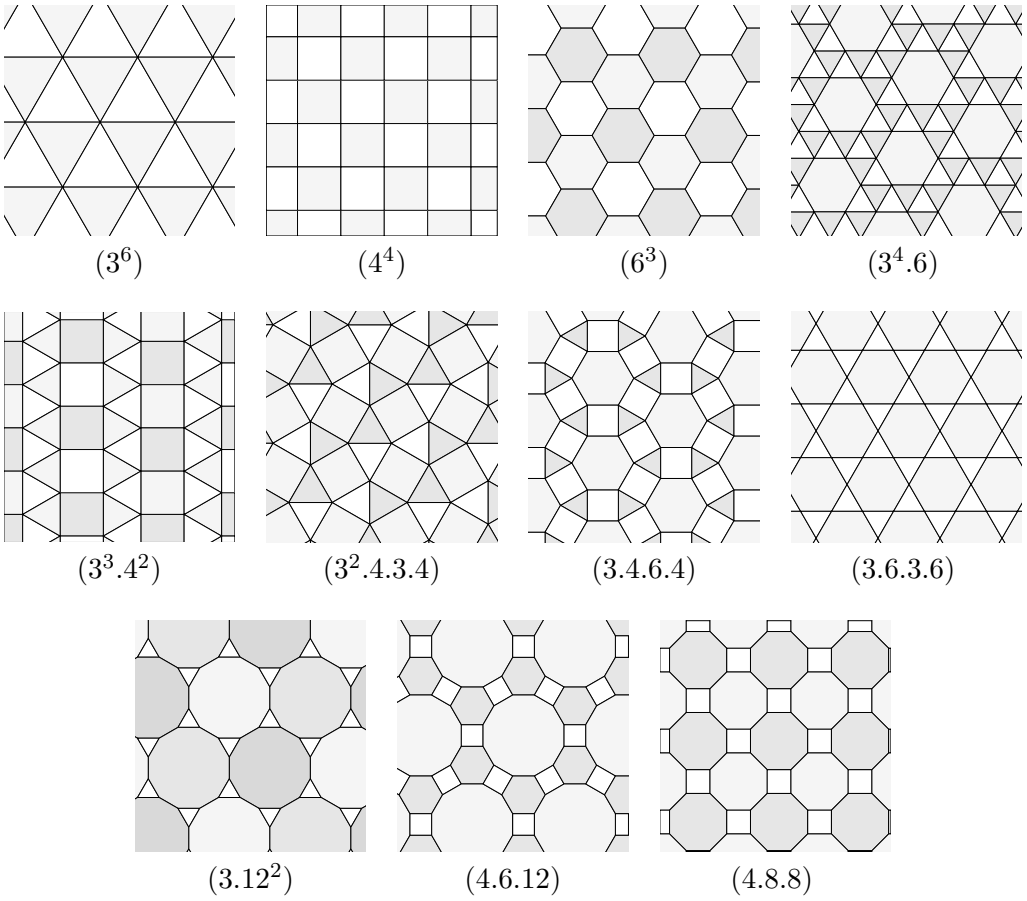


Figure 7: The eleven uniform Euclidean tilings, also known as Archimedean tilings. The tiling $(3^4.6)$ occurs in left-handed and right-handed forms.

Transitive equivalence is an equivalence relation that partitions the tiles into *transitivity classes*. When a tiling has only one transitivity class, we call the tiling *isohedral*. More generally, a k-isohedral tiling has k transitivity classes (and may be m-hedral for any $m \leq k$). An isohedral tiling is one in which a single prototile can cover the entire plane through repeated application of rigid motions from the tiling's symmetry group. In an isohedral tiling, there is effectively no way to tell any tile from any other, since the "local views" outward from any two tiles are identical.

Two tiles in the same transitivity class must obviously be congruent, but the converse need not be true. Figure 9 shows a monohedral tiling with two transitivity classes. The two classes of tiles can be distinguished by the arrangement of a tile's neighbours around it. This example also demonstrates that being k-isohedral is a property of a tiling, not of its prototiles. The underlying shape used as a prototile in Figure 9 also admits an isohedral tiling. We therefore define a *k-anisohedral* set of prototiles as a set of shapes that admits a k-isohedral tiling, but no m-isohedral tiling for $m < k$. This new definition is truly tied to the prototiles themselves, and places a lower bound on the complexity of the tilings they admit. In 1900, Hilbert seemed to take it for granted that no anisohedral prototile can exist [16, Section 9.6]. In 1935, however, Heesch demonstrated an anisohedral prototile [17], reproduced in Figure 10. Since then

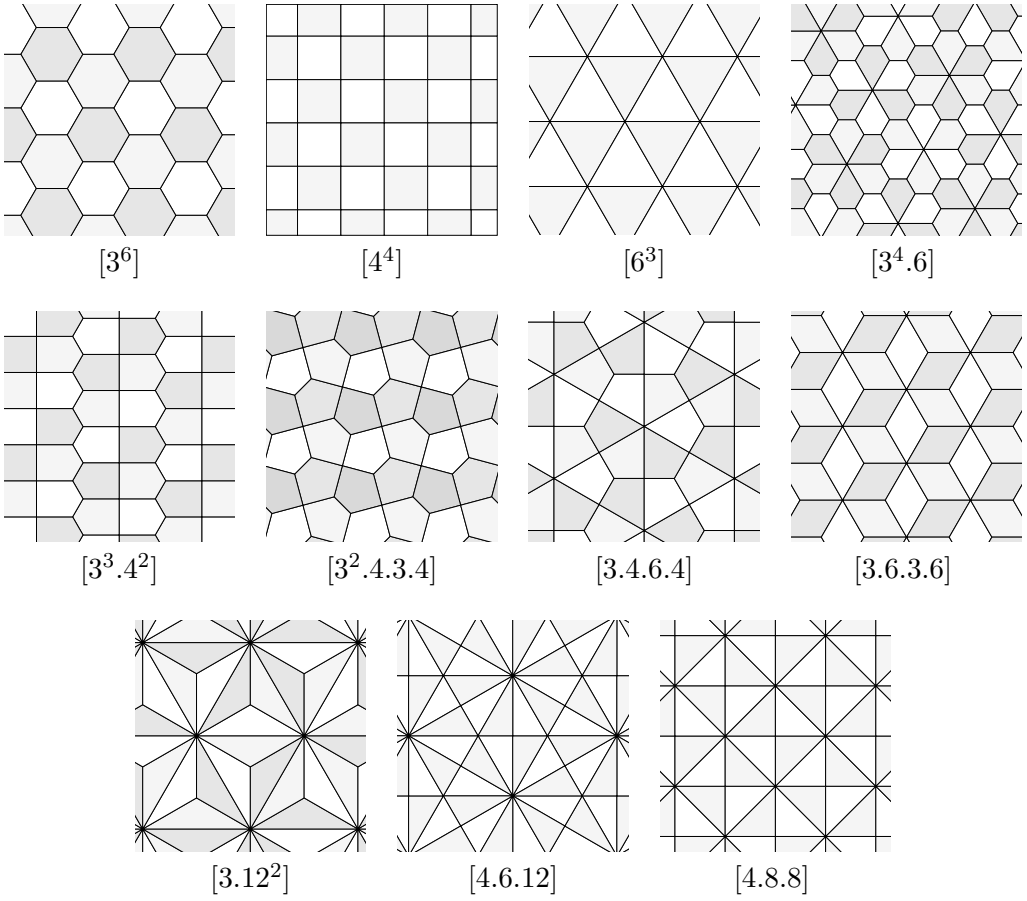


Figure 8: The eleven Laves tilings. Each one is the dual of a corresponding Archimedean tiling. The tiling $[3^4.6]$ occurs in left-handed and right-handed forms.

many more examples have been found, and the search for k-anisohedral prototiles remains an active area of research in computational tiling theory [3]. This distinction between k-isohedral and k-anisohedral is a subtle one, and of greater relevance to tiling theory than to computer graphics. However, I wish to emphasize it here because it is important not to confuse properties of tilings with properties of prototiles, particularly in the upcoming section on nonperiodic tilings.

Qualitatively, the isohedral tilings play a valuable role in art and ornamental design. They correspond to an intuitive notion of "regularity" in monohedral tilings: every tile plays an equivalent role relative to the whole. Despite that constraint, they still permit a wide range of expression. Decorative tilings developed without explicit mathematical knowledge are frequently isohedral. M.C. Escher developed his own "layman's theory" for his regular divisions of the plane [27]; each of his tiling types is equivalent to an isohedral type. This flexibility, combined with the efficient data structures and algorithms presented below, permit the isohedral tilings to be used effectively in a variety of computer graphics applications. In this section, I present an extended discussion of the mathematical structure of isohedral tilings and computational techniques for them.

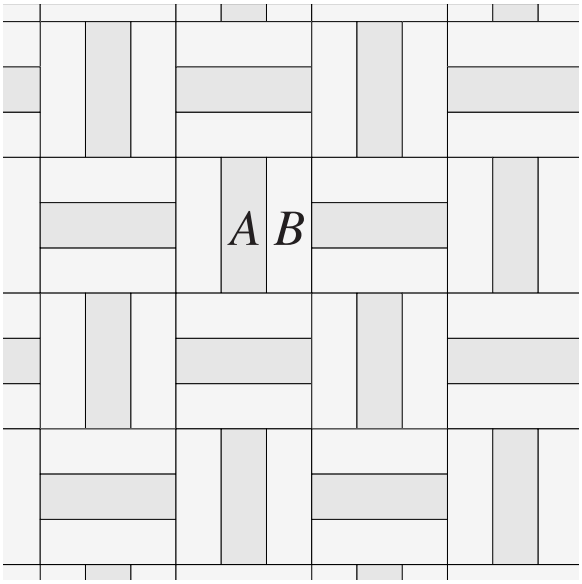


Figure 9: The "cheese sandwich tiling", an example of a monohedral tiling that is not isohedral. The tiles fall into two distinct transitivity classes: the "cheese" tiles and "bread" tiles, examples of which are labeled respectively A and B in the tiling. Any two bread tiles correspond via a symmetry, as do any two cheese tiles. But there is no symmetry of the tiling that can make tile A correspond with tile B. This tiling is therefore 2-isohedral.

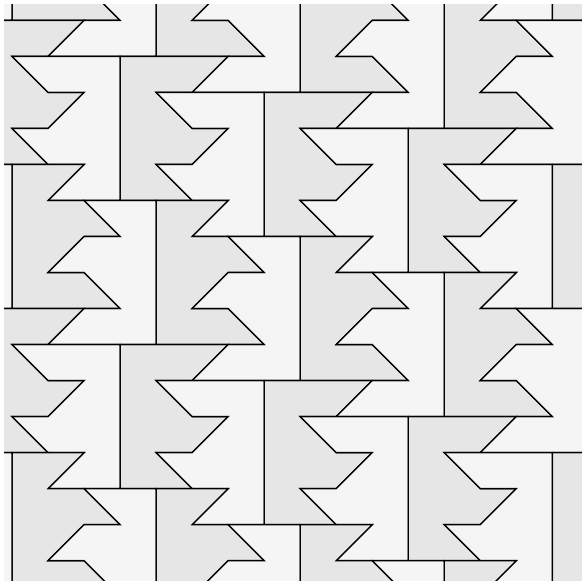


Figure 10: Heesch's anisohedral prototile. No tiling that can be assembled from this shape will be isohedral.

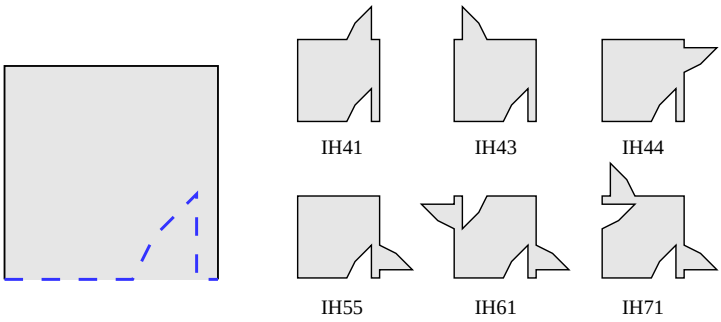


Figure 11: An isohedral tiling type imposes a set of adjacency constraints on the tiling edges of a tile. When the bottom edge of the square deforms into the dashed line, the other edges must respond in some way to allow the new shape to tile. The six resulting prototiles tiles here are from six different isohedral types, and show six of the possible responses to the deformation.

7.1 Isohedral tiling types

By definition, an isohedral tiling is bound by a set of geometric constraints: congruences between tiles must be symmetries of the tiling. Grünbaum and Shephard show that those geometric constraints can be equated with a set of *combinatorial* constraints expressing the adjacency relationships a tile maintains across its edges with its neighbours. They prove that the constraints yield a division of the isohedral tilings into precisely 93 distinct *types* or *families*, referred to individually as IH1, . . . , IH93 and collectively as IH [16, Section 6.2]. Each family encodes information about how a tile's shape is constrained by the adjacencies it is forced to maintain with its neighbours. In 12 of these types, the adjacency relationships can be realized only by placing *markings* on tiles that indicate their orientations. We will primarily be concerned with the other 81 types, where the combinatorial structure of the tiling can be expressed geometrically through deformations of the tiling edges. A change to a tiling edge is counterbalanced by deformations in other edges; which edges respond and in what way is dependent on the tiling type, as shown in Figure 11. In what follows, we review the classification and notation used with the isohedral tilings.

The *combinatorial structure* of an isohedral tiling $\mathcal{T}$ is an infinite graph whose vertices are the tiling vertices of $\mathcal{T}$, and where two vertices are connected by an edge if the two corresponding tiling vertices are connected by a tiling edge. Two isohedral tilings can then be said to be *combinatorially equivalent* when their combinatorial structures are isomorphic. Combinatorial equivalence partitions the isohedral tilings into eleven classes, referred to as *combinatorial types*, or more commonly as *topological types*.[1] Each topological type has one of the eleven Laves tilings as a distinguished representative, and we name the type using the vertex symbol of the corresponding Laves tiling. For example, Figure 12 shows an isohedral tiling of type IH16. We can see that every tile has six tiling vertices, all of valence three, meaning that IH16 is of topological type 3^6.

Every isohedral tiling is both monohedral and periodic, meaning that its behaviour over the entire plane can be summarized by specifying the aspects of the single prototile that make up a translational unit, and two linearly-independent translation vectors that replicate that unit over the plane. IH16 has three aspects, shown in varying shades of blue in Figure 12. These three tiles comprise one possible translational

[1]The use of the term "topological type" would seem to suggest that the two tilings are topologically, and not combinatorially equivalent. In fact, for normal tilings the two forms of equivalence are identical [16, Section 4.1] and so both terms are valid.

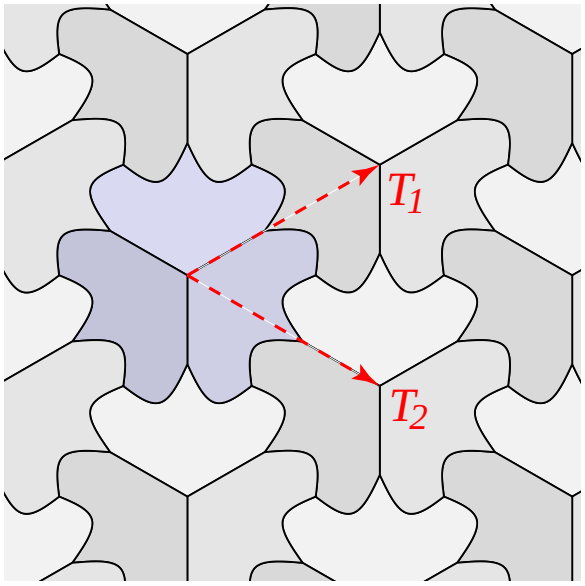


Figure 12: An example of an isohedral tiling of type IH16. A single translational unit of the tiling is shown through the two translation vectors $\vec{T}_1$ and $\vec{T}_2$ and the three coloured aspects.

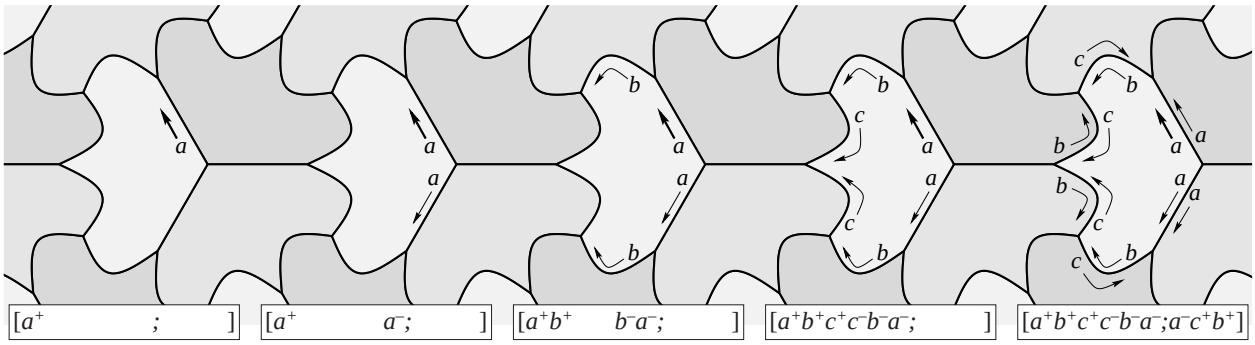


Figure 13: Five steps in the derivation of an isohedral tiling's incidence symbol.

unit with translation vectors $\vec{T}_1$ and $\vec{T}_2$.

The adjacency constraints between the tiling edges of a tile are summarized by an *incidence symbol*. Given a rendering of an isohedral tiling, the incidence symbol can be derived in a straightforward manner.

Figure 13 shows five steps in the derivation of an incidence symbol for our sample tiling. To obtain the first part of the incidence symbol, we pick an arbitrary tiling edge as a starting point, assign that edge a single-letter name, and draw an arrow pointing counterclockwise around the tile (Step 1). Then, we copy the edge's label to all other edges of the tile related to it through a symmetry of the tiling (Step 2). Should the edge get mapped to itself with a reversal of direction, it becomes undirected and is given a double-headed arrow. We then proceed counterclockwise around the tile to the next unlabeled edge (if there is one) and repeat the process (Step 3). The first half of the symbol is obtained by reading off the assigned edge names

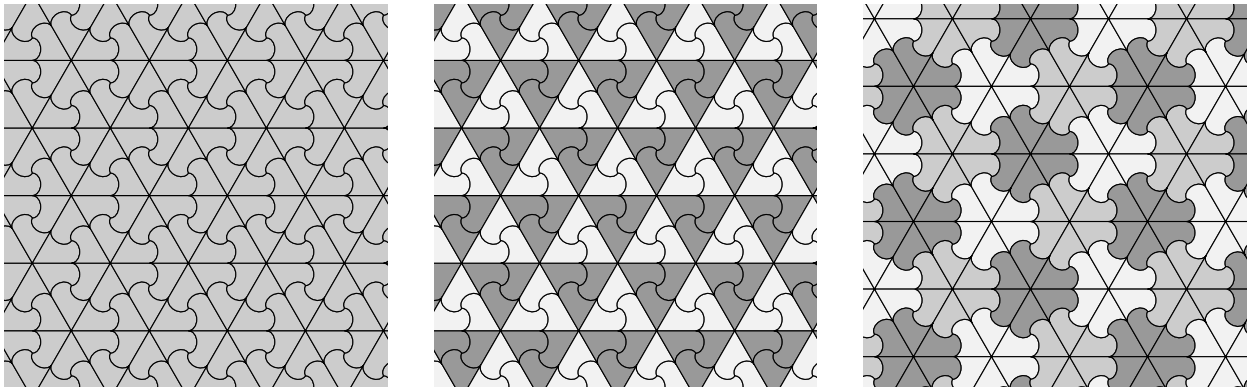


Figure 14: An isohedral tiling of type IH30, shown uncoloured on the left, and with perfect colourings using two and three colours in the middle and on the right.

(Step 4). A directed edge is superscripted with a sign indicating the agreement of its arrow with the traversal direction. Here, a plus sign is used for a counterclockwise arrow and a minus sign for a clockwise arrow.

The second half of an incidence symbol records how, for each different label, a tiling edge with that label is related to the corresponding edge of the tile adjacent to it. To derive this part of the symbol, we copy the labeling of the tile to its neighbours (Step 5). Then, for each unique edge letter assigned in the first step, we write down the edge letter adjacent to it in the tiling. If the original edge was directed, we also write down a plus or minus sign, depending on whether edge direction is respectively preserved or reversed across the edge. That is, a plus sign is used if the arrows on the two sides of a tiling edge are pointing in opposite directions, and a minus sign is used otherwise. For the running example, the incidence symbol turns out to be $[a^+b^+c^+c^-b^-a^-; a^-c^+b^+]$. Note that the incidence symbol is not unique; edges can be renamed and a different starting point can be chosen. But it can easily be checked whether two incidence symbols refer to the same isohedral type.

Every isohedral type is fully described in terms of a topological type and an incidence symbol. Enumerating all possible topological types and incidence symbols and eliminating the ones that do not result in valid tilings or that are trivial renamings of other symbols leads to the classification given by Grünbaum and Shephard.

7.2 Coloured tilings and transitivity

Up to now, we have ignored the possibility of colouring tiles in a tiling. To understand the structure of a given tiling, we might imagine colour as being superficial, to be disregarded when deciding whether two tiles are "the same." It is also possible to take colour into account, adding a layer of richness and complexity to a tiling. The colouring can have a great deal of structure, particularly when it acts compatibly with the symmetry-theoretic properties of the tiling [6]. Grünbaum and Shephard provide an extensive account of the relationship between colouring and tilings [16, Chapter 8]. I briefly restate two important definitions here. In Section 7.4.1 I will describe a simple representation of colourings for isohedral tilings.

A *k-colouring* of a tiling is a function c from tiles to the set $\{1, \ldots, k\}$ that assigns an abstract "colour" $c(T)$ to each tile T (with the assumption that every colour is used at least once in the tiling). A colouring is a *perfect colouring* if every symmetry of the tiling acts as a permutation of the colours. That is,

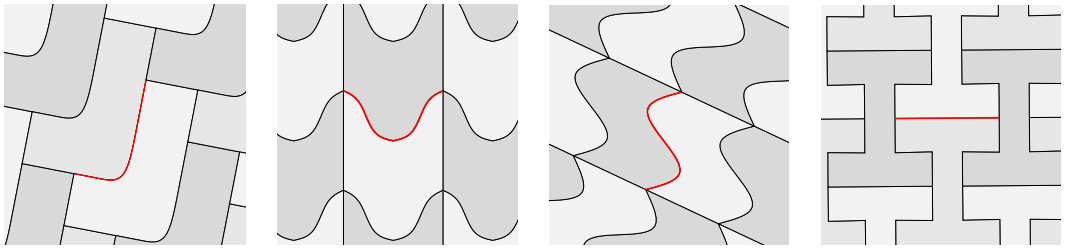


Figure 15: Examples (from left to right) of **J**, **U**, **S** and **I** edges. In each case, the tiling edge with the given shape is highlighted in red.

if σ is some rigid motion that maps the tiling onto itself, then two tiles have the same colour if and only if their images under σ have the same colour. Figure 14 shows two different perfect colourings of an isohedral tiling.

Just as the isohedral tilings formalize an intuitive notion of regularity, perfect colourings are a natural way to colour tiles in an orderly way. A perfect colouring of an isohedral tiling operates compatibly with the tiling's symmetries.

7.3 Parameterizing the isohedral tilings

Within a single isohedral type, different prototiles are distinguished from each other by their shapes, determined by the positions of the tiling vertices and the shapes of the curves that join them. In order to move from the combinatorial description of isohedral tilings to a geometric one, we must understand how incidence symbols dictate the range of possible prototile shapes for a given isohedral type. We parameterize the space of isohedral tilings by giving, for each type, an *edge shape parameterization* and a *tiling vertex parameterization*. The former encodes the minimal non-redundant geometric information sufficient to reconstruct the tiling edges. The latter determines the legal configurations of tiling vertices.

7.3.1 Edge shape parameterization

The constraints on the shapes of tiling edges in an isohedral tiling are simple to describe. Although the underlying choice of how to represent a curve is left open, the tiling's symmetries imply a great reduction in the tiling edges' degrees of freedom. These constraints can be extracted directly from the tiling's incidence symbol. We enumerate the four cases for the structure of a tiling edge. For each case, Figure 15 shows a tiling with such an edge.

If some directed edge is adjacent to itself without a flip, then a tile's neighbour across that edge is adjacent through a half-turn. This rotation forces the edge shape to itself be symmetric through a half-turn about its centre. We call such an edge an **S** edge as a visual mnemonic. Only half of an **S** edge is free; the other half must complete the rotational symmetry. In an incidence symbol, we can identify an **S** edge as an edge name x that is adjacent to x^+.

An undirected edge must look the same starting from either end, meaning it must have a line of mirror symmetry through its midpoint. If an edge name x appears in an incidence symbol without a sign, and is adjacent to some other edge name $y \neq x$, then x is free to take on any curve with this bilateral symmetry. We call it a **U** edge. Again, only half of a **U** edge is free.

If an undirected edge is adjacent to itself, or if a directed edge is adjacent to itself with a change in sign, that edge must have both **S** symmetry and **U** symmetry. The only shape that has both is a straight line, leading us to call such an edge an **I** edge.

The remaining case is when a directed edge is adjacent to some other directed edge. Such an edge is free to take on any shape, and we call it a **J** edge.

Note also that if an edge x is adjacent to an edge y, then x and y have the same shape (even though they have different names). In this case, we need only represent one tiling edge, since the other is entirely constrained to it. Thus, referring back to the derivation presented in Figure 13, the tiling edges of IH16 can be summarized by one curve: the shape of the edge labeled b. Edges labeled a are **I** edges and have no degrees of freedom, and edges labeled c are constrained to b.

7.3.2 Tiling vertex parameterization

Like the shape vertices, tiling vertices cannot move entirely independently of each other. Moving one tiling vertex forces the others to move to preserve tileability. The exact nature of this movement depends on the tiling type in question. The incidence symbol for a tiling type implies a set of constraints on the tiling polygon's edge lengths and interior angles. Any tile of that type will have a tiling polygon that obeys those constraints.

In a constructive model of isohedral tilings, it is not sufficient merely to recognize the constraints on the shape vertices: we need a way to explicitly navigate the space of legal tiling polygons. This section provides explicit parameterizations of the tiling vertex configurations for IH. They can be derived by determining angle and length constraints from the incidence symbols and parameterizing the unconstrained degrees of freedom. In some cases, parameterizations are shared between tiling types: nine tiling types have squares as tiling polygons (implying a parameterization with zero parameters), and seven have parallelograms (implying two parameters). In all, the 93 isohedral types require 45 different parameterizations.

Diagrams of the tiling vertex parameterization are given in Figures 16 and 17. Figure 18 shows example tilings of type IH16 that can result from different values of its single free parameter.

These parameterizations can be seen as an elaboration of those provided by Heesch and Kienzle for the 28 Heesch tiling types [18]. Each Heesch type is identical to one of the isohedral types, and for those types the parameterizations coincide. The remaining isohedral types have parameterizations where degrees of freedom are coalesced to yield more symmetric tiling polygons.

7.4 Data structures and algorithms for IH

In this section, I provide more details on how the edge shape and tiling vertex parameterizations can be developed into a concrete implementation. I discuss **Tactile**, a library I developed to represent, manipulate and render isohedral tilings. At the top level, the library provides two classes: `IsohedralTemplate`, an abstraction of an isohedral tiling type, and `IsohedralTile`, an abstraction of a specific prototile. The template contains information about a tiling type in general, information that doesn't change from instance to instance. The prototile refers to a template and contains the information needed to determine the locations, shapes, and colours of tiles. I describe each of these components in detail, and then show how they can be used to support efficient editing and viewing.

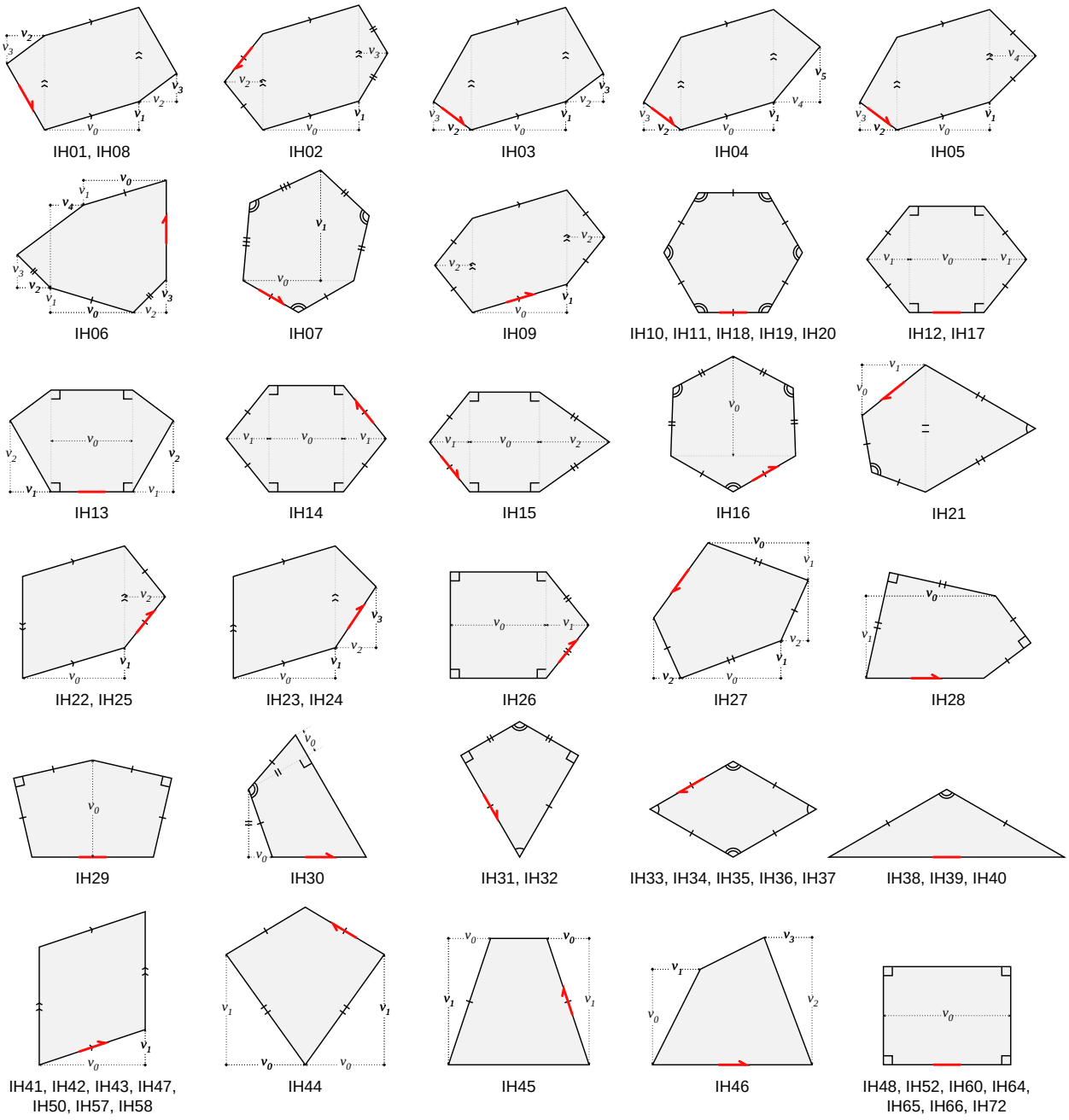


Figure 16: The complete set of tiling vertex parameterizations for the isohedral tilings. In each tile, the edge marked with a red line is the first edge in the tiling type's incidence symbol. When that first edge is directed, the red line has an arrowhead. Labelled dotted lines represent parameter values, and are horizontal or vertical (with the exception of one guide line in the diagram for IH30). Since the diagrams are scale independent, distances that do not depend on parameters can be taken to have unit length. Tile edges cut with the same number of short lines have the same length, and edges cut with chevrons are additionally parallel. A single arc, a small square, and a double arc at vertices represent $60°$, $90°$, and $120°$ angles, respectively.

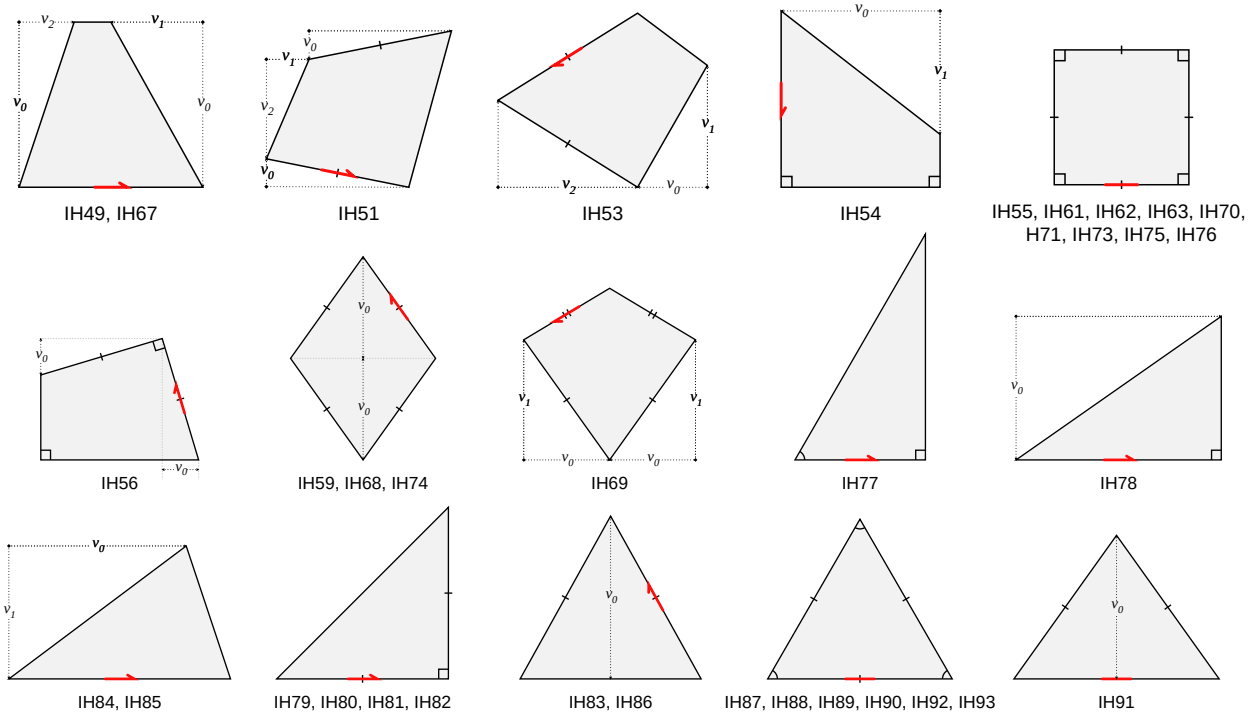


Figure 17: The complete set of tiling vertex parameterizations for the isohedral tilings (continued).

7.4.1 Isohedral templates

The templates are derived once ahead of time, and stored in a master file (`isohedral.ih`) designed to be computer readable. This file has been publicly available on the internet since 2000, and has been extensively debugged in that time. A copy can be downloaded from `http://www.cgl.uwaterloo.ca/~csk/projects/escherization/`.

The template file contains one record for each isohedral type. A sample of such a record appears in Figure 19. It reproduces some of the information tabulated by Grünbaum and Shephard, such as the topological type (Line 1), the incidence symbol (Line 2), and the number of aspects (Line 4). It also gives a default colouring (Line 3). The remaining information, the `rules` section (Lines 5–9), is a symbolic description of how to compute the tiling's translation vectors and transformation matrices for its aspects. We execute the symbolic description and cache the resulting transforms in an `IsohedralTile` to permit efficient rendering of tilings. In what follows, I provide more details about the `colouring` field and `rules` section.

The `colouring` field provides a default rule for assigning colours to tiles (colourings of tilings are described in Section 7.2). An `IsohedralTile` may override this default with its own colouring. Here we follow Escher's lead and aim to provide perfect colourings. Recall that in a perfect colouring, every symmetry of the tiling is a permutation of the set of colours.

The actions of all the symmetries can be summarized by giving the permutations associated with the two translation vectors of the tiling and an assignment of colours to the aspects in a single translational unit. Successive translations will permute this default assignment appropriately. The colouring field in the

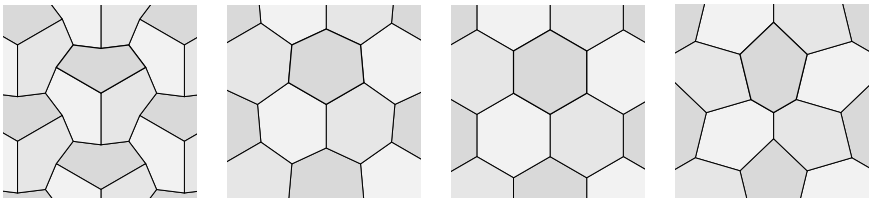


Figure 18: Some examples of IH16 with different values for the single parameter in its tiling vertex parameterization.

```
    template IH16 {
1      topology 3^6
2      symbol [a+b+c+c-b-a-;a-c+b+]
3      colouring (1 2 3) (1 2 3) (1 2 3)
4      aspects 3
5      rules
6         aspect 2 1
7         aspect 3 6
8         translate T1 1,5
9         translate T2 1,3
    }
```

Figure 19: The tiling type information stored for IH16

template gives, in order, the number of colours, the assignment of colours to aspects in a translational unit, and the permutations of the assignment associated with the two translation vectors. A permutation ρ of the numbers $\{1, \ldots, n\}$ is very simply represented as a sequence $(s_1 \ldots s_n)$, with $\rho(k) = s_k$.

In particular, consider a tiling with translation vectors $\overrightarrow{T_1}$ and $\overrightarrow{T_2}$ and their associated colour permutations ρ_1 and ρ_2. Let the tiling have n aspects, with the default colours in a translational unit given as $(c_1, \ldots, c_n)$. Then aspect k in the translational unit located at $a\overrightarrow{T_1} + b\overrightarrow{T_2}$ will have the colour $\rho_2^b(\rho_1^a(c_k))$. This encoding can in fact express a superset of the perfect colourings, but it is easy to check empirically whether a given colouring is perfect.

The `rules` section gives a collection of rules that, when applied to a tiling polygon, yield rigid motions (in the form of transformation matrices) for all the aspects of a translational unit, as well as for the two translation vectors. These transforms cannot be computed ahead of time, as they depend on the tiling polygon. I speed up the drawing of the tiling by storing these transform matrices in the `IsohedralTile` instance, and recomputing them only when the tiling vertices move.

Every tile in an isohedral tiling is surrounded in a consistent way by its neighbours, and so for every tiling edge there is a well-defined rigid motion that carries the tile on one side of that edge to the tile on the other side. The motion will either be a half-turn around the edge's center (in the case of an **S** edge), a reflection across the edge (in the case of an **I** edge), a glide reflection (in the case of some **J** edges) or a translation. The kind of transform that applies can be determined from the tiling type's incidence symbol, and the numeric values in the transform matrix depend on the positions of the tiling vertices that delimit

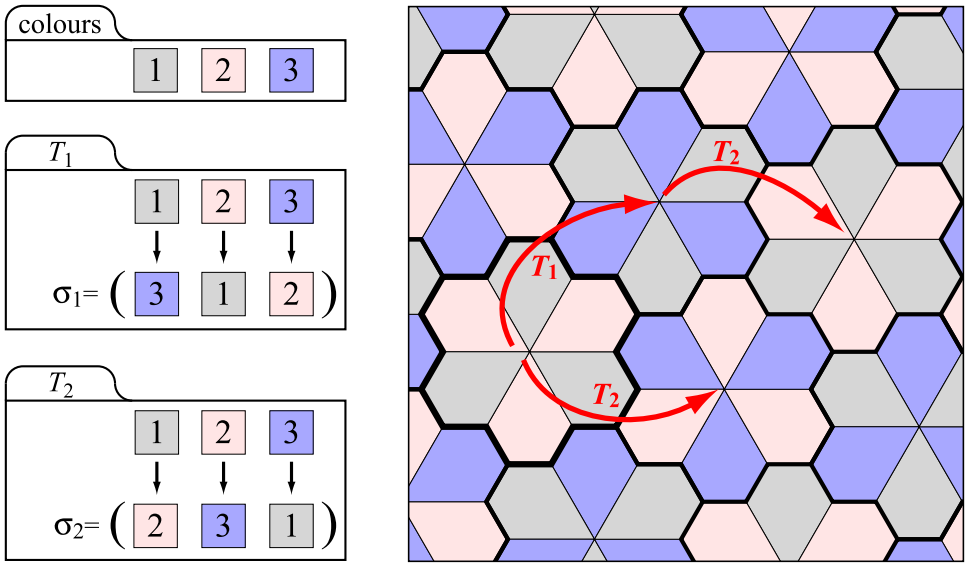


Figure 20: A demonstration of how the colouring information in the isohedral template (for IH21 in this case) is used to apply colours to tiles. The translational units (each containing six aspects) are outlined in bold. There are three symbolic colours, $\{1, 2, 3\}$, and they are associated respectively with gray, pink, and blue. On the left, the permutations for the two translation vectors are indicated by showing with arrows the mapping from original to permuted colours; the permutation's textual desciption can be read off of the bottom row of this mapping. On the right, the permutations are applied when moving between translational units. The colouring for this tiling can be read from the diagram as `colouring 3 (1 2 1 2 1 2) (3 1 2) (2 3 1)`.

the edge. We call such a rigid motion a "hop" across a tile edge. In a tile with n edges, we can label the hops unambiguously as $H_1, \ldots, H_n$. Each `rule` encodes a sequence of hops that, when chained together, transform a tile to a new aspect or to the same aspect in a neighbouring translational unit.

Aspect 1 is always given the identity matrix as its transform, and the other aspect transforms are computed from it. In the example, the first rule (Line 6) says that the transform for creating aspect 2 from the first aspect is the hop across edge 1 of the first aspect — that is, a reflection across the first edge, labelled `a+`, in the incidence symbol. We will store H_1 as the aspect's transform matrix. Similarly, the second rule (Line 7) says that the transform for creating aspect 3 from the first aspect is H_6, a reflection about the edge labelled `a-`.

The two translation vectors are specified in the same way. Here, we can obtain translation vector $\overrightarrow{T_1}$ in two hops, first from the first aspect across edge 1 into some neighbouring tile, and from *that* tile across edge 5. The resulting transform matrix would be $H_1 H_5$. Note, however, that this matrix does not necessarily represent a translation, and so we cannot just take $\overrightarrow{T_1}$ to be the translational component of that matrix. The problem is that the matrix may contain internal symmetries of the tile shape, which were accumulated when composing the hops together. Fortunately, we can still extract the translation in a simple way as the vector joining the centroids of the transformed and untransformed tiling vertices. This calculation works because the centroid is independent of internal tile symmetries, operations that merely permute the vertices.

In general, a `rule` may specify any number of hops to get from the first aspect to another aspect or

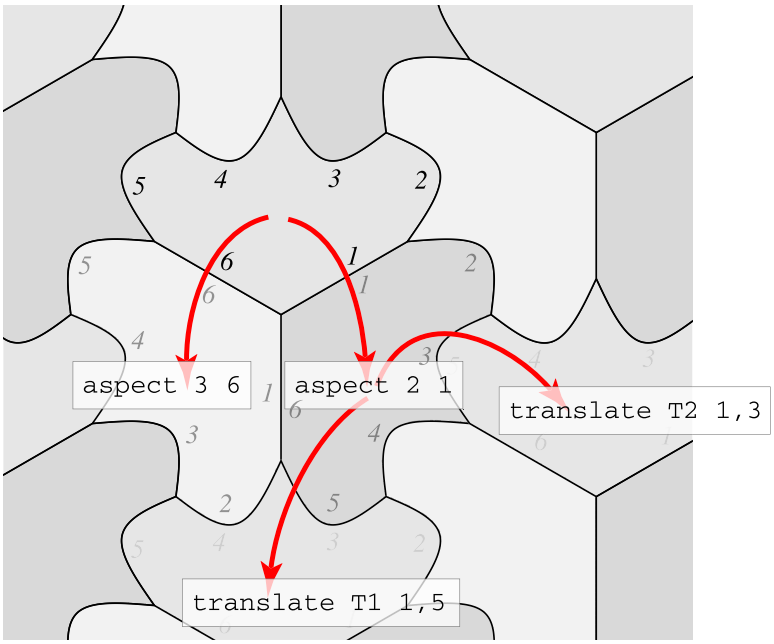


Figure 21: A visualization of how aspect transforms and translation vectors are computed for IH16, using the information in the `rules` section of the isohedral template (see Figure 19). In the order that they are referenced in the template, the aspects are coloured blue, pink, and gray. The edges are numbered as they are given in the incidence symbol. Each red arrow represents a single hop, a rigid motion that brings a tile into coincidence with one of its neighbours. The end of every sequence of hops is labeled with the corresponding rule from the template for IH16 (see Figure 19).

a translation. Each step in the `rule` names an edge of the tile, and the transform is computed by composing together the associated hops.

One piece of per-tiling-type information missing from the template file is the set of tiling vertex parameterizations. When developing this library, I described the parameterizations in code rather than in a table-driven format. Each is expressed as a C++ class. The file `params.py`, available at the same location as `isohedral.ih`, provides equivalent Python code. An example from `params.py` is given in Figure 22. With hindsight, it is clear that the coordinates of the tiling vertices are all linear functions of the parameters, as are the entries in the hop transformation matrices. In a revised implementation, I would include coefficients for all these linear functions directly in `isohedral.ih`.

This representation of isohedral tilings suffers from a flaw related to degenerate edges in the tiling polygon. If two consecutive tiling vertices are made to coincide, then the hop across their shared edge is undefined, and any `rules` that use the degenerate edge give invalid transforms. In a purely mathematical treatment of the subject there is no problem, because there is no such thing as a degenerate edge in the tiling polygon. As two adjacent tiling vertices merge, they fuse into a single vertex and the tiling as a whole slips into a different (but related) isohedral type. The representation given here can manipulate non-degenerate tiles without any difficulty, but it cannot handle these discontinuous transitions.

```
def ih16_params( v0 ):
    m = 0.5 / math.sqrt( 3.0 )
    T1 = match( Point( 0.5, v0 ), Point( 1, 0 ) )
    T2 = match( Point( 0, 0 ), Point( 0.5, v0 ) )

    return (
        Point( 0.5, -m ),
        Point( 1, 0 ),
        T1 * Point( 0.5, m ),
        Point( 0.5, v0 ),
        T2 * Point( 0.5, m ),
        Point( 0, 0 ) )
```

Figure 22: Sample Python code implementing the tiling vertex parameterization for IH16. When called with a single real parameter `v0`, the function returns a tiling polygon. The function `match` takes two points as arguments and returns a direct rigid motion that maps the unit interval onto the line segment given by the two points.

7.4.2 Isohedral prototiles

All the information related to a specific isohedral prototile is stored in the `IsohedralTile` class. A great deal of data is stored in every `IsohedralTile`:

- **Geometry information** includes the parameters for the tiling vertex parameterization, a cached tiling polygon, and the cached aspect transforms and translation vectors derived from the `rules` section of the `IsohedralTemplate`.
- **Shape information** contains polygonal paths that make up the non-redundant portion of the tile's outline (called the "fundamental edge shapes"). The shape information also includes a cached copy of the tile's outline for fast drawing.
- **Colouring information** contains a colouring (like the one that appears in `isohedral.ih`) and actual RGB triples for each symbolic colour.

A callback mechanism ensures that when part of the tile's description changes (for example, when a vertex parameter is adjusted), all cached information that depends on it is automatically updated.

Each fundamental edge shape is an array of points representing a path starting at $(0, 0)$ and ending at $(1, 0)$. Any path description can be used here. My implementation supports piecewise-linear paths and subdivision curves.

The shape information in the prototile contains a hierarchical model of rigid motions whose leaves are the fundamental edge shapes. The model makes multiple references to fundamental edges to express the redundancy inherent in the tile's outline. To rebuild the tile shape, we apply the tiling vertex parameterization to obtain the positions of the tiling vertices and use the hierarchical model to construct edge shapes between them.

There are at most three levels in the hierarchical model between a fundamental edge shape and a point on the outline of the tile. The first level takes into account the symmetries of **U** and **S** edges. Half of

the **U** or **S** edge comes directly from the fundamental edge. The other half is derived from the first half as needed through rotation or reflection. **J** edges are passed unmodified through this level, and since **I** edges are immutable, all tiles share a single system-wide copy of an **I** edge.

At the next level up, we recognize that edges with different names in the incidence symbol may still have related shapes. In IH16, for example, the edge named `b+` is adjacent to `c+`, forcing the two edge shapes to be congruent. In this case, the two edges share the same shape passed up from the level below.

Finally, the topmost level maps the unit interval to an edge of the tiling polygon; this mapping will move an edge shape from its normalized coordinate system into a portion of the tile's outline. At this level, all edges with the same label in the incidence symbol share a lower-level shape object.

7.5 Beyond isohedral tilings

Before the isohedral tiling types were enumerated, Heesh and Kienzle developed a family of what are now known as *Heesch tilings* [18, 27]. Each Heesch type is represented by a symbol that encodes the type of adjacency across the tiling edges and the order of rotational symmetry (if any) at the vertices. There are 28 Heesch types, corresponding to the "primitive" isohedral types—those in which no non-trivial symmetry of the tiling maps a tile to itself. In many typical applications the Heesch types are sufficient, since the internal symmetries of tiles can always be contrived by choosing edge shapes appropriately. In any case, the isohedral types are at least as flexible, and permit a finer level of discrimination that might be useful in some contexts.

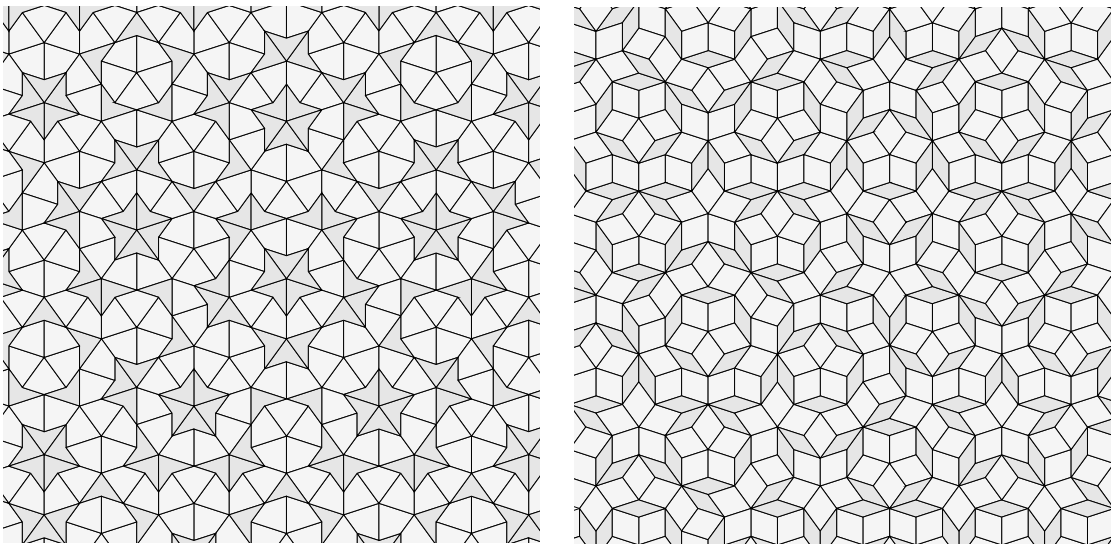
Since the work of Grünbaum and Shephard on the classification of isohedral tilings of the Euclidean plane, other tiling theorists have gone on to search for generalizations to related tilings. In particular, a group led by Dress, Delgado Friedrichs, and Huson pioneered the use of *Delaney symbols* in the study of what they call *combinatorial tiling theory* [9, 19]. A Delaney symbol completely summarizes the combinatorial structure of a k-isohedral tiling of the Euclidean plane, the hyperbolic plane, or the sphere. They can also be generalized to tilings in spaces of dimension three and higher. Delaney symbols form the basis for an efficient software implementation, and Delgado Friedrichs and Huson have created a series of software tools for exploring, rendering, and editing tilings from their combinatorial descriptions (see the Gavrog project at `http://gavrog.sourceforge.net/` for more information). With some additional work, Delaney symbols might be used in place of incidence symbols above, offering additional flexibility not described here.

8 Nonperiodic and aperiodic tilings

Aperiodic tilings have received a great deal of attention over the past few decades, both from tiling theorists and lay audiences. Perhaps best well known are those invented by Penrose. Two examples are shown in Figure 23, one constructed from "kites" and "darts" and the other from rhombs of two sizes.

A tiling that is not periodic is called *nonperiodic*. A frequent but incorrect assumption is that the notions of aperiodicity and nonperiodicity coincide. In fact, the aperiodic tilings are a very special subset of the nonperiodic tilings. It is worth clarifying the distinction between the two (which is similar to the distinction made in Section 7 between k-isohedral and k-anisohedral), showing why the aperiodic tilings are an active and exciting area of research.

Lack of periodicity is not in itself a very surprising property. It is easy to construct tilings where every tile shape is unique – consider, for example, the Voronoi diagram induced by an infinite integer lattice whose points have been jittered randomly. Clearly there can be no hope of periodicity in such a case, but



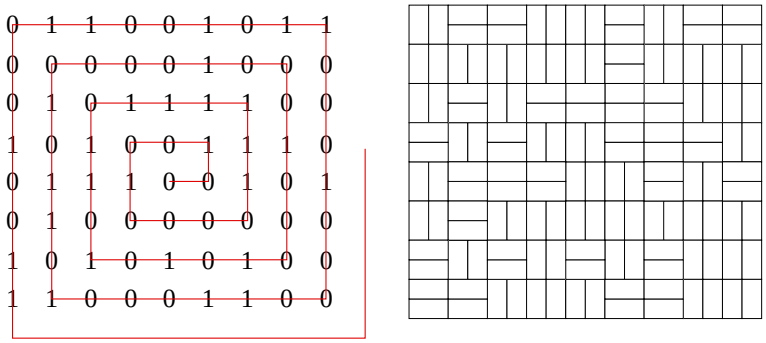
(a) (b)

Figure 23: The two famous aperiodic tilings of Penrose. The "kite and dart" tiling is shown in (a) and thin and thick rhombs in (b).

this fact seems unimpressive. In developing a definition of aperiodicity, we therefore consider only those nonperiodic tilings with a finite number of prototiles, i.e., those that are k-hedral for some k.

Under this restricted definition, we can still construct very simple nonperiodic tilings. Even a 2-by-1 brick yields an infinite variety, as demonstrated in Figure 24. These tilings seem contrived, however, because the same brick can easily be made to tile periodically. Nonperiodic tilings become truly interesting when we take into account all possible alternative tilings that can be constructed from the same set of prototiles. We call a set of prototiles an *aperiodic tile set* when the set admits at least one tiling, but none that are periodic. We can then define an *aperiodic tiling* as a tiling whose prototiles are an aperiodic tile set. An aperiodic tiling is one that is "essentially nonperiodic", in the sense that no rearrangement of its tiles will achieve periodicity. When used to refer to a particular tiling, aperiocity is therefore a far reaching concept—it encompasses all possible tilings that can be formed from the same prototiles. As a result, it can be very difficult to establish the aperiodicity of a set of prototiles. Periodicity is comparatively easy: one need only exhibit a translational unit.

The Penrose tilings highlight the special behaviour of aperiodic tilings. Consider the tiling in Figure 23(b). By themselves, the two rhombs do not form an aperiodic tile set; they can be arranged into both periodic and nonperiodic tilings. What makes the rhombs aperiodic are additional "matching conditions" that are imposed on them, limiting the adjacencies that may occur in a complete tiling. As discussed in Section 3, these matching conditions can be expressed in a number of ways. One possibility is to modify the shapes of the tiles by adding protrusions to the rhomb edges, so that the tiles must snap together like pieces in a jigsaw puzzle [16, Section 10.3]. A set of protusions that express the matching conditions is shown in Figure 25. It is these two modified shapes that form an aperiodic tile set, known as the Penrose tile set $P3$ (the modified kite and dart are known as $P2$). Many sets of prototiles must be endowed with similar matching conditions to enforce aperiodicity. The matching conditions are typically not shown when



(a) (b)

Figure 24: A contrived example of how even a very simple shape may yield nonperiodic tilings. A spiral path is used to place digits from the binary expansion of π. Each digit is then used to place a pair of bricks, oriented vertically to represent a 0 and horizontally to represent a 1. The resulting tiling, when extended to the whole plane, is (probably) nonperiodic, even though the brick prototile could easily be used to construct periodic tilings. There exist uncountably many nonperiodic tilings based on this prototile.

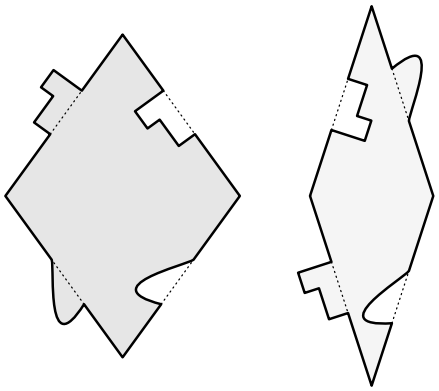


Figure 25: Sample matching conditions on the rhombs of Penrose's aperiodic tile set $P3$. The unmodified rhombs (indicated by dotted lines) can form many periodic tilings. The puzzle-piece deformations on the tile edges guarantee that any tiling formed from these new shapes will be nonperiodic.

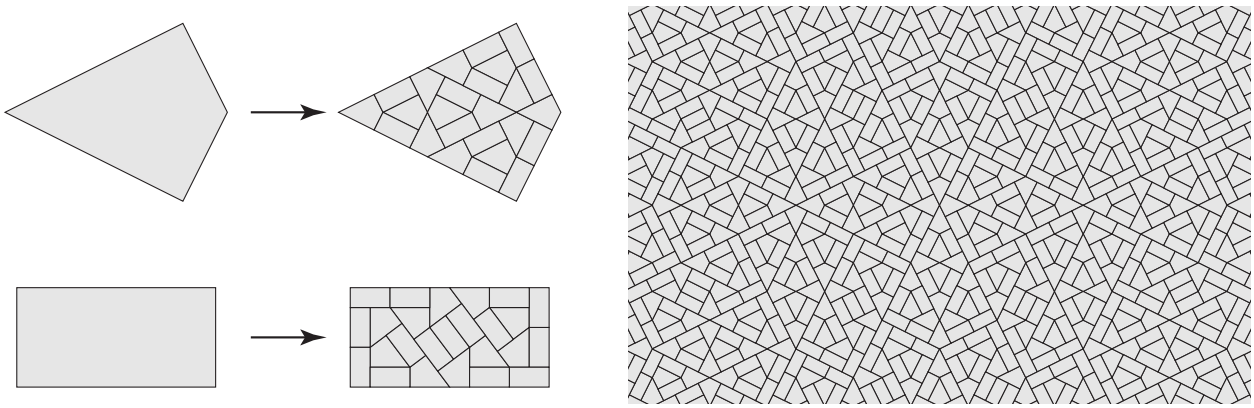


Figure 26: A simple example of a substitution tiling, the Kite-Domino tiling by Frettlöh and Baake. The prototiles are made from edges of side lengths 1 and 2. The diagram on the left shows how each of the prototiles is composed from smaller tiles. A sample patch of tiles appears on the right.

the tilings are rendered, perhaps leading to the confusion between nonperiodicity and aperiodicity.

In computer graphics applications, true aperiodicity is rarely a requirement. We often ask only that we can construct large patches of tiles that do not appear too "orderly". Many algorithms might produce such patches, as part of tilings that are nonperiodic without being aperiodic. This section will explore both nonperiodicity and aperiodicity, returning to Penrose tilings after a discussion of substitution systems and rep-tiles, and an introduction to Wang tiles.

8.1 Substitution tilings and rep-tiles

A square can trivially be divided into four smaller congruent squares. We can then scale the subdivided figure by a factor of two to obtain four squares congruent to the original. By iterating this procedure, we can build ever larger patches of squares. Without any appeal to periodicity, The Extension Theorem (Section 3) tells us that we can in fact tile the plane with squares. Alternatively, we can cover any region of the plane with squares to any level of subdivision by surrounding the region with a single large square and subdividing repeatedly, omitting the rescaling operation.

This simple example captures many of the fundamental properties of *substitution tilings*. In a general substitution tiling, one or more prototiles are given, together with rules that assign a patch of tiles to each prototile. If the prototiles are enumerated as $\{T_1, \ldots, T_n\}$, then the substitution rule for T_k can be written as $\{(i_1, M_1), \ldots, (i_{r_k}, M_{r_k})\}$, where each pair consists of an index between 1 and n and a similarity transformation. The rule indicates that T_k should be replaced with r_k tiles: a copy of T_{i_1} transformed by M_1, a copy of T_{i_2} transformed by M_2, and so on. An example of a substitution tiling with two prototiles is given in Figure 26.

A patch of tiles is constructed by starting with any single tile (or initial patch) and repeatedly applying the substitution rules to all tiles. If the similarity transformations cause each tile to be replaced by a congruent union of smaller tiles, then we can apply these rules as above to cover any region of the plane to any level of detail. We simply choose an initial T_i and transform it so that it surrounds the region. Then we can apply the substitution rules as many times as desired. A simple recursive implementation that accumulates transformations, similar to the rendering of self-similar curves like the Koch snowflake, can serve as a

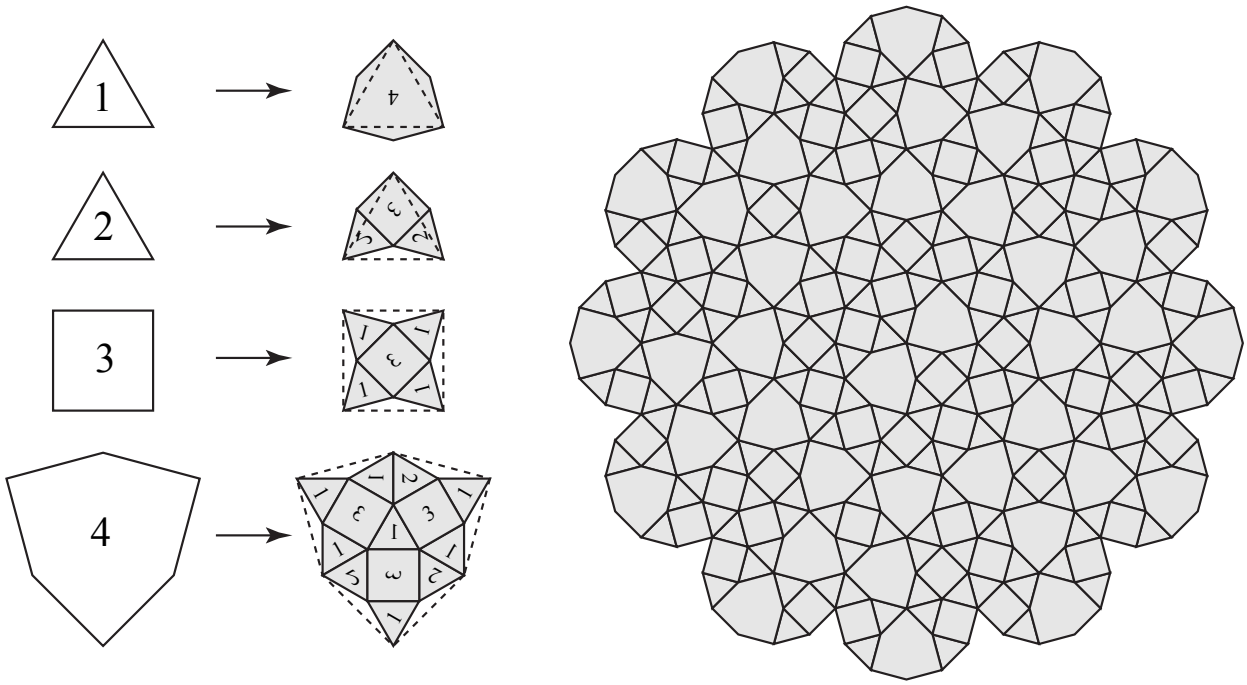


Figure 27: A set of substitution rules for Gähler's Shield tiling, together with a sample patch of tiles. The substituted tiles leave indentations and protusions in their parents' outlines. Each prototile is marked with a number; the numbers are shown transformed on the right-hand sides of the rules to indicate which prototiles should be used and how they should be transformed.

drawing algorithm.

Unlike the example in Figure 26, a substitution rule need not replace a tile T_i with a patch of smaller tiles whose union is congruent to T_i. (Such tilings are referred to as *similarity tilings* by Grünbaum and Shephard [16, Section 10.1], and the substitutions are said to be *compositions*.) The substituted tiles can extend beyond the parent or leave indentations, as long as those inconsistencies are accounted for by corresponding geometry in neighbouring subdivided tiles (see Figure 27). The rules may also produce tiles that overlap, as long as congruent tiles overlap in their entirety and can be identified with each other (see Figure 28). This added complexity occasionally requires extra bookkeeping during substitution: there might exist two prototiles T_i and T_j that are congruent, but distinguished by their indices and the rules associated with them. Implementing such cases is equally straightforward, but special anotations might be required when visualizing the substitution rules graphically.

In the special case that $n = 1$, the single prototile is called a *rep-tile*. All triangles and parallelograms are easily seen to be rep-tiles. Two well-known examples are the Chair tiling and the Sphinx tiling; both are shown in Figure 29. These two rep-tiles produce nonperiodic tilings. Indeed, many substitution systems can be shown to produce nonperiodic tilings, provided the rules are in some sense unique [16, Theorem 10.1.1].

Polyominoes [14] can serve as a rich source of substitution tilings and rep-tiles. A polyomino is a finite set of squares, joined edge-to-edge. Many polyominoes are rep-tiles and lead immediately to tilings of the plane. More generally, if a set of polyominoes can be assembled into any rectangle, then they can serve as prototiles in a substitution system. We arrange sufficiently many of these rectangles to form a square, and

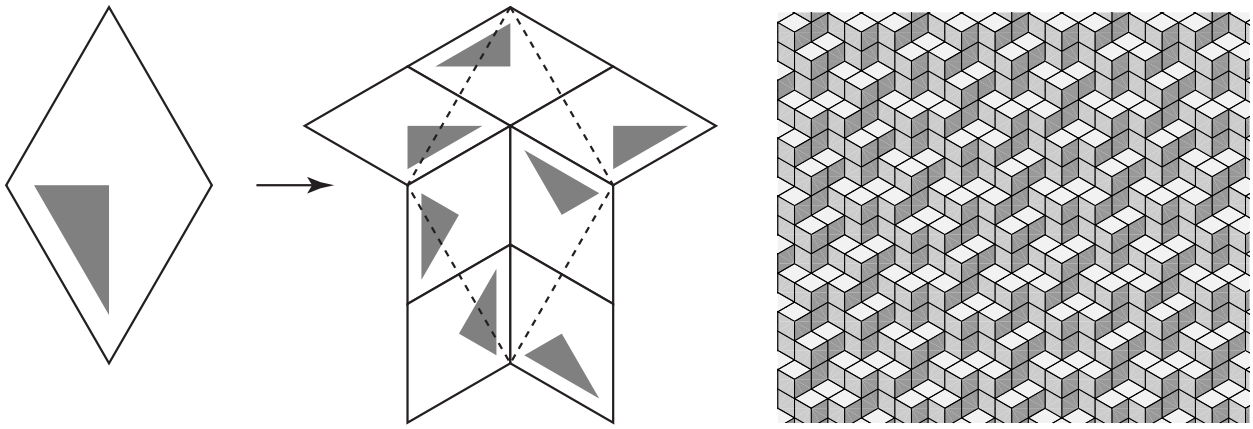


Figure 28: A set of substitution rules for Lord's nonperiodic tiling, together with a sample patch of tiles. The rules cause tiles to overlap with each other, but in such a way that overlaps happen in their entirety, avoiding any inconsistencies. A shaded triangle is used to indicate the orientations of substituted tiles.

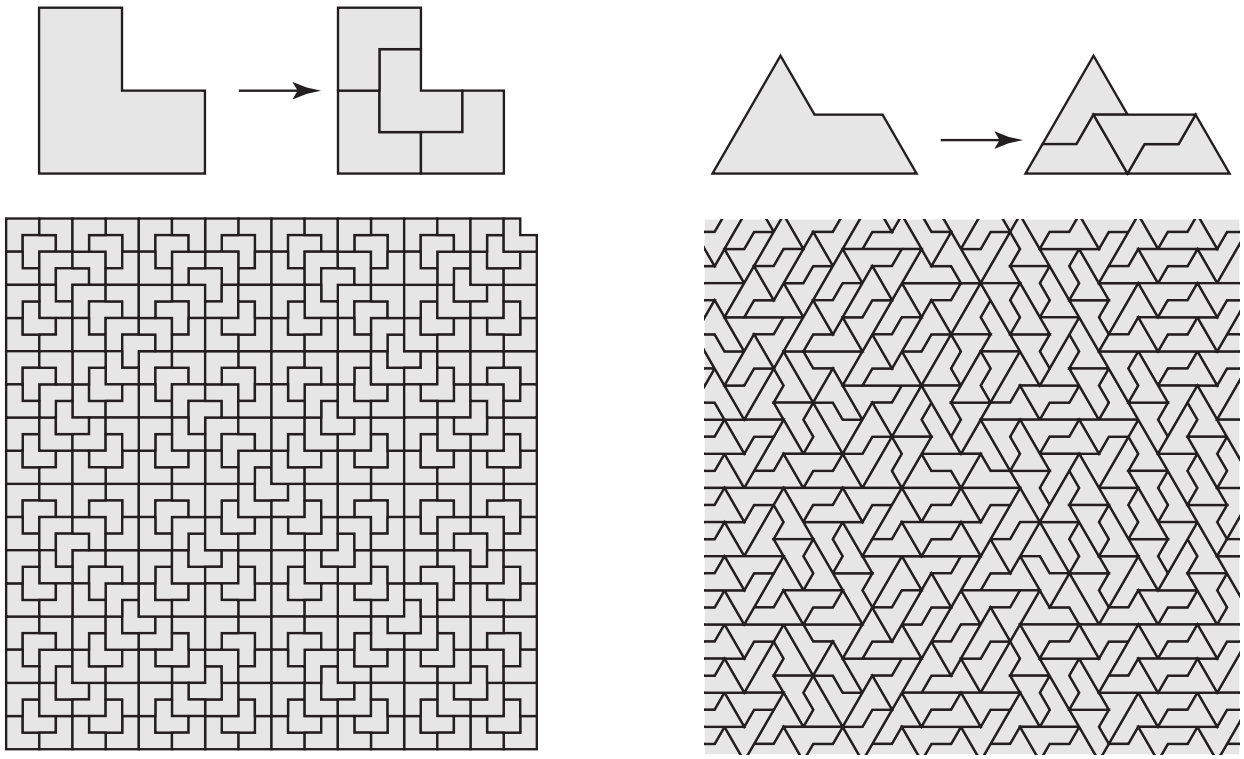


Figure 29: Two well-known examples of rep-tiles: the Chair tiling on the left and the Sphinx tiling on the right.

substitute this square for every unit in each of the prototiles.

A large collection of substitution tilings (including those illustrated in Figures 26–29) is maintained by Harriss and Frettlöh at `http://tilings.math.uni-bielefeld.de/tilings/index`.

8.2 Wang tiles and Aperiodicity

In Section 3, I mentioned the *Tiling Problem*: does a given set of shapes admit any tilings of the plane? A solution to the tiling problem should take the form of an algorithm that decides in finite time whether or not any tilings exist.

Let us examine this question in the context of a restricted family of tiles. *Wang tiles* are square prototiles with marked edges. The markings are usually indicated by colours, with the understanding that like colours must meet across edges in any tiling by these tiles. We impose two additional restrictions. First, tiles must meet edge-to-edge. Second, they must be placed in a tiling by translation only—rotations and reflections are forbidden. Note that all of these restrictions can be expressed geometrically if desired. Wang tiles effectively reduce the problem of fitting tiles together to one that is discrete and combinatorial.

As explained by Grünbaum and Shephard, Hao Wang articulated four possible outcomes for any set of shapes [16, Section 11.3]:

1. They do not admit any tilings.
2. They admit tilings that are always periodic.
3. They admit both periodic and nonperiodic tilings.
4. They admit tilings that are always nonperiodic.

The fourth case is precisely our definition of an aperiodic tile set. Wang conjectured that this situation could never arise—a reasonable conjecture at the time, but one that we now know to be false. With this assumption in hand, he was able to formulate an algorithm that would decide the Tiling Problem for marked square tiles. The algorithm iterates over the positive integers. For each integer m, it constructs all possible $m \times m$ blocks of prototiles. If the given Wang tiles admit any periodic tilings, the algorithm will eventually find an m for which there exists an $m \times m$ translational unit. If not (and assuming that Case 4 is impossible), the algorithm will encounter an m for which no $m \times m$ block can be constructed that is consistent with the tile markings (the Extension Theorem guarantees the existence of such an m).

In 1966, Berger presented the first aperiodic tile set, a collection of over 20 000 Wang tiles that admit only nonperiodic tilings. Berger's discovery invalidates Wang's algorithm, because the algorithm assumes that a set of prototiles will tile via a finite translational unit or not at all. It was subsequently shown that Wang tiles could be used to simulate Turing machines [13], with a set of prototiles tiling the plane if and only if a corresponding Turing machine never halted. The halting problem is therefore reducible to the tiling problem, proving that the latter must be undecidable.

Since Berger's initial discovery, tiling theorists have sought smaller aperiodic sets of Wang tiles. The current record holder is a set of 13 prototiles [8]. Of course, smaller aperiodic sets exist outside the restrictions of Wang tiles, as the next section will demonstrate.

In computer graphics, Wang tiles are gaining popularity as a way to cover the plane with tiles while avoiding obvious repetition [4, 23]. In this context, aperiodicity is irrelevant; what matters is the ability to produce nonperiodic tilings efficiently. For this purpose, a set of eight Wang tiles can be described over two colours (or four, if we distinguish between horizontal and vertical occurences of the same colour), as

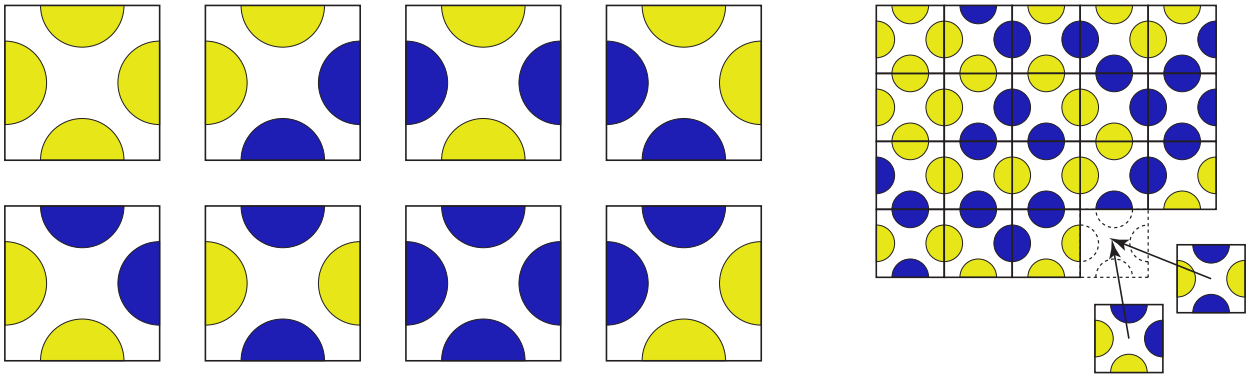


Figure 30: A "universal" set of Wang tiles with two edge colours. For any choices of colours to the north and west of a tile location, there are always two possible tiles that can be placed compatibly in that location.

shown in Figure 30. This set has the property that given a tile location with any two tiles adjacent across the northern and western borders, there are always two tiles that can be placed in that location without violating the matching conditions. We can therefore construct an arbitarily large patch of tiles without obvious repetitions simply by filling the region in row-major order, always choosing randomly from the two possibilities available at every step. The key in graphics applications is to fill the tiles with information (textures, sample positions, etc.) that embodies the underlying matching rules. Since the contents of the tiles are usually computed once ahead of time, it is relatively easy to suppress artifacts further by adding more tiles. Note that this technique also adapts naturally to three dimensions.

8.3 Penrose tilings

As was mentioned at the beginning of this section, the Penrose tile sets $P2$ (the "kite" and "dart") and $P3$ (thin and thick rhombs) are both aperiodic with suitable matching conditions (illustrated for $P3$ in Figure 25). Of course, proving the aperiodicity of $P2$ or $P3$ is difficult. First, one must establish that the matching conditions prevent the prototile set from admitting any periodic tilings. Then one must show that it *does* admit at least one nonperiodic tiling. Details of the proof can be found in Grünbaum and Shephard [16, Section 10.3].

Of course, for computer graphics purposes we are primarily concerned with the problem of filling arbitrary regions of the plane with interlocking tiles from either set; the matching rules are only of academic interest. There are several very different algorithms for laying out Penrose tilings. Some of these algorithms are themselves sophisticated results in algebra, geometry, and number theory. Details on the so-called "multigrid" and "lattice projection" methods can be found in the book by Senechal [28].

On the other hand, a rendering approach based on substitution leads to a far simpler implementation. Figure 31 shows substitution rules for $P2$ and $P3$, which can be applied recursively to draw finite patches of any size. These rules are very efficient in that they never lead to overlapping tiles, which need to be detected and coalesced. On the other hand, they are less than ideal for filling regions of the plane with tiles. The substituted tiles do not entirely cover their parents, and so it does not suffice simply to ensure that a given region is completely surrounded by a single tile—substitution might ultimately leave part of the region uncovered.

To obtain more control over the substitution process, we can use modified rules based on an analysis

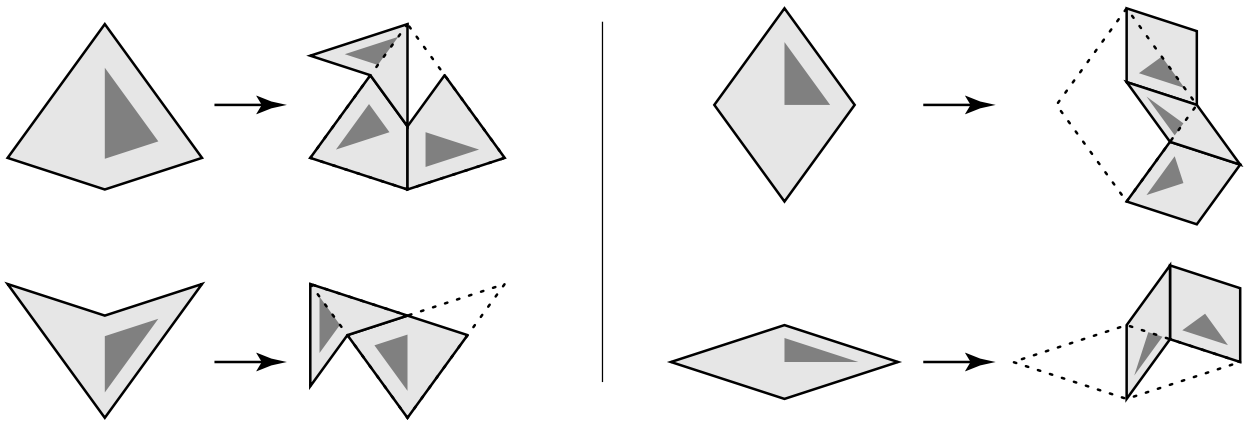


Figure 31: A set of substitution rules for Penrose's aperiodic tile sets $P2$ and $P3$. These rules are often presented in the style of the Lord tiling in Figure 28, where overlapping tiles must be pruned after substitution. Here, the rules fill the plane exactly, without introducing any overlaps. Shaded triangles are used to indicate the orientations of substituted tiles.

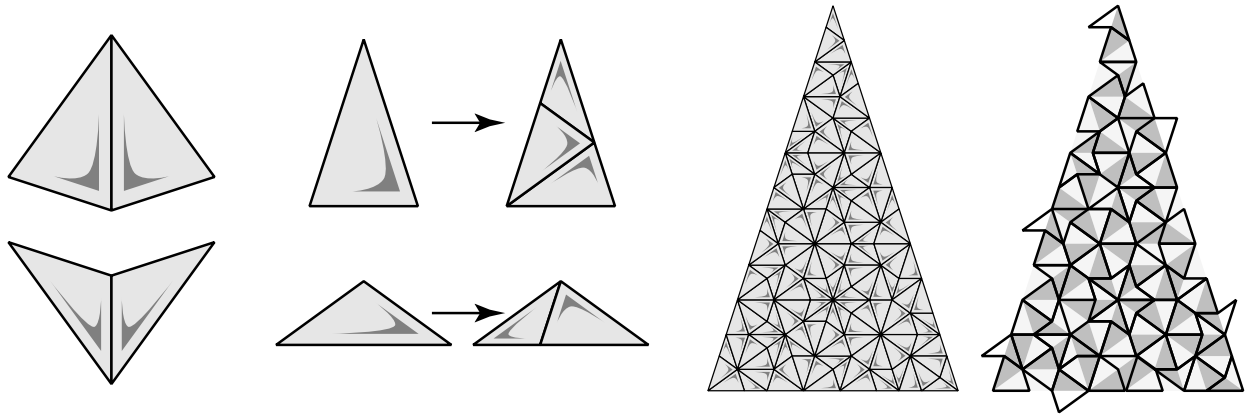


Figure 32: Modified substitution rules for Penrose's set $P2$. The kite and dart are each bisected into direct and reflected copies of isosceles triangles (orientation is indicated by a shaded marking). The substitutions for both triangles are shown. Repeated substitution exactly fills the initial patch of tiles. On the far right, the final configuration of kites and darts can be determined consistently from one of the orientations of each of the two triangles.

by Robinson, as described by Grünbaum and Shephard. Figure 32 illustrates the process for $P2$ (a similar argument applies to $P3$). Each prototile can be divided into left-handed and right-handed isosceles triangles by splitting them along lines of mirror reflection. These triangles can be given their own substitution rules. After any desired number of substitutions, the right-handed triangles can be discarded and the kites and darts can be recovered consistently from the left-handed triangles alone. The resulting patch will still fail to cover the entire starting tile, but the indentations can be worked around easily.

What about the shapes of the tiles themselves? The substitution systems above can provide us with a list of locations at which to draw prototiles. Geometric matching conditions such as those shown in Figure 25 suggest that we are free to choose two arbitrary **J** edge shapes to make up the outlines of the tiles.

Unfortunately, this interpretation of the possible shapes of Penrose tiles is limited, as can be seen in Grünbaum and Shephard's reproduction of Penrose's aperiodic chicken tiling [16, Figure 10.3.13]. They overlay the chickens with the corresponding unmodified tiling. The registration of these two tilings reveals that the chickens have tiling vertices that are different from those of the original tiling! Although these tiling vertices can be simulated by introducing degeneracies in the form of partially overlapping tile edges, there is some benefit in being able to parameterize the locations of these vertices directly.

However, we cannot simply provide a tiling vertex parameterization as was done in Section 7.3 for the isohedral tiling types. In a tiling by $P2$ or $P3$, copies of a given prototile will be surrounded in a finite number of different ways, causing tiling vertices to be located differently around its boundary. But it is possible to parameterize an extended set of *quasivertices*, points on a prototile's boundary that are tiling vertices anywhere in a tiling, or that are forced into existence by those tiling vertices. Quasivertex parameterizations are provided for $P2$ in Figure 33 and for $P3$ in Figure 34.

Given this extended set of quasivertices, we must revise the original matching conditions to account for the new tile edges that have been introduced. Taking inspiration from the use of incidence symbols in isohedral tilings, the possible edge shapes can be specified by labeling the edges around each tile and indicating adjacency rules for the labels. The edges of the kite and dart can be labeled $abcdaefghd$ and $ghefgcdabf$ respectively, where the enumerations start at the edges marked with arrows in Figure 33. To enforce matching between adjacent tiles, we require that the pairs (a, d), (b, h), (c, e), and (f, g) interlock. In effect, a given kite and dart will have only four non-congruent edge shapes between them. Similarly, we can label the edges of the thick and thin rhombs respectively as $abcdefegabhg$ and $afcdegabhg$, with the requirement that pairs (a, g), (b, e), (c, d), and (f, h) interlock. These labelings of the edges of the Penrose tiles are shown in Figure 35.

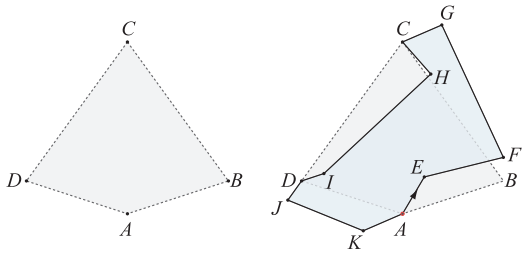
9 Survey

I conclude this document with a brief survey of some previous research and applications in computer graphics that make use of tiling theory.

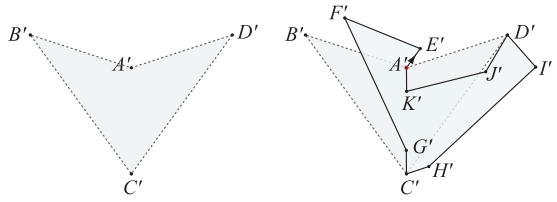
9.1 Drawing periodic tilings

Software specifically geared towards the construction of tilings of the plane has been around for nearly thirty years. For the most part, these tools are based on the Heesch tilings, which can be seen as a precursor to the isohedral tilings in which internal symmetries of tiles are not recognized (see Section 7.5).

Chow had a very successful FORTRAN program [2] that let the user input the portion of a tile's boundary that is "independent," *i.e.*, not constrained to some other part of the boundary through a symmetry

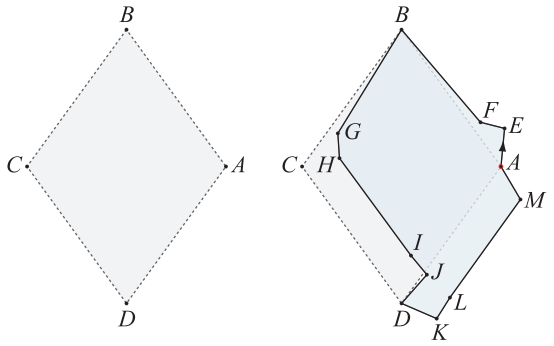


$$
\begin{aligned}
A &= (0, 0) \\
B &= (\cos(\tfrac{\pi}{10}), \sin(\tfrac{\pi}{10})) \\
C &= (0, \tfrac{1+\sqrt{5}}{2}) \\
D &= (-\cos\tfrac{\pi}{10}, \sin\tfrac{\pi}{10}) \\
\\
E &= A + r_1(\cos(\theta_1 + \tfrac{\pi}{10}), \sin(\theta_1 + \tfrac{\pi}{10})) \\
F &= B + r_2(\theta_2 + \cos(\tfrac{7\pi}{10}), \sin(\theta_2 + \tfrac{7\pi}{10})) \\
G &= C + r_1(\cos(\theta_1 - \tfrac{\pi}{10}), \sin(\theta_1 - \tfrac{\pi}{10}) \\
H &= \mathrm{rotate}(C, -\tfrac{2\pi}{5}, H) \\
I &= \mathrm{rotate}(C, -\tfrac{2\pi}{5}, G) \\
J &= \mathrm{rotate}(A, -\tfrac{4\pi}{5}, G) \\
K &= \mathrm{rotate}(A, -\tfrac{4\pi}{5}, F)
\end{aligned}
$$



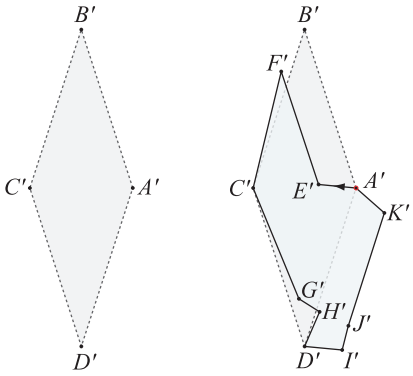
$$
\begin{aligned}
A' &= (0, 0) \\
B' &= (-\cos\tfrac{\pi}{10}, \sin\tfrac{\pi}{10}) \\
C' &= (0, -1) \\
D' &= (\cos(\tfrac{\pi}{10}), \sin(\tfrac{\pi}{10})) \\
\\
E' &= A' + r_2(\cos(\theta_2 + \tfrac{\pi}{2}), \sin(\theta_2 + \tfrac{\pi}{2})) \\
F' &= B' + r_1(\cos(\theta_1 - \tfrac{\pi}{10}), \sin(\theta_1 - \tfrac{\pi}{10})) \\
G' &= C' + r_2(\cos(\theta_2 + \tfrac{7\pi}{10}), \sin(\theta_2 + \tfrac{7\pi}{10})) \\
H' &= \mathrm{rotate}(C', -\tfrac{2\pi}{5}, G') \\
I' &= \mathrm{rotate}(C', -\tfrac{2\pi}{5}, F') \\
J' &= \mathrm{rotate}(A', -\tfrac{4\pi}{5}, F') \\
K' &= \mathrm{rotate}(A', -\tfrac{4\pi}{5}, E')
\end{aligned}
$$

Figure 33: A tiling vertex parameterization for generalized Penrose kites and darts, controlled by four real-valued parameters r_1, θ_1, r_2, and θ_2. The vertices are enumerated in counterclockwise order starting at A for the kite and A' for the dart. The function $\mathrm{rotate}(p, \theta, q)$ rotates point q by angle θ about point p.



$$
\begin{aligned}
A &= (\sin\tfrac{\pi}{5}, 0) \\
B &= (0, \cos\tfrac{\pi}{5}) \\
C &= (-\sin\tfrac{\pi}{5}, 0) \\
D &= (0, -\cos\tfrac{\pi}{5})
\end{aligned}
$$

$$
\begin{aligned}
E &= A + r_1(\cos(\tfrac{7\pi}{10} - \theta_1), \sin(\tfrac{7\pi}{10} - \theta_1)) \\
F &= A + r_2(\cos(\tfrac{7\pi}{10} - \theta_2), \sin(\tfrac{7\pi}{10} - \theta_2)) \\
G &= \mathrm{rotate}(B, -\tfrac{2\pi}{5}, F) \\
H &= \mathrm{rotate}(B, -\tfrac{2\pi}{5}, E) \\
I &= D + r_2(\cos(\tfrac{\pi}{2} - \theta_2), \sin(\tfrac{\pi}{2} - \theta_2)) \\
J &= D + r_1(\cos(\tfrac{\pi}{2} - \theta_1), \sin(\tfrac{\pi}{2} - \theta_1)) \\
K &= \mathrm{rotate}(D, -\tfrac{2\pi}{5}, J) \\
L &= \mathrm{rotate}(D, -\tfrac{2\pi}{5}, I) \\
M &= \mathrm{rotate}(D, -\tfrac{2\pi}{5}, H)
\end{aligned}
$$



$$
\begin{aligned}
A' &= (\sin\tfrac{\pi}{10}, 0) \\
B' &= (0, \cos\tfrac{\pi}{10}) \\
C' &= (-\sin\tfrac{\pi}{10}, 0) \\
D' &= (0, -\cos\tfrac{\pi}{10})
\end{aligned}
$$

$$
\begin{aligned}
E' &= A' + r_1(\cos(\tfrac{6\pi}{5} - \theta_1), \sin(\tfrac{6\pi}{5} - \theta_1)) \\
F' &= B' + r_2(\cos(\tfrac{7\pi}{5} - \theta_2), \sin(\tfrac{7\pi}{5} - \theta_2)) \\
G' &= \mathrm{rotate}(C', -\tfrac{4\pi}{5}, F') \\
H' &= D' + r_1(\cos(\tfrac{3\pi}{5} - \theta_1), \sin(\tfrac{3\pi}{5} - \theta_1)) \\
I' &= D' + r_1(\cos(\tfrac{\pi}{5} - \theta_1), \sin(\tfrac{\pi}{5} - \theta_1)) \\
J' &= \mathrm{rotate}(A', \tfrac{4\pi}{5}, F') \\
K' &= \mathrm{rotate}(A', \tfrac{4\pi}{5}, E')
\end{aligned}
$$

Figure 34: A tiling vertex parameterization for generalized Penrose rhombs, controlled by four real-valued parameters r_1, θ_1, r_2, and θ_2. The vertices are enumerated in counterclockwise order starting at A for the thick rhomb and A' for the thin rhomb.

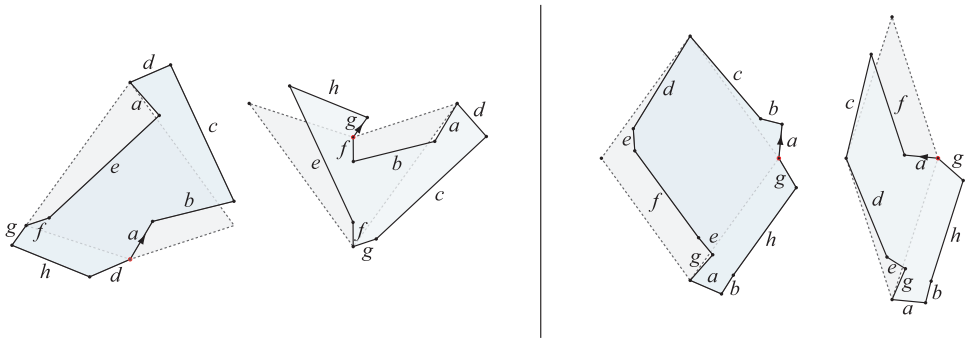


Figure 35: Edge labels for the tiling edges of the two sets of Penrose tiles, in the spirit of the incidence symbols used for the isohedral tilings. The kite and dart are shown on the left, and the two rhombs on the right. (The edge labels are not related between the two sets.) Pairs of labels correspond as described in the text.

of the tiling. The program then filled in the remaining part of the tile and replicated it in the plane. Chow also discussed possible applications of his software in manufacturing.

For many years, Kevin Lee has offered a commercial software package called TesselMania! that makes it easy to draw and decorate Escher-like tilings. His system is geared towards the use of tilings as a tool for mathematics education, and the most recent version of TesselMania! includes tutorials, games and puzzles designed for teaching concepts of geometry.

Tupper's Tess (`http://www.peda.com/tess/Welcome.html`) has traditionally allowed the user to create drawings belonging to the frieze and wallpaper groups. Recently, he modified Tess to support a set of tilings directly. Like TesselMania!, Tess is geared towards pedagogical use.

There exist many software tools for drawings based on symmetry groups. In many such tools it is easy to draw tilings, but there is no In many of these tools is it easy to draw tilings, but no explicit use of tiling theory and no tiling-like constraints imposed on the user. The classic example of such a tool is Kali (`http://www.geom.umn.edu/java/Kali/`). Weeks's KaleidoTile (`http://www.geometrygames.org/KaleidoTile/`) supports more drawing styles and non-Euclidean symmetry groups. A popular commercial product is Artlandia's Symmetry Works (`http://artlandia.com/products/SymmetryWorks/`), a plug-in for Adobe Illustrator.

9.2 Drawing nonperiodic tilings

Quasitiler (`http://www.geom.umn.edu/apps/quasitiler/`) was an application that made it possible to visualize a wide variety of nonperiodic tilings produced via the "lattice projection method". These included tilings by the Penrose rhombs, as well as generalizations of to rotational orders other than five. Quasitiler was originally written for the NeXT computer, and was then given a CGI-based web interface. Unfortunately, the back end is no longer in operation, and so Quasitiler is unavailable. Murray has created a Java implementation as part of a collection of screensavers (`http://screensavers.dev.java.net/`).

I have created a simple Java applet for exploring the parameterization of edge shapes and tiling vertices for Penrose tilings, as described in Section 8.3. It is available at `http://www.cgl.uwaterloo.ca/~csk/software/penrose/`.

9.3 Escher-like tilings

M.C. Escher had a lifelong fascination with tilings of the plane. He spent years filling a notebook with drawings of tilings by fish, birds, people, and dozens of other lifelike forms [27]. Escher has long been a source of inspiration for mathematicians and computer scientists; it is natural to ask whether computer graphics can assist in the creation of Escher-like tilings.

The software mentioned above for drawing tilings can be seen as implementing a "forward" process of experimentation similar to Escher's manual work. Starting from a simple tile shape with adjacency rules, the user can modify edges and the software will enforce constraints that preserve tileability. With skill, intuition, and luck, the user can eventually produce a tiling in the style of Escher's drawings.

Is it possible to design an "inverse" algorithm for tile design? In other words, given an arbitrary "goal shape", can an algorithm find a monohedral tiling of the plane by a prototile that resembles that shape? Kaplan and Salesin explored this question in the context of the isohedral tilings [21]. They developed an "Escherization" algorithm, a continuous optimization that searched over the parameterizations of the shapes isohedral prototiles as presented in Section 7.3. The objective function for the optimization realized the tile shape parameters as a polygon, and compared this polygon with the goal shape using an efficient L^2 distance metric [1]. The resulting algorithm could discover attractive Escher-like tilings from a variety of real-world goal shapes.

In later work, Kaplan and Salesin adapted the Escherization algorithm to several varieties of dihedral tilings [22]. Escher produced many dihedral tilings by beginning with one of his monohedral systems and dividing the prototile into two pieces with a path connecting two points on its boundary. They applied the same technique to the isohedral tilings, augmenting the isohedral shape parameterization with parameters controlling the end points and shape of a "splitting path". Here, the objective function constructs the two prototile shapes, compares each with its respective goal shape, and returns the maximum of the two comparisons. They showed that by restricting this representation to the special case of Dress's "Heaven and Hell patterns" [10], they could produce metamorphoses in the style of Escher's *Sky and Water*. They also applied the same optimization framework to build Escher-like tilings based on the shape parameterizations of the Penrose tilings in Section 8.3.

Escher also created a small number of carved wooden sculptures featuring spherical interpretations of his tesselations. Using these as a starting point, Yen and Séquin created an "Escher Sphere Construction Kit" [33], a system that allows the user to design ornamental spherical tilings much as one could create Euclidean tilings in the drawing tools above. Their software was based on Grünbaum and Shephard's classification of the isohedral tilings of the sphere [15], analogous to the planar classification discussed in Section 7.1. As an added feature, the tilings they create could be exported to rapid prototyping hardware and constructed as real artifacts.

9.4 Sampling

Recently, substitution tilings have emerged as a powerful technique for generating fast sampling patterns with blue noise properties. Typically, a single sample location or a set of sample locations is pre-computed for each tile. Given a probability distribution (such as an importance map), a sampling pattern is then generated by selectively applying the substitution rules selectively to individual tiles where greater sample density is needed.

Ostromoukhov et al. used this approach with Penrose tiles, demonstrating their technique in an environment mapping application [26]. They selectively apply rules from a substitution system derived from the Penrose rhombs, and place a sample in each remaining tile. Precomputed perturbation vectors are used

to displace the samples to improve their distribution, based on the local configuration around each tile. Most recently, Ostromoukhov presented a new sampling method based on selective subdivision of polyominoes [25]. The polyominoes avoid frequency-domain artifacts that arose in the use of Penrose tiles.

Kopf et al. developed a sampling method based on recursive subdivision of Wang tiles [24]. They generate a set of progressive Poisson distributions over each tile, which can be used for fine-grained control of sample density. Coarser changes can be handled by subdividing tiles. They can make random choices when placing every Wang tile in a grid, suppressing obvious repetition.

9.5 Texture generation

Another popular use of Wang tiles is to create non-repeating arrangements of textures or geometry. In these applications, each tile is decorated with a small fragment of an overall pattern, and tiles are joined together to extend the pattern over a large region. Clearly, a single tile can be decorated with a unrolled toroidal design to create a periodic pattern. In these techniques, the challenge is to ensure that an entire collection of decorated tiles can meet smoothly across their boundaries in multiple distinct configurations. This smoothness is usually achieved by representing a matching condition as a fragment of a pattern that straddles a tiling edge. Each Wang tile is then constructed by assembling the four fragments associated with its edge colours and merging those fragments somehow in the tile's interior.

Stam was the first to consider the use of Wang tiles for texturing [30]. He used a recursive subdividion scheme: a set of 16 Wang tiles, each with a substitution rule that satisfied the matching conditions. A recursive algorithm was therefore necessary to generate patches of tiles.

Cohen et al. used the simpler set of eight Wang tiles shown in Figure 30 [4], recognizing that the ability to assemble tiles stochastically is more relevant for graphics applications than true aperiodicity. The introduced the scanline algorithm for placing tiles, in which each new tile is chosen from the possibilities that are compatible with previously placed neighbours to the north and west. They also articulated the *corner problem*: Wang tiles with simple matching conditions have no control over their diagonal neighbours, a property which can lead to artifacts near tiling vertices. They solve this problem by augmenting the matching conditions with additional colour information for tile vertices, effectively increasing the number of colours used in the matching conditions. The combination of edge colours and vertex colours forces them to generate many different tiles. They apply their technique to texture synthesis and primitive distribution. More recently, Fu and Leung showed how this approach could be adapted to a quad-based parameterization of an arbitrary surface [12].

Lagae showed that s similar approach could be applied to Wang tiles in which matching conditions are expressed purely at the vertices rather than the edges. In their technique, a tile's vertices are given colours; four tiles meeting around a tiling vertex must have the same colour at that vertex. In this scheme, it is easier to control the continuity of information across tile corners (as well as edges). They also show that Wang tiles can be looked up with "random access": the identity of any tile in the plane can be computed numerically without first laying out all tiles to the north and west of it. A hash function is used to compute vertex colours reliably anywhere in the plane. The tile for a particular lattice square can be chosen based on the colours at its vertices.

References

[1] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell. An efficiently computable metric for comparing polygonal shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13:209–216, 1991.

[2] William W. Chow. Automatic generation of interlocking shapes. *Computer Graphics and Image Processing*, 9:333–353, 1979.

[3] Paul Church. Snakes in the plane. Master's thesis, School of Computer Science, University of Waterloo, 2008.

[4] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Trans. Graph.*, 22(3):287–294, 2003.

[5] John H. Conway, Heidi Burgiel, and Chaim Goodman-Strauss. *The Symmetries of Things*. A. K. Peters.

[6] H. S. M. Coxeter. Coloured symmetry. In H. S. M. Coxeter *et al.*, editor, *M.C. Escher: Art and Science*, pages 15–33. Elsevier Science Publishers B.V., 1986.

[7] Hallard T. Croft, Kenneth J. Falconer, and Richard K. Guy. *Unsolved Problems in Geometry*. Springer-Verlag, 1991.

[8] Karel Culik. An aperiodic set of 13 wang tiles. *Discrete Math.*, 160(1-3):245–251, 1996.

[9] Olaf Delgado Friedrichs. Data structures and algorithms for tilings i. 2003.

[10] Andreas W. M. Dress. The 37 combinatorial types of regular "Heaven and Hell" patterns in the euclidean plane. In H. S. M. Coxeter *et al.*, editor, *M.C. Escher: Art and Science*, pages 35–45. Elsevier Science Publishers B.V., 1986.

[11] David W. Farmer. *Groups and Symmetry: A Guide to Discovering Mathematics*. American Mathematical Society, 1996.

[12] Chi-Wing Fu and Man-Kang Leung. Texture tiling on arbitrary topological surfaces. In *Proceedings of Eurographics Symposium on Rendering 2005 (EGSR 2005)*, pages 99–104, June 2005.

[13] Andrew Glassner. Andrew Glassner's notebook: Penrose tiling. *IEEE Computer Graphics & Applications*, 18(4), jul–aug 1998. ISSN 0272-1716.

[14] Solomon W. Golomb. *Polyominoes: Puzzles, Patterns, Problems and Packings*. Princeton University Press, second edition, 1994.

[15] Branko Grünbaum and G. C. Shephard. Spherical tilings with transitivity properties. In Chandler Davis, Branko Grünbaum, and F. A. Sherk, editors, *The Geometric Vein: The Coxeter Festschrift*, pages 65–94. Springer-Verlag, New York, 1982.

[16] Branko Grünbaum and G. C. Shephard. *Tilings and Patterns*. W. H. Freeman, 1987.

[17] H. Heesch. Aufbau der ebene aus kongruenten bereichen. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen*, pages 115–117, 1935. John Berglund provides an online English translation at `http://www.angelfire.com/mn3/anisohedral/heesch35.html`.

[18] H. Heesch and O. Kienzle. *Flachenschluss*. Springer–Verlag, 1963.

[19] Daniel H. Huson. The generation and classification of tile-k-transitive tilings of the euclidean plane, the sphere, and the hyperbolic plane. *Geometriae Dedicata*, 47:269–296, 1993.

[20] Craig S. Kaplan. *Computer Graphics and Geometric Ornamental Design*. PhD thesis, Department of Computer Science & Engineering, University of Washington, 2002.

[21] Craig S. Kaplan and David H. Salesin. Escherization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques (SIGGRAPH 2000)*, pages 499–510. ACM Press/Addison-Wesley Publishing Co., 2000.

[22] Craig S. Kaplan and David H. Salesin. Dihedral Escherization. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 255–262. Canadian Human-Computer Communications Society, 2004.

[23] Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. Recursive wang tiles for real-time blue noise. *ACM Trans. Graph.*, 25(3):509–518, 2006.

[24] Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. Recursive wang tiles for real-time blue noise. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2006)*, 25(3):509–518, 2006.

[25] Victor Ostromoukhov. Sampling with polyominoes. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 78. ACM, 2007.

[26] Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. Fast hierarchical importance sampling with blue noise properties. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 488–495. ACM, 2004.

[27] Doris Schattschneider. *M.C. Escher: Visions of Symmetry*. W.H. Freeman, 1990.

[28] Marjorie Senechal. *Quasicrystals and Geometry*. Cambridge University Press, 1996.

[29] A. V. Shubnikov and V. A. Koptsik. *Symmetry in Science and Art*. Plenum Press, 1974.

[30] Jos Stam. Aperiodic texture mapping. Technical Report 01/97-R046, The Europeran Research Consotium for Informatics and Mathematics, 1997.

[31] Dorothy K. Washburn and Donald W. Crowe. *Symmetries of Culture*. University of Washington Press, 1992.

[32] Hermann Weyl. *Symmetry*. Princeton Science Library, 1989.

[33] Jane Yen and Carlo Séquin. Escher sphere construction kit. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 95–98. ACM Press, 2001.

Part III

Tile-Based Methods for Surface Modeling

Chapter 5

Slides

SIGGRAPH2008

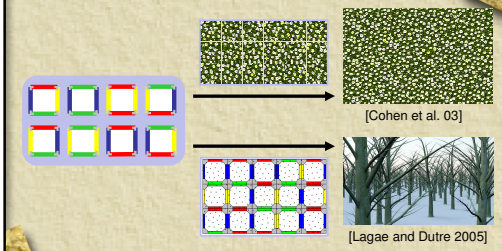
Part III: Tile-based Methods for Surface Modeling

Chi-Wing FU, Philip
Nanyang Technological University, Singapore

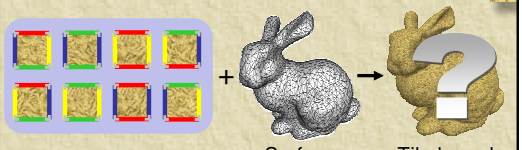
Outline of my part

- Motivation
- Texture Tiling on Surfaces
- More Applications!!!
  - Bump-mapped Tiles
  - BTF Tiles
  - Poisson disk Tiles

From the first part:

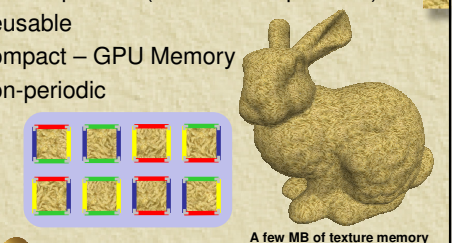


Can we use Wang Tiles on Surfaces?



If we can do so,

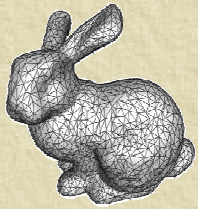
- Pre-computable (Surface Independent)
- Reusable
- Compact – GPU Memory
- Non-periodic



A few MB of texture memory

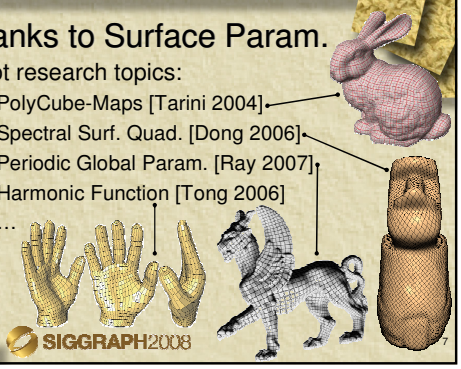
But wait…

- How to put square tiles on 3D models?
- 3D models are mostly triangular models.



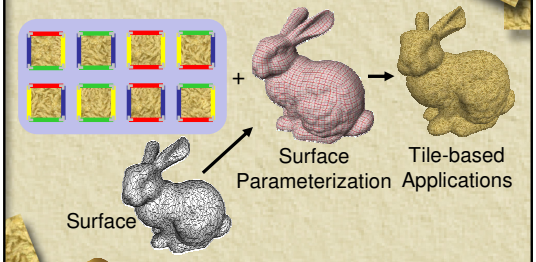
Thanks to Surface Param.

- Hot research topics:
  - PolyCube-Maps [Tarini 2004]
  - Spectral Surf. Quad. [Dong 2006]
  - Periodic Global Param. [Ray 2007]
  - Harmonic Function [Tong 2006]
  - …



SIGGRAPH2008 7

We apply surface param.



SIGGRAPH2008 8

Preview

Tiling RGB Textures Tiling Bump Maps Tiling the BTFs

SIGGRAPH2008 9

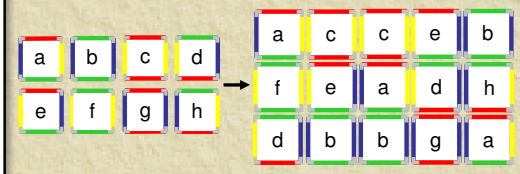
Outline of my part

- Motivation
- Texture Tiling on Surfaces
- More Applications!!!
  - Bump-mapped Tiles
  - BTF Tiles
  - Poisson disk Tiles

SIGGRAPH2008 10

Technical Challenges

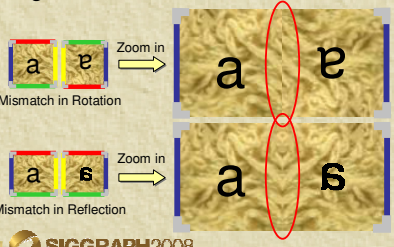
- Wang tiles are defined on 2D planes.



SIGGRAPH2008 11

Technical Challenges

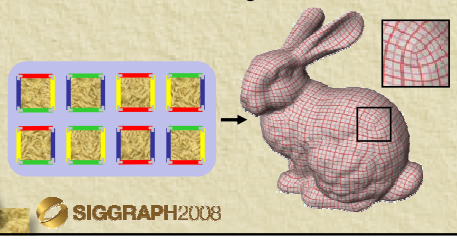
- Wang tiles cannot be rotated nor reflected



SIGGRAPH2008 12

Technical Challenges

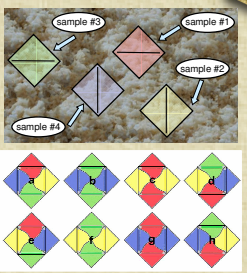
- However… For non-planar manifolds, we have to rotate Wang tiles



SIGGRAPH2008 13

Observation

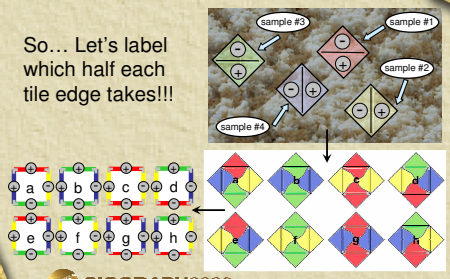
Note:
- Edge samples are divided into two halves
- Matching colors restore the textures



SIGGRAPH2008 14

Wang Tiles with Signs

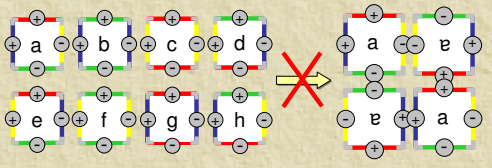
So… Let's label which half each tile edge takes!!!



SIGGRAPH2008 15

Wang Tiles with Signs

- Example Tiling: Matching only with color

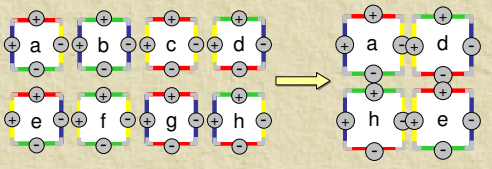


An invalid tiling

SIGGRAPH2008 16

Wang Tiles with Signs

- Example Tiling: Matching both color and sign

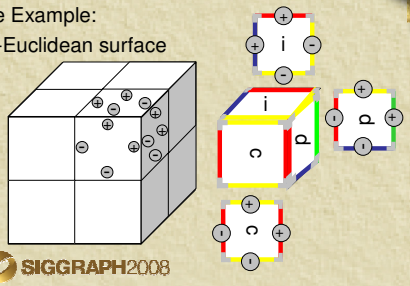


A valid tiling

SIGGRAPH2008 17

Wang Tiles with Signs

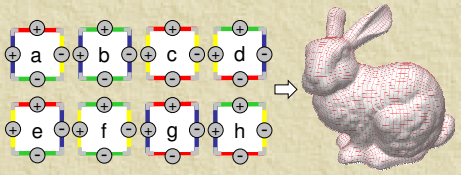
- More Example: Non-Euclidean surface



SIGGRAPH2008 18

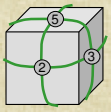
What's next?

- How can we **consistently** orientate tiles on the quad-based geometry?

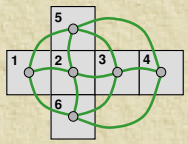


Orientate the Tiles

- Dual graph on quad-based geometry
  - Each Quad → A vertex in graph
  - Each Connection → An edge in graph



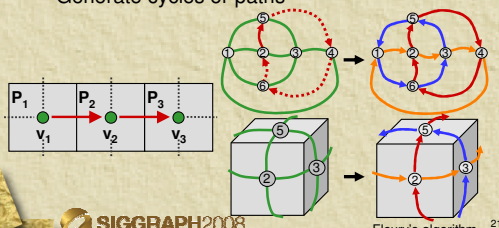
$|V| = 6, |E| = 12$



Unfolded view of a dual graph

Orientate the Tiles

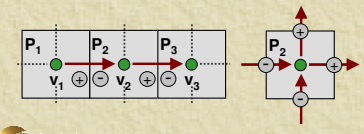
- Dual graph: make it a directed graph
  - Generate cycles or paths



Fleury's algorithm

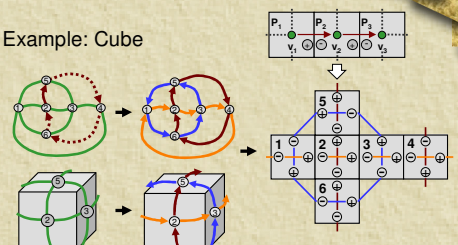
Orientate the Tiles

- After making a directed graph
  - Assign signs on edges:
    +ve for out-going and **–ve for incoming**
  - Each quad: exactly **two +ve** and **two –ve signs**



Orientate the Tiles

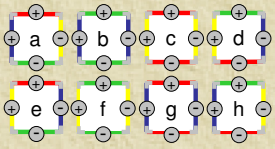
- Example: Cube

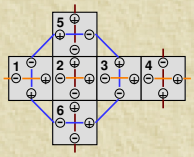


A valid sign assignment on a cube

Orientate the Tiles

- Example: Cube

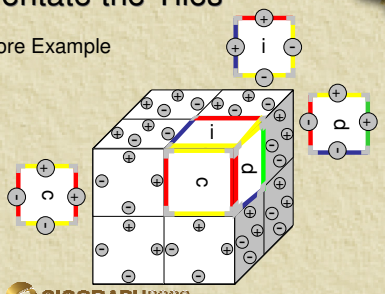




A valid sign assignment on a cube

Orientate the Tiles

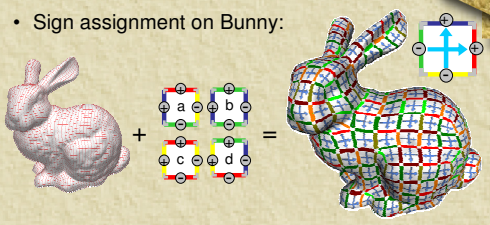
- More Example



SIGGRAPH2008 25

Tiling

- Sign assignment on Bunny:

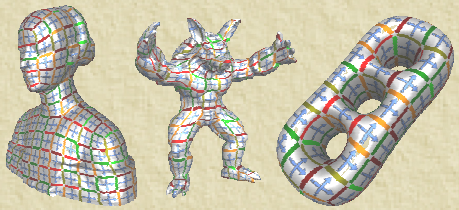


Arrows on tiles indicate the signs

SIGGRAPH2008 26

Tiling

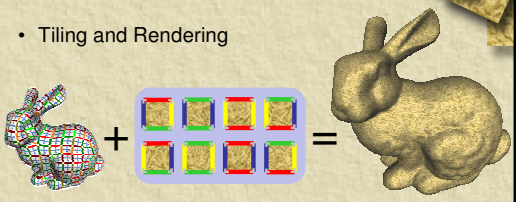
- More examples on sign assignment



SIGGRAPH2008 27

Results

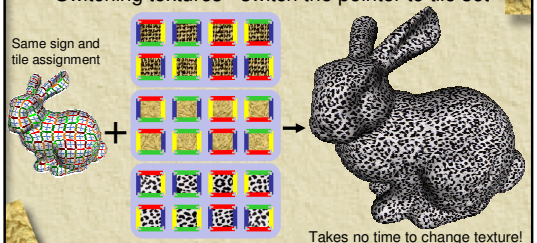
- Tiling and Rendering



SIGGRAPH2008 28

Results

- Switching textures - switch the pointer to tile set



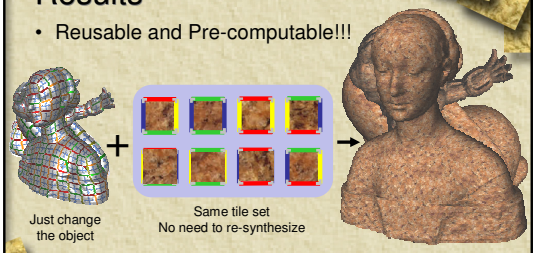
Same sign and tile assignment

Takes no time to change texture!

SIGGRAPH2008 29

Results

- Reusable and Pre-computable!!!



Just change the object

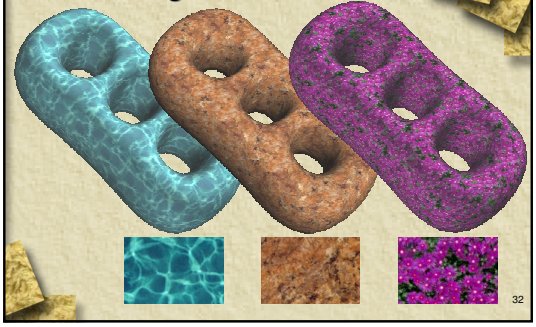
Same tile set
No need to re-synthesize

SIGGRAPH2008 30

Rendering Results: Bunny

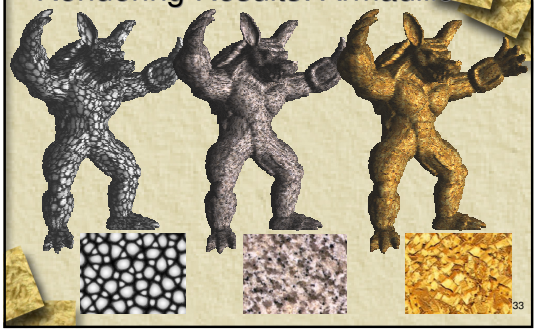
31

Rendering Results: Holes3



32

Rendering Results: Armadillo



33

Rendering Results: Laurana

34

Outline of my part

- Motivation
- Texture Tiling on Surfaces
- More Applications!!!
  - Bump-mapped Tiles
  - BTF Tiles
  - Poisson disk Tiles

SIGGRAPH2008

35

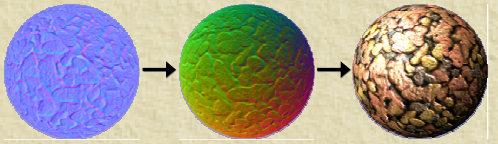
#2 - Bump-mapped tiles

SIGGRAPH2008

36

Bump-mapped tiles

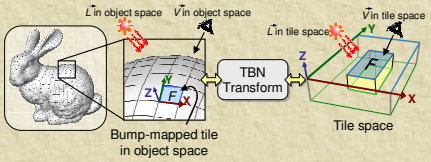
- Normals are in the texture/tile space
  Need to be transformed to object space



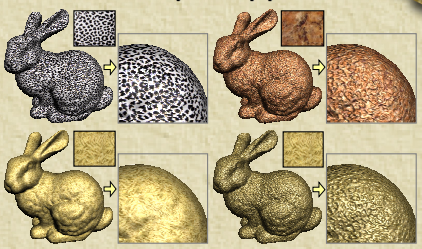
Normals in tile space Normals in object space Rendering Result

Bump-mapped tiles

- Per-tile **tangent basis transform** (TBN) to transform the normals from **texture/tile space** to **object space**



Results: Bump-mapped tiles



No Bump-mapped tiles With Bump-mapped tiles

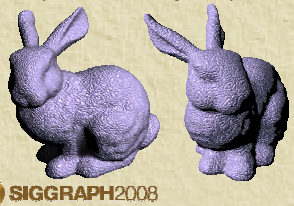
Results: Bump-mapped tiles

No Bump-mapped tiles With Bump-mapped tiles

#3 - BTF tiles

BTF Tiling

- How about making BRDF or BTF tiles?
  - Higher quality and more realistic lighting effects
  - View dependent and light dependent

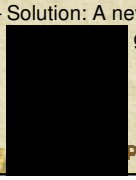


Technical Challenges

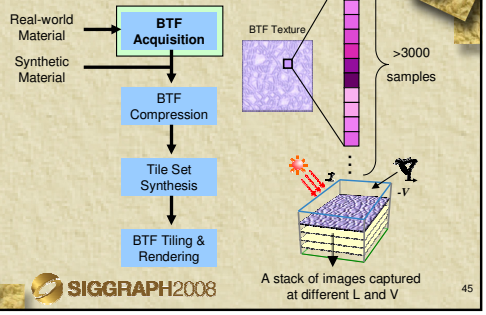
- The BTF data is very large in size
- The BTF Tile Synthesis is hard
  - It's 6D, as compared to 2D for color textures
  - Each pixel has a 4D BRDF

Any Solution?

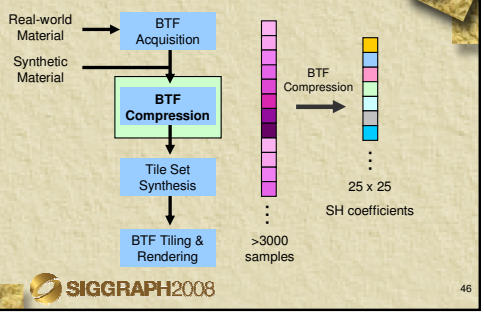
- For the BTF data
  - Solution step 1: Double SH compression
  - Solution step 2: Quantization
- For the BTF tile synthesis
  - Solution: A new tile synthesis method that generates the BTF tile set



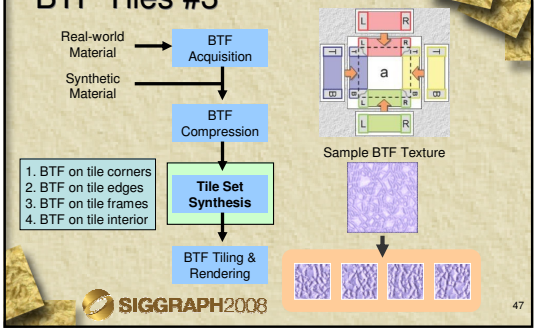
BTF Tiles #1



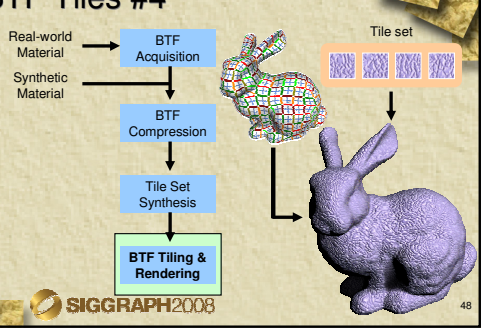
BTF Tiles #2



BTF Tiles #3

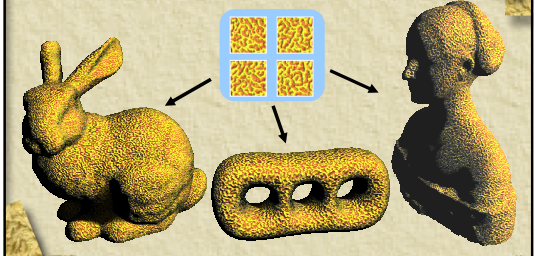


BTF Tiles #4



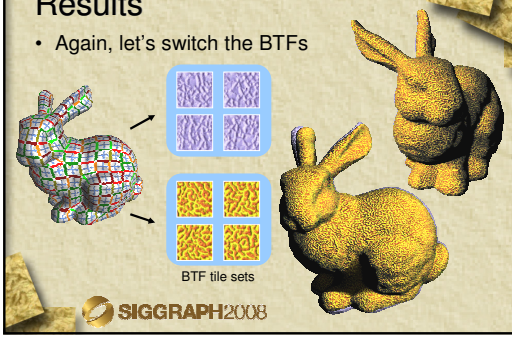
Results

- Reusable: Same tile set on different models



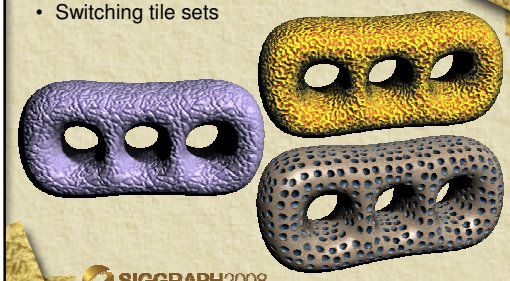
Results

- Again, let's switch the BTFs



Results

- Switching tile sets



#4 – Poisson disk tiles

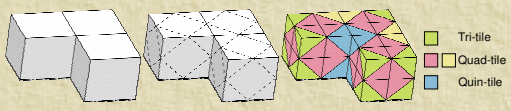
How about point distribution?

- Applications:
  - Pattern distribution, BTF sample, geometry, etc.

Our Approach

First, make it a dual surface

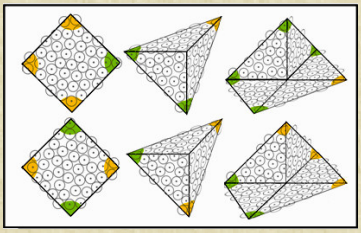
All tile junctions are shared by exactly 4 quads



Now, all corners meet four edges!!!

Dual Poisson-disk Tiling

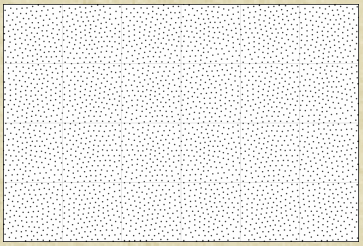
Then, create a dual tile set: Interior tiles



55

Dual Poisson-disk Tiling

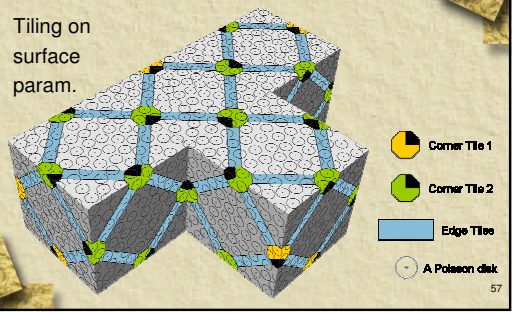
Then, dual tiling on 2D plane



56

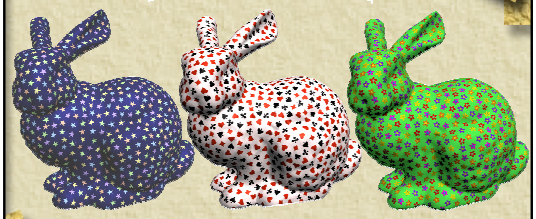
Dual Poisson-disk Tiling

Tiling on surface param.



57

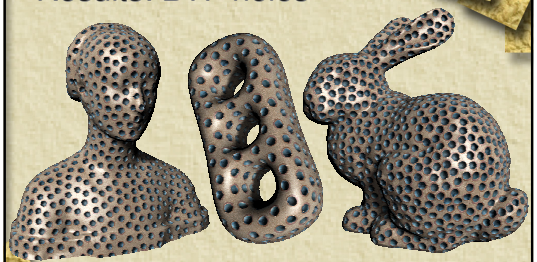
Results: Distribute Patterns



SIGGRAPH2008

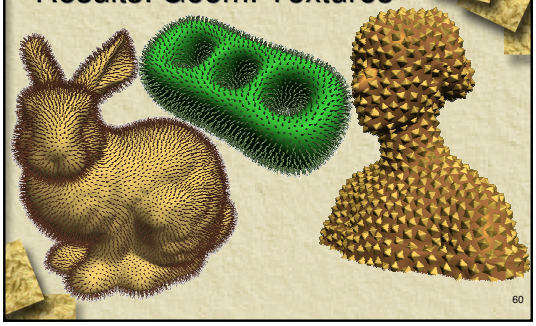
58

Results: BTF holes



59

Results: Geom. Textures



60

Conclusion: One Formula!

Data + Tile set + Surface Tiling ⇨ Applications

- Color texture tiles
- Bump-mapped tiles
- BTF Tiles
- Dual Poisson -disk tiles

Acknowledgments

I would like to thank

- Marco Tarini: the PolyCube-Mapped models and the associated fragment program.
- Nicolas Ray: the Periodic Global Param. models
- Bonn Database and MSRA: BTF data
- My Collaborators: Raymond Leung, Tien-Tsin Wong, Raymond Pang, Felix Lo, Hongwei Li, and Pheng-Ann Heng

Thank you!!!

- **Texture Tiling on Arbitrary Topological Surfaces using Wang Tiles.** Chi-Wing Fu and Man-Kang Leung, EGSR 2005.
- **Tileable BTF.** Man-Kang Leung, Wai-Man Pang, Chi-Wing Fu, Tien-Tsin Wong, and Pheng-Ann Heng. IEEE TVCG 2007.
- **Dual Poisson-Disk Tiling: An Efficient Method for Distributing Features on Arbitrary Surfaces.** Hongwei Li, Kui-Yip Lo, Man-Kang Leung, and Chi-Wing Fu, IEEE TVCG 2008.

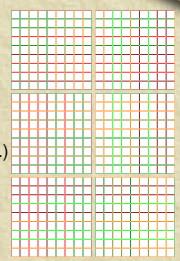
Backup Slides

For polycube models:

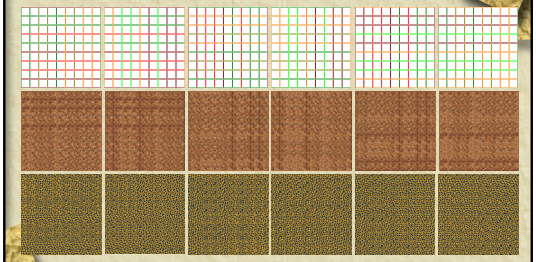
- Number of cycles: 3
- Number of colors per cycle: 3
- Number of tiles = $3^2 \times 3^2 \times 6$
- An example tile set:
  - Either 6 RGB textures OR
  - 6 index texture (2 RGB int. Tex.) + 1 RGB data texture

    Using OpenGL texture rect.

    Resolution: (64x9) x (64x9)



Two example Tile sets



For polycube models:

- Corresponding Results:

SIGGRAPH2008 67

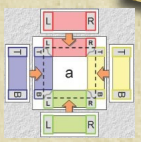
BTF Tiles #1

Real-world Material → BTF Acquisition

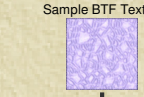
Synthetic Material → BTF Compression

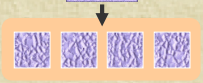
1. BTF on tile corners
2. BTF on tile edges
3. BTF on tile frames
4. BTF on tile interior

Tile Set Synthesis → BTF Tiling & Rendering



Sample BTF Texture





SIGGRAPH2008 68

BTF Tiles #2

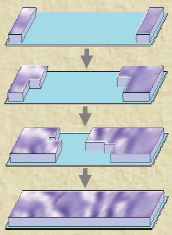
Real-world Material → BTF Acquisition

Synthetic Material → BTF Compression

1. BTF on tile corners
2. BTF on tile edges
3. BTF on tile frames
4. BTF on tile interior

Tile Set Synthesis → BTF Tiling & Rendering

One BTF edge for each color in Wang tiles



SIGGRAPH2008 69

BTF Tiles #3

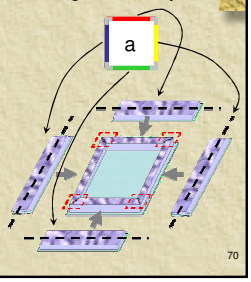
Real-world Material → BTF Acquisition

Synthetic Material → BTF Compression

1. BTF on tile corners
2. BTF on tile edges
3. BTF on tile frames
4. BTF on tile interior

Tile Set Synthesis → BTF Tiling & Rendering

Synthesize BTF Borders with edges as boundary



SIGGRAPH2008 70

BTF Tiles #4

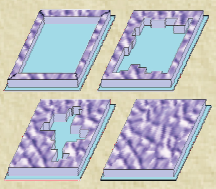
Real-world Material → BTF Acquisition

Synthetic Material → BTF Compression

1. BTF on tile corners
2. BTF on tile edges
3. BTF on tile frames
4. BTF on tile interior

Tile Set Synthesis → BTF Tiling & Rendering

A BTF Tile

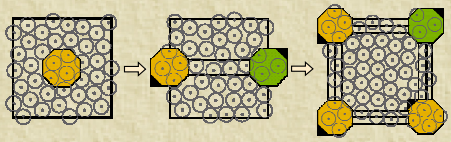


SIGGRAPH2008 71

Dual Poisson-disk Tiling

Now, create a dual tile set:

– Corner tiles → Edge tiles → Interior tiles



SIGGRAPH2008 72

Part IV

Non-Periodic Tilings for Computer Graphics Applications

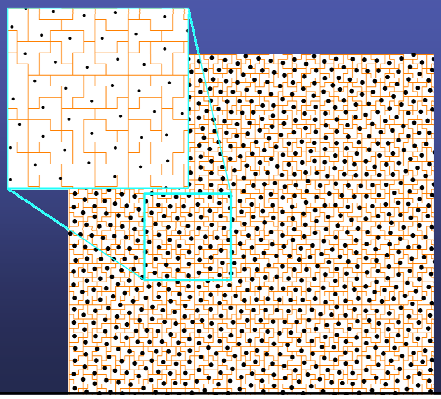
Chapter 6

Slides

Sampling Systems Based on Non-Periodic Tilings

Victor Ostromoukhov

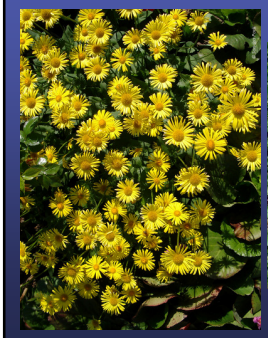
University of Montreal



Presentation Outline

- Motivations
- Importance Sampling System Based on Penrose Tiling
- Sampling with Polyominoes
- 2D Low-Discrepancy Sequences Based on Dodecagonal Tiling
- Conclusions and Challenges

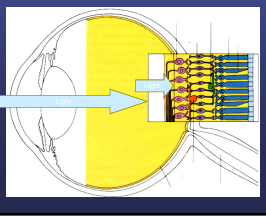
What is Even Distribution?

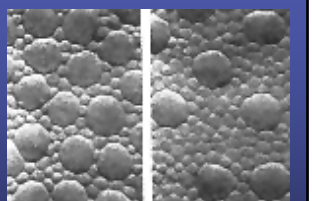


In Nature

What is Even Distribution?

In Human Eye

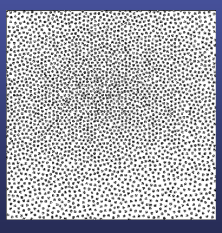


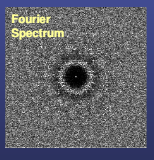


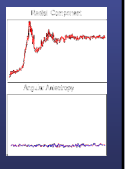
Distribution of Cones and Rods

Rods Cones

Blue-Noise Sampling







Problem Statement

Given:

- Importance Density $I(x,y)$

high

low

Problem Statement

Find:

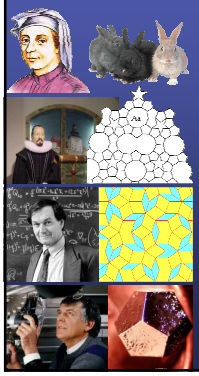
- Discrete Sample Distribution Locally Proportional to $I(x,y)$

Importance Sampling System Based on Penrose Tiling

- Extension to Penrose Tiling
- Fibonacci Number System
- Adaptive Subdivision
- Structural Indices
- Corrective Vectors Lookup Table

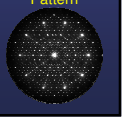
Ref: Ostromoukhov, V., Donohue, C., and Jodoin, P.-M. (2004) Fast hierarchical importance sampling with blue noise properties. ACM Transactions on Graphics, 23(3). Proc. SIGGRAPH 2004.

Penrose Tiling: Milestones

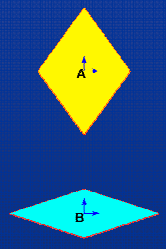


- Circa 1200 AD, Fibonacci (Leonardo of Pisa)
  Rabbit Sequence, Fibonacci Numbers
- 1619, Johannes Kepler
  Harmonice Mundi, 5-fold Tiling Problem
- 1974, Sir Roger Penrose
  Pentaplexity, Penrose Tiling
- 1984, Dan Shechtman *et al.*
  Discovery of Quasi-Crystals

Diffraction Pattern

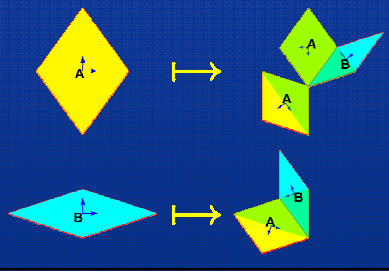


Penrose Tiling: Definition by Production Rules

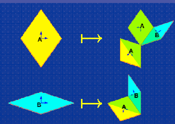


Penrose Tiling: Definition by Production Rules

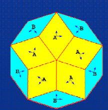
$$\mathcal{P}_{Penrose} := \begin{cases} A \mapsto \{A, B, A\} \\ B \mapsto \{A, B\} \end{cases}$$



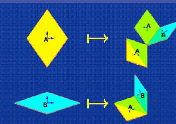
Penrose Tiling:
Definition by Production Rules



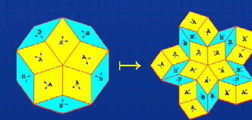
$$\mathcal{P}_{Penrose} := \begin{cases} A \mapsto \{A, B, A\} \\ B \mapsto \{A, B\} \end{cases}$$



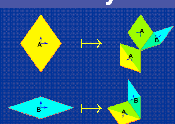
Penrose Tiling:
Definition by Production Rules



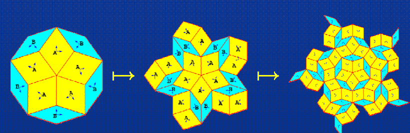
$$\mathcal{P}_{Penrose} := \begin{cases} A \mapsto \{A, B, A\} \\ B \mapsto \{A, B\} \end{cases}$$



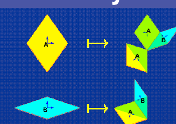
Penrose Tiling:
Definition by Production Rules



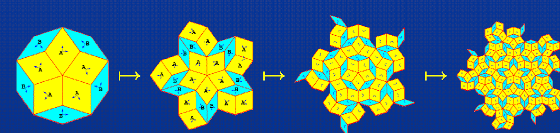
$$\mathcal{P}_{Penrose} := \begin{cases} A \mapsto \{A, B, A\} \\ B \mapsto \{A, B\} \end{cases}$$



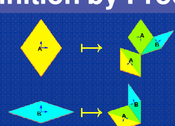
Penrose Tiling:
Definition by Production Rules



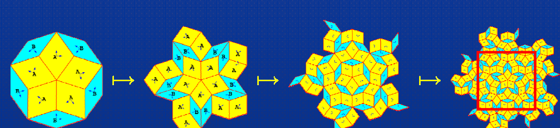
$$\mathcal{P}_{Penrose} := \begin{cases} A \mapsto \{A, B, A\} \\ B \mapsto \{A, B\} \end{cases}$$



Penrose Tiling:
Definition by Production Rules

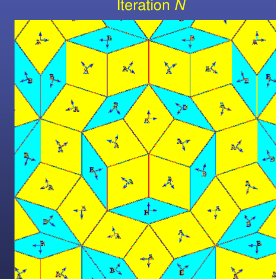


$$\mathcal{P}_{Penrose} := \begin{cases} A \mapsto \{A, B, A\} \\ B \mapsto \{A, B\} \end{cases}$$



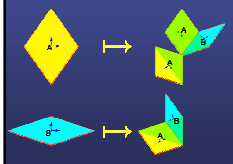
Penrose Tiling:
Definition by Production Rules

Iteration *N*

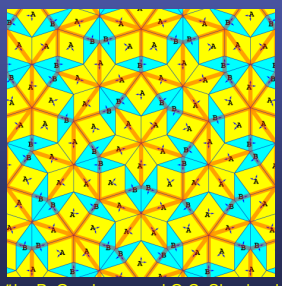


Penrose Tiling: Definition by Production Rules

$\mathcal{P}_{Penrose} := \begin{cases} A \mapsto \{A, B, A\} \\ B \mapsto \{A, B\} \end{cases}$



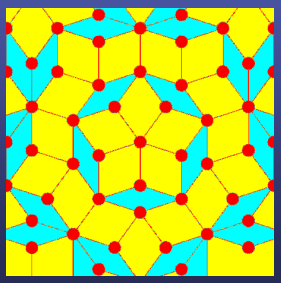
Iteration *N+1*



Ref: *"Tilings and Patterns"* by B. Grunbaum and G.C. Shephard

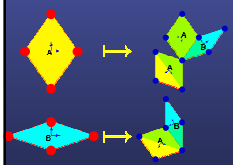
Penrose Tiling: Vertices

Iteration *N*

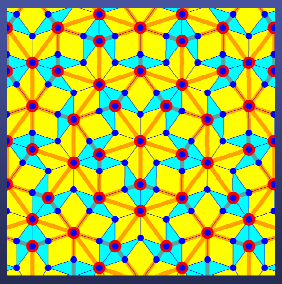


Penrose Tiling: Vertices

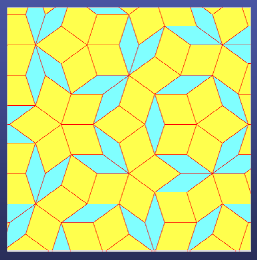
$\mathcal{P}_{Penrose} := \begin{cases} A \mapsto \{A, B, A\} \\ B \mapsto \{A, B\} \end{cases}$



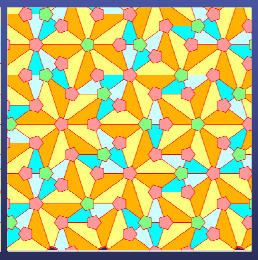
Iteration *N+1*



Extension of Penrose Tiling

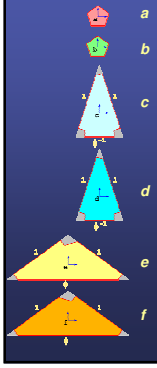


Original

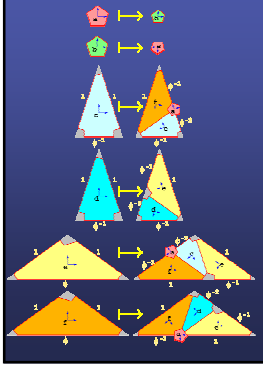


Our Extension

Extension of Penrose Tiling



Extension of Penrose Tiling

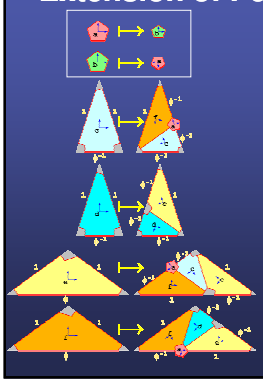


$\mathcal{P} := \begin{cases} a \mapsto \{b\} \\ b \mapsto \{a\} \\ c \mapsto \{f, c, a\} \\ d \mapsto \{e, d\} \\ e \mapsto \{f, c, e, a\} \\ f \mapsto \{e, d, f, a\} \end{cases}$

$\frac{A_{new}}{A_{old}} = \phi^2$

$\phi = \frac{1+\sqrt{5}}{2}$ *(Golden Ratio)*

Extension of Penrose Tiling

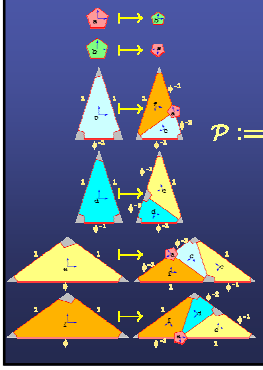


$$\mathcal{P} := \begin{cases} a \mapsto \{b\} \\ b \mapsto \{a\} \\ c \mapsto \{f, c, a\} \\ d \mapsto \{e, d\} \\ e \mapsto \{f, c, e, a\} \\ f \mapsto \{e, d, f, a\} \end{cases}$$

$$\frac{A_{new}}{A_{old}} = \phi^2$$

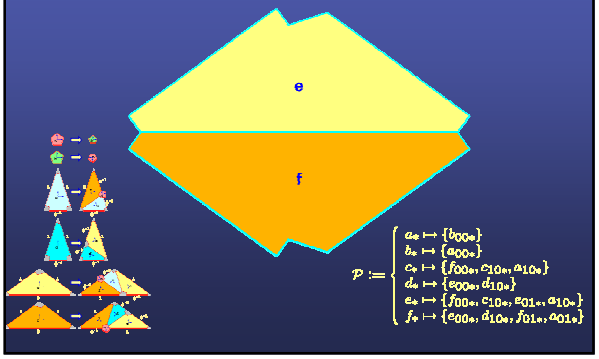
$\phi = \frac{1+\sqrt{5}}{2}$ (Golden Ratio)

Fibonacci Number System



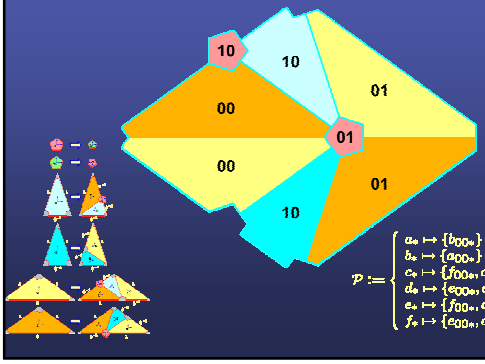
$$\mathcal{P} := \begin{cases} a_* \mapsto \{b_{00*}\} \\ b_* \mapsto \{a_{00*}\} \\ c_* \mapsto \{f_{00*}, c_{10*}, a_{10*}\} \\ d_* \mapsto \{e_{00*}, d_{10*}\} \\ e_* \mapsto \{f_{00*}, c_{10*}, e_{01*}, a_{10*}\} \\ f_* \mapsto \{e_{00*}, d_{10*}, f_{01*}, a_{01*}\} \end{cases}$$

Fibonacci Number System



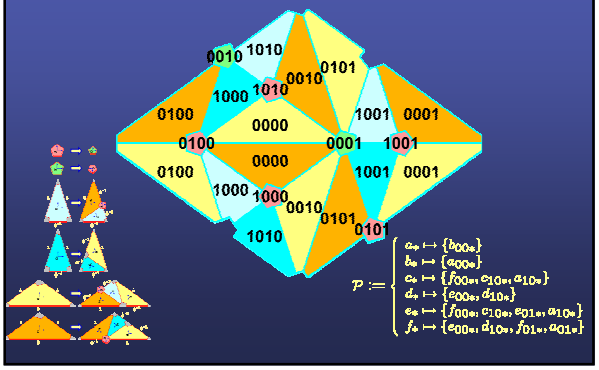
$$\mathcal{P} := \begin{cases} a_* \mapsto \{b_{00*}\} \\ b_* \mapsto \{a_{00*}\} \\ c_* \mapsto \{f_{00*}, c_{10*}, a_{10*}\} \\ d_* \mapsto \{e_{00*}, d_{10*}\} \\ e_* \mapsto \{f_{00*}, c_{10*}, e_{01*}, a_{10*}\} \\ f_* \mapsto \{e_{00*}, d_{10*}, f_{01*}, a_{01*}\} \end{cases}$$

Fibonacci Number System



$$\mathcal{P} := \begin{cases} a_* \mapsto \{b_{00*}\} \\ b_* \mapsto \{a_{00*}\} \\ c_* \mapsto \{f_{00*}, c_{10*}, a_{10*}\} \\ d_* \mapsto \{e_{00*}, d_{10*}\} \\ e_* \mapsto \{f_{00*}, c_{10*}, e_{01*}, a_{10*}\} \\ f_* \mapsto \{e_{00*}, d_{10*}, f_{01*}, a_{01*}\} \end{cases}$$

Fibonacci Number System



$$\mathcal{P} := \begin{cases} a_* \mapsto \{b_{00*}\} \\ b_* \mapsto \{a_{00*}\} \\ c_* \mapsto \{f_{00*}, c_{10*}, a_{10*}\} \\ d_* \mapsto \{e_{00*}, d_{10*}\} \\ e_* \mapsto \{f_{00*}, c_{10*}, e_{01*}, a_{10*}\} \\ f_* \mapsto \{e_{00*}, d_{10*}, f_{01*}, a_{01*}\} \end{cases}$$

Fibonacci Number System

Ref: *"The Art of Computer Programming, Vol. 1,"* by D.E. Knuth

- Binary Number System:

$$n = (a_m a_{m-1} \ldots a_1 a_0)_2 \iff n = \sum_{j=0}^{m} a_j 2^j.$$

- Fibonacci Number System:

$$n = (b_m b_{m-1} \ldots b_3 b_2)_F \iff n = \sum_{j=2}^{m} b_j F_j.$$

- Fibonacci Numbers:

$F_i = F_{i-1} + F_{i-2}$ $\{0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots\}$

Fibonacci Number System

Binary Number System

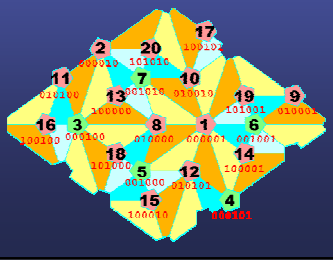
N_{10}	2^3 (8)	2^2 (4)	2^1 (2)	2^0 (1)
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0

Fibonacci Number System

N_{10}	F_6 (8)	F_5 (5)	F_4 (3)	F_3 (2)	F_2 (1)
0	0	0	0	0	0
1	0	0	0	0	1
2	0	0	0	1	0
3	0	0	1	0	0
4	0	0	1	0	1
5	0	1	0	0	0
6	0	1	0	0	1
7	0	1	0	1	0
8	1	0	0	0	0
9	1	0	0	0	1
10	1	0	0	1	0
11	1	0	1	0	0
12	1	0	1	0	1

Fibonacci Number System

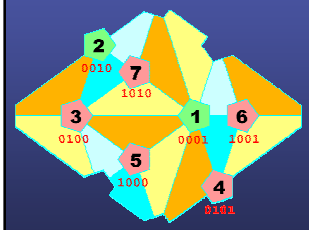
$(00001)_F = 1$, $(00010)_F = 2$, $(00100)_F = 3$,
$(00101)_F = 4$, $(01000)_F = 5$, $(01001)_F = 6$,
$(01010)_F = 7$, $(10000)_F = 8$, $(10001)_F = 9$,
$(10010)_F = 10$, $(10100)_F = 11$, $(10101)_F = 12$.



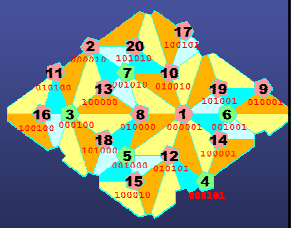
Pentagonal Tiles Only

Fibonacci Number System

Iteration *4*



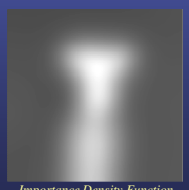
Iteration *5*



Adaptive Subdivision

$$\lambda(x) = \log_{\phi^2}(x) + C$$

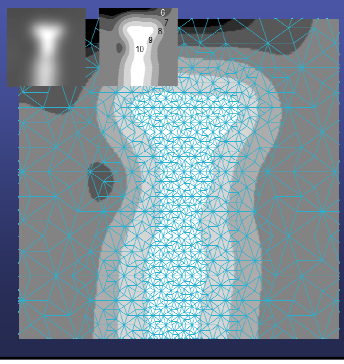
$$\kappa = \lceil \max_{tile} \lambda(mag \cdot I(x,y)) \rceil$$



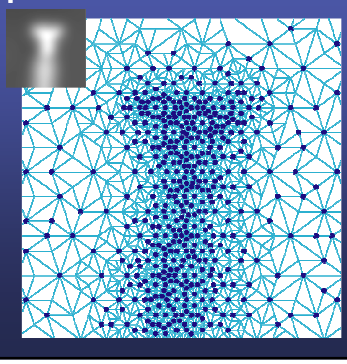
Importance Density Function
$I(x,y)$

$\lceil \lambda(I(x,y)) \rceil$

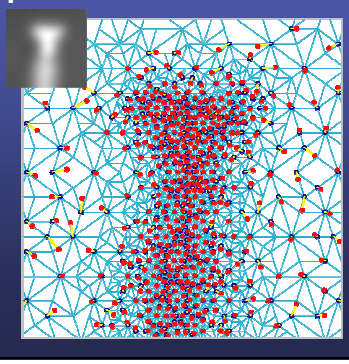
Adaptive Subdivision



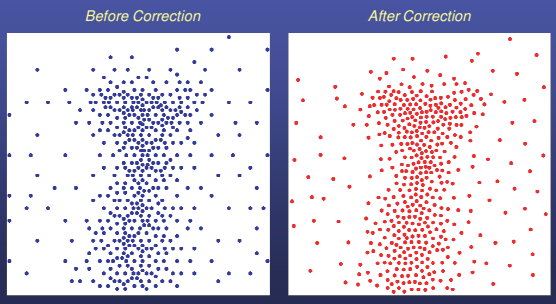
Corrective Vectors
Lookup table



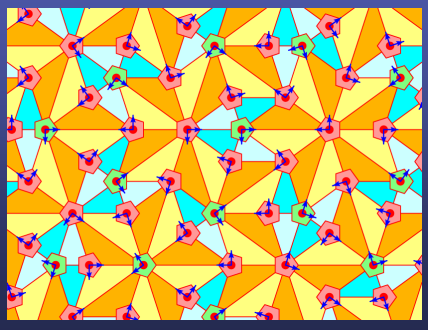
Corrective Vectors Lookup table



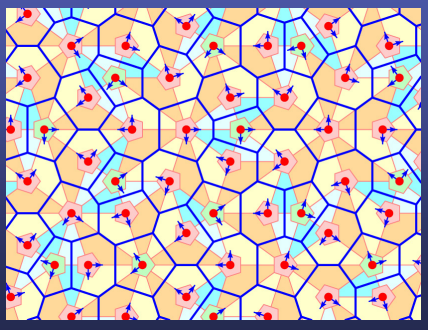
Corrective Vectors Lookup table



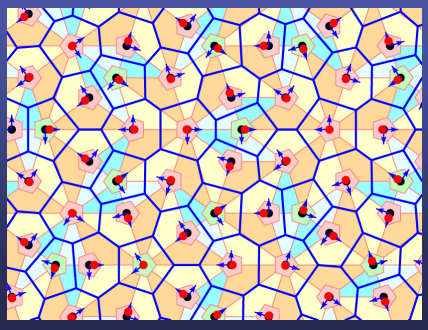
Offline Lloyd's Relaxation: init pts + basis frames



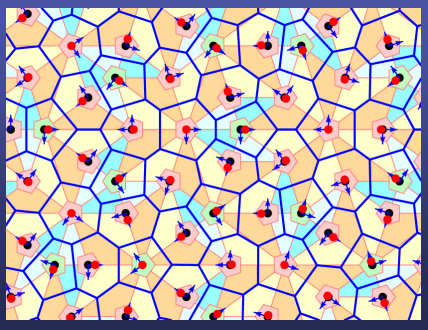
Offline Lloyd's Relaxation: init pts + basis frames



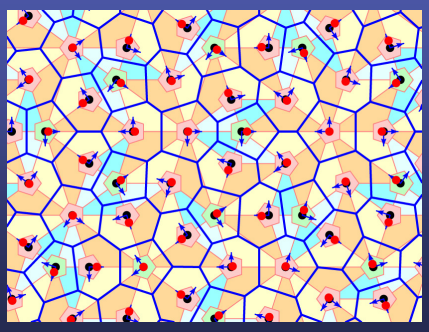
Offline Lloyd's Relaxation: iter 1



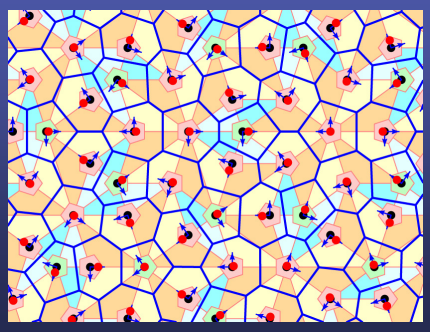
Offline Lloyd's Relaxation: iter 2



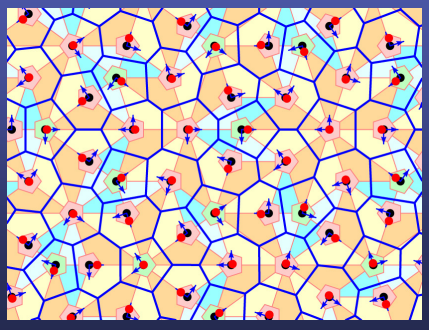
Offline Lloyd's Relaxation:
iter 3



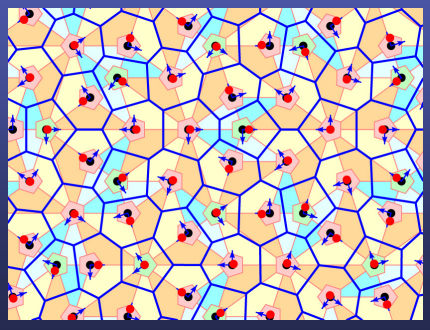
Offline Lloyd's Relaxation:
iter 4



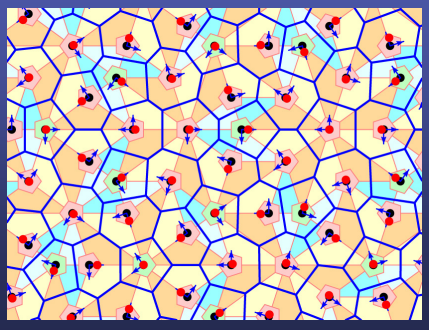
Offline Lloyd's Relaxation:
iter 5



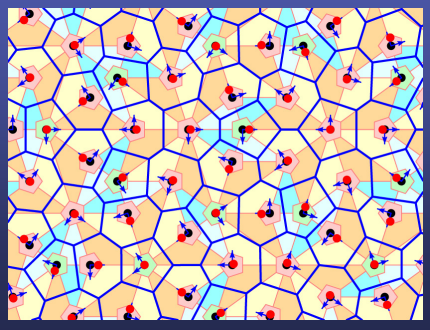
Offline Lloyd's Relaxation:
iter 6



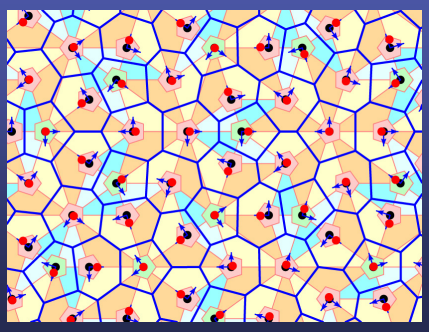
Offline Lloyd's Relaxation:
iter 7



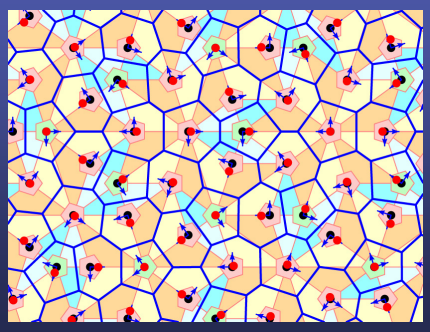
Offline Lloyd's Relaxation:
iter 8



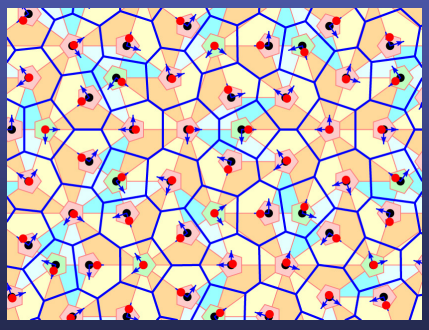
Offline Lloyd's Relaxation: iter 9



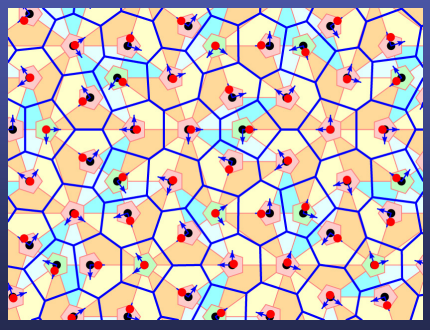
Offline Lloyd's Relaxation: iter 10



Offline Lloyd's Relaxation: iter 11



Offline Lloyd's Relaxation: iter 12



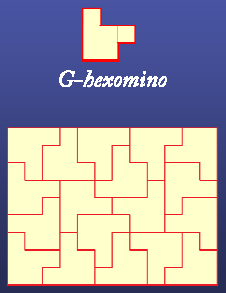
Presentation Outline

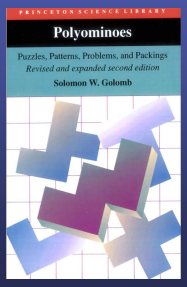
- Motivations
- Importance Sampling System Based on Penrose Tiling
- Sampling with Polyominoes
- 2D Low-Discrepancy Sequences Based on Dodecagonal Tiling
- Conclusions and Challenges

Polyominoes

Ref: Sampling with Polyominoes [SIGGRAPH'07]

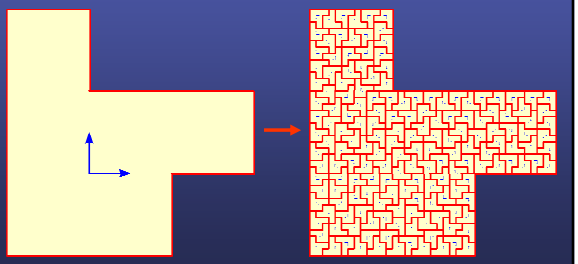
Rectifiable polyominoes



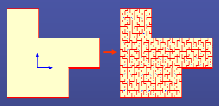


Self-similar (A-rep) rectifiable polyominoes, or reptiles

9^2-rep or 81-rep G-hexominoes
Area scaling factor $\mathcal{A} = 9^2 = 81$



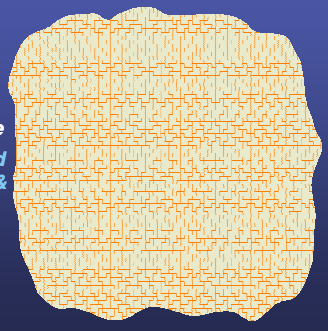
$\mathcal{A}$ -rep rectifiable polyominoes, or reptiles



- *Unique production rule*

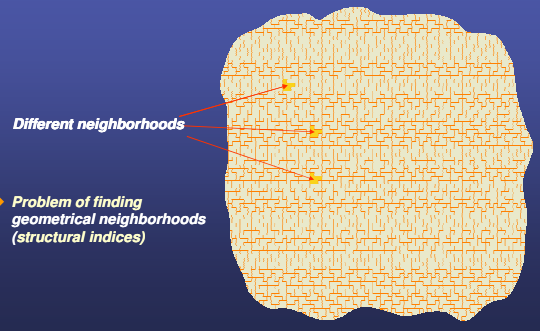
According to "Tilings and Patterns" [Grunbaum & Shephard '86]

- *Fills the entire plane non-periodically*
- *Self-similarity*



Infinite plane

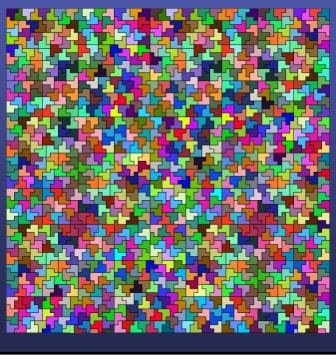
81-rep G-hexominoes



- *Problem of finding geometrical neighborhoods (structural indices)*

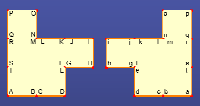
Infinite plane

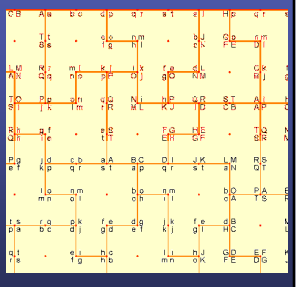
Apparent Chaos



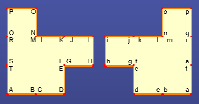
Different colors = Different neighborhoods = Different structural indices

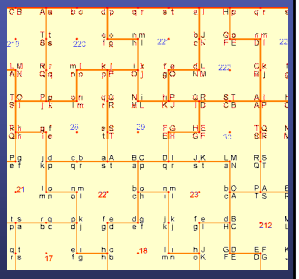
Finding structural indices



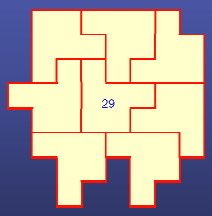


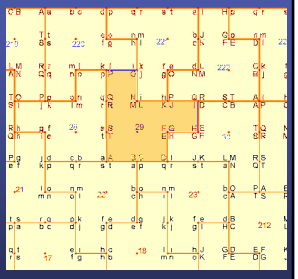
Finding structural indices



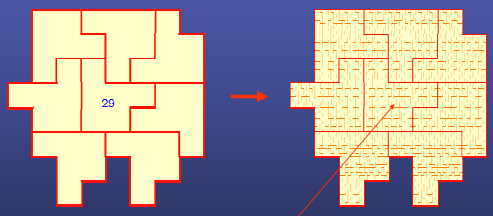


Finding structural indices



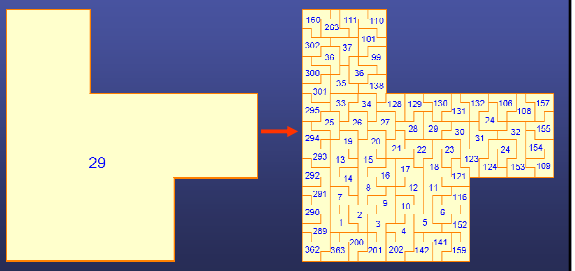


Finding structural indices

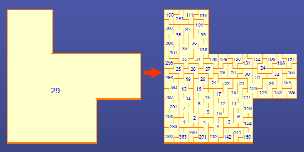


Well-determined neighborhoods

Finding structural indices



Optimization



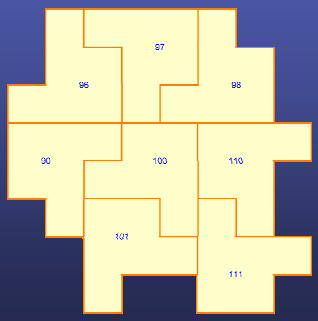
Decomposition process:

- ***Production rules***
- ***Structural indices characterize local geometrical neighborhoods***
- ***Mapping of structural indices (finite number)***

➔ ***Tiles with similar geometrical neighborhoods (structural indices) behave similarly***

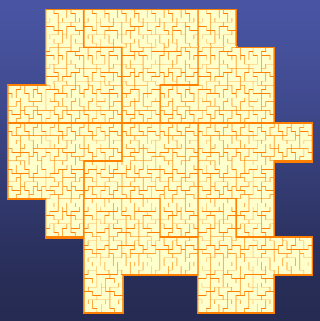
Optimization I

Patch



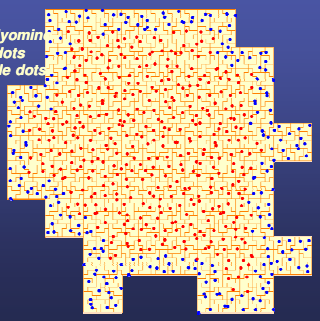
Optimization I

Subdivision



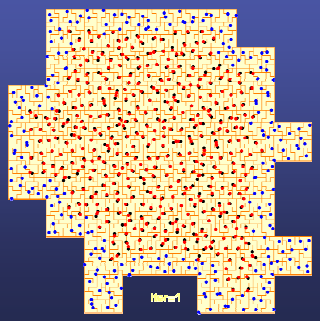
Optimization I

Random dots
within each polyomin
Red: movable dots
Blue: immovable dots



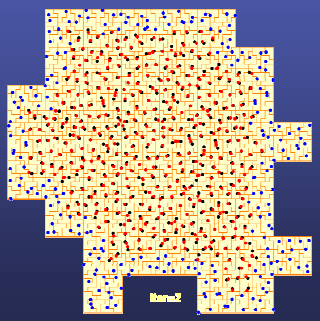
Optimization I

Relaxation



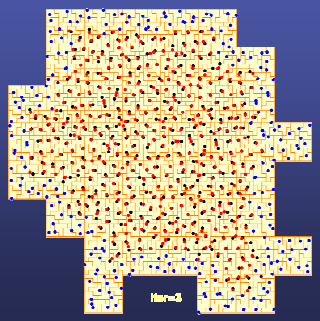
Optimization I

Relaxation



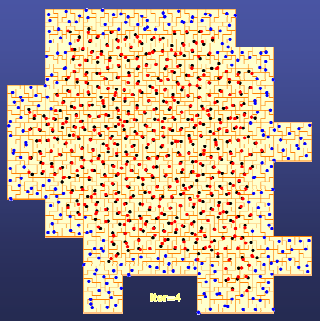
Optimization I

Relaxation



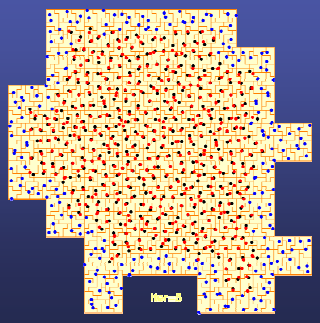
Optimization I

Relaxation



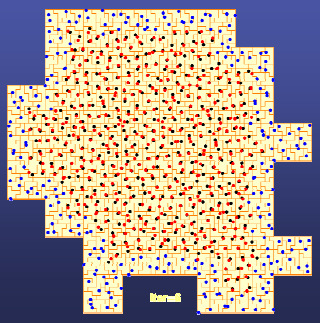
Optimization I

Relaxation



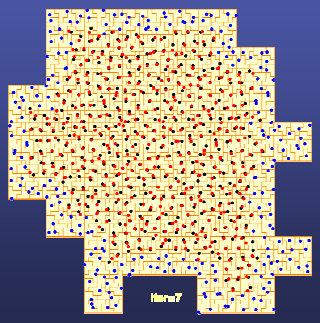
Optimization I

Relaxation



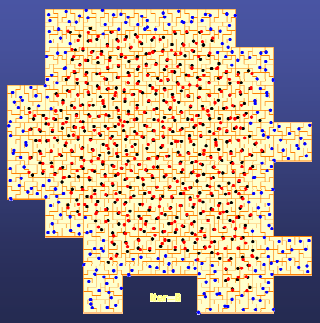
Optimization I

Relaxation



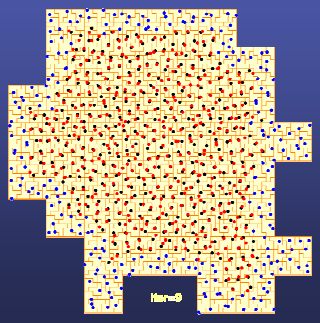
Optimization I

Relaxation



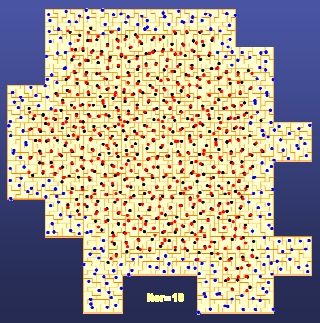
Optimization I

Relaxation



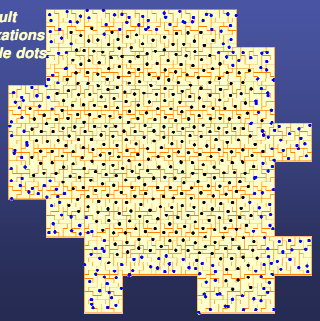
Optimization I

Relaxation

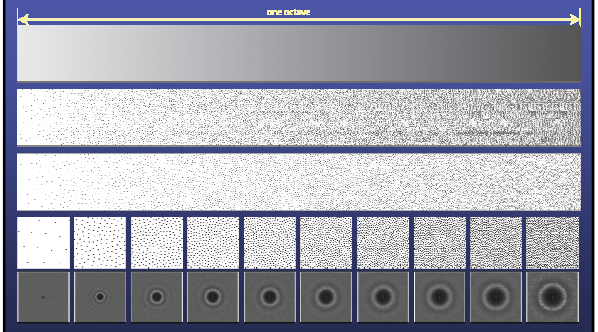


Optimization I

Black: final result after 10 relaxations
Blue: immovable dots



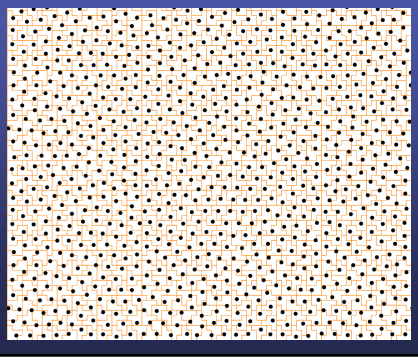
Optimization



Run-time pseudo-code

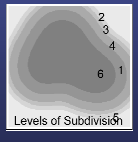
```
AdaptiveSamp(p  of type polyomino)
        ▷ Structure polyomino contains the fields:
        ▷   structuralIndex
        ▷   LOS: level of subdivision
        ▷   refPoint
        ▷   v1, v2: polyomino's basis vectors
        ▷   code: used for computing threshold
local_LOS ← GetMaxLOSwithinPolyomino(p)
If p.LOS ≥ local_LOS
    then    ▷ Terminal: no need for more subdivisions
        local_importance ← GetLocalImportance(p)
        threshold ← IntegralInBase(p.code)
        If local_importance ≥ threshold
            then    ▷ Output selected sample
                i_imp ← GetImportanceIndex(local_importance)
                {k1, k2} ← list[p.structuralIndex, i_imp]
                position ← p.refPoint + k1 * p.v1 + k2 * p.v2
                OutputSample(position)
        return
    else    ▷ Need more subdivisions
        {p1,...,p_st} ← SubdivideUsingProductionRules(p)
        return {AdaptiveSamp(p1),...,AdaptiveSamp(p_st)}
```

Results – constant density

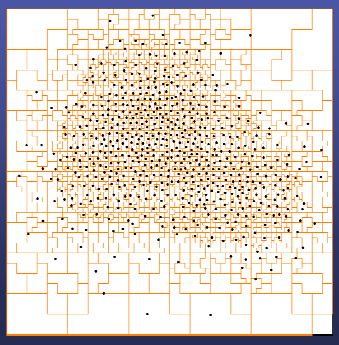


Results – variable density


Importance Image



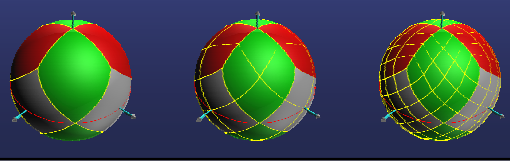
Levels of Subdivision



Sampling on a sphere

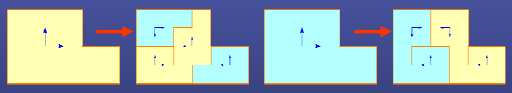
HEALPix spherical mapping [Gorski et al. 2005]

- 12 Equiareal high-level quads
- Jacobian-preserving mapping
  - Subdivision into equiareal quads
- Low distortion

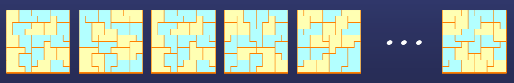


Sampling on a sphere

Pentominoes with two subdivision rules

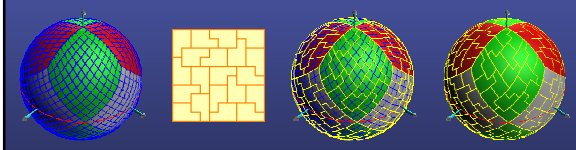


Square patches of 20 pentominoes on 10x10 squares



Sampling on a sphere

Pentominoes on 12 HEALPix quads



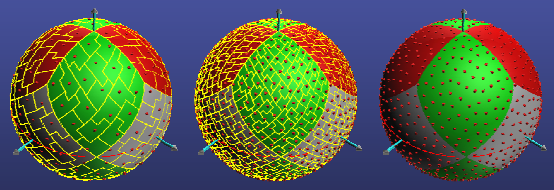
10x10 grid on each quad

20 pentominoes on a 10x10 square

20 pentominoes on each quad

Sampling on a sphere

Subdivision and optimization



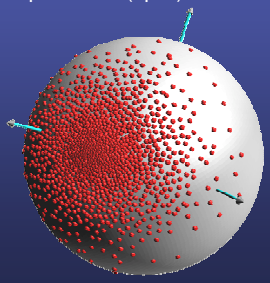
Level 1

Level 2 (after one subdivision)

Level 2 (dots only)

Sampling on a sphere

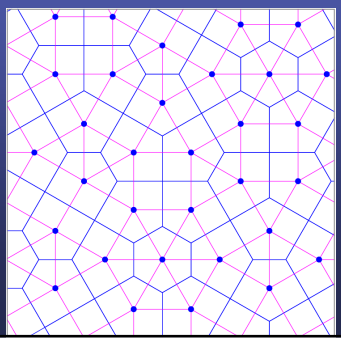
An arbitrary importance (spot) function



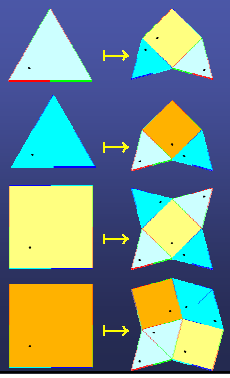
Presentation Outline

- Motivations
- Importance Sampling System Based on Penrose Tiling
- Sampling with Polyominoes
- 2D Low-Discrepancy Sequences Based on Dodecagonal Tiling
- Conclusions and Challenges

Dodecagonal Tiling

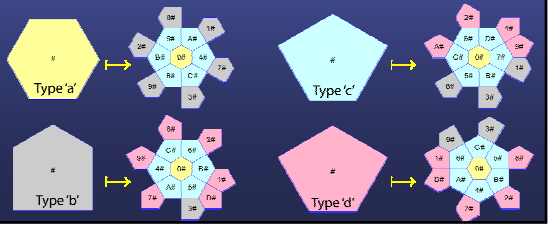


Dodecagonal Tiling, Triangles and Squares-Based Production Rules

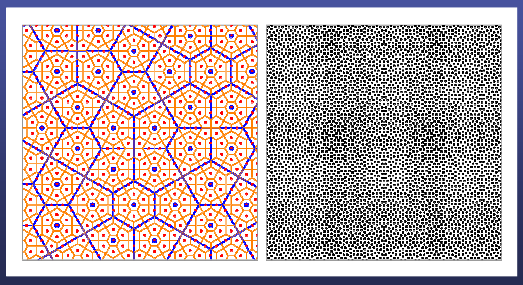


Dodecagonal Tiling, Voronoi Polygons-Based Production Rules

$$\mathcal{P}_{dodecagonal} := \begin{cases} a_{\#} \mapsto \{a_{0\#}, c_{4\#}, c_{5\#}, c_{6\#}, c_{A\#}, c_{B\#}, c_{C\#} \\ \quad b_{1\#}, b_{2\#}, b_{3\#}, b_{7\#}, b_{8\#}, b_{9\#}\}, \\ b_{\#} \mapsto \{a_{0\#}, c_{4\#}, c_{5\#}, c_{6\#}, c_{A\#}, c_{B\#}, c_{C\#} \\ \quad d_{1\#}, d_{2\#}, b_{3\#}, a_{7\#}, d_{8\#}, d_{9\#}, d_{D\#}\}, \\ c_{\#} \mapsto \{a_{0\#}, c_{5\#}, c_{6\#}, a_{7\#}, c_{B\#}, c_{C\#}, c_{D\#} \\ \quad b_{1\#}, d_{2\#}, b_{3\#}, d_{4\#}, b_{8\#}, d_{9\#}, d_{A\#}\}, \\ d_{\#} \mapsto \{a_{0\#}, c_{4\#}, c_{5\#}, c_{6\#}, c_{A\#}, c_{B\#}, c_{C\#} \\ \quad d_{1\#}, d_{2\#}, b_{3\#}, a_{7\#}, d_{8\#}, b_{9\#}, d_{D\#}\} \end{cases}$$



Dodecagonal Tiling, Applying Voronoi Polygons-Based Production Rules



Conclusions

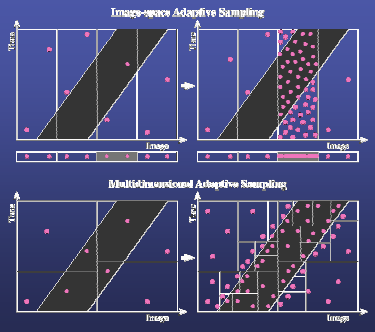
Tile-based Sampling allows:

- Very good low-noise low-artifact sampling distribution
- Very fast
- Simple and easy to implement

Challenges

- Fast tile-based multi-dimensional sampling
- Looking for optimum between the size of pre-calculated data and the quality

N-Dim Optimal Sampling

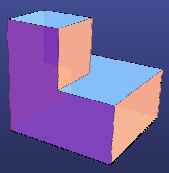


Source: Multidimensional Adaptive Sampling and Reconstruction for Ray Tracing
T. Hachisuka, W. Jarosz, R. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. Wann Jensen (SIGGRAPH 2008)

Challenges

- Fast tile-based multi-dimensional sampling

3D: Polycubes



N-D: ???

Challenges

- Fast tile-based multi-dimensional sampling
- Looking for optimum between the size of pre-calculated data and the quality

Part V

Tile-Based Methods for Non-Photorealistic Rendering and Landscape Modeling

Chapter 7

Slides

Tile-Based Methods
in
Non-Photorealistic Rendering
and
Landscape Modeling

JOHANNES KOPF
UNIVERSITY OF KONSTANZ

Outline

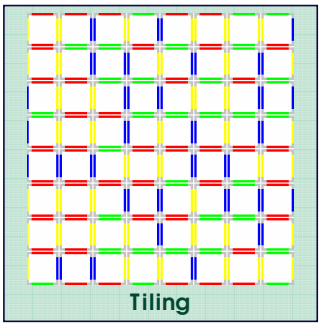
- Part One: Wang Tiles
  - Texture synthesis on tiles
  - Simple point distributions
- Part Two: Recursive Tiles
  - Definition, algorithm, etc.
  - Applications: Non-photorealistic rendering, Texture painting

Wang Tiles

Michael Cohen, Jonathan Shade, Stefan Hiller, Oliver Deussen
Wang Tiles for Image and Texture Generation
Proceedings of SIGGRAPH 2003

Tile Set

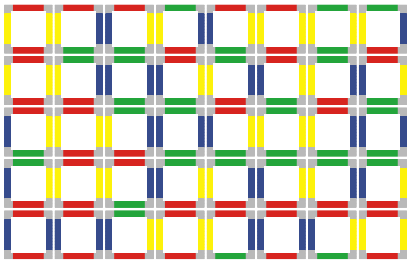
1.) Square tiles
2.) Colored edges
3.) Match colors
4.) Don't rotate



Tiling

Stochastic Tiling Algorithm

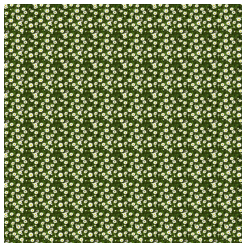
Tile Set



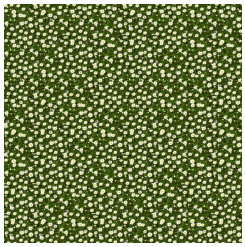
...

...

Why Wang Tiles?

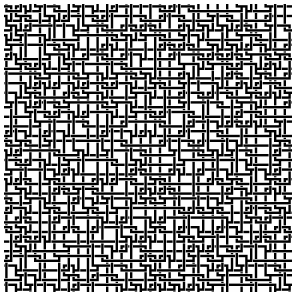


Periodic Tiling



Wang Tiles

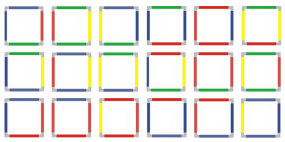
Why Wang Tiles?



Wang Tiles

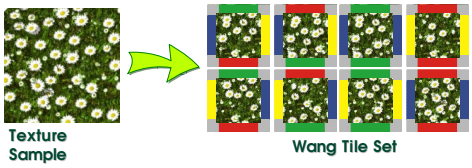
- 2 horizontal and vertical colors = 8 tiles

- 3 horizontal and vertical colors = 18 tiles

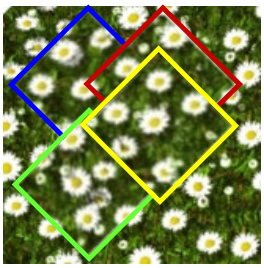


- C horizontal and vertical colors = 2 x C^2 Tiles

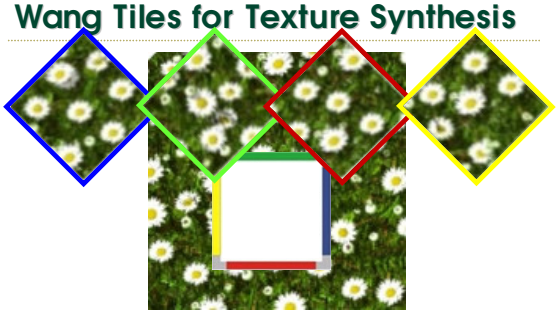
Wang Tiles for Texture Synthesis



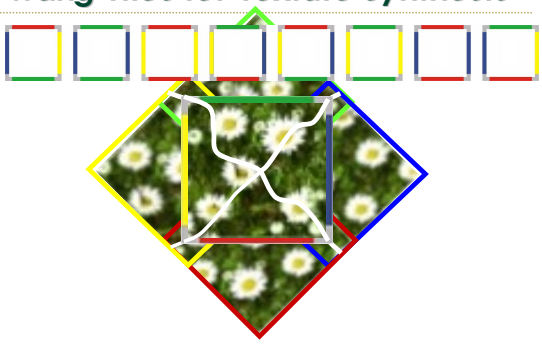
Wang Tiles for Texture Synthesis



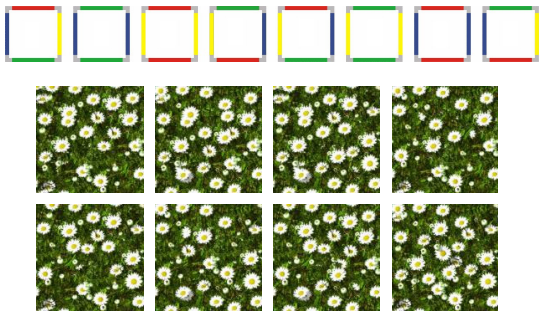
Wang Tiles for Texture Synthesis

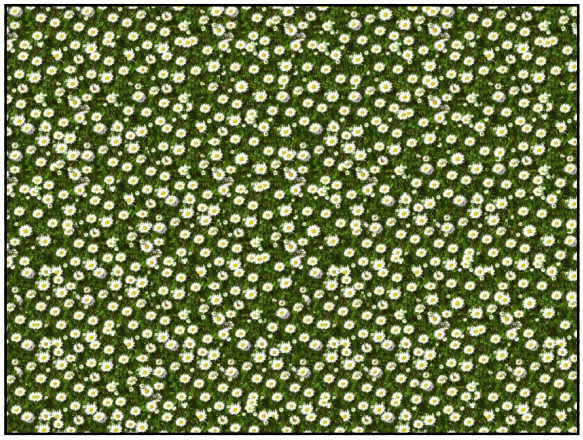


Wang Tiles for Texture Synthesis

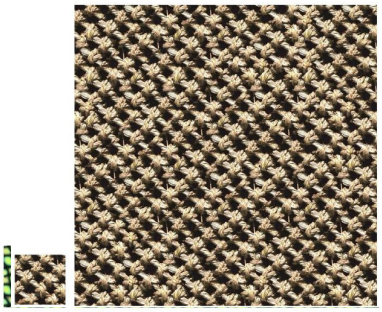


Wang Tiles for Texture Synthesis



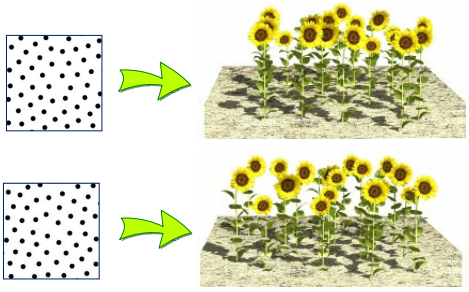


Wang Tiles for Texture Synthesis



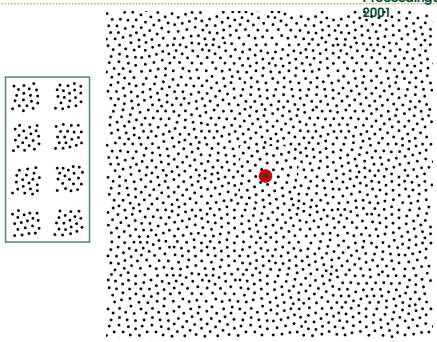
Wang Tiles for Distributions

Wang Tiles for Distributions



Wang Tiles for Distributions

Hiller, Deussen, Keller
Tiled Blue Noise Samples
Proceedings of VMV 2001

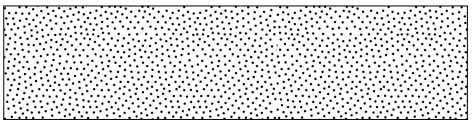


Wang Tiles for Distributions

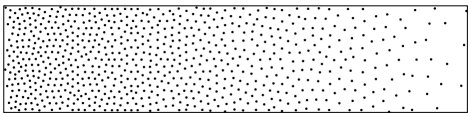
Single Tile (160 plants)

8 Wang Tiles (20 plants each)

Progressive & Recursive Tiles

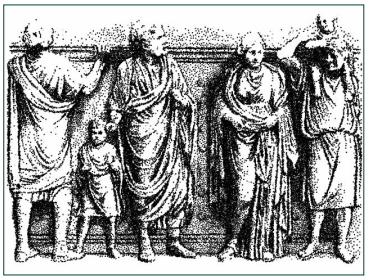


So far: only **constant** density



Coming up: **non-uniform** density

More Motivation



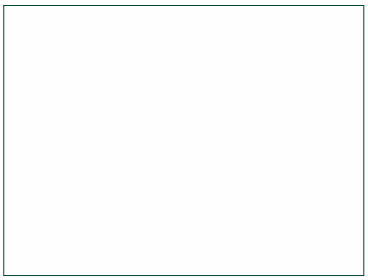
Overview

- Tile based method
  - Wang tiles ⇒ infinite non-periodic tilings
  - Progressive sets ⇒ non-uniformity
  - Recursive tiles ⇒ adaptive subdivision

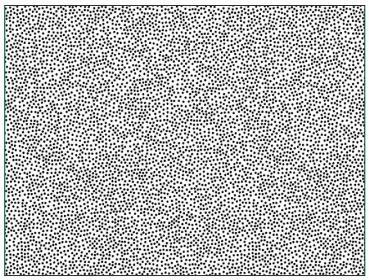
Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, Dani Lischinski
Recursive Wang Tiles for Real-Time Blue Noise
Proceedings of SIGGRAPH 2006

Progressive Point Sets

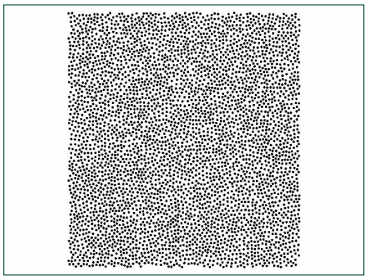
Michael McCool, Eugene Fiume
Hierarchical Poisson Disk Sampling Distributions
Proceedings of Graphics Interface '92



Progressive Point Sets

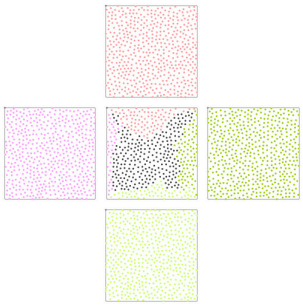


Progressive Tile

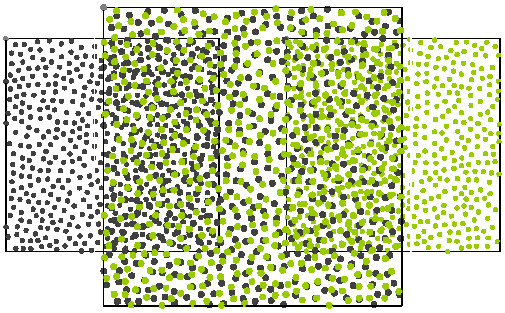


Progressive Wang Tiles

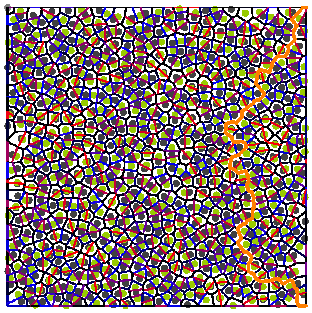
Progressive Wang Tiles



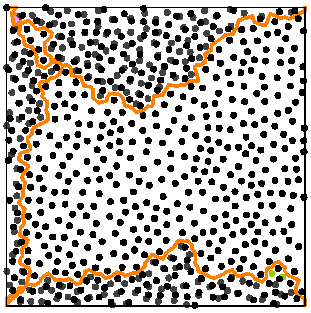
Progressive Wang Tiles



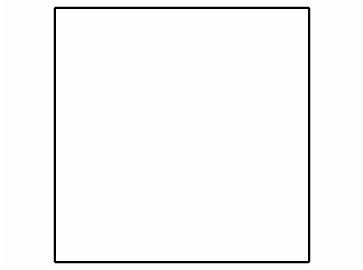
Progressive Wang Tiles



Progressive Wang Tiles



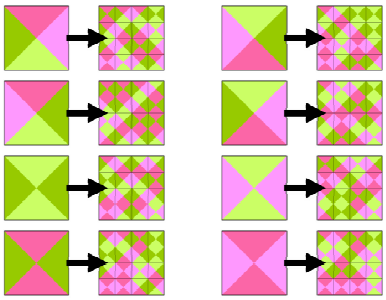
Progressive Wang Tiles



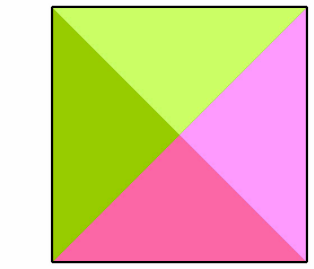
Progressive Wang Tiles Summary

- So far:
  - Progressive Wang Tile set
  - Non-periodic
  - Non-uniform
- Coming up next:
  - Overcoming bounded maximum density

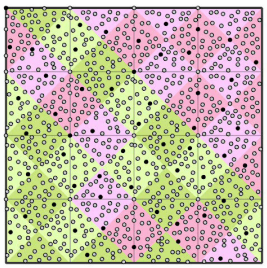
Recursive Wang Tiles



Recursive Wang Tiles



Progression & Recursion



Results

- Tile construction: approx. 10 mins
- Runtime:
  - Extremely fast (6,000,000+ points/s)
    - GPU implementation: MUCH faster
  - Non-uniform density
  - Local random access to infinite patterns
  - Constant memory footprint
  - Cost ~ Integral over density in local window

Application: NPR

Real-Time Stippling Demo

Application: Texture Painting

Conclusion

- Wang Tiles for Poisson disk point se
- Non-uniform density using *progressive* and *recursive* tiles
- Applications
  - Non-photorealistic rendering
  - Object instancing
- Download tile set & source co

Thanks

Questions?

Bibliography

Ball, W. W. R. *Mathematical recreations and essays*. MacMillan and Co., 1926.

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

Chenney, S. Flow tiles. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 233–242. 2004.

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.

Culik, II, K. An aperiodic set of 13 Wang tiles. *Discrete Mathematics*, 160(1-3):245–251, 1996.

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

Decaudin, P. and Neyret, F. Packing square tiles into one texture. In *Eurographics '04 (short papers)*, pages 49–52. 2004a.

Decaudin, P. and Neyret, F. Rendering forest scenes in real-time. In *Rendering Techniques '04 (Eurographics Symposium on Rendering)*, pages 93–102. 2004b.

Dungan, W., Jr., Stenger, A., and Sutty, G. Texture tile considerations for raster graphics. In *SIGGRAPH '78: Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 130–134. 1978.

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

Glassner, A. Aperiodic tiling. *IEEE Computer Graphics & Applications*, 18(3):83–90, 1998.

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

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.

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

Kaplan, C. S. Voronoi diagrams and ornamental design. In *ISAMA'99: The first annual symposium of the International Society for the Arts, Mathematics, and Architecture*, pages 277–283. 1999.

Kaplan, C. S. Computer generated Islamic star patterns. In *Bridges 2000: Mathematical Connections in Art, Music and Science*, pages 105–112. 2000.

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

Kaplan, C. S. Islamic star patterns from polygons in contact. In *GI '05: Proceedings of the 2005 conference on Graphics interface*, pages 177–185. Canadian Human-Computer Communications Society, 2005.

Kaplan, C. S. A meditation on Kepler's Aa. In *Bridges 2006: Mathematical Connections in Art, Music and Science*, pages 465–472. 2006.

Kaplan, C. S. The trouble with five. *Plus Magazine*, 2007. 15 pages, to appear. Invited article on five-fold tilings.

Kaplan, C. S. and Hart, G. W. Symmetrohedra: polyhedra from symmetric placement of regular polygons. In *Bridges 2001: Mathematical Connections in Art, Music and Science*, pages 21–28. 2001.

Kaplan, C. S. and Salesin, D. H. Escherization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 499–510. ACM Press/Addison-Wesley Publishing Co., 2000.

Kaplan, C. S. and Salesin, D. H. Dihedral Escherization. In *GI '04: Proceedings of the 2004 conference on Graphics interface*, pages 255–262. Canadian Human-Computer Communications Society, 2004a.

Kaplan, C. S. and Salesin, D. H. Islamic star patterns in absolute geometry. *ACM Trans. Graph.*, 23(2):97–119, 2004b.

Kari, J. A small aperiodic set of Wang tiles. *Discrete Mathematics*, 160(1-3):259–264, 1996.

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.

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.

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.

Lagae, A. and Dutré, P. Generating well-distributed point sets with a self-similar hierarchical tile. Report CW 462, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2006b.

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, 2006c.

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

Lagae, A. and Dutré, P. A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum*, 2007a. To appear.

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

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.

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

Lemmen, H. V. *Tiles: 1000 Years of Architectural Decoration*. Harry N. Abrams, Inc., 1993.

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.

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. *IEEE Transactions on Visualization and Computer Graphics (TVCG)*, ???? Accepted for publication.

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.

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

Lukkarila, V. The square tiling problem is NP-complete for deterministic tile sets. Technical Report TUCS Technical Report No 754, Turku Centre for Computer Science, 2006.

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

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

Ng, T.-Y., Wen, C., Tan, T.-S., Zhang, X., and Kim, Y. J. Generating an ω-tile set for texture synthesis. In *Proceedings of Computer Graphics International 2005*, pages 177–184. 2005.

Ostromoukhov, V. Sampling with polyominoes. *ACM Transactions on Graphics*, 26(3), 2007.

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.

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.

Robinson, R. M. Seven polygons which admit only nonperiodic tilings of the plane (abstract). *Notices of the American Mathematical Society*, 14:835, 1967.

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

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.

Sibley, P., Montgomery, P., and Marai, G. E. Wang cubes for video synthesis and geometry placement. In *ACM SIGGRAPH 2004 Poster Compendium*. 2004.

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

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.

Wang, H. Notes on a class of tiling problems. *Fundamenta Mathematicae*, 82:295–305, 1975.

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.

Xu, J. and Kaplan, C. S. Vortex maze construction. *Journal of Mathematics and the Arts*, 1(1):7–20, 2007. A shorter version appeared in the proceedings of Bridges 2006.