EFFICIENT TEXTURE SYNTHESIS WITH A SMALL SET OF TILES

WEN CONG HUA

NATIONAL UNIVERSITY OF SINGPORE

2004

EFFICIENT TEXTURE SYNTHESIS WITH A SMALL SET OF TILES

WEN CONG HUA
(B.S. in Math., Shandong University, China)

A THSIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2004

Acknowledgement

First and foremost, I would like to express my gratitude especially to my supervisors:

- A/P Tan Tiow Seng, for valuable guidance and for pushing the project ahead. I have learned from him both the research philosophy and the attitude to life.
- Dr. Xu Shu Hong, for his continuous support and encouragement. Without his support and kindness, I could not fulfill this project.

My immense gratitude goes to my parents and sisters. It is your loves that support me to tackle the difficulties in life and study. I also want to thank all my colleagues in graphics lab of the School of Computing, and student lab of IHPC.

Special thanks go to all my friends, especially Jin Xingang, Zhang Xinyu, Chen Xin and Ji Liping who kindly help me when I am in difficulty. I am grateful for your warmhearted assistance and continuous comfort.

Last but not least, I would like to thank Institute of High Performance Computing and National University of Singapore for providing me scholarship for this project.

Abstract

Time efficiency and visual effect demand effective synthesis techniques to real time synthesize textures with the same statistical distribution of the texture primitives as the example. This paper presents an approach to generate, from an input texture, a set of small textures that can be tiled to synthesize large textures. Such a small set can be useful in, for example, real-time graphics applications to texture any large area realistically and efficiently while utilizing small amount of texture memory. We apply cutting curve to the large space to eliminate the line artifacts. The introduction of new image patch in each tile makes the texture information diverse. The new tile combination rule based on the tile indices combines the small tiles without artifacts and creates a stochastic distribution of the tiles. Our approach is interesting in its ability to generate a small number of tiles that embed many more texture patterns as compared to previous approaches. This feature of a small tile set can avoid generating highly repetitive patterns in synthesized textures. This and its attempt to preserve consistent local features of the synthesized textures are examined in our experiments with input textures of stochastic, semi-structured and structured nature. Our approach performs well in general and is particularly superior to previous work for input textures of stochastic and semi-structured nature.

Keywords: texture synthesis, image tiling, cutting-path, cutting-curve, Wang tile

Contents

List of Figures

Chapter 1 Introduction

1.1 Background

Texture synthesis in graphics can be formulated as follows: given an input texture, synthesize an output one, which is usually different from the input while keeping the underlying global distribution pattern [Liang01]. For computer graphics, a general use of texture synthesis is to pre-compute a large texture for mapping to a large area in an application while only a small sample texture is available. Most of these utilities are applied during the process of visualizing or rendering large-scale scenes with homogeneous texture patterns, such as terrains. In visualizing terrains, user often resorts to synthesizing image plane from samples rather than constructing the entire geometric model. Another use of texture synthesis is to generate a set of small textures that can be combined or tiled into large textures during some real time applications. Keeping many immense images is a burden for visualization applications. Therefore, user can apply tiling algorithms to texture a large plane from a tile set in real time speed. This use can avoid the need of physically having a large texture memory when texturing a large area. This second application is the focus of this thesis, though our results can be used in the first application.

In the past decade, a wave of algorithms has been explored in texture synthesis. Many approaches of pixel-based [Heeger95, de.Bonet97, Efros99, Wei00, Ashikhmin01] can create a great variety of vivid texture patterns through matching parameters of

either statistical models or statistical features. However, they are not feasible to real time applications as they incrementally synthesize pixels in the output image, even though a lot of techniques have been utilized to speed up pixel sampling. Recently, researchers propose to synthesize texture with image patches instead of pixels [Xu00, Efros01, Liang01, Kwatra03, Neubeck03]. Such patch-based methods can approximately achieve orders of magnitude faster than pixel-based methods. However, they are not applicable to our focus application, as they do not generate a set of textures for efficient tiling. On the other hand, the work of Cohen et al. [Cohen03] demonstrates such a way. Each square tile, called Wang tile (adapted from [Wang65]), has (conceptual) a color on each of its four sides representing the texture pattern contributing to that side. In efficiently synthesizing a large texture, one lays Wang tiles such that the adjacent sides of two Wang tiles have the same color.

1.2 Problem Statement

In general, there are three levels of challenges to generate a set of small tiles from an input texture $\mathcal{S}$. First, the content of each tile should be seamless as if it was a continuous part from $\mathcal{S}$. This is generally impossible due to the next challenge of fitting tiles across their boundaries. Whether the artifacts in each tile can be detected out of their neighboring textures are determined by our human discrimination ability. However, the local difference of the textures directly contributes to our discrimination. Therefore, various techniques have been investigated in the literatures to decrease the local difference around the discontinuous area in image. In generating a good Wang tile, a cutting path approach [Efros01] is applied to combine a set of four patches that

is randomly selected from $\mathcal{S}$. Second, tiles should fit together seamlessly across their boundaries in their synthesized textures. As Cohen et al. mentioned, each side of a Wang tile is coded with a color and only sides of the same color can be adjacent in a tiling. Third, the synthesized texture should maintain the underlying global distribution pattern of $\mathcal{S}$. High repetition or high regularity of local features will make the generated texture plane too artificial to be natural. Therefore, the way of laying the tiles in the output image will determine the overall visual effect of the created texture pattern. In Wang tile approach, this is addressed in a set of Wang tiles with at least two choices of tiles to be used at each tiling step.

1.3 Main Contributions

Our work proposes an alternative approach to address the above three challenges. Similar to Cohen et al.'s approach, it generates its tiles, called ω-tiles from a set $\mathcal{F}$ of four patches that are extracted from the input sample $\mathcal{S}$. However, it is different in that each side of an ω-tile has effectively two colors and its interior is obtained from $\mathcal{S}$, rather than restricted to those of $\mathcal{F}$ only. For a tile set of 16 small ω-tiles, we use four textures to contribute to four corners and 16 other textures to the interiors of the generated 16 tiles.

Specifically, our approach first addresses the possible seams within a tile by providing a large search space so as to control seams that result from combining patches in $\mathcal{F}$.

A large search space means more pre-processing time needed, but it also generates ω-tiles with less prominent seams. A cutting curve technique is applied in the large space of each tile to eliminate the line seams. Current cutting path methods [Efros01, Cohen03, Kwatra03] always focus on eliminating the line seams in small rectangular overlapping regions. Although such a method can find a heuristic solution to adapt the pixel values among the rectangular overlapping region of the adjacent patches, the results are often not satisfactory. As the area in which the cut is supposed to lie tends to be rather small, the leeway in getting seamless transitions is restricted and boundaries may remain visible [Neubeck03]. Therefore we seek a transition curve in a lager domain and get a better visual effect.

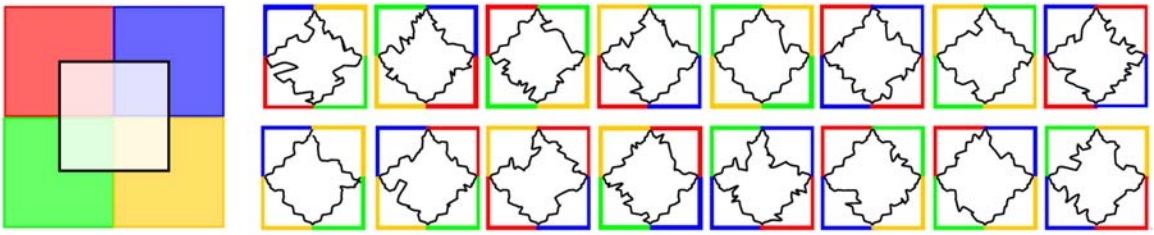
Second, seams between two ω-tiles lying adjacent to each other are avoided by having matching two-color sides between them. A subtle consequence of this, in any synthesized texture, is the result of a junction at the middle of each side of the ω-tile where four but not necessarily distinct patches of $\mathcal{F}$ meet. Such a junction in the method of Wang tiles appears in the middle of each tile where two cutting paths meet and at the four corners of each tile where cutting paths start or end. Junctions/corners must be treated with care, as they are particularly conspicuous to human eyes as found from experiments in neuroscience; see [Fahle94, Shevelev03]. In comparison, our approach allows more degree of control of the content at junctions as two of the four textures at each junction are obtained through careful searching.

Third, our approach achieves good underlying global distribution pattern of $\mathcal{S}$ in the synthesized texture. In the process of constructing each tile, we search for a new patch

from the input texture $\mathcal{S}$ to replace some parts of the distribution of $\mathcal{F}$. This makes the tiles embed more texture patterns (20 in the case of Figure 1.1). In contrast, a set of Wang tiles uses mainly four textures when synthesizing a large texture and thus tends to generate repetitive patterns in a large texture.

1.4 Tiling Scheme Overview

The key to design an efficient tiling algorithm is laying the tiles in such a way that the tiles are stochastically located in the image plane and the combination between the potential neighboring tiles should satisfy combination constraints. Otherwise, the synthesized image will either produce high repetition pattern or create visible intensity gap between the small tiles. In our tiling scheme, we first transfer the distortions from the sides of each tile to its internal area (see Figure 1.1-(a)). Then, we search a cutting curve in the large overlapping space of the four patches $\mathcal{F}$ and the new image patch selected from the sample to find a new pattern to cover the internal line seams in each tile (see Figure 1.1-(b)). To minimize the difference of pixel values between the four patches $\mathcal{F}$ and the new image patch, our algorithm can eliminate the line seams and preserve low difference of the intensities around the cutting curve in each tile. Like the tile set shown in Figure 1.1-(b), two tiles can be potentially neighboring only if their edge colors match. Such a constraint can make the tiles fit together without line seams across their boundaries and create the potential for tiling the small ω-tiles stochastically as well. Therefore, our tiling scheme not only produces smooth combination between tiles, but also preserves the image pattern of the input sample $\mathcal{S}$.



(a) (b)

Figure 1.1 Schematic showing the tiling approach. Four small patches (represented in red, green, blue and yellow) are chosen from an input texture to form 16 blocks; an example of a block is shown on the left. An *ω*-tile (out of 16) of the same size as a patch is cut from the center of each block. The color at each corner of an *ω*-tile indicates the patch of that color that contributes to the corner. Its interior (shown within weaving closed curve) is obtained through searching the input texture.

1.5 Outline of this Thesis

This thesis is organized in six chapters discussing various topics related to our tiling scheme. This chapter introduces the background and main problems of image tiling, our contributions, and tiling overview. Chapter 2 reviews the related previous works in texture synthesis. We make a classification of current texture synthesis algorithms. Then, we review some previous work including the technique of cutting path [Efros01, Kwatra03] and the approach of Wang tile [Cohen03]. Chapter 3 and Chapter 4 illustrate the tiling process. Chapter 3 addresses the creation of each ω-tile that is seamless. Chapter 4 presents an approach to generate various small sets of ω-tiles that can fit together across their boundaries and can synthesize large textures with less repetitive patterns. Chapter 5 describes the implementation details and results. The visual effect of the synthesized images verifies the efficiency of our tiling technique. And Chapter 6 gives the conclusion and discusses the future work.

Chapter 2 Previous Works

Statistical texture synthesis has been investigated for more than a decade. Two main categories for texture synthesis are procedural methods and statistical sampling methods. These two approaches are complementary in their strength and weakness. In this chapter, we first in Section 2.1 give a classification of texture synthesis algorithms introduced by current literatures. Then, in Section 2.2 we discuss the statistical sampling method, where we focus on the main characteristics of synthesis process in non-parametric sampling approach. At the same time, we introduce one key technique to eliminate the line seams—cutting path technique [Efros01, Kwatra03]. Finally, in Section 2.3 we introduce the procedural methods and focus on discussing the approach of Wang tile [Cohen03].

2.1 Synthesis Algorithm Classification

In this section, we introduce an overview of the texture synthesis algorithms and bring forward a classification, which generalizes other authors' classifications [Tuceryan98, Xu00, Liang01, Levina02, Kwatra03].

Besides the use of an explicit texture to texture a large area, the texturing algorithms can be classified as procedural method [Peachey85, Perlin85, Lewis89, Upstill89, Turk91, Witkin91, Worley96] and statistical sampling method [Heeger95, de.Bonet97, Portilla99, Zhu00, Efros99]. Procedural method has been proved itself valuable in

texture synthesis, allowing for complex surfaces to be rendered without requiring image mapping or modeling geometrical details. They can be fast and support memory efficient texture rendering by not storing the synthesized texture explicitly but synthesizing them on the fly [Upstill89]. However, most procedural methods are only specialized emulators of the generative processes of certain types of textures, such as marbles, seashells, and animal skins. Procedural methods are usually classified into procedural texturing approach and pattern based texturing approach, according to whether the algorithms apply patches of samples to define texture patterns.

Statistical sampling methods can synthesize a wide variety of textures, as long as appropriate sample textures are provided. But they need great computational cost. The major problems for statistical sampling methods are: (1) modeling- how to estimate the stochastic process from a given finite texture sample and (2) sampling- how to develop an efficient sampling procedure to produce new textures from a given model. Both the modeling and sampling parts are essential for the success of texture synthesis: the visual fidelity of generated textures will depend primarily on the accuracy of the modeling, while the efficiency of the sampling procedure will directly determine the computational cost of texture generation [Wei00]. From the modeling perspective, the synthesis algorithms can be classified from the approaches to model texture in computer graphics: statistical model [Cross83, Chellappa85, Paget99, Parada01] and feature matching [Heeger95, Portilla99, de. Bonet97, Zhu97, Zhu00]. From the sampling perspective, the synthesis algorithms can be classified in view of the characteristics in sampling information from the given image: parametric sampling

[Heeger95, Portilla99, Zhu00, Soatto01, Wang02] and Non-parametric sampling [de. Bonet97, Efros99, Efros01, Liang01, Neubeck03].

In general, the classification of current texture synthesis algorithms can be illustrated in Figure 2.1. Both categories are divided into subclasses according to their specific characteristics in synthesis.

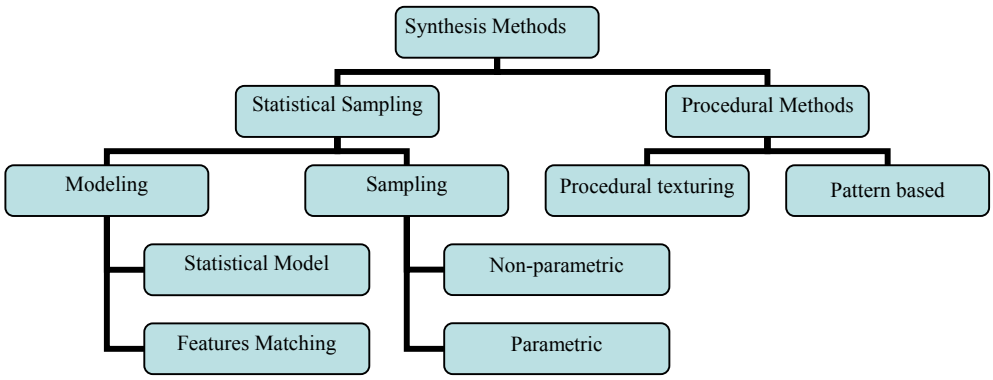


Figure 2.1 Texture synthesis algorithm classification: statistical sampling methods are classified by the characteristics in modeling and sampling, and procedural methods are classified by the principle to define texture patterns.

2.2 Statistical Sampling Method

In statistical sampling method, parametrical sampling approach uses a number of parameters within a compact parametric model to describe a variety of textures and then sample pixels within the parametric restriction. The computation for such approach is especially costly. Therefore, the new trend of non-parametric sampling approach becomes popular in graphics field, which is also closely related to our tiling technique. From the inspiration of these literatures, we design our scheme to cut the

patch set $\mathcal{F}$ from the input sample $\mathcal{S}$ to construct new pattern and to eliminate the seams across the image patches to create small tiles.

2.2.1 Non-parametric Method

Non-parametric sampling method tends to estimate the local conditional probability density function and synthesizes pixel incrementally. Efros and Leung [Efro99] introduce the seminal paper that uses the non-parametric sampling method. Given all its neighbors synthesized, the conditional distribution of each pixel is estimated by searching the sample image, finding all similar neighborhoods, and then randomly choosing one among them. The algorithm produces good results for a wide range of textures, but is terribly slow, as a full search of the input image is required to synthesize each pixel.

Many variations of their method have been published to speed up and optimize the original algorithm in different ways [Wei00, Wei01, Ashikhimin01]. These algorithms sample information from the random fields directly, without constructing an explicit model for the distribution. Wei and Levoy [Wei00] optimize Efros and Leung's algorithm [Efros99] using neighborhood with a fixed shape so that it can conduct the searching deterministically. Their method (denoted as WL algorithm) can be accelerated by tree structured vector quantization (TSVQ). They also succeed in extending their algorithm to arbitrary surface [Wei01]. Ashikhmin [Ashikhmin01] presents a modification of the WL algorithm to perform better on a particular class of textures, which consists of identification primitives of irregular but similar patterns, such as grass, leafs, pebbles, bushes and so on. This method has made a small step

towards patch-based synthesis by greedily extending existing patches whenever possible, rather than by searching the entire sample texture.

All the pixel-based sampling methods do not perform fast. They usually cost a lot of memories and computations on determining each pixel value. Thus these algorithms including accelerated versions are not appropriate to real time applications. Actually during synthesis process, most pixels have their values totally determined by those that have been synthesized so far. Thus algorithms do not need to do searching on each pixel. This is an important reason that researchers develop patch-based sampling approaches.

Recently, researchers extend the idea of pixel-based synthesis to synthesize image patches incrementally [Efros01, Liang01, Kwatra03]. They search for the image patches in the sample by computing the distance of boundary area between the candidate patch from the sample and those in the output. This method in theory is to estimate the local conditional Markov Random Field (MRF) density in the overlapping region of these patches. Soler [Soler02] extends this approach for synthesizing texture on surfaces. This searching process can control the intensity gaps between the image patches. However, in most cases the line seams are still visible. Therefore, the critical problem left is how to eliminate the line seams between the image patches. For Liang et al.'s algorithm [Liang01], they just apply feathering technique to blend the pixels in the adjacent areas. In [Efros01] and [Kwatra03], they seek the cutting path in the overlapping region of the neighboring patches and adapt the pixel values according to the tracks of the cutting path. This technique is similar as ours to eliminate the seams between the set $\mathcal{F}$ of four patches. However, we seek a

close path, called cutting curve, in a quite large space of each tile. In the following, we give a brief introduction to the cutting path technique.

2.2.2 Cutting Path Technique

Efros and Freeman [Efros01] present a cutting path technique in image quilting method. The algorithm uses dynamical programming to choose the minimum cost path from one end of the overlapping region to the other. The following schematic figure shows the cutting path in the overlapping region across the two image patches (see Figure 2.2).

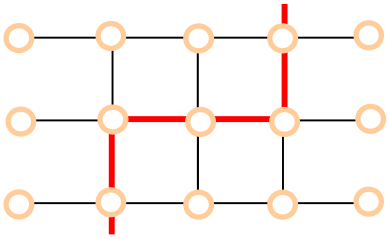


Figure 2.2 Schematic drawing to show graph formulation: the red line showing the cutting path in the overlap between two patches.

The weight of the node in the graph is the error of the pixel value in two image patches.

$$M(s, A, B) = \| A(s) - B(s) \| \quad (2\text{-}1)$$

where $A(s)$ and $B(s)$ denotes the value of pixel s in the patch A and B, respectively. The vertical directional cutting path is a path with the minimum sum of the error computed by dynamical programming from the top to the bottom lines [Sedgewick01]. The horizontal directional cutting path can be sought with the same algorithm.

Such a criterion does not incorporate well the aim of finding a smooth transition path in the overlap, although it preserves the nature of minimum distance along the cutting path. Kwatra et al [Kwatra03] use another criterion to seek the cutting path. They regard the link between any two adjacent pixels as graph edge and assign each edge a weight as the flow capacity. The cutting path is a path with the minimum error to transit the pixel intensities from patch *A* (SOURCE) to patch *B* (SINK) in the overlapping region (see Figure 2.3).

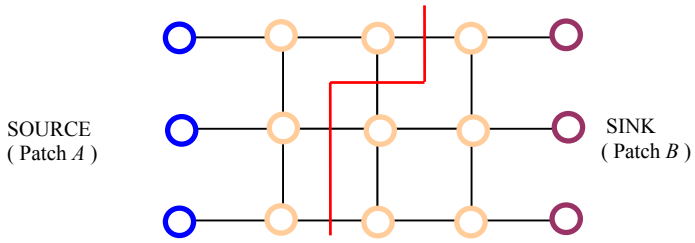


Figure 2.3 Schematic drawing to show graph formulation: the red line showing the graph cut in the overlap between two patches.

The weight is the sum of the transition errors of the two nodes divided by the sum of their gradients in two patches, which is also the penalty value for pixels changing color from Patch *A* to Patch *B*, and vice-versa (see Equation 2-2).

$$M(s,t,A,B)=\frac{\|A(s)-B(s)\|+\|A(t)-B(t)\|}{\|G_A^d(s)\|+\|G_A^d(t)\|+\|G_B^d(s)\|+\|G_B^d(t)\|} \tag{2-2}$$

where *s*, *t* are neighboring pixels in the overlapping patches *A*, *B*; $G_A^d(s)$ indicates the gradient of pixel *s* in the patch *A* along *d* (gradient direction between *s* and *t*), the same meaning as $G_A^d(t)$, $G_B^d(s)$ and $G_B^d(t)$. Then the cutting path is the min-cut [Sedgewick01], which cuts the graph edges with the minimum sum of flow capacities. In implementation, the efficiency of cutting path to minimize the difference will be

affected by the size of the overlapping region. If it is too small, the line with abrupt intensity change will be left at various places along the cut.

Overcoming the drawback of pixel-based sampling methods in the memory and computational cost, the patch-based sampling algorithms often get several orders of magnitude faster speed. They are feasible to interactive systems. Although they are not in our focus, as they do not generate a set of small tiles to be combined or tiled in the output image plane, their technique of finding the cutting path gives us valuable inspirations to find the cutting curve in the small ω-tile.

2.3 Procedural Method

Procedural method textures arbitrary surface or large plane by a procedural means. In general, there are three different approaches in this category. First, procedural texturing ([Perlin85, Worley96, Ebert94]) can generate details at arbitrary resolutions with no periodicity and very low memory. However, there are certain material aspects of textures that cannot be generated using these techniques of low computational cost.

Second, pattern based texturing uses a set of different small texture patches to define a pattern. These small patches are used to tile a large area while avoiding the periodicity and repetitiveness of a naïve tiling. Examples of this method are aperiodic tiling [Stam97], triangular patterns [Neyret99], virtual atlases [Soler02], sparse convolution [Lewis89, Ebert94], chaos mosaic [Xu00], and Wang tile [Cohen03]. These

approaches generally need more texture memory than those with procedural texturing, but lower computational cost during rendering.

Third, the recent pattern based procedural texturing [Lefebvre03], which combines the previous two approaches, determines the texture value at any surface location by aperiodically combining provided patterns according to user-defined controls such as a probability distribution and animation of textures. This method provides a sophisticated control to texture a large area with the texture indirection ability of recent graphics boards. It requires, in general, more computation during rendering as compared to pattern based texturing as studied in our work.

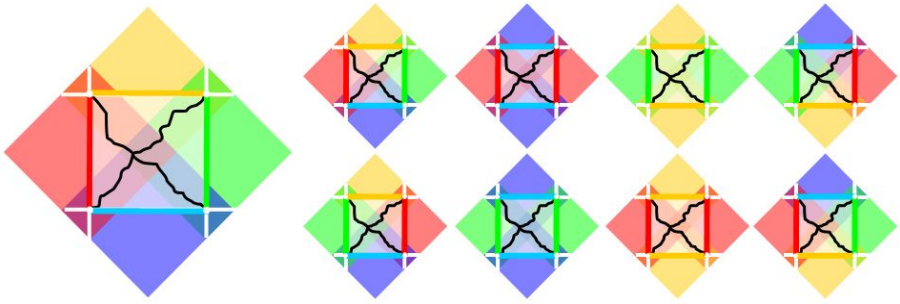
Our work is an approach close in spirit to that of Wang tiles in Cohen et al.'s [Cohen03], which is a pattern based texturing approach. Their differences have been discussed in the previous Chapter. In the following, we shall provide a short review on the generation of Wang tiles to facilitate subsequent discussion.

2.3.1 Wang Tile Approach

The approach of Wang tiles first extracts from an input texture a set of four small square patches, shown as red, green, blue and yellow in Figure 2.4 (a). These patches are rotated and arranged, with a small amount of overlap, as a diamond shape block, to cut a Wang tile from its center along four diagonals of the patches such as the one shown.

The overlapping regions are used to compute cutting paths shown as the black curves in Figure 2.4, starting and ending at the corners of the Wang tile, to define the four distinct areas to be contributed by each patch. Cutting paths can result in visible seams across patches and are thus computed with criteria, such as minimizing the difference in pixel values across the patches. Methods to compute cutting paths have been discussed, for example, [Efros01] and [Kwatra03].

A set of Wang tiles can be created where each member is obtained from a different arrangement of four but not necessarily distinct patches into a block. The paper shows the sets of arrangements to generate sets of 8 (such as in Figure 2.4(b)), 12, and 18 Wang tiles that can be used to tile any large area while maintaining no seams across two Wang tiles.



(a) **(b)**

Figure 2.4 Wang Tile Generation: (a) Four small square patches are combined into a diamond block to extract a Wang tile at the center; (b) A set of 8 Wang tiles constructed in this manner.

In this chapter, we briefly review the categories of texture synthesis algorithms and give an overview in this field. We illustrate the characteristics of non-parametric sampling methods and introduce the cutting path technique applied in patch-based

sampling approach. We present a classification of procedure methods in texture synthesis and introduce the closely related approach--Wang tile approach. In the following chapters, we introduce a tiling scheme that empowers users to synthesize nice textures using tiling algorithm.

Chapter 3 Tile Generation

In this and the next chapters, we introduce a new scheme of image tiling in detail. All steps from the creation of the initial ω-tiles to the process of tiling the plane with the tile set are presented. The process of constructing a fine ω-tile is presented in this chapter. We first introduce the overview of constructing ω-tiles for image tiling in Section 3.1. Then we discuss the difference of cutting path and cutting curve in Section 3.2 and finally introduce how to seek the cutting curve in each ω-tile in Section 3.3.

3.1 Constructing a Tile

Our approach first creates a block from an arrangement of four small patches obtained from the input texture $\mathcal{S}$, but without overlapping of the patches. Figure 3.1 shows an example of four such blocks A, B, C and D with different arrangements of the four patches and the intermediate tiles A_i, B_i, C_i and D_i cut from the center of these blocks. The seams across patches in each intermediate tile are removed by replacing the interior, other than the four corners, of the tile with another pattern from $\mathcal{S}$. This generates one out of the 16 ω-tiles in our experiment for texturing any large area; see Figure 3.1 where four such ω-tiles are shown as A_ω, B_ω, C_ω and D_ω, and they are used to tile a 2-by-2 area. The remaining part of this chapter discusses our approach to remove the vertical and horizontal seams across the patches.

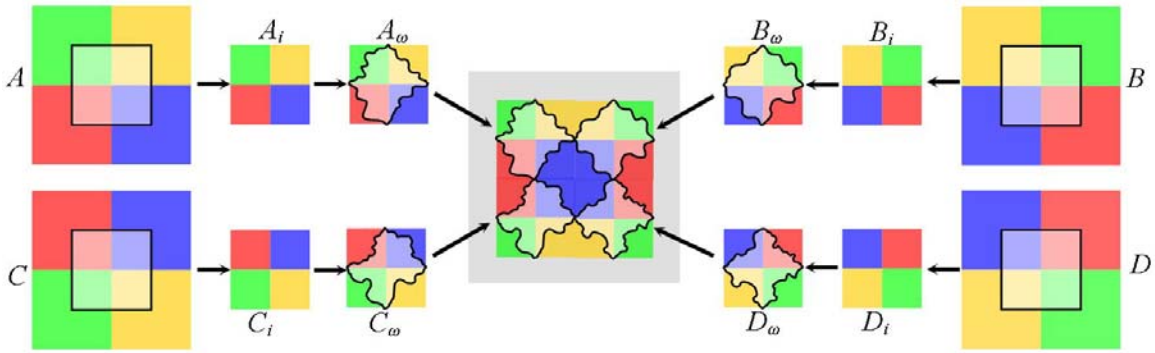


Figure 3.1 Tiling Process with ω-tiles. Four intermediate tiles A_i, B_i, C_i and D_i cut from blocks A, B, C and D, respectively, are used to generate four ω-tiles A_ω, B_ω, C_ω and D_ω. These are used to tile the center 2-by-2 area.

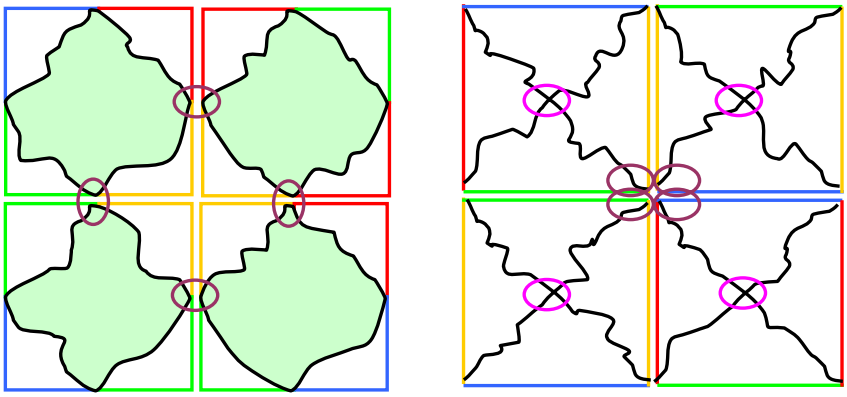
3.2 Cutting Path and Cutting Curve

We recall that the Wang tile approach minimizes the visibility of seams by overlapping patches to compute cutting paths. The effectiveness of a cutting path depends on the content at the overlapping region of patches. In general, the larger the overlapping region, the better the chance a good cutting path can be computed. It is, however, not clear how one should pick a fixed overlapping amount that is good for all the cutting paths in generating a set of Wang tiles.

In addition, we observe that the middle of each Wang tile is a junction where four patches meet. Such a kind of junction also indirectly occurs at the corners of each Wang tile in a tiling of a large area (see Figure 3.2(b)). They are seams that can be prominent to the naked eyes; contents around them should be controlled if possible. To avoid visible seams in a tile's center, one may compute a cutting path for a pair of diagonal corners, and then compute two good "half" cutting paths starting from the other pair of diagonal corners till they end at the first cutting path. This, however, may

not necessarily be effective to avoid visible seams. For junctions at corners of four Wang tiles, there is no good way known to control the cutting paths to minimize the visibility of junctions. The authors call this the ***corner problem***, and suggest expanding from, for example, a set of 8 Wang tiles to at least 64 Wang tiles with matching corners. This demands more texture memory during rendering.

Our approach aims to minimize the above corner problem, and in general, reduce the visibility of junctions. It thus generates ω-tiles of matching pattern at four corners. And, it removes the vertical and horizontal seams between patches by searching for a good pattern $\mathcal{C}$ to replace the interior portion inclusive of the seams in each intermediate tile. Such a pattern $\mathcal{C}$ is enclosed by a closed curve, termed cutting curve, passing through the middle point of each side of the tile. As a result, when these tiles are laid into a large area with matching sides, junctions where four textures meet appear at the middle point of each side (see Figure 3.2(a)). Due to the amount of four junctions per ω-tile, the total number of junctions is the same as that of Wang tile when synthesizing a large texture. However, the content at each junction in our approach can be controlled better. This is because two out of four patches meeting at a junction are obtained through searching a good $\mathcal{C}$ from $\mathcal{S}$ and thus are of good chances to minimize the visibility of junctions.



(a) **(b)**

Figure 3.2 Schematic drawing to show the corner problem: (a) artifacts in junctions controlled in ω-tile approach (b) artifacts in corners and junctions appear in Wang tile approach.

Variable	Meaning
$\mathcal{S}$	2D input sample image
$\mathcal{F}$	the set of four image patches to create the initial block
$\mathcal{I}$	intermediate tile cut from the central of initial block
$\mathcal{O}$	target patch searching from $\mathcal{S}$, to compute a cutting curve
$\mathcal{C}$	content in the circle of $\mathcal{O}$ to replace the corresponding content in $\mathcal{I}$
I, O	two given neighboring image patches which have an overlapping area
s, t	two adjacent pixels/nodes in the overlapping area
$I(s)$, $O(s)$	pixel value of s in image patch I, O respectively
d	gradient direction from s to t
$G_I^d(s)$	the gradient of pixel s in the patch I along d

Table 3.1 List of used variables and terms

3.3 Cutting Curve from a Graph

Now, we give a description of our algorithm for constructing a cutting curve from a graph. For variables/terms and their meanings, see Table 3.1. Specifically, for an intermediate tile $\mathcal{I}$ (Figure 3.3(a)), one way is to go through each pixel of $\mathcal{S}$ to extract

a candidate $\mathcal{O}$ starting from it, which has the same size as $\mathcal{I}$, (Figure 3.3(b)). Each candidate $\mathcal{O}$ is then superimposed onto $\mathcal{I}$ to compute a cutting curve (Figure 3.3(c)), passing through the middle points of the four sides of $\mathcal{I}$. The curve is also restricted to lie within a (pink) circle with the same center as $\mathcal{I}$ and having the diameter equal to the width of $\mathcal{I}$. The area of $\mathcal{I}$ within the cutting curve is to be replaced by the corresponding content $\mathcal{C}$ in $\mathcal{O}$. The best $\mathcal{C}$, among all candidates $\mathcal{O}$, is one where the seams along the cutting curve are least prominent.

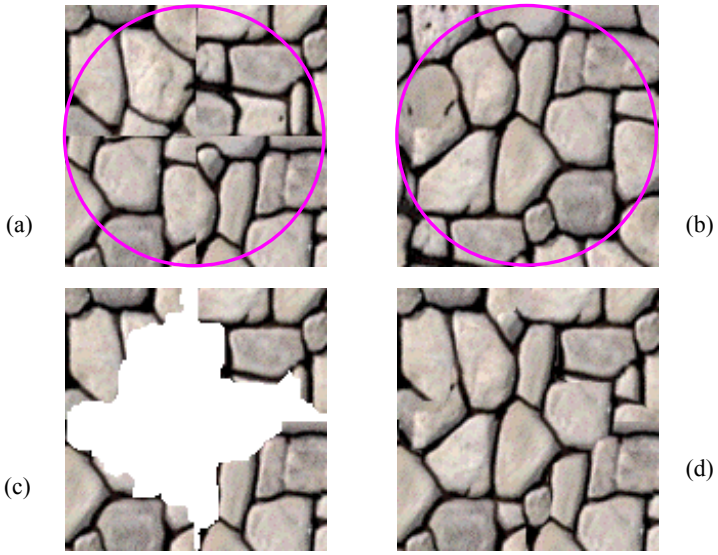


Figure 3.3 Tile Generation in our Technique. (a) Intermediate tile with vertical and horizontal seams across patches. (b) A candidate $\mathcal{O}$ extracted from $\mathcal{S}$. (c) The intermediate tile of (a) with its interior removed along a cutting curve. (d) The removed part is replaced by the corresponding part $\mathcal{C}$ of $\mathcal{O}$ to obtain an ω-tile.

We adapt Kwatra et al. [KSET03] to derive $\mathcal{C}$. Very briefly, their work computes a cutting path for a given pair of patches I and O that overlap at some pixels as a graph

problem. It regards pixels in the overlap as graph nodes, and the link between any two neighboring pixels as graph edge. Each edge is assigned a weight as a flow capacity, which is the sum of the transition errors of the two nodes divided by the sum of their gradients in the patches:

$$Weight(s,t) = \frac{\|I(s) - O(s)\| + \|I(t) - O(t)\|}{\|G_I^d(s)\| + \|G_I^d(t)\| + \|G_O^d(s)\| + \|G_O^d(t)\|} \tag{3-1}$$

where *s*, *t* are neighboring pixels in the overlap; $G_I^d(s)$ denotes the gradient of pixel *s* in the patch *I* along *d* (gradient direction from *s* to *t*), and $G_I^d(t)$, $G_O^d(s)$, $G_O^d(t)$ are similarly defined. Then, the cutting path is the min-cut which cuts the graph edges with the minimum sum of flow capacities from source pixels of *I* to sink pixels of *O*.

Applying the above to our $\mathcal{I}$ and $\mathcal{O}$ to compute a (green) cutting curve (as in Figure 3.4), we have (red) pixels of $\mathcal{I}$ outside the (pink) circle defined as sources, two vertical columns and two horizontal rows of (blue) pixels at the center of $\mathcal{O}$ defined as sinks, and weights of edges incident to white pixels defined as in Equation (3-1). Notice that the area involved in the search for a cutting curve is generally larger than that of overlapping region in generating a Wang tile. Thus, the consequence of less visible seams within an ω-tile is expected here, though with higher pre-processing cost.

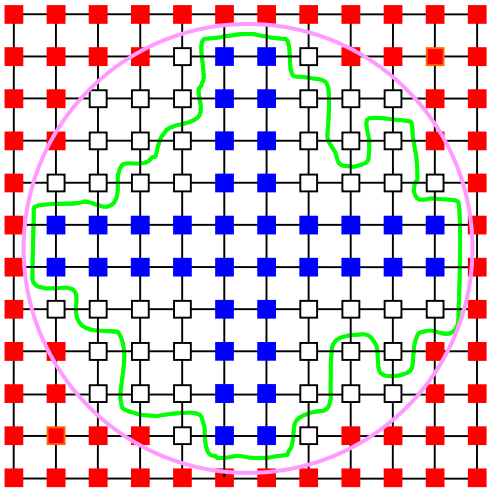


Figure 3.4 Cutting curve (green) generation in the graph.

In practice, the above cutting curve computations for all choices of $\mathcal{O}$ can be costly. We can adapt heuristics to choose just one or a few $\mathcal{O}$ to perform the computation of a cutting curve to obtain a good $\mathcal{C}$. One possible choice is to pick $\mathcal{O}$ with the smallest sum of weights for pixels around the (pink) circle. This is an attempt to select $\mathcal{O}$ that is similar to $\mathcal{I}$ and thus with a good chance of obtaining a cutting curve with less visible seams. We use this in our experiments reported in Chapter 5. Another choice is to pick $\mathcal{O}$ with the smallest sum of weights for pixels near the middle of the four sides. This is to emphasize the importance of having matching content at junctions.

In this Chapter, we introduce the process of constructing an ω-tile. From the constructing scheme, we ensure the potential neighboring ω-tile is combined without line seams across the patches. Then, the cutting curve is computed in the large space of each ω-tile to remove the internal seams in each ω-tile. Through this technique, we

can find our approach is better to control the corner problem than Cohen et al.'s Wang tile approach. In the next chapter, we will present how to form the sets from the ω-tiles and make them stochastically distributed in the output image.

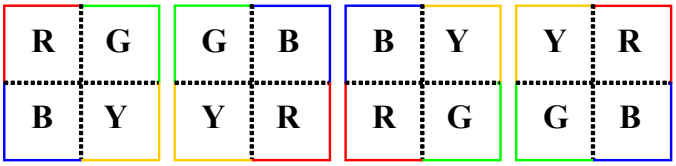
Chapter 4 Tile Set Formation

In this chapter, we introduce how to form various tile sets that can be tiled to synthesize large textures. We focus on introducing the 16-tile set formation. We use the approach introduced in Chapter 3 to generate a set of ω-tiles from different arrangements of the four sample patches. Recall our goal is to use such a small set while preserving the global distribution pattern of the input texture in a synthesized texture. In texture synthesis, ω-tiles are used to tile a large area from left to right and top to bottom. We use the set of patches $\mathcal{F}=\{R,G,B,Y\}$ obtained from $\mathcal{S}$ to form blocks and then to extract intermediate tiles to derive ω-tiles.

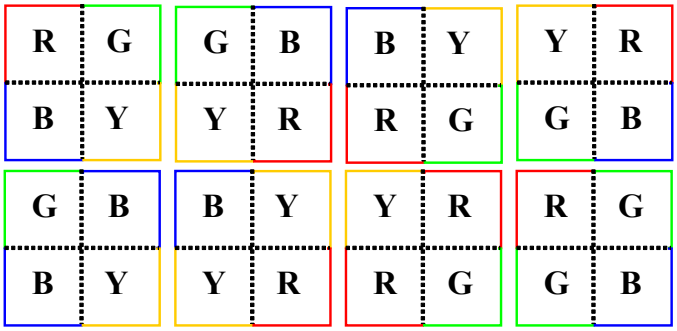
4.1 Sets of 1, 4, and 8 Tiles

It is clear that the smallest such set with only 1 ω-tile that is generated from a block with the same four patches will not serve our purpose, while the largest such set with 4^4 is unnecessary. Figure 4.1 shows examples of a set of 4 ω-tiles, and a set of 8 ω-tiles that can tile any large area. However, for the former, once the top-leftmost ω-tile is laid, the rest of the ω-tiles are also determined from left to right, top to bottom; for the latter, there is an additional choice of ω-tile for each one at the leftmost column. Thus, both tend to generate undesirable repetitive patterns. One approach to overcome this is to take the intermediate tile that generates each ω-tile in Figure 4.1 to compute

another ω-tile using a different $\mathcal{O}$. This way effectively doubles the number of ω-tiles, but provides at least two choices of tiles at each tiling step.



(a) A set of ω-tiles of size 4



(b) A set of ω-tiles of size 8

Figure 4.1 Sets of ω-tiles of size 4 and size 8.

4.2 Sets of 16 and 32 Tiles

Alternatively, we discuss in the following an approach to generate directly a set of 16 ω-tiles that is effective in synthesizing textures that preserve the global distribution of the input texture. With such a set, there are always at least two choices of ω-tiles at each tiling step of the top row and the leftmost column. We can again have a variant to this where each intermediate tile is used to generate two ω-tiles using two different $\mathcal{O}$ to obtain a total of 32 ω-tiles. This larger set provides at least 2 choices of ω-tiles at each tiling step to synthesize a texture.

Figure 1.1 shows a possible set of ω-tiles of size 16, and Figure 4.2 shows two other examples of the same size created with the same design principle. The following details such principle as four properties of the sets of 16 ω-tiles that can tile any large area without seams across the boundaries of ω-tiles. One can possibly devise other properties to obtain different kinds of ω-tile sets of possibly different sizes too.

An ω-tile has four corners contributed by patches in $\mathcal{F}$. We use a tuple $\langle -,-,-,- \rangle$ of four elements to capture the four corners where the first element captures its top-left corner; second element, bottom-left; third element, top-right; and fourth element, bottom-right. We use "–" in a tuple to mean "don't know currently". For example, we have $\langle R,G,-,- \rangle$ to mean a tile with left column occupied by *R* on top of *G*, while we have no information about its right column. Let $a \downarrow b$, where $a,b \in \mathcal{F}$, denotes there is no tuple of the form $\langle a,b,-,- \rangle$. For example, we use $R \downarrow Y$ to mean no tuple of the form $\langle R,Y,-,- \rangle$ in the set of ω-tiles. In generating a set of ω-tiles, we adopt the following four properties:

(1) Each ω-tile $\langle a,b,c,d \rangle$ where $a,b,c,d \in \mathcal{F}$ is such that $a \neq b$, $a \neq c$, $b \neq d$ and $c \neq d$. That is, no two quadrants sharing a side are from the same patch of $\mathcal{F}$.

(2) (i) For any $a \in \mathcal{F}$, there is one $a' \in \mathcal{F}$, such that $a \downarrow a'$ and $a \neq a'$; (ii) for any $a,b \in \mathcal{F}$ and $a \neq b$, we have $a \downarrow a'$ and $b \downarrow b'$ where $a',b' \in \mathcal{F}$ and $a' \neq b'$. In

other words, each patch placed at the top-left quadrant of a ω-tile has a forbidden counterpart at the bottom-left quadrant, and two different patches have different forbidden counterparts.

With the above two properties, we have two types of situations for distinct members $a,b,c,d \in \mathcal{F}$: (i) $a \downarrow b,\ b \downarrow c,\ c \downarrow d$ and $d \downarrow a$ in one (such as Figure 4.2(a)) and (ii) $a \downarrow b,\ b \downarrow a,\ c \downarrow d$ and $d \downarrow c$ (such as Figure 4.2(b)) in the other. Also, there are only $4 \times 2 = 8$ possible tuples $\langle a,b,-,-\rangle$ where $a \neq b$.

(3) For each $\langle a,b,-,-\rangle$ in the set, we can find a tuple $\langle a,-,b,-\rangle$ in the set too, and vice-versa.

From property (3), we have, for example, $\langle R,G,G,-\rangle$, $\langle R,G,B,-\rangle$, $\langle R,B,G,-\rangle$ and $\langle R,B,B,-\rangle$ as in Figure 4.2(a). There are now 16 tuples with "don't know currently" their fourth elements. The next property defines them:

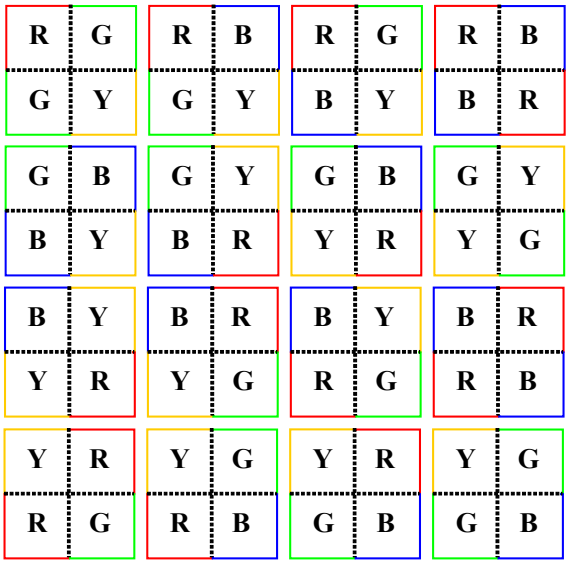
(4) For each of the 16 tuples $\langle a,b,c,-\rangle$, we select a d, where neither $b \downarrow d$ nor $c \downarrow d$ is true, to obtain one ω-tile $\langle a,b,c,d\rangle$.

Continuing with the example up to property (3), we get $\langle R,G,G,Y\rangle$, $\langle R,G,B,Y\rangle$, $\langle R,B,G,Y\rangle$ and $\langle R,B,B,R\rangle$ where the first and the last tuples have the alternative choices of $\langle R,G,G,B\rangle$ and $\langle R,B,B,Y\rangle$, respectively. Formally, we show in the next

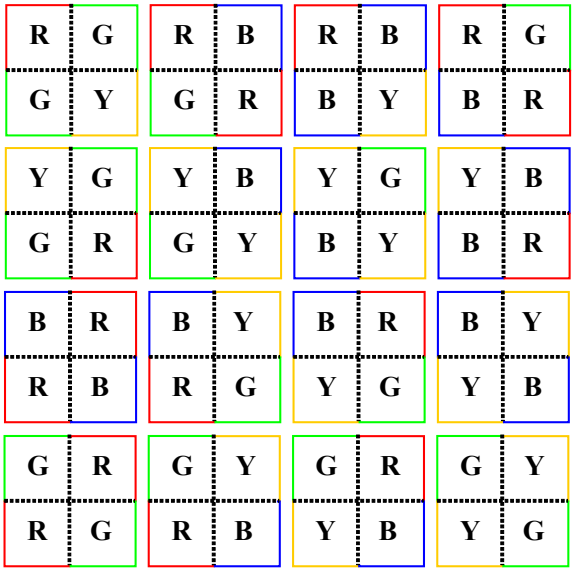
paragraph that property (4) is well defined in generating sets of ω-tiles such as those in Figure 4.2.

There are two types of tuples obtained from properties (1) to (3): $\langle a,b,b,-\rangle$ and $\langle a,b,c,-\rangle$ where $a,b,c \in \mathcal{F}$ and $a \neq b \neq c$. For $\langle a,b,b,-\rangle$, it is clear that property (4) has two choices to assign a patch to the fourth element, and thus there is no issue. For $\langle a,b,c,-\rangle$, we derive from property (3) that $a \downarrow c$ is not true. Thus, by property (2)(i), $a \downarrow d$ for a unique $d \in \mathcal{F} - \{a,b,c\}$. By property (2)(ii), neither $b \downarrow d$ nor $c \downarrow d$ is true, so property (4) can thus generate $\langle a,b,c,d\rangle$ (and possibly $\langle a,b,c,a\rangle$ if neither $b \downarrow a$ nor $c \downarrow a$ is true).

We next show that a set $\mathcal{W}$ of ω-tile obeying the four properties can tile any large area without seams across boundaries of ω-tiles. That is, at any tiling step to place a tile at position (*i*, *j*), i.e. row *i* and column *j*, we can find one in $\mathcal{W}$ to match (if any) the bottom side of the tile at (*i*–1, *j*), and (if any) the right side of the tile at (*i, j*–1). If the needed tile is of the form $\langle a,b,b,-\rangle$, then property (4) applying to the tile at (*i, j*–1) means that $a \downarrow b$ is not true, and thus the needed tile is in $\mathcal{W}$ by properties (1) to (3). If the needed tile is of the form $\langle a,b,c,-\rangle$ where $a \neq b \neq c$, then we have as before $a \downarrow b$ is not true, and $a \downarrow c$ is not true by property (4) applying to the tile at (*i*–1, *j*), and thus the needed tile is in $\mathcal{W}$ by properties (1) to (3).



(a)



(b)

Figure 4.2 Examples of ω-tile sets obeying properties (1) to (4). In (a), we use $R \downarrow Y$, $G \downarrow R$, $B \downarrow G$, and $Y \downarrow B$; in (b), we use $R \downarrow Y$, $Y \downarrow R$, $B \downarrow G$, and $G \downarrow B$.

4.3 Sets of 36 and 72 Tiles

Here we still use a tuple $\langle -,-,-,- \rangle$ of four elements to capture the four corners of an ω-tile. We use " | " to denotes there are some options for choosing the elements. For

example, we have $\langle a \mid b, -, -, - \rangle$ to mean we can choose either a or b for the first element of this tuple. To form the sets of 36 and 72 tiles, we still adopt the property (1) of the Section 4.2. That is, each ω-tile $\langle a, b, c, d \rangle$ where $a, b, c, d \in \mathcal{F}$ is such that $a \neq b$, $a \neq c$, $b \neq d$ and $c \neq d$.

We construct the tile set by counting the number of possible tuples occurring in each position (i, j). Once the tile set is formed such that in each tiling step user can find at least a feasible tile, a large plane can be textured without seams using it. For the tuple $\langle -, -, -, - \rangle$ at position (i, j) where i ≠1 and j ≠1, the tuples at positions (i–1, j) and (i, j-1) become joint constraints for us to choose its four elements. For its first two elements in tuple $\langle a, b, -, - \rangle$, where $a \neq b$, there are in total 12 choices. According to our tiling scheme, the third element of the tuple in position (i, j-1) is the same as the second element of tuple in the position (i-1, j). Due to their joint constraint, for the tuple in position (i, j) which has chosen its first two elements, there are correspondingly three choices for its third element. Therefore, determined by the first three elements (in the format of $\langle a, b, c, - \rangle$ and $\langle a, b, b, - \rangle$), we need 12×3 types of tuples to form the tile set for all positions. These tuples can be divided into two patterns $\langle a, b, c, a \mid d \rangle$ and $\langle a, b, b, a \mid c \mid d \rangle$ (according to whether the second and third elements are the same), where a, b, c and d are distinct members of $\mathcal{F}$. For each of such 36 types of tuples, to determine its fourth element we can choose one format from its corresponding pattern. For example, tuple $\langle R, G, G, Y \rangle$ is one chosen from the pattern $\langle R, G, G, R \mid B \mid Y \rangle$, and tuple $\langle R, G, B, Y \rangle$ is one chosen from the

pattern $\langle R, G, B, R \mid Y \rangle$. Although this step randomly chooses one from the corresponding pattern, it will not create irresolvable constraints for the following tiling steps, because during defining the first three elements of the tuples, we have counted and created all possible combinations. Thus, we form the set of 36 tiles, and during each tiling step, user can use at least one feasible tile from the tile set for each position (i, j), where i ≠1 and j ≠1 as the tile set of 36 tiles satisfies all possible constraints in positions (i–1, j) and (i, j-1). In this way, for the position in the leftmost column and top row except that in top left corner, there are 3 available tiles from the tile set as there is only one directional constraint for user to choose its four elements.

To form the tile set with at least 2 choices of ω-tiles at each tiling step, we can apply two methods. First, as we have discussed, for each intermediate tile, we generate two ω-tiles using different $\mathcal{C}$ to obtain a total of 72 ω-tiles from the above 36 ω-tiles. Second, for the two patterns $\langle a, b, c, a \mid d \rangle$ and $\langle a, b, b, a \mid c \mid d \rangle$, we choose both formats from the former and two out of three from the latter. Therefore, using the predefined first three elements, we have two choices for the fourth element for each tuple. In this way, we create the set of 72 ω-tiles, which has the same property as that created with the first approach.

In this chapter, we present methods to form various tile sets, from which we can tile the whole plane. We focus on our presentation about the formation of tile set of 16 tiles, as we apply this method to synthesize various images of different patterns in our experiment. In the following chapter, we will introduce our experimental details and results.

Chapter 5 Experimental Results

We implement our ω-tile generation algorithm using C++ under MS Windows on a Pentium P4 1.6GHz with 256 MB main memory. The sample textures are from MIT VisTex set [MIT] and Texture Library of Paul Bourke [Bourke]. Section 5.1 shows our tiling process in a graphics user interface (GUI). Section 5.2 presents the synthesized results for samples of various patterns, the result comparison between ω-tile approach and Wang tile approach, and the results created with the tile sets of different size. Section 5.3 discusses some limitations of this algorithm and the possible solutions. Refer to our project webpage: http://www.comp.nus.edu.sg/~tants/w-tile/ for more results of larger synthesized textures and comparisons.

Compared with other techniques, this algorithm is quite trivial to be implemented. Although the results currently shown in our experiments are synthesized without hardware support, the synthesis speed is still very fast. Usually, the process of tiling a 4-by-4 area from 16 ω-tiles each of 64×64 pixels only needs about one minute. Most of the time is spent on searching the appropriate patches $\mathcal{O}$ to replace the area of $\mathcal{C}$ within the cutting curve of each tile. In application, users can synthesize large space textures by using our algorithm to generate ω-tiles in the preprocessing stage. That is, user can create the set of tiles of appropriate size in the preprocessing stage, keep the tile combination rule and then tile the plane on the run.

5.1 Tiling Process Overview

In implementation, we provide a friendly and easy-to-use graphics user interface (GUI) to synthesize the images. In this section, we present this GUI showing the whole process of the tiling framework from uploading the input sample to generating the output image plane (see Figure 5.1).

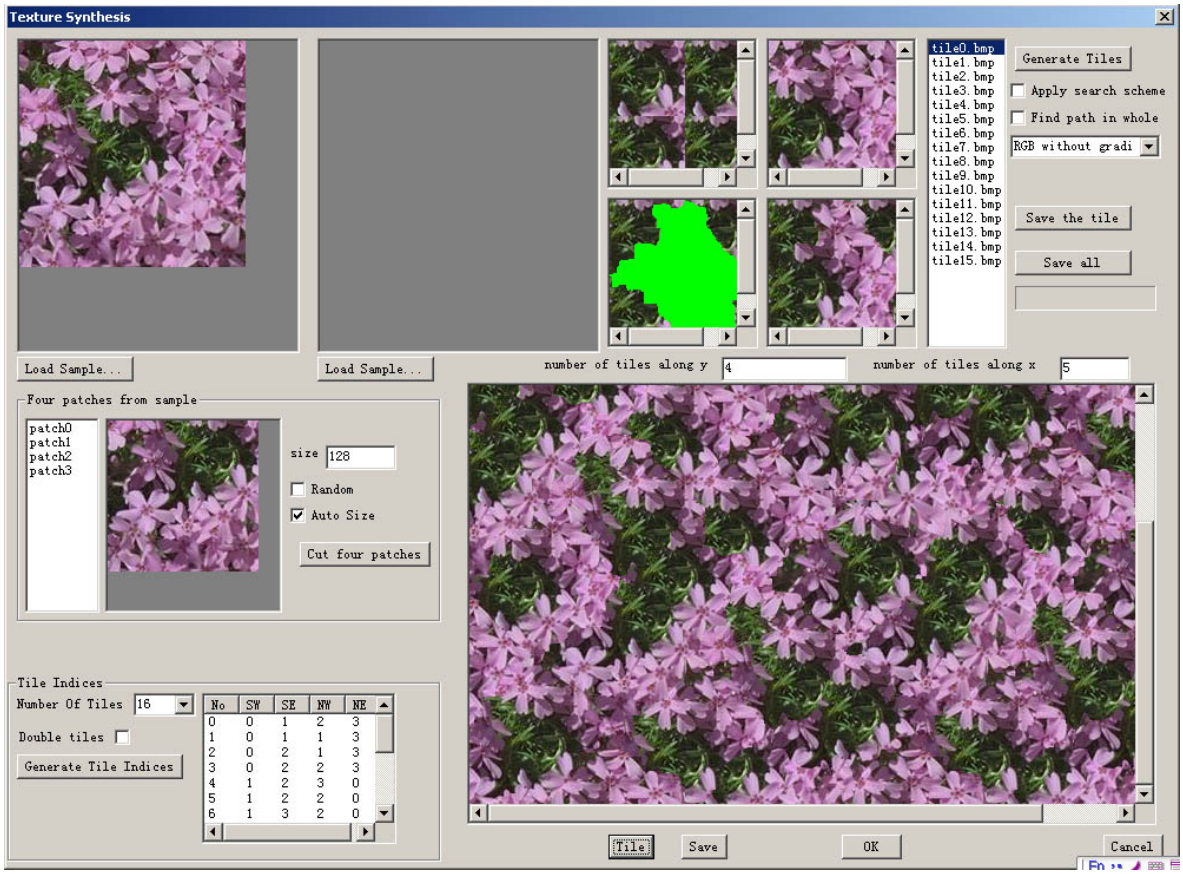


Figure 5.1 Overview of tiling process in graphics user interface (GUI)

Figure 5.1 demonstrates that a user can easily synthesize a nice image with our technique. User first loads one sample (or two samples to create multiple patterns) in the North West window. Then four image patches are extracted from the sample in the West South window. The pop-up dialog bar is for user to choose the number of tiles in

the tile set. The *Size* controls the dimension (width and height) of the image patch obtained from the source. The dialog bar docked at the bottom shows the indices on each intermediate tile $\mathcal{I}$, which are listed in the *SW, SE, NW, NE* order. Then in the North East part, for each tile, the windows show the intermediate tile $\mathcal{I}$, the targeted image patch $\mathcal{O}$ from the sample, the shape of the cutting curve and the final seamless tile after cutting. User can choose searching the target patch or randomly cutting them from the sample. At the same time, in the pop-up menu, user can choose various methods to assign the weights for the graph in seeking the cutting curve. The dialog bar shows the file names of the seamless tiles after cutting. Finally in the South East window, it shows the synthesized image created by the tile set. User can save the generated results in .BMP format.

5.2 Synthesized Results

The results of the synthesis process for a wide range of input textures and some result comparisons are shown in this section. This algorithm is particularly effective for stochastic textures, and the performance is also good on semi-structured textures. For structured textures, such as bricks, mostly due to the way that we randomly combine image patches $\mathcal{F}$ to create the tiles, the results may not appear fine.

5.2.1 Stochastic Texture Samples

The following figures show a group of synthesized results of stochastic pattern. All the results are generated by tiling a 4-by-4 area with the set of 16 tiles. The tile choice in

the leftmost column and top row is stochastic from two options, while the tile choice in the other place is deterministic. The samples are all positioned at top left corner.

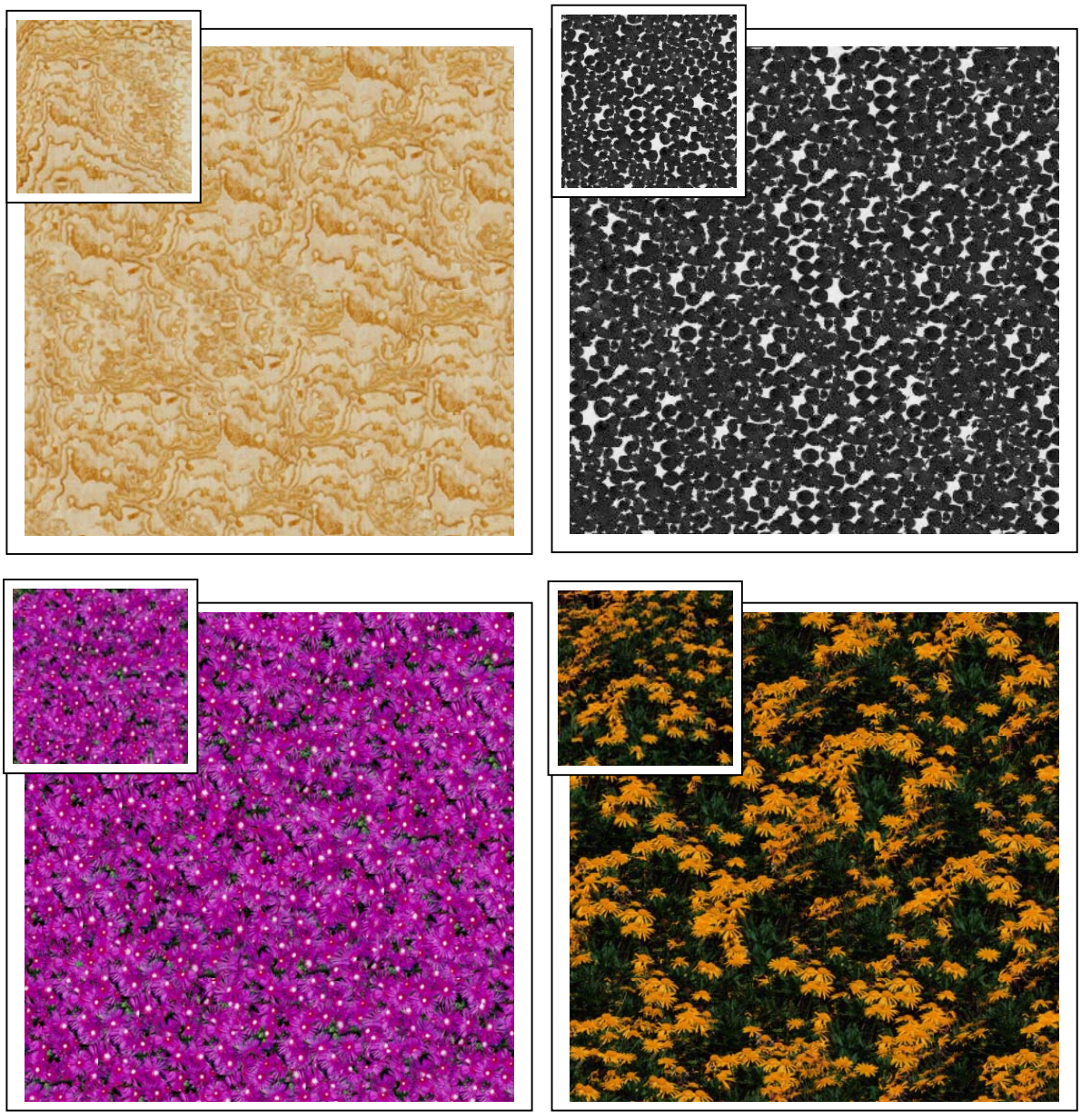


Figure 5.2 Texture synthesis results of our ω-tile approach with input of stochastic pattern. In each case, top left corner is the input texture of 256×256 pixels, and the result is a 4-by-4 area tiled with 16 ω-tiles each of 128×128 pixels.

Figure 5.3 Texture synthesis results of our ω-tile approach with input of stochastic pattern. In each case, top left corner is the input texture of 256×256 pixels, and the result is a 4-by-4 area tiled with 16 ω-tiles each of 128×128 pixels.

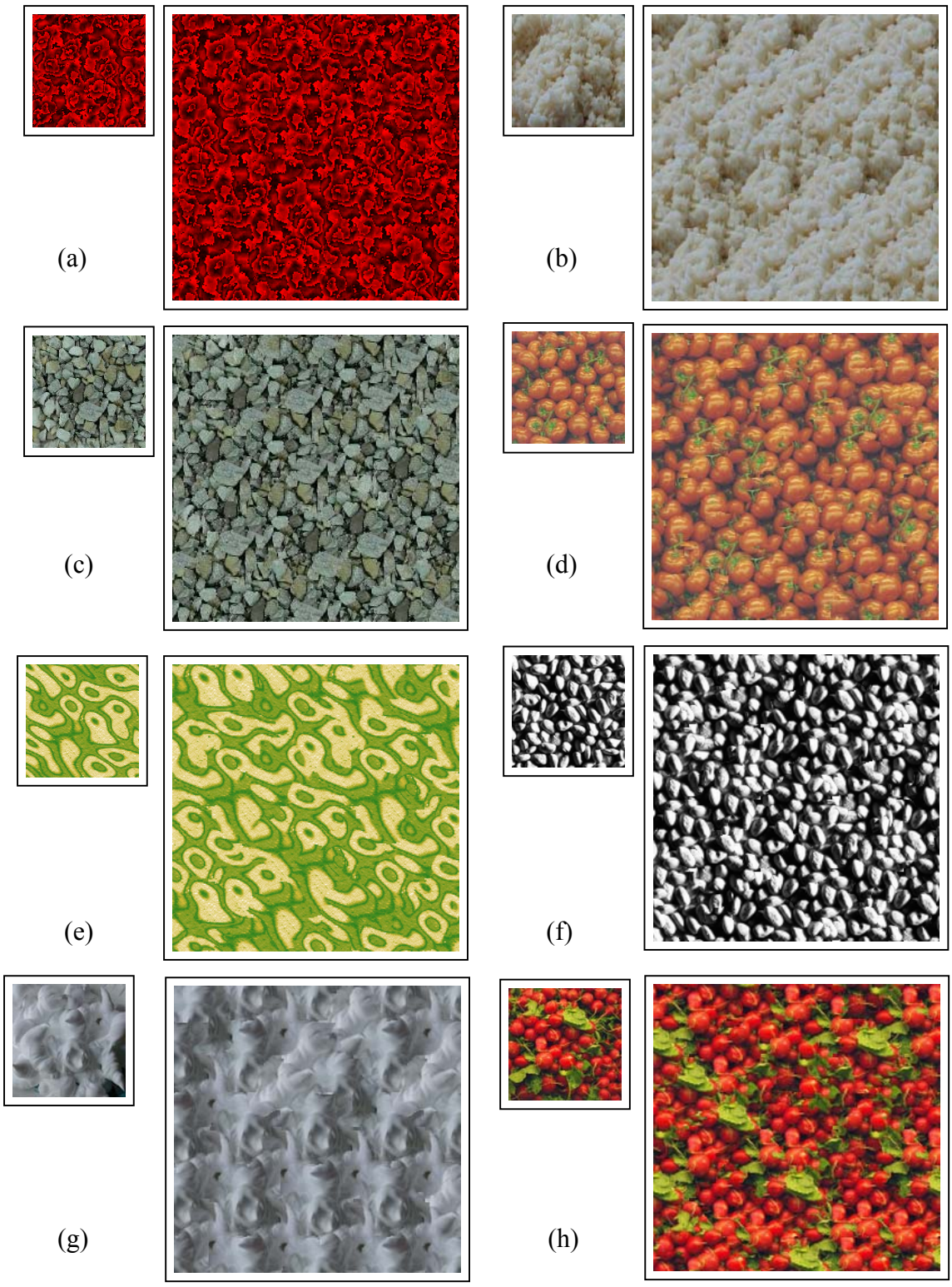


Figure 5.4 Texture synthesis results of our ω-tile approach with input of stochastic pattern. In each case, left is the input texture of 128×128 pixels, and right is the result of a 4-by-4 area from 16 ω-tiles each of 64×64 pixels.

From these examples, it can be illustrated that our approach is quite effective to tile stochastic textures. The synthesized images are quite naturally smooth without local line seams and the pattern of high repetition. At the same time the cutting curve can efficiently preserve the pattern of the primitives in the sample images $\mathcal{S}$. In some samples such as (d) and (h) in Figure 5.4, the cutting curve may not have a better choice but cut through the primitives of the image. The visual effects of such results are still quite good because the stochastic distribution of such a lot of small primitives can mask the artifacts in the image. Then in the following subsection, we present some semi-structured examples created with our technique.

5.2.2 Semi-structured Texture Samples

The following figures show the results of semi-structured samples. From the examples shown here, we can find the advantage of our technique to tile textures with such distribution. Same as the above section, all the results are generated by tiling the 4-by-4 area with the set of 16 tiles; the tile choice in the leftmost column and top row is stochastic from two options, while in the other places is deterministic.

Figure 5.5 Texture synthesis results of our ω-tile approach with input of semi-structured pattern. In each case, top left corner is the input texture of 256×256 pixels, and the result is a 4-by-4 area tiled with 16 ω-tiles each of 128×128 pixels.

Figure 5.6 Texture synthesis results of our ω-tile approach with input of semi-structured pattern. In each case, top left corner is the input texture of 256×256 pixels, and the result is a 4-by-4 area tiled with 16 ω-tiles each of 128×128 pixels.

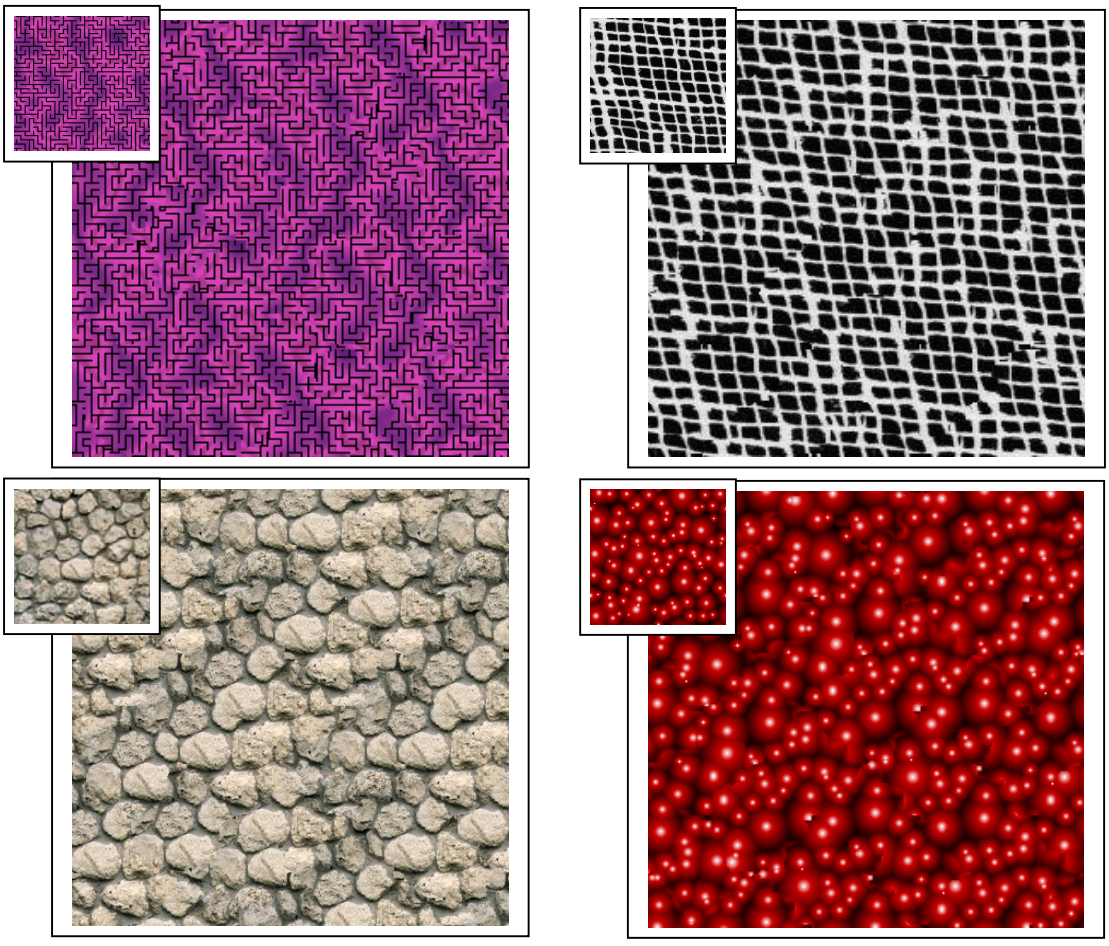


Figure 5.7 Texture synthesis results of our ω-tile approach with input of semi-structured pattern. In each case, left is the input texture of 128×128 pixels, and right is the result of a 4-by-4 area from 16 ω-tiles each of 64×64 pixels.

Semi-structured textures are always the hardest for statistical texture synthesis [Efros01]. Because such textures have repetitions while without excessive repetitions, the pixel based statistical sampling method cannot efficiently find appropriate neighborhood for the local pixels and thus leads to "wrong" targets. At the same time, simple tilting cannot create good results for this class either, as for the distortions across the combined patches. Our technique efficiently preserves the patterns of the sample images by using patches to synthesize image, and preserves the smooth transition across the patches using the tile combination scheme. Therefore, the synthesized results are quite natural to human visual perception. For the last category, we present some synthesized results for structured textures.

5.2.3 Structured Texture Samples

The following figures show the results for structured textures. We can find the results are fairly good for some samples. Same as the above section, all the results are generated by tiling the 4-by-4 area with the set of 16 tiles; the tile choice in the leftmost column and top row is stochastic from two options, while in the other place is deterministic.

Figure 5.8 Texture synthesis results of our ω-tile approach with input of structured pattern. In each case, top left corner is the input texture of 256×256 pixels, and the result is a 4-by-4 area tiled with 16 ω-tiles each of 128×128 pixels.

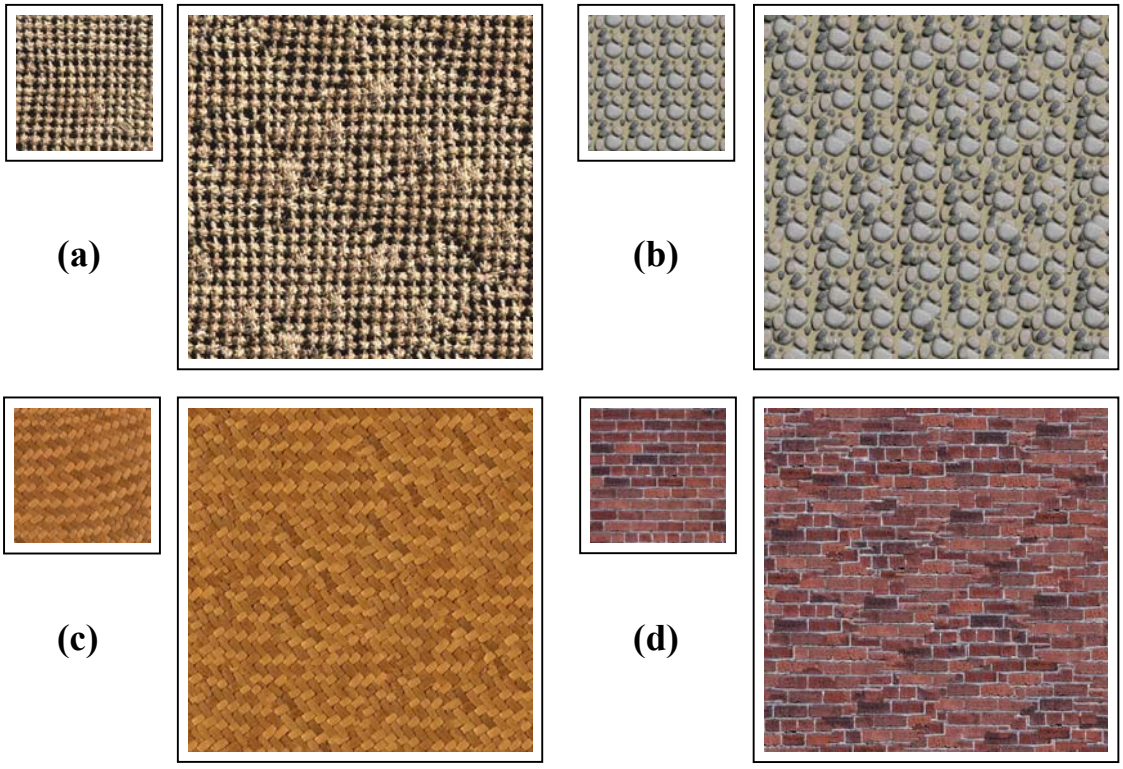


Figure 5.9 Texture synthesis results of our ω-tile approach with input of structured pattern. In each case, left is the input texture of 128×128 pixels, and right is the result of a 4-by-4 area from 16 ω-tiles each of 64×64 pixels.

Usually, for structured texture that has highly repetitive pattern, simple tiling has been able to create good results. Due to requirement of our method that user randomly combines four image patches to create the intermediate tiles, it will create mismatch in structure between the four image patches. Cutting curve as well as cutting path is not effective to adapt such mismatch. Therefore, the generated image cannot show quite regular pattern. Examples of (a), (c), (d) in Figure 5.9 and keyboard in Figure 5.8 illustrate this problem. However, if the pattern of sample image is quite complicated, such as the wire netting and wallpaper in Figure 5.8 or the patterns have slim horizontal and vertical lines, such as the right fabric in the second row of Figure 5.8, the mismatch distortions will be masked by such details and can not be easily detected by human eyes.

Therefore, in some of the structured examples, our technique can also create good results.

From our tests on input textures of stochastic, semi-structured to structured nature, we observe in general that synthesized textures from ω-tiles can preserve well the global distribution of the underlying pattern of the input textures, especially for the first two types. For input texture of structured nature, such as a brick texture of a wall, the alignment of the bricks may not be good in a set of ω-tiles. In such a case, it may be necessary to interactively rather than randomly select the four patches in $\mathcal{F}$ when creating ω-tiles. In general, for a highly structured input texture, our approach can be inferior, as it tends to "destroy" the structure with more patterns embedded in each ω-tile.

5.2.4 Result Comparison

Two comparisons are presented in the subsection. First we compare results generated in Wang tile approach and ours. Then for the variation of the size of the tile set, we present one example to show their visual difference. More examples of result comparison can also be found in our project webpage [Wen04].

Wang tile Vs ω-tile

Burke has implemented Cohen's Wang tile approach in [Burke03]. In the following Figure 5.10, the results of Wang tile approach are generated by the code from http://www.mle.ie/~rob/wang/.

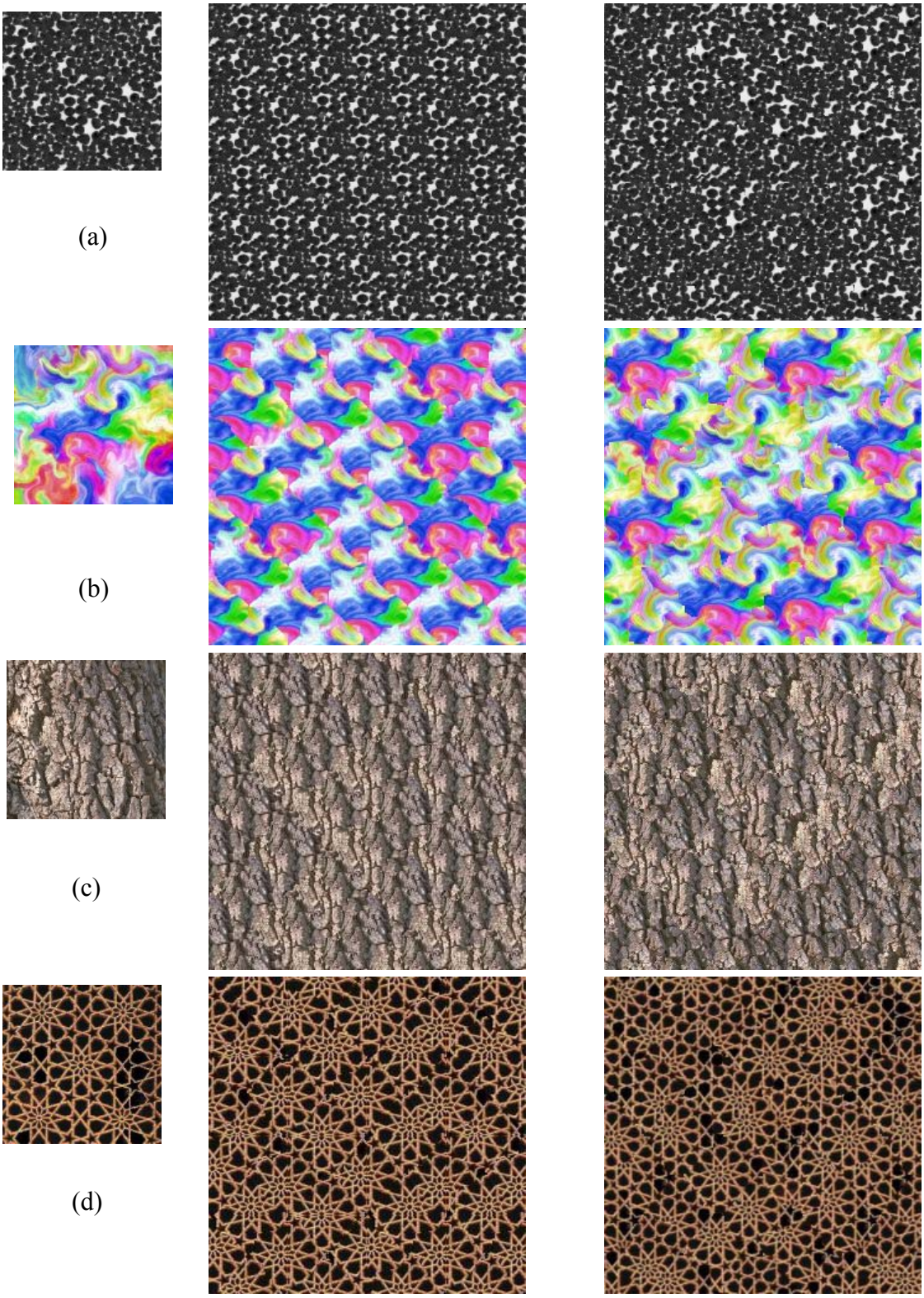


Figure 5.10 Comparison of synthesized textures by Wang tiles and ω-tiles. In each case, left shows input texture of 128×128 pixels, middle shows a result of tiling a 4-by-4 area selected from 18 Wang tiles each of size 64×64 pixels, and right shows a result of tiling a 4-by-4 area selected from 16 ω-tiles of size 64×64 pixels.

Note that we do not use the function of one pixel overlap across the tiles designed in Paul's GUI as this overlap creates obvious line seams in the output image. The tile code list for placing the Wang tiles in his GUI is created by us according to the tile combination rule of Wang tile approach. Our approach uses four patches each of size 64×64 pixels whereas Wang tile approach uses four patches each of size 45×45 (45 is the integer most close to $64/\sqrt{2}$). Both approaches generate tiles of size 64×64 pixels.

In general, our approach works quite well when applied to texture synthesis and produce results that are equal or better than those created by Wang tile approach for stochastic and semi-structured textures. For the sample (a) in Figure 5.10, the objects are small and stochastically distributed in the plane, thus the corner problem is hard to be detected. Therefore, both approaches achieve good visual effects. However, for all the first three samples, it is obvious that our approach can better preserve the distribution of objects in the image than Wang tile approach. Furthermore, the seams in ω-tiles are generally not very visible, especially for those input textures of stochastic and semi-structured nature. As for the fourth sample, the result of our approach may be no better than that of Wang tile approach because it embeds more patterns in each ω-tile and thus breaks the structure.

Furthermore, Figure 5.11 can illustrate obvious advantages of ω-tile approach over Wang tile approach. In this example, abrupt color changes occur at the narrow overlapping areas of the four patches in Wang tiles and they are quite noticeable as such seams always occur in the diagonal direction. It is also quite obvious that in the junctions of four patches in each Wang tile and the corners of adjacent Wang tiles, this approach cannot preserve continuous texture feature of the colorful waves and even

generates four discontinuous wave segments of distinct colors in some of these junctions/corners. However, the internal seams in ω-tiles are replaced by a wave texture from the wave sample to keep its feature continuous through cutting curve technique. The mismatches of the junctions in each side of ω-tile are also well controlled by careful searching for the texture $\mathcal{O}$ from the large wave texture.

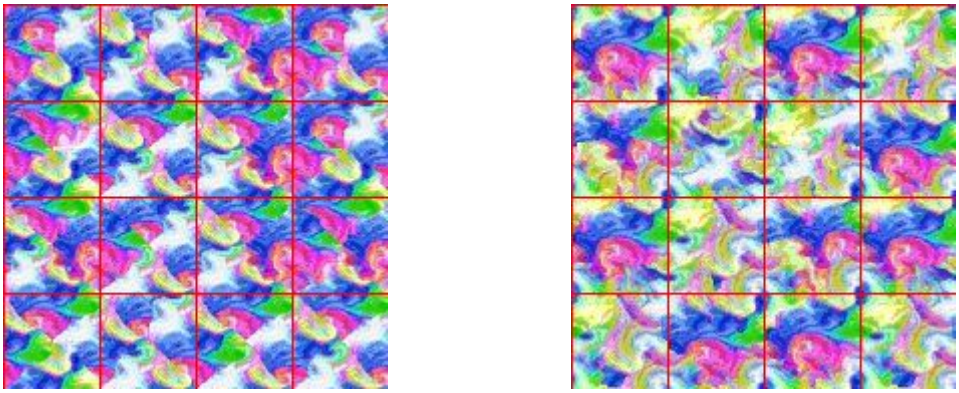
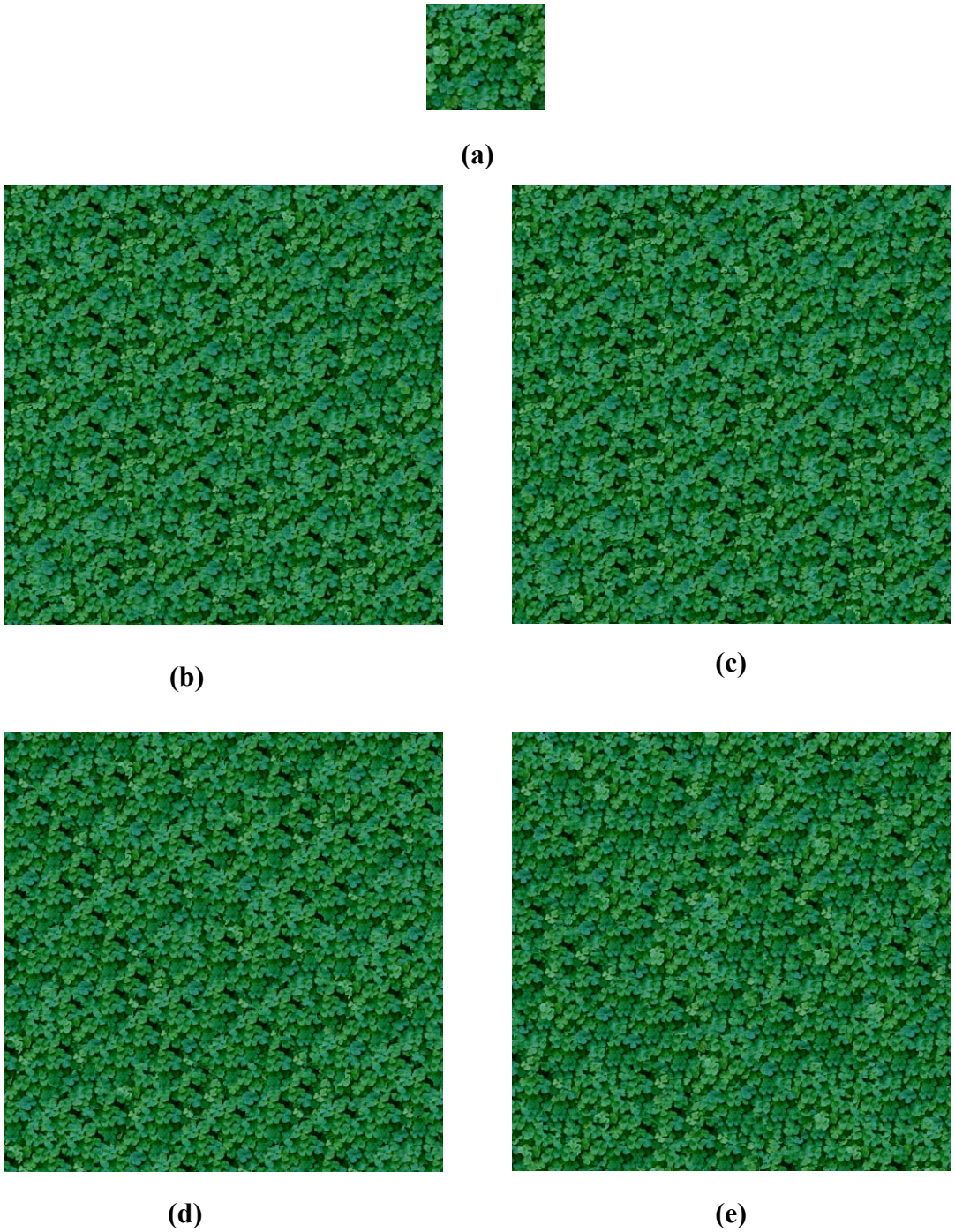


Figure 5.11 Visual comparison of two approaches to deal with corner problems. Left is the result by Wang tile approach and right is by ω-tile approach.

Tile Set Variation

One key parameter for our algorithm is the number of tiles in the tile set. As we have discussed in Chapter 4, 16 ω-tiles can tile the plane and in each of the tiling step except in the position of top row and leftmost column, there are only one tile available. We can obtain 32 ω-tiles, where each intermediate tile of the former tile set is replaced by different $\mathcal{O}$. Using the approach introduced in Chapter 4, we can also create the tile set of 36 ω-tiles. By the same means, we can create the tile set of 72 tiles. For the set of 36 tiles, we still have only one option for each tiling step in the place except top row and leftmost column. While, for the set of 32 and 72 ω-tiles, there are two options for these

tiling places. In this section, we present one example in Figure 5.12 to show the visual difference of the synthesized images selected from different tile sets. The size of the sample is 128×128 pixels, ω-tiles are 64×64 pixels, and the synthesized results are 512×512 pixels.



(a)

(b) **(c)**

(d) **(e)**

Figure 5.12 Synthesized leaves by tile sets of various size: input sample (a), tiling results created by tile set of 16 tiles (b), 32 tiles (c), 36 tiles (d) and 72 tiles (e)

From the presented results, we can find the stochastic tiling with 16 ω-tiles has already created quite good visual result. Tiling with more ω-tiles such as 36 tiles does not show great difference, as we only texture an 8×8 area from the tile set. However when we select the tile sets of 32 or 72 tiles, as it texture the plane with each tiling step at least two optional ω-tiles, the results seem more natural.

5.2.5 Multiple Source Images

As our tile generation approach searches a new patch $\mathcal{O}$ to replace the central information of each intermediate tile $\mathcal{I}$ from the input sample $\mathcal{S}$, we can create different image patterns by using multiple source images to replace some of the intermediate tile information. A valuable and interesting experiment is to create ω-tiles with different levels of density or different shapes of primitives. In this way, the generated image plane will seem more natural or of more diverse patterns. One interesting example is shown in the following Figure 5.13.

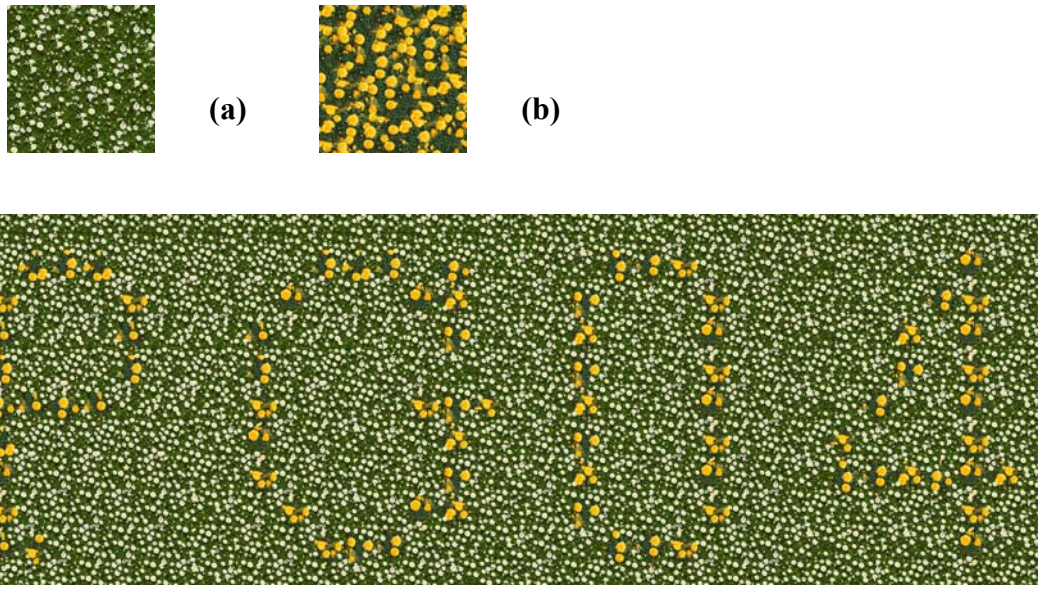


Figure 5.13 Image plane created by multiple sources

In this image, the background of the plane is textured from the image source of while flowers. Then in the appropriate positions, the tiles will use image patch from the yellow flowers to create the four characters of “PG04”. We can find the background plane is a naturally tiling with the white flower sample and the yellow flowers are elaborately decorated in the while flower plane.

Although our tiling scheme is effective to synthesize images, it is also not a general approach that can synthesize all kinds of texture patterns. In the following section, we discuss some patterns that are not suitable to be tiled with our method.

5.3 Limitations and Solutions

There are two classes of images from which we cannot obtain satisfactory synthesized results. The first one is those samples that have quite fine texture primitives so that the whole image seems very smooth, but do not have the same distribution of intensity, such as the brown sugar in Figure 5.14 (a). The image has quite fine small grains of sugar, while intensity of the image gradually changes from bright to dark. Therefore in the synthesized result, the intensity gap across the image patches is visible. With the constraint of the intensity distribution, the tiling algorithm may not be a good solution, while the pixel based method would be a better choice as it can create each pixel prudentially. At the same time, for the input textures with a mess of relatively large tubes of 3 distinct bright colors -- cutting curve in such a case does not perform well either and the seams will be visible. The corresponding Wang tiles for this input have the same problem.

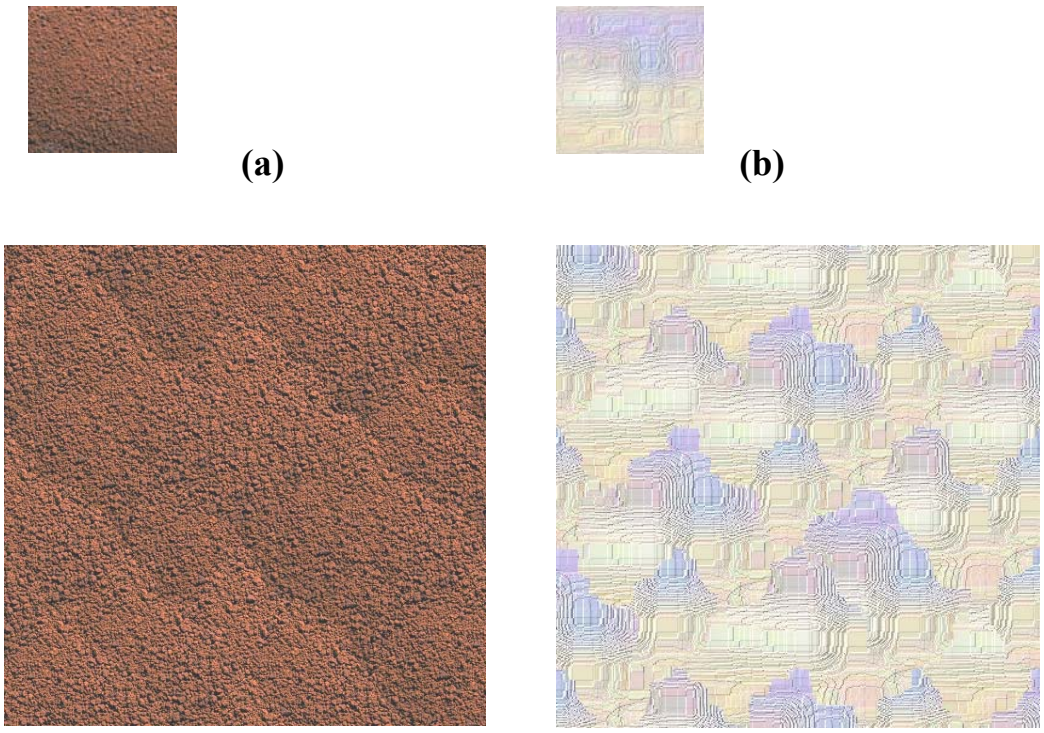


Figure 5.14 Unsatisfactory examples

The other class of sample is those that have inhomogeneous patterns, such as the input sample of Figure 5.14 (b). Our algorithm first randomly extracts four image patches. Therefore, the image patches cannot be smoothly combined. Even after eliminating the line seams, they may have quite different properties, such as colors, gradients. Pixel based method determines each pixel value from their neighborhood. While for this class of samples, the pixel has no direct relationship with its neighborhood and thus the mapping of pixel value between them is hard to be determined. Therefore, pixel based synthesis method cannot be applied to this class of samples either. For such texture sample, we may segment it into several homogeneous components first and then extract image patches of $\mathcal{F}$ within each domain of distinct texture pattern. Such an approach will create the ω-tiles with multiple patterns from different domains of the sample

image. Through stochastically placing these ω-tiles of multiple patterns, we could preserve the color distribution of the sample image.

In this chapter, we present comprehensive synthesized results created by our ω-tile approach. The efficiency of our technique has been verified by the examples, which have the nature of stochastic, semi-structured and structured patterns. We also compare the results created by Wang tile approach and our approach, and the results created by the sets of different size. At the same time, we introduce a method to create images using multiple sources. We discuss two unsuccessful cases with our approach and also propose proper solutions to them. In the next chapter, we summarize our thesis and discuss the improvement for our future work.

Chapter 6 Conclusion

In this chapter, characteristics of our tiling algorithm and its advantages are first summarized in Section 6.1. Then Section 6.2 discusses some directions for future research.

6.1 Algorithm Summary

Plenty of methods have been proposed to synthesize textures. However these algorithms often lay particular stress either on synthesis speed or on visual effect and hardly achieve satisfactory results on both aspects. This paper presents a novel technique to empower user to generate a small set of ω-tiles to real time synthesize a large texture. It proposes the use of cutting curve to be found in a large area to generate a seamless tile, and discusses a few schemes to generate a small set of 8, 16, 32, 36, or 72 ω-tiles that can tile any large area without seams across the boundaries of ω-tiles. The approach allows the embedding of many more texture patterns of the input texture into the generated ω-tiles to synthesize textures with the underlying global distribution pattern of the input texture.

We create small tiles in a novel means that guarantees the potential matching tiles to be combined smoothly in structure. Through searching new image patch and applying cutting curve, our approach not only successfully eliminates the visible seams in each

tile but also makes the corner contents under control. In addition, we adapt the graph cut technique to seek image cuts in a great domain to achieve more satisfactory results for visual perception. Furthermore, the tile combination rule investigated in this thesis can make the small ω-tiles stochastically placed in the output image, which can to a great extent preserve the natural pattern of the sample image. The experimental results presented in this thesis have also verified the efficiency of our technique. Generally, it can be concluded that this tiling approach has the following two special advantages over other algorithms, regarding to the synthesis criteria of the speed of synthesis process and the visual quality of synthesis result: applicable to real time applications where image synthesis are utilized when a small set of ω-tiles are generated in the preprocessing stage and able to create natural visual effect from the global perspective.

6.2 Future Work

There are some limitations and potential improvements for our future work.

- **Weighting Function for Graph Edge**

During seeking current cutting curve, we divide the intensity gap of pixels in adjacent edges by their gradients in two image patches. Therefore, we have taken the structure of primitives in the texture into account in defining the weights for the edges. However, we need to analyze more specifically the structure of patterns in the texture. For example, intuitively, the penalty value of such weights should be accumulated corresponding to the boundary length of the objects in the texture and finally determined by the size of objects.

One possible future work is to investigate the possibility of incorporating pattern detection in searching good cutting curves.

- **Tracks for Cutting Curve**

Currently, each ω-tile is created by superimposing a selected candidate patch $\mathcal{O}$ onto the intermediate tile $\mathcal{I}$ to compute the cutting curve. At the same time, to minimize the prominence of seams across and in the tiles, our approach needs to balance the demands of keeping the contents around the four sides of $\mathcal{I}$ and removing the two line seams in $\mathcal{I}$. Thus the cutting curve is enforced to pass through the middle points of the four sides of $\mathcal{I}$. Therefore from a global view some good choices for the track of cutting curve may be lost. A good improvement for our approach is to free such a restriction. That is, we may need to modify the current combinations of the patches in $\mathcal{F}$ or shape of the tile $\mathcal{I}$ to modify the source and sink area in the overlapping graph of each tile.

- **Properties for Global Patterns**

To realize the objective of making the generated image globally smooth, we currently only consider avoiding periodic effect. There are some other factors such as scale and orientation to be explored. While, by the constraint of the approach of creating tiles, our algorithm is not very effective to deal with structured textures. One improvement can be interactively specifying the image patches of our interest when extracting the

patches to create intermediate tiles $\mathcal{I}$. However, it is not necessary for us to apply such procedure to textures of other patterns.

References

[Ashikhmin01] M. Ashikhmin. Synthesizing natural texture. Proceedings of ACM Symposium on Interactive 3D Graphics, pp. 217-226, Mar. 2001.

[Bourke] P. Bourke. Texture Library. http://astronomy.swin.edu.au/~pbourke/texture/.

[Bourke03] R. Bourke. Image quilting, wang tiling and texture transferring. http:// www.mle.ie/rob/wang/.

[Chellappa85] R. Chellappa, S. Chatterjee, and R. Bagdazian. Texture synthesis and compression using Gaussian-Markov Random Field Models. IEEE Transactions on Systems, Man, and Cybernetics, SMC-15, pp. 298-303, 1985.

[Cohen03] M. F. Cohen, J. Shade, S. Hiller and O. Deussen. Wang tiles for image and texture generation. Proceedings of SIGGRAPH 03, pp. 287-302, Jul. 2003.

[Cross83] G. R. Cross and A. K. Jain. Markov random field texture models. IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-5, No. 1, pp. 25-39, Jan. 1983.

[de.Bonet97] J. S. de.Bonet. Multiresolution sampling procedure for analysis and synthesis of texture image. Proceedings of SIGGRAPH 97, pp. 361-368, Aug. 1997.

[Ebert94] D. Ebert, K. Musgrave, D. Peachey, K. Perlin and Worley. Texture and modeling: a procedural approach. Academic Press, ISBN 0-12-228760-6, 1994.

[Efros99] A. A. Efros and T. K. Leung. Texture synthesis by non-parametric sampling. International Conference on Computer Vision, Vol. 2, pp. 1033-1038, Sept. 1999.

[Efros01] A. A. Efros and W. T. Freeman. Image quilting for texture synthesis and transfer. Proceedings of SIGGRAPH 01, pp. 341-346, Aug. 2001.

[Fahle94] M. Fahle. Human pattern recognition: parallel processing and perceptual learning. Perception, Vol. 23, pp. 411-427, 1994.

[Heeger95] D. J. Heeger and J. R. Bergen. Pyramid-based texture analysis/synthesis. Proceedings of SIGGRAPH 95, pp. 229-238, Sept. 1995.

[Kwatra03] V. Kwatra, A. Schodl, I. Essa, G. Turk and A. Bobick. Graphcut texture: image and video synthesis using graph cuts. Proceedings of SIGGRAPH 03, pp. 277-286, Jul. 2003.

[Lefebvre03] S. Lefebvre and F. Neyret. Pattern based procedural texture. Proceedings of Symposium on Interactive 3D Graphics, pp. 203-212, 2003.

[Levina02] E. Levina. Statistical issue in texture analysis. PhD thesis, University of California, Berkeley, USA, 2002.

[Lewis89] J. P. Lewis. Algorithms for solid noise synthesis. Proceedings of SIGGRAPH 89, Vol. 23, pp. 263-270, Jul. 1989.

[Liang01] L. Liang C. Liu, Y. Q. Xu, B. Guo and H.Y. Shum. Real-time texture synthesis by patch-based sampling. ACM Transactions on Graphics, Vol. 20, No. 3, pp. 127-150, Jul. 2001.

[MIT] MIT Media Lab. Vision Texture. http://www-white.media.mit.edu/vismod/-imagery/VisionTexture /Vistex.html.

[Neubeck03] A. Neubeck, A. Zalesny and L. v. Gool. Cut-primed smart copying, Proceedings of Texture 2003, pp. 71-76, 2003.

[Neyret99] F. Neyret and M. Cani. Pattern-based texturing revisited. Proceedings of SIGGRAPH 99, pp. 235-242, Aug. 1999.

[Paget99] R. D. Paget. Nonparametric Markov random field models for natural texture images. PhD thesis, University of Queensland, Australia, Dec. 1999.

[Parada01] P. Parada and J. Ruiz-del-Solar. Texture synthesis using image pyramids and self-organizing maps. The 11th International Conference on Image Analysis and Processing – ICIAP 2001, pp. 244-249, Sept. Italy.

[Peachey85] D. R. Peachey. Solid texturing of complex surfaces. Proceedings of SIGGRAPH 85, pp. 279-286, Jul. 1985.

[Perlin85] K. Perlin. An image synthesizer. Proceedings of SIGGRAPH 85, pp. 287-296, Jul. 1985.

[Portilla99] J. Portilla and E. Simoncelli. Texture modeling and synthesis using joint statistics of complex wavelet coefficients. Proceedings of IEEE Workshop on Statistical and Computational Theories of Vision, 1999.

[Sedgewick01] R. Sedgewick. Algorithm in C part 5: Graph algorithms. Addison-Wesley, Reading, Massachusetts, 2001.

[Shevelev03] I. A. Shevelev, V. M. Kamenkovich and G. A. Sharaev. The role of lines and corners of geometric figures in recognition performance. Acta Neurobiol Exp, Vol. 63, No. 4, pp. 361-368, 2003.

[Soatto01] S. Soatto, G. Doretto and Y. Wu. Dynamic texture. Proceedings of IEEE International Conference on Computer Vision 2001, Vol. 2, pp. 493-446, 2001.

[Soler02] C. Soler, M. P. Cani and A. Angelidis. Hierarchical pattern mapping. ACM Transactions on Graphics, Vol. 21, No. 3, pp. 673-680, Jul. 2002.

[Stam97] J. Stam. Aperiodic texture mapping. Tech. Rep. R046, European Research Consortium for Informatics and Mathematics (ERCIM), 1997.

[Tuceryan98] M. Tuceryan and A. K. Jain. Texture analysis. The Handbook of Pattern Recognition and Computer Vision (2), pp. 207-248, World scientific publishing Co. 1998.

[Turk91] G. Turk. Generating textures on arbitrary surfaces using reaction-diffusion. Proceedings of SIGGRAPH 91, pp. 289-298, Jul. 1991.

[Upstill89] S. Upstill. The renderman companion. Addison Westley, 1989.

[Walter01] M. Walter, A. Fournier and D. Menevaus. Integrating shapes and pattern in mammalian models. Proceedings of SIGGRAPH 01, pp. 317-326, Aug. 2001.

[Wang65] H. Wang. Games, logic and computer. Scientific American, pp. 98-106, Nov. 1965.

[Wang02] Y. Wang and S. Zhu. A generative method for textured motion: analysis and synthesis. European Conference on Computer Vision-1, pp. 583-598, 2002.

[Wei00] L. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. Proceedings of SIGGRAPH 00, pp. 479-488, Jul. 2000.

[Wei01] L. Wei and M. Levoy. Texture synthesis over arbitrary manifolds surfaces. Proceedings of SIGGRAPH 01, pp. 355-360, Aug. 2001.

[Wen04] C. H. Wen, X. Y. Zhang, and T. S. Tan. Generating an ω-tile set for texture synthesis. http://www.comp.nus.edu.sg/~tants/w-tile/, May 2004.

[Witkin91] A. Witkin and M. Kass. Reaction-diffusion textures. Proceedings of SIGGRAPH 91, pp. 299-308, Jul. 1991.

[Worley96] S. P. Worley. A cellar texturing basis function. Proceedings of SIGGRAPH 96, pp. 291-294, Aug. 1996.

[Xu00] Y. Y. Xu, B. Guo, and H. Shum. Chaos mosaic: fast and memory efficient texture synthesis. Technical Report, MSR-TR-2000-32, Microsoft Research, Apr. 2000.

[Zhu97] S. C. Zhu, Y. Wu and D.B.Mumford. Minmax entropy principle and its application to texture modeling. Neural Computing, Vol. 9, No. 8, pp. 1627-1660, 1997.

[Zhu00] S. C. Zhu, X. Liu, and Y. N. Wu. Exploring texture ensembles by efficient Markov Chain Monte Carlo-Toward a “trichromacy” theory of texture. Pattern Analysis and Machine Intelligence, Vol. 22, No. 6, pp. 554-569, 2000.