University of Alberta

Texture Analysis/Synthesis Using Gray Level Aura Matrices

By

Xuejie Qin

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Department of Computing Science

Edmonton, Alberta, Canada
Fall 2006

Your file Votre référence
ISBN: 978-0-494-23096-1
Our file Notre référence
ISBN: 978-0-494-23096-1

Abstract

Texture modeling plays an important role in computer graphics, vision and image processing. Although various techniques have been developed for the study of texture analysis and synthesis, the mathematical definition of texture is still unclear. Due to the vague definition of texture, each technique has its own advantages and disadvantages, and thus fails to model certain types of textures.

This thesis presents a new unified mathematical framework for modeling textures using BGLAMs (Basic Gray Level Aura Matrices). The new framework will provide important understanding in texture modeling in both computer vision and computer graphics. It is proved that BGLAMs form a basis of GLAMs (Gray Level Aura Matrices), and that two images are identical if and only if their corresponding BGLAMs are the same. It is also proved that the number of different BGLAMs of a given image is no more than the number of pixels in the image. This work clarifies the relationship between BGLAMs, GLAMs, SGLAMs (Symmetric GLAMs), and GLCMs (Gray Level Cooccurrence Matrices), and demonstrates that BGLAMs outperform both SGLAMs and GLCMs in texture modeling.

Based on the theory, new techniques have developed new techniques for 2D and 3D texture synthesis, and a new method for classifying texture images using BGLAMs. The experimental results show that our new techniques can successfully apply to a wide range of textures and the results are either better or comparable to existing techniques.

Acknowledgments

I wish to thank Dr. Herb Yang sincerely for mentoring and supporting me throughout my graduate education. Herb has been a wonderful supervisor for me. His devotion, encouragement, wisdom and guidance have been of great help in the completion of this research project. The high standards Herb holds in both scholarship and research are an invaluable inspiration that certainly shaped my academic pursuits and will continually encourage me all my life. I would like to thank the members of my supervisory and candidacy examination committees, Dr. Walter Bischof (Chair), Dr. Mario A. Nascimento, Dr. Joerg Sander, Dr. Arturo Sanchez-Azofeifa, and Dr. Xiaobo Li, for their support, helpful insights and useful suggestions for the progression of my research. Special thanks go to Dr. David A. Clausi, for being the external examiner of my thesis defense, and Dr. Eric Donovan and Dr. Mikko Syrjäsuo of the Institute for Space Research at the University of Calgary for collecting and interpreting the ASI (All Sky Imager) data for the experiments on texture image classification.

I would like to thank the present and past members of the RAMA (Rendering, Analysis, Modeling and Animation) Group, Hai Mao, Daniel Neilson, Cheng Lei, Danielle Sauer, Jason Selzer, Nathan Funk, Tong Guan, Jiayuan Zhu, Yi Xu and Dr. Minglun Gong, for their tremendous help, warm friendship and great fun. I enjoyed working in this extraordinary laboratory. I would also like to thank the supporting and academic staff of the Department of Computing Science at the University of Alberta for supplying a first-class research and education environment.

Last, but not least, I would like to thank my wife, Jing, for her tireless encouragement, support and belief that I could count on through these years in Canada. I would like to extend my gratitude to my parents Guofu Qin and Suming Mi and my mother-in-law Suhua Huang for their unconditional love and support throughout my life and in pursuit of my Ph.D. degree.

To Jing, my wife, for all your love and support
To Gary, my son, for all your love and support

Table of Contents

List of Tables

List of Figures

List of Abbreviations, Symbols and Nomenclature

Λ_s	The state space of site s
$\boldsymbol{\Omega}$	The configuration space of a random field $\boldsymbol{X}$
$\vartheta_B(A)$	The aura of A with respect to B
$\|A\|$	The total number of elements in set A
2D	2 Dimensional
3D	3 Dimensional
$\boldsymbol{A}$, $\boldsymbol{A}(\Im)$	The aura matrix of $\Im$
ACGMRF	Anisotropic Circular Gaussian Markov Random Field
ASI	All Sky Imager
BGLAM	Basic Gray Level Aura Matrix
BTF	Bidirectional Texture Functions
CANOPUS	Canadian Auroral Network for the Open Program Unified Study
$d(...)$	The distance function
E	The neighborhood structuring element
FRAME	Filters, Random Fields and Maximum Entropy
G	The total number of gray levels for a pixel in the image
GLAM	Gray Level Aura Matrix
GLCM	Gray Level Cooccurrence Matrix
GPU	Graphics Processing Unit
GRF	Gibbs Random Field
GUI	Graphical User Interface
ICM	Iterative Conditional Modes
$K(.,.)$	The kernel function
kNN	k-Nearest-Neighbors
LBG	Linde, Buzo and Gray
LCPDF	Local Conditional Probability Density Function

$m(A, B)$	The aura measure of A with respect to B.
MAP	Maximum a Posterior
MPM	Maximum Posterior Marginal
MRA	Multi-Resolution Analysis
MRF	Markov Random Field
MRI	Magnetic Resonance Imaging
$\boldsymbol{N}$	The neighborhood system of an image
N_s	The neighborhood of site s
$\overline{N}_s$	The complement neighborhood of site s
N_s^d	The neighborhood of radius d at site s
PMRF	Parametric Markov Random Field
$P(x, y, z)$	A point in the 3D space with xyz-coordinates
$p(x_s)$	The probability of site s having pixel value x_s
R, G, B	The R, G, B channels of a color image
RBF	Radial Basis Function
$\Re^n$	The n-dimensional space
S	A finite rectangular lattice
SAR	Synthetic Aperture Radar
SGLAM	Symmetric Gray Level Aura Matrix
S_i	The set of all sites in S with intensity value i
SVD	Singular Value Decomposition
SVM	Support Vector Machine
$\boldsymbol{T}$	The color-space transformation based on SVD
TSVQ	Tree-Structured Vector Quantization
$\boldsymbol{V}$	A solid texture, or a volume in the 3D space
$\boldsymbol{X}$	A random field
X_s	A random variable at site s

Chapter 1

Introduction

1.1 Textures

Textures appear everywhere. The world would not be so beautiful without the meticulous presence of textures. However, what exactly is texture? Although the term appears to be understood by a layman, there is still no formal mathematical definition of textures. In computer graphics, vision and image processing, textures are commonly considered as visual patterns that appear on the surfaces of objects or in images. In the real world, textures can be seen as either microstructures or macrostructures on the surfaces of objects, for example, the fine ripples on the surface of water in a lake or river, the arrangement of bricks on a wall, the field of grass in a meadow, the fluffiness of clouds in the sky, and the varying fur on animals. Figure 1-1 gives an example of wood-like texture on the surface of a teapot.

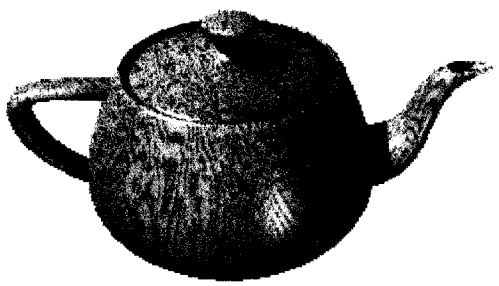


Figure 1-1: An example of wood-like textures on the surface of a teapot.

In image analysis, textures are used as visual cues to differentiate one image region (or one image) from other regions (or other images). Oftentimes, different parts of an image, or different images, are recognized by textures rather than by shapes. Some examples of textures, such as grass, flower, rug, wood, brick, and stone, are given in Figure 1-2.

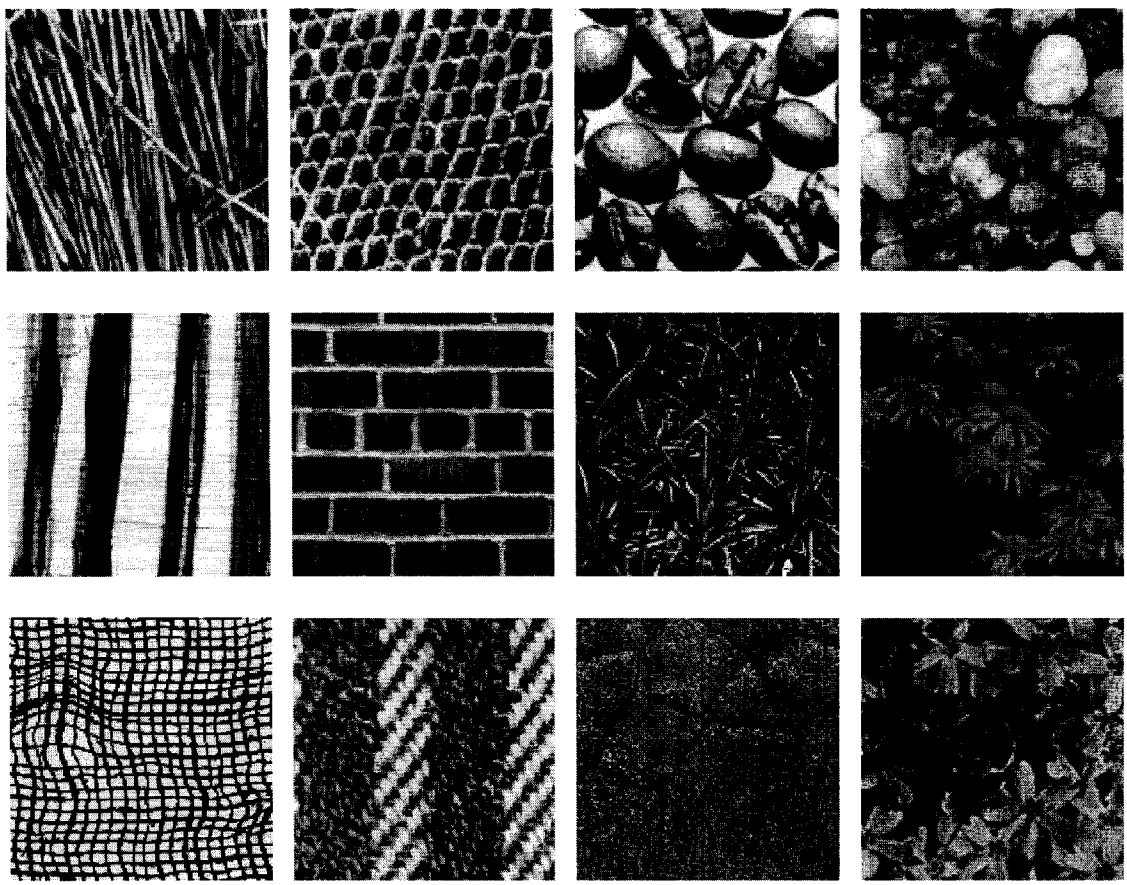


Figure 1-2: Examples of textures in images (image source: the Brodatz and the Vistex textures [124]).

Textures can be very easily recognized in images by a human observer but it is very difficult to quantify their differences precisely [8, 77, 178]. The difficulty is

demonstrated by the large number of ambiguous definitions of textures (see [178]) proposed by researchers during the last several decades, many of which lead to serious problems in computational complexity and in imprecision. In computer vision, it is desirable to define textures with mathematical precision.

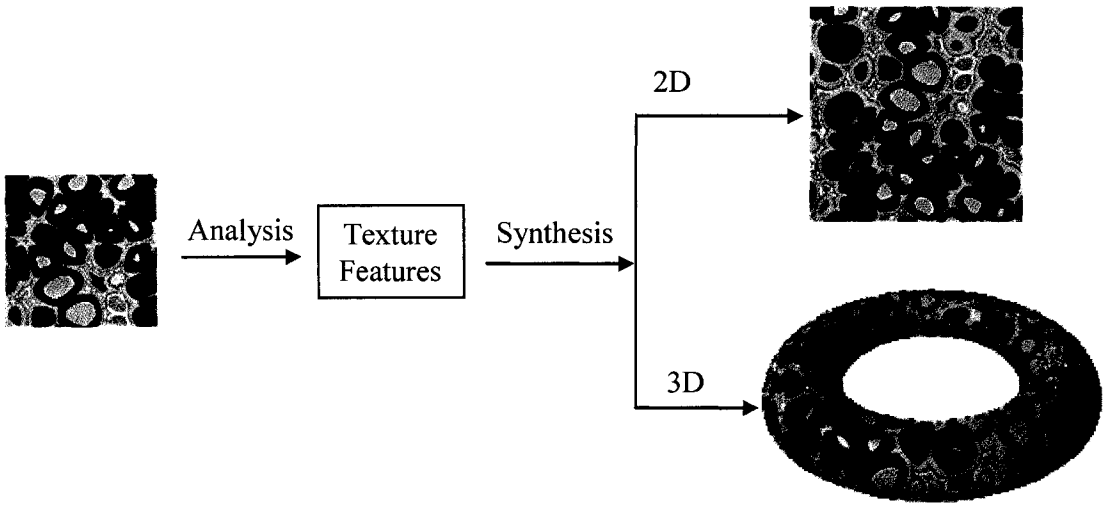


Figure 1-3: An example of texture analysis and synthesis.

1.2 Texture Analysis and Synthesis

Texture analysis and synthesis, also called texture modeling in this thesis, can be viewed as a two-phase process: 1) characterization of textures and 2) regeneration of textures. In the first phase, textures are analyzed using various techniques and important information of textures, called texture features, are characterized using either mathematical or non-mathematical descriptions [8, 77, 178]. Based on the information obtained in the analysis phase, in the second phase, synthetic 2D or 3D textures are generated using various sampling techniques such as stochastic relaxation [63], pixel-based sampling [49, 189], or patch-based sampling [50, 101, 106, 186]. An example of texture analysis and synthesis is given in Figure 1-3.

Since the pioneering work done by Julesz [93], texture analysis/synthesis has been an active research topic in computer graphics, vision and image processing. In computer vision, researchers have developed many techniques for analyzing textures; while in computer graphics, researchers are more interested in developing techniques to synthesize textures for generating appealing imagery. Texture analysis and synthesis have important applications in document processing, automated inspection, bioinformatics, data compression, animated movies, and computer games [8, 19, 47, 178]. For example, a texture-synthesis technique can be used in computer games for generating interesting textures, e.g. skin, onto animated figures.

1.3 Motivation

Despite many techniques [3, 6, 32, 35, 49, 81, 101, 129, 190] have been proposed for texture analysis and synthesis, texture modelling is far from being understood. In fact, each existing technique has its own advantages and disadvantages, and thus fails to correctly model certain types of textures. The present thesis work is motivated by the following challenging problems in texture modeling:

- Given a texture sample, what can be used to represent the sample with the necessary and sufficient information? In computer vision, it is desirable to define textures with mathematical precision. However, this problem is challenging and has been studied by researchers in the vision area for decades without much success. Consequently, the advancement of texture synthesis techniques in computer graphics is limited because of the lack of understanding in the analysis of textures. In existing approaches, textures are

often represented by using some characteristics of input examples, which may not represent the input texture appropriately. For instance, in feature-matching approaches [6, 35, 81, 143], a set of filter responses at multiple scales and orientations are used to characterize an example texture. However, mathematically it requires an infinite number of filters, each of which is as large as the given texture image, to model a given texture with the necessary and sufficient information. Hence, a set of predefined filters are used in filter-based techniques for modelling textures. In general, it is not an easy task to automatically select filters for different textures. The objective of this thesis work is to develop a mathematical framework for modelling textures without using filters and demonstrate that texture analysis and synthesis can be effectively carried out using BGLAMs (basic gray level aura matrices).

- There is a lack of good unified frameworks that work well for both analysis and synthesis. In exiting texture modeling approaches, a good analysis technique may not work well for synthesis [178]; while a good synthesis technique that generates impressive results may not be able to do analysis at all [178].
- It is difficult, if not impossible, to perform 3D texture analysis and synthesis. Techniques for 2D texture analysis and synthesis [3, 6, 32, 35, 49, 81, 101, 129, 190] cannot be easily extended and applied to 3D texturing. In general, existing 3D-texture techniques (e.g. [20, 81, 86, 125, 179, 191, 204]) work for only a limited range of textures.

- How to evaluate the results quantitatively? Due to the imprecise representation of textures in existing synthesis techniques, none of them is able to evaluate the results quantitatively. Visual inspection is the only way to evaluate the results.

The goal of this thesis work is to develop a new vigorous mathematical model to characterize textures with sufficient and necessary information. Under the unified mathematical framework, we present new techniques for 2D and 3D texture analysis and synthesis. Within the same framework, we develop a quantitative method for evaluating texture synthesis results, which can be used to automate the conventional visual inspection process for determining whether or not the output texture is a successful synthesis of the input. For applications, we apply our new framework in texture image classification on the Brodatz database, the Vistex database, and the ASI (All-Sky Imager) database.

Our new framework for texture analysis and synthesis will provide important understanding in texture modeling in both computer vision and computer graphics. Most importantly, it is the only approach that uses the same framework for both analysis and synthesis. In other words, getting better analysis results will help us synthesize more realistic textures and vice versa.

1.4 The Thesis Work

Figure 1-4 gives an overview of the thesis work. In the work, a new mathematical framework based on BGLAMs (Basic Gray Level Aura Matrices) for texture modeling is

presented. The mathematical theory of BGLAMs is formulated and proved using the concepts of aura sets, aura measures, and aura matrices introduced by Elfadel and Picard [51]. We prove in Chapter 3 that BGLAMs form a basis of GLAMs (Gray Level Aura Matrices) – a powerful tool for texture modeling, and that two images are identical if and only if their corresponding BGLAMs are the same. We also prove that the number of different BGLAMs of a given image is no more than the number of pixels in the image.

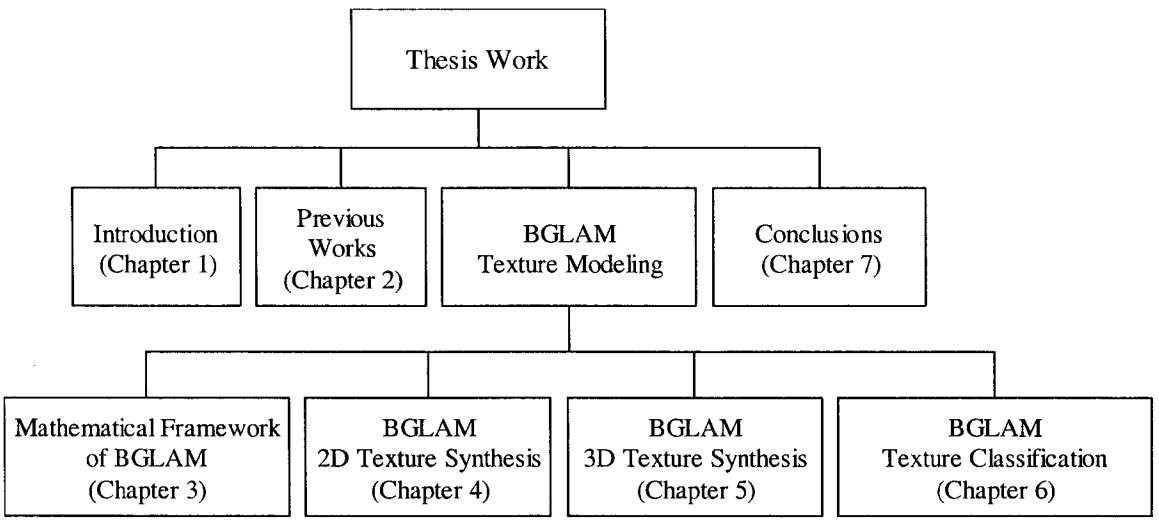


Figure 1-4: An overview of the thesis work.

Based on the new theory, in Chapter 4, we develop a new 2D texture-synthesis method, which generates synthetic textures by sampling the BGLAMs of input textures. We demonstrate that in practice a small set of BGLAMs (e.g. 48 BGLAMs for an image of size 64×64) calculated from the input texture samples can be used to generate textures of arbitrary sizes. However, the actual number of BGLAMs (usually $O(n)$ for an input image of size $n\times n$) used for generating textures is much smaller than the number of pixels in the image and thus the computational cost is significantly reduced.

Experiments have shown that a broad range of 2D textures can be successfully synthesized using BGLAMs and the synthesis results are comparable to existing techniques (e.g. [101, 106, 190]).

In addition to 2D texture analysis and synthesis, in Chapter 5, we demonstrate that BGLAMs can be used to generate solid textures, also called 3D textures. A solid texture is considered as a block of colored points in 3D space to represent a real-world material, for example, a wood trunk. Given one or more input texture samples, our method first creates the BGLAM representations of the input samples and then generates a solid texture by sampling the BGLAMs constrained in multiple view directions. Once the solid texture is available, any given 3D object can be textured by carving the object out of the volumetric data. Our method is fully automatic, requires no user interaction in the process, and can generate faithful results over a wide range of textures. The experimental results also show that the new method for 3D textures outperforms previous approaches (e.g. [86, 188]).

We also define a new distance function based on BGLAMs for measuring the similarity between texture images. The distance function satisfies the metric properties [160] of non-negativity, symmetry, and triangle inequality. Furthermore, one unique property of the new distance function, which is proved in the thesis, is that it is one-to-one. Namely, a zero value of the distance measure will guarantee that the two images are identical. Since the distance function is continuous, the one-to-one property implies that if the distance of image Y from image X gradually changes (i.e. converges) to zero, image Y will gradually get close (i.e. converge) to X. A distance measure without the one-to-one property cannot guarantee this.

Based on the well-defined and one-to-one BGLAM distance measure, we present an original quantitative method to simulate the visual inspection process of determining whether or not the synthesized texture is a successful synthesis of the input sample. Extensive user studies have shown that if the distance value is below a threshold value (0.1 used for the experiments in the thesis), then the output texture is guaranteed to be a successful synthesis of the input. For existing texture synthesis techniques, human visual inspection is the only effective way to evaluate the results.

Presented in Chapter 6 of the thesis is a BGLAM-based method for texture image classification. Given an unseen texture image, our approach classifies it into one of the pre-learned classes. There are two stages in our algorithm: a learning stage and a classification stage. In the first stage, models of texture classes are learned from the BGLAMs of training examples using the Support Vector Machine (SVM), and in the second stage, a given texture image is classified into one of the pre-learned classes, to which the input image is most similar. We compare our approach experimentally with existing approaches by performing texture classification over the Brodatz textures, the Vistex textures, and the All Sky Image (ASI) textures. For both the Brodatz database and the Vistex database, the experimental results show that the proposed new approach performs better than existing approaches with an average success classification rate of 99% vs 87% using other approaches. For the ASI database, the results have shown that our approach significantly outperforms existing approaches with an average successful rate of 97% vs 66%.

1.5 Summary of Contributions

The main contributions of this thesis work are as follows:

1. The mathematical theory for BGLAMs. It is proved that BGLAMs form the basis of GLAMs, and that two images of the same size are identical if and only if their corresponding independent BGLAMs are the same. Therefore, an image can be uniquely represented by its BGLAMs (Chapter 3).
2. A new distance function based on BGLAMs for measuring texture similarity. In addition to the metric properties, the BGLAM distance function is one-to-one. This one-to-one property implies that a zero value of the distance measure between two images guarantees that they are identical. With this one-to-one property, we demonstrate that the new distance function can be used for evaluating texture synthesis results quantitatively. We have shown that if the distance value is below a threshold value, then the output texture is guaranteed to be a successful synthesis of the input. For existing texture synthesis techniques, human visual inspection is the only effective way to evaluate the synthesis results (in Chapter 3 and 4).
3. A new BGLAM-based method for 2D texture analysis and synthesis. For a given input texture sample, synthetic 2D textures can be generated by sampling a small set of BGLAMs (e.g. 64 BGLAMs for an image of size 64×64) that are calculated from the input (Chapter 4).
4. An original BGLAM-based algorithm for synthesizing solid (i.e. 3D) textures from one or more input samples. Our method generates solid textures by

sampling the BGLAMs of the input samples constrained in multiple view directions (Chapter 5).

5. A BGLAM-based method for texture image classification. We test our method by performing image classification on the Brodatz database and the Vistex database. For real application, we have successfully applied our method to classifying ASI (All Sky Imager) texture images (Chapter 6).

1.6 Outline of the Thesis

This thesis is organized as shown in Figure 1-4. Following the Introduction, Chapter 2 reviews the previous works in the field of texture modeling. Chapter 3 presents the BGLAM theory and its mathematical proofs. Chapter 4 describes 2D texture synthesis and the evaluation of synthesis results using BGLAMs; Chapter 5 describes 3D texture synthesis using BGLAMs and its evaluation. In Chapter 6, we present the BGLAM-based method for texture classification. Conclusions are given in Chapter 7.

Chapter 2

Previous Work

In this chapter, we review various techniques in texture modeling. For the ease of discussion, we divide existing techniques into two categories: 2D texture modeling and 3D texture modeling. In 2D texture modeling, we describe the MRF (Markov Random Field) texture models [17, 32, 38, 39, 63, 64], the pixel-based sampling approach [49, 81, 190], the patch-based sampling approach [50, 101, 106], the feature matching approach [6, 35, 143, 167], the cooccurrence matrix approach [21, 24, 34, 67, 75, 112, 160, 209], and structural texture modeling [74, 75, 92, 104, 110, 206]. In 3D texture modeling, we discuss techniques in texture mapping [10, 79, 80], procedural texturing [28, 47, 132, 134], image-based surface texturing [179, 191, 201, 204], and image-based solid texturing [42, 43, 81, 86, 103, 188]. Figure 2-1 gives an overview of the taxonomy of the existing techniques in texture modeling that are discussed in the rest of this chapter.

2.1 2D Texture Modeling

In 2D texture analysis and synthesis, textures are first analyzed using various techniques and important information of textures, called texture features, are characterized [77, 178]. Based on the information obtained in the analysis phase, synthetic 2D textures are then generated using various sampling techniques such as stochastic relaxation [63], nearest neighborhood searching [49, 189], etc. As shown in Figure 2-1, techniques in 2D texture analysis and synthesis are classified into six

categories: the MRF (Markov Random Field) texture models, the pixel-based sampling approach, the patch-based sampling approach, the featured-based matching approach, the cooccurrence matrix approach, and structural texture modeling, which are described in the following subsections.

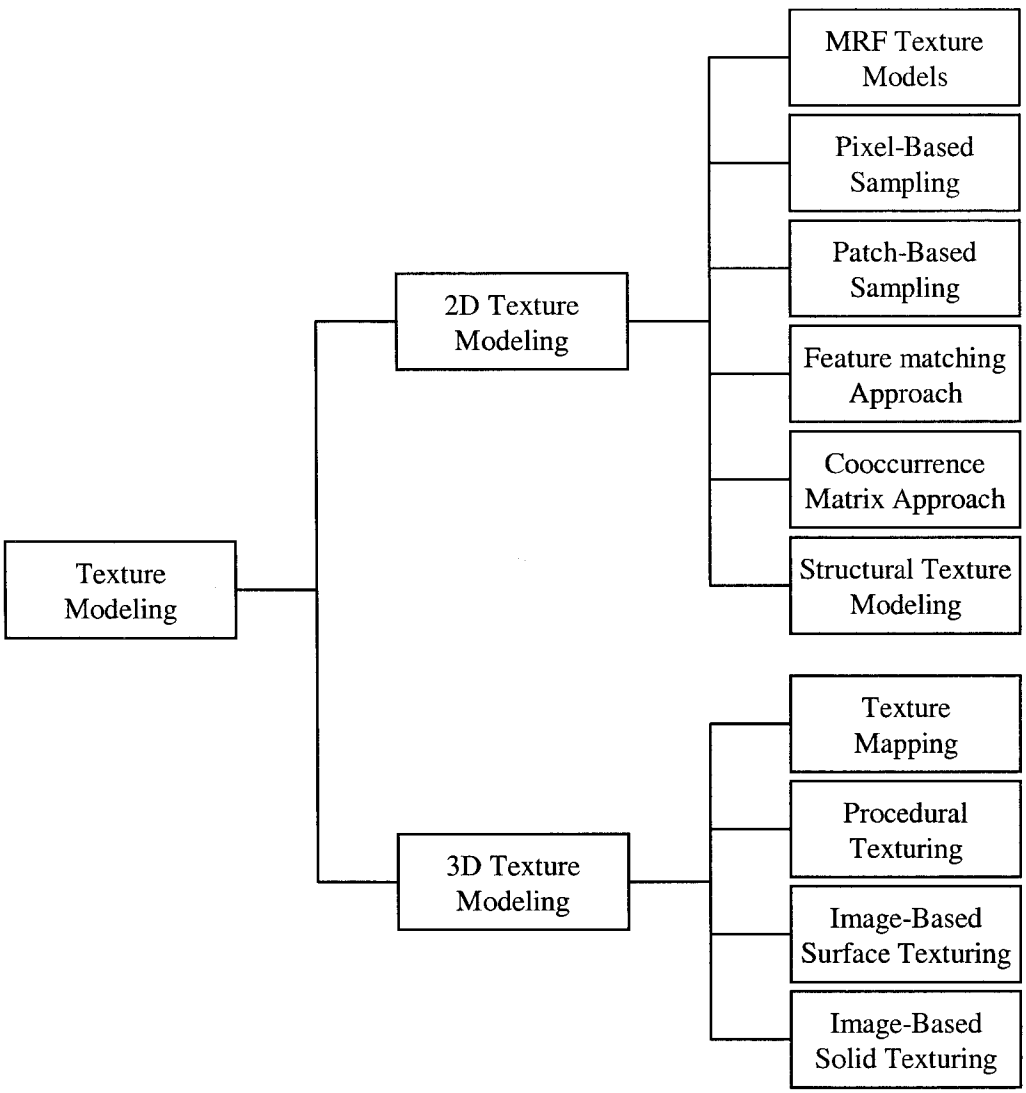


Figure 2-1: Taxonomy of the various existing techniques in texture modeling.

2.1.1 MRF Texture Models

The MRF models are important statistical techniques in texture modeling. The underlying theory of MRF texture models is that the information at a pixel location depends on the information of its neighboring pixels. Earlier research study on MRF includes the Ising models [98, 136, 171], the auto models [17, 25, 32], the GRF (Gibbs Random Field) models [40, 62, 63], etc. Recent work in this area includes the MRF model for color texture image segmentation [37, 130], the nonparametric multiscale MRF model [127, 129], the strong MRF model [126], and the advanced Gaussian MRF model for anisotropic textures [38, 39]. There are also variations of MRF texture models proposed recently [49, 141, 190, 208].

In the MRF models, an image is represented as a random field $\boldsymbol{X}$ defined on a finite rectangular lattice S. Let X_s be a *random variable* at site s, then the *random field* $\boldsymbol{X}$ on S is the set of all random variables X_s, i.e. $\boldsymbol{X} = \{X_s \mid s \in S\}$. The set of all possible values of X_s, denoted by Λ_s, is called the *state space* of s. In general, a common discrete state space for all s is assumed, i.e. $\Lambda_s = \Lambda = \{0,1,...,255\}$ for all $s \in S$. The *configuration space* on $\boldsymbol{X}$, denoted by $\boldsymbol{\Omega}$, is defined as $\boldsymbol{\Omega} = \prod_{s \in S} \Lambda_s$, and a particular configuration (i.e. an observed sample image) $\boldsymbol{x} = (x_s)_{s \in S}$, denoted by $\boldsymbol{X} = \boldsymbol{x}$, is called a realization of the random field $\boldsymbol{X}$. The *joint probability* on $\boldsymbol{\Omega}$ is denoted by p and the *local conditional probability density function* (*LCPDF*) at site $s \in S$, denoted by $p(x_s)$, is given by $p(x_s) = p(X_s = x_s \mid X_r = x_r, r \neq s)$, which says that the probability of site s having pixel value $x_s \in \Lambda_s$ depends on the pixel values of its neighboring pixels.

The *neighborhood* at site s is given by $N_s^d = \{r \in S \mid 0 < |r - s|^2 \leq d\}$, where d is an integer, which determines the size of the neighborhood. The set of all neighborhoods N_s^d is called a *neighborhood system* on S, which is denoted by $\boldsymbol{N} = \{N_s^d \mid s \in S\}$ (or $\boldsymbol{N} = \{N_s \mid s \in S\}$ if the value of d is clear in the context). The neighborhood system has two important properties: for any $s, t \in S$, (1) $s \notin N_s^d$, and (2) $s \in N_t^d$ if and only if $t \in N_s^d$, where property (1) says that site s is excluded from its neighborhood and property (2) implies that the neighborhood is symmetric. In the next chapter, it is shown that this symmetric condition can be relaxed to include asymmetric neighborhood systems for modeling anisotropic textures.

Mathematically, an MRF, which is defined as a random field $\boldsymbol{X} = \{X_s \mid s \in S\}$ with a joint probability function p defined on it, has the following three properties:

$$\begin{aligned} &(i)\ \ p(\mathbf{x}) = p(\mathbf{X} = \mathbf{x}) > 0, \forall \mathbf{x} \in \Omega \\ &(ii)\ \ \textstyle\sum_{x \in \Omega} p(\mathbf{x}) = 1 \\ &(iii)\ p(x_s) = p(X_s = x_s \mid X_r = x_r, r \in N_s^d), \forall \mathbf{x} \in \Omega, \forall s \in S \end{aligned} \quad . \qquad (2.1)$$

Properties (*i*) and (*ii*) ensure that p is a probability distribution with a positive value for any configuration $\boldsymbol{x} \in \Omega$. Property (*iii*) is defined on a neighborhood system, which ensures that the random field is an MRF, i.e. the LCPDF of a given site s in a given image $\boldsymbol{x}$ can be calculated using the information of its neighboring pixels.

Based on the equivalence between the MRF and Gibbs distribution established by the well-known Hammersley-Clifford Theorem [9]: Given any observed image $\boldsymbol{x}$, the probability $p(\boldsymbol{x})$ in Eq. 2.1 can be expressed in terms of Gibbs energy as follows:

$$p(\boldsymbol{x}) = \frac{1}{Z} \exp\left(-\frac{1}{T} E(\boldsymbol{x})\right), \qquad (2.2)$$

where $E(\boldsymbol{x})$ is the Gibbs energy (simply called energy function), which is model dependent [98, 127], T is the temperature, and Z is the partition function (i.e. the normalizing constant).

[**Parametric MRF Models**] In the parametric MRF (PMRF) models, parameters must be estimated for characterizing a given input texture. Many PMRF texture models have been proposed in the literature (see [98, 127] for a complete survey). The simplest and earliest PMRF models are the auto-models (e.g. the Ising model [136] and the auto-binomial model [32]), in which the energy function E is dependent only on the cliques (see [32, 62, 63, 78] for the definition of cliques) that contain no more than two sites:

$$E(\boldsymbol{x}) = \sum_{s \in S} V_s(x_s) + \sum_{s \in S} \sum_{r \in N_s} V_{sr}(x_s, x_r),$$

where V_s's and V_{sr}'s are the potentials by which model parameters are incorporated. To estimate model parameters, the maximum likelihood estimator [165] can be used. Once model parameters are estimated, stochastic relaxation such as the Metropolis algorithm, the Gibbs sampler, or the ICM (Iterative Conditional Modes) algorithm [63, 127] can be used to generate a synthesized texture from a given texture sample.

The types of textures that the auto-models can represent are severely limited because of the following two problems [129, 208]: 1) the cliques are too small to capture texture general features, and 2) only the first-order and second-order statistics (e.g. mean and covariance) are specified on the cliques. Many textures have local structures much larger than three or four pixels, have high-order statistics in addition to low-order statistics (Julesz [93] hypothesized that third- or higher-order statistics, e.g. skewnesses, kurtosises, are required to model natural textures), and are strongly non-Gaussian in most cases. One possible solution is to increase both the size of the cliques and the order of the

statistics. However, even with a modest neighborhood (e.g. 13 x 13), the number of parameters will be too large for any practical inference [208].

To address the above problems, Zhu et al. propose the FRAME (Filters, Random Fields and Maximum Entropy) model [208], which incorporates filtering theory into the MRF texture modeling. For a given texture X, the FRAME model assumes that there exists a true joint probability density $f(X)$ over the image space, which characterizes the given texture. Therefore, the objective of texture modeling is to make inference about the joint probability density $f(X)$. Texture synthesis in the FRAME model is carried out as follows. Firstly, a set of filters is selected from a predefined filter database to capture the texture features, and these filters are applied to the observed image X to get a set of filtered images. The histograms of the filtered images, which estimate the marginal distributions of the joint probability density $f(X)$, are then extracted, and this step is called feature extraction. Secondly, the maximum entropy principle is used to derive a probability distribution $p(X)$ that has the same marginal distributions as that calculated in the feature extraction step, and the derived probability distribution $p(X)$ is taken as an estimate of the true joint probability $f(X)$. Finally, the Gibbs sampler [63, 127] is employed to sample a synthesized texture from the estimated probability distribution $p(X)$. The algorithm of the FRAME model for texture synthesis is summarized below:

1. Input: a texture image X.
2. Select a set of K filters $F = \{f_i \mid 1 \le i \le K\}$.
3. Compute the histograms of the filtered images (i.e. the marginal distributions) of the input X: $\{H_i(X) \mid 1 \le i \le K\}$, where $H_i(X) = \{h_{ij} \mid 1 \le j \le L\}$ is the histogram

of the filtered image obtained by applying the filter f_i to the input $\boldsymbol{X}$, where L is total number of gray levels (e.g. 256),

4. Let $\lambda = \{\lambda_i \mid 1 \le i \le K\}$, where $\lambda_i = \{\lambda_{ij} \mid 1 \le j \le L\}$, which are called the Lagrange multipliers. Initialize $\lambda_{ij} \leftarrow 0$, $i = 1,2,...,K$ and $j = 1,2,...,L$.
5. Initialize the output $\boldsymbol{Y}$ as a uniform white noise image with L gray levels.
6. Repeat step a-d until $\sum_{i=1}^{K}\sum_{j=1}^{L}(h_{ij}(\boldsymbol{X}) - h_{ij}(\boldsymbol{Y}))^2 < \varepsilon$, where $\varepsilon \in (0,1)$ is a user specified parameter:
   a. Compute the histograms of the filtered images of the output $\boldsymbol{Y}$: $\{H_i(\boldsymbol{Y}) \mid 1 \le i \le K\}$.
   b. Update λ by $\frac{d\lambda_i}{dt} = H_i(\boldsymbol{Y}) - H_i(\boldsymbol{X})$, $1 \le i \le K$.
   c. Calculate the estimated probability distribution using
      $p(\boldsymbol{Y},\lambda) = \frac{1}{Z(\lambda)}\exp\left(-\sum_{i=1}^{K}\lambda_i \bullet H_i(\boldsymbol{Y})\right)$, where the operator $\bullet$ denotes dot product of two vectors, and $Z(\lambda)$ is the normalization constant.
   d. Use Gibbs sampler to update the output $\boldsymbol{Y}$ according to the estimated probability distribution $p(\boldsymbol{Y},\lambda)$ obtained in the previous step.

For a given type of texture, in step 2 of the above algorithm, a set of filters has to be correctly selected to represent the given texture. Zhu et al. give a greedy algorithm for filter selection [208]. In the FRAME model, every filter introduces the same number of parameters regardless of its size. In other words, the number of parameters is independent

of the neighborhood size; while in the auto-models, the number of parameters increases exponentially with respect to the neighborhood size. Therefore, the FRAME model makes the parameter estimation feasible for large size (e.g. 33 x 33) filters. However, because of the Gibbs sampler and the iterative numerical approximation of Lagrange multipliers in Step 6b, the FRAME model is still very slow. The expensive computation cost can be reduced by Monte Carlo Markov Chain methods [207], but the quality of the synthesis results is not guaranteed.

[**Non-Parametric MRF Models**] As suggested by Julesz [93], higher-order statistics are required to model natural textures. However, it is very difficult for the conventional parametric MRF models (e.g. auto-models) to incorporate higher-order statistics because of the number of parameters required. As described before, Zhu et al. [208] have proposed a solution in their FRAME model by incorporating filter responses into the MRF models. An alternative solution is to use non-parametric MRF models [129, 142].

In the non-parametric MRF model proposed by Popat and Picard [142], the high-order statistical information (called the multi-dimensional histogram) of a given texture is captured by kernel estimation and cluster analysis. The histogram data is clustered by the LBG (Linde, Buzo and Gray) algorithm [65, 107] for vector quantization, and each cluster is estimated by a standard multi-dimensional Gaussian density. One limitation of their method is that it can only model up to 14-dimensional histograms and higher dimensional histograms cannot be inferred well.

To overcome the problem of Popat and Picard's model, Paget proposes a noncausal non-parametric multiscale MRF model [129], in which large multi-dimensional histograms (e.g. 81 dimensions) can be used to represent a texture. Instead of modeling a cluster of points in the space of a multi-dimensional histogram by a Gaussian density, he models each point in the space with a standard multi-dimensional Gaussian density using the Parzen density estimation technique [45]. Recently, the above Paget's texture model is extended to the strong MRF model (non-parametric) [126], which can be used not only for texture synthesis but also for texture segmentation. Experimental results have shown that Paget's models can generate good results for a wide range of textures, which includes stochastic to structured textures. However, there are two limitations in Paget's models. The first is that the Gaussian density by Parzen density estimation has smoothing effects to the output textures. The second is that the model is computationally expensive as discussed in Paget's thesis [127].

2.1.2 Pixel-Based Sampling Approach

In all the MRF models discussed before, the joint conditional probability distributions of textures are first estimated from the input samples and synthesized textures are generated by sampling the estimated probability distributions. In general, these models are very slow because of the long iterative sampling and the expensive calculation of probability distributions.

To avoid explicit probability construction and sampling, Efros and Leung [49] propose a pixel-based non-probabilistic sampling technique, in which the estimation of the conditional probability distribution of a pixel on its neighbors is approximated by the

nearest neighborhood search. Given an input texture image, a new texture image is synthesized one pixel at a time. To synthesize a pixel p in the output image, Efros and Leung's algorithm first finds a candidate set C of all the pixels in the sample image whose neighborhoods are similar to that of pixel p, and then randomly selects one pixel from C and copies its color to pixel p.

However, Efros and Leung's algorithm is still slow for any practical application. Wei and Levoy [190] propose a fast texture synthesis algorithm using a fixed-size neighborhood searching and tree-structured vector quantization (TSVQ). Their algorithm can also perform texture synthesis in a multiresolution fashion. Both Efros & Leung's algorithm and Wei & Levoy's algorithm produce good results for a wide range of textures.

2.1.3 Patch-Based Sampling Approach

Texture synthesis performed at the pixel level of images, in general, is slow. For real time applications, the pixel based approaches [49, 81, 208] are impractical unless the accelerations of the algorithms are available (e.g. [3, 190]). A solution to this problem is to use patch based approaches [50, 101, 106]. The basic idea of patch based approaches is to synthesize a texture image by copying small patches from the input texture image. Figure 2-2 gives an example of a simple patch based approach, in which a small patch is randomly chosen from the input and copied into the output. The final output texture image is generated by tiling together the small patches that are randomly chosen from the input. The problem of this simple method is that there are obvious seams between the tiled patches (see Figure 2-2). Therefore, the main goal of the patch based approaches is

to find techniques to remove or reduce the visible seams between patches when they are placed into the output. Three recent patch based techniques: image quilting [50], graphcut textures [101], and Liang's approach [106], are discussed below.

[Image Quilting] As a patch based approach for texture synthesis, image quilting [50] uses the overlap constraints between neighboring patches and the minimum error path in the overlap region of two patches to remove or reduce the visible seams in the output images. The basic idea of image quilting works as follows.

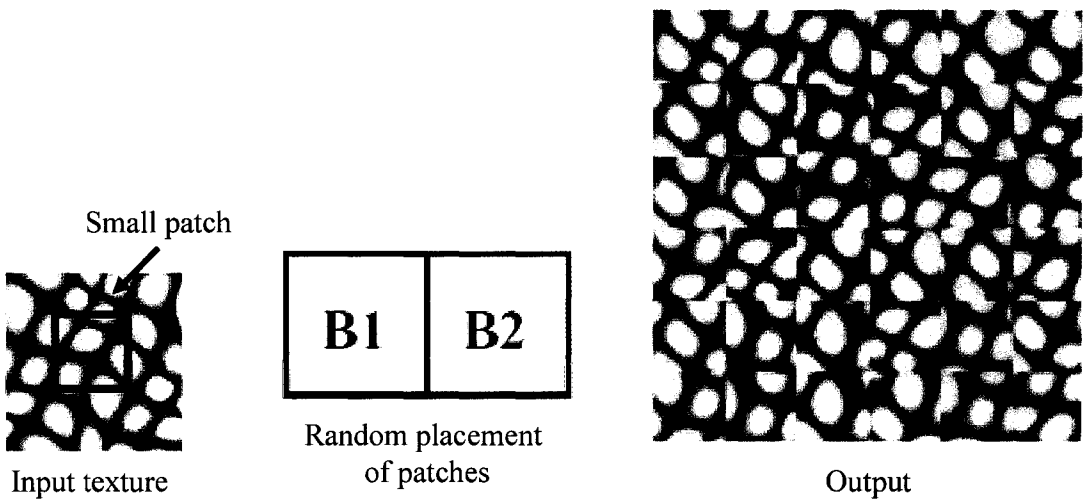


Figure 2-2: An example of a simple patch based approach.

In image quilting [50], neighboring patches, i.e. patches that meet together in the output, are first placed together with overlap constraints. To insert a new patch into the output image, instead of choosing the patch randomly from the input, the algorithm first searches among all the available patches for a patch that will have the smallest visible difference in the overlap region where the new patch meets with the old patches in the output. The purpose of this step is to incorporate the overlap constraints to make sure that

the inserted patch agrees with its neighboring patches along the overlap region as much as possible.

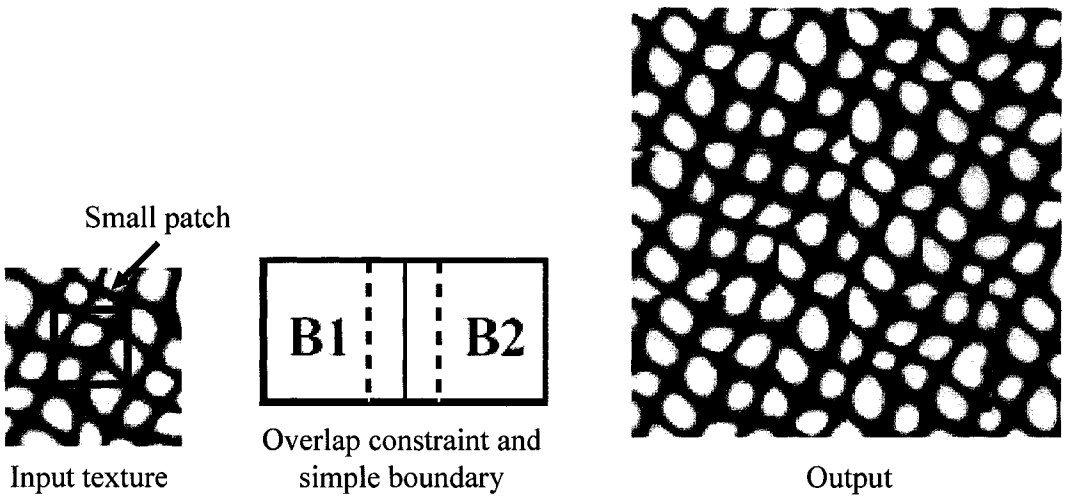


Figure 2-3: An example of synthesis results by selecting a new patch (e.g. B2) with the smallest visible difference to the old patches (e.g. B1) in the overlap region and using a simple boundary between patches with overlap. The simple boundary is shown as a solid line between two dashed lines.

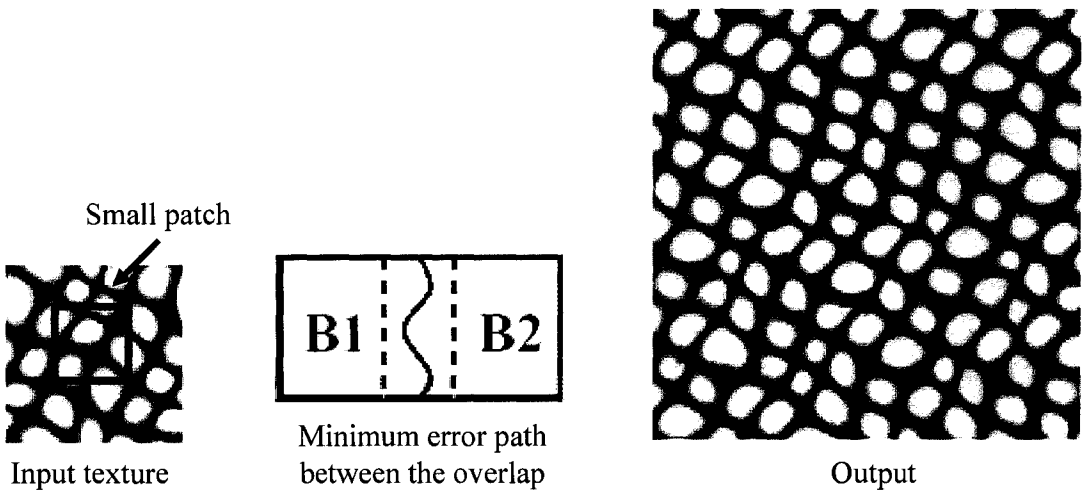


Figure 2-4: An example of synthesis results using the minimum error path (shown as a solid curve between two dashed lines) as the boundary between B1 and B2.

Once two patches are placed together with overlap, the algorithm then decide a boundary where the two patches can separate from each other (i.e. the boundary that determines from which patch a pixel in the overlap region comes from). A simple solution is to place the boundary in the middle of the overlap region, which unfortunately will still cause noticeable edges between patches as shown in Figure 2-3. To reduce the visible edges as much as possible, a minimum error path (see Figure 2-4) in the overlap region is used as the boundary, which can be efficiently calculated with dynamic programming [50].

In general, the size of the patch is difficult to be predetermined and may be different for different types of input textures [50]. The criterion is that the patch should be big enough to capture the relevant structures of the given texture. In the image quilting paper [50], the patch size is left as a user controlled parameter and the size of the overlap is 1/6 of the patch size for all of the experimental results. The image quilting algorithm is summarized below:

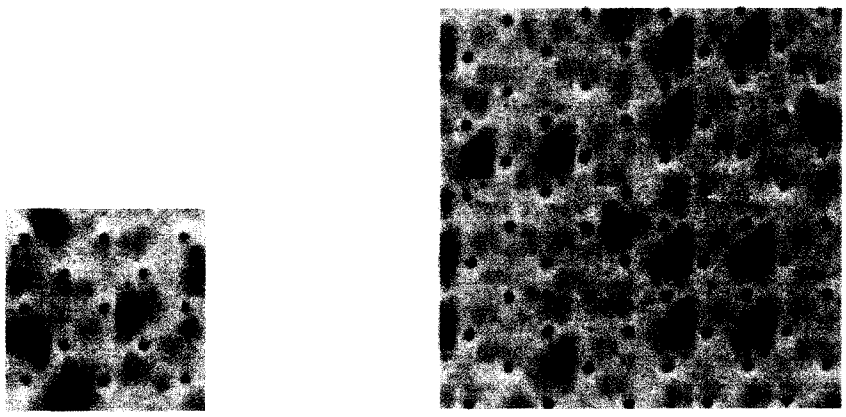
1. Go through the output image in raster scan order step by step. At each step, a new patch is inserted.
2. At each location where a new patch is inserted, search the input texture for a set of candidate patches that satisfy the overlap constraints within some error tolerance.
3. Randomly choose one patch from the candidate set and perform the following steps.
    1) Use dynamic programming to compute the minimum error path along the overlap region between the newly chosen patch and the old patches.

2) Make the minimum error path as the boundary of the new patch.

3) Paste the new patch into the output texture.

4. Repeat Step 2 and 3 until the output texture is finished:

[Graphcut Textures] Although the image quilting algorithm [50] can produce good results for a broad range of textures, seams between patches in the output texture may still be quite noticeable for some types of textures (see Figure 2-5). To solve this problem, the algorithm can be modified to remember the old seams between the patches that are already inserted into the output image during the synthesis process, and to further reduce the old visible seams in the output texture whenever necessary. Since dynamic programming used in the image quilting algorithm [50] cannot keep track of the seams between existing patches in the output, a new optimization technique should be deployed so that the output results can be refined as required. In graphcut textures [101], Kwatra et al. use graph cuts [13, 184] to refine the output texture as well as to find the minimum error paths between patches that have overlaps with each other. The basic idea of graphcut textures [101] is described as follows.



Input Output

Figure 2-5: An example of visible seams in the output using the image quilting algorithm.

The approach is based on the optimization techniques of graph cuts [13, 184], which have become popular recently for efficiently solving the labeling problems [184] in computer vision and computer graphics. In a labeling problem, a set of sites P (e.g. the set of all pixels in an image) and a set of labels L (e.g. the set of all possible gray levels for a pixel) are given, the objective is to find the global (or nearly global) optimal labeling f (i.e. a map from P to L) which minimizes the energy of f. In general, the labeling problem is intractable. For example, for an image of size 32×32 (i.e. 1024 sites) with 256 gray levels (i.e. 256 labels) for each pixel, there are 256^{1024} labelings.

Recent research works have shown that graph cuts can be used to *efficiently* find the global or nearly global optimal solutions for the labeling problems with energy functions that incorporate everywhere smooth, piecewise constant, and piecewise smooth prior constraints [13, 184]. The basic idea of using graph cuts for solving a labeling problem is to construct a weighted graph in a way such that there is a one-to-one correspondence between the set of all cuts of certain type in the graph (e.g. the set of all elementary cuts in an $\alpha - expansion$ graph [13, 184]) and the set of all labelings. With the one-to-one correspondence established between the set of graph cuts and the set of labelings, the labeling problem is converted to finding the min-cost cut in the graph [13, 184]. In graphcut textures [101], a specific type of $\alpha - expansion$ graph, which is discussed in details in a separate technical report [152], is built to find the minimum error cut (i.e. path) between the existing patches in the output and the new patch to be inserted into the output.

The technique of graphcut textures [101] is more general than image quilting [50]. In fact, the image quilting algorithm can be carried out in a different way by replacing the dynamic programming with the standard graph-cuts techniques. Suppose a new patch *B* is inserted into the output that overlaps with an existing patch *A*, a simple graph as shown in Figure 2-6 can be constructed as follows. The existing patch in the output is represented by the source node (i.e. the node with label "Patch *A*") in the left, and the new patch is represented by the sink node (i.e. the node with label "Patch *B*") in the right. Both the source node and the sink node are called terminal nodes. For simplicity reasons, it is assumed that there are only 9 pixels in the overlap region between the new patch *B* and the existing patch *A*. Suppose that 4-nearest-neighbor interaction is assumed in graphcut textures [101], then each node has edges connecting its left, right, top, and bottom nodes. For example, in Figure 2-6, node 5 has edges connecting to node 2, 4, 6, and 8, respectively.

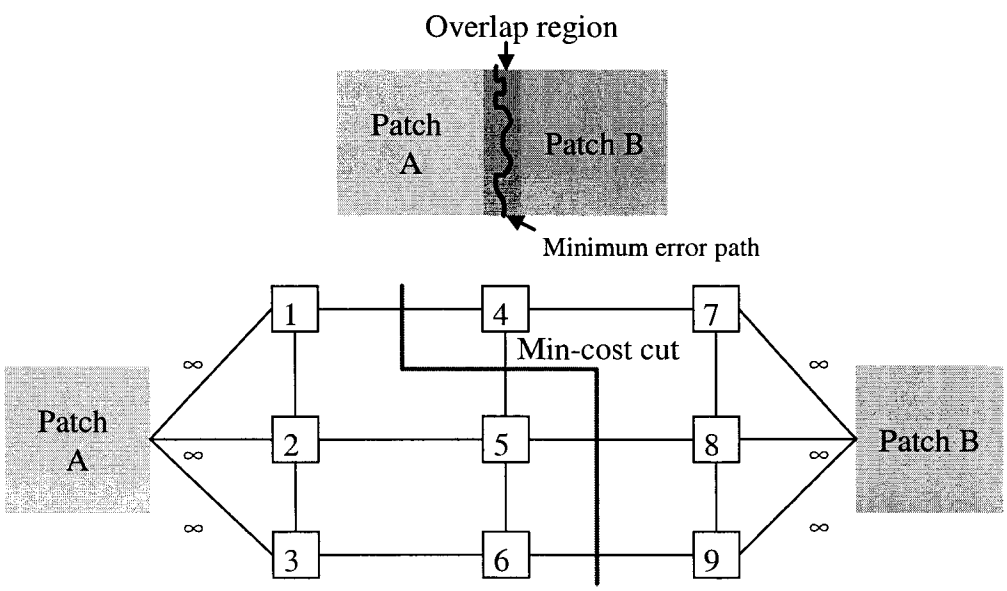


Figure 2-6: An example of graph formulation of finding the minimum error path.

For each edge e_{pq} in the overlap region connecting neighboring nodes, p and q, a weighted cost $w(p,q,A,B)$ of e_{pq} is defined as:

$$w(p,q,A,B) = |A(p) - B(p)| + |A(q) - B(q)|, \qquad (2.3)$$

where $A(p)$ is the gray level of pixel p from patch A, and $B(p)$ is the gray level of p from patch B. If an edge between a terminal node (A or B) and a non-terminal node is assigned an infinite cost, the non-terminal node will be insisted to come from the patch represented by the terminal node. For example, in Figure 2-6, if both edges e_{A1} and e_{8B} have ∞ costs, then node 1 retains its old patch label (i.e. patch A) and node 8 is assigned to the new patch B. The minimum error path calculated by dynamic programming in image quilting [50] is equivalent to the min-cost cut in the graph shown in Figure 2-6, which can be calculated by standard graph-cuts techniques [163].

For a given input texture image, the objective is to generate an output texture image by iteratively copying small patches from the input to the output so that the visible seams are as few or as invisible as possible. The first patch is copied at random into the output. Now, suppose several patches have already been placed into the output, and a new patch will be inserted into a region where multiple patches already meet. There are seams (referred as old seams later) along the border between old patches, and in image quilting algorithm [50], the old seams are not taken into account as constraints when laying down a new patch into the output texture. Therefore, the graph shown in Figure 2-6 cannot be used to incorporate old-seam constraints. In other words, once the old seams are created, they cannot be reduced later in the image quilting algorithm.

To incorporate the old-seam constraints, $\alpha-expansion$ graphs [13, 184] are constructed in graphcut textures. As illustrated in Figure 2-7, the old patches, which are already placed in the output, are represented by the source node with label "Existing Patches A," and the new patch to be inserted is represented by the sink node with label "New Patch B." For each node p in the overlap region, let A_p represent the particular patch that pixel p comes from. For each pair of neighboring pixels p and q, if $A_p = A_q$ (i.e. p and q have the same initial patch label), then there is no old seam between p and q, according to the properties of $\alpha-expansion$ graphs [13, 184], a weight (i.e. cost) of $w(p,q,A_p,B)$ given by Eq. 2.3 is assigned to edge e_{pq}.

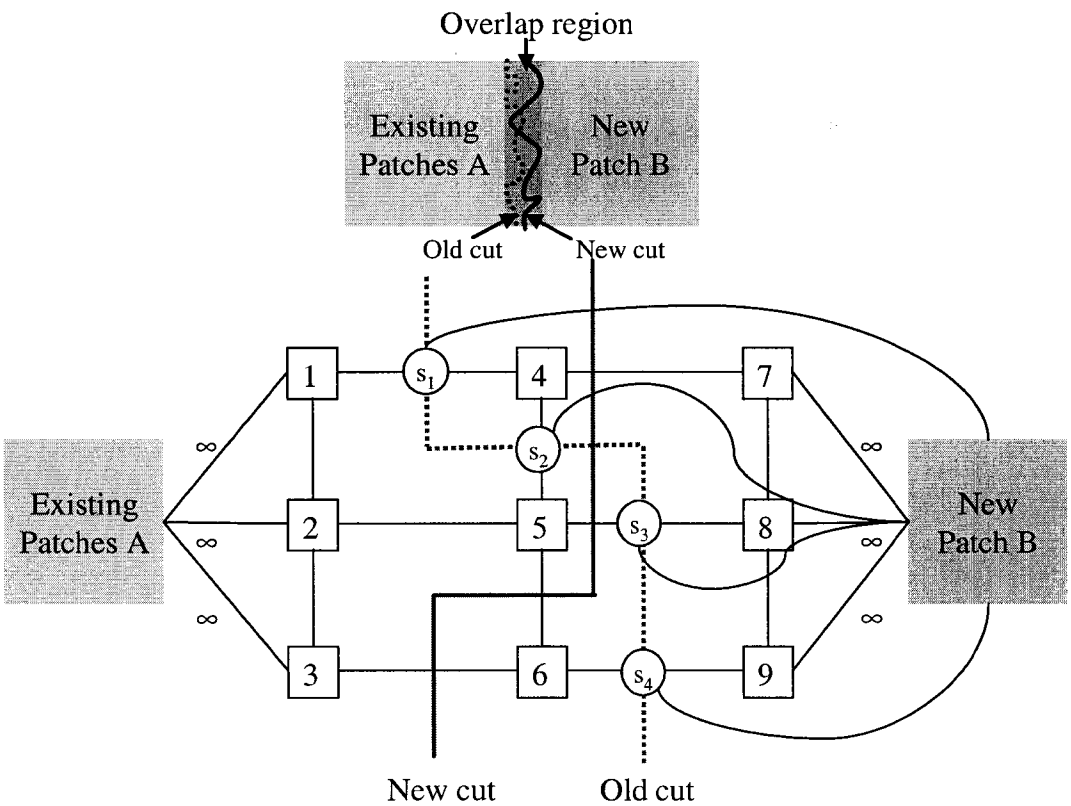


Figure 2-7: The $\alpha-$expansion graph constructed for graphcut textures.

For each pair of neighboring pixels p and q in the overlap region, if $A_p \neq A_q$ (i.e. p and q come from different existing patches), then there is an old seam between p and q and a seam node s is created between p and q. According to the fundamental theory on $\alpha - expansion$ graphs [13, 184], a triple of edges, e_{ps}, e_{sq} and e_{sB}, are created at s each with an appropriate weight assigned. The weights for edge e_{ps}, e_{sq}, and e_{sB} are $w(p,q,A_p,B)$, $w(p,q,B,A_q)$, and $w(p,q,A_p,A_q)$, respectively.

For example, in Figure 2-7, there is no old seam between node 2 and 5, which implies that pixel 2 and 5 come from the same old patch A (i.e. $A_2 = A_5 = A$), the weight for edge e_{25} is $w(2,5,A,B)=|A(2)-B(2)|+|A(5)-B(5)|$. On the other hand, since there is an old seam between node 1 and 4, thus a seam node s_1 is created between 1 and 4. In addition, the seam node s_1 is connected to the sink node B by a weighted edge, whose weight is the old matching cost [101] when the old seam between node 1 and 4 is created, which is given by $w(1,4,A_1,A_4)$ (see Eq. 2.3). The edge between 1 and s_1 is assigned a weight $w(1,4,A_1,B)$, which measures the matching cost when pixel 4 comes from the new patch B. Similarly, the edge between s_1 and 4 is assigned a weight $w(1,4,B,A_4)$, which measures the matching cost when pixel 1 comes from the new patch B.

Kwatra et al. have argued in their paper [101] that in a graph constructed in such a way as shown in Figure 2-7, the min-cost cut C can cut at most one of the three edges created at any seam node and this is true only if the cost measure defined by Eq. 2.3 is metric [184]. If none of the three edges at a seam node is cut by the min-cost cut C, then the old seam is removed. Otherwise, if the edge between a seam node s and the new patch node is cut by the min-cost cut, then the old seam at s remains. If the edge between a

seam node (e.g. s_3 in Figure 2-7) and one of its adjacent nodes (e.g. 5 in Figure 2-7) is cut by the min-cost cut, then the old seam at seam node is removed and a new seam is introduced at the same edge position where the old seam passes through (i.e. the old seam is overwritten by the new seam). Finally, if an edge without a seam node is cut by the min-cost cut, then a new seam is introduced at that edge.

For example, in the graph shown in Figure 2-7, the new cut in blue color, denoted by *C*, is the min-cost cut calculated using graph cuts techniques [13, 184]. At seam node s_1, since *C* cuts edge $e_{s_1 B}$, the old seam at s_1 remains. At seam node s_2, the same situation as at s_1 happens, thus the old seam at s_2 also remains. At s_3, *C* cuts the edge between 5 and s_3, thus the seam at s_3 is replaced by a new seam at the same location. At s_4, the new cut does not cut any of the three edges from it, thus the old seam at s_4 is removed. Finally, new seams are introduced between node 4 and 7, 5 and 6, and 3 and 6 since the new cut passes through the edges between them.

In summary, the discussion on the theoretic side of the graph constructed for the graphcut textures is very weak, and few information is provided in the original paper [101] to convince the reader the correctness of the graph's construction. After carefully reading the paper, the reader may still not sure how the graph constructed in the paper (e.g. Figure 3 in [101]) fits into the general framework of $\alpha - expansion$ graphs [13, 184]. There are, for example, some important theoretical issues on the graph used in the paper remains unanswered, which are listed below:

1. What properties a cut *C* has? For example, in the graph shown in Figure 2-7, can a cut *C* cuts edge e_{12}? What edges it can cut and what edges it cannot cut?

2. Given a cut C in the graph, how to define the corresponding labeling f^C?
3. Why a min-cost cut C can cut at most one of the three edges at any seam node? Is this true for any cut C in the graph?
4. In general, an $\alpha-expansion$ graph has a set of *elementary cuts*, which has a one-to-one correspondence to the set of all labelings within one $\alpha-expansion$ of the initial labeling, and this is the key to solving a labeling problem using $\alpha-expansion$ graphs [13, 184]. How is an elementary cut defined in the graph used in the graphcut textures paper? How can the one-to-one correspondence be established?

In a separate technical report [150], we have addressed the above important theoretic issues under the framework of $\alpha-expansion$ graphs [13, 184]. We have developed the concept of *complete* $\alpha-expansion$ graphs in 2D to give the theory support and mathematical proofs of the graphcut textures [101].

[Liang et al.'s Approach] Concurrent to the work of image quilting [50], Liang et al. propose a patch-based sampling technique for real-time texture synthesis [106]. The general algorithm of their approach is similar to that of the image quilting approach. The only difference is that when handling overlaps between patches, Liang et al.'s approach uses a blending technique, called feathering [176], to give a smooth transition between textures in the overlap regions, while Efros and Freeman's approach uses dynamic programming to calculate a minimum error boundary cut to reduce the visible seams between textures in the overlap region as much as possible. Liang et al.'s patch-based sampling algorithm for texture synthesis is summarized below:

1. Randomly choose a texture patch B_0 from the input texture $\boldsymbol{X}$. Paste B_0 into the output $\boldsymbol{Y}$. Set $k = 1$.
2. For any new patch B_k to be inserted into the output $\boldsymbol{Y}$, let $\boldsymbol{B} = \{B_0, ..., B_{k-1}\}$ be the texture patches already inserted into $\boldsymbol{Y}$, E_{out}^k be the boundary zone of $\boldsymbol{B}$ that will overlap with the boundary zone E_{B_k} of B_k (see Figure 2-8). Find all texture patches in the input $\boldsymbol{X}$ whose boundary zones match E_{out}^k, and let $\psi(\boldsymbol{B})$ be the set of all such texture patches, i.e.:

$$\psi(\boldsymbol{B}) = \{B \mid B \text{ is a patch in } \boldsymbol{X} \text{ such that } d(B, E_{out}^k) < \varepsilon\}, \qquad (2.4)$$

3. If $\psi(\boldsymbol{B})$ is empty, set $\psi(\boldsymbol{B}) = \{B_{\min}\}$, where $B_{\min}$ is chosen from $\boldsymbol{X}$ such that its boundary zone is the closest to E_{out}^k.
4. Randomly choose an element from $\psi(\boldsymbol{B})$ as the new patch B_k. Paste B_k into the output $\boldsymbol{Y}$. Set $k = k + 1$.
5. Repeat steps 2, 3, and 4 until the output $\boldsymbol{Y}$ is finished.
6. Perform blending in the boundary zones using the feathering technique [176].

An illustration of the above algorithm is shown in Figure 2-8. The gray area is the already synthesized region in the output. The areas in dashed lines are the boundary zones. Figure 2-8 (a) shows that when a new patch E_{B_k} is inserted, the boundary zones E_{out}^k and E_{B_k} should match. Figure 2-8 (b) is the configuration for boundary zone matching in the beginning of texture synthesis. Figure 2-8 (c) is the configuration in the middle. And, Figure 2-8 (d) is the configuration in the end. The overlapping boundary

zones are blended together by using the feathering technique [176]. In Eq. 2.4, the parameter ε is controlled by the user, and $d(E_{B_k}, E^k_{out})$ is given by:

$$d(E_{B_k}, E^k_{out}) = \left[\frac{1}{M} \sum_{i=1}^{M} (p_i(E_{B_k}) - p_i(E^k_{out}))^2 \right]^{1/2}, \qquad (2.5)$$

where M is the number of pixels in the boundary zone, and $p_i(E_{B_k})$ and $p_i(E^k_{out})$ are the color values of the i^{th} pixel in the boundary zones E_{B_k} and E^k_{out}, respectively.

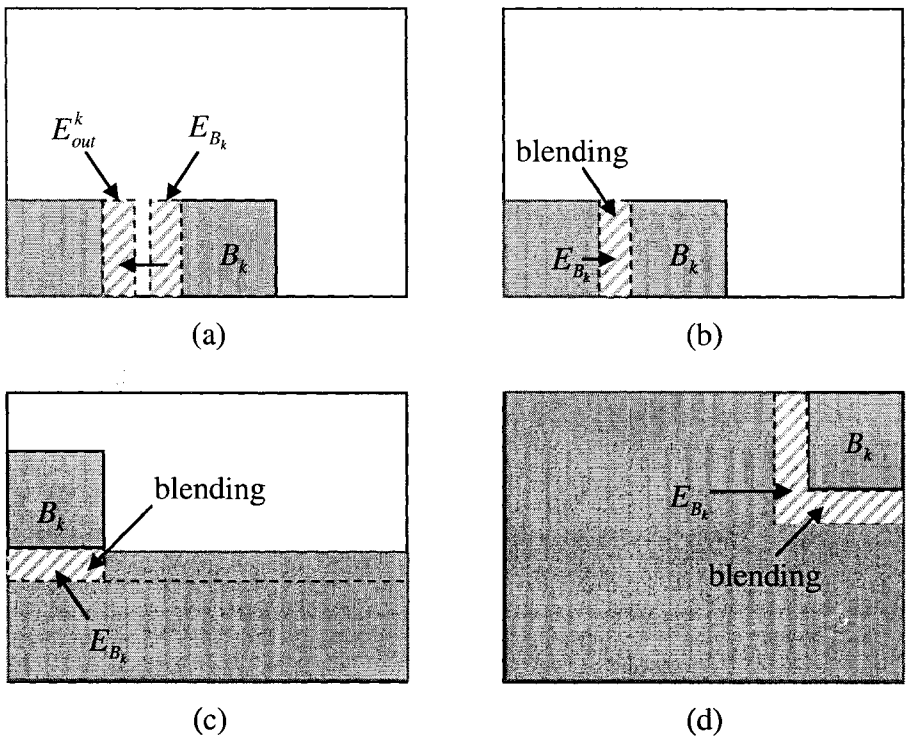


Figure 2-8: An illustration of the patch-based sampling approach for texture synthesis by Liang et al.'s approach.

The calculation of the set $\psi(\boldsymbol{B})$ in step 2 of the algorithm is essentially an ANN (approximate nearest neighbors) search in high-dimensional space, and an efficient optimization technique called the quadtree pyramid data structure is employed for the search. To further speed up the algorithm, PCA (Principal Component Analysis) [91] can be used to reduce the dimension of the search space. This is done by first finding the

eigenvalues and eigenvectors of the covariance matrix of the data points in the search space. The original data points are then projected into the subspace spanned by the eigenvectors of the largest eigenvalues that contains the main variations of the data distribution. The dimension of the subspace, where the search is carried out, is much smaller than that of the original search space (see [106] for the details).

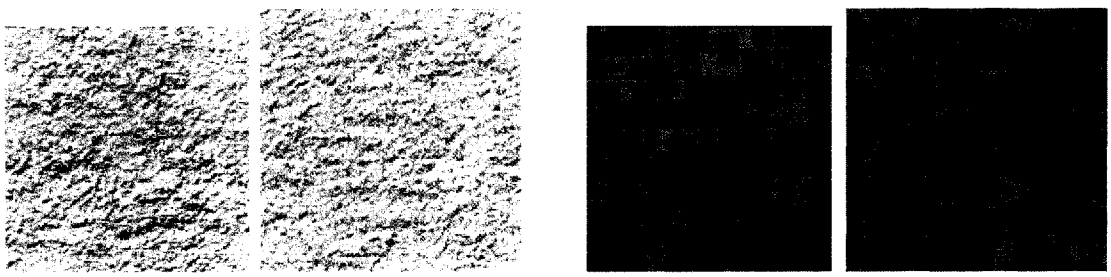
Before ending this section, we give some comments on patch-based techniques for texture synthesis. In general, they are much faster than pixel-based techniques and thus the algorithms can run in real time. However, there are several challenging issues that need further research. The first is how the size of patches used during synthesis affects the results and how to determine an optimal patch size for a given input texture image. Currently, all techniques use a fixed size of patches (e.g. 16x16 or 32x32). The second is how to choose an optimal size between overlap regions, and this problem is related to and dependent on the first problem. The last is that sometimes even a minimum error path is found between two merging patches, there are still visible seams at the boundary of the patches. The cause of the problem is due to the inaccurate similarity measure based on the SSD (sum of square differences) of pixel color values. Wu and Yu [200] have addressed this problem and give a solution based on structural texture feature matching and deformation. Their approach has achieved some degree of success.

2.1.4 Feature Matching Approach

In this approach, textures are modeled as a set of features, and synthesized textures are generated by matching the features computed from a given input texture. Techniques in this approach includes the multiresolution histogram matching [81], the

feature-based pyramid sampling [35], the joint statistics matching of complex wavelet coefficients [143, 167], and the statistical learning based method [6]. These algorithms are usually faster than the MRF models.

In Heeger and Bergen's method [81], new textures are synthesized by coercing a white noise image into a given sample texture by matching the histograms of filter responses at different spatial scales and orientations. Given a texture image $\boldsymbol{X}$, their algorithm first decomposes $\boldsymbol{X}$ into a steerable pyramid $\boldsymbol{P}(\boldsymbol{X})$ [166]), which is a set of images (called *pyramid subbands*) filtered from the input image at different spatial scales and orientations. Then, starting from a random noise image $\boldsymbol{Y}$, the algorithm modifies each pyramid subband in $\boldsymbol{P}(\boldsymbol{Y})$ so that its histogram matches the histogram of the corresponding pyramid subband in $\boldsymbol{P}(\boldsymbol{X})$. The above histogram matching process is repeated several times (e.g. 5 times or so reported in the original paper [81]) until there is no further improvement in the quality of the output.



(a) (b)

Figure 2-9: Examples of texture synthesis using Heeger and Bergen's algorithm. In each pair, the left image is the input and the right image is the synthesized texture. Texture in (a) is an example of success, and texture in (b) a failure.

Heeger and Bergen technique [81] works well for highly stochastic textures but fails on some structural textures such as bricks (see Figure 2-9) since histograms are insufficient to differentiate textures [143, 144]. De Bonet [35] presents an approach to synthesize new textures by sampling from a given input texture conditioned on the local feature responses (called local parent structures in the original paper [35]) at different spatial scales. De Bonet's algorithm also synthesizes textures in a multiresolution fashion, which is similar to the Heeger and Bergen algorithm [81]. However, instead of matching the global histogram at each pyramid subband between the output and the input, De Bonet's algorithm tries to match the local features of the corresponding pixels in the output and input images. As a result, his technique works better than Heeger and Bergen's histogram-matching technique [81] on structural textures, and thus on a wider range of textures. However, if the input texture is not tileable, there will be boundary artifacts in the output.

Using statistical learning, Bar-Joseph et al. introduce a more general model, which can be viewed as an extension of De Bonet's approach [35]. Instead of operating on a single input texture sample, which is the case of De Bonet's algorithm, Bar-Joseph's algorithm takes multiple texture samples as input and synthesizes a new texture from a mutual source of input samples. Another significant improvement is that Bar-Joseph's model can be used to synthesize time-varying textures (i.e. texture movies) in addition to static textures.

The general algorithm of Bar-Joseph's approach is as follows. Given k n-dimensional signals of input texture samples, the algorithm constructs a hierarchical multiresolution analysis (MRA) of each signal sample using wavelet transformations [33,

166]. Each MRA is represented as a $2^n - ary$ tree. It is assumed that all the paths in a particular tree are realizations of the same stochastic process. The algorithm generates a new random MRA tree by statistically merging the MRA trees of the input samples. Finally, the algorithm transforms the newly generated MRA back into an n-dimensional signal by applying a process inverse to the MRA. The new n-dimensional signal is used to produce a synthesized texture that is statistically and perceptually similar to each of the input samples, but at the same time different from them.

Another research work on texture synthesis using feature matching is done by Simoncelli and Portilla [143, 167]. Their approach synthesizes new textures by matching the joint statistics of the steerable pyramids of the input and output images. The joint statistics used in their model include the marginal statistics, the wavelet coefficient correlations, the magnitude correlations, and the cross-scale phase statistics. The matching between the joint statistics is done by a greedy entropy-minimization approach similar to that of Zhu et al.'s approach [208]. Simoncelli and Portilla's algorithm can also apply to multiple input samples to generate texture mixtures. In general, Simoncelli and Portilla's model can successfully capture global textural structures but fails to preserve local patterns.

2.1.5 Cooccurrence Matrix Approach

Cooccurrence matrices have been used as a powerful tool for texture analysis, synthesis, segmentation and classification. Various texture analysis and synthesis approaches based on cooccurrence matrices have been proposed in the literature [21, 23, 24, 31, 34, 51, 55, 58, 59, 67, 75, 76, 90, 112, 131, 137]. Since cooccurrence matrices are

normally calculated by considering pixel gray level values, models based on cooccurrence matrices are generally called the GLCM (Gray Level Cooccurrence Matrix) models [77]. The basic idea of texture analysis and synthesis using the GLCM models is described in this subsection. For detailed information on this, the reader is referred to [31, 77].

For a given texture image $\boldsymbol{X}$ with m gray levels, consider a displacement vector $\boldsymbol{d} = (d_x, d_y)$, the *gray level cooccurrence matrix corresponding* to $\boldsymbol{d}$, denoted by $\boldsymbol{C}(\boldsymbol{d}) = (c_{ij}^{\boldsymbol{d}})_{0 \le i,j \le m-1}$, is an $m \times m$ matrix, where its entry $c_{ij}^{\boldsymbol{d}}$ counts the number of pairs of pixels generated by the displacement $\boldsymbol{d}$ over image $\boldsymbol{X}$, which have gray level i and j, respectively. The calculation of $c_{ij}^{\boldsymbol{d}}$ is given by:

$$c_{ij}^{\boldsymbol{d}} = \sum_{\substack{(x,y)\in I \\ (x,y)+\boldsymbol{d}\in I}} \delta(i - g(x,y)) * \delta(j - g(x+d_x, y+d_y)) + \sum_{\substack{(x,y)\in I \\ (x,y)-\boldsymbol{d}\in I}} \delta(i - g(x,y)) * \delta(j - g(x-d_x, y-d_y)) \quad , \quad (2.6)$$

where δ is the Dirac delta function, i.e. $\delta(x) = \begin{cases} 1, & if\ x = 0 \\ 0, & otherwise \end{cases}$, and $g(x,y)$ is the gray level at pixel location (x,y).

1	0	1
0	0	0
0	0	0

(a)

	0	1
0	8	2
1	2	0

(b)

	0	1
0	2/3	1/6
1	1/6	0

(c)

Figure 2-10: An example of an image and its GLCM with $\boldsymbol{d} = (1,0)$.

Figure 2-10 gives an example of a binary image in (a) and its corresponding GLCM in (b) with a displacement of $\boldsymbol{d} = (1,0)$. If the cooccurrence matrix $\boldsymbol{C}(\boldsymbol{d})$ is normalized such that $\sum_{i,j=0}^{m-1} c_{ij}^{d} = 1$ (e.g. Figure 2-10 (c)), then a discrete probability distribution of c_{ij}^{d} is obtained, based on which a new texture image can be sampled.

Based on Julesz's important conjecture on the role of high order statistics in human vision [93], texture features such as contrast, correlation, variance, inertia, entropy, cluster shade, and local homogeneity (see [77] for definitions) can be computed from cooccurrence matrices to analyze and characterize textures [77]. It has, however, been shown that the transformation of cooccurrence matrices into secondary features has led to a significant loss of information that are originally captured by cooccurrence matrices [31, 112]. Studies on comparing the relative power of various texture analysis techniques have convinced people that directly using the elements of cooccurrence matrices as texture features generally outperforms the methods that use secondary features derived from cooccurrence matrices [26, 27, 76, 122, 193]. As well, it has been shown that cooccurrence matrices themselves can be directly used in texture analysis and synthesis [55, 58, 59, 67, 112, 131].

For a given pixel in image $\boldsymbol{X}$, consider its eight neighboring pixels, which correspond to four displacements $\boldsymbol{d}_0 = (1,0)$, $\boldsymbol{d}_1 = (1,1)$, $\boldsymbol{d}_2 = (0,1)$, and $\boldsymbol{d}_3 = (-1,1)$. Let $\boldsymbol{C}_i = \boldsymbol{C}(\boldsymbol{d}_i)$ be the cooccurrence matrix generated by displacement $\boldsymbol{d}_i$, which is calculated using Eq. 2.6. To synthesize a new texture image $\boldsymbol{Y}$, which has the same (or as close as possible) cooccurrence distribution (i.e. cooccurrence matrices) as the input texture $\boldsymbol{X}$, the following algorithm can be used:

1. Initialize $\boldsymbol{Y}$ as a random noise image, which has the same histogram as the input image $\boldsymbol{X}$.
2. Use the Metropolis sampling [32, 63] to iteratively transform the initial image $\boldsymbol{Y}$ into a final texture image which has the desired cooccurrence distribution. At each iteration, the following steps are performed:
   - Randomly choose two pixels p_1 and p_2 from image $\boldsymbol{Y}$.
   - Exchange the gray levels of p_1 and p_2 if the exchange will cause an improvement (i.e. image $\boldsymbol{Y}$ has a closer cooccurrence distribution to that of input image $\boldsymbol{X}$). To measure the closeness of the cooccurrence distribution between $\boldsymbol{Y}$ and $\boldsymbol{X}$, the following formula is used:

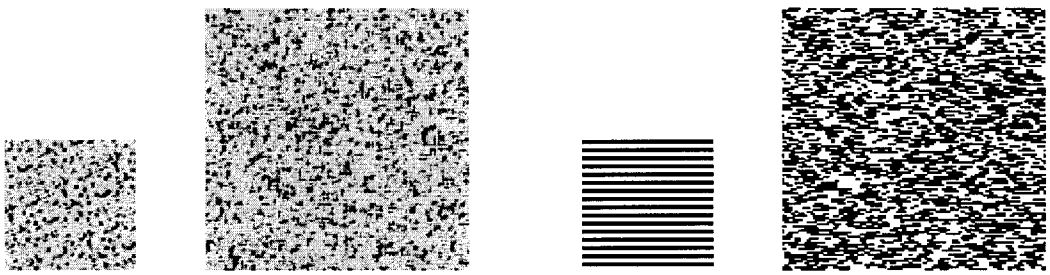
$$E = \frac{1}{4}\sum_{k=0}^{3}\sum_{i,j=0}^{G-1}[c_{ij}^{k}(\boldsymbol{Y}) - c_{ij}^{k}(\boldsymbol{X})]^{2}, \qquad (2.7)$$

     where $\boldsymbol{C_k}(\boldsymbol{X}) = [c_{jk}^{k}]_{0\le i,j\le G-1}$ is the cooccurrence matrix of image $\boldsymbol{X}$ corresponding to displacement $\boldsymbol{d_k}$, $k = 0,1,2,3$. The smaller the value E is, the closer the cooccurrence distributions between $\boldsymbol{X}$ and $\boldsymbol{Y}$.
   - If the gray level exchange between p_1 and p_2 failed to cause an improvement, the exchange is performed with a probability p given as follows:

$$p = \frac{1}{1+\exp(E/T)}, \qquad (2.8)$$

     where E is given in Eq. 2.7 and T is a temperature that is slowly cooled down.
3. Step 2 is repeated until the value E reaches zero or does not change any further.

The above algorithm is essentially the same algorithm used in Lohmann's work [112]. Since only four smallest displacements (measured by $|\boldsymbol{d}|$) are used, Lohmann's approach will fail for structural textures (see Figure 2-11(b)) although it works well for some stochastic textures with micro particles (see Figure 2-11 (a)) . To overcome this problem, large displacements and large number of cooccurrence matrices can be used, and this can be done by increasing the size of a target pixel's neighborhood over which the displacements are considered [31]. However, the number of cooccurrence matrices will increase drastically when the neighborhood size increases and this drastically increases the algorithm run-time. The solution to this problem is to use a multiresolution scheme [81, 158, 166, 190].



(a) (b)

Figure 2-11: Examples of texture synthesis using Lohmann's approach.

Another major problem with cooccurrence matrix approaches is that the distance function (e.g. Eq. 2.7) for measuring the similarity between the synthesized texture $\boldsymbol{Y}$ and the input texture $\boldsymbol{X}$ is not well defined in the sense that for a given input texture $\boldsymbol{X}$, the algorithm described in the previous subsection could generate a completely dissimilar texture $\boldsymbol{Y}$ but still has a zero distance to $\boldsymbol{X}$ (i.e. $E = 0$ in Eq. 2.7). The cooccurrence matrix approach [31, 34, 112] has been generalized by Elfadel and Picard under the aura

framework [51, 52, 137-139], which is described in the next chapter as background knowledge of the proposed work.

2.1.6 Structural Texture Modeling

Structural textures [92] are viewed as two dimensional patterns consisting of a set of micro-structures (i.e. primitives) which are arranged according to certain placement rules. Examples of structural textures are checker-board patterns, bricks, stones, periodic regular (or nearly regular) patterns, etc (see Figure 2-12). Correctly characterizing and synthesizing structural textures are difficult [74, 75, 104, 206].

Figure 2-12: Examples of structural patterns (image source: the Brodatz and the Vistex textures [124]).

In general, existing statistical texture models are unsuitable for synthesizing structural textures. Procedural texturing approaches [47, 103] have been used successfully for synthesizing some specific types of structural textures such as checkerboard patterns, bricks, etc. In Zhu et al.'s work [206], a texton-based approach is used for analyzing and synthesizing structural textures. In their model, a 2D structural pattern is viewed as a superposition of a set of image bases from an over-complete dictionary, and a texton is defined as a primitive (i.e. mini-template) consisting of a set of image bases with some specific geometric configurations and photometric properties. For

a given structural pattern, it has been shown in their paper that a small number of textons can be first learned from a set of training samples as repeating micro-structures and then used to synthesize a similar pattern.

Recently, a mathematical framework [110] based on the theory of crystallographic groups [70] has been proposed for modeling periodic regular (or nearly regular) patterns. The underlying mathematical theory for their model is the fact that there are only a finite number of basic patterns, called *symmetry groups*, for all possible periodic patterns in an n-dimensional space [120]. In particular, it is proved that in the 2D space there are only 7 frieze groups describing monochrome patterns that repeat along one direction and only 17 wallpaper groups for patterns that repeat along linearly independent directions to tile a plane.

Based on the above mathematical theory, algorithms are developed for analyzing and synthesizing 2D periodic patterns [110]. The goal of the analysis process is to characterize a given 2D periodic pattern by extracting the underlying translational lattices, classifying the pattern's symmetry groups, and identifying the representative motifs that perceptually characterize the pattern. Synthesized patterns are generated by tiling the representative motifs according to some placement rules (i.e. by centering the motifs on distinct centers of the highest rotation [110]).

There are two limitations in Liu et al.'s model [110]. The first is that only periodic patterns with Gaussian noise are assumed in the model, and thus geometric variations and distortions are not handled well. To solve this problem, the authors suggest an approach based on the correspondence of geometric visual elements (e.g. points of high boundary curvature) or an approach based on feature correspondence. The second limitation is that

the transformation of scaling is not considered in the model during the process of classification, which implies that all patterns must be taken by a camera perpendicular to the image plane. The authors suggest that affine and perspective imaging models [109] or local deformations of approximate periodic patterns can be used to address this problem [110, 111].

2.2 3D Texture Modeling

3D textures are visual patterns that appear on surfaces of 3D objects (see Figure 1-1). There are four approaches to generating synthetic textures onto 3D surfaces: texture mapping, procedural texturing, image-based surface texturing, and image-based solid texturing. In subsequent sections, we review representative techniques in each of the four approaches.

2.2.1 Texture Mapping

Texture mapping [10, 79, 80, 198] is the earliest technique for generating synthetic textures on surfaces of computer generated objects, which was first introduced by Catmull in 1974 [Catmull, 1974]. It is commonly used to add realism to otherwise dull synthesized images. Recently, research in this area has changed its focus from algorithms for software-based rendering systems to algorithms for high performance graphics hardware.

In texture mapping, a 2D texture pattern is mathematically mapped onto the surface of a 3D model that is used to represent a real world object. Then, the textured 3D surface is projected onto the output image-viewing plane. Three different coordinate systems are commonly used in texture mapping: texture space (*st*-space), object space

(*xyz*-space), and image space (*uv*-space). Two transformations are used to transform textures from the *st*-space to the *uv*-space. One is between the *st*-space and the *xyz*-space, which is called *texture-object* transformation (see Figure 2-13). If the texture is mapped orthogonally onto a planar quadrilateral, then the texture-object transformation may be as simple as an affine or bilinear transformation [79]. Otherwise, it may be a parametric transformation when the texture coordinates are used to represent non-Cartesian coordinates such as cylindrical or spherical. The other transformation used in texture mapping is between the *xyz*-space and the *uv*-space, which is called *object-image* transformation. It is usually either an orthographic projection for orthographic viewing or perspective projection for perspective viewing. For discussions on various kinds of texture-object transformations, the reader is referred to [10, 79, 80].

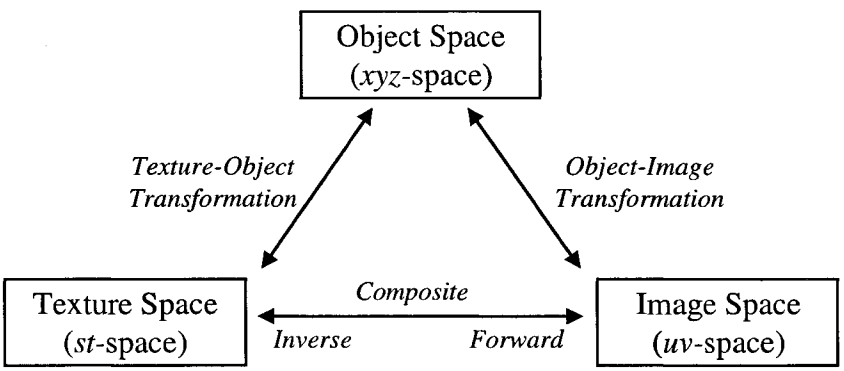


Figure 2-13: The coordinate systems and transformations in texture mapping.

Since the transformation from the texture space to the image space is a composite of the texture-object transformation and object-image transformation, one often concatenates the two transformations to save computations. The resulting composite transformation usually can be formulated either as a forward (texture-to-image) transformation or as an inverse (image-to-texture) transformation. Each method has its

own advantages and disadvantages, which have been discussed at length in [79, 198]. Figure 2-13 shows the relationship between the two transformations described before. Figure 2-14 gives a simple example of texturing an object's surface using texture mapping.

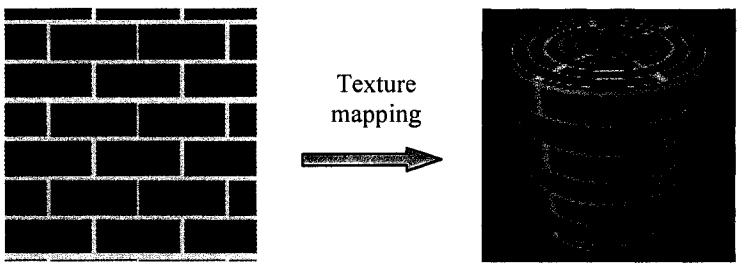


Figure 2-14: A simple example of texture mapping.

Various texture mapping techniques have been proposed in the literature. Following the work done by Catmull [16], Blinn and Newell [12] introduce a popular mapping technique called *reflection mapping* to map 2D textures onto the surfaces of objects. Using their technique, the texture-mapped surface appears to be reflecting an image of its surroundings. The environment mapping introduced by Greene [69] can be considered as an extension of the reflection mapping [12], in which a real 180-degree fisheye image of the sky is combined with the computer-generated image of a desert terrain to create a full-view environment cube. Greene also shows in his paper [69] that environment mappings can be pre-filtered and indexed with summed-area tables to solve the aliasing problem, which was first addressed by Williams in 1983 using the mip-mapping technique [196]. Since then, research on environment mappings has been carried out extensively. Recent advancements in this area include the extended prefiltered

environment maps [15, 95-97], the efficient irradiance environment maps [157], and the frequency domain environment maps [156].

Another important technique of texture mapping is *bump mapping* introduced by Blinn in 1978 [11]. This technique modifies the surface normal rather than the surface point as in reflection mapping [12] to generate bump-like textures on surfaces of objects. As an extension to bump mapping, Cook et al. [30] introduce *displacement mapping* in which textures are used to modify the surface point, not just the surface normal. Since modifying a surface point does change the surface normal at that point, displacement mapping often looks like bump mapping except that the bumps created by the former are visible on the silhouettes of objects. Recently, in Jagnow and Dorsey's work [85], the displacement-mapping technique has been used in virtual sculpting [85]. In Wang et al.'s work, the displacement mapping has been extended to the view-dependent case [187].

There are other texture mapping techniques such as the texture tiling [46], the decal mapping [7], the cell texturing [48], and the image-based transformations [119, 159, 194], which are discussed and reviewed in the surveys done by Weinhaus [192] and by Haeberli [73]. Recent works in this area on efficiently generating textures over an arbitrary surface using a 2D example include the lapped textures [145], the hierarchical pattern mapping [170], the jump maps [203], the triangle mesh texture map [114], the matchmaker for constraint texture maps [99], and the TextureMontage technique [205].

Texture mapping suffers several problems. One is the unacceptable artifact problem. To cover the surface of a large object, the algorithm must replicate the texture. This can cause either visible seams, or visible repetitions, or both, which are unacceptable. Another problem is the distortion problem (see the brick texture on the top

of the cylinder in Figure 2-14). This is caused by the fact that there is no natural and well-defined mapping from 2D texture images to arbitrary 3D surfaces due to the complexities of textures and 3D models. To solve these problems, procedural texturing [47] can be used, which is discussed in the next section.

2.2.2 Procedural Texturing

The second important approach for generating textures on the surfaces of objects is called procedural texturing [47], which was first introduced by Cook in 1984 [28]. With the introduction of solid texturing [132, 134] and texture basis function such as the Perlin's noise [134], the use of procedural texturing has been widely accepted in the computer graphics community. In this approach, procedures are used to generate synthetic textures without requiring input textures. By calling a compact procedure, textures are generated directly on surfaces of 3D objects without seams and without discontinuity. Using different procedures, realistic images of brick, marble, wood, stone, water, cloud, flame, and crumpled wrinkle can be generated efficiently.

Although, recent works on non-procedural texture synthesis [111, 145, 177, 179, 191, 201, 202, 204] have achieved some success in applying textures onto 3D surfaces, the procedural approach still has some advantages over the non-procedural approach:

1) Some natural phenomena with motions, such as gas, fire, fluid, cloud, etc., are difficult to synthesize using non-procedural approaches, while it is more appropriate and relatively straightforward to model using procedural texturing.
2) To generate hypertexture [135] using procedural texturing is easy and efficient, while it is difficult to do using non-procedural approaches.

3) Other advantages include compact representation, storage efficiency, no seams, no repetition, and no discontinuity.

Some of the useful procedural-texturing techniques include the shade tree [28], the pixel stream editor and solid texturing [132, 134], the hypertexture [135], the reaction-diffusion systems [56, 180, 197], cellular texture basis functions [154, 199], and the multiresolution procedural parameter estimator using genetic algorithms [153]. These techniques are briefly discussed below.

Cook's shade trees [28] are one of the earliest systems used to generate procedural textures during rendering. Shade trees enable the use of different tree-structured shading models for different surfaces such as copper, wood, grass, etc. The input parameters to the shading models can be manipulated procedurally. In this way, shade trees make it possible to use textures to control any part of the shading calculation. Color and transparency textures, reflection mapping, bump mapping, displacement mapping and solid texturing can all be implemented using shade trees. Cook's shade trees have provided the basis for most of the subsequent procedural shading works [123, 140, 147, 181, 187].

As an extension to Cook's shade trees, Perlin [134] introduces a complete procedural texture generation language, called Pixel Stream Editor (PSE), and lays the foundation for the most popular class of procedural textures in use today, in particular those based on *Perlin noise* [133, 134], a stochastic texture generation basis function that produces random numbers with a band-limited frequency spectrum and plays a major role in many procedural shaders. By using his noise basis function, Perlin has generated very

convincing representations of clouds, fire, water, stars, marble, wood, rock, soap films, crystal, etc.

Before the work done by Peachey [132] and Perlin [134], some researchers proposed procedural textures over a two-dimensional domain [57, 61, 161]. By extending their work to three dimensions, Peachey [132] and Perlin [134] independently introduce the concept of solid texturing in 1985, in which volumetric textures are generated by calling compact procedures. In general, solid texturing procedures are built on texture basis functions such as Perlin noise [134], wavelet noise [29], Worley's cellular texture basis function [199], the generalized cellular texture basis function [154], and the object distribution function [102]. A variety of solid textures such as wood, marble, water, cloud, flame, etc, can be modeled using solid texturing. For example, using a simple procedure defined in Eq. 2.9, rainbow-like textures can be generated on the surfaces of an object, where given a surface point p, the procedure $rainbow(p)$ returns a color at that point. Figure 2-15 shows an example of textured images with $k = 10, 8, 6, 4, 2$ in Eq. 2.9.

$$rainbow(p) = (noise(k * p), noise(k * p), noise(k * p)) \qquad (2.9)$$

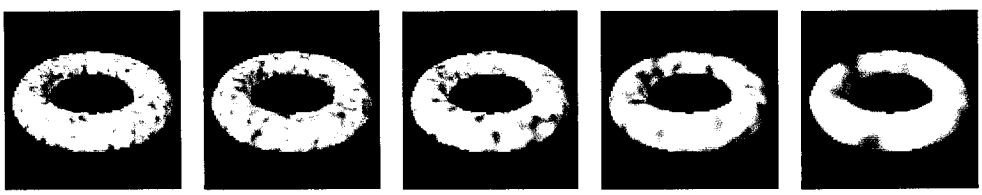


Figure 2-15: An example of solid texturing generated using the procedure defined in Eq. 2.9 with k = 10, 8, 6, 4, and 2, respectively.

Texturing procedures often require the setting of a large number of parameter values, a process that is time-consuming and non-trivial. This makes it difficult, if not impossible, to manually estimate the parameters of a given procedural texture. This problem is addressed in Qin and Yang's work [153] using a genetic-based multiresolution approach for estimating parameters. The key idea of the approach is to use an efficient search method, called the genetic-based search algorithm, to find appropriate values of the parameters for a given procedure. During the search process, for each set of the parameter values, the algorithm generates a temporary texture image using the given texturing procedure, then it compares the temporary texture image with the given target texture image to check if they match. The comparison between two texture images is done by using a multi-resolution MRF texture model. The search process stops when there is a match found. The estimated values of the parameters for a given procedure are the values of the parameters to the procedure to generate a texture image that matches the target texture image.

In the paper [153], it is also demonstrated that the proposed parameter estimation approach can be used to procedurally synthesize an input texture onto 3D surfaces. This, however, assumes that for each type of input textures, the user provides an appropriate texturing procedure to model them. It would be interesting to have a systematic way to design texturing procedures automatically based on given texture samples. Lefebvre and Poulin [103] have done some work in this direction, but their approach only works for structural textures such as brick and wood patterns. In Chapter 5, we present a new and general method to generate solid textures from input samples

The design of solid texture basis functions (i.e. texturing primitives) has also been studied by some other researchers. In addition to Perlin's noise [133, 134], Worley [199] proposes a cellular texture basis function, called Worley's noise, which can be used as a solid texturing primitive in procedural texturing for generating cellular textures on the surfaces of objects. As a generalization of Worley's noise, Qin and Yang [154] propose a generalized cellular texture basis function. For procedural texturing, all of the above texture basis functions can be used in color mappings, bump mappings, and fractals to produce visually interesting and impressive effects. By combining color or bump mapping with fractal technique, a variety of texturing procedures can be implemented based on them to generate crumpled wrinkle, wood, marble, cloud, water and flame-like textures. In Lagae and Dutre's work [102], a procedural object distribution function is used to generate regular-tile patterns that appear on cloth.

Another important technique to generate procedural textures is to use a reaction-diffusion process, in which two or more chemicals diffuse at unequal rates over a surface and react with one another to form stable patterns such as spots and strips that appear on the skins of animals. Earlier work in this area is to design the biological systems to generate simple patterns [5, 118, 121]. Later, Turk [180], and Witkin and Kass [197] design more complex reaction-diffusion systems to generate interesting patterns such as zebra stripes, sand ripples, and the swirling patterns of fingerprints, the rosettes found on leopards, and the multiple-width stripes found on some fish and snakes. The main disadvantage of the reaction-diffusion systems is that they are computationally expensive.

Other approaches in the area of procedural texturing include the sine wave approach [60], Wiener interpolation and sparse convolution [105], hypertexture [135],

spot noise [195], the artificial-evolution system[168], and the cellular particle simulator [56]. There are also recent works on real-time programmable procedural texturing [123, 147, 187], which will not be discussed in this thesis.

The disadvantages of procedural texturing include: (1) only limited types of textures can be modeled, (2) the design of procedures is based on the experience of the designer and is largely a manual process, and (3) the parameters of a texturing procedure are difficult to tune.

2.2.3 Image-Based Surface Texturing

In image-based surface texturing, a 2D texture sample and a 3D model are given, then similar textures to the input sample are generated onto the surface of the 3D model without visible seams and repetitions. Several techniques have been proposed in this area, which are briefly described as follows.

Wei and Levoy have extended their 2D texture synthesis algorithm [190] for generating 3D textures based on input texture samples [191]. In the 2D case, Wei and Levoy's approach generates synthetic texture by the nearest neighborhood search over rectangular lattices. To synthesize a given input texture sample onto 3D surfaces, the definition of search neighborhoods in 2D is generalized to the 3D case. For each mesh vertex, their extended algorithm [191] first establishes a local parameterization surrounding the vertex. Then, the algorithm uses the vertex's local parameterization to create a small rectangular neighborhood centered at the vertex and searches the input texture for similar neighborhood. Since a wide range of textures can be synthesized by

their 2D texture synthesis algorithm, their extended algorithm [191] performs well in generating 3D textures.

Concurrently, Turk [179] has also extended Wei and Levoy's 2D texture synthesis algorithm for generating 3D textures. The only difference between the two approaches is that in Turk's algorithm [179], a global tangent vector for the surface mesh has to be given by the user for the local parameterization of a vertex on the mesh. Another concurrent work to extend texture-from-sample methods (e.g. Wei and Levoy's [190] and Ashikhmin's [3] 2D texture synthesis algorithms) to 3D surfaces is done by Ying et al. [201].

In Zhang et al.'s work [204], progressively-variant textures (i.e. texture mixtures) are generated over arbitrary 3D surfaces based on texton masks. Recent research works [108, 177] have been done on generating bidirectional texture functions (BTF) on 3D mesh surfaces. In Fang's work [54], existing techniques of shape-from-shading and texture synthesis from input samples are combined to generate textures onto objects in photographs. In Chen's work [20], shell texture functions are used to synthesize realistic textures with translucency variations on surfaces from either 2D or 3D samples, e.g. a block of CT scan.

Compared with procedural texturing, image-based surface texturing can synthesize a wide range of textures. However, the approach may suffer the distortion problem on surfaces where the curvature is large. Another problem of the approach is that textures generated for one surface cannot be used for other surfaces. This limitation makes the techniques difficult to be used in procedural shaders [47].

2.2.4 Image-Based Solid Texturing

A solid texture is considered as a block of colored points in 3D space to represent a real-world material, for example, a wood trunk. Once the solid texture is available, any given 3D object can be textured by carving the object out of the volumetric data. Since solid textures define colors for each point in 3D space, they avoid the problems of distortion and discontinuity. However, solid textures are far more difficult to obtain than 2D textures; there is no easy way to obtain solid textures from real-world materials.

To combine the advantages of the procedural texturing and the image-based 2D texture analysis and synthesis, a number of researchers have developed techniques for generating solid textures from input samples, which we call image-based solid texturing. Different from image-based surface texturing, these techniques first synthesize a volumetric texture data from input samples, and then generate synthetic textures onto 3D surfaces by carving a given 3D object out of the volumetric data. While in image-based surface texturing, synthetic textures are directly synthesized onto surfaces of objects without generating the volumetric texture data first.

Some techniques in 2D texture analysis and synthesis are extended for generating solid textures based on input samples. As an extension to their 2D histogram-matching-based texture analysis and synthesis technique, for a given input texture image, Heeger and Bergen [81] generate solid textures by first initializing a volume of 3D noise and then coercing the noise so that the histogram of the volume matches that of the input image from coarse to fine resolutions. Since histograms cannot accurately characterize textures [143, 144], Heeger and Bergen's approach can only synthesize certain types of input textures, e.g. isotropic textures, onto 3D surfaces, and their approach fails for structural

textures. By using spectral and histogram analysis, Dischler et al.'s method [42] is able to synthesize some structural solid textures from input samples. Wei [188] and Paget [125] have also extended their respective 2D texture synthesis algorithms [129, 190] to generate structural solid textures as well as stochastic textures. However, both approaches work for only a limited range of textures.

There are other techniques developed by a number of researchers in image-based solid texturing. In Jagnow et al.'s work [86], a stereology-based approach is presented to successfully generate solid textures on some texture classes, e.g. marble-like textures. In their approach, in order to generate the correct results, extensive user interactions are required in creating 3D particles of desired shapes and of required distributions. While the user interaction provides flexibility, it is nontrivial to design a complex texture. If the shapes of predesigned 3D particles do not match the profiles of input textures, their algorithm will likely generate incorrect results. In Lefebvre and Poulin's work [103], structural textures representing regular tiles and wood are generated by analyzing and extracting parameters from input images. In Dischler and Ghazafarpour's work [42], a hybrid approach, based on 2D texture analysis and geometric modeling, has been developed for synthesizing structural solid textures of certain types. Like Jagnow et al.'s method, both Lefebvre and Poulin's approach and Dischler and Ghazafarpour's approach involve extensive user interactions.

As discussed before, techniques in image-based solid texturing either are not fully automatic, which involve nontrivial user interactions [42, 86, 103], or may apply to only limited types of textures [81, 125, 188]. In Chapter 5, we present a new technique, called *aura 3D textures*, for synthesizing solid textures from input examples. Our method is

fully automatic and requires no user interaction in the process, and can generate faithful results over a wide range of textures.

2.3 Discussions

Although various techniques in texture analysis and synthesis have been proposed in the literature, textures are still far from being well understood. Because there is no known complete definition of texture, each existing texture model has its own advantages and disadvantages, and thus fails to model certain types of textures. For the same reason, existing texture synthesis techniques cannot determine whether or not the synthesized texture is a successful synthesis of the input texture; visual inspection is the only method to evaluate the synthesis results. The lack of accurate understanding in textures has also caused problems in 3D texture modeling. In fact, existing 3D texture models can apply to only limited types of textures. Therefore, it is crucial to understand and characterize textures with mathematical precision.

In this thesis, a new mathematical framework is presented to model textures with sufficient and necessary information using BGLAMs. The new framework provides important insight in texture modeling in both computer vision and computer graphics. Under a unified framework, new algorithms for 2D and 3D texture synthesis using BGLAMs are developed, respectively, and an original quantitative method for evaluating texture synthesis results is also presented. It has been shown that a wide range of textures can be faithfully modeled using BGLAMs. In addition, we demonstrate that BGLAMs can be used in texture image classification.

Chapter 3

Mathematical Framework of BGLAM

This Chapter presents the mathematical theory for BGLAMs (Basic Gray Level Aura Matrices). It is proved that a given texture image can be uniquely represented and reconstructed by its independent BGLAMs. A new BGLAM-based distance function is presented, and it is proved that the distance function is metric and one-to-one. New algorithms for 2D and 3D texture synthesis using BGLAMs are presented in Chapter 4 and 5, respectively. Chapter 6 presents a BGLAM-based method for texture image classification.

3.1 Background Knowledge

The concepts of aura sets, aura measures and aura matrices were first introduced by Elfadel and Picard in their work of analyzing and predicting texture patterns generated by MRF models in the states of equilibrium [52]. Interesting structures in texture images can be captured by gray level aura matrices (GLAMs) [137]. Research has been done in studying the behavior of the Gibbs texture models [52, 138, 139] using GLAMs, and the relationship between GLCMs and the Gibbs/Markov Random Fields [51, 137]. It has been shown that the aura matrix is a generalization of the cooccurrence matrix [31, 34, 51, 75].

Following the notations used in Section 2.1.1 in Chapter 2, an image $\boldsymbol{X}$ is modeled as a finite rectangular lattice of $m \times n$ sites $S = \{s = (i, j) \mid 0 \le i \le m-1, 0 \le j \le n-1\}$ with a neighborhood system $\boldsymbol{N} = \{N_s \mid s \in S\}$, where N_s is the neighborhood at site s. In conventional MRF models, the neighborhood system $\boldsymbol{N}$ has two important properties:

1) *Exclusive*: for any $s \in S$, $s \notin N_s$.

2) *Symmetric*: for any $s, t \in S$, $s \in N_t$ if and only if $t \in N_s$. (3.1)

Property 1) says that site s is excluded from its neighborhood and property 2) implies that the neighborhood is symmetric. In the next section, we relax the symmetric condition so that the neighborhood can be of any shape. The neighborhood N_s in a neighborhood system $\boldsymbol{N} = \{N_s \mid s \in S\}$ can be viewed as a translation of the basic neighborhood, which is called a *neighborhood structuring element* [51, 137], and is denoted by E.

The neighborhood, denoted by N_s^d, at site s is defined as the set of all nearby sites within a radius of d (d is a given integer), which is defined mathematically as:

$$N_s^d \overset{def}{=} N_{s=(i,j)} = \{r = (k,l) \in S \mid 0 < |(k-i)^2 + (l-j)^2 \le d\}. \quad (3.2)$$

Examples of the neighborhoods of the first, second, and fifth order are shown in Figure 3-1, where $s = '\bullet'$ is the target site and its neighboring sites are labeled by orders.

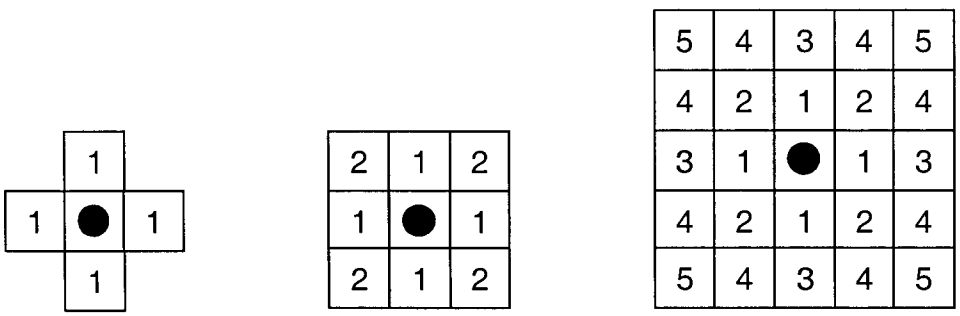
Definition 3-1 *Aura* [51]: Given two subsets $A, B \subseteq S$, the *aura* (i.e. *aura set*) of A with respect to B for neighborhood system $\boldsymbol{N} = \{N_s \mid s \in S\}$, denoted by $\vartheta_B(A, \boldsymbol{N})$ (or $\vartheta_B(A)$ if the neighborhood system $\boldsymbol{N}$ is clear in the context), is a set given by:

$$\vartheta_B(A) = \vartheta_B(A, \boldsymbol{N}) = \bigcup_{s \in A} (N_s \cap B), \quad (3.3)$$

Definition 3-2 *Aura Measure* [51]: With the same notations as in Definition 3-1, the *aura measure* of A with respect to B, denoted by $m(A,B)$, is given by:

$$m(A,B) = m(A,B,N) = \sum_{s\in A} |\, N_s \cap B \,|, \tag{3.4}$$

where for a given subset $A \subseteq S$, $|A|$ is the total number of elements in A.



(a) (b) (c)

Figure 3-1: Examples of neighborhoods. (a) The first order neighborhood with $d = 1$, (b) the second order neighborhood with $d = 2$, and (c) the fifth order neighborhood with $d = 8$, where the center site with a solid circle is the target site s, and its neighboring site are numbered by its order. The order of the neighborhood system is the largest order number of a neighboring site. For example, since the largest order number in the neighborhood shown in (c) is five, the order of the neighborhood in (c) is five.

Definition 3-3 The set $\{S_i \mid 0 \le i \le n-1\}$ is a partition of the lattice S if $S_i \cap S_j = \phi$ for $\forall i \ne j$, and $S = \bigcup_{i=0}^{n-1} S_i$.

Definition 3-4 *Aura Matrix* [51]: Let $\Im = \{S_i \mid 0 \le i \le n-1\}$ be a partition of the lattice S, then the aura matrix of $\Im$ over S, denoted by $A(\Im)$ (or simply A), is given by:

$$A = A(\Im) = [m(S_i, S_j)]_{0\le i,j\le n-1},$$

where $m(S_i, S_j)$ is the aura measure of S_i w.r.t. S_j given by Eq. 3.4, $0 \le i, j \le n-1$.

Of various types of partitions of S, the gray level sets are used in this thesis. For each site s in S, we assume that its gray level x_s is an integer between 0 and $G-1$ (G is the total number of gray levels for a pixel in the image), i.e. $x_s \in \Lambda = \{0,1,\ldots,G-1\}$, Λ is called the *state space* of s. The gray level sets of S are given by:

$$S_g = \{s \in S \mid x_s = g\}, \tag{3.5}$$

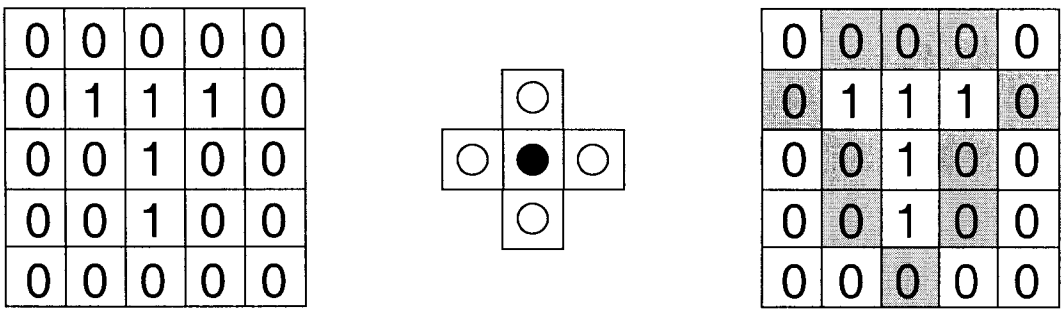
where $g = 0,1,\ldots,G-1$. Since $S = \bigcup_{g\in\Lambda} S_g$ and $S_i \cap S_j = \phi$, $\forall i \neq j$, $\{S_g \mid g \in \Lambda\}$ is a partition of S.

Definition 3-5: *Gray Level Aura Matrix* (*GLAM*) [51]: Given an image with rectangular lattice S and a neighborhood system $\boldsymbol{N}$, the aura matrix defined on the gray level sets $\{S_g \mid g \in \Lambda\}$ is called the *gray level aura matrix* of the image over $\boldsymbol{N}$.

The aura of A with respect to B characterizes how the subset B is present in the neighborhood of A. An example of an aura on a binary lattice with the four-nearest-neighbor neighborhood system is shown in Figure 3-2. The aura measure $m(A,B)$ measures the amount of B's sites presented in the neighborhood of A. It is noteworthy that $m(A,B)$ does not measure the number of elements in the aura of A with respect to B, i.e. in general, $m(A,B) \neq |\vartheta_B(A)|$. In the example shown in Figure 3-2, we have $m(A,B) = 12 \neq 10 = |\vartheta_B(A)|$. The GLAM of an image measures the amount of each gray level in the neighborhood of each other gray level. The GLAM for the binary image shown in Figure 3-2 (a) is

$$\boldsymbol{A} = \begin{bmatrix} 48 & 12 \\ 12 & 8 \end{bmatrix},$$

which is calculated using the structuring element of the four nearest-neighbor neighborhood system as shown in Figure 3-2 (b).

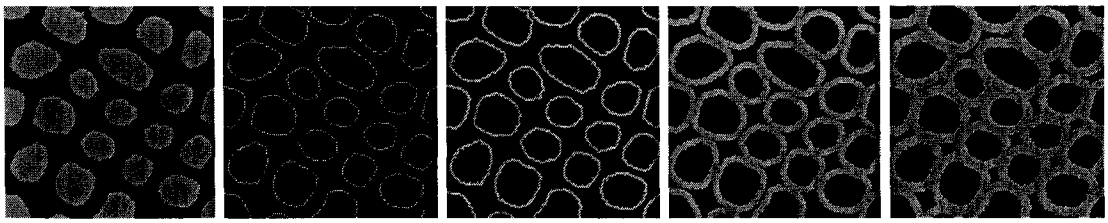


(a) (b) (c)

Figure 3-2: An example of aura on a binary lattice with the nearest four neighbors. (a) The sample binary lattice S, where the subset A is the set of all 1's and B is the set of all 0's. (b) The neighborhood structuring element of the four nearest neighbors (in circles) and the target pixel (in solid). (c) The set of shaded sites is the aura of A w.r.t. B.

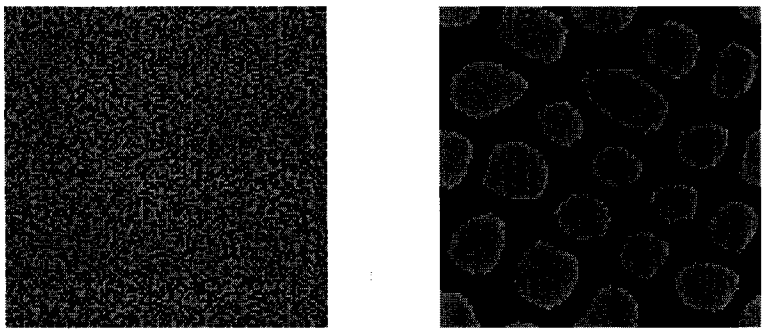
When handling a target pixel on the image boundaries, we only consider its neighboring pixels inside the image and discard those outside of the image. Definition 3-1 suggests that the aura depends on the size of the neighborhood system. Figure 3-3 gives an example of the auras calculated over neighborhood systems of different sizes.

The aura measure $m(A,B)$ can be used to characterize the interaction or the relationship between A and B in an image. Since the aura measure $m(A,B)$ calculates the amount of mixing sites between subset A and B, a large value (relative to the total number of sites in S) of $m(A,B)$ implies that subsets A and B are mixed together; while a small value implies that A and B are separate from each other, i.e. the region represented by each set is more likely to form its own cluster. Figure 3-4 gives an example to demonstrate this point.



A = grey region $d = 1$ $d = 5$ $d = 25$ $d = 50$
B = black region

Figure 3-3: An example of the aura of A with respect to B over neighborhood systems of different sizes. The grey pixels in the last four images are the aura of A w. r. t. B calculated over neighborhood systems of $d = 1, 5, 25$, and 50 (see Eq. 3.2), respectively.



(a) $m(A,B) = 16382$ (b) $m(A,B) = 1900$

Figure 3-4: An example of how the aura measure $m(A,B)$ interprets the relationship between A and B, where A is the set of all grey pixels and B the set of all black pixels, and N the four-nearest-neighbor neighborhood system. The size of both images is $128 \times 128 = 16384$. Compared with the total number of 16384 sites in S, $m(A,B) = 16382$ is large in (a) and $m(A,B) = 1900$ in (b) is small, which indicate that A and B mix together in (a), but separate from each other and form their own clusters in (b).

The GLAM of an image characterizes the probability distribution of each gray level in the neighborhood of each other gray level, and thus generalizes the GLCM (Gray

Level Cooccurrence Matrix). In fact, if the structuring element of a neighborhood system contains only two symmetric neighboring sites with respect to the target site, then a GLAM is a GLCM. For example, the GLCM shown in Figure 2-10 (b) is a special GLAM that can be calculated using a symmetric structuring element as shown in Figure 3-5. For illustration purpose, the definition of GLCM based on GLAM is given in Definition 3-6, which is consistent with the definition given in Section 2.1.5 in Chapter 2.

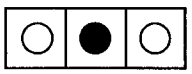


Figure 3-5: The structuring element used for calculating the GLCM in Figure 2-10 (b).

Definition 3-6: A GLCM is a GLAM computed from a neighborhood system whose structuring element contains only two symmetric neighboring sites with respect to the target site.

Some important mathematical properties of aura sets, aura measures, and aura matrices are summarized in the following three lemmas, and they are used in the rest of the chapter. For the proofs, the reader is referred to Elfadel and Picard's paper [51].

Lemma 3-1 *Aura Properties* [51]: Let S be a rectangular lattice and $N = \{N_s \mid s \in S\}$ is neighborhood system defined on S. Given subsets $A, B, C \subseteq S$, the following aura properties hold:

1) The aura of any set with respect to the empty set is empty, i.e. $\vartheta_{\phi}(A) = \phi$, $\forall A \subseteq S$.
2) The aura of A with respect to B is a subset of B, i.e. $\vartheta_B(A) \subseteq B$.
3) The aura of A with respect to itself (i.e. the self-aura of A) is a subset of itself, i.e. $\vartheta_A(A) \subseteq A$.

4) The aura of union is equal to the union of auras, i.e. $\vartheta_C(A \cup B) = \vartheta_C(A) \cup \vartheta_C(B)$.

5) The aura of intersection is included in the intersection of auras, i.e. $\vartheta_C(A \cap B) \subseteq \vartheta_C(A) \cap \vartheta_C(B)$.

6) If $A \subseteq B$, then $\vartheta_C(A) \subseteq \vartheta_C(B)$.

7) $\vartheta_{B \cup C}(A) = \vartheta_B(A) \cup \vartheta_C(A)$.

8) $\vartheta_{B \cap C}(A) = \vartheta_B(A) \cap \vartheta_C(A)$.

9) In general, the aura operation is not symmetric, i.e. $\vartheta_B(A) \neq \vartheta_A(B)$.

As a contribution of this thesis work, several new aura measure properties are given as properties 6) – 11) in Lemma 3-2, which are used in developing the BGLAM theory in the rest of the chapter. To my best knowledge, these new aura measure properties have not been discussed in the original aura framework [51] or elsewhere, and thus their proofs are given at the end of Lemma 3-2. The proofs of Properties 1) – 5) can be found in Elfadel and Picard's paper [51].

Lemma 3-2 *Aura Measure Properties*: Using the same notations as in Lemma 3-1, the following aura measure properties hold:

1) The function $m(.,.)$ is nonnegative, i.e. $m(A,B) \geq 0$, $\forall A, B \subseteq S$.

2) $m(.,B)$ is monotonic, i.e. $A_1 \subseteq A_2 \Rightarrow m(A_1,B) \leq m(A_2,B)$.

3) Similarly, $m(A,.)$ is monotonic, i.e. $B_1 \subseteq B_2 \Rightarrow m(A,B_1) \leq m(A,B_2)$.

4) $m(.,B)$ is subadditive, i.e. $m\left(\bigcup_{i=0}^{n-1} A_i, B\right) \leq \sum_{i=0}^{n-1} m(A_i,B)$, $\forall A_i \subseteq S$, $i = 0,1,...,n-1$.

5) $m(A,.)$ is subadditive, i.e. $m\left(A, \bigcup_{i=0}^{n-1} B_i\right) \le \sum_{i=0}^{n-1} m(A, B_i)$, $\forall B_i \subseteq S$, $i = 0,1,...,n-1$.

6) $m(A \cup B, C) = m(A, C) + m(B, C) - m(A \cap B, C)$. Moreover, if $A \cap B = \phi$, then $m(A \cup B, C) = m(A, C) + m(B, C)$.

7) $m(A, B \cup C) = m(A, B) + m(A, C) - m(A, B \cap C)$. Moreover, if $B \cap C = \phi$, then $m(A, B \cup C) = m(A, B) + m(A, C)$.

8) $m(A - B, C) = m(A, C) - m(A \cap B, C)$.

9) $m(C, A - B) = m(C, A) - m(C, A \cap B)$.

10) If $\{A_i \mid 0 \le i \le n-1\}$ is pair-wise disjoint (i.e. $A_i \cap A_j = \phi$, $\forall i \ne j$), then $m(.,B)$ is additive, i.e. $m\left(\bigcup_{i=0}^{n-1} A_i, B\right) = \sum_{i=0}^{n-1} m(A_i, B)$.

11) If $\{B_i \mid 0 \le i \le n-1\}$ is pair-wise disjoint (i.e. $B_i \cap B_j = \phi$, $\forall i \ne j$), then $m(A,.)$ is additive, i.e. $m\left(A, \bigcup_{i=0}^{n-1} B_i\right) = \sum_{i=0}^{n-1} m(A, B_i)$.

Proof: [Properties 6) – 11)] For the proof of property 6), by the definition of aura measure, we have:

$$\begin{aligned} m(A \cup B, C) &= \sum_{s \in A \cup B} |N_s \cap C| = \sum_{s \in A} |N_s \cap C| + \sum_{s \in B} |N_s \cap C| - \sum_{s \in A \cap B} |N_s \cap C| \\ &= m(A, C) + m(B, C) - m(A \cap B, C) \end{aligned}.$$

In the above equation, the term after the "-" symbol is used to compensate the double counting effect caused by the two added terms on both sides of the "+" symbol. Moreover, if $A \cap B = \phi$, then $m(A \cap B, C) = m(\phi, C) = 0$, which implies that $m(A \cup B, C) = m(A, C) + m(B, C)$. The proof of property 7) is similar to that of 6).

To prove property 8), any given set $A \subseteq S$ has the following partition: $A = (A - B) \cup (A \cap B)$ with $(A - B) \cap (A \cap B) = \phi$, $\forall B \subseteq S$ [82]. By using property 6), we have:

$$m(A, C) = m((A - B) \cup (A \cap B), C) = m(A - B, C) + m(A \cap B, C).$$

The above equation implies that $m(A - B, C) = m(A, C) - m(A \cap B, C)$. Using property 7), property 9) can be proved in a similar way.

Based on properties 6) and 7), properties 10) and 11) can be proved by using Mathematical Induction [68]. ■

Lemma 3-3 *Aura Matrix Properties* [51]: Let $\Im = \{S_i \mid 0 \le i \le n-1\}$ be a partition of the lattice S, $\boldsymbol{A} = [a_{ij}]_{0 \le i, j \le n-1}$ be the aura matrix of $\Im$ over S as defined in Definition 3-4, and E be the neighborhood structuring element. Assume periodic boundary conditions [51] are used for handling pixels on image boundaries, then, the following aura matrix properties hold:

1) The sum of each row satisfies $\sum_{j=0}^{n-1} a_{ij} = |E| * |S_i|$, $i = 0,1,...,n-1$.

2) The sum of each column satisfies $\sum_{i=0}^{n-1} a_{ij} = |E| * |S_j|$, $j = 0,1,...,n-1$.

3) If E is symmetric, then $\boldsymbol{A}$ is symmetric, and $\sum_{j=0}^{n-1} a_{kj} = \sum_{i=0}^{n-1} a_{ik} = |E| * |S_k|$, $k = 0,1,...,n-1$.

3.2 BGLAM Concepts

In the previous studies on GLAMs [51, 52, 137, 138, 152], the neighborhood system is assumed to be *symmetric*, i.e. for any $s, t \in S$, $s \in N_t$ if and only if $t \in N_s$. To describe BGLAMs, we need asymmetric neighborhood systems, whose definition is given below. This section gives the definition of BGLAMs (basic gray level aura matrices).

Definition 3-7 Let $N = \{N_s \mid s \in S\}$ be a neighborhood system on lattice S. N is *asymmetric* if its neighborhood structuring element E is not symmetric (see Eq. 3.1 for the definition of a symmetric neighborhood).

Definition 3-8 Using the same notations as in Definition 3-7, let $r, r', s \in S$, r and r' are symmetric to each other w.r.t. s if $r - s = -(r' - s)$ (i.e. $r' = 2 * s - r$).

In the above definition, sites $r, r', s \in S$ are considered as points of two coordinates in 2D space, and $r - s$ and $r' - s$ are vectors in the plane. Figure 3-6 gives an explanation of the relationship between the two vectors $r - s$ and $r' - s$, which are represented as two arrows in the figure with the same length but pointing in opposite directions (i.e. $r - s = -(r' - s)$).

Definition 3-9 A neighborhood system $N = \{N_s \mid s \in S\}$ is *completely asymmetric* if N is asymmetric and for any $r \in N_s$, its symmetric site r' with respect to s is not in N_s, $\forall s \in S$.

Definition 3-10 For any neighborhood N_s (symmetric or asymmetric) at site $s \in S$, its *complement neighborhood* with respect to s, denoted by $\overline{N}_s$, is defined as the

set of the symmetric sites of the sites in N_s with respect to s that are not in N_s, i.e. $\overline{N}_s$ is given as:

$$\overline{N}_s = \{r' \in S \mid r' = 2 * s - r, r' \notin N_r, r \in N_r\} \qquad (3.6)$$

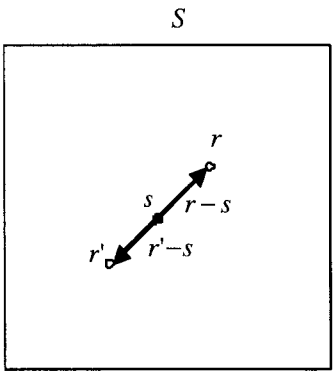


Figure 3-6: An explanation of symmetric sites in Definition 3-7.

Definition 3-11 Given an arbitrary neighborhood system N over S with a structuring element E, its single site neighborhood system decomposition is a set of single site neighborhood systems defined over E as: $\{N_r, r \in E\}$, where $N_r = \{N_s^r, s \in S\}$ and N_s^r is the single site neighborhood of s that contains r.

Definition 3-12 *Basic Gray Level Aura Matrix* (*BGLAM*): Given a lattice system S, a *basic GLAM* (*BGLAM*) on S is a *GLAM* computed from a single site neighborhood system, i.e. a neighborhood system whose structuring element contains only a single neighboring site.

Figure 3-7 gives examples of asymmetric, completely asymmetric and symmetric neighborhood structures, where (a)-(e) are asymmetric, and (f) is symmetric. The

neighborhoods in (a)-(d) are completely asymmetric, while the neighborhood in (e) is asymmetric but not completely asymmetric since the symmetric site r' of r with respect to s is contained in the neighborhood. The neighborhood pairs (a) and (b) as well as (c) and (d) are complement of each other. The complement of a symmetric neighborhood is empty, e.g. the complement neighborhood of (f) in Figure 3-7 is empty. The union of a neighborhood and its complement is a symmetric neighborhood. Neighborhoods (a) and (b) are single site neighborhoods, the GLAMs computed by them are BGLAMs.

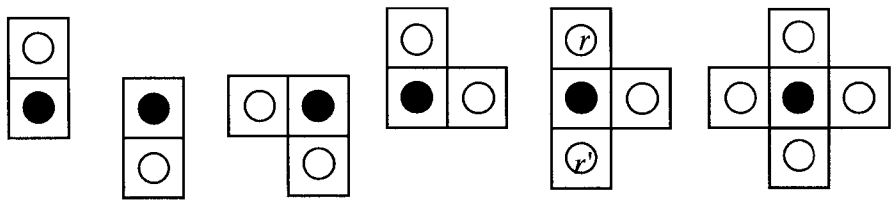


Figure 3-7: Examples of asymmetric, symmetric, and complement neighborhoods. The first four, (a) – (d), are completely asymmetric, the neighborhood in (e) is asymmetric but not completely asymmetric since the symmetric site r' of r is in the neighborhood, and the last, (f), is symmetric, where $s = '\bullet'$ is the target pixel in N_s and $r = '\circ'$ is a neighboring pixel of s. The complement neighborhoods of (a) and (c) are (b) and (d), respectively, and the complement neighborhood of (f) is empty.

In the rest of the thesis, SGLAM stands for a symmetric GLAM commonly used in existing techniques [51, 52, 137, 138, 152], GLAM for a gray level aura matrix computed using arbitrary (i.e. either symmetric or asymmetric) neighborhood systems, and BGLAM for a basic GLAM.

3.2 BGLAM Theory

Textures cannot be effectively differentiated using SGLAMs. Two images with different textures may have the same SGLAMs (see Figure 3-8). In this section, we present the BGLAM theory and prove that BGLAMs can give the necessary and sufficient information to differentiate between images.

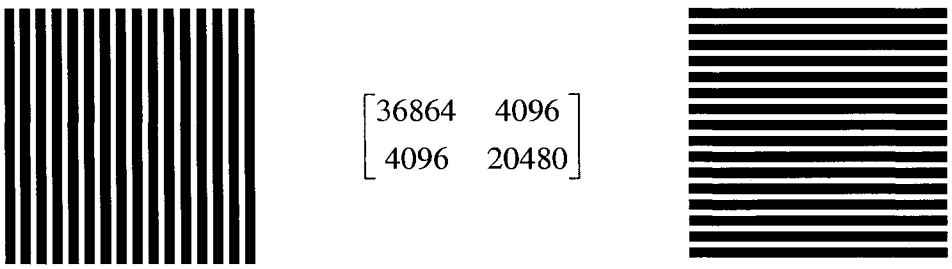


Figure 3-8: An example of the inefficiency of SGLAMs for differentiating textures. The right stripe-texture image is a rotation of the left image by 90 degrees, and both images have the same SGLAM that is shown in the middle. Both images (size 128x128) are binary, and the four-nearest-neighbor neighborhood system is used to compute the SGLAM.

Lemma 3-4 Let S be the image lattice, $N = \{N_s \mid s \in S\}$ be an arbitrary neighborhood system, and $\{S_i \mid 0 \le i \le n-1\}$ be a partition of S. For any $s \in S$, let $\{N_s^i \mid 0 \le i \le m-1\}$ be a partition of the neighborhood N_s, and $\boldsymbol{A}$ and $\boldsymbol{A}_i$ be the aura matrix computed from N and $N_i = \{N_s^i \mid s \in S\}$, respectively, $i = 0,1,...,m-1$. Then, we have $\boldsymbol{A} = \sum_{i=0}^{m-1} \boldsymbol{A}_i$.

Proof: By the definition of aura matrix (Definition 3-4), we have $A=[m(S_i,S_j,\boldsymbol{N})]_{0\le i,j\le n-1}$. Since $\{N_s^i \mid 0\le i\le m-1\}$ is a partition of N_s for any $s\in S$, by Definition 3-3 we have $N_s=\bigcup_{i=0}^{m-1} N_s^i$ with $N_s^i\cap N_s^j=\phi$ for $\forall i\neq j$, which implies:

$$\begin{aligned} m(S_i,S_j,\boldsymbol{N}) &= \sum_{s\in S_i}|N_s\cap S_j|=\sum_{s\in S_i}|(\bigcup_{k=0}^{m-1}N_s^k)\cap S_j|=\sum_{s\in S_i}|\bigcup_{k=0}^{m-1}(N_s^k\cap S_j)| \\ &=\sum_{s\in S_i}\sum_{k=0}^{m-1}|N_s^k\cap S_j|=\sum_{k=0}^{m-1}\sum_{s\in S_i}|N_s^k\cap S_j| \end{aligned} \quad .(3.7)$$

On the other hand, for any $k=0,1,...,m-1$, we have $\boldsymbol{A}_k=[m(S_i,S_j,\boldsymbol{N}_k)]_{0\le i,j\le n-1}$, where

$$m(S_i,S_j,\boldsymbol{N}_k)=\sum_{s\in S_i}|N_s^k\cap S_j|\,. \qquad (3.8)$$

Plug Eq. 3.8 into Eq. 3.7, we have $m(S_i,S_j,\boldsymbol{N})=\sum_{k=0}^{m-1}m(S_i,S_j,\boldsymbol{N}_k)$, which implies $\boldsymbol{A}=\sum_{i=0}^{m-1}\boldsymbol{A}_i$.

In Eq. 3.7, from step 2 to step 3 the set property: $(\bigcup_{i=0}^{m-1}A_i)\cap B=\bigcup_{i=0}^{m-1}(A_i\cap B)$ for any A_i and B, is used, and from step 3 to step 4 the property of a pair-wise disjoint sets: $|\bigcup_{i=0}^{m-1}A_i|=\sum_{i=0}^{m-1}|A_i|$, where $A_i\cap A_j=\phi$, $\forall i\neq j$, is used [82]. ■

Theorem 3-1 Any GLAM can be represented as a sum of BGLAMs; any GLCM can be represented as a sum of two BGLAMs.

Proof: Let S be the image lattice, $\boldsymbol{A}$ be a GLAM computed from an arbitrary neighborhood system $\boldsymbol{N}=\{N_s \mid s\in S\}$ with a neighborhood structuring element E,

$\{N_r \mid r \in E\}$ be its single site neighborhood decomposition of N as defined in Definition 3-11, and A_r be the gray level aura matrix computed from the neighborhood system N_r, $\forall r \in E$. By Lemma 3-4, we have $A = \sum_{r \in N} A_r$. Since N_r is a single site neighborhood system, each A_r is a BGLAM. Therefore, we have proved that each GLAM can be represented as a sum of BGLAMs. By Definition 3-6, a GLCM is a GLAM defined over a neighborhood system whose structuring element E contains only two symmetric neighboring sites w.r.t. the target site. Therefore, a GLCM can be represented a as a sum of two BGLAMs. ■

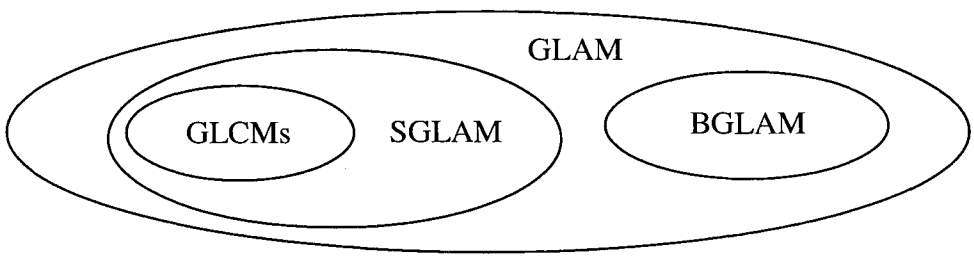


Figure 3-9: The relationship between the set of all GLCMs, the set of all SGLAMs, the set of all GLAMs and the set of all BGLAMs, where a smaller oval represents a subset of the bigger oval to which it belongs.

Theorem 3-1 indicates that BGLAMs can serve as a basis of GLAMs. In fact, by the technique of ICA (Independent Component Analysis) [83], a set of independent BGLAMs can be efficiently identified, and used as the basis of GLAMs. Although independent GLCMs form the basis of SGLAMs, they cannot form the basis of GLAMs. The relationship between the set of all GLCMs, the set of all SGLAMs, the set of GLAMs, and the set of all BGLAMs is shown in Figure 3-9.

Lemma 3-5 As shown in Figure 3-10, let $\boldsymbol{X}_1$ be an image defined on lattice S with a single site neighborhood system $\boldsymbol{N} = \{N_s \mid s \in S\}$, and p is a given site in S. Suppose image $\boldsymbol{X}_2$ is obtained from image $\boldsymbol{X}_1$ by changing p's intensity value from g_1 to g_2 and $g_1 \neq g_2$ (all other pixels' intensity values remain unchanged). Let $\boldsymbol{A}_1 = [a_1(i,j)]_{0 \le i,j \le G-1}$ and $\boldsymbol{A}_2 = [a_2(i,j)]_{0 \le i,j \le G-1}$ be the GLAMs of $\boldsymbol{X}_1$ and $\boldsymbol{X}_2$ over $\boldsymbol{N}$, respectively, where G is the total number of gray levels of a pixel in images. Let $N_p = \{r\}$ and $\overline{N}_p = \{r'\}$ be the neighborhood and the complement neighborhood at p, and let $g = \boldsymbol{X}_1(r)$ and $g' = \boldsymbol{X}_1(r')$ be the gray levels of r and r', respectively, in $\boldsymbol{X}_1$. Then one of the following relationships between $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ holds:

1) If $(g', g) = (g_1, g_1)$ (i.e. $g' = g_1$ and $g = g_1$), then

$$a_2(g_1, g_1) = a_1(g_1, g_1) - 2,$$

$$a_2(g_2, g_1) = a_1(g_2, g_1) + 1,$$

$$a_2(g_1, g_2) = a_1(g_1, g_2) + 1,$$

$$a_2(i,j) = a_1(i,j), \ \forall (i,j) \notin \{(g_1,g_1),(g_2,g_1),(g_1,g_2)\}.$$

2) If $(g', g) = (g_1, g_2)$ or $(g', g) = (g_2, g_1)$, then

$$a_2(g_1, g_1) = a_1(g_1, g_1) - 1,$$

$$a_2(g_2, g_2) = a_1(g_2, g_2) + 1,$$

$$a_2(i,j) = a_1(i,j), \ \forall (i,j) \notin \{(g_1,g_1),(g_2,g_2)\}.$$

3) If $(g', g) = (g_2, g_2)$, then

$$a_2(g_2, g_2) = a_1(g_2, g_2) + 2,$$

$$a_2(g_2, g_1) = a_1(g_2, g_1) - 1,$$

$$a_2(g_1,g_2) = a_1(g_1,g_2) - 1,$$

$$a_2(i,j) = a_1(i,j),\ \forall (i,j) \notin \{(g_2,g_2),(g_2,g_1),(g_1,g_2)\}$$

4) If $(g',g) \notin \{(g_1,g_1),(g_1,g_2),(g_2,g_1),(g_2,g_2)\}$, then

$$a_2(g_1,g) = a_1(g_1,g) - 1,\ a_2(g_2,g) = a_1(g_2,g) + 1,$$

$$a_2(g',g_1) = a_1(g',g_1) - 1,\ a_2(g',g_2) = a_1(g',g_2) + 1,$$

$$a_2(i,j) = a_1(i,j),\ \forall (i,j) \notin \{(g_1,g),(g_2,g),(g',g_1),(g',g_2)\}.$$

Therefore, in any of the above four cases, we have $\boldsymbol{A}_2 \neq \boldsymbol{A}_1$.

Proof: We only give the proof of 1), and the proofs of 2) - 4) are similar to that of 1). It is also easy to check that in any of the four cases 1) - 4), $\boldsymbol{A}_2 \neq \boldsymbol{A}_1$.

We prove by the definition of aura matrices (Definition 3-4) and the aura measure properties 8) - 9) proved in Lemma 3-2, which are restated as follows for easy reading:

$$m(A-B,C) = m(A,C) - m(A \cap B,C)$$
$$m(C,A-B) = m(C,A) - m(C,A \cap B)$$

Let $S_g^{X_i} = \{s \in S \mid \boldsymbol{X}_i(s) = g\}$, $i = 1,2$, and $g \in \Lambda = \{0,1,...,G-1\}$, by using the above two properties, if $(g',g) = (g_1,g_1)$ (see Figure 3-10) then we have:

$$\begin{aligned} a_2(g_1,g_1) &= m(S_{g_1}^{X_2}, S_{g_1}^{X_2}) = m(S_{g_1}^{X_1} - \{p\}, S_{g_1}^{X_1} - \{p\}) \\ &= m(S_{g_1}^{X_1}, S_{g_1}^{X_1}) - m(S_{g_1}^{X_1}, \{p\}) - m(\{p\}, S_{g_1}^{X_1}) + m(\{p\},\{p\}) \\ &= m(S_{g_1}^{X_1}, S_{g_1}^{X_1}) - 1 - 1 + 0 \\ &= m(S_{g_1}^{X_1}, S_{g_1}^{X_1}) - 2 \\ &= a_1(g_1,g_1) - 2 \end{aligned},$$

$$\begin{aligned} a_2(g_2,g_1) &= m(S_{g_2}^{X_2}, S_{g_1}^{X_2}) = m(S_{g_2}^{X_1} \cup \{p\}, S_{g_1}^{X_1} - \{p\}) \\ &= m(S_{g_2}^{X_1}, S_{g_1}^{X_1}) - m(S_{g_2}^{X_1}, \{p\}) + m(\{p\}, S_{g_1}^{X_1}) - m(\{p\},\{p\}) \\ &= m(S_{g_2}^{X_1}, S_{g_1}^{X_1}) - 0 + 1 - 0 \\ &= m(S_{g_2}^{X_1}, S_{g_1}^{X_1}) + 1 = a_1(g_2,g_1) + 1 \end{aligned},$$

$$
\begin{aligned}
a_2(g_1,g_2) &= m(S_{g_1}^{X_2}, S_{g_2}^{X_2}) = m(S_{g_1}^{X_1} - \{p\}, S_{g_2}^{X_1} \cup \{p\}) \\
&= m(S_{g_1}^{X_1}, S_{g_2}^{X_1}) + m(S_{g_1}^{X_1}, \{p\}) - m(\{p\}, S_{g_2}^{X_1}) - m(\{p\},\{p\}) \\
&= m(S_{g_1}^{X_1}, S_{g_2}^{X_1}) + 1 - 0 - 0 \\
&= m(S_{g_1}^{X_1}, S_{g_2}^{X_1}) + 1 \\
&= a_1(g_1,g_2) + 1
\end{aligned}
$$

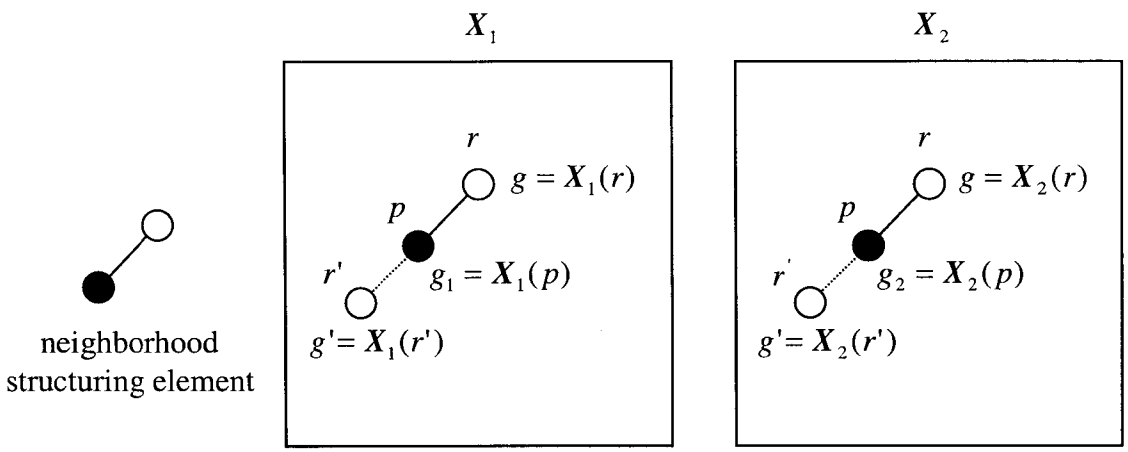


Figure 3-10: An illustration of Lemma 3-5.

To prove $a_2(i,j) = a_1(i,j)$, $\forall (i,j) \notin \{(g_1,g_1),(g_2,g_1),(g_1,g_2)\}$, we use the following relationship between $S_k^{X_1}$ and $S_k^{X_2}$:

$$
S_k^{X_2} = \begin{cases} S_k^{X_1}, & k \neq g_2, k \neq g_1 \\ S_k^{X_1} - \{p\}, & k = g_1 \\ S_k^{X_1} \cup \{p\}, & k = g_2 \end{cases}, \quad \forall k \in \Lambda = \{0,1,\ldots,G-1\}. \qquad (3.9)
$$

For i, there are two cases: $i = g_2$ and $i \neq g_2$. We prove for $i = g_2$. The proof for $i \neq g_2$ is similar. Assume the first case, i.e. $i = g_2$, by $(i,j) \notin \{(g_1,g_1),(g_2,g_1),(g_1,g_2)\}$, we have $j \neq g_1$; otherwise $(i,j) = (g_2,g_1)$, which is a contradiction. For j, we have either $j = g_2$ or $j \neq g_2$.

If $j = g_2$, then $i = g_2$ and $j = g_2$, applying Eq. 3.9 for $k = i, j$, respectively, we have:

$$\begin{aligned} a_2(i,j) &= m(S_i^{X_2}, S_j^{X_2}) = m(S_i^{X_1} \cup \{p\}, S_j^{X_1} \cup \{p\}) \\ &= m(S_i^{X_1}, S_j^{X_1}) + m(S_i^{X_1}, \{p\}) + m(\{p\}, S_j^{X_1}) + m(\{p\}, \{p\}) \\ &= m(S_i^{X_1}, S_j^{X_1}) + 0 + 0 + 0 = m(S_i^{X_1}, S_j^{X_1}) \\ &= a_1(i,j) \end{aligned} \quad . (3.10)$$

In the proof of Eq. 3.10, since $(i,j) = (g_2, g_2)$, $(g', g) = (g_1, g_1)$, $g = \boldsymbol{X}_1(r)$, and $g' = \boldsymbol{X}_1(r')$, we have $m(S_i^{X_1}, \{p\}) = m(S_{g_2}^{X_1}, \{p\}) = 0$, $m(\{p\}, S_j^{X_1}) = m(\{p\}, S_{g_2}^{X_1}) = 0$, and $m(\{p\}, \{p\}) = 0$.

If $j \neq g_2$, then $i = g_2$, and $j \neq g_2$ and $j \neq g_1$, again applying Eq. 3.9 for $k = i, j$, respectively, we have:

$$\begin{aligned} a_2(i,j) &= m(S_i^{X_2}, S_j^{X_2}) = m(S_i^{X_1} \cup \{p\}, S_j^{X_1}) \\ &= m(S_i^{X_1}, S_j^{X_1}) + m(\{p\}, S_j^{X_1}) \\ &= m(S_i^{X_1}, S_j^{X_1}) + 0 \\ &= m(S_i^{X_1}, S_j^{X_1}) \\ &= a_1(i,j) \end{aligned}$$

Now, we have proved that $a_2(i,j) = a_1(i,j)$, $\forall (i,j) \notin \{(g_1, g_1), (g_2, g_1), (g_1, g_2)\}$. ■

From the above lemma, we can see that when the gray value of a pixel p in an image changes, only a few elements of the GLAM of the image will be affected. In fact, under the conditions of Lemma 3-5, $\boldsymbol{A}_2$ can be calculated from $\boldsymbol{A}_1$ in the following two steps:

1) Initialize $\boldsymbol{A}_2$ as $\boldsymbol{A}_1$, i.e. $\boldsymbol{A}_2 \leftarrow \boldsymbol{A}_1$.

2) Update $\boldsymbol{A}_2$:

$$\begin{matrix} a_2(g_1,g) \leftarrow a_2(g_1,g)-1,\ a_2(g_2,g) \leftarrow a_2(g_2,g)+1 \\ a_2(g',g_1) \leftarrow a_2(g',g_1)-1,\ a_2(g',g_2) \leftarrow a_2(g',g_2)+1 \end{matrix} . \qquad (3.11)$$

There are four cases in Lemma 3-5 to be analyzed. We derive the first case, the other three cases can be derived similarly. The first case states that if $(g',g)=(g_1,g_1)$ (i.e. $g'=g_1$ and $g=g_1$), then $\boldsymbol{A}_2$ differs from $\boldsymbol{A}_1$ only by the following three matrix elements (all other corresponding elements are the same):

$$\begin{aligned} a_2(g_1,g_1) &= a_1(g_1,g_1)-2 \\ a_2(g_2,g_1) &= a_1(g_2,g_1)+1, \\ a_2(g_1,g_2) &= a_1(g_1,g_2)+1 \end{aligned}$$

Plug $g'=g_1$ and $g=g_1$ into Eq 3.11, $\boldsymbol{A}_2$ can be updated from its previous version by the following four steps:

$$\begin{matrix} a_2(g_1,g_1) \leftarrow a_2(g_1,g_1)-1 \\ a_2(g_2,g_1) \leftarrow a_2(g_2,g_1)+1 \\ a_2(g_1,g_1) \leftarrow a_2(g_1,g_1)-1 \\ a_2(g_1,g_2) \leftarrow a_2(g_1,g_2)+1 \end{matrix} ,$$

Since $\boldsymbol{A}_2$ is set as $\boldsymbol{A}_1$ initially, after the above four update steps (step 1 and 3 are the same, which implies that $a_2(g_1,g_1)=a_1(g_1,g_1)-2$), $\boldsymbol{A}_2$ differs from $\boldsymbol{A}_1$ only by the same three matrix elements as those for the first case of Lemma 3-5.

In Lemma 3-5, a single site neighborhood system is assumed. For an arbitrary neighborhood system, Algorithm 3-1, which is given below, proves that when a pixel's gray level changes, the GLAM of the image can be updated by processing only the pixels in both the neighborhood and the complement neighborhood at pixel p by iteratively using Eq. 3.11. Precisely, we have the following algorithm to do this.

Algorithm 3-1 Let X_1 be an image defined on lattice S with an arbitrary neighborhood system $N = \{N_s \mid s \in S\}$, and p is a given site in S. Suppose image X_2 is obtained from image X_1 by only changing p's intensity value from g_1 to g_2. Let $A_1 = [a_1(i,j)]_{0 \le i,j \le G-1}$ and $A_2 = [a_2(i,j)]_{0 \le i,j \le G-1}$ are the GLAMs of X_1 and X_2 over N, respectively (G is the total number of gray levels of a pixel in an image). Then A_2 can be obtained from A_1 by performing the following steps:

1) Initialize A_2 as A_1, i.e. $A_2 \leftarrow A_1$.

2) For each $r \in N_p$, do

   2.1) $a_2(g_1, g) \leftarrow a_2(g_1, g) - 1$
   2.2) $a_2(g_2, g) \leftarrow a_2(g_2, g) + 1$

   where $g = X_1(r) = X_2(r)$ is the gray level value of r.

3) For each $r' \in \overline{N}_r$, do

   3.1) $a_2(g', g_1) \leftarrow a_2(g', g_1) - 1$
   3.2) $a_2(g', g_2) \leftarrow a_2(g', g_2) + 1$

   where $\overline{N}_r$ is the complement neighborhood of N_p, and $g' = X_1(r') = X_2(r')$ is the gray level value of r'.

Lemma 3-6 Two images are identical if and only if their corresponding GLAMs on all possible neighborhood systems are the same.

Proof: Suppose that two texture images X_1 and X_2 are defined on lattice S, and that they are identical, i.e. $X_1(s) = X_2(s)$ for any $s \in S$. It is obvious that their corresponding GLAMs on all possible neighborhood systems are the same.

Suppose the corresponding GLAMs of X_1 and X_2 on all possible neighborhood systems are the same, we have to show $X_1 = X_2$. This is equivalent to proving that if $X_1 \neq X_2$ then there must exist neighborhood system N such that the corresponding GLAMs A_1 and A_2 of X_1 and X_2 over N are not equal (i.e. $A_1 \neq A_2$).

Assume that $X_1 \neq X_2$. Let $S = E \cup D$ be a partition on S, where $E = \{s \in S \mid X_1(s) = X_2(s)\}$ is the region in which each site has the same gray level in X_1 and X_2, and $D = \{s \in S \mid X_1(s) \neq X_2(s)\}$ is the region in which each site has different gray level in X_1 and X_2. But $X_1 \neq X_2$, D is not empty, i.e. $D \neq \phi$. Let $|D| = n$ and $D = \{r_i \in S \mid 0 \leq i \leq n-1\}$, and choose a neighborhood system N such that its neighborhood structuring element E is large enough to contain D, i.e. $E \supseteq D$. Let A_1 and A_2 be the GLAMs of X_1 and X_2 over N, respectively. Using Lemma 3-5 and mathematical induction, one can prove that $A_1 \neq A_2$ since A_2 can be obtained from A_1 by iteratively applying the updating process on each site in D as described in Algorithm 3-1. ■

The proof of Lemma 3-6 indicates that two images can be differentiated by their corresponding GLAMs over a specific neighborhood system (either symmetric or asymmetric). In the worst case, the structuring element of the neighborhood system could be as large as S. It is impractical to test all possible neighborhood systems on S since the number of possible neighborhood systems is $2^{|S|} - 1$. However, by Theorem 1, any

GLAM can be represented as a sum of BGLAMs. The following theorem indicates that two images can be differentiated by their corresponding BGLAMs, by which the computation cost can be significantly reduced from $O(2^{|S|})$ to $O(|S|)$.

Theorem 3-2 Two images are identical if and only if their corresponding BGLAMs calculated by the largest neighborhood system are the same.

Proof: Suppose X_1 and X_2 are two images defined on lattice *S*, we only have to prove that if $X_1 \neq X_2$ then there exists a single site neighborhood system N_r such that $A_1 \neq A_2$, where A_1 and A_2 are the GLAMs of X_1 and X_2 over N_r, respectively. A_1 and A_2 are BGLAMs because they are calculated from a single site neighborhood system.

Suppose $X_1 \neq X_2$, then by Lemma 3-6, there exists a neighborhood system N such that their corresponding GLAMs $A_1(N)$ and $A_2(N)$ are not equal. Let $\{N_r \mid r \in E\}$ (where E is the structuring element of N) be the single site neighborhood system decomposition of N (see Definition 3-11), then by Lemma 3-4, we have:

$$A_1(N) = A_1(\bigcup_{r \in N} N_r) = \sum_{r \in N} A_1(N_r),\ A_2(N) = A_2(\bigcup_{r \in N} N_r) = \sum_{r \in N} A_2(N_r). \qquad (3.12)$$

Since $A_1(N) \neq A_2(N)$, there exist $r \in N$ such that BGLAM $A_1(N_r) \neq A_2(N_r)$ because otherwise we have $A_1(N) = A_2(N)$ by Eq. 3.12, which contradicts the assumption. ■

From Theorem 3-1 and Theorem 3-2, we conclude that an image can be uniquely represented by its BGLAMs, but not by SGLAMs nor by GLCMs. In other words, the information captured in both SGLAMs and GLCMs are less precise. In the next chapter,

we demonstrate that BGLAMs outperform both SGLAMs and GLCMs in 2D texture synthesis as well as in evaluating the 2D texture synthesis results.

3.4 BGLAM Distance Measure

Definition 3-13 A distance function d over a pair of images is *metric* if for any image X, Y, and Z, d satisfies the following three properties [160]:

1) Non-negativity: $d(X,Y) \geq 0$.

2) Symmetry: $d(X,Y) = d(Y,X)$.

3) Triangle inequality: $d(X,Y) \leq d(X,Z) + d(Z,Y)$.

Definition 3-14 A distance function d over a pair of images is *strong metric* if it is metric, and for any images X and Y, d satisfies:

4) One-to-one: $d(X,Y) = 0$ if and only if $X = Y$.

Definition 3-15 For a given aura matrix $\boldsymbol{A} = [a(i,j)]_{0 \leq i,j \leq n-1}$, it is *normalized* if $\sum_{i,j=0}^{n-1} a(i,j) = 1$. In the definition an absolute sign on a_{ij} is not necessary because a_{ij} is never negative for an aura matrix.

Definition 3-16 Given two images X and Y defined on a rectangular lattice S. Let $A(X) = \{A_k(X) \mid 0 \leq k \leq m-1\}$ and $A(Y) = \{A_k(Y) \mid 0 \leq k \leq m-1\}$ be their corresponding normalized BGLAMs, then the BGLAM distance measure between X and Y is given by:

$$d(X,Y) = d(A(X), A(Y)) = \frac{1}{m} \sum_{k=0}^{m-1} \| A_k(X) - A_k(Y) \|, \qquad (3.13)$$

where for a matrix $A = [a(i,j)]_{0 \le i,j \le n-1}$, $\| A \| = \sum_{i,j=0}^{n-1} | a(i,j) |$.

Theorem 3-3 The distance function defined in Eq. 3.13 is strong metric, i.e. for any images X, Y, and Z defined on S, $d(.,.)$ satisfies the following four properties:

1) Non-negativity: $d(X,Y) \ge 0$.

2) Symmetry: $d(X,Y) = d(Y,X)$.

3) Triangle inequality: $d(X,Y) \le d(X,Z) + d(Z,Y)$.

4) One-to-one: $d(X,Y) = 0$ if and only if $X = Y$.

Proof: It is easy to check that $d(.,.)$ satisfies properties of 1) and 2). For the proof of 3), by noting that $|a+b| \le |a| + |b|$ for any real numbers a and b, we have:

$$\begin{aligned} d(X,Y) &= \frac{1}{m}\sum_{k=0}^{m-1} \| A_k(X) - A_k(Y) \| = \frac{1}{m}\sum_{k=0}^{m-1}\sum_{i,j=0}^{n-1} \left| a_k^X(i,j) - a_k^Y(i,j) \right| \\ &= \frac{1}{m}\sum_{k=0}^{m-1}\sum_{i,j=0}^{n-1} \left| \left[a_k^X(i,j) - a_k^Z(i,j)\right] + \left[a_k^Z(i,j) - a_k^Y(i,j)\right] \right| \\ &\le \frac{1}{m}\sum_{k=0}^{m-1}\sum_{i,j=0}^{n-1} \left| a_k^X(i,j) - a_k^Z(i,j) \right| + \frac{1}{m}\sum_{k=0}^{m-1}\sum_{i,j=0}^{n-1} \left| a_k^Z(i,j) - a_k^Y(i,j) \right| \\ &= d(X,Z) + d(Z,Y) \end{aligned},$$

where $A(X) = \{A_k(X) \mid 0 \le k \le m-1\}$, $A(Y) = \{A_k(Y) \mid 0 \le k \le m-1\}$, and $A(Z) = \{A_k(Z) \mid 0 \le k \le m-1\}$ are the normalized BGLAMs of X, Y, and Z, respectively; and $A_k(X) = [a_k^X(i,j)]_{0 \le i,j \le n-1}$, $A_k(Y) = [a_k^Y(i,j)]_{0 \le i,j \le n-1}$, and $A_k(Z) = [a_k^Z(i,j)]_{0 \le i,j \le n-1}$, $k = 0,1,\ldots,m-1$.

Theorem 3-2 states that two images X and Y are identical (i.e. $X = Y$) if and only if their corresponding BGLAMs are the same. By the definition of BGLAM distance function (see Definition 3-16), one can check that $d(X,Y) = 0$ if and only if $X = Y$. ■

The significance of the BGLAM distance function is the one-to-one property. A zero value of the distance measure guarantees that the two images are identical. Although many metric distance measures [1, 4, 36, 53, 71, 72, 113, 116, 117, 152, 160, 172, 173] have been proposed, to our best knowledge, none of them is one-to-one.

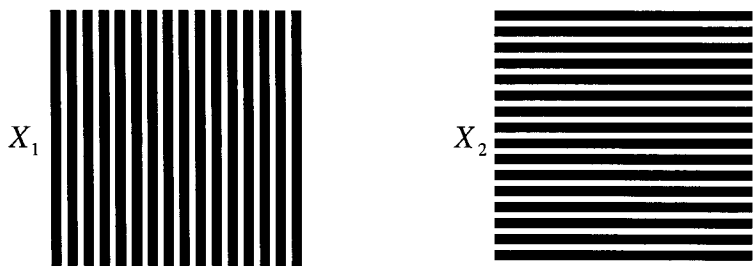


Figure 3-11: An example of demonstrating the importance of using BGLAMs in defining a one-to-one distance function. The right stripe-texture image is a rotation of the left image (a binary image of size 128×128) by 90 degrees.

In texture modeling, the one-to-one property is crucial in measuring the similarity between texture samples. Since the distance function is continuous, the one-to-one property implies that if the distance of Y over X gradually converges to zero, image Y will gradually converge to X. A distance measure (e.g. [53, 72, 116, 117, 152]) without the one-to-one property cannot guarantee this. In the next chapter, we demonstrate that if the distance of two texture images X and Y is below a threshold, X and Y are guaranteed to look similar to each other.

In the distance function given in Definition 3-16, BGLAMs play an important role in assuring the one-to-one property. Instead of BGLAMs, if other terms such as SGLAMs

or GLCMs are used in the definition, the one-to-one property may not hold. Figure 3-11 gives an example to illustrate this point as explained below.

Denote the original BGLAM distance measure (see Eq. 3.13) by $d_B(.,.)$. Suppose SGLAMs replace BGLAMs in the definition in Eq. 3.13, and $d_S(.,.)$ denotes the corresponding distance measure. For the two binary texture images X_1 and X_2 of size 128×128 as shown in Figure 3-11, assume that the four nearest-neighbor neighborhood system $\boldsymbol{N}$ is used. Let $\boldsymbol{N}_0$, $\boldsymbol{N}_1$, $\boldsymbol{N}_2$, and $\boldsymbol{N}_3$ be the left-, right-, top-, and bottom-neighbor neighborhood system decomposition of $\boldsymbol{N}$, respectively; $\boldsymbol{A}_1$ and $\boldsymbol{A}_2$ be the SGLAMs of X_1 and X_2 over $\boldsymbol{N}$, respectively; and $\boldsymbol{A}_{1k}$ and $\boldsymbol{A}_{2k}$ be the BGLAMs of X_1 and X_2 over $\boldsymbol{N}_i$ respectively for $k = 0,1,2,3$. Then we have the following:

$$\boldsymbol{A}_1 = \boldsymbol{A}_2 = \begin{bmatrix} 36864 & 4096 \\ 4096 & 20480 \end{bmatrix},$$

$$\boldsymbol{A}_{10} = \boldsymbol{A}_{11} = \begin{bmatrix} 10240 & 0 \\ 0 & 6144 \end{bmatrix},\ \boldsymbol{A}_{12} = \boldsymbol{A}_{13} = \begin{bmatrix} 8192 & 2048 \\ 2048 & 4096 \end{bmatrix},$$

$$\boldsymbol{A}_{20} = \boldsymbol{A}_{21} = \begin{bmatrix} 8192 & 2048 \\ 2048 & 4096 \end{bmatrix},\ \boldsymbol{A}_{22} = \boldsymbol{A}_{23} = \begin{bmatrix} 10240 & 0 \\ 0 & 6144 \end{bmatrix},$$

$$d_S(X_1, X_2) = d_S(\boldsymbol{A}_1, \boldsymbol{A}_2) = \sum_{i,j=0}^{1} |a_1(i,j) - a_2(i,j)| = 0,$$

$$d_B(X_1, X_2) = \sum_{k=0}^{3} \|\boldsymbol{A}_{1k} - \boldsymbol{A}_{2k}\| = \sum_{k=0}^{3} \sum_{i,j=0}^{1} |\, a_{1k}(i,j) - a_{2k}(i,j)\,| = 32768 \neq 0.$$

Although two images X_1 and X_2 are different, i.e. $X_1 \neq X_2$, the SGLAM-based distance measure between X_1 and X_2 is zero, i.e. $d_S(X_1, X_2) = 0$. This implies that SGLAM-based distance is not one-to-one, and thus cannot differentiate the two texture

images as shown in Figure 3-11. However, the BGLAM-based distance function can differentiate the two texture images because $d_B(X_1, X_2) \neq 0$. In the above distance calculations, we have used the original aura matrices instead of the normalized ones because X_1 and X_2 are defined on the same lattice, and the results will not be affected whether or not normalized aura matrices are used.

3.5 Summary

In this chapter, we present the BGLAM mathematical framework. By clarifying the relationship between BGLAMs, SGLAMs, GLAMs, and GLCMs, we show that BGLAMs form the basis of GLAMs; while GLCMs forms the basis of SGLAMs. We prove that two images are identical if and only if their corresponding BGLAMs are the same. Therefore, an image can be uniquely represented by and faithfully reconstructed from its BGLAMs. However, the statement does not hold with GLCMs or SGLAMs.

In addition, we propose a BGLAM-based distance function, and prove that the new distance function is metric and one-to-one. The one-to-one property is not guaranteed by conventional metric distance functions [53, 72, 116, 117, 152], and its significance is that a zero value of the distance measure guarantees that the two images are identical. In the next chapter, we demonstrate that the BGLAM-based distance function can be used as a quantitative measure in evaluating synthesis results w.r.t. input textures to determine if the output is a successful synthesis of the input, and that if the BGLAM distance of two texture images is below a threshold, they are guaranteed to look similar to each other.

The BGLAMs of an image characterize the cooccurrence probability distributions of gray levels at all possible displacement configurations and thus estimate the underlying stochastic process that is used to generate a given texture sample. BGLAMs should not be confused with GLCMs. It is proved that any GLCM can be represented as a sum of two BGLAMs. In fact, all of the theorems on BGLAMs described in the chapter do not hold for GLCMs. To illustrate the representative property of BGLAM, in the next chapter, we show that BGLAMs outperforms GLCMs as well as other existing techniques in 2D texture synthesis.

Chapter 4

BGLAM 2D Texture Synthesis

4.1 Introduction

For a given input texture sample, the problem of 2D texture synthesis is to generate a new texture image that looks similar to the input. One major problem of existing 2D texture synthesis techniques is that the output textures are often generated by using some characteristics of input examples, which may not represent the input texture appropriately. For instance, in existing feature-matching approaches [6, 35, 81, 143, 190], a set of filter responses at multiple scales and orientations are used to characterize an example texture. However, as suggested by Zhu et al. in their FRAME (Filters, Random Fields and Maximum Entropy) model [208], it requires an infinite number of filters (each filter is as big as the given texture image) to model a given texture with the necessary and sufficient information. In addition, it is not an easy task to select the filters or to determine the number of filters to model a typical texture [208]. Because of using ambiguous definitions of textures, existing synthesis techniques cannot determine whether or not the synthesis result is acceptable. Visual inspection is the only way to evaluate the synthesis results.

To address the above problems, this chapter presents a new technique, called *aura 2D textures*, for generating 2D synthetic textures from input texture samples using BGLAMs. The technique is based on the BGLAM mathematical framework developed in the previous chapter. We demonstrate that the new technique can successfully synthesize

a wide range of textures using a small set of BGLAMs (e.g. 48 BGLAMs for an input texture of size 64×64). In addition, based on the BGLAM distance measure, our new technique is able to automatically evaluate the results and determine whether or not the output is a successful synthesis of the input. To our best knowledge, none of the previous techniques has the ability to evaluate synthesis results.

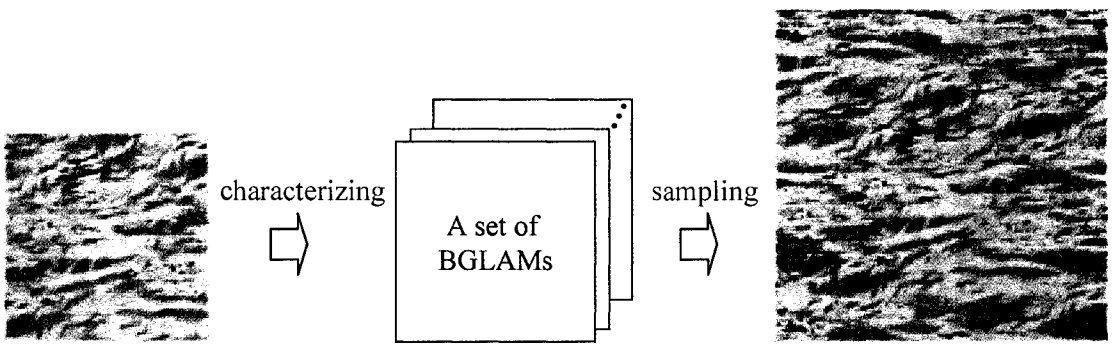


Figure 4-1: The basic idea of the approach of BGLAM-based 2D texture synthesis. The input example (a) is first characterized by a set of BGLAMs (b), and then the BGLAMs are used to generate an output texture (c).

The main idea of our approach is shown in Figure 4-1. Given a texture sample, our method first characterizes it by a set of BGLAMs. Then, by sampling the BGLAMs only, our method generates an output texture similar to the input with similar BGLAMs. This is done by iteratively modifying the gray level of each pixel in the output image, which is initialized as a random noise image, until the distance between the corresponding BGLAMs of the output and those of the input is small enough or until there are no further changes in the gray level values of the output.

The BGLAM distance function defined in Section 3.4 is used to evaluate the synthesis result quantitatively for determining whether or not the output looks similar to

the input. Experiments have shown that if the distance value is below a threshold value (0.1 used in our experiments), then the output is guaranteed as a successful synthesis of the input. It is noteworthy that the one-to-one property is crucial for measuring the similarity between textures. Without this property, which is the case in existing techniques [35, 143, 190, 208], a less similar texture image might be given a higher degree of similarity to the input. Hence, existing techniques only show some synthesized results without evaluating them.

The advantages of using BGLAMs in texture analysis and synthesis include: (1) easy to compute, (2) accurate representations of example textures, (3) able to evaluate the results, and (4) no filters required. By thorough experiments, we show that BGLAMs outperform both SGLAMs and GLCMs, and can successfully synthesize a broad range of textures with comparable results to those of existing techniques [81, 106, 190].

4.2 Related Works

Being complete on its own, this section gives a brief discussion on related works in 2D texture analysis and synthesis. For the detailed discussions, the interested readers are referred to Chapter 2. Since Julesz's pioneering work in texture analysis [93], various approaches have been proposed for texture analysis and synthesis. One of the most influential approaches is the MRF models [32, 63]. Only a limited range of textures can be modeled with earlier MRF techniques because of the limited size of the cliques and of the low-order statistics used in modeling. To address these problems, Zhu et al. propose the FRAME model, which incorporates filtering theory into the MRF models to synthesize a wider range of textures [208], and Deng and Clausi propose the ACGMRF

(Anisotropic Circular Gaussian MRF) model to model anisotropic textures [38]. The conventional MRF texture models are also generalized by Popat and Picard to the cluster-based probability model [141] and by Paget to the strong MRF model [126] for modeling textures with high order statistics. Different from Zhu et al.'s FRAME model, both approaches are nonparametric. In general, MRF models are slow because of the expensive local probability construction (normally based on exponential functions) at each pixel location during the sampling. To speed up, nonprobabilistic pixel-based sampling techniques [3, 49, 190] are proposed by a number of researchers, which are further improved by the patch-based sampling techniques [50, 100, 101, 106, 186].

Techniques are also developed to synthesize textures by matching features in multiple scales and orientations, pioneered by Heeger and Bergen's work [81] using a global histogram-matching strategy. Later, in the work of Portilla and Simoncelli [143], it is shown that new textures can be synthesized by matching the corresponding joint statistics of complex wavelet coefficients between the input and output image pyramids. Rather than using global joint statistics, DeBonet and Viola use joint occurrence of local features in multiresolutions to model texture images [35]. Their approach has been generalized by Bar-Joseph et al. to texture mixture and video texture using statistical learning [6].

Another influential approach called GLCMs (Gray Level Cooccurrence Matrices) [21, 34, 76] can be used as a powerful tool for texture analysis, segmentation, classification, and synthesis. The disadvantage of the GLCMs is that they contain cooccurrence information between two pixels only, and thus cannot capture the spatial relationship between three or more pixels in the image. This problem can be addressed by

using GLAMs (Gray Level Aura Matrices) [51, 52, 137, 138], which incorporate neighborhood systems to model the relationship between the target pixel and its neighboring pixels, and thus can capture the relationship between any number of pixels. Recently, GLAMs are successfully applied to texture similarity measure and learning by Qin and Yang [152]. All previous studies on GLAMs assume that the neighborhood systems are symmetric, which have caused difficulties in modeling anisotropic textures (see Figure 3-8).

The work in this chapter presents a new technique for 2D texture analysis and synthesis based on the BGLAM mathematical framework developed in Chapter 3. With respect to synthesis of textures and evaluation of the results, the performance of our approach is extensively evaluated and compared with symmetric GLAMs and with GLCMs.

4.3 The Approach

Figure 4-2 gives an overview of the BGLAM-based texture synthesis approach. Given an input texture X, its BGLAMs $A(X)$ are computed using an algorithm described later (Section 4.3.1). Then the ICA (Independent Component Analysis) [83] is used to identify the independent BGLAMs. For simplicity reasons, we also use $A(X)$ to represent the independent BGLAMs of X. The output Y is initialized as a random noise image, and its BGLAMs $A(Y)$ corresponding to $A(X)$ are computed. Then, a BGLAM-based random sampling procedure is employed to iteratively update the output until the BGLAM distance $d(A(X), A(Y))$ (see Eq. 3.13) is small enough or until there is no further change in pixel's gray level values in the output. During an iteration of the

sampling process, the gray level of a pixel in the output Y is modified such that the newly assigned gray level to the pixel will decrease or at least do not increase the BGLAM distance between Y and X.

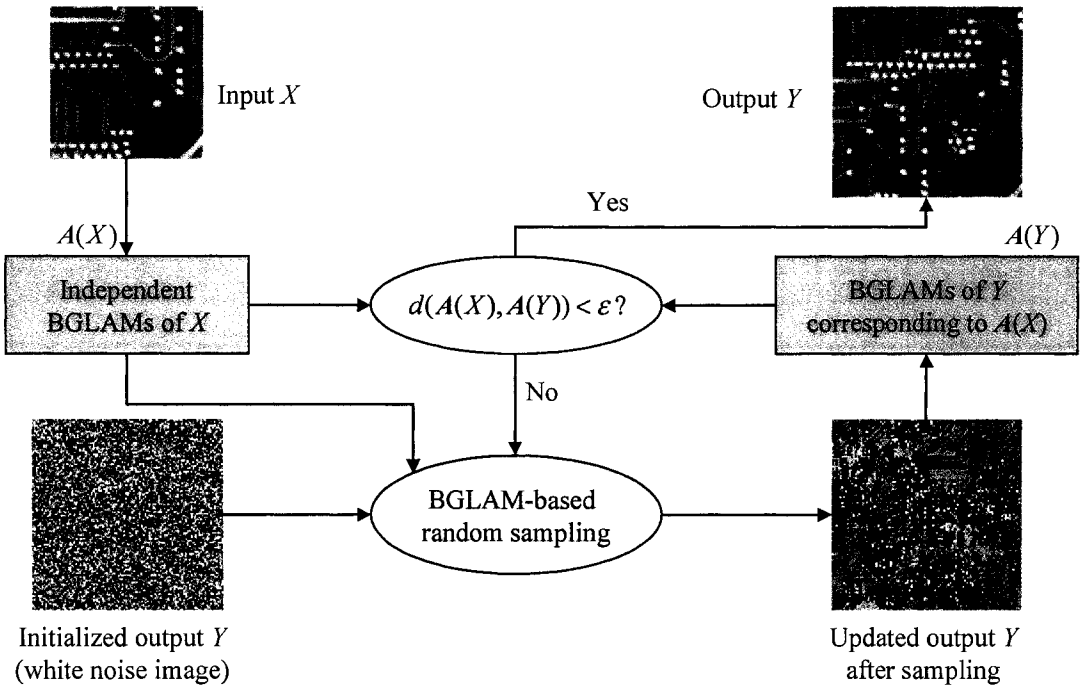


Figure 4-2: An overview of the approach of aura 2D texture synthesis.

4.3.1 Calculating BGLAMs

As proved in the previous chapter, two images are identical if and only if their corresponding BGLAMs are the same, and thus an image can be uniquely represented by its BGLAMs. In this chapter, a compact set of BGLAMs defined over a neighborhood system (e.g. a 9×9 square window) is used to characterize input samples. For an $m\times m$ (m is an odd number and $m>1$) neighborhood system, the total number of BGLAMs is m^2-1 because there are m^2-1 neighboring pixels around the central target pixel, and

each neighboring pixel accounts for a BGLAM. An example of a 5×5 binary image and its BGLAMs calculated over a 3×3 square window are shown in Figure 4-3.

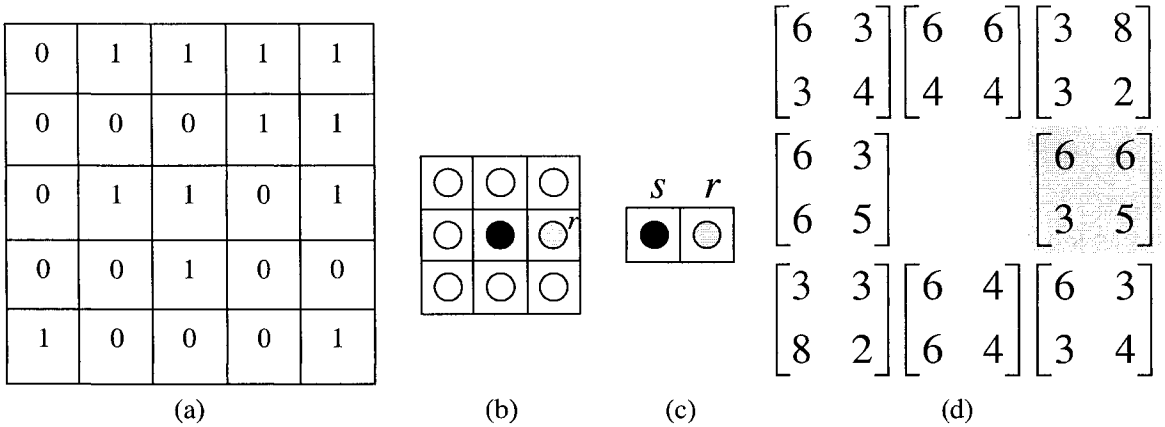


Figure 4-3: The BGLAMs of a 5×5 binary image. The binary image, the 3×3 neighborhood system, the displacement configuration of neighboring pixel r, and the corresponding BGLAMs of eight displacement configurations are in (a), (b), (c), and (d), respectively. For the ease of reference, the BGLAMs in (d) are placed according to their displacement configurations in (b). For example, the BGLAM in highlighted color in (d) is for the displacement configuration of the pixel r in (b).

Given an image X on lattice S with a neighborhood system $\boldsymbol{N}$, we assume that a common state space $\Lambda=\{0,1,...,G-1\}$ is used for all sites $s\in S$, where G is the total number of gray levels available in the image. A straightforward way to compute the BGLAM of a displacement configuration (see Figure 4-3 (c)) is to first compute all the gray level sets $\{S_g \mid g\in\Lambda\}$ using Eq 3.5 in Section 3.1. Then, for each pair S_g and $S_{g'}$ where $g,g'\in\Lambda$, one can use Eq. 3.4 to compute the aura measure $m(S_g,S_{g'})$, which

takes $O(|S_g|*|S_{g'}|)$ to calculate. Since $\sum_{g=0}^{G-1}|S_g|=|S|$, the total time to compute a BGLAM is $O(|S|^2)$.

The above brute-force algorithm is slow, a simple but fast algorithm with complexity $O(|S|)$ can be achieved for calculating a BGLAM by processing through each site s of the lattice S only once. To calculate the BGLAM of a specific displacement configuration, e.g. the one shown Figure 4-3 (c), the fast algorithm works as follows. Initialize each entry of the BGLAM $A=[m(S_i,S_j]_{0\le i,j\le G-1}$ to zero, i.e. $m(S_i,S_j)=0$ for $0\le i,j\le G-1$. For each site s, let g be its gray level, one checks its neighboring site r in the displacement configuration and finds its gray level g'. Then we increment the value of $m(S_g,S_{g'})$ by 1. After all the sites in the image have been processed, the calculation of the BGLAM is finished. When handling a target site on the image boundaries, we consider only its neighboring sites inside the image and discard those outside of the image. Once the BGLAMs are computed for the input texture, they are stored and used as the only representation of the input to generate the output during synthesis. In other words, the input texture itself will not be needed any more once its BGLAMs are computed.

4.3.2 Similarity Measure

During synthesis, it is important to have an accurate measure to determine how close the output texture matches the input. In our method, the similarity between two texture images is measured by the sum of the distances between their corresponding BGLAMs, where the distance of two matrices is the Manhattan distance of the two

matrix vectors. Precisely, given two texture images X and Y defined on S. Let $A(X)=\{A_k \mid 0 \le k \le m-1\}$ and $A(Y)=\{B_k \mid 0 \le k \le m-1\}$ be their corresponding normalized BGLAMs, then the similarity measure between X and Y is given by:

$$d(X,Y)=d(A(X),A(Y))=\frac{1}{m}\sum_{k=0}^{m-1}\| A_k - B_k \|, \qquad (4.1)$$

where for a given matrix $A=[a(i,j)]_{0\le i,j\le G-1}$, its norm is $\| A \|=\sum_{i,j=0}^{G-1}| a(i,j) |$, and a aura matrix $A=[a(S_i,S_j)]$ is *normalized* if $\sum_{i,j=0}^{G-1} a(S_i,S_j)=1$.

Section 3.4 includes a proof that the distance function defined in Eq. 4.1 is one-to-one in the sense that a distance measure of zero guarantees that the two images are identical. Since the distance function is continuous, the one-to-one property implies that the smaller the distance value, the closer the two texture images look to each other. In fact, as demonstrated later, this one-to-one property enables our algorithm to evaluate the synthesis results automatically. As far as we know, none of the existing techniques has this feature. In the rest of the chapter, we assume that all BGLAMs are normalized.

4.3.3 BGLAM-Based Random Sampling

The BGLAM-based random sampling procedure iteratively modifies the output such that its BGLAMs match those of the input. In the beginning, the output texture is initialized as a white noise image (see Figure 4-2). During an iteration of sampling, each pixel of the output is visited *randomly* once, and its gray level is modified so that the BGLAMs of the output get closer to those of the input. More precisely, when visiting a pixel, the algorithm first finds the candidate set of all gray levels (different from the

current pixel value) that decrease or at least do not increase the BGLAM-based distance (defined in Eq. 4.1) between the output and the input. Then it randomly chooses a gray level from the candidate set and sets the pixel value to the newly selected gray level. Note that even when a gray level does not decrease the distance, i.e. at the same distance as the current gray level, the algorithm also includes it into the candidate set in order to increase the randomness in the output. It is possible that the candidate set is empty at the end of search, which implies that any gray level that is different from the current pixel value will increase the distance. In such a case, the pixel retains its current gray level, and the algorithm goes to process the next pixel in the output image. When the BGLAM-based distance between the output and the input is below a threshold or there is no change in gray level values in any pixel of the output, the sampling process returns the output texture as the final result.

4.3.4 Algorithm

The pseudo code of the BGLAM-based 2D texture synthesis algorithm is given in Figure 4-4. Given an input texture image X, the algorithm generates a synthesized texture image Y that looks similar to the input X. There are four steps in the algorithm. The first step is to initialize the output texture image as a white noise image using a pseudo random number generator. The normalized BGLAMs of the input and output texture images are then computed in the next two steps using the fast algorithm that is described in Section 4.3.1. After BGLAMs are computed, the independent BGLAMs are computed using the standard ICA algorithm [83]. The last step is the BGLAM-based random sampling procedure as described in the previous subsection.

BGLAM-Based 2D Texture Synthesis

Input:
$X \leftarrow$ sample texture image.
$\varepsilon \leftarrow$ a given small threshold value in (0,1)
Output:
$Y \leftarrow$ the synthesized texture image.

Begin
1 Initialize Y as a random noise image.
2 $A(X) \leftarrow$ the normalized BGLAMs of X.
$A(X) \leftarrow$ Independent BGLAMs of $A(X)$ by ICA.
3 $A(Y) \leftarrow$ the normalized BGLAMs of Y.
$A(Y) \leftarrow$ Independent BGLAMs of $A(Y)$ by ICA.
4 While $d=d(A(X), A(Y)) \geq \varepsilon$ **do**
4.1 While there are unvisited sites in Y, randomly choose an unvisited site s **do**
grayLevel(s) $\leftarrow$ *bglamBased2DRandomSampling*(s, d, $A(X)$, $A(Y)$,Y).
End of while
End of while
End of begin

bglamBased2DRandomSampling(s, d, $A(X)$, $A(Y)$, Y)
b.1 C=empty (the candidate set of gray levels for site s).
b.2 For each gray level $j = 0$ to G - 1 **do**
b.2.1 $A_j(Y) \leftarrow$ the normalized BGLAMs of Y when site s has gray level j.
b.2.2 $d_j \leftarrow d(A(X), A_j(Y))$, which is calculated using Eq. 4.1.
b.2.3 if $d_j \leq d$, then $C = C \cup \{j\}$.
b.3 If C is empty, **then** $g \leftarrow$ the current gray level value of s,
Else $g \leftarrow$ a randomly chosen gray level from C.
b.4 Recalculate $A(Y)$.
b.5 Return g.

Figure 4-4: The BGLAM-based 2D texture synthesis algorithm.

The major computation cost of the algorithm is spent in the two *while* loops in step 4. In an iteration of step 4 (i.e. one pass of going through step 4.1 by visiting all sites in Y), a brute force method would perform fresh recalculations each time in computing the BGLAMs of the output and the BGLAM-based distance in sampling (i.e. in *bglamBased2DRabdomSampling*). The time complexity of an iteration of step 4 using the brute-force calculation is at least $O[m*np*G*(np+G^2)]$ (the proof is given below), where m is the total number of BGLAMs, np the number of pixels in the output, and G number of gray levels in the image. As proved below, a more efficient way is to perform

an iterative update based on existing information, which can be done with a computation cost of $O(m*np*G)$ because when a pixel changes its gray level value, only its neighboring pixels will be affected. In fact, according to Lemma 3-5, each gray level aura matrix $\boldsymbol{A}_j(Y)$ in the step b.2.1 of *bglamBased2DRabdomSampling* in Figure 4-4 can be efficiently updated from its previous version in four simple arithmetic operations as described in Algorithm 3-1 in Chapter 3. At the end of the procedure *bglamBased2DRabdomSampling*, one can always keep an updated version of $\boldsymbol{A}(\boldsymbol{Y})$ for the selected gray level g at site s for the next update. In addition, using a few simple arithmetic operations, distance d can be efficiently updated from its previous value and distance d_j can be efficiently calculated from distance d without a complete recalculation. To achieve the above fast iterative updates, however, the algorithm must store the BGLAMs of the input and of the output as well as the distance between each pair of the corresponding BGLAMs of the input and of the output. The detailed proofs of the above time complexities are given as follows.

[Proof of the brute-force time complexity] We would like to prove that the time complexity for one iteration in step 4, i.e. one pass that goes through step 4.1 by visiting all pixels in image Y, is at least $O(m*np*G*(np+G))$, where m is the total number of BGLAMs, np the number of pixels in the output, and G number of gray levels in the image. Let $T(4.1)$ and $T(b)$ be the time complexity for step 4.1 and for the procedure *bglamBased2DRabdomSampling*, respectively, then the time for one pass of step 4, is given by:

$$T(4.1) = np*T(b). \tag{4.2}$$

From pseudo code of *bglamBased2DRandomSampling* given in Figure 4-4, we have:

$$
\begin{aligned}
T(b) &= T(b.1) + T(b.2) + T(b.3) + T(b.4) + T(b.5) \\
&\geq O(1) + G*[T(b.2.1) + T(b.2.2) + T(b.2.3)] + O(1) + O(m*np) + O(1) \\
&= G*[T(b.2.1) + T(b.2.2) + T(b.2.3)] + O(m*np) \\
&= G*[O(m*np) + O(m*G^2) + O(1)] + O(m*np) \\
&= G*O[m*(np+G^2)] + O(m*np) \\
&= O(m*G*(np+G^2)] + O(m*np) \\
&= O[m*G*(np+G^2) + m*np] \\
&= O(m*G*np + m*G^3 + m*np) \\
&= O(m*np*G + m*G^3) \\
&= O[m*G*(np+G^2)]
\end{aligned}
\qquad .(4.3)
$$

In the above equation, the time for step b.3 is at least $O(1)$. Since there are m BGLAMs in $\boldsymbol{A}(Y)$ and each BGLAM takes time $O(np)$ to compute by the algorithm described in Section 4.3.1, the time complexity of step b.4 is $T(b.4) = O(m*np)$. When calculating the time for step b.2.1, there are m BGLAMs in $\boldsymbol{A}_j(Y)$ and each BGLAM takes time $O(np)$ to compute; thus the total time for step b.2.1 is $O(m*np)$. By Eq. 4.1, the time for step b.2.2 is $O(m*G^2)$ because there are m matrices of size $G \times G$ and each matrix needs a time of $O(G^2)$ to compute its norm. By Eq. 4.2 – 4.3, we have:

$$T(4.1) \geq np*O[m*G*(np+G^2)] = O[m*np*G*(np+G^2)]. \blacksquare$$

[Proof of the fast-version time complexity] Let $A(X) = \{B_k(X) \mid 0 \leq k \leq m-1\}$, $\boldsymbol{A}_j(Y) = \{B_{jk}(Y) \mid 0 \leq k \leq m-1\}$, and $\boldsymbol{A}(Y) = \{B_k(Y) \mid 0 \leq k \leq m-1\}$. Based on Lemma 3-5 and Algorithm 3-1, for each $k = 0,1,...,m-1$, matrix $B_{jk}(Y)$ can be efficiently calculated from matrix $B_k(Y)$ by changing the values of only four entries as shown in Eq. 3.11 in Chapter 3, which implies that $B_{jk}(Y)$ can be updated from $B_k(Y)$ using constant time. Thus, the total time for step b.2.1 is $m*O(1) = O(m)$. Furthermore, since there are

only four different entries between $B_{jk}(Y)$ and $B_k(Y)$, the distance d_j in step b.2.2 , which is $d_j = d(A(X), A_j(Y)) = \sum_{k=0}^{m-1} \| B_k(X) - B_{jk}(Y) \|$, can be efficiently updated from distance d, which is $d = d(A(X), A(Y)) = \sum_{k=0}^{m-1} \| B_k(X) - B_k(Y) \|$, using a computation cost of $m * O(1) = O(m)$. The time for step b.3 is at most $O(G)$, and matrix $A(Y)$ in step b.4 can be updated from its previous version using time $O(m)$. Thus, we have the following:

$$\begin{aligned} T(b) &= T(b.1) + T(b.2) + T(b.3) + T(b.4) + T(b.5) \\ &\le O(1) + G * [T(b.2.1) + T(b.2.2) + T(b.2.3)] + O(G) + O(m) + O(1) \\ &= G * [T(b.2.1) + T(b.2.2) + T(b.2.3)] + O(m) \\ &= G * [O(m) + O(m) + O(1)] + O(m) \\ &= G * O(m) + O(m) \\ &= O(m * G + m) \\ &= O(m * G) \end{aligned} \qquad .(4.4)$$

By Eq. 4.2 and Eq. 4.4, we have $T(4.1) \le np * O[m * G] = O[m * np * G]$. ■

4.3.5 Color Image

For color input texture images, one cannot simply apply the above basic algorithm to each of the RGB channels separately since the RGB components of a color image are dependent on each other. Before applying the basic aura texture synthesis algorithm, a color-space transformation $\boldsymbol{T}$ based on the singular value decomposition technique (SVD) [146] is used to transform the R, G, and B components of the color image into three independent components R', G', and B' in another color space. After this RGB-color-decorrelation step, the basic synthesis algorithm is applied to each of the independent color components R', G', and B' to generate three output textures in the

transformed color space, which are then transformed back into the RGB color space to produce the final synthesized color texture image using the inverse transformation of $\boldsymbol{T}$. The pseudo code of the RGB color-space transformation algorithm based on SVD is given in Figure 4-5.

SVD RGB Color Transform

Input:
$X \leftarrow$ the RGB color image.
Output:
$Y \leftarrow$ the color image with independent color channels.
$\boldsymbol{T} \leftarrow$ the RBG color transformation.
$\boldsymbol{T}^{-1} \leftarrow$the inverse transformation of $\boldsymbol{T}$.

Begin
1 Subtract the mean color from each RGB color channel.
1.1 $R, G, B \leftarrow$ the red, green, and blue channel of X, respectively.
1.2 $r, g, b \leftarrow$ the mean color of R, G, and B, respectively.
1.3 $R' \leftarrow R-r$ (subtract the mean of the red values from the red value at each pixel of R).
1.4 $G' \leftarrow G-g$.
1.5 $B' \leftarrow B-b$.
1.6 $X' \leftarrow$ the color image of R', G', B' channels.
2 Calculate the 3x3 covariance matrix C of X'.
2.1 $n \leftarrow$ the number of pixels in X'.
2.2 $D \leftarrow$ the 3xn matrix whose columns are color values of each pixel in X'.
2.3 $C \leftarrow DD^{t}$, where D^{t} is the transpose of matrix D.
3 Perform SVD on C by decomposing C into the product of three of matrices.
$C = US^{2}U^{t}$, where U is orthonormal and S is diagonal.
4 Calculate the transformation $\boldsymbol{T}$ and the inverse of $\boldsymbol{T}$.
$\boldsymbol{T} \leftarrow S^{-1}U$, where S^{-1} is the inverse matrix of S.
$\boldsymbol{T}^{-1} \leftarrow US$.
5 $Y \leftarrow TD$
Y is a 3xn matrix with each row representing a color channel.
6 Return Y, $\boldsymbol{T}$ and $\boldsymbol{T}^{-1}$.
End

Figure 4-5: The algorithm of RGB-color transformation using SVD.

4.4 Experiments

In our approach of 2D texture synthesis, the neighborhood size is an important parameter that affects the synthesis results. In general, an image containing large structural textures (see textures in the 1st column in Figure 4-9) requires a relatively large neighborhood size. For a given input texture, different synthesis results can be generated with different neighborhood sizes. Figure 4-6 below gives an example texture and its synthesized textures generated with different neighborhood sizes. It is an interesting future research topic to systematically determine the optimal neighborhood size (e.g. 11x11 for the input texture shown in Figure 4-6) for a given input texture image to obtain the best run-time performance.

Figure 4-6: An example of the synthesis results using the neighborhoods of different sizes given under each output.

4.4.1 Comparison with SGLAMs and GLCMs

For simplicity reasons, we only describe the algorithm of comparing BGLAMs with GLCMs, and the one of comparing BGLAMs with SGLAMs is similar. A fair comparison scheme is to use the same general algorithm as described in Figure 4-4, but sampling with BGLAMs and GLCMs, respectively, calculated over the same square neighborhood structuring element *E*. In addition, for fairness in comparison, when using BGLAMs, the step of ICA is omitted, i.e. all BGLAMs calculated over *E* are used. For

example, if the neighborhood structuring element E is a square window of size 11×11 with the target pixel at the center, then $11\times11-1=120$ BGLAMs and $120/2=60$ GLCMs will be used for generating synthetic textures because each BGLAM is calculated over a single site structuring element and each GLCM is calculated over a symmetric structuring element of two sites.

We generate a database of 848 texture images (with repetitions or very similar texture images removed using the BGLAM distance measure defined in Eq. 4.1) from various sources (see [124]), randomly select half of the images from the database, and use them as input samples. We generate two outputs for each input by sampling 120 BGLAMs and 60 GLCMs (calculated from the input over a square window of size 11×11), respectively. We evaluate the synthesis results subjectively. Each output is evaluated by 10 people. Among the 10 people, 5 of them are researchers in the same research lab and have the knowledge on texture analysis and synthesis; the other 5 people are graduate students in the department and have the general knowledge in computer vision and image processing. Each subject is asked to determine whether or not the output looks similar to the input. If over 50% of the subjects agree that a given output texture looks similar to its corresponding input texture, then a SUCCESS is assigned to the output; otherwise a FAILURE is assigned.

Experiments have shown that the average percentages of SUCCESS for BGLAMs, GLCMs and SGLAMs are 81.4%, 44.7% and 39.7%, respectively, which indicates that BGLAMs significantly outperform both GLCMs and SGLAMs. Figure 4-7 gives some examples of texture synthesis by sampling BGLAMs, GLCMs, and SGLAMs, respectively.

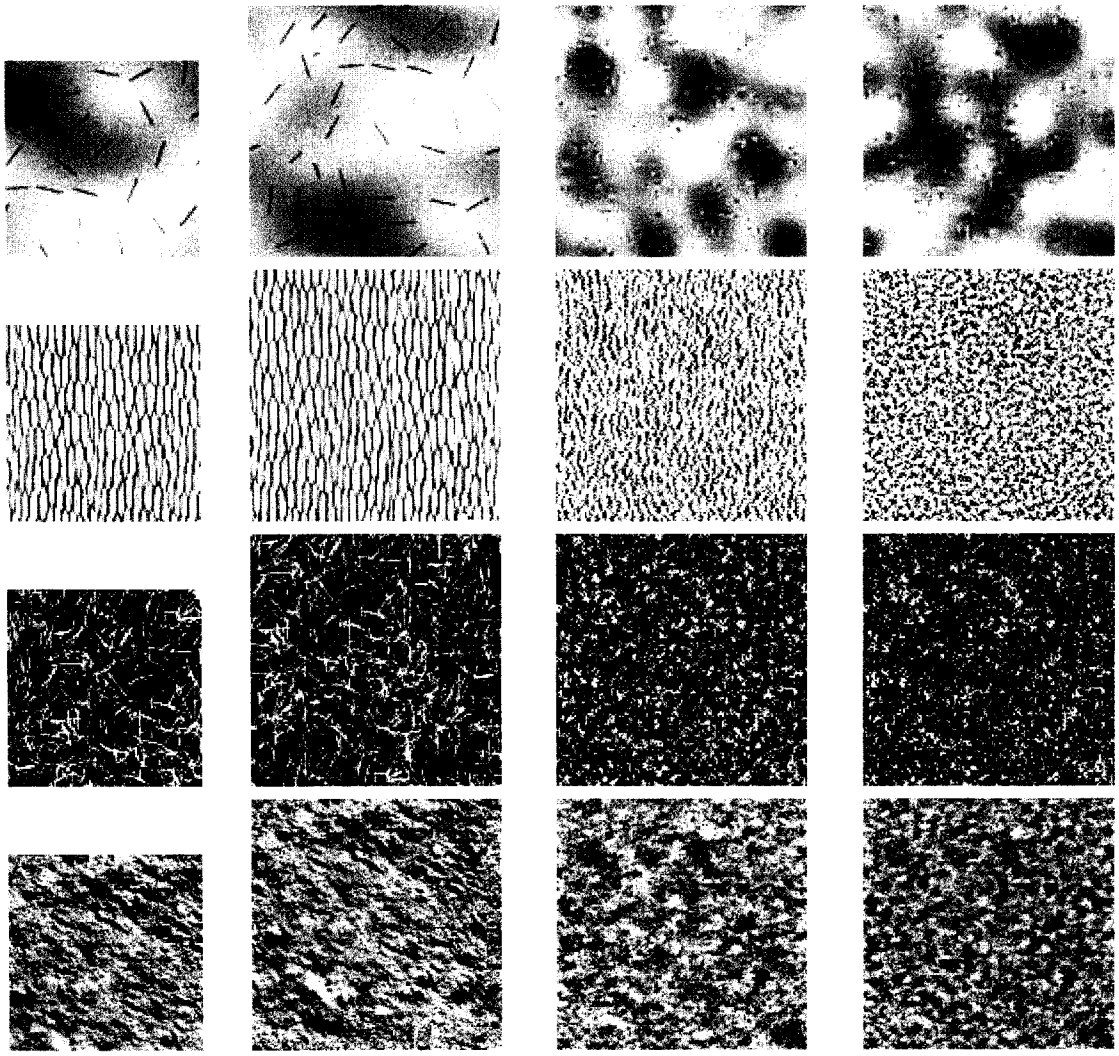


Figure 4-7: The comparison results of texture synthesis using BGLAMs, GLCMs and SGLAMs. Images in the 1st column are the input (size 100×100), the synthesized images (size 128×128) by sampling BGLAMs, GLCMs and SGLAMs are in the 2nd, 3rd and 4th columns, respectively. The results indicate that BGLAMs outperforms both GLCMs and SGLAMs.

BGLAMs, GLCMs, and SGLAMs are also evaluated with each other by the ability of measuring the similarity between textures. We use the distance function defined

in Eq. 4.1, but working with BGLAMs, GLCMs and SGLAMs, respectively. For each synthesized texture, we calculate the distance value to the input. If the value is below 0.1, then the output is considered similar to the input, and thus a SUCCESS. Otherwise, it is a FAILURE. On the other hand, each output is evaluated by 10 people. It is considered as a SUCCESS if over 50% of the subjects answer YES. Otherwise, it is FAILURE. For each output, the evaluation result from the distance measure is compared with the subjective evaluation to determine if it is a MATCH. The average percentage of match among all output textures is used to determine the ability of a distance measure for measuring the similarity between textures. The experimental results show that the average percentages of MATCH for BGLAMs, GLCMs and SGLAMs are 75.8%, 49.8% and 41.3%, respectively. This test indicates that the BGLAM distance measure has the ability to measure the similarity between textures, while none of GLCM and SGLAM distance measures has.

4.4.2 Comparison with Existing Techniques

Figure 4-8 gives some comparison results of texture synthesis, where images in column 1 are the input texture samples, and images in the last four columns are the synthesized results of: our algorithm, the Heeger and Bergen algorithm [81], the Wei and Levoy algorithm [190], and the Liang et al. algorithm [106]. We implement both Heeger's and Wei's algorithms, in which Heeger's algorithm is based on the steerable pyramid [81] and Wei's algorithm is based on the Gaussian pyramid [190].

The results of our algorithm are generated using 48 BGLAMs calculated from a square window of size 7x7 around a target pixel. The results for Heeger's algorithm are generated using steerable pyramids with 3 levels and 4 orientations (i.e. 0, 45, 90, and

135 degrees as used in the Heeger and Bergen's work [81]). For Wei's algorithm, a Gaussian pyramid of 3 levels is used to synthesize from a given input texture. The neighborhood sizes used for a Gaussian pyramid are {3x3,1}, {5x5,2}, {7x7,2} from the lowest resolution level to the highest resolution level, where {7x7,2} means a multiresolution neighborhood of 2 levels (with size 7x7 at the higher resolution level and 3x3 at the lower resolution level) is used to generate the highest resolution level. The results for the Liang et al.'s algorithm are taken from Paget's website [124].

As shown in Figure 4-8, Heeger's algorithm is able to capture the overall appearance of a given texture sample, but fails to capture the local structures in the texture because of the global histogram-matching scheme used in the algorithm. Wei's algorithm is able to capture the details of a given texture using a pixel-based sampling scheme, but has a smoothing effect in the output because of the inaccurate SSD (sum of squares differences) measure used to measure the similarity between the output and the input and the Gaussian pyramid used to represent a texture image. Although, Liang's algorithm can generate good results, our algorithm generates better results for the input textures in the 1^{st}, 2^{nd}, and 5^{th} rows. For other input textures in the figure, the results for our algorithm are comparable to those of Liang's algorithm.

The number under each output texture is the value of the BGLAM-based distance measure (see Eq. 4.1) of the output compared to the input. The smaller the BGLAM distance value, the greater the similarity between two texture images. The quantitative evaluation based on the BGLAM distance function also shows that the synthesis results from our approach are better than those from Heeger's and Wei's algorithm, and are comparable to those of Liang's algorithm.

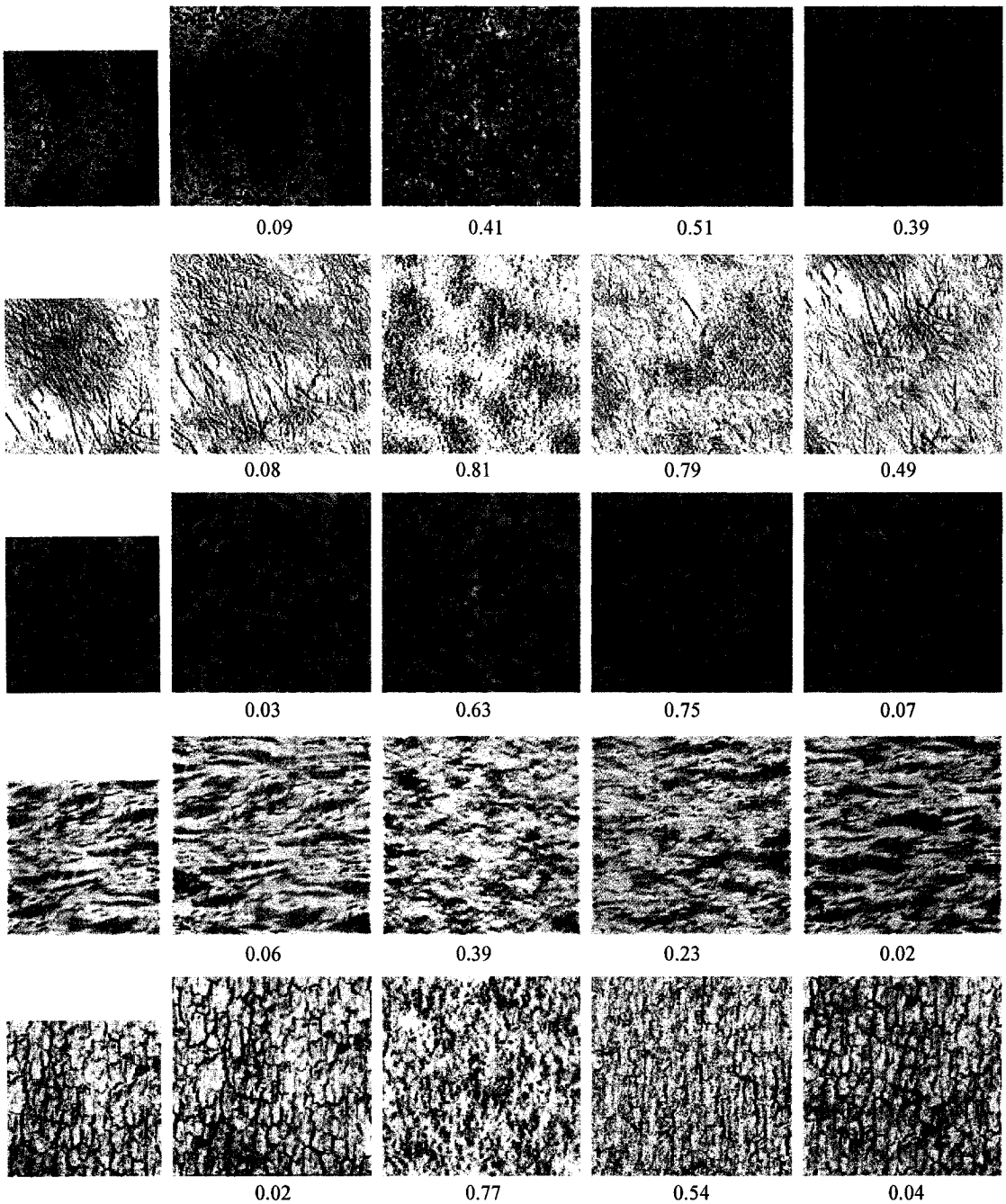


Figure 4-8: The comparison of results of our approach (column 2) with Heeger and Bergen's algorithm (column 3), Wei and Levoy's algorithm (column 4), and Liang et al.'s algorithm (column 5), where the input textures are in column 1.

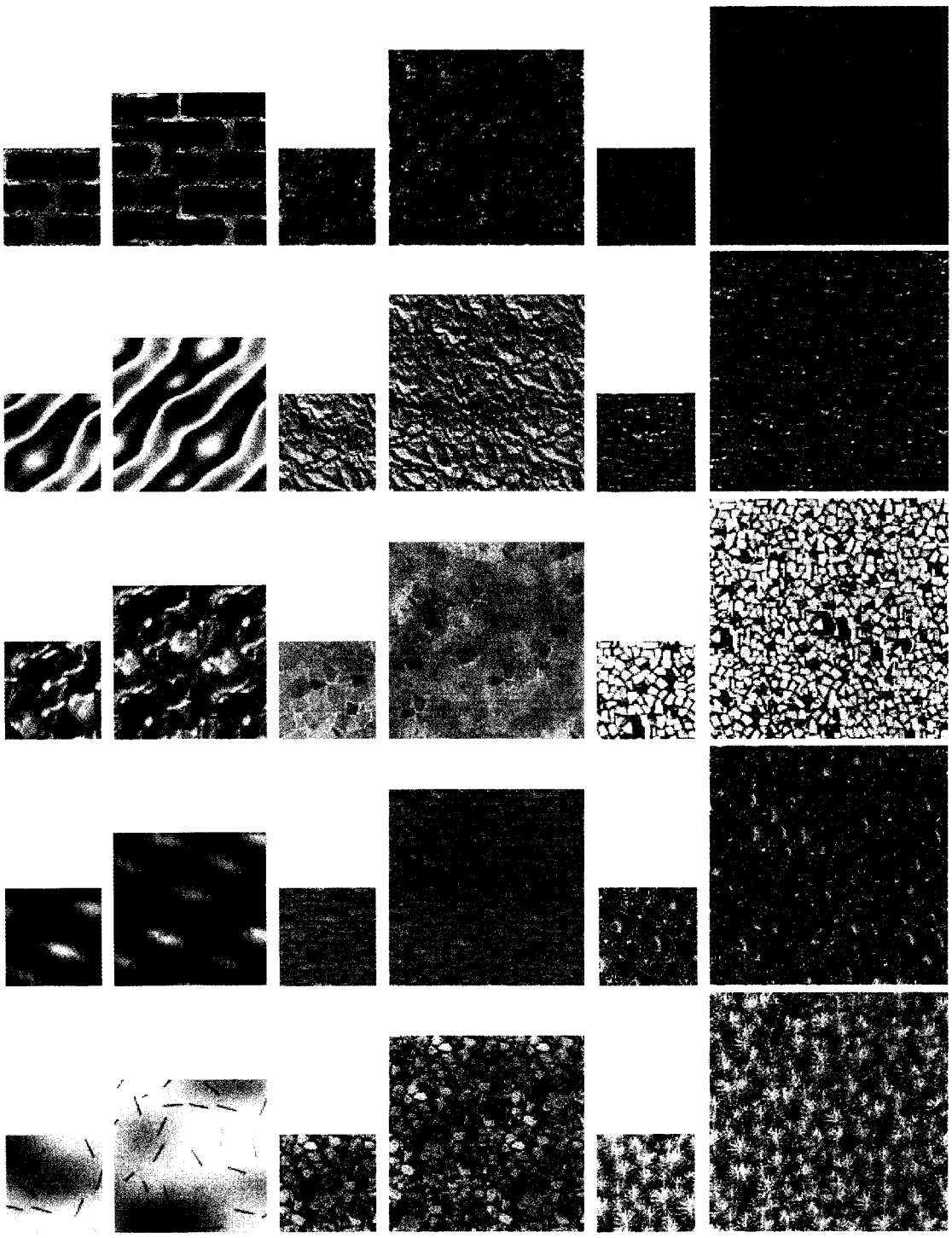


Figure 4-9: Examples of BGLAM-based 2D texture synthesis. The smaller image in each pair is the input texture (size 64x64), and the larger image is the synthesized texture. The

sizes of output texture in column 2, 4, and 6 are 100×100, 128×128, and 156×156, respectively. Since the textures in the first column contain large structures, 120 BGLAMs calculated over a neighborhood system of size 11 are used to generate the output in the second column. The output textures in the 4th and 6th column are generated using 48 characteristic BGLAMs.

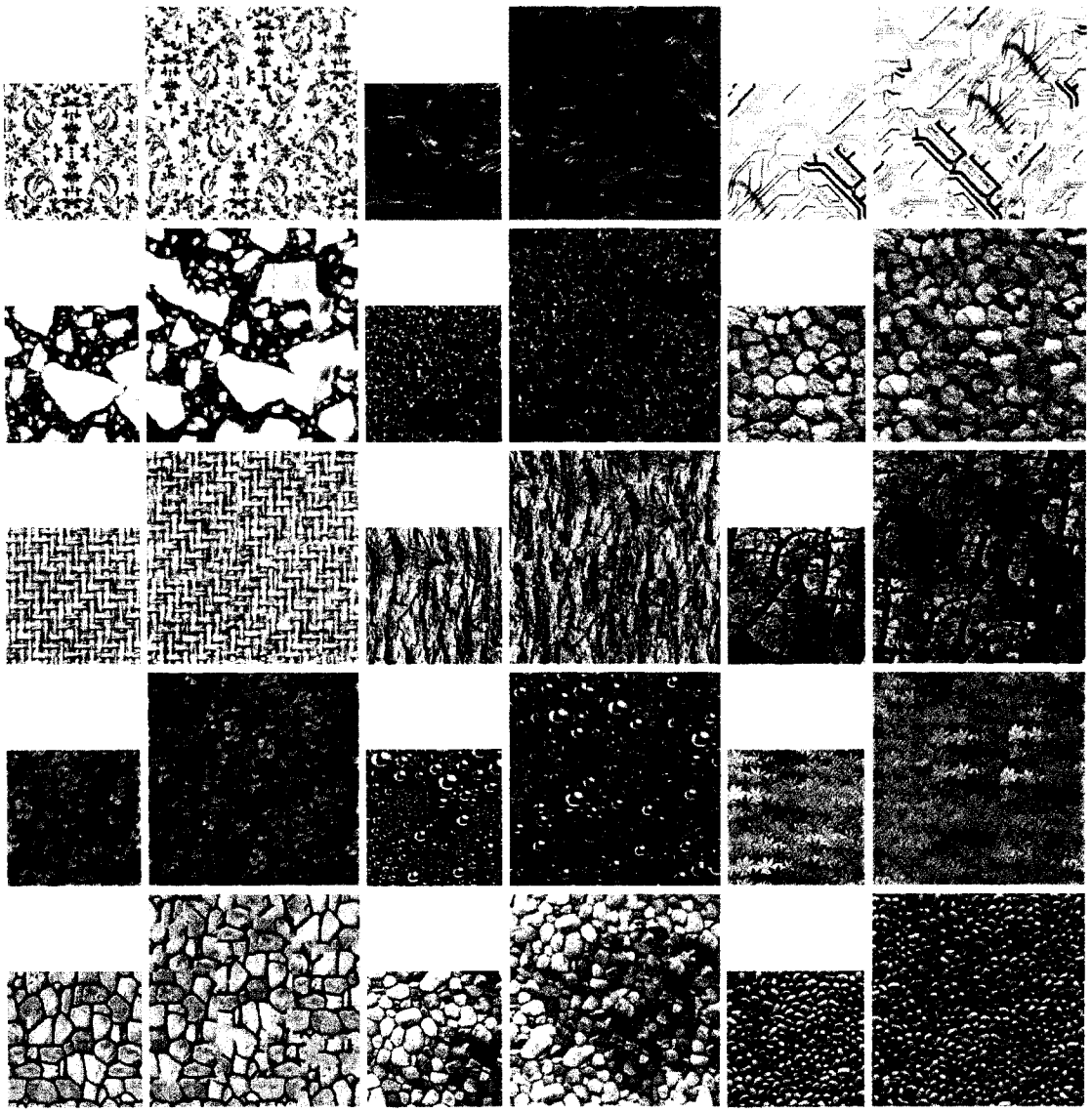


Figure 4-10: Examples of BGLAM-based 2D texture synthesis. The smaller images are the input textures (size 128×128), and the larger images are the synthesized textures

(size 200×200) that are generated using 80 BGLAMs calculated over a neighborhood system of size 9.

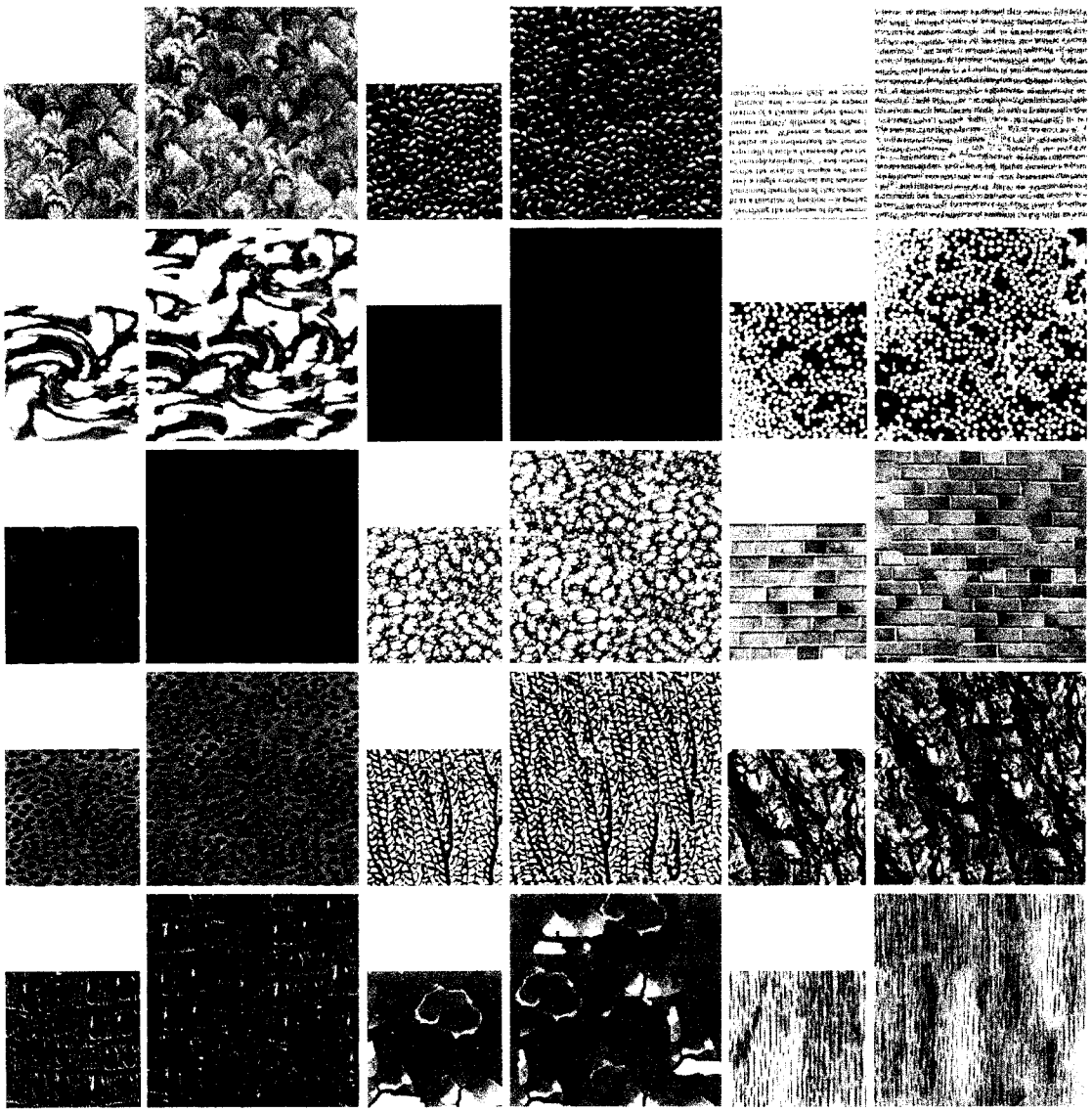
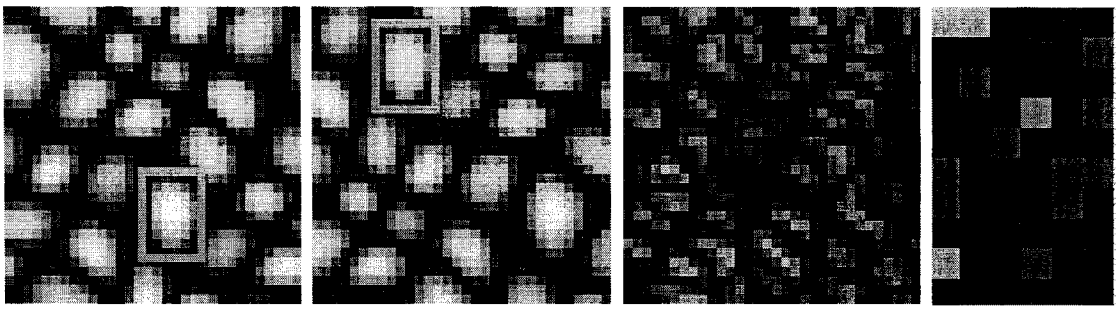


Figure 4-11: Examples of BGLAM-based 2D texture synthesis. The smaller images are the input textures (size 128×128), and the larger images are the synthesized textures (size 200×200) that are generated with 80 BGLAMs calculated over a neighborhood system of size 9.

4.4.3 Synthesis Results

Examples of texture synthesis using BGLAMs with various input textures are shown in Figure 4-9, Figure 4-10, and Figure 4-11. The smaller image in each pair is the input texture, and the larger image is the synthesized texture. More results can be found at the author's webpage [149]. The experimental results have shown that a broad range of textures can be faithfully synthesized using our approach.



Input | Output | Difference of Input and output | Difference of the highlighted part of input and output

Figure 4-12: An example of a synthesized texture with duplication effects. Both the input (it is enlarged in the figure) and output have the same size (32×32). The synthesized texture is generated by the algorithm described in Figure 4-4 with 120 BGLAMs and a threshold value 0.001. The difference image of the input and output shows that the input and output is not identical. In addition, the difference image (enlarged) of parts of the input and output in highlighted windows (where the duplication effect occurs) shows that the two parts are not identical.

Some discussions on the duplicate effect from parts of the input in the synthesized texture are given as follows. As shown in some examples (e.g. the result for the first input in Figure 4-7 and the result for the input in row 4 and column 4 in Figure 4-11), it seems that parts of the output texture are directly duplicated from the input; while in fact it is

not. This effect is more noticeable when a very small threshold value (e.g. 0.001) is used in the sampling of our algorithm. Our method generates a synthesized texture by iteratively modify the output such that the BGLAMs of the output match those of the input, it is possible that parts of the output look like duplicated from the input if the threshold value is very close to zero. However, those parts are not exactly the same. Figure 4-12 gives an example of an input texture, its synthesized texture of the same size by BGLAMs, the difference image of the input and output, and the difference image of the highlighted parts of the input and output where the duplication effect occurs.

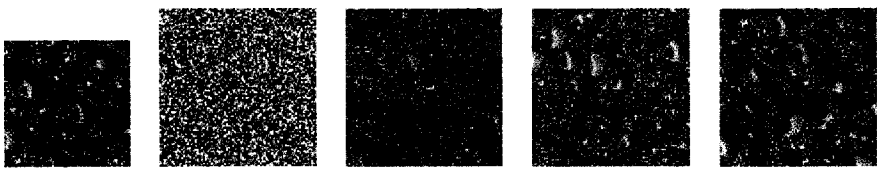


Figure 4-13: An example using BGLAM-based distance measure to evaluate the synthesized results against the input.

4.4.4 Evaluating Synthesis Results

One significant advantage of the BGLAM-based approach for 2D texture synthesis over existing approaches is that the BGLAM-based distance measure defined in Eq. 4.1 can be used to evaluate the synthesis result to determine whether or not the output looks similar to the input. By our experimental results, we found that if two texture images have a distance value greater than 1.0, then they are dissimilar. If the value is below 0.1, then the output is assured similar to the input. However, if the distance value is between 0.1 and 1.0, then the similarity between the two textures is difficult to determine, in this case we consider the output with a distance value below 0.5 a success

and a failure otherwise. This observation is made by our extensive experiments. Figure 4-13 gives an example to demonstrate this point. Note that in Figure 4-8, each output texture has a number beside it to show its BGLAM-distance to the input.

4.4.5 Running Time and Acceleration

On the running time, for an input color texture sample of size 64×64 and an output color texture of size 128×128, the average running time is about 30 minutes on a 1.4GHz Pentium 4 PC running Windows XP Professional.

For acceleration, we extend our algorithm so that it can perform texture synthesis in multiresolutions, similar to the non-filter-based multiscale method used in Paget's work [129], to build the multiresolution representation of a given image. With a multiresolution scheme of 4 levels and 24 BGLAMs used for each level, the running time is reduced to about 2-3 minutes. For color images, our algorithm is extended to synthesize the three independent color channels in parallel after the step of color-space transformation as described in Section 4.3.5. In this case, the above running time can be further reduced to about 1 minutes.

4.5 Limitations and Future Work

One limitation of the current implementation of the BGLAM-based 2D texture synthesis algorithm is the gray level update scheme during the sampling as described in Section 4.3.3. It is quite possible that after a few iterations, the number of candidates of possible gray levels for a target pixel is less than 3, which may sometimes cause the gray

level values of pixels in the output texture to quickly converge to local minima, and thus generate visible seams in the output textures as shown in Figure 4-14. In this case, fortunately, the BGLAM-based distance measure between the output and the input cannot decrease any further, and a large distance value, normally above 0.5, is returned to indicate a failure (see Figure 4-14). Future research should be carried out to address this problem. One possible solution is to extend the current single-point search scheme to a multiple-point search scheme during sampling so that the convergence to the local minima can be avoided as much as possible. Since genetic algorithms [66, 153] are well suited for searching in multiple directions, it is an interesting future research topic to explore using genetic algorithms to address this local minima problem.

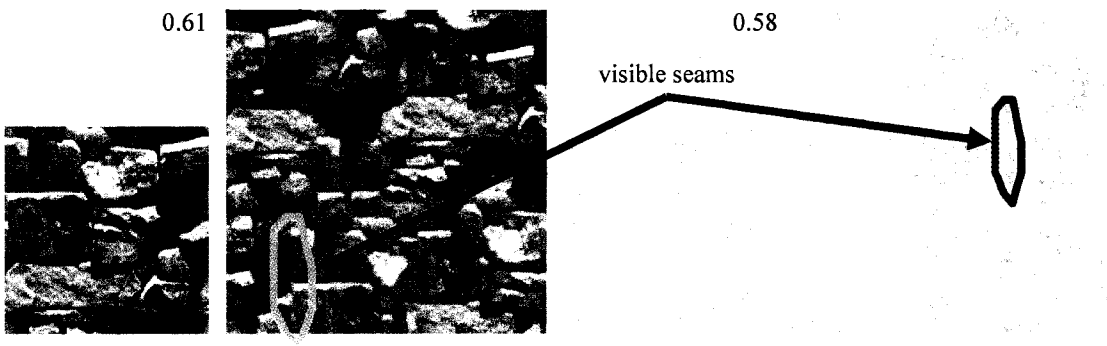


Figure 4-14: Example of visible seams in the synthesized textures. The visible seams are located within the areas bounded by the dashed lines. The number beside each output texture is its distance measure calculated using Eq. 4.1. Since those values are greater than 0.5, the output textures are considered as failures based on the evaluation criterion described in Section 4.4.4.

Another interesting future work is the application of our method to evaluating synthesized textures from various existing approaches, such as Wei and Levoy's, Liang et

al.'s [106] and Kwatra et al.'s [101], and choosing the best one. Other future works include the comparison of BGLAM distance measure with other existing distance measures, and the study of the sensitivity of the threshold values used in the BGLAM distance function for measuring the similarity of texture samples and for evaluating the synthesis results.

4.6 Summary

In this chapter, a new 2D texture synthesis approach, called aura 2D texture, is proposed. Given an input texture, our algorithm first calculates a set of independent BGLAMs to represent the texture, and then generates the synthesized texture by sampling only the BGLAMs of the input. The experimental results show that the new technique can successfully synthesize a wide range of textures and is comparable to several existing techniques. In addition, based on a new distance measure defined by BGLAMs, our technique is able to automatically evaluate the results and determine whether or not the output is a successful synthesis of the input. To our best knowledge, none of the existing techniques has the ability to evaluate their synthesis results. In the next chapter, we present a new method for synthesizing solid textures using BGLAMs.

Chapter 5

BGLAM 3D Texture Synthesis

5.1 Introduction

In computer graphics and computer games, 3D texture synthesis has been widely recognized as an important tool in generating realistic textures for rendering complex graphic scenes. Recent advances in 2D texture synthesis [3, 49, 50, 81, 101, 106, 190] have ignited the development of many successful techniques for generating surface textures from input samples [20, 43, 108, 179, 191, 204]. Although a wide range of textures can be synthesized in 2D, there is still a lack of techniques in generating 3D textures. When 2D textures are used in texturing 3D objects, the following disadvantages are found: (1) the distortion problem on large-curvature surfaces, and (2) non-reusable – textures generated for one surface cannot be used for other surfaces. The second limitation makes 2D surface textures difficult, if not impossible, to be used in procedural shaders [47].

To overcome the above problems, solid textures [132, 134] can be used. A solid texture is considered as a block of colored points in 3D space to represent a real-world material, for example, a wood trunk. Once the solid texture is available, any given 3D object can be textured by carving the object out of the volumetric data. Since solid textures define colors for each point in 3D space, they avoid the problems of distortion and discontinuity. However, solid textures are far more difficult to obtain than 2D textures; there is no easy way to obtain solid textures from real-world materials. Over the

last two decades, procedural techniques and image-based techniques have been developed to generate solid textures. In procedural approaches [47], procedures are designed and called to generate solid textures with the surface appearance of realistic objects, such as wood, stone, smoke, fire, fluid, cloud, etc. However, these techniques can model only a limited range of textures. In addition, the procedures are difficult to understand and control because there are many parameters in the procedures and these parameters are not intuitive for a user to determine their appropriate values. To address these problems, a number of researchers have developed image-based techniques [41, 42, 81, 86, 103, 188] for synthesizing solid textures from input samples, and appealing results have been obtained. Unfortunately, some of these techniques are not fully automatic, which involve nontrivial user interactions [41, 86]; while others may apply to only limited types of textures [42, 81, 86 , 103, 188].

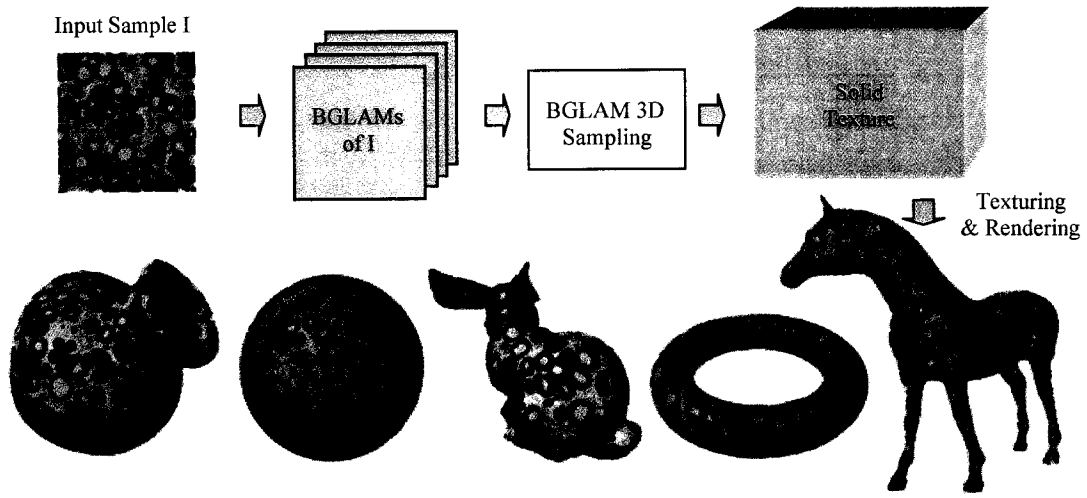


Figure 5-1: An overview of BGLAM 3D textures.

In this chapter, we present a new technique, called *aura 3D textures*, for synthesizing solid textures from input examples using BGLAMs. Our method is fully

automatic and requires no user interaction in the process. In theory, our method can take any number of input samples. As shown in Figure 5-1, given one or more input textures, our method first characterizes each input sample as a set of special aura matrices called BGLAMs that are introduced in Chapter 3. Once the BGLAMs are calculated, the input will not be needed. A solid texture is generated by sampling the BGLAMs of the input constrained in multiple view directions. The details of the aura 3D sampling are described in Section 5.3.1. After the solid texture is generated, any given object can be textured by the solid texture using a shader.

We have compared our algorithm with two recently proposed algorithms: Wei & Levoy's [188]; and Jagnow et al.'s [86]. The experimental results show that our method outperforms Wei & Levoy's and is comparable to that of Jagnow et al.'s. However, the latter method involves extensive user interactions in designing appropriate 3D shapes as well as in estimating the correct cross sectional profile; while our method is fully automatic with no user interactions in generating solid textures. In addition, their method can take a single input only; while ours can generate solid textures from multiple inputs.

To test the accuracy of our aura 3D texture approach, we present an evaluation method based on extensive user studies in Section 5.6. To avoid manual paper work, we have designed a GUI-based system to collect data and to perform the evaluation efficiently. The evaluation results show that our algorithm can generate faithful results for a wide range of textures, including both stochastic and structural textures, with an average successful rate of 76.4%.

5.2 Related Works

In 3D texturing, there are four ways to generate synthetic textures onto 3D surfaces: texture mapping, procedural texturing, image-based surface texturing, and image-based solid texturing. Texture mapping [79] is the earliest approach to generating synthetic textures on surfaces of computer-generated objects. Since Blinn's work [12], various techniques [85, 99, 170, 187, 202, 205] have been developed to synthesize high quality textures efficiently on 3D surfaces. In general, texture mapping suffers the well-known problems of distortion, discontinuity, and unwanted seams.

The second approach is called procedural texturing [47]. Since the seminal works of Cook [28], Peachey [132], and Perlin [134], procedural techniques have been widely accepted in the computer graphics community. In most existing techniques, storage-efficient procedures built on basis functions [29, 102, 134, 199] are used to generate high quality 3D textures with no distortion and no discontinuity. Some techniques use the reaction-diffusion processes [56, 180] to generate biological patterns, e.g. zebra stripes and cellular patterns, that are found on animal skins. The disadvantages of procedural texturing include: (1) only limited types of textures can be modeled, (2) the design of procedures is based on the experience of the designer and is largely a manual process, and (3) the parameters of a texturing procedure are difficult to tune or estimate [153].

The third approach is the image-based surface texturing developed by a number of researchers recently. Wei & Levoy [191], Ying et al. [201], and Turk [179] have concurrently extended Wei & Levoy's 2D texture-synthesis algorithm [190] to synthesize textures onto arbitrary mesh surfaces. Using feature-based warping and texton masks, Zhang et al. [204] have successfully synthesized progressively-variant textures onto 3D

surfaces from multiple input samples. In Chen's work [20], shell texture functions are used to synthesize realistic textures with translucency variations on surfaces from either 2D or 3D samples, e.g. a block of CT scan. Recent research works [108, 177] have also been done in generating bidirectional texture functions (BTF) onto 3D mesh surfaces. Compared with procedural texturing, image-based surface texturing can synthesize a wide range of textures. However, the approach may suffer the distortion problem on surfaces where the curvature is large. Another problem of the approach is that textures generated for one surface cannot be used for other surfaces. This limitation makes the techniques difficult to be used in procedural shaders [47].

To combine the advantages of procedural texturing and image-based 2D texture analysis and synthesis, a number of researchers have developed techniques for generating solid textures from input samples, which we call image-based solid texturing. Different from image-based surface texturing, these techniques synthesize a volumetric texture data from input samples. Once the volumetric data is generated, it can be used to texture different objects. In Heeger and Bergen's work [81], homogeneous and stochastic 3D textures are successfully generated by matching the histogram of a volumetric data with that of the input sample from coarse to fine resolutions. However, their approach fails for structural textures. To address this problem, Dischler et al. [42] propose a method based on spectral and histogram analysis to synthesize a wider range of solid textures from input samples. Although only a limited range of textures can be modeled, Dischler et al.'s method [42] is the first approach capable of generating structural solid textures such as wood and marble. By analyzing and extracting parameters from input images, Lefebvre and Poulin's algorithm [103] is able to synthesize some structural textures such as wood

and regular tiles. Wei [188] and Paget [125] have extended their respective 2D texture synthesis algorithms [129, 190] to generate structural solid textures as well as stochastic textures. However, both approaches work for only a limited range of textures. In Jagnow et al.'s work [86], a stereology-based approach is presented to successfully generate solid textures on some texture classes, e.g. marble-like textures. In their approach, in order to generate the correct results, extensive user interactions are required in creating 3D particles of desired shapes and of required distributions. Dischler and Ghazafarpour [41] have also developed an interactive image-based framework for synthesizing structural solid textures of certain types.

Our work belongs to the category of image-based solid texturing. In particular, we present a BGLAM-based framework for synthesizing solid textures from 2D input samples. Additionally, we describe how to evaluate the results of our method using extensive user studies based on a carefully designed GUI-based system. The new approach is motivated by and extended from the work on 2D texture analysis and synthesis using BGLAMs as described in the last two chapters. Our work is most related to Heeger and Bergen's [81] and Dischler et al.'s methods [42]. However, the texture analysis process of our method is done using BGLAMs rather than using gray level histograms [81] or spectrum in the frequency domain [42] (Note: Dischler et al.'s method also uses histogram-analysis to characterize textures). In the synthesis process, our method generates solid textures by sampling only the BGLAMs of the inputs. On the other hand, Heeger and Bergen's method needs filters to build pyramids for the input and output, and the synthesis results of their method heavily depend on the selection of filters. While there is no need for filters in Dischler et al.'s approach, it cannot synthesize

textures with edges [42]. Both Heeger and Bergen's and Dischler et al.'s methods fail for large structural textures such as bricks; while our method can generate appealing results for such structural textures as shown in the experimental section.

5.3 The Approach

The general flow of our BGLAM 3D texture synthesis is given in Figure 5-2. Our approach can take a single input sample or multiple input samples. Given an input texture sample, as shown in Figure 5-2, our method first characterize the input so that the given sample texture can be well represented. As demonstrated in previous chapters, a texture image can be accurately represented by and faithfully reconstructed from BGLAMs, we use BGLAMs to characterize and parameterize a texture sample. In BGLAM 3D sampling, a solid texture is generated by matching the BGLAMs of volumetric data's slices with the BGLAMs of the input in multiple view directions, e.g. the positive directions of the x, y, and z-axes of the 3D coordinate system. Once the solid texture is generated, a shader can be used to texture different objects. The details of our approach are described as follows.

5.3.1 BGLAM 3D Sampling

For illustration purposes, we describe the BGLAM 3D sampling in the case of three input samples. The situation for fewer or more input samples can be handled similarly. As shown in Figure 5-2, in the beginning, the BGLAMs of input samples are calculated using the algorithm described in Section 4.3.1 and a volume of white noise is initialized. The BGLAMs of each input is used to define constraints in a specific view

direction during sampling such that the final synthesized volume will have similar texture to the corresponding input sample when a cross section perpendicular to the view direction is cut from the volume. In Figure 5-2, for example, the aura matrices of input I_x are used to constrain the sampling in the direction of *x-axis* to make sure the slices of the output volume in that direction look similar to I_x. For the case of single input sample, the aura matrices constrained in a view direction is calculated either from the input or from the rotated version of the input.

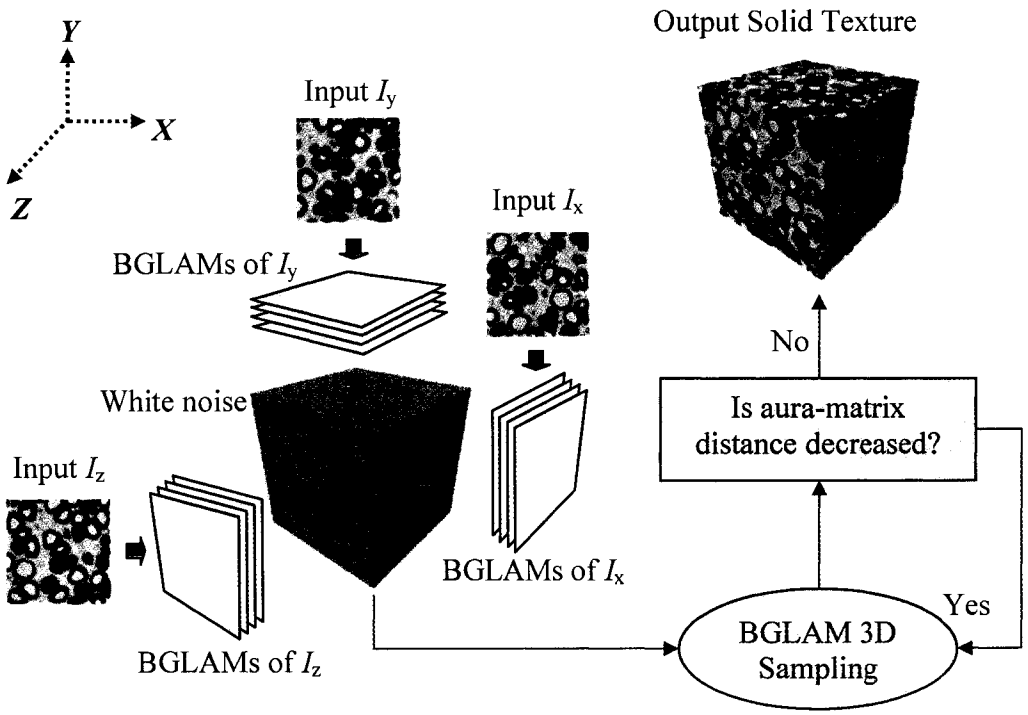


Figure 5-2: The general flow of BGLAM 3D texture synthesis.

The view directions for adding constraints can be arbitrary in our algorithm. For example, to generate a solid texture of a regular octahedron (a polyhedron with eight equilateral triangles as faces), eight input samples can be placed along the norm

directions of the octahedron's eight faces. For the purpose of illustration, the view directions in Figure 5-2 are demonstrated as in the positive directions of the *xyz*-axes.

After initialization, the algorithm iteratively modifies the noise such that the aura-matrix distances defined in Eq.5.1 in Section 5.2 between the *xyz*-slices of the volume and the input samples is decreased as much as possible. The intuition behind this is as follows: two textures are guaranteed to look similar if their corresponding BGLAMs are close enough as demonstrated in Chapter 3 (also see [151]). We use the weighted-sum distance (see Eq. 5.1) because we want to make sure that the points in the volume closer to a view direction are more likely synthesized by the input sample constrained in that direction and that there is a smooth transition between textures of different views. The calculation of weights, which is discussed later, depends only on points in the volume and the view directions and thus is automatically done by the algorithm.

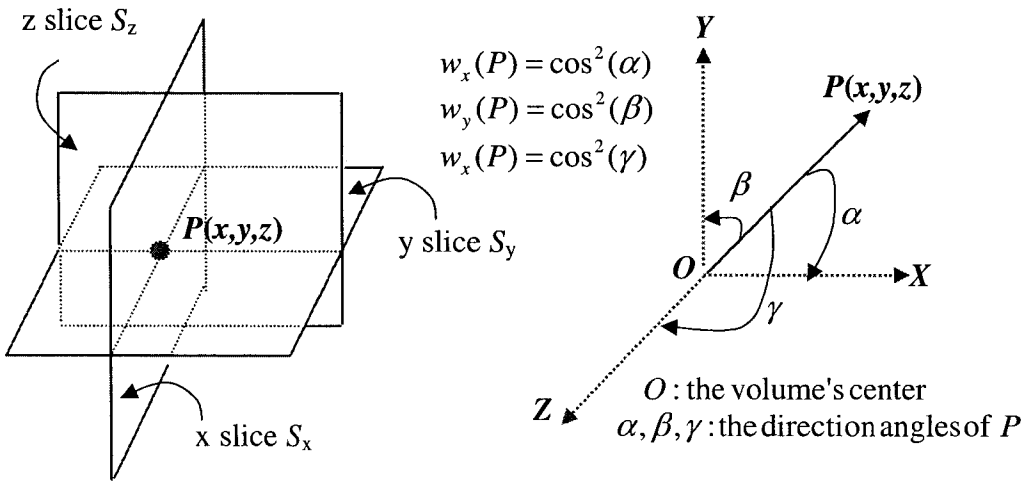


Figure 5-3: The view slices S_x, S_y, and S_z at point $P(x, y, z)$ and its direction angles α, β, and γ.

During an iteration of sampling, each point in the volume is visited *randomly* once, and its color is modified so that the distance defined in Eq.5.1 between the BGLAMs of the *view slices* (see Figure 5-3) of the volume and the BGLAMs of the input samples is decreased. More precisely, when visiting a point, the algorithm first finds the candidate set of all colors different from the current color that decrease or at least do not increase the aura-matrix distance. Then, it randomly chooses a color from the candidate set as the color of the point. Note that even when a color does not change the distance, i.e. at the same distance as the current color, the algorithm still includes it into the candidate set in order to increase the randomness in the output. It is possible that the candidate set is empty at the end of search, which implies that any color different from the current color will increase the distance. In such a case, the point retains its current color and the algorithm goes to process the next point in the volume. When the distance is below a predefined threshold or there is no change in colors in any point of the volume, the algorithm returns the volume as the final solid texture.

5.3.2 Aura-Matrix Distance

The aura-matrix distance used in the aura 3D sampling is defined by

$$d = d(\boldsymbol{V}, I_x, I_y, I_z) = \frac{1}{|V|}\sum_{P \in V}[w_x(P) * d(S_x, I_x) + w_y(P) * d(S_y, I_y) + w_z(P) * d(S_z, I_z)], \quad (5.1)$$

where $|V|$ is the total number of points in volume $\boldsymbol{V}$; $w_x(P)$, $w_y(P)$, and $w_z(P)$ are the weights calculated from the direction cosines of α, β, and γ of point P in V as shown

in Figure 5-3; and $d(S_x, I_x)$ is the distance between the BGLAMs of view slice S_x and the BGLAMs of input sample I_x. Given two images X and Y, let $\boldsymbol{A}(X) = \{\boldsymbol{A}_k \mid 0 \le k \le m-1\}$ and $\boldsymbol{A}(Y) = \{\boldsymbol{B}_k \mid 0 \le k \le m-1\}$ be their corresponding normalized BGLAMs, then the BGLAM distance $d(X,Y)$ is defined in Eq. 3.13 in Section 3.4, which is restated below for ease of reference:

$$d(X,Y) = d(\boldsymbol{A}(X), \boldsymbol{A}(Y)) = \frac{1}{m}\sum_{k=0}^{m-1} \| \boldsymbol{A}_k - \boldsymbol{B}_k \|, \tag{5.2}$$

where for a given matrix $\boldsymbol{A} = [a_{ij}]_{0 \le i,j \le G-1}$, $\| \boldsymbol{A} \| = \sum_{i,j=0}^{G-1} | a_{ij} |$, and an aura matrix $\boldsymbol{A} = [a_{ij}]_{0 \le i,j \le G-1}$ is *normalized* if $\sum_{i,j} a_{ij} = 1$. Since two images X and Y may have different sizes, the aura matrices must be normalized to make sure that there is no bias in the values of $d(S_x, I_x)$, $d(S_y, I_y)$, and $d(S_z, I_z)$ when the distance defined by Eq. 5.1 is calculated.

As shown in the right of Figure 5-3, when a point in the volume is closer to a view direction, e.g. the x-axis, there is more chance during sampling for the point to be colored by the input sample constrained in that direction. Since $\cos(\alpha)$, $\cos(\beta)$, and $\cos(\gamma)$ are continuous functions, there is a smooth transition in the synthesized textures from one view direction (e.g. the x-axis) to the other (e.g. the y-axis). For a given point $P(x, y, z)$ in the volume $\boldsymbol{V}$, let $O(x_0, y_0, z_0)$ be the center of $\boldsymbol{V}$, then the weights can be calculated by

$$w_x(P) \overset{def}{=} \cos^2(\alpha) = \frac{(x - x_0)^2}{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}$$

$$w_y(P) \overset{def}{=} \cos^2(\beta) = \frac{(y-y_0)^2}{(x-x_0)^2+(y-y_0)^2+(z-z_0)^2}. \quad (5.3)$$

$$w_z(P) \overset{def}{=} \cos^2(\gamma) = \frac{(z-z_0)^2}{(x-x_0)^2+(y-y_0)^2+(z-z_0)^2}$$

One can verify that $w_x(P)+w_y(P)+w_z(P)=1$. When P coincides with O, we let $w_x(P)=w_y(P)=w_z(P)=1/3$.

BGLAM-Based 3D Texture Synthesis

Input:
I_x, I_y, I_z ← sample texture images.
ε ← a given small positive number in (0,1)
Output:
V ← the synthesized solid texture.

Begin
1 Initialize V as a volume of random noise.
2 **While** $d=d(V, I_x, I_y, I_z) \geq \varepsilon$ **do**
  2.1 While there are unvisited points in V, randomly choose an unvisited point P **do**
    grayLevel(p) ← *bglamBased3DRandomSampling*(P, d, I_x, I_y, I_z,V).
  End of while
End of while
End of begin

bglamBased3DRandomSampling(P, d, I_x, I_y, I_z,V)
b.1 C=empty (the candidate set of gray levels for point P).
b.2 For each gray level $j = 0$ to G - 1 **do**
  b.2.1 $S_x(j)$, $S_y(j)$, $S_z(j)$ ← the view slices of V at point P when P has gray level j.
  b.2.2 d_j ← $d(V, S_x(j), S_y(j), S_z(j))$.
  b.2.3 if $d_j \leq d$, then $C = C \cup \{j\}$.
b.3 If C is empty, **then** g ← the current gray level value of P,
  Else g ← a randomly chosen gray level from C.
b.4 Return g.

Figure 5-4: The pseudo code of the BGLAM-based 3D texture synthesis algorithm.

5.3.3 Algorithm

The pseudo code of the algorithm for aura 3D textures is given in Figure 5-4. The definition of distance d in the step 2 of the main algorithm is given in Eq. 5.1. In the step

b.2.1 of *bglam3DBasedRandomSampling*, the view slices of V at point P are defined as shown in Figure 5-3.

The major computation cost of the above 3D aura texture synthesis algorithm is in the two *while* loops in step 2. In an iteration of step 2 (i.e. step 2.1, which is one pass of visiting all points in V), a brute force method would perform fresh recalculations each time computing the aura matrices of the volume's view slices and the aura-matrix distance (see Eq. 5.1) with a cost of at least $O[m*np^2*G*(S+G^2)]$, where G is total number of gray levels in the input image, m the total number of BGLAMs used in the sampling, np the number of points in the volume, and S is the size of the view slices of volume V. A more efficient way is to perform an iterative update based on existing information, which can be done with a computation cost of $O(np*(m+S)*G)$ because when a pixel changes its gray level value, only its neighboring pixels are affected. The proofs of the above time complexities are given below.

[**Proof of the brute-force time complexity**] In the following, the time complexity for one iteration in step 2, i.e. one pass that goes through step 2.1 by visiting all points in volume V, is proved to be at least $O[m*np^2*G*(S+G^2)]$, where G is the total number of gray levels in the input image, m the total number of BGLAMs used in the sampling, np the number of points in the volume, and S the size of the view slices of volume V.

Let $T(2.1)$, and $T(b)$ be the time complexity for step 2.1 and procedure *bglamBased3DRabdomSampling*, respectively, then $T(2.1)$ is given by:

$$T(2.1) = np*T(b). \tag{5.4}$$

From the pseudo code of *bglamBased3DRandomSampling* given in Figure 5-4, we have:

$$
\begin{aligned}
T(b) &= T(b.1) + T(b.2) + T(b.3) + T(b.4) \\
&\geq O(1) + G*[T(b.2.1) + T(b.2.2) + T(b.2.3)] + O(1) + O(1) \\
&= G*[T(b.2.1) + T(b.2.2) + T(b.2.3)] \\
&= G*\{O(S) + O[m*np*(S+G^2)] + O(1)\} \\
&= G*O[m*np*(S+G^2)] \\
&= O[m*np*G*(S+G^2)]
\end{aligned}
\quad . \quad (5.5)
$$

In the above equation, the time for both step b.1 and step b.4 is a constant, and the time for step b.3 is at least $O(1)$. Step b.2.1 computes the view slices at a point in volume V, and takes a computation time of $O(S)$ to finish. Based on the definition of aura-matrix distance (see Eq. 5.1), one can prove that the time for step b.2.2 is $O[m*np*(S+G^2)]$ because the brute-force time for computing $d(S_v, I_v)$, $v = x, y, z$, is $O[m*(S+G^2)]$. Thus, we have:

$$
\begin{aligned}
T(2.1) &\geq np*T(b) \\
&= np*O[m*np*G*(S+G^2)]. \\
&= O[m*np^2*G*(S+G^2)]
\end{aligned}
$$

■

[Proof of the fast-version time complexity] Based on Lemma 3-5 and Algorithm 3-1, using an analysis method similar to the one used for the BGLAM-based 2D texture synthesis as described in Section 4.3.4 in Chapter 4, the distance d_j in step b.2.2 of the algorithm as shown in Figure 5-4 can be efficiently updated from distance d with a computation cost of $O(m)$. The time for step b.3 is at most $O(G)$. The time for step b.2.1 and step b.2.3 is $O(S)$ and $O(1)$, respectively. Thus, we have the following:

$$
\begin{aligned}
T(b) &= T(b.1) + T(b.2) + T(b.3) + T(b.4) \\
&\le O(1) + G*[T(b.2.1) + T(b.2.2) + T(b.2.3)] + O(G) + O(1) \\
&= G*[T(b.2.1) + T(b.2.2) + T(b.2.3)] + O(G) \\
&= G*[O(S) + O(m) + O(1)] \\
&= G*O(m+S) \\
&= O[G*(m+S)]
\end{aligned}
\qquad .(5.6)
$$

By Eq. 5.4 and Eq. 5.6, we have $T(2.1) \le np*O[G*(m+S)] = O[np*G*(m+S)]$. ■

5.3.4 Color Input Texture

For color input texture samples, one cannot simply apply the above basic algorithm to each of the RGB channels separately since the RGB components of a color image are dependent on one another. Before applying the basic aura 3D texture synthesis algorithm, a color-space transformation $\boldsymbol{T}$ based on the singular value decomposition technique [146] is used to transform the R, G, and B components of an color image into three independent components R', G', and B' in another color space. After this RGB-color-decorrelation step, the basic synthesis algorithm is applied to each of the independent color components R', G', and B' to generate three gray-scale solid textures in the transformed color space. Using the inverse transformation of $\boldsymbol{T}$, the final synthesized three gray-scale sold textures are transformed back into the RGB color space to produce the final synthesized color solid texture. The algorithm of transforming RGB color channels into independent color channels is given in Figure 4-5 in Chapter 4.

5.4 Acceleration

For acceleration, we extend our algorithm so that it can run texture synthesis in multiresolutions, similar to the pyramid method used in Heeger and Bergen's work [81]. However, from our experience, we find that the filtering process only complicates our

algorithm. Thus we have used a simpler non-filter-based method, called local decimation [129] to build the multiresolution representations of the input and output. For an input color texture sample of size 64×64 with 80 characteristic BGLAMs and an output volume of size $128\times128\times128$, the average running time in single resolution is about 10 hours on a 1.4GHz Penntium 4 PC running Windows XP Professional. With a multiresolution scheme of 4 levels and 24 BGLAMs used for each level, the running time is reduced to about 3 hours. For color images, our algorithm is further extended to synthesize the three independent color channels in parallel after the step of color-space transformation as described in Section 4.3.5. In this case, the above running time can be further reduced to about 1 hour. Once the solid texture is generated, a given object can be textured within seconds. An average runtime of 6 seconds is recorded in our experiments.

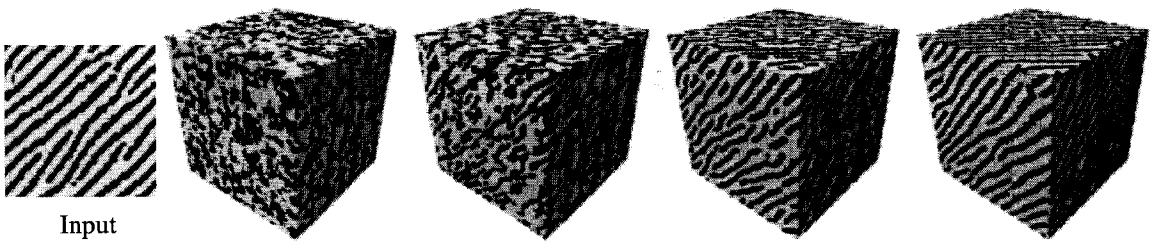


Figure 5-5: An example of aura 3D textures using different window sizes given under each output.

5.5 Results

The window size used to calculate BGLAMs in our algorithm is an important parameter that affects the synthesis results. In general, an input texture containing large structures or favorable orientations requires a relatively large neighborhood size. For a given input texture, different synthesis results can be generated with different window

sizes. Figure 5-5 gives an example texture and its solid textures generated by using windows of different sizes.

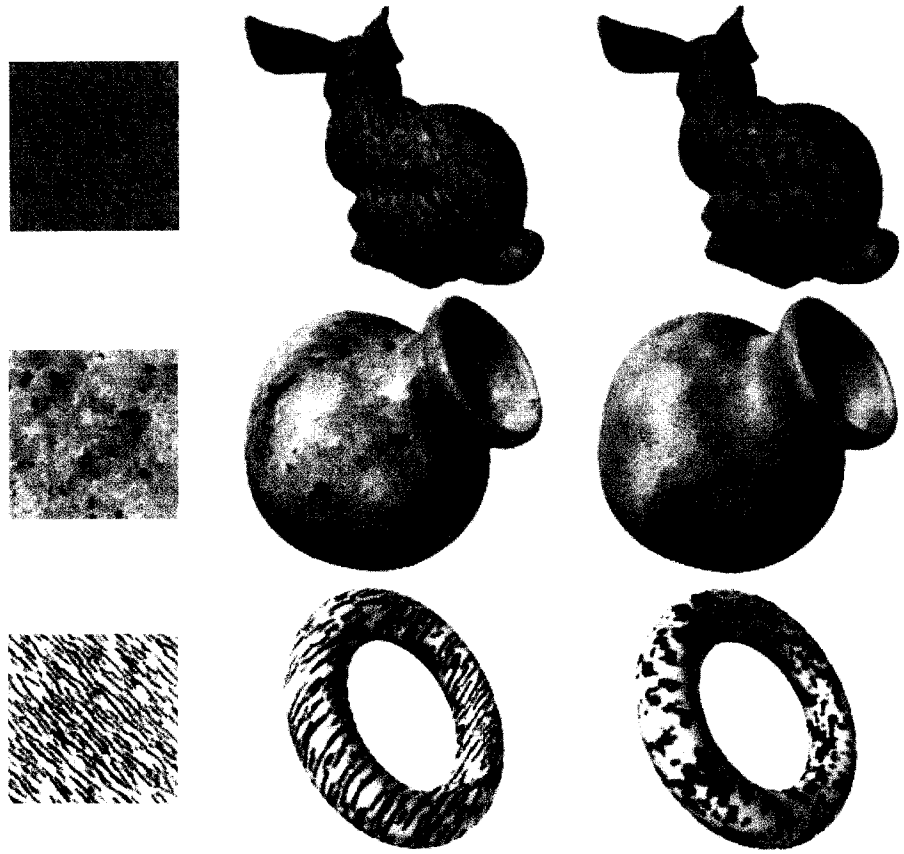


Figure 5-6: Comparison results of our method with Wei & Levoy's. The inputs (size 64×64) are shown in the first column, and the synthesis results (size $128\times128\times128$) are displayed in the same row as the corresponding inputs. Images in column 2 are the results of our algorithm, and images in the last column are the results of Wei & Levoy's.

We compare our aura 3D textures with two existing approaches. Figure 5-6 gives some comparison results of aura 3D textures with Wei & Levoy's approach [188]. Images in the first column are input samples of size 64×64, images in column 2 are results of our algorithm, and images in the last column are generated by Wei & Levoy's algorithm. For each input texture, a solid texture of size $128\times128\times128$ is generated

using our algorithm and Wei & Levoy's, respectively. In our algorithm, we use 80 BGLAMs, which are calculated over a 9×9 window, to characterize all input textures and to generate results shown in Figure 5-6 and in the rest of this section. Figure 5-7 gives some comparison results of aura 3D textures with Jagnow et al.'s algorithm [86], where images in column 2 are results of our algorithm, and images in the last column are results of Jagnow et al.'s. As shown in Figure 5-6, the results of our method are better than those of Wei & Levoy's algorithm. Compared to Jagnow et al.'s algorithm, although the result of our algorithm is not as good as theirs for the input in the last row, our algorithm generates better results for the inputs in row 1 and 2.

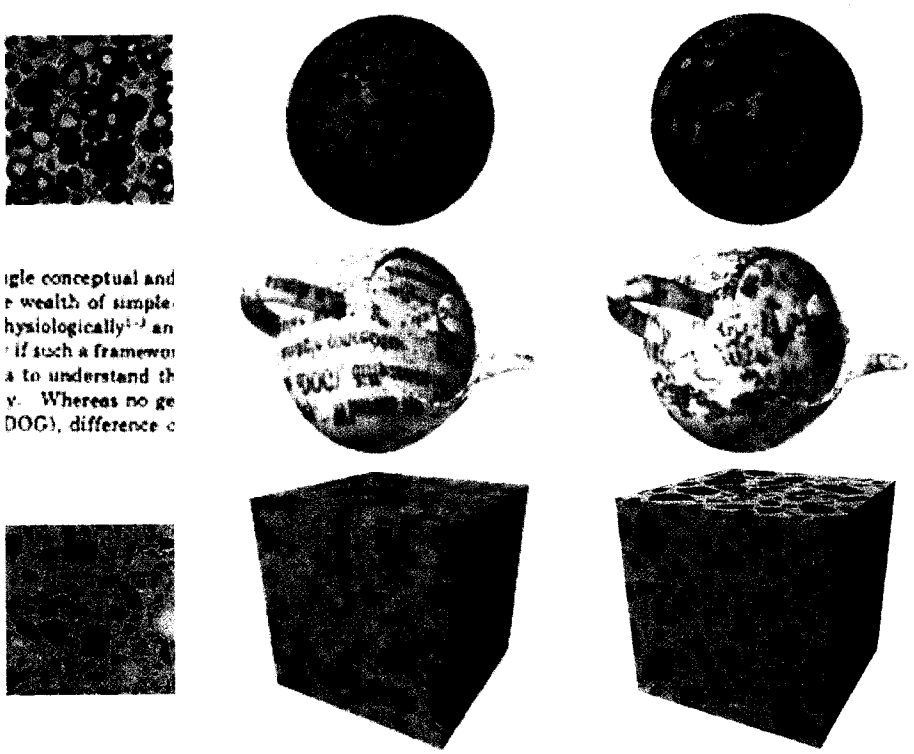


Figure 5-7: Comparison results of our method with Jagnow et al.'s. The inputs (size 64×64) are shown in the first column, and the synthesis results (size $128\times128\times128$) are

displayed in the same row as the corresponding inputs. Images in column 2 are the results of our algorithm, and images in the last column are the results of Jagnow et al.'s.

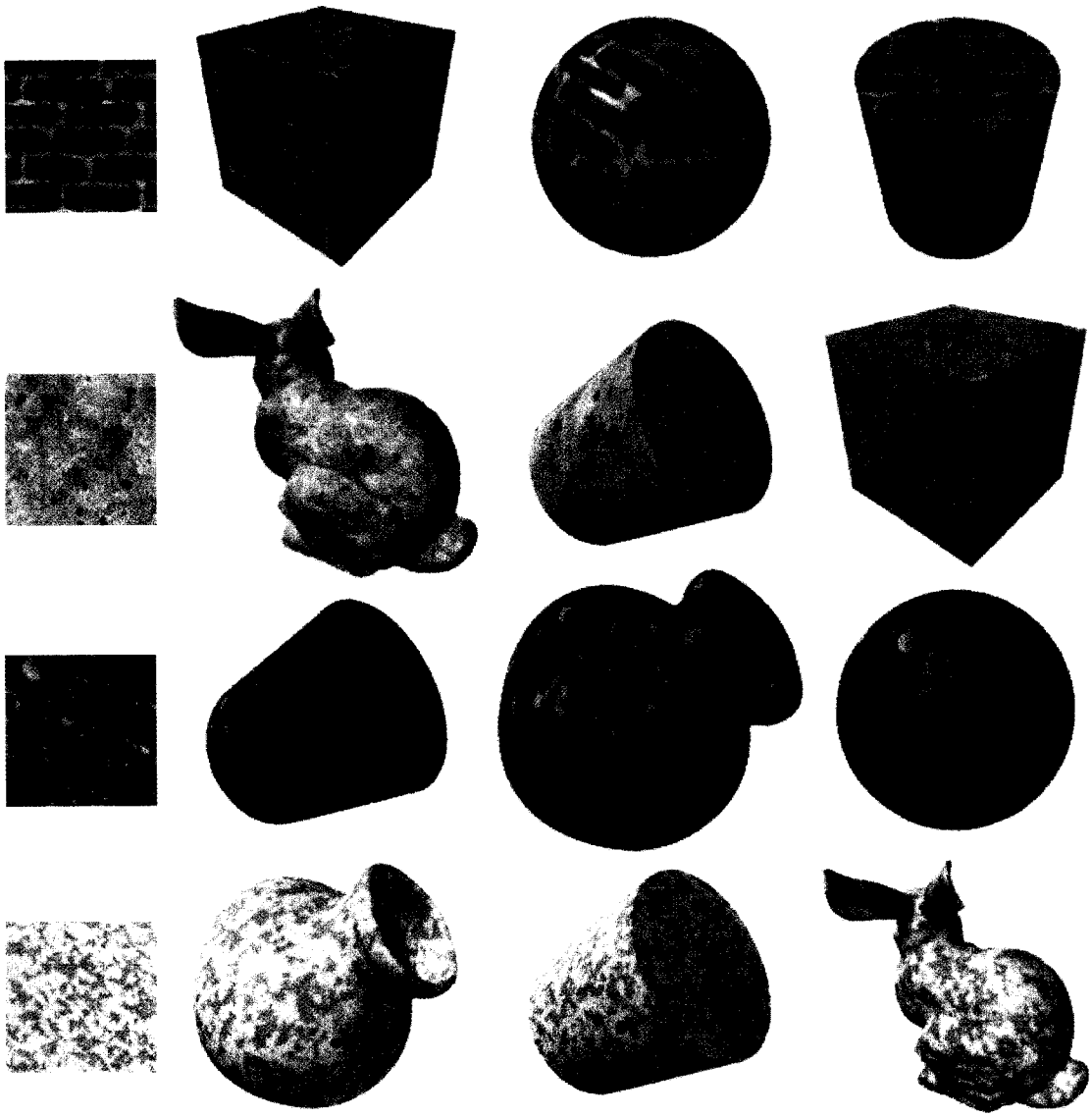


Figure 5-8: Results of aura 3D textures. Small patches are input samples and results of solid textures generated on the surfaces of different objects are displayed beside the corresponding inputs.

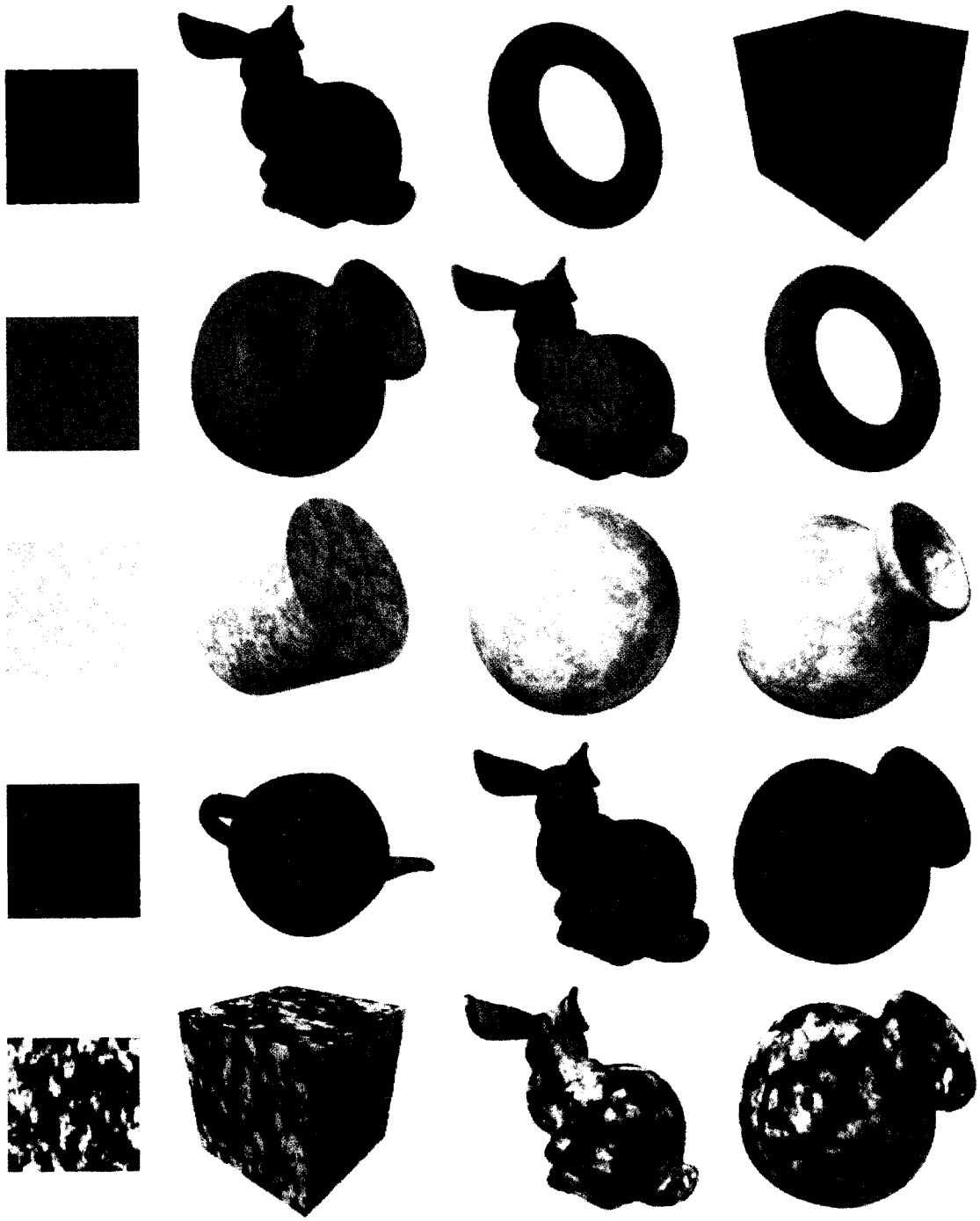


Figure 5-9: More results of aura 3D textures. Small patches are input samples and results of solid textures generated on the surfaces of different objects are displayed beside the corresponding inputs.

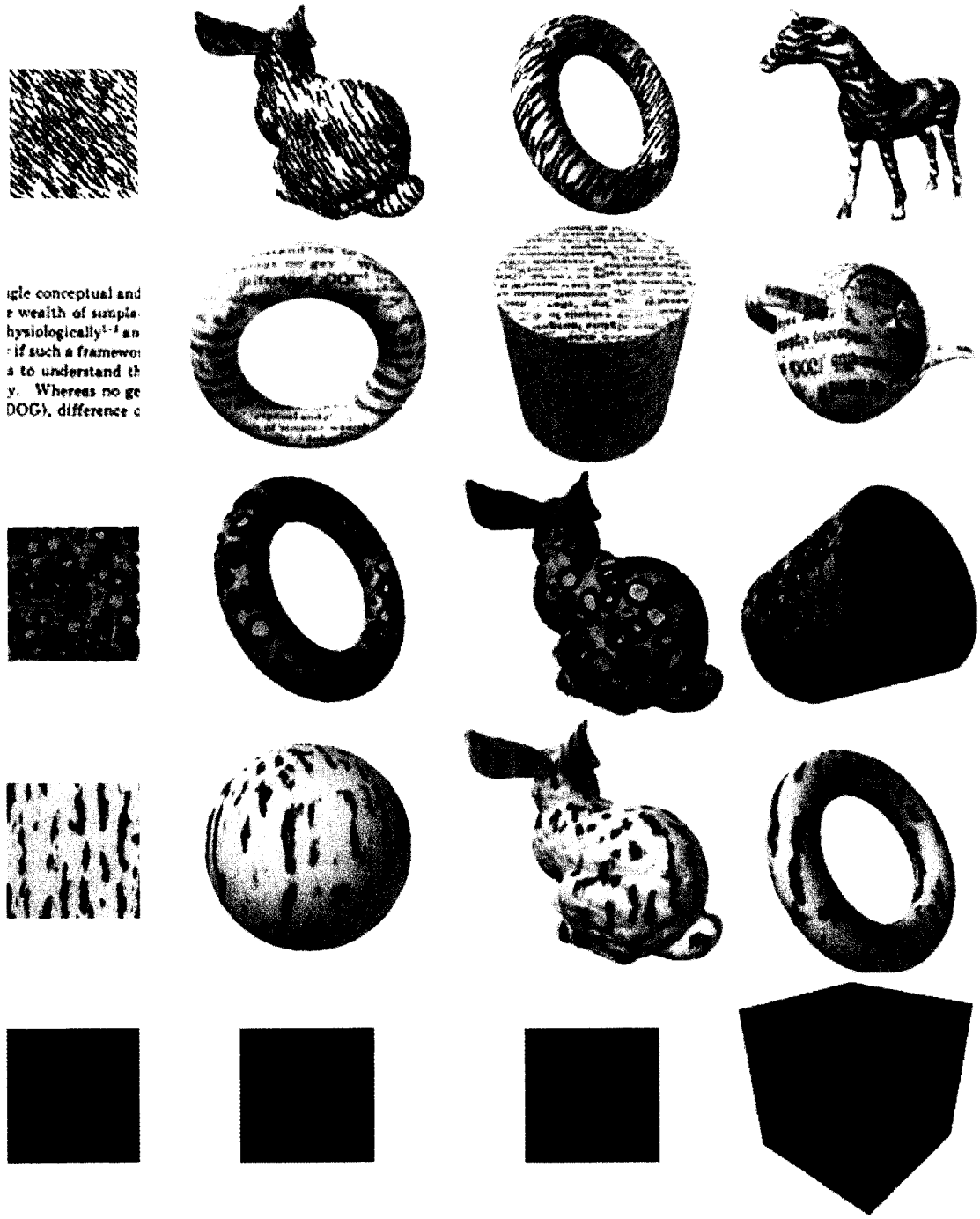


Figure 5-10: More results of aura 3D textures. Small patches are input samples and results of solid textures generated on the surfaces of different objects are displayed beside the corresponding inputs. In the last row, three input samples are used to generate the solid texture.

It is noteworthy that Jagnow et al.'s method requires a user to manually design and to edit 3D particles to match the texture profiles of a given sample. While it provides flexibility, it is nontrivial to design a complex texture. If the shapes of predesigned 3D particles do not match the profiles of input textures, the algorithm will likely generate incorrect results as the one shown in the second row in Figure 5-7. More results of our method can be found in Figure 5-8 - Figure 5-10, and at the author's webpage [149].

5.6 Evaluation

We present a method based on user studies for evaluating our aura 3D textures results. We only describe the algorithm for the case of single input textures. It is straightforward to extend the algorithm to multiple input textures. To test the accuracy of our aura 3D textures, it is reasonable to have the following two evaluation goals: (1) we test whether or not the slices of the solid texture in each constrained direction, i.e. the direction in which the BGLAMs of the input are used to constrain the aura 3D sampling, look similar to the corresponding input; and (2) we determine whether or not textures change smoothly between consecutive slices in any view direction, including view directions that are *not* used to constrain the aura 3D sampling.

Suppose a solid texture $\boldsymbol{V}$ of size $n \times n \times n$ is synthesized from an input texture. To test the first goal, for each direction in which the BGLAMs of the input are used to constrain the sampling, we obtain all n slices of $\boldsymbol{V}$ and randomly select m $(< n)$ of them. We mix the selected m slices with other texture images that are randomly drawn from a database of texture images (a database of over 2000 images is used in our experiments),

and display them in random order on the screen. Each display is evaluated by 18 people. Among them, 6 are researchers in the same research lab as the authors and have the knowledge in texture analysis and synthesis; the other 6 are graduate students in the same department and have some general knowledge in computer vision and image processing, and the remaining 6 are from outside of the department but in the same university and are in completely unrelated disciplines.

Each subject is asked to select all texture images that look similar to a specified input image. If all m slices in each of the constrained directions are selected, we consider the solid texture as a SUCCESS in terms of subject's evaluation. If more than 50% of the subjects give an evaluation of SUCCESS on a solid texture, the solid texture is considered as a success for the first goal. Otherwise, it is a failure.

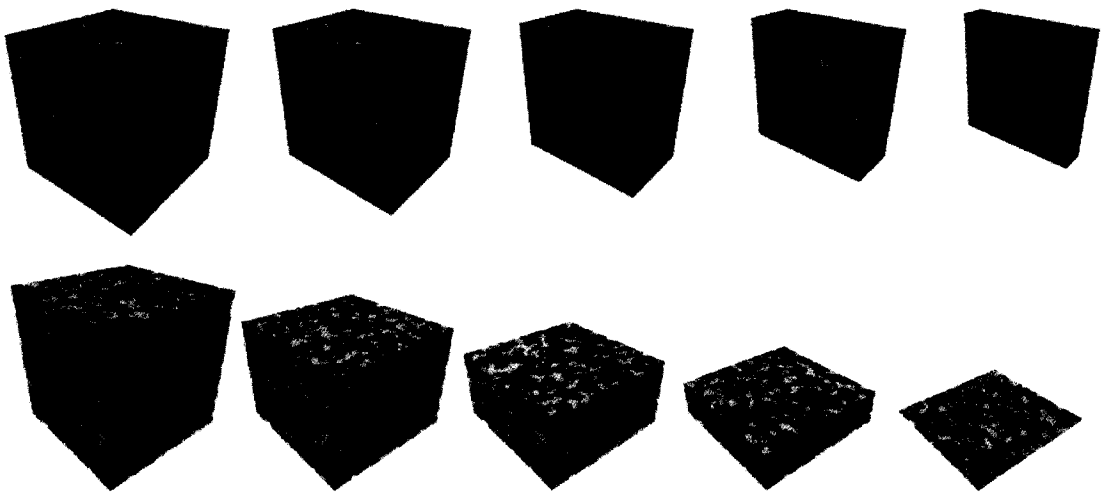


Figure 5-11: Animation sequences of cross sections of two solid textures that are generated by our algorithm. The texture in the first row is a cloud-like solid texture, and the one in the second row is a green-marble-like solid texture.

For the second goal, we randomly select v view directions (six are used in our experiments). For each view direction, we generate an animation of all cross sections of the solid texture that are cut in order along that direction. A subject is asked to watch the animation to determine if the texture changes smoothly from frame to frame. We repeat this test for 18 people. If no sudden change has been identified in all the animations of the selected views, we consider the synthesized solid texture as a SUCCESS by the subject. If more than 50% of the subjects assign a SUCCESS to a solid texture, the solid texture is considered as a success for the second goal. Otherwise, it is a failure. Figure 5-11 shows example sequences of animation frames of two solid textures that are generated by our method, one for a cloud-like solid texture and another for a green-marble-like solid texture.

To avoid the manual data collection process and to make our evaluation efficient, we have designed a GUI-based evaluation system. Figure 5-12 gives a screen shot of the system when it is used for evaluating the first goal. The input is displayed as a smaller image in the top-left corner of the window. The m slices together with other texture images randomly selected by the system from two texture databases (one for selecting the m slices and one for selecting the other textures) are displayed on the screen in random order as larger images below the input. Texture images similar to the input are selected by just clicking the button labeled "Similar" under them. When a user finishes the test, the system calculates and stores the evaluation results. The user does not know the score until after the experiment is complete.

In our GUI-based evaluation system, the user does not have any information on the value of m because otherwise he/she will use the information to guide his/her

selection, which will cause a biased evaluation. In fact, in our implementation, the value of m is assigned randomly each time by the system to make sure that the user will not guess it. Furthermore, we found that the color provides an important cue in user's selection that causes bias. For example, in Figure 5-12, the user may make selections by following the color instead of the textures of the input. To avoid this bias, we converted all color images into gray scale images when testing the first goal. For testing the second goal, we used color images because there is no such color-bias problem.

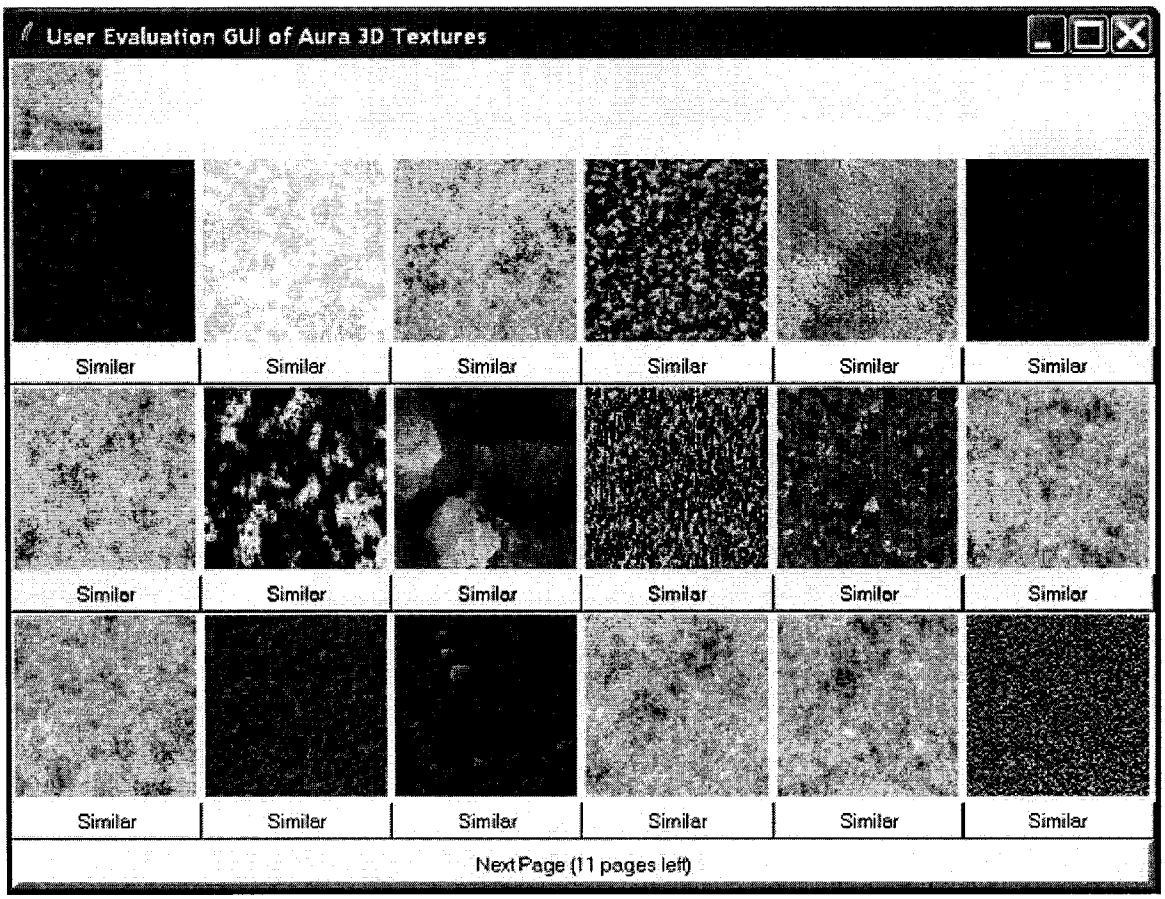
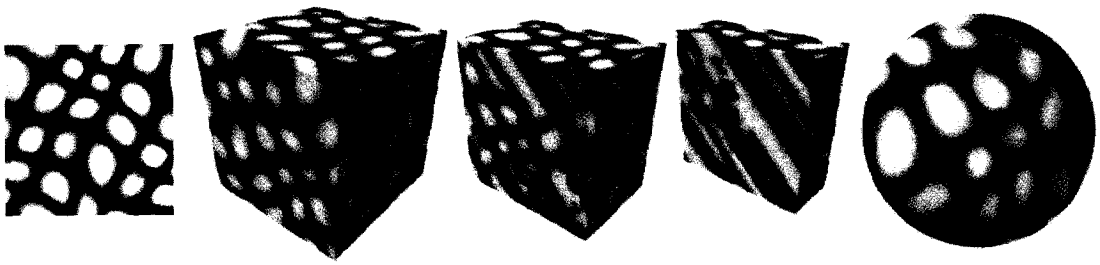


Figure 5-12: The GUI-based user evaluation system of aura 3D textures.

For the evaluation of both goals, we have used 126 input textures, which include stochastic and structural textures. The experimental results show that the average

percentage of success (for the experiments of both goals) for our aura 3D textures is 76.4%. The results of the three different groups of subjects are, respectively, 76.8%, 77.2%, and 75.1%. Thus, there is no significant difference between the results of different groups.



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

Figure 5-13: An example of failure from our algorithm that is identified during the evaluation. The input is shown in (a), the solid texture and its two cross sections are shown in (b), (c) and (d), respectively. A textured sphere by the solid texture is shown in (e). The failure of the solid texture in (b) is identified by viewing its cross sections as shown in (c) and (d).

Figure 5-13 gives an example of failure of our algorithm that is identified from the evaluation. This example also demonstrates the importance of the evaluation process as discussed below. In the figure, the input and the generated solid texture of the input are shown in (a) and (b), respectively. The two cross sections of the solid texture are shown in (c) and (d). A textured sphere by the solid texture is shown in (e). By just looking at the solid texture and the textured sphere, the results look reasonably good. However, by evaluation, we have actually identified the problems of the solid texture as shown in (c) and (d), which do not appear in the textured sphere.

5.7 Limitations and Future Work

Similar to the BGLAM-based 2D texture synthesis algorithm as described in the previous chapter, one limitation of the current implementation of the aura 3D texture is the color update scheme during the sampling as described in Section 5.3.1. It is quite possible that after a few iterations, the number of candidates of possible colors for a target point is less than 3, which may sometimes cause the color values for points in the output volume to quickly converge to local minima and thus generate visible seams in the output textures as shown in Figure 5-13. The possible solution to this problem is to use genetic algorithms [66] as discussed in Chapter 4.

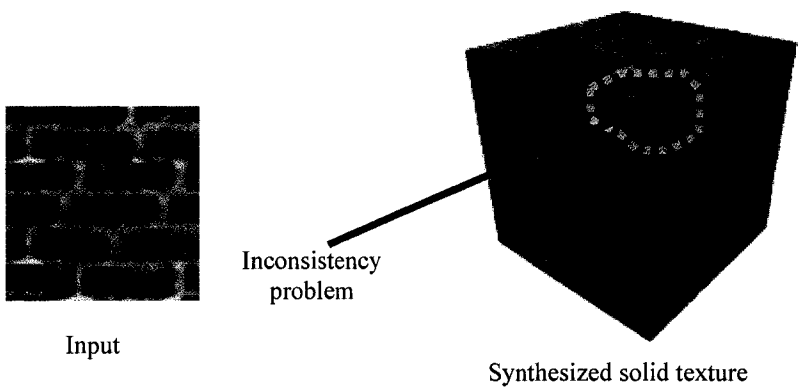


Figure 5-14: An example of inconsistency problems in oriented structural solid textures.

The second limitation is that although our method can generate faithful results for oriented structural textures, such as the brick texture as shown in Figure 5-8, we find one problem related to the issues of constraints, which we still have not found a satisfactory solution. It is relatively easy to see that the orientations of three adjacent structural textures at a junction may create an inconsistency problem. An example of such problems is shown in Figure 5-14, where the input sample is shown in the left and the

corresponding synthesized solid texture generated by our method is shown in the right. Although the orientation of the brick at the solid's corner highlighted by the dashed line is consistent with the orientation of the bricks on both sides of the solid, it is inconsistency with that of the bricks at the top. The solution to the inconsistency problems in oriented structural textures, if it exists, depends on the interpretation of the given surface textures, which is a very interesting inverse problem for future research. For example, can we detect inconsistencies? If the textures are consistent, is the solution unique?

The third limitation is related to the current implementation of the aura 3D sampling. Although our method is general enough to handle the situation in which input samples can be placed along non-orthogonal view directions to constrain the aura 3D sampling, our current implementation only handles orthogonal view directions in 3D space. We are currently considering a new implementation of our algorithm for handling non-orthogonal-view constraints. Other future research may include: the extension of our approach for GPU-based texture synthesis and the evaluation of solid textures generated by other approaches, e.g. Jagnow et al.'s algorithm [86].

5.8 Summary

In this chapter, a new method for generating solid textures from input samples is presented. Given one or more input textures, the BGLAMs of input samples are calculated first; a solid texture is then generated by sampling the BGLAMs of input samples. We evaluate the results of our method based on extensive user studies. The evaluation results show that our algorithm can generate faithful results over a wide range

of textures with an average successful rate of 76.4%. The synthesized results show that our algorithm outperforms Wei & Levoy's and are comparable to Jagnow et al.'s. However, the latter method involves extensive user interactions in designing correct 3D shapes while our method is fully automatic with no user interactions in generating solid textures.

Chapter 6

BGLAM Texture Classification

6.1 Introduction

The problem of classifying texture images into pre-learned classes is a very demanding task in computer vision and has important applications in automated inspection, medical image analysis, document processing, and bioinformatics [77, 178]. Given a set of texture classes and a set of training samples for each texture class, the goal of texture classification is to estimate the class labels of texture images in a given database. The success of texture classification depends on whether or not different textures can be correctly analyzed mathematically. However, analyzing textures with mathematical precisions is challenging and has been studied by researchers in the vision area for decades with only limited success.

In existing approaches [4, 36, 57, 72, 113, 116, 143, 167], the statistics of a set of filter responses are used to characterize an example texture for classification. However, mathematically, it requires an infinite number of filters to model a given texture with the necessary and sufficient information, each of which is as big as the given texture sample. In general, it is a difficult task to automatically design and select appropriate filters for different textures. As a result, in texture modeling, most filter-based approaches have used predefined filters, which are usually insufficient to differentiate textures. Portilla and Simoncelli [143, 144] have shown that two different texture samples may have identical means, variances, covariances, skewnesses, kurtosises or histograms computed

from filtered images [143, 144]. When those characterization statistics (e.g. means, variances, skewnesses, and kurtosises) are used in texture classification, they may cause the problem of assigning the same class label to different textures.

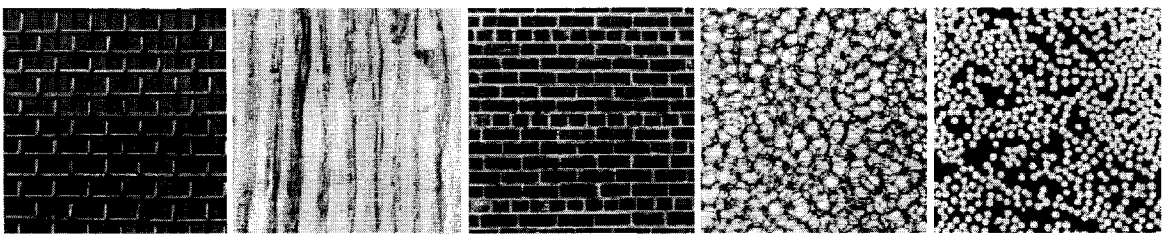


Figure 6-1: Example of the classes that are learned from training texture samples.

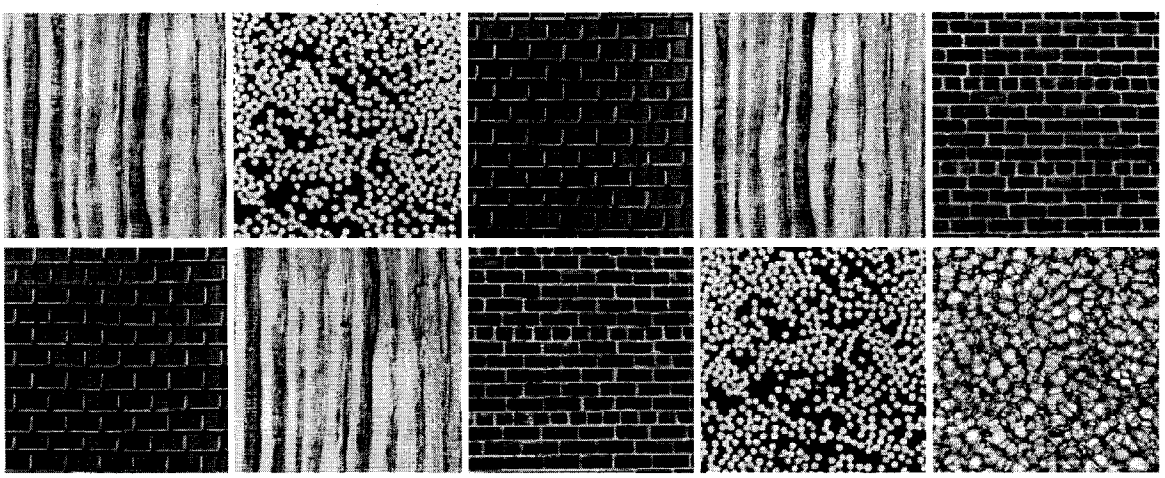


Figure 6-2: An example of texture images to be classified by the classes shown in Figure 6-1. Images are taken from the Brodatz textures [14].

Therefore, it is crucial that the characterization statistics extracted from a given texture sample have the necessary and sufficient information to represent the texture sample. The BGLAM mathematical framework developed in Chapter 3 suggests that BGLAMs can be used to characterize texture samples with this property. We have proved (in Chapter 3) that two images are identical if and only if their corresponding BGLAMs are the same, and thus an image can be uniquely represented by its BGLAMs.

Based on the BGLAM theory developed in Chapter 3, in this chapter, we present a new method of classifying texture images using BGLAMs. Given a texture sample (e.g. the first image in Figure 6-2) from an image database, our goal is to classify it into one of the pre-learned classes (e.g. the class represented by the second image with label c_2 in Figure 6-1). Our algorithm consists of two stages: a learning stage and a classification stage. In the first stage, models of texture classes are learned from the BGLAMs of training examples using the SVM (Support Vector Machine) [182], and in the second stage, a given texture image is classified into one of the pre-learned classes.

We compare our approach experimentally with existing approaches by performing texture classification using the Brodatz database, the Vistex database, and the ASI (All Sky Imager) database. The experimental results show that the proposed approach has obtained an average successful classification rate of 100%, 97%, and 97% vs 91%, 83%, and 66% using other approaches over the Brodatz textures, the Vistex textures, and the ASI textures, respectively. The test results indicate that the proposed method outperforms both Guo et al.'s algorithm [72] and the SGLAM method [152].

6.2 Related Works

The work in this chapter is closely related to texture discrimination in computer vision and image processing. One of the major research areas in texture discrimination is texture similarity measure and leaning for image classification and image retrieval. Various techniques have been developed in this area. The conventional approach is based on Gabor texture features. In Ma & Manjunath's work [113, 116] and Puzicha et al.'s work [148], the first and second order statistics of Gabor filter responses are used for measuring and characterizing textures using neural network. They have applied their

method to the classification of texture images from the Brodatz database and to the content-based image data retrieval from large satellite or aerial photographs. As a superset of Gabor filters, the windowed Fourier filter responses are used for texture similarity measure and learning in Azencott et al.'s work [4]. Based on the Gabor filter responses, Guo et al. [71, 72] use a SVM algorithm to learn similarities between texture images. Both Azencott et al.'s approach and Guo et al.'s approach have better performance in texture classification than that of Ma and Manjunath's approach.

The second approach in texture discrimination is based on wavelet-type features. In the work by Simoncelli and Portilla [143, 167], it is shown that texture images can be faithfully characterized by and reconstructed from a set of global joint statistics of complex wavelet coefficients at multiple scales and orientations. Rather than using global joint statistics, DeBonet and Viola use local joint statistics of wavelet distributions in multiresolutions to measure the similarity between images [36]. In addition to texture image classification, DeBonet and Viola's method can perform object detection in SAR (Synthetic Aperture Radar) images and faithful results have been obtained. In Arivazhagan and Ganesan's work [2], texture classification is performed using a combination of the wavelet statistical features and the wavelet cooccurrence features. As a variation of the joint statistics of filter responses, a number of researchers model textures using probability distributions of textons [206], and learn textons and texture models from training images. The classification of a given texture image is done by comparing the texton distribution of the image with the pre-learned models. The texton-based approach has been successfully used in classifying texture images under unknown viewpoint and illumination [104, 162, 183].

Another important approach for texture discrimination is the Markov/Gibbs random field (MRF/GRF) texture models [32, 63]. Researchers [17, 18, 38-40, 64, 143] have used MRF/GRF models for supervised classification and segmentation, where the number of texture classes is known as a prior. In those approaches, texture classification is performed by using either the MAP (Maximum a Posterior) estimate [64] or the MPM (Maximum Posterior Marginal) estimate [18]. Supervised texture recognition techniques have difficulties in classifying images with complex textures, for example SAR images, because there might be unknown texture types that are not in any of the pre-learned classes. To address the problem, several researchers have proposed MRF/GRF-based techniques for the unsupervised texture classification or segmentation [94, 115, 128, 155].

Gray level cooccurrence matrices (GLCMs) have also been studied in texture modeling for discrimination and characterization. Research studies [21, 34, 67, 75, 76, 112, 209] have shown that GLCMs can be used as a powerful tool for texture analysis, classification, segmentation, and synthesis. The disadvantage of a GLCM is that it only contains cooccurrence information between two pixels, and thus cannot capture the spatial relationship between three or more pixels in the image. This problem can be addressed by using symmetric gray level aura matrices (SGLAMs) [51, 137, 138], which incorporate neighborhood systems to model the relationship between the target pixel and its neighboring pixels, and thus are capable of capturing the relationship between any number of pixels. In the recent work of Qin and Yang [152], SGLAMs combined with wavelet transforms [166] have been successfully used in learning texture similarity for texture image classification and retrieval.

The work in this chapter demonstrates that BGLAMs without combining with any filters can perform better in classification than SGLAMs with wavelet transforms. For real applications, it is demonstrated that BGLAMs significantly outperforms SGLAMs and other existing approaches. In our approach, texture features are represented by BGLAMs calculated directly from images, and texture models are learned using the SVM technique [182]. Since the texture feature vectors in our approach have a very high dimension (e.g. 1536 used for experiments in our work), we adopt Joachims's fast implementation of the SVM learning algorithm [89]. The details of our approach are described in the next section.

6.3 The Approach

The classification problem addressed in this chapter is the following: Given a database of texture images of unknown categories, classify each image in the database into one of the classes that are learned from training samples. Figure 6-3 gives an overview of the BGLAM-based algorithm for texture classification. In the beginning, two texture databases are given: one is the training database that is used for learning texture classes; the other is the query database that needs to be classified and is used for evaluation. The first step of the algorithm is the characterization of texture images in both the training database and the query database using BGLAMs. The outputs of this step are two databases of BGLAMs: one for the training images and one for the images to be classified. Once this step is finished, the original texture databases are not needed any more, and the calculated BGLAMs are taken as texture features and used in the rest of the algorithm for learning and classification.

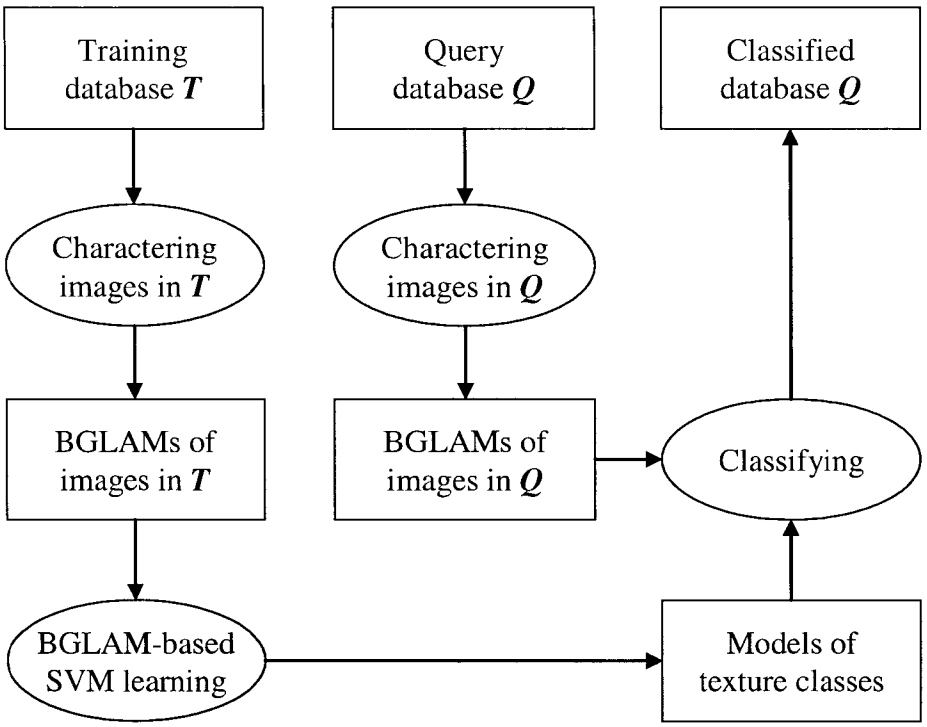


Figure 6-3: An overview of the BGLAM-based algorithm for texture classification.

In the second step, the models of texture classes are learned from the BGLAMs of training texture samples using the SVM [182]. The model of each texture class learned is a hyperplane represented by a set of support vectors (SVs) in the texture feature space. Once the models of texture classes are learned from the BGLAMs of training samples, they are used to classify all images in the query database in the final step. The three steps of the algorithm are shown as ovals in Figure 6-3.

6.3.1 Characterizing Texture Images

In our method, before learning the models of texture classes, texture images of both the training database and the query database are characterized by BGLAMs. Intuitively, the BGLAMs of an image characterize the cooccurrence probability distributions of gray levels at all possible displacement configurations and thus estimate the underlying stochastic process that is used to generate a given texture sample. In fact,

in Chapter 3, we have proved that that BGLAMs form a basis of GLAMs, which is a superset of GLCMs, and that two images are identical if and only if their corresponding BGLAMs are the same. Since a given texture sample can be accurately represented by and faithfully reconstructed from its BGLAMs, we use BGLAMs to characterize a texture sample. In our work, a compact set of BGLAMs defined over a 5×5 square window is used to characterize texture images. Once the BGLAMs are calculated, the original texture image is no longer needed and only the BGLAMs are used in the subsequent processes of learning and classification.

A fast algorithm for computing the BGLAMs of a given texture image is given in Section 4.3.1 in Chapter 4. An example of the BGLAMs calculated from a binary image can be found in Figure 4-3. In our algorithm of leaning and classification, the BGLAMs are normalized (see Definition 3-15) because texture images in the databases may have different sizes. To achieve the best performance both in the quality of results and in running time, we have also quantized [8] texture images from 256 gray levels to 8 gray levels. In the experimental section, we discuss the performances of using different number of gray levels.

6.3.2 BGLAM-Based SVM Learning

The BGLAMs of texture images in the training database are treated as texture features for learning the models of texture classes. More precisely, for a given training sample, its BGLAMs are combined together to form a vector in the texture feature space, and the vector is called a *feature vector* of the training sample. For each texture sample X in training database $\boldsymbol{T}$, let $\{\boldsymbol{B}_i \mid 0 \le i \le m-1\}$ be its m BGLAMs and $\boldsymbol{x}$ be its feature vector, then $\boldsymbol{x}$ is given as:

$$\boldsymbol{x} = (\boldsymbol{B}_0, \boldsymbol{B}_1, ..., \boldsymbol{B}_{m-1}). \qquad (6.1)$$

If each BGLAM in $\{\boldsymbol{B}_i \mid 0 \le i \le m-1\}$ has b entries, then the dimension of $\boldsymbol{x}$ is $n = m * b$. Figure 6-4 gives an example of a feature vector $\boldsymbol{x}$ computed from four normalized BGLAMs.

$$\boldsymbol{B}_0 = \begin{bmatrix} \frac{6}{16} & \frac{3}{16} \\ \frac{3}{16} & \frac{4}{16} \end{bmatrix} \quad \boldsymbol{B}_1 = \begin{bmatrix} \frac{6}{20} & \frac{6}{20} \\ \frac{4}{20} & \frac{4}{20} \end{bmatrix} \quad \boldsymbol{B}_2 = \begin{bmatrix} \frac{3}{16} & \frac{8}{16} \\ \frac{3}{16} & \frac{2}{16} \end{bmatrix} \quad \boldsymbol{B}_3 = \begin{bmatrix} \frac{6}{20} & \frac{6}{20} \\ \frac{3}{20} & \frac{5}{20} \end{bmatrix}$$

$$\boldsymbol{x} = \left(\frac{6}{16}, \frac{3}{16}, \frac{3}{16}, \frac{4}{16}, \frac{6}{20}, \frac{6}{20}, \frac{4}{20}, \frac{4}{20}, \frac{3}{16}, \frac{8}{16}, \frac{3}{16}, \frac{2}{16}, \frac{6}{20}, \frac{6}{20}, \frac{3}{20}, \frac{5}{20} \right)$$

Figure 6-4: An example of a feature vector $\boldsymbol{x}$ computed from four BGLAMs $\boldsymbol{B}_0$, $\boldsymbol{B}_1$, $\boldsymbol{B}_2$, and $\boldsymbol{B}_3$. The dimension of vector $\boldsymbol{x}$ is $4 \times 4 = 16$.

The learning is performed with the feature vectors of training samples. Suppose we have a set of training vectors of two different classes, $(\boldsymbol{x}_i, y_i)$, $0 \le i \le L-1$, where $\boldsymbol{x}_i \in \Re^n$ is a feature vector calculated from a texture image as described before, and $y_i \in \{-1, +1\}$ is the class label for $\boldsymbol{x}_i$, where label +1 means a positive example (i.e. a pattern of a target class) and -1 means a negative example (i.e. a pattern not in the target class). We assume that during the learning process, all training texture samples of a desired category (e.g. all images that contain a specific texture type) are grouped into one class and labeled as positive examples, while samples not in the desired category (e.g. images that do not contain the target texture type) are considered as negative examples. We also assume that an image database consists of disjoint classes. Namely, each texture image in the database has and only has one class label.

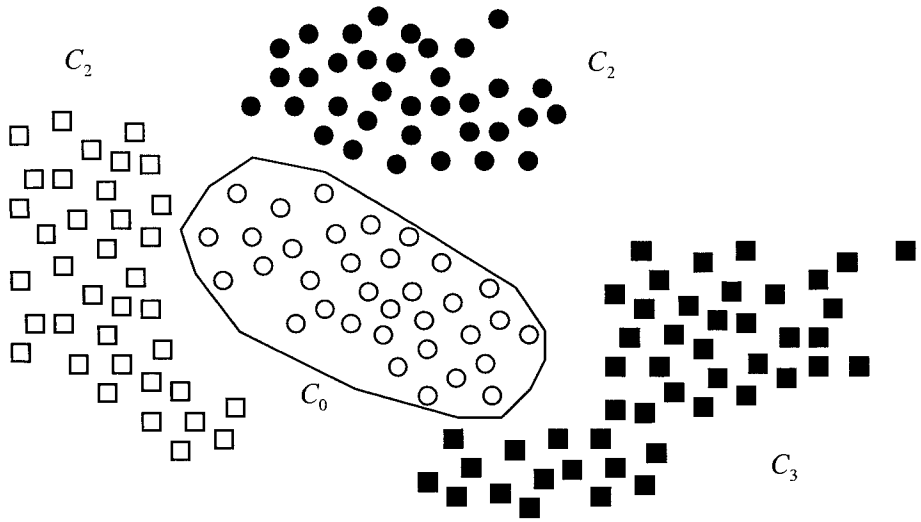


Figure 6-5: An example of the boundary of a sample class C_0 in $\Re^2$. The boundary of C_0, which separates C_0 from the other three classes C_1, C_2, and C_3, is shown as a closed curve around C_0.

The goal of learning is to find the optimal boundary that separates the positive and negative examples. The boundary is a hyperplane in feature space $\Re^n$. Figure 6-5 gives an example of the boundary of a sample class C_0, which separates the class from the other three classes in space $\Re^2$.

In fact, finding the boundary leads to the following quadratic optimization problem [182]:

$$\alpha^* = \arg\min_{\alpha}\left[-\sum_{i=0}^{L-1}\alpha_i + \frac{1}{2}\sum_{i,j=0}^{L-1} y_i y_j \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j)\right], \quad (6.2)$$
$$\text{subject to}: \sum_{i=0}^{L-1}\alpha_i y_i = 0, \ \forall i: 0 \le \alpha_i \le C$$

where the vector $\alpha = (\alpha_i)_{0 \le i \le L-1}$ is the parameter to be estimated, $K(\boldsymbol{x}, \boldsymbol{y})$ is the kernel function, and C is a constant. There are a number of choices for the kernel function. For the discussion on how to choose an appropriate kernel function for different applications,

the reader is referred to Vapnik's book on statistical learning [182]. In our work, we use the RBF (Radial Basis Function) because it is best suited for classification as discussed in Joachims's work [88, 89]. The RBF kernel is given as $K(\boldsymbol{x},\boldsymbol{y}) = \exp(-\frac{1}{2\sigma^2}\| \boldsymbol{x}-\boldsymbol{y} \|^2)$, where σ is a constant given by the user. In all the experimental results shown in this chapter, we set $\alpha = 0.02$.

In Eq. 6.2, if all the training samples are used for optimization, then it is very inefficient for large training sets. It has been shown [182] that a set of SVs (support vectors), which is a small subset of the training samples, can be used in Eq. 6.2 for optimization. Once the optimal parameter $\alpha^* = (\alpha_i^*)_{0\le i\le m-1}$ is estimated in Eq. 6.2 using m SVs, the optimal boundary B is given by:

$$\sum_{i=0}^{m-1} \alpha_i^* y_i K(\boldsymbol{x}_i^*, \boldsymbol{x}) + b^* = 0, \qquad (6.3)$$

where $\{\boldsymbol{x}_i^* \mid 0 \le i \le m-1\}$ is a set of SVs, b^* is a constant. The parameters α_i^*, $\boldsymbol{x}_i^*$, and b^* are all learned from the SVM. In our work, we have used a fast implementation of the SVM, called SVM^{light}, developed by Joachims [88, 89] to learn parameters α_i^*, $\boldsymbol{x}_i^*$, and b^*, $i = 0,1,...,m-1$.

The model of a specific texture class is represented by its boundary B, whose equation is given as in Eq. 6.3. For an image database of multiple disjoint classes, the above SVM learning algorithm is applied to each class to learn its optimal boundary. Once the boundaries of all classes are learned from the training samples, they will be used to classify each texture image in the query database.

6.3.3 Classifying Texture Images

For a given texture image Z in query database $\boldsymbol{Q}$, its feature vector z calculated by Eq. 6.1 is used to classify it into one of the classes that are learned from the training database $\boldsymbol{T}$ using the SVM as discussed in the previous section. Suppose there are k pre-learned classes $\{C_j \mid 0 \le j \le k-1\}$ with optimal boundaries $\{B_j \mid 0 \le j \le k-1\}$ defined by Eq. 6.3, then the signed distance of feature vector z to the boundary B_j is given by:

$$d_j = d(z, B_j) = [\sum_{i=0}^{m-1} \alpha^*_{ji} y_{ji} K(\boldsymbol{x}^*_{ji}, z) + b^*_j] \Big/ \| \sum_{i=0}^{m-1} \alpha^*_{ji} y_{ji} \boldsymbol{x}^*_{ji} \|, j = 0,1,...,k-1. \quad (6.4)$$

Since a one-against-all strategy is used in learning, for class C_j with boundary B_j, if the signed distance $d_j = d(z, B_j)$ calculated from Eq.6.4 is positive, then texture image $Z \in C_j$; otherwise, it belongs to one of the other classes. For multiple classes, the final class label l of texture image Z is determined by the boundary of the class to which z has the largest distance, i.e.

$$l = \arg\max_j \{d_j \mid 0 \le j \le k-1\}. \quad (6.5)$$

If a class contains subclasses, the SVM algorithm can be further applied to images in that class to learn subboundaries.

6.3.4 Algorithm

The pseudocode of the BGLAM-based algorithm for texture classification by learning is given in Figure 6-6. The inputs are two databases of texture images: one is the training database, where each image has an assigned class label; the other is the query database, where texture images are to be classified. The output of the algorithm is a classified database in which each image has a class label associated with it.

BGLAM-Based Texture Classification

Input:
$\boldsymbol{T}$ ← the training texture database.
$\boldsymbol{Q}$ ← the query texture database to be classified.
Output:
$\boldsymbol{Q}$ ← the classified texture database with each image having an assigned class label.

Begin
1 Calculate BGLAMs of each image in both $\boldsymbol{T}$ and $\boldsymbol{Q}$, and store them.
2 Learn models of texture classes with BGLAMs of images in $\boldsymbol{T}$ using SVM.
  2.1 For each class C_j, learn its optimal boundary B_j using the SVM, $0 \le j \le k\text{-}1$.
3 Classify texture images in $\boldsymbol{Q}$, i.e. for each image Z in $\boldsymbol{Q}$
  3.1 z ← the feature vector of Z calculated by Eq. 6.1.
  3.2 For each class C_j, $0 \le j \le k\text{-}1$
    3.2.1 Calculate the signed distance d_j using Eq. 6.4.
  3.3 $l \leftarrow maxarg\{d_0, d_1, \ldots, d_{k-1}\}$
  3.4 Assign the class label l to image Z.
  End of 3
4 Return $\boldsymbol{Q}$
End of begin

Figure 6-6: The pseudo code of the BGLAM-based algorithm for texture classification.

There are three major steps in the algorithm. The first step is to calculate the BGLAMs of texture images in both the training database $\boldsymbol{T}$ and the query database $\boldsymbol{Q}$. The BGLAMs of images are used as features to characterize texture images. Once the BGLAMs are calculated, they are used in the subsequent learning and classification. In the second step, the optimal boundary of each class is learned with the BGLAMs of training samples from that class using the SVM based on the one-against-all strategy as described before. The classification of each texture image in the query database $\boldsymbol{Q}$ is done in the third step.

6.4 Experiments

In this section, we present the experimental results to evaluate our BGLAM-based approach for texture classification by learning. We compare our approach with existing approaches. The Brodatz database, the Vistex database, and the ASI (All Sky Imager) database are used for testing. Since Gabor filters are used in our experiments for extracting texture features for existing approaches, they are briefly described first. The experimental results for each texture database are then followed.

6.4.1 Gabor Filters

Gabor filters [87, 116] are derived from a two dimensional Gabor function $g(x, y)$ and its Fourier transform $G(u, v)$, whose definitions are given below:

$$g(x,y) = \frac{1}{2\pi\sigma_x\sigma_y}\exp\left[-\left(\frac{x^2}{2\sigma_x^2}+\frac{xy^2}{2\sigma_y^2}\right)+2\pi jWx\right], \tag{6.6}$$

$$G(u,v) = \exp\left[-\left(\frac{(u-W)^2}{2\sigma_u^2}+\frac{v^2}{2\sigma_v^2}\right)\right], \tag{6.7}$$

$$\sigma_u = \frac{1}{2\pi\sigma_x} \text{ and } \sigma_v = \frac{1}{2\pi\sigma_y}, \tag{6.8}$$

where W is constant. The standard deviations σ_x and σ_y of the Gaussian function in Eq. 6.6 are usually called the bandwidth parameters because they determine the bandwidth of the Gabor function $g(x, y)$.

Gabor filters are obtained by appropriate scales and rotations of $g(x, y)$ using the generating function:

$$
\begin{aligned}
g_{mn}(x,y) &= a^{-m}G(x_1,y_1) \\
x_1 &= a^{-m}\left(x\cos\frac{n\pi}{T} + y\sin\frac{n\pi}{T}\right) \quad , \\
y_1 &= a^{-m}\left(-x\sin\frac{n\pi}{T} + y\cos\frac{n\pi}{T}\right)
\end{aligned} \tag{6.9}
$$

where $a>1$, $m=0,1,...,S-1$, $n=0,1,...,T-1$, S is the total number of scales, and T is the total number of orientations.

Let L and U be the lower and upper center frequencies of interest. Given the total number of scales S and the total number of orientations T, by using a method proposed by Manjunath and Ma [116], the optimal values for the filter parameters σ_u and σ_v are computed as follows:

$$
\begin{aligned}
a &= (L/U)^{1/(S-1)} \text{ and } \sigma_u = \frac{(a-1)U}{(a+1)\sqrt{2\ln 2}} \\
\sigma_v &= \tan\left(\frac{\pi}{2T}\right)\left[U - 2\ln\left(\frac{\sigma_u^2}{U}\right)\right]\left[2\ln 2 - \frac{(2\ln 2)^2\sigma_u^2}{U^2}\right]^{-1/2} ,
\end{aligned} \tag{6.10}
$$

Once the values of σ_u and σ_v are obtained using Eq. 6.10, the bandwidth parameters σ_x and σ_y can be computed using Eq. 6.8. For good performance, Manjunath & Ma [116] and Clausi & Jernigan [22] have suggested that $L=0.05$, $U=0.5$, $S=4$, $T=6$, and $W=U$. In the experiments described in this chapter, we use $L=0.05$, $U=0.5$ and $W=U$, and show results for different values of S and T. For the general guidance and detailed discussions on how to choose optimal values for S and T, the interested reader is referred to Clausi and Jernigan's work [22].

Given an image I, the Gabor filter response of a given m and n at pixel location (x,y), denoted by $f_{mn}(x,y)$, is given by:

$$f_{mn}(x,y) = \int I(\xi,\eta) g_{mn}(x-\xi, y-\eta) d\xi d\eta \,, \tag{6.11}$$

where $m = 0,1,...,S-1$, $n = 0,1,...,T-1$, S is the total number of scales, T is the total number of orientations.

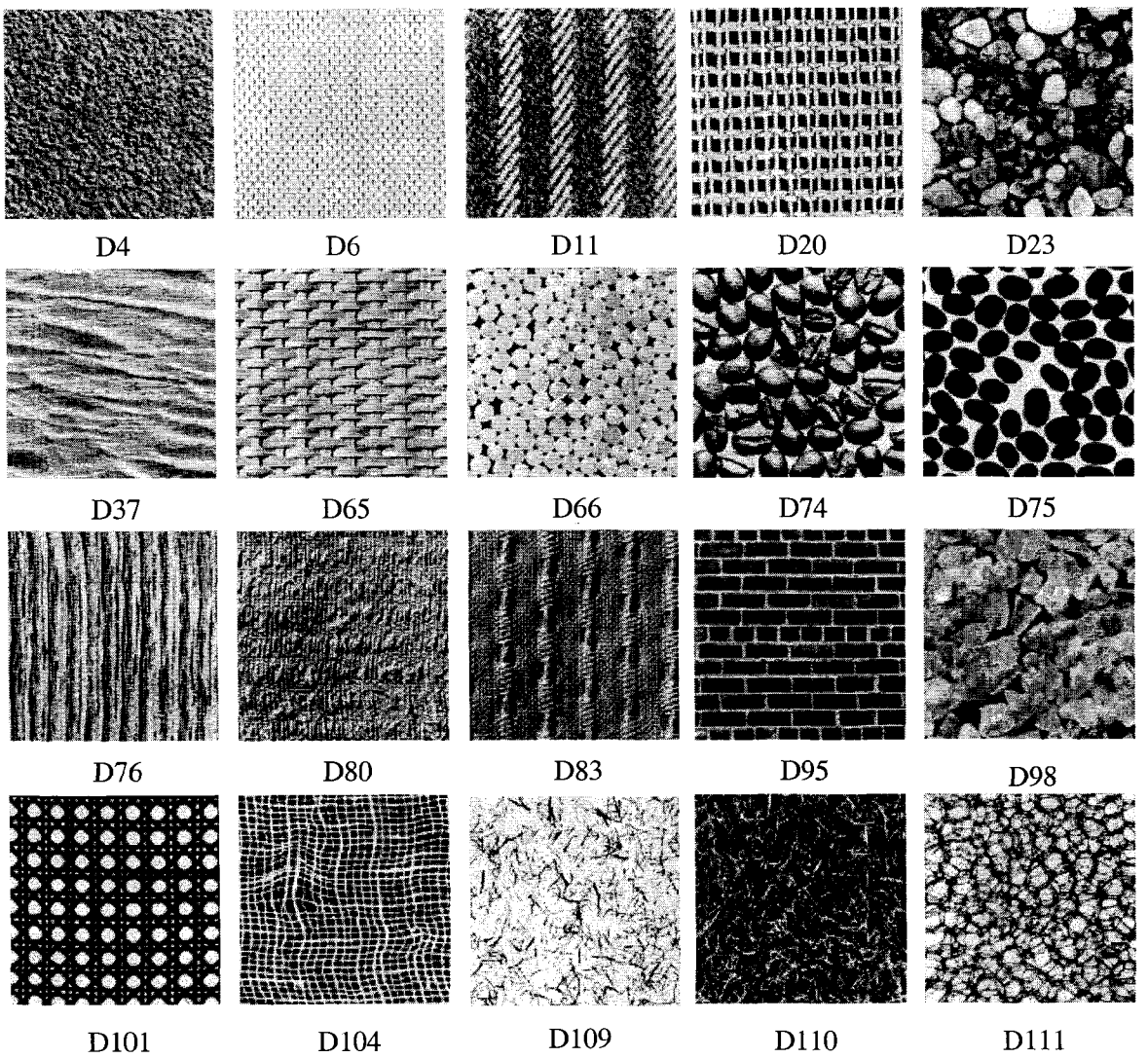


Figure 6-7: The 20 images from the Brodatz database used for experiments. The names of texture images (e.g. D4) are the original names from the Brodatz Album [14].

6.4.2 Brodatz Textures

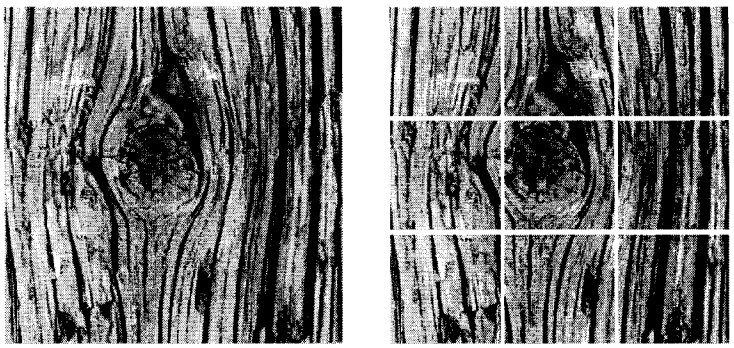
The Brodatz texture database contains 112 texture images and each texture is a 256x256 image. Among the 112 images, we select 20 images that are homogenous textures. Figure 6-7 shows the 20 Brodatz texture images. Each of the 20 texture images is considered as a texture class. Table 6-1 gives the class labels assigned to the 20 images shown in Figure 6-7. Those labels are used as references to the images in the rest of this subsection. For testing, each texture image is divided into 16 disjoint subsamples of size 64x64. Among the 16 subimages, eight images are randomly selected and used for training; the remaining 8 subimages are for classification. Therefore, we have a database of 160 images for learning, and a database of 160 images for testing in classification. At the end of this subsection, we also present the classification results by varying the number of learning samples (from 1 to 8) for each class.

Table 6-1: Class labels of the 20 Brodatz images that are shown in Figure 6-7.

Class	Name in the Brodatz Database	Class	Name in the Brodatz Database
1	D4	11	D76
2	D6	12	D80
3	D11	13	D83
4	D20	14	D95
5	D23	15	D98
6	D37	16	D101
7	D65	17	D104
8	D66	18	D109
9	D74	19	D110
10	D75	20	D111

The reason for selecting homogenous texture images from the database is that in the experiments all subsamples of an image are treated as a single class. If an image contains inhomogeneous textures, then its disjoint subsamples may contain different

textures, thus should belong to different classes. An example is given in Figure 6-8 to illustrate the problem.



D7 Subsamples

Figure 6-8: An example of a Brodatz image (D72) with inhomogeneous textures and its disjoint subsamples. The middle subsample contains textures different from those in other subimages. However, it has the same class label as those used for other subsamples. For example, if the middle subsample were used for training, then a wrong model would be learned.

For each texture sample X in the test database, it has a true label, denoted by $l_0(X)$; it also has a label assigned by a classification algorithm, denoted by $l_1(X)$. If label $l_1(X)$ matches label $l_0(X)$, which indicates that image X is correctly classified by the algorithm, then we record a *success*; otherwise we record a *failure*. For each class C_i, the success rate R_i of a classification algorithm is given by:

$$R_i = \frac{The\ total\ number\ of\ sucesses\ over\ test\ images\ in\ C_i}{The\ total\ number\ of\ test\ images\ in\ C_i},\ i = 0,1,...,N-1, \quad (6.12)$$

where N is the total number of classes. The average success rate R over the entire test database is given by:

$$R = \frac{The\ total\ number\ of\ sucesses\ over\ images\ in\ the\ test\ database}{The\ total\ number\ of\ images\ in\ the\ test\ database}. \quad (6.13)$$

For example, if there are 700 images correctly classified over a test database of 1000 images, then the average success rate of classification is $700/1000 = 70\%$.

We compare the BGLAM algorithm with Guo et al.'s algorithm [72] and the SGLAM method [152]. We have implemented both algorithms, which require filters for extracting texture features. For a fair comparison, the same filters (e.g. Gabor filters [87, 116] as that used in Guo et al.'s original paper [72]) are used to extract texture features for both Guo et al.'s method and the SGLAM method. Note that our new method does not need filters.

For Guo et al.'s approach [72], Gabor filters of 4 scales and 6 orientations, i.e. 24 filters in total, are used. To extract texture features of an image, Guo et al.'s algorithm first applies the Gabor filters to the image to obtain 24 filtered images. Then, it calculates the mean and standard deviation of pixels' gray levels of each filtered image. Finally, it combines the 24 means and standard deviations to form a feature vector of the image. The dimension of each feature vector for Guo et al.'s algorithm is $2 \times 24 = 48$. In the experiments for Guo et al's algorithm, an image is not quantized to reduce the total number gray level in the image because otherwise the performance of the algorithm will be decreased [22].

For the SGLAM approach [152], Gabor filters of 4 scales and 6 orientations, i.e. 24 filters in total, are used to extract texture features. Each image is quantized evenly from 256 gray levels into 8 gray-level bins. A SGLAM is calculated for each filtered image using a 5×5 window. Since there are 8 gray levels, the size of each SGLAM is 8×8. We calculate SGLAMs for all the 24 filtered images and combine the 24 SGLAMs

together (see Eq. 6.1) to form the feature vector. Thus, the dimension of each feature vector for the SGLAM algorithm is $24\times8\times8=1536$.

For our approach, we calculate BGLAMs directly from images using a 5×5 neighborhood. The calculated BGLAMs are then combined to form an image's feature vector. Since there are 24 BGLAMs calculated for each image, and each BGLAM has 8×8 entities, the dimension of each feature vector the BGLAM algorithm is $24\times8\times8=1536$.

Table 6-2: The comparison results of our algorithm with Guo et al.'s algorithm and the SGLAM method on the Brodatz textures.

	Average Success Rate of Classification (R)
Guo et al.	90.0%
SGLAM	91.9%
BGLAM	100%

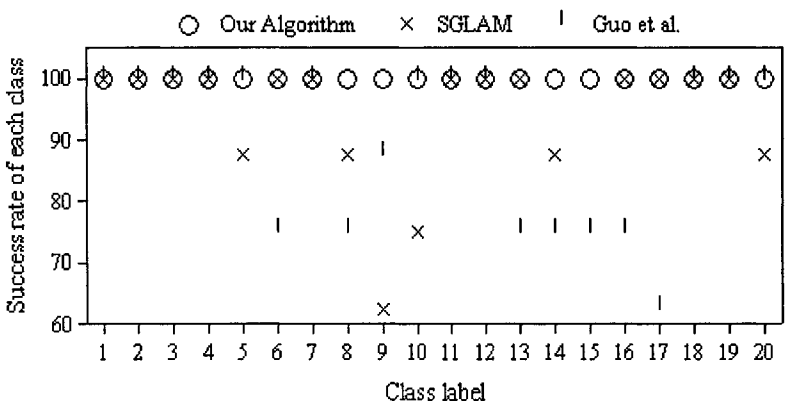


Figure 6-9: The comparison results of our algorithm with SGLAM and Guo et al.'s algorithm. The class labels are shown along the horizontal axis, and the success rates (see Eq. 6.12) of the three algorithms for the 32 classes are shown along the vertical axis.

Table 6-2 gives the comparison results of the three algorithms in terms of the average success rate over the 20 Brodatz textures. Table 6-2 shows that our algorithm

outperforms both SGLAM and Guo et al.'s algorithm. Figure 6-9 gives the comparison results of the three algorithms in terms of the success rate for each of the 20 classes. In Figure 6-9, for each class, the success rates of our algorithm, SGLAM, and Guo et al.'s algorithm are marked by a circle (o), a cross (×), and a short vertical line (|), respectively. Table 6-2 and Figure 6-9 have shown that our algorithm outperforms both SGLAM and Guo et al.'s algorithm. In fact, our algorithm has correctly classified all of the 160 images in the test database, while SGLAM and Guo et al.'s algorithm have misclassified, respectively, 8.1% and 10% of the 160 images.

One advantage of our approach is that our algorithm has intuitive user-specified parameters, while both SGLAM and Guo et al.'s algorithm have nonintuitive ones. Our algorithm has two parameters that can be tuned by the users: the number of BGLAMs and the size of BGLAMs. The number of BGLAMs is determined by the size of the neighborhood system. For an $m \times m$ neighborhood system, the total number of BGLAMs is $m^2 - 1$ (see Section 4.3.1). In general, the larger the neighborhood size, the better the results in classification. The computation time is, however, longer for a larger neighborhood system. In the experiments, we found that when $m = 5$, 7, or 9, our algorithm has good performance in both classification and running time. Figure 6-10 gives the classification performance of our algorithm in terms of the average success rate on the 20 texture images from the Brodatz database with different values of m. Figure 6-11 shows the corresponding running times. The results of Figure 6-10 and Figure 6-11 are summarized in Table 6-3 for the ease of comparison. To achieve the best performance in classification's accuracy, Table 6-3 suggests that the value of m must be at least 5.

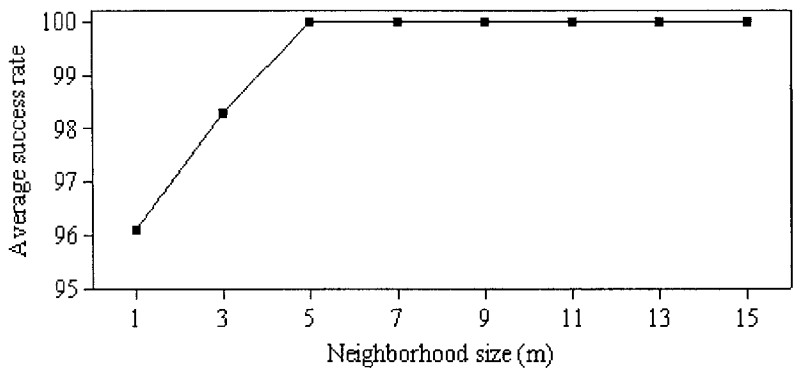


Figure 6-10: The classification results of our algorithm on the Brodatz database with $m = 1, 3, 5, 7, 9, 11, 13, 15$, respectively. If $m = 1$, then the BGLAMs of an image are calculated with the four-nearest neighbors; otherwise, they are calculated with an $m \times m$ square window. The results are obtained with BGLAMs of size 8×8.

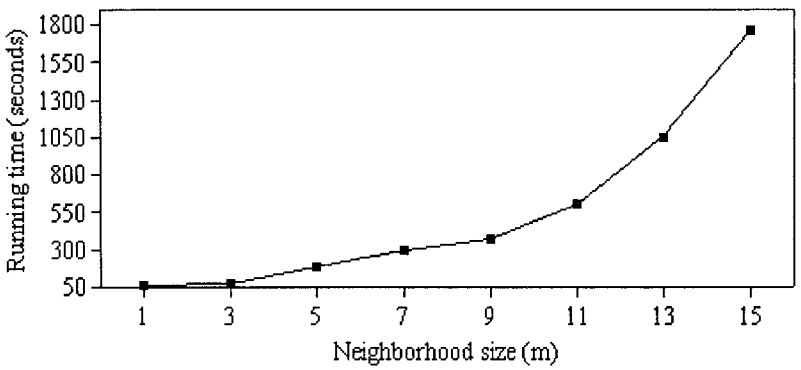


Figure 6-11: The running time of our algorithm on the Brodatz textures with neighborhood size $m = 1, 3, 5, 7, 9, 11, 13, 15$, respectively. The results of this figure are obtained with BGLAMs of size 8×8.

Table 6-3: The average success rates of our algorithm on the Brodatz texture with neighborhood systems of different sizes and the corresponding running times. The results are obtained with BGLAMs of size 8×8.

Neighborhood size (m)	Average success rate (R)	Running time (seconds)
1	96.1	61
3	98.3	75
5	100	185
7	100	302
9	100	370
11	100	611
13	100	1057
15	100	1761

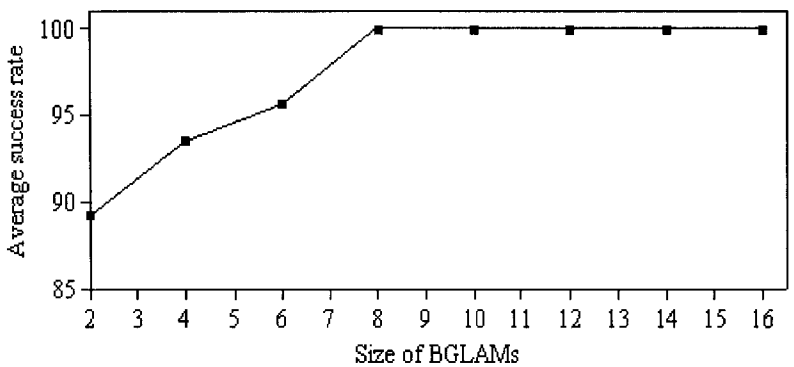


Figure 6-12: The classification results of our algorithm in terms of average success rate on the Brodatz images with BGLAMs of different sizes. A number 8 along the x-axis means the size of BGLAMs is 8×8. The results are obtained with a 5×5 neighborhood.

The second parameter of our algorithm is the size of BGLAMs, which is determined by the total number of gray levels in images. Similar to the first parameter discussed before, the larger the parameter, the better the results in classification (see Figure 6-12), but the more expensive the computation cost (see Figure 6-13). In the experiments, we found that when the size of BGLAMs is 8×8 to 16×16, our algorithm has good performance in both classification and running time. Table 6-4 shows the

average success rates of our algorithm on the Brodatz textures with BGLAMs of different sizes and the corresponding running times. Table 6-4 indicates that when the size of BGLAMs is greater than 8, our algorithm reaches the optimal performance in classification's accuracy with an average success rate of 100%.

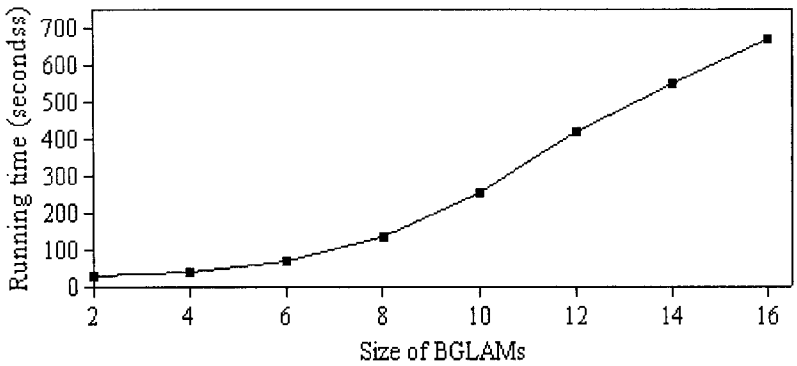


Figure 6-13: The running time of our algorithm on the Brodatz textures with BGLAMs of different sizes. The results are obtained with a 5×5 neighborhood.

Table 6-4: The average success rates of our algorithm on the Brodatz texture with BGLAMs of different sizes and the corresponding running times.

Size of BGLAMs	Average success rate (R)	Running time (seconds)
2	89.3	31
4	93.5	40
6	95.7	69
8	100	134
10	100	255
12	100	420
14	100	551
16	100	669

Both SGLAM and Guo et al.'s algorithm use Gabor filters to calculate texture features. Before applying Gabor filters, two parameters must be specified by the users: the number of scales and the number of orientations (for the details, see Section 6.4.1). Figure 6-14 and Figure 6-15 show the classification results of both SGLAM and Guo et

al.'s algorithm in terms of the average success rate on the Brodatz textures with different number of scales and different number of orientations, respectively. From the results, we cannot conclude that the larger the parameter values, the better the results.

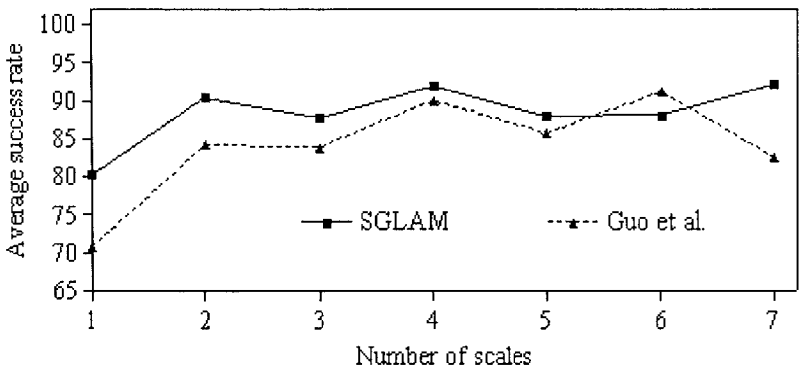


Figure 6-14: The classification results of SGLAM and Guo et al.'s algorithm in terms of average success rate on the Brodatz textures with different number of scales. For all scales, the number of orientations used is six.

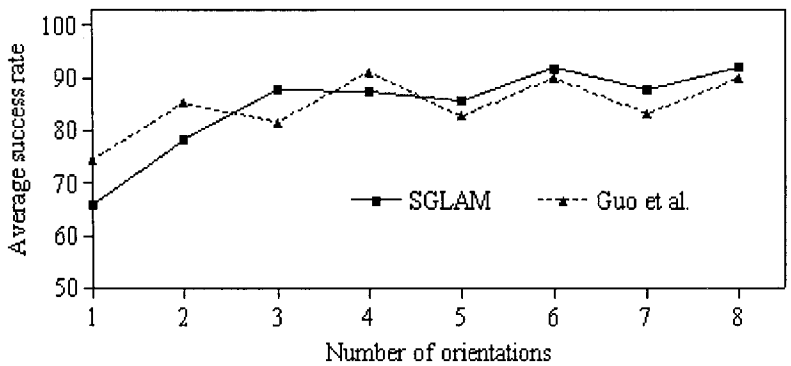


Figure 6-15: The classification results of SGLAMs and Guo et al.'s algorithm on the Brodatz textures with different number of orientations. The results are obtained with four image scales.

In the above experiments, the number of training samples for each class is equal to the number of testing samples. When the number of training samples is decreased, the classification performance of our algorithm decreases only slightly; while the performance of Guo et al.'s algorithm and of SGLAM decreases significantly. Figure

6-16 shows the comparison results of our algorithm with Guo et al.'s algorithm and SGLAM in terms of the average success rate when the number of training subsamples for each class decreases from 8 to 1 while the number of testing subsamples increases from 8 to 15 (Note that the total number of samples of each class in training and testing is 16). When only one training subsample is used, our method has an average success rate of above 90%; while Guo et al.'s algorithm and the SGLAM algorithm have low average success rates of about 13% and 44%, respectively.

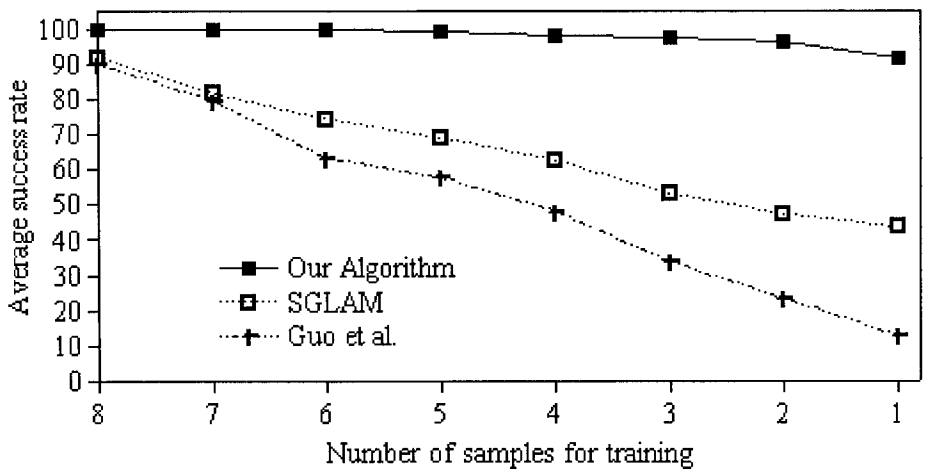


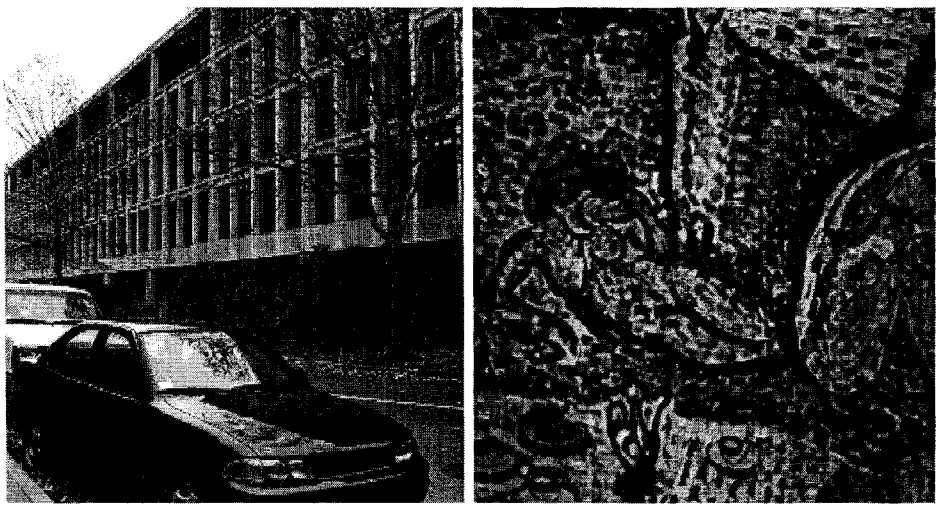
Figure 6-16: The comparison results of our algorithm with Guo et al.'s algorithm and SGLAM with the number of training samples for each class decreased from 8 to 1 while the number of testing subsamples is increased from 8 to 15.

With respect to running time, our algorithm is faster than SGLAM, but slower than Guo et al.'s algorithm. In the experiments, our algorithm uses 24 BGLAMs and each BGLAM's size is 8×8; SGLAM algorithm uses Gabor filters of with 4 scales and 6 orientations, and each SGLAM's size is 8×8; Guo et al.'s algorithm use Gabor filters of with 6 scales and 4 orientations. For the 20 Brodatz images as shown in Figure 6-7 (160 learning images and 160 testing images of size 64×64), the running times for our new

algorithm, SGLAM, and Guo et al.'s algorithm, are 134, 315, and 39 seconds, respectively.

Table 6-5: Classes in the Vistex texture database. The name of an image, which indicates the type of textures the image contains, is the original name from the MIT Vistex [185].

Class	Name in the Vistex Database	Class	Name in the Vistex Database
1	Bark	11	Misc
2	Brick	12	Paintings
3	Buildings	13	Sand
4	Clouds	14	Stone
5	Fabric	15	Terrain
6	Flowers	16	Tile
7	Food	17	Water
8	Grass	18	WheresWaldo
9	Leaves	19	Wood
10	Metal		



buildings0005 paintings0001

Figure 6-17: Examples of Vistex images that contain inhomogeneous textures.

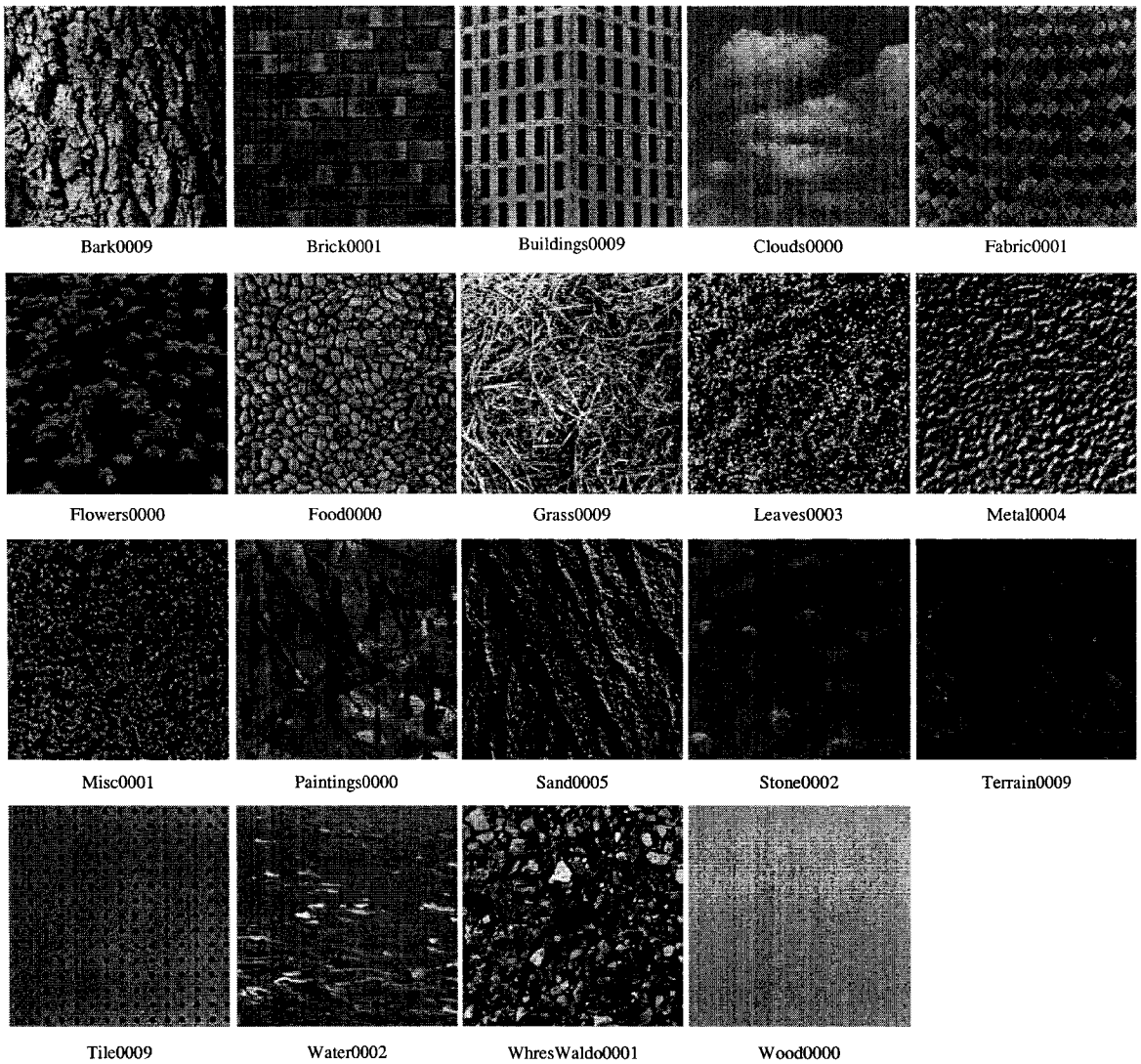


Figure 6-18: The 19 Vistex images used for the experiments. The names of images (e.g. Bark0009) are the original names from the MIT Vistex [185].

6.4.2 Vistex Textures

The Vistex texture database contains 167 texture images, and each texture is a 256x256 image. They are originally grouped into 19 meaningful classes as shown in Table 6-5. Vistex images that contain inhomogeneous textures (see Figure 6-17) are not used in the experiments for the same reason as discussed in the previous subsection. Therefore, only Vistex images with homogeneous textures are used for experiments.

Each of the 19 texture classes (Table 6-5) has different number of samples; for example, there are 11 Building images but only two Cloud images. To avoid bias in the number of samples for each class in learning, we use one Vistex image for each class. Figure 6-18 shows all the 19 Vistex images with their original names from the MIT Vistex [185].

Following a similar experimental setup to the one used for the Brodatz textures as described in the previous subsection, we obtain 16 disjoint subimages for each Vistex image. Eight subimages are randomly selected and used for training; the remaining 8 subimages are used for classification. Thus, we have a learning database of 152 images, and a testing database of 152 images.

Table 6-6: The comparison results of our algorithm with Guo et al.'s and SGLAM on Vistex textures.

	Average Success Rate of Classification (R)
Guo et al.	82.2%
SGLAM	84.2%
BGLAM	97.4%

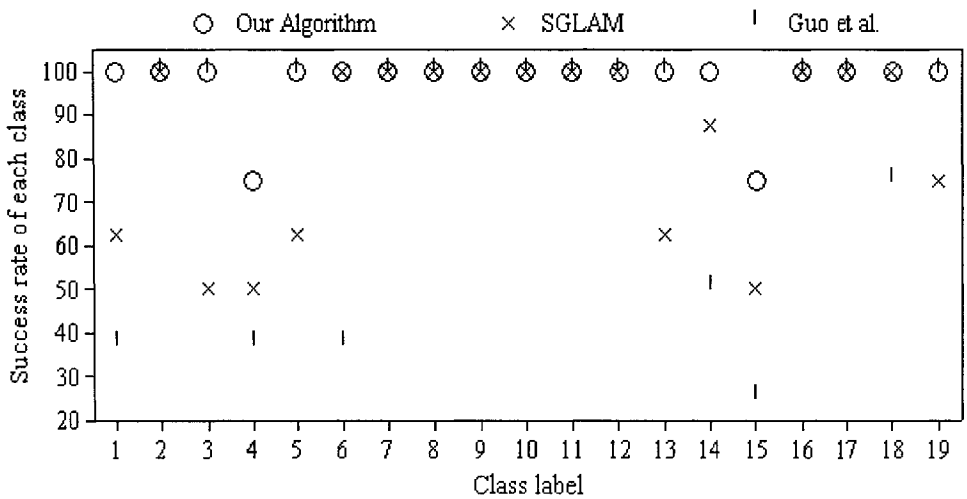


Figure 6-19: The comparison results of our algorithm with SGLAM and Guo et al.'s algorithm for each class in the Vistex database.

We compare our algorithm with Guo et al.'s algorithm [72] and SGLAM [152]. Table 6-6 gives the comparison results of the three algorithms in terms of the average success rate over the 19 Vistex textures. Figure 6-19 gives the comparison results of the three algorithms for each class. Table 6-6 and Figure 6-19 show that our algorithm outperforms both SGLAM and Guo et al.'s algorithm with an average successful rate of 97% vs 84% and 82% of SGLAM and Guo et al.'s, respectively.

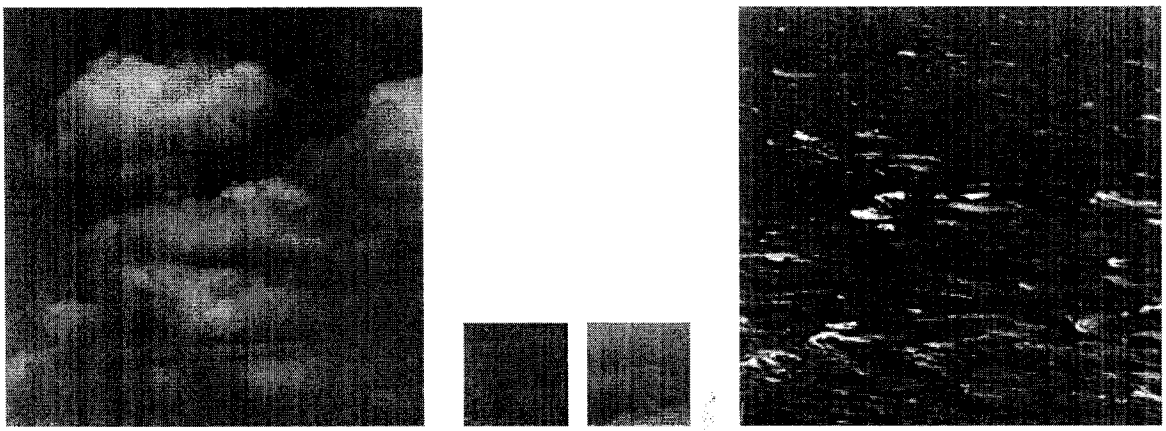


Figure 6-20: Examples of misclassified subsamples of Clouds0000.

Some discussions on the results in Figure 6-19 are given as follows. Except for some subsamples of Clouds0000 (class 4) and Terrains0000 (class 15), our method have correctly classified all subsamples in other classes. For failed cases, our method has misclassified two subsamples of Clouds0000 into Water0002, and two subsamples of Terrain0000 into WheresWaldo0001. The misclassification is caused by the similarity between subsamples of two different classes. For example, as shown in Figure 6-20, the two misclassified subsamples of Clouds0000 look similar to parts of Water0002. Since the classification is done in feature space, the closer the feature vectors of the two samples, the more likely they will be classified into the same class. Our method uses

BGLAMs as feature vectors (see Eq. 6.1). It has been shown in Chapter 4 that when the BGLAMs of two texture samples are close enough, the two samples look similar. Therefore, our BGLAM-based classification method tends to classify similar texture samples into the same class.

However, for failed cases, both Guo et al.'s algorithm and the SGLAM method have classified a Vistex image into a class that has very dissimilar textures. For example, Guo et al's algorithm has misclassified five subsamples of Clouds000 into four classes of different textures, namely Buildings0009, Flowers0000, Paintings0000, and Tile0009 as shown in Figure 6-18, where the two subsamples in Figure 6-20 are classified as Flowers0000. The SGLAM algorithm has misclassified four subsamples of Clouds000 into four different classes: Bark0009, Food0000, Grass0009, and Metal0004, where the two subsamples in Figure 6-20 are classified as Food0000 and Grass0009, respectively. Since the means and standard deviations of image subbands are inadequate to differentiate textures [143, 144], Guo et al's algorithm may classify a texture sample into a class that has completely different textures. As discussed in Chapter 4, textures cannot be effectively differentiated by SGLAMs either, thus the SGLAM method suffers the same problem of classifying an image sample into a class that has dissimilar textures.

With respect to running time, similar to the results for Brodatz textures, our algorithm is faster than SGLAM, but slower than Guo et al.'s algorithm. In the experiments, we have used 24 BGLAMs of size 8×8 for our algorithm, Gabor filters of 4 scales and 6 orientations for both SGLAM algorithm and Guo et al.'s algorithm. For the Vistex textures of 152 learning images and 152 testing images, the running times for our new algorithm, SGLAM, and Guo et al.'s, are 119, 279, and 36 seconds, respectively.

6.4.3 ASI Textures

ASI stands for All-Sky Imager. In recent years, the digital ASI has become an important tool in auroral and magnetospheric physics [174]. Images captured from the ASI play very important roles in studying the environment of the near-Earth space where auroral phenomena occur. Typical examples of auroral phenomena are the northern (or southern) lights, which are caused by the solar-terrestrial interaction. Auroral phenomena usually modify atmospheric properties, e.g. those affecting radio wave propagation. The study of auroras is of great scientific and practical interests. For example, observations of auroras and their associated effects on radio wave propagation provide a means of testing scientific models of the precipitation mechanisms, remote sensing, magnetospheric dynamics, and better understanding the solar-terrestrial interaction [174].

Auroral phenomena are observed and captured through special ground-based network cameras called all-sky imagers [174]. There are tens of millions of ASI images acquired annually around the world. Manual analysis of the ASI images by subjects (or researchers) is tedious and impractical. Full utilization of these powerful ASI data sets demands automated analysis tools. However, one of the challenges in developing such tools is that automatically classifying ASI images according to auroral structures is difficult [174]. In the rest of this subsection, we present classification results of our BGLAMs on ASI images and compare with existing approaches.

The ASI images used for learning and testing are a set of 3400 images (size 128×128), which are randomly taken from a database of 222000 auroral ASI images at the CANOPUS (Canadian Auroral Network for the Open Program Unified Study) [44, 159]. The 3400 texture images are grouped manually by researchers at the Institute for

Space Research at the University of Calgary [84] into three classes, namely Arcs, Patches and Omega-bands as shown in Table 6-7 and Figure 6-21. The number of images for Arc, Patch, and Omega-band is 1200, 1200, and 1000, respectively. For each class, one-third of the images are randomly selected and used for learning and the rest of them are used for testing. For example, among the 1200 Arc images, 400 images are randomly selected and used for learning and the rest of 800 images are used for testing.

Table 6-7: Classes in the 3400 ASI auroral texture images. Example images of each class are given in Figure 6-21. The number in the bracket beside each class name gives the total number of images in that class among the 3400 images.

Class	Name (number of images)	Description
1	Arc (1200)	East-west or north-south aligned arc-like structures. They usually occur at nights.
2	Patch (1200)	Irregular blob-like structures. They usually occur in the late morning to dawn.
3	Omega-band (1000)	Structures that usually occur either following or during substorm activity in the midnight.

The ASI data is collected and given to us by Dr. Eric Donovan and Dr. Mikko Syrjäsuo at the Institute for Space Research at the University of Calgary. The data is in fact a time series. However, to avoid temporal bias, the images in the database are scrambled; thus, it is very difficult to classify images in the database manually without deep understanding in auroral and magnetospheric physics. The testing has also been done in a doubly-blinded manner.

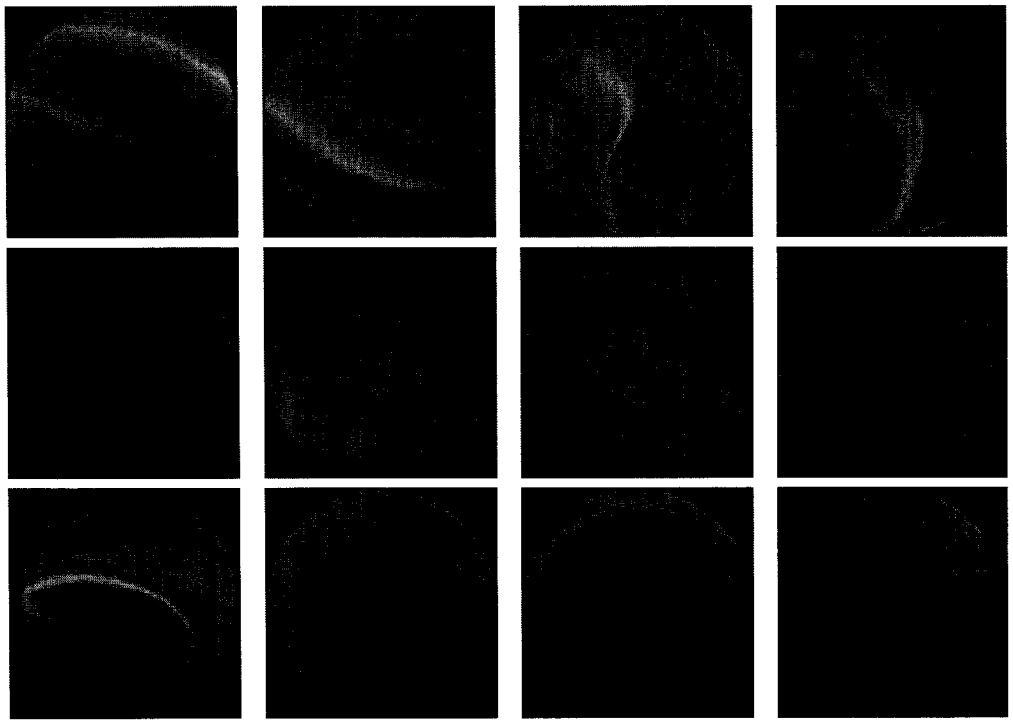


Figure 6-21: Example images of the three classes in ASI database. Images in the 1st, 2nd, and 3rd row contain arcs, patches, and omega-bands, respectively. In the 1st row, the first two images contain east-west aligned arc-like structures, and the last two contain north-south aligned arc-like structures.

The most recent work on classifying ASI images, which is also the only previous work to our best knowledge, was done by Syrjäsuo and Donovan [175]. In their approach, simple statistics such as means and histograms from image pyramids are used to represent texture images in the feature space and the kNN (k-Nearest-Neighbors) classifier is used to classify a given ASI image based on the training images. Table 6-8 gives the comparison results of our approach with theirs. For ease of reference, the results of Guo et al.'s algorithm [72] and SGLAM [152] are also given in the table. For an auroral class, the total number of test images in that class is shown in the bracket beside the class name. For each class, both the total number of images correctly classified by an

algorithm and the associated success rate (see Eq. 6.12) are shown in the table. For example, Table 6-8 shows that there are a total of 800 arc-class images for testing, and that BGLAM correctly classifies 795 out of 800, with a success rate of 99.4% ($795/800 = 99.4\%$). Table 6-8 shows that our algorithm has the best performance.

Table 6-8: The comparison results of our approach with Syrjäsuo and Donovan's, Guo et al.'s algorithm, and SGLAM on ASI images.

	Arcs (800)	Patchy (800)	Omega (667)	Average Success Rate
BGLAM	795 (99.4%)	777 (97.1%)	627 (94.0%)	97.0%
Syr. & Don.	792 (99.0%)	712 (89.0%)	80 (12.0%)	69.9%
Guo et al.	784 (98.0%)	553 (69.1%)	79 (11.8%)	52.5%
SGLAM	786 (98.3%)	636 (79.5%)	309 (46.3%)	76.4%

As demonstrated in previous chapters, a texture image can be uniquely represented by and faithfully reconstructed from BGLAMs, but not by SGLAMs. Therefore, BGLAMs are able to differentiate Omega images from other classes; while SGLAMs are not. Portilla and Simoncelli [143, 144] have shown that simple statistics such as means, variances, histograms from image pyramids are inadequate for differentiating textures. Thus, both Guo et al's method and Syrjäsuo and Donovan's method have difficulty in separating Omega images from images of other types.

6.5 Summary

This chapter describes texture image classification using BGLAMs. Given an unseen texture image, our approach classifies it into one of the pre-learned classes. There are two stages in our algorithm: a learning stage and a classification stage. In the first

stage, models of texture classes are learned from the BGLAMs of training examples using the SVM, and in the second stage, a given texture image is classified into one of the pre-learned classes, to which the image has the largest signed distance. We compare our approach experimentally with existing approaches by performing texture classification over the Brodatz textures, the Vistex textures, and the ASI textures. The experimental results show that our approach has better performance than existing approaches.

Chapter 7

Conclusions

7.1 Summary

Texture modeling plays important roles in computer graphics, vision and image processing. Although many techniques have been developed for the study of texture analysis and synthesis, the mathematical definition of texture is still unclear. Due to the vague definition of texture, each technique has its own advantages and disadvantages, and thus fails to model certain types of textures. Challenging problems in texture modeling are as follows:

1. In computer vision, it is difficult to define textures with mathematical precision. Given a texture sample, what can be used to represent the sample with the necessary and sufficient information?
2. In exiting texture modeling approaches, a good analysis technique may not work well for synthesis; while a good synthesis technique that generates impressive results may have done the analysis poorly or may not be applicable to analysis at all. Therefore, there is a lack of good unified frameworks that work well for both analysis and synthesis.
3. It is difficult to perform 3D texturing. In general, existing 3D-texture techniques are complex to understand and work for only a limited range of textures.
4. It is a challenging problem to evaluate texture-synthesis results quantitatively. Visual inspection is the only way for previous approaches to evaluate their synthesis results.

This thesis presents a new unified mathematical framework for modeling textures using BGLAMs that has successfully addressed the above issues. Our new framework for texture analysis and synthesis will provide important understanding in texture modeling in both computer vision and computer graphics. We prove that BGLAMs forms a basis of GLAMs, and that two images are identical if and only if their corresponding BGLAMs are the same. We also proved that the number of different BGLAMs of a given image is at most equal to the number of pixels in the image. BGLAMs should not be confused with GLCMs. In fact, we have proved that a GLCM can be represented as a sum of two BGLAMs, and have shown that BGLAMs significantly outperforms GLCMs in texture modeling.

Based on the theory, we have developed new techniques for 2D and 3D texture synthesis, respectively, and a new method for classifying texture images using BGLAMs. For 2D texture synthesis, given a sample, our method first characterizes it by a set of BGLAMs. Then, by sampling from the BGLAMs only, our method generates an output texture similar to the input by iteratively modifying the gray level of each pixel in the output image until the distance between the corresponding BGLAMs of the output and those of the input is small enough or until there is no further change in pixel's gray level values in the output. The experimental results show that the new 2D texture-synthesis technique can successfully synthesize a wide range of textures and is comparable to the existing techniques. In addition, based on the BGLAM-based distance measure, our technique is able to automatically evaluate the results and determine whether the output is a successful synthesis of the input. To our best knowledge, none of the previous techniques has the ability to evaluate their synthesis results.

For 3D texture synthesis, we have developed a new technique, called *aura 3D textures*, for synthesizing solid textures from input examples using BGLAMs. Our method is fully automatic and requires no user interaction in the process. Given one or more input textures, a solid texture is generated by sampling the BGLAMs of the input(s) constrained in multiple view directions. Once the solid texture is generated, any given object can be textured by the solid texture using a shader. The evaluation results have shown that our algorithm for solid textures can generate faithful results of both stochastic and structural textures with an average successful rate of 76.4%. As well, the new method outperforms Wei & Levoy's method [188] and is comparable to that proposed by Jagnow, Dorsey, and Rushmeier [86].

In addition to 2D and 3D texture synthesis, we demonstrate that BGLAMS can be used in classifying texture image databases by learning. In the learning stage, models of texture classes (i.e. the optimal boundaries of classes in the feature space) are learned from the BGLAMs of training examples using a SVM algorithm. Given an unknown texture image, our approach classifies it into one of the pre-determined texture classes, to which the image has the largest signed distance. We compare our approach experimentally with existing approaches by performing texture classification using the Brodatz database, the Vistex database, and the ASI database (a real application database). The experimental results show that our classification method has better performance than previous methods. For both the Brodatz database and the Vistex database, the experimental results show that the proposed new approach has performance better than Guo et al.'s algorithm and the SGLAM method. For the ASI database, the results have

shown that our approach significantly outperforms existing approaches with an average successful rate of 97% vs 66%.

7.2 Contributions

The main contributions of this thesis work are as follows:

1. The mathematical theory of BGLAMs. It is proved that BGLAMs form the basis of GLAMs, and that two images are identical if and only if their corresponding independent BGLAMs are the same. At present, to our best knowledge, this work clarifies the relationship between BGLAMs, SGLAMs, GLAMs, and GLCMs. We demonstrate that an image can be uniquely represented by its BGLAMs, but not by GLCMs nor by SGLAMs.
2. A new BGLAM-based method for 2D texture analysis and synthesis. For a given input texture sample, synthetic 2D textures can be generated by sampling a small set of BGLAMs (e.g. 64 BGLAMs for an image of size 64×64) that are calculated from the input.
3. An original BGLAM-based algorithm for synthesizing solid (i.e. 3D) textures from one or more input samples. Our method generates solid textures by sampling the BGLAMs of the input samples constrained in multiple view directions.
4. A new distance function based on BGLAMs for measuring texture similarity. In addition to the metric properties, the BGLAM distance function is one-to-one. This one-to-one property implies that a zero value of the distance measure on two images will guarantee that they are identical, and that the smaller the

distance value, the closer the two texture images look to each other. Based on the BGLAM distance measure, we present an original method for evaluating texture synthesis results quantitatively. For previous synthesis techniques, human visual inspection is the only effective way to evaluate the synthesis results.

5. A BGLAM-based method for texture image classification. We test our method by performing image classification on the Brodatz database, the Vistex database, and the ASI database.

7.3 Future Work

The work in this thesis encourages future research in several directions. When BGLAMs are used in synthesizing both 2D and 3D textures from input samples, our sampling methods tend to converge to local minima quickly, and thus generate visible seams in the synthesized textures. One solution to this problem is to use a genetic-based search method [66] (a multiple-point search scheme) rather than a single-point search scheme during sampling (see Chapter 4 and Chapter 5 for more details). Alternatively, optimization methods based on graph cut techniques [13, 184] can be used to avoid the problem of local minima. It has been demonstrated [13, 184] that graph cuts can be used to efficiently find the global or nearly global optimal solutions for labeling problems in image restoration and stereo and motion analysis. By correctly formulating the labeling problems and by defining appropriate energy functions for our BGLAM-based 2D and 3D texture synthesis, the problem of visible seams can be well resolved using graph cut techniques [13, 184].

Another future research is the GPU (Graphics Processing Unit)-based texture synthesis using BGLAMs. Real-time texture synthesis is in high demand to run in computer games and computer-generated movies. Existing texture-synthesis techniques are computationally expensive, and thus difficult to run in real-time. State-of-the-art graphics hardware makes the GPU available for real-time texture synthesis. The challenge in GPU-based texture synthesis is to develop new techniques that can fit into the graphics processing pipeline. This requires not only a compact representation of textures but also a parallel sampling process in texture synthesis. So far, no technique has been successfully developed to perform GPU-based texture synthesis on given input samples. Using independent BGLAMs, textures can be efficiently stored in memory. Since the BGLAM-based texture synthesis uses a random site-visitation scheme, it is possible to turn the sampling into a parallel process by segmenting the output texture into independent and identically distributed regions.

Future work can also be conducted on unsupervised texture image classification using BGLAMs. The method for texture image classification presented in the thesis is supervised in the sense that the number of texture classes is known a priori. For some applications, the number of classes might not be known first. Therefore, it is desirable to perform unsupervised texture classification by which texture classes are dynamically detected. By treating BGLAMs as texture features and by using BGLAM-based distance measure, it is possible to perform unsupervised texture classification.

Other future research may include: the application of BGLAMs to evaluating synthesized textures from various existing approaches and choosing the best, the study of the sensitivity of the threshold values used in the BGLAM-based distance function for

measuring the similarity of texture samples and for evaluating the synthesis results, dynamic texture [169] synthesis using BGLAMs, texture image segmentation using BGLAMs and its applications to segmenting MRI (Magnetic Resonance Imaging) scans in order to identify brain tumors or breast cancers, and the extension of the BGLAM framework to image or video analysis and synthesis using mathematical morphology [164].

References

1. Abbadeni, N., *A New Similarity Matching Measure: Application to Texture-based Image Retrieval.* The 3rd intl. workshop on texture analysis and synthesis, 2003: p. 1-6.
2. Arivazhagan, S. and L. Ganesan, *Texture Classification Using Wavelet Transform.* Pattern Recognition Letters, 2003: p. 1513 - 1521.
3. Ashikhmin, M., *Synthesizing Natural Textures.* The ACM Symposium on Interactive 3D Graphics, 2001: p. 217-226.
4. Azencott, R., J.P. Wang, and L. Younes, *Texture Classification Using Windowed Fourier Filters.* IEEE PAMI, 1997. **19**(2): p. 148-153.
5. Bard, J., *A Model for Generating Aspects of Zebra and Other Mammalian Coat Patterns.* Journal of Theoretical Biology, 1981. **93**(4): p. 363-385.
6. Bar-Joseph, Z., R. El-Yaniv, D. Lischinski, and M. Werman, *Texture Mixing and Texture Movie Synthesis Using Statistical Learning.* IEEE TVCG, 2001. **7**(2): p. 120-135.
7. Barr, A., *Decal Projections, Course 15: Mathematics of Computer Graphics.* ACM SIGGRAPH, 1984: p. 124-133.
8. Barrett, A.N., *Computer Vision and Image Processing.* New York: Chapman and Hall, 1991.
9. Besag, J., *Spatial Interaction and the Statistical Analysis of Lattice Systems (with Discussion).* Journal of the Royal Statistical Society, Series B, 1974. **36**: p. 192-326.

10. Bier, E.A. and K.R. Sloan, *Two-Part Texture Mappings.* IEEE Computer Graphics and Application, 1986: p. 40-53.

11. Blinn, J.F., *Simulation of Wrinkled Surface.* Computer Graphics, 1978. **12**(3): p. 286-292.

12. Blinn, J.F. and M.E. Newell, *Texture and Reflection in Computer Generated Images.* CACM: Communications of the ACM, 1976. **19**(10): p. 542-547.

13. Boykov, Y., O. Veksler, and R. Zabih, *Fast Approximate Energy Minimization via Graph Cuts.* IEEE PAMI (also in ICCV 99), 2001. **23**: p. 1222-1239.

14. Brodatz, P., *Textures: A Photographic Album for Artists & Designers.* New York: Dover., 1966.

15. Cabral, B., M. Olano, and P. Nemec, *Reflection Space Image Based Rendering.* ACM SIGGRAPH, 1999: p. 165-170.

16. Catmull, E., *A Subdivision Algorithm for Computer Display of Curved Surfaces.* A Subdivision Algorithm for Computer Display of Curved Surfaces, Ph. D. Thesis, Dept. of Computer Science, Univ. of Utah, 1974.

17. Chellappa, R. and S. Chatterjee, *Classification of Textures Using Gaussian Markov Random Fields.* IEEE Transactions on Acoustics, Speech, and Signal Processing, 1985. **ASSP-33**(4): p. 959-963.

18. Chellappa, R., R.L. Kashyap, and B.S. Manjunath, *Model-Based Texture Segmentation and Classification.* in Handbook of Pattern Recognition and Computer Vision (C. H. Chen, L. F. Pau, and P. S. P. Wang, eds.), Singapore: World Scientific., 1993: p. 277-310.

19. Chen, C.H., L.F. Pau, and P.S.P. Wang, *The Handbook of Pattern Recognition and Computer Vision (2nd ed).* World Scientific Publishing Co., 1998.

20. Chen, Y., X. Tong, J. Wang, S. Lin, B. Guo, and H.-Y. Shum, *Shell Texture Functions.* Siggraph, 2004: p. 343-353.

21. Clausi, D.A. and H. Deng, *Fusion of Gabor Filter and Co-occurrence Probability Features for Texture Recognition.* IEEE Transactions on Image Processing, 2005. **14**(7): p. 925-936.

22. Clausi, D.A. and M.E. Jernigan, *Designing Gabor Filters for Optimal Texture Separability.* Pattern Recognition, 2000. **33**(11): p. 1835-1849.

23. Clausi, D.A. and M.E. Jernigan, *A Fast Method to Determine Co-occurrence Texture Features.* IEEE Transactions on Geosciences and Remote Sensing, 1998. **36**(1): p. 298-300.

24. Clausi, D.A. and Y. Zhao, *Grey Level Co-occurrence Integrated Algorithm (GLCIA): A Superior Computational Method to Determine Co-occurrence Texture Features.* Computers and Geosciences, 2003. **29**(7): p. 837-850.

25. Cohen, F.S. and D.B. Cooper, *Simple Parallel Hierarchical and Relaxation Algorithms for Segmenting Noncausal Markovian Random Fields.* IEEE PAMI, 1987. **9**: p. 195-219.

26. Conners, R.W. and C.A. Harlow, *A Theoretical Comparison of Texture Algorithms.* IEEE PAMI, 1980: p. 204–222.

27. Conners, R.W. and C.T. Ng, *Developing a Quantitative Model of Human Preattentive Vision.* IEEE Trans. Syst. Man Cybern., 1989. **19**(6): p. 1384-1407.

28. Cook, R.L., *Shade Trees.* ACM SIGGRAPH, 1984. **18**: p. 223-231.

29. Cook, R.L. and T. DeRose, *Wavelet Noise.* Siggraph, 2005: p. 803-811.

30. Cook, R.L., T. Porter, and L. Carpenter, *Distributed Ray Tracing.* ACM SIGGRAPH, 1984. **18**(3): p. 137-145.

31. Copeland, A.C., G. Ravichandran, and M.M. Trivedi, *Texture Synthesis Using Gray-Level Co-occurrence Models: Algorithms, Experimental Analysis, and Psychophysical Support.* Optical Engineering, 2001. **40**(11): p. 2655-2673.

32. Cross, G.C. and A.K. Jain, *Markov Random Field Texture Models.* IEEE PAMI, 1983. **5**(2): p. 25-39.

33. Daubechies, I., *Orthonormal Bases of Compactly Supported Wavelets.* Communications on Pure and Applied Mathematics, 1988. **41**(7): p. 909-996.

34. Davis, L.S., S.A. Johns, and J.K. Aggarwal, *Texture Analysis Using Generalized Cooccurrence Matrices.* IEEE PAMI, 1979. **PAMI-1**(3): p. 251-259.

35. DeBonet, J.S., *Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images.* ACM SIGGRAPH, 1997: p. 361-368.

36. DeBonet, J.S. and P. Viola, *Texture Recognition Using a Non-parametric Multi-Scale Statistical Model.* IEEE CVPR, 1998: p. 641-647.

37. Deng, H. and D.A. Clausi, *Unsupervised Image Segmentation Using a Simple MRF Model with a New Implementation Scheme.* Pattern Recognition, 2004. **37**(12): p. 2323-2335.

38. Deng, H. and D.A. Clausi, *Gaussian MRF Rotation-Invariant Features for SAR Sea Ice Classification.* IEEE PAMI, 2004. **26**(7): p. 951-955.

39. Deng, H. and D.A. Clausi, *Advanced Gaussian MRF Rotation-Invariant Texture Features for Classification of Remote Sensing Imagery.* IEEE CVPR, 2003: p. 685-690.

40. Derin, H. and H. Elliott, *Modeling and Segmentation of Noisy Textured Images Using Gibbs Random Fields.* IEEE PAMI, 1987. **9**(1): p. 39-55.

41. Dischler, J.-M. and D. Ghazafarpour, *Interactive Image Based Modeling of Macrostructured Textures.* IEEE Computer Graphics and Application, 1999. **19**(1): p. 66-74.

42. Dischler, J.-M., D. Ghazanfarpour, and R. Freydier, *Anisotropic Solid Texture Synthesis Using Orthogonal 2D Views.* Special issue of Proceedings of Eurographics, 1998: p. 87-96.

43. Dischler, J.-M., K. Maritaud, B. Levy, and D. Ghazafarpour, *Texture Particles.* Computer Graphics Forum Vol.21(3), Special Issue of Proceedings of EUROGRAPHICS'02, 2002.

44. Donovan, E.F., T.S. Trondsen, L.L. Cogger, and B.J. Jackel, *Auroral Imaging in Canadian CANOPUS and NORSTAR Programs.* Proc. 28th Annual European Meeting on Atmospheric Studies by Optical Methods, Oulu, Finland, 2002: p. 109–112.

45. Duda, R.O. and P.E. Hart, *Pattern Classification and Scene Analysis.* Stanford Research Institute, Menlo Park, California: John Wiley, 1973.

46. Dungan, W., A. Stenger, and G. Sutty, *Texture Tile Considerations for Raster Graphics.* ACM SIGGRAPH, 1978. **12**(3): p. 130-134.

47. Ebert, D.S., F.K. Musgrave, K.P. Peachey, K. Perlin, and S. Worley, *Texturing & Modeling: A Procedural Approach (3rd Edition).* 2002, Academic Press.

48. Economy, R. and M. Bunker, *Advanced Video Object Simulation.* In Proceedings of the National Aerospace and Electronics Conference (IEEE, New York, May), 1984: p. 1065-1071.

49. Efros, A. and T. Leung, *Texture Synthesis by Non-Parametric Sampling.* IEEE ICCV, 1999: p. 1033-1038.

50. Efros, A.A. and W.T. Freeman, *Image Quilting for Texture Synthesis and Transfer.* ACM SIGGRAPH, 2001: p. 341-346.

51. Elfadel, I.M. and R.W. Picard, *Gibbs Random Fields, Cooccurrences, and Texture Modeling.* IEEE PAMI, 1994. **16**(1): p. 24-37.

52. Elfadel, I.M. and R.W. Picard, *Miscibility Matrices Explain the Behavior of Grayscale Textures Generated by Gibbs Random Fields.* SPIE Conference on Intelligent Robots and Computer Vision IX, 1990: p. 524-535.

53. Fan, L. and K.K. Sung, *A Combined Feature-Texture Similarity Measure for Face Alignment Under Varying Pose.* IEEE CVPR, 2000: p. 1308-1313.

54. Fang, H. and J.C. Hart, *Textureshop: Texture Synthesis as a Photograph Editing Tool.* ACM SIGGRAPH, 2004.

55. Figueiras-Vidal, A.R., J.M. Paez-Borrallo, and R. Garcia-Gomez, *On Using Cooccurrence Matrices to Detect Periodicities.* IEEE Trans. Acoust., Speech, Signal Process, 1987. **35**(1): p. 114–116.

56. Fleischer, K.W., D.H. Laidlaw, B.L. Currin, and A.H. Barr, *Cellular Texture Generation.* ACM SIGGRAPH, 1995: p. 239-248.

57. Fournier, A., D. Fussel, and L. Carpenter, *Computer Rendering of Stochastic Models.* CACM: Communications of the ACM, 1982. **25**(6): p. 371-384.

58. Gagalowicz, A., *A New Method for Texture Fields Synthesis: Some Applications to the Study of Human Vision.* IEEE PAMI, 1981. **3**(5): p. 520-533.

59. Gagalowicz, A. and S.D. Ma, *Sequential Synthesis of Natural Textures.* Comput. Vis. Graph. Image Process., 1985: p. 289-315.

60. Gardner, G., *Visual Simulation of Clouds.* ACM SIGGRAPH, 1985: p. 297-303.

61. Gardner, G., *Simulation of Natural Scenes Using Textured Quadric Surfaces.* ACM SIGGRAPH, 1984. **18**(3): p. 11-20.

62. Geman, D., *Random Fields and Inverse Problems in Imaging.* Lecture Notes in Mathematics, 1991. **1427**: p. 113-193.

63. Geman, S. and D. Geman, *Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images.* IEEE PAMI, 1984. **6**: p. 721-741.

64. Geman, S. and C. Graffigne, *Markov Random Field Image Models and their Applications to Computer Vision.* Proceedings of the International Congress of Mathematicians, 1986: p. 1496-1517.

65. Gersho, A. and R.M. Gray, *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1991.

66. Goldberg, D.E., *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Publishing Company, 1989.

67. Gotlieb, C.C. and H.E. Kreyszig, *Texture Descriptors Based on Co-occurrence Matrices.* Comput. Vis. Graph. Image Process., 1990. **51**(70-86).

68. Graham, R., D. Knuth, and O. Patashnik, *Concrete Mathematics (2nd ed).* Addison-Wesley, 1994.

69. Greene, N., *Environment Mapping and Other Applications of World Projection.* IEEE Computer Graphics Application, 1986: p. 21-29.

70. Grunbaum, B. and G.C. Shephard, *Tiling and Patterns.* New York: W. H. Freeman and Company, 1987.

71. Guo, G., A.K. Jain, W.MA, and H. Zhang, *Learning Similarity Measure for Natural Image Retrieval with Relevance Feedback.* IEEE Transactions on Neural Networks, 2002: p. 811-820.

72. Guo, G., S.Z. Li, and K.L. Chan, *Learning Similarity for Texture Image Retrieval.* ECCV, 2000: p. 178-190.

73. Haeberli, P. and M. Segal, *Texture Mapping as a Fundamental Drawing Primitive.* Fourth Eurographics Workshop on Rendering, 1993: p. 259-266.

74. Haindl, M., *Texture Synthesis.* CWI Quarterly, 1991. **4**(4): p. 305-331.

75. Haralick, R.M., *Statistical and Structural Approaches to Texture.* In Proc. 4th Int. Joint Conf. Pattern Recognition, 1978: p. 45-69.

76. Haralick, R.M., K. Shanmugan, and I.H. Dinstein, *Textural Features for Image Classification.* IEEE Trans. Syst. Man Cybern., 1973: p. 610–621.

77. Haralick, R.M. and L.S. Shapiro, *Computer Vision.* Addison Wesley, 1992.

78. Hassner, M. and J. Sklansky, *The Use of Markov Random Fields as Models of Texture.* The Journal of Computer Graphics and Image Processing, 1980. **12**: p. 357-370.

79. Heckbert, P.S., *Fundamentals of Texture Mapping and Image Warping. Master's Thesis*, in *Dept. of Elec. Eng. and Compt. Sci.* 1989, Univ. of California: Berkeley.

80. Heckbert, P.S., *Survey of Texture Mapping.* IEEE Computer Graphics and Applications, 1986: p. 56-67.

81. Heeger, A.J. and J.R. Bergen, *Pyramid-Based Texture Analysis/Synthesis.* ACM SIGGRAPH, 1995: p. 229-238.

82. Hrbacek, K. and T. Jech, *Introduction to Set Theory (3rd ed).* New York : Marcel Dekker, 1999.

83. Hyvarinen, A., J. Karhunen, and E. Oja, *Independent Component Analysis.* Wiley Interscience, 2001.

84. ISR, U., *Institute for Space Research.* http://www.phys.ucalgary.ca/, 2006.

85. Jagnow, R. and J. Dorsey, *Virtual Sculpting with Haptic Displacement Maps.* Proceedings of Graphics Interface, 2002: p. 125-132.

86. Jagnow, R., J. Dorsey, and H. Rushmeier, *Stereological Techniques for Solid Textures.* Siggraph, 2004. **23**(3): p. 329-335.

87. Jain, A.K. and F. Farrokhnia, *Unsupervised Texture Segmentation Using Gabor Filters.* Pattern Recognition, 1991. **24**(12): p. 1167-1186.

88. Joachims, T., *SVM-Light.* www.cs.cornell.edu/People/tj/svm_light. Univ of Dortmund, Informatik, AI-Unit., 2004.

89. Joachims, T., *Chapter 11 in: Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.).* 1999, MIT Press.

90. Jobanputra, R. and D.A. Clausi, *Preserving Boundaries for Image Texture Segmentation Using Grey Level Co-occurring Probabilities.* Pattern Recognition, 2006. **39**(2): p. 234-245.

91. Jollife, I.T., *Principal Component Analysis.* Springer-Verlag, New York, 1986.

92. Julesz, B., *Textons, the Elements of Texture Perception, and Their Interactions.* Nature, 1981. **290**: p. 91-97.

93. Julesz, B., *Visual Pattern Discrimination.* IEEE Transactions on Information Theory, 1962: p. 84-92.

94. Kato, Z., J. Zerubia, and M. Berthod, *Unsupervised Parallel Image Classification Using a Hierarchical Markovian Model.* IEEE ICCV, 1995: p. 169-174.

95. Kautz, J., K. Daubert, and H.-P. Seidel, *Advanced Environment Mapping in VR Applications.* Computers & Graphics, 2004. **28**(1): p. 99-104.

96. Kautz, J. and M. McCool, *Approximation of Glossy Reflection with Prefiltered Environment Maps.* In Proceedings of Graphics Interface, 2000: p. 119-126.

97. Kautz, J., P.-P. Vazquez, W. Heidrich, and H.-P. Seidel, *A Unified Approach to Prefiltered Environment Maps.* In Eleventh Eurographics Workshop on Rendering, 2000: p. 185–196.

98. Kindermann, R. and J.L. Snell, *Markov Random Fields and Their Applications.* American Mathematical Society, 1980.

99. Kraevoy, V., A. Sheffer, and C. Gotsman, *Matchmaker: Constructing Constrained Texture Maps.* ACM SIGGRAPH, 2003. **22**(3): p. 326-333.

100. Kwatra, V., I. Essa, A.F. Bobick, and N. Kwatra, *Texture Optimization for Example-Based Synthesis.* ACM SIGGRAPH, 2005: p. 795-802.

101. Kwatra, V., A. Schodl, I.A. Essa, G. Turk, and A.F. Bobick, *Graphcut Textures: Image and Video Synthesis using Graph Cuts.* ACM SIGGRAPH, 2003. **22**(3): p. 227-286.

102. Lagae, A. and P. Dutré, *A Procedural Object Distribution Function.* ACM TOG, 2005. **24**(4): p. 1442-1461.

103. Lefebvre, L. and P. Poulin, *Analysis and Synthesis of Structural Textures.* Graphics Interface, 2000: p. 77-86.

104. Leung, T. and J. Malik, *Representing and Recognizing the Visual Appearance of Materials using Three-dimensional Textons.* IJCV, 2001. **43**(1): p. 29-44.

105. Lewis, J.P., *Algorithms for Solid Noise Synthesis.* ACM SIGGRAPH, 1989. **23**(3): p. 263-270.

106. Liang, L., C. Liu, Y. Xu, B. Guo, and H. Shum, *Real-Time Texture Synthesis by Patch-Based Sampling.* ACM Transactions on Graphics, 2001. **20**(3): p. 127-150.

107. Linde, Y., A. Buzo, and R.M. Gray, *An Algorithm for Vector Quantizer Design.* IEEE Transactions on Communications, 1980(COM-28): p. 702-710.

108. Liu, X., Y. Hu, X. Tong, B. Guo, and H.-Y. Shum, *Synthesis and Rendering of Bidirectional Texture Functions on Arbitrary Surfaces.* IEEE Transactions on Visualization and Computer Graphics, 2004. **10**(3): p. 278-289.

109. Liu, Y. and R.T. Collins, *Skewed Symmetry Groups.* IEEE CVPR, 2001: p. 872-879.

110. Liu, Y., R.T. Collins, and Y. Tsin, *A Computational Model for Periodic Pattern Perception Based on Frieze and Wallpaper Groups.* IEEE PAMI, 2004. **26**(3): p. 354-371.

111. Liu, Y. and W.C. Lin, *Deformable Texture: The Irregular-Regular-Irregular Cycle.* Proc. Third International Workshop on Texture Analysis and Synthesis, 2003.

112. Lohmann, G., *Co-occurrence-Based Analysis and Synthesis of Textures.* The 12th IAPR Intl. Conf. Patt. Recog. (ICPR), 1994. **1**: p. 449-453.

113. Ma, W. and B.S. Manjunath, *Texture Features and Learning Similarity.* IEEE CVPR, 1996: p. 425-430.

114. Magda, S. and D. Kriegman, *Fast Texture Synthesis on Arbitrary Meshes.* Proceedings of the 14th Eurographics Workshop on Rendering, 2003: p. 82-89.

115. Manjunath, B.S. and R. Chellappa, *Unsupervised Texture Segmentation Using Markov Random Field Models.* IEEE PAMI, 1991. **13**(5): p. 478-482.

116. Manjunath, B.S. and W.Y. Ma, *Texture Features for Browsing and Retrieval of Image Data.* IEEE PAMI, 1996. **18**(8): p. 837-842.

117. Mathiassen, J.R., A. Skavhaug, and K. Bø, *Texture Similarity Measure Using Kullback-Leibler Divergence Between Gamma Distributions.* ECCV, 2002: p. 133-147.

118. Meinhardt, H., *Models of Biological Pattern Formation.* Academic Press, London, 1982.

119. Miller, G., *The Definition and Rendering of Terrain Maps.* ACM SIGGRAPH, 1986. **20**(4): p. 39-48.

120. Miller-Jr, W., *Symmetry Groups and Their Applications.* New York: Academic Press, 1972.

121. Murray, J.D., *On Pattern Formation Mechanism for Lepidopteran Wing Patterns and Mammalian Coat Makings.* Philosophical Transactions of the Royal Society B, 1981. **295**: p. 473-496.

122. Ohanian, P.P. and R.C. Dubes, *Performance Evaluation for Four Classes of Textural Features.* Pattern Recognition, 1992: p. 819-833.

123. Olano, M. and A. Lastra, *A Shading Language on Graphics Hardware: the Pixelflow Shading System.* ACM SIGGRAPH, 1998: p. 159-168.

124. Paget, R., *Texture Images' Sources.* www.vision.ee.ethz.ch/~rpaget/links.htm, 2005.

125. Paget, R., *An Automatic 3D Texturing Framework.* Proceedings of the Intl. Workshop on Texture Analysis and Synthesis, 2005: p. 1-6.

126. Paget, R., *Strong Markov Random Field Model.* IEEE PAMI, 2004: p. 408-413.

127. Paget, R., *Non-Parametric Markov Random Field Models for Natural Texture Images (Ph. D. Thesis)*, in *Dept. of Computèr Science & Electrical Engineering.* 1999, The Univ. of Queensland: Queensland.

128. Paget, R. and D. Longstaff, *Texture Synthesis and Unsupervised Recognition with a Nonparametric Multiscale Markov Random Field Model.* ICPR, 1998: p. 1068-1070.

129. Paget, R. and I.D. Longstaff, *Texture Synthesis via a Noncausal Nonparametric Multiscale Markov Random Field.* IEEE Transactions on Image Processing, 1998. **7**(6): p. 925-931.

130. Panjwani, D.K. and G. Healey, *Markov Random Field Models for Unsupervised Segmentation of Textured Color Images.* IEEE PAMI, 1995. **17**(10): p. 939-954.

131. Parkinnen, J., K. Selkainaho, and E. Oja, *Detecting Texture Periodicity From the Cooccurrence Matrix.* Pattern Recognition Letter, 1990: p. 43–50.

132. Peachey, D.R., *Solid Texturing of Complex Surfaces.* ACM SIGGRAPH, 1985. **19**(3): p. 279-286.

133. Perlin, K., *Improving Noise.* ACM SIGGRAPH, 2002. **21**(3): p. 681-682.

134. Perlin, K., *An Image Synthesizer.* ACM SIGGRAPH, 1985. **19**(3): p. 287-296.

135. Perlin, K. and E.M. Hoffert, *Hypertexture.* ACM SIGGRAPH, 1989: p. 253-262.

136. Picard, D.K., *Inference for General Ising Models.* Journal of Applied Probability, 1982. **19A**: p. 345-357.

137. Picard, R.W. and I.M. Elfadel, *Structure of Aura and Co-occurrence Matrices for the Gibbs Texture Model.* Journal of Mathematical Imaging & Vision, 1992(2): p. 5-25.

138. Picard, R.W., I.M. Elfadel, and A.P. Pentland, *Markov/Gibbs Texture Modeling: Aura Matrices and Temperature Effects.* IEEE CVPR, 1991: p. 371-377.

139. Picard, R.W. and A.P. Pentland, *Temperature and Gibbs Image Modeling.* 1995, MIT, Cambridge, MA.

140. Pixar, *The RenderMan Interface: Version 3.1.* Pixar, San Rafael, California, 1989.

141. Popat, K. and R.W. Picard, *Cluster-Based Probability Model and its Application to Image and Texture Processing.* IEEE Transactions on Image Processing, 1997. **6**(2): p. 268-284.

142. Popat, K. and R.W. Picard, *Novel Cluster-based Probability Model for Texture Synthesis, Classification, and Compression.* in Proceedings SPIE visual Communications and Image Processing (Boston), 1993.

143. Portilla, J. and E.P. Simoncelli, *A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients.* Int'l Journal of Computer Vision, 2000. **40**(1): p. 49-71.

144. Portilla, J. and E.P. Simoncelli, *Texture Representation and Synthesis Using Correlation of Complex Wavelet Coefficient Magnitudes.* CSIC (the Consejo Superior de Investigaciones Cientificas) Technical Report #54. Deposito Legal: M-10904, 1999.

145. Praun, E., A. Finkelstein, and H. Hoppe, *Lapped Textures.* ACM SIGGRAPH, 2000: p. 465-470.

146. Press, W.H., S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery, *Numerical Recipes in C++: The Art of Scientific Computing (2nd ed).* Cambridge University Press, 2002.

147. Proudfoot, K., W.R. Mark, S. Tzvetkov, and P. Hanrahan, *A Real-Time Procedural Shading System for Programmable Graphics Hardware.* ACM SIGGRAPH, 2001: p. 159-170.

148. Puzicha, J., T. Hofmann, and J.M. Buhmann, *Non-parametric Similarity Measures for Unsupervised Texture Segmentation and Image Retrieval.* IEEE CVPR, 1997: p. 267-272.

149. Qin, X., *Texture Synthesis Results using BGLAMs.* http://www.cs.ualberta.ca/~xuq, 2005.

150. Qin, X. and Y.H. Yang, *Theoretical Analysis of Graphcut Textures.* Dept. of Computing Science, University of Alberta, Technical Report: TR05-26, 2005.

151. Qin, X. and Y.H. Yang, *Basic Gray Level Aura Matrices: Theory and its Application to Texture Synthesis.* IEEE ICCV, Beijing, China, 2005: p. 128-135.

152. Qin, X. and Y.H. Yang, *Similarity Measure and Learning with Gray Level Aura Matrices (GLAM) for Texture Image Retrieval.* IEEE CVPR, Washington, D.C., USA, 2004. **1**: p. 326-333.

153. Qin, X. and Y.H. Yang, *Estimating Parameters for Procedural Texturing by Genetic Algorithms.* Graphical Models, 2002. **64**(1): p. 19-39.

154. Qin, X.J. and Y.H. Yang, *A Generalized Cellular Texture Basis Function.* Proceedings of the Twelfth Annual Graduate Symposium on Computer Science. Dept. of Compt. Sci., Univ. of Saskatchewan, 2000: p. 153-162.

155. Raghu, P.P., R. Poongodi, and B. Yegnanarayana, *Unsupervised Texture Classification using Vector Quantization and Deterministic Relaxation Neural Network.* IEEE PAMI, 1997. **6**(10): p. 1376-1387.

156. Ramamoorthi, R. and P. Hanrahan, *Frequency Space Environment Map Rendering.* ACM SIGGRAPH, 2002: p. 517-526.

157. Ramamoorthi, R. and P. Hanrahan, *An Efficient Representation for Irradiance Environment Maps.* ACM SIGGRAPH, 2001: p. 497-500.

158. Ravichandran, G., E.J. King, and M.M. Trivedi, *Texture Synthesis: A Multiresolution Approach.* in Proc. Ground Target Modeling and Validation Conf., Houghton, 1994.

159. Robertson, P., *Spatial Transformations for Rapid Scan-Line Surface Shadowing.* ACM SIGGRAPH, 1989: p. 35-42.

160. Santini, S. and R. Jain, *Similarity Measures.* IEEE PAMI, 1999. **21**(9): p. 871-883.

161. Schachter, B.J., *Long-Crested Wave Models.* CVGIP: Computer Graphics and Imaging Processing, 1980. **12**: p. 32-41.

162. Schmid, C., *Constructing Models for Content-Based Image Retrieval.* IEEE CVPR, 2001. **2**: p. 39-45.

163. Sedgewick, R., *Algorithms in C, Part 5: Graph Algorithms.* Addison-Wesley, Reading, Massachusetts, 2001.

164. Serra, J., *Image Analysis and Mathematical Morphology.* Academic Press, London, England, 1982.

165. Seymour, L., *Parameter Estimation and Model Selection in Image Analysis Using Gibbs-Markov Random Fields.* The University of North Carolina, Chapel Hill (PhD Thesis), 1993.

166. Simoncelli, E.P., W.T. Freeman, E.H. Adelson, and D.J. Heeger, *Shiftable Multi-Scale Transforms.* IEEE Transactions on Information Theory, 1992. **38**(Special Issue on Wavelets): p. 587-607.

167. Simoncelli, E.P. and J. Portilla, *Texture Characterization via Joint Statistics of Wavelet Coefficient Magnitudes.* In 5th IEEE International Conference on Image Processing, 1998. **1**: p. 62-66.

168. Sims, K., *Artificial Evolution for Computer Graphics.* ACM SIGGRAPH, 1991. **25**(4): p. 319-328.

169. Soatto, S., G. Doretto, and Y. Wu, *Dynamic Textures.* IEEE ICCV, 2001: p. 439-446.

170. Soler, C., M. Cani, and A. Angelidis, *Hierarchical Pattern Mapping.* ACM SIGGRAPH, 2002. **21**(3): p. 673-680.

171. Spitzer, F., *Markov Random Fields and Gibbs Ensembles.* American Mathematical Monthly, 1971. **78**: p. 142-154.

172. Stricker, M. and M. Orengo, *Similarity of Color Images.* Proc. SPIE Storage and Retrieval for Image and Video Databases, 1995: p. 381-392.

173. Swain, M.J. and D.H. Ballard, *Color Indexing.* Int'l Journal of Computer Vision, 1991. **7**(1): p. 11-32.

174. Syrjäsuo, M.T., *Auroral Imaging.* http://aurora.phys.ucalgary.ca/, 2006.

175. Syrjäsuo, M.T. and E.F. Donovan, *Diurnal Auroral Occurrence Statistics Obtained via Machine Vision.* Annales Geophysicae, 2004. **22**(4): p. 1103-1113.

176. Szeliski, R. and H.-Y. Shum., *Creating Full View Panoramic Mosaics and Environment Maps.* ACM SIGGRAPH, 1997: p. 251-258.

177. Tong, X., J. Zhang, L. Liu, X. Wang, B. Guo, and H.Y. Shum, *Synthesis of Bidirectional Texture Functions on Arbitrary Surfaces.* ACM SIGGRAPH, 2002. **21**(3): p. 665-672.

178. Tuceryan, M. and A.K. Jain, *Texture Analysis.* in Handbook of Pattern Recognition and Computer Vision, C. H. Chen and P. S. P. Wang (eds.), World Scientific Publishing Co., 1998: p. 235-276.

179. Turk, G., *Texture Synthesis on Surfaces.* ACM SIGGRAPH, 2001: p. 347-354.

180. Turk, G., *Generating Textures for Arbitrary Surfaces Using Reaction-Diffusion.* ACM SIGGRAPH, 1991. **25**(3): p. 289-298.

181. Upstill, S., *The RenderMan Companion: A Programmer's Guide to Realistic Computer Graphics.* Addison-Wesley, 1989.

182. Vapnik, V.N., *Statistical Learning Theory.* 1998: John Wiley & Sons, New York, 1998.

183. Varma, M. and A. Zisserman, *A Statistical Approach to Texture Classification from Single Images.* IJCV, 2005. **62**(1-2): p. 61-81.

184. Veksler, O., *Efficient Graph-Based Energy Minimization Methods in Computer Vision.* Ph. D. Thesis, Cornell University, 1999.

185. Vistex, vismod.media.mit.edu/vismod/imagery/VisionTexture/vistex.html, 2002.

186. Wang, B., W. Wang, H. Yang, and J. Sun, *Efficient Example-Based Painting and Synthesis of 2D Directional Texture.* IEEE Trans. Vis. Comput. Graph., 2004. **10**(3): p. 266-277.

187. Wang, L., X. Wang, X. Tong, S. Lin, S. Hu, B. Guo, and H.-Y. Shum, *View-Dependent Displacement Mapping.* ACM SIGGRAPH, 2003. **22**(3): p. 334-339.

188. Wei, L., *Texture Synthesis from Multiple Sources.* Siggraph Sketches and Applications, 2003.

189. Wei, L., *Texture Synthesis by Fixed Neighborhood Searching (Ph.D. Thesis)*, in *The Dept. of Elec. Eng.* 2001, Stanford Univ.: Stanford.

190. Wei, L. and M. Levoy, *Fast Texture Synthesis Using Tree-Structured Vector Quantization.* ACM SIGGRAPH, 2000: p. 479-488.

191. Wei, L.Y. and M. Levoy, *Texture Synthesis over Arbitrary Manifold Surfaces.* ACM SIGGRAPH, 2001: p. 355-360.

192. Weinhaus, F.M. and V. Devarajan, *Texture Mapping 3D Models of Real-World Scenes.* ACM Computing Surveys, 1997. **29**(4): p. 325-365.

193. Weszka, J.S., C.R. Dyer, and A. Rosenfeld, *A Comparative Study of Texture Measures for Terrain Classification.* IEEE Trans. Syst. Man Cybern, 1976: p. 269-285.

194. Whiteside, A.E., *Preparing Data Bases for Perspective Scene Generation.* In Proceedings of SPIE, Bellingham, WA, 1989. **1075**: p. 230-237.

195. Wijk, J.J., *Spot Noise - Texture Synthesis for Data Visualization.* ACM SIGGRAPH, 1991. **25**(3): p. 309-318.

196. Williams, L., *Pyramidal Parametrics.* ACM SIGGRAPH, 1983. **17**(3): p. 1-11.

197. Witkin, A. and M. Kass, *Reaction-Diffusion Textures.* ACM SIGGRAPH, 1991. **25**(3): p. 299-308.

198. Wolberg, G., *Digital Image Warping.* IEEE Computer Society Press, Los Alamitos, CA, 1990.

199. Worley, S., *Cellular Texture Basis Function.* ACM SIGGRAPH, 1996: p. 291-294.

200. Wu, Q. and Y. Yu, *Feature Matching and Deformation for Texture Synthesis.* ACM SIGGRAPH, 2004: p. 362-365.

201. Ying, L., A. Hertzmann, H. Biermann, and D. Zorin, *Texture and Shape Synthesis on Surfaces.* Eurographics Workshop on Rendering, 2001: p. 301-312.

202. Zelinka, S. and M. Garland, *Interactive Texture Synthesis on Surfaces Using Jump Maps.* Eurographics Symposium on Rendering, 2003: p. 90-96.

203. Zelinka, S. and M. Garland, *Towards Real-Time Texture Synthesis with the Jump Map.* Eurographics Symposium on Rendering, 2002: p. 99-104.

204. Zhang, J., K. Zhou, L. Velho, B. Guo, and H. Shum, *Synthesis of Progressively-Variant Textures on Arbitrary Surfaces.* ACM SIGGRAPH, 2003. **22**(3): p. 295-302.

205. Zhou, K., X. Wang, Y. Tong, M. Desbrun, B. Guo, and H.-Y. Shum, *TextureMontage: Seamless Texturing of Arbitrary Surfaces From Multiple Images.* Siggraph, 2005: p. 1148-1155.

206. Zhu, S.C., C.E. Guo, Y.N. Wu, and Y.Z. Wang, *What are Textons?* ECCV, 2002: p. 793-807.

207. Zhu, S.C., X. Liu, and Y.N. Wu, *Exploring Texture Ensembles by Efficient Markov Chain Monte Carlo--Toward a "Trichromacy" Theory of Texture.* IEEE PAMI, 2000. **22**(6): p. 554-569.

208. Zhu, S.C., Y. Wu, and D. Mumford, *Filters, Random Fields and Maximum Entropy - Towards a Unified Theory for Texture Modeling.* Int'l Journal of Computer Vision, 1998. **27**(2): p. 1-20.

209. Zucker, S.W., *Finding Structure in Co-occurrence Matrices for Texture Analysis.* CVGIP, 1980. **12**: p. 286-308.