Texton Finding and Lattice Creation for Near-Regular Texture

A Thesis Submitted to the
College of Graduate Studies and Research
in Partial Fulfillment of the Requirements
for the degree of Master of Science
in the Department of Computer Science
University of Saskatchewan
Saskatoon

By
Kevin Sookocheff

Permission to Use

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
176 Thorvaldson Building
110 Science Place
University of Saskatchewan
Saskatoon, Saskatchewan
Canada
S7N 5C9

Abstract

A regular texture is formed from a regular congruent tiling of perceptually meaningful texture elements, also known as textons. If the tiling statistically deviates from regularity, either by texton structure, colour, or size, the texture is called near-regular. If we continue to perturb the tiling, the texture becomes stochastic. The set of possible textures that lie between regular and stochastic make up the texture spectrum: regular, near-regular, irregular, near-stochastic, and stochastic.

In this thesis we provide a solution to the problem of creating, from a near-regular texture, a lattice which defines the placement of textons. We divide the problem into two distinct sub-areas: finding textons within an image, and lattice creation using both an ad-hoc method and a graph-theoretic method.

The problem of finding textons within an image is addressed using correlation. A texton selected by the user is correlated with the image and points of high correlation are extracted using non-maximal suppression. To extend this framework to irregular textures, we present early results on the use of feature space during correlation. We also present a method of correcting for a specific type of error in the texton finding result using frequency-space analysis.

Given texton locations, we provide two methods of creating a lattice. The ad-hoc method is able to create a lattice in spite of inconsistencies in the texton locating data. However, as texture becomes irregular the ad-hoc lattice construction method fails to correctly connect textons. To overcome this failure we adapt methods of creating proximity graphs, which join two textons whose neighbourhoods satisfy certain criteria, to our problem. The proximity graphs are parameterized for selection of the most appropriate graph choice for a given texture, solving the general lattice construction problem given correct texton locations.

In the output of the algorithm, centres of textons will be connected by edges in the lattice following the structure of texton placement within the input image. More precisely, for a texture T, we create a graph $G = (V, E)$ dependent on T, where V is a set of texton centres, and $E = (v_i, v_j)$ is a set of edges, where $v_i, v_j \in V$. Each edge $e \in E$ connects texton centre $v \in V$ to its most perceptually sensible neighbours.

Acknowledgements

It is a pleasure to thank the many people who made this thesis possible.

I am deeply indebted to my M.Sc. supervisor, Dr. David Mould. Throughout my time at the University of Saskatchewan, he has provided encouragement, sound advice, good teaching, and lots of good ideas. I would have been lost without him.

I wish to thank my close friends for the camaraderie, entertainment, and caring they provided. I am thankful for my brothers, Jon and Warren, for providing support, encouragement, and enthusiasm.

I am grateful to Manon Sanscartier for providing love and understanding through a difficult time in my life.

Lastly, and most importantly, I wish to thank my parents, Miles and Joanna Sookocheff. They bore me, raised me, supported me, taught me, and loved me. To them I dedicate this thesis.

To Miles and Joanna Sookocheff.

Contents

List of Tables

List of Figures

List of Abbreviations

DT	Delaunay Triangulation
FRAME	Filters, Random Fields, and Maximum Entropy
GG	Gabriel Graph
HSV	Hue, Saturation, Value
MST	Minimum Spanning Tree
NNG	Nearest Neighbour Graph
RGB	Red, Green, Blue
RNG	Relative Neighbourhood Graph
YIQ	Luminance, In-Phase, Quadrature

Chapter 1

Introduction

Almost all visual scenes contain textured area; tiled floors, grass covered hills, and cloudy skies provide a small sampling. Due to its prevalence, texture plays a role in most applications of computer vision and computer graphics. Image segmentation, depth perception, and scene recognition are problems in computer vision that can partially be solved using knowledge of texture within a scene. In computer graphics, texture is applied to almost all objects in a scene to provide realism or to enhance visual effect.

The research presented in this thesis addresses a problem in texture analysis and synthesis, with a solution providing applications in computer vision and computer graphics. Section 1.1 presents background knowledge on the problem, Section 1.2 describes the problem exactly, Section 1.3 provides motivation for the problem in the context of texture synthesis, Section 1.4 discusses our contribution to the computer science literature, and Section 1.5 provides an outline for the remainder of this thesis.

1.1 Background

Texture is often described as the feel of a surface or fabric. In an image, texture provides a measure of the intensity variation of an image region, quantifying properties such as smoothness, coarseness, and regularity. However, there is no universally agreed upon definition of texture despite its prevalence. One thing most researchers agree on is that a texture possesses both stationarity and locality [70]. Stationarity implies that the mean, variance, and auto-correlation of a function do not change at different function locations, making different neighbourhoods within the function look similar. Locality implies that function values are related to a small set of neighbours, so that each point of the function exerts local influence on its neighbourhood. By varying the neighbourhood size used in the definitions of stationarity and locality we arrive at different texture definitions, from stochastic texture with wide variability within a small neighbourhood, to structured texture with similar neighbourhoods distributed regularly throughout the image. Thus, texture can be defined as a statistical arrangement of perceptually meaningful texture neighbourhoods [77]. Commonly, the neighbourhoods are referred to as *textons* [29].

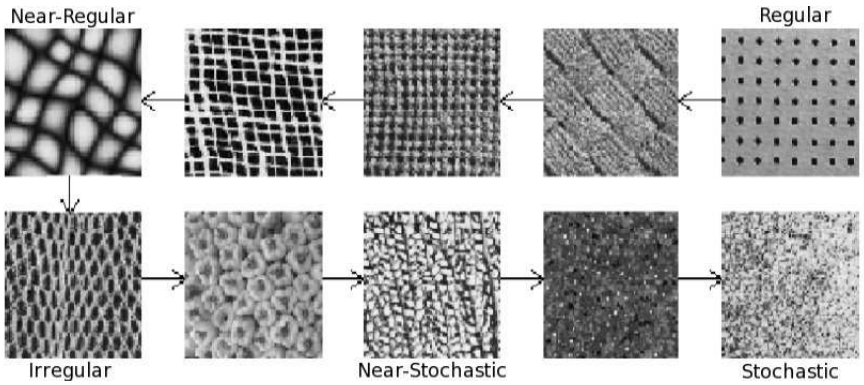
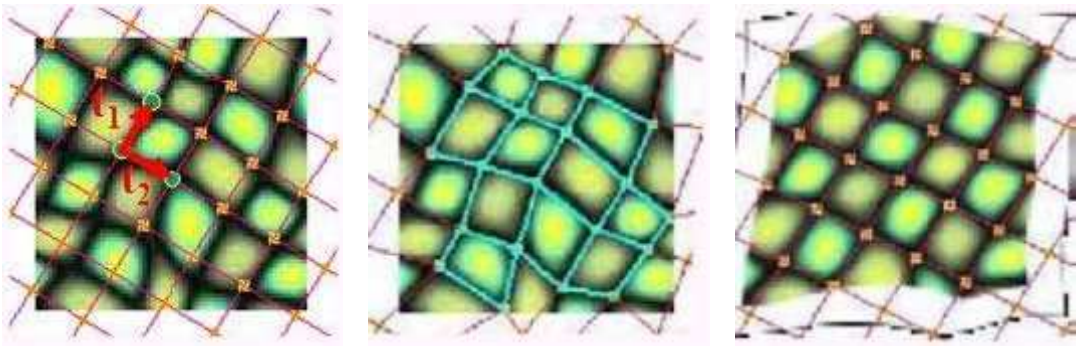


Figure 1.1: The texture spectrum.

Statistical deviations in the probability distribution governing texton placement can determine a texture's regularity. A regular texture is formed from a congruent tiling of textons with no statistical deviations in texton placement. As the tiling deviates from regularity, either in texton structure, colour, or size, we call the texture near-regular [45]. If we continue to perturb the tiling, the texture becomes stochastic. The set of possible textures that lie between regular and stochastic make up the texture spectrum: regular, near-regular, irregular, near-stochastic, and stochastic. Figure 1.1 depicts the texture spectrum, and a sampling of textures within it.

Liu et al. [43–46] provide both motivation and background for the problem at hand. Liu et al. approach the problem of texture synthesis by treating near-regular texture as a deformation of regular texture. A lattice is used to extract the deformation. The lattice extraction procedure begins by having the user create two vectors that approximate texton placement, which the computer overlays on the texture as in Figure 1.2(a). The user must then alter the overlaid lattice to create the final lattice, as in Figure 1.2(b). Finally, the computer uses an energy minimization procedure to deform the texture to a regular version, as in Figure 1.2(c). The regular version of the texture is synthesized and the extracted deformation applied to create a near-regular texture synthesis result.

Liu et al. provide a measure of texture regularity computed along two axes: geometric structure (G) and colour appearance (A). After manually extracting a lattice from a texture, G and A scores are computed by comparing the contents of each texton of the lattice. A texture with a low G score has more regular placement of textons than a texture with a high G score. Similarly, a texture with a low A score has more similarly coloured textons than a texture with a high A score. Depending on the results of these measurements, the textures were classified as one of Type I, II, or III. Type I texture implies regular texton geometry with irregular texton appearance, Type II texture implies irregular texton geometry with regular texton appearance, and Type III texture implies irregular texton geometry and appearance. The near-regular texture synthesis algorithm of Liu et al. depends heavily on the geometric regularity of the input texture, or the G score, and



(a) The user created vectors overlaid on the input texture.

(b) The user-edited lattice.

(c) The regular version of the input texture.

Figure 1.2: Motivation for this work is provided through texture synthesis; we wish to automate the extraction of a lattice in order to automate the texture synthesis method of Liu et al. [45]. Image source: Liu et al. [45]

less heavily on the appearance regularity of the input texture, or the A score; having a low G score significantly aids the texture synthesis procedure. In fact, textures having a high G score cannot be synthesized using the method of Liu et al. In this thesis, we provide a method of extracting a lattice from texture with both low and high G scores, implying that our method will work for some irregular as well as near-regular texture.

1.2 The Problem

In mathematics, a lattice commonly refers to a regularly spaced array of points, which can be subdivided into regions using particular polygonal shapes. Within this work, conventions developed in texture analysis and synthesis are followed [44–46], and the lattice is referred to in an imprecise sense; a lattice is a graph joining textons in a perceptually sensible way.

We provide a solution for the following problem: given a near-regular texture, extract a lattice defining the placement of textons. In the output of the algorithm, centres of textons will be connected by edges in the lattice following the structure of texton placement. More precisely, for a texture T, we wish to create a graph $G = (V, E)$ dependent on T, where V is a set of texton centres, and $E = (v_i, v_j)$ is a set of edges, where $v_i, v_j \in V$. Each edge $e \in E$ connects texton centre $v \in V$ to its most perceptually sensible neighbours.

Perceptually sensible is an imprecise term, and we provide no exact definition. We use the term in an informal sense which appeals to intuition. Good characteristics that we strive for are graph planarity, adequate coverage of empty space, edges of similar length, and lattice regions of similar size. Bad characteristics that we wish to avoid are large areas of empty space, long graph edges, distorted lattice regions, and spurious edges.

We divide the problem of extracting a lattice from near-regular texture into two distinct subareas. The first is finding textons within an image, which we address in Chapter 3. The second is lattice creation using both an ad-hoc method and a graph-theoretic method. We address lattice creation in Chapter 6.

1.3 Motivation

The user-guided lattice extraction method described by Liu et al. [45] provides immediate motivation for our work in automatically extracting a lattice from near-regular texture to further automate near-regular texture synthesis. Furthermore, motivation for this work exists outside of texture synthesis. A texture lattice has been used to perform texture replacement in images and in video [41]. Applications of this work include interior design and digital movie making. In the method, each texton is located and its local scale computed, textons are then replaced by the textons of another texture. Another common problem that motivates this thesis comes in the form of detecting and grouping repeated elements in a scene [36]. A solution to this problem provides applications in computer vision and object recognition. Furthermore, a texture lattice can be used to determine the shape of an object by analyzing the relative placement of textons on the object [19]. Lastly, future applications of a texture lattice may include texture classification, texture compression, and texture blending.

1.4 Contribution

The contribution of this thesis is a novel lattice extraction method for near-regular and some irregular texture. We require a single texton selection from the user as input to the algorithm. Our method involves locating image areas that are highly correlated with a user-selected sample texton, and then connecting these locations to form a lattice. We also provide two extensions to this method. The first comes from using image features to more accurately compute the correlation between the texton and the image. The second comes from using frequency-space analysis to find textons that were not identified using the correlation process.

We provide two methods of constructing a lattice from texton locations. The first method is ad-hoc and correctly connects texton locations while ignoring erroneous data. However, the method fails as texture becomes irregular. The second method is graph-theoretic and uses proximity graphs to connect texton locations to their sensible neighbours.

A limitation of our approach is that we do not provide a method of automatically evaluating the quality of an extracted lattice. This limitation is important because in difficult lattice extraction cases we require user input in deciding the right combination of techniques required to correctly extract a lattice. If lattice quality could be accurately and automatically measured, we could

fully automate the lattice extraction procedure following the initial texton selection. We leave this limitation as an area of future work.

1.5 Thesis Outline

In the next chapter of this thesis, we review literature pertinent to the problem of lattice extraction from near-regular texture. Chapter 3 then details our method of locating textons in an image using correlation. In some cases our texton locating method returns erroneous data. To improve our texton locating result we provide extensions to our texton locating method, which are described in Chapters 4 and 5. We then present two methods of connecting texton locations in Chapter 6. Finally, we conclude the thesis and present avenues for future work in Chapter 7.

Throughout the thesis reference is made to several image processing techniques. Consult Appendix A for brief explanations of Canny edge detection, the Laplacian-of-Gaussian filter, the Sobel operator, morphological shrinking, and non-maximal suppression. Detail on colour spaces can be found in Appendix B. An introduction to the Fourier transform is presented in Appendix C. Appendix D provides listings of the algorithms used in the thesis. Finally, Appendix E reports the full test results presented in this thesis.

Chapter 2

Literature Review

In this chapter we present a survey of past and current research. This survey outlines research in texture modelling, texture analysis, feature extraction from images, texture regularity measurement, texture synthesis, and proximity graphs. These rather disparate areas all contribute to, and provide precedent for, lattice extraction from near-regular texture.

Texture modelling establishes a working definition of texture. We do not deal explicitly with a texture model, but use the ideas from texture modelling theory to approach the problem at hand. Texture analysis generally uses properties of a texture model to understand the underlying properties of an image. We may use these ideas to characterize image attributes when locating potential textons in the image. Feature extraction and image segmentation provide methods of locating objects within an image, providing us with ideas for locating textons. Regularity measurement aids in the problem of locating textons within an image, and in deciding where textons lie in relation to one another. Lastly, texture synthesis provides the initial motivation for this problem: automatically extracting a texton lattice.

2.1 Texture Modelling and Texture Analysis

No consensus texture model has been established; the wide variety of texture has led to a wide variety of explanatory models. Currently the most widely adopted model, and arguably the most sophisticated, is FRAME [77]. FRAME is a statistical modelling technique combining ideas from filter modelling theory, where a texture is decomposed into sub-bands using filters, with ideas from statistical modelling theory, where texture is a realization of a probability distribution on a random field. The combination of these two theories produces a probability model, with a maximum likelihood estimate of the model parameters providing the simplest explanation of the given texture.

Zhu and Guo [23,76], along with other researchers, provide a thorough treatment of textons with respect to texture modelling. Specifically, they integrate descriptive texton models, such as Markov random field, with generative texton models, such as hidden variable, under a single framework. The result is a multi-layer texture model, each layer made up of textons embedded in the image by a stochastic process. Treated within a probabilistic framework, the model assumes a number

of hidden variables: the number of layers, the arrangement of textons at each layer, and noise. Computationally learning the variables amounts to discovering the texture model.

Gimel'farb [20] models texture using Gibbs random fields, which are specific instances of Markov random fields. He argues that true model parameters, approximated using the FRAME methodology, are not the goal of texture synthesis. Rather, the goal is proximity between a training distribution and a generated distribution. The training distribution is derived from a texture sample; Gimel'farb produces a probability model from the sample based on translation invariant pixel pairs and minimizes the error between distributions using simulated annealing. Using pixel pairs increases the capability of modelling structural texture by accounting for relative spatial distance between pixels in the distribution. The Gibbs random field model has generated interest in texture modelling, leading Gimel'farb [21] to address the short-comings of the FRAME model and argue for the use of a Gibbs model (or something similar) in its place.

Many attempts at defining texture properties derive from the assumption that textures sharing second order statistics cannot be immediately distinguished by the human visual system [29, 30]. Using second order statistics, texture analysis can consider texture structure and provide global methods of quantifying texture properties [2,22,25]. Another method of analyzing texture is through the frequency-space representation of an image [28, 42], providing a global similarity measure for texture. The steerable pyramid, presented by Simoncelli and Freeman [61], defines a translation and rotation invariant wavelet representation for images that can be used to provide a statistical model of texture at different resolutions [58].

The choice of texture model is important in defining our problem and deriving a solution. We do not deal explicitly with a texture model, though many ideas and insights into a lattice extraction algorithm come from assuming various properties of near-regular texture, as defined by the models presented here.

A texture model provides the basis for methods of texture analysis, which provide a working definition of texture, as well as techniques to analyze texture properties. Such texture information can be used to identify textons with certain characteristics. However, we have found that statistical texture analysis does not allow for the segmentation of textons from near-regular texture, due to the variability of textons within each texture.

2.2 Texture Synthesis

Texture synthesis methods can be divided into two broad categories: statistical and structural. Statistical methods involve manipulation of the parameters responsible for creating random fields, and sampling these fields to synthesize a texture. Structural methods, on the other hand, involve manipulation of pixels while preserving their spatial relationships, thereby preserving the appearance

of structures in a texture.

A widely used statistical texture synthesis method based on noise was given by Perlin [57]. Adequately defined texture models can be used to synthesize texture by sampling pixels from the texture model [77]. Another common statistical texture synthesis method involves modifying an image of random white noise to portray the characteristics of a texture sample; the statistical model of the texture sample may be found using an image histogram, co-occurrence matrix, sub-band decomposition, or other texture analysis artifact [23, 26, 76, 77].

Moving away from statistical synthesis methods, structural texture synthesis originated as the copying of individual pixels from a sample texture into a new texture, taking care to preserve local relationships in the sample texture [7, 18, 70], and has been extended in a number of ways to increase the speed, appearance, quality, and dimension of synthesis [1, 14, 17, 32, 37, 52, 72, 73]. Around the year 2000, researchers began using texton structure to guide the texture synthesis process [15, 59, 71, 74, 75].

Liu et al. [44–46] provide a near-regular texture synthesis method using the idea of deformation fields. Deformation fields model near-regular texture as a deformation of regular texture. This deformation may occur along one of three axes: texton colour, scale, and geometry. A lattice is used to extract the deformation. The lattice extraction procedure begins by having the user create two vectors that approximate texton placement which the computer overlays on the texture. The user must then alter the overlaid vectors to create the final lattice. Finally, the computer uses an energy minimization procedure to deform the texture to a regular version. The regular version of the texture is synthesized and the extracted deformation applied to create a near-regular texture synthesis result. The user-guided lattice extraction process provides immediate motivation for our work in automatically extracting a lattice from near-regular texture.

The work of Liu et al. inspired several researchers to examine near-regular texture more thoroughly. Specifically, Nicoll et al. [53] use frequency-space texture analysis to determine the periodicity of a texture, which guides the texture synthesis process. Work by Djado [16] discusses a problem similar to ours; the extraction of a representative tile from a near-regular texture. However, the method assumes that textons are geometrically identical, which severely limits the applicability of the result.

Our research is directly related to the work of Liu et al. [44–46]. Specifically, having to manually extract the deformation field required for near-regular texture synthesis provides motivation for this thesis. In this thesis, we develop a semi-automatic procedure for the extraction of a lattice, which can in turn be used to automatically extract the deformation field required for near-regular texture synthesis.

2.3 Feature Extraction and Segmentation

For our purposes, determining the correct location of textons in an image is integral to constructing a sensible lattice. Much research has been performed in extracting objects from images, with results that correctly extract textons in certain situations. However, since we are dealing with near-regular texture our approach requires extraction of textons with varying geometry, colour, and size. This is a previously unsolved problem.

The most common approach to texton extraction is measurement of the response of an image to a filter set. If a location in the image provides high response to an applicable filter set, a texton exists at this location. If a location in the image provides low response to an applicable filter set, no texton exists at this location.

Blostein and Ahuja [6] convolve many Laplacian-of-Gaussian filters over an image, each filter having a different standard deviation. The filter is particularly sensitive to regions having little colour variation; these regions are reported as texture elements. Unfortunately, textons with varying colour will be incorrectly extracted, and the general texton extraction problem remains.

Grouping of repeated image elements has been examined by Malik, Leung, and colleagues [36, 47, 48] as a method of segmenting texture. In 1996 [36], they proposed a method of detecting, localizing, and grouping repeated image elements using ideas from temporal tracking; repeating scene elements can be represented by affine transformations of a single parent scene element. Given a parent element, the search for transformed versions is straightforward, though time consuming. A problem lies in determining the parent element, where Malik and Leung use a second moment statistic computed within a window to determine the significance of a region. After computing the statistic at each point in the image, the most significant region is selected as the parent element. Unfortunately, through our own implementation, we have found that the second moment statistic does not allow us to locate textons in general.

Malik and Leung [47,48] continued to make progress on segmenting textons in 1999 and 2001 by measuring the response of each image pixel to a set of linear filters of different size and orientation. A total of 36 filters are used and their response is mapped to a 36-dimensional space. Clusters in this space represent image locations with similar filter response, and hence similar appearance. By combining cluster information with data from contours, texton locations can be found. This feature extraction method shows promise for texton locating but is computationally demanding.

In 1999, Schaffalitzky and Zisserman [60] reported a different algorithm for detecting repeated image elements. In their algorithm, elements are extracted based on edge information, corners, and closed curves. Regions with roughly uniform pixel intensity surrounded by closed curves are assumed to represent textons. Textons are grouped by searching the image for affine transformed versions of each scene element. However, near-regular texture allows for textons with varying colour,

and regions with uniform pixel intensity do not correspond directly to textons.

Belongie et al. [3–5] developed a shape descriptor based on image context. First, the image is processed to highlight edges. Sample points from the image edges are then used to compute a descriptor roughly describing the shape of a region. This is done by creating a set of bins subdividing the image into partitioned concentric circles surrounding a sample point. A histogram records how many sample points lie within each bin. The set of histograms for all sample points becomes the descriptor of shape for the image, which highlights contextual information in the form of location with respect to other sample points. However, having to evaluate the descriptor for each potential texton location in the image is computationally demanding.

Feature extraction and image segmentation applies to our research. If a method of extracting textons from any texture is found, our research can create a perceptually sensible lattice. However, we have found that the feature extraction literature does not provide significant insight into general texton extraction. Because of this, we require the user to select a typical texton sample from the image, which we use to search for locations in the image with characteristics similar to the texton sample. This approach is able to locate the majority of textons in a texture, but failure cases do occur.

2.4 Regularity Measurement

Quantifying texture regularity commonly involves extracting a periodic signal based on texton locations. The most widely used tools in this pursuit, the Fourier transform and auto-correlation, come from the digital signal processing literature.

Matsuyama et al. [50] analyze texture structure by looking at the frequency-space representation of an image using the Fourier transform. Peaks in the frequency surface represent frequencies that occur regularly in the image. For near-regular texture, such regularly repeating frequencies correspond to the frequency of texton placement. Vectors representing relative texton placement are extracted from the frequency surface and region growing around frequency peaks is used to isolate individual textons.

Assuming that textures are characterized by textons and their spatial arrangement, Vilnrotter et al. [68] compute an edge repetition array, essentially a co-occurrence matrix for edges, to determine texton size and spacing. From this description, texton primitives are extracted based on similarity to the edge repetition array.

In 1989, Hamey [24] proved many of the results necessary for further work in texture regularity measurement. Working with image features rather than raw pixel intensities, Hamey showed how a dominant feature can be tiled along the plane using a minimally short vector, called the fundamental frequency vector, to produce a regular texture. These fundamental frequency vectors

form an orthogonal texton basis of minimum size. Hamey extracts features, using Blostein and Ahuja's algorithm [6], and associates a prominence value to each feature. Using the dominant feature assumption, Hamey connects prominent features using minimal length fundamental frequency vectors. Regular texture is synthesized by placing the dominant feature at integer displacements of the fundamental frequency vectors.

Most attempts at measuring texture regularity find locations of a repeating element, then connect repeating elements with fundamental frequency vectors. However, finding locations of repeating elements using a frequency-space image representation is difficult; extracting peaks from the frequency image is challenging due to noise. Auto-correlation, on the other hand, produces an image with significantly less noise, making finding fundamental frequency vectors robust.

Lin et al. [39] were early adopters of auto-correlation for regularity extraction. In their method an auto-correlation surface is found for an image, and peaks within this surface are isolated using a novel Gaussian smoothing algorithm. Finally, a generalized Hough transform determines the direction of the fundamental frequency vectors, which are found by truncating lines found by the Hough transform at peaks of the auto-correlation surface. Later, Lin et al. [40] used texture features computed for a texture primitive found using this method as a means of classifying images for database retrieval. We have adapted the work from Lin et al. to extract the locations of textons within an image. However, instead of using auto-correlation to compute a surface with peaks defining image regularity, we use correlation to create a surface with peaks defining texton locations.

Chetverikov [11] used the auto-correlation approach to find a contrast function of an image. The contrast function is a profile of the auto-correlation surface along the direction of a fundamental frequency vector. This profile provides a measure of texture regularity based on how close the profile models a harmonic function. Chetverikov [12] showed that a grey-level difference histogram can be used as an alternative to auto-correlation, with the same purpose and result. Leu [35] used a similar approach to provide a measure of regularity using the auto-correlation of an image's gradient.

Detecting regular repetitions under perspective skew has been addressed by Tuytelaars et al. [66]. Regularity is detected as periodicity, symmetry, and reflection. The authors wish to extract repeating elements from real-world scenes, rather than from images. They avoid auto-correlation, which requires a homogeneous input image; they use filter-based search for regions of similar intensity or geometry. Groupings of repeated elements are found via Hough transform.

Liu et al. [43] provide an algorithm for determining the underlying pattern of regular textures. Again, auto-correlation and the Hough transform are used. However, Liu et al. use a measurement of auto-correlation peak prominence that is indirectly related to peak height; the region of dominance of each peak is computed as the distance of a peak to one of larger height. After finding

the fundamental frequency vectors with the Hough transform, the pattern may be classified into a frieze or wallpaper group.

As an alternative to the signal processing approach to regularity measurement, Parkinnen et al. [56] use co-occurrence matrices to detect periodicity. By computing the co-occurrence matrix at different distances, and comparing these matrices using χ^2 and κ statistics, periodicity of a texture can be measured. Starovoitov et al. [63] use co-occurrence to measure periodicity, defining structural features based on the co-occurrence matrix. The authors compare Haralick's texture features [25] with their own, concluding that no feature correctly quantifies structure for all texture samples. Finally, Oh et al. [54] use the diagonal property of a regular texture's co-occurrence matrix to find an equivalent, but less computationally expensive, method of calculating regularity.

Lastly, as part of the MPEG-7 image standard, Manjunath et al. [49] define a Perceptual Browsing Component, which measures degree of texture structure. To obtain this measure, an image is processed with a set of Gabor filters of varying size and orientation, with the result of each filter stored. A projection of the filtered images along the horizontal and vertical axes is made, creating a periodic signal for regular textures.

In much of our early research on lattice extraction from near-regular texture we relied on an estimate of texture regularity gained using auto-correlation to determine the placement of textons within the image [62]. Our current method uses correlation as a means of extracting texton locations. Given a sample texton, we correlate the sample with the image and isolate maxima in the resultant surface to determine the locations of textons in the image. The procedure correctly extracts a lattice from most near-regular texture, but fails as texture becomes irregular. We offer improvements to this method using correlation within feature space, and by analyzing the frequency-space representation of texton locations to correct for errors in the texton locating result. We also offer a method of creating a lattice from irregular texture using proximity graphs.

2.5 Proximity Graphs

We use proximity graphs as a means of connecting texton locations in a perceptually sensible manner. The material in this section provides background on proximity graphs but is not intended as an exhaustive review of the literature.

Given a set of points in the plane, we can construct a graph that connects every two points satisfying certain conditions. For our purposes, we take the location of each texton and construct a graph connecting each texton to its appropriate neighbours. The problem of creating a graph given a set of points is well studied, and we adapt the literature on proximity graphs to the lattice construction problem.

The most sparse proximity graph that could realistically be applied to our problem is the nearest

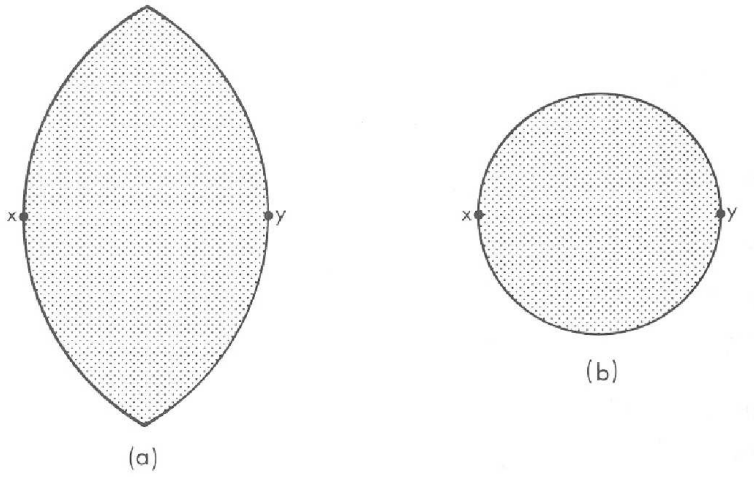


Figure 2.1: The lune of influence (a) and circle of influence (b) of a point pair. Image source: Kirkpatrick and Radke [31].

neighbour graph (NNG). For a set of points S, a NNG connects each point p to the point $q \in S \setminus p$ having minimum Euclidean distance to p. The NNG is not necessarily connected, and so fails as a suitable lattice. Insisting that the texton lattice be described by a connected graph, we turn to the minimum spanning tree (MST) as a means of constructing a lattice. The definition of a spanning tree ensures that every point in the graph is connected through some sequence of edges. The MST chooses such a tree with minimal total edge length. Unfortunately, the MST is a sparsely connected graph, and thus we cannot use this graph to construct a texton lattice where textons may have many neighbours.

Toussaint [64, 65] provides a definition of the relative neighbourhood graph (RNG), which is a slight variation of the original definition given by Lankford [34]. Two points x and y in point set S are said to be relative neighbours, and joined by an edge in the graph, if

$$d(x, y) \leq min\,(max\,(d(x, z), d(y, z))\,|z \in S),$$

where $d(x, y)$ measures Euclidean distance between points x and y. Visually, x and y are relative neighbours if the lune of influence described by the equation above is empty. The lune of influence is constructed as the intersection of the two circles of radius $d(x, y)$, centred at points x and y. Figure 2.1(a) depicts the region of influence of two points x and y.

A graph constructed in a manner similar to the RNG is described using the circle enclosing two points. The Gabriel graph (GG) [51] connects points x and y in S if the unique circle with radius $\frac{d(x,y)}{2}$ passing through points x and y is empty. Figure 2.1(b) depicts the circle of influence of two points x and y.

A unifying property of the preceding graphs is that they are all subgraphs of the Delaunay

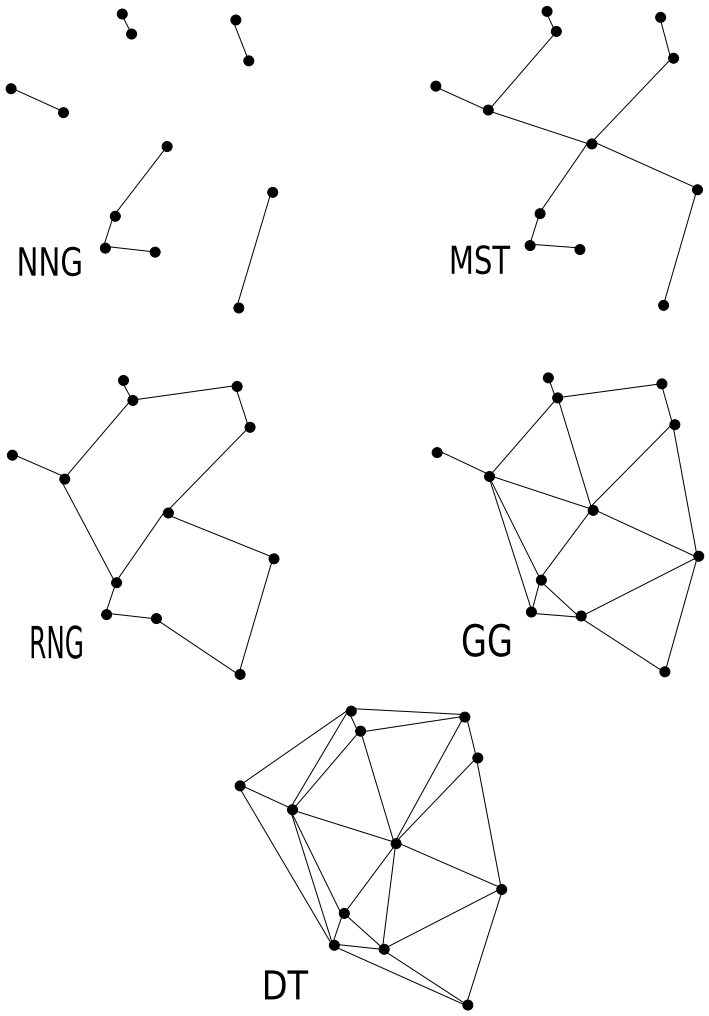


Figure 2.2: NNG, MST, RNG, GG, and DT of a fixed point set. Image adapted from: Kirkpatrick and Radke [31].

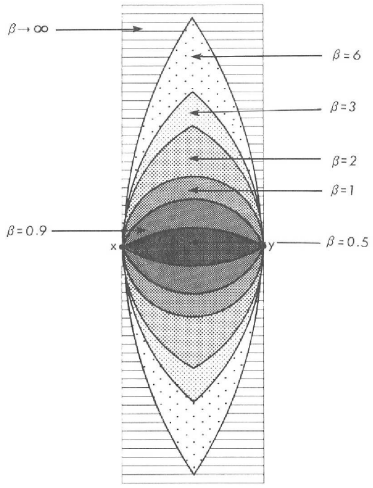


Figure 2.3: Lune-based neighbourhoods for various β. Image source: Kirkpatrick and Radke [31].

triangulation (DT). The DT is the unique triangulation that maximizes the minimum of the angles in the triangulation, creating a triangle of edges from each three points residing in a circle enclosing only themselves. In general, NNG $\subseteq$ MST $\subseteq$ RNG $\subseteq$ GG $\subseteq$ DT [65]. Figure 2.2 depicts the preceding graphs for a fixed point set.

Kirkpatrick and Radke [31] extend the idea of proximity graphs to a continuous spectrum. The hierarchy of the NNG, MST, RNG, GG, and DT suggest a unifying framework based on the region of influence between each pair of points. Kirkpatrick and Radke call such a unification the β-skeleton spectrum of a point set; for a given β, they construct a graph called the skeleton, and the combination of all such skeletons defined for $0 \leq \beta < \infty$ make up the β-skeleton spectrum. For $\beta \geq 1$, the β region of influence is defined as the intersection of two circles of radius $\frac{\beta d(p_i, p_j)}{2}$ centred at points $(1 - \frac{\beta}{2})p_i + (\frac{\beta}{2})p_j$ and $(\frac{\beta}{2})p_i + (1 - \frac{\beta}{2})p_j$. By this definition, when $\beta = 1$ the β region of influence is equal to a circle, and the β-skeleton equals the GG, and when $\beta = 2$ the β region of influence between two points p_i and p_j is a lune, and the β-skeleton equals the RNG. As β approaches ∞, the β region of influence between two points p_i and p_j is the strip between points p_i and p_j, and the β-skeleton equals the MST. Extending the definition for $0 \leq \beta < 1$, the β region of influence between two points p_i and p_j becomes the intersection of two circles of radius $\frac{d(p_i, p_j)}{2\beta}$ passing through p_i and p_j. As β approaches 0, the β region of influence becomes the line segment joining p_i and p_j, resulting in a fully connected graph except in cases with three or more collinear points. Figure 2.3 shows the effect that varying β has on the region of influence of two points.

In our solution to the lattice extraction problem, we use proximity graphs to connect texton locations that have been found using correlation. We have also developed a second method of

connecting texton locations which provides an ad-hoc solution to the problem [62]. Procedures for connecting texton locations using both the ad-hoc method and proximity graph method are discussed in Chapter 6. However, before textons can be connected, they must be found. The next chapter discusses how we use correlation to locate textons within a texture.

Chapter 3

Locating Textons

Locating textons is the major problem encountered in creating a lattice from near-regular texture. We cast the problem as a search for locations in the image similar to a candidate texton. Other methods of locating textons exist (see Section 2.3 of the Literature Review), and the problem is closely related to research in image segmentation, but our method is robust, while still being simple.

The primary tool of our texton locating method is correlation. In Section 3.1 we discuss the properties of correlation, and how correlation has been applied to the problem of finding texton locations. We then present, in Section 3.2, analysis of the success of our texton locating procedure.

3.1 Sample Correlation

To locate textons within an image we create a surface with maxima at texton locations. Such a surface may be computed using frequency-space analysis, correlation, co-occurrence matrices, or object recognition [11, 12, 40, 43, 50, 54, 56, 63, 66, 68]; we chose correlation. We have found, and recent literature confirms [43, 66], that correlation is a robust and effective technique for producing such a surface.

Correlation exists in many fields, and possesses many definitions depending on context and need. We refer to correlation as it exists in the signal processing literature. To this end, we view texture as a two-dimensional signal. Figure 3.1(a) shows a sample texture, Figure 3.1(b) shows a small region of the texture, and Figure 3.1(c) shows the texture sample in Figure 3.1(b) represented as a two-dimensional signal.

To produce a correlation surface with peaks corresponding to texton centres, a sample texton is correlated with the input texture. We have attempted to automatically extract a sample texton from the input image without success. Automatically extracting a sample texton is a difficult problem because of the generality of texture. To correctly solve the problem would require a method which can correctly locate a sample brick, tomatoe, roof tile, rock, reptile scale, etc., with no a priori information about the image. Furthermore, we have not been able to find a method from the image segmentation literature which can be applied to our problem. Therefore, we request

(a) A texture. Image Source: Brodatz [9].

(b) A sample from the texture.

(c) The sample represented as a two-dimensional signal.

Figure 3.1: A texture may be represented as a two-dimensional signal.

user input in selecting a typical texton from the image. In our implementation the selected region is always rectangular; we have found that rectangular regions are easy to implement and provide good results for the majority of texture samples.

Correlation can be described as a filtering operation. A correlation coefficient is calculated as the piecewise multiplication of the pixels in the sample window with the pixels of the image under the sample window. In general, when given a sample window h of size $I \times J$ pixels, and an input image f, we can compute correlation value for window position (m, n) as:

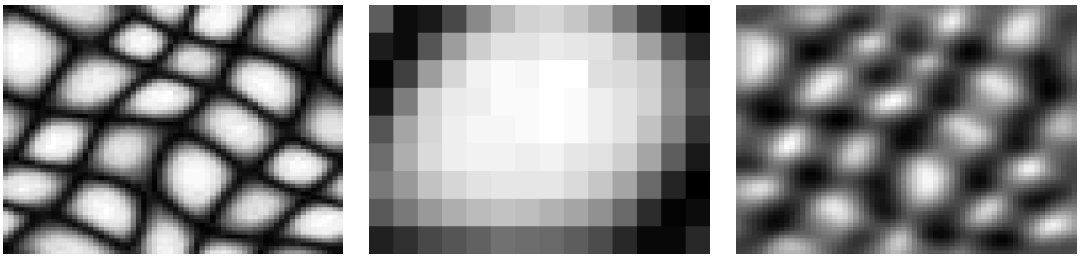
$$g(m,n) = \sum_{i=\frac{-I}{2}}^{\frac{I}{2}} \sum_{j=\frac{-J}{2}}^{\frac{J}{2}} h(i,j) f(m+i, n+j)$$

We produce a correlation surface by repeating the above computation at each location in the image, multiplying the contents of the user-selected sample texton, h, with the contents of the image, f. However, care must be taken near image boundaries; the contents of the sample window centred at a particular location may extend beyond the image domain. In our implementation pixels outside the image domain contribute a coefficient of zero so that

$$h(i,j) f(m+i, n+j) = 0,$$

for $(m+i, n+j) \notin D$, where D is the image domain. As a final note, the image must be mean centred before extracting a texton sample and before correlation of the sample with the input image. Mean centreing is accomplished by subtracting the average value in the image from every pixel, resulting in a correlation surface that is independent of image brightness.

Within the correlation surface, maxima represent image locations that are similar to the sample window and minima represent image locations that are dissimilar to the sample window. Figure 3.2(a) depicts a texture, and Figure 3.2(b) depicts a sample extracted from the texture. The correlation procedure described above is applied to filter Figure 3.2(a) with the contents of Figure



(a) A texture. (b) A sample from the texture. (c) The result of (b) correlated with (a).

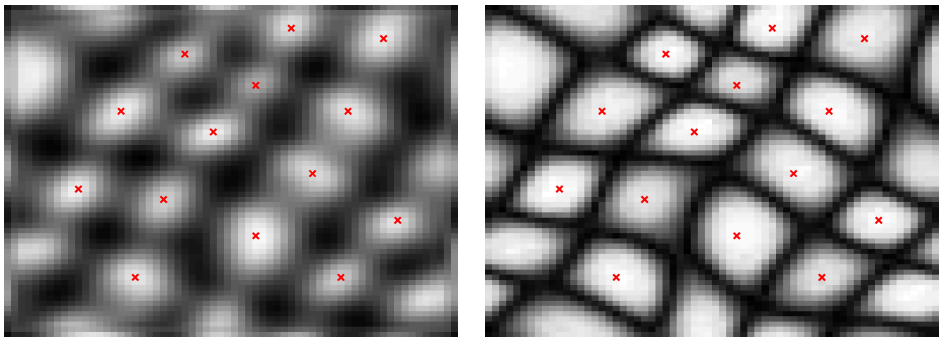
Figure 3.2: A texture is correlated with a sample texton to produce a correlation surface with maxima at texture locations that appear similar to the sample texton.

3.2(b) to produce the correlation surface given in Figure 3.2(c). Appendix D references Matlab* code that implements the above correlation procedure; Program Listing D.2 is used to select a sample from the input image and Program Listing D.3 performs the correlation procedure.

The correlation surface is sometimes noisy. However, the work of Lin [38, 39] provides a convenient way to smooth a correlation surface using a Gaussian function. A Gaussian function, also known as a normal function, is symmetric about the origin, with amplitude decreasing with movement away from the origin. The spread parameter, σ, defines the rate of amplitude decrease. A small value of σ produces a Gaussian with narrow width and rapid amplitude decrease. A large value of σ produces a Gaussian with thick width and slow amplitude decrease. Filtering the image using a Gaussian function averages the value of each pixel using contributions from the pixel's neighbours; the contribution of each pixel corresponds to the amplitude of the Gaussian function at that pixel. Therefore, small σ will smooth each pixel using small contributions from its neighbourhood, smoothing slightly, while large σ will smooth the surface using large contributions from its neighbourhood, smoothing greatly.

Lin's smoothing algorithm initially sets σ to a small value, $\sigma = 1$, and filters the surface with the Gaussian. The smoothed surface should have fewer maxima than the original surface. If this is true, σ is increased to the square root of the maximum distance between any two minimally close peaks and filtering is repeated. This sequence repeats until the number of maxima does not decline as σ is increased, meaning that no maxima are eliminated between iterations, and thus maxima attributed to noise have been eliminated. Though the potential for over-smoothing exists, using this algorithm generally provides improved texton locating results compared to texton locating without smoothing. Finally, in order to renew the prominence of maxima in the correlation surface, we add a step to Lin's algorithm by sharpening the image once using a 3×3 unsharp contrast enhancement filter, described in Section A.6. Matlab code for the smoothing procedure is provided in Program

*©1994-2006 by The MathWorks, Inc. MATLAB is a registered trademark of The MathWorks, Inc.



(a) NMS result overlaid on correlation surface. (b) NMS result overlaid on input texture.

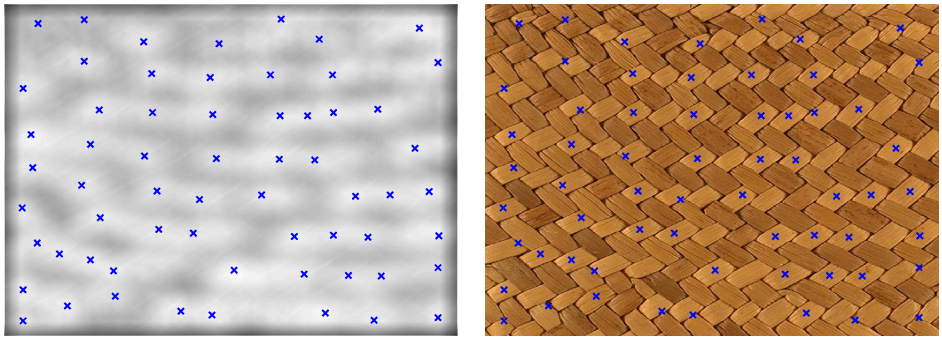
Figure 3.3: The result of non-maximal suppression applied to a correlation surface.

Listing D.4.

The correlation surface contains maxima at locations with appearance similar to the user-selected sample texton. We can extract these maxima, using non-maximal suppression, to obtain locations of potential texton centres; the locations are termed potential texton centres because the extraction process will sometimes return an incorrect location. This incorrect location is either an artifact of the correlation process or of the non-maximal suppression process; when a location in the image which is not an actual texton correlates highly with the sample texton this location may be reported as a texton centre in error. Our ad-hoc lattice construction algorithm, presented in Section 6.1, adds potential texton centres to the lattice based on the likelihood that they belong in the lattice. In most cases, the algorithm correctly ignores incorrect texton locations, but the potential for incorrect lattice extraction exists whenever incorrect texton locations are present. The graph-theoretic lattice construction algorithm, presented in Section 6.2, will include all erroneous texton locations in the output, implying that errors in the texton locating procedure are propagated to the lattice extraction result.

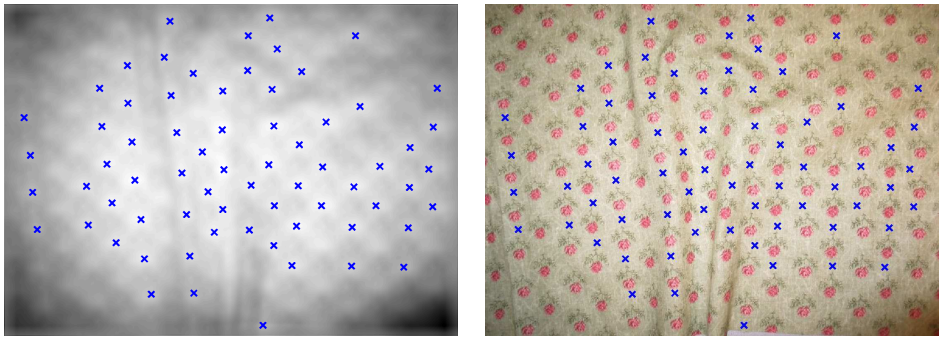
Figure 3.3(a) shows the result of applying non-maximal suppression to a smoothed correlation surface. Figure 3.3(b) shows the result of non-maximal suppression overlaid on the original texture of Figure 3.2(a). Program Listing D.5 provides the Matlab implementation.

The texton locating procedure is not without faults. In some cases, too few textons are located, too many textons are located, or textons are located in improper positions. Figures 3.4 and 3.5 depict inaccurate texton locating results due to poor correlation. In Figure 3.4, non-maximal suppression does not locate all textons in the image since the correlation surface does not contain peaks at locations which differ in colour from the user-selected sample texton. In Figure 3.5, the correlation surface contains peaks at incorrect texton locations, and at too few locations.



(a) NMS result overlaid on correlation surface. (b) NMS result overlaid on input texture.

Figure 3.4: Non-maximal suppression is not accurate all of the time.



(a) NMS result overlaid on correlation surface. (b) NMS result overlaid on input texture.

Figure 3.5: Another poor texton locating result due to inaccurate correlation.

3.2 Results

We tested our texton finding algorithm on 100 images gathered from various sources [9, 13, 33, 67]. Unfortunately, there is currently no automatic way of measuring the quality of a texton locating result. Therefore, we use human judgement as a measure of quality and compare results of our texton locating algorithm to "ground-truth" results obtained from manually locating textons.

To test our algorithm, we attempted to re-create the ground-truth results using our texton locating algorithm. We consider the texton locating algorithm successful if we are able to re-create the ground-truth result without significant errors. We classify significant errors into two categories: (1) missing textons and (2) misplaced textons. In missing texton errors, the texton locating procedure fails to identify a texton where a manually located texton exists. In misplaced texton errors, the texton locating procedure identifies textons where no manually located texton exists, or identifies multiple textons where only one manually located texton texton exists. The success of texton locating is judged with a pass-fail marking, ignoring errors for textons partially occluded by the image boundary. The reason for this pass-fail marking scheme is that it is difficult to manually evaluate the quality of the texton locating result, and with a simple pass-fail grade we are able to quickly provide an objective measure of the quality of our texton finding algorithm.

During the test, the user manually selected a texton which was used as input to our algorithm. If the result was unsatisfactory, the user was allowed to select another texton, up to a maximum of 5 times. If after 5 attempts at texton locating the algorithm fails to properly select textons, we consider the case a failure. Using this criteria, we tested our algorithm on 100 near-regular textures. In total, the ground-truth result can be successfully obtained in 71 cases using our texton locating process. The results of this test are summarized in Figure 3.6.

We classified the 29 failed texton locating cases using the two error categories previously discussed: (1) missing textons and (2) misplaced textons. In total, 27 of the failed texton locating results contain missing textons, while 13 contain misplaced textons. 11 cases contain both missing textons and misplaced textons. Full results of these classifications are given in Table E.1. This classification of errors has provided motivation for developing methods of correcting or accounting for errors in the final lattice construction result. The ad-hoc lattice construction method, described in Section 6.1, ignores misplaced textons, but fails for missing texton errors, which account for the majority of texton locating errors. Due to this short coming, we developed a method of correcting missing texton errors, which we describe in Chapter 5.

The algorithm that we have developed is sensitive to the initial choice of texton. To test the robustness of our algorithm to alterations in initial texton choice, we selected five textons from 25 images, and compared each result to the ground-truth result. The textons were chosen based on their suitability as an "average" texton in the image. The results of this test show that sensitivity to

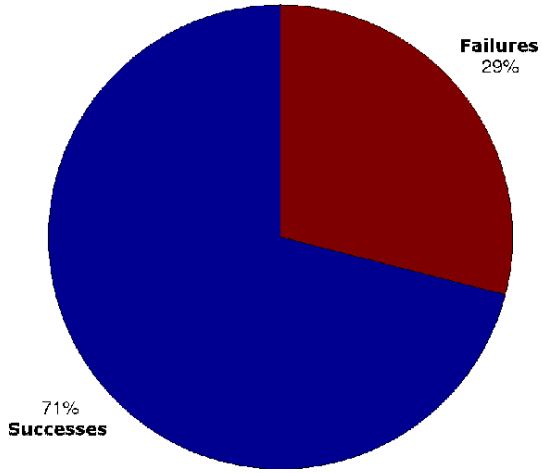


Figure 3.6: Our procedure correctly extracts textons in a total of 71 out of 100 images.

Table 3.1: Success rates for each texton choice.

Texton Choice Number	Success Rate on Texton Choice Number
1	76%
2	60%
3	76%
4	68%
5	68%

Table 3.2: Cumulative success rates after each texton choice.

Texton Choice Number	Cumulative Success Rate After Texton Choice Number
1	76%
2	80%
3	92%
4	96%
5	100%

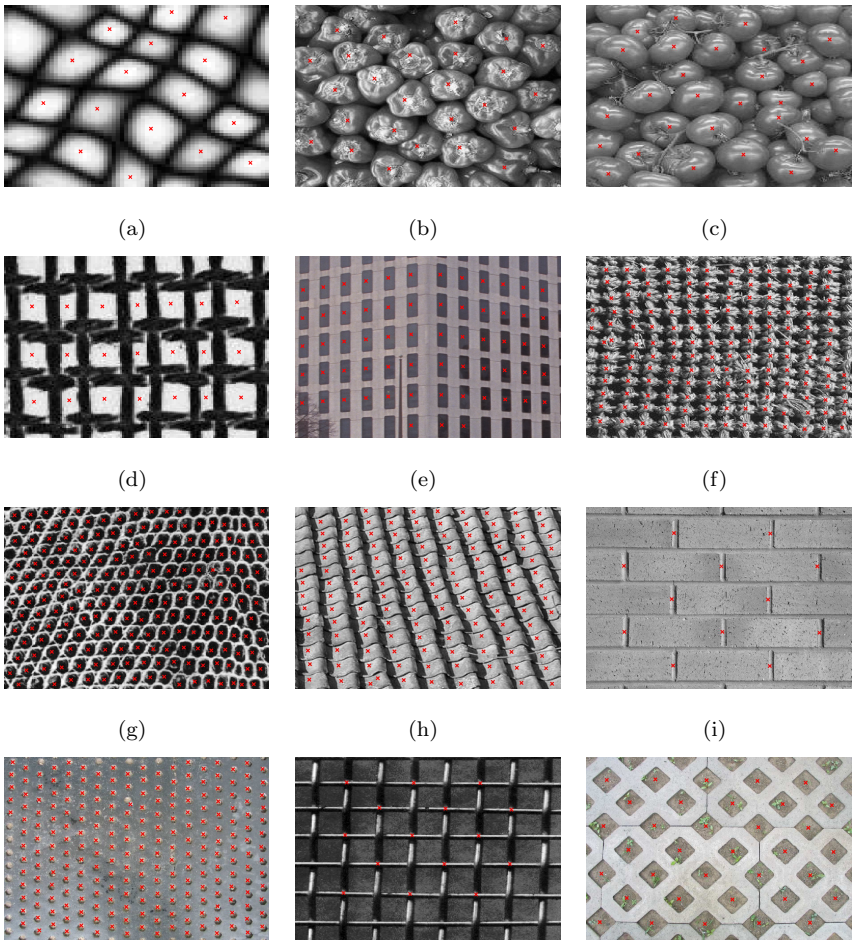
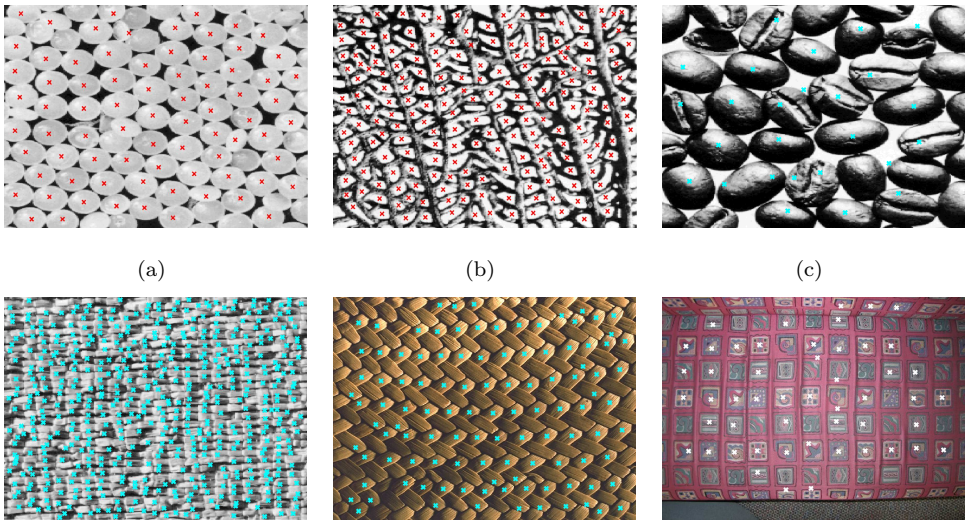


Figure 3.7: Texton locating successes. Figure 3.7(a) is from the database of DeBonet. Figures 3.7(i), 3.7(j), 3.7(l), and 3.7(h) are from the CMU near-regular texture database [13]. Figures 3.7(d), 3.7(k), and 3.7(g) are from the Brodatz texture album [9]. Figures 3.7(f) and 3.7(e) are from the VisTex texture database [33]. Figures 3.7(b) and 3.7(c) are from the database of Simoncelli.



(a) (b) (c)

(d) (e) (f)

Figure 3.8: Texton locating failures. Figures 3.8(a) to 3.8(b) are from the Brodatz texture album [9]. Figure 3.8(e) is from the VisTex texture database [33]. Figure 3.8(f) is from the CMU near-regular texture database [13].

texton choice is largely dependent on the input texture. In some cases the choice of texton does not affect the texton locating result, while in other cases the choice of texton greatly affects the texton locating result. In total, we have found that, on average, the texton locating result is correct 3.48 times out of 5 initial texton choices. Using this data, we also determined the number of successful texton locating results after each texton selection. The number of successful texton locating results after the first, second, third, fourth, and fifth texton selection is 19, 15, 19, 17, and 17 respectively. In these tests, the average number of successful texton locating results for each of the five texton choices is 17.4. This data is summarized in Table 3.1. We also determined the cumulative success rate of texton locating after each texton choice. Each entry of Table 3.2 reports the total number of successful texton locating results after each texton selection, remembering previously successful results. Full results are given in Table E.2.

Figure 3.7 depicts successful texton locating results. Figures 3.7(a) to 3.7(c) show successful texton locating results in images with varying texton appearance and geometry. Figures 3.7(d) to 3.7(h) depict successful results in images with varying texton geometry, but similar appearance. Finally, Figures 3.7(i) to 3.7(l) depict successful texton extractions from images with varying texton appearance, but similar geometry.

Figure 3.8 depicts unsuccessful texton locating results. Figures 3.8(a) and 3.8(b) show two near-misses. The majority of textons in these two images are found properly, but some noticeable errors exist. Figure 3.8(c) is a failure case. This failure is due to the texture being irregular. Figure

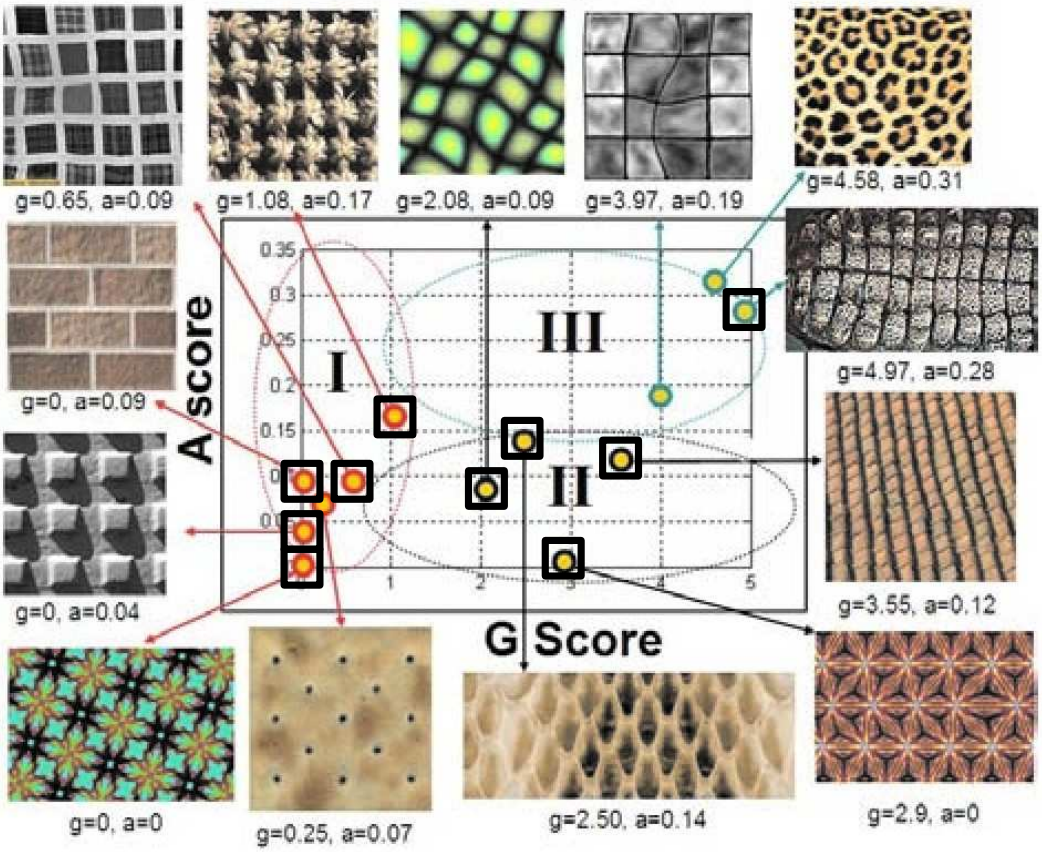


Figure 3.9: Results of texton extraction for textures with known G and A scores. Squares around circles indicate successful texton extraction. Image Source: Liu et al. [45].

3.8(d) fails due to high frequency components in the image. The algorithm we have developed has difficulty in such cases. Finally, Figures 3.8(e) and 3.8(f) fail due to wide variations in texton colour.

The content of this thesis has been inspired by the work of Liu et al. [45], who provide a near-regular texture synthesis algorithm described in the Literature Review of Chapter 2. Further results from the research provide a method of measuring regularity along two axes: geometric structure (G) and colour appearance (A). After manually extracting a lattice from a texture, G and A scores are computed by comparing the contents of each texton of the lattice. Depending on the results of these measurements, the textures were classified as one of Type I, II, or III. Type I texture implies regular texton geometry with irregular texton appearance, Type II texture implies irregular texton geometry with regular texton appearance, and Type III texture implies irregular texton geometry and appearance.

To validate the effectiveness of our texton extraction method with textures of varying regularity

we obtained a set of 13 images with known G and A scores, shown in Figure 3.9, and applied our texton locating method to each image. Currently this is the largest set of available images with G and A scores. We then noted the textures where our algorithm successfully locates textons. The results of our validation are also shown in Figure 3.9, where successful texton locating results are outlined with a black box. In total, our algorithm resulted in 2 out of 3 Type III textures having incorrect texton locations, though the leopard skin sample could be viewed as an irregular texture, and the remaining failure case contains a highly irregular patch in the centre of the image. The only other failure case occurs with the Type I soda cracker texture. In this case, the majority of textons are similar in appearance, leading to a low A score, and textons with drastically varying appearance, i.e. dark patches in the cracker, do not get properly extracted. Unfortunately, no strong trends emerge from the data, due mostly to the small data set and the small size of the texture samples within the data set.

We can see from these tests that in some cases our algorithm is able to successfully locate textons from texture with high G scores. The results of Liu et al. [45] have difficulty synthesizing irregular textures with a high G score. In this thesis, we suggest alternate uses for the lattice extracted from irregular texture. These include texture replacement [41], grouping of repeated scene elements [36], object recognition, shape from texture, texture classification, texture compression, and texture blending.

Our texton locating procedure is robust and accurate for the majority of texton locating cases [62]. However, we have also identified two error cases where our texton locating procedure fails. As a means of overcoming both errors, we introduce a potentially more accurate correlation method using feature spaces in Chapter 4. We also provide methods of overcoming each error case individually. For the case of missing textons, we have developed a procedure that uses frequency-space analysis to introduce unreported texton locations into the texton locating result. This procedure is described in Chapter 5. For the case of misplaced textons, we have developed an ad-hoc lattice construction method that correctly ignores erroneous textons. We discuss this procedure in Chapter 6.

Chapter 4

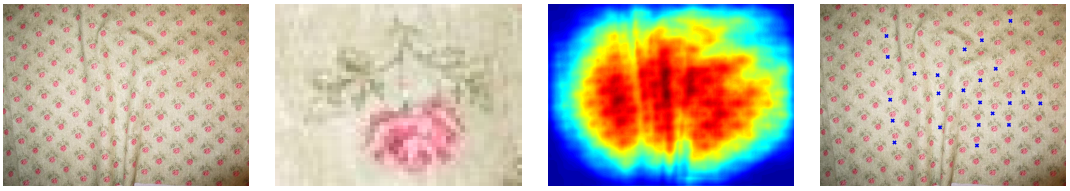
Improving Texton Locating Using Image Features

The texton locating procedure described in Chapter 3 can be used to create a perceptually sensible lattice for near-regular texture [62]. However, the range of texture that allows successful texton locating is limited using this procedure. In some cases, textons within a texture do not correlate with the colour of a chosen texton sample. In Section 4.1, we discuss some common image features and how they may be applied to the problem of texton locating. In Section 4.2, we present test results of the use of image features for texton locating.

4.1 Correlation Within Feature Space

As an example, Figure 4.1 depicts a failed texton extraction. The texton in Figure 4.1(b) is correlated with Figure 4.1(a) to produce the surface in Figure 4.1(c). The correlation surface in Figure 4.1(c) is noisy, peaks in the surface are not visually identifiable, and non-maximal suppression fails to properly extract texton locations, resulting in Figure 4.1(d). Note that the correlation surfaces in this section are displayed using a cool to warm colour map, with blue representing low values and red representing high values. Figure 4.2 provides the mapping from pixel intensity to colour used for the images that follow.

We can improve the texton locating result of Figure 4.1 by first computing the image's colour gradient magnitude and correlating this gradient image with the gradient magnitude of a texton sample. Figure 4.3 shows the result of correlation using the magnitude of the image's gradient. This



(a) A texture. (b) A texton sample from (a). (c) The result of (b) correlated with (a). (d) Non-maximal suppression of (c) overlaid on (a).

Figure 4.1: Failed texton extraction using RGB colour space as feature space.

Figure 4.2: The colour bar used in the images of this section. Pixel intensity is mapped to a colour between light blue and dark red.

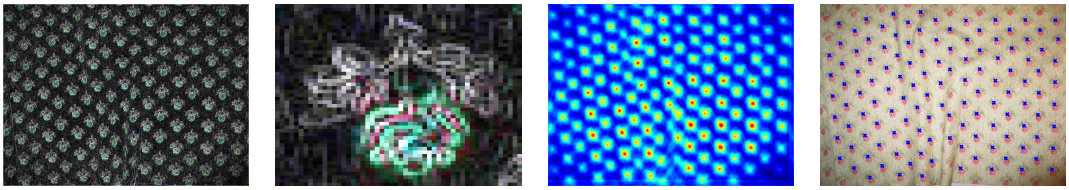
example shows how correlation with image features may improve the result of the texton locating process.

An *image feature* is a characteristic of an image calculated using image data. We call the *feature space* for an image the set of image features used during correlation. In feature space, each pixel within the image is characterized by a vector, where each vector component measures a particular feature value. Correlation within feature space operates as expected; correlation is applied independently to each component of the feature vector at each pixel. For example, correlating within the red, green, blue (RGB) colour space, as in Figure 4.1, implies a 3-dimensional feature space, where each pixel in the image is specified as a vector in feature space.

A careful selection of the feature space for a texture may improve the results of texton locating considerably. However, it is difficult to know a priori what features may improve a result; feature selection is a large open problem. Instead of performing an exhaustive study of the performance of image features for texton locating, we tested four simple feature spaces along with several feature spaces derived from alternate colour spaces. We report these results here. For technical information on each of the feature spaces discussed, consult Appendix A.

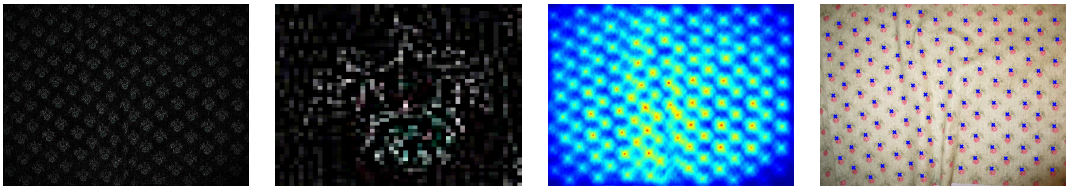
We have already seen, in Figures 4.1 and 4.3, how using an image's gradient magnitude as feature space can improve the results of texton locating. In addition to the gradient magnitude, the second derivative of an image, computed using a Laplacian-of-Gaussian filter, provides another feature space. Typically, the second derivative of the image is noisy and the feature space produced by the second derivative does not allow us to easily locate textons. However, in some cases the second derivative feature space can provide good texton locating results, as depicted in Figure 4.4. In general, features derived from changes in pixel intensity provide a good feature space for locating textons. These features include the gradient magnitude and second derivative, which we have already discusses, as well as image edges, which are discussed below.

Edge information provides another example of the improvements gained through correlation in feature space. The Sobel filter can be used to detect strong image edges; given the approximation of the first derivative, edges are detected as local maxima. Local maxima are found using a threshold determined by Otsu's method [55]. We have obtained favourable results using the Sobel edge detection feature space. The Sobel edge detection process only detects strong edges, and as a result only prominent features of textons are detected. Typically, the surface produced by the Sobel edge detector is sparsely populated and devoid of noise, allowing non-maximal suppression to detect



(a) A gradient magnitude image. (b) A texton sample from (a). (c) The result of (b) correlated with (a). (d) Non-maximal suppression of (c) overlaid on (a).

Figure 4.3: Improved texton extraction using image gradient magnitude as feature space.



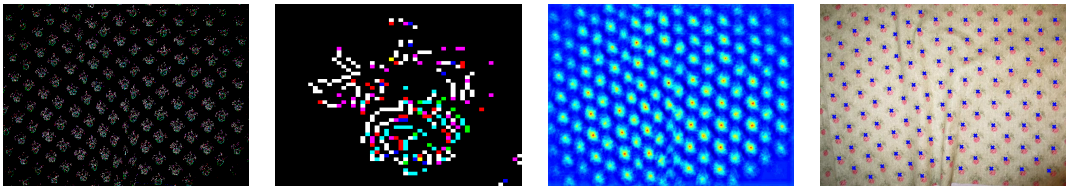
(a) A Laplacian-of-Gaussian image. (b) A texton sample from (a). (c) The result of (b) correlated with (a). (d) Non-maximal suppression of (c) overlaid on (a).

Figure 4.4: Improved texton extraction using a Laplacian-of-Gaussian feature space.

maxima in the correlation surface. Figure 4.5 depicts the result of locating textons using features detected by the Sobel edge detection filter.

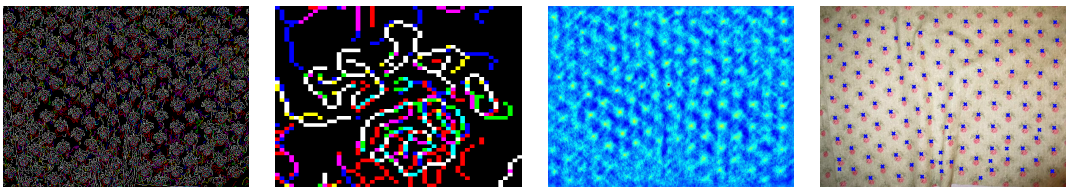
Another method for calculating image edges is the Canny edge detector [10]. The method finds edges by looking for local maxima of the image's gradient magnitude. The strength of the method lies in its use of two thresholds for detecting strong and weak edges. Weak edges are included in the output only if they are connected to strong edges, making the edge detector less likely to produce errors due to noise. We use this method as an alternative to Sobel edge detection, which doesn't detect weak edges. Figure 4.6 depicts the result of locating textons using features detected by the Canny edge detection filter.

The features discussed up to this point have been derived from gray scale information or from RGB colour space. However, it is known that distances in RGB colour space do not accurately map to perceptual distance [69]. As an alternative, we tested the texton locating procedure using the hue, saturation, value (HSV) colour space, which mimics more accurately the method in which humans perceive colour. For our testing, we created four feature spaces from the HSV colour space. The first feature space is the full HSV colour space. The remaining three are the individual components of the HSV colour space: hue, saturation, and value. A failed result of our texton location algorithm using RGB feature space is shown in Figure 4.7. In comparison, Figures 4.8 to



(a) A Sobel edge detection image. (b) A texton sample from (a). (c) The result of (b) correlated with (a). (d) Non-maximal suppression of (c) overlaid on (a).

Figure 4.5: Improved texton extraction using Sobel edge detection as feature space.

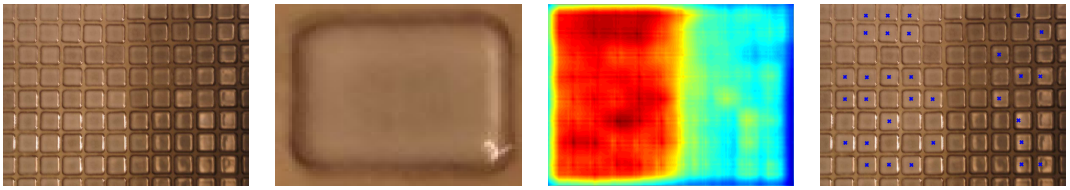


(a) A Canny edge detection image. (b) A texton sample from (a). (c) The result of (b) correlated with (a). (d) Non-maximal suppression of (c) overlaid on (a).

Figure 4.6: Improved texton extraction using Canny edge detection as feature space.

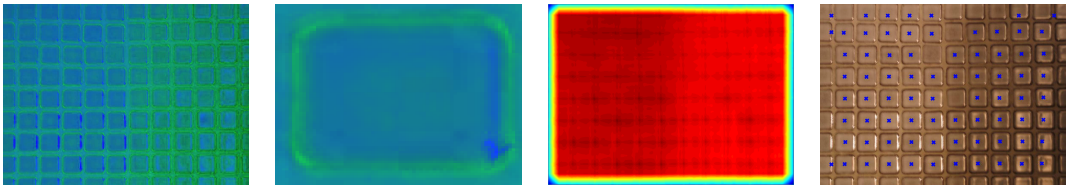
4.11 demonstrate the improvements obtained using portions of the HSV colour space. Figure 4.8 depicts a result using HSV feature space, Figure 4.9 depicts a result using hue feature space, Figure 4.10 depicts a result using saturation feature space, and Figure 4.11 depicts a result using value feature space.

Finally, we followed the suggestions of Hertzmann et al. [27] and tested our texton locating procedure using luminance feature space. Hertzmann et al. suggest that a feature space based on luminance alone offers improvements over RGB feature space, with a claim that the human eye is more sensitive to changes in luminance than changes in colour. To compute luminance, they use



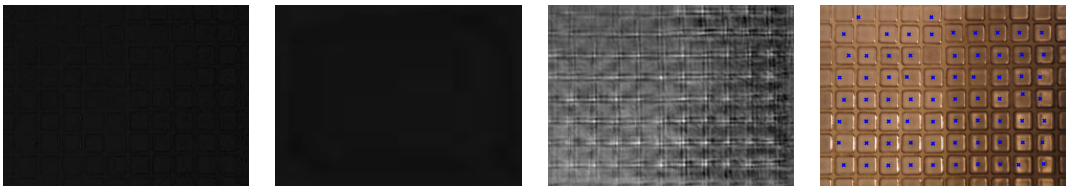
(a) A texture. (b) A texton sample from (a). (c) The result of (b) correlated with (a). (d) Non-maximal suppression of (c) overlaid on (a).

Figure 4.7: Failed texton extraction using RGB colour space as feature space.



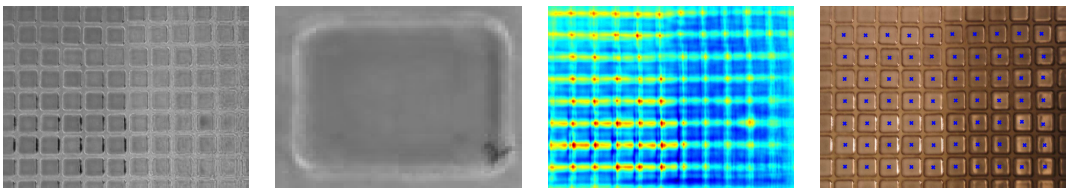
(a) An HSV image. (b) A texton sample from (a). (c) The result of (b) correlated with (a). (d) Non-maximal suppression of (c) overlaid on (a).

Figure 4.8: Texton extraction using HSV colour space as feature space.



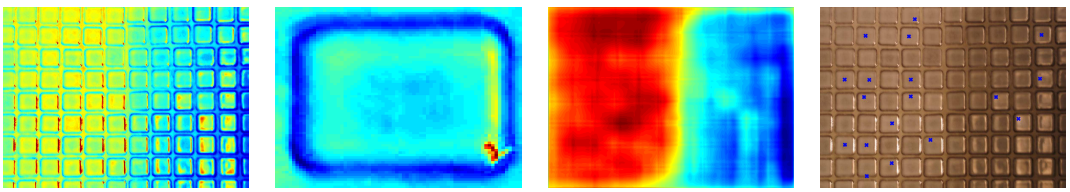
(a) A hue image. (b) A texton sample from (a). (c) The result of (b) correlated with (a). (d) Non-maximal suppression of (c) overlaid on (a).

Figure 4.9: Texton extraction using colour hue as feature space.



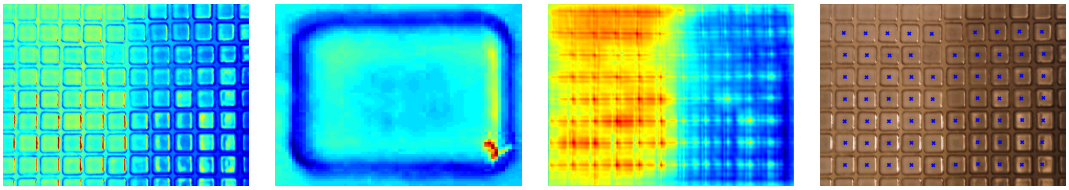
(a) A saturation image. (b) A texton sample from (a). (c) The result of (b) correlated with (a). (d) Non-maximal suppression of (c) overlaid on (a).

Figure 4.10: Texton extraction using colour saturation as feature space.



(a) A value image. (b) A texton sample from (a). (c) The result of (b) correlated with (a). (d) Non-maximal suppression of (c) overlaid on (a).

Figure 4.11: Texton extraction using value as feature space.



(a) A luminance image. (b) A texton sample from (a). (c) The result of (b) correlated with (a). (d) Non-maximal suppression of (c) overlaid on (a).

Figure 4.12: Texton extraction using colour luminance as feature space.

the Y channel from the YIQ colour space. Figure 4.12 depicts the result of using luminance as feature space.

The consideration of feature space has improved our texton locating procedure. However, due to its vast literature a complete investigation of feature space is outside the scope of this thesis. We therefore defer further exploration of this topic to future work. A particularly interesting avenue of future work lies in using sophisticated edge detection and feature selection methods to obtain a feature space that consistently improves failed texton locating results. Wu and Yu [71] use an edge detection approach to improve texture synthesis for structured texture. Another avenue of interest lies in using features computed on multiple scales to improve a texton locating result. The steerable pyramid [61] offers a structure useful for computing features on multiple scales, and several researchers have used multiple scale analysis to improve texture synthesis results [7, 26, 27].

4.2 Results

To determine if any of the features discussed in this chapter can consistently locate the correct textons, we performed some additional tests. We use the same measure of success that was used when testing the texton locating procedure of Chapter 3. That is, the texton locating algorithm is successful if no significant errors exist, where significant errors are classified into the two categories of (1) missing textons and (2) misplaced textons.

We first ran the lattice extraction algorithm on 50 images using four feature spaces, starting with the same initial texton selection. The feature spaces were the gradient magnitude obtained using a Sobel filter, the second derivative approximated using the Laplacian-of-Gaussian filter, the Sobel edge detection method, and the Canny edge detection method. We then compared the texton locating results of correlation using each feature space to the result obtained using RGB feature space. We attempted to determine whether using any of these four feature spaces leads to consistently accurate texton locating results. For each of the four feature spaces, we totalled the number of successful texton locating results observed and derived a success rate by dividing this

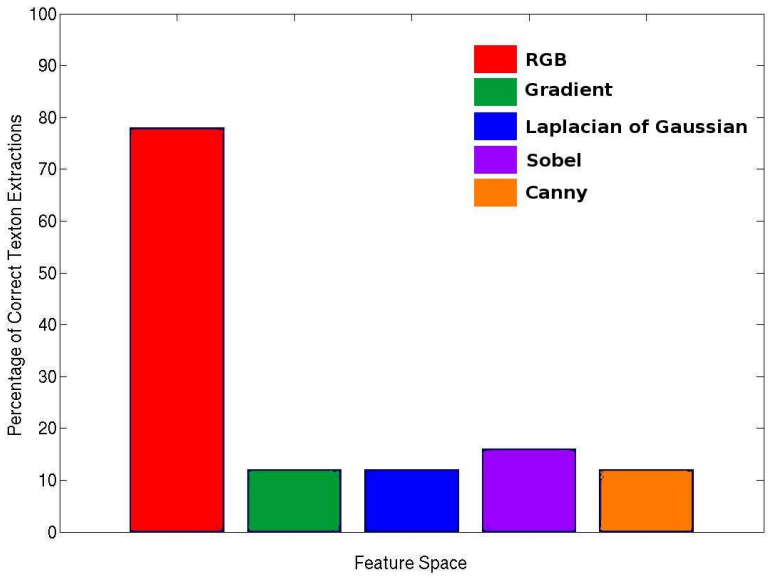


Figure 4.13: Common image features generally do not improve upon RGB colour space.

total by the number of images tested.

The results are decidedly in favour of using the RGB feature space, with a success rate of 78%, for the majority of lattice extraction problems. The gradient magnitude, derivative, and Canny edge detection feature spaces had texton extraction success rates of 12%, while Sobel edge detection feature space had a texton extraction success rate of 16%. Figure 4.13 shows a bar graph of these results. Full test results are given in Tables E.3 and E.4.

We ran a similar test using feature spaces derived from colour. The feature spaces consist of image hue, saturation, value, and luminance. We tested these features on a a new test set of 50 colour images with a texton extraction success rate of 64% using semi-automatic extraction in RGB colour space. Again, the test results are in favour of using the RGB feature space for the majority of texton extraction problems. The hue, saturation, and value feature spaces correctly extracted textons from the image 34%, 36%, and 28% of the time, respectively. The full HSV feature space correctly extracted textons from the image in 38% of cases. Finally, the luminance feature space correctly extracted textons from the image in 50% of cases, lending support to Hertzmann et al.'s [27] claim that luminance is an accurate indicator of visual similarity, though we found that it is not as accurate as RGB colour space for locating textons. Figure 4.14 shows a bar graph of these results. Full test results are given in Tables E.5 and E.6.

Finally, we tested the 29 failure cases from the 100 test textures cited in Section 3.2. From the 29 failure cases, we successfully located textons in 14 cases, approximately 50% of previous failures.

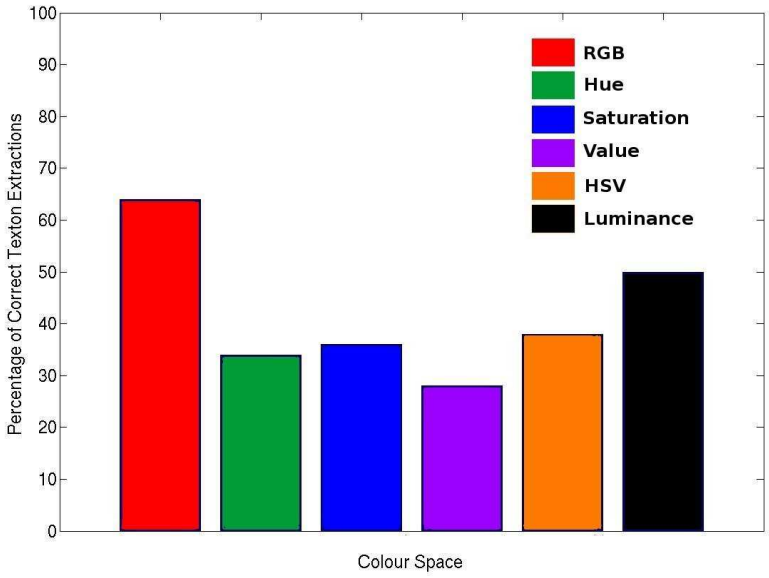


Figure 4.14: Changing the colour space generally does not improve upon texton locating.

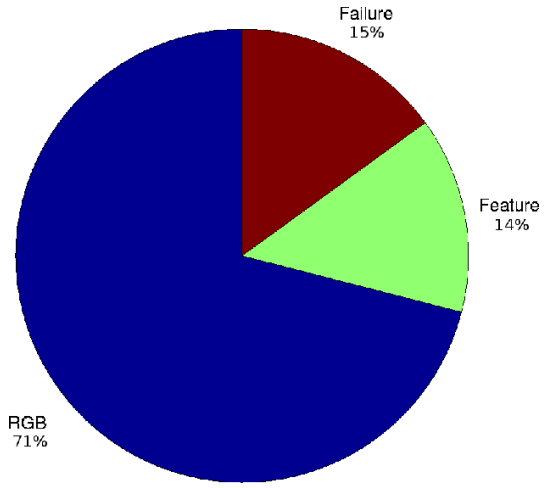


Figure 4.15: Using image features we successfully locate textons in 85% of cases.

This results in a total of 85 successful texton locating results out of 100 test images when using the simple image features presented in this chapter. Figure 4.15 graphically depicts these results.

Chapter 5

Correcting Localized Texton Locating Errors

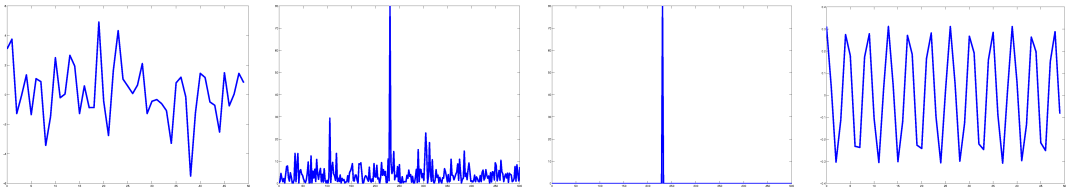
In Chapter 3 we showed how to locate textons in an image, and in the previous section we showed how to improve the texton locating procedure using correlation with feature space. In this section, we use frequency-based repair to correct errors in the texton locating procedure. The feature-based correlation and the frequency-based improvements can be used in sequence or in isolation, depending on the situation. We have found that locating textons in difficult cases requires first choosing an appropriate feature space for the particular texture, which reduces errors during texton locating. This is followed by frequency-based repair, which can correct any remaining errors. In Section 5.1, we discuss the frequency-based repair method, and in Section 5.2, we discuss the results of correcting localized errors using frequency-based repair.

5.1 Frequency-Based Repair

In the discussion that follows we use the Fourier and inverse Fourier transforms. For an introduction to these transforms, refer to Appendix C.

To reduce noise in a function, we can analyze the function's power spectrum; applying the inverse Fourier transform to only the dominant peaks in a function's power spectrum eliminates random noise from the function. With this approach, the contributions from non-dominant frequencies in the power spectrum are not mapped to the time domain, and therefore the resulting signal will contain contributions only from the dominant frequencies of the original function.

Using this knowledge, a typical method for eliminating noise from a function is by thresholding the function's power spectrum, and then applying an inverse Fourier transform to the result. Thresholding removes frequencies below a certain power, isolating dominant frequencies from the power spectrum. For example, consider Figure 5.1. Figure 5.1(a) depicts a sinusoidal function with random noise, and Figure 5.1(b) depicts the power spectrum of the function in Figure 5.1(a). In Figure 5.1(c), a threshold has been applied to Figure 5.1(b). The threshold eliminates frequencies from the power spectrum which contribute an insignificant power to the function. Figure 5.1(d) depicts the inverse Fourier transform of this thresholded power spectrum, which captures the dominant frequency of the function in Figure 5.1(a).



(a) A function with random noise.

(b) The function's power spectrum.

(c) The thresholded power spectrum.

(d) The inverse Fourier transform of the thresholded power spectrum.

Figure 5.1: The Fourier transform projects a function varying over time to a function varying over frequency. Applying a threshold to the frequency representation and inverse Fourier transforming the result removes unwanted frequencies from the image.

The principles discussed above also apply to functions in two dimensions using the following equations of the Fourier and inverse Fourier transforms:

$$\begin{aligned} F(u,v) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x,y)\; e^{-i2\pi(ux+vy)}\; dx\; dy, \\ f(x,y) &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u,v)\; e^{i2\pi(ux+vy)}\; du\; dv. \end{aligned}$$

An image can be represented as a discretization of a two-dimensional function. Furthermore, the Fourier transform can be applied to any physically realizable function [8] and thus a power spectrum exists for any image. However, since an image is a discrete entity, we must apply the discrete Fourier transform rather than the continuous Fourier transform. The discrete Fourier transform applies the same principles as the continuous Fourier transform, but for discrete functions. For an image of size $M \times N$ the discrete Fourier transform in two dimensions is given by

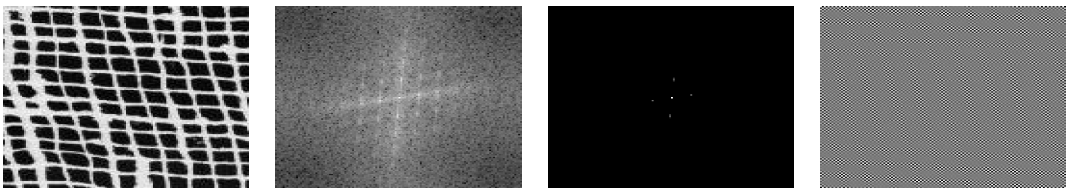
$$F(u,v) = \frac{1}{MN}\sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)\; e^{-i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)},$$

and the discrete inverse Fourier transform in two dimensions is given by

$$f(x,y) = \sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)\; e^{i2\pi\left(\frac{ux}{M}+\frac{vy}{N}\right)}.$$

For a two-dimensional discrete function, the Fourier transform is represented using complex numbers arranged on a grid. Each discrete location in the image's power spectrum encodes the power in the image attributed to a specific frequency; the specific frequency is provided by the (x, y) location in the grid, and the power is provided by the value stored at the (x, y) location. To illustrate this, consider the image in Figure 5.2(a). It is represented using a power spectrum in Figure 5.2(b).

If we extract the dominant frequencies of texton placement from the power spectrum of an image and then recreate the image using only these frequencies, maxima in the recreated image would



(a) A texture.

(b) The texture's power spectrum.

(c) A threshold applied to the power spectrum.

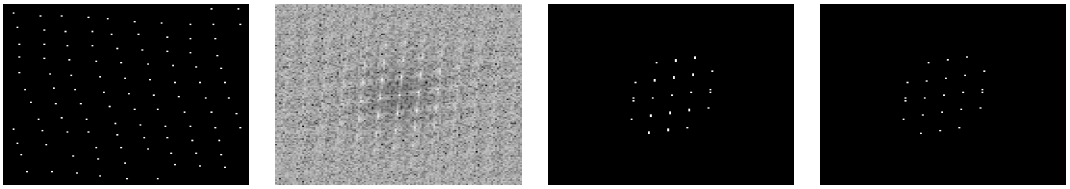
(d) Inverse Fourier transform of the thresholded power spectrum.

Figure 5.2: An image's power spectrum can be thresholded, but this does not allow us to locate textons.

likely correspond to texton locations, much as removing noise from the one-dimensional function in Figure 5.1 allowed us to isolate the function's dominant frequency and remove random noise. Unfortunately, using a threshold to isolate the dominant frequencies in an image's power spectrum does not provide a solution to our problem; it does not allow us to locate textons because in texture many frequencies interact in concert to create an image. Isolating only the dominant frequencies does not give an accurate representation of the frequency of texton placement. As an example, we can take the power spectrum of Figure 5.2(b), and apply a threshold, resulting in Figure 5.2(c). Applying the inverse Fourier transform to this thresholded power spectrum results in Figure 5.2(d), whose maxima are difficult to extract and do not accurately represent the locations of textons in the original texture of Figure 5.2(a), and our texton locating method fails to accurately locate textons in this case.

Fortunately, if a rough estimate of texton locations is provided, rather than the original image, we can use the frequency of texton placement in the rough estimate as a means of determining the dominant texton placement frequencies. We obtained a rough estimate of texton locations using the correlation procedure described in Chapter 3 and the improvements described in Chapter 4. The output of the texton locating algorithm is a binary image encoding texton locations. The discrete Fourier transform calculated on this binary image is able to isolate the dominant frequencies that determine texton placement. Figure 5.3(a) depicts the output of the texton finding procedure for the texture of Figure 5.2(a), and Figure 5.3(b) depicts the power spectrum of Figure 5.3(a). The power spectrum of the texton location binary image contains distinct peaks at frequencies responsible for texton placement. If we isolate the proper peaks in this spectrum we can remove any noise from the original texton location data, offering an improved estimate of texton locations.

We offer a sophisticated thresholding algorithm that isolates the dominant frequencies of texton placement from texton location data. As a pre-processing step, we apply a binary threshold to the power spectrum. This is depicted in Figure 5.3(b). We set the threshold to one third of the average power in the image, given by the DC component in the centre of the power spectrum. We have



(a) A texton locating result.

(b) The power spectrum of (a).

(c) A threshold applied to the power spectrum.

(d) Morphological shrinking of (c).

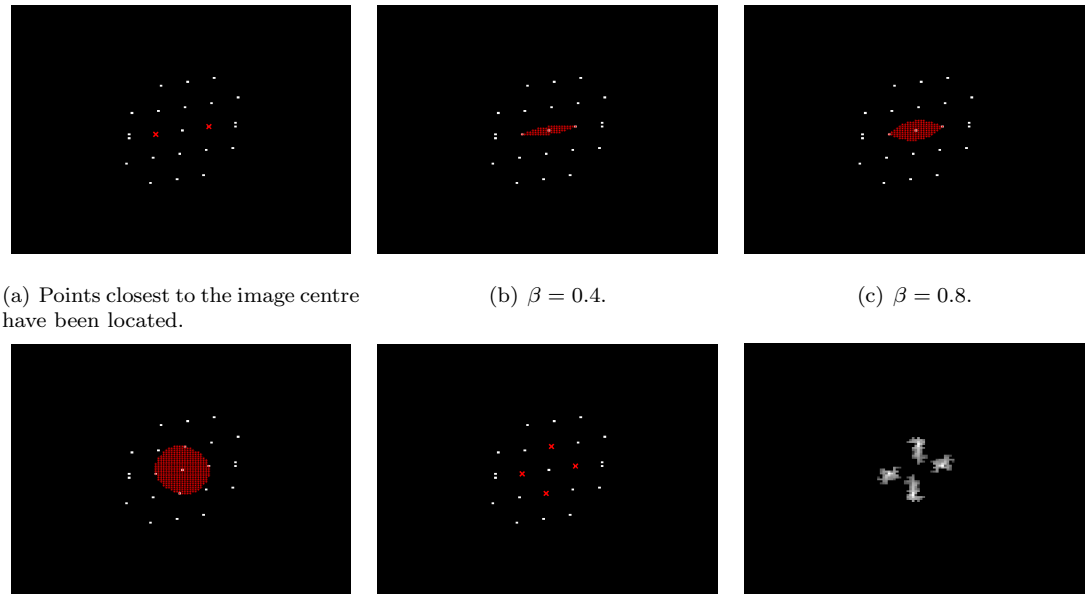
Figure 5.3: Thresholding the texton location binary image allows us to correctly extract the frequencies responsible for texton placement.

obtained positive results with this threshold, and did not need to change it for any image samples. However, it is conceivable that some texture samples would require a different threshold and so we set it as an optional parameter of the algorithm. Figure 5.3(c) depicts the binary threshold of the power spectrum of Figure 5.3(b). To locate exact points within the thresholded image, morphological shrinking is applied, which removes pixels from the boundary of a region, ultimately reducing each region in the image to a single pixel. This result is depicted in Figure 5.3(d).

To isolate the dominant frequencies from the thresholded and morphologically shrunk power spectrum, we make some assumptions. First, from the definition of near-regular texture [45] we assume that texton placement can be approximated using two translation vectors [24], implying that texton placement is determined by two dominant frequencies. Second, we assume that the binary image specifying texton locations contains few errors. Unfortunately, we have not been able to obtain an analytical rule stating the tolerated amount of error; the acceptable quantity allowed depends on each particular texture.

The locations in the power spectrum closest to the image centre correspond to the lowest frequencies in the image. Thus, the frequency located at the peak closest to the power spectrum's centre determines the lowest frequency necessary for texton placement. Figure 5.4(a) highlights this peak and its symmetric partner (the power spectrum is symmetric about the origin [8]). These peaks specify one frequency that governs texton placement. Through the definition of near-regular texture, we assume that the two peaks determining the second frequency that governs texton placement are at locations approximately orthogonal to the line between the two peaks determining the first frequency governing texton placement. We find these next two peaks by searching within the β region of influence defined by the closest peak and its symmetric pair.

The success of the β region of influence in constructing a lattice from texton location data lends support to the use of the same structure in determining the frequency of texton placement. In fact, peaks in the power spectrum of the texton locating data are placed in arrangements similar to the local arrangements found in texton location data, especially when assuming that texton placement



(a) Points closest to the image centre have been located.

(b) $\beta = 0.4$.

(c) $\beta = 0.8$.

(d) $\beta = 1.3$.

(e) Four peaks defining texton placement have been found.

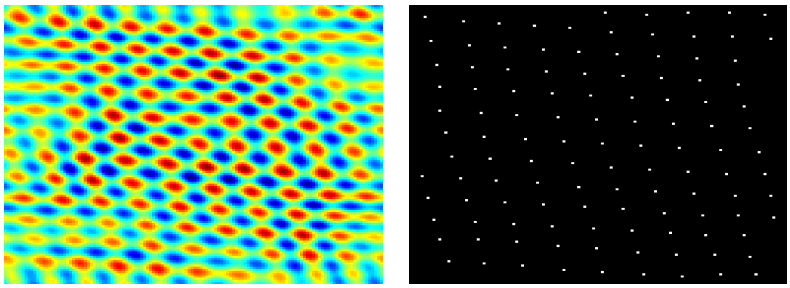
(f) Gradient descent creates a region around each peak.

Figure 5.4: Locating the translation vectors of texton placement.

is approximated by two translation vectors.

We thus search for points within the β region of influence determined by the peak closest to the image centre and its symmetric pair. β is initially set to 0, and iteratively increased until a peak lies within the lune defined by β. At each iteration β is increased by 0.1, which has provided good results for our tests. Figure 5.4 depicts several iterations of the iterative procedure. Once a peak is found within the β region of influence, that peak and its symmetric partner are deemed the next nearest peaks to the image centre and the four peaks necessary in extracting the translation vectors which determine texton placement have been found. Figure 5.4(e) highlights all four peaks discovered using this procedure.

However, the peaks that have been found each represent single frequencies, and texton placement is often defined by multiple frequencies. We account for this by expanding the region around each peak found using a gradient search, performed within the power spectrum, to include multiple frequencies in the final output. The gradient search is straight-forward. First, we account for errors in the morphological shrinking process by performing a gradient ascent at each of the four found peaks. If a neighbour of one of the peaks is larger than the peak itself, we move the peak to the larger neighbour, pushing each peak to a local maxima. From these local maxima we create regions encompassing a selection of frequencies by marking the points visited during a gradient descent. The marked regions form an image mask that selects the proper frequencies from the power spectrum (Figure 5.4(f)).



(a) Inverse Fourier transform of Figure 5.4(f)

(b) Non-maximal suppression applied to the inverse Fourier transform.

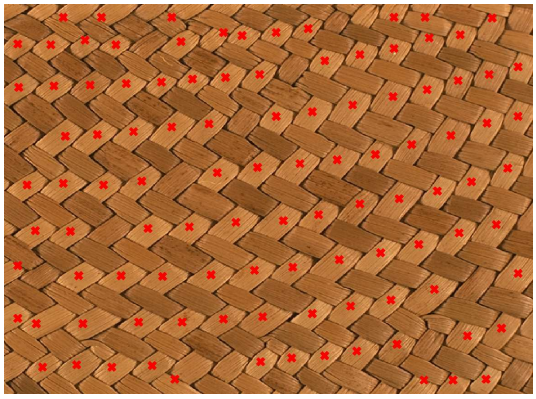
Figure 5.5: Inverse Fourier transformation of the frequencies responsible for texton placement followed by extraction of maxima.

The gradient descent procedure isolates the frequencies responsible for texton placement from the power spectrum. To determine an estimate of the actual locations of textons, these frequencies are transformed to the spatial domain using the inverse Fourier transform, resulting in Figure 5.5(a). Non-maximal suppression locates peaks in this new surface, and provides an estimate of texton locations in the original image. The final result is depicted in Figure 5.5(b).

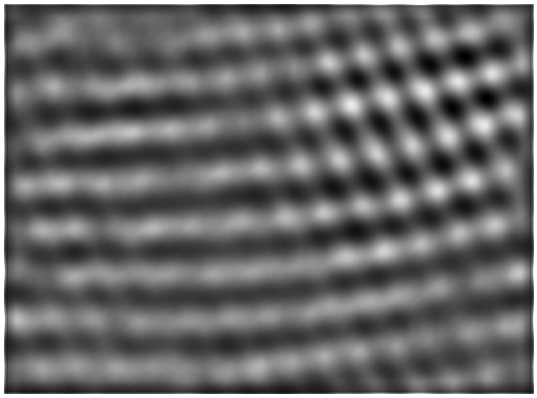
Maxima in the surface of Figure 5.5(a) offer an alternate vision of texton locations. There is now a problem with choosing between this vision and the one already obtained through correlation. To obtain improved texton location data, we can combine these two visions using multiplication or addition, producing a new surface by applying the mathematical operations to the original correlation surface and the surface obtained through inverse Fourier transforming the proper frequencies of the power spectrum. Non-maximal suppression is then applied to the new surface to obtain a repaired texton locating result.

Figure 5.6 depicts the result of using multiplication and addition to repair a texton locating result. Viewing the surfaces derived from each repair procedure, it is clear that for this texture multiplication results in prominent peaks, allowing non-maximal suppression to easily locate textons. However, multiplication does not take into account much information about the original correlation surface and may introduce errors into the texton locating result. We have found during testing that multiplication introduces errors into the texton locating result more often than addition. On the other hand, addition often does often not introduce textons at improper locations. However, addition may not include new textons where they may be needed because of the similarity of the new surface to the original correlation surface. From this discussion, it is evident that both methods of repair fail because they do not take into account the data already present from the original texton locating procedure. Thus, we have developed a procedure that takes this data into account to obtain an improved estimate of texton locations. We call this procedure *local examination*.

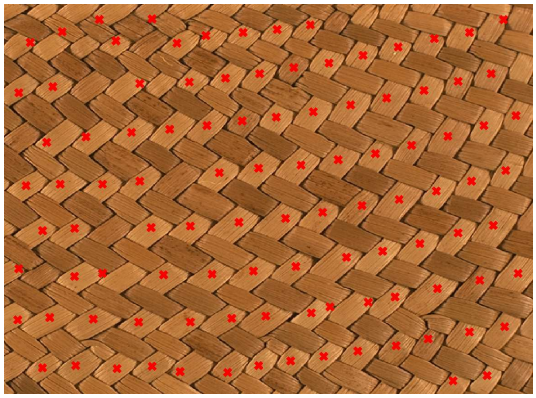
(a) Original correlation surface.



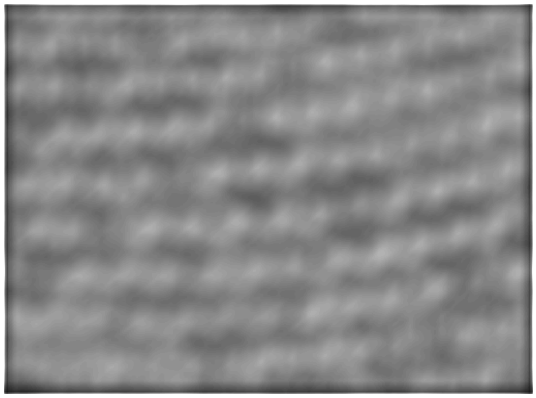
(b) Result of non-maximal suppression applied to the original correlation surface.



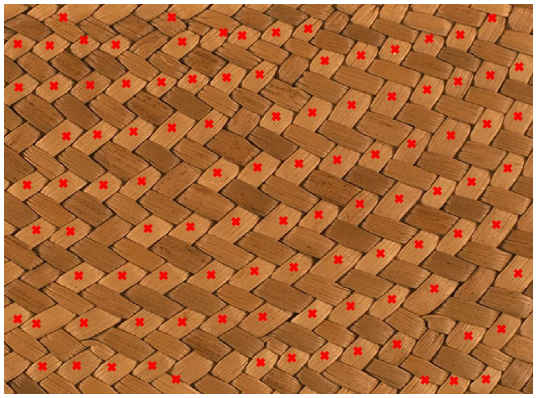
(c) Multiplication of original and inverse Fourier transformed surfaces.



(d) Result of non-maximal suppression applied to the multiplication of surfaces.



(e) Addition of original and inverse Fourier transformed surfaces.



(f) Result of non-maximal suppression applied to the addition of surfaces.

Figure 5.6: The results of frequency-based repair on the success rate of texton locating using addition and multiplication.

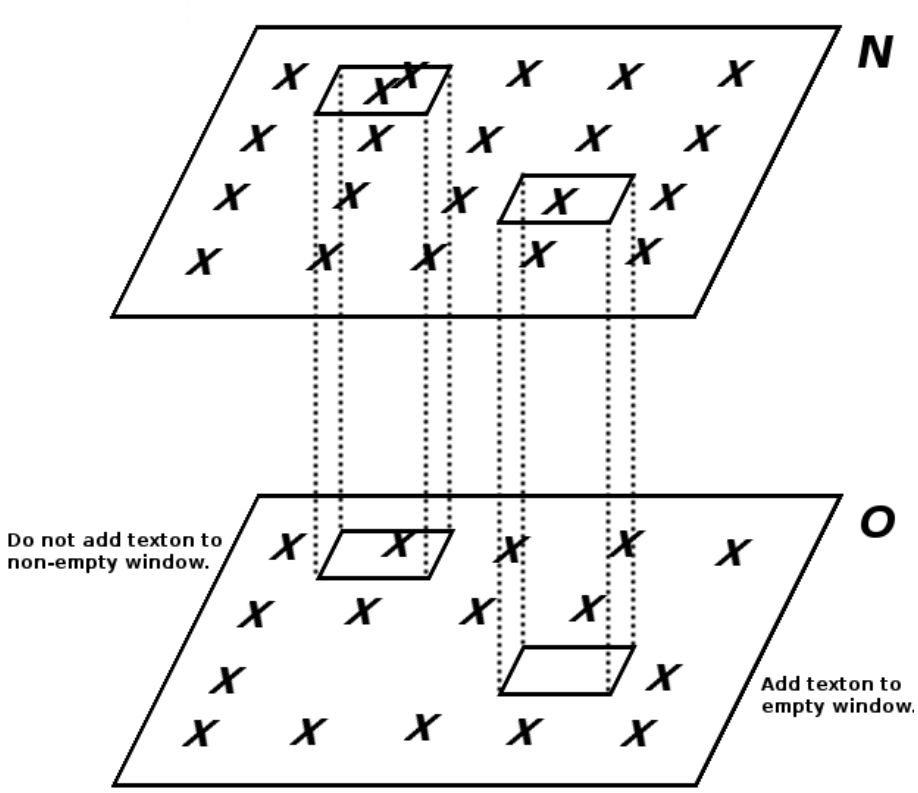
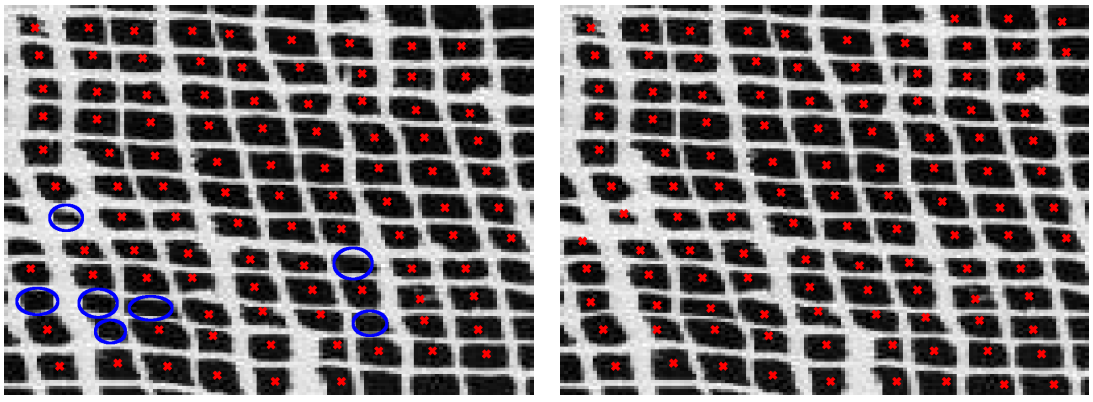


Figure 5.7: We use N to augment O by searching for empty regions in O which are centred at texton locations in N.



(a) Before frequency-based repair using local examination.

(b) After repair frequency-based repair using local examination.

Figure 5.8: The results of frequency-based repair using local examination. Errors in the original texton locating process are highlighted with circles in Figure 5.8(a).

To describe the local examination repair procedure we make a few definitions. We call the non-maximal suppression result given as input to the repair algorithm O, and the non-maximal suppression result acquired from the isolation of dominant frequencies of the power spectrum N. In summary, we begin with O, and use N to add peaks to O that were not properly extracted during non-maximal suppression. Figure 5.7 provides a visual example of this process. For each texton location in N we centre a region at that texton location. We leave the size of the region as a parameter of the algorithm, but have obtained good results with a size four thirds the width and height of the user-selected texton. We now use N to augment O by introducing new texton locations in O that previously did not exist. To do this, we look at each texton location $n \in N$ and project the region centred at n to O. We then search for texton locations in O that lie within this projected region. If no texton locations in O lie within the projected region, we augment O by including n. If a texton location in O lies within the projected region, we do not augment O. Figure 5.8 depicts the result of repair overlaid on a texture sample. Program Listing D.8 lists the frequency-based repair algorithm using local examination in full.

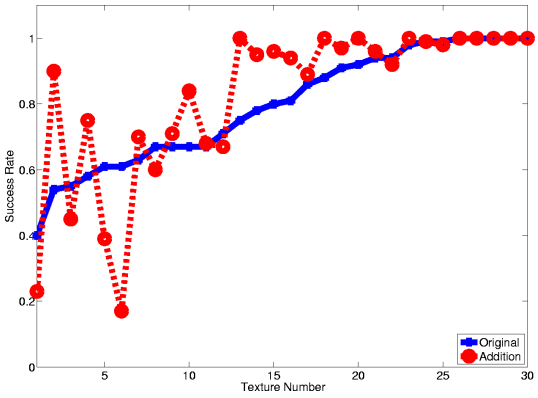
5.2 Results

We tested our frequency-based repair procedure and found that the method produces a corrected estimate of texton locations given an accurate estimate of original texton locations. However, the method is only useful on a small subset of textures, those that have few errors in the texton location data despite correlation with an adequate feature space. Unfortunately, we have been unable to find an analytical method for determining how many errors are allowed in the original texton locating result in order for our procedure to still work. Furthermore, the procedure works only if textons are placed using the assumption of two dominant texton placement frequencies. Despite these drawbacks, we have found frequency-based repair an adequate method of obtaining texton locations in difficult cases.

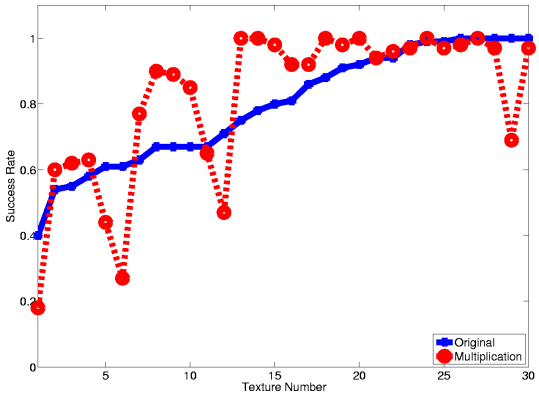
To test the performance of the frequency-based repair procedure, we repaired the texton locating result of 30 textures. We first recorded a texton locating success rate for each texture given a single user-selected sample texton. The success rate, S, is calculated as the number of correct texton locations detected, C, minus the number of erroneous texton locations detected, E, divided by the total number of textons in the image, T:

$$S = \frac{C - E}{T}.$$

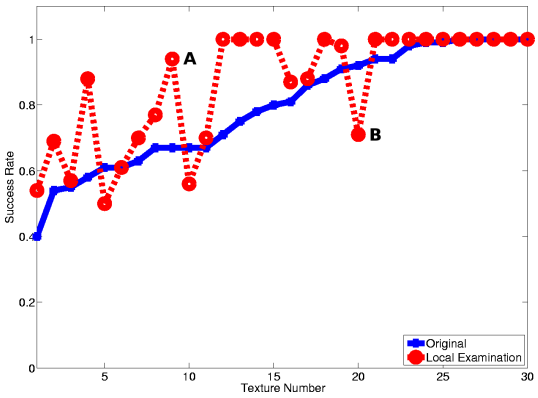
Within the 30 images tested, the success rate ranged from 0.40 to 1.00 on a scale of 0.00 to 1.00. We then ran the frequency-based repair algorithm using the multiplication, addition, and local examination strategies on each of the 30 results, and recorded the new success rate. Note that the local examination repair strategy can be applied given any estimate of texton locations. We



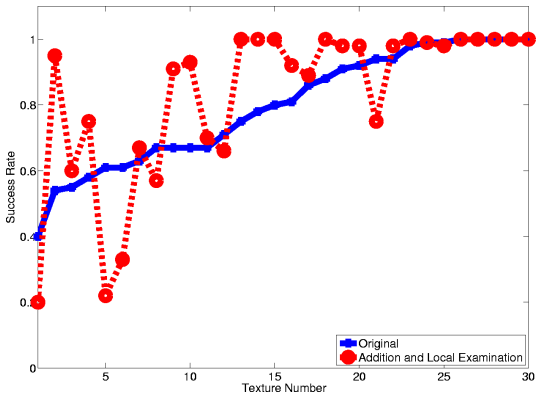
(a) Addition of surfaces.



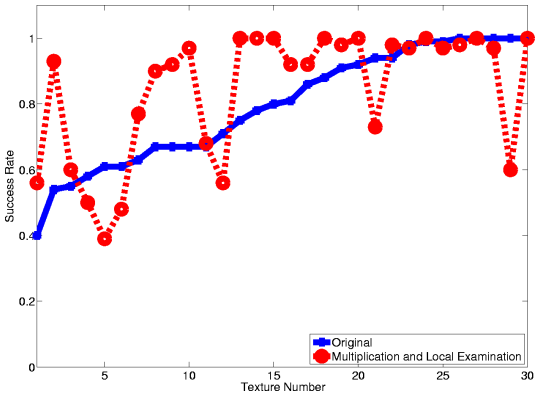
(b) Multiplication of surfaces.



(c) Local examination of surfaces.



(d) Addition of surfaces followed by local examination.



(e) Multiplication of surfaces followed by local examination.

Figure 5.9: The results of frequency-based repair on the success rate of texton locating.

therefore applied local examination to the texton location data produced from the multiplication and addition repair strategies to see if combining these strategies can create a more accurate repaired texton locating result than using these strategies in isolation.

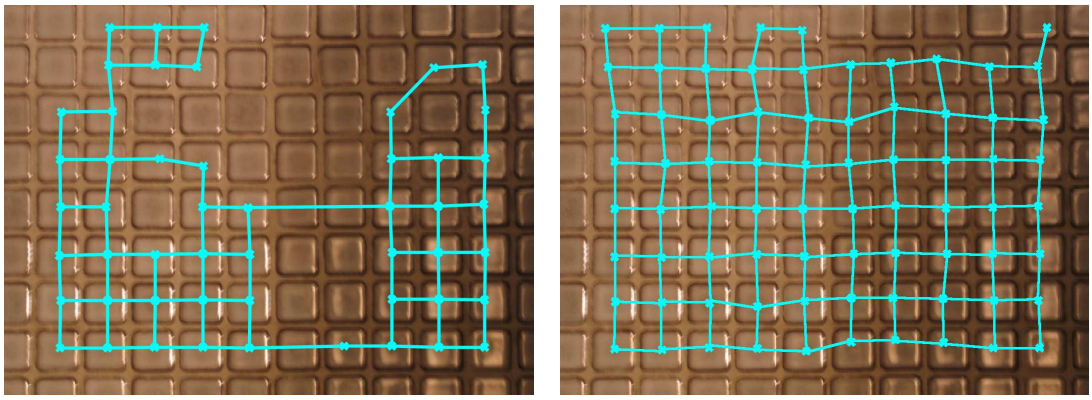
During testing, we encountered some cases where the texton locating success rate increased dramatically after frequency-based repair, and some cases where the texton locating success rate decreased. Figure 5.9 graphically depicts the results of these tests. Table E.7 contains the full results of these tests. In the figures, the vertical axis records the success rate of texton locating and the horizontal axis records the texture being tested. For all test cases, the local examination strategy used a region size of four thirds of the width and height of the user-selected sample texton size.

We have found that no repair strategy is able to improve the texton locating result in every case. From the data we can establish that multiplication is not a good method of repairing a texton locating result. A common problem with this technique is that too many texton locations are reported. These erroneous locations dramatically decrease the texton locating success rate. Addition provides better results than multiplication, and at times better results than local examination. However, local examination most consistently improves upon the texton locating result. Overall, repairing the texton locating result using local examination either increases or does not change the success rate of texton locating in 27 out of 30 cases. Sequencing local examination with multiplication or addition generally provides better results than repair with multiplication or addition alone. However, repair using local examination alone still provides the most consistent results out of any method, and we examine the results of local examination more thoroughly in the remainder of this section.

Two interesting data points in the results of repair with local examination have been highlighted with letters in Figure 5.9(c). Data Point A depicts a vastly improved texton locating result, while Data Point B depicts a reduced texton locating result.

Figure 5.10 depicts the successful repair result of Data Point A. The original texton locating result is shown in Figure 5.10(a). The texton locating success rate in this example is 56% (textons partially occluded by the image boundary are ignored when calculating success rate). After repair, the texton locating success rate increases dramatically to 100%. The result is shown in Figure 5.10(b). This example shows how the frequency-based repair procedure is particularly useful in correcting texton locating defects in textures with regularly arranged textons.

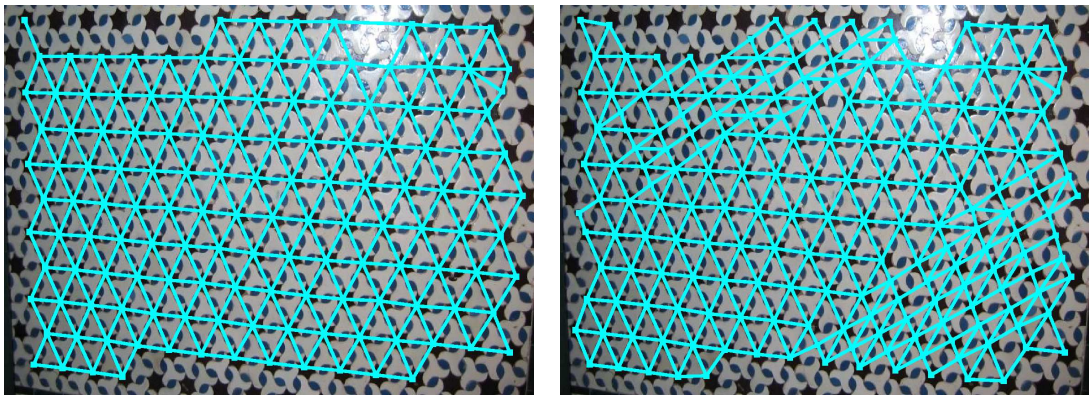
Figure 5.11 depicts the unsuccessful repair result of Data Point B. Figure 5.11(a) shows a correct texton locating result. However, after applying the frequency-based repair procedure, errors in the texton location data are introduced. The repaired result is shown in Figure 5.11(b). The repair procedure fails on this texture sample because the placement of textons within the texture are not accurately approximated by two translation vectors due to the perspective skew of the image.



(a) Before frequency-based repair using local examination.

(b) After frequency-based repair using local examination.

Figure 5.10: The results of frequency-based repair using local examination with a window four thirds the width and height of the user-selected texton size. Given a texture with regularly spaced textons our procedure can repair many texton locating defects.



(a) Before frequency-based repair using local examination.

(b) After frequency-based repair using local examination.

Figure 5.11: The results of frequency-based repair using local examination with a window four thirds the width and height of the user-selected texton size. In this case the repair procedure introduces errors in the texton locating result.

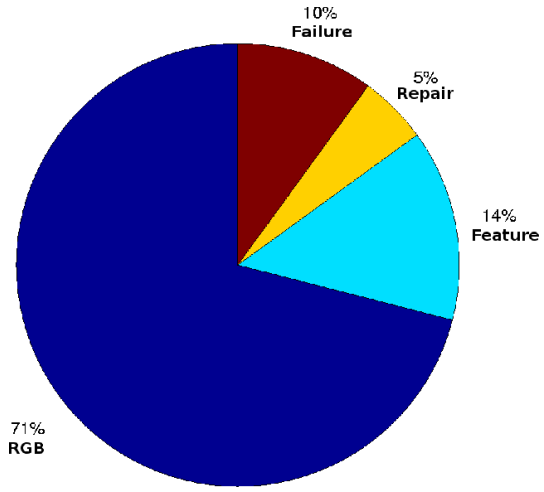


Figure 5.12: Using image features and frequency-based repair we are successfully able to locate textons 90 out of 100 times.

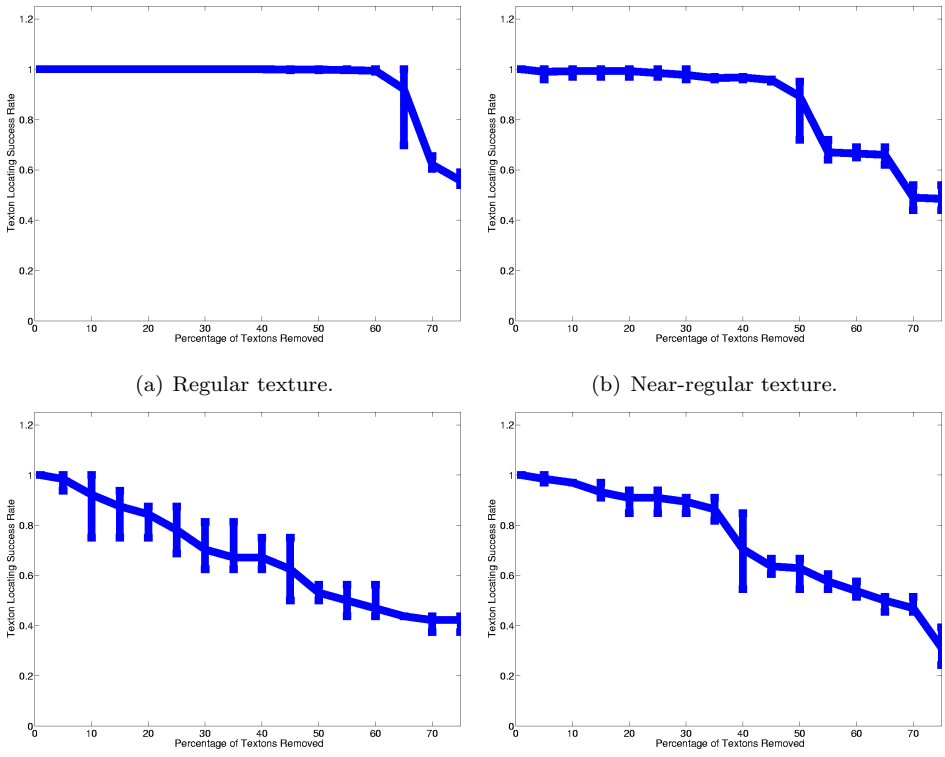
In further testing, we attempted to correctly isolate textons in the 15 remaining failure cases out of the 100 near-regular test textures cited in Section 3.2. Within these failure cases, we determined that 5 results could be repaired. This gives a total of 90 successful texton extractions out of 100 textures using a combination of the standard correlation procedure of Chapter 3, and the extensions discussed in this chapter. Figure 5.12 visually depicts this result.

As a last test, we tried to determine the number of textons which can be removed from a correct texton locating result before frequency-based repair with local examination fails. To run the test we first gathered 20 near-regular textures with perfect texton locating success rates. Success rate is measured in the same way reported earlier:

$$S = \frac{C - E}{T}.$$

Texton locations were then randomly removed from the texton locating data of each of the 20 texture samples. At intervals where 1, 5, 10, 15, ..., and 75 percent of the texton locations had been removed, we recorded the texton locating success rate after frequency-based repair. We then repeated the experiment four times for each texture to remove any bias related to the random order with which texton locations had been removed.

Figure 5.13 depicts the results of these tests for four textures. Full results can be found in Figures E.1 to E.9. The curve in each figure represents the average texton locating success rate of the four trials. Error bars at each data point record the lowest and highest texton locating success rate of the four trials. The four results displayed here have been chosen based on the perceived regularity of each texture. We can see from Figure 5.13(a) that the frequency-based repair procedure using local examination is able to create a correct texton locating result from regular texture when as many as 60% of textons have been removed. The results are not as strong



(a) Regular texture.

(b) Near-regular texture.

(c) Slightly irregular texture.

(d) Irregular texture.

Figure 5.13: Results of testing for robustness of the local examination frequency-based repair strategy. The horizontal axis records the number of texton locations removed from the data and the vertical axis records the texton locating success rate after frequency-based repair.

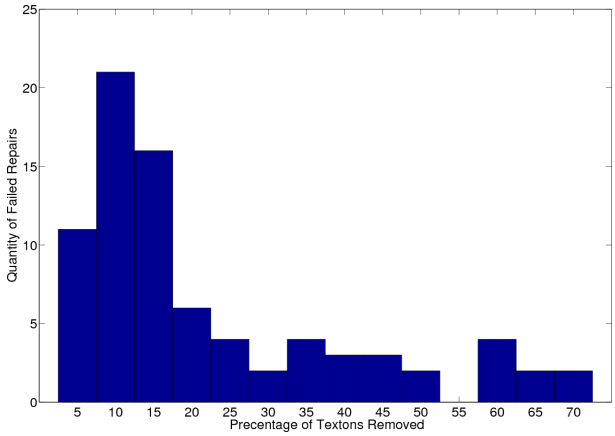


Figure 5.14: A histogram recording the point of first failure after a percentage of textons have been removed.

when texture becomes irregular, where in some cases as little as 5% of textons need to be removed from the data before the frequency-based repair procedure using local examination fails to create a correct texton locating result. In fact, 5% and 60% represent the extreme ranges of the number of texton locations which may be removed from the data before frequency-based repair fails. The median value of the number of textons that may be removed before the repair algorithm fails is 10%, however, this number does not take into account the severity of error, which is in most cases 1 error in the entire texton locating result. On average during our tests, about 18% of textons may be removed from the data before frequency-based repair fails.

Figure 5.14 depicts a histogram recording the point where frequency-based repair first fails to recreate the texton locating result. The horizontal axis reports the percentage of texton locations removed from the data, and each bin stores the quantity of test textures which failed when that percentage of texton locations had been removed. From the histogram, we can see that the majority of failures occur when between 10% and 15% of texton locations have been removed from the data. Thus, the texton locating data must be nearly correct for our frequency-based repair procedure to improve the texton locating result.

We have found evidence that frequency-based repair works well on textures whose textons are located approximately the interesection of two translation vectors. In other words, textures without much geometric variation in texton placement are good candidates for frequency-based repair. A second trend we have witnessed during evaluation of frequency-based repair is that errors in the texton locating result must be localized for frequency-based repair to work. If, for example, an entire row of texton locations is missing from the data, the frequency-based repair procedure will incorrectly estimate the frequencies responsible for texton placement, and fail to repair the result.

The frequency-based repair algorithm discussed in this section can be applied to only a small number of texture cases. The algorithm works well in cases where the majority of textons are correctly extracted, with a only a few textons missing. Although there is no method of analytically determining the maximum number of holes that are allowed before the algorithm fails, we have estimated that approximately 90% of textons must be present for the algorithm to work correctly. This seems like an imposing number, but the texton locating techniques already developed ensure we are able to locate most, if not all, textons in a typical texture. The algorithm does not work well when given a poor estimate of initial texton locations. If this is the case the algorithm has a difficult time determining the true frequencies responsible for texton placement. The results of our testing are not strong enough to allow us to recommend using frequency-based repair in all texton locating cases. However, the frequency-based repair procedure is meant to be used as a last resort when texton locating is mostly accurate, and in this situation it provides a corrected estimate of texton locations suitable for repairing a texton locating result.

Chapter 6

Lattice Construction

In Chapter 3 we discussed the problem of locating textons in a near-regular texture. Given this texton location information, we now turn to the problem of joining the textons in a graph structure in order to form a lattice. The texton locations will be connected by edges in the graph. More precisely, we wish to create a graph $G = (V, E)$, where V is a set of texton locations, and $E = (v_i, v_j)$ is a set of edges, where $v_i, v_j \in V$. Each edge $e \in E$ connects texton location $v \in V$ to its most perceptually sensible neighbours.

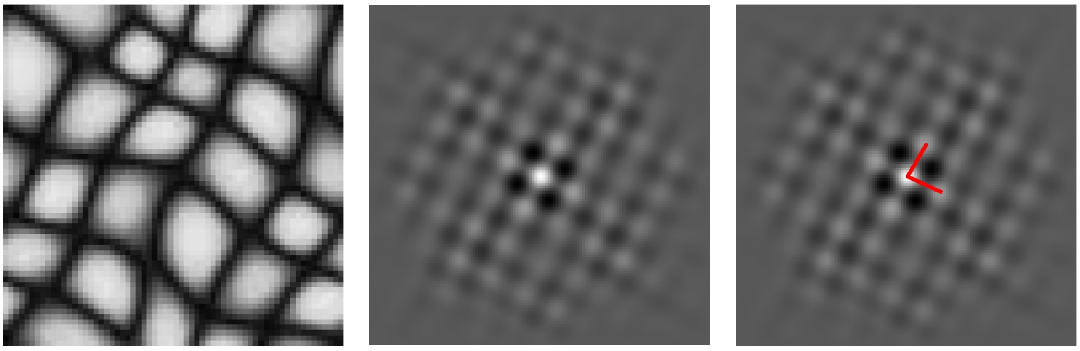
We present two methods of constructing a lattice from texton locations. The first method is ad-hoc, and produces a sensible lattice for near-regular texture [62]. The ad-hoc method ignores erroneous texton locations, but as texture becomes irregular this method fails to construct a perceptually sensible lattice. We discuss the ad-hoc method in Section 6.1. To extend our result for constructing a sensible lattice from irregular texture, we apply results from the proximity graph literature to our problem. This graph-theoretic lattice construction method is described in Section 6.2

6.1 Ad-Hoc Lattice Construction

We present here an ad-hoc lattice construction procedure. Given a binary image of texton locations, the procedure creates a lattice connecting the texton locations. This procedure can be used to create a sensible lattice for many textures [62]. Furthermore, this procedure is able to create a sensible lattice despite errors in the texton location data; the procedure correctly ignores texton locations that are at improper positions, successfully overcoming errors due to misplaced textons.

The ad-hoc lattice construction procedure assumes that textons are placed in the texture at the intersection points of two translation vectors defining texton placement, as described by Hamey [24]. This assumption is valid for many texture samples by the definition of near-regular texture [45]. We provide a method of estimating translation vectors by examining the correlation surface of a texture sample.

A texture sample is given in Figure 6.1(a), and its smoothed correlation surface is given in Figure 6.1(b). The largest peak in the correlation surface is the location where the image is exactly



(a) An input texture.

(b) The input texture's smoothed auto-correlation surface.

(c) Translation vectors determined from the texture's auto-correlation surface.

Figure 6.1: The correlation surface of a texture can be used to find translation vectors.

correlated with itself, which serves as the origin of the translation vectors. The next largest peak in the surface is the maximum correlation of the image with a translated version of itself. The vector from the image origin to this largest peak is the first translation vector. Similarly, the vector from the origin to the third largest peak in the correlation surface is the second translation vector. Following the advice of Leu [35] we require that the angle between the two translation vectors lie between $\frac{\pi}{4}$ and $\frac{3\pi}{4}$, ensuring that the translation vectors can be used to approximate a tiling of the plane. Until this condition is satisfied, we continually choose the next largest peak in the correlation surface as the new second translation vector. Figure 6.1 depicts the result of translation vector estimation.

During the lattice construction algorithm, texton locations can be of two types: *actual* and *potential*. Texton locations are called actual when they have been added to the constructed lattice. Texton locations are called potential when they have yet to be added to the constructed lattice, but are still under consideration for addition to the lattice. Potential texton centres are points of high correlation with the user-selected texton sample, and are determined by the locations of local maxima in the surface produced by correlating the user-selected texton sample with the input texture.

The lattice is constructed iteratively by extending edges from actual texton locations, and searching for a potential texton location to add to the lattice. The algorithm makes use of a queue storing actual texton locations; texton locations newly attached to the lattice are inserted into the queue to be processed at a later time. Each texton location in the queue is processed by extending edges to potential texton locations. The processing stops when the queue becomes empty, signalling that no other potential texton locations can be added to the lattice.

Listing 6.1: Pseudo-code of the ad-hoc lattice construction algorithm.

```
init := starting texton centre
Q.push(init)
while (Q is not empty)
  current := Q.pop()
  if (current has not been visited)
    current.visited := true
    for each 4 translation vectors starting at current
      locate best potential texton centre p near translation vector endpoint
      Q.push(p)
    end for
  end if
end while
```

We will now explain the lattice extraction algorithm in more detail. Program Listing 6.1 provides pseoudo-code of the algorithm under discussion. First, we insert the texton location initially selected by the user into the queue. This location should have the highest value in the correlation surface for this texture. However, Lin's smoothing algorithm may have distorted the correlation surface, creating a higher correlation value at a different location, or altering the location of maxima. Therefore, we require a search for the potential texton location closest to the user-selected texton location. This potential texton location is the first location to be added to the queue and deemed an actual texton location of the lattice. Having populated the queue with one texton location, we begin a processing loop of removing a location from the queue, attaching the location to four neighbouring potential texton locations using translational vectors, and adding these potential texton locations to the queue as actual texton locations.

After removing a texton location from the queue, we check to see if it has been processed. If so, we ignore this point and process the next location in the queue. If not, we mark this point as processed and attempt to connect it to potential texton locations. To connect to potential texton locations, we first compute the most likely position that actual locations exist. These locations are given by the position of the current texton location added to the two translation vectors and their negations. In total, we attempt to connect the actual texton location to four potential locations.

To make a connection we search within an ellipse described by a third of the length of each translation vector, centred at the most likely texton location. The potential texton location with highest correlation value lying within this ellipse is connected to the lattice; its endpoint is added to the priority queue as an actual texton location. If no potential texton location exists within this ellipse, we increase the ellipse size until a connection can be made. This increase in ellipse size continues until a potential texton location is found and connected to the lattice. To avoid searching

over a disproportionately large area of the image, we cancel the search if the ellipse extends outside of the image boundary, and do not connect to a potential texton location in this direction.

Figure 6.2 depicts one lattice extraction iteration. The algorithm begins by inserting the first actual texton location, the one selected by the user, into the processing queue (Figure 6.2(a)). This texton location, call it a for actual, is then removed from the queue and processed. Processing begins by extending a translation vector from a (Figure 6.2(b)), and searching in an ellipse around this translation vector's endpoint (Figure 6.2(c)). Once a texton location, call it p for potential, has been found within this ellipse, it is marked as an actual texton centre and the edge (a, p) is stored as an edge of the lattice (Figure 6.2(d)). Finally, p is added to the queue for later processing. The algorithm continues by extending a translation vector from a in the opposite direction of the first translation vector (Figure 6.2(e)), and a search is performed at the vector's endpoint for a potential texton location to add to the lattice (Figure 6.2(f)). If the search is successful, the edge is added to the lattice, and the location found added to the queue (Figure 6.2(g)). The algorithm continues in the same manner for the second translation vector (Figures 6.2(h) to 6.2(l)), until we have completed processing a and extended the lattice in four directions (Figure 6.2(m)).

Each texton location added to the queue is then processed in turn in the same manner. Upon completion of the algorithm, each texton location in the lattice will have been connected to its perceptually sensible neighbours, given the assumption that two translation vectors approximate texton placement. Figure 6.3 depicts the final result of lattice construction using this algorithm. The output of the algorithm is the edge list corresponding to the texture's lattice.

There are three special cases to note. The first special case occurs when our search for a texton location extends outside of the image boundary, or if the translation vector endpoint lies outside of the image boundary. In this case, we cancel the search since the correlation surface is often not reliable near the boundary, and any texton locations found near the boundary may be erroneous. This special case is depicted in Figure 6.4. We begin at the scenario depicted in Figure 6.4(a), with the next actual texton location being processed circled in Figure 6.4(b). Figure 6.4(c) shows the extension of a translation vector, and Figure 6.4(d) depicts the search ellipse centred at the translation vector endpoint. No texton location is found within this ellipse, and so the size of the search ellipse is increased. Eventually, in Figure 6.4(e), the size of the search ellipse is increased until it intersects with the image boundary, and the search for a new texton location is cancelled, resulting in Figure 6.4(f).

The second special case is if within our search ellipse we encounter two or more potential texton locations. In this case we choose the texton location with highest correlation value. This case is depicted in Figure 6.5. Figure 6.5(a) depicts the starting point, when an actual texton centre has connected to three of its potential neighbours. We then extend a translation vector to the neighbourhood of a fourth potential texton location in Figure 6.5(b), and in Figure 6.5(c) we

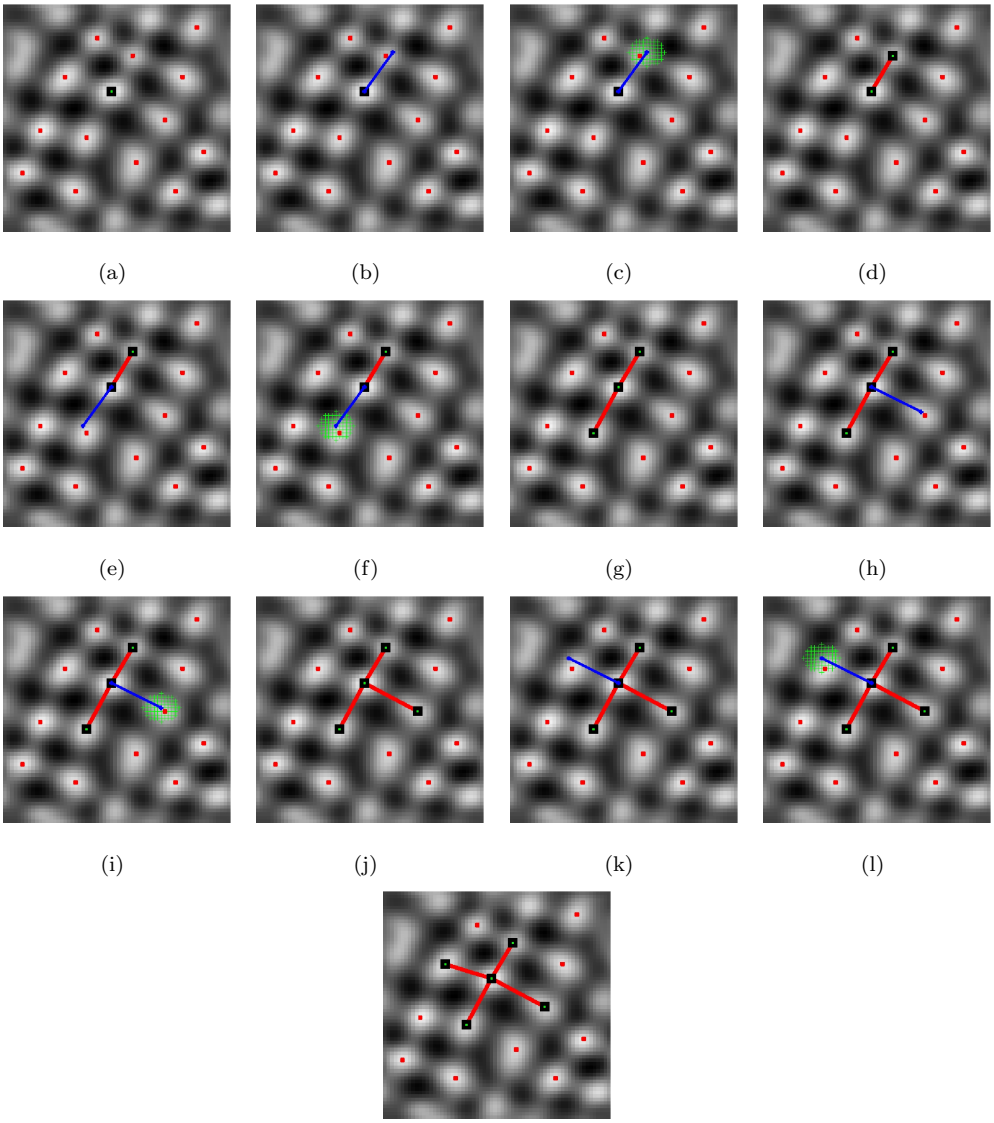
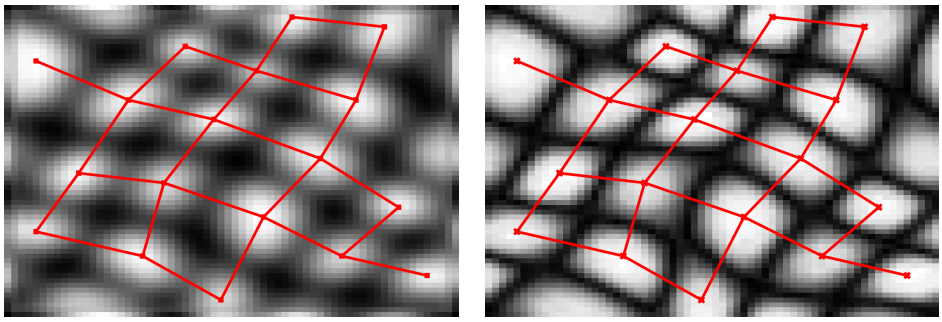


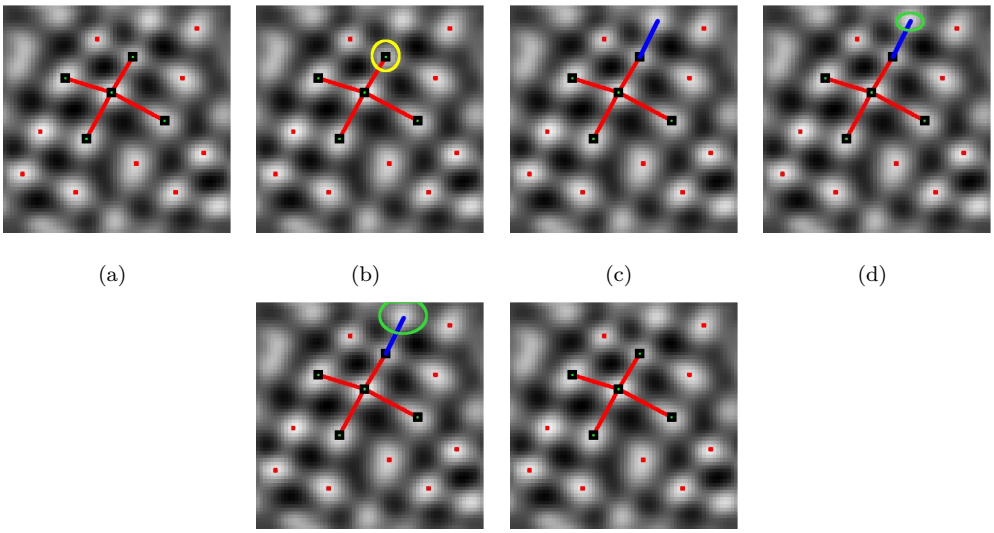
Figure 6.2: One iteration of the lattice construction algorithm. The algorithm begins at the user-selected sample texton and extends translation vectors out in search of texton locations.



(a) Extracted lattice overlaid on the correlation surface.

(b) Extracted lattice overlaid on the input texture.

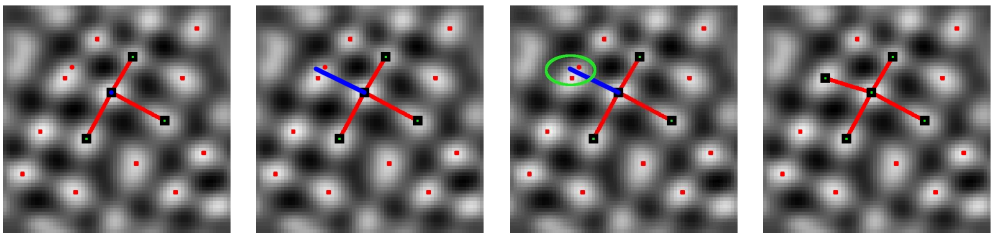
Figure 6.3: A lattice extraction result.



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

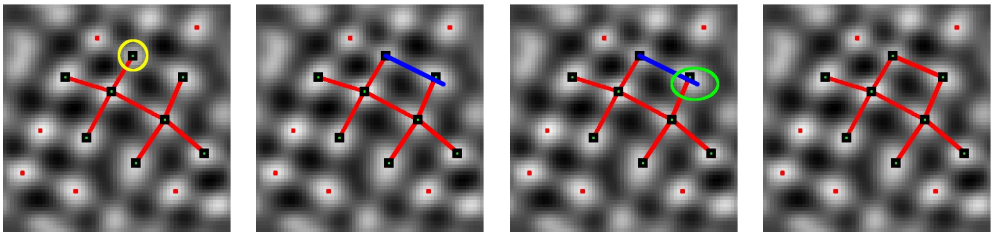
(e) (f)

Figure 6.4: When a search for a texton location extends outside of the image boundary, we cancel the search, and make no lattice edge extending in this direction.



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

Figure 6.5: When two potential textons lie within the search ellipse, the one with higher value in the correlation surface is chosen.



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

Figure 6.6: When the search ellipse encounters an actual texton location, we connect to this texton location regardless of the contents of the remainder of the search ellipse.

encounter within our search ellipse two potential texton locations. Here we choose the potential texton centre most highly correlated with the user-selected texton sample, leaving the extra texton not connected to the lattice in its current position. The resulting scenario is depicted in Figure 6.5(d).

The third special case is if within our search ellipse we encounter an actual texton location. In this case we create an edge to the actual texton location without regarding the remaining contents of the search ellipse. This case is depicted in Figure 6.6. The starting point is shown in Figure 6.6(a) with the texton location being processed highlighted. From this location we extend a translation vector to the right, which is shown in Figure 6.6(b), and create a search ellipse, which is shown in Figure 6.6(c). Within the search ellipse lies an actual texton location, and regardless of the remainder of the contents of the search ellipse we connect to the actual texton location, resulting in Figure 6.6(d). A Matlab implementation of the lattice construction algorithm is given in Program Listing D.6.

We have tested this lattice construction algorithm on a wide variety of textures, with good results [62]. A distinct advantage of this algorithm is its adaptation to improperly located textons. Specifically, the second special case, depicted in Figure 6.5, shows how the algorithm selects between multiple potential texton locations. However, the algorithm fails when texton placement is not determined by two translation vectors; even local irregularities in texton placement may result in an inaccurate lattice construction. We have overcome this problem by obtaining more accurate texton locating data using feature space, described in Chapter 4, using frequency-based repair to correct for errors in the texton locating result, described in Chapter 5, or by using graph-theoretic lattice construction methods, which do not assume that texton placement is determined using two translation vectors. We describe the graph-theoretic methods in Section 6.2.

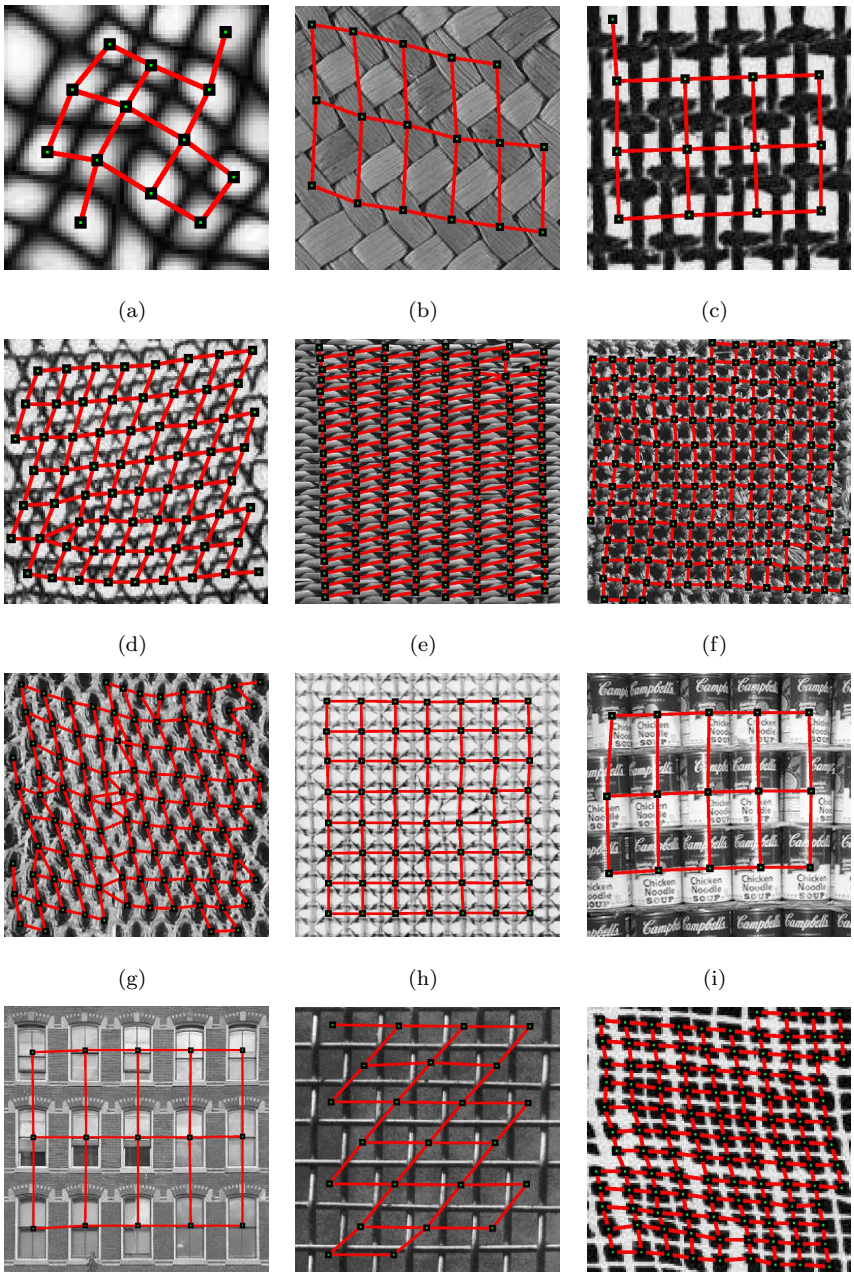
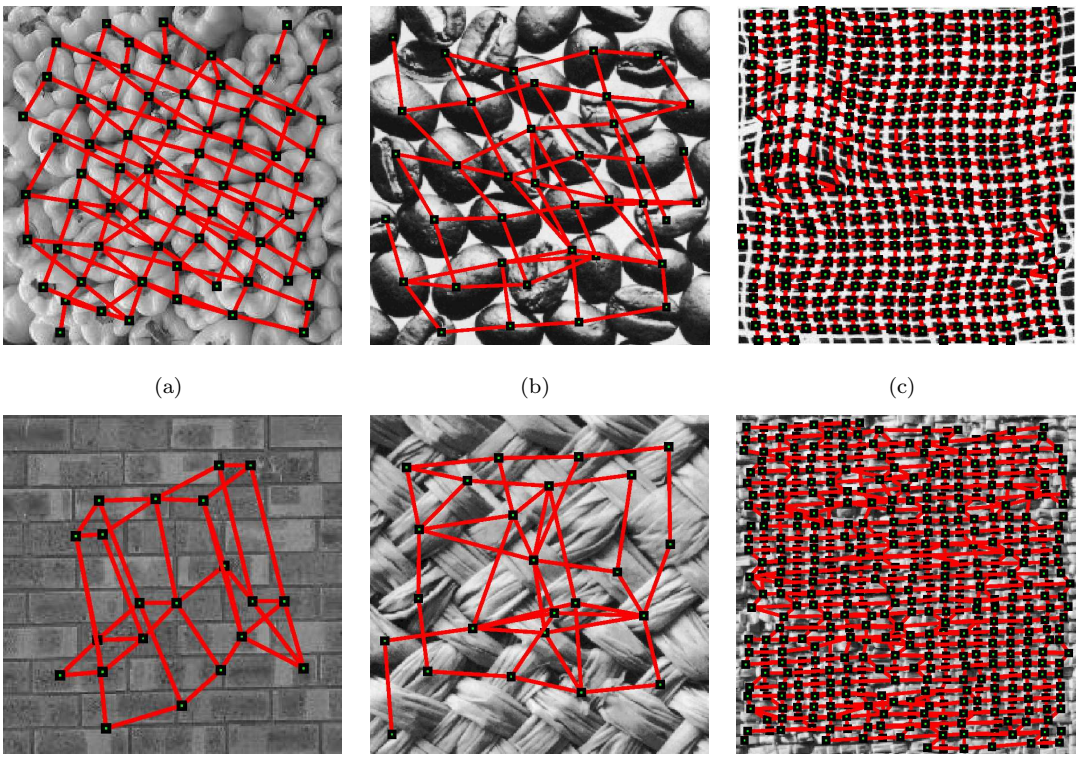


Figure 6.7: Ad-hoc lattice construction successes. Figure 6.7(b) is from the VisTex database [33]. Figures 6.7(e) and 6.7(f) are from the database of Simoncelli. Figure 6.7(a) is from the database of DeBonet. All remaining figures are from the Brodatz texture album [9].



(a) (b) (c)

(d) (e) (f)

Figure 6.8: Ad-hoc lattice construction failures. Figure 6.8(a) is from the database of Simoncelli. The remaining figures are from the Brodatz texture album [9].

6.1.1 Results

Figure 6.7 depicts lattice extraction successes. Figure 6.7(a) shows our algorithm handle significant irregularity in geometry and pixel intensity. Figures 6.7(b) to 6.7(f) depict extraction results from a common class of near-regular textures, containing textons of slightly varying geometry and colour. The honeycomb geometry of Figure 6.7(d) presents no problems for our algorithm. Figures 6.7(g) and 6.7(h) depict a correct lattice extraction result from textures with significant irregularities. Figures 6.7(i) to 6.7(k) depict successful lattice extractions from textures with similar texton geometry, but varying texton colour. Finally, Figure 6.7(l) depicts a successful lattice extraction from a texture with varying texton geometry, but similar texton colour.

Figure 6.8 depicts lattice extraction failures. Figures 6.8(a) and 6.8(b) are from irregular texture. Strong texton similarity exists, but irregular texton placement hampers lattice extraction results. Figure 6.8(c) contains a local irregularity, where our algorithm will sometimes fail. The brick texture of Figure 6.8(d) is a failure case due to significant variation in brick colour. In Figures 6.8(e) and 6.8(f) the graph construction begins admirably but becomes lost amidst the high frequency noise.

The ad-hoc lattice extraction algorithm has been shown successful for a wide variety of textures [62]. Our algorithm is successful particularly for textures with little geometric variation. In this class of texture, if at least one potential texton centre lies within each texton, the ad-hoc lattice construction algorithm will correctly construct a lattice by searching for the most plausible texton centre in the area. Knowing this, we can relax the accuracy of the correlation and non-maximal suppression procedures by selecting a smaller sample texton to use during correlation, increasing the likelihood of more texton centres being located. This leads to the main strong point of the algorithm, which is ignoring incorrectly placed texton centres to arrive at a sensible lattice despite errors in the texton locating procedure.

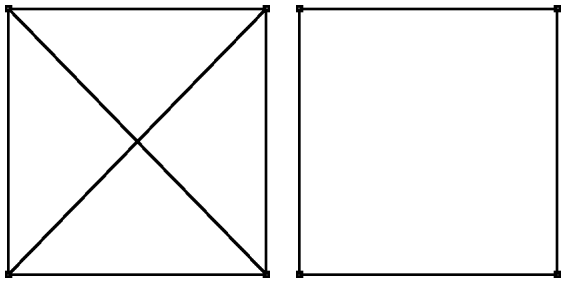
Unfortunately, our algorithm may fail when textons are not aligned according to two translation vectors. Even small irregularities in texton placement have the potential to severely affect the output of the algorithm since errors may become compounded as new potential texton locations are inserted into the processing queue based on erroneous texton locations already included in the lattice. Due to this drawback, we have adapted research on proximity graphs to properly extract a lattice from textures with irregular texton placement.

6.2 Graph-Theoretic Lattice Construction

Proximity graphs are a natural fit for the problem of connecting each point in a set to its nearest neighbour. We reviewed proximity graphs in the Section 2.5 of the Literature Review, and showed how a continuum of graphs are available for any point set. In what follows we apply the idea of proximity graphs to the problem of lattice construction for near-regular texture.

Each of the graph types discussed in Section 2.5 may be applied to the problem of lattice extraction, with varying results. As previously mentioned, the NNG may not be connected, and so fails as a candidate lattice. The MST is by definition connected, but to have minimum length it will connect each texton location to only one neighbouring texton location, violating the conditions of a perceptually sensible lattice. We call these two graph structures sparse, in the sense that the number of edges in the graph is below the required amount for a sensible lattice. On the other hand, the DT is dense, in the sense that too many edges in the graph are present to provide a sensible lattice. Eliminating these graphs leaves the RNG and the GG as candidates for constructing a sensible lattice.

We have successfully used both the RNG and the GG to produce a lattice given accurate texton locations. To compare the two graph types, consider four texton locations forming an approximate square. The RNG connects the perimeter of the four points with edges, while the GG will add a diagonal edge between the four points (see Figure 6.9). In this case, it is arguable which graphing method produces the most sensible lattice. Now consider a set of texton locations that form a



(a) The Gabriel graph of four points.

(b) The relative neighbourhood graph of four points.

Figure 6.9: The Gabriel graph adds additional diagonal edges to a lattice connecting four points approximating a square, while the relative neighbourhood graph does not.

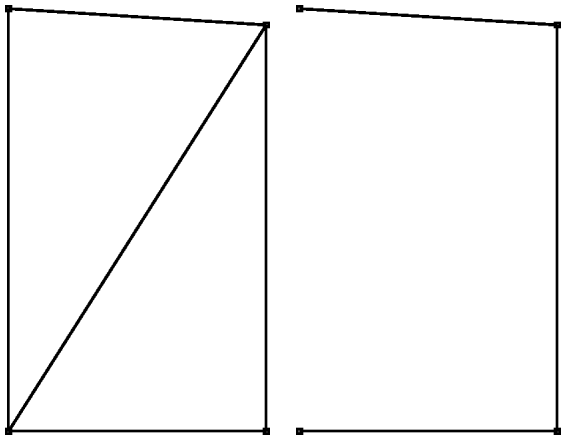
rectangle with height significantly greater than width. In this case, the RNG may fail to create an edge between the long sides of the rectangle, while the GG will create edges around the perimeter of the points (see Figure 6.10). In this case, the GG is the sensible choice for lattice construction.

These observations imply that the RNG and GG can be used to create a sensible lattice in certain situations, and that they can be used as complements towards the same goal. Often times, where the RNG fails the GG will succeed, and vice-versa. Together, these two graphing methods succeed on a wide variety of texture samples, but in some cases provide poor notions of neighbourliness between texton locations, and thus fail to create a sensible lattice. We thus turn to a parameterized notion of neighbourliness to provide a means of constructing a sensible lattice under conditions where the RNG and GG fail.

We use the β-skeleton spectrum as a means of constructing a lattice from a set of texton locations. Varying the value of β results in varying graph formations, ranging from the MST for β approaching ∞, to a fully connected graph for $\beta = 0$. Figure 6.11 depicts the result of varying β on a set of points, and Figures 6.12 and 6.13 depict the result of varying β in two lattice construction examples. We have found that the β-skeleton spectrum allows for lattice construction that can be tailored for the user's needs. It would be advantageous to predetermine the β value necessary for constructing a sensible lattice, so that no user intervention is necessary, but we leave this idea as an area of future work.

6.2.1 Results

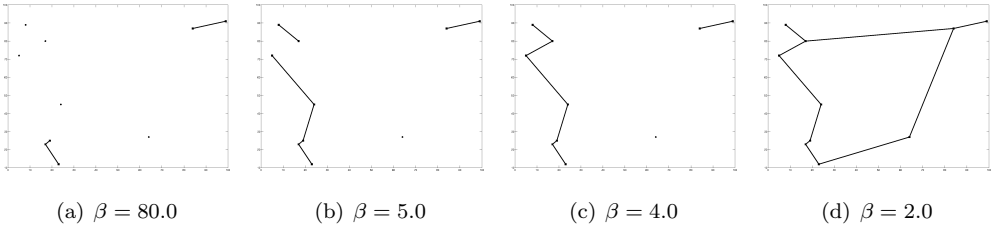
Lattice construction using a graph-theoretic approach provides a solution to our problem of extracting a lattice from near-regular texture. Unfortunately, a specific problem with the approach is that it is unclear whether a certain value of β produces a more sensible lattice than another value of β. However, because of the continuum of possibilities that are given by the β-skeleton spectrum,



(a) Gabriel Graph of four points.

(b) Relative Neighbourhood Graph of four points.

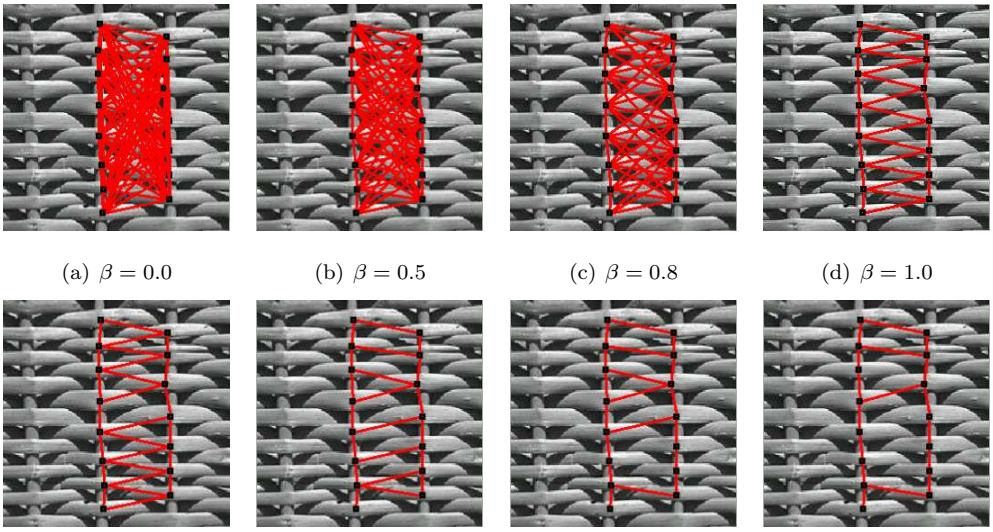
Figure 6.10: The Gabriel graph sensibly connects four points approximating a rectangle, while the relative neighbourhood graph does not.



(a) $\beta = 80.0$ (b) $\beta = 5.0$ (c) $\beta = 4.0$ (d) $\beta = 2.0$

(e) $\beta = 1.0$ (f) $\beta = 0.8$ (g) $\beta = 0.6$ (h) $\beta = 0.4$

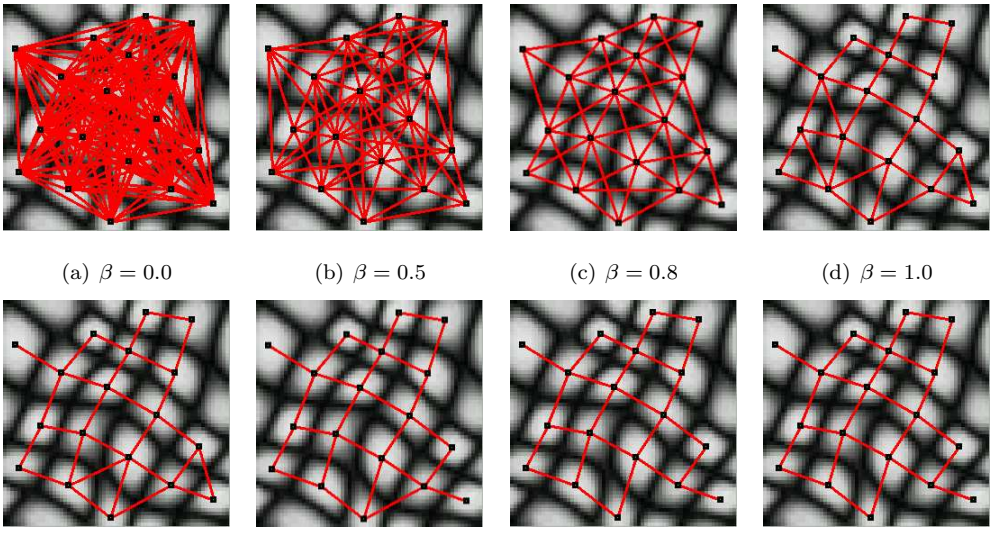
Figure 6.11: A sequence of β-skeletons for a texture sample.



(a) $\beta = 0.0$ (b) $\beta = 0.5$ (c) $\beta = 0.8$ (d) $\beta = 1.0$

(e) $\beta = 1.3$ (f) $\beta = 1.5$ (g) $\beta = 1.8$ (h) $\beta = 2.0$

Figure 6.12: A sequence of β-skeletons for a texture sample.



(a) $\beta = 0.0$ (b) $\beta = 0.5$ (c) $\beta = 0.8$ (d) $\beta = 1.0$

(e) $\beta = 1.3$ (f) $\beta = 1.5$ (g) $\beta = 1.8$ (h) $\beta = 2.0$

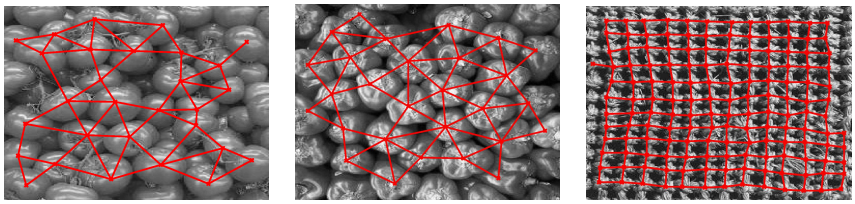
Figure 6.13: A sequence of β-skeletons for a texture sample.

if we are given correct texton locations a suitable lattice may be found using this method. We leave automatic determination of an appropriate β value as future work. Program Listing D.7 references a Matlab program for constructing β skeleton for a set of textons.

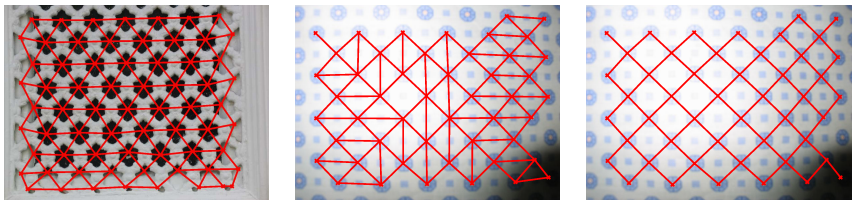
Figure 6.14 shows some results of graph-theoretic lattice construction. Figures 6.14(a) and 6.14(b) show how graph-theoretic lattice construction can produce a sensible lattice in cases where the ad-hoc method fails. Specifically, the textons in these two images follow an irregular placement pattern, which poses no problem for graph-theoretic lattice construction when the textons are correctly located. Figures 6.14(c) and 6.14(d) depict successful lattice construction in quite regular images using β values of 1.0 and 2.0, respectively. Figures 6.14(e) and 6.14(f) show two forms of lattice construction using β values of 1.0 and 2.0, respectively. The result of using β values of 1.0 and 2.0 are shown again in Figures 6.14(g) and 6.14(h). Figure 6.14(i) depicts a lattice construction result with a β value of 1.8.

These results highlight the importance of using a proper value of β for lattice construction. In our tests, we have found that a good value for β is in the range $1 \leq \beta \leq 2$, with values of $\beta = 1$ (GG) and $\beta = 2$ (RNG) satisfactory in most cases. As noted previously, it is common for a β value of 1 to create additional edges between textons that form an approximate square, and for a β value of 2 to fail when considering a lattice with textons that form an approximate rectangle. By altering β between the values of 1 and 2, we can consistently arrive at a correct lattice extraction. A notable area of future work would be to adaptively choose a β value depending on the layout of texton locations.

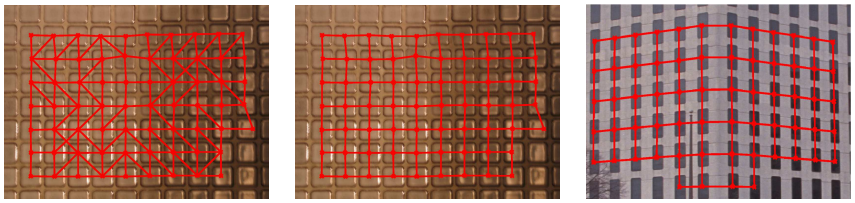
In this chapter, we have discussed two methods of constructing a lattice given the location of textons. Both methods construct a sensible lattice given appropriate input, with the ad-hoc method having the benefit of being able to ignore erroneous data, and the graph-theoretic method having the benefit of being able to construct a lattice given irregularly placed textons.



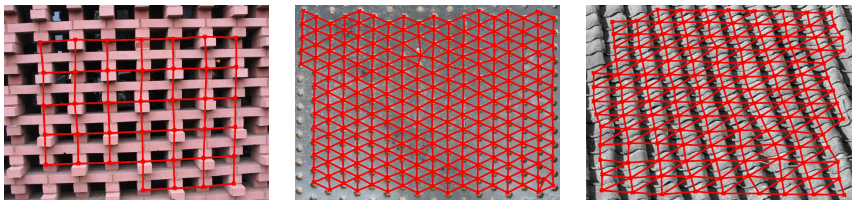
(a) $\beta = 2.0$ (b) $\beta = 2.0$ (c) $\beta = 1.0$



(d) $\beta = 2.0$ (e) $\beta = 1.0$ (f) $\beta = 2.0$



(g) $\beta = 1.0$ (h) $\beta = 2.0$ (i) $\beta = 1.8$



(j) $\beta = 1.0$ (k) $\beta = 2.0$ (l) $\beta = 2.0$

Figure 6.14: Graph-theoretic lattice construction results. Figures 6.14(a) and 6.14(b) are from the database of Simoncelli. Figure 6.14(c) is from the Brodatz texture album [9]. Figure 6.14(i) is from the VisTex texture database [33]. The remaining figures are from the CMU Near-Regular Texture Database [13].

Chapter 7

Conclusions and Future Work

In this thesis, we have presented a method for extracting a lattice from near-regular texture. The method has been divided into two core portions: texton locating and texton graphing. We use correlation to locate textons; correlation of a sample texton with the input texture produces a surface whose maxima correspond to texton locations. Extracting these maxima using non-maximal suppression locates textons in the input texture. To connect textons to form a lattice, we have presented two texton graphing procedures: an ad-hoc procedure is used to connect textons which are placed in the texture at the intersections of two translation vectors, and the graph theoretic procedure uses the idea of proximity graphs to connect textons whose neighbourhoods contain no other texton locations. Furthermore, we have presented work on improving the accuracy of the texton locating method by correlating within feature space, and by repairing a result using frequency-space analysis.

Our texton locating procedure has minor shortcomings. One of the goals of this research was to automate lattice extraction for near-regular texture, which is only partially achieved with the results presented in this thesis. A specific problem is the reliance on a user-selected sample texton which is used to locate subsequent textons in the image. A method for algorithmically locating such a sample texton would increase the automation of our result. Though many advances in image segmentation and object recognition hold promise in automatically locating such a texton, the generality of near-regular texture implies that this problem will remain unsolved for the foreseeable future.

A similar problem arises when the user must blindly select the sample texton that yields the best results for a texture sample. In some cases the choice of sample texton greatly affects the texton locating result. However, it is not known a priori which sample texton will produce the best result. If, given a texture sample, a method of evaluating the likelihood of success of texton locating was available, we could further automate and further improve our algorithm.

Image features have improved the texton locating procedure in special cases. However, a large avenue for future work is in evaluating and choosing image features suitable for texton locating. For a given texture, a certain set of features may improve the texton locating result considerably, while for a separate texture the same set of features may decrease the texton locating result. Applying

and evaluating new image features is a rich area for future work.

Frequency-based repair may improve or reduce the texton locating success rate. In some cases the procedure can repair a result to perfectly locate every texton in an image, while in other cases the procedure can introduce errors into an already perfect result. A method for evaluating when frequency-based repair should be applied would increase the automation and effectiveness of our lattice extraction method.

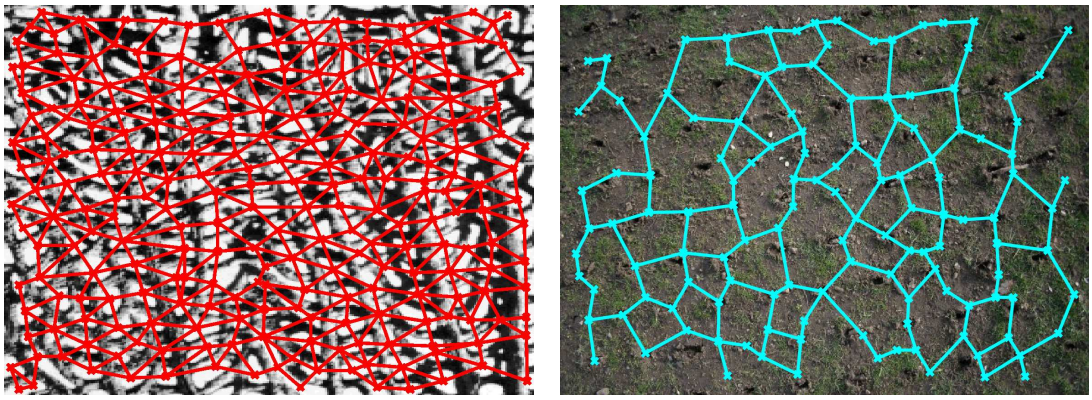
In addition to the texton locating algorithm and its enhancements, we have provided two methods for constructing a lattice from texton locations. The first method is an ad-hoc procedure designed specifically for the problem at hand. This procedure takes advantage of the structure of most near-regular textures in assuming that texton placement is determined by two translation vectors. The second graphing procedure handles those textures which do not meet this criteria, solving the lattice construction problem in the general case.

Reliance on textons being placed using two translation vectors limits the applicability of our ad-hoc lattice creation algorithm. More specifically, the range of textures that allow for successful lattice extraction is limited. On the other hand, the procedure is able to ignore incorrect texton locations, and therefore contains an implicit method for removing unwanted texton locations from the result, which we have found produces a correct lattice for a large variety of near-regular texture [62].

The graph-theoretic lattice construction algorithms are able to construct a sensible lattice from arbitrary input. However, a careful parameter selection for the algorithm is important as the parameter choice affects the output result; an automated method for choosing the appropriate parameter for each specific texture would be a worthwhile contribution. Furthermore, the graph-theoretic algorithm has no method for ignoring incorrect texton locations, and so any errors in texton locating propagate into the lattice construction process.

Another avenue of future work lies in combining the two lattice construction methods. Specifically, texton locations could be discovered using the ad-hoc lattice construction method of extending translation vectors from textons which have already been located, and β graphs could be used to determine if an edge should join each texton location found. Combining the two lattice construction methods may result in a new method which correctly ignores incorrect texton locations and yet creates a sensible lattice for arbitrary input.

A general problem we discovered during this research is the need for a method of evaluating the quality of a lattice. For future work, an automatic lattice quality measure would be immensely useful in automating lattice extraction for near-regular texture. Using the procedures provided in this thesis along with a quality measure would allow us to locate textons using combinations of several image features along with frequency-based repair, evaluating the texton locating result for each combination, and choosing the combination best suited for the input texture. Through



(a) Failure due to an irregularity of the input texture. (b) Failure due to high frequency detail within textons.

Figure 7.1: There are two cases where lattice extraction fails: irregularity of the input texture and high frequency detail within textons.

the experience gained in this thesis, we make the claim that an adequate lattice will be planar, and that in the majority of near-regular texture each texton will be connected to four neighbours. These criteria combined with an analysis of the per-pixel deviation within each region defined by the lattice may be used as a starting point for developing a lattice quality measure.

Perceptually sensible is an imprecise term, and we have provided no exact definition in this thesis. An area of future work, closely related to measuring lattice quality, lies in creating a formal specification of perceptually sensible. A starting point for this research may be creating an energy function containing terms related to edge length, angle between adjacent edges, and the size of lattice regions. Minimizing such an energy function may amount to discovering a quality lattice.

A last area of future work lies in removing any failure cases from our algorithm. In our database of 100 near-regular textures, we observed 10 textures where we could not correctly extract a lattice. Of these 10 failures, 3 may be classified as irregular. Although our procedure works for some irregular textures, irregular texture in general is outside the scope of this thesis. Recalling the statement of our problem, we wish to extract a lattice from near-regular texture, and we do not consider failures due to texture irregularity as failures of our algorithm. Figure 7.1(a) depicts a failed lattice extraction due to a texture being irregular. The 7 remaining failure cases occur when textures possess high frequency detail within textons. In these cases, correlating a candidate texton with other textons in the image proves difficult since all components of each texton must be similarly aligned. An area of future work is in allowing our algorithm to correctly extract a lattice from texture containing textons with high frequency detail. Figure 7.1(b) depicts a failed lattice extraction due to a texture having textons which possess high frequency detail.

Applications of our work are varied. The problem was initially approached as a means of further automating the near-regular texture synthesis algorithm of Liu et al. [45]. We have also extended

our lattice extraction method to handle some textures which Liu et al.'s algorithm has difficult synthesizing. Specifically, we are able to extract a lattice from textures with high G score, implying that the texture is irregular. Given this limitation of Liu et al.'s algorithm, we have suggested a few applications of an irregular texture lattice: texture replacement [41], grouping of repeated scene elements [36], object recognition, shape from texture, texture classification, texture compression, and texture blending.

Our algorithm can successfully extract a lattice from a wide variety of near-regular texture [62]. The procedure does have drawbacks, which have been outlined above, but these drawbacks are not significant enough to limit the applicability of our result. The generality of near-regular texture ensures that the problem of lattice extraction will be difficult to completely solve, and we believe that our result is a step in the right direction.

References

[1] M. Ashikhmin. Synthesizing natural textures. In *Symposium on Interactive 3D Graphics*, pages 217–226, 2001.

[2] S. Baheerathan, F. Albregtsen, and H.E. Danilesen. New texture features based on the complexity curve. *Pattern Recognition Society*, 23:605 – 618, 1999.

[3] S. Belongie, J. Malik, and J. Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *Proceedings of NIPS*, pages 831 – 837, November 2000.

[4] S. Belongie, J. Malik, and J. Puzicha. Matching shapes. In *Proceedings of ICCV*, pages 454 – 463, July 2001.

[5] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. on PAMI*, 24(24), April 2002.

[6] D. Blostein and N. Ahuja. A multi-scale region detector. *Computer Graphics Vision and Image Processing*, 45(1):22 – 41, January 1989.

[7] J.S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 361–368, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[8] R.N. Bracewell. *The Fourier Transform and its Applications*. McGraw-Hill Book Company, 1978.

[9] P. Brodatz. *Textures: A Photographic Album for Artists and Designers*. Dover, New York, 1966.

[10] J. Canny. A computational approach to edge detection. *IEEE Transactions on Patteran Analysis and Machine Intelligence*, 8(6), November 1986.

[11] D. Chetverikov. Pattern regularity as a visual key. *Image and Vision Computing*, 18:975 – 985, 2000.

[12] D. Chetverikov. Fundamental structural properties of textures. *D.Sc. Dissertation (MTA SZTAKI - Budapest)*, 2002.

[13] CMU. CMU NRT database. Maintained by James Hays at the CMU Graphics Lab. `http://http://graphics.cs.cmu.edu/data/texturedb/gallery/`, 2006.

[14] M.F. Cohen, J. Shade, S. Hiller, and O. Deussen. Wang tiles for image and texture generation. *ACM Transactions on Graphics*, 22(3):287–294, 2003.

[15] J-M. Dischler, K. Maritaud, B. Lévy, and Djamchid Ghazanfarpour. Texture particles. *Computer Graphics Forum*, 21(3), 2002.

[16] K. Djado, R. Egli, and F. Deschenes. Extraction of a representative tile from a near-periodic texture. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 331–337, Dunedin, New Zealand, 2005. ACM Press.

[17] A.A. Efros and W.T. Freeman. Image quilting for texture synthesis and transfer. *Proceedings of SIGGRAPH 2001*, pages 341–346, August 2001.

[18] A.A. Efros and T.K. Leung. Texture synthesis by non-parametric sampling. *ICCV '99*, September 1999.

[19] David A. Forsyth. Shape from texture without boundaries. *Proceedings of the ECCV*, 3:225–239, 2002.

[20] G.L. Gimel'farb. Modeling image textures by Gibbs random fields. *Pattern Recognition Letters*, 20(11 - 13):1123 – 1132, 1999.

[21] G.L. Gimel'farb, L.J. Van Gool, and A. Zalesny. To FRAME or not to FRAME in probabilistic texture modelling? In *ICPR (2)*, pages 707–711, 2004.

[22] L.V. Gool, P. Dewaele, and A. Oosterlinck. Survey: Texture analysis annotations. *Computer Vision, Graphics, and Image Processing*, 29:336 – 357, 1985.

[23] C-E. Guo, S-C. Zhu, and Y.N. Wu. Modeling visual patterns by integrating descriptive and generative methods. *International Journal of Computer Vision*, 53(1):5 – 29, 2003.

[24] L.G.C. Hamey. Computer analysis of regular repetitive textures. In *Proceedings of DARPA Image Understanding Workshop*, pages 1076 – 1088, May 1989.

[25] R.M. Haralick. Statistical and structural approaches to texture. *Proceedings of the IEEE*, 67:786 – 804, 1979.

[26] D.J. Heeger and J.R. Bergen. Pyramid-based texture analysis/synthesis. In *SIGGRAPH 2003*, pages 229–238, 1995.

[27] A. Hertzmann, C.E. Jacobs, N. Oliver, B. Curless, and D.H. Salesin. Image analogies. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 327–340, New York, NY, USA, 2001. ACM Press.

[28] T-I. Hsu, A.D. Calway, and R. Wilson. Texture analysis using the multiresolution Fourier transform. In *Proceedings of the 8th Scandinavian Conference on Image Analysis*, pages 823 – 830. IAPR, July 1993.

[29] B. Julesz. Experiments in the visual perception of texture. *Science America*, 232:34 – 43, April 1975.

[30] B. Julesz, E. Gilbert, and J.D. Victor. Visual discrimination of textures with identical third-order statistics. *Biological Cybernetics*, 31:137 – 140, 1978.

[31] D.G. Kirkpatrick and J.D. Radke. A framework for computational morphology. *Machine Intelligence and Pattern Recognition*, 2:217 – 248, 1985.

[32] V. Kwatra, A. Schdl, I. Essa, G. Turk, and A. Bobick. Graphcut textures: Image and video synthesis using graph cuts. *ACM Transactions on Graphics, SIGGRAPH 2003*, 22(3):277–286, July 2003.

[33] MIT Media Lab. VisTex: Vision texture database. Maintained by the Vision and Modeling group at the MIT Media Lab. `http://whitechapel.media.mit.edu/vismod/`, 1995.

[34] P.M. Lankford. Regionalization: theory and alternative algorithms. *Geographical Analysis*, 1:196 – 212, 1969.

[35] J.G. Leu. On indexing the periodicity of image textures. *IVC*, 19(13):987–1000, November 2001.

[36] T.K. Leung and J. Malik. Detecting, localizing and grouping repeated scene elements from an image. In *ECCV '96: Proceedings of the 4th European Conference on Computer Vision-Volume I*, pages 546–555. Springer-Verlag, 1996.

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

[38] H-C. Lin, L-L. Wang, and S-N. Yang. Automatic determination of the spread parameter in Gaussian smoothing. *Pattern Recognition Letters*, 17(12):1247 – 1252, 1996.

[39] H-C. Lin, L-L. Wang, and S-N. Yang. Extracting periodicity of a regular texture based on autocorrelation functions. *Pattern Recognition Letters*, 18(5):433 – 443, 1997.

[40] H-C. Lin, L-L. Wang, and S-N. Yang. Regular-texture image retrieval based on texture-primitive extraction. *Image Vision Computing*, 17(1):51 – 63, 1999.

[41] W-C. Lin and Y. Liu. NRT-based texture replacement in real videos. *Technical Sketch: SIGGRAPH 2005*, 2005.

[42] F. Liu and R.W. Picard. Periodicity, directionality, and randomness: Wold features for image modeling and retrieval. *IEEE Transactions on PAMI*, 18(7), July 1996.

[43] Y. Liu, R. Collins, and Y. Tsin. A computational model for periodic pattern perception based on frieze and wallpaper groups. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(3):354 – 371, March 2004.

[44] Y. Liu and W-C. Lin. Deformable texture: The irregular-regular-irregular cycle. In *Proceedings of the 3rd International Workshop on Texture Analysis and Synthesis (Texture 2003)*, October 2003.

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

[46] Y. Liu, Y. Tsin, and W-C. Lin. The promise and perils of near-regular texture. *International Journal of Computer Vision*, 62(1-2):145 – 159, April 2005.

[47] J. Malik, S. Belongie, T. Leung, and J. Shi. Contour and texture analysis for image segmentation. *Internation Journal of Compututer Vision*, 43(1):7–27, 2001.

[48] J. Malik, S. Belongie, J. Shi, and T. Leung. Textons, contours and regions: Cue integration in image segmentation. In *ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2*, page 918. IEEE Computer Society, 1999.

[49] B.S. Manjunath, P. Wu, S. Newsam, and H. Shin. A texture descriptor for browsing and image retrieval. *International Communications Journal*, 16:33 – 43, September 2000.

[50] T. Matsuyama, S.I. Miura, and M. Nagao. Structural analysis of natural textures by Fourier transformation. *CVGIP*, 24(3):347–362, December 1983.

[51] D.W. Matula and R.R. Sokal. Properties of Gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical Analysis*, 12:205 – 222, 1980.

[52] A. Nealen and M. Alexa. Hybrid texture synthesis. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 97–105, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[53] A. Nicoll, J. Meseth, G. Muller, and R. Klein. Fractional Fourier texture masks: Guiding near-regular texture synthesis. *EUROGRAPHICS 2005*, 24(3), 2005.

[54] G. Oh, S. Lee, and S. Yong Shin. Fast determination of textural periodicity using distance matching function. *Pattern Recognition Letters*, 20(2):191–197, 1999.

[55] N. Otsu. A threshold selection method from the gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-9:62–67, January 1979.

[56] J. Parkkinen, K. Selkainaho, and E. Oja. Detecting texture periodicity from the cooccurrence matrix. *Pattern Recognition Letters*, 11:43 – 50, 1990.

[57] K. Perlin. An image synthesizer. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 287–296, New York, NY, USA, 1985. ACM Press.

[58] J. Portilla and E.P. Simoncelli. A parametric texture model based on joint statistics of complex wavelet coefficients. *IEEE Transactions on Image Processing*, 12(11):1338 – 1351, November 2000.

[59] E. Praun, A. Finkelstein, and H. Hoppe. Lapped textures. In *Proceedings of ACM SIGGRAPH 2000*, pages 465–470, July 2000.

[60] F. Schaffalitzky and A. Zisserman. Geometric grouping of repeated elements within images. In *Shape, Contour and Grouping in Computer Vision*, pages 165–181. Springer-Verlag, 1999.

[61] E.P. Simoncelli and W.T. Freeman. The steerable pyramid: A flexible architecture for multi-scale derivative computation. In *ICIP '95: Proceedings of the 1995 International Conference on Image Processing (Vol. 3)-Volume 3*, page 3444, Washington, DC, USA, 1995. IEEE Computer Society.

[62] K. Sookocheff and D. Mould. One-click lattice extraction from near-regular texture. In *GRAPHITE '05: Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 265–268, Dunedin, New Zealand, 2005. ACM Press.

[63] V.V. Starovoitov, S-Y. Jeong, and R-H. Park. Texture periodicity detection: Features, properties, and comparisons. *IEEE Transactions on Systems, Man, and Cybernetics*, 28(6):839 – 849, November 1998.

[64] G.T. Toussaint. Pattern recognition and geometrical complexity. *Proceedings of the 5th International Conference on Pattern Recognition*, pages 1324 – 1347, Dec 1980.

[65] G.T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern Recognition*, 12:261 – 268, 1980.

[66] T. Tuytelaars, A. Turina, and L.V. Gool. Non-combinatorial detection of regular repetitions under perspective skew. *IEEE Transactions on PAMI*, 25(4):418 – 432, April 2003.

[67] Queensland University. MeasTex: Image texture database and test suite. Maintained by the CCSIP at Queensland University. `http://www.cssip.uq.edu.au/meastex/meastex.html`, 1997.

[68] F.M. Vilnrotter, R. Nevatia, and K.E. Price. Structural analysis of natural textures. *IEEE Transactions on PAMI*, 8(1), January 1986.

[69] B. Wandell. *Foundations of Vision*. Sinauer Associates Inc., 1995.

[70] L-Y. Wei and M. Levoy. Fast texture synthesis using tree-structured vector quantization. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 479–488, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[71] Q. Wu and Y. Yu. Feature matching and deformation for texture synthesis. *SIGGRAPH '04: ACM Transactions on Graphics*, 23(3):364–367, 2004.

[72] S. Zelinka and M. Garland. Towards real-time texture synthesis with the jump map. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 99–104, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[73] S. Zelinka and M. Garland. Interactive texture synthesis on surfaces using jump maps. In *EGRW '03: Proceedings of the 14th Eurographics workshop on Rendering*, pages 90–96, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[74] J. Zhang, K. Zhou, L. Velho, B. Guo, and H-Y. Shum. Synthesis of progressively-variant textures on arbitrary surfaces. *ACM Transactions on Graphics*, 22(3):295 – 302, 2003.

[75] D. Zhou and G. Gimel'farb. Model-based estimation of texels and placement grids for fast realistic texture synthesis. In *Proceedings of the 3rd International Workshop on Texture in Conjunction with ICCV 2003*, pages 119 – 123, 2003.

[76] S-C. Zhu, C. Guo, Y. Wang, and Z. Xu. What are textons? *International Journal of Computer Vision*, 62, 2005.

[77] S.C. Zhu, Y.N. Wu, and D. Mumford. Filters, random fields, and maximum entropy (FRAME): Towards a unified theory for texture modeling. *International Journal of Computer Vision*, 27:1 – 20, 1998.

Appendix A

Image Processing Techniques

A.1 Canny Edge Detection

The Canny edge detection algorithm [10] contains five steps. First, filter out noise from the original image by convolution with a Gaussian filter. Second, compute the image gradient using the Sobel operator described in Section A.5. Third, compute edge direction of each pixel using the Sobel method discretized into one of four directions: north, south, east, and west. Fourth, use non-maximal suppression to create a eliminate thick edges. Fifth, use hysteresis to eliminate broken edges.

A.2 Laplacian-of-Gaussian Filter and the Second Derivative

To approximate the second derivative of an image a Laplacian-of-Gaussian filter is used. The Laplacian, denoted $\bigtriangledown^2$, is defined for continuous functions as

$$\bigtriangledown^2 g(x,y) = \frac{\partial^2 g(x,y)}{\partial x^2} + \frac{\partial^2 g(x,y)}{\partial y^2}$$

and calculates the magnitude of the second derivative. However, the Laplacian function applied to images is sensitive to noise. Smoothing the image with a Gaussian function,

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}},$$

removes noise from the image. Furthermore, both the Gaussian and the Laplacian functions are applied to an image using convolution, which is a linear operation. Because of this linearity the two functions can be combined before being applied to the image without compromising the result. The resultant function is called the Laplacian-of-Gaussian,

$$h(x,y) = \frac{(x^2+y^2-2\sigma^2)G(x,y)}{2\pi\sigma^6 \sum_x \sum_y G(x,y)}.$$

We set the size of our Laplacian-of-Gaussian filter to a constant 5×5, with $\sigma = 0.5$.

A.3 Morphological Shrinking

Morphological shrinking reduces each connected region in a binary image to a single point using multiple applications of erosion.

Erosion is performed by filtering a binary image with the structuring element

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}.$$

For each image location matching the structuring element we populate a new image with a single value 1. This operation eliminates values of 1 from a regions boundary, making the region smaller. To shrink each region to a single point we continually apply erosion, noting when only one pixel remains in a region. When each region is reduced to a single pixel, morphological shrinking is complete.

A.4 Non-Maximal Suppression

Non-maximal suppression eliminates an image pixel p if there is any pixel in p's neighbourhood with value larger than p. p is also eliminated if the neighbourhood around p intersects with the image boundary.

Non-maximal suppression is sensitive to neighbourhood choice, and so care must be taken to choose an appropriate neighbourhood. In our algorithm neighbourhood size is equal to the user-selected sample window size.

A.5 Sobel Edge Detection and Image Gradient

The Sobel operator performs a gradient measurement on an image, emphasizing regions of high frequency which correspond to edges. The filter can be used to approximate the image gradient, as well as to detect image edges.

The Sobel operator consists of the pair of 3×3 filters

$$h_1 = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix}, h_3 = \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{pmatrix}.$$

These filters have high response to vertical and horizontal edges, respectively. The filters are applied separately to the image and return a gradient measurement in each direction. For an input image g, we represent the horizontal gradient as $\frac{\partial g(x,y)}{\partial x}$ and the vertical gradient as $\frac{\partial g(x,y)}{\partial y}$. Gradient magnitude at each pixel position (x, y) is thus calculated as

$$|\bigtriangledown g(x,y)| = \sqrt{\left(\frac{\partial g(x,y)}{\partial x}\right)^2 + \left(\frac{\partial g(x,y)}{\partial y}\right)^2}$$

using the Pythagorean theorem.

To detect edges in the image, we isolate maxima in the gradient image using a threshold; values in the gradient greater than the threshold are reported as image edges. We use Otsu's method [55] to select the threshold.

The output of the Sobel operator can also be used to compute edge direction. Given the image gradient in the x and y directions, edge direction is computed as

$$\tan^{-1}\left(\frac{y}{x}\right).$$

A.6 Unsharp Contrast Enhancement Filter

The unsharp contrast enhancement filter, also known as the unsharp mask, is a well-known technique for increasing the contrast of a photograph by sharpening image edges. In our implementation, we use Matlab's built-in unsharp filter, which is computed as the negative of the Laplacian with parameter α. We set α to 0.2. The filter is given by

$$h = \frac{1}{\alpha + 1}\begin{pmatrix} -\alpha & \alpha - 1 & -\alpha \\ \alpha - 1 & \alpha + 5 & \alpha - 1 \\ -\alpha & \alpha - 1 & -\alpha \end{pmatrix}.$$

Appendix B

Colour Spaces

B.1 HSV Colour Space

HSV stores individual values of hue, saturation, and value. Hue varies from red through yellow, green, cyan, blue, and magenta, specifying a colour. Saturation measures the amount of white in the colour; a fully saturated colour contains no white and an unsaturated colour is a shade of gray. Saturation is often referred to as the vibrancy or purity of the colour. Finally, value measures colour brightness.

B.2 RGB Colour Space

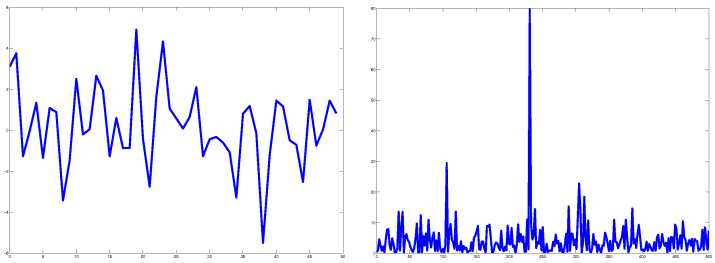
RGB stores individual values of red, green, and blue. RGB is an additive colour model where combinations of red, green, and blue are combined to form a colour.

B.3 YIQ Colour Space

YIQ stores a luminance value (Y) with two chrominance values (I and Q). The luminance records the brightness of a colour in an image. Another way of looking at it is that luminance alone provides gray-scale information for an image. The two chrominance values I and Q roughly correspond to the amounts of blue and red in the image.

Appendix C

The Fourier Transform



(a) A signal with added random noise.

(b) The signals frequency based representation.

Figure C.1: The Fourier transform projects a function varying over time to a function varying over frequency. Applying a threshold to the frequency representation and inverse Fourier transforming the result removes unwanted frequencies from the image.

The Fourier transform calculates the series expansion of a function in terms of its sine and cosine components. For a continuous function $f(x)$, the Fourier transform is defined as

$$F(s) = \int_{-\infty}^{\infty} f(x) e^{-i2\pi xs} dx.$$

The Fourier transform returns complex numbers, but is typically represented by its magnitude and phase rather than its real and imaginary parts. The magnitude, commonly referred to as the power spectrum, is defined as

$$M_F = \sqrt{\Re(F)^2 + \Im(F)^2},$$

where $\Re(x)$ and $\Im(x)$ return, respectively, the real and imaginary factors of a complex number x. The power spectrum provides the quantity of each frequency component in the function. On the other hand, the phase provides the position where the frequency component lies within the function. The phase is defined as

$$P_F = \arctan\left(\frac{\Im(F)}{\Re(F)}\right).$$

To interpret the preceding formulae, it is advantageous to visualize the result in terms of a function varying over time. The Fourier transform projects a function in the time domain to the frequency domain, which is conveniently analyzed using the power spectrum. As an example, Figure C.1(a) depicts a function varying in time. The function is a sinusoid with added random noise. It is difficult to visually ascertain the frequency of the sinusoid from this function. However, the

power spectrum of the function, depicted in Figure C.1(b) contains a distinct peak showing the prominent frequency in the image.

The inverse Fourier transform is the reverse of the Fourier transform, projecting a function in the frequency domain to the time domain. Mathematically, we can apply the inverse Fourier transform to $F(s)$ using the formula

$$f(x) = \int_{-\infty}^{\infty} F(s) e^{i2\pi xs} ds.$$

As a visual example, given the function in Figure C.1(b), the inverse Fourier transform reproduces the original signal in Figure C.1(a).

Appendix D

Program Listings

Listing D.1: The lattice extraction algorithm.

```
function [edges] = latticeExtract(im)
% Perform the lattice extraction algorithm
%
% Input:
%   im      The image to extract a lattice from
%
% Output:
%   edges   The edge list representing the extracted lattice

% Mean centre the image
imm = meancentre(im);

% Compute image features here (optional)

% Display the input image
figure;
imagesc(im);
colormap gray;

% Select a sample
sample = selectsample(imm);

% Correlation
surface = correlate(sample, imm);

% Smooth (sharpens upon completion)
surface = smooth(surface);

% NMS
nms = nonmaximalsuppression(surface, [size(sample,1) size(sample,2)]);

% Frequency-based repair (optional)
nms = repair(nms, sample);
```

```
% Get the centres
[Y X] = find(nms ~= 0);

% Graph
hold on;
edges = relativeNeighbourhoodGraph([Y X]);
plotgraph([Y X], edges);
```

Listing D.2: Selection of a sample from the input image.

```
function sample = selectsample(im, fignum)
% Select a rectangle ([xmin ymin width height]) from the figure
% passed in to the function.  If no argument is given, the last
% active figure is used.
%
% Input:
%   im       The image to select a rectangle from
%   fignum   The figure where im is drawn (optional)
%
% Output:
%   sample   A rectangle ([xmin ymin width height])

% Check if we are given a figure number
if (nargin == 2)
    figure(fignum);
end

% Get a rectangle from the figure
rect = getrect;
% Round the result
rect = round(rect)
% Extract a window from the image selected by the user
sample = im(rect(2):rect(2)+rect(4)-1, rect(1):rect(1)+rect(3)-1,:);
```

Listing D.3: Correlation of an image window with an image.

```
function surface = correlate(window,im)
% Correlate the contents of window with the contents of the image.
%
% Inputs:
```

```
%   window  The window to correlate with
%   im      The image to correlate with
%
% Outputs:
%   surface The correlation surface.

% Correlate each image plane
for (i = 1:size(im,3))
    surface(:,:,i) = filter2(window(:,:,i), im(:,:,i));
end

% Average the results to get a single surface
surface = mean(surface, 3);
```

Listing D.4: Lin's smoothing algorithm [39].

```
function smoothed = smooth(surface, windowSize)
% Smooths the surface parameter using the automatic Gaussian filter choice
% given by Lin et al. 1997.  Sharpens once before returning.
%
% Inputs:
%   surface     The surface to smooth
%   windowSize  The size of the Gaussian window (optional, default is 5x5)
%
% Outputs:
%   smoothed    The smoothed surface

% Check if window size is specified
if (nargin < 2)
    windowSize = 5;
end

% Set the initial parameters
sigma = 1;
p0 = Inf;

% Keep looping
iteration = 0;
while (iteration <= 10)
    % Create a Gaussian filter
```

```
g = fspecial('gaussian', windowSize, sigma);

% Create an image of the Gaussian filter padded with zeros
gCorr = zeros(size(surface));
gCorr(1:size(g,1),1:size(g,2)) = g;

% Fourier transform each image
f1 = fft2(gCorr, size(surface,1), size(surface,2));
f2 = fft2(surface, size(surface,1), size(surface,2));

% Multiply the transforms
fResult = f1 .* f2;

% Inverse transform the result
smoothed = ifft2(fResult);

% Circularly shift the result back to the proper location
% This value depends on the size of the window used in filtering
smoothed = circshift(smoothed, [-floor(windowSize/2) -
    ... floor(windowSize/2)]);

% Extract the number of peaks in the surface
peaks = find(imregionalmax(smoothed) == 1);
p1 = length(peaks);

% Check if the peaks have increased or are less than or equal to zero
if ((p1 >= p0) || (p1 <= 1))
    break;
end

% Get the peaks actual locations of peaks from the surface
[I J] = ind2sub(size(surface), peaks);

% Build a kd tree based on the peaks
kd = kdTreeInit([I J]);

% Find the maximin distance between points
[n m delta] = kdTreeMaxiMin([I J], kd);
```

```
    % Update sigma using the new distance
    sigma = sqrt(delta);

    % Update p0
    p0 = p1;

    % Increase the iteration count
    iteration = iteration + 1;
end

% Sharpen the image as a final step
smoothed = filter2(fspecial('unsharp'), smoothed);
```

Listing D.5: The non-maximal suppression algorithm.

```
function Y = nonmaximalsuppression(X, windowSize)
% Compute the non-maximal suppression of the image X with a sliding window
% of size windowSize(1) by windowSize(2).  windowSize is an array of size
% two, first element is width and second element is height.
%
% Inputs:
%   X           The image to suppress
%   windowSize  The size of the sliding window (default = 5).
%
% Outputs:
%   Y           The suppressed image

% Initialize the output
Y = X;

% Set the default window size
if (nargin < 2)
    windowSize(1) = 5;
    windowSize(2) = 5;
end

% Make sure that the window size is odd
if (mod(windowSize(1), 2) == 0)
    windowSize(1) = windowSize(1) - 1;
end
```

```
if (mod(windowSize(2), 2) == 0)
    windowSize(2) = windowSize(2) - 1;
end

% Initialize some variables
window = zeros(windowSize);
middle = ceil(windowSize/2);

% Pass the sliding window over the image
for (i = 1:size(X,1))
  for (j = 1:size(X,2))
        % Test if we are inside the valid region
        if ((i >= middle(1)) && (i <= (size(X,1) - middle(1) + 1))
                ... && (j >= middle(2))
                ... && (j <= (size(X,2) - middle(2) + 1)))
          % Gather the current window
          window = X(i-middle(1)+1:i+middle(1)-1,
                ... j-middle(2)+1:j+middle(2)-1);

      % Gather the centre pixel
          midval = window(middle(1),middle(2));

          % If any values in the window are greater than the centre pixel,
          % delete the centre pixel
          if (max(max(window)) ~= midval)
              Y(i,j) = 0;
      end
    else
      % We don't have a proper window so set the color to black
      Y(i,j) = 0;
    end
  end
end

% Shrink until each point is exactly one
Y = bwmorph(Y, 'shrink', Inf);
```

Listing D.6: Ad-hoc lattice construction.

```
function edges = graph(centres, startIndex, primary, secondary, partial)
```

```
% Connect the centres found during texton locating into a lattice.  The
% construction begins at startIndex, and extends out using the primary and
% secondary translation vectors to find new texton locations to add to the
% lattice.
%
% Inputs:
%   centres         2xn array of texton locations
%   startIndex      The intial texton location
%   primary         Primary translation vector
%   secondary       Secondary translation vector
%   partial         Correlation surface
%
% Outputs:
%   edges           The graph as an edge list

% Keep a list of nodes already processed, edge number, node degree and
% known centres
processed = zeros(size(centres,1),1);
edges = zeros(1,2);
edgeNumber = 0;
degree = zeros(size(centres,1), 1);
knownCentres = zeros(size(centres,1),1);
boundary = zeros(size(centres,1),1);

% Create a priority queue to hold the nodes of the graph
pq = pqInit();

% Create and insert the first node of the queue
point.location = startIndex;
point.priority = 1;
pq = pqInsert(point, pq);

% Check which vector lies closer to the x-axis and then:
% Compute a threshold based on the size of the major and minor axes of the
% ellipse defined by the primary and secondary translation vectors
if (abs(secondary(2)) > abs(primary(2)))
    xAxis = dist(0, 0, secondary(1), secondary(2));
    yAxis = dist(0, 0, primary(1), primary(2));
else
```

```
    yAxis = dist(0, 0, secondary(1), secondary(2));
    xAxis = dist(0, 0, primary(1), primary(2));
end

% Divide x and y axis by three to create a threshold
% The yAxis is the ellipse axis lying on the y axis (x = 0)
yAxis = floor(yAxis/3);
xAxis = floor(xAxis/3);

% Loop until all points have been processed
while (pq.curSize ~= 0)
    % Get the next node to process
    [point pq] = pqRemove(pq);

    % Process this node if it hasn't already been processed and it's not a
    % boundary node
    if ((processed(point.location) == 0) &&
        ... (boundary(point.location) == 0))
        % Mark this region as processed
        processed(point.location) = 1;

        % Get the i,j location of the centre
        I = centres(point.location,1);
        J = centres(point.location,2);

        % Get the translation vectors
        Py = primary(1,1); Px = primary(1,2);
        Sy = secondary(1,1); Sx = secondary(1,2);

        % Get the locations of this node
        % plus the translational vectors
        for (i = 1:4)
            % Compute iNew jNew depending on the direction
            switch (i)
                case 1,
                    % (I,J) + (Px,Py)
                    iNew = I + Py;
                    jNew = J + Px;
                case 2,
```

```
            % (I,J) - (Px,Py)
            iNew = I - Py;
            jNew = J - Px;
        case 3,
            % (I,J) + (Sx,Sy)
            iNew = I + Sy;
            jNew = J + Sx;
        case 4,
            % (I,J) - (Sx,Sy)
            iNew = I - Sy;
            jNew = J - Sx;
    end

    % Check if the location we are connected to lies outside the
    % boundary
    if ((iNew <= 1) || (iNew >= size(partial,1)) || (jNew <= 1)
        ... || (jNew >= size(partial,2)))
        boundary(point.location) = 1;
    else
        ellipseIndex = 1;    % The number of points within ellipse
        b = yAxis;           % The Y axis treshold
        a = xAxis;           % The X axis threshold

        if (exist('withinEllipse') == 1)
            clear withinEllipse;
        end

        % Check if each point lies within an ellipse
        % defined by the primary and secondary
        % translation vectors.
        done = 0;
        while (true)
            % Specify some ellipse parameters
            h = jNew; k = iNew;

            for (j = 1:size(centres,1))
                % Get a point to compare to
                iComp = centres(j,1);
                jComp = centres(j,2);
```

```
            % Define ellipse parameters
            x = jComp; y = iComp;

            % Check if this point lies within the ellipse
            if (ellipse(x,y,h,k,a,b) <= 1)
                if (j ~= point.location)
                    withinEllipse(ellipseIndex,1) = j;
                    ellipseIndex = ellipseIndex + 1;
                end
            end
        end

        % Check if we connected to something
        if (exist('withinEllipse') == 0)
            % No?  Increase the threshold
            b = b + 0.25;
            a = a + 0.25;
        else
            % Yes?  Break from the loop
            break;
        end

        % Check if this ellipse is searching outside the
        % boundary.
        if (((h-b) <= 1) || ((h+b) >= size(partial,1)) ||
            ... ((k-a) <= 1) || ((k+a) >= size(partial,2)))
            done = 1;
            break;
        end
    end

    % Check if we have exceeded the boundary in our search
    if (~done)
        % Check if one of these is a known texton centre
        known = 0;
        for (j = 1:size(withinEllipse,1))
            if (knownCentres(withinEllipse(j)) == 1)
                known = 1;
```

```
        knownLocation = withinEllipse(j);
    end
end

% Check if a point is a known centre do that
if (known)
    % Yes?  Set this as the next location
    bestIdx = knownLocation;
else
    % No?
    % Find the highest correlation location
    maxCorrelation = 0;
    for (j = 1:size(withinEllipse,1))
        % Get the coordinates of a point
        iLoc = centres(withinEllipse(j),1);
        jLoc = centres(withinEllipse(j),2);

        % Obtain the correlation value
        correlation = partial(iLoc, jLoc);

        % Check if we have a maximum
        if (correlation > maxCorrelation)
            maxCorrelation = correlation;
            bestIdx = withinEllipse(j);
        end
    end
end

% bestIdx = highest correlation value

% Make sure that the degree of the current
% point is less than four
if (degree(point.location) < 4)
    % Check if this edge is already part
    % of the graph
    done = 0;
    for (k = 1:size(edges,1))
        if (isequal(edges(k, :),
            ... [point.location bestIdx])
```

```
                    ... || isequal(edges(k, :),
                    ... [bestIdx point.location]))
                    done = 1;
                end
            end

            % Is this edge already part of the graph
            if (~done)
                % No?
                % Store this edge as part of the graph
                edges(edgeNumber + 1,:) =
                    ... [point.location bestIdx];
                edgeNumber = edgeNumber + 1;

                % Increase the degree of each of these nodes
                degree(point.location) =
                    ... degree(point.location) + 1;
                degree(bestIdx) = degree(bestIdx) + 1;

                % Label these as known centres
                knownCentres(point.location) = 1;
                knownCentres(bestIdx) = 1;

                % If this is not a boundary node
                % add the new centres
                if (boundary(point.location) == 0)
                    % Create the new point to add
                    newPoint.location = bestIdx;
                    newPoint.priority = point.priority + 1;

                    % Add the new centre to the queue
                    pq = pqInsert(newPoint, pq);
                end
            end
        end
    end
end
end
```

```
end

function z = ellipse(x,y,h,k,a,b)

z = (((x-h)\^2)/a\^2) + (((y-k)\^2)/b\^2);
```

Listing D.7: Beta skeleton lattice construction.

```
function edges = betaGraph(centres, betaV)
% Construct a beta graph of the texton centres given.
% Use the brute force O(n^3) algorithm.
%
% Input:
%   centres     The texton centres to graph
%   beta        The beta value for graph construction
%
% Output:
%   edges       The edges of the relative neighbourhood graph

% Calculate whether a point lies within the region defined by
% and the two input points
edgeNumber = 0;
edges = zeros(1,2);
for (i = 1:size(centres,1))
    for (j = 1:size(centres,1))
        if (i == j)
          break;
        end

        draw = 1;
        for (x = 1:size(centres,1))
            if ((x == i) || (x == j))
                % Do nothing
            else
                % Check if x is inside the beta region
                inside = insideBeta(centres(x,:),centres(i,:),
                    ... centres(j,:),betaV);
                if (inside == 1)
                    draw = 0;
                    break;
```

```
            end
        end
    end

    if (draw)
        % Check if this edge is already part of the graph
        done = 0;
        for (k = 1:size(edges,1))
            if (isequal(edges(k, :),[i j])
                ... || isequal(edges(k, :), [j i]))
                done = 1;
            end
        end

        if (~done)
            % Create an edge
            edgeNumber = edgeNumber + 1;
            edges(edgeNumber,:) = [i j];
        end
    end
  end
end
```

Listing D.8: Frequency-based repair. Uses Program D.9 and D.10.

```
function rep = repair(nms, sample, threshold)
% Look at the frequency domain representation of the surface peaks in order
% to repair it
%
% Input:
%   nms         The surface to repair, represented by a sequence of
%               impulses
%   sample      The sample window
%   threshold   The threshold applied to the power spectrum
%
% Output:
%   rep         The repaired surface

% Create the output surface
seg = zeros(size(nms));
```

```
% Transform the surface to the frequency domain
fp = fftshift(fft2(double(nms)));

% Find the centre of the image
Icentre = floor(size(nms,1)/2) + 1;
Jcentre = floor(size(nms,2)/2) + 1;

% Equalize the image
mfp = equalize(abs(fp));

% Find the maximum value in the image
maxval = max(max(mfp));

% Set threshold
if (nargin == 3)
    thresh = mfp > threshold;
else
    thresh = mfp > maxval/3;
end

% Shrink until each point is exactly one
thresh = bwmorph(thresh, 'shrink', Inf);

% Find the four regions in the surface corresponding to the fundamental
% frequencies of the image

% Find point locations in the image
[I J] = find(thresh ~= 0);

% Calculate the distance to the centre
if (size(I,1) == 1)
    rep = zeros(size(nms));
    return;
end

for (i = 1:size(I,1))
    distance(i) = mdist([Icentre Jcentre], [I(i) J(i)]);
end
```

```
% Sort the peaks by distance
[sorted idx] = sort(distance);

% Find the first point (closest)
freqp(1,1) = I(idx(2));
freqp(1,2) = J(idx(2));

% Find the second point (symmetric to closest)
freqp(2,1) = Icentre - (freqp(1,1) - Icentre);
freqp(2,2) = Jcentre - (freqp(1,2) - Jcentre);

% Iteratively increase the beta region determined by points close1 and
% close2 until one of the other points in the image intersect with this
% region. The first point to intersect is the next fundamental frequency
% (along with its symmetric pair)
for (betaV = 0.1:0.1:Inf)
    % Scroll through all the points
    found = 0;
    for (i = 2:size(I,1))
        % Check if we have encountered this point before
        done = 0;
        for (j = 1:size(freqp,2))
            if ((I(idx(i)) == freqp(j,1)) && (J(idx(i)) == freqp(j,2)))
                done = 1;
            end
        end

        % Check if a point is inside the beta region
        if (~done)
            if (insideBeta([I(idx(i)) J(idx(i))],
                ... freqp(1,:), freqp(2,:), betaV))
                found = 1;
                break;
            end
        end
    end

    % Check if we've found a region inside the beta graph
```

```
    if (found)
        % Mark the point inside the beta region
        freqp(3,1) = I(idx(i));
        freqp(3,2) = J(idx(i));

        % Find its symmetric pair (subtract vector to previous point from
        % origin)
        freqp(4,1) = Icentre - (freqp(3,1) - Icentre);
        freqp(4,2) = Jcentre - (freqp(3,2) - Jcentre);

        % Quit all loops
        break;
    end
end

% Expand the regions found with the beta graph
repnms = expandregions(freqp, mfp);

% Mutiply the segmented image by the fourier image
repnms = fftshift(repnms.*fp);

% Transform the result back to the spatial domain
repnms = real(ifft2(repnms));

% Find the maxima of the repaired correlation surface
repnms = nonmaximalsuppression(repnms, [size(sample,1) size(sample,2)]);

% Combine the old nms result and the new repaired nms result. To do this,
% find all of the peaks in the repaired nms and centre the sample window
% around this point. If any peaks from the old surface lie in this window,
% keep the old peaks. If the window is empty, we have a new peak.

% Find the peaks in the repaired nms
peak = find(repnms ~= 0);

% For each peak in the repaired nms surface
for (i = 1:size(peak,1))
   % Convert the peak to (i,j) coordinates
   [I J] = ind2sub(size(nms), peak(i));
```

```
    % Extract a window within the old nms surface around the peak in the new
    % nms surface

    % Get the window coordinates
    Ismall = I - round(size(sample,1)/3);
    Ilarge = I + round(size(sample,1)/3);
    Jsmall = J - round(size(sample,2)/3);
    Jlarge = J + round(size(sample,2)/3);

    % Make sure the coordinates are within image range
    if (Ismall <= 0)
        Ismall = 1;
    end
    if (Ilarge > size(nms,2))
        Ilarge = size(nms,2);
    end
    if (Jsmall <= 0)
        Jsmall = 1;
    end
    if (Jlarge > size(nms,1))
        Jlarge = size(nms,1);
    end

    % Extract the window from the old image
    window = nms(Ismall:Ilarge, Jsmall:Jlarge);

    % Find the peaks within the window
    contents = find(window ~= 0);

    % Check if the window is empty
    if (size(contents,1) == 0)
        % The window is empty, so include the new peak in the old surface
        nms(I,J) = 1;
    end
end

% Output the repaired surface
rep = nms;
```

Listing D.9: Test if a point is indside a β region.

```
function inside = insideBeta(x, p, q, betaV)
% Returns 1 if x is inside the region defined by p, q, and beta, 0
% otherwise.
%
% Input:
%   x       The point to test for inclusion
%   p       Point 1 of the beta
%   q       Point 2 of the beta
%   betaV   beta parameter
%
% Output:
%   insise  1 if x is inside the region, 0 otherwise

% Error check
if (betaV < 0)
    betaV
    error('beta out of range.  Range is: 0 <= beta <= Inf');
    return;
end

% Handle the two special cases
if (betaV == Inf)
    % The neighbourhood for beta = Inf is a strip between p and q

    % Find the slope of the line perpindicular to line p,q. (slope of
    % perpindicular line is negative reciprocal of regular slope)
    m = -(q(1) - p(1))/(q(2) - p(2));

    % Find a second point on the line y = mx - mp(1) + p(2) by
    % setting x to 0 and computing y
    y1 = -m*p(1) + p(2);

    % Find a second piont on the other line y = mx - mq(1) + q(2) by
    % setting x to 0 and computing y
    y2 = -m*q(1) + q(2);

    % Check for inclusion in this strip
    % Use signed area to determine this (O'Rourke)
```

```
    if (((delta(p,[0 y1],x) == 1) && (delta(q,[0 y2],x) == -1))
        ... || ((delta(p,[0 y1],x) == -1) && (delta(q,[0 y2],x) == 1)))
        inside = 1;
    else
        inside = 0;
    end

    return;
end

if (betaV == 0)
    % The neighbourhood is the line between p and q.
    % Return true if x is on this line.

    % Use signed area to determine this (O'Rourke)
    if (delta(p,q,x) == 0)
    inside = 1;
    else
        inside = 0;
    end

    return;
end

% Calculate the distance between p and q
distance = mdist(p,q);

if (betaV >= 1)
    % The neighbourhood for beta >= 1 is the intersection of two circles
    % of radius beta*d(p,q)/2 centred at (1-beta/2)p + (beta/2)q
    % and (beta/2)p + 1-beta/2)q.

    % Find the centre and radius of ball one
    centreB1 = ((1 - betaV/2).*p + (betaV/2).*q);
    radiusB1 = (betaV/2)*distance;

    % Find the centre and radius of ball two
    centreB2 = ((1 - betaV/2).*q + (betaV/2).*p);
    radiusB2 = radiusB1;
```

```
else
    % The neighbourhood for beta < 1 is the intersection of two circles
    % of radius d(p,q)/2beta passing through both p and q.
    % http://mathforum.org/library/drmath/view/53027.html

    % The midpoint between p and q
    mid = [(p(1) + q(1))/2 (p(2) + q(2))/2];

    % Find the centre and radius of ball one
    radiusB1 = distance/(2*betaV);
    centreB1(1) = mid(1) + sqrt(radiusB1\^2 -
        ... (distance/2)\^2)*((p(2) - q(2))/distance);
    centreB1(2) = mid(2) + sqrt(radiusB1\^2 -
        ... (distance/2)\^2)*((q(1) - p(1))/distance);

    % Find the centre and radius of ball two
    radiusB2 = radiusB1;
    centreB2(1) = mid(1) - sqrt(radiusB2\^2 -
        ... (distance/2)\^2)*((p(2) - q(2))/distance);
    centreB2(2) = mid(2) - sqrt(radiusB2\^2 -
        ... (distance/2)\^2)*((q(1) - p(1))/distance);
end

% Test if x is inside the two circles
if ((circle(x(1),x(2),centreB1(1),centreB1(2)) <= radiusB1)
    ... && (circle(x(1),x(2),centreB2(1),centreB2(2)) <= radiusB2))
  inside = 1;
else
    inside = 0;
end

function z = circle(x, y, i, j)
% Function for a circle

z = sqrt((x-i)\^2 + (y-j)\^2);

function a = area(v1,v2,v3)
% Calculate the signed area of the vertices v1, v2, v3
```

```
a = ((v2(1)-v1(1))*(v3(2)-v1(2))) - ((v3(1)-v1(1))*(v2(2)-v1(2)));

function d = delta(v1,v2,v3)
% Calculate the delta value.  1 if v3 is right of v1-v2, -1 if left, and 0
% if on.

if (area(v1, v2, v3) < 0)
    d = 1;
elseif (area(v1, v2, v3) == 0)
    d = 0;
elseif (area(v1, v2, v3) > 0)
    d = -1;
end
```

Listing D.10: Expand regions using a gradient descent.

```
function exp = expandregions(reg, surface)
% Expand the regions using the surface as a guide.  First ascend to the
% highest connected pixel, then gradient descend and add every pixel
% encountered to the list.
%
% Input:
%   reg      The regions to expand
%   surface The surface used to control expansion
%
% Output:
%   exp      The segmentation

exp = zeros(size(surface));

% At each point, gradient ascend until we find the local maxima
% neighbouring this point
for (i = 1:size(reg,1))
    % Select the point to deal with
    point = reg(i,:);

    done = 0;
    while (~done)
        % Check if one of the 8 neighbours is greater than this point
        org = surface(point(1), point(2));
```

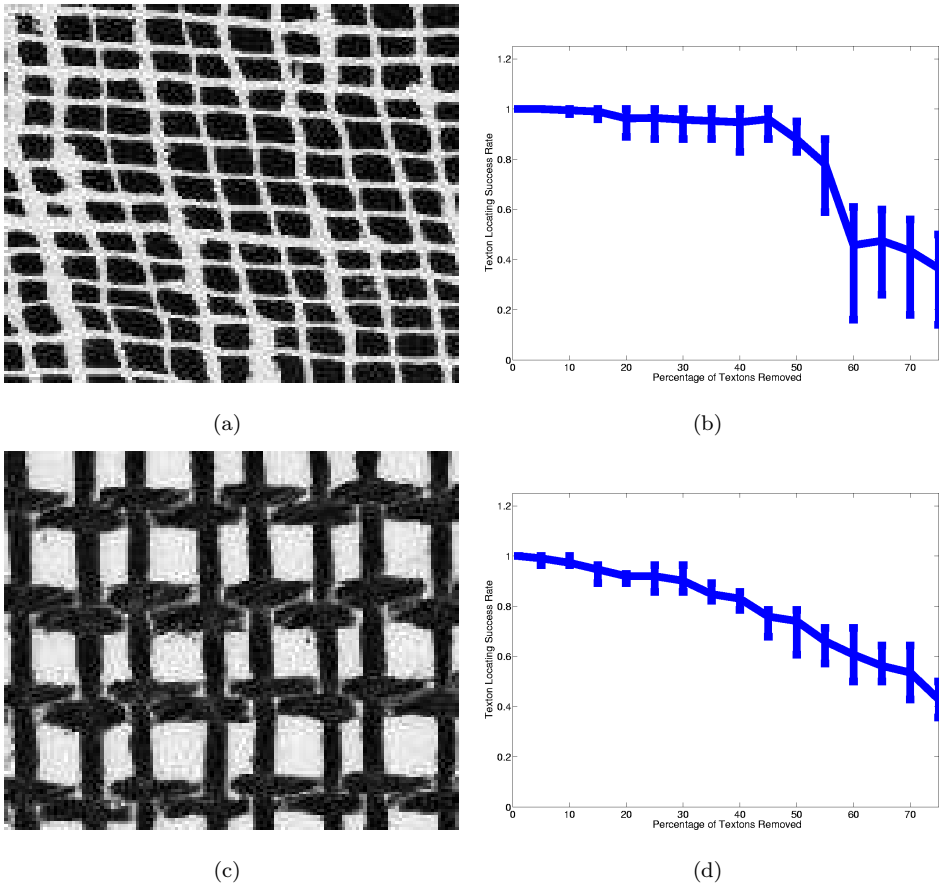
```
% North
curr = surface(point(1) - 1, point(2)) - org;
ptemp = [point(1) - 1, point(2)];

% North-East
temp = surface(point(1) - 1, point(2) + 1);
if ((temp - org) > curr)
    ptemp = [point(1) - 1, point(2) + 1];
    curr = temp - org;
end

% East
temp = surface(point(1), point(2) + 1);
if ((temp - org) > curr)
    ptemp = [point(1), point(2) + 1];
    curr = temp - org;
end

% South-East
temp = surface(point(1) + 1, point(2) + 1);
if ((temp - org) > curr)
    ptemp = [point(1) + 1, point(2) + 1];
    curr = temp - org;
end

% South
temp = surface(point(1) + 1, point(2));
if ((temp - org) > curr)
    ptemp = [point(1) + 1, point(2)];
    curr = temp - org;
end

% South-West
temp = surface(point(1) + 1, point(2) - 1);
if ((temp - org) > curr)
    ptemp = [point(1) + 1, point(2) - 1];
    curr = temp - org;
end
```

```
        % West
        temp = surface(point(1), point(2) - 1);
        if ((temp - org) > curr)
            ptemp = [point(1), point(2) - 1];
            curr = temp - org;
        end

        % North-West
        temp = surface(point(1) - 1, point(2) - 1);
        if ((temp - org) > curr)
            ptemp = [point(1) - 1, point(2) - 1];
            curr = temp - org;
        end

        % Move the point in the proper direction
        if (curr > 0)
            point = ptemp;
        else
            done = 1;

            reg(i,1) = point(1);
            reg(i,2) = point(2);
        end
    end
end

% Create priority queue nodes out of the points
for (i = 1:size(reg,1))
    ireg(i).location = reg(i,:);
    ireg(i).priority = 1;
end

% Now, p1, p2, p3, p4 are the proper locations for peaks in the surface.
% Do a gradient descent from these locations to find regions to segment.
pq = pqInit();
for (i = 1:size(ireg,2))
    pq = pqInsert(ireg(i), pq);
end
```

```
while (pq.curSize ~= 0)
    % Get the current point
    [point pq] = pqRemove(pq);

    % Mark this point in the output
    exp(point.location(1), point.location(2)) = 1;

    % Get this locations four neighbours, and process them
    p1 = [point.location(1) - 1, point.location(2)]; % North
    p2 = [point.location(1), point.location(2) + 1]; % East
    p3 = [point.location(1) + 1, point.location(2)]; % South
    p4 = [point.location(1), point.location(2) - 1]; % West

    for (i = 1:4)
        switch(i)
            case 1
                t.location = p1;
            case 2
                t.location = p2;
            case 3
                t.location = p3;
            case 4
                t.location = p4;
        end

        % Set the points priority
        t.priority = 1;

        % Check the difference in magnitude of the frequency domain
        if (surface(point.location(1), point.location(2)) -
            ... surface(t.location(1), t.location(2)) > 0)
            % Add this new point to the queue
            pq = pqInsert(t, pq);
        end
    end
end
```

Appendix E

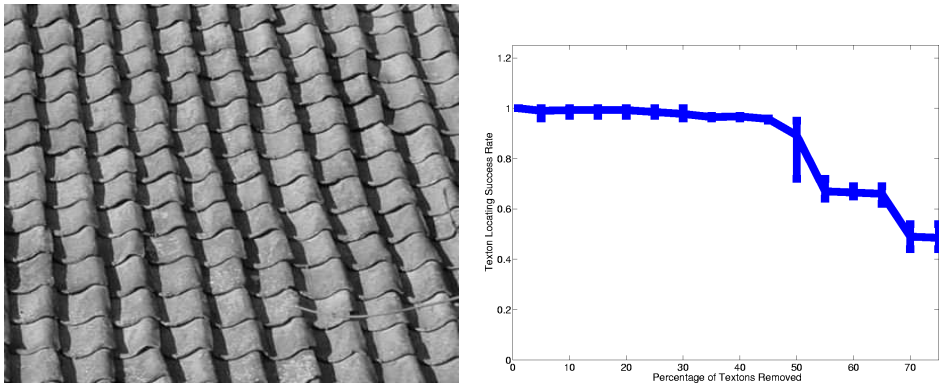
Data from Testing



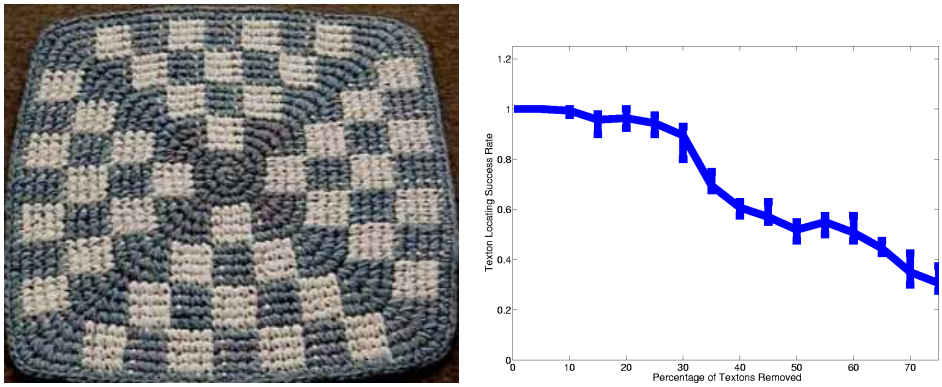
(a) (b)

(c) (d)

Figure E.1: Results of testing for robustness of the local examination frequency-based repair strategy. The horizontal axis records the number of texton locations removed from the data and the vertical axis records the texton locating success rate after frequency-based repair. Part One.

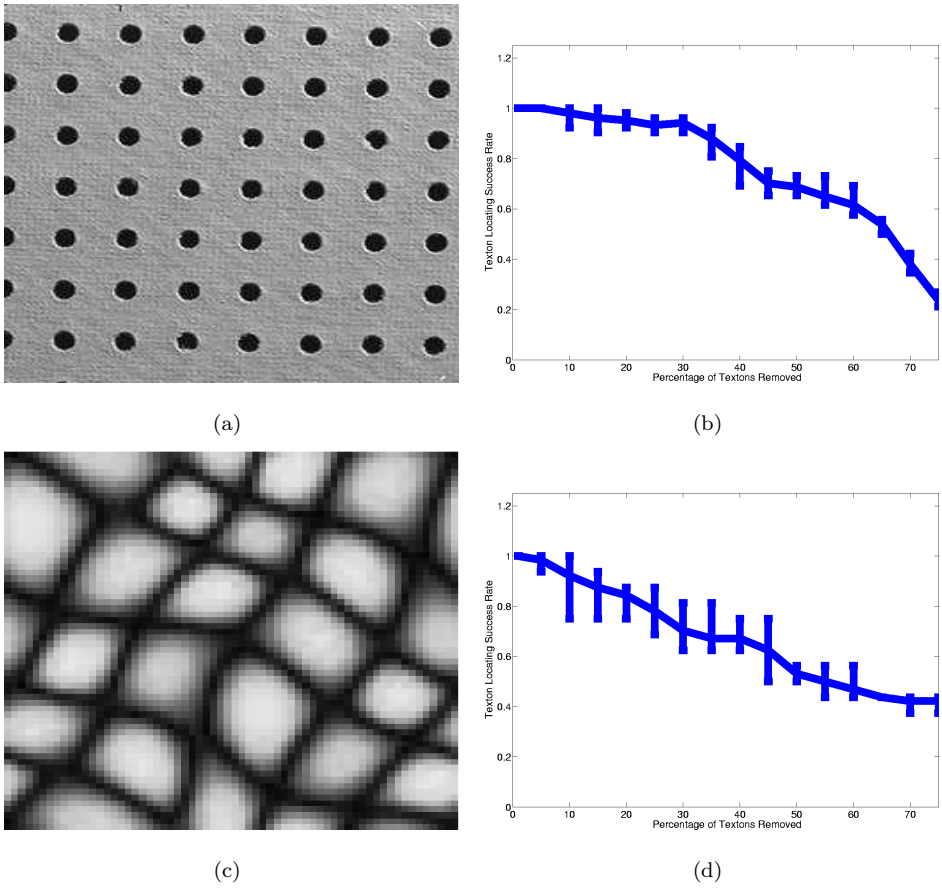


(a) (b)



(c) (d)

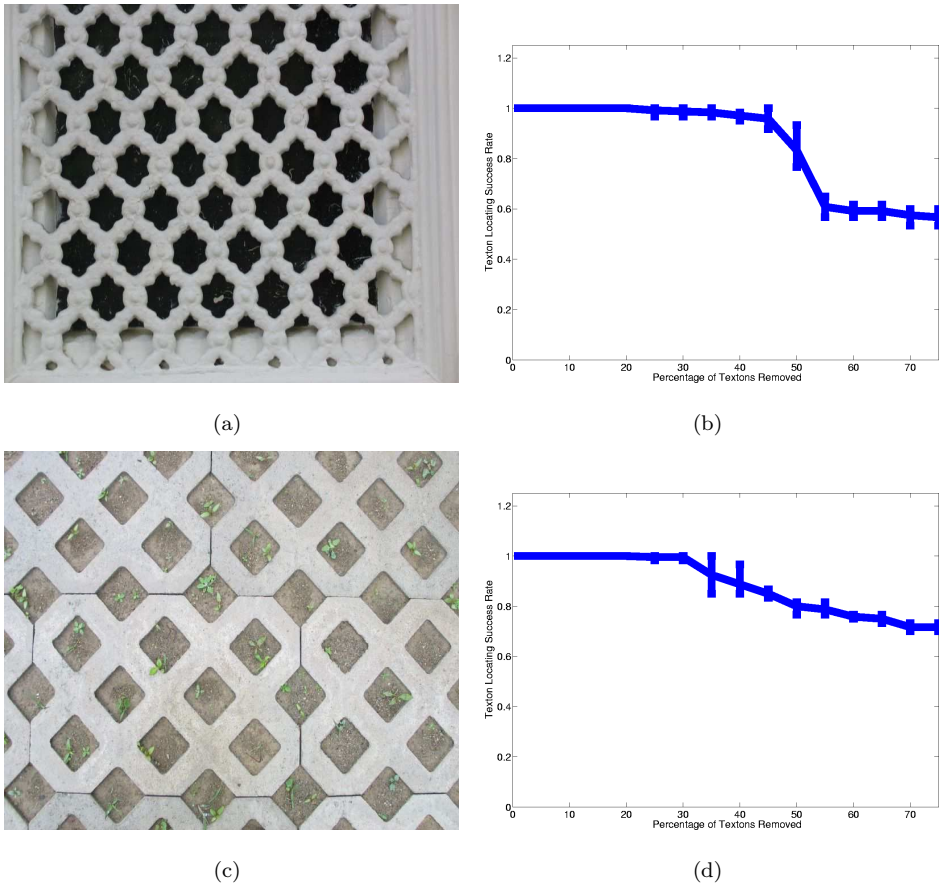
Figure E.2: Results of testing for robustness of the local examination frequency-based repair strategy. The horizontal axis records the number of texton locations removed from the data and the vertical axis records the texton locating success rate after frequency-based repair. Part Two.



(a) (b)

(c) (d)

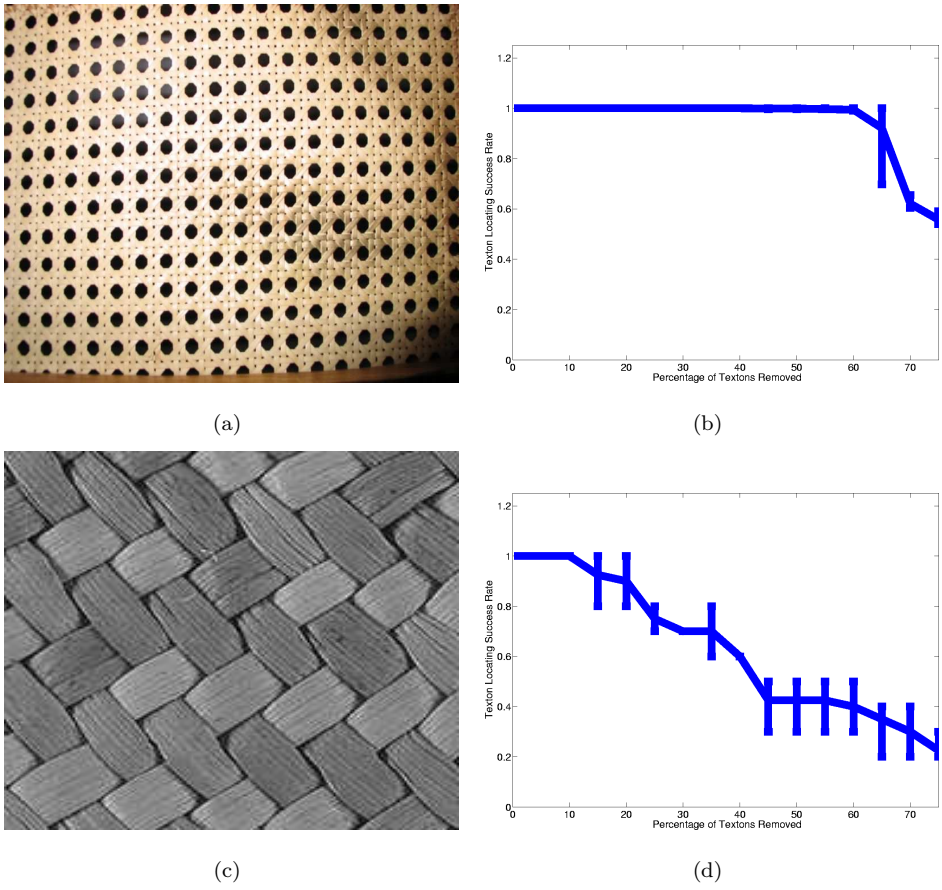
Figure E.3: Results of testing for robustness of the local examination frequency-based repair strategy. The horizontal axis records the number of texton locations removed from the data and the vertical axis records the texton locating success rate after frequency-based repair. Part Three.



(a) (b)

(c) (d)

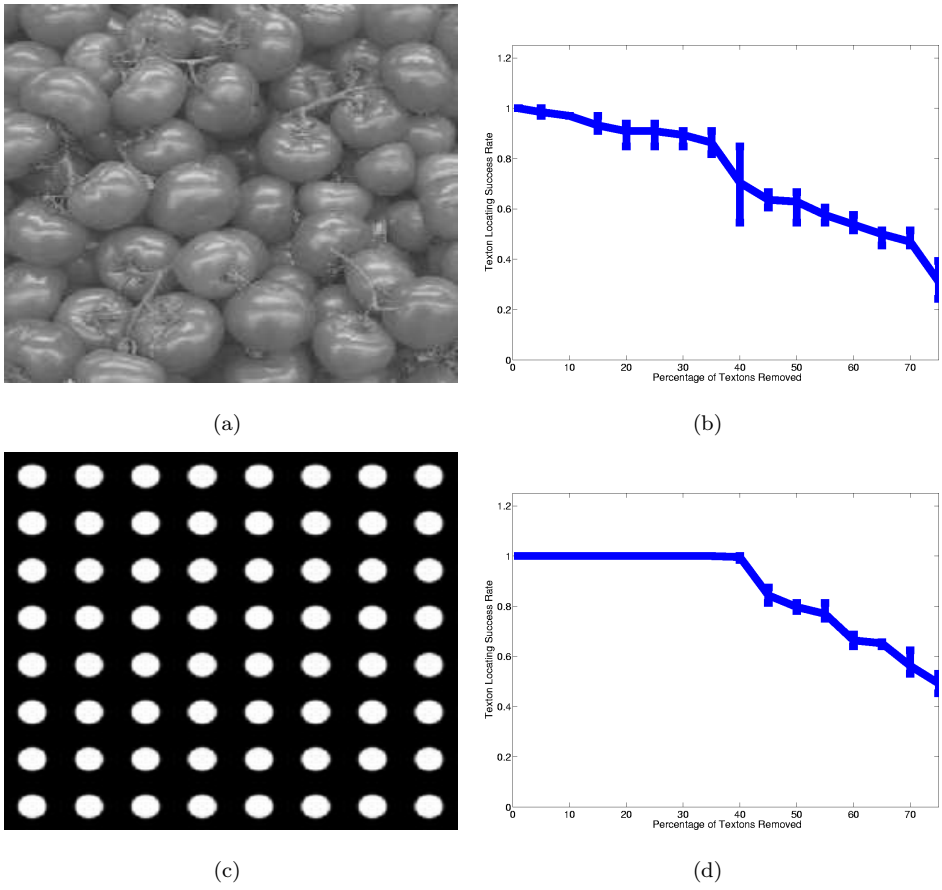
Figure E.4: Results of testing for robustness of the local examination frequency-based repair strategy. The horizontal axis records the number of texton locations removed from the data and the vertical axis records the texton locating success rate after frequency-based repair. Part Four.



(a) (b)

(c) (d)

Figure E.5: Results of testing for robustness of the local examination frequency-based repair strategy. The horizontal axis records the number of texton locations removed from the data and the vertical axis records the texton locating success rate after frequency-based repair. Part Five.



(a) (b)

(c) (d)

Figure E.6: Results of testing for robustness of the local examination frequency-based repair strategy. The horizontal axis records the number of texton locations removed from the data and the vertical axis records the texton locating success rate after frequency-based repair. Part Six.

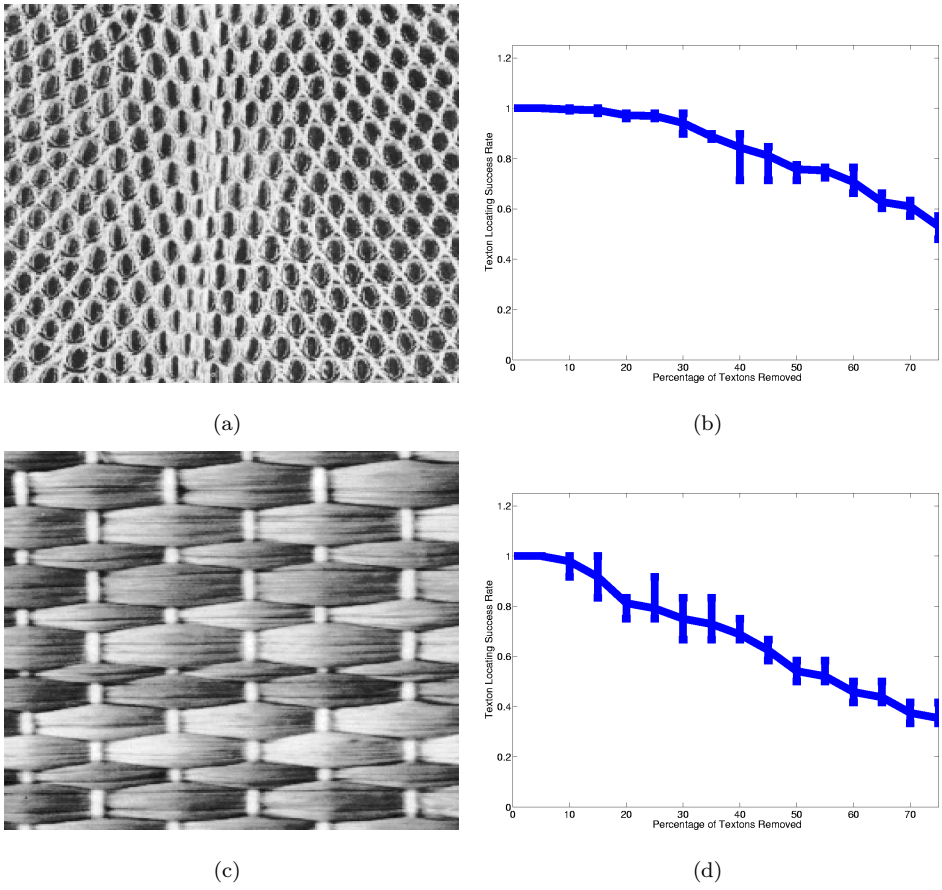
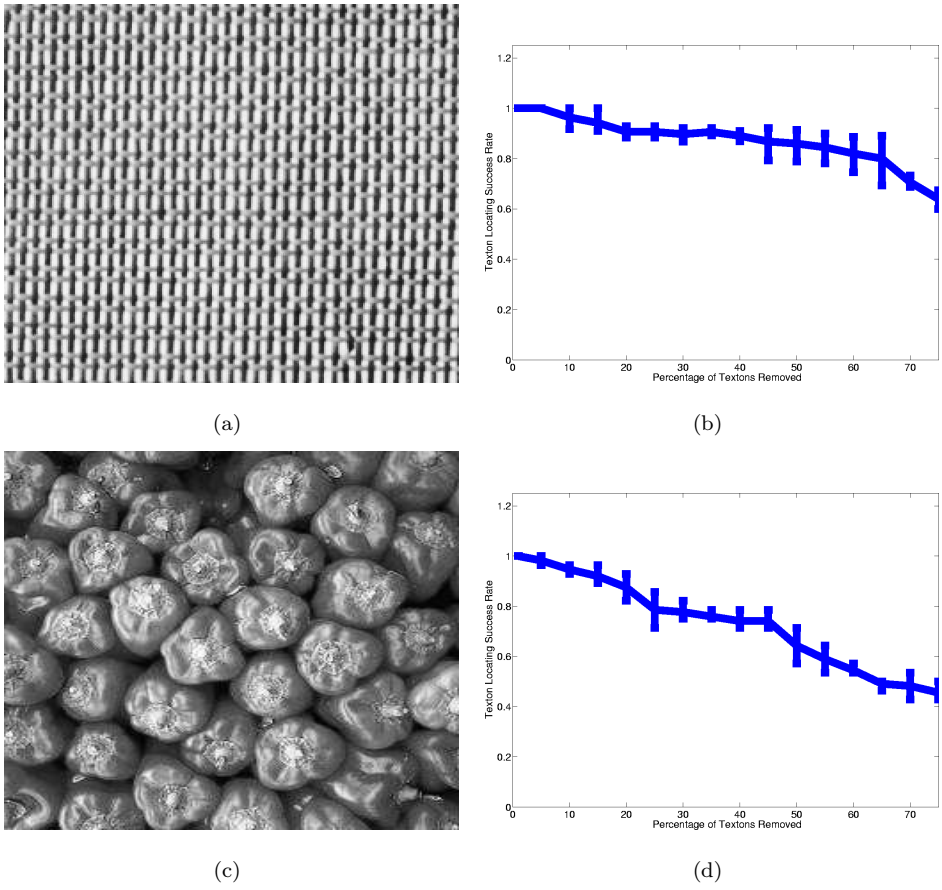


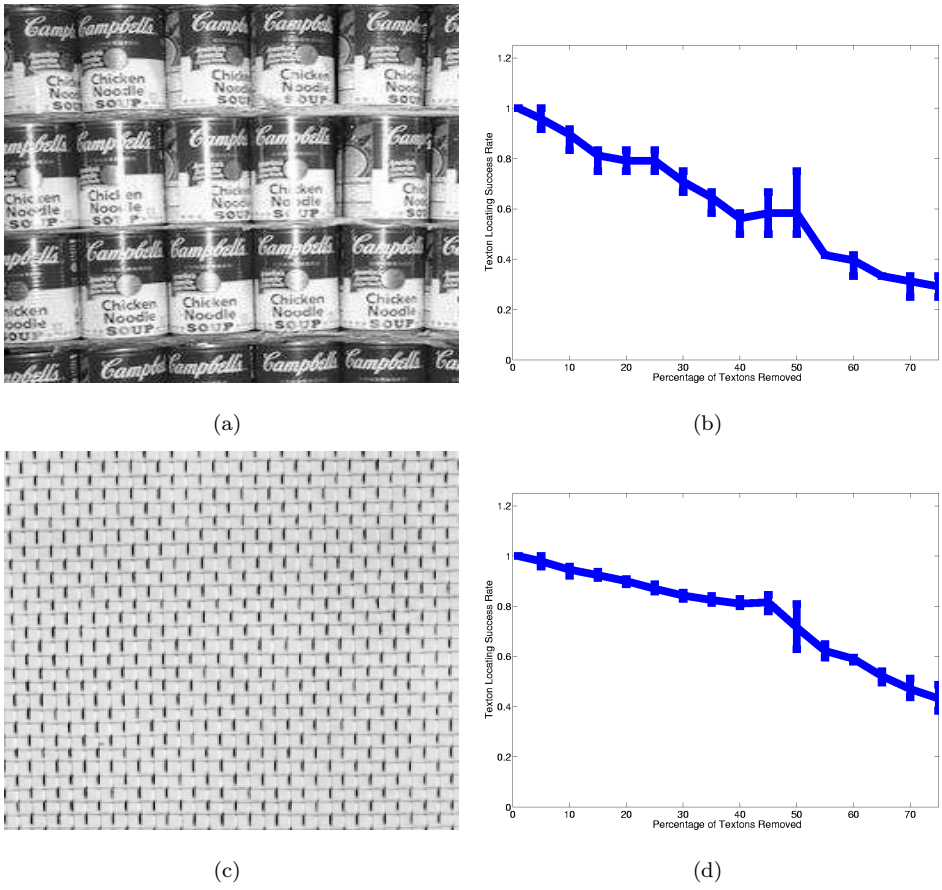
Figure E.7: Results of testing for robustness of the local examination frequency-based repair strategy. The horizontal axis records the number of texton locations removed from the data and the vertical axis records the texton locating success rate after frequency-based repair. Part Seven.



(a) (b)

(c) (d)

Figure E.8: Results of testing for robustness of the local examination frequency-based repair strategy. The horizontal axis records the number of texton locations removed from the data and the vertical axis records the texton locating success rate after frequency-based repair. Part Eight.



(a) (b)

(c) (d)

Figure E.9: Results of testing for robustness of the local examination frequency-based repair strategy. The horizontal axis records the number of texton locations removed from the data and the vertical axis records the texton locating success rate after frequency-based repair. Part Nine.

Table E.1: Classification of errors in the texton locating procedure.

Texture	Missing Textons	Misplaced Textons
7.jpg	1	0
3.jpg	1	1
D18_s.o.jpg	1	1
D56_s.o.jpg	1	0
d18.pgm	1	1
d55.pgm	1	0
d66.pgm	1	0
d74.pgm	1	1
d84.pgm	1	1
d87.pgm	1	0
stones.jpg	1	0
yellow-peppers256.o.jpg	1	1
106_0631_IMG_001.sized.jpg	1	0
Fabric.0002.ppm	1	0
IMG_0060_001.sized.jpg	1	0
IMG_0066.sized.jpg	1	0
IMG_0070.sized.jpg	1	0
IMG_0071.sized.jpg	1	0
IMG_0081.sized.jpg	1	0
IMG_0094.sized.jpg	1	1
IMG_0112.sized.jpg	1	0
IMG_0237.sized.jpg	0	1
Texture2_025.sized.jpg	0	1
database_74.sized.jpg	1	1
fabric_01.sized.jpg	1	1
fabric_46.sized.jpg	1	1
fabric_66.sized.jpg	1	1
weave_mod.jpg	1	0
Totals	27	13

Table E.2: Test results of the sensitivity of texton extraction to choice of texton. Column one reports the name of the texture, columns two through six report the current texton selection, and column seven reports the total number of successful texton extractions. 1 in the table represents a successful texton extraction. The bottom row of the table reports the average number of successful texton extractions.

Texture	One	Two	Three	Four	Five	Successes
11.jpg	1	1	1	1	1	5
161.jpg	1	1	1	1	1	5
33_06_04_web.jpg	1	1	0	1	0	3
697.jpg	1	1	1	1	1	5
9.jpg	1	1	1	1	1	5
D103.jpg	0	1	1	0	1	3
D20.jpg	1	1	1	0	1	4
D22.jpg	1	1	1	1	1	5
D36.jpg	1	1	0	1	1	4
Fabric.0008_s.o.jpg	0	0	1	1	1	3
Fabric.0014_s.o.jpg	0	0	1	1	1	3
brick3_s.o.jpg	0	0	0	1	0	1
campbell256.o.jpg	1	1	1	1	1	5
d1.pgm	1	1	1	1	1	5
d22.pgm	1	0	1	0	0	2
d3.pgm	1	1	1	1	1	5
d35.pgm	1	1	1	1	0	4
d52.pgm	0	0	1	0	0	1
d55.pgm	0	0	0	0	1	1
d66.pgm	1	0	0	0	0	1
red-peppers256.o.jpg	1	0	0	1	1	3
roofing2.jpg	1	0	1	0	0	2
Buildings.0009.ppm	1	1	1	1	1	5
107_0703_IMG_001.sized.jpg	1	1	1	0	0	3
IMG_0203.sized.jpg	1	0	1	1	1	4
Average Success Rate						3.48

Table E.3: Test results of correlation with image features. Part One. Column one reports the name of the texture, columns two through six report the current feature space. Values of one in the table represent a successful texton extraction. The bottom row of the table reports the percentage of successful texton extractions per column.

Texture	RGB	Gradient	Laplacian	Sobel	Canny
10.jpg	1	0	0	0	0
11.jpg	1	0	0	0	0
161.jpg	1	0	0	0	0
33_06_04_web.jpg	1	0	0	1	0
3.jpg	0	0	0	0	0
697.jpg	1	0	0	0	0
7.jpg	0	0	0	1	0
9.jpg	1	0	0	0	0
brick3_s.o.jpg	1	0	0	0	0
Buildings.0009.ppm	1	0	0	0	0
campbell256.o.jpg	1	0	0	0	0
cftpg45.jpg	1	0	0	0	0
checkerboard.o.jpg	1	0	0	0	0
college-inn256.o.jpg	1	1	1	1	1
d101.pgm	1	0	0	0	1
D103.jpg	1	0	0	0	0
d18.pgm	0	0	1	0	0
D18_s.o.jpg	0	0	0	0	0
d1.pgm	1	1	1	1	0
D20.jpg	1	0	0	0	0
D22.jpg	1	0	0	0	0
d22.pgm	1	0	0	0	0
D34.jpg	1	0	0	0	0
d34.pgm	1	0	0	0	0
d35.pgm	1	0	0	0	0
Percentage Successful	0.84	0.08	0.12	0.16	0.08

Table E.4: Test results of correlation with image features. Part Two. Column one reports the name of the texture, columns two through six report the current feature space. Values of one in the table represent a successful texton extraction. The bottom row of the table reports the percentage of successful texton extractions per column.

Texture	RGB	Gradient	Laplacian	Sobel	Canny
D36.jpg	1	0	0	0	0
d36.pgm	1	0	0	0	0
d3.pgm	1	0	0	0	0
d52.pgm	1	0	0	0	0
d53.pgm	1	0	0	0	0
d55.pgm	0	0	0	0	0
d56.pgm	1	0	0	0	0
D56_s.o.jpg	0	0	0	0	0
d64.pgm	1	0	0	1	0
d66.pgm	1	0	0	0	0
d6.pgm	1	1	0	0	1
d74.pgm	0	0	0	0	0
d84.pgm	0	0	0	0	0
d87.pgm	0	0	0	0	0
disc-lattice.o.jpg	1	1	0	0	0
Fabric.0008_s.o.jpg	1	0	0	0	0
Fabric.0014_s.o.jpg	1	0	0	0	0
fabric_30.sized.jpg	1	0	0	1	1
red-peppers256.o.jpg	1	1	1	0	1
roofing2.jpg	1	0	0	0	0
Tile.0007_c.o.jpg	1	1	1	1	0
tomatoes256.o.jpg	1	0	0	0	0
windowsP256.o.jpg	1	0	1	1	1
yellow-peppers256.o.jpg	0	0	0	0	0
zesta-crackers224.o.jpg	1	0	0	0	0
Percentage Successful	0.76	0.16	0.12	0.16	0.16

Table E.5: Test results of correlation with alternate colour spaces. Part One. Column one reports the name of the texture, columns two through seven report the current colour space. Values of one in the table represent a successful texton extraction. The bottom row of the table reports the percentage of successful texton extractions per column.

Texture	RGB	Hue	Saturation	Value	HSV	Luminance
106_0631_IMG_001.sized.jpg	0	0	0	0	0	0
107_0702_IMG_001.sized.jpg	1	0	1	1	1	1
107_0703_IMG_001.sized.jpg	1	1	0	1	1	1
Buildings.0009.ppm	1	0	0	0	1	1
Fabric.0000.ppm	1	0	0	0	0	0
Fabric.0002.ppm	0	0	0	0	0	0
IMG_0029_001.sized.jpg	1	1	1	1	0	0
IMG_0055_001.sized.jpg	1	1	1	0	0	0
IMG_0060_001.sized.jpg	0	0	0	0	0	0
IMG_0066.sized.jpg	0	0	0	0	0	0
IMG_0068.sized.jpg	1	0	1	1	1	1
IMG_0070.sized.jpg	0	0	0	0	1	1
IMG_0071.sized.jpg	0	0	0	0	0	0
IMG_0075.sized.jpg	1	1	0	0	0	0
IMG_0079.sized.jpg	1	0	0	0	0	1
IMG_0080.sized.jpg	1	1	0	0	1	1
IMG_0081.sized.jpg	0	0	0	0	0	0
IMG_0083.sized.jpg	1	1	0	0	0	0
IMG_0085.sized.jpg	1	1	0	0	0	0
IMG_0086.sized.jpg	1	0	1	1	1	1
IMG_0094.sized.jpg	0	0	0	0	0	0
IMG_0098.sized.jpg	1	0	1	1	1	1
IMG_0112.sized.jpg	0	0	0	0	0	0
IMG_0118.sized.jpg	1	0	1	1	1	1
IMG_0125.sized.jpg	1	0	1	0	0	1
Percentage Successful	0.64	0.28	0.32	0.28	0.36	0.40

Table E.6: Test results of correlation with alternate colour spaces. Part Two. Column one reports the name of the texture, columns two through seven report the current colour space. Values of one in the table represent a successful texton extraction. The bottom row of the table reports the percentage of successful texton extractions per column.

Texture	RGB	Hue	Saturation	Value	HSV	Luminance
IMG_0134.sized.jpg	1	1	1	0	1	1
IMG_0169.sized.jpg	1	0	0	0	0	0
IMG_0170.sized.jpg	0	0	0	0	0	0
IMG_0175.sized.jpg	1	0	0	0	0	0
IMG_0193.sized.jpg	1	1	1	1	0	0
IMG_0203.sized.jpg	1	0	1	0	0	1
IMG_0209.sized.jpg	1	1	1	1	1	1
IMG_0235.sized.jpg	1	0	1	0	0	0
IMG_0237.sized.jpg	0	0	0	1	1	1
IMG_1491.sized.jpg	1	1	0	0	1	1
IMG_4476.sized.jpg	0	0	0	0	0	1
IMG_4839.sized.jpg	1	0	0	0	0	1
Texture1_008.sized.jpg	1	0	0	1	1	1
Texture2_025.sized.jpg	0	0	0	0	0	0
cftpg45.jpg	1	1	0	1	1	1
database_74.sized.jpg	0	0	0	0	0	1
fabric_01.sized.jpg	0	1	1	0	0	0
fabric_30.sized.jpg	1	0	1	1	1	1
fabric_46.sized.jpg	0	1	1	0	0	0
fabric_66.sized.jpg	0	1	1	0	0	0
honeycomb2.jpg	1	1	0	0	1	1
james5.sized.jpg	1	0	1	0	0	0
weave_mod.jpg	0	0	0	0	0	0
z1_honeycomb-texture.jpg	1	1	0	0	1	1
z2_101.jpg	1	0	0	1	1	1
Percentage Successful	0.64	0.40	0.40	0.28	0.40	0.60

Table E.7: The effect of frequency-based repair on the success rate of texton locating. Column one reports the name of the texture, columns two reports the texton locating success rate before frequency-based repair, columns three to six report the texton locating success rate after frequency-based repair using addition, multiplication, averaging, and local repair methods.

Texture	Original	Addition	Multiplication	Averaging	Local
IMG_4839.sized.jpg	0.40	0.54	0.18	0.20	0.23
IMG_0235.sized.jpg	0.54	0.69	0.60	0.95	0.90
IMG_0070.sized.jpg	0.55	0.57	0.62	0.48	0.45
IMG_0085.sized	0.58	0.88	0.63	0.88	0.88
IMG_0075.sized.jpg	0.61	0.50	0.44	0.22	0.39
107_0703_IMG_001.sized.jpg	0.61	0.61	0.27	0.18	0.18
IMG_0083.sized	0.63	0.70	0.77	0.80	0.70
IMG_0079.sized.jpg	0.67	0.77	0.90	0.73	0.60
Fabric.0000.ppm	0.67	0.94	0.89	0.68	0.71
Fabric.0002.ppm	0.67	0.56	0.85	0.89	0.84
database_74.sized.jpg	0.67	0.70	0.65	0.66	0.68
IMG_0175.sized.jpg	0.71	1.00	0.47	0.69	0.67
z2_101.jpg	0.75	1.00	1.00	1.00	1.00
IMG_0203.sized.jpg	0.78	1.00	1.00	0.95	0.95
107_0702_IMG_001.sized.jpg	0.80	1.00	0.98	0.96	0.96
IMG_0125.sized.jpg	0.81	0.87	0.92	0.91	0.94
Buildings.0009.ppm	0.86	0.88	0.92	0.94	0.89
IMG_0209.sized.jpg	0.88	1.00	1.00	1.00	1.00
fabric_30.sized.jpg	0.91	0.98	0.98	0.97	0.97
IMG_0080.sized.jpg	0.92	1.00	1.00	1.00	1.00
IMG_1491.sized.jpg	0.94	0.71	0.94	0.96	0.96
z1_honeycomb-texture.jpg	0.94	1.00	0.96	0.90	0.92
IMG_0134.sized.jpg	0.98	1.00	0.97	1.00	1.00
IMG_0068.sized.jpg	0.99	1.00	1.00	0.99	0.99
IMG_0098.sized.jpg	0.99	1.00	0.97	0.98	0.98
cftpg45.jpg	1.00	1.00	0.98	1.00	1.00
honeycomb2.jpg	1.00	1.00	1.00	1.00	1.00
IMG_0118.sized.jpg	1.00	1.00	0.97	1.00	1.00
IMG_0086.sized	1.00	1.00	0.69	1.00	1.00
Texture1_008.sized.jpg	1.00	1.00	0.97	1.00	1.00