Space Frequency Shape Inference and Segmentation of 3D Surfaces

John Krumm

CMU-RI-TR-93-32

The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

December 1993

This research was sponsored by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, OH 45433-6543 under Contract F33615-90-C-1465, Arpa Order No. 7597 and by NASA under the Graduate Students Researchers Program, Grant No. NGT-50423. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of ARPA, NASA or the U.S. Government.

Carnegie Mellon

School of Computer Science

DOCTORAL THESIS
in the field of
Robotics

Space/Frequency Analysis of Image Texture

JOHN KRUMM

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy

ACCEPTED:

Steven A Shafer — THESIS COMMITTEE CHAIR — 14 December 1993 — DATE

Steven A Shafer — DEPARTMENT HEAD — 14 December 1993 — DATE

APPROVED:

DEAN — 12/14/93 — DATE

Abstract

Image texture is useful for segmentation and for computing surface orientations of uniformly textured objects. If texture is ignored, it can cause failure for stereo and gray-scale segmentation algorithms. In the past, mathematical representations of image texture have been applied to only specific texture problems, and no consideration has been given to the models' generality across different computer vision tasks and different image phenomena. We advocate the space/frequency representation, which shows the local spatial frequency content of every point in the image. From several different methods of computing the representation, we pick the spectrogram. The spectrogram elucidates many disparate image phenomena including texture boundaries, texture in perspective, aliasing, zoom, and blur.

Many past shape-from-texture algorithms require potentially unreliable feature detection and "magic numbers" (arbitrary parameters), and none of them were developed in the context of a more general texture-understanding system. Toward this end, we show that the spatial frequency shifts caused by perspective can be approximated by an affine transformation which is a function of the textured surface's surface normal. We use this relationship in three different shape-from-texture algorithms. Two of them require no feature-finding and work on the raw spectrogram, giving a high-level scene parameter directly from low-level image data. The first algorithm includes an analytical sensitivity analysis. The third algorithm works with just the peak frequencies and gives a fast way of computing local surface normals from periodic texture. The algorithms need only a few magic numbers. On real textures, the average error in computed surface normal is only about four degrees.

We use the third algorithm to solve a long-standing problem in image texture analysis: segmenting images of textured, 3D surfaces. Past work in texture segmentation and shape-from-texture is based on assumptions that make this problem impossible to solve. 3D perspective effects warp the otherwise uniform textures so segmentation fails. We develop a region-growing algorithm that accounts for the surface normals by unwarping the frequency distribution. It uses a minimum description length merge criterion. Our algorithm successfully segments images of real texture. We conclude by showing how the space/frequency representation has the potential for unifying several different computer vision algorithms.

Acknowledgments

Steve Shafer is a world-class professor. He took me in to his group when my first advisor left CMU, and he arranged for me to stay. He gave me enough freedom to keep me interested and enough guidance to keep me relevant. He defended my work when others were skeptical. At meetings Steve was quick to understand even the most technical details, while still giving valuable advice on high-level strategy and presentation. I leave CMU having learned many valuable things from him about research, persuasion, and integrity.

Thank you to my committee, Takeo Kanade, Andy Witkin, and Sandy Pentland, for helpful suggestions and for holding me to a high standard.

Thank you to CMU, NASA, and ARPA for financial support.

Since I have been here a while, I've had the benefit of many kind and interesting officemates. They were Rich Volpe, Masatoshi Okutomi, Rich Wallace, Yoshinoro Kuno, Jin-Oh Kim, Brad Nelson, and Rich Voyles (three Rich's!). They're all fine people from whom I've learned much. Rich Wallace taught me about minimum description length. Trying to match Brad's long hours helped me get a lot more work done.

Alan Christiansen was my first friend in Pittsburgh, and has remained my friend since then. I enjoyed many filling dinners and fun discussions with my cafeteria friends Richard Volpe, Madeleine Strum, and Djamal Bouzida.

For both technical help and fun times I thank Carol Novak, Reg Willson, Ron Stone, and Mark Maimone - all of the Calibrated Imaging Lab (CIL) group. By using Reg's nifty camera calibration programs, I could confidently attribute any deficiencies in my results to my own mistakes. Thank you also to Carlo Tomasi for being friendly and helpful. The other good folks from the CIL group include Gudrun Klinker, Larry Matthies, Bruce Maxwell, and Yalin Xiong.

I benefited from the excellent research facilities of the Vision and Autonomous Systems Center (VASC), the Robotics Institute, and the School of Computer Science. Thanks to Ralph Hyre, Jim Moody, and Bill Ross for making it easy to get things done in VASC. Special thanks to Takeo Kanade for creating VASC in the first place. Thank you to Mary Jo Dowling for help with graphics, including printing the final draft of this thesis.

CMU's Engineering and Science library had almost everything I needed, and got the rest of it quickly. Its staff is a dedicated group.

I appreciate the many all-around good people at CMU that I was lucky enough to befriend. They include Mike Gillinov, Ben Brown, Jim Rehg, Dina Berkowitz, Chuck Thorpe, Vladimir Brajovic, Omead Amidi, David Simon, Dan Morrow, Sing Bing Kang, Fred Solomon, Kevin Lynch, and Srinivas Akella. I've rewritten this paragraph more times than any other in this thesis, and I suppose I've still forgotten someone.

I benefited from several secretaries who took time to help me even though they didn't have to. Thanks to Tracy Lewis, Lori Fauschnacht, Patty Mackiewicz, Marie Elm, Carolyn Kraft, and Pauletta Pan.

Thank you to my Pittsburgh friends Joe Bielecki, Hydie Houston, and Dana Bookey for being friendly.

Thanks to Joseph Fourier for inventing his transform, and thanks to Karel Vander Lugt for getting me interested in Fourier transforms as an undergraduate Physics major. Thank you to Sir Isaac Newton and Gottfried Wilhelm von Leibniz for thinking up calculus.

My parents raised me right so I could do my best. They worked hard and saved so I could go to college, and they supported me in my decision to stay in school for so long. Thank you to them. Thanks to Grandma for many pleasant telephone conversations and the subscription to *Reader's Digest*, which I read while my slow programs were running. Thank you to my parents-in-law for being interested and fun, and for raising such a beautiful daughter.

The best idea I had while at CMU was to marry Ellen. She keeps me happy all the time. Leaving this fine school, these nice friends, and this good city will be much easier knowing that she is going with me.

Table of Contents

Chapter 1 Analyzing Image Texture

1.1 The Theses of This Thesis

Automatic recognition and understanding of image texture is critical for machine understanding of general images. Almost every scene, either natural or man-made, contains some texture. Texture analysis is important because it can tell us many things about the scene. If an object is uniformly textured, then the object's surface normals can be computed from the perspective distortion of the texture in the image (shape-from-texture). Regions with similar texture are likely from the same object -- a good clue for image segmentation. Understanding texture is also important because texture can confound algorithms that do not account for it. For instance, segmentation algorithms are usually based on an assumption of smoothly varying gray levels, which is not true for texture. Stereo matching often fails on repetitive texture. Thus, to avoid errors with other algorithms and to exploit what we can from texture, we need to recognize and understand it in images.

This thesis presents algorithms for analyzing images of textured objects based on the space/frequency representation. This representation shows an image's local spatial frequency characteristics at every pixel. Since texture is a phenomenon of repeated patterns, its spatial frequency distribution is an effective way to understand it.

Our major algorithmic results are programs for shape-from-texture and for the segmentation of textured, 3D surfaces. All use the space/frequency representation, and the segmentation program uses one of the shape-from-texture algorithms as a subroutine. Although there have been algorithms developed previously to perform each of these tasks separately, we are the first to solve them simultaneously by doing segmentation and shape analysis in the same image. This attests to the power and versatility of the space/frequency representation.

The theses of this thesis are:

1. The space/frequency representation is an effective method of analyzing texture in images. It is a natural way to reason about texture, and it makes many phenomena obvious and understandable.

2. Although there are many ways to compute the space/frequency representation, the spectrogram is best for understanding many phenomena as well as doing shape-from-texture and segmentation.

3. The space/frequency representation is a natural one for shape-from-texture. The resulting algorithms are simple, accurate, and do not require feature-finding or many "magic numbers."

4. Shape-from-texture and texture segmentation are not separate problems, but must be solved simultaneously in order to understand texture as it occurs in the real world. The space/frequency representation can be used to solve both problems in a relatively simple manner by explicitly accounting for shape effects.

The next section of this chapter discusses various texture models that have been used for image texture analysis and explains the advantages of the space/frequency representation. We then present several different images of textured objects showing various effects like shape, aliasing and blur. We show how the spectrogram (one version of the space/frequency representation) makes these phenomena clear and how it can be used to understand them.

Chapter 2 illustrates some important issues in computing the space/frequency representation. We need to understand this tool before using it for fixing computer vision problems. It is impossible to do a perfect job of computing the representation, especially when faced with the nonstationarities in most images. We explain some of the inherent trade-offs we must contend with. After analyzing the representation, we survey several different methods for computing the representation and explain why we chose the spectrogram.

In Chapter 3 we develop three different shape-from-texture algorithms based on the spectrogram. The algorithms exploit depth-induced, local spatial frequency shifts to find surface normals. We get accurate results on both periodic and irregular textures. The algorithms require no strong assumptions about the textures and do not use many "magic numbers" to guide them toward the solution.

Chapter 4 uses one of the shape-from-texture algorithms as part of a texture segmentation algorithm. By explicitly accounting for shape, we can successfully segment images of textured 3D surfaces.

We summarize the thesis and present future research directions in Chapter 5.

1.2 Models of Image Texture

A texture model is a transformation (and normally a compression) of an image texture into a set of quantities that describe the texture. Usually the image data itself is inadequate for this task, because it is incoherent in its raw form. A texture always varies quickly in gray level, and a good texture model gives a description of this variation. A simple example is a sinusoidal texture: instead of simply listing the raw intensity values, it makes sense to describe it in terms of its amplitude, frequency, phase, and constant offset.

We must chose a texture model before we can analyze image texture. The left column of Table 1 lists some frequently used texture models in computer vision. The next column gives the reference to the first significant use of the model for image texture. These are papers that show how the particular model can be used to concisely express certain features of the texture or to generate realistic looking textures. There are also texture surveys in the literature[35][112][108].

There are three primary problems in computer vision that can be solved using texture models: texture classification, texture segmentation, and shape-from-texture. Brief definitions of these problems are:

Classification: Assign a texture image (or part of a texture image) to one of a finite number of previously defined texture classes.

Segmentation: Group together the pixels in the image that have the same texture. The number and type of textures are not known in advance.

Shape from Texture: Given an image region covering a uniformly textured, 3D surface, find the surface normals of the surface.

A cross product of texture models and texture problems gives the matrix of Table 1. The elements of this matrix are citations of significant, early use of the model for the particular problem. For each

Representation	Modeling	Classification	Segmentation	Shape
first-order statistics			Muerle and Allen 1968 [74]	
cooccurrence matrices		Haralick *et al.* 1973 [34]	Holyer & Peckinpaugh 1989 [41]	
run length matrices		Galloway 1975 [30]		
autocorrelation	Krueger 1988 [55]	Chellappa & Chatterjee 1985 [14]		Brown & Shvaytser 1990 [11]
Markov random fields	Hassner & Sklansky 1980 [37]	Kaneko and Yodogawa 1982 [52]	Cohen & Cooper 1987 [20]	Patel & Cohen 1992 [82]
fractal Brownian surfaces	Mandelbrot 1977 [69]			Pentland 1984 [83]
random mosaics	Schachter *et al.* 1978 [93]			
Wold decomposition	Francos *et al.* 1993 [28]			
low-level features			Rosenfeld & Thurston 1971 [91]	Witkin 1981 [115]
high-level features	Rosenfeld & Lipkin 1970 [90]		Tsuji & Tomita 1973 [105]	Kender 1979 [53]
local frequency filters	Bajcsy 1973 [4]	Gramenopoulos 1973 [33]	Triendl 1972 [103]	Jau & Chin 1990 [45]

Table 1: Texture models and significant, early applications to texture problems in computer vision.

entry, there are usually several other related works that could be cited as well. Not all models have been used for all the problems, because not all the models are expressive enough. We note that the local frequency filter model that we advocate has been used for modeling, classification, segmentation, and shape, indicating that it is indeed very expressive. The next several subsections give brief descriptions of the models in Table 1 along with their applications. We summarize and compare the models at the end. This dissertation advocates the space/frequency representation for modeling texture.

1.2.1 First-Order Statistics

First-order statistics are quantities like mean, variance, and skew. Some textures can be distinguished by these numbers. Muerle and Allen[74] compare histograms (sample probability distributions) from small sections of the image for segmentation by region growing. Although this model is appropriate for distinguishing most textures, some textures could have identical first-order statistics and still appear to be quite different. For instance, a sinusoidal image texture will have the same mean, variance, and histogram regardless of its frequency. This is because the model does not account for the spatial pattern of the texture. This deficiency also means the model is inadequate for exploiting the spatial distortions necessary for doing shape-from-texture. It is also sensitive to lighting variations, which is a disadvantage.

1.2.2 Cooccurrence Matrices

This is also called the "gray tone spatial dependence" approach[35], and it was first popularized by Haralick *et al.* in 1973[34]. The cooccurrence matrices are a sort of histogram telling how often each possible pair of gray levels occur in a given spatial relationship. The spatial relationships between pairs of pixels are parametrized by d and θ, where d is the separation in pixels and θ is the angle between the pixels. There is one cooccurrence matrix for each (d, θ) pair. In practice, the possible spatial relationships are limited by the grid structure of the image pixels. If the number of possible gray levels in the image is N_g, then the cooccurrence matrices have dimensions $N_g \times N_g$. The matrices are given by $P(i, j; d, \theta)$, where i and j represent the two gray levels. Each entry tells how many times gray levels i and j occurred at (d, θ) with respect to each other.

The matrices are computed over a given region in an image for a given set of (d, θ) pairs. From each matrix, certain scalar features are computed. For instance, the "angular second-moment" is the sum of the squares of the matrix elements. This will be high for images with a few, dominant gray level transitions. Other features are "contrast," and "correlation." Haralick *et al.*[34] describe 14 different features that can be computed from the cooccurrence matrices. They go on to use these features for texture classification.

Cooccurrence matrices and their associated features have proven to be powerful texture descriptors, and are still being used frequently for texture classification and more recently for texture edge detection by Holyer and Peckinpaugh[41]. New work is concentrating on deriving better features from the matrices. Although cooccurrence matrices do account for the spatial pattern of the texture (as opposed to first-order statistics), they have not been used for shape-from-texture.

1.2.3 Gray Level Run Length Matrices

Gray level run length matrices (GLRLM's) are similar to cooccurrence matrices in that the matrices are computed from the image data and then scalar descriptors are computed from the matrices. The GLRLM is a histogram of the gray level and length of linear runs of pixels of equal gray level in a given direction. There is one GLRLM for each direction (usually $\theta = 0°, 45°, 90°,$ or $135°$), and each GLRLM is $N_g \times N_g$. The matrix element $P(i, j; \theta)$ gives the number of run lengths of gray level i of length j in direction θ. Galloway[30] defines five scalar features based on these matrices with names like "short runs emphasis," and "gray level nonuniformity." She uses these features for texture classification. Like cooccurrence matrices, GLRLM's do not lend themselves to 3D texture analysis.

1.2.4 Autocorrelation

The autocorrelation function of an image texture measures the correlation between the texture's gray levels as a function of their relative positions. For a continuous image function $f(x, y)$, the autocorrelation is

$$A(x, y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f^*(x, y) f(x - \alpha, y - \beta) d\alpha d\beta \quad (1)$$

where * means complex conjugate. The autocorrelation of a real function (the only kind we deal with) is real and symmetric. If the texture is random, then $A(x, y)$ is usually a unimodal function. The autocorrelation functions of random rough textures drop off more quickly than those of random smooth textures. Periodic textures have periodic autocorrelations. $A(x, y)$ and the power spectrum are Fourier transforms of each other, so there is a theoretical link between the autocorrelation model and the frequency filter model.

Krueger[55] describes how to use various autocorrelation functions of surface height to generate realistic-looking random textures for computer graphics. Chellappa and Chatterjee[14] used estimates of the autocorrelation function over 21x21 pixel windows for texture classification. Brown and Shvaytser[11] use the autocorrelation function for shape-from-texture. They assume the frontal texture has a circularly symmetric autocorrelation, and they compute slant and tilt based on the deviations from this symmetry.

Our shape-from-texture and segmentation algorithms both use local power spectra, which are just Fourier transforms of local autocorrelations. The Fourier domain is better because it lets us easily express such effects as aliasing and blur that would be harder to formulate with the autocorrelation.

1.2.5 Markov Random Fields

The Markov random field (MRF) model says that the probability density function governing a pixel's intensity is a function only of that pixel's neighbors. A Gaussian MRF models each pixel intensity as a linear combination of the intensities in the neighborhood plus zero-mean, Gaussian noise. A first order MRF model defines the pixel's neighborhood as the four-connected neighbors. A second order model uses the eight-connected neighbors. The order of the model can be arbitrarily high, but real implementations are usually no higher than fourth-order, which involves a surrounding neighborhood of 20 pixels.

Some of the first work in using the MRF model for image texture was in generating textures based on the model. Hassner and Sklansky[37] generated binary textures using first-order, binary MRF's, and Chellappa[13] reproduced some textures from the Brodatz[10] album by fitting fourth-order and eighth-order models. Kaneko and Yodogawa[52] fit Gaussian MRF models to various textures and use the parameters in a distance measure for texture classification. Cohen and Cooper[20] give a detailed account of an algorithm for segmenting images based on MRF models. The MRF model allows them to apply the algorithmic tools of maximum likelihood to the segmentation, resulting in very accurate region boundaries. In their shape-from-texture work, Patel and Cohen[82] use a Gaussian MRF model for texture. They first find a frontal texture by looking at a pair of windowed regions. They compute surface normals by finding the surface orientation most consistent with the deformation this frontal texture. This MRF is a popular texture model, and several other researchers are exploiting it.

1.2.6 Fractal Brownian Surfaces

A fractal Brownian surface $z = f(x, y)$ has Gaussian distributed increments $f(x_2, y_2) - f(x_1, y_1)$ with variance proportional to a power of the Euclidean distance separating the two planar points[109]. That is

$$E[|f(x_2, y_2) - f(x_1, y_1)|^2] \propto [(x_2 - x_1)^2 + (y_2 - y_1)^2]^H . \qquad (2)$$

H controls the roughness of the surface. This is the model that Mandelbrot used to produce realistic looking mountainscapes[69]. He and others have shown how fractals can model many different natural phenomena. Pentland[83] argues that most natural surfaces follow a fractal model not only in their height distribution but also in their surface normals. From this he concludes that these natural fractal surfaces produce fractal intensity images (assuming Lambertian reflection and constant albedo). He goes on to develop an algorithm using this model that does shape-from-texture and shape-from-shading simultaneously.

1.2.7 Random Mosaics

A random mosaic is an image that has been randomly divided into regions, each with a randomly assigned gray level. Schachter *et al.*[93] review some random mosaic models, including

Occupancy Model: A Poisson process puts points on the image, and the region boundaries are given by the Voronoi diagram.

Poisson Line Model: Boundary lines of the form $x\cos\theta + y\sin\theta = \rho$ are randomly chosen from the space (ρ, θ) by a Poisson process.

Rotated Checkerboard Model: An (x, y) origin and orientation are randomly chosen, and the image is divided into a grid of square cells of a given dimension along these axes.

Bombing Model: Points are put onto the image like the occupancy model, and each point serves as the center of a predefined shape (*e.g.* a circle of a given radius). This divides the image into a foreground and background.

The Poisson line model and the rotated checkerboard model are amenable to statistical analysis, and Schachter *et al.* fit theses models to actual texture images. Certain mosaic models are appropriate for certain kinds of texture, but none of their models appear general enough for a wide class of textures. We have not seen any classification, segmentation, or shape algorithms based on these models.

1.2.8 Wold Decomposition

The Wold decomposition expresses a signal as the sum of a random and deterministic part. In the context of texture modeling, Francos *et al.*[28] model textures as the sum of a deterministic periodic component, an evanescent component, and a purely random component. The periodic component is extracted from the peaks of the Fourier transform of the texture, and the evanescent component comes from extended, straight ridges in the Fourier transform. The remaining texture is modeled as an autoregressive process. They successfully reconstruct a variety of textures based on this model. Since the idea is new, there have not been applications to classification, segmentation, or shape, although the model's generality is attractive for these tasks.

1.2.9 Low-Level Features

By "low-level features" we mean easily-detectable, simple image features like edges. Even though most textures are not made up of tiny line segments, the density, direction, or size of the edges found by an edge detector may vary depending on the type of texture or its surface orientation. Rosenfeld and Thurston[91] show how to find texture boundaries based on edge density. Witkin[115] uses simple edges in his shape-from-texture algorithm. He detects edges as the zero-crossings after computing $\nabla^2 G$ of the image. The slant of the surface causes a nonuniform distribution of edge directions, and he finds the surface normal that best accounts for this deviation. Edges are really just local, high-frequency effects, so these methods are related to the local frequency filter model.

1.2.10 High-Level Features

High-level *features* are discrete texture elements like line segments, circles, and polygons. Some textures can be thought of as a feature along with a placement rule for copies of that feature. Rosenfeld and Lipkin[90] discuss several different placement rules, both deterministic and probabilistic. Using a set of region-splitting heuristics based on the moments, areas, and perimeters of texture-elements, Tsuji and Tomita[105] describe how they segmented images of textures with discrete texture elements. For shape-from-texture, Kender[53] defines the "normalized textural property map." This idea applies to a variety of texture-element features such as orientation, length, interior angles, area, and density. The maps allow him to constrain the possible surface orientations of a textured plane based on assumptions about the unprojected texture elements.

This high-level-feature model applies only to textures made of discrete elements that are large enough to be detected. Detecting these features in an image is difficult. Blake and Marinos said in 1990:

> Our greatest practical problems arise from isolating independent oriented [texture] elements from an image.[7]

And Aloimonos said in 1988:

> There is no known algorithm that can successfully detect texels from a natural image.[2]

1.2.11 Local Frequency Filters

These models consider texture as a spatial frequency phenomenon, and they treat the image data in a few or many frequency bands. Bajcsy[4] defines several different features for describing textures based on windowed power spectra of an image texture. Using six features based on the power spectra taken from 32×32 pixel windows, Gramenopoulos[33] classified eight different kinds of terrain from aerial photographs. Triendl[103] segmented aerial terrain images using filtered and smoothed images that corresponded roughly to "brightness" (low spatial frequencies) and "roughness" (high spatial frequencies). Even though some textures can be discriminated in the frequency domain, Julesz and Caelli[48] show that humans can discriminate some textures with identical Fourier power spectra. Jau and Chin[45] used the Wigner distribution to do shape-from-texture by exploiting the frequency shifts caused by 3D effects.

1.2.12 Summary of Models

The best possible model of image texture would apply to all kinds of texture and work effectively for all three texture problems. It would be versatile enough to account for the inevitable annoyances of scale change, noise, blur, and aliasing. At their current state of development, none of these models can do all these things, but some come closer than others.

First-order statistics apply to any texture. Since they are independent of the spatial pattern of the texture, they are independent of scale and aliasing. This also means they are insensitive to the textured surface's surface normal, and thus cannot be used for shape-from-texture. Several very different-looking textures can have the same first-order statistics, meaning that this model's discriminatory power is low. The mean and variance of the texture's gray levels can be altered arbitrarily with an offset and scaling. This is a case of "model aliasing," a term introduced by Shafer[94]. It means that the model is not expressive enough to make the distinctions that are important, mapping different states (image textures) to the same model.

Cooccurrence matrices and run length matrices remedy one of the problems with first-order statistics: they are second-order statistics that account for the spatial pattern of the texture. The heuristic features based on these matrices work very well for classification, meaning that they are expressive enough to describe a variety of textures. For run length matrices, Loh *et al.*[63] show how to factor out effects of variations in scene depth (scale), 2D rotation, and 3D slant angles (with a previously known tilt angle). We have not found any work using either of these matrices for shape-from-texture, probably because they have no clear relationship to 3D shape.

The implementations using autocorrelation, Markov random fields, and fractal Brownian surfaces all use specific models of the textures' second-order statistics. They naturally allow considerations for noise. The fractal model is probably the most restrictive, in that it has been applied only to random, crinkly surfaces. Markov random fields can model a wide variety of textures, from random to periodic and in between. The segmentation results using this model are impressive. We have found only one reference[82] to its application to the shape-from-texture problem, and this seems like a promising research area. In their work using autocorrelation for classification and shape, respectively, Chellappa and Chatterjee[14] and Brown and Shvaytser[11] assume special forms for the autocorrelation. Tomita and Tsuji[102] point out that the power spectrum, autocorrelation, and autoregressive (Markov random field) models can all be derived from each other. They also point out the interrelationships between these and cooccurrence matrices and the Fourier transform.

Both the random mosaic and high-level feature models of texture apply only to a restricted class of textures. They operate at a high level, and therefore cannot account for low-level phenomena like noise, aliasing, and blur.

The low-level feature model is like a thresholded version of the local frequency model. Points with enough high-frequency energy are marked as edge features. This nonlinear step makes it difficult to account for noise, aliasing, and blur.

We note that there are other simple texture models. Several of these were considered by Tamura *et al.*[100] in 1978. They gave them the names coarseness, contrast, directionality, line-likeness, regularity, and roughness. These early attempts at texture analysis were largely heuristic, with no considerations or expectations for versatility.

We think the local frequency model is, on whole, the best texture model. It has been used for classification, segmentation, and shape. Since it is grounded in signal processing, it comes with deep, well-documented theory, including noise considerations. We show in the next section how several different, important effects in computer vision can be easily expressed in terms of local spatial frequency. None of the other models are this expressive.

One important effect that we will consider briefly here is scale. The problem with scale is that many textures have energy at different scales. For instance, a school of fish is a texture with the fish as texture elements. However, if we scale to the scale of the fishes' scales, they are also a texture. Changes in scene magnification due to zoom or relative depth, as well as variations in focus blur, cause texture at different scales to appear and disappear. Texture models that work only at a small number of discrete scales (*e.g.* cooccurrence matrices, run length matrices, Markov random fields) cannot easily account for these changes. (Although sometimes these problems can be solved with a multiresolution analysis.) By using a dense set of local frequency filters, we account for scale changes implicitly, because we cover frequencies from almost zero up to the Nyquist limit.

There are few empirical comparisons of texture models in the literature. The most commonly cited comparison was done by Weszka *et al.*[114] for the task of terrain classification. They compared the local frequency model, cooccurrence matrices, first-order gray level difference statistics (which we have not discussed), and gray level run length matrices. The local frequency model consisted of 16 intersections of rings and wedges computed from 64x64 windows. This model proved to be the poorest for classification, while the other three were about equal. A follow-up study by Dyer and Rosenfeld[25], in which they suppressed the aperture effects of the discrete Fourier transform, did not change the results significantly. Two more recent, albeit smaller-scale, studies indicate that different variations of the local frequency model are superior. Tan and Constantinides[101] compare their

Gabor filter texture features to cooccurrence matrix features on a texture classification task. They find the Gabor features work better, even with significant amounts of added noise. Lonnestand[64] shows that texture features based on the Haar transform, which can be considered a frequency decomposition, work better for texture classification than features based on the cooccurrence matrix or run length matrix.

1.3 Seeing Image Phenomena in Space/Frequency

The significant strength of the space/frequency representation is that it can efficiently represent several different image phenomena. This is important, because one of the things that makes computer vision difficult is the variety of image phenomena that combine to make an image from a scene. Typically, computer vision algorithms are written to exploit one effect at a time and to implicitly ignore other phenomena. For instance, all previous texture segmentation algorithms have been written to exploit texture differences, but they assume away any 3D perspective effects. The advantage of the space/frequency representation is that it gives a basis for expressing the effects of many different image phenomena simultaneously. This means we can use it write computer vision algorithms that account for the inevitable combination of phenomena that go into every realistic image.

Some effects in computer vision are best described in spatial coordinates and some are best described in frequency coordinates, as we show in the following list:

spatial effects	frequency effects
segmentation	discrete sampling
shape	moire patterns
shading	optical effects
perspective	texture
stereo	filtering

When these effects are combined, as they inevitably are in most images, they should be explicitly accounted for. If the effects have different preferred representations, then we need to find one representation to accommodate all of them. The space/frequency representation is an ideal solution, because it combines the preferred representation of many important effects.

On representations in computer vision, Marr said

> A representation is a formal system for making explicit certain entities or types of information, together with a specification of how the system does this.[70] (p. 20)

The space/frequency representation takes a variety of implicit frequency-related image phenomena and makes them explicit. It reveals certain coherence properties and trends that are only visible when plotted as a function of both space and frequency *simultaneously*. This section illustrates a number of different image phenomena in terms of their effects in space/frequency. It is not intended to demonstrate deep technical details nor to develop specific algorithms. Rather we show some images along with spectrograms of their rows. Specifically, we illustrate texture segmentation, 3D shape effects, aliasing, zoom, and focus. (We develop specific algorithms for shape-from-texture in Chapter 3 and for segmentation in Chapter 4.) This suggests that the representation is a remarkably rich one, in that its behavior can be easily characterized for a wide variety of phenomena. While the space/frequency representation is not necessarily optimal for analyzing each of the phenomena independently, it is the only one so far presented that works well for all of them.

We use the spectrogram to compute the space/frequency representation. We discuss alternatives in the next chapter. In this section, we are only concerned with showing spectrograms of image rows, so the input signals are 1D. The spectrogram of a signal $f(x)$ is simply a series of windowed power spectra of the signal. We illustrate its computation in Figure 1,.

It is defined in terms of the Fourier transform[1] as

$$\begin{aligned} S(x,u) &= \left| \int_{-\infty}^{\infty} w(\alpha - x) f(\alpha) e^{-j2\pi u\alpha} d\alpha \right|^2 \\ &= \left| \Im_{a \Rightarrow u} \{ w(a-x) f(a) \} \right|^2 \end{aligned} \quad (3)$$

where $w(x)$ is the window function. The frequency u is measured in cycles/pixel. $S(x_0, u)$ is computed by windowing a neighborhood of $f(x)$ centered at x_0, and then taking the squared magnitude

[1] Our notation for the 1D Fourier transform has u as the frequency variable, measured in cycles/unit distance. The 1D Fourier transform of $f(x)$ is given by

$$F(u) = \Im_{x \Rightarrow u} \{f(x)\} = \int_{-\infty}^{\infty} f(x) e^{-j2\pi ux} dx \ .$$

The 1D inverse Fourier transform is given by

$$f(x) = \Im^{-1}_{u \Rightarrow x} \{F(u)\} = \int_{-\infty}^{\infty} F(u) e^{j2\pi ux} du$$

We use uppercase functions to indicate the Fourier transform of the corresponding, lower-case functions.

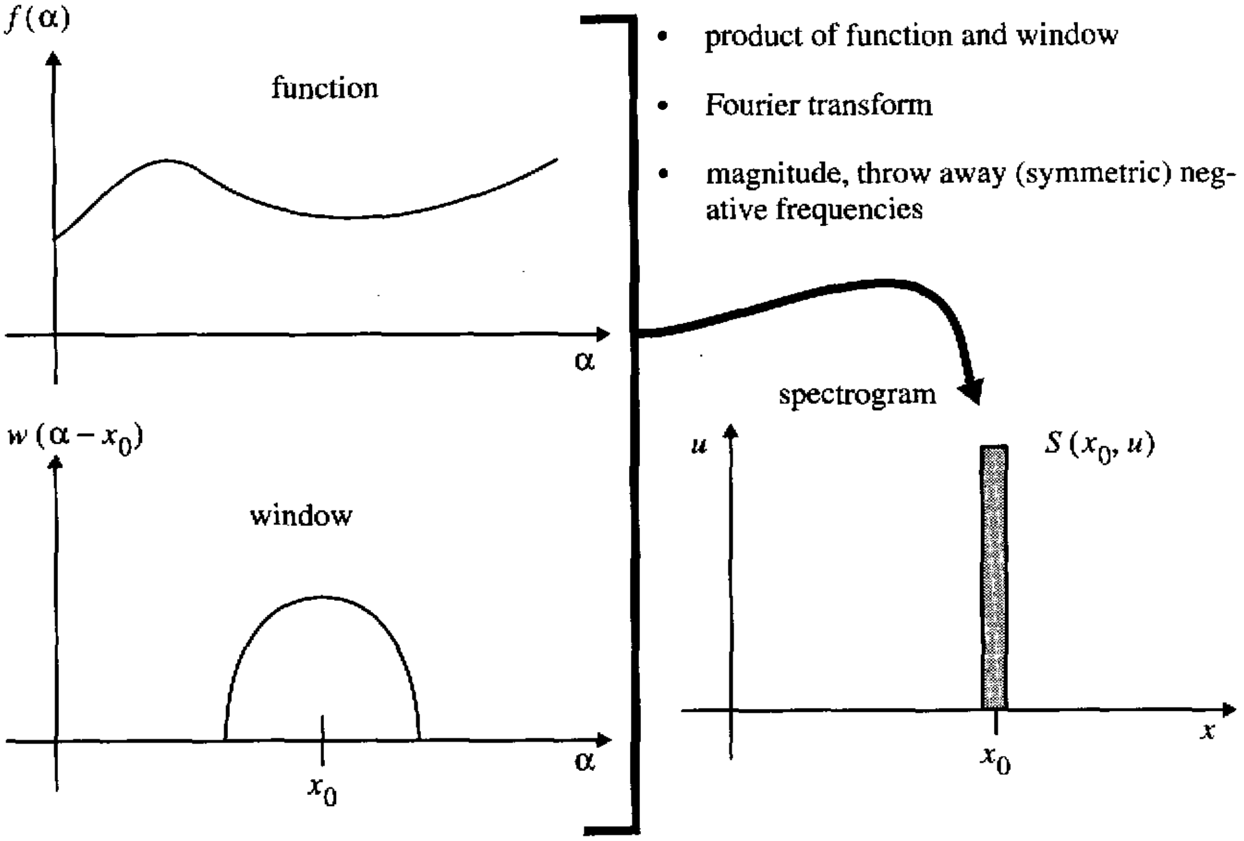


Figure 1: The spectrogram of a 1D signal is a series of Fourier power spectra computed from windowed versions of the signal.

of the Fourier transform of this windowed neighborhood. Since the power spectrum of a real signal is symmetric, we can throw away the part corresponding to negative frequencies. For the spectrogram in this chapter, we used a "Blackman-Harris minimum four-sample" window, recommended by experts for Fourier analysis[36][23]. It is

$$w(l) = w_0 + w_1 \cos\left(\frac{2\pi}{L-1}l\right) + w_2 \cos\left(\frac{4\pi}{L-1}l\right) + w_3 \cos\left(\frac{6\pi}{L-1}l\right) \tag{4}$$

where L is the (even integer) width of the window in pixels, $0 \le l \le (L-1)/2$. The coefficients are $(w_0, w_1, w_2, w_3) = (0.35875, 0.48829, 0.14128, 0.01168)$. We used a window 64 pixels wide.

A typical spectrogram computed this way is shown in Figure 2 as the rectangular block superimposed on the image. The x-axis is horizontal and the u-axis is vertical. The x-axis is registered with the appropriate columns in the image. The frequency runs from zero to one-half cycles/pixel. We have zeroed the low frequencies because they tend to dominate the more interesting activity in the higher frequencies.

1.3.1 Texture Segmentation

It is often the case that the only common characteristic across a region in an image is its texture. In order to locate texture boundaries, we need a texture model that transforms uniform texture into some uniform set of quantities and that transforms different textures into different sets of quantities. We discussed several texture models in Section 1.2.

The spectrogram is a good model for texture segmentation. This idea is not new, as it was used for aerial image segmentation as long ago as 1973[33]. And even though it is not necessarily the best choice in terms of classification accuracy[114][25], it serves our purpose because the same space/frequency representation can be used for shape-from-texture as well. In addition, we show in Sections 1.3.3 and 1.3.6 that the space/frequency representation is affected in a predictable and orderly way by aliasing and focus blur. It is not at all clear such a simple characterization could be found for other texture transformations.

Figure 2 shows the spectrogram of the center row of an image with two different, periodic textures. The textures are from the Brodatz[10] book of textures, which we scanned using a high-resolution page scanner. The textures are cotton canvas (D77) and woven aluminum wire (D6). It is clear from this figure that the spectrogram succeeds in maintaining coherence over uniform texture and distinguishing between different textures. This suggests that an edge-detection or region-growing algorithm could be applied to segment the textures. We apply a region-growing algorithm in Chapter 4.

Since both the textures in Figure 2 are periodic, a 1D slice through them can be described with a Fourier series.[2] The Fourier series for $f(x)$ is

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi n u_0 x} \tag{5}$$

[2] Any periodic function that meets the Dirichlet conditions can be described with a Fourier series[32]. For a function $f(x)$, the Dirichlet conditions are that in any interval $x_1 \le x \le x_2$, the function must

1. be single valued,
2. have a finite number of extrema,
3. have no infinite discontinuities, and at most a finite number of finite discontinuities, and
4. be absolutely integrable, *i.e.* $\int_{x_1}^{x_2} |f(x)|\, dx < \infty$.

Any physically realizable function will satisfy the Dirichlet conditions.

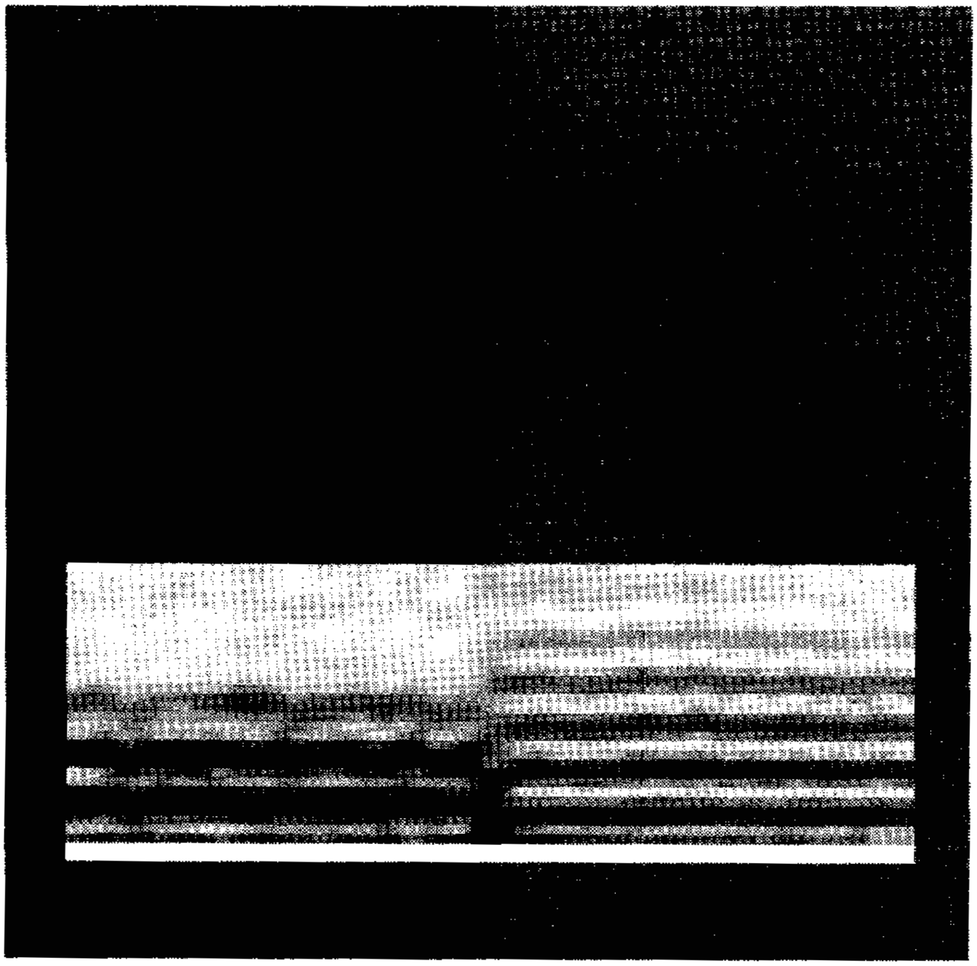


Figure 2: Two periodic textures and the spectrogram of the center row. The spectrogram over similar texture is similar and changes significantly over the boundary. This suggests that the spectrogram is a good basis for segmenting textures.

where u_0 is the fundamental frequency and the c_n are the complex Fourier series coefficients given by

$$c_n = u_0 \int_{x}^{(x + \frac{1}{u_0})} f(\alpha) e^{-j2\pi n u_0 \alpha} d\alpha \tag{6}$$

This integral is taken over any one period of the function. The spectrogram of this Fourier series (derived in Appendix 1) is

$$S(x, u) = \sum_{r=-\infty}^{\infty} |c_r|^2 |W(u - ru_0)|^2$$

$$+ 2 \sum_{n=-\infty}^{\infty} \sum_{m=n+1}^{\infty} \mathrm{Re}\left[c_n c_m^* e^{j2\pi x u_0 (n-m)} W^* (u - mu_0) W(u - nu_0) \right] \tag{7}$$

If W (u), the Fourier transform of the window, is sufficiently narrow and/or the fundamental frequency u_0 is sufficiently large, the harmonics will not overlap significantly and the cross terms in the double summation will be negligeable. Then the spectrogram of this periodic function will be a series of power spectra of the window function, shifted in increments of u_0 and weighted by the squared magnitudes of Fourier series coefficients. This is what we see in the spectrogram in Figure 2. The ridges correspond to the $|c_r|^2 |\mathrm{W}\ (u - ru_0)|^2$. Since the power spectrum of a real function is even symmetric, we only show the non-negative frequencies, corresponding to the non-negative values of r and n in Equation (7).

If the textures are not periodic but random, the spectrogram looks different. Figure 3 shows the spectrogram of the center row of an image with two different, random textures. The textures are Brodatz beach sand (D29) and pigskin (D92). Here the power spectra have much less structure than in the periodic case. For these two textures, the spectrogram still shows a degree of coherence across like textures, and it still shows a distinction between them. Unfortunately, this coherence is only apparent after a significant amount of averaging. The spectrogram in this figure is actually the average of the spectrograms taken over the center 201 rows of the image. By contrast, the spectrogram for the periodic texture in Figure 2 was averaged over only the center 21 rows. The power spectrum of a random signal is itself random. Using only the center 21 rows of the random textures, we get a much less coherent spectrogram, as shown in Figure 4. Because it takes more spatial averaging of the spectrogram, random textures are more difficult than periodic textures to "understand" with the spectrogram.

1.3.2 3D Shape

Even though most texture segmentation algorithms assume that textures in images are flat and viewed frontally, this is usually not the case. The changing depth on a non-frontally viewed texture cause the image texture to change. The texture elements on a flat, receding surface appear smaller as they go into the distance. This smallness translates into higher frequencies in the space/frequency representation. An example is shown in Figure 5. This is the Brodatz cotton canvas (D77) texture mapped onto a receding, flat plate with a computer graphics program. The spectrogram of the center row shows that the frequencies are rising as the plate recedes to the right. This is because the texture

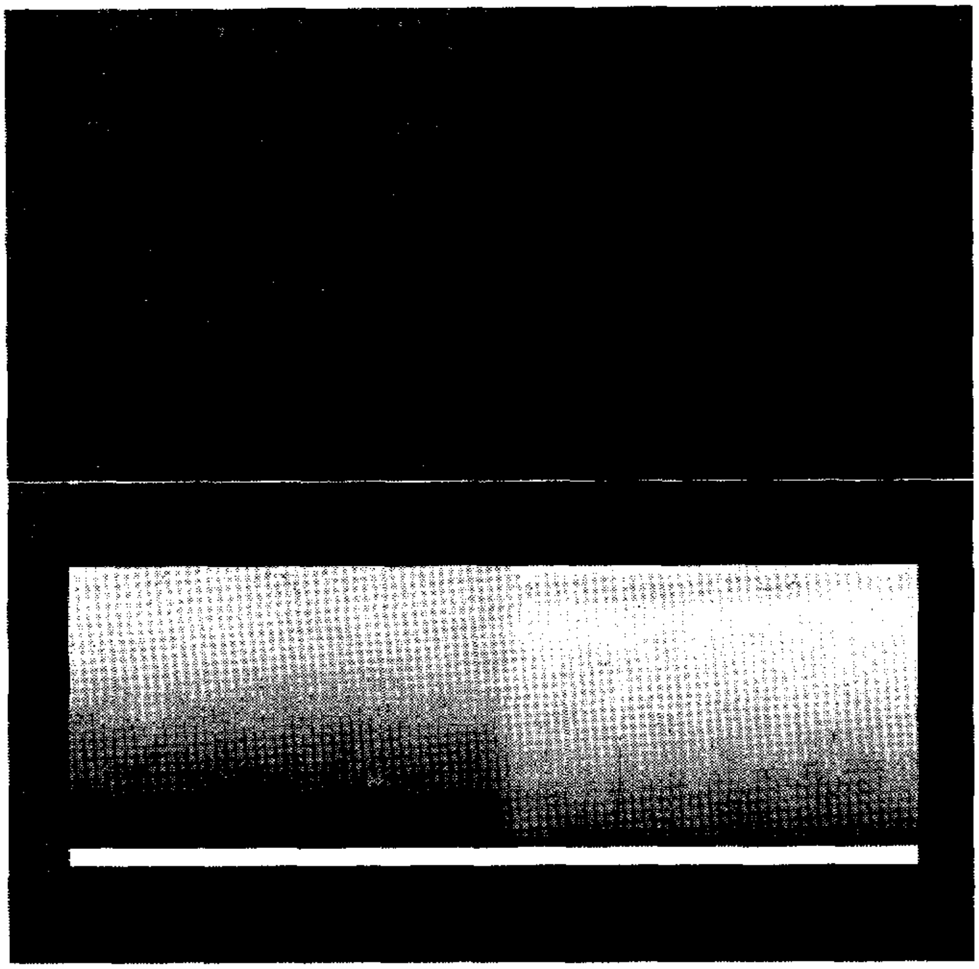


Figure 3: Two random textures and the average spectrogram of the 201 center rows. Random textures require more frequency averaging to make their similarity and differences obvious.

elements appear smaller, which means they have a smaller period, which means they have a higher frequency. (We attribute the bump in frequency to a locally denser region of the canvas weave.) The frequency gradient has a mathematical relationship to the surface normal of the plate. We develop this relationship and exploit it in our shape-from-texture algorithms in Chapter 3.

The same kind of frequency shifts occur for random textures, as shown in Figure 6. This is the Brodatz beach sand (D29) texture, and the spectrogram in the figure is the average of 201 spectrograms taken on the center 201 rows of the image. Except for the texture, the plate is the same as the plate in Figure 5. Since the power spectrum of this texture is much less sharp than that of periodic textures, it is more difficult to detect the changes in frequency. Super and Bovik[98] have had some success with these kinds of textures by tracking the second-order moments of the frequency distribution from point to point. In Section 3.5.4 we modify our basic shape-from-texture algorithm to work on random textures by averaging the spectrogram.

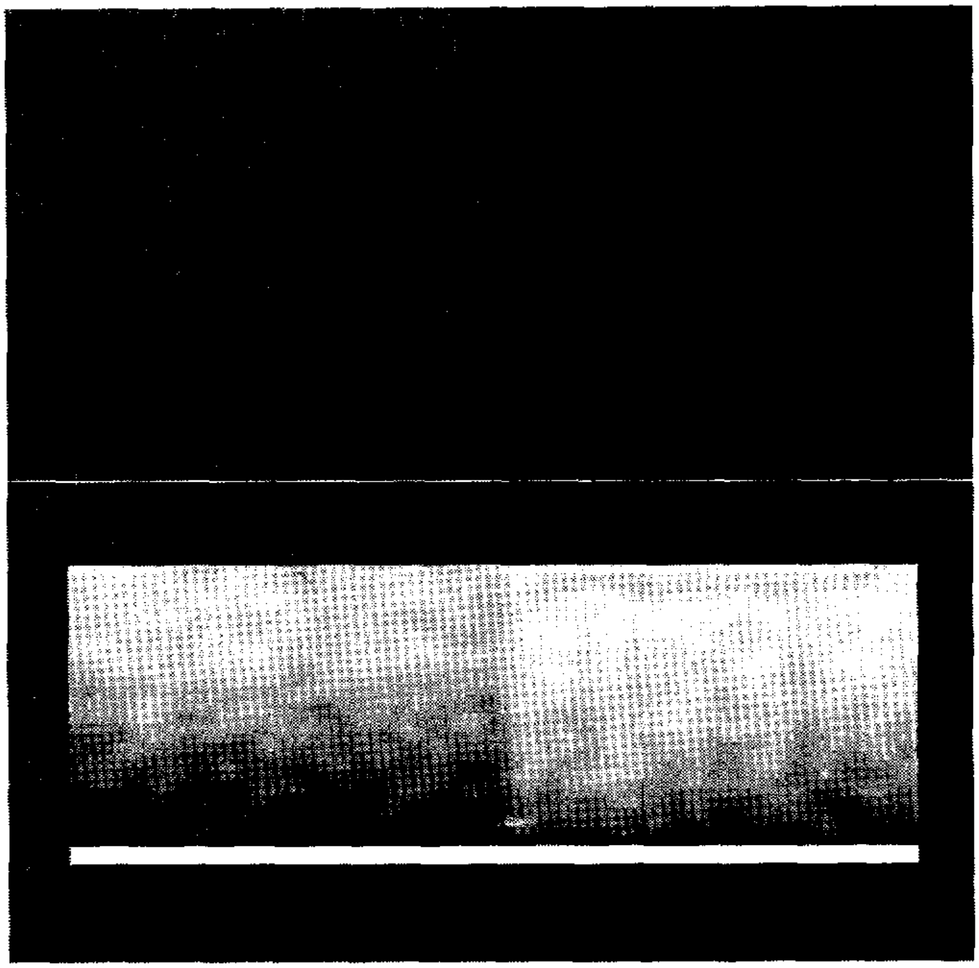


Figure 4: **Two random textures and the average spectrogram of the 21 center rows. Less averaging (as compared with Figure 3) leads to a noisier spectrogram.**

Figure 7 shows an interesting correspondence between an object's vanishing line and its spectrogram. Perspective effects impose a vanishing line on the plate. Moving to the right along the plate, the line of sight comes closer and closer to being parallel with the plate. None of the plate can project onto the image plane past this parallel line of sight. This column in the image is the vanishing line of the plate.

This plate's texture is a simple sinusoid, so the spectrogram shows only a single peak in frequency. The projected frequency increases without bound up to the vanishing line. Thus, the vanishing line for the plate is also an asymptote for the plate's frequency peak. In practice, however, the spectrogram is bounded above by the Nyquist limit, so the sinusoid is aliased after a certain point. This is shown in the actual spectrogram in Figure 7 as the fuzz just to the left of the vanishing line. We demonstrate some details of aliasing and the spectrogram in the next section.

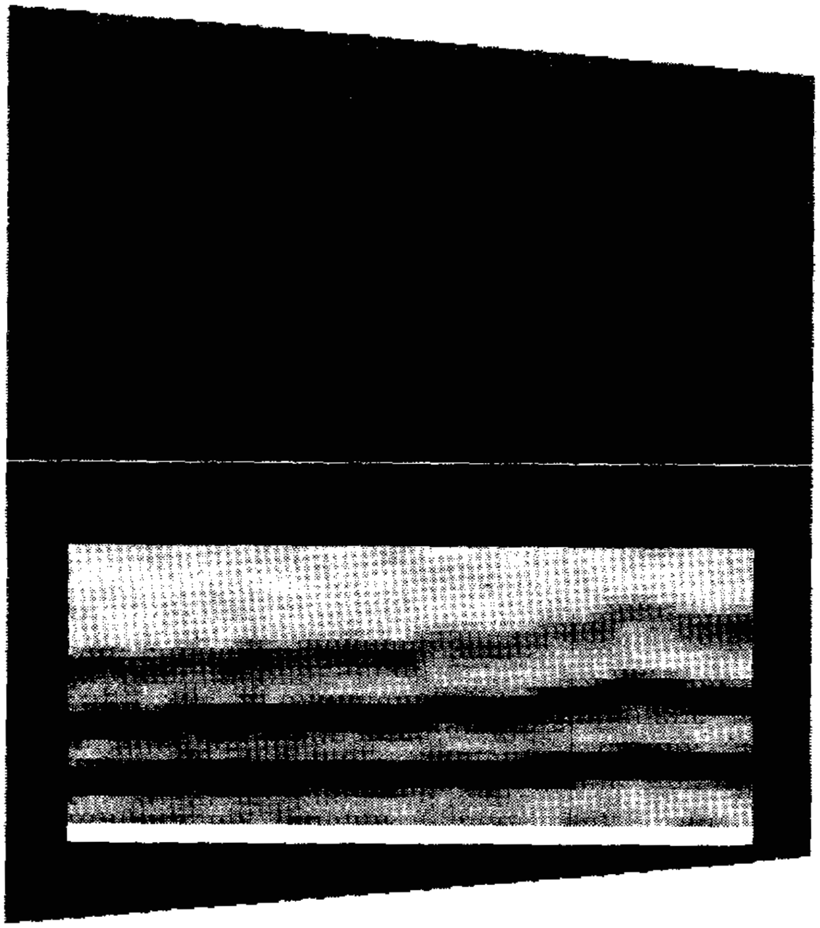


Figure 5: The surface normal and changing depth combine to cause a frequency gradient on this receding, textured plate.

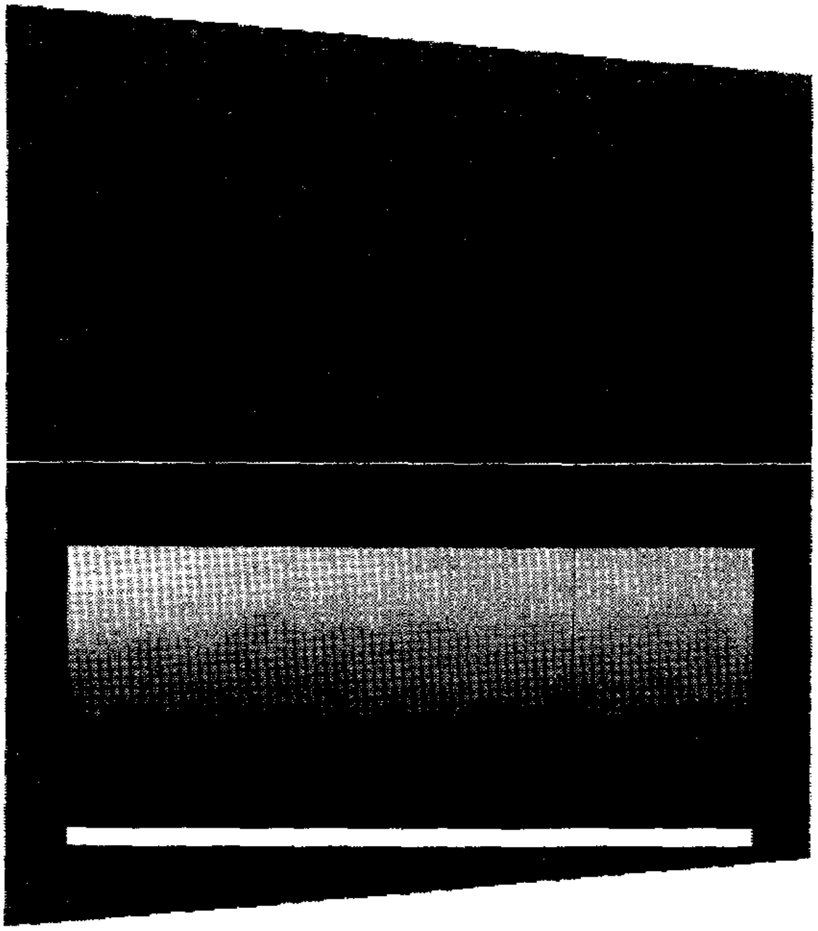


Figure 6: The frequency shifts due to 3D *effects for this* random texture are more difficult to detect.

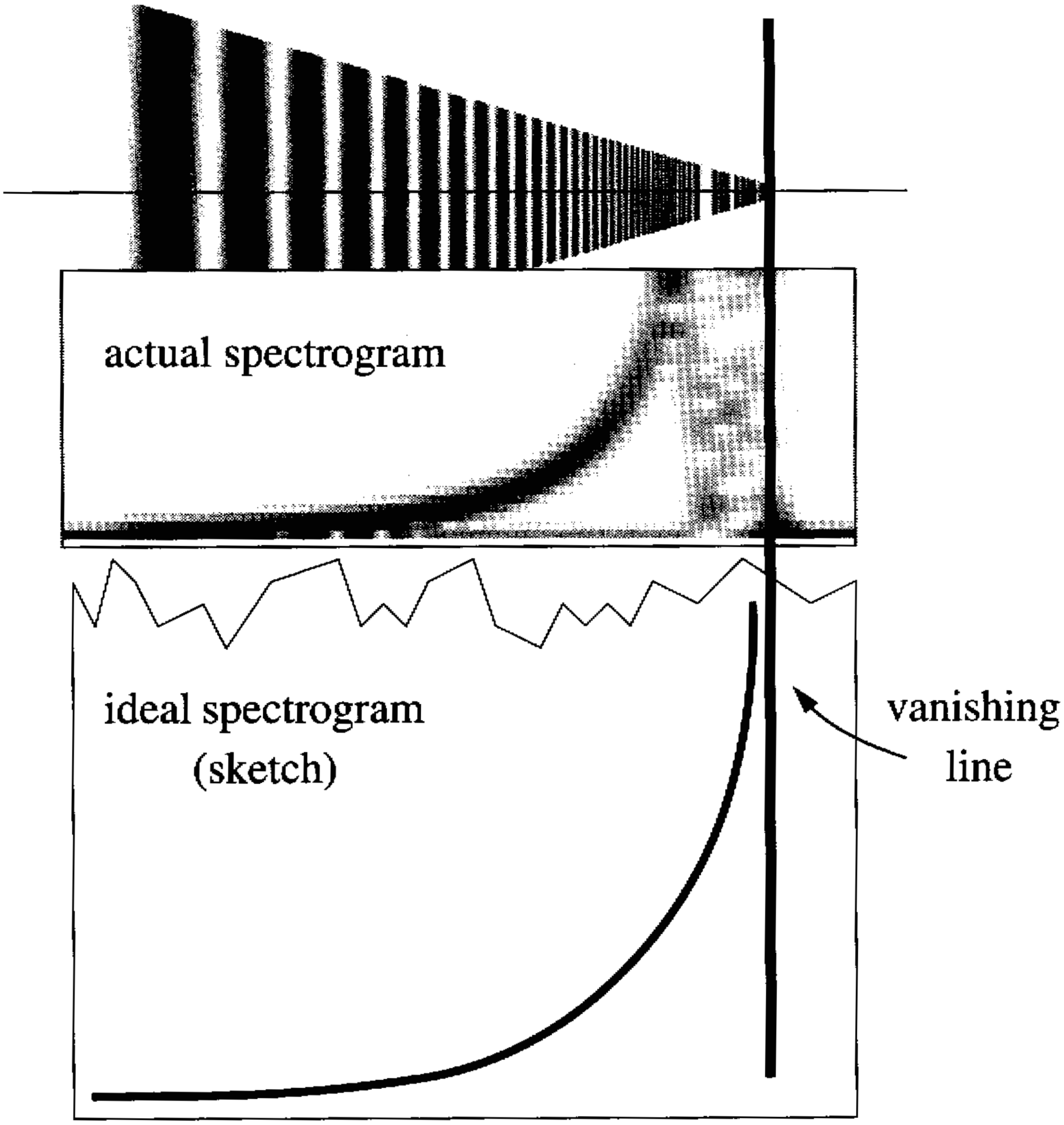


Figure 7: The vanishing line in the image is also an asymptote in the spectrogram. The actual spectrogram has an upper bound in frequency due to the Nyquist limit.

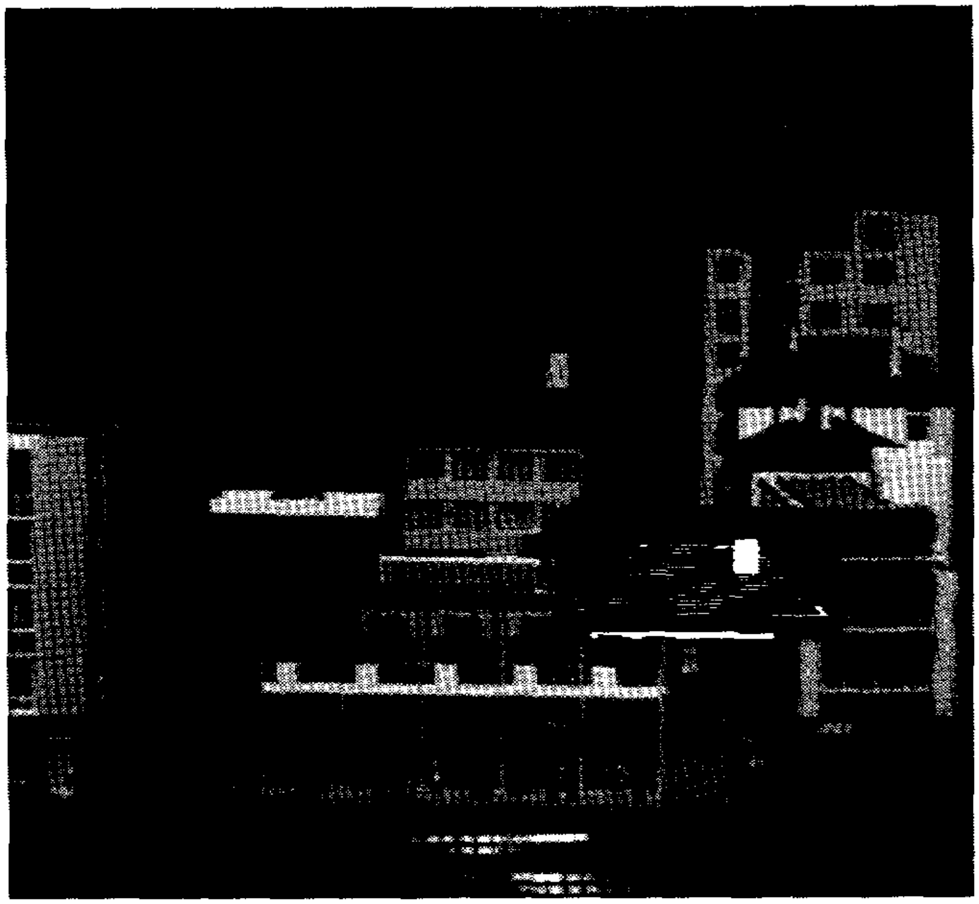


Figure 8: A moire pattern due to aliasing was accidently produced on the roof of the house in this image. The inset shows the pattern enhanced by thresholding.

1.3.3 Aliasing

Aliasing in digital images is caused by the finite spatial sampling rate of the image sensor. It is common for digital images to have some aliasing, because camera operators rarely take steps to avoid it. An example of accidental aliasing is shown in Figure 8. This is an image of a scale model village taken in the Calibrated Imaging Laboratory at Carnegie Mellon University. The shingles on the roof of the building in the center of the image were so close together that they made a moire pattern caused by aliasing. The binary inset is a thresholded region of this image that shows the low-frequency, sinusoidal moire pattern whose ridges are at about 30° from the horizontal.

In most texture models, it is not clear how aliasing can be expressed, but in terms of the spectrogram, it has definite, simple structure. Figure 9 shows a plate with a sinusoidal texture and the spectrogram of the center row. According to the Nyquist limit, the spectrogram cannot show any frequencies above 0.5 cycles/pixel. Thus, the frequency range of all the spectrograms in this chapter

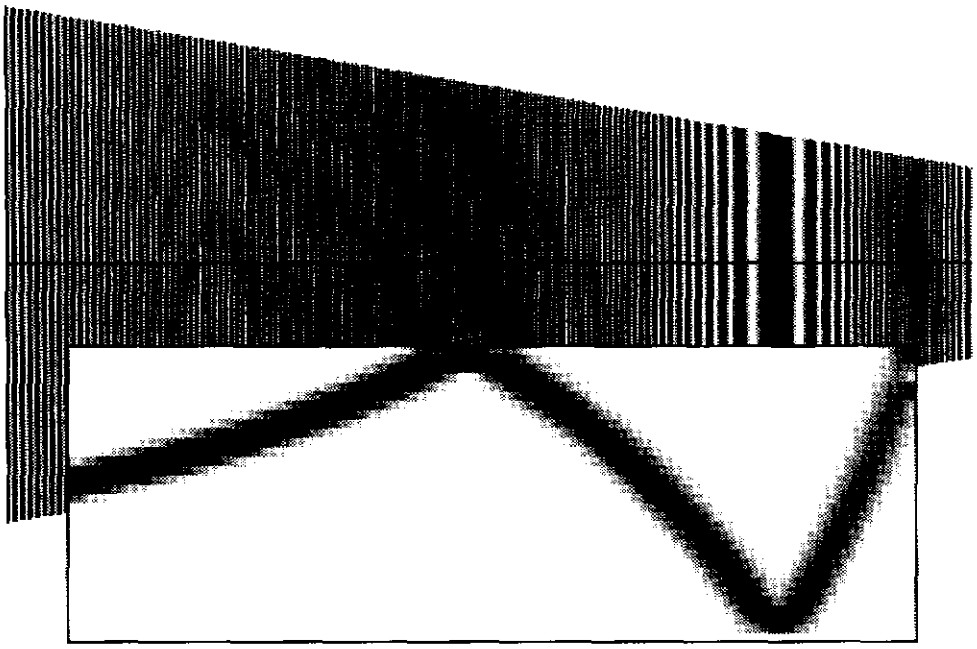


Figure 9: Aliasing causes the frequency peak to bounce in the spectrogram.

are $0 \le u \le 0.5$. When the projected sinusoid's frequency exceeds 0.5 cycles/pixel, aliasing occurs. The frequency peak appears to bounce off the top of the spectrogram and head toward zero. It bounces again and goes back up. This bouncing would continue if the plate were longer. Everything on the plate to the right of the first bounce is aliased, and the spectrogram is showing the frequency of the resulting moire pattern.

Figure 10 shows a graphical explanation of this bouncing. On the left are eight sinusoids of increasing frequencies, starting at 0.15 cycles/pixel and ending at 0.85 cycles/pixel. Each of these sinusoids was sampled at unit intervals (every pixel) and then reconstructed using standard $\mathrm{sinc}\,(x) = \sin(\pi x) / (\pi x)$ function interpolation. As long as the original frequency is below the Nyquist limit of 0.5, the reconstruction is perfect. But, any frequency above this limit causes aliasing, and the reconstructed sinusoid has a lower frequency than the original. In fact, the frequencies of the four aliased sinusoids are a mirror image of the four unaliased sinusoids. If we tried to reconstruct even higher frequency sinusoids, the aliased frequencies would oscillate between zero and one-half, like in the spectrogram.

To explain the bouncing mathematically, we imagine that the texture brightness is a simple sinusoid, $f(x) = \cos(2\pi u_o x)$. Its Fourier transform is a pair of delta functions placed symmetrically in frequency, $F(u) = 0.5\,[\delta(u - u_0) + \delta(u + u_0)]$. Sampling at a frequency of u_x cycles per unit

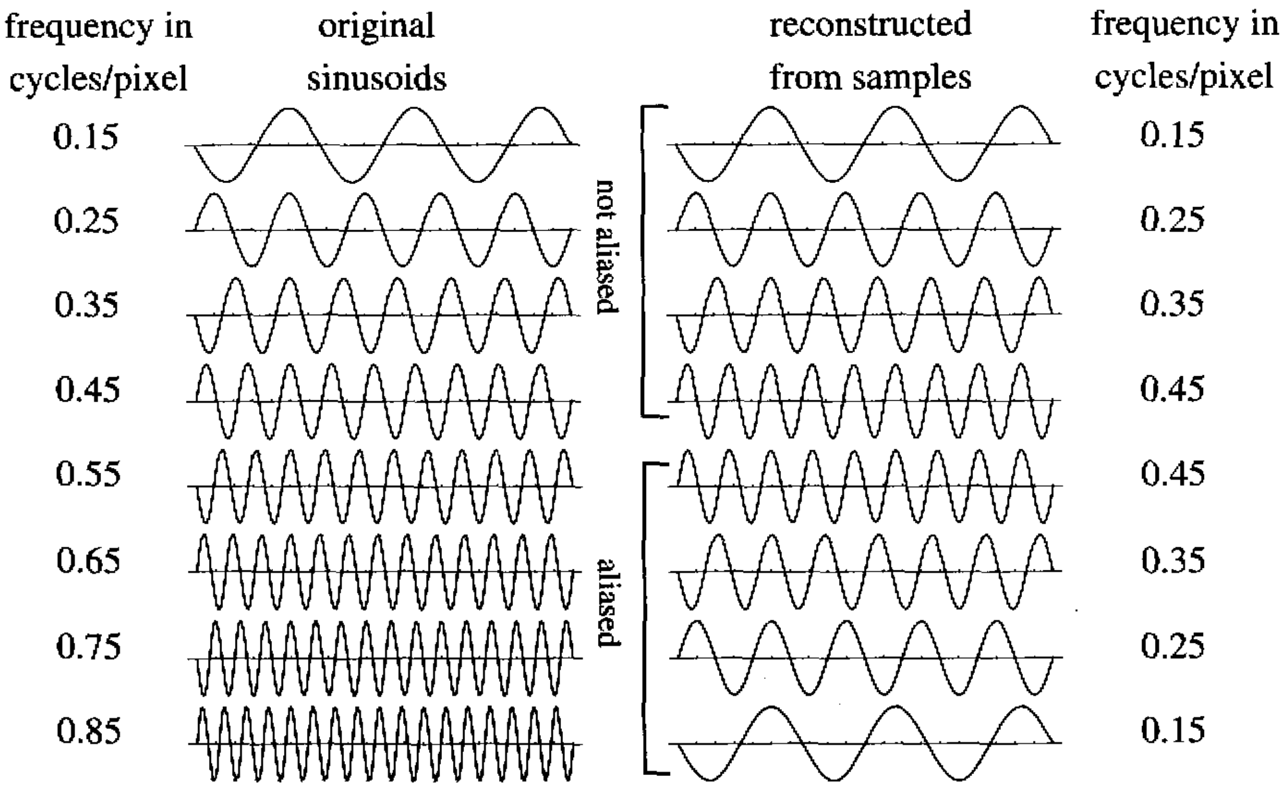


Figure 10: Each of the sinusoids on the left was sampled at a rate of one sample per unit distance. The ticks mark the sampling points. Any sinusoid with a frequency above 0.5 cycles per unit distance is aliased, and it appears as a lower frequency sinusoid.

distance is modeled in the spatial domain by multiplying the function by a series of equally-spaced delta functions. The sampled signal is

$$f_s(x) = \cos(2\pi u_o x) \sum_{o_{ss} = -\infty}^{\infty} \delta(x - o_s/u_s) \qquad (8)$$

Its Fourier transform is the convolution of the Fourier transform of the signal and the Fourier transform of the sum of delta functions. This is

$$F_x(u) = \{0.5[\delta(u - u_0) + \delta(u + u_0)]\} *u_s \sum_{o_s = -\infty}^{\infty} \delta(u - o_s u_s)$$

$$= 0.5u_s \left\{ \sum_{o_s = -\infty}^{\infty} [\delta(u - u_0 - o_s u_s) + \delta(u - u_0 - o_s u_s)] \right\} \qquad (9)$$

This is a series of repeated versions of the Fourier transform of the original function. The repetitions are indexed by o_s, which corresponds to the "spectral order" of the repetition. The locations of these delta functions are illustrated in Figure 11. Since the spectrogram only covers frequencies from zero to $u_s/2$, it shows different spectral orders depending of the frequency of the original signal. If the spectral order is zero, there is no aliasing. If we happen to know the spectral order, the location of the frequency in the spectrogram will be

$$u = \begin{cases} u_0 & \text{if } o_s = 0 \\ o_s u_s - \operatorname{sgn}(o_s) u_0 & \text{if } o_s \neq 0 \end{cases} \tag{10}$$

where

$$\operatorname{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{11}$$

If there is more than one sinusoidal component in the signal, the individual frequency peaks will bounce independently in the spectrogram. This is shown in Figure 12, where the plate's texture is the sum of two sinusoids. Most periodic textures are composed of a fundamental frequency and overtones at integer multiples of the fundamental. A good example is Brodatz woven aluminum wire (D6) shown mapped onto a receding plate in Figure 13. As the fundamental frequency grows, the top harmonic is the first to bounce, then the second from the top, and so on. Most other texture models would have a hard time accounting for aliasing. It is easy to account for it with the spectrogram. In Section 3.5.3 we show how one of our spectrogram-based shape-from-texture algorithms can work in spite of aliasing.

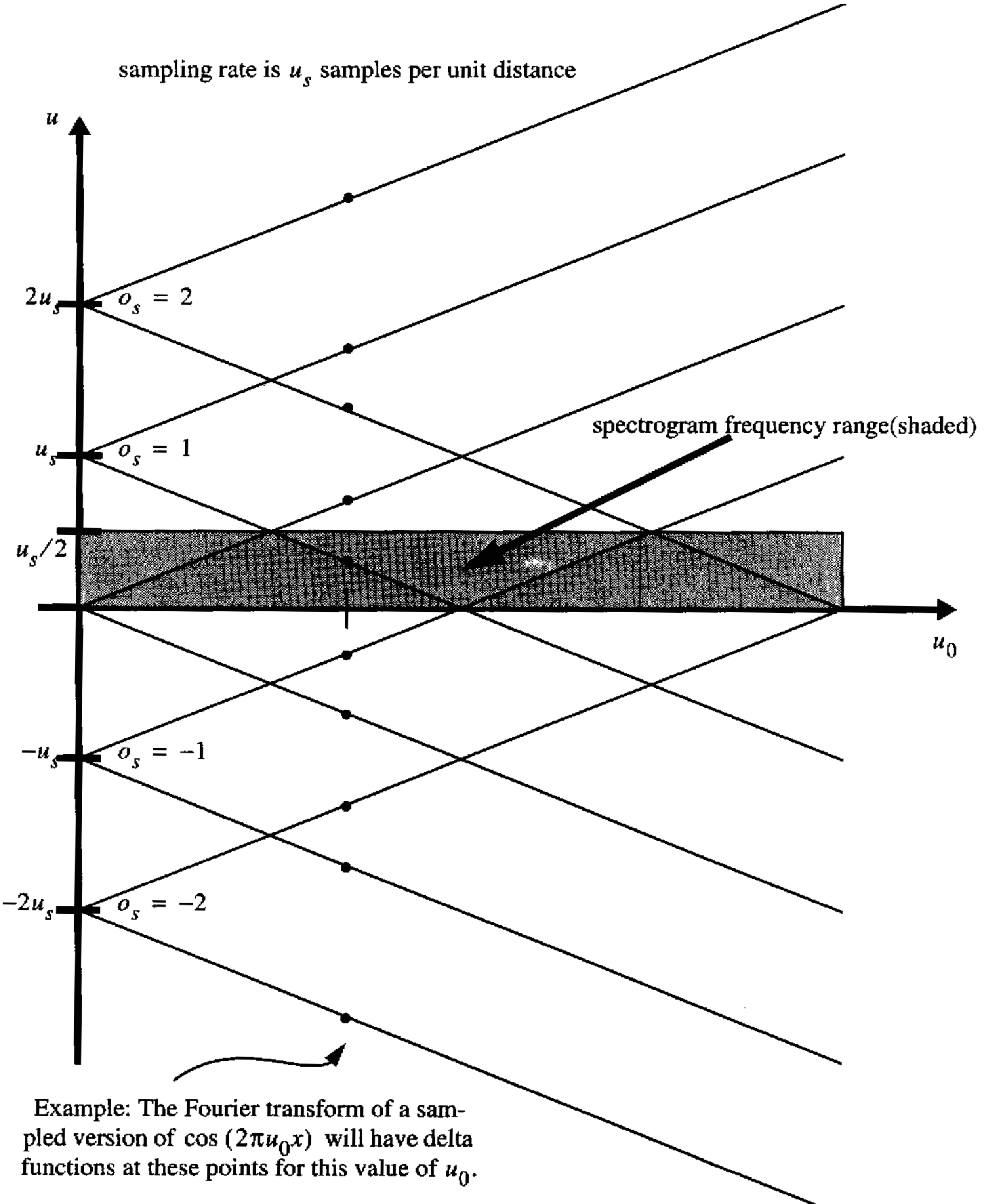

sampling rate is u_s samples per unit distance
u
$2u_s$
$o_s = 2$
u_s
$o_s = 1$
spectrogram frequency range(shaded)
$u_s/2$
u_0
$-u_s$
$o_s = -1$
$-2u_s$
$o_s = -2$
Example: The Fourier transform of a sampled version of $\cos(2\pi u_0 x)$ will have delta functions at these points for this value of u_0.

Figure 11: **For a single sinusoid of frequency u_0, this plot shows where the delta functions of its spectral orders will fall for increasing values of u_0. The frequency range covered by the spectrogram intersects different spectral orders depending on the value of u_0. This explains the bouncing frequency peaks in the spectrogram.**

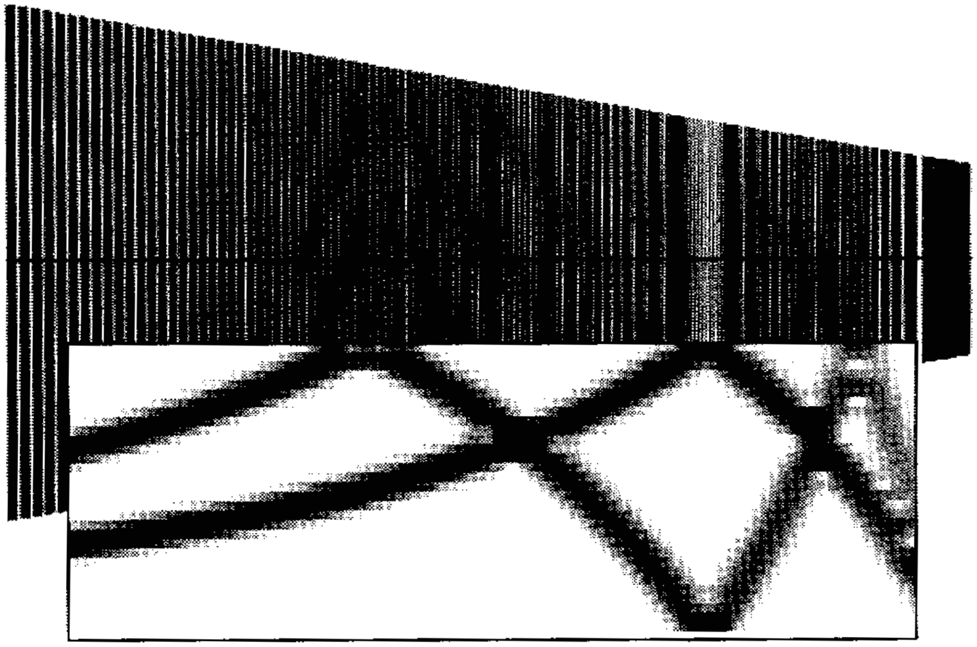


Figure 12: Different sinusoidal components bounce independently of each other when they are aliased.

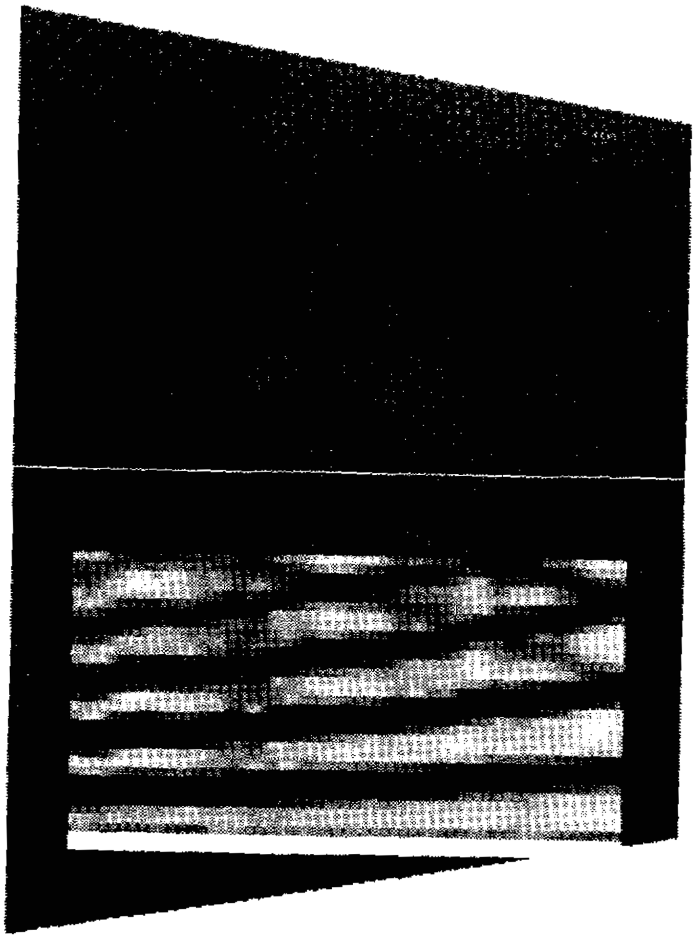


Figure 13: The highest harmonics in a periodic texture are aliased first, then the next highest and so on.

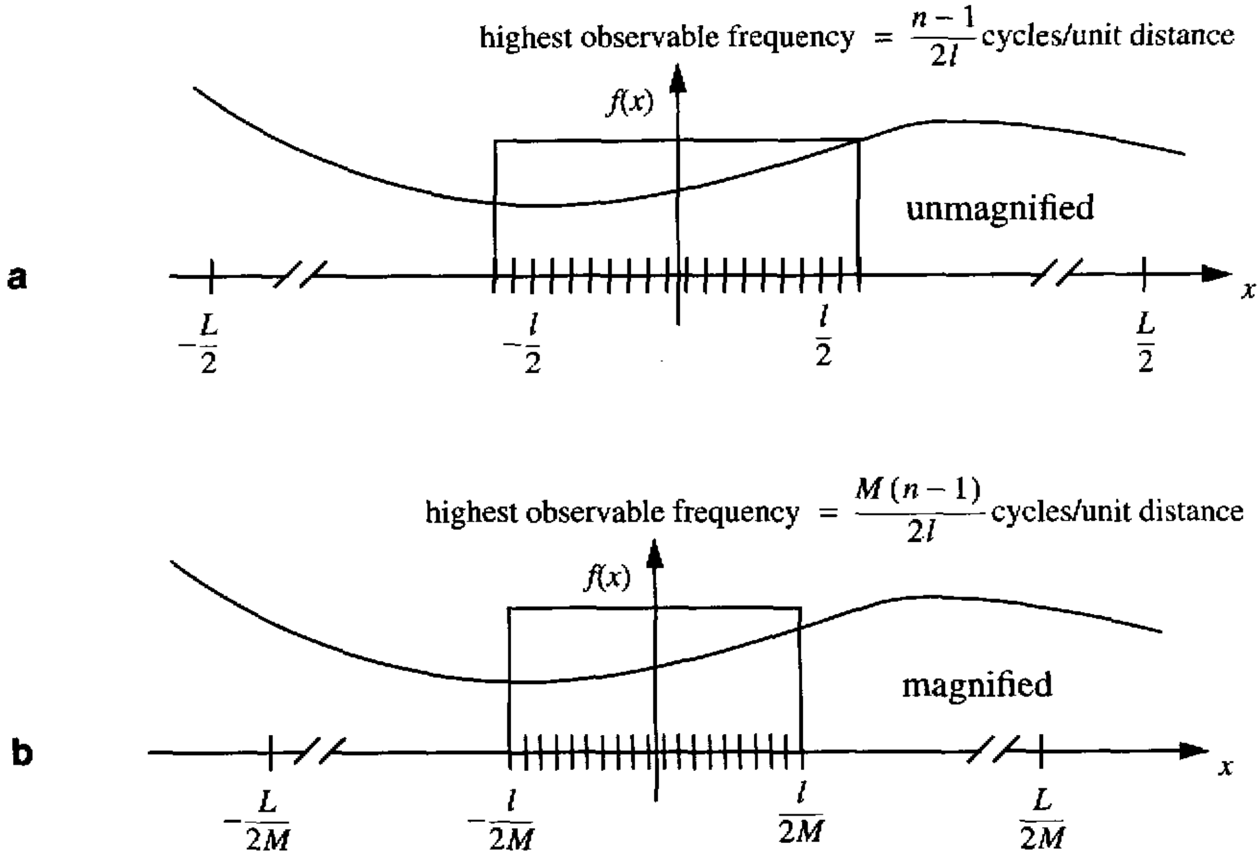


Figure 14: Effect of zooming an imaged signal by a factor of M.

1.3.4 Zoom

Much research in "active vision" concerns the control of the three lens parameters: zoom, focus, and aperture. We show in this section and the next section how these parameters affect the spectrogram, which in turn provides new insights into how they affect the image. This point of view can lead to algorithms that let us deduce intrinsic scene parameters by purposefully altering the lens settings.

In equifocal camera lenses (such as most one-touch zoom lenses) a change in zoom can be modeled as simply a change in magnification. We can imagine the situation in Figure 14a, where the section of the signal that falls on the center window of the spectrogram extends from $-l/2$ to $l/2$. We will arbitrarily call the magnification here one, and we will say that the entire portion of the signal seen by the camera is of length L. Both l and L are measured on the image plane. If there are n pixels in the spectrogram window, the sampling frequency is $(n-1)/l$ pixels per unit distance, making the highest observable frequency $(n-1)/(2l)$. The spectrogram resulting from this signal will cover the region indicated by the short, wide box in Figure 15.

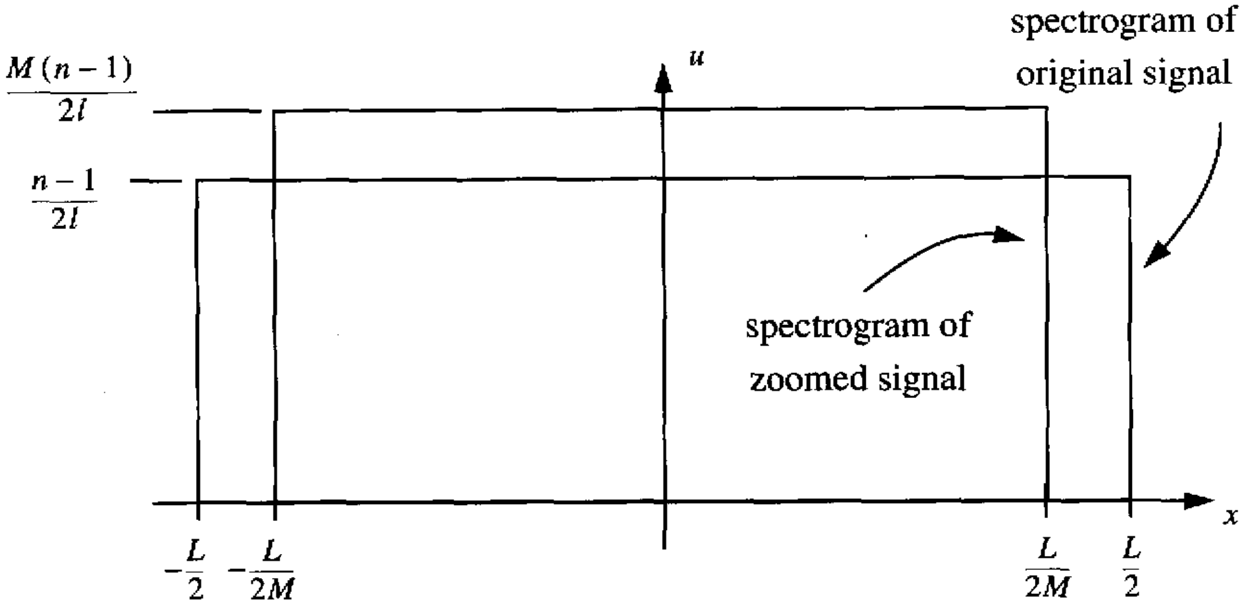


Figure 15: Zoom changes the region covered by the spectrogram in space/frequency but not the area.

Figure 14b shows what happens to the sampling rate after the magnification is changed from one to M. The n samples in the center spectrogram window are spread over a distance of l/M, meaning that the highest frequency in the spectrogram is $M(n-1)/(2L)$. The entire signal seen by the camera goes from $-l/(2M)$ to $l/(2M)$.

The spectrogram after the magnification change is shown in Figure 15. For an increase in magnification, the spectrogram covers more in frequency but less in space. The "area" of the spectrogram (actually a unitless quantity, "spatial dynamic range") is $(L(n-1))/(2l)$ and is independent of the magnification. Thus for changes in zoom, there is a direct trade-off between coverage in space and spatial frequency. These arguments also apply to the four-dimensional hypervolume of the spectrogram of a two-dimensional signal.

Figure 16 shows a horizontally split image of a textured plate at two different zoom settings. The lower half was zoomed about 1.09 times the upper half. The spectrograms of the center rows of both are shown in the same figure. Until the first aliasing-induced bounce, the peak frequency is in the same location in both spectrograms. However, once aliasing sets in, the peak shifts. This is because the position of the aliased peak is a function of the sampling rate.

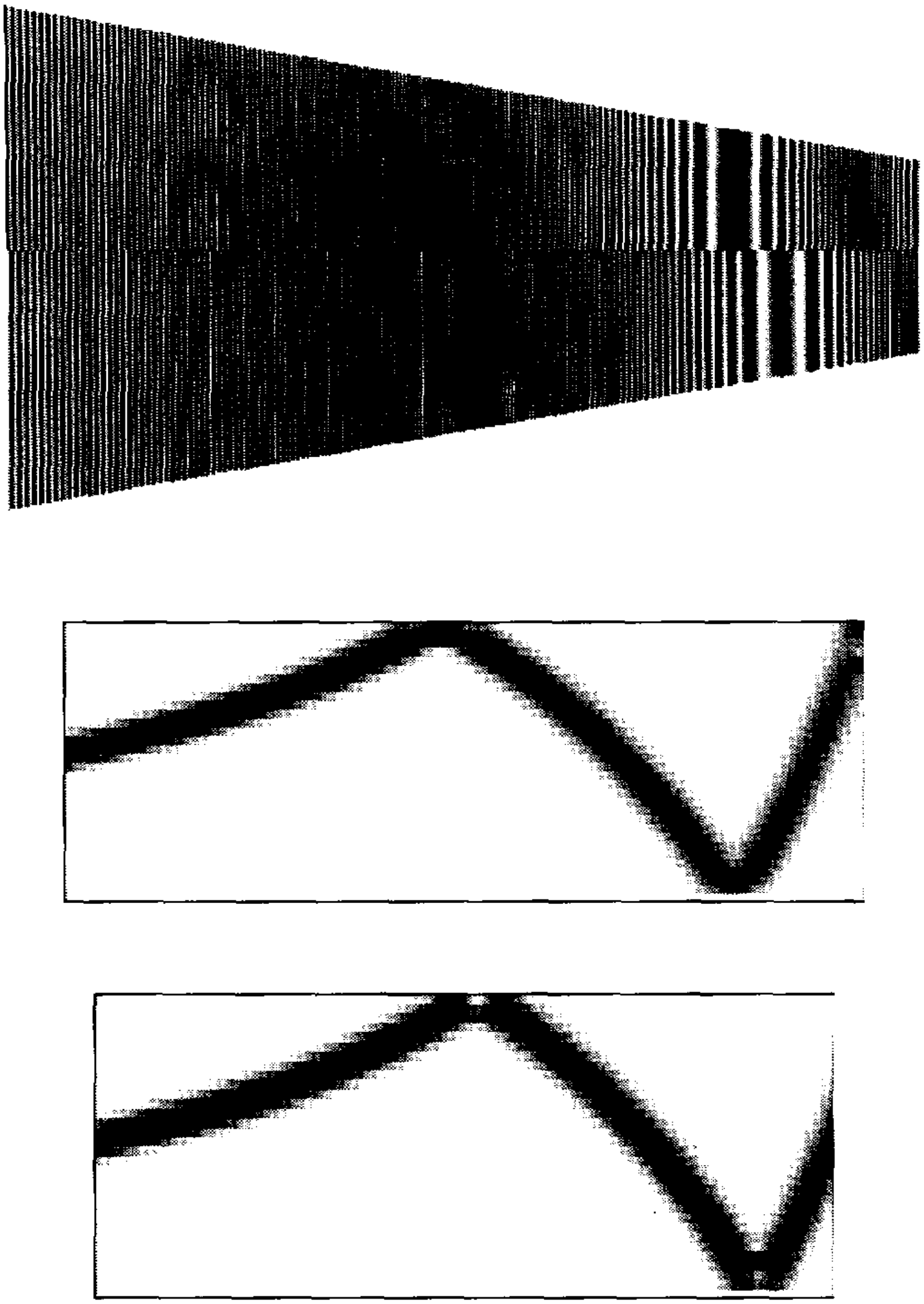


Figure 16: Two plates taken at slightly different zoom settings. Aliased peaks are shifted with respect to each other in the spectrograms.

1.3.5 Dealiasing with Zoom

We will briefly depart from the series of pictures here to develop a small algorithm for dealiasing simple signals using the spectrogram. It depends on the explanation of aliasing and zoom in the previous two sections. The method works by using two images of the signal at slightly different zoom settings.

Suppose as above that we have a sinusoid $\cos(2\pi u_o x)$, whose frequency u_0 cycles per unit distance is sampled at a rate of u_s samples per unit distance. The signal may be sampled above or below the Nyquist rate. Referring to Figure 11, we can see there will be only one spectral order contributing to the spectrogram for any frequency u_0. We will arbitrary say that the magnification of the first image of the signal is $M = 1$. If the spectral order of the signal is o_s, its apparent (possibly aliased) frequency u_1 seen in the spectrogram of the first image will be given by Equation (*10*):

$$u_1 = \begin{cases} u_0 & \text{if } o_s = 0 \\ o_s u_s - \operatorname{sgn}(o_s)\, u_0 & \text{if } o_s \neq 0 \end{cases} \tag{12}$$

If the same signal is re-imaged at a slightly different magnification M, then the effective sampling rate measured in cycles per unit distance of the *unmagnified* image will be Mu_s. The apparent frequency u_2 in the spectrogram will then be

$$u_2 = \begin{cases} u_0 & \text{if } o_s = 0 \\ o_s M u_s - \operatorname{sgn}(o_s)\, u_0 & \text{if } o_s \neq 0 \end{cases} \tag{13}$$

We can eliminate the unknown frequency of the original signal by subtracting Equation (12) from Equation (13). Solving this difference for the spectral order gives

$$o_s = \frac{u_2 - u_1}{u_s (M - 1)} \tag{14}$$

This equation applies for both $o_s = 0$ and $o_s \neq 0$. Thus, the difference in apparent frequency between the two images is proportional to the spectral order. After solving for o_s, we can use Equation (12) or (13) to solve for u_0, which is the true frequency of the signal. The dealiasing does not require the solution of a correspondence problem, since the two signals are related by a simple difference in magnification.

An implicit assumption here is that o_s remains the same in both images. This will be true for small changes in magnification unless the frequency peak is very close to either extreme of the spectrogram's frequency range.

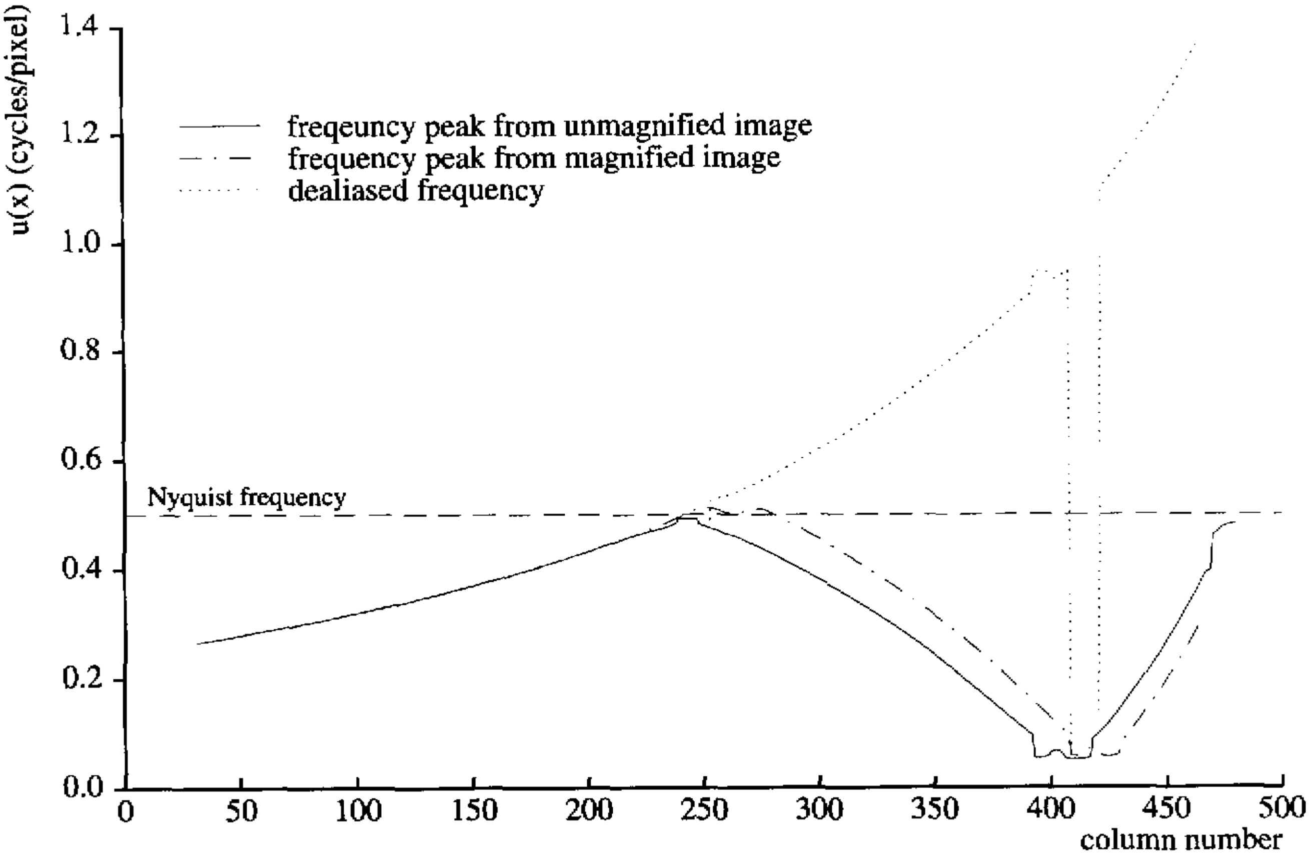


Figure 17: The differences in the aliased frequency peaks can be use to find the true, unaliased frequency of the signal, no matter what its frequency.

We have applied this technique to the two spectrograms of the receding plate in Figure 16. We extracted the peak frequency in each spectrogram, and these are plotted in Figure 17. The frequency data from the magnified image has been adjusted so it is shown in terms of the space and frequency units of the unmagnified image. The dotted line shows the dealiased frequency based on the technique outlined above. Except for the glitches at the frequency extremes, the figure shows correctly the dealiased frequency. Thus, the spectrogram has been dealiased without detailed *a priori* knowledge of the scene. It could be argued that with twice the number of samples (because we have two images), it is of course possible to double the Nyquist limit of a single image. This is true, but we have gone beyond twice the Nyquist limit. This is primarily because we limit the input to a single sinusoid. With more careful matching of frequencies between the two spectrograms, it should be possible to extend this algorithm to more general signals.

1.3.6 Focus and Aperture

Changes in the lens' focus and aperture combine to change the point spread function (psf) of the lens, which can be easily visualized with the spectrogram. (The psf is a function which is convoluted with an ideal, sharp signal to model the effects of blur.) In general, points in sharper focus will show more high frequencies than if they are blurred. For blurred parts of the image, a smaller aperture tends to have the same general effect as sharper focus. In fact, in the pinhole camera model, the aperture is infinitesimally small, meaning that every point in the scene is in perfect, sharp focus.

We can generalize the pinhole model to account for focus and aperture by introducing a single, thin lens with a variable aperture as shown in Figure 18. The aperture of the lens is a, the focal length of the lens is b, and the distance to the image plane is d. We can approximate the effects of focus and aperture with geometric optics. Each point in the scene with a different value of Z_o will be in sharp focus at only one point behind the lens. This point, Z_i, is given by the Gaussian Lens Law:

$$\frac{1}{Z_i} + \frac{1}{-Z_o} = \frac{1}{b} \qquad (15)$$

If the image plane is not at the proper distance behind the lens for a given point, *i.e.* $d \neq Z_i$, the point will be spread into a blur circle. Using geometric optics, the radius of the blur circle is given by[96]

$$r(Z_o) = \frac{ad}{2}\left|\frac{1}{b} - \frac{1}{d} + \frac{1}{Z_o}\right| \qquad (16)$$

A point can be out of focus by having the image plane in front of or behind the point of best focus. The equation above applies to both cases. $r(Z_o)$ goes to zero when $1/d - 1/Z_i = 1/b$, which is a restatement of the Gaussian Lens Law above. In the one-dimensional imaging case illustrated here, the shape of the blur "circle" is actually a rectangle of width $2r(Z_o)$. Thus, the point spread function of the 1D camera system is

$$h(x, Z_o) = \frac{1}{2r(Z_o)}\text{rect}\left(\frac{x}{2r(Z_o)}\right) \qquad (17)$$

where we have normalized so the area under the $\text{rect}(x)$ is one. [3] The corresponding transfer function, H, is the Fourier transform of h with respect to x:

[3] This psf ignores three optical effects. One is diffraction, whose magnitude is much smaller than defocus effects in typical TV images. The second is the fact that points which are occluded in the pinhole image can actually be seen by parts of the lens in an image with a finite aperture. The third is that, by normalizing the area of the psf to one, we are ignoring the most obvious effect of a change in aperture: a change in the overall brightness of the image.

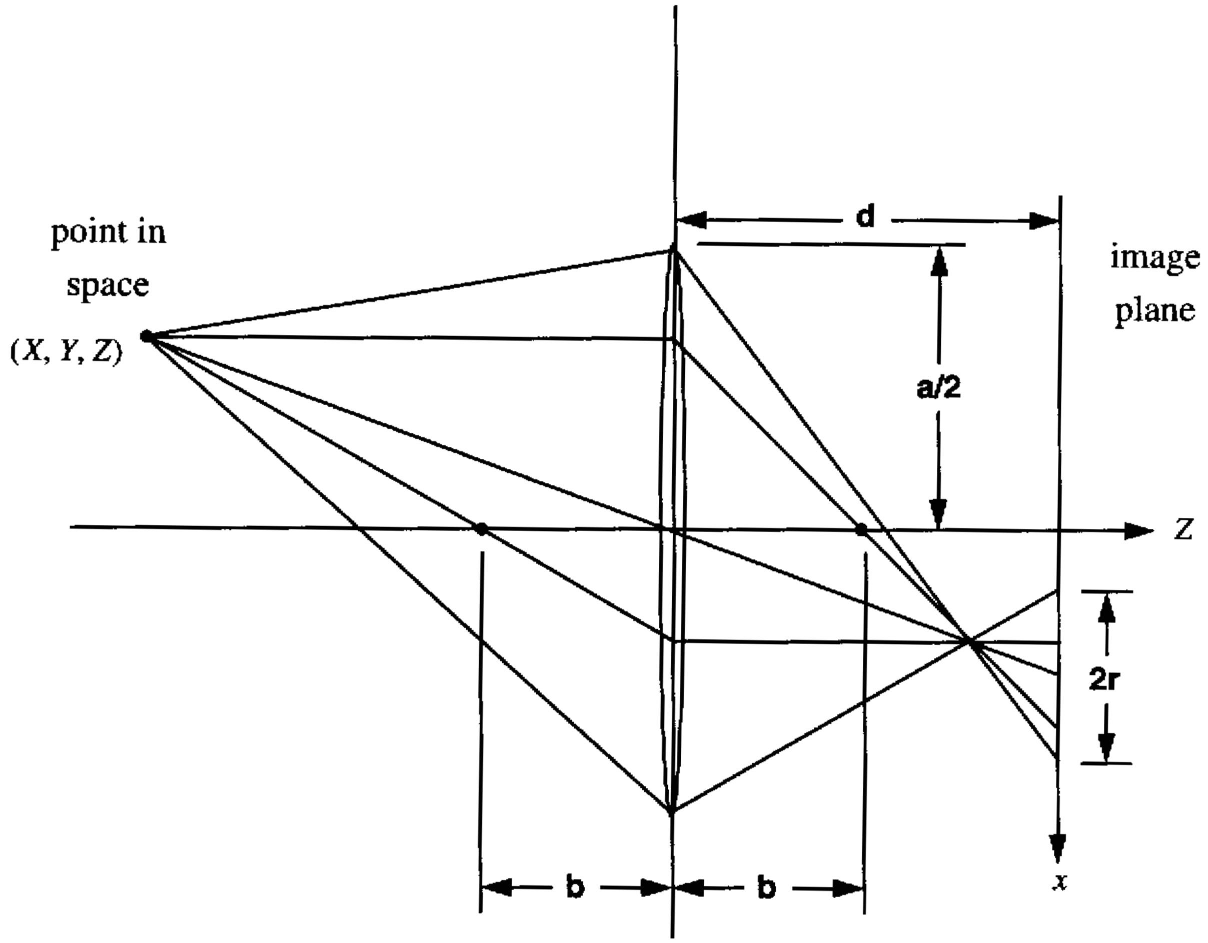


Figure 18: Thin lens camera model for deriving effects of focus and aperture.

$$H(u, Z_o) = \text{sinc}\,[2ur(Z_o)] \tag{18}$$

In order to calculate the effect of $h(x, Z_o)$ on the spectrogram, we suppose there exists a function $f(x)$ which is an unblurred, pinhole projection of the scene. The new image, $f_h(x)$, taking into account the point spread function, is a convolution of the unblurred image with h. Thus,

$$f_h(x) = \int_{-\infty}^{\infty} f(\alpha)h(x-\alpha, Z_o(x))d\alpha \tag{19}$$

where the $Z_o(x)$ is the depth corresponding to x on the image plane. This equation holds for changes in the camera's aperture. It does not apply for change in the focus distance d, because this causes a change in magnification as well as a change in the point spread function.

The point spread function h is *not* space-invariant, because it depends on the depth of the surface. This means that the Fourier convolution theorem does not apply to the Fourier transform of the whole signal. If h were space-invariant, *e.g.* if the depth of the scene were constant, then the effect on the spectrogram would be simple to describe: each windowed Fourier transform would be multiplied by the Fourier transform of the point spread function. This is also approximately true for the space-variant point spread function if the surface depth varies slowly and/or the window used for the spectrogram is small. The spectrogram of the unblurred signal is

$$S(x, u) = \left| [e^{-j2\pi ux} W(u)] * F(u) \right|^2 \quad (20)$$

Then the spectrogram of the blurred signal is

$$S(x, u) = \left| \{ [e^{-j2\pi ux} W(u)] * F(u) \} H(u, \overline{Z_0}(x)) \right|^2 \quad (21)$$

where $\overline{Z_0}(x)$ is a representative depth value for the region centered at x. Each windowed Fourier transform has associated with it its own transfer function that depends on the approximate depth of the region within the window. As the blur circle becomes larger, its Fourier transform $H(u, Z)$ becomes narrower in frequency and attenuates more of the high frequencies.

This is the approximation used for most depth-from-focus and depth-from-defocus algorithms in computer vision. The spectrogram can be used as a criterion function to calculate the point of best focus over several images taken at different focus settings, as shown by Krotkov[54]. The setting closest to perfect focus is the one that gives the most high frequency energy in the spectrogram at that point. Knowing this setting, along with a precalibrated table of focus distances, the depth to all points in the scene can be calculated. Pentland[84] uses a spectrogram, essentially, to calculate depth from defocus based on only two focus settings. He uses the two spectrograms to directly calculate the depth to each scene point by calculating the width of the psf.

We can see the effects of this space-variant point spread function in the spectrogram. Figure 19 shows an example scene. It is Brodatz woven aluminum wire (D6) mapped onto a flat, receding plate. The spectrogram of the center row shows the frequencies increasing to the right due to the 3D effects. At the right side, the topmost harmonic is aliased, as evidenced by the bounce. This image was produced using a pinhole camera model, so every part is in sharp focus.

Figure 20 shows the same scene as in Figure 19, except the simulated camera had an aperture opening corresponding to f/4. The camera was focused on the left side of the plate, and the small depth-of-field means the right side is out of focus. The corresponding effect on the spectrogram is that the higher frequencies on the right are attenuated. The blurring here could even be considered benefi-

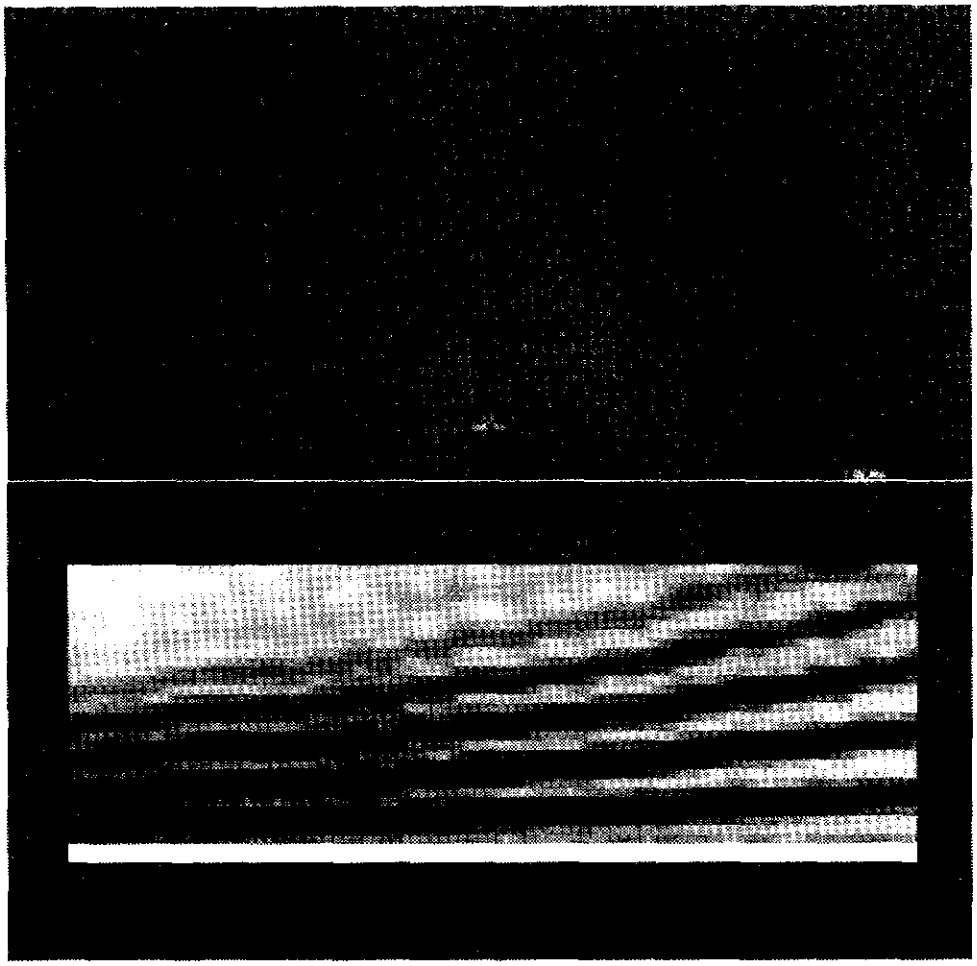


Figure 19: Everything in this image is in sharp focus, because it was produced with a pinhole camera model.

cial, because the aliased harmonic has been blurred away. Figure 20 shows the same scene with an aperture opening of f/2. The depth of field here is even smaller, and the attenuation is more severe.

Formulating the effects of the psf in terms of the spectrogram is a natural way to reason about the space-variant nature of the transfer function. For example, it reveals how precisely each point can be focused. Points in the scene with no high frequencies will never show high frequencies no matter how well they are focused, meaning that a focusing criterion function based on frequency would not be sensitive to such points. Another issue is the separation of the space-invariant part of the psf (due to, say, pixel averaging and the camera electronics) from the space variant part. It may be that the space-invariant psf is so large that depth effects are insignificant.

Matters of focus are very clear when formulated in terms of the spectrogram. Accounting for this effect with most other texture models would be difficult.

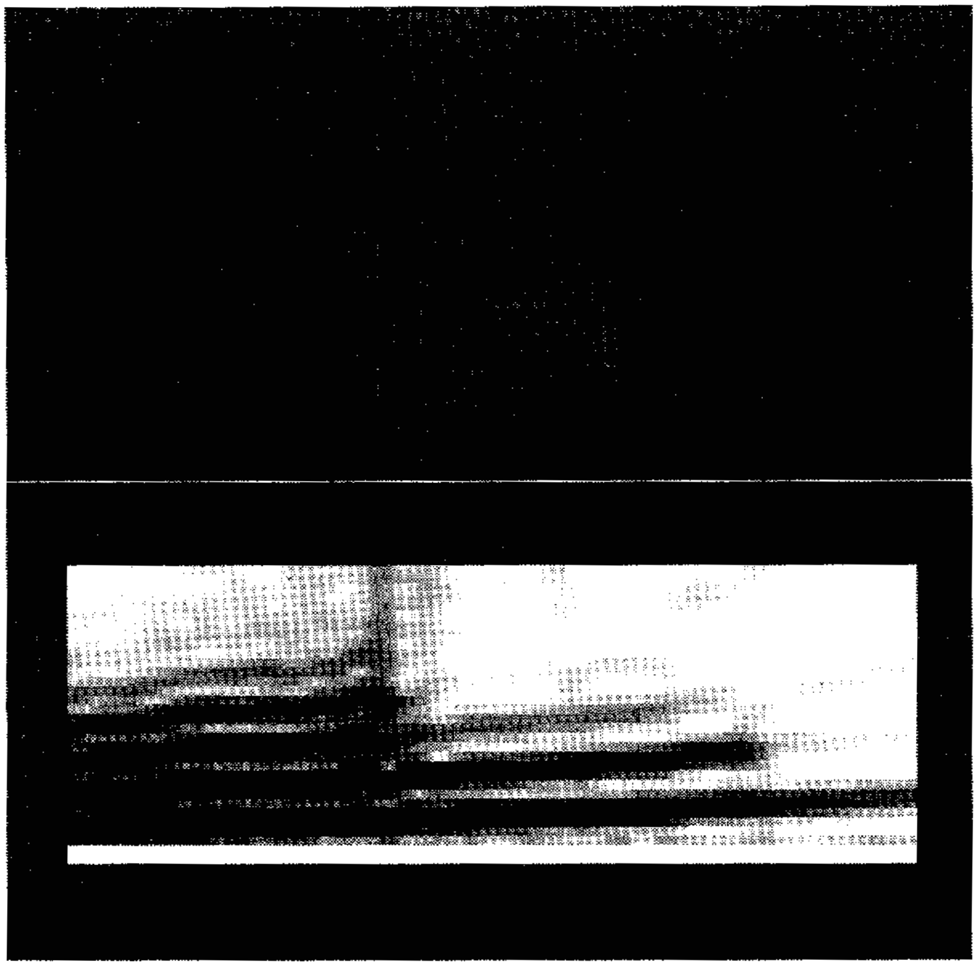


Figure 20: This is the same scene as in Figure 19, except the aperture is now f/4 instead of a pinhole. The camera is focused on the left side of the plate. The spectrogram shows the effect of the varying amount of blur due to the shallow depth of field.

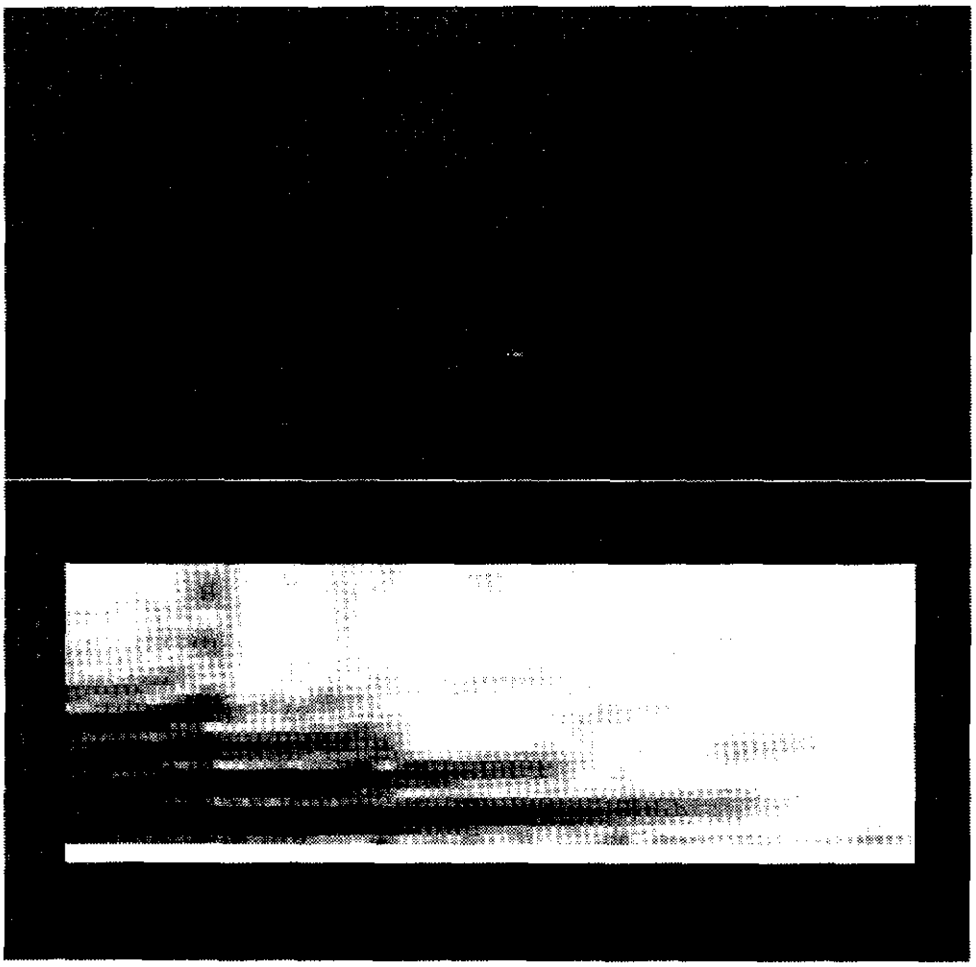


Figure 21: This is the same scene as in Figure 19, except the aperture is now f/ 2 instead of a pinhole. The camera is focused on the left side of the plate. The depth of field is even smaller than the f/4 image in Figure 20, and the spectrogram makes this obvious.

1.3.7 Summary

This section gives an idea of the versatility of the space/frequency representation for work in image texture. We showed that it is simple to express how the local spatial frequency content of an image texture can be used in algorithms for texture segmentation and shape-from-texture, and for the analysis of aliasing, zoom, and focus. An while we used the spectrogram to illustrate these effects, other space/frequency representations could be similarly adept. On the other hand, we cannot think of any non-frequency-based texture model that is this versatile.

Chapter 2 The Space/Frequency Representation

2.1 The Space/Frequency Representation

The space/frequency representation makes explicit a signal's behavior in frequency as a function of time or space. Signals whose frequency characteristics change over time or space are called nonstationary, and the traditional Fourier transform is inadequate for representing this behavior. Figure 22a shows an example of such a signal. It consists of two sinusoids - a sinusoid of 20 cycles/unit distance on the left, a sinusoid of 40 cycles/unit distance in the middle, and the sum of both on the right. The nonnegative side of this signal's Fourier power spectrum (squared magnitude of the Fourier transform) is shown below the signal. The power spectrum is a function of frequency u in cycles/unit distance. It shows peaks corresponding to the two constituent frequencies, but it does not show *where* they occur. This is the inadequacy of the Fourier transform for nonstationary signals: each point in the Fourier transform is a function of the signal for its entire duration, meaning that the Fourier transform destroys the time or spatial coherence of the input. Although all the information of the original signal is preserved in the Fourier transform, its time or space information is hidden in complicated phase relationships.

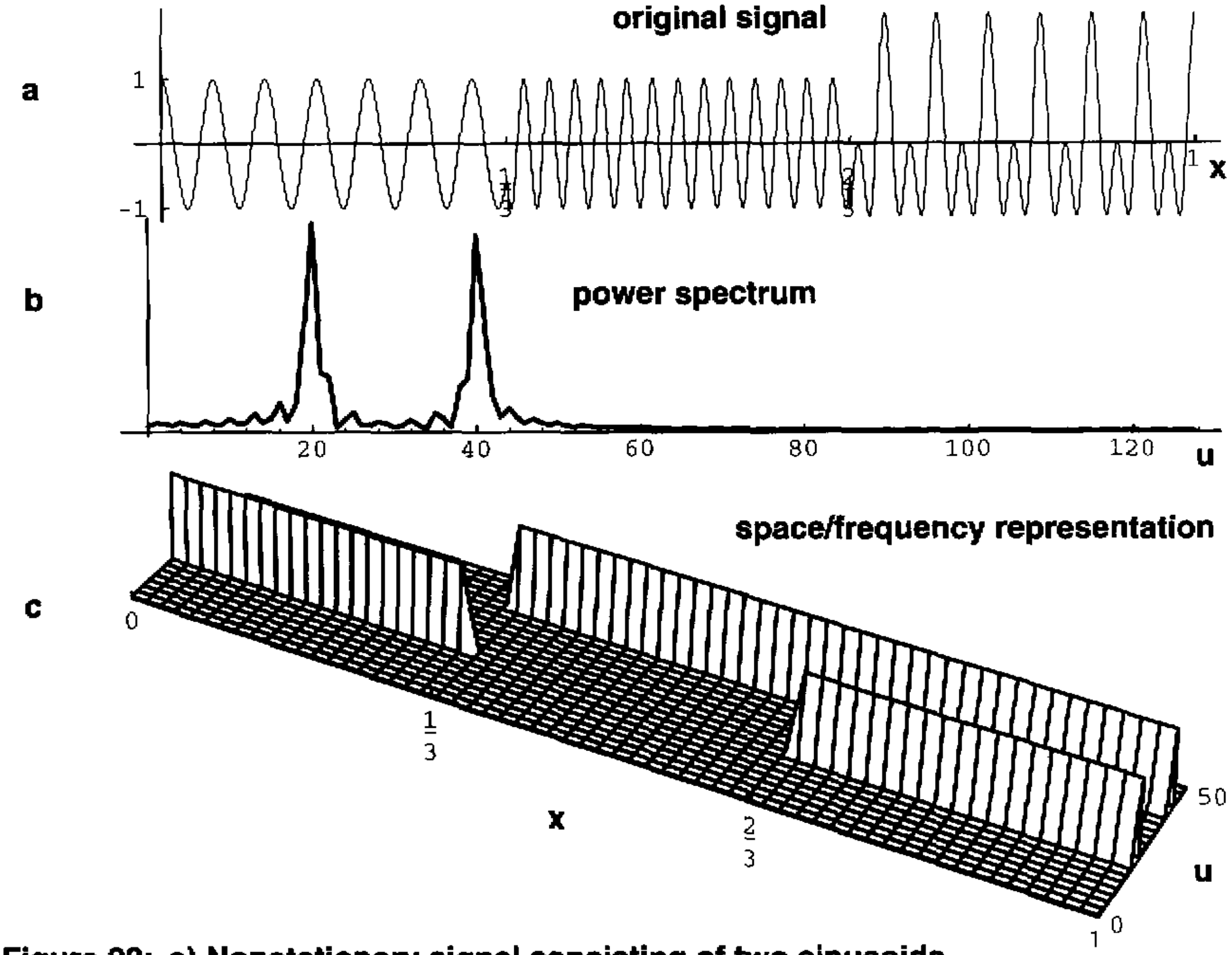


Figure 22: a) Nonstationary signal consisting of two sinusoids
b) Fourier power spectrum of signal
c) Idealized space/frequency representation of signal

A better representation of the nonstationary signal in Figure 22a is the space/frequency representation in Figure 22c. This is a two-dimensional function of both space and frequency, with the x axis running horizontally and the u axis running vertically. Each point in the x-u plane corresponds to one frequency at one point in space. It shows the local frequency characteristics of the signal. Thus, the ridge on the left is at 20 cycles/unit distance, and the ridge in the middle is at 40 cycles/unit distance. The right side shows both ridges, corresponding to both sinusoids. This is a much more intuitive representation of nonstationary signals, and it makes analysis much easier.

Unfortunately, the crisp space/frequency representation in Figure 22c is impossible to compute using any practical, generally applicable methods. Any real space/frequency representation computed from the original signal would have wider, smoother peaks and less sharp transitions from ridge to ridge. This is because no technique of decomposing a spatial function into space/frequency can achieve arbitrarily high resolution along both axes simultaneously.

The next section examines this issue more closely, showing that we can never get a perfect representation of general signals with standard techniques. We conclude, though, that we can do well enough if we remember the inherent limitations. The last section reviews several different methods of computing the space/frequency representation and explains why we chose the spectrogram.

2.2 Why It Cannot Be Perfect

The most common way of computing the space/frequency representation is with the "short time Fourier transform" (STFT), where the frequency distribution at a particular x_0 is computed with a Fourier transform of the signal in the neighborhood around that point. The implied assumption is that the signal is locally stationary. The neighborhood around x_0 is established by multiplying the signal $f(x)$ by a window centered at x_0, $w(x-x_0)$. This window is normally symmetric and falls to zero a short distance from its center. The space/frequency representation is then

$$\begin{aligned} SF(x_0, u) &= \int_{-\infty}^{\infty} w(x-x_0) f(x) e^{-j2\pi ux} dx \\ &= \Im_{x \Rightarrow u}\{w(x-x_0) f(x)\} \\ &= e^{-j2\pi x_0 u} W(u) * F(u) \end{aligned} \tag{22}$$

where $*$ means convolution. This can also be thought of as filtering $f(x)$ by sets of windowed sinusoids centered at various points in space/frequency. Of course, we need a way of analyzing 2D functions instead of 1D. We will concentrate on 1D signals in this chapter for purposes of illustration. All the principles we outline for 1D signals apply to 2D in a straightforward way.

In the next several sections we outline the problems with computing an ideal space/frequency representation. Although the basic concept of the space/frequency representation is easy to understand, there are several subtle issues involved in its actual realization. These issues limit what we can accomplish with the representation and also guide us in making choices for how to compute it. We need to understand these limitations and choices before we can apply the representation to computer vision problems. Ultimately we will apply the space/frequency representation to 2D signals, but we will use only 1D signals for the explanations in this section.

2.2.1 The Ambiguity Function

Another way to think of building up a space/frequency representation is that of looking for basis functions centered at different locations in space and frequency. These functions are usually based on a symmetric window function $w(x)$ centered at the origin. The function can be moved to (x_0, u_0) in space/frequency by translating in space and modulating in frequency, giving $w(x-x_0)\exp(j2\pi u_0 x)$, where $j = \sqrt{-1}$. The usual method of detecting a given basis function in an input signal $f(x)$ is by complex cross correlation. This is the same as using a matched filter. Using this linear method, the space/frequency representation is

$$SF(x_0, u_0) = \int_{-\infty}^{\infty} f(x)\, w^*(x-x_0)\, e^{-j2\pi u_0 x} dx \tag{23}$$

Except for the conjugation of $w(x-x_0)$ (which is usually inconsequential, since $w(x)$ is usually real), this is the same as the STFT in Equation (22).

One way of assessing the resolution of this method is to determine the response of the basis function to shifted copies of itself. If $f(x)$ is exactly the same as the basis function, *i.e.* $f(x) = w(x-x_0)\exp(j2\pi u_0 x)$, the response will be

$$\begin{aligned} E &= \int_{-\infty}^{\infty} w(x-x_0)\, e^{j2\pi u_0 x}\left[w(x-x_0)\, e^{j2\pi u_0 x}\right]^* dx \\ &= \int_{-\infty}^{\infty} w(x)\, w^*(x)\, dx \end{aligned} \tag{24}$$

which is just the energy of the window function $w(x)$. What really matters is how nearby basis functions in the space/frequency representation will react to this $f(x)$ centered at (x_0, u_0). Ideally, the basis functions should show no reaction for anything but an exact copy of themselves. However, if the basis function is shifted by $(\Delta x, \Delta u)$, the response will be

$$\begin{aligned} E_{\Delta x, \Delta u} &= \int_{-\infty}^{\infty} w(x-x_0)\, e^{j2\pi u_0 x}\left[w(x-x_0-\Delta x)\, e^{j2\pi(u_0+\Delta u)x}\right]^* dx \\ &= e^{-j2\pi\Delta u\left(\frac{\Delta x}{2}+x_0\right)} \int_{-\infty}^{\infty} w\left(\alpha+\frac{\Delta x}{2}\right) w^*\left(\alpha-\frac{\Delta x}{2}\right) e^{-j2\pi\Delta x\alpha} d\alpha \end{aligned} \tag{25}$$

Dropping the phase factor in front of $E_{\Delta x,\,\Delta u}$ gives the autoambiguity function:

$$
\begin{aligned}
A(\Delta x, \Delta u) &= \int_{-\infty}^{\infty} w\left(\alpha + \frac{\Delta x}{2}\right) w^*\left(\alpha - \frac{\Delta x}{2}\right) e^{-j2\pi\Delta x\alpha} d\alpha \\
&= \Im_{\alpha \Rightarrow \Delta u}\left\{ w\left(\alpha + \frac{\Delta x}{2}\right) w^*\left(\alpha - \frac{\Delta x}{2}\right) \right\}
\end{aligned}
\tag{26}
$$

This function is usually used for describing the resolution of range and range rate measurements for Doppler radar signals[107]. In our context, it tells how much response can be expected from the basis functions surrounding the one centered on the input signal. In order to get a high resolution space/frequency representation, we would like the window function's ambiguity function to be as compact as possible in both dimensions. Two examples follow.

Example 1: Let the window function be a unit energy rectangle of width b

$$w(x) = \frac{1}{\sqrt{b}} \operatorname{rect}\left(\frac{x}{b}\right) \tag{27}$$

where

$$\operatorname{rect}\left(\frac{x}{b}\right) = \begin{cases} 1 \text{ for } \left|\frac{x}{b}\right| < \frac{1}{2} \\ \frac{1}{2} \text{ for } \left|\frac{x}{b}\right| = \frac{1}{2} \\ 0 \text{ for } \left|\frac{x}{b}\right| > \frac{1}{2} \end{cases} \tag{28}$$

This is like using sharply truncated sinusoids for the STFT. The ambiguity function is

$$\begin{aligned} A(\Delta x, \Delta u) &= \Im_{\alpha \Rightarrow \Delta u}\left\{ \frac{1}{\sqrt{b}} \mathrm{rect}\left(\frac{\alpha + \frac{\Delta x}{2}}{b} \right) \frac{1}{\sqrt{b}} \mathrm{rect}\left(\frac{\alpha - \frac{\Delta x}{2}}{b} \right) \right\} \\ &= \begin{cases} \Im_{\alpha \Rightarrow \Delta u}\{ \frac{1}{b} \mathrm{rect}\,(\frac{\alpha}{b - |\Delta x|}) \} & \text{for } |\Delta x| < b \\ \Im\{0\} & \text{for } |\Delta x| > b \end{cases} \\ &= \begin{cases} \frac{1}{b}(b - |\Delta x|)\, \mathrm{sinc}\,[\, (b - |\Delta x|)\, \Delta u] & \text{for } |\Delta x| < b \\ 0 & \text{for } |\Delta x| > b \end{cases} \end{aligned} \tag{29}$$

where

$$\mathrm{sinc}(x) = \begin{cases} 1 & \text{for} \quad x = 0 \\ \dfrac{\sin(\pi x)}{\pi x} & \text{otherwise} \end{cases} \tag{30}$$

This function is plotted for three different values of b in Figure 23. As it gains more resolution in space (smaller b) is wider in frequency, it loses resolution in frequency.

Example 2: If the window function is a unit energy Gaussian with width proportional to b

$$w(x) = \left(\frac{\sqrt{2}}{b}\right)^{1/2} e^{-\pi \left(\frac{x}{b}\right)^2} \tag{31}$$

then

$$A(\Delta x, \Delta u) = \Im_{\alpha \Rightarrow \Delta u}\left\{ \left(\frac{\sqrt{2}}{b}\right)^{1/2} e^{-\pi \left(\frac{\alpha + \frac{\Delta x}{2}}{b} \right)^2} \left(\frac{\sqrt{2}}{b}\right)^{1/2} e^{-\pi \left(\frac{\alpha - \frac{\Delta x}{2}}{b} \right)^2} \right\}$$

$$= e^{-\frac{\pi}{2}\left[\left(\frac{\Delta x}{b}\right)^2 + (b \Delta u)^2 \right]} \tag{32}$$

> This corresponds to a STFT with a Gaussian window, and it is close to what we actually use for image analysis. Three versions of this function for different values of b are plotted in Figure 23. There is still a trade-off between resolution in space and frequency, but resolution can be adequate in both dimensions simultaneously.

Both of these examples show that a function centered at some point in space/frequency will cause a peak not only at that point, but also the points around it. This limits the resolution of the space/frequency representation. For unit energy signals, the volume of the squared magnitude and the maximum of the ambiguity function are both one, *i.e.*

$$|A(0,0)| = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} |A(\Delta x, \Delta u)|^2 d\Delta x \; d\Delta u = 1 \tag{33}$$

Furthermore, the maximum is always at the origin. This means there is a "conservation of ambiguity," because the volume cannot be squeezed to an arbitrarily compact shape near the origin, and any reduction along one axis will increase the ambiguity somewhere else.

Four our application, this means that we cannot expect to precisely isolate "events" in space/frequency. We must tolerate some ambiguity. It also guides in our choice of a window function. Although the rectangle gives good spatial resolution, it has a lot of spread in frequency. It is general better to use windows like the Gaussian that fall smoothly to zero.

2.2.2 The Uncertainty Principle

A concept related to the ambiguity function is the uncertainty principle. The basis functions used to measure the frequency content of a signal at a given point in space/frequency will have some inherent spread, and thus will not be able to measure the signal's energy at a pinpoint in space/frequency. There is a direct trade-off between a basis function's spread in space and its spread in frequency that is given by the Fourier transform identity

$$\Im\{w(\frac{x}{b})\} = |b| W(bu) \tag{34}$$

Decreasing the width in space (by decreasing b) increases the width in frequency.

Gabor[29] is usually credited with showing that a function cannot be arbitrarily compact in both dimensions. If the width of a function $f(x)$ and its Fourier transform $F(u)$ are defined as their second moments around their centroids $\bar{f}$ and $\bar{F}$, then the widths are

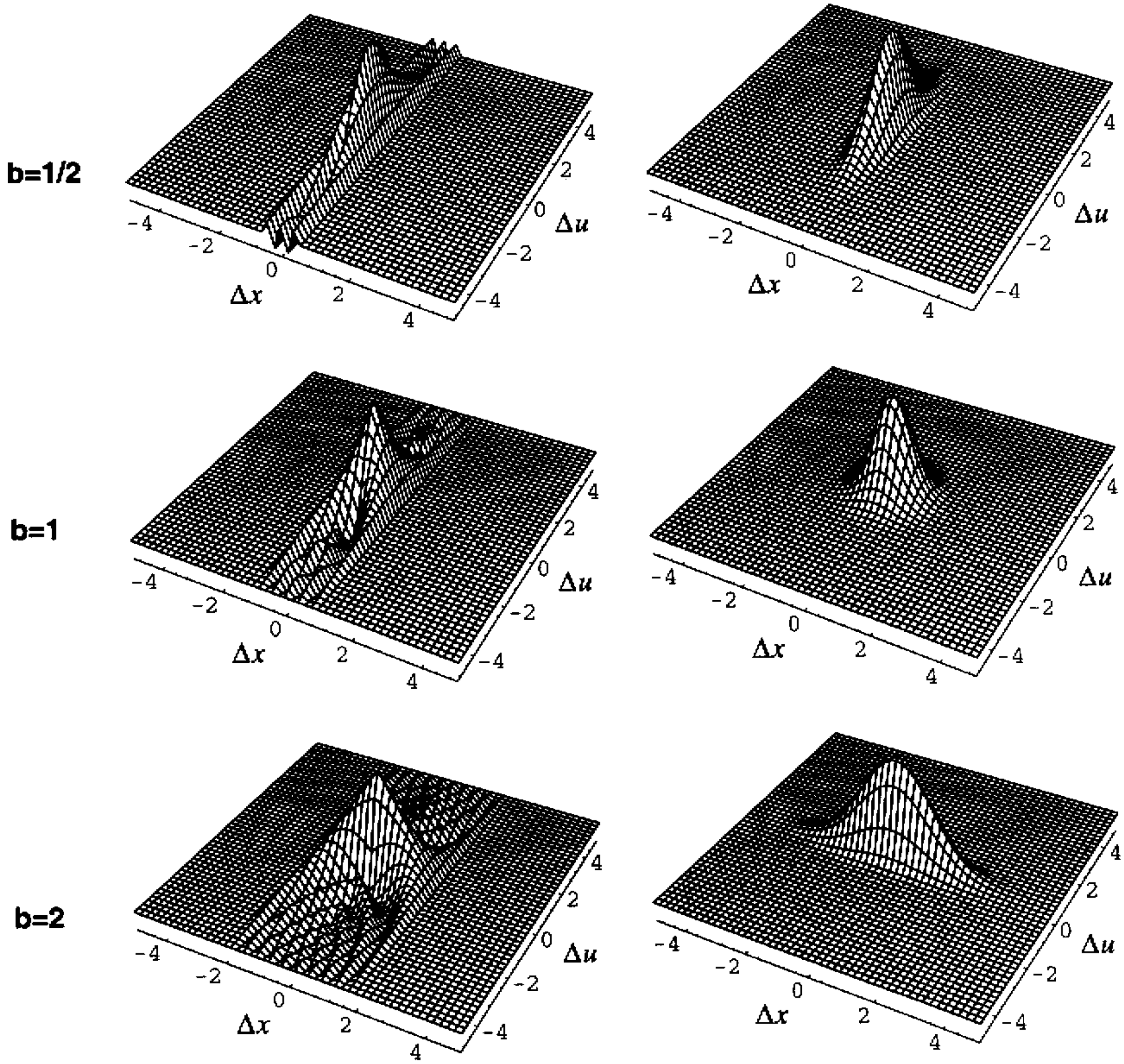


Figure 23: Ambiguity functions for a rectangle in the left column and a Gaussian in the right column. As the width parameter b increases, the ambiguity in the spatial direction grows, while the ambiguity in frequency shrinks. The Gaussian's ambiguity function is generally more compact than the rectangle's.

$$\sigma_x^2 = \frac{\int_{-\infty}^{\infty} x^2 [f(x) - \bar{f}] [f(x) - \bar{f}]^* dx}{\int_{-\infty}^{\infty} [f(x) - \bar{f}] [f(x) - \bar{f}]^* dx}$$

$$\sigma_u^2 = \frac{\int_{-\infty}^{\infty} u^2 [F(u) - \bar{F}] [F(u) - \bar{F}]^* du}{\int_{-\infty}^{\infty} [F(u) - \bar{F}] [F(u) - \bar{F}]^* du} \tag{35}$$

For any function with a Fourier transform, it can be shown that

$$\sigma_x \sigma_u \geq \frac{1}{4\pi} \tag{36}$$

This is called the "uncertainty principle" in signal processing. Gabor showed that the functions with the minimum product of widths are Gaussian-modulated, complex sinusoids. This accounts for their frequent use in computing the space/frequency representation. Even with these "Gabor functions" as basis functions, the uncertainty principle shows that they do have some degree of spread in space/frequency, and that any attempt to decrease the spread in one dimension will increase it in the other.

2.2.3 Side Lobes

The ambiguity function and uncertainty principle show primarily the effect of the window's width. Its shape is also important. The effect of shape can be seen by imagining $f(x)$ as a complex sinusoid of frequency u_0. The space/frequency representation at $x = 0$ will then be

$$\Im_{\alpha \Rightarrow u} \{w(\alpha) e^{j2\pi u_0 \alpha}\} = W(u) * \delta(u - u_0) = W(u - u_0) \tag{37}$$

which is just a shifted version of the Fourier transform of the window function. Thus, the Fourier transform of the window is an important factor in how the space/frequency representation will look. The simplest window function is a unit-width rectangle, rect (x), whose Fourier transform is $\Im\ \{\text{rect}\ (x)\} = \text{sinc}\ (u)$. As shown in Figure 24b, this will produce significant side lobes in the space/frequency representation, which could be mistaken for frequency peaks.

There are many, better window functions to choose, and the large choice is confusing. *Numerical Recipes*[86] (p. 425) says:

> There is perhaps a lot of unnecessary lore about the choice of a window function, and practically every function which rises from zero to a peak and then falls again has been named after someone. ... However, at the level of this book, there is effectively no difference between any of these (or similar) window functions.

We use a window that has been named after two people, the "Blackman-Harris minimum 4-sample" window[23]. It was recommended in Harris' comprehensive comparison of window shapes for Fourier analysis[36]. Its equation is

$$w(l) = \mathrm{rect}\left(\frac{x}{L}\right)\left[w_0 + w_1\cos\left(\frac{2\pi}{L}l\right) + w_2\cos\left(\frac{4\pi}{L}l\right) + w_3\cos\left(\frac{6\pi}{L}l\right)\right] \qquad (38)$$

where L is the width of the window. The coefficients are $(w_0, w_1, w_2, w_3) = (0.35875, 0.48829, 0.14128, 0.01168)$. This is plotted for $L = 1$ in Figure 24c. The Fourier transform of the Blackman-Harris minimum 4-sample window is

$$\begin{aligned} W(u) &= w_0\mathrm{sinc}\,(Lu) \\ &+ \frac{w_1L}{2}\left\{\mathrm{sinc}\left[L\left(u-\frac{1}{L}\right)\right] + \mathrm{sinc}\left[L\left(u-\frac{1}{L}\right)\right]\right\} \\ &+ \frac{w_2L}{2}\left\{\mathrm{sinc}\left[L\left(u-\frac{2}{L}\right)\right] + \mathrm{sinc}\left[L\left(u-\frac{2}{L}\right)\right]\right\} \\ &+ \frac{w_3L}{2}\left\{\mathrm{sinc}\left[L\left(u-\frac{3}{L}\right)\right] + \mathrm{sinc}\left[L\left(u-\frac{3}{L}\right)\right]\right\} \end{aligned} \qquad (39)$$

This is shown in Figure 24d for $L = 1$. Compare this to the Fourier transform of a unit-width rectangle in Figure 24b. Using the smoothly falling window gives practically no positive or negative sidelobes that could be mistaken for peaks or interfere with neighboring peaks. Although there are no convenient formulae to show it, the price of avoiding sidelobes is a decrease in frequency resolution. This is apparent from the figure, which shows that the Fourier transform of the rectangular window is much sharper than that of the Blackman-Harris window. In general, and for our application, it is better to sacrifice a little frequency resolution for the reduction in side lobes.

2.2.4 Frequency Discontinuities

When computing the space/frequency representation of an image with two different, adjacent signals, there will be some basis functions that fall onto both signals. In computer vision, this is the situ-

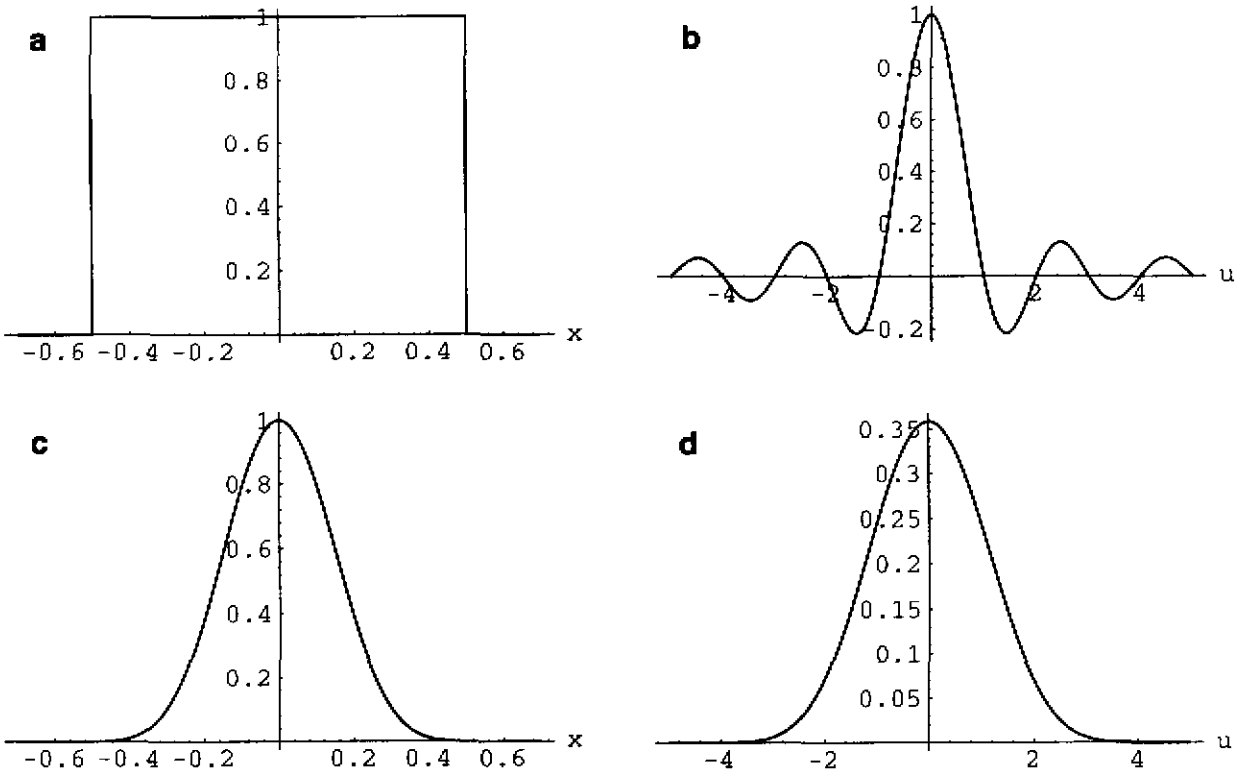


Figure 24: Smooth windows are better than sharp windows.
a) Rectangular window
b) Fourier transform of "a", significant side lobes
c) Blackman-Harris minimum 4-sample window
d) Fourier transform of "c", no side lobes

ation at a texture boundary. This discontinuity causes a broad frequency disturbance. We will compute the effect in the simplest configuration possible by assuming the signal consists of two, complex exponentials[1] of frequencies u_1 and u_2, one on each side of the origin. The function is

[1] Of course a real image will have only real sinusoids. The Fourier transform of a complex exponential is a single delta function at the exponential's frequency. The Fourier transform of a real sinusoid is two delta functions at the sinusoid's frequency, one on either side of the frequency origin. If there is no significant overlap between the functions centered on either side of the frequency origin (which there usually is not in this case) then working with only one side is sufficient for illustrations.

$$f(x) = \begin{cases} a_1 e^{j2\pi u_1 x} & \text{for } x < 0 \\ \dfrac{a_1 + a_2}{2} & \text{for } x = 0 \\ a_2 e^{j2\pi u_2 x} & \text{for } x > 0 \end{cases} \tag{40}$$

Using a unit-length rectangle as the window function, the short-time Fourier transform is

$$\mathrm{STFT}(x, u) = \Im_{\alpha \Rightarrow u} \{\mathrm{rect}\,(\alpha - x) f(\alpha)\}$$

$$= \begin{cases} \dfrac{a_1}{2} e^{-j2\pi x (u - u_1)} \mathrm{sinc}(u - u_1) & \text{for } x < -\dfrac{1}{2} \\ \dfrac{a_1}{2} e^{-j\pi (x - 1/2)\,(u - u_1)} (\dfrac{1}{2} - x)\, \mathrm{sinc}\left[(\dfrac{1}{2} - x)\ (u - u_1) \right] & \\ + \dfrac{a_2}{2} e^{-j\pi (x + 1/2)\,(u - u_2)} (\dfrac{1}{2} + x)\, \mathrm{sinc}\left[(\dfrac{1}{2} + x)\ (u - u_2) \right] & \text{for } -\dfrac{1}{2} < x < \dfrac{1}{2} \\ \dfrac{a_2}{2} e^{-j2\pi x (u - u_2)} \mathrm{sinc}(u - u_2) & \text{for } x > \dfrac{1}{2} \end{cases} \tag{41}$$

The magnitude of this function for $a_1 = a_2$ is plotted as a surface in Figure 25, and the lower 20% of the magnitude is plotted as shaded contours in the same figure. There is significant broadening of the ridges in frequency as they descend to zero. This is because the window is seeing a shorter and shorter section of the sinusoid, causing its Fourier transform to broaden. The frequency ridges coexist in x over a length equal to the length of the window, even though the two original frequencies do not overlap. This tends to make a narrower window more attractive.

2.2.5 Frequency Shifts

Texture boundaries cause one type of frequency nonstationarity in images. Another is caused by the 3D projection effects of a texture on a nonfrontal surface. For instance, as a textured plate recedes from the camera, its frequency appears higher because the projection is more compressed. This tends to violate the local stationarity assumption behind the STFT. If the frequency within the STFT window varies, it causes a smeared Fourier transform at that point. A simple nonstationary signal is a linear, frequency-modulated chirp:

$$f(x) = \exp\left[j2\pi \left(u_0 x + \frac{u_1 x^2}{2} \right) \right] \tag{42}$$

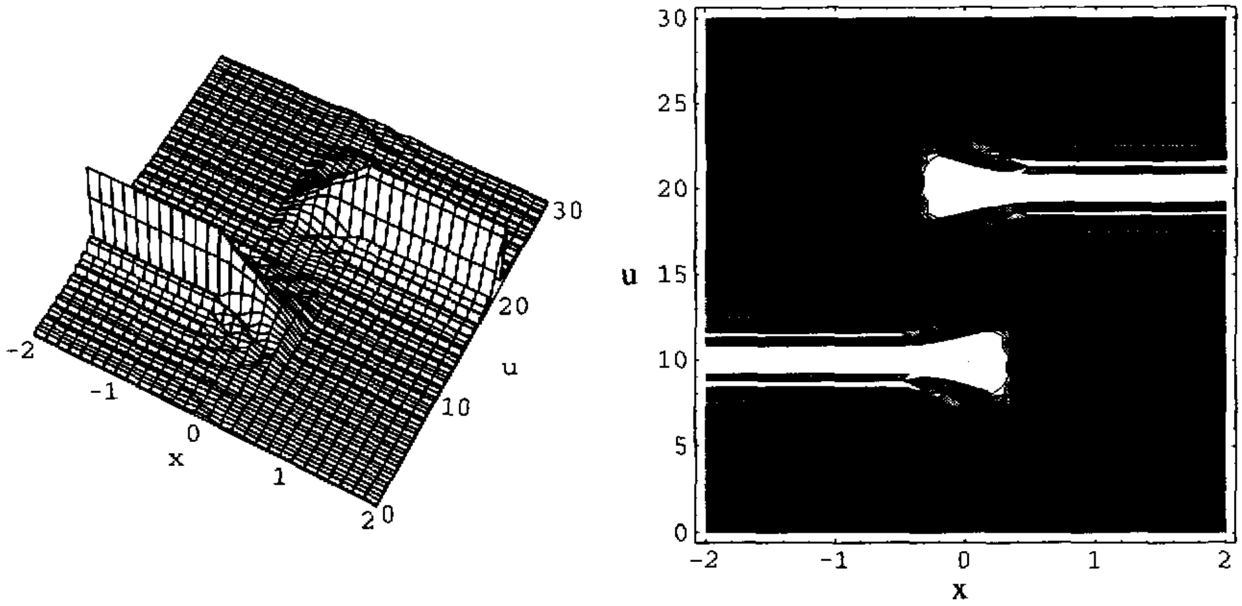


Figure 25: The space/frequency representation of two sinusoids of different frequencies shows broadening of the frequency ridges as they descend to zero and an unrealistic overlap equal to the width of the window.

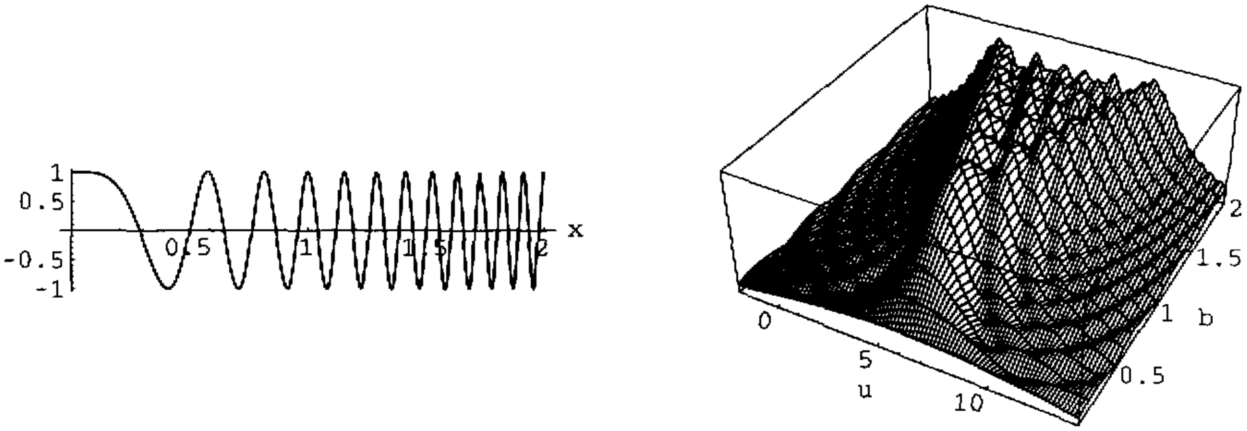


Figure 26: Complex exponential linear chirp (real part) and the magnitude of its Fourier transform for different window widths *b*.

Its "instantaneous frequency" (defined in Section 2.3.1) is $u_0 + u_1 x$. An example of the real part of such a linear chirp is shown in Figure 26, where $u_0 = 0$ and $u_1 = 6$.

Using a rectangular window of width b, the STFT is[2]

$$\mathrm{STFT}\,(x,u) \;=\; \frac{1}{2}\sqrt{\frac{2}{u_1}}\exp\left[j2\pi\left(ux-\frac{(u-u_0)^2}{2u_1}\right)\right][\mathrm{C}\,(X_1)+j\mathrm{S}\,(X_1)+\mathrm{C}\,(X_2)+j\mathrm{S}\,(X_2)] \tag{43}$$

where

$$\begin{aligned} X_1 &= \sqrt{\frac{2}{u_1}}\left[u_1\,(\frac{b}{2}-x)+(u-u_0)\right] \\ X_2 &= \sqrt{\frac{2}{u_1}}\left[u_1\,(\frac{b}{2}+x)-(u-u_0)\right] \end{aligned} \tag{44}$$

and the Fresnel integrals are

$$\begin{aligned} \mathrm{C}\,(X) &= \int_0^X\cos\left(\frac{\pi y^2}{2}\right)dy \\ \mathrm{S}\,(X) &= \int_0^X\sin\left(\frac{\pi y^2}{2}\right)dy \end{aligned} \tag{45}$$

The most apparent feature of this STFT is that it is complicated. Even the simplest frequency shift (linear) is difficult to express in terms of a Fourier transform. Thus, for analyzing texture in images,

[2] This comes from Cook and Bernfeld[21] (p. 137-138) who show that

$$\int_{-T/2}^{T/2}\exp\,(\omega_0 x+\frac{\mu x^2}{2})\,e^{-j\omega x}dx \;=\; \sqrt{\frac{\pi}{2}}\exp\left[-j\left(\frac{(\omega-\omega_0)^2}{2\mu}\right)\right][\mathrm{C}\,(X_1)+j\mathrm{S}\,(X_1)+\mathrm{C}\,(X_2)+j\mathrm{S}\,(X_2)]$$

where

$$\begin{aligned} X_1 &= \frac{1}{\sqrt{\pi\mu}}\left[\frac{\mu T}{2}+(\omega-\omega_0)\right] \\ X_2 &= \frac{1}{\sqrt{\pi\mu}}\left[\frac{\mu T}{2}-(\omega-\omega_0)\right] \end{aligned}$$

We made the following substitutions:

$$\begin{aligned} T &= b \\ \omega &= 2\pi u \\ \omega_0 &= 2\pi\,(u_0+u_1x) \\ \mu &= 2\pi u_1 \end{aligned}$$

we must be content with making an assumption of local stationarity rather than trying to analytically account for higher order frequency shifts.

The other important aspect of this STFT is its spread in frequency. Since the chirp passes smoothly through a range of frequencies over any given window, the corresponding Fourier transform will contain energy at all these frequencies. This is shown on the right side of Figure 26, which is the magnitude of the STFT of the complex exponential whose real part is plotted at the left. We show STFT $(1.0, u)$ for the window width b going from 0.1 to 2.0. For the small width, the chirp is close to a stationary, complex sinusoid, so the STFT is approximately a shifted sinc (u), albeit a wide one since the support in space is small. As the window width grows, the frequency resolution grows, but the window covers more and more frequencies of the chirp, causing the Fourier transform to spread. In summary, for a small window width, the Fourier transform is smeared due to the uncertainty principle, while for large widths, the Fourier transform is smeared because the spatial function contains a range of frequencies. Varying the window width varies the "quality" of the frequency representation for nonstationary signals, and this can be adaptively optimized based on the underlying signal[47].

2.2.6 The Importance of Phase

The spectrogram (local power spectrum) is the squared magnitude of the STFT, and thus contains no phase information. This can be detrimental in some cases if the goal is to distinguish textures. The power spectrum cannot be used to distinguish all textures that appear different to the eye. Julesz[49] argues this in an attempt to refute the notion that the human preattentive visual system performs Fourier analysis. For example, Figure 27 shows sections of five periodic textures that are distinguishable by the human eye, and yet have nearly identical power spectra[3].

The reason that the power spectrum cannot distinguish these textures is that it is insensitive to phase, and phase is the only significant difference between these textures. We can represent an arbitrary periodic function as a Fourier series

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi n u_0 x} \tag{46}$$

where u_0 is the fundamental frequency and the c_n are the complex Fourier series coefficients. The power spectrum of this function is

[3] The power spectra are only "nearly" identical because we have shifted and scaled the intensities of each of them to fit the range of gray levels we can display.

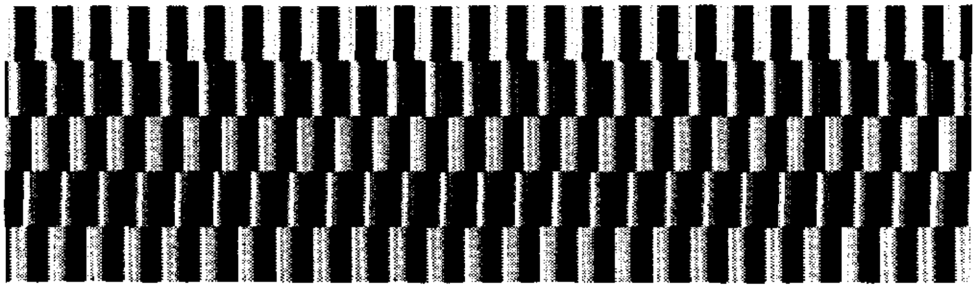


Figure 27: These five texture have nearly identical power spectra.

$$|\Im\{f(x)\}|^2 = \sum_{n=-\infty}^{\infty} |c_n|^2 \delta(u - nu_0) \tag{47}$$

As long as the magnitudes of the Fourier series coefficients stay the same, the power spectrum will stay the same.

We generated the textures in Figure 27 by starting with a truncated Fourier series of a square wave. The full Fourier series has coefficients $c_n = 0.5\mathrm{sinc}\,(n/2)$. We zeroed all the c_n for $|n| > 5$ and plotted the series in the top rows of Figure 27. We generated the other four textures by applying different, random rotations to the c_n. That is, we picked random θ_n's, computed $c'_n = \exp\,(j\theta_n)\,c_n$, and used the c'_n's in the Fourier series. (We had to enforce the hermitian symmetry of the coefficients to ensure a real function.) The actual Fourier series coefficients are shown in Table 2. These rotations of the complex coefficients only affect their phases and not their magnitudes. It is apparent from Equation (47) that the power spectrum is only sensitive to the magnitude of the Fourier series coefficients. We can change the phase arbitrarily without affecting the power spectrum.

In some applications, phase may be important for distinguishing texture. But, for the textures we have tested, there have always been significant differences in the power spectra. We discuss other reasons for ignoring phase in Section 2.3.8 ("Why We Picked the Spectrogram" on page 66).

2.2.7 Good Enough

The STFT is only one way to compute the space/frequency representation, and we look at others in the next section. Even though we have used only the STFT in this section, it does illustrate a compromise common to several different methods of computing the space/frequency representation: this size of the window. Arguments for a small window are increased spatial resolution and decreased frequency smearing due to frequency discontinuities and frequency shifts. The main argument for a large

	n	-5	-3	-1	0	1	3	5
1	real	0.127	-0.212	0.637	1.000	0.637	-0.212	0.127
	imaginary	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	magnitude	0.127	0.212	0.637	1.000	0.637	0.212	0.127
2	real	-0.114	-0.205	0.134	1.000	0.134	-0.205	-0.114
	imaginary	-0.056	-0.056	0.622	0.000	-0.622	0.056	0.056
	magnitude	0.127	0.212	0.637	1.000	0.637	0.212	0.127
3	real	-0.120	0.102	0.426	1.000	0.426	0.102	-0.120
	imaginary	-0.042	0.186	-0.473	0.000	0.473	-0.186	0.042
	magnitude	0.127	0.212	0.637	1.000	0.637	0.212	0.127
4	real	-0.115	-0.193	-0.569	1.000	-0.569	-0.193	-0.115
	imaginary	-0.055	-0.087	-0.285	0.000	0.285	0.087	0.055
	magnitude	0.127	0.212	0.637	1.000	0.637	0.212	0.127
5	real	0.041	-0.212	-0.251	1.000	-0.251	-0.212	0.041
	imaginary	-0.121	0.012	0.585	0.000	-0.585	-0.012	0.121
	magnitude	0.127	0.212	0.637	1.000	0.637	0.212	0.127

Table 2: Fourier series coefficients of the five textures in Figure 27. The magnitudes are identical from texture to texture. (The coefficients for n = -4, -2, 2, 4 and |n|>5 are zero.)

window is increased frequency resolution. Any choice is a compromise. Some of the techniques in the next section have ways of varying the window size for certain effects, but this still involves some size choices. Our experience was that we could find a size that worked well for everything we wanted to do.

Based on this section, we can conclude the following about computing a space/frequency representation of a signal:

- We cannot single out isolated "events" in space/frequency with arbitrary precision. The ambiguity function says that basis functions respond to nearby copies of themselves, and that this response neighborhood has a certain minimum size.
- There is a trade-off in resolution in space and resolution in frequency according to the uncertainty principle. Even if we choose the best window shape (the Gaussian), we must still pick a size that involves a compromise.
- The shape of the window is important. Windows with a sharp cutoff generally give sharper peaks in frequency, but they also cause side lobes. It is usually better to use windows that fall smoothly to zero. Although this dulls the frequency resolution, it also reduces the sidelobes dramatically.

- Both frequency discontinuities and frequency shifts cause smearing in the space/frequency representation.
- For some applications, it may be important to use phase to discriminate signals.

Thus the space/frequency representation is not perfect. Being aware of these issues allows one to make informed choices when using it. In spite of the problems, it is still a powerful representation for analyzing image texture.

2.3 Methods of Computing the Space/Frequency Representation

There is no single best way to compute the space/frequency representation. There is not even a definition of what the ideal space/frequency representation of an arbitrary signal should be. Different methods of computing the representation optimize different criteria. These criteria include simplicity, redundancy, resolution, uncertainty, and coverage in space/frequency. This section contains a brief review of the basic choices and some mention of how they have been used in computer vision, and it concludes with our reasons for choosing the spectrogram.

2.3.1 Instantaneous Frequency

The instantaneous frequency of a sinusoid like $A\cos(2\pi g(x))$ is defined to be[79]

$$u(x) = \frac{d}{dx}g(x) \tag{48}$$

A stationary sinusoid $A\cos(2\pi ux)$ has an instantaneous frequency of u. A linear chirp $A\cos(2\pi ax^2)$ has a frequency that changes linearly with x. Its instantaneous frequency is $2ax$, which is reasonable. For more general signals $f(x)$, the instantaneous frequency is defined as[80]

$$u(x) = \frac{1}{2\pi}\frac{d}{dx}\arg(f(x)) \tag{49}$$

where $\arg(|z|e^{j\theta}) = \theta$. For real signals, the (complex) analytic version of the signal must be used for this definition.

The instantaneous frequency is a single-valued function of x. Both these definitions apply only to signals that have only a single frequency component, so they do not apply to the more general signals that we see in images.

2.3.2 Short-Time Fourier Transform and Spectrogram

The short-time Fourier transform (STFT) of $f(x)$ is a series of Fourier transforms taken over finite, equally long sections of the x axis. For 1D, continuous signals, it is

$$STFT(x, u) = \Im_{\alpha \Rightarrow u}\{w(\alpha - x)f(\alpha)\} = \int_{-\infty}^{\infty} w(\alpha - x)f(\alpha)e^{-j2\pi\alpha u}d\alpha \tag{50}$$

This can be thought of as decomposing a function into basis functions $w(\alpha - x)\exp(-j2\pi\alpha u)$, which are just complex sinusoids modulated by shifted versions of the window function $w(\alpha)$. The window function's shape and width must be chosen based on the task at hand.

The coverage of the STFT in the space/frequency domain is illustrated in Figure 28. Each point covers some region whose extend depends on the window function. Since the window is the same for every point, the space/frequency domain is divided uniformly. The size of the window alters the shape of the region according to the uncertainty principle.

The spectrogram is the squared magnitude of the STFT. It does away with phase and it is never negative. Phase is usually not important for texture analysis, as we are only concerned with the texture's constituent frequencies and how they change with position. It is the phase that encodes the spatial information of a signal, and we are already representing the spatial coordinates explicitly in the space/frequency representation. The non-negativity of the spectrogram is important from an intuitive point of view. It makes sense that all energies should be positive. One common criticism of other space/frequency representations like the Wigner distribution is that they are sometimes negative.

The generalization of the 1D STFT and spectrogram to 2D images is straightforward. For image analysis, the STFT and spectrogram are based on 2D Fourier transforms, and this leads to a four-dimensional representation. For a 2D function $f(x, y)$, the image STFT is

$$\begin{aligned} STFT(x, y, u, v) &= \Im_{(\alpha, \beta) \Rightarrow (u, v)}\{w(\alpha - x, \beta - y)f(\alpha, \beta)\} \\ &= \int_{-\infty}^{\infty}\int_{-\infty}^{\infty}(w(\alpha - x, \beta - y)f(\alpha, \beta))e^{-j2\pi(\alpha u + \beta v)}d\alpha d\beta \end{aligned} \tag{51}$$

Each 2D point on the image has a local, 2D Fourier transform.

The image spectrogram is the squared magnitude of the image STFT. For discrete signals, the Fourier transforms are implemented with an FFT algorithm. Except for aliasing, which we analyze in Sections 1.3.3 and 3.5.3, all the intuition about the continuous STFT applies in the discrete case. In

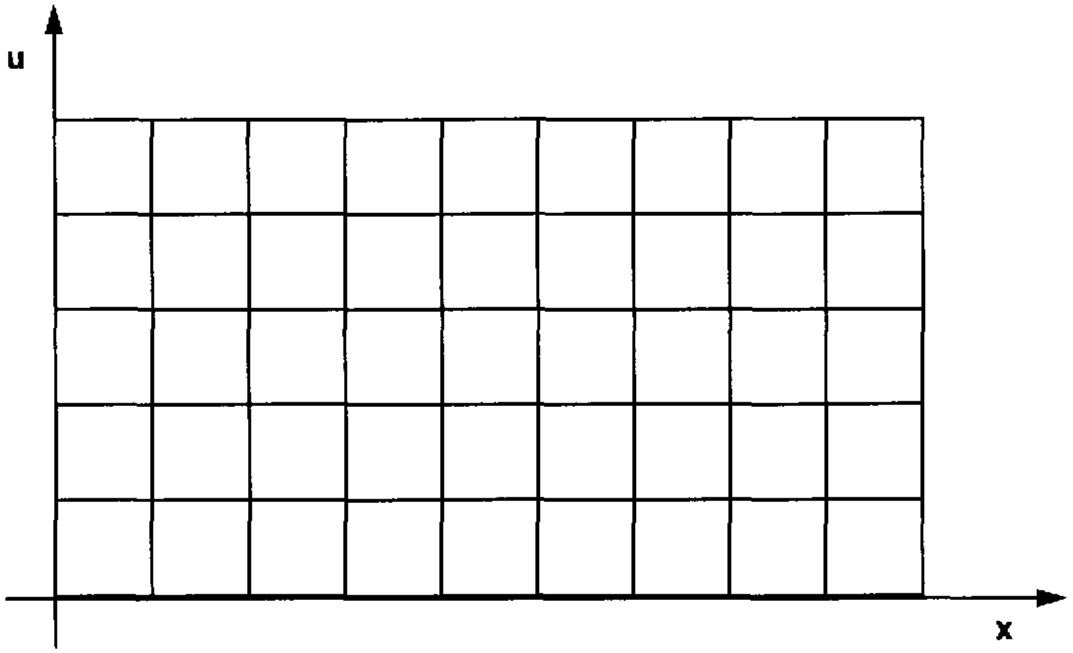


Figure 28: The STFT and spectrogram cover the space/frequency domain with identically shaped and sized basis functions centered at different points in (x,u). (The basis functions actually overlap significantly, but we have drawn them adjacent to each other.) (Redrawn from [88].)

practice, it may not be necessary to compute a Fourier transform around every point. This reduces the spatial resolution of the representation.

Bajcsy[4] was the first to propose using the spectrogram for the description of image texture. She defines several texture features based on polar coordinates in the local power spectrum and shows how these features vary from one texture to another. She and Lieberman[5] went on to show how the texture gradient on a receding ground plane is related to frequency shifts in the image spectrogram. Gramenopoulos[33] measured the energy in four subsections of each local power spectrum for terrain classification. In their comparison of different texture features for classification, Weszka *et al.*[114] used features based on the intersections of rings and wedges of the image's power spectrum. Brown and Shvaytser[11] used the autocorrelation of an image of a slanted, textured surface to find the surface's normal. This is related to frequency methods since the autocorrelation and the power spectrum are Fourier transforms of each other.

2.3.3 General Filters

The STFT can be thought of as a decomposition using a set of densely spaced, equally sized, equally shaped filters in the space/frequency domain. Other versions of the space/frequency representation use a set of filters that are spaced, sized, and shaped to suit the task. This is a broad category of space/frequency representation, but we distinguish it in that the spacing, size, and shape of the filters used here can be arbitrarily chosen by the user, whereas the filters for the STFT and wavelets have some predetermined relationships among them.

One of the earliest and most basic representatives of this category is Triendl's 1972 work on texture segmentation[103]. He decomposed terrain images into a low frequency image (using a 3x3 averaging filter) and a high frequency image (using a 3x3 Laplacian derivative). Each of these was in turn smoothed with an 11x11 averaging filter to produce images that corresponded to scalar measures of "brightness" and "roughness." Each pixel was classified based on these two features. Although he did not interpret his filters in terms of the space/frequency representation, they could be, so this work belongs in this category. Laws' texture energy measures are a more sophisticated version of this idea. He used 1D and 2D filters of dimensions three and five, along with a nonlinear averaging step, to classify textures.

Many researchers use biologically inspired filters for texture analysis. Malik and Perona[66] derive and implement a model of human, preattentive texture edge-finding. They use filters consisting of linear combinations of two and three Gaussians, because the filters fit physiological measurements and are simple to implement. By far the most popular class of biologically plausible filter is the Gabor function. Clark *et al.*[19] segment textures using two or three Gabor filters chosen manually to work on the specific textures in the image. Fogel and Sagi[27] use 32 different Gabor filters (four frequencies, four orientations, two phases) for segmenting textures. They use the smoothed, thresholded output of the filters as the input to a Laplacian edge detector. For shape-from-texture, Super and Bovik[97] used a set of 56 and a set of 80 Gabor filters to estimate the peak frequency at every point in an image. In later work[98] they use a set of 72 filters for estimating the moments of the frequency distribution at every point, which are in turn used for shape-from-texture.

For shape-from-texture, it is better to have a dense sampling of the space/frequency domain, as given by the spectrogram, than a sparse sampling as in the work above. This is because the surface orientation of textured surfaces causes slight shifts in frequency, and we measure these shifts to estimate surface orientation. In Super and Bovik's first paper on the topic[97], they found the approximate location of the peak frequency and then refined this estimate by searching the area with a denser set of filters. In the shape-from-texture algorithms we present in this thesis, we examine the shift of at least several frequencies, and sometime the entire frequency plane. This means we require a dense sampling in frequency.

2.3.4 Wavelets

Wavelets are basis functions of the form

$$h_a(x) = \frac{1}{\sqrt{a}} h\left(\frac{x}{a}\right) \tag{52}$$

These are scaled versions of the "mother wavelet" $h(x)$, which is usually a function that integrates to zero. The scale factor a is inversely proportional to frequency. The continuous wavelet transform (CWT)[88] of $f(x)$ is

$$\mathrm{CWT}(x, a) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(\alpha)\, h\left(\frac{\alpha - x}{a}\right) d\alpha. \tag{53}$$

If the mother wavelet is a complex exponential modulated by a window function $w(x)$, *i.e.* $h(x) = w(x) \exp(-j2\pi u_0 x)$, then the CWT will be

$$\mathrm{CWT}(x, a) = \frac{1}{\sqrt{|a|}} \int_{-\infty}^{\infty} f(\alpha)\, w\left(\frac{\alpha - x}{a}\right) e^{-j2\pi \frac{u_0}{a} \alpha} d\alpha. \tag{54}$$

As the scale factor increases, the center frequency u_0/a decreases and the window width grows proportionally to a. In this form, the continuous wavelet transform is like a short time Fourier transform, only with a window width proportional to wavelength. Thus, the basis functions will cover a constant number of wavelengths, meaning that lower frequencies get wider windows and higher frequencies get narrower windows. The coverage of some of these basis functions is shown in Figure 29. The Gabor filters used by Super and Bovik[97][98], discussed in the previous section, follow this kind of scaling. In accordance with the uncertainty principle, the narrower windows at higher frequencies will have more spread in frequency than the wider windows at lower frequencies. This width scaling makes sense for some applications where frequency effects are expected to persist in proportion to their wavelength.

The idea of using filters whose width is proportional to their wavelength existed long before the term "wavelets" was invented. Besides unifying the mathematics of such filters, researchers in wavelets has discovered certain, discrete, finite-width mother wavelets that are *orthogonal* to scaled and translated versions of themselves[22]. This is one of the new, important discoveries in the field. This orthogonality means that the wavelet representation can be nonredundant, and therefore can be encoded in the same amount of space as the original, discretely sampled function. The discrete wavelet transform can be implemented with a cascade of filter banks[88]. The coverages in space/fre-

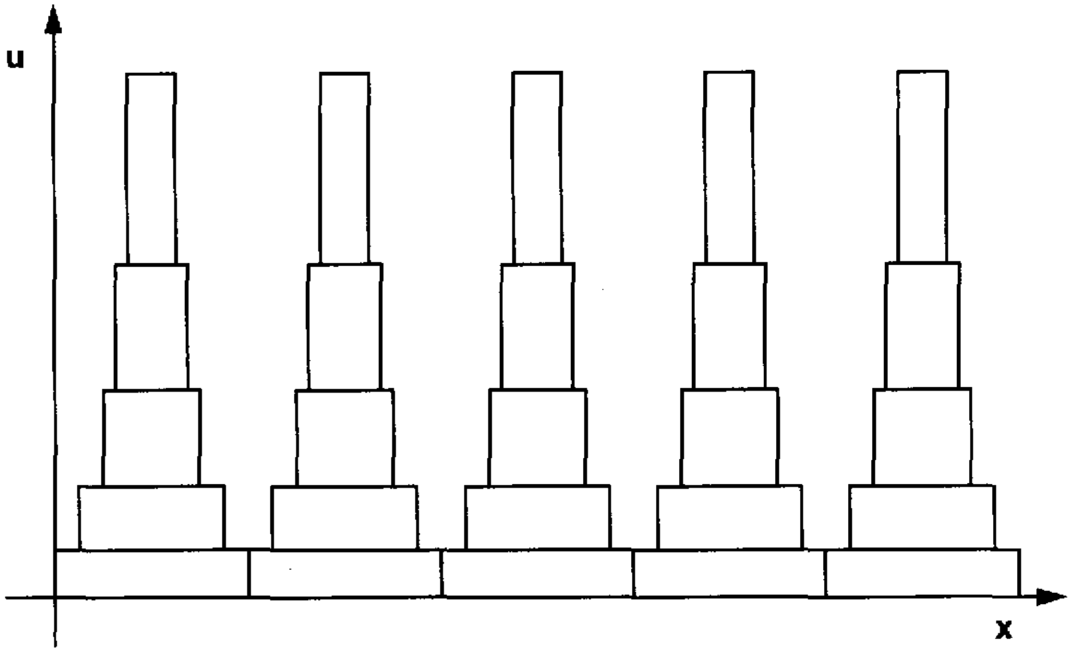


Figure 29: Coverage of some of the basis functions for the continuous wavelet transform. The width in x is proportional to the wavelength. The frequency resolution drops as the windows get narrower.

quency of the basis functions for a discrete wavelet transform are related by powers of two. Each successively higher frequency basis function has twice the bandwidth and half the spatial width. This is shown in Figure 29.

The main difference between wavelets and the spectrogram is that wavelets use a variable-size window, while the spectrogram uses a constant-size window.

2.3.5 Wigner Distribution

The Wigner distribution of $f(x)$ is defined as

$$\begin{aligned} \mathrm{WD}\,(x,u) &= \int_{-\infty}^{\infty} f(x+\frac{\alpha}{2})f^*\,(x-\frac{\alpha}{2})\,e^{-j2\pi u\alpha}d\alpha \\ &= \Im_{\alpha\Rightarrow u}\{f(x+\frac{\alpha}{2})f^*\,(x-\frac{\alpha}{2})\,\} \end{aligned} \tag{55}$$

This is the simplest of several different bilinear space/frequency representations. It is attractive in terms of its resolution, preservation of space and frequency support, and other properties. A good review of the Wigner distribution is a three-part series by Claasen and Mecklenbrauker[16]-[18].

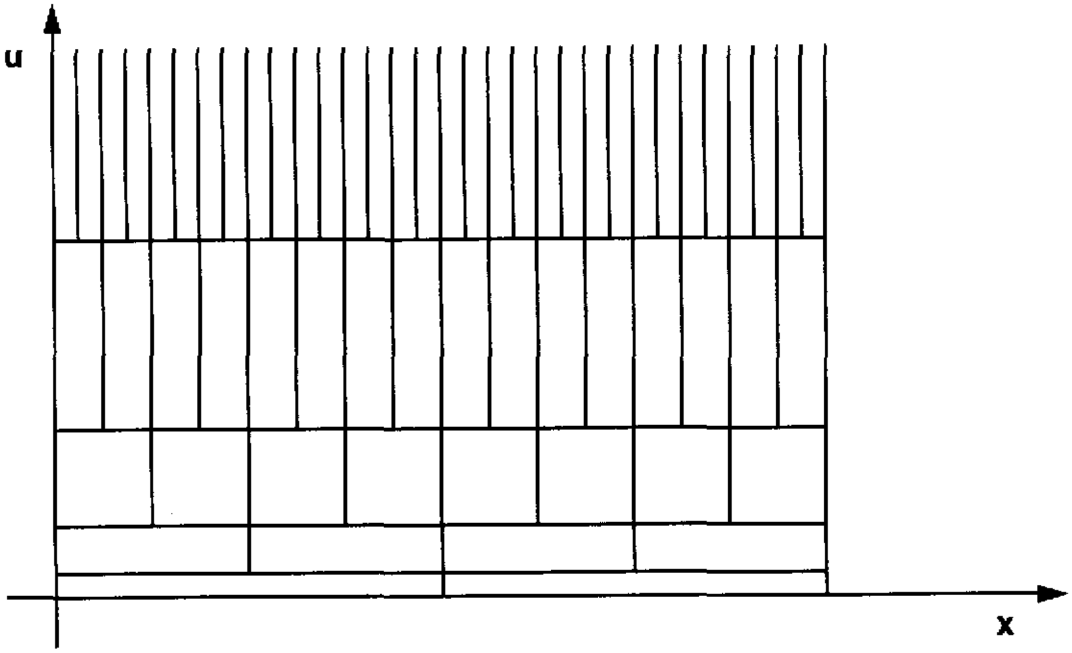


Figure 30: Coverage of all the basis functions for the discrete wavelet transform. The widths in x and u are related by powers of two.(Redrawn from [88].)

Three examples show an important advantage and an important disadvantage of this method. In working with actual data, the WD must be modified with a window function that limits the support of the Fourier transform. This is called the "pseudo-WD". This reduces the resolution of the representation somewhat, but the basic properties of the WD and pseudo-WD are the same.

Example 1: The Wigner distribution of a single, complex exponential $f(x) = \exp(j2\pi u_0 x)$ is

$$\begin{aligned} \mathrm{WD}(x, u) &= \Im_{\alpha \Rightarrow u}\left\{ e^{j2\pi u_0 (x + \frac{\alpha}{2})} e^{-j2\pi u_0 (x - \frac{\alpha}{2})} \right\} \\ &= \Im_{\alpha \Rightarrow u}\{ e^{j2\pi u_0 \alpha} \} \\ &= \delta(u - u_0) \end{aligned} \tag{56}$$

This constant ridge in space/frequency is just what we would like.

Example 2: The Wigner distribution of a linear, complex exponential chirp $f(x) = \exp[j2\pi(u_0 x + u_1 x^2/2)]$ (same as Equation (42)) is

$$
\begin{aligned}
WD(x,u) &= \Im_{\alpha \Rightarrow u}\{\exp[j2\pi\alpha(u_0+u_1x)]\} \\
&= \delta[u-(u_0+u_1x)]
\end{aligned}
\qquad (57)
$$

This ridge exactly tracks the instantaneous frequency, which is much simpler than the STFT of this signal in Equations (43) and (44).

Example 3: If the function to be analyzed is periodic with fundamental frequency u_0, it can be expressed as a Fourier series, *i.e.*

$$
f(x) = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi n u_0 x} \qquad (58)
$$

While we would expect the space/frequency representation to be a series of ridges at integer multiples of the fundamental frequency, we show in Appendix 2 that the Wigner distribution is actually

$$
\begin{aligned}
WD(x,u) &= \sum_{r=0}^{\infty} |a_n|^2 \delta(u - r u_0) \\
&+ \sum_{n=0}^{\infty} \sum_{m=n+1}^{\infty} \mathrm{Re}\,\{a_n a_m^{*} e^{j2\pi u_0 x(n-m)}\}\,\delta\left[u - \frac{(n+m)}{2}u_0\right]
\end{aligned}
\qquad (59)
$$

where Re (z) means the real part of z, and

$$
\begin{aligned}
a_0 &= c_0 \\
a_n &= 2c_n \text{ for } n \neq 0
\end{aligned}
\qquad (60)
$$

The first term of this WD is exactly what we would like, but the second term represents the cross terms that occur in the WD of any sum. The bilinearity of the WD means that the WD is not linear, giving oscillating cross terms in addition to auto-terms. There are several variations on the WD that are reviewed in [40]. Some of these represent attempts to reduce the magnitude of the cross terms, but they are always still there.

Reed and Wechsler[87] use the Wigner distribution for segmenting flat, frontal textures. They keep the frequency with the highest energy at every pixel and use a relaxation technique to label regions. Jau and Chin[45] use the Wigner distribution for shape-from-texture by measuring variations in high-frequency energy due to 3D effects.

2.3.6 Adaptive Filters

All the methods for computing the space/frequency representation discussed until now work independently of the underlying function that is being analyzed. Jones and Parks[47] developed an "adaptive time-frequency representation" (ATFR). It is a short-time Fourier transform with a Gaussian window given by $\exp\left[c_{x_0,\,u_0}(x-x_0)^2\right]$. The complex Gaussian parameter $c_{x_0,\,u_0}$ varies from point to point in space/frequency. The real part controls the width of the window, and the imaginary part controls the "chirpiness." Jones and Park develop a criterion function that measures the peakiness of the space/frequency representation at any point (x_0, u_0) as a function of this parameter. They perform an exhaustive search over a set of candidate values of $c_{x_0,\,u_0}$ and pick the one that maximizes the peakiness at every point in space/frequency. Using several different signals as examples, they show how this produces higher-resolution representations than the spectrogram without the cross terms of the Wigner distribution.

2.3.7 Specially Tuned Filters

Super[99] has recently developed a clever, new shape-from-texture algorithm that uses filters matched to the frequency distortions caused by 3D effects. His filters have a Gaussian envelope, and they are parameterized by pixel location, 2D frequency, and the slant and tilt of the plane viewed in the scene. He finds which filter best fits a section of the image and declares that filter's slant and tilt parameters as the surface orientation of the underlying textured plane.

2.3.8 Why We Picked the Spectrogram

At first glance, the spectrogram seems to be the simplest, most obvious, and easiest-to-implement method for computing the space/frequency representation. While this is true, a deeper analysis also shows that it is the best method for our particular application (segmentation and shape from periodic texture). We arrived at this conclusion essentially by eliminating all the other possibilities.

Using instantaneous frequency is impractical, because it only makes sense for single sinusoids, and textures are hardly ever this simple.

The STFT is like the spectrogram except that it includes phase. Although it has been shown that phase is at least as important as magnitude in representing images[78], and some distinctly different textures have identical spectrograms (Section 2.2.6), including phase would have unnecessarily complicated our algorithm by giving an extra pair of parameters (phase in two dimensions) to search over

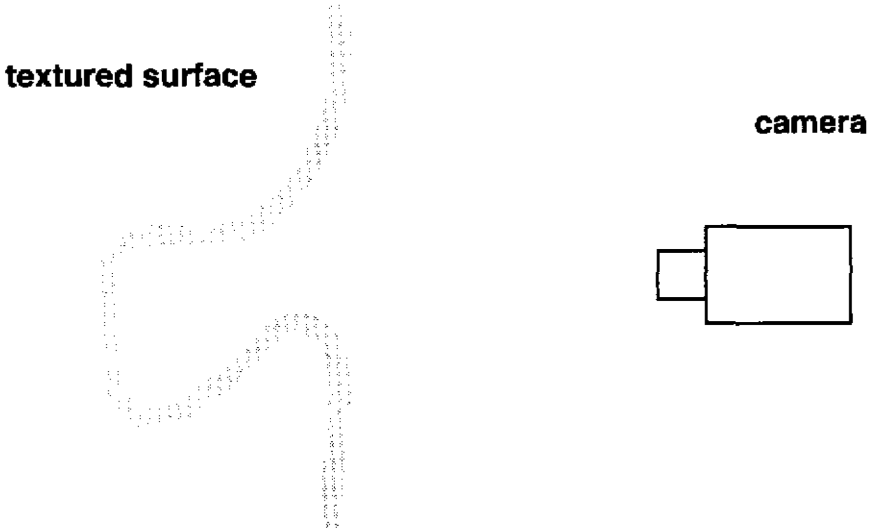


Figure 31: Self occlusion of a textured surface can cause a phase discontinuity in the image.

when matching texture patches. Irregular textures, which we analyze in Section 3.5.4, do not have coherent phase properties, so it would be a disadvantage to depend on phase for shape-from-texture. Also, in some cases as in Figure 31, self-occlusion of a textured surface can cause an apparent violation of phase continuity in the image.

The spectrogram can be considered a special case of using general filters for creating the space/frequency representation. Usually, filters are used to create a relatively sparse sampling of the space/frequency domain, while the spectrogram covers it densely. In shape-from-texture, it is important to track small shifts in frequency due to 3D effects, and this calls for a dense sampling in frequency. In most of Malik's work, he uses a set of filters. However, in his work with Rosenholtz on shape-from-texture, they used the spectrogram, because the sparse sampling that they tried originally did not work[67].

The main difference between wavelets and the spectrogram is that wavelets use a variable width window, with narrower windows for higher frequencies. For our purposes of analyzing texture, a variable window size space/frequency representation is less attractive than the spectrogram. Textures usually have sharp boundaries and slowly varying frequency content (due to 3D effects), so the high frequency components have the same extent as the low frequency components. There is no use, then, in having the high frequency windows be narrower than the low frequency windows. Of course, narrower windows are less prone to being corrupted by texture boundaries because they are less likely to overlap into two different textures. But, if a wide-window low frequency component is corrupted, it is really no consolation that the high frequency components are still pure.

The Wigner distribution and the other, associated bilinear distributions are unsuitable because of their inherent cross terms. Besides making the space/frequency representation messier, the oscillating cross terms would confound attempts at matching frequencies for segmentation or computing shifts in frequencies for shape-from-texture.

The adaptive approach of Jones and Parks[47] to computing the space/frequency representation is, at each point in (x, u), dependent on the signal for its window. In addition to being very slow to compute, it is difficult to predict how the signal will affect the window. This approach has potential for our problem, but the theory is not mature enough to build applications with it.

Super's specially tuned filters for shape-from-texture[99] work well for that problem. More work needs to be done to make them work for random textures, which the spectrogram can handle in theory. It is not clear how to use these filters for texture segmentation either. In addition, the Fast Fourier Transform cannot be used to apply these filters. However, this is a promising approach to texture filtering that deserves more attention.

2.4 Summary

This chapter was about the space/frequency representation - the representation that we use in the next two chapters. It shows a signal's spatial and frequency characteristics simultaneously. Unlike the Fourier transform of the whole signal, the space/frequency representation maintains the spatial coherence of the signal. Actual computation of the space/frequency representation suffers from several effects that limit its resolution. Some of these, like the ambiguity function and the uncertainty principle, are important for general applications. Others, like frequency shifts and frequency discontinuities, are especially important for image texture analysis. There are several ways to compute the space/frequency representation. We chose the spectrogram because it suits our needs better than the other methods.

Chapter 3 Orientation of Textured Surfaces from Frequency Shifts

3.1 Overview of Problem and Solution

Shape-from-texture algorithms compute surface normals of uniformly textured surfaces from images. A simple example of an appropriate image for this analysis appears in Figure 32. If we assume that the plate is uniformly textured, then all the variations in texture can be attributed to the plate's changing depth and surface normal. As the plate recedes, the texture elements appear smaller and closer together. There are mathematical relationships between the texture's change in appearance (*e.g.* spacing, area, perimeter, frequency) and the surface normal of the textured surface. Shape-from-texture algorithms work by detecting this change in appearance and then applying the mathematical relationships to find the surface normal.

Our shape-from-texture algorithm works by comparing the local spatial frequency of different regions. An increase in depth causes the texture to appear more compressed in the image. This is manifested as an increase in spatial frequency. Figure 32 illustrates this effect. The two light-colored patches show the 2D Fourier power spectra (part of the spectrogram) of the underlying pixels. Since

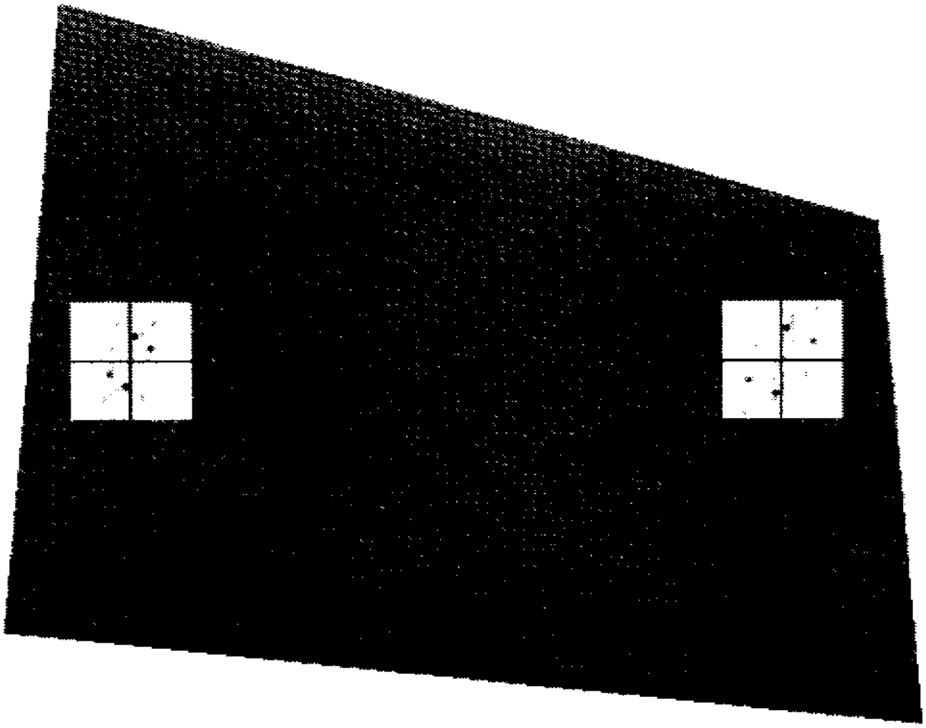


Figure 32: Local Fourier power spectra show frequency shift due to surface normal.

the texture is periodic, the Fourier transform consists of peaks at the texture's fundamental and harmonic frequencies. The peaks in the right patch are farther from the frequency origin, indicating higher frequencies. We show in this chapter that the relationship between the two Fourier transform patches is approximately affine. These affine parameters are functions of known quantities and the unknown surface normal. Our algorithms work by searching for the surface normal that accounts for the affine transformation between two (or more) patches.

Shape-from-texture algorithms can be categorized based on their assumptions about the texture. One split concerns whether or not the texture is composed of discrete, observable texture elements (texels). Examples of texels include spots on a leopard and bricks in a wall. Textures without texels include clouds and dryer lint. As we mentioned in Section 1.2.10 ("High-Level Features" on page 9), texture elements are very difficult to reliably find in images. One of the main advantages of our algorithm is that it does not require the detection of texels.

Another dichotomy concerns periodic versus random textures. Periodic textures are generally easier to analyze by shape-from-texture. Although the first algorithm we develop in this chapter applies to both kinds of textures, it really only works on periodic textures. Finding surface normals from random textures usually requires a large region of the texture and strong assumptions about how it would appear if viewed from the front.

A third way of splitting shape-from-texture algorithms concerns the standard to which the shape-induced deformations are compared. Some algorithms are based on an assumption about the appearance of the frontally-viewed texture. Deviations from this assumed form are attributed to shape effects. A common such assumption is that the frontally-viewed texture is isotropic in some way. For example, Witkin[115] assumed a uniform distribution of edge directions, while Brown and Shvaytser[11] assumed a circularly symmetric autocorrelation function. The other type of algorithm compares the texture to itself in different parts of the image, attributing differences to depth changes caused by the surface normal. Our algorithm is of this latter type. We compare the spatial frequency of the texture in different parts of the image and find the surface normal that accounts for the difference. This algorithm lets us make fewer assumptions about the frontal texture, but it requires us to rely on possibly small depth changes to compute the surface normal.

Of course, all shape-from-texture algorithms presuppose that the textured surfaces in the image have either been segmented or that the image contains only one texture. We make the same assumption in this chapter. In Chapter 4 we eliminate this assumption and do segmentation and shape-from-texture simultaneously.

The remainder of this chapter describes three different shape-from-texture algorithms. Sections 3.2 and 3.3 contain a derivation of some basic equations that apply to all the algorithms. They show that the local Fourier transform of an image patch from a nonfrontal texture is approximately an affine transformation of the Fourier transform of the frontally-viewed texture. Section 3.4 shows that the connection between two local power spectra from a similarly textured plane are approximately related by an affine transformation. Sections 3.5 and 3.6 use this fact to develop two algorithms based on direct comparisons of power spectrum patches. One of them works on any pair of patches (even if one is aliased), and the other is designed to work on patches that are close together in the image. Section 3.7 presents a third algorithm based on the assumption that the texture is periodic.

3.2 Approximate Local Image of a Textured Surface

This section contains a derivation of the connection between a frontally-viewed texture pattern and the approximate image of that pattern when the texture has been mapped to an arbitrary surface in the scene and viewed in perspective. We will use this relationship later to develop a relationship between the Fourier transforms of texture patches taken from the same textured surface.

Figure 33 shows the coordinate frames used in the derivation. The camera's pinhole is at the origin of the (X, Y, Z) frame. This serves as the world coordinate frame, and points defined in it will be referred to with upper-case (X, Y, Z). The $-Z$ axis is coincident with the camera's optical axis and points into the scene being imaged. The image plane is the (x, y) frame with its origin on the optical axis at a distance d behind the pinhole. It is parallel to the XY plane. d is known as the camera's effective focal length.

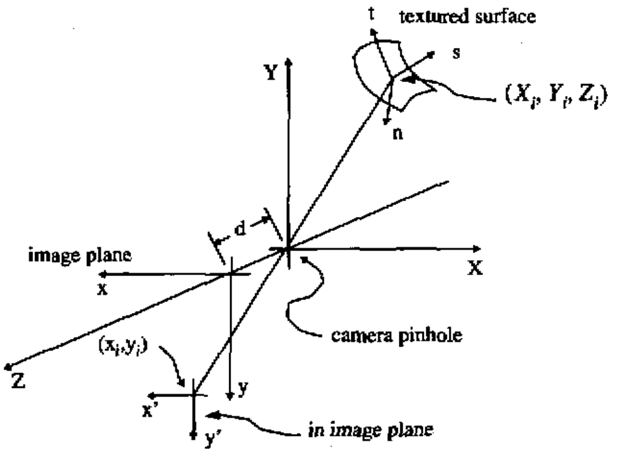


Figure 33: Coordinate frames used in derivation

We imagine that each point on the locally planar textured surface has its own coordinate frame (s, t, n), with the n axis coincident with the surface normal. The surface normal is defined with the gradient space variables (p, q), thus the unit vector along the n axis is $\hat{n} = \frac{1}{r}(p, q, 1)$, with $r = \sqrt{p^2 + q^2 + 1}$, in the world frame. The origin of this surface frame is (X_i, Y_i, Z_i) with respect to the world frame.

The 4x4 homogeneous transformation matrix that locates and orients the surface frame with respect to the world frame is

$$\begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \frac{1}{r}\begin{bmatrix} \frac{p^2 + rq^2}{p^2 + q^2} & \frac{pq(1-r)}{p^2 + q^2} & p & rX_i \\ \frac{pq(1-r)}{p^2 + q^2} & \frac{rp^2 + q^2}{p^2 + q^2} & q & rY_i \\ -p & -q & 1 & rZ_i \\ 0 & 0 & 0 & r \end{bmatrix} \tag{61}$$

This was derived by making a single rotation of the (s, t, n) frame around the unit vector $(-q, p, 0) / (\sqrt{p^2+q^2})$ by an angle ϕ with $\cos\phi = 1/r$ and $\sin\phi = (\sqrt{p^2+q^2})/r$.

We assume the texture on the surface is "painted" on and not a relief pattern. (The resulting algorithms can work even when this assumption is violated, as shown in Figure 41.) The pattern is locally characterized in the (s, t, n) surface frame as a pattern of surface markings given by $f(s, t)$. Points on this locally planar surface are given by coordinates $(s, t, 0)$. Applying the transformation matrix, the corresponding world coordinates are

$$\begin{aligned} X &= t_{11}s + t_{12}t + X_i \\ Y &= t_{21}s + t_{22}t + Y_i \\ Z &= t_{31}s + t_{32}t + Z_i \end{aligned} \tag{62}$$

Under perspective, these points project to the image plane at

$$\begin{aligned} x &= -d\frac{X}{Z} = -d\frac{t_{11}s + t_{12}t + X_i}{t_{31}s + t_{32}t + Z_i} \\ y &= -d\frac{Y}{Z} = -d\frac{t_{21}s + t_{22}t + Y_i}{t_{31}s + t_{32}t + Z_i} \end{aligned} \tag{63}$$

The origin of the (s, t, n) frame thus projects to $(x_i, y_i) = (-dX_i/Z_i, -dY_i/Z_i)$ on the image plane. In order to avoid carrying a coordinate offset through the calculations, we define another coordinate system, (x', y'), on the image plane that is centered at (x_i, y_i) with its axes parallel to those of the image plane. In terms of s and t, x' and y' are

$$\begin{aligned} x' &= x - x_i = -d\frac{t_{11}s + t_{12}t + X_i}{t_{31}s + t_{32}t + Z_i} - x_i \\ y' &= y - y_i = -d\frac{t_{21}s + t_{22}t + Y_i}{t_{31}s + t_{32}t + Z_i} - y_i \end{aligned} \tag{64}$$

Solving these two equations for s and t will give equations that give a point in the surface frame for any corresponding (x', y'). Doing so, using $(X_i, Y_i) = (-x_iZ_i/d, -y_iZ_i/d)$ and the orthonormality relationships among the vectors in the transformation matrix, we have

$$
\begin{aligned}
s(x', y') &= -\frac{Z_i\,[d\,(y' t_{12} - x' t_{22}) + t_{32}\,(y' x_i - x' y_i)\,]}{d\,[t_{13}\,(x' + x_i) + t_{32}\,(y' + y_i) - Z_i d]} \\
t(x', y') &= \frac{Z_i\,[d\,(y' t_{11} - x' t_{21}) + t_{31}\,(y' x_i - x' y_i)\,]}{d\,[t_{13}\,(x' + x_i) + t_{32}\,(y' + y_i) - Z_i d]}
\end{aligned}
\tag{65}
$$

Thus, if the brightness pattern on a locally planar patch on a textured surface is $f(s, t)$, then the projected pattern on the image plane is a nonlinear warping of the pattern given by $f\,[s(x', y'), t(x', y')]$.

To make the frequency analysis tractable, we will linearize this warping using a truncated Taylor series around $(x', y') = (0, 0)$. The approximation is justified since the spectrogram examines only a relatively small window of intensities around each point of interest. We have

$$
\begin{aligned}
s(x', y') &\approx s_x x' + s_y y' \\
t(x', y') &\approx t_x x' + t_y y'
\end{aligned}
\tag{66}
$$

with

$$
\begin{aligned}
s_x &= \frac{\partial}{\partial x'} s(x', y')\big|_{(x', y') = (0, 0)} = A\,[d\,(rp^2 + q^2) - q y_i\,(p^2 + q^2)\,] \\
s_y &= \frac{\partial}{\partial y'} s(x', y')\big|_{(x', y') = (0, 0)} = Aq\,[dp\,(r - 1) + x_i\,(p^2 + q^2)\,] \\
t_x &= \frac{\partial}{\partial x'} t(x', y')\big|_{(x', y') = (0, 0)} = Ap\,[dq\,(r - 1) + y_i\,(p^2 + q^2)\,] \\
t_y &= \frac{\partial}{\partial y'} t(x', y')\big|_{(x', y') = (0, 0)} = A\,[d\,(p^2 + rq^2) - p x_i\,(p^2 + q^2)\,]
\end{aligned}
\tag{67}
$$

where

$$
A = \frac{Z_i}{d\,(p^2 + q^2)\,(p x_i + q y_i - d)}
\tag{68}
$$

and where we have substituted the values of t_{ij} from Equation (61). The projected version of $f(s, t)$ is then approximately $f(s_x x' + s_y y', t_x x' + t_y y')$, which is just an affine transformation (without translation) of the coordinates. The four affine parameters are functions of the 3D location and surface orientation of the textured surface and the camera's effective focal length.

There are other ways to approximate perspective projection with an affine transformation. One of them is from Ohta *et al.*[76], whose approximation assumes that each texture element on a slanted plane is projected to a *frontal plane* that passes through the element's center-of-mass. Aloimonos[2] used the same projection model for some of his work in shape-from-texture. Kanade and Kender[50] developed the basis of their shape-from-texture algorithm by deriving the affine transformation between two texels on different planes under orthographic projection. Our approach is different from these others in that we write the exact perspective equations and then approximate mathematically using a truncated Taylor series. The others make a physical approximation first that leads directly to an affine transform.

3.3 Local Fourier Transform of an Image Texture

Since we are working with the spectrogram, our real interest is in the Fourier transform of the image patch. A property of the Fourier transform is that if a function in space undergoes an affine transformation, its Fourier transform will also undergo an affine transformation. This is given by the following Fourier transform[1] pairs[32]:

$$\begin{aligned} \Im\{f(x', y')\} &= F(u, v) \\ \Im\{f(s_x x' + s_y y', t_x x' + t_y y')\} &= \frac{1}{|D|} F\left(\frac{t_y}{D}u - \frac{t_x}{D}v, -\frac{s_y}{D}u + \frac{s_x}{D}v\right) \end{aligned} \qquad (69)$$

where $D = s_x t_y - s_y t_x$. We just showed that the connection between a frontal texture pattern and its perspective projection is approximately affine. The relationship above means the connection between their frequency distributions is also approximately affine.

The relationship in Equation (69) does not take the window into account. We show in Appendix 4 that when a window *is* taken into account, it is more accurate to drop the $1/|D|$ scale factor. We do this in the rest of this thesis and in our algorithms.

[1] Our notation for the 2D Fourier transform has u and v as the frequency variables, measured in cycles/unit distance. The 2D Fourier transform of $f(x, y)$ is given by

$$F(u, v) = \Im_{(x, y) \Rightarrow (u, v)}\{f(x, y)\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} f(x, y)\, e^{-j2\pi(ux + vy)}\, dx\, dy$$

The 2D inverse Fourier transform is given by

$$f(x, y) = \Im^{-1}_{(u, v) \Rightarrow (x, y)}\{F(u, v)\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} F(u)\, e^{j2\pi(ux + vy)}\, du\, dv$$

Uppercase functions usually indicate the Fourier transform of the corresponding, lower-case functions.

The *affine frequency scaling* induced on the frontal Fourier transform is not easy to interpret intuitively based on Equations (67). It is easier to interpret if we consider a special case: surface normal slanted to the right ($p > 0$) with no vertical component ($q = 0$) that is projected to the center of the image ($(x_i, y_i) = (0, 0)$). The affine parameters in the spatial domain then become

$$(s_x, s_y, t_x, t_y) = \frac{-Z_i}{d}(\sqrt{1+p^2}, 0, 0, 1) \tag{70}$$

(The Z_i turns out to be negated because all points in the visible scene have negative Z values.) The affine transformed Fourier transform is

$$F\left(\frac{d}{-Z_i\sqrt{1+p^2}}u, \frac{d}{-Z_i}v\right) \tag{71}$$

We can verify these frequency effects with our intuition in two ways. First, an increase in depth (larger absolute value of Z) dilates the Fourier transform equally in both dimensions. This makes sense, because objects appear smaller at greater distances, which translates into higher spatial frequencies. Second, slanting the patch more to the right (increased p) dilates the frequency along the horizontal direction, but has no effect on the vertical frequencies. This is plausible, since the increased slant will compress the texture only in the horizontal direction, leading to increased frequency along that direction.

3.4 Affine Connection

Since there is usually no way to determine what the frontally viewed texture looks like, we resort to comparing patches of the same texture at different locations in the image. We showed above that the Fourier transform of each patch is related to the Fourier transform of the frontally viewed texture by an affine transformation. This means that the *Fourier transforms of patches themselves are related by affine transformations*. We will show that if we assume two patches come from the same textured plane, then the affine parameters connecting them are functions of known camera and image parameters and the plane's surface normal.

Suppose the two texture patches in the scene are related to the frontally viewed texture by the affine parameters $(s_{x_1}, s_{y_1}, t_{x_1}, t_{y_1})$ and $(s_{x_2}, s_{y_2}, t_{x_2}, t_{y_2})$. The subscripts differentiate between the two patches. Transforming the Fourier transform of the first into the second with affine parameters (a_1, b_1, a_2, b_2) gives

$$F_1\left[a_1\left(\frac{t_{y_1}}{D_1}u-\frac{t_{x_1}}{D_1}v\right)+b_1\left(-\frac{s_{y_1}}{D_1}u+\frac{s_{x_1}}{D_1}v\right),a_2\left(\frac{t_{y_1}}{D_1}u-\frac{t_{x_1}}{D_1}v\right)+b_2\left(-\frac{s_{y_1}}{D_1}u+\frac{s_{x_1}}{D_1}v\right)\right]$$

$$= F_2\left[\frac{t_{y_1}}{D_2}u-\frac{t_{x_1}}{D_2}v,-\frac{s_{y_1}}{D_2}u+\frac{s_{x_1}}{D_2}v\right] \tag{72}$$

where $F_1(u, v)$ and $F_2(u, v)$ are the Fourier transforms of the two image patches. Note that we have ignored phase differences here. In reality, the Fourier phases of the two patches will be different. This difference is masked because each patch is defined with respect to its own local coordinate system. In our formulation, phase would only complicate the derivation, since we discard it by computing the Fourier transform's squared magnitude (power spectrum) in our algorithm.

Equating coefficients on u and v in Equation (72) leads to the following linear equation:

$$\frac{1}{D_1}\begin{bmatrix} t_{y_1} & -s_{y_1} & 0 & 0 \\ -t_{x_1} & s_{x_1} & 0 & 0 \\ 0 & 0 & t_{y_1} & -s_{y_1} \\ 0 & 0 & -t_{x_1} & s_{x_1} \end{bmatrix}\begin{bmatrix} a_1 \\ b_1 \\ a_2 \\ b_2 \end{bmatrix} = \frac{1}{D_2}\begin{bmatrix} t_{y_2} \\ -t_{x_2} \\ -s_{y_2} \\ s_{x_2} \end{bmatrix} \tag{73}$$

whose solution is

$$\begin{bmatrix} a_1 \\ b_1 \\ a_2 \\ b_2 \end{bmatrix} = \frac{1}{D_2}\begin{bmatrix} s_{x_1}t_{y_2}-s_{y_1}t_{x_2} \\ t_{x_1}t_{y_2}-t_{x_1}t_{y_2} \\ s_{x_1}t_{y_2}-s_{y_1}t_{x_2} \\ s_{x_1}t_{y_2}-s_{y_1}t_{x_2} \end{bmatrix} \tag{74}$$

Thus, the affine parameters connecting the two Fourier transforms are functions of the affine parameters connecting the two patches to the frontally viewed texture. In order to relate this equation to the physical parameters of the camera and the textured surface, we take the values of $(s_{x_1}, s_{y_1}, t_{x_1}, t_{y_1})$ and $(s_{x_2}, s_{y_2}, t_{x_2}, t_{y_2})$ from Equation (67). Before doing this, however, we will make the assumption that the two texture patches have the same surface normal, *i.e.* $(p_1, q_1) = (p_2, q_2) = (p, q)$, and that both patches are on the same plane, *i.e.*

$$\frac{Z_2}{Z_1} = \frac{d-px_1-qy_1}{d-px_2-qy_2} \tag{75}$$

Substituting values from Equation (67), the affine parameters connecting the Fourier transforms of the two patches are then

$$\begin{aligned} a_1 &= B\left[(-d^2r)(p^2+q^2) + dr(p^3x_1 + p^2qy_1 + pq^2x_2 + q^3y_2) + dpq(q\Delta x - p\Delta y) - pq(p^2+q^2)(x_1y_2 - x_2y_1)\right] \\ b_1 &= Bq\left[(-drp)(p\Delta x + q\Delta y) - dq(q\Delta x - p\Delta y) + q(p^2+q^2)(x_1y_2 - x_2y_1)\right] \\ a_2 &= Bp\left[(drq)(p\Delta x + q\Delta y) - dp(q\Delta x - p\Delta y) + p(p^2+q^2)(x_1y_2 - x_2y_1)\right] \\ b_2 &= B\left[(-d^2r)(p^2+q^2) + dr(p^3x_2 + p^2qy_2 + pq^2x_1 + q^3y_1) - dpq(q\Delta x - p\Delta y) + pq(p^2+q^2)(x_1y_2 - x_2y_1)\right] \end{aligned} \tag{76}$$

where

$$\begin{aligned} B &= \frac{px_1 + qy_1 - d}{dr(p^2+q^2)(px_2 + qy_2 - d)^2} \\ r &= \sqrt{p^2+q^2+1} \\ \Delta x_0 &= x_1 - x_2 \\ \Delta y_0 &= y_1 - y_2 \end{aligned} \tag{77}$$

The notable feature of these equations is that the only unknowns are (p, q). This allows us to write a simple algorithm that determines the correct surface normal by finding the (p, q) that generates the affine parameters that best transform one patch into another. In our algorithm we actually use the squared magnitude of the Fourier transform, but the same affine parameters apply.

These affine parameters are hard to interpret intuitively, but several special cases support their plausibility. One special case is orthography. *Mathematica* says that as the focal length d goes to infinity, the affine parameters go to the identity transformation, *i.e.*

$$\lim_{d \to \infty} (a_1, b_1, a_2, b_2) = (1, 0, 0, 1) \tag{78}$$

This is what we expect. The frequency shifts depend on a depth difference, and the depth differences become zero in the case of orthography. We also get the identity transform as the surface normal becomes parallel with the camera's optical axis, that is

$$\lim_{(p, q) \to (0, 0)} (a_1, b_1, a_2, b_2) = (1, 0, 0, 1) \tag{79}$$

In addition, we get the identity transform when the patches are coincident, which is when both $x_1 = x_2$ and $y_1 = y_2$.

Another special case has a planar surface slanted to the right ($p > 0$) with no vertical component ($q = 0$). If the two patches are both on the center row ($y_1 = y_2 = 0$) and placed symmetrically around the center column separated by a distance a ($x_1 = -a/2$ and $x_2 = a/2$), then

$$\begin{aligned} a_1 &= \left(\frac{2d-ap}{2d+ap}\right)^2 \\ b_1 &= 0 \\ a_2 &= 0 \\ b_2 &= \frac{2d-ap}{2d+ap} \end{aligned} \qquad (80)$$

Recalling that the connection between the two local power spectra is $|F_1(a_1u+b_1v, a_2u+b_2v)|^2 \approx |F_2(u, v)|^2$, we see that the frequency shifts in this case are a simple scaling of the frequency axes with no shearing. It is usually the case that $2d > |ap|$. In this case, for positive p, both a_1 and b_2 will be less than one, meaning that the patch on the left will have lower frequencies than the patch on the right. Since the right patch is more distant in this case, this makes sense.

Typical numerical values can be computed by assuming a 35mm camera ($0\text{mm} < a < 35\text{mm}$) with a 50mm lens ($d = 50\text{mm}$). If the surface is slanted to the right at 45°, then $p = 1$. If the two power spectrum patches are on opposite sides of the image, then $a = 35\text{mm}$ and $(a_1, b_1, a_2, b_2) \approx (0.23, 0, 0, 0.48)$. If the patches are only 1mm apart on the film, then $(a_1, b_1, a_2, b_2) \approx (0.96, 0, 0, 0.98)$.

The derivation in Sections 3.2 to 3.4 constitutes the basic math for our first two shape-from-texture algorithms. To summarize, we first showed how a locally planar surface patch projects by perspective into the image. Since this projection is complicated, we approximated it with a truncated Taylor series. This gave an affine relationship between the frontally viewed texture and the projected texture. A property of the Fourier transform says that an affine transformation in space is an affine transformation in frequency. Since the Fourier transform of each image patch is related by an affine transformation to the Fourier transform of the frontally viewed texture, the Fourier transforms of the image patches are also related by an affine transformation. If we assume the two patches are on the same plane, the affine parameters that connect their Fourier transforms are functions of known camera parameters and the unknown surface normal.

3.5 Algorithm1: SSD Patch Matching

Our first shape-from-texture algorithm consists of an exhaustive search for the (p, q) that gives the best affine parameters between two patches. This section describes the algorithm, gives results, shows how it can work in spite of aliasing, and discusses sensitivity issues.

3.5.1 Search in (p,q) Space

Our algorithm begins with two points on the image plane chosen by the user. We multiply the 64x64 neighborhood around each point by a radially symmetric Blackman-Harris minimum four-sample window (Section 2.2.3). We compute the power spectrum of these products using a 2D FFT program. This gives us part of the spectrogram: $S(x_1, y_1, u, v)$ and $S(x_2, y_2, u, v)$. We could also include phase information by using the STFT instead. Phase could be useful for periodic textures in a light-striping-like algorithm. However, the phase information in a random texture would be useless. In addition, if part of a texture is occluded as in Figure 31, the phase information would be misleading, because the number of wavelengths traversed by the texture in the occluded region is unknown. In order to match the phases of two patches, we would have to use a six-parameter affine transformation (including translation) rather than the four-parameter version (no translation) that we use now. By ignoring phase, we can reduce the complexity of the affine transformation and speed up the program.

The last step of our core algorithm is an exhaustive search for the (p, q) that best transforms the power spectrum of one patch into another. Given a candidate (p, q) from the search grid, we compute the corresponding affine parameters from Equation (76), use these to transform the power spectrum of the first patch using bilinear interpolation, and compute the sum of squared differences (ssd) between the two power spectra. Our search grid is 60x60, spread linearly over $(|p|, |q|) \leq (2, 2)$. We take the (p, q) that generates the minimum ssd as the solution.

We summarize the magic numbers in this algorithm in Table 7. We investigate the effect of window size in the next section, and the effect of the positions of the two patches in Section 3.5.5 on sensitivity.

3.5.2 Results

In Figure 34 - Figure 43 we show ten images of periodic, textured plates. The first four were created with a computer graphics program. We used a page-scanner to digitize Brodatz[10] textures and mapped these onto four geometrically identical flat plates. Although this eliminates the 3D relief

description	value(s)
positions of two image patches	same row, spaced far horizontally
window size	64x64 pixels
search grid resolution	60x60
search grid limits	$(\|p\|, \|q\|) \le (2, 2)$

Table 3: Magic numbers used in SSD patch-matching algorithm

effect of nonfrontal textures, it gives us exact ground truth. The effect of camera noise is also introduced by the scanner.

The last six images, Figure 38 - Figure 43 were taken in Carnegie Mellon University's Calibrated Imaging Laboratory (CIL) with a cooled, slowscan, Photometrics camera. We used a Nikon 50mm lens set at f/22 (small aperture) focused at about two meters. Our shape-from-texture equations require that we know the pixel size and focal length of the camera. We took the pixel size from the manufacturer's specifications (0.023mm x 0.023mm). For calibrating the focal length, we used Willson's camera calibration program (publicly archived, anonymous ftp to FTP.TELEOS.COM). The input to the program is a file of 3D points in space and their corresponding 2D points on the image plane. We created these from images of a back-lit aluminum plate with small holes drilled on one inch centers. We mounted the plate on an optical table so it was approximately perpendicular to the optical axis of the camera, took the first calibration image, moved the plate back 36 inches, took another calibration image, moved the plate back another 36 inches, and took the final calibration image. The relative positions of the 3D points were known accurately because of the accurately drilled holes on the optical table and the aluminum plate. We found the 2D image locations by finding the centroids.

Willson's program first performs an approximate calibration using Tsai's calibration method[104]. It then uses an IMSL optimization routine to refine the result. This program finds the six extrinsic parameters that locate the camera as well as the five intrinsic parameters of focal length, pixel height to width ratio, Tsai's radial lens distortion parameter κ_1, and the image center. The focal length for these images turned out to be 52.11 mm. We estimated the actual surface normals of the textured plates either by trigonometry based on their position on the optical table or by an electronic level on the camera platform if the object were lying flat on the floor.

The results are given in Table 8. The surface normal error varies from a low of $1.2°$ for the "bumpy plastic" texture to a high of $6.1°$ for the "cotton canvas" and the "grid paper". The average error for the ten images is $3.5°$, which is about as accurate as any shape-from-texture program on

texture	true (p, q)	true (σ, τ)	computed (p, q)	computed (σ, τ)	error
woven aluminum wire (D6)	(0.614, 0.364)	(35.5°, 30.7°)	(0.576, 0.305)	(33.1°, °27.9)	2.9°
French canvas (D21)	(0.614, 0.364)	(35.5°, 30.7°)	(0.576, 0.373)	(34.5°, 32.9°)	1.7°
oriental straw cloth (D53)	(0.614, 0.364)	(35.5°, 30.7°)	(0.644, 0.305)	(35.5°, 25.3°)	3.1°
cotton canvas (D77)	(0.614, 0.364)	(35.5°, 30.7°)	(0.644, 0.237)	(34.5°, 20.2°)	6.1°
screen guard	(0.714, 0.0)	(35.5°, 0.0°)	(0.576, 0.034)	(30.0°, 3.4°)	5.8°
grid paper	(−1.878, 0.0)	(62.0°, 180.0°)	(−1.525, −0.10	(56.8°, 183.8°)	6.1°
checkered paper	(−1.878, 0.0)	(62.0°, 180.0°)	(−1.797, 0.034	(60.1°, 179.0°)	1.4°
egg crate	(−1.373, 0.0)	(53.9°, 180.0°)	(−1.186, −0.03	(49.9°, 181.6°)	4.3°
bumpy plastic	(−1.373, 0.0)	(53.9°, 180.0°)	(−1.390, −0.03	(54.3°, 181.4°)	1.2°
doormat	(−1.373, 0.0)	(53.9°, 180.0°)	(−1.254, −0.10	(51.5°, 184.7°)	2.8°
average error					3.5°

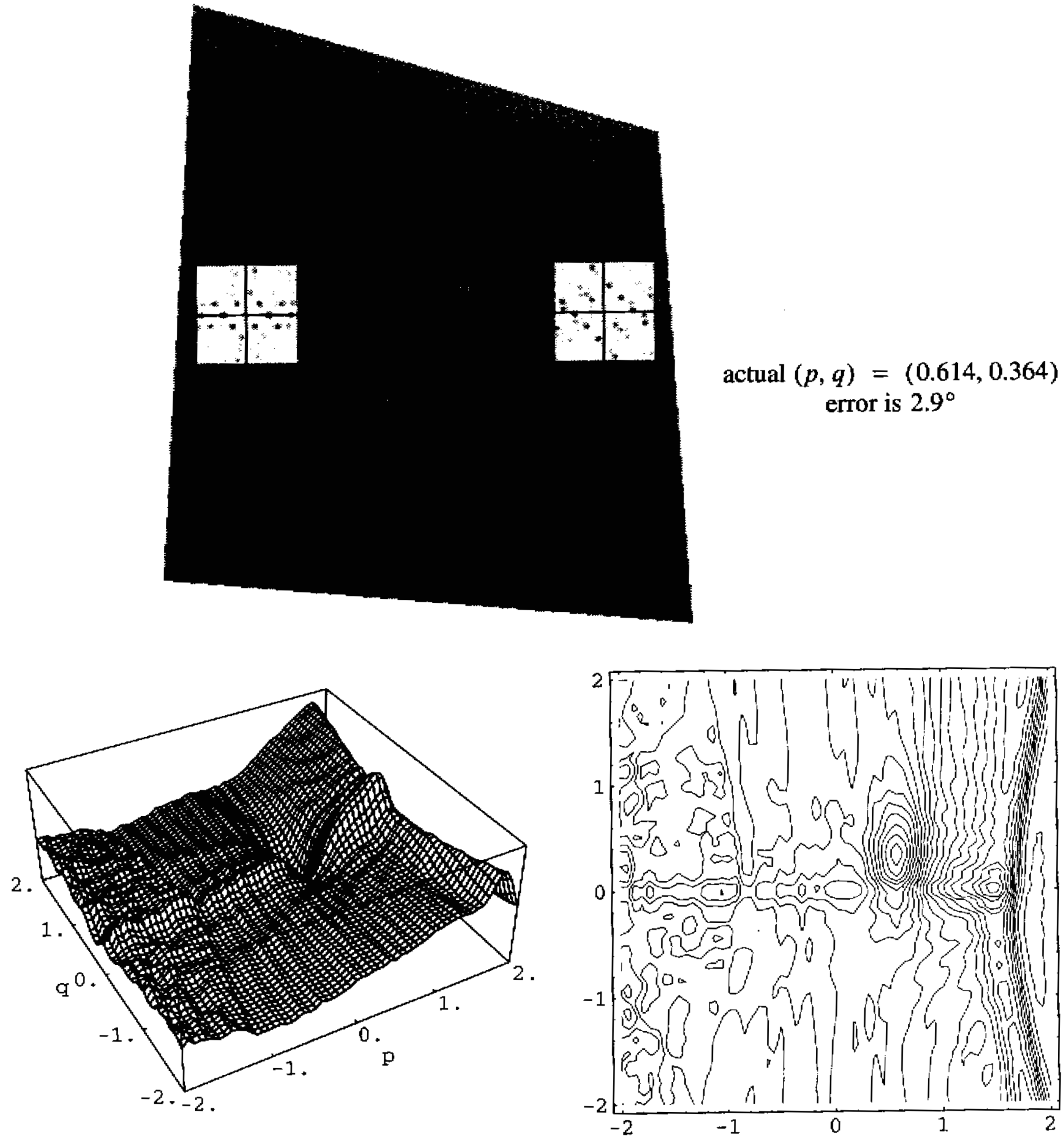
Table 4: Results of ssd patch matching on textured plates in Figure 34 - Figure 43.

these types of texture. (See Table 11 on page 135 for a comparison.). This is better when considered with the fact that we are using less than 4% of the pixels in each image, while most shape-from-texture programs use 100% of the image pixels. The egg crate texture in Figure 41 shows that the program gives a reasonable result in spite of the effects of 3D relief. In this image, the black background shows through on the right, but gradually disappears toward the left.

The ssd search spaces are shown along with each test image. The shape and broadness of the valleys around the minima give some indication of the sensitivity of the solution. We examine these shapes more closely in Section 3.5.5. The ssd surfaces are bumpy, and they have multiple, local minima (indicated by some of the closed contours in the contour plots). This is one reason that we use an exhaustive search for the minimum.

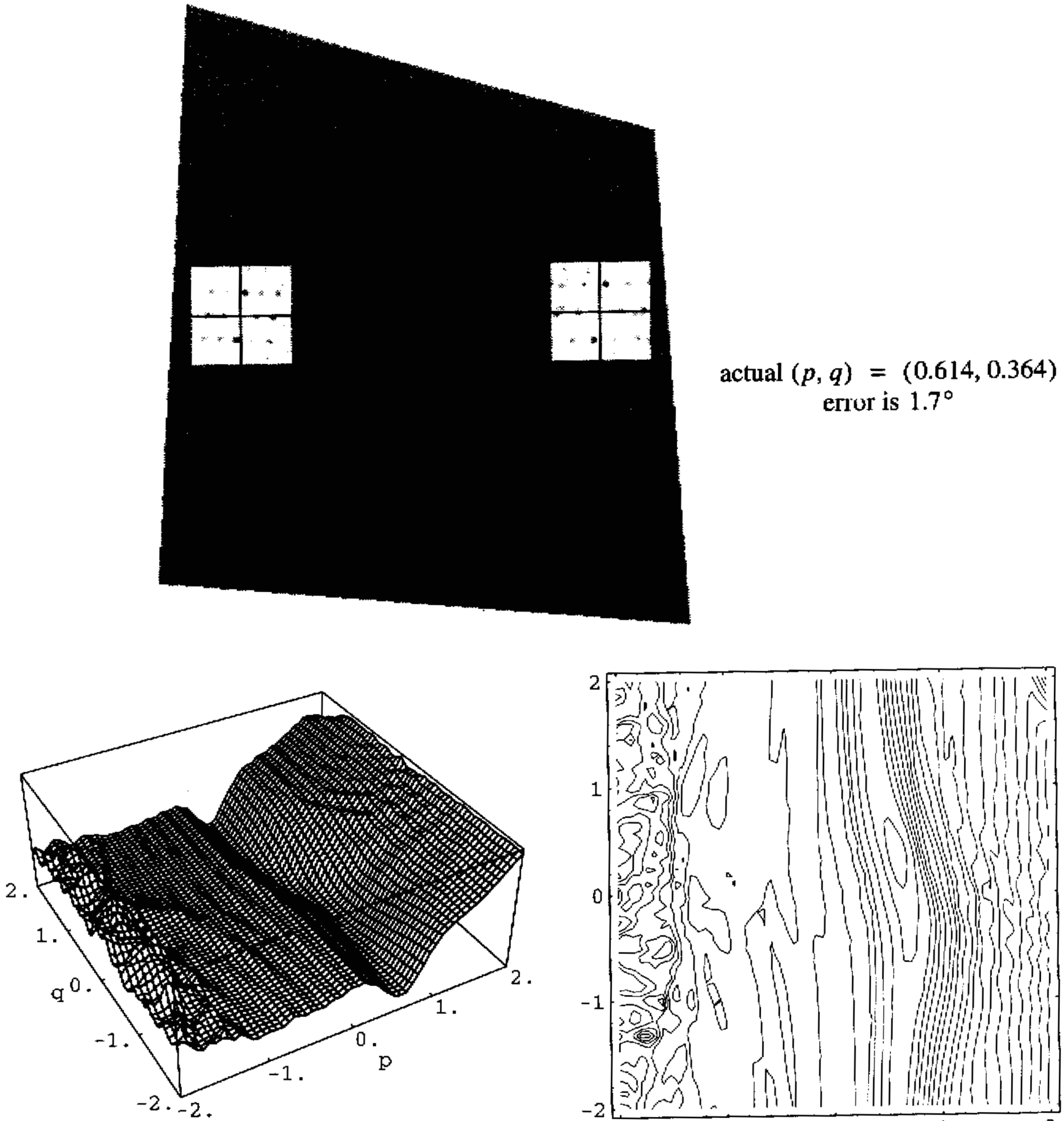
We investigated the effect of window size by running our program on the four Brodatz textures with different window dimensions. For these tests, we increased the resolution of our search in (p, q) space from 60x60 to 200x200 to factor out any effects of a too-coarse search grid. The windows were centered at the same two points in the images regardless of the window size. The results are show in Figure 34. The abscissa is the length of a side of the square window in pixels. The ordinate shows the surface normal error in degrees. For all four textures, an undersized window causes inaccuracy. This is probably because the small window gives only a low resolution Fourier transform. For these textures, window sizes between 30 and 120 are best. In this range the error is nearly independent of the window size, which is a good feature of the algorithm. The left and right windows started overlapping the left and right edges of the textured regions when the windows' size reached

Algorithm1: SSD Patch Matching



actual $(p, q) = (0.614, 0.364)$
error is 2.9°

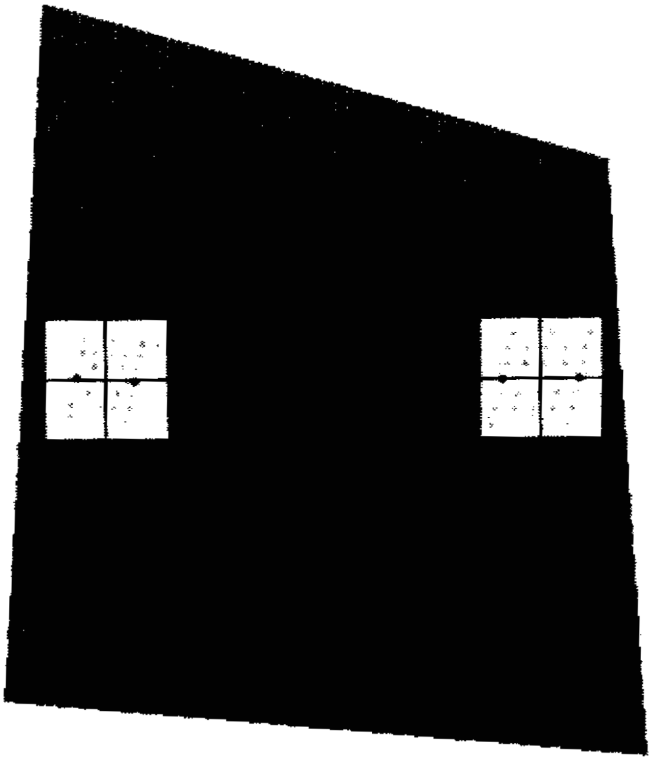
Figure 34: Textured plate used for testing shape-from-texture algorithms. This is a page-scanned Brodatz texture, "woven aluminum wire" (D6), mapped onto a flat plate with a computer graphics program. The two power spectra shown on the plate were used for the ssd patch matching algorithm. The resulting (p,q) search space is shown below.



actual (p, q) = (0.614, 0.364)
error is 1.7°

Figure 35: Page-scanned Brodatz "French canvas" (D21)

Algorithm1: SSD Patch Matching



actual (p, q) = (0.614, 0.364)
error is 3.1°

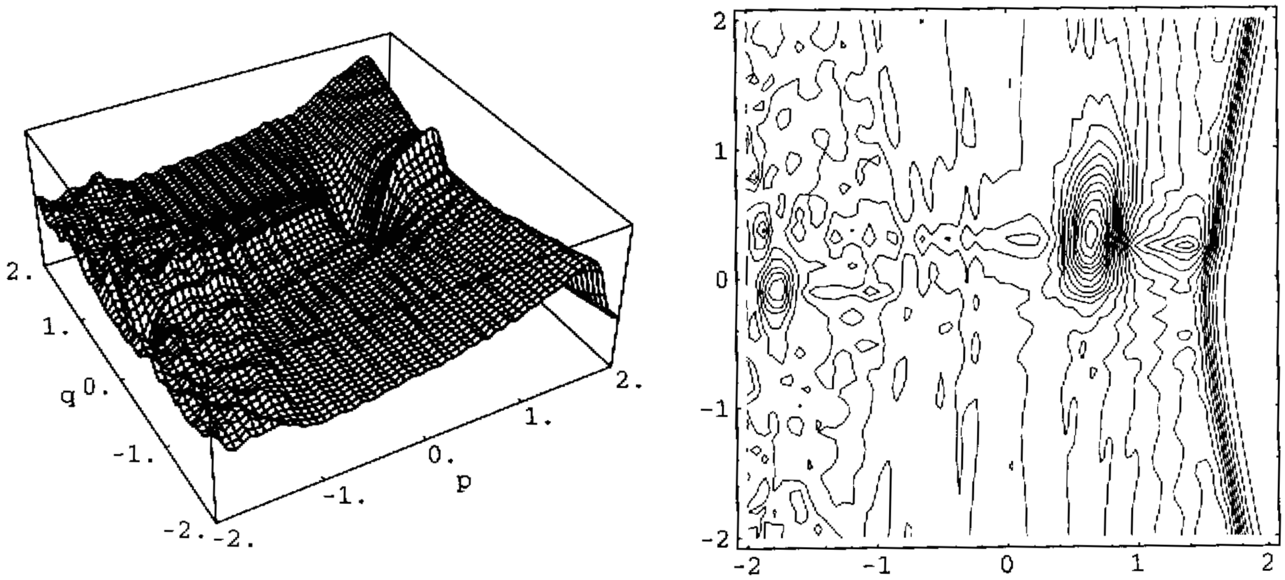
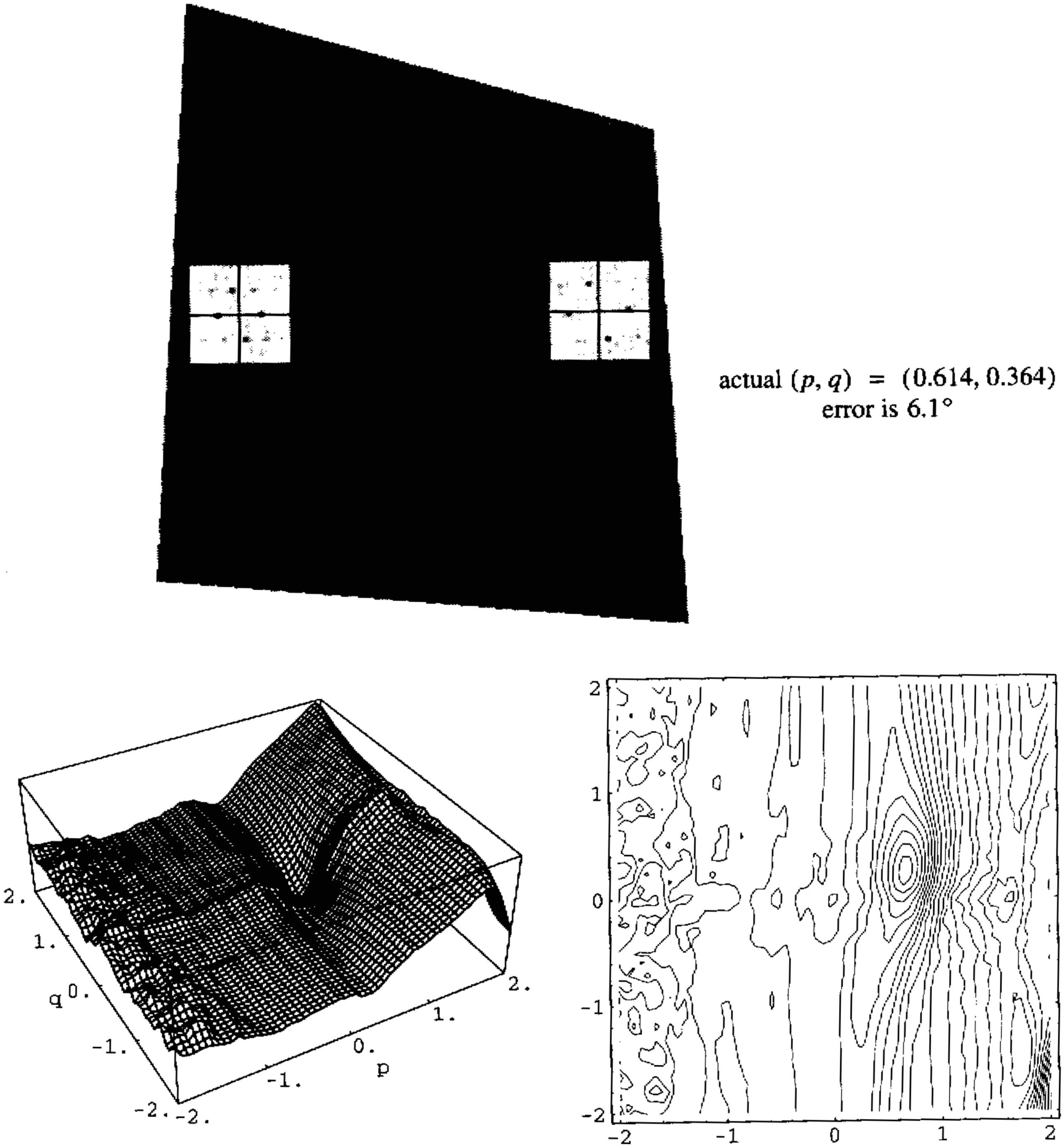
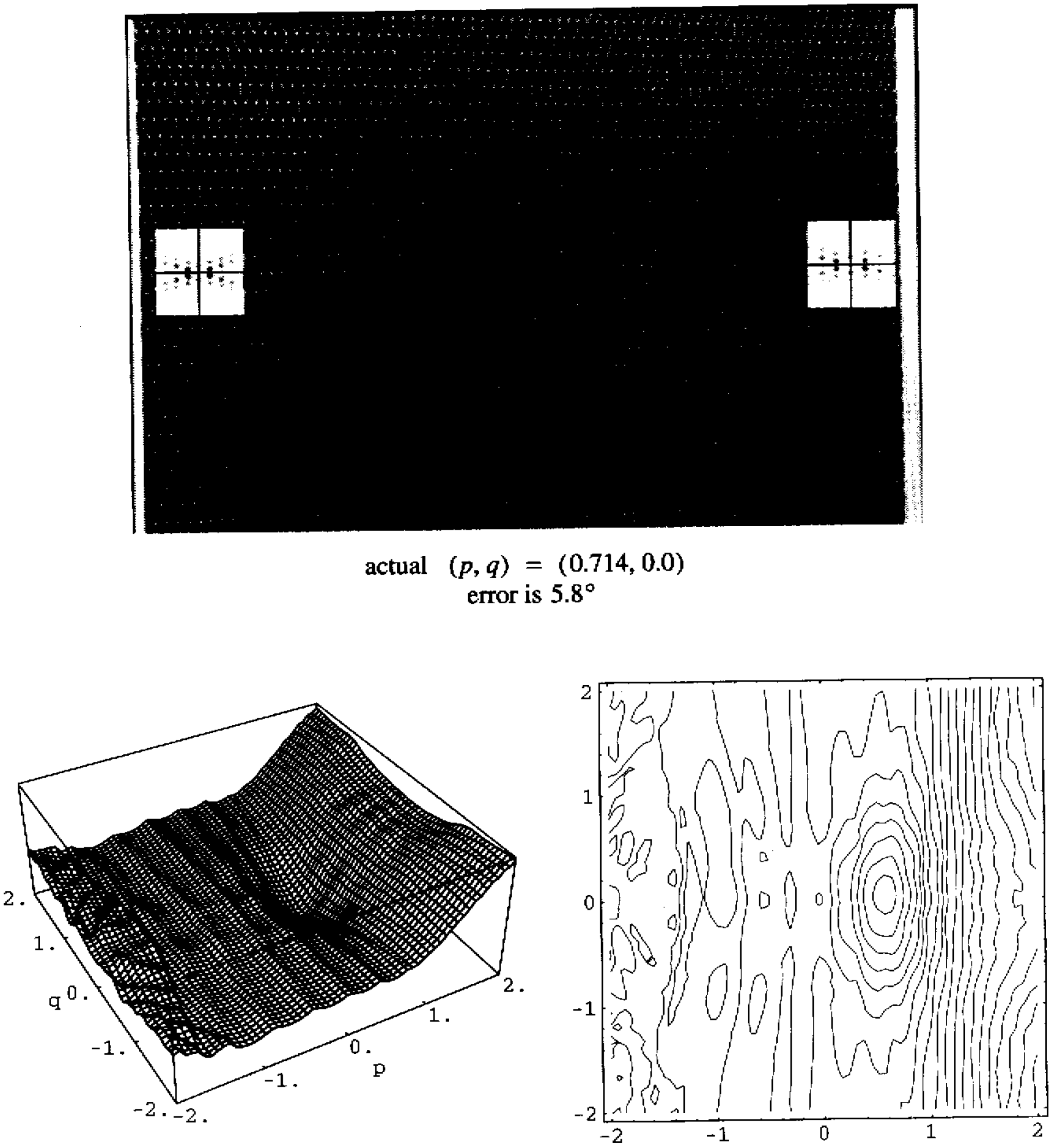


Figure 36: Page-scanned Brodatz "oriental straw cloth" (D53)



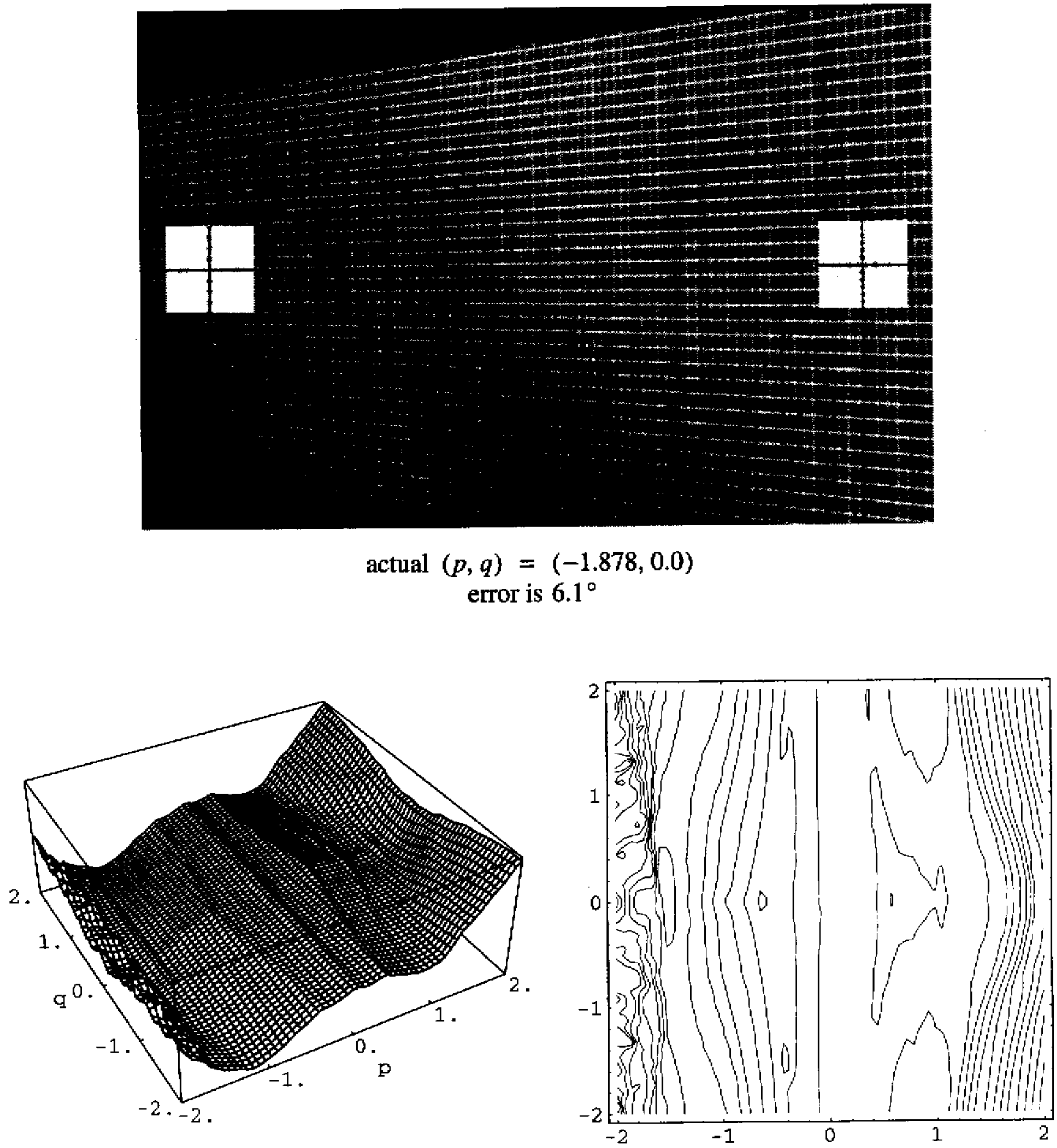
actual (p, q) = (0.614, 0.364)
error is 6.1°

Figure 37: Page-scanned Brodatz "cotton canvas" (D77)



actual $(p, q) = (0.714, 0.0)$
error is 5.8°

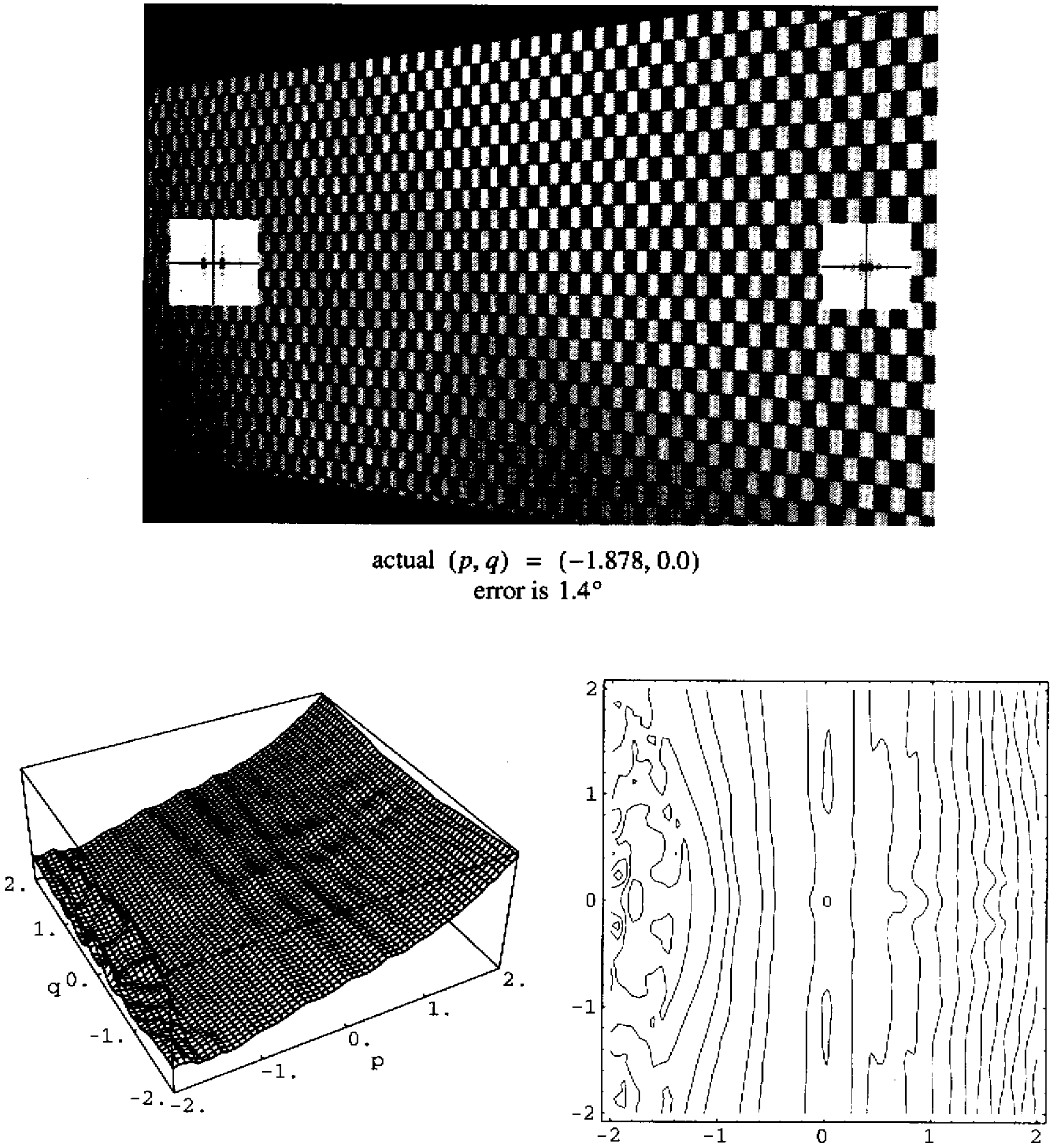
Figure 38: This is an actual image taken in the Calibrated Imaging Laboratory (CIL) at CMU. It is a painted metal screen guard.



actual $(p, q) = (-1.878, 0.0)$
error is 6.1°

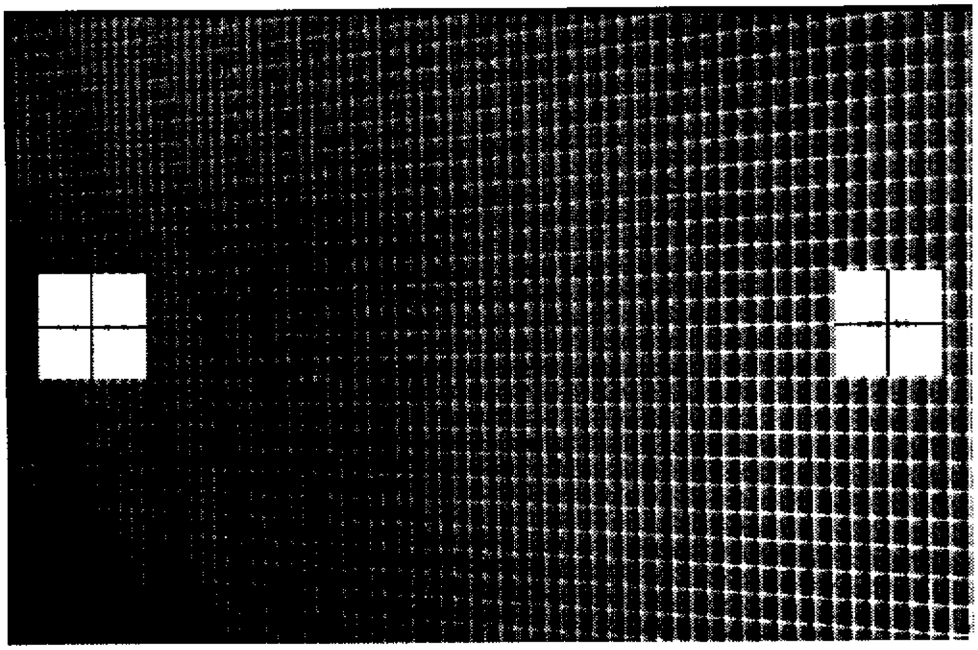
Figure 39: CIL image of shelf paper with a grid pattern.

Algorithm1: SSD Patch Matching



actual $(p, q) = (-1.878, 0.0)$
error is 1.4°

Figure 40: CIL image of checkered shelf paper



actual $(p, q) = (-1.373, 0.0)$
error is 4.3°

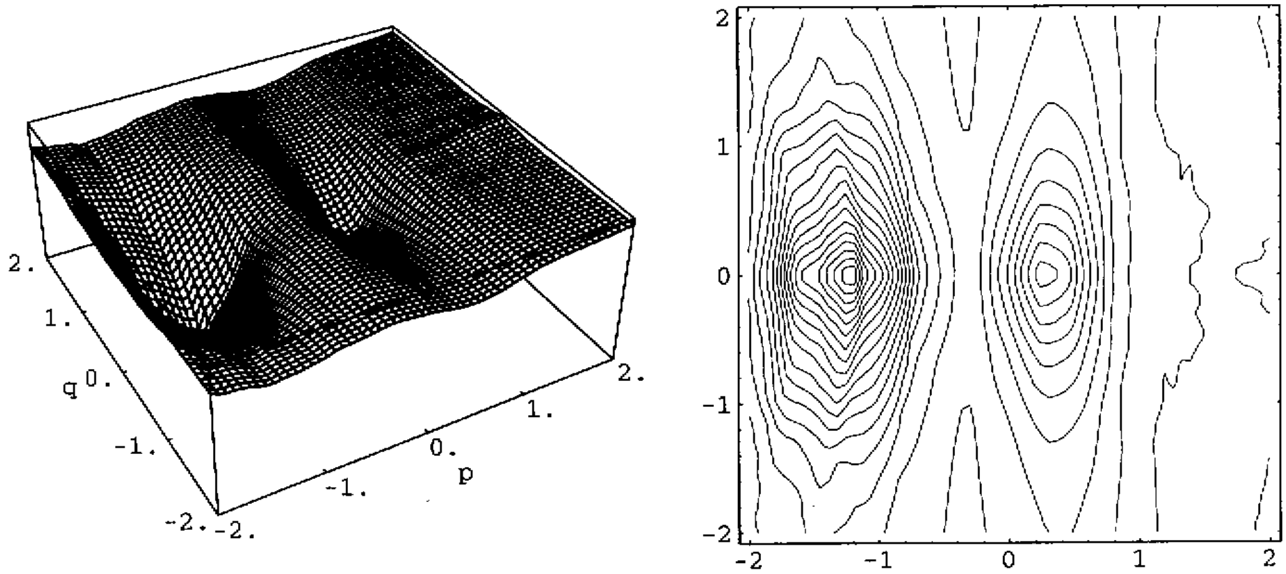
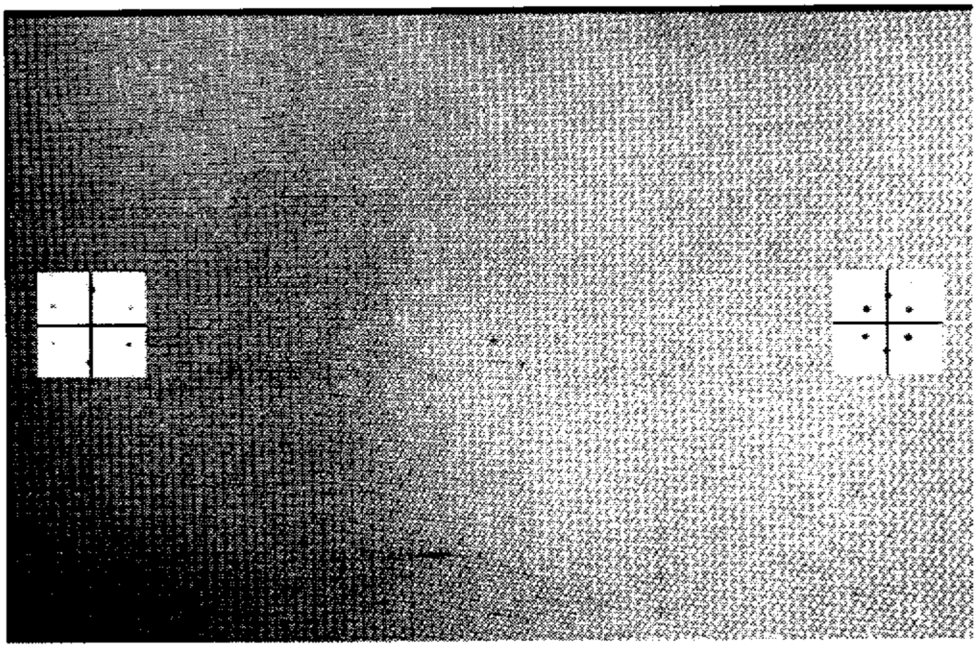


Figure 41: CIL image of a plastic, fluorescent light cover with an egg crate pattern. We get a reasonable result in spite of the pronounced, 3D relief effects.

Algorithm1: SSD Patch Matching



actual $(p, q) = (-1.373, 0.0)$
error is 1.2°

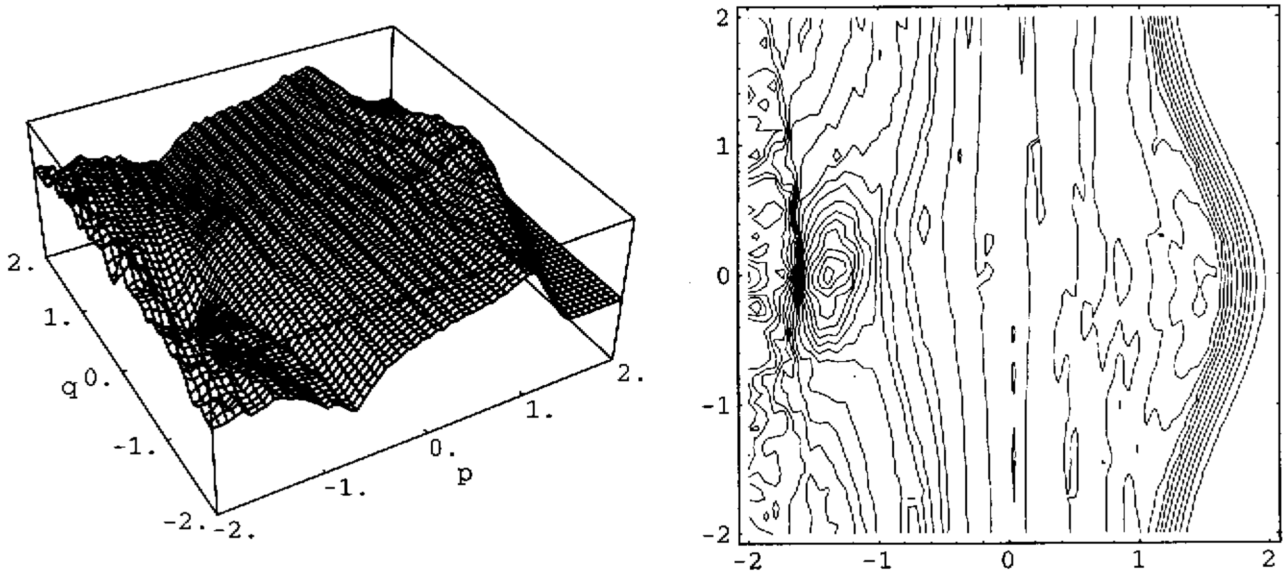
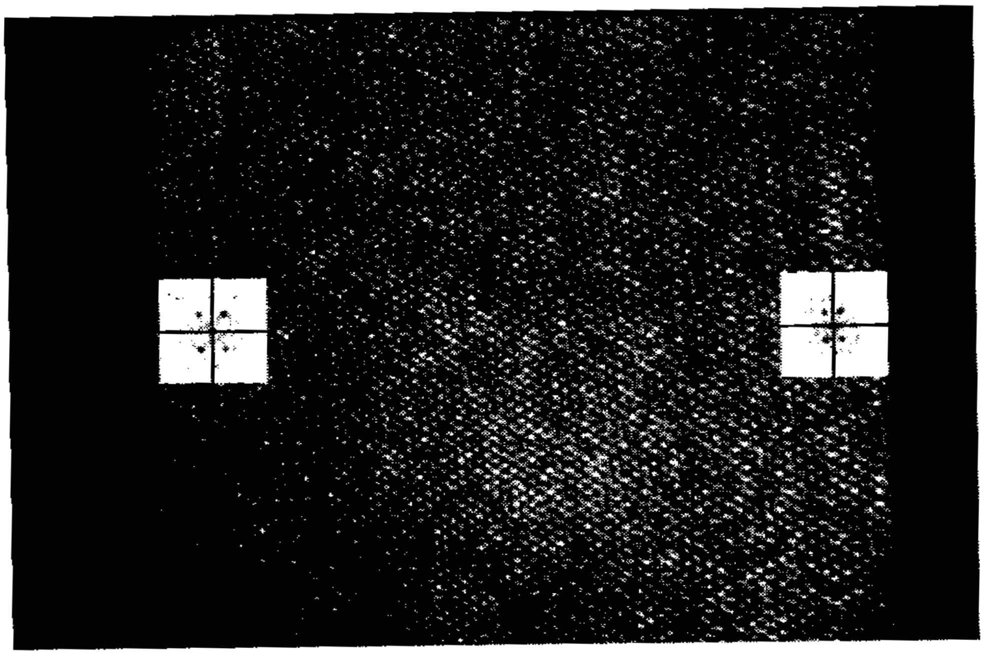


Figure 42: CIL image of a plastic, fluorescent light cover with small bumps



actual $(p, q) = (-1.373, 0.0)$
error is 2.8°

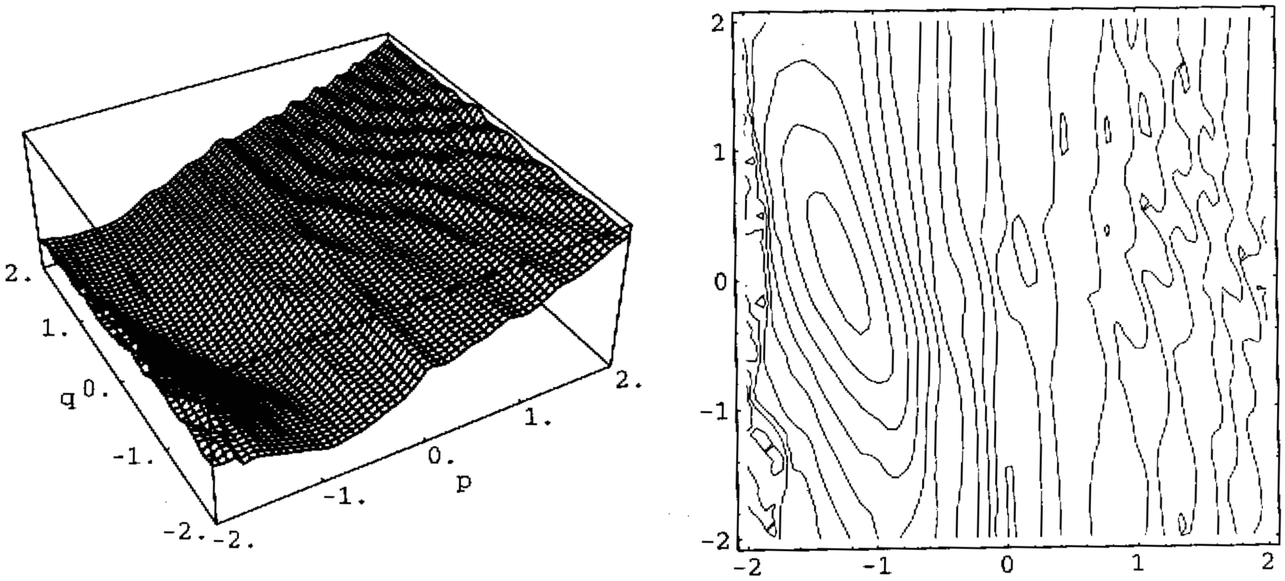


Figure 43: CIL image of a doormat

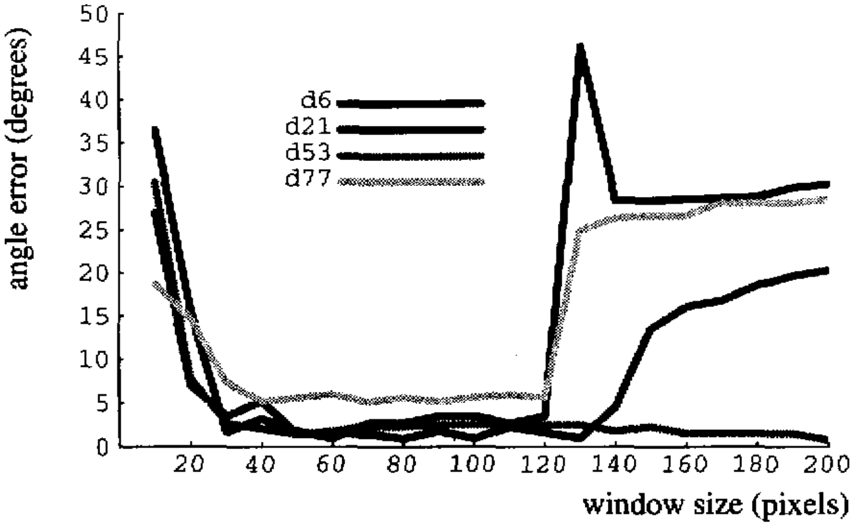


Figure 44: Angle error versus window size for the four Brodatz textures in Figure 34 through Figure 37. The windows begin overlapping the edge of the textured region at a window size of 90.

about 90 pixels. This did not adversely affect the accuracy until the size reached 120, at which point either the overlapping into the background or the smearing caused by the large shift in the textures' frequencies caused a sharp increase in error for all but one of the textures.

3.5.3 Aliasing

Although aliasing can cause real problems in image understanding, it is rarely dealt with explicitly in machine vision algorithms. This section describes how a simple modification to our SSD patch-matching algorithm can make it work in spite of aliasing.

Aliasing occurs when the image projected on the sampling grid has spatial frequencies that are higher than half the spatial sampling rate. If the aliased pattern is periodic, moire patterns appear. We showed in Figure 8 how a moire pattern was accidentally produced in a stereo pair that caused erroneous results. Another moire pattern is shown in Figure 45. The texture here consists of two fairly high frequency 2D cosines. One cosine runs diagonally from the lower left to the upper right, and the other is perpendicular to this one. The left side of the plate is not aliased, while the right side is, because the projected frequencies have grown beyond half the sampling rate. The series of local power spectra across the center of the image show what happens to the frequencies. As the peaks move out from the center, they approach the edges of the squares. The squares' edges are at half the sampling rate, and thus represent the highest frequencies that can be successfully sampled. The peaks in the first and third quadrants hit the edges in the fourth square from the left. In the next square to the right, they

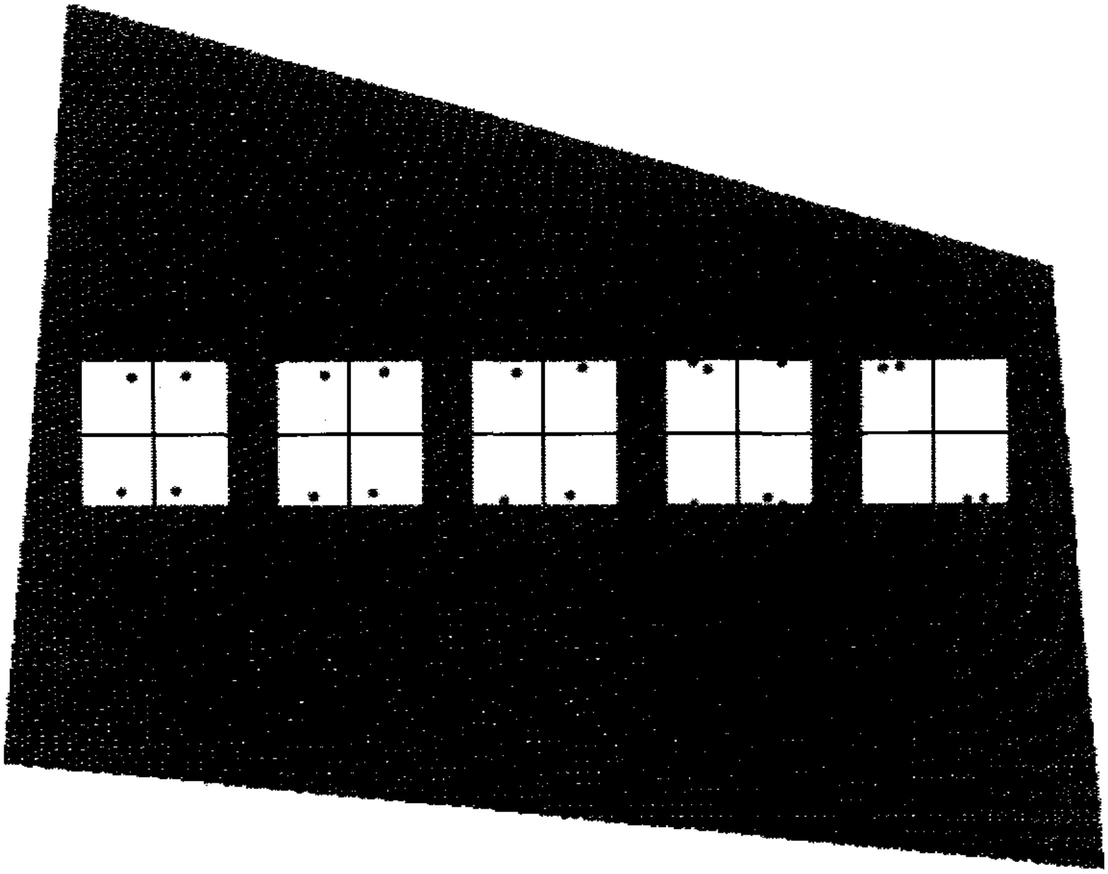


Figure 45: Plate showing aliasing on the right. There is no aliasing on the left. Going to the right, two of the delta functions in the power spectrum move to the edge of the window and then reappear elsewhere.

reappear in the second and fourth quadrants along with the peaks that were already there. This is the onset of aliasing. In the last square the aliased peaks have moved a little more back into the square.

If the sampling rates in the x and y directions are u_s and v_s respectively, then any (u, v) outside the boundaries $(\pm u_s/2, \pm v_s/2)$ will be aliased. It can be shown that the aliased frequency will be given by

$$(u_{\text{aliased}}, v_{\text{aliased}}) = 0.5\,(u_s \, \text{saw}_{u_s}(u), v_s \, \text{saw}_{v_s}(v)) \tag{81}$$

where

$$\text{saw}_T(x) = \frac{2}{T}\left(x - T\left\lfloor\frac{x + \frac{T}{2}}{T}\right\rfloor\right) \tag{82}$$

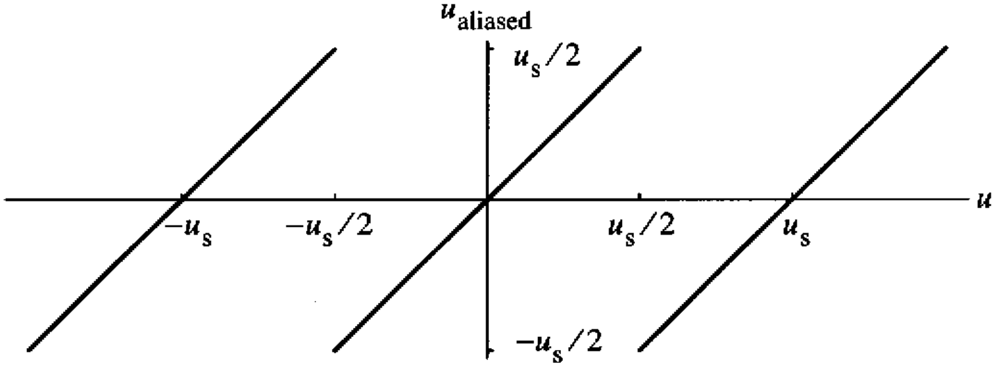


Figure 46: Aliased frequency as a function of actual frequency when the sampling frequency is u_s.

with $\lfloor x \rfloor$ being the "floor" function, returning the largest integer not exceeding x. The function $\text{saw}_T(x)$ has a period of T. We show a plot of u_{aliased} as a function of u in Figure 46. It shows how the unaliased frequency rises and then reappears at a different frequency when aliasing occurs.

Our algorithm allows us to account for aliasing very easily. When we test a given (p, q), we warp the frequency coordinates in one power spectrum by an affine transformation. We simply put all the transformed (u, v)'s through Equation (81) to adjust them for aliasing. This way, if a given (p, q) causes frequencies to be transformed outside the half-sampling-frequency limits, they will be aliased back in at the proper coordinates. This is also a convenient way of making sure both frequency patches overlap exactly, instead of having one skewed off the other with no corresponding frequencies in the other patch after the affine transformation.

We ran our algorithm on the left and right patches in Figure 45 and got an error of about 4.5^o. Thus the method successfully accounts for aliasing. The one restriction is the assumption that the first patch is not aliased.

We know of no other shape-from-texture algorithm that can account for aliasing even in this simple case. We attribute the ability to the fact that the space/frequency representation preserves essentially all the data in the original signal and that frequency is the natural domain for the analysis of aliasing.

3.5.4 Irregular Textures

We take "irregular" textures as those that are not periodic (and not skew-periodic, see Appendix 3). Compared to the periodic textures that we have been working with, it is more difficult to get a good estimate of the local spectra of irregular textures, and thus they are harder to analyze using the

description	value(s)
center-to-center window spacing	16 pixels horizontally and vertically
window size	128x128 pixels
neighborhood for spectrum patch averaging	7x7 patches
search grid resolution	40x40
search grid limits	$(\lvert p\rvert, \lvert q\rvert) \leq (2, 2)$

Table 5: Magic numbers used in SSD patch-matching algorithm for irregular textures

spectrogram. There is a large literature on spectrum estimation. Some of the techniques use a parametric model of the expected spectral power distribution. This would require us to make an assumption about the shape of the texture's power spectrum. A simpler technique is to average in the power spectrum domain. This is our approach. We began with a spectrogram using 128x128-pixel windows on 16-pixel centers. Larger windows gave us slightly better results for these test images. We computed an average spectrogram by averaging over 7x7 neighborhoods of power spectrum patches. We compared more than just one pair of patches. We used the averaged patches on the perimeter of the largest rectangle of patches that would fit within the textured part of the image. We paired each averaged patch with the patch directly opposite across the center of the rectangle, as shown in Figure 47. Each pair of patches gave one ssd surface. We summed these surfaces and took the location of the minimum as the surface normal. The magic numbers for this algorithm are shown in Table 7.

We ran this algorithm on eight different Brodatz textures, all mapped onto identical flat plates. The first four are the same as in Figure 34 to Figure 37, and the second four are irregular textures. The first two irregular textures are Brodatz "oriental straw cloth" (D78) and "loose burlap" (D104) in Figure 48 and Figure 49. These two textures are, perhaps, "pseudo-periodic" in that they are repeated patterns on a randomly perturbed grid. The averaging of the power spectrum patches tends to smooth over the perturbations. The second two irregular textures are Brodatz "handmade paper" (D57) and "plastic bubbles" (D111) in Figure 50 and Figure 51. These are more random than the other textures.

The results of our algorithm on these eight textures are shown in Table 8. The average error for the four periodic textures is 4.9°, while the average for the irregular textures is 7.5°. The average error for the four periodic textures using this averaging algorithm is not much worse than for the non-averaging algorithm. The slight increase in error may be because the larger windows tend to smear the shifting frequency peaks. In general, less regularity leads to more error. The periodic textures have the lowest error, the two pseudo-periodic textures are next, and the two most random textures have the highest error. We note that the ssd surfaces for the irregular textures have much wider depressions

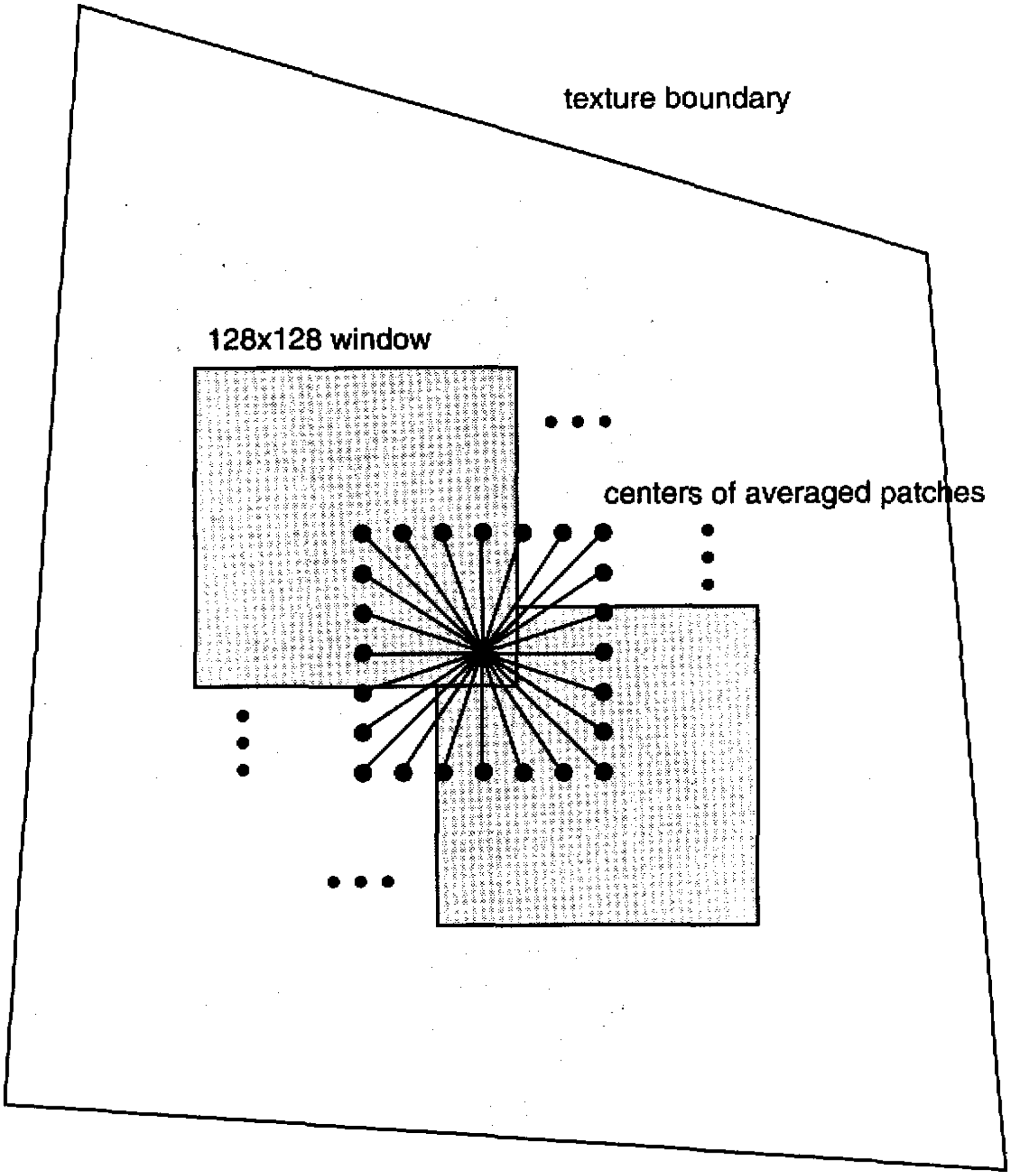


Figure 47: Scale drawing of windows used for computing surface normals of irregular textures. We used 12 unique pairs of windows arranged around the perimeter of a rectangle.

around their minima that for the sharp minima of the periodic textures. Part of this is due to the irregularity of the texture, and part is due to averaging the power spectra. Figure 52 shows the ssd surfaces for the four periodic textures using this averaging algorithm. We see that these surfaces are smoother than those from the original algorithm where we did not do any averaging of the power spectrum

texture	true (p, q)	true (σ, τ)	computed (p, q)	computed (σ, τ)	error
woven aluminum wire (D6)	(0.614, 0.364)	(35.5°, 30.7°)	(0.462, 0.256)	(27.8°, 29.0°)	7.7°
French canvas (D21)	(0.614, 0.364)	(35.5°, 30.7°)	(0.667, 0.359)	(37.1°, 28.3°)	2.1°
oriental straw cloth (D53)	(0.614, 0.364)	(35.5°, 30.7°)	(0.564, 0.359)	(33.8°, 32.5°)	2.0°
cotton canvas (D77)	(0.614, 0.364)	(35.5°, 30.7°)	(0.462, 0.256)	(27.8°, 29.0°)	7.7°
average error					4.9°
oriental straw cloth (D78)	(0.614, 0.364)	(35.5°, 30.7°)	(0.564, 0.256)	(31.8°, 24.4°)	5.1°
loose burlap (D104)	(0.614, 0.364)	(35.5°, 30.7°)	(0.564, 0.359)	(33.8°, 32.5°)	2.0°
handmade paper (D57)	(0.614, 0.364)	(35.5°, 30.7°)	(0.461, 0.154)	(25.9°, 18.5°)	11.4°
plastic bubbles (D111)	(0.614, 0.364)	(35.5°, 30.7°)	(0.359, 0.359)	(27.0°, 45.0°)	11.3°
average error					7.5°

Table 6: Results of averaged ssd patch matching on periodic textured plates in Figure 34 - Figure 37 and on irregularly textured plates in Figure 48 - Figure 51.

patches. In general, we conclude that computing surface normals from irregular textures is harder than for periodic textures, and that we must use much more of the image data to get good estimates.

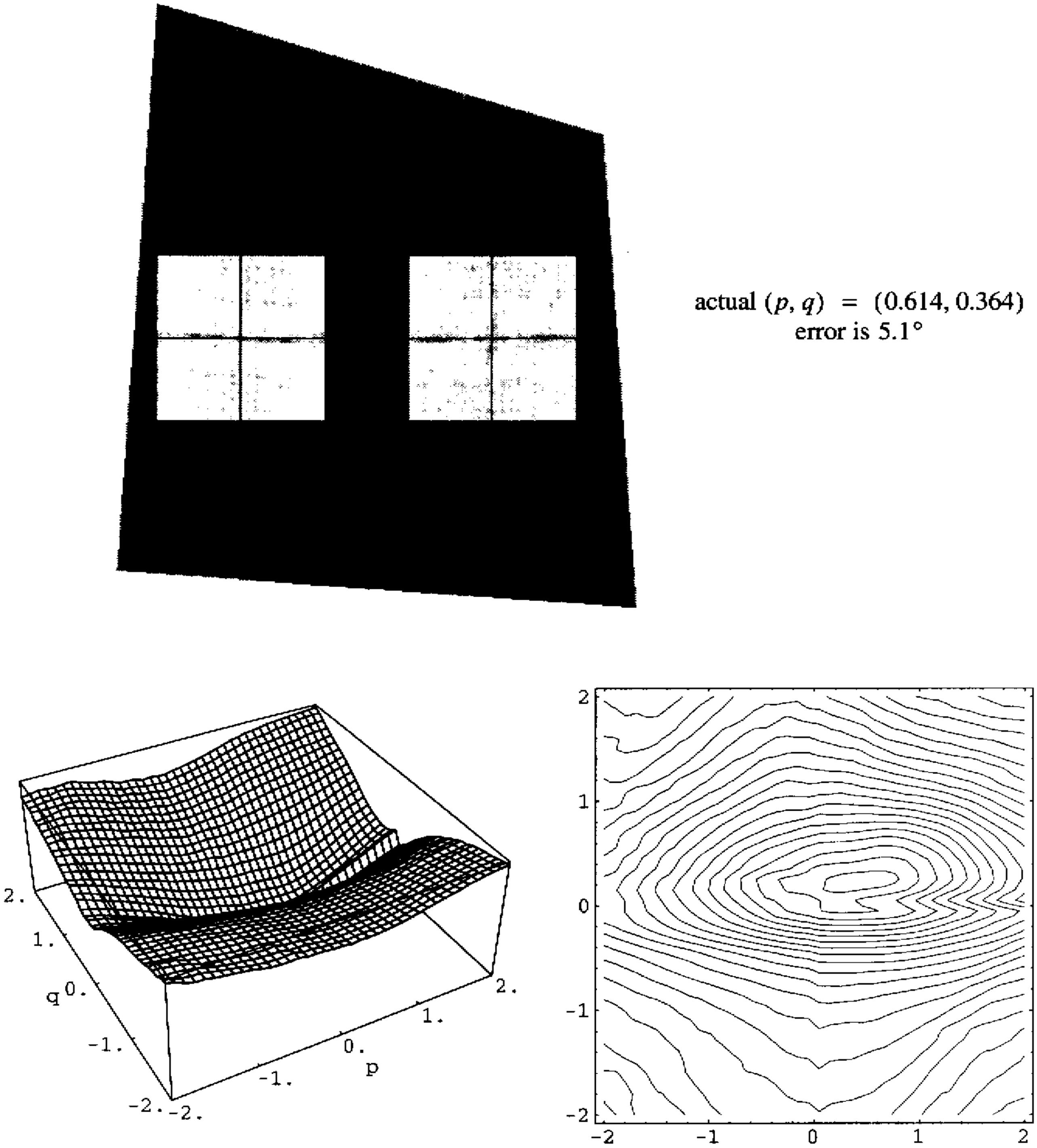
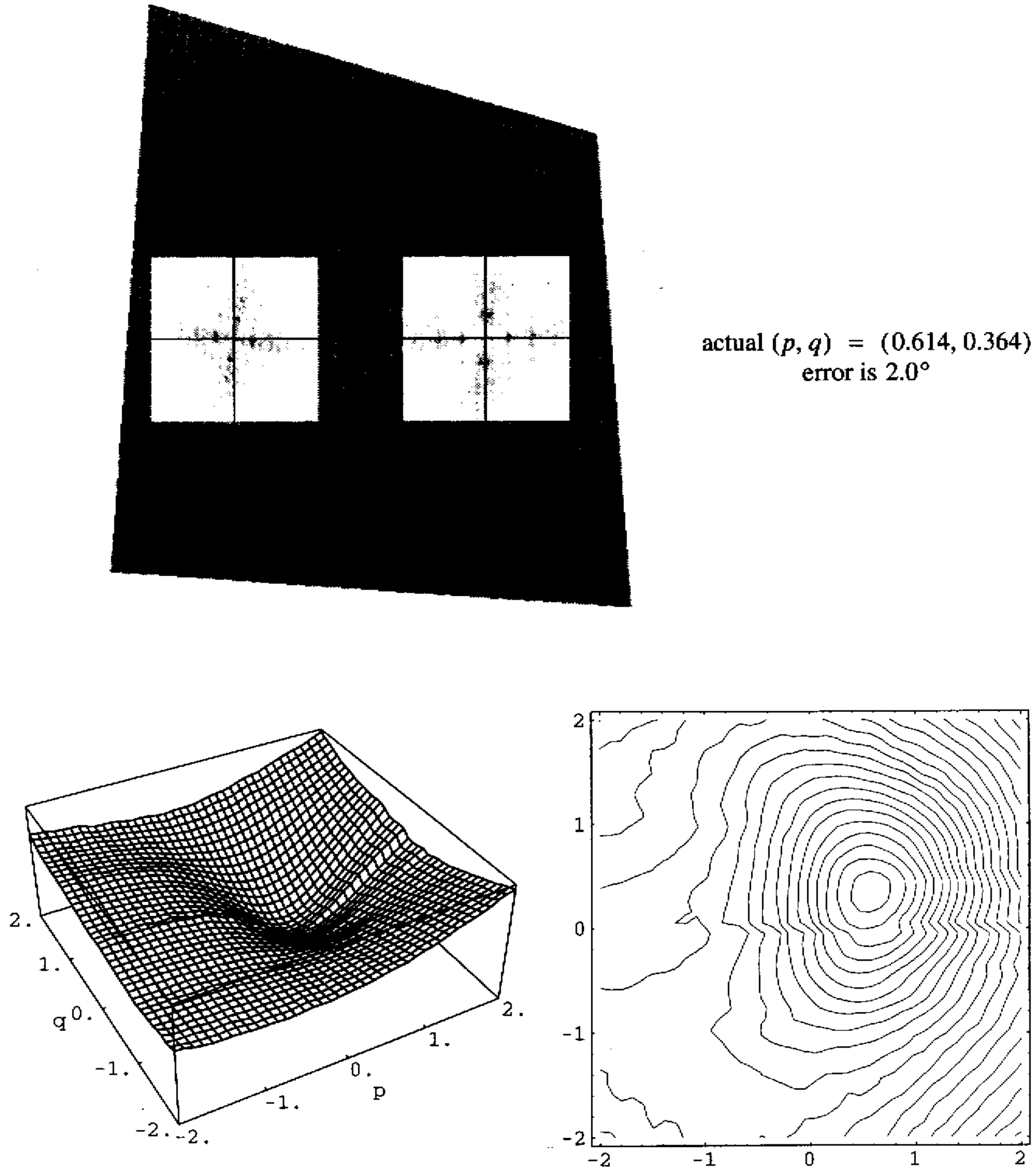
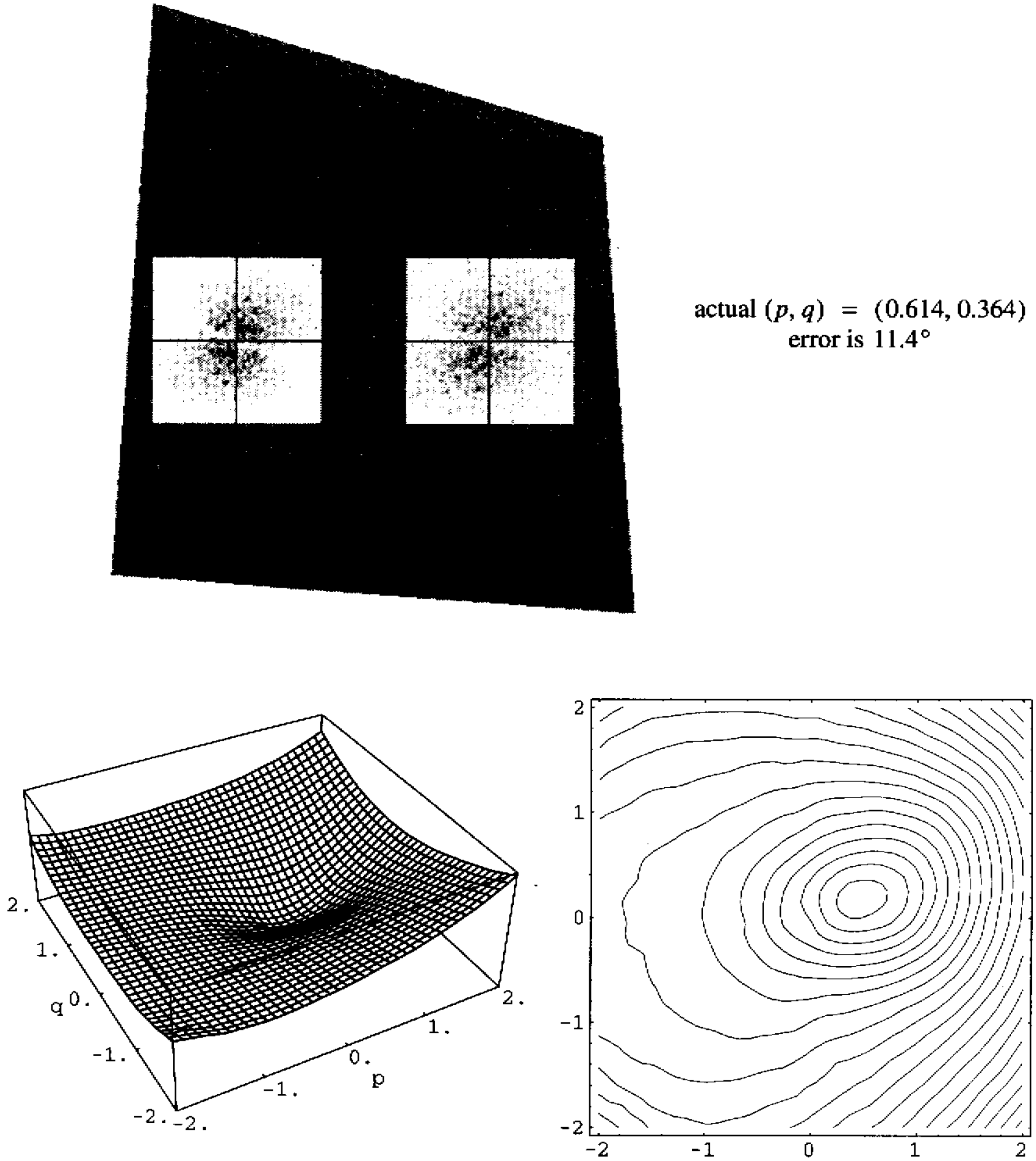


Figure 48: This Brodatz straw cloth (D78) is sort of periodic, but not exactly. The two 128x128 power spectrum patches show some peaks but also some other, less structured frequency content.



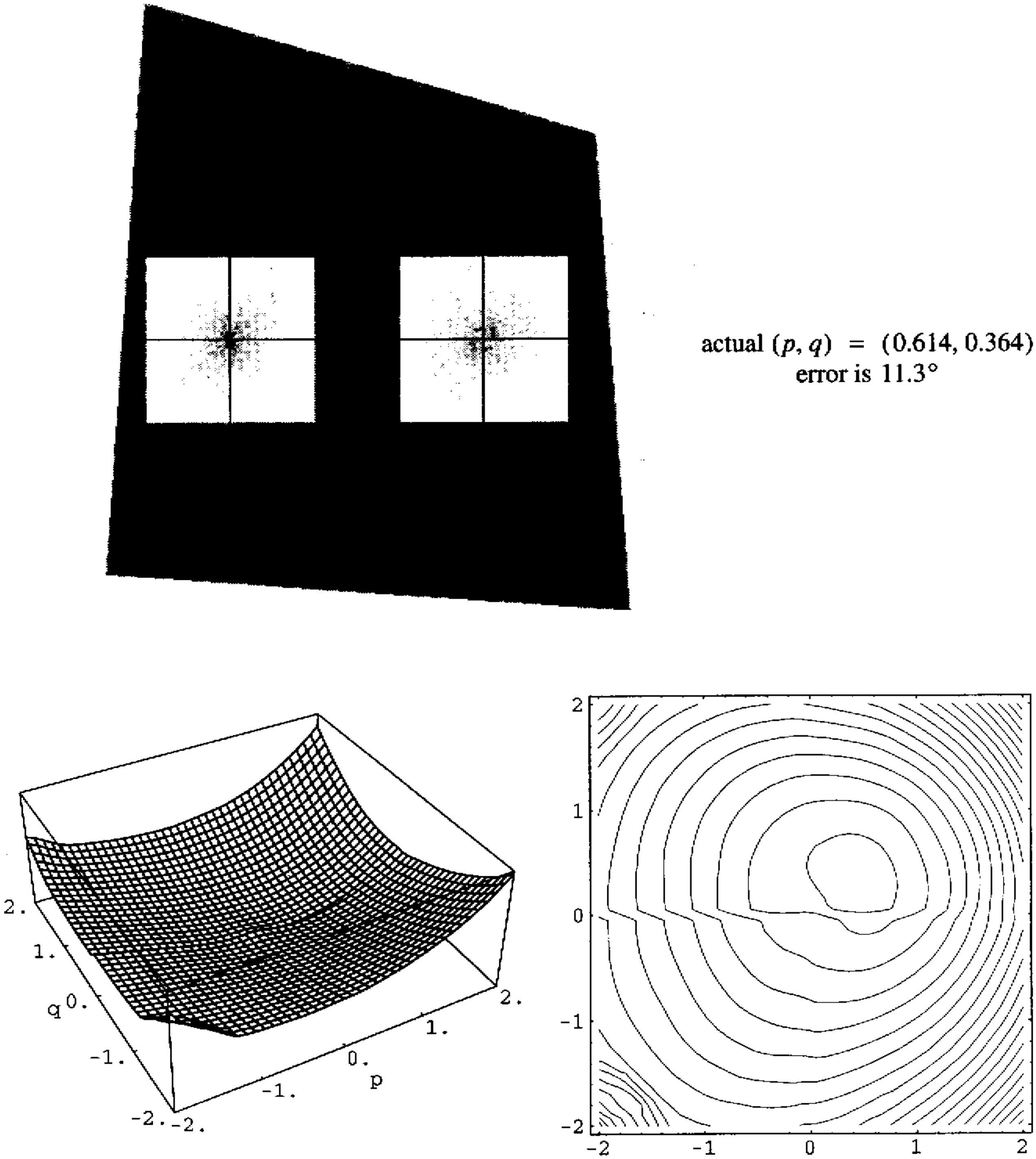
actual $(p, q) = (0.614, 0.364)$
error is 2.0°

Figure 49: This Brodatz loose burlap (D104) is a perturbed periodic pattern, and the averaging of the power spectrum patches tends to smooth out the perturbations.



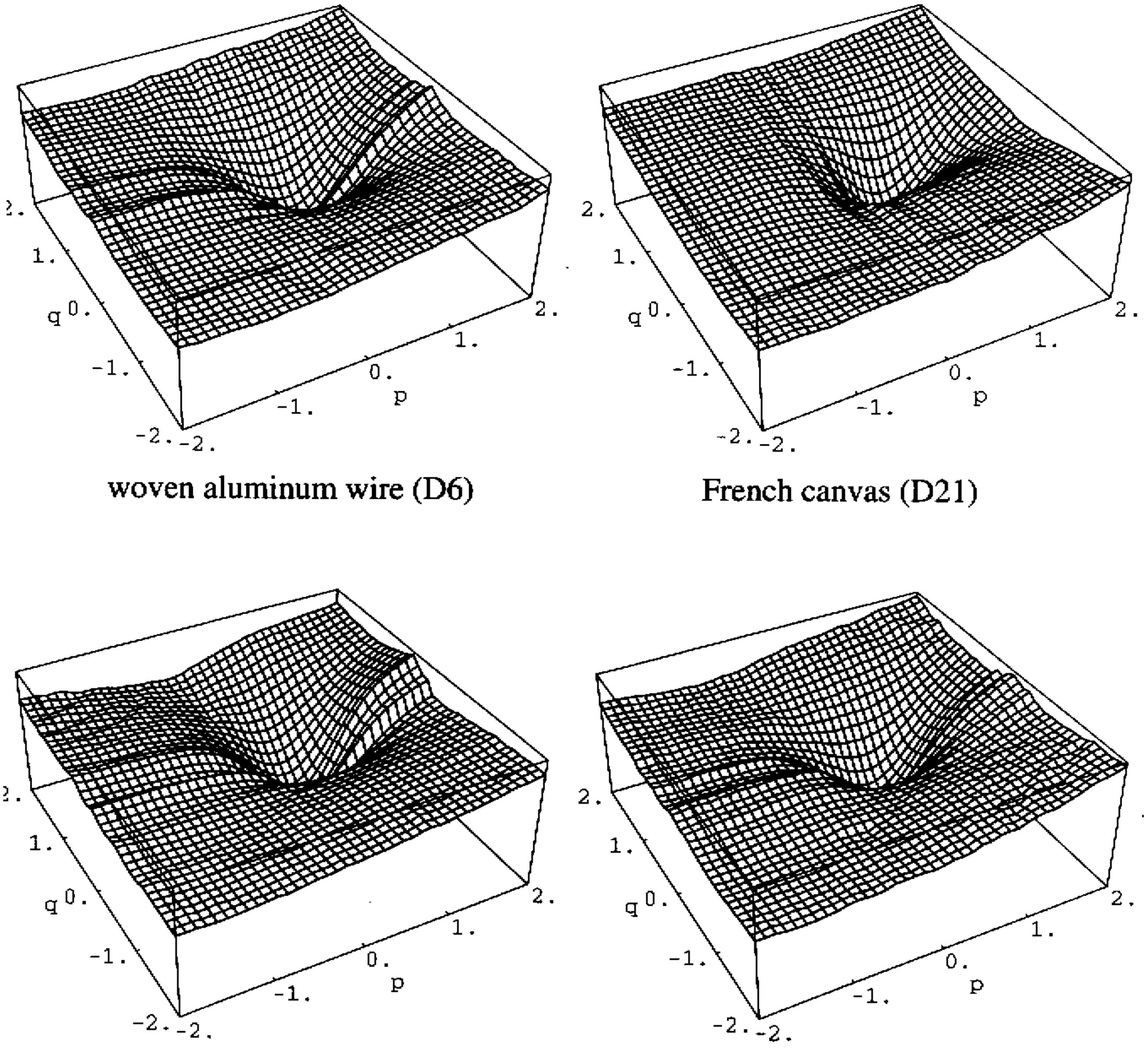
actual $(p, q) = (0.614, 0.364)$
error is 11.4°

Figure 50: This Brodatz handmade paper (D57) is fairly random, and the ssd surface is quite smooth, meaning it is more difficult to find a precise minimum.



actual $(p, q) = (0.614, 0.364)$
error is 11.3°

Figure 51: These Brodatz bubbles (D111) are fairly random, and the ssd surface is quite smooth.



woven aluminum wire (D6)

French canvas (D21)

oriental straw cloth (D53)

cotton canvas (D77)

Figure 52: These are the ssd surfaces for the four periodic textures in Figure 34 - Figure 37 when we use the averaged ssd patch-matching algorithm. These surfaces are smoother than with the original algorithm.

3.5.5 Bias and Variance

After working with a shape-from-texture program, it becomes apparent that the accuracy of the results depends on many factors. Some textures give consistently more accurate surface normals than others. The relative positions of the two power spectrum patches also makes a difference. If we can characterize these effects, we can make our algorithms more predictable and robust, and we can find good ways to set the values of the magic numbers.

The following sections give an experimental and analytical characterization of the accuracy of our ssd patch-matching algorithm. We do experiments on synthetic, sinusoidal textures with added noise. These experiments show how the bias and variance of the computed surface normal are affected by the underlying texture and by the relative positions of the two power spectrum patches. The simplicity of our ssd patch-matching algorithm lends itself to an analytical examination of the variance in the surface normal results by examining the Hessian of the ssd surface at the minimum. We show how our analytical variance predictions match the experimental results.

3.5.5.1 Experimental Bias and Variance

All our experiments were performed with simulated sinusoidal textures mapped onto a plate with a surface normal of $(p, q) = (0.257, 0.257)$. This was achieved by rotating the square, frontal plate by 20° around a line from the upper left to the lower right corner. One of the power spectrum patches was always placed at the center of the image. Some of the experimental images appear in Figure 53. In the first set of experiments we varied the parameters of the sinusoidal texture. In the second set of experiments, we varied the location of one of the power spectrum patches. For each configuration of texture and patches, we ran our ssd patch-matching algorithm 100 times. Before each run, we corrupted the synthetic image by adding zero-mean Gaussian noise with a standard deviation of ten. (The range of the gray scales is [0,255].) We added the noise to simulate real noise and to induce some variation in the results so we could get statistics on the errors. Instead of using an exhaustive search for the minimum ssd, we used a numerical, multi-variate function minimizer with a finite-difference gradient from IMSL ("dumimf"). Since we knew what the solution was, we could give the actual surface normal as the initial guess. Using this routine also relieved us from worrying about the speed and magic numbers of the exhaustive search used in the original algorithm.

The left column of Figure 53 shows some of the images used for the first part of the first experiment. These textures are single sinusoids at various angles θ_u. We let this angle vary from 0° to 170° in increments of 10° (Going beyond 180° would have produced the same set of images.) For each angle, we computed (p, q) 100 times, each time with different, random noise added to the image. The statistics on the results of this experiment are shown in Figure 54. In the upper left we show the average angle error plotted against θ_u. Below that are the errors in p and q. The minimum average error is 0.5° at $\theta_u = 20°$, and the maximum is 34.6° at occurs at $\theta_u = 100°$. From the plots for the errors in p and q, it is apparent that most of the error in angle is due to the error in q. When θ_u is near 100°, the lines on the slanted plate run almost horizontally through the two patches. The cause of the bias is difficult to guess. We note that the sign of the bias in q flips between $\theta_u = 90°$ and $\theta_u = 100°$, where lines go through horizontal and where the variance is greatest. Part of the bias is probably because the texture in the windows is changing due to perspective, while our algorithm is based on the assumption that it is locally stationary. More error comes from the truncated Taylor series approximation that we made in Equation (66) for the perspective projection. We note also that high bias is usually accompanied by high variance.

Another factor that we thought might contribute to the bias is that, even ignoring the effects of discetization and approximation, there is *no* affine transformation that will make one of the power spectra exactly match the other one. This is because of the window effect. We showed that the power spectrum patches are related by

$$\left|F_1\left(a_1u+b_1v, a_2u+b_2v\right)\right|^2 \approx \left|F_2\left(u, v\right)\right|^2 \tag{83}$$

But, we must multiply the patches in the image by a window before computing the power spectra. This multiplication in space corresponds to convolution in frequency. The affine connection that we test in the algorithm is then

$$\left|W\left(a_1u+b_1v, a_2u+b_2v\right) * F_1\left(a_1u+b_1v, a_2u+b_2v\right)\right|^2 \approx \left|W\left(u, v\right) * F_1\left(u, v\right)\right|^2 \tag{84}$$

This is quite different from the approximation in Equation (83). However, at least for the single-sinusoid texture we are discussing, we take this effect into account in our analytical examination of bias and variance in Section 3.5.5.3. This analysis predicts no bias, so the window effect is probably not the reason for the error.

The right column of Figure 54 shows the variance of the computed surface normals. For both p and q, the maximum variance occurs at $\theta_u = 90°$. As we mentioned above, at this angle the lines of the texture run almost horizontally through the two patches. Any change in surface normal in this configuration will cause only a small change in the texture in the two windows, so noise tends to dominate the solution. A change in p will be more apparent than a change in q, so the variance for q is higher. Figure 55 shows the ssd surfaces for $\theta_u = 20°$ (which gives a low variance) and for $\theta_u = 100°$ (which gives a high variance). It is clear that the sharper minimum gives a lower variance.

We ran the same set of experiments on the crossed-sinusoids texture shown in the right column of Figure 53. This texture consists of the sum of two sinusoids at a right angle. Figure 56 shows that the bias and variance are much smaller and much less sensitive to θ_u than those for the single-sinusoid. (All the bias and variance plots in this section have the same range on their vertical axes for comparison.) The extra frequency content goes a long way toward improving the results, because it helps the algorithm accurately localize the solution along both p and q. The variance for q is higher than the variance for p because the relative position of the windows makes the algorithm less sensitive to changed in q.

The second experiment examines the effect of the relative position of the two power spectrum patches on the computed surface normal. We picked one single-sinusoid image and one crossed-sinusoids image from the previous experiment. We chose the ones with $\theta_u = 50°$, because this did not give extreme values of bias and variance in the previous experiment. These images are shown in Figure 57. One of the power spectrum patches was always at the center of the image, and the other was 150 pixels away at an angle of θ_x measured from horizontal on the image. For the single-sinusoid texture, the variance is maximum when the lines of the texture are parallel to a line connecting the centers of the windows. The is the same phenomenon that we noticed in the first experiment. For the crossed-sinusoids texture, whose results are shown in Figure 59, the bias and variance are consistently lower.

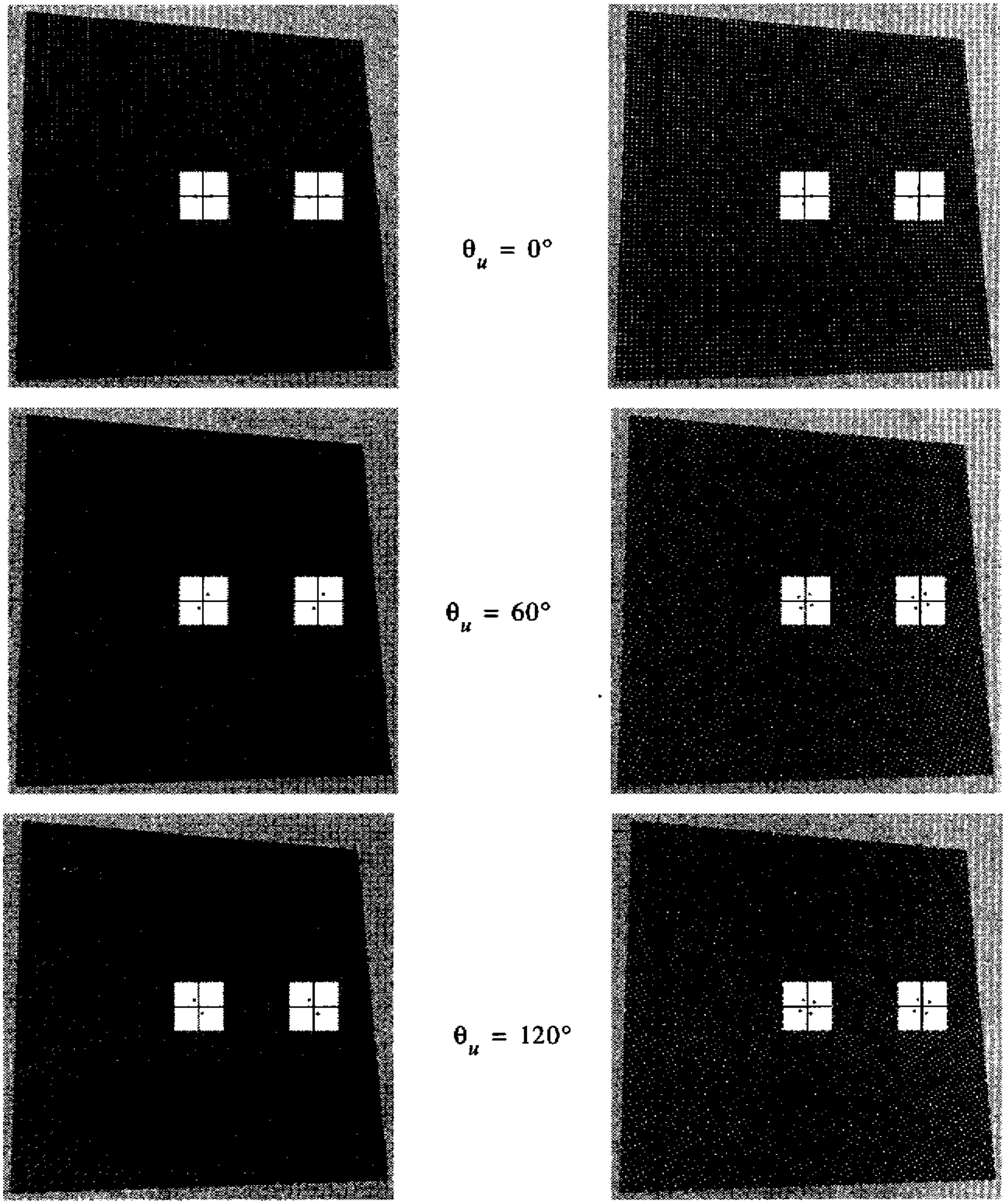


Figure 53: These are some of the images used to test the sensitivity of our ssd patch-matching algorithm to the underlying texture. The left column of images has a single sinusoid at various angles, and the right column has two sinusoids crossed at a right angle. We get more accurate results for the two-sinusoid texture.

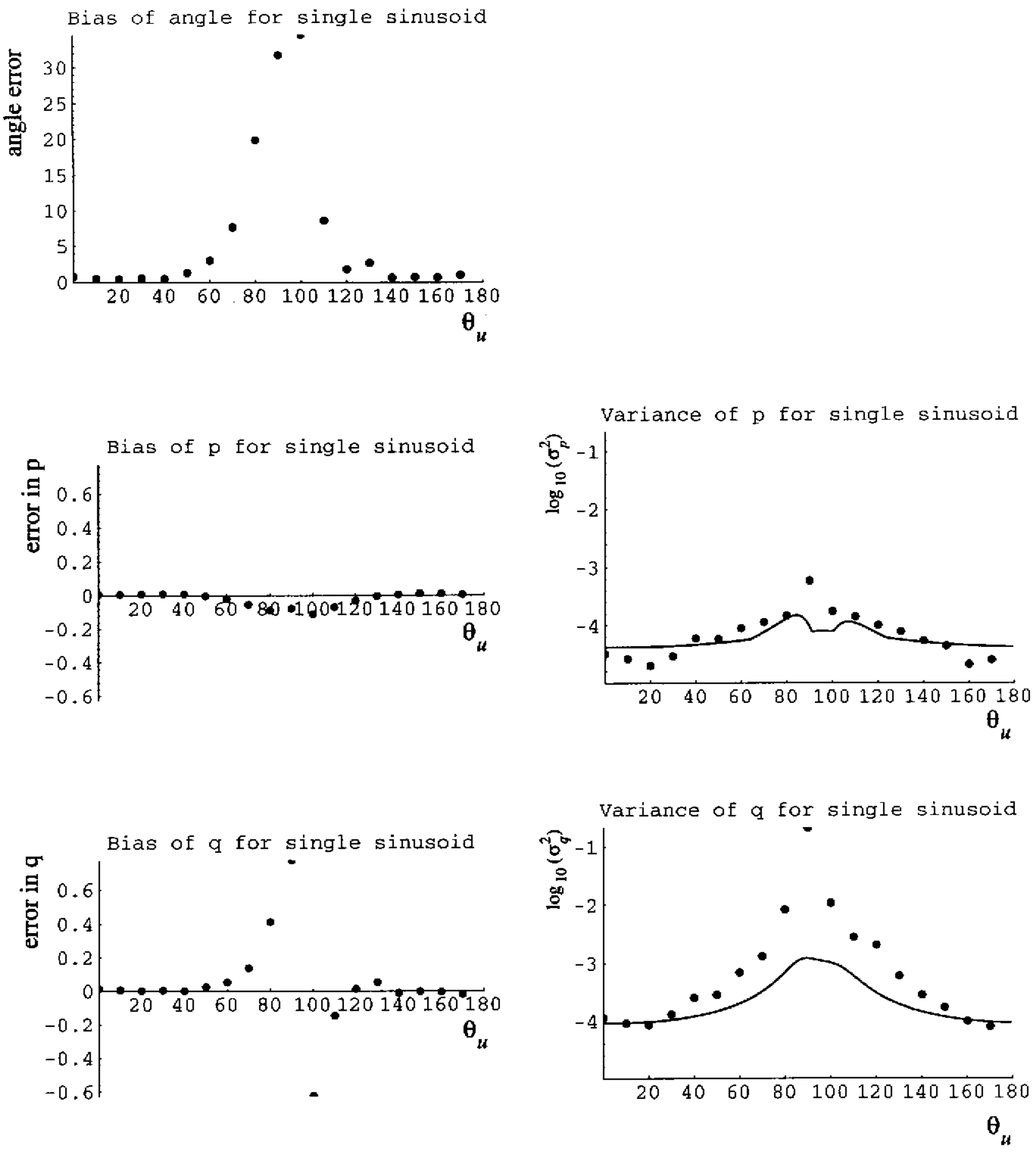
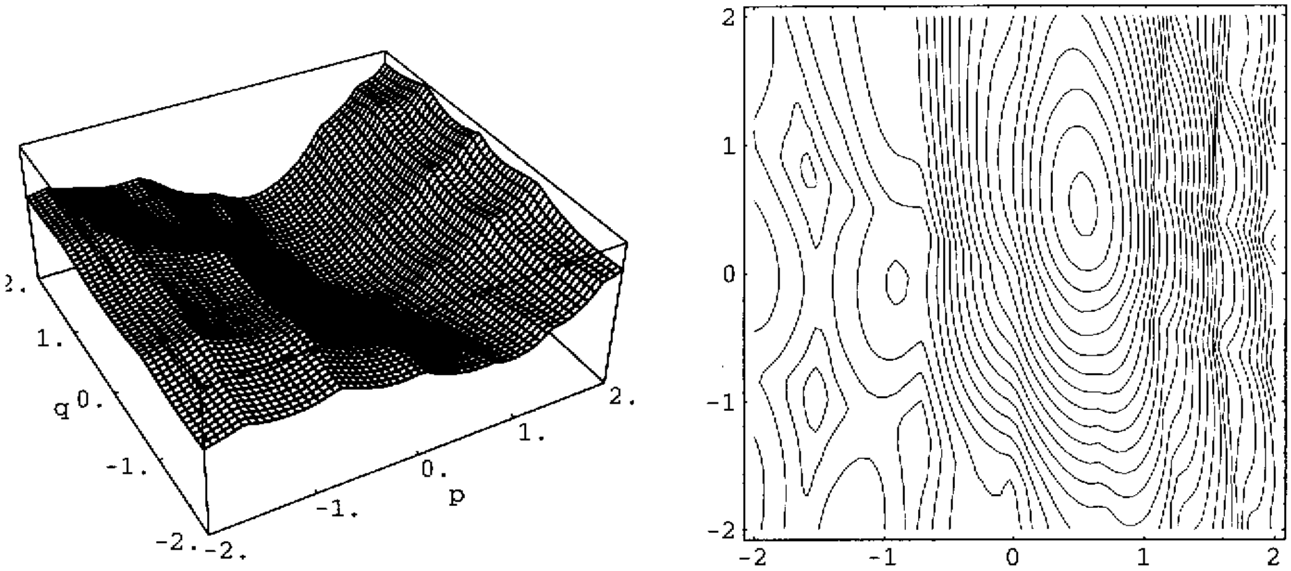
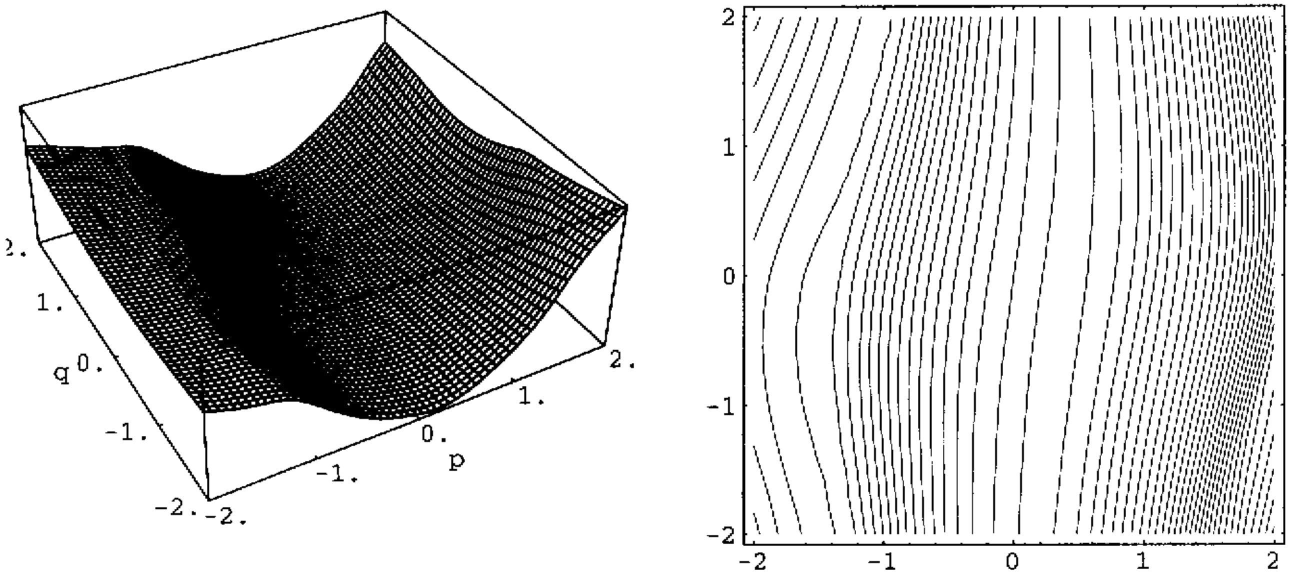


Figure 54: Bias and variance of surface normal estimates when angle of single sinusoidal texture is varied.



SSD surface for single-sinusoid texture when $\theta_u = 20°$



SSD surface for single-sinusoid texture when $\theta_u = 100°$

Figure 55: The ssd surfaces show why the variance in computed surface normals for one angle of sinusoids is greater than for another angle. A sharper minimum yields less variance.

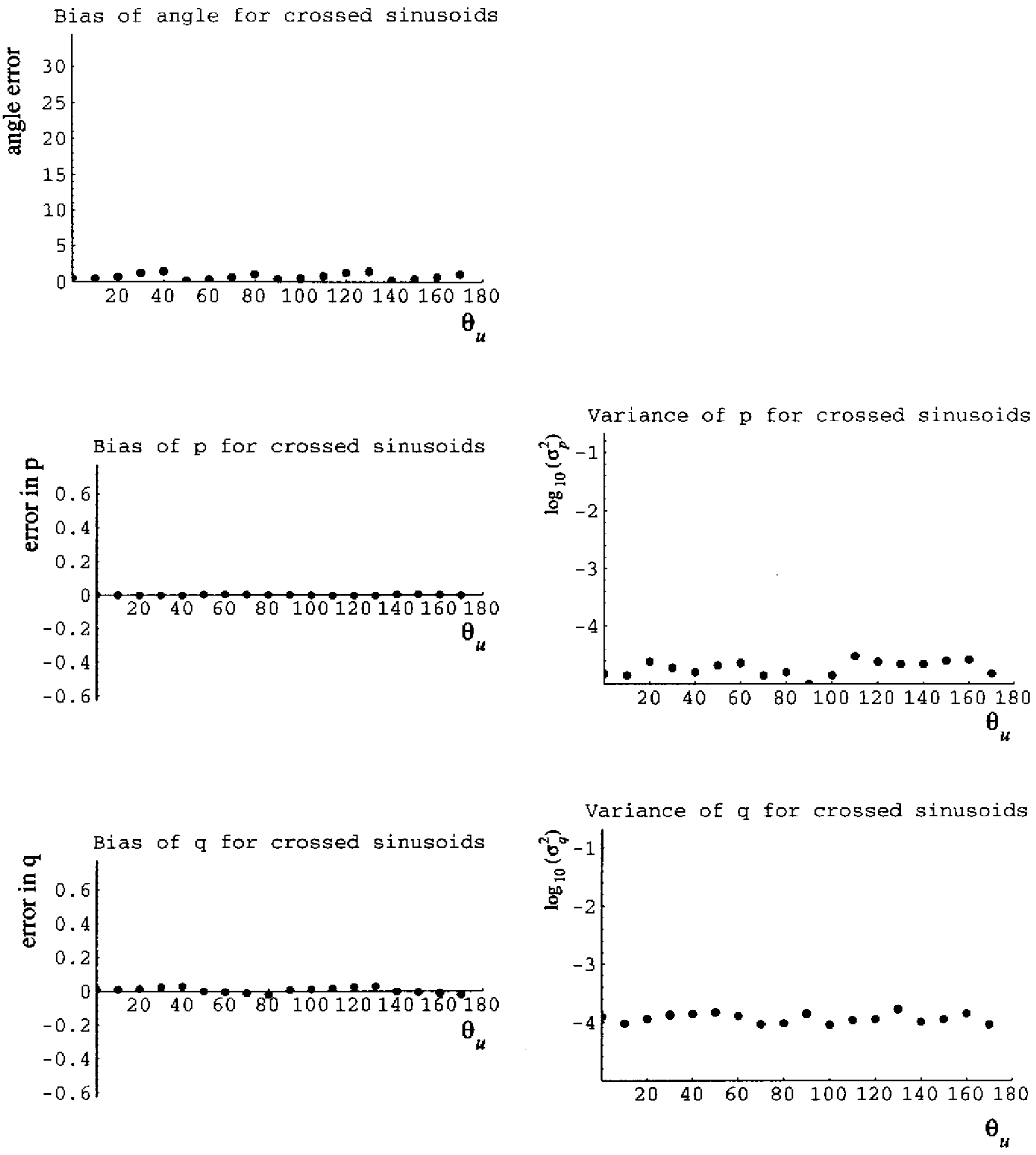


Figure 56: Bias and variance of surface normal estimates when angle of crossed sinusoidal texture is varied.

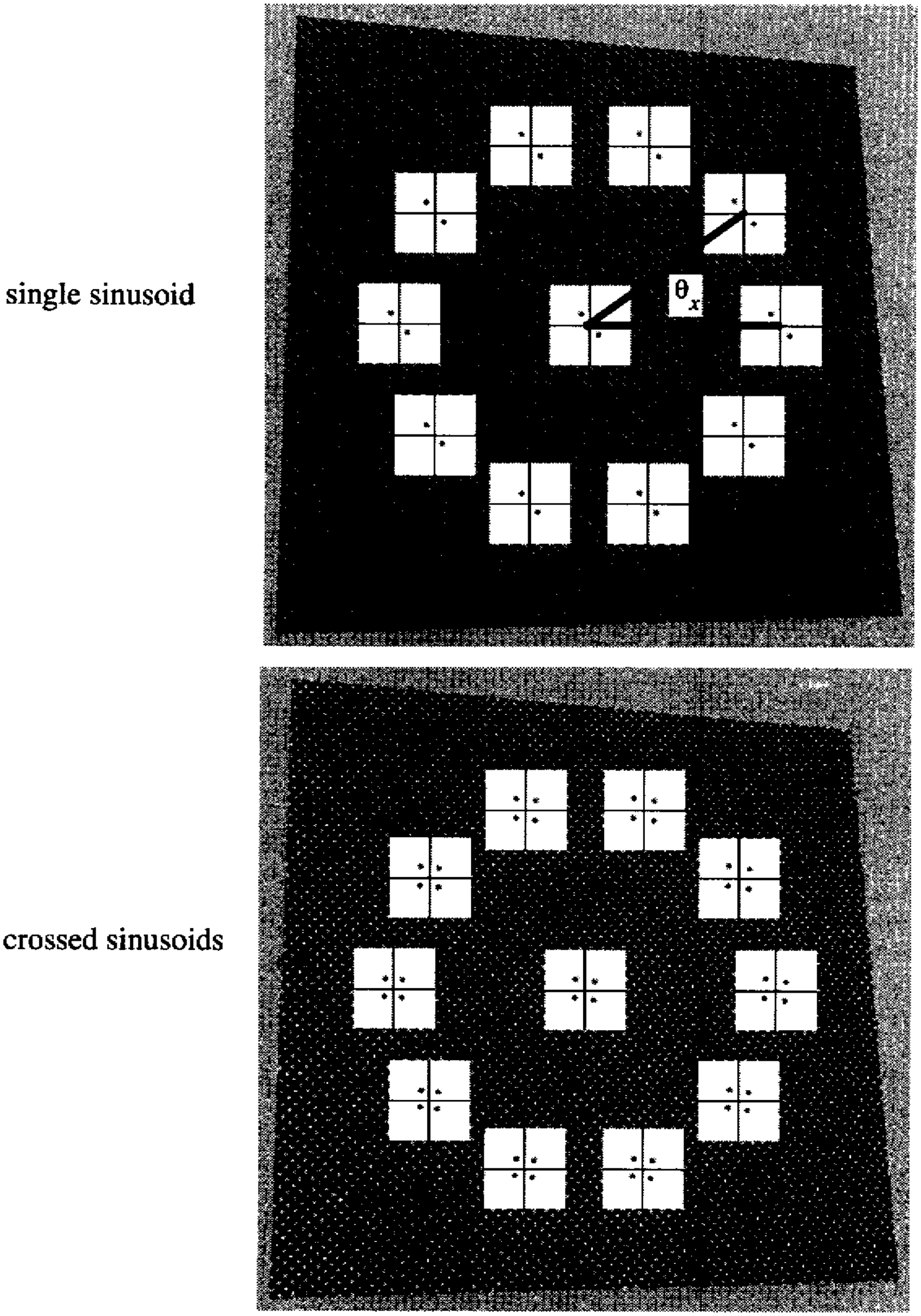


Figure 57: These are some of the power spectrum patches used to test the sensitivity of the ssd patch-matching algorithm to the relative orientation of the patches.

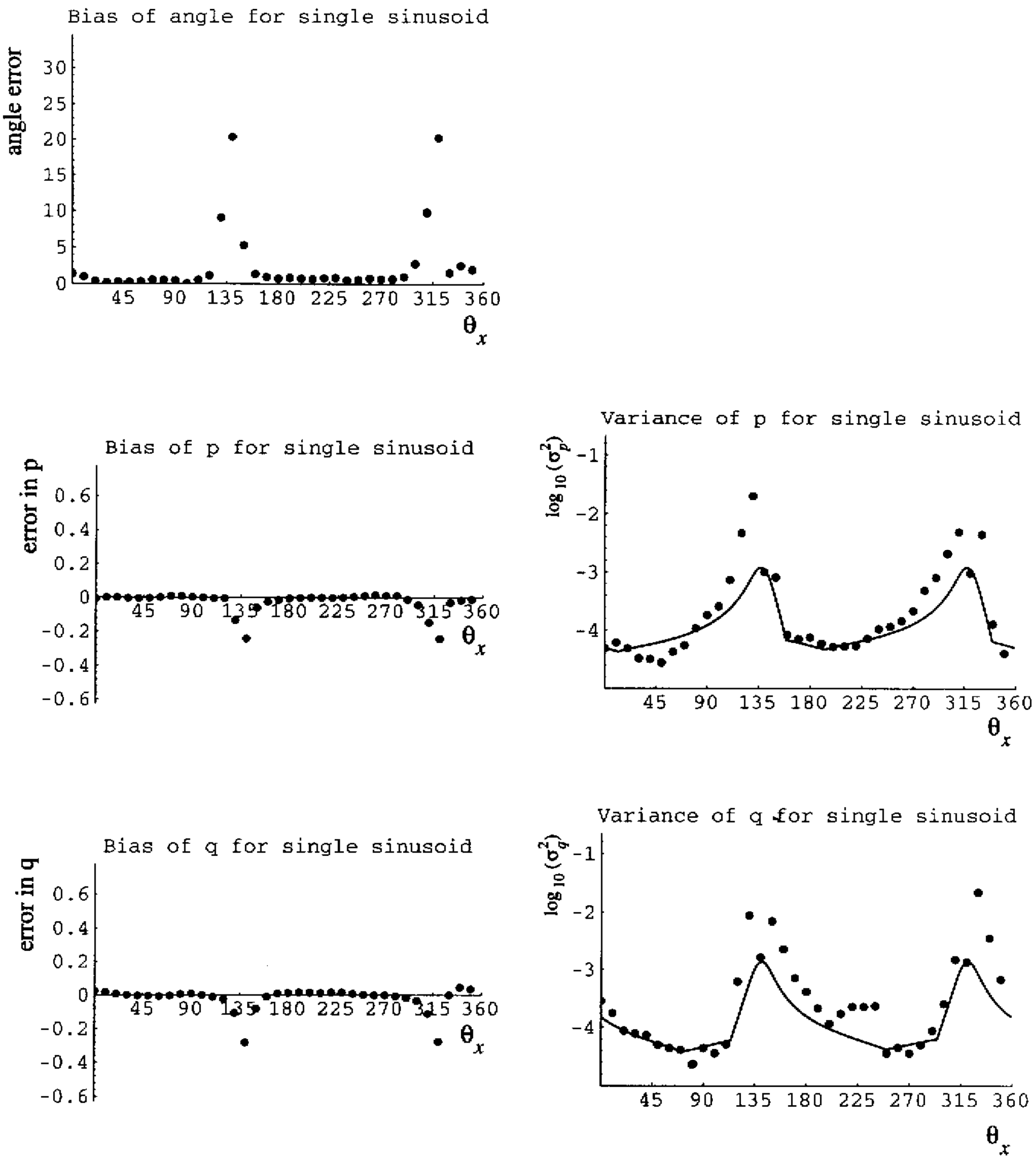


Figure 58: Bias and variance of surface normal estimates when angle between power spectrum patches is varied on a single sinusoid texture

Algorithm1: SSD Patch Matching

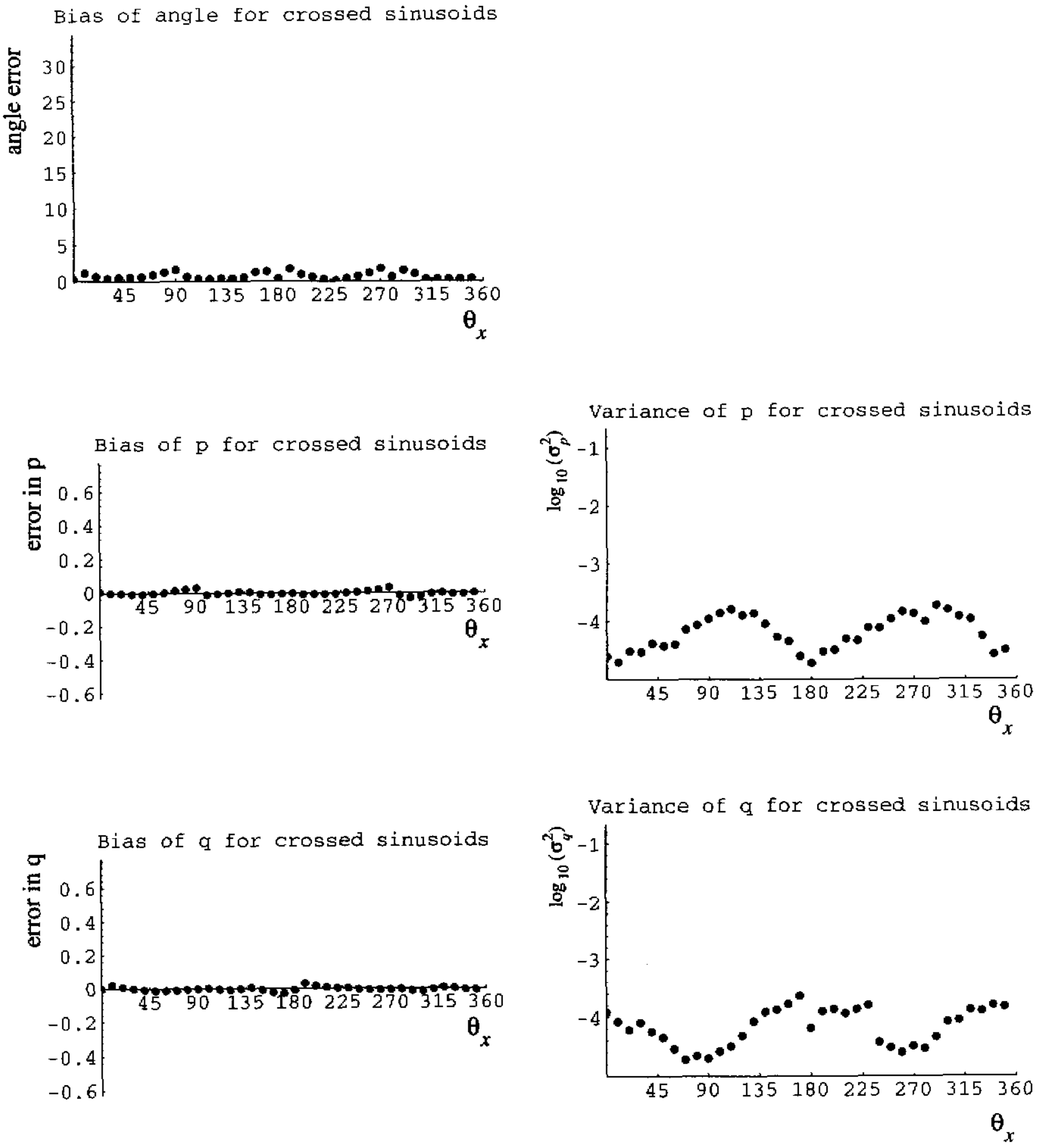


Figure 59: Bias and variance of surface normal estimates when angle between power spectrum patches is varied on a crossed sinusoids texture

3.5.5.2 Recipe for Analytical Variance Prediction

Numerical Recipes[86] (Section 14.5, "Confidence Limits on Estimated Model Parameters") gives the recipe for predicting the variance of the results of an algorithm that uses ssd. In this section we will quickly review the recipe for our particular problem. We serve the results in Section 3.5.5.4. The method depends on examining the shape of the depression around the minimum of the ssd surface. If this depression is very sharp, the variance in the solution will be small. If it is shallow along p or q, the variance will be larger for that variable. The shape of the depression is characterized by the eigenvalues and eigenvectors of the Hessian matrix at the minimum point.

The basis of the recipe is a χ^2 merit function given by

$$\chi^2(\vec{a}) = \sum_{i=1}^{N} \left[\frac{y_i - y(x_i;\vec{a})}{\sigma_i} \right] \qquad (85)$$

where $\vec{a}$ is the vector solution to find, (x_i, y_i) are the measurements, σ_i are the standard deviations of the measurements of the y_i, and $y(x;\vec{a})$ is the model to fit. If the errors in the measurements are normally distributed, the minimum of this function over $\vec{a}$ is the maximum likelihood solution.

In discrete terms, our ssd is given by

$$\mathrm{ssd}(p, q) = \sum_{j=0}^{l-1} \sum_{i=0}^{l-1} \{ S_1 [a_1(p, q) u + b_1(p, q) v, a_2(p, q) u + b_2(p, q) v] - S_2 [u, v] \}^2 \qquad (86)$$

where $S_1(u, v)$ and $S_2(u, v)$ are the two power spectrum patches, and $a_1(p, q)$, $b_1(p, q)$, $a_2(p, q)$, $b_2(p, q)$ are the affine parameters as functions of the surface normal given by Equations (76) and (77). In terms of $\chi^2(\vec{a})$, $\vec{a} = (p, q)$. The difference between $\mathrm{ssd}(p, q)$ and $\chi^2(p, q)$ is that the ssd has no ideal model; both terms in the ssd difference are measurements. Also, the differences in the ssd are not divided by σ_{ij}; we will assume $\sigma_{ij} = \sigma$. Despite these differences, there is still enough similarity to apply the recipe.

In order to predict the variance of the solution, we compute the Hessian of the error surface at the minimum of the error surface:

$$\alpha = \frac{1}{2} \begin{bmatrix} \frac{\partial^2}{\partial p^2} \chi^2(p^*, q^*) & \frac{\partial^2}{\partial p \partial q} \chi^2(p^*, q^*) \\ \frac{\partial^2}{\partial p \partial q} \chi^2(p^*, q^*) & \frac{\partial^2}{\partial q^2} \chi^2(p^*, q^*) \end{bmatrix} \qquad (87)$$

where (p^*, q^*) is the true solution.

If the measurement errors are normally distributed, then the covariance matrix of the solution will be the inverse of the Hessian. For our 2D case, this covariance matrix defines an ellipse in (p, q) that is approximately the same shape as the constant-ssd contours of the ssd surface around the minimum point. The directions of the axes of the ellipse are given by the eigenvectors of the inverse Hessian, and the lengths of the axes are given by the eigenvalues. The variances along p and q are given by the projection of the ellipse onto the p and q axes. Since our errors are not normally distributed, we can only hope that the analysis still holds. The non-normality means we can only approximate the covariance matrix up to a scale factor. This is still useful for predicting the relative variance of solutions for different situations.

3.5.5.3 Analytical Expression for the SSD Surface and Hessian

Computing the Hessian at the minimum of the ssd surface is conceptually simple, but complicated in practice. We will do it for the simple case of a single sinusoid texture, but we could do the same analysis for any texture. The frontal texture is

$$e^{j2\pi(u_0 s + v_0 t)} \tag{88}$$

whose Fourier transform is

$$\delta(u - u_0, v - v_0) \tag{89}$$

Even though this texture is complex and our images are real, this is still an accurate model of our algorithm for a real sinusoid. In our implementation, we only work with the upper half of the frequency plane, since it is symmetric about the u axis. A real sinusoid has a pair of delta functions as a Fourier transform. Unless they are both on the u axis, one will be above the u axis and one will be below. Therefore, the algorithm is really only using one delta function.

The frontal texture is projected to image patches $i = 1, 2$ with image coordinates (x_i, y_i) and scene coordinates Z_i. The image pattern is

$$f_i(x, y) = \exp\left[j2\pi\left(u_0\left(s_{x_i}x + s_{y_i}y\right) + v_0\left(t_{x_i}x + t_{y_i}y\right)\right)\right] \tag{90}$$

where (s_x, s_y, t_x, t_y) comes from Equations (67) and (68). From Gaskill[32] (p. 90), the Fourier transform of this patch is

$$F_i(u, v) = \delta(u - u_i, v - v_i) \tag{91}$$

where

$$\begin{aligned} u_i &= s_{x_i} u_o + t_{x_i} v_0 \\ v_i &= s_{y_i} u_o + t_{y_i} v_0 \end{aligned} \tag{92}$$

To make the analysis simpler, we will use a Gaussian window instead of the Blackman-Harris window that use in our algorithms. The Gaussian window is

$$w(x, y) = \frac{1}{l^2} \exp\left[-\frac{\pi}{l^2} (x^2 + y^2) \right] \tag{93}$$

The Fourier transform of this window is

$$W(u, v) = \exp\left[-\pi l^2 (u^2 + v^2) \right] \tag{94}$$

The Fourier power spectrum of a windowed patch is then

$$\begin{aligned} S_i(u, v) &= \left| W(u, v) * F_i(u, v) \right|^2 \\ &= \exp\left[-2\pi l^2 \left((u - u_i)^2 + (v - v_i)^2 \right) \right] \end{aligned} \tag{95}$$

In finding the surface normal, $S_1(u, v)$ will be affine-transformed and compared to $S_2(u, v)$. In Appendix 4 we argue that there should be no scale factor applied to either power spectrum before forming the ssd. That analysis was based on the fact that the window we use in our algorithm has finite extent. The Gaussian window that we use here goes on forever, so we will include scale factors to make the two power spectra comparable. We will choose the scale factors to make the power spectra have unit volume.

A useful integral for the following analysis is

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \exp\left[-c \left((a_1 u + b_1 v - u_0)^2 + (a_2 u + b_2 v - v_0)^2 \right) \right] du\ dv = \frac{\pi}{cD} \tag{96}$$

where $D = a_1 b_2 - a_2 b_1$.

The volume of the affine-transformed first patch is

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} S_1 (a_1 u + b_1 v, a_2 u + b_2 v)\, du\ dv = \frac{1}{2l^2 D} \tag{97}$$

The volume of the second patch is

$$\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} S_2 (u, v)\, du\ dv = \frac{1}{2l^2} \tag{98}$$

Our algorithm computes the ssd between the second power spectrum and an affine-transformed version of the first power spectrum. Normalizing both patches to unit volume gives the following for the ssd:

$$\begin{aligned} \mathrm{ssd}\,(p, q) &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \{2l^2 D S_1 (a_1 u + b_1 v, a_2 u + b_2 v) - 2l^2 S_2 (u, v)\}^2 du\ dv \\ &= \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \{2l^2 D \exp [-2\pi l^2 ((a_1 u + b_1 v - u_1)^2 + (a_2 u + b_2 v - v_1)^2)] - 2l^2 \exp [-2\pi l^2 ((u - u_2)^2 + (v - v_2)^2)]\}^2 du\ dv \\ &= t_1 + t_2 + t_3 \end{aligned} \tag{99}$$

where the dependence on (p, q) comes through the dependence of (a_1, b_1, a_2, b_2) on (p, q) given in Equations (76) and (77), and where

$$\begin{aligned} t_1 &= 4l^4 D^2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \exp [-4\pi l^2 ((a_1 u + b_1 v - u_2)^2 + (a_2 u + b_2 v - v_2))]\, du\ dv \\ t_2 &= -8l^4 D^2 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \exp [-2\pi l^2 ((a_1 u + b_1 v - u_1)^2 + (a_2 u + b_2 v - v_1))] \exp [-2\pi l^2 ((u - u_2)^2 + (v - v_2))]\, du\ dv \\ t_3 &= 4l^4 \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \exp [-4\pi l^2 ((u - u_1)^2 + (v - v_1))]\, du\ dv \end{aligned} \tag{100}$$

From Equation (96) we have

$$\begin{aligned} t_1 &= l^2 D \\ t_3 &= l^2 \end{aligned} \tag{101}$$

With some help from Macsyma, the expression for t_2 is

$$t_2 = \frac{-4l^2 D}{\sqrt{1+a_1^2+b_1^2+a_2^2+b_2^2+D^2}} \exp\left(\frac{-2\pi l^2 (\lambda_1 - 2(\lambda_2+\lambda_3))}{1+a_1^2+b_1^2+a_2^2+b_2^2+D^2}\right) \tag{102}$$

where

$$\begin{aligned}
\lambda_1 &= u_1^2(1+a_2^2+b_2^2+D^2)+v_1^2(1+a_1^2+b_1^2+D^2)+u_2^2(1+a_1^2+a_2^2+D^2)+v_2^2(1+b_1^2+b_2^2+D^2)\\
\lambda_2 &= u_1v_1(a_1a_2+b_1b_2)-u_2v_2(a_1b_1+a_2b_2)\\
\lambda_3 &= u_1u_2(a_1(1+b_2^2)-a_2b_1b_2)+u_1v_2(b_1(1+a_2^2)-a_1a_2b_2)+u_2v_1(a_2(1+b_1^2)-a_1b_1b_2)+v_1v_2(b_2(1+b_2^2)-a_1a_2b_1)
\end{aligned} \tag{103}$$

The equation for the ssd surface is long and complicated, even for the single-sinusoid texture. Its first and second partial derivatives are even longer and more complicated - so much so that we will not even write them out. We calculated the partial derivatives with Macsyma, converted this to input for Mathematica, and used Mathematica to give C computer code. When we evaluated the first partial derivatives at what should have been the minimum, they were always zero.

3.5.5.4 Results of Variance Prediction

In order to reproduce the experimental variance results, the variables in the partial derivatives were set to match those used in the experiments. We set $(p, q) = (0.257, 0.257)$. We set $(x_1, y_1) = (0, 0)$, because the first patch was always the center of the image. The depth at this point was $Z_1 = -9.958$. The two windows were always 150 pixels apart, which translated to $r_x = 0.1570$ in world units. The Gaussian window that best fit a 64x64 Blackman-Harris window has $l = 22.724$ pixels or $l = 0.02379$ in our simulated world units. The other parameters were allowed to vary depending on the experiment we were trying to simulate.

The predicted variances are shown as the solid lines in the variance plots in Figure 54 and Figure 58. We used a robust estimation scheme (least-median-of-squares[72]) to compute the best scale factor between the experimental and analytical variances. This method eliminates the effect of a few outliers. The predicted variances are fairly close to the actual variances. Even more important, the predicted variances reproduce the peaks and valleys of the actual variances.

In order to use the variance prediction in a real application, we need to know the power spectrum of the frontal texture. We could either compute this by "frontalizing" the texture based on the computed surface normal (Section 4.2.3), or we could just take a power spectrum patch as an approximation.

Being able to analytically predict the variance of our algorithm has practical benefits. The accuracy of our algorithm could probably be improved by altering the location of the power spectrum windows such that the solution will have minimum variance. The variance prediction could be used to find these locations. In fact, we could pick separate pairs of patches for computing the two components of the surface normal. Being able to predict variances, even to within a scale factor, might be useful for formulating the segmentation algorithm in Chapter 4 in terms of maximum likelihood.

3.6 Algorithm 2: Differential Patch Matching

This section describes an algorithm for computing shape-from-texture using pairs of power spectrum patches that are close together in the image. By using the differential method of Lucas and Kanade[65], we can quickly compute the affine transformation between two nearby patches. This has several advantages. If the scene has several textures, using nearby patches means the two patches will more likely be on the same textured surface than if they were farther apart. If the textured surface is curved, the "same plane" assumption will be close to true for nearby patches, and we can compute an approximate local surface normal. For large planar regions of texture, we can cover the whole region with patches and find a single surface normal using all the data. This would be prohibitively time consuming using Algorithm 1, but the Lucas and Kanade method lets us quickly compute the affine transformations between all adjacent patches. These are used in turn to find the surface normal. Since we developed this algorithm, a similar algorithm was reported by Malik and Rosenholtz[68].

3.6.1 Computing Affine Parameters Between Neighboring Patches

We showed in Section 3.4 that there is an affine transformation that relates two local power spectra taken from different parts of the image of the same textured surface, *i.e.* $S(x_1, y_1, a_1 u + b_1 v, a_2 u + b_2 v) \approx S(x_2, y_2, u, v)$. If we can find the affine parameters, we can find the surface normal using Equations (76) and (77), which relate (a_1, b_1, a_2, b_2) to (p, q).

Lucas and Kanade[65] sketched a method for finding the affine transformation between two almost-equal image patches. Our goal is to find a closed form that lets us solve for the affine parameters based on the two power spectrum patches. We begin by writing the affine-transformed version of $S(x_1, y_1, u, v)$ in terms of $(\Delta u, \Delta v)$, letting $S_1(u, v) = S(x_1, y_1, u, v)$ and $S_2(u, v) = S(x_2, y_2, u, v)$:

$$S_1(a_1 u + b_1 v, a_2 u + b_2 v) = S_1(u + \Delta u, v + \Delta v) \tag{104}$$

with $\Delta u = (a_1 - 1) u + b_1 v$ and $\Delta v = a_2 u + (b_2 - 1) v$. We can approximate $S_1(u + \Delta u, v + \Delta v)$ with a truncated Taylor series.

$$S_1(u+\Delta u, v+\Delta v) \approx S_1(u,v) + \Delta u \frac{\partial S_1}{\partial u} + \Delta v \frac{\partial S_1}{\partial v} \approx S_2(u,v) \qquad (105)$$

This approximation only holds for small Δu and Δv. If we substitute for Δu and Δv, we have

$$S_1(u,v) + [(a_1-1)u + b_1 v]\frac{\partial S_1}{\partial u} + [a_2 u + (b_2-1)v]\frac{\partial S_1}{\partial v} \approx S_2(u,v) \qquad (106)$$

Our goal is to find the affine parameters that minimize the difference between these two functions. The L_2 norm (sum of squared differences) is

$$e = \iint [S_1(a_1 u + b_1 v, a_2 u + b_2 v) - S_2(u,v)]^2 du\ dv \qquad (107)$$

Using the Taylor series approximation for S_1 gives

$$e \approx \iint \left[S_1(u,v) + [(a_1-1)u + b_1 v]\frac{\partial S_1}{\partial u} + [a_2 u + (b_2-1)v]\frac{\partial S_1}{\partial v} - F_2(u,v) \right]^2 du\ dv \qquad (108)$$

We minimize the norm by setting to zero its partial derivatives with respect to the affine parameters. That is, we have

$$\left[\frac{\partial e}{\partial a_1}\ \frac{\partial e}{\partial b_1}\ \frac{\partial e}{\partial a_2}\ \frac{\partial e}{\partial b_2} \right]^T = [0\ 0\ 0\ 0]^T \qquad (109)$$

This leads to a matrix equation consisting of four equations with four unknowns:

$$A\vec{a} = \vec{b} \qquad (110)$$

with

$$A = \begin{bmatrix} \iint u^2 (\frac{\partial S_1}{\partial u})^2 du\ dv & \iint uv (\frac{\partial S_1}{\partial v})^2 dx\ dy & \iint u^2 \frac{\partial S_1}{\partial u}\frac{\partial S_1}{\partial v} du\ dv & \iint uv \frac{\partial S_1}{\partial u}\frac{\partial S_1}{\partial v} du\ dv \\ \iint uv (\frac{\partial S_1}{\partial u})^2 du\ dv & \iint v^2 (\frac{\partial S_1}{\partial u})^2 du\ dv & \iint uv \frac{\partial S_1}{\partial u}\frac{\partial S_1}{\partial v} du\ dv & \iint v^2 \frac{\partial S_1}{\partial u}\frac{\partial S_1}{\partial v} du\ dv \\ \iint u^2 \frac{\partial S_1}{\partial u}\frac{\partial S_1}{\partial v} du\ dv & \iint uv \frac{\partial S_1}{\partial u}\frac{\partial S_1}{\partial v} du\ dv & \iint u^2 (\frac{\partial S_1}{\partial u})^2 du\ dv & \iint uv (\frac{\partial S_1}{\partial v})^2 du\ dv \\ \iint uv \frac{\partial S_1}{\partial u}\frac{\partial S_1}{\partial v} du\ dv & \iint v^2 \frac{\partial S_1}{\partial u}\frac{\partial S_1}{\partial v} du\ dv & \iint uv (\frac{\partial S_1}{\partial v})^2 du\ dv & \iint v^2 (\frac{\partial S_1}{\partial v})^2 du\ dv \end{bmatrix}$$

$$\vec{a} = (a_1, b_1, a_2, b_2)^T$$

$$\vec{b} = -\begin{bmatrix} \iint u\frac{\partial S_1}{\partial u}(S_1(u, v) - S_2(u, v) - u\frac{\partial S_1}{\partial u} - v\frac{\partial S_1}{\partial v})du\ dv \\ \iint v\frac{\partial S_1}{\partial u}(S_1(u, v) - S_2(u, v) - u\frac{\partial S_1}{\partial u} - v\frac{\partial S_1}{\partial v})du\ dv \\ \iint u\frac{\partial S_1}{\partial v}(S_1(u, v) - S_2(u, v) - u\frac{\partial S_1}{\partial u} - v\frac{\partial S_1}{\partial v})du\ dv \\ \iint v\frac{\partial S_1}{\partial v}(S_1(u, v) - S_2(u, v) - u\frac{\partial S_1}{\partial u} - v\frac{\partial S_1}{\partial v})du\ dv \end{bmatrix} \qquad (111)$$

Solving Equation (110) for $\vec{a}$ gives the affine parameters. Note that A is symmetric.

For our problem, we approximate the integrals with simple sums and the partial derivatives with two-point differences. Once we compute the affine parameters between two patches, we find (p, q) using Equations (76) and (77), which give the affine parameters in terms of the surface normal. Since these equations cannot be inverted to give a closed form solution for the surface normal in terms of the affine parameters, we resort again to an exhaustive search in (p, q) with the same range and resolution as for Algorithm 1. For each candidate surface normal, we compute the corresponding affine parameters. We take the surface normal whose affine parameters are closest, in the least squares sense, to the affine parameters computed from the two patches. As in Algorithm 1, we used a window size of 64 pixels. We spaced the windows 15 pixels apart from center to center, so there was significant overlap. We show the magic numbers for this algorithm in Table 7.

3.6.2 Results

Our first test is for computing local surface normals. We show two test images and the resulting surface normals in Figure 60. Each of these images was produced with a computer graphics program that mapped Brodatz textures onto shapes. We computed the affine parameters between every patch

description	value(s)
window center spacing	15 pixels horizontally and vertically
window size	64x64 pixels
search grid resolution	60x60
search grid limits	$(\lvert p\rvert, \lvert q\rvert) \le (2, 2)$
(p, q) smoothing (local only)	$0.5\,(p_{\mathrm{left}} + p_{\mathrm{right}}, q_{\mathrm{above}} + q_{\mathrm{below}})$

Table 7: Magic numbers used in differential patch-matching algorithm

and its immediate neighbors to the right and below. This gave us surface normals based on pairs of horizontally or vertically adjacent patches. To approximate the surface normals at the patch centers (and to smooth the results), we took p as the average of the two p's to the left and right and we took q as the average of the two q's above and below. We did the averaging this way because horizontally adjacent patches are not very sensitive to q and vertically adjacent patches are not very sensitive to p.

We can get moderate accuracy from this method. Figure 61 shows the ideal needle diagrams for both test images. This data was used to assess the accuracy of the algorithm by computing the angle between the actual and computed surface normals. These errors are also shown in Figure 61. The average error for the cylinder is 22.1°. Much of this error comes from the center of the image where the fundamental frequency of the texture is low. Because the texture is nearly frontal here, the affine transformations are close to the identity transform. This, combined with the low frequencies, means the frequencies are shifted by only a small amount, so random fluctuations prevail. There is very little work in curved shape-from-texture, and we know of no other results with ground truth. For the image with three textured plates, the average error is 30.3°, due mostly to patches that overlap two textures or one texture and the background. Taking a representative sample of the interior, non-overlapping windows gives an average error of 11.2°.

For our second test, we used the ten test images from Figure 34 - Figure 43. We get one set of affine parameters for each adjacent pair of patches. We searched to find the single surface normal that was most consistent in the least squares sense with all these affine parameters. Table 8 summarizes the results. The average error is 5.1°, which is comparable with the average error of 3.5° for the SSD patch-matching algorithm.

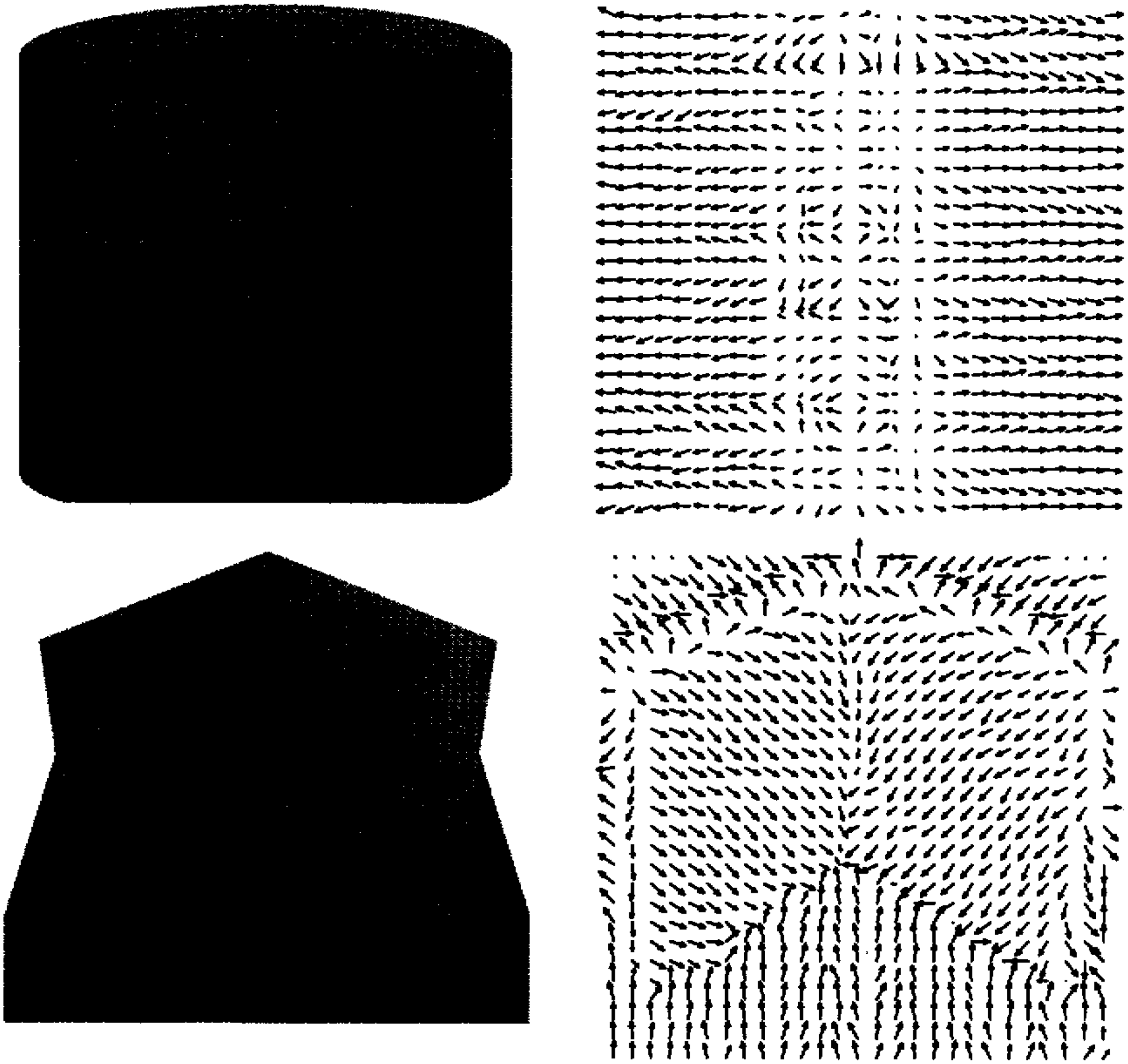


Figure 60: Tests images and needle diagrams of surface normals from differential patch matching. Cylinder has Brodatz woven aluminum wire (D6), and plates have woven aluminum wire (D6), netting (D34), and cotton canvas (D77).

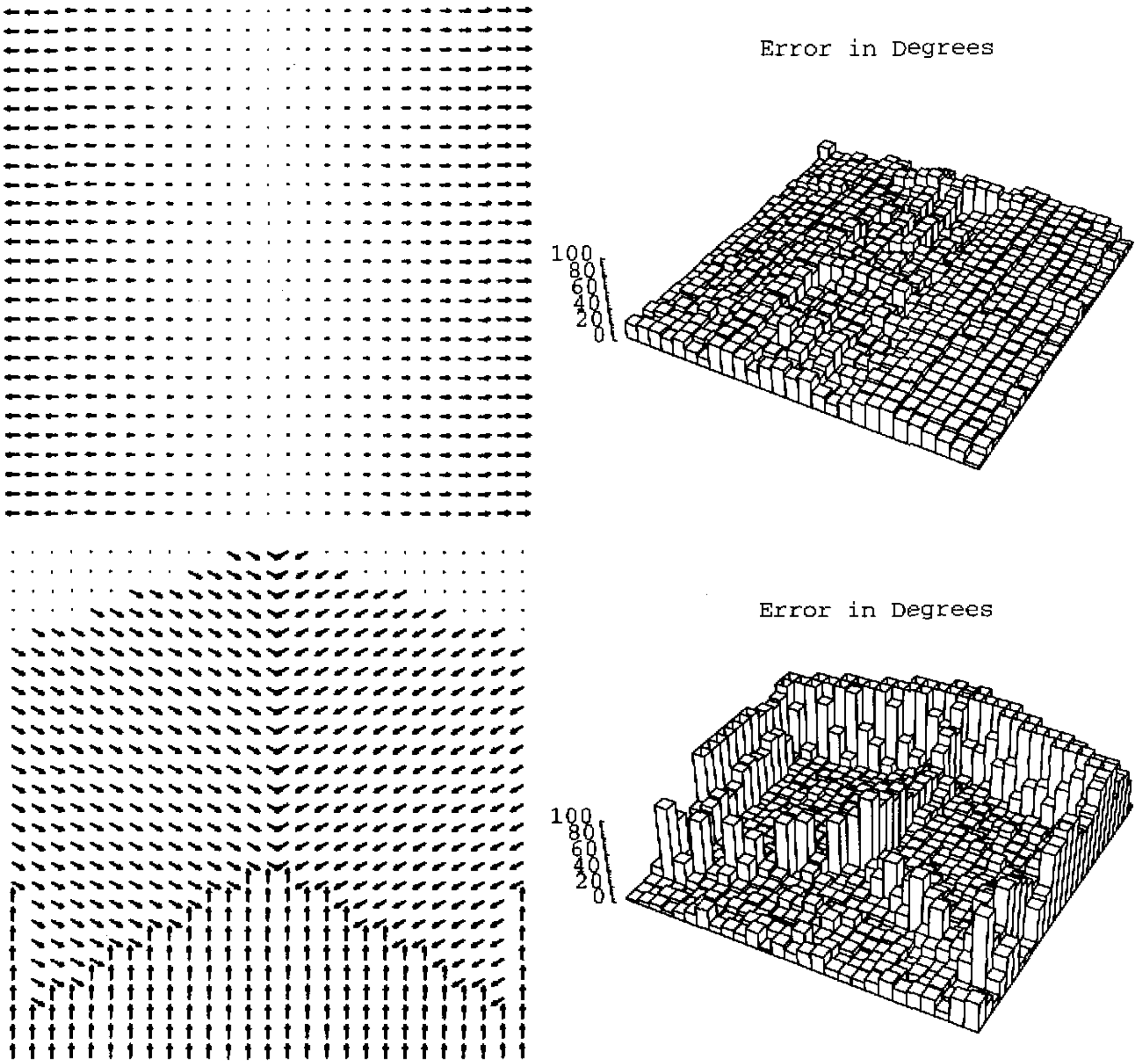


Figure 61: Ideal surface normals and error in degrees of surface normals shown in Figure 60.

texture	true (p, q)	true (σ, τ)	computed (p, q)	computed (σ, τ)	error
woven aluminum wire (D6)	(0.614, 0.364)	(35.5°, 30.7°)	(0.576, 0.237)	(31.9°, 22.4°)	5.8°
French canvas (D21)	(0.614, 0.364)	(35.5°, 30.7°)	(0.712, 0.441)	(39.9°, 31.8°)	4.5°
oriental straw cloth (D53)	(0.614, 0.364)	(35.5°, 30.7°)	(0.576, 0.305)	(33.1°, 27.9°)	2.9°
cotton canvas (D77)	(0.614, 0.364)	(35.5°, 30.7°)	(0.847, 0.441)	(43.7°, 27.5°)	8.4°
screen guard	(0.714, 0.0)	(35.5°, 0.0°)	(0.712, 0.102)	(35.7°, 8.1°)	4.7°
grid paper	(−1.878, 0.0)	(62.0°, 180.0°)	(−1.932, 0.034)	(62.6°, 179.0°)	1.1°
checkered paper	(−1.878, 0.0)	(62.0°, 180.0°)	(−2.000, 0.169)	(63.5°, 175.2°)	4.6°
egg crate	(−1.373, 0.0)	(53.9°, 180.0°)	(−0.983, −0.034	(44.5°, 182.0°)	9.5°
bumpy plastic	(−1.373, 0.0)	(53.9°, 180.0°)	(−1.729, 0.034)	(60.0°, 178.9°)	6.1°
doormat	(−1.373, 0.0)	(53.9°, 180.0°)	(−1.525, 0.034)	(56.8°, 178.7°)	3.0°
average error					5.1°

Table 8: Results of differential patch matching on textured plates in Figure 34 - Figure 43

3.7 Algorithm 3: Peak Matching

This section presents a third shape-from-texture algorithm based on the explicit assumption that the texture is periodic. We show in Appendix 3 that the spectrogram of a periodic function is a set of sharp peaks. This is demonstrated in Figure 62. The top texture, Brodatz D29 beach sand, is random, and its power spectrum is a blob centered at the origin. The bottom texture, Brodatz D77 cotton canvas, is periodic, and its power spectrum is a group of sharp peaks. In this section we show how these peaks move due to 3D perspective effects. We use this theory in an algorithm that tracks the location of the peaks from point to point in the image. This algorithm is more efficient and about as accurate as the two algorithms in the previous sections that consider the whole power spectrum.

3.7.1 Affine Connection

We assume like before that the periodic texture is mapped to a planar patch in the scene. If we show how the projection affects a single, sinusoidal texture pattern, we can easily see what happens to periodic textures, because they are just summed sinusoids (according to the Fourier series). Suppose the brightness pattern on the textured surface is given by $\cos(2\pi(u_0 s + v_0 t))$, then the corresponding projected textures from two different points on this surface would be given by

$$\begin{aligned} &\cos\left(2\pi\left(\left(s_{x_1}x' + s_{y_1}y'\right)u_0 + \left(t_{x_1}x' + t_{y_1}y'\right)v_0\right)\right) \\ &\cos\left(2\pi\left(\left(s_{x_2}x' + s_{y_2}y'\right)u_0 + \left(t_{x_2}x' + t_{y_2}y'\right)v_0\right)\right) \end{aligned} \tag{112}$$

The frequencies of the sinusoids are

$$\begin{aligned} \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} &= \begin{bmatrix} s_{x_1} & t_{x_1} \\ s_{y_1} & t_{y_1} \end{bmatrix} \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \\ \begin{bmatrix} u_2 \\ v_2 \end{bmatrix} &= \begin{bmatrix} s_{x_2} & t_{x_2} \\ s_{y_2} & t_{y_2} \end{bmatrix} \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \end{aligned} \tag{113}$$

Some linear algebra shows that the frequencies of the two projected sinusoids are themselves related by an affine transform (without translation):

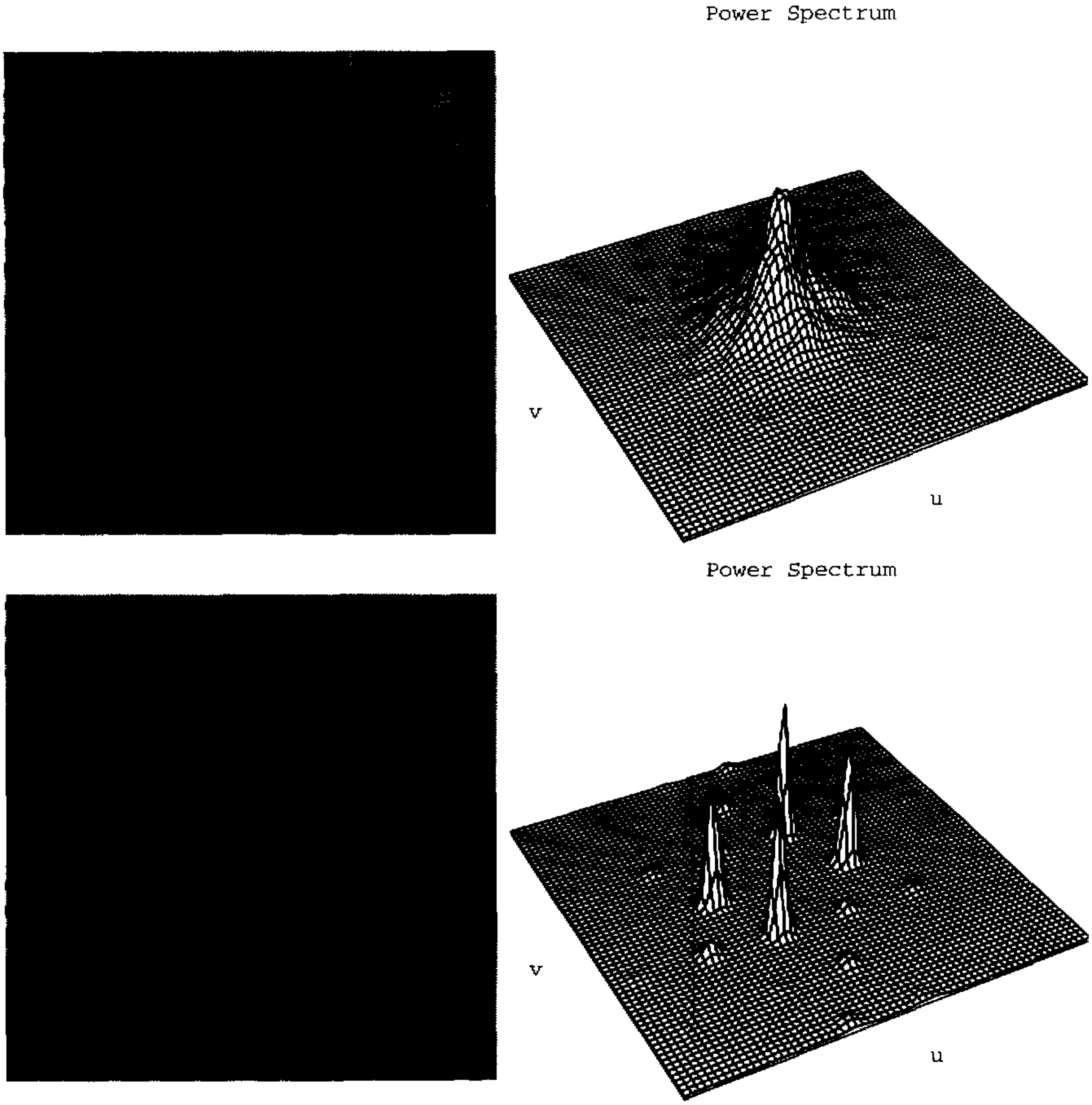


Figure 62: The power spectrum of a random texture (Brodatz D29 beach sand) is a blob, while the power spectrum of a periodic texture (Brodatz D77 cotton canvas) is a group of sharp peaks.

$$\begin{bmatrix} u_2 \\ v_2 \end{bmatrix} = \begin{bmatrix} a_1 & b_1 \\ a_2 & b_2 \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} = \frac{1}{s_{x_1} t_{y_1} - s_{y_1} t_{x_1}} \begin{bmatrix} s_{x_2} t_{y_1} - s_{y_1} t_{x_2} & s_{x_1} t_{x_2} - s_{x_2} t_{x_1} \\ s_{y_2} t_{y_1} - s_{y_1} t_{y_2} & s_{x_1} t_{y_2} - s_{y_2} t_{x_1} \end{bmatrix} \begin{bmatrix} u_1 \\ v_1 \end{bmatrix} \quad (114)$$

To get the full relation in terms of quantities we know, we plug in for the s's and t's from Equation (67). We assume the two points on the textured surface are both on the same plane, thus $(p_1, q_1) = (p_2, q_2) = (p, q)$ and

$$\frac{Z_2}{Z_1} = \frac{d - px_1 - qy_1}{d - px_2 - qy_2} \quad (115)$$

Then

$$\begin{bmatrix} a_1 & a_1 \\ a_2 & b_2 \end{bmatrix} = C \begin{bmatrix} d - px_1 - qy_2 & p(y_2 - y_1) \\ q(x_2 - x_1) & d - px_2 - qy_1 \end{bmatrix} \quad (116)$$

where

$$C = \frac{(d - px_1 - qy_1)}{(px_2 + qy_2 - d)^2} \quad (117)$$

and (x_1, y_1) and (x_2, y_2) are the two points on the image plane being compared.

We conclude that the frequencies of a single sinusoid projected from the same plane to two different points in the image are approximately related by an affine transformation. The affine parameters are functions of the position of the two points on the image, the camera's effective focal length, and the plane's surface normal.

3.7.2 Finding and Matching Peaks

This method requires that we find and match peaks between power spectrum patches. We created a spectrogram preprocessor that performs this task. Given a spectrogram, it first finds the peaks in each patch in order of height. Peaks are found by searching for the maximum value in the power spectrum. Whenever a peak is found, the program computes the location in frequency to subpixel accuracy using the center-of-masses. The peak is then zeroed, and the next largest peak is found.

We know that each peak should have the shape of the power spectrum of the window function, $|W(u, v)|^2$. For zeroing the peaks and computing their centers-of-mass, we can approximate the peaks' support size by examining $|W(u, v)|^2$. The peak-finder keeps looking until the current peak is less than 20% of the magnitude of the largest peak, or until it finds six peaks, whichever comes first. It also ignores peaks below a frequency of 0.03 cycles/pixel. This helps eliminate low frequencies due to shading.

In order to track frequency shifts for computing surface normals, we need to know which peaks in one patch correspond to those in neighboring patches. Our preprocessor matches peaks between each patch and its two neighboring patches to the right and below. We do this pairwise matching by considering every possible match combination between the two sets of peaks, including leaving some peaks unmatched. Appendix 5 shows that with six peaks in each patch, there are 13,327 possible sets of matches to consider, and there are fewer to consider if there are fewer peaks. We pick the combination that has simultaneously the most matches and no matches whose corresponding surface normal exceeds the largest surface normal we expect in the scene.

After this preprocessing step we do not need the original spectrogram for any of the subsequent operations. It is adequately represented by the peaks and peak matches.

3.7.3 Algorithm and Results

We compute surface normals by finding the (p, q) that best accounts for the observed frequency shifts between neighboring patches. This computation usually involves several different peaks in several different patches. We gather all the peaks, peak matches, and their patch coordinates into parallel arrays. For each patch in the region of interest, we check to see if either the patch to the right and/or the patch below is also in the region. If so, we check each peak in the original patch to see if it has a match in either of these two neighboring patches. We only consider adjacent pairs of patches, that is, the patches that have had their frequency peaks matched by the preprocessor. We put all the pairs of peak matches and the corresponding patch coordinates into parallel arrays: (u_{1i}, v_{1i}) is matched with (u_{2i}, v_{2i}) from patches centered at (x_{1i}, y_{1i}) and (x_{2i}, y_{2i}), respectively, for $i = 1$ to m pairs of peaks. If we write the affine parameters from Equation (116) as functions of the surface normal and patch coordinates, we have

$$\mathrm{ssd}\ (p, q) = \sum_{i=1}^{m} \left| \begin{bmatrix} u_{2i} \\ v_{2i} \end{bmatrix} - \begin{bmatrix} a_1(p, q, x_{1i}, y_{1i}, x_{2i}, y_{2i}) & b_1(p, q, x_{1i}, y_{1i}, x_{2i}, y_{2i}) \\ a_2(p, q, x_{1i}, y_{1i}, x_{2i}, y_{2i}) & b_2(p, q, x_{1i}, y_{1i}, x_{2i}, y_{2i}) \end{bmatrix} \begin{bmatrix} u_{1i} \\ v_{1i} \end{bmatrix} \right|^2 \tag{118}$$

This will be small if we have the correct surface normal and the correct matches among the peaks. We perform an exhaustive search over a grid in (p, q) and take the surface normal that minimizes $\mathrm{ssd}\ (p, q)$ as the solution. We use the same 60x60 search in gradient space that we used for the other two algorithms. This algorithm is similar to one developed by Super and Bovik[97]. One difference is

description	value(s)
window center spacing	15 pixels horizontally and vertically
window size	64x64 pixels
peak width for zeroing found peaks	1/16 cycles/pixel
minimum frequency	1/32 cycles/pixel
minimum (peak height)/(maximum peak height)	1/5
maximum number of peaks per patch	6
maximum distance between matched peaks	1/20 cycles/pixel
search grid resolution	60x60
search grid limits	$(\lvert p\rvert, \lvert q\rvert) \leq (2, 2)$

Table 9: Magic numbers used in peak-matching algorithm

that ours uses multiple frequency peaks from a single texture, while theirs uses a single, dominant frequency at each point. The magic numbers used in this algorithm are shown in Table 7.

We ran our peak-matching algorithm on the ten images in Figure 34 - Figure 43. We computed the spectrograms with 15-pixel center-to-center distances and 64x64 windows - the same as for Algorithm 2. The results are shown in Table 8, where the average error is 3.9°.

Two important parameters that affect the accuracy of our solution are the number of patches used to compute the surface normal and the center-to-center spacing of the power spectrum patches. For a given center-to-center spacing, we would like to use as many patches as possible, as long as they all

texture	true (p, q)	true (σ, τ)	computed (p, q)	computed (σ, τ)	error
woven aluminum wire (D6)	(0.614, 0.364)	(35.5°, 30.7°)	(0.508, 0.237)	(29.3°, 25.0°)	6.9°
French canvas (D21)	(0.614, 0.364)	(35.5°, 30.7°)	(0.644, 0.237)	(34.5°, 20.2°)	6.1°
oriental straw cloth (D53)	(0.614, 0.364)	(35.5°, 30.7°)	(0.576, 0.169)	(31.0°, 16.4°)	9.0°
cotton canvas (D77)	(0.614, 0.364)	(35.5°, 30.7°)	(0.712, 0.305)	(37.8°, 23.1°)	5.0°
screen guard	(0.714, 0.0)	(35.5°, 0.0°)	(0.576, 0.034)	(30.0°, 3.4°)	5.8°
grid paper	(−1.878, 0.0)	(62.0°, 180.0°)	(−1.797, −0.034	(60.9°, 181.0°)	1.4°
checkered paper	(−1.878, 0.0)	(62.0°, 180.0°)	(−1.932, −0.034	(62.6°, 181.0°)	1.1°
egg crate	(−1.373, 0.0)	(53.9°, 180.0°)	(−1.322, −0.034	(52.9°, 181.5°)	1.6°
bumpy plastic	(−1.373, 0.0)	(53.9°, 180.0°)	(−1.390, 0.034)	(54.3°, 178.6°)	1.2°
doormat	(−1.373, 0.0)	(53.9°, 180.0°)	(−1.390, 0.034)	(54.3°, 178.6°)	1.2°
average error					3.9°

Table 10: Results of peak matching on textured plates in Figure 34 - Figure 43

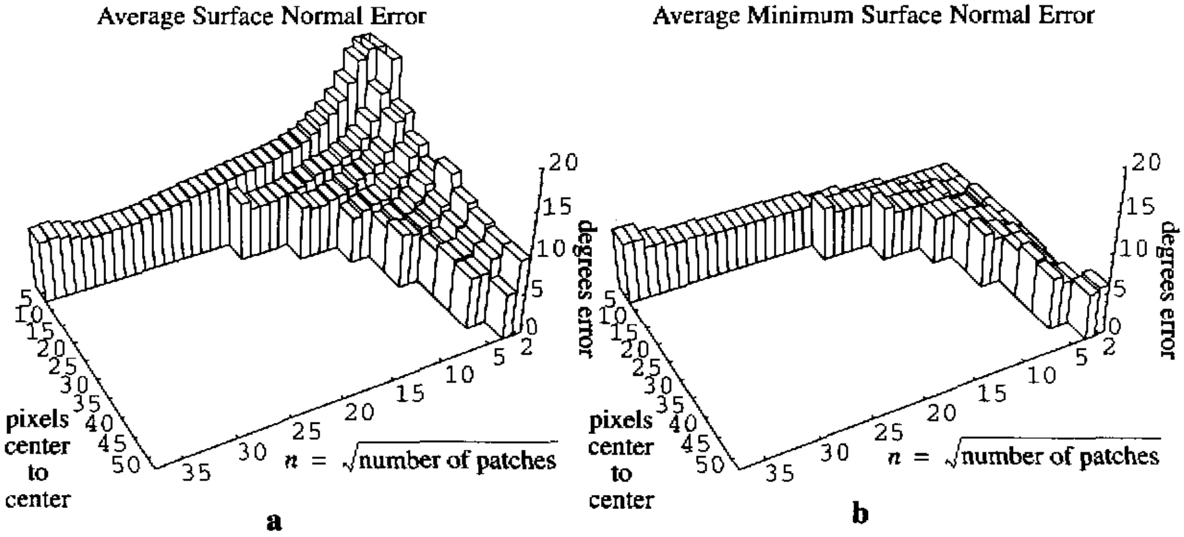


Figure 63: Average errors in surface normal from the four test images for different patch center-to-center distances and different numbers of patches.

fall on the same textured plane, in order to have more data contributing to the solution. We would also like to avoid small center-to-center distances, because the shape-induced frequency shifts would be dominated by noise and approximation errors. We tested our algorithm on the four Brodatz textures in Figure 34 using different numbers of patches and different center-to-center spacing. In each trial, the center-to-center spacing was equal in x and y. We let this parameter vary from 5 to 50 pixels in increments of 5. For each center-to-center distance, we computed the surface normal using as many unique $n \times n$ squares of adjacent patches as would fit on the textured part of the image, starting with $n = 2$.

Figure 63a shows the average errors in degrees of our surface normal estimates for different numbers of patches and different center-to-center spacings. The average was taken over all four images and over all the $n \times n$ squares of patches that would fit on the texture. As expected, the error decreases for larger numbers of widely spaced patches, with the best estimates being in error by about six degrees. Our shape-from-texture algorithm succeeds in giving good results on periodic textures without the need for image feature detection.

Since we use the space/frequency representation, it is possible to integrate our surface normal algorithm into a segmentation algorithm that works on 3D textured, planar surfaces. Unfortunately the need for accuracy conflicts with the requirements of our segmentation algorithm in terms of the number of patches and center-to-center spacing. Our segmentation algorithm begins by estimating surface normals using small parts of the image. Using small support for these estimates is important, because we do not want the support to overlap texture boundaries. This means we have to keep n and the center-to-center spacing small, which tends to compromise accuracy according to Figure 63a. Fortunately, though, some of the estimates from the n x n squares are still good, even with small support and small n. Figure 63b shows the average minimum error in surface normal, where the minimum is taken over all the n x n squares and the average over the four images. In almost every case, at least one of the n x n squares gave a fairly accurate surface normal. Since we start our segmentation with many seed regions, we are likely to have some that are "good", even with small support. This affects our choice of n and the center-to-center spacing in the segmentation algorithm that we present in the next chapter.

3.8 Summary: Us vs. Them

This chapter presents three different shape-from-texture algorithms, all based on the spectrogram. The key observation was that perspective induces an affine transformation on projected texture patches, and thus the Fourier transform also undergoes an affine transformation. For two patches from the same textured plane, the only unknown in the affine transformation is the plane's surface normal. The errors from all three algorithms are about the same. SSD patch-matching had an average error of 3.5°, differential patch-matching 5.1°, and peak-matching 3.9°. For irregular textures, the SSD patch-matching had an average error of 7.5°. Formulating and solving the shape-from-texture problem with local spatial frequency is an advantage, since the same representation can be used to understand aliasing and focus. As we show in the next chapter, it is also good for segmentation.

The first two algorithms, SSD patch-matching and differential patch-matching, operate directly on the raw spectrogram values. Their strength is their simplicity, as they require no feature-finding and few magic numbers.

The peak-matching algorithm requires some potentially unreliable peak-finding and peak-matching. Our preprocessor is robust enough, however, that the accuracy is as good as the other two algorithms. We gain in speed, simplicity, and memory usage by considering only peaks as opposed to the whole frequency plane. The peak-matching algorithm has a subtle advantage in that the affine transformation affects only the position of the peaks. If we assume that the underlying texture is periodic, then we know that the positions of the frequency peaks in two patches will be related by an affine transformation. Ideally these peaks are delta functions. In reality, when we compute a discrete-space power spectrum of these patches, a peak at (u_0, v_0) will be broadened into $|W(u-u_0, v-v_0)|^2$. The corresponding peak in another patch will be at a different frequency, but it will be the same shape. This violates our premise for the first two algorithms in that they work on the assumption that the whole broadened peak undergoes an affine transformation, not just its position. The peak-matching algorithm avoids this error by just tracking the position of the peaks.

Judged in terms of accuracy, our algorithms are about the same as most other shape-from-texture algorithms. Table 11 presents the average angular error of 11 other shape-from-texture algorithms for planar textures. Our algorithms are at the bottom of the table. Many shape-from-texture algorithms are not presented along with ground truth, so it's not possible to compare with these. Nor is there a standard set of test textures, so comparisons like this must be judged both on the raw results and the difficulty and number of test textures.

As we expect, the best results are on synthetic, purely computer-generated textures, as seen in the algorithms by Aloimonos and Kanatani. The errors here are 0.3°and 1.3° respectively. The algorithm by Aloimonos and Swain, which works on discrete texture elements, has an error of only 1.7°. These textures were created by sticking paper texture elements onto otherwise uniform objects. For real periodic textures, our SSD patch-matching and peak-matching algorithms have a slight edge over the others in accuracy. The two most impressive algorithms, in terms of accuracy and difficulty of test images, are those by Aloimonos and by Jau and Chin.

Aloimonos' algorithm achieves an average error of 4.0° on seven real images of periodic and irregular textures. His algorithm starts by finding edges in the image. It computes surface normals based on deviations from uniform edge density. As we mentioned in Section 1.2.9 on low-level feature models of image texture, edges can be considered a high-passed, thresholded version of a local frequency representation. It is not clear from Aloimonos' paper what all the magic numbers were or that the same magic numbers were used for all the test images.

The other impressive algorithm in terms of accuracy is the one by Jau and Chin. They achieved an average error of 2.3° on three different irregular Brodatz textures. Their images were only 96x64 pixels, making the problem even harder. They used the Wigner distribution, which is one version of the space/frequency representation, discussed in Section 2.3.5. They picked a frequency threshold and computed the amount of high frequency energy at every pixel. The spatial variation in high frequency energy is related to the surface normal.

Both of these algorithms are similar in that they examine the spatial variation in high frequency content. We like our algorithm better for two reasons. The first reason is that setting a hard threshold to designate high frequency could cause a problem. These algorithms depend on a smooth change in high frequency content. But, if an isolated peak were to shift across the threshold, there would be a jump in high frequency content that would be incorrectly attributed to the surface normal. Jau and Chin report a substantial jump in error when the threshold is too high or too low, and Aloimonos does not report on how his edge detector affects his results. The second reason is that we avoid integrating out the high frequency structure of the local frequency distribution. By reducing the frequency content to one number, the arrangement of the frequency distribution is lost. Retaining this structure allows us to do detailed reasoning about the frequencies for segmentation (Chapter 4) and to account for aliasing and potentially camera blur.

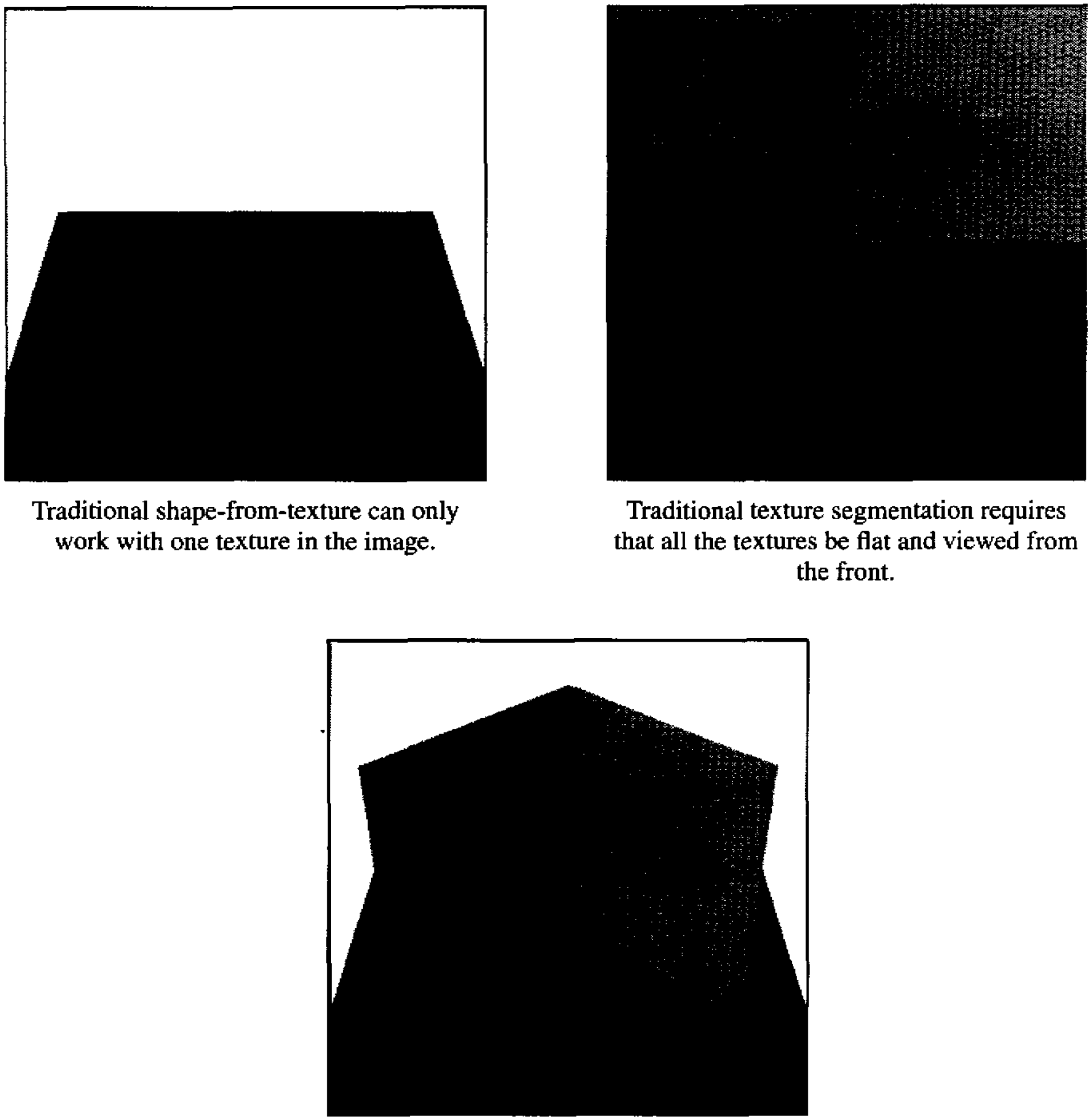
Citation	Comments	Type of Test Images	Number of Test Textures	Average Error
Aloimonos[2]	uniform texel density	synthetic random	2	0.3°
	uniform edge density	real periodic & random	7	4.0°
Aloimonos & Swain[3]	randomly placed, discrete texels	objects with texels stuck on	1	1.7°
Brown & Shvaytser[11]	isotropic frontal autocorrelation	real random	3	8.9°
Chen *et al.*[15]	fractal textures	Brodatz random	2	13.9°
Garding[31]	weakly isotropic edges	random wallpaper	2	9.6°
Jau & Chin[45]	Wigner distribution	Brodatz random	3	2.3°
Kanatani & Chou[51]	uniform density of dots or lines	synthetic periodic	6	1.3°
		synthetic random	6	8.5°
Malik & Rosenholtz[68]	spectrogram	Brodatz periodic	3	3.7°
Patel & Cohen[82]	Gauss Markov random fields	real periodic	1	5.5°
Super & Bovik[97]	Gabor filters	real periodic	10	5.9°
Super[99]	(p,q) tuned filters	real periodic	10	4.7°
Krumm	ssd patch-matching	real & Brodatz periodic	10	3.5°
Krumm	averaged ssd patch-matching	Brodatz random	4	7.5°
Krumm	differential patch-matching	real & Brodatz periodic	10	5.1°
Krumm	peak-matching	real & Brodatz periodic	10	3.9°

Table 11: Average angular errors of several different shape-from-texture algorithms

Chapter 4 Segmenting Textured 3D Surfaces

4.1 Combining Texture Segmentation and Shape-from-Texture

As we showed in the first chapter, there has been much work in both texture segmentation and shape-from-texture. Unfortunately, almost none of this work can be directly applied to textured surfaces viewed under general conditions. All shape-from-texture work is based on the assumption that the image contains only one texture, or at least that the texture has been segmented from the rest of the image, as in the upper left of Figure 64. This is because the shape algorithms attribute changes in the texture's appearance in the image to shape effects, not to changes in the actual, frontally-viewed texture such as might be caused by two or more textures in the image. A more realistic image appears at the bottom of Figure 64. It contains multiple textures that are not viewed frontally. It is clear that if a conventional, whole-image, shape-from-texture algorithm were applied to this image, it would give an incorrect result. On the other hand, almost all texture segmentation algorithms are based on the assumption that shape effects are negligeable, as in the upper right of Figure 64. Without this assumption, changes in depth and surface normal could alter the appearance of an otherwise uniform texture enough to warrant splitting it into different regions. We show an example of this problem in Figure 65. Both images have three textured surfaces in them. In the left image the surfaces are all frontal, and



Traditional shape-from-texture can only work with one texture in the image.

Traditional texture segmentation requires that all the textures be flat and viewed from the front.

A more realistic scene contains multiple textures that are not viewed from the front. Our aim is to segment images like this.

Figure 64: Combining old texture problems into a new one with relaxed assumptions

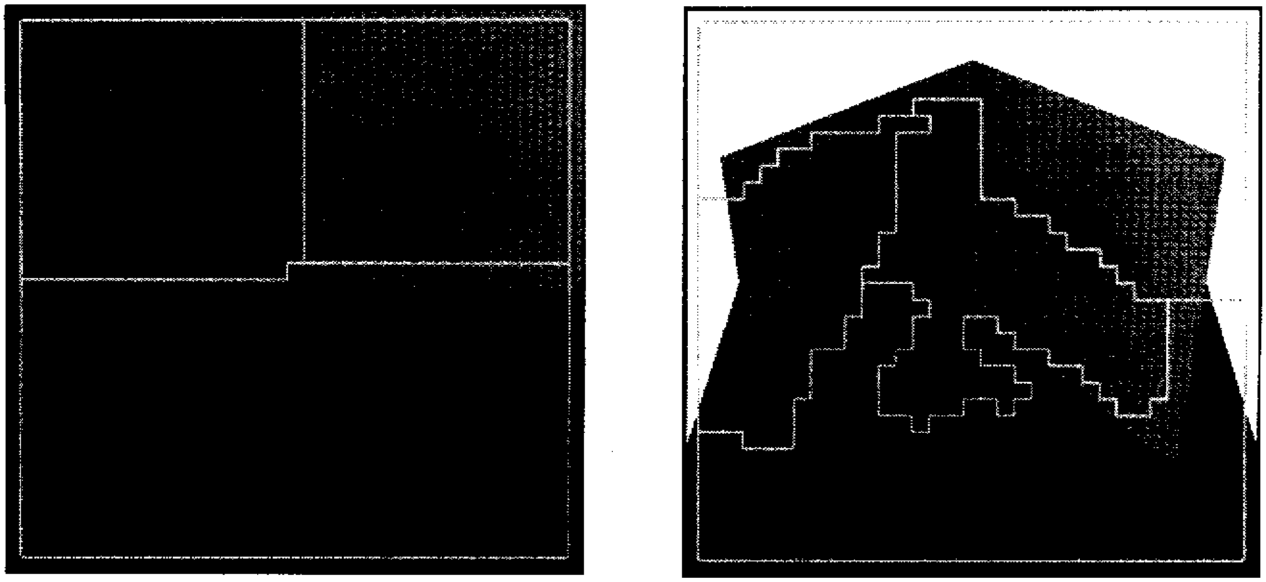


Figure 65: A traditional-style texture segmentation program works well on the three frontal textures in the left image. When the same textures are not viewed frontally, however, the program fails.

the edges found by a segmentation program are about right. (These edges were found by a simplified version of the algorithm we develop in this chapter.) When we apply the same segmentation program to the image with nonfrontal textures, the edges are no good.

Our goal was to create an algorithm that can segment images of textured 3D surfaces. The next section tells how we did it by explicitly accounting for 3D shape effects. To our knowledge, there have not been any other successful algorithms for this problem. Our algorithm is based on the spectrogram of the image -- the same representation we used to compute surface normals in Chapter 3. This shows how the space/frequency representation is a versatile way of solving disparate problems in computer vision. The algorithm we describe is based on assumptions that the textures are periodic and the surfaces are flat. The results in Section 4.3 show that the algorithm works on computer-generated Brodatz images and real images taken in the lab. In particular, the algorithm works well for the image on the right of Figure 65, on which a traditional segmentation fails.

4.2 Algorithm for Segmenting Textured 3D Surfaces

Our algorithm for segmenting 3D textured surfaces consists of building a dendrogram (hierarchical region-growing) using a description length merging criterion. We start with an image spectrogram computed with centers 15 pixels apart. The segmentation begins with each power spectrum patch as its own, separate region. The power spectra are represented by just their peaks, using the peak-finding program described in Section 3.7.2. We build a dendrogram by merging adjacent regions. The quality of a potential merge between two regions is judged by the compactness of a 2D clustering of the peaks from the regions' patches. We measure compactness using a minimum description length criteria. Before we assess the compactness, however, we determine the surface normal of the proposed region (using the peak-matching algorithm from Section 3.7.2) and use this to compute where the frequency peaks would be if the texture were viewed from the front. This "frontalization" step undoes the 3D effects and makes adjacent regions of similar texture appear similar in spite of depth changes.

This section first explains the four most important subparts of this algorithm in detail -- the dendrogram with its description length criteria, the frontalization step, a fast way to compute surface orientation, and a method for refining region boundaries -- and then summarizes the entire algorithm.

4.2.1 Clustering With a Dendrogram and a Description Length Criterion

We use a dendrogram twice in our segmentation algorithm: once for clustering frequency peaks from frontalized power spectrum patches and once for merging the patches themselves. This section describes a simple clustering algorithm based on the dendrogram that we use for clustering frequency peaks. As described by Duda and Hart[24], a dendrogram is a tree data structure used for unsupervised, hierarchical clustering. We show an example dendrogram in Figure 66. This hierarchical clustering of sample points is one in which each point is first considered to be its own cluster. Two of these clusters are merged to form a new clustering with one less cluster. At each subsequent level, two more clusters are merged, reducing the number of clusters by one. There is only one cluster at the end. In "agglomerative" clustering, the tree is built starting at its leaves, with each sample point first serving as its own cluster. In "divisive" clustering, the tree is built from its root, with the first cluster being all the sample points together. We use agglomerative clustering in our algorithm. The desired clustering is taken as a horizontal cut across the tree at some level giving some number of clusters.

One way of building an agglomerative dendrogram is to simply merge the two clusters that are most similar at any given level. This does not ensure the best dendrogram, in that it will not necessarily pass through the "best" clustering, but it works well for us. Actually, our inspiration for this technique came from the more sophisticated agglomerative algorithm of R. Wallace[110]. His algorithm can start with a dendrogram and improve it. In the example in Figure 66, the merge criterion was simply the distance between the cluster means. At every level we merged the two clusters whose means were closest.

The dendrogram gives a series of good candidate clusterings, each with a different number of clusters. The remaining problem is to choose the number of clusters from these candidates. A clever

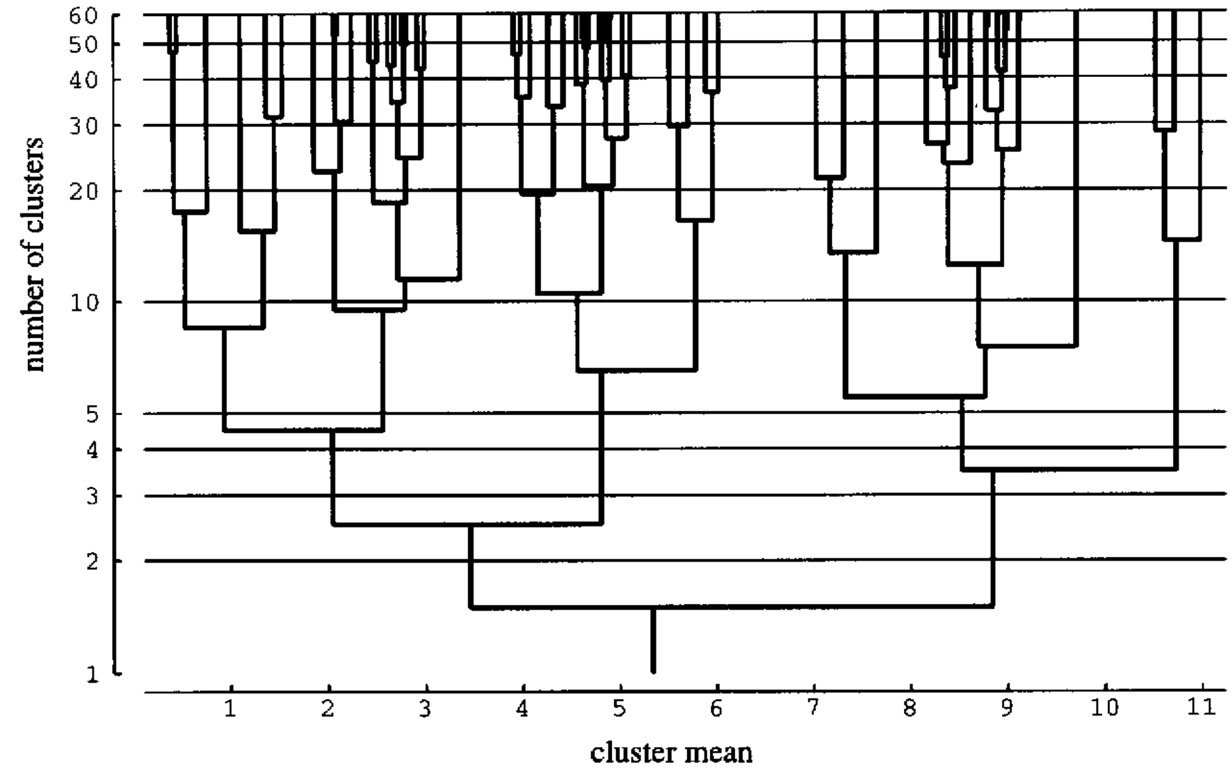


Figure 66: This dendrogram was built starting with 20 samples each of Gaussian distributed scalars with means of two, five, and nine, and a standard deviation of one. The merge criterion was simple distance. The horizontal grid lines represent various possible clusterings, with three being the "correct" number.

method for solving this problem comes also from R. Wallace. He proposed using a *description length* criterion for making this choice. The description length is a numerical measure of the length of a description of a set of data. This idea was begun, and continues to be refined, by Jorma Rissanen[89]. For most problems the description length consists of two parts: the "model" description length and the "error" description length. In our case the "model" is the number of clusters along with which points belong in which cluster. Each point is modeled by its cluster center. The "error" is simply the deviation between the actual data points and the model. The description length expresses the trade-off between the complexity of the model and the amount of error. More clusters means the model is larger, but the error is smaller. Fewer clusters reduce the size of the model but increase the size of the error.

We can express the description length in terms of bits. The length of the "error" term comes from a well-known result in coding theory. The ideal code length, in bits, for a set of points drawn from a stochastic process is the negative of the log-base-two of the probability of that set of points. For scalar data, we say that the probability distribution of a point in cluster i is $p(x;\theta_i)$, where θ_i give the parameters of the distribution (*e.g.* the mean and variance for a normal distribution). The probability of seeing any given x is zero, but the probability of seeing any given x to within a certain range Δ (which is the best we can do with a computer's finite precision) is just

$$P\left(-\frac{\Delta}{2} < x < \frac{\Delta}{2}\right) = \int_{x-\frac{\Delta}{2}}^{x+\frac{\Delta}{2}} p(\tau|\theta_i)\,d\tau \approx \Delta p(x|\theta_i) \tag{119}$$

where the approximation holds for small Δ. We need the probability of a given clustering if there are n_c clusters. If cluster i has n_i independent points x_{ij}, then the probability is

$$\prod_{i=1}^{n_c} \prod_{j=1}^{n_i} \Delta p(x_{ij}|\theta_i) \tag{120}$$

The ideal code length is then

$$-n\log_2(\Delta) - \sum_{i=1}^{n_c} \sum_{j=1}^{n_i} \log_2(p(x_{ij}|\theta_i)) \tag{121}$$

where n is the total number of points.

The length of the "model" term in the description length is not as well-defined. Part of the model consists of the parameters used to describe the probability distributions of the clusters. Rissanen recommends the formula $\frac{k}{2}\log_2(m)$, where k is the number of free parameters in the model and m is the number of data points. This formula holds for real-valued parameters in the limit as $m \to \infty$. Although our m is finite, the formula works for us. We also include the number of bits it would take to label each point with its integer cluster number. This amounts to $n\log_2(n_c)$. We have not seen this term included in any other description length formulae, but it is effective in our application.

If we apply Rissanen's formula for the model parameters on a cluster-by-cluster basis, the total description length becomes

$$n\log_2(n_c) + \frac{1}{2}\sum_{i=1}^{n_c} k_i\log_2(n_i) - n\log_2(\Delta) - \sum_{i=1}^{n_c} \sum_{j=1}^{n_i} \log_2(p(x_{ij}|\theta_i)) \tag{122}$$

For lack of anything better, we take the probability distribution as Gaussian. This makes $k_i = 2$ and

$$p(x_{ij}|\theta_i) = \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{1}{2}\left(\frac{x_{ij}-\mu_i}{\sigma_i}\right)} \tag{123}$$

Taking the Gaussian density gives a description length formula of

$$\begin{aligned} & n\log_2(n_c) + \sum_{i=1}^{n_c} \log_2(n_i) - n\log_2(\Delta) + \\ & n\log_2(\sqrt{2\pi}) - \sum_{i}^{n_c} n_i \log_2(\sigma_i) + \frac{1}{2}\log_2(e) \sum_{i=1}^{n_c} \sum_{j=1}^{n_i} \left(\frac{x_{ij}-\mu_i}{\sigma_i}\right)^2 \end{aligned} \tag{124}$$

Since our goal is to minimize the description length, we can drop the constant terms. For our algorithm, we will also assume that we know the standard deviations of the clusters beforehand, and that they are equal to σ for all the clusters. (We estimate the μ_i from the clusters themselves.) This gives a final description length formula of

$$n\log_2(n_c) + \sum_{i=1}^{n_c} \log_2(n_i) + \frac{1}{2}\log_2(e) \sum_{i=1}^{n_c} \sum_{j=1}^{n_i} \left(\frac{x_{ij}-\mu_i}{\sigma}\right)^2 \tag{125}$$

The first two addends of this formula give the "model" description length, and the last one gives the "error" description length. In this case, the "error" description length is proportional to the sum of the squared Mahalanobis distances.

A plot of the description length for the dendrogram in Figure 66 appears in Figure 67. The "model" description length grows quickly with an increasing number of clusters, while the "error" term drops quickly. The sum is minimized at three, which is the actual number of clusters used to generate the original data.

Combining the dendrogram and a minimum description length criteria gives a good method of clustering data with a minimum of "magic numbers". The dendrogram serves to merge data points with, effectively, a progressively larger and larger similarity threshold. The user must supply a merge criterion, but for low-dimensional data, it is usually easy to find one that is satisfactory. We used simple Euclidean distance for our dendrogram. No other input parameters are necessary for building the dendrogram. The description length suffers from one magic number and perhaps a few "magic formulae". The magic number is the standard deviation. Although this can sometimes be estimated from the

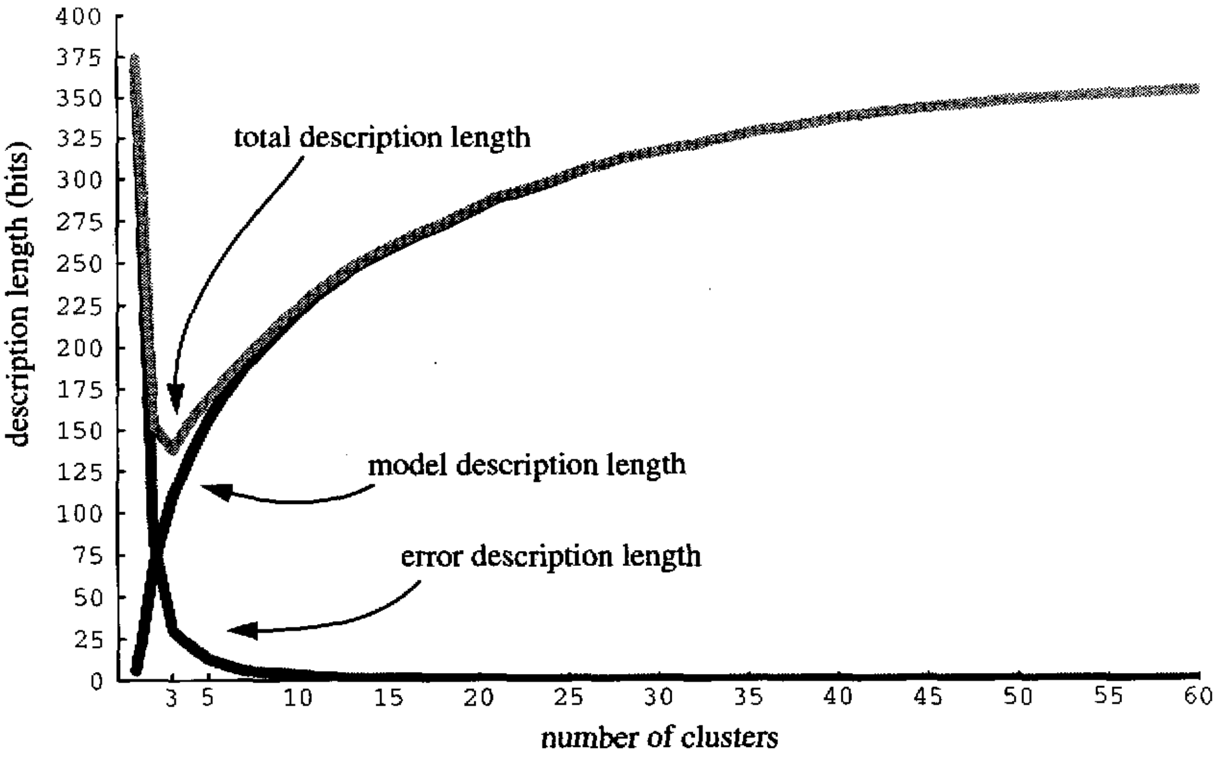


Figure 67: The "model" description length and "error" description length sum to form a total description length with its minimum at the correct number of clusters.

clusters themselves, it cannot be estimated from single-point clusters, and the estimate would not be stable for small clusters. We must assume a form of the probability distribution for the "error" description length. It is usually possible to argue for a Gaussian distribution, appealing either to the central limit theorem or the fact that everyone else does it and it seems to work. The "model" part of the formula is more difficult. Rissanen continues to update his formula, and we have found a new variation that works well.

4.2.2 Dendrograms Within a Dendrogram

We segment the image by building a dendrogram of the power spectrum patches. Each image region is represented by its frequency peaks. At the beginning, each patch is its own region, which gives roughly 1000 initial regions for a spectrogram computed on windows that are 15 pixels from center to center. We only allow merges between adjacent, four-connected regions. The key element in building this dendrogram is the similarity criterion between image regions. The question is how well two sets of frequency peaks go together. (As we describe below in Section 4.2.3, we first undo the 3D effects that move the peaks around, so we don't have to worry at this point about variations due to perspective distortion.) A simple method of making this assessment would be to compute the sum of the distances between matching peaks. This has at least three problems:

1. Because of noise, even similar textures will not always have the same number of peaks. Would we attribute unmatched peaks to actual differences in texture or to noise?

2. It would be hard to arbitrate between similarity measures based on different numbers of peaks. Would it better to merge a region with two out of two peaks matched or a region with five out of five peaks matched? Combined with the problem above, we might have to decide between two out of two versus four out of five. In addition, we would have to account for the quality of the matches. So, would it be better to merge regions with two out two peaks matched precisely or four out of five peaks matched not-so-precisely?

3. In order to avoid mistakes in matching, our spectrogram preprocessor only matches peaks of adjacent, four-connected patches. This is sufficient for computing surface normals. But for assessing the similarity of a group of patches, we need a more global measure of similarity. We could chain through the local matches, but the chain could be broken or misdirected by even one missing or bad match. Also, image patches that straddle two different texture regions could have peaks from both regions, creating a bridge between the regions that would mask their differences.

We found that we could neatly avoid giving explicit answers to these questions by measuring similarity as the minimum description length (mdl) of a clustering of the 2D peaks in pairs of regions that are candidates for merging. We expect that regions that belong together will have frequency peaks that give compact clusters. For each candidate pair of regions, we gather all their frequency peaks into an array and pass them off to our 2D clustering program. This program builds a dendrogram of the points using Euclidean distance as a merge criterion. It returns the minimum description length over all levels of the dendrogram as the measure of similarity between the two regions. For the 2D clustering we use a formula similar to the 1D clustering in Equation (125), except we use the 2D Mahalanobis distance and $k = 4$. The formula is

$$n\log_2(n_c) + 2\sum_{i=1}^{n_c}\log_2(n_i) + \frac{1}{2}\log_2(e)\sum_{i=1}^{n_c}\sum_{j=1}^{n_i}\left(\frac{u_{ij}-\bar{u}_i}{\sigma}\right)^2 + \left(\frac{v_{ij}-\bar{v}_i}{\sigma}\right)^2 \qquad (126)$$

where $(\bar{u}_i, \bar{v}_i)$ are the estimates of the cluster centers (computed from the clusters) and σ is the standard deviation of the frequency peaks (which must be supplied by the user). After a few experiments, we found that $\sigma = 0.01$ cycles/pixel is a good value to use.

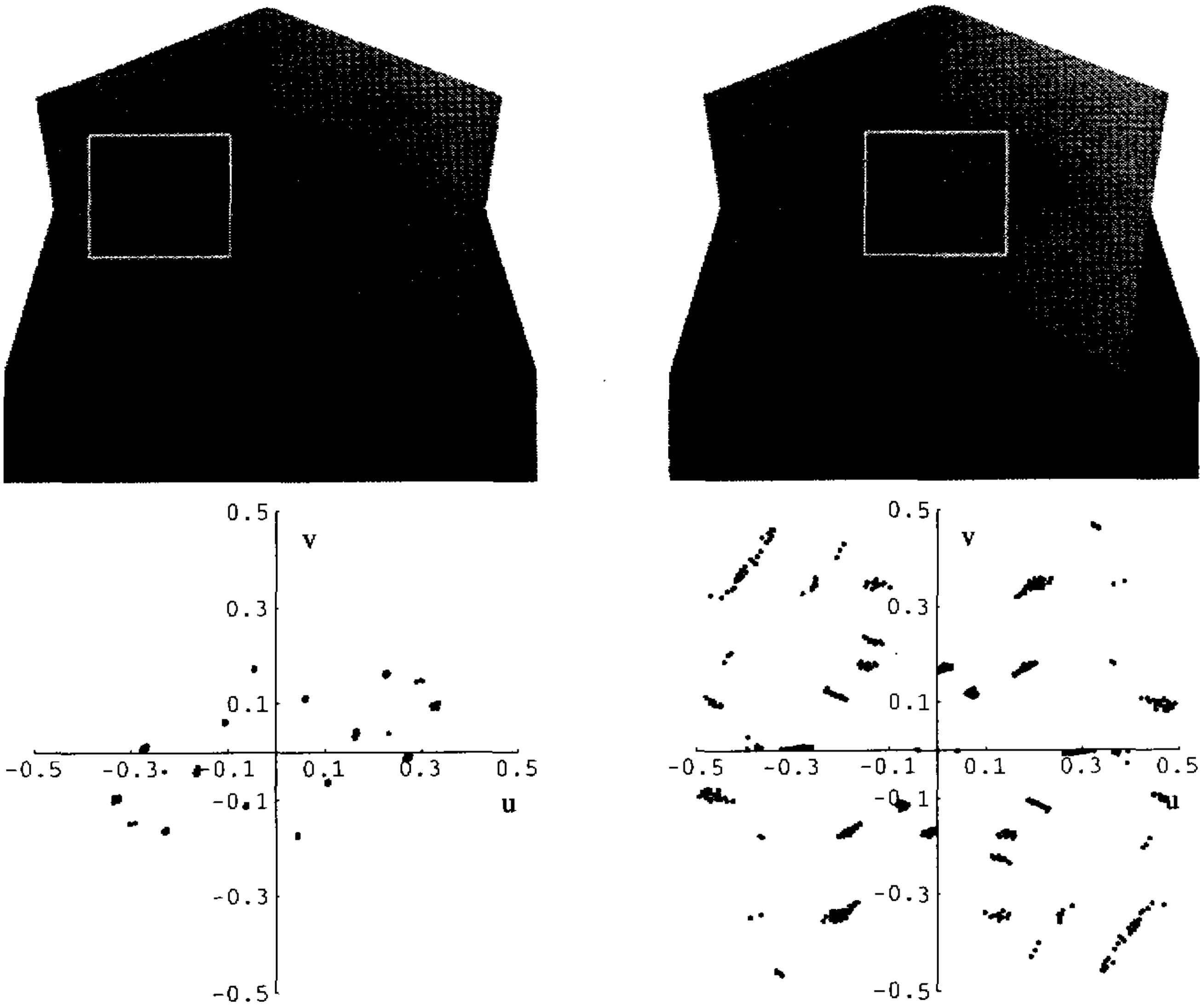


Figure 68: The frequency peaks from the enclosed image regions of equal area are plotted. After frontalization (described in Section 4.2.3), the peaks from the window covering only one texture cluster more compactly than those from the window covering two textures. The minimum description length (mdl) is a good way to measure the compactness of the clustering. The mdl's for the peaks in the left and right outlined regions are 1237.9 bits and 2726.9 bits, respectively.

Figure 68 shows frequency peaks from two regions of an image with textured surfaces. The region in the left image covers only one texture, and its peaks form compact clusters. The peaks from the region in the right image, which covers two different textures, are scattered. The minimum description length of the clustering reflects the compactness of the points. The single-texture peaks have an mdl of 1237.9 bits, while the two-texture peaks have an mdl of 2726.9 bits. (Both sets of peaks have been frontalized according to the derivation in the next section.

We add one other term to the description length above in order to evaluate a potential merge. In his paper on gray-level image segmentation, Leclerc[62] argues for a term that measures the description length of the boundary of image regions. He describes boundaries by a chain code of unit-length line segments. For four-connected regions, each segment of the chain code can go one of three ways from the previous segment. Thus, the description length should be a scalar b times the length of the region's boundary. For infinitely long chain codes, $b = \log_2(3) \approx 1.585$. For shorter chains, b should be slightly larger, and Leclerc recommends using nothing larger than two. For our algorithm, $b = 2$ is satisfactory. The total description length for a potential merge between two patches is then the cluster description length of Equation (126) plus b times the length of the boundary formed by the new region. The description length of a candidate merge pair must be considered against the description lengths of the two regions separately. Our segmentation program actually computes the *decrease* in description length for each candidate pair of regions and picks the pair with the greatest decrease for the next merge. Examples of dendrograms built this way appear in the results in Section 4.3. We show selected cuts from the image dendrograms, where each region is shaded differently. In all cases, the dendrogram passes through a good segmentation, so the description length is a valid merge criterion.

Just as the description length criterion can be used to pick the right number of clusters for the frequency peaks, we would like to use the same concept to pick the right number of regions from the image dendrogram. If our description length criterion were perfect, we would pick the image dendrogram cut with the minimum description length as the best segmentation. With our current formulation, however, the minimum occurs well before the best segmentation, so this choice must be made manually. This is an area for further research.

4.2.3 Frontalizing

This section describes our frequency peak frontalization algorithm. Our goal is to determine what a group of frequency peaks on different patches would be if we viewed the texture from the front. We know from Equation (114) in Chapter 3 that a frequency (u_0, v_0) on a non-frontal textured surface in the scene is related by an affine transformation to a frequency (u_i, v_i) on the image plane. In matrix form, this is

$$\begin{bmatrix} u_i \\ v_i \end{bmatrix} = \begin{bmatrix} s_{x_i} & t_{x_i} \\ s_{y_i} & t_{y_i} \end{bmatrix} \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = \boldsymbol{S}_i \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \tag{127}$$

We cannot simply invert this relationship for the frontalization, because we don't know the Z_i coordinate of the surface, and this is required to compute the matrix S_i. In fact, we can never compute $[u_0, v_0]^T$, because we never know the depth of the patch.

For frontalizing a given image patch, imagine we have a frontalized reference image patch, $(p, q) = (0, 0)$, with a corresponding depth of Z_{ref} in the scene. It is from the same plane and with the same texture as the given patch. There is a 3D surface patch corresponding to the reference patch whose 4x4 homogeneous transformation matrix would be

$$\begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & X_{ref} \\ 0 & 1 & 0 & Y_{ref} \\ 0 & 0 & 1 & Z_{ref} \\ 0 & 0 & 0 & 1 \end{bmatrix} . \tag{128}$$

Using these transformation parameters and solving Equation (64) in Chapter 3 for s and t gives

$$\begin{aligned} s_{ref}(x', y') &= -x'Z_{ref}/d \\ t_{ref}(x', y') &= -y'Z_{ref}/d \end{aligned} \tag{129}$$

Then the projected frequency from this frontal patch will be approximated as before as an affine transformation of the scene frequency. The affine transformation parameters come from the first partial derivative terms of the Taylor series of $s_{ref}(x', y')$ and $t_{ref}(x', y')$. The frontalized frequency is then

$$\begin{bmatrix} u_{frontal} \\ v_{frontal} \end{bmatrix} = \begin{bmatrix} -Z_{ref}/d & 0 \\ 0 & -Z_{ref}/d \end{bmatrix} \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} = S_{ref} \begin{bmatrix} u_0 \\ v_0 \end{bmatrix} \tag{130}$$

Solving Equation (127) for $[u_0, v_0]^T$ and inserting this into Equation (130) gives

$$\begin{bmatrix} u_{frontal} \\ v_{frontal} \end{bmatrix} = S_{ref} S_i^{-1} \begin{bmatrix} u_i \\ v_i \end{bmatrix} = F_i \begin{bmatrix} u_i \\ v_i \end{bmatrix} \tag{131}$$

When F_i is multiplied out, it elements become

$$\begin{aligned} f_{11} &= C\,[d\,(p^2 + rq^2) - px_i\,(p^2 + q^2)\,] \\ f_{12} &= Cp\,[dq\,(1 - r) - y_i\,(p^2 + q^2)\,] \\ f_{21} &= Cq\,[dp\,(1 - r) - x_i\,(p^2 + q^2)\,] \\ f_{22} &= C\,[d\,(rp^2 + q^2) - qy_i\,(p^2 + q^2)\,] \end{aligned} \tag{132}$$

where

$$C = \frac{Z_{ref}}{dr\,(p^2 + q^2)\,Z_i} \tag{133}$$

This still contains the unknown depth value Z_i. But, since the reference patch is on the same plane, then we have from Equation (75) in Chapter 3:

$$\frac{Z_{ref}}{Z_i} = \frac{d - px_i - qy_i}{d - px_{ref} - qy_{ref}} \tag{134}$$

Putting this ratio into Equation (132) gives the affine frontalization parameters for an arbitrary patch i in terms of known quantities:

$$\begin{aligned} f_{11} &= D\,[d\,(p^2 + rq^2) - px_i\,(p^2 + q^2)\,] \\ f_{12} &= Dp\,[dq\,(1 - r) - y_i\,(p^2 + q^2)\,] \\ f_{21} &= Dq\,[dp\,(1 - r) - x_i\,(p^2 + q^2)\,] \\ f_{22} &= D\,[d\,(rp^2 + q^2) - qy_i\,(p^2 + q^2)\,] \end{aligned} \tag{135}$$

where

$$D = \frac{(px_i + qy_i - d)}{dr\,(p^2 + q^2)\,(px_{ref} + qy_{ref} - d)} \tag{136}$$

The frontalization step works this way: For a group of patches hypothesized to be on the same plane, we arbitrarily pick one patch as the reference patch. In our case we pick the first in the list. We compute the (p, q) of these patches using our peak-matching shape-from-texture algorithm in Section 3.7. The affine frontalization transformation is then computed for each patch according to Equation (135), and each peak frequency is transformed accordingly. This does not tell us what the true frontalized frequencies are, but it tells us what the frequencies would be if all the patches had the same depth as the reference patch, which is good enough for comparing them.

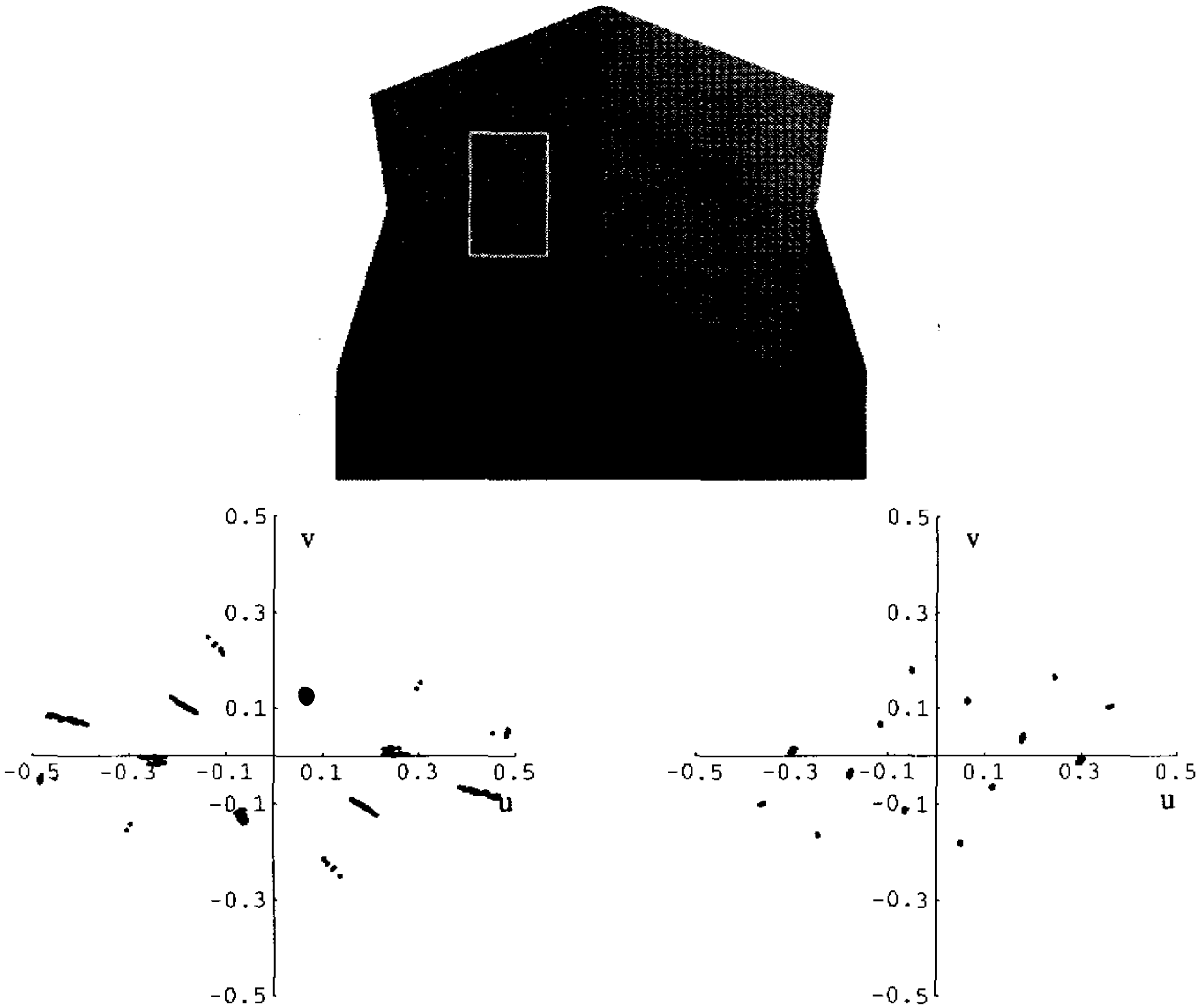


Figure 69: The plot on the left shows the raw peaks from the outlined region of the image. The right plot shows how the clusters are compacted after frontalization. The compaction is reflected in the minimum description length of a clustering of the peaks. The mdl of the peaks on the left is 836.5 bits, while the mdl of the peaks on the right is 537.1 bits.

An example of the effects of frontalization is show in Figure 69. The frequency peaks are significantly more compact after frontalization.

4.2.4 A Fast Way of Computing Shape

Each time we merge two regions in the image, we must recompute several different surface normals. Without a fast way of doing this computation, the segmentation program would take much too long, even for research purposes.

Every region keeps a list of its four-connected neighbors as potential merges. Each of these potential merges is evaluated based on the description length formula presented in Section 4.2.2. Before we can do the peak clustering, we must frontalize the peaks of the two regions. And before we can frontalize the peaks, we must compute the surface normal associated with them. Thus, not only do we have to maintain a surface normal for each region, we must also maintain a surface normal for each unique pair of adjacent, four-connected regions. When two regions are merged, we must update the surface normals between the new region and all its neighbors. Our image dendrograms start with about 1000 separate regions, and to build a dendrogram down to one region, we must perform a number of merges that is just one less than the original number of regions. This means we perform several thousand surface normal computations for each image.

The most obvious way to compute (p, q) for a group of patches is to use the method we described in Section 3.7. Here, we gathered all the peaks, peak matches, and their patch coordinates into parallel arrays. For each patch in the region of interest, we checked to see if either the patch to the right and/or the patch below was also in the region. If so, we checked each peak in the original patch to see if it had a match in either of these two neighboring patches. We put all the pairs of peak matches and the corresponding patch coordinates into parallel arrays: (u_{1k}, v_{1k}) is matched with (u_{2k}, v_{2k}) from patches centered at (x_{1k}, y_{1k}) and (x_{2k}, y_{2k}), respectively, for $k = 1$ to m pairs of peaks. The ssd surface is then given by

$$\text{ssd }(p, q) = \sum_{i=1}^{m} \left\| \begin{bmatrix} u_{2k} \\ v_{2k} \end{bmatrix} - \begin{bmatrix} a_1(p, q, x_{1k}, y_{1k}, x_{2k}, y_{2k}) & b_1(p, q, x_{1k}, y_{1k}, x_{2k}, y_{2k}) \\ a_2(p, q, x_{1k}, y_{1k}, x_{2k}, y_{2k}) & b_2(p, q, x_{1k}, y_{1k}, x_{2k}, y_{2k}) \end{bmatrix} \begin{bmatrix} u_{1k} \\ v_{1k} \end{bmatrix} \right\|^2 . \qquad (137)$$

where (a_1, b_1, a_2, b_2) come from Equation (116) in Chapter 3. We took the location of the minimum of ssd (p, q) as the surface normal.

In our segmentation algorithm, using this method for every surface normal computation would actually result in much redundant computation. We can think of ssd (p, q) as simply a sum of error surfaces from all the unique pairs of patches in the region. This breakdown is possible because we only consider matches between peaks in adjacent patches, not matches over longer distances. We give each patch an integer index, and we say that $\text{ssd}_{ij}(p, q)$ is the error surface generated by considering only the matched peaks between patch i and j. At the beginning of our image dendrogram, we must compute all these pairwise ssd surfaces, because all pairs of adjacent patches are candidates for merging. If we store all the possible pairwise ssd surfaces in memory, we can easily compute the ssd surface for an arbitrary region by simply adding the pairwise ssd surfaces that have both their patches in the region. That is

$$\mathrm{ssd}\,(p,q) = \sum_{i,j \in R} \mathrm{ssd}_{ij}\,(p,q) \tag{138}$$

where R is the region of interest. This saves having to recompute the complicated equations for (a_1, b_1, a_2, b_2), resulting in a significant speedup.

We can do even better once the image dendrogram is under way. Since the region-growing is hierarchical, once two regions are merged, they remain merged. This means we can get another speedup by maintaining an ssd surface for each region. The situation is not as simple as it appears, however, because the ssd surfaces we maintain for each region pertain only to the internal peak matches of the region. When two regions are merged, there are also new matches to consider along their common border. In addition, each time we merge two regions, we must update the surface normals of the potential merges between the newly formed region and all its adjacent neighbors.

In order to properly update the ssd surfaces, we must store three kinds of ssd surfaces for the regions. Each region i has an ssd surface pertaining only to its internal peak matches, which we will call P_i. The peak matches along a common border between two regions i and j also generate an ssd surface, which we will call P_{ij}. In order to assess potential merges, we must compute what the ssd surface would be if two adjacent regions were merged. We designate the ssd surface of a potential merge between regions i and j as M_{ij}, and it is given by $M_{ij} = P_i + P_j + P_{ij}$. This accounts for the internal matches of both regions as well as the matches across their common border. Each ssd surface is a function of all the peaks in a particular region, so no weighting is necessary when combining them. We note that these ssd surfaces are transitive, *i.e.* $P_{ij} = P_{ji}$ and $M_{ij} = M_{ji}$. This save on storage, since we have neighboring regions both pointing to the same memory when appropriate.

We can derive the updating rules for the ssd surfaces by considering a merge between regions "a" and "b" in Figure 70. (For clarity, we will instantiate i and j with the letters a, b, c, d, e, and f rather than numbers.) Before the merge occurs, we have P_i for each region and P_{ij} and M_{ij} for each pair of adjacent regions. After the merge, we will have new ssd surfaces given by P_i^*, P_{ij}^*, and M_{ij}^*. The new region formed by "a" and "b" will be called "a", and "b" will no longer exist.

Clearly the only internal ssd surface that will change is the one for the new region. Thus

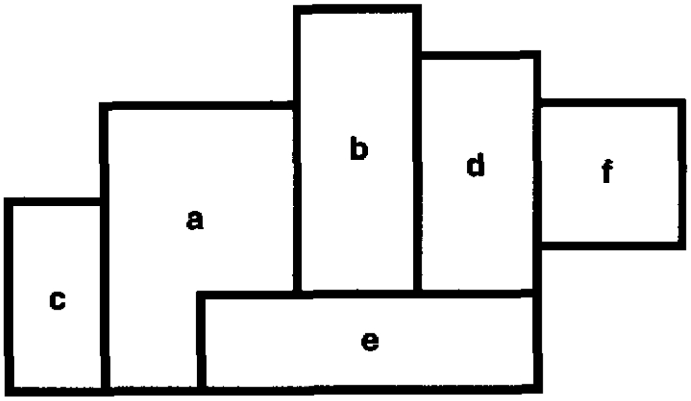


Figure 70: Merging regions "a" and "b" will require updating the ssd surfaces associated with the new region and all its neighbors.

$$\begin{aligned} P_a^* &= M_{ab} \\ P_c^* &= P_c \\ P_d^* &= P_d \\ P_e^* &= P_e \\ P_f^* &= P_f \end{aligned} \tag{139}$$

Updating the P_{ij} and M_{ij} is more interesting. There are four types of regions to consider:

1. The region was a neighbor of "a" but not a neighbor of "b", *e.g.* "c". Then

$$\begin{aligned} P_{ac}^* &= P_{ac} \\ M_{ac}^* &= P_a + P_c + P_{ac} + P_b + P_{ab} \\ &= M_{ac} + P_b + P_{ab} = M_{ab} + P_c + P_{ac} \end{aligned} \tag{140}$$

2. The region was a neighbor of "b" but not a neighbor of "a", *e.g.* "d". Then "a" gets a new neighbor, and

$$\begin{aligned} P_{ad}^* &= P_{bd} \\ M_{ad}^* &= P_d + P_b + P_{bd} + P_a + P_{ab} \\ &= M_{bd} + P_a + P_{ab} = M_{ab} + P_d + P_{bd} \end{aligned} \tag{141}$$

3. The region was a neighbor of both "a" and "b", *e.g.* "e". Then

$$\begin{aligned} P_{ae}^{\ *} &= P_{ae} + P_{be} \\ M_{ae}^{\ *} &= P_a + P_e + P_{ae} + P_b + P_{be} + P_{ab} \\ &= M_{ab} + P_{ae} + P_{be} + P_e \end{aligned} \tag{142}$$

4. The region was neither a neighbor of "a" nor "b", *e.g.* "f". Then there are no changes associated with this region.

The ssd surfaces associated with region "b" are no longer needed after the merge. We note that the memory requirements of the segmentation program decrease as the program progresses.

These updating rules mean that we evaluate the complicated affine parameters in Equation (137) only the minimum number of times required. We found that storing and updating the ssd surfaces made our segmentation program run much faster, to the point that we could experiment with the algorithm much more comfortably.

4.2.5 Nontextured Regions

Nontextured regions need special attention in our algorithm. Our peak-finding algorithm characterizes power spectrum patches in these types of regions by a single, d.c. peak at $(u, v) = (0, 0)$. Such patches contribute nothing to the surface normal computation, because d.c. peaks are not shifted by 3D effects. The ssd surface for a pair of matched, d.c. peaks is a constant zero. In addition, these patches can be included in any textured region at almost no cost in description length, because their d.c. peaks are compatible with any surface normal. To keep nontextured patches from being helplessly usurped into neighboring textured regions, we artificially create a minimum in their ssd surfaces at $(p, q) = (0, 0)$.

The magnitude of the artificial minimum is computed based on the ssd surfaces from pairs of textured patches. At the beginning our the image dendrogram, each patch has its own ssd surface. Since there is no way to compute a surface normal from a single patch, all these ssd surfaces are zero. There are also ssd surfaces computed between all pairs of adjacent patches. We find all the ssd surfaces that came from pairs of *textured* (non-d.c.) patches, and we compute the average difference between the minimum and maximum of these surfaces. The magnitude of the artificial minimum is five times this average.

4.2.6 Region Refinement

The final step in our segmentation algorithm is to refine the regions. After we pick the best level in the dendrogram, we give the regions to a program that perturbs the edges slightly to search for a better segmentation. The algorithm checks through the image in scanline order. When it encounters a patch on a region boundary, it reassigns that patch to the neighboring region(s). We recompute the surface normals, frontalized peaks, and description lengths of the affected regions. If the total description length for the entire image is lower with the reassignment, the change is preserved. Otherwise the patch is assigned back to its original region. We continue to scan through the image until we have made a scan through the whole image in which no patches are reassigned.

We have to take special precautions for regions of the image with no texture. Patches in these regions have their peak(s) at or near $(u, v) = (0, 0)$. Near-uniform patches can be convinced to take on almost any surface normal, since an affine transformation of a peak near zero is still a peak near zero. We count the number of patches in each region with singular peaks at $(u, v) = (0, 0)$. If these patches constitute more than a third of all the patches in the region, our refinement program will not reassign any patches in the region.

4.2.7 Summary of Algorithm

This section summarizes our segmentation algorithm. We start by computing the spectrogram of the image using 64x64 windows on 15-pixel centers. Our spectrogram preprocessor then finds up to six peaks in each power spectrum window. These are exactly the same as the steps we used in our peak-matching shape-from-texture algorithm in Section 3.7. We then compute and store the $\text{ssd}_{ij}(p, q)$ search spaces of all adjacent patches. We start building the dendrogram by considering each patch as its own region. Each region maintains a list of its neighboring regions. For each neighboring region, this list contains the decrease in description length that would result if these regions were merged. We compute the description length using $\sigma = 0.01$ cycles/pixel for the frequency peaks and Leclerc's $b = 2.0$. At each level of the dendrogram, we merge the two regions that give the greatest decrease in description length. Each merge requires updating of the new region and its neighbors. This updating goes fairly quickly, because we maintain $\text{ssd}(p, q)$ search spaces for all the regions and all the common borders between pairs of regions. We used a lower resolution search grid for the images in this section. Once the dendrogram is complete, we manually pick the best level and then refine the regions as described in the previous section. The magic numbers for this algorithm are shown in Table 12. The first 9 magic numbers are for the shape-from-texture part, and they are the same as the numbers for the peak-matching algorithm in Section 3.7, except for the lower-resolution search grid.

description	value(s)
window center spacing	15 pixels horizontally and vertically
window size	64x64 pixels
peak width for zeroing found peaks	1/16 cycles/pixel
minimum frequency	1/32 cycles/pixel
minimum (peak height)/(maximum peak height)	1/5
maximum number of peaks per patch	6
maximum distance between matched peaks	1/20 cycles/pixel
search grid resolution	40x40 or 60x60
search grid limits	$(\|p\|, \|q\|) \le (2, 2)$ or $(\|p\|, \|q\|) \le (3, 3)$
(σ_u, σ_v) of frequency peaks	$(0.01, 0.01)$ cycles/pixel
Leclerc's b for chain-code description length	2.0 bits/segment
nontextured region ssd minimum multiplier	5
number of regions in image	selected manually
fraction of uniform patches to be a uniform region	1/3

Table 12: All magic numbers for segmentation algorithm.

4.3 Results

We show results of our segmentation algorithm on 11 different images. Two of them were synthetically generated, four were taken in the Calibrated Imaging Laboratory at CMU, and five were taken outdoors in downtown Pittsburgh, PA. All the results use the same set of magic numbers, except we increased the range of the (p, q) search grid from $(|p|, |q|) \le (2, 2)$ for the synthetic and CIL images to $(|p|, |q|) \le (3, 3)$ for the outdoor images. We used a 40x40 search grid for the lower-range search and a 60x60 grid for the higher-range search. The CIL images were taken with the Photometrics camera described in Section 3.5.2. The outdoor images were taken with a Canon Xap Shot camera. This is a fairly low resolution, portable, still video camera. We describe its use and calibration in Appendix 6.

The results for each test image are shown on one page. The upper left shows selected levels of the image dendrogram, with different regions shaded differently. The dendrogram proceeds from the upper left, where each patch is a separate region, to the lower right, where the whole image is one region. The selected levels of the dendrograms is shown in Figure 71. The upper right of each page shows the edges associated with the dendrogram level that we manually chose as the best one. The middle left shows a needle diagram of the surface normals of the selected dendrogram level. The mid-

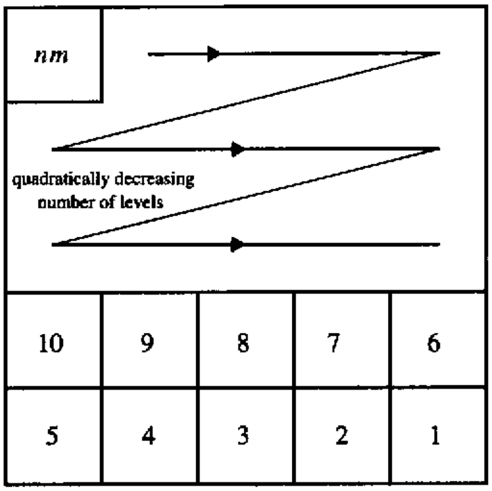
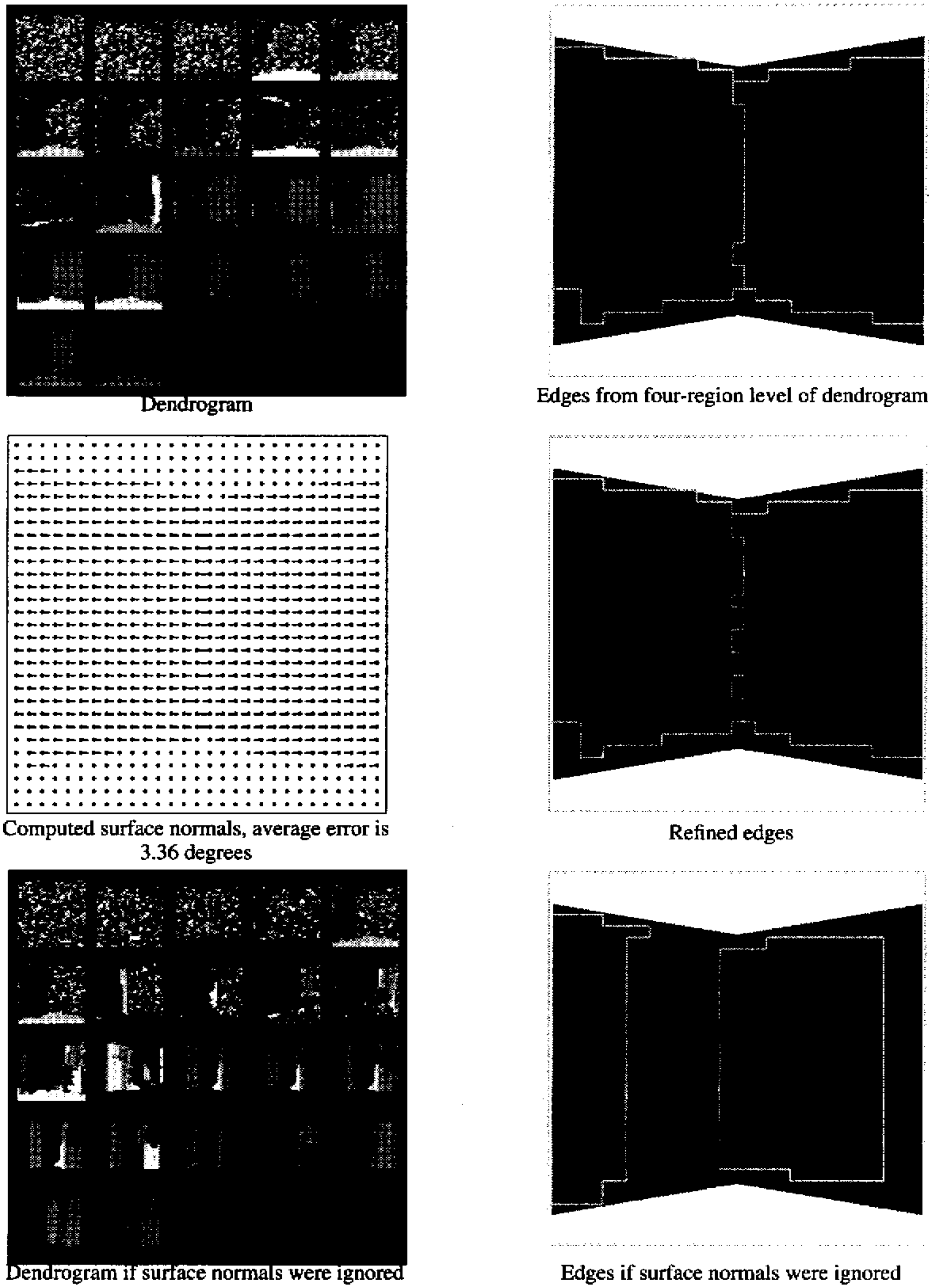


Figure 71: These are the "level numbers" of the dendrogram that we show in the segmentation results. The "level number" is the same as the number of regions. The segmentation starts with (n)x(m) regions.

dle right shows the edges of the regions after region refinement. The lower left shows another dendrogram, computed with the shape-from-texture module altered to return $(p, q) = (0, 0)$. This shows the effect of ignoring shape. The lower right shows the edges of what we consider to be the best level of the dendrogram computed by ignoring shape.

The captions give specific commentary on the results. In general, we see can get a good segmentation for all these images and the surface normals are fairly accurate. The edge refinement algorithm usually gives a slight improvement, with the most noticeable improvement for Figure 73. Except for two of the images, ignoring shape gives a dendrogram with no good levels. For the two exceptions (Figure 75 and Figure 76), at least one of the textures is close to frontal, so shape effects are minimized. The last CIL image, Figure 77, shows how the algorithm can discriminate the textured and untextured parts of a scene. This is also demonstrated in the outdoor images in Figure 72, Figure 72, and Figure 72. Figure 72 and Figure 72 show how our algorithm can segment identical textures with different surface normals.



Dendrogram

Edges from four-region level of dendrogram

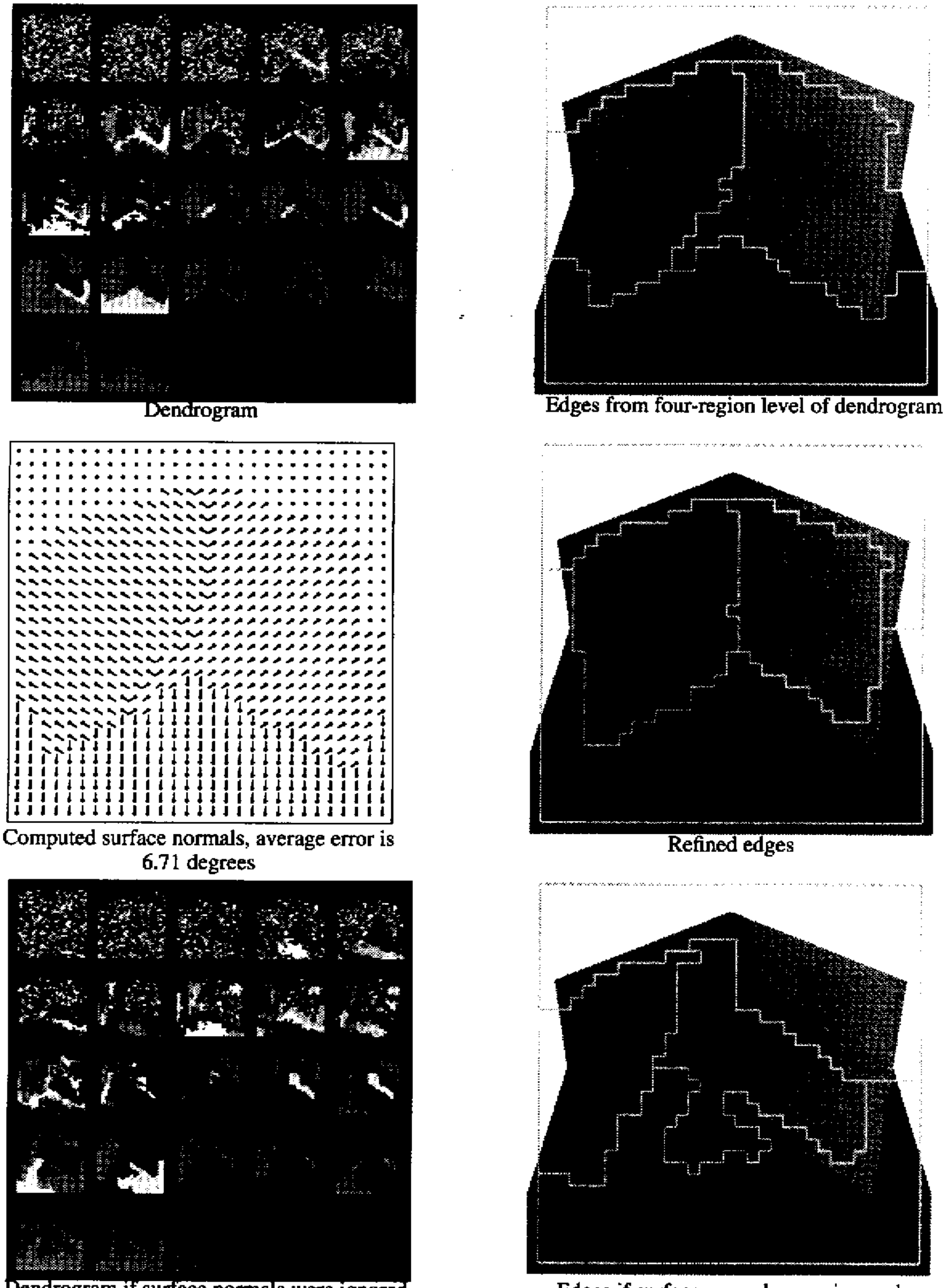
Computed surface normals, average error is 3.36 degrees

Refined edges

Dendrogram if surface normals were ignored

Edges if surface normals were ignored

Figure 72: Synthetic image with Brodatz textures "oriental straw cloth" (D53) and "cotton canvas" (D77). Ignoring the surface normals gives a dendrogram with no good segmentations.



Dendrogram

Edges from four-region level of dendrogram

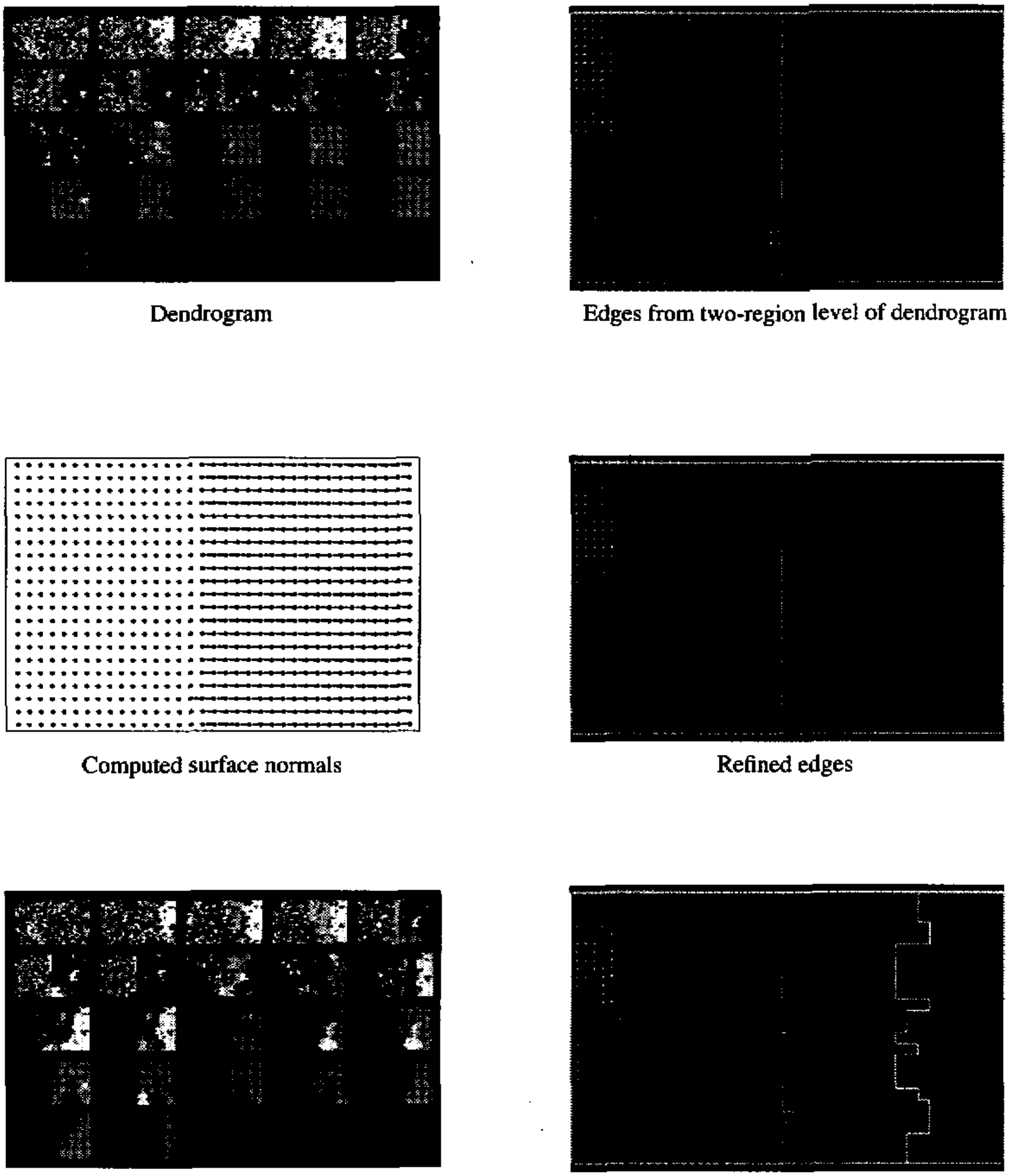
Computed surface normals, average error is 6.71 degrees

Refined edges

Dendrogram if surface normals were ignored

Edges if surface normals were ignored

Figure 73: Synthetic image with Brodatz textures "woven aluminum wire" (D6), "netting" (D34), and "cotton canvas" (D77). Edge refinement gives a noticeable improvement. Ignoring the surface normals gives a dendrogram with no good segmentations.



Dendrogram

Edges from two-region level of dendrogram

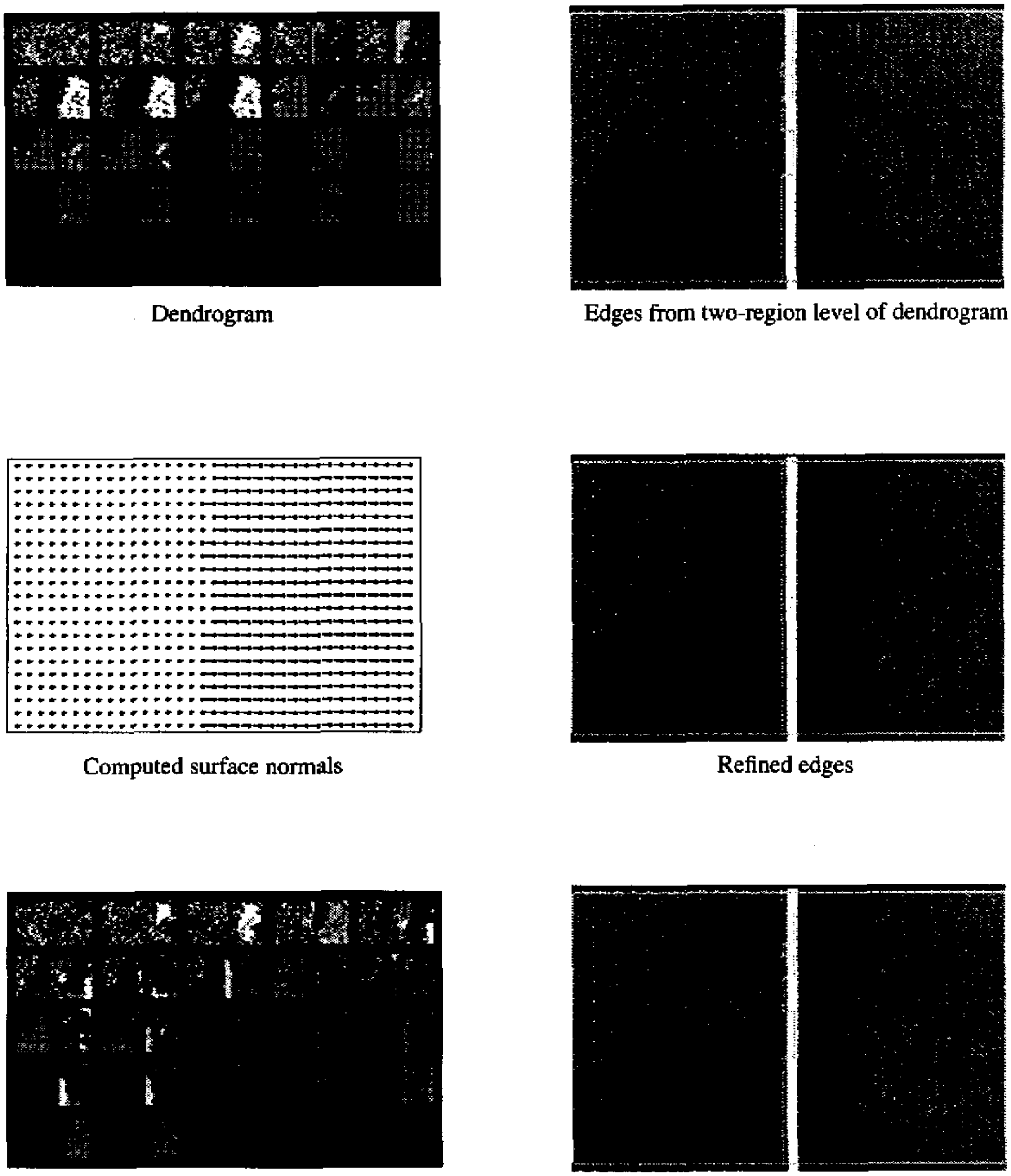
Computed surface normals

Refined edges

Dendrogram if surface normals were ignored

Edges if surface normals were ignored

Figure 74: CIL image with egg crate pattern light cover and doormat. We did not measure ground truth for the surface normals. Ignoring the surface normals gives a dendrogram with no good segmentations.



Dendrogram

Edges from two-region level of dendrogram

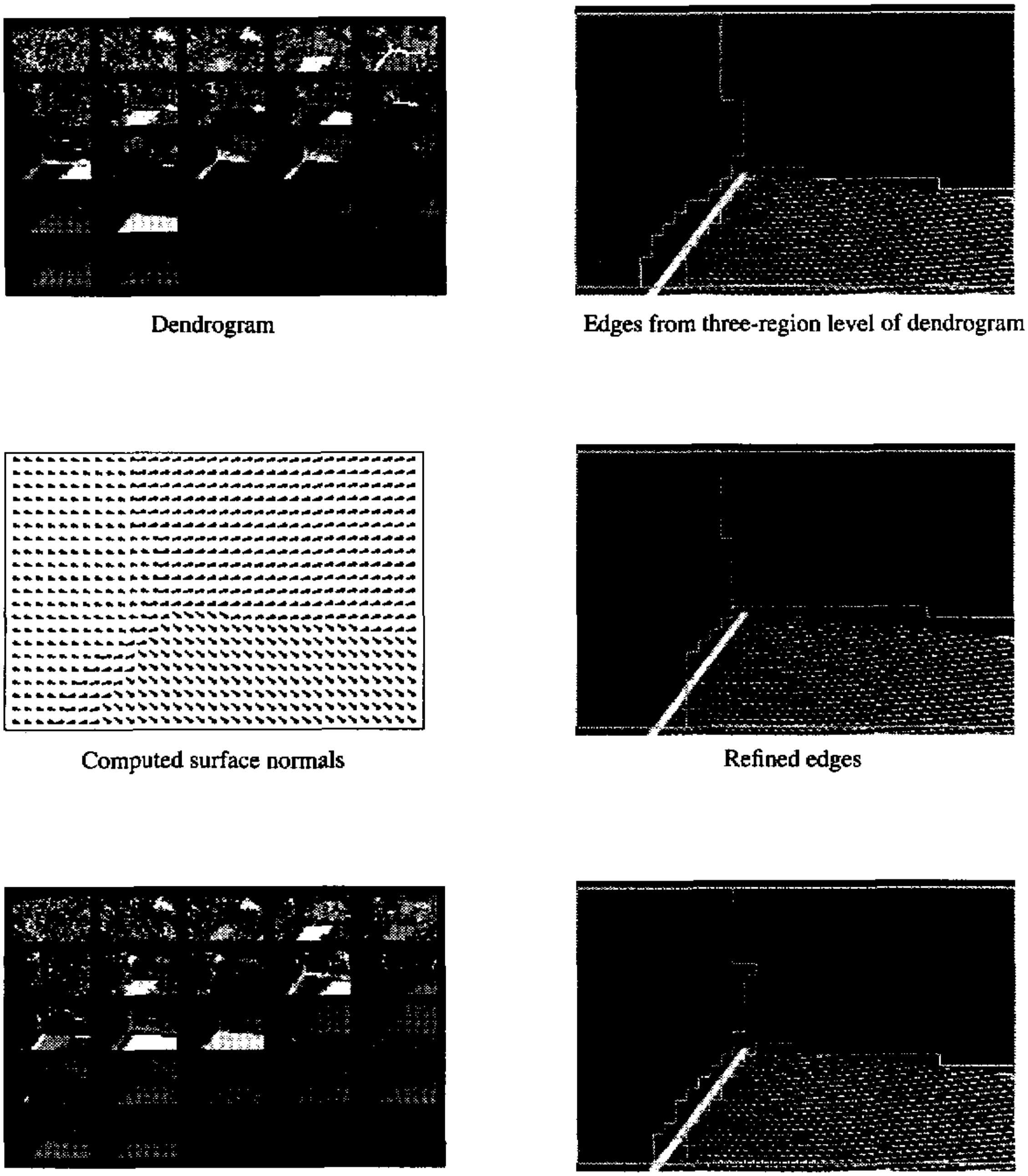
Computed surface normals

Refined edges

Dendrogram if surface normals were ignored

Edges if surface normals were ignored

Figure 75: CIL image with screen guard and doormat. We did not measure ground truth for the surface normals. We can get a good segmentation even if we ignore the surface normals, partly because the screen guard is almost frontal.



Dendrogram

Edges from three-region level of dendrogram

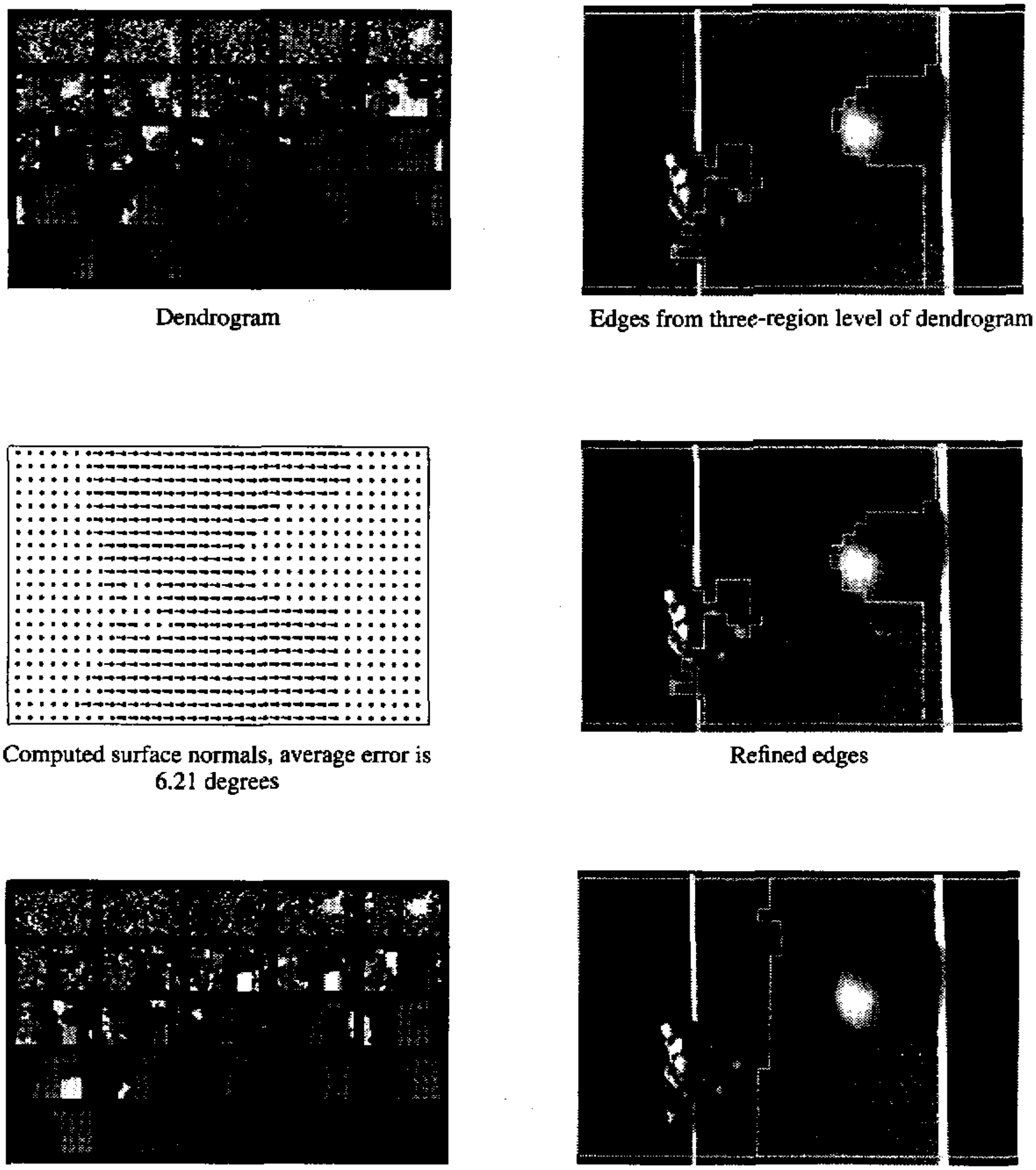
Computed surface normals

Refined edges

Dendrogram if surface normals were ignored

Edges if surface normals were ignored

Figure 76: CIL image with bumpy light cover, doormat and screen guard. We did not measure ground truth for the surface normals. We can get a good segmentation even if we ignore the surface normals, because the textures are almost frontal.



Dendrogram

Edges from three-region level of dendrogram

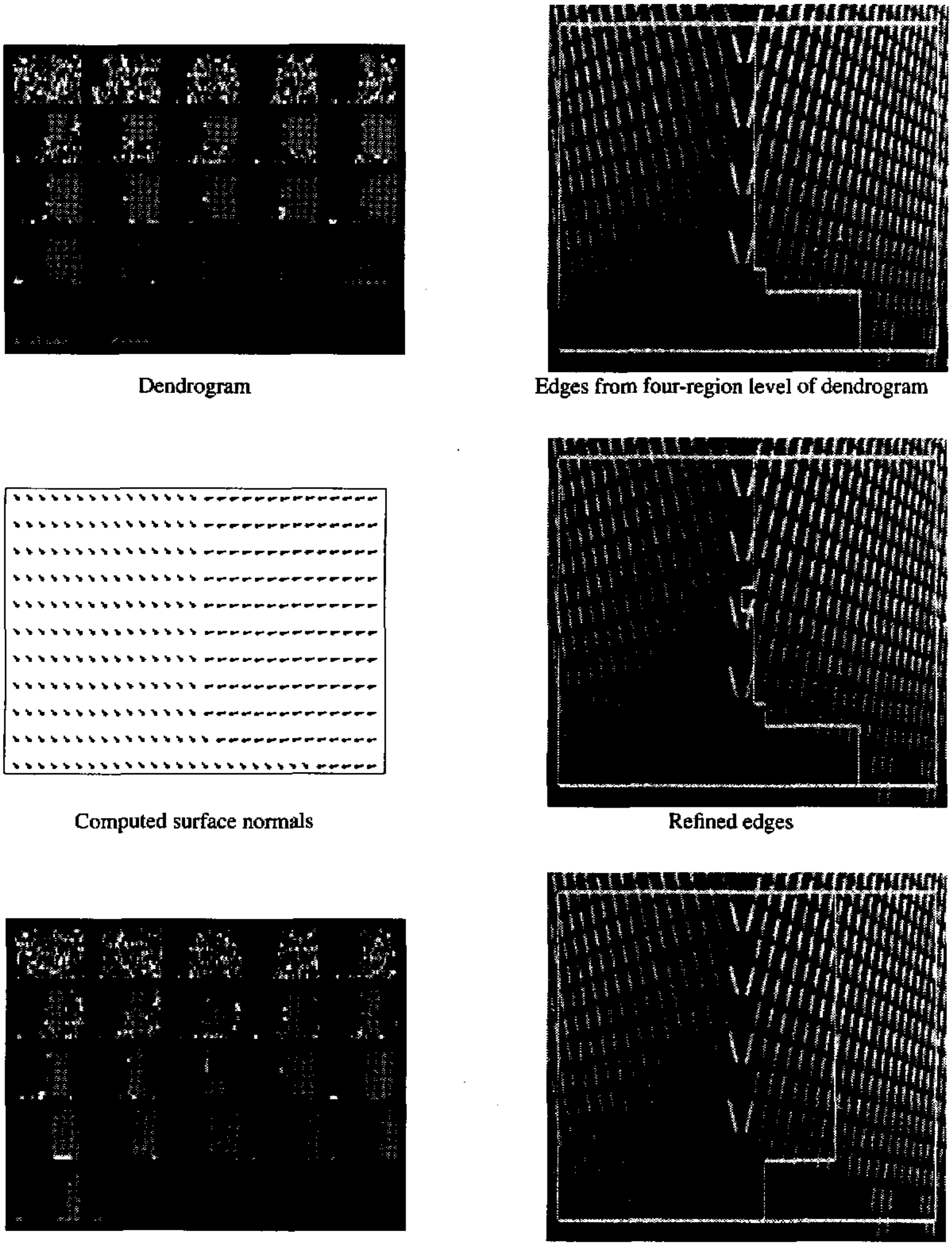
Computed surface normals, average error is 6.21 degrees

Refined edges

Dendrogram if surface normals were ignored

Edges if surface normals were ignored

Figure 77: CIL image of screen guard and other objects. Our program can distinguish the textured and untextured regions of an image. Ignoring the surface normals gives a dendrogram with no good segmentations.



Dendrogram

Edges from four-region level of dendrogram

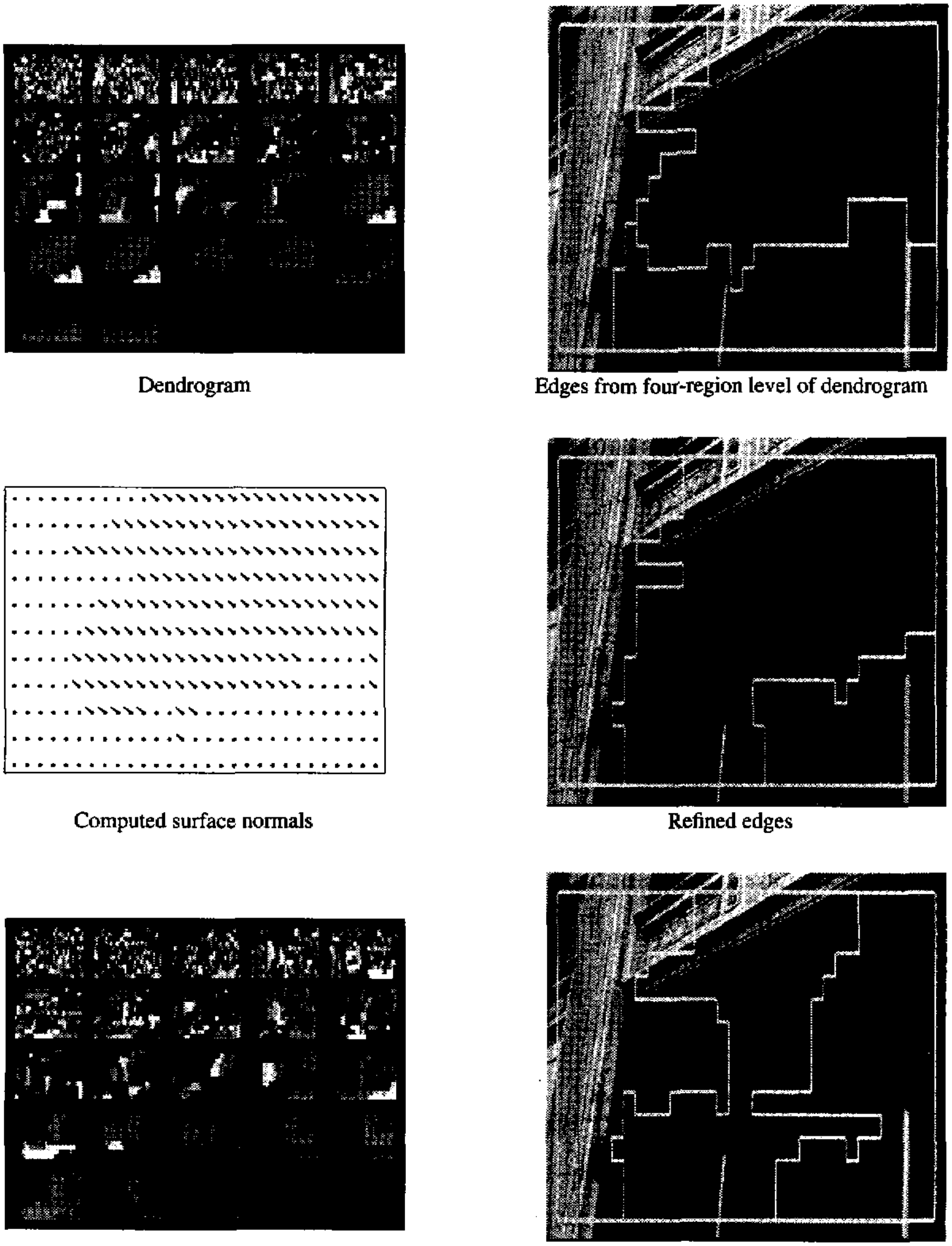
Computed surface normals

Refined edges

Dendrogram if surface normals were ignored

Edges if surface normals were ignored

Figure 78: This is a corner of the U.S. Steelworkers Building in downtown Pittsburgh, PA. Our program finds the corner. Ignoring the surface normals gives a dendrogram with no good segmentations.



Dendrogram

Edges from four-region level of dendrogram

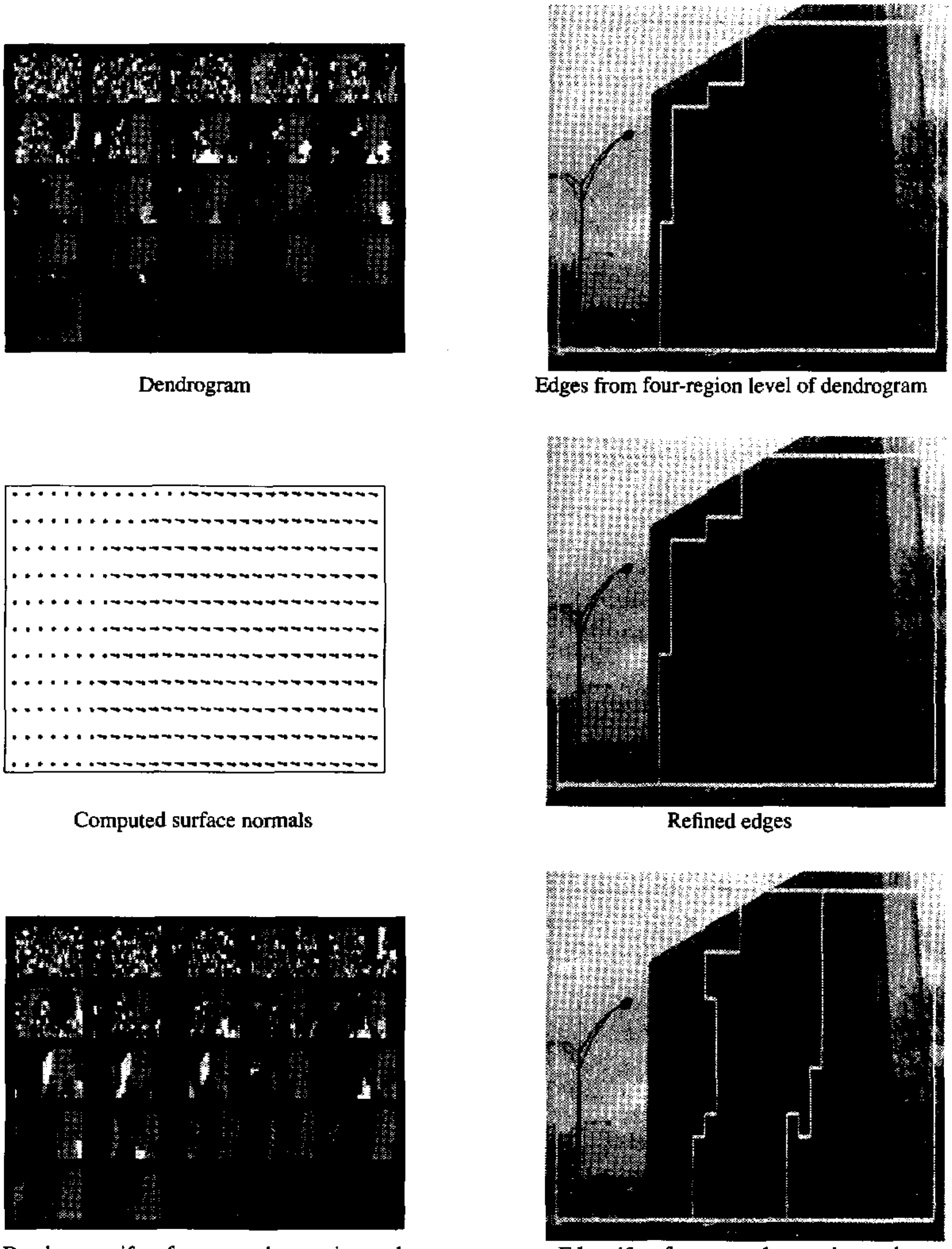
Computed surface normals

Refined edges

Dendrogram if surface normals were ignored

Edges if surface normals were ignored

Figure 79: Air vents on a building in downtown Pittsburgh, PA. Refining the edges makes them worse in this case. Ignoring the surface normals gives a dendrogram with no good segmentations.



Dendrogram

Edges from four-region level of dendrogram

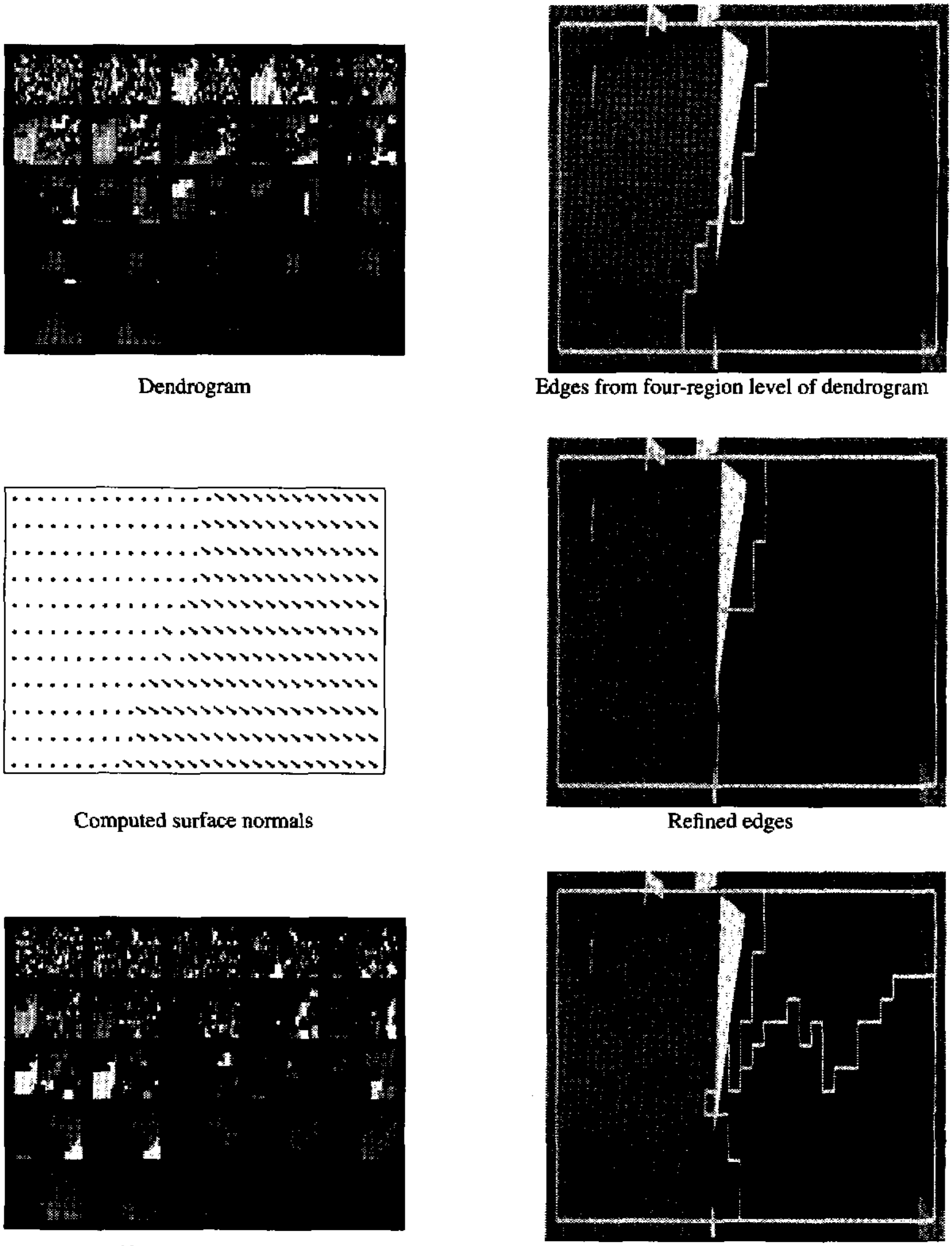
Computed surface normals

Refined edges

Dendrogram if surface normals were ignored

Edges if surface normals were ignored

Figure 80: This is a large apartment building in Pittsburgh, PA. Our program separates the texture from the non-texture. Ignoring the surface normals gives a dendrogram with no good segmentations.



Dendrogram

Edges from four-region level of dendrogram

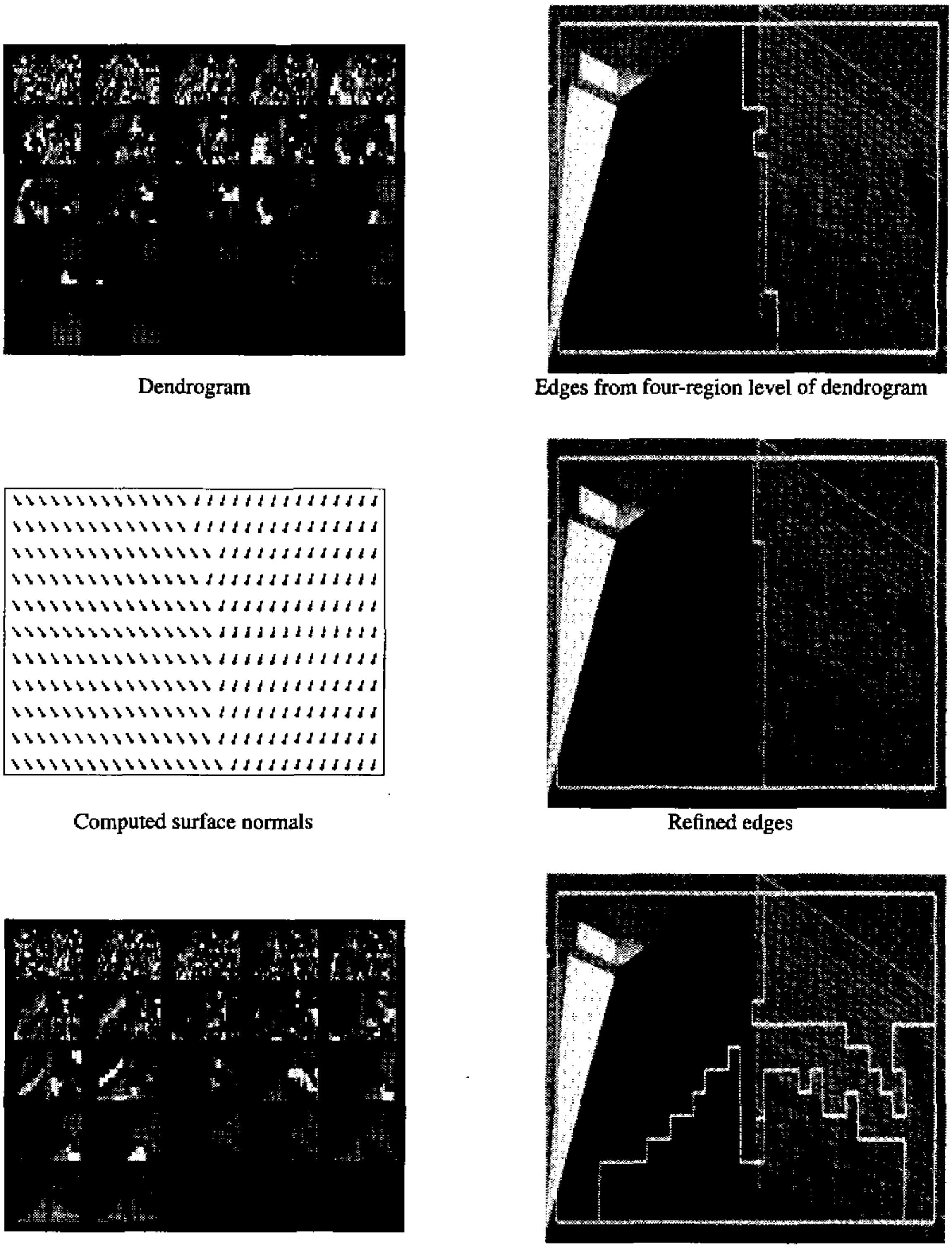
Computed surface normals

Refined edges

Dendrogram if surface normals were ignored

Edges if surface normals were ignored

Figure 81: The region to the left of this shingled roof is considered non-textured by our program. Ignoring the surface normals gives a dendrogram with no good segmentations.



Dendrogram

Edges from four-region level of dendrogram

Computed surface normals

Refined edges

Dendrogram if surface normals were ignored

Edges if surface normals were ignored

Figure 82: This is the same scene as in Figure 72, but with the camera pointing more to the right showing two shingle-covered surfaces. Edge refinement improves the result slightly. Ignoring the surface normals gives a dendrogram with no good segmentations.

4.4 Discussion

There are several things we like about our segmentation algorithm. Most important, to our knowledge, it is the only texture segmentation algorithm that explicitly accounts for 3D shape. And while we cannot claim this is a necessary prerequisite for segmenting images of textured, 3D objects, it is at least important for our algorithm, because it usually fails when we turn off the shape-from-texture module. It is reasonable to believe that accounting for shape is the right thing to do for all general image segmentation algorithms that depend on measures that are affected by 3D shape.

The the dendrogram and minimum description length criterion combine to make a segmentation algorithm that is fairly simple and does not require many magic numbers. It is much simpler than some of our earlier attempts at solving the same problem[59]. The dendrogram, by its nature, begins with a tight threshold on what can be merged and then loosens the threshold just enough to keep merging. The minimum description length principle lets us neatly avoid complex questions of how to evaluate combinations of disparate numbers of peaks. We use it not only to find the best clustering of peaks, but also to evaluate the compactness of the clustering. The description length taken over the whole image is an effective merge criterion for the dendrogram. It is also useful for our refinement step.

There are at least three ways this algorithm could be improved with a reasonable amount of effort. First, it would be good if the description length criterion could signal the correct level in the image dendrogram. Ideally, we would like to stop merging after we find no merges that would decrease the total description length, or at least take the segmentation with the minimum description length. Unfortunately, both these points come much before the right level. Figure 83 show description length as a function of the number of regions for the dendrogram in Figure 73. The right number of regions is four, but the minimum description length occurs at 381 regions. The description length of the chaincoded borders behaves as we expect: decreasing as the number of regions gets smaller. This is because larger and larger regions are being merged, meaning more and more bits are saved by not having to account for their common borders. The problem is with the description length of the clustered peaks. It starts rising too soon. This could be caused by merging patches that overlap into two or more textures. It may also be that our estimate of the standard deviation of the peaks (0.01 cycles/pixel) is too low, giving too much weight to the deviations between the cluster centers and the peaks. Although the description length criteria provides an effective way of ranking potential merges, it is not yet up to the task of evaluating them on an absolute scale. Even Rissanen, the father of description length, continues to refine his formulation. Ours also needs refining.

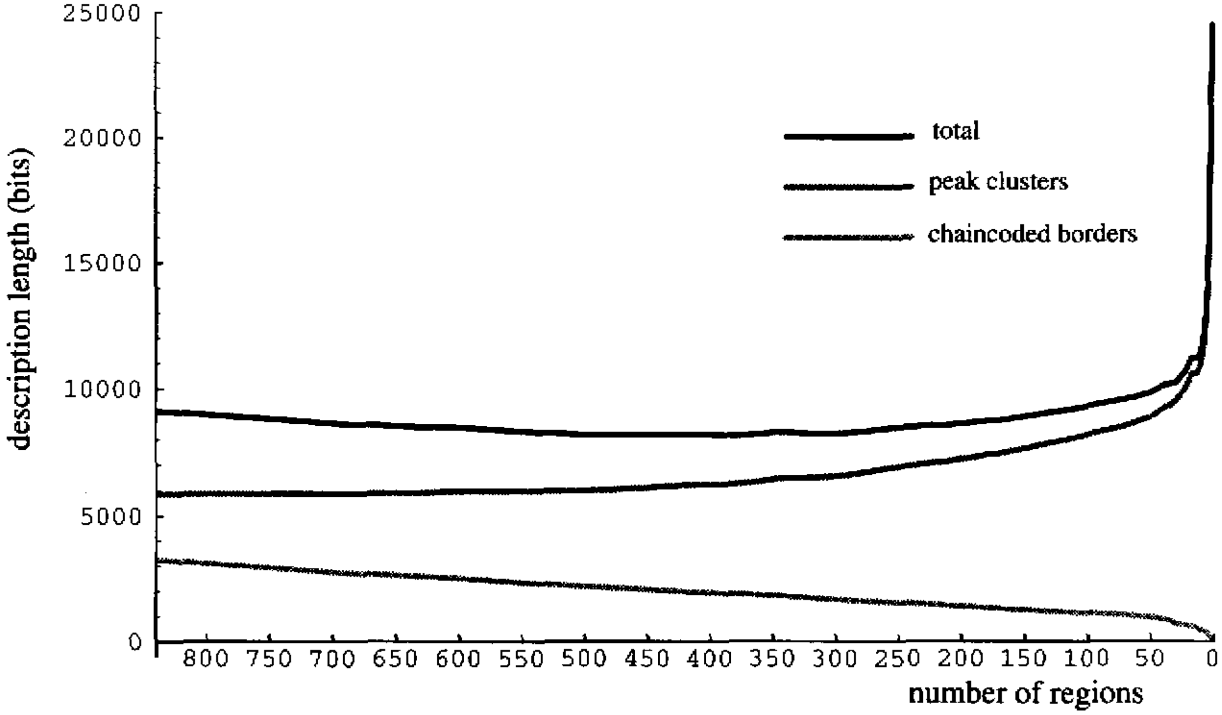


Figure 83: This shows the description length as a function of the number of regions for the image of three textured plates in Figure 73. The minimum in the total description length occurs at 381 regions, whereas we would like it to occur at four.

The other two improvements we have already discussed in Chapter 3. It would be good if the algorithm could work on curved surfaces and on random textures. Conceptually, all we need for curved surfaces is a 3D shape representation that would allow us to frontalize all the patches in a region to a common depth. A list of local surface normals would be the most general representation, but it would not be as robust as a parameterized surface where all the patches contribute to the coefficients of a surface equation. Dealing with random textures could also give noisy surface normal estimates, but this too could probably be solved with parameterized surfaces. Although random textures cannot be represented by peaks in their power spectra, the dendrogram, frontalization, and description length concepts could be adapted to account for more general textures. One potential problem is that small regions of random texture may give only very uncertain surface normals.

Chapter 5 Where We've Been and Where We Could Go

5.1 What We Have Shown

The goal of this thesis was to show how the space/frequency representation, specifically the spectrogram, is useful for analyzing image texture. We began in Chapter 1 by reviewing all the major models for image texture that have been used in computer vision. These were first-order statistics, cooccurrence matrices, gray level run length matrices, autocorrelation, Markov random fields, fractal Brownian surfaces, random mosaics, low-level features, high-level features, and local frequency filters. Of all these models, only local frequency filters (the space/frequency representation) and Markov random fields have been use for all three tasks of texture classification, texture segmentation, and shape-from-texture. We went on to show in pictures how the space/frequency representation elucidates several different image phenomena. These phenomena include texture boundaries, texture in perspective, aliasing, zoom, and blur. The common characteristic of these phenomena is that they all show some kind of spatial frequency phenomena that changes from point to point in the image. Thus they are perfectly suited for analysis with the space/frequency representation.

Having shown the general usefulness of the space/frequency representation for image texture analysis, we discussed in Chapter 2 the important issues in computing the space/frequency representation for our application. Although any algorithm for computing the representation will suffer from inherent resolution limits and problems with nonstationary signals, we can still do well enough for our purposes. We considered all the top contenders in the pool of algorithms for computing the space/frequency representation. These were instantaneous frequency, the short-time Fourier transform and spectrogram, general frequency filters, wavelets, the Wigner distribution, adaptive filters, and specially tuned filters. For our application of general image texture understanding, the spectrogram was the best, and we used it for the remainder of the thesis.

Our first all-out effort at using the spectrogram was an algorithm for shape-from-texture. Based on our mathematical analysis of frequency shifting due to 3D effects, we developed three related shape-from-texture algorithms in Chapter 3. The first two algorithms, ssd patch-matching and differential patch-matching, work directly on the spectrogram data and require only a few magic numbers. While the spectrogram is a low-level representation of the image data, these algorithms showed that it can be used to directly estimate high-level scene parameters, without going through potentially unreliable feature-detection in between. Our third algorithm, peak-matching, was based on finding and matching peaks in the spectrograms of periodic textures. This gave use a fast algorithm for estimating local surface normals, which is what we needed for our segmentation algorithm in the next chapter.

We showed how to unify the solutions to two major texture problems in Chapter 4. Until now, texture segmentation and shape-from-texture were two independent problems, with mutually exclusive assumptions that prevented their being reliably applied to images of 3D textured surfaces. Using our peak-matching shape-from-texture algorithm, we showed how to segment such images by making tentative hypotheses about the surface normal and frontal texture in small regions of the image. We grow the hypotheses using a minimum description length merge criteria, and demonstrated the algorithm on images of real texture. This algorithm shows how the space/frequency representation can indeed be used to unify formerly disparate tasks in computer vision, and it solves a problem that had not yet been solved.

5.2 What We Know Now That We Didn't Know Before

This is a list of new research results that we developed in this thesis.

- Shape-from-Texture Theory
    - We developed the mathematical relationship between the 3D orientation of a textured surface in a scene and the local Fourier power spectrum of its perspective projection.
    - We used the relationship above to show that the connection between the Fourier

power spectra of any two image regions from the same texture in the scene are approximately related by an affine transformation. The four affine parameters are functions of the 3D location and orientation of the corresponding surfaces in the scene and the camera parameters. If we assume the two scene points are on the same plane, then the only unknown is the plane's surface normal.

  - We developed a method of analytically predicting the relative variance of the computed surface normals using the Hessian of the ssd surface at the minimum point. Our predictions were verified with experiments.

- Shape-from-Texture Algorithms

  - SSD Patch-Matching: This is a simple algorithm based the power spectra of two pre-specified points in the image. If we assume that the two points are from the same textured plane, then we can search the space of surface orientations for the one that gives the best affine transform of one power spectrum into the other. The algorithm requires no feature detection and few magic numbers. It works on periodic and random textures. In simple cases it works in spite of aliasing.

  - Differential Patch-Matching: This is a simple algorithm based on nearby power spectrum patches. By directly computing the affine transformation between neighboring patches, we can avoid having to affine transform one power spectrum onto another. We search through the space of surface normals for the one that gives the computed affine parameters. This gives us local surface normal estimates, so it works for curved as well as flat textured surfaces.

  - Peak-Matching: For periodic textures, peaks preserve everything we need to know about the power spectrum. We search through the space of surface normals to find the one that best accounts for the peak shifts between two patches. This algorithm is good for quickly estimating local surface normals of periodic textures. We use it as the basis of our segmentation algorithm.

- Algorithm for Segmenting 3D Textured Surfaces

  - We use our peak-matching algorithm for estimating local surface normals. Based on these estimates, we show how to compute where the frequency peaks would be if the texture were viewed from the front.

  - We developed a merge criterion based on the minimum description length principle. We use the description length to measure how well the frontalized peaks in a region cluster together. Our description length formula is a conventional one with an extra term added for labeling the points with their cluster numbers. This term appears to give a sharper minimum when evaluating the description length for different numbers of clusters.

  - We developed a fast algorithm for computing the surface normal of combined regions. We maintain in memory the (p, q) search spaces for all the current regions in the image. When we combine regions, we simply sum their search spaces and take the location of the new minimum as the surface normal. This minimizes redundant computation.

5.3 Good Things to Do Next

We left off with solving the problem of segmenting textured 3D surfaces using the spectrogram. We assumed that the textures were periodic and the surfaces were flat. Starting at this point, there are several new things to try.

- **More General Scenes:** We would like our shape-from-texture and segmentation algorithms to work better on curved surfaces and irregular textures. Although we can compute local surface normals of curved surfaces, we cannot segment curved surfaces. For this task, it would probably be best to use a parameterization of the surface to help smooth errors in the surface normals and to evaluate potential merges.

  Our ssd patch-matching algorithm works on irregular textures, but it is slow and requires a large region of texture. If we are to segment irregularly textured 3D surfaces, we need to find local surface normals based on small regions of irregular texture.

- **Better Adaptation to Input Data:** Jones and Parks[47] showed how to adaptively change the spectrogram window size to achieve a higher resolution space/frequency representation. For stereo, Okutomi and Kanade[77] showed how to optimize the matching window size to get the best disparity estimates. For our application, the window size and shape could be optimized to give the best surface normal estimate.

  Super[99] has a good idea in using spatial filters tuned to different surface orientations. This deserves more attention. It would be interesting to try this idea on irregular textures.

  There is some limit to how accurate any shape-from-texture algorithm can be. A comprehensive sensitivity analysis could take into account all factors, including the randomness of the texture, and then predict the accuracy and give the best parameters (*e.g.* window size) to use to get the surface normal.

- **More Phenomena Integrated:** The space/frequency representation is a good way to reason about aliasing and blur, as we showed. It should be possible to integrate such reasoning into our segmentation algorithm to let it handle these phenomena. And while aliasing and blur are usually considered an annoyance, there is a thriving literature in how to exploit them to compute surface orientation and depth, respectively. These could be used to aid the segmentation.

- **Better Segmentation:** We compute our spectrogram on 15-pixel centers, so the edges of our segmentation are blocky. It would be good to refine these edges using special reasoning about texture boundaries. This could involve computing new power spectra at a smaller pixel spacing near the edges, computing power spectra with smaller windows, or detecting and eliminating spectra that overlap into two or more distinct textures.

Our description length formula proved effective for finding the right regions to merge for segmentation. Ideally, it would also tell us when to stop merging by indicating there are no more merges that would reduce the description length of the image.

5.4 The Future of Space/Frequency in Computer Vision

This thesis has shown how the space/frequency representation is useful for analyzing many image phenomena. The representation's versatility is evidenced by the wide variety of computer vision problems it has been used to solve, as shown along the left side of Figure 84. All this research has used either the space/frequency representation or the Fourier transform of the whole image. (Presumably, if the problem can be solved with the Fourier transform of the whole image, it can be solved locally with the space/frequency representation.) Given that the representation is useful for all these problems by themselves, it should be useful for solving the problems in combination. For instance, our earlier work in moire patterns[57] was based in the frequency domain, and this meant we were prepared to account for aliasing in the ssd patch-matching algorithm for shape-from-texture that we presented in Section 3.5.3. Likewise, we showed how to solve the combined problem if segmentation and shape from texture in Chapter 4. The common representation makes it much easier to formulate the equations and implement the algorithm. Since so many other algorithms are based on the same representation, we predict a gradual unification of all these algorithms in terms of the space/frequency representation. We give this final theory the grand title of "The Unified Theory of Spatial Vision".

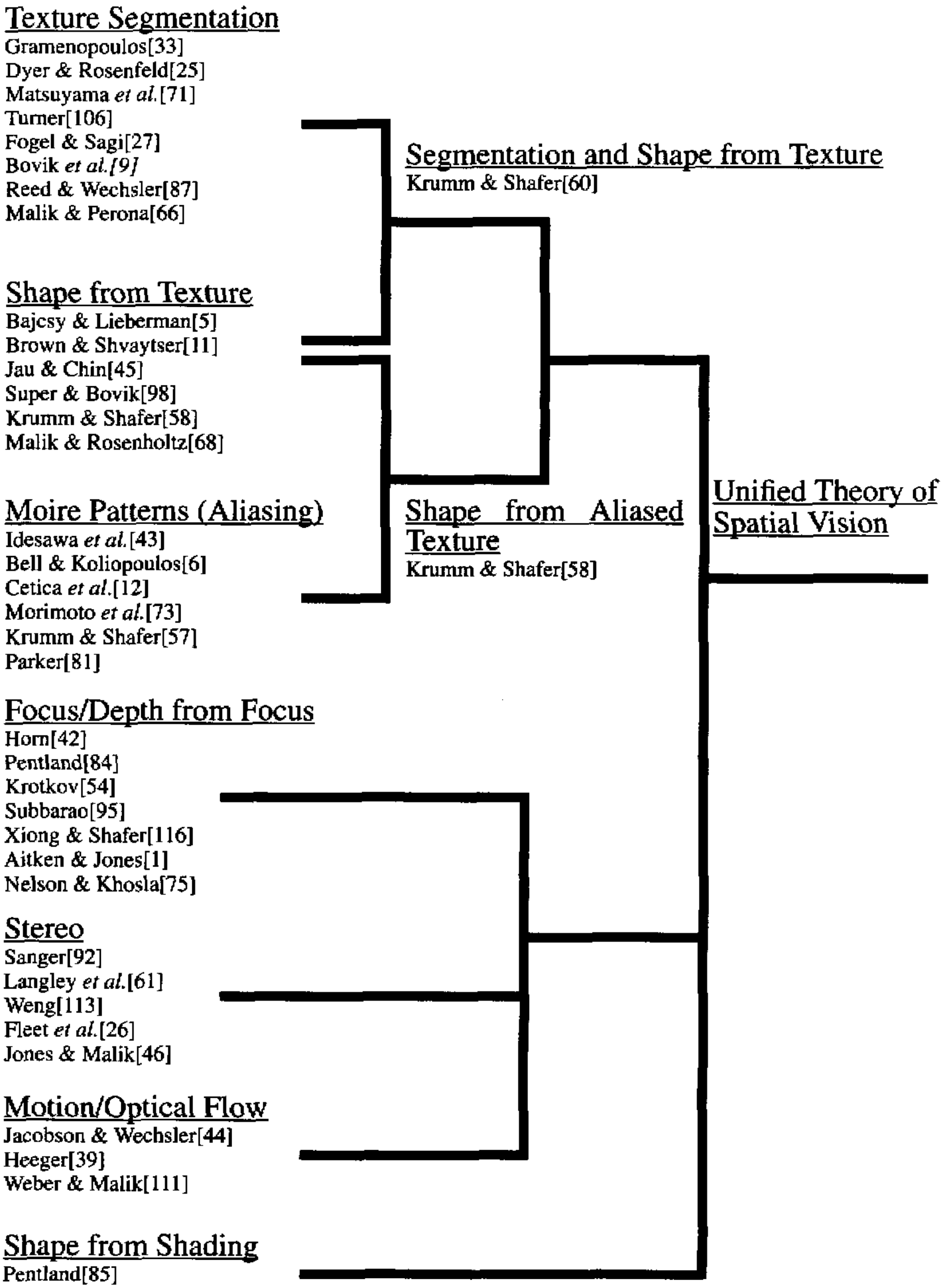


Figure 84: This work in computer vision has used spatial frequency or local spatial frequency representations, and indicates that many different algorithms can be unified because of their common representation.

Appendix 1 The Spectrogram of a 1D Fourier Series

The Fourier series is

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi n u_0 x}. \tag{A1 1}$$

If the window function is $w(x)$, then the spectrogram is

$$\begin{aligned}
S(x,u) &= \left|\Im_{\alpha \Rightarrow u}\{w(\alpha - x) f(\alpha)\}\right|^2 \\
&= \left|\Im_{\alpha \Rightarrow u}\left\{w(\alpha - x) \sum_{n=-\infty}^{\infty} c_n e^{j2\pi n u_0 \alpha}\right\}\right|^2 \\
&= \left|[e^{-j2\pi u x} W(u)] * \sum_{n=-\infty}^{\infty} c_n \delta(u - n u_0)\right|^2 \\
&= \left|\sum_{n=-\infty}^{\infty} c_n e^{-j2\pi (u - n u_0) x} W(u - n u_0)\right|^2 \\
&= \left[\sum_{n=-\infty}^{\infty} c_n e^{-j2\pi (u - n u_0) x} W(u - n u_0)\right]\left[\sum_{m=-\infty}^{\infty} c_m{}^{*} e^{j2\pi (u - m u_0) x} W^{*}(u - m u_0)\right]
\end{aligned} \tag{2}$$

The last equation, a product of two sums, can be thought of as a sum of pairwise products in a matrix indexed by m and n, as shown below.

n \ m	...	-2	-1	0	1	2	...
...	■	▓	▓	▓	▓	▓	▓
-2	░	■	▓	▓	▓	▓	▓
-1	░	░	■	▓	▓	▓	▓
0	░	░	░	■	▓	▓	▓
1	░	░	░	░	■	▓	▓
2	░	░	░	░	░	■	▓
...	░	░	░	░	░	░	■

The terms on the diagonal are the auto-terms, corresponding to $m = n$, while the off-diagonal elements are the undesirable cross-terms. If we separate the auto-terms and cross-terms, the spectrogram becomes

$$S(x,u) = \sum_{r=-\infty}^{\infty} |c_r|^2 |W(u - ru_0)|^2 + \sum_{\substack{n=-\infty \\ m \neq n}}^{\infty} \sum_{m=-\infty}^{\infty} c_n c_m^* e^{j2\pi x u_0 (n-m)} W^* (u - mu_0) W(u - nu_0) \tag{A1 3}$$

Every term in the lower triangle can be summed with a corresponding complex conjugate term in the upper triangle. This gives the final expression:

$$S(x,u) = \sum_{r=-\infty}^{\infty} |c_r|^2 |W(u - ru_0)|^2 + 2 \sum_{n=-\infty}^{\infty} \sum_{m=n+1}^{\infty} \mathrm{Re}\left[c_n c_m^* e^{j2\pi x u_0 (n-m)} W^* (u - mu_0) W(u - nu_0)\right] \tag{A1 4}$$

Appendix 2 The Wigner Distribution of a 1D Fourier Series

The Fourier series is

$$f(x) = \sum_{n=-\infty}^{\infty} c_n e^{j2\pi n u_0 x}. \tag{A2 1}$$

Boashash[8] recommends using the analytic version of the function for the Wigner distribution. This can be obtained by zeroing the amplitudes of all the negative frequency components, doubling the amplitudes of the positive frequency components, and doing nothing to the d.c. component. The analytic version of $f(x)$ is then

$$f_a(x) = \sum_{n=0}^{\infty} a_n e^{j2\pi n u_0 x} \tag{A2 2}$$

where

$$\begin{aligned} a_0 &= c_0 \\ a_n &= 2c_n \text{ for } n \neq 0 \end{aligned} \tag{A2 3}$$

The Wigner distribution is then

$$\begin{aligned} \mathrm{WD}\,(x, u) &= \Im_{\alpha \Rightarrow u}\{f(x+\tfrac{\alpha}{2}) f^*\,(x-\tfrac{\alpha}{2})\} \\ &= \Im_{\alpha \Rightarrow u}\left\{\left(\sum_{n=0}^{\infty} a_n e^{j2\pi n u_0 (x+\frac{\alpha}{2})}\right)\left(\sum_{m=0}^{\infty} a_m{}^* e^{-j2\pi m u_0 (x-\frac{\alpha}{2})}\right)\right\} \\ &= \Im_{\alpha \Rightarrow u}\left\{\sum_{n=0}^{\infty} \sum_{m=0}^{\infty} a_n a_m{}^* e^{j2\pi u_0 x(n-m)} e^{j2\pi u_0 \frac{\alpha}{2}(n+m)}\right\} \end{aligned} \tag{A2 4}$$

Grouping the product pairs into auto- and cross-terms, like we did in Appendix 1, we get

$$\mathrm{WD}\,(x, u) \;=\; \Im_{\alpha \Rightarrow u}\left\{\sum_{r=0}^{\infty} \left|a_r\right|^2 e^{j2\pi r u_0 \alpha} + \sum_{\substack{n=0\\ m \neq n}}^{\infty} \; \sum_{n=0}^{\infty} a_n a_m{}^{*} e^{j2\pi u_0 x\,(n-m)}\, e^{j2\pi u_0 \frac{\alpha}{2}\,(n+m)}\right\} \tag{A2 5}$$

Summing the complex conjugate terms in the double sum gives

$$\begin{aligned}\mathrm{WD}\,(x, u) \;=\; \Im_{\alpha \Rightarrow u}\Bigg\{ & \sum_{r=0}^{\infty} \left|a_r\right|^2 e^{j2\pi r u_0 \alpha} + \\ & 2\sum_{n=0}^{\infty} \; \sum_{m=n+1}^{\infty} \mathrm{Re}\,\left(a_n a_m{}^{*} e^{j2\pi u_0 x\,(n-m)}\right) e^{j2\pi u_0 \frac{\alpha}{2}\,(n+m)} \Bigg\}\end{aligned} \tag{A2 6}$$

Taking the Fourier transform gives the final expression:

$$\begin{aligned}\mathrm{WD}\,(x, u) \;=\; & \sum_{r=0}^{\infty} \left|a_r\right|^2 \left(\delta\,(u - r u_0)\right) + \\ & 2\sum_{n=0}^{\infty} \; \sum_{m=n+1}^{\infty} \mathrm{Re}\,\left(a_n a_m{}^{*} e^{j2\pi u_0 x\,(n-m)}\right) \delta\,(u - \frac{n+m}{2} u_0)\end{aligned} \tag{A2 7}$$

Appendix 3 The Spectrogram of a 2D Skew-Periodic Function

Our peak-matching algorithm for shape-from-texture (Section 3.7) is based on the assumption that the texture is periodic. This lets us assume the spectrogram is composed of peaks. A 2D periodic function, strictly speaking, is arranged on a uniform rectangular lattice. A "skew-periodic" function[32] is more general in that it is arranged on a uniformly skewed rectangular lattice. Periodic functions are a special case of skew-periodic functions. This appendix derives the spectrogram of a 2D skew-periodic function, showing that it consists of peaks. Therefore, the peak-matching algorithm applies to not only strictly periodic textures, but to the broader class of skew-periodic textures.

We can write a 2D skew-periodic function as a sum of regularly spaced elementary functions[117]. That is

$$f(x, y) = \frac{1}{|D|} \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} c\left(x - \frac{b_2}{D} n - \frac{b_1}{D} m, y - \frac{a_2}{D} n - \frac{a_1}{D} m\right)$$

$$= c(x, y) * \frac{1}{|D|} \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \delta\left(x - \frac{b_2}{D} n - \frac{b_1}{D} m, y - \frac{a_2}{D} n - \frac{a_1}{D} m\right) \qquad \text{(A3 1)}$$

where $D = a_1 b_2 - a_2 b_1$. We have written the skew coefficients in this strange way to make the resulting spectrogram simpler. The double-sum of delta functions represents the nodes of a skewed lattice. A strictly periodic function would have $b_1 = a_2 = 0$ or $b_2 = a_1 = 0$.

The major part of this derivation involves the Fourier transform of a skew-periodic function. We took this from Gaskill[32], p. 310. .If $w(x, y)$ is the window function, then the spectrogram is

$$
\begin{aligned}
S(x, y, u, v) &= \left|\Im_{(\alpha,\beta) \Rightarrow (u,v)} \{ w(\alpha - x, \beta - y) f(x, y) \}\right|^2 \\
&= \left|\Im_{(\alpha,\beta) \Rightarrow (u,v)} \left\{ w(\alpha - x, \beta - y) \left[c(x, y) * \frac{1}{|D|} \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \delta\left(x - \frac{b_2}{D} n - \frac{b_1}{D} m, y - \frac{a_2}{D} n - \frac{a_1}{D} m\right) \right] \right\}\right|^2 \\
&= \left| e^{j2\pi(ux + vy)} W(u, v) * \left[C(u, v) \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \delta(u - a_1 n - a_2 m, v - b_1 n - b_2 m) \right] \right|^2 \\
&= \left| e^{j2\pi(ux + vy)} W(u, v) * \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} c_{mn} \delta(u - a_1 n - a_2 m, v - b_1 n - b_2 m) \right|^2
\end{aligned}
\tag{A3 2}
$$

where $c_{mn} = C(a_1 n + a_2 m, b_1 n + b_2 m)$. We continue by doing the convolution.

$$
\begin{aligned}
S(x, y, u, v) &= \left| \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} c_{mn} e^{j2\pi[(u - a_1 n - a_2 m)x + (v - b_1 n - b_2 m)y]} W(u - a_1 n - a_2 m, v - b_1 n - b_2 m) \right|^2 \\
&= \left[\sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} c_{mn} e^{j2\pi[(u - a_1 n - a_2 m)x + (v - b_1 n - b_2 m)y]} W(u - a_1 n - a_2 m, v - b_1 n - b_2 m) \right] \times \\
&\quad \left[\sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} c_{mn}^{*} e^{-j2\pi[(u - a_1 n - a_2 m)x + (v - b_1 n - b_2 m)y]} W^{*}(u - a_1 n - a_2 m, v - b_1 n - b_2 m) \right] \\
&= \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \sum_{n'=-\infty}^{\infty} \sum_{m'=-\infty}^{\infty} [c_{mn} c_{m'n'}^{*} e^{j2\pi[(u - a_1 n - a_2 m)x + (v - b_1 n - b_2 m)y]} e^{-j2\pi[(u - a_1 n' - a_2 m')x + (v - b_1 n' - b_2 m')y]} \\
&\qquad W(u - a_1 n - a_2 m, v - b_1 n - b_2 m) W^{*}(u - a_1 n' - a_2 m', v - b_1 n' - b_2 m') \quad]
\end{aligned}
$$

(A3 3)

(A3 4)

The next step is to split out the complex conjugate auto-terms as we did in Appendix 1 and Appendix 2.

$$S(x, y, u, v) = \sum_{s=-\infty}^{\infty} \sum_{r=-\infty}^{\infty} |c_{rs}|^2 |W(u - a_1 s - a_2 r, v - b_1 s - b_2 r)|^2 +$$

$$\sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \sum_{n'=-\infty}^{\infty} \sum_{m'=-\infty}^{\infty} [c_{mn} c_{m'n'}{}^{*} e^{j2\pi[(u - a_1 n - a_2 m)x + (v - b_1 n - b_2 m)y]} e^{-j2\pi[(u - a_1 n' - a_2 m')x + (v - b_1 n' - b_2 m')y}$$

$$m \neq m' \text{ or } n \neq n' \qquad W(u - a_1 n - a_2 m, v - b_1 n - b_2 m)\, W^{*}\, (u - a_1 n' - a_2 m', v - b_1 n' - b_2 m') \quad]$$

(A3 5)

We can pair complex conjugate terms in the second term to get the final, real expression.

$$S(x, y, u, v) = \sum_{s=-\infty}^{\infty} \sum_{r=-\infty}^{\infty} |c_{rs}|^2 |W(u - a_1 s - a_2 r, v - b_1 s - b_2 r)|^2 +$$

$$2 \sum_{n=-\infty}^{\infty} \sum_{m=-\infty}^{\infty} \sum_{n'=-\infty}^{\infty} \sum_{m'=n'+1}^{\infty} \mathrm{Re}\, [c_{mn} c_{m'n'}{}^{*} e^{j2\pi[(u - a_1 n - a_2 m)x + (v - b_1 n - b_2 m)y]} e^{-j2\pi[(u - a_1 n' - a_2 m')x + (v - b_1 n' - b_2 m')y}$$

$$W(u - a_1 n - a_2 m, v - b_1 n - b_2 m)\, W^{*}\, (u - a_1 n' - a_2 m', v - b_1 n' - b_2 m') \quad]$$

(A3 6)

The double-sum contains the auto-terms that we want to see. The quadruple sum contains the undesirable cross terms. Since the two $W(u, v)$ functions in each cross term do not fall exactly on top of each other, they will tend to zero each cross term if they are narrow enough.

Appendix 4 The Windowed Fourier Transform of an Affined Transformed Function

In Section 3.3 ("Local Fourier Transform of an Image Texture" on page 75), we used a property of the Fourier transform that says if a function in space undergoes an affine transform, its Fourier transform will also undergo an affine transform. In equation form, if $f(x, y)$ is the image in one patch and $f(a_1x + b_1y, a_2x + b_2y)$ is the image in the other patch, the relationship between their Fourier transforms is[32]

$$\begin{aligned} \Im\,\{f(x, y)\} &= F(u, v) \\ \Im\,\{f(a_1x + b_1y, a_2x + b_2y)\} &= \frac{1}{|D|}F\left(\frac{b_2}{D}u - \frac{a_2}{D}v, -\frac{b_1}{D}u + \frac{a_1}{D}v\right) \end{aligned} \tag{A4 1}$$

with $D = a_1b_2 - a_2b_1$.

If we use this relationship directly, however, we would be ignoring windowing effects. The Fourier transforms of the two functions are taken over only a limited region in the image. This region does not undergo an affine transformation. In what follows, we will show how this situation can be desribed more accurately by dropping the $1/|D|$ scale factor in front of the affine transformed Fourier transform.

We will begin with a 1D formulation to make the reasoning clearer. If we use a rectangular window of width l, then the windowed Fourier transform of the undilated patch $f(x)$ is

$$F_1(u) = \int_{-l/2}^{l/2} f(x) e^{-j2\pi ux} dx$$

$$= \Im\{\mathrm{rect}(\tfrac{x}{l}) f(x)\}$$

$$= l\,\mathrm{sinc}(lu) * F(u) \qquad \text{(A4 2)}$$

The dilated (or compressed) version of $f(x)$ is $f(ax)$. If we disregard the windowing, its Fourier transform is $\frac{1}{a}F(u/a)$ which has a scale factor $1/a$. If we account for the window, we get

$$F_2(u) = \int_{-l/2}^{l/2} f(ax) e^{-j2\pi ux} dx$$

$$= \frac{1}{a} \int_{-al/2}^{al/2} f(\alpha) e^{-j2\pi \frac{u}{a} \alpha} d\alpha \qquad \text{(A4 3)}$$

This comes from substituting $\alpha = ax$.

We proceed by making an assumption about the second integral in Equation (A4 3). We note that the limits of integration are just a scaled version of the limits of integration of the first integral in Equation (A4 2). The integrand is a stationary texture multiplied by a complex exponential. Since this integrand fills the space between the limits of integration (*i.e.* it is not a short pulse), it is reasonable to assume that scaling the limits of integration by a factor of a simply scales the integral by the same factor. This approximation gives

$$F_2(u) = \frac{1}{a} \int_{-al/2}^{al/2} f(\alpha) e^{-j2\pi \frac{u}{a} \alpha} d\alpha$$

$$\approx \int_{-l/2}^{l/2} f(x) e^{-j2\pi \frac{u}{a} x} dx$$

$$= l\,\mathrm{sinc}(lu) * F(u/a) \qquad \text{(A4 4)}$$

Comparing $F_1(u)$ and $F_2(u)$, we see that, except for the convolution with $l\text{sinc}(lu)$, they are simply dilated versions of each other. If l is sufficiently large, then $l\text{sinc}(lu)$ will be narrow and the convolution will have a negligiable effect. To summarize, if we ignore the window effect, then

$$\begin{aligned} \Im\{f(x)\} &= F(u) \\ \Im\{f(bx)\} &= \frac{1}{a}F(u/a) \end{aligned} \tag{A4 5}$$

If we account for a wide rectangular window, then we have the approximations

$$\begin{aligned} F_1(u) &\approx F(u) \\ F_2(u) &\approx F(u/a) \end{aligned} \tag{A4 6}$$

The only difference is the $1/a$ scale factor in the Fourier transform of the dilated function. The remainder of this Appendix develops the same argument for 2D functions.

The 2D texture function is $f(x, y)$, and it is windowed by a 2D rectangular window with dimensions l_x and l_y, shown in Figure A4.1. The windowed Fourier transform is

$$\begin{aligned} F_1(u, v) &= \int_{-l_y/2}^{l_y/2} \int_{-l_x/2}^{l_x/2} f(x, y)\, e^{-j2\pi(ux+vy)}\, dxdy \\ &= \Im\{\text{rect}(\frac{x}{l_x}, \frac{y}{l_y}) f(x, y)\} \\ &= l_x l_y \text{sinc}(l_x u, l_y v) * F(u, v) \end{aligned} \tag{A4 7}$$

The affine transformed version of $f(x, y)$ is $f(a_1x + b_1y, a_2x + b_2y)$. Its windowed Fourier transform is

$$F_2(u, v) = \int_{-l_y/2}^{l_y/2} \int_{-l_x/2}^{l_x/2} f(a_1x + b_1y, a_2x + b_2y)\, e^{-j2\pi(ux+vy)}\, dxdy \tag{A4 8}$$

We make the following substitutions with the following Jacobian determinant:

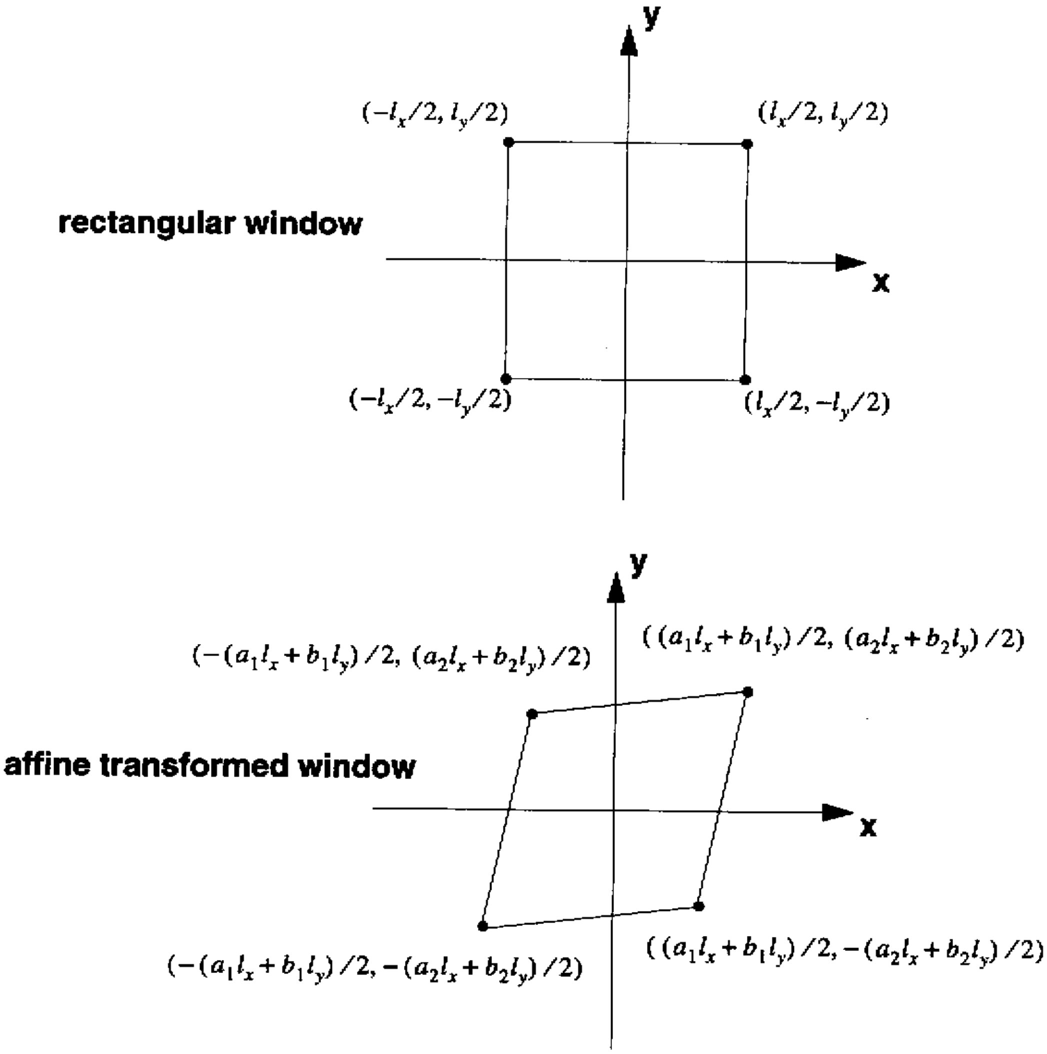

y
(-l_x/2, l_y/2)
(l_x/2, l_y/2)
rectangular window
x
(-l_x/2, -l_y/2)
(l_x/2, -l_y/2)
y
(-(a_1 l_x + b_1 l_y)/2, (a_2 l_x + b_2 l_y)/2)
((a_1 l_x + b_1 l_y)/2, (a_2 l_x + b_2 l_y)/2)
affine transformed window
x
(-(a_1 l_x + b_1 l_y)/2, -(a_2 l_x + b_2 l_y)/2)
((a_1 l_x + b_1 l_y)/2, -(a_2 l_x + b_2 l_y)/2)

Figure A4.1:Window boundaries give the limits of integration.

$$\alpha = a_1 x + b_1 y$$

$$\beta = a_2 x + b_2 y$$

$$|J| = \begin{vmatrix} \frac{\partial x}{\partial \alpha} & \frac{\partial y}{\partial \alpha} \\ \frac{\partial x}{\partial \beta} & \frac{\partial y}{\partial \beta} \end{vmatrix} = \frac{1}{D} \qquad \text{(A4 9)}$$

Then

$$F_2(u,v) = \frac{1}{D}\iint f(\alpha,\beta)\exp\left\{-j2\pi\left[u\left(\frac{\alpha b_2-\beta b_1}{D}\right)+v\left(\frac{\beta a_1-\alpha a_2}{D}\right)\right]\right\}d\alpha d\beta \qquad \text{(A4 10)}$$

This integral is evaluated over the affined transformed window in Figure A4.1. This window has an area of Dl_xl_y which is D times the area of the original rectangular window. We will make the same assumption for this window as we did for the 1D case. That is, the value of this integral taken over a window D times as large as the window in Equation (A4 8) is D times as large as that integral. The approximation gives

$$\begin{aligned}F_2(u,v) &= \int_{-l_y/2}^{l_y/2}\int_{-l_x/2}^{l_x/2}\int f(\alpha,\beta)\exp\left\{-j2\pi\left[u\left(\frac{\alpha b_2-\beta b_1}{D}\right)+v\left(\frac{\beta a_1-\alpha a_2}{D}\right)\right]\right\}d\alpha d\beta \\ &= \int_{-l_y/2}^{l_y/2}\int_{-l_x/2}^{l_x/2}\int f(\alpha,\beta)\exp\left\{-j2\pi\left[\left(\frac{b_2u-a_2v}{D}\right)\alpha+\left(\frac{-b_1u+a_1v}{D}\right)\beta\right]\right\}d\alpha d\beta \\ &= l_xl_y\text{sinc}\,(l_xu,l_yv) * F\left(\frac{b_2}{D}u-\frac{a_2}{D}v,-\frac{b_1}{D}u+\frac{a_1}{D}v\right) \qquad \text{(A4 11)}\end{aligned}$$

Comparing $F_1(u,v)$ and $F_2(u,v)$, we see that they are affine transforms of each other and both convolved with sinc (l_xu, l_yv). If l_x and l_y are large enough, then sinc (l_xu, l_yv) will be narrow, and we can neglect the convolution. To summarize, when we ignore the window effect we get

$$\begin{aligned}\Im\{f(x,y)\} &= F(u,v) \\ \Im\{f(a_1x+b_1y,a_2x+b_2y)\} &= \frac{1}{|D|}F\left(\frac{b_2}{D}u-\frac{a_2}{D}v,-\frac{b_1}{D}u+\frac{a_1}{D}v\right) \qquad \text{(A4 12)}\end{aligned}$$

Accounting for a large window, we get instead

$$\begin{aligned}F_1(u,v) &\approx F(u,v) \\ F_2(u,v) &\approx F\left(\frac{b_2}{D}u-\frac{a_2}{D}v,-\frac{b_1}{D}u+\frac{a_1}{D}v\right) \qquad \text{(A4 13)}\end{aligned}$$

This is the relationship we used in our shape-from-texture algorithms. In spite of the approximations, and in spite of the fact that we use a window that falls off to zero smoothly at the edges, the data and results support its use.

Appendix 5 Enumerating Pairs of Point Matches

In Section 3.7.2 ("Finding and Matching Peaks" on page 128), we discussed an algorithm that matches discrete peaks between two power spectra. In order to find the best set of matches, we consider every possible set of matches, including leaving some of the peaks unmatched. This appendix shows how many matches we have to consider.

Suppose there are n_1 points in the first set and n_2 points in the second set. There can be from 0 to $\min(n_1, n_2)$ matched pairs between these two sets. If there are m matched pairs, then there are $\binom{n_1}{m}$ possible sets of points from the first set to match with $\binom{n_2}{m}$ sets of points from the second set, where

$$\binom{n}{m} = \frac{n!}{m!\,(n-m)\,!} = \begin{array}{c}\text{number of combinations of n}\\ \text{distinct objects taken m at a time}\end{array} \qquad \text{(A5 1)}$$

If there are m points to match in each set, then there are $m!$ possible ways of pairing these points. Considering all possible numbers of matched pairs, the total number of sets of distinct matches to consider is

$$M(n_1, n_2) = \sum_{m=0}^{\min(n_1, n_2)} \binom{n_1}{m}\binom{n_2}{m} m!$$

$$= n_1!n_2! \sum_{m=0}^{\min(n_1, n_2)} \frac{1}{m!\,(n_1-m)\,!\,(n_2-m)\,!} \qquad \text{(A5 2)}$$

The number of matches that must be considered for various values of n_1 and n_2 are given in Table A5.1. In our case, we allow a maximum of 6 peaks for each patch, so we never have to consider more than 13,327 sets of matches.

n_1 n_2	1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10	11
2	3	7	13	21	31	43	57	73	91	111
3	4	13	34	73	136	229	358	529	748	1021
4	5	21	73	209	501	1045	1961	3393	5509	8501
5	6	31	136	501	1546	4051	9276	19,081	36,046	63,591
6	7	43	229	1045	4051	13,327	37,633	93,289	207,775	424,051
7	8	57	358	1961	9276	37,633	130,922	394,353	1,047,376	2,501,801
8	9	73	529	3393	19,081	93,289	394,353	1,441,729	4,596,553	12,975,561
9	10	91	748	5509	36,046	207,775	1,047,376	4,596,553	17,572,114	58,941,091
10	11	111	1021	8501	63,591	424,051	2,501,801	12,975,561	58,941,091	234,662,231

Table A5.1: Number of distinct sets of matches between sets of n_1 and n_2 points.

Appendix 6 Using the Canon Xap Shot Camera

We used a Canon Xap Shot camera, model RC-250, to take some of our test images. The Xap Shot is a portable still video camera that can store up to 50 images on its floppy disk. It can be connected to a video monitor to display the images or to a digitizer to digitize the images. We had to calibrate the camera, because our shape-from-texture algorithms need the focal length, pixel size, and pixel center of the image. This appendix describes the calibration and preprocessing necessary for using these images.

We used Reg Willson's convenient camera calibration program (publicly archived, anonymous ftp to FTP.TELEOS.COM). The input to this program is a list of 3D scene points and their corresponding image points. Our calibration target was a 10x16 lattice of dots on one-inch centers on a flat board. We positioned the target on a rail table whose motion was approximately parallel to the camera's optical axis. We took three pictures of the target at three different distances from the camera. The closest distance was approximately 65 inches. We moved the target back nice inches for the second image and another nine inches for the third image.

A close look at the calibration images showed that the dots were split horizontally, as shown in the close-up in Figure A6.1. The manufacturer told us that during playback the camera inserts new rows in between the rows actually produced by the 250-line CCD. Apparently their algorithm for generating the new lines cannot maintain small features, or we would not have noticed the phenomenon. Our Matrox digitizer gave images with 480 rows, but we threw out the artificial rows starting with the second from the top and alternating down the image. This made the images look squashed vertically, which we corrected by displaying and printing them with each row repeated twice.

Willson's program requires data on the camera's CCD and the digitizer. Specifically, we had to supply the physical size of the CCD pixels, the number of rows and columns of the CCD, the corresponding physical size of the digitizer's pixels, and the number of rows and columns of the digitizer. The manufacturer's specifications give the size of the CCD as 4.8 mm vertically and 6.4 mm horizontally. The manufacturer told us that the CCD has 250 rows and 782 columns. Thus, the physical size

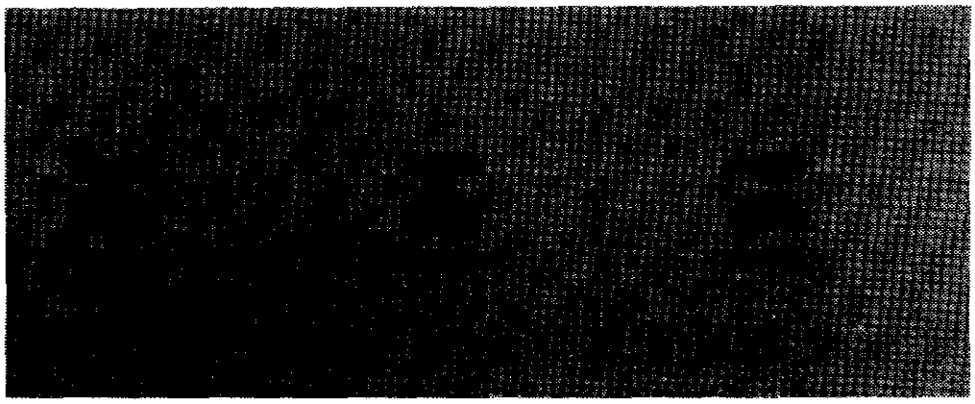


Figure A6.1:The Canon Xap Shot split these three dots by inserting new rows in between those actually produced by the CCD.

of the pixels on the CCD is $4.8\text{mm}/250 = 0.0192\text{mm}$ vertically and $6.4\text{mm}/782 \approx 0.00818\text{mm}$ horizontally. The camera effectively halves the vertical size of the CCD pixels by inserting new rows, which we effectively double by throwing out alternate rows. Thus, the vertical size of the digitizer pixels is the same as the vertical size of the CCD pixels. The Matrox digitizer gives images with 480 rows and 512 columns. For the horizontal size of the digitizer pixels, we estimate that the digitizer spreads it 512 columns over the horizontally scanned length of the CCD. We were told by the manufacturer that the CCD is underscanned to 242 rows and 739 columns. This means that the digitizer's 512 rows are spread across $739 \times 0.00818\text{mm}$, giving a horizontal size of about 0.0118mm for digitizer pixels. Willson's program calibrates this value by computing s_x - the "uncertainty factor for scale of horizontal scanline." In our run, this came out to be 1.027753. We scaled our estimate of the size of the horizontal digitizer pixels by s_x to get 0.0121mm.

The calibration program gave a focal length of 10.94 mm, which is close to the manufacturer's figure of 11 mm. The center rows was at 119.91 pixels from the top of the image and the center column was 254.43 pixels from the left of the image.

We noticed that the Xap Shot camera adds a high-frequency periodic pattern to its images, perhaps for dithering. Figure A6.2 shows a Xap Shot image of Jim Moody after removing alternate rows. We computed the spectrogram of this image using 64x64 windows on 64-pixel centers. The mean of all the power spectra is shown in Figure A6.3. Even though there don't appear to be many textured objects in this scene, the mean spectrogram shows four high-frequency peaks at (row,column) coordinates of (0,11), (0,49), (63,11), and (63,49). After computing each power spectra for the spectrogram, we zeroed these points and their neighbors within a radius of four pixels for all the Xap Shot images.

Figure A6.2:Xap Shot image of Jim Moody after removing alternate rows. None of the objects in this image show much texture, yet the mean of the spectrogram in Figure A6.3 shows high-frequency peaks.

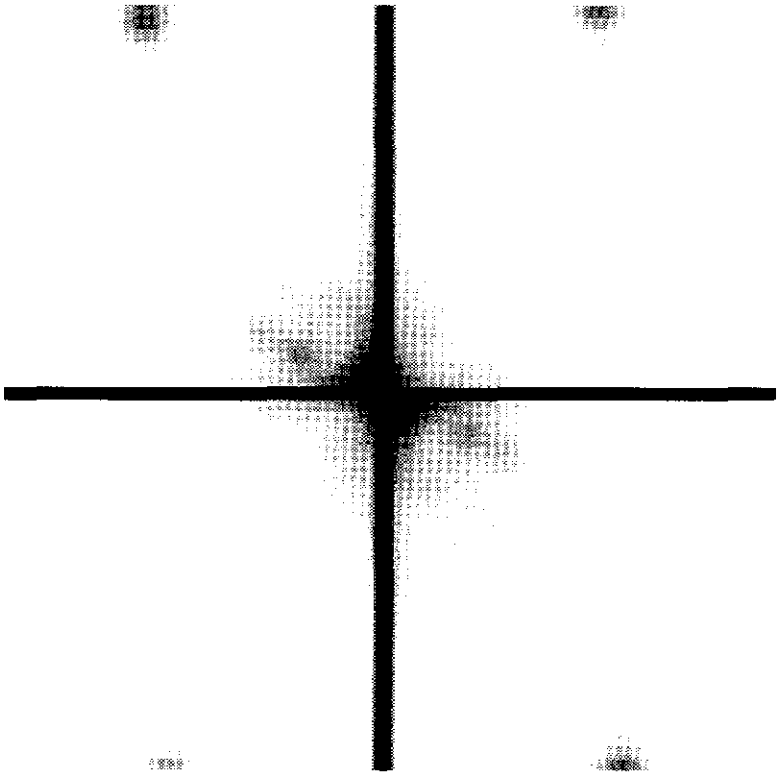


Figure A6.3:The mean of the spectrogram of the image in Figure A6.2 shows four high-frequency peaks that were artificially produced by the Xap Shot camera. We removed these peaks in the spectrograms.

References

[1] Aitken, G.J.M. and P.F. Jones. "Three-Dimensional Image Capture by Volume Imaging." In *Proceedings of the SPIE, Sensing and Reconstruction of Three-Dimensional Objects and Scenes*, vol.1260, 2-9, 1990.

[2] Aloimonos, J. "Shape from Texture." *Biological Cybernetics* 58 (1988): 345-360.

[3] Aloimonos, John and Michael Swain. "Shape from Patterns: Regularization" *International Journal of Computer Vision* 2 (1989): 171-187.

[4] Bajcsy, Ruzena. "Computer Description of Textured Surfaces." *Third International Joint Conference on Artificial Intelligence*, 572-577, August 1973.

[5] Bajcsy, Ruzena and Lawrence Lieberman. "Texture Gradient as a Depth Cue." *Computer Graphics and Image Processing* 5 (1976): 52-67.

[6] Bell, Bernard W. and Chris L. Koliopoulos. "Moire Topography, Sampling Theory, and Charged-Coupled Devices." *Optics Letters* 9 (no. 5, May 1984): 171-173.

[7] Blake, Andrew and Constantinos Marinos. "Shape from Texture: Estimation, Isotropy and Moments." *Artificial Intelligence* 45 (1990): 323-380.

[8] Boashash, Boualem. "Note on the Use of the Wigner Distribution for Time-Frequency Signal Analysis." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 36 (no. 9, September 1988): 1518-1521.

[9] Bovik, Alan Conrad, Marianna Clark, and Wilson S. Geisler. "Multichannel Texture Analysis Using Localized Spatial Filters." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12 (January 1990): 55-73.

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

[11] Brown, Lisa Gottesfeld and Haim Shvaytser. "Surface Orientation from Projective Foreshortening of Isotropic Texture Autocorrelation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12 (June 1990): 584-588.

[12] Cetica, Maurizio, Franco Francini, and Duilio Bertani. "Moire With One Grating and a Photodiode Array." *Applied Optics* 24 (no. 11, 1 June 1985): 1565-1566.

[13] Chellappaa, R. "Two-Dimensional Discrete Gaussian Markov Random Field Models for Image Processing." In *Progress in Pattern Recognition* 2. ed. L. N. Kanal and A. Rosenfeld, Elsevier Science Publishers B.V. 1985.

[14] Chellappa, Rama and Shankar Chatterjee. "Classification of Textures Using Gaussian Markov Random Fields." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 33 (no. 4, August 1985): 959-963.

[15] Chen, Susan S. and James M. Keller and Richard M. Crownover. "Shape from Fractal Geometry." *Artificial Intelligence* 43 (1990): 199-218.

[16] Claasen, T. A. C. M. and W. F. G. Mecklenbrauker. "The Wigner Distribution - A Tool for Time-Frequency Signal Analysis, Part I: Continuous-Time Signals." *Philips Journal of Research* 35 (1980): 217-250.

[17] Claasen, T. A. C. M. and W. F. G. Mecklenbrauker. "The Wigner Distribution - A Tool for Time-Frequency Signal Analysis, Part II: Discrete-Time Signals." *Philips Journal of Research* 35 (1980): 276-300.

[18] Claasen, T. A. C. M. and W. F. G. Mecklenbrauker. "The Wigner Distribution - A Tool for Time-Frequency Signal Analysis, Part III: Relations With Other Time-Frequency Signal Transforms." *Philips Journal of Research* 35 (1980): 372-389.

[19] Clark, Marianna, and Alan C. Bovik and Wilson S. Geisler. "Texture Segmentation Using Gabor Modulation/Demodulation." Pattern Recognition Letters 6 (1987): 261-267.

[20] Cohen, Fernand S. and David B. Cooper. "Simple Parallel Hierarchical and Relaxation Algorithms for Segmenting Noncausal Markovian Random Fields." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 9 (no. 2, March 1987): 195-219.

[21] Cook, Charles E. and Marvin Bernfeld. *Radar Signals: An Introduction to Theory and Application.* New York: Academic Press, 1967.

[22] Daubechies, Ingrid. "Orthonormal Bases of Compactly Supported Wavelets." *Communications on Pure and Applied Mathematics* 41 (November 1988): 909-996.

[23] DeFatta, David J., Joseph G. Lucas, and William S. Hodgkiss. *Digital Signal Processing: A System Design Approach.* New York: John Wiley & Sons, 1988, p. 270.

[24] Duda, Richard O. and Peter E. Hart. Pattern Classification and Scene Analysis. New York: John Wiley & Sons, 1973, p. 228.

[25] Dyer, Charles S. and Azriel Rosenfeld. "Fourier Texture Features: Suppression of Aperature Effects." *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6 (no. 10, October 1976): 703-705.

[26] Fleet, David J. and Allan D. Jepson, and Michael R. M. Jenkin. "Phase-Based Disparity Measurement." *CVGIP: Image Understanding* 53 (no. 2, March 1991): 198-210.

[27] Fogel, I. and D. Sagi. "Gabor Filters as Texture Discriminator." *Biological Cybernetics* 61 (June 1989): 103-113.

[28] Francos, Joseph M., A. Zvi Meiri, and Boaz Porat. "A Unified Texture Model Based on a 2-D Wold-Like Decomposition." *IEEE Transactions on Signal Processing* 41 (no. 8, August 1993): 2665-2678.

[29] Gabor, D. "Theory of Communication." *The Journal of the Institute of Electrical Engineers, Part III* 93 (21, January 1946): 429-457.

[30] Galloway, Mary M. "Texture Analysis Using Gray Level Run Lengths." *Computer Graphics and Image Processing* 4 (1975): 172-179.

[31] Garding, Jonas. "Shape from Surface Markings." Ph.D. Dissertation, Department of Numerical Analysis and Computing Science, Royal Institute of Technology, Stockholm, Sweden, May 1991.

[32] Gaskill, Jack D. *Linear Systems, Fourier Transforms, and Optics.* New York: John Wiley & Sons, 1978.

[33] Gramenopoulos, Nicholas. "Terrain Type Recognition Using ERTS-1 MSS Images." In *Symposium on Significant Results Obtained from the Earth Resources Technology Satellite-1*, Vol. 1, *Technical Presentations*, Section B, 1229-1241, 1973.

[34] Haralick, Robert M., K. Shanmugam, and Its'hak Dinstein. "Textural Features for Image Classification." *IEEE Transactions on Systems, Man, and Cybernetics* SMC-3 (no. 6, November 1973): 610-623.

[35] Haralick, Robert M. "Statistical and Structural Approaches to Texture." Proceedings of the IEEE 67 (no. 5, May 1979): 786-804.

[36] Harris, Fredric, J. "On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform." *Proceedings of the IEEE* 66 (January 1978): 51-83.

[37] Hassner, Martin and Jack Sklansky. "The Use of Markov Random Fields as Models of Texture." *Computer Graphics and Image Processing* 12 (1980): 357-370.

[38] Haykin, Simon. *An Introduction to Analog and Digital Communications.* New York: John Wiley & Sons, 1989.

[39] Heeger, David J. "Optical Flow Using Spatiaotemporal Filters." *International Journal of Computer Vision* 1 (no. 4, January 1988): 279-302.

[40] Hlawatsch, F. and G. F. Boudreaux-Bartels. "Linear and Quadratic Time-Frequency Signal Representations." *IEEE Signal Processing Magazine*, April 1992, 21-67.

[41] Holyer, Ronald J. and Sarah H. Peckinpaugh. "Edge Detection Applied to Satellite Imagery of the Oceans." IEEE Transactions on Geoscience and Remote Sensing 27 (no. 1, January 1989): 46-56.

[42] Horn, Berthold. "Focusing." *Massachusetts Institute of Technology Artificial Intelligence Memo*, No. 160, May 1968.

[43] Idesawa, Masanori, Toyohiko Yatagai, and Takashi Soma. "Scanning Moire Method and Automatic Measurement of 3-D Shapes." *Applied Optics* 16 (no. 8, August 1977): 2152-2162.

[44] Jacobson, Lowell and Harry Wechsler, "Derivation of Optical Flow Using a Spatiotemporal-Frequency Approach." *Computer Vision, Graphics, and Image Processing* 38, (1987): 29-65.

[45] Jau, Jack Y. and Roland T. Chin. "Shape from Texture Using the Wigner Distribution." *Computer Vision, Graphics, and Image Processing* 52 (1990): 248-263.

[46] Jones, D. G. and J. Malik. "A Computational Framework for Determining Stereo Correspondence from a Set of Linear Spatial Filters." *European Conference on Computer Vision*, 395-410, 1992.

[47] Jones, Douglas L. and Thomas W. Parks. "A High Resolution Data-Adaptive Time-Frequency Representation." IEEE Transactions on Acoustics, Speech, and Signal Processing 38 (no. 12, December 1990): 2127-2135.

[48] Julesz, B. and T Caelli. "On the Limit of Fourier Decompositions in Visual Texture Perception." *Perception* 8: 69-73.

[49] Julesz, B. and J. R. Bergen. "Textons, The Fundamental Elements in Preattentive Vision and Perception of Textures." The Bell System Technical Journal 62 (no. 6, July-August 1983): 1619-1645.

[50] Kanade, Takeo and John P. Kender. "Mapping Image Properties into Shape Constraints: Skewed Symmetry, Affine-Transformable Patterns, and the Shape-from-Texture Paradigm." Carnegie Mellon University Computer ScienceTechnical Report No. CMU-CS-80-133, July 1980.

[51] Kanatani, Ken-ichi and Tsai-Chia Chou. "Shape from Texture: General Principle." Artificial Intelligence 38 (1989): 1-48.

[52] Kaneko H. and E. Yodogawa. "A Markov Random Field Application to Textrue Classification." *IEEE Pattern Recognition and Image Processing Conference*, 221-225, June 1982.

[53] Kender, John R. "Shape from Texture: A Computational Paradigm." *Image Understanding Proceedings*, 134-138, April 1979.

[54] Krotkov, Eric. "Focusing." *International Journal of Computer Vision* 1 (no. 3, 1987): 223-237.

[55] Krueger, Wolfgang. "Intensity Fluctuations and Natural Texturing." *SIGGRAPH '88 Conference Proceedings*, 213-220, August 1988.

[56] Krumm, John and Steven A. Shafer. "Local Spatial Frequency Analysis for Computer Vision." Carnegie Mellon University Robotics Institute Technical Report No. CMU-RI-TR-90-11, May 1990.

[57] Krumm, John and Steven A. Shafer. "Sampled-Grating and Crossed-Grating Models of Moire Patterns from Digital Imaging." *Optical Engineering* 30 (no. 2, February 1991): 195-206.

[58] Krumm, John and Steven A. Shafer. "Shape from Periodic Texture Using the Spectrogram." *IEEE Conference on Computer Vision and Pattern Recognition*, 284-289, 1992.

[59] Krumm, John and Steven A. Shafer. "Segmenting Textured 3D Surfaces Using the Space/Frequency Representation." Carnegie Mellon University Robotics Institute Technical Report No. CMU-RI-TR-93-14, April 1993.

[60] Krumm, John and Steven A. Shafer. "Segmenting Textured 3D Surfaces Using the Space/Frequency Representation." *Spatial Vision*, 1994, to appear.

[61] Langley, K., T. Atherton, R. Wilson, and M. Larcombe. "Vertical and Horizontal Disparities from Phase." *European Conference on Computer Vision*, 315-325, 1990.

[62] Leclerc, Yvan G. "Constructing Simple Stable Descriptions for Image Partitioning." *International Journal of Computer Vision* 3 (1989): 73-102.

[63] Loh, Horng-Hai and Jia-Guu and Ren C. Luo. "The Analysis of Natural Textures Using Runn Length Features." *IEEE Transactions on Industrial Electronics* 35, (no. 2, May 1988): 323-327.

[64] Lonnestad, Tor. "A New Set of Texture Features Based on the Haar Transform." *International Conference on Pattern Recognition*, vol. III, 676-679, September 1992.

[65] Lucas, Bruce D. and Takeo Kanade. "An Iterative Image Registration Technique with an Application to Stereo Vision." Proceedings of the Seventh International Joint Conference on Artificial Intelligence, August 1981, 674-679.

[66] Malik, Jitendra and Pietro Perona. "Preattentive Texture Discrimination with Early Vision Mechanisms." *Journal of the Optical Society of America - A* 7 (no. 5, May 1990): 923-932.

[67] Malik, Jitendra. Personal Communication, 5 March 1993.

[68] Malik, Jitendra and Ruth Rosenholtz. "A Differential Method for Computing Local Shape-From-Texture for Planar and Curved Surfaces." Computer Vision and Pattern Recognition Conference, June 1993, 267-273.

[69] Mandelbrot, Benoit B. *Fractals: Form, Chance, and Dimension.* San Francisco: W. H. Freeman and Company, 1977.

[70] Marr, David. *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information.* San Francisco: W.H. Freeman and Company, 1982.

[71] Matsuyama, Takashi, Shu-Ichi Miura, and Makoto Nagao. "Structural Analysis of Natural Textures by Fourier Transformation." *Computer Vision, Graphics and Image Processing* 24 (1983): 347-362.

[72] Meer, Peter and Doron Mintz and Don Yoon Kim and Azriel Rosenfeld. "Robust Regression Methods for Computer Vision: A Review." International Journal of Computer Vision 6 (no. 1, April 1991): 59-70.

[73] Morimoto, Yoshiharu, Yasuyuki Seguchi, and Toshihoko Higashi. "Application of Moire Analysis of Strain Using Fourier Transform." *Optical Engineering* 27 (no. 8, August 1988): 650-656.

[74] Muerle, John L. and Donald C. Allen. "Experimental Evaluation of Techniques for Automatic Segmentation of Objects in a Complex Scene." In George C. Cheng, Robert S. Ledley, Donald K. Pollock, and Azriel Rosenfeld, Editors, *Pictorial Pattern Recognition*, Thompson Book Company, 1968.

[75] Nelson, Brad and Pradeep Khosla. "Integrating Sensor Placement and Visual Tracking Strategies." *Third International Symposium on Experimental Robotics*, 1993.

[76] Ohta, Yu-ichi, Kiyoshi Maenobu, and Toshiyuki Sakai. "Obtaining Surface Orientation from Texels Under Perspective Projection." *Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 746-751, August 1981.

[77] Okutomi, Masatoshi and Takeo Kanade. "A Locally Adaptive Window for Signal Matching." *International Journal of Computer Vision* 7 (no. 2, 1992): 143-162.

[78] Oppenheim, Alan V. and Jae S. Lim. "The Importance of Phase in Signals." *Proceedings of the IEEE* 69 (no. 5, May 1981): 529-541.

[79] Oppenheim, Alan V. and Alan S. Willsky. *Signals and Systems.* Englewood Cliffs, New Jersey: Prentice-Hall, 1983.

[80] Papoulis, Athanasios. *Signal Analysis.* New York: McGraw-Hill Book Company, 1977.

[81] Parker, David H. "Moire Patterns in Three-Dimensional Fourier Space." *Optical Engineering* 30 (no. 10, October 1991): 1534-1541.

[82] Patel, Maqbool A. S. and Fernand S. Cohen. "Shape from Texture using Markov Random Field Models and Stero-Windows." *IEEE Conference on Computer Vision and Pattern Recognition*, 290-295, 1992.

[83] Pentland, Alex P. "Shading Into Texture." *DARPA Image Understanding Proceedings*, 179-183, October 1984.

[84] Pentland, Alex P. "A New Sense for Depth of Field." In Aravind Joshi, Editor, *Ninth International Joint Conference on Artificial Intelligence*, 988-994, 1985.

[85] Pentland Alex P. "The Transform Method for Shape from Shading." M.I.T Media Lab Vision Sciences Technical Report 106, July 15, 1988.

[86] Press, William H., Brian P. Flannery, Saul A. Teukolsky, and William T. Vetterling. *Numerical Recipes: The Art of Scientific Computing*, First Edition, Cambridge University Press, 1986.

[87] Reed, Todd R. and Harry Wechsler. "Segmentation of Textured Images and Gestalt Organization Using Spatial/Spatial Frequency Representations." IEEE Transactions on Pattern Analysis and Machine Intelligence 12 (no. 1, January 1990): 1-12.

[88] Rioul, Olivier and Martin Vetterli. "Wavelets and Signal Processing." *IEEE Signal Processing Magazine*, October 1991, 14-38.

[89] Rissanen, Jorma. "A Universal Prior for Integers and Estimation by Minimum Description Length." *The Annals of Statistics* 11 (no. 2, 1983): 416-431.

[90] Rosenfeld, Azriel and Bernice S. Lipkin. "Texture Synthesis." In Bernice Sacks Lipkin and Azirel Rosenfeld, Editors, *Picture Processing and Psychopictorics*, Academic Press, 1970.

[91] Rosenfeld, Azriel and Mark Thurston. "Edge and Curve Detection for Visual Scene Analysis." *IEEE Transactions on Computers* 20 (no. 5, May 1971): 562-569.

[92] Sanger, Terence D. "Stereo Disparity Computation Using Gabor Filters." *Biological Cybernetics* 59 (1988): 405-418.

[93] Schachter, Bruce J., Azriel Rosenfeld, and Larry S. Davis. "Random Mosaic Models for Textures." *IEEE Transactions on Systems, Man, and Cybernetics* SMC-8 (no. 9, September 1978): 694-702.

[94] Shafer, Steven A. "Why Can't We Model the Physical World?" In *CMU Computer Science: A 25th Anniversary Commemorative*, ed. Rick Rashid. Addison-Wesley Publishing Co., 1991.

[95] Subbarao, Murali. "Parallel Depth Recovery by Changing Camera Parameters." *Second International Conference on Computer Vision*, 149-155, 1988.

[96] Subbarao, Murali and Natarajan Gurumoorthy. "Depth Recovery from Blurred Edges." *IEEE Conference on Computer Vision and Pattern Recognition*, 498-503, 1988.

[97] Super, Boaz J. and Alan C. Bovik. "Three-Dimensional Orientation from Texture Using Gabor Wavelets." *SPIE Conference on Visual Communications and Image Processing*, November 1991.

[98] Super, Boaz J. and Alan C. Bovik. "Shape-from-Texture by Wavelet-Based Measurement of Local Spectral Moments." *IEEE Conference on Computer Vision and Pattern Recognition*, 296-301, June 1992.

[99] Super, Boaz. "Filters for Directly Detecting Surface Orientation in an Image." *SPIE Conference on Visual Communications and Image Processing*, November 1992.

[100] Tamura, Hideyuki, Shunji Mori and Takashi Yamawaki. "Textural Features Corresponding to Visual Perception." IEEE Transactions on Systems, Man, and Cybernetics SMC-8 (no. 6, June 1978): 460-473.

[101] Tan, T. N. and A. G. Constantinides. "Texture Analysis Based on a Human Visual Model." *International Conference on Acoustics, Speech, and Signal Processing*, vol. 4, *Multidimensional Signal Processing*, 2137-2140, April 1990.

[102] Tomita, Fumiaki and Saburo Tsuji. *Computer Analysis of Visual Textures*. Kluwer Academic Publishers, Norwell, Massachusetts, 1990.

[103] Triendl, E. E. "Automatic Terrain Mapping by Texture Recognition." *Proceedings of the Eighth International Symposium on Remote Sensing of Environment*, vol. 1, 771-776, October 1972.

[104] Tsai, Roger Y. "An Efficient and Accurate Camera Calibration Technique for 3D Machine Vision." *IEEE Conference on Computer Vision and Pattern Recognition*, 364-374, 1986.

[105] Tsuji, Saburo and Fumiaki Tomita. "A Structural Analyzer for a Class of Texture." *Computer Graphics and Image Processing* 2 (1973) 216-231.

[106] Turner, M.R. "Texture Discrimination by Gabor Functions." *Biological Cybernetics* 55, (October 1986): 71-82.

[107] Vakman, D. E. *Sophisticated Signals and the Uncertainty Principle in Radar.* Edited by Ernest Jacobs, Translated by K. N. Trirogoff. Springer-Verlag, 1968.

[108] Van Gool, L., P. Dewaele, and A. Oosterlinck. "Texture Analysis Anno 1983." *Computer Vision, Graphics, and Image Processing* 29 (1985): 336-357.

[109] Voss, R. F. "Fractals in Nature." Course notes on *FRACTALS: Introduction, Basics, and Perspectives*, 1987.

[110] Wallace, Richard S. "Finding Natural Clusters through Entropy Minimization" (Ph.D. Thesis). Carnegie Mellon University Computer Science Technical Report No. CMU-CS-89-183, June 1989.

[111] Weber, Joseph and Jitendra Malik. "Robust Computation of Optical Flow in a Multi-Scale Differential Framework." University of California Berkeley, Computer Science Division, Technical Report No. UCB/CSD 92/709, November 1992.

[112] Wechsler, Harry. "Texture Analysis - A Survey." *Signal Processing* 2 (1980): 271-282.

[113] Weng, Juyang. "A Theory of Image Matching." *Third International Conference on Computer Vision,* 200-209, 1990.

[114] Weszka, Joan S. and Charles R. Dyer and Azriel Rosenfeld. "A Comparative Study of Texture Measures for Terrain Classification." *IEEE Transactions on Systems, Man, and Cybernetics* SMC-6 (no. 4, April 1976): 269-285.

[115] Witkin, Andrew P. "Recovering Surface Shape and Orientation from Texture." *Artificial Intelligence* 17 (1981): 17-45.

[116] Xiong, Yalin and Steven A. Shafer. "Depth from Focusing and Defocusing." *IEEE Conference on Computer Vision and Pattern Recognition*, 68-73, 1993.

[117] Yu, Feihong, Yinzhong Liang, Zhengmin Li, and Liren Liu. "Lau Effect of a Skew-Periodic Object: Theoretical and Experimental Investigation." Journal of the Optical Society of America A 9 (no. 11, November 1992):2013-2020.