Computer Vision
A Modern Approach

David A. Forsyth
University of California at Berkeley

Jean Ponce
University of Illinois at Urbana-Champaign

An Alan R. Apt Book

Prentice Hall
Upper Saddle River, New Jersey 07458

9

Texture

Texture is a phenomenon that is widespread, easy to recognise and hard to define. Typically, whether an effect is referred to as texture or not depends on the scale at which it is viewed. A leaf that occupies most of an image is an object, but the foliage of a tree is a texture. Texture arises from a number of different sources. Firstly, views of large numbers of small objects are often best thought of as textures. Examples include grass, foliage, brush, pebbles and hair. Secondly, many surfaces are marked with orderly patterns that look like large numbers of small objects. Examples include: the spots of animals like leopards or cheetahs; the stripes of animals like tigers or zebras; the patterns on bark, wood and skin.

There are three standard problems to do with texture:

- **Texture segmentation** is the problem of breaking an image into components within which the texture is constant. Texture segmentation involves both representing a texture, and determining the basis on which segment boundaries are to be determined. In this chapter, we deal only with the question of how textures should be *represented* (Section 9.1); chapters 14 and 16 show how to segment textured images using this representation.
- **Texture synthesis** seeks to construct large regions of texture from small example images. We do this by using the example images to build probability models of the texture, and then drawing on the probability model to obtain textured images. There are a variety of methods for building a probability model; three successful current methods are described in Section 9.3.
- **Shape from texture** involves recovering surface orientation or surface shape from image texture. We do this by assuming that texture "looks the same" at different points on a surface; this means that the deformation of the texture from point to point is a cue to

Figure 9.1 A set of texture examples, used in experiments with human subjects to tell how easily various types of textures can be discriminated. Note that these textures are made of quite stylized subelements, repeated in a meaningful way. *Reprinted from A Computational Model of Texture Segmentation, J. Malik and P. Perona, Proc. Computer Vision and Pattern Recognition, 1989, © 1989, IEEE*

the shape of the surface. In Section 9.4, we describe the main lines of reasoning in this (rather technical) area.

9.1 REPRESENTING TEXTURE

Image textures generally consist of organised patterns of quite regular subelements (sometimes called *textons*). For example, one texture in Figure 9.1 consists of triangles. Similarly, another texture in that figure consists of arrows. One natural way to try and represent texture is to find the textons, and then describe the way in which they are laid out.

The difficulty with this approach is that there is no known canonical set of textons, meaning that it isn't clear what one should look for. Instead of looking for patterns at the level of arrow-

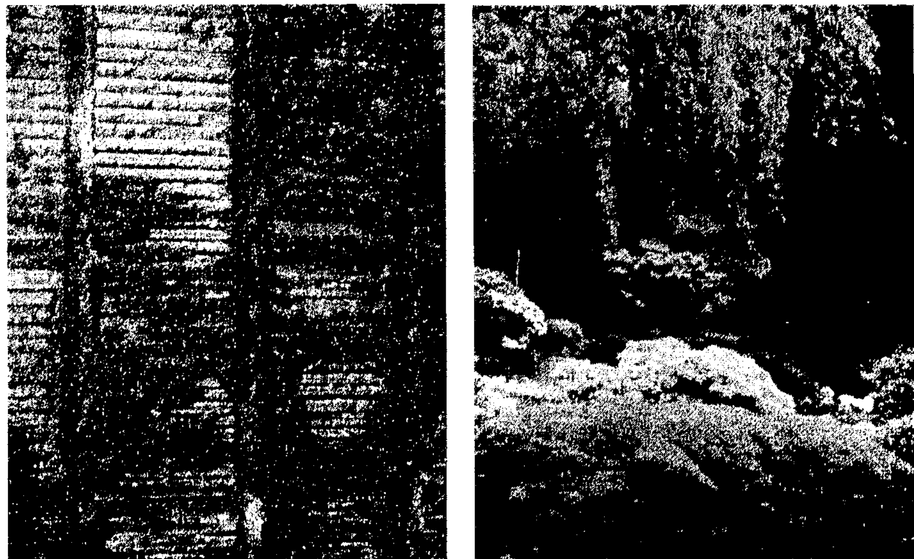

Figure 9.2 Typical textured images. For materials such as brush, grass, foliage and water, our perception of what the material is is quite intimately related to the texture (for the figure on the **left**, what would the surface feel like if you ran your fingers over it? is it wet?, etc.). Notice how much information you are getting about the type of plants, their shape, etc. from the textures in the figure on the **right**. These textures are also made of quite stylized subelements, arranged in a rough pattern.

heads and triangles, we could look for even simpler pattern elements—dots and bars, say—and then reason about their spatial layout. The advantage of this approach is that it is easy to look for simple pattern elements by filtering an image.

9.1.1 Extracting Image Structure with Filter Banks

In Section 7.5, we saw that convolving an image with a linear filter yields a representation of the image on a different basis. The advantage of transforming an image to the new basis given by convolving it with a filter, is that the process makes the local structure of the image clear. This is because there is a strong response when the image pattern in a neighborhood looks similar to the filter kernel, and a weak response when it doesn't.

This suggests representing image textures in terms of the response of a collection of filters. The collection of different filters would consist of a series of patterns—spots and bars are usual—at a collection of scales (to identify bigger or smaller spots or bars, say). The value at a point in a derived image represents the local "spottiness" ("barriness", etc.) at a particular scale at the corresponding point in the image. While this representation is now heavily redundant, it exposes structure ("spottiness", "barriness", etc.), in a way that has proven helpful.

Generally, spot filters are useful because they respond strongly to small regions that differ from their neighbors (for example, on either side of an edge, or at a spot). The other attraction is that they detect non-oriented structure. Bar filters, on the other hand, are oriented, and tend to respond to oriented structure.

Spots and Bars by Weighted Sums of Gaussians But what filters should we use? There is no canonical answer. A variety of answers have been tried. By analogy with the human visual cortex, it is usual to use at least one spot filter and a collection of oriented bar filters at different orientations, scales and *phases*. The phase of the bar refers to the phase of a cross-section perpendicular to the bar, thought of as a sinusoid (i.e., if the cross section passes through zero at the origin, then the phase is $0°$).

One way to obtain these filters is to form a weighted difference of Gaussian filters at different scales; this technique was used for the filters of Figure 9.3. The filters for this example consist of

- **A spot**, given by a weighted sum of three concentric, symmetric Gaussians, with weights 1, -2 and 1, and corresponding sigmas 0.62, 1 and 1.6.
- **Another spot**, given by a weighted sum of two concentric, symmetric Gaussians, with weights 1 and -1, and corresponding sigmas 0.71 and 1.14.
- **A series of oriented bars**, consisting of a weighted sum of three oriented Gaussians, which are offset with respect to one another. There are six versions of these bars; each is a rotated version of a horizontal bar. The Gaussians in the horizontal bar have weights -1, 2 and -1. They have different sigma's in the x and in the y directions; the σ_x values are all 2, and the σ_y values are all 1. The centers are offset along the y axis, lying at $(0, 1)$, $(0, 0)$ and $(0, -1)$.

You should understand that the details of the choice of filter are almost certainly immaterial. There is a body of experience that suggests that there should be a series of spots and bars at various scales and orientations—which is what this collection provides—but very little reason to believe that optimizing the choice of filters produces any major advantage.

Figures 9.4 and 9.5 illustrate the absolute value of the responses of this bank of filters to an input image of a butterfly. Notice that, while the bar filters are not completely reliable bar

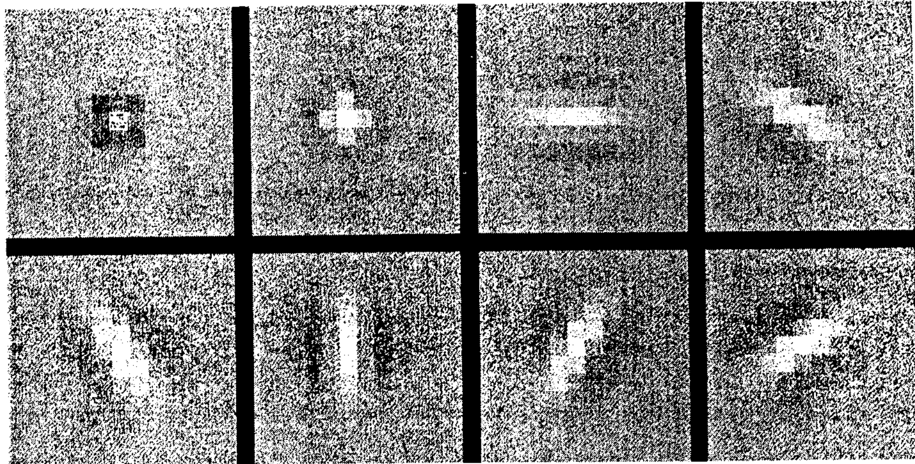

Figure 9.3 A set of eight filters used for expanding images into a series of responses. These filters are shown at a fixed scale, with zero represented by a mid-grey level, lighter values being positive and darker values being negative. They represent two distinct spots, and six bars; the set of filters is that used by Malik and Perona (1990).

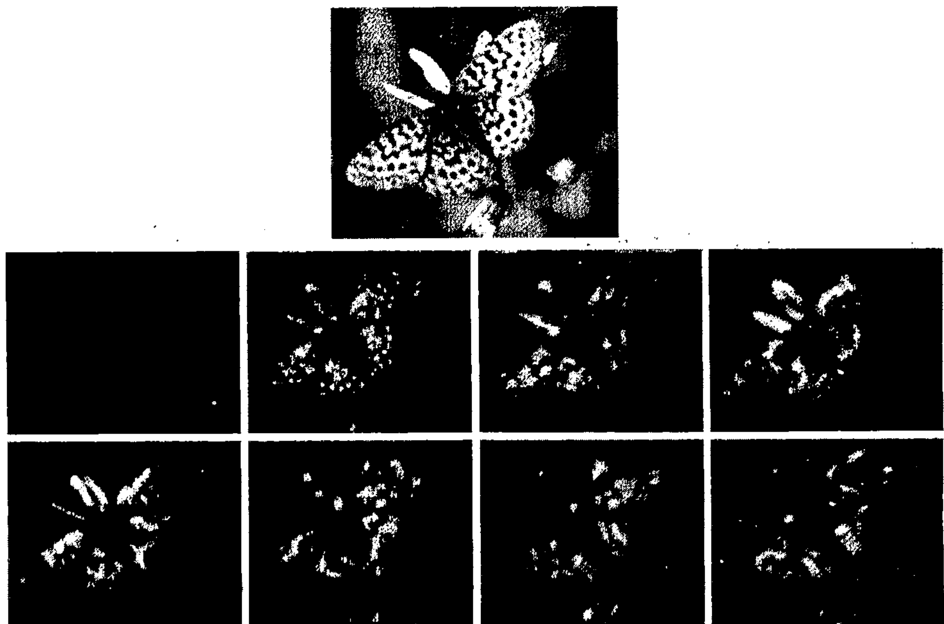

Figure 9.4 At the top, an image of a butterfly at a fine scale, and below, the result of applying each of the filters of Figure 9.3 to that image. The results are shown as absolute values of the output, lighter pixels representing stronger responses, and the images are laid out corresponding to the filter position in the top row.

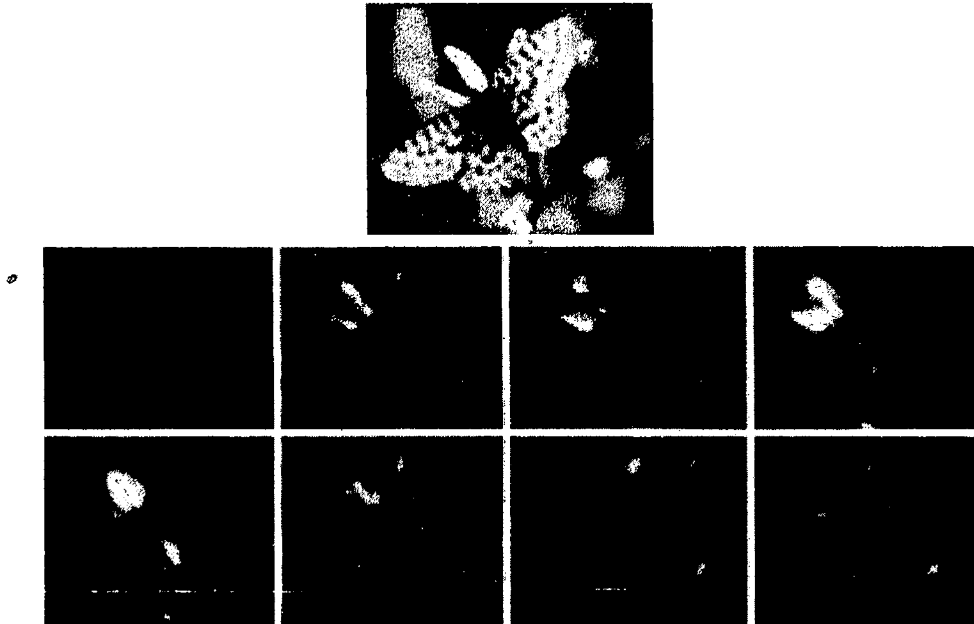

Figure 9.5 The input image of a butterfly and responses of the filters of Figure 9.3 at a coarser scale than that of Figure 9.4. Notice that the oriented bars respond to the bars on the wings, the antennae, and the edges of the wings; the fact that one bar has responded does not mean that another will not, but the size of the response is a cue to the orientation of the bar in the image.

detectors (because a bar filter at a particular orientation responds to bars of a variety of sizes and orientations), the filter outputs give a reasonable representation of the image data. Generally, bar filters respond strongly to oriented bars and weakly to other patterns, and the spot filter responds to isolated spots.

How Many Filters and at What Orientation? It is not known just how many filters are "best" for useful texture algorithms. Perona (1995) lists the number of scales and orientation used in a variety of systems; numbers run from four to eleven scales and from two to eighteen orientations. The number of orientations varies from application to application and does not seem to matter much, as long as there are at least about six orientations. Typically, the "spot" filters are Gaussians and the "bar" filters are obtained by differentiating oriented Gaussians.

Similarly, there does not seem to be much benefit in using more complicated sets of filters than the basic spot and bar combination. There is a tension here: using more filters leads to a more detailed (and more redundant representation of the image); but we must also convolve the image with all these filters, which can be expensive. One way to simplify the process is to control the amount of redundant information we deal with, by building a pyramid.

9.1.2 Representing Texture Using the Statistics of Filter Outputs

A set of filtered images, in itself, is not a representation of texture, because we need some representation of the overall distribution of texture elements. For example, a field of yellow flowers may consist of many small yellow spots, with some vertical green bars; a zebra may consist of black stripes on a white background. We are implicitly assuming a scale over which the texture is

being described. A small image window on a field of yellow flowers may contain only a flower; on a zebra it may consist of a constant black or white region. Similarly, a window that is too large may contain some background as well as the relevant texture. Notice that there are two scales here: firstly, the scale of the filters and secondly, the scale over which we consider the distribution of the filters.

Assume that we know the size of the relevant image window in which we wish to represent a texture. A typical representation involves a set of statistics of filter outputs for that window. Outputs are commonly squared (among other things, this has the advantage of counting black next to white stripes in the same way as white next to black stripes). For example, in Figure 9.6, we illustrate a putative representation in terms of horizontal and vertical textures. This representation is obtained by taking the outputs of horizontal (resp. vertical) bar filters and squaring them. We then smooth the result at a coarse scale. This smoothing is equivalent to estimating the mean of the squared filter outputs in some window. Finally, the smoothed outputs are passed to a classifier that describes the texture. In the example of Figure 9.6, the texture is placed in one of four classes, depending on whether the vertical or the horizontal output or both or neither are large.

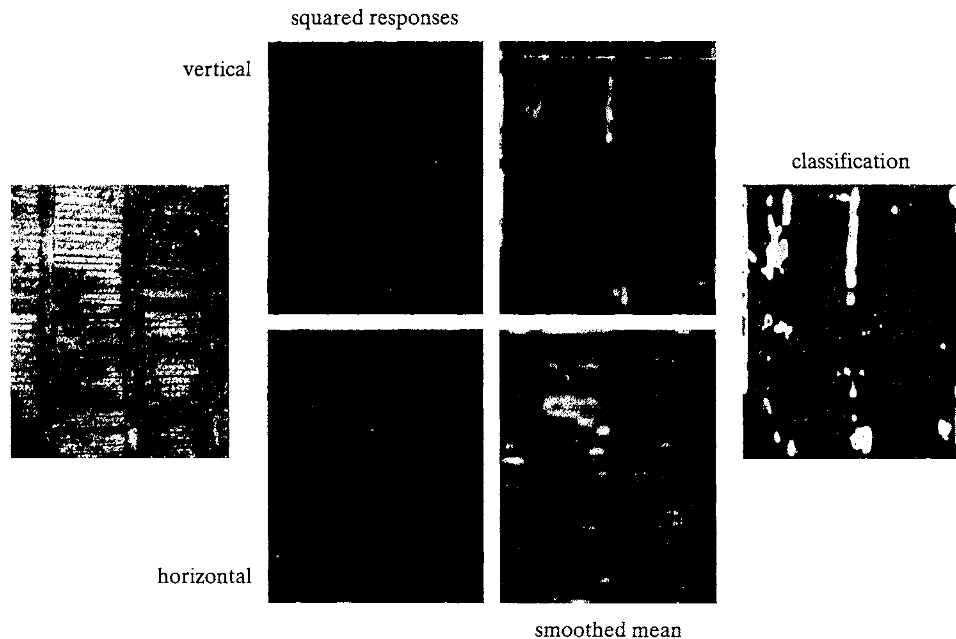

Figure 9.6 A putative texture representation in terms of filter outputs. We have sharply reduced the number of filters (there are two derivative filters, one vertical and one horizontal). The image on the **left** is the input; notice it has components that could reasonably be described as horizontal, vertical and fuzzy. The images in the **center left** column show the squared values of the filter outputs (which have been squared so that black-to-white transitions count the same as white-to-black transitions). The values are shown on the same linear scale, with lighter points indicating stronger responses. These have been smoothed to yield the images on the **center right** (which can be interpreted as the mean of the squared response over a small window). The mean response to vertical stripes is strong in the vertical map, and that to horizontal stripes in the horizontal map. Finally, we have thresholded these two images and combined them to get the image on the **right** (black values are neither horizontal nor vertical; dark grey values are horizontal; light grey values are vertical; and white values are "both").

The Choice of Statistic The question of *what* statistics should be collected depends to some extent on what we intend to represent. However, work on texture synthesis has indicated some constraints on appropriate choices of model, which is why we spend so much ink on the topic in Section 9.3. Assume, for the moment, that the scale of the window over which statistics should be collected has been set. One strategy is to compute the mean of the squared filter outputs for a range of filters (Malik and Perona, 1989). A window is then described by a vector of numbers, each of which is the mean of the squared response of some filter over the window. This approach can tell, for example, spotty windows—where the mean response of the spot filters will be high—from stripey windows—where the mean response of the bar filters will be high. This is the approach of Figure 9.6, but with a richer set of filters.

An alternative approach is to compute the mean and standard deviation of the filter outputs over the window, and use these for the feature vector (Ma and Manjunath, 1996). Texture descriptions of this form can be used to recover image windows based on examples (Figure 9.7). This is useful, because in a satellite image, whether a region depicts housing or vegetation can be determined from the texture. This means that, if we can match textures, we can find all regions of,

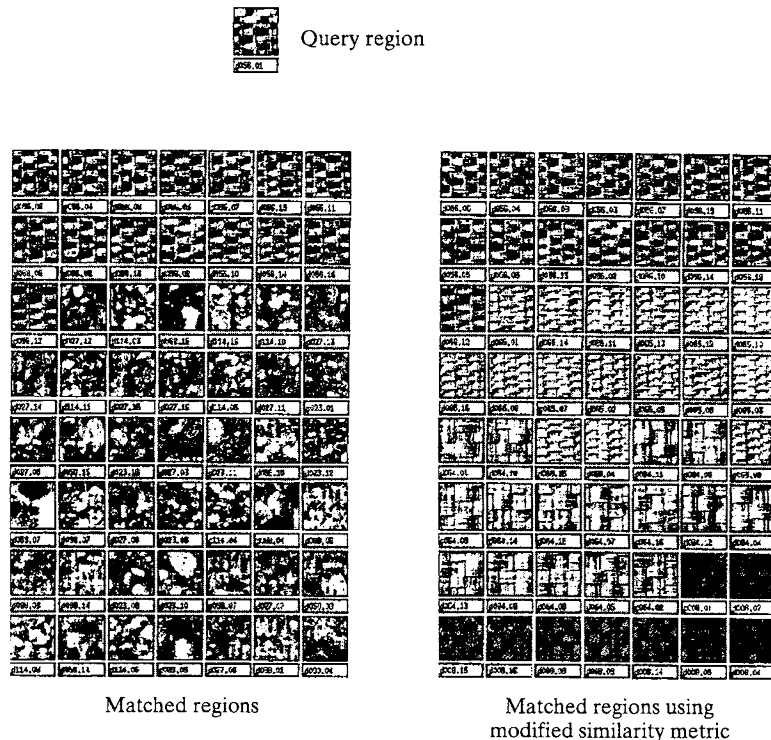

Figure 9.7 Textures can be represented as the mean and standard deviation of filter outputs taken over a window. If we use a collection of different filters, this yields a vector of numbers to represent the window; spotty textures will have large mean spot filter outputs, stripey textures will have large mean bar filter outputs, and so on. This means that an image window can be compared to others by computing a distance based on the feature vector. A pure Euclidean distance yields acceptable results (**left**), but modifying the distance function can yield very good results (**right**). *Reprinted from "Texture Features and Learning Similarity," by W.Y. Ma and B.S. Manjunath, Proc. IEEE Conf. Computer Vision and Pattern Recognition, 1996, © 1996, IEEE*

say, vegetation in a satellite image. Two textures with quite different feature vectors may appear to be similar; this can be dealt with by modifying the way that we measure differences between feature vectors.

Neither mean nor mean and standard deviation is an ideal representation, because the relationships between filter responses can be significant. For example, imagine a texture that consists of small spots arranged in stripes—an aerial view of a field of cabbages, say. In a texture like this, the small spot filter will respond strongly and the large bar filter will respond strongly, but *the responses are correlated*—the large bar filter responds where the small spot filter responds. In the case of a texture consisting of many large bars scattered around with small spots in the background, the small spot filter will respond strongly and the large bar filter will respond strongly, but the responses may not be correlated. We could try and record the covariance of the filter outputs—which would handle the example of the field of cabbages—but there will generally be too many terms to form an accurate estimate. Instead, one typically identifies some covariance terms that may be useful in a particular application and uses those.

The Choice of Scale Another question that is typically dealt with as a practical matter is the choice of the scale to use in representing a texture. Generally, one chooses a small window at the point of interest and then increases the size of the window until an increase does not cause a significant change. For example, imagine selecting a pixel on an image of a zebra. In a very small window around that pixel, the image has a constant value, say black. As the window gets slightly larger, there is a sharp change and there are some black and some white pixels in the window. Once the window is somewhat larger and contains several stripes, enlarging the window further will produce no significant change—it will just be a bigger, stripey window.

One statistic that can be used to determine when to stop expanding the window is the *polarity*. We first determine the *dominant orientation* of the window—the average direction of the gradient. Now for each gradient vector, we form the dot product between the gradient vector and the dominant orientation. We then form a smoothed average of the positive dot products and a smoothed average of the magnitude of the negative dot products, and take the difference of the two. This measures the extent to which gradients in a region point along the dominant orientation (positive dot products) vs. against the dominant orientation (negative dot products).

We can measure this statistic for any particular window scale. We do so for a range of window sizes, and then start at the finest scale and look at increasingly large windows until the polarity has not changed when the scale changed. Notice that there is some possibility that this criterion is not unique. For example, imagine a very high resolution photograph of a zebra. If we start with a sufficiently small window (which may span a single hair), this criterion will select a scale at which a window contains several hairs. If we start with a somewhat larger window, containing many hairs, the criterion should select a scale that encompasses several stripes.

9.2 ANALYSIS (AND SYNTHESIS) USING ORIENTED PYRAMIDS

Representing texture using the statistics of a series of filter outputs requires convolving the image with many filters at many scales. There are quite good methods for doing this systematically, which we describe in this section. Many readers may be content to leave this issue unexamined, and such readers should skip to Section 9.3.

The process of convolving an image with a range of filters is referred to as *analysis*; convolving an image with a collection of oriented filters is sometimes, rather loosely, described as *analyzing orientation* or *representing orientation*. The Gaussian pyramid (Section 7.7.1) is an example of image analysis by a bank of filters—in this case, smoothing filters. The Gaussian pyramid handles scale systematically by subsampling the image once it has been smoothed. This

say, vegetation in a satellite image. Two textures with quite different feature vectors may appear to be similar; this can be dealt with by modifying the way that we measure differences between feature vectors.

Neither mean nor mean and standard deviation is an ideal representation, because the relationships between filter responses can be significant. For example, imagine a texture that consists of small spots arranged in stripes—an aerial view of a field of cabbages, say. In a texture like this, the small spot filter will respond strongly and the large bar filter will respond strongly, but *the responses are correlated*—the large bar filter responds where the small spot filter responds. In the case of a texture consisting of many large bars scattered around with small spots in the background, the small spot filter will respond strongly and the large bar filter will respond strongly, but the responses may not be correlated. We could try and record the covariance of the filter outputs—which would handle the example of the field of cabbages—but there will generally be too many terms to form an accurate estimate. Instead, one typically identifies some covariance terms that may be useful in a particular application and uses those.

The Choice of Scale Another question that is typically dealt with as a practical matter is the choice of the scale to use in representing a texture. Generally, one chooses a small window at the point of interest and then increases the size of the window until an increase does not cause a significant change. For example, imagine selecting a pixel on an image of a zebra. In a very small window around that pixel, the image has a constant value, say black. As the window gets slightly larger, there is a sharp change and there are some black and some white pixels in the window. Once the window is somewhat larger and contains several stripes, enlarging the window further will produce no significant change—it will just be a bigger, stripey window.

One statistic that can be used to determine when to stop expanding the window is the *polarity*. We first determine the *dominant orientation* of the window—the average direction of the gradient. Now for each gradient vector, we form the dot product between the gradient vector and the dominant orientation. We then form a smoothed average of the positive dot products and a smoothed average of the magnitude of the negative dot products, and take the difference of the two. This measures the extent to which gradients in a region point along the dominant orientation (positive dot products) vs. against the dominant orientation (negative dot products).

We can measure this statistic for any particular window scale. We do so for a range of window sizes, and then start at the finest scale and look at increasingly large windows until the polarity has not changed when the scale changed. Notice that there is some possibility that this criterion is not unique. For example, imagine a very high resolution photograph of a zebra. If we start with a sufficiently small window (which may span a single hair), this criterion will select a scale at which a window contains several hairs. If we start with a somewhat larger window, containing many hairs, the criterion should select a scale that encompasses several stripes.

9.2 ANALYSIS (AND SYNTHESIS) USING ORIENTED PYRAMIDS

Representing texture using the statistics of a series of filter outputs requires convolving the image with many filters at many scales. There are quite good methods for doing this systematically, which we describe in this section. Many readers may be content to leave this issue unexamined, and such readers should skip to Section 9.3.

The process of convolving an image with a range of filters is referred to as *analysis*; convolving an image with a collection of oriented filters is sometimes, rather loosely, described as *analyzing orientation* or *representing orientation*. The Gaussian pyramid (Section 7.7.1) is an example of image analysis by a bank of filters—in this case, smoothing filters. The Gaussian pyramid handles scale systematically by subsampling the image once it has been smoothed. This

means that generating the next coarsest scale is easier, because we don't process redundant information.

In fact, the Gaussian pyramid is a highly redundant representation because each layer is a low pass filtered version of the previous layer—this means that we are representing the lowest spatial frequencies many times. A layer of the Gaussian pyramid is a prediction of the appearance of the next finer scale layer—this prediction isn't exact, but it means that it is unnecessary to store all of the next finer scale layer. We need keep only a record of the errors in the prediction. This is the motivating idea behind the *Laplacian pyramid.*

The Laplacian pyramid will yield a representation of various different scales that has fairly low redundancy, but it doesn't immediately deal with orientation. By thinking about pyramids in the Fourier domain, we obtain a method for encoding orientation as well (Section 9.2.2). In Section 9.2.3, we will sketch a method that obtains a representation of orientation as well.

9.2.1 The Laplacian Pyramid

The Laplacian pyramid makes use of the fact that a coarse layer of the Gaussian pyramid predicts the appearance of the next finer layer. If we have an upsampling operator that can produce a version of a coarse layer of the same size as the next finer layer, then we need only store the difference between this prediction and the next finer layer itself.

Clearly, we cannot create image information, but we can expand a coarse scale image by replicating pixels. This involves an upsampling operator $S^{\uparrow}$ which takes an image at level $n+1$ to an image at level n. In particular, $S^{\uparrow}(\mathcal{I})$ takes an image, and produces an image twice the size in each dimension. The four elements of the output image at $(2j-1, 2k-1)$; $(2j, 2k-1)$; $(2j-1, 2k)$; and $(2j, 2k)$ all have the same value as the j, kth element of $\mathcal{I}$.

Analysis—Building a Laplacian Pyramid from an Image The coarsest scale layer of a Laplacian pyramid is the same as the coarsest scale layer of a Gaussian pyramid. Each of the finer scale layers of a Laplacian pyramid is a difference between a layer of the Gaussian pyramid and a prediction obtained by upsampling the next coarsest layer of the Gaussian pyramid. This means that:

$$P_{\text{Laplacian}}(\mathcal{I})_m = P_{\text{Gaussian}}(\mathcal{I})_m$$

(where m is the coarsest level) and

$$\begin{aligned} P_{\text{Laplacian}}(\mathcal{I})_k &= P_{\text{Gaussian}}(\mathcal{I})_k - S^{\uparrow}(P_{\text{Gaussian}}(\mathcal{I})_{k+1}) \\ &= (Id - S^{\uparrow}S^{\downarrow}G_{\sigma})P_{\text{Gaussian}}(\mathcal{I})_k \end{aligned}$$

All this yields Algorithm 9.1. While the name "Laplacian" is somewhat misleading—there are no differential operators here—it is not outrageous, because each layer is approximately the result of a difference of Gaussian filter.

Each layer of the Laplacian pyramid can be thought of as the response of a band-pass filter. This is because we are taking the image at a particular resolution, and subtracting the components that can be predicted by a coarser resolution version—which corresponds to the low spatial frequency components of the image. This means in turn that we expect that an image of a set of stripes at a particular spatial frequency would lead to strong responses at one level of the pyramid and weak responses at other levels (Figure 9.8).

Because different levels of the pyramid represent different spatial frequencies, the Laplacian pyramid can be used as a reasonably effective image compression scheme.

Algorithm 9.1: Building a Laplacian Pyramid from an Image

Form a Gaussian pyramid
Set the coarsest layer of the Laplacian pyramid to be
  the coarsest layer of the Gaussian pyramid
For each layer, going from next to coarsest to finest
  Obtain this layer of the Laplacian pyramid by
    upsampling the next coarser layer, and subtracting
    it from this layer of the Gaussian pyramid
end

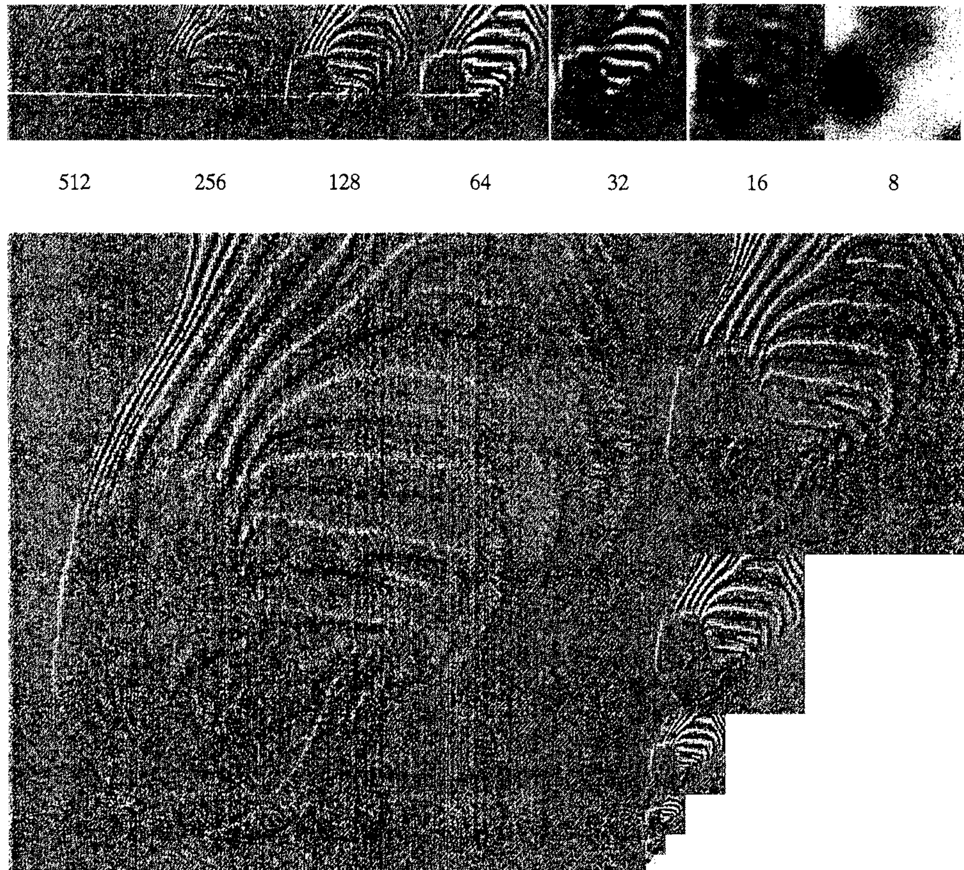

Figure 9.8 A Laplacian pyramid of images, running from 512x512 to 8×8. A zero response is coded with a mid-grey; positive values are lighter and negative values are darker. Notice that the stripes give stronger responses at particular scales, because each layer corresponds (roughly) to the output of a band-pass filter.

Synthesis—Recovering an Image from its Laplacian Pyramid Laplacian pyramids have one important feature. It is easy to recover an image from its Laplacian pyramid. We do this by recovering the Gaussian pyramid from the Laplacian pyramid, and then taking the finest scale of the Gaussian pyramid (which is the image). It is easy to get to the Gaussian pyramid from the Laplacian. Firstly, the coarsest scale of the Gaussian pyramid is the same as the coarsest scale of the Laplacian pyramid. The next-to-coarsest scale of the Gaussian pyramid is obtained by taking the coarsest scale, upsampling it, and adding the next-to-coarsest scale of the Laplacian pyramid (and so on up the scales). This process is known as *synthesis* and is described in Algorithm 9.2.

Algorithm 9.2: Synthesis: Obtaining an Image from a Laplacian Pyramid

```
Set the working image to be the coarsest layer
For each layer, going from next to coarsest to finest
  Upsample the working image and add the current layer
    to the result
  Set the working image to be the result of this operation
end
The working image now contains the original image
```

9.2.2 Filters in the Spatial Frequency Domain

The convolution theorem (that convolution in the spatial domain is the same as multiplication in the Fourier domain) yields some intuition about what filters do and what information pyramids contain. We shall illustrate this theorem by showing a natural analogy between smoothing and low-pass filtering; that some kinds of band-pass filter naturally respond to oriented structure; and that a form of local spatial frequency analysis can be obtained using a particular family of filters.

Smoothing and Low-Pass Filters The convolution theorem yields that convolving an image with an isotropic Gaussian with standard deviation σ is the same as multiplying the Fourier transform of the image by an isotropic Gaussian of standard deviation $1/\sigma$. Now a Gaussian falls off quite quickly, particularly if its standard deviation is large. This means that the Fourier transform of the result will have relatively little energy at high spatial frequencies, where a high spatial frequency is a few multiples of $1/\sigma$. We can interpret this as a *low-pass filter*—one that has a high gain for low spatial frequencies and a low gain for high spatial frequencies. This is quite a satisfactory interpretation: if we smooth with a Gaussian with a very small standard deviation, all but the highest spatial frequencies are preserved; and if we smooth with a Gaussian with a very large standard deviation, the result will be pretty much the average value of the image. This means that a Gaussian pyramid is, in essence, a set of low-pass filtered versions of the image.

Band-Pass Filters and Orientation Selective Operators A *band-pass filter* is one that has high gain for some range of spatial frequencies and a low gain for higher and for lower spatial frequencies. One type of band-pass filter is insensitive to orientation. A natural example of such a filter is to smooth an image with the difference of two isotropic Gaussians; one with a small standard deviation and one with a large standard deviation. In the Fourier domain, the kernel of this filter looks like an annulus of large values (the left half of Figure 9.9); this means that it selects a range of spatial frequencies, but is not selective to orientations (because points at

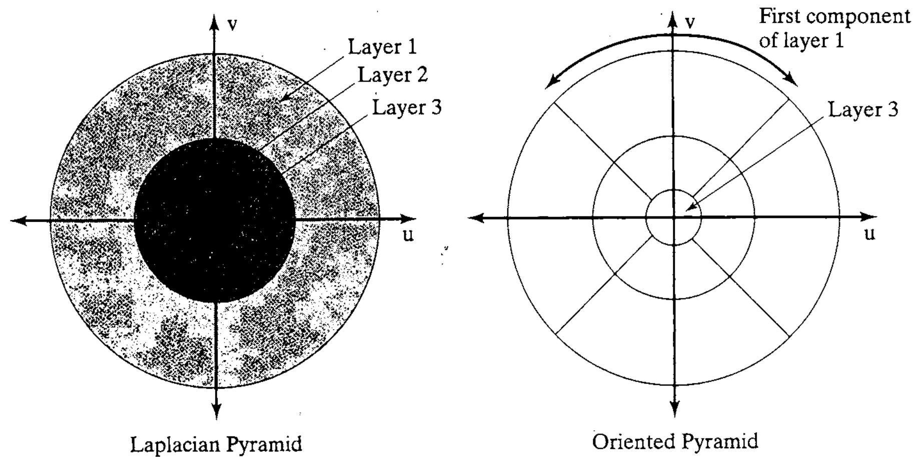

Figure 9.9 Each layer of the Laplacian pyramid consists the elements of a smoothed and resampled image that are not represented by the next smoother layer. Assuming that a Gaussian is a sufficiently good smoothing filter, each layer can be thought of as representing the image components within a range of spatial frequencies—this means that the Fourier transform of each layer is an annulus of values from the Fourier transform space (u, v) space (recall that the magnitude of (u, v) gives the spatial frequency). The sum of these annuluses is the Fourier transform of the image, so that each layer cuts an annulus out of the image's Fourier transform. An oriented pyramid cuts each annulus into a set of wedges. If (u, v) space is represented in polar coordinates, each wedge corresponds to an interval of radius values and an interval of angle values (recall that $\arctan(u/v)$ gives the orientation of the Fourier basis element).

the same distance from the origin in Fourier space refer to basis elements of the frequency, but at different orientations). While an ideal bandpass filter would have a unit value within the annulus and a zero value outside, such a filter would have infinite spatial support—making it difficult to work with—and the difference of Gaussians appears to be a satisfactory practical choice. Of course, this difference of Gaussians is the filter used to obtain the Laplacian pyramid, so the Laplacian pyramid consists of a set of band-pass filtered versions of the image.

An alternative type of band-pass filter has a Fourier transform that is large within a wedge of the annulus, and small outside (the right half of Figure 9.9)—this filter is *orientation selective*, meaning that it responds most strongly to signals that have a particular range of spatial frequencies *and* orientations.

Local Spatial Frequency Analysis and Gabor Filters One difficulty with the Fourier transform is that Fourier coefficients depend on the entire image; the value of the Fourier transform for some particular (u, v) is computed using every image pixel. This is an inconvenient way to think of images, because we have lost all spatial information. For example, the stripes of Figure 9.12 get wider as one moves across the image. If we think in terms of spatial frequency only locally defined, then we can think of this phenomenon in terms of the spatial frequency content of the image changing as we move across it. In some window around a point, the narrow stripes look like high spatial frequency terms and the wide stripes look like low spatial frequency terms.

Gabor filters achieve this. The kernels look like Fourier basis elements that are multiplied by Gaussians, meaning that a Gabor filter responds strongly at points in an image where there are

components that *locally* have a particular spatial frequency and orientation. Gabor filters come in pairs, often referred to as *quadrature pairs*; one of the pair recovers symmetric components in a particular direction, and the other recovers antisymmetric components. The mathematical form of the symmetric kernel is

$$G_{\text{symmetric}}(x, y) = \cos\left(k_x x + k_y y\right) \exp - \left\{ \frac{x^2 + y^2}{2\sigma^2} \right\}$$

and the antisymmetric kernel has the form

$$G_{\text{antisymmetric}}(x, y) = \sin\left(k_0 x + k_1 y\right) \exp - \left\{ \frac{x^2 + y^2}{2\sigma^2} \right\}$$

The filters are illustrated in Figures 9.10 and 9.11; (k_x, k_y) give the spatial frequency to which the filter responds most strongly, and σ is referred to as the *scale* of the filter. In principle, by applying a very large number of Gabor filters at different scales, orientations and spatial frequencies, one can analyze an image into a detailed local description. There is an analogy between Gabor filtering with $\sigma = \infty$ and a Fourier transform; this explains why there are two types of filter, and indicates why we can think of a Gabor filter as performing a local spatial frequency analysis.

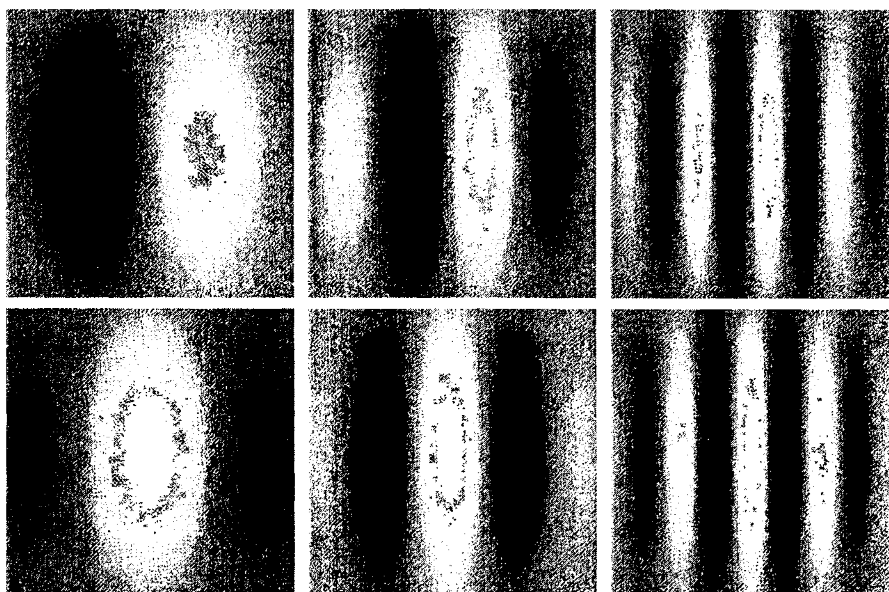

Figure 9.10 Gabor filter kernels are the product of a symmetric Gaussian with an oriented sinusoid; the form of the kernels is given in the text. The images show Gabor filter kernels as images, with mid-grey values representing zero, darker values representing negative numbers and lighter values representing positive numbers. The top row shows the antisymmetric component, and the bottom row shows the symmetric component. The symmetric and antisymmetric components have a phase difference of $\pi/2$ radians, because a cross-section perpendicular to the bar (horizontally, in this case) gives sinusoids that have this phase difference. The scale of these filters is constant, and they are shown for three different spatial frequencies. Figure 9.11 shows Gabor filters at a finer scale.

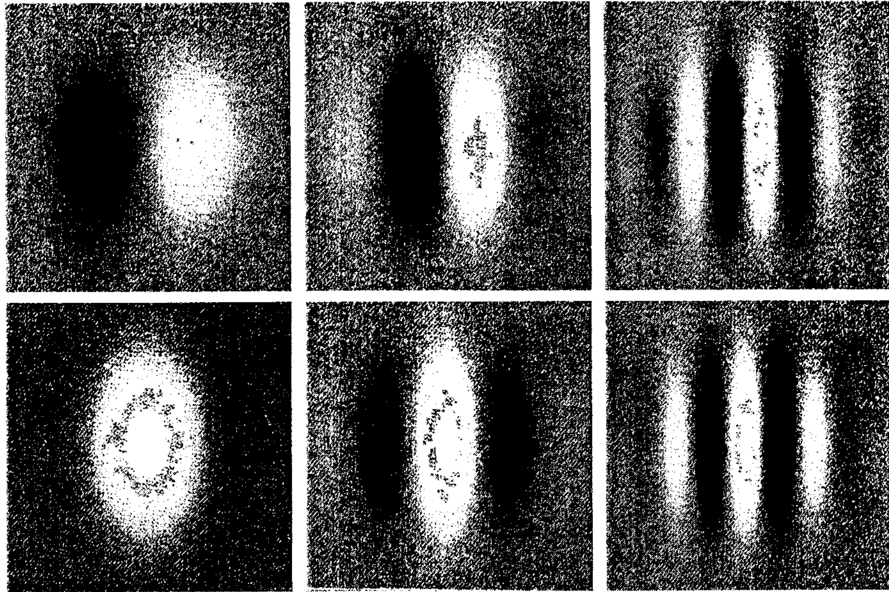

Figure 9.11 The images shows Gabor filter kernels as images, with mid-grey values representing zero, darker values representing negative numbers and lighter values representing positive numbers. The top row shows the antisymmetric component, and the bottom row shows the symmetric component. The scale of these filters is constant, and they are shown for three different spatial frequencies. These filters are shown at a finer scale than those of Figure 9.10.

9.2.3 Oriented Pyramids

A Laplacian pyramid does not contain enough information to reason about image texture, because there is no explicit representation of the orientation of the stripes. A natural strategy for dealing with this is to take each layer and decompose it further, to obtain a set of components each of which represents a energy at a distinct orientation. Each component can be thought of as the response of an oriented filter at a particular scale and orientation. The result is a detailed analysis of the image, known as an *oriented pyramid* (Figure 9.13).

A comprehensive discussion of the design of oriented pyramids would take us out of our way. The first design constraint is that the filter should select a small range of spatial frequencies and orientations, as in Figure 9.9. There is a second design constraint for our analysis filters: synthesis should be easy. If we think of the oriented pyramid as a decomposition of the Laplacian pyramid (Figure 9.14), then synthesis involves reconstructing each layer of the Laplacian pyramid, and then synthesizing the image from the Laplacian pyramid. The ideal strategy is to have a set of filters that have oriented responses *and* where synthesis is easy. It is possible to produce a set of filters such that reconstructing a layer from its components involves filtering the image a second time with the same filter (as Figure 9.15 suggests). An efficient implementation of these pyramids is available at `http://www.cis.upenn.edu/~eero/steerpyr.html`. The design process is described in detail in Karasaridis and Simoncelli (1996) and Simoncelli and Freeman (1995).

9.3 APPLICATION: SYNTHESIZING TEXTURES FOR RENDERING

Renderings of object models look more realistic if they are textured (it's worth thinking about why this should be true, even though the point is widely accepted as obvious). There are a va-

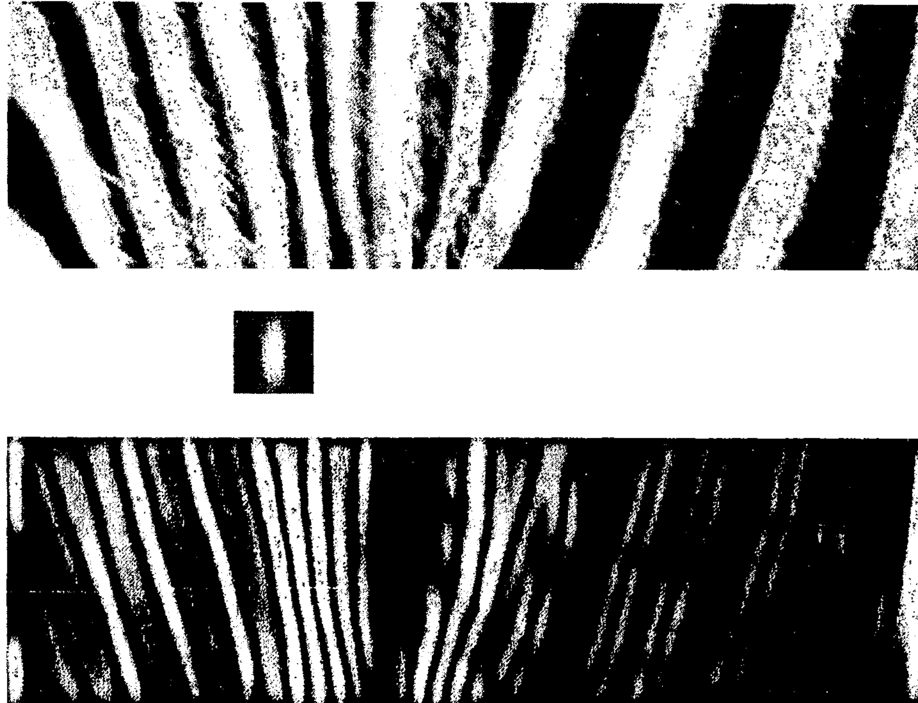

Figure 9.12 The image on the **top** shows a detail from an image of a zebra, chosen because it has a stripes at somewhat different scales and orientations. This has been convolved with the kernel in the center, which is a Gabor filter kernel. The image at the **bottom** shows the absolute value of the result; notice that the response is large when the spatial frequency of the bars roughly matches that windowed by the Gaussian in the Gabor filter kernel (i.e., the stripes in the kernel are about as wide as, and at about the same orientation as, the three stripes in the kernel). When the stripes are larger or smaller, the response falls off; thus, the filter is performing a kind of local spatial frequency analysis. This filter is one of a quadrature pair (it is the symmetric component). The response of the anti-symmetric component is similarly frequency selective. The two responses can be seen as the two components of the (complex valued) local Fourier transform, so that magnitude and phase information can be extracted from them.

riety of techniques for texture mapping; the basic idea is that when an object is rendered, the reflectance value used to shade a pixel is obtained by reference to a **texture map**. Some system of coordinates is adopted on the surface of the object to associate the elements of the texture map with points on the surface. Different choices of coordinate system yield renderings that look quite different, and it is not always easy to ensure that the texture lies on a surface in a natural way (for example, consider painting stripes on a zebra—where should the stripes go to yield a natural pattern?). Despite this issue, texture mapping seems to be an important trick for making rendered scenes look more realistic.

Texture mapping demands textures, and texture mapping a large object may require a substantial texture map. This is particularly true if the object is close to the view, meaning that the texture on the surface is seen at a high resolution, so that problems with the resolution of the texture map will become obvious. Tiling texture images can work poorly, because it can be difficult to obtain images that tile well—the borders have to line up, and even if they did, the

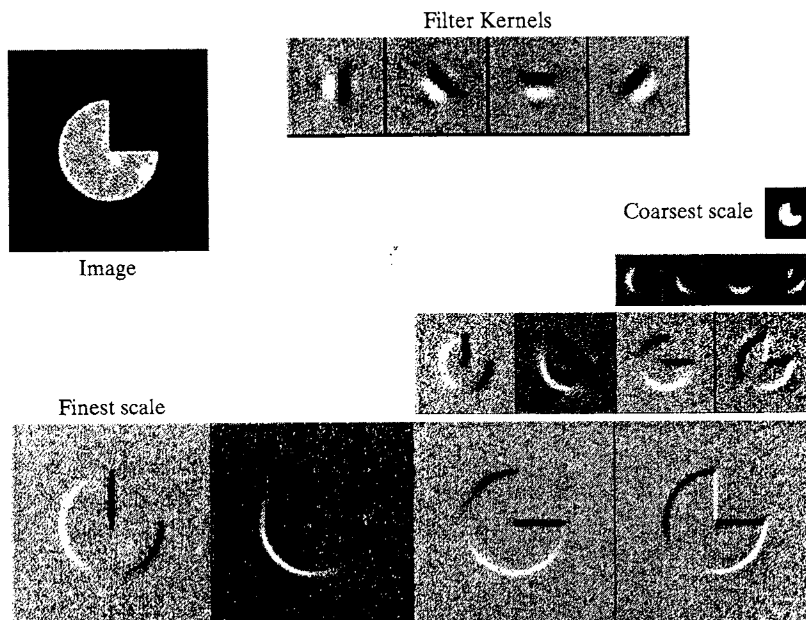

Figure 9.13 An oriented pyramid, formed from the image at the **top left**, with four orientations per layer. This is obtained by firstly decomposing an image into subbands which represent bands of spatial frequency (as with the Laplacian pyramid), and then applying oriented filters (**top right**) to these subbands to decompose them into a set of distinct images, each of which represents the amount of energy at a particular scale and orientation in the image. Notice how the orientation layers have strong responses to the edges in particular directions, and weak responses at other directions. Code for constructing oriented pyramids, written and distributed by Eero Simoncelli, can be found at `http://www.cis.upenn.edu/~eero/steerpyr.html`. *Reprinted from "Shiftable MultiScale Transforms," by Simoncelli et al., IEEE Transactions on Information Theory, 1992, © 1992, IEEE*

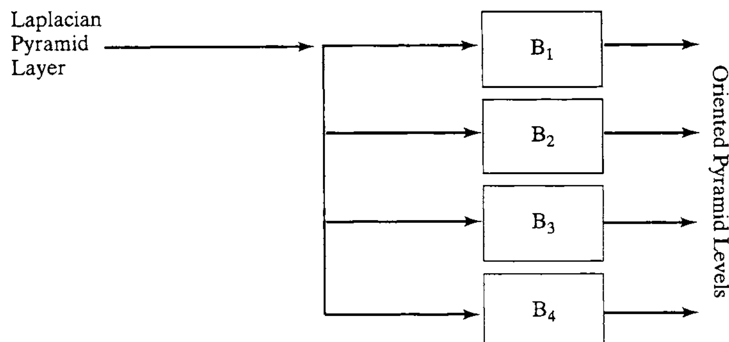

Figure 9.14 The oriented pyramid is obtained by taking layers of the Laplacian pyramid, and then applying oriented filters (represented in this schematic drawing by boxes). Each layer of the Laplacian pyramid represents a range of spatial frequencies; the oriented filters decompose this range of spatial frequencies into a set of orientations.

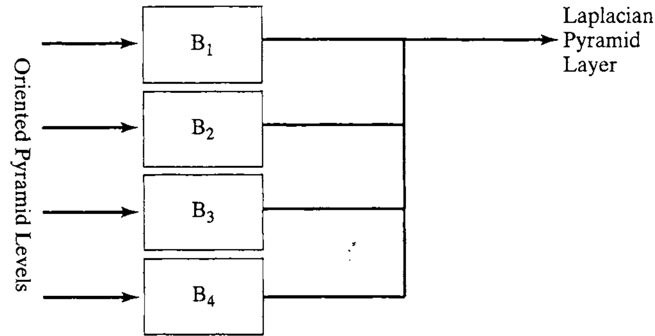

Figure 9.15 In the oriented pyramid, synthesis is possible by refiltering the layers and then adding them, as this schematic indicates. This property is obtained by appropriate choice of filters.

resulting periodic structure can be annoying. It is possible to buy image textures from a variety of sources, but an ideal would be to have a program that can generate large texture images from a small example. Quite sophisticated programs of this form can be built, and they illustrate the usefulness of representing textures by filter outputs.

9.3.1 Homogeneity

The general strategy for texture synthesis is to think of a texture as a sample from some probability distribution and then to try and obtain other samples from that same distribution. To make this approach practical, we need to obtain a probability model from the sample texture. The first thing to do is assume that the texture is *homogenous*. This means that local windows of the texture "look the same", from wherever in the texture they were drawn. More formally, the probability distribution on values of a pixel is determined by the properties of some neighborhood of that pixel, rather than by, say, the position of the pixel.

An assumption of homogeneity means that we can construct a model for the texture outside the boundaries of our example region, based on the properties of our example region. The assumption often applies to natural textures over a reasonable range of scales. For example, the stripes on a zebra's back are homogenous, but remember that those on its back are vertical and those on its legs, horizontal. We can use the example texture to obtain the probability model for the synthesized texture in various ways; we describe only one here.

9.3.2 Synthesis by Sampling Local Models

As Efros and Leung (1999) point out, the example image can serve as a a probability model. Assume for the moment that we have every pixel in the synthesized image, except one. To obtain a probability model for the value of that pixel, we could match a neighborhood of the pixel to the example image. Every matching neighborhood in the example image has a possible value for the pixel of interest. This collection of values is a conditional histogram for the pixel of interest. By drawing a sample uniformly and at random from this collection, we obtain the value that is consistent with the example image.

Finding Matching Image Neighbourhoods The essence of the matter is to take some form of neighbourhood around the pixel of interest, and to compare it to neighbourhoods in the example image. The size and shape of this neighbourhood is significant, because it codes

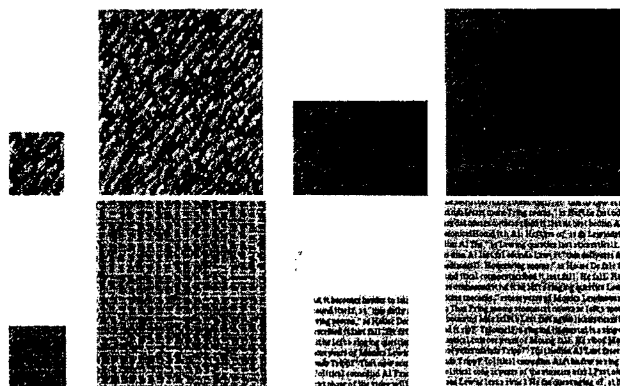

Figure 9.16 Efros' texture synthesis algorithm (Algorithm 9.3) matches neighbourhoods of the image being synthesized to the example image, and then chooses at random amongst the possible values reported by matching neighbourhoods. This means that the algorithm can reproduce complex spatial structures, as these examples indicate. The small block on the **left** is the example texture; the algorithm synthesizes the block on the **right**. Note that the synthesized text looks like text; it appears to be constructed of words of varying lengths that are spaced like text; and each word looks as though it is composed of letters (though this illusion fails as one looks closely). *Figure from Texture Synthesis by Non-parametric Sampling, A. Efros and T.K. Leung, Proc. Int. Conf. Computer Vision, 1999 © 1999, IEEE*

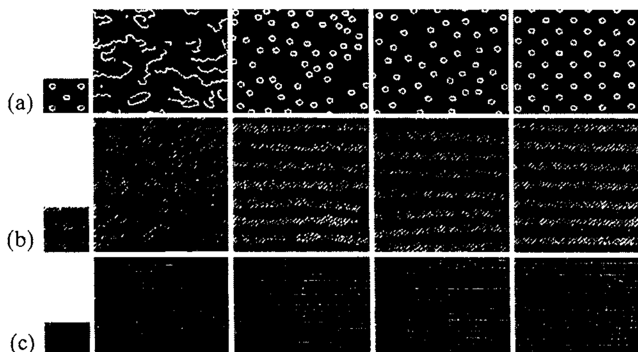

Figure 9.17 The size of the image neighbourhood to be matched makes a significant difference in Algorithm 9.3. In the figure, the textures at the right are synthesized from the small blocks on the **left**, using neighbourhoods that are increasingly large as one moves to the **right**. If very small neighbourhoods are matched, then the algorithm cannot capture large scale effects easily. For example, in the case of the spotty texture, if the neighbourhood is too small to capture the spot structure (and so sees only pieces of curve), the algorithm synthesizes a texture consisting of curve segments. As the neighbourhood gets larger, the algorithm can capture the spot structure, but not the even spacing. With very large neighbourhoods, the spacing is captured as well. *Figure from Texture Synthesis by Non-parametric Sampling, A. Efros and T.K. Leung, Proc. Int. Conf. Computer Vision, 1999 © 1999, IEEE*

the range over which pixels can affect one another's values directly (see Figure 9.17). Efros uses a square neighborhood, centered at the pixel of interest.

The similarity between two image neighbourhoods can be measured by forming the sum of squared differences of corresponding pixel values. This value is small when the neighbourhoods are similar, and large when they are different (it is essentially the length of the difference vector). Of course, the value of the pixel to be synthesized is not counted in the sum of squared differences.

Synthesizing Textures using Neighbourhoods Now we know how to obtain the value of a single missing pixel: choose uniformly and at random amongst the values of pixels in the example image whose neighborhoods match the neighbourhood of our pixel (i.e., where the sum of squared differences between the two neighbour hoods is smaller than some threshold).

Generally, we need to synthesize more than just one pixel. Usually, the values of some pixels in the neighborhood of the pixel to be synthesized are not known—these pixels need to be synthesized too. One way to obtain a collection of examples for the pixel of interest is to count only the known values in computing the sum of squared differences, and to adjust the threshold pro rata. The synthesis process can be started by choosing a block of pixels at random from the example image, yielding Algorithm 9.3.

Algorithm 9.3: Non-parametric Texture Synthesis

```
Choose a small square of pixels at random from the example image
Insert this square of values into the image to be synthesized
Until each location in the image to be synthesized has a value
  For each unsynthesized location on
    the boundary of the block of synthesized values
    Match the neighborhood of this location to the
      example image, ignoring unsynthesized
      locations in computing the matching score
    Choose a value for this location uniformly and at random
      from the set of values of the corresponding locations in the
      matching neighborhoods
  end
end
```

9.4 SHAPE FROM TEXTURE

A patch of texture of viewed frontally looks very different from a same patch viewed at a glancing angle, because foreshortening causes the texture elements (and the gaps between them!) to shrink more in some directions than in others. This suggests that we can recover some shape information from texture, at the cost of supplying a texture model. This is a task at which humans excel (Figure 9.18). Remarkably, quite general texture models appear to supply enough information to infer shape. This is most easily seen for planes (Section 9.4.1); while the details remain opaque in the case of curved surfaces, the general issues remain the same.

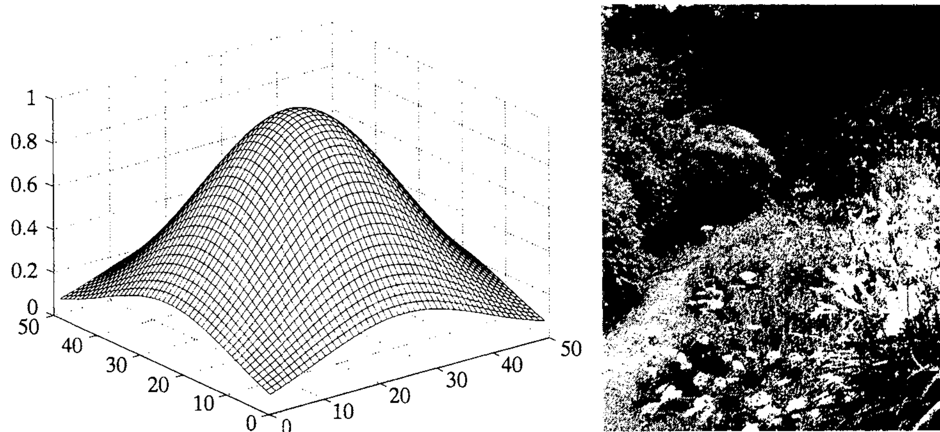

Figure 9.18 Humans obtain information about the shape of surfaces in space from the appearance of the texture on the surface. The figure on the **left** shows one common use for this effect—away from the contour regions, our only source of information about the surface depicted is the distortion of the texture on the surface. On the **right**, the texture of the bushes gives a sense they form rounded surfaces.

9.4.1 Shape from Texture for Planes

If we know we are viewing a plane, shape from texture boils down to determine the configuration of the plane relative to the camera. Assume that we hypothesize a configuration; we can then project the image texture back onto that plane. If we have some model of the "uniformity" of the texture, then we can test that property for the backprojected texture. We now obtain the plane with the "best" backprojected texture on it. This general strategy works for a variety of texture models. We will confine our discussion to the case of an orthographic camera. If the camera is not orthographic, the arguments we use will go through, but require substantially more work and more notation. We discuss other cases in the commentary.

Representing a Plane Now assume that we are viewing a single textured plane in an orthographic camera. Because the camera is orthographic, there is no way to measure the depth to the plane. However, we can think about the orientation of the plane. Let us work in terms of the camera coordinate system. We need to know firstly, the angle between the normal of the textured plane and the viewing direction—sometimes called the *slant*—and secondly, the angle the projected normal makes in the camera coordinate system—sometimes called the *tilt* (Figure 9.19). In an image of a plane, there is a *tilt direction*—the direction in the plane parallel to the projected normal.

Isotropy Assumptions An *isotropic* texture is one where the probability of encountering a texture element does not depend on the orientation of that element. This means that a probability model for an isotropic texture need not depend on the orientation of the coordinate system on the textured plane.

If we assume that the texture is isotropic, both slant and tilt can be read from the image. We could synthesize an orthographic view of a textured plane by first rotating the coordinate system by the tilt and then secondly contracting along one coordinate direction by the cosine of the slant—call this process a **viewing transformation**. The easiest way to see this is to assume

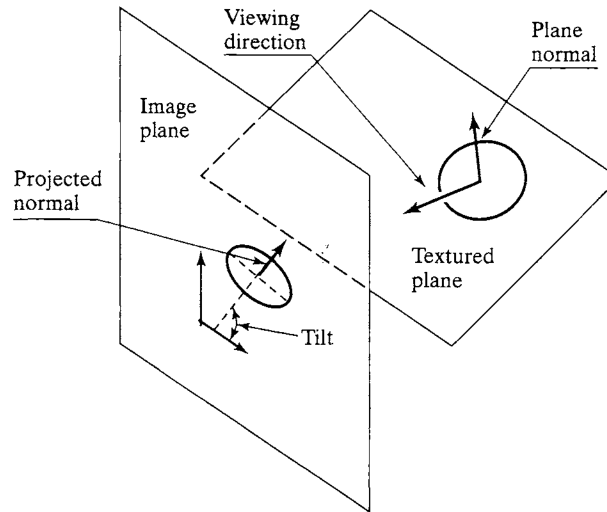

Figure 9.19 The orientation of a plane with respect to the camera plane can be given by the slant—which is the angle between the normal of the textured plane and the viewing direction—and the tilt—which is the angle the projected normal makes with the camera coordinate system. The figure illustrates the tilt, and shows a circle projecting to an ellipse.

that the texture consists of a set of circles, scattered about the plane. In an orthographic view, these circles will project to ellipses, whose minor axes will give the tilt, and whose aspect ratios will give the slant (see the exercises and Figure 9.19).

An orthographic view of an isotropic texture is *not* isotropic (unless the plane is parallel to the image plane). This is because the contraction in the slant direction interferes with the isotropy of the texture. Elements that point along the contracted direction get shorter. Furthermore, elements that have a component along the contracted direction have that component shrunk. Now corresponding to a viewing transformation is an **inverse viewing transformation** (which turns an image plane texture into the object plane texture, given a slant and tilt). This yields a strategy for determining the orientation of the plane: find an inverse viewing transformation that turns the image texture into an isotropic texture, and recover the slant and tilt from that inverse viewing transformation.

There are variety of ways to find this viewing transformation. One natural strategy is to use the energy output of a set of oriented filters. This is the squared response, summed over the image. For an isotropic texture, we would expect the energy output to be the same for each orientation at any given scale, because the probability of encountering a pattern does not depend on its orientation. Thus, a measure of isotropy is the standard deviation of the energy output as a function of orientation. We could sum this measure over scales, perhaps weighting the measure by the total energy in the scale. The smaller the measure, the more isotropic the texture. We now find the inverse viewing transformation that makes the image looks most isotropic by this measure, using standard methods from optimization.

Notice that this approach immediately extends to perspective projection, spherical projection, and other types of viewing transformation. We simply have to search over a larger family of transformations for the transformation that makes the image texture look most isotropic. One does need to be careful, however. For example, scaling an isotropic texture will lead to another isotropic texture, meaning that it isn't possible to recover a scaling parameter, and it's a bad idea

to try. The main difficulty with using an assumption of isotropy to recover the orientation of a plane is that there are very few isotropic textures in the world.

Homogeneity Assumptions It isn't possible to recover the orientation of a plane in an orthographic view by assuming that the texture is homogeneous (the definition is in Section 9.3.1). This is because the viewing transformation takes one homogeneous texture into another homogeneous texture. However, if we assume that the view is perspective, it becomes possible.

To see this, first notice that homogeneity means that, if a large even grid is imposed on the plane, the number of events that occur in each box should be (approximately) the same. For example, if a texture consists of a homogenous pattern of spots, the expected number of spots per box is the same for any box. However, if we were to see a perspective view of a textured plane with a grid superimposed, then some grid elements would project to large quadrilaterals and others to very small quadrilaterals (unless the view is frontal). In turn, this means that the projected texture cannot be homogenous *in the image plane*—because some elements in a grid of boxes on the image plane will have many projected quadrilaterals and hence many texture events in them, and others will have few. For the example of the spotted plane, this just means that the spots that project close to the plane's horizon appear small. The appropriate strategy is now to choose a transformation that will make the image plane texture "most homogenous"; notice that we can determine the orientation of the plane with respect to the camera plane, but not its depth, because a frontal view of a homogenous texture is homogenous—everything scales by the same amount.

9.5 NOTES

We have aggressively compressed the texture literature in this chapter. Over the years, there have been a wide variety of techniques for representing image textures, typically looking at the statistics of how patterns lie with respect to one another. The disagreements are in how a pattern should be described, and what statistics to look at. While it is a bit early to say that the approach that represents patterns using linear filters is correct, it is currently dominant, mainly because it is very easy to solve problems with this strategy. Readers who are seriously interested in texture will probably most resent our omission of the Markov Random Field model, a choice based on the amount of mathematics required to develop the model and the absence of satisfactory inference algorithms for MRF's. We refer the interested reader to Chellappa and Jain (1993), Cross and Jain (1983), Manjunath and Chellappa (1991), or Speis and Healey (1996).

Another important omission is the discussion of wavelet methods for representing texture. While these methods follow the rather rough lines given above—represent a texture by thinking about the output of a lot of filters—there is a comprehensive theory behind those filters. We refer the interested reader to Ma and Manjunath (1995), (1996) or Manjunath and Ma (1996*b,c*).

Filters, Pyramids and Efficiency

If we are to represent texture with the output of a large range of filters at many scales and orientations, then we need to be efficient at filtering. This is a topic that has attracted much attention; the usual approach is to try and construct a tensor product basis that represents the available families of filters well. With an appropriate construction, we need to convolve the image with a small number of separable kernels, and can estimate the responses of many different filters by combining the results in different ways (hence the requirement that the basis be a tensor product). Significant papers include Freeman and Adelson (1991), Greenspan, Belongie, Perona, Good-

man, Rakshit and Anderson (1994), Hel-Or and Teo (1996), Perona (1992), (1995), Simoncelli and Farid (1995), and Simoncelli and Freeman (1995).

Texture Synthesis

Texture synthesis exhausted us long before we could exhaust it. The most significant omission, apart from MRF's, is the work of Zhu, Wu and Mumford (1998), which uses sophisticated entropy criteria to firstly choose filters by which to represent a texture and secondly construct probability models for that texture.

Shape from Texture

There are surprisingly few methods for recovering a surface model from a projection of a texture field that is assumed to lie on that surface. **Global methods** attempt to recover an entire surface model, using assumptions about the distribution of texture elements. Appropriate assumptions are **isotropy** (Witkin, 1981) (the disadvantage of this method is that there are relatively few natural isotropic textures) or **homogeneity** (Aloimonos, 1986, Blake and Marinos, 1990). Methods based around homogeneity assume that texels are the result of a homogenous Poisson point process on a plane; the gradient of the density of the texel centers then yields the plane's parameters. However, deformation of individual texture elements is not accounted for.

Local methods recover some differential geometric parameters at a point on a surface (typically, normal and curvatures). This class of methods, which is due to Garding (1992), has been successfully demonstrated for a variety of surfaces by Malik and Rosenholtz (1997) and Rosenholtz and Malik (1997); a reformulation in terms of wavelets is due to Clerc and Mallat (1999). The methods have a crucial flaw; it is necessary either to know that texture element coordinate frames form a frame field that is locally parallel around the point in question, or to know the differential rotation of the frame field (see Garding, 1995 for this point, which is emphasized by the choice of textures displayed in Rosenholtz and Malik, 1997; the assumption is known as **texture stationarity**). For example, if one were to use these methods to recover the curvature of a doughnut dipped in chocolate sprinkles, it would be necessary to ensure that the sprinkles were all parallel on the surface (or that the field of angles from sprinkle to sprinkle was known). As a result, the method can be demonstrated to work only on quite a small class of textured surfaces. A second, important, difficulty lies in the data recovered; these methods all make local estimates of normal *and curvature*. But curvature is a derivative of the normal; as a result, while one local estimate may be helpful, there is no reason to believe that a collection of local estimates will be consistent. This is a problem of **integrability**. Surface interpolation methods have largely fallen out of fashion in computer vision, due to the uncertainty regarding the semantic status of surface patches in regions where data is absent. Shape from texture is a problem where an interpolate has an unquestionably useful role—it expresses the fact that, because one has a prior belief that surfaces are relatively slowly changing, incomplete local measurements of the surface normal can constrain one another and lead to good global estimates of the normal at some points.

PROBLEMS

9.1. Show that a circle appears as an ellipse in an orthographic view, and that the minor axis of this ellipse is the tilt direction. What is the aspect ratio of this ellipse?

9.2. We will study measuring the orientation of a plane in an orthographic view, given the texture consists of points laid down by a homogenous Poisson point process. Recall that one way to generate points

according to such a process is to sample the x and y coordinate of the point uniformly and at random. We assume that the points from our process lie within a unit square.

(a) Show that the probability that a point will land in a particular set is proportional to the area of that set.

(b) Assume we partition the area into disjoint sets. Show that the number of points in each set has a multinomial probability distribution.

We will now use these observations to recover the orientation of the plane. We partition the *image texture* into a collection of disjoint sets.

(c) Is the area of each set, *backprojected onto the textured plane*, a function of the orientation of the plane?

(d) Is it possible to determine the plane's orientation using this information? Use the result of (c).

Programming Assignments

9.3. Texture synthesis: Implement the non-parametric texture synthesis algorithm of Section 9.3.2. Use your implementation to study:

(a) the effect of window size on the synthesized texture;

(b) the effect of window shape on the synthesized texture;

(c) the effect of the matching criterion on the synthesized texture (i.e., using weighted sum of squares instead of sum of squares, etc.).

9.4. Texture representation: Implement a texture classifier that can distinguish between at least six types of texture; use the scale selection mechanism of Section 9.1.2, and compute statistics of filter outputs. We recommend that you use at least the mean and covariance of the outputs of about six oriented bar filters and a spot filter. You may need to read up on classification in chapter 22; use a simple classifier (nearest neighbor using Mahalanobis distance should do the trick).