Chapter 11

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.

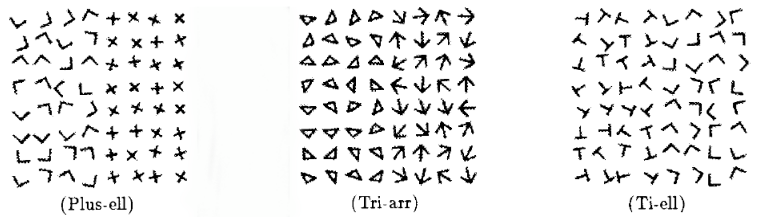


Figure 11.1. A set of texture examples, used in experiments with human subjects to tell how easily various types of textures can be discriminated. Note that these textures are made of quite stylised subelements, repeated in a meaningful way. *figure from the Malik and Perona, A Computational Model of Texture Segmentation, p.331, in the fervent hope, etc.*

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 11.1); chapters **??** and 18 show how to segment textured images using this representation.

- **Texture synthesis** seeks to construct large regions of texture from small

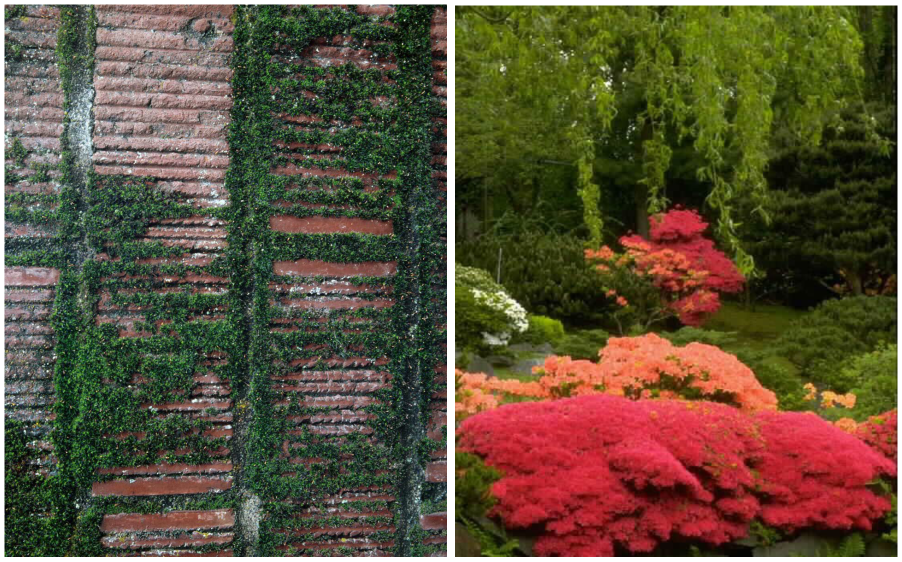


Figure 11.2. Typical textured images. For materials such as brush, grass, foliage and water, our perception of what the material is is quite intimately related to the texture (for the figure on the **left**, what would the surface feel like if you ran your fingers over it? is it wet?, etc.). Notice how 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 stylised subelements, arranged in a rough pattern.

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 11.3.

- **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 the shape of the surface. In section 11.4, we describe the main lines of reasoning in this (rather technical) area.

11.1 Representing Texture

Image textures generally consist of organised patterns of quite regular subelements (sometimes called **textons**). For example, one texture in figure 11.1 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 arrowheads 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.

11.1.1 Extracting Image Structure with Filter Banks

In section 10.1, 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 neighbourhood looks similar to the filter kernel, and a weak response when it doesn't.

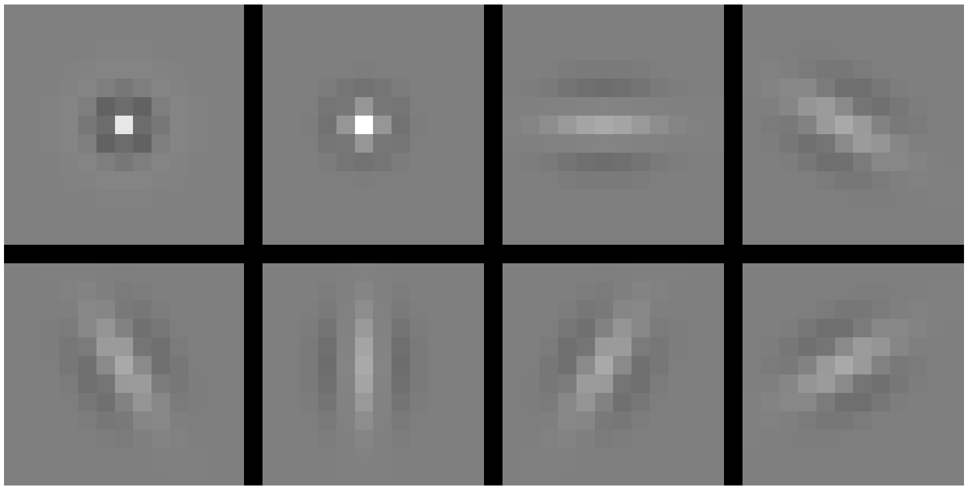


Figure 11.3. A set of eight filters used for expanding images into a series of responses. These filters are shown at a fixed scale, with zero represented by a mid-grey level, lighter values being positive and darker values being negative. They represent two distinct spots, and six bars; the set of filters is that used by [Malik and Perona, 1990].

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. The process of convolving an image with a range of filters is referred to as **analysis**.

Generally, spot filters are useful because they respond strongly to small regions that differ from their neighbours (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 (this property is sometimes, rather loosely, described as **analysing orientation** or **representing orientation**).

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^o.

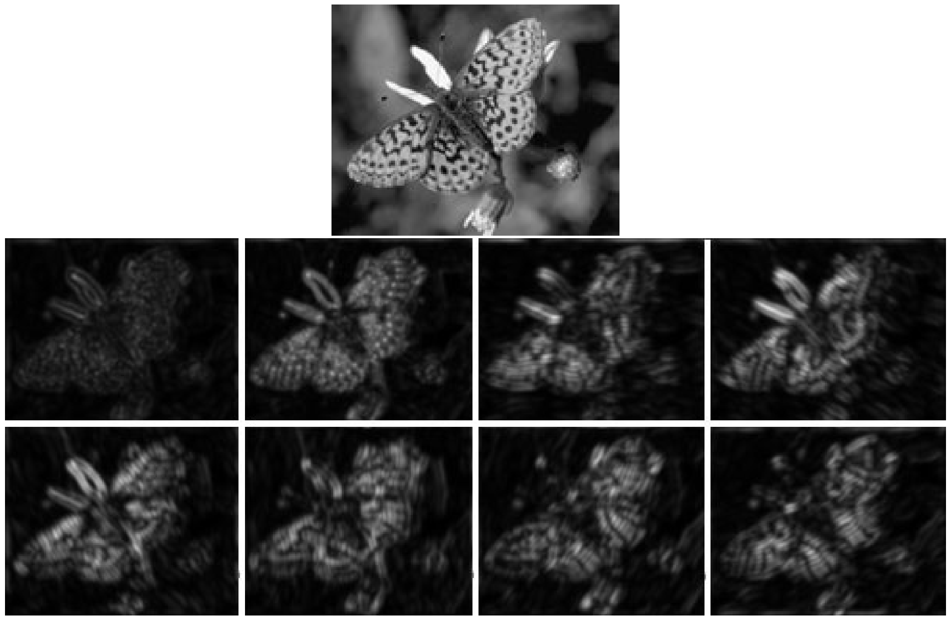


Figure 11.4. At the top, an image of a butterfly at a fine scale, and below, the result of applying each of the filters of figure 11.3 to that image. The results are shown as absolute values of the output, lighter pixels representing stronger responses, and the images are laid out corresponding to the filter position in the top row.

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 11.3. 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 $(0, -1)$.

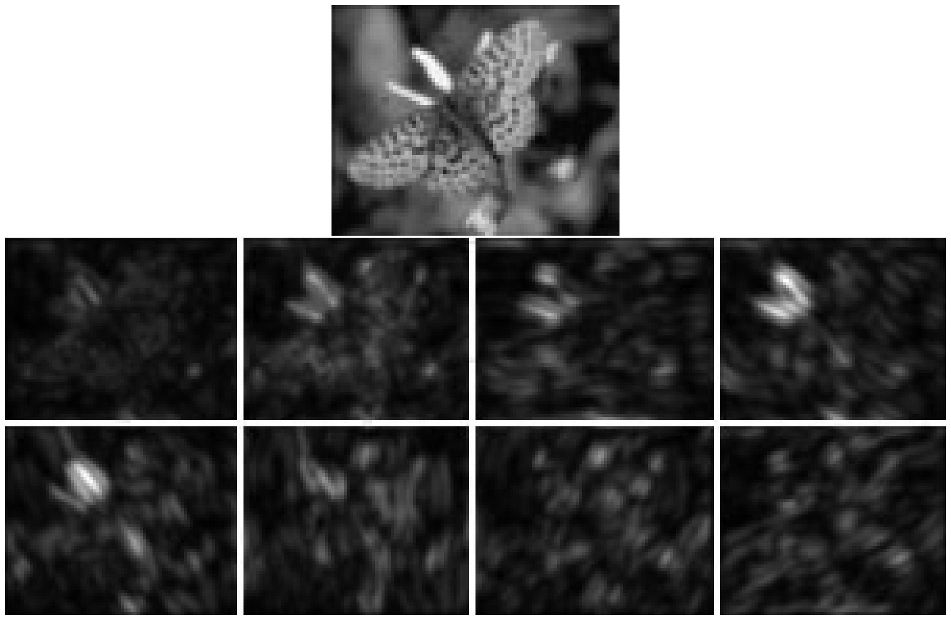


Figure 11.5. The input image of a butterfly and responses of the filters of figure 11.3 at a coarser scale than that of figure 11.4. Notice that the oriented bars respond to the bars on the wings, the antennae, and the edges of the wings; the fact that one bar has responded does not mean that another will not, but the size of the response is a cue to the orientation of the bar in the image.

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 optimising the choice of filters produces any major advantage.

Figures 11.4 and 11.5 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 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.

This, in itself, is not a representation of texture, because we need some representation of the overall distribution of spots and bars. Typical representations involve various statistics of filter outputs. For example, in figure 11.6, we illustrate a putative representation in terms of horizontal and vertical textures, which is obtained by looking at a smoothed local mean of the filter outputs. 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). The question of *what* statistics should be collected depends to some extent on what we intend to represent. Typically, we wish to represent the texture by some form of probability model, and the choice of this model determines the choice of statistics. What the probability model should be is, again, hard to determine from first principles. 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 **??**.

How many Filters and at what Orientation?

It is not known just how many filters are "best" for useful texture algorithms. Perona 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.

11.2 Analysis (and Synthesis) Using Oriented Pyramids

Analysing images using filter banks presents a computational problem — we have to convolve an image with a large number of filters at a range of scales. The computational demands can be simplified by handling scale and orientation systematically. The Gaussian pyramid (section 8.5.1) 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.

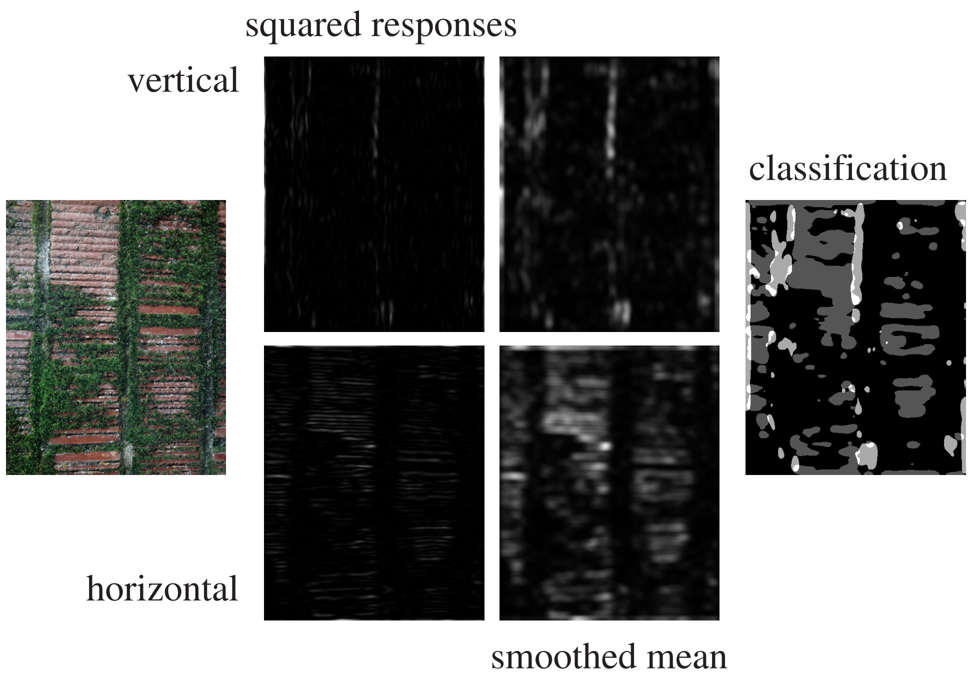


Figure 11.6. A putative texture representation in terms of filter outputs. We have sharply reduced the number of filters (there are two derivative filters, one vertical and one horizontal). The image on the **left** is the input; notice it has components that could reasonably be described as horizontal, 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").

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 11.2.2). In section 11.2.3, we will sketch a method that obtains a representation of orientation as well.

11.2.1 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, k'th 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

$$\begin{aligned} P_{\text{Laplacian}}(\mathcal{I})_k &= P_{\text{Gaussian}}(\mathcal{I})_k - S^{\uparrow}(P_{\text{Gaussian}}(\mathcal{I})_{k+1}) && (11.2.1) \\ &= (Id - S^{\uparrow}S^{\downarrow}G_{\sigma})P_{\text{Gaussian}}(\mathcal{I})_k && (11.2.2) \end{aligned}$$

All this yields algorithm 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 11.7).

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

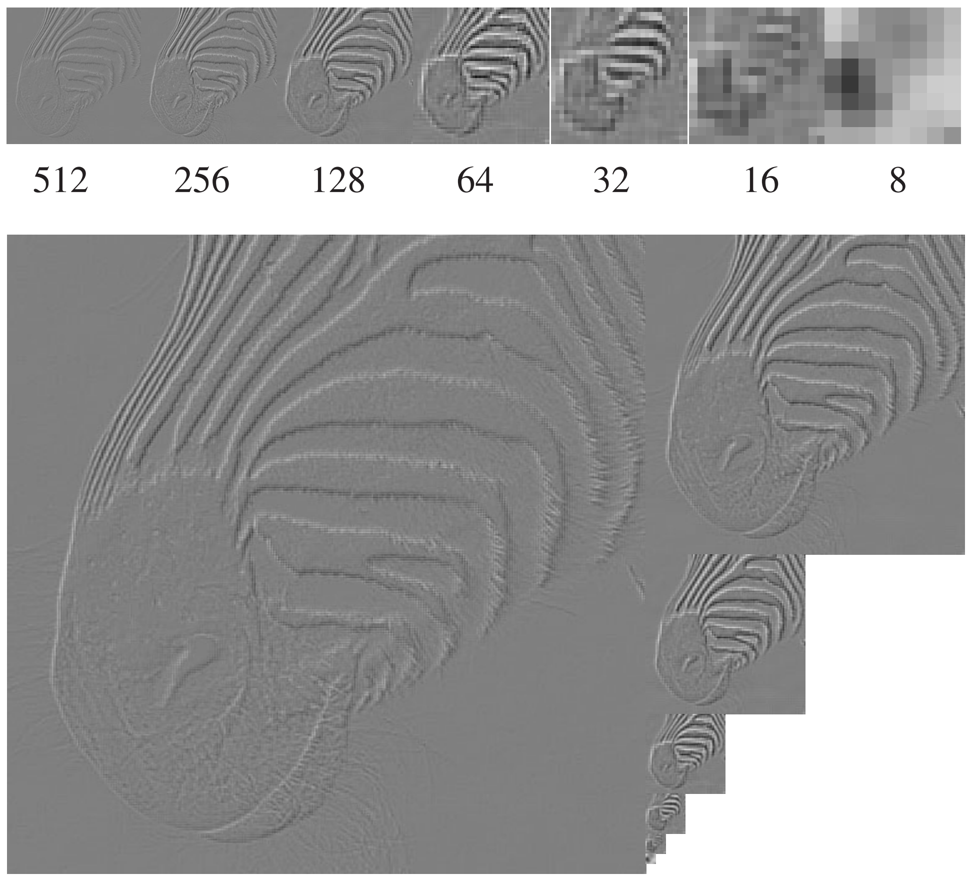


Figure 11.7. A Laplacian pyramid of images, running from 512x512 to 8x8. A zero response is coded with a mid-grey; positive values are lighter and negative values are darker. Notice that the stripes give stronger responses at particular scales, because each layer corresponds (roughly) to the output of 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** (algorithm 2).

```
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
```

Algorithm 11.1: *Building a Laplacian pyramid from an image*

```
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
```

Algorithm 11.2: *Synthesis: obtaining an image from a Laplacian pyramid*

11.2.2 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 11.8); this means that it selects a range of spatial frequencies, but is not selective to orientations (because points at 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 11.8) — 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 11.11 get wider as one moves across the image. If we think in terms of spatial frequency

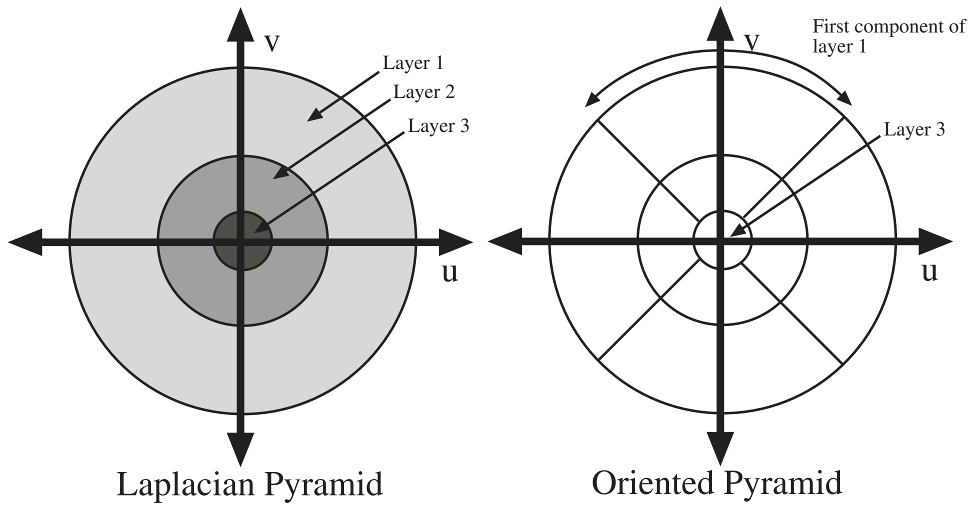


Figure 11.8. Each layer of the Laplacian pyramid consists the elements of a smoothed and resampled image that are not represented by the next smoother layer. Assuming that a Gaussian is a sufficiently good smoothing filter, each layer can be thought of as representing the image components within a range 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).

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

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

and the antisymmetric kernel has the form

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

The filters are illustrated in figures 11.9 and 11.10; (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 analyse 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.

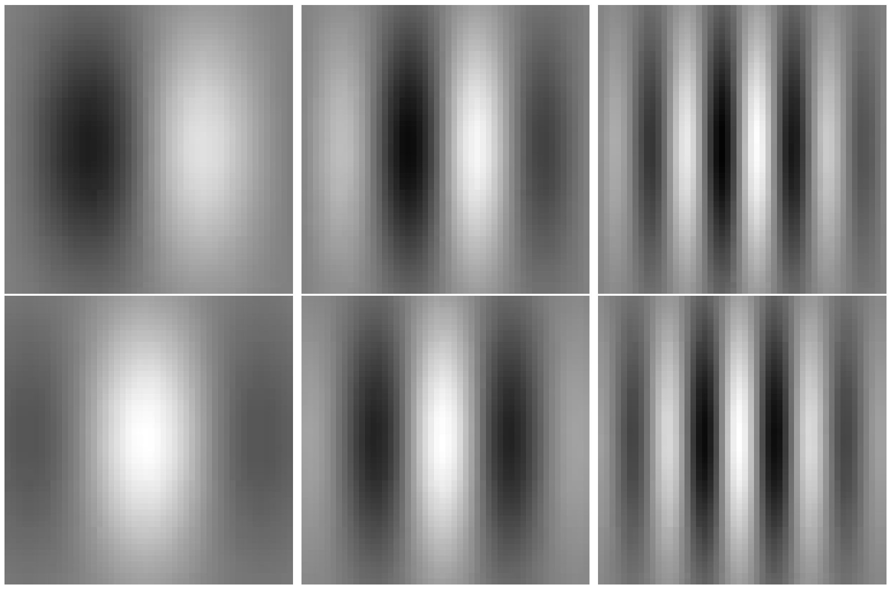


Figure 11.9. Gabor filter kernels are the product of a symmetric Gaussian with an oriented sinusoid; the form of the kernels is given in the text. The images show Gabor filter kernels as images, with mid-grey values representing zero, darker values representing negative numbers and lighter values representing 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 11.10 shows Gabor filters at a finer scale.

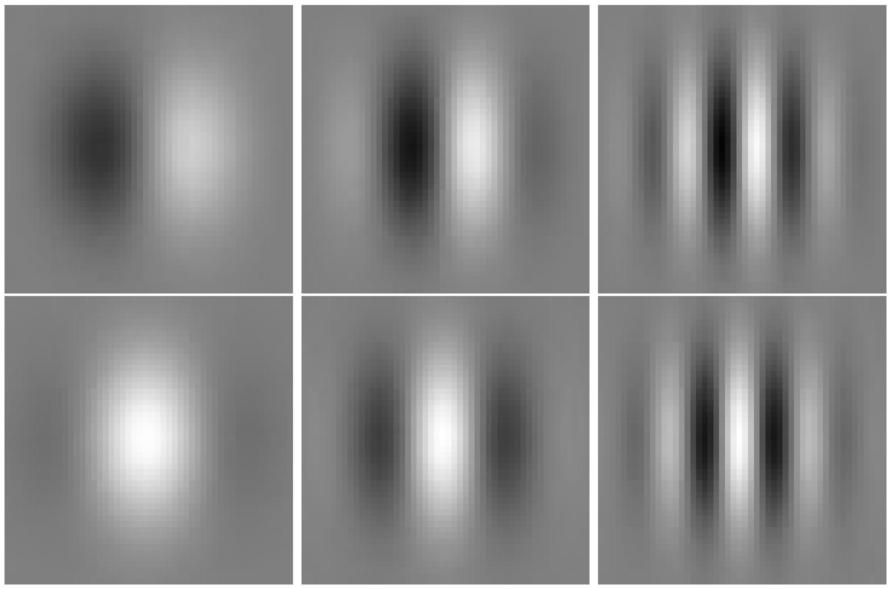


Figure 11.10. The images shows Gabor filter kernels as images, with mid-grey values representing zero, darker values representing negative numbers and lighter values representing positive numbers. The top row shows the antisymmetric component, and the bottom row shows the symmetric component. The scale 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 11.9.

11.2.3 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 11.12).

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 11.8. 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 11.13), 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

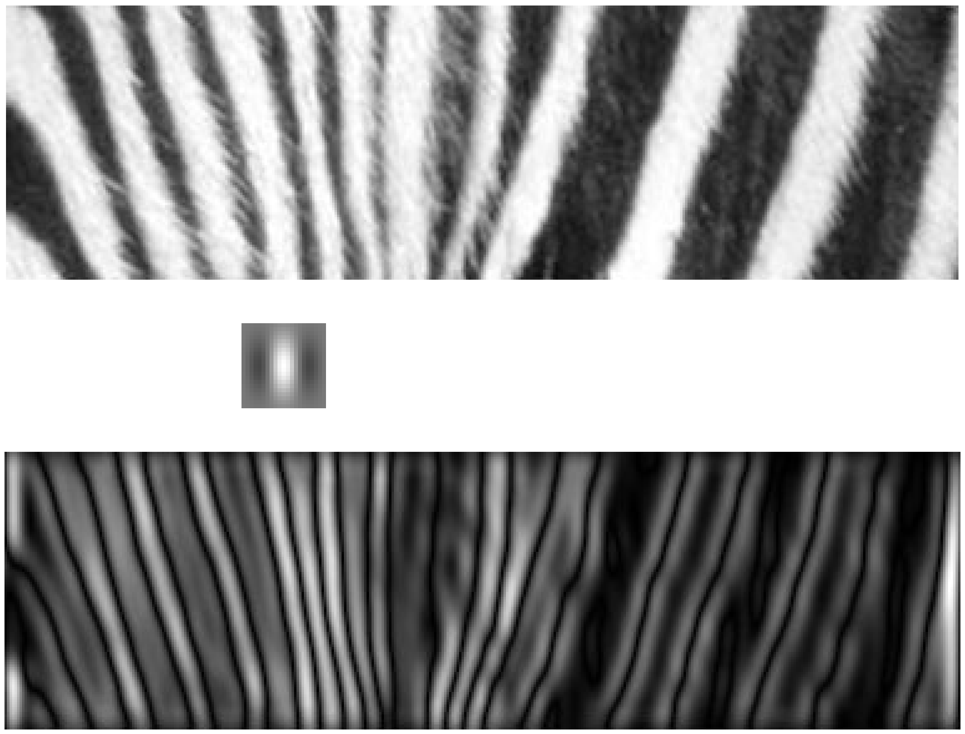


Figure 11.11. The image on the **top** shows a detail from an image of a zebra, chosen because it has a stripes at somewhat different scales and orientations. This has been convolved with the kernel in the center, which is a Gabor filter kernel. The image at the **bottom** shows the absolute value of 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 antisymmetric 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.

components involves filtering the image a second time with the same filter (as figure 11.14 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; Simoncelli and Freeman, 1995].

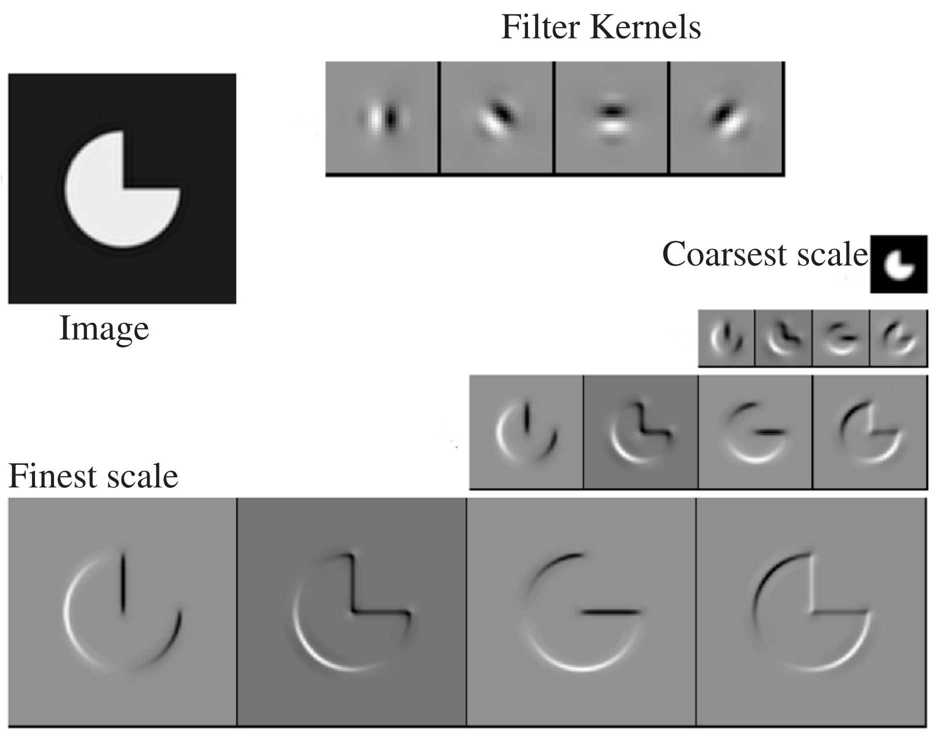


Figure 11.12. An oriented pyramid, formed from the image at the **top left**, with four orientations per layer. This is obtained by firstly decomposing an image into subbands which represent bands of spatial frequency (as with the Laplacian pyramid), and then applying oriented filters (**top right**) 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`. *Figure from "Shiftable MultiScale Transforms", Simoncelli et al., IEEE Transactions on Information Theory, 1992, © 1992, IEEE*

11.3 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 variety 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

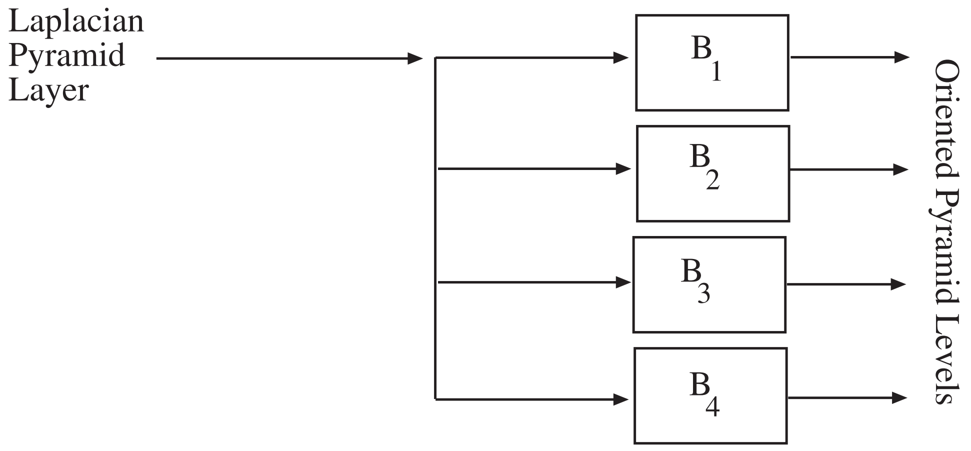


Figure 11.13. The oriented pyramid is obtained by taking layers of the Laplacian pyramid, and then applying oriented filters (represented in this schematic drawing by boxes). Each layer of the Laplacian pyramid represents a range of spatial frequencies; the oriented filters decompose this range of spatial frequencies into a set of orientations.

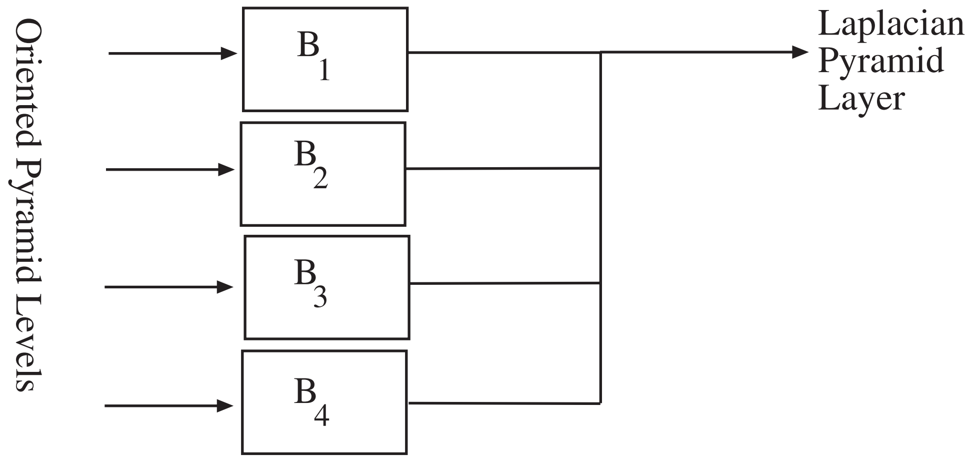


Figure 11.14. In the oriented pyramid, synthesis is possible by refiltering the layers and then adding them, as this schematic indicates. This property is obtained by appropriate choice of filters.

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 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.

11.3.1 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. 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. This assumption 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 now use the example texture to obtain the probability model for the synthesized texture in various ways.

11.3.2 Synthesis by Matching Histograms of Filter Responses

If two homogenous texture samples are drawn from the same probability model, then (if the samples are big enough) histograms of the outputs of various filters applied to the samples will be the same. Heeger and Bergen use this observation to synthesize a texture using the following strategy: take a noise image and adjust it until the histogram of responses of various filters on that noise image looks like the histogram of responses of these filters on the texture sample [Heeger and Bergen, 1995]. Using an arbitrary set of filters is likely to be inefficient; we can avoid this problem by using an oriented pyramid. As we have seen, each orientation of each layer represents the response of an oriented filter at a particular scale, so the whole pyramid represents the response of a large number of different filters. The reason this is efficient is that we have thrown away redundant information in subsampling the images to get the coarser scale layers.

If we represent texture samples as oriented pyramids, we can adjust the pyramid corresponding to the image to be synthesized, and then synthesize the image from the pyramid, using the methods of section 11.2.3. We will adjust each layer separately, and then synthesize an image. There are two steps: firstly, we need to know how to adjust a layer, and secondly, we need to make these local adjustments converge to the correct overall structure.

Histogram Equalization

We think about each layer in an oriented pyramid as an image, and the problem of adjusting a layer becomes the following. We have two images, and we should like to adjust image two so that it has the same histogram as image one. The process is known as **histogram equalization**. Histogram equalization is easiest for images that are continuous functions. In this case, we record for each value of the image the percentage of the image that takes the value less than or equal to this one-this record is known as the **cumulative histogram**. The cumulative histogram is a continuous, monotonically increasing function that maps the range of the image to the unit interval. Because it is continuous and monotonically increasing, the inverse exists. The inverse of the cumulative histogram takes a percentage — say 25 % — and gives the image value v such that the given percentage of the image has value less than or equal to v — i.e. 0.3, if 25% of the image has value less than or equal to 0.3.

The easiest way to describe histogram equalisation is slightly inefficient in space. We create a temporary image, image three. Now choose some value v from image two. The cumulative histogram of image two yields that p percent of the image has value less than v. Now apply the inverse cumulative histogram of image one to p, yielding a new value v' for v. Wherever image two has the value v, insert the value v' in image three. If this is done for every value, image three will have the same histogram as image one. This is because, for any value in image three, the percentage of image three that has that value is the same as the percentage of image one that has that value. In fact, image three isn't necessary, as we can transform image two in place, yielding algorithm 3.

Things are slightly more difficult for discrete images and images that take discrete values. For example, if image one is a binary image in which every pixel but one is black, and image two is a binary image in which half the pixels are white, some but not all of the white pixels in image two will need to be mapped to black — but which ones should we choose? usually the choice is made uniformly and at random.

Adjusting a Pyramid to Obtain an Image

Now assume we have started with a noise image, formed a pyramid, and then adjusted each layer to have the same histogram as each layer of the example pyramid. The resulting pyramid will not, in general, be a pyramid that could be obtained from any image, because we have assumed that the layers are independent and they are not. This means that, if we synthesize an image from the adjusted pyramid and then compute the pyramid from the synthesized image, the resulting pyramid

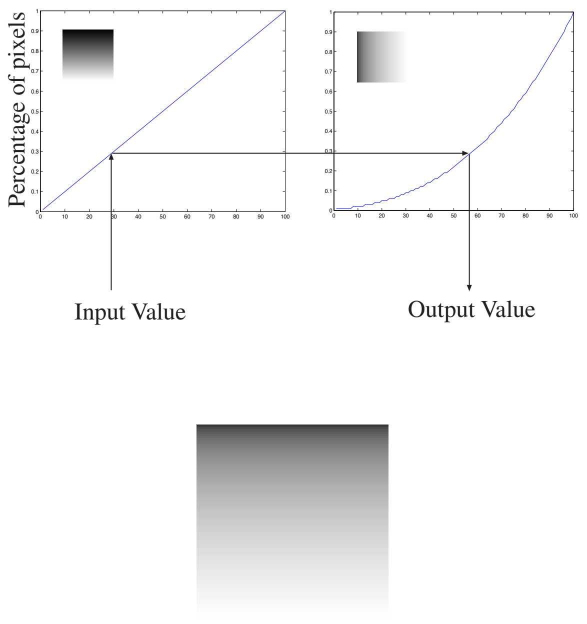


Figure 11.15. Histogram equalization uses cumulative histograms to map the grey levels of one image so that it has the same histogram as another image. The figure at the top shows two cumulative histograms, with the relevant images inset in the graphs. To transform the left image so that it has the same histogram as the right image, we take a value from the left image, read off the percentage from the cumulative histogram of that image, and obtain a new value for that grey level from the inverse cumulative histogram of the right image. The image on the left is a linear ramp (it looks non-linear because the relationship between brightness and lightness is not linear); the image on the right is a cube root ramp. The result — the linear ramp, with grey levels remapped so that it has the same histogram as the cube root ramp — is shown on the bottom row.

will not, in general, match our original adjusted pyramid. In particular, we are not guaranteed that each layer in the new pyramid has the histogram we want it to. If the layer histograms are not satisfactory, we readjust the layers, resynthesize the image, and iterate. While convergence is not guaranteed, in practice the process appears to converge.

The overall technique looks like algorithm 4. This algorithm yields quite good results on a substantial variety of textures, as figure 11.16 indicates. It is inclined to fail when there are conditional relations in the texture that are important — for example, in figure 11.17, the method has been unable to capture the fact that the spots on the coral lie in stripes. This problem results from the assumption that the histogram at each spatial frequency and orientation is independent of that at every other.

```
Form the cumulative histogram c1(v) for image 1
Form the cumulative histogram c2(v) for image 2
Form ic1(p), the inverse of c1(v)

for every value v in image 2, going from smallest to largest
  Obtain a new value vnew=ic1(c2(v))
  Replace the value v in image 2 with vnew
end
```

Algorithm 11.3: *Histogram Equalization*

```
Make a working image from noise
Match the working image histogram to the example image histogram
Make a pyramid pe from the example image

until convergence
  Make a pyramid pw from the working image
  for each layer in the two pyramids
    Match the histogram of pw's layer to that of pe's layer
  end
  Synthesize the working image from the pyramid pw
end
```

Algorithm 11.4: *Iterative texture synthesis using histogram equalisation applied to an oriented pyramid*

11.3.3 Synthesis by Sampling Conditional Densities of Filter Responses

A very successful algorithm due to DeBonet retains the idea of synthesizing a texture by coming up with an image pyramid that looks like the pyramid associated with an example texture [de Bonet, 1997]. However, this approach does not assume that the layers are independent, as the previous algorithm did.

For each location in a given layer of the pyramid, there are a set of locations in coarser scale layers associated with it by the sampling process (as in figure 11.18). The set of values of in these locations is called the **parent structure** of the location.

We can use this parent structure for synthesis. Firstly, let us make the coarsest scale in the new pyramid the same as the coarsest scale — say the m'th level —

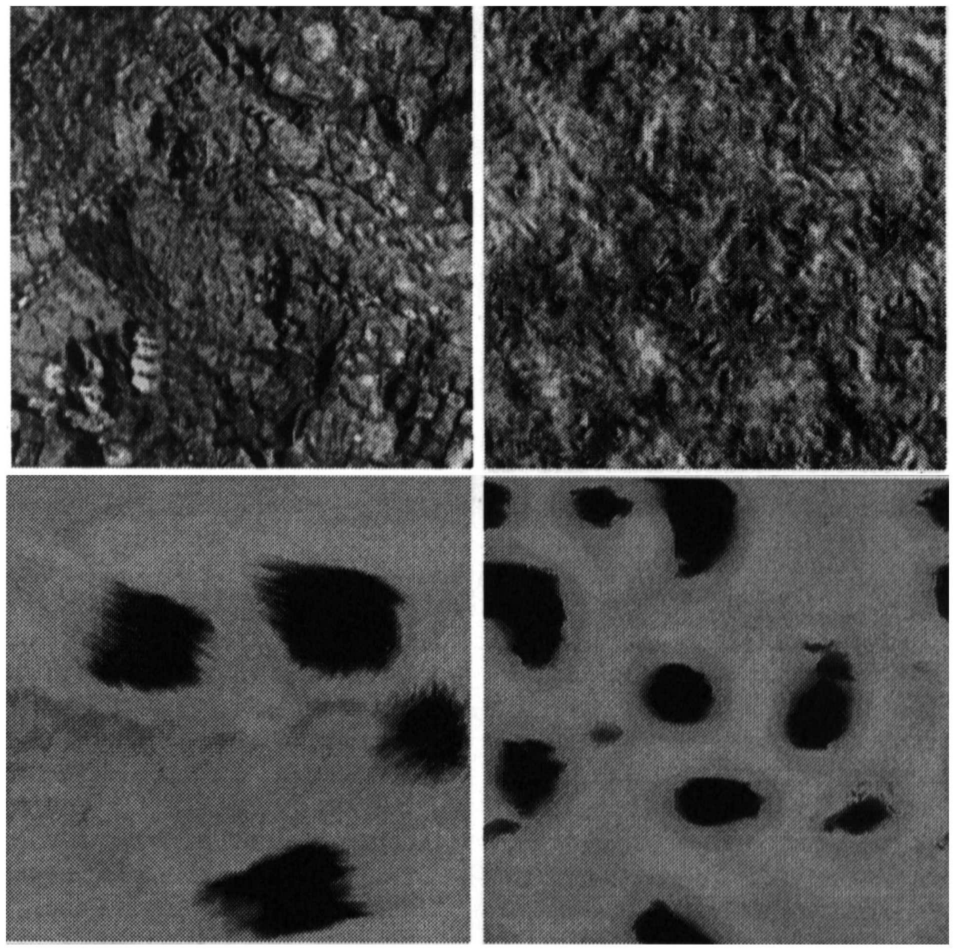


Figure 11.16. Examples of texture synthesis by histogram equalisation. On the left, the example textures and on the right, the synthesized textures. For the top example, the method is unequivocally successful. For the bottom example, the method has captured the spottiness of the texture but has rather more (and smaller) spots than one might expect. *figure from Heeger and Bergen, Pyramid-based Texture Analysis and Synthesis, p. figure 3, in the fervent hope, etc.*

in the example pyramid. Now choose a location to be synthesized in the $m-1$'th level of the pyramid. We know the parent structure of this location, so we can go to the example pyramid and collect all values in the corresponding level that have a similar parent structure. This collection forms a probability model for the values for our location, conditioned on the parent structure that we observed. If we choose an element from this collection uniformly and at random, the values at the m'th

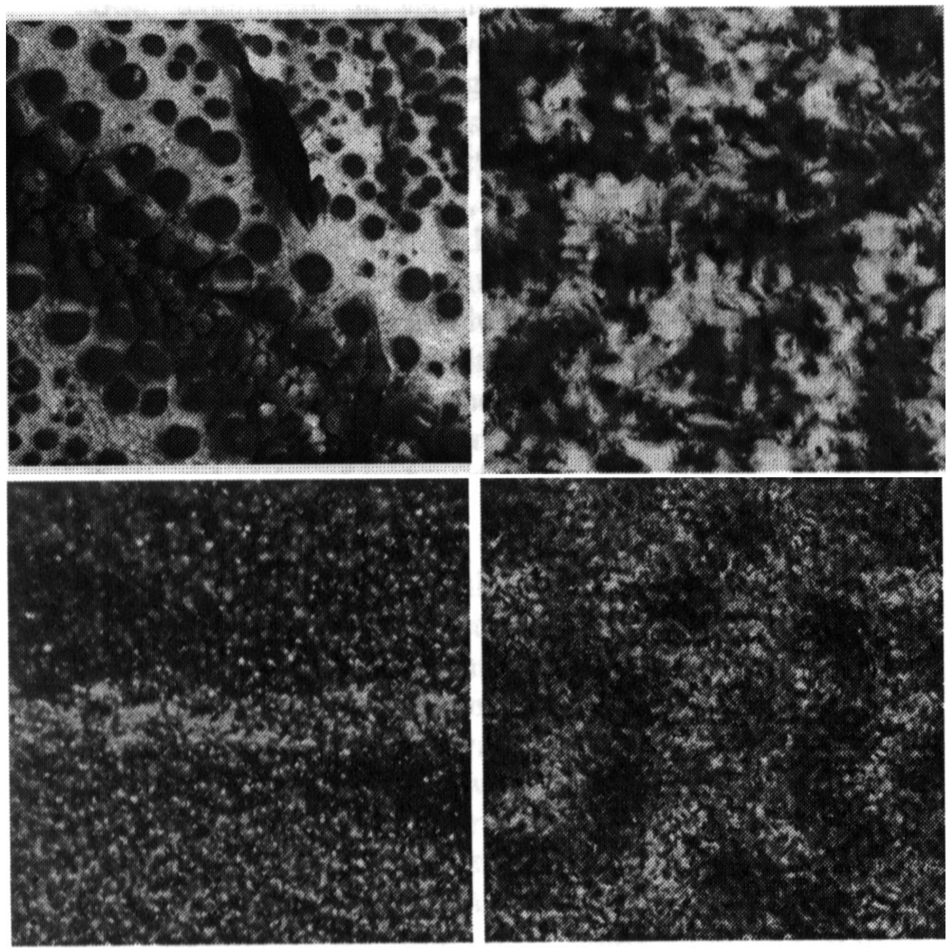


Figure 11.17. Examples of texture synthesis by histogram equalisation failing. The left column shows example textures, and the right hand column shows synthesized textures. The main phenomenon that causes failure is that, for most natural textures, the histogram of filter responses at different scales and orientations is not independent. In the case of the coral (top left), this independence assumption suppresses the fact that the small spots on the coral lie in a straight line. *figure from Heeger and Bergen, Pyramid-based Texture Analysis and Synthesis, p. figure 8, in the fervent hope, etc., figure from Heeger and Bergen, Pyramid-based Texture Analysis and Synthesis, p. figure 7, in the fervent hope, etc.*

level and at the $m-1$'th level of the pyramid being synthesized have the same *joint* histogram as the corresponding layers in the example pyramid.

This is easiest to see if we think of histograms as a representation of a proba-

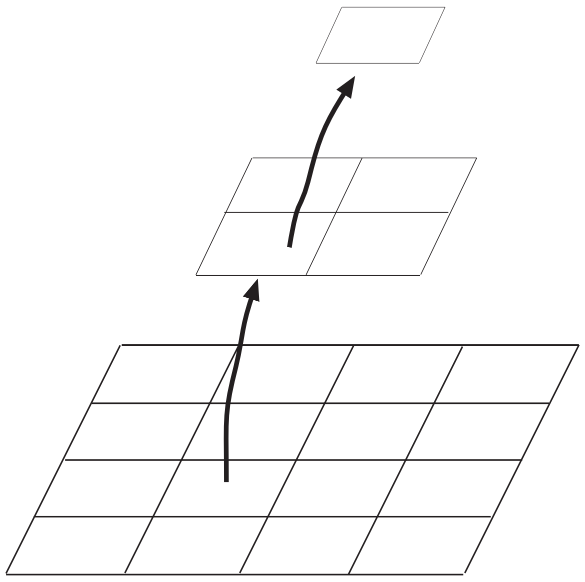


Figure 11.18. The values of pixels at coarse scales in a pyramid are a function of the values in the finer scale layers. We associate a parent structure with each pixel, which consists of the values of pixels at coarse scales which are used to predict our pixel's value in the Laplacian pyramid, as indicated in this schematic drawing. This parent structure contains information about the structure of the image around our pixel for a variety of differently sized neighbourhoods.

bility distribution. The joint histogram is a representation of the joint probability distribution for values at the two scales. This joint distribution is the product of a marginal distribution on the values at the m'th level with the conditional distribution on values at the $m-1$'th level, *conditioned* on the value at the m'th level.

The m'th level layers must have the same histograms (that is, the same marginal distributions). The sampling procedure for the $m-1$'th layer means that a histogram of the pixels in the $m-1$'th layer whose parents have some fixed value will be the same for the pyramid being synthesized as for the example pyramid. This histogram — which is sometimes called a **conditional histogram** — is a representation of the conditional distribution on values at the $m-1$'th level, *conditioned* on the value at the m'th level.

Nothing special is required to synthesize a third (or any other) layer. For any location in the third layer, the parent structure involves values from the coarsest and the next to coarsest scale. To obtain a value for a location, we collect every element from the corresponding layer in the example pyramid with the same parent structure, and choose from a uniformly and at random from this collection. The fourth, fifth and other layers follow from exactly the same approach. Furthermore,

```
Make a pyramid pe from the example image
Make an empty pyramid pw, corresponding to the image to
  be synthesized

Set the coarsest scale layer of pw to be the same as the
  coarsest scale level of pe; if pw is bigger than pe, then
  replicate copies of pe to fill it

for each other layer l of pe, going from coarsest to finest
  for each element e of the layer

    Obtain all elements with
      the same parent structure

    Choose one of this collection uniformly at random

    Insert the value of this element into e

  end
end

Synthesize the texture image from the pyramid pw
```

Algorithm 11.5: *Texture Synthesis using Conditional Histograms*

the joint histogram of all these layers in the synthesized pyramid will be the same as that for the example pyramid, using the same argument as above.

There are three important details to address before we have a usable algorithm.

- Firstly, what does it mean for parent structures to be the same? In practice, it is sufficient to regard the parent structures as vectors and require that they are close together — an appropriate distance threshold should be set by experiment.

- Secondly, what components of the parent structure should be preserved in matching? If we represent a pixel at a layer by all pixels in previous layers, we have a comprehensive record of its ancestors, but there may be no comparable pixels. Again, this criterion is usually satisfied by experiment.

- Finally, how do we obtain all pixels with the same parent structure as a given location? one strategy is to search all locations in the example image for every

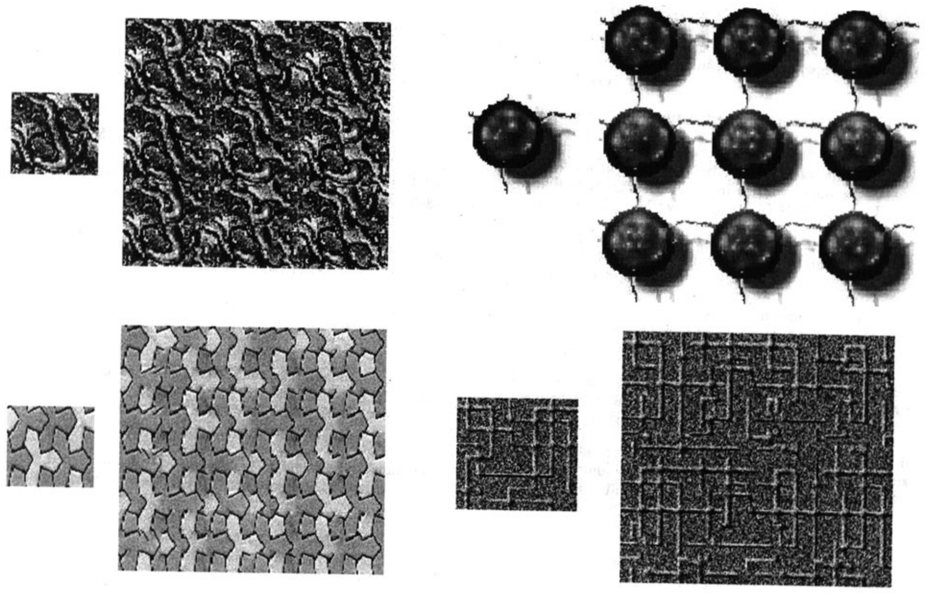


Figure 11.19. Four examples of textures synthesized using De Bonet's algorithm (algorithm 5). In each case, the example texture is the small block on the left, and the synthesized texture is the larger image block on the right. Note that the method has captured apparent periodic structure in the textures; in the case of the blob with wires (top right), it has succeeded in joining up wires. This is because the method can capture larger scale structure in a texture in greater detail, by not assuming that responses at each level of the pyramid are independent. *figure from De Bonet, Multiresolution Sampling Procedure for Analysis and Synthesis of Image Textures, p figure 10, in the fervent hope, etc.*

pixel value we wish to synthesize, but this is crude and expensive. We explore alternate strategies in the exercises.

11.3.4 Synthesis by Sampling Local Models

As Efros points out, it isn't essential to use an oriented pyramid to build a probability model [Efros and Leung, 1999]. Instead, the example image itself supplies 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

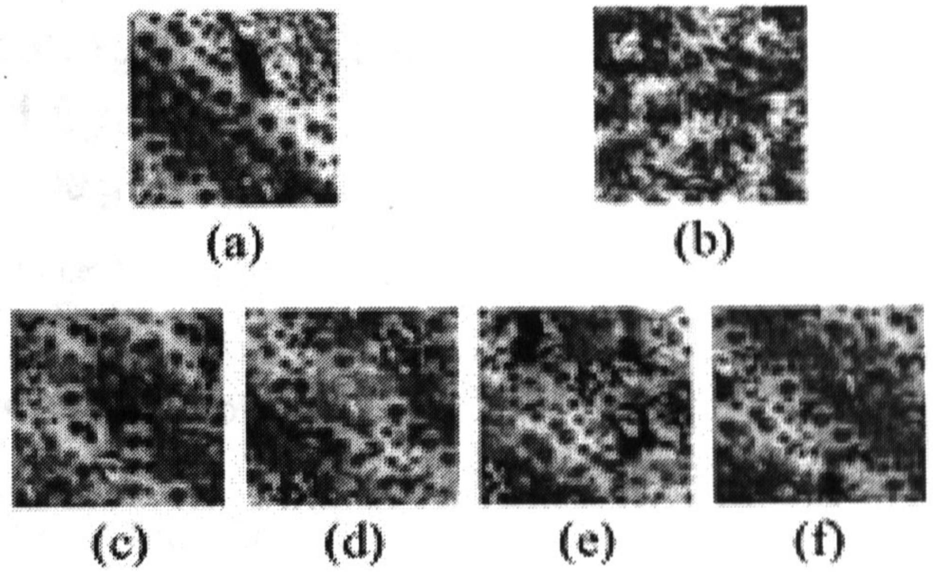


Figure 11.20. Figure 11.16 showed texture synthesis by histogram equalisation failing on the coral texture example shown on the top left here, because the independence assumption suppresses the fact that the small spots on the coral lie in a straight line. The texture synthesized by histogram equalization is shown on the top right. The bottom row shows textures synthesized using algorithm 5, which doesn't require an independence assumption. These textures appear to have the same structure as the example. *figure from De Bonet, Multiresolution Sampling Procedure for Analysis and Synthesis of Image Textures, p figure 14, in the fervent hope, etc.*

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 the range over which pixels can affect one another's values directly (see figure 11.22). 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 6.

```
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
```

Algorithm 11.6: *Non-parametric Texture Synthesis*

11.4 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 11.23). Remarkably, quite general texture models appear to supply enough information to infer shape. This is most easily seen for planes (section 11.4.1); while the details remain opaque

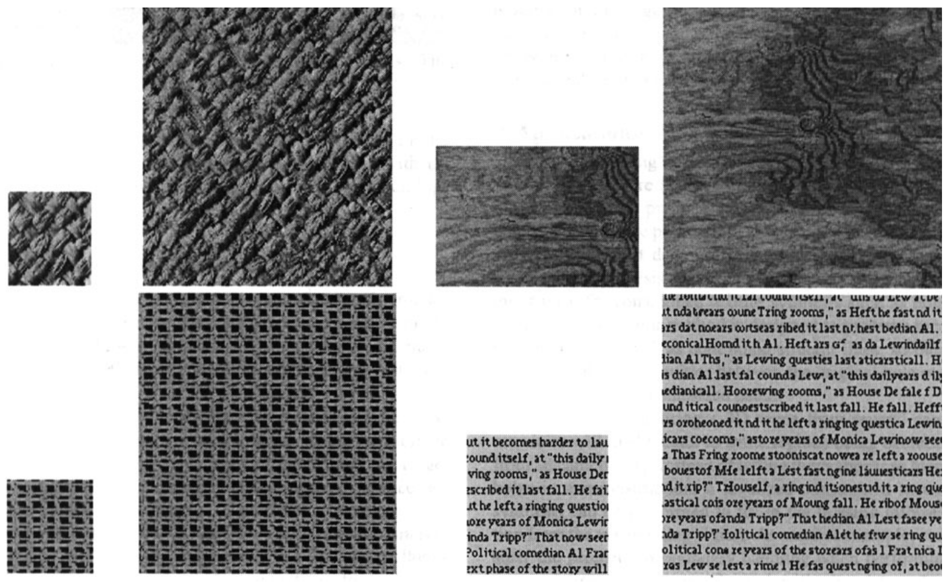


Figure 11.21. Efros' texture synthesis algorithm (algorithm 6) matches neighbourhoods of the image being synthesized to the example image, and then chooses at random amongst the possible values reported by matching neighbourhoods. This means that the algorithm can reproduce complex spatial structures, 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 Efros, Texture Synthesis by Non-parametric sampling, p. figure 3, in the fervent hope, etc.*

in the case of curved surfaces, the general issues remain the same (we scratch the surface in section 11.4.2).

11.4.1 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.

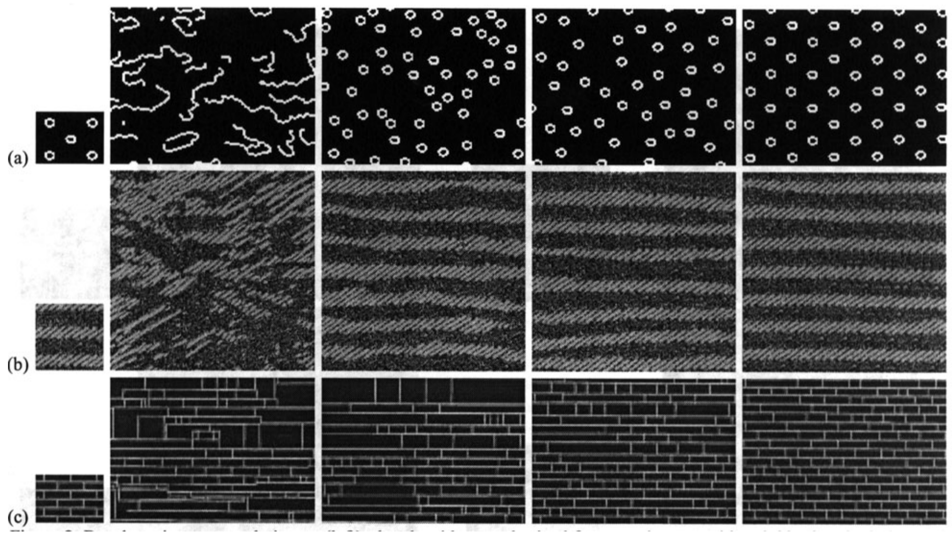


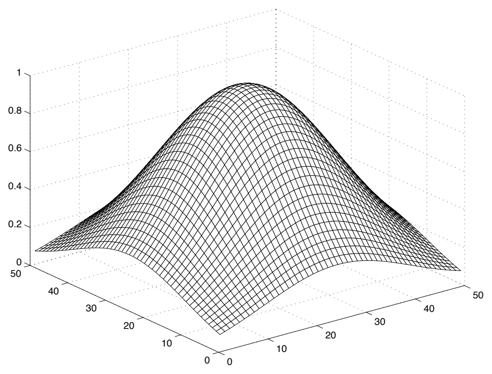
Figure 11.22. The size of the image neighbourhood to be matched makes a significant difference in algorithm 6. In the figure, the textures at the right are synthesized from the small blocks on the left, using neighbourhoods that are increasingly large as one moves to the right. If very small neighbourhoods 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 Efros, Texture Synthesis by Non-parametric sampling, p. figure 2, in the fervent hope, etc.*

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 11.24). 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



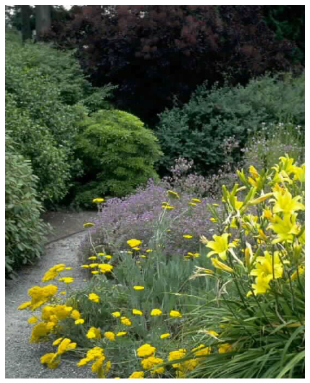


Figure 11.23. Humans obtain information about the shape of surfaces in space from the appearance of the texture on the surface. The figure on the left shows one common use for this effect — away from the contour regions, our only source of information about the surface depicted is the distortion of the texture on the surface. On the right, the texture of the stones gives a clear sense of the orientation of the (roughly) plane surface leading up to the waterhole.

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 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 11.24).

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

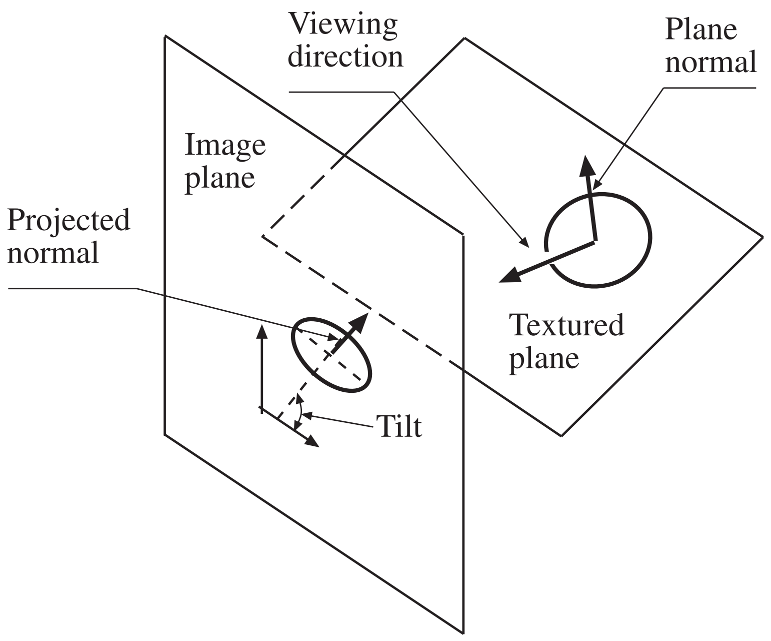


Figure 11.24. The orientation of a plane with respect to the camera plane can be given by the slant — which is the angle between the normal of the textured plane and the viewing direction — and the tilt — which is the angle the projected normal makes with the camera coordinate system. The figure illustrates the tilt, and shows a circle projecting to an ellipse.

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 11.3.1). This is because the viewing transformation takes one homogeneous texture into another homogeneous texture. However, if we make some other assumptions about the structure of the texture, it becomes possible. One possible assumption is that the texture is a **homogenous marked Poisson point process**. This is a special case of the Poisson point process described in section 10.4.1; in particular, the texture is obtained by (1) marking points on the plane with a homogenous Poisson point process and then (2) dropping a texture element (a "mark") at each point, with the choice of element and orientation being random with some fixed distribution.

Assume that we can identify each texture element. Now recall that the core property of a homogenous Poisson point process is that the expected number of points in a set is proportional to the area of that set. Consider a set of rectangles *in the image* oriented by the slant-tilt coordinate system: a rectangle that is long in the slant direction and short in the tilt direction will contain more texture elements than a rectangle that is long in the tilt direction and short in the slant direction. This is because the slant direction is foreshortened — so that image length in this direction is shorter than length on the plane in this direction — but the tilt direction is not. However, the foreshortening does not affect the count of texture elements. These observations mean that we can obtain the slant and tilt direction of a plane textured according to our model by searching over plane orientations to find one that makes the back-projected texture most uniform in space.

11.4.2 Shape from Texture for Curved Surfaces

For many textures lying on a curved surface, we can recover information about the differential geometry of that surface. The reasoning is as follows:

- We assume that the texture is a homogeneous marked Poisson point process. This means that, if we know the configuration of one of the tangent planes on the surface, then we know what the texture elements look like frontally.
- Now we assume that we know the configuration of one of the tangent planes.
- Now we can reconstruct other tangent planes — possibly every other tangent plane — from this information, because we know the rule by which the texture foreshortens.

Of course, we don't know the configuration of any of the tangent planes, so we need to reason about relative configurations. The texture distorts from place to place in the image, because it undergoes different projections into the image: we keep track of those distortions, and use them to reason about the shape of the surface (figures 11.25). Shape from texture for curved surfaces tends to require some technical geometry, however, and we will pursue it in no further detail.

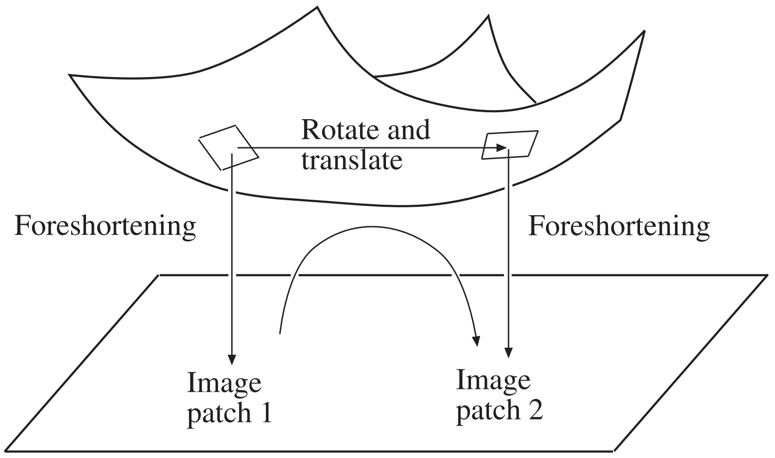


Figure 11.25. If the texture on a surface is homogenous, then the texture at each point on the surface "looks like" the texture at other points. This means that the deformation of the texture in the image is a cue to surface geometry. In particular, the texture around one point in the image is related to the texture around another point by: mapping from the image to the surface, transforming on the surface, and then mapping back to the image. By keeping track of these transformations, we can reconstruct surfaces up to some ambiguity.

11.5 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; 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; Ma and Manjunath, 1996; Manjunath and Ma, 1996b].

11.5.1 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 [Perona, 1991; Perona, 1995; Freeman and Adelson, 1991; Greenspan *et al.*, 1994; Perona, 1992; Freeman and Adelson, 1990; Simoncelli and Farid, 1995; Hel-Or and Teo, 1996; Simoncelli and Freeman, 1995; Simoncelli and Farid, 1995].

11.5.2 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 *et al* [Zhu *et al.*, 1998], which uses sophisticated entropy criteria to firstly choose filters by which to represent a texture and secondly construct probability models for that texture.

11.5.3 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 [Garding, 1992], has been successfully demonstrated for a variety of surfaces by Malik and Rosenholtz [Malik and Rosenholtz, 1997; Rosenholtz and Malik, 1997]; a reformulation in terms of wavelets is due to Clerc [Clerc and Mallat, 1999]. The method has 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.

Assignments

Exercises

1. The texture synthesis algorithm of section 11.3.3 needs to obtain parent structures in the example image that match the parent structure of a pixel to be synthesized. These could be obtained by blank search. An alternative is to use a hashing process. It is essential that every parent structure that could match a given structure is obtained by this hashing process. One strategy is to compute a hash key from the parent structure, and then look at nearby keys as well, to ensure that no matches are missed.

   - Describe how this strategy could work.
   - What savings could be obtained by using it?

2. 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?

3. 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.

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

- Show that the area of each set, *backprojected onto the textured plane*, is a function of the orientation of the plane.
- Use this function to suggest a method for obtaining the plane's orientation.

Programming Assignments

- **Texture synthesis - a:** Implement the texture synthesis algorithms of section 11.3.2 and of section 11.3.3. Use the steerable filter implementation available at `http://www.cis.upenn.edu/ eero/steerpyr.html` to construct steerable pyramid representations. Use your implementation to find examples where the independence assumption fails. Explain what is going on in these examples.
- **Texture synthesis - b:** Extend the algorithms of section 11.3.2 and of section 11.3.3 to use pyramids obtained using an analysis based on more orientations; you will need to ensure that you can do synthesis for the set of filters you choose. Does this make any difference in practice to (a) the quality of the texture synthesis or (b) the speed of the synthesis algorithm?
- **Texture synthesis - c:** Implement the non-parametric texture synthesis algorithm of section 11.3.4. Use your implementation to study:
    1. the effect of window size on the synthesized texture;
    2. the effect of window shape on the synthesized texture;
    3. the effect of the matching criterion on the synthesized texture (i.e. using weighted sum of squares instead of sum of squares, etc.).

Part IV

EARLY VISION: MULTIPLE IMAGES