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.

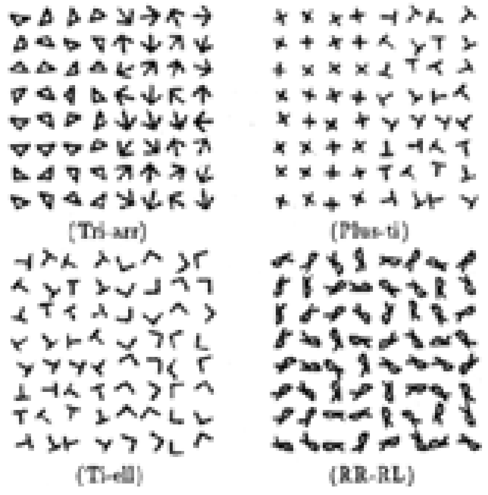


Figure 11.1. A set of texture examples, used in experiments with human subjects to tell how easily various types of textures can be discriminated. Note that these textures are made of quite stylised subelements, repeated in a meaningful way. *figure from the Malik and Perona, A Computational Model of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

There are three standard problems to do with texture:

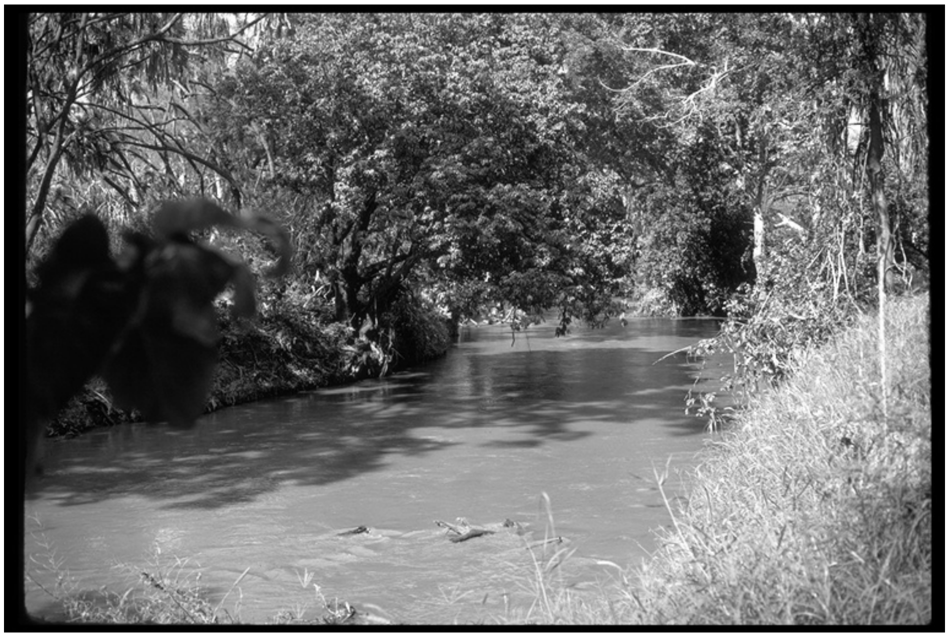


Figure 11.2. A typical textured image. For materials such as brush, grass, foliage and water, our perception of what the material is is quite intimately related to the texture. These textures are also made of quite stylised subelements, arranged in a pattern. *figure from the Malik and Perona, A Computational Model of Texture Segmentation, p.331, in the fervent hope, etc. figure from the Calphotos collection, number. 0057, in the fervent hope, etc.*

- **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); chapter **??** shows 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 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 and section 11.5, 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.

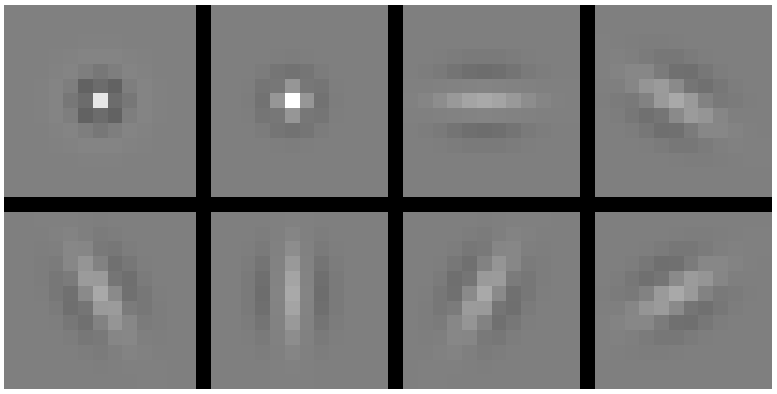


Figure 11.3. A set of eight filters used for expanding images into a series of responses. These filters are shown at a fixed scale, with zero represented by a mid-grey level, lighter values being positive and darker values being negative. They represent two distinct spots, and six bars; the set of filters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

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 $(0, -1)$.

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

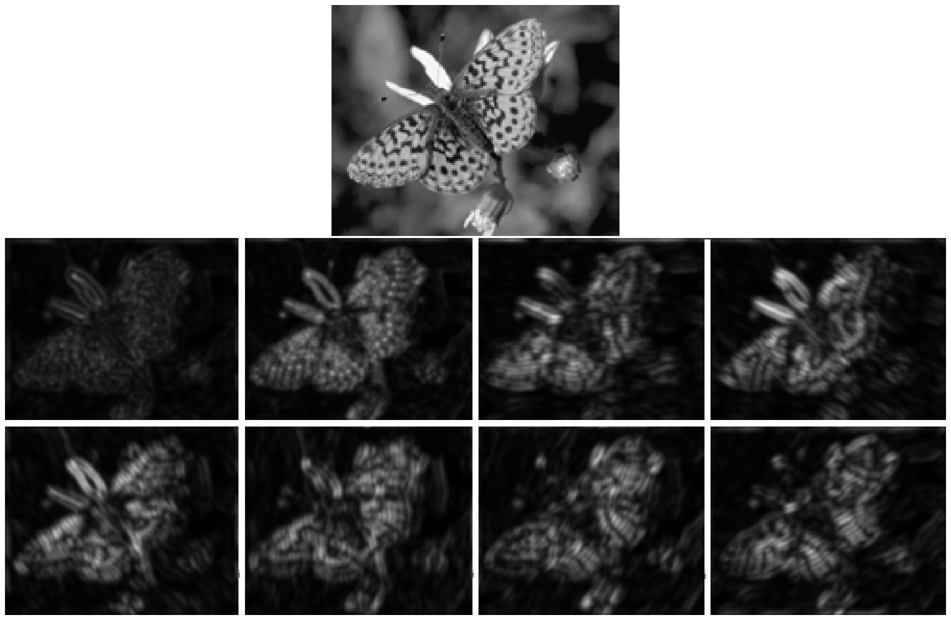


Figure 11.4. At the top, an image of a butterfly at a fine scale, and below, the result of applying each of the filters of figure 11.3 to that image. The results are shown as absolute values of the output, lighter pixels representing stronger responses, and the images are laid out corresponding to the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

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.

Spots and Bars by Gabor Filters

Another way to build spot and bar filters is to use **Gabor filters**. 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

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

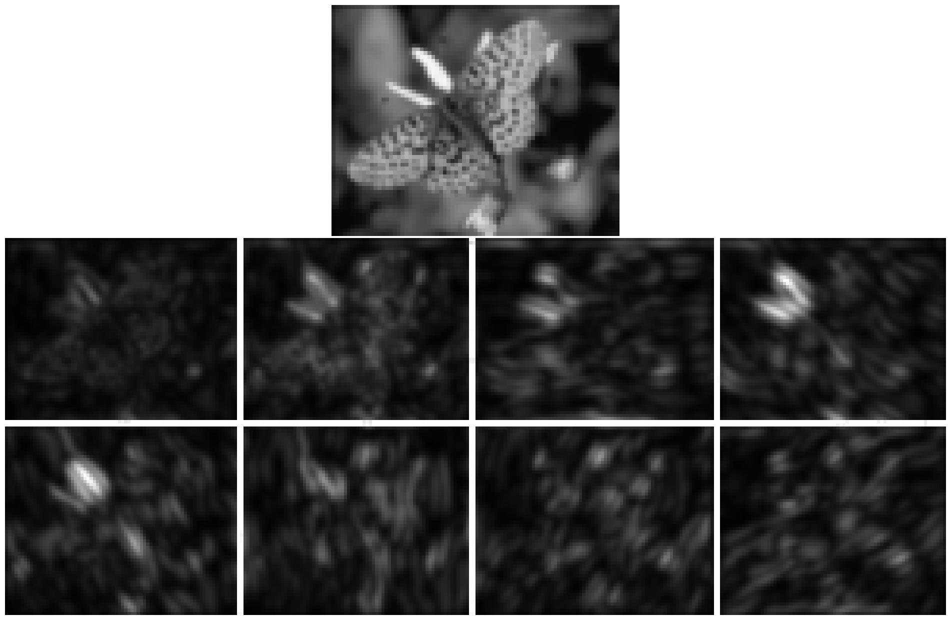


Figure 11.5. The input image of a butterfly and responses of the filters of figure 11.3 at a coarser scale than that of figure 11.4. Notice that the oriented bars respond to the bars on the wings, the antennae, and the edges of the wings; the fact that one bar has responded does not mean that another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

and the antisymmetric kernel has the form

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

The filters are illustrated in figures 11.6 and 11.7; (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.

Gabor filter kernels look rather a lot like smoothed derivative kernels, for different orders of derivative. For example, if the spatial frequency of the Gabor filter is low compared to the scale and the phase is zero, we get a kernel that looks a lot like a derivative of Gaussian filter (top left of figure 11.6); if the phase is $\pi/2$, then the kernel looks a lot like a second derivative of Gaussian filter (bottom left of figure 11.6). Another way to think of Gabor filter kernels is as assemblies of bars — as the spatial frequency goes up compared to the scale, the filter looks for patches of parallel stripes rather than individual bars.

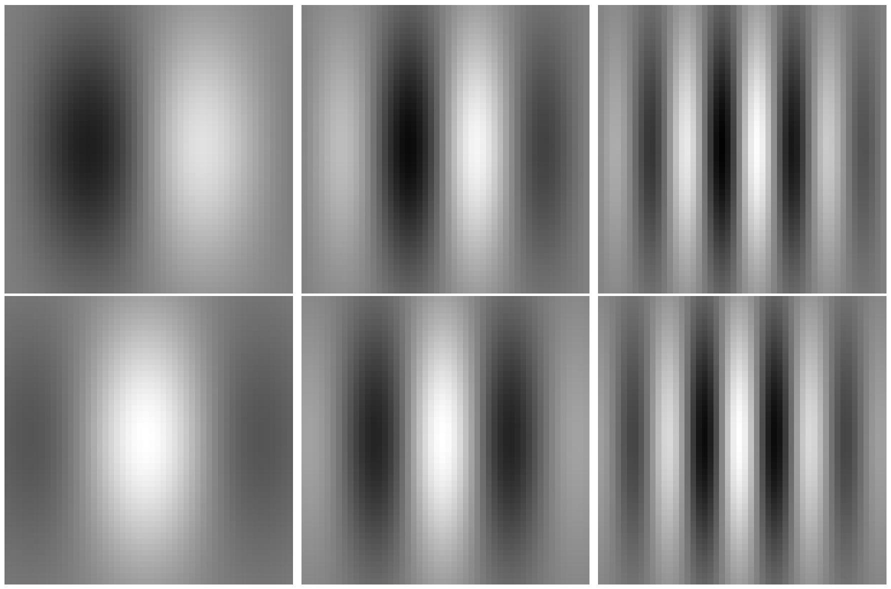


Figure 11.6. Gabor filter kernels are the product of a symmetric Gaussian with an oriented sinusoid; the form of the kernels is given in the text. The images show Gabor filter kernels as images, with mid-grey values representing zero, darker values representing negative numbers and lighter values representing 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. Notice how these filters look rather like derivative of Gaussian filters — as the spatial frequency goes up, so does the derivative in the derivative of Gaussian model. It can be helpful to think of these filters as seeking groups of bars. Figure 11.7 shows Gabor filters at a finer scale.

How many Filters and at what Orientation?

It is not known just how many filters are required 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

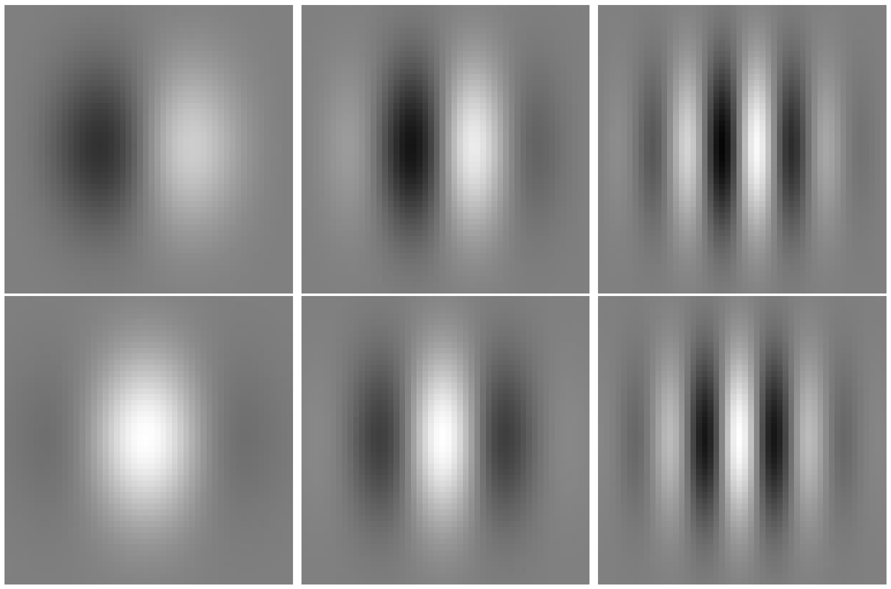


Figure 11.7. The images shows Gabor filter kernels as images, with mid-grey values representing zero, darker values representing negative numbers and lighter values representing positive numbers. The top row shows the antisymmetric component, and the bottom row shows the symmetric component. The scale 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 11.6.

the image); but we must also convolve the image with all these filters, which can be expensive. Section **??** illustrates a variety of the tricks that are used to reduce the computational expense.

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 **??**) 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; in section 11.2.2, 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 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 (that is, the components of the image that lie within a particular range of spatial frequencies. 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

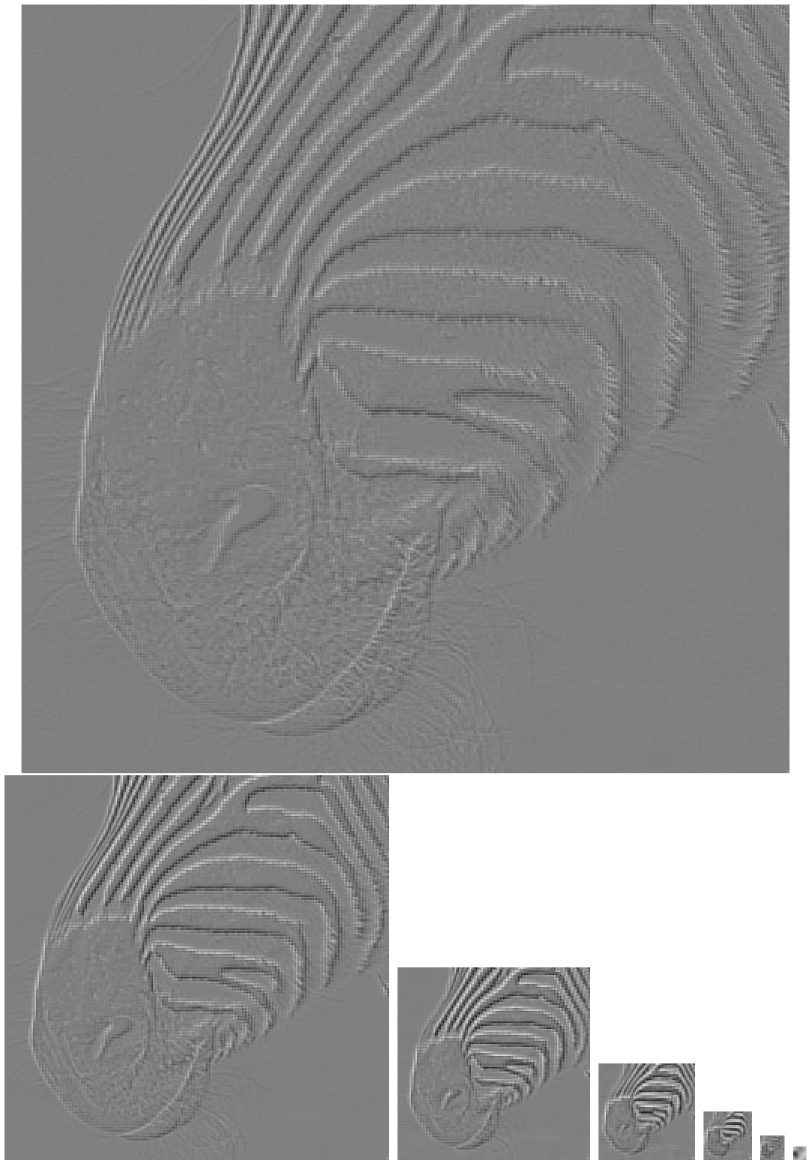


Figure 11.8. A Laplacian pyramid of images, running from 512x512 to 8x8. A zero response is coded with a mid-grey; positive values are lighter and negative values are darker. Notice that the stripes give stronger responses at particular scales, because each layer corresponds (roughly) to the output of a band-pass filter.

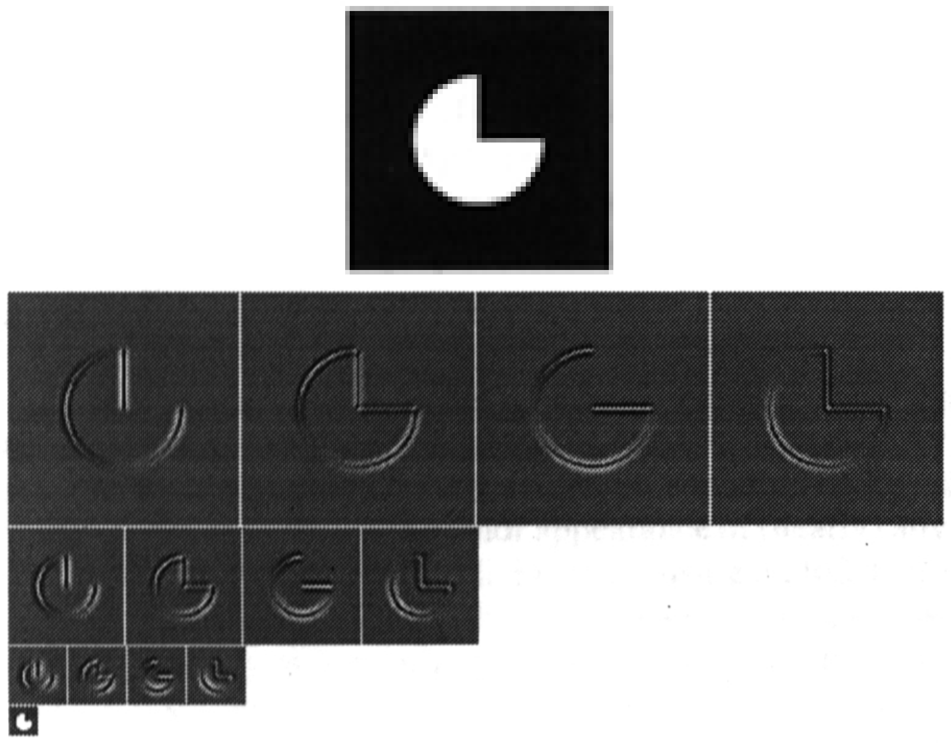


Figure 11.9. An oriented pyramid, formed from the image at the top, with four orientations per layer. This is obtained by firstly decomposing an image into subbands which represent bands of spatial frequency (as with the Laplacian pyramid), and then applying oriented filters to these subbands to decompose 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 Heeger and Bergen, Pyramid-based Texture Analysis and Synthesis, p. figure 1, in the fervent hope, etc.*

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 11.8).

Because different levels of the pyramid represent different spatial frequencies, the Laplacian pyramid can be used as a reasonably effective image compression scheme. Laplacian pyramids are also used for image blending (figure **??**).

```
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*

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. 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).

11.2.2 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 images each of which represents a energy at a distinct orientation. Each subimage represents the response of an oriented filter at a particular scale — this is a detailed analysis of the image.

If we have a strategy for reconstructing each layer from its components, then synthesis is easy: we reconstruct the layers, and then reconstruct the image from them. 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

```
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*

time with the same filter (as figure 11.10 suggests). The design process involves imposing our two design criteria. Discussing the detailed design of these filters — which are extremely useful — would take us somewhat out of the way. The technique is useful; fortunately, an efficient implementation of these pyramids is available at `http://www.cis.upenn.edu/ eero/steerpyr.html`. Those who wish to understand the design processes can look at the same site for papers on the topic.

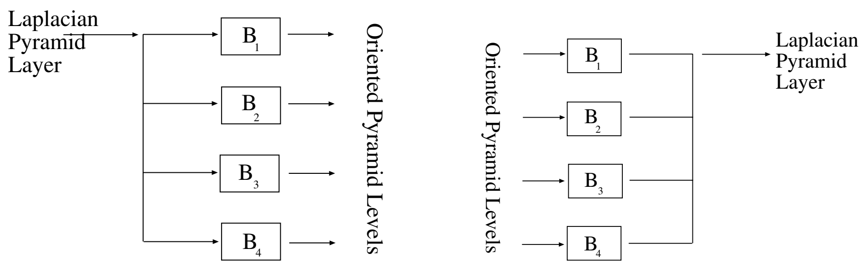


Figure 11.10. The oriented pyramid is obtained by taking layers of the Laplacian pyramid, and then applying oriented filters (represented in this schematic drawing by boxes). By appropriate choice of filters, synthesis is possible by refiltering the layers and then adding them, as the schematic on the right indicates.

11.3 Application: Synthesizing Textures for Rendering

Objects rendered using computer graphics systems look more realistic if real textures are rendered on their faces. 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 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.

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[1].

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. We will adjust each layer separately, and then synthesize an image. The details of the process for adjusting the layers is given in section 4; for the moment, we will assume that this process works, and discuss what we do with it.

Once we have obtained an image from the adjusted pyramid, we form a pyramid from that image (the two pyramids will not, in general, be the same, because we've assumed, incorrectly, that the layers are independent). 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.

```
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 11.3: *Iterative texture synthesis using histogram equalisation applied to an oriented pyramid*

The overall technique looks like algorithm 3. This algorithm yields quite good results on a substantial variety of textures, as figure 11.11 indicates. It is inclined

[1]The reason this is efficient is that we have thrown away redundant information in subsampling the images to get the coarser scale layers.

to fail when there are conditional relations in the texture that are important — for example, in figure 11.12, 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.

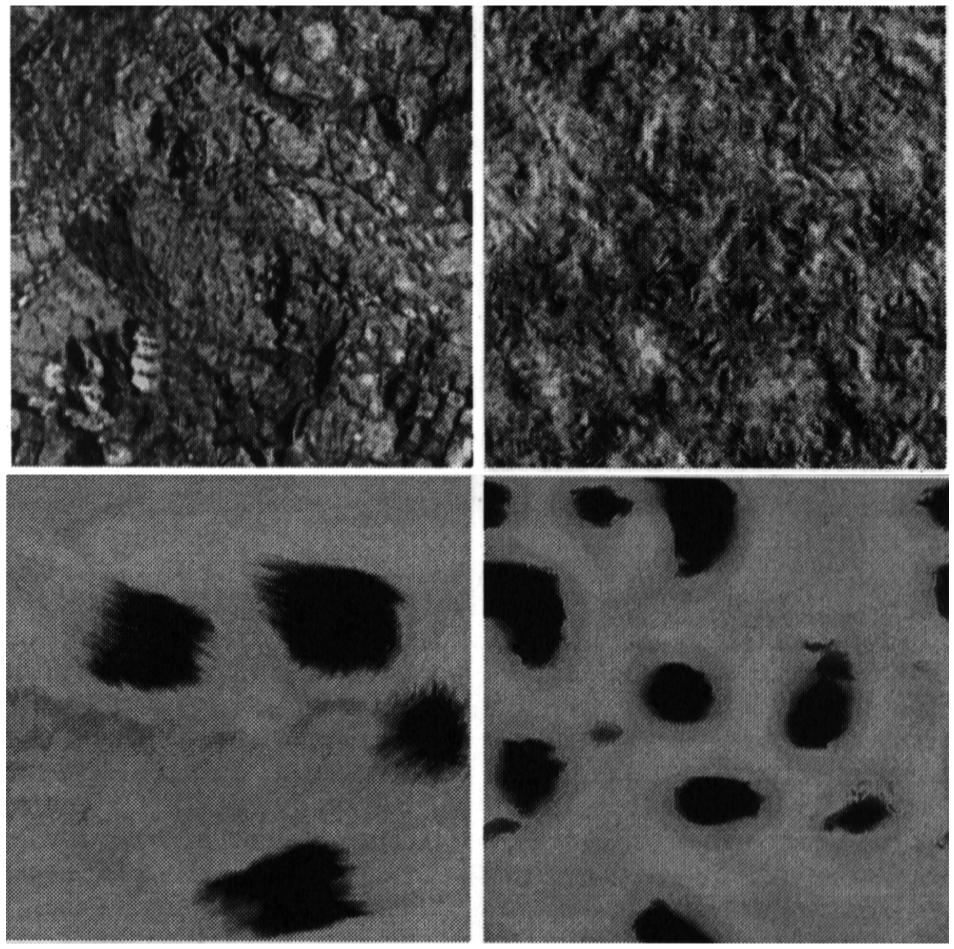


Figure 11.11. Examples of texture synthesis by histogram equalisation. On the left, the example textures and on the right, the synthesized textures. For the top example, the method is unequivocally successful. For the bottom example, the method has captured the spottiness of the texture but has rather 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.*

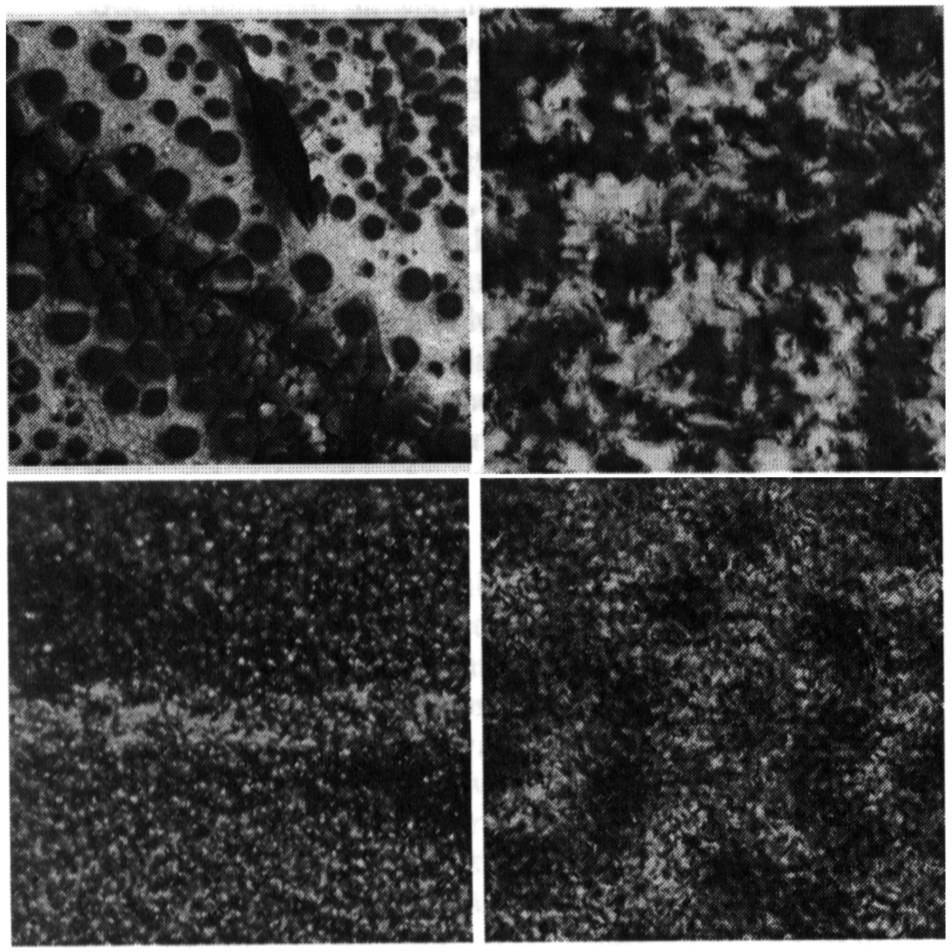


Figure 11.12. Examples of texture synthesis by histogram equalisation failing. The left column shows example textures, and the right hand column shows synthesized textures. The main phenomenon that causes failure is that, for most natural textures, the histogram of filter responses at different scales 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.*

Histogram Equalization

We have two images — which might be layers from the oriented pyramid — 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.

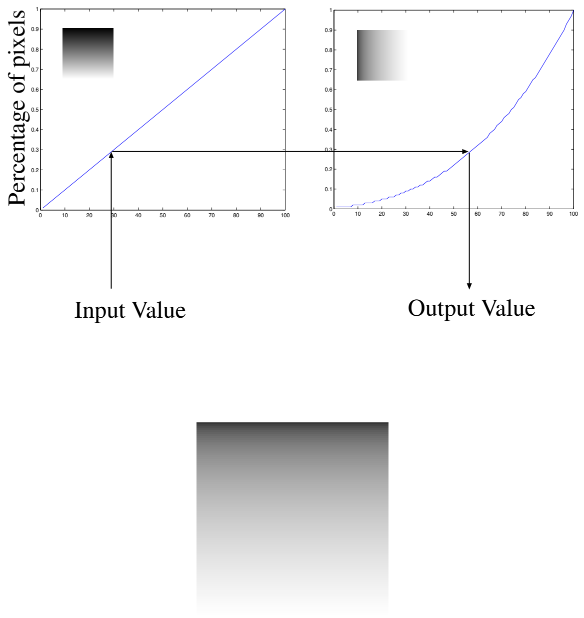


Figure 11.13. Histogram equalization uses cumulative histograms to map the grey levels of one image so that it has the same histogram as another image. The figure at the top shows two cumulative histograms, with the relevant images inset in the graphs. To transform the left image so that it has the same 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.

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 4.

```
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 11.4: *Histogram Equalization*

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.

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. 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 11.14). 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 — 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

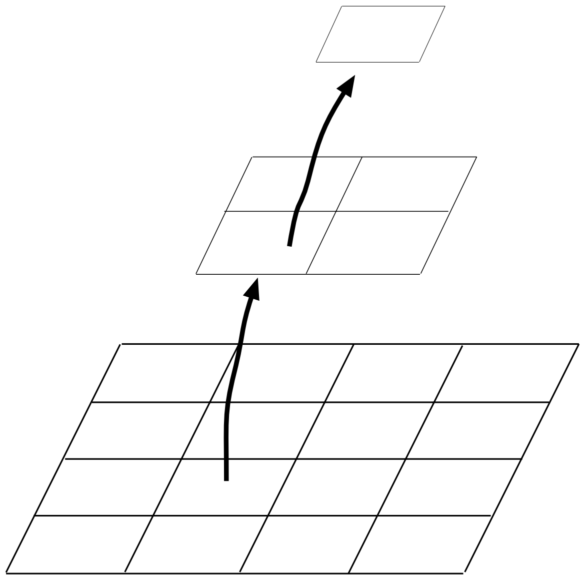


Figure 11.14. The values of pixels at coarse scales in a pyramid are a function of the values in the finer scale layers. We associate a parent structure with each pixel, which consists of the values of pixels at coarse scales which are used to predict our pixel's value in the Laplacian pyramid, as indicated 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.

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 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 probability distribution. The joint histogram is a representation of the joint probability distribution for values at the two scales. From section **??**, 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

```
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*

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, 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 two 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, 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

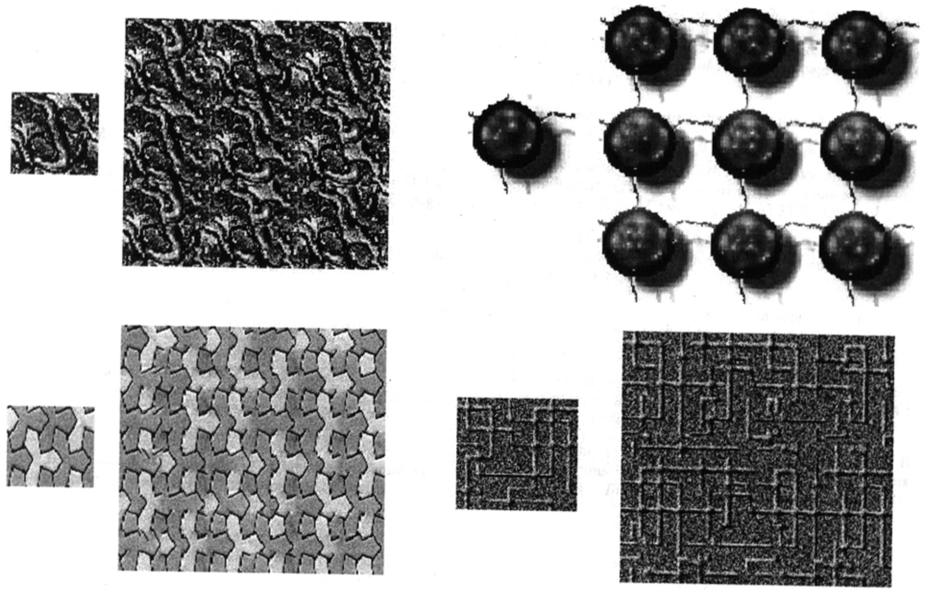


Figure 11.15. Four examples of textures synthesized using De Bonet's algorithm (algorithm 5). In each case, the example texture is the small block on the left, and the synthesized texture is the larger image block on the right. Note that the method has captured apparent periodic structure in the textures; 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.*

for every pixel value we wish to synthesize, but this is crude and expensive. We explore alternate strategies in exercise **??**.

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. 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 value that is consistent with the

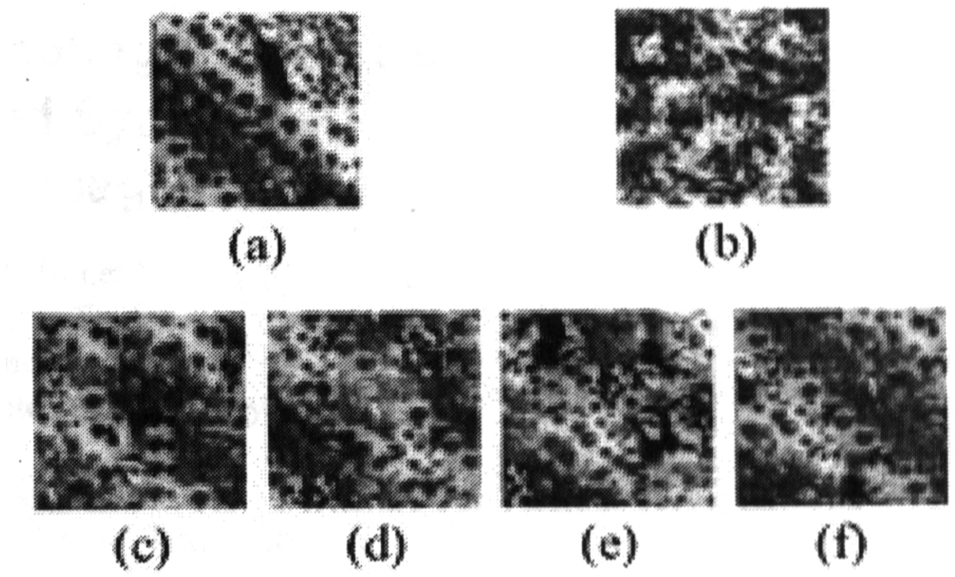


Figure 11.16. Figure 11.11 showed texture synthesis by histogram equalisation failing on the coral texture example shown on the top left here, because the independence assumption suppresses the fact that the small spots on the coral lie in a straight line. The texture synthesized by histogram equalization 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.*

example image. Section **??** describes the details of the matching process.

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.

Matching Image Neighbourhoods

Efros uses a square neighborhood, centered at the pixel of interest. The size of the neighborhood is a parameter that significantly affects the appearance of the synthesized image (see figure 11.18). 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

```
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*

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.

The set of possible values for the pixel of interest comes from any neighborhood of the example image whose sum of squared differences with the neighborhood of interest is smaller than some threshold. Other choices of neighbourhood, and of matching criterion, might work well; little is known about what is best.

11.4 Shape from Texture: Planes and Isotropy

A patch of texture of viewed frontally looks very different from a same patch viewed at a glancing angle, because foreshortening appears to make the texture elements smaller, and move them closer together. This means that, if a surface is covered with the same texture, we should be able to tell elements that are frontal from those that are viewed at a glancing angle. By doing so, we can recover the shape of the surface (figure 11.19).

To construct useful algorithms, we need to be crisp about what it means for a texture to be the same. In the first case, let us assume that we are looking at textured planes. There are two useful notions of similarity for this case. We discussed homogenous textures above (section 11.3.1); 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

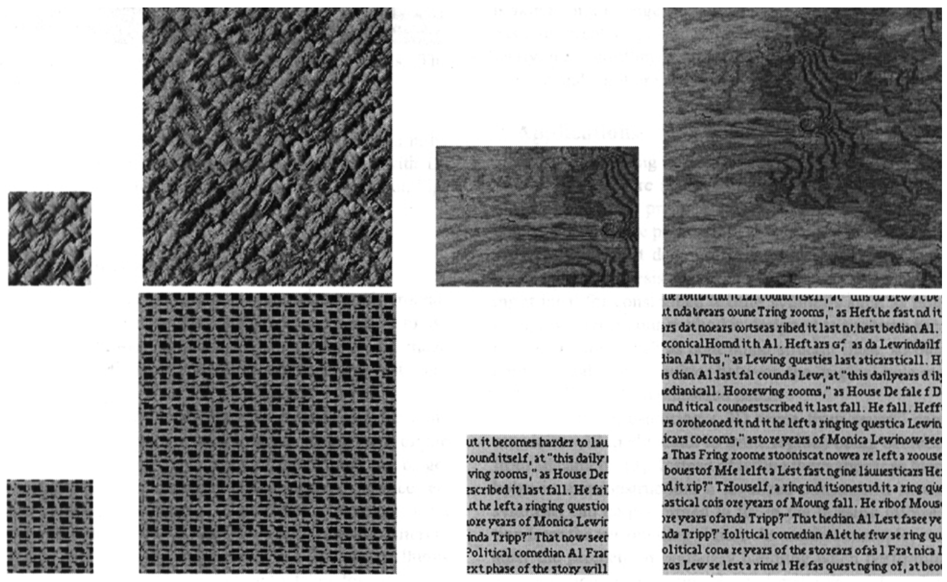


Figure 11.17. Efros' texture synthesis algorithm (algorithm 6) matches neighbourhoods of the image being synthesized to the example image, and then chooses at random amongst the possible values reported by matching neighbourhoods. This means that the algorithm can reproduce complex spatial structures, 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.*

plane.

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. Derivations for other cases appear in [].

11.4.1 Recovering the Orientation of a Plane from an Isotropic Texture

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

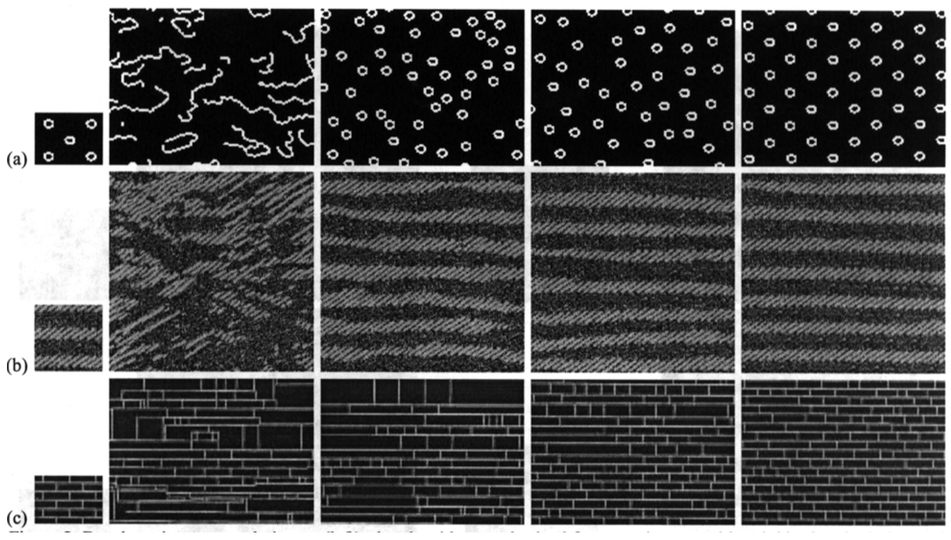
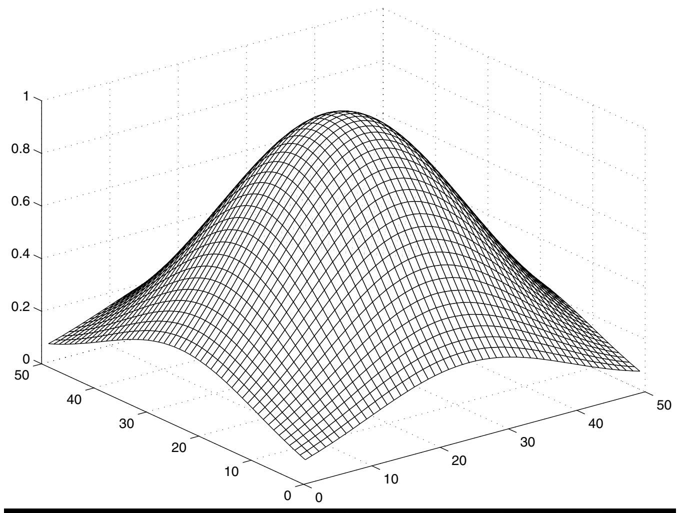


Figure 11.18. The size of the image neighbourhood to be matched makes a significant difference in algorithm 6. In the figure, the textures at the right are synthesized from the small blocks on the left, using neighbourhoods that are increasingly large as one moves to the right. If very small neighbourhoods 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.*

camera coordinate system — sometimes called the **tilt** (figure 11.20). In an image of a plane, there is a **tilt direction** — the direction in the plane parallel to the projected normal.

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 exercise **??** and figure 11.20).

The process of contraction interferes with the isotropy of the texture, because elements that point along the contracted direction get shorter. Furthermore, elements that have a component along the contracted direction have that component shrunk. This yields a strategy for determining the orientation of the plane: find a



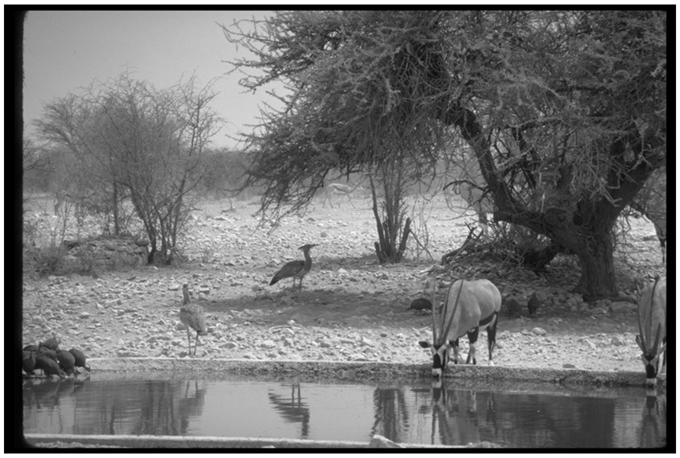


Figure 11.19. Humans obtain information about the shape of surfaces in space from the appearance of the texture on the surface. The figure on the left shows one common use for this effect — away from the contour regions, our only source of information about the surface depicted is the distortion of the 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. *figure from the Calphotos collection, number. 0127, in the fervent hope, etc.*

viewing transformation that turns the image texture into an isotropic texture, and recover the slant and tilt from that 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

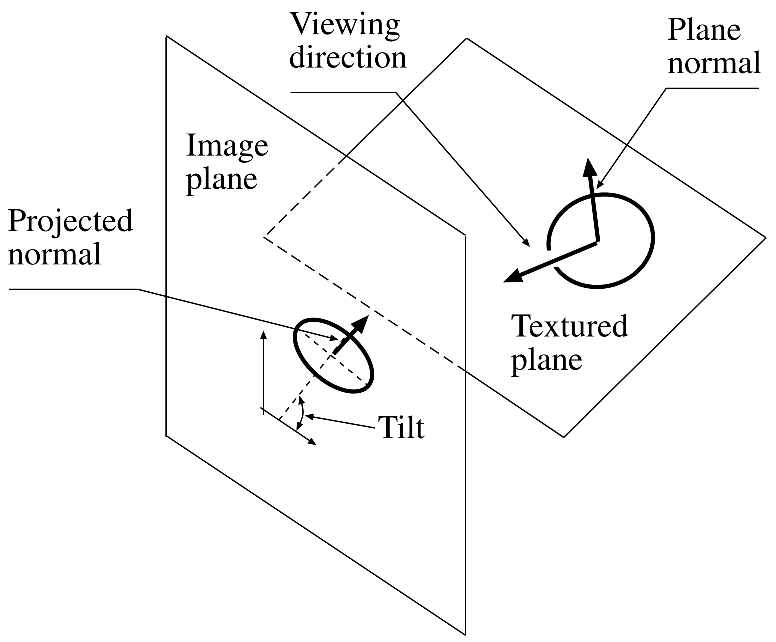


Figure 11.20. The orientation of a plane with respect to the camera plane can be given by the slant — which is the angle between the normal of the textured plane and the viewing direction — and the tilt — which is the angle the projected normal makes with the camera coordinate system. The figure illustrates the tilt, and shows a circle projecting to an ellipse.

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. Notice also that it isn't possible to recover the configuration of a plane from an orthographic image if one assumes the plane is homogenous — an affine transformation of a homogenous texture is homogenous.

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. Curved surfaces have a richer geometric structure — basically, the texture at different points is distorted in different ways — and we can recover that structure with more realistic assumptions.

11.5 Shape from Texture: Curved Surfaces

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 the texture lies on a curved surface, we can recover information about the differential geometry of that surface.

The reasoning is as follows:

- We assume that texture is homogeneous. This means that, if we know the configuration of one of the tangent planes on the surface, then we know what the texture looks 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 are going to keep track of those distortions, and use them to reason about the shape of the surface (figure 11.21). Shape from texture for curved surfaces tends to require some quite substantial technical geometry; we will do only a simple example, which involves recovering a cylinder from a single view.

11.5.1 Shape from Texture for Cylinders

All cylinders can be obtained by extruding plane curves (which need not be circles — we are interested in cylinders with arbitrary cross-sections); this means that, in some coordinate system, we can parametrise a cylinder as

$$\boldsymbol{r}(u,v) = (x(u), y(u), v)^T = \boldsymbol{X}(u) + v\boldsymbol{V}$$

where $\boldsymbol{X}(u) = (x(u), y(u), 0)^T$ (this is called the **generating curve**) and $\boldsymbol{V} = (0,0,1)$. Notice that every plane section perpendicular to the axis looks like the generating curve. This means that there is a natural coordinate system for each tangent plane on the surface — we use

$$(\boldsymbol{e}_1, \boldsymbol{e}_2) = \left(\frac{\frac{d\boldsymbol{X}}{du}}{\left|\frac{d\boldsymbol{X}}{du}\right|}, \boldsymbol{T}\right)$$

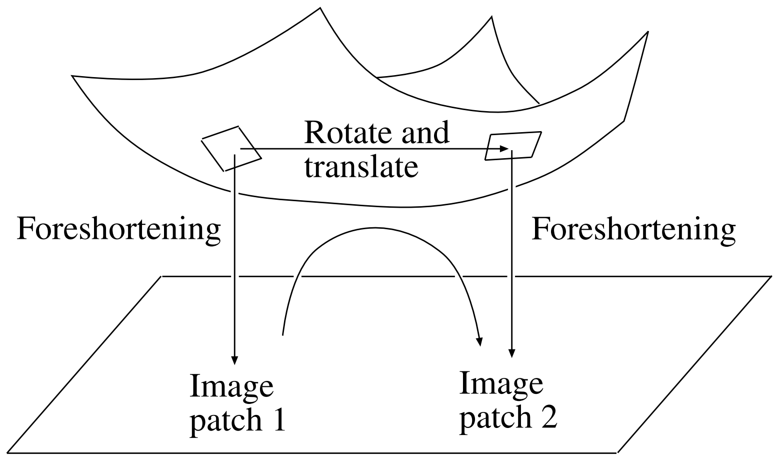


Figure 11.21. If the texture on a surface is homogenous, then the texture at each point on the surface "looks like" the texture at other points. This means that the deformation of the texture in the image is a cue to surface geometry. In particular, the texture around one point in the image is related 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.

as a coordinate system. Now we assume that the texture we are dealing with is homogeneous. We interpret this as meaning the probability of encountering a texture element is independent of u and of v (this definition requires much more care for more complicated geometries).

Now consider a patch of texture around (u_1, v_1) in the surface. We assume that this patch is small enough that we can represent the texture as a patch on the tangent plane at this point, so we can measure the position of each texture element in the $(\boldsymbol{e}_1, \boldsymbol{e}_2)$ coordinate system at this point. We can compare this patch of texture with one around (u_2, v_2) (again, small enough that we can represent it as a patch on the tangent plane, in the $(\boldsymbol{e}_1, \boldsymbol{e}_2)$ coordinate system on that plane). We now ask what transformation makes the textures look "most like" one another — and this must be the identity. We avoid discussing what "most like" means until later (section 11.5.2).

The easiest way to see this is to consider extreme cases. For example, assume that we have a texture process that paints the surface with elliptical blobs, whose major axis points along $\boldsymbol{e}_2$ (i.e. the axis of the cylinder) and whose minor axis points along $\boldsymbol{e}_1$. You should satisfy yourself that this is, in fact, a homogenous texture (hint: homogenous is not the same as isotropic). Furthermore, it is quite obvious that we can determine the transformation between textures at two points — just align the long axes of the ellipses — and that this will be the identity *in the* $(\boldsymbol{e}_1, \boldsymbol{e}_2)$ *coordinate system.* The $(\boldsymbol{e}_1, \boldsymbol{e}_2)$ is an example of a **texture coordinate**

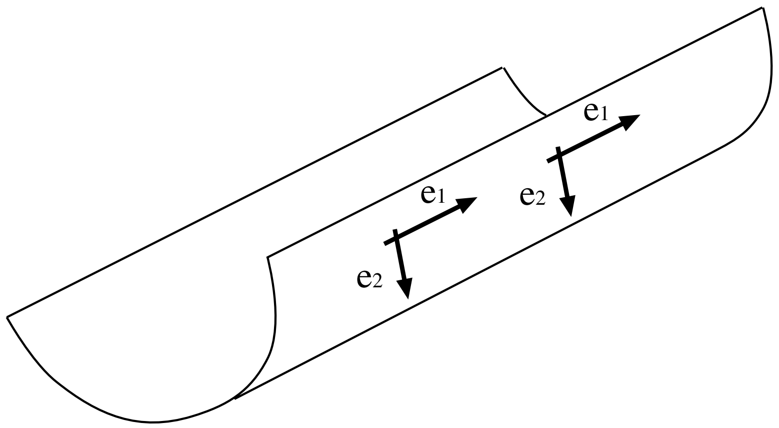


Figure 11.22. There is a natural coordinate system in each tangent plane on a cylinder. At every point we choose the direction that is parallel to the cylinder's axis ($\boldsymbol{e}_2$) and the direction that is tangent to the generating curve ($\boldsymbol{e}_1$). This coordinate system is well adapted to the structure of the cylinder.

system — a coordinate system within which the texture has desirable uniformity properties.

Now if we consider two regions of texture *in the image*, then one will, in general, be more distorted than the other. This relative distortion gives us the cue to the shape of the surface. Before we can recover any surface geometry, we need to determine what can be recovered in general.

The Geometry of an Orthographic View of a Cylinder

Shape from texture for orthographic views of cylinders is ambiguous. You can see this by inspecting the line of reasoning (the elliptical texture is a good example to keep in mind). We are assuming that the relative distortion of textures informs us about the shape of the surface, but as we move along the axis of the cylinder, the orientation of the $(\boldsymbol{e}_1, \boldsymbol{e}_2)$ coordinate system doesn't change with respect to the view. This means that we don't have any texture distortion in this direction. This means that we can determine the axis direction in the image (by looking for directions in which the texture doesn't distort).

It also means that, unless we know what a frontal texture looks like, it isn't possible to determine the angle between the axis of the cylinder and the plane. For example, in the case of the texture with elliptical spots, the texture could consist of very highly elongated ellipses viewed at an angle, or only slightly elongated ellipses viewed frontally (figure 11.25).

This also means we can't get the cross-section curve of the cylinder either (because we can't determine the orientation of the cylinder). The best we can do is to reconstruct a slice through the cylinder at some arbitrary angle to the axis. This

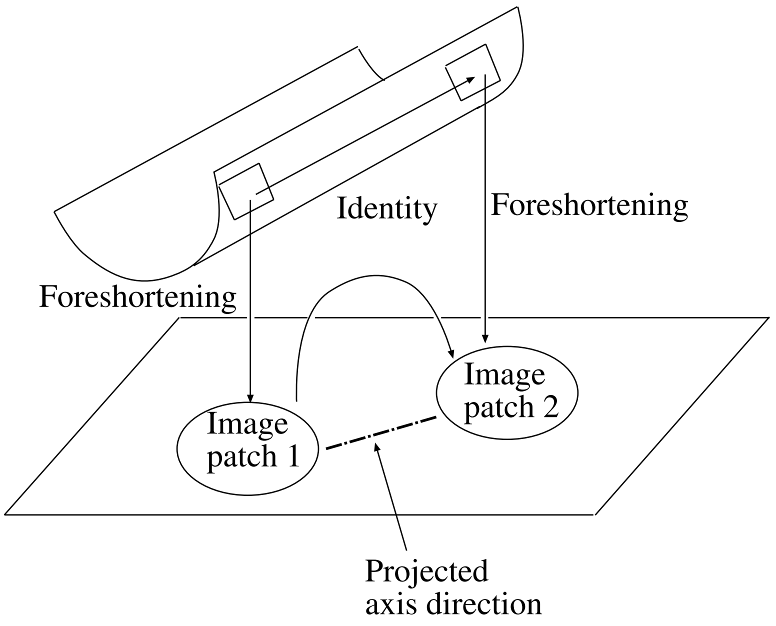


Figure 11.23. In the $(\boldsymbol{e}_1, \boldsymbol{e}_2)$ coordinate system on a cylinder, the texture at different patches is related by the identity. Now if we look at two image patches that *lie along the projected axis direction* the surface textures are foreshortened by the same amount; this means that they look the same already, and the transformation that makes them look most the same is the identity. In turn, we can use this property to identify the axis of the cylinder.

is a useful thing to do, because we can clearly tell different cylinders apart with this information. For example, if we slice a right circular cylinder at some arbitrary angle to the axis, the slice is an ellipse; but if we slice a cylinder with a square cross-section at an arbitrary angle to the axis, the cross-section is quadrilateral.

All this means that we can think about this problem as a one-dimensional reconstruction problem. We set the problem up as follows:

- Obtain the direction of the projected axis in the image, by determining the direction in which image texture transformations are the identity.
- Construct a perpendicular to that direction. We will study how texture distorts along this perpendicular. This will yield a slice of the cylinder (figure 11.26).

Reconstructing the Slice

As figure 11.28 indicates, reconstructing a slice of surface can be thought of as building up the graph of a function. We have one parameter — which we shall call

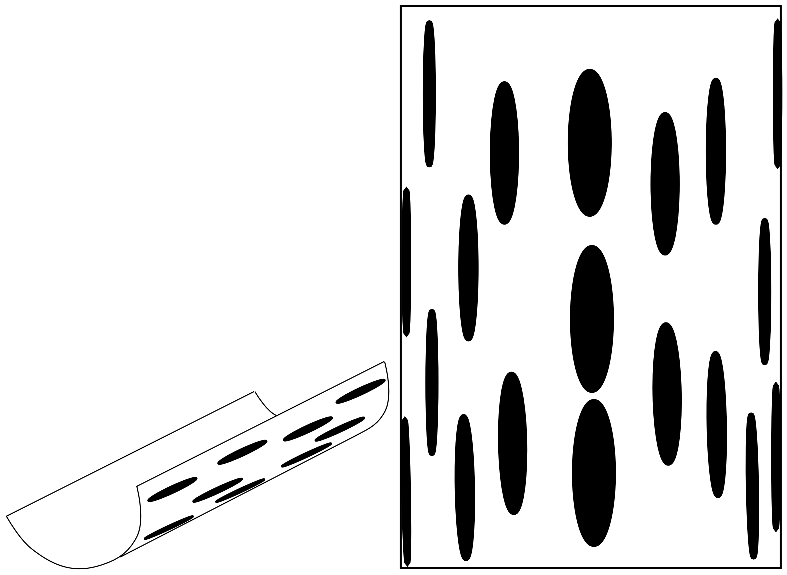


Figure 11.24. **Left:** A view of a spotted cylinder, where the spots are heavily elongated along the axis of the cylinder. In this view we have drawn in the boundaries of the cylinder, and shown the cylinder in 3/4 view, so that the geometry is clear. Notice that the spots foreshorten; as the cylinder turns away from the eye, the minor axis of the ellipse (which is oriented along the $\boldsymbol{e}_1$ direction) looks shorter. **Right:** a frontal view of this cylinder. Notice that the axis direction is obvious — it's the direction in which the texture is unchanged.

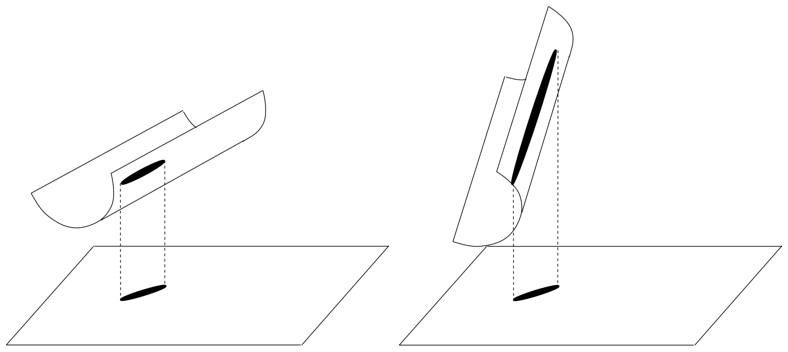


Figure 11.25. Shape from texture for a cylinder in an orthographic view is fundamentally ambiguous; we can't recover the angle between the axis of the cylinder and the image plane *unless* we know a frontal view of the texture. This is because the texture simply repeats as we move along the axis direction in the image; and the image view could come from (say) a long spot, slightly foreshortened (on the **left**) or a very long spot, heavily foreshortened (on the **right**).

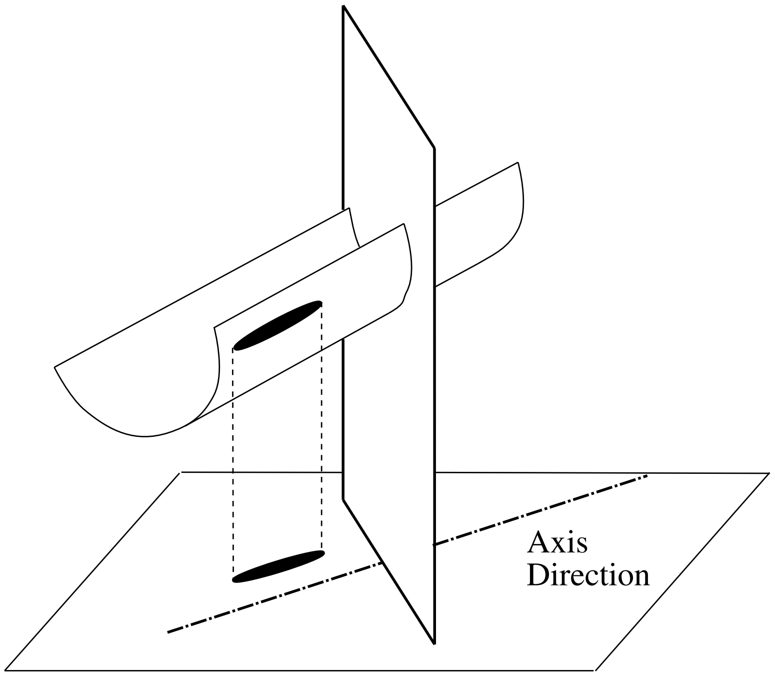


Figure 11.26. While we cannot recover any information about the slant of the cylinders axis with respect to the surface, we *can* obtain information about a cross-section through the cylinder. We construct an image direction perpendicular to the axis, and look at the foreshortening along that direction; this involves reconstructing a curve (figure 11.28).

t — running along the image line; we would like to reconstruct the depth to the surface as a function of t, which we shall write $f(t)$.

Now assume we have some texture element on the surface at $t = t_1$. We write $f'(t_1)$ for df/dt evaluated at $t = t_1$, etc. This is foreshortened by

$$\cos\theta_1 = \frac{1}{\sqrt{1 + f'(t_1)}}$$

Similarly, a texture element at $t = t_2$ is foreshortened by

$$\cos\theta_2 = \frac{1}{\sqrt{1 + f'(t_2)}}$$

Now if we construct the transformation that makes a patch of the image section along the perpendicular at $t = t_1$ look "most like" a patch of the the image section along the perpendicular at $t = t_2$, the transformation will have the form

$$\gamma(t_1, t_2) = \frac{\sqrt{1 + f'(t_1)^2}}{\sqrt{1 + f'(t_2)^2}}$$

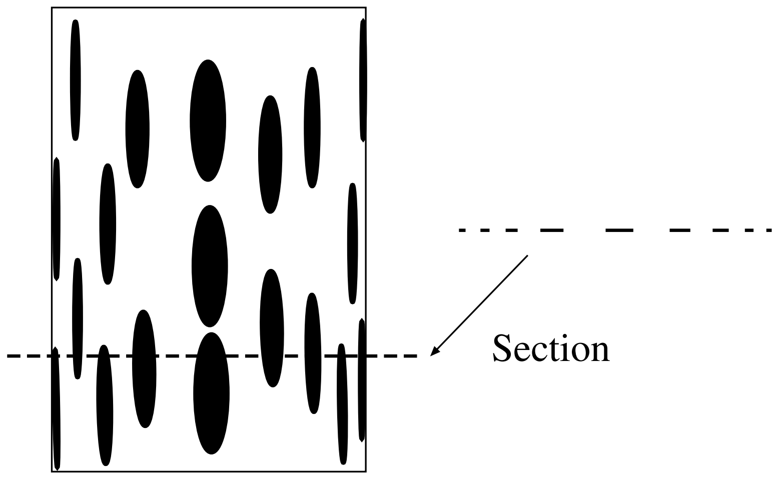


Figure 11.27. There are still foreshortening cues in a section perpendicular to the axis. We show the spotted cylinder of figure 11.24, but now with a section perpendicular to the axis direction (as in figure 11.26); notice the width of the black components of the section changes. This is primarily a foreshortening effect.

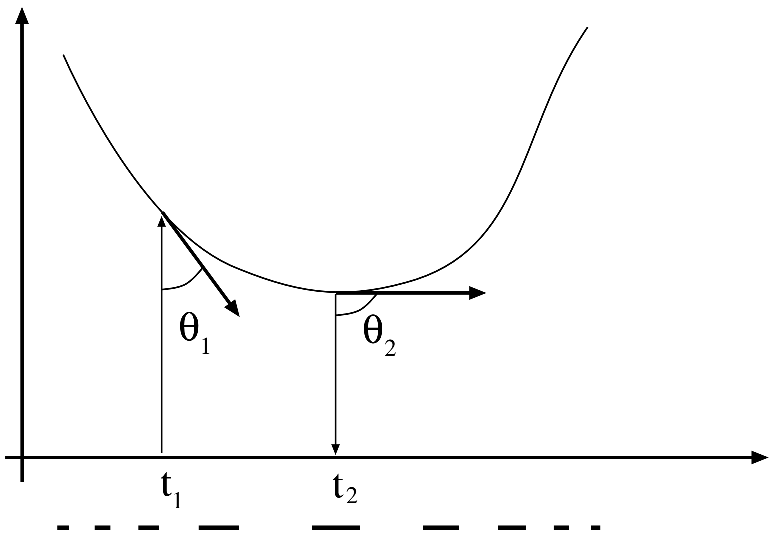


Figure 11.28. If we confine our attention to a plane perpendicular to the axis direction and to the image plane (as in figure 11.26), then reconstructing the curve where the cylinder intersects this plane is possible. We do this by regarding the curve as a function of a coordinate on the cross-section — call this coordinate t. Now the curve is given by $(t, f(t))$ and the transformation from image patch 1 to image patch 2 — which follows from the foreshortening illustrated in figure **??** — gives us information about the derivative of f; from this we can integrate to get the curve.

(i.e. we "unforeshorten" the texture to get the texture on the surface at $t = t_1$; this texture is the same as at $t = t_2$, which is then projected). For any values of t_1 and t_2, we can *measure* $\gamma(t_1, t_2)$ from the image.

Now choose some value of t to act as a reference point — call this t_r. At this point, we need to specify (1) the value of $f(t_r$ (we can't reconstruct this, because it's an orthographic camera) and (2) the orientation of the section (i.e. $f'(t_r)$). Once these have been specified, reconstructing the cross-section is a simple exercise in differential equations. In particular, we form $\phi(t) = \gamma(t, t_r)$; now we have

$$f'(t) = \sqrt{\phi(t)^2 - 1}$$

Since we know $f(t_r)$ and $f'(t_r)$, we have an initial value problem, which is easy to solve using standard methods. If you are careful, you will worry about the sign of the square root. We assume that the cylinder is continuous, so the only time things get interesting is when $f'(t) = 0$; in this case, we can't tell whether $f'(t + \epsilon)$ is positive or negative using a continuity argument. Geometrically, this is a natural ambiguity: when $f'(t)$ is zero, the curve is locally parallel to the viewing line — as we move along from these points, we can't tell whether the curve is turning towards or away from the viewing line (figure 11.29).

Currently, we are missing a parameter in the reconstruction of the cross-section — we supplied $f'(t_r)$, which is the orientation at the base point of the reconstruction. It turns out that we cannot choose this value freely. This is because foreshortening can make texture elements look smaller, but never bigger. In turn, if we use (say) the smallest texture elements as a base point, then we must ensure that the curve is sufficiently foreshortened there to account for the larger texture elements.

Formally, we have $\gamma(t, t_r)\sqrt{1 + f'(t_r)^2} \geq 1$ for any value of t. We can rewrite this inequality as

$$\frac{1}{\sqrt{1 + f'(t_r)^2}} \leq \gamma(t, t_r)$$

for any value of t. This means that if we have chosen t_r, then the value of $f'(t_r)^2$ that we choose for this point must have the property

$$\max_{t} \left\{ \frac{1}{\gamma(t, t_r)} \right\} \leq f'(t_r)^2$$

You may wish to use this inequality to choose a base point, by arguing that the most highly constrained point — the one where the lower bound on $f'(t_r)^2$ is highest — is a good choice. This is actually a bad idea, because at this point the texture is most foreshortened, and as a result difficult to measure. However, if we know that the cross-section is a closed curve, we have that the texture is frontal at any point where the lower bound is zero (exercises) and these points are clearly good base points.

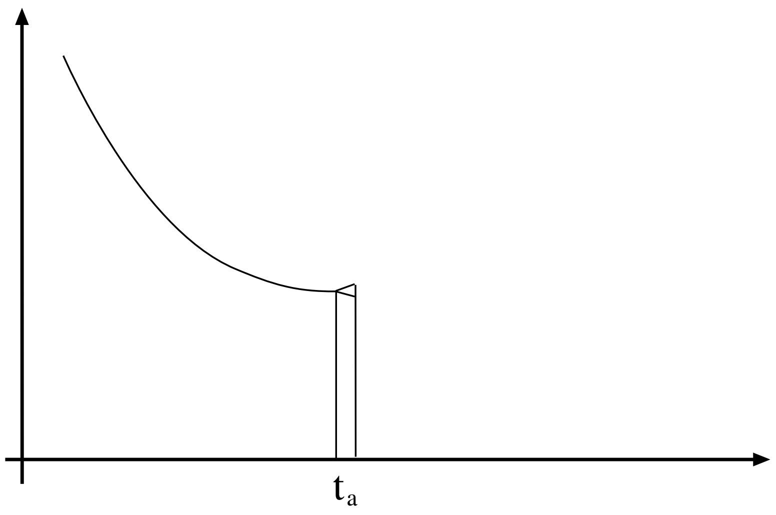


Figure 11.29. The reconstruction of cross-section curves is ambiguous, because we can obtain only $f'(t)^2$, and so the sign of the square root is ambiguous. If we assume that the curve has a continuous first derivative, this ambiguity disappears *except* wherever the curve is frontal — this corresponds to $\phi(t) = 1$, or $f'(t)^2 = 0$. In this case, the reconstruction at the next step could go up — corresponding to one sign of $f'(t)$ — or down — corresponding to the other sign. The drawing shows how the two cases yield the same foreshortening. This means that the ambiguity is fundamental; to break it, we need to make some other assumption about the shape of surfaces.

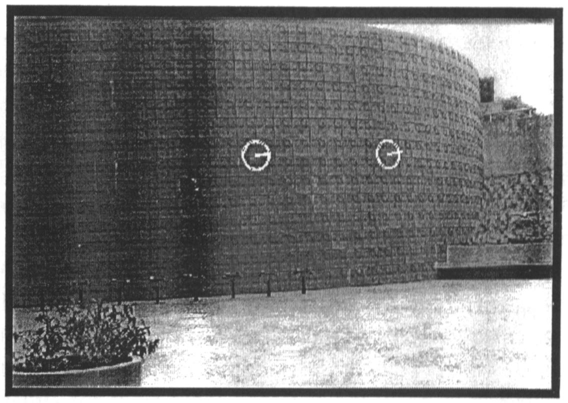


Figure 11.30. A textured cylinder, with a representation of surface normals obtained using a variant of the shape from texture method described in the text. *figure from the Malik and Rosenholtz, Computing Local Surface Orientation and Shape from Texture for Curved Surfaces, p.161, in the fervent hope, etc.*

11.5.2 Obtaining the Transformation between Image Patches

Since the texture is homogeneous, the translation component of the affine transformation is not interesting. In fact, we should not be able to measure it at all, and this constrains our test of similarity. One natural test of similarity can be obtained from a Fourier transform. We look at the lower spatial frequency components of the magnitude spectrum — these are unaffected by translation. We look only at the lower spatial frequency components, to avoid confusion from image noise. The low spatial frequency components should be similar for two patches drawn from the same homogeneous texture. From section **??**, when a patch is subject to a linear transformation, the magnitude spectrum — in fact, the whole Fourier transform — behaves quite simply. In particular, if $g(\boldsymbol{x})$ is some function of the image plane, and $G(\boldsymbol{\omega})$ is the Fourier transform of that function, we have

$$\mathcal{F}(g(\mathcal{A}\boldsymbol{x})) = \frac{1}{\mid \mathcal{A} \mid} G(\mathcal{A}^{-1}\boldsymbol{\omega})$$

This means we can obtain the transformation between the image patches by looking for the linear transformation that makes the magnitude spectrum of one patch look most like the magnitude spectrum of the other patch. This should be approached as a minimization problem (exercises).

11.6 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 [].

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

11.6.1 Shape from Texture

We have handled shape from texture in what we believe is a completely new way; it is certainly rather different from the traditional literature [**?**; **?**]. We have done so, basically, because our method is clearer (you can check this claim very easily!). In particular, our formalism exposes the crucial role of integrability rather clearly, and makes ambiguities clear, too. To our knowledge, no-one has done either doubly curved surfaces or the perspective case in this way; there appear to be no particular difficulties in doing this. We haven't expounded the doubly curved case here because (a) it requires a fair amount of technical differential geometry and (b) it requires a bit of work on partial differential equations as well. We encourage a creative and courageous reader to do this case in our stead.

We have been slightly sloppy in the account of shape from texture for cylinders, in that we estimated $\gamma(t_1, t_2)$ from a single section perpendicular to the axis. Obviously, this is a bad idea; it is also unneccessary. We could have estimated it using *every* section perpendicular to the axis. This is the right way to do things, but would have cluttered up the presentation. We encourage the reader with a sense of adventure to formulate and solve this estimation problem.

There is no really satisfactory definition of homogeneity of which we are aware. If a surface has constant Gaussian curvature, then there is an isometry connecting points on the surface, and this can be used to obtain a definition of homogeneity; but this isn't terribly satisfactory, because most interesting surfaces don't have constant Gaussian curvature. Furthermore, it leads to a fairly nasty formalism, because if we really are going to assume that a surface has constant Gaussian curvature, we should use this fact in reconstructing the surface. This means that there is some room for a study of other assumptions about what it means for a texture to be "constant" on a curved surface.

Most of what we have described applies only where the scale of the variation in the surface is much larger than the scale of variation in the texture — this should be a source of some unease, too. There is a great deal that can be done in this area, and the tools for understanding texture are now much better than they used to be.

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. In section 11.4.1, we stated that the configuration of a plane could not be recovered from an assumption that its texture is homogenous: but a plane is a cylinder. In this exercise, we explore why there is no contradiction.

   - Is there a unique choice of axis direction for a plane? use your answer to explain why the configuration cannot be recovered.
   - Assume that you have obtained an axis direction in the image. Give a closed form expression for $\gamma(t_1, t_2)$ for a plane.
   - Now explain why this form of γ also means you cannot recover the configuration of the plane.

4. Show that if a cylinder has a closed generating curve, then for any orthographic view, there is some point such that the bound

$$\max_{t} \left\{ \frac{1}{\gamma(t, t_r)} \right\} \leq f'(t_r)^2$$

   is zero. Show that this point is viewed frontally. Is this true of every point for which the bound is zero?

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.).