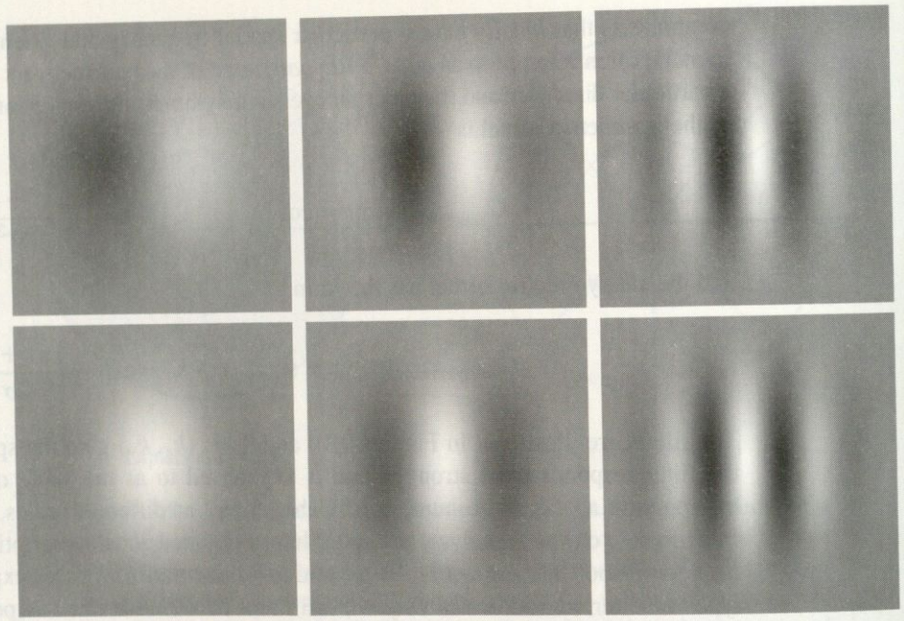

Figure 9.11 The images shows Gabor filter kernels as images, with mid-grey values representing zero, darker values representing negative numbers and lighter values representing positive numbers. The top row shows the antisymmetric component, and the bottom row shows the symmetric component. The scale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

9.2.3 Oriented Pyramids

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

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

9.3 APPLICATION: SYNTHESIZING TEXTURES FOR RENDERING

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

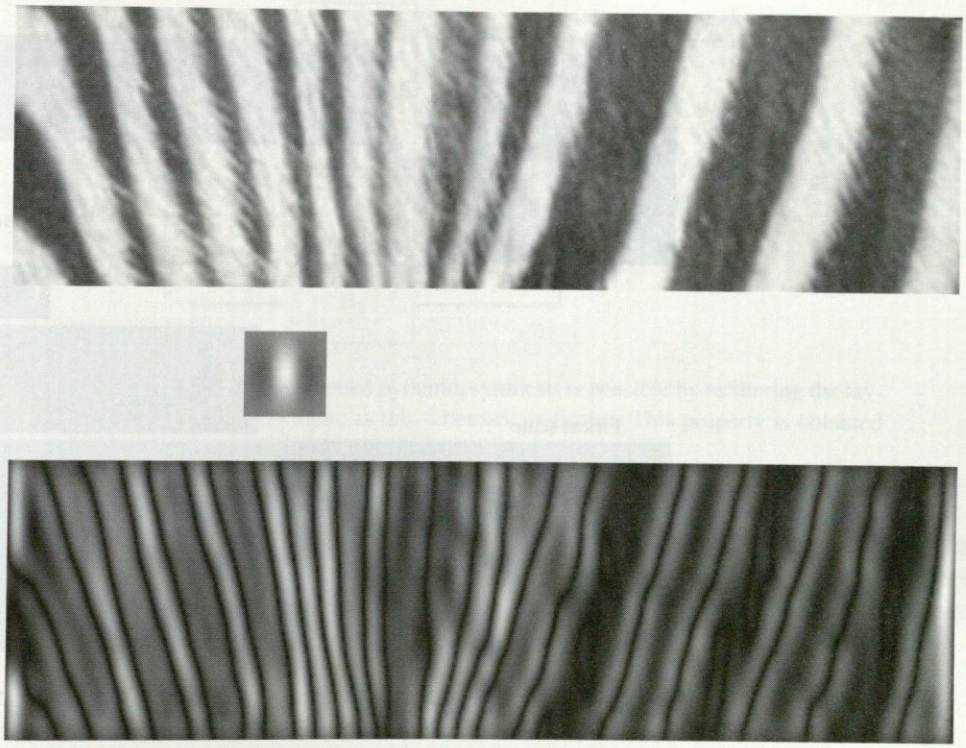

Figure 9.12 The image on the **top** shows a detail from an image of a zebra, chosen because it has a stripes at somewhat different scales and orientations. This has been convolved with the kernel in the center, which is a Gabor filter kernel. The image at the **bottom** shows the absolute value of the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

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

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

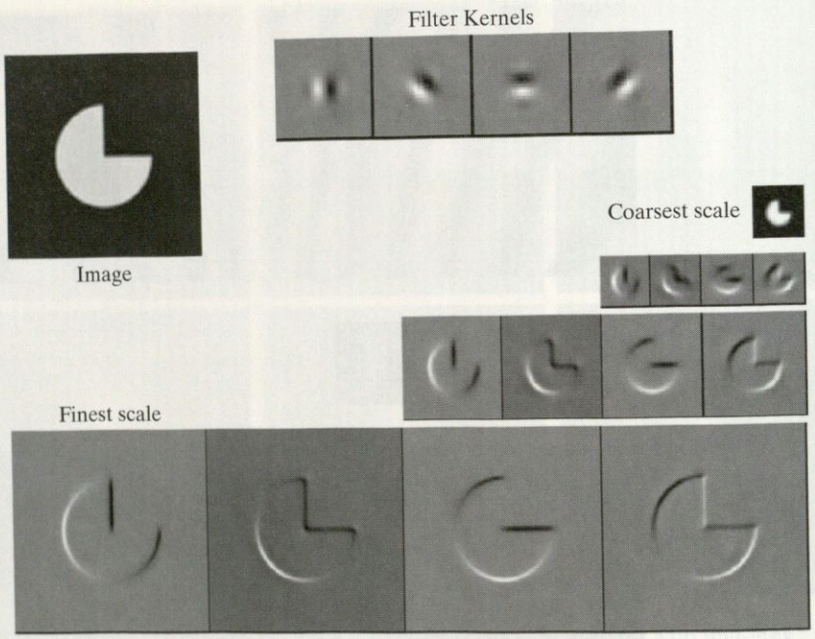

Figure 9.13 An oriented pyramid, formed from the image at the **top left**, with four orientations per layer. This is obtained by firstly decomposing an image into subbands which represent bands of spatial frequency (as with the Laplacian pyramid), and then applying oriented filters (**top right**) to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

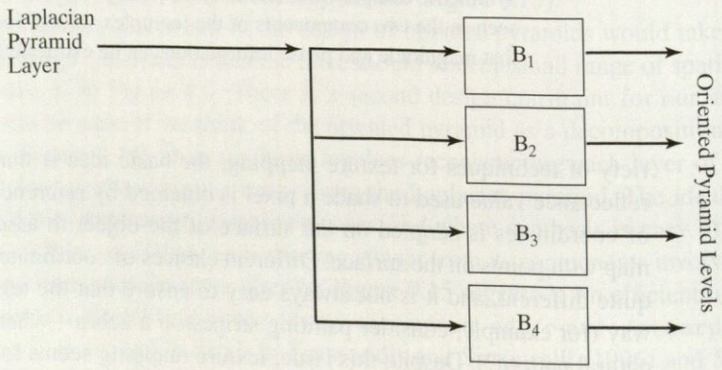

Figure 9.14 The oriented pyramid is obtained by taking layers of the Laplacian pyramid, and then applying oriented filters (represented in this schematic drawing by boxes). Each layer of the Laplacian pyramid represents a range of spatial frequencies; the oriented filters decompose this range of spatial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

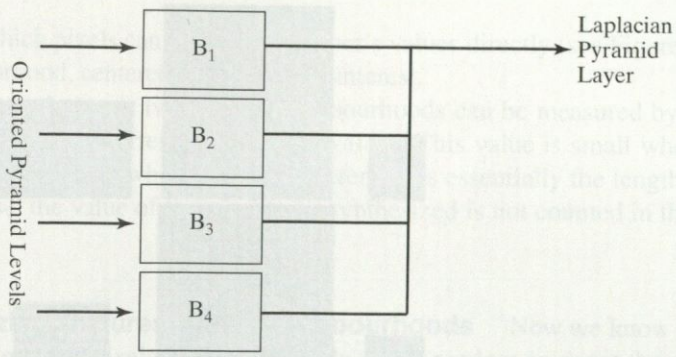

Figure 9.15 In the oriented pyramid, synthesis is possible by refiltering the layers and then adding them, as this schematic indicates. This property is obtained by appropriate choice of filters.

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

9.3.1 Homogeneity

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

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

9.3.2 Synthesis by Sampling Local Models

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

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

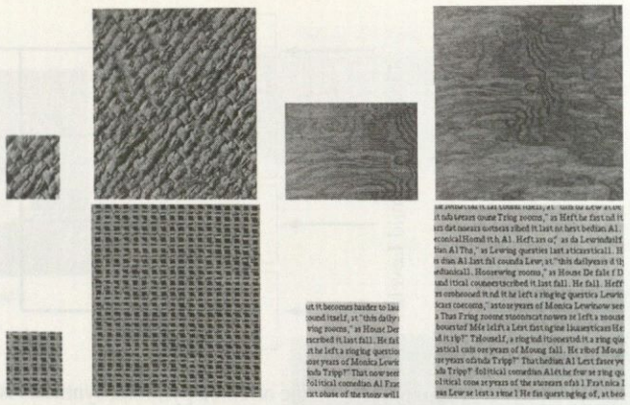

Figure 9.16 Efros' texture synthesis algorithm (Algorithm 9.3) matches neighbourhoods of the image being synthesized to the example image, and then chooses at random amongst the possible values reported by matching neighbourhoods. This means that the algorithm can reproduce complex spatial structures,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

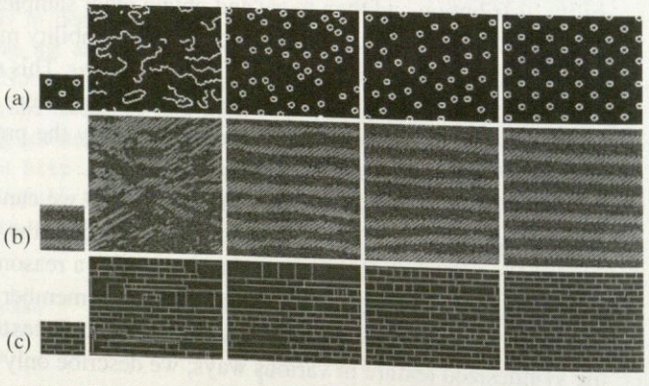

Figure 9.17 The size of the image neighbourhood to be matched makes a significant difference in Algorithm 9.3. In the figure, the textures at the right are synthesized from the small blocks on the **left**, using neighbourhoods that are increasingly large as one moves to the **right**. If very small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

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

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

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

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

Algorithm 9.3: Non-parametric Texture Synthesis

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

9.4 SHAPE FROM TEXTURE

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