event causes the interface to change state—a new menu item is
process continues. In some states, some events cause the syst
channel might change. All this means that a state machine is a

One way for vision to fit into this model is to provide
are generally few different kinds of event, and we know wh
care about in any particular state. As a result, the vision syst
either nothing or one of a small number of known kinds of
possible to build systems that meet these constraints.

A relatively small set of events is required to simulat
that look like button presses (e.g., to turn the television on or off), and even
motion (e.g., to increase the volume; it is possible to do this with buttons, too). With these
the television can be turned on, and an on-screen menu system navigated.

Finding Hands Freeman, Anderson and et al. (1998) produced an interface where an open hand turns the television on. This can be robust because all the system needs to do is determine whether there is a hand in view. Furthermore, the user will cooperate by holding their hand up and open. Because the user is expected to be a fairly constant distance from the camera—so the size of the hand is roughly known, and there is no need to search over scales—and in front of the television, the image region that needs to be searched to determine whether there is a hand is quite small.

The hand is held up in a fairly standard configuration and orientation to turn the television set on, and it usually appears at about the same distance from the television (so we know what it looks like). This means that a normalized correlation score is sufficient to find the hand. Any points in the correlation image where the score is high enough correspond to hands. This approach can be used to control volume and so on, as well as turn the television on and off. To do so, we need some notion of where the hand is going—to one side turns the volume up, to the other turns it down—and this can be obtained by comparing the position in the previous frame with that in the current frame. The system displays an iconic representation of its interpretation of hand position so the user has some feedback as to what the system is doing (Figure 7.16). Notice that an attractive feature of this approach is that it could be self-calibrating. In this approach, when you install your television set, you sit in front of it and show it your hand a few times to allow it to get an estimate of the scale at which the hand appears.

7.7 TECHNIQUE: SCALE AND IMAGE PYRAMIDS

Images look quite different at different scales. For example, the zebra's muzzle in Figure 7.17 can be described in terms of individual hairs—which might be coded in terms of the response of oriented filters that operate at a scale of a small number of pixels—or in terms of the stripes on the zebra. In the case of the zebra, we would not want to apply large filters to find the stripes. This is because these filters are inclined to spurious precision—we don't wish to represent the disposition of each hair on the stripe—inconvenient to build, and slow to apply. A more practical approach than applying large filters is to apply smaller filters to smoothed and resampled versions of the image.

7.7.1 The Gaussian Pyramid

An *image pyramid* is a collection of representations of an image. The name comes from a visual analogy. Typically, each layer of the pyramid is half the width and half the height of the previous layer; if we were to stack the layers on top of each other, a pyramid would result. In a **Gaussian**

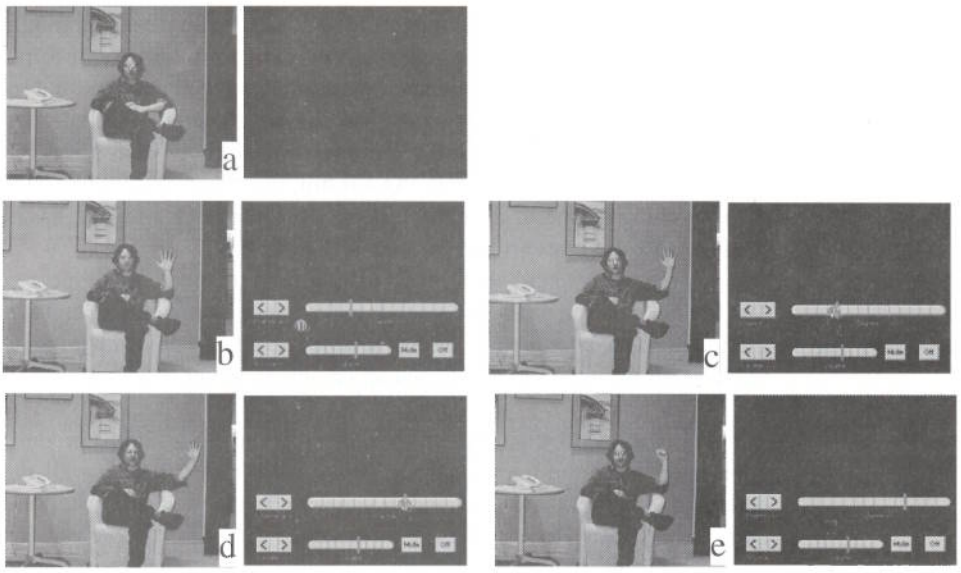

Figure 7.16 Examples of Freeman *et al.*'s system controlling a television set. Each state is illustrated with what the television sees on the **left** and what the user sees on the **right**. In **(a)**, the television is asleep, but a process is watching the user. An open hand causes the television to come on and show its user interface panel **(b)**. Focus on the panel tracks the movement of the user's open hand in **(c)**, and the user can change channel by using this tracking to move an icon on the screen in **(d)**. Finally, the user displays a closed hand in **(e)** to turn off the set. *Reprinted from "Computer Vision for Interactive Computer Graphics," by W.Freeman et al., IEEE Computer Graphics and Applications, 1998 © 1998 IEEE*

Algorithm 7.2: Forming a Gaussian Pyramid

Set the finest scale layer to the image
For each layer, going from next to finest to coarsest
  Obtain this layer by smoothing the next finest
  layer with a Gaussian, and then subsampling it
end

pyramid, each layer is smoothed by a symmetric Gaussian kernel and resampled to get the next layer (Figure 7.17). These pyramids are most convenient if the image dimensions are a power of two or a multiple of a power of two. The smallest image is the most heavily smoothed; the layers are often referred to as *coarse scale* versions of the image.

With a little notation, we can write simple expressions for the layers of a Gaussian pyramid. The operator $S^{\downarrow}$ downsamples an image; in particular, the j, kth element of $S^{\downarrow}(\mathcal{I})$ is the $2j, 2k$th element of $\mathcal{I}$. The nth level of a pyramid $P(\mathcal{I})$ is denoted $P(\mathcal{I})_n$. With this notation, we have

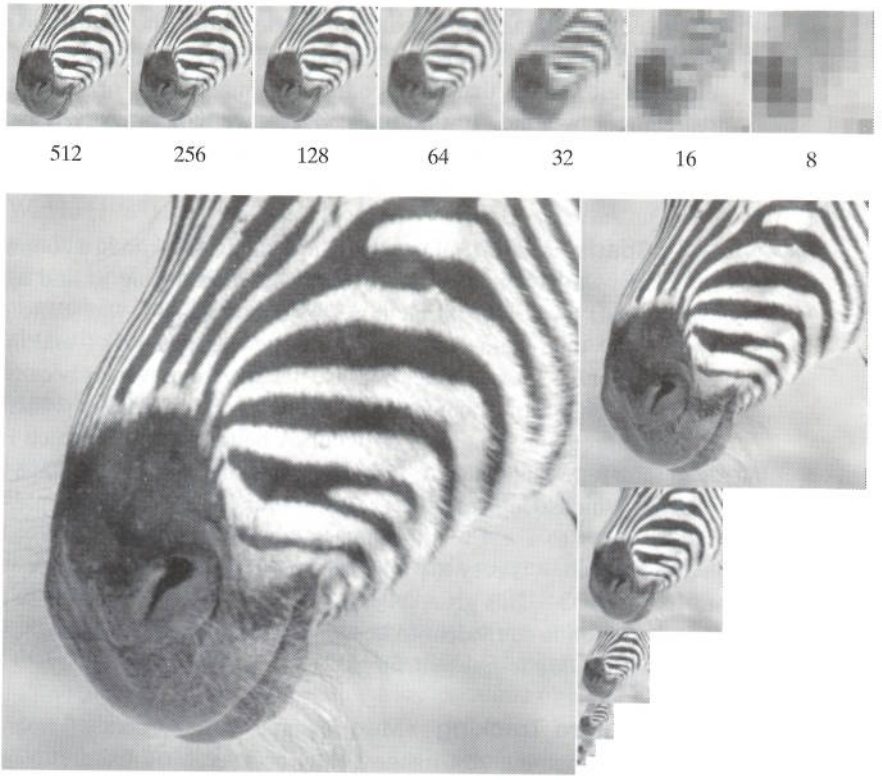

Figure 7.17 A Gaussian pyramid of images running from 512x512 to 8x8. On the top row, we have shown each image at the same size (so that some have bigger pixels than others), and the lower part of the figure shows the images to scale. Notice that if we convolve each image with a fixed size filter, it responds to quite different phenomena. An 8x8 pixel block at the finest scale might contain a few hairs; at a coarser scale, it might contain an entire stripe; and at the coarsest scale, it contains the animal's muzzle.

$$\begin{aligned} P_{\text{Gaussian}}(\mathcal{I})_{n+1} &= S^{\downarrow}(G_\sigma ** P_{\text{Gaussian}}(\mathcal{I})_n) \\ &= S^{\downarrow} G_\sigma (P_{\text{Gaussian}}(\mathcal{I})_n) \end{aligned}$$

(where we have written G_σ for the linear operator that takes an image to the convolution of that image with a Gaussian). The finest scale layer is the original image:

$$P_{\text{Gaussian}}(\mathcal{I})_1 = \mathcal{I}.$$

7.7.2 Applications of Scaled Representations

Gaussian pyramids are useful because they make it possible to extract representations of different types of structure in an image. There are three standard applications.

Search over Scale Numerous objects can be represented as small image patterns. A standard example is a frontal view of a face. Typically, at low resolution, frontal views of faces have a quite distinctive pattern: The eyes form dark pools, under a dark bar (the eyebrows),

separated by a lighter bar (specular reflections from the nose), and above a dark bar (the mouth). There are various methods for finding faces that exploit these properties (see chapter 22). These methods all assume that the face lies in a small range of scales. All other faces are found by searching a pyramid. To find bigger faces, we look at coarser scale layers, and to find smaller faces we look at finer scale layers. This useful trick applies to many different kinds of feature, as we see in the chapters that follow.

Spatial Search One application is spatial search, a common theme in computer vision. Typically, we have a point in one image and are trying to find a point in a second image that corresponds to it. This problem occurs in stereopsis—where the point has moved because the two images are obtained from different viewing positions—and in motion analysis—where the image point has moved either because the camera moved or because it is on a moving object.

Searching for a match in the original pairs of images is inefficient because we may have to wade through a great deal of detail. A better approach, which is now pretty much universal, is to look for a match in a heavily smoothed and resampled image and then refine that match by looking at increasingly detailed versions of the image. For example, we might reduce 1024×1024 images down to 4×4 versions, match those, and then look at 8×8 versions (because we know a rough match, it is easy to refine it); we then look at 16×16 versions, and so on, all the way up to 1024×1024. This gives an extremely efficient search because a step of a single pixel in the 4×4 version is equivalent to a step of 256 pixels in the 1024×1024 version. This strategy is known as *coarse-to-fine matching*.

Feature Tracking Most features found at coarse levels of smoothing are associated with large, high-contrast image events because for a feature to be marked at a coarse scale a large pool of pixels need to agree that it is there. Typically, finding coarse scale phenomena misestimates both the size and location of a feature. For example, a single pixel error in a coarse-scale image represents a multiple pixel error in a fine-scale image.

At fine scales, there are many features, some of which are associated with smaller, low-contrast events. One strategy for improving a set of features obtained at a fine scale is to track features across scales to a coarser scale and accept only the fine scale features that have identifiable parents at a coarser scale. This strategy, known as *feature tracking* in principle, can suppress features resulting from textured regions (often referred to as noise) and features resulting from real noise.

7.8 NOTES

We don't claim to be exhaustive in our treatment of linear systems, but it wouldn't be possible to read the literature on filters in vision without a grasp of the ideas in this chapter. We have given a fairly straightforward account here; more details on these topics can be found in the excellent books by Bracewell (1995), (2000).

Real Imaging Systems versus Shift Invariant Linear Systems

Imaging systems are only approximately linear. Film is not linear—it does not respond to weak stimuli, and it saturates for bright stimuli—but one can usually get away with a linear model within a reasonable range. CCD cameras are linear within a working range. They give a small, but non zero response to a zero input as a result of thermal noise (which is why astronomers cool their cameras) and they saturate for very bright stimuli. CCD cameras often contain electronics that transforms their output to make them behave more like film because consumers are used to

film. Shift invariance is approximate as well because lenses tend to distort responses near the image boundary. Some lenses—fish-eye lenses are a good example—are not shift invariant.

Scale

There is a large body of work on scale space and scaled representations. The origins appear to lie with Witkin (1983) and the idea was developed by Koenderink and van Doorn (1986). Since then, a huge literature has sprung up (one might start with ter Haar Romeny, Florack, Koenderink and Viergever, 1997 or Nielsen, Johansen, Olsen and Weickert, 1999). We have given only the briefest picture here because the analysis tends to be quite tricky. The usefulness of the techniques is currently hotly debated, too.

Anisotropic Scaling

One important difficulty with scale space models is that the symmetric Gaussian smoothing process tends to blur out edges rather too aggressively for comfort. For example, if we have two trees near one another on a skyline, the large-scale blobs corresponding to each tree may start merging before all the small-scale blobs have finished. This suggests that we should smooth differently at edge points than at other points. For example, we might make an estimate of the magnitude and orientation of the gradient: For large gradients, we would then use an oriented smoothing operator that smoothed aggressively perpendicular to the gradient and little along the gradient; for small gradients, we might use a symmetric smoothing operator. This idea used to be known as *edge-preserving smoothing*.

In the modern, more formal version, due to Perona and Malik (1990*a,b*), we notice the scale space representation family is a solution to the *diffusion equation*

$$\begin{aligned}\frac{\partial \Phi}{\partial \sigma} &= \frac{\partial^2 \Phi}{\partial x^2} + \frac{\partial^2 \Phi}{\partial y^2} \\ &= \nabla^2 \Phi,\end{aligned}$$

with the initial condition

$$\Phi(x, y, 0) = \mathcal{I}(x, y)$$

If this equation is modified to have the form

$$\begin{aligned}\frac{\partial \Phi}{\partial \sigma} &= \nabla \cdot (c(x, y, \sigma) \nabla \Phi) \\ &= c(x, y, \sigma) \nabla^2 \Phi + (\nabla c(x, y, \sigma)) \cdot (\nabla \Phi)\end{aligned}$$

with the same initial condition, then if $c(x, y, \sigma) = 1$, we have the diffusion equation we started with, and if $c(x, y, \sigma) = 0$ there is no smoothing. We assume that c does not depend on σ. If we knew where the edges were in the image, we could construct a mask that consisted of regions where $c(x, y) = 1$, isolated by patches along the edges where $c(x, y) = 0$; in this case, a solution would smooth *inside* each separate region, but not over the edge. Although we do not know where the edges are — the exercise would be empty if we did—we can obtain reasonable choices of $c(x, y)$ from the magnitude of the image gradient. If the gradient is large, then c should be small and vice versa. There is a substantial literature dealing with this approach; a good place to start is ter Haar Romeny (1994).