CHAPTER 8

Detecting Edges

An object is separated from its background in an image by an *occluding contour*. Draw a path in the image that crosses such a contour. On one side, pixels lie on the object, and on the other, the background. Finding occluding contours is an important challenge, because the outline of an object—which is one cue to its shape—is formed by occluding contours. We can expect that, at occluding contours, there are often substantial changes in image brightness. There are other important causes of sharp changes in image brightness, including sharp changes in albedo, in surface orientation, or in illumination. Each can provide interesting information about the objects in the world. Occluding contours carry shape information; sharp changes in albedo carry texture information; sharp changes in surface orientation tell us about shape; and illumination changes might tell us where the sun is. All this means it is useful to find and reason about sharp changes in image intensity.

Sharp changes in brightness cause large image gradients. Section 8.1 describes methods to extract image gradients. One important use of gradients is to find *edges* or **edge points**, where the brightness changes particularly sharply (Section 8.2.1). The edge points produced tend to be sensitive to changes in contrast (i.e., the size of the difference in brightness across the edge), which can result from changes in lighting. Often, it is helpful to use the orientation of the gradient vector (Section 8.2.2), which does not depend on contrast. For example, at corners, the image gradient vector swings sharply in orientation.

Corners are important, because they are easy to match from image to image. At a corner, we expect to see strong image gradients that turn fast locally, and this cue yields a corner detector (Section 9.1.1). If we can describe a neighborhood around a corner, we can match descriptions across images. Such matching is an important basic subroutine in computer vision.

8.1 COMPUTING THE IMAGE GRADIENT

For an image $\mathcal{I}$, the gradient is

$$\nabla \mathcal{I} = (\frac{\partial \mathcal{I}}{\partial x}, \frac{\partial \mathcal{I}}{\partial y})^T,$$

which we could estimate by observing that

$$\frac{\partial \mathcal{I}}{\partial x} = \lim_{\delta x \to 0} \frac{\mathcal{I}(x + \delta x, y) - \mathcal{I}(x, y)}{\delta x} \approx \mathcal{I}_{i+1,j} - \mathcal{I}_{i,j}.$$

By the same argument, $\partial \mathcal{I}/\partial y \approx \mathcal{I}_{i,j+1} - \mathcal{I}_{i,j}$. These kinds of derivative estimates are known as *finite differences*. Image noise tends to result in pixels not looking like their neighbors, so that simple finite differences tend to give strong responses to noise. As a result, just taking one finite difference for x and one for y gives noisy

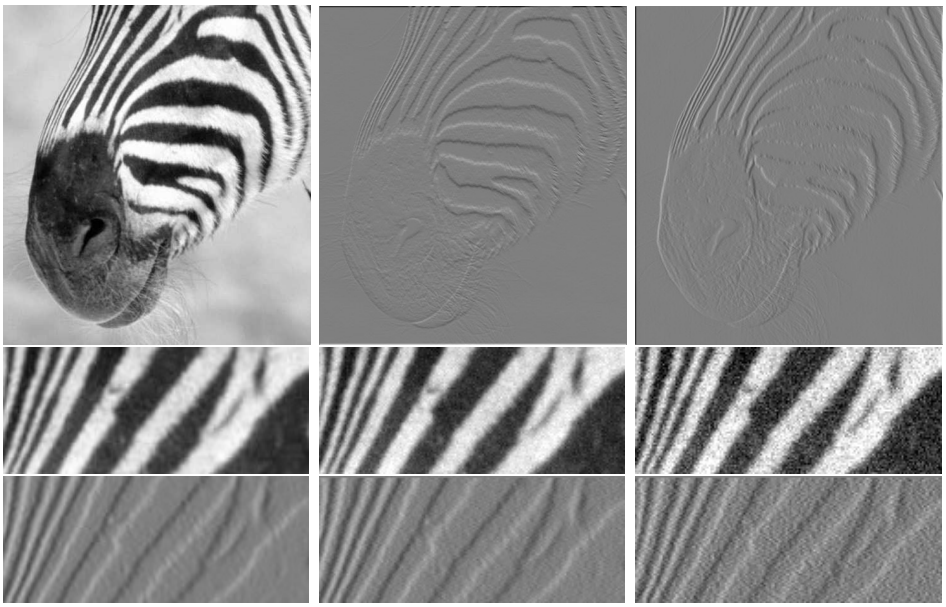

FIGURE 8.1: *The* **top row** *shows estimates of derivatives obtained by finite differences. The image at the* **left** *shows a detail from a picture of a zebra. The* **center** *image shows the partial derivative in the y-direction—which responds strongly to horizontal stripes and weakly to vertical stripes—and the* **right** *image shows the partial derivative in the x-direction—which responds strongly to vertical stripes and weakly to horizontal stripes. However, finite differences respond strongly to noise. The image at* **center left** *shows a detail from a picture of a zebra; the next image in the row is obtained by adding a random number with zero mean and normal distribution ($\sigma = 0.03$; the darkest value in the image is 0, and the lightest 1) to each pixel; and the third image is obtained by adding a random number with zero mean and normal distribution ($\sigma = 0.09$) to each pixel. The* **bottom row** *shows the partial derivative in the x-direction of the image at the head of the row. Notice how strongly the differentiation process emphasizes image noise; the derivative figures look increasingly grainy. In the derivative figures, a mid-gray level is a zero value, a dark gray level is a negative value, and a light gray level is a positive value.*

gradient estimates. The way to deal with this problem is to smooth the image and then differentiate it (we could also smooth the derivative).

As Figure 8.1 suggests, finite differences give a most unsatisfactory estimate of the derivative. This is because finite differences respond strongly (i.e., have an output with large magnitude) at fast changes, and fast changes are characteristic of noise. Roughly, this is because image pixels tend to look like one another. For example, if we had bought a discount camera with some pixels that were stuck at either black or white, the output of the finite difference process would be large at those pixels because they are, in general, substantially different from their neighbors. All this suggests that some form of smoothing is appropriate before differentiation.

Smoothing works because, in general, any image gradient of significance to us has effects over a pool of pixels. For example, the contour of an object can result in a long chain of points where the image derivative is large. As another example, a corner typically involves many tens of pixels. If the noise at each pixel is independent and additive, then large image derivatives caused by noise are a local event. Smoothing the image before we differentiate will tend to suppress noise at the scale of individual pixels, because it will tend to make pixels look like their neighbors. However, gradients that are supported by evidence over multiple pixels will tend not to be smoothed out. This suggests differentiating a smoothed image (Figure 8.2).

8.1.1 Derivative of Gaussian Filters

Smoothing an image and then differentiating it is the same as convolving it with the derivative of a smoothing kernel. This fact is most easily seen by thinking about continuous convolution.

First, differentiation is linear and shift invariant. This means that there is some kernel—we dodge the question of what it looks like—that differentiates. That is, given a function $I(x, y)$,

$$\frac{\partial I}{\partial x} = K_{(\partial/\partial x)} * I.$$

Now we want the derivative of a smoothed function. We write the convolution kernel for the smoothing as S. Recalling that convolution is associative, we have

$$(K_{(\partial/\partial x)} * (S * I)) = (K_{(\partial/\partial x)} * S) * I = (\frac{\partial S}{\partial x}) * I.$$

This fact appears in its most commonly used form when the smoothing function is a Gaussian; we can then write

$$\frac{\partial (G_\sigma * I)}{\partial x} = (\frac{\partial G_\sigma}{\partial x}) * I,$$

that is, we need only convolve with the derivative of the Gaussian, rather than convolve and then differentiate. As discussed in Section **??**, smoothed derivative filters look like the effects they are intended to detect. The x-derivative filters look like a vertical light blob next to a vertical dark blob (an arrangement where there is a large x-derivative), and so on (Figure **??**). Smoothing results in much smaller noise responses from the derivative estimates (Figure 8.2).

The choice of σ used in estimating the derivative is often called the *scale* of the smoothing. Scale has a substantial effect on the response of a derivative filter. Assume we have a narrow bar on a constant background, rather like the zebra's whisker. Smoothing on a scale smaller than the width of the bar means that the filter responds on each side of the bar, and we are able to resolve the rising and falling edges of the bar. If the filter width is much greater, the bar is smoothed into the background and the bar generates little or no response (Figure 8.3).

The most usual noise model is the *additive stationary Gaussian noise* model, where each pixel has added to it a value chosen independently from the same Gaussian probability distribution. This distribution almost always has zero mean.

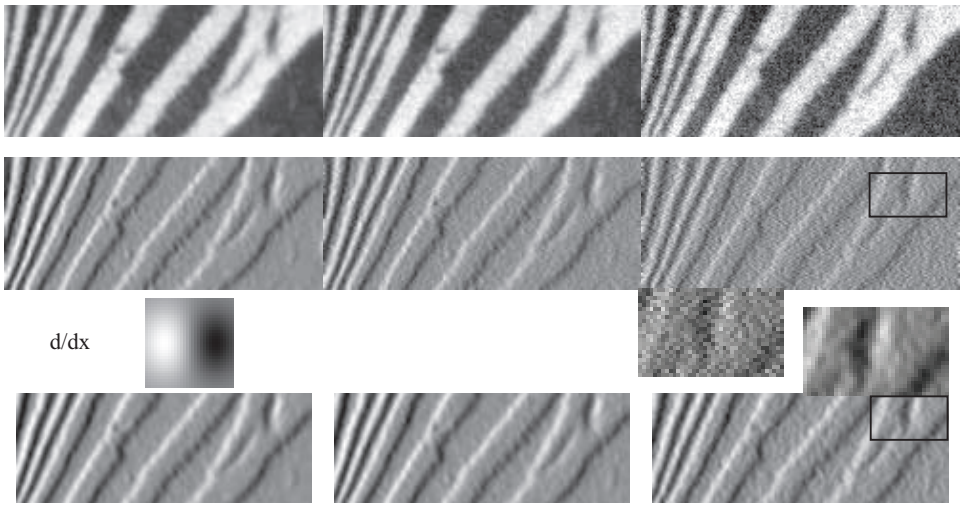

FIGURE 8.2: *Derivative of Gaussian filters are less extroverted in their response to noise than finite difference filters. The image at* **top left** *shows a detail from a picture of a zebra;* **top center** *shows the same image corrupted by zero mean stationary additive Gaussian noise, with* $\sigma = 0.03$ *(pixel values range from 0 to 1).* **Top right** *shows the same image corrupted by zero mean stationary additive Gaussian noise, with* $\sigma = 0.09$*. The second row shows the finite difference in the* x*-direction of each image. These images are scaled so that zero is mid-gray, the most negative pixel is dark, and the most positive pixel is light; we used a different scaling for each image. Notice how the noise results in occasional strong derivatives, shown by a graininess in the derivative maps for the noisy images. The final row shows the partial derivative in the* x*-direction of each image, in each case estimated by a derivative of Gaussian filter with* σ *one pixel. Again, these images are scaled so that zero is mid-gray, the most negative pixel is dark, and the most positive pixel is light; we used a different scaling for each image. The images are smaller than the input image, because we used a* 13×13 *pixel discrete kernel. This means that the six rows (resp. columns) on the top and bottom of the image (resp. left and right) cannot be evaluated exactly, because for these rows the kernel covers some points outside the image; we have omitted these values. Notice how the smoothing helps reduce the impact of the noise; this is emphasized by the detail images (between the second and final row), which are doubled in size. The details show patches that correspond from the finite difference image and the smoothed derivative estimate. We show a derivative of Gaussian filter kernel, which (as we expect) looks like the structure it is supposed to find. This is not to scale (it'd be extremely small if it were).*

The standard deviation is a parameter of the model. The model is intended to describe thermal noise in cameras and is illustrated in Figure 8.5.

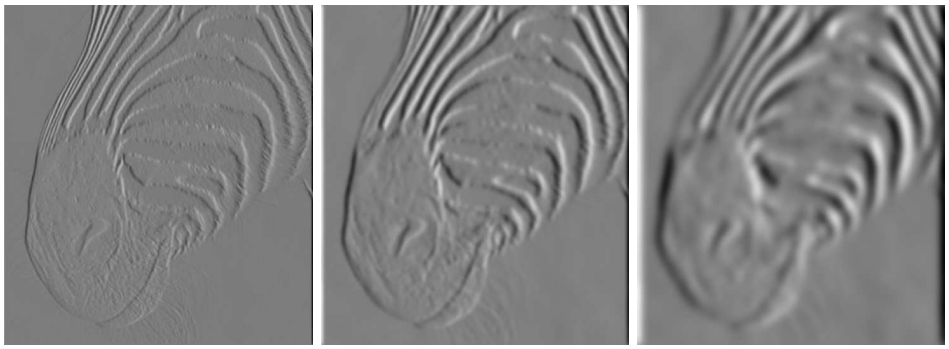

FIGURE 8.3: *The scale (i.e., σ) of the Gaussian used in a derivative of Gaussian filter has significant effects on the results. The three images show estimates of the derivative in the x direction of an image of the head of a zebra obtained using a derivative of Gaussian filter with σ one pixel, three pixels, and seven pixels (***left*** to* ***right****). Note how images at a finer scale show some hair, the animal's whiskers disappear at a medium scale, and the fine stripes at the top of the muzzle disappear at the coarser scale.*

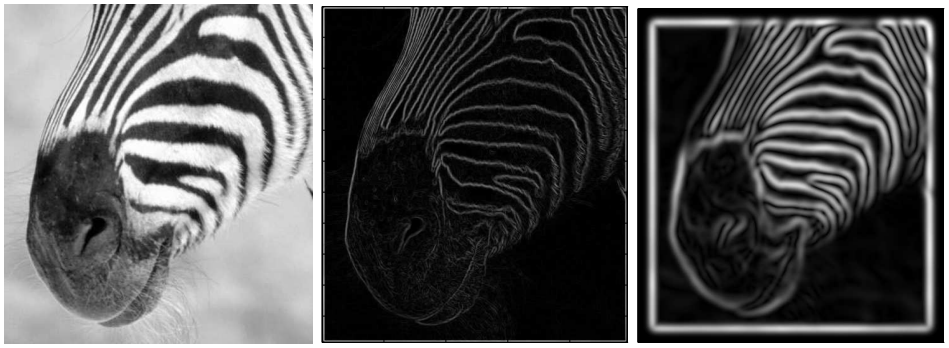

FIGURE 8.4: *The gradient magnitude can be estimated by smoothing an image and then differentiating it. This is equivalent to convolving with the derivative of a smoothing kernel. The extent of the smoothing affects the gradient magnitude; in this figure, we show the gradient magnitude for the figure of a zebra at different scales. At the* ***center****, gradient magnitude estimated using the derivatives of a Gaussian with $\sigma = 1$ pixel; and on the* ***right****, gradient magnitude estimated using the derivatives of a Gaussian with $\sigma = 2$ pixel. Notice that large values of the gradient magnitude form thick trails.*

8.2 REPRESENTING THE IMAGE GRADIENT

There are two important representations of the image gradient. The first is to compute edges, where there are very fast changes in brightness. These are usually seen as points where the magnitude of the gradient is extremal (Section 8.2.1). The second is to use gradient orientations, which are largely independent of illumination

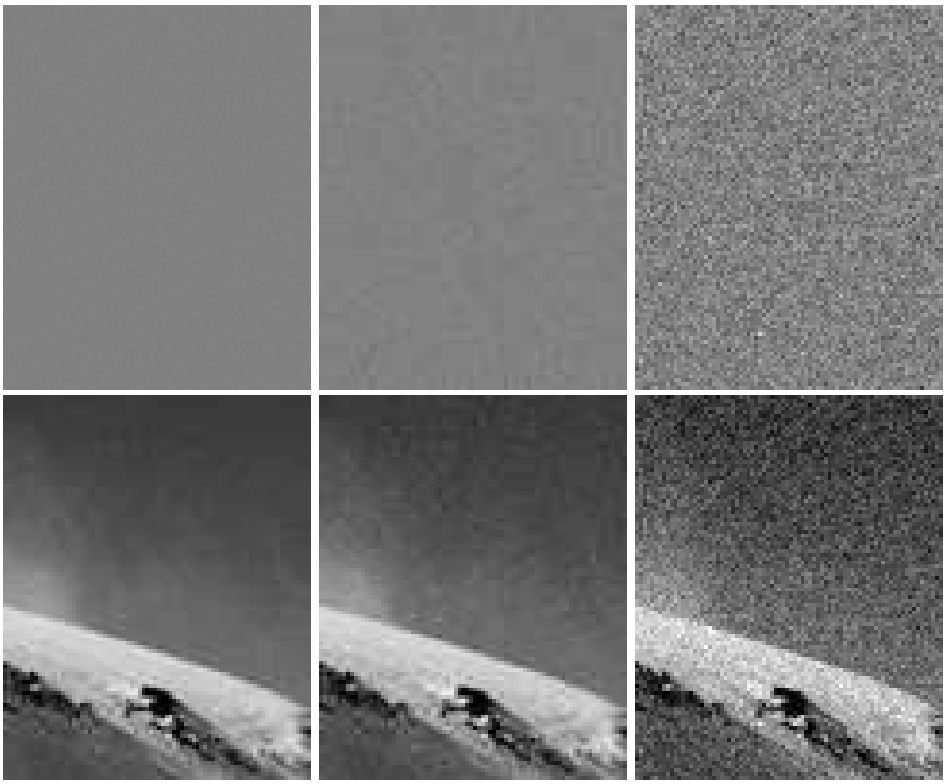

FIGURE 8.5: *The* **top row** *shows three realizations of a stationary additive Gaussian noise process. We have added half the range of brightnesses to these images to show both negative and positive values of noise. From left to right, the noise has standard deviation* $1/256$*,* $4/256$*, and* $16/256$ *of the full range of brightness, respectively. This corresponds roughly to bits zero, two, and five of a camera that has an output range of eight bits per pixel. The* **lower row** *shows this noise added to an image. In each case, values below zero or above the full range have been adjusted to zero or the maximum value accordingly.*

intensity (Section 8.8).

8.2.1 Gradient-Based Edge Detectors

We think of sharp changes in image intensity as lying on curves in the image, which are known as *edges*; the curves are made up of *edge points*. Many effects can cause edges; worse, each effect that can cause an edge is not guaranteed to cause an edge. For example, an object may happen to be the same intensity as the background, and so the occluding contour will not result in an edge. This means that interpreting edge points can be very difficult. Nonetheless, they are worth finding.

In the most common method for finding edges, we start by computing an estimate of the gradient magnitude. The gradient magnitude is large along a thick

```
Form an estimate of the image gradient
Compute the gradient magnitude
While there are points with high gradient
magnitude that have not been visited
  Find a start point that is a local maximum in the
    direction perpendicular to the gradient
    erasing points that have been checked
  While possible, expand a chain through
    the current point by:
      1) predicting a set of next points, using
           the direction perpendicular to the gradient
      2) finding which (if any) is a local maximum
           in the gradient direction
      3) testing if the gradient magnitude at the
           maximum is sufficiently large
      4) leaving a record that the point and
           neighbors have been visited
      record the next point, which becomes the current point
  end
end
```

Algorithm 8.1: *Gradient-Based Edge Detection.*

trail in the image (Figure 8.4), but occluding contours are curves, so we must obtain a curve of the most distinctive points on this trail.

There is clearly no objective definition, and we can proceed by reasonable intuition. The gradient magnitude can be thought of as a chain of low hills. Marking local maxima would mark isolated points—the hilltops in the analogy. A better criterion is to slice the gradient magnitude along the gradient direction, which should be perpendicular to the edge, and mark the points along the slice where the magnitude is maximal. This would get a chain of points along the crown of the hills in our chain. Each point in the chain can be used to predict the location of the next point, which will be in a direction roughly at right angles to the gradient at the edge point (Figure 8.6). Forming these chains is called *nonmaximum suppression*. It is relatively straightforward to identify the location of these chains at a resolution finer than that of the pixel grid (Figure 8.6).

There are too many of these chains to come close to being a reasonable representation of object boundaries. In part, this is because we have marked maxima of the gradient magnitude without regard to how large these maxima are. It is more usual to apply a threshold test to ensure that the maxima are greater than some lower bound. This in turn leads to broken edge curves. The usual trick for dealing with this is to use *hysteresis*; we have two thresholds and refer to the *larger* when starting an edge chain and the *smaller* while following it. The trick often results in an improvement in edge outputs. These considerations yield Algorithm 8.1. Most current edgefinders follow these lines.

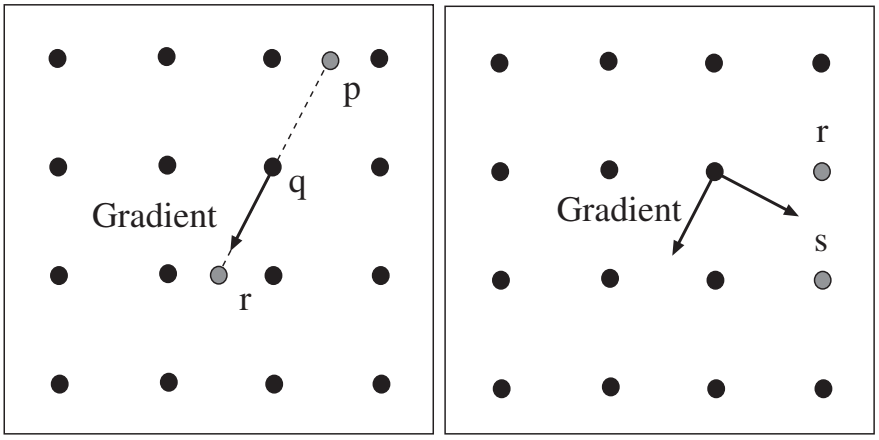

FIGURE 8.6: *Nonmaximum suppression obtains points where the gradient magnitude is at a maximum* along the direction of the gradient. *The figure on the* **left** *shows how we reconstruct the gradient magnitude. The dots are the pixel grid. We are at pixel* $\mathbf{q}$*, attempting to determine whether the gradient is at a maximum; the gradient direction through* $\mathbf{q}$ *does not pass through any convenient pixels in the forward or backward direction, so we must interpolate to obtain the values of the gradient magnitude at* $\mathbf{p}$ *and* $\mathbf{r}$*. If the value at* $\mathbf{q}$ *is larger than both,* $\mathbf{q}$ *is an edge point. Typically, the magnitude values are reconstructed with a linear interpolate, which in this case would use the pixels to the left and right of* $\mathbf{p}$ *and* $\mathbf{r}$*, respectively, to interpolate values at those points. On the* **right***, we sketch how to find candidates for the next edge point given that* $\mathbf{q}$ *is an edge point; an appropriate search direction is perpendicular to the gradient, so that points* $\mathbf{s}$ *and* $\mathbf{t}$ *should be considered for the next edge point. Notice that, in principle, we don't need to restrict ourselves to pixel points on the image grid, because we know where the predicted position lies between* $\mathbf{s}$ *and* $\mathbf{t}$*. Hence, we could again interpolate to obtain gradient values for points off the grid.*

8.2.2 Orientations

As the light gets brighter or darker (or as the camera aperture opens or closes), the image will get brighter or darker, which we can represent as a scaling of the image value. The image $\mathcal{I}$ will be replaced with $s\mathcal{I}$ for some value s. The magnitude of the gradient scales with the image, i.e., $\|\nabla\mathcal{I}\|$ will be replaced with $s|\nabla\mathcal{I}\|$. This creates problems for edge detectors, because edge points may appear and disappear as the image gradient values go above and below thresholds with the scaling. One solution is to represent the *orientation* of image gradient, which is unaffected by scaling (Figure 8.8). The gradient orientation field depends on the smoothing scale at which the gradient was computed. Orientation fields can be quite characteristic of particular textures (Figure 8.10), and we will use this important property to come up with more complex features below.

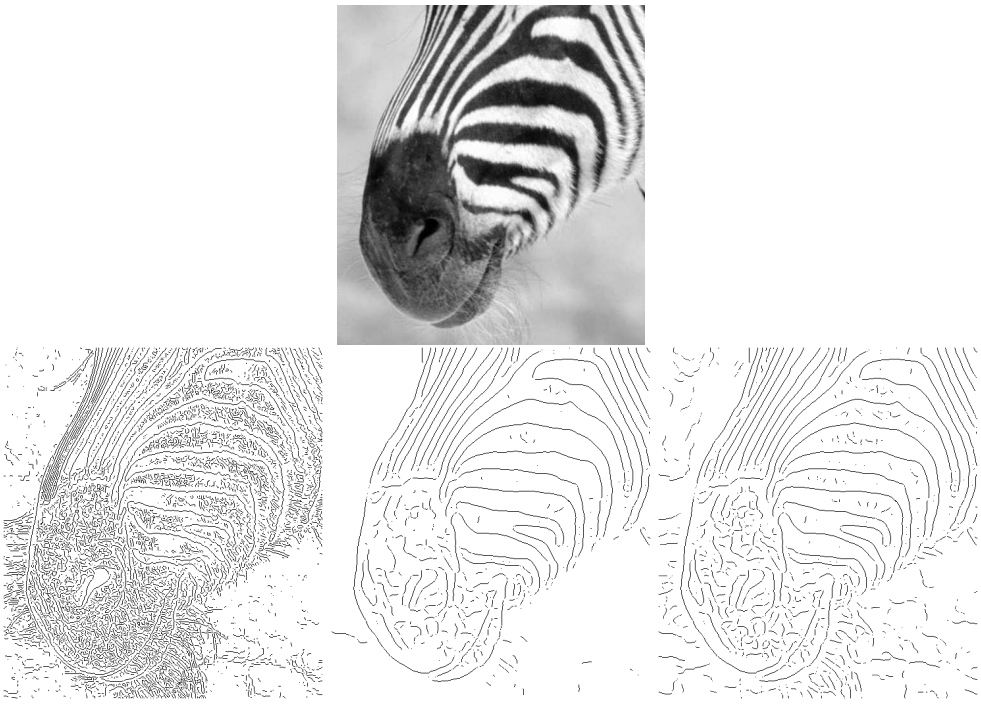

FIGURE 8.7: *Edge points marked on the pixel grid for the image shown on the* **top**. *The edge points on the* **left** *are obtained using a Gaussian smoothing filter at σ one pixel, and gradient magnitude has been tested against a high threshold to determine whether a point is an edge point. The edge points at the* **center** *are obtained using a Gaussian smoothing filter at σ four pixels, and gradient magnitude has been tested against a high threshold to determine whether a point is an edge point. The edge points on the* **right** *are obtained using a Gaussian smoothing filter at σ four pixels, and gradient magnitude has been tested against a low threshold to determine whether a point is an edge point. At a fine scale, fine detail at high contrast generates edge points, which disappear at the coarser scale. When the threshold is high, curves of edge points are often broken because the gradient magnitude dips below the threshold; for the low threshold, a variety of new edge points of dubious significance are introduced.*

8.3 FINDING CORNERS AND BUILDING NEIGHBORHOODS

Points worth matching are corners, because a corner can be *localized*, which means we can tell where a corner is. This motivates the more general term *interest point* often used to describe a corner. In this view, corners are interesting because we can tell where they are. Place a small window over a patch of constant image value. If you translate the window in any direction, the image in the window will not change significantly. This means you cannot give a reliable estimate of the location of the window from its gray levels. Similarly, if you translate a window up and down an edge, the image in the window doesn't change, so you cannot estimate location

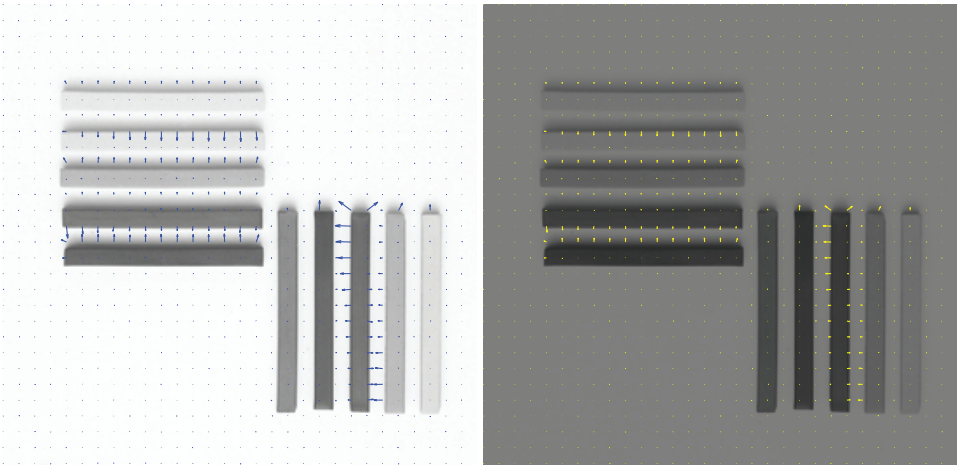

FIGURE 8.8: *The magnitude of the image gradient changes when one increases or decreases the intensity. The orientation of the image gradient does not change; we have plotted every 10th orientation arrow, to make the figure easier to read. Note how the directions of the gradient arrows are fixed, whereas the size changes.* Philip Gatward © Dorling Kindersley, used with permission.

along the edge (this observation used to be known as the *aperture problem*). But with a corner, any movement of the window changes the image in the window (i.e., the patch of image around the corner is not *self-similar*), so you can estimate the location of the corner. Corners are not the only type of local image structure with this property (Section **??**)

There are many ways of representing a neighborhood around an interesting corner. Methods vary depending on what might happen to the neighborhood. In what follows, we will assume that neighborhoods are only translated, rotated, and scaled (rather than, say, subjected to an affine or projective transformation), and so without loss of generality we can assume that the patches are circular. We must estimate the radius of this circle. There is technical machinery available for the neighborhoods that result from more complex transformations, but it is more intricate; see Section **??**.

8.3.1 Finding Corners

One way to find corners is to find edges, and then walk the edges looking for a corner. This approach can work poorly, because edge detectors often fail at corners. At sharp corners or unfortunately oriented corners, gradient estimates are poor because the smoothing region covers the corner.

At a corner, we expect two important effects. First, there should be large gradients. Second, in a small neighborhood, the gradient orientation should swing sharply. We can identify corners by looking at variations in orientation within a

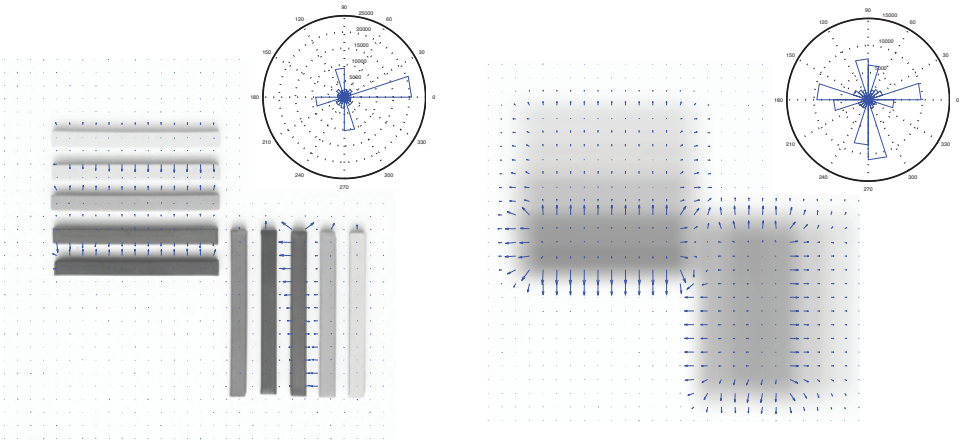

FIGURE 8.9: *The scale at which one takes the gradient affects the orientation field. We show the overall trend of the orientation field by plotting a rose plot, where the size of a wedge represents the relative frequency of that range of orientations.* **Left** *shows an image of artists pastels at a fairly fine scale; here the edges are sharp, and so only a small set of orientations occurs. In the heavily smoothed version on the* **right***, all edges are blurred and corners become smooth and blobby; as a result, more orientations appear in the rose plot.* Philip Gatward © Dorling Kindersley, used with permission.

window. In particular, the matrix

$$\begin{aligned}\mathcal{H} &= \sum_{window} \left\{(\nabla I)(\nabla I)^T\right\} \\ &\approx \sum_{window} \left\{ \begin{array}{ll} (\frac{\partial G_\sigma}{\partial x} * \mathcal{I})(\frac{\partial G_\sigma}{\partial x} * \mathcal{I}) & (\frac{\partial G_\sigma}{\partial x} * \mathcal{I})(\frac{\partial G_\sigma}{\partial y} * \mathcal{I}) \\ (\frac{\partial G_\sigma}{\partial x} * \mathcal{I})(\frac{\partial G_\sigma}{\partial y} * \mathcal{I}) & (\frac{\partial G_\sigma}{\partial y} * \mathcal{I})(\frac{\partial G_\sigma}{\partial y} * \mathcal{I}) \end{array} \right\}\end{aligned}$$

gives a good idea of the behavior of the orientation in a window. In a window of constant gray level, both eigenvalues of this matrix are small because all the terms are small. In an edge window, we expect to see one large eigenvalue associated with gradients at the edge and one small eigenvalue because few gradients run in other directions. But in a corner window, both eigenvalues should be large.

The *Harris corner detector* looks for local maxima of

$$\det(\mathcal{H}) - k(\frac{\text{trace}(\mathcal{H})}{2})^2$$

where k is some constant [**?**]; we used 0.5 for Figure 9.1. These local maxima are then tested against a threshold. This tests whether the product of the eigenvalues (which is $\det(\mathcal{H})$) is larger than the square of the average (which is $(\text{trace}(\mathcal{H})/2)^2$). Large, locally maximal values of this test function imply the eigenvalues are both big, which is what we want. Figure 9.1 illustrates corners found with the Harris detector. This detector is unaffected by translation and rotation (Figure 9.2).

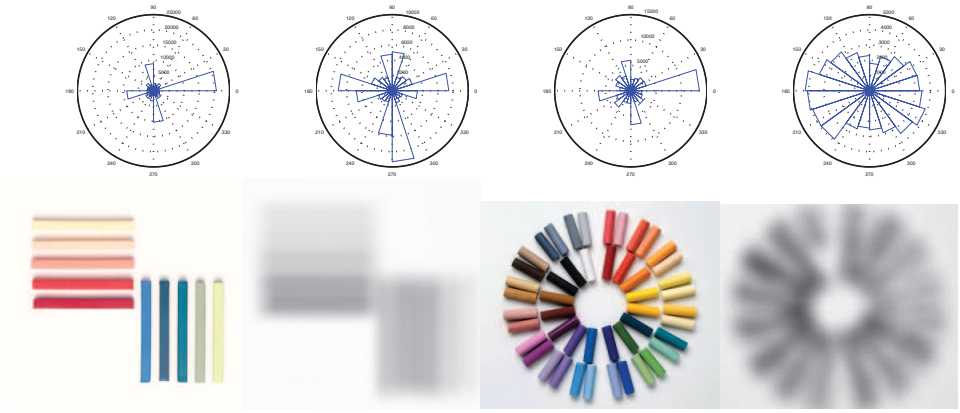

FIGURE 8.10: *Different patterns have quite different orientation histograms. The* **left** *shows rose plots and images for a picture of artists pastels at two different scales; the* **right** *shows rose plots and images for a set of pastels arranged into a circular pattern. Notice how the pattern of orientations at a particular scale, and also the changes across scales, are quite different for these two very different patterns.* Philip Gatward © Dorling Kindersley, used with permission.

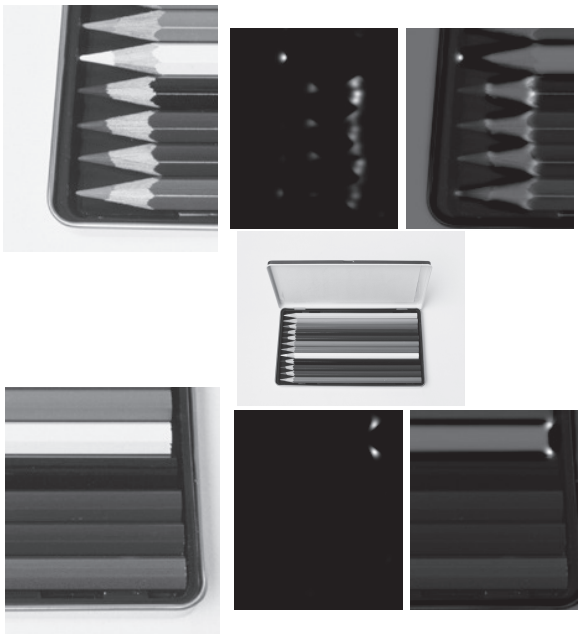

FIGURE 8.11: *The response of the Harris corner detector visualized for two detail regions of an image of a box of colored pencils (***center***). **Top left**, a detail from the pencil points; **top center**, the response of the Harris corner detector, where more positive values are lighter. The **top right** shows these overlaid on the original image. To overlay this map, we added the images, so that areas where the overlap is notably dark come from places where the Harris statistic is negative (which means that one eigenvalue of $\mathcal{H}$ is large, the other small). Note that the detector is affected by contrast, so that, for example, the point of the mid-gray pencil at the top of this figure generates a very strong corner response, but the points of the darker pencils do not, because they have little contrast with the tray. For the darker pencils, the strong, contrasty corners occur where the lead of the pencil meets the wood. The* **bottom** *sequence shows corners for a detail of pencil ends. Notice that responses are quite local, and there are a relatively small number of very strong corners.* Steve Gorton © Dorling Kindersley, used with permission.

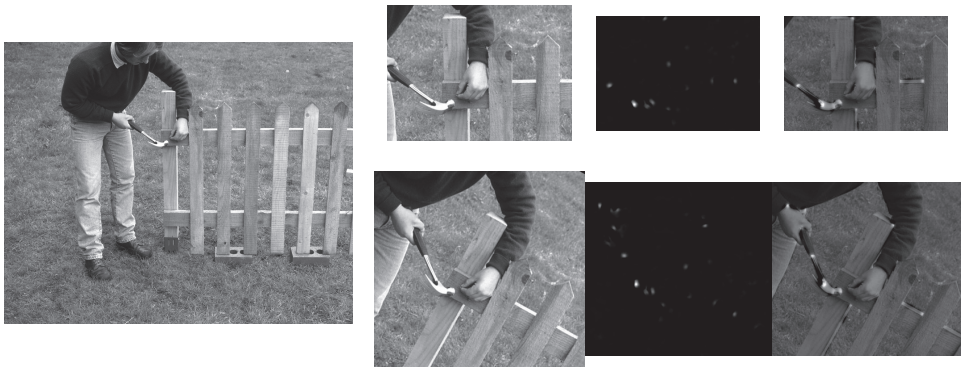

FIGURE 8.12: *The response of the Harris corner detector is unaffected by rotation and translation. The* **top row** *shows the response of the detector on a detail of the image on the* **far left**. *The* **bottom row** *shows the response of the detector on a corresponding detail from a rotated version of the image. For each row, we show the detail window (***left***); the response of the Harris corner detector, where more positive values are lighter (***center***); and the responses overlaid on the image (***right***). Notice that responses are quite local, and there are a relatively small number of very strong corners. To overlay this map, we added the images, so that areas where the overlap is notably dark come from places where the Harris statistic is negative (which means that one eigenvalue of $\mathcal{H}$ is large, the other small). The arm and hammer in the top row match those in the bottom row; notice how well the maps of Harris corner detector responses match, too.*