CHAPTER 8

Edges and Interest Points

Sharp changes in image intensity are evidence that something interesting is happening. Such changes could be caused by a change in the color of the surface; a shadow; a sharp change in the orientation of the surface (which changes how much light it collects, Section 27.2.1); or an *occluding contour* (an image curve which separates object pixels from background pixels). These sharp changes occur at *edges*, curves where the brightness changes particularly sharply (Section 8.1.1). Traditionally, edges were used for inference based methods – a program would observe some interesting structure in the edges, then draw conclusions about the scene from them. Edges aren't much used for inference anymore, but are still useful for other purposes because they represent a form of summary of an image – a representation in terms of edges is small, but contains a lot of information about the image.

A really useful summary describes an image in terms of *interest points*. For the moment, these are points in the image that are easy to find and that support distinctive descriptions (as an example, the corners of texture patterns). They are important because they offer an extremely small representation of the image that is very useful in practice. Interest points can be used to register images – one matches the interest points, then computes a transformation using the matches (Section 15.10). They can be used to infer movement from a sequence of images – one matches interest points, then solves a geometric problem (Section 15.10).

8.1 EDGES AND ORIENTATIONS

Edges are curves of edge points, where the brightness changes particularly sharply. At edges, the gradient has a large magnitude, and the gradient magnitude is larger than at nearby points. Constructing edge points and edges out of them is relatively straightforward (Section 15.10). But edge points are identified by checking gradient magnitude against a threshold. This means that scaling the intensity of the image (perhaps by changing the brightness of the illumination or by changing the camera aperture) will change the edges. An alternative image representation that is unaffected by intensity scaling is the orientation of the image gradient. This is useful in a range of applications.

There is mostly no advantage in looking for edges in RGB images, It is mostly enough to find edges in intensity images, because (somewhat surprisingly) there are very few *isoluminant color edges* – that is, color changes where the intensity does not change. As Figure 8.1 shows, when there is a gradient in any color component, there is almost always a gradient in the same place in the intensity image.

8.1.1 Gradient-Based Edge Detectors

Edges occur where the image intensity changes particularly sharply, so the first step to find edges is to compute an estimate of the gradient magnitude using a derivative

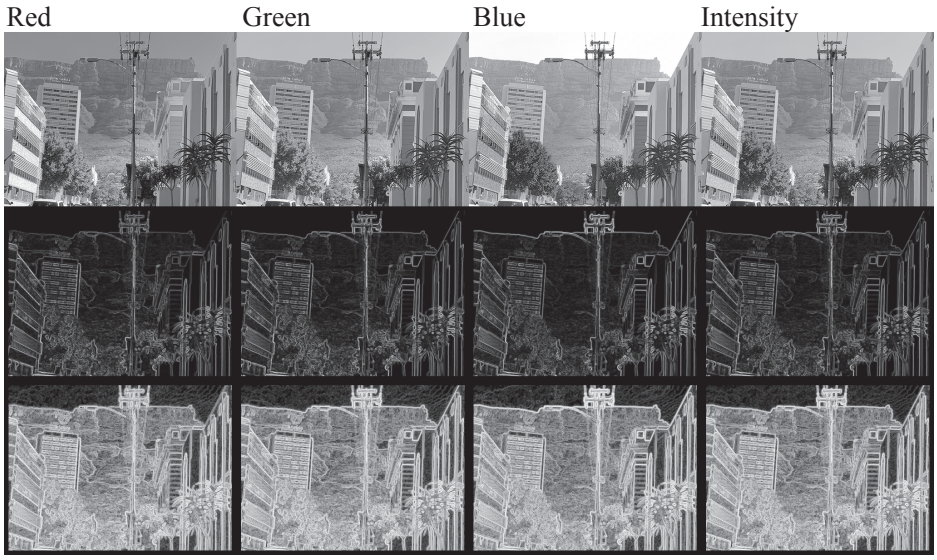

FIGURE 8.1: *Edges almost always involve a change in image intensity, and so image color can be ignored.* **Top row** *shows red, green and blue separations of an image, together with the image intensity. Notice that the intensity of the separations changes (for example, the sky is blue, so it is brighter in the blue separation and darker in the red separation), but neighboring regions that are different in one color separation are different in intensity, too.* **Center row** *shows gradient magnitude, computed with derivative of gaussian filters ($\sigma = 1$). These have been scaled with a single scale so that the largest magnitude is light (same scale applied to each separation and to intensity). While the magnitudes differ from separation to separation, if there is a gradient of any magnitude in one separation, it appears in the intensity image as well.* **Bottom row** *shows log of scaled gradient magnitude, with a scale chosen to show gradients with small magnitude. Again, anything that appears in one separation appears in the intensity image as well. This is just a fact about color images, but it is a useful fact: edges can safely be considered an intensity phenomenon.* Image credit: *Figure shows my photograph of Table Mountain.*

of gaussian filter (Section 5.2.2). The gradient magnitude forms thick trails in the image (Figure 8.2) and edges are curves. These curves are built out of grid points, where the grid for the curves might be finer than the image grid (because gradient magnitude values can be interpolated). Each edge point should should be a local maximum of the gradient in the direction normal to the curve (otherwise one could move the edge point to a location where the change in intensity is larger). This is easily checked for any given point (Figure 8.3). Because edge points join up to form curves, it should be easy to find the next point by searching along the tangent to the curve. The tangent should be normal to the gradient, so the next point on the edge is easy to find (Figure 8.3). Check the available candidates, and if one is

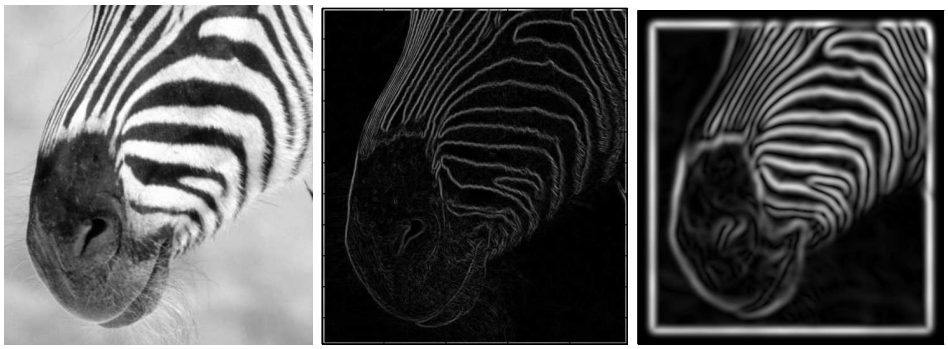

FIGURE 8.2: *The gradient magnitude can be estimated by smoothing an image and then differentiating it. This is equivalent to convolving with the derivative of a smoothing kernel. The extent of the smoothing affects the gradient magnitude; in this figure, we show the gradient magnitude for the figure 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.*

an edge point add it to the chain of points. Once you run out of points on a given chain (the search for the next edge point fails), just look for another edge point. Forming chains of points using this process is called *nonmaximum suppression*.

There are too many chains to come close to being a reasonable representation of object boundaries. In part, this is because the marking procedure does not look at how large the 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*; use 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 the procedure in the box below. Most current edgefinders follow these lines.

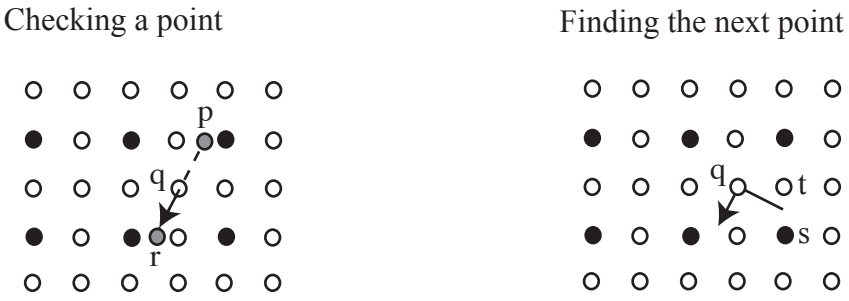

FIGURE 8.3: *The figure on the* **left** *shows how to tell whether a location is an edge point. In this figure, the edge point grid is finer than the image grid by a factor of two. The filled circles are points on the image grid (which are also on the edge point grid), and the open circles are on the edge point grid. Currently,* $\mathbf{q}$ *is under consideration as an edge point. To determine whether the gradient is at a maximum the value of the magnitude must be interpolated at both* $\mathbf{p}$ *and* $\mathbf{r}$*, neither of which are grid points. If the value at* $\mathbf{q}$ *is larger than both,* $\mathbf{q}$ *is an edge point. The figure on the* **right** *shows that, once one has an edge point, the next point in the chain can be found relatively easily. The tangent to the curve of edge points is perpendicular to the gradient, so if* $\mathbf{q}$ *is an edge point,* $\mathbf{s}$ *and* $\mathbf{t}$ *are candidates for the next edge point. Gradient magnitudes that aren't known are reconstructed by interpolation.*

Procedure: 8.1 *Gradient-Based Edge Detection.*

```
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
```

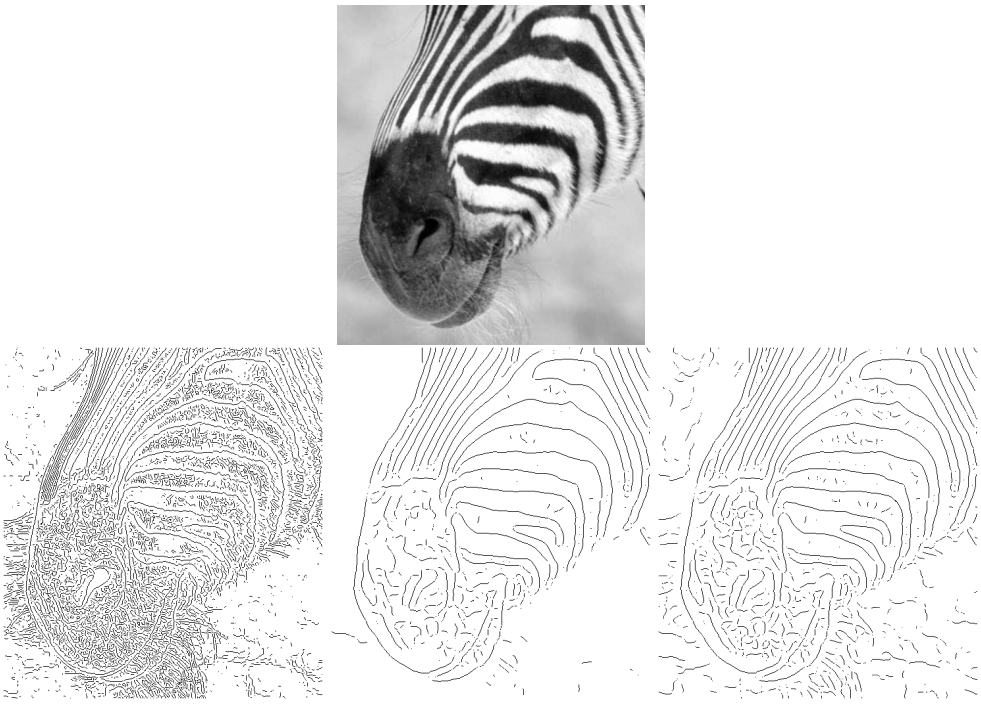

FIGURE 8.4: *Edge points marked on the pixel grid for the image shown on the* **top**. *The edge points on the* **left** *are obtained using a Gaussian smoothing filter at σ one pixel, and gradient magnitude has been tested against a high threshold to determine whether a point is an edge point. The edge 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.*

8.1.2 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 8.5). The gradient orientation field depends on the smoothing scale

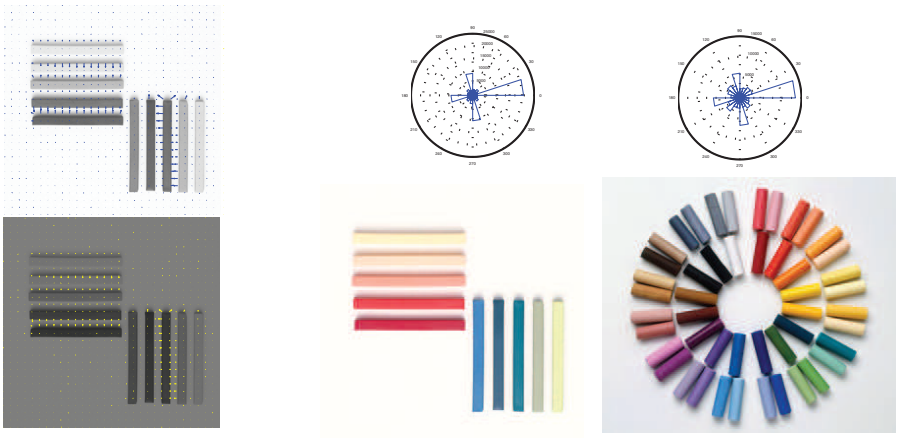

FIGURE 8.5: *Orientations are important because (a) the orientation of the gradient does not change when the image intensity changes and (b) different patterns tend to have different orientations. On the* **left**, *two versions of the same image, at different intensity, with every 10th orientation arrow plotted (otherwise the arrows overlap and the image is hard to read). The directions of the gradient arrows are fixed, whereas the size changes. On the* **right**, *two quite different simple images with rose plots showing orientation patterns. In a rose plot, the size of a wedge represents the relative frequency of that range of orientations. The rose plots are different for these different patterns.* Philip Gatward © Dorling Kindersley, used with permission.

at which the gradient was computed. Orientation fields can be quite characteristic of particular textures (Figure **??**), and we will use this important property to come up with more complex features below.

Remember this: *Edges are image locations where intensity changes strongly. An edge could be evidence of: an object boundary; a change in surface orientation; a change in surface albedo; or a shadow. Edge points are typically points where the gradient magnitude is a local maximum and also large. They can be found by estimating the gradient using derivative of gaussian filters. Edge points can be linked into edge curves. These curves are improved by hysteresis in the threshold.*

8.2 INTEREST POINTS

One strategy for registering an image to another is to find corresponding pairs of points and register those. Correspondence is important – registration requires

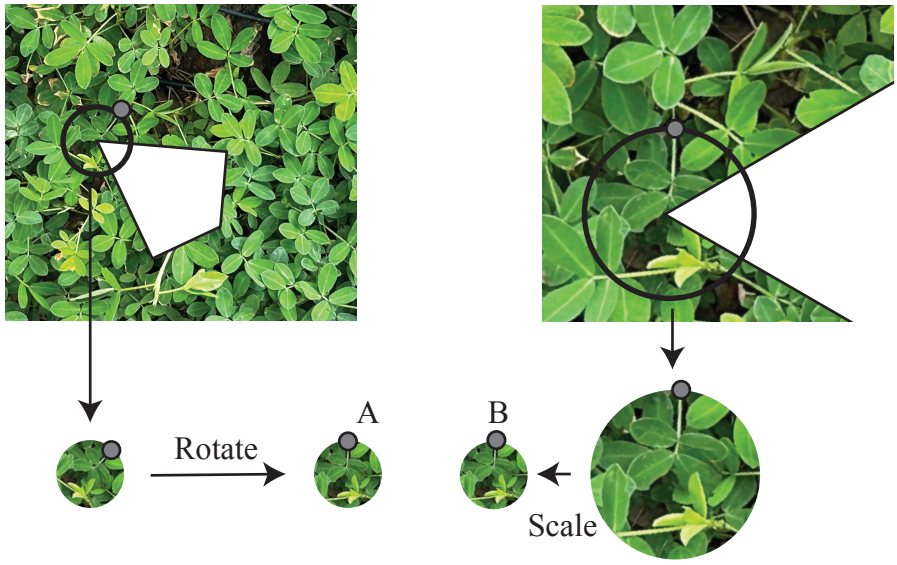

FIGURE 8.6: *Interest points are neighborhoods in the image that can be described in a way that is unaffected by translation, rotation and scale. The* **top row** *shows two images of the same polygon lying on a leaf texture; the camera was rotated and translated between pictures, and has zoomed in on the second image. The corner of the polygon is a promising location to describe, because it can be localized. But to obtain a description that can be compared from image to image requires cutting out a neighborhood whose size changes appropriately when the image is zoomed in or out (the* **dark circle** *on the two images), and knowing how to rotate those neighborhoods into a comparable configuration (in this case, the filled gray circle indicates the zero angle). Such neighborhoods can be compared by, for example, translating, rotating and scaling them to a comparable configuration (***A** *and* **B***) and then comparing a feature description of the contents of the neighborhood.*

knowing which point in image 1 will go to which point in image 2. Appropriate points, together with their neighborhoods, are usually called *interest points*. One way to establish correspondence is to build a neighborhood around a point, describe the contents of the neighborhood in a distinctive way, then match these descriptions. For this to work, the neighborhood should behave appropriately when image 2 is a translated, rotated and scaled version of image 1. Interest points have the following important properties:

Localization: It must be possible to *localize* the point (ie tell where the point is) by looking at an image window around the point. For example, a corner can be localized; but a point along a straight edge can't, because sliding a window around the point along the edge leads to a new window that looks like the original. It must be possible to find them reasonably reliably, even when image brightness changes.

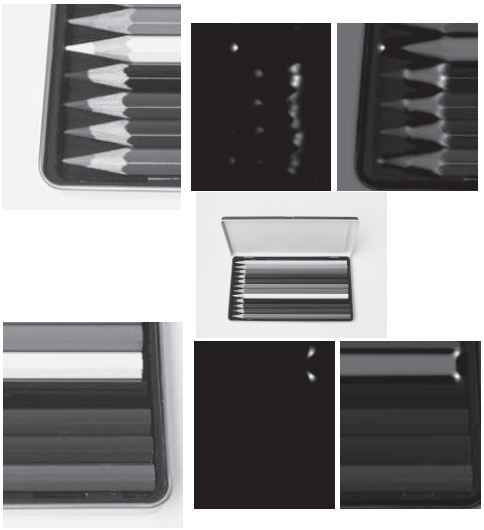

FIGURE 8.7: *The response of the Harris corner detector visualized for two detail regions of an image of a box of colored pencils (***center***).* **Top left***, a detail from the pencil points;* **top center***, the response of the Harris corner detector, where more positive values are lighter. The* **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.

Covariance: The location of the point must be *covariant* under at least some natural image transformations. This means that, if the image is transformed, the point will be found in an appropriate spot in the transformed image. Equivalently, the points "stick to" objects in the image – if the camera moves, the point stays on the object where it was, and so moves in the image. So if, for example, if $\mathcal{I}_2$ is obtained by rotating $\mathcal{I}_1$, then there should be an interest point at each location in $\mathcal{I}_2$ obtained by rotating the position of an interest point in $\mathcal{I}_1$.

Covariant neighborhood: It must be possible to compute a description of the image in the neighborhood of the point, so the point can be matched. Ideally, corresponding points in different images will have similar descriptions, and different points will have different images. Computing this description requires that the neighborhood associated with the interest point is covariant. So, for example, if the image is zoomed in, the neighborhood in the image gets bigger; and if it is zoomed out, the neighborhood gets smaller. Using a fixed size neighborhood when

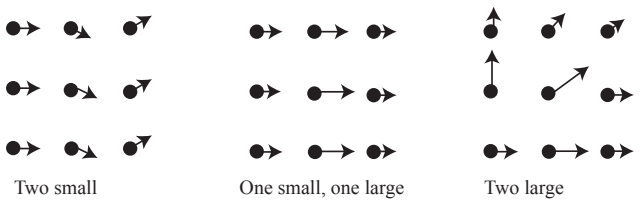

FIGURE 8.8: *The matrix $\mathcal{H}$, defined in the text, behaves rather like a covariance matrix. The figure shows examples of three different cases for 3×3 image windows (the filled dots are the pixel locations). On the* **left***, all gradients are small; verify that, in this case, there will be two small eigenvalues. In the* **center***, there are large gradients, but all point in the same direction; verify, that in this case, there will be one large and one small eigenvalue. On the* **right***, there are large gradients that point in a range of directions, as one would expect at a corner; verify that, in this case, there will be two large eigenvalues.*

the image zooms won't work, because the neighborhood in the zoomed in image will contain patterns that aren't in the neighborhood in the zoomed out image.

Figure 8.6 illustrates these ideas. You can write covariance as:

$$\text{neighborhoods}(\text{transformation}(\mathcal{I})) = \text{transformation}(\text{neighborhoods}(\mathcal{I})).$$

It is usual to require covariance under rotations, translations and scale, and I will focus on that case.

8.2.1 Finding Corners

Interest points are usually constructed at corners, because corners can be localized and are quite easy to find with a straightforward detector. At a corner, there are two important effects. First, there should be large gradients. Second, in a small neighborhood, the gradient orientation should swing sharply. Identify corners by looking at variations in orientation within a window. In particular, the matrix

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

gives a good idea of the behavior of the orientation in a window (Figure 8.8). In a window of constant gray level, both eigenvalues of this matrix are small because all the terms are small (exercises). In an edge window, there will be one large eigenvalue (from gradients at the edge) and one small eigenvalue (few gradients run in other directions; **exercises**). But in a corner window, both eigenvalues should be large (**exercises**).

The *Harris corner detector* looks for local maxima of

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

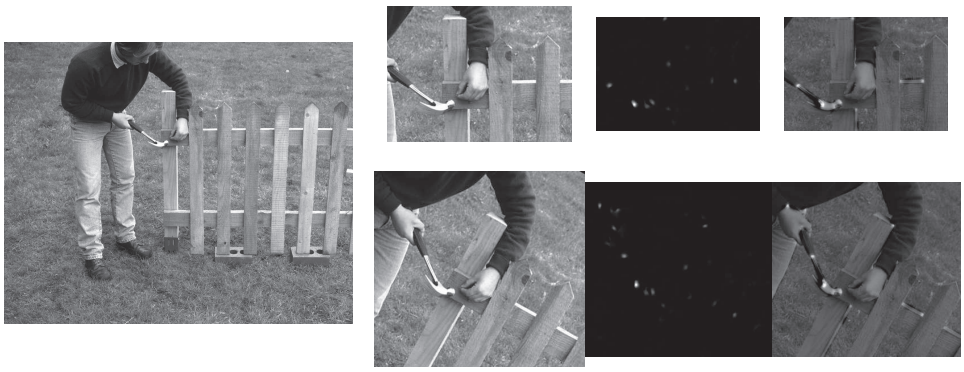

FIGURE 8.9: *The response of the Harris corner detector is unaffected by rotation and translation. The* **top row** *shows the response of the detector on a detail of the image on the* **far left**. *The* **bottom row** *shows the response of the detector on a corresponding detail from a rotated version 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.*

where k is some constant [**?**]; I used 0.5 for Figure 8.7. 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. Figure 8.7 illustrates corners found with the Harris detector. This detector is unaffected by translation and rotation (Figure 8.9).

8.2.2 The Radius of the Neighborhood

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, 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 assume that the patches are circular. Obtaining a neighborhood at a corner involves estimating the radius of the circle centered on the corner. There is technical machinery available for the neighborhoods that result from more complex transformations, but it is more intricate (see []). In most applications, the construction here is sufficient.

To turn a corner into an image neighborhood, estimate the radius of the circular patch (equivalently, its scale). The radius estimate should get larger proportionally when the image gets bigger. For example, in a 2x scaled version of the

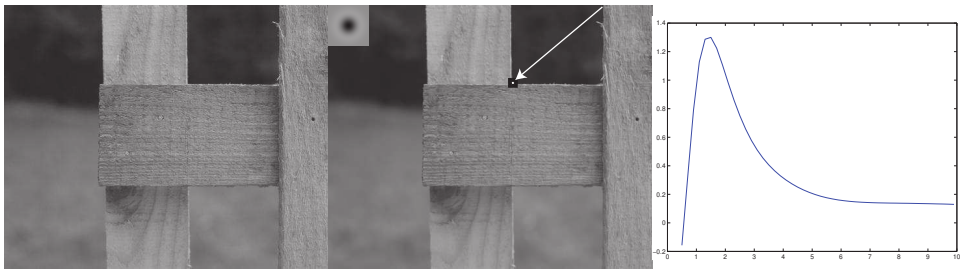

FIGURE 8.10: *The scale of a neighborhood around a corner can be estimated by finding a local extremum, in* scale *of the response at that point to a smoothed Laplacian of Gaussian kernel. On the* **left**, *a detail of a piece of fencing. In the center, a corner identified by an arrow (which points to the corner, given by a white spot surrounded by a black ring).* Overlaid *on this image is a Laplacian of Gaussian kernel, in the* **top right** *corner; dark values are negative, mid gray is zero, and light values are positive. Notice that, using the reasoning of Section* **??**, *this filter will give a strong positive response for a dark blob on a light background, and a strong negative response for a light blob on a dark background, so by searching for the strongest response at this point as a function of scale, we are looking for the size of the best-fitting blob. On the* **right**, *the response of a Laplacian of Gaussian* at the location of the corner*, as a function of the smoothing parameter (which is plotted in pixels). There is one extremal scale, at approximately 2 pixels. This means that there is one scale at which the image neighborhood looks most like a blob (some corners have more than one scale).*

original image, the method should double its estimate of the patch radius. This property helps choose a method. Center a blob of fixed appearance (say, dark on a light background) on the corner, and then choose the scale to be the radius of the best fitting blob. An efficient way to do this is to use a Laplacian of Gaussian filter.

The *Laplacian* of a function in 2D is defined as

$$(\nabla^2 f)(x, y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}.$$

It is natural to smooth the image before applying a Laplacian. Notice that the Laplacian is a linear operator and is shift invariant (exercises), meaning that we could represent taking the Laplacian as convolving the image with some kernel (which we write as K_{∇^2}). Because convolution is associative, we have that

$$(K_{\nabla^2} * (G_\sigma * \mathcal{I})) = (K_{\nabla^2} * G_\sigma) * \mathcal{I} = (\nabla^2 G_\sigma) * \mathcal{I}.$$

The reason this is important is that, just as for first derivatives, smoothing an image and then applying the Laplacian is the same as convolving the image with the Laplacian of the smoothing kernel. Figure 8.10 shows what happens for the usual case where smoothing is Gaussian smoothing. This kernel looks like a dark blob with a light ring around it on a gray background (closed form expression in the exercises). As the scale of the Gaussian gets larger, the blob gets bigger.

This means the Laplacian of Gaussian can be used to find the radius of the neighborhood around a given corner located at i, j in $\mathcal{I}$. Write $\mathcal{L}_\sigma$ for a Laplacian of Gaussian kernel where the Gaussian has scale σ. Now, for each σ, place the kernel on the image, centered at the corner, and compute

$$V(\sigma) = \sum_{uv} \mathcal{I}_{i-u,j-v} \mathcal{L}_{\sigma;uv}.$$

The σ that maximizes (or minimizes) $V(\sigma)$ is the radius to use. This value has strong covariance properties. Imagine image $\mathcal{S}$ is the same as $\mathcal{I}$, but is upsampled factor of k. Then the value of σ chosen by this procedure for $\mathcal{S}$ will be k times the value chosen for $\mathcal{I}$. This is because the value is, essentially a dot-product (as in Section 15.10), and scores the match between the image and the kernel. If the image is upsampled by k, the best matching kernel should just k times the scale. This argument works for downsampling as well as upsampling, and is fine as long as there is no serious loss of information in the upsampling or downsampling (which is most of the time).

Searching for the right σ efficiently requires some care. If σ is large, then the kernel will be large. It is better to apply a small kernel to a downsized image than a large kernel to the original image, so construct a gaussian pyramid from the image. A very large σ (say, half the image size) is implausible, so the pyramid does not need too many layers. If there are enough interest points, it is more efficient to convolve the image with the kernel then look at the values at relevant locations than it is to apply the kernel at each interest point. These considerations lead to:

- Choose some values of σ to search, say $\sigma \in \{\sigma_1, \ldots, \sigma_n\}$. Construct Laplacian of gaussian filters for these values.
- Construct a gaussian pyramid, downsampling by d for each layer.
- Convolve each layer of the pyramid with each of the filters.
- As a result, you know the value of the Laplacian of gaussian at each interest point for $\sigma \in \{\sigma_1, \ldots, \sigma_n, d\sigma_1, \ldots, d\sigma_n, d^2\sigma_1, \ldots\}$. Interpolate these values and choose the smallest σ that achieves a maximum or minimum of the interpolate.

8.2.3 The Rotation of the Neighborhood

Orientation histograms are a natural representation of image patches. However, you cannot represent orientations in image coordinates (for example, using the angle to the horizontal image axis), because the patch you are matching to might have been rotated. You need a reference orientation so all angles can be measured with respect to that reference (this is the filled gray circle in Figure 8.6). A natural reference orientation is the most common orientation in the patch. Compute a histogram of the gradient orientations in this patch, and find the largest peak. This peak is the reference orientation for the patch. If there are two or more peaks of the same magnitude, make multiple copies of the patch, one at each peak orientation.

8.2.4 Describing Neighborhoods with Orientations

The final step is describing the image within the neighborhood. There are some important constraints on this representation:

- The representation should be detailed and distinctive, so that points that don't match have different representations.
- The representation should not change if the image gets somewhat brighter or darker.
- The representation should be robust to small errors in the estimates of the location, scale and orientation of the neighborhood.

The first two requirements motivate using orientations, because complex textures will have complicated – and so distinctive – orientation fields, and because orientation fields don't change when the brightness changes. Very dark or very bright neighborhoods probably can't be described accurately. For dark neighborhoods, quantization of the pixel values creates problems; for bright neighborhoods, some pixel values will have saturated. This motivates some form of thresholding on the gradient values. Finally, robustness to small errors suggests using some form of histogram.

Just constructing a histogram of the orientations is not enough, because it confuses too many patterns with one another (for example, a narrow vertical dark stripe will be confused with a broad vertical dark stripe). An alternative is to break the neighborhood into overlapping bins and compute histograms within the bins.

These considerations underly the very important *SIFT descriptor* (for Scale Invariant Feature Transform) is constructed out of image gradients, and uses both magnitude and orientation. The descriptor is a set of histograms of image gradients that are then normalized. The descriptor is normalized to suppress the effects of change in illumination intensity. The histograms expose general spatial trends in the image gradients in the patch but suppress detail. For example, if the interest point finder makes small errors estimating the center, scale, or orientation of the patch, then the rectified patch will shift slightly. As a result, simply recording the gradient at each point yields a representation that changes between instances of the patch. A histogram of gradients will be robust to these changes. Rather than histogramming the gradient at a set of sample points, histogram local averages of image gradients; this helps avoid noise.

The standard SIFT descriptor is obtained by first dividing the rectified patch into an $n \times n$ grid. Then subdivide each grid element into an $m \times m$ subgrid of subcells. At the center of each subcell, compute a gradient estimate. The gradient estimate is obtained as a weighted average of gradients around the center of the cell, weighting each by $(1 - d_x/s_x)(1 - d_y/s_y)/N$, where d_x (resp. d_y) is the x (resp. y) distance from the gradient to the center of the subcell, and s_x (resp. s_y) is the x (resp. y) spacing between the subcell centers. This means that gradients make contributions to more than one subcell, so that a small error in the location of the center of the patch leads to a small change in the descriptor.

Now use these gradient estimates to produce histograms. Each grid element has a q-cell orientation histogram. The magnitude of each gradient estimate is

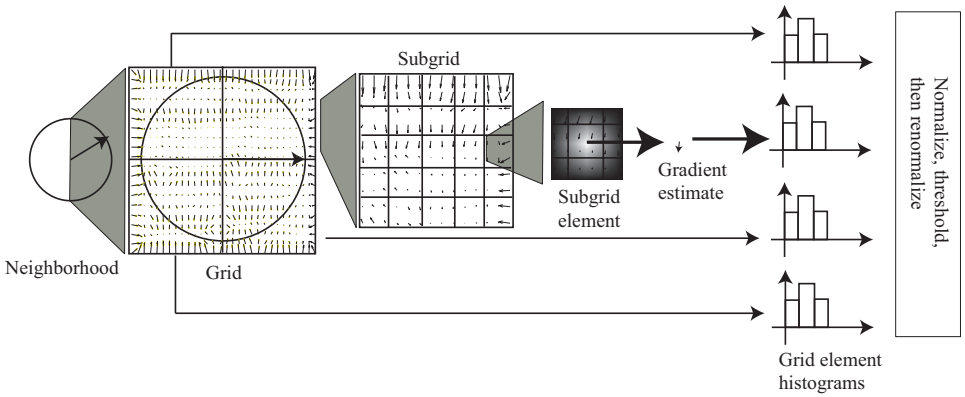

FIGURE 8.11: *To construct a SIFT descriptor for a neighborhood, place a grid over the rectified neighborhood. Each grid is divided into a subgrid, and a gradient estimate is computed at the center of each subgrid element. This gradient estimate is a weighted average of nearby gradients, with weights chosen so that gradients outside the subgrid cell contribute. The gradient estimates in each subgrid element are accumulated into an orientation histogram. Each gradient votes for its orientation, with a vote weighted by its magnitude and by its distance to the center of the neighborhood. The resulting orientation histograms are stacked to give a single feature vector. This is normalized to have unit norm; then terms in the normalized feature vector are thresholded, and the vector is normalized again.*
TODO: Source, Credit, Permission: SIFTPIC

accumulated into the histogram cell corresponding to its orientation; the magnitude is weighted by a Gaussian in distance from the center of the patch, using a standard deviation of half the patch.

Concatenate each histogram into a vector of $n \times n \times q$ entries. If the image intensity were doubled, this vector's length would double (because the histogram entries are sums of gradient magnitudes). To avoid this effect, normalize this vector to have unit length. Very large gradient magnitude estimates tend to be unstable (for example, they might result from a lucky arrangement of surfaces in 3D so that one faces the light directly and another points away from the light). This means that large entries in the normalized vector are untrustworthy. To avoid difficulties with large gradient magnitudes, each value in the normalized vector is thresholded with threshold t, and the resulting vector is renormalized. Standard parameter values are $n = 4$, $m = 4$, $q = 8$, and $t = 0.2$. SIFT features are patented (US6711293B1), but the patent has expired. David Lowe (the inventor) provides a reference object code implementation at `http://www.cs.ubc.ca/~lowe/keypoints/`. There is also an implementation in OpenCV (currently at `https://docs.opencv.org/4.x/da/df5/tutorial_py_sift_intro.html`). There is now extensive experimental evidence that image patches that match one another will have similar SIFT feature representations, and patches that do not will tend not to.