Edge Detection

Shree K. Nayar

Monograph: FPCV-2-1

Module: Features

Series: First Principles of Computer Vision

Computer Science, Columbia University

May 15, 2022

FPCV Channel

FPCV Website

Edge Detection

Shree K. Nayar
Columbia University

Topic: Edge Detection, Module: Features
First Principles of Computer Vision

1

Edge Detection

Convert a 2D image into a set of points where image intensity changes rapidly.

Topics:

(1) What is an Edge?

(2) Edge Detection Using Gradients

(3) Edge Detection Using Laplacian

(4) Canny Edge Detector

(5) Corner Detection

2

In this lecture, we will discuss the detection of edges in an image. From the perspective of information theory, edges are critical to computer vision. An edge can be loosely defined as any location where there is a rapid change in image intensity along one direction.

We will begin by looking at the physical phenomena that give rise to edges, the attributes of an edge that we would like to compute, and the performance criteria that a good edge detector should satisfy. Next, we will develop a theoretical framework for edge detection. We will construct an edge detector that is based on the gradient operator, which uses the first derivatives of an image. Then, we will look at edge detection using the Laplacian operator, which uses the second derivatives of the image. We will examine the advantages and disadvantages of the gradient operator and Laplacian operator. That will lead us to the widely used Canny edge detector, which uses the best attributes of both of these detectors to create a reliable and powerful edge detector.

Finally, we will look at the problem of corner detection. A corner is defined as any location where two edges meet at an angle. We want to be able to find the precise location of the corner without needing to know the intensity values on either side of the corner or the angle at which the edges meet. We will describe the Harris corner detector, which is widely used.

What is an Edge?

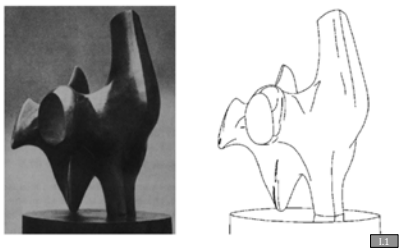
Shree K. Nayar
Columbia University

Topic: Edge Detection, Module: Features
First Principles of Computer Vision

3

What is an Edge?

Rapid change in image intensity within small region

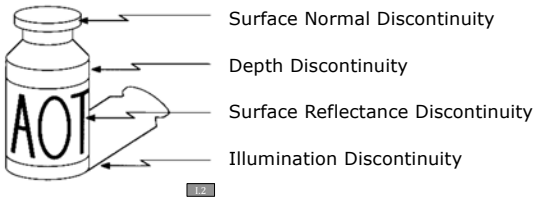


4

Loosely speaking, we can define an edge as a rapid change in image intensity within a small window in an image. Let us take a look at why edges are crucial in computer vision. Here is an example from Vic Nalwa's book. On the left is a photograph of a sculpture by Henry Moore, and on the right is an artist's sketch of the sculpture. With just a few strokes, the artist is able to convey essential information about the sculpture — its three-dimensional structure, its shading, its highlights, and so on. From this example, we can see that edges, even when sparse in an image, are capable of conveying vital visual information.

Causes of Edges

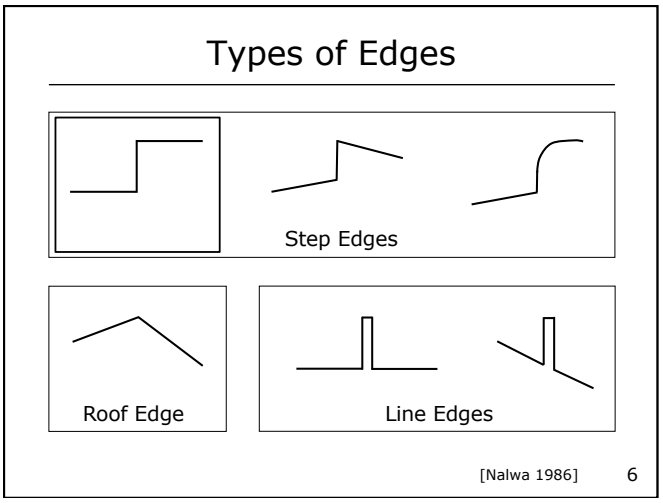
Rapid changes in image intensity are caused by various physical phenomena.



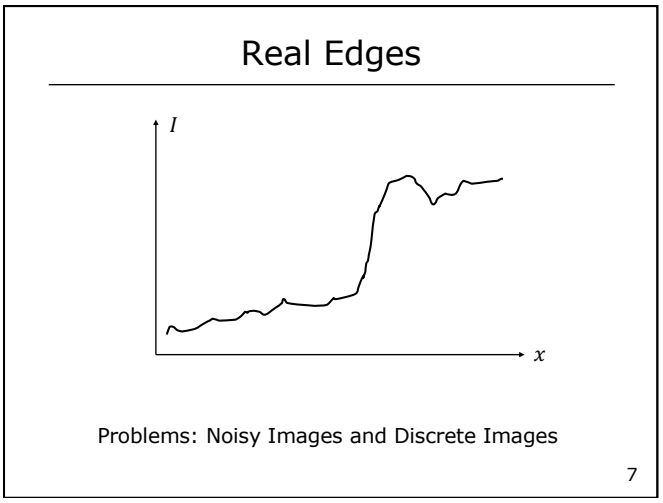
5

Now let us take a look at some of the physical phenomena in the real world that cause edges in images. If one object is located in front of another, there will likely be a sudden change in intensity along the boundary between the two objects. We refer to these as edges that result from a discontinuity in depth. Even if two surfaces are made of the same material, if they have different surface orientations where they meet, they will likely receive different amounts of light from the light sources in the scene, and hence have different brightness values. We refer to these edges as arising from a discontinuity in surface normal. Additionally, if the surface is marked, such as the letters on the bottle here, it will have surface reflectance (material) discontinuities, that again result in edges. Finally, we can have shadows, or illumination discontinuities. Here, the bottle casts a sharp shadow on the background, resulting in a significant difference in the amount of light falling within and outside the shadow. This, again, gives rise to edges.

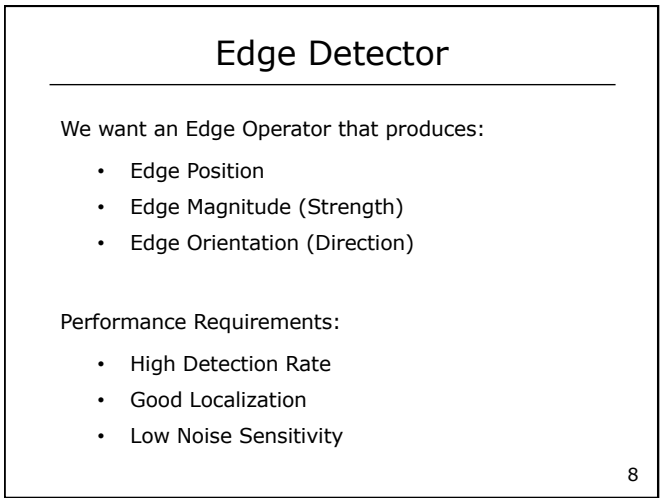
In an image, edges can manifest with many different profiles, a few of which are shown here. They include the simple step function, a step edge with a gradient on either side, a linear gradient on one side with a non-linear gradient on the other side, and a roof edge. When you consider the profile of a line, it can be viewed as being composed of a rising and a falling edge. For our purposes here, we need to choose a single model for the edge that we can use to develop a theory of edge detection. To this end, we will take the simplest of these models —the step function.



It would be nice if edges appeared in images like the very clean step function shown above. That would make the problem of edge detection a lot easier. Unfortunately for us, they tend to look like the function shown here. This is because the edge profile itself can deviate from the step function, and the image has noise due to the various factors we discussed in the lecture on image sensing.



Our edge detector should be able to find the position of an edge. Note that the exact location of an edge can lie within a pixel. We therefore would like to find edges with sub-pixel accuracy. We also want to measure the magnitude (strength) of the edge, so that we can decide whether it is worthy of attention. It is often useful to know the orientation of the edge as well.



We would like our detector to perform well in a few different respects. First, it should have a high detection rate — we want to not only detect all the edges, but also not detect edges where they do not exist. We also want it to have good localization, which means it is able to find an edge as close as possible to where it actually occurs. Finally, we want the detector to be resilient to noise.

Edge Detection Using Gradients

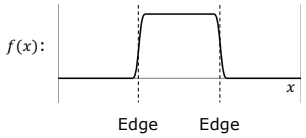
Shree K. Nayar
Columbia University

Topic: Edge Detection, Module: Features
First Principles of Computer Vision

9

1D Edge Detection

Edge is a rapid change in image intensity in a small region.



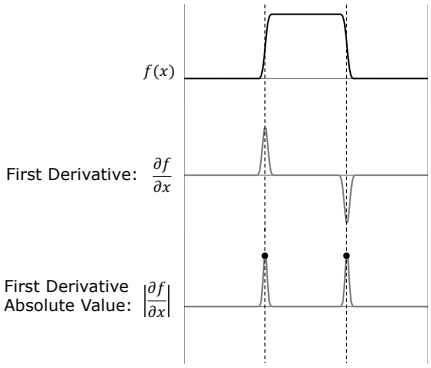
Basic Calculus: Derivative of a continuous function represents the amount of change in the function.

10

The first edge detector we will discuss, the gradient operator, is based on the first derivatives of the image. Let us start with a simple example of a 1D signal, $f(x)$. We know that an edge is a rapid change in image intensity within a small region. Shown here is a rising edge on one side and a falling edge on the other. Calculus tells us that the derivative of a continuous function represents the amount of change in the function, and change is what we are trying to measure. Thus, it makes sense for us to find the first derivative of this function and see what that does to both the edges.

Here we see the first derivative of f with respect to x. It has a maximum at the point where the rising edge is located. For the falling edge, we have exactly the same output, but it is flipped. Thus, the local extrema in the first derivative tell you where the edges are located. If we take the absolute value of the first derivative, the locations of the peaks (maxima) correspond to the locations of the edges, while the magnitudes of the peaks reveal the strengths of the edges.

Edge Detection Using 1st Derivative

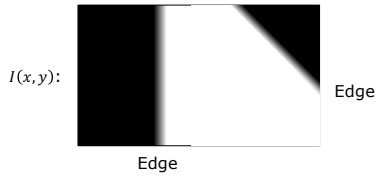


Provides Both Location and Strength of an Edge

11

How do we apply this idea of using the first derivative to two-dimensional images? We know from calculus that the partial derivatives of a 2D continuous function with respect to its two dimensions, x and y, represent the amount of change along each of the two dimensions.

2D Edge Detection



Basic Calculus: Partial Derivatives of a 2D continuous function represents the amount of change along each dimension.

12

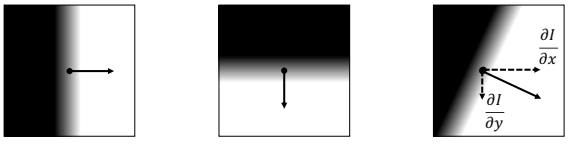
That brings us to the gradient operator, also called the "del" operator, which produces two numbers as a vector. The first is the derivative of the image with respect to x, while the second is the derivative of the image with respect to y. These two numbers tell us everything we need to know about the local change in intensity. Let us take a look at a few examples. On the left we see a vertical edge. In this case, we get a non-zero value for the x component and zero for the y component as there is no change along the y direction. For the second image, we get zero for the x component and a non-zero value for the y component. Finally, for the third image of a tilted edge, we get non-zero values for both components. From just these two numbers at each pixel, we can find both the strength (magnitude) of the edge, as well as its orientation (direction).

Gradient (∇)

Gradient (Partial Derivatives) represents the direction of most rapid change in intensity

$$\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$$

Pronounced as "Del I"



$\nabla I = \left[\frac{\partial I}{\partial x}, 0\right]$ $\nabla I = \left[0, \frac{\partial I}{\partial y}\right]$ $\nabla I = \left[\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right]$

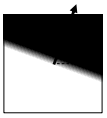
13

The equations for magnitude and orientation are shown here. The magnitude of an edge is simply the square root of the sum of the squares of the partial derivatives. The orientation the edge is the inverse tangent of the first derivative with respect to y divided by the first derivative with respect to x.

Gradient (∇) as Edge Detector

Gradient Magnitude $S = \|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$

Gradient Orientation $\theta = \tan^{-1}\left(\frac{\partial I}{\partial y} / \frac{\partial I}{\partial x}\right)$



14

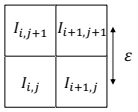
All of the above was done in continuous domain. Now let us see how we might apply our edge detector to a discrete image. In the discrete case, we know that derivatives are implemented using finite differences. Shown here are the finite difference approximations for the partial derivatives with respect to x and y. To find a difference, we need at least two neighboring pixels. So, to find finite differences in both directions (x and y) we need at least two pixels in each direction. Thus, we use a window of 2x2 = 4 pixels. Let us assume that the physical distance between adjacent pixels is ε. Then, the derivative in the x direction is the equation on the top, and we have similar equation for the y component.

Discrete Gradient (∇) Operator

Finite difference approximations:

$$\frac{\partial I}{\partial x} \approx \frac{1}{2\varepsilon}\left(\left(I_{i+1,j+1} - I_{i,j+1}\right) + \left(I_{i+1,j} - I_{i,j}\right)\right)$$

$$\frac{\partial I}{\partial y} \approx \frac{1}{2\varepsilon}\left(\left(I_{i+1,j+1} - I_{i+1,j}\right) + \left(I_{i,j+1} - I_{i,j}\right)\right)$$



Can be implemented as Convolution!

$$\frac{\partial}{\partial x} \approx \frac{1}{2\varepsilon}\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \qquad \frac{\partial}{\partial y} \approx \frac{1}{2\varepsilon}\begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

Note: Convolution flips have been applied

15

We can compute the two derivatives using convolutions with the two kernels (filters) shown below. Here, the "flips" that are needed for each convolution have already been applied. Even if we do not know the physical distance, ε, between adjacent pixels, it does not matter — the resulting derivatives will simply be scaled by an unknown factor that is the same for both components and for all pixels in the image. Once we have computed the two components of the gradient, we use them to find the magnitude and the orientation of the edge as previously discussed. If ε is unknown, the computed magnitude will simply be a scaled version of the true value.

Based on this approach, a variety of gradient operators have been proposed over the last few decades, starting with the Roberts operator shown on the left. In this case, note that the derivatives are being computed in the two diagonal directions. This is not a problem as the derivatives in any two orthogonal directions can be used to find the magnitude and direction of an edge. The Prewitt operator appeared around 1965. The Sobel operator was popular for a long time. We can see that these operators are getting larger as we move from left to right. When we have a small operator, we can expect very good localization for the edges, as small operators will not be affected by image content that is far away from the pixel of interest. On the other hand, a small detector is bound to be extremely sensitive to noise. In the case of the Roberts detector, since only four numbers are being used to determine whether a point has an edge, if noise corrupts even one of these four numbers, the computed derivatives will have large errors.

Comparing Gradient (∇) Operators

Gradient	Roberts	Prewitt	Sobel (3x3)	Sobel (5x5)
$\frac{\partial I}{\partial x}$	0 1 -1 0	-1 0 1 -1 0 1 -1 0 1	-1 0 1 -2 0 2 -1 0 1	-1 -2 0 2 1 -2 -3 0 3 2 -3 -5 0 5 3 -2 -3 0 3 2 -1 -2 0 2 1
$\frac{\partial I}{\partial y}$	1 0 0 -1	1 1 1 0 0 0 -1 -1 -1	1 2 1 0 0 0 -1 -2 -1	1 2 3 2 1 2 3 5 3 2 0 0 0 0 0 -2 -3 -5 -3 -2 -1 -2 -3 -2 -1

Good Localization
Noise Sensitive
Poor Detection

Poor Localization
Less Noise Sensitive
Good Detection

16

Looking at the larger operators, we can see that these can be viewed as just the smaller operators convolved with a smoothing function like a Gaussian. In effect, smoothing is being incorporated into the derivative operators to reduce the effect of noise. Therefore, the larger operators have much better noise resilience. However, they have poor localization, because if we are trying to find an edge at a pixel, the derivatives computed are going to be influenced by image content that is far from the pixel. This is not a problem if the edge continues to be an edge over the entire extent of the operator, but if the edge turns into something else, or new information appears, the output of the operator will be affected.

Let us take a look at an example. Here, we have the gradient operator, specifically the 3x3 Sobel operator, applied to an image that is popularly known as the Lena image. On the left is the first derivative with respect to x, in the middle is the first derivative with respect to y, and on the right is the gradient magnitude computed from both. Now, there is one last step: we have the magnitude and the orientation at every pixel, but we still need to declare each pixel as being an edge or not.

Gradient (∇) Using Sobel Filter

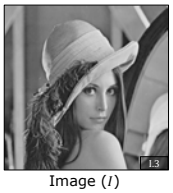


Image (I)

$\partial I/\partial x$

$\partial I/\partial y$

Gradient Magnitude

17

In other words, we need to threshold the edge map. There are essentially two simple ways to do this. The first is to use a single threshold — if the magnitude is less than some value T it is not an edge, and if the magnitude is greater than or equal to T, then it is an edge. We can do a bit better than this by introducing some hysteresis in the thresholding, by using two thresholds. We declare no edge if the magnitude is less that the lower threshold, and an edge if it is greater than the higher threshold. If the magnitude lies between the two thresholds, we can make a decision based on the how "edgy" the neighboring pixels are.

Edge Thresholding

Standard: (Single Threshold T)

$\|\nabla I(x,y)\| < T$	Definitely Not an Edge
$\|\nabla I(x,y)\| \geq T$	Definitely an Edge

Hysteresis Based: (Two Thresholds $T_0 < T_1$)

$\|\nabla I(x,y)\| < T_0$	Definitely Not an Edge
$\|\nabla I(x,y)\| \geq T_1$	Definitely an Edge
$T_0 \leq \|\nabla I(x,y)\| < T_1$	Is an Edge if a Neighboring Pixel is Definitely an Edge

18

In the bottom right is the final thresholded edge map. It is still a scattered set of edges. In the next lecture, we will develop methods to go from this kind of an edge map to clean boundaries. At first glance, that may seem to be a simple step. However, there is something a bit deceptive here which is worth pointing out. When we look at this edge map, we can immediately group edges into curves. That is because our brain is doing the work for us. For a computer, as we shall see, this task can be challenging.

Sobel Edge Detector

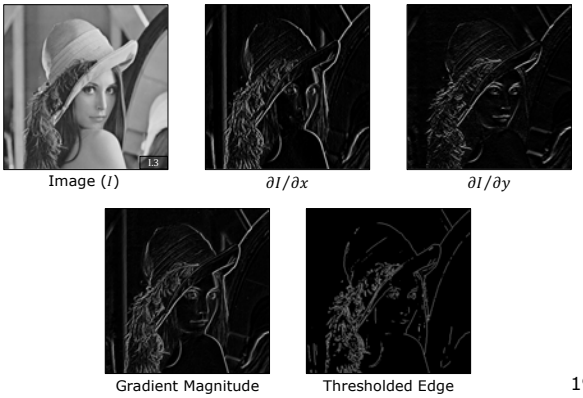


Image (I) $\partial I/\partial x$ $\partial I/\partial y$

Gradient Magnitude Thresholded Edge

19

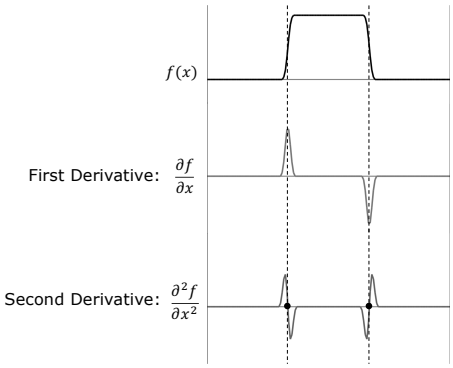
Edge Detection Using Laplacian

Shree K. Nayar
Columbia University

Topic: Edge Detection, Module: Features
First Principles of Computer Vision

20

Edge Detection Using 2nd Derivative



21

Now let's look at how we can use the second derivative of an image to detect edges. That brings us to the Laplacian operator. Once again, we will start with our 1D signal, $f(x)$, and its first derivative, which we already know. The question is, what is the derivative of the derivative, which would be the second derivative? Let us consider the rising edge shown here. We know that where the first derivative is increasing, the second derivative will have positive values, and where the first derivative is decreasing, it will have negative values. In between, when the first derivative reaches a peak, the second derivative will be zero. Therefore, exactly at the location of the edge, we get a sharp change from positive to negative values, called a zero-crossing. In the case of the falling edge on the right, we get exactly the flipped version of the second derivative of the rising edge. Therefore, edge detection boils down to finding sharp zero-crossings in the second derivative.

How do we exploit this property of the second derivation in the context of 2D images? That brings us to the Laplacian operator, also called the "del-squared" operator. The Laplacian is simply the sum of the second derivative of the image with respect to x and the second derivative of the image with respect to y. When we apply the Laplacian operator to an image, we are going to end up with zero-crossings where the edges lie. Note that the Laplacian operator does not provide the direction, or the orientation, of the edge.

Laplacian (∇^2) as Edge Detector

Laplacian: Sum of Pure Second Derivatives

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

Pronounced as "Del Square I"

- Edges are "zero-crossings" in Laplacian of image
- Laplacian does not provide directions of edges

[Marr 1980] 22

How do we apply the Laplacian operator to a discrete image? In terms of finite differences, the second derivative is the difference of the difference. If we want to find this, we need at least three pixels along each of the two dimensions, x and y. So, we use a 3x3 window like the one shown in the top right of the slide. Let us say we want to find the output of the Laplacian operator for the center pixel (i, j). The equations for the x and y components of the second derivative are shown here. To obtain the Laplacian, we simply add these two expressions. This implies that to obtain the Laplacian at all image pixels we only need to convolve the image with the single mask shown in the bottom left of the slide. Note that the corner pixels of the mask are 0 because we never use these pixels in computing the Laplacian.

Discrete Laplacian(∇^2) Operator

Finite difference approximations:

$$\frac{\partial^2 I}{\partial x^2} \approx \frac{1}{\varepsilon^2}\left(I_{i-1,j} - 2I_{i,j} + I_{i+1,j}\right)$$

$$\frac{\partial^2 I}{\partial y^2} \approx \frac{1}{\varepsilon^2}\left(I_{i,j-1} - 2I_{i,j} + I_{i,j+1}\right)$$

$$\nabla^2 I = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2}$$

$I_{i-1,j+1}$	$I_{i,j+1}$	$I_{i+1,j+1}$
$I_{i-1,j}$	$I_{i,j}$	$I_{i+1,j}$
$I_{i-1,j-1}$	$I_{i,j-1}$	$I_{i+1,j-1}$

ε

Convolution Mask:

$$\nabla^2 \approx \frac{1}{\varepsilon^2}\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad \text{OR} \quad \nabla^2 \approx \frac{1}{6\varepsilon^2}\begin{bmatrix} 1 & 4 & 1 \\ 4 & -20 & 4 \\ 1 & 4 & 1 \end{bmatrix}$$ (More Accurate)

23

This operator works fine. However, there is a slight problem that arises from the fact that pixels on an image sensor lie on a square grid. We have found the second derivatives with respect to x and y, assuming that the distance between adjacent pixels is ε. Now, let us say an edge appears at 45 degrees. In that case, we see that we have not taken into account the fact that the distance between pixels in the diagonal direction is greater than ε. In order to account for this variability, we can consider edges in a few difference orientations, develop convolution masks for each one, and then average the masks to end up with the one shown on the right. The output of this operator is less sensitive to the orientation of the edge and hence it is widely used.

Let us take a look at how the above operator can be used to find edges. Here is the Lena image on the left and the output of the Laplacian operator in the middle. Since an image cannot be displayed with negative values, we use the level 128 for 0, such that pixels darker than 128 are negative and pixels brighter than 128 are positive, in terms of the output of the Laplacian. Wherever there is a zero-crossing, the pixel value will be exactly 128, with large positive and negative values around it. These zero-crossings have been detected to produce the final edge map shown on the right.

Laplacian Edge Detector

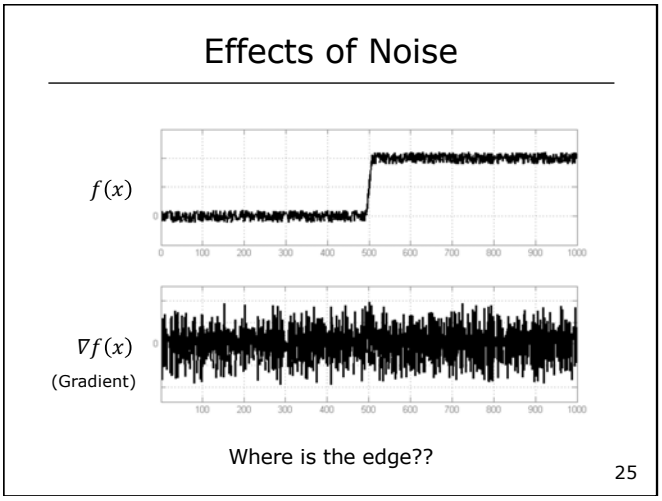
Image (I)

Laplacian
(0 maps to 128)

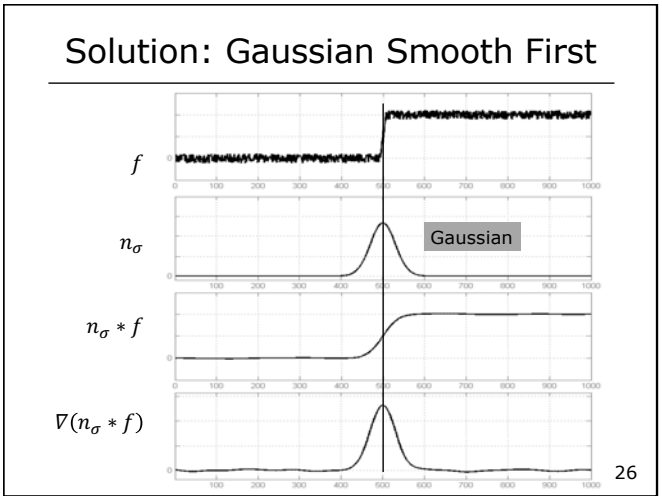
Laplacian
"Zero Crossings"

24

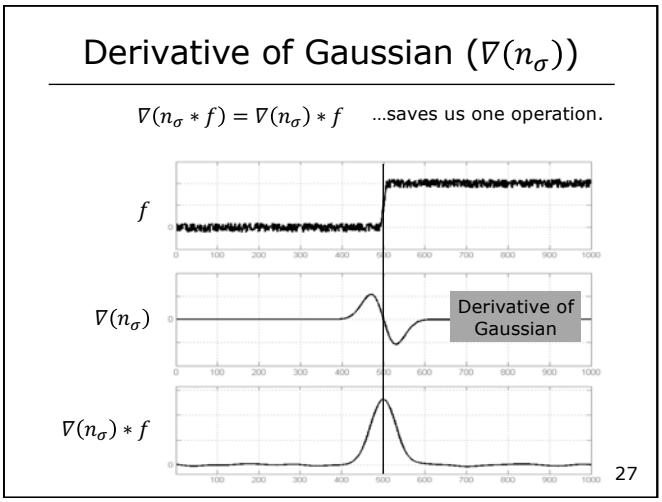
Let us talk about an important issue that we have set aside in our discussion above, which is noise. In the beginning of the lecture, we said that noise is why edges do not appear perfect in images. So, we have to contend with noise. On the top here is a 1D example of what noise on an edge looks like. The problem with noise is that it is rapidly changing everywhere. Since we defined edge detection as the problem of finding rapid changes, noise poses a major challenge. When we find the first derivative of $f(x)$, we see that the edge is completely lost.



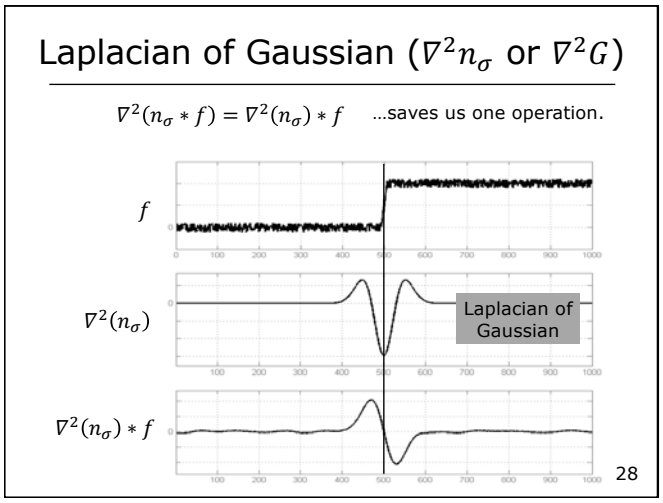
We need to have some way of dealing with noise. That is, we need to either remove, or at least suppress, the noise before we apply the gradient or the Laplacian operator. From our lecture on image processing, we know that one way to do this is by using Gaussian smoothing. Given our noisy signal $f(x)$, we can first smooth it with a Gaussian. Note that when we do this, most of the noise has been removed, and, as expected, the edge has been somewhat blurred. Now when we apply our gradient operator, we get a nice peak that is located at the edge.



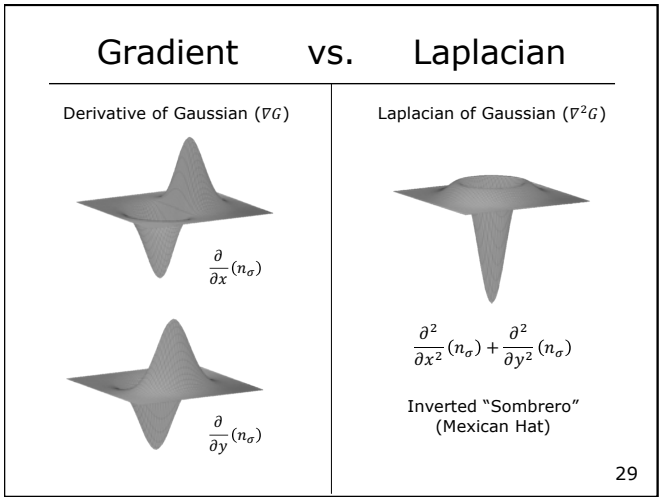
There is an interesting observation to be made here. Remember that we took the signal $f(x)$, convolved it with a Gaussian to reduce the noise, and then found its first derivative. We can achieve exactly the same result in a different way. We know that the first derivative is a linear operation, and we know that Gaussian smoothing is linear as well. Therefore, we can find the first derivative of the Gaussian, which gives us a new operator (filter) that we can convolve our signal with. Comparing the results in this slide and the previous one, we see that they are exactly the same.



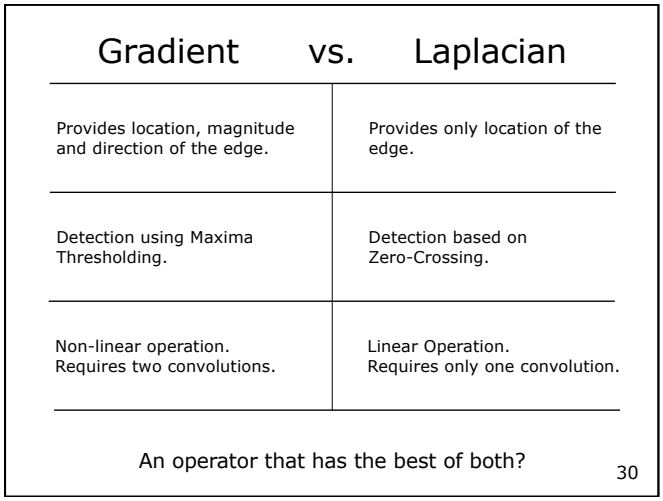
We can do the same thing with the Laplacian operator as well. Instead of applying the Laplacian operator to the input signal after it is smoothed, we can apply the Laplacian operator to the Gaussian to get what is called a Laplacian of Gaussian (LoG) operator, and then simply apply that to the input signal. As expected, we end up with a strong zero-crossing at the location of the edge.



On the left we see the gradient operator in 2D. We take the first derivative of the Gaussian with respect to x and with respect to y, and then convolve the image with both operators to obtain two numbers for each pixel. From those two numbers, we can calculate the strength and the orientation of the edge. Alternatively, we can apply the Laplacian to the Gaussian to get the operator shown on the right. This operator looks like an inverted Mexican hat (sombrero). This single operator can be applied to an image and the resulting zero-crossings correspond to edges.



Let us compare the properties of the gradient operator and the Laplacian operator. While the gradient operator gives us the location, magnitude and direction of the edge, the Laplacian operator just gives us the location of the edge. The gradient operator does require some form of thresholding to identify an edge. The threshold is chosen based on the needs of the application. In contrast, with the Laplacian operator, we are just looking for zero-crossings. In practice, however, we need to threshold each zero-crossing in some way to determine how rapid it is. Finally, the gradient operator requires two convolutions followed by a nonlinear operation for finding the magnitude and the



orientation, whereas the Laplacian operator is a single convolution which is a linear operation. These differences between the two operators are subtle and may not be critical in most applications. The more important question is whether we can come up with an edge detector that exploits the best characteristics of both of these operators and works better than either one of them.

The Canny edge detector is probably the most widely used edge detector in computer vision. It uses the best attributes of both the gradient operator and the Laplacian operator. Let us take a look at how it works.

Canny Edge Detector

Shree K. Nayar

Columbia University

Topic: Edge Detection, Module: Features

First Principles of Computer Vision

31

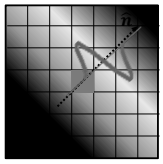
Canny Edge Detector

- Smooth Image with 2D Gaussian: $n_\sigma * I$
- Compute Image Gradient using Sobel Operator: $\nabla n_\sigma * I$
- Find Gradient Magnitude at each pixel: $\|\nabla n_\sigma * I\|$
- Find Gradient Orientation at each Pixel:

$$\hat{\boldsymbol{n}} = \frac{\nabla n_\sigma * I}{\|\nabla n_\sigma * I\|}$$

- Compute Laplacian along the Gradient Direction $\hat{\boldsymbol{n}}$ at each pixel

$$\frac{\partial^2 (n_\sigma * I)}{\partial \hat{\boldsymbol{n}}^2}$$



$\|\nabla n_\sigma * I\|$

[Canny 1986] 32

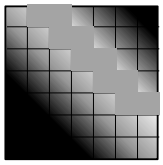
Canny Edge Detector

- Smooth Image with 2D Gaussian: $n_\sigma * I$
- Compute Image Gradient using Sobel Operator: $\nabla n_\sigma * I$
- Find Gradient Magnitude at each pixel: $\|\nabla n_\sigma * I\|$
- Find Gradient Orientation at each Pixel:

$$\hat{\boldsymbol{n}} = \frac{\nabla n_\sigma * I}{\|\nabla n_\sigma * I\|}$$

- Compute Laplacian along the Gradient Direction $\hat{\boldsymbol{n}}$ at each pixel

$$\frac{\partial^2 (n_\sigma * I)}{\partial \hat{\boldsymbol{n}}^2}$$



$|\nabla n_\sigma * I|$

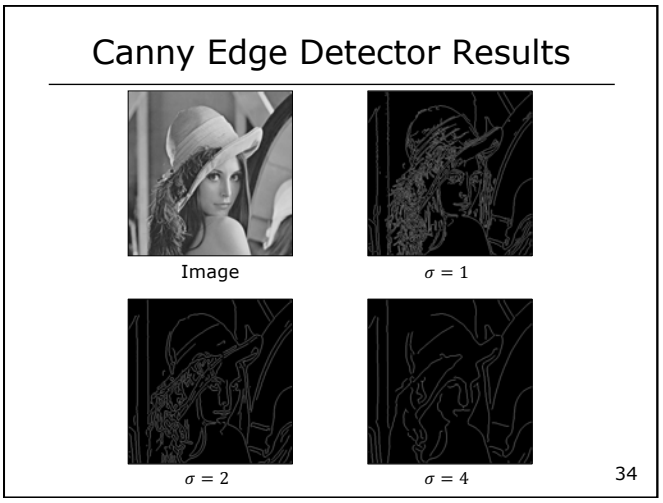
- Find Zero Crossings in Laplacian to find the edge location

[Canny 1986] 33

First, we smooth the input image with a Gaussian of width σ. Then, we apply the gradient operator (for instance, the 3x3 Sobel operator) to the smoothed image to get two numbers at each pixel — the derivatives in the x and y directions. From these, we can compute the magnitude and the orientation of the gradient at each pixel. The magnitude is what is being shown in the image here; the brighter the point, the greater the magnitude.

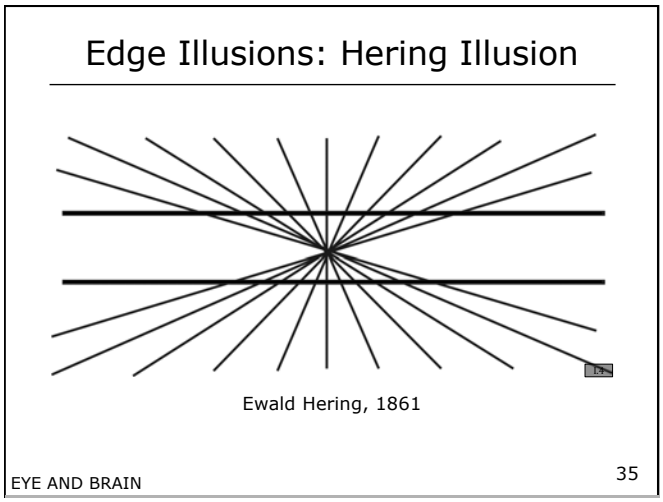
Next, at each pixel, a 1D Laplacian (second derivative) is applied along the gradient direction at the pixel. If a zero-crossing is detected, then an edge is declared at the location of the zero-crossing. The advantage of applying a 1D Laplacian along the gradient direction is that it is less effected by pixels that are unrelated to the edge. In the image in the slide on the right, the highlighted pixels correspond to the detected zero-crossings (edges).

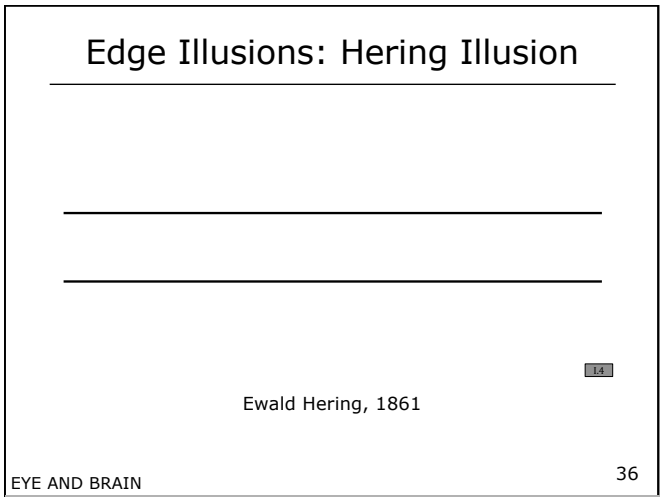
Now, let us take a closer look at the effect of the smoothing parameter, σ. If we start with a very small σ of 1, we end up with many edges after applying the Canny detector. If we increase σ to 2, we see that the number of edges decreases. If we increase σ further to 4, we see that we get even fewer edges. This phenomenon is related to the "scale space" associated with an image, where scale represents the resolution of the image.



Imagine that we had an image of a brick wall. If the image is of very high resolution, the edges would include not only the borders of the bricks, but also the fine pores inside each brick. As we smooth this image to lower its resolution, the details of the pores begin to fade and we get edges that mainly correspond to the borders of the bricks. So, when we talk about edges, they can exist at many different scales of resolution. What the Canny detector allows us to do is to simply change one parameter, σ, and extract the edges corresponding to the specific scale that is of interest to us.

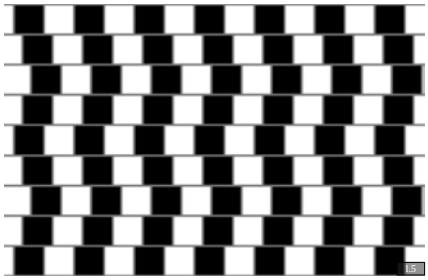
Next, let us take a look at a couple of illusions associated with edges. Below is the Hering Illusion. We can all agree that, in the slide on the left, the two horizontal lines appear to be bulging out. Now, if we remove all of the other lines in the image (right slide), we see that the two lines are perfectly straight and parallel to each other. The reason this happens is that when there are acute angles in an image, the human visual system tends to see these as less acute angles. As a result, the acute angles made by these two lines with respect to the other lines (spokes) in the original image (left slide) are overestimated such that the error increases with the acuteness of the angle. This bias makes us believe that the two horizontal lines are curved lines when they are not.





Here is another interesting illusion called the Cafe Wall illusion. Here, the horizontal gray lines appear tilted with respect to each other.

Edge Illusions: Café Wall Illusion



Gregory and Heard, 1979

EYE AND BRAIN 37

If we simply remove the black tiles on the wall, we see that the lines are perfectly parallel. This is because the human visual system tends to see black patches as being slightly smaller than they actually are. We do not fully understand the reasons for this. However, this bias causes the black tiles in the original image to appear smaller than they are. Consequently, the lines appear to be tilted, when they are not.

Edge Illusions: Café Wall Illusion

Gregory and Heard, 1979

EYE AND BRAIN 38

Corner Detection

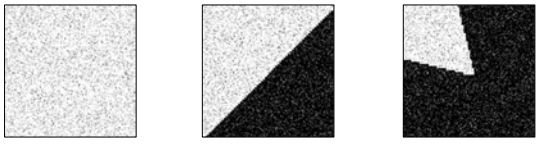
Shree K. Nayar
Columbia University

Topic: Edge Detection, Module: Features
First Principles of Computer Vision

39

Corners

Corner: Point where Two Edges Meet. i.e., Rapid Changes of Image Intensity in Two Directions within a Small Region

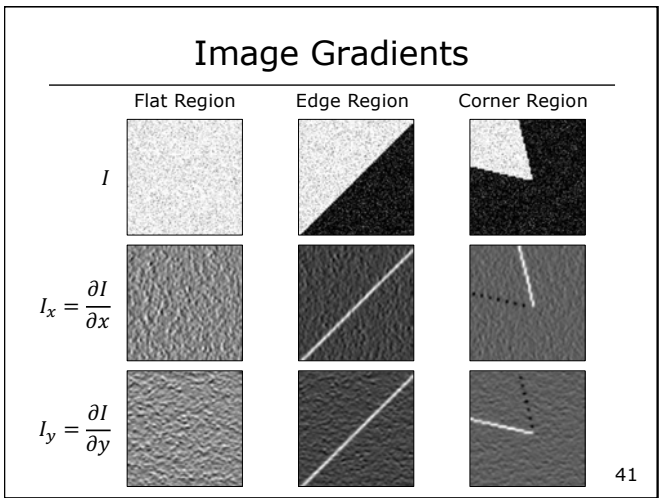


"Flat" Region "Edge" Region "Corner" Region

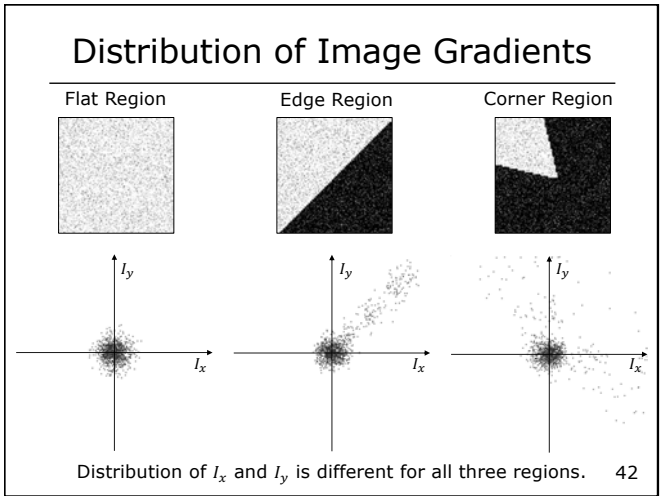
40

Let us now talk about how we can find corners in an image. On the right, we see three image regions – a flat region, an edge region and a corner region. A corner is essentially a point where two edges meet. Unlike an edge where we have a rapid change in image intensity along one direction, in this case, if we

place a window around the corner, we are going to see a rapid change in image intensity along two directions. As in the case of edge detection, we are going to use the derivatives of the image to perform corner detection.

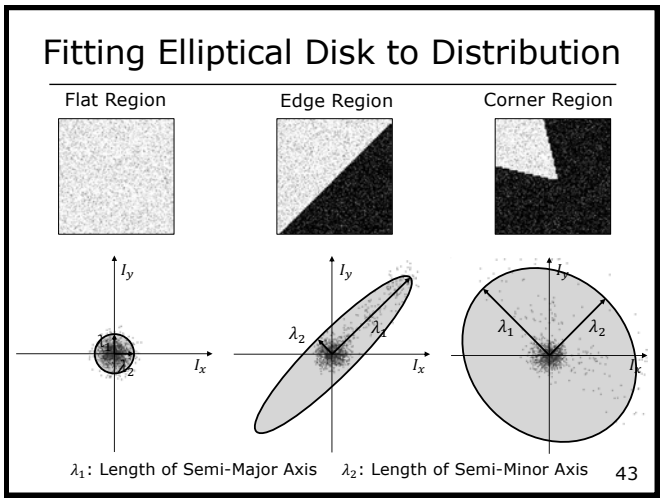


Here we see the three regions once again. We are now going to find the derivative of each image with respect to the x-direction (middle row) and the y-direction (bottom row). For each image, we are going to normalize it so that white corresponds to a strong positive value, black corresponds to a strong negative value, and shades of gray are values in between. For the flat region, as expected, we have mostly gray all over because we have low gradients in x and y, except for those produced by the noise in the image. In the edge case, we again get low gradients on both sides of the edge, but very strong, positive gradients in the x and y directions along, or close to, the edge. In the case of the corner, we see something interesting. We get low values pretty much everywhere, but we get strong positive values along one edge and strong negative values along the other for the gradient in the x-direction. We see the same for the y-gradient, except that the signs are reversed.

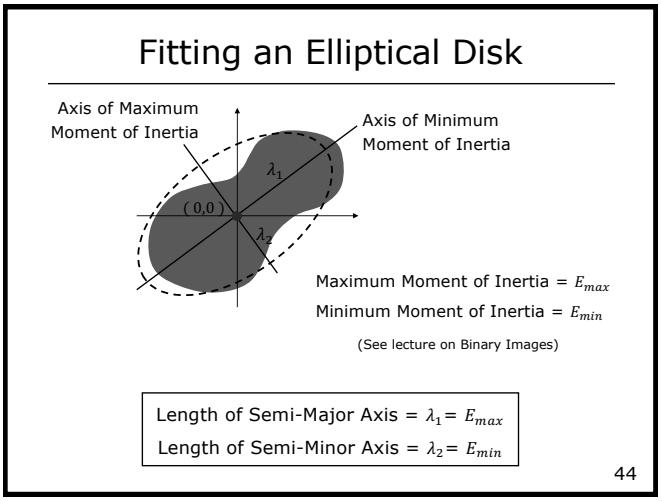


Let us now construct a 2D space whose dimensions correspond to the x and y image gradients, I_x and I_y. For each of the three image regions, we will plot the gradient values at each pixel in this space. For the flat region, as expected, we get a very compact cluster close to the origin. In fact, if the region did not have any noise, the cluster would shrink to a single point, namely, the origin. When we look at the edge region, we once again get a compact cluster because there are significant flat regions on both sides of the edge, but we also get a thin and long cluster. This cluster corresponds to the large gradient values obtained at or near the edge. In the case of the corner, we again get a compact cluster because of the flat regions, but we get two additional clusters, one for each of the two edges that make up the corner. Our goal is to quantify the structure of the distribution of points in this gradient space with a simple model that can be described using a small number of parameters. Then, we can use these parameters to classify each local image region as being flat, an edge, or a corner.

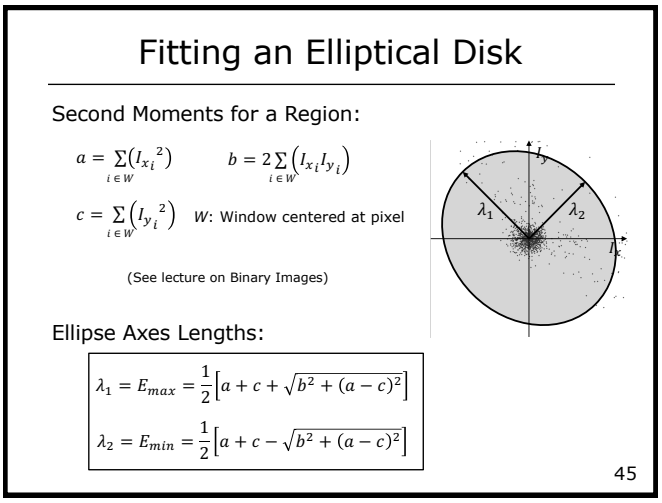
To do this, we will fit an ellipse to the distribution, which is centered at the origin. Based on the lengths of the major and minor axes of the ellipse, we will classify the region.



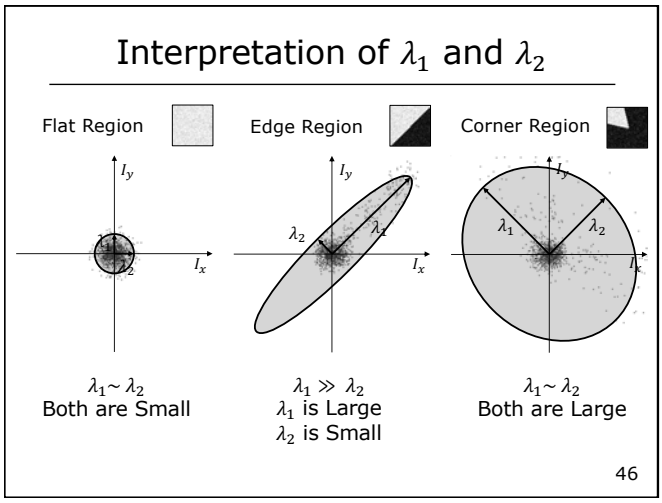
How do we fit an ellipse to a distribution? This takes us back to our lecture on binary images. Given a binary object, we know how to find the axis of minimum second moment. Perpendicular to this axis is the axis of maximum second moment. Let us say we want to find the ellipse that best fits the binary object. We will say that the length λ_1 of the semi-major axis of the ellipse is equal to maximum second moment E_{max}, and the length λ_2 of the semi-minor axis is equal to minimum second moment E_{min}.



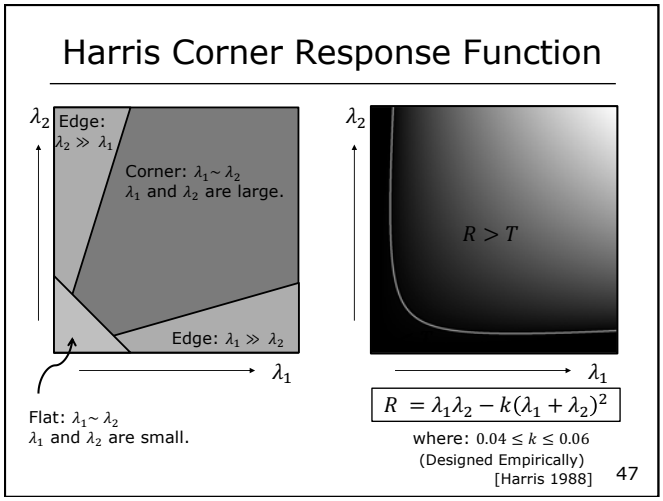
We will apply the same technique to quantity the structure of our distribution in gradient space. In other words, we are going to treat our distribution like a binary object, where each point in the distribution is a 1. We use the points in the distribution to compute the second moments, a, b, and c. From these three numbers, we can compute the maximum second moment E_{max}, and the minimum second moment E_{min}, which correspond to the semi-major axis λ_1 and semi-minor axis λ_2 of the ellipse that best fits the distribution.



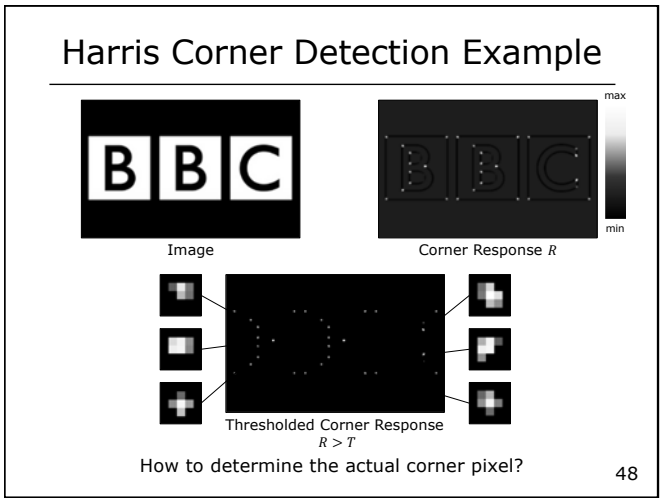
For the flat region, as expected, we are going to get small values for λ_1 and λ_2, as it is a compact cluster. In the case of an edge, λ_1 is large and λ_2 is small, and in the case of a corner, both are large. The Harris corner detector uses λ_1 and λ_2 to classify local image regions to detect corners.



Harris has designed a simple expression, called the response function, that maps λ_1 and λ_2 to a single number *R*. *R* is being plotted here on the right as a function of λ_1 and λ_2. If we now apply a threshold to *R*, the threshold corresponds to a curve (white line) in (λ_1, λ_2) space. If *R* is above the threshold, then both λ_1 and λ_2 are large, and it is likely a corner. Conversely, if *R* is less than the threshold, the image region is either flat or includes an edge.



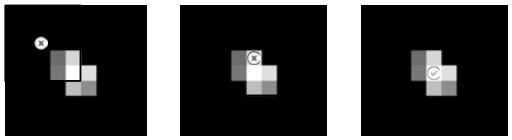
Here we have a very simple image to which we apply the Harris corner detector. One of the problems with feature detection, in general, is that the detector is likely to produce large responses not only at the exact location of the feature but also close to it. We see the same phenomenon with the Harris detector — the magnified image regions show large *R* values at and around each of the corners in the image. To find the exact locations of the corners, we need to detect the peak of each of these clusters in the response image.



This peak-finding problem is one that appears in many different detection and classification tasks, and the method we are going to describe is a general one that is widely applicable. It is called non-maximal suppression. In our case, we wish to find peaks in the Harris response image *R*. We take a small window and slide it over the entire response image. At each pixel, if its value happens to be the maximum value within the window, we retain it. If it is not the maximum value, which means that one or more pixels within the window have larger values, then it is either suppressed (reduced in value), or eliminated completely (set to zero). If suppression is used, the method will need to be applied repeatedly until the peaks are the only locations with non-zero values.

Non-Maximal Suppression

1. Slide a window of size k over the image.
2. At each position, if the pixel at the center is the maximum value within the window, label it as positive (retain it). Else label it as negative (suppress it).



Suppress Suppress Retain

Used for finding Local Extrema (Maxima/Minima)

49

Applying this method to our simple BBC image, we can see that the corners are more or less in all the right places. We do see some corners detected that may or may not be deemed to be corners, but we know that these were detected at locations where the letters in the image have high curvature. That is, they are features that are corner-like.

Harris Corner Detection Example

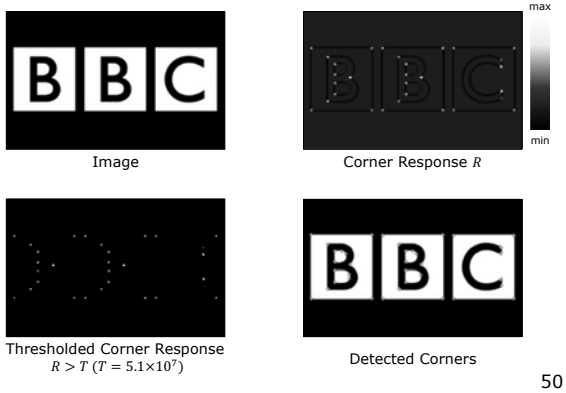


Image Corner Response R

Thresholded Corner Response $R > T$ ($T = 5.1\times10^7$) Detected Corners

50

Let us now apply the corner detector to a more challenging image. This is a printed circuit board with many weak and strong corners. Once again, we have our *R* image, which is thresholded to obtain the bottom left image. Note that this is not a binary image but rather one in which the original *R* value are retained if they are above the threshold. We then applied non-maximal suppression to get the corners shown in the bottom right image. If we take a close look at the detected corners, we see that most of the corners in the original image were successfully detected.

Harris Corner Detection Example

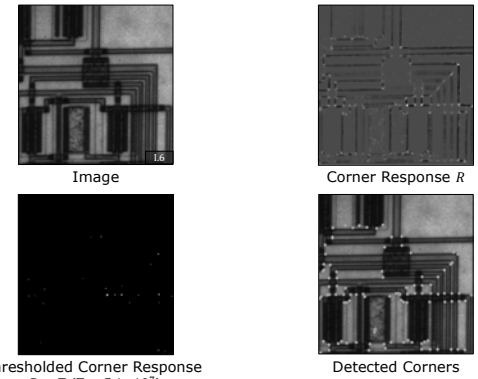
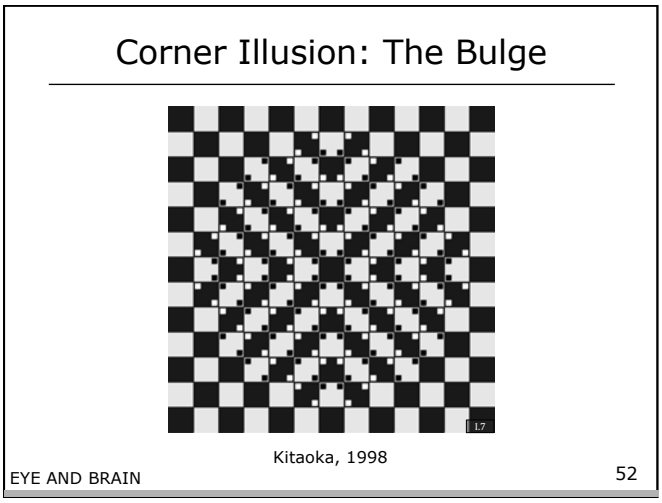


Image Corner Response R

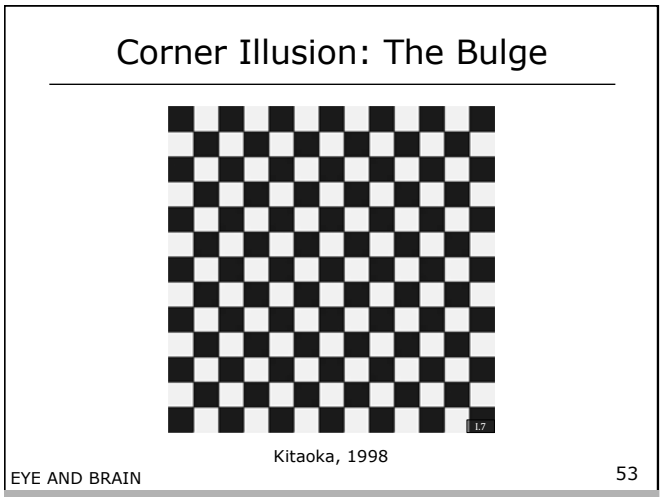
Thresholded Corner Response $R > T$ ($T = 5.1\times10^7$) Detected Corners

51

Since we are talking about corners, let us end with an illusion that is related to corners. In this image, the checkerboard pattern appears to bulge in the center. Note the tiny white and black squares close to the corners of the checkerboard pattern.



If we remove the tiny black and white squares, we see that we actually have a perfect checkerboard pattern. All the lines are either parallel or perpendicular to each other; there is no bulge. It turns out that by placing tiny squares close to the intersections in the checkerboard pattern, we are biasing the human visual system to believe that the corners are at slightly different locations from where they actually are. This is yet another bias in the human visual system that leads us to perceive something — the bulge — that does not exist. Once again, the punchline is: we are not perfect!



References and Credits

Shree K. Nayar
Columbia University

Topic: Edge Detection, Module: Features
First Principles of Computer Vision

54

References: Textbooks

A Guided Tour of Computer Vision (Chapter 3)
Nalwa, V., Addison-Wesley Pub

Computer Vision: A Modern Approach (Chapter 8)
Forsyth, D and Ponce, J., Prentice Hall

Digital Image Processing (Chapter 3)
González, R and Woods, R., Prentice Hall

Robot Vision (Chapter 8)
Horn, B. K. P., MIT Press

55

References: Papers

[Canny 1986] Canny, J., A Computational Approach To Edge Detection, IEEE Trans. Pattern Analysis and Machine Intelligence, 8(6):679–698, 1986.

[Harris 1988] Harris, C. and Stephens, M., A combined corner and edge detector. Proceedings of the 4th Alvey Vision Conference. pp. 147–151.

[Marr 1980] Marr, D. and Hildreth, E., "Theory of Edge Detection," Proc. R. Soc. London,B 207, 187-217, 1980.

[Nalwa 1986] Nalwa, V. S. and Binford, T. O., "On detecting edges," IEEE Trans. Pattern Analysis and Machine Intelligence, 1986.

56

Image Credits

I.1 A Guided Tour of Computer Vision. V. Nalwa. Addison-Wesley, 1993. Used with permission.
I.2 A Guided Tour of Computer Vision. V. Nalwa. Addison-Wesley, 1993. Used with permission.
I.3 Lena. Dwight Hooker, 1973.
I.4 commons.wikimedia.org/wiki/File:hering_illusion.svg Licensed under CC BY SA 3.0.
I.5 en.wikipedia.org/wiki/File:Café_wall.svg Licensed under CC BY SA 3.0.
I.6 https://www.mathworks.com/help/vision/ug/detect-lines-in-images.html. Mathworks.
I.7 A. Kitaoka. Used with permission.

57

Acknowledgements: Thanks to Nisha Aggarwal and Jenna Everard for their help with transcription, editing and proofreading.