

COMP 558

Lecture 4

## Edge Detection

1

Today we will consider the problem of how to detect edges in an image.

## What is an Edge?



2

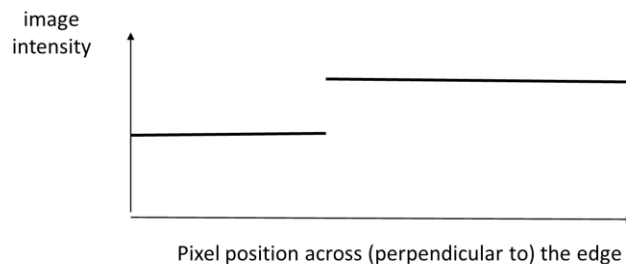
Loosely defined, an edge is a (1D) contour in the image across which the intensity changes significantly. In this figure, there is a diagonal edge separating a light gray and dark gray regions. There are also edges around this rectangular figure which separate it from the white background.

In a real image, an edge might occur at the boundary of an object. Or it might occur where there is a change in the material of an object. Or it could occur where there is a change in illumination, for example, at the edge of a shadow.

Detecting edges is useful if we want to find the boundaries of objects.

## Why is edge detection difficult?

Ideal (but not realistic) step edge:



3

In the previous slide, the edges were ideal. There was one image intensity on one side of the edge and another image intensity on the other side of the edge. This is illustrated with the ideal edge shown here, which shows the image intensity across the edge, that is, perpendicular to the edge.

Edges in real images are not so simple. There may not be a single position where the intensity changes value, but rather there may be a gradual slope – namely the edge may be blurred. A second reason is that the intensity (or RGB) may not be constant on each side of the edge, but rather the intensities may be sloped. In this case, we say there is an intensity *gradient*. The intensities may also have “texture” in them, which are small local and typically random changes in intensity. An example of texture is image noise. But many images will have texture that is independent of noise, for example, dirt, rocks, clouds, hair, grass all give rise to random local varying intensities.

## Example image



4

So let's take an example image. This is Pascal Siakem, one of my favorite players on the Toronto Raptors. (He is also known as "Spicy P".)

I'm sure you can identify many edges in this image: the boundaries of his headband, his jersey, the lettering, his arms, the ball.

Imagine yourself with a blank piece of paper and a pencil, and I ask you to make a line drawing reproduction of this image. What would you do? You would try to draw the various edges. But you would find it is sometimes difficult to identify where the edges are. It turns out it is also difficult to write computer programs to do it, at least, if you want the answer to be satisfying.

## Example: edge detection



`edge(I, 'Canny')`

5

Here is an example of what Matlab's Canny edge detector outputs. We'll discuss the Canny edge detector later today.

The results may not seem so impressive at first glance. But what did you expect to see? Perhaps your first reaction is that there are way too many edges?

Let's look at a few examples. The lettering on the jersey and the Nike logo seem to have been detected nicely.

Now look at the black stripes on the ball. You see that each stripe has two edges. That's correct, since the stripe has a finite thickness and so the intensity drops (into black) and then rises (back into "gray").

## Today: classical edge detection

- Prewitt and Sobel edge detectors (1960's)
- Marr & Hildreth edge detector (1979)
- Canny edge detector (1986)

6

Today we will discuss some classical edge detectors. The first ones were invented in the 1960's and are simplest to describe. In the late 1970's and the mid 1980's two more methods were proposed. In fact, many edge detection schemes were presented over these years. I am only covering a few of the most influential ones. All of these early methods are based on basic ideas in image filtering.

[To understand more modern edge detection methods, we need to go beyond simple image filtering. We'll come back to this problem in a few weeks, when we discuss image segmentation.]

## Edge detection based on a threshold

Given an image  $I(x, y)$ , find all  $(x, y)$  such that

$$| I(x + 1, y) - I(x - 1, y) | > \tau$$

where  $\tau$  is some arbitrary *threshold*.

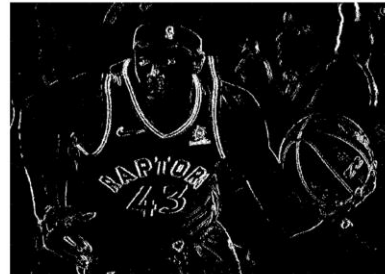
7

Let's start with the most basic idea. Suppose we are given a 2D image. We consider each row in the image, that is, each value of  $y$ . We would like to find the positions in the image where the intensity takes a large step from the left neighbor to the right neighbor. To do so, we define some threshold and examine where the image difference takes a value that is larger than this threshold.

Note we are using a "central difference" here. We could alternatively use a forward difference  $I(x+1, y) - I(x, y)$  or a backward difference  $I(x, y) - I(x-1, y)$ .



Input image



Output of naïve 1D edge detector  
(threshold = 100).

8

We are applying this edge detection method *within each row*. We set the threshold to be 100, and on the right we show in white those pixels whose absolute intensity difference is above that threshold.

This method does seem to detect some of the edges. But not all. Which edges are missing? Do you see a pattern here?





Input image



Output of naïve 1D edge detector  
(threshold  $\tau = 100$ ).

9

It is missing horizontal edges ! That's because it is looking for changes in intensity within rows. But the horizontal edges don't produce changes within rows. Rather they produce changes within columns (across rows). On the right I have marked several locations where there are horizontal edges but they are not detected.

Another issue is that some of the edges are "thick". There are consecutive pixels in a row which are marked as edges. We saw last class how this can happen. If the image has a step in intensity, then since the filter is three samples wide, you may get large values of the difference at two consecutive pixels.

## Image gradient

$$\nabla I(x, y) \equiv \left( \frac{\partial}{\partial x} I(x, y), \frac{\partial}{\partial y} I(x, y) \right)$$

$$\approx \left( \frac{1}{2} I(x+1, y) - \frac{1}{2} I(x-1, y), \frac{1}{2} I(x, y+1) - \frac{1}{2} I(x, y-1) \right)$$

$$I(x, y) * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \qquad I(x, y) * \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

10

We would have the same problems if we looked for edges within each column as well. Let's deal just with the first problem, that we are missing edges because we are looking only within rows or columns. An easy solution to this problem is to measure the local differences in both the x and y directions, and to combine these difference estimates into a vector. This is an approximation of the gradient which you recall from Calculus. On the slide, I show how to do this using central differences.

In Matlab terms, we can think of convolving an image with a 1x3 filter on the left and a 3x1 filter on the right. One subtlety on the right is that the -1 is at the bottom rather than the top element. The reason is that in Matlab, the y index of the image grows from top to bottom.

There is another subtlety here: in Matlab, when you have a matrix I that represents an image, the first index refers to the row and the second index refers to the column. So when indexing in Matlab, we sometimes want to index by I(y, x) rather than I(x, y). Heads up – you will very likely have an error in your Matlab code at some point because of this.

### "Prewitt" filter (1970)

1	0	-1
1	0	-1
1	0	-1

1	1	1
0	0	0
-1	-1	-1

### "Sobel" filter (1968)

1	0	-1
2	0	-2
1	0	-1

1	2	1
0	0	0
-1	-2	-1

11

Rather than using  $1 \times 3$  and  $3 \times 1$  filters for the partial derivative, we could use  $3 \times 3$  filters or other shaped filters. By averaging over more points, these filters will be more reliable in the presence of noise. In the very early days of computer vision, there were several such filters used. Two of them are shown here: the Prewitt and Sobel filters. The only difference is that the Sobel filter gives slightly more weight near the central pixel.

Notice that because the Sobel filter has larger weights, we would expect the Sobel filter to produce larger output values. If we really wanted to approximate the gradient as a "change in intensity per pixel distance", then we should multiply by some normalizing constant. We will omit this step here.

$$\| \nabla I(x, y) \| \equiv \sqrt{\left( \frac{\partial}{\partial x} I(x, y) \right)^2 + \left( \frac{\partial}{\partial y} I(x, y) \right)^2}$$

As before, given an image  $I(x, y)$ , find locations  $(x, y)$  such that

$$\| \nabla I(x, y) \| > \tau$$

where  $\tau$  is an arbitrary threshold.

12

How do we detect edges using the gradient? The idea is to consider the magnitude of the gradient vector. We find positions in the image where the magnitude of the gradient is larger than some chosen threshold.



Here is the Matlab result for the Prewitt filter. The result for the Sobel filter is almost identical so I am not showing it.

For the example on the top, you will notice that there are a few parameters passed to the edge function. There is a constant (.05) which I chose somewhat arbitrarily, and this constant has to do with how the threshold is chosen. (You can see the matlab implementation with the command “edit”, namely “edit edge” in this case.) There is also a parameter “nothinning” which addresses the thick edge problem that I mentioned a few slides ago. In the upper image, the lettering on the jersey (and many other edges) is thick, since the gradient is high for many neighboring pixels. The example on the bottom leaves this parameter out, as the default is that the edges should be thin. And that’s what happens. We’ll return to this idea of thinning later in the lecture when we discuss how the Canny edge detector works.

## Second Derivative

$$\frac{d^2 I(x)}{dx^2} \approx I(x+1) - I(x) - (I(x) - I(x-1))$$

↑  
Forward difference:  
Estimate of derivative  
at  $x + \frac{1}{2}$

↑  
Backwards difference:  
Estimate of derivative  
at  $x - \frac{1}{2}$

$$= I(x+1) - 2I(x) + I(x-1)$$

1	-2	1
---	----	---

14

We will next turn to a method that was developed in the late 1970's. This method posed the problem slightly differently. It used filters that were based on second derivatives, rather than first derivatives.

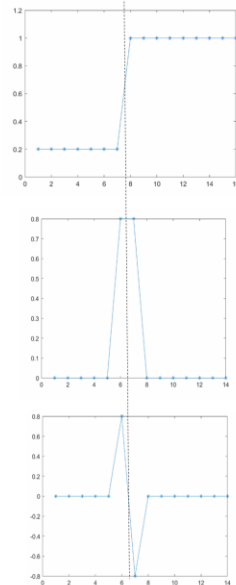
We can approximate the second derivative as shown here, namely as the difference of two first derivatives: we use a *forward difference* and a *backward difference* and we take the difference of these. This gives us the filter shown at the bottom. It is a filter defined on  $x = -1, 0$ , and  $1$  as before. But the weights are different than before.

## Example: step edge

$I(x)$

$$\frac{d I(x)}{dx} \approx \frac{1}{2} I(x+1) - \frac{1}{2} I(x-1)$$

$$\frac{d^2 I(x)}{dx^2} \approx I(x+1) - 2 I(x) + I(x-1)$$



15

Let's take a step edge function and compare what happens when we take the first and second derivatives. Think of the edge location in this example as occurring between two pixel positions – see the top row. There is no pixel exactly at the edge.

The first derivative produces a maximum (positive values) at the two pixels that straddle the edge.

The second derivative produces a positive and negative value on the left and right side of the edge.

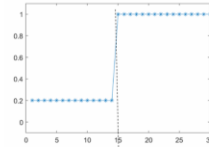
The dashed line marks the alignment of the different curves.

We say that the second derivative produces a “zero-crossing” at the edge itself.

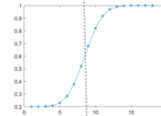
Note that the number of samples is different in the top row versus the middle and bottom row. This is because the Matlab ‘valid’ parameter was used in the convolution. I have chosen to line up the plots so that their edges coincide along a vertical line. Note that the x indices of the top plot do not coincide with the other two plots.

## Example: blurred step edge (no noise)

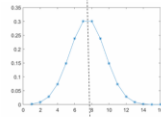
$$I(x) =$$



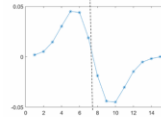
$$G(x, \sigma) * I(x) =$$



$$\frac{d G(x, \sigma) * I(x)}{dx} \approx$$



$$\frac{d^2 G(x, \sigma) * I(x)}{dx^2} \approx$$



When we take first and second derivatives as we just did, the results will be very sensitive to image noise. For this reason we typically *smooth* the image prior to taking derivatives. Here we take the same step edge as before (with more samples this time), and we want to know what is the effect on the step edge when we blur. Note that I have not added noise to the step edge. The reason is that we want to understand the effects of the blurring on the step edge signal alone here.

We use *Gaussian smoothing*, with some standard deviation  $\sigma$  which provides a convenient way of adjusting the amount of smoothing.

Let's ignore the effects at the image boundary and concentrate instead on what happens at the edge.

Blurring the edge by convolving with a Gaussian causes the step edge to be smoothed out, so that there is a gradual slope from the low intensity to the high intensity.

Note that the number of samples is much smaller in the second plot than the first. The reason is that the Gaussian filter is 13 samples wide, and I used the 'valid' parameter when convolving the step image with the Gaussian.

If we take the first derivative of the blurred image, then we see it is zero except near the edge. In particular the derivative is positive near the edge since the slope of the intensity function is positive. The first derivative curve is a maximum at the location of the edge. In order to show this in the slide, I had to manually align the plots and I shrunk the second one because it has fewer samples.



For the last plot: the second derivative of the blurred image crosses zero at the edge. This zero-crossing effect is similar to what we saw in the non-blurred case on the previous slide.

Associative property

$$f_1 * (f_2 * I) = (f_1 * f_2) * I$$

One can show (in the continuous case) that

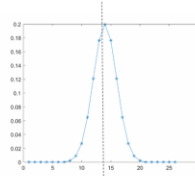
$$\frac{d}{dx} (G(x, \sigma) * I(x)) = \frac{d G(x, \sigma)}{dx} * I(x)$$

17

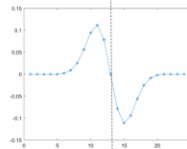
Recall the associativity property of convolution. Let's apply this in a surprising way. The operations of (1) blurring with a Gaussian and (2) taking a derivative can both be written as convolutions, as we have seen. We have shown this only for the discrete derivative, but one can show that the same statement holds for a continuous convolution, where the summation in the definition of convolution is replaced by an integral. That is what is meant by the convolution operator  $*$  in the second equation. Here, convolving with function  $G$  and taking the first derivative gives the same as defining a filter to be the first derivative of  $G$  and then convolving with it. To emphasize, on the right side of the lower equation, the image is being convolved with a derivative of Gaussian function.

## Derivative(s) of Gaussian

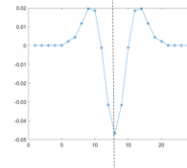
$$G(x, \sigma)$$



$$\frac{d G(x, \sigma)}{dx}$$



$$\frac{d^2 G(x, \sigma)}{d x^2}$$



18

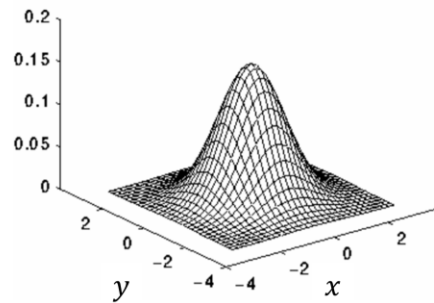
The first derivative of a Gaussian filter is similar to the filter  $f = [1, 0, -1]$  but it is a smoothed version of that filter. (When I say it is similar, I mean that it is positive for negative values of  $x$ , and negative for positive values of  $x$ .)

We also show the second derivative of a Gaussian. It is similar to the filter  $[1, -2, 1]$  that we saw a few slides ago, but again it is a smoothed version.

Let's next look at these filters in 2D.

## Recall 2D Gaussian

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

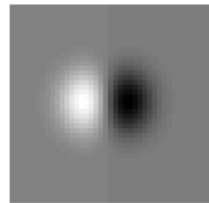
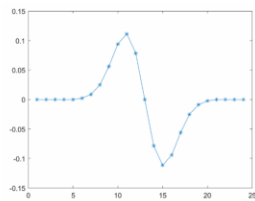


19

First recall the 2D Gaussian function. We are assuming here that the mean is 0 and the standard deviations in the x and y directions are the same.

## 2D Derivative of Gaussian

$$\frac{\partial G(x, y, \sigma)}{\partial x} \approx \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} * G(x, y, \sigma)$$

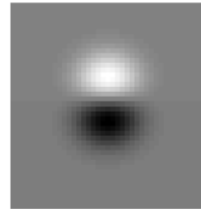
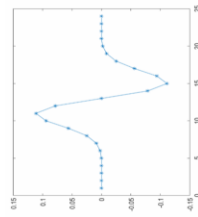


20

If we take the derivative of a 2D Gaussian in the x direction, we get the filter shown at the bottom right. White is positive and black is negative. If you take a horizontal slice through the image and plot the intensity then you get a plot similar to the one shown on the left (which was from two slides ago). I say similar rather than same because the parameters are not the same. To understand why they are similar, just plug  $y=0$  into the equation of a 2D Gaussian.

## 2D Derivative of Gaussian

$$\frac{\partial G(x, y, \sigma)}{\partial y} \approx \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} * G(x, y, \sigma)$$



21

If we take the derivative of a 2D Gaussian in the y direction, we get the filter shown at the bottom right. White is positive and black is negative. The plot on the left is the same as on the previous slide but rotated by 90 degrees.

## Gradient of smoothed image

$$\nabla G(x, y, \sigma) * I(x, y)$$

$$\equiv \left( \frac{\partial}{\partial x} G(x, y, \sigma) * I(x, y), \quad \frac{\partial}{\partial y} G(x, y, \sigma) * I(x, y) \right)$$

22

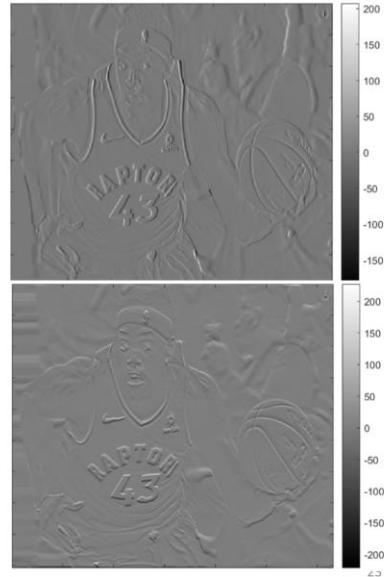
We can define the gradient of the smoothed image as shown here. In the discrete case, we approximate the gradient of the smoothed image by convolving the image with two filters which are first derivatives of Gaussians, namely the filters on the previous few slides. At each point  $(x, y)$ , we have a vector.

In the slides that follow, we will write gradient but the understanding is that we are using the discrete approximation.

## Example

$$I(x, y) * \frac{\partial G(x, y, \sigma)}{\partial x}$$

$$I(x, y) * \frac{\partial G(x, y, \sigma)}{\partial y}$$



For example, here we show the result of convolving with the x derivative of a Gaussian and the y derivative of a Gaussian. Recall that this is the same thing as blurring with a Gaussian and then taking the derivatives in the x direction (above) and the y direction (below).

You can check for yourself that you get very similar images if you convolve with the discrete derivative as we did at the start of the lecture, rather than convolving with the derivative of a Gaussian. The main difference is that the discrete derivative on its own is more sensitive to image noise that is present.

As we observed earlier in the lecture, when we take the derivative in the horizontal direction (within rows), we do not detect horizontal edges. Look at the center of the headband or the top edge of the basketball. Notice that these locations are more clearly marked in the bottom image where we take the derivative in the y direction. (On the other hand, in the bottom image, notice that the shoulder straps of the jersey are not detected well. )





$$| \nabla G(x, y, \sigma) * I(x, y) |$$

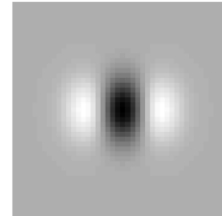
24

Now here is a plot of the magnitude of the gradient of the smoothed image. You can see that it highlights many of the edges. If we were to threshold it, we would get a result similar to what we saw with the Prewitt edge detector. But because we have smoothed the image, (the claim is that) it would be less sensitive to noise.

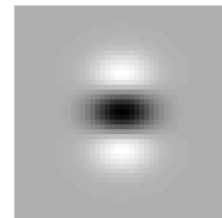
Later we will see a classical method (Canny edge detection) that is based on the magnitude of the gradient of the smoothed image.

## Second Derivative of Gaussian

$$\begin{bmatrix} 1 & -2 & 1 \end{bmatrix} * G(x, y) \approx \frac{\partial^2 G(x, y)}{\partial x^2}$$



$$\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} * G(x, y) \approx \frac{\partial^2 G(x, y)}{\partial y^2}$$



25

We can similarly define a filter that is a *second* derivative of a Gaussian in the x direction and another filter that is a second derivative of a Gaussian in the y direction. These filters are illustrated on the right. Again, black means negative and white means positive.

## Laplacian (operator)

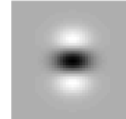
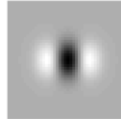
$$\nabla^2 I(x, y) \equiv \frac{\partial^2 I(x, y)}{\partial x^2} + \frac{\partial^2 I(x, y)}{\partial y^2}$$

26

Next define the Laplacian operator, which is the sum of second derivatives in the x and y directions. (The Laplacian operator comes up in many physical models.) Notice that, unlike with the gradient operator, here the result is a single valued function of (x,y). That is, it is a scalar field.

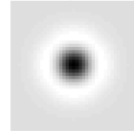
## Laplacian of a Gaussian

$$\nabla^2 G(x, y, \sigma) \equiv \frac{\partial^2 G(x, y, \sigma)}{\partial x^2} + \frac{\partial^2 G(x, y, \sigma)}{\partial y^2}$$



Not difficult to show that ....

$$= -\frac{1}{\pi\sigma^4} \left( 1 - \frac{x^2 + y^2}{2\sigma^2} \right) e^{-\frac{x^2 + y^2}{2\sigma^2}}$$



27

In particular, consider the Laplacian of a (2D) Gaussian function.

It is not difficult to show using basic Calculus that the Laplacian of a Gaussian is given by the second expression here.

If you substitute in either  $x=0$  or  $y=0$  into this expression, then you get a cross section in the  $y$  and  $x$  directions respectively. You can see that this is a 1d second derivative of a Gaussian. You can also see that the Laplacian of Gaussian is a radially symmetric function, as it depends only on  $x^2 + y^2$ . So it has the same cross section (through the origin) regardless of orientation. Recall that the 2D Gaussian also had this radial symmetry property.

## Example: vertical step edge

$I(x, y)$



$\nabla^2 G(x, y, \sigma)$

Q: What happens when we convolve with  $\nabla^2 G(x, y, \sigma)$  ?

$$\nabla^2 G(x, y, \sigma) * I(x, y) \equiv \frac{\partial^2 G(x, y, \sigma) * I(x, y)}{\partial x^2} + \frac{\partial^2 G(x, y, \sigma) * I(x, y)}{\partial y^2}$$

$= 0$

28

Let's now return to the edge detection problem. Take a 2D image with a vertical edge shown at the upper left.

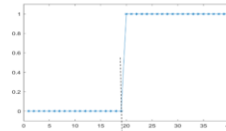
What happens when we convolve the step edge with a Laplacian of a Gaussian function?

The first observation is that the second derivative of  $y$  will not play any role. The reason is that  $I(x, y)$  does not depend on  $y$  for this image and so  $G(x, y, \sigma) * I(x, y)$  does not depend on  $y$ , and so the first derivative in the  $y$  direction is 0 everywhere and so the second derivative in the  $y$  direction will also be 0 everywhere.

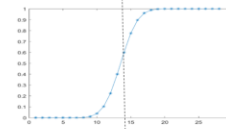
In particular, convolving the vertical step with a Laplacian of a Gaussian will give the same result as in the 1D case when one convolves with a second derivative of a 1D Gaussian.

## Recall: blurred step edge

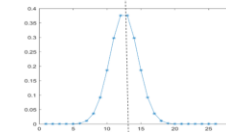
$$I(x) =$$



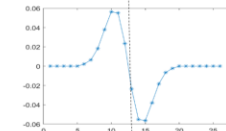
$$G(x, \sigma) * I(x) =$$



$$\frac{d G(x, \sigma) * I(x)}{dx} \approx$$




$$\frac{d^2 G(x, \sigma) * I(x)}{dx^2} \approx$$



This is the same slide shown earlier. So we recall that when we convolved the 1D step edge with a first derivative of a Gaussian, we got a zero crossing at the location of the edge. What this means in the 2D case of a vertical edge (two slides ago) is that we get a zero crossing *along the vertical edge*, namely the edge that separates the left and right constant intensity regions.

## Example: horizontal image edge



The diagram illustrates a horizontal image edge. On the left, a square is divided horizontally by a dashed line. The top half is light gray and the bottom half is dark gray. To the left of this square is the label  $I(x, y)$ . To the right of the square is the label  $\nabla^2 G(x, y, \sigma)$  followed by a small square icon containing a central dot with a radial gradient, representing the Laplacian of a Gaussian kernel. Below the diagram is the convolution equation:

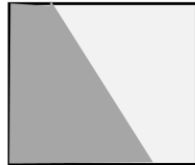
$$\nabla^2 G(x, y, \sigma) * I(x, y) = \frac{\partial^2 G(x, \sigma)}{\partial y^2} * I(y)$$

30

We can make a similar argument for a horizontal edge. When we convolve with a Laplacian of a Gaussian, now the second derivative of a Gaussian in the x direction is 0 everywhere since the image is constant along all rows. So only the second derivative of Gaussian in the y direction plays a role. Again we get a zero crossing, but now it is along the horizontal edge.

## Edge of arbitrary orientation

$I(x, y)$



$\nabla^2 G(x, y, \sigma)$



Since  $\nabla^2 G(x, y, \sigma)$  is radially symmetric,

$\nabla^2 G(x, y, \sigma) * I(x, y)$  will have a *zero-crossing along an edge of any orientation.*

31

Since the Laplacian of a Gaussian is radially symmetric, one can show that convolving with a Laplacian of a Gaussian will produce a zero crossing at the location of any edge, regardless of its orientation !



## Marr-Hildreth edge detection (1979)

$I(x, y)$



Compute  $\nabla^2 G(x, y, \sigma) * I(x, y)$

The edges are the points  $(x, y)$  where there is a zero-crossing.

32

This leads to the basic idea of the Marr-Hildreth method of edge detection: convolve the image with a Laplacian of a Gaussian function and look for zero-crossings.

There is more to the Marr-Hildreth theory than this. For example, they suggest using several different Gaussians that have different standard deviations (called “scales” for now) and they suggest only considering edges that produce zero-crossings over multiple scales. We will not discuss the details here, but related ideas about scale will come up in a few weeks.

[ASIDE: A second important part of the Marr-Hildreth theory they connected the Laplacian of Gaussian operator to neural operations basic operations in the eyes and in the brain. We will not discuss this aspect of their theory. ]

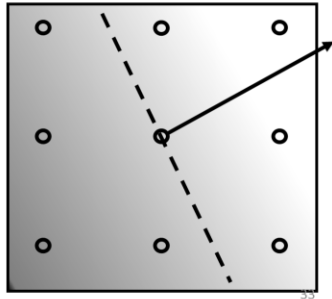
I encourage you to scan through their paper (link given in slide title) and at least look at the figures. It was one of the most influential papers in computer vision at that time, and it is now regarded as classic.

## Canny Edge Detection (1986)

Step 1: Compute gradient at each pixel.

$$\nabla G(x, y, \sigma) * I(x, y)$$

$$\equiv \left( \frac{\partial}{\partial x} G(x, y, \sigma) * I(x, y), \quad \frac{\partial}{\partial y} G(x, y, \sigma) * I(x, y) \right)$$

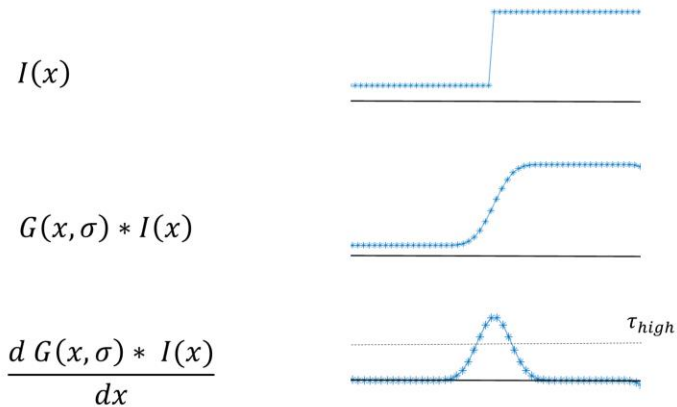


Mark a pixel as a candidate edge if its gradient magnitude is greater than  $\tau_{high}$ .

Let's finally turn to another classical edge detection method, which was due to Canny. Like the methods we described earlier from the 1960's, this method computes the gradient at each image point. It does so slightly differently though, namely it first blurs the image to remove noise. One of the contributions of the paper is to analyze the effects of noise. I'll say a bit more about that in a few slides.

Like the edge detection methods from the 1960's, Canny's method marks an image location as an edge if the (smoothed) gradient there has a magnitude that is larger than some threshold. We will call the threshold  $\tau_{high}$ . (There will be another gradient magnitude  $\tau_{low}$  that comes into play.)

Recall for a blurred 1D step edge, many points may have first derivative above threshold.



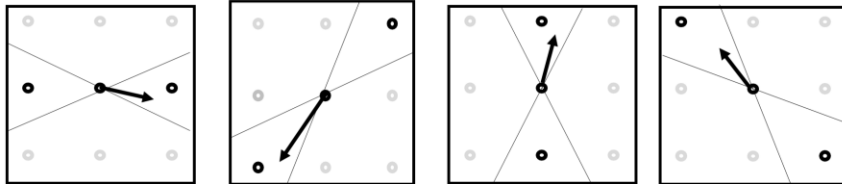
34

As we can see in the 1D case, the magnitude of the gradient will have a maximum at the location of the edge. If we define a threshold for the magnitude of the gradient, then the gradient magnitude might be greater than this threshold at many points. The number of points depends on how big a step in intensity there is at the edge. It also depends on the sigma value of the Gaussian, since a larger Gaussian will smooth the edge more, so the slope will be less and it will be less likely to exceed the threshold.

To localize the edge, we would like to keep only those detected points that are right at the edge, not in some thick neighborhood around the edge. This thinning issue is discussed on the next slide.

# Canny Edge Detection

## Step 2: “non-maximum suppression”



Compare the gradient magnitude of an edge from step 1 to the gradient magnitude of the neighbors who are *nearest to the direction of that gradient*. Eliminate edge pixels if the magnitude of their gradient is *not* a local maximum relative to the neighbors.

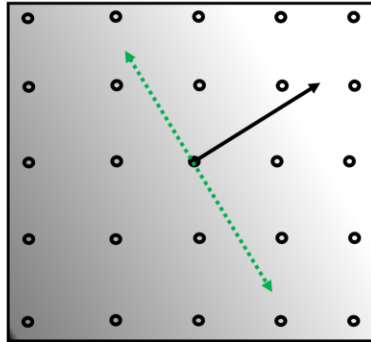
35

The second step of Canny’s method is called “non-maximum suppression”. Consider a pixel, centered in some 3x3 neighborhood. (One can use a large neighborhood, but here we use only 3x3 to keep it simple.) Suppose that the gradient magnitude at the central pixel is above the threshold  $\tau_{high}$ . The question is, should we really consider this pixel to be an edge? To decide this, we need to decide if its gradient is a local maximum. Consider four pairs of ranges of directions about each pixel, which are shown in the figure. For example, for the leftmost example, we have a pair of ranges  $[-45, 45]$  and  $[135, 225]$  degrees. If the gradient vector at that central point is in those ranges of directions, then we keep that central point as an edge point if the magnitude of the gradient at that central point is greater than the magnitude of the gradients at the two neighboring points in the two sections – namely the neighbors to the left and right shown in the figure. Similarly, if the gradient is in a different sector as shown in one of the other three cases, then we compare the gradient magnitude at that pixel to the magnitude at the pixels in the appropriately chosen sections.

This step thins out an edge. It tends to keep only those edge points whose gradient magnitudes are a maximum across the edge. The maximum tends to occur very close to the exact location of the edge.

# Canny Edge Detection

Step 3: “hysteresis” thresholding (details omitted)



For each edge pixel that survives step 2, search neighbors in two opposite directions perpendicular to the gradient and find a chain of pixels whose gradient magnitudes are all greater than a threshold  $\tau_{low}$  ( $\tau_{low} < \tau_{high}$ ). Consider the pixels in such chains to be edge pixels too.

36

There is a 3<sup>rd</sup> step to Canny’s method. This may seem like a “hack”, but it does turn out to be useful. The idea is that edges don’t occur at isolated points, but rather they tend to occur along 1D contours that extend for some distance in the direction perpendicular to the gradient (namely the green direction in the figure). So if you conclude that there is an edge at some pixel because of what happened at step 2, then this edge is evidence that there will also be an edge at those pixels that lie near the dashed green line shown. So, you might be willing to consider such nearby points as edge points even if their gradient magnitudes don’t satisfy the thresh\_high threshold.

The idea of step 3 is to set a lower threshold (thresh\_low) and to mark those pixels (in the green line direction) as edges too, if their gradient magnitudes meet a weaker criterion, namely they gradient magnitudes are larger than thresh\_low. This argument can continue along a chain perpendicular to each successive pixel until the thresh\_low condition fails, at which point the chain ends.

In addition to detecting more edge points, this method has the advantage of linking edges into contours, which could be useful in itself, for example, for shape analysis.

## J. Canny: “A Computational Approach to Edge Detection”

Canny also introduced theoretical criteria for a filter to detect *1D step edges in the presence of noise*.



1. An edge detector should have a much larger response to the edge than it should to noise.
2. An edge detector should accurately estimate the *location* of edges.

He showed mathematically how varying the sigma of the Gaussian smoothing trades off (1) versus (2): increasing sigma improves the ratio of response of edge to response of noise, but reduces localization accuracy.

37

I have put a link to Canny's paper in the title of this slide. I suggest that you spend five minutes scanning through the paper, since it is a classic. You'll see that it quite mathematical. Canny worked hard to analyze the 1D case of a simple step edge with noise, namely he tried to derive what is the *\*best\** detector of a step edge. What did he mean by best? He defined two objectives, which are listed as points 1 and 2 on my slide (read them).

He showed that a first derivative of a Gaussian filter satisfies these criteria well, but that there is a tradeoff. As we increase the sigma of the Gaussian, we increase the response of the filter relative to the response of the noise – which is good because we don't like noise. This property suggests that one should use a big sigma (large blur). However, as you increase sigma, you also reduce the accuracy of the localization of the edge. The reason intuitively is that the maximum of the gradient magnitude is not as high when sigma is bigger, and the response curve becomes flatter at the top, so it is more difficult to say exactly where the maximum of the gradient magnitude is.

Another problem with using a large sigma in practice is that the ideal step edge model which Canny assumes is less likely to be correct as one gets further from the location of the edge. For example, many edges curve, and the intensities are often not constant on each side of the edge but rather they may be increasing or decreasing.