M30242 Graphics and Computer Vision

Lecture 4 Edge Detection (Continued)

Overview

- Detecting edges by second derivatives: Laplacian filter.

- Smoothing images: Gaussian filters

- Edge detection from zero-cross: Laplacian of Gaussian (LoG)

- Canny edge detector

Recap: Sobel Edge Detection

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

Find vertical edges

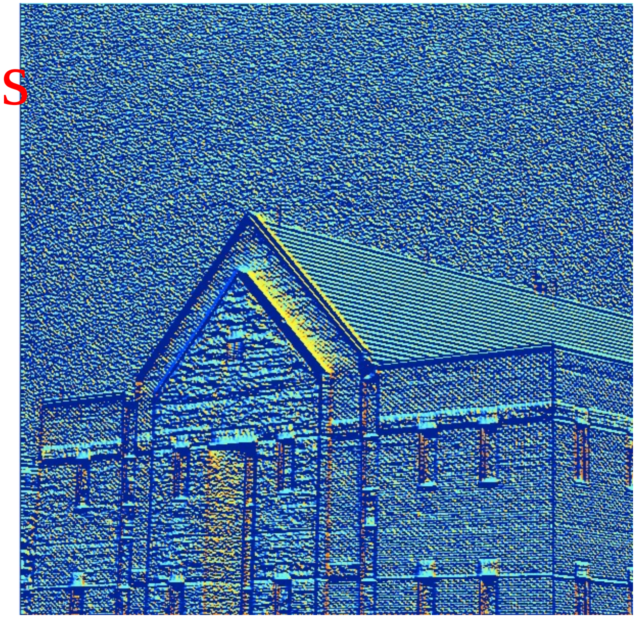
1	2	1
0	0	0
-1	-2	-1

Find horizontal edges

Find combined edge

bd_edge = sqrt(bx. ^2+by.^2);

$$|\nabla f| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$



Show gradient angles

ang=atan2(by, bx);

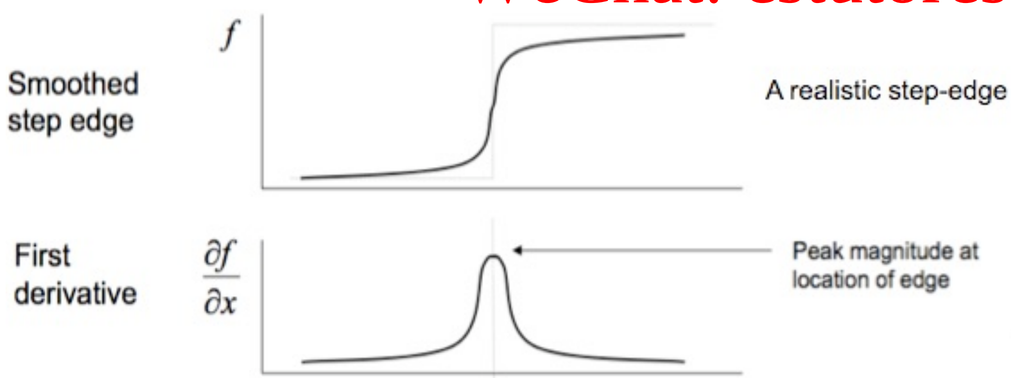
$$\theta \approx \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

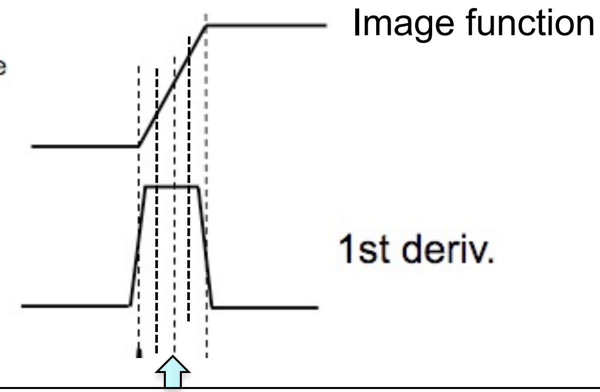
Properties of Good Edge Detectors

- Isotropic
  - Work well for edges of every directions
- Good localisation
  - Give accurate edge location and orientation
- Good signal-to-noise characteristics
  - Robust to noise

How Good Are Prewitt & Sobel?

- Isotropic
  - Two filters are needed : vertical and horizontal
- Good localisation & orientation
  - An edge may exist across several (rows/columns of) pixels
- Good signal-to-noise characteristics
  - Sensitive to noises if used alone

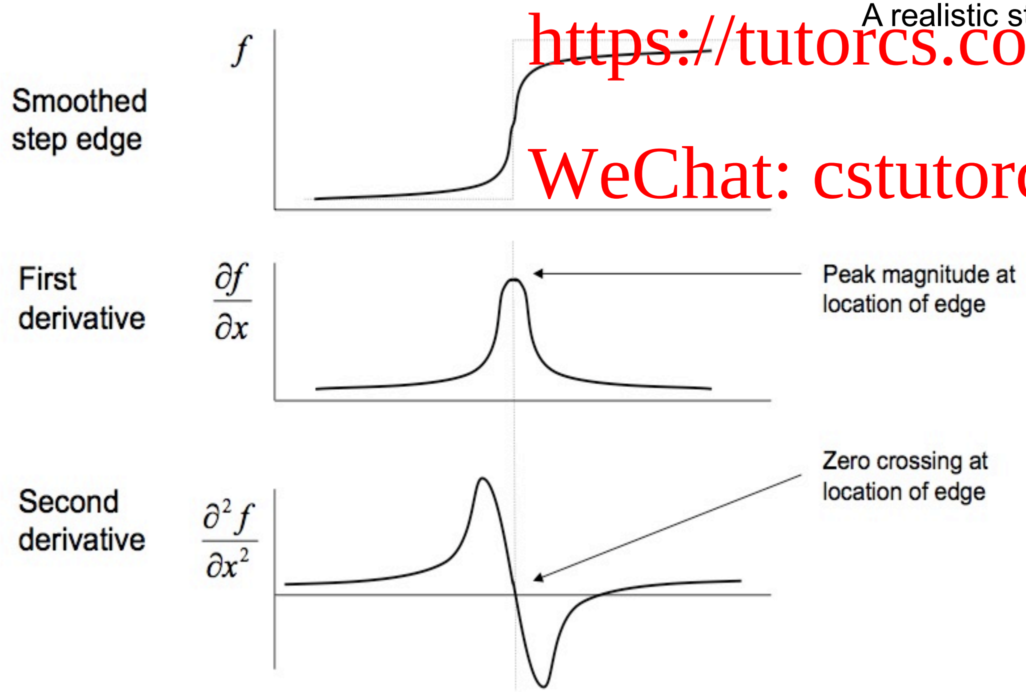




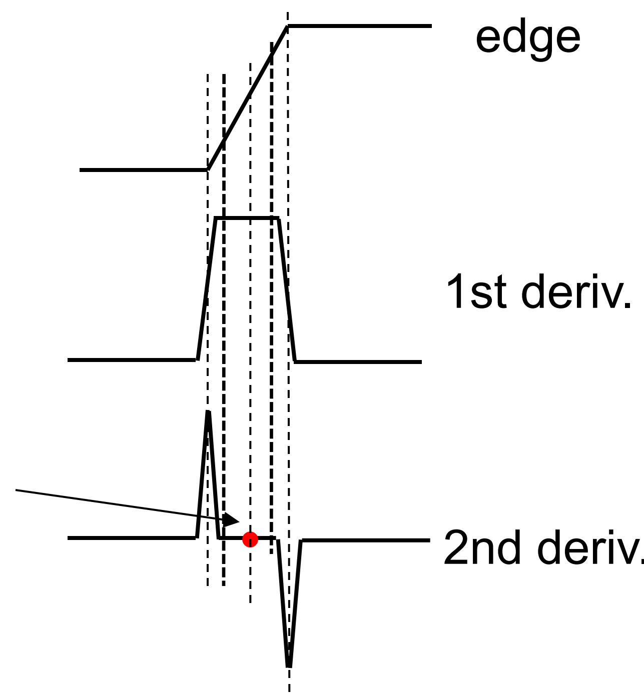
Intensity transition may happen across many pixels and the location of the edge could be anywhere within the transition region

Second Derivatives

- Use 2nd derivatives to detect edges can overcome some of the problems
  - Isotropic
  - Better localisation (zero crossings)



A realistic step-edge



Second Derivatives in 2D

- Discrete second derivatives can be calculated/approximated by

$$\frac{\partial^2 f}{\partial x^2} = [f(x+1,y) - f(x,y)] - [f(x,y) - f(x-1,y)]$$

$$= f(x+1,y) - 2f(x,y) + f(x-1,y)$$

Similarly

$$\frac{\partial^2 f}{\partial y^2} = f(x,y+1) - 2f(x,y) + f(x,y-1)$$

	$f(x, y+1)$	
$f(x-1,y)$	$f(x, y)$	$f(x+1,y)$
	$f(x, y-1)$	

Second Derivatives in 2D

- The **sum** of 2nd derivatives is the so called *Laplacian* operator in calculus:

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

$$= [f(x+1,y) - 2f(x,y) + f(x-1,y)] + [f(x,y+1) - 2f(x,y) + f(x,y-1)]$$

$$= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

2^{nd} Derivative Filter

Filter/mask that implements 2^{nd} directives in both x- and y-directions:

$$\begin{matrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{matrix}$$

	$f(x, y+1)$	
$f(x-1,y)$	$f(x, y)$	$f(x+1,y)$
	$f(x, y-1)$	

In practice, their negatives are used

$$\begin{matrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{matrix}$$

Laplacian with Diagonals

In image processing, to make the filter more sensitive to the edges in diagonal directions, 2nd derivatives of **diagonal** directions are brought into the formula. The filter is still called Laplacian and it becomes:

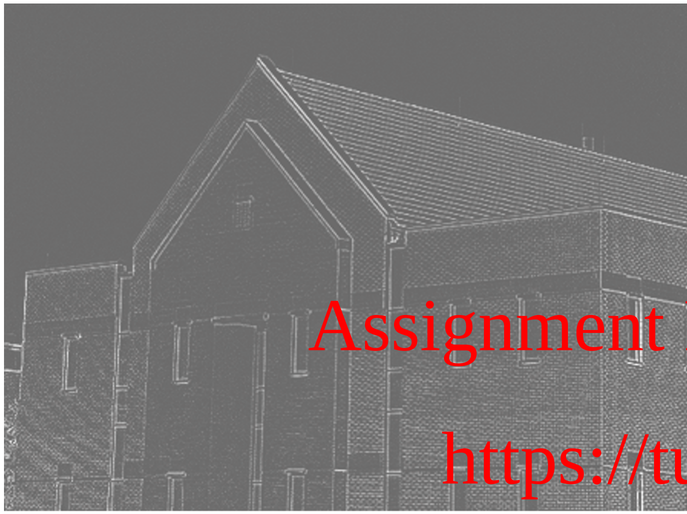
$$\begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix}$$

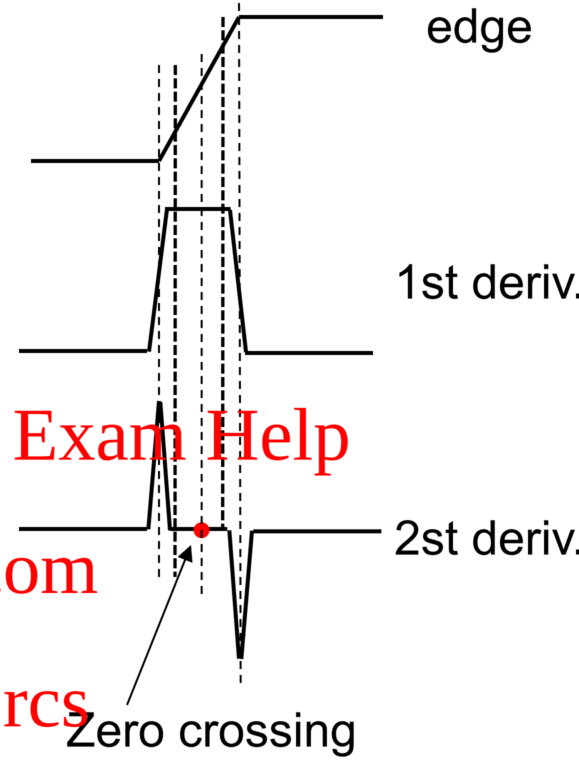
And its negation:

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{matrix}$$

$f(x-1, y+1)$	$f(x, y+1)$	$f(x+1, y+1)$
$f(x-1,y)$	$f(x, y)$	$f(x+1,y)$
$f(x-1, y-1)$	$f(x, y-1)$	$f(x+1, y-1)$

$\nabla^2 f$





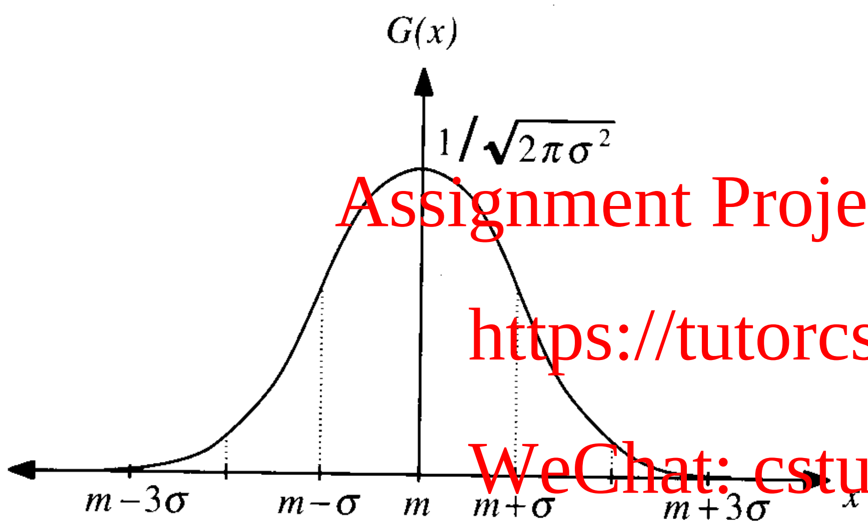
- Laplacian is an isotropic filter - invariant under rotation (because the derivatives in the diagonal directions)
- If used alone, the filter
  - produces double edges - inaccurate localisation
  - is sensitive to noise
  - gives no directional info of the edges

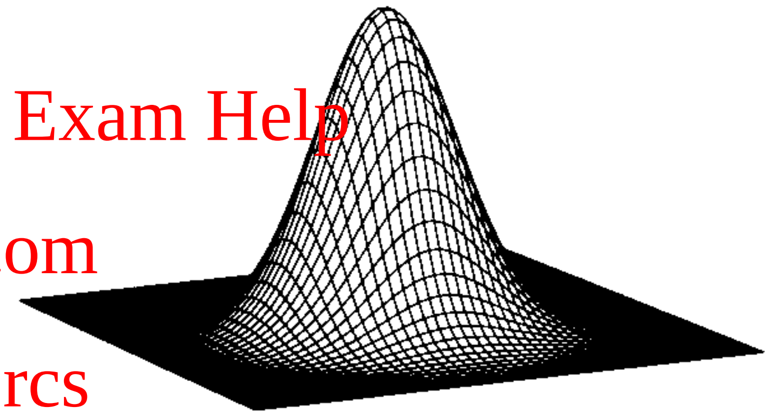
- Improvements can be done by:
  - Filtering out high frequency noises (by averaging)
  - Finding "zero crossings" to get better localisation

Smoothing By Averaging

- Averaging will make the pixels look more like their close neighbours.
- To get good smoothing result, i.e., to remove noise but not overly smooth the image, one needs to
  - Control the size of neighbourhood by choosing an appropriate size for the averaging (filtering) operations.
  - Control the influence from different neighbours. The closer a pixel is to the current pixel (corresponding to the centre of the mask), the more should it contribute to the result of averaging – this can be realised by setting different weights to the neighbour pixels according to their distances from the centre pixel.

Setting Weights According to Gaussian Distribution





$$\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(x-m)^2}{2\sigma^2}\right)$$

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

σ is called standard deviation. The bigger is this value, the flatter is the shape

Gaussian distribution in 2D – the form used for image processing

Useful Properties of Gaussian Function

- Weights decrease smoothly to zero as the distance from origin increases – the pixels nearer to the centre pixel are more important than remote pixels.
- The shape of the distribution is controlled by the value of σ : the bigger σ is , the more flat the shape is.
- The distribution is symmetrical about the centre - invariant to rotation.

Parameters

- Therefore, a Gaussian filter has two parameters: size and σ.
  - Bigger size means more pixels get involved in the averaging calculation and image gets more blurred.
  - Bigger σ means the weights are more spread out (“flatter”).

σ=1, 3x3

0.0751	0.1238	0.0751
0.1238	0.2042	0.1238
0.0751	0.1238	0.0751

σ=2, 3x3

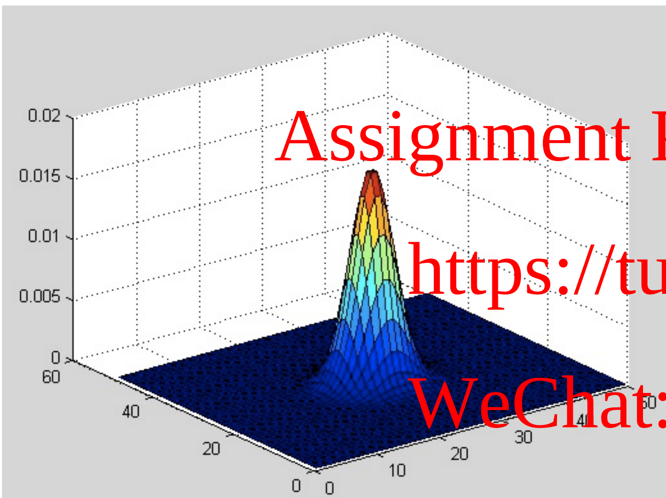
0.1019	0.1154	0.1019
0.1154	0.1308	0.1154
0.1019	0.1154	0.1019

σ=1, 5x5

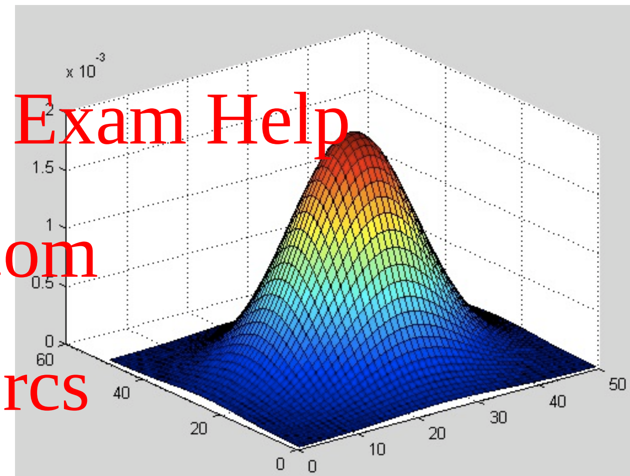
0.0030	0.0133	0.0219	0.0133	0.0030
0.0133	0.0596	0.0983	0.0596	0.0133
0.0219	0.0983	0.1621	0.0983	0.0219
0.0133	0.0596	0.0983	0.0596	0.0133
0.0030	0.0133	0.0219	0.0133	0.0030

Create and Visualise Gaussian Filters in Matlab

50x50 square, σ=3



σ=9



Create Gaussian filter in Matlab

a=5, sig =1;

g=fspecial('gaussian', [a,a], sig); % create a gaussian filter: a-by-a square, σ=1

surf (1:a, 1:a, g) % visualise it

Application: Filtering Using Gaussian

Original

5x5, σ=2

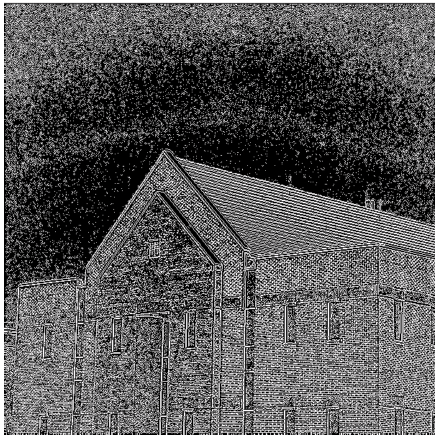
5x5, σ=4

10x10, σ=2

10x10, σ=4

Laplacian of Gaussian

- Filtering images with a Gaussian and then by the Laplacian (calculating the second derivatives) is called Laplacian of Gaussian (LoG).
- When zero crossing detection is applied to LoG, we have the so called Marr-Hildreth method.



Zero crossings without Gaussian filtering

Zero crossings with Gaussian filtering (13X13)

Zero crossings with Gaussian filtering (50X50)

Edge Detection by Marr-Hildreth Method

- Procedure:
  - Smooth by a Gaussian,
  - Filter by the Laplacian,
  - Threshold the 2nd derivatives,
  - Find zero-crossings:


```
i=imread('some_image.jpg');
filter_g=fspecial('gaussian', [5 5], 3); %create a Gaussian filter
im_g=filter2(filter_g, i)/255; % filter the image with Gaussian and scale it
filter_la=fspecial('laplacian',0); %create a Laplacian filter, where the
        %second parameter (alpha=0) control the shape of Laplacian
% detect edge from zero-cross using a threshold value
building_edge=edge(im_g, 'zerocross', threshold, filter_la);
imshow(building _edge)
```

Example

- In both cases on the right, a Gaussian filter of 13x13, sigma=2 are used

- Case (a): Threshold=0.01
- Case (b): Threshold=0.005

(a)

(b)

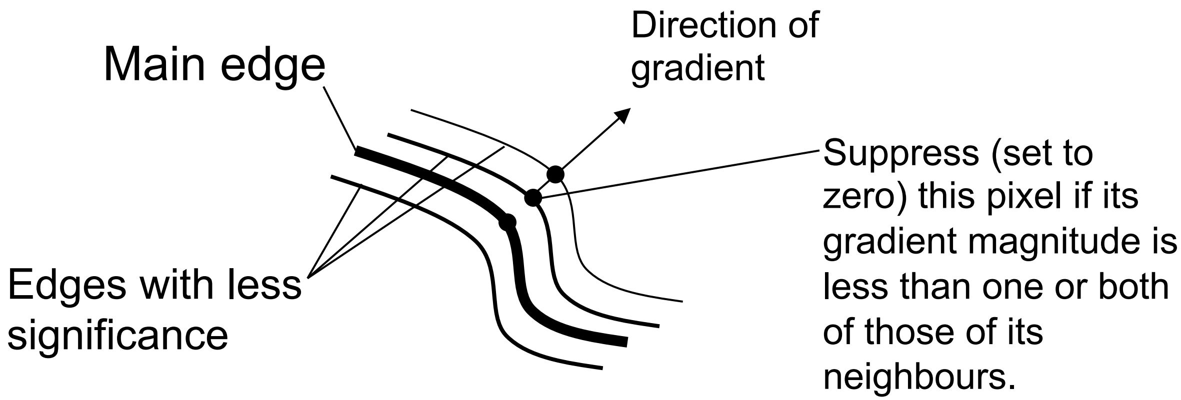
Canny Edge Detector

- Canny edge detector is one of the most popular edge detectors
- It detects edges by
  - (1) Smoothing image with Gaussian
  - (2) Then intensity gradient (1st derivative) *magnitude and direction* are computed at each pixel.
  - (3) *Nonmaximum suppression*
  - (4) *Linking edge*

- There is nothing new in Step 1 & 2. Steps 3 & 4 characterise the Canny detector.

Nonmaximum Suppression

- In this step, the Canny edge detector “thins” the wide ridges in the gradient/edge image obtained in the previous step
- Thinning is done by suppressing an edge pixel whose intensity is not higher than the two neighbouring pixels on either side of it along the direction of the gradient (i.e., edges that are not the local maxima) - hence that name of **Nonmaximum suppression**.



Linking Edge

- A single threshold does not work well when the intensity varies along an edge:
  - If the threshold is too high, some edge points are discarded (false negative). This may make an otherwise continuous edge broken.
  - If it is too low, non-edge points are included as edge (false positive)
- Canny detector traces and links edges using two thresholds: *high_thresh* and *low_thresh*.
  - First tracing an edge using the *high_thresh*.
  - If edge becomes broken then check if any points above the *low_thresh* can link them together. If so, continue tracing using the *low_thresh* until revert to the *high_thresh.*

Implementation & Example

- Canny detector is implemented in function **edge** in IPT

>>BW =edge(grayimage, 'canny', [0.08, 0.2])

low and high thresholds

Readings

- Shapiro, L.G., Stockman, G.C., Computer Vision, Prentice-Hall, 2001, ISBN 0-13-030796-3
  – Section 5.1 to 5.8