

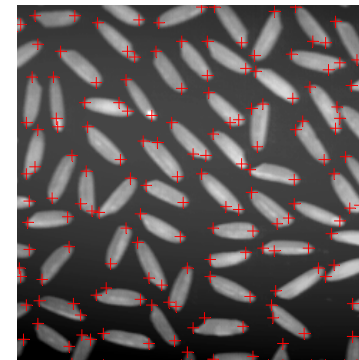
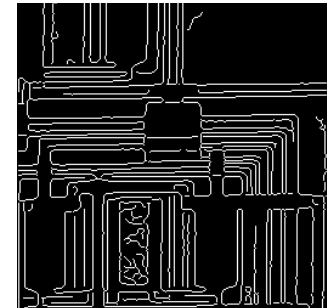
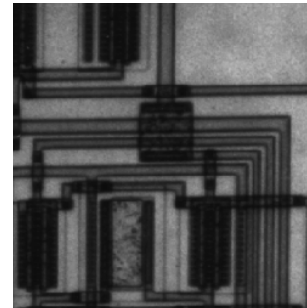
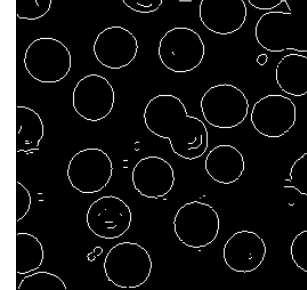
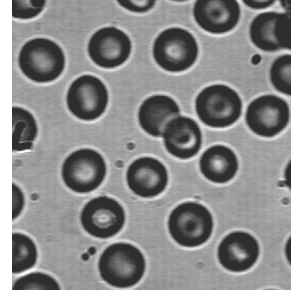
# Image Features

Course: Computer Vision

Lecture by: Maneesh Dewan

# Image Features

- What are image features?
- Parts/Regions of the image that are
  - Local, in general
  - Meaningful
    - Intensity variations
    - Uniform intensity/texture regions
  - Detectable
- Usually associated with feature descriptors
  - Eg: circle – radius and center.
  - Line – end point locations



Feature extraction usually an intermediate step

# Features that we'll cover

- Derivative Operators – Edge detection
- The Canny Edge Detector
- Corners
- Laplacian Pyramid
- SIFT Features

# Quick Summary on Convolution

$$R(i, j) = \sum_{h=-\infty}^{\infty} \sum_{k=-\infty}^{\infty} T(h, k) I(i-h, j-k)$$

T1	T2	T3
T4	T5	T6
T7	T8	T9

kernel

\*

I1	I2	I3
I4	I5	I6
I7	I8	I9

Image

=

R5
----

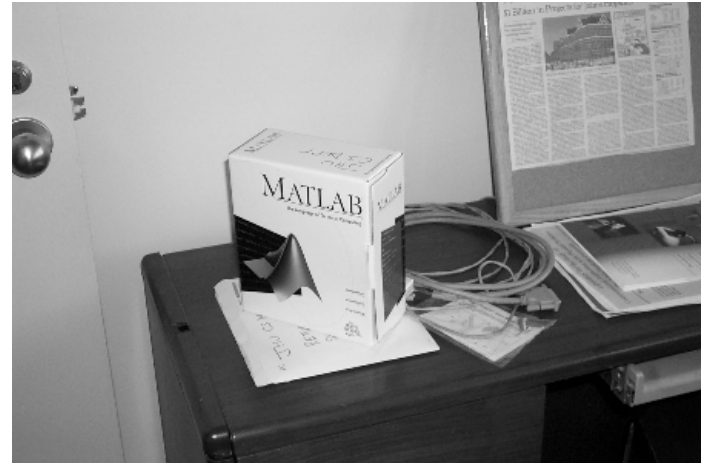
Filtered Image

$$\begin{aligned} R5 = & T9 \times I1 + T8 \times I2 + T7 \times I3 \\ & + T6 \times I4 + T5 \times I5 + T4 \times I6 \\ & + T3 \times I7 + T2 \times I8 + T1 \times I9 \end{aligned}$$

# What Else Can You Do With Convolution?

- Thus far, we've only considered convolution kernels that are smoothing filters.
- Consider the following kernel:

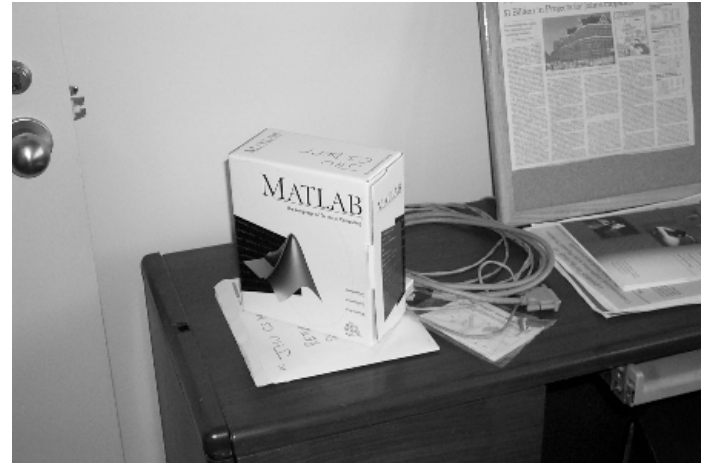
$$\begin{bmatrix} 1 & -1 \end{bmatrix}$$



# What Else Can You Do With Convolution?

- Now Consider the following kernel:

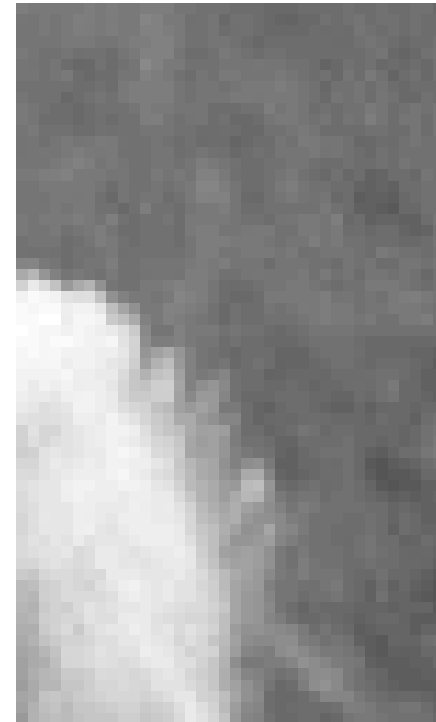
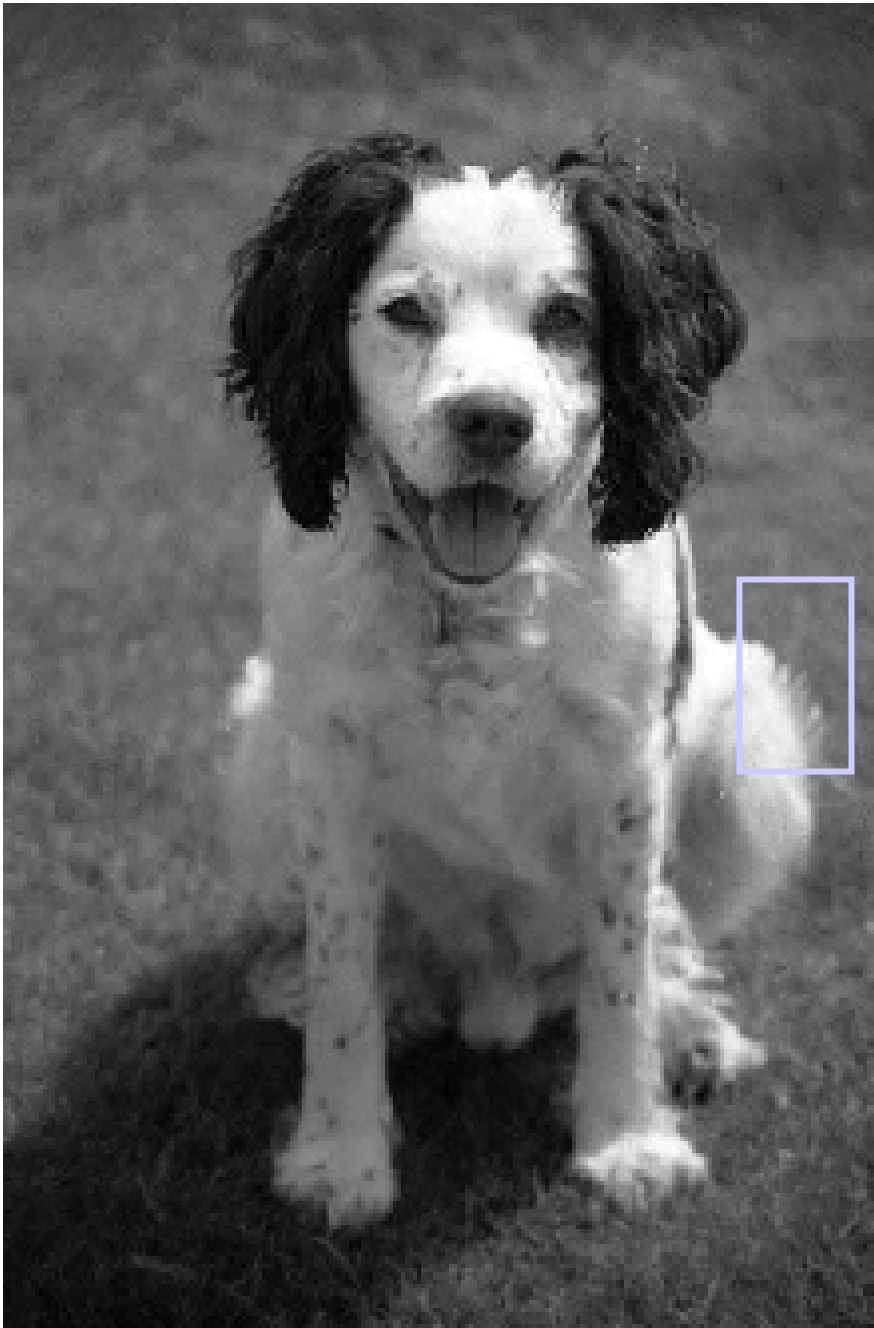
$$\begin{bmatrix} 1 \\ -1 \end{bmatrix}$$



# Physical causes of edges

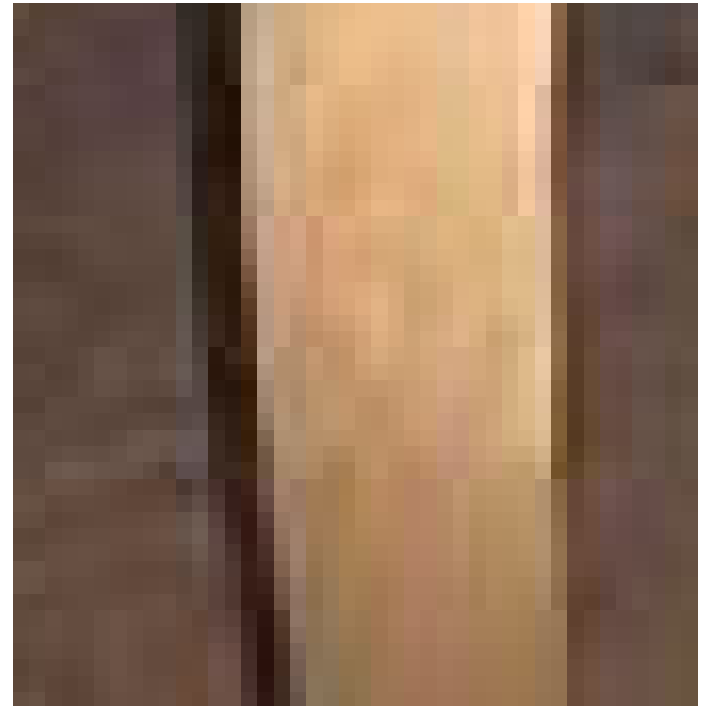
1. Object boundaries
2. Surface normal discontinuities
3. Reflectance (albedo) discontinuities
4. Lighting discontinuities

# Object Boundaries

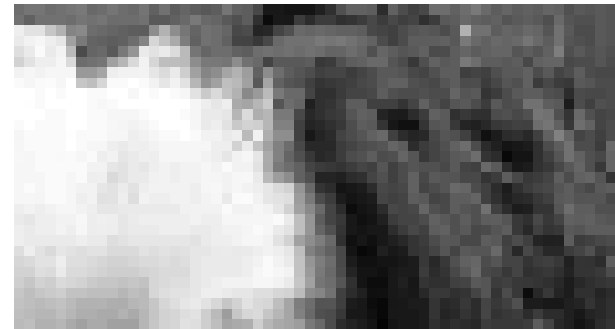
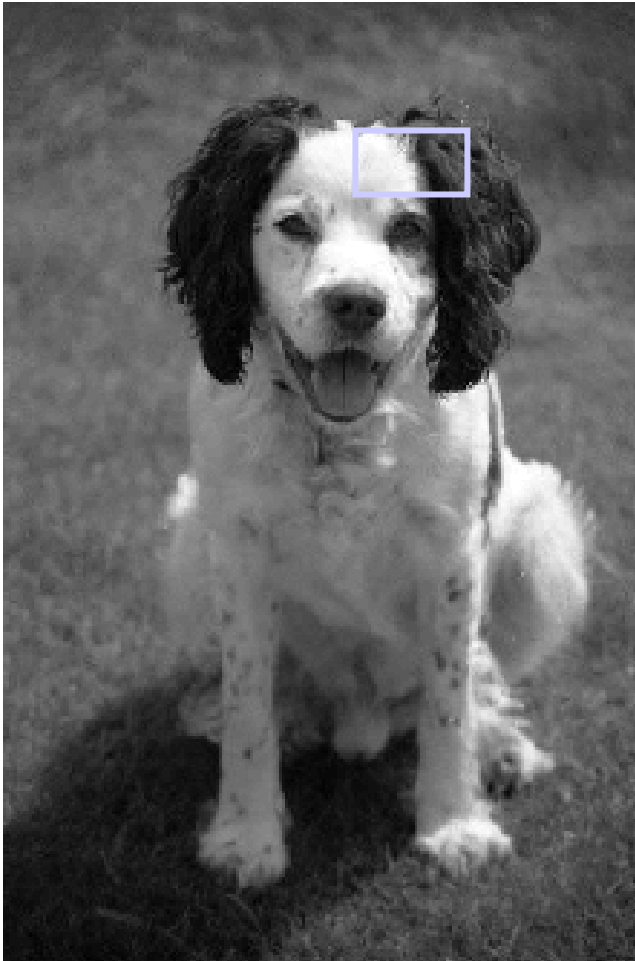




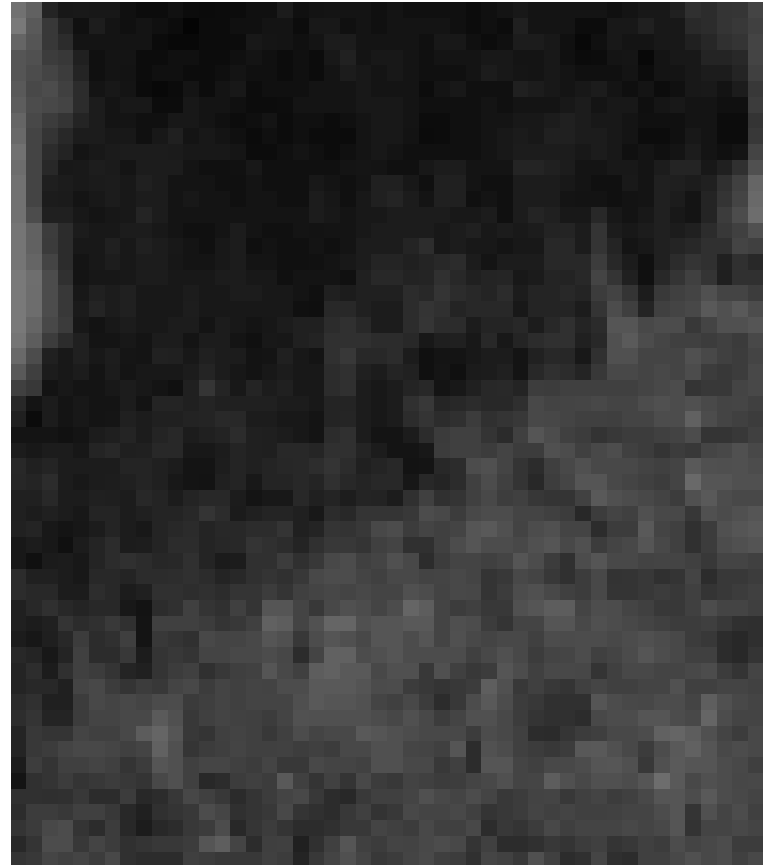
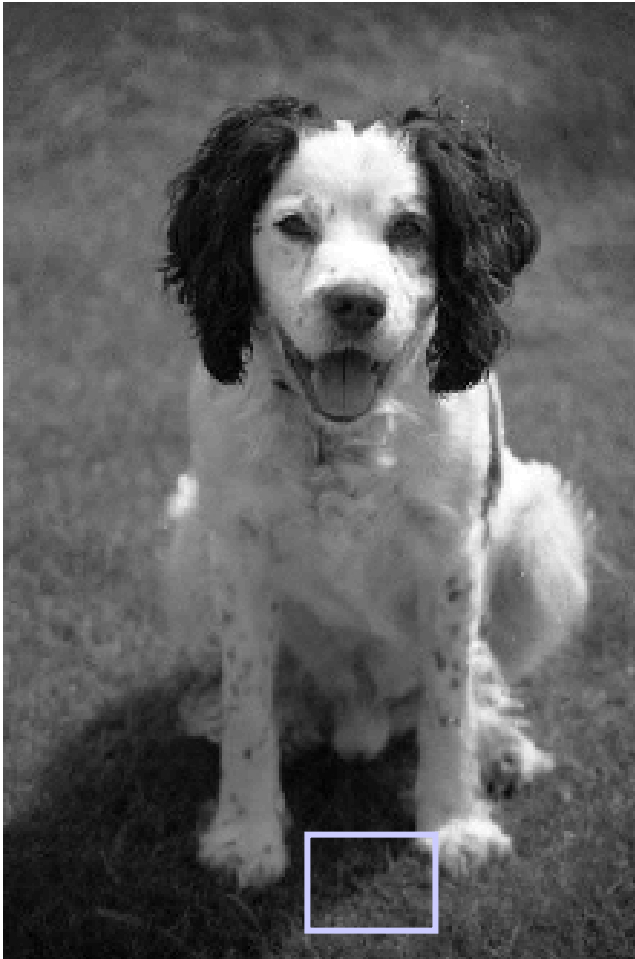
# Surface normal discontinuities



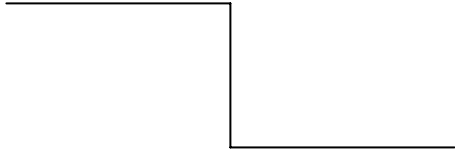
# Boundaries of material properties



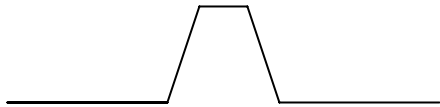
# Boundaries of lighting



# Edge Types



Step



Ridge

usually most  
common ones



Roof

Which of these do you suppose  
a derivative filter detects best?

# The Image Gradient

- Recall from calculus for a function of two variables  $f(x,y)$  that the gradient of the function  $f$  is given by :

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

- The kernel  $[1,-1]$  is a way of computing the x derivative
- The kernel  $[1;-1]$  is a way of computing the y derivative

# Central Finite Difference Approximation

- The 2 kernels  $[1, -1]$  and  $[1, -1]^T$  are forward finite difference approximation
- Better way of computing derivatives is through central finite difference approximation (why?)
- The kernels using central finite difference are:
  - Kernel for x-derivative -  $[1/2, 0, -1/2]$
  - Kernel for y-derivative -  $[1/2, 0, -1/2]^T$

# Some Other Interesting Derivative Kernels

The Roberts Operator

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

The Sobel Operator

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

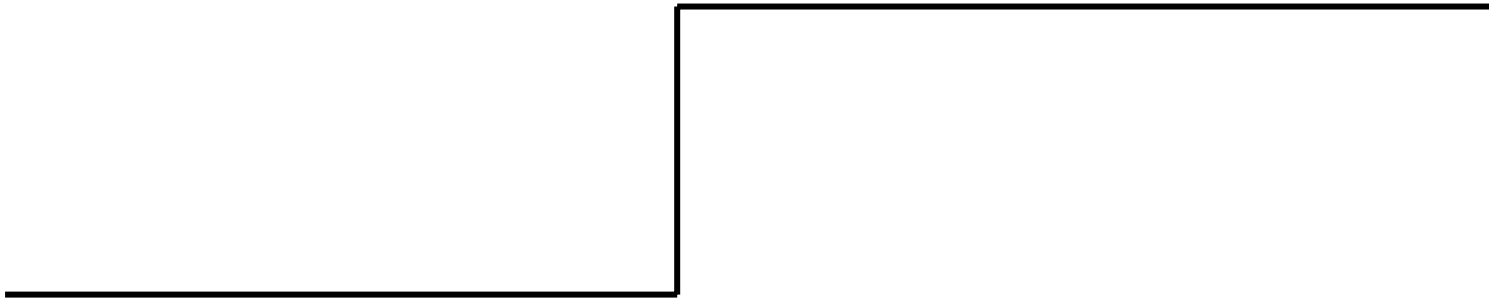
The Prewitt Operator

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 \\ -1 & 0 & 1 \\ -1 & -1 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

# Edge is Where Change Occurs

## 1-D

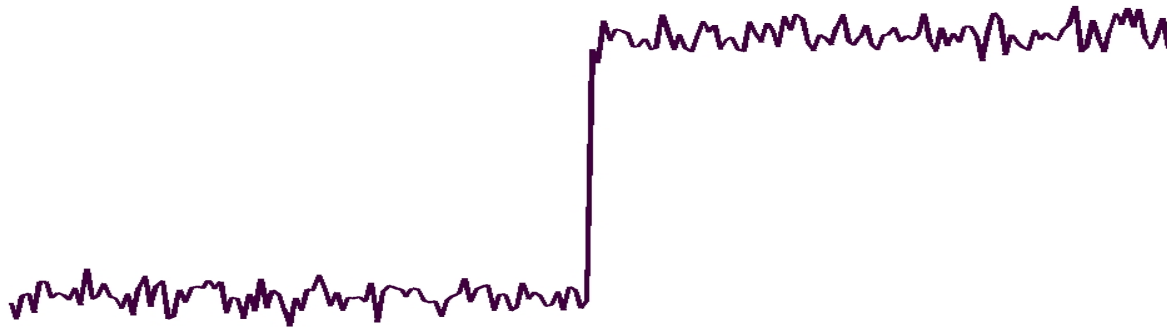
- Change is measured by derivative in 1D
  - Biggest change, derivative has maximum magnitude
  - Or  $2^{\text{nd}}$  derivative is zero.





# Noisy Step Edge

- Derivative is high everywhere.
- Must smooth before taking gradient.



# Smoothing Plus Derivatives

- One problem with differences is that they by definition reduce the signal to noise ratio (can you show this?)
- Recall smoothing operators (the Gaussian!) reduce noise.
- Hence, an obvious way of getting clean images with derivatives is to combine derivative filtering and smoothing: e.g.

$$- D_x * (G * I) = (D_x * G) * I \quad \text{(convolution is associative)}$$

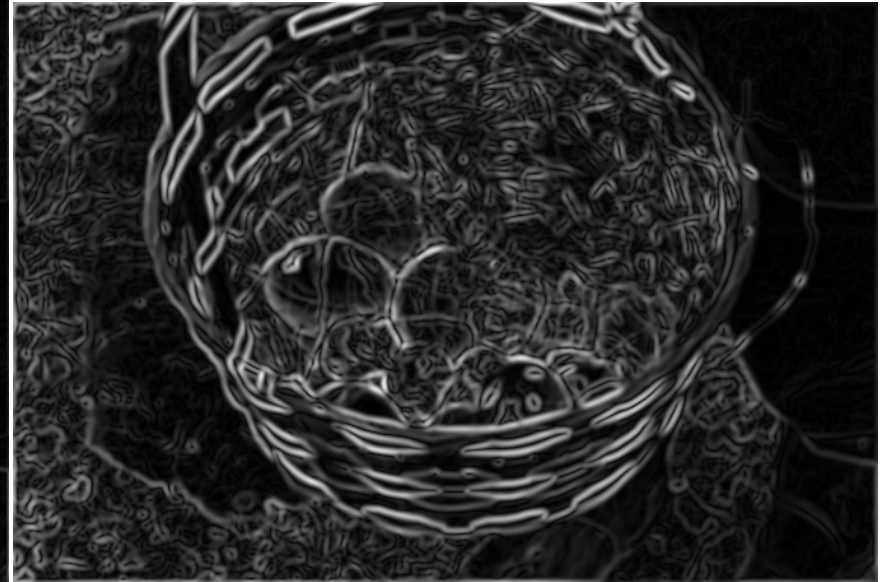
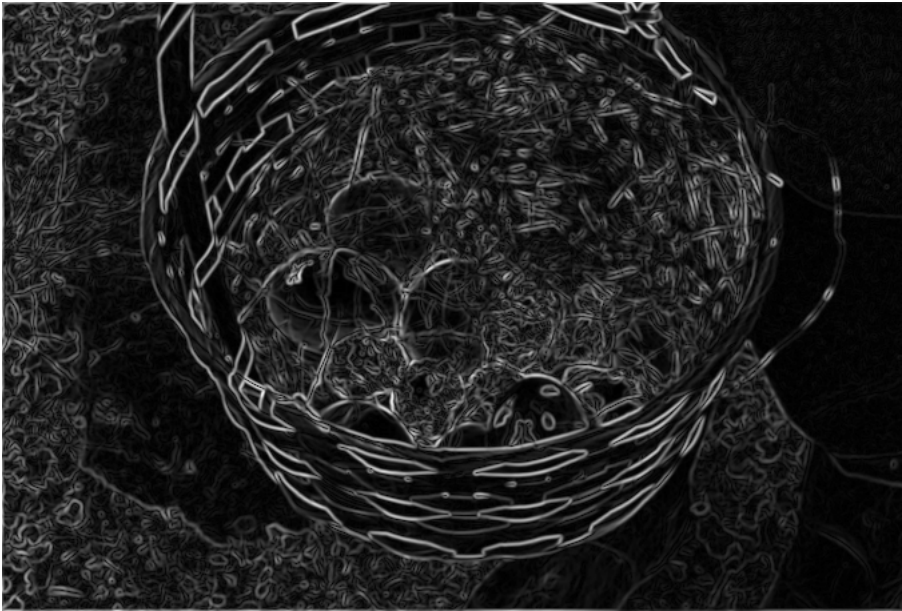
# Algorithm: Simple Edge Detection

- 1. Compute  $I_x = I_g * (G(\sigma) * G(\sigma)' * [1, -1; 1, -1])$
- 2. Compute  $I_y = I_g * (G(\sigma) * G(\sigma)' * [1, -1; 1, -1]')$
- 3. Compute  $I_{mag} = \text{sqrt}(I_x.^* I_x + I_y.^* I_y)$
- 4. Threshold:  $I_{res} = I_{mag} > \tau$
- It is interesting to note that if we wanted an edge detector for a specific direction of edges, we can simply choose the appropriate projection (weighting) of the component derivatives.

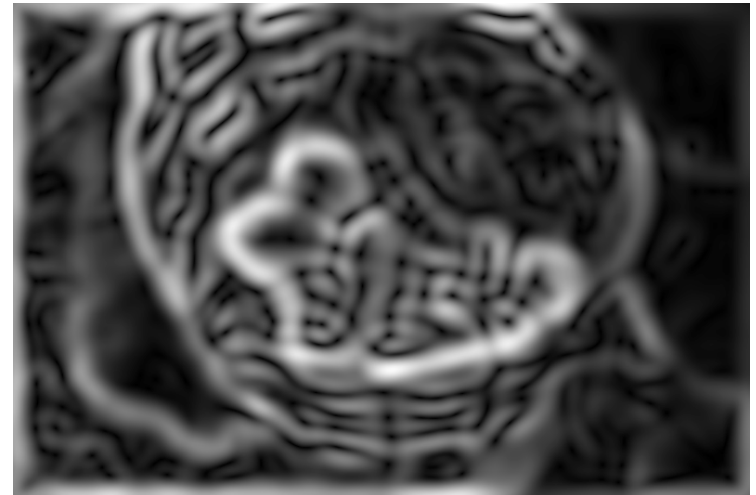
$\sigma = 1$

# Example

$\sigma = 2$



$\sigma = 5$



$\sigma = 10$

# Laplacian

- We know from calculus that in 1 D, second derivative is zero when the derivative magnitude is an extremum
- Rotationally invariant Laplacian – extension to 2D

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

The Laplacian Operator

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \text{ or } \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

A good exercise: derive the Laplacian from 1-D derivative filters.

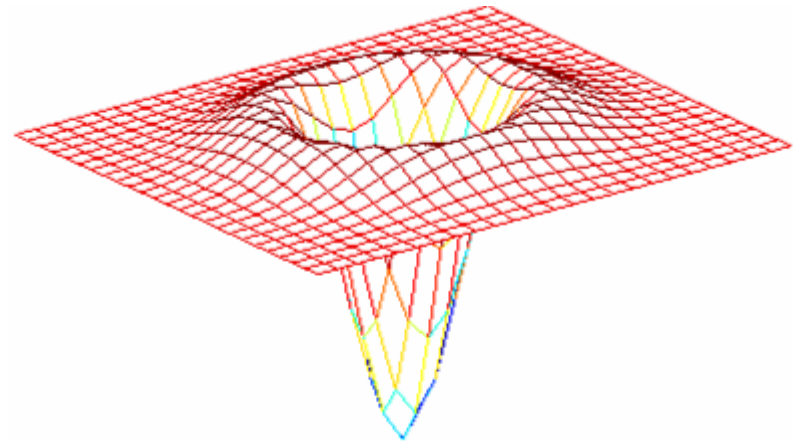
Try with both forward and central difference 1-D kernels

# Laplacian of a Gaussian (LOG)

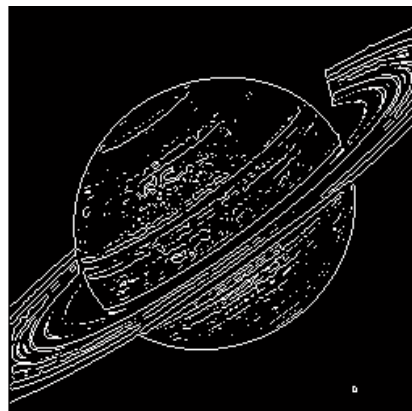
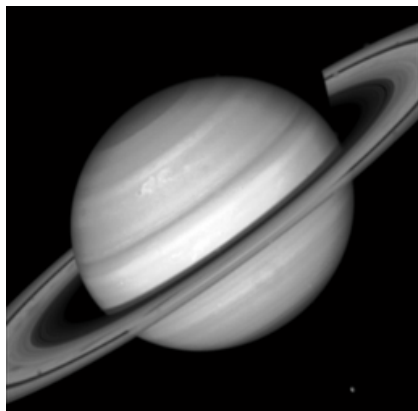
- Taking second derivatives so one should definitely smooth the image like earlier
- As convolution is associative we have

$$(K_{\nabla^2} ** (G_{\sigma} ** I)) = (K_{\nabla^2} ** G_{\sigma}) ** I = \underbrace{(\nabla^2 G)}_{\text{LOG}} ** I$$

- After applying LOG at a particular scale, mark zero points where the gradient magnitude is large



# LOG examples



sigma = 1



sigma = 2



sigma = 3

- Also used for sharpening the edges – response positive on one side of edge and negative on the other
- Drawbacks
  - Poor performance at the corners (filter not oriented – average response)

# From Pixels to Edges

- 3 steps of edge detection
  - Noise reduction or smoothing (discussed in last lecture)
  - Edge enhancement - *enhance* rapid contrast changes and suppress everything else (various operators (discussed earlier) can be used)
  - Edge localization - Detecting these contrast changes (involves *thresholding* to separate noise from signal)
- *Edges* are a result of *grouping* pixels (sometimes called “edgels”) into groups forming continuous curves.



# From Pixels to Edges

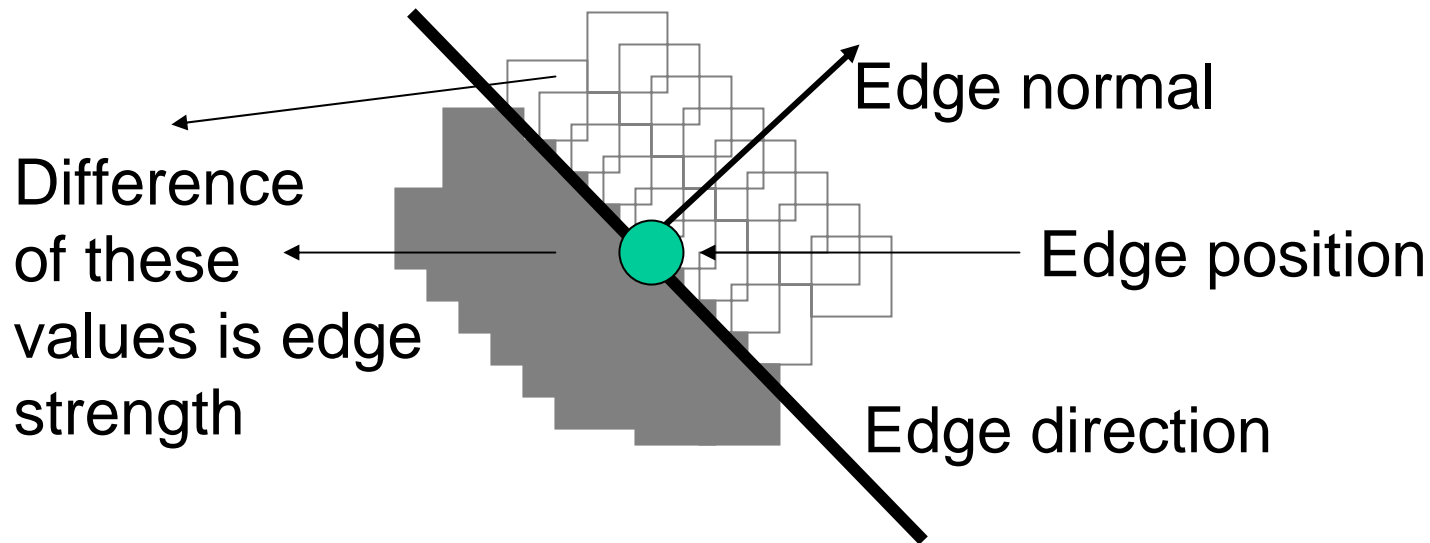
Definitions:

*Edge normal:* Unit vector in direction of maximum intensity variation

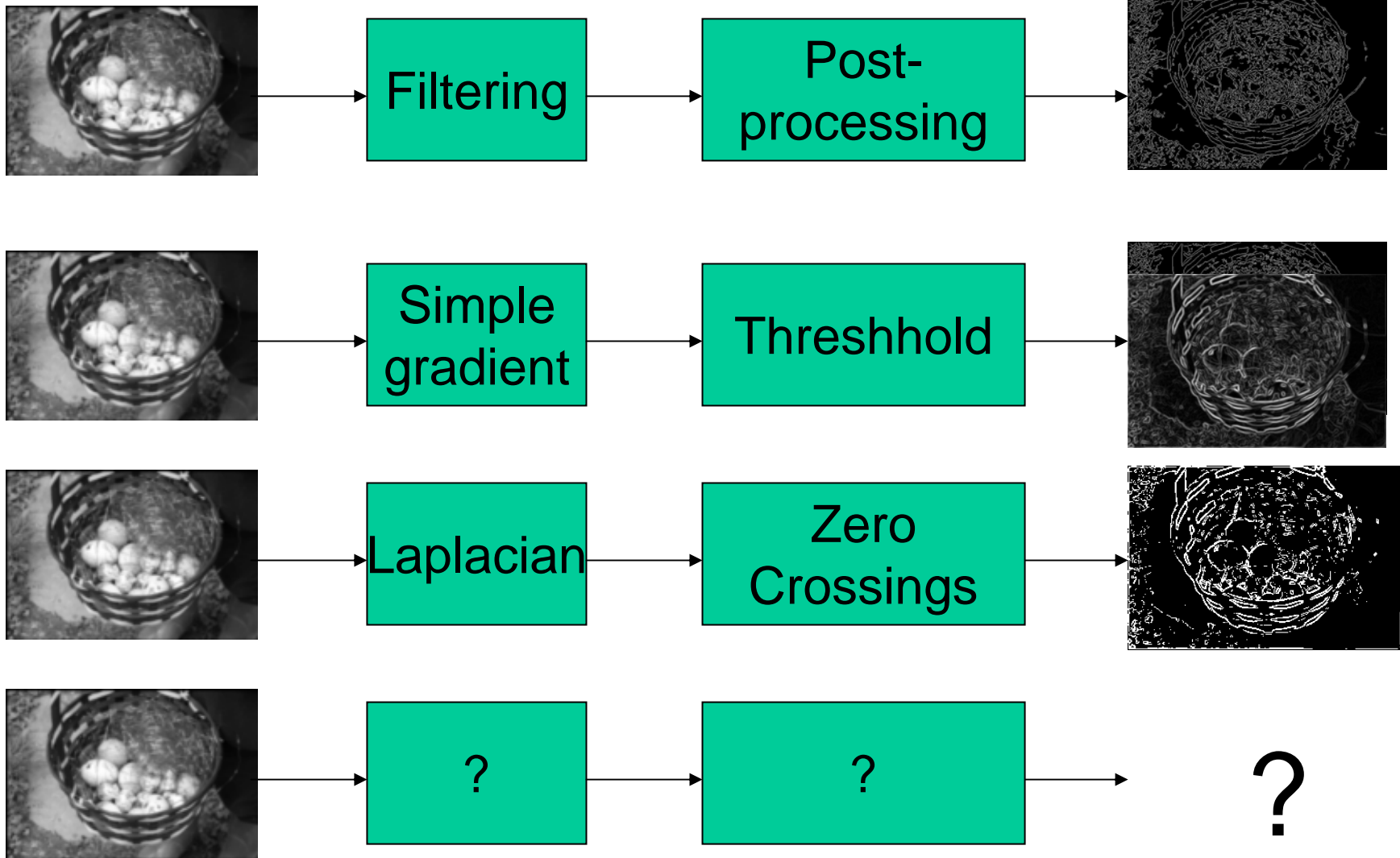
*Edge direction:* Perpendicular to edge normal

*Edge position:* Image position of pixels of edge

*Edge strength:* Change in contrast along normal



# Edges: The Problem



What is optimal?

# Canny Edge Detector

- The steps involved:
  - Formulate an optimization problem for detection on 1-D signals
  - Generalize to 2D signals
  - Apply thresholding with hysteresis
  - Apply this operator at various scales
- The Assumptions:
  - Edge enhancement filter is linear
  - The edge model is step edges with amplitude  $A$
  - Noise is additive, white and Gaussian

# Optimization Criteria

- **Good Detection:** *Minimize the probability of false positives and false negatives*

- Maximize the SNR

$$\frac{A}{n_0} \Sigma(f) = \left| \frac{A \int_{-W}^0 f(t) dt}{n_0 \sqrt{\int_{-W}^W f^2(t) dt}} \right|$$

- **Good Localization:** *Edgels detected should lie as close as possible to the true edge*

- Maximize 1/distance to edge center which leads to maximizing LOC

$$\frac{A}{n_0} \Lambda(f') = \left| \frac{A |f'(0)|}{n_0 \sqrt{\int_{-W}^W f'^2(t) dt}} \right|$$

# Optimization Cont'd

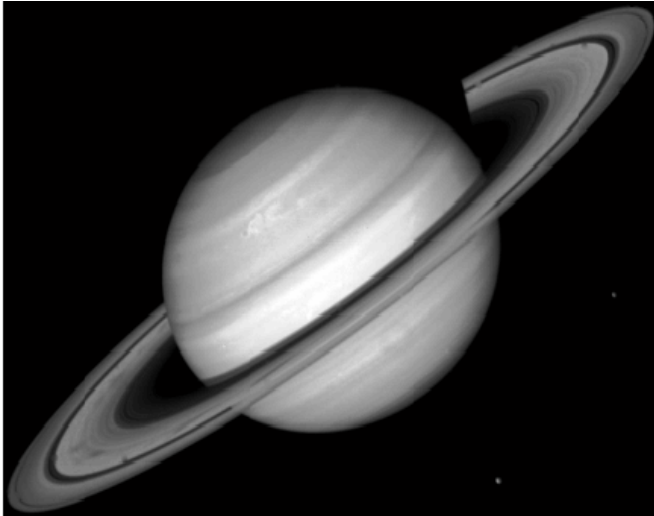
- Consider the maximizing the product of both criteria
  - result is itself a simple difference operator
  - Difference operators are noise amplifying!
- Additional criterion: single response constraint:
  - detector should minimize the number of local maxima around the true edge created by noise
  - **RESULT1:** localization vs. detection tradeoff

$$\Sigma(f_w) = \sqrt{w} \Sigma(f) \text{ and } \Lambda(f'_w) = \frac{1}{\sqrt{w}} \Lambda(f') \text{ where } f_w(x) = f(x/w)$$

- Larger filter improves detection but worsens localization by the same amount.
- **RESULT2:** *optimal detector is very close to the first derivative of a Gaussian (DoG).*

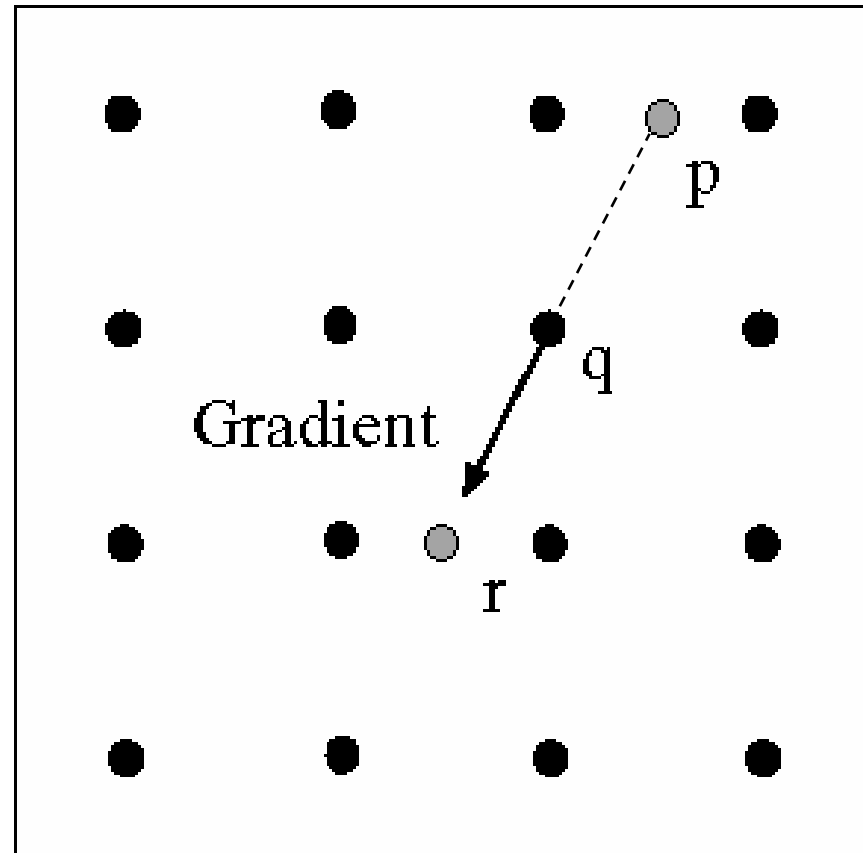
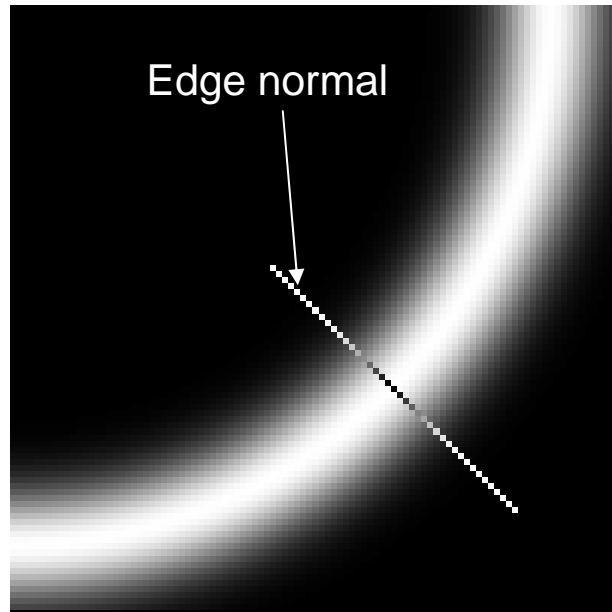
# The Procedure

- First Step - Enhancement:
  - compute x and y derivatives using DoG's.
  - compute direction and magnitude of gradient (two images)



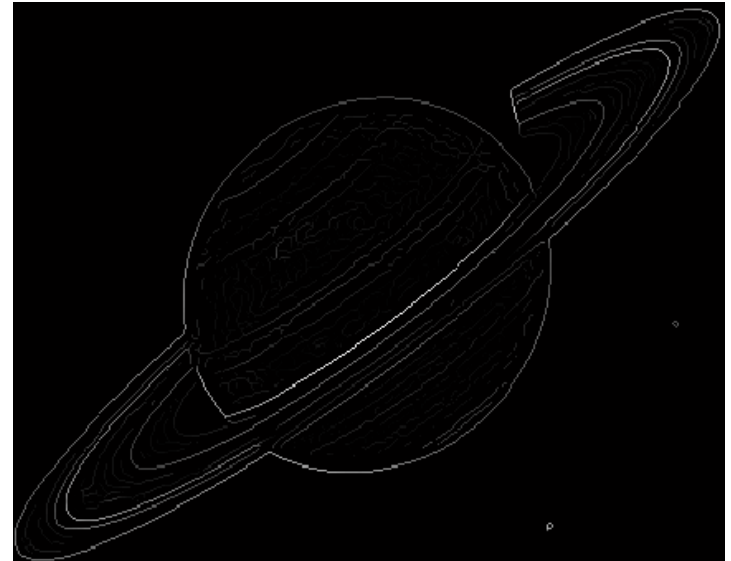
# Nonmaximal Suppression

- Sample along the gradient direction
- If given pixel is smaller than neighbor, set it to zero



Images taken from Marc Pollefeys lecture

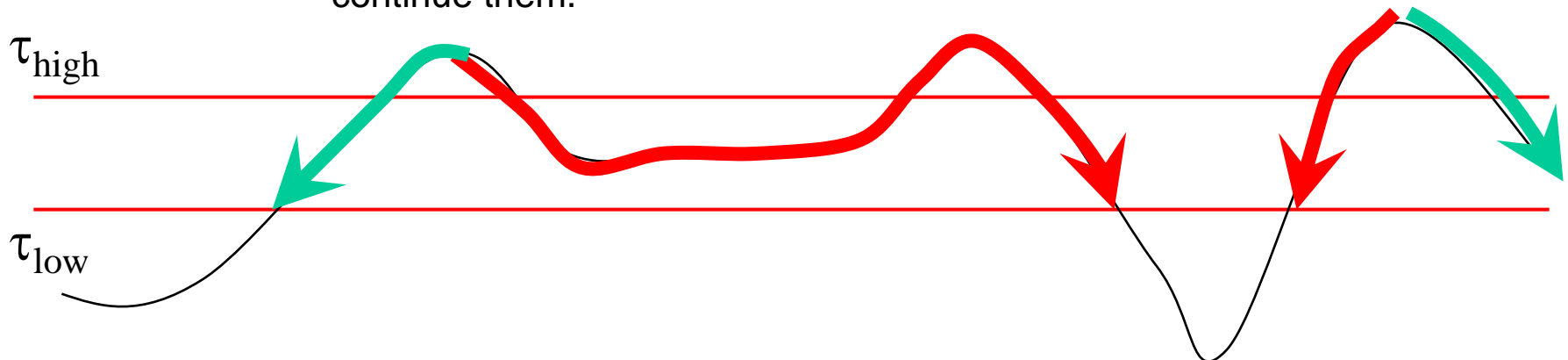
# Nonmaximal Suppression



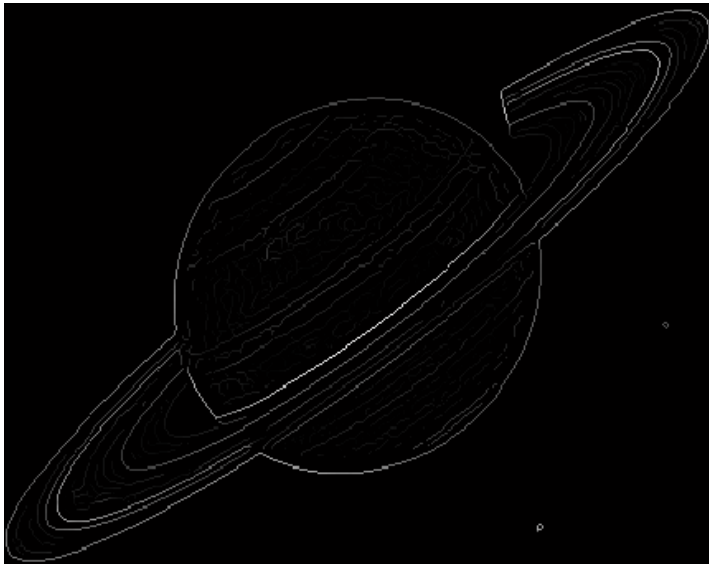


# Hysteresis

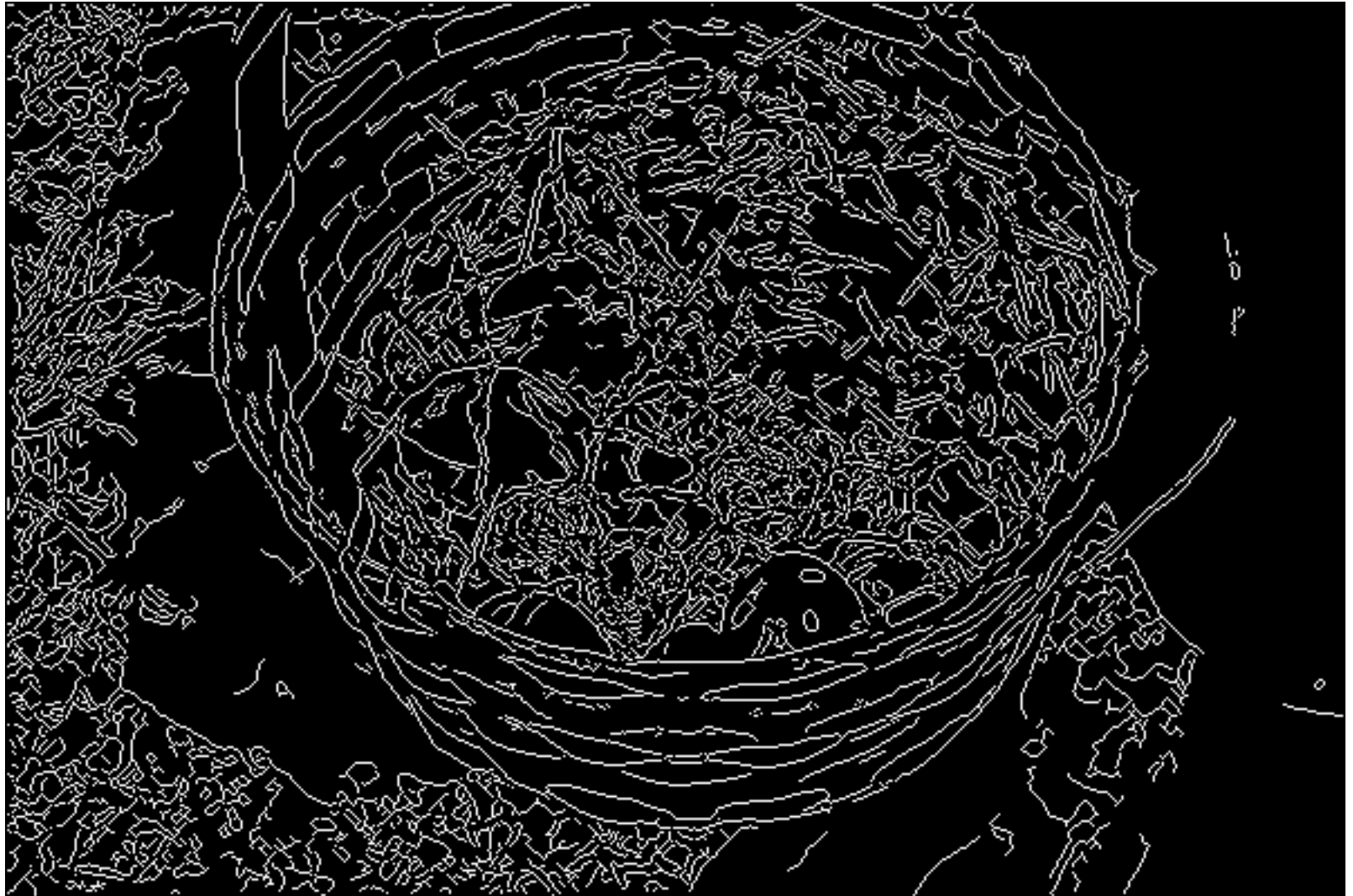
- Problems with simple thresholding
  - False contours - if low threshold is set to capture true but weak edges
  - Streaking – values of connected contours may fluctuate above and below the threshold
- Hysteresis thresholding
  - Track edge points by starting at point where gradient magnitude  $> \tau_{\text{high}}$ .
  - Follow edge in direction orthogonal to gradient.
  - Stop when gradient magnitude  $< \tau_{\text{low}}$ .
    - i.e., use a high threshold to start edge curves and a low threshold to continue them.



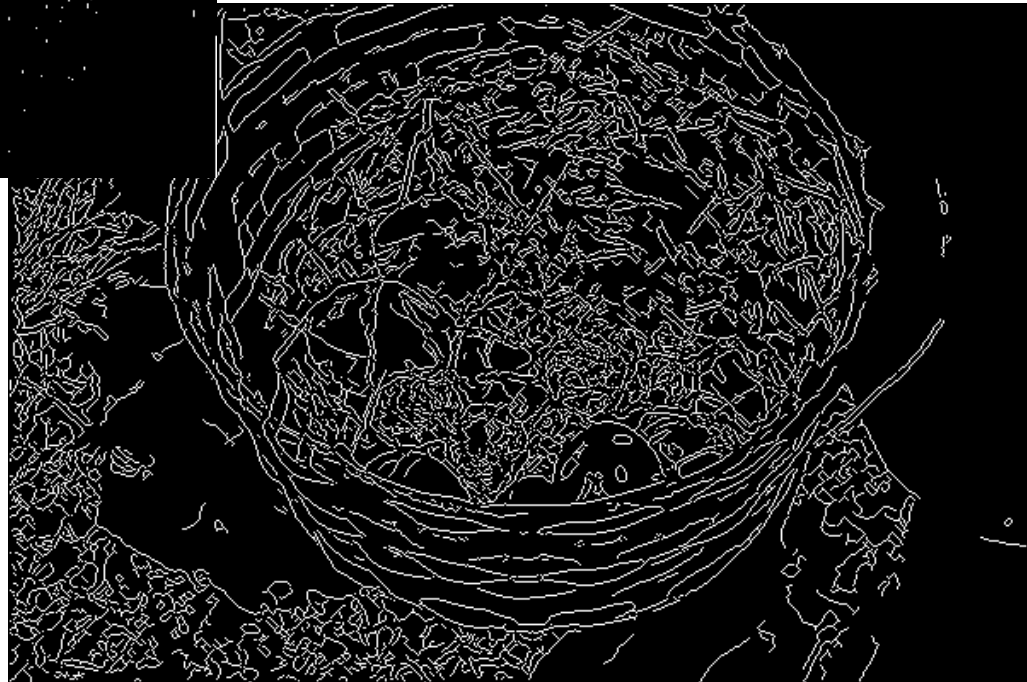
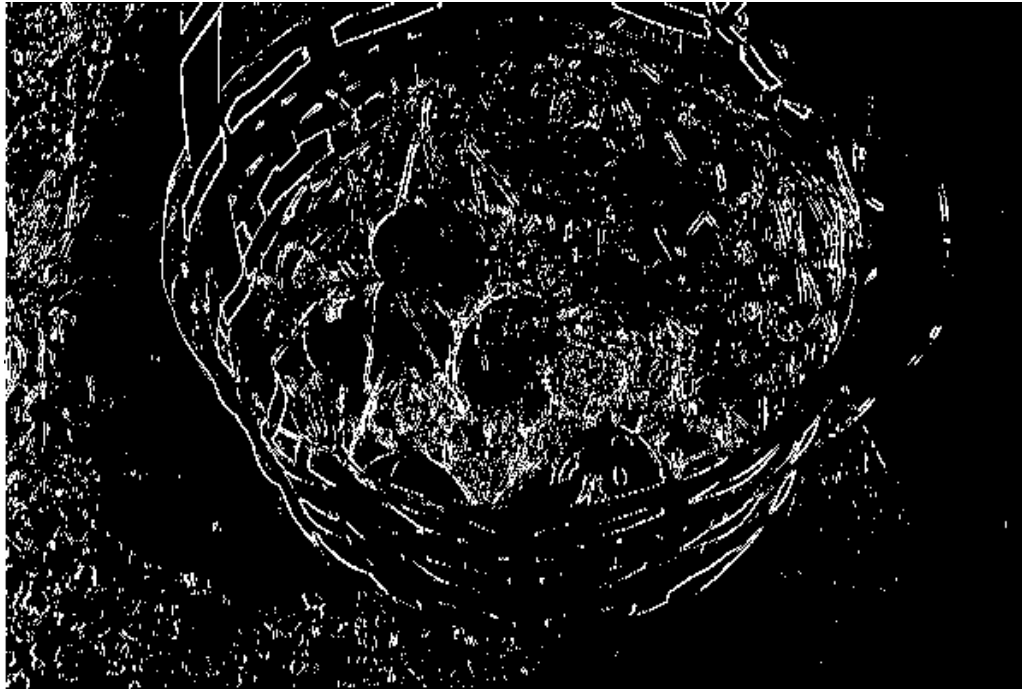
# Hysteresis Thresholding



# Canny Output

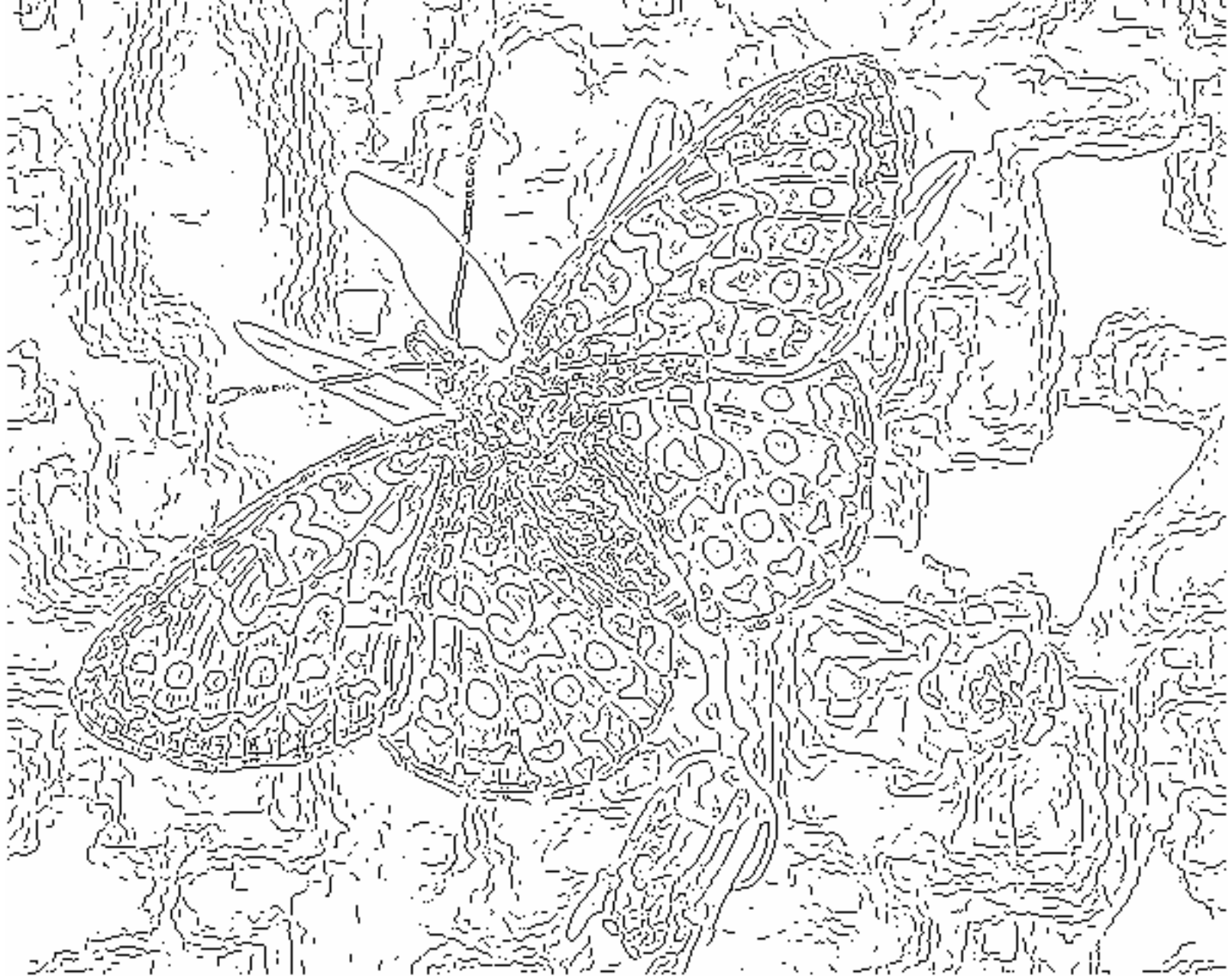


# Canny Comparison





Lets do edge detection & play with some parameters



fine scale, high threshold



coarse scale, high threshold



coarse scale, low threshold



# Why is Canny so Dominant

Still widely used after 20 years.

1. Theory is nice (but end result same,).
2. Details good (magnitude of gradient, non-max suppression).
3. Hysteresis an important heuristic.
4. Code was distributed.

# Summary: Types of Edge Detection Operators

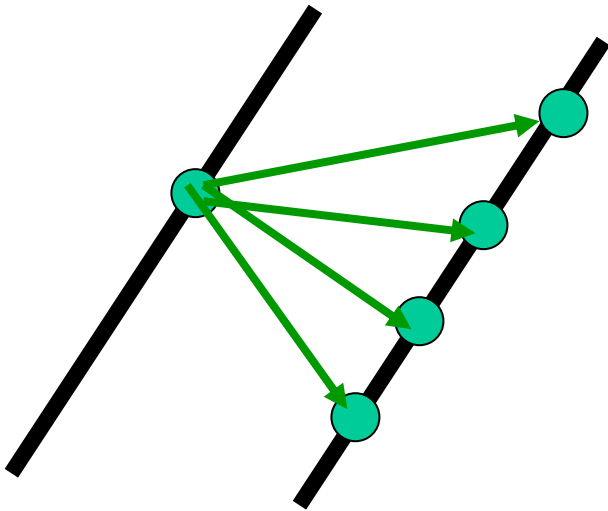
1. Operators approximating derivatives using differences.
  - directional: Roberts, Prewitt, DoG, etc.
  - Rotationally invariant: Laplacian (sum of second derivatives)
2. Operators based on the zero crossing of the second derivative (e.g. Canny).

Do Edge Demo in MATLAB

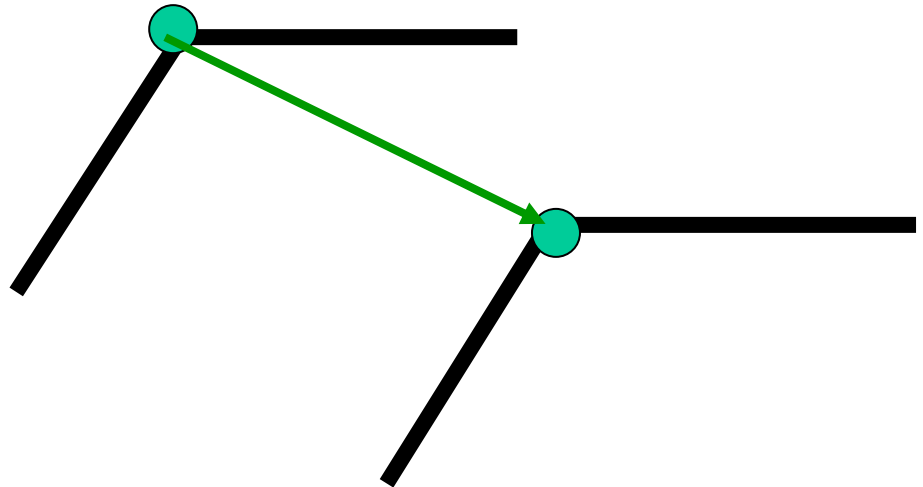
# Corners

- Corners contain more info than lines.

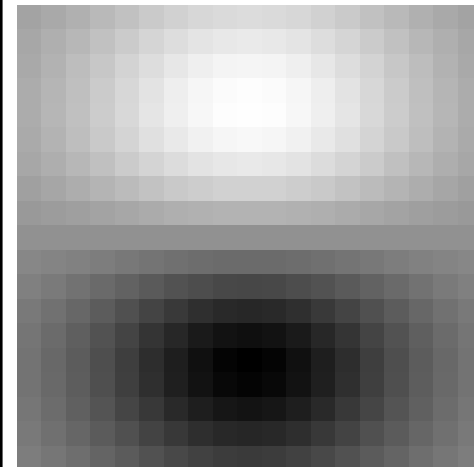
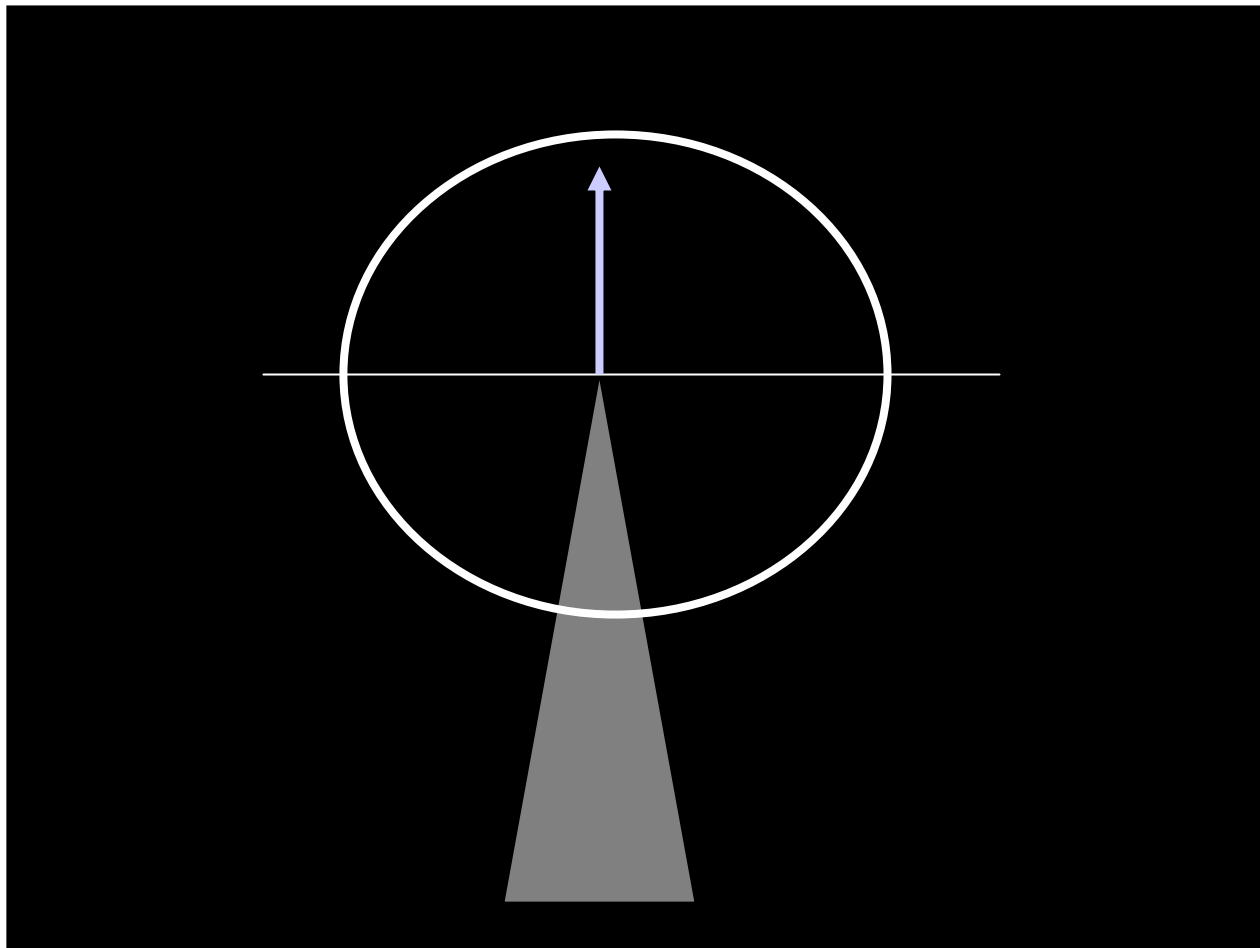
A point on a line is hard to match.



A corner is easier to match



# Edge Detectors Tend to Fail at Corners



difference of  
gaussian

# Finding Corners

Intuition:

- Right at corner, gradient is ill defined.
- Near corner, gradient has two different values.

# Formula for Finding Corners

We look at matrix:

Sum over a small region,  
the hypothetical corner

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

Gradient with respect to x,  
times gradient with respect to y

$$\begin{bmatrix} \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

Matrix is symmetric

**WHY THIS?**

First, consider case where:

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

This means all gradients in neighborhood are:

(k,0) or (0, c) or (0, 0) (or off-diagonals cancel).

What is region like if:

1.  $\lambda_1 = 0$ ?
2.  $\lambda_2 = 0$ ?
3.  $\lambda_1 = 0$  and  $\lambda_2 = 0$ ?
4.  $\lambda_1 > 0$  and  $\lambda_2 > 0$ ?

# General Case

From Linear Algebra we haven't talked about it follows that since  $C$  is symmetric:

$$C = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

So every case is like one on last slide.



# Corners Algorithm

- Compute image gradient over the entire image  $I$
- For every point  $(u,v)$  in image  $I$ , compute  $C$  for a neighborhood  $(N \times N)$  about  $(u,v)$
- Sort by the minimum singular value of  $C$
- Read off locations starting with highest values and working down until enough locations are found or we run out of locations
  - optional: as we read down, discard corners that are within a small distance of corners that have appeared higher in the list
- Issues with corner detection
  - Selection of neighborhood size
    - No simple criterion (usually  $N = 2$  to  $10$  is good in practice)
  - Threshold value
    - One good way is look for valley in histogram of minimum singular values

# Other Measures - Harris Corner

$$1) P = \det M - k(\text{trace}(M))^2$$

$$\det M = \lambda_1 \lambda_2$$

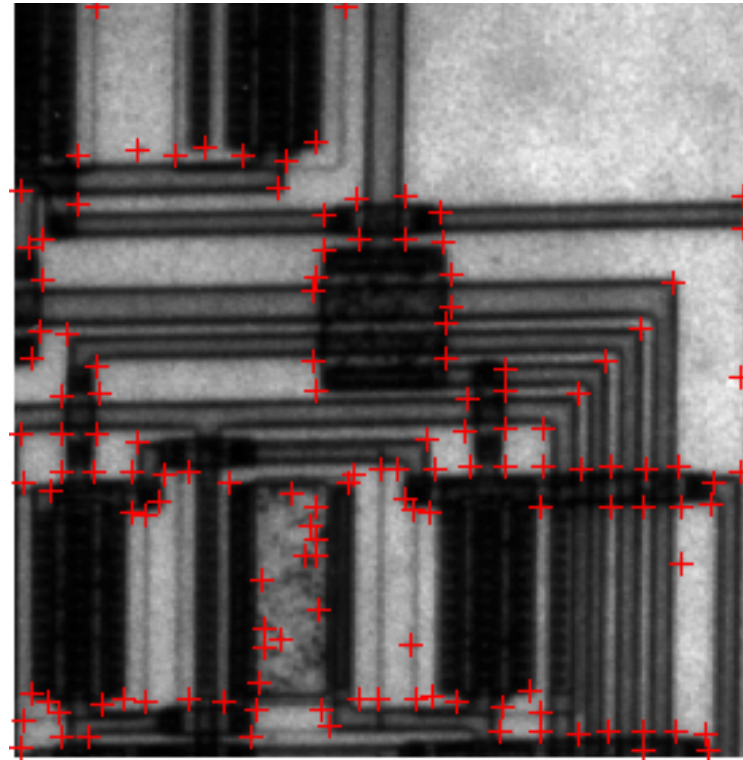
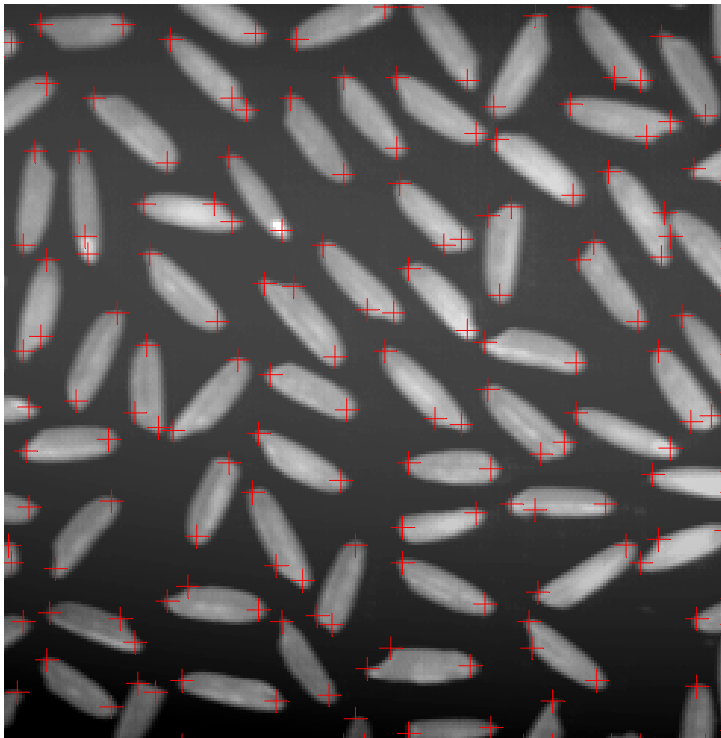
$$2) P = \det M / \text{trace}(M)$$

$$\text{trace } M = \lambda_1 + \lambda_2$$

( $k$  – empirical constant,  $k = 0.04$ - $0.06$ )

Usually one runs a nonmaximal suppression to get the peak of the response

# Corner detection



# Thresholding Issue



Increasing threshold



# Notion of Invariance

- Property of a feature descriptor (in this specific case) to be tolerant (for detection) to a particular set of transformations
  - Rotation
  - Translation
  - Scale
  - Intensity
  - Affine
  - Illumination
  - and others...
- Which ones of these do you think the harris corner detector is invariant to?

# Filter Pyramids



512

256

128

64

32

16

8



A bar in the big images is a hair on the zebra's nose; in smaller images, a stripe; in the smallest, the animal's nose

Image taken from forsyth and ponce book

# Constructing pyramids by subsampling

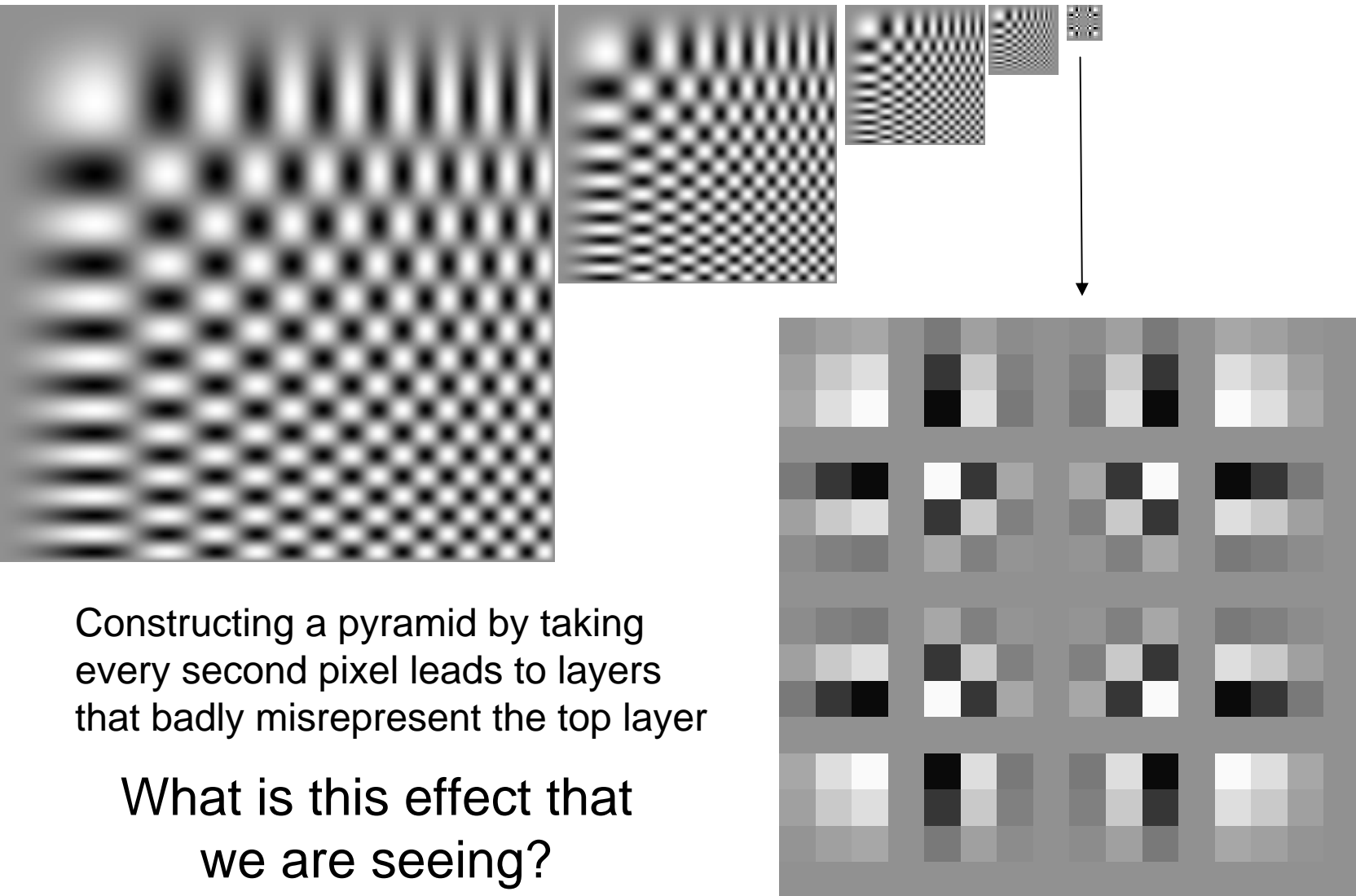
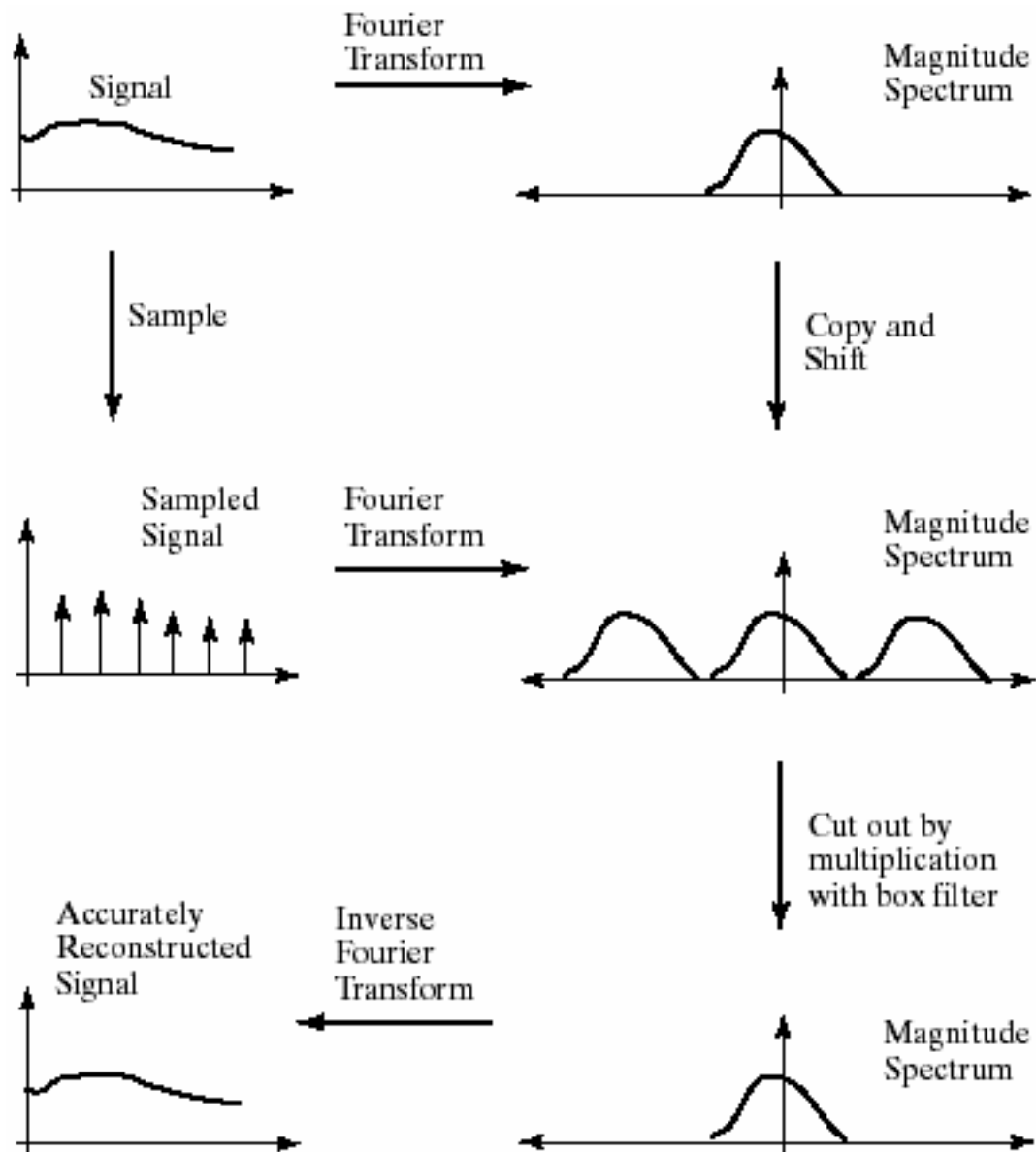


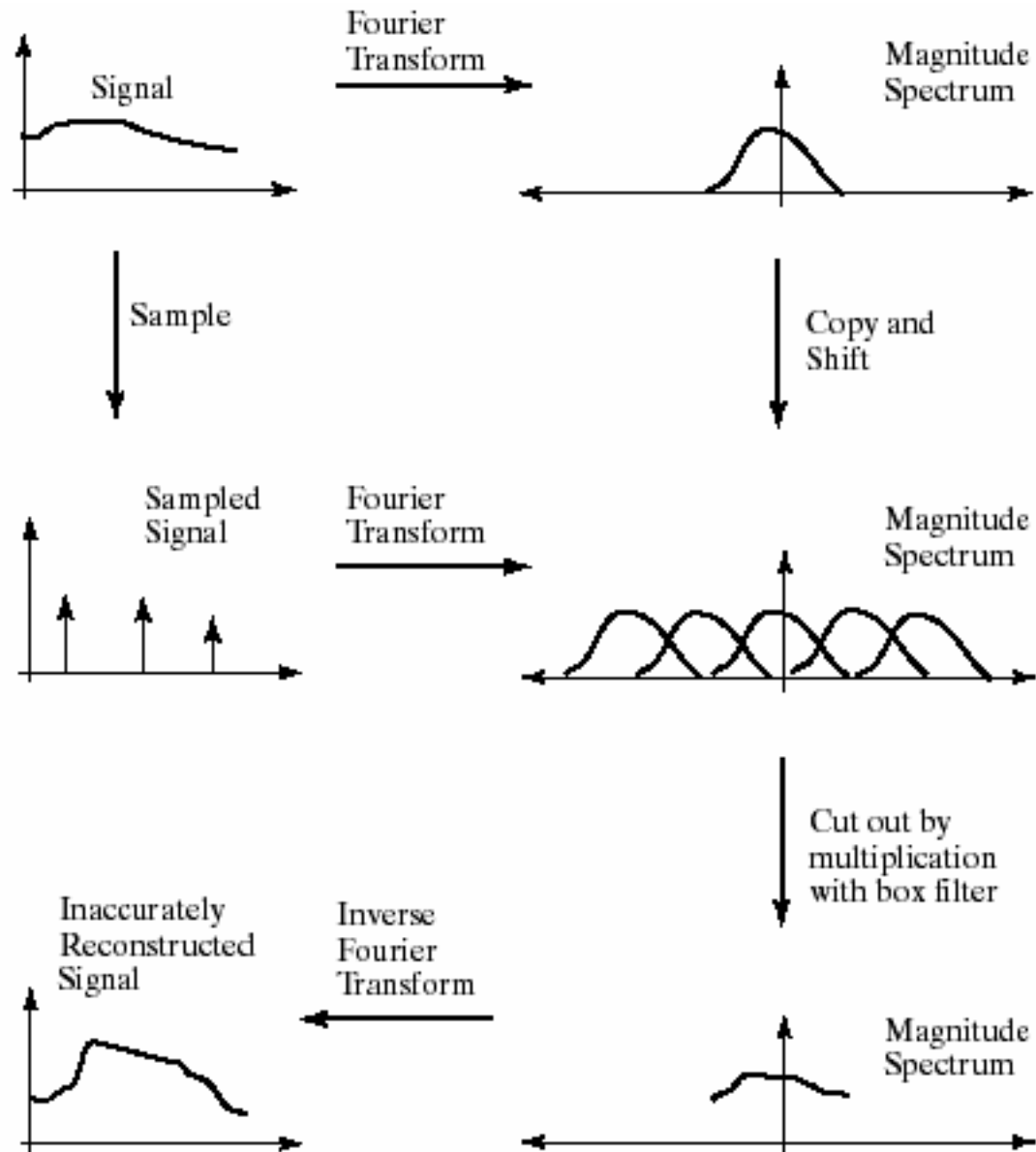
Image taken from forsyth and ponce lecture

# Sampling and reconstruction



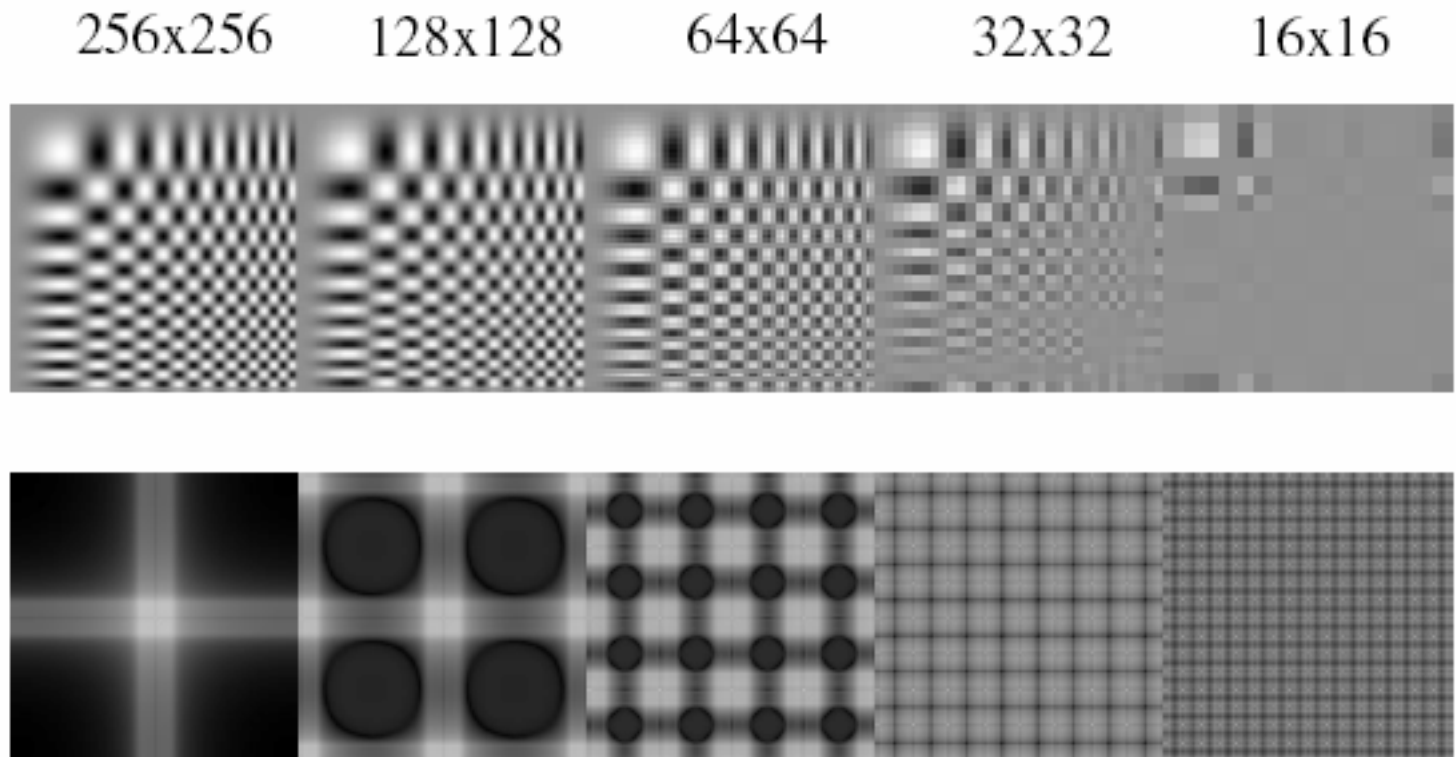


# Aliasing



# Gaussian Pyramid

- Sampling with smoothing
- Get the next image by smoothing the image with a Gaussian
- Sample at every second pixel to get the next.



# The Gaussian pyramid contd.

- Smooth with gaussians, because
  - a gaussian\*gaussian=another gaussian
- Synthesis
  - smooth and sample
- Gaussians are low pass filters, so **representation is redundant**

# Laplacian Pyramid Algorithm

- Create a Gaussian pyramid by successive smoothing with a Gaussian and down sampling
- Set the coarsest layer of the Laplacian pyramid to be the coarsest layer of the Gaussian pyramid
- For each subsequent layer  $n-1$ , compute
  - $L(n-1) = G(n-1) - \text{upsample}(G(n))$
- In general, the idea of using a series of Gaussians with different values of  $\sigma$  leads to the idea of *scale space* in computer vision
$$L(\sigma) = I * G(\sigma)$$
  - i.e. a continuous family of images

# Laplacian of Gaussian Pyramid



512

256

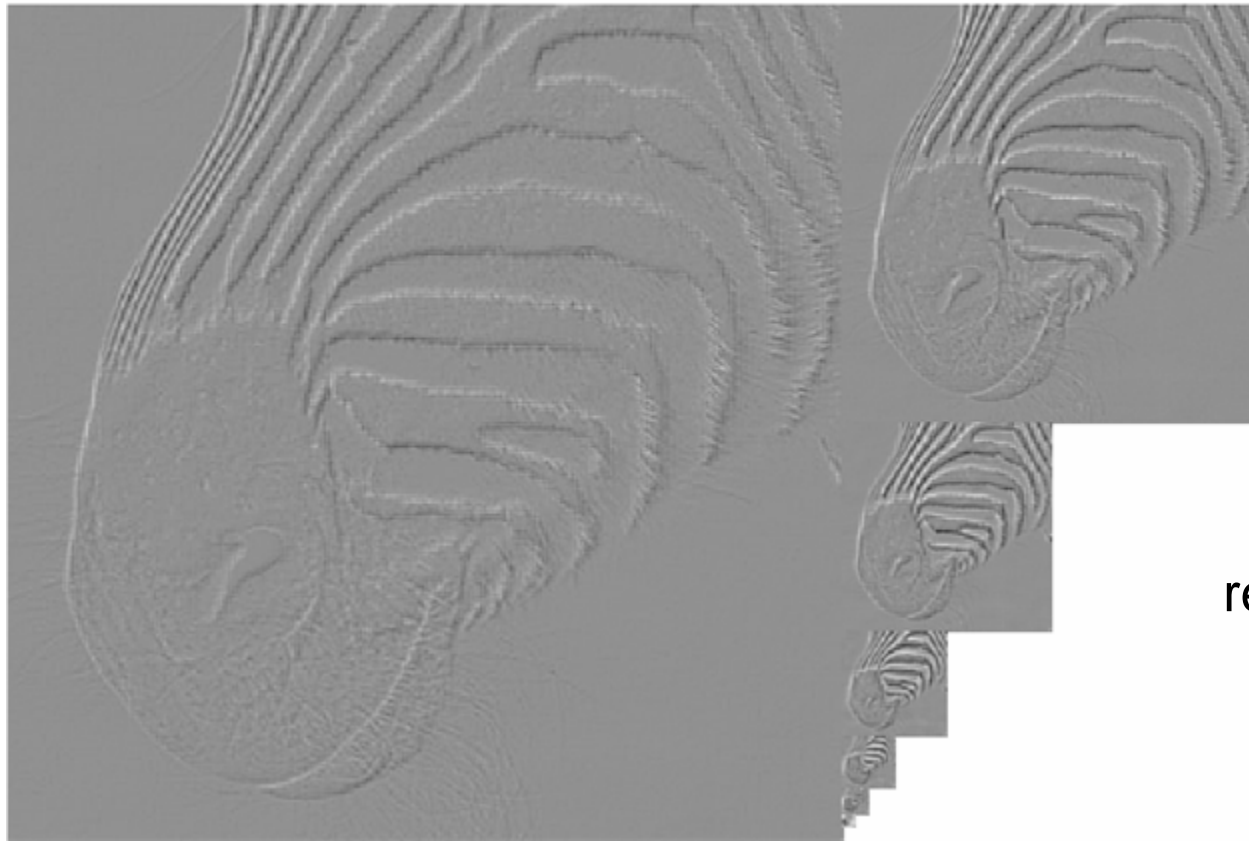
128

64

32

16

8



Different levels  
represent different  
frequencies.

# SIFT Features (by David Lowe)

- SIFT – Scale Invariant Feature Transform – One of the most popular image features being used presently.
- Approach for detecting and extracting local feature descriptors that are reasonably invariant to
  - Scale
  - Translation
  - Rotation
  - Illumination
  - Noise
  - Small changes in viewpoint

# SIFT feature detection steps

- Detection stages for SIFT features:
  - Scale-space extrema detection
  - Keypoint localization
  - Orientation assignment
  - Generation of keypoint descriptors.

# Scale-space extrema detection

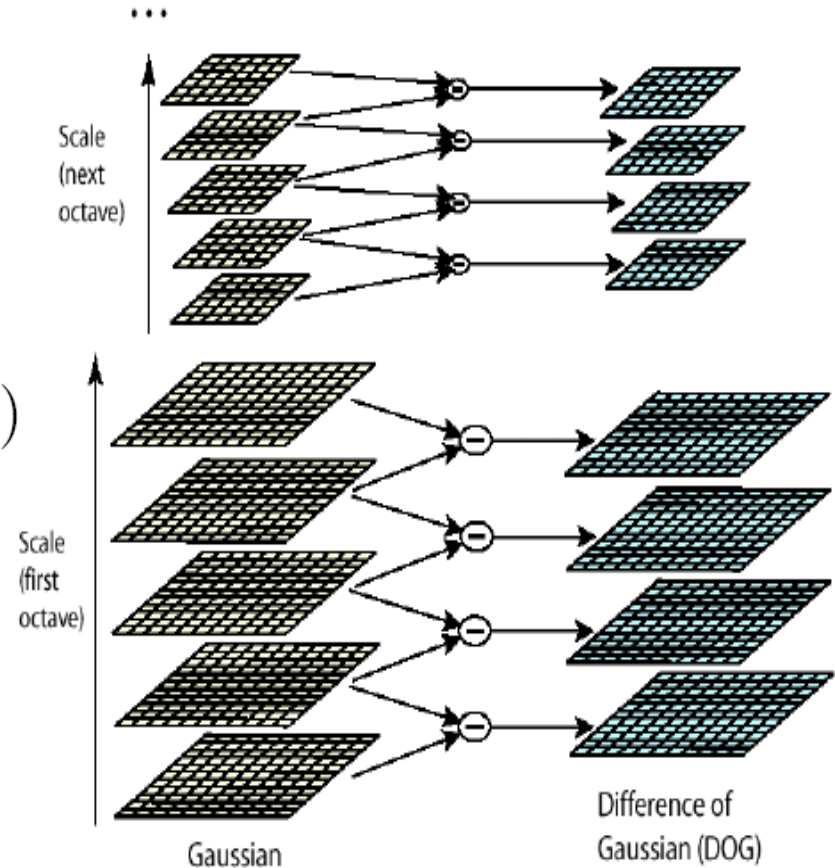
Scale space

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

Difference of Gaussian (DoG)

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

1. Convolution of the image with Gaussian filters at different scales
2. Generation of DoG images from adjacent blurred images

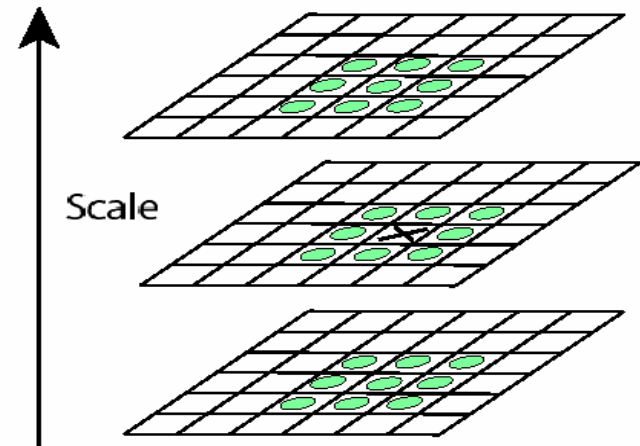


- Shown maxima and minima of  $\sigma^2 \nabla^2 G$  produce stable image features
- Difference of gaussian gives an approximation to  $\sigma^2 \nabla^2 G$



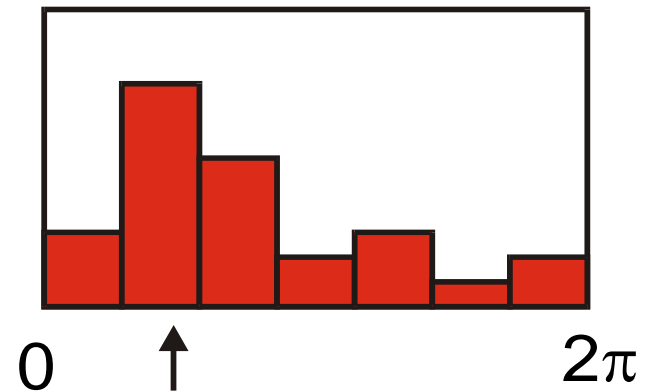
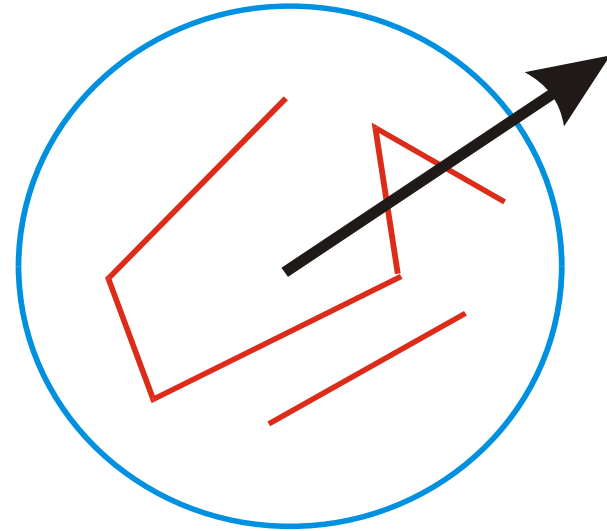
# Key point localization

- Detect maxima and minima of difference-of-Gaussian in scale space
- Fit a quadratic to surrounding values for sub-pixel and sub-scale interpolation (Brown & Lowe, 2002)
- Remove low-contrast points
- Also remove Edge-like features (using the harris measure we saw earlier)



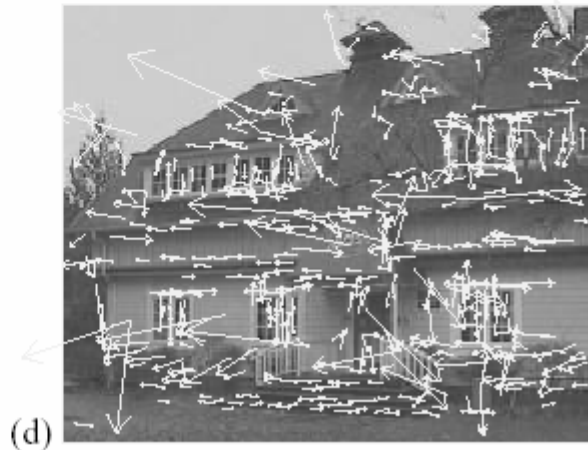
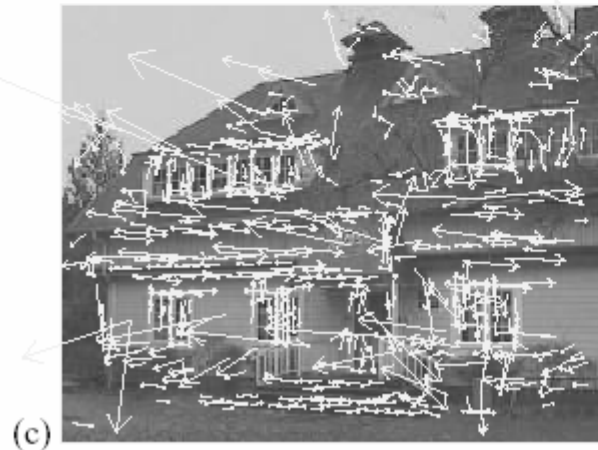
# Select canonical orientation

- Create histogram of local gradient directions computed at selected scale
- Assign canonical orientation at peak of smoothed histogram
- Each key specifies stable 2D coordinates (x, y, scale, orientation)



# Example of keypoint detection

Threshold on value at DOG peak and on ratio of principle curvatures (Harris approach)



- (a) 233x189 image
- (b) 832 DOG extrema
- (c) 729 left after peak value threshold
- (d) 536 left after testing ratio of principle curvatures

# SIFT vector formation

- Thresholded image gradients are sampled over 16x16 array of locations in scale space
- Create array of orientation histograms
- 8 orientations x 4x4 histogram array = 128 dimensions

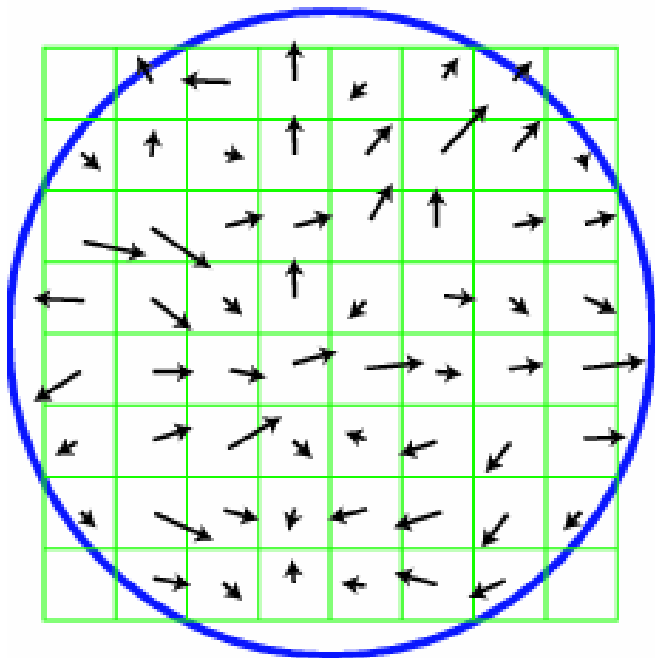
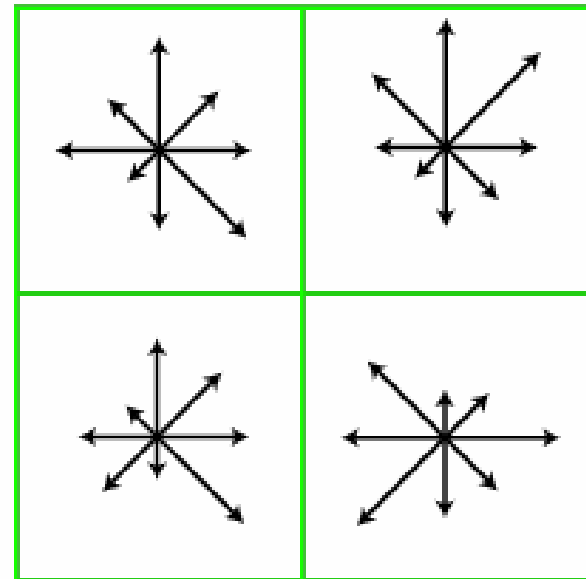
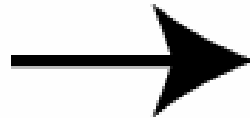


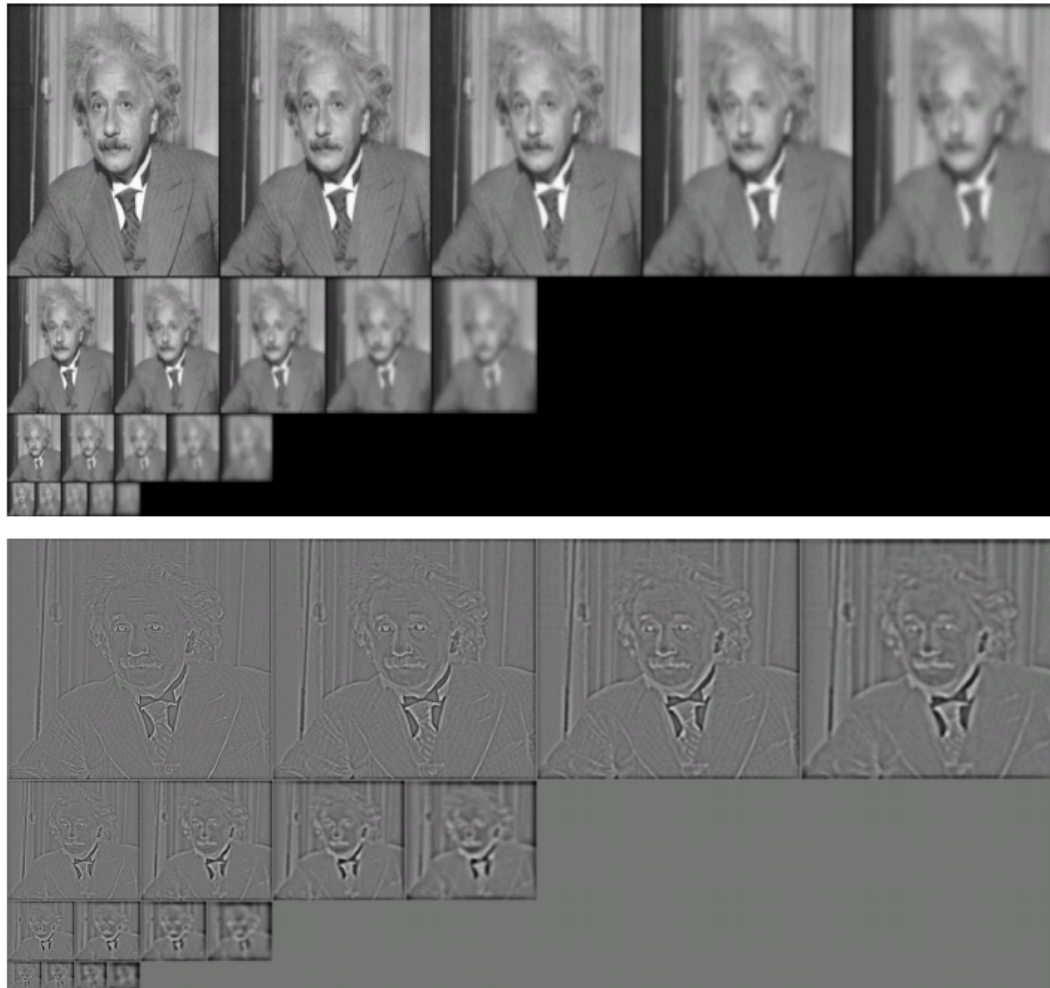
Image gradients



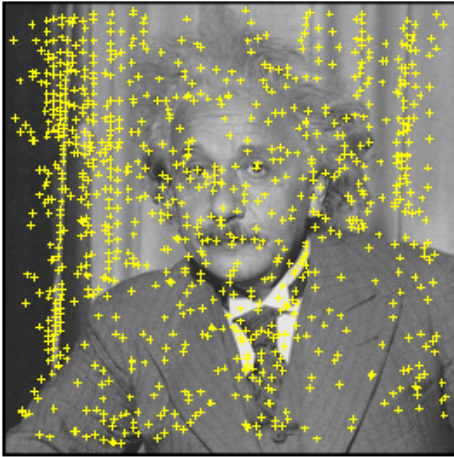
Keypoint descriptor

# Lets review the steps again

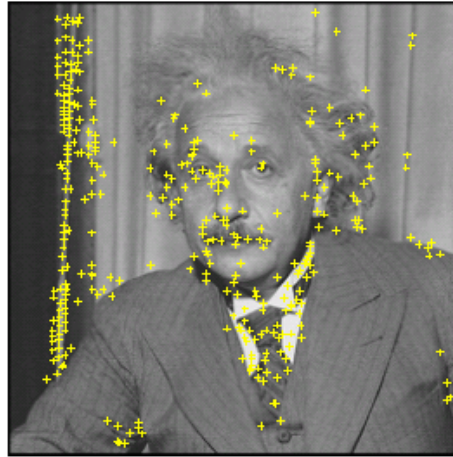
## 1. Compute Gaussian filtered and DOG images



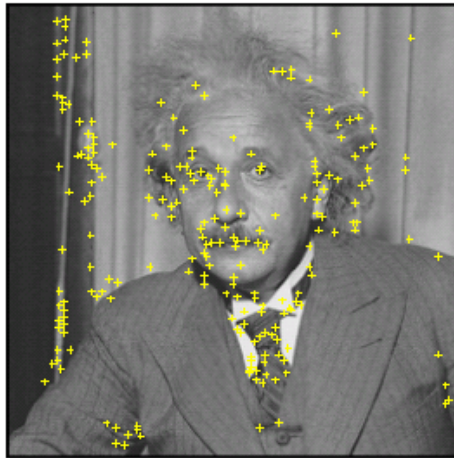
## 2. Key point detection



a)



b)

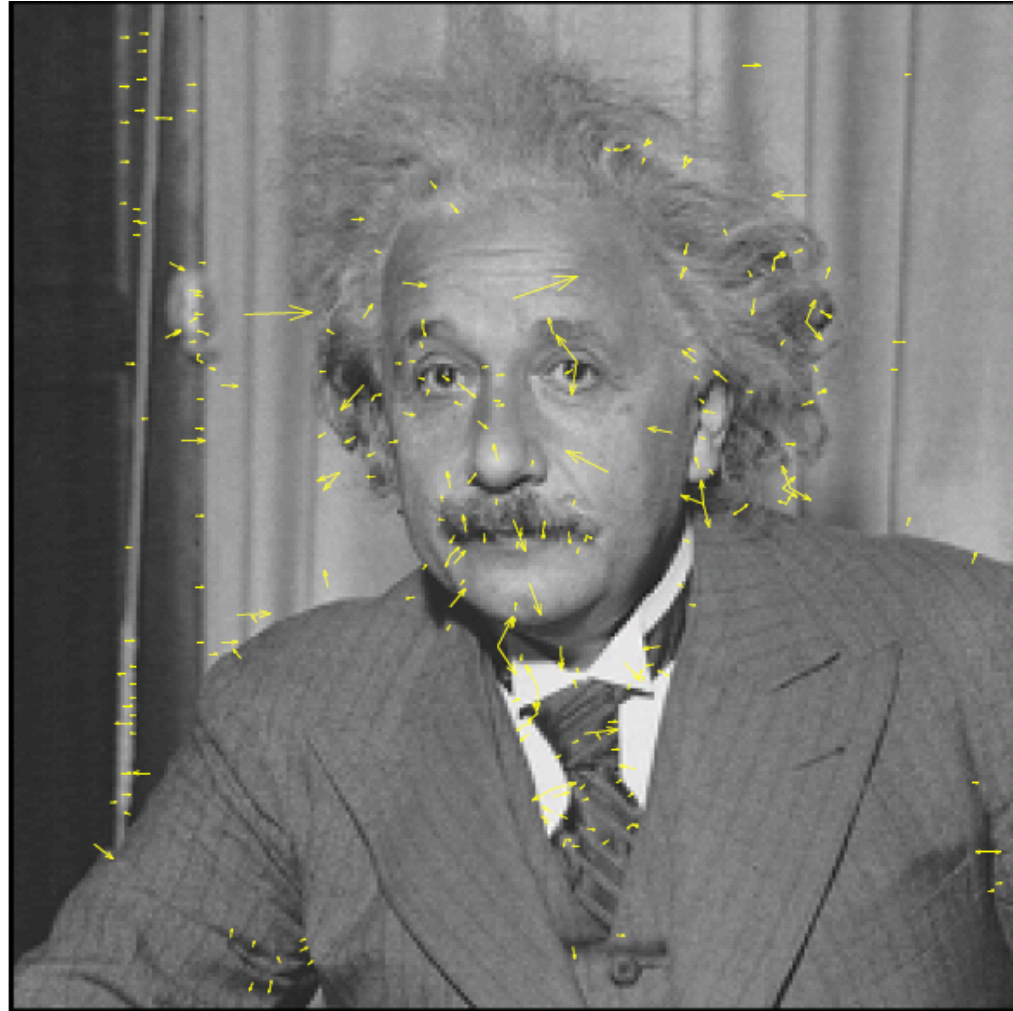


c)

- a) Maxima/minima of DoG
- b) After removing low-contrast points
- c) After removal of edge-like responses

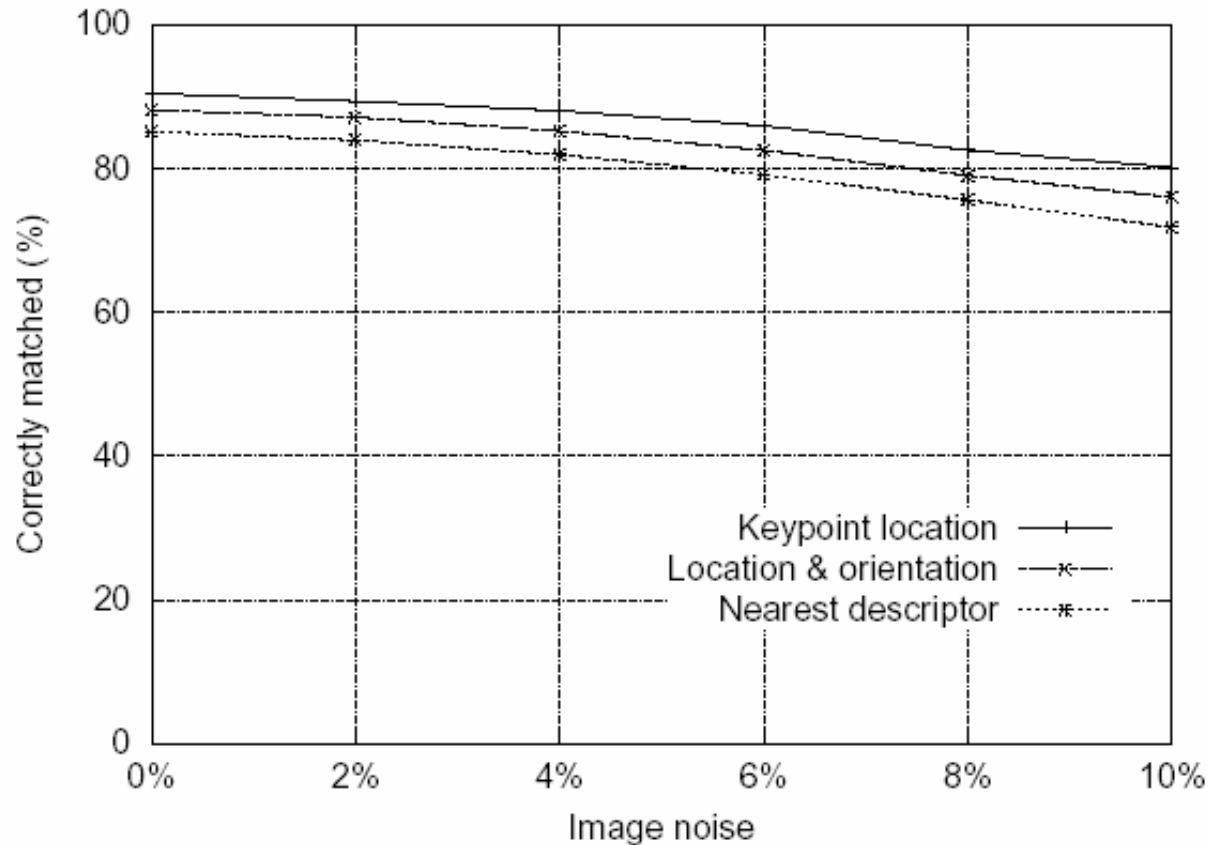


### 3. Final keypoints (with selected orientation and scale)



# Feature stability to noise

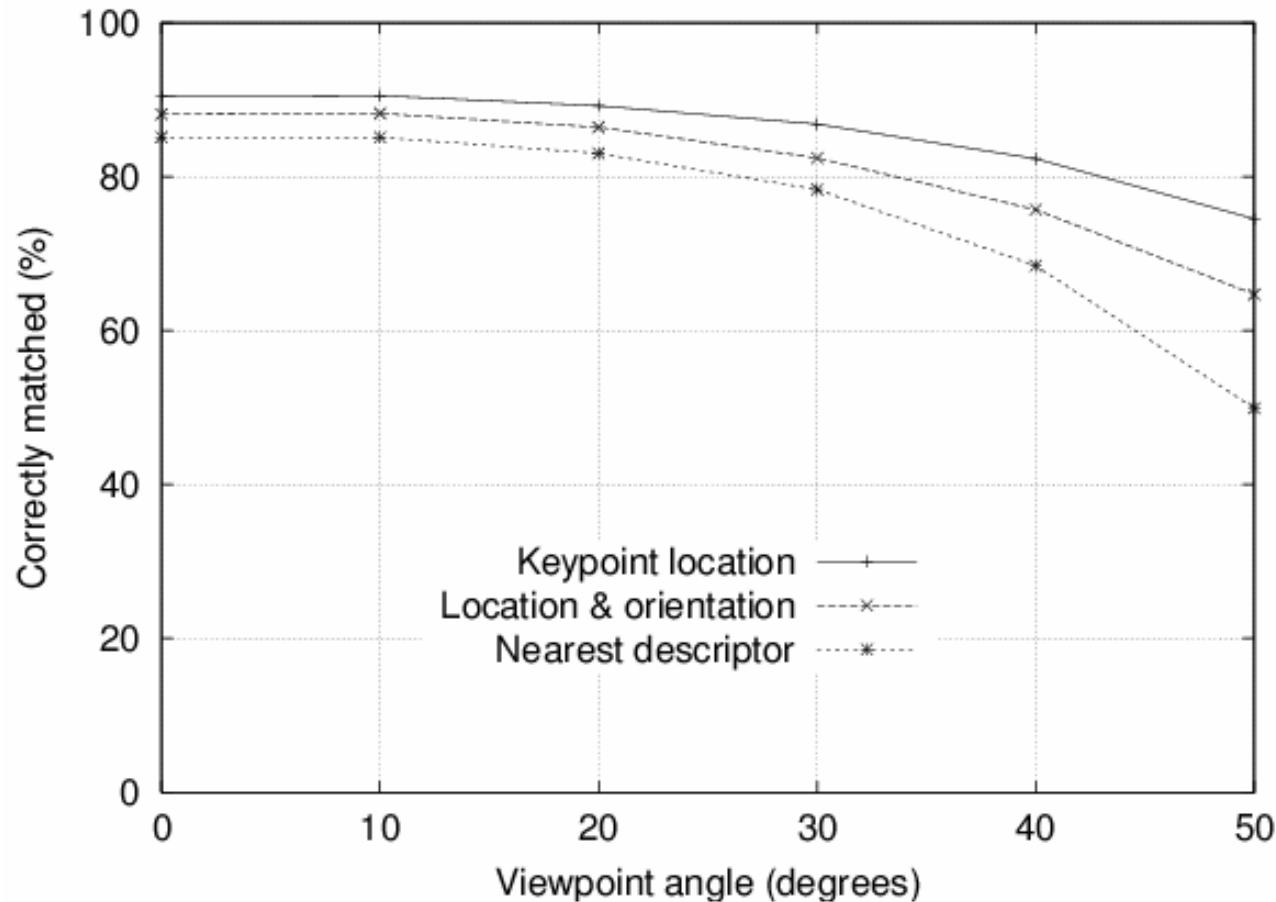
- Match features after random change in image scale & orientation, with differing levels of image noise
- Find nearest neighbor in database of 30,000 features





# Feature stability to affine change

- Match features after random change in image scale & orientation, with 2% image noise, and affine distortion
- Find nearest neighbor in database of 30,000 features



# Application – Matching & Alignment

Original View



Range: [0, 1]  
Dims: [384, 512]

Reference View



Range: [0, 1]  
Dims: [384, 512]

Aligned View



Range: [-0.0273, 1]  
Dims: [384, 512]

Reference minus Aligned View



Range: [-0.767, 0.822]  
Dims: [384, 512]