Elements of Image (Pre)-Processing and Feature Detection

Acknowledgements: slides from Steven Seitz, Aleosha Efros, David Forsyth, and Gonzalez & Woods

Image Processing Basics

The University of Western Ontario

- **Point Processing**
  Extra Reading: Szeliski, Sec 3.1
  intensities, colors
  - gamma correction
  - window-center correction
  - histogram equalization
- **Filtering** (linear and non-linear)
  Extra Reading: Szeliski, Sec 3.2-3.3
  contrast edges
  texture
  templates, patches
  - mean, Gaussian, and median filters
  - image gradients, Laplacian
  - normalized cross-correlation (NCC)
  - etc…: Fourier, Gabor, wavelets (Szeliski, Sec 3.4-3.5)
- **Other features**
  Extra Reading: Szeliski, Sec. 4.1
  Harris corners, MOPS, SIFT, etc.

Summary of image transformations

- An **image processing** operation (or transformation) typically defines a new image g in terms of an existing image f.

Examples:

Summary of image transformations

- An **image processing** operation (or transformation) typically defines a new image *g* in terms of an existing image *f*.

Examples:

  - **Geometric (domain) transformation**:
    - What kinds of operations $g(x, y) = f(t_x(x, y), t_y(x, y))$

Summary of image transformations

- An **image processing** operation (or transformation) typically defines a new image *g* in terms of an existing image *f*.

Examples:

- **Geometric (domain) transformation**:
  - What kinds of operations $g(x,y) = f(t_x(x,y), t_y(x,y))$
- **Range transformation**:
  - What kinds of operations $g(x,y) = t(f(x,y))$

Summary of image transformations

- An **image processing** operation (or transformation) typically defines a new image *g* in terms of an existing image *f*.

Examples:

- **Geometric (domain) transformation**:
  - What kinds of operations $g(x,y) = f(t_x(x,y), t_y(x,y))$
- **Range transformation**:
  - What kinds of operations $g(x,y) = t(f(x,y))$

point processing

- **Filtering** also generates new images from an existing image

neighborhood processing

$$g(x,y) = \int\limits_{\substack{|u|<\varepsilon \\ |v|<\varepsilon}} h(u,v) \cdot f(x-u, y-v) \cdot du \cdot dv$$

  - more on filtering later

Point Processing

$$g(x, y) = t(f(x, y))$$

for each original image intensity value I function $t(\cdot)$ returns a transformed intensity value $t(I)$.

$$I' = t(I)$$

NOTE: we will often use notation I_p instead of $f(x,y)$ to denote intensity at pixel $p=(x,y)$

- **Important:** every pixel is for itself
  - spatial information is ignored!

- What can point processing do?

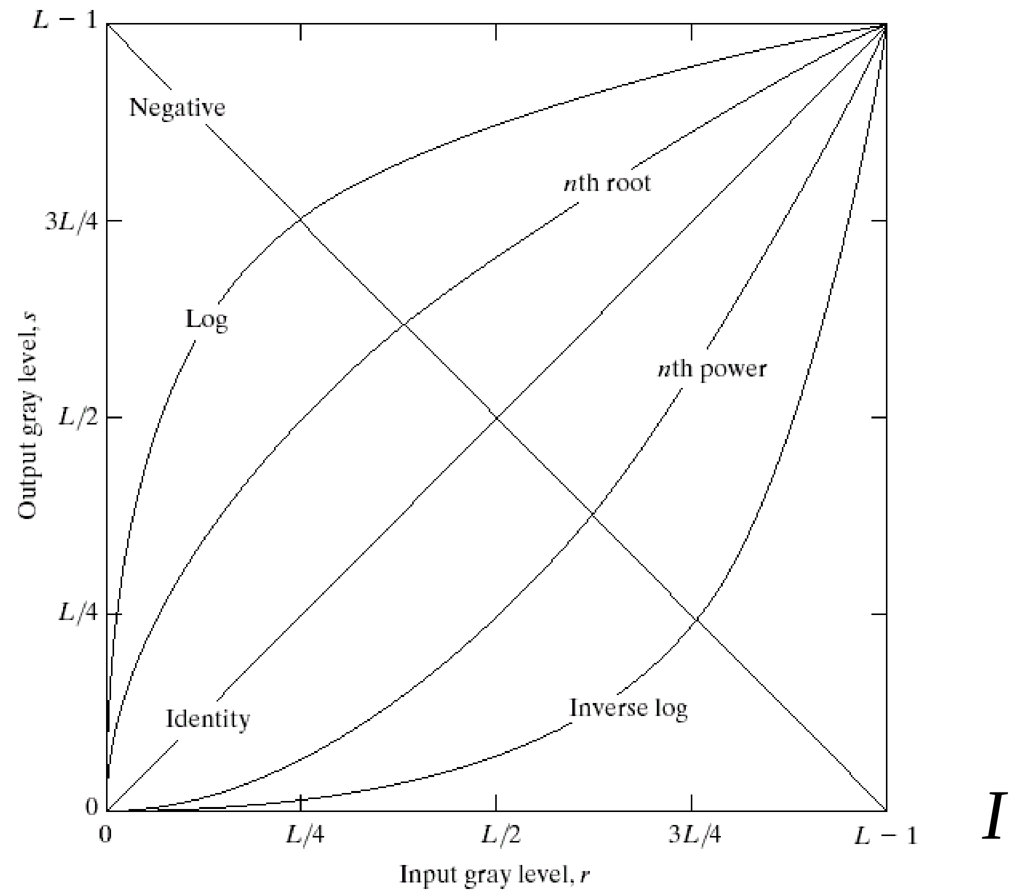
(we will focus on grey scale images, see Szeliski 3.1 for examples of point processing for color images)

Point Processing:

Examples of gray-scale transforms t

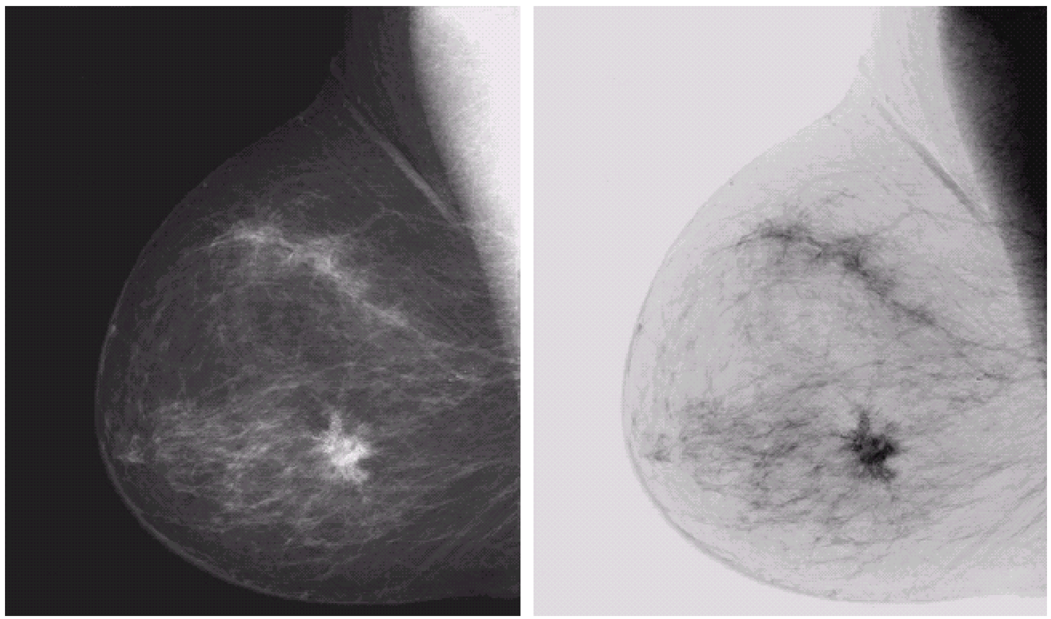
$$I' = t(I)$$

FIGURE 3.3 Some basic gray-level transformation functions used for image enhancement.



Point Processing:

Negative



a b

FIGURE 3.4 (a) Original digital mammogram. (b) Negative image obtained using the negative transformation in Eq. (3.2-1). (Courtesy of G.E. Medical Systems.)

I_p or $f(x,y)$ $\qquad$ I'_p or $g(x,y)$

$$t(I) = 255 - I$$

$$g(x,y) = t(f(x,y)) = 255 - f(x,y)$$

Point Processing:

Power-law transformations t

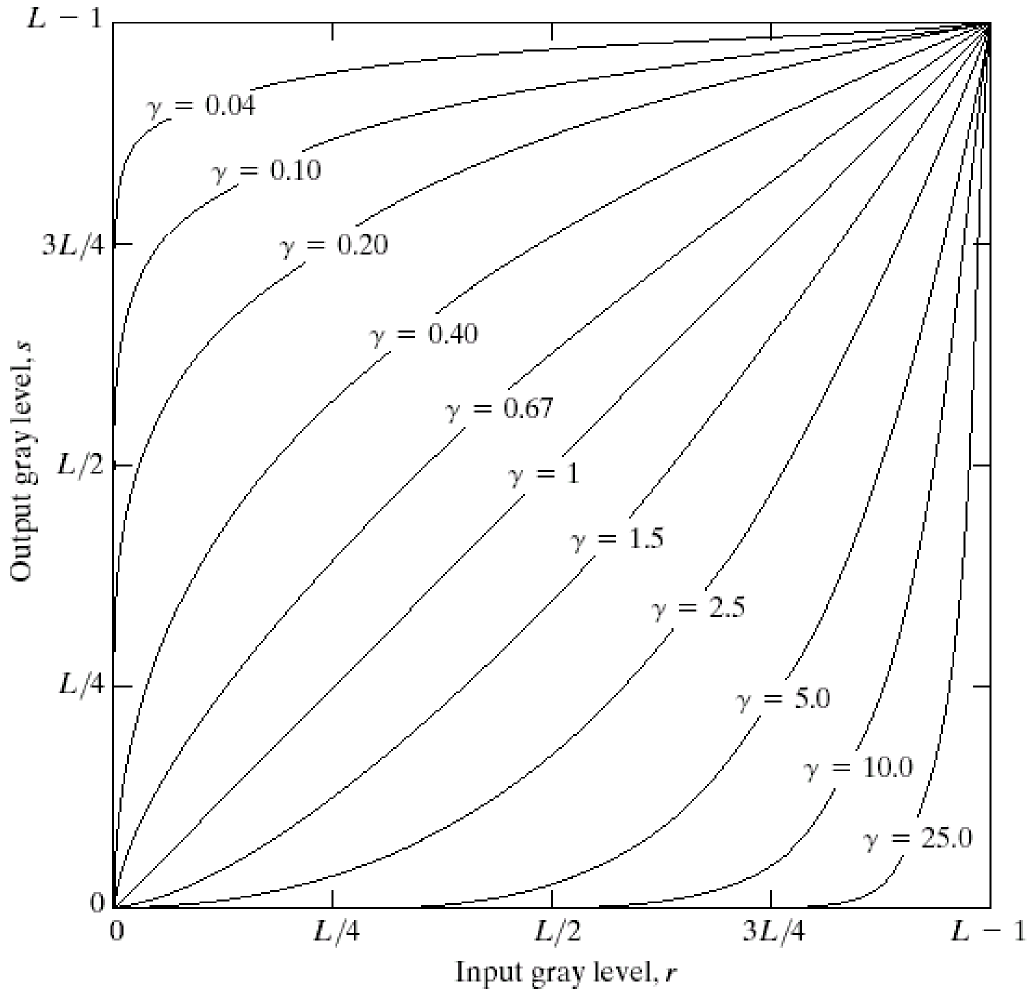


FIGURE 3.6 Plots of the equation $s = cr^{\gamma}$ for various values of γ ($c = 1$ in all cases).

Point Processing:

Gamma Correction

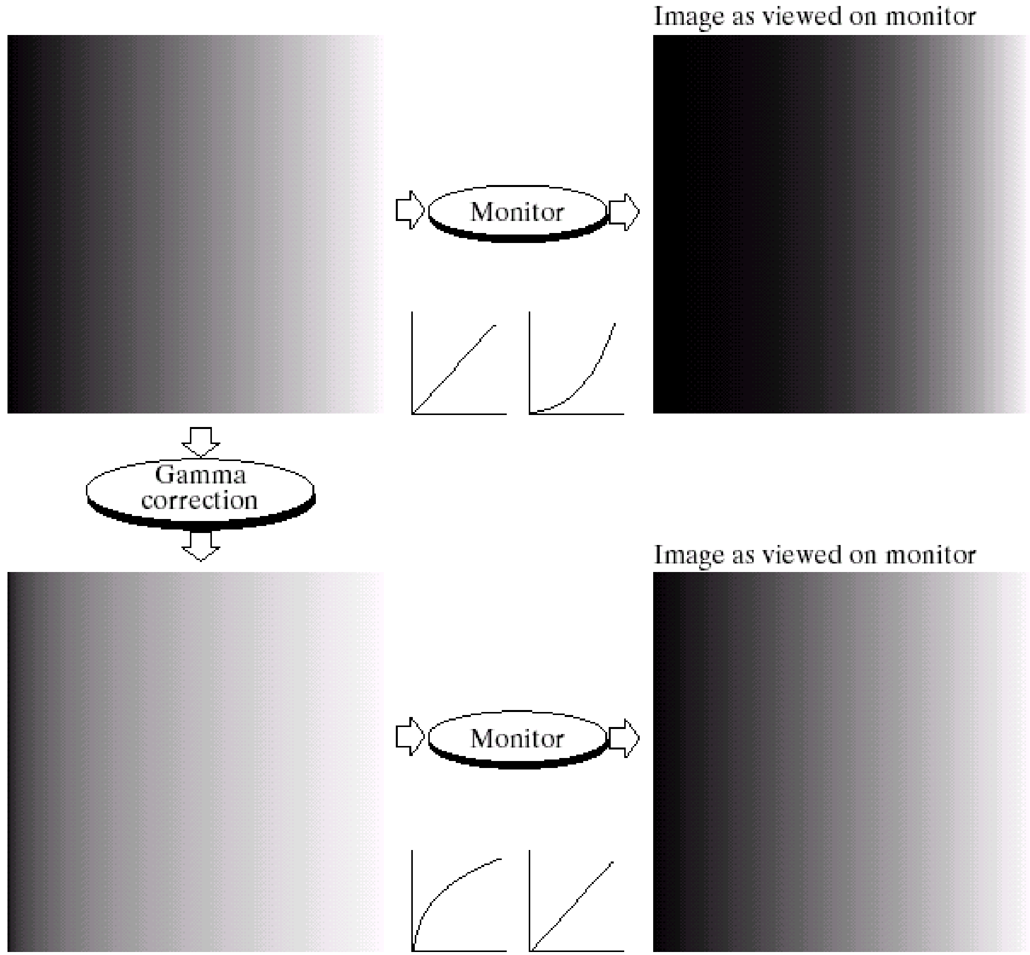


FIGURE 3.7 (a) Linear-wedge gray-scale image. (b) Response of monitor to linear wedge. (c) Gamma-corrected wedge. (d) Output of monitor.

Gamma Measuring Applet:

http://www.cs.berkeley.edu/~efros/java/gamma/gamma.html

Point Processing:

Enhancing Image via Gamma Correction

FIGURE 3.9
(a) Aerial image. (b)–(d) Results of applying the transformation in Eq. (3.2-3) with $c = 1$ and $\gamma = 3.0, 4.0,$ and 5.0, respectively. (Original image for this example courtesy of NASA.)

Point Processing:

Understanding Image Histograms

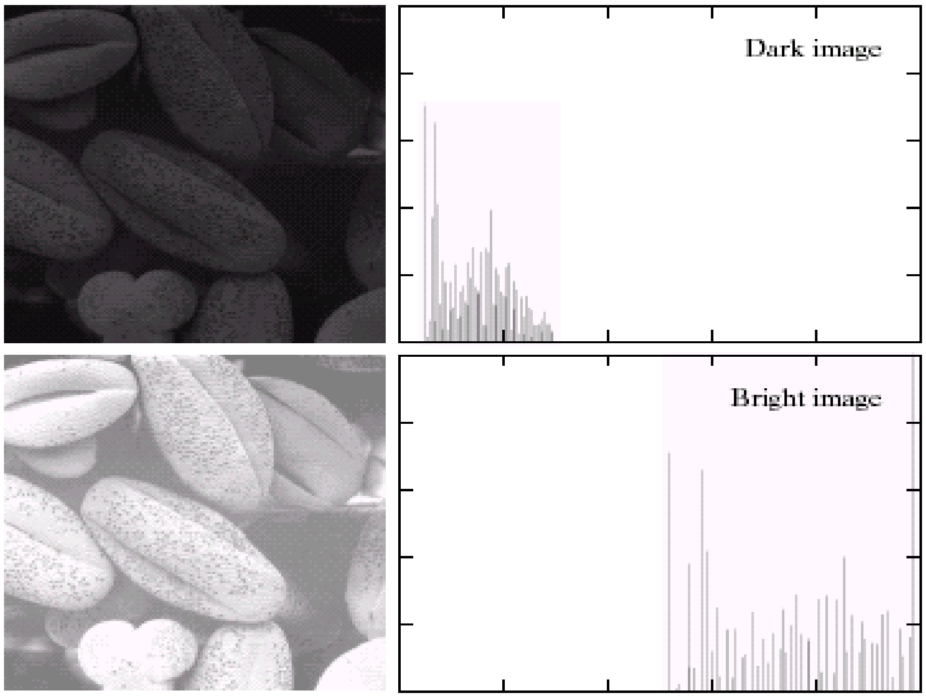


Image Brightness

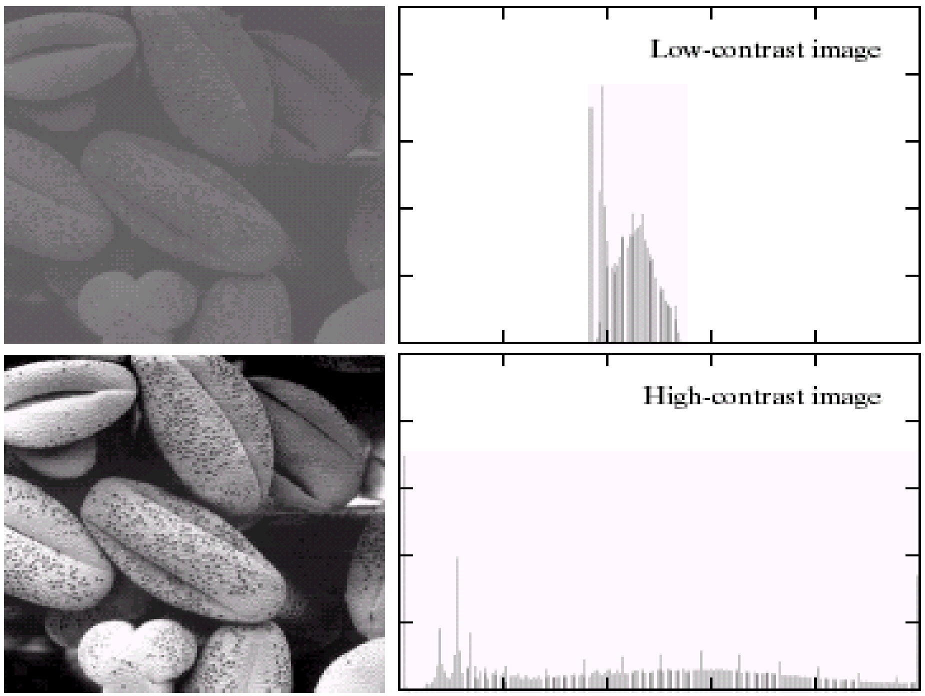


Image Contrast

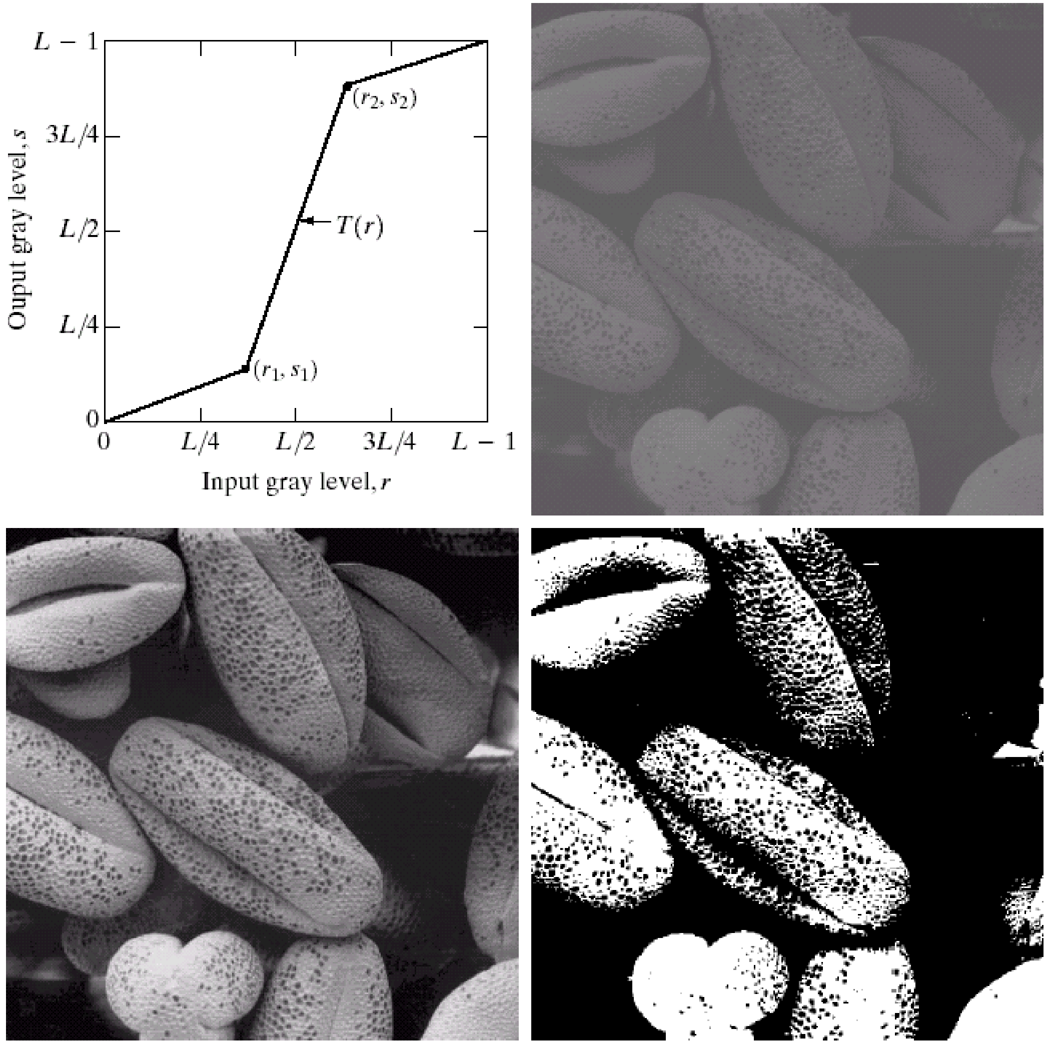
probability of intensity i : $$p(i) = \frac{n_i}{n}$$

n_i ---number of pixels with intensity i

n ---total number of pixels in the image

Point Processing:

Contrast Stretching



a b
c d

FIGURE 3.10 Contrast stretching. (a) Form of transformation function. (b) A low-contrast image. (c) Result of contrast stretching. (d) Result of thresholding. (Original image courtesy of Dr. Roger Heady, Research School of Biological Sciences, Australian National University, Canberra, Australia.)

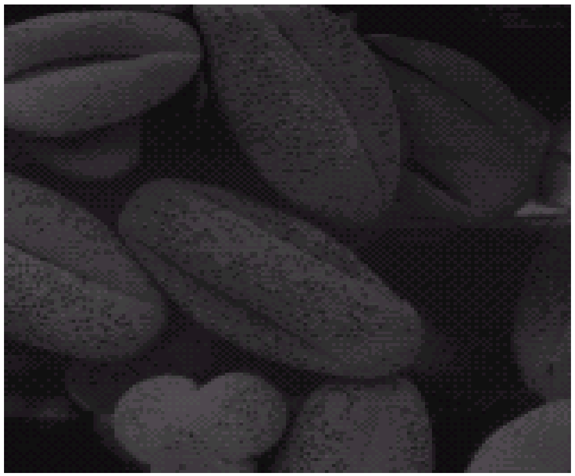
The University of Western Ontario

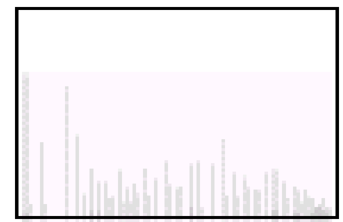
Point Processing:

Contrast Stretching

Original images | Histogram corrected images

1)





2)

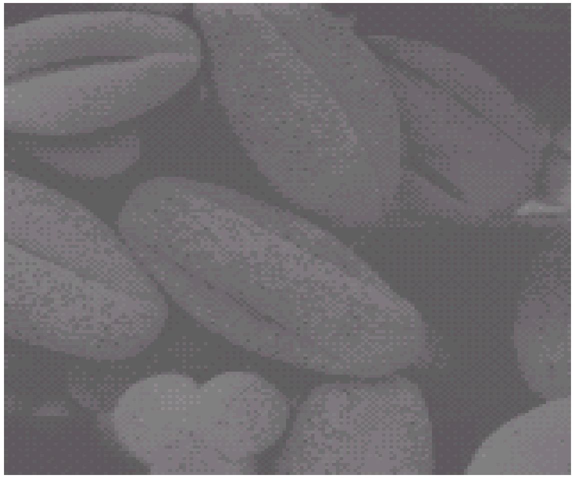
Point Processing:

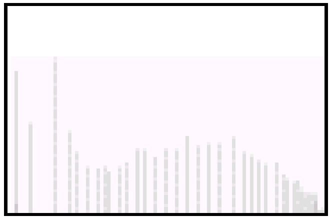
Contrast Stretching

Original images

Histogram corrected images

3)



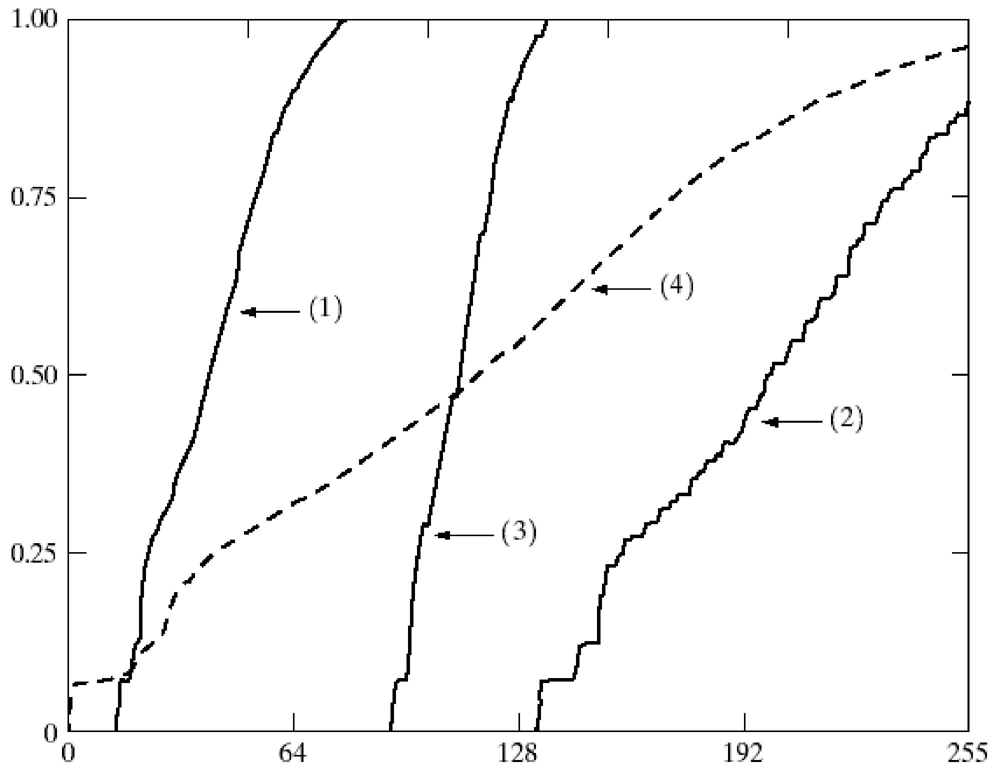


4)

One way to automatically select transformation t :

Histogram Equalization

FIGURE 3.18 Transformation functions (1) through (4) were obtained from the histograms of the images in Fig.3.17(a), using Eq. (3.3-8).



$$t(i) = \sum_{j=0}^{i} p(j) = \sum_{j=0}^{i} \frac{n_j}{n}$$

= cumulative distribution of image intensities

…see Gonzalez and Woods, Sec3.3.1, for more details

Point processing

Histogram Equalization

$$t(i) = \sum_{j=0}^{i} p(j) = \sum_{j=0}^{i} \frac{n_j}{n}$$

= cumulative distribution of image intensities

Why does that work?

Answer in probability theory:

I – random variable with *probability* distribution *p(i)* over *i* in *[0,1]*

If *t(i)* is a *cumulative* distribution of *I* then

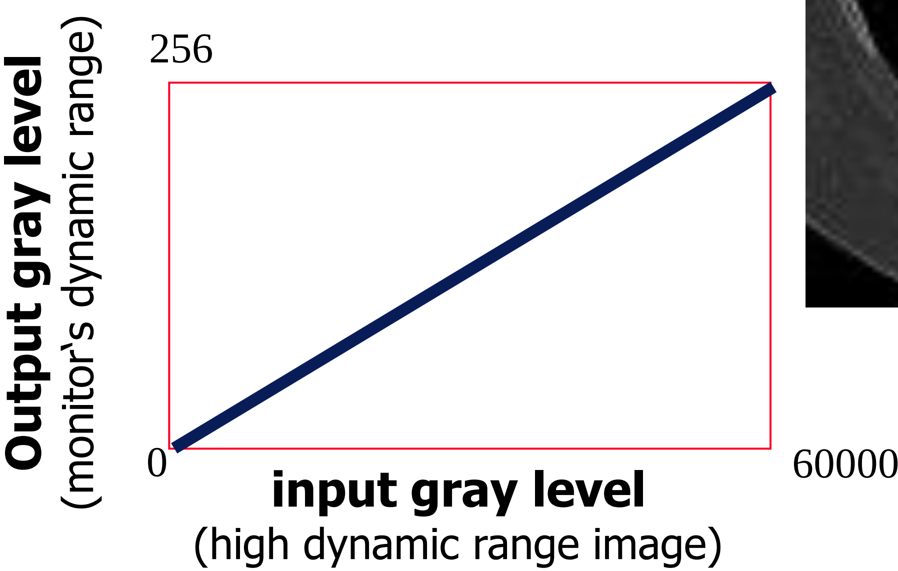
I'=t(I) – is a random variable with *uniform* distribution over its range [0,1]

That is, transform image *I'* will have a uniformly-spread histogram (good contrast)

The University of Western Ontario

Point Processing:

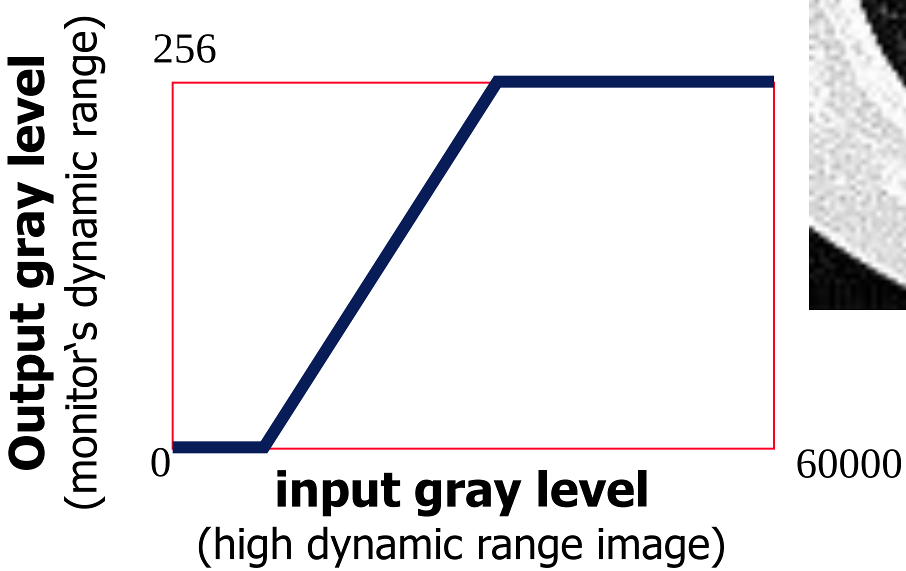
Window-Center adjustment



The University of Western Ontario

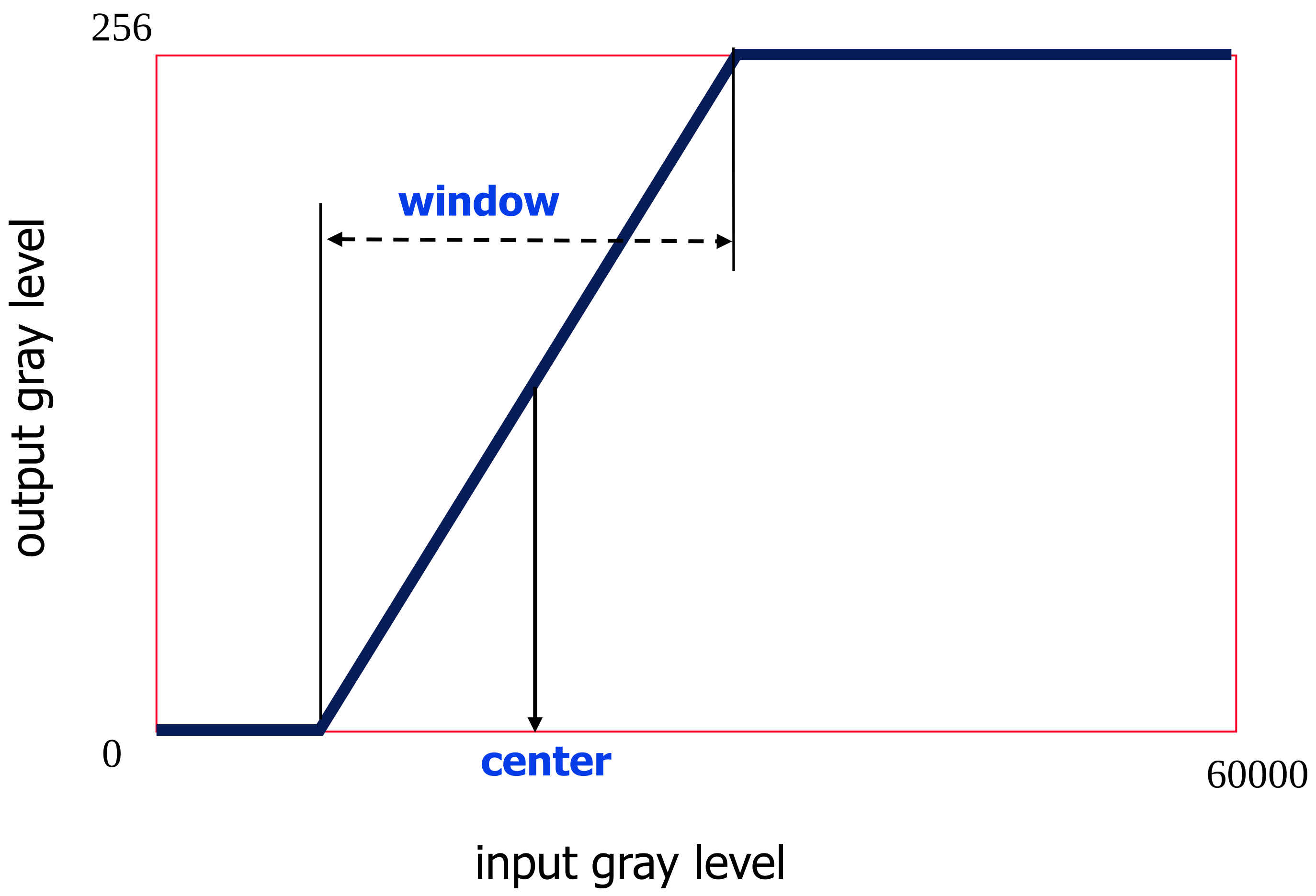
Point Processing:

Window-Center adjustment



Point Processing:

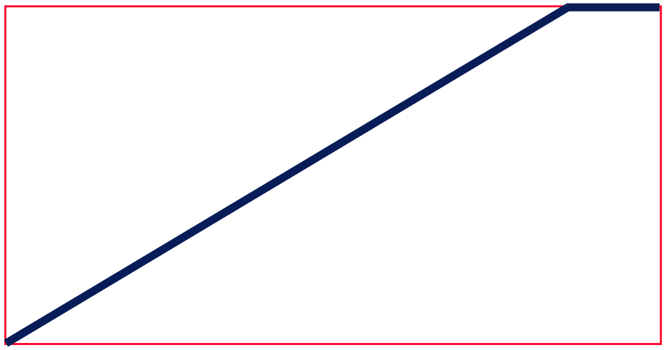
Window-Center adjustment



Point Processing:

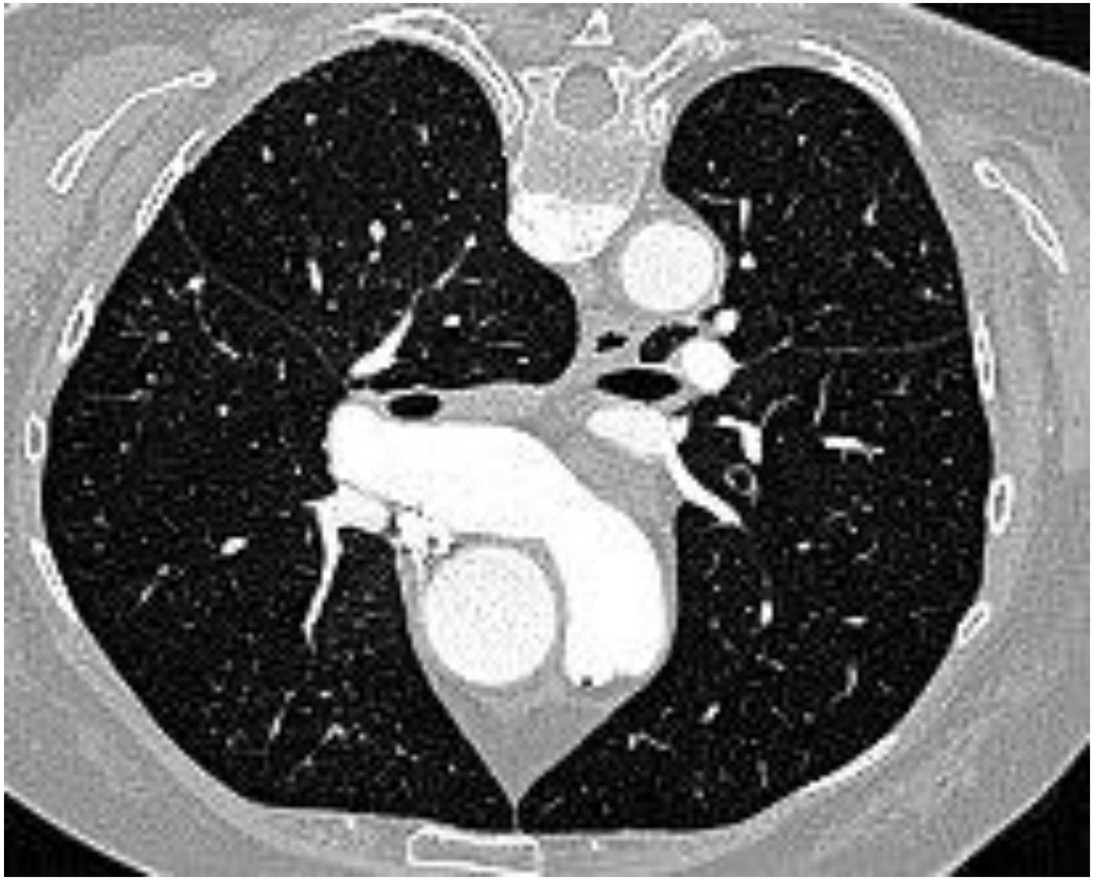
Window-Center adjustment

Window = 4000
Center = 500

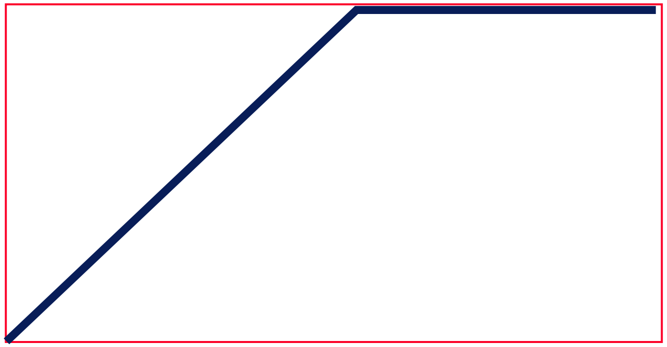


Point Processing:

Window-Center adjustment



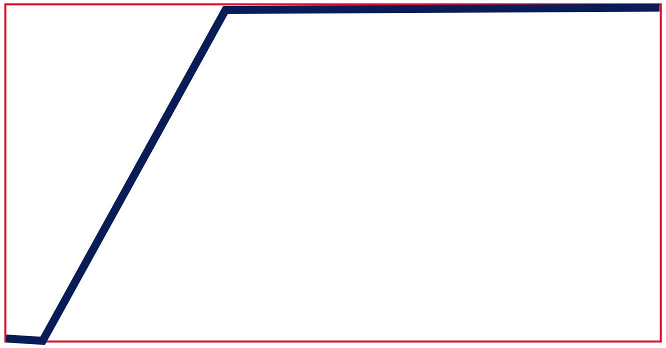
Window = 2000
Center = 500



Window-Center adjustment

The University of Western Ontario

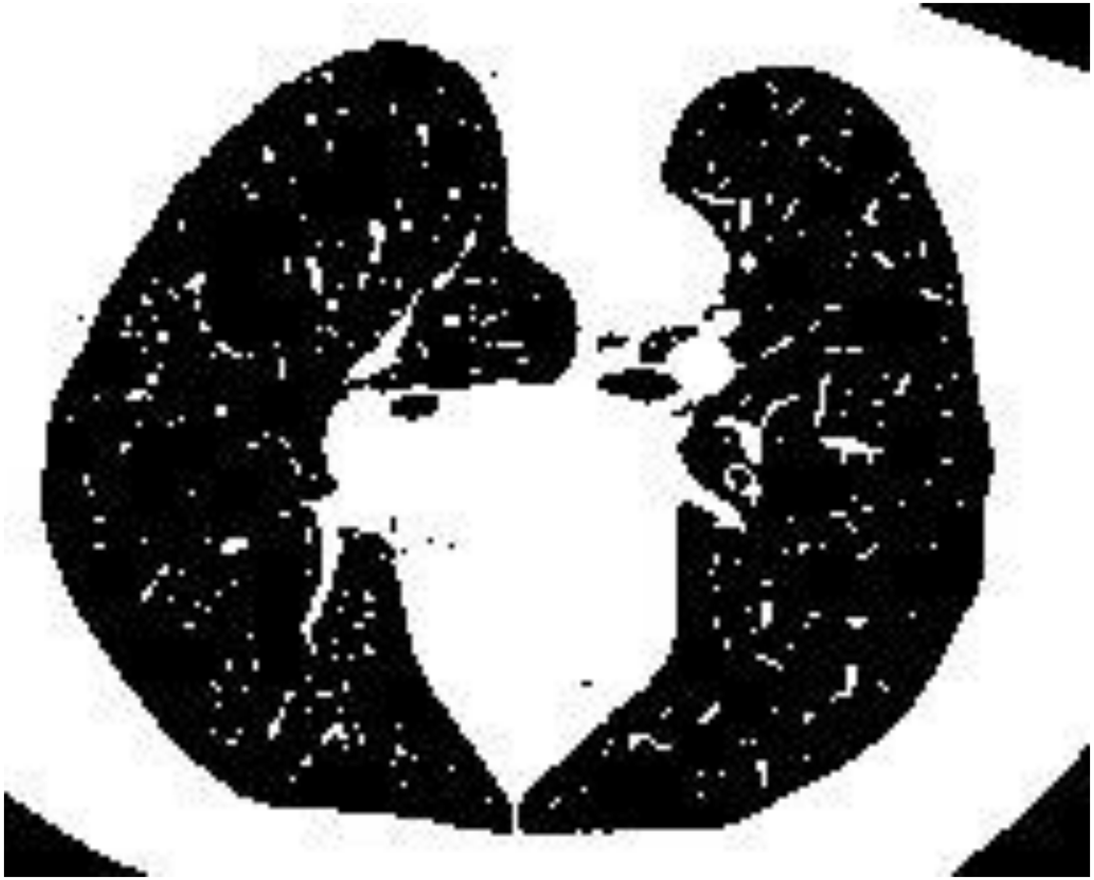
Window = 800
Center = 500



Window-Center adjustment

The University of Western Ontario

Window = 0
Center = 500

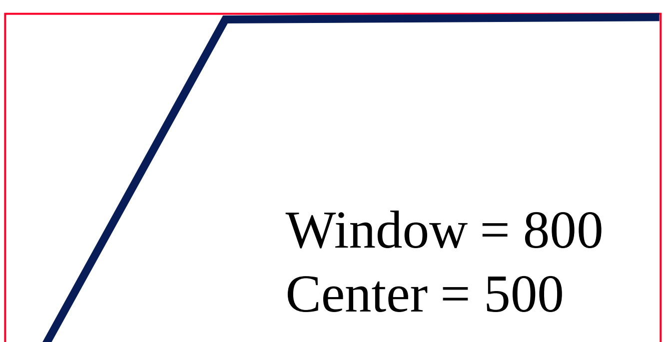


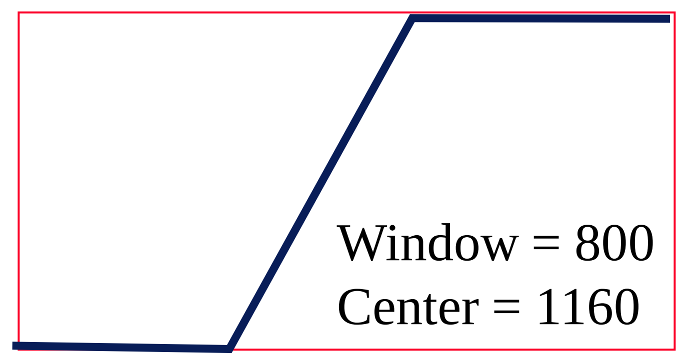
If *window=0* then we get binary image *thresholding*

The University of Western Ontario

Point Processing:

Window-Center adjustment

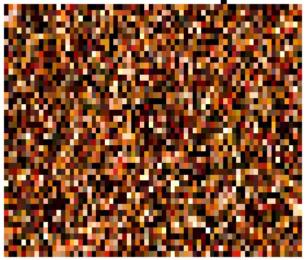




Neighborhood Processing (or filtering)

The University of Western Ontario

- Q: What happens if I reshuffle all pixels within the image?



- A: It's histogram won't change.
  No point processing will be affected…

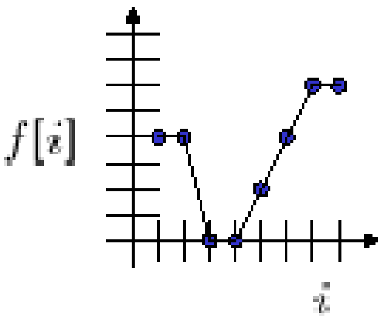
- Images contain a lot of "spatial information"

Readings: Szeliski, Sec 3.2-3.3

Neighborhood Processing (filtering)

Linear image transforms

Let's start with 1D image (a signal): f[i]



$$f[i] = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

A very general and useful class of transforms are the **linear transforms** of f, defined by a matrix M

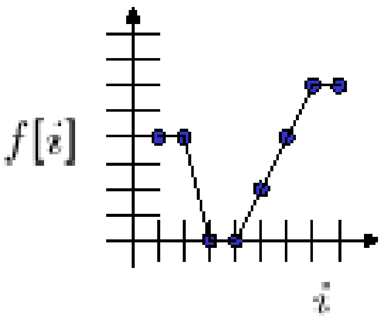
$$\underbrace{\begin{bmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & * \end{bmatrix}}_{M[i,j]} \underbrace{\begin{bmatrix} * \\ * \\ \vdots \\ * \end{bmatrix}}_{f[i]} = \underbrace{\begin{bmatrix} * \\ * \\ \vdots \\ * \end{bmatrix}}_{g[i]}$$

$$g[i] = \sum_{j=1} M[i,j] f[j]$$

Neighborhood Processing (filtering)

Linear image transforms

Let’s start with 1D image (a signal): f[i]



$$f[i] = \begin{bmatrix} \\ \\ \\ \\ \end{bmatrix}$$

matrix M

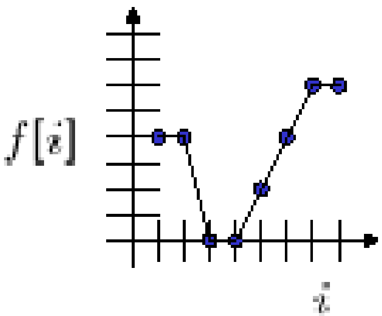
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \qquad f[i] \longrightarrow$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \qquad f[i] \longrightarrow$$

Neighborhood Processing (filtering)

Linear image transforms

Let’s start with 1D image (a signal): f[i]



$$f[i] = \begin{bmatrix} \\ \\ \\ \end{bmatrix}$$

matrix M

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \qquad f[i] \longrightarrow$$

$$\frac{1}{2}\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \qquad f[i] \longrightarrow$$

Neighborhood Processing (filtering)

Linear shift-invariant filters

matrix M

$$\begin{bmatrix} * & * & 0 & 0 & 0 & 0 & 0 & 0 \\ a & b & c & 0 & 0 & 0 & 0 & 0 \\ 0 & a & b & c & 0 & 0 & 0 & 0 \\ 0 & 0 & a & b & c & 0 & 0 & 0 \\ 0 & 0 & 0 & a & b & c & 0 & 0 \\ 0 & 0 & 0 & 0 & a & b & c & 0 \\ 0 & 0 & 0 & 0 & 0 & a & b & c \\ 0 & 0 & 0 & 0 & 0 & 0 & * & * \end{bmatrix}$$

This pattern is very common
- same entries in each row
- all non-zero entries near the diagonal

$$g = M \cdot f$$

It is known as a **linear shift-invariant filter** and is represented by a **kernel** (or **mask**) h:

$$h[i] = [a \;\; b \;\; c]$$

and can be written (for kernel of size $2k+1$) as:

$$g[i] = \sum_{u=-k}^{k} h[u] \cdot f[i+u]$$

The above allows negative filter indices. When you implement need to use: $h[u+k]$ instead of $h[u]$

Neighborhood Processing (filtering)

2D linear transforms

We can do the same thing for 2D images by concatenating all of the rows into one long vector (in a "*raster-scan*" order):

$$\hat{f}[i] = f[\lfloor i/m \rfloor, i\%m]$$

$$\underbrace{\begin{bmatrix} * & * & \cdots & * \\ * & * & \cdots & * \\ \vdots & \vdots & \ddots & \vdots \\ * & * & \cdots & * \end{bmatrix}}_{M[i,j]} \underbrace{\begin{bmatrix} * \\ * \\ \vdots \\ * \end{bmatrix}}_{\hat{f}[i]} = \underbrace{\begin{bmatrix} * \\ * \\ \vdots \\ * \end{bmatrix}}_{\hat{g}[i]}$$

Neighborhood Processing (filtering)

2D filtering

A 2D image *f[i,j]* can be filtered by a 2D kernel *h[u,v]* to produce an output image *g[i,j]*:

$$g[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] \cdot f[i+u, j+v]$$

This is called a **cross-correlation** operation and written:

$$g = h \circ f$$

h is called the "**filter**," "**kernel**," or "**mask**."

Neighborhood Processing (filtering)

2D filtering

A **convolution** operation is a cross-correlation where the filter is flipped both horizontally and vertically before being applied to the image:

$$g[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] \cdot f[i-u, j-v]$$

It is written: $\boxed{g = h * f}$ $\quad = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[-u,-v] \cdot f[i+u, j+v]$

How does convolution differ from cross-correlation?

If $h[u,v] = h[-u,-v]$ then there is no difference between convolution and cross-correlation

convolution has additional "technical" properties: commutativity, associativity. Also, "nice" properties wrt Fourier analysis.
(see Szeliski Sec 3.2, Gonzalez and Woods Sec. 4.6.4)

2D filtering

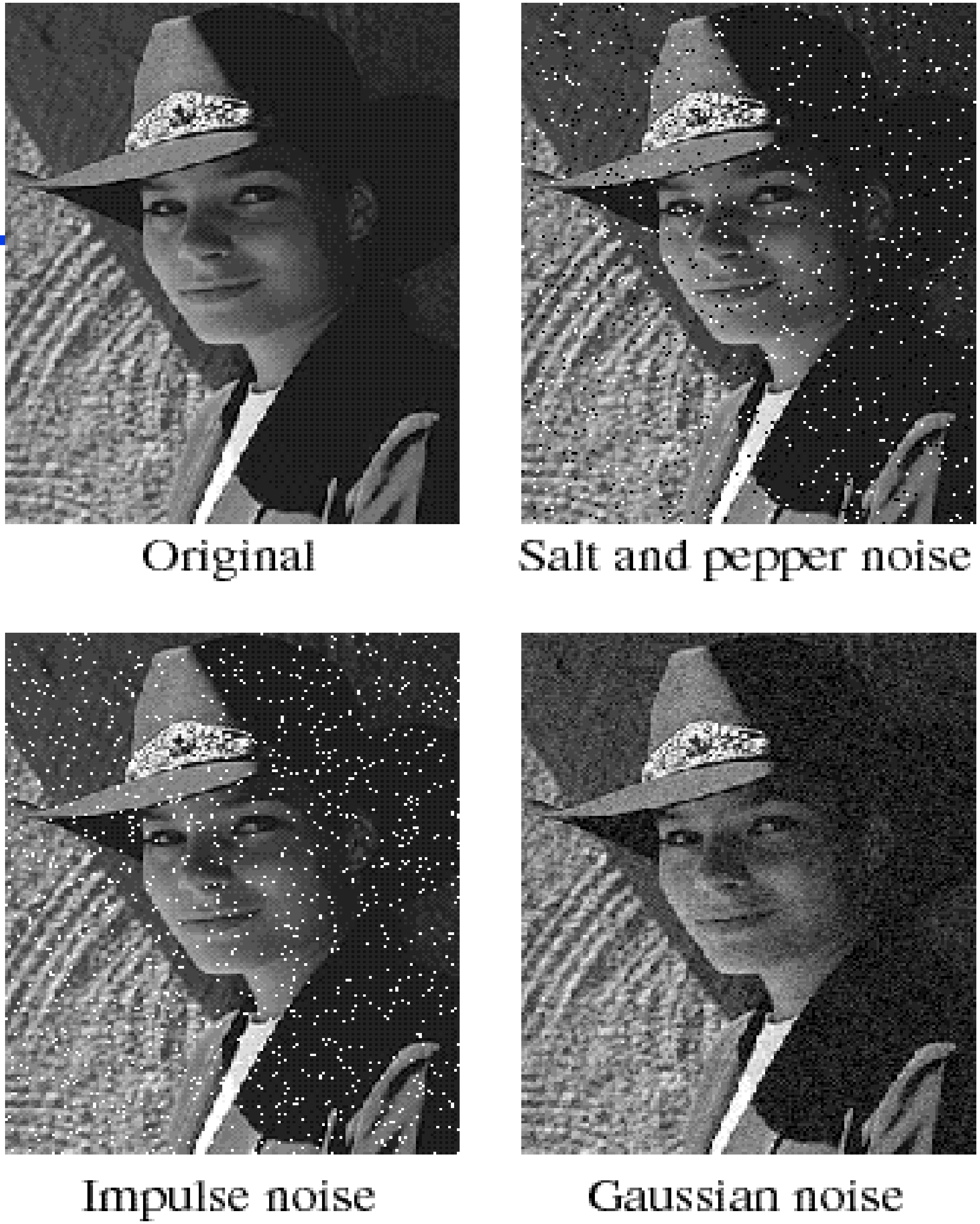
Noise

Filtering is useful for noise reduction...

(side effects: **blurring**)

Common types of noise:

- **Salt and pepper noise**: random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution



Practical noise reduction

How can we “smooth” away noise in a single image?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	100	130	110	120	110	0	0
0	0	0	110	90	100	90	100	0	0
0	0	0	130	100	90	130	110	0	0
0	0	0	120	100	130	110	120	0	0
0	0	0	90	110	80	120	100	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Neighborhood Processing (filtering)

Mean filtering

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$F[x, y]$$

$$G[x, y]$$

Mean filtering

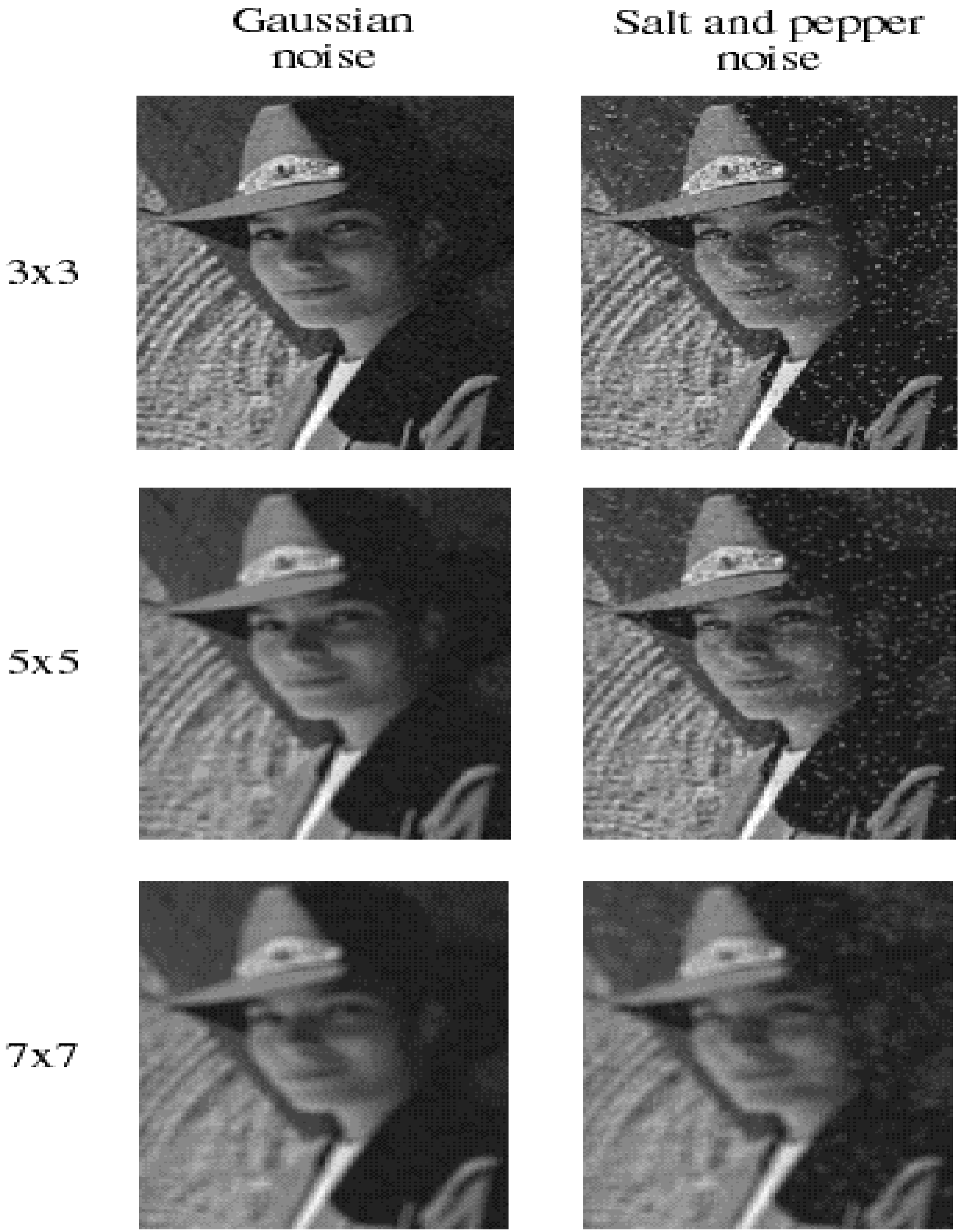
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$G[x, y]$

Effect of mean filters



Mean kernel

- What's the kernel for a 3x3 mean filter?

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x,y]$

$H[u,v]$

Gaussian Filtering

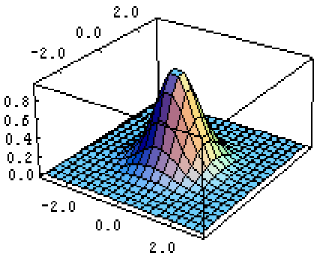
- A Gaussian kernel gives less weight to pixels further from the center of the window

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$F[x, y]$

$$\frac{1}{16} \cdot \boxed{\begin{array}{|c|c|c|} \hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}}$$

$H[u, v]$

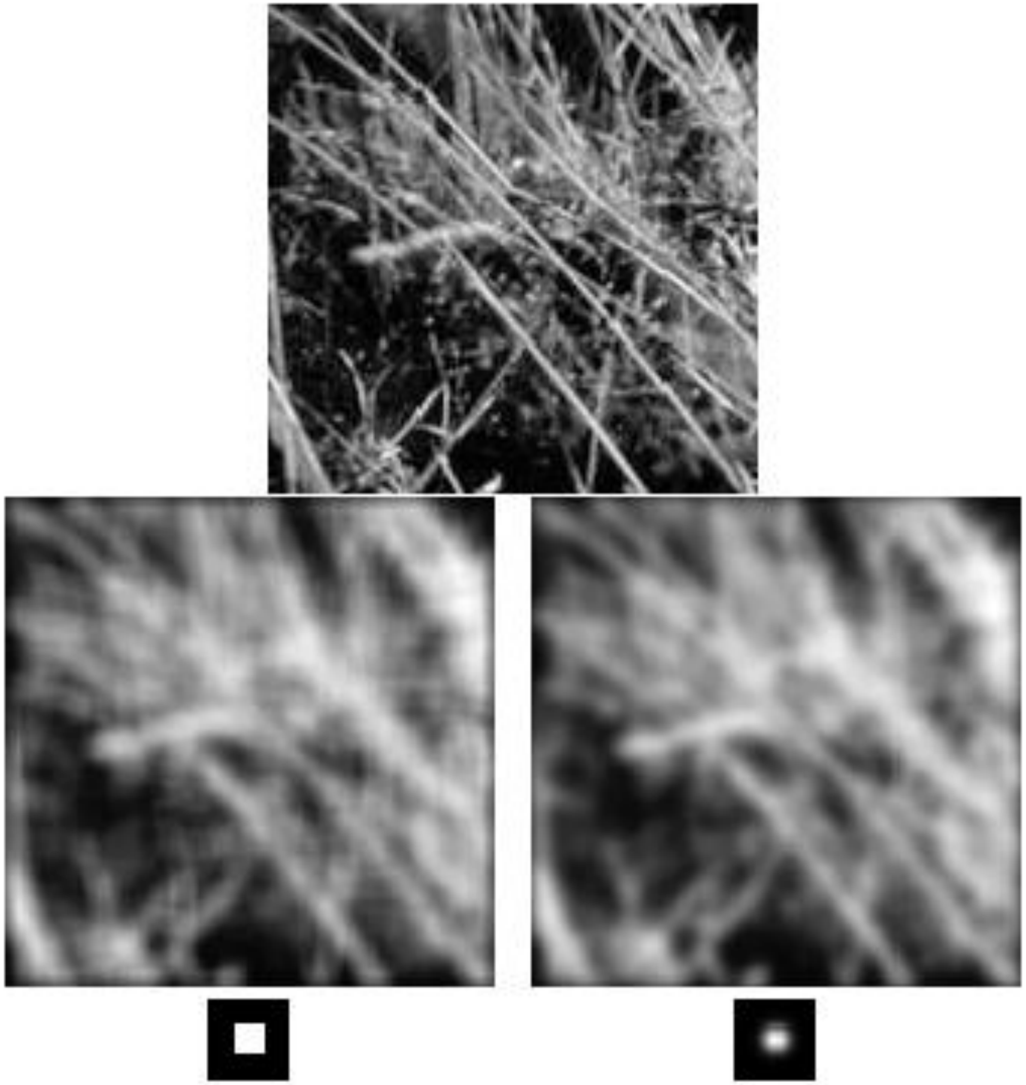


$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$

This kernel is an approximation of a Gaussian function:

Neighborhood Processing (filtering)

Mean vs. Gaussian filtering



Median filters

- A **Median Filter** operates over a window by selecting the median intensity in the window.

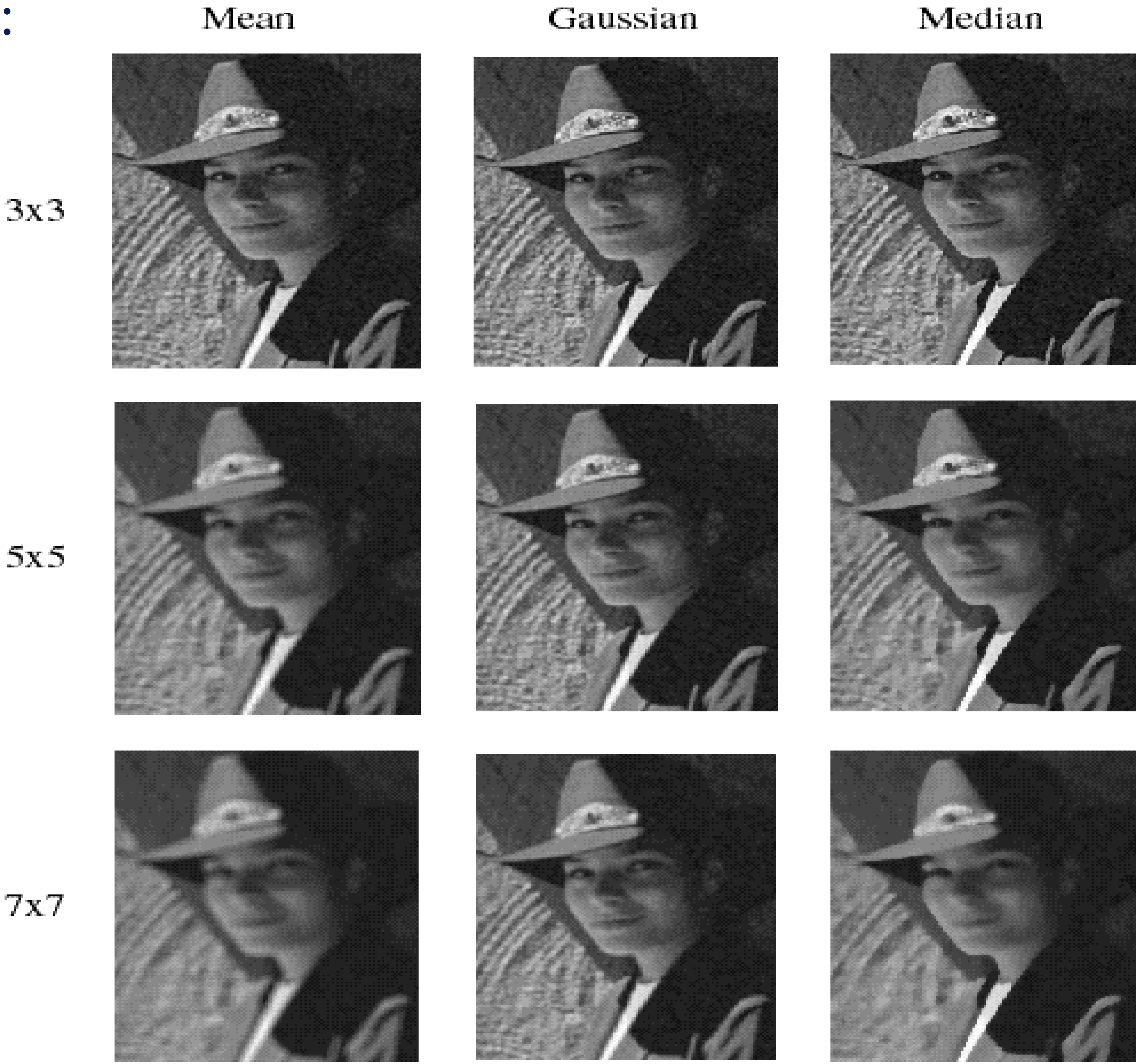
- What advantage does a median filter have over a mean filter?

- Is a median filter a kind of convolution?

  - No, median filter is an example of non-linear filtering

Comparison: salt and pepper noise

Comparison: Gaussian noise



Reading: Forsyth & Ponce, 8.1-8.2

Differentiation and convolution

- Recall

$$\frac{\partial}{\partial x} f = \lim_{\varepsilon \to 0} \left(\frac{f(x+\varepsilon, y) - f(x, y)}{\varepsilon} \right)$$

- Now this is linear and shift invariant, so must be the result of a convolution.

- We could approximate this as

$$\frac{\partial}{\partial x} f \approx \frac{f(x_{i+1}, y) - f(x_{i-1}, y)}{2 \cdot \Delta x}$$

$$= \nabla_x * f \quad \text{(convolution)}$$

with kernel $\frac{1}{2\Delta x} \cdot$

0	0	0
1	0	-1
0	0	0

$\nabla_x[u, v]$

sometimes this may not be a very good way to do things, as we shall see

Reading: Forsyth & Ponce, 8.1-8.2

Differentiation and convolution

- Recall

$$\frac{\partial}{\partial x} f = \lim_{\varepsilon \to 0}\left(\frac{f(x+\varepsilon, y) - f(x, y)}{\varepsilon} \right)$$

- Now this is linear and shift invariant, so must be the result of a convolution.

- We could approximate this as

$$\frac{\partial}{\partial x} f \approx \frac{f(x_{i+1}, y) - f(x_{i-1}, y)}{2 \cdot \Delta x}$$

$$= \nabla_x * f \quad \text{(convolution)}$$

with kernel $\frac{1}{2\Delta x} \cdot \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 1 & 0 & -1 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$

$\nabla_x[u, v]$

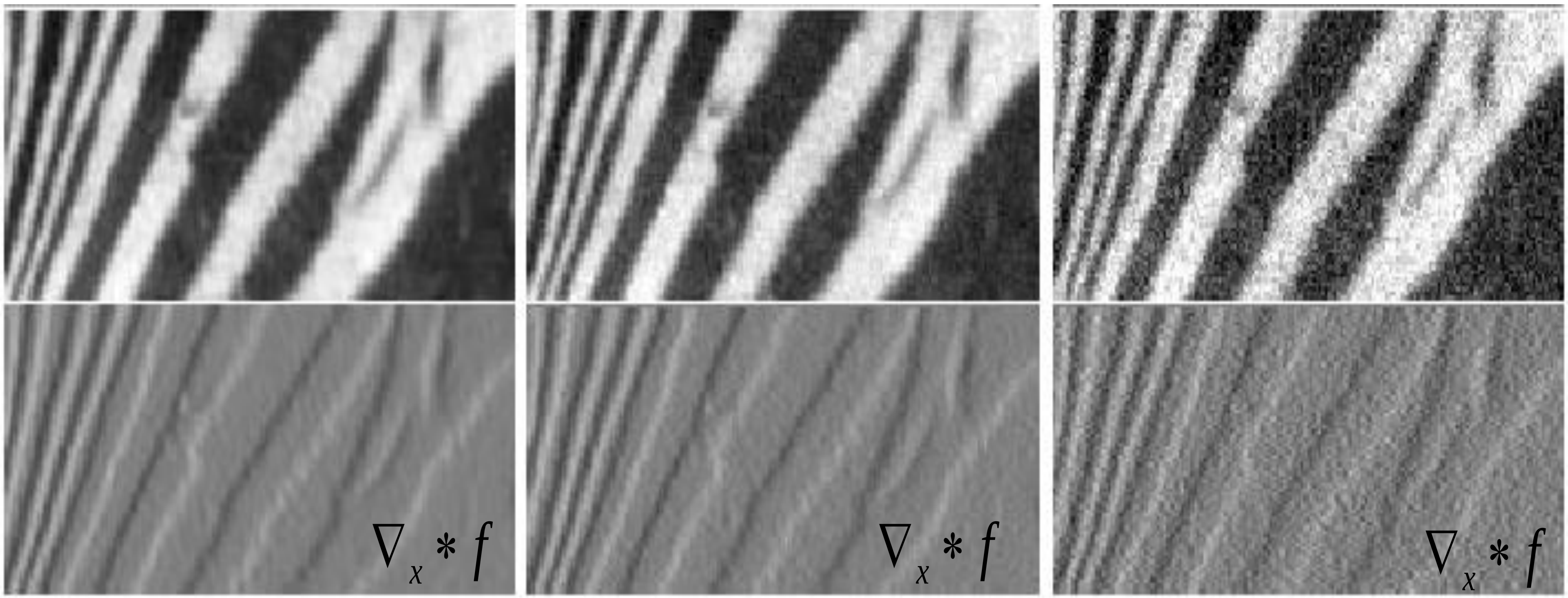
sometimes this may not be a very good way to do things, as we shall see

Finite differences

$$\nabla_x * f$$

Finite differences responding to noise

The University of Western Ontario



Increasing noise ->
(this is zero mean additive gaussian noise)

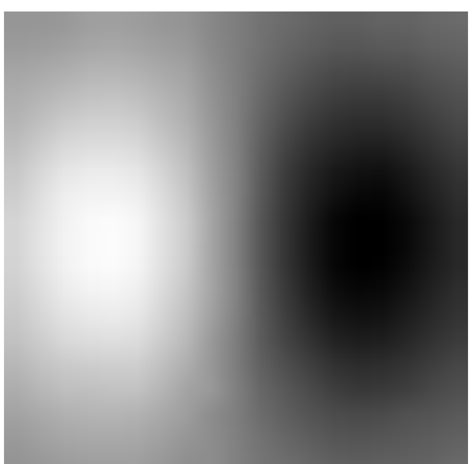
Finite differences and noise

- Finite difference filters respond strongly to noise
  - obvious reason: image noise results in pixels that look very different from their neighbours
- Generally, the larger the noise the stronger the response
- What is to be done?
  - intuitively, most pixels in images look quite a lot like their neighbours
  - this is true even at an edge; along the edge they're similar, across the edge they're not
  - suggests that smoothing the image should help, by forcing pixels different to their neighbours (=noise pixels?) to look more like neighbours

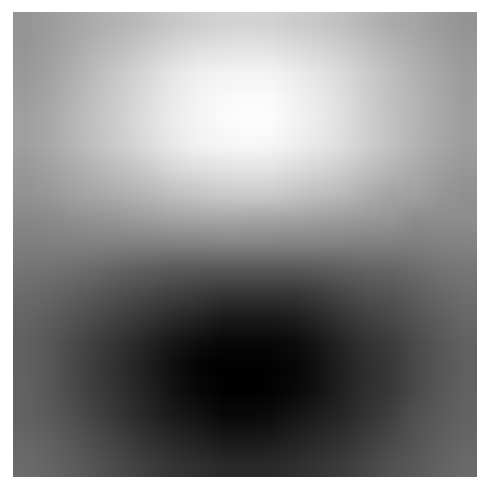
Smoothing and Differentiation

- Issue: noise
  - smooth before differentiation
  - two convolutions: smooth, and then differentiate?
  - actually, no - we can use a derivative of Gaussian filter
    - because differentiation is convolution, and convolution is associative $\nabla_x * (H * f) = (\nabla_x * H) * f$

$\nabla_x * H$

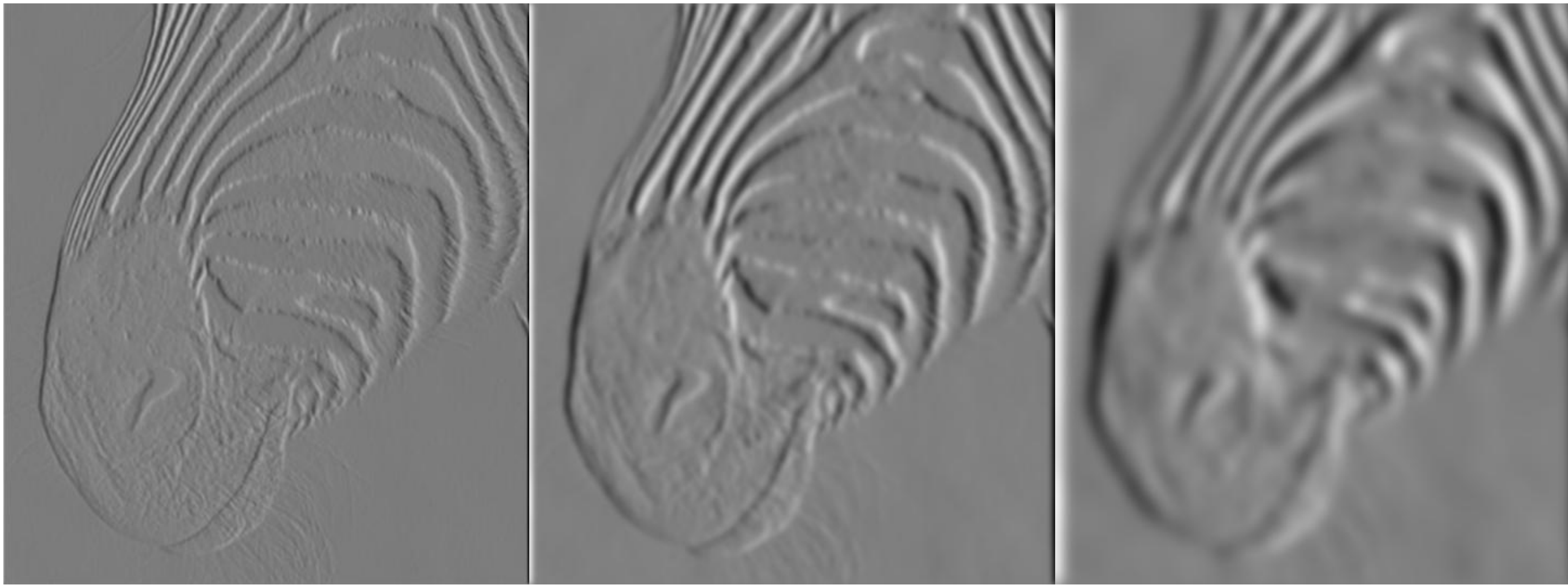


$\nabla_y * H$



The University of Western Ontario

$(\nabla_x * H) * f$



1 pixel 3 pixels 7 pixels

The scale of the smoothing filter affects derivative estimates, and also the semantics of the edges recovered.

Sobel derivative kernels

$$\frac{\partial}{\partial x} f \qquad\qquad \frac{\partial}{\partial y} f$$

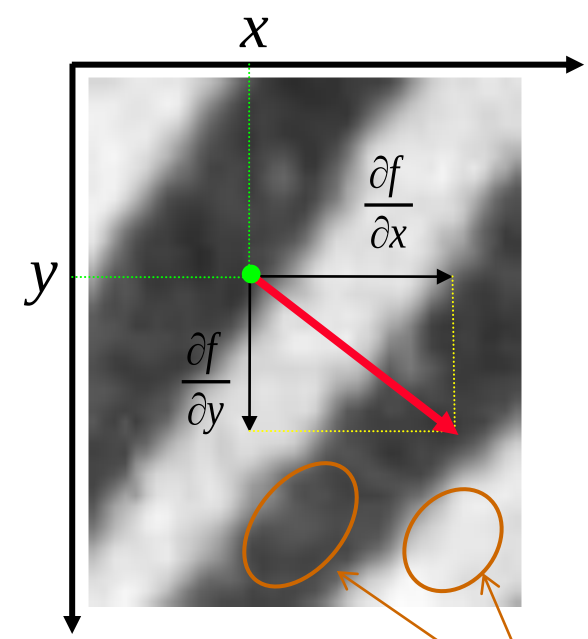
$$\frac{1}{8\Delta x} \cdot \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad\qquad \frac{1}{8\Delta y} \cdot \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$\nabla_x[u,v] \qquad\qquad \nabla_y[u,v]$$

The University of Western Ontario

Image Gradients

- Recall for a function of two (or more) variables $f(x, y)$



Gradient at point (x,y)

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \approx \begin{bmatrix} \nabla_x * f \\ \nabla_y * f \end{bmatrix}$$

a two (or more) dimensional vector

•The absolute value $|\nabla f| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2} \approx \sqrt{(\nabla_x * f)^2 + (\nabla_y * f)^2}$ is large at image boundaries

small image gradients in low textured areas

•The direction of the gradient corresponds to the direction of the "steepest ascend"

- normally gradient is orthogonal to object boundaries in the image.

Comment: vector ∇f is independent of specific coordinate system

The University of Western Ontario

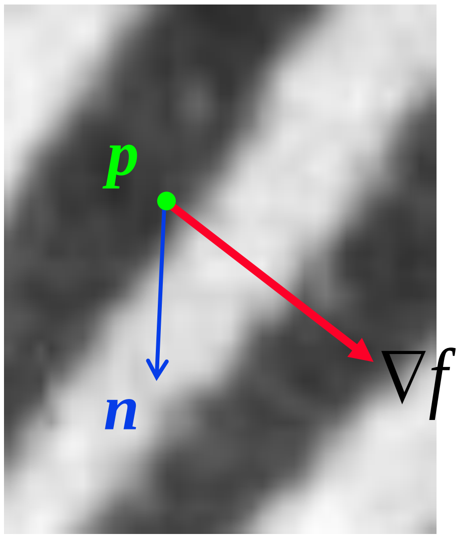
- Equivalently, *gradient* of function $f(p)$ at point $p \in R^2$ can be defined as a vector ∇f s.t. for any unit vector $\vec{n}$

Gradient at point *p*

$$\boxed{(\nabla f \cdot \vec{n}) = \frac{\partial f}{\partial \vec{n}}} \approx \frac{f(p+\varepsilon\cdot\vec{n}) - f(p)}{\varepsilon}$$

dot product

directional derivative of function ***f*** along direction ***n***

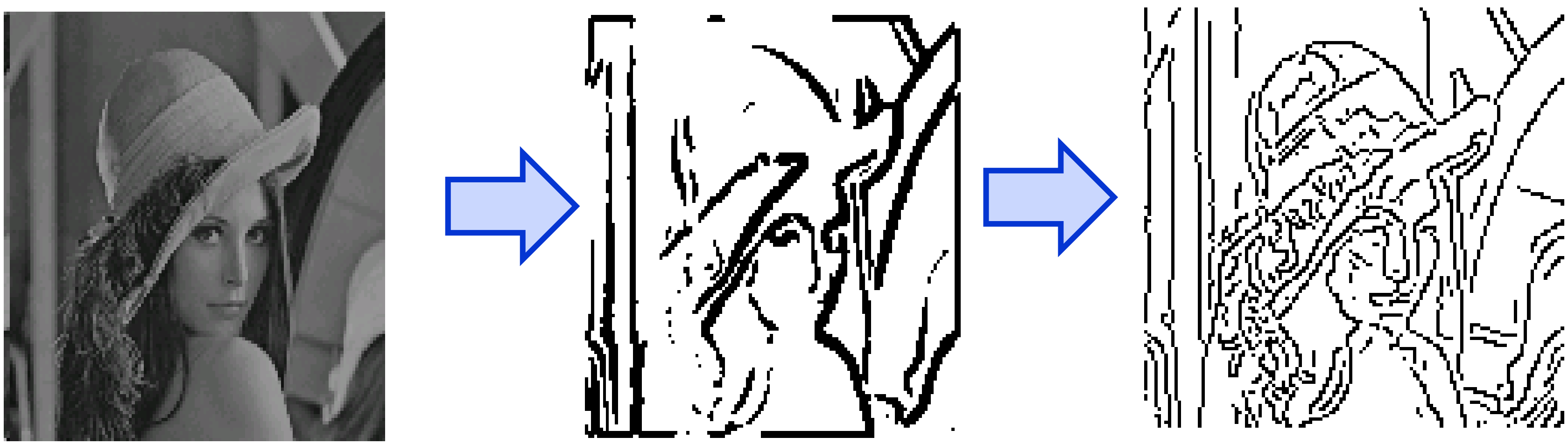


- pure vector algebra, specific coordinate system is irrelevant
- works for functions of two, three, or any larger number of variables
- previous slide gives a specific way for computing coordinates $(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$ of vector ∇f w.r.t. given orthogonal basis (axis X and Y).

Image Gradient

- Typical application where image gradients are used is *image edge detection*
  - find points with large image gradients

"edge features"



Canny edge detector suppresses non-extrema Gradient points

Second Image Derivatives
(Laplace operator Δf)

$$\Delta f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = [\tfrac{\partial}{\partial x} \quad \tfrac{\partial}{\partial y}] \cdot \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \cdot f = \nabla \cdot \nabla f$$

rotationally invariant
second derivative for 2D functions

"divergence of gradient"

$$\underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix}}_{\begin{bmatrix} 0 & 0 & 0 \\ 0 & -1 & 1 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ -1 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}} + \underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}}_{\begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 0 \end{bmatrix}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

rate of change for the *rate of change* in *x*-direction

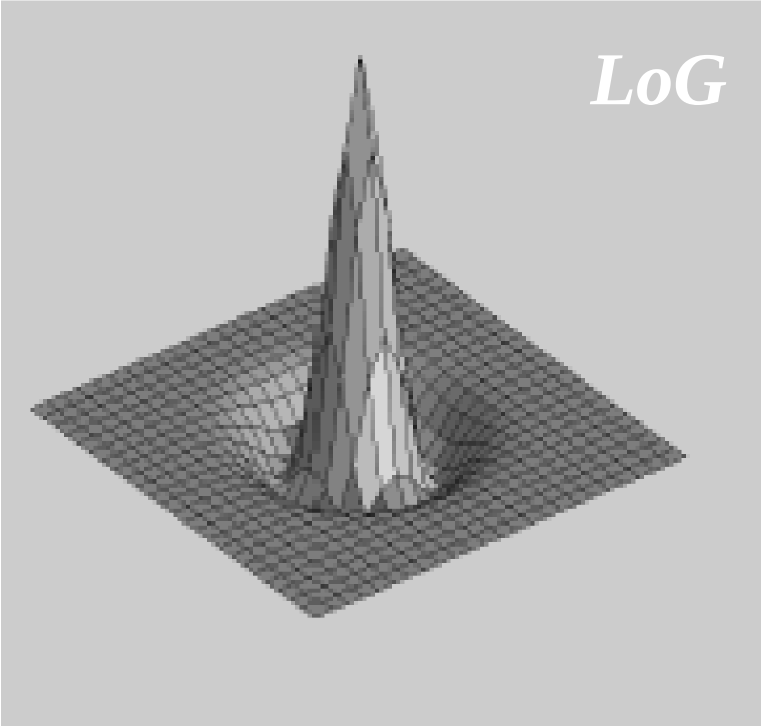
rate of change for the *rate of change* in *y*-direction

$$\frac{\partial^2 f}{\partial x^2} = \frac{\partial f}{\partial x}\left(\tfrac{+1}{2}\right) - \frac{\partial f}{\partial x}\left(\tfrac{-1}{2}\right) \qquad \frac{\partial^2 f}{\partial y^2} = \frac{\partial f}{\partial y}\left(\tfrac{+1}{2}\right) - \frac{\partial f}{\partial y}\left(\tfrac{-1}{2}\right)$$

Laplacian of a Gaussian (LoG)

$\Delta * G$ image should be smoothed a bit first

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



MATLAB: logfilt = fspecial('log',25,4);

http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

The University of Western Ontario

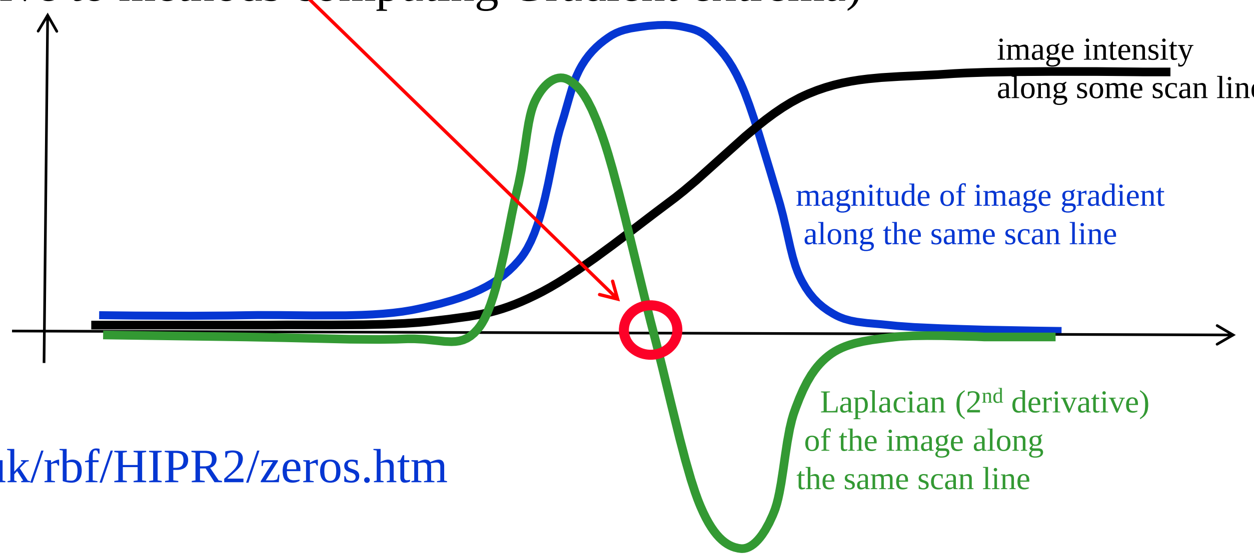
Second Image Derivatives
(Laplace operator Δf)

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

For simplicity, assume $f(x,y) = const(y)$. Then, Laplacian of f is simply a second derivative of $f(x) = f(x,y)$

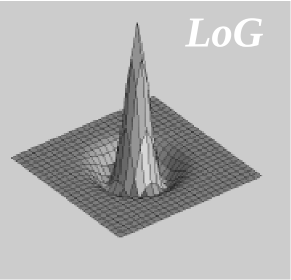
Application: *Laplacian Zero Crossings* are used for edge detection

(alternative to methods computing Gradient extrema)

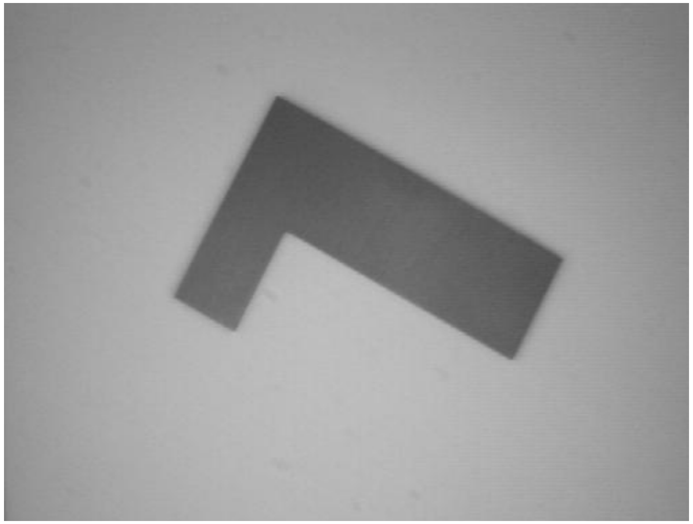


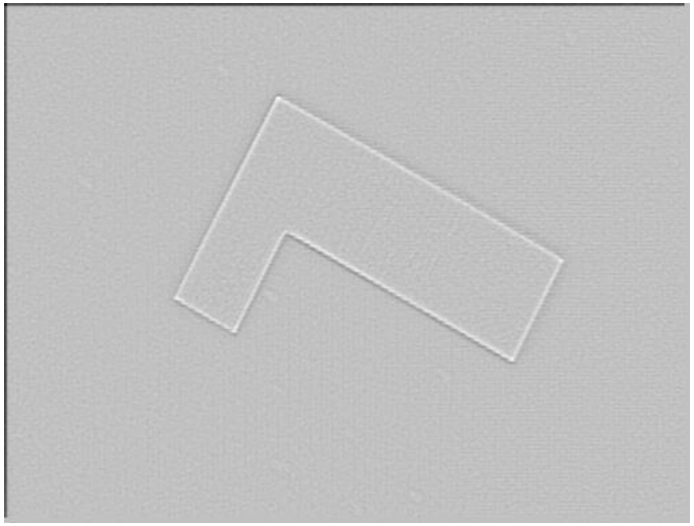
http://homepages.inf.ed.ac.uk/rbf/HIPR2/zeros.htm

Laplacian of a Gaussian (LoG)



$\Delta * G$ image should be smoothed a bit first





http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm

The University of Western Ontario

Unsharp masking

- What does blurring take away?

$$U = I - G * I$$

-

=

$+\alpha$

=

The University of Western Ontario

Unsharp masking

$$(1+\alpha)I - \alpha \cdot G * I \quad \approx \quad [(1+\alpha)G_{\sigma 1} - \alpha \cdot G_{\sigma 2}] * I$$

$\sigma_1 << \sigma_2$

$+\alpha$

$U = I - G * I$

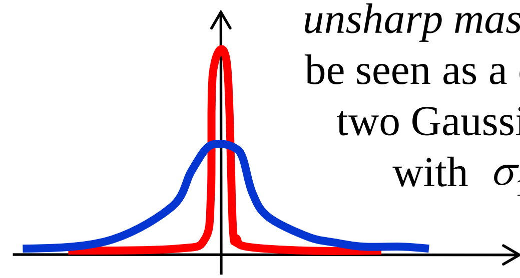
$=$

$I + \alpha \cdot U$

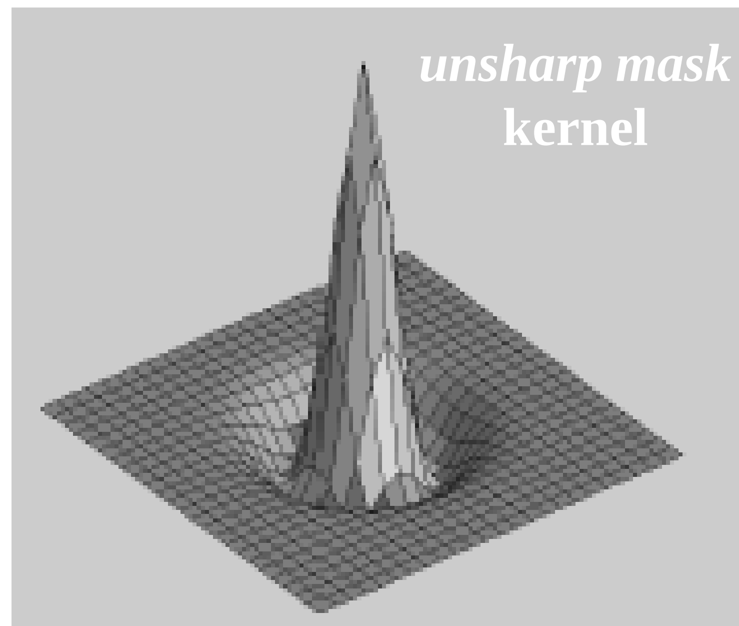
Unsharp masking

MATLAB

```
Imrgb = imread(‘file.jpg’);
im = im2double(rgb2gray(imrgb));
g= fspecial('gaussian', 25,4);
imblur = conv2(im,g,‘same');
imagesc([im imblur])
imagesc([im im+.4*(im-imblur)])
```



unsharp mask kernel can be seen as a difference of two Gaussians (DoG) with $\sigma_1 << \sigma_2$.



$$DoG \equiv G_{\sigma 1} - G_{\sigma 2} \approx LoG$$

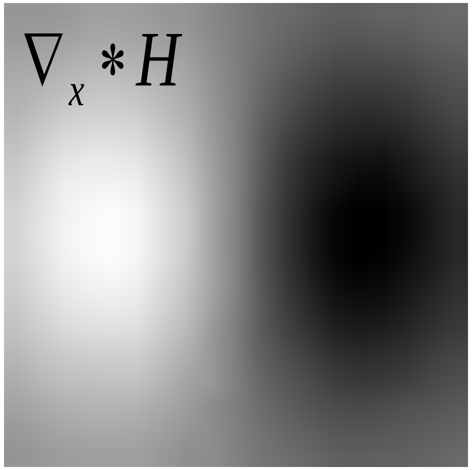
http://homepages.inf.ed.ac.uk/rbf/HIPR2/unsharp.htm

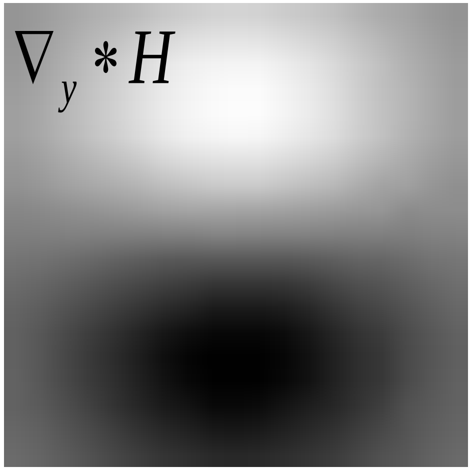
Reading: Forsyth & Ponce ch.7.5

Filters and Templates

- Applying a filter at some point can be seen as taking a dot-product between the image and some vector
- Filtering the image is a set of dot products

- Insight
  - filters may look like the effects they are intended to find
  - filters find effects they look like





Normalized Cross-Correlation (NCC)

The University of Western Ontario

- filtering as a **dot product**
- now **measure the angle:**

  NCC output is filter output divided by root of the sum of squares of values over which filter lies

cross-correlation of h and f at $t=(x,y)$

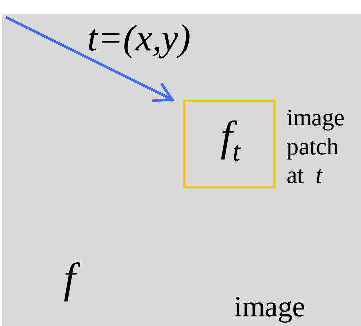
$$\frac{\sum_{u=-k}^{k}\sum_{v=-k}^{k} h[u,v]\cdot f[x+u,y+v]}{|h|\cdot|f_t|}$$

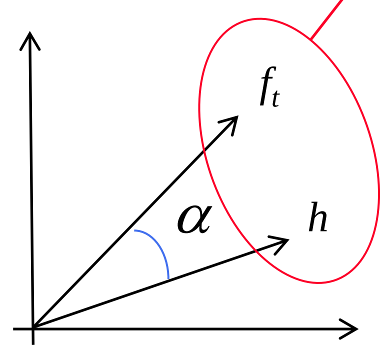
division makes this a non-linear operation

h template (filter, kernel, mask) of size n = (2k+1) x (2k+1)

dot product

$h \cdot f_t = \sum_{i=1}^{n} h[i] f_t[i]$





raster-scan h and f_t as vectors in R^n

$$g[t] = \frac{h\cdot f_t}{|h|\cdot|f_t|} = \cos(\alpha)$$

vector lengths $|z| = \sqrt{\sum_{i=1}^{n} z_i^2}$

Normalized Cross-Correlation (NCC)

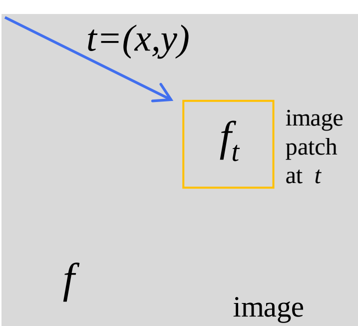
- filtering as a **dot product**
- now **measure the angle:**

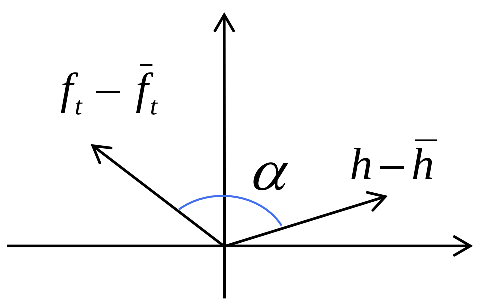
  NCC output is filter output divided by root of the sum of squares of values over which filter lies

- **Tricks:**
  - subtract *template average* $\bar{h}$

    (to give zero output for constant regions, reduces response to irrelevant background)
  - subtract *patch average* $\bar{f}_t$ when computing the normalizing constant (i.e. subtract the image mean in the neighborhood)

h template (filter, kernel, mask) of size n = (2k+1) x (2k+1)





these vectors do not have to be in the "positive" quadrant

$$\boxed{g[t] = \frac{(h-\bar{h})\cdot(f_t - \bar{f}_t)}{|h-\bar{h}|\cdot|f_t - \bar{f}_t|}}$$

NCC

Normalized Cross-Correlation (NCC)

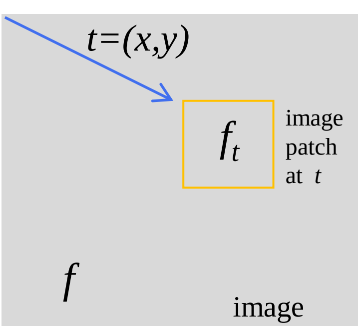
- filtering as a **dot product**
- now **measure the angle:**

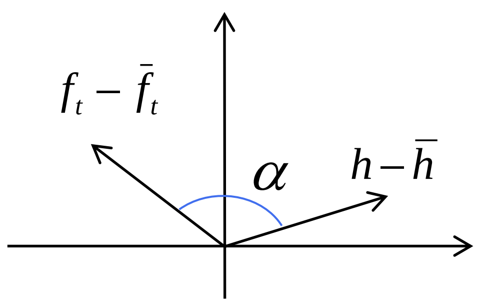
  NCC output is filter output divided by root of the sum of squares of values over which filter lies

- Tricks:
  - subtract *template average* $\bar{h}$

    (to give zero output for constant regions, reduces response to irrelevant background)
  - subtract *patch average* $\bar{f}_t$ when computing the normalizing constant (i.e. subtract the image mean in the neighborhood)

h template (filter, kernel, mask) of size n = (2k+1) x (2k+1)





these vectors do not have to be in the “positive” quadrant

equivalently using statistical term σ (standard diviation)

NCC

$$g[t] = \frac{(h-\bar{h})\cdot(f_t - \bar{f}_t)}{n\cdot\sigma_h\cdot\sigma_{f_t}} = cov(h, f_t)$$

Remember: st.div. $\sigma_z \equiv \sqrt{\frac{1}{n}\sum_{i=1}^{n}(z_i - \bar{z})^2} = \sqrt{\frac{1}{n}}\cdot|z - \bar{z}|$

Normalized Cross-Correlation (NCC)

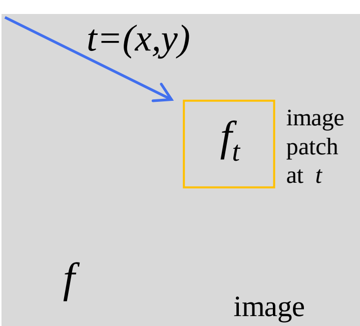
- filtering as a **dot product**
- now **measure the angle:**

  NCC output is filter output divided by root of the sum of squares of values over which filter lies

- Tricks:
  - subtract *template average* $\bar{h}$

    (to give zero output for constant regions, reduces response to irrelevant background)
  - subtract *patch average* $\bar{f}_t$ when computing the normalizing constant (i.e. subtract the image mean in the neighborhood)

h template (filter, kernel, mask) of size n = (2k+1) x (2k+1)



standard in statistics *correlation coefficient*

$\rho \longleftrightarrow$

between h and f_t

equivalently using statistical term cov (covariance)

$$\boxed{g[t] \quad = \quad \frac{\mathrm{cov}(h, f_t)}{\sigma_h \cdot \sigma_{f_t}}}$$

NCC

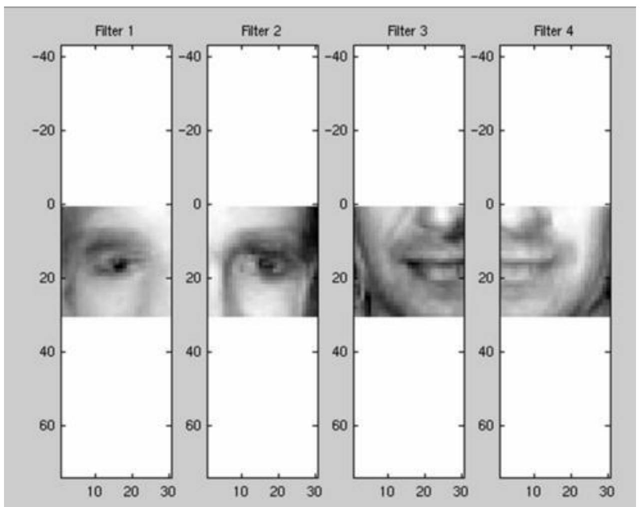
$$\mathrm{cov}(a,b) \equiv E(a-\bar{a})(b-\bar{b}) = \tfrac{1}{n}\sum_{i=1}^{n}(a_i-\bar{a})(b_i-\bar{b}) = \frac{(a-\bar{a})\cdot(b-\bar{b})}{n}$$

pictures from Silvio Savarese

The University of Western Ontario

Normalized Cross-Correlation (NCC)

h_A h_B h_C h_D



templates

NCC for h and f

A

B

C

D

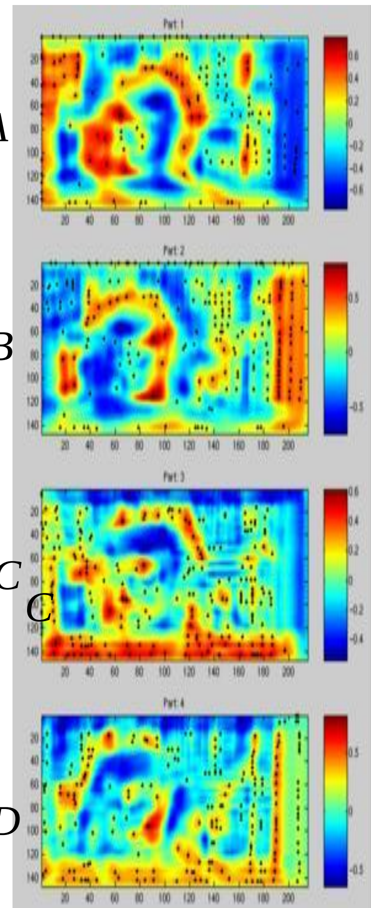
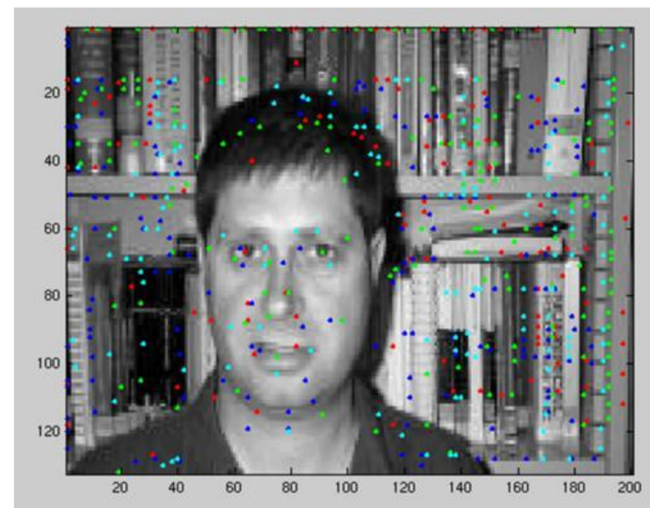


image f



points mark local maxima of NCC for each template

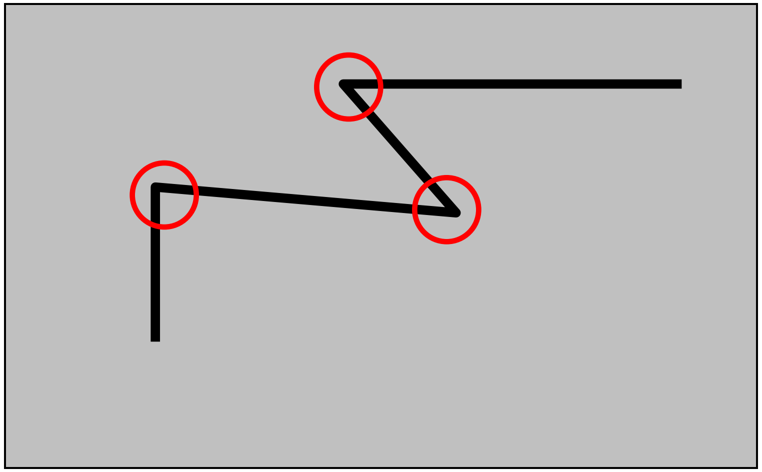
points of interest or **feature points**

Other features… (Szeliski sec 4.1.1)

- Feature points are used for:
  - Image alignment (homography, fundamental matrix)
  - 3D reconstruction
  - Motion tracking
  - Object recognition
  - Indexing and database retrieval
  - Robot navigation
  - … other

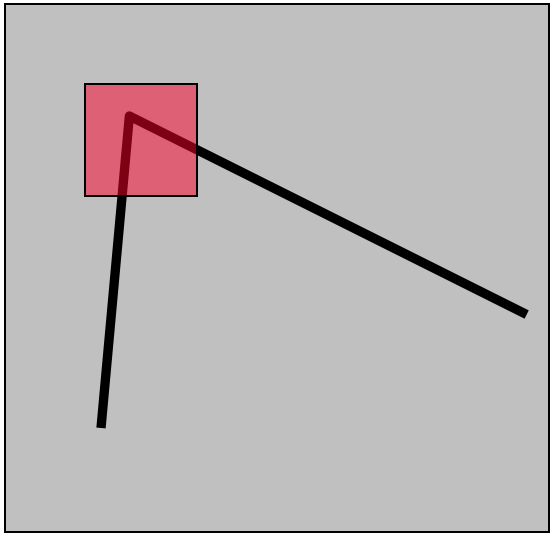
Harris corner detector

- C.Harris, M.Stephens. "A Combined Corner and Edge Detector". 1988



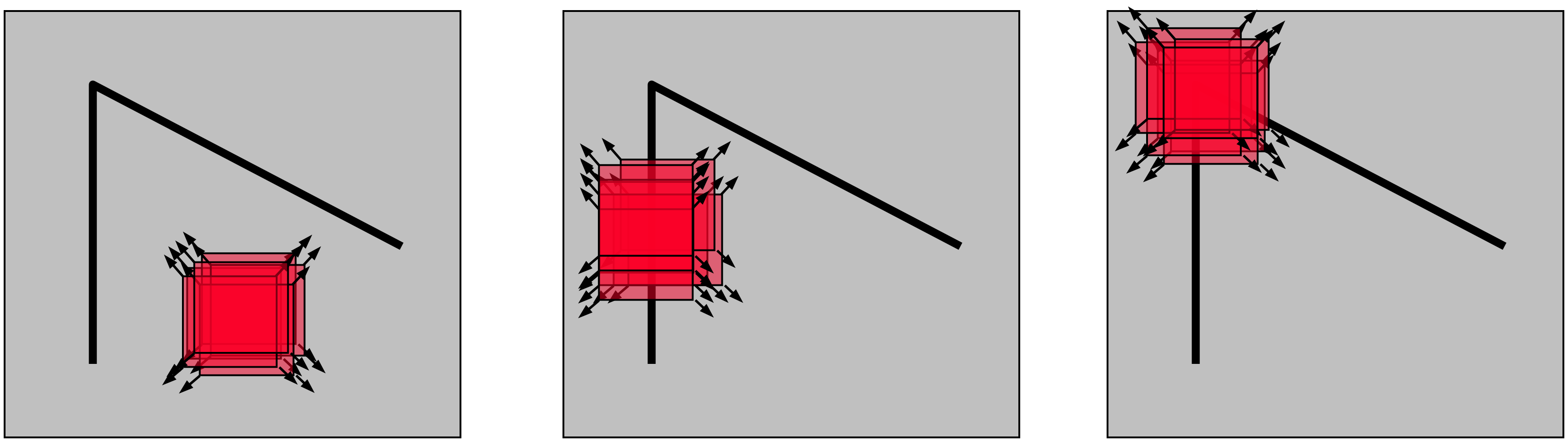
The Basic Idea

- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give *a large change* in intensity



Harris Detector: Basic Idea

The University of Western Ontario



“flat” region:
no change in all directions

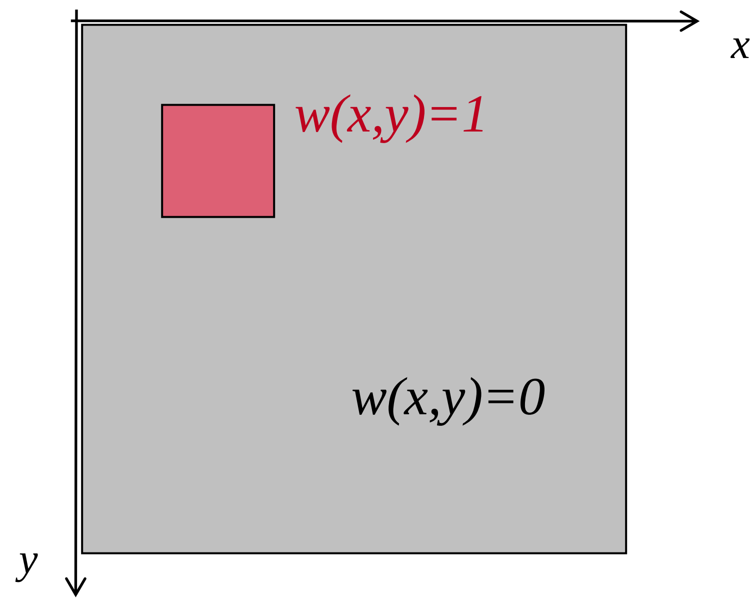
“edge”:
no change along the edge direction

“corner”:
significant change in all directions

Harris Detector: Mathematics

For any given image patch or window w we should measure how it changes when shifted by $ds = \begin{bmatrix} u \\ v \end{bmatrix}$

Notation: let patch be defined by its support function $w(x,y)$ over image pixels



Harris Detector: Mathematics

patch *w* change measure for shift $ds = \begin{bmatrix} u \\ v \end{bmatrix}$:

weighted sum of squared differences

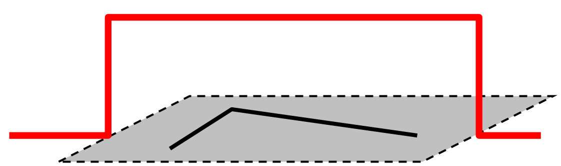
$$E_w(u,v) := \sum_{x,y} w(x,y)\cdot[I(x+u,y+v)-I(x,y)]^2$$

Window function

Shifted intensity

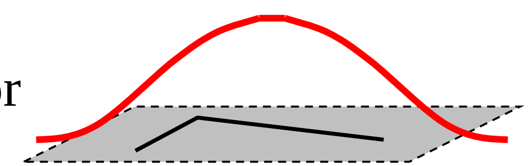
Intensity

NOTE: window support functions $w(x,y)$ =



1 in window, 0 outside

or



Gaussian (weighted) support

Harris Detector: Mathematics

Change of intensity for the shift $ds = \begin{bmatrix} u \\ v \end{bmatrix}$ assuming image gradient $\nabla I \equiv \begin{bmatrix} I_x \\ I_y \end{bmatrix}$

$$I(x+u, y+v) - I(x,y) \;\approx\; I_x \cdot u + I_y \cdot v \;=\; ds^T \cdot \nabla I$$

rate of change for I at (x,y) in direction (u,v) = ds

(remember gradient definition on earlier slides!!!!)

this is 2D analogue of 1st order Taylor expansion

$$[\, I(x+u, y+v) - I(x,y) \,]^2 \;\approx\; ds^T \cdot \nabla I \cdot \nabla I^T \cdot ds$$

$$E_w(u,v) \;=\; \sum_{x,y} w(x,y) \cdot [I(x+u, y+v) - I(x,y)]^2$$

$$\approx\; ds^T \cdot \underbrace{\left(\sum_{x,y} w(x,y) \cdot \nabla I \cdot \nabla I^T \right)}_{M_w} \cdot ds \;=\; ds^T \cdot M_w \cdot ds$$

Harris Detector: Mathematics

Change of intensity for the shift $ds = \begin{bmatrix} u \\ v \end{bmatrix}$ assuming image gradient $\nabla I \equiv \begin{bmatrix} I_x \\ I_y \end{bmatrix}$

$$E_w(u,v) \cong [u \;\; v] \cdot M_w \cdot \begin{bmatrix} u \\ v \end{bmatrix} = ds^T \cdot M_w \cdot ds$$

where M_w is a 2×2 matrix computed from image derivatives inside patch w

matrix M is also called
Harris matrix or ***structure tensor***

$$\ldots \cdot \underbrace{\left(\sum_{x,y} w(x,y) \cdot \overbrace{\nabla I \cdot \nabla I^T}^{\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}} \right)}_{M_w} \cdot \ldots$$

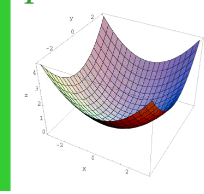
This tells you how to compute M_w at any window w (t.e. any image patch)

Harris Detector: Mathematics

Change of intensity for the shift $ds = \begin{bmatrix} u \\ v \end{bmatrix}$ assuming image gradient $\nabla I \equiv \begin{bmatrix} I_x \\ I_y \end{bmatrix}$

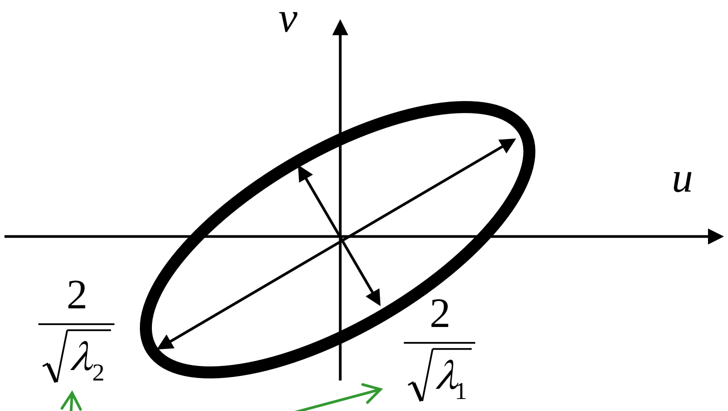
$$E_w(u,v) \cong [u \;\; v] \cdot M_w \cdot \begin{bmatrix} u \\ v \end{bmatrix} = ds^T \cdot M_w \cdot ds$$

paraboloid



M is a positive semi-definite matrix (**Exercise**: show that $ds^T \cdot M \cdot ds \geq 0$ for any ds)

M can be analyzed via its *isolines*, e.g. $ds^T \cdot M_w \cdot ds = 1$ (ellipsoid)

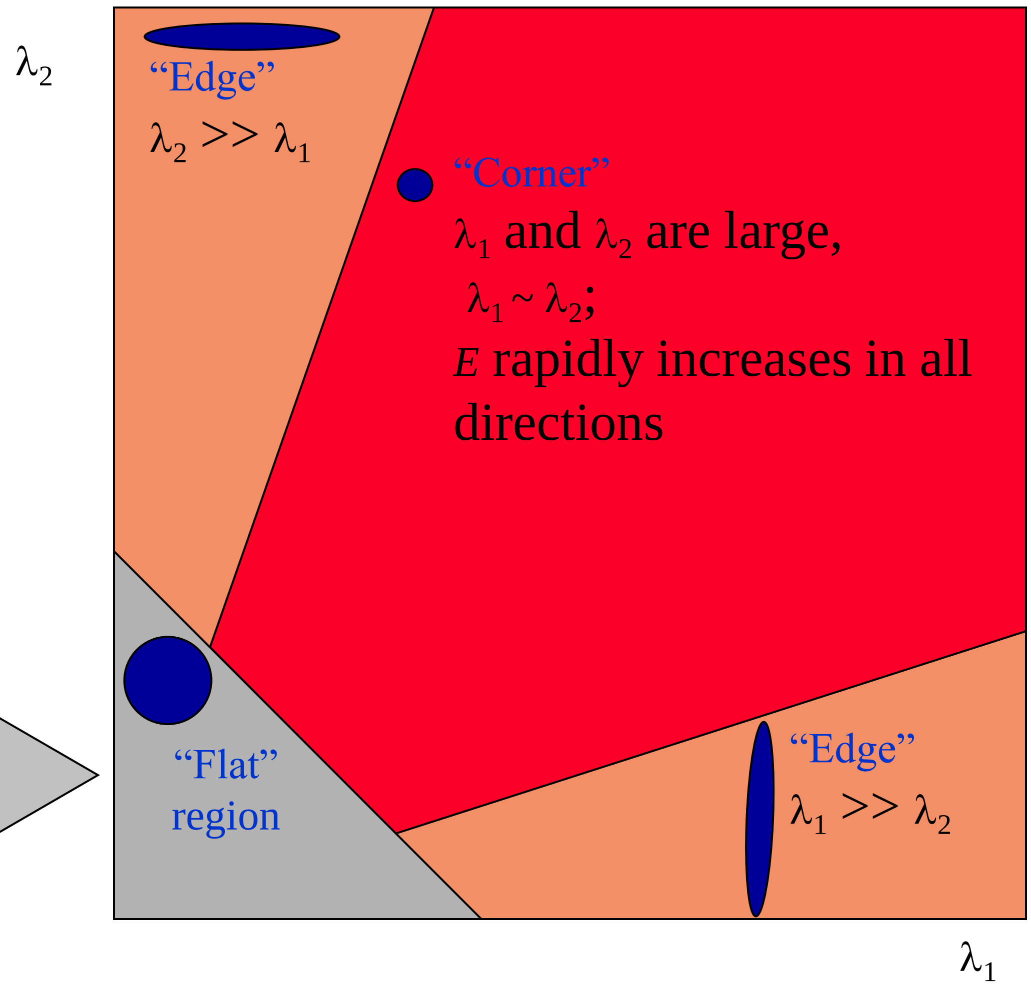


two eigen values of matrix M_w

Points on this ellipsoid are shifts $ds=(u,v)$ giving the same value of energy $E(u,v)=1$. Thus, the ellipsoid allows to visually compare sensitivity of energy E to shifts ds in different directions

Harris Detector: Mathematics

Classification of image points using eigenvalues of M:



λ_1 and λ_2 are small;
E is almost constant in all directions

Harris Detector: Mathematics

Measure of corner response:

$$R = \frac{\det M}{\operatorname{Trace} M}$$

$$\det M = \lambda_1 \lambda_2$$
$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

***R* should be large**
(it implies that both λ are far from zero)

Harris Detector

- The Algorithm:
  - Find points with large corner response function R

    $R >$ threshold

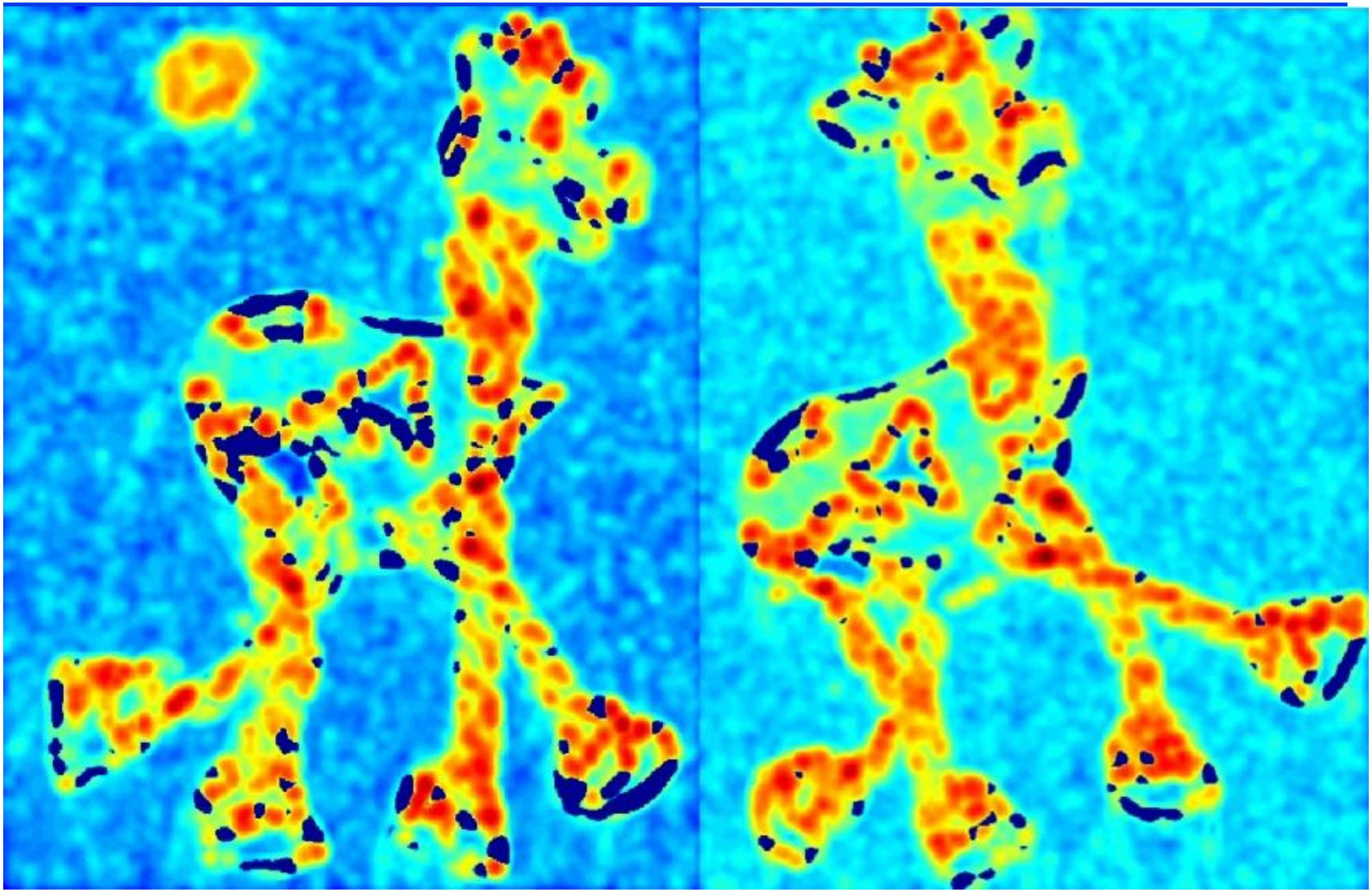
  - Take the points of local maxima of R

Harris Detector: Workflow

Harris Detector: Workflow

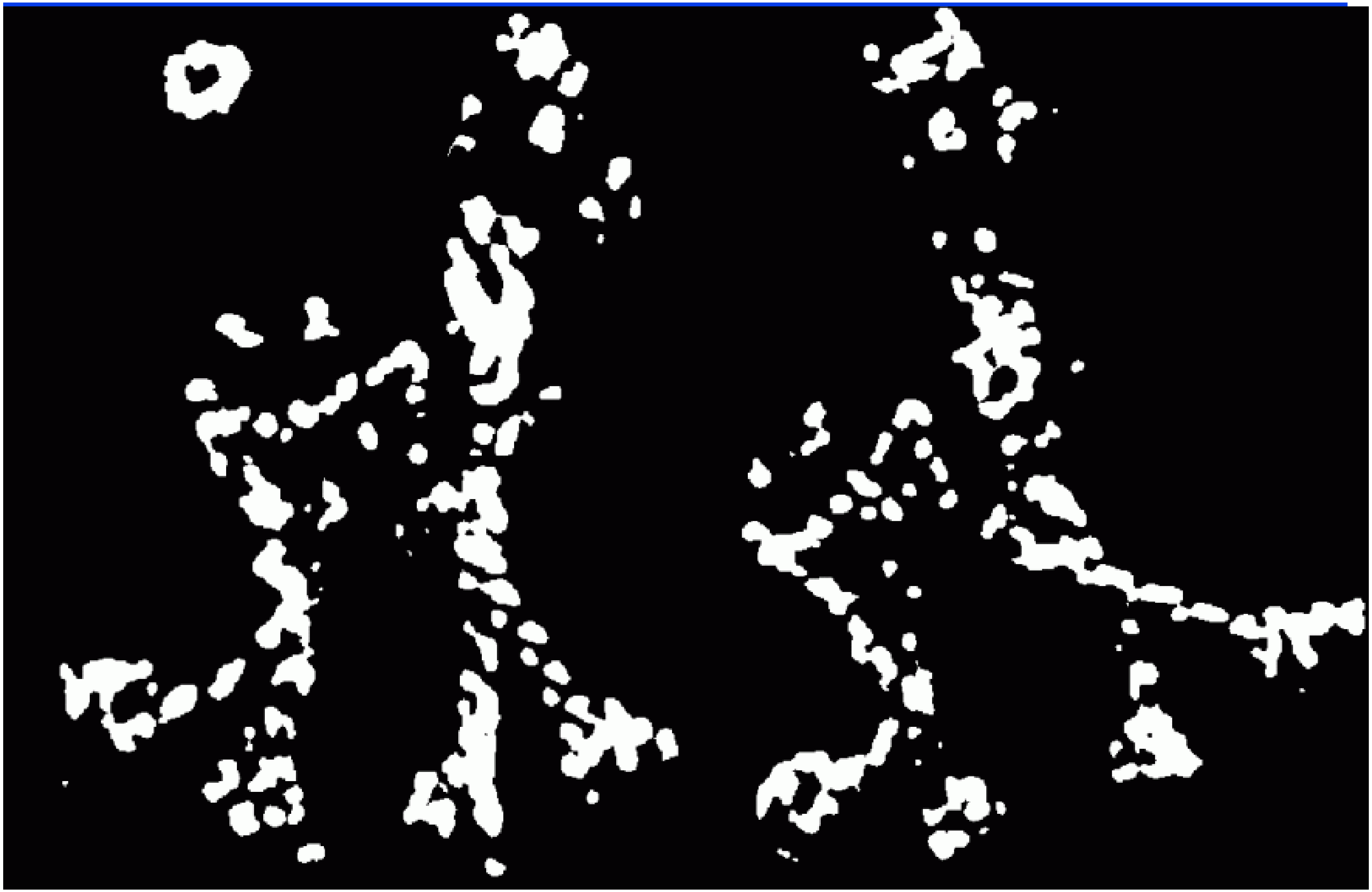
The University of Western Ontario

Compute corner response R



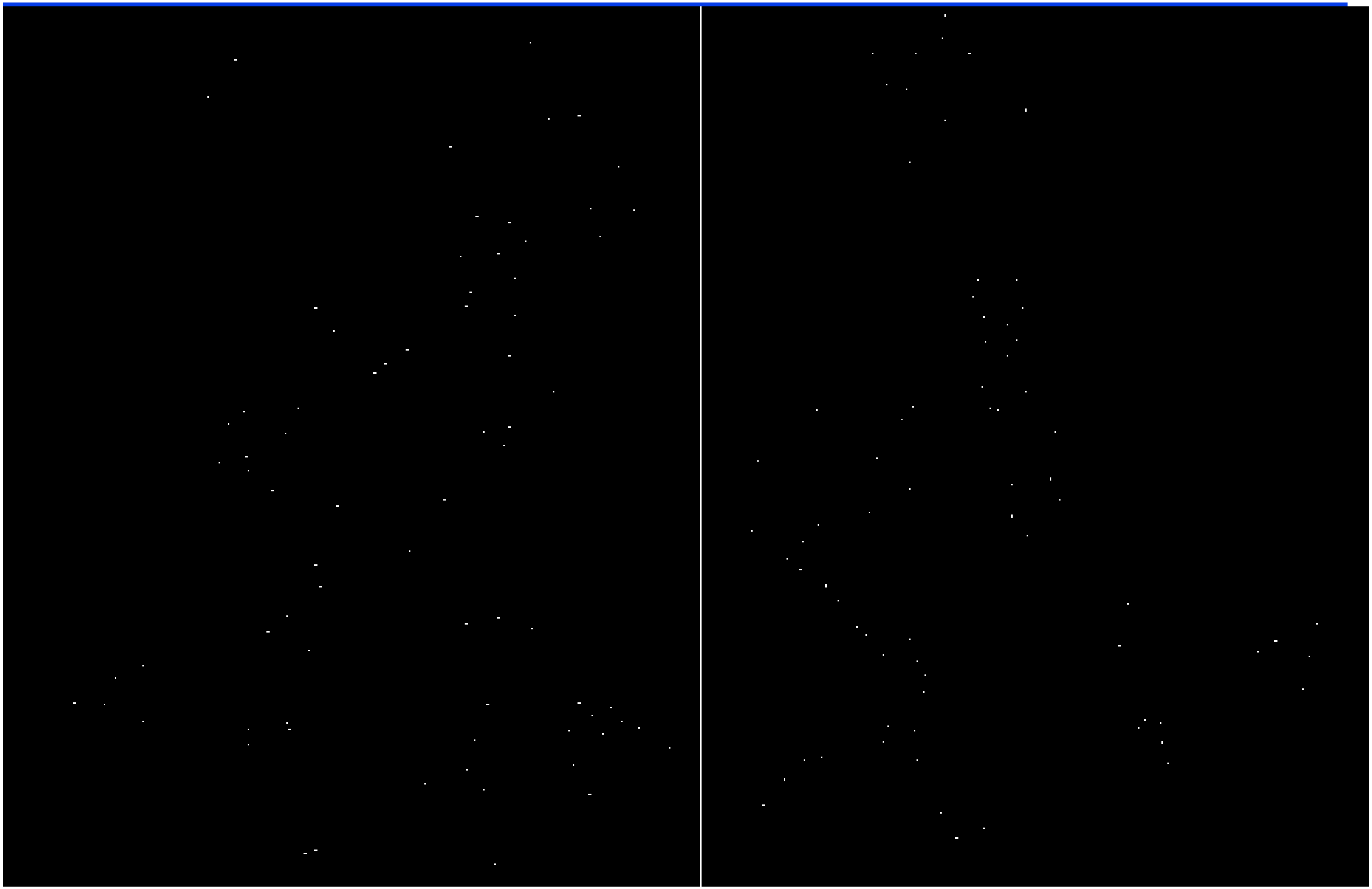
Harris Detector: Workflow

Find points with large corner response: $R > \text{threshold}$



Harris Detector: Workflow

Take only the points of local maxima of R



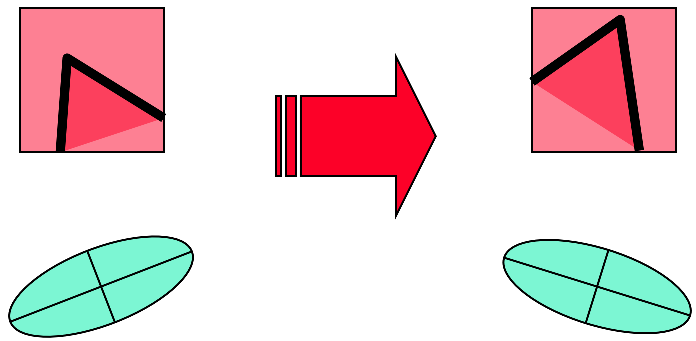
Harris Detector: Workflow

The University of Western Ontario

The University of
Western
Ontario

Harris Detector: Some Properties

- Rotation invariance



Ellipse rotates but its shape (i.e. eigenvalues) remains the same

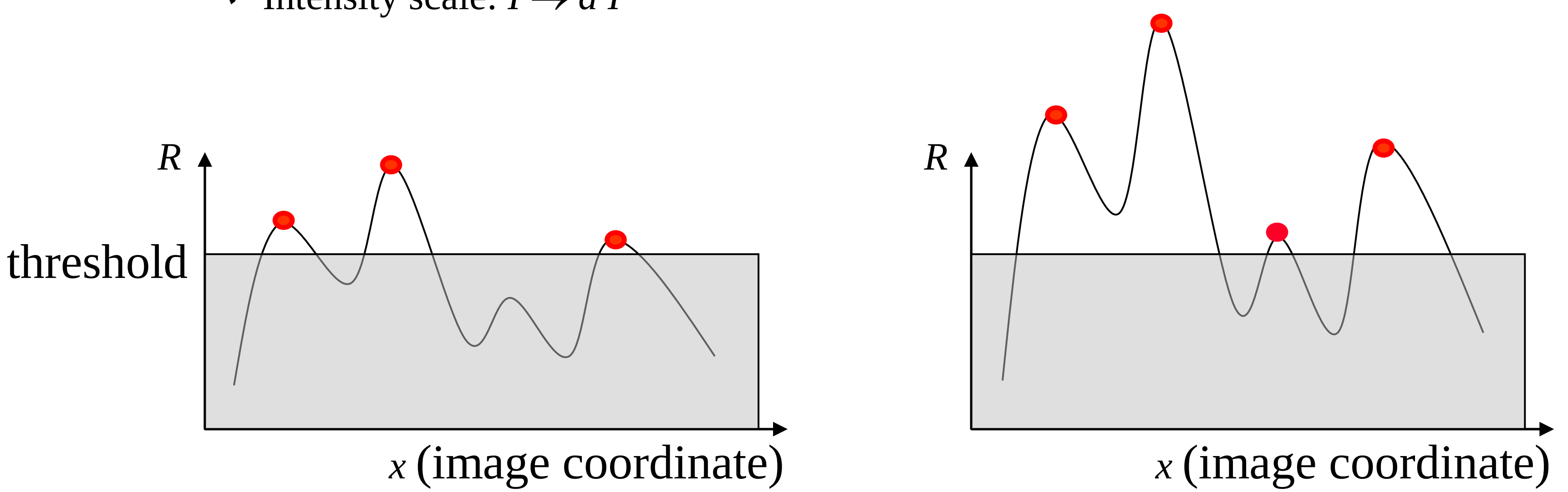
Corner response R is invariant to image rotation

Harris Detector: Some Properties

- Partial invariance to *affine intensity* change

✓ Only derivatives are used => invariance to intensity shift $I \rightarrow I + b$

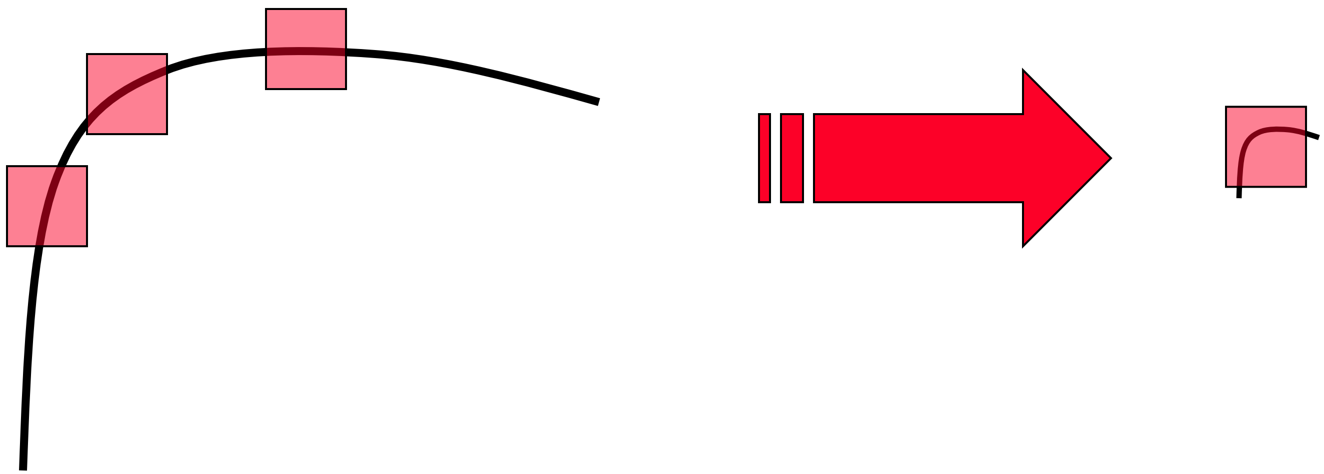
✓ Intensity scale: $I \rightarrow a\, I$



features locations stay the same,
but some may appear or disappear depending on gain a

Harris Detector: Some Properties

- But: non-invariant to *image scale*!

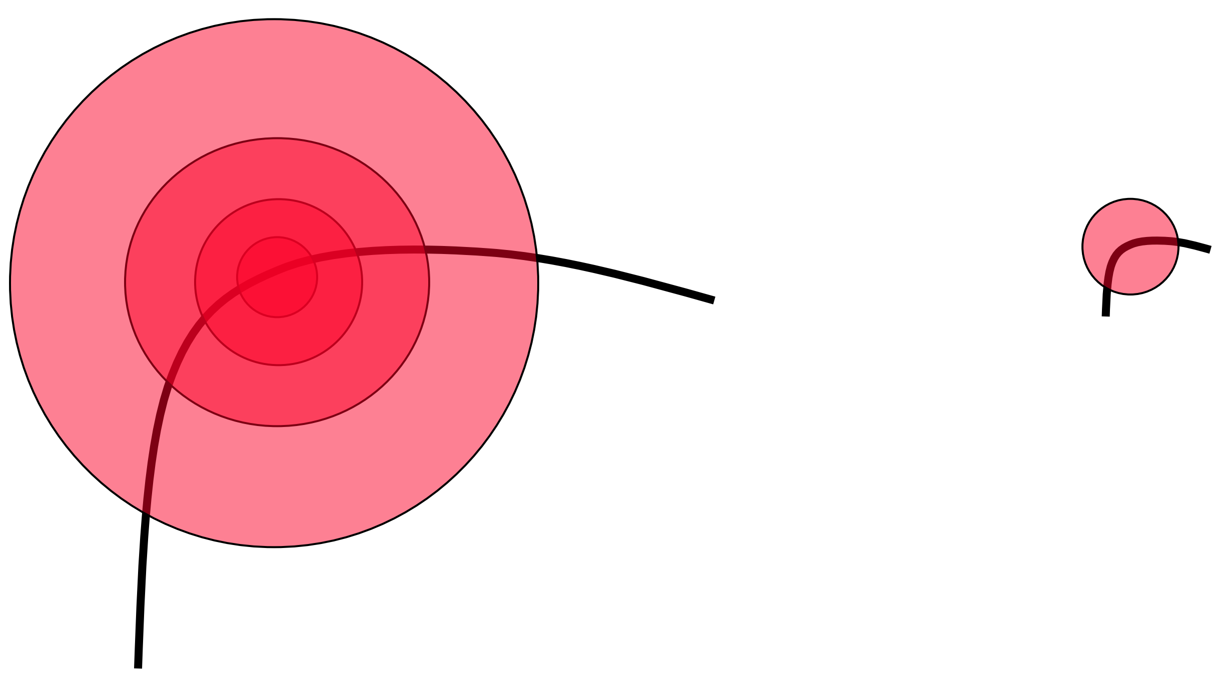


All points will be classified as edges

Corner !

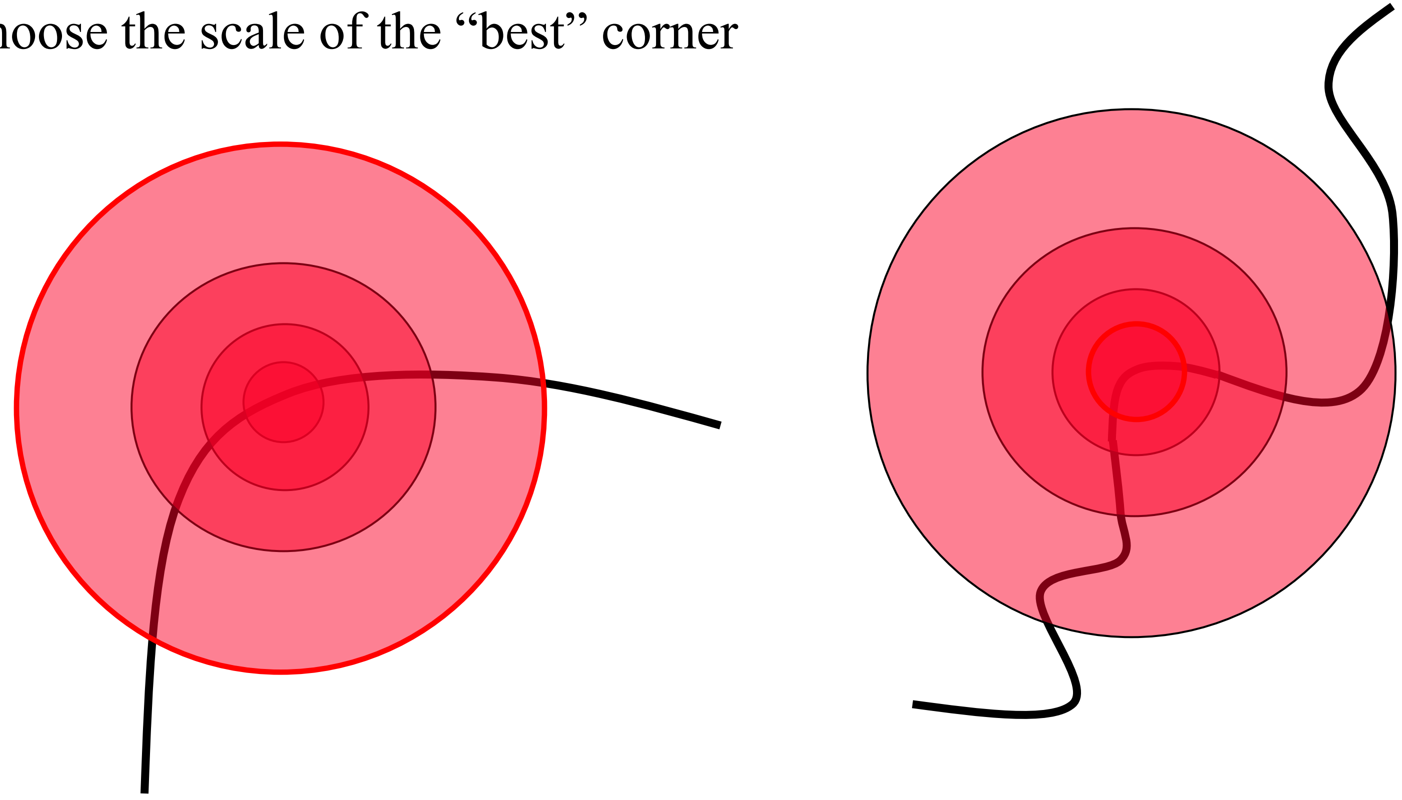
Scale Invariant Detection

- Consider regions (e.g. circles) of different sizes around a point
- Regions of corresponding sizes will look the same in both images



Scale Invariant Detection

- The problem: how do we choose corresponding circles ***independently*** in each image?
- Choose the scale of the "best" corner



Other feature detectors

- *LoG* and *DoG* operators are also used to detect "features"

- they find reliable "blob" features (at appropriate scale)

- these operators also respond to edges. To improve "selectivity", post-processing is necessary.
  - e.g. eigen-values of the Harris matrix cold be used as in the corner operator. If the ratio of the eigen-values is too high, then the local image is regarded as too edge-like and the feature is rejected.

Other features

- MOPS, Hog, SIFT, …

Features are characterized by	**location** and	**descriptor**
color	any pixel	RGB vector
edge	Laplacian zero crossing	image gradient
corner	local max of R	magnitude of R
MOPS	corners	normalized intensity patch
HOG SIFT	LOG extrema points or other interest points	gradient orientation histograms

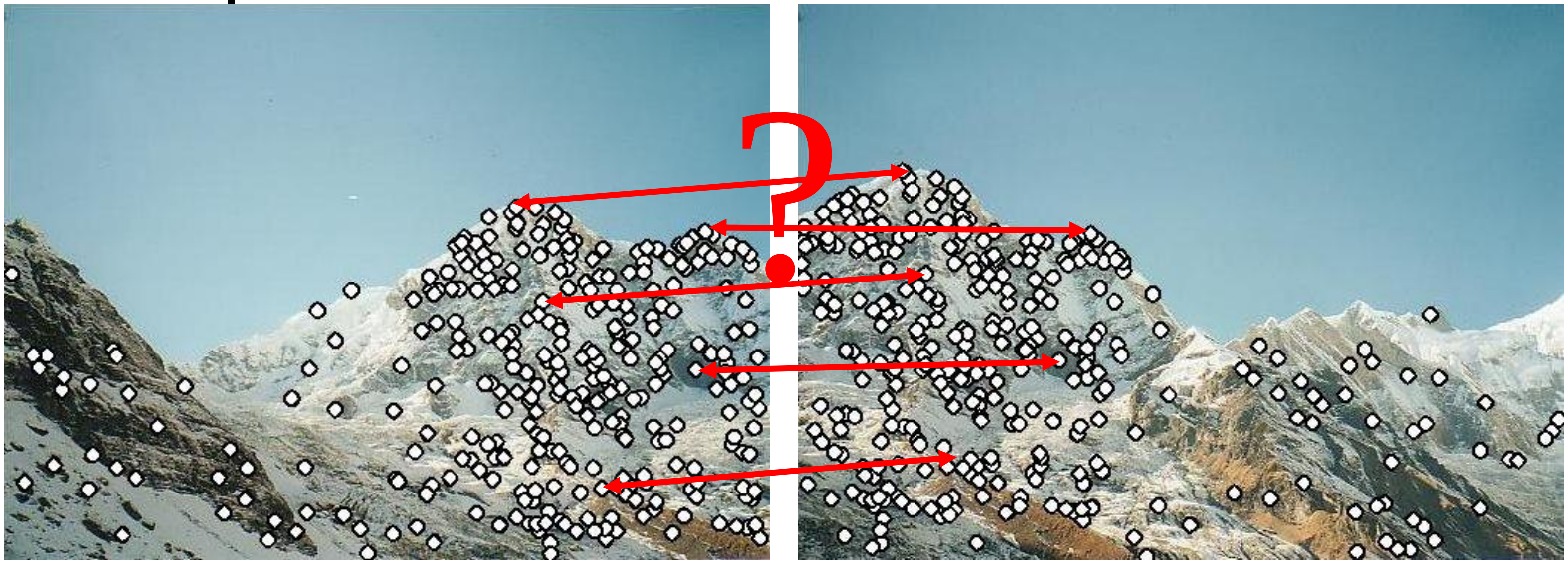
more below

highly discriminative

(see Szeliski, Sec. 4.1.2)

Feature descriptors

- We know how to detect points
- Next question: **How to match them?**

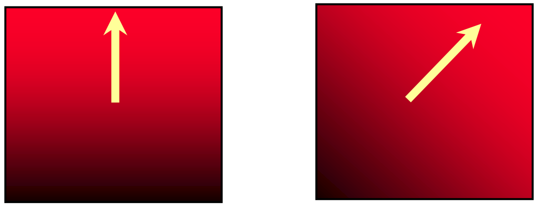


Point descriptor should be:
1. Invariant
2. Distinctive

Descriptors Invariant to Rotation

- Find local orientation

Dominant direction of gradient



- Extract image patches relative to this orientation

Multi-Scale Oriented Patches (MOPS)

- Interest points
  - Multi-scale Harris corners
  - Orientation from blurred gradient
  - Geometrically invariant to rotation
- Descriptor vector
  - Bias/gain normalized sampling of local patch (8x8)
  - Photometrically invariant to affine changes in intensity

[Brown, Szeliski, Winder, CVPR’2005]

Descriptor Vector

- Orientation = blurred gradient
- Rotation Invariant Frame
  - Scale-space position (x, y, s) + orientation (θ)

Detections at multiple scales

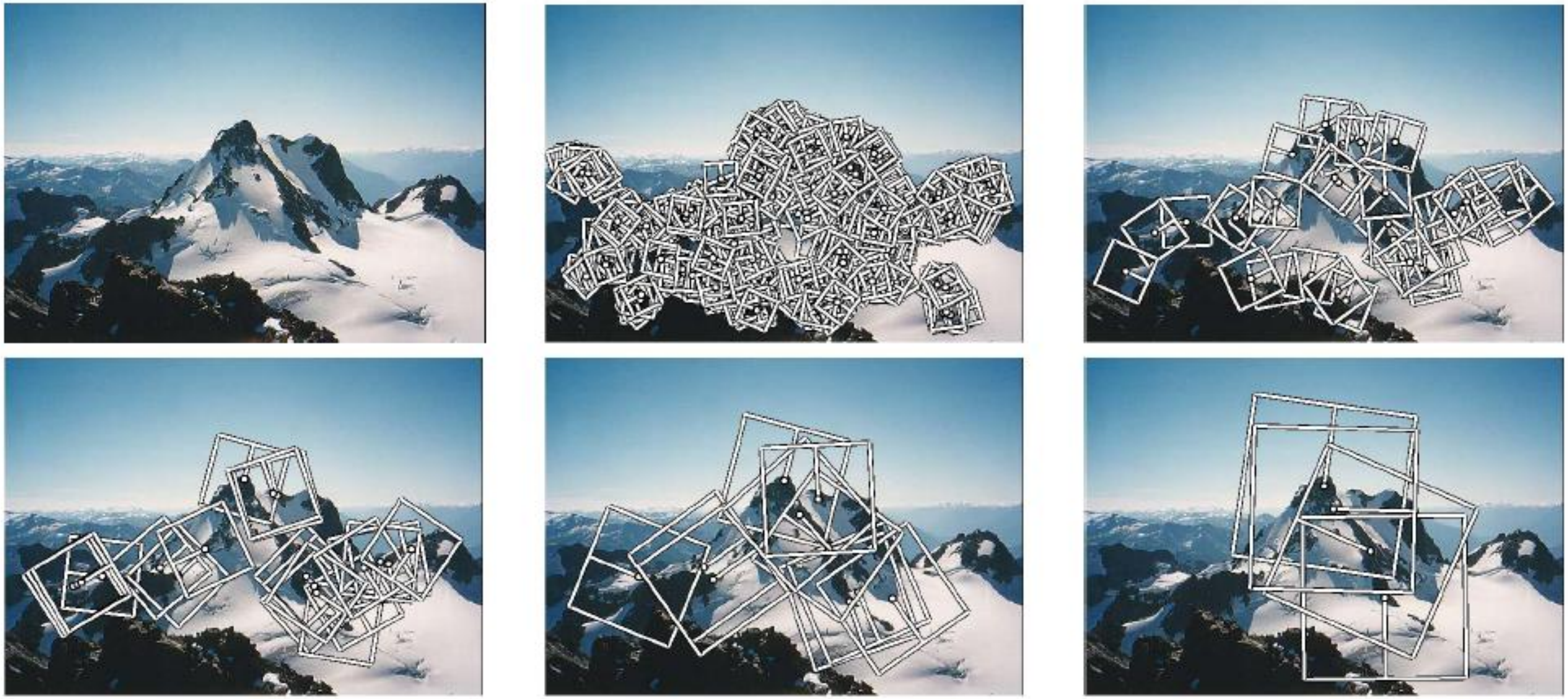


Figure 1. Multi-scale Oriented Patches (MOPS) extracted at five pyramid levels from one of the Matier images. The boxes show the feature orientation and the region from which the descriptor vector is sampled.

MOPS descriptor vector

- 8x8 oriented patch
  - Sampled at 5 x scale
- Bias/gain normalisation: $I' = (I - \mu)/\sigma$