

# SPATIAL FILTERING



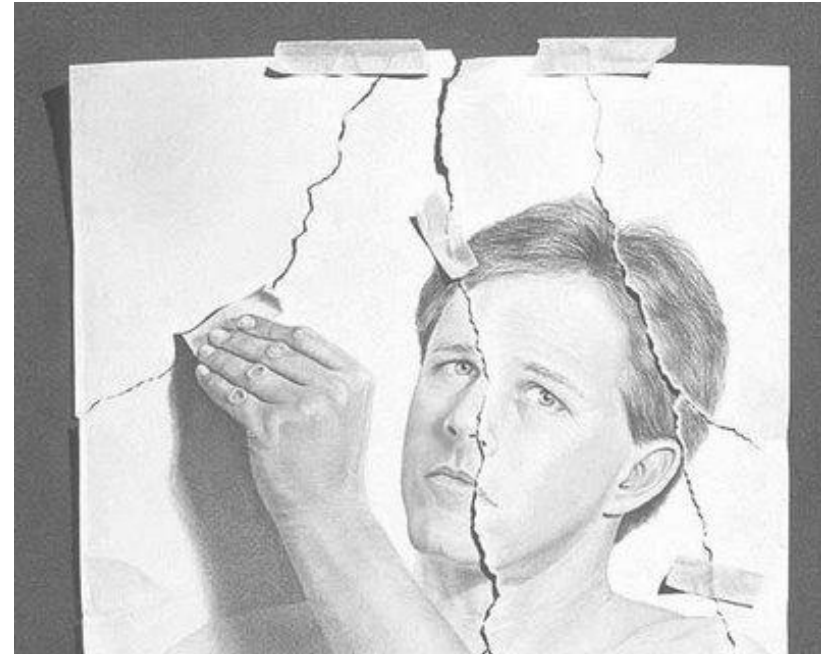
Amartya Kundu Durjoy

Lecturer

CSE Department (UGV)

# General Image Operations

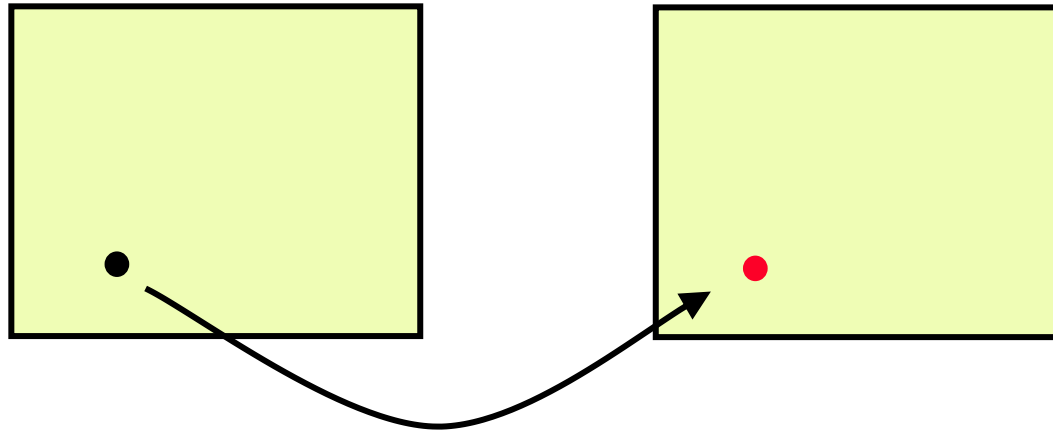
- Three type of image operations
  1. Point operations
  2. Geometric operations
  3. Spatial operations
  4. Global Operations (Freq. domain)
  5. Multi-Resolution Operations



# Point Operations



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



# Point Operations

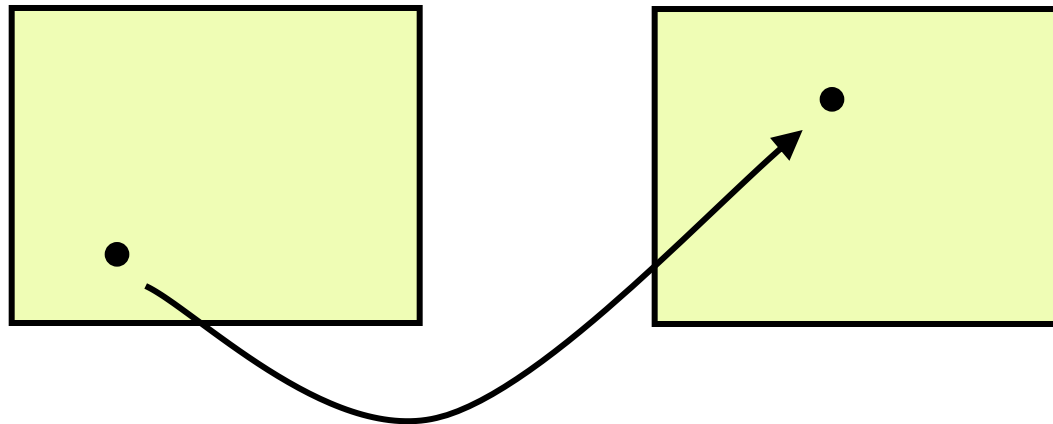
- Operation depends on Pixel's value.
- Context free (memory-less).
- Operation can be performed on the Histogram.
- Example:

$$g(x, y) = \alpha \cdot f(x, y) + \beta$$

# Geometric Operations



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

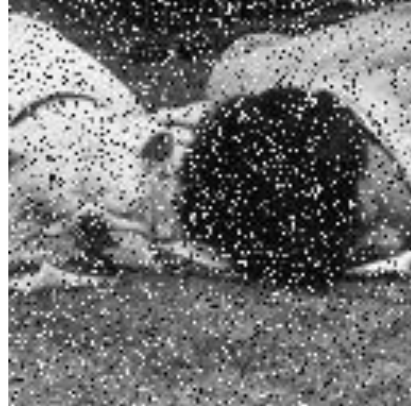


# Geometric Operations

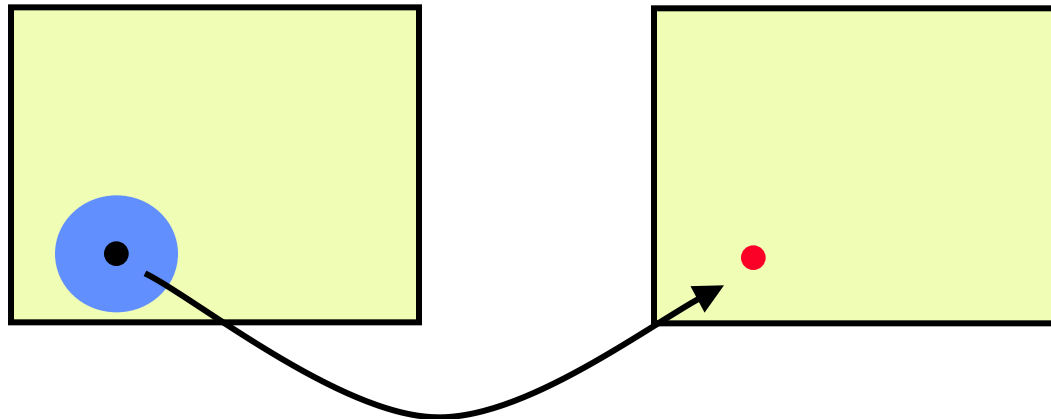
- Operation depend on pixel's coordinates.
- Context free.
- Independent of pixels value.
- Example:

$$g(x, y) = f(x + a, y + b)$$

# Spatial Operations



$$g(x, y) = T\left(\{f(i, j) \mid (i, j) \in N(x, y)\}\right)$$



# Spatial Operations

---

- Operation depends on Pixel's value and coordinates.
- Context dependant.
- Spatial v.s. Frequency domain.
- Example:

$$g(x, y) = \sum_{i, j \in N(x, y)} f(i, j) / n$$

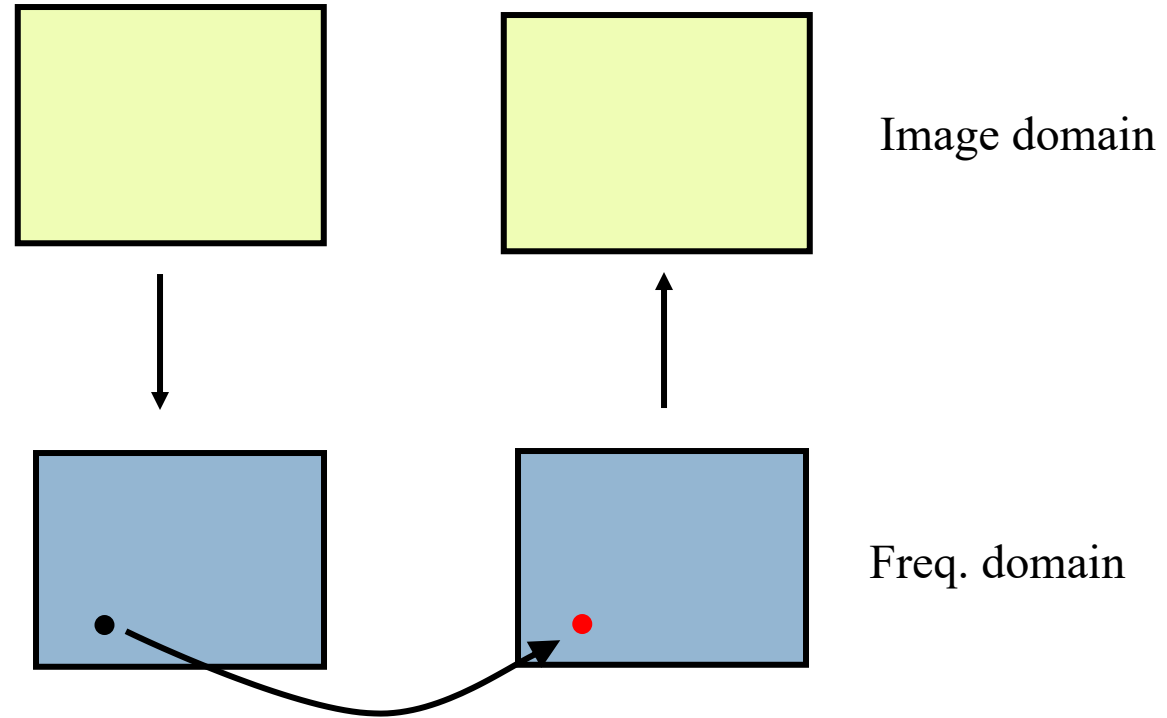


# Global Operations

---



# Global Operations



# Multi-Resolution

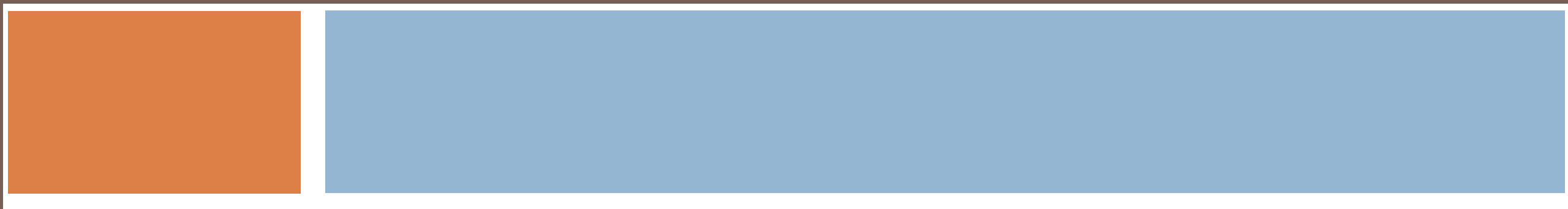
Low resolution



High resolution



# Convolution and Correlation



# 1D Continuous Convolution

□ Convolution is defined as follows:

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(a)g(x-a)da$$

1. Flip  $g(a) \rightarrow g(-a)$
2. Shift  $g(x-a)$   $-\infty < x < \infty$
3. Compute area overlap  
 $f(a)g(x-a)$

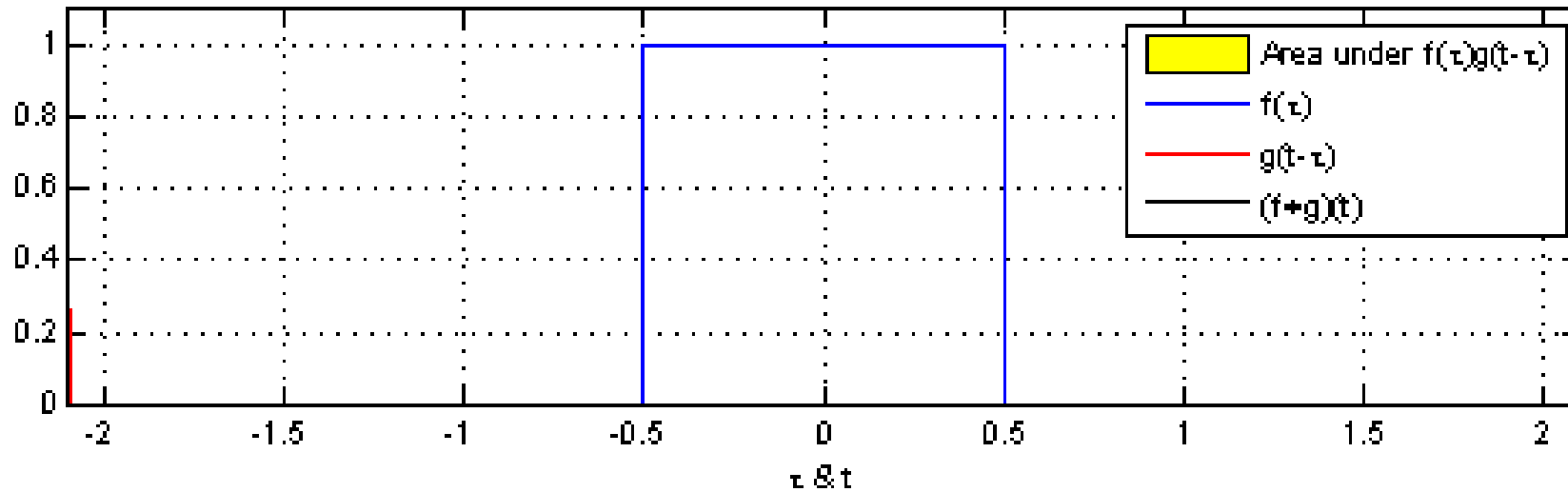
□ Convolution is commutative:

$$f(x) * g(x) = g(x) * f(x)$$

Think of  $f(x)$  as the image and  $g(x)$  as the mask although you can reverse their roles!

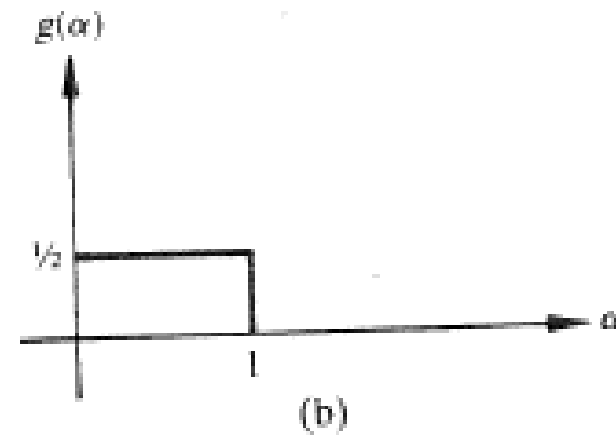
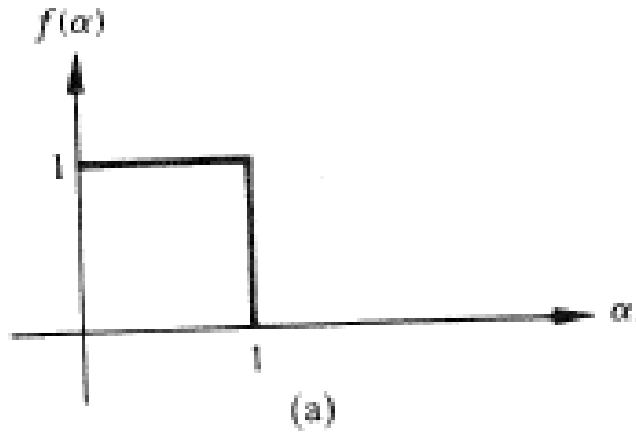
# Example 1

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(a)g(x-a)da$$



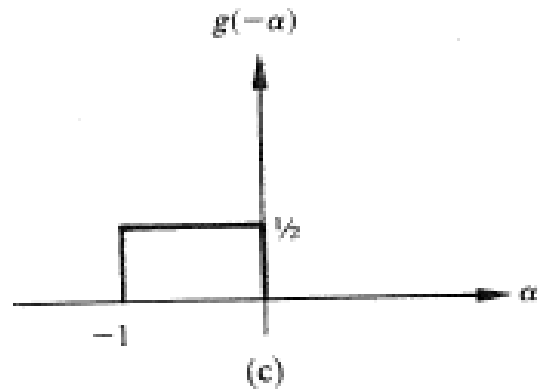
## Example 2

- Compute the convolution of the following two functions:

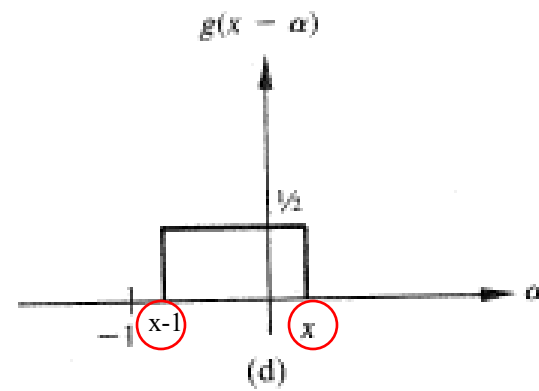


## Example 2 (cont'd)

Step1: find  $g(-a)$



Step2: find  $g(x - a)$

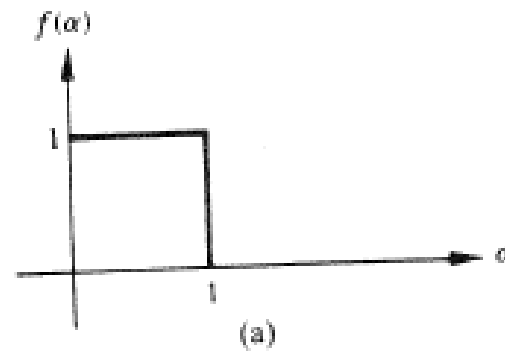
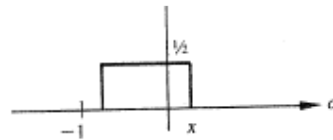




## Example 2 (cont'd)

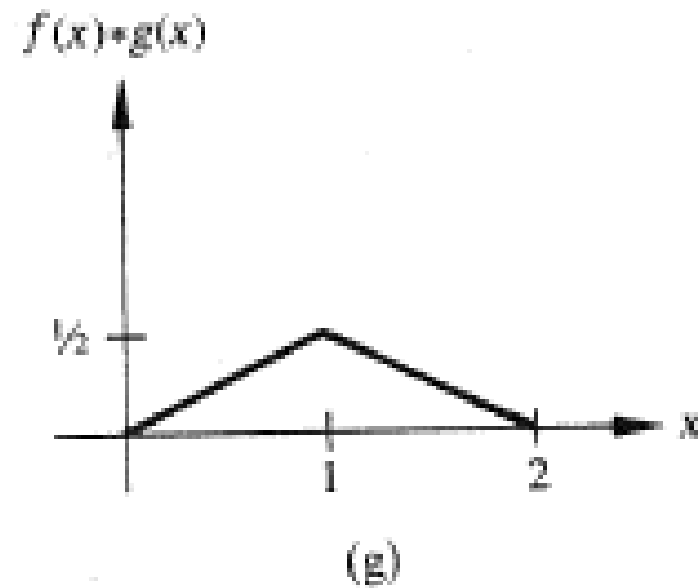
Step 3: Compute the integral for  $-\infty < x < \infty$

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(a)g(x-a)da$$



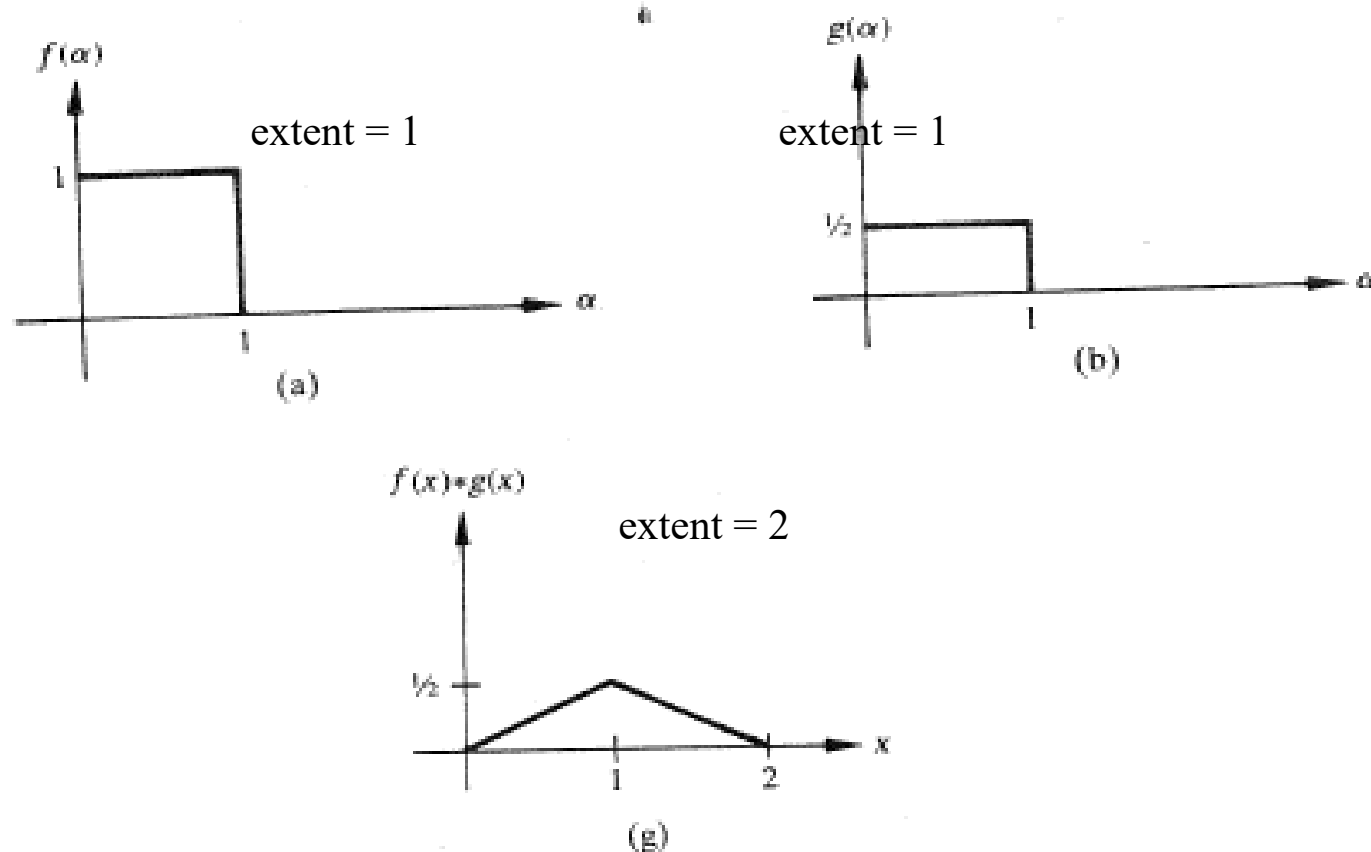
## Example 2 (cont'd)

$$f(x) * g(x) = \begin{cases} x/2 & 0 \leq x \leq 1 \\ 1 - x/2 & 1 \leq x \leq 2 \\ 0 & \text{elsewhere} \end{cases}$$



# Important Observations

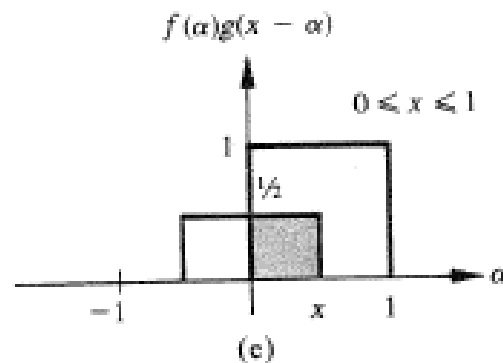
- The extent of  $f(x) * g(x)$  is equal to the extent of  $f(x)$  plus the extent of  $g(x)$



# Important Observations (cont'd)

- For every  $x$ , the limits of the integral are determined as follows:
  - ▣ Lower limit: MAX (left limit of  $f(x)$ , left limit of  $g(x-a)$ )
  - ▣ Upper limit: MIN (right limit of  $f(x)$ , right limit of  $g(x-a)$ )

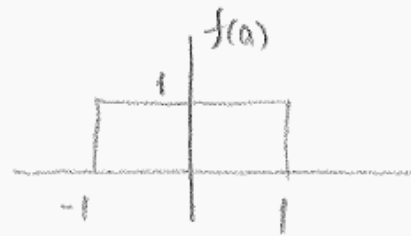
Case 2:  $0 \leq x \leq 1$



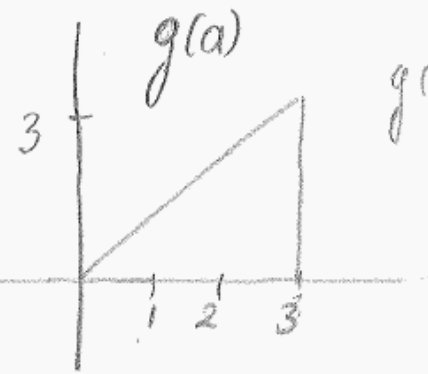
$$\int_{-\infty}^{\infty} f(a)g(x-a)da = \int_0^x 1 \frac{1}{2} da = \frac{x}{2}$$

# Example 3

one more example: convolution



\*



$$g(a) = \begin{cases} a & 0 \leq a \leq 3 \\ 0 & \text{elsewhere} \end{cases}$$

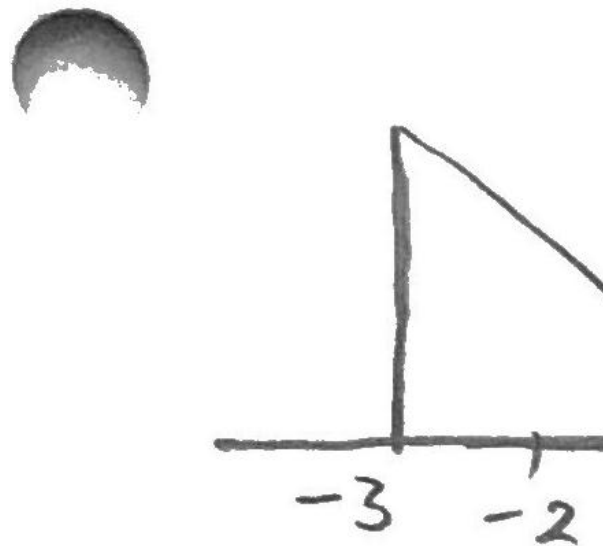
$$f(x) * g(x) = \begin{cases} \frac{1}{2}(x+1)^2 & -1 \leq x \leq 1 \\ 2x & 1 \leq x \leq 2 \\ 4+x-\frac{1}{2}x^2 & 2 \leq x \leq 4 \\ 0 & \text{otherwise} \end{cases}$$

width = 2



## Example 3 (cont'd)

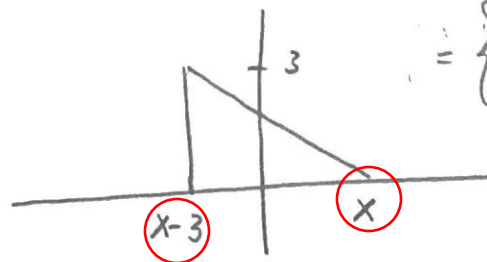
Step 1: find  $g(-a)$



+3

(reflection about the vertical axis)

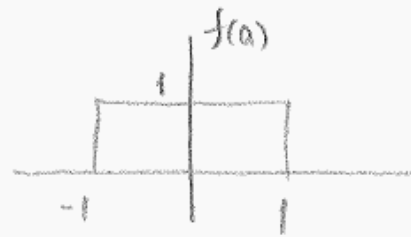
Step 2: find  $g(x-a) =$



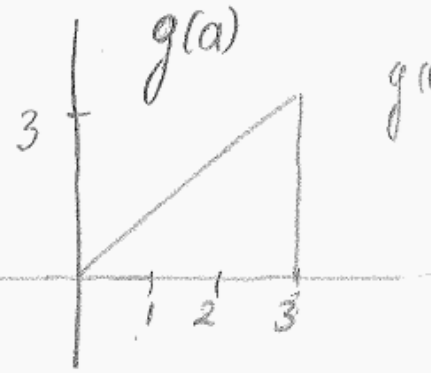
$$= \begin{cases} x-a & 0 \leq x-a \leq 3 \\ 0 & \text{elsewhere} \end{cases}$$

## Example 3 (cont'd)

one more example: convolution

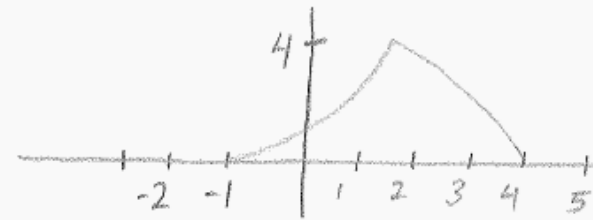


\*



$$g(a) = \begin{cases} a & 0 \leq a \leq 3 \\ 0 & \text{elsewhere} \end{cases}$$

$$f(x) * g(x) = \begin{cases} \frac{1}{2}(x+1)^2 & -1 \leq x \leq 1 \\ 2x & 1 \leq x \leq 2 \\ 4+x - \frac{1}{2}x^2 & 2 \leq x \leq 4 \\ 0 & \text{otherwise} \end{cases}$$



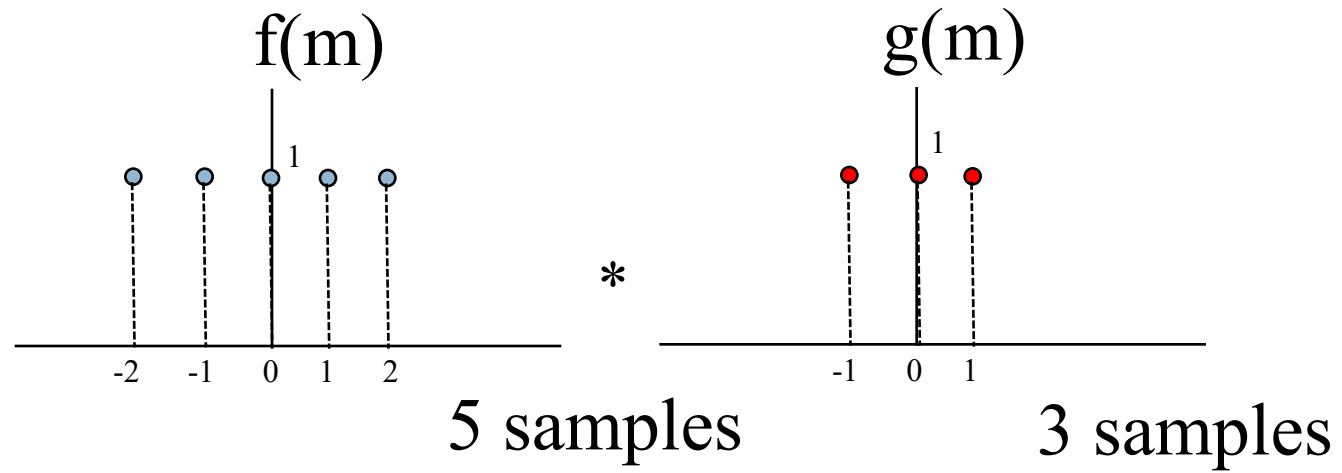
# Discrete Convolution

- Replace integral with summation
- Integration variable becomes an index.
- Displacements take place in **discrete increments**

$$f(x) * g(x) = \sum_{m=-\infty}^{\infty} f(m)g(x - m), -\infty < x < \infty$$

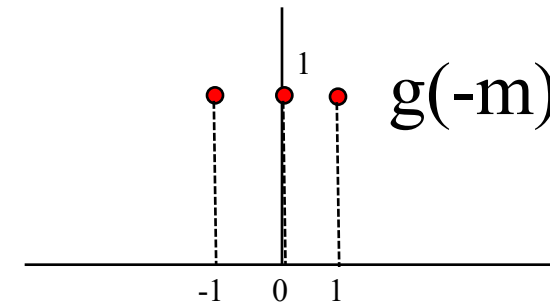
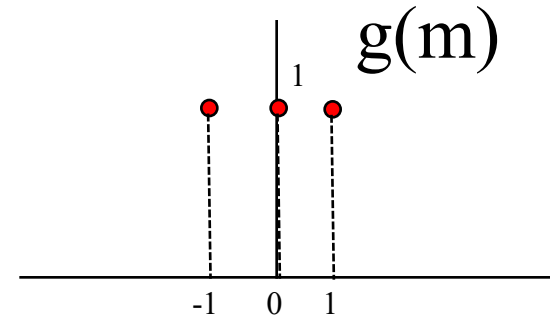
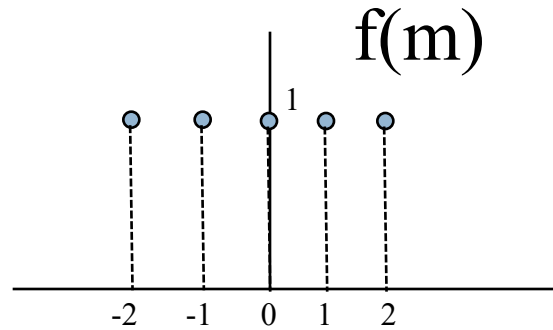


# Example

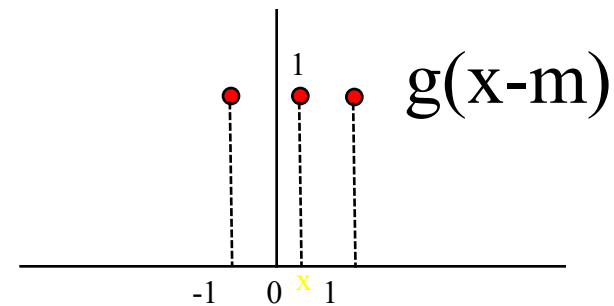


$$f(x) * g(x) = \sum_{m=-\infty}^{\infty} f(m)g(x-m), \quad -\infty < x < \infty$$

# Example (cont'd)

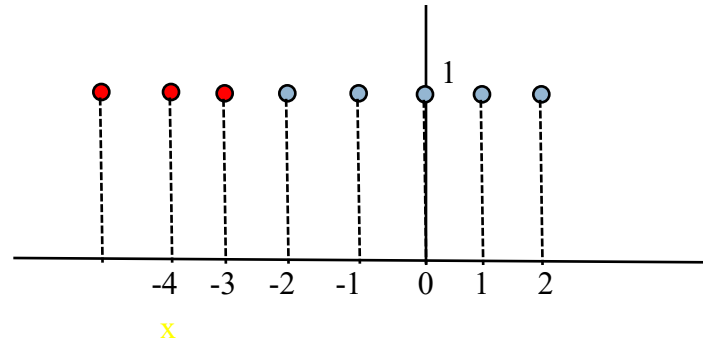


Compute the convolution  
assuming discrete values  
for  $x$ ,  $-\infty < x < \infty$



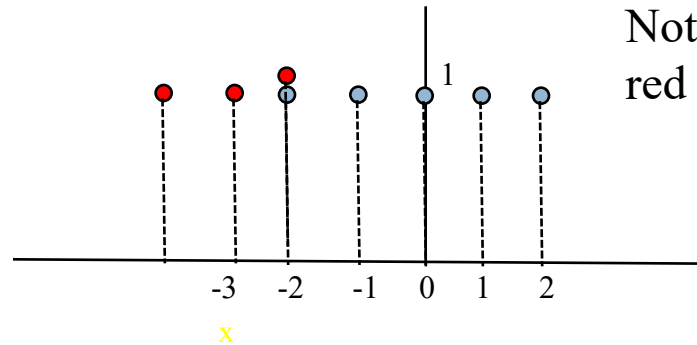
# Example (cont'd)

$x = -4$  or  $x < -4$   
no overlap



$$f * g = 0$$

$x = -3$

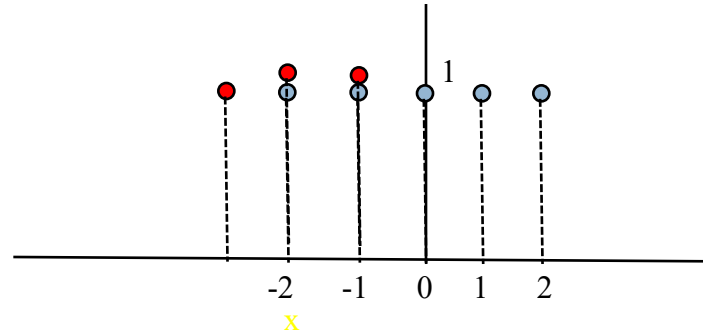


Note that I show some  
red samples “taller” for clarity!

$$f * g = 1$$

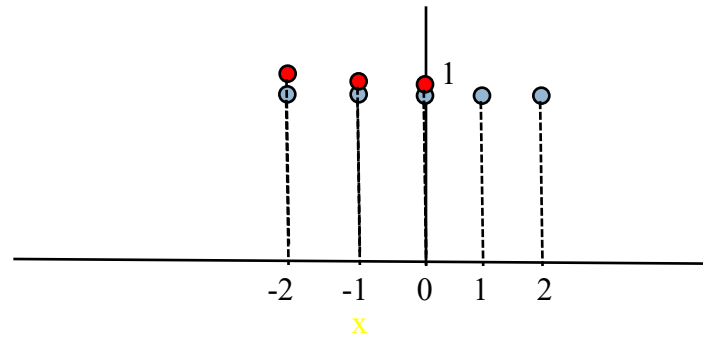
# Example (cont'd)

$$x = -2$$



$$f * g = 2$$

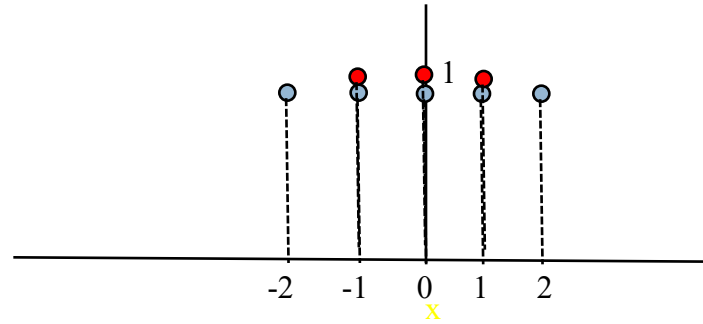
$$x = -1$$



$$f * g = 3$$

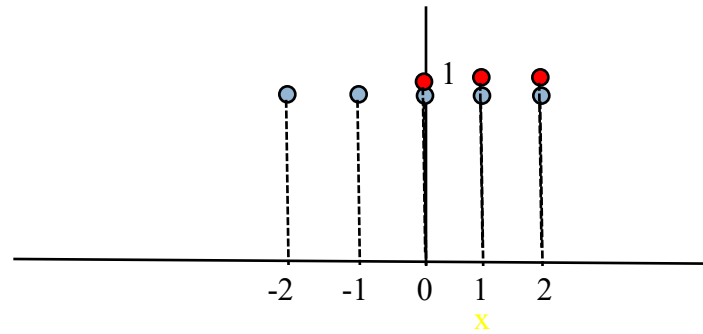
# Example (cont'd)

$$x = 0$$



$$f * g = 3$$

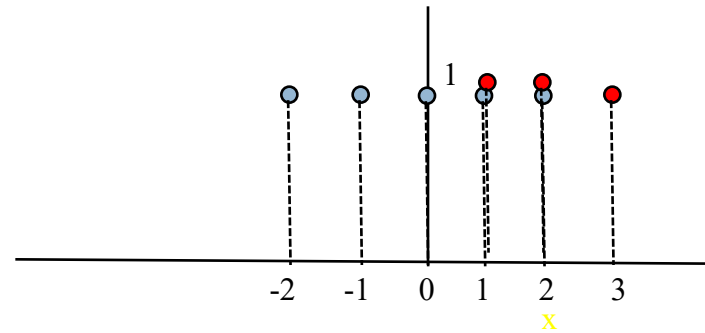
$$x = 1$$



$$f * g = 3$$

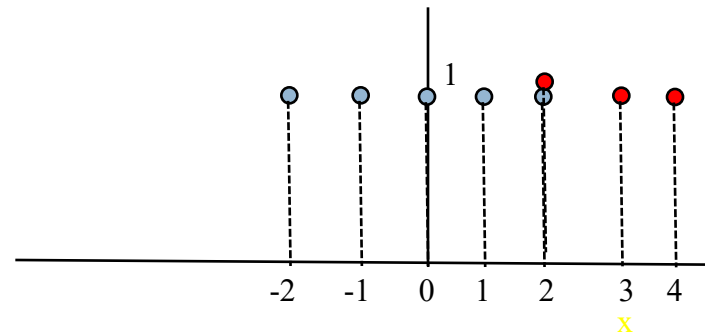
# Example (cont'd)

$$x = 2$$



$$f * g = 2$$

$$x = 3$$



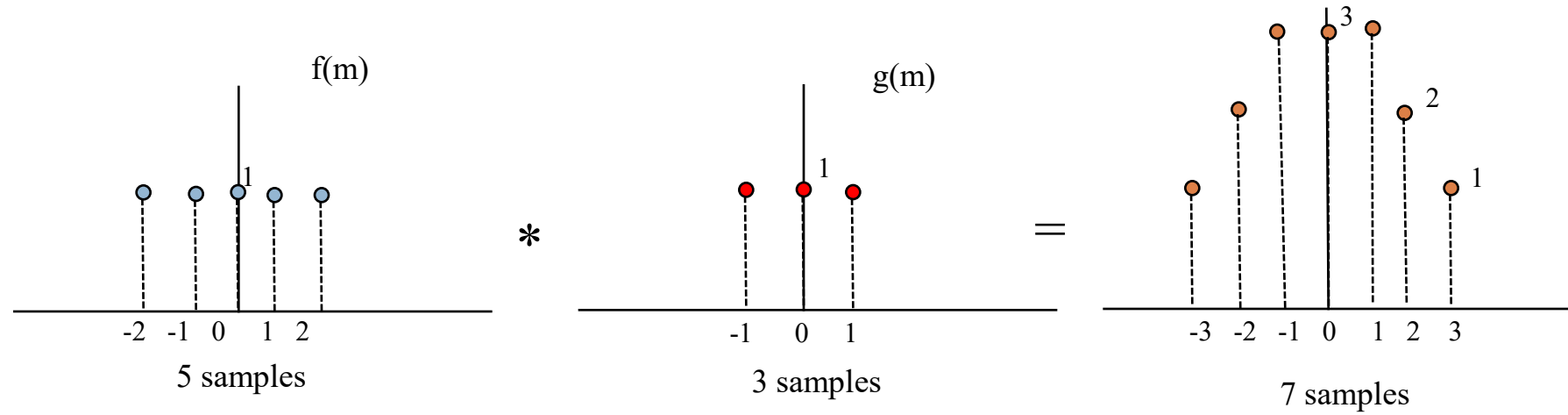
$$f * g = 1$$

$$x \geq 4$$

$$f * g = 0$$

no overlap

# Example



$$\text{length of } f * g = \text{length of } f + \text{length of } g - 1$$

# Convolution in Discrete Case

□ Input sequences:

$$\{\bar{f}(0), f(1), \dots, f(A-1)\}, \{g(0), g(1), \dots, g(B-1)\}$$

□ Length of  $f*g$  sequence is:  $M=A+B-1$

□ **Extended** input sequences: make them length  $M$  by padding with zeroes:

$$f_e(x) = \begin{cases} f(x) & 0 \leq x \leq A-1 \\ 0 & A \leq x \leq M-1 \end{cases} \quad g_e(x) = \begin{cases} g(x) & 0 \leq x \leq B-1 \\ 0 & B \leq x \leq M-1 \end{cases}$$



# Discrete 2D convolution

- Suppose  $f(x,y)$  is  $A \times B$  and  $g(x,y)$  is  $C \times D$
- The size of  $f(x,y) * g(x,y)$  would be  $N \times M$  where
$$N=A+C-1 \text{ and } M=B+D-1$$
- Form **extended** images (i.e., **pad with zeroes**):

$$f_e(x,y) = \begin{cases} f(x,y) & 0 \leq x \leq A-1 \text{ and } 0 \leq y \leq B-1 \\ 0 & A \leq x \leq M-1 \text{ and } B \leq y \leq N-1 \end{cases}$$

$$g_e(x,y) = \begin{cases} g(x,y) & 0 \leq x \leq C-1 \text{ and } 0 \leq y \leq D-1 \\ 0 & C \leq x \leq M-1 \text{ and } D \leq y \leq N-1 \end{cases}$$

# Discrete 2D convolution

- The convolution holds true for the extended images only!

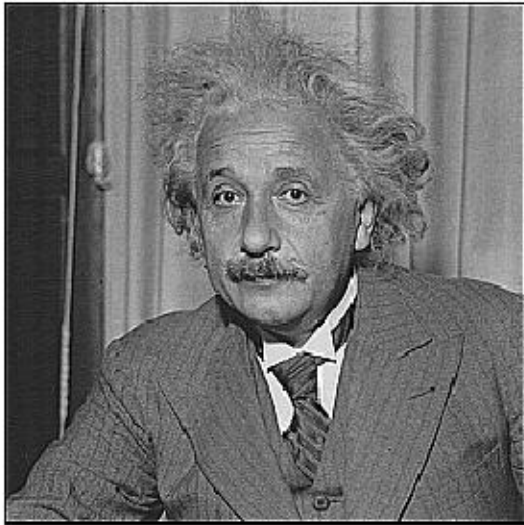
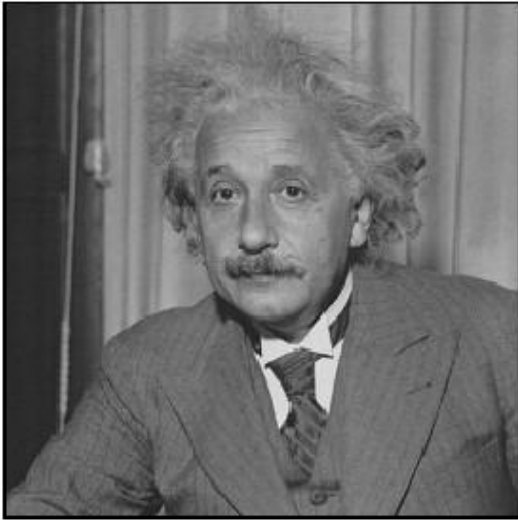
$$f_e(x, y) * g_e(x, y) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_e(m, n) g_e(x - m, y - n)$$
$$(x = 0, 1, \dots, M - 1, y = 0, 1, \dots, N - 1)$$

# Image filtering

---

- Image filtering: compute function of local neighborhood at each position
- Really important!
  - ▣ Enhance images
    - Denoise, resize, increase contrast, etc.
  - ▣ Extract information from images
    - Texture, edges, distinctive points, etc.
  - ▣ Detect patterns
    - Template matching

# Today: Image Filters



*Smooth/Sharpen Images...*

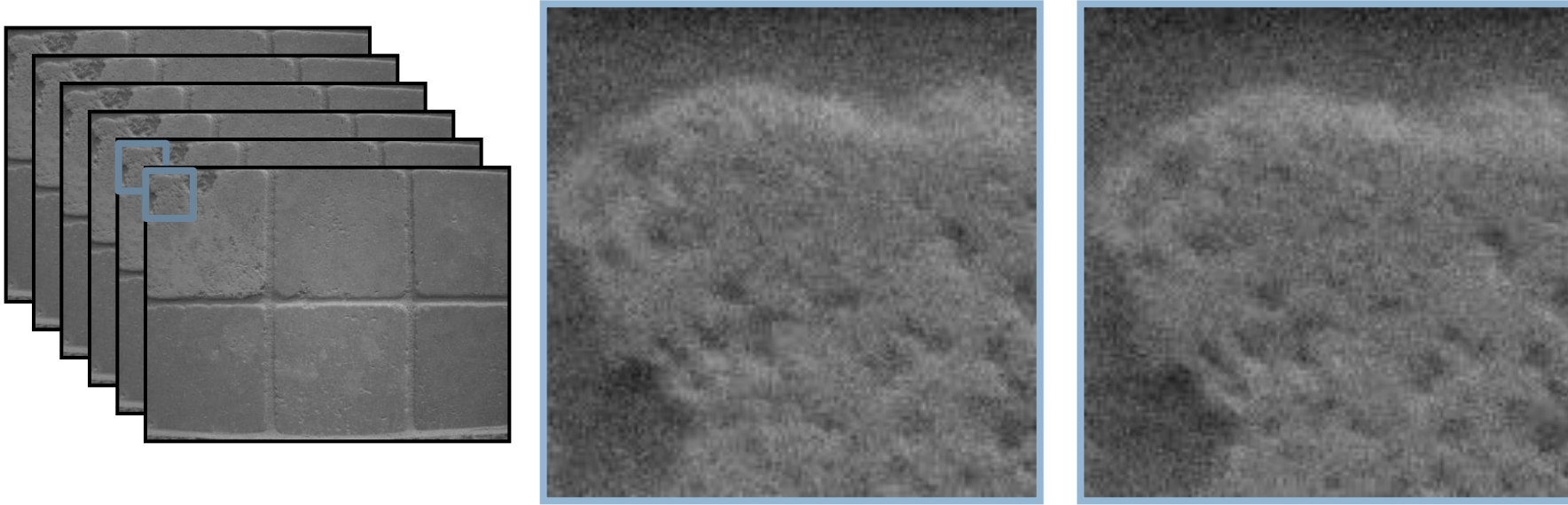


*Find edges...*



*Find waldo...*

# Motivation: noise reduction

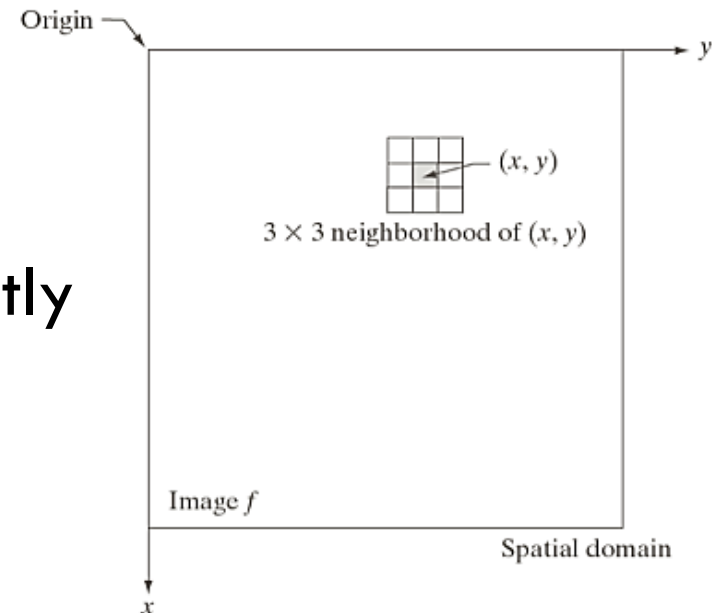


- How could we reduce the noise, i.e., give an estimate of the true intensities?
- **What if there's only one image?**

# Filtering

- **Filter** term in “Digital image processing” is referred to the subimage
  - ▣ Others terms - **mask, kernel, template, or window**
- The value in a filter subimage are referred as coefficients, rather than pixels.
- The word “filtering” has been borrowed from the frequency domain.

- Spatial filtering term is the filtering operations that are performed directly on the pixels of an image



# Mechanics of spatial filtering

---

□ Need to define:

(1) a neighborhood (or mask) - Typically, rectangular and its size is much smaller than that of  $f(x,y)$

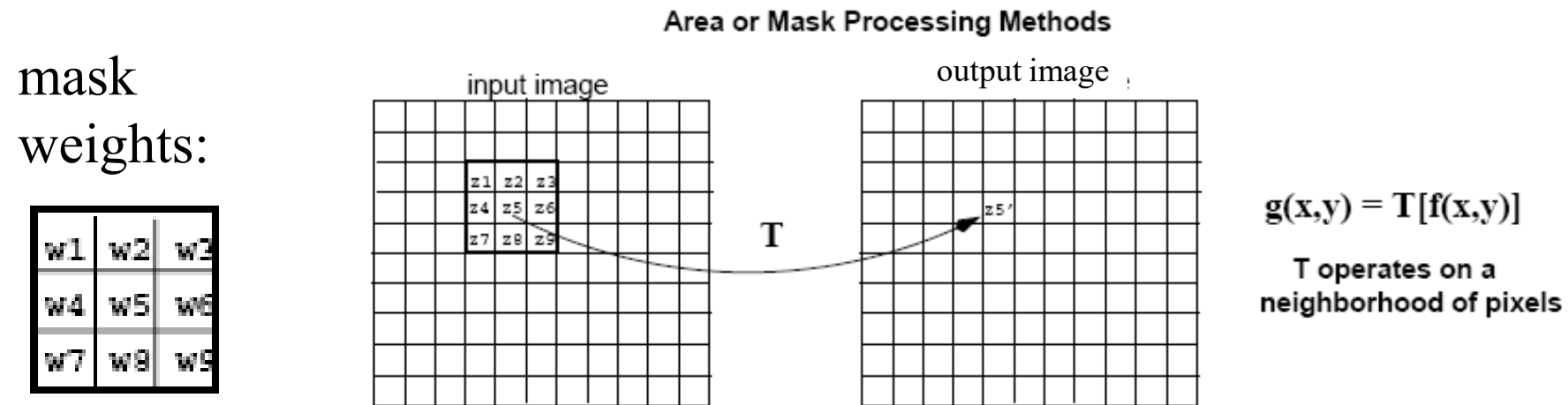
■ e.g. - 3x3 or 5x5

(2) an operation

■ e.g. - weighted sum of input pixels

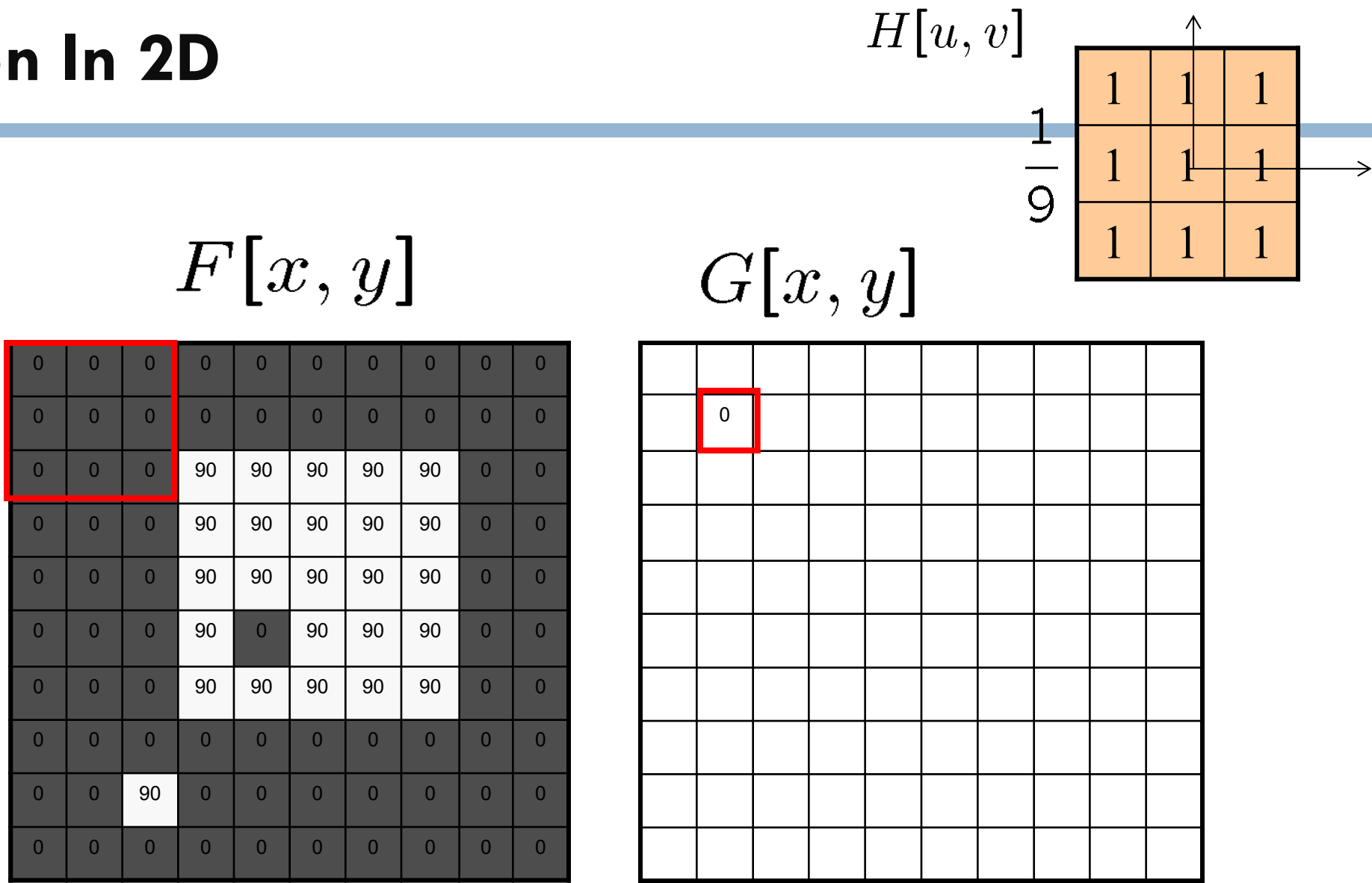
# Mechanics of spatial filtering

- The process consists simply of moving the **center** of the filter mask from point to point in an image.
- At each point  $(x,y)$  the response of the filter at that point is calculated using a predefined relationship





# Convolution In 2D



# Convolution In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0	10							

# Convolution In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0	10	20						

# Convolution In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0	10	20	30					

# Convolution In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0	10	20	30	30				

# Convolution In 2D

	0	10	20	30	30	30	20	10											
	0	20	$F[x, y]$			60	60	$G[x, y]$			40	20							
	0	30	60	90	90	90	60	30											
	0	0	0	0	0	0	0	0	0	0		80	90	60	30				
	0	0	0	0	0	0	0	0	0	0		80	90	60	30				
	0	0	0	90	90	90	90	90	0	0		50	60	40	20				
	10	0	0	90	90	90	90	90	0	0		30	30	20	10				
	10	0	0	90	90	90	90	90	0	0		0	0	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								

# Correlation filtering

Say the averaging window size is  $2k+1 \times 2k+1$ :

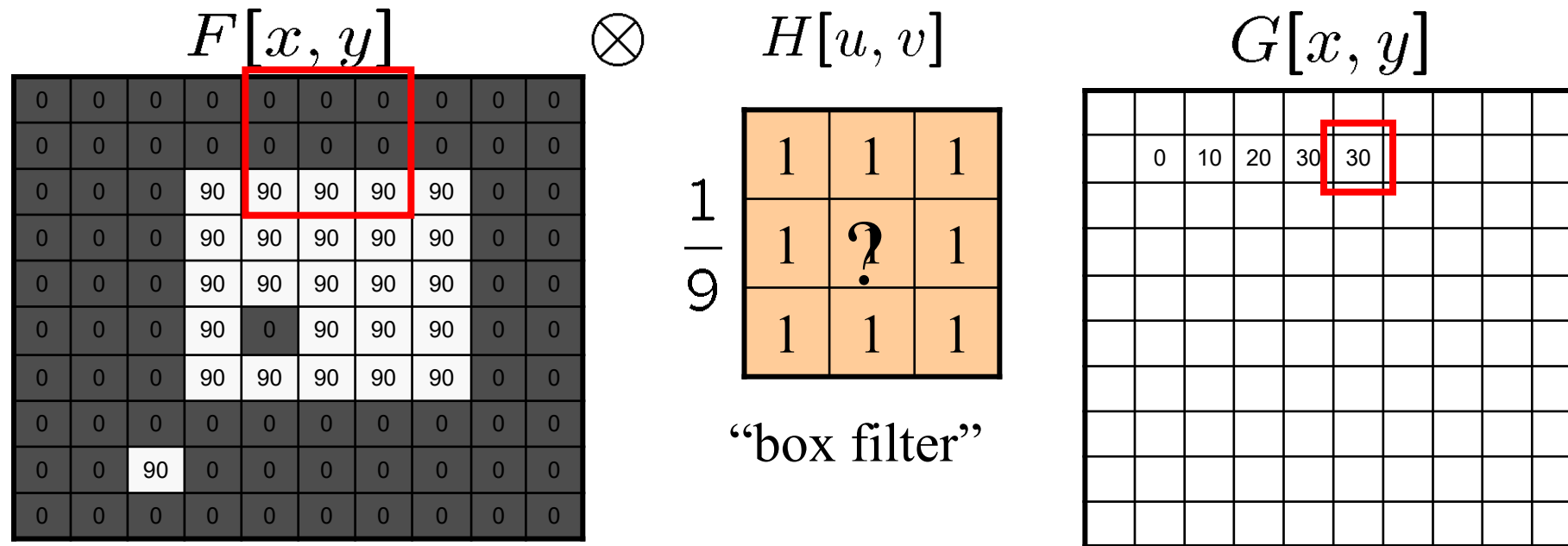
$$G[i, j] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{u=-k}^k \sum_{v=-k}^k F[i+u, j+v]}_{\text{Loop over all pixels in neighborhood around image pixel } F[i,j]}$$

Now generalize to allow different weights depending on neighboring pixel's relative position:

$$G[i, j] = \sum_{u=-k}^k \sum_{v=-k}^k \underbrace{H[u, v]}_{\text{Non-uniform weights}} F[i+u, j+v]$$

# Averaging filter

- What values belong in the kernel  $H$  for the moving average example?

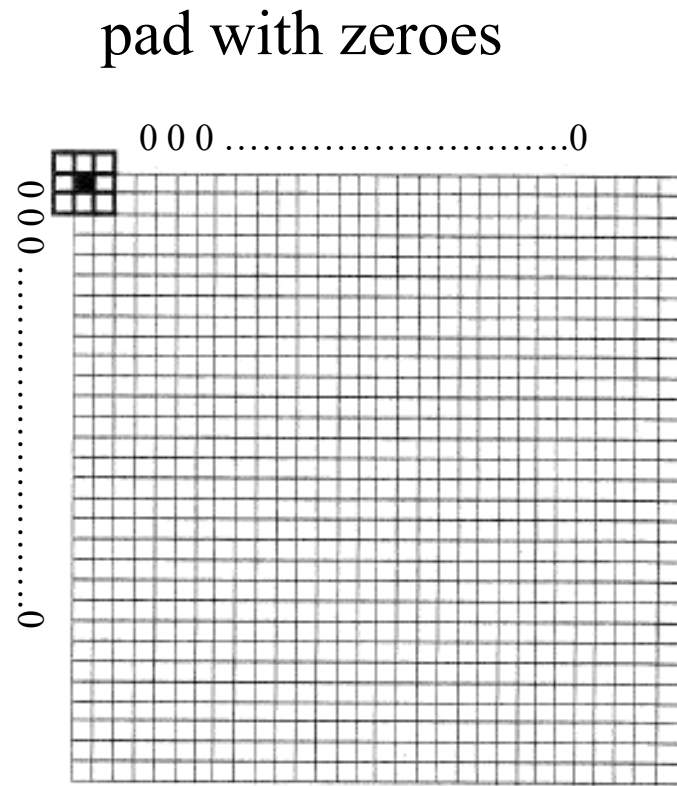


$$G = H \otimes F$$

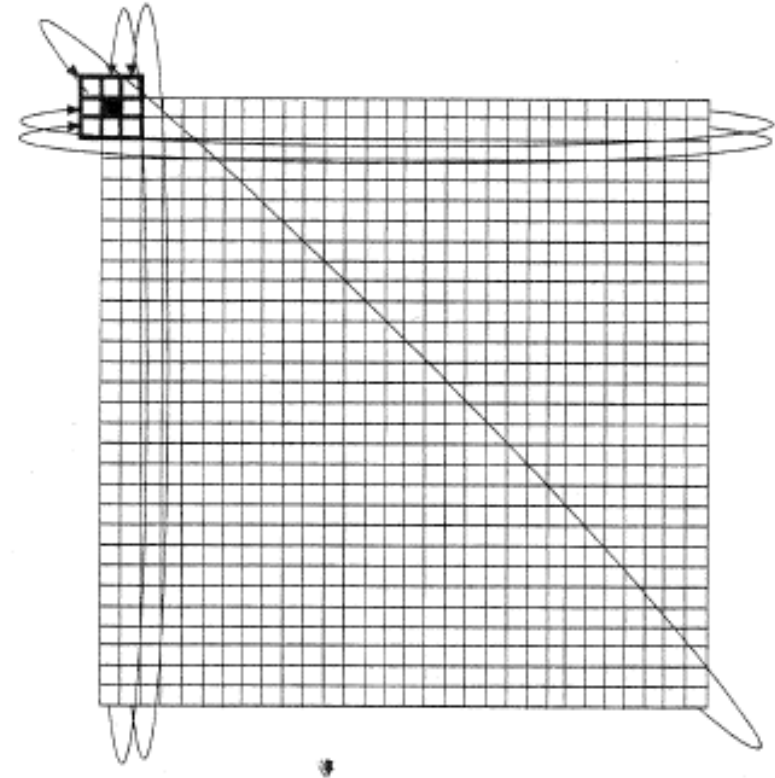


# Mechanics of spatial filtering

## □ Handling Pixels Close to Boundaries

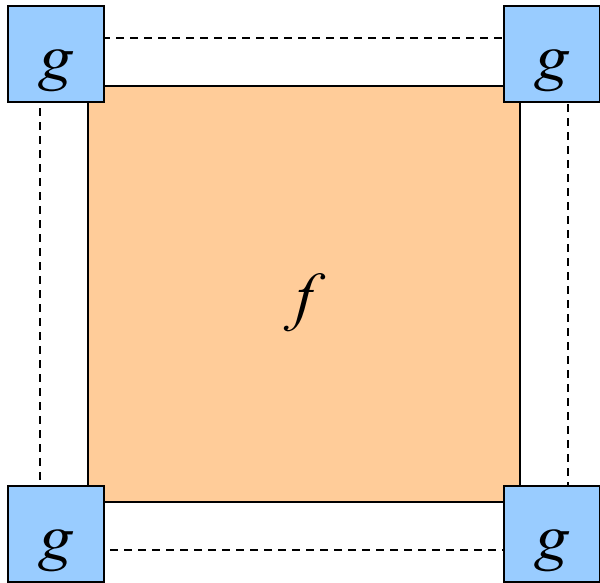


or

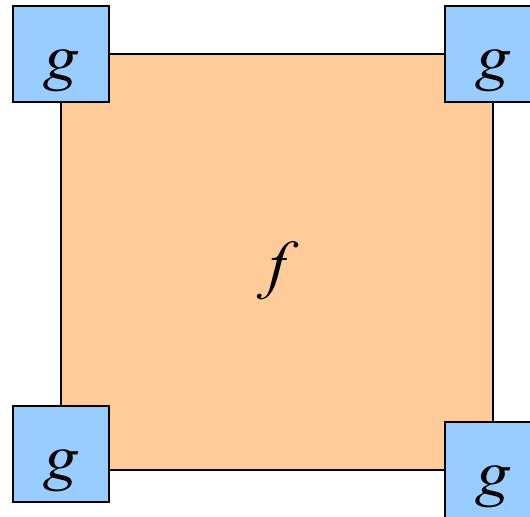


# Boundary issues

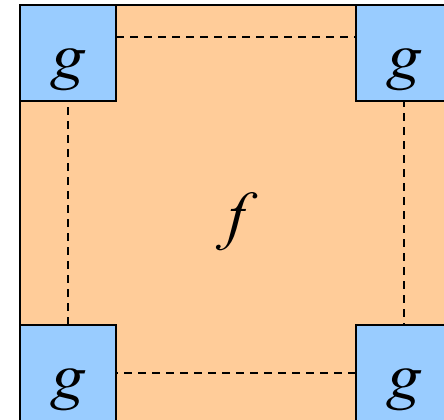
- What is the size of the output?



Full



Same



valid

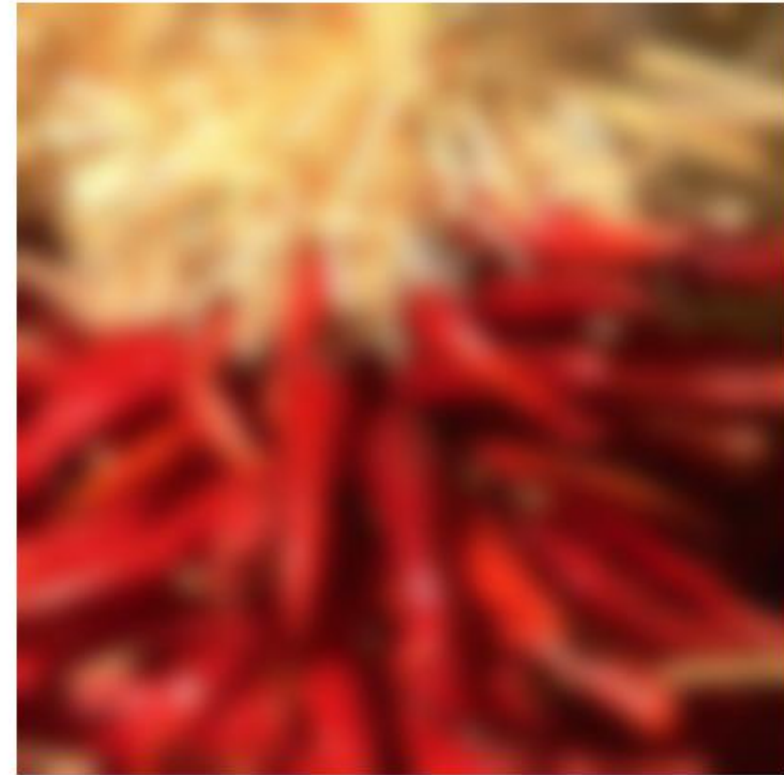
# Boundary issues

- What about near the edge?
  - ▣ the filter window falls off the edge of the image
  - ▣ need to extrapolate
  - ▣ methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge

$$F[x, y]$$

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

		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

 $G[x, y]$ 

**Reflect** : S. Marschner

# Linear vs Non-linear filter

- If the computations performed on the pixels of the neighborhoods are linear, the operation is called *linear spatial filtering*;
  - ▣ Linear - when the output is a weighted sum of the input pixels.
- otherwise it is called *nonlinear spatial filtering*.
  - ▣ e.g.

$$z'_5 = \max(z_k, k = 1, 2, \dots, 9)$$

# Linear spatial filtering

Pixels of image

	$w(-1,-1)$ $f(x-1,y-1)$	$w(-1,0)$ $f(x-1,y)$	$w(-1,1)$ $f(x-1,y+1)$	
	$w(0,-1)$ $f(x,y-1)$	$w(0,0)$ $f(x,y)$	$w(0,1)$ $f(x,y+1)$	
	$w(1,-1)$ $f(x+1,y-1)$	$w(1,0)$ $f(x+1,y)$	$w(1,1)$ $f(x+1,y+1)$	

The result is the sum of products of the mask coefficients with the corresponding pixels directly under the mask

Mask coefficients

$w(-1,-1)$	$w(-1,0)$	$w(-1,1)$
$w(0,-1)$	$w(0,0)$	$w(0,1)$
$w(1,-1)$	$w(1,0)$	$w(1,1)$

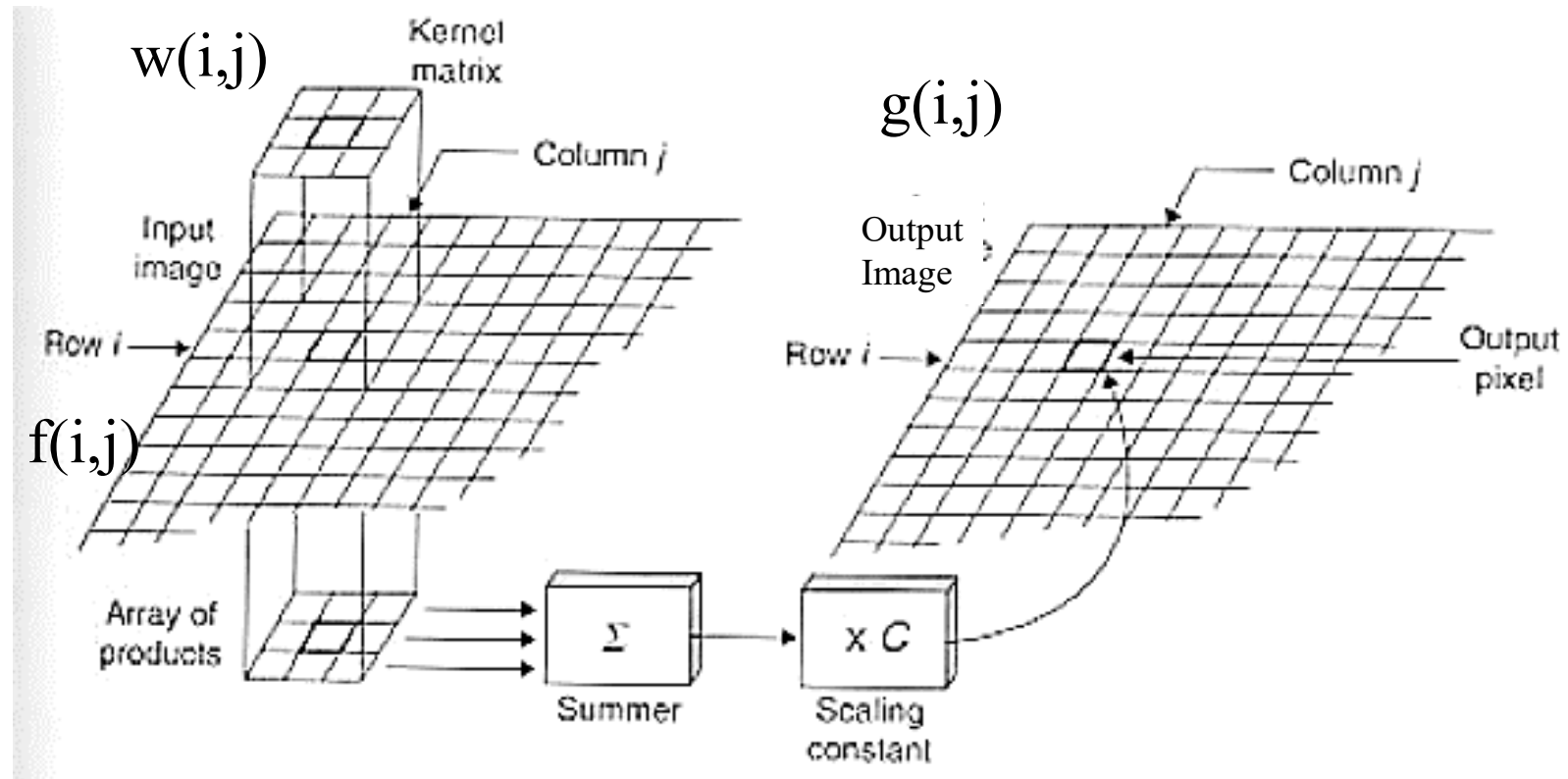
$$\begin{aligned} g(x,y) = & w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + w(-1,1)f(x-1,y+1) + \\ & w(0,-1)f(x,y-1) + w(0,0)f(x,y) + w(0,1)f(x,y+1) + \\ & w(1,-1)f(x+1,y-1) + w(1,0)f(x+1,y) + w(1,1)f(x+1,y+1) \end{aligned}$$

# Linear Spatial Filtering Methods

---

- Main linear spatial filtering methods:
  - ▣ Correlation
  - ▣ Convolution

# Correlation



$$w(x,y) \star f(x,y) = \sum_{s=-a}^a \sum_{t=-b}^b w(s,t) f(x+s, y+t)$$

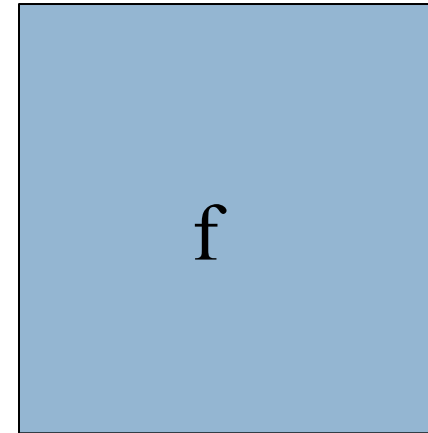
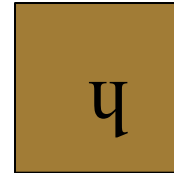
# Convolution

- Flip the filter in both dimensions (bottom to top, right to left)
  - ▣ Then apply cross-correlation

$$g(x, y) = \sum_{s=-a}^a \sum_{t=-b}^b h(s, t) f(x - a, y - b)$$

$$g(x, y) = h(x, y) \star f(x, y)$$

*Notation for convolution  
operator*



- **Note:** if  $w(i, j)$  is symmetric, that is  $w(i, j) = w(-i, -j)$ , then convolution is **equivalent** to correlation!



# Properties of convolution

- Linear & shift invariant

- Commutative:

$$f * g = g * f$$

- Associative

$$(f * g) * h = f * (g * h)$$

- Identity:

unit impulse  $e = [..., 0, 0, 1, 0, 0, ...]$ .  $f * e = f$

- Differentiation:

$$\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$$

# Convolution properties

- Commutative:  $a * b = b * a$ 
  - ▣ Conceptually no difference between filter and signal
  - ▣ But particular filtering implementations might break this equality, e.g., image edges
- Associative:  $a * (b * c) = (a * b) * c$ 
  - ▣ Often apply several filters one after another:  $((a * b_1) * b_2) * b_3$
  - ▣ This is equivalent to applying one filter:  $a * (b_1 * b_2 * b_3)$
  - ▣ Correlation is not associative (rotation effect)
- Distributes over addition:  $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out:  $ka * b = a * kb = k(a * b)$
- Identity: unit impulse  $e = [0, 0, 1, 0, 0]$ ,  $a * e = a$

# Properties of Linear SHIFT INVARIANT

- Amplitude properties:

- Additivity 
$$S[f_i[n, m] + f_j[n, m]] = S[f_i[n, m]] + S[f_j[n, m]]$$

**Meaning:** The response to the sum of two inputs is the sum of their individual responses.

- Homogeneity 
$$S[\alpha f_i[n, m]] = \alpha S[f_i[n, m]]$$

- Superposition

$$S[\alpha f_i[n, m] + \beta f_j[n, m]] = \alpha S[f_i[n, m]] + \beta S[f_j[n, m]]$$

- ▣ Shift invariance:

$$f[n - n_0, m - m_0] \xrightarrow{S} g[n - n_0, m - m_0]$$

**Meaning:** Shifting the input shifts the output by the same amount **without changing the system's behavior**.

# Filtering an impulse signal

What is the result of filtering the impulse signal (image)  $F$  with the arbitrary kernel  $H$ ?

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$



a	b	c
d	e	f
g	h	i

$H[u, v]$


$G[x, y]$

# Filtering an impulse signal - correlation

What is the result of filtering the impulse signal (image)  $F$  with the arbitrary kernel  $H$ ?

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x, y]$



a	b	c
d	e	f
g	h	i

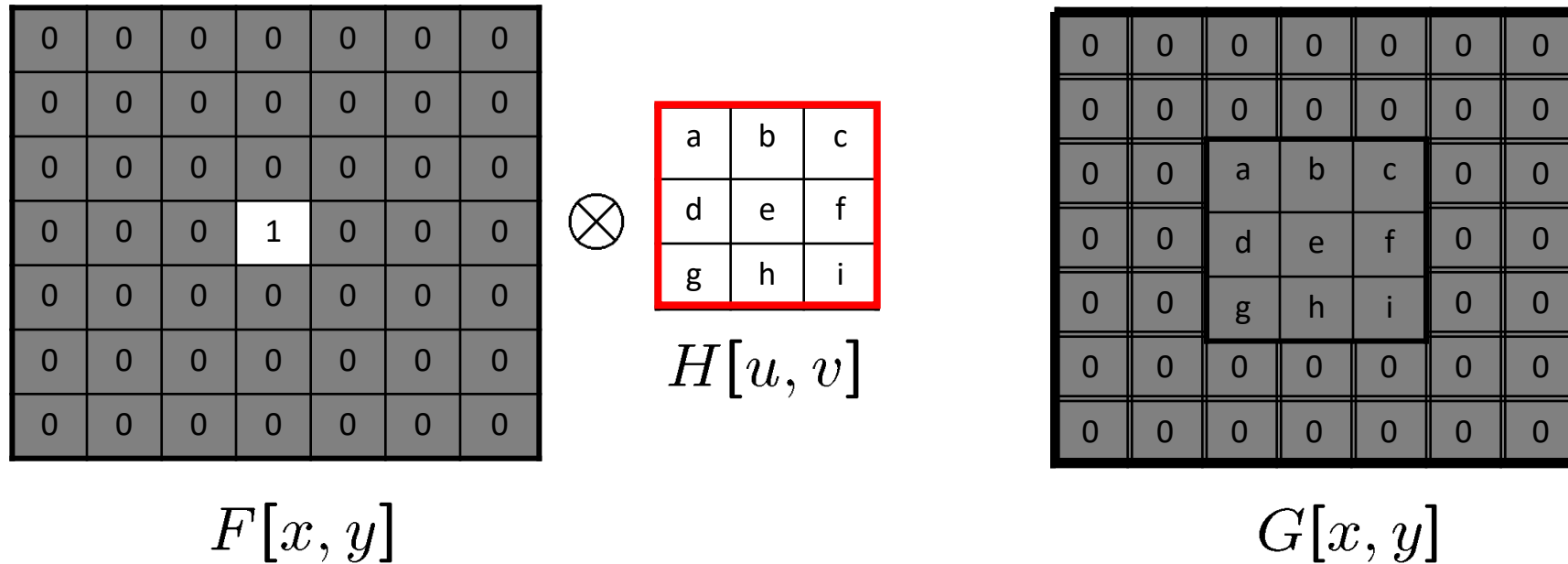
$H[u, v]$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	i	h	g	0	0
0	0	f	e	d	0	0
0	0	c	b	a	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$G[x, y]$

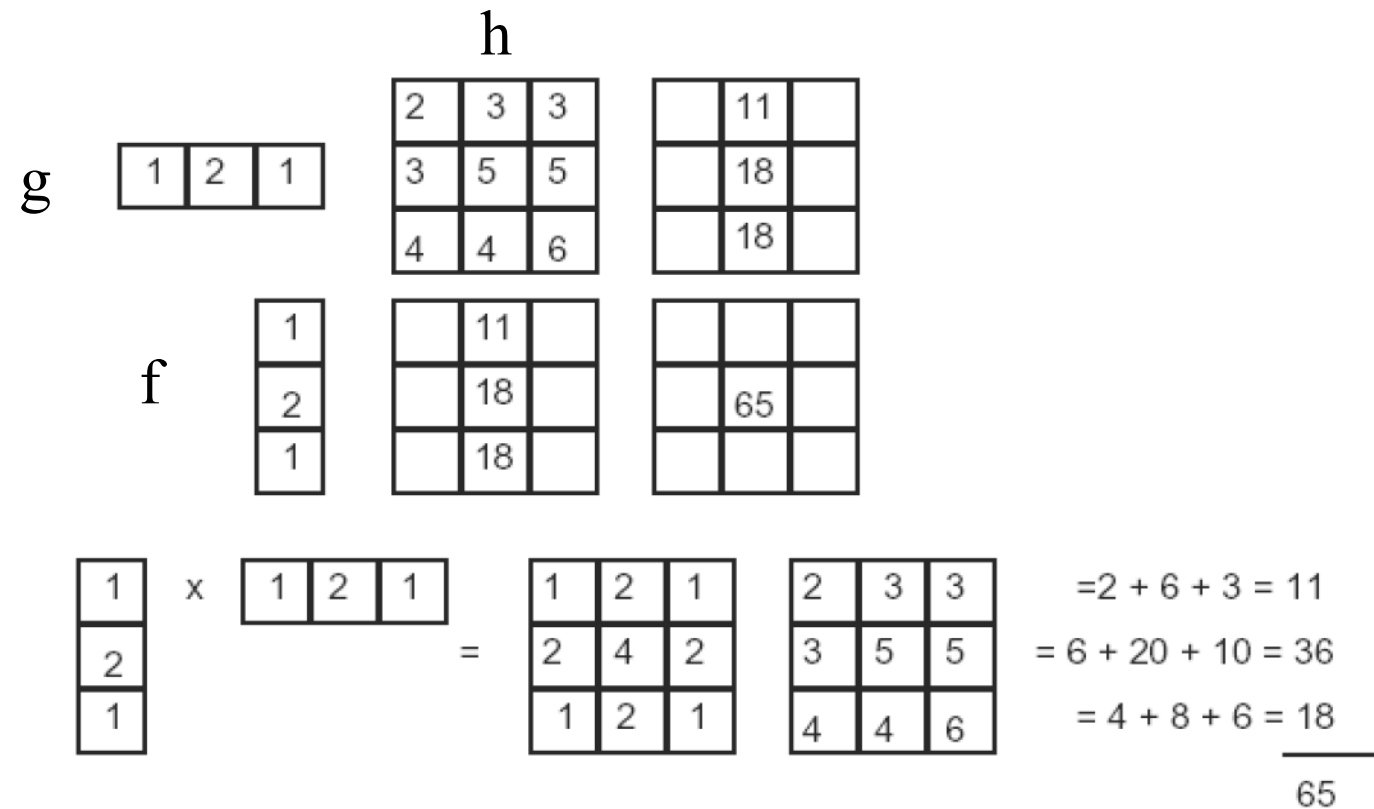
# Filtering an impulse signal - Convolution

What is the result of filtering the impulse signal (image)  $F$  with the arbitrary kernel  $H$ ?



# Separability

- In some cases, filter is separable, and we can factor into two steps: e.g.,
  - ▣ Convolve all rows
  - ▣ Convolve all columns



$$f * (g * h) = (f * g) * h$$

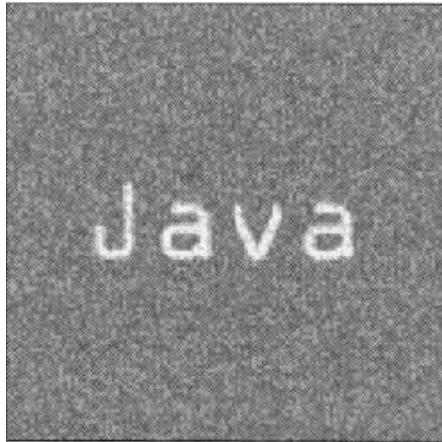
# Separability

- What is the computational complexity advantage for a separable filtering terms of number of operations per output pixel?
- Filtering an  $M$ -by- $N$  image with a  $P$ -by- $Q$  filter kernel requires roughly  $MNPQ$  multiplies and adds (assuming we aren't using an implementation based on the FFT).
- If the kernel is separable, you can filter in two steps.
  - ▣ The first step requires about  $MNP$  multiplies and adds.
  - ▣ The second requires about  $MNQ$  multiplies and adds
  - ▣ for a total of  $MN(P + Q)$ .
- The computational advantage of separable convolution versus nonseparable convolution is therefore:  $PQ/(P+Q)$
- For a 9-by-9 filter kernel, that's a theoretical speed-up of 4.5.



# Correlation (cont'd)

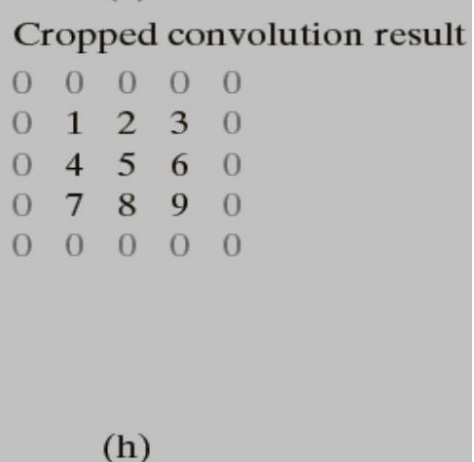
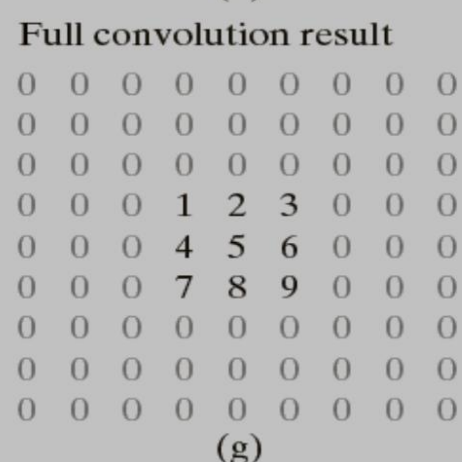
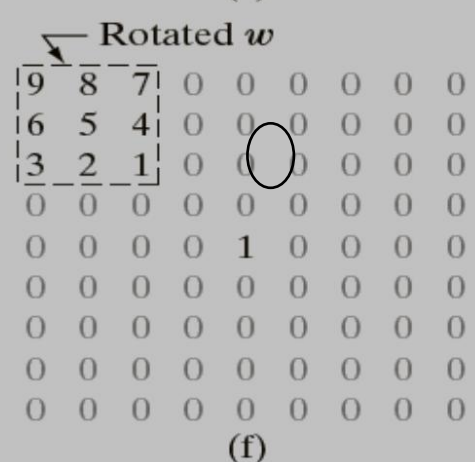
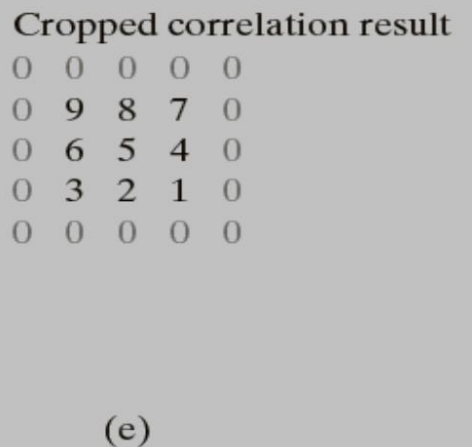
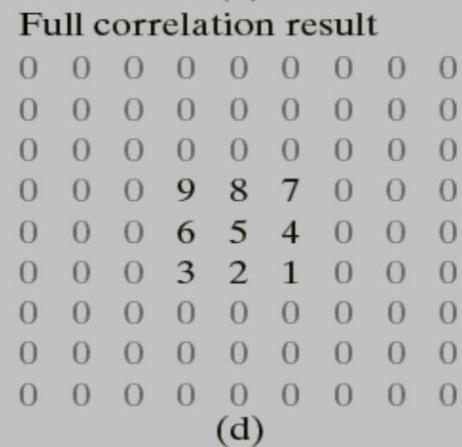
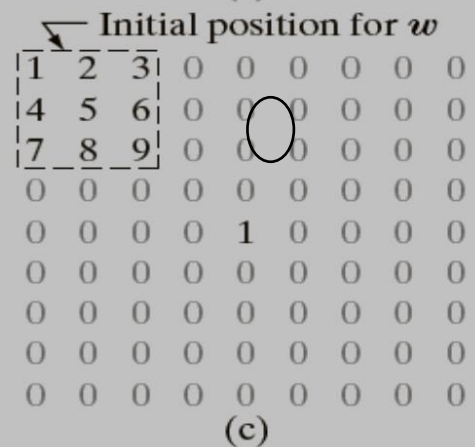
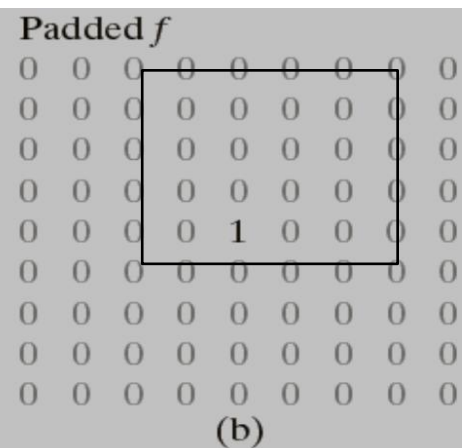
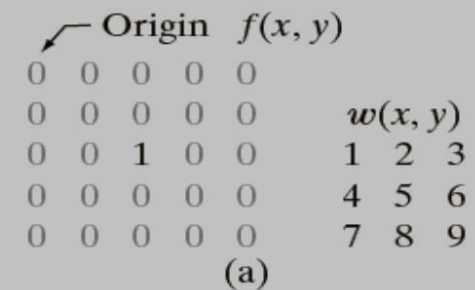
Often used in applications where we need to measure the similarity between images or parts of images (e.g., [template matching](#)).



# Example

Correlation:

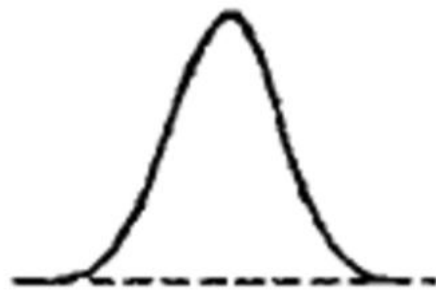
Convolution:



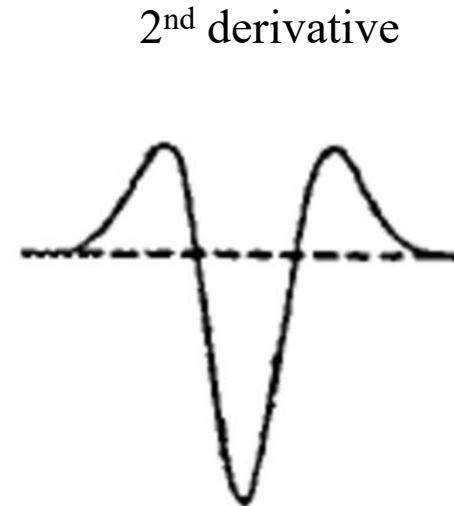
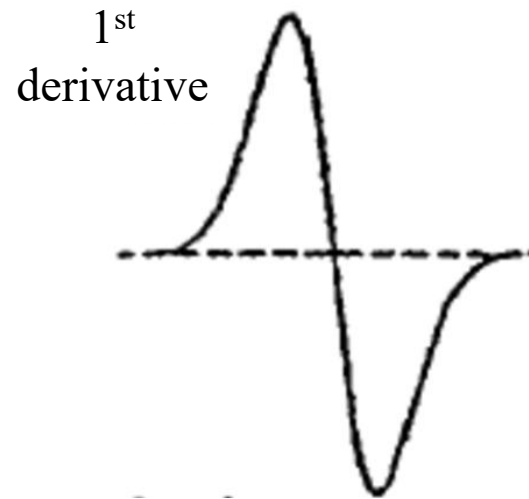
# How do we choose the mask weights?

- Typically, by sampling certain functions:

w1	w2	w3
w4	w5	w6
w7	w8	w9



Gaussian



# Filters for Image Enhancement

---

- We will mainly focus on two types of filters:
  - ▣ Smoothing (low-pass)
  - ▣ Sharpening (high-pass)

# Smoothing Spatial Filters

- Smoothing filters are used for blurring and for noise reduction.
  - ▣ Blurring is used in preprocessing steps, such as removal of small details from an image prior to object extraction, and bridging of small gaps in lines or curves
    - Noise reduction can be accomplished by blurring
    - Reduce “sharp” transitions in intensities
    - smoothing of false contours
- There are 2 way of smoothing spatial filters
  - ▣ Smoothing Linear Filters
  - ▣ Order-Statistics Filters

# Smoothing Linear Filters

- Simply the average of the pixels contained in the neighborhood of the mask.
- Sometimes called “averaging filters”
- Two 3x3 Smoothing Linear Filters

$$\frac{1}{9} \times$$

1	1	1
1	1	1
1	1	1

Standard average

$$\frac{1}{16} \times$$

1	2	1
2	4	2
1	2	1

Weighted average

# Smoothing Linear Filters

- 5x5 mask

$$\frac{1}{25} \times \begin{array}{|c|c|c|c|c|} \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline \end{array}$$

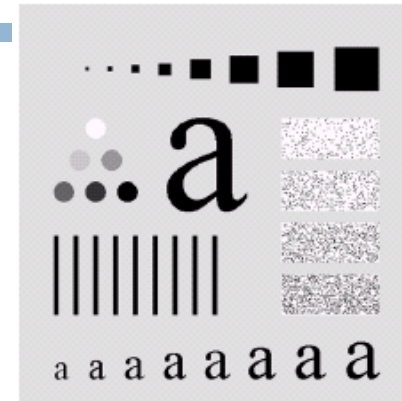
- The general implementation for filtering an  $M \times N$  image with a weighted averaging filter of size  $m \times n$  is given by the expression

$$g(x, y) = \frac{\sum_{s=-a}^a \sum_{t=-b}^b h(s, t) f(x-s, y-t)}{\sum_{s=-a}^a \sum_{t=-b}^b h(s, t)}$$

# Smoothing Linear Filters - example

- Mask size determines the degree of smoothing (loss of detail).

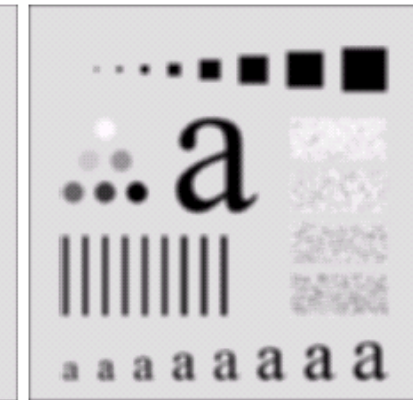
original



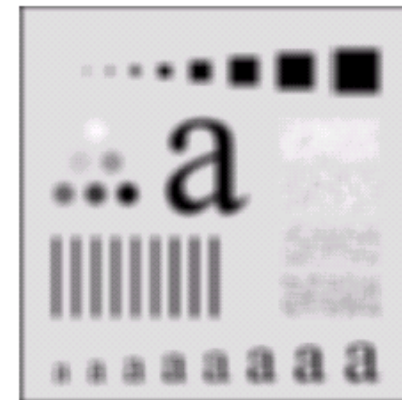
3x3



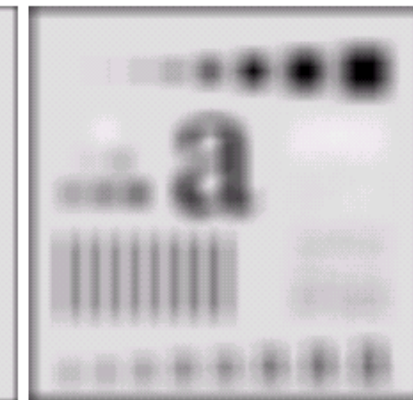
5x5



9x9



15x15

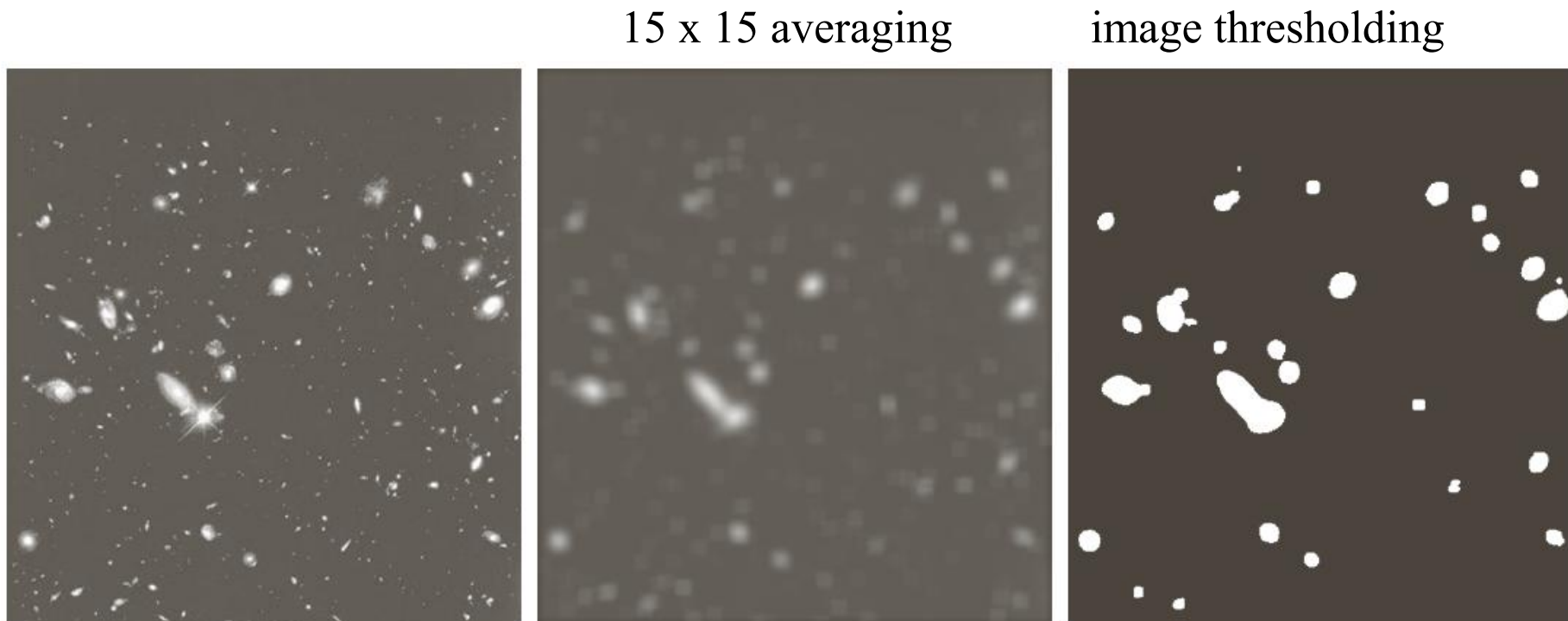


35x35



# Smoothing Linear Filters - example

**Example:** extract largest, brightest objects



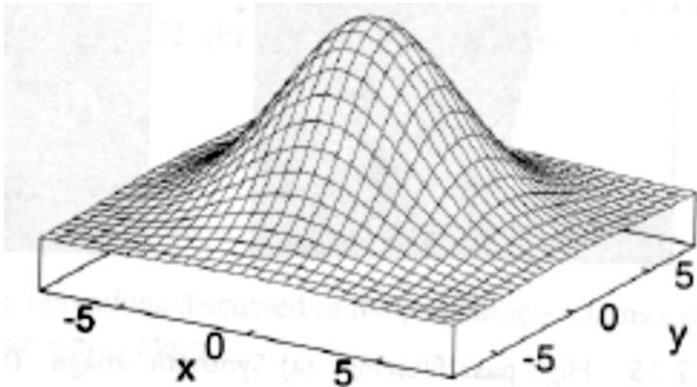
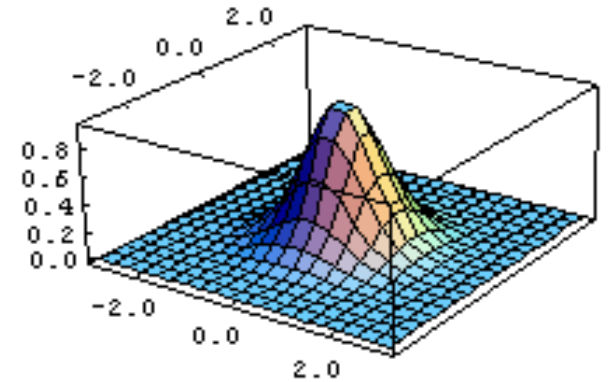
# Smoothing filters: Gaussian filter

- This kernel is an approximation of a Gaussian function:

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

mask size is  
a function of  $\sigma$ :

*height = width =  $5\sigma$  (subtends 98.76% of the area)*



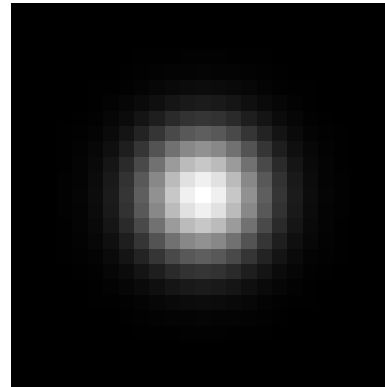
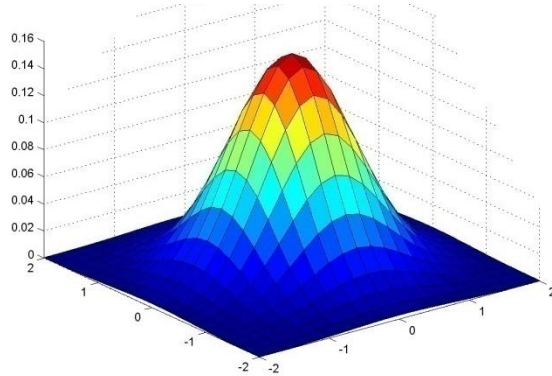
7 × 7 Gaussian mask

1	1	2	2	2	1	1
1	2	2	4	2	2	1
2	2	4	8	4	2	2
2	4	8	16	8	4	2
2	2	4	8	4	2	2
1	2	2	4	2	2	1
1	1	2	2	2	1	1

$$\sigma = 1.4$$

# Gaussian filter

- Weight contributions of neighboring pixels by nearness



0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

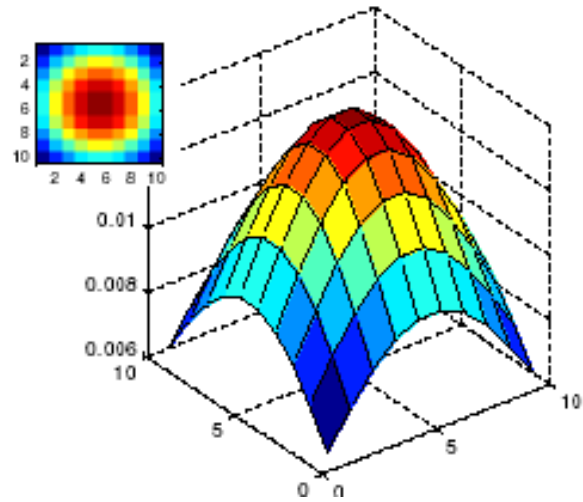
5 × 5,  $\sigma = 1$

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

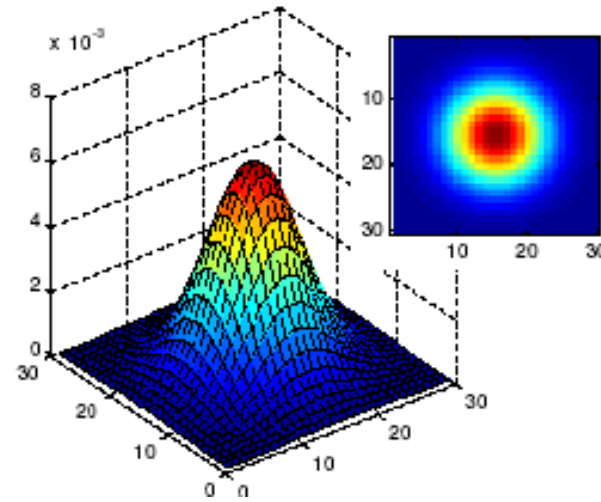
1	4	7	4	1
4	20	33	20	4
7	33	55	33	7
4	20	33	20	4
1	4	7	4	1

# Gaussian filters

- What parameters matter here?
- **Size of kernel or mask**
  - ▣ Note, Gaussian function has infinite support, but discrete filters use finite kernels



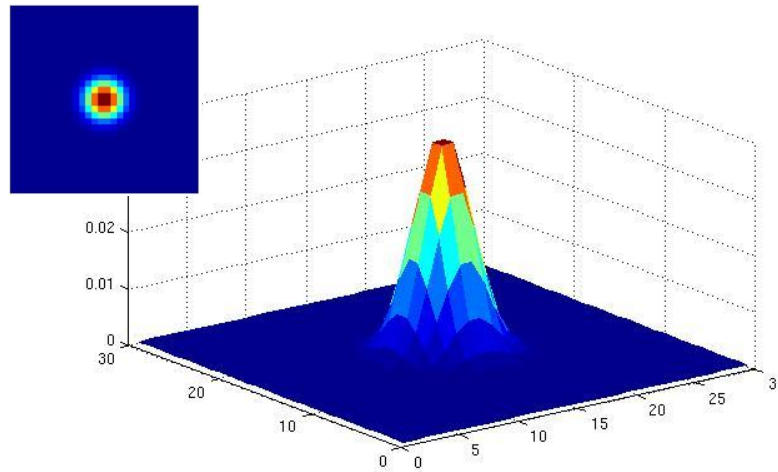
$\sigma = 5$  with 10 x 10  
kernel



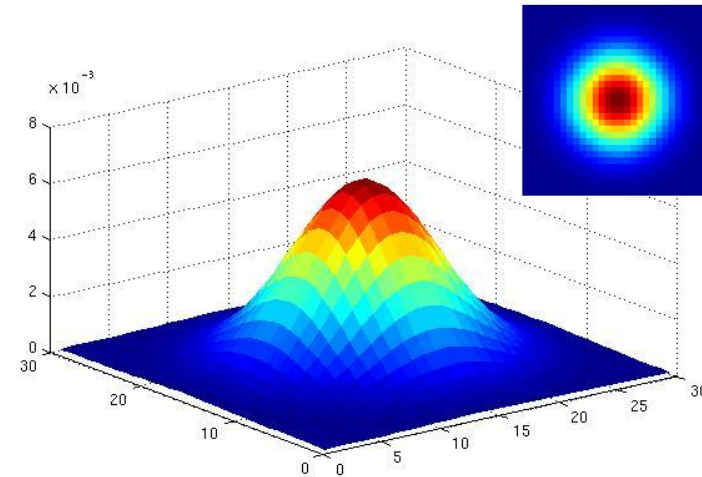
$\sigma = 5$  with 30 x 30  
kernel

# Gaussian filters

- What parameters matter here?
- **Std Deviation** of Gaussian: determines extent of smoothing



$\sigma = 2$  with 30  
x 30 kernel

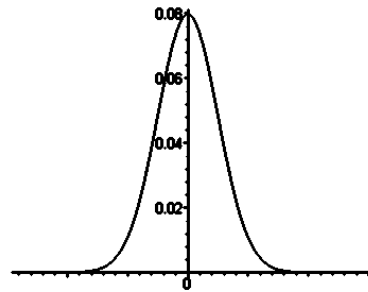


$\sigma = 5$  with 30  
x 30 kernel

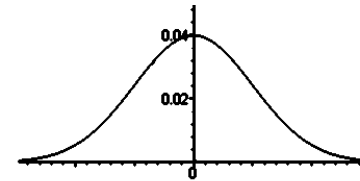
# Smoothing filters: Gaussian (cont'd)



Original Image



Small  $\sigma$



Large  $\sigma$



Limited smoothing



Strong smoothing

# Gaussian filters

- Remove “high-frequency” components from the image (low-pass filter)
  - ▣ Images become more smooth
- Convolution with self is another Gaussian
  - ▣ **Prove it.**
  - ▣ So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  - ▣ Convoluting two times with Gaussian kernel of width  $\sigma$  is same as convoluting once with kernel of width  $\sigma\sqrt{2}$
- *Separable* kernel
  - ▣ Factors into product of two 1D Gaussians

# Separability of the Gaussian filter

$$\begin{aligned} G_{\sigma}(x, y) &= \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2 + y^2}{2\sigma^2}} \\ &= \left( \frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}} \right) \left( \frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{y^2}{2\sigma^2}} \right) \end{aligned}$$

The 2D Gaussian can be expressed as the product of two functions, one a function of  $x$  and the other a function of  $y$

In this case, the two functions are the (identical) 1D Gaussian



# Nonlinear spatial filtering

---

- Nonlinear spatial filters also operate on neighborhoods, and the mechanics of sliding a mask past an image are the same as was just outlined.
- The filtering operation is based conditionally on the values of the pixels in the neighborhood under consideration

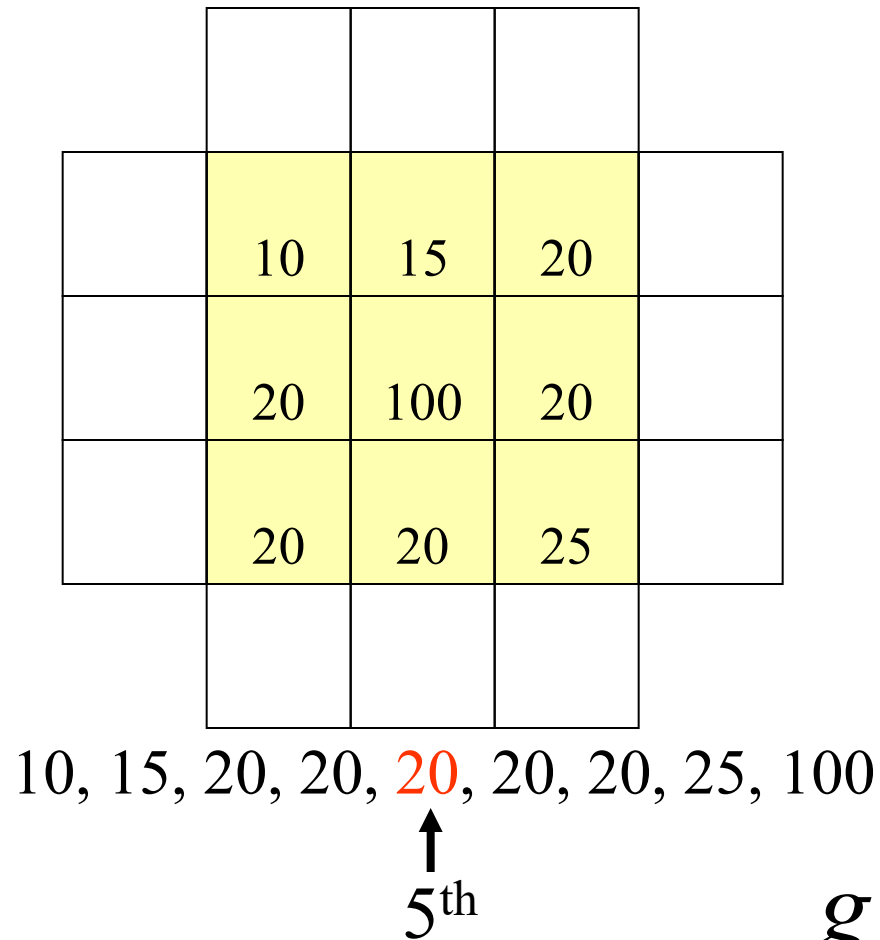
# Order-Statistics Filters

---

- nonlinear spatial filters
- response is based on ordering (ranking) the pixels contained in the image area encompassed by the filter, and then replacing the value of the center pixel with the value determined by the ranking result.
- Best-known “median filter”
  - ▣ Good noise-reduction capabilities with less smoothing (e.g. impulse noise, or salt-and-pepper noise )

# Non-linear smoothing

## Median filter

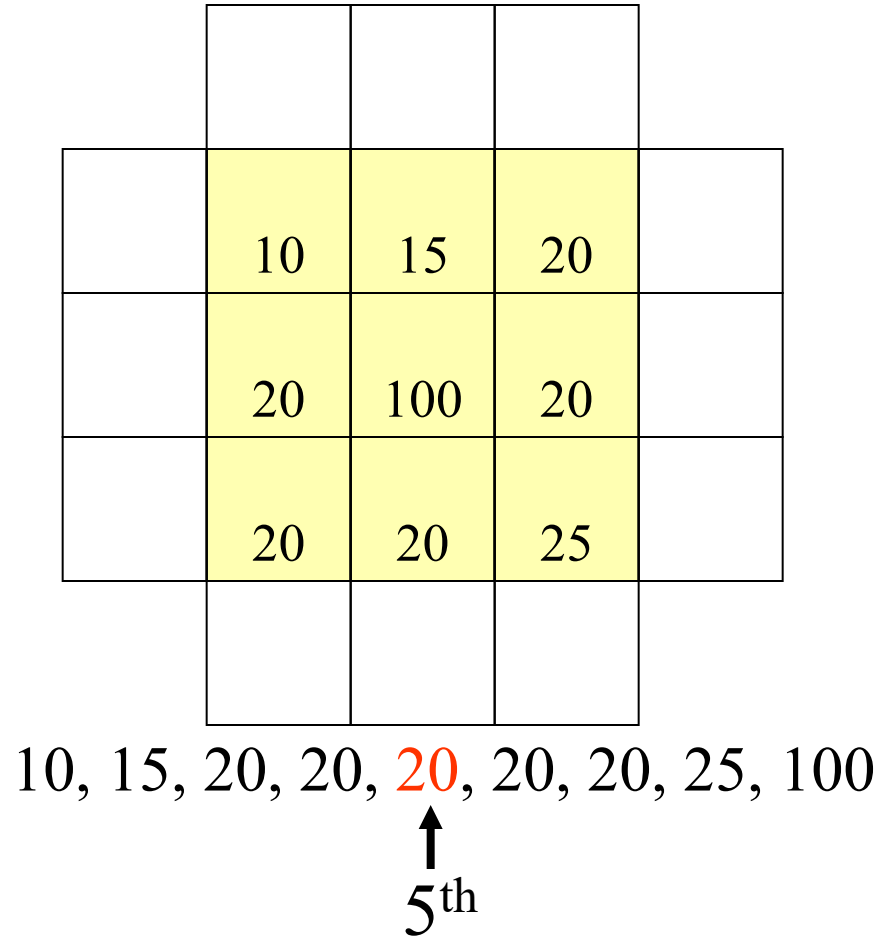


- Crop region of neighborhood
- Sort the values of the pixel in our region
- In the **mxn** mask the median is **(mxn div 2)+1**

$$g(x, y) = \underset{(s, t) \in W_{(x, y)}}{median} \{f(s, t)\}$$

# Non-linear smoothing

## Median filter



- It is widely used as it is very effective at removing noise while **preserving edges**.
- It is particularly effective at removing 'salt and pepper' type noise.

# Smoothing filters: **Box filter**

- This kernel is an approximation of a Gaussian function:

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

$F[x, y]$

1	1	1
1	1	1
1	1	1

$\frac{1}{9}$

$H[u, v]$

-	-	-	-	-	-	-	-	-	-
-	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	-
-	-	-	-	-	-	-	-	-	-

# Smoothing filters: Gaussian filter

- This kernel is an approximation of a Gaussian function:

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

$F[x, y]$

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$H[u, v]$

-	-	-	-	-	-	-	-	-	-
-	0	6	17	23	23	23	17	6	-
-	0	17	51	68	68	68	51	17	-
-	0	23	68	90	90	90	68	23	-
-	0	23	62	79	84	90	68	23	-
-	0	23	56	68	79	90	68	23	-
-	0	17	45	56	62	68	51	17	-
-	6	17	23	23	23	23	17	6	-
-	11	23	11	0	0	0	0	0	-
-	-	-	-	-	-	-	-	-	-

# Smoothing filters: Median 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

$F[x, y]$

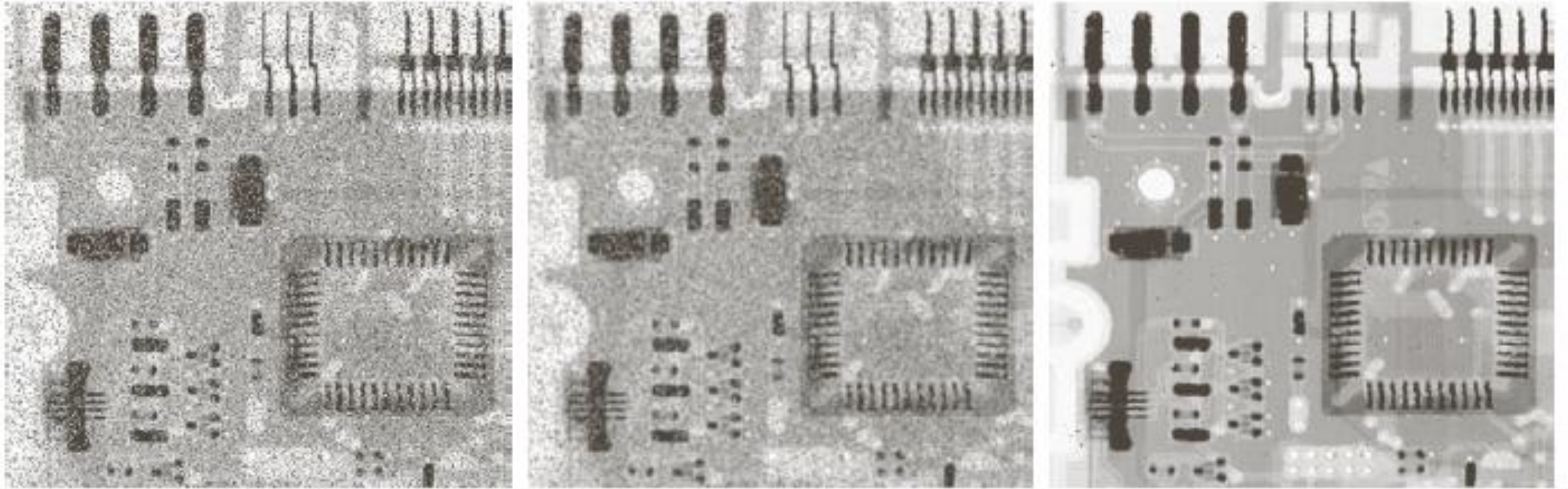
-	-	-	-	-	-	-	-	-	-
-	0	0	0	0	0	0	0	0	-
-	0	0	0	90	90	90	0	0	-
-	0	0	90	90	90	90	90	0	-
-	0	0	90	90	90	90	90	0	-
-	0	0	90	90	90	90	90	0	-
-	0	0	0	90	90	90	0	0	-
-	0	0	0	0	0	0	0	0	-
-	0	0	0	0	0	0	0	0	-
-	-	-	-	-	-	-	-	-	-

# Non-linear smoothing- Median filtering

- nonlinear
  - ▣  $\text{median}\{x(m) + y(m)\} \neq \text{median}\{x(m)\} + \text{median}\{y(m)\}$
- Odd window size is commonly used
  - ▣ 3x3, 5x5, 7x7
  - ▣ 5-pixel “+”-shaped window
- for even-sized windows take the average of two middle values as output
- resilient to statistical outliers
- incurs less blurring
- simple to implement



# Median filter example



a b c

**FIGURE 3.35** (a) X-ray image of circuit board corrupted by salt-and-pepper noise. (b) Noise reduction with a  $3 \times 3$  averaging mask. (c) Noise reduction with a  $3 \times 3$  median filter. (Original image courtesy of Mr. Joseph E. Pascente, Lixi, Inc.)

# median filter



Noisy Image



median (f)

3x3 neighborhood

# Degraded Image



Source: Freeman and Durand

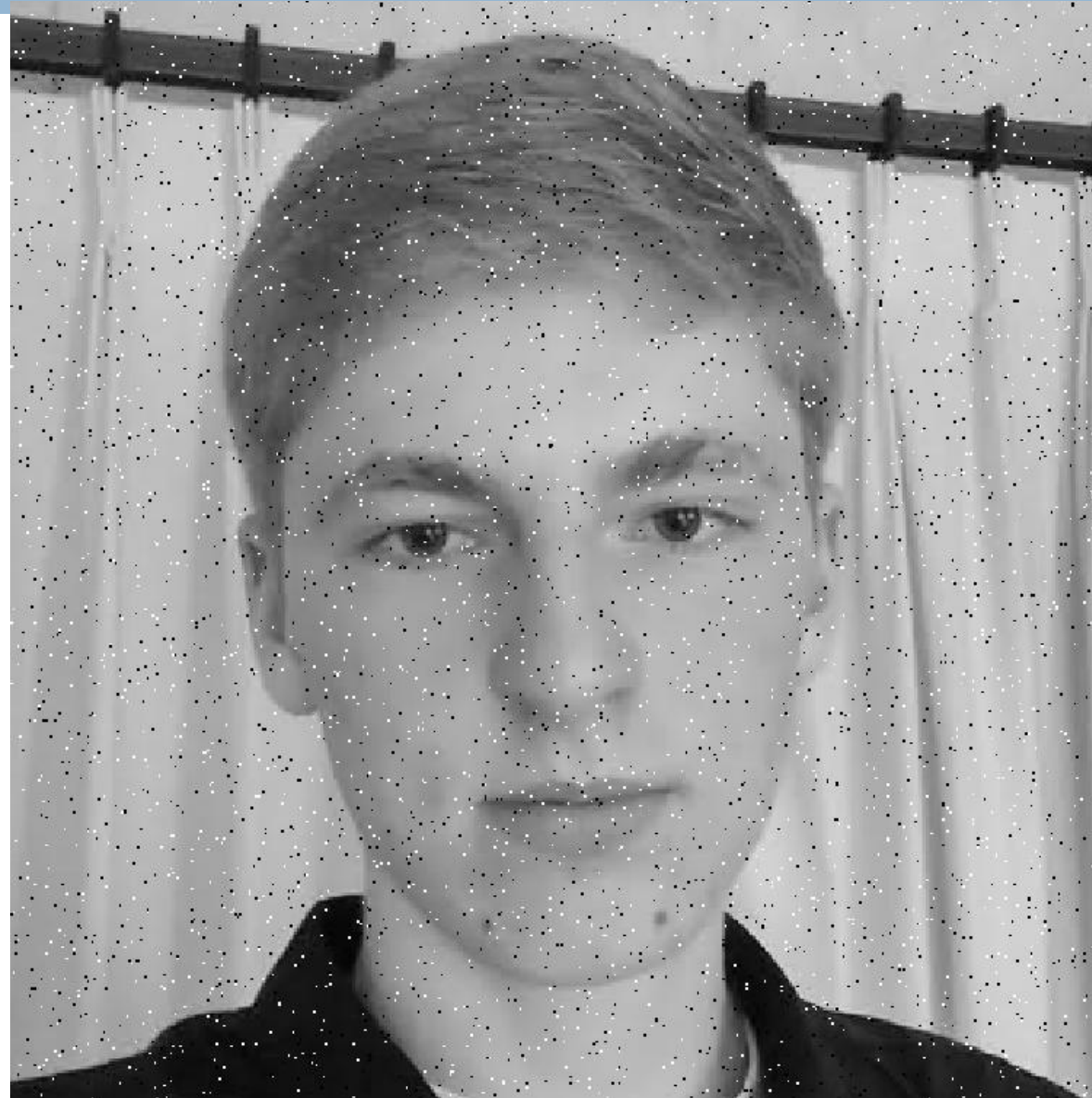


# 3x3 median filter



Source: Freeman and Durand

# Noise – Salt and Pepper Jack



# Mean Jack – 3 x 3 filter



## Very Mean Jack – 11 x 11 filter



# Noisy – Salt and Pepper Jack





# Median Jack – 3 x 3

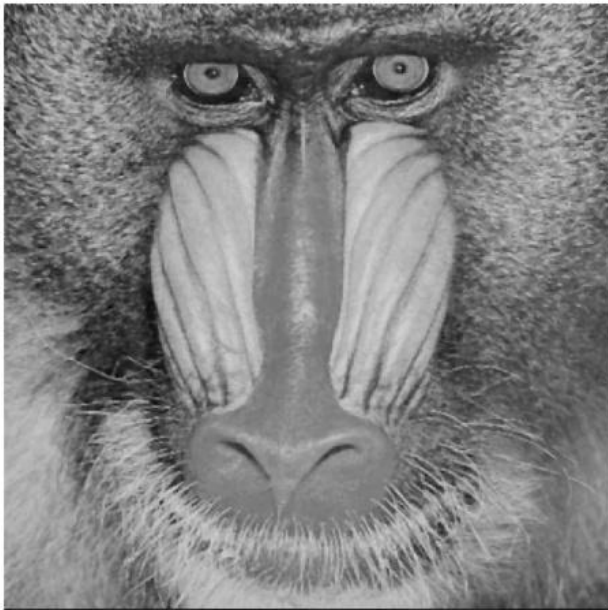
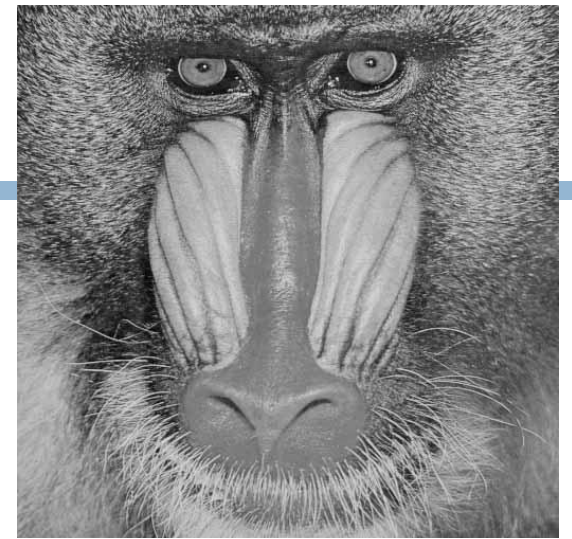


# Order statistics filters

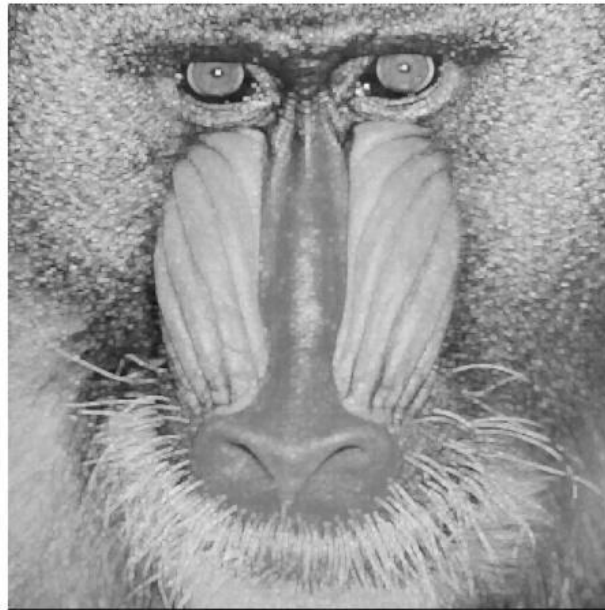
## Other order statistics:

- Min filter, max filter, x-percentile ...

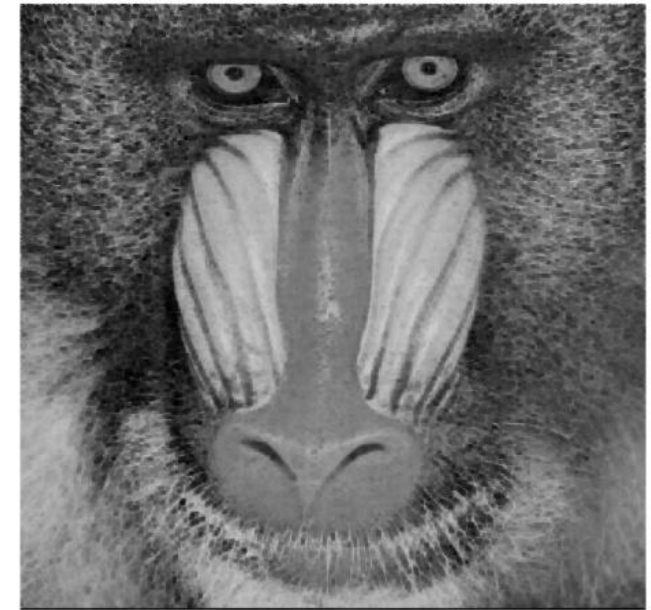
original



$$g(x, y) = \underset{(s, t) \in W_{(x, y)}}{\text{median}} \{f(s, t)\}$$



$$g(x, y) = \min_{(s, t) \in W_{(x, y)}} \{f(s, t)\}$$



$$g(x, y) = \max_{(s, t) \in W_{(x, y)}} \{f(s, t)\}$$

# Sharpening Spatial Filters

---

- The principal objective
  - ▣ highlight fine detail in an image or to enhance detail that has been blurred
    - either in error or as a natural effect of a particular method of image acquisition.
- Normally, sharpening is done by highlighting the transitions in intensity
- Methods
  - ▣ Unsharp masking
  - ▣ High Boost filtering
  - ▣ Laplacian ( $2^{\text{nd}}$  order derivative)
  - ▣ Gradient ( $1^{\text{st}}$  order derivative)

# Unsharp masking and High boost Filtering

1. Blur the original image  $f(x,y)$
2. Subtract the blurred image  $\bar{f}(x,y)$  from the original (result called the mask):

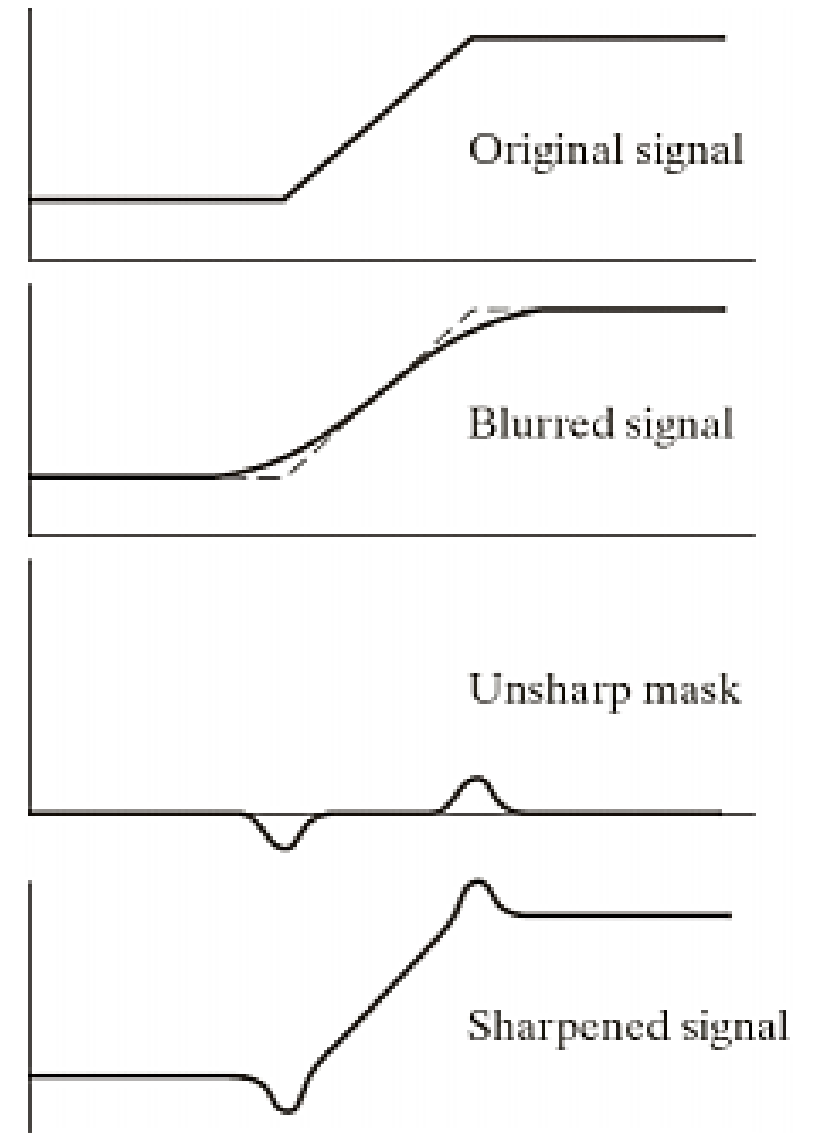
$$g_{mask}(x,y) = f(x,y) - \bar{f}(x,y)$$

3. Add the mask to the original:

$$g(x,y) = f(x,y) + k \times g_{mask}(x,y)$$

$$\check{I} \leftarrow I + a \cdot (I - \tilde{I}) = (1 + a) \cdot I - a \cdot \tilde{I}.$$

- ~~$k = 1 \Rightarrow$  unsharp masking~~
- $k > 1 \Rightarrow$  highboost filtering



# Example

---



Original



Blurred



Mask (scaled)



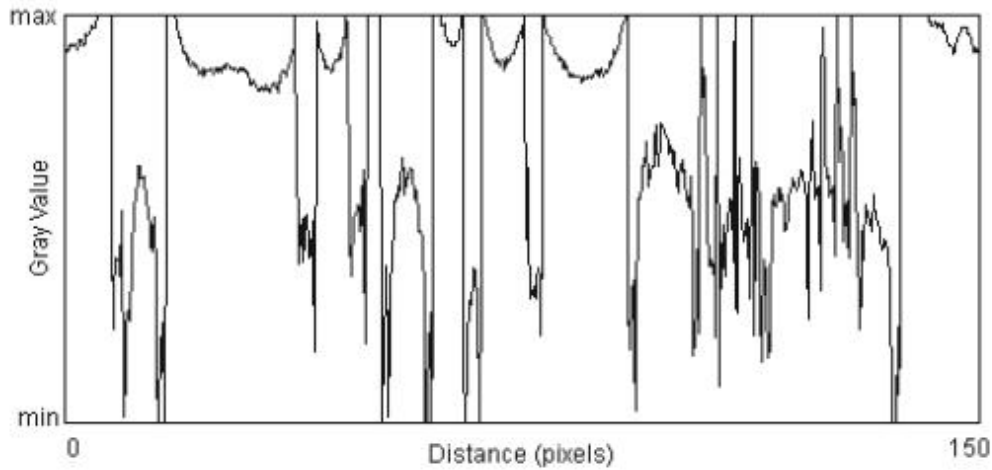
Unsharp Mask



High boost

# Example

Gaussian  
blurring kernel



(c)  $\sigma = 10.0$

# Foundation

## □ Definition of 1<sup>st</sup> / 2<sup>nd</sup> derivative

- ▣ A basic definition of the first-order derivative of a one-dimensional function  $f(x)$  is

$$\frac{\partial f}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h} = f(x+1) - f(x) \quad (h=1)$$

- ▣ We define a second-order derivative as the difference

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

# Finite differences

High-school reminder: definition of a derivative using forward difference

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+h) - f(x)}{h}$$

Alternative: use central difference

$$f'(x) = \lim_{h \rightarrow 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$


For discrete signals: Remove limit and set  $h = 2$

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

What convolution kernel does this correspond to?

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

 ?

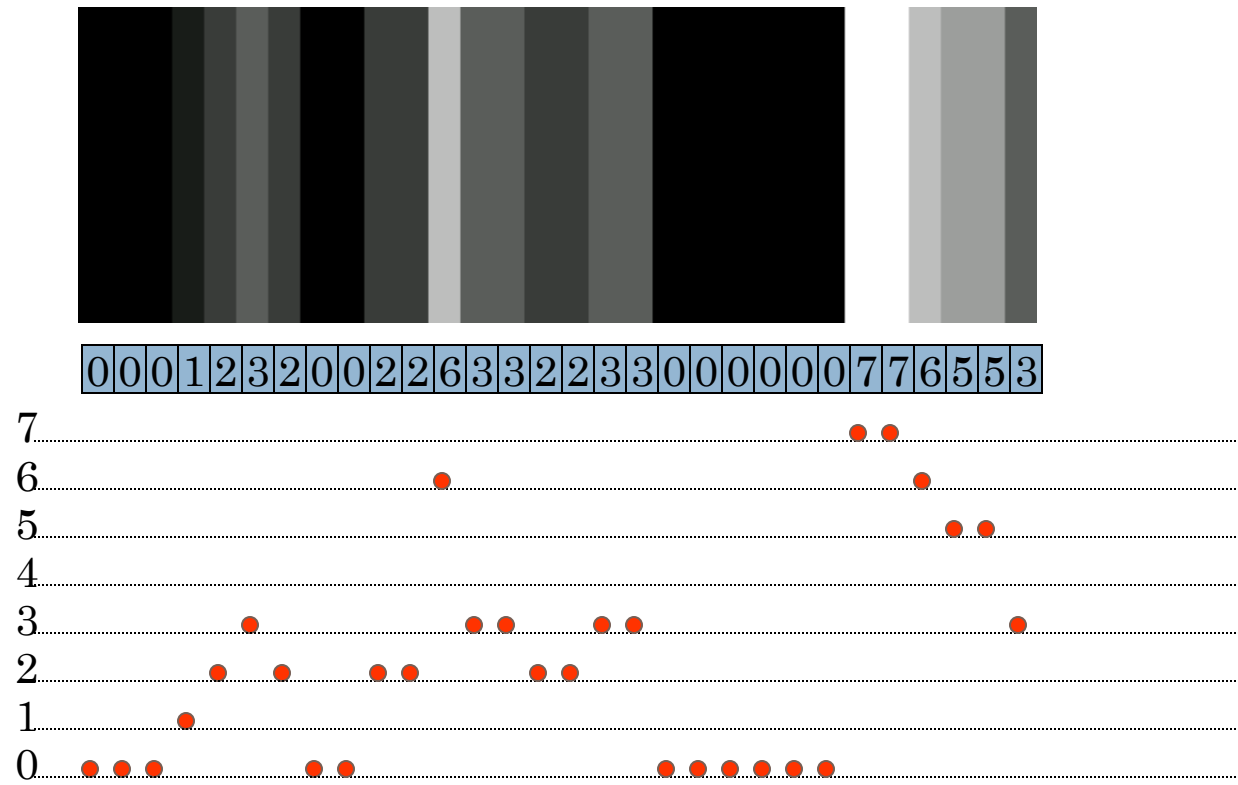


1	0	-1
---	---	----

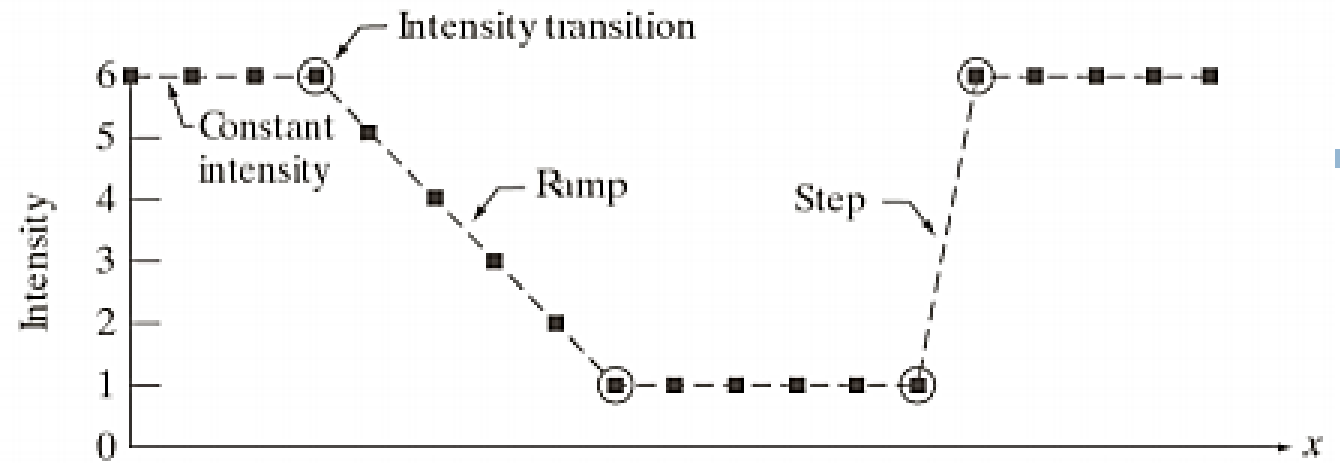
 ?



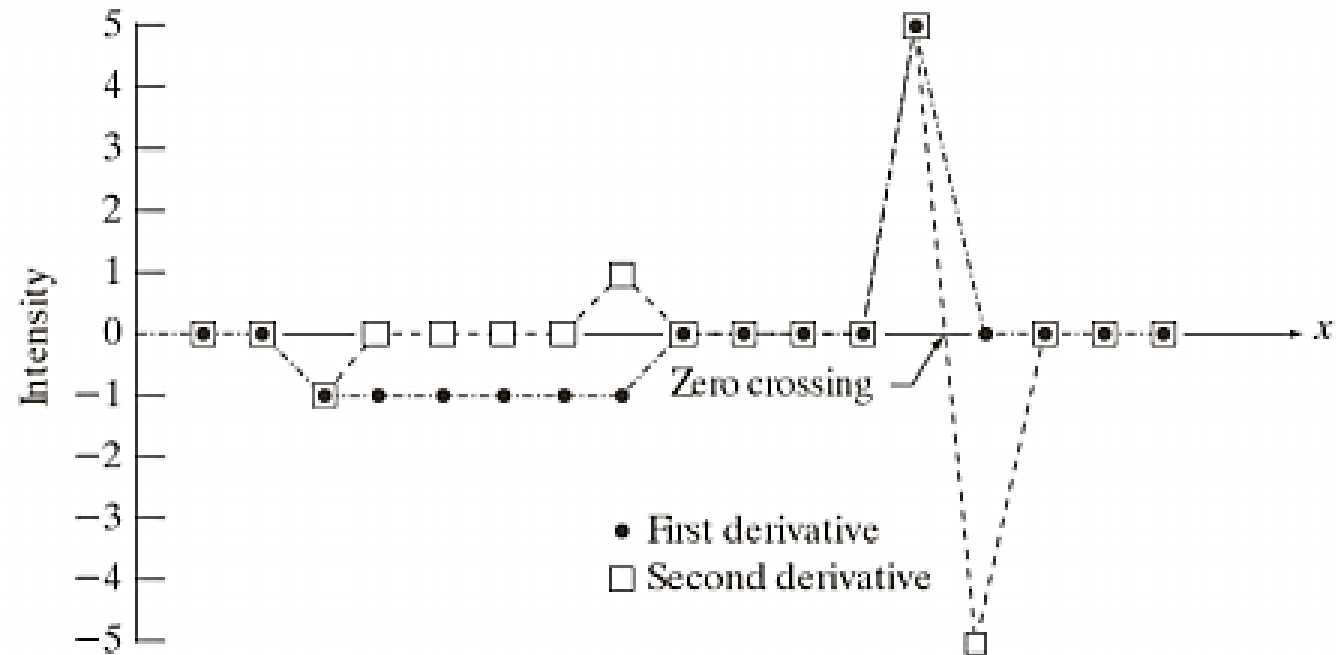
# Gray-level profile



# Foundation



Scan line	6	6	6	6	5	4	3	2	1	1	1	1	1	1	6	6	6	6	6	$\rightarrow x$
1st derivative	0	0	0	-1	-1	-1	-1	-1	0	0	0	0	0	0	5	0	0	0	0	
2nd derivative	0	0	0	-1	0	0	0	0	1	0	0	0	0	0	5	-5	0	0	0	



# Analyze

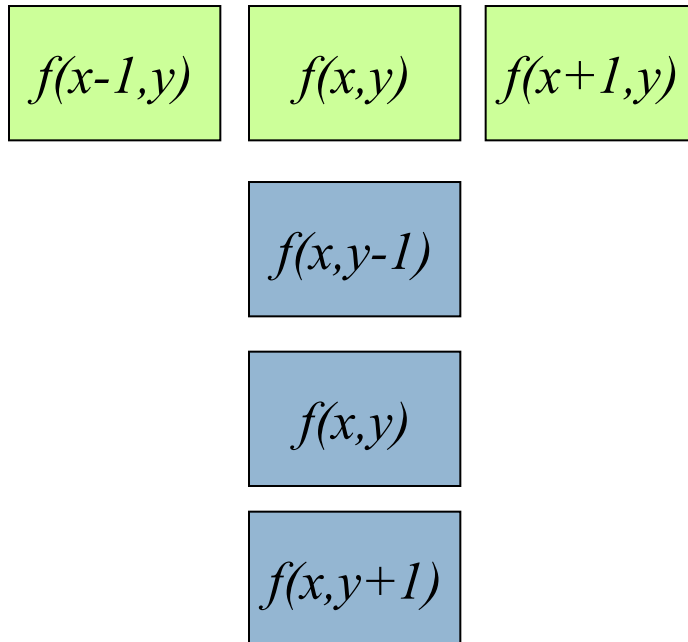
- Both derivative is zero for constant areas
- The 1<sup>st</sup>-order derivative is nonzero along the entire ramp, while the 2<sup>nd</sup>-order derivative is nonzero only at the onset and end of the ramp.
- The response at and around the step point is much stronger for the 2<sup>nd</sup>- than for the 1<sup>st</sup>-order derivative
- Edges in digital images often are ramp like
  - ▣ 1<sup>st</sup> derivative make thick edge and 2<sup>nd</sup> derivative make a thin double edge
  - ▣ 2<sup>nd</sup> derivative is better than 1<sup>st</sup> to enhance fine detail

# The Laplacian (2<sup>nd</sup> order derivative)

- Shown by Rosenfeld and Kak[1982] that the simplest **isotropic** derivative operator is the **Laplacian** is defined as

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

- Discrete form of derivative



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

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

# The Laplacian - mask

- The digital implementation of the 2-D Laplacian is obtained by summing 2 components

$$\nabla^2 f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y)$$

0	0	0
1	-2	1
0	0	0

 + 

0	1	0
0	-2	0
0	1	0

 = 

0	1	0
1	-4	1
0	1	0

Edges can be found  
by detect the zero-crossings

5	5	5	5	5	5
5	5	5	5	5	5
5	5	10	10	10	10
5	5	10	10	10	10
5	5	5	10	10	10
5	5	5	5	10	10

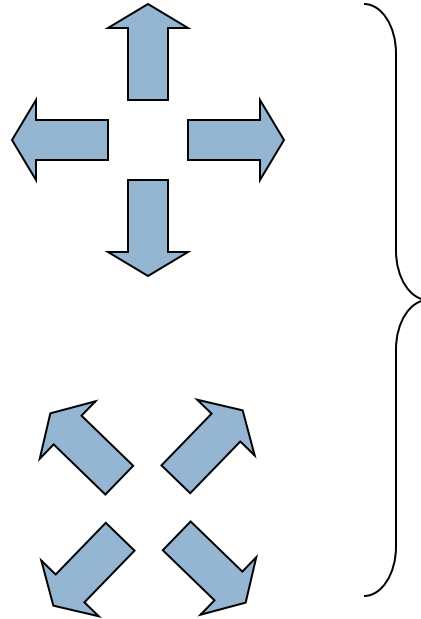
-	-	-	-	-	-
-	0	5	5	5	-
-	5	-10	-5	-5	-
-	5	-10	0	0	-
-	0	10	-10	0	-
-	-	-	-	-	-

# The Laplacian - mask

Isotropic mask  
with  $90^\circ$  rotation  
increment

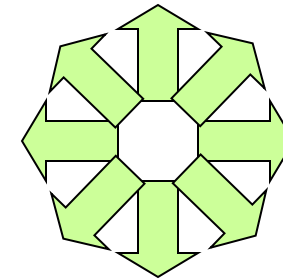
0	1	0
1	-4	1
0	1	0

1	0	1
0	-4	0
1	0	1



Isotropic mask  
with  $45^\circ$  rotation  
increment

1	1	1
1	-8	1
1	1	1

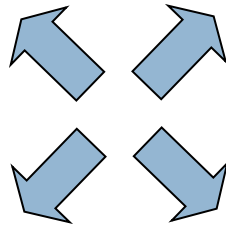
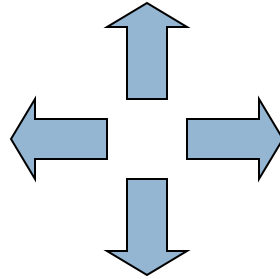


# Laplacian mask

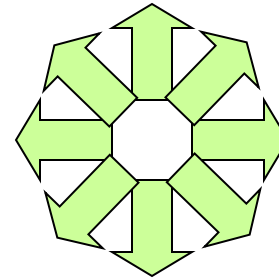
## □ variants

0	-1	0
-1	4	-1
0	-1	0

-1	0	-1
0	4	0
-1	0	-1



-1	-1	-1
-1	8	-1
-1	-1	-1



$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

# Sharpening using Laplacian

$$g(x, y) = \begin{cases} f(x, y) - w \cdot \nabla^2 f(x, y) & \text{If the center coefficient is negative} \\ f(x, y) + w \cdot \nabla^2 f(x, y) & \text{If the center coefficient is positive} \end{cases}$$

Where  $f(x, y)$  is the original image

$\nabla^2 f(x, y)$  is Laplacian filtered image

$g(x, y)$  is the sharpen image

$w$  is sharpening strength

0	-1	0
-1	5	-1
0	-1	0

-1	-1	-1
-1	9	-1
-1	-1	-1



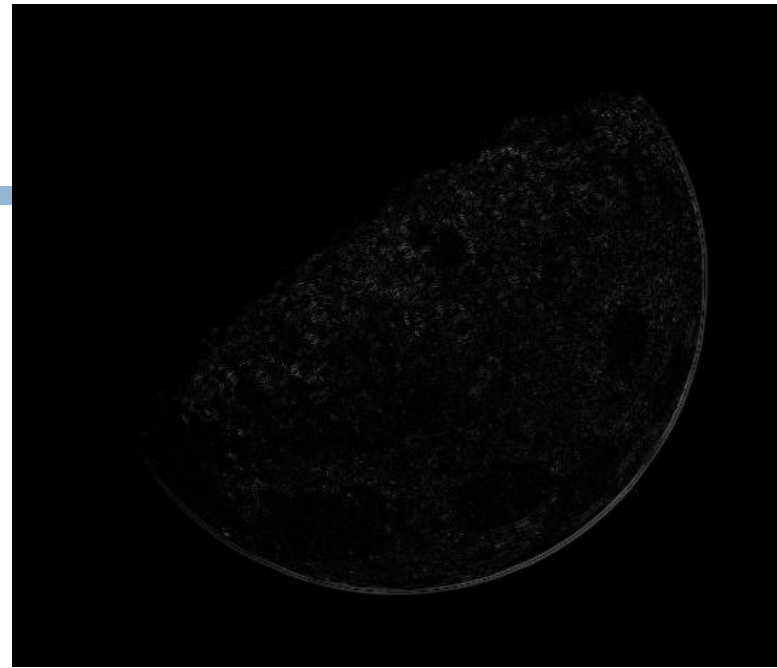
# Laplacian - Example



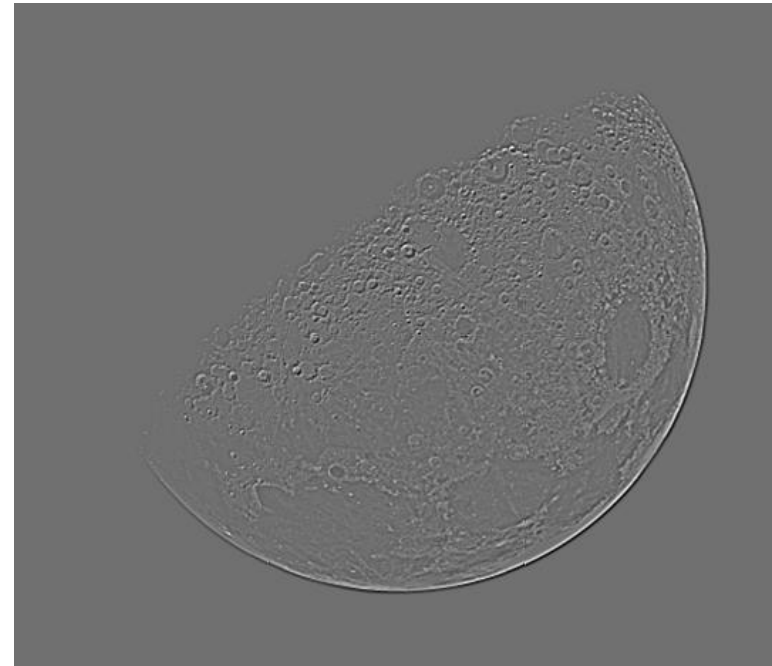
Original

0	-1	0
-1	4	-1
0	-1	0

with scaling



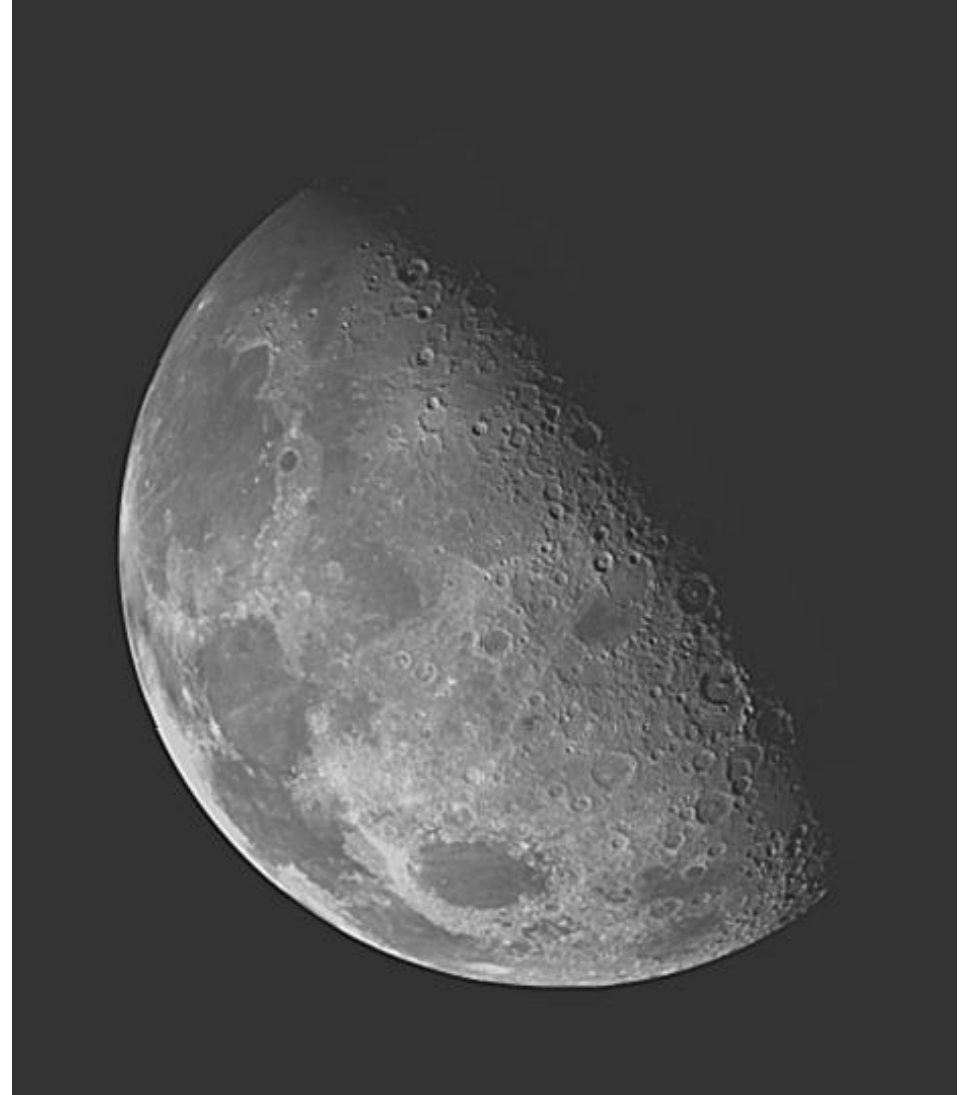
Laplacian without scaling



# Sharpening: Laplacian - example



Original



Sharpened with Laplacian

# The Gradient (1<sup>st</sup> order derivative)

- First Derivatives in image processing are implemented using the magnitude of the gradient.

- The gradient of function  $f(x,y)$  is  $\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$

- The magnitude of this vector

$$mag(\nabla f) = \sqrt{G_x^2 + G_y^2}$$

- Gradient is a linear but not rotation invariant (isotropic)
- Magnitude is not linear, but is isotropic

- Approximation

$$mag(\nabla f) \approx |G_x| + |G_y|$$

Isotropic property lost (in general)

# Sobel's Method

- Mask of even size are awkward to apply.
- The smallest filter mask should be 3x3.

$$\nabla f \approx |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)| + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)|$$

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

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

# The Sobel filter

Horizontal Sober filter:

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

Vertical Sobel filter:

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

local gradient components are obtained from the filter results by appropriate scaling

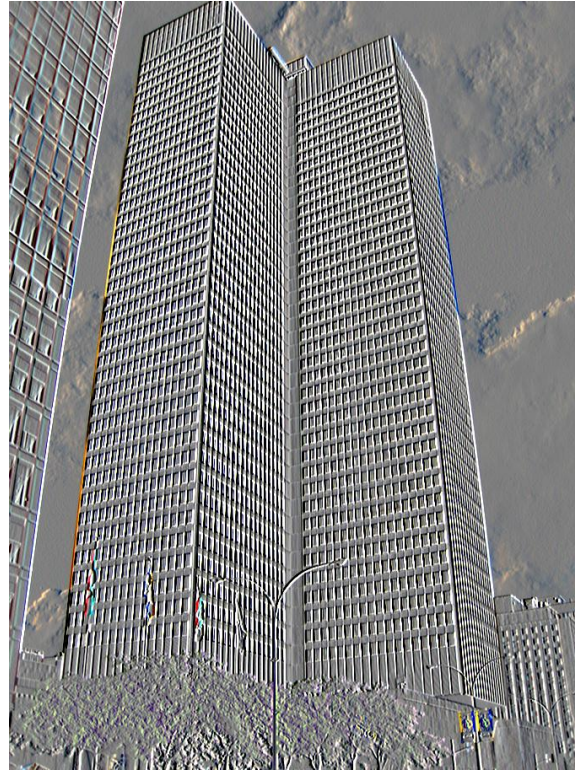
$$\nabla I(u, v) \approx \frac{1}{8} \cdot \begin{pmatrix} (I * H_x^S)(u, v) \\ (I * H_y^S)(u, v) \end{pmatrix}$$



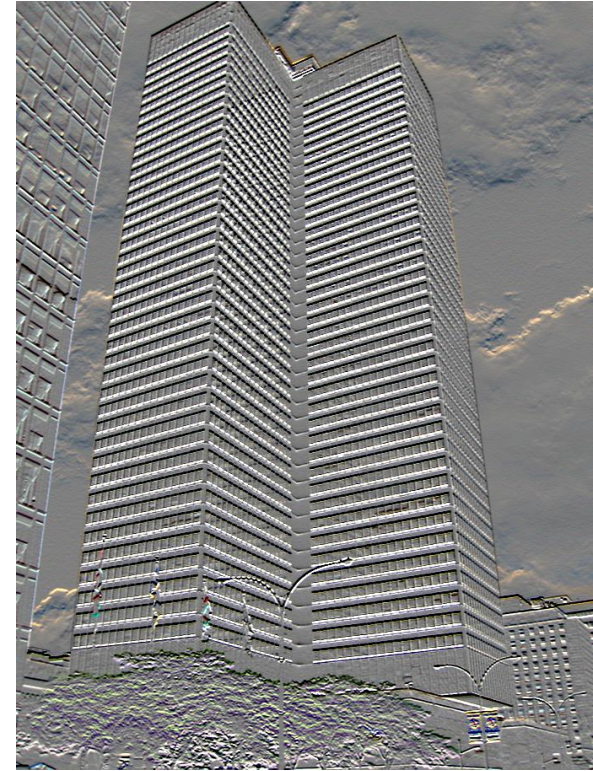
# Sobel filter example



original

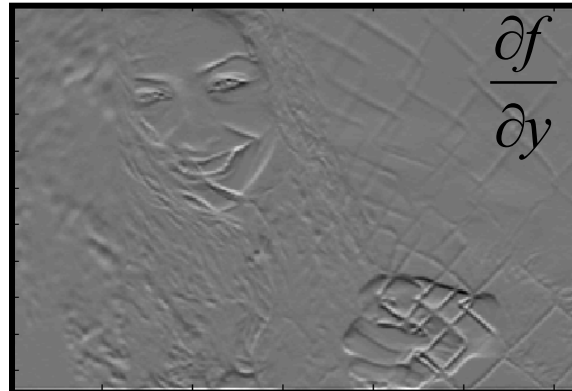
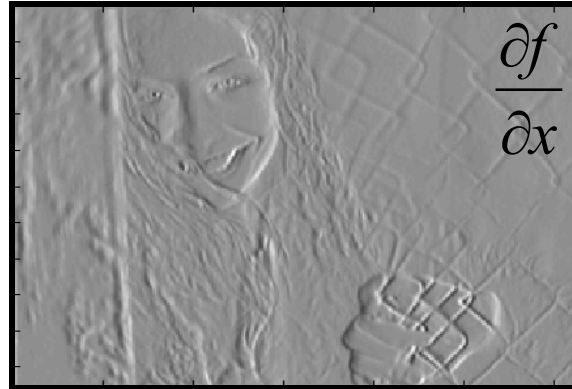


which Sobel filter?  
horizontal



which Sobel filter?  
Vertical

# Example: Gradient Magnitude Image



Gradient Magnitude

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



(isotropic)



# Example: Laplacian vs Gradient



Sobel

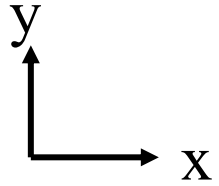


Laplacian



# Other operators

## □ Prewitt



$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -1 & 0 & +1 \\ -1 & 0 & +1 \end{bmatrix}$$

$$G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ +1 & +1 & +1 \end{bmatrix}$$

## □ Scharr

$$G_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix} \quad G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix}$$

## □ Kirsch

$$H_0^K = \begin{bmatrix} -5 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & 3 & 3 \end{bmatrix},$$

$$H_1^K = \begin{bmatrix} -5 & -5 & 3 \\ -5 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix},$$

$$H_2^K = \begin{bmatrix} -5 & -5 & -5 \\ 3 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix},$$

$$H_3^K = \begin{bmatrix} 3 & -5 & -5 \\ 3 & 0 & -5 \\ 3 & 3 & 3 \end{bmatrix},$$

$$H_4^K = \begin{bmatrix} 3 & 3 & -5 \\ 3 & 0 & -5 \\ 3 & 3 & -5 \end{bmatrix}$$

$$H_5^K = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & -5 \\ 3 & -5 & -5 \end{bmatrix}$$

$$H_6^K = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{bmatrix}$$

$$H_7^K = \begin{bmatrix} 3 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & -5 & 3 \end{bmatrix}$$

# Combining Spatial Enhancement Methods

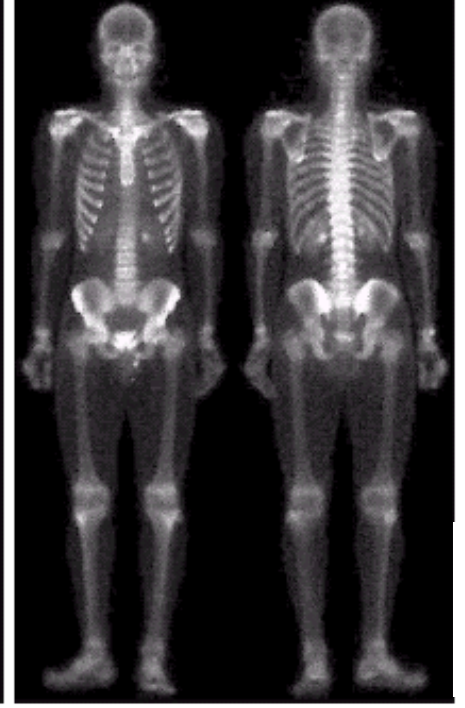
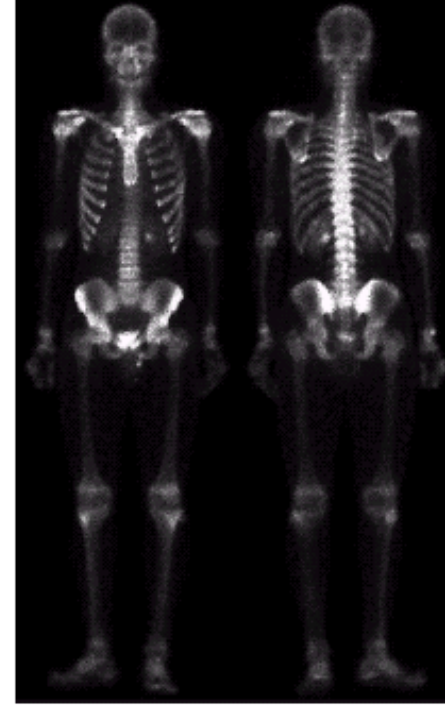
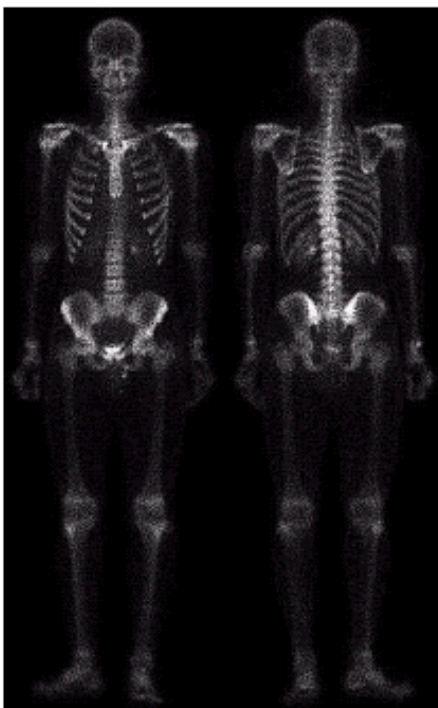
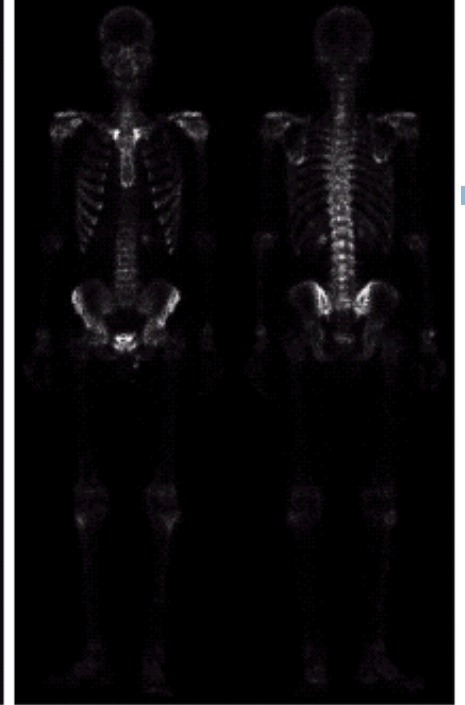
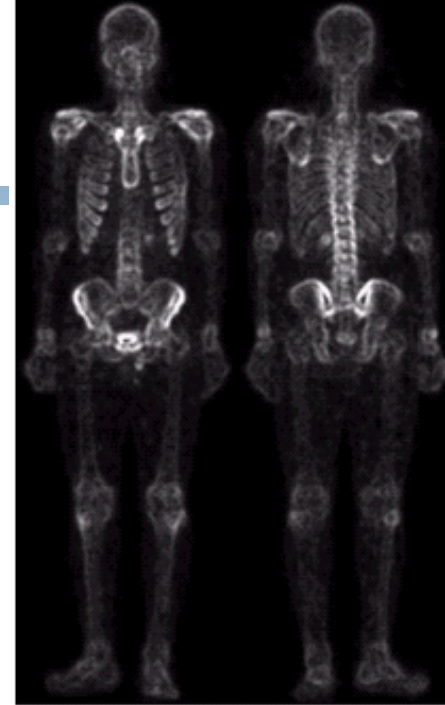
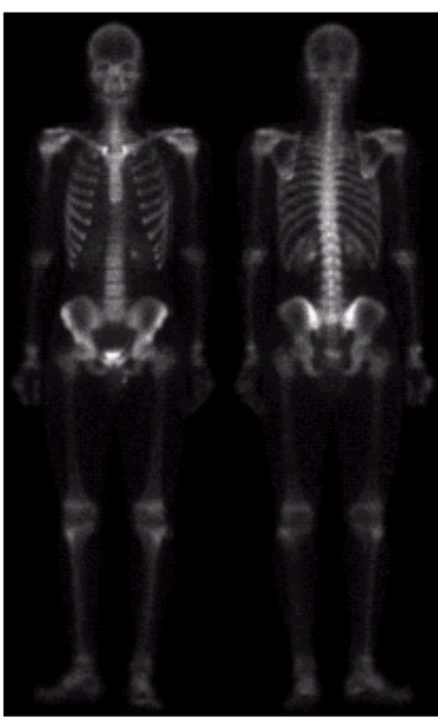
---

- An example
  - ▣ Laplacian to highlight fine detail
  - ▣ Gradient to enhance prominent edges
  - ▣ Smoothed version of the gradient image used to mask the Laplacian image
  - ▣ Increase the dynamic range of the gray levels by using a gray-level transformation

# Application

- a) original
- b) Laplacian
- c)  $a + b$
- d) Sobel
- e) Avg of d
- f)  $c * e$
- g)  $a + f$
- h)  $\gamma$  transform

a	b
c	d



e	f
g	h

# Think-Pair-Share

\* = Convolution operator

a)  $\_ = D * B$

b)  $A = \_ * \_$

c)  $F = D * \_$

d)  $\_ = D * D$

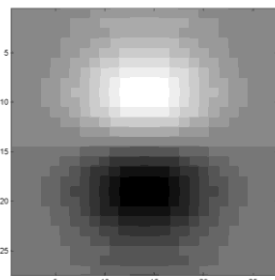
D



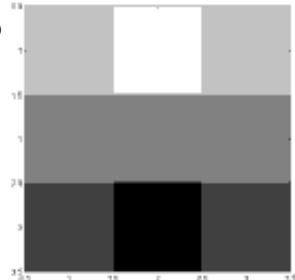
H



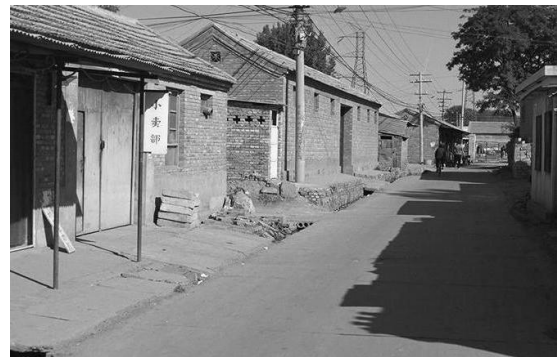
A



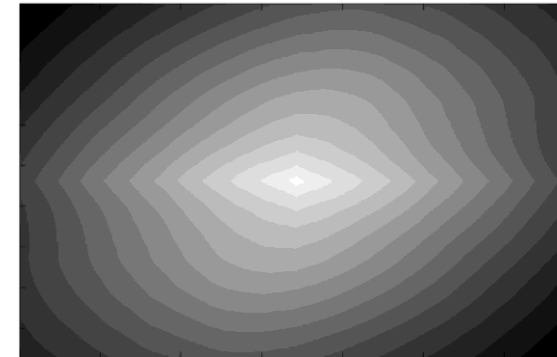
B



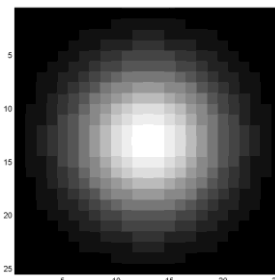
F



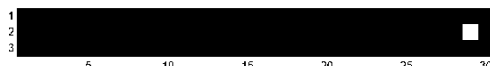
I



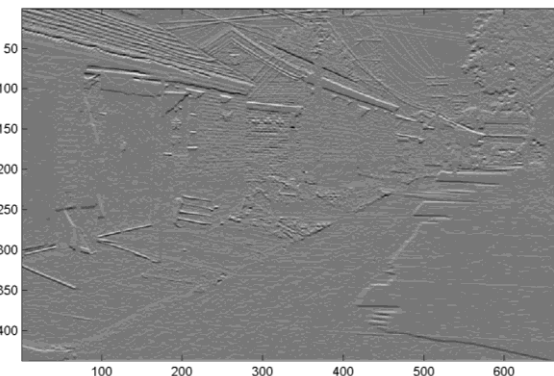
C



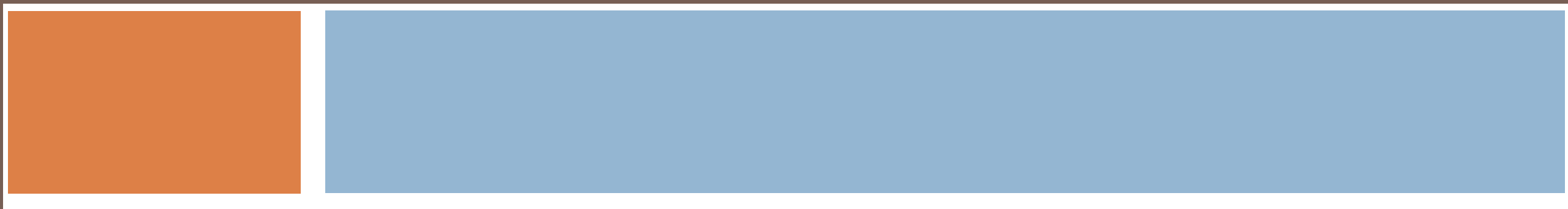
E



G



# ADAPTIVE FILTERING



# Jitter Filter

**Exercise 5.14.** The “jitter” filter is a (quite exotic) example for a *nonhomogeneous filter*. For each image position, it selects a space-variant filter kernel (of size  $2r + 1$ ) containing a single, randomly placed impulse (1); for example,

$$H_{u,v} = \begin{bmatrix} 0 & 0 & 0 & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{0} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (5.49)$$

for  $r = 2$ . The position of the 1-value in the kernel  $H_{u,v}$  is uniformly distributed in the range  $i, j \in [-r, r]$ ; thus the filter effectively picks a random pixel value from the surrounding  $(2r + 1) \times (2r + 1)$  neighborhood. Implement this filter for  $r = 3, 5, 10$ , as shown in [Fig. 5.24](#). Is this filter linear or nonlinear? Develop another version using a Gaussian random distribution.



# Jitter Filter - example



Original



$r = 3$



$r = 5$



$r = 10$

# Adaptive Filtering

---

- The convolution is a *non-adaptive* filtering in the sense that the convolution mask is space invariant.
- *Adaptive* filtering refers to image operations that adapt their performance based on the input signal.
- Example for adaptive-filtering: **The Bilateral Filter**



# Objective of bilateral filtering

---

Smooth texture

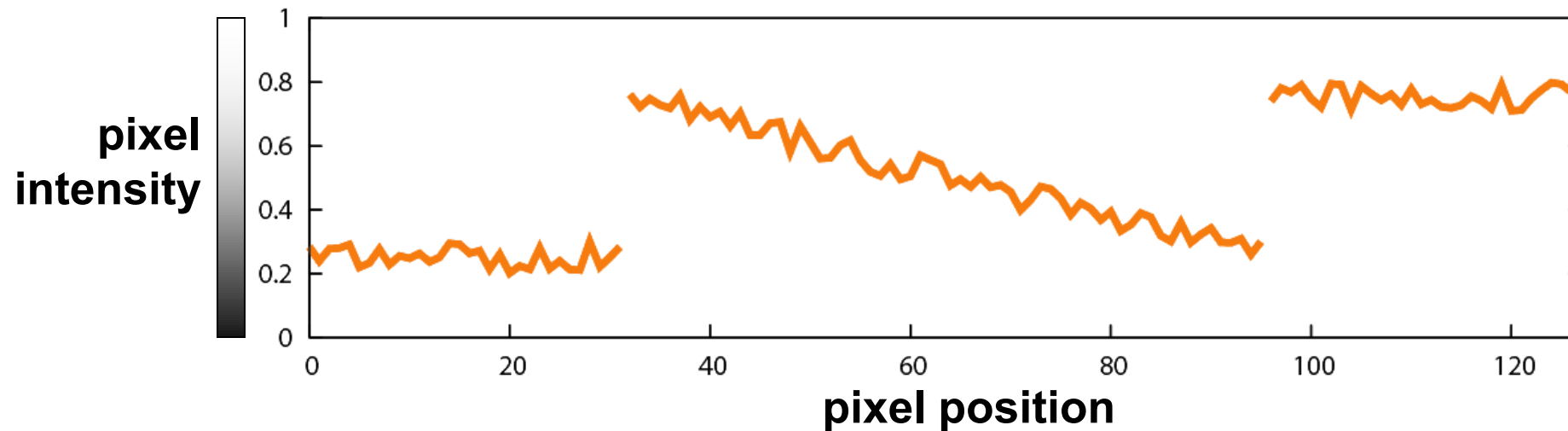
Preserve edges

# Illustration a 1D Image

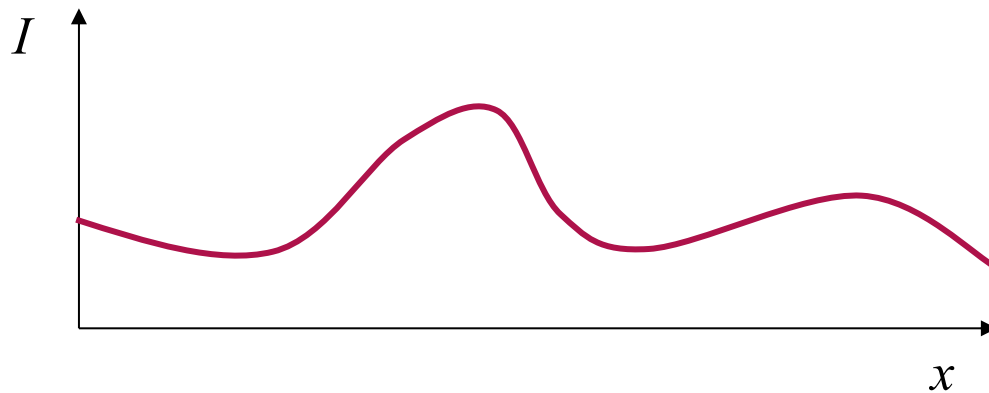
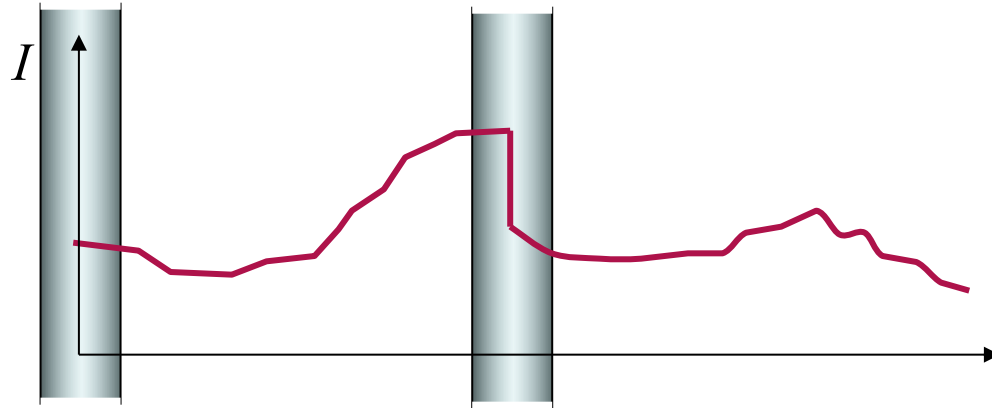
- 1D image = line of pixels



- Better visualized as a plot



# Gaussian Filter

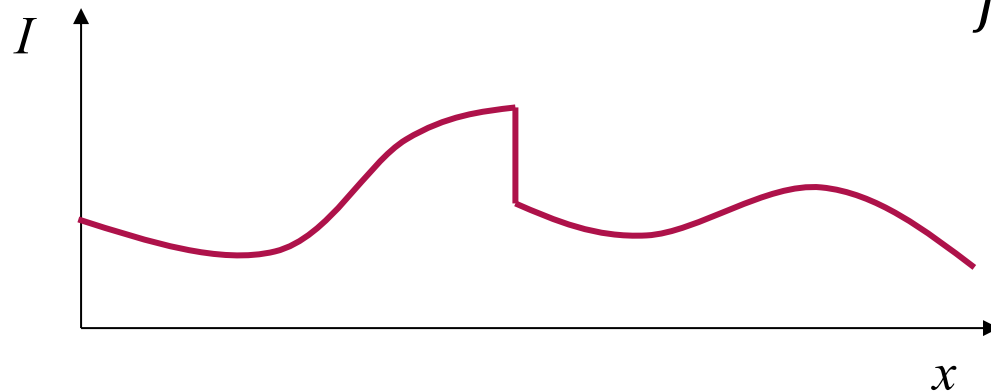
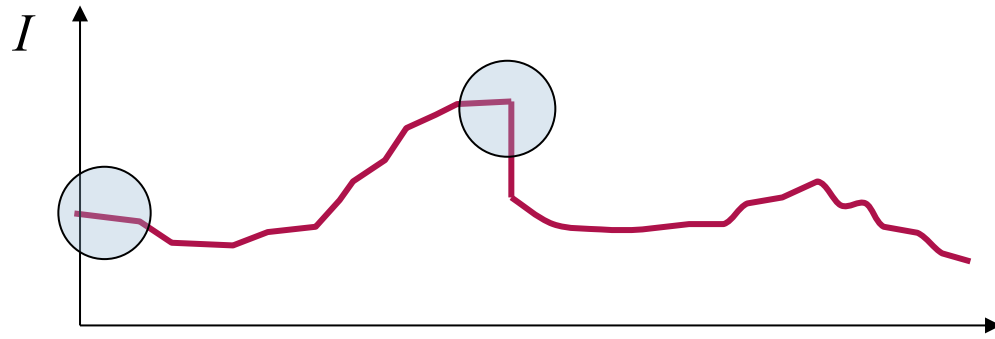


Smooth edges

$$\hat{f}_p = \sum_{j \in \Omega} W_s(p-j) f_j$$

$$W_s(t) = e^{-\frac{t^2}{2\sigma_s^2}}$$

# Bilateral Filter



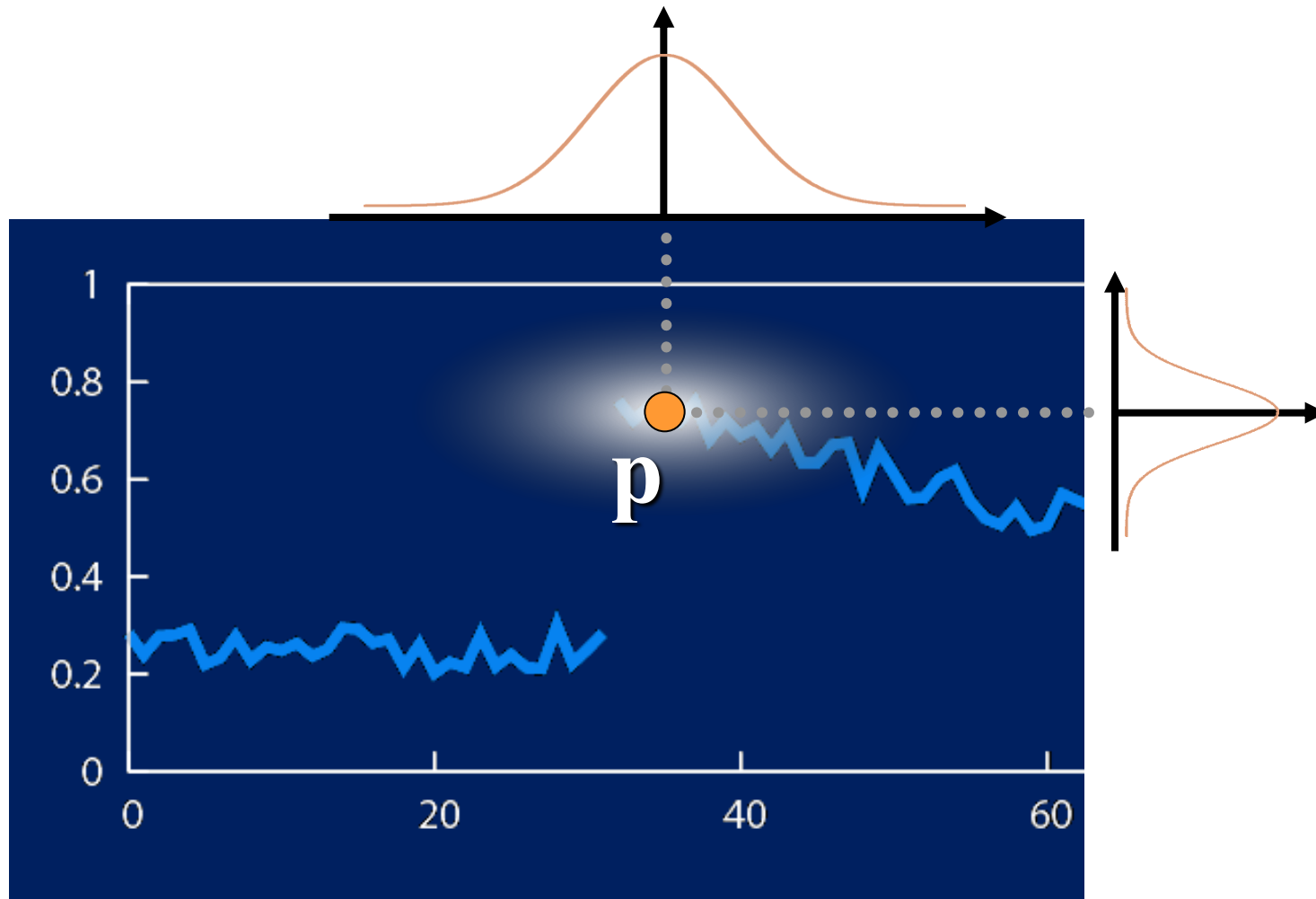
$$\hat{f}_p = \frac{\sum_{j \in \Omega} W_s(p - j) W_r(f_p - f_j) f_j}{\sum_{j \in \Omega} W_s(p - j) W_r(f_p - f_j)}$$

$$W(t) = e^{-\frac{t^2}{2\sigma_p^2}}$$

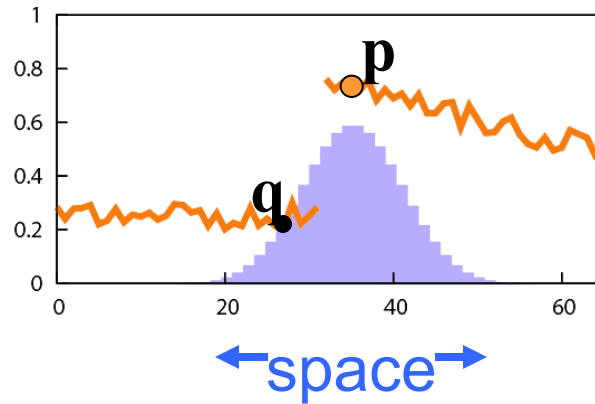
Preserves discontinuities

# Higher-dimensional Space

- “Product of two Gaussians” = higher dim. Gaussian



# Definition



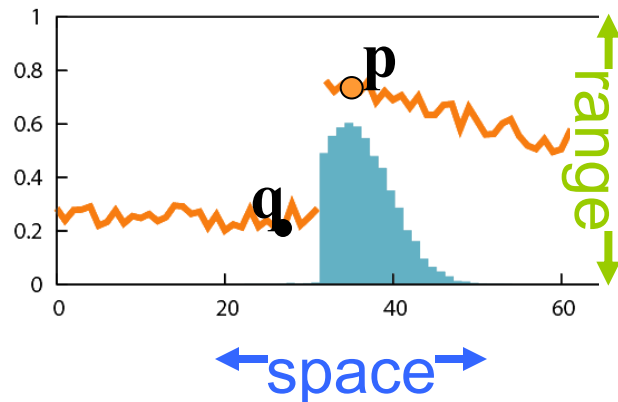
## Gaussian blur

$$I_{\mathbf{p}}^b = \sum_{\mathbf{q} \in \mathcal{S}} \underbrace{G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)}_{\text{space}} I_{\mathbf{q}}$$

- only spatial distance, intensity ignored

## Bilateral filter

[Aurich 95, Smith 97, Tomasi 98]



$$I_{\mathbf{p}}^{\text{bf}} = \underbrace{\frac{1}{W_{\mathbf{p}}^{\text{bf}}}}_{\text{normalization}} \sum_{\mathbf{q} \in \mathcal{S}} \underbrace{G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)}_{\text{space}} \underbrace{G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)}_{\text{range}} I_{\mathbf{q}}$$

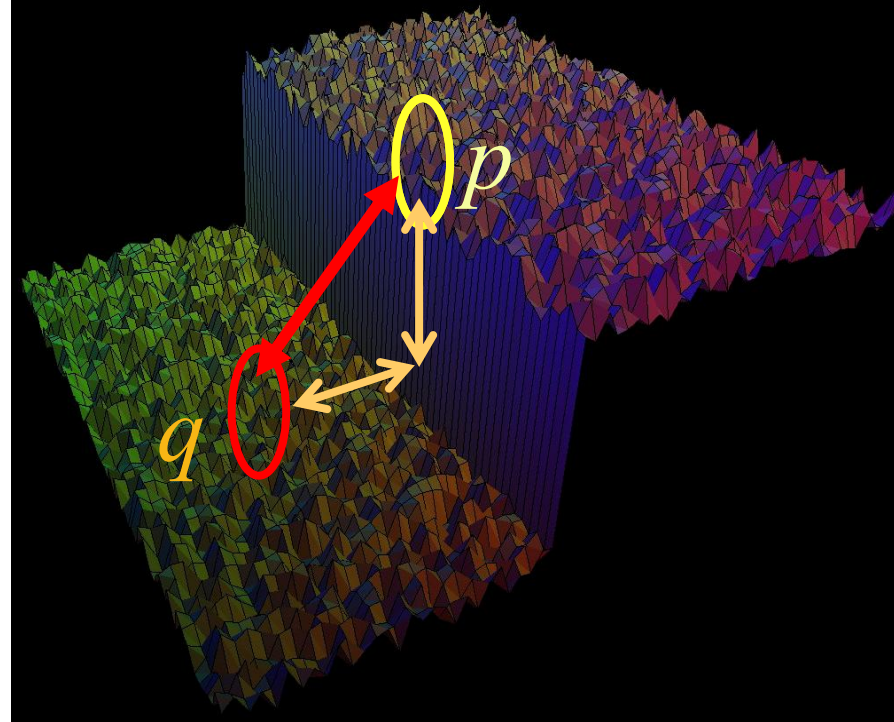
- spatial and range distances
- weights sum to 1

# Normalizing constant

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I_{\mathbf{p}} - I_{\mathbf{q}}\|) I_{\mathbf{q}}$$
$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|) G_{\sigma_r}(\|I_{\mathbf{p}} - I_{\mathbf{q}}\|)$$

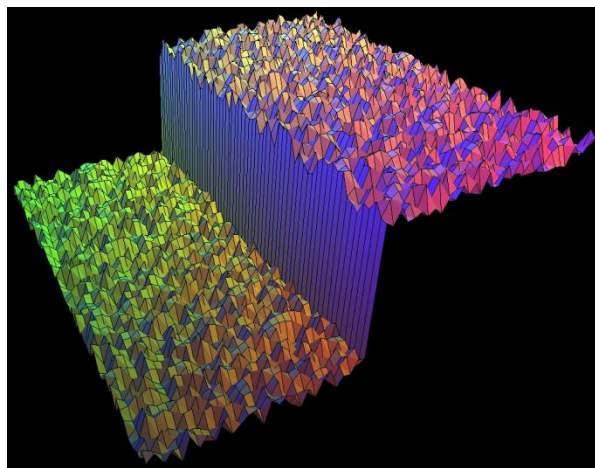
## Typical bilateral weighting functions:

$$W_s(p - q) = e^{-\left(\frac{p-q}{2\sigma_s}\right)^2}$$
$$W_r(f_p - f_q) = e^{-\left(\frac{f_p - f_q}{2\sigma_r}\right)^2}$$

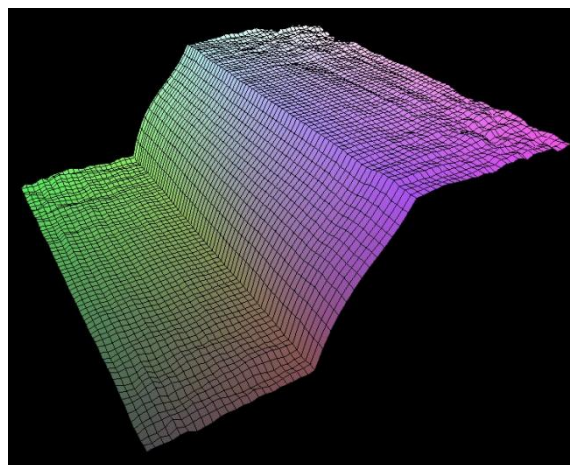
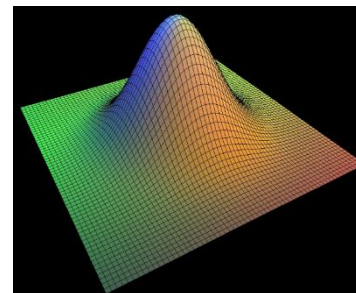




# Gaussian Filtering:

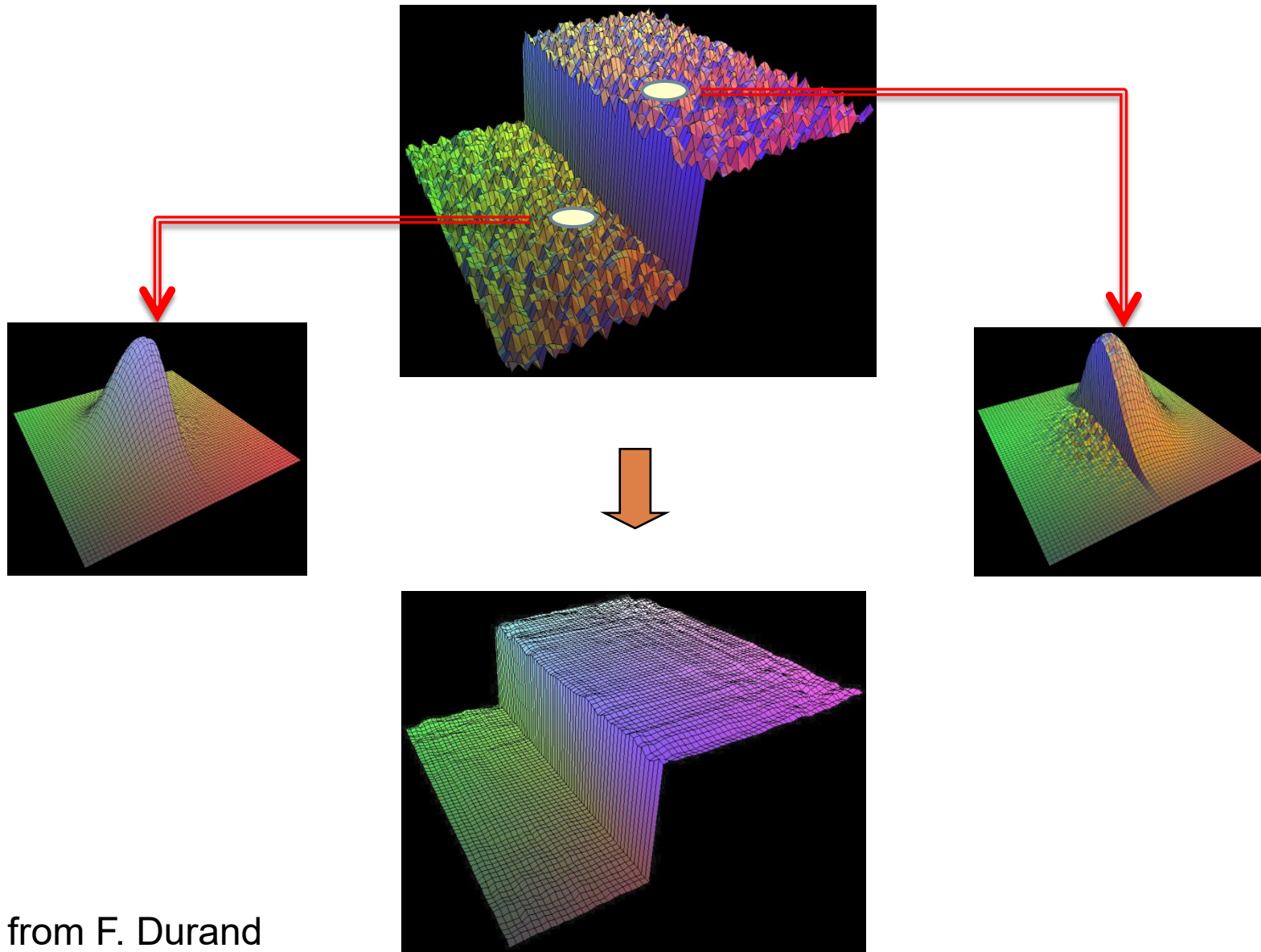


\*



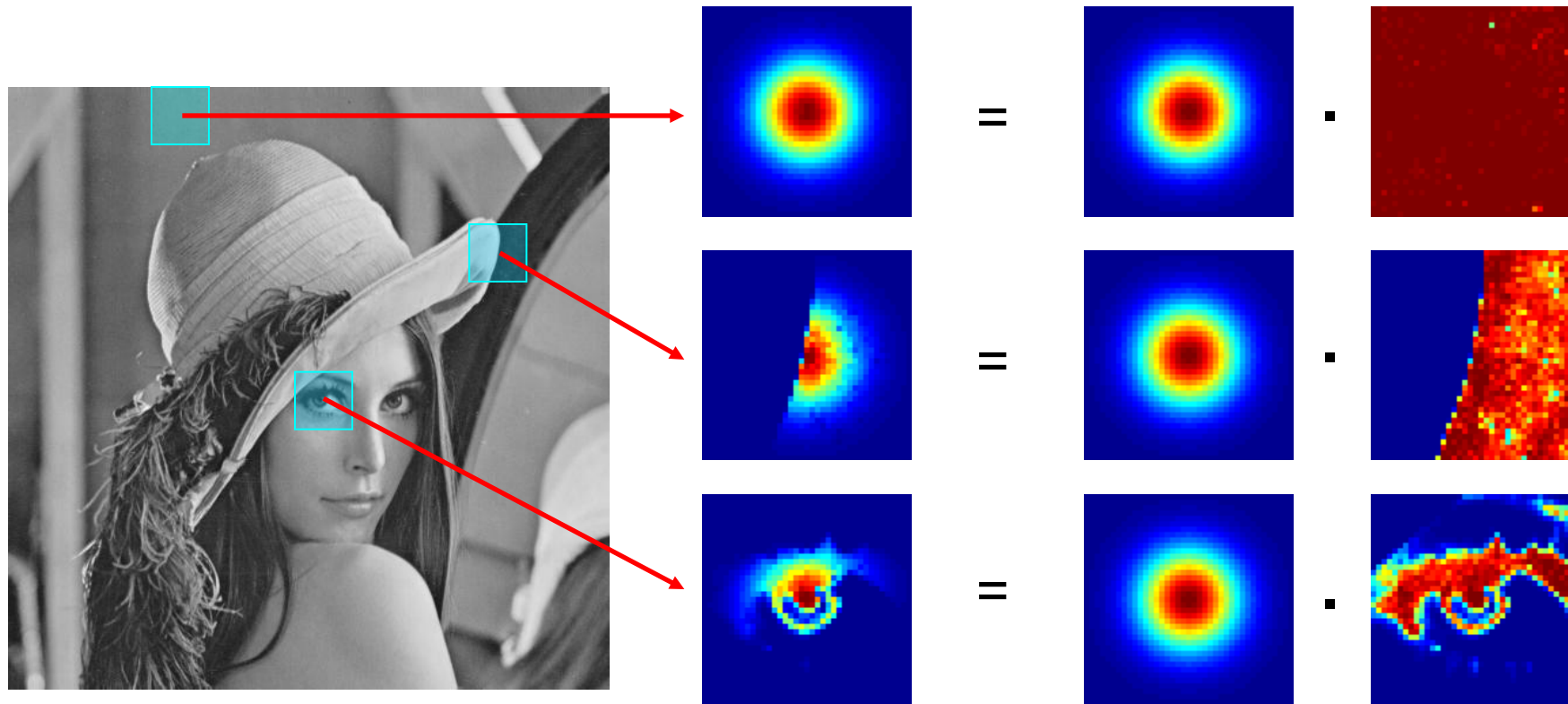
Slide from F. Durand

# Bilateral Filtering:



Slide from F. Durand

# Bilateral weights:



from P. Milinfar.

# Bilateral Filter is Expensive

- ❑ Brute-force computation is slow (several minutes)
  - ▣ Two nested for loops:  
*for each pixel, look at all pixels*
  - ▣ Non-linear, depends on image content  
⇒ no FFT, no pre-computation...
- ❑ Fast approximations exist [Durand 02, Weiss 06]
  - ▣ Significant **loss of accuracy**
  - ▣ **No formal understanding** of accuracy versus speed

# Gaussian Smoothing



# Bilateral (edge-preserving) Smoothing





Noisy  
Image





# Gaussian Smoothing





# Bilateral Smoothing



# Ref.

- Digital Image Processing, 4<sup>th</sup> ed, Gonzalez & Woods
- **Read this on seriously**
  - **Chapter 5 of Digital Image Processing using Java**
    - **By - Wilhelm Burger • Mark J. Burge**
- [www.cs.su.ac.th/~kanawong/courses/517483/ppt/SpatialFiltering.ppt](http://www.cs.su.ac.th/~kanawong/courses/517483/ppt/SpatialFiltering.ppt)
- [www.cse.unr.edu/~bebis/CS474/Lectures/SpatialFiltering.ppt](http://www.cse.unr.edu/~bebis/CS474/Lectures/SpatialFiltering.ppt)
- [www.acfr.usyd.edu.au/courses/amme4710/Lectures/AMME4710-Chap3-SpatialFiltering.pdf](http://www.acfr.usyd.edu.au/courses/amme4710/Lectures/AMME4710-Chap3-SpatialFiltering.pdf)
- [www.csie.nuk.edu.tw/~tkyin/2016Spring/ImageProcessing2016Spring/Slides/3/chap3.ppt](http://www.csie.nuk.edu.tw/~tkyin/2016Spring/ImageProcessing2016Spring/Slides/3/chap3.ppt)