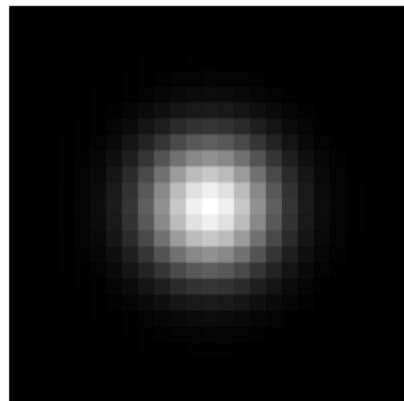
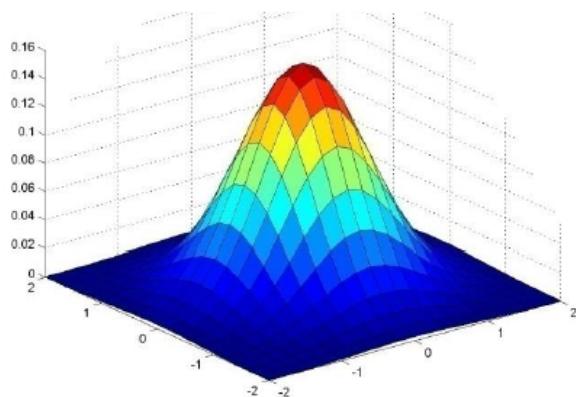


# Image Enhancement and Frequency domain operations

# Gaussian filters

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

$5 \times 5, \sigma = 1$

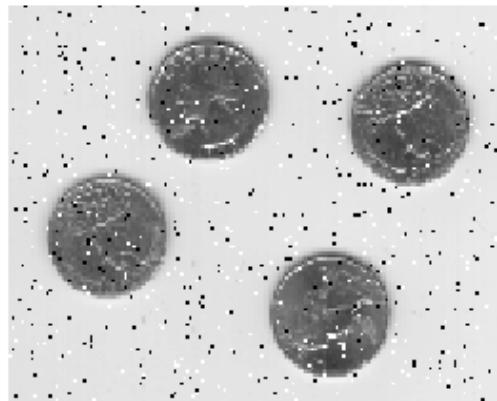
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Like the box filter, the Gaussian is a smoothing filter.

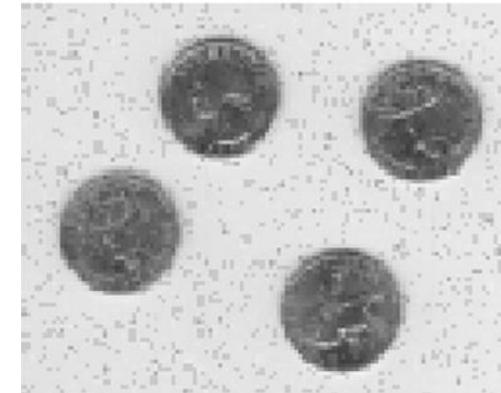
# Comparison with a Gaussian



(a)



(b)



(c)

(a) original image, (b) image+ 'salt and pepper' noise (c) After Gaussian filtering

# Effect of $\sigma$ on Gaussian filters



$\sigma=2$



$\sigma=4$



$\sigma=8$

# Gaussian filtering

## Resolution Sequence

Original Image

$$\sigma_0 = 0$$



# Gaussian filtering

Resolution  
Sequence

Gaussian LPF

$$\sigma_1 = 1$$



# Gaussian filtering

Resolution  
Sequence

Gaussian LPF

$$\sigma_2 = 2$$



# Gaussian filtering

Resolution  
Sequence

Gaussian LPF

$$\sigma_3 = 4$$



# Gaussian filtering

Resolution  
Sequence

Gaussian LPF

$$\sigma_4 = 8$$



# Gaussian filtering

Resolution  
Sequence

Gaussian LPF  
 $\sigma_5 = 16$

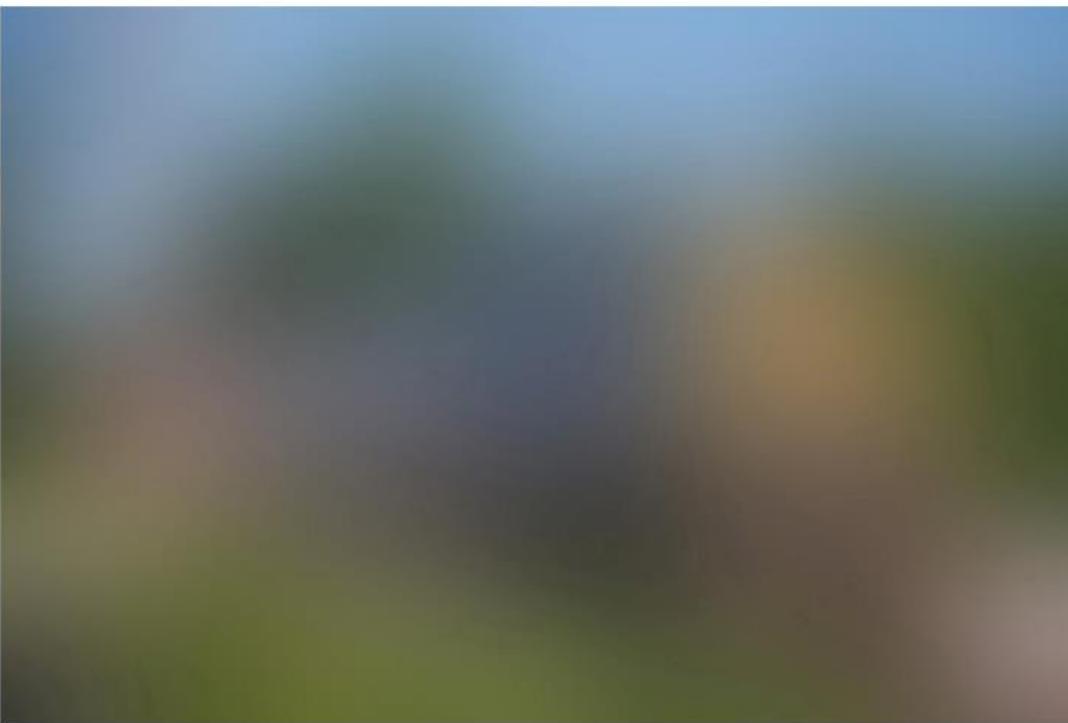


# Gaussian filtering

Resolution  
Sequence

Gaussian LPF

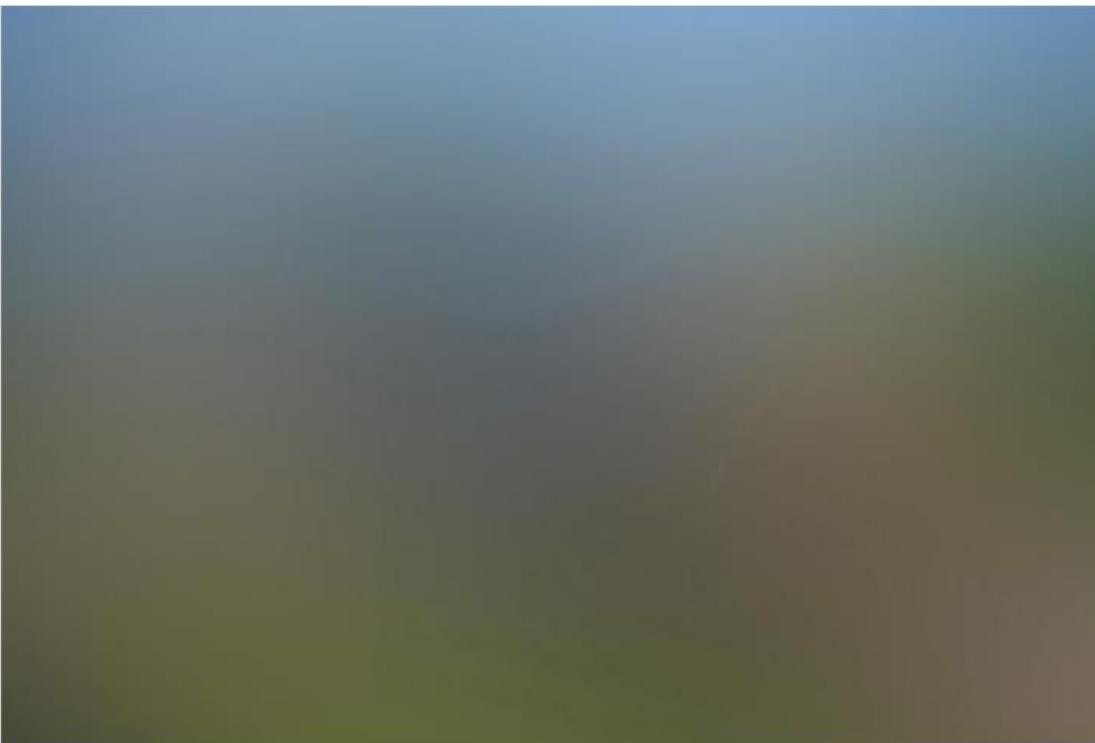
$$\sigma_6 = 32$$



# Gaussian filtering

Resolution  
Sequence

Gaussian LPF  
 $\sigma_7 = 64$



# Gaussian filtering

Resolution  
Sequence

Gaussian LPF  
 $\sigma_8 = 128$



# Gaussian filtering

Resolution  
Sequence

Gaussian LPF

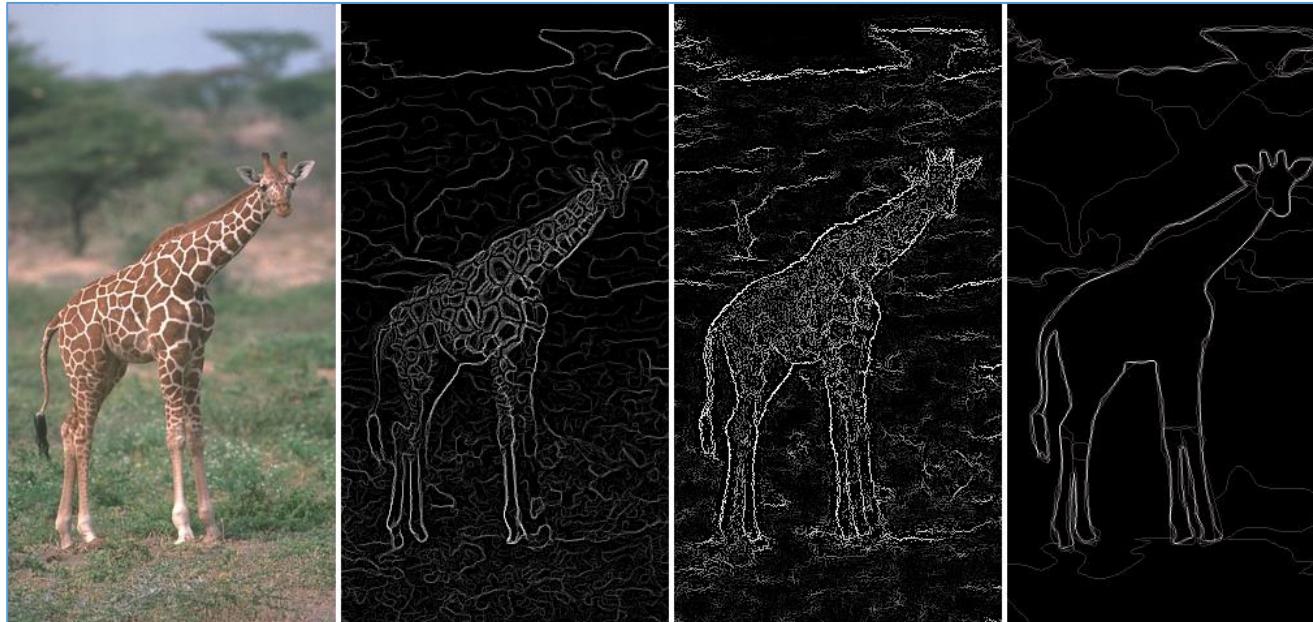
$\sigma \rightarrow \infty$



Using a sequence of Gaussian filters the image converges to a constant plan with a value that is equal to the mean Intensity value in the image

# Using filters for edge detection and image enhancement

- So far we have been focusing on smoothing filters.
- Another class of filters are the edge detection filters and they allow use to detect edges and other regions where high magnitude changes of intensity occurs.



# Simple edge detectors

**Roberts**

<b>0</b>	-1
1	<b>0</b>

**Prewitt**

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

**Sobel**

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

X derivative

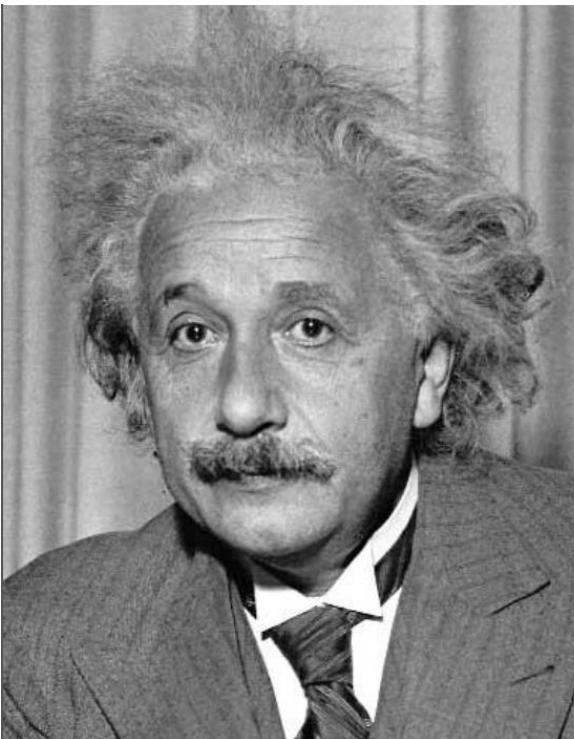
-1	0
0	1

Y derivative

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

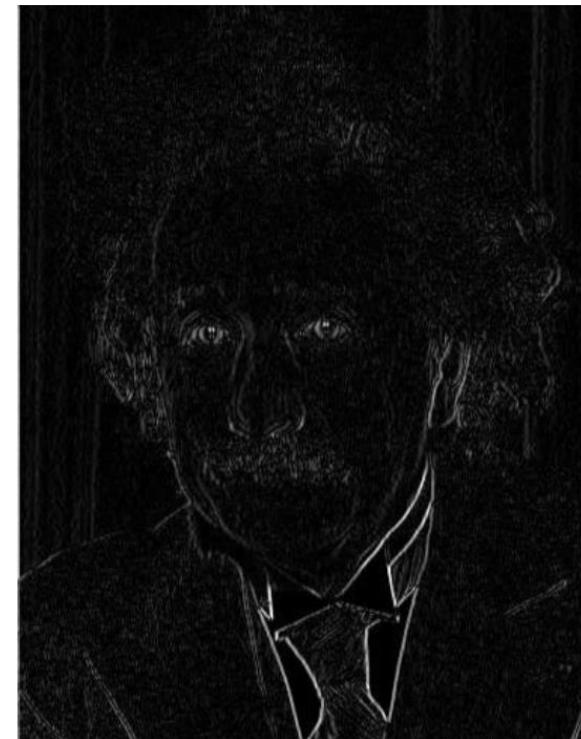
1	2	1
0	0	0
-1	-2	-1

# Simple edge detectors

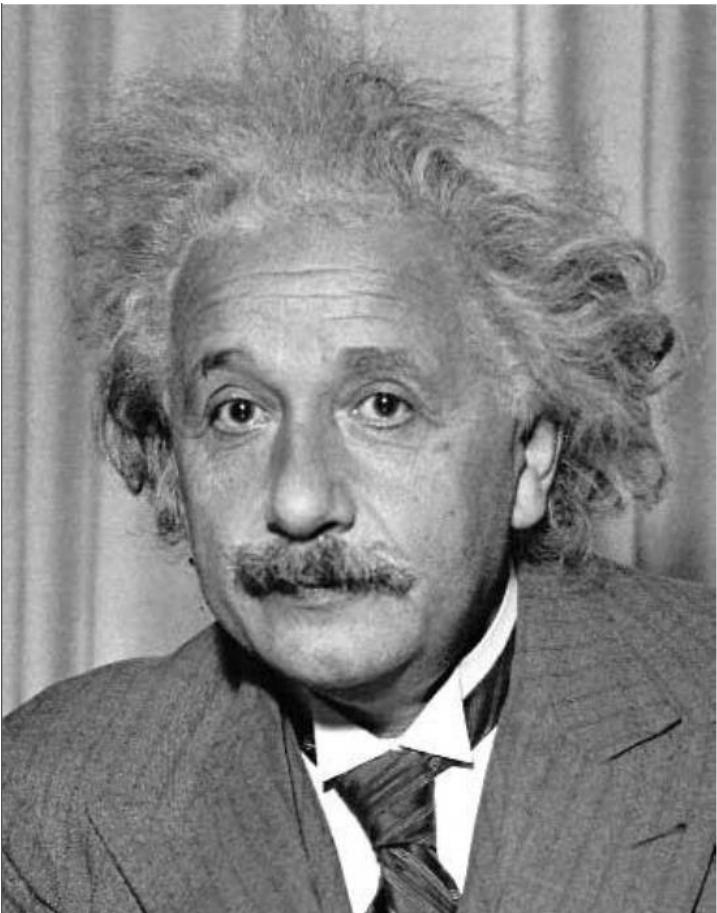


1	0	-1
2	0	-2
1	0	-1

Sobel



# Simple edge detectors



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

Sobel



# Smoothing prior to edge detection

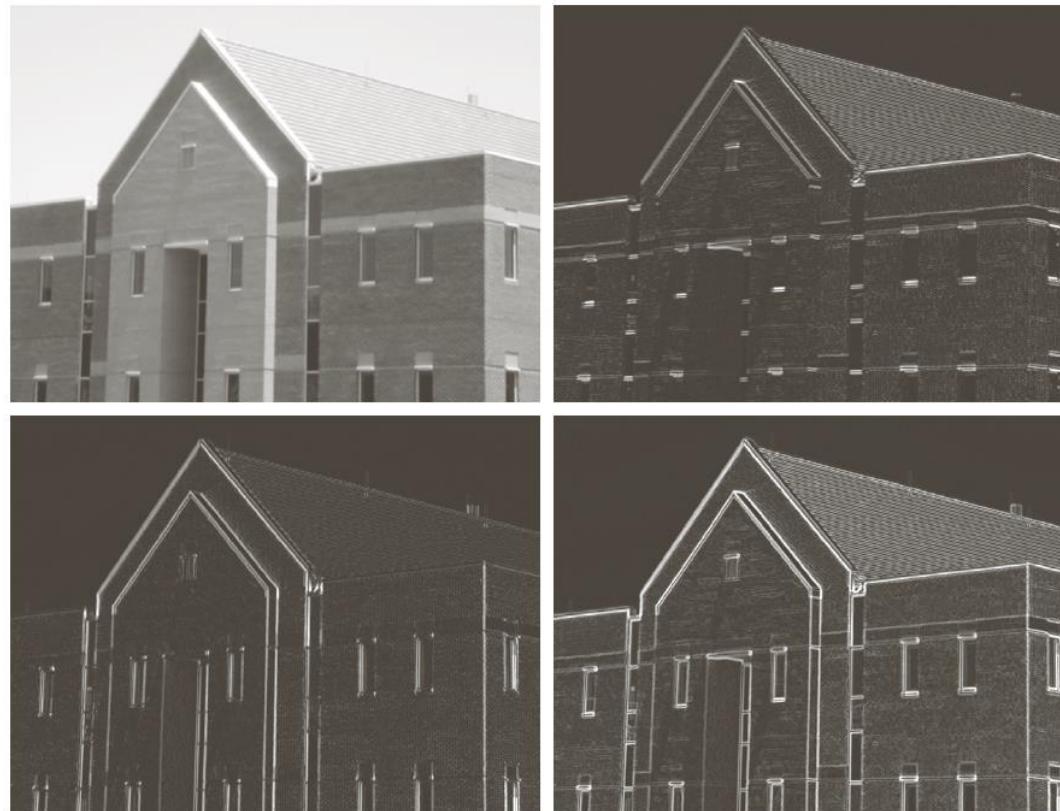
- Smoothing an image can improve the quality of edge detection.



a b  
c d

**FIGURE 10.16**  
(a) Original image of size  $834 \times 1114$  pixels, with intensity values scaled to the range  $[0, 1]$ .  
(b)  $|g_x|$ , the component of the gradient in the  $x$ -direction, obtained using the Sobel mask in Fig. 10.14(f) to filter the image.  
(c)  $|g_y|$ , obtained using the mask in Fig. 10.14(g).  
(d) The gradient image,  $|g_x| + |g_y|$ .

# Smoothing prior to edge detection



a b  
c d

**FIGURE 10.18**  
Same sequence as  
in Fig. 10.16, but  
with the original  
image smoothed  
using a  $5 \times 5$   
averaging filter  
prior to edge  
detection.

# Derivative operators: First order derivatives

$$I = \begin{bmatrix} Kul & Ku & Kur \\ Kl & K & Kr \\ Kdl & Kd & Klr \end{bmatrix}$$

- Foreword difference method

$$Gx(I)_{2,2} = \frac{\partial f}{\partial x} \Big|_{x=2} = Kr - K \rightarrow \frac{\partial f}{\partial x} \Big|_x = f(x+1, y) - f(x, y)$$

- Backward difference method

$$Gx(I)_{2,2} = \frac{\partial f}{\partial x} \Big|_{x=2} = K - Kl \rightarrow \frac{\partial f}{\partial x} \Big|_x = f(x, y) - f(x - 1, y)$$

# First order derivatives

$$I = \begin{bmatrix} Kul & Ku & Kur \\ Kl & K & Kr \\ Kdl & Kd & Klr \end{bmatrix}$$

- Centered difference method

$$Gx(I)_{2,2} = \left. \frac{\partial f}{\partial x} \right|_{x=2} = \frac{Kr - Kl}{2} \rightarrow \left. \frac{\partial f}{\partial x} \right|_x = \frac{f(x+1,y) - f(x-1,y)}{2}$$

- At boundaries use forward difference or backward difference.
- Central difference is the preferred method for computing derivatives since it generates less error

# Second order derivatives

- Using centered difference in second order derivate

$$\dot{f}|_x = \frac{f(x+1, y) - f(x-1, y)}{2}$$

$$\ddot{f}|_x = \frac{\dot{f}|_{x+1} - \dot{f}|_{x-1}}{2}$$

$$\ddot{f}|_x = \frac{f(x+1, y) - f(x, y)}{2} - \frac{f(x, y) - f(x-1, y)}{2}$$

$$\ddot{f}|_x = \frac{f(x+1, y) - 2f(x, y) + f(x-1, y)}{2}$$

- What is  $\ddot{f}|_y$  ?

# Laplacian filter

- Laplacian operator

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

- Discrete Laplacian operator

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

# Laplacian filter

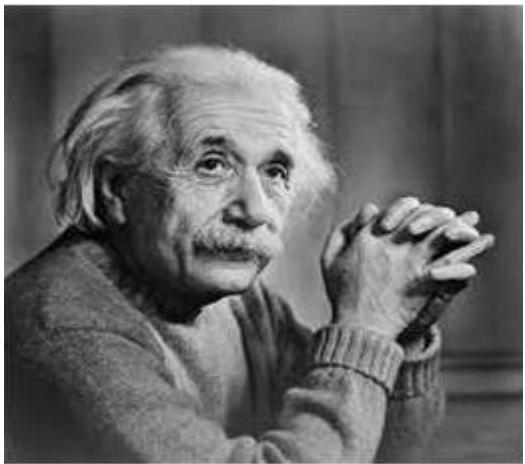
- Laplacian filter

<b>0</b>	<b>1</b>	<b>0</b>
<b>1</b>	<b>-4</b>	<b>1</b>
<b>0</b>	<b>1</b>	<b>0</b>

- Insensitive to diagonal changes.
- Solution

<b>1</b>	<b>1</b>	<b>1</b>
<b>1</b>	<b>-8</b>	<b>1</b>
<b>1</b>	<b>1</b>	<b>1</b>

# Laplacian filter



The Laplacian filter highlights discontinuities.

# Laplacian edge enhancement

- Take the original image and subtract the laplacian

$$I_{output}(x, y) = I_{in}(x, y) - \nabla^2 I_{in}(x, y)$$

↑  
Laplacian



Original Image



Laplacian “edges”



Sharpened Image

# Laplacian of Gaussian

- Smooth the image with a Gaussian filter before applying the Laplacian (Laplacian of Gaussian, LOG)
- Recall that smoothing improves edge detection



Original Image



LoG “edges”



Sharpened Image

# Image Sharpening

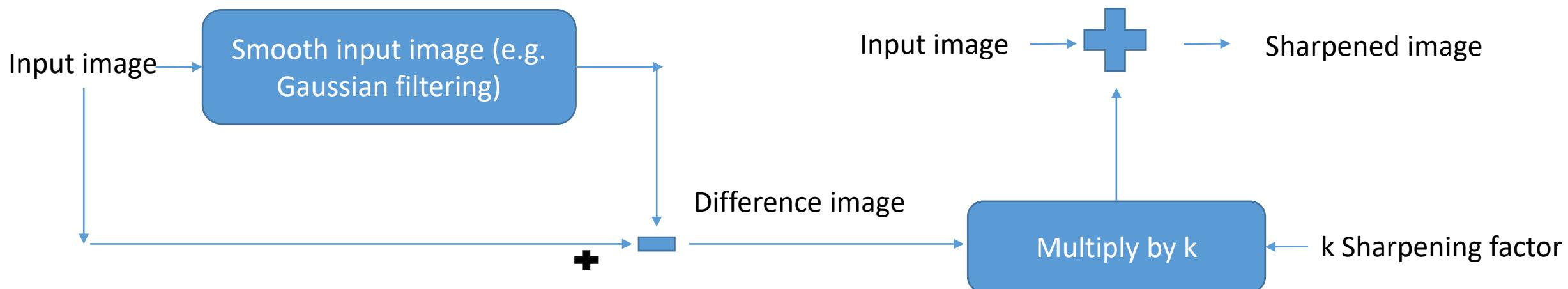
- Sharpening is achieved by enhancing high-frequency content of images (*i.e.* edges)
- In Unsharp masking, an image is sharpened by adding some of the difference between the original image and the smoothed image

$$g_{\text{sharp}} = f + k (f - h_{\text{smoothing}} * f)$$

# Unsharp mask filter

1. Generate a smoothed image  $I_{\text{smoothed}} (h_{\text{smoothing}} * f)$
2. Compute  $I_{\text{diff}} = I_{\text{original}} - I_{\text{smoothed}}$ .
3. Add the difference image to the original

$$I_{\text{enhanced}} = I_{\text{original}} + k (I_{\text{diff}})$$



# Unsharp mask filter



Original Image



Edge image



$k=0.3$



$k=0.5$



$k=0.7$



$k=2.0$

# Unsharp mask filter

Input image

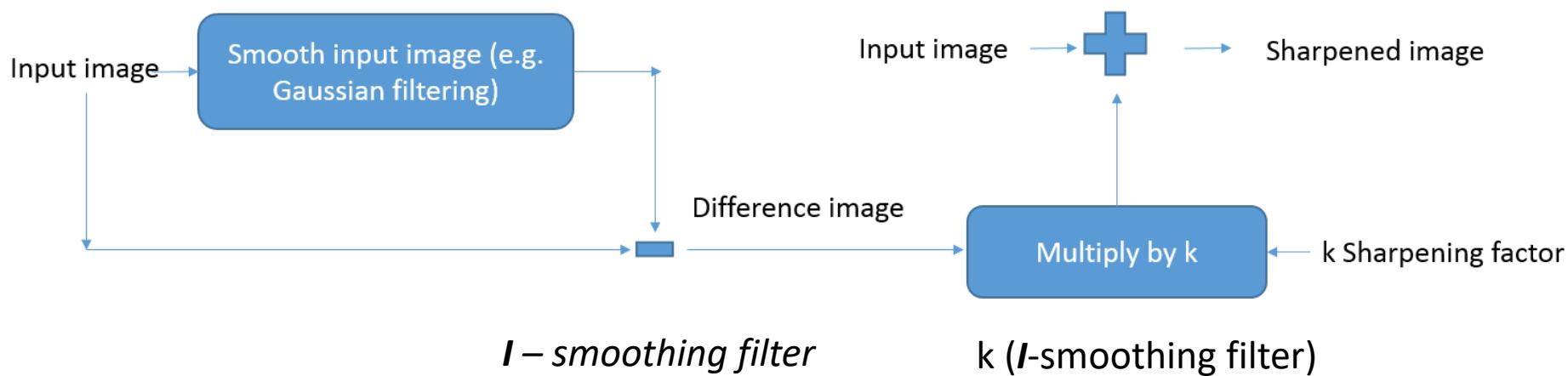


# Unsharp mask filter: faster implementation

- The identity filter returns the original image.

$$I = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$k (I - \text{smoothing filter}) + I$$



# Unsharp mask filter

- Unsharp mask filtering can be implemented using a single filtering operation with the filter  $k(I - \text{smoothing filter}) + I$

Where  $I = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$

# Today's Class

- Speeding up filter computation (filter separability)
- Fourier transform and Frequency domain filtering

# Exercise

- Is this a sharpening or smoothing filter?
- Why is (2) used as the center value of the first filter.

$$\begin{matrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{matrix} - \frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix}$$

Hint: Compare this with unsharp mask filtering

$$k(I - \text{smoothing filter}) + I$$

# Exercise



Original

$$\begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$$

-

$$\frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$



**Sharpening filter**

# Exercise

What is the effect of applying this filter to a sample image?

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

Answer: The filter replaces every intensity with the intensity two pixels to the right, shifting the image to the left by two pixels.

# Filter Separability

- A separable 2D filter kernel can be written as a multiplication of two 1D kernels.
- Separable filters reduce computational cost.
- $2K$  operations for 1D kernels,  $K^2$  for 2D kernels

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} [1 \quad 0 \quad -1]$$

- In separable filters kernel rows or columns are multiples of each other
- If the 2D filter kernel has a rank of 1 then it is separable
- The rank of a matrix is defined as the maximum number of linearly independent column vectors in the matrix OR the maximum number of linearly independent row vectors in the matrix.

# Filter Separability

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

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

# Filter Separability

2D convolution  
(center location only)

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix}$$

The filter factors  
into a product of 1D  
filters:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Perform convolution  
along rows:

$$\begin{bmatrix} 1 & 2 & 1 \end{bmatrix} * \begin{bmatrix} 2 & 3 & 3 \\ 3 & 5 & 5 \\ 4 & 4 & 6 \end{bmatrix} = \begin{bmatrix} 11 \\ 18 \\ 18 \end{bmatrix}$$

Followed by convolution  
along the remaining column:

$$\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 11 \\ 18 \\ 18 \end{bmatrix} = \begin{bmatrix} 65 \end{bmatrix}$$

# Filter Separability

- We would like to smooth a 3D image by a Gaussian kernel of size  $10 \times 10 \times 10$
- Smoothing using the 3D filter kernel requires 1000 operations per voxel
- A better approach is to smooth using three 1D Gaussian kernels of sizes  $10 \times 1 \times 1$ ,  $1 \times 10 \times 1$ , and  $1 \times 1 \times 10$
- Operations required becomes 30 operations **per voxel**.

Significant saving !!

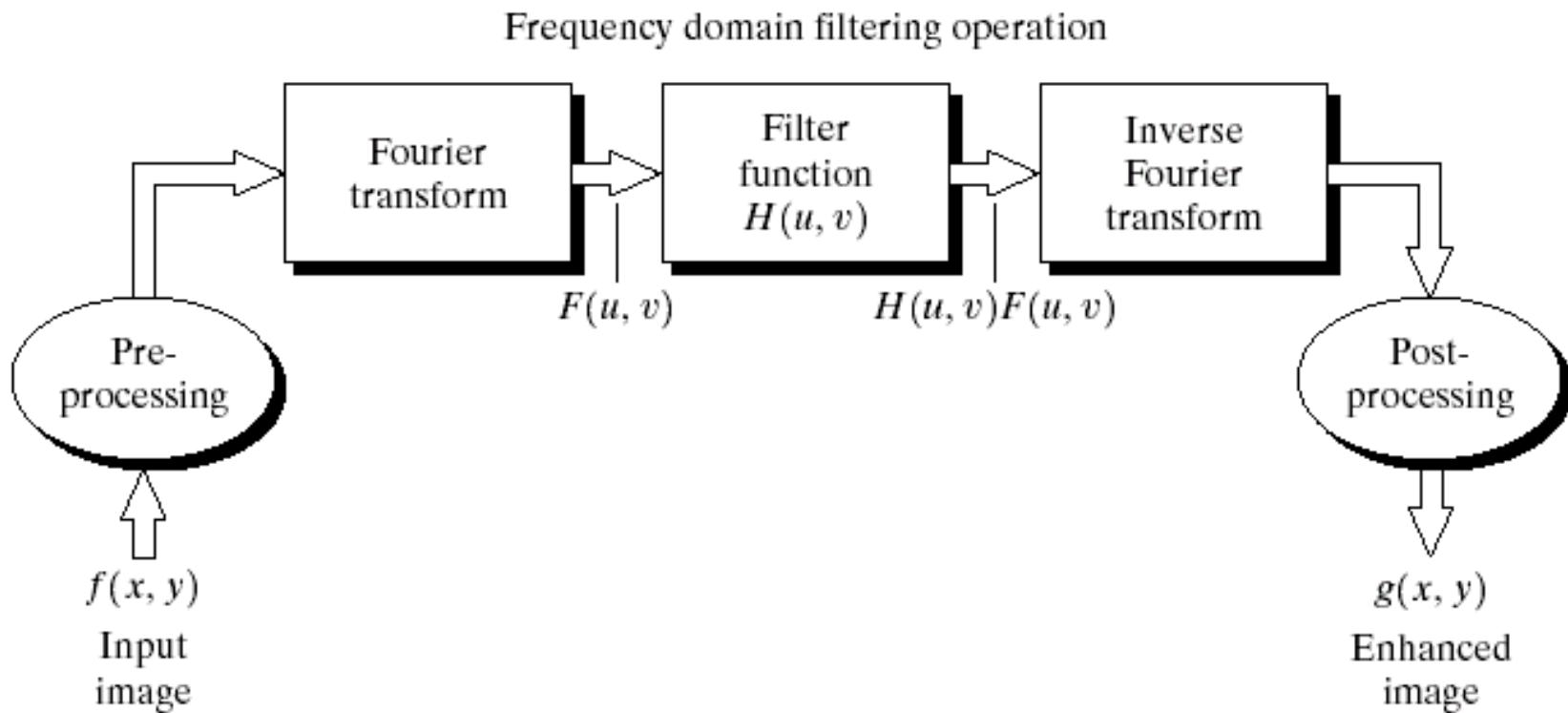
# Filter Separability

Assume that you have an image of size  $M \times N$  and 2D separable kernel of size  $k \times k$ .

Computational advantage = # of operations using the 2D kernel/# of operations using the 1D kernel

$$= \frac{MNk^2}{MNk+MNK} = \frac{MNk^2}{MN2k} = \frac{k^2}{2k} = \frac{k}{2}$$

# Frequency domain filtering



**FIGURE 4.5** Basic steps for filtering in the frequency domain.

# Using DFTs to represent signals

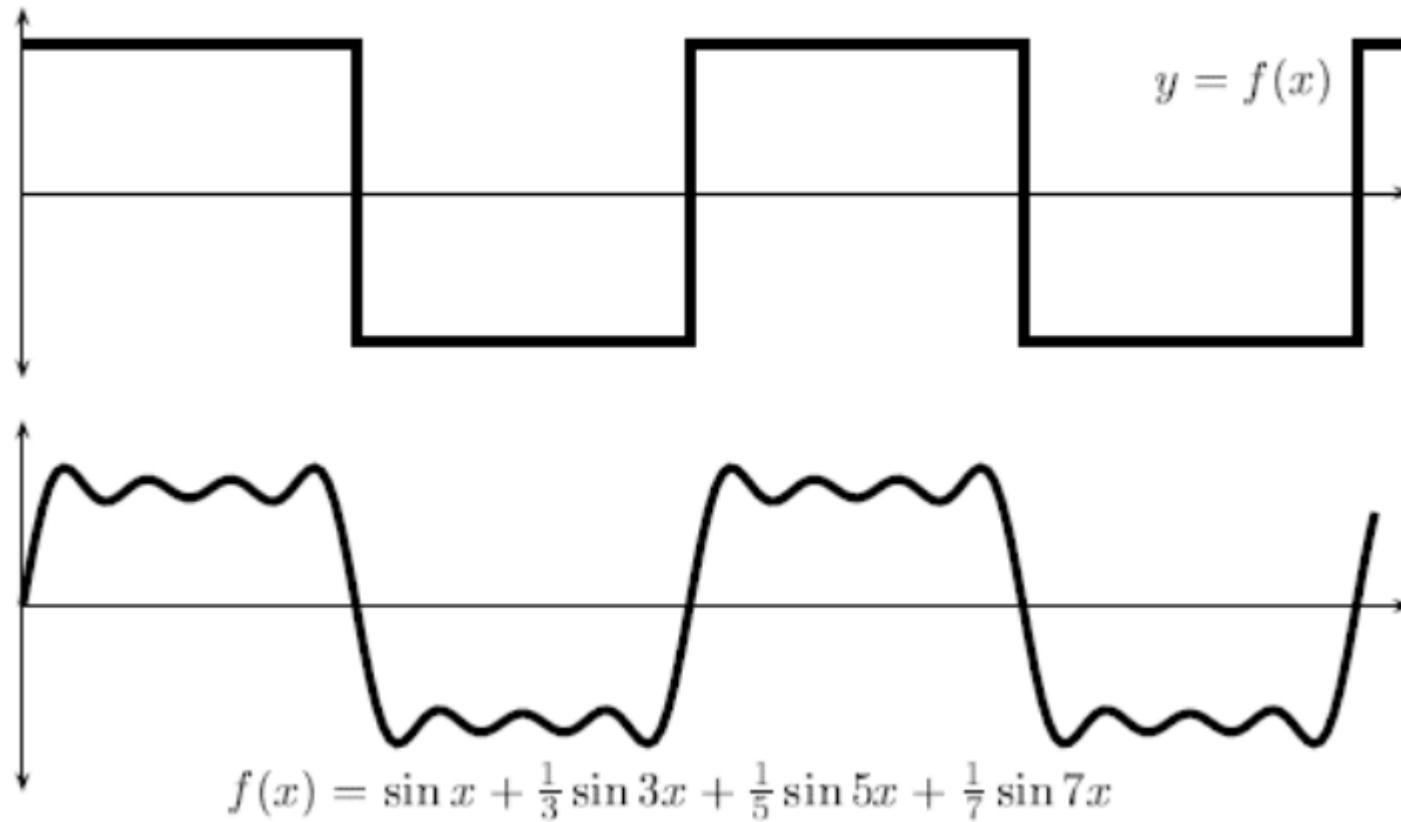
- Theorem (by Fourier, 1768-1830)

A periodic signal  $x(t)$  with period  $T$  can be written as an infinite sum of sinusoids, all of which have frequencies that are multiples of  $1/T$ .

- But most signals in practice are not infinite?

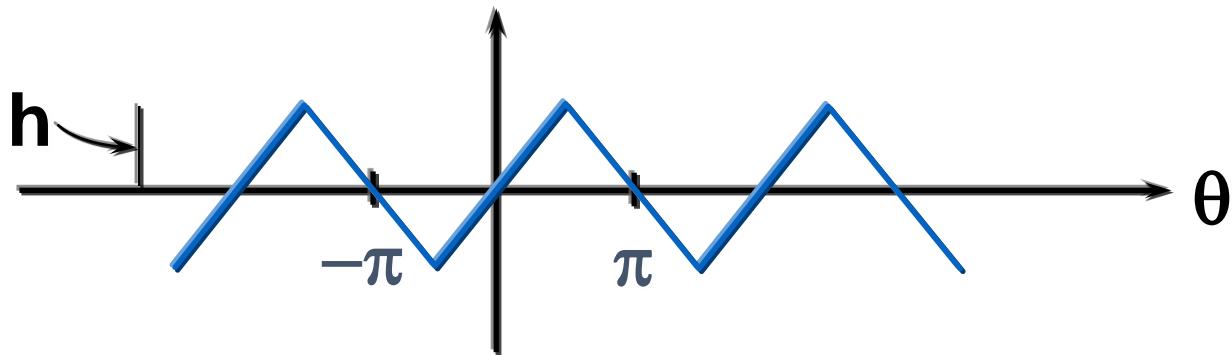
In this case, it's assumed they extend to infinity by replication

# Fourier Analysis



# Fourier Analysis

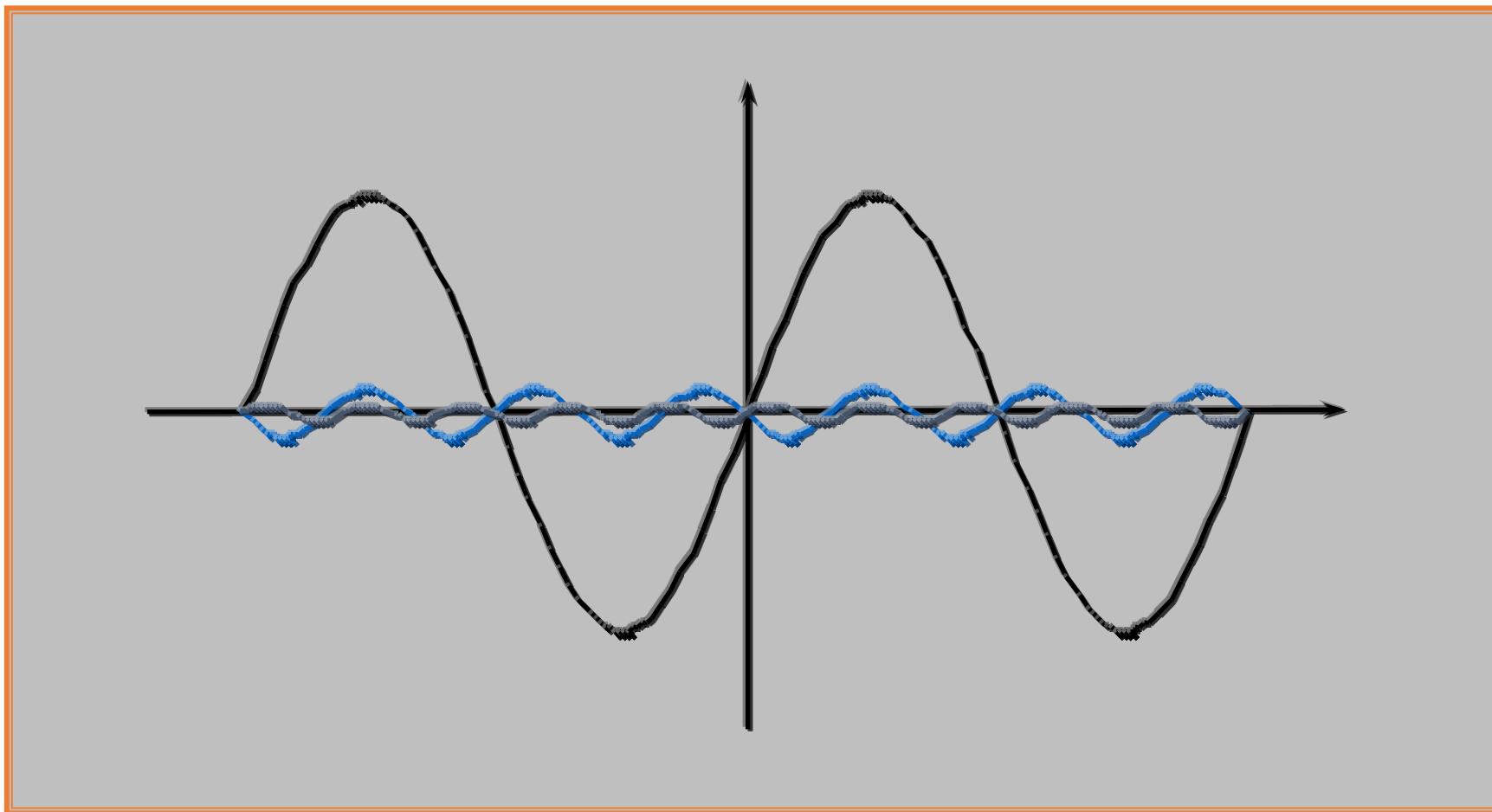
Triangular wave



$$f(q) = \left( \frac{8h}{p^2} \right) \left( \frac{\sin q}{1^2} - \frac{\sin 3q}{3^2} \right. \\ \left. \dots + \frac{\sin 5q}{5^2} - \frac{\sin 7q}{7^2} \dots \right)$$

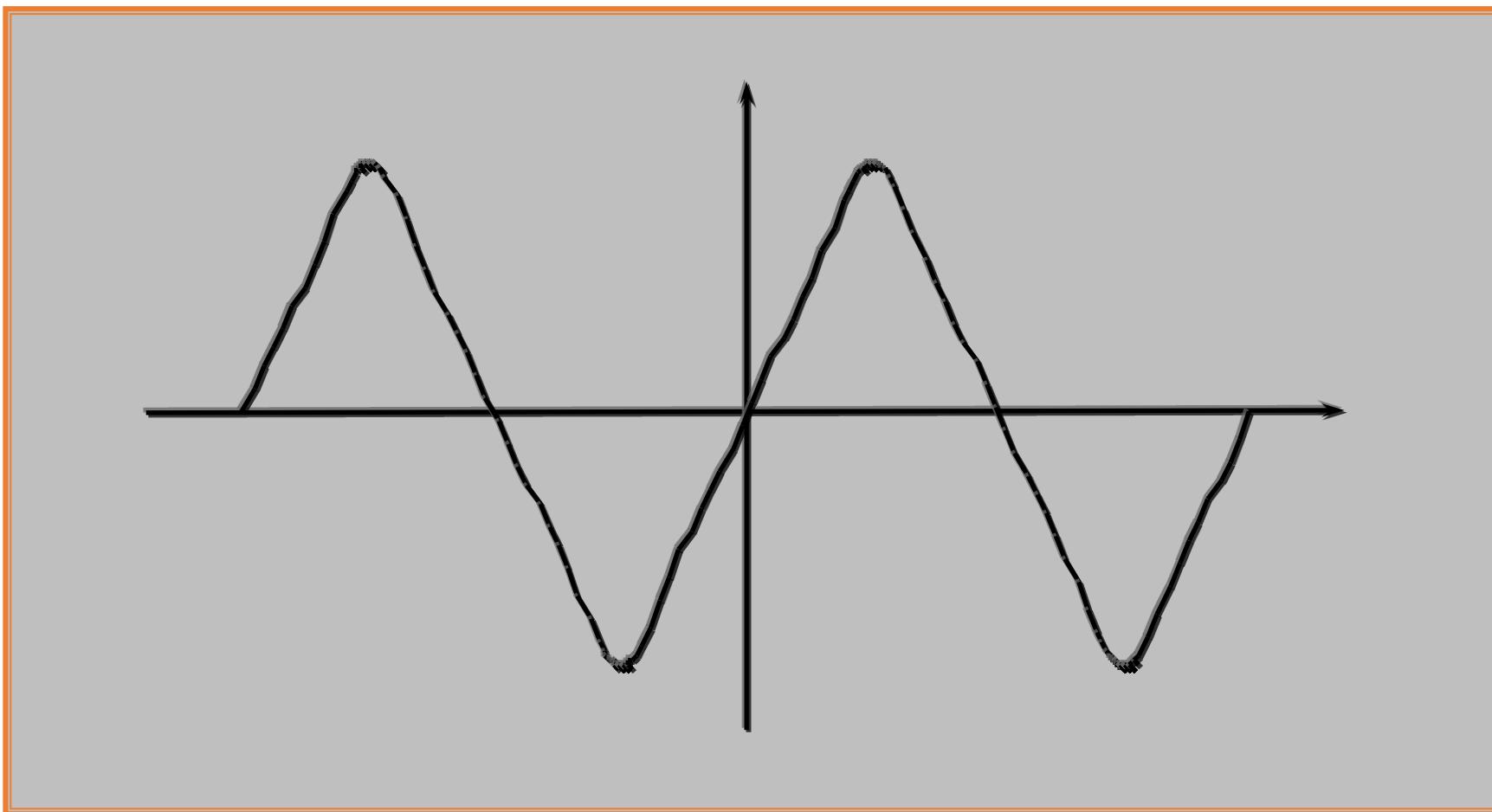
# Triangular wave

- First 3 series terms:



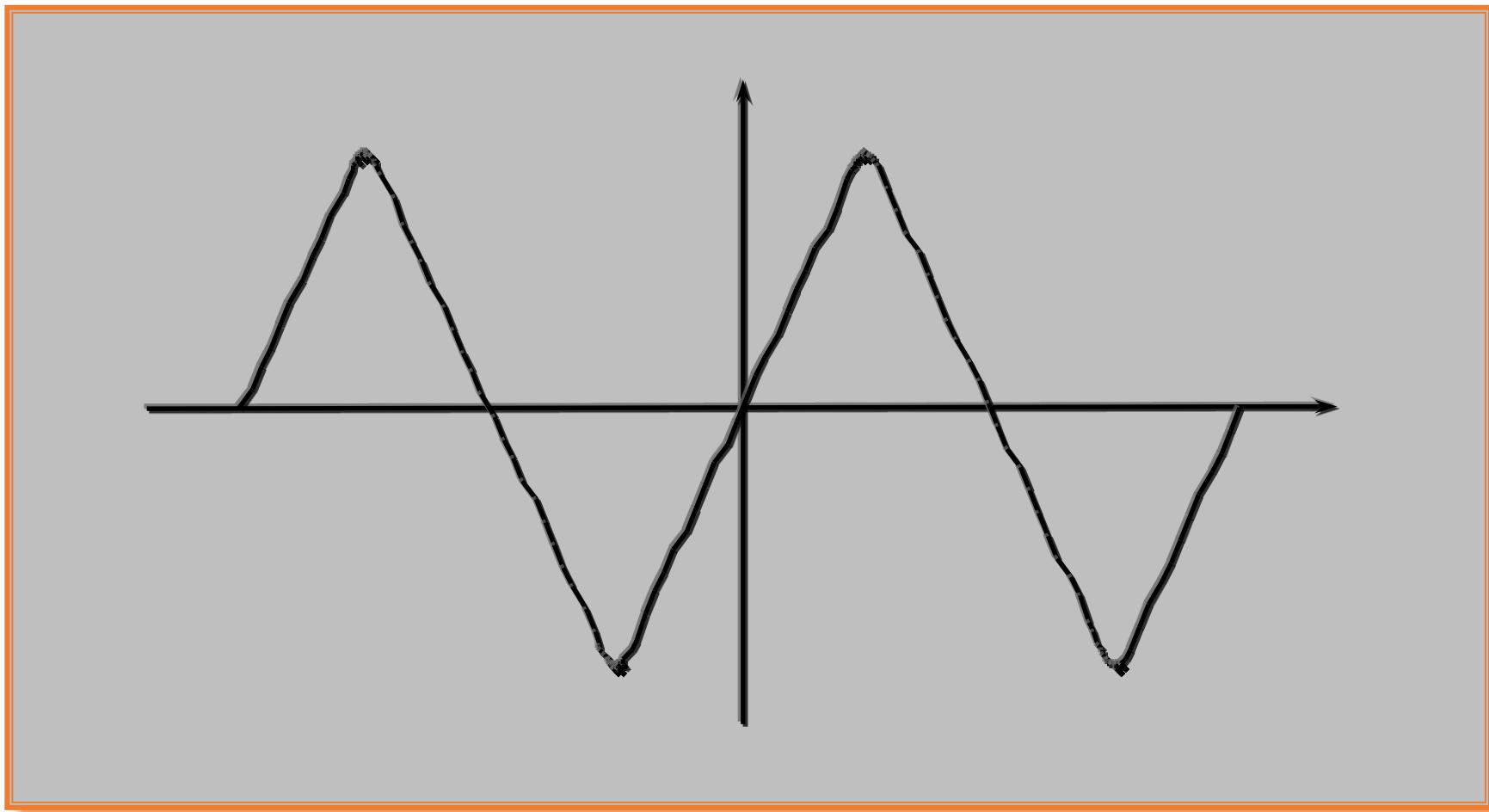
# Triangular wave

- Sum of first 3 series terms:



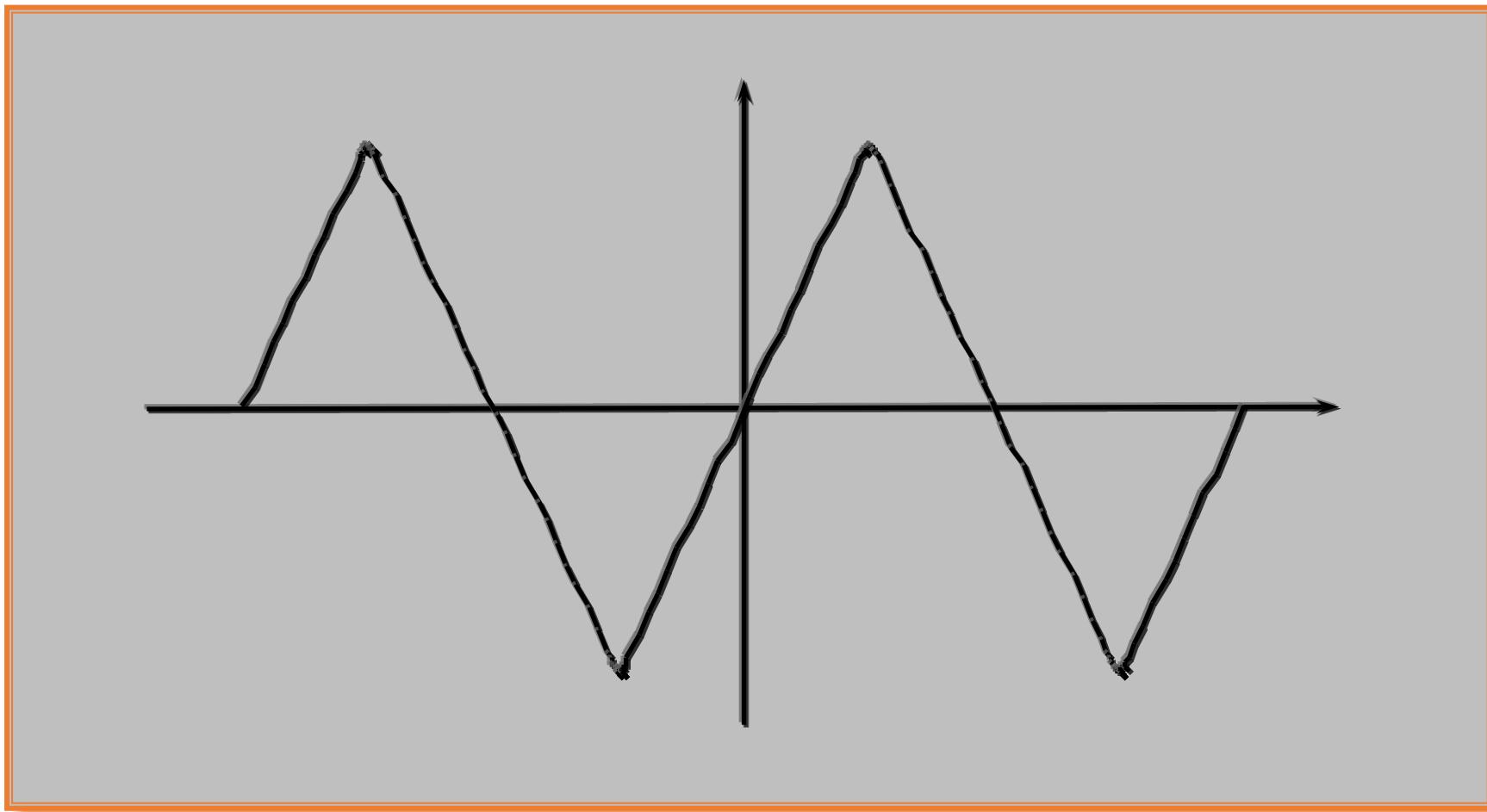
# Triangular wave

- Sum of first five terms:



# Triangular wave

- Sum of first 10 terms:



# Fourier coefficients

- A periodic signal  $f(x)$  with period  $T$  can be constructed exactly using

$$f(x) = \sum_{n=0}^{\infty} a_n \cos\left(\frac{2\pi n x}{T}\right) + \sum_{n=0}^{\infty} b_n \sin\left(\frac{2\pi n x}{T}\right)$$

- $a_n$  and  $b_n$  represent the contribution of the  $n$ th frequency to the signal.
- $a_n$  and  $b_n$  at lower values for  $n$  describe the low frequency content (i.e. broad structural details).
- $a_n$  and  $b_n$  at high  $n$  values describe fine details in the signal.

# Fourier coefficients

$$f(x) = \sum_{n=0}^{\infty} a_n \cos\left(\frac{2\pi n x}{T}\right) + \sum_{n=0}^{\infty} b_n \sin\left(\frac{2\pi n x}{T}\right)$$

- For most meaningful signals and images  $a_n$  and  $b_n$  at the first “few”  $n$  values can reconstruct a signal close-enough to the original signal
- This idea “**sparsity**” is the basis of audio, image, and video compression formats (e.g. MP3, JPEG, MPG)

# Computation of Fourier coefficients

$$f(x) = \sum_{n=0}^{\infty} a_n \cos\left(\frac{2\pi n x}{T}\right) + \sum_{n=0}^{\infty} b_n \sin\left(\frac{2\pi n x}{T}\right)$$

$$a_n = \frac{2}{T} \int_{-T/2}^{T/2} f(x) \cos\left(\frac{2\pi n}{T}\right) dx$$

$$b_n = \frac{2}{T} \int_{-T/2}^{T/2} f(x) \sin\left(\frac{2\pi n}{T}\right) dx$$

This can be expressed in complex form as

$$f(x) = \sum_{n=-\infty}^{\infty} c_n \exp\left(\frac{-i2\pi n x}{T}\right)$$

where

$$c_n = \frac{1}{T} \int_{-T/2}^{T/2} f(x) \exp\left(\frac{-i2\pi n}{T}\right) dx$$

# Discrete Fourier transform

DFT

$$F[k] = \sum_{n=0}^{N-1} f[n] e^{-i2\pi u k} = \sum_{n=0}^{N-1} f[n] e^{-i2\pi \frac{n}{N} k}$$

IDFT

$$f[n] = \frac{1}{N} \sum_{k=0}^{N-1} F[k] e^{i2\pi u n} = \frac{1}{N} \sum_{k=0}^{N-1} F[k] e^{i2\pi \frac{n}{N} k}$$

# Properties of Fourier transform

## Properties of the DFT

Property	Time Domain	Frequency Domain
Linearity	$ax_1[n] + bx_2[n]$	$aX_1[k] + bX_2[k]$
Time-shifting	$x[n - m]$	$e^{-j2\pi km}X(k)$
Frequency-shifting (modulation)	$e^{-j2\pi k_0 n/N}x[n]$	$X(k - k_0)$
Time-convolution	$x_1[n] \otimes x_2[n]$	$X_1[k]X_2[k]$
Frequency-convolution	$x_1[n]x_2[n]$	$X_1[k] \otimes X_2[k]$
Parseval's relation	$E_x = \sum_{n=0}^{N-1}  x[n] ^2$	$E_x = \sum_{k=0}^{N-1}  X[k] ^2$

# Fourier transform in two dimensions

- The **two-dimensional** Fourier transform and its inverse
  - Fourier transform

$$F(u, v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x, y) e^{-j2\pi(ux/M + vy/N)}$$

for  $u = 0, 1, 2, \dots, M - 1, v = 0, 1, 2, \dots, N - 1$

- Inverse Fourier transform:

$$f(x, y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u, v) e^{j2\pi(ux/M + vy/N)}$$

for  $x = 0, 1, 2, \dots, M - 1, y = 0, 1, 2, \dots, N - 1$

- $u, v$  : the transform or frequency variables
- $x, y$  : the spatial or image variables

# Fourier transform in two dimensions

- We define the Fourier spectrum, phase angle, and power spectrum as follows:

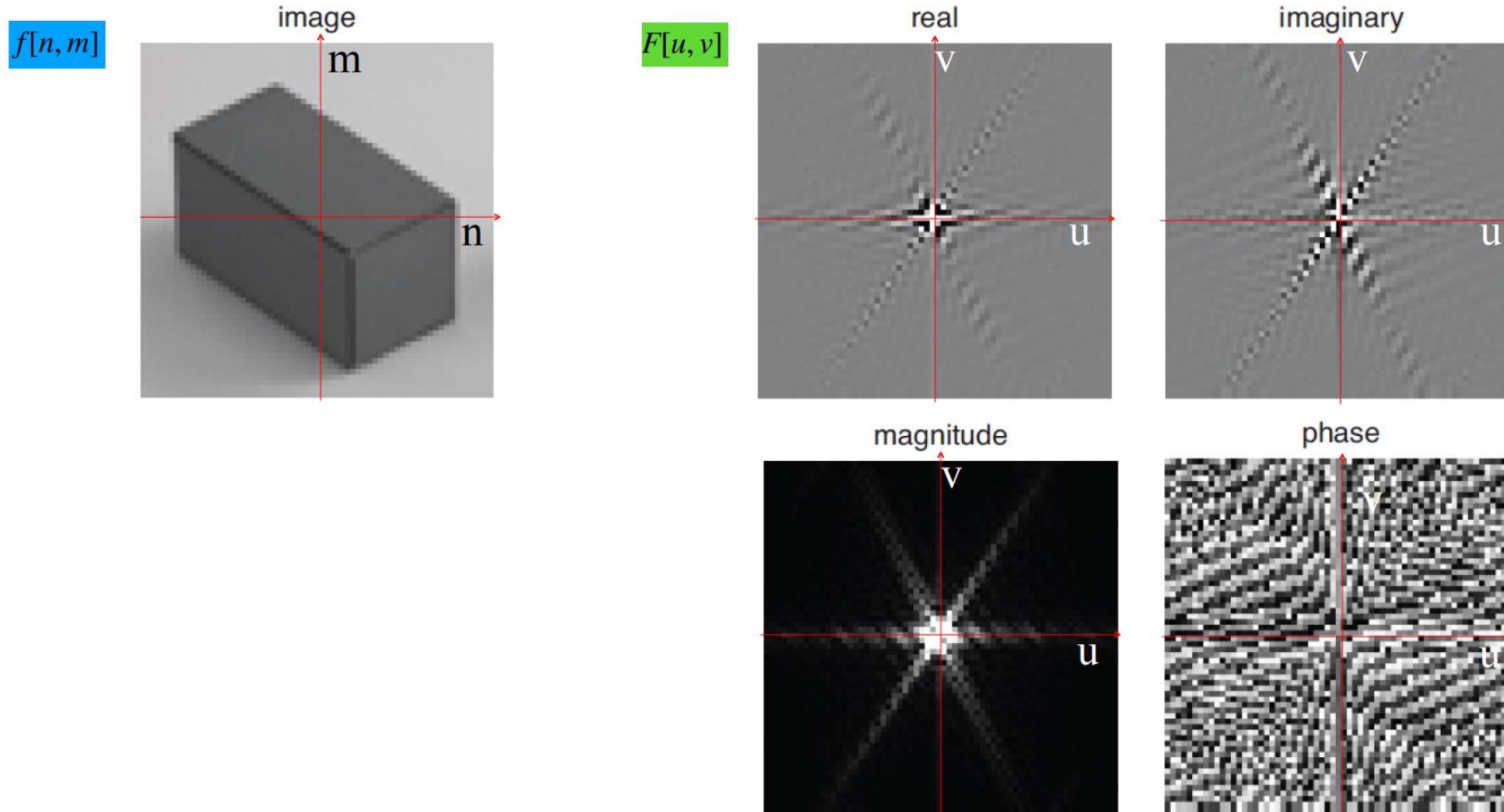
$$|F(u, v)| = \left[ R^2(u, v) + I^2(u, v) \right]^{\frac{1}{2}} \quad \text{(Magnitude)}$$

$$\phi(u, v) = \tan^{-1} \left[ \frac{I(u, v)}{R(u, v)} \right] \quad \text{(phase angle)}$$

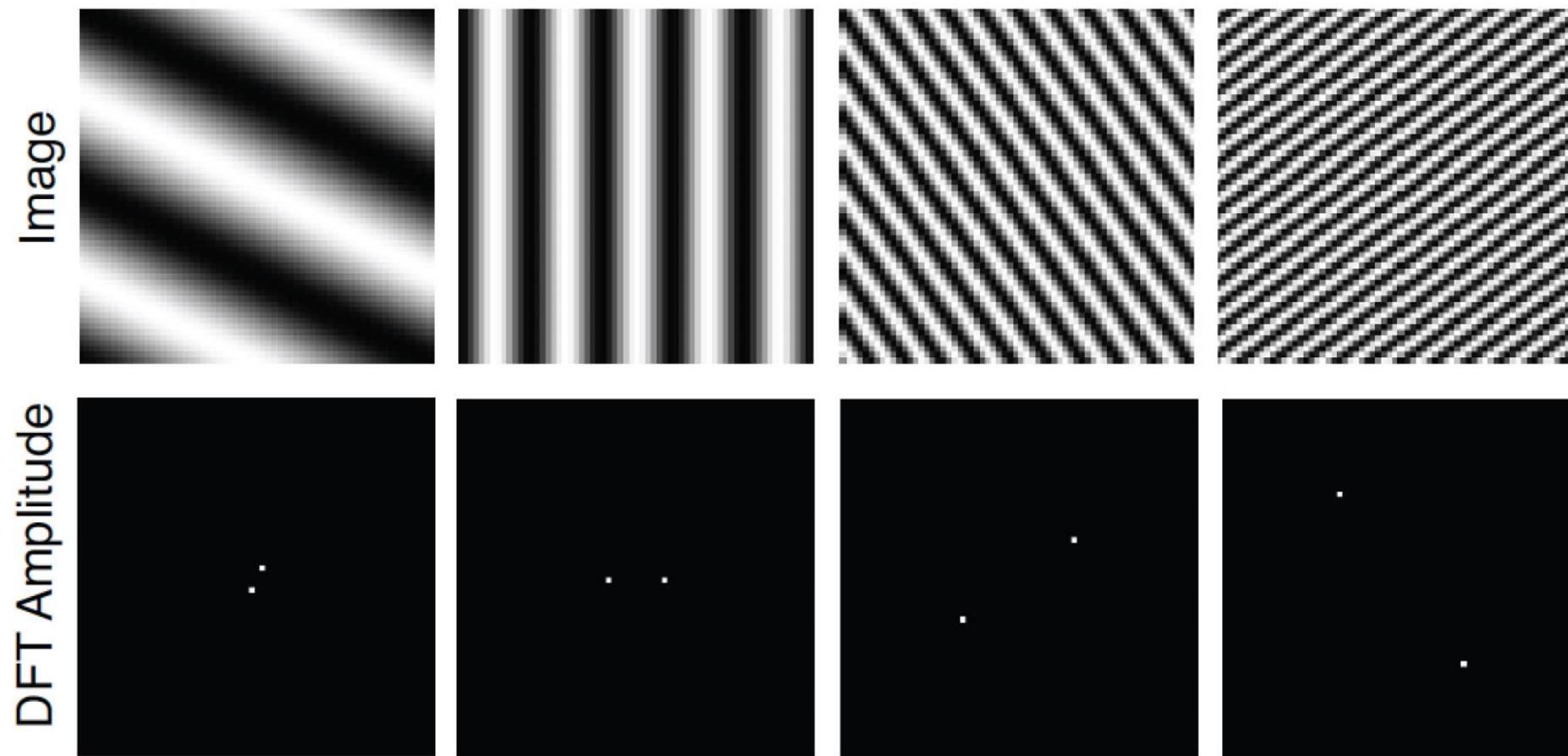
$$P(u, v) = |F(u, v)|^2 = R^2(u, v) + I^2(u, v) \quad \text{(power spectrum)}$$

- $R(u, v)$ : the real part of  $F(u, v)$
- $I(u, v)$ : the imaginary part of  $F(u, v)$

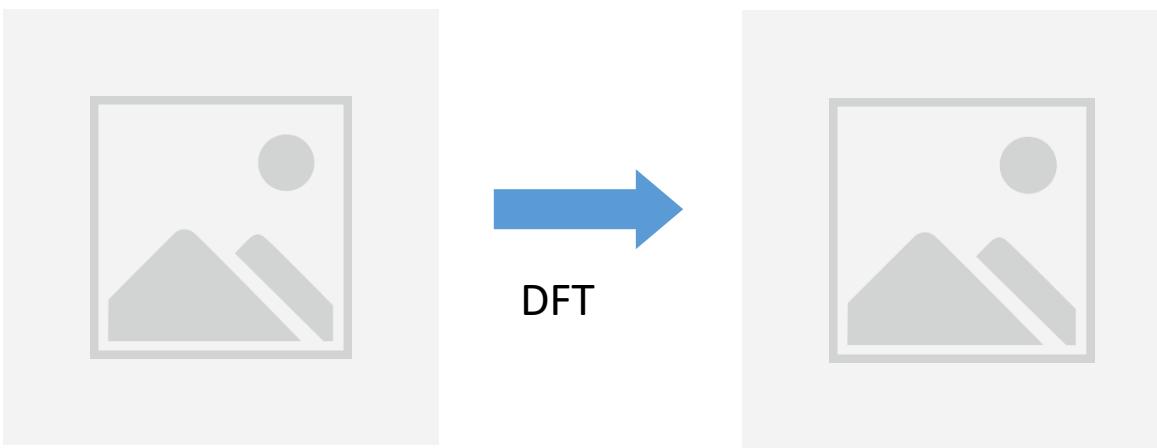
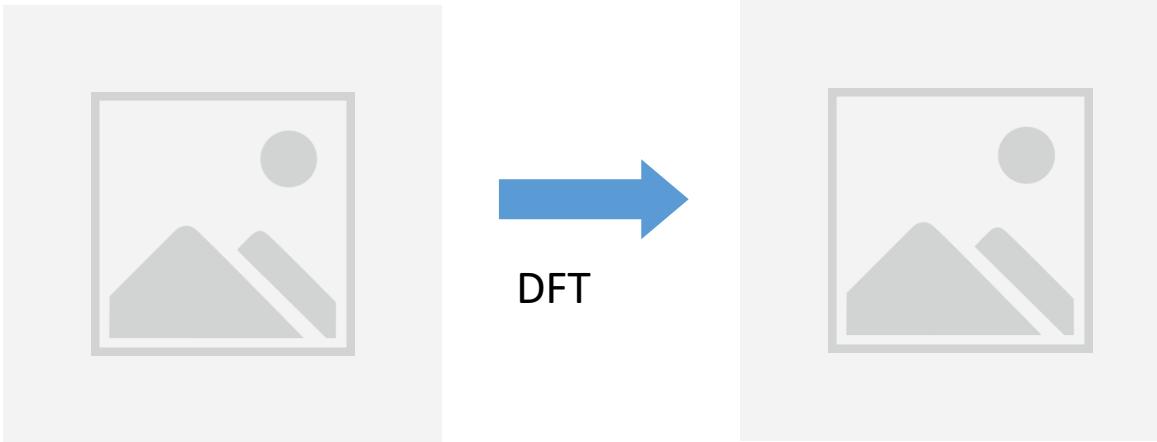
# Visualizing the 2D Fourier Transform



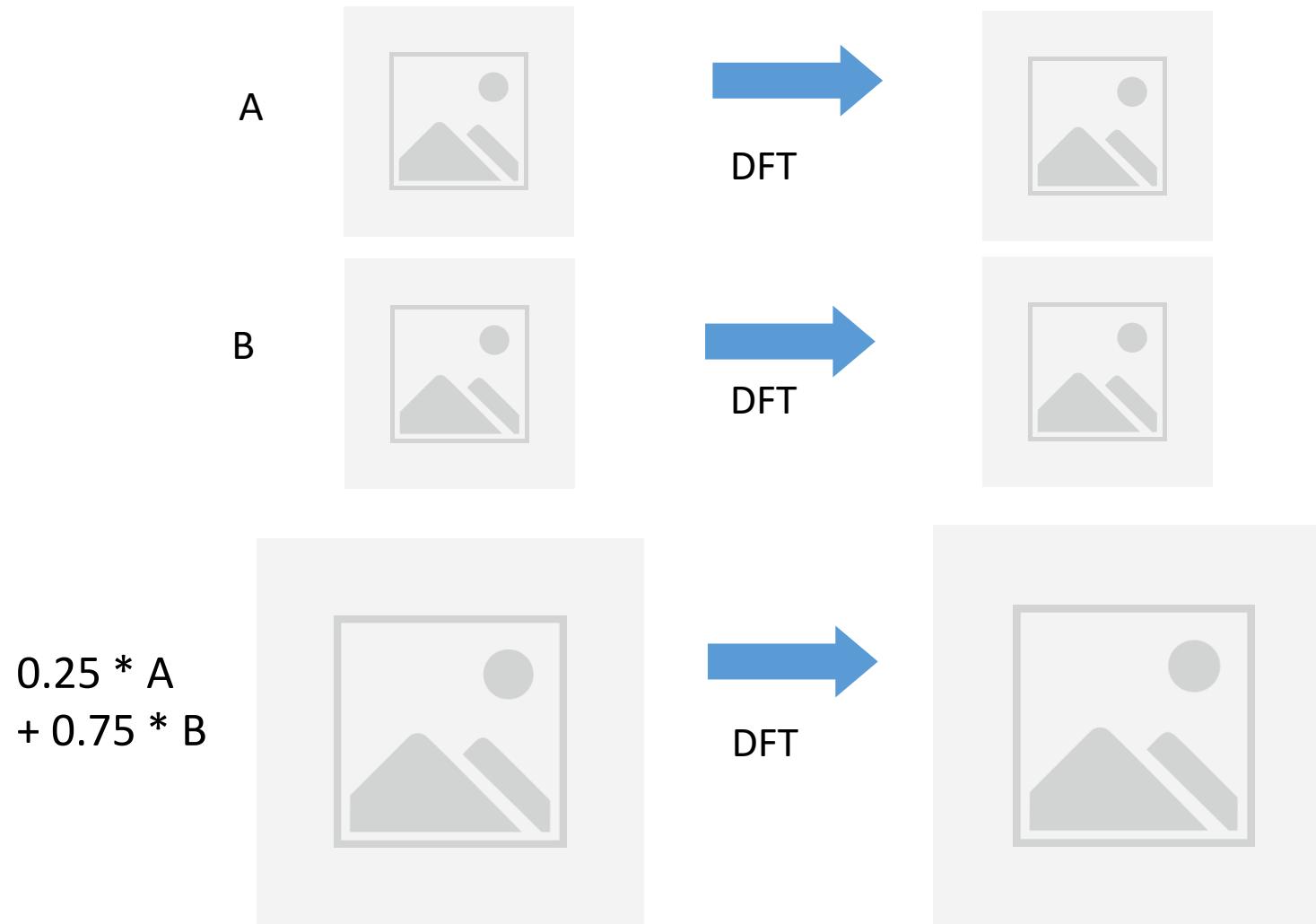
# Examples of 2D DFT Pairs



# 2D DFT Rotation property



# DFT properties: Linearity



# 2D DFT of sample functions



2D Gaussian  
function

Its DFT magnitude  
is another Gaussian

# 2D DFT in Matlab

- fft2 command, using fftshift to relocate the center frequency.

```
>> A=imread('boat.png');
>> B=fft2(A);
>> B(3, 6)

ans =

-3.5586e+05 + 1.6945e+05i

>> imagesc(log(abs(B)));
>> C=fftshift(fft2(A));
>> figure;imagesc(log(abs(C))');
```

Source code



Original image

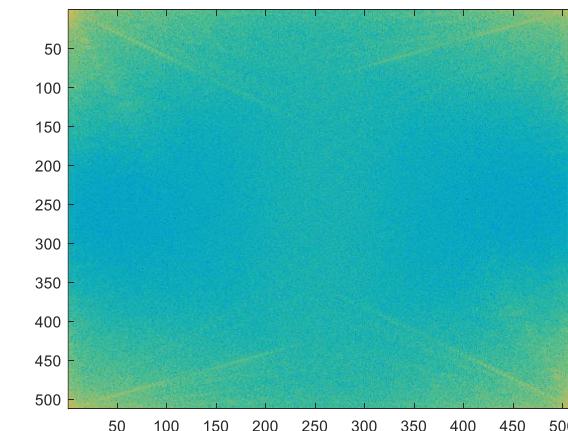


Image B log-magnitude spectrum

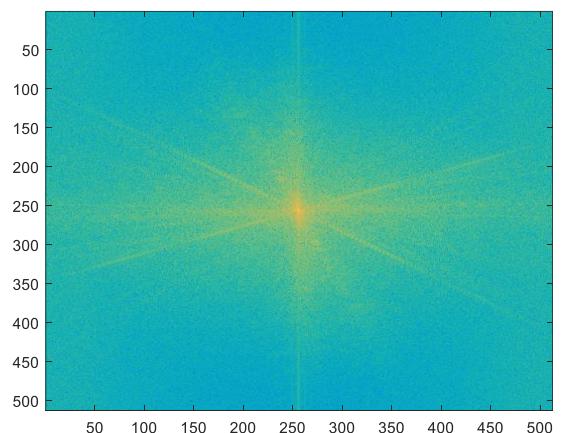
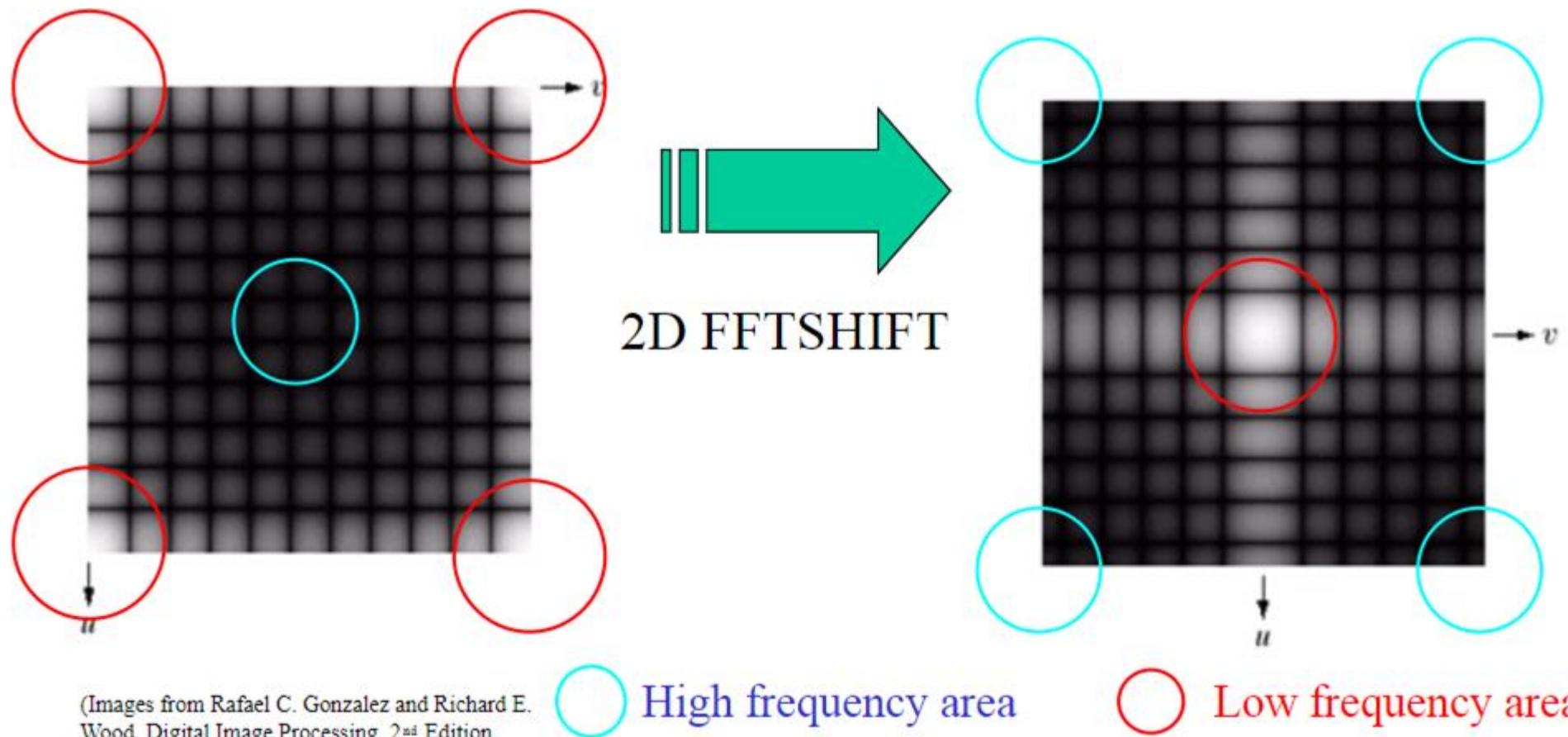


Image C log-magnitude spectrum

# Shifting quadrants



# Shifting quadrants ‘fftshift’

- fftshift shifts quadrants to move the zero-frequency component to the center of the array

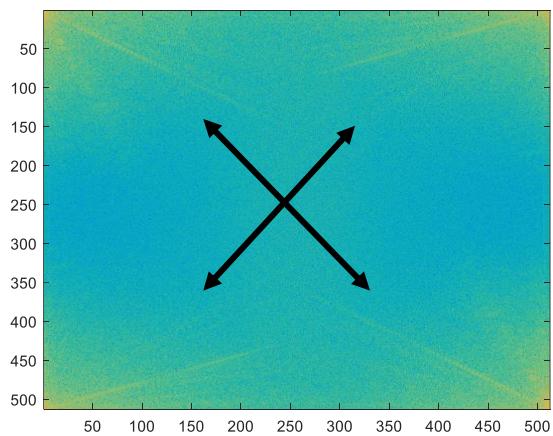
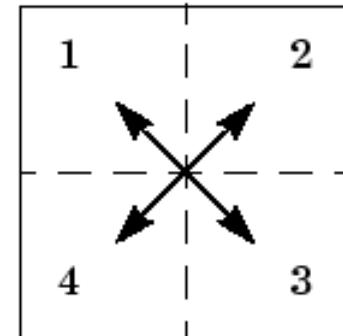


Image B

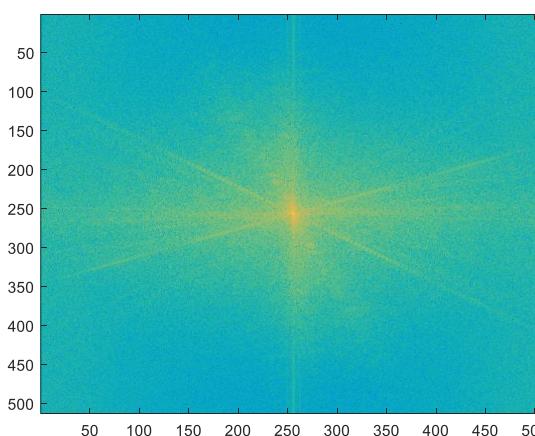


Image C

# Properties of the 2D DFT

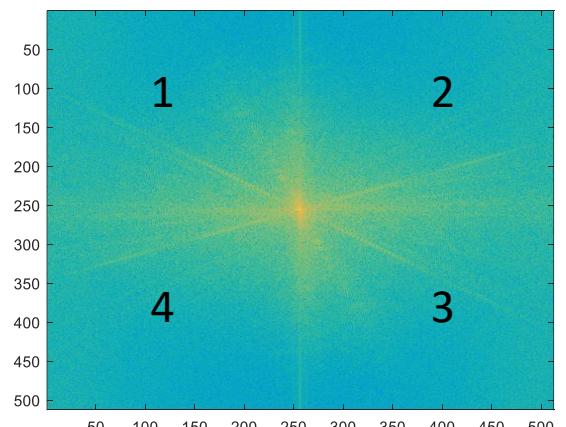
- Some properties of 2D DFT:

$$F(u, v) = F^*(-u, -v) \quad (\text{conjugate symmetric})$$

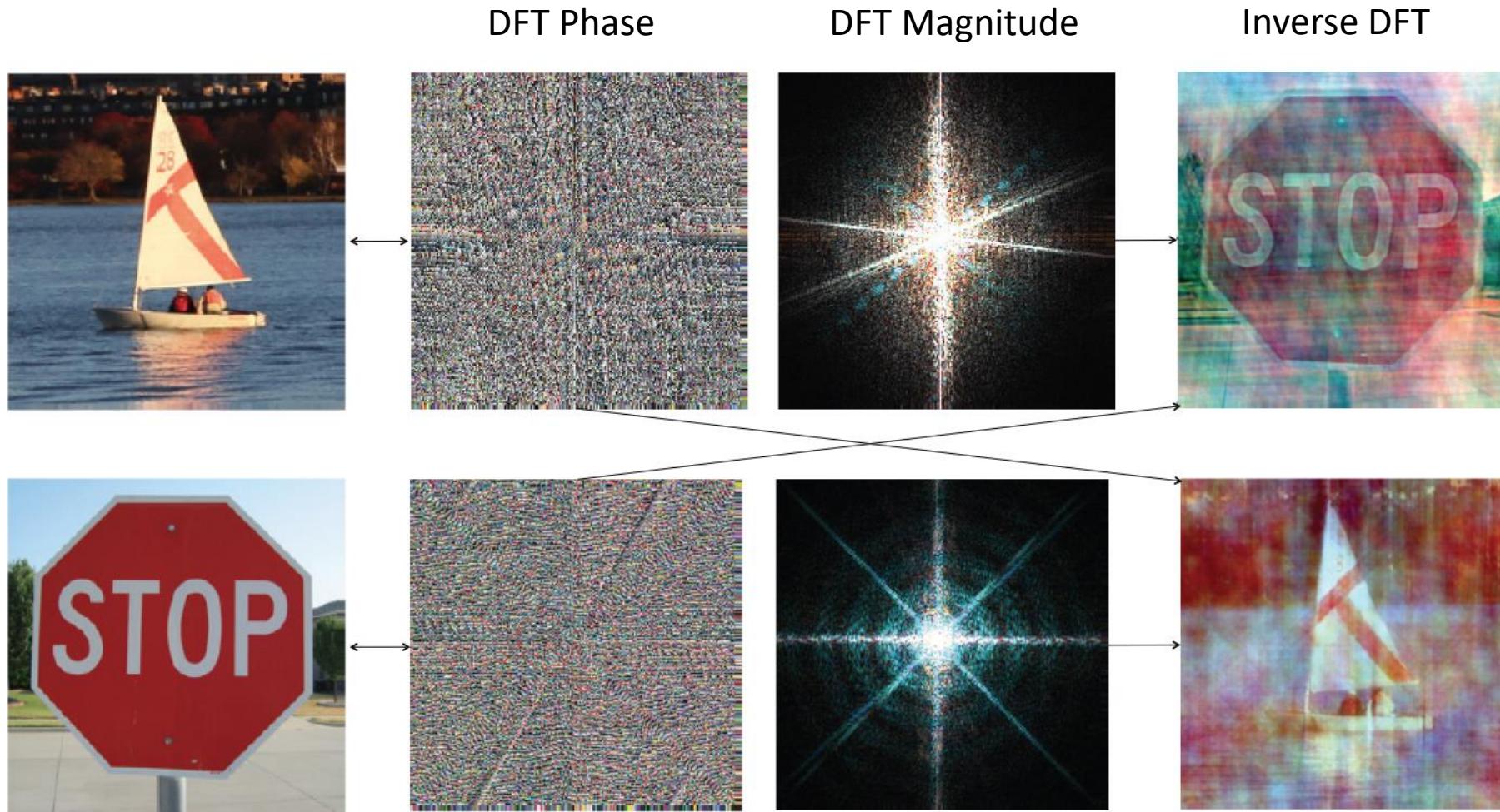
$$|F(u, v)| = |F(-u, -v)| \quad (\text{symmetric})$$

This is symmetry around the origin. DFT values in the first quadrant are the complex conjugate of values in the third quadrant.

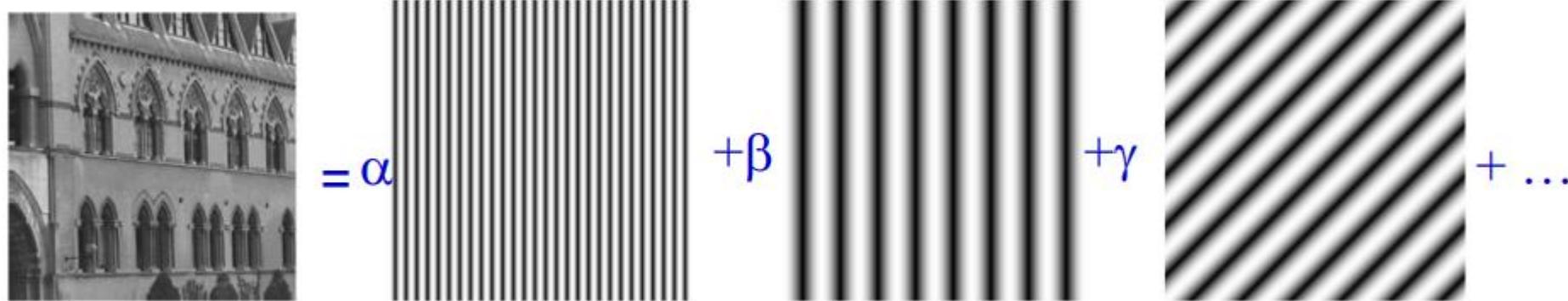
Similarly, DFT values in the second quadrant are complex conjugates of values in the fourth quadrants.



# Mixing the phase and magnitude

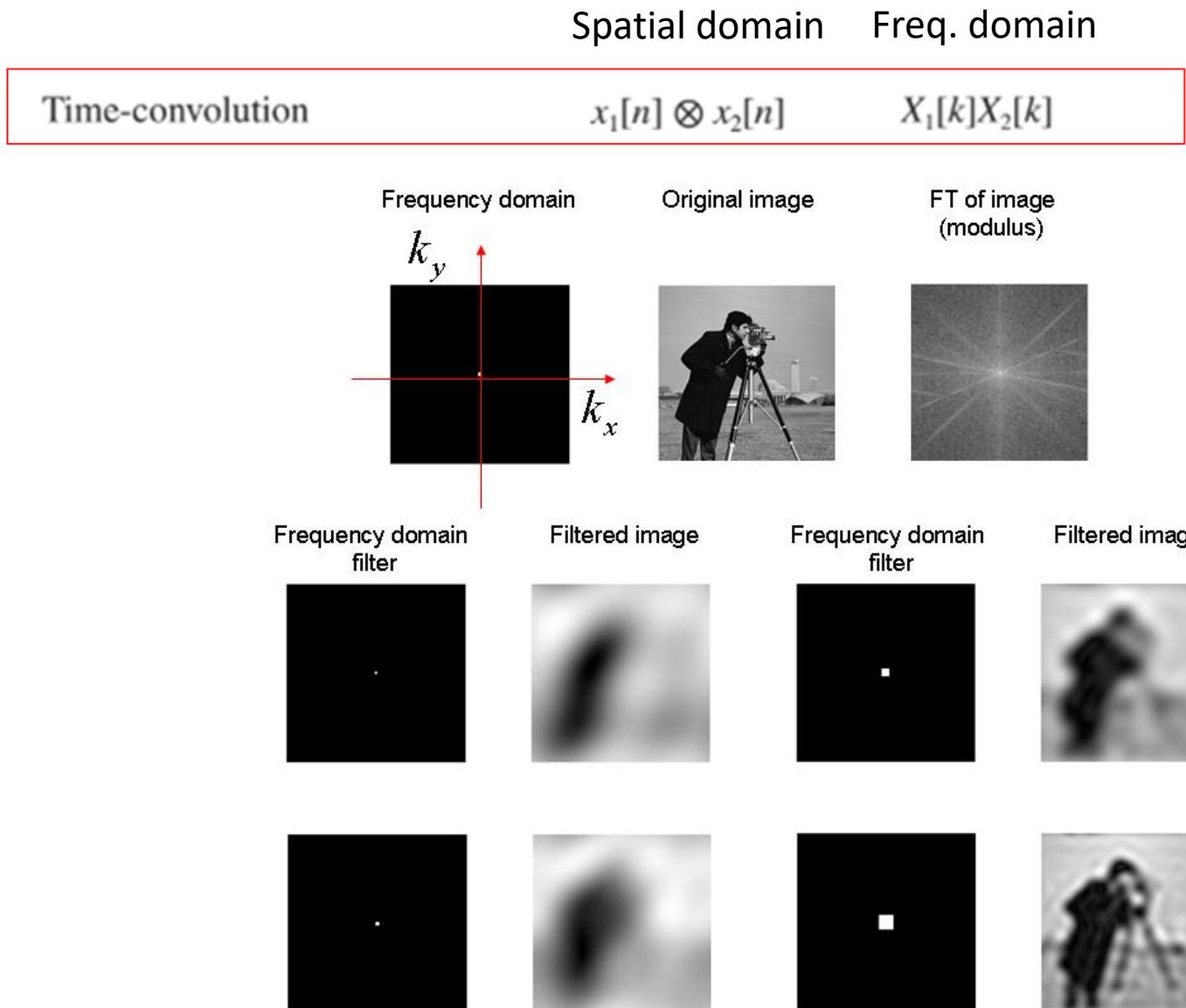


# Reconstructing an image from its Fourier coefficients



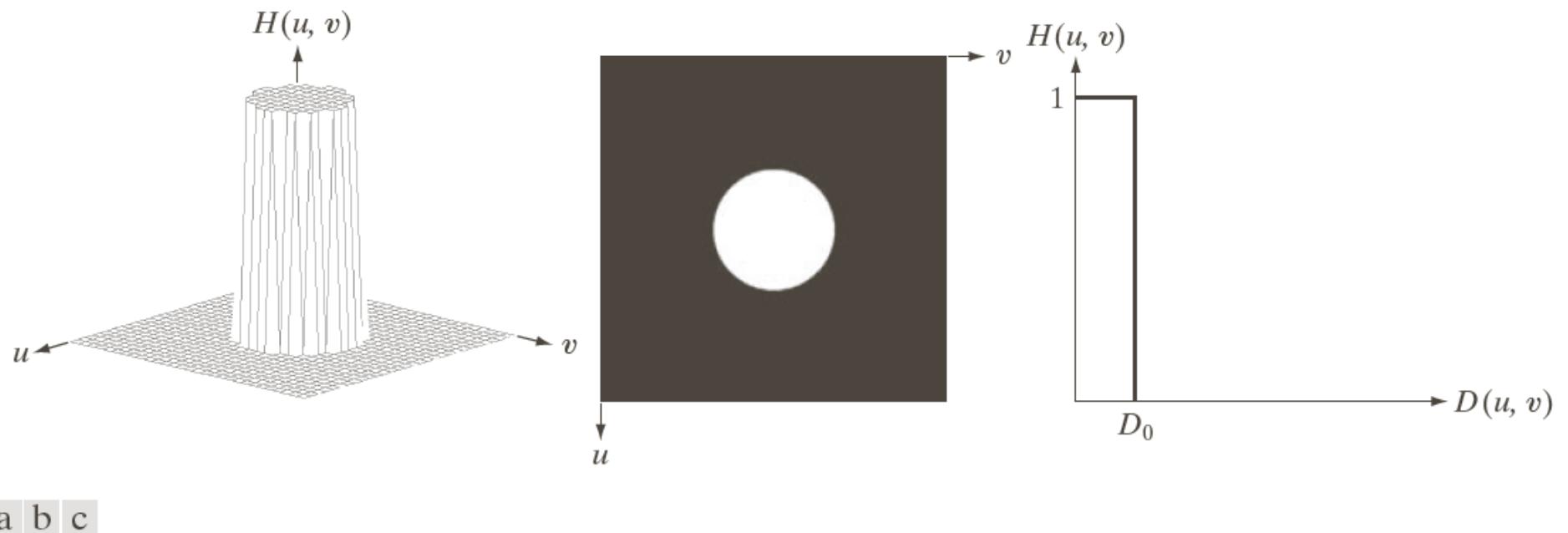
The subimages are obtained by taking the inverse 2D DFT of each point and its complement in the  $(u, v)$  plane

# Frequency domain filtering example



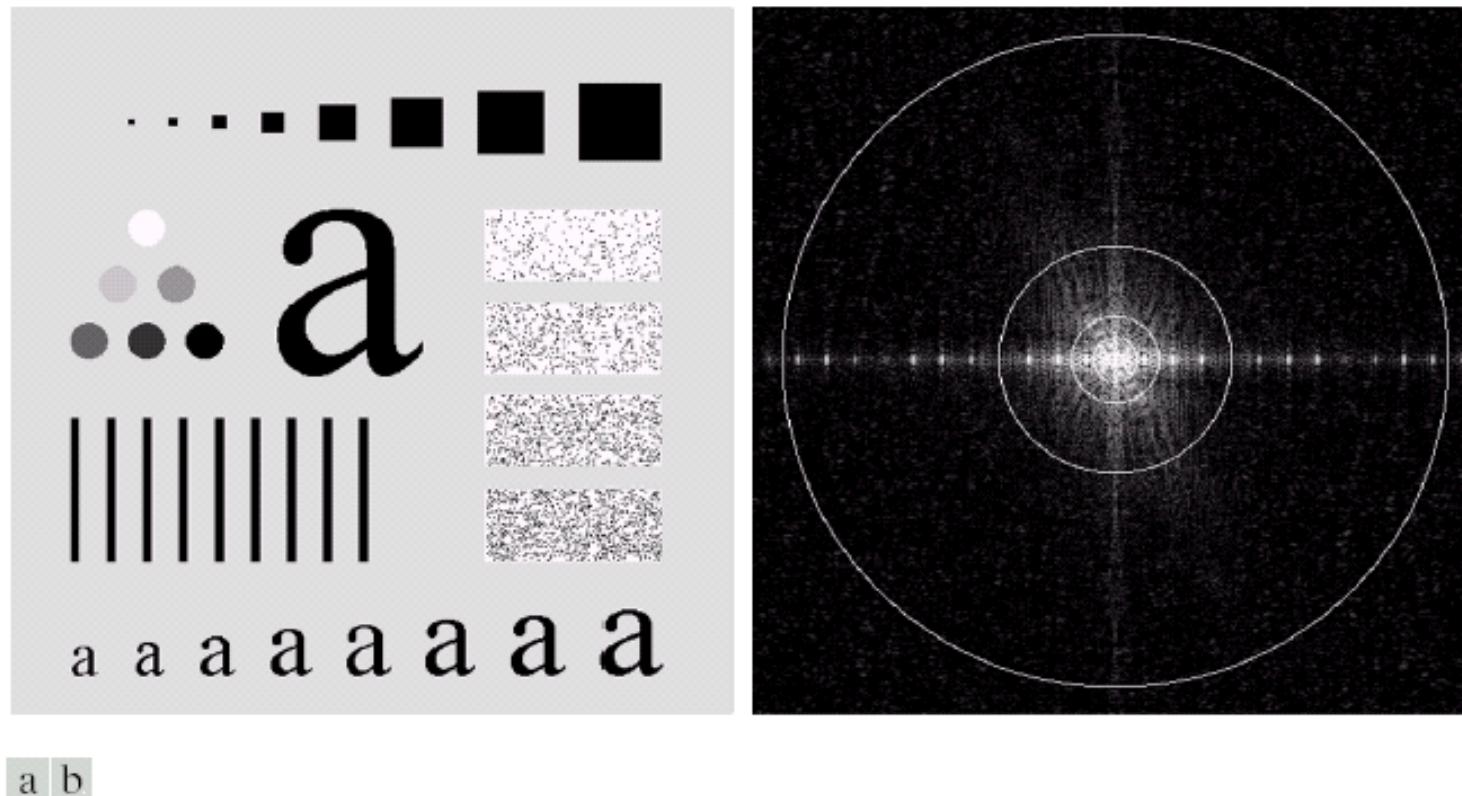
Low pass filtering  
of a sample image

# Ideal Lowpass filter



**FIGURE 4.40** (a) Perspective plot of an ideal lowpass-filter transfer function. (b) Filter displayed as an image. (c) Filter radial cross section.

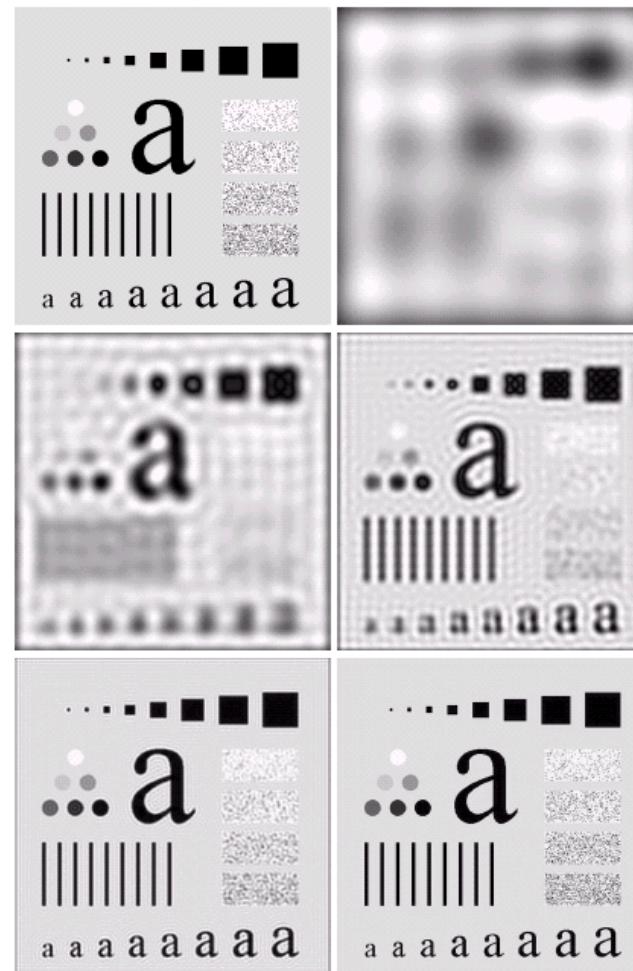
# Ideal Lowpass filter



a b

**FIGURE 4.11** (a) An image of size  $500 \times 500$  pixels and (b) its Fourier spectrum. The superimposed circles have radii values of 5, 15, 30, 80, and 230, which enclose 92.0, 94.6, 96.4, 98.0, and 99.5% of the image power, respectively.

# Ideal Lowpass filter

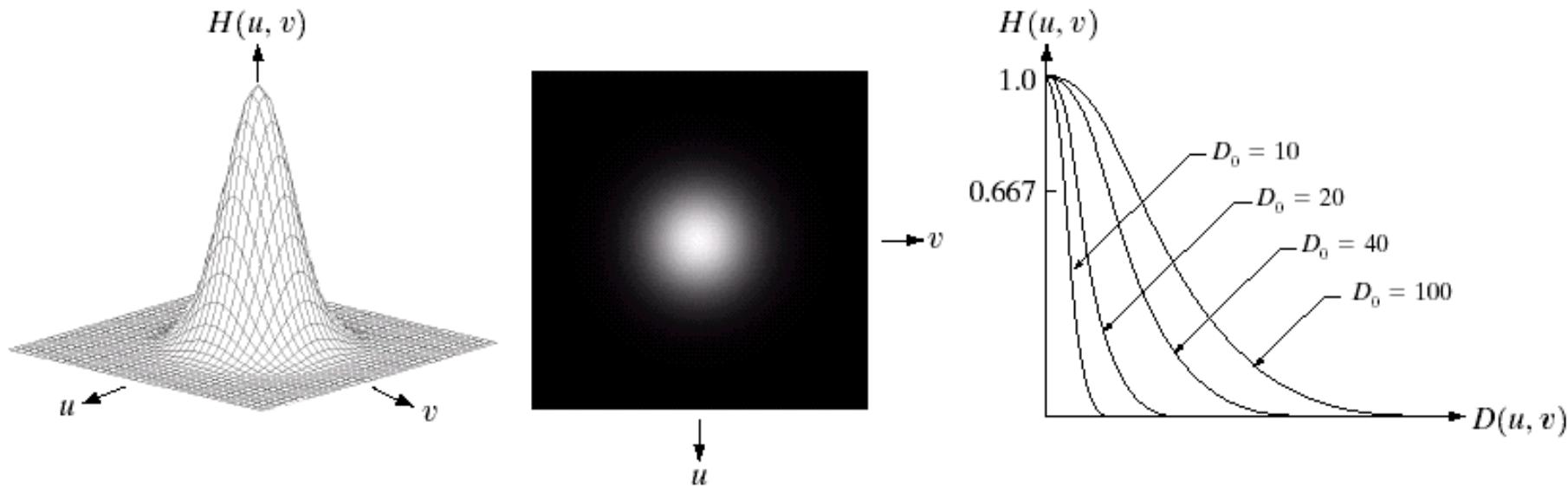


a b  
c d  
e f

**FIGURE 4.12** (a) Original image. (b)–(f) Results of ideal lowpass filtering with cutoff frequencies set at radii values of 5, 15, 30, 80, and 230, as shown in Fig. 4.11(b). The power removed by these filters was 8, 5.4, 3.6, 2, and 0.5% of the total, respectively.

# Gaussian Lowpass filters

$$H(u, v) = e^{-D^2(u,v)/2D_0^2}$$



a b c

**FIGURE 4.17** (a) Perspective plot of a GLPF transfer function. (b) Filter displayed as an image. (c) Filter radial cross sections for various values of  $D_0$ .

# Gaussian LPF's

$D_0=5, 15, 30, 80,$  and  $230$

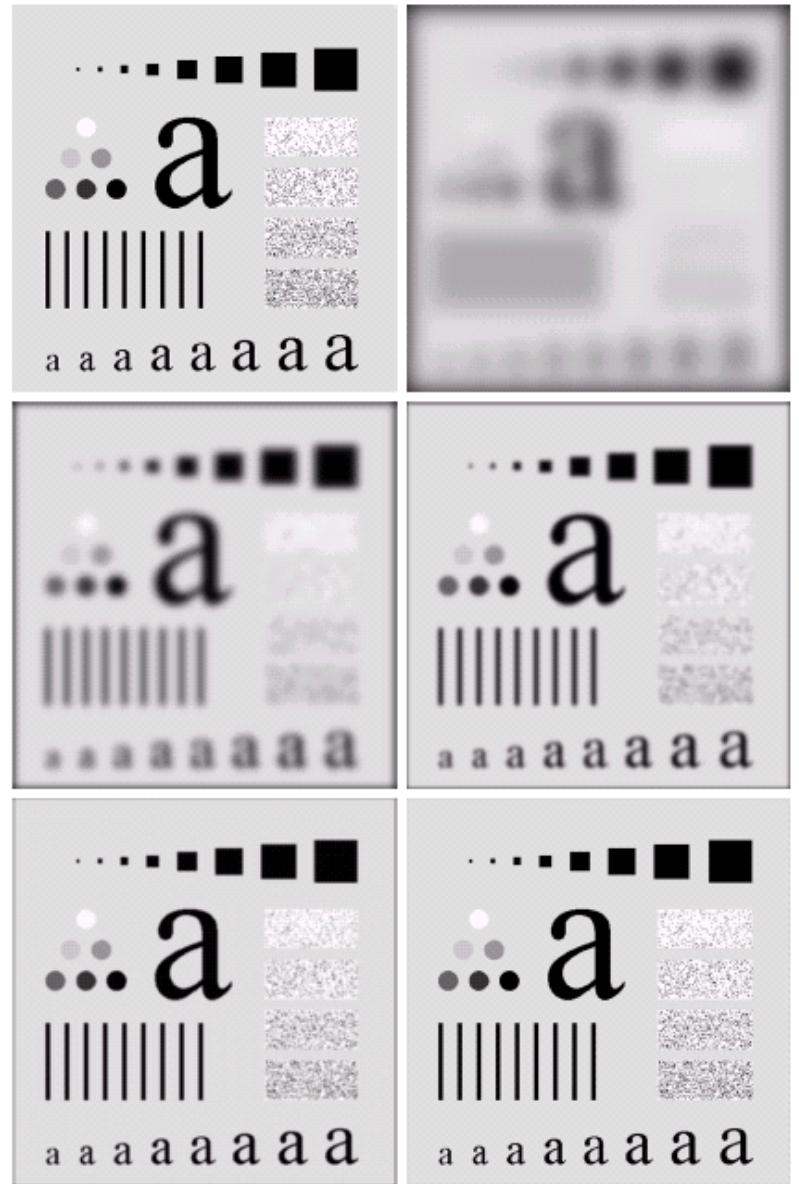


FIGURE 4.18 (a) Original image. (b)–(f) Results of filtering with Gaussian lowpass filters with cutoff frequencies set at radii values of 5, 15, 30, 80, and 230, as shown in Fig. 4.11(b). Compare with Figs. 4.12 and 4.15.

a  
b  
c  
d  
e  
f

# Gaussian Lowpass filters

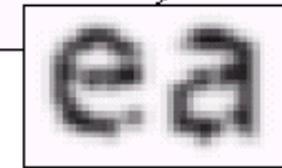
a b

**FIGURE 4.19**  
(a) Sample text of poor resolution (note broken characters in magnified view).  
(b) Result of filtering with a GLPF (broken character segments were joined).

Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.



Historically, certain computer programs were written using only two digits rather than four to define the applicable year. Accordingly, the company's software may recognize a date using "00" as 1900 rather than the year 2000.



# Laplacian in the frequency domain

From Fourier Tr. Property:

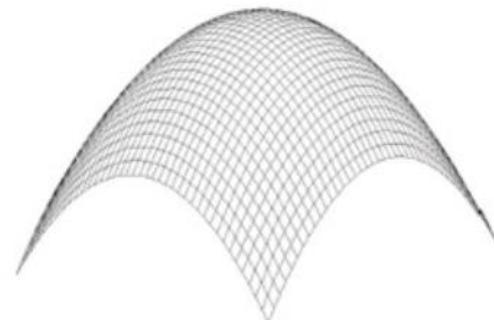
$$\frac{d^n f(x)}{dx^n} \Leftrightarrow (ju)^n F(u)$$

Then for Laplacian operator

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \Leftrightarrow -(u^2 + v^2)F(u, v)$$

We get

$$\nabla^2 \Leftrightarrow -(u^2 + v^2)$$



Surface plot

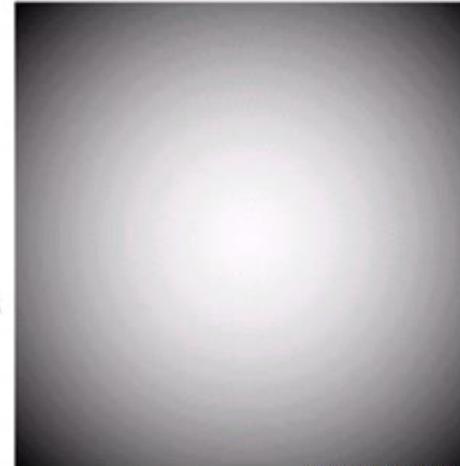


Image of  
 $-(u^2 + v^2)$

(Images from Rafael C. Gonzalez and Richard E. Wood, Digital Image Processing, 2<sup>nd</sup> Edition.)

# Laplacian in the frequency domain

- The Laplacian filter

$$H(u, v) = -(u^2 + v^2)$$

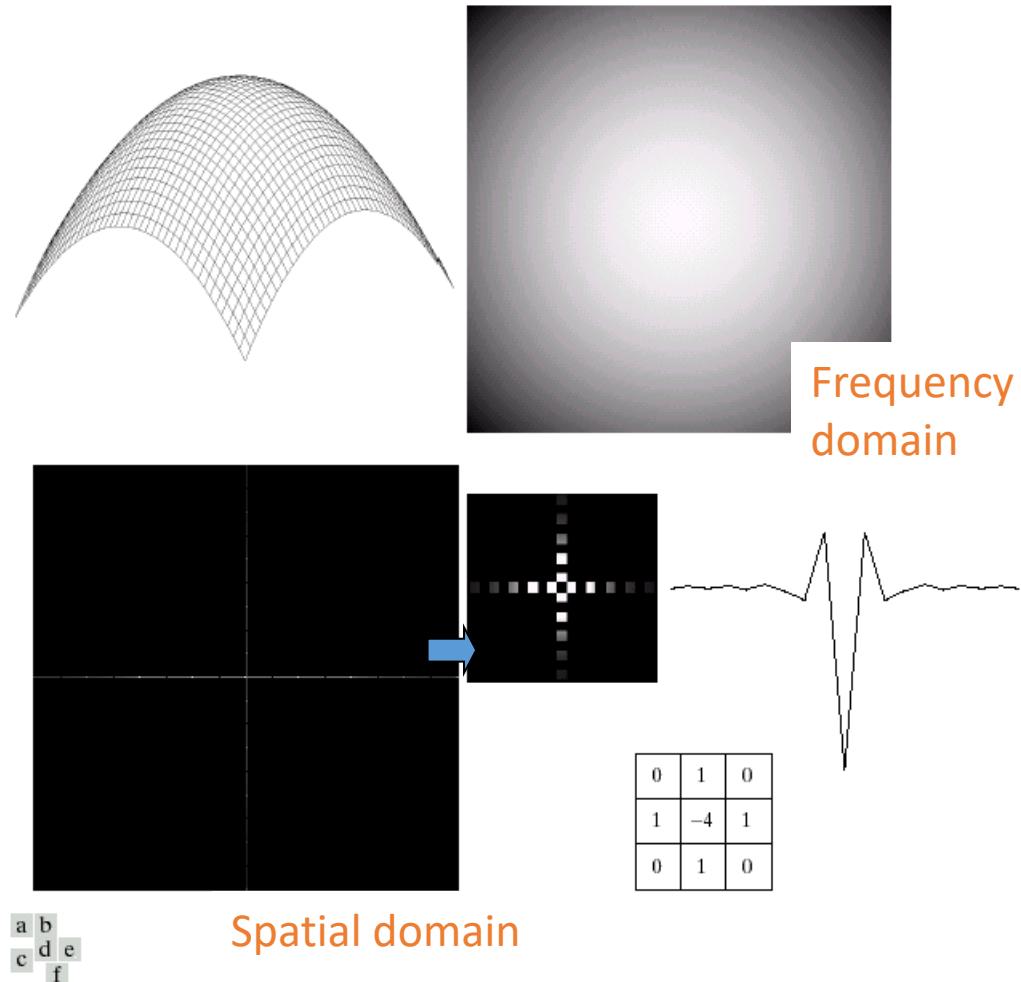


FIGURE 4.27 (a) 3-D plot of Laplacian in the frequency domain. (b) Image representation of (a). (c) Laplacian in the spatial domain obtained from the inverse DFT of (b). (d) Zoomed section of the origin of (c). (e) Gray-level profile through the center of (d). (f) Laplacian mask used in Section 3.7.

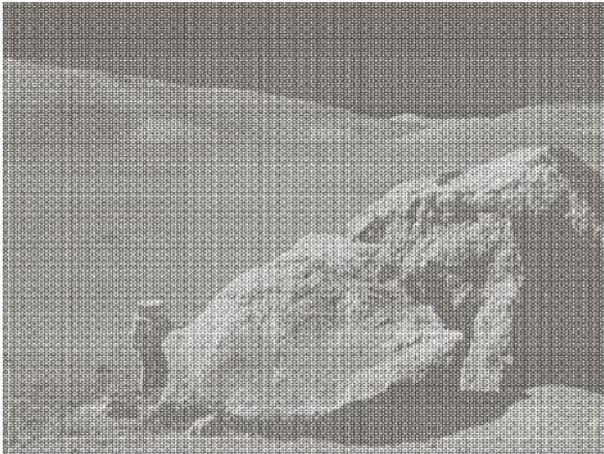
# Laplacian sharpening (edge enhancement)



a b

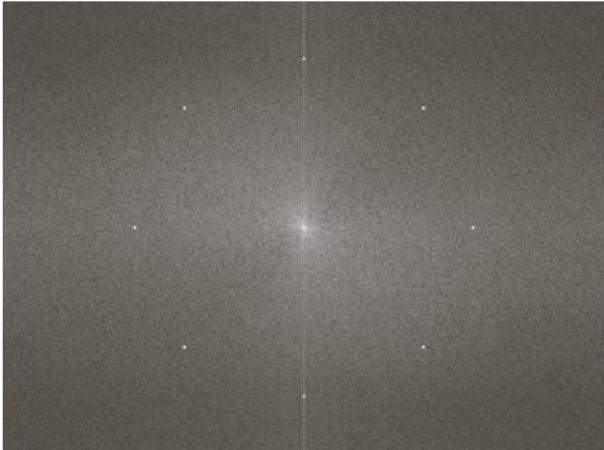
**FIGURE 4.58**  
(a) Original,  
blurry image.  
(b) Image  
enhanced using  
the Laplacian  
in the frequency  
domain.

# Selective filtering



a  
b

**FIGURE 5.5**  
(a) Image corrupted by sinusoidal noise.  
(b) Spectrum (each pair of conjugate impulses corresponds to one sine wave). (Original image courtesy of NASA.)

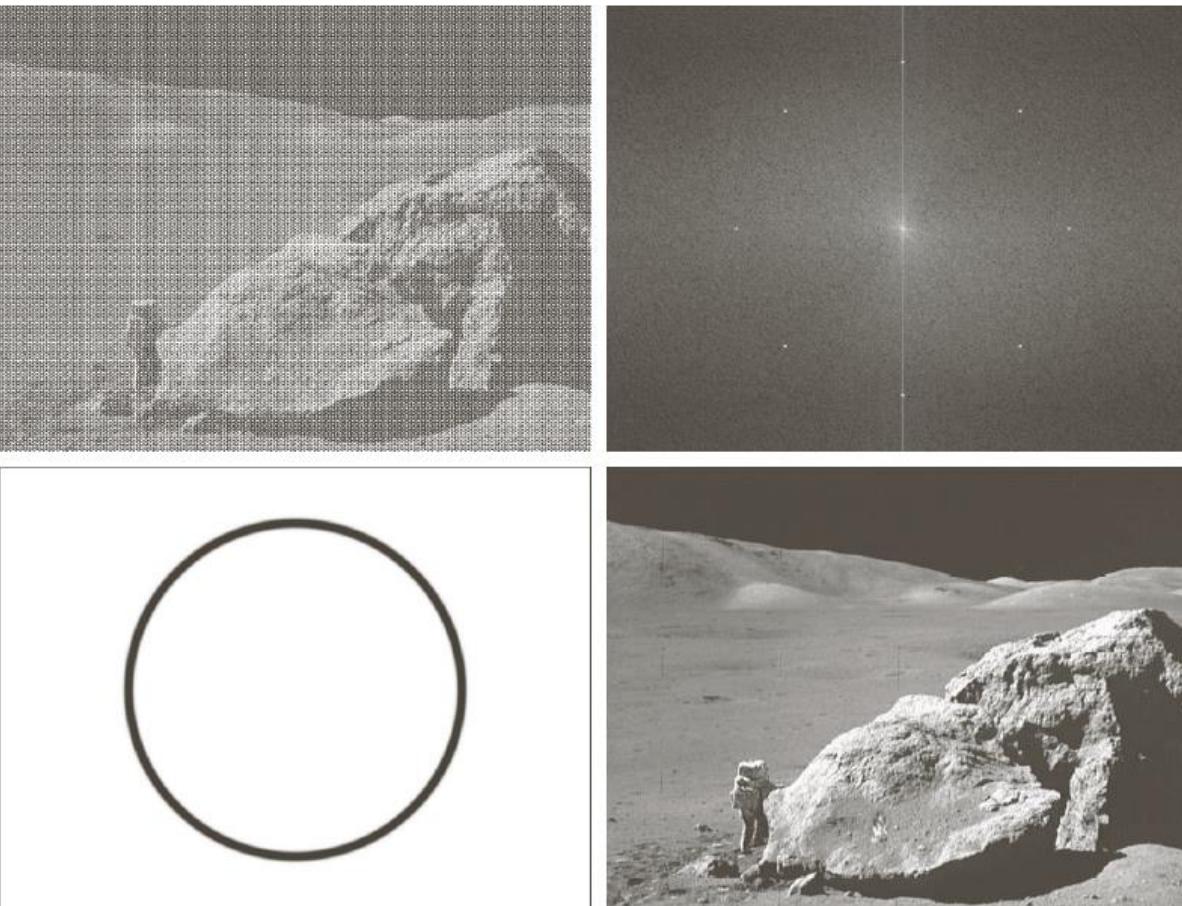


# Selective filtering

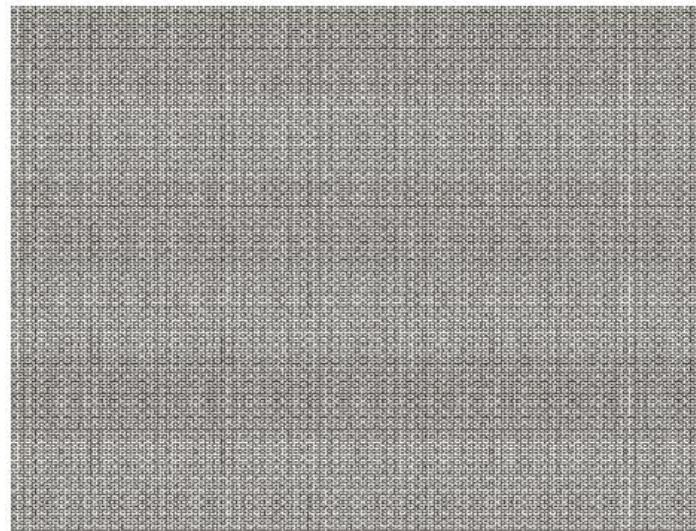
a  
b  
c  
d

**FIGURE 5.16**

(a) Image corrupted by sinusoidal noise.  
(b) Spectrum of (a).  
(c) Butterworth bandreject filter (white represents 1). (d) Result of filtering.  
(Original image courtesy of NASA.)

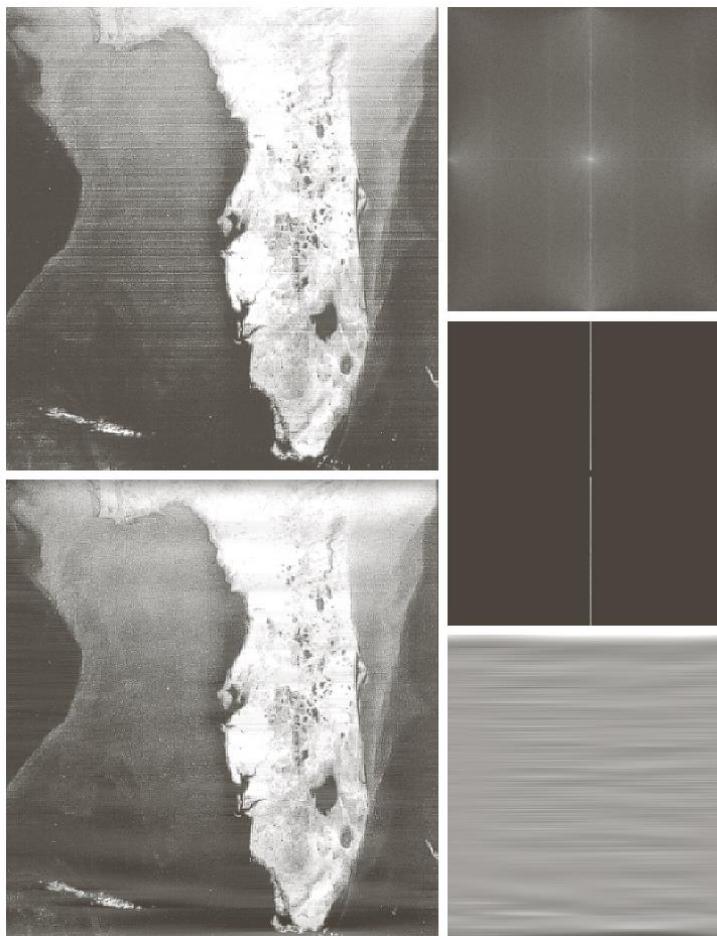


# Selective filtering



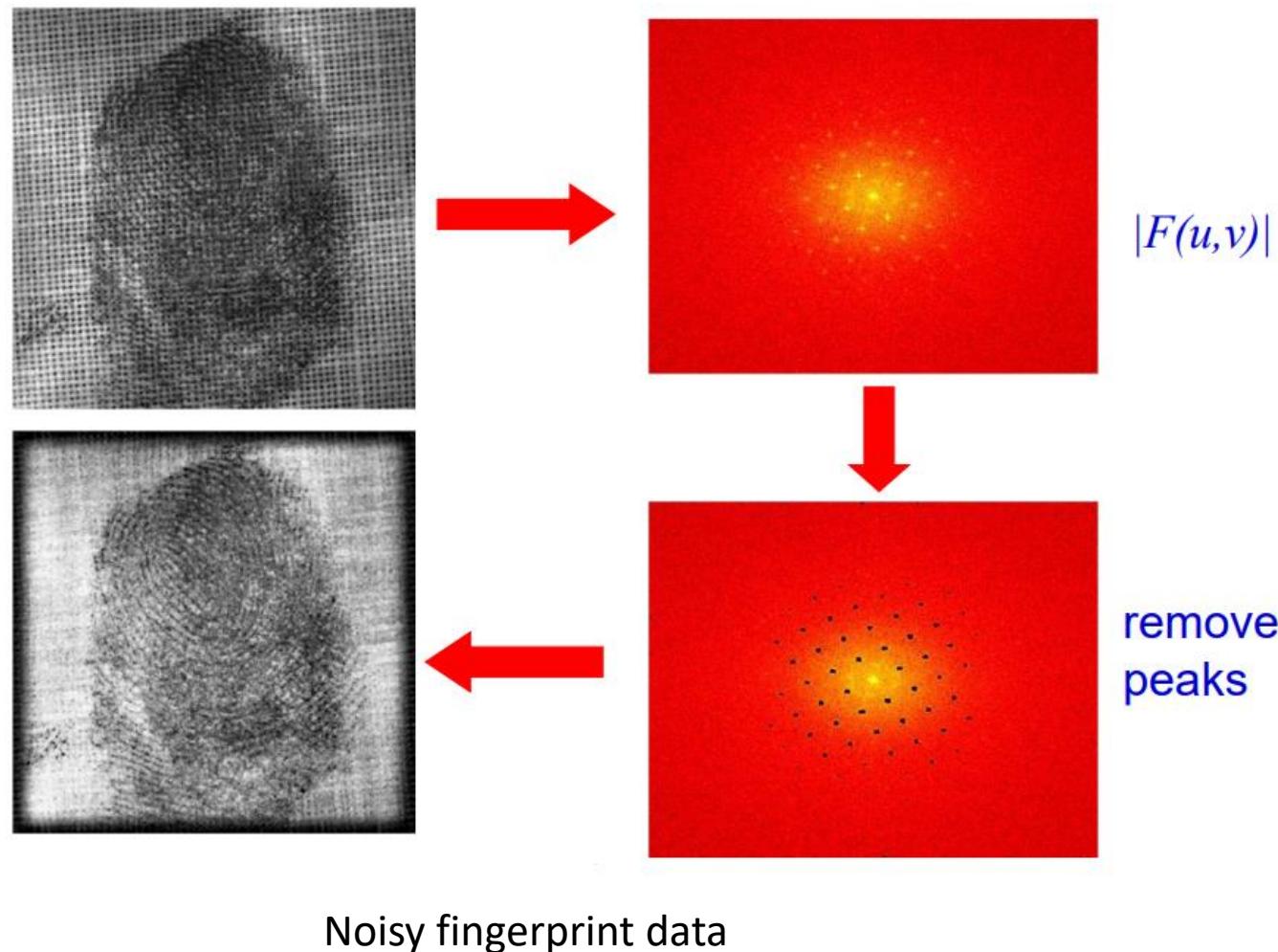
**FIGURE 5.17**  
Noise pattern of  
the image in  
Fig. 5.16(a)  
obtained by  
bandpass filtering.

# Selective filtering



**FIGURE 5.19**  
(a) Satellite image of Florida and the Gulf of Mexico showing horizontal scan lines.  
(b) Spectrum. (c) Notch pass filter superimposed on (b). (d) Spatial noise pattern. (e) Result of notch reject filtering.  
(Original image courtesy of NOAA.)

# Frequency domain operations



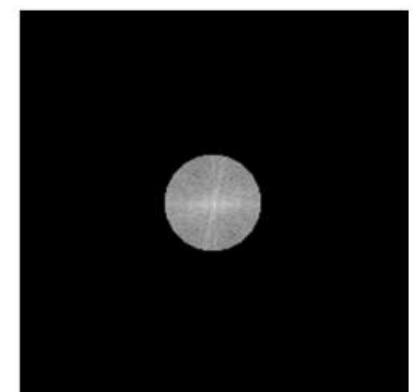
# Frequency domain filtering algorithms

- To get the low pass image:

1. Define a binary mask that is equal to 1 inside a circle and zero outside
2. Take the 2D Fourier transform of the image.
3. Multiply the 2D Fourier transform of the image by the binary mask
4. Take the inverse Fourier transform to get the low-passed image back

Change the filter to get the high pass image

low pass



# Frequency domain operations

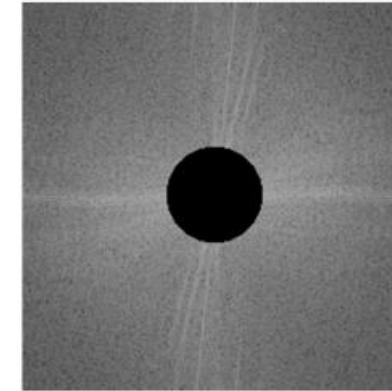
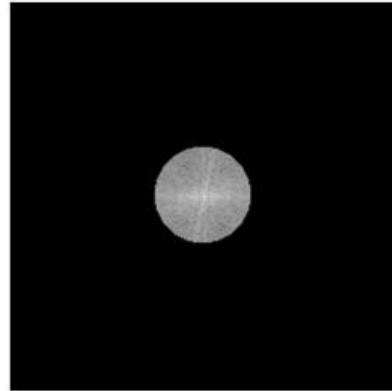
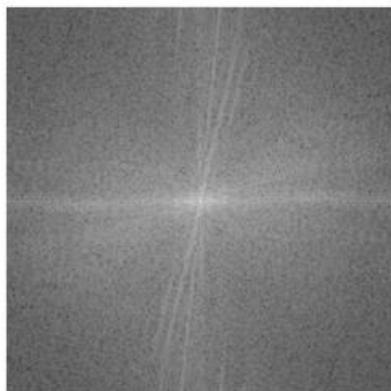
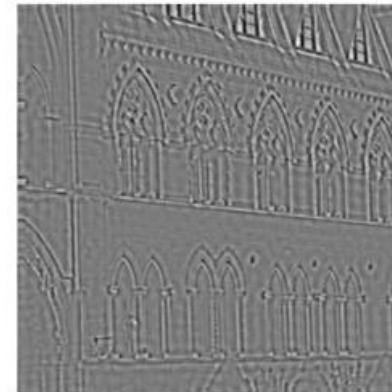
original



low pass



high pass

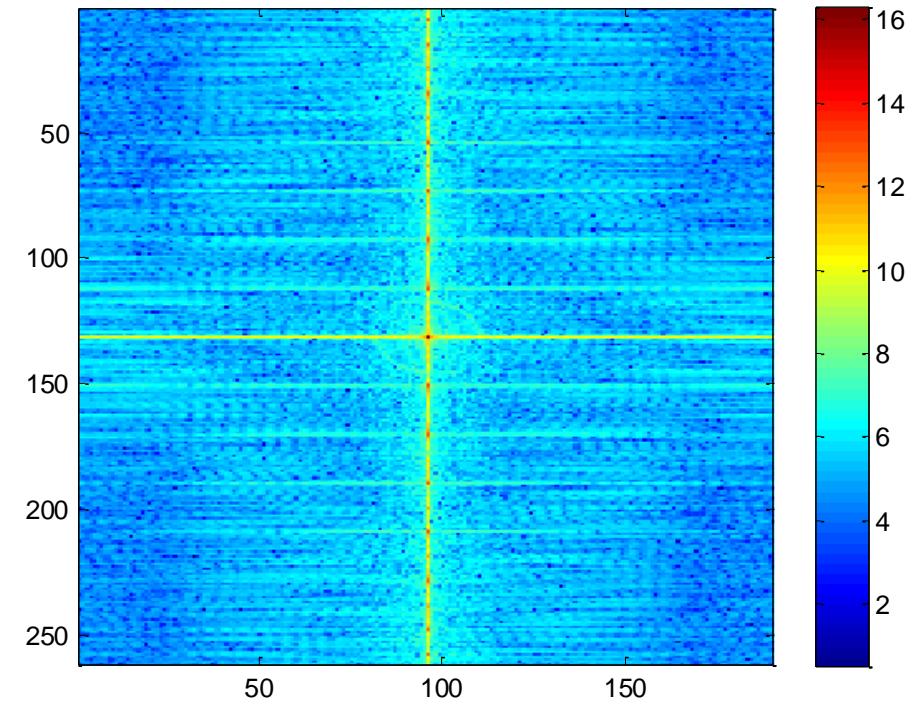


# Frequency domain filtering

- Example: Removing vertical/horizontal lines



Original image



Log-magnitude of its Fourier coefficients

# Frequency domain filtering

- Matlab example removeverticalline.m

# Filtering in the frequency domain

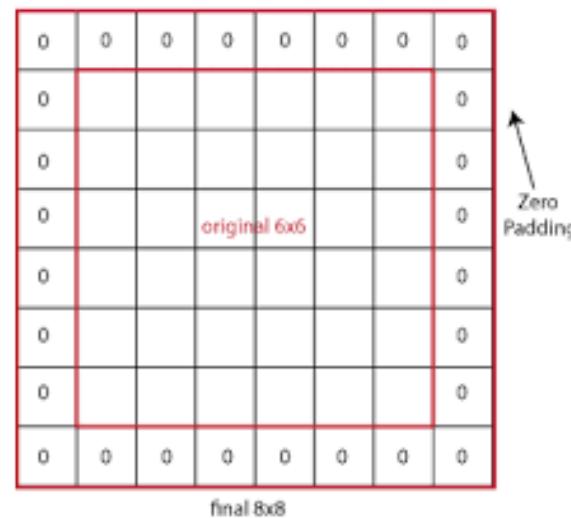
- Convolution in the time domain is equivalent to multiplication in the frequency domain

$$x_1(t) * x_2(t) \xleftrightarrow{F} X_1(f) \cdot X_2(f)$$

- In the spatial domain the filter and the image differ in size and the equation assumes that the signal is periodic. Zero-padding can be used to address both issues.

# Frequency Domain Filtering

- Assume that we like to filter an image in the frequency domain using a spatial domain filter (e.g. Gaussian, box,...etc)
- Filtering in the frequency domain is normally faster since it requires one multiplication per pixel
- Because of the mismatch in size, the filter should be zeropadded.
- Periodic extension or image zero padding will also improve results as they facilitate the periodic assumption in the Fourier convolution property.



# Filtering in the frequency domain

```
img = imread('cameraman.jpg');
%img = rgb2gray(img);      %Use of the image is in color
img=double(img);
[imh, imw] = size(img);
hs = 50; % filter half-size
fil = fspecial('gaussian', hs*2+1, 10); %size of the filter is hs*2+1 (odd) and sigma=10
fftsize = 1024; % preferred be be a power of 2 (for speed) and include padding
img_fft = fft2(img, fftsize, fftsize); % 1) fft im with padding; automatically zero-pads
fil_fft = fft2(fil, fftsize, fftsize); % 2) fft fil, pad to same size as image
img_fil_fft = img_fft .* fil_fft; % 3) multiply fft images
img_fil = ifft2(img_fil_fft); % 4) inverse fft2
img_fil = img_fil(1+hs:size(img,1)+hs, 1+hs:size(img, 2)+hs); % 5) remove padding
figure(1), imagesc(log(abs(fftshift(img_fft)))), axis image, colormap jet
figure(2), imagesc(img_fil), axis image, colormap gray
```

# Filtering in the frequency domain

- Matlab example filteringfreq.m

# Hybrid Images

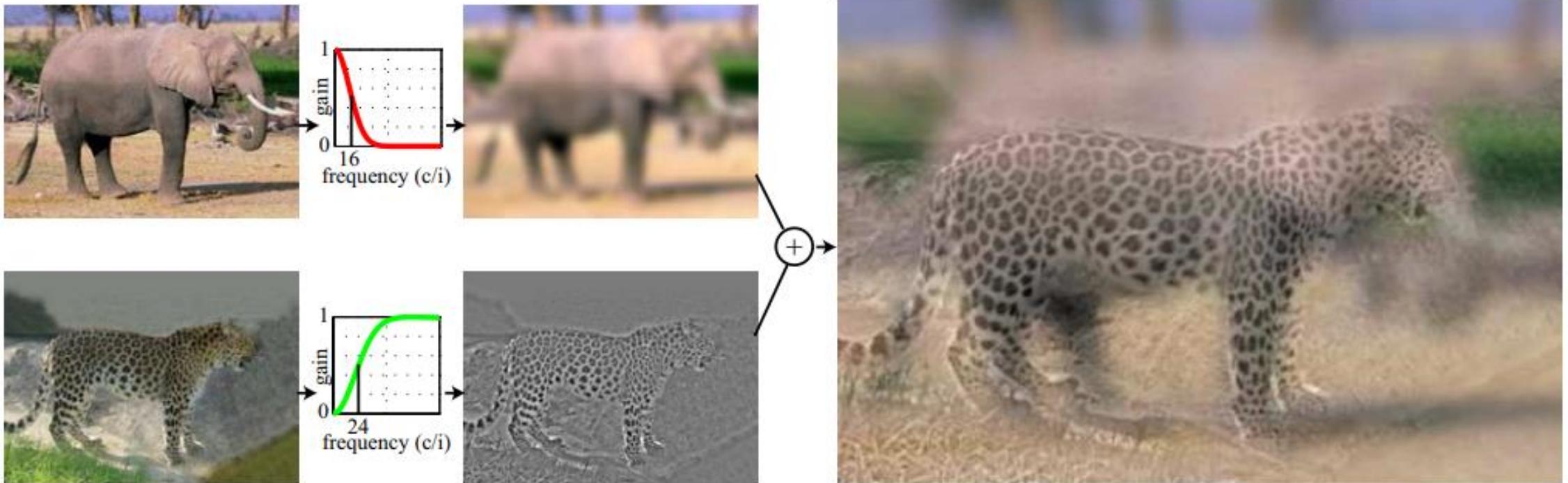


Figure 2: hybrid images are generated by superimposing two images at two different spatial scales: the low-spatial scale is obtained by filtering one image with a low-pass filter, and the high spatial scale is obtained by filtering a second image with a high-pass filter. The final hybrid image is composed by adding these two filtered images.

Combining two images:

Low resolution image- the one to be seen at a far distance

High resolution image- the one to be seen at a close distance

[Hybrid images paper](#)