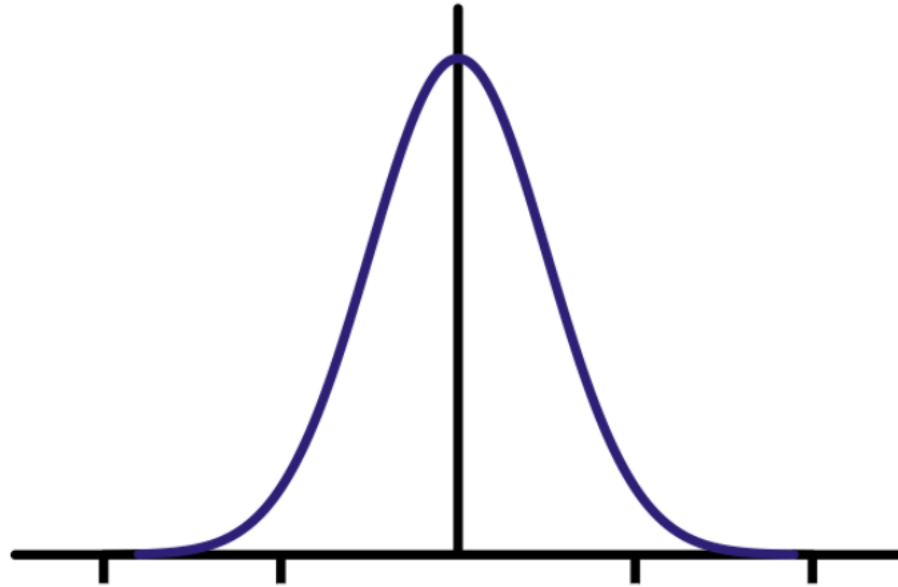


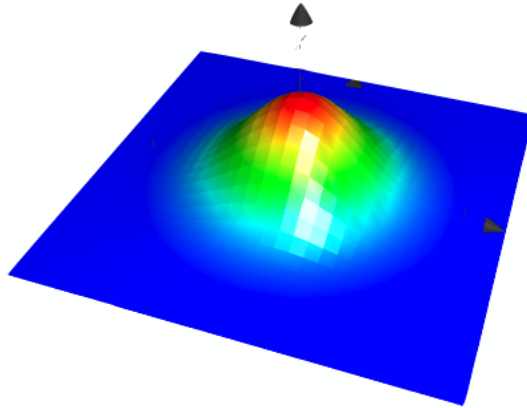
# **Lecture 5**

## **Transforms of 2D Signals**



Gaussian Filter

# The Gaussian filter



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

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

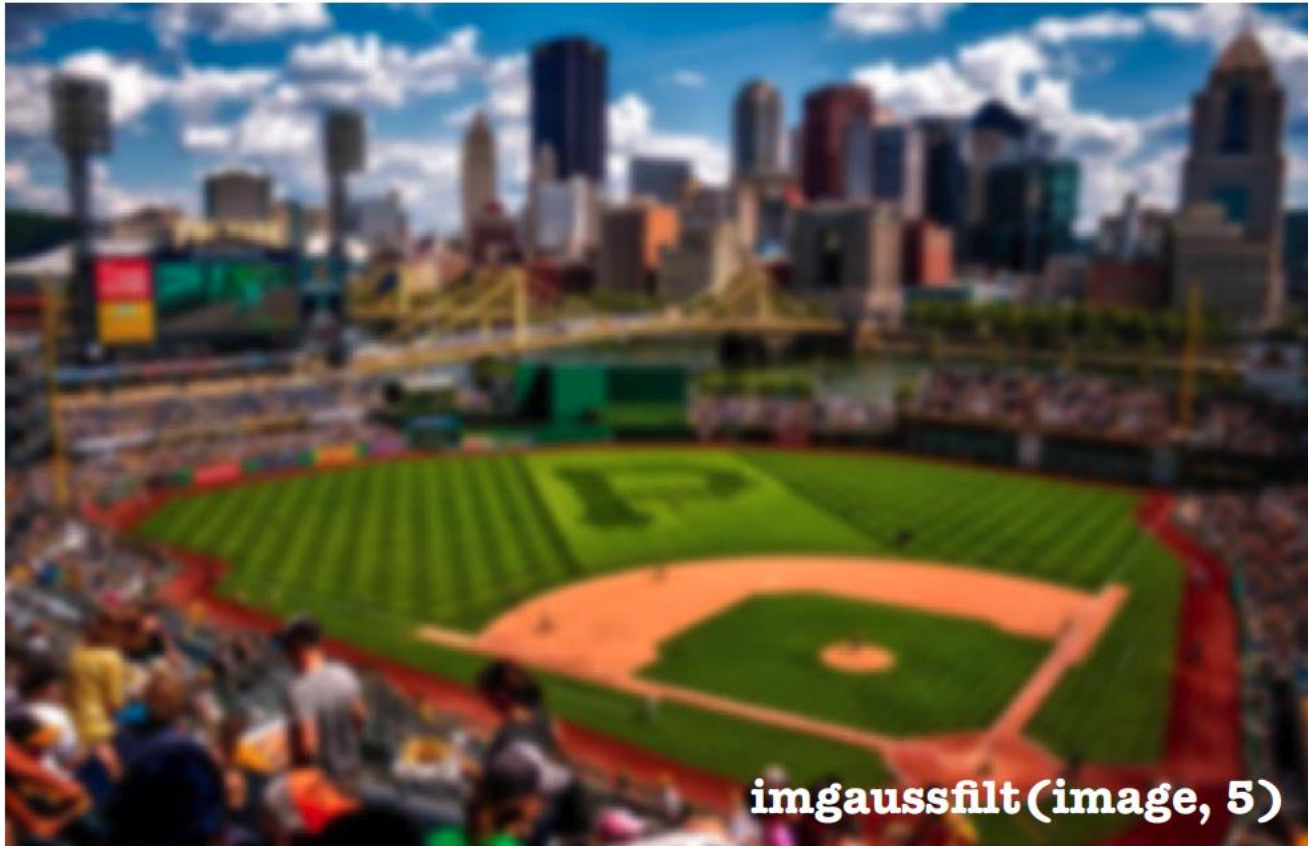
$\frac{1}{16}$	1	2	1
	2	4	2
	1	2	1

This 3 x 3 kernel is an approximation of a 2D Gaussian function

## Gaussian Filtering versus Box Filter

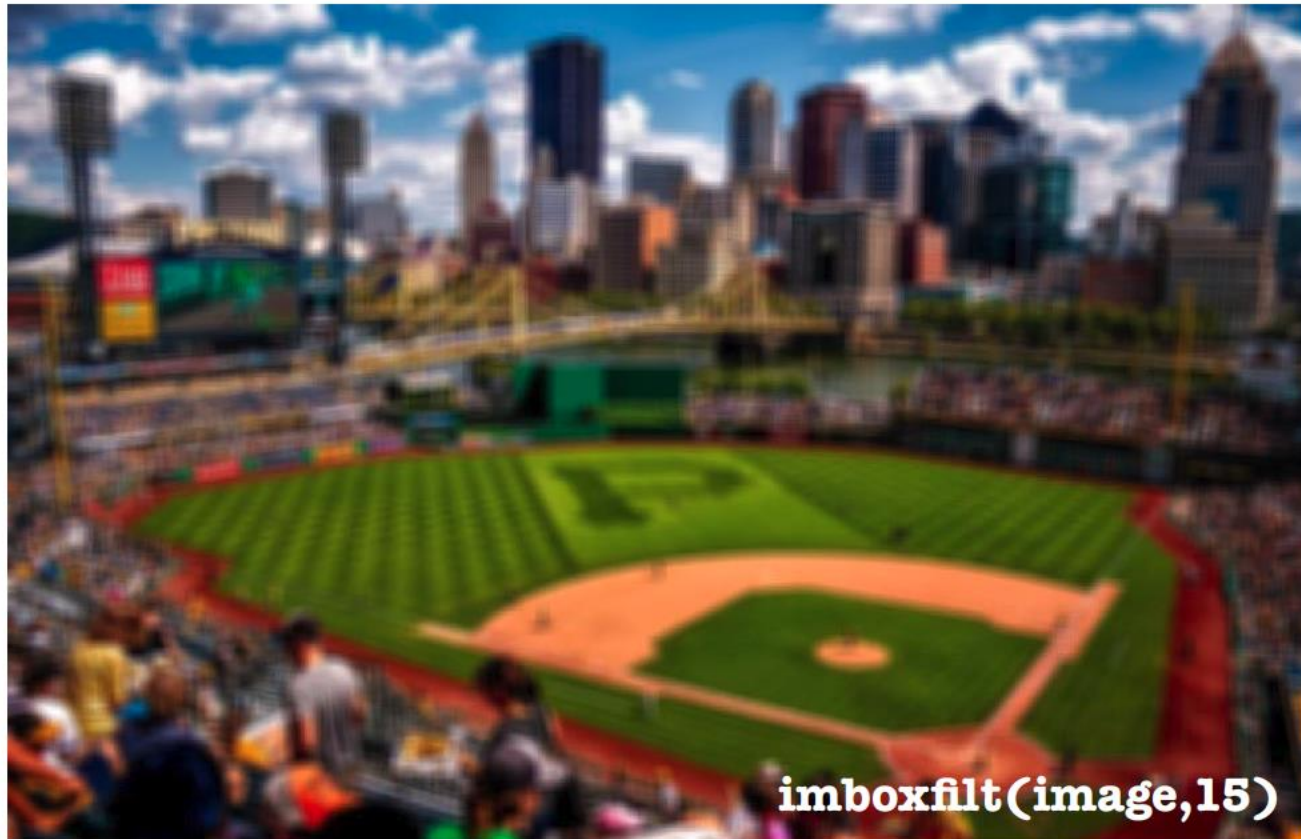


## Gaussian Filtering versus Box Filter





## Gaussian Filtering versus Box Filter



How would you create a shadow effect?

**UARK**



**UARK**

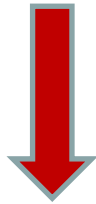
How would you create a shadow effect?

**UARK**



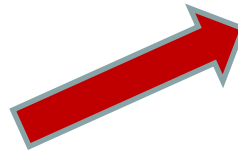
**UARK**

Overlay



**UARK**

Gaussian Blur





*How would you create a soft focus effect?*



*How would you create a soft focus effect?*



# Tilt Shift Effect



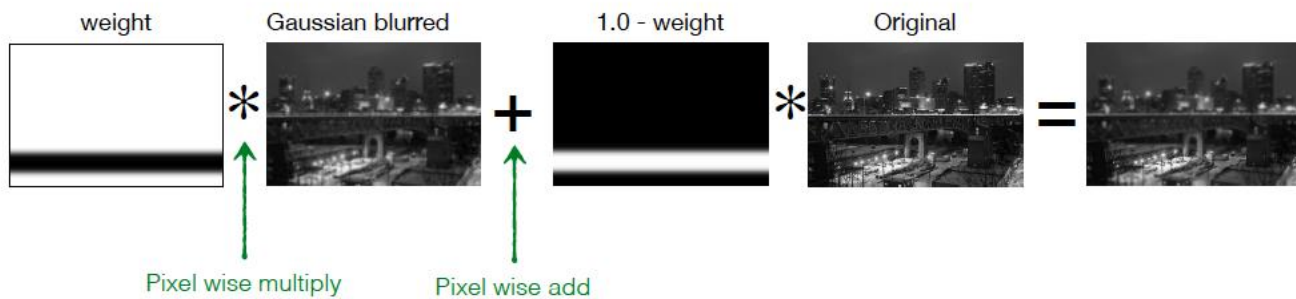
<http://www.flickr.com/photos/ender079/2704450659/>

*How would you create a (super low-budget) tilt-shift effect?*



[http://farm8.staticflickr.com/7061/6867631897\\_8377709b9\\_z.jpg](http://farm8.staticflickr.com/7061/6867631897_8377709b9_z.jpg)

*How would you create a (super low-budget) tilt-shift effect?*





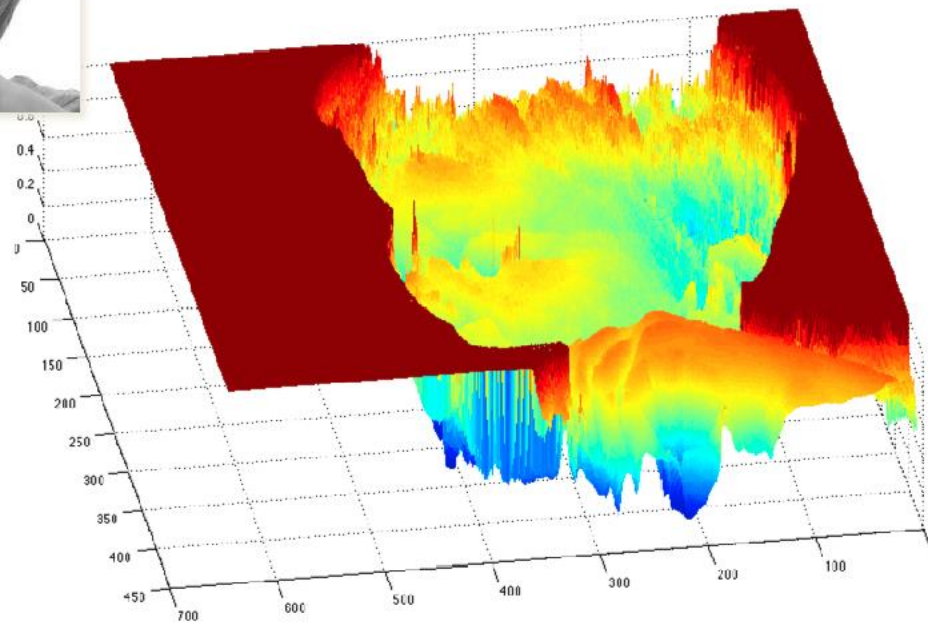
*Tell me everything wrong with this wannabe tilt-shift image*

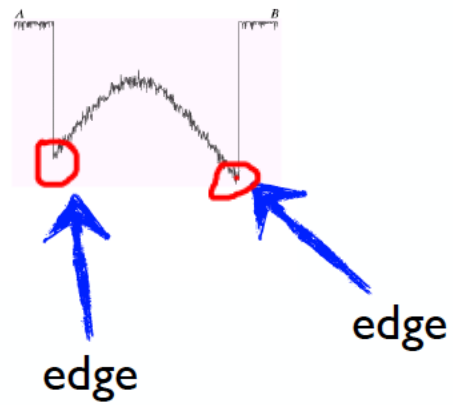
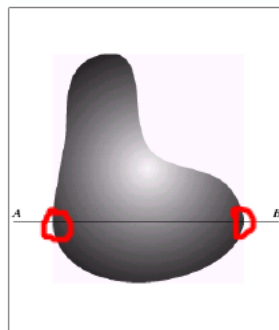


Recall that an image is a 2D function



$$f(x)$$





*How would you detect an edge?*  
*What kinds of filter would you use?*

*Do you remember this from high school?*

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

***Do you remember this from high school?***

The derivative of a function  $f$  at a point  $x$  is defined by the limit

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

Approximation of the derivative when  $h$  is small

This definition is based on the ‘forward difference’ but ...

... it turns out that using the 'central difference'  
is more accurate

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

How do we compute the derivative of a discrete signal?

10	20	10	200	210	250	250
----	----	----	-----	-----	-----	-----

↑

... it turns out that using the 'central difference'  
is more accurate

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

How do we compute the derivative of a discrete signal?

10	20	10	200	210	250	250
----	----	----	-----	-----	-----	-----



$$f'(x) = \frac{f(x+1) - f(x-1)}{2} = \frac{210 - 10}{2} = 100$$

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

1D derivative filter



# The 'Sobel' filter

$$\frac{1}{8} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

a derivative filter  
(with some smoothing)

*Filter returns large response on vertical or horizontal edges?*

# The 'Sobel' filter

$$\frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

a derivative filter  
(with some smoothing)

*Filter returns large response on vertical or horizontal edges?*

*Is the output always positive?*

# The 'Sobel' filter

$$\frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

a derivative filter  
(with some smoothing)

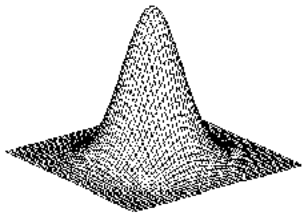
Responds to horizontal lines

Output can be positive or negative

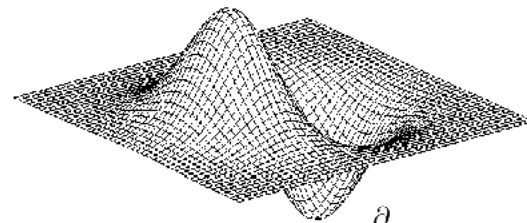
# The 'Sobel' filter

$$\frac{1}{8} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

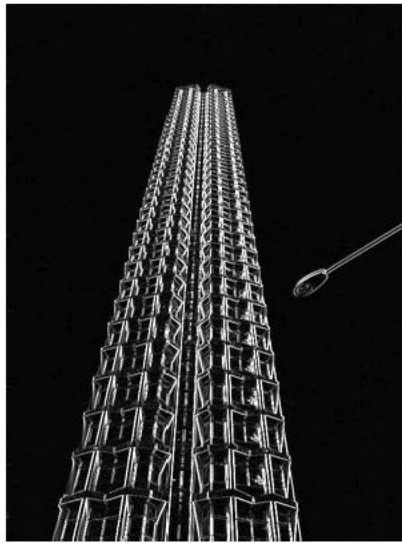
Approximation of the derivative of a Gaussian



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



Derivative of Gaussian  $\frac{\partial}{\partial x} h_{\sigma}(u, v)$



*Output of which Sobel filter?*



*Output of which Sobel filter?*

*How do you visualize negative derivatives/gradients?*

# The 'Sobel' filter

$$\frac{1}{8} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

*Where does this filter come?*



# Decomposing the Sobel filter

$$\frac{1}{8} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \frac{1}{8} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

Sobel                      What this?

# Decomposing the Sobel filter

$$\frac{1}{8} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = \frac{1}{8} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

Sobel

weighted average  
and scaling

The Sobel filter only returns the x and y edge responses.

*How can you compute the **image gradient**?*

How do you compute the image gradient?

Choose a derivative filter

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

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

*What is this filter called?*

Run filter over image

$$\frac{\partial f}{\partial x} = S_x \otimes f$$

$$\frac{\partial f}{\partial y} = S_y \otimes f$$

*What are the dimensions?*

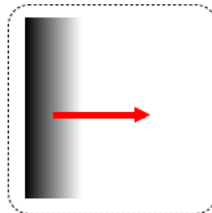
Image gradient

$$\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$

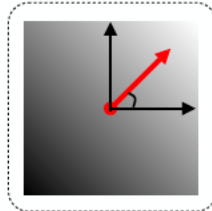
*What are the dimensions?*

# MATCH THAT GRADIENT !

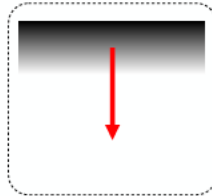
(a)



(b)



(c)



(1)

$$\nabla f = \left[ 0, \frac{\partial f}{\partial y} \right]$$

(2)

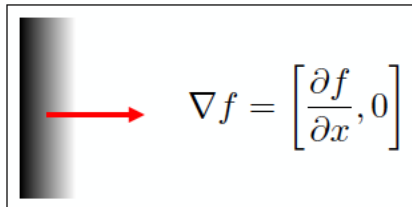
$$\nabla f = \left[ \frac{\partial f}{\partial x}, 0 \right]$$

(3)

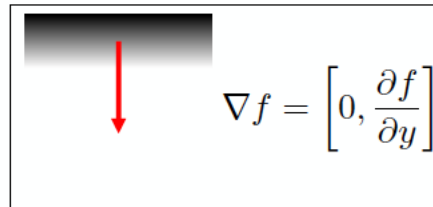
$$\nabla f = \left[ \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right]$$

# Image Gradient

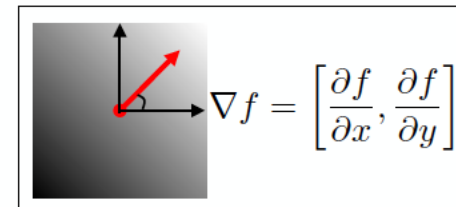
Gradient in x only



Gradient in y only



Gradient in both x and y



Gradient direction

?

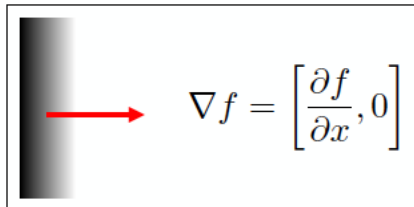
Gradient magnitude

?

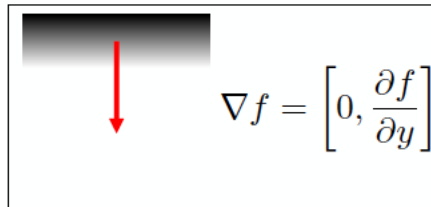


# Image Gradient

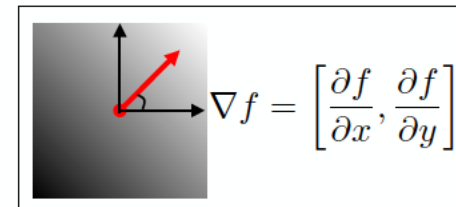
Gradient in x only



Gradient in y only



Gradient in both x and y



Gradient direction

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

*How does the gradient direction relate to the edge?*

Gradient magnitude

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

*What does a large magnitude look like in the image?*

## Common 'derivative' filters

Sobel

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

Scharr

3	0	-3	3	10	3
10	0	-10	0	0	0
3	0	-3	-3	-10	-3

Prewitt

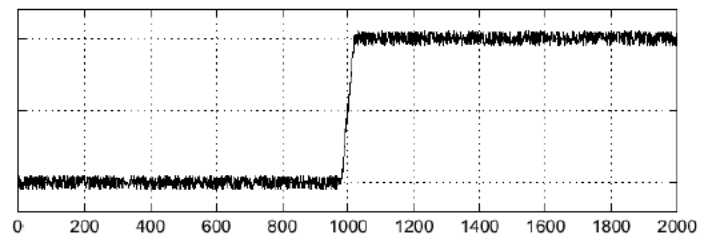
1	0	-1	1	1	1
1	0	-1	0	0	0
1	0	-1	-1	-1	-1

Roberts

0	1	1	0
-1	0	0	-1

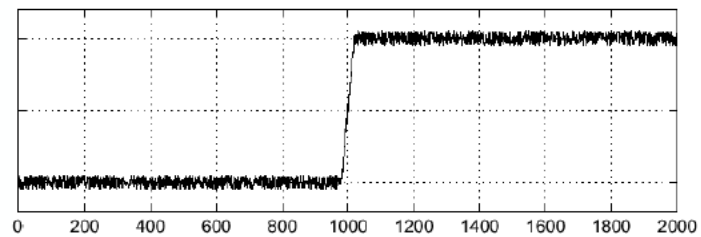
*How do you find the edge from this signal?*

Intensity plot



*How do you find the edge from this signal?*

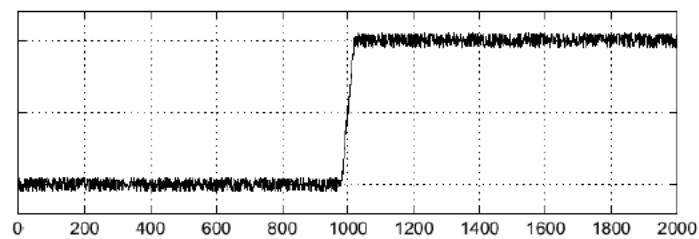
Intensity plot



*Use a derivative filter!*

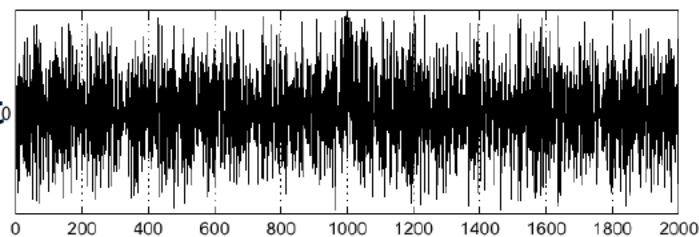
*How do you find the edge from this signal?*

Intensity plot



*Use a derivative filter!*

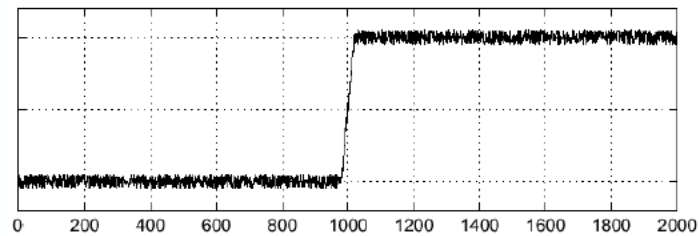
Derivative plot



*What happened?*

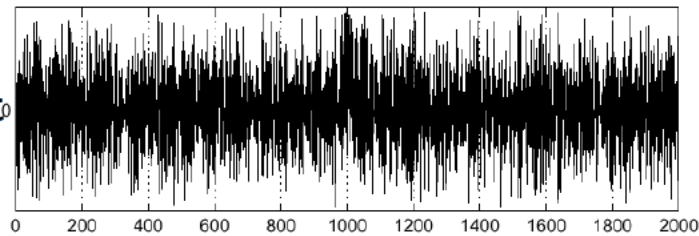
*How do you find the edge from this signal?*

Intensity plot

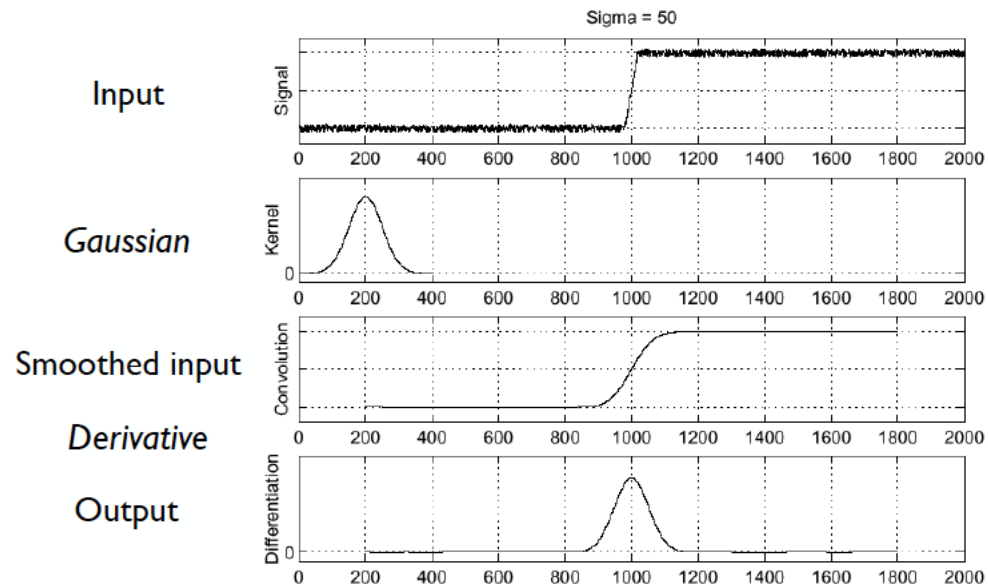


*Use a derivative filter!*

Derivative plot



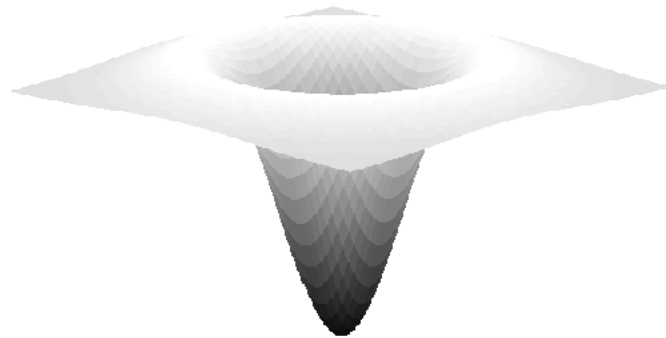
Derivative filters are sensitive to noise



*Don't forget to smooth before running derivative filters!*

# Laplace filter

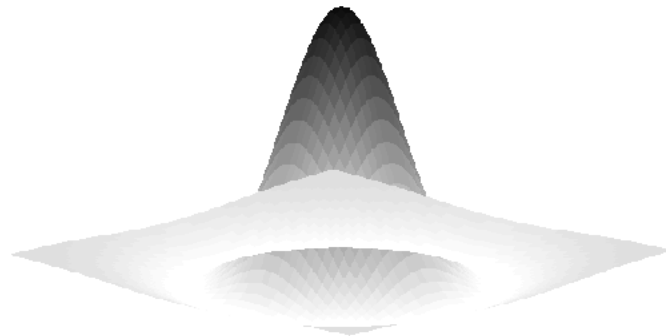
A.K.A. Laplacian, Laplacian of Gaussian (LoG), Marr filter, Mexican Hat Function





# Laplace filter

A.K.A. Laplacian, Laplacian of Gaussian (LoG), Marr filter, Mexican Hat Function



# Laplace filter

A.K.A. Laplacian, Laplacian of Gaussian (LoG), Marr filter, Mexican Hat Function



first-order  
finite difference

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

derivative filter

	0	-
--	---	---

second-order  
finite difference

$$f''(x) \approx \frac{\delta_h^2[f](x)}{h^2} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

Laplace filter

?	?	?
---	---	---

first-order  
finite difference

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

derivative filter

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

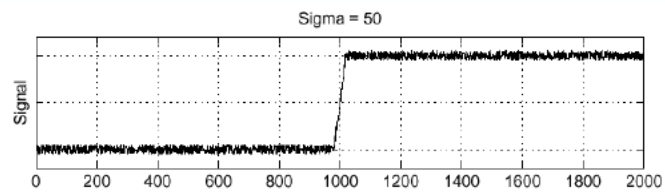
second-order  
finite difference

$$f''(x) \approx \frac{\delta_h^2[f](x)}{h^2} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}.$$

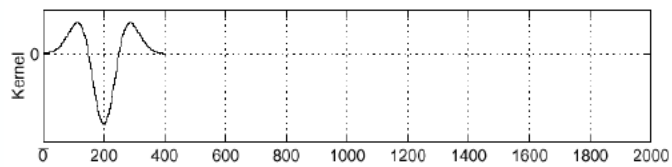
Laplace filter

1	-2	1
---	----	---

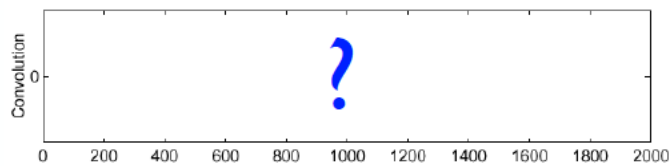
Input



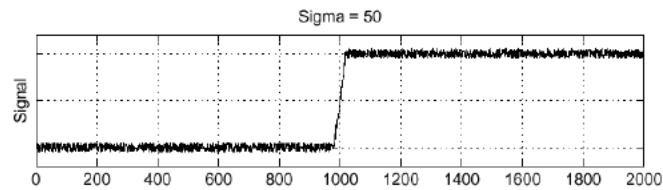
Laplacian



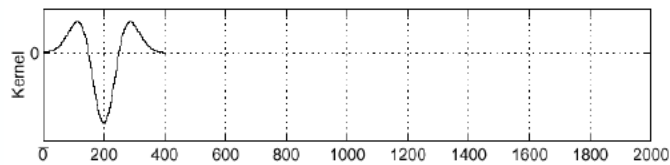
Output



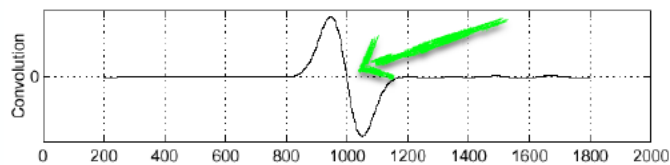
Input



Laplacian



Output



Zero crossings are more accurate at localizing edges  
Second derivative is noisy

## 2D Laplace filter

1	-2	1
---	----	---

1D Laplace filter

?	?	?
?	?	?
?	?	?

2D Laplace filter

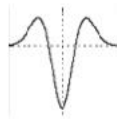
## 2D Laplace filter

1	-2	1
---	----	---

1D Laplace filter

?	?	?
?	?	?
?	?	?

2D Laplace filter



hint





## 2D Laplace filter

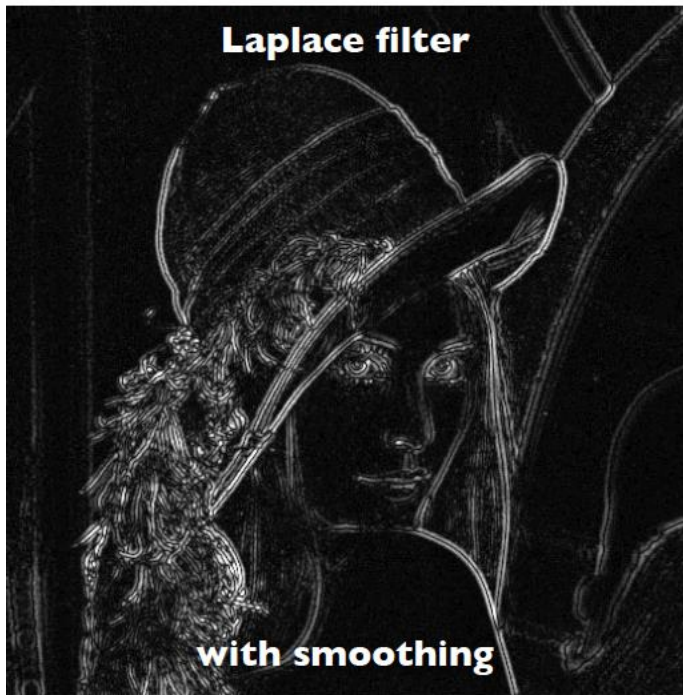
1	-2	1
---	----	---

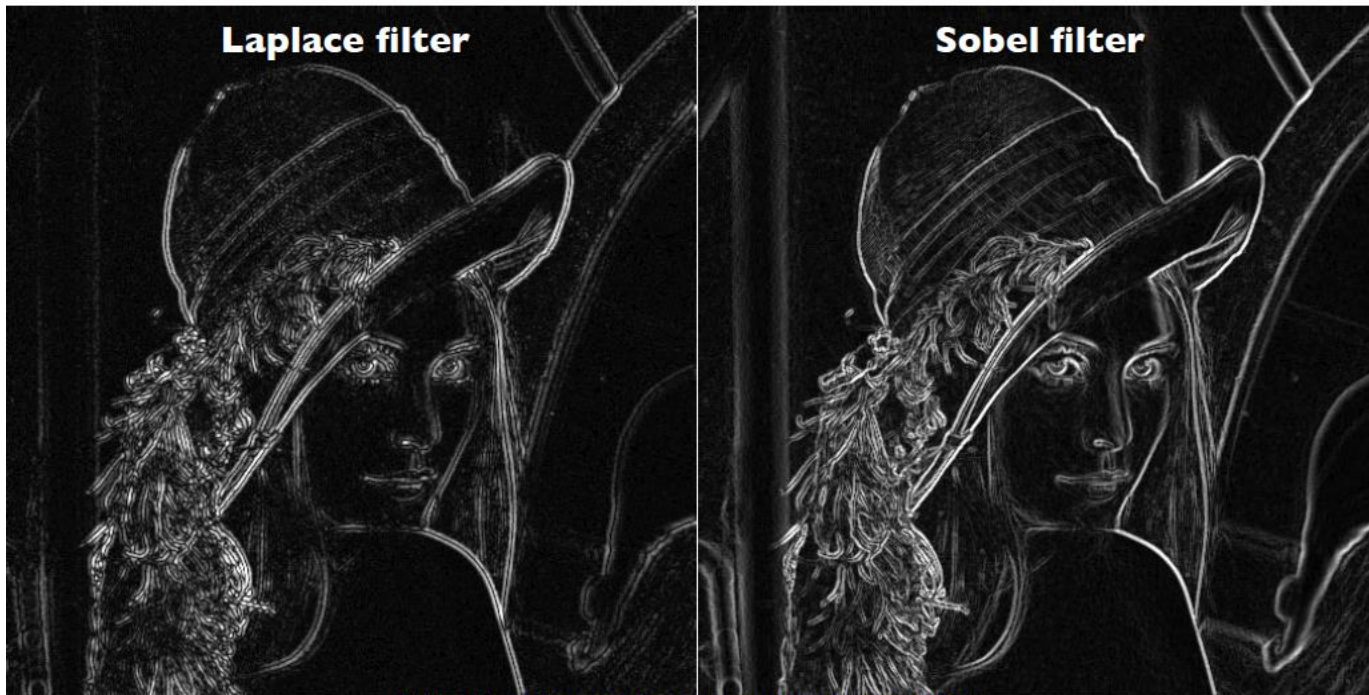
1D Laplace filter

0	1	0
1	-4	1
0	1	0

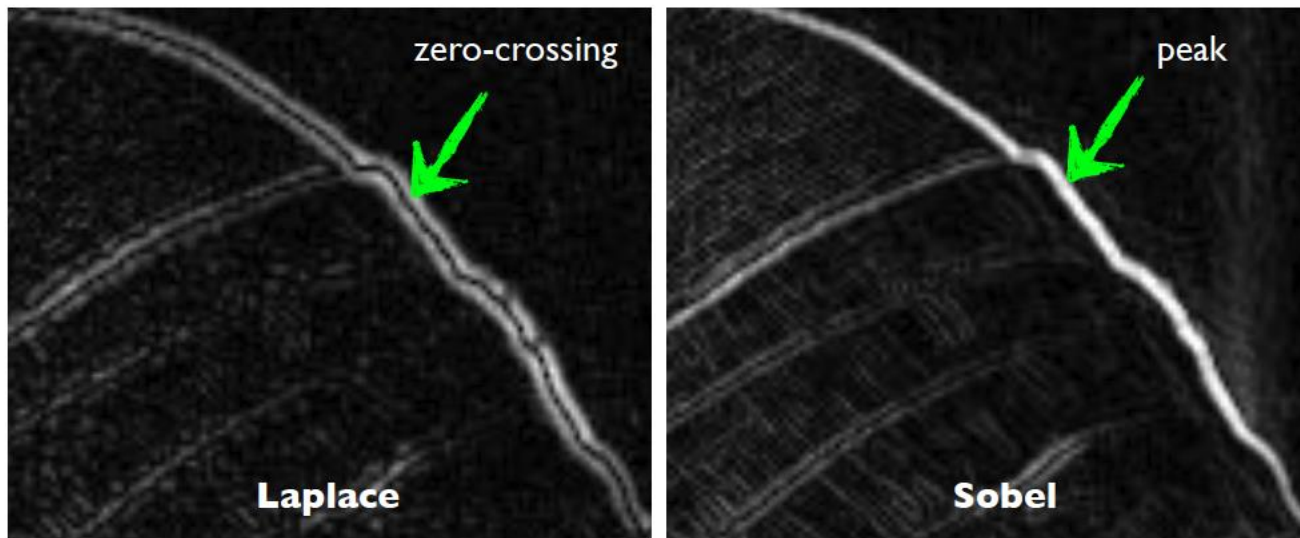
2D Laplace filter

*If the Sobel filter approximates the first derivative, the Laplace filter approximates ....?*

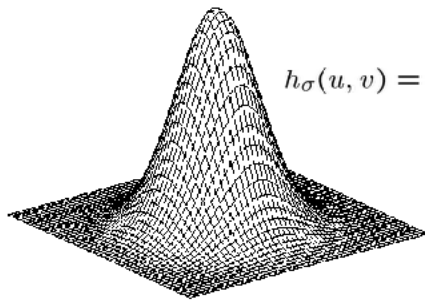




*What's different between the two results?*

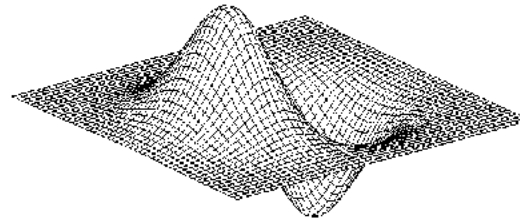


Zero crossings are more accurate at localizing edges  
(but not very convenient)



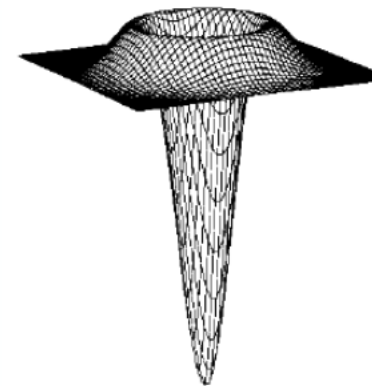
Gaussian

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



Derivative of Gaussian

$$\frac{\partial}{\partial x} h_{\sigma}(u, v)$$



Laplacian of Gaussian

$$\nabla^2 h_{\sigma}(u, v)$$

## 2D Gaussian Filters

# Differences Between 1D and Multi-Dimensional Signal Processing

- Much more data for M-D signal processing
  - 1) 1-D speech  $\implies$  10K samples/sec
  - 2) M-D television  $\implies$  500x500 pixels/frame, 30 frames/sec, hence 7.5 Mega samples/sec.

This is why we need to learn about transforms
- Mathematics for M-D is not as complete as 1-D
  - 1) 1-D systems are described by diff. eqs, M-D by PDEs.
  - 2) Fundamental results of algebra do not hold in M-D, e.g. factorization of polynomials in 1-D is known but not in M-D.
  - 3) This effects filter design, filter stability, reconstruction, etc...

# Signals, Systems and Transforms

Digital Signal Processing  $\Rightarrow$  Digital Image Processing

An image is a signal in 2-D or  $m$ -D

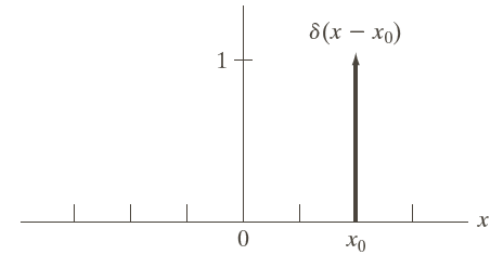
Signal carries physical information like speech, music, images

- continuous  $x(t)$
- discrete  $x[n]$

- Transform: input into output  $\Rightarrow$  **System**  $x[n] \longrightarrow \boxed{T} \longrightarrow y[n]$
- Linear system  $T(ax_1[n] + bx_2[n]) = T(ax_1[n]) + T(bx_2[n]) = ay_1[n] + by_2[n]$
- Time (shift) invariant system  
if  $T(x[n]) = y[n]$  then  $T(x[n-n_0]) = y[n-n_0]$
- Linear Time Invariant system (LTI): **Impulse response can entirely characterize an LTI system.**
- Impulse response and convolution
- Separable LTI system: computational efficiency
- Bounded input bounded output (BIBO) stability
- Finite Impulse Response (FIR): **always stable**
- Infinite Impulse Response (IIR): **not always stable**



Special signal: Impulse  $\delta(x) = \begin{cases} 1 & x = 0 \\ 0 & x \neq 0 \end{cases}$



$\delta[n] \longrightarrow \boxed{\text{LTI}} \longrightarrow h[n]$  Impulse response

**Theorem:** For a LTI system, if  $h[n] = \mathcal{T}(\delta[n])$  then

$$y[n] = \mathcal{T}(x[n]) = h[n] * x[n] \quad \text{Where } * \text{ denotes "convolution"}$$

<https://en.wikipedia.org/wiki/Convolution>

Given a LTI system  $\mathcal{T}$ , we can easily obtain output signal for *all possible* input signals  $x[n]$  from the Impulse response! This is a powerful result.

$$\int_{-\infty}^{\infty} \delta(x) d(x) = 1$$

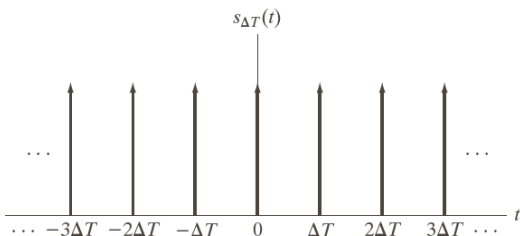
$$\sum_{-\infty}^{\infty} \delta[n] = 1$$

$$\int_{-\infty}^{\infty} f(x) \delta(x) d(x) = f(0)$$

$$\sum_{-\infty}^{\infty} f[n] \delta[n] = f[0]$$

$$\int_{-\infty}^{\infty} f(x) \delta(x - x_0) d(x) = f(x_0)$$

$$\sum_{-\infty}^{\infty} f[n] \delta[n - n_0] = f[n_0]$$



$$s_n(x) = \sum_{-\infty}^{\infty} \delta(x - n) \quad \text{impulse train}$$

$$x[n] = \sum_k x[k] \delta[n - k]$$



Filters we have learned so far ...

The 'Box' filter

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

1	1	1
1	1	1
1	1	1

Gaussian filter

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

1	2	1
2	4	2
1	2	1

Sobel filter

$$\frac{1}{8}$$

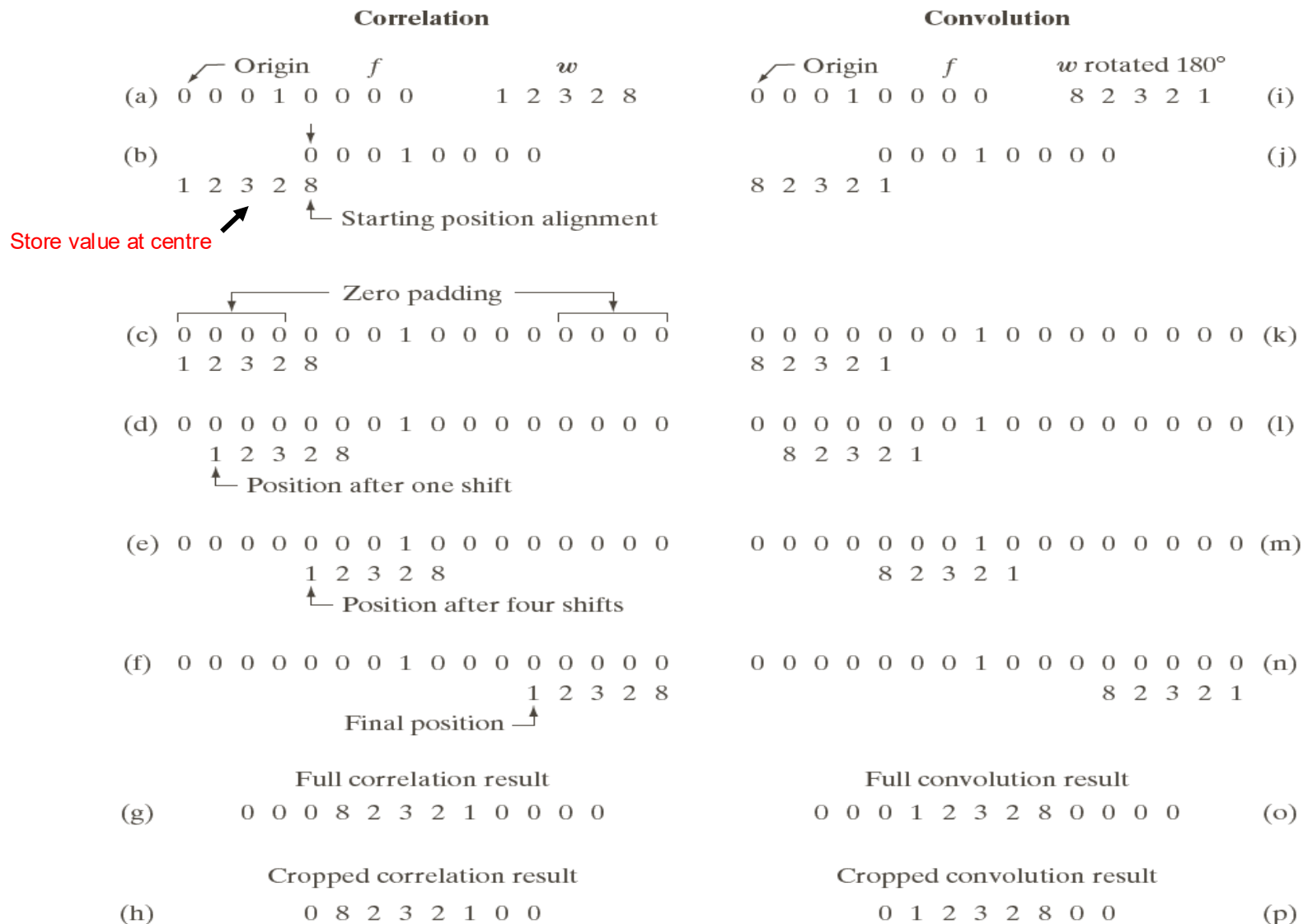
1	0	-1
2	0	-2
1	0	-1

Laplace filter

$$\frac{1}{8}$$

0	1	0
1	-4	1
0	1	0

# Computation of Correlation and Convolution (1-D)



**FIGURE 3.29** Illustration of 1-D correlation and convolution of a filter with a discrete unit impulse. Note that correlation and convolution are functions of *displacement*.

# Filtering vs Convolution

filtering  
(cross-correlation)

$$h = g \otimes f$$

$$\overset{\text{output}}{h[m, n]} = \sum_{k, l} \overset{\text{filter}}{g[k, l]} \overset{\text{image}}{f[m + k, n + l]}$$

*What's the  
difference?*

convolution

$$h = g \star f$$

$$h[m, n] = \sum_{k, j} g[k, l] f[m - k, n - l]$$

# Filtering vs Convolution

filtering  
(cross-correlation)

$$h = g \otimes f$$

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

filter flipped vertically  
and horizontally

convolution

$$h = g \star f$$

$$h[m, n] = \sum_{k, j} g[k, l] f[m - k, n - l]$$

# Filtering vs Convolution

filtering  
(cross-correlation)

$$h = g \otimes f$$

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

filter flipped vertically  
and horizontally

convolution

$$h = g \star f$$

$$h[m, n] = \sum_{k, j} g[k, l] f[m - k, n - l]$$

*Suppose  $g$  is a Gaussian filter.  
How does convolution differ from filtering?*

Recall...

1	2	1
2	4	2
1	2	1

$\frac{1}{16}$

**Commutative**

“can move stuff around”

$$a \star b = b \star a .$$

**Associative**

“can regroup things”

$$(((a \star b_1) \star b_2) \star b_3) = a \star (b_1 \star b_2 \star b_3)$$

**Distributes over addition**

“can take things through parenthesis”

$$a \star (b + c) = (a \star b) + (a \star c)$$

**Scalars factor out**

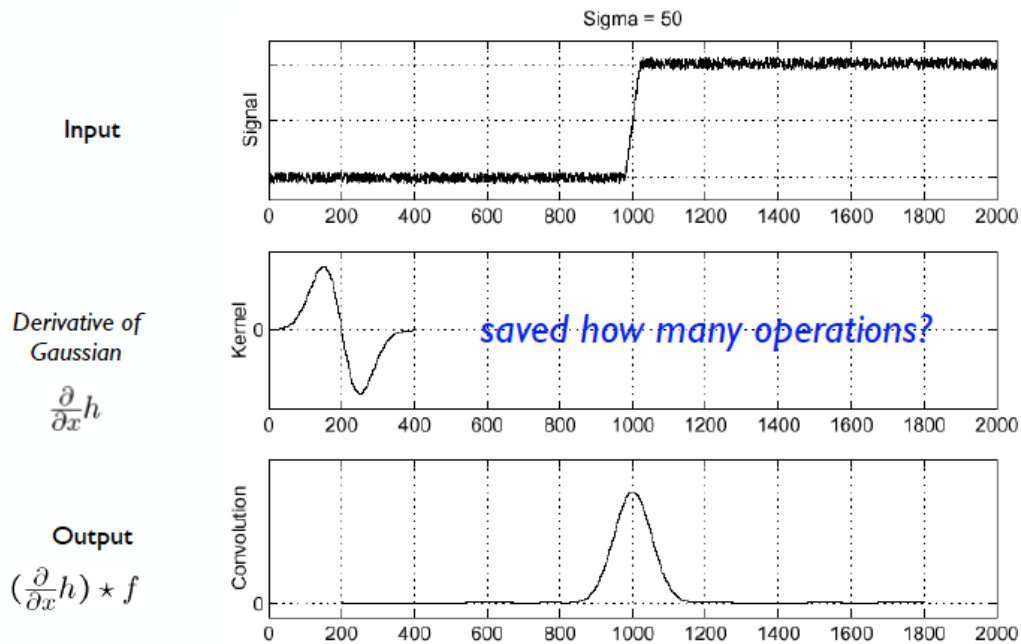
$$\lambda a \star b = a \star \lambda b = \lambda(a \star b)$$

**Derivative Theorem of Convolution**

$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

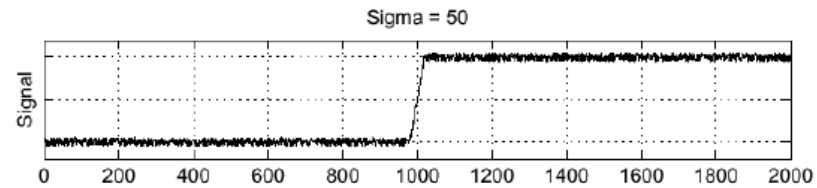
can precompute this

$$\text{Derivative Theorem of Convolution} \quad \frac{\partial}{\partial x}(h \star f) = \left(\frac{\partial}{\partial x}h\right) \star f$$

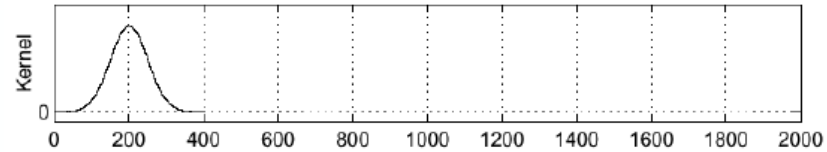


Recall ...

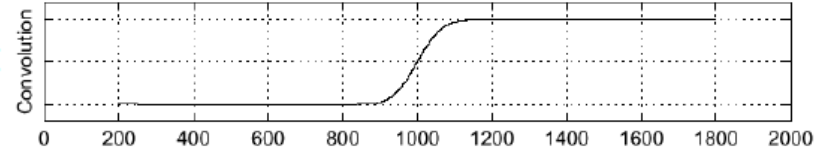
Input



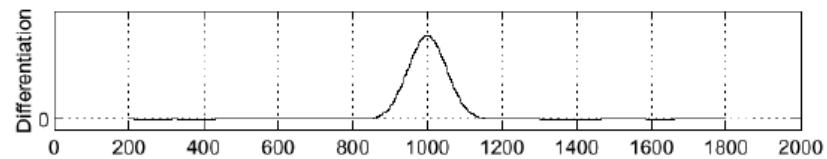
*Gaussian*



Smoothed input



*Derivative*



Output



# Continuous Fourier Transform

## 1-D and continuous

$$F(\omega) = \int_{-\infty}^{\infty} f(x) e^{-j2\pi\omega x} dx$$

$$f(x) = \int_{-\infty}^{\infty} F(\omega) e^{j2\pi\omega x} d\omega$$

$$F(\omega) = \int_{-\infty}^{\infty} \delta(x) e^{-j2\pi\omega x} dx = e^{-j2\pi\omega \cdot 0} = 1$$

$$F(\omega) = \int_{-\infty}^{\infty} \delta(x - x_0) e^{-j2\pi\omega x} dx = e^{-j2\pi\omega \cdot x_0} = \cos(2\pi\omega \cdot x_0) - j \sin(2\pi\omega \cdot x_0)$$

$$H(\omega) = F[f(x - \alpha)] = F(\omega) e^{-j2\pi\omega \cdot \alpha} \quad \text{where } F(\omega) = F[f(x)]$$

<http://www.cs.cornell.edu/courses/cs5540/2010sp/lectures/Lec5.Transforms.pdf>

# Convolution Theorem

*Definition :*  $f(x) * h(x) = \int_{-\infty}^{\infty} f(t)h(x-t)dt$  continuous

$$\begin{aligned} F[f(x) * h(x)] &= \int_{-\infty}^{\infty} \left[ \int_{-\infty}^{\infty} f(t)h(x-t)dt \right] e^{-j2\pi\omega x} dx \\ &= \int_{-\infty}^{\infty} f(t) \underbrace{\left[ \int_{-\infty}^{\infty} h(x-t) e^{-j2\pi\omega x} dx \right]}_{F[h(x-t)]} dt \\ &\quad F[h(x-t)] = H(\omega) e^{-j2\pi\omega.t} \end{aligned}$$

$$\begin{aligned} F[f(x) * h(x)] &= \int_{-\infty}^{\infty} f(t) \left[ H(\omega) e^{-j2\pi\omega.t} \right] dt \\ &= H(\omega) \int_{-\infty}^{\infty} f(t) e^{-j2\pi\omega.t} dt \\ &= H(\omega).F(\omega) \end{aligned}$$

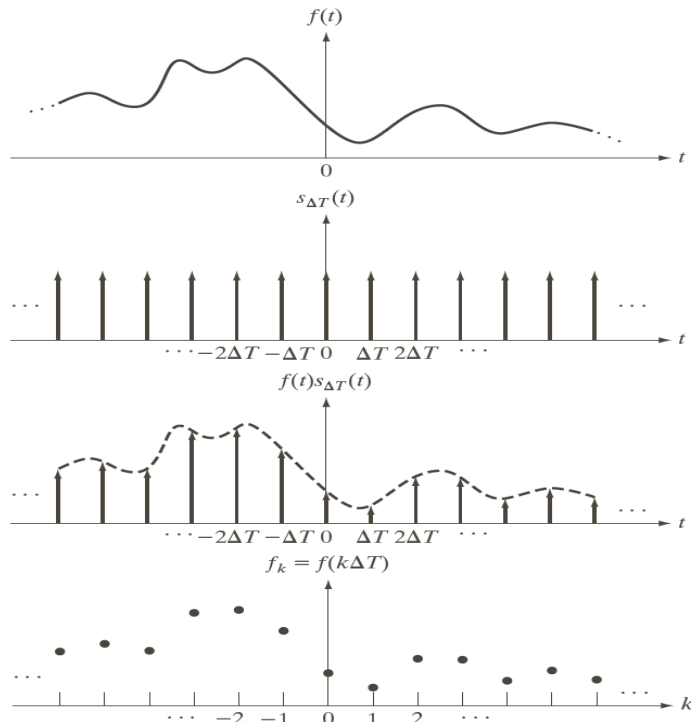
**Theorem:** The FT of the convolution of two functions is equal to the product of the FTs of the two functions.

**Reverse:** The inverse FT of  $H(\omega).F(\omega)$  gives the convolution of the two functions  $f(x) * h(x)$  in spatial domain.

**In General:**  $f(x) * h(x) \Leftrightarrow F(\omega).H(\omega)$

$f(x).h(x) \Leftrightarrow F(\omega) * H(\omega)$

## Discrete Fourier Transform (1-D)



a  
b  
c  
d

**FIGURE 4.5**

(a) A continuous function. (b) Train of impulses used to model the sampling process. (c) Sampled function formed as the product of (a) and (b). (d) Sample values obtained by integration and using the sifting property of the impulse. (The dashed line in (c) is shown for reference. It is not part of the data.)

# Discrete Fourier Transform (1-D)

Let us define a set of orthonormal basis vectors

$$\frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \quad \frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ e^{j\frac{2\pi}{N} \cdot 1} \\ e^{j\frac{2\pi}{N} \cdot 2} \\ \vdots \\ e^{j\frac{2\pi}{N} \cdot (N-1)} \end{bmatrix}, \quad \frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ e^{j\frac{2\pi}{N} \cdot 2 \cdot 1} \\ e^{j\frac{2\pi}{N} \cdot 2 \cdot 2} \\ \vdots \\ e^{j\frac{2\pi}{N} \cdot 2 \cdot (N-1)} \end{bmatrix}, \quad \dots \quad \frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ e^{j\frac{2\pi}{N} \cdot (N-1) \cdot 1} \\ e^{j\frac{2\pi}{N} \cdot (N-1) \cdot 2} \\ \vdots \\ e^{j\frac{2\pi}{N} \cdot (N-1) \cdot (N-1)} \end{bmatrix}$$

$b_0 \qquad \qquad \qquad b_1 \qquad \qquad \qquad b_2 \qquad \dots \qquad \qquad \qquad b_{N-1}$

Any vector  $\vec{x}$  can be written as :

$$\vec{x} = \sum_{k=0}^{N-1} X_k \vec{b}_k \quad \text{or}$$

$$x[n] = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X_k e^{j\frac{2\pi}{N}kn}$$

$$X_k = \frac{1}{\sqrt{N}} \sum_n x[n] e^{-j\frac{2\pi}{N}kn}$$

Discrete Fourier Transform pair

# Discrete Fourier Transform (1-D)

DTFT :  $x[n]$  is discrete,  $X(\omega)$  is continuous

$$X(\omega) = \sum_{-\infty}^{\infty} x[n] e^{-j\omega n}$$

$\omega$  can take any value

$$x[n] = \frac{1}{2\pi} \sum_{-\pi}^{\pi} X(\omega) e^{j\omega n}$$

## DFT

$$X[k] = \sum_{n=0}^{N-1} x[n] e^{-j\frac{2\pi}{N}kn}$$

$$\omega = \frac{2\pi k}{N}$$

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] e^{j\frac{2\pi}{N}kn}$$

## 2-D D.F.T

$$D.F.T \{x(n_1, n_2)\} \triangleq X(k_1, k_2)$$

$$X(k_1) = \begin{cases} \sum_{n_1=0}^{N_1-1} x(n_1) e^{-j \frac{2\pi n_1 k_1}{N_1}} & 0 \leq k_1 < N_1 \\ 0 & \text{Otherwise} \end{cases}$$

$$I.D.F.T \{X(k_1, k_2)\} \triangleq x(n_1, n_2)$$

$$x(n_1, n_2) = \begin{cases} \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X(k_1, k_2) e^{j \frac{2\pi k_1 n_1}{N_1}} e^{j \frac{2\pi k_2 n_2}{N_2}} & 0 \leq n_1 < N_1 \\ & 0 \leq n_2 < N_2 \\ 0 & \text{Otherwise} \end{cases}$$

## 2-D D.F.T

$$D.F.T \{x(n_1, n_2)\} \triangleq X(k_1, k_2)$$

$$X(k_1, k_2) = \begin{cases} \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\frac{2\pi n_1 k_1}{N_1}} e^{-j\frac{2\pi n_2 k_2}{N_2}} & 0 \leq k_1 < N_1 \\ & 0 \leq k_2 < N_2 \\ 0 & \text{Otherwise} \end{cases}$$

$$I.D.F.T \{X(k_1, k_2)\} \triangleq x(n_1, n_2)$$

$$x(n_1, n_2) = \begin{cases} \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X(k_1, k_2) e^{j\frac{2\pi k_1 n_1}{N_1}} e^{j\frac{2\pi k_2 n_2}{N_2}} & 0 \leq n_1 < N_1 \\ & 0 \leq n_2 < N_2 \\ 0 & \text{Otherwise} \end{cases}$$

# Computation of 2D-DFT

- Fourier transform matrix:

$$\mathbf{F}_N = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_N & W_N^2 & \dots & W_N^{N-1} \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ \cdot & \cdot & \cdot & & \cdot \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \dots & W_N^{(N-1)^2} \end{bmatrix}$$

where  $W_N = e^{-j2\pi/N}$



# Computation of 2D-DFT

- Inverse Fourier transform matrix:

$$\mathbf{F}_N^{-1} = \frac{1}{N} \mathbf{F}_N^*$$

where

$$\mathbf{F}_N^* = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & W_N^{-1} & W_N^{-2} & \dots & W_N^{1-N} \\ \vdots & \vdots & \vdots & & \vdots \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & W_N^{1-N} & W_N^{2(1-N)} & \dots & W_N^{-(N-1)^2} \end{bmatrix}$$

# Computation of 2D-DFT

- Example.  $N = 4$ :

$$\mathbf{F}_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \quad \mathbf{F}_4^* = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & j & -1 & -j \\ 1 & -1 & 1 & -1 \\ 1 & -j & -1 & j \end{bmatrix}$$

- Matlab function: **dfmtx**

# Computation of 2D-DFT

- 1D-DFT (of 1D signal  $\mathbf{x}$ ):

$$\tilde{\mathbf{x}} = \mathbf{F}_N \mathbf{x}$$

- Inverse 1D-DFT:

$$\mathbf{x} = \frac{1}{N} \mathbf{F}_N^* \tilde{\mathbf{x}}$$

- 2D-DFT (of 2D image  $\mathbf{X}$ ):

$$\tilde{\mathbf{X}} = \mathbf{F}_N \mathbf{X} \mathbf{F}_N$$

- Inverse 2D-DFT:

$$\mathbf{X} = \frac{1}{N^2} \mathbf{F}_N^* \tilde{\mathbf{X}} \mathbf{F}_N^*$$

# Computation of 2D-DFT

- Verify with Matlab:

```
clear
```

```
A = dftmtx(4);
```

```
I = [1 2 3 4; 0 0 5 5; -2 -2 4 4; 5 4 2 1];
```

```
Adft = A*I*A
```

```
Afft = fft2(I)
```

```
Aidft = 1/16 * conj(A)*Adft*conj(A)
```

```
Aifft = ifft2(Afft)
```

# How to compute 2D D.F.T ?

1. Direct Method
2. Row/Column Decomposition
3. Vector-Radix (Tree) : FFT

## Factors effecting speed:

1. Number of arithmetic ops.
2. Complexity of program: memory usage
3. Language and platform

**Direct Method:** Assume  $N_1 = N_2 = N$ , for each  $(k_1, k_2)$  we need  $N^2$  multiplications and  $N^2$  adds.

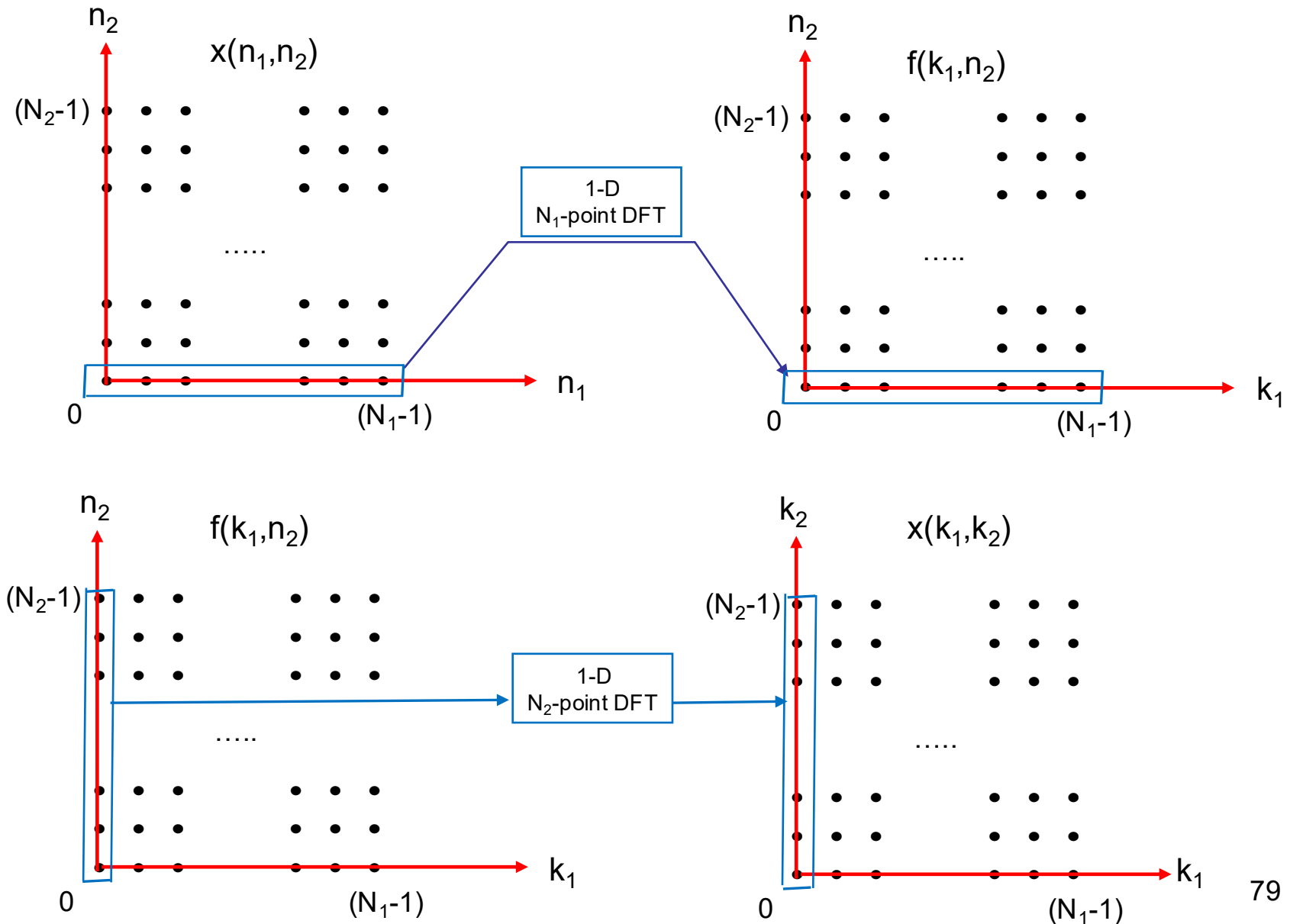
How many  $(k_1, k_2)$  ? Ans.  $N^2 \implies N^4$  mults. ,  $N^4$  adds.

- Row/Column Decomposition:

$$X(k_1, k_2) = \sum_{n_2=0}^{N-1} \underbrace{\left( \sum_{n_1=0}^{N-1} x(n_1, n_2) e^{-j \frac{2\pi n_1 k_1}{N}} \right)}_{f(k_1, n_2)} e^{-j \frac{2\pi n_2 k_2}{N}}$$

- There are  $N$  1-D DFTs of rows, each 1-D DFT is  $N$ -points
- For each 1-D DFT we can use FFT using  $N \log_2 N$  mults and  $(N/2) \log_2 N$  adds. For  $N$  rows:  $N^2 \log_2 N$  mults.
- For  $N$  columns: the method requires  $2N^2 \log_2 N$  mults. A lot less than Direct Method.
- For  $N=1000$ , direct method needs  $N^{12}$  ops. Row/Col method requires  $N^6 \times 10 \approx N^7$ . The difference is between 30 hrs and 1 sec!

# How to compute 2D D.F.T ?



# How to compute 2D D.F.T ?

	Number of multiplications
	$(N_1 = N_2 = 512)$
Direct computation	$N_1^2 N_2^2$ (100%)
Row/Column decomposition with Direct 1-D computation	$N_1 N_2 (N_1 + N_2)$ (0.4%)
Row/Column decomposition with 1-D FFT computation	$\frac{N_1 N_2}{2} \log_2(N_1 N_2)$ (0.0035%)

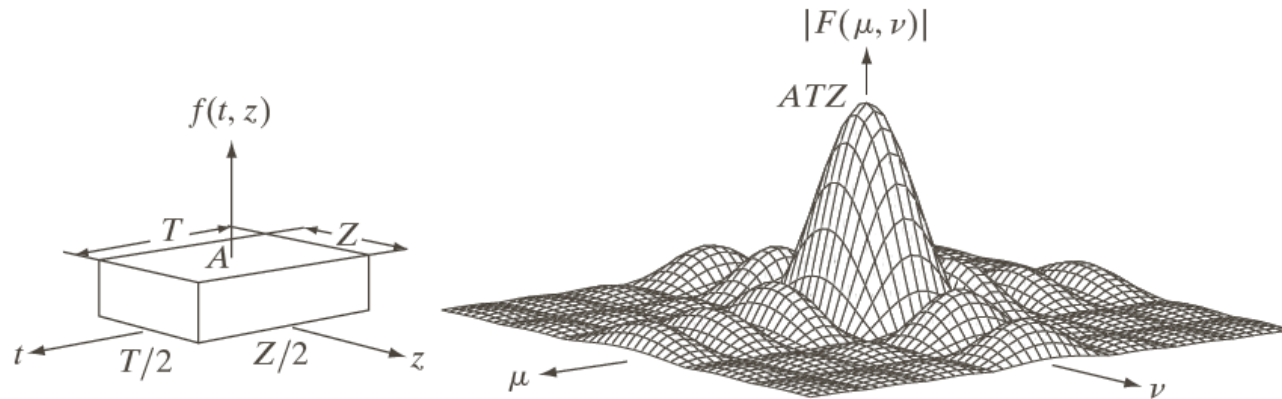


- Row/Col decomposition requires matrix transpose.
- Note: in the computer we don't store the 2D matrices but we do row by row. Hence we need special method for matrix transpose.
- Aklundhs algorithm for transposition:
  - recursive
  - fast, non I/O intensive for matrix transpose
  - based on divide-and-conquer
- Basis idea:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} A_{11}^T & A_{21}^T \\ A_{12}^T & A_{22}^T \end{bmatrix}$$

- Total I/O cost:  $2N \log_2 N$

# 2-D Fourier Transform



a b

**FIGURE 4.13** (a) A 2-D function, and (b) a section of its spectrum (not to scale). The block is longer along the  $t$ -axis, so the spectrum is more “contracted” along the  $\mu$ -axis. Compare with Fig. 4.4.

Do example 4.1 of the Text

# Systems and Transforms (2-D)

- Transform (Operator): input into output  $T[X(n_1, n_2)] = Y(n_1, n_2)$
- Linear System (Operator)

$$T[aX_1(n_1, n_2) + bX_2(n_1, n_2)] = aT[X_1(n_1, n_2)] + bT[X_2(n_1, n_2)]$$

- Time Invariant System *If*  $T[X(n_1, n_2)] = Y(n_1, n_2)$

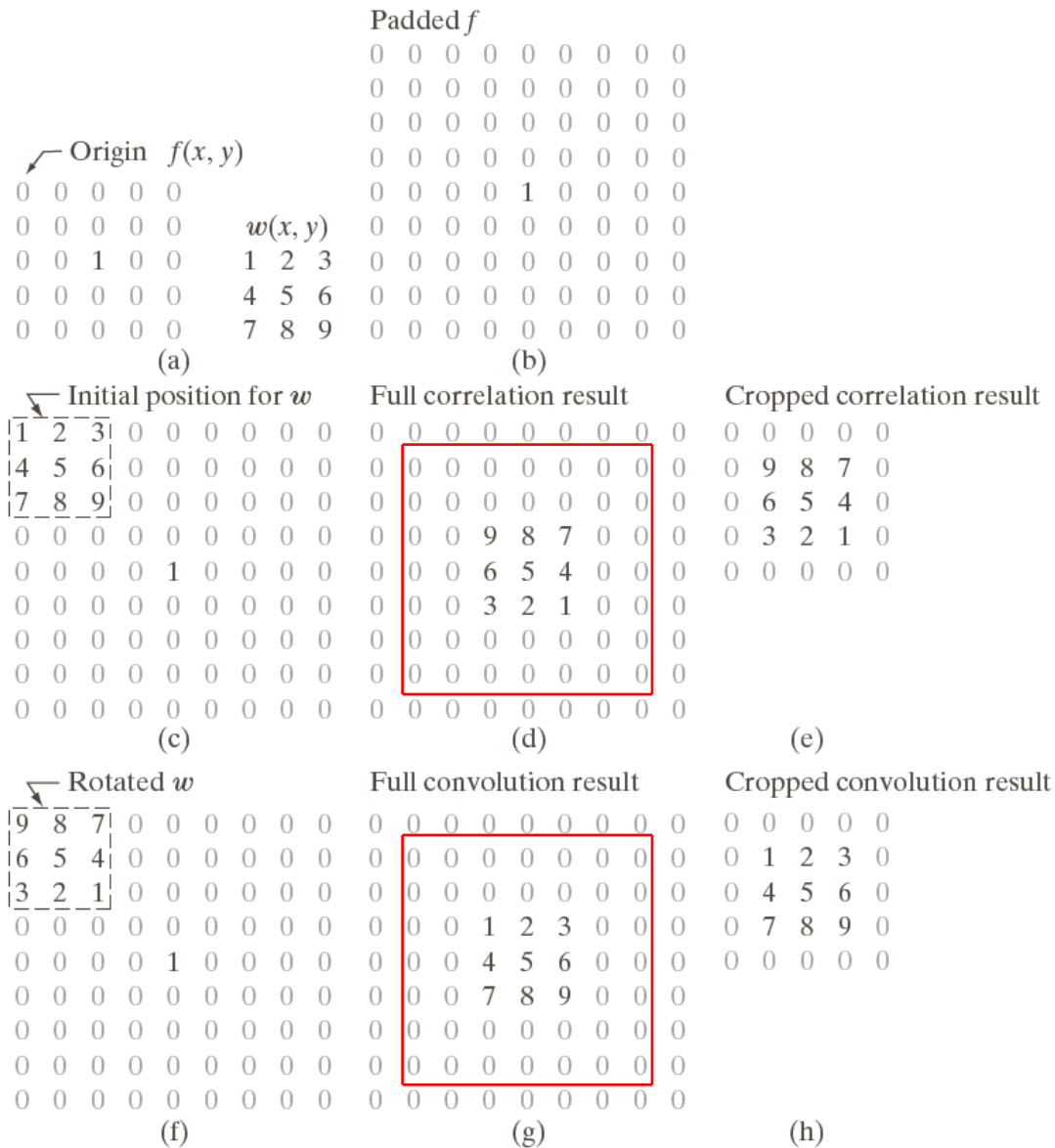
$$\text{then } T[X(n_1 - k_1, n_2 - k_2)] = Y(n_1 - k_1, n_2 - k_2)$$

- Impulse Response

$$\begin{aligned} Y(n_1, n_2) &= T[X(n_1, n_2)] = T\left[\sum_{k_1} \sum_{k_2} X(k_1, k_2) \delta(n_1 - k_1, n_2 - k_2)\right] \\ &= \sum_{k_1} \sum_{k_2} X(k_1, k_2) T[\delta(n_1 - k_1, n_2 - k_2)] \\ Y(n_1, n_2) &= \sum_{k_1} \sum_{k_2} X(k_1, k_2) h(n_1 - k_1, n_2 - k_2) \quad \text{(2-D convolution)} \end{aligned}$$

- LTI system can be completely characterized by its response to the impulse and its shift.

# Computation of Correlation and Convolution (2-D)



# LTI Systems Are Utmost Important

An  $N \times N$  image convolves with an  $M \times M$  filter (impulse response) requires:

$$(N + M - 1)^2 M^2 \text{ for } N \gg M: \approx N^2 M^2$$

$X(n_1, n_2)$ is $N \times N$ $h(n_1, n_2)$ is $M \times M$
--

Separable LTI gives efficient computations:  $h(n_1, n_2) = h_1(n_1) h_2(n_2)$

$$\begin{aligned} Y(n_1, n_2) &= T[X(n_1, n_2)] = \sum_{k_1} \sum_{k_2} X(k_1, k_2) h_1(n_1 - k_1) h_2(n_2 - k_2) \\ &= \sum_{k_1} h_1(n_1 - k_1) \sum_{k_2} X(k_1, k_2) h_2(n_2 - k_2) \end{aligned}$$

$$f(k_1, n_2) = \sum_{k_2} X(k_1, k_2) h_2(n_2 - k_2) = X(k_1) * h_2 \quad \Rightarrow \quad 1\text{-D}$$

$$Y(n_1, n_2) = \sum_{k_1} h_1(n_1 - k_1) f(k_1, n_2) \quad \Rightarrow \quad 1\text{-D}$$

Computational costs  $\approx 2N^2 M$

# DTFT: How Does It Work?

- Recall that FT uses complex exponentials (sinusoids) as building blocks.
$$e^{j\omega t} = \cos(\omega t) + j \sin(\omega t)$$
- For each frequency of complex exponential, the sinusoid at that frequency is compared to the signal.
- If the signal consists of that frequency, the correlation is high  $\rightarrow$  large FT coefficients.
- If the signal does not have any spectral component at a frequency, the correlation at that frequency is low or zero  $\rightarrow$  small or zero FT coefficient.

$$DTFT \text{ of } x(n_1, n_2): X(\omega_1, \omega_2) = \sum_{n_1} \sum_{n_2} x(n_1, n_2) e^{-j\omega_1 n_1} e^{-j\omega_2 n_2}$$

# Observations:

1.  $X(\omega_1, \omega_2)$  is a complex function, has real and imaginary parts
2. It is a continuous function of  $\omega_1, \omega_2$
3. If  $x(n_1, n_2)$  is separable so is its FT:

$$x(n_1, n_2) = x_1(n_1)x_2(n_2)$$

$$X(\omega_1, \omega_2) = X_1(\omega_1)X_2(\omega_2)$$

4. Inverse FT:

$$x(n_1, n_2) = \left(\frac{1}{2\pi}\right)^2 \int_{-\pi}^{\pi} \int_{-\pi}^{\pi} X(\omega_1, \omega_2) e^{j\omega_1 n_1} e^{j\omega_2 n_2} d\omega_1 d\omega_2$$

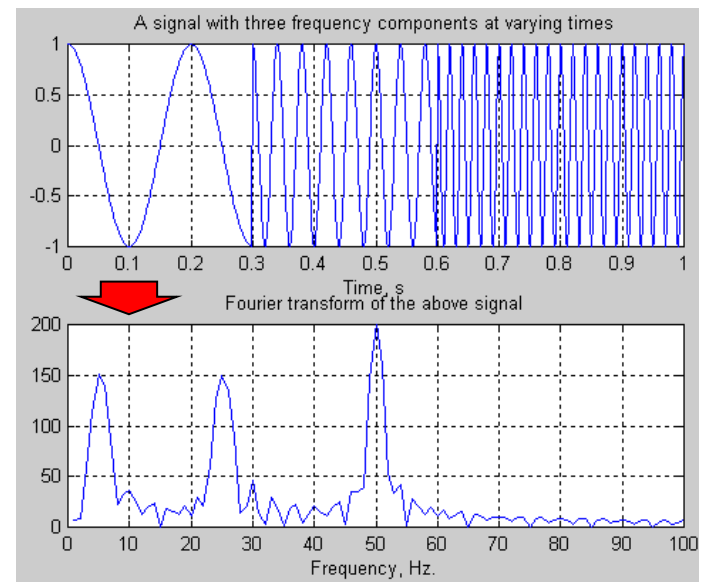
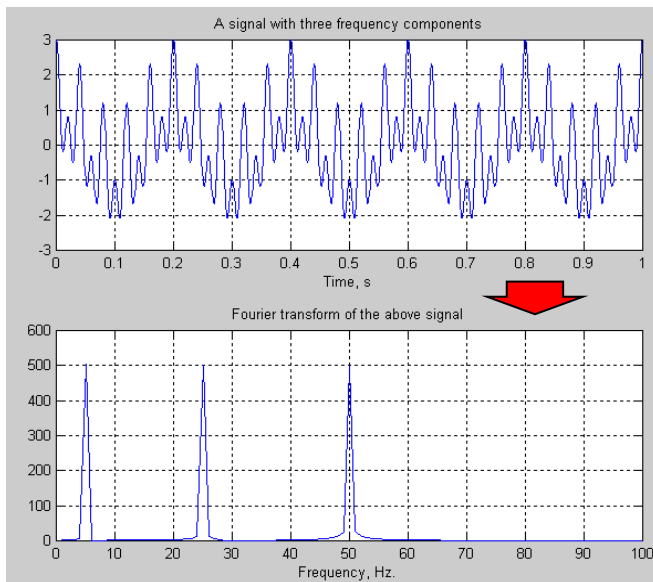
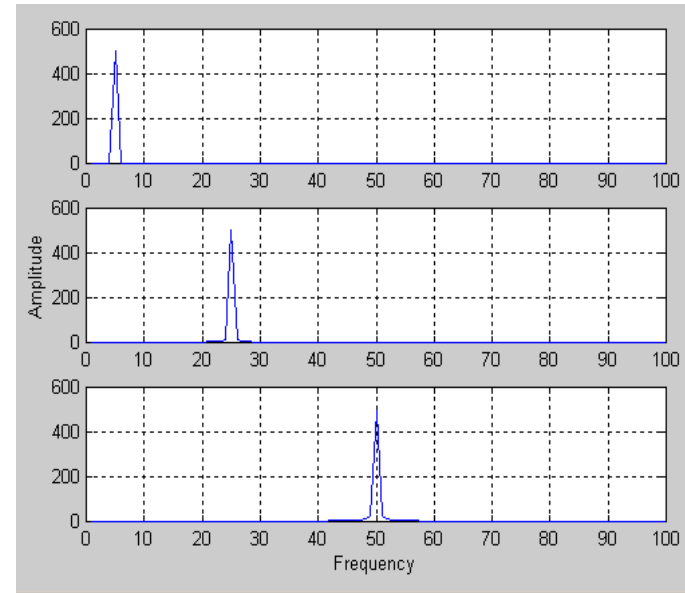
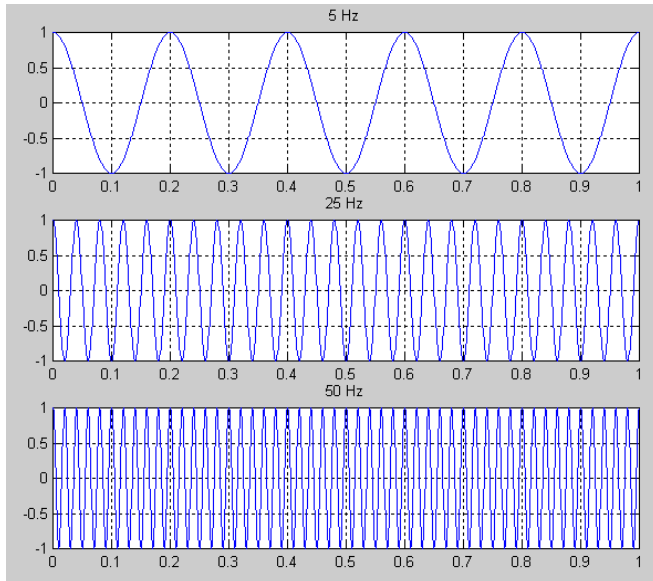
$$X(\omega_1, \omega_2) = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} x(s, t) e^{-i(\omega_1 s + \omega_2 t)} ds dt$$

# Properties of Fourier Transform

- Linearity
- Spatial shift
- Multiplication in time corresponds to convolution in frequency domain
- Differentiation in frequency domain corresponds to multiplicative by  $j$  in the space domain
- Transposition
- Reflection
- Complex conjugate
- Parseval's Theorem

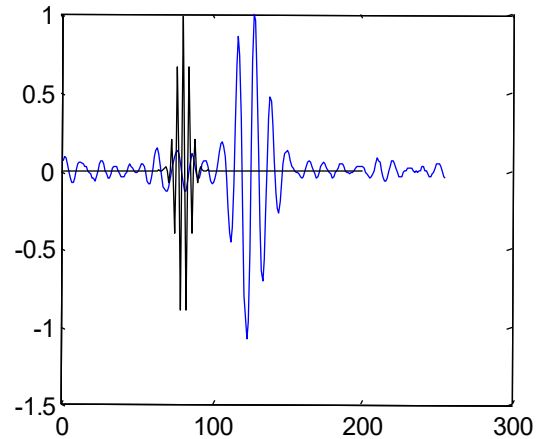
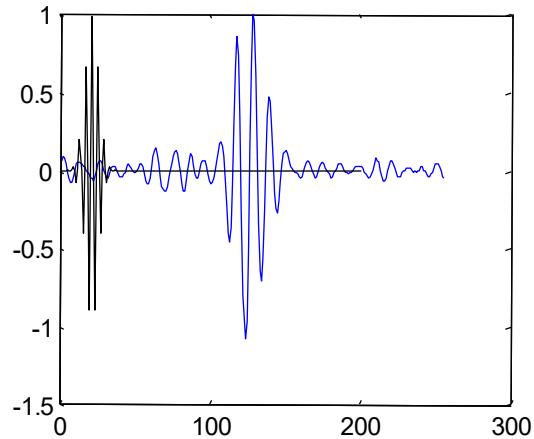


# Examples of Fourier Transform

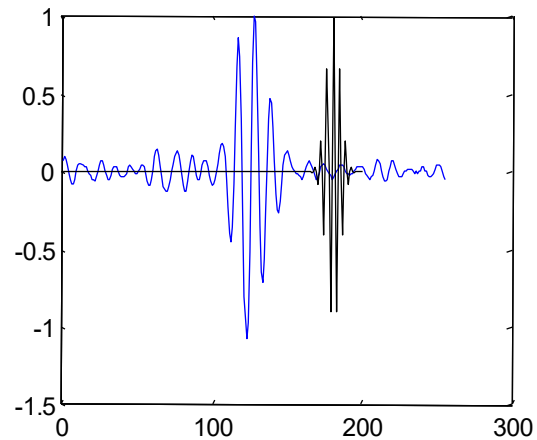
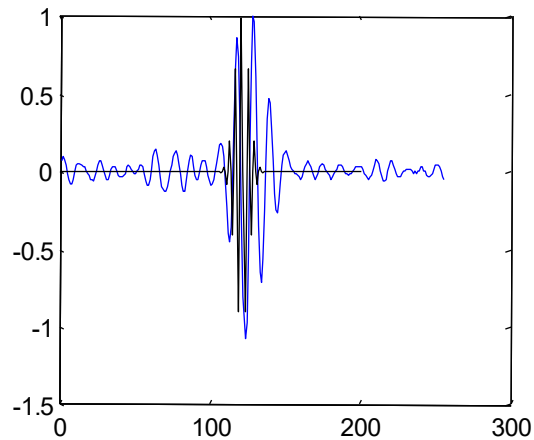


# Short Time Fourier Transform

$$STFT_f^\omega(t', \omega) = \int_t [f(t) \cdot W(t - t')] \cdot e^{-j\omega t} dt$$



*t - time*



# Selection of STFT Window

## Two extreme cases:

- $W(t)$  infinitely long: STFT becomes FT, providing good frequency resolution, but no time information
- $W(t)$  infinitely short:

$$W(t) = \delta(t)$$

$$STFT_x^\omega(t', \omega) = \int_t [x(t) \cdot \delta(t - t')] \cdot e^{-j\omega t} dt = x(t') \cdot e^{-j\omega t'}$$

STFT then gives the time signal back, with a phase factor.  
Excellent time information, but no frequency information.

*Wide window → poor time resolution, good frequency resolution*

*Narrow window → good time resolution, poor frequency resolution*

*Once the window is chosen, the resolution is set for both time and frequency.*

# Heisenberg Principle

$$\Delta t \cdot \Delta f \geq \frac{1}{4\pi}$$

**Both time and frequency resolutions cannot be arbitrarily high!** We cannot precisely know at what time instance a frequency component is located. We can only know what *interval of frequencies* are present in which *time intervals*

## Wavelet Transforms

Analysis windows of different lengths are used for different frequencies:

At high frequencies → Use narrower windows for better time resolution

At low frequencies → Use wider windows for better frequency resolution

Heisenberg principle holds!!!

The function used to window the signal is called ***the wavelet*** .

*An interesting article about wavelets:*

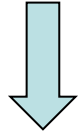
<http://www.bloomberg.com/news/articles/2016-08-29/want-a-hedge-fund-job-knowing-about-wavelets-improves-your-odds>

# Application: Tomography

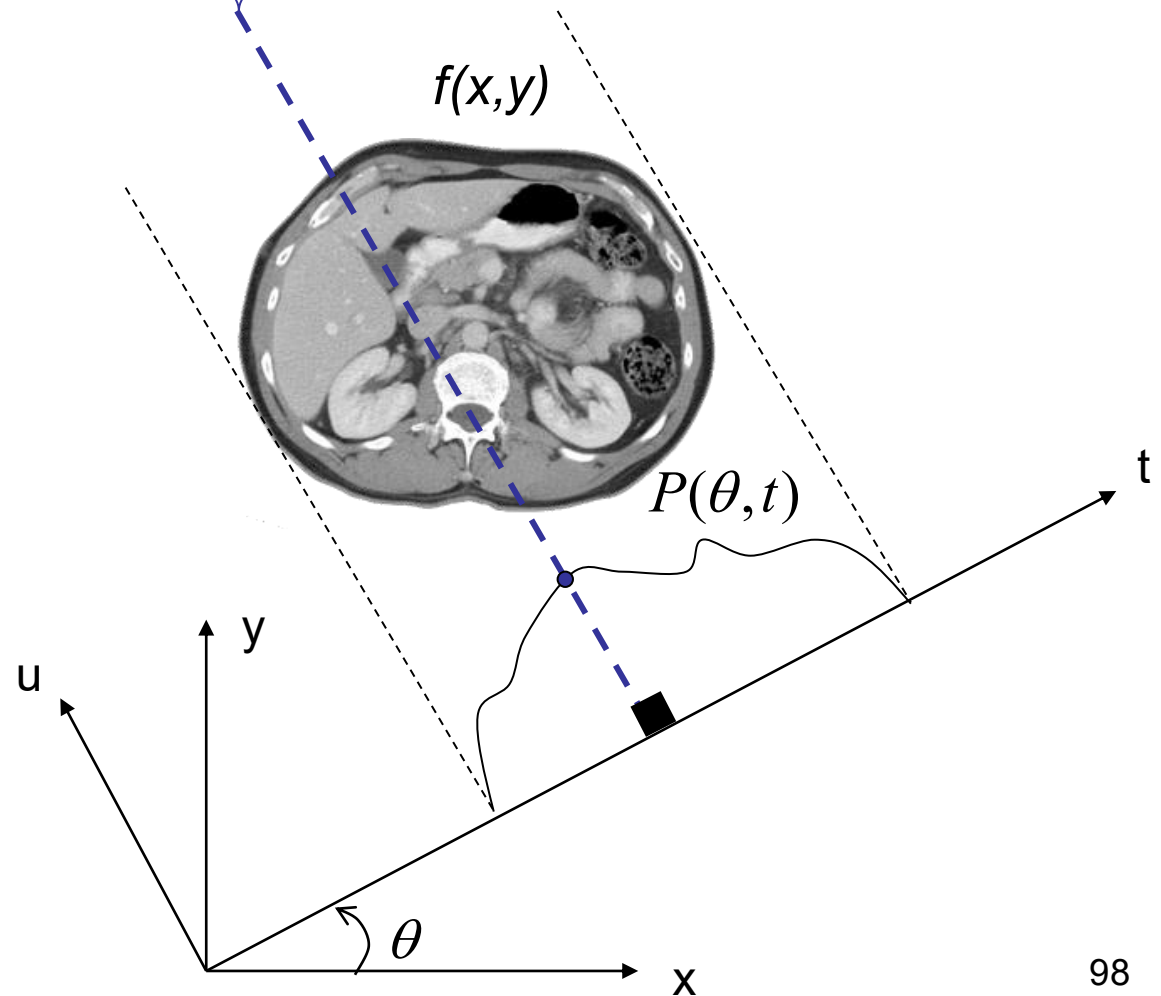
- An object in 3-D and a source like X-ray that gives projections in 2-D.
- Rotate the object (or patient) by an angle that gives another projection.
- $f(x, y)$  is the continuous function in 2-D; and  $P(\theta, t)$  is the projection of  $f(x, y)$  onto the line  $t$  that forms an angle  $\theta$  with the x-axis.
- Then  $x = t \cos \theta - u \sin \theta$  and  $y = t \sin \theta + u \cos \theta$  where  $t, u$  are the coordinates of the new coordinate system.
- $P(\theta, t)$  is actually the line integral along each projection line (perpendicular to the  $t$ -axis).

# Tomography: Imaging From Projections

$$P(\theta, t) = \iint f(x, y) \delta(x \cos \theta + y \sin \theta - t) dx dy$$



- Projection
- Raysum
- Line Integral
- Other names



# Radon Transform Property

- Inner-Product Slice Theorem\*

$$\int P(\theta, t)H(t)dt = \iint f(x, y)H(x \cos \theta + y \sin \theta)dxdy$$

- Proof: Plug in  $P(\theta, t) = \iint f(x, y)\delta(x \cos \theta + y \sin \theta - t)dxdy$

$$\begin{aligned}\int P(\theta, t)H(t)dt &= \int \left[ \iint f(x, y)\delta(x \cos \theta + y \sin \theta - t)dxdy \right] H(t)dt \\ &= \iint \left[ \int \delta(x \cos \theta + y \sin \theta - t)H(t)dt \right] f(x, y)dxdy \\ &= \iint f(x, y)H(x \cos \theta + y \sin \theta)dxdy\end{aligned}$$

\*H(t) must be square-integrable

# Famous Special Case:

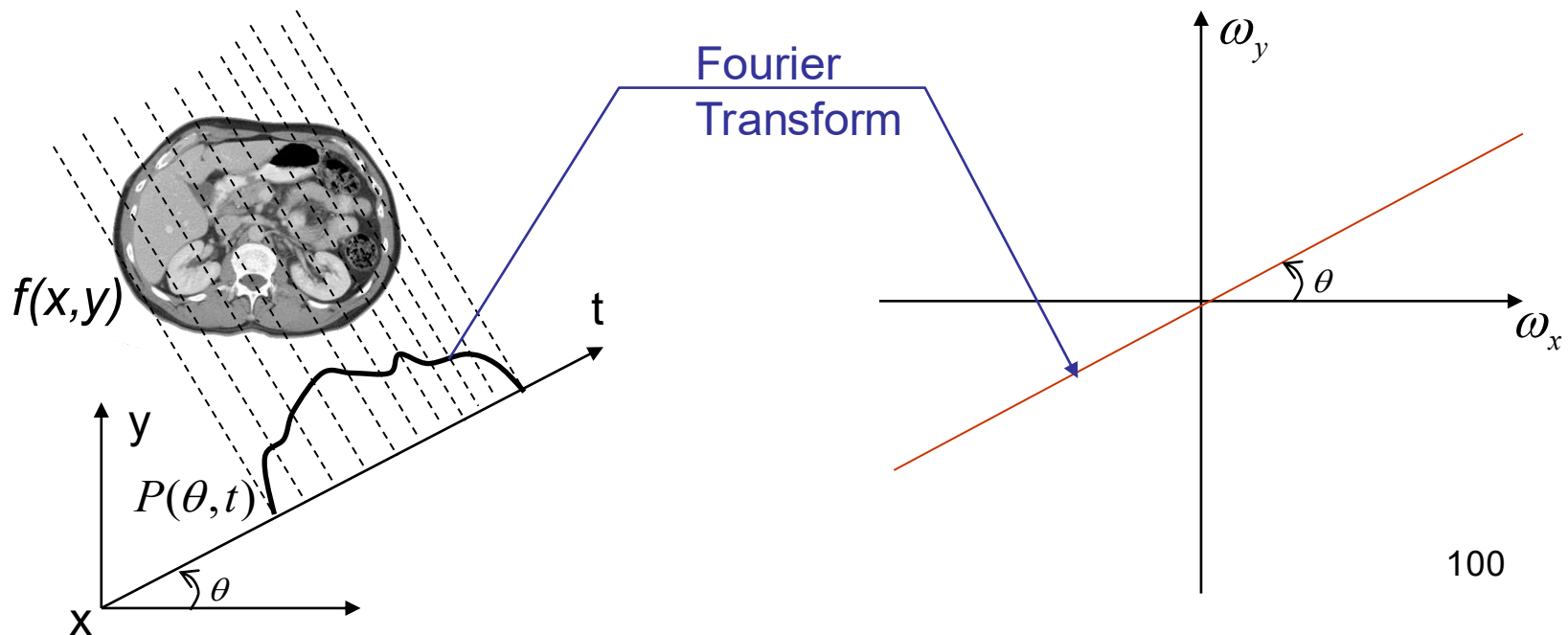
- Projection-Slice Theorem: Let  $H(t) = e^{-2\pi j t \omega}$

$$\int P(\theta, t) H(t) dt = \iint f(x, y) H(x \cos \theta + y \sin \theta) dx dy$$



$$\int P(\theta, t) e^{-2\pi j t \omega} dt = \iint f(x, y) e^{-2\pi j (x \cos \theta + y \sin \theta) \omega} dx dy$$

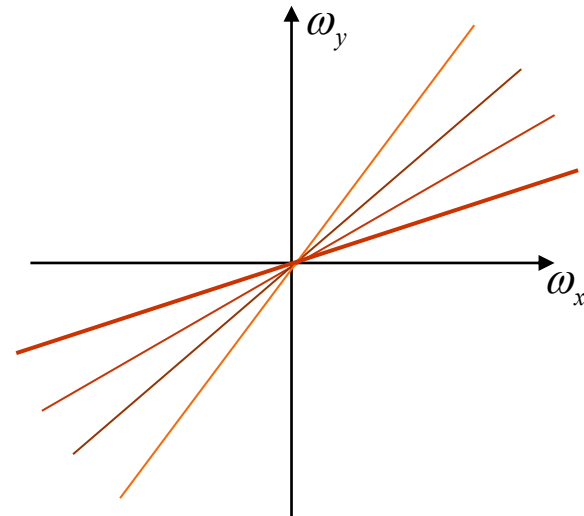
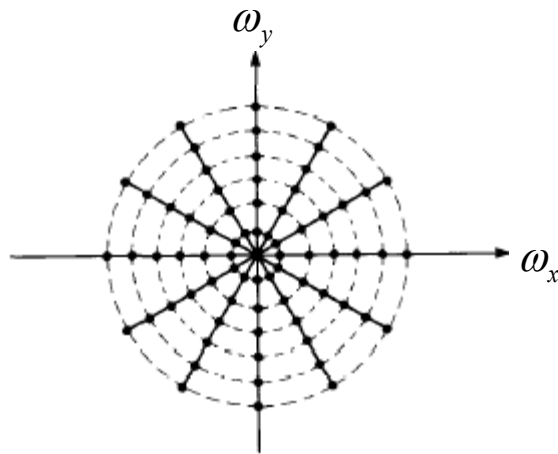
- The FT of the projection (1D) at an angle  $\theta$  is the slice at that angle of the 2D FT





# Fourier Domain Reconstruction

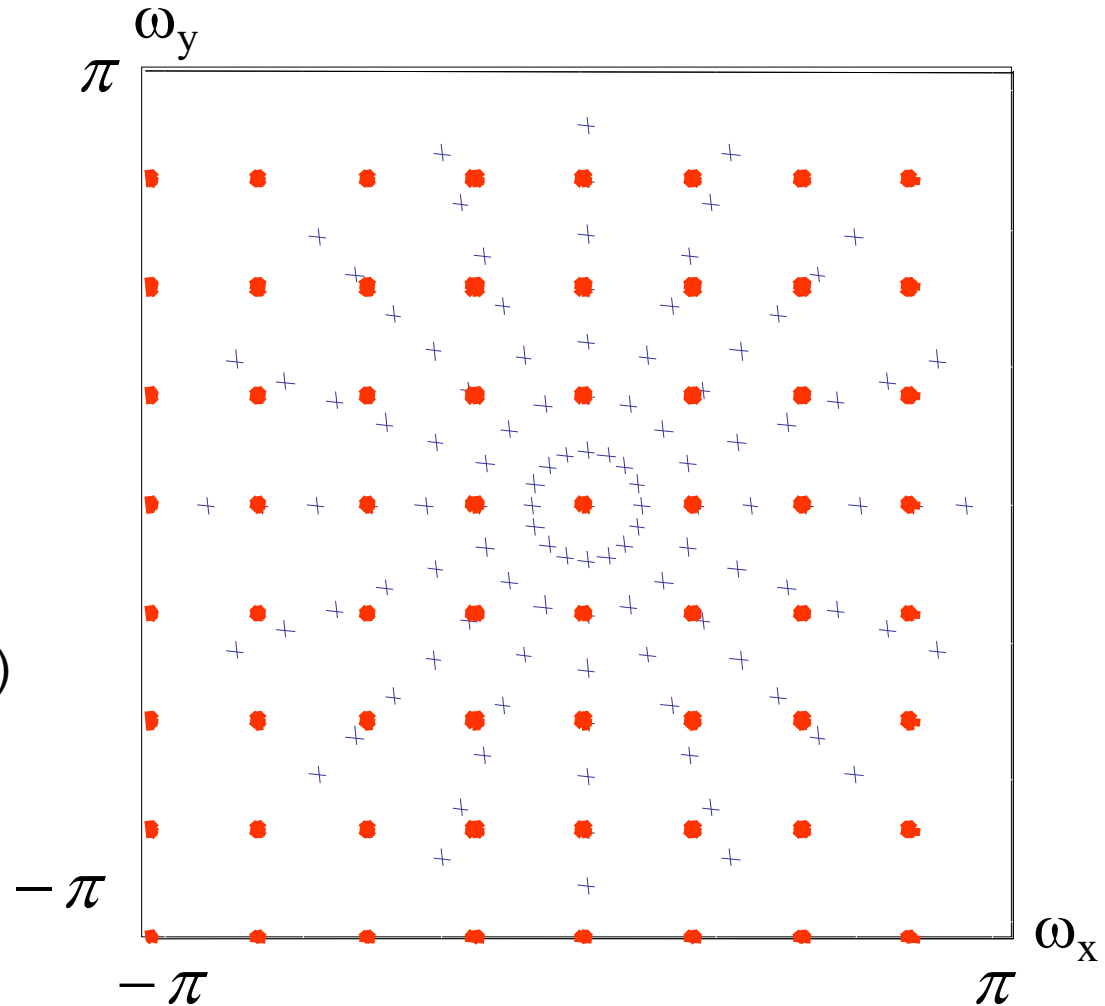
1. Collect many projections at diverse angles over the range 0-180 degrees. This yields many central slices in the spectrum
2. Interpolate the given spectral samples from polar to rectangular grid
3. Compute inverse discrete Fourier Transform to obtain an estimate of  $f(x,y)$ .



# Difficulties of Fourier Reconstruction Method

- + Polar Samples
- Cartesian grid

**Problem:** Interpolation is very problematic unless there are very many lines sampled. Sampling is dense near the origin (DC) and very sparse at higher frequencies.

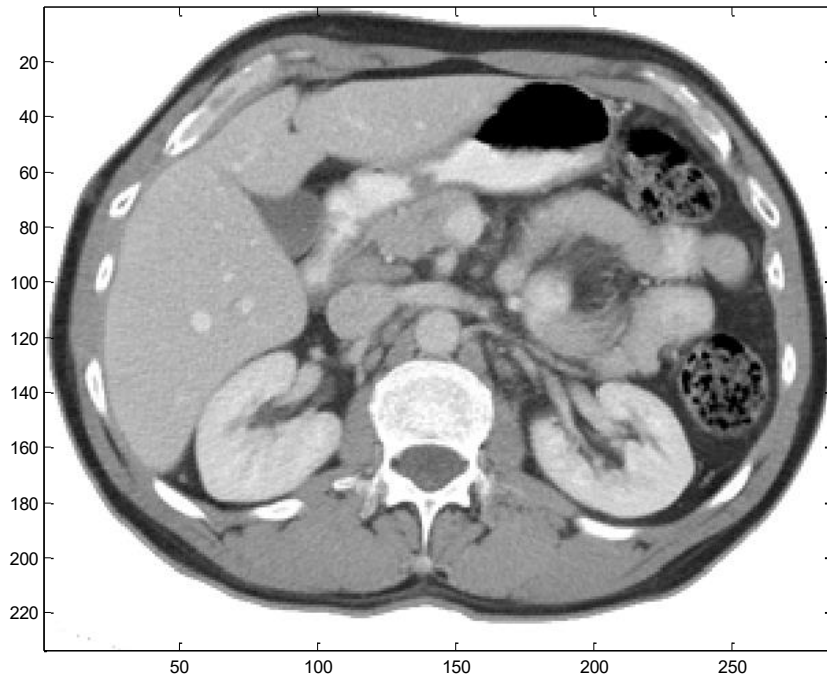


# Inverse Problem

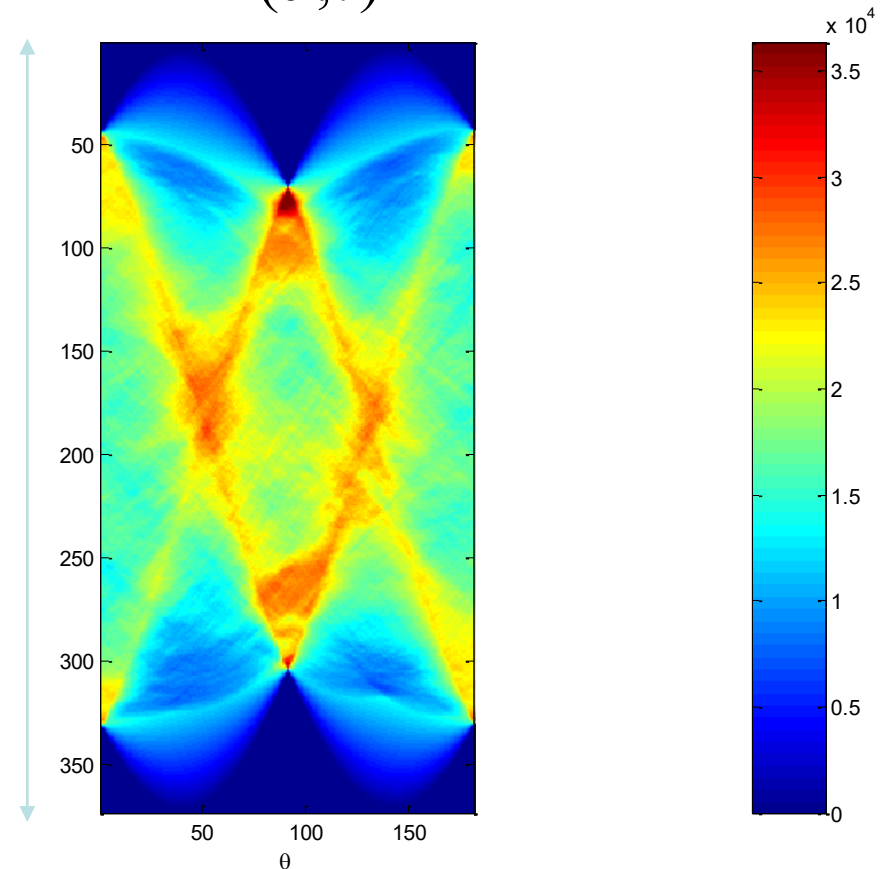
Given  $P(\theta, t)$ , find  $f(x, y)$ .  $\Rightarrow$

- Incomplete, sparse data
- Noise sensitivity
- Computational complexity

$f(x, y)$



$P(\theta, t)$



# The Projection Slice Theorem

$$P(\theta, \omega) = F(\omega_1, \omega_2) \bigg|_{\substack{\omega_1 = \omega \cos \theta \\ \omega_2 = \omega \sin \theta}}$$

$$P(\theta, \omega) = F(\omega \cos \theta, \omega \sin \theta)$$

- The FT of the projection (1D) at an angle  $\theta$  is the slice at that angle of the 2D FT.

**Basic idea:** Take multiple projections, each projection is a 1D function, then take FT of this projection. This gives a slice of the 2D FT of the object. For many projections at different angles we can construct the 2D FT of the whole object. Inverse 2D FT gives the object  $f(x, y)$  itself.