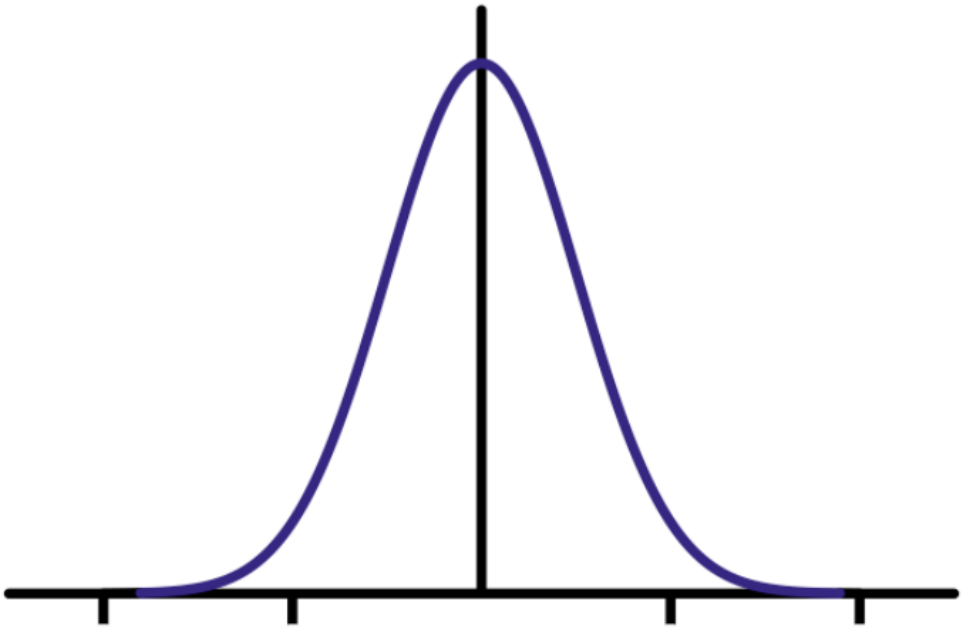
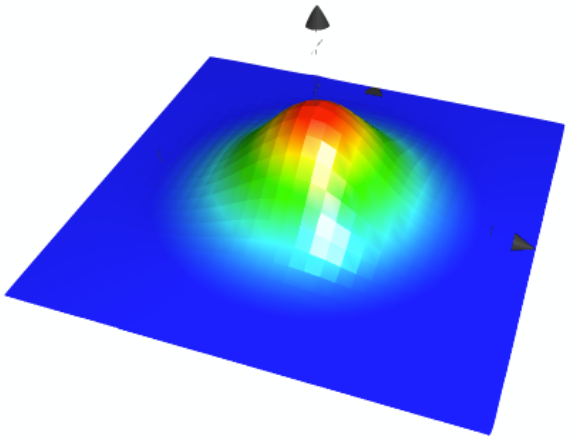
Review

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

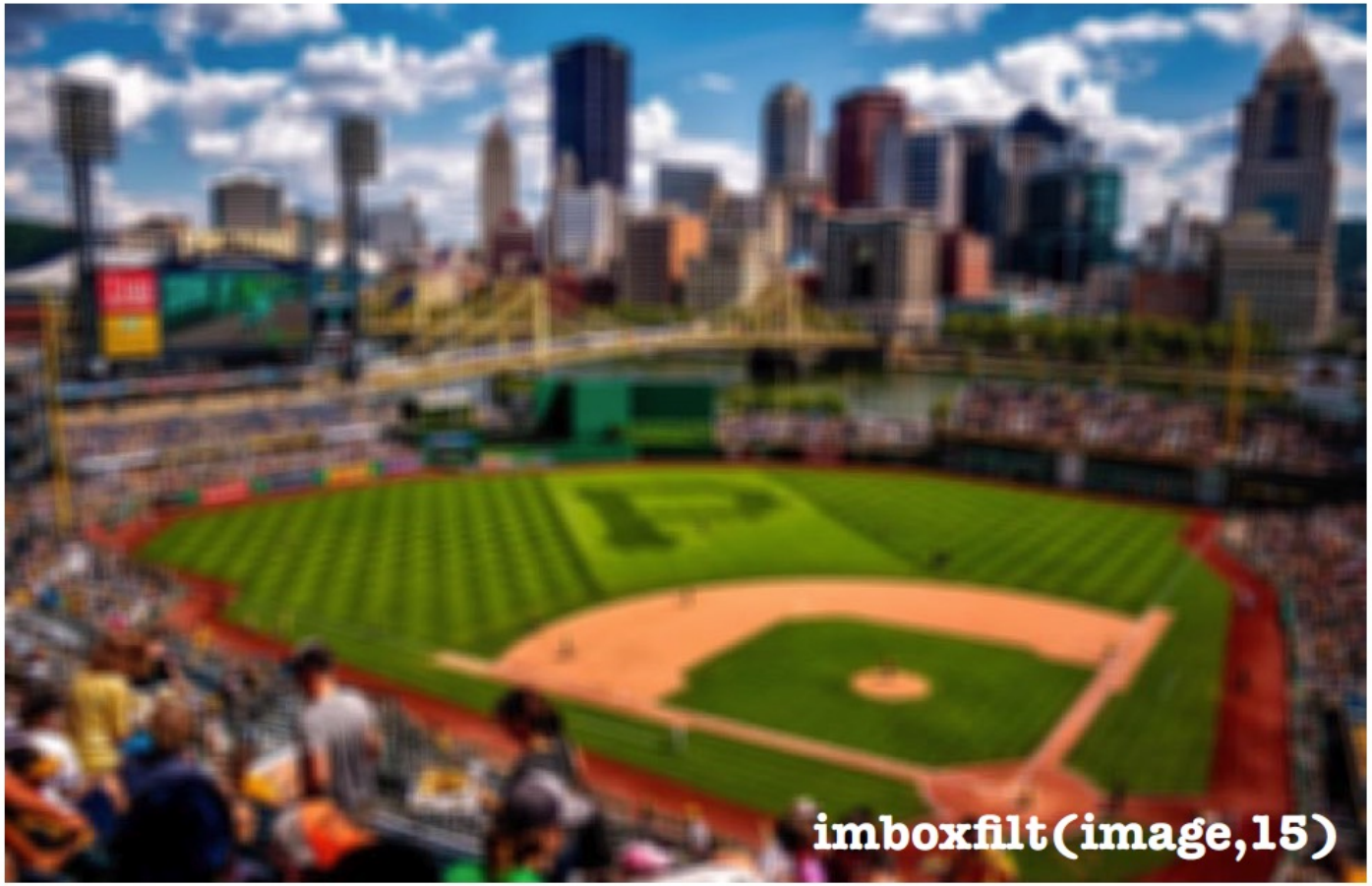
This 3 x 3 kernel is an approximation of a 2D Gaussian function

Credits: Kris Kitani

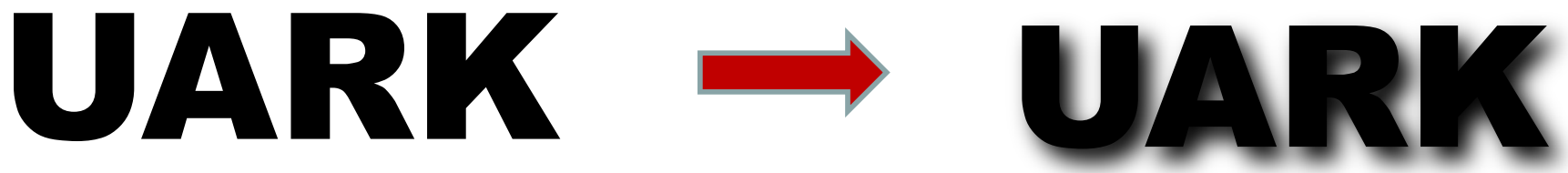
Gaussian Filtering versus Box Filter

Gaussian Filtering versus Box Filter

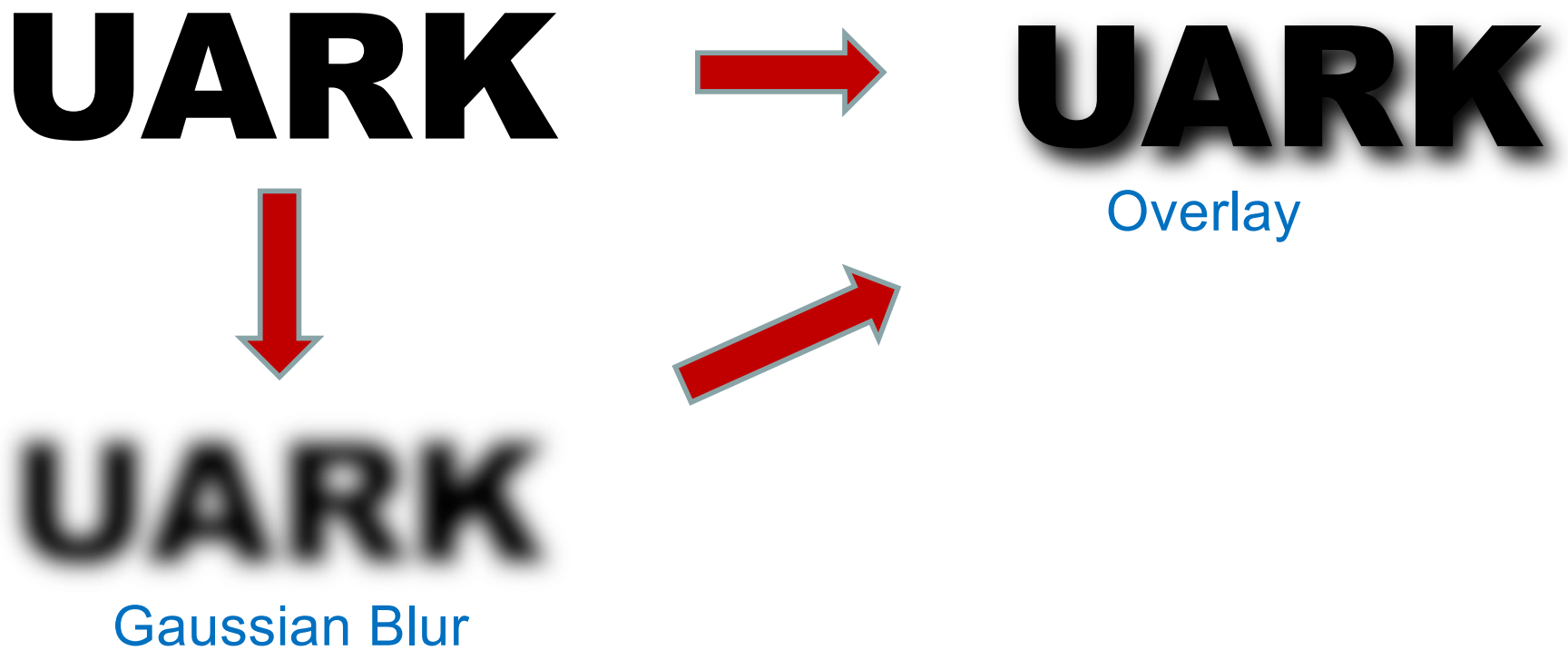
Gaussian Filtering versus Box Filter



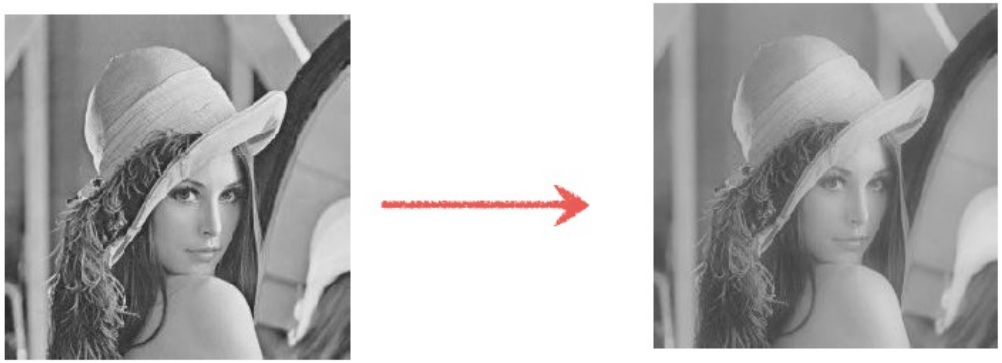
How would you create a shadow effect?



How would you create a shadow effect?



How would you create a soft focus effect?



How would you create a soft focus effect?

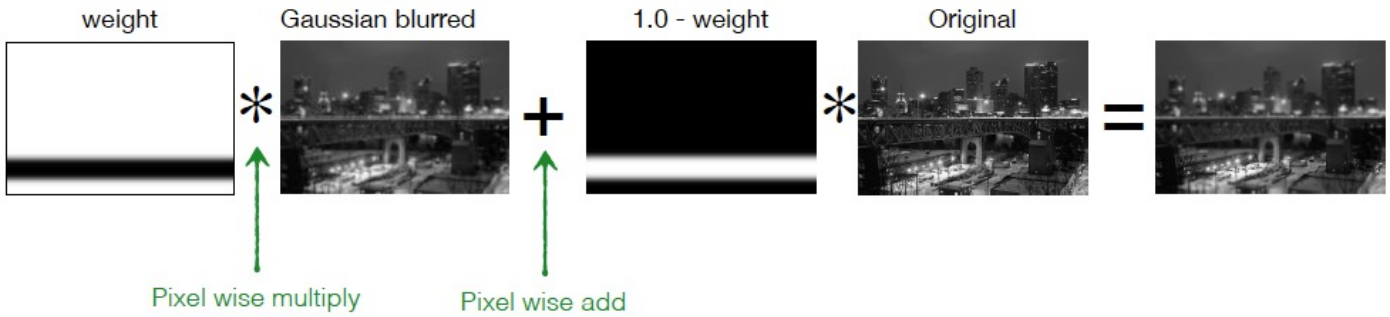
Tilt Shift Effect

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

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

http://farm8.staticflickr.com/7061/6867631897_f8377709b9_z.jpg

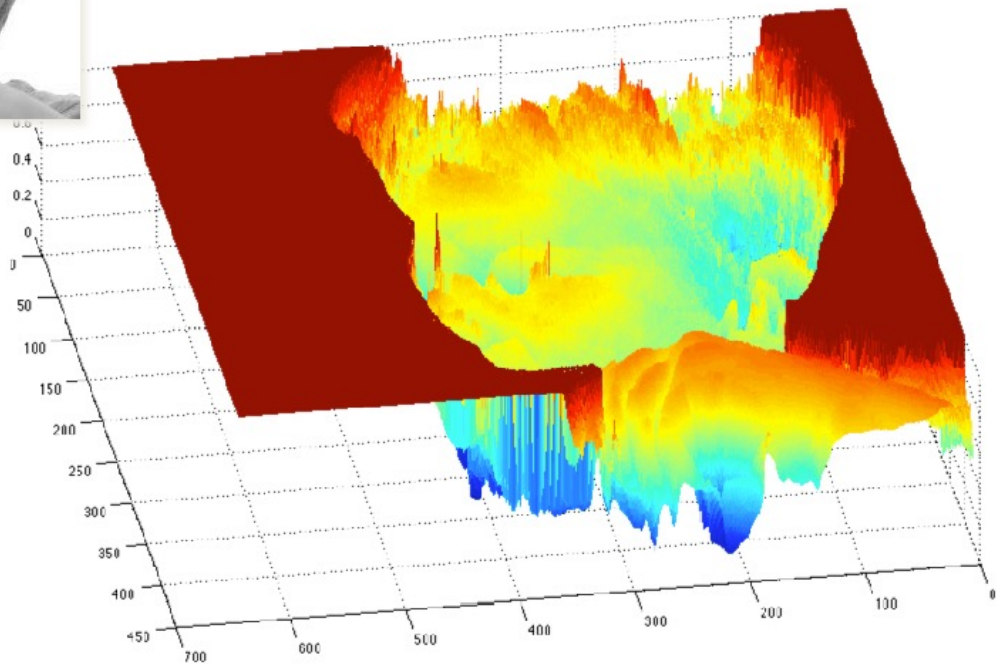
How would you create a (super low-budget) tilt-shift effect?

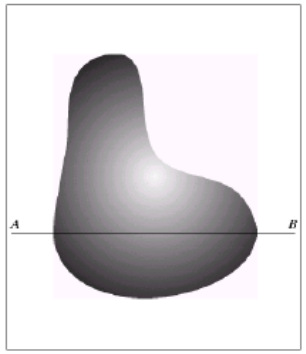


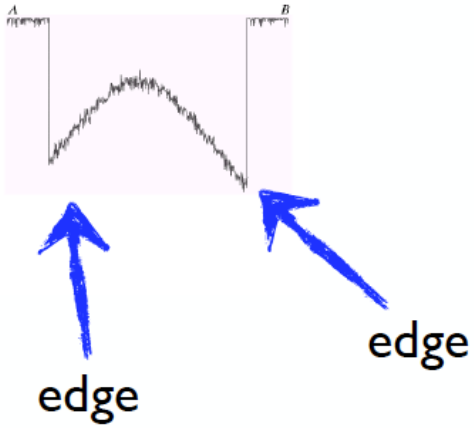
Tell me everything wrong with this wannabe tilt-shift image

Recall that an image is a 2D function

$f(\boldsymbol{x})$







How would you detect an edge?

What kinds of filter would you use?

Do you remember this from high school?

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

Do you remember this from high school?

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

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

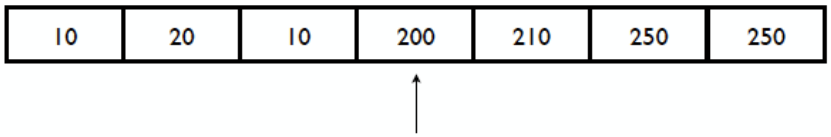
Approximation of the derivative when h is small

This definition is based on the 'forward difference' but ...

… it turns out that using the ‘central difference’ is more accurate

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

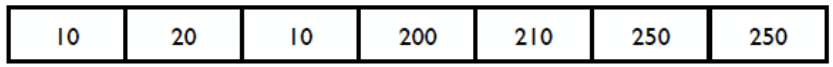
How do we compute the derivative of a discrete signal?



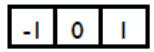
… it turns out that using the ‘central difference’ is more accurate

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

How do we compute the derivative of a discrete signal?



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



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)

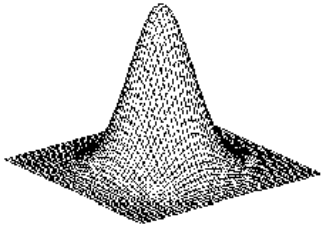
Responds to horizontal lines

Output can be positive or negative

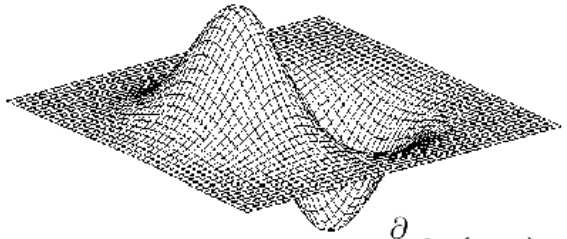
The ‘Sobel’ filter

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

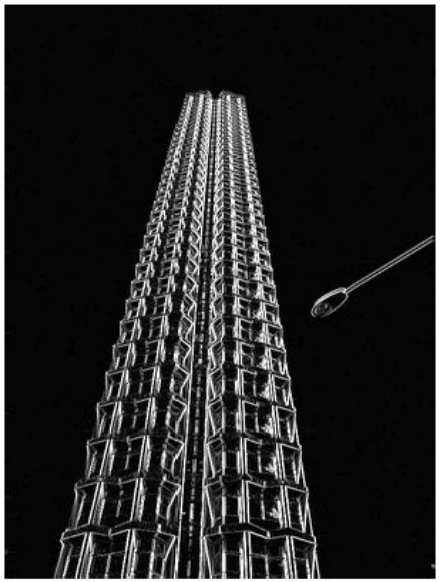
Approximation of the derivative of a Gaussian



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?

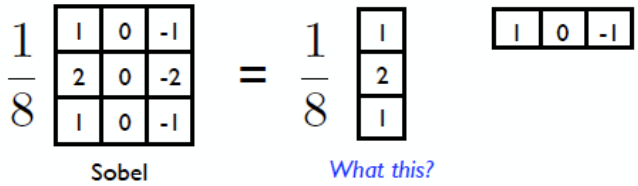
How do you visualize negative derivatives/gradients?

The ‘Sobel’ filter

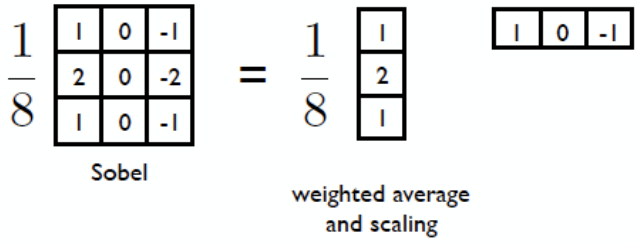
$$\frac{1}{8}\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Where does this filter come?

Decomposing the Sobel filter



Decomposing the Sobel filter



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 $\boldsymbol{S}_x =$

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

$\boldsymbol{S}_y =$

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

What is this filter called?

Run filter over image $\dfrac{\partial \boldsymbol{f}}{\partial x} = \boldsymbol{S}_x \otimes \boldsymbol{f}$ $\dfrac{\partial \boldsymbol{f}}{\partial y} = \boldsymbol{S}_y \otimes \boldsymbol{f}$

What are the dimensions?

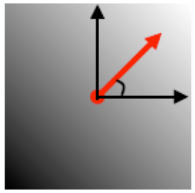
Image gradient $\nabla \boldsymbol{f} = \left[\dfrac{\partial \boldsymbol{f}}{\partial x}, \dfrac{\partial \boldsymbol{f}}{\partial y}\right]$

What are the dimensions?

MATCH THAT GRADIENT !

(a)

(b)



(c)

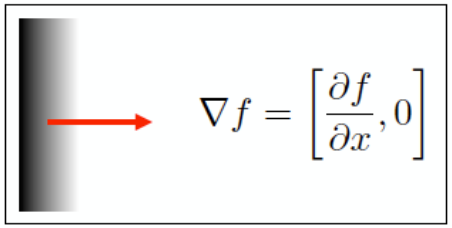
(1) $\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$

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

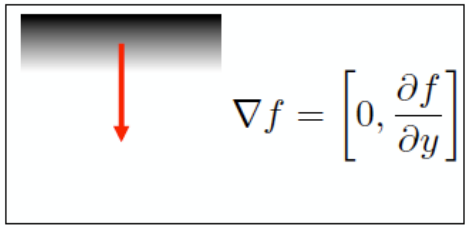
(3) $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$

Image Gradient

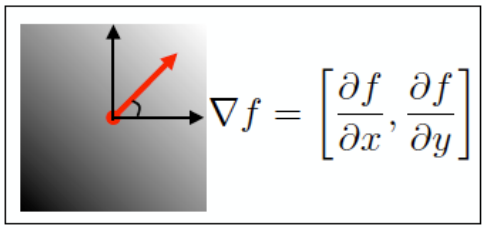
Gradient in x only



Gradient in y only



Gradient in both x and y



Gradient direction

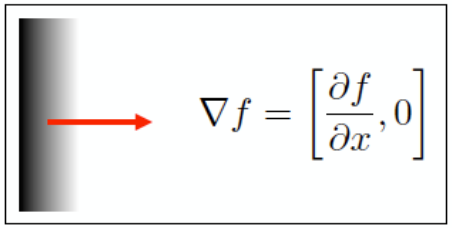
?

Gradient magnitude

?

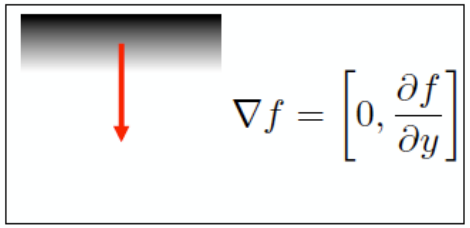
Image Gradient

Gradient in x only



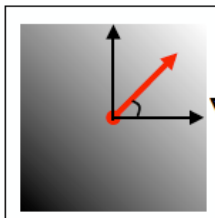
$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$

Gradient in y only



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

Gradient in both x and y



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

Gradient direction

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

How does the gradient direction relate to the edge?

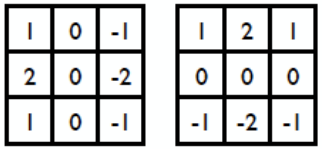
Gradient magnitude

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

What does a large magnitude look like in the image?

Common ‘derivative’ filters

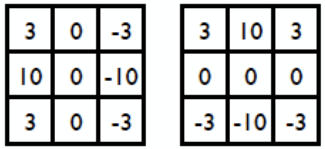
Sobel



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

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

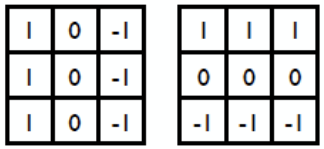
Scharr



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

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

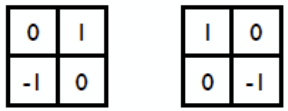
Prewitt



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

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

Roberts

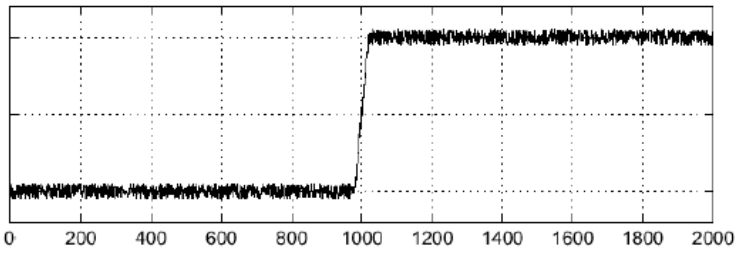


0	1
-1	0

1	0
0	-1

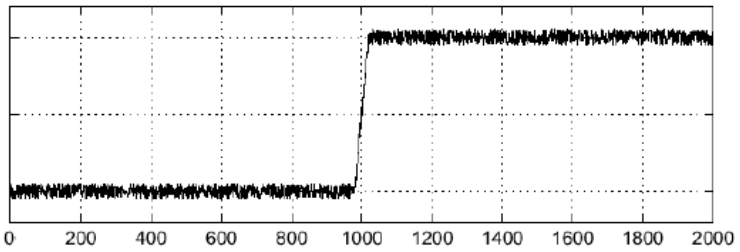
How do you find the edge from this signal?

Intensity plot



How do you find the edge from this signal?

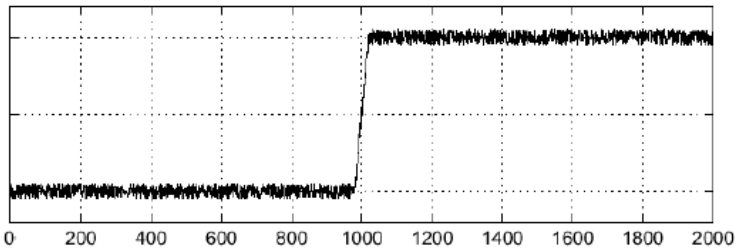
Intensity plot



Use a derivative filter!

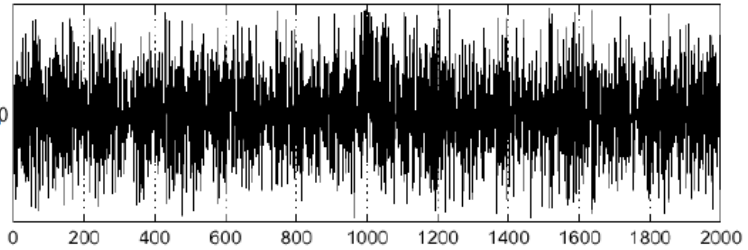
How do you find the edge from this signal?

Intensity plot



Use a derivative filter!

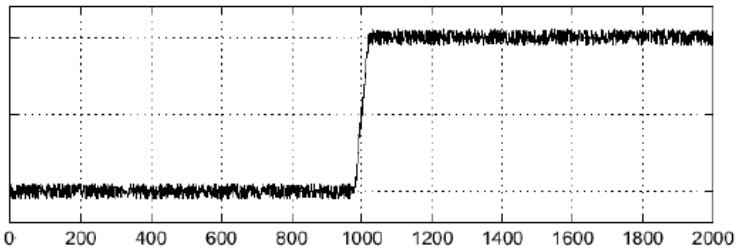
Derivative plot



What happened?

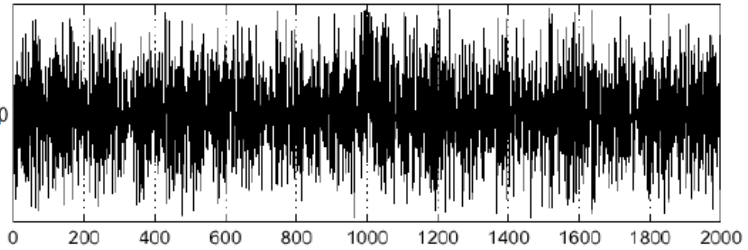
How do you find the edge from this signal?

Intensity plot



Use a derivative filter!

Derivative plot



Derivative filters are sensitive to noise

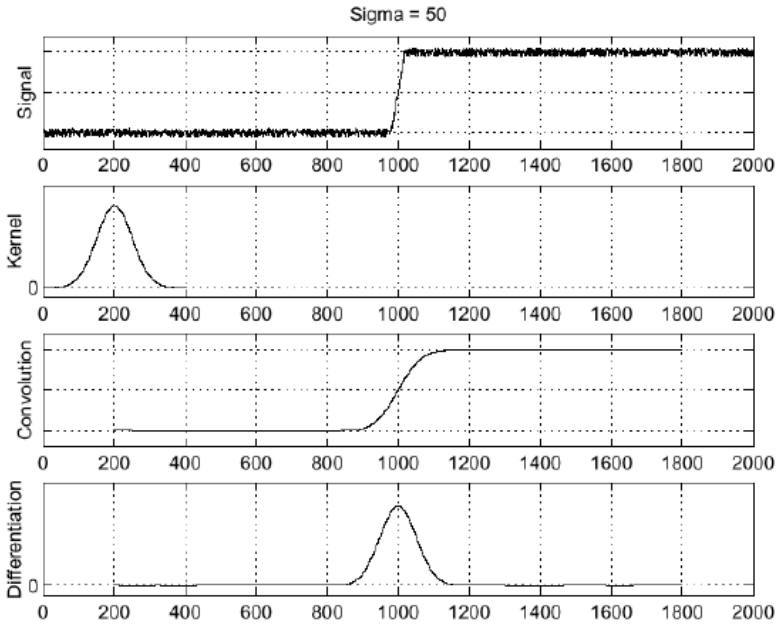
Input

Gaussian

Smoothed input

Derivative

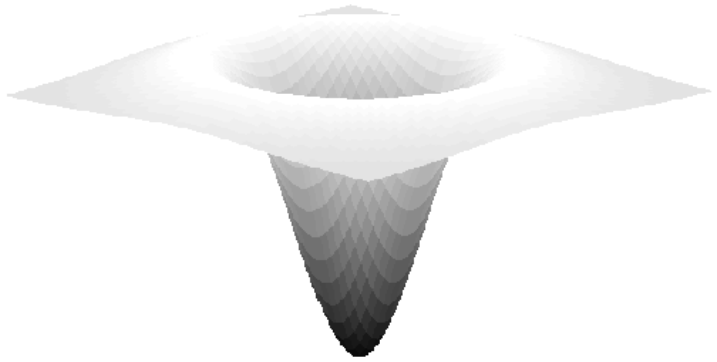
Output



Don't forget to smooth before running derivative filters!

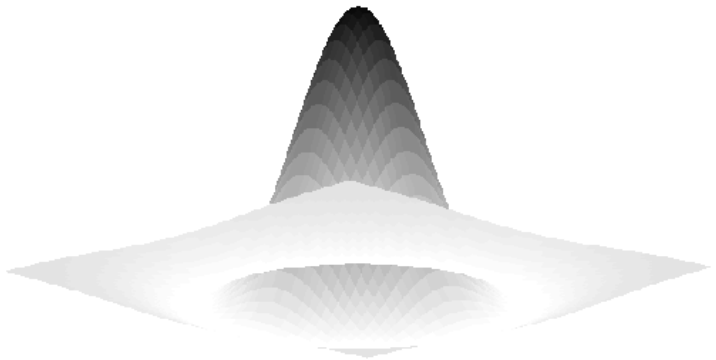
Laplace filter

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

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

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

?	?	?

first-order
finite difference

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

derivative filter

1	0	-1

second-order
finite difference

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

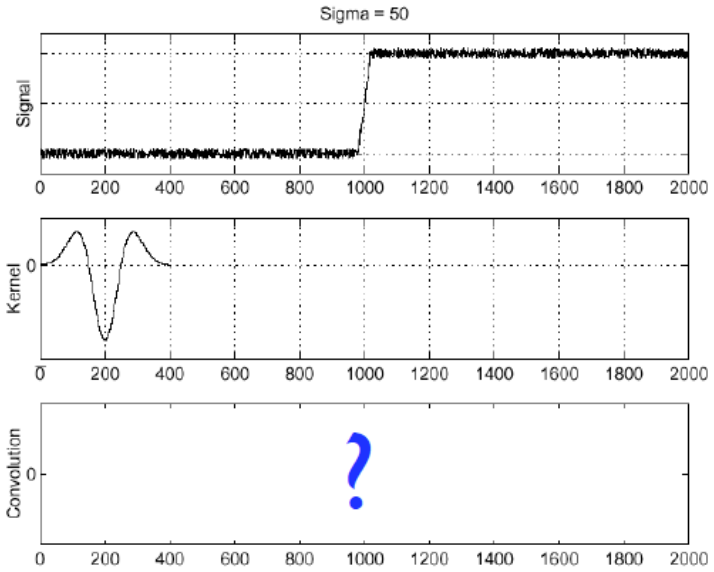
Laplace filter

1	-2	1

Input

Laplacian

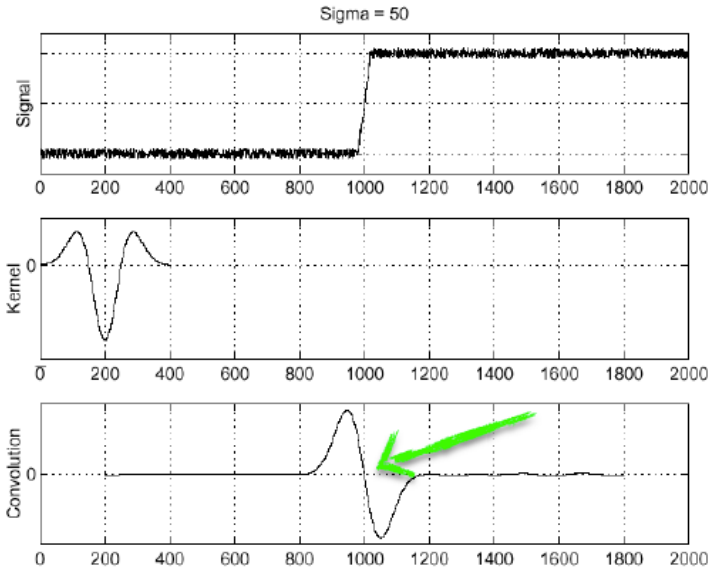
Output



Input

Laplacian

Output

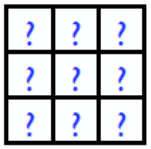


Zero crossings are more accurate at localizing edges

Second derivative is noisy

2D Laplace filter

1D Laplace filter



2D Laplace filter

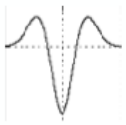
2D Laplace filter

1	-2	1

1D Laplace filter

?	?	?
?	?	?
?	?	?

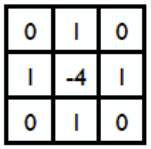
2D Laplace filter



hint

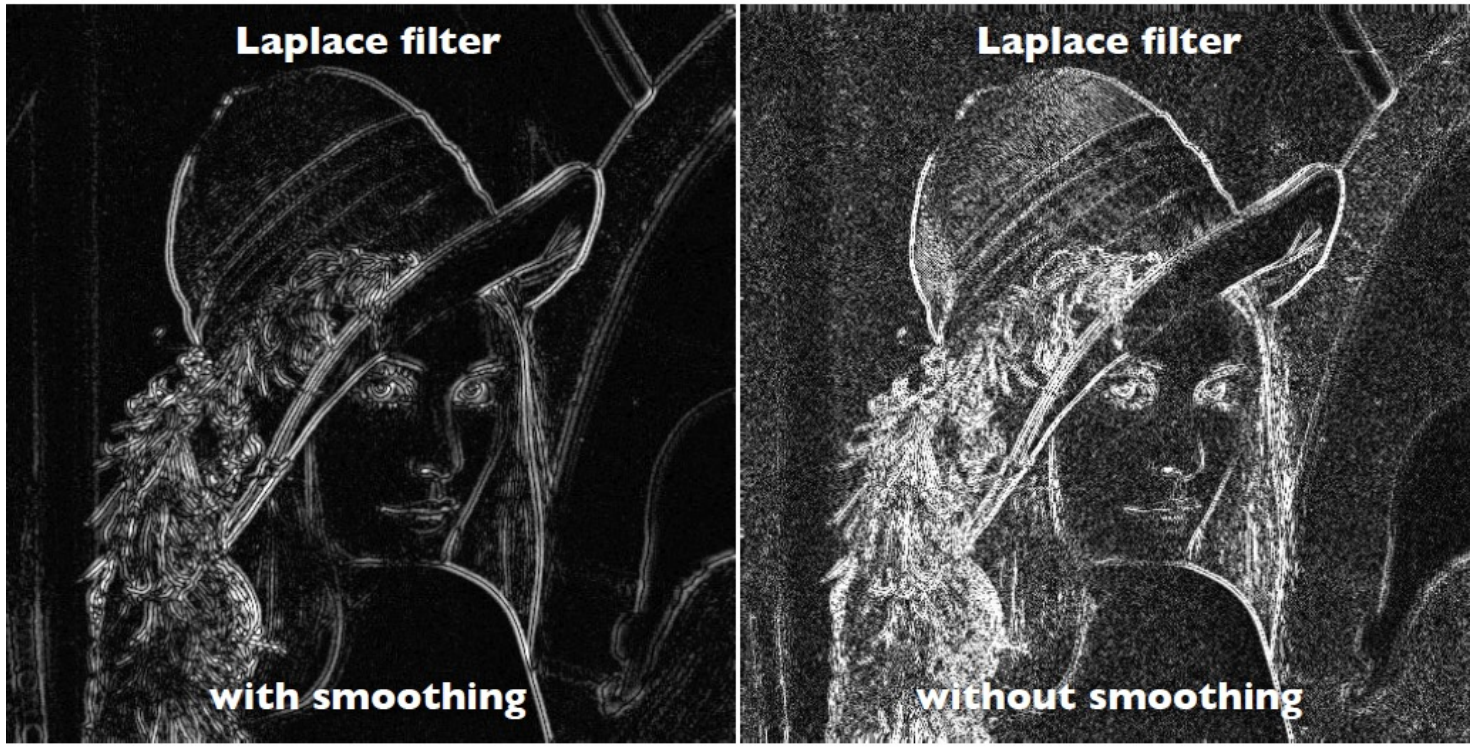
2D Laplace filter

1D Laplace filter

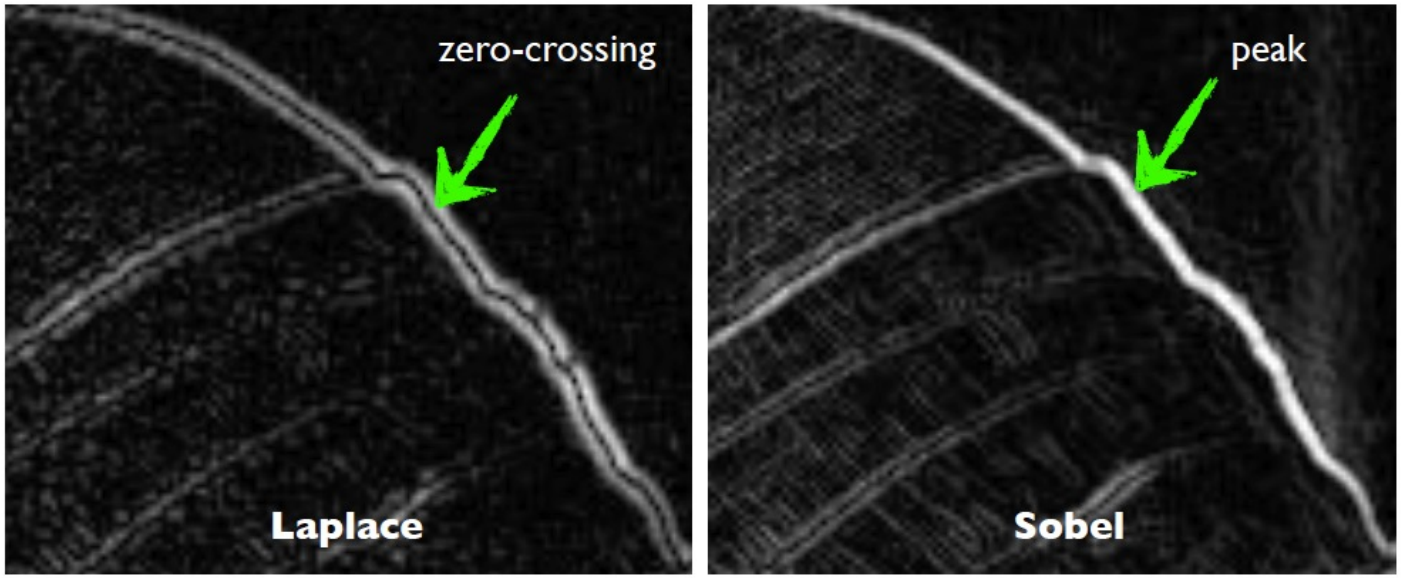


2D Laplace filter

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

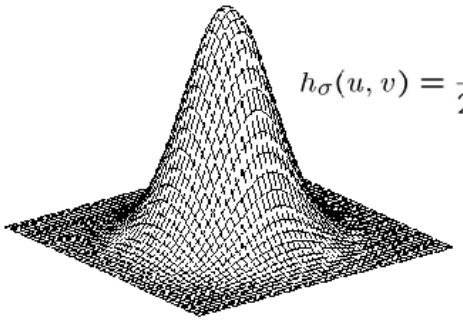

Laplace filter
with smoothing
Laplace filter
without smoothing

What's different between the two results?



Zero crossings are more accurate at localizing edges

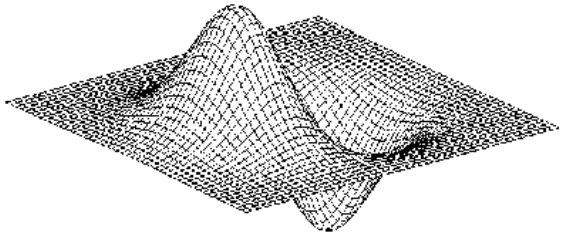
(but not very convenient)



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

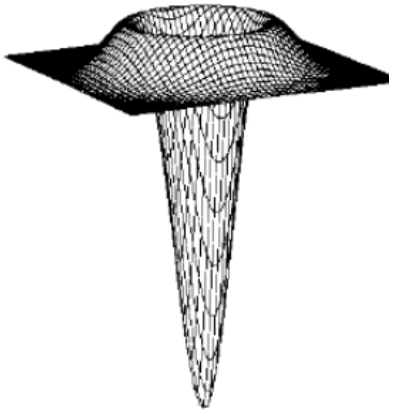
Gaussian

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



Derivative of Gaussian

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



Laplacian of Gaussian

2D Gaussian Filters

Differences Between 1D and Multi-Dimensional Signal Processing

- Much more data for M-D signal processing

  1) 1-D speech ⟹ 10K samples/sec

  2) M-D television ⟹ 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…

Credits: Tien Bui

Signals, Systems and Transforms

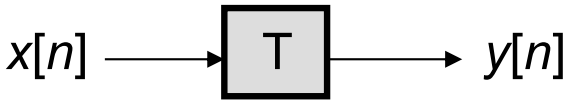
Digital Signal Processing ➡ Digital Image Processing

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 ⇨ System

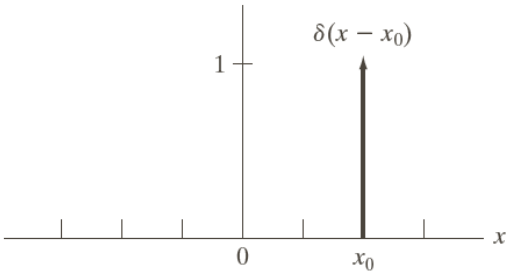


- Linear system $T(ax_1[n] + bx_2[n]) = T(ax_1[n]) + T(bx_2[n]) = ay_1[n] + by_2[n]$
- Time (shift) invariant system

  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
- FIR: always stable
- IIR: not always stable

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



$\delta[n]$ → LTI → $h[n]$ Impulse response

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

$y[n] = T(x[n]) = h[n] * x[n]$ Where $*$ denotes "convolution"

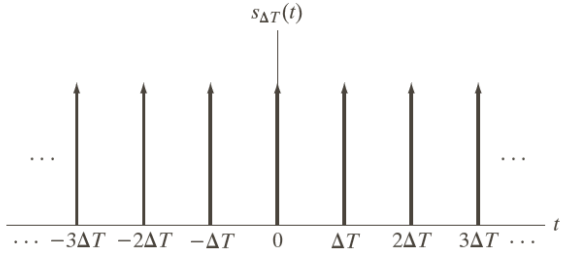
https://en.wikipedia.org/wiki/Convolution

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

$$\int_{-\infty}^{\infty} \delta(x)d(x) = 1 \qquad \sum_{-\infty}^{\infty} \delta[n] = 1$$

$$\int_{-\infty}^{\infty} f(x)\delta(x)d(x) = f(0) \qquad \sum_{-\infty}^{\infty} f[n]\delta[n] = f[0]$$

$$\int_{-\infty}^{\infty} f(x)\delta(x - x_0)d(x) = f(x_0) \qquad \sum_{-\infty}^{\infty} f[n]\delta[n - n_0] = f[n_0]$$



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

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

Filters we have learned so far …

The 'Box' filter

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Gaussian filter

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Sobel filter

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

Laplace filter

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

Computation of Correlation and Convolution (1-D)

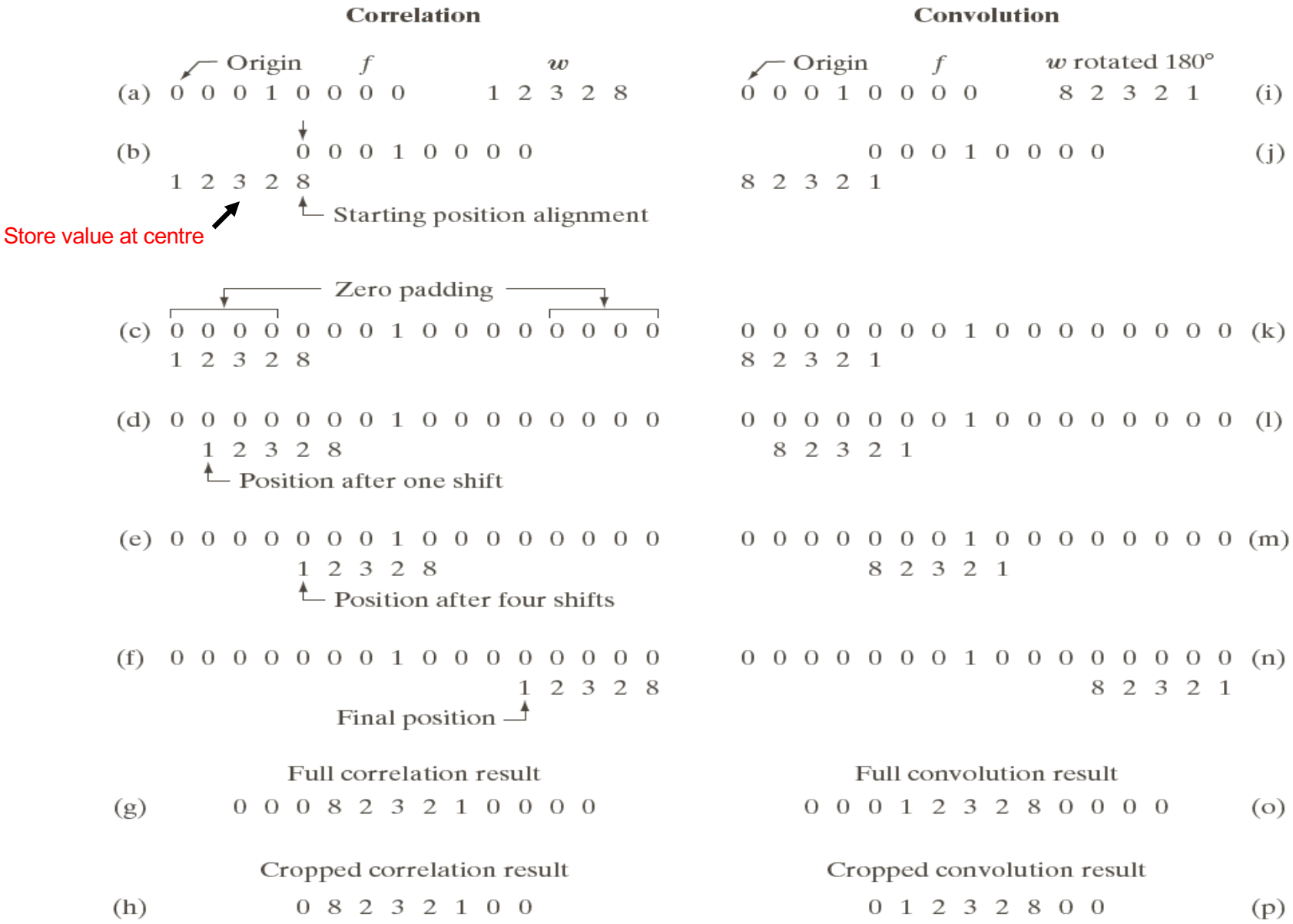


FIGURE 3.29 Illustration of 1-D correlation and convolution of a filter with a discrete unit impulse. Note that correlation and convolution are functions of *displacement*.

Filtering vs Convolution

filtering
(cross-correlation)

$h = g \otimes f$

output filter image

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

What's the difference?

convolution

$h = g \star f$

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

Credit: Steve Seitz

Filtering vs Convolution

filtering
(cross-correlation)

$h = g \otimes f$

output filter image

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

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)

$h = g \otimes f$

output filter image

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

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...

$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

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

Derivative Theorem of Convolution

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

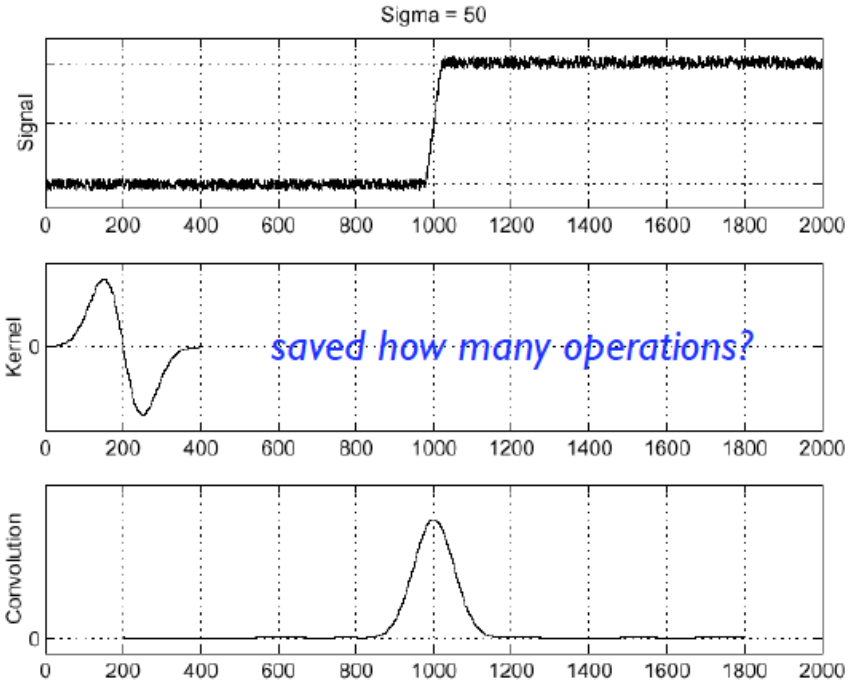
Input

Derivative of Gaussian

$\frac{\partial}{\partial x}h$

Output

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



Recall ...

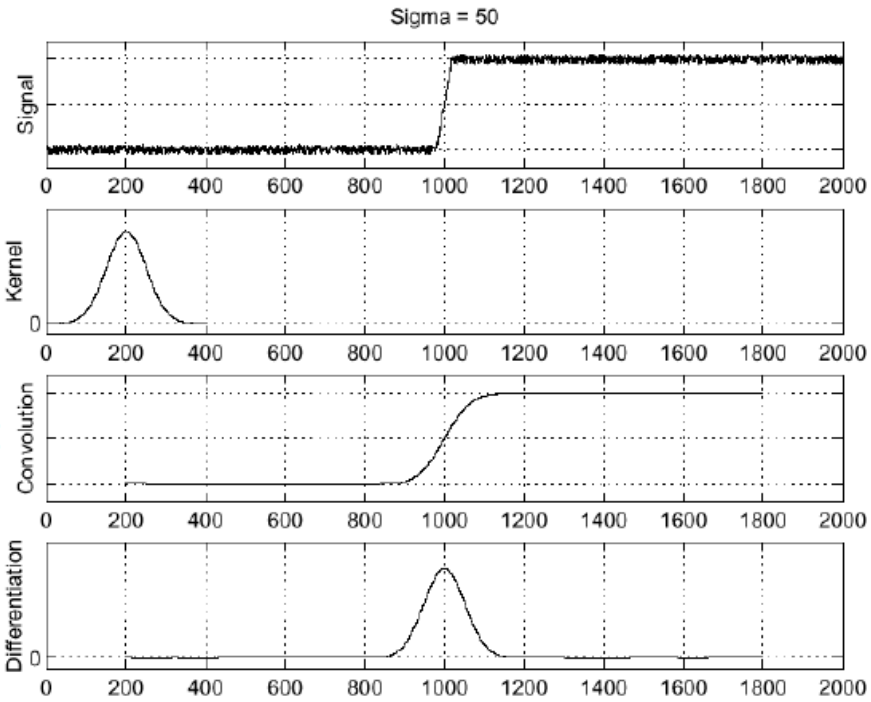
Input

Gaussian

Smoothed input

Derivative

Output



Continuous Fourier Transform

1-D and continuous

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

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

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

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

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

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

$$F[f(x) * h(x)] = \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} f(t)h(x-t)dt\right] e^{-j2\pi\omega x}dx$$

$$= \int_{-\infty}^{\infty} f(t)\left[\int_{-\infty}^{\infty} h(x-t)\, e^{-j2\pi\omega x}dx\right]dt$$

$$F[h(x-t)] = H(\omega)\, e^{-j2\pi\omega.t}$$

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

$$= H(\omega)\int_{-\infty}^{\infty} f(t)\, e^{-j2\pi\omega.\mathrm{t}}\, dt$$

$$= H(\omega).F(\omega)$$

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

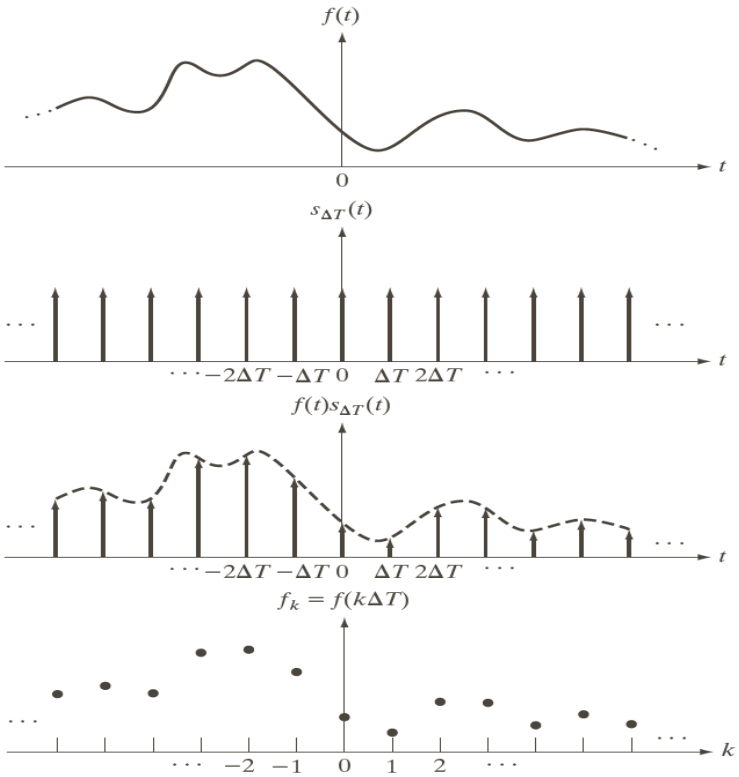
Reverse: The inverse FT of $H(\omega).F(\omega)$ gives the convolution of the two functions $f(x)*h(x)$ in spatial domain.

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

$$\frac{1}{\sqrt{N}}\begin{bmatrix}1\\1\\.\\.\\1\end{bmatrix},\ \frac{1}{\sqrt{N}}\begin{bmatrix}1\\e^{j\frac{2\pi}{N}.1}\\e^{j\frac{2\pi}{N}.2}\\.\\e^{j\frac{2\pi}{N}.(N-1)}\end{bmatrix},\ \frac{1}{\sqrt{N}}\begin{bmatrix}1\\e^{j\frac{2\pi}{N}.2.1}\\e^{j\frac{2\pi}{N}.2.2}\\.\\e^{j\frac{2\pi}{N}.2(N-1)}\end{bmatrix},\ \ldots\ \frac{1}{\sqrt{N}}\begin{bmatrix}1\\e^{j\frac{2\pi}{N}.(N-1).1}\\e^{j\frac{2\pi}{N}.(N-1).2}\\.\\e^{j\frac{2\pi}{N}.(N-1)(N-1)}\end{bmatrix}$$

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

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

$$\vec{x} = \sum_{k=0}^{N-1} X_k \, \vec{b}_k \quad \text{or} \quad \boxed{\begin{aligned} x[n] &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X_k \, e^{j\frac{2\pi}{N}kn} \\ X_k &= \frac{1}{\sqrt{N}} \sum_{n} x[n] \, e^{-j\frac{2\pi}{N}kn} \end{aligned}}$$

Discrete Fourier Transform pair

Discrete Fourier Transform (1-D)

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

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

ω 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

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

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

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

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

2-D D.F.T

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

$$X(k_1,k_2)=\begin{cases}\sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1}x(n_1,n_2)\,e^{-j\frac{2\pi n_1k_1}{N_1}}e^{-j\frac{2\pi n_2k_2}{N_2}} & 0\le k_1<N_1 \\ & 0\le k_2<N_2 \\ 0 & Otherwise\end{cases}$$

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

$$x(n_1,n_2)=\begin{cases}\frac{1}{N_1N_2}\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1}X(k_1,k_2)\,e^{j\frac{2\pi k_1n_1}{N_1}}e^{j\frac{2\pi k_2n_2}{N_2}} & 0\le n_1<N_1 \\ & 0\le n_2<N_2 \\ 0 & Otherwise\end{cases}$$

Computation of 2D-DFT

- Fourier transform matrix:

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

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

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

Computation of 2D-DFT

- Example. N = 4:

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

- Matlab function: **`dftmtx`**

Computation of 2D-DFT

- 1D-DFT (of 1D signal **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 **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.*

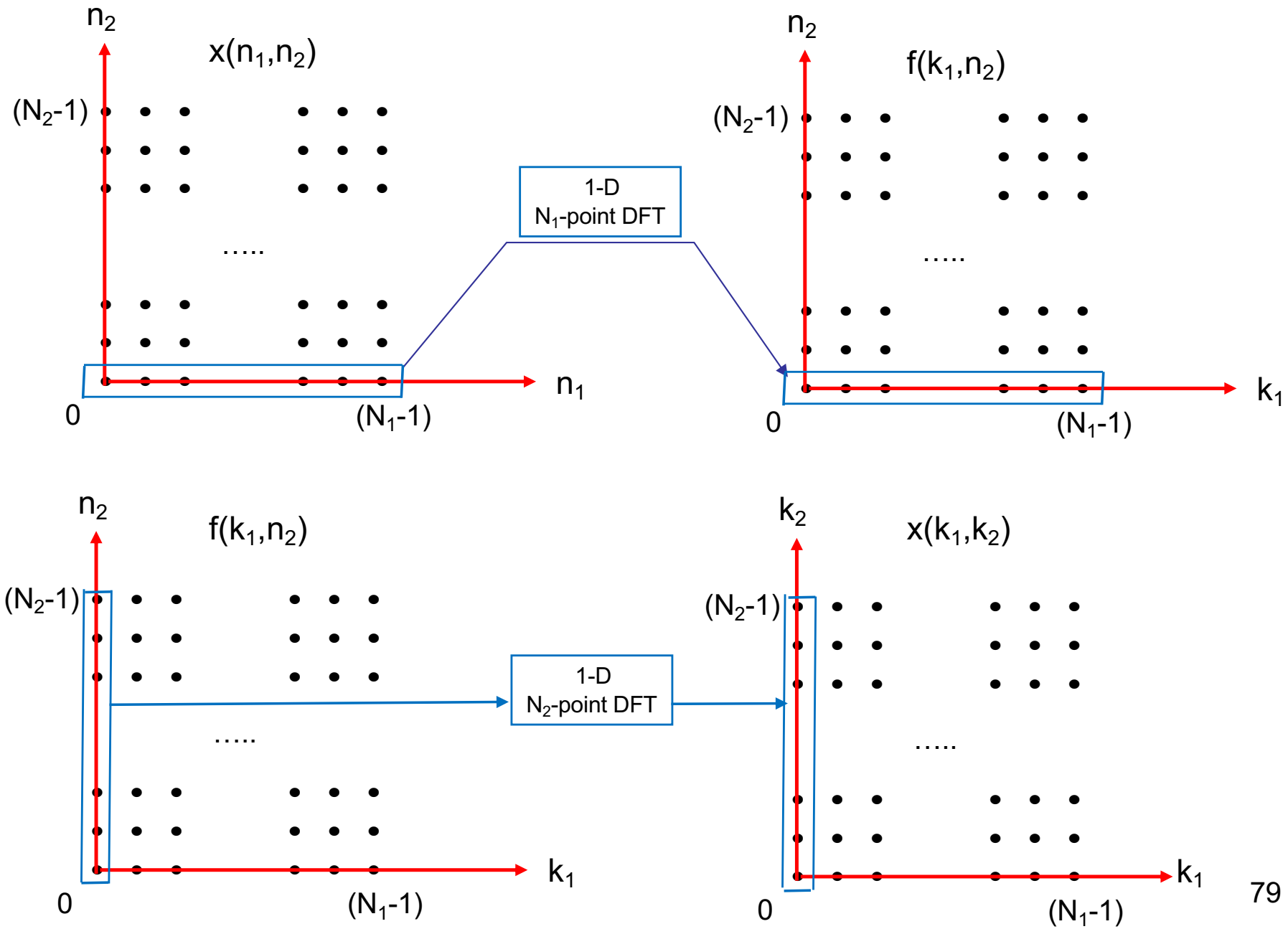
How many (k_1, k_2) *? Ans.* $N^2 \Longrightarrow N^4$ *mults. ,* N^4 *adds.*

- 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 ?



After J.S.Lim: 2-D Signal and Image Processing

How to compute 2D D.F.T ?

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

- 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

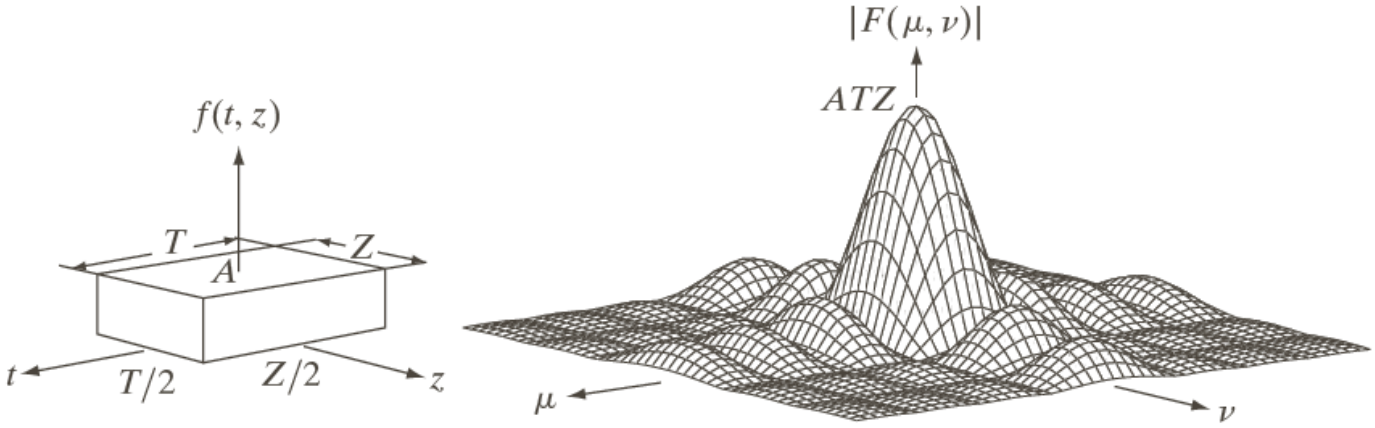
  - fast, non I/O intensive for matrix transpose

  - based on divide-and-conquer
- Basis idea:

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

- 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 μ-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

$$If \;\; T[X(n_1,n_2)] = Y(n_1,n_2)$$

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

- Impulse Response

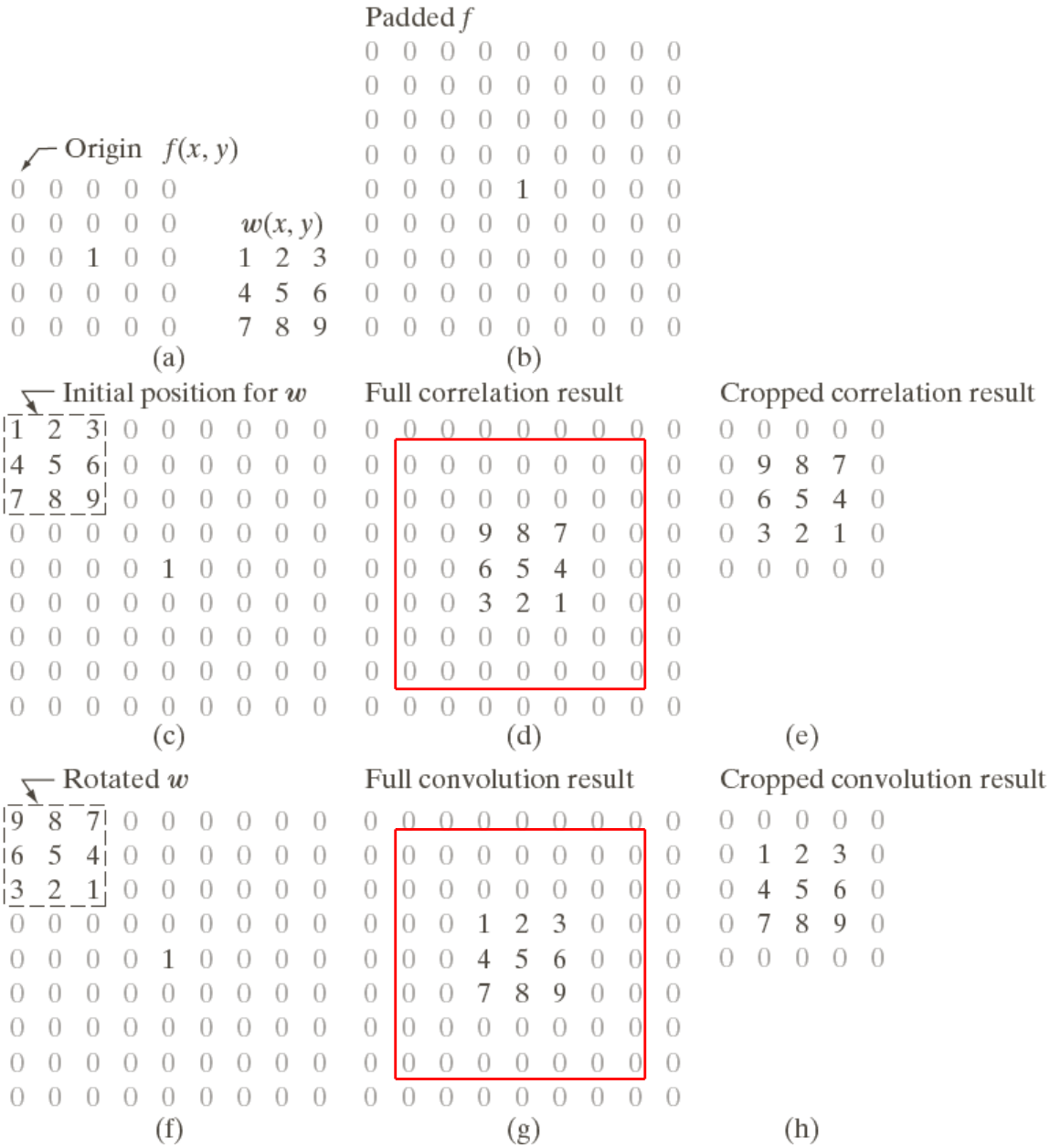
$$Y(n_1,n_2) = T[X(n_1,n_2)] = T[\sum_{k_1}\sum_{k_2} X(k_1,k_2)\delta(n_1-k_1,n_2-k_2)]$$

$$= \sum_{k_1}\sum_{k_2} X(k_1,k_2)T[\delta(n_1-k_1,n_2-k_2)]$$

$$Y(n_1,n_2) = \sum_{k_1}\sum_{k_2} X(k_1,k_2)h(n_1-k_1,n_2-k_2) \qquad \text{(2-D convolution)}$$

- 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 NxN image convolves with an MxM filter (impulse response) requires:

$(N+M-1)^2 M^2 \quad for\ N >> M:\ \approx N^2 M^2$

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

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

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

$$= \sum_{k_1} h_1(n_1-k_1) \sum_{k_2} X(k_1,k_2)\, h_2(n_2-k_2)$$

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

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

Computational costs $\approx 2N^2 M$