Digital Image Processing

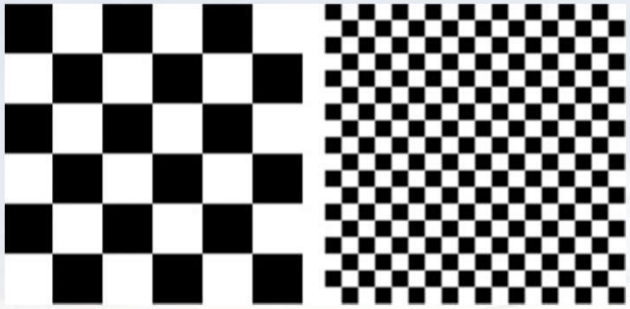
Linear image processing: principles and spatial domain filtering

Erchan Aptoula

Spring 2016-2017

Recall

- Point processing has very limited capacity.
- Exploiting spatial information and pixel neighborhoods is imperative.



We now leave behind binary images, and enter the world of **digital grayscale images** (e.g. at 8-bit: $f: \mathbb{Z}^2 \rightarrow [0,255] \cap \mathbb{Z}$) and how to process them with linear tools.

- Theoretical principles and properties of linear image processing
- Spatial domain filtering: smoothing, edge detection, sharpening

Linear image processing is one of the oldest image processing approaches.

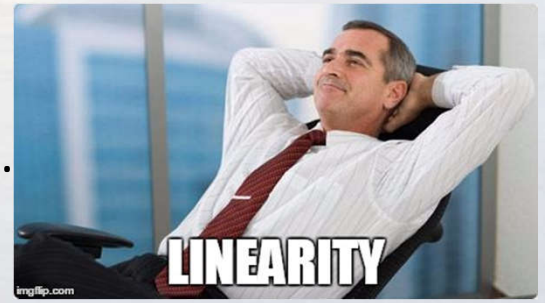
	Image Analysis Approaches	
Linear	**Linear**	**Statistical**
Non linear	**Morphological**	**Syntactic**

It provides efficient solutions for noise reduction, image enhancement, and more, and has unique tools in its arsenal, such as Fourier analysis.

Linearity:

$O(\alpha f + \beta g) = \alpha O(f) + \beta O(g)$, where f, g are images and α, β real constants

And why do we like linearity?..



A linear operator O is characterized by its ***point spread function*** (PSF),
i.e. its response to a point source of brightness at (x, y):

$$O(point\ source) = O\big(\delta(\alpha - x, \beta - y)\big) = h(x, \alpha, y, \beta)$$

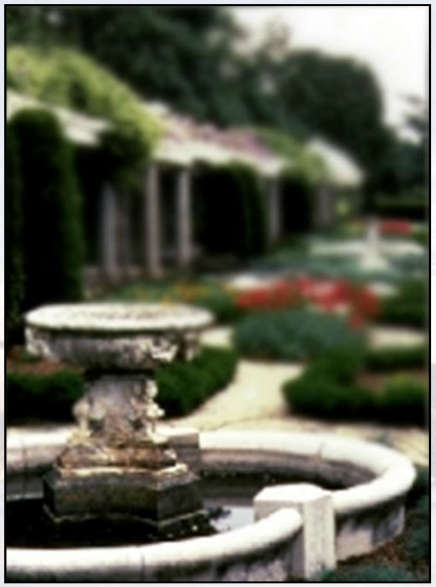
where $\delta(x - \alpha, y - \beta) = \begin{cases} 1, \text{if}(x = \alpha \wedge y = b) \\ 0, \text{otherwise} \end{cases}$

With a point source that is a times brighter: $O\big(a\delta(\alpha - x, \beta - y)\big) = ah(x, \alpha, y, \beta)$

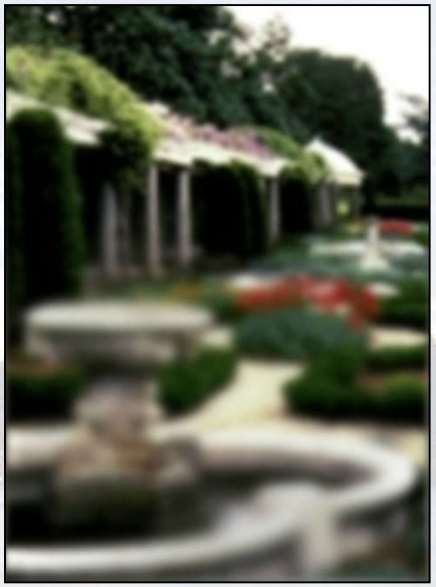
PSF is used to characterize
both cameras
and linear operators.

Example of camera PSFs:

normal vision



myopia



hyperopia

Remember that digital images are also collections of point sources at various brightness levels; for instance in the case of a 3x3 image:

$$f = \begin{bmatrix} f(0,0) & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & f(0,1) & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \cdots + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & f(2,2) \end{bmatrix}$$

so the effect of an operator characterized by the PSF h on an image f of size $N \times N$ can be written as:

$$g(\alpha,\beta) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) h(x,\alpha,y,\beta)$$

h expresses how much the input value at (x,y) influences the output at (α,β), *and this influence can vary depending on* (x,y)

If however this influence depends only on relative positions then it is a **shift-invariant** PSF:

$$h(x, \alpha, y, \beta) = h(\alpha - x, \beta - y)$$

In other words we are dealing with a **linear shift invariant** system (**LSI**), thus leading to a familiar convolution:

$$g(\alpha, \beta) = (f * h)(\alpha, \beta) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) h(\alpha - x, \beta - y)$$

If we stack the image column by column as a vector of size $N^2 \times 1$:

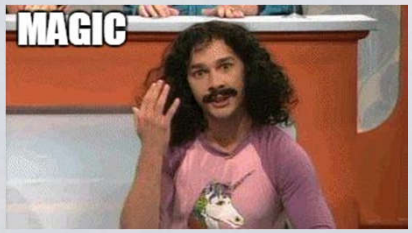
$$\boldsymbol{f} = [f(0,0), f(1,0), \dots, f(N-1,0), \dots, f(0,N-1), \dots, f(N-1,N-1)]^T$$

Then the linear operator can be expressed through the fundamental equation of linear image processing:

$$\boldsymbol{g} = \boldsymbol{H}\boldsymbol{f}$$

where H is a matrix of size $N^2 \times N^2$ and the input and output images are vectors in the N^2-dimensional vector space.

$$H = \begin{pmatrix} \alpha\downarrow\binom{x\rightarrow}{y=0,\ \beta=0} & \alpha\downarrow\binom{x\rightarrow}{y=1,\ \beta=0} & \cdots & \alpha\downarrow\binom{x\rightarrow}{y=N\text{-}1,\ \beta=0} \\ \alpha\downarrow\binom{x\rightarrow}{y=0,\ \beta=1} & \alpha\downarrow\binom{x\rightarrow}{y=1,\ \beta=1} & \cdots & \alpha\downarrow\binom{x\rightarrow}{y=N\text{-}1,\ \beta=1} \\ \vdots & \vdots & & \vdots \\ \alpha\downarrow\binom{x\rightarrow}{y=0,\ \beta=N\text{-}1} & \alpha\downarrow\binom{x\rightarrow}{y=1,\ \beta=N\text{-}1} & \cdots & \alpha\downarrow\binom{x\rightarrow}{y=N\text{-}1,\ \beta=N\text{-}1} \end{pmatrix}$$



Operator application as a **matrix product**, given $f = \begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix}$:

$$\begin{bmatrix} 0 & 0.25 & 0.25 & 0.25 & 0 & 0 & 0.25 & 0 & 0 \\ 0.25 & 0 & 0.25 & 0 & 0.25 & 0 & 0 & 0.25 & 0 \\ 0.25 & 0.25 & 0 & 0 & 0 & 0.25 & 0 & 0 & 0.25 \\ 0.25 & 0 & 0 & 0 & 0.25 & 0.25 & 0.25 & 0 & 0 \\ 0 & 0.25 & 0 & 0.25 & 0 & 0.25 & 0 & 0.25 & 0 \\ 0 & 0 & 0.25 & 0.25 & 0.25 & 0 & 0 & 0 & 0.25 \\ 0.25 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0.25 & 0.25 \\ 0 & 0.25 & 0 & 0 & 0.25 & 0 & 0.25 & 0 & 0.25 \\ 0 & 0 & 0.25 & 0 & 0 & 0.25 & 0.25 & 0.25 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$$

(9x9)x(9x1)

(for an image of dimension **512 x 512**, H will be **262144 x 262144**; not practical)

By the way, what does this filter do if the image is repeated in all directions?

Operator application as a **convolution**:

$$\begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix} * \begin{bmatrix} 0 & 0.25 & 0 \\ 0.25 & 0 & 0.25 \\ 0 & 0.25 & 0 \end{bmatrix}$$

(3x3)*(3x3)

$$\begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix} \quad \begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ \mathbf{2} & 5 & 8 \end{bmatrix} \quad \begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 3 & \mathbf{6} \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix} \quad \begin{bmatrix} 0 & \mathbf{3} & 6 \\ \mathbf{1} & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix} \quad \begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix} \quad \begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix} \quad \begin{bmatrix} 0 & 3 & 6 \\ 1 & 4 & 7 \\ 2 & 5 & 8 \end{bmatrix}$$

If the columns of an image are influenced independently from the rows of an image then the PSF is **separable**:

$$h(x,\alpha,y,\beta) = h_Y(x,\alpha)h_X(y,\beta)$$

$$g(\alpha,\beta) = \sum_{x=0}^{N-1} h_Y(x,\alpha) \sum_{y=0}^{N-1} f(x,y)h_X(y,\beta)$$

Or in matrix form: $\boldsymbol{g = H_Y{}^T f H_X}$

$$\mathbf{H}_y = \begin{bmatrix} h_y[0,0] & h_y[0,1] & \cdots & h_y[0,L_g-1] \\ h_y[1,0] & h_y[1,1] & \cdots & h_y[1,L_g-1] \\ \vdots & \vdots & & \vdots \\ h_y[L-1,0] & h_y[L-1,1] & \cdots & h_y[L-1,L_g-1] \end{bmatrix}$$

$$\mathbf{H}_x = \begin{bmatrix} h_x[0,0] & h_x[0,1] & \cdots & h_x[0,N_g-1] \\ h_x[1,0] & h_x[1,1] & \cdots & h_x[1,N_g-1] \\ \vdots & \vdots & & \vdots \\ h_x[N-1,0] & h_x[N-1,1] & \cdots & h_x[N-1,N_g-1] \end{bmatrix}$$

where f and g are of size L x N and the operators are L x L and N x N; much smaller!

The operator matrices will be square if they don't change image size.

If the linear operator is **separable** AND **shift invariant**, then its application to the image becomes a **cascade of two 1D convolutions**:

$$g(\alpha,\beta) = \left({h_Y}^T * f * h_X\right)(\alpha,\beta) = \sum_{x=0}^{N-1} h_Y(\alpha - x) \sum_{y=0}^{N-1} f(x,y) h_X(\beta - y)$$

where h_Y and h_X are vectors of length N and $h = h_Y \otimes h_X$ is their outer product.

Examples of separable convolution masks

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} [1 \quad -2 \quad 1] \qquad \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} [1/3 \quad 1/3 \quad 1/3]$$

TL;DR; If your operator is both **linear** and **shift invariant** (i.e. a **LSI** system) then its output is the input's convolution with the operator's PSF. They can also be expressed as matrix products. Linear image processing is all about LSI systems and convolutions.

Spatial domain filtering

The PSF matrix is known as **filter**, **mask**, **kernel**, subimage, template, window, etc.

Given a convolution filter h of dimensions $(J \times K)$ its coordinate system is assumed as:

$$\mathbf{h} = \begin{bmatrix} h\left[-\left(J-\frac{1}{2}\right),-\left(K-\frac{1}{2}\right)\right] & \cdots & \cdots & h\left[0,-\left(K-\frac{1}{2}\right)\right] & \cdots & \cdots & h\left[\left(J-\frac{1}{2}\right),-\left(K-\frac{1}{2}\right)\right] \\ \vdots & \ddots & \vdots & \vdots & \vdots & \iddots & \vdots \\ \vdots & \cdots & h[-1,-1] & h[0,-1] & h[1,-1] & \cdots & \vdots \\ h\left[-\left(J-\frac{1}{2}\right),0\right] & \cdots & h[-1,0] & h[0,0] & h[1,0] & \cdots & h\left[\left(J-\frac{1}{2}\right),0\right] \\ \vdots & \cdots & h[-1,1] & h[0,1] & h[1,+1] & \cdots & \vdots \\ \vdots & \iddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ h\left[-\left(J-\frac{1}{2}\right),\left(K-\frac{1}{2}\right)\right] & \cdots & \cdots & h\left[0,\left(K-\frac{1}{2}\right)\right] & \cdots & \cdots & h\left[\left(J-\frac{1}{2}\right),\left(K-\frac{1}{2}\right)\right] \end{bmatrix}$$

Its values are referred to as **coefficients**; they usually sum up to 1, as the PSF has usually an integral of 1 ⇒preserves the image's mean.

The discrete 2D Convolution of an image f of dimension $N \times N$ with a filter h of dimension $K \times L$:

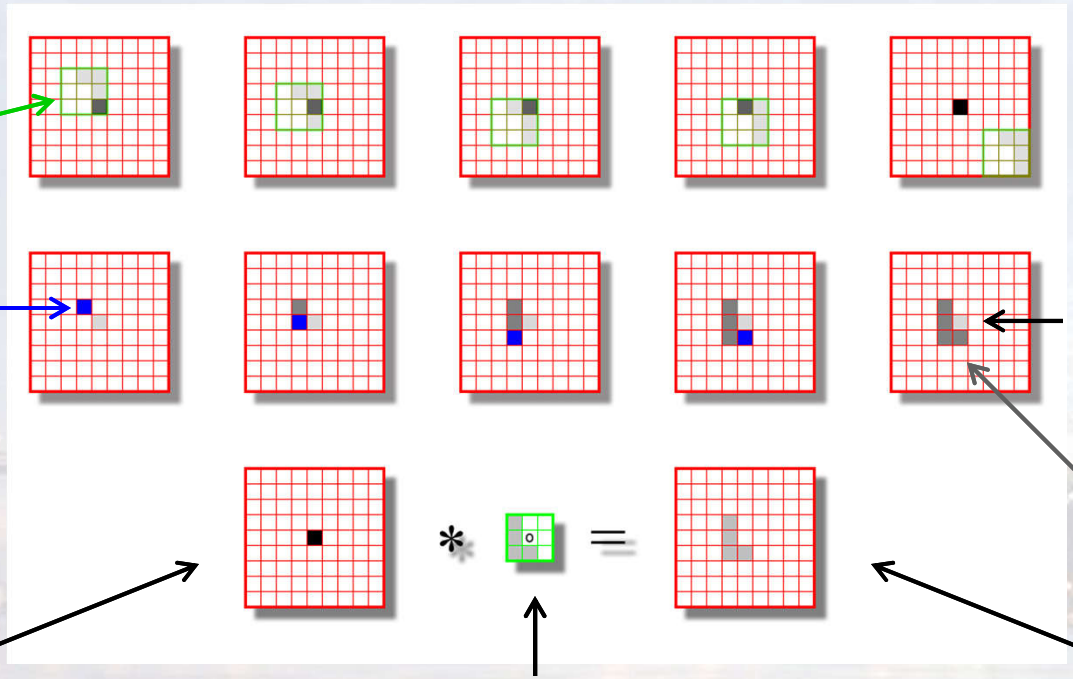
$$(f * h)(\alpha, \beta) = \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) h(\alpha - x, \beta - y) = \sum_{x=-k}^{k} \sum_{y=-l}^{l} h(x,y) f(\alpha - x, \beta - y)$$

where $k = K/2, l = L/2$

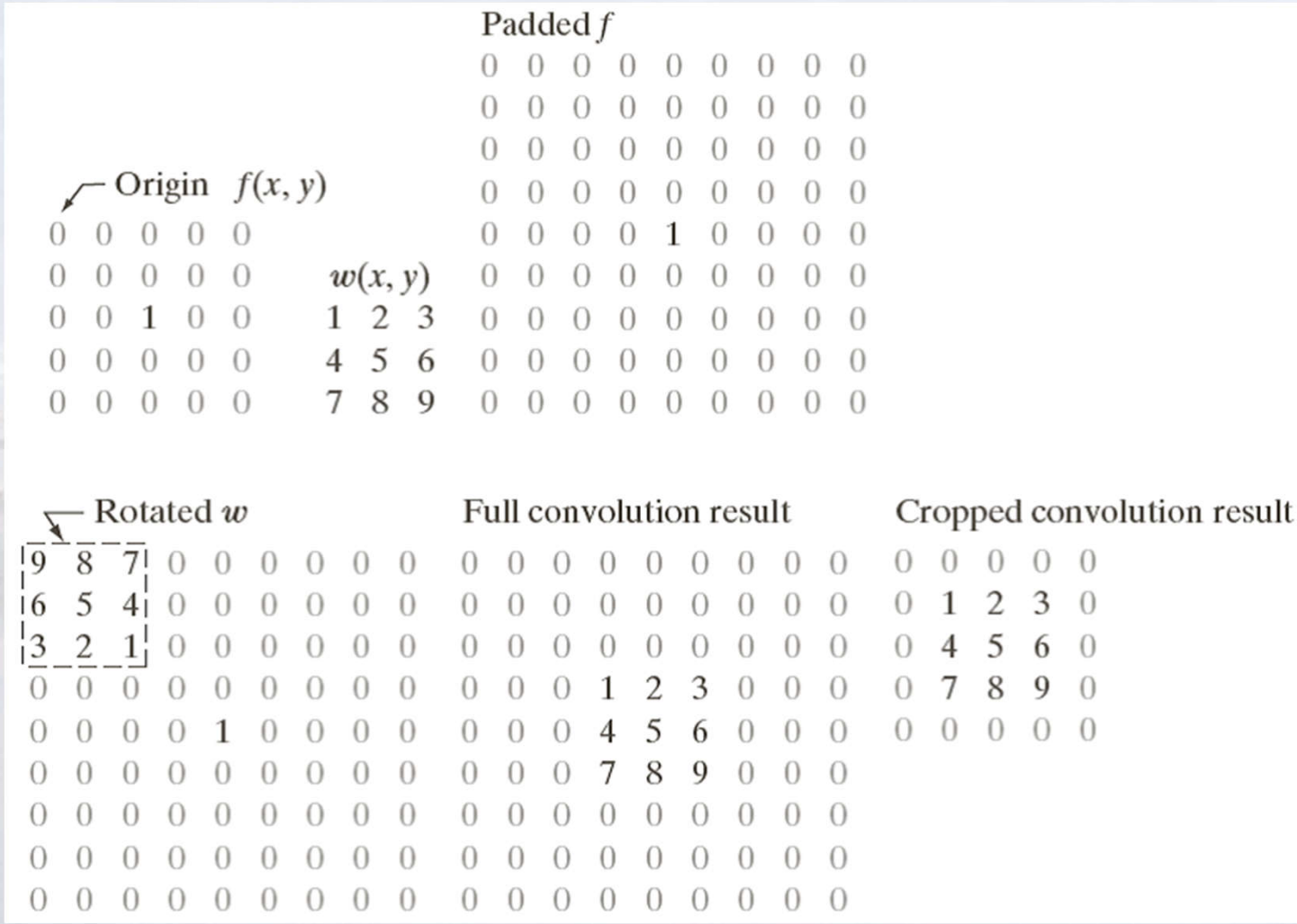
The output will have dimensions $(N + K - 1) \times (N + L - 1)$

It basically produces a linear combination of pixel neighborhoods as output.

Example


Shifted weight matrix
Result of sum of products
At the locations not shown, the results were zeros.
Location of impulse
Accumulated nonzero results
The original image has a black impulse at the center and zeros (white) elsewhere.
The weight matrix has a gray 'L' at its left and zeros (white) elsewhere.
The resulting image has a copy of the weight matrix pegged to the impulse location.

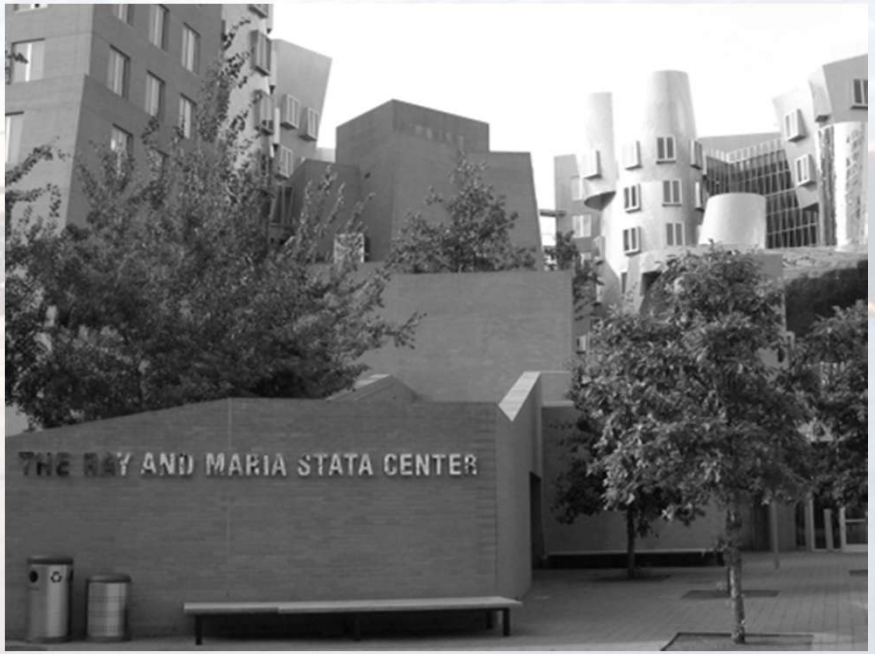
2D discrete convolution in practice



Real world convolution examples: camera shake

=

*



Source: Fergus, *et al.* "Removing Camera Shake from a Single Photograph", SIGGRAPH 2006

Linear filter example

*

0	0	0
0	1	0
0	0	0

=

Original

Identical…

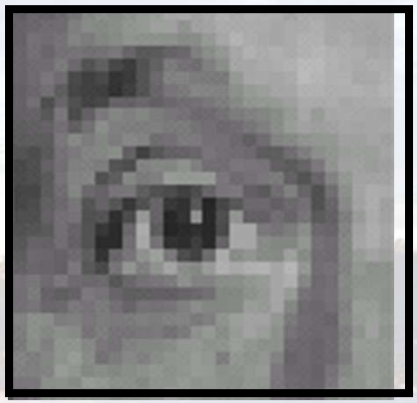
Linear filter example

Original

$*$

0	0	0
1	0	0
0	0	0

$=$



Shifted left by 1 pixel

What happens at the borders of the image?

- You can fill in the missing pixels with a constant; e.g. zero
- You can replicate the last row or column before the missing area, as many times as needed
- You can wrap the image, assuming it is periodical; e.g. column[-1] = column[width-1], column[-2] = column[width-2],etc.
- You can reflect/mirror the image; e.g. column[-1] = column[1], column[-2] = column[2], etc.

Convolution properties

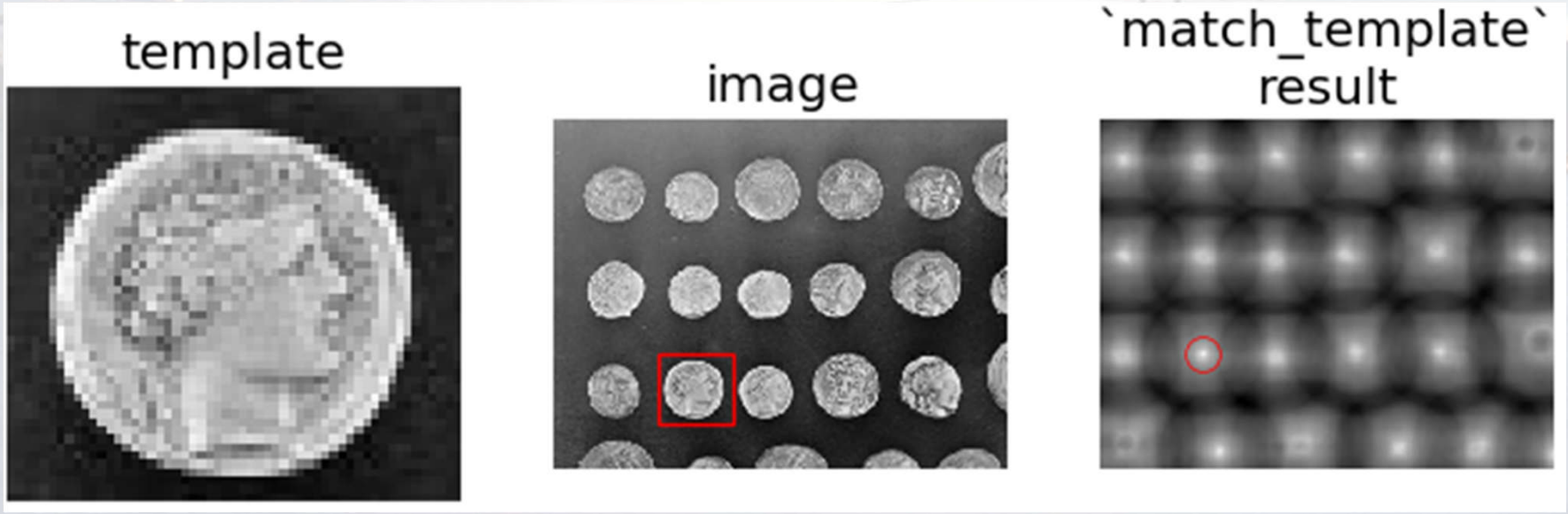
$f * h = h * f$: commutativity

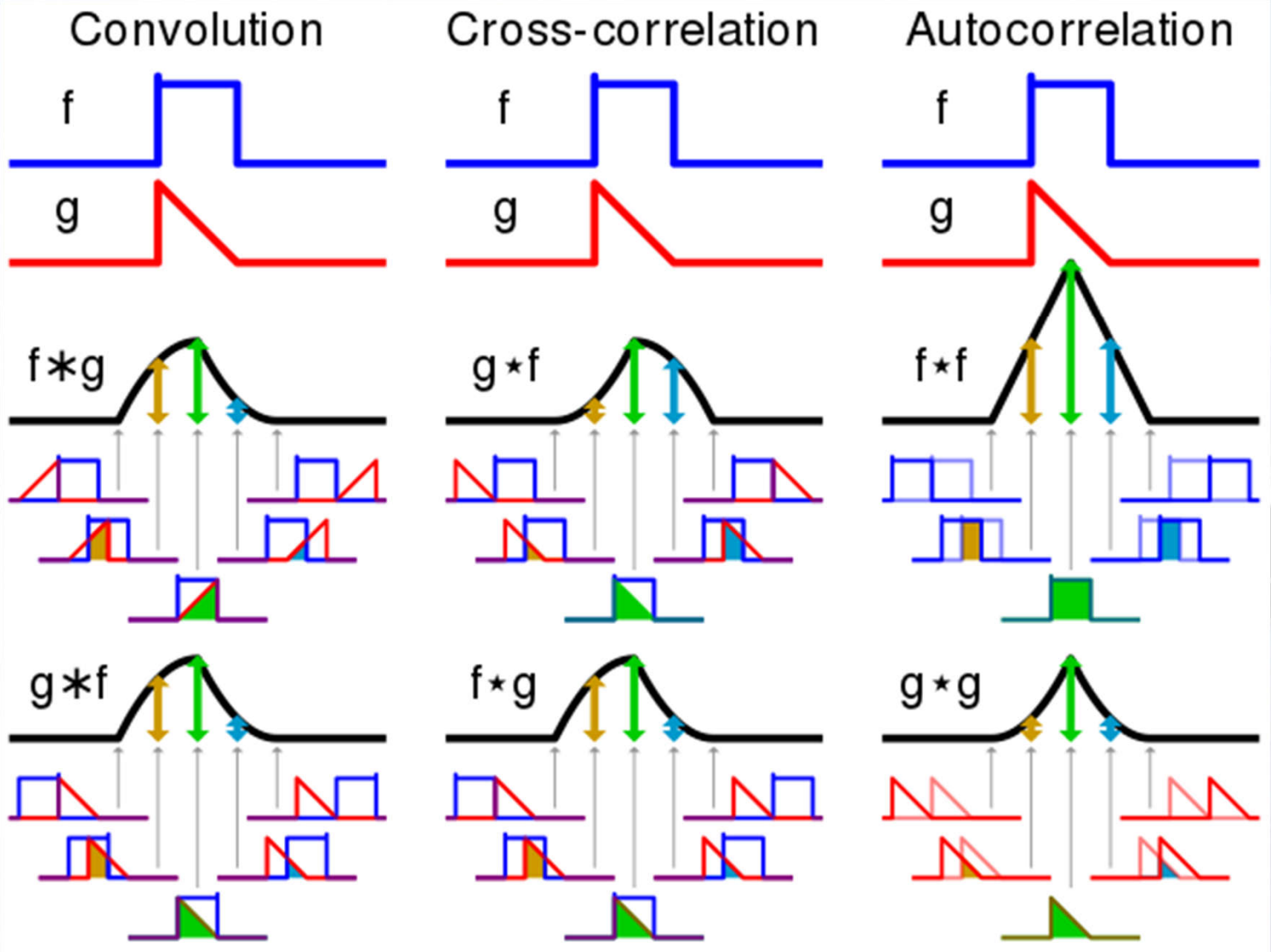
$(f * h) * g = f * (h * g)$: associativity

$f * \delta = f$: multiplicative identity

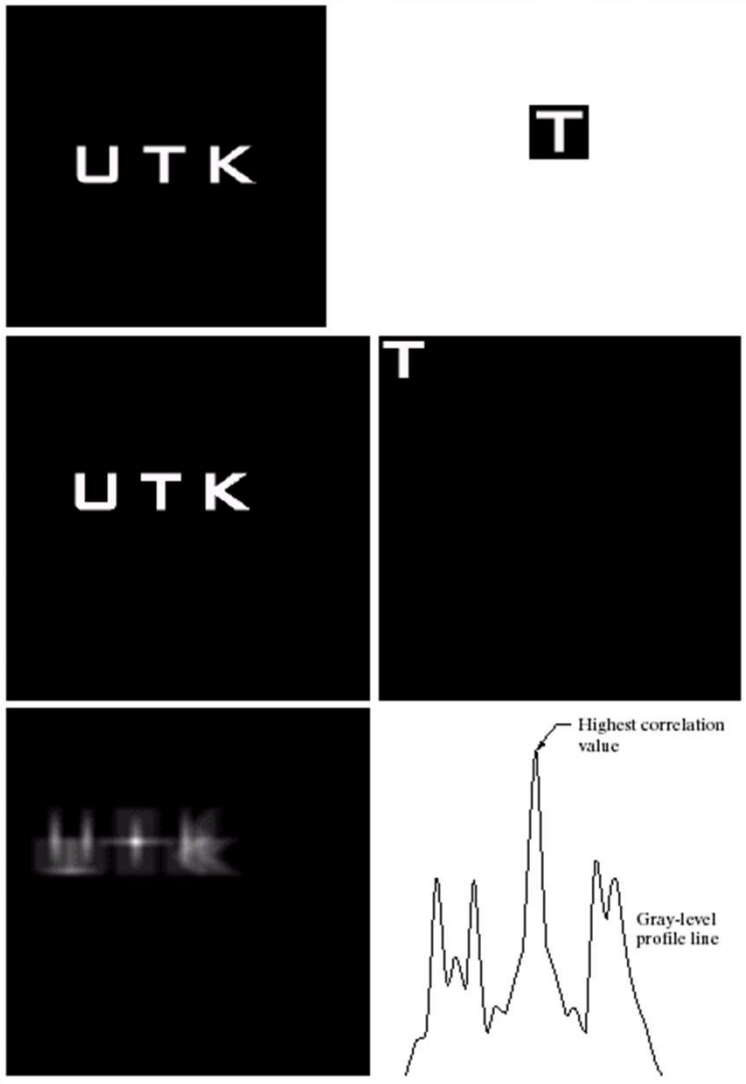
Cross-correlation: same as convolution but without the reflection: $f \star h$

Autocorrelation: an image's cross-correlation with itself: $f \star f$




Convolution
Cross-correlation
Autocorrelation
f
g
f∗g
g⋆f
f⋆f
g∗f
f⋆g
g⋆g

Cross-correlation example



a b
c d
e f

FIGURE 4.41
(a) Image.
(b) Template.
(c) and
(d) Padded images.
(e) Correlation function displayed as an image.
(f) Horizontal profile line through the highest value in (e), showing the point at which the best match took place.

Templates for face detection

And the detected faces

The convolution of an image with a $K \times L$ mask has a complexity of $O(KL)$ **per pixel**.

Remember **separability**!

Separable PSF ⇒ separable filter

$$h = h_C \cdot [h_R]^T$$
$$(J \times K) = (J \times 1) \cdot (1 \times K)$$

2D convolution becomes equivalent to a cascade of 2 1D convolutions

$$g(\alpha, \beta) = \sum_{x=0}^{N-1} h_C(\alpha - x) \sum_{y=0}^{N-1} f(x, y) h_R(\beta - y)$$

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix} [1/3 \quad 1/3 \quad 1/3]$$

$$\begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} [1 \quad -2 \quad 1]$$

Complexity reduces to $O(K + L)$

Sharpening filters
Good for highlighting contours

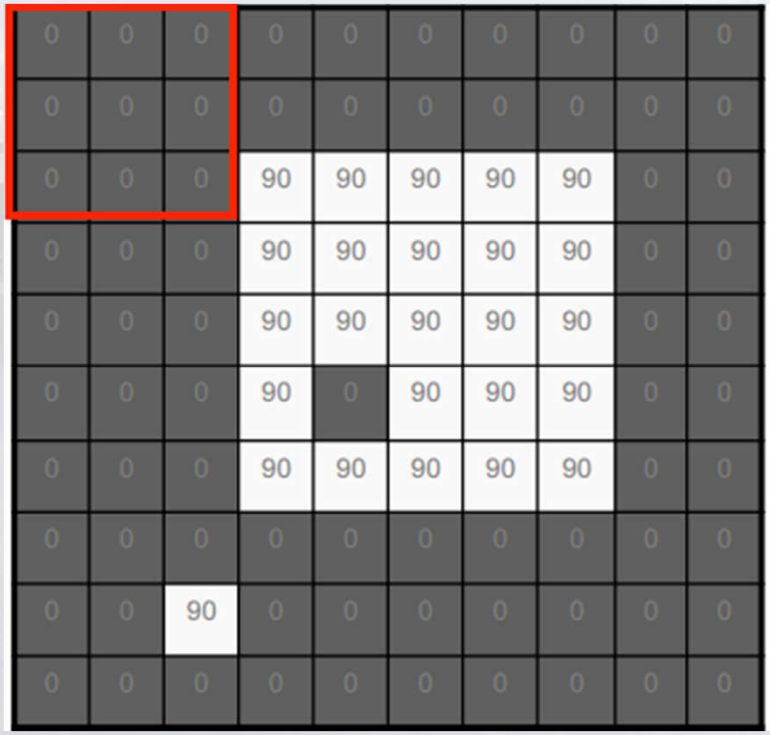
Smoothing filters
Good for blurring and noise reduction

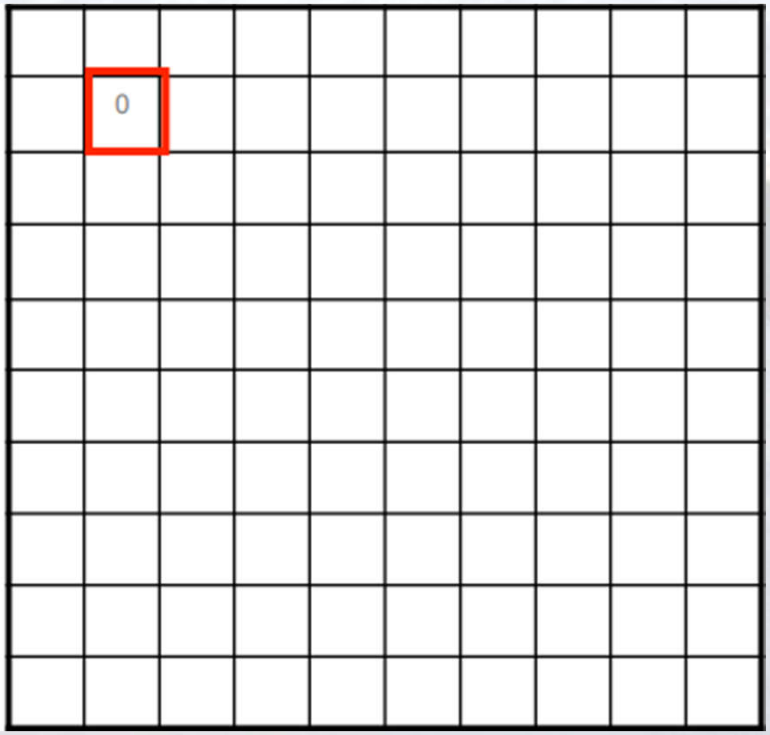
The mean (box) filter is a simple smoothing filter.

Its output is the average of the neighborhood.

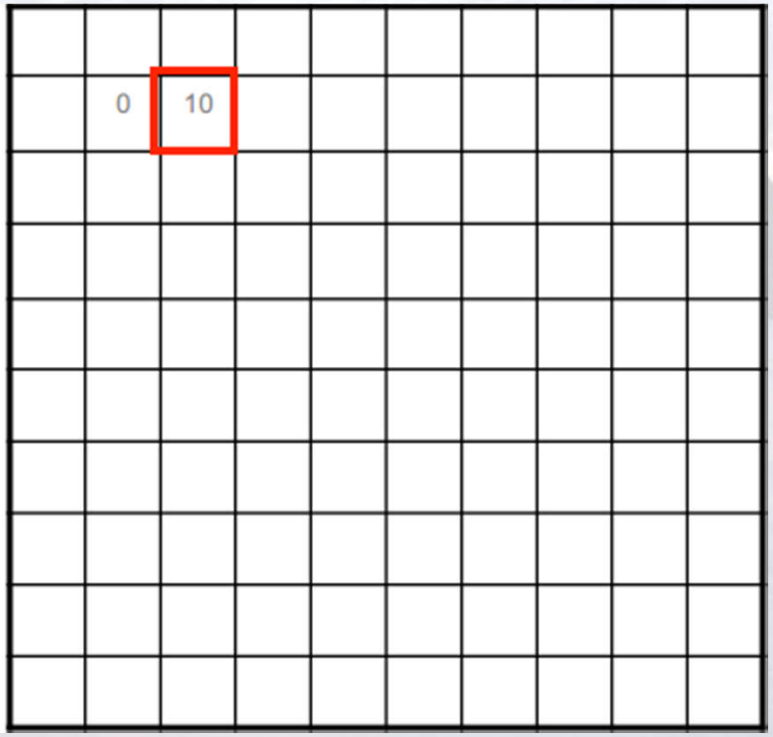
It's **isotropic**, so there is no need for reflection.

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

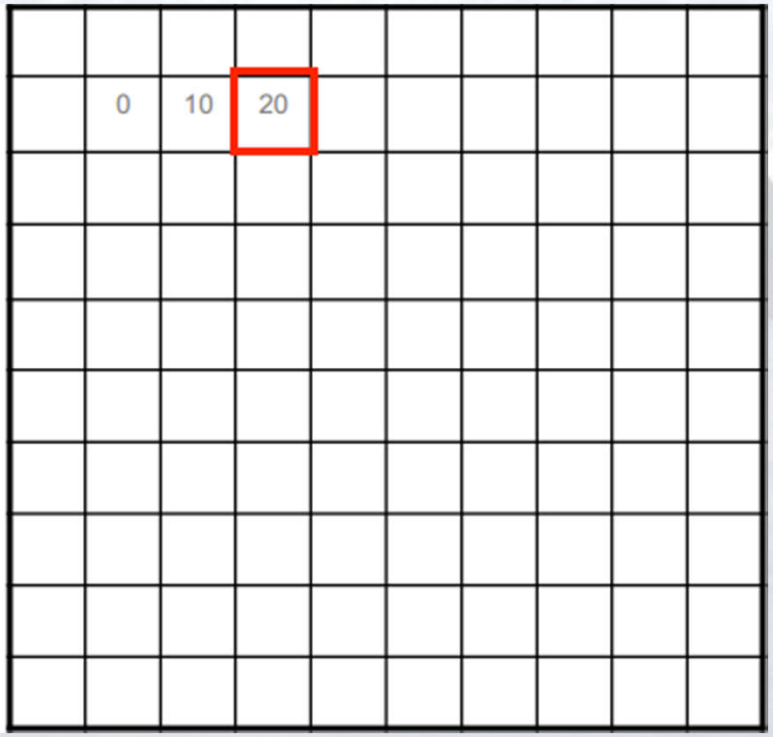




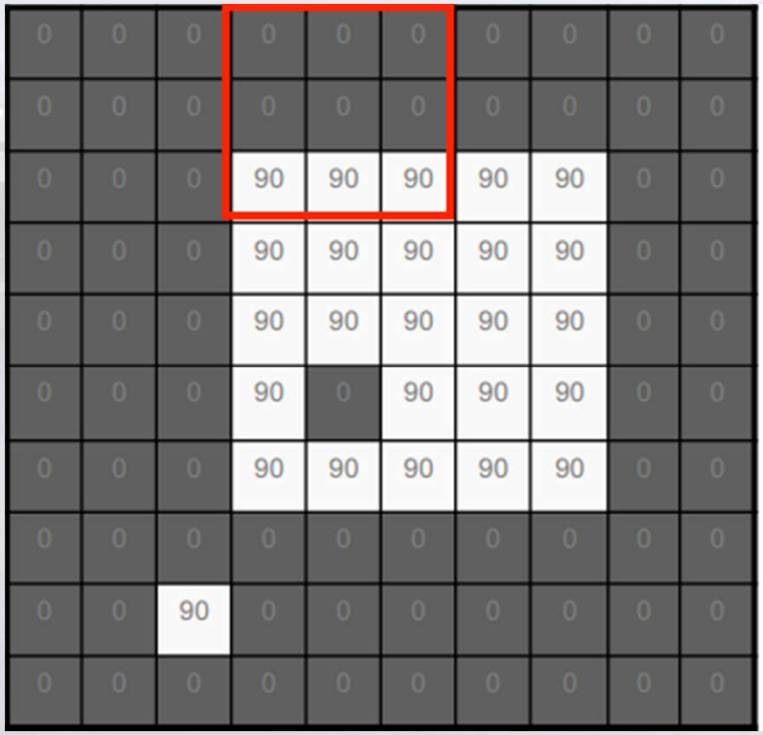
$$\frac{1}{9} \times \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

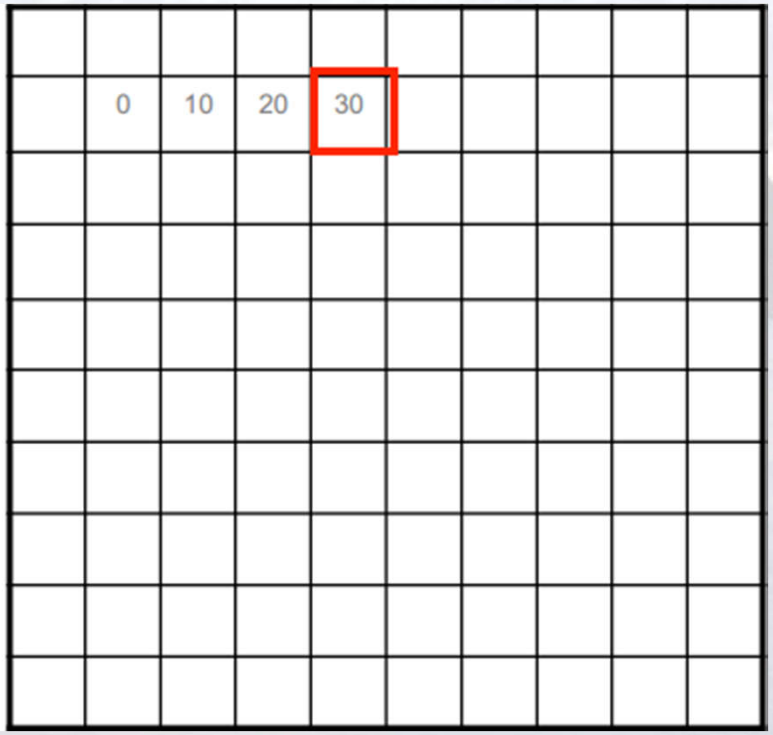


$$\frac{1}{9}\times\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$



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





Linear image processing

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

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

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

Original

500x500

n = 3

n = 5

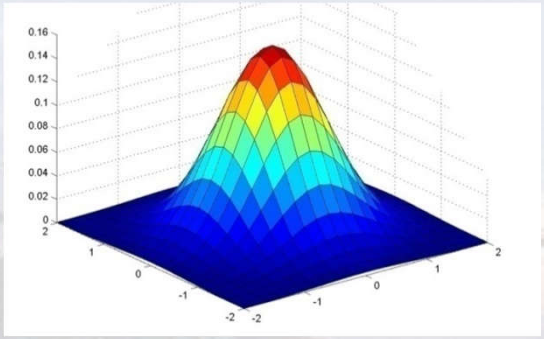
n = 9

n = 15

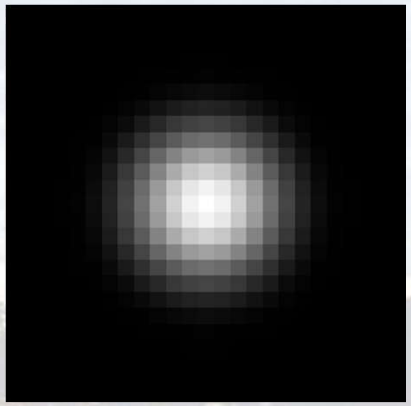
n = 35

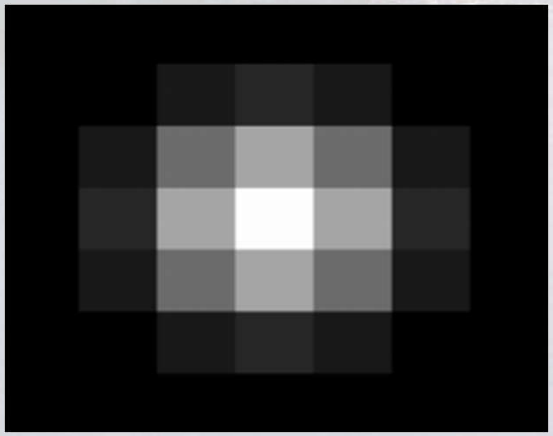
The mean filter attributes equal weight to its coefficients. What if we use a different distribution of weights, that favors the center? → **Gaussian smoothing**.

The Gaussian filter uses a discrete approximation of a 2D Gaussian function



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



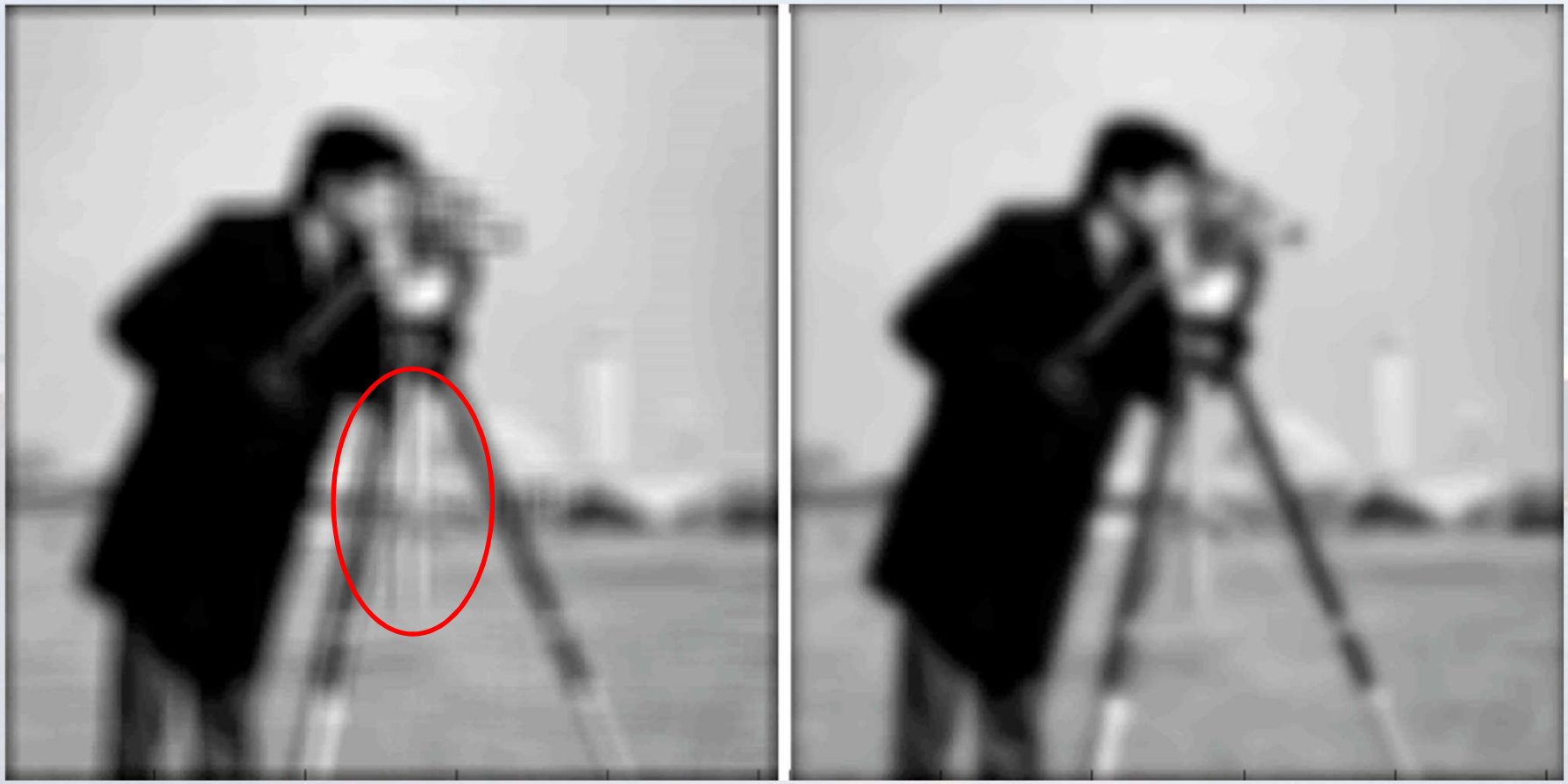


5x5 Gaussian kernel

example with $\sigma = 1$

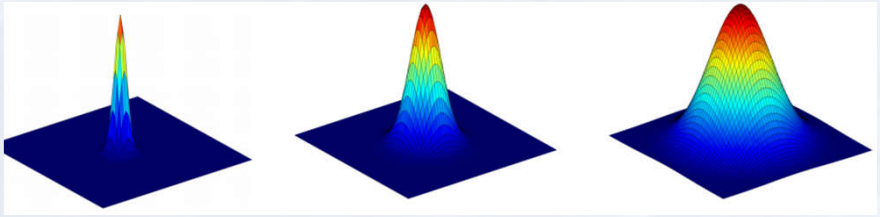
$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

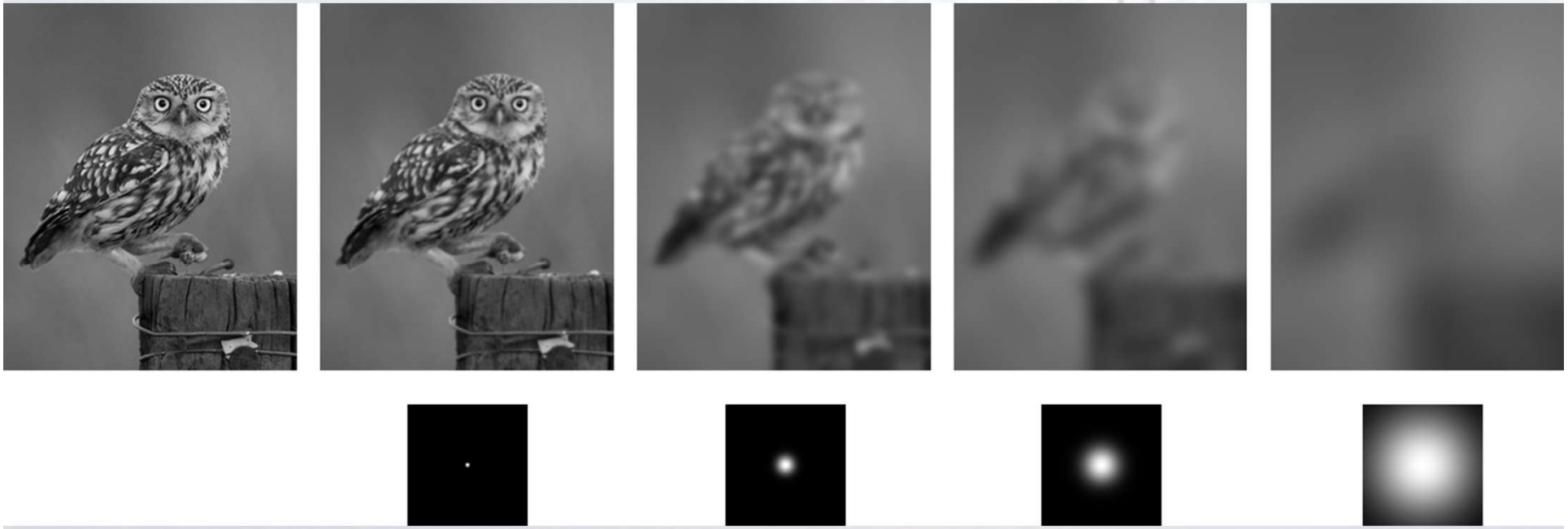
Notice the ringing effect of the mean filter (left) vs Gaussian (right)



Properties of the Gaussian filter

- Larger σ → wider peak → greater blurring
- Kernel size needs to also increase along with σ in order to maintain the Gaussian nature of the filter.





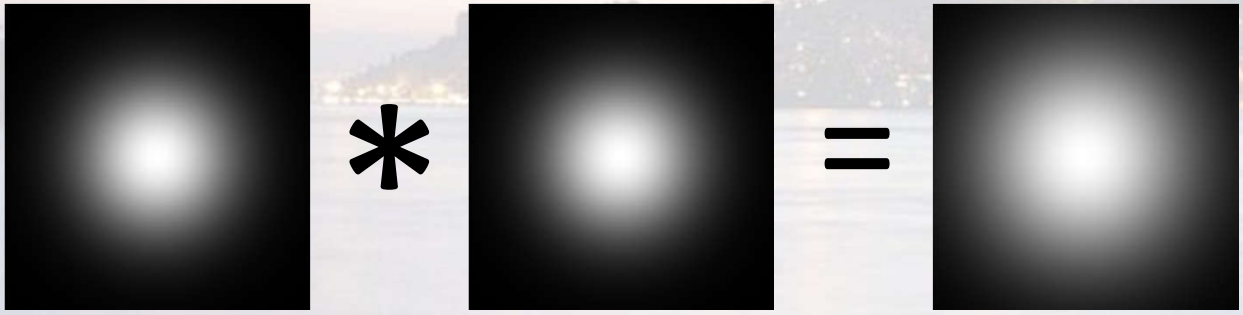
More properties of the Gaussian filter

- The kernel is isotropic, i.e. rotationally symmetric
- The kernel is separable

$$f * G = f * G_x * G_y$$

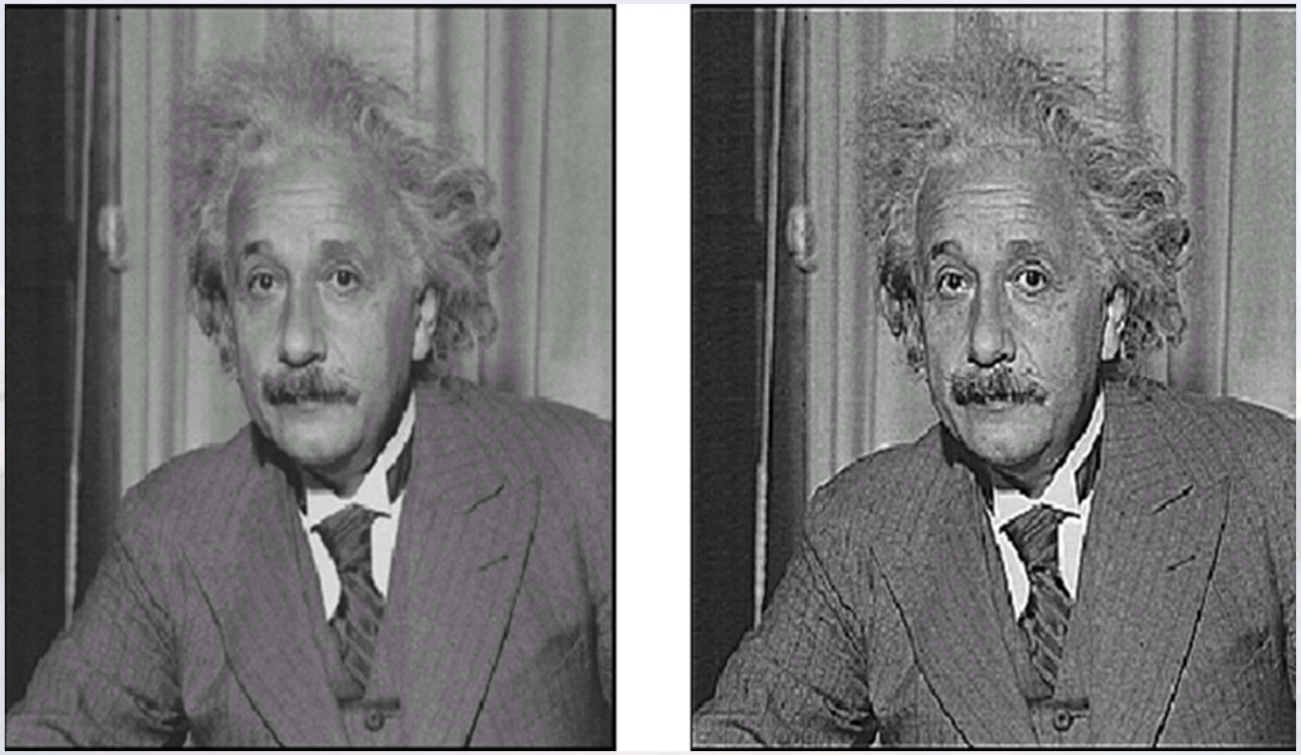
$$\begin{aligned} G_\sigma(x,y) &= \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{x^2}{2\sigma^2}\right)\cdot\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{y^2}{2\sigma^2}\right) \\ &= g_\sigma(x)\cdot g_\sigma(y) \end{aligned}$$

- Convolving a Gaussian with a Gaussian is still a Gaussian



- Convolving twice with a Gaussian kernel of σ is identical to convolving once with a kernel of $\sigma\sqrt{2}$

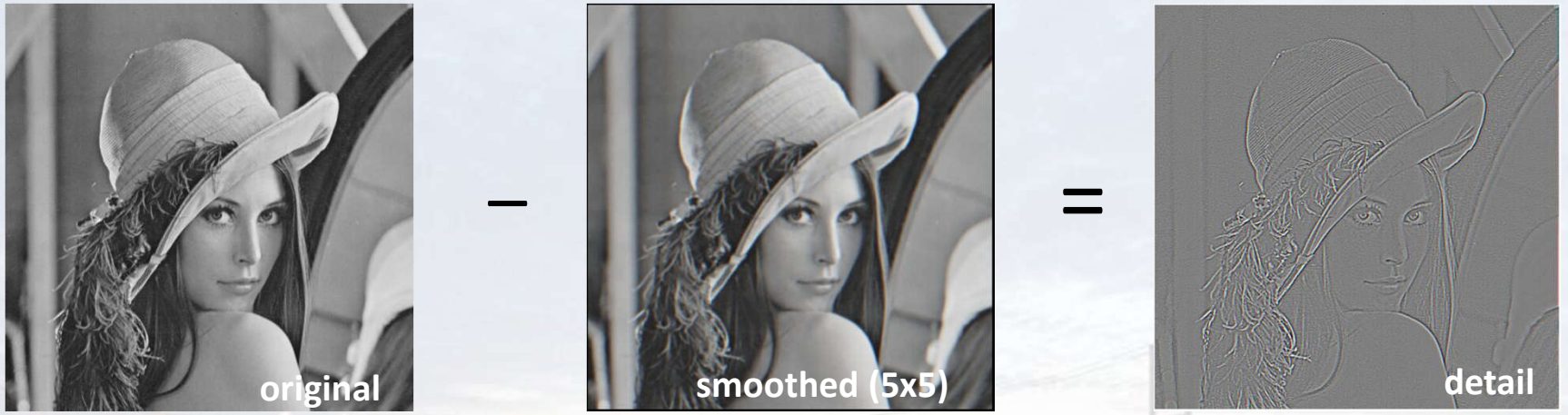
Sharpening is about highlighting contours, making them more visible.



Source: D. Lowe

What does blurring take away?

Source: S. Lazebnik



What if you add it back?

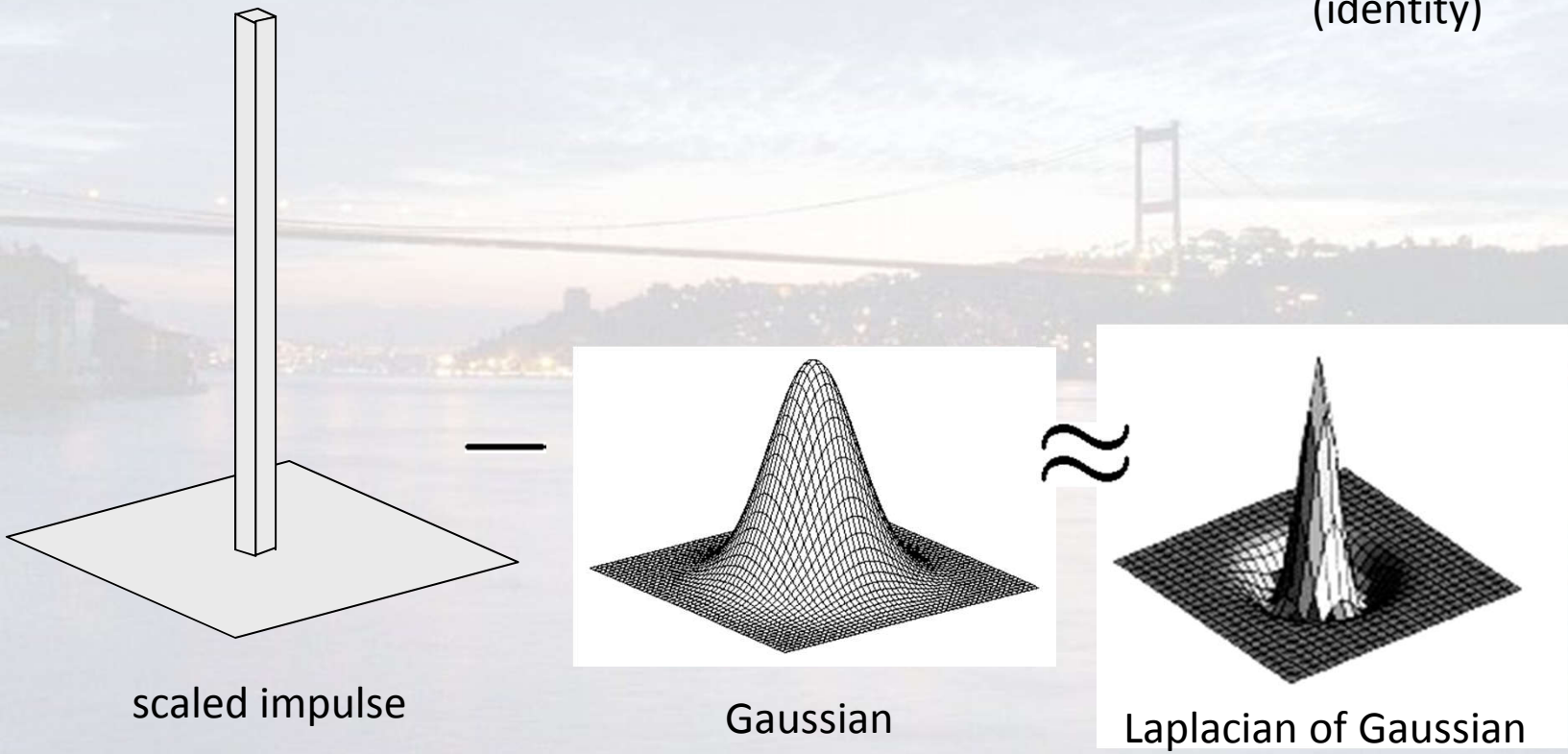
Sharpening through **unsharp masking**

$$F + \alpha\,(F - F * H) = (1+\alpha)\,F - \alpha\,(F * H) = F * ([1+\alpha]e - H)$$

image

blurred image

unit impulse (identity)



scaled impulse

Gaussian

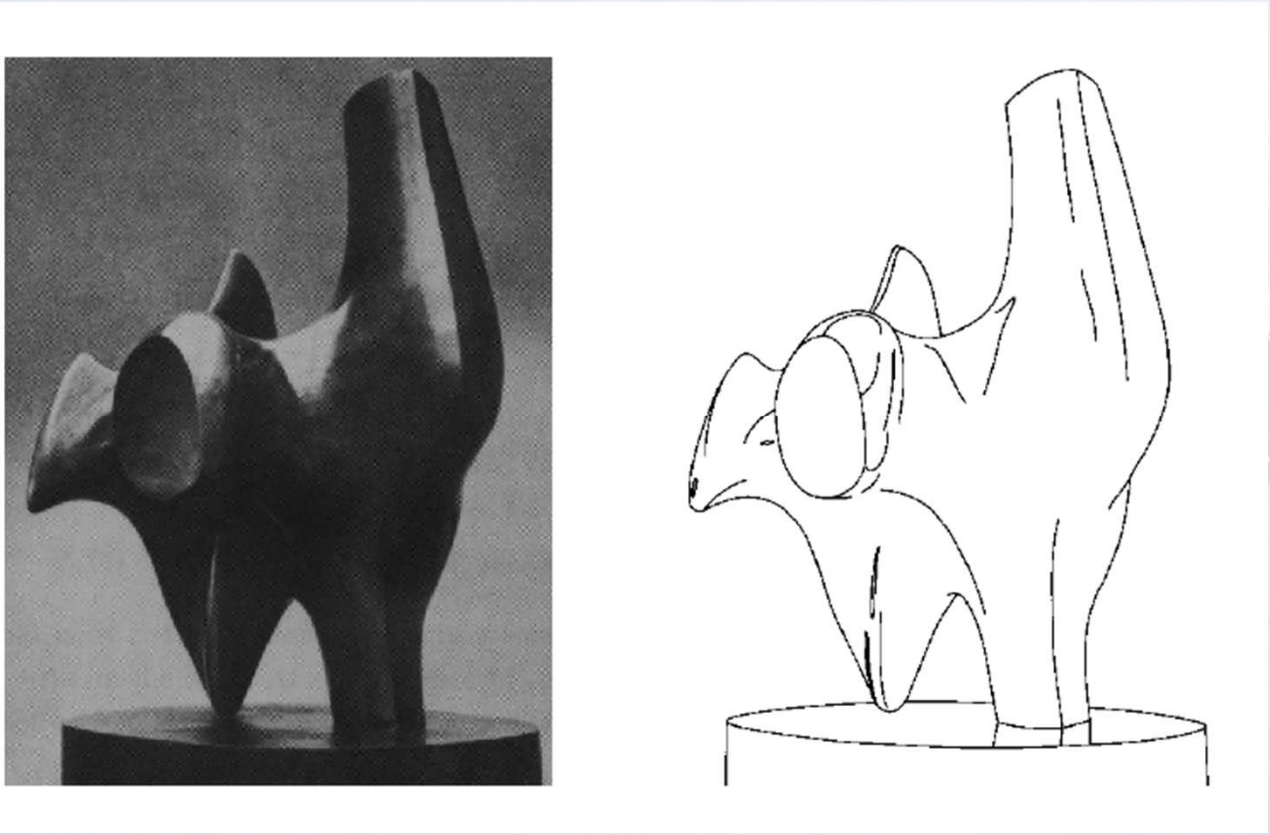
Laplacian of Gaussian

Sharpening through **unsharp masking**

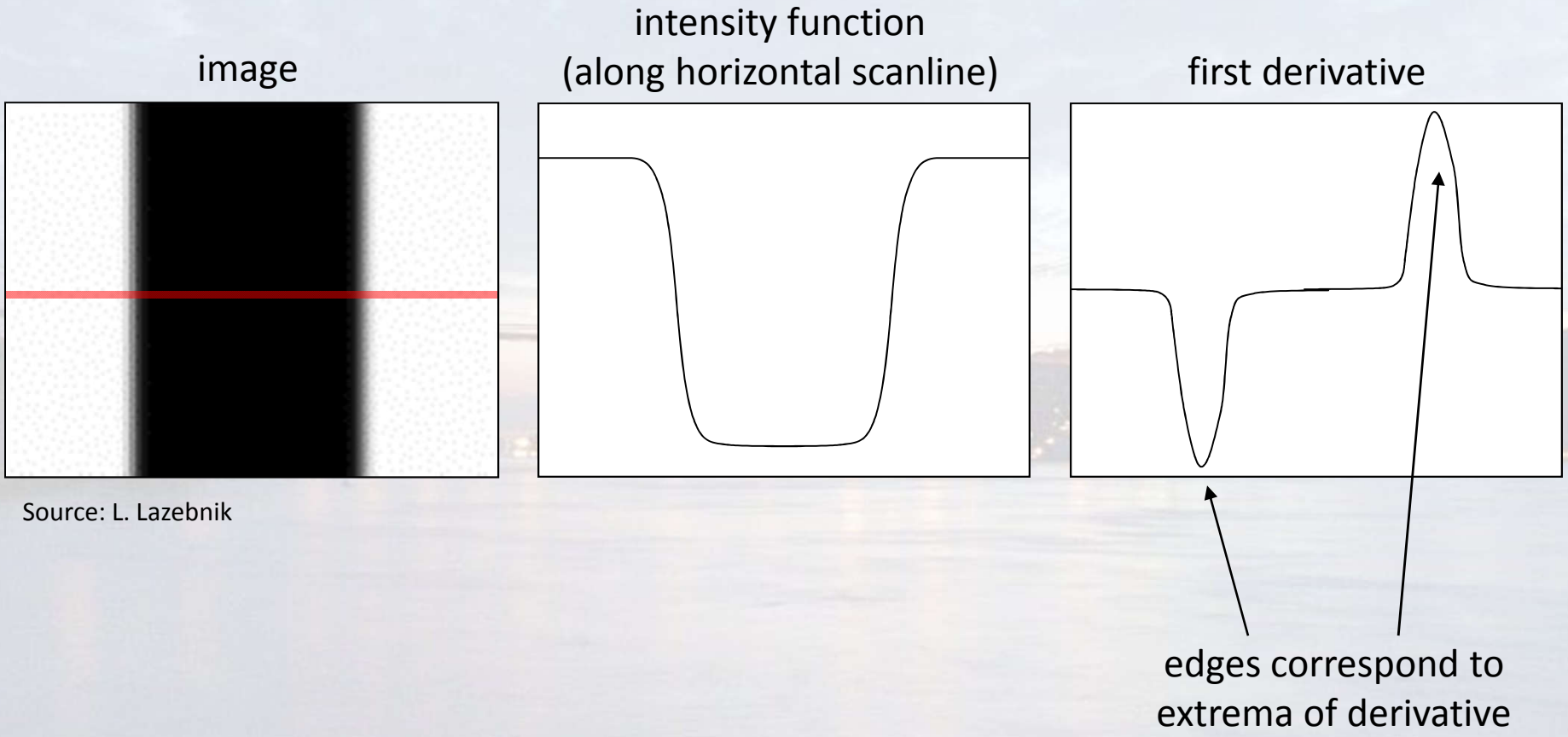
unfiltered

filtered

Edge detection converts an image into a set of curves, that correspond to locations of “significant” pixel intensity change.



And since edges are locations of variation, they can be detected through **derivatives**.



Source: L. Lazebnik

- In more detail, the **gradient** of image f is denoted as

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

- The strength of an edge is given by the gradient magnitude

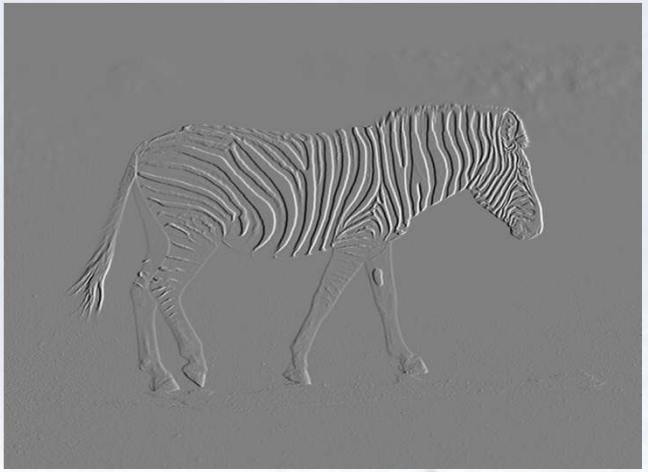
$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

- The gradient direction is given by

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

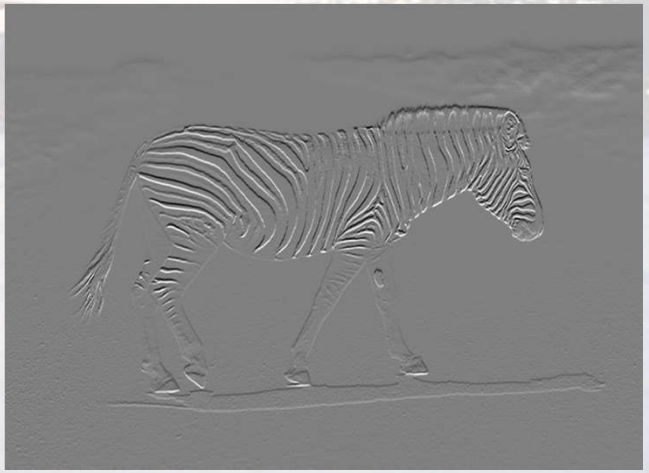
Linear image processing

f



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

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



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

How to calculate these in the **discrete case**? With finite differences!

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

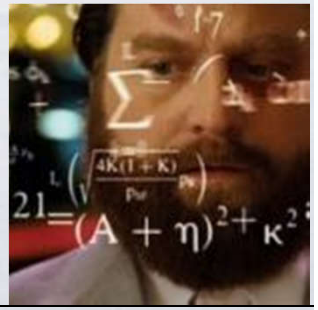
Converted into filters: $[+1 \quad -1]$ for horizontal and $\begin{bmatrix}+1\\-1\end{bmatrix}$ for vertical edges.

And $\begin{bmatrix}+1 & 0\\ 0 & -1\end{bmatrix}\begin{bmatrix}0 & -1\\ +1 & 0\end{bmatrix}$ for diagonal edges (**Robert's operator, 1963**)

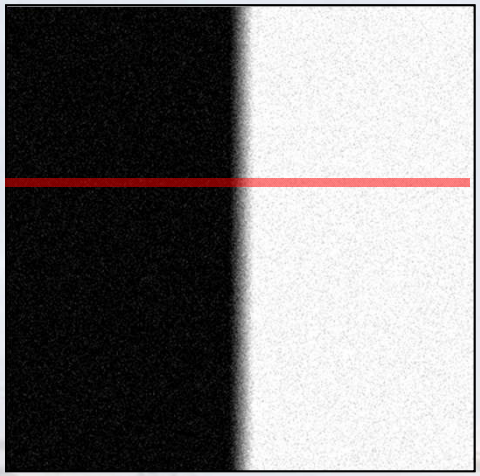
BUT! Even length mask → its center is associated with a non-integer position ☹

Resolved by **Prewitt**: $\begin{bmatrix}-1 & 0 & +1\\ -1 & 0 & +1\\ -1 & 0 & +1\end{bmatrix}\begin{bmatrix}-1 & -1 & -1\\ 0 & 0 & 0\\ +1 & +1 & +1\end{bmatrix}$

They are separable and fast!

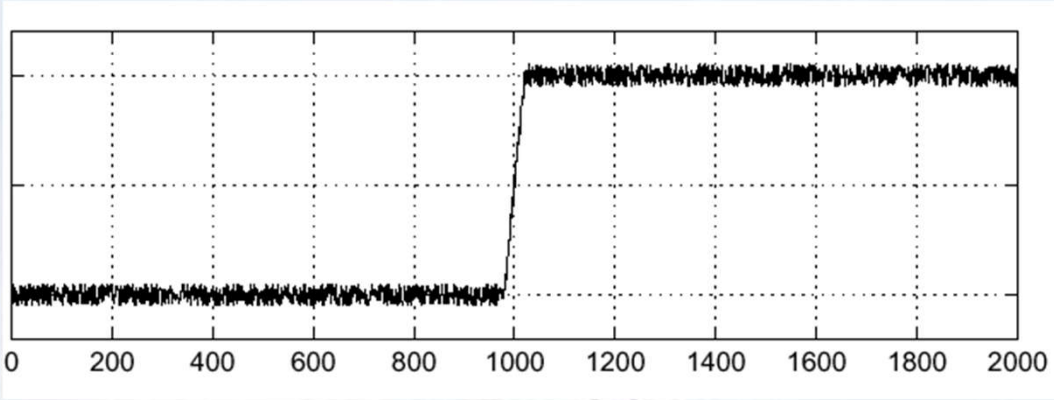


It’s all beautiful in theory, but then there is noise:

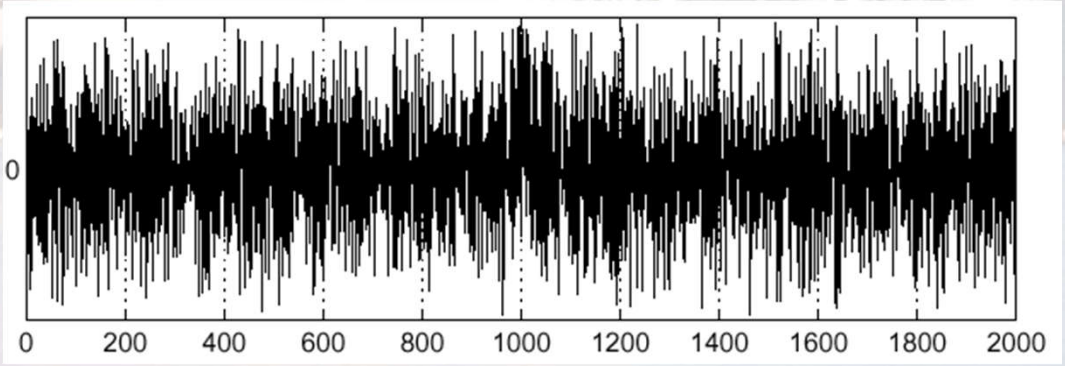


Noisy input image

$f(x)$



$\frac{d}{dx}f(x)$



Source: S. Seitz

Where is the edge?

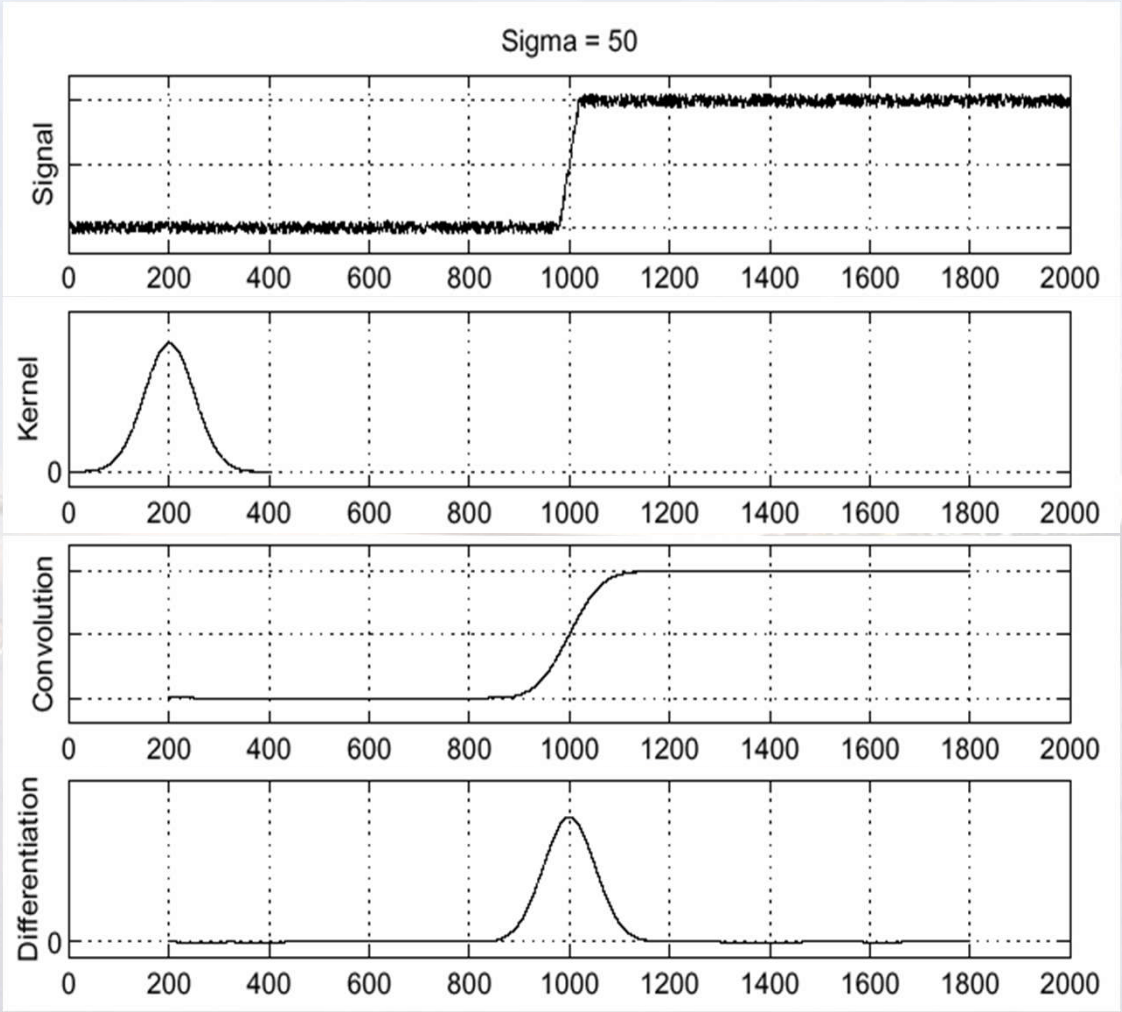
It makes sense to smooth first before computing the differences

f

h

$f * h$

$$\frac{d}{dx}(f * h)$$

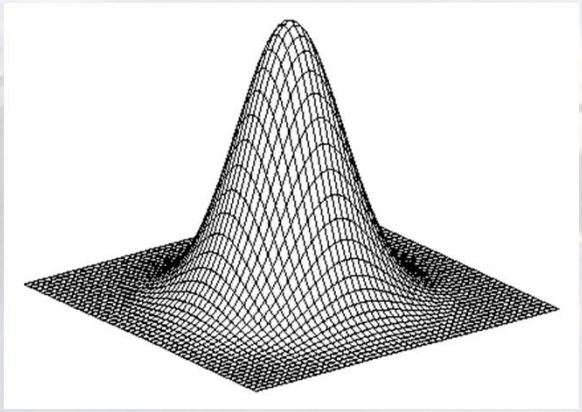


Source: S. Seitz

Remember that convolution is associative and differentiation is also a convolution, so:

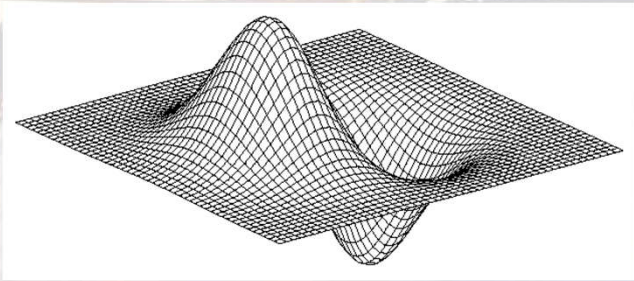
$$\frac{d}{dx}(f * h) = f * \frac{d}{dx}h$$

i.e. differentiate the smoothing filter and convolve only once with that; two convolutions for the price of one!



Gaussian

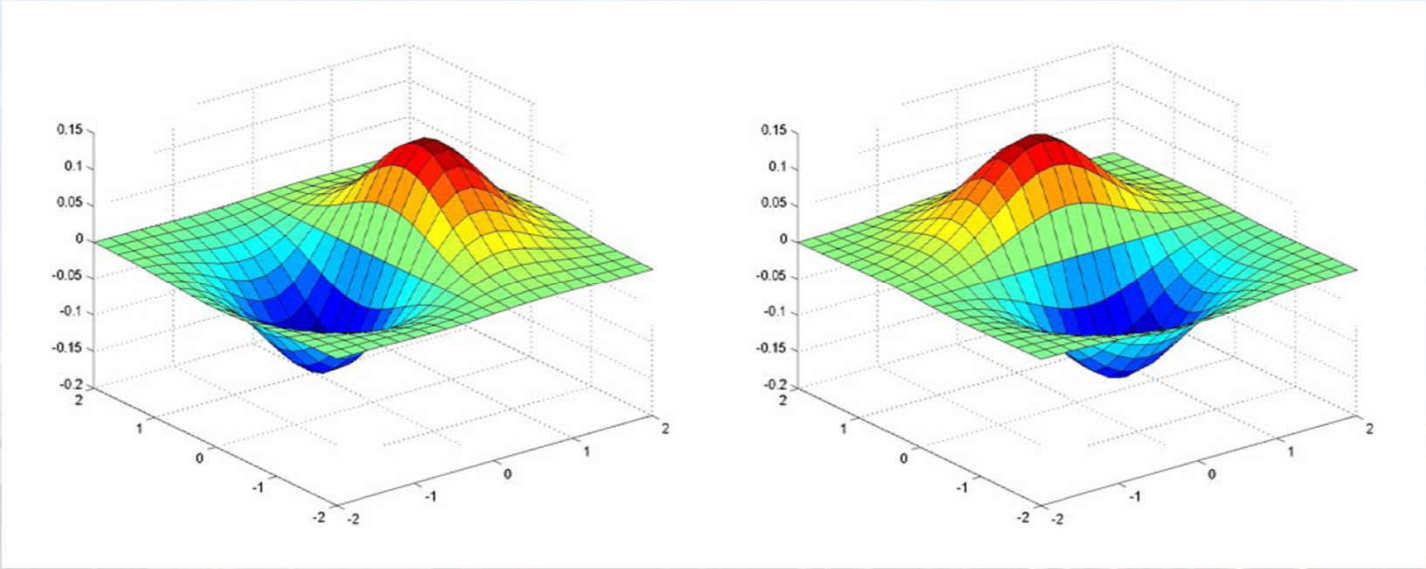
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2}e^{-\frac{u^2+v^2}{2\sigma^2}}$$



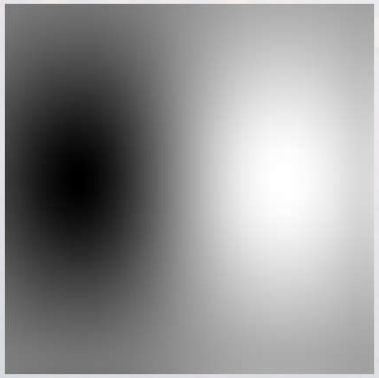
derivative of Gaussian (x)

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

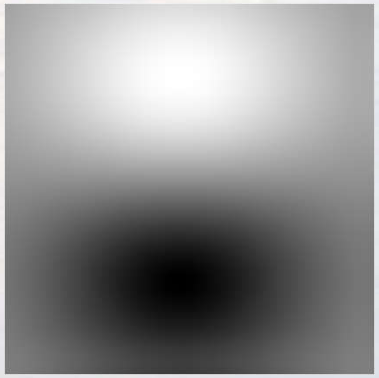
Derivative of the Gaussian filter



x-direction



y-direction



Approximated by the Sobel operator

$\frac{1}{8}$

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

s_x

$\frac{1}{8}$

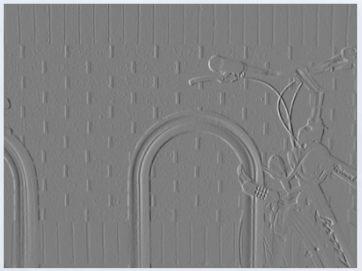
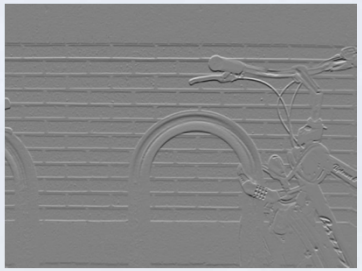
1	2	1
0	0	0
-1	-2	-1

s_y

The standard definition doesn't have the 1/8 term, but it is necessary in order to get the correct gradient value. Why 3x3? Because larger dimensions blur the edges.

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

Linear image processing

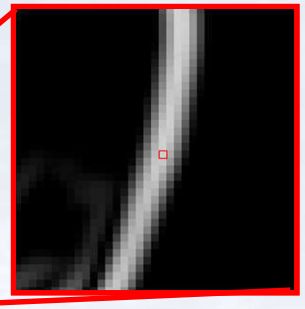
Source: Wikipedia

Canny edge detector (1986): still popular but has many parameters.

Original image: Lenna

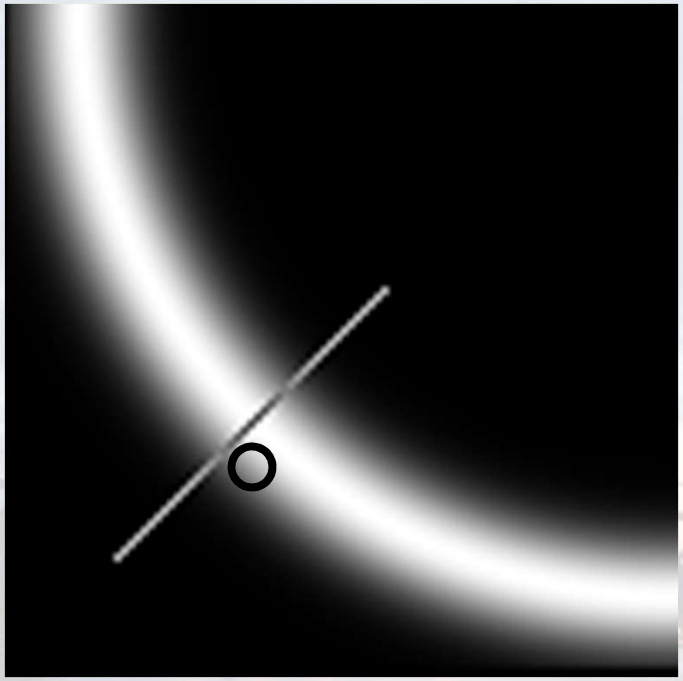
Its gradient magnitude

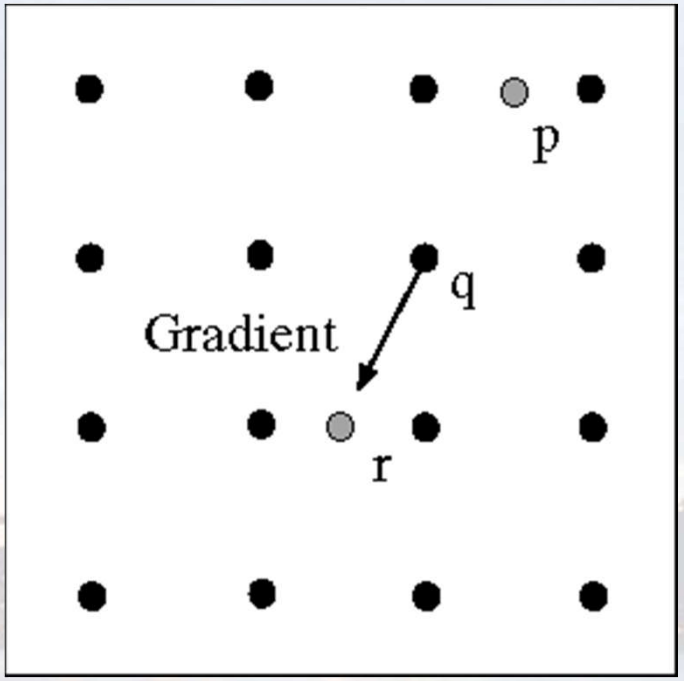
Thresholded

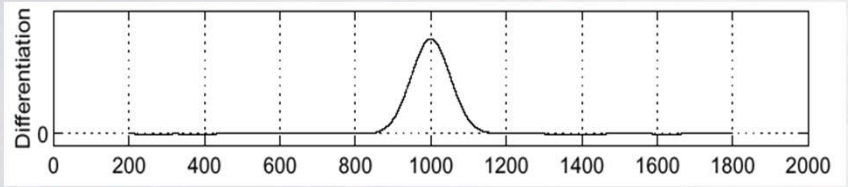


where is the edge?

Suppression of non-maxima







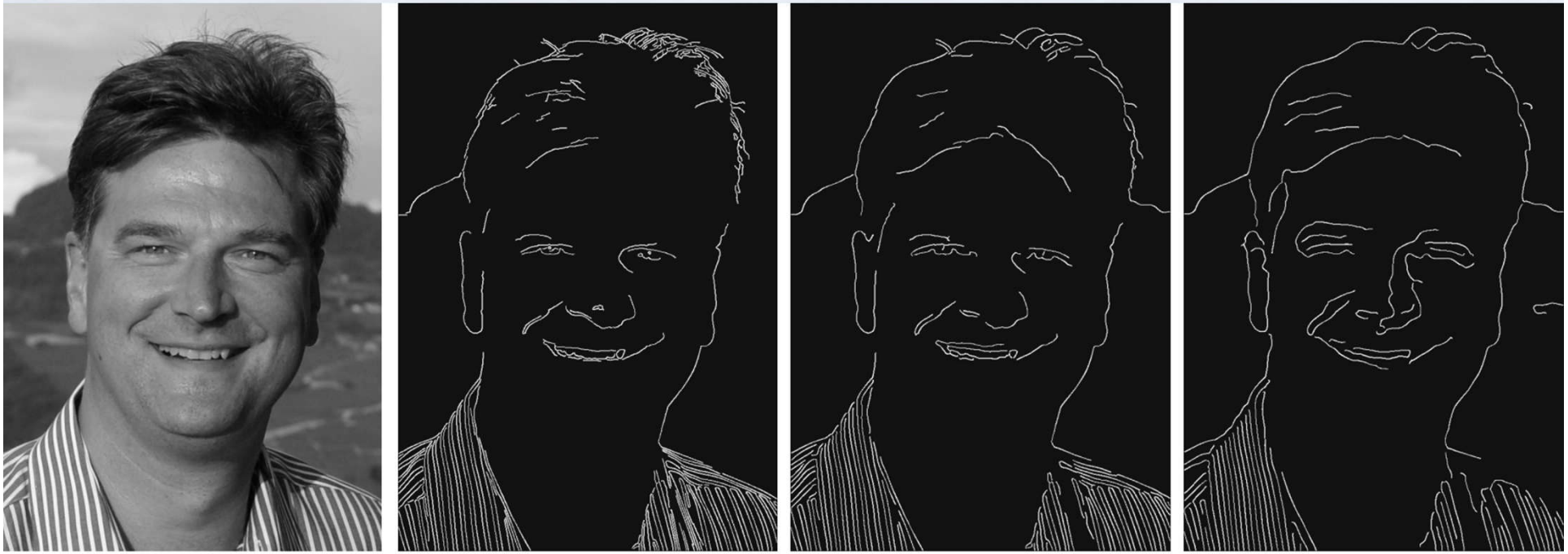
- Check if pixel is local maximum along gradient direction
  - requires *interpolating* pixels p and r

Suppression of non-maxima

Linking and hysteresis thresholding: define two thresholds: low and high

Use the high threshold to start edge curves and the low threshold to continue them

Canny edge detection example



$\sigma = \sqrt{2}$ $\sigma = 2\sqrt{2}$ $\sigma = 4\sqrt{2}$

From Bernd Girod, Stanford

The second-order derivative: the **Laplacian**

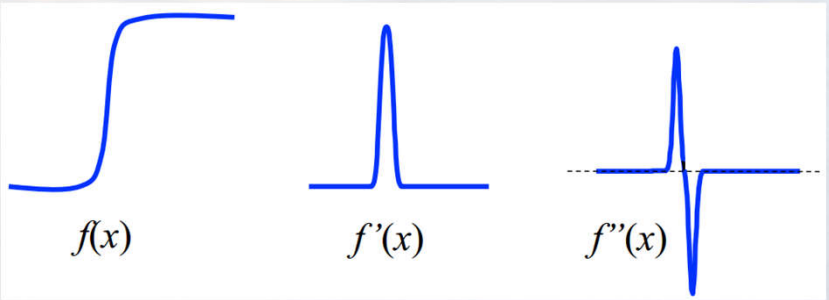
$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

In the discrete case:

$$\begin{aligned}\Delta\big(\Delta(f_x(x,y)\big) &= \Delta\big(f(x+1,y) - f(x,y)\big) \\ &= f(x+1,y) - f(x,y) - \big(f(x,y) - f(x-1,y)\big) \\ &= f(x+1,y) - 2f(x,y) + f(x-1,y)\end{aligned}$$

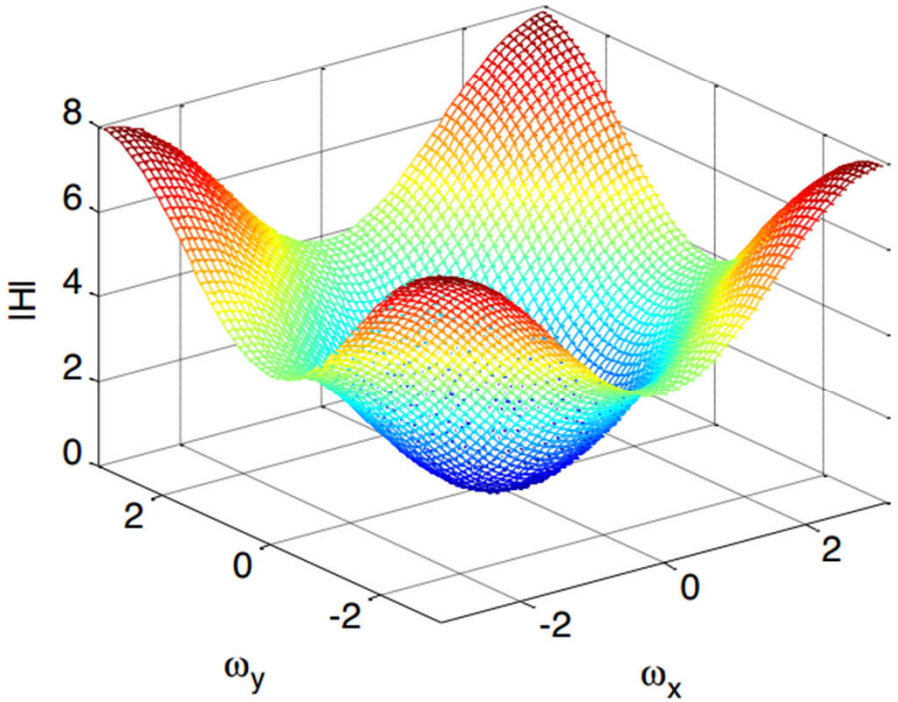
$$\begin{aligned}\Delta^2 f(x,y) &= \Delta^2 f_x(x,y) + \Delta^2 f_y(x,y) \\ &= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)\end{aligned}$$

And the kernel becomes: $\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

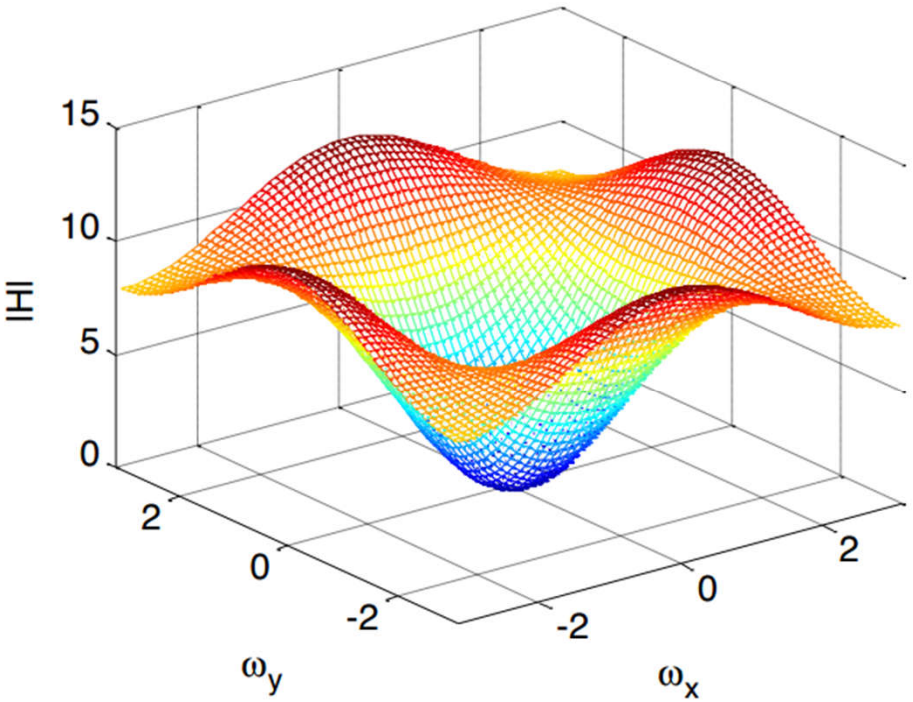


- The Laplacian

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & [-4] & 1 \\ 0 & 1 & 0 \end{pmatrix}$$



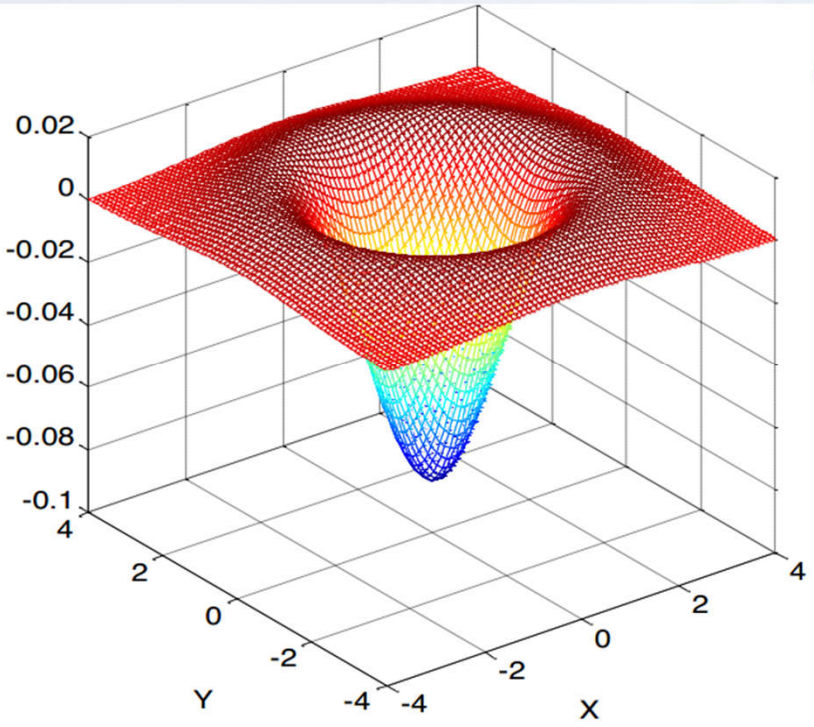
$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & [-8] & 1 \\ 1 & 1 & 1 \end{pmatrix}$$



The **laplacian** is sensitive to very fine details and noise (*you can add to any image to sharpen it!*). That is why we first smooth the image with a Gaussian filter. As before, both operations can be done at once with a **Laplacian of Gaussian (LoG)**.

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

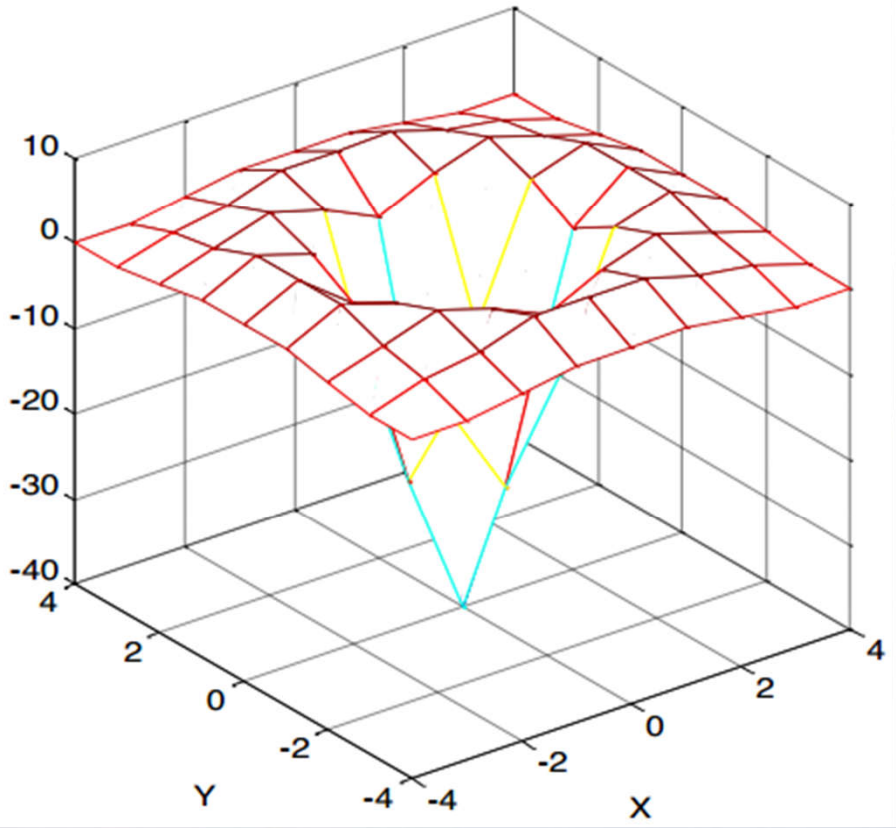
$\sigma = \sqrt{2}$



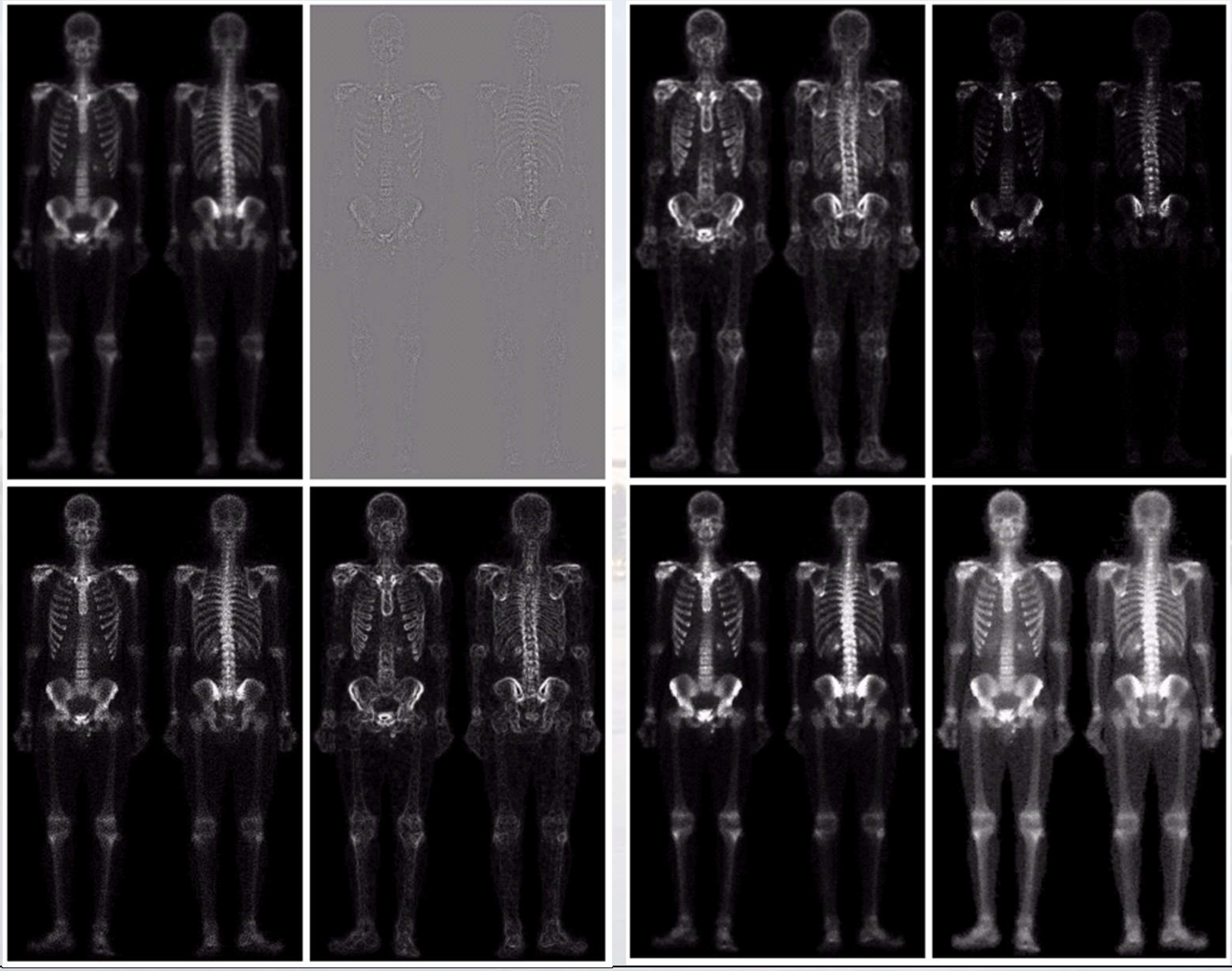
And its discrete form (note: LoG can also be approximated by a Difference of Gaussians - DoG)

$$\sigma = \sqrt{2}$$

0	0	1	2	2	2	1	0	0
0	2	3	5	5	5	3	2	0
1	3	5	3	0	3	5	3	1
2	5	3	-12	-23	-12	3	5	2
2	5	0	-23	-40	-23	0	5	2
2	5	3	-12	-23	-12	3	5	2
1	3	5	3	0	3	5	3	1
0	2	3	5	5	5	3	2	0
0	0	1	2	2	2	1	0	0



Combining multiple enhancement methods



a b
c d

FIGURE 3.46
(a) Image of whole body bone scan.
(b) Laplacian of (a). (c) Sharpened image obtained by adding (a) and (b). (d) Sobel of (a).

e f
g h

FIGURE 3.46
(Continued)
(e) Sobel image smoothed with a 5×5 averaging filter. (f) Mask image formed by the product of (c) and (e).
(g) Sharpened image obtained by the sum of (a) and (f). (h) Final result obtained by applying a power-law transformation to (g). Compare (g) and (h) with (a). (Original image courtesy of G.E. Medical Systems.)