Master in
Computer Vision
*Barcelona*

**Module: Introduction to Human and Computer Vision**

**Lecture 6:** Space-frequency representation (II)

**Lecturer:** Javier Ruiz Hidalgo

# Outline

- **Low-pass and high-pass LSI filters**
  - Filters design in the spatial domain
  - Filters design in the frequency domain
  - Noise reduction example

- Linear transformations
  - Introduction
  - Discrete Cosine Transform (DCT)
  - Karhunen-Loeve Transform (KLT)

# Filters designed in the spatial domain

- Filters are defined in the spatial domain when their **size** (area of support) is relatively **small**. In other cases, filters are usually defined in the frequency domain.
- Filters are commonly **square, odd in size and symmetric** (3x3, 5x5, …)
- Two different types of filters are going to be studied:
  - **Low-pass filters**, that perform a spatial average, and their application to noise removal.
  - **High-pass filters**, that perform an estimation of the derivative, and their application to contour detection.

# Low-pass filters designed in the spatial domain (1)

- Study low-pass filters in the context of **noise reduction**.
- In image processing, there are three main types of noise:

- **Gaussian noise**: It can appear in the sensors of a CCD camera.

$$p(n) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(n-\mu)^2}{\sigma^2}}$$

- **Uniform noise**: It is produced by the quantization process.

$$p(n) = \begin{cases} \frac{1}{b-a} & \text{if } a \le n \le b \\ 0 & \text{otherwise} \end{cases}$$

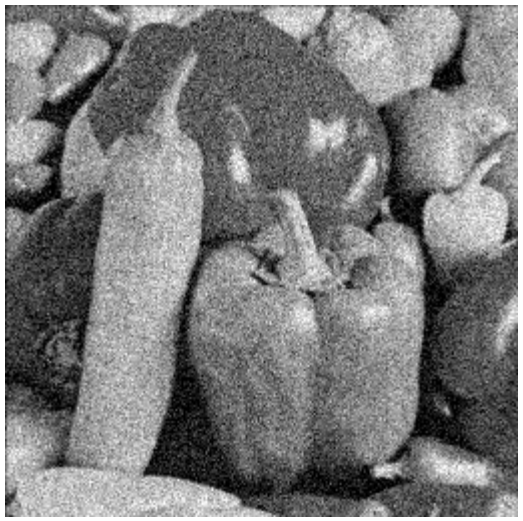
- **Salt and pepper noise**: Typical of error transmissions or damaged CCD sensors.

$$\tilde{i}(m,n) = \begin{cases} s & \text{with probability } p \\ i(m,n) & \text{with probability } 1-p \end{cases}$$

*s* is a random variable that takes values in {black (0), white (255)}

# Low-pass filters designed in the spatial domain (2)

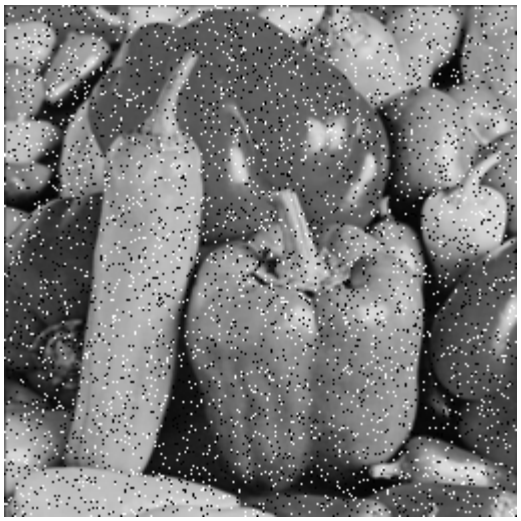
Noise examples



**Gaussian noise**

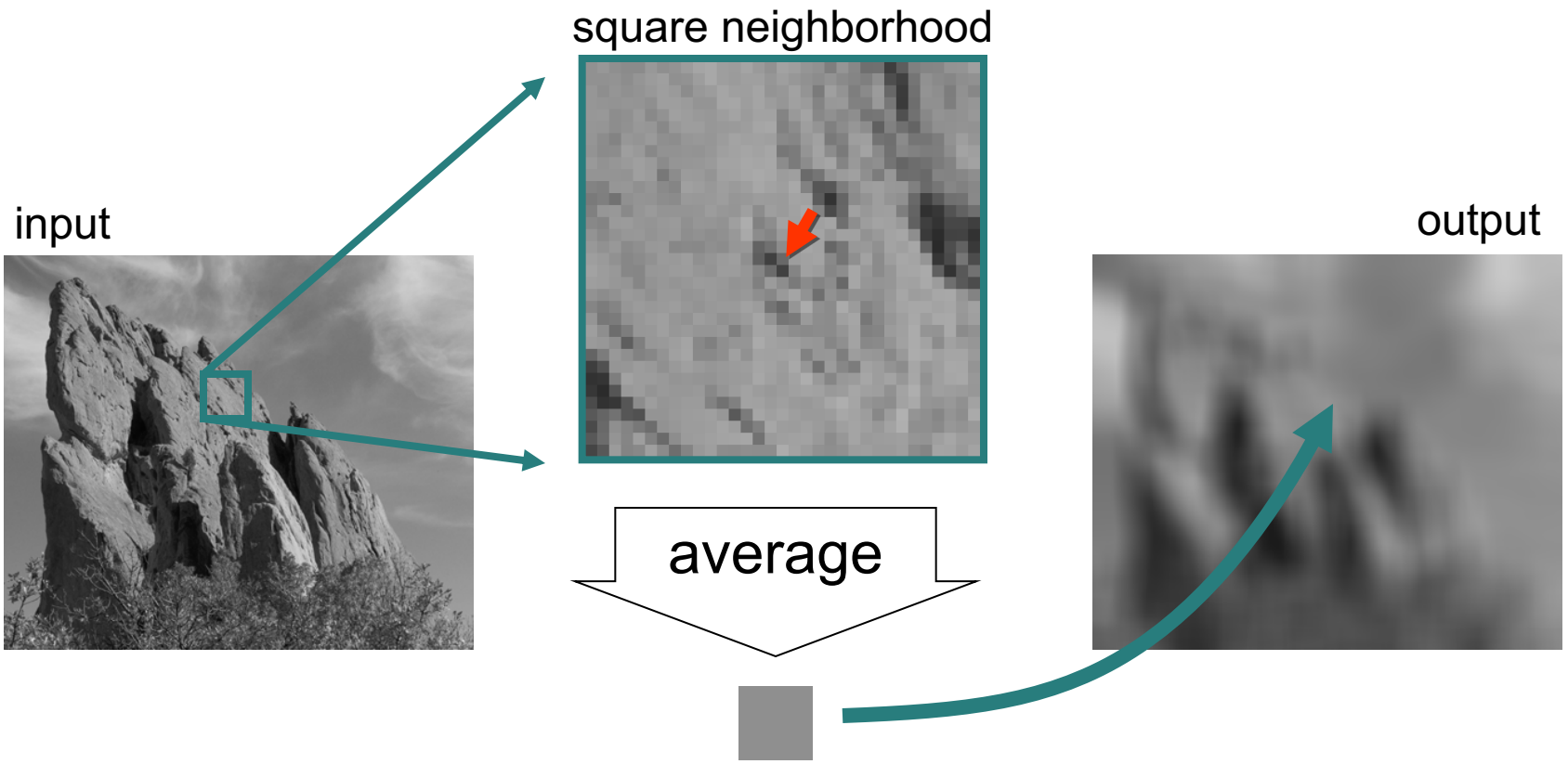
**($m=0$, $\sigma^2=1$)**

**Uniform noise**



**Salt and pepper noise (p=0.08)**

# Average filter (1)



# Average filter (2)

## Convolution

- The impulse response of an average filter of size N (NxN) is a matrix with equal values (adding up to 1).

$$h[m,n] = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & \underline{1/9} & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = h_1[m]h_2[n]$$

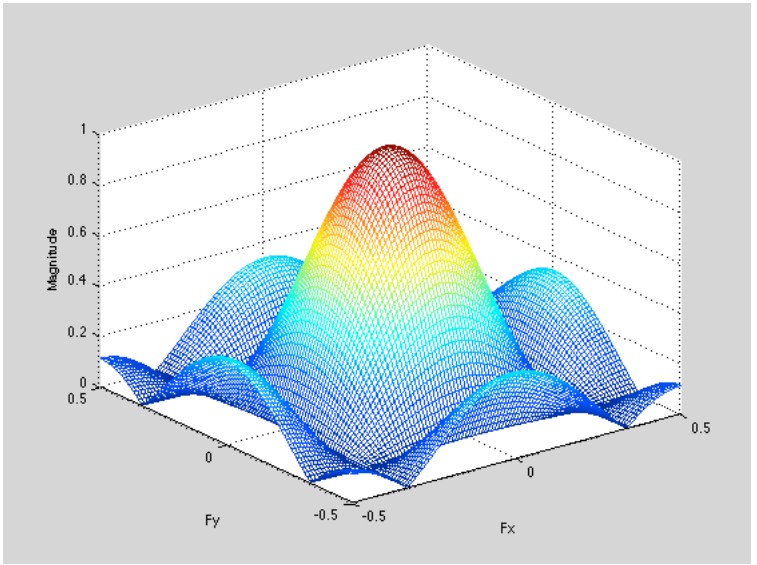
$$y[m,n] = x[m,n] * h[m,n] = \sum_{m'} \sum_{n'} x[m',n']h[m-m',n-n']$$

# Average filter (3)

## Frequency representation

- The average filter is separable and, therefore:

$$h[m,n] = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & \underline{1/9} & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = h_1[m]h_2[n]$$




$$h_1[m] = \frac{1}{3}\Pi_3[m] \qquad h_2[n] = \frac{1}{3}\Pi_3[n]$$

$$H(F_1,F_2) = H_1(F_1)\,H_2(F_2) \quad \Rightarrow \quad H(F_1,F_2) = \frac{1}{9}\frac{\sin[3\pi F_1]}{\sin[\pi F_1]}\frac{\sin[3\pi F_2]}{\sin[\pi F_2]}$$

# Average filter (4)

## Pixel representation

- For a pixel $\mathbf{p}$, the convolution equation can be interpreted as a weighted average

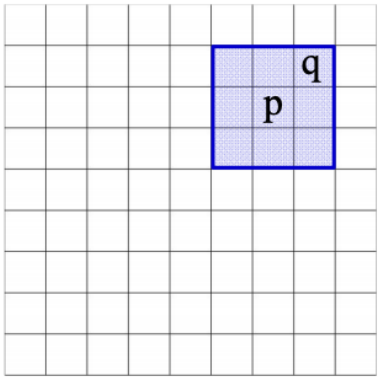
$$I_A(p) = \sum_{q \in F} A_N(p-q) I(q)$$

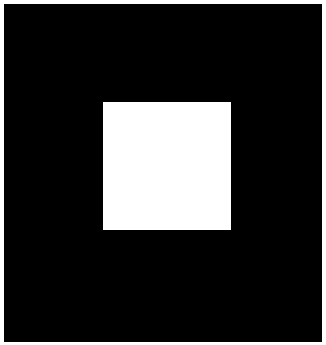
Output result at pixel $\mathbf{p}$

sum over all support pixels $\mathbf{q}$ of filter $\mathbf{F}$

normalized average function

intensity at pixel $\mathbf{q}$

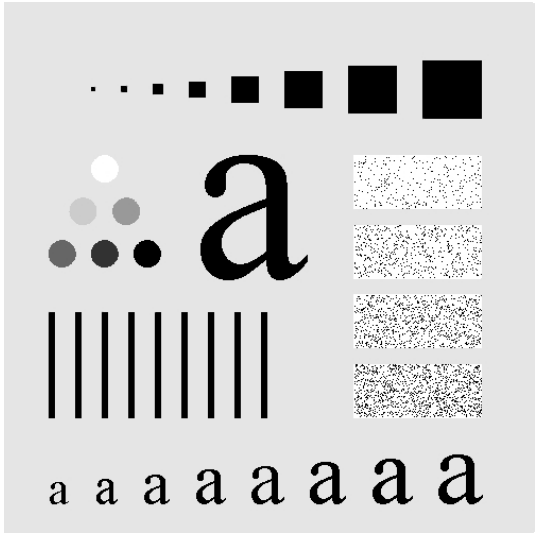


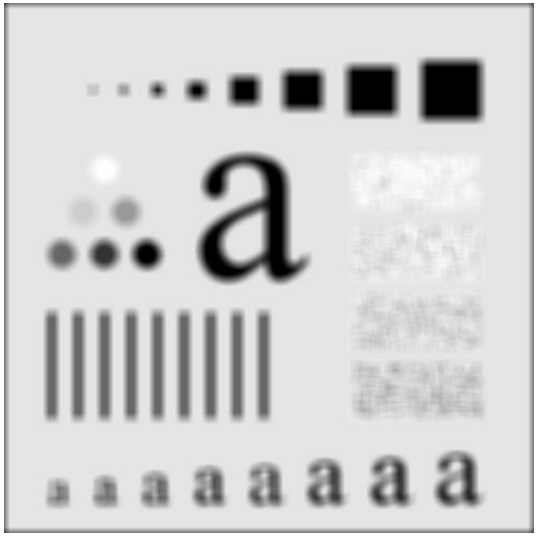
# Average filter (5)

## Filtering examples

Original image



Average 5x5



Average 11x11

# Average filter (6)

## Defects

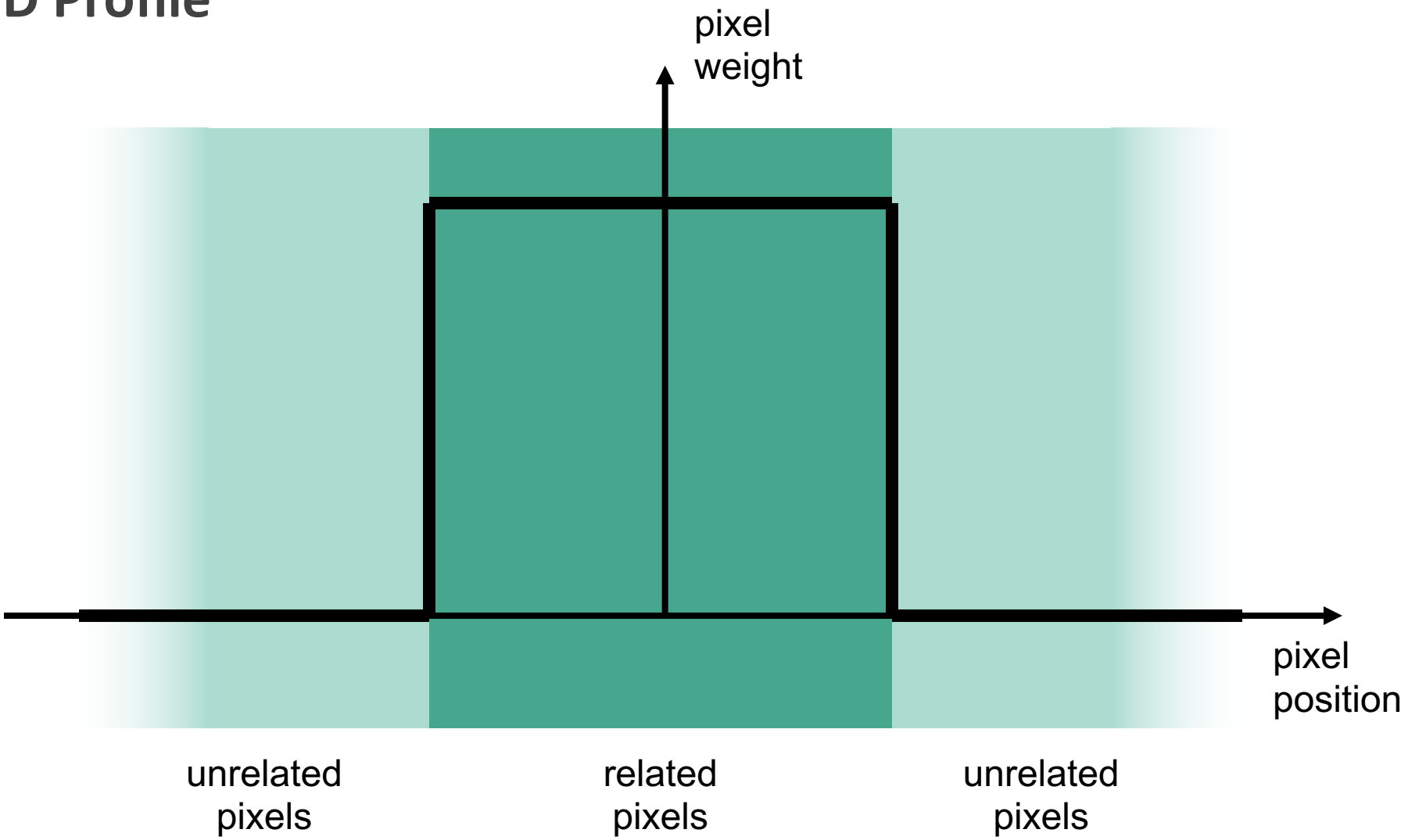
- Axis-aligned streaks
- Artefacts due to square impulse response

output

input

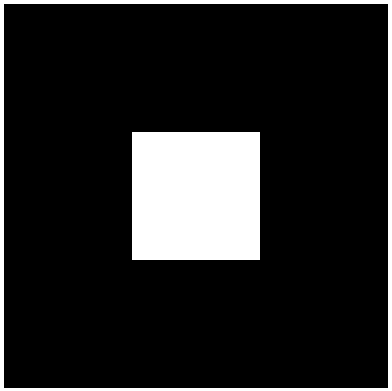
# Average filter (7)

## 1D Profile

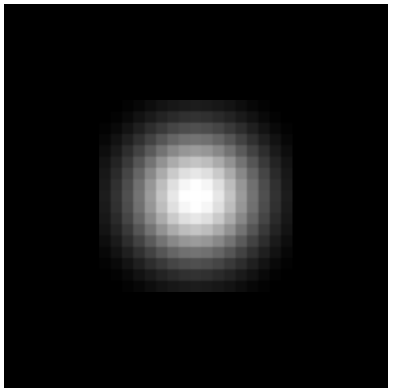


# Strategy to Solve these Problems

- Use an isotropic (*i.e.* circular) window.
- Use an impulse response with a smooth falloff.



box window

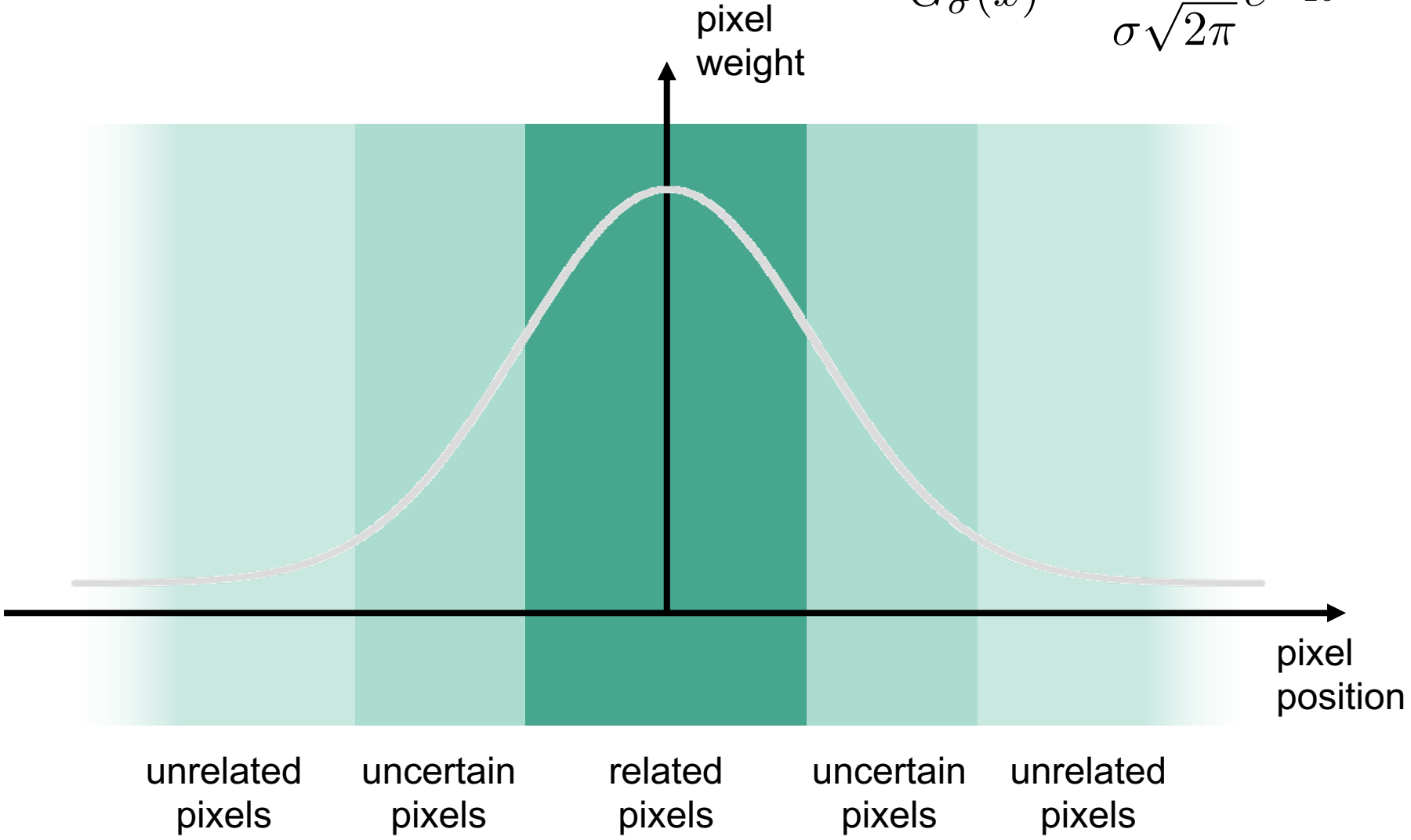


Gaussian window

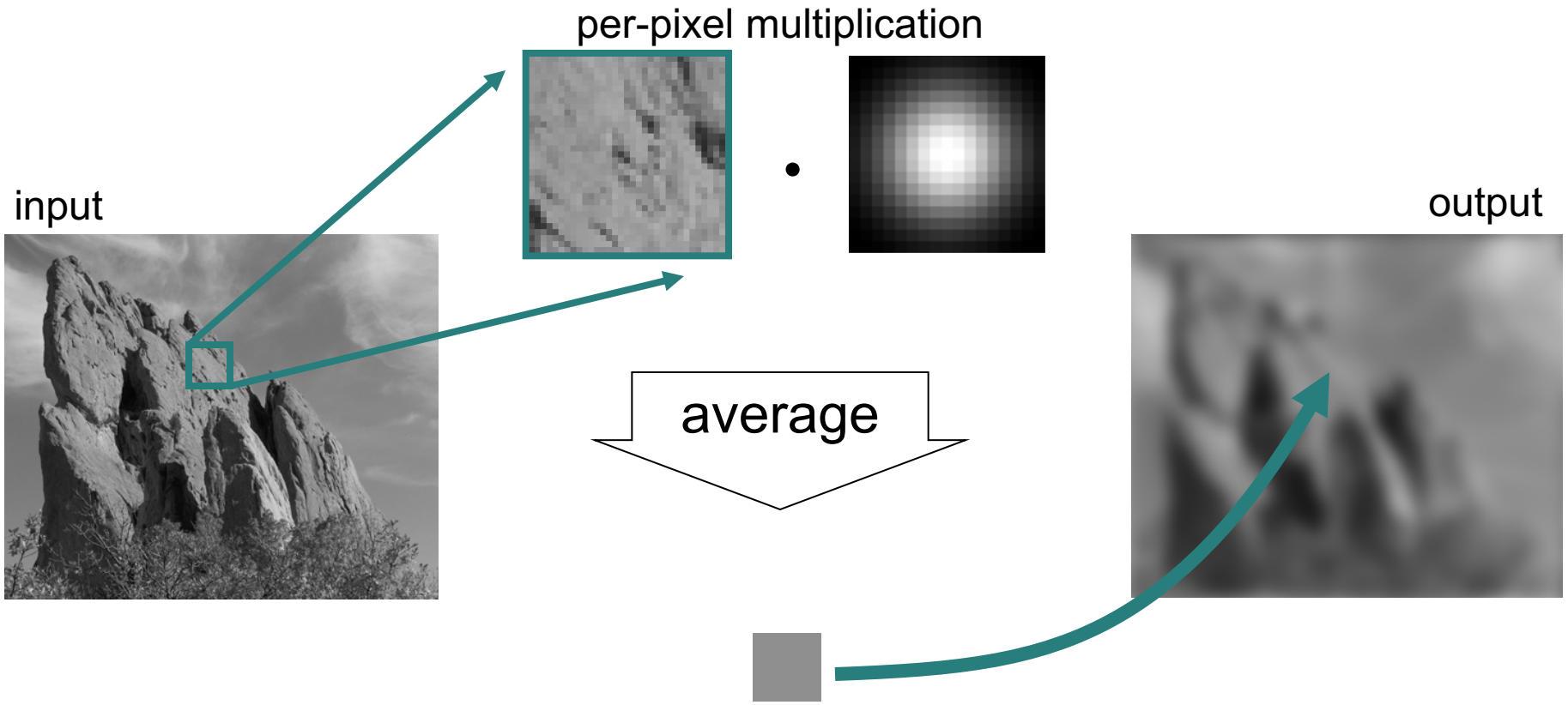
# Gaussian filter (5)

## 1D Profile

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



# Gaussian filter (1)



# Gaussian filter (2)

## Convolution

- The impulse response of a gaussian filter follows a gaussian distribution of standard deviation σ

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

- Example of a filter 5x5 with σ = 1

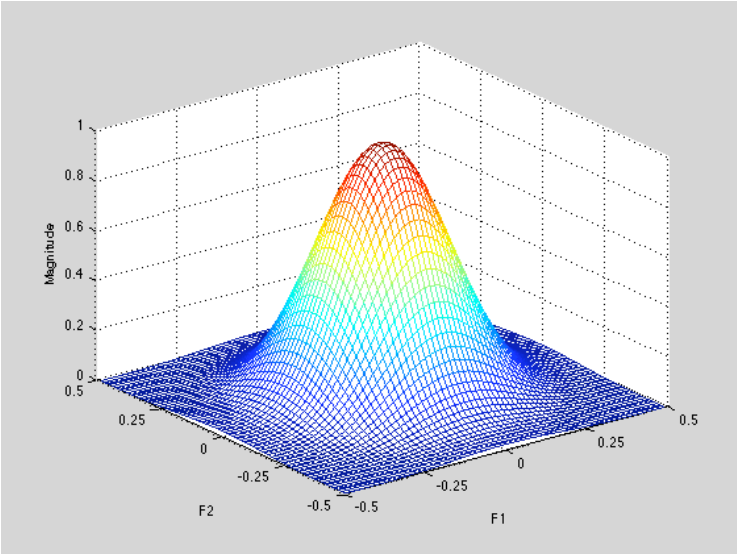
$$h = \begin{bmatrix} 0.00 & 0.01 & 0.02 & 0.01 & 0.00 \\ 0.01 & 0.06 & 0.10 & 0.06 & 0.01 \\ 0.02 & 0.10 & \underline{0.16} & 0.10 & 0.02 \\ 0.01 & 0.06 & 0.10 & 0.06 & 0.01 \\ 0.00 & 0.01 & 0.02 & 0.01 & 0.00 \end{bmatrix}$$

$$y[m,n] = x[m,n] * h[m,n] = \sum_{m'} \sum_{n'} x[m',n'] h[m-m', n-n']$$

# Gaussian filter (3)

## Frequency representation

- The gaussian filter also performs averaging between pixels and therefore its frequency representation is a low pass filter



# Gaussian filter (4)

## Pixel representation

- For a pixel $\mathbf{p}$, the convolution equation can be interpreted as a weighted average

$$I_G(p) = \sum_{q \in F} G_\sigma(||p - q||) I(q)$$

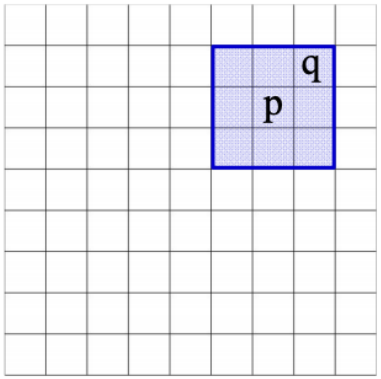
Output result at pixel $\mathbf{p}$

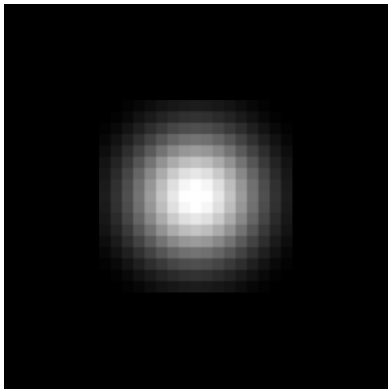
sum over all support pixels $\mathbf{q}$ of filter $\mathbf{F}$

size of the window

normalized gaussian function

intensity at pixel $\mathbf{q}$

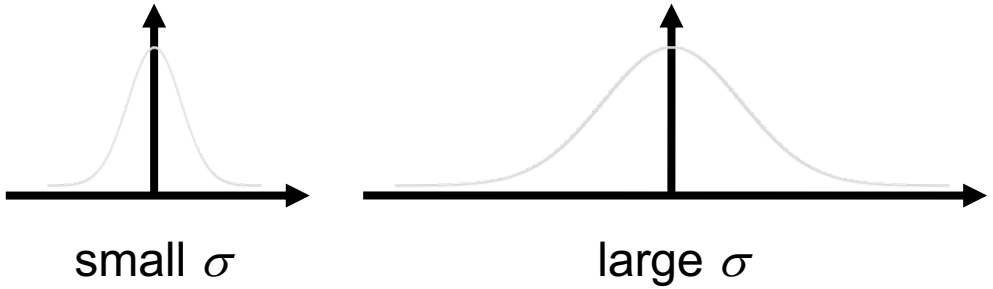


# Gaussian filter (6)

## Filtering examples

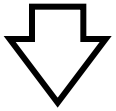
**Original image**




# Gaussian filter (7)

- Blur images
- But smoothes too much: **edges are blurred**.
  - Only spatial distance matters
  - No edge term

input



output

$$I_G(p) = \sum_{q \in F} G_\sigma(||p-q||) I(q)$$

spatial

# Outline

- **Low-pass and high-pass LSI filters**
  - Filters design in the spatial domain
  - Filter design in the frequency domain
  - **Noise reduction example**

- Linear transformations
  - Introduction
  - Discrete Cosine Transform (DCT)
  - Karhunen-Loeve Transform (KLT)

# Noise reduction

**Original image**

**Uniform noise**

**Averaging filter**

$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & \underline{1/9} & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

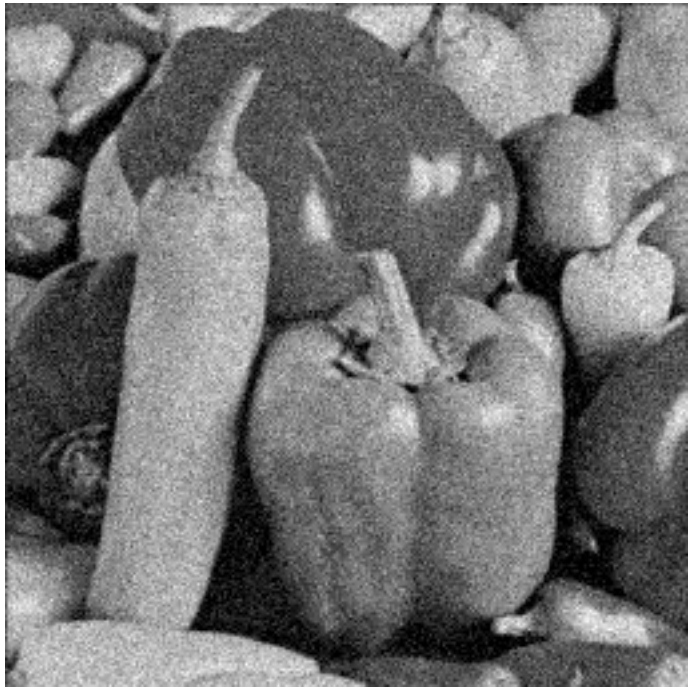
**Gaussian filter**

$$\begin{bmatrix} 0.011 & 0.084 & 0.011 \\ 0.084 & \underline{0.619} & 0.084 \\ 0.011 & 0.084 & 0.011 \end{bmatrix}$$

# Filter design: Noise reduction example (1)

**Ideal image**

**Image corrupted by additive Gaussian noise**



PSNR=27,88 dB
SSIM=0,448

# Filter design: Noise reduction example (2)

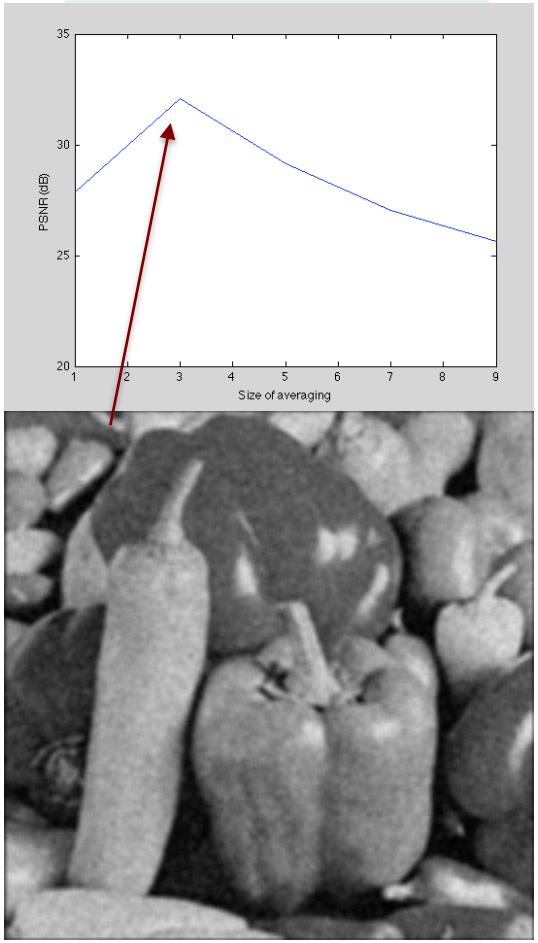
Assume we use an averaging filter: how to define the size of the **impulse response** ?

- Small impulse responses: do not remove much noise
- Large windows: blur edges

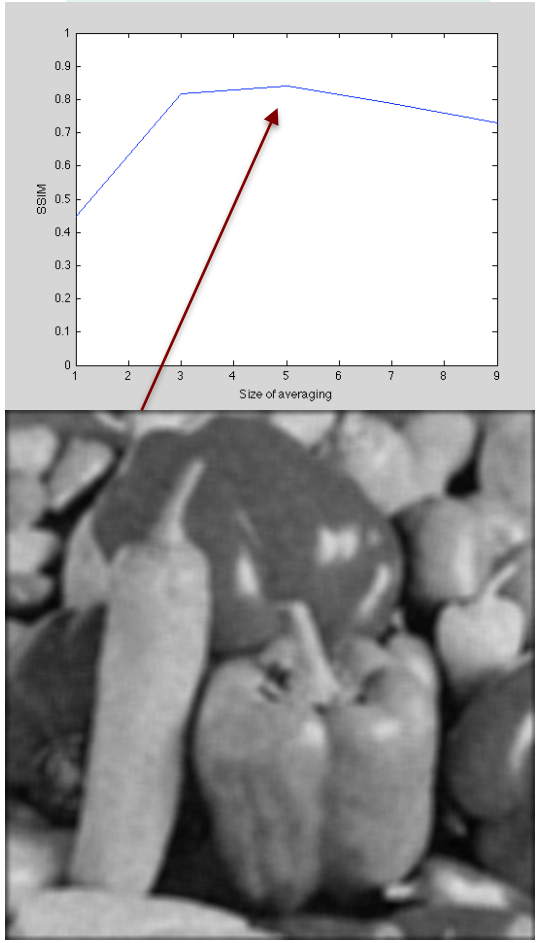
# Filter design: Noise reduction example (3)

Study of the compromise: **noise reduction/blurring**

PSNR

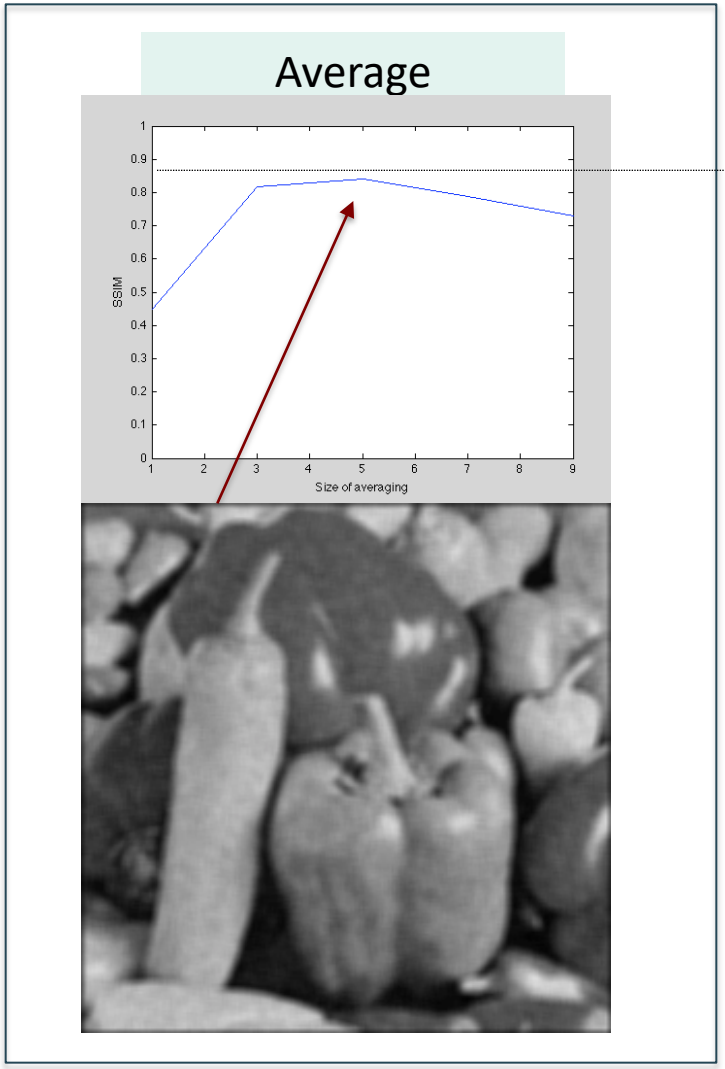


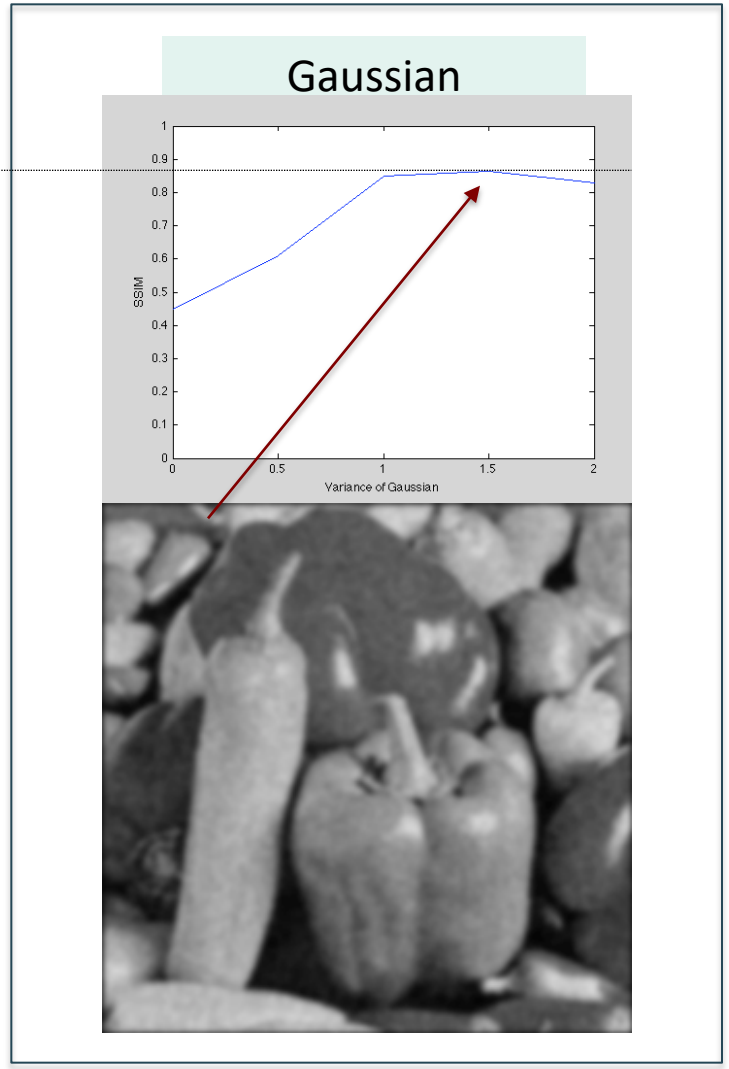

SSIM



# Filter design: Noise reduction example (4)

Is the **Gaussian filter** better than the averaging?



# Filter design: Noise reduction example (5)

Study in the case of **impulsive noise** (salt & pepper)

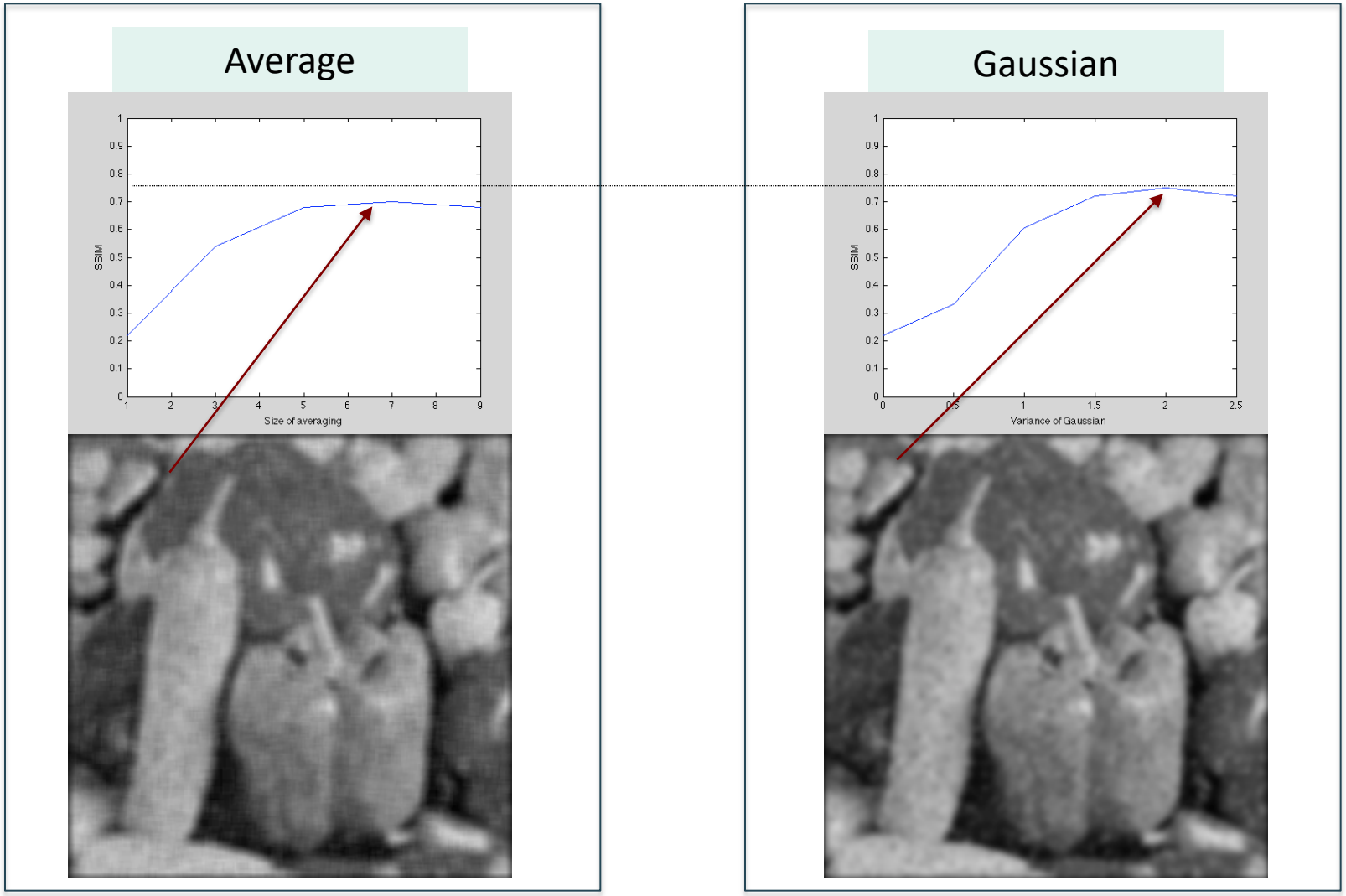
**Ideal image**

**Image corrupted by impulsive noise**

PSNR=19,22 dB
SSIM=0,22

# Filter design: Noise reduction example (6)

Study in the case of **impulsive noise** (salt & pepper)



# Filter design: Noise reduction example (7)

Gaussian filter is better than Averaging filter…. **But can we do better**?

(Original noisy images: PSNR=19,22 dB; SSIM=0,22)

Gaussian

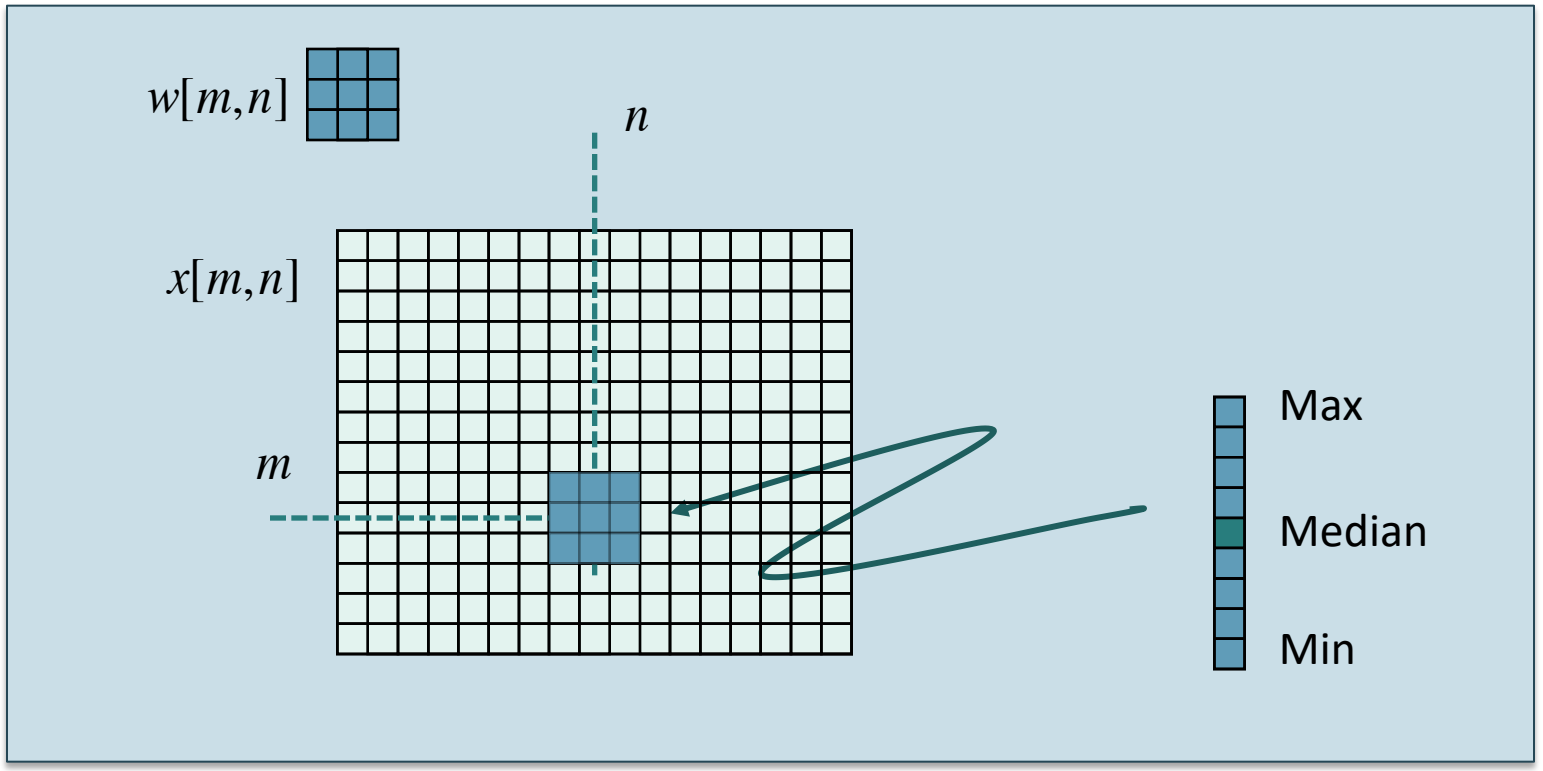
PSNR=27,50 dB
SSIM=0,75

?

PSNR=34,91 dB
SSIM=0,97

# Filter design: Noise reduction example (8)

**Median filter**: a non linear filter that is very efficient for impulsive noise removal!



# Outline

- **Low-pass and high-pass LSI filters**
  - Filters design in the spatial domain
  - Filters design in the frequency domain
  - Noise reduction example

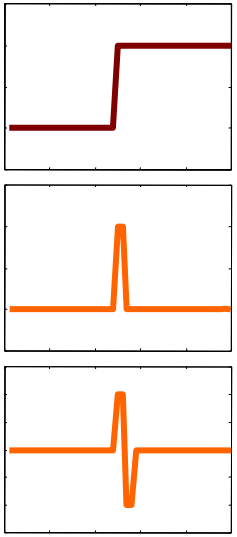
- Linear transformations
  - Introduction
  - Discrete Cosine Transform (DCT)
  - Karhunen-Loeve Transform (KLT)

# High-pass filters designed in the spatial domain (1)

- High-pass filters in the context of **contour detection**.

Contours (**transitions**) in a signal can be detected by:

- **First derivative**: There is a **local maximum** (or minimum) in the derivative in the position of the transition.
- **Second derivative**: There is a **zero crossing** in the second derivative in the position of the transition.



- In 2D, the **gradient** or the **Laplacian** of the image are studied:

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

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

# High-pass filters designed in the spatial domain (2)

- In image processing, gradients can be approximated by means of **finite differences equations**, one for each direction (*x*, *y*).

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\Delta x \to 0}\left[\frac{f(x+\Delta x,y)-f(x,y)}{\Delta x}\right] \approx \frac{f(x+\Delta x,y)-f(x,y)}{\Delta x}$$

**Forward difference**

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\Delta x \to 0}\left[\frac{f(x,y)-f(x-\Delta x,y)}{\Delta x}\right] \approx \frac{f(x,y)-f(x-\Delta x,y)}{\Delta x}$$

**Backward difference**

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\Delta x \to 0}\left[\frac{f(x+\Delta x,y)-f(x-\Delta x,y)}{2\Delta x}\right] \approx \frac{f(x+\Delta x,y)-f(x-\Delta x,y)}{2\Delta x}$$

**Symmetric difference**

$$\left.\frac{\partial f(x,y)}{\partial x}\right|_{\substack{x=m\\y=n}} \approx f(m+1,n)-f(m,n)$$

$$\left.\frac{\partial f(x,y)}{\partial x}\right|_{\substack{x=m\\y=n}} \approx f(m,n)-f(m-1,n)$$

$$\left.\frac{\partial f(x,y)}{\partial x}\right|_{\substack{x=m\\y=n}} \approx \frac{f(m+1,n)-f(m-1,n)}{2}$$

# High-pass filters designed in the spatial domain (3)

- Each of these equations can be implemented as an **LSI filter**.

$$\left.\frac{\partial f(x,y)}{\partial x}\right|_{\substack{x=m\\y=n}} \approx f(m+1,n)-f(m,n) \qquad h_x^+[m]=[1,\underline{-1}] \quad \Rightarrow \quad \frac{\partial f}{\partial x}\approx f*h_x^+$$

$$\left.\frac{\partial f(x,y)}{\partial x}\right|_{\substack{x=m\\y=n}} \approx f(m,n)-f(m-1,n) \qquad h_x^-[m]=[\underline{1},-1] \quad \Rightarrow \quad \frac{\partial f}{\partial x}\approx f*h_x^-$$

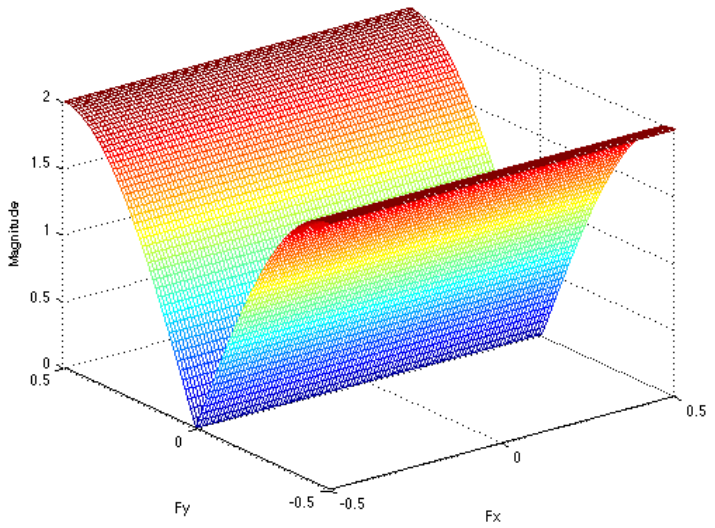
$$\left.\frac{\partial f(x,y)}{\partial x}\right|_{\substack{x=m\\y=n}} \approx \frac{f(m+1,n)-f(m-1,n)}{2} \qquad h_x^s[m]=[1,\underline{0},-1] \quad \Rightarrow \quad \frac{\partial f}{\partial x}\approx f*h_x^s$$

- In the symmetric filter, **the factor ½ has been removed** since usually the exact value of the gradient is not sought.
- Analogous filters can be defined in the **vertical direction**.

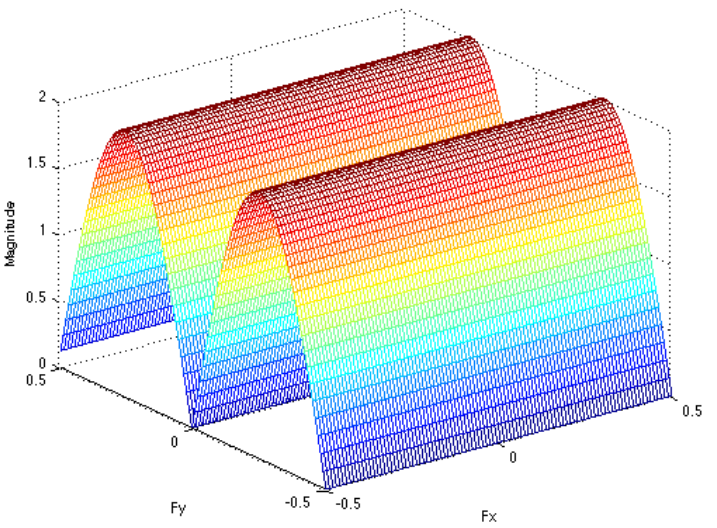
# High-pass filters designed in the spatial domain (4)

- Frequency response of the basic derivative approximations

$$h_y^-[m,n] = \begin{bmatrix} 1 \\ \underline{0} \\ -1 \end{bmatrix}$$




$$h_y^s[m,n] = \begin{bmatrix} 1 \\ \underline{0} \\ -1 \end{bmatrix}$$



# High-pass filters designed in the spatial domain (5)

- There exist **several implementations** of the filter that approximates the gradient in a given direction. Let us denote them, in general, as $h_x$ ($h_y$).
- Typically, these filters do not estimate the gradient relying only on the data in one row (column) but in a **set of neighbor rows** (columns)
- This allows averaging among several rows (columns) and being more **robust against the presence of noise**

$$h_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & \underline{0} & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

**Horizontal derivative**
**Detects vertical contours**

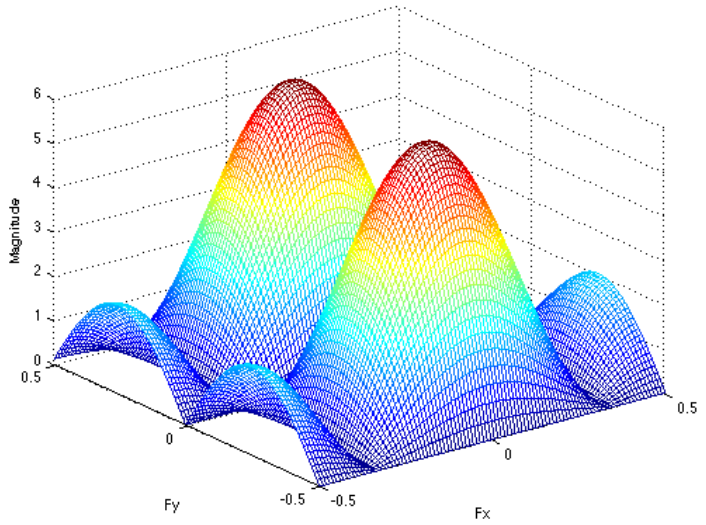
$$h_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & \underline{0} & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

**Vertical derivative**
**Detects horizontal contours**

# High-pass filters designed in the spatial domain (6)

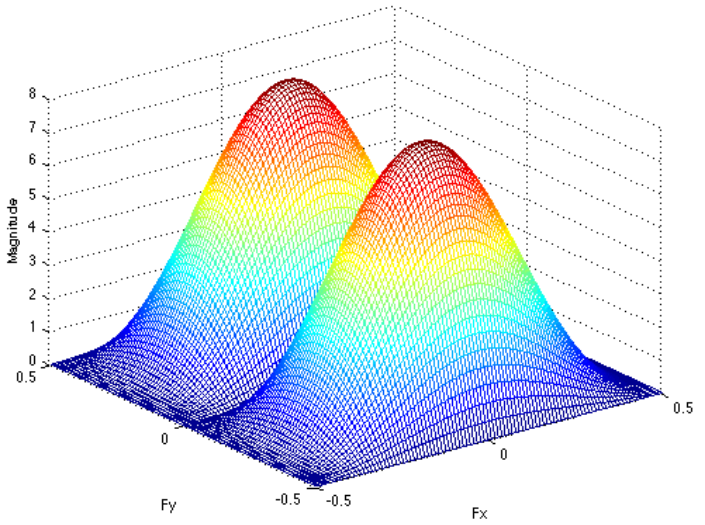
Prewitt

$$h_y[m,n] = \begin{bmatrix} 1 & 1 & 1 \\ 0 & \underline{0} & 0 \\ -1 & -1 & -1 \end{bmatrix}$$




Sobel

$$h_y[m,n] = \begin{bmatrix} 1 & 2 & 1 \\ 0 & \underline{0} & 0 \\ -1 & -2 & -1 \end{bmatrix}$$




**Gradient approximation**

# High-pass filters designed in the spatial domain (7)

- Given the previous filters, a set of images can be defined that help analyzing the gradient of the input image:

$$\nabla f[m,n] = \begin{bmatrix} g_x[m,n] \\ g_y[m,n] \end{bmatrix}$$

**Estimation of the gradient**

$$g_x[m,n] = f[m,n] * h_x[m,n]$$

**Image that stores the estimation of the horizontal gradient**

$$g_y[m,n] = f[m,n] * h_y[m,n]$$

**Image that stores the estimation of the vertical gradient**

$$\nabla f[m,n] = \sqrt{g_x^2[m,n] + g_y^2[m,n]}$$

**Image that stores the estimation of the gradient magnitude**

$$\theta_{\nabla f}[m,n] = \arctan\left[\frac{g_x[m,n]}{g_y[m,n]}\right]$$

**Image that stores the estimation of the gradient phase**

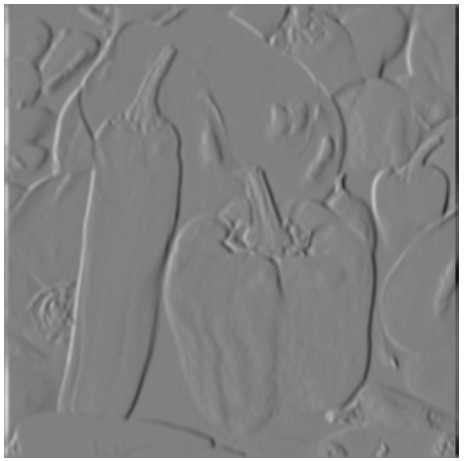
# High-pass filters designed in the spatial domain (8)

- The **Prewitt filter** detects both horizontal and vertical contours and it is robust with respect to the noise.

**Original image**

**Horizontal derivative**
**Detects vertical contours**

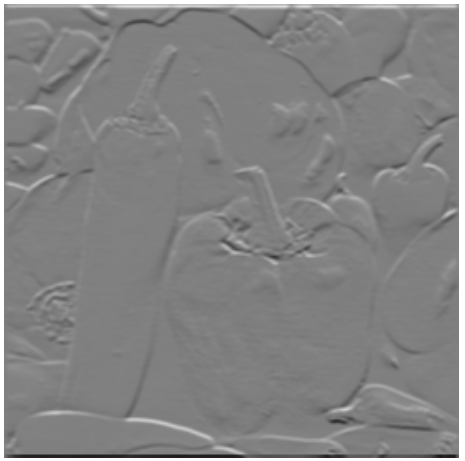
$$h_x = \begin{bmatrix} 1 & 0 & -1 \\ 1 & \underline{0} & -1 \\ 1 & 0 & -1 \end{bmatrix}$$



**Gradient magnitude**

**Vertical derivative**
**Detects horizontal contours**

$$h_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & \underline{0} & 0 \\ -1 & -1 & -1 \end{bmatrix}$$



# High-pass filters designed in the spatial domain (9)

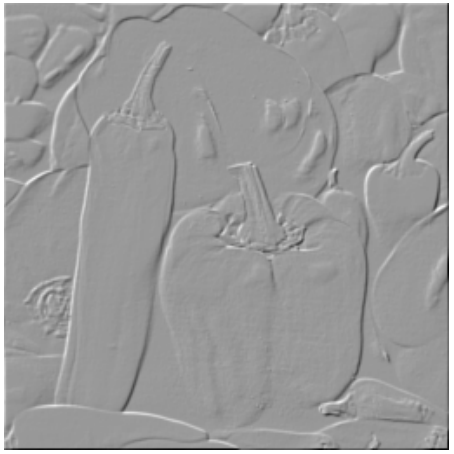
- The **Roberts filter** detects contours in both diagonal directions and it is not robust with respect to the noise.

**Original image**

**Gradient magnitude**

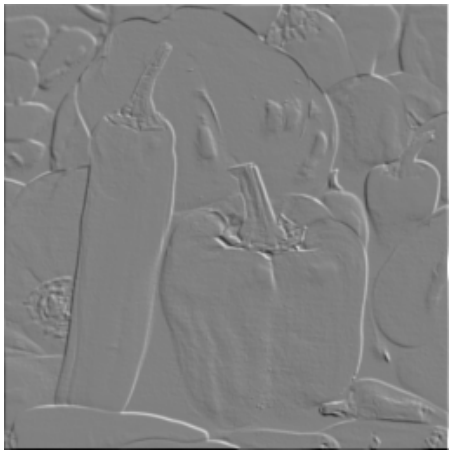
**Diagonal direction**
**Detects contours at 45º**

$$h_1 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$



**Diagonal direction**
**Detects contours at 135º**

$$h_2 = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$



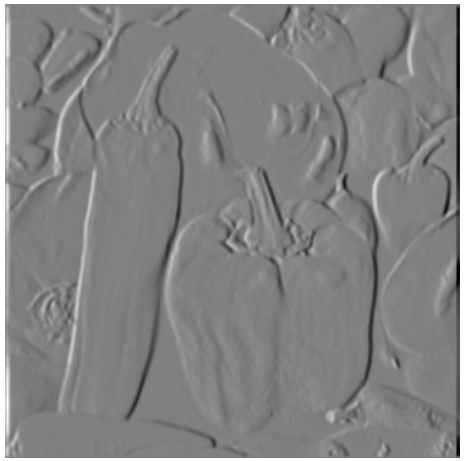
# High-pass filters designed in the spatial domain (10)

- The **Sobel filter** detects both horizontal and vertical contours and increases the robustness with respect to the noise.

**Original image**

**Horizontal derivative**
**Detects vertical contours**

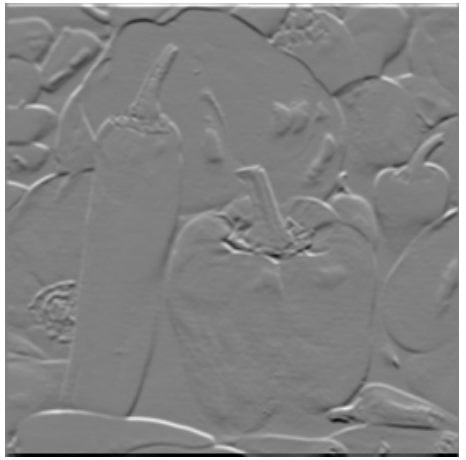
$$h_x[m,n] = \begin{bmatrix} -1 & 0 & 1 \\ -2 & \underline{0} & 2 \\ -1 & 0 & 1 \end{bmatrix}$$



**Gradient magnitude**

**Vertical derivative**
**Detects horizontal contours**

$$h_y[m,n] = \begin{bmatrix} 1 & 2 & 1 \\ 0 & \underline{0} & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



# High-pass filters designed in the spatial domain (11)

- In image processing, the Laplacian operator can be approximated by means of **equations in finite differences**.

$$\left.\frac{\partial^2 f(x,y)}{\partial x^2}\right|_{\substack{x=m\\y=n}} \approx f(m+1,n)-2f(m,n)+f(m-1,n)$$

$$\left.\frac{\partial^2 f(x,y)}{\partial y^2}\right|_{\substack{x=m\\y=n}} \approx f(m,n+1)-2f(m,n)+f(m,n-1)$$

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

$$\Delta f(x,y)\big|_{\substack{x=m\\y=n}} = \left.\frac{\partial^2 f}{\partial x^2}+\frac{\partial^2 f}{\partial y^2}\right|_{\substack{x=m\\y=n}} \approx f(m+1,n)+f(m-1,n)+f(m,n+1)+f(m,n-1)-4f(m,n)$$

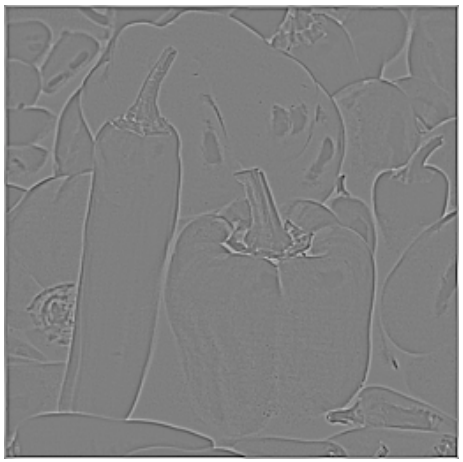
- This equation can be implemented by means of the convolution with the so-called **Laplacian filter**:

$$h=\begin{bmatrix}0 & 1 & 0\\ 1 & -4 & 1\\ 0 & 1 & 0\end{bmatrix} \quad \text{or} \quad h=\begin{bmatrix}0 & -1 & 0\\ -1 & 4 & -1\\ 0 & -1 & 0\end{bmatrix}$$

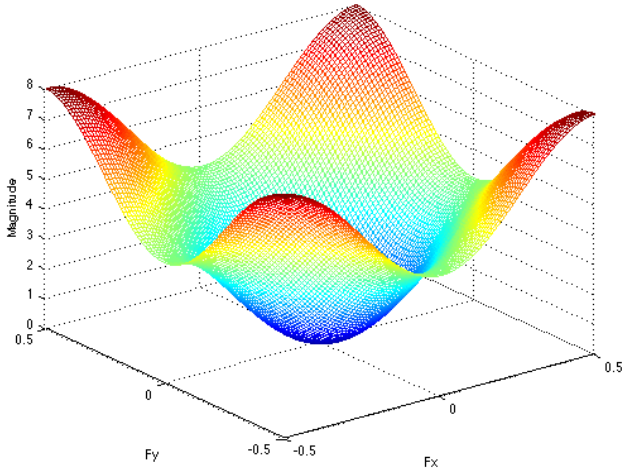
# High-pass filters designed in the spatial domain (12)

- The **Laplacian filter** detects contours estimating the zeros of the Laplacian function. It is not very robust against noise:
  - Typically, a filter to remove noise is applied previous to the Laplacian filter.
- It also (as the directional gradient) produces positive and negative values and, to visualize the result, a **contrast transform** is applied.

**Original image**



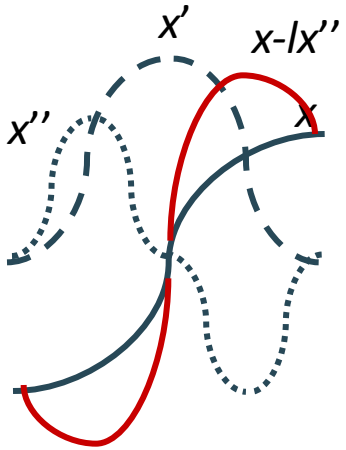
**Laplacian image**




**Frequency response**

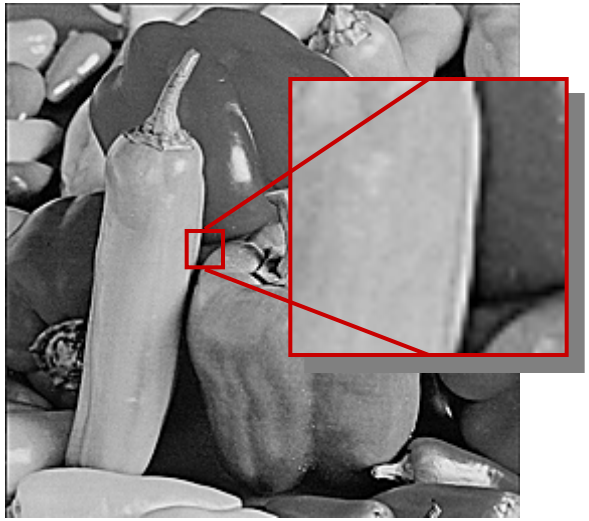
# High-pass filters designed in the spatial domain: Application to enhancement

- For visualization purposes, it may be interesting to highlight the presence of contours in the image. This can be done by **unsharp masking**; that is, adding a weighted version of the Laplacian image to the original one.




Original image

Laplacian image



Enhanced image

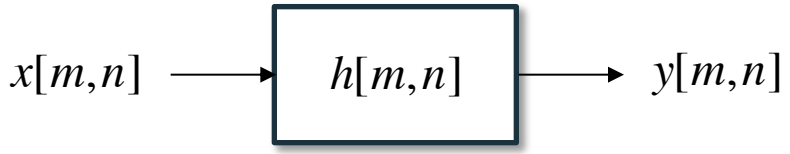
# Outline

- **Low-pass and high-pass LSI filters**
  - Filters design in the spatial domain
  - **Filter design in the frequency domain**
  - Noise reduction example

- Linear transformations
  - Introduction
  - Discrete Cosine Transform (DCT)
  - Karhunen-Loeve Transform (KLT)

# Filters in the frequency domain

- Ideal filters (with **sharp transitions**) can be defined in the frequency domain.




$$y[m,n] = x[m,n] * h[m,n]$$
$$\tilde{Y}[k,l] = X[k,l] \cdot H[k,l]$$

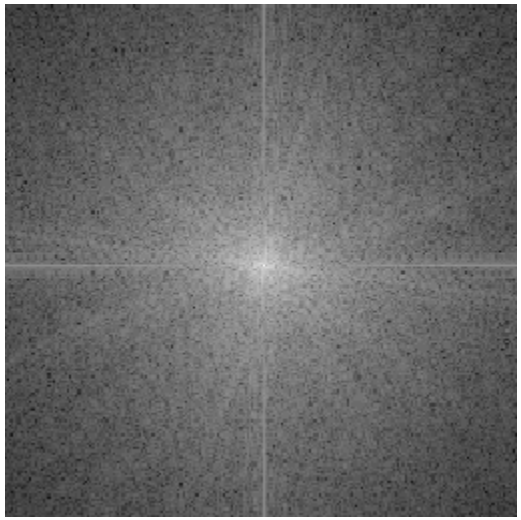
- The filtering process implemented in the frequency domain implies:
  - Computing the DFT of $x[m, n]$ and center it: $X[k, l]$.
    - Filters are commonly defined centered.
  - Multiply the image transform by the filter: $\tilde{Y}[k,l] = X[k,l] \cdot H[k,l]$
  - Compute the inverse DFT of the resulting signal: $\tilde{Y}[k,l]$
  - Compute the real part of the result and de-center it again: $\tilde{y}[m,n]$
    - Although usually the filter has real values and, therefore, the filtered signal should be real, due to rounding effects final values may not be real.
  - For visualization purposes, the image may need to be re-quantized.

# Filters in the frequency domain: Example 1

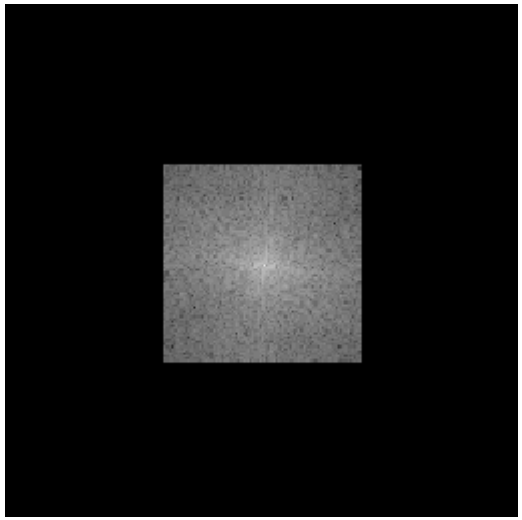
Original image

Ideal low-pass filtered image

Transform magnitude

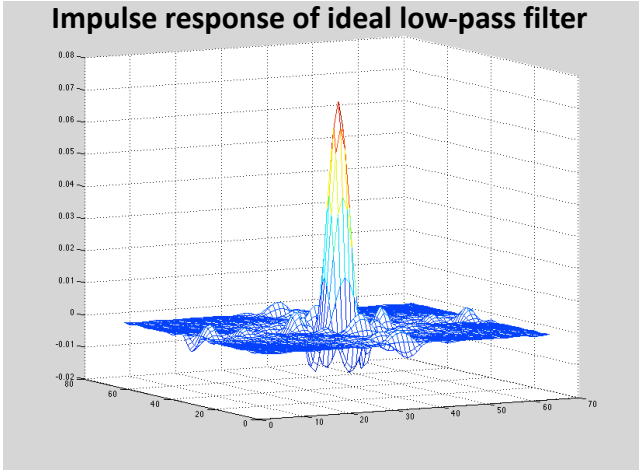


Ideal low-pass filtered transform magnitude



# Filters in the frequency domain: Ringing

- The ideal low-pass filtered image presents **ringing effect**.
- Ideal filters have as inverse transform **sinc functions**, whose lobes convolve with the transitions of the image producing these rings around contours.



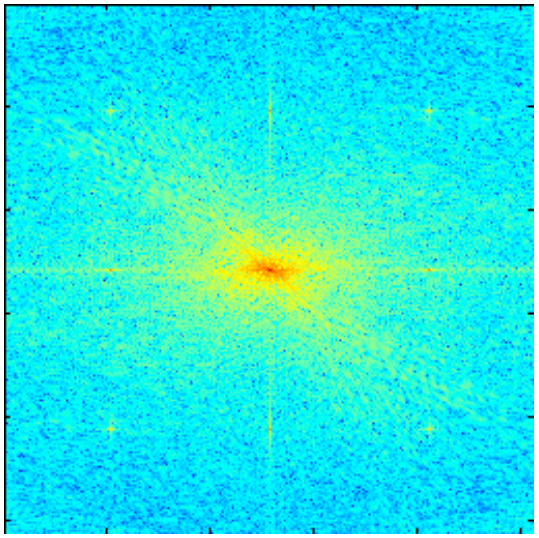

Original image

Ideal low-pass filtered image

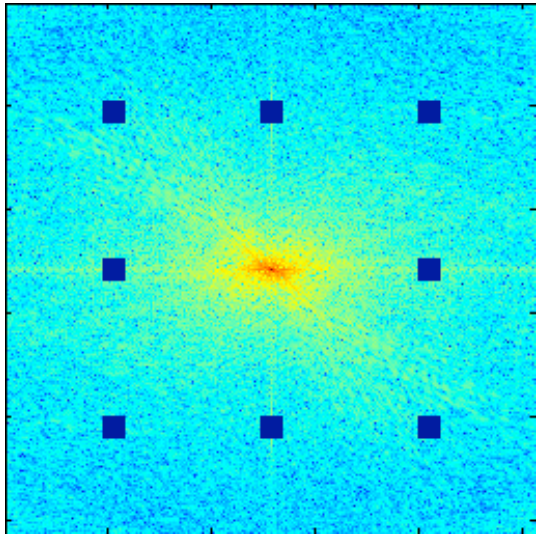
# Filters in the frequency domain: Example 2

**Original Image with "screen" noise**

**FT magnitude Original image**



**Filtered FT magnitude**



**Filtered image**

# Summary filter design

- Image filters may be implemented in the **spatial domain** (typically, when the impulse response is short enough) or in the **frequency domain** (for long impulse responses or specific applications).

- Low-pass filters can be used for **noise removal**. They "average" the values within a neighborhood given by the impulse response of the filter. Filter values usually add up 1 to preserve the **continuous component of the image**.

- High-pass filters can be used for **contour detection**. They are designed as estimations of the **gradient** or the **Laplacian**. Gradient estimations approximate the derivative in the horizontal and vertical directions and, afterwards, combine these results into a single output.

- Ideal filters are designed in the frequency domain. Their sharp frequency transitions translates into the so-called **ringing effect** (which is the effect of the **spatial sinc function** convolving with the spatial transitions).

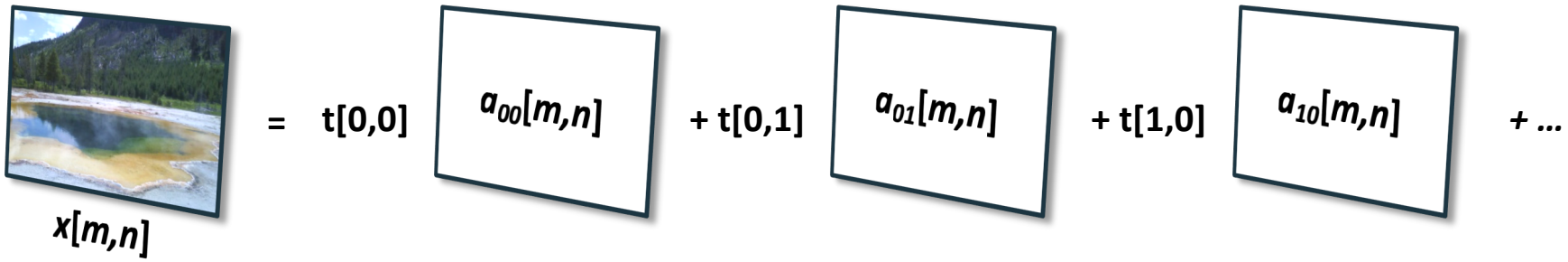
# Outline

- Low-pass and high-pass LSI filters
  - Filters design in the spatial domain
  - Filter design in the frequency domain
  - Noise reduction example

- **Linear transformations**
  - **Introduction**
  - Discrete Cosine Transform (DCT)
  - Karhunen-Loeve Transform (KLT)

# Linear transforms: Introduction (1)

- In the **space/frequency image model**, the image $x[m,n]$ is understood and represented as a **linear combination** of simpler (elementary) functions :




- Therefore, the image can be represented by a generic expression in terms of these elementary functions:

$$x[m,n] = \sum_{k=0}^{M-1}\sum_{l=0}^{N-1} t[k,l]\,a_{kl}[m,n] \qquad with \begin{cases} a_{kl}[m,n] & \textit{Set of elementary functions} \\ t[k,l] & \textit{Coefficients associated to this image and to this set of elementary functions} \end{cases}$$

# Linear transforms: Introduction (2)

- So far, we have analyzed two sets of elementary functions:
  - Impulse functions, as in the **canonical representation**.
  - Complex exponentials, as in the **Fourier representation**.

$$x[m,n] = \sum_{m'=0}^{M-1} \sum_{n'=0}^{N-1} x[m',n'] \delta[m-m',n-n']$$

$$x[m,n] = \frac{1}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} X[k,l] e^{j2\pi\left(\frac{k}{M}m + \frac{l}{N}n\right)}$$

- For any of these two representations, the set of functions being used:
  - Allows us to perfectly reconstruct the original image. These functions form a **basis of the *M*x*N* dimensional image space**.
  - Contains exactly *M*x*N* elements. The representation is said to be **complete**.
  - Is formed by **orthogonal elements**. It is an orthogonal basis.

# Linear transforms: Introduction (3)

- Every representation presents **different characteristics** that can be useful in different situations/applications:
  - Canonical representation: allows spatial detection and implementing **convolution**
  - Fourier representation: allows **frequency analysis**
- Other linear representations are possible. They present other characteristics while preserving the completeness and orthogonality.
- To obtain the coefficients to create the representation, it is necessary to compute a **linear transform** on the input data.

# Linear transforms: Direct linear transform

- The expression that, given a space representation $a_{kl}[m,n]$ and an image $x[m,n]$, obtains the coefficients representing the image in this space is the **direct transform**:

$$t[k,l] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m,n] a_{kl}^{*}[m,n]$$

Signal analysis

Forward transform

Coefficient computation

$$t[k,l] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} x[m,n] a_{kl}^{*}[m,n]$$

Signal synthesis

Inverse transform

Signal computation

$$x[m,n] = \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} t[k,l] a_{kl}[m,n]$$

# Linear transforms: Separable transform

- If 2D elementary functions can be expressed as a product of two 1D elementary functions, the transform is said to be **separable**. If the two 1D elementary functions are the same, the transform is said to be **symmetric**.

$$a_{kl}[m,n] = a_k[m]b_l[n] = a_k[m]a_l[n]$$

- Symmetric and separable 2D transforms allow a **simpler expression**:

$$t[k,l] = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} a_{kl}^*[m,n]x[m,n] = \sum_{n=0}^{N-1} a_l^*[n]\sum_{m=0}^{M-1} a_k^*[m]x[m,n]$$

# Linear transforms: Energy preservation

- Depending the normalizations of the direct and inverse transforms, orthogonal and complete transforms can be forced to have elementary functions of unitary norm. In these cases, the transform is said to be an **unitary transform** (**orthonormal basis**).

- In the case of unitary transforms, the **signal energy is preserved** when applying the transform. This is an extension of the **Parseval's theorem**:

$$\sum_{k=0}^{M-1}\sum_{l=0}^{N-1}\|t[k,l]\|^2 = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\|x[m,n]\|^2$$

- Nevertheless, if the transform is not unitary only due to a normalization constant, the **relative relevance in terms of energy** between the various transformed coefficients is kept:

  (In several applications non-unitary transform definitions are preferred since they present other useful properties, e.g.: **fast implementations**)

# Outline

- Low-pass and high-pass LSI filters
  - Filters design in the spatial domain
  - Filter design in the frequency domain
  - Noise reduction example

- **Linear transformations**
  - Introduction
  - **Discrete Cosine Transform (DCT)**
  - Karhunen-Loeve Transform (KLT)

# Discrete Cosine Transform: Definition

- The **Discrete Fourier Transform** allows analyzing the image in the frequency domain but implies the use of **complex exponentials** as elementary functions, which produces complex transformed coefficients.
- In turn, the **Discrete Cosine Transform** (**DCT**) uses as elementary functions cosine functions which produce **real transformed coefficients**.
- There exist several ways of defining the representation **in terms of cosine functions**, depending on the type of symmetry and normalization that are to be used. One of the most common ones is the so-called **DCT-type II**:

$$x[m,n] = \frac{\alpha(m)\alpha(n)}{MN} \sum_{k=0}^{M-1} \sum_{l=0}^{N-1} X_{DCT}[k,l] \cos\left[\frac{(2m+1)k\pi}{2M}\right] \cos\left[\frac{(2n+1)l\pi}{2N}\right]$$

where $\alpha(u) = \begin{cases} 1 & u = 0 \\ 2 & u \neq 0 \end{cases}$

# Discrete Cosine Transform: Elementary functions (1)

- The elementary DCT functions of type-II are **separable and symmetric**

$$a_{kl}[m,n] = a_k[m]a_l[n] \qquad a_k[m] = \alpha(k)\cos\left[\frac{(2m+1)k\pi}{2M}\right] \qquad \alpha(u) = \begin{cases} 1 & u = 0 \\ 2 & u \neq 0 \end{cases}$$

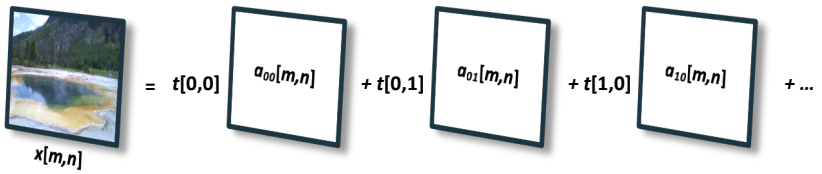
and the direct Discrete Cosine Transform is defined as:

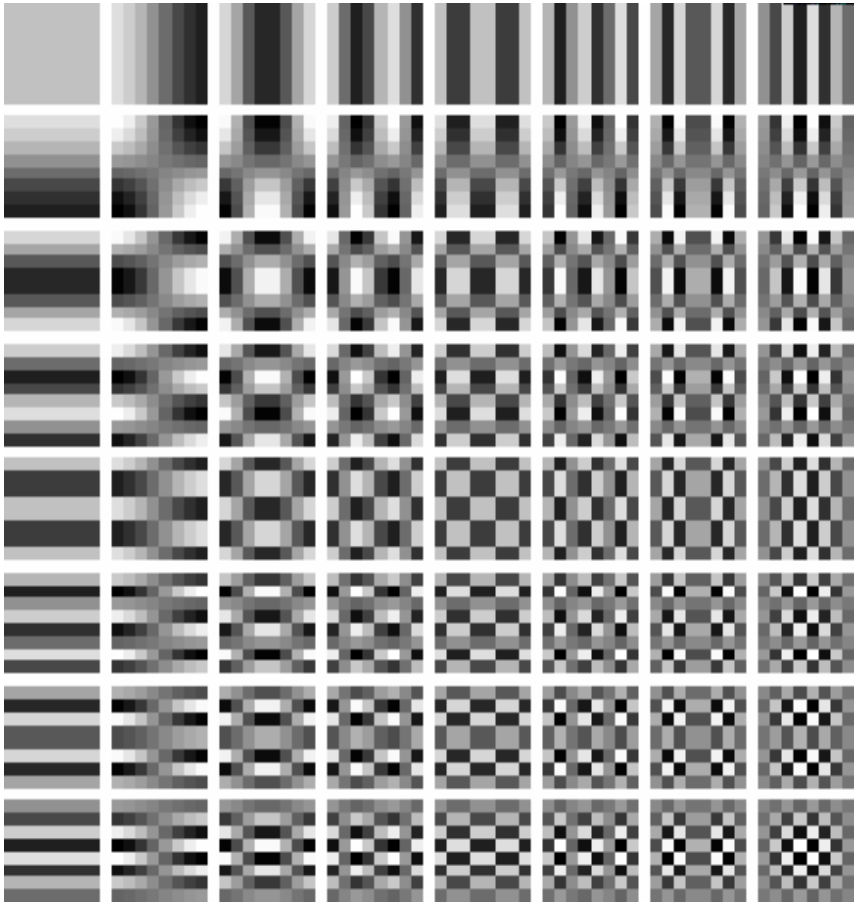
$$X_{DCT}[k,l] = \alpha(k)\alpha(l)\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} x[m,n]\cos\left[\frac{(2m+1)k\pi}{2M}\right]\cos\left[\frac{(2n+1)l\pi}{2N}\right] \qquad \alpha(u) = \begin{cases} 1 & u = 0 \\ 2 & u \neq 0 \end{cases}$$

- This representation in terms of separable and symmetric functions enables **fast implementations**:
  - Two 1D transforms can be used
  - The use of images of size $2^B$ and the symmetries of the cosine transform allows **FFT-like implementations**.

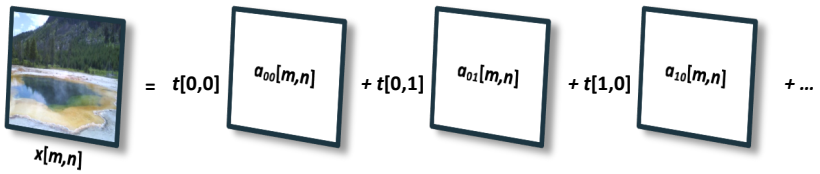
# Discrete Cosine Transform: Elementary functions (2)

- Given the real nature of the DCT elementary functions, they can be **visualized** (after normalization and quantization).
- Elementary functions are shown in increasing ***k* value by rows** and in increasing ***l* value by columns**.
- In this representation, images have been assumed to be of size ***M* = *N* = 8**.
- This is a common practice since usually (mainly in compression) **square, $2^B$ size sub-images** are analyzed rather than the whole image at once.

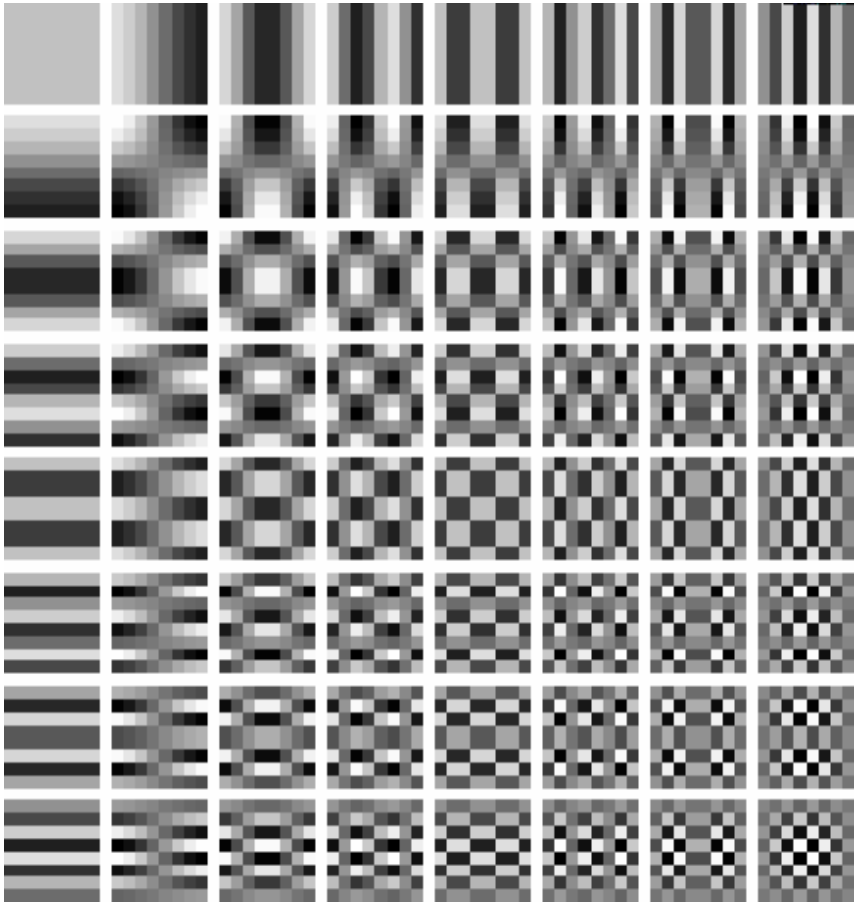


# Discrete Cosine Transform: Elementary functions (3)



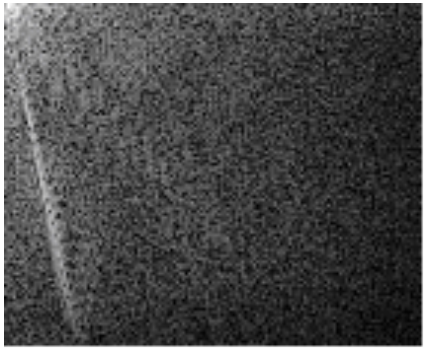

- As $M = N = 8$, a **64 Dimensional-space** has been defined
- The 64 elementary functions form **the DCT basis of the 64D-space**.
  - Note that each elementary function has 64 samples (pixels)
- The DCT basis is formed by a set of **orthogonal vectors**.
  - The cosine functions that form the elementary functions are orthogonal.



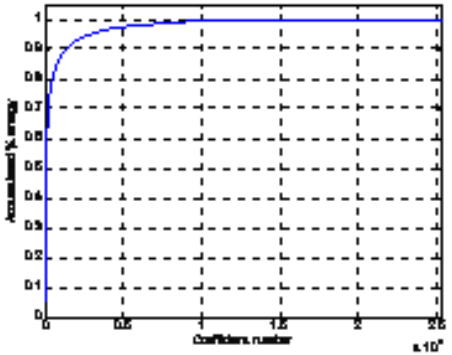
# Discrete Cosine Transform: Good compactness

- The DCT **compacts the energy** of the *M*x*N* original samples into a much lower number of transformed samples.

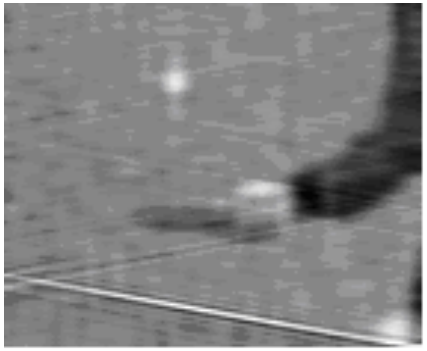
Original Image



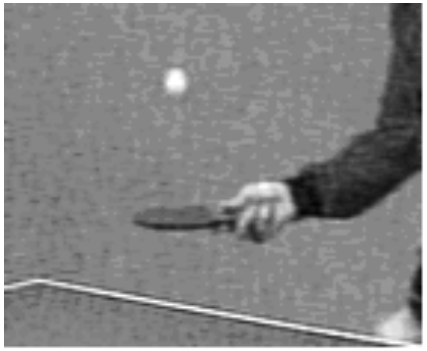
DCT coefficients



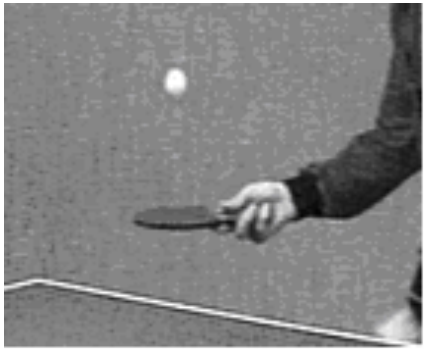
Energy distribution



NCoef = 396 (Energy:79%)



NCoef = 1.584 (Energy:91%)



NCoef = 3.168 (Energy:95%)

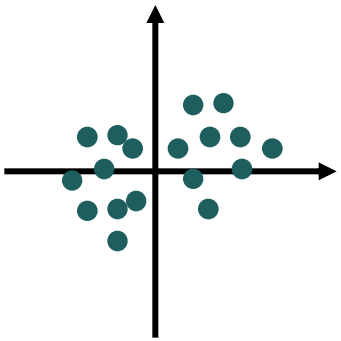
# Outline

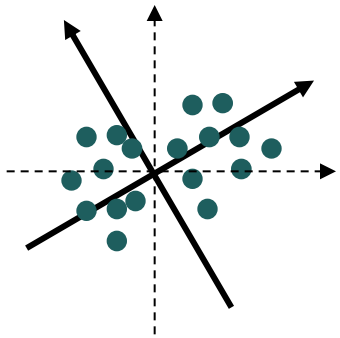
- Low-pass and high-pass LSI filters
  - Filters design in the spatial domain
  - Filter design in the frequency domain
  - Noise reduction example

- **Linear transformations**
  - Introduction
  - Discrete Cosine Transform (DCT)
  - **Karhunen-Loeve Transform (KLT)**

# Karhunen-Loeve Transform: Optimal compactness (1)

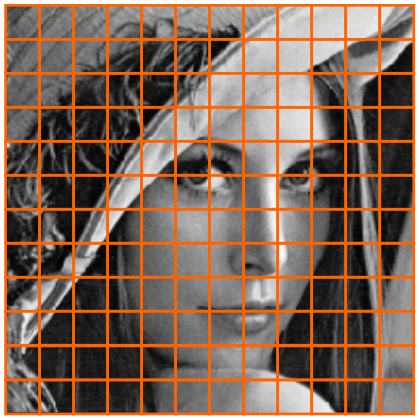
- Even though the DCT compacts the energy of the information into a few coefficients, **it is not the optimal transform** in the sense of being the transform that most energy compacts into the smallest number of transformed coefficients.
- The optimal transform in terms of **energy compaction** is the so-called Karhunen-Loeve Transform (**KLT**) and it is a transform that takes advantage of the **statistical characteristics** of the signal to be processed.
- The main concept consists in selecting the **representation that better adapts to the variability of the data**

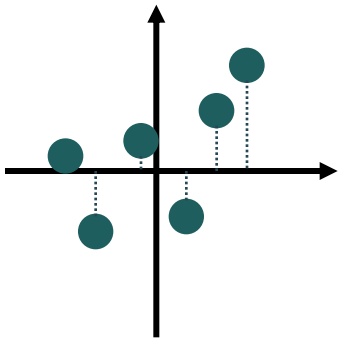


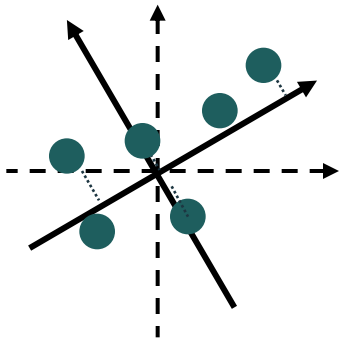


# Karhunen-Loeve Transform: Optimal compactness (2)

- Analyze the data distribution and select as vectors of the basis, those vectors that are in the directions of **maximal variance of the data distribution**.
- Example:
  - An image is partitioned into sub-images of size 8x8 pixels (blocks)
  - **Each one of these blocks is a point in a 64D-space**
  - For a given image (or collection of images), these points have a specific structure.
- By **analyzing the covariance matrix** of the set of points, the directions of maximal variance can be established. They are represented by the **eigenvectors** of the covariance matrix.
- The first direction provides the best representation of the set of points with only one component: it **minimizes the energy of the representation error**. The process can be iterated to obtain a complete orthogonal basis.







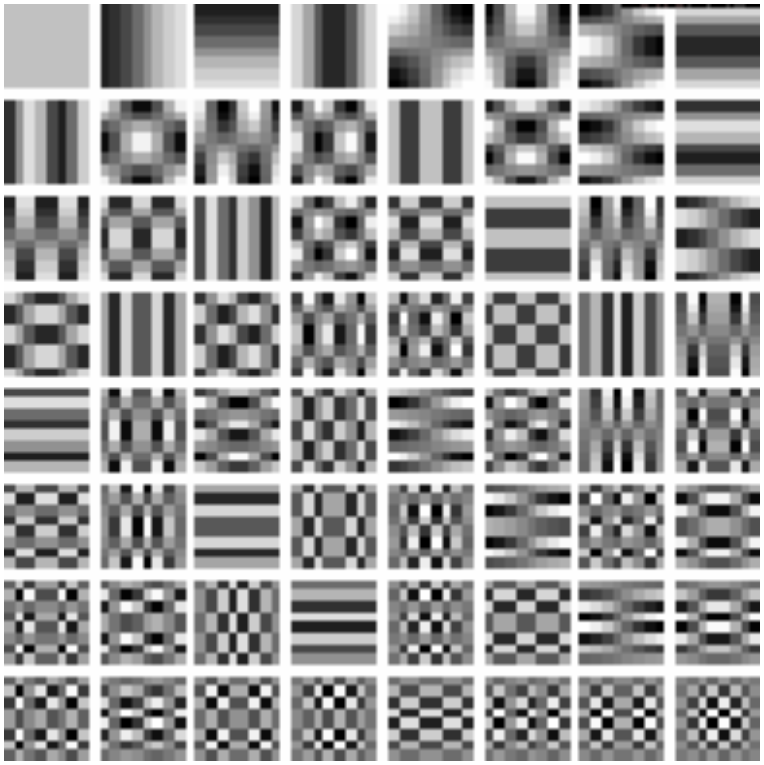
Example for blocks of 2 pixels

# Karhunen-Loeve Transform: Weak and strong points

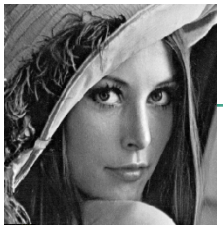
- The KLT is the **optimal transform** in terms of compacting the energy in the minimum number of transformed samples.
- Nevertheless, the KLT basis is not a fixed one but **it is data dependent**. It exploits the statistical properties of the data.
  - For each collection of data, a new basis has to be computed.
  - The computation of the basis is neither simple nor recursive.

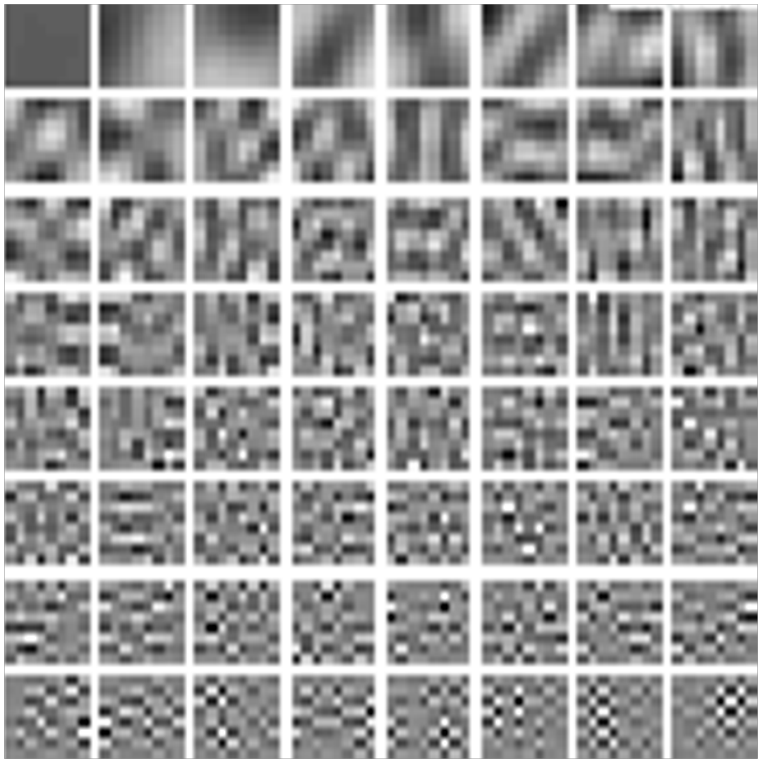
- Usually, given this data dependency, the optimal vectors cannot be represented in terms of analytical functions and, therefore, the elementary functions are not easy to handle:
  - KLT elementary functions are **not separable**
  - Fast implementations of the KLT are not possible

# How does DCT compare with KLT? (1)



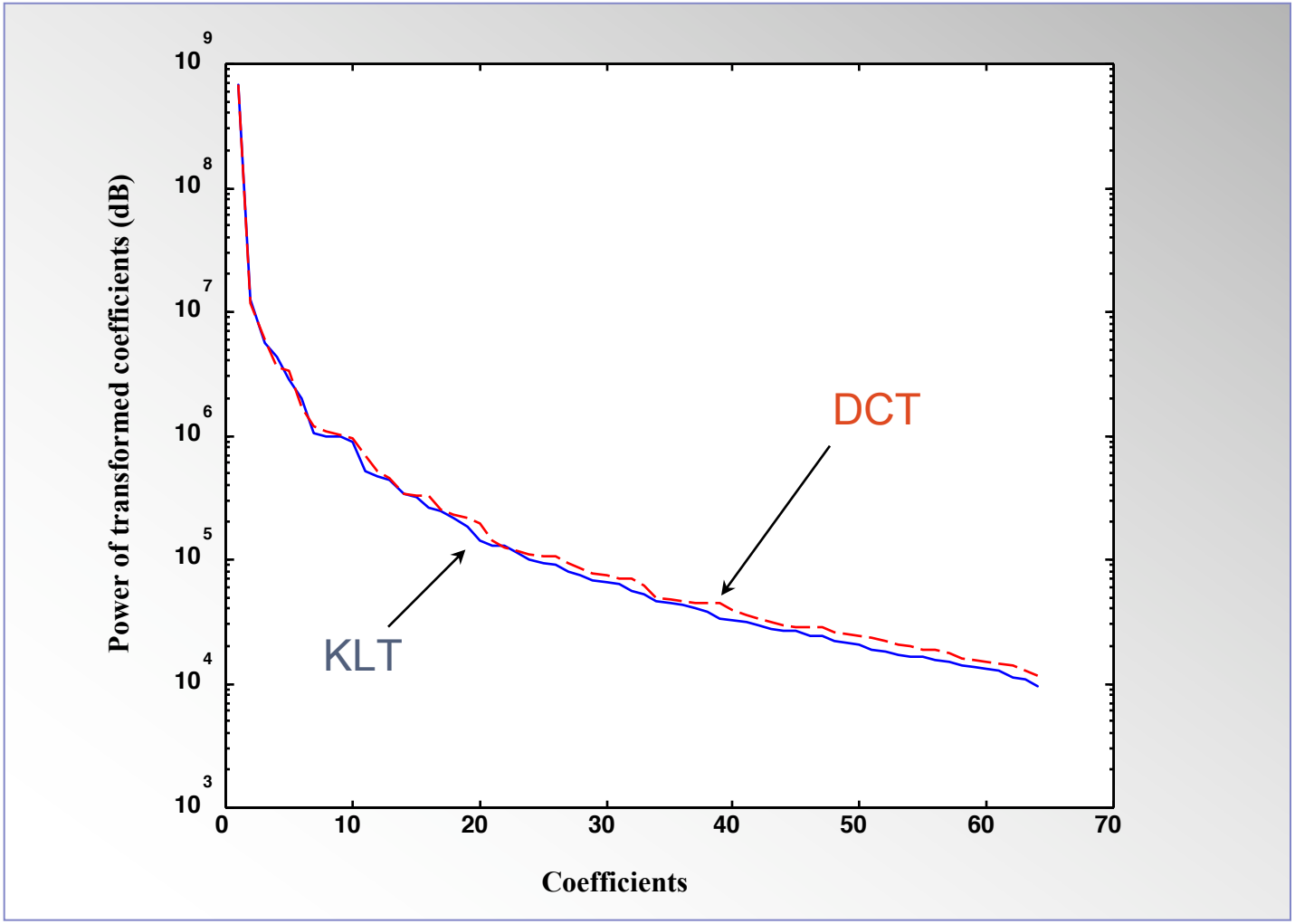
DCT basis





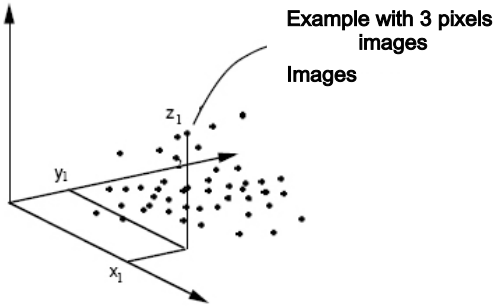
KLT basis

# How does DCT compare with KLT? (2)

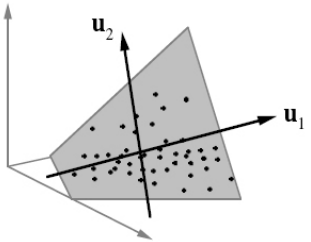


# Application of biometry: Face recognition using KLT

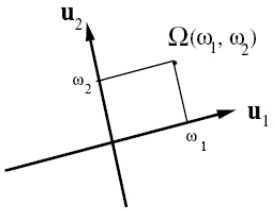
- Face images (with common pose) are very redundant. The face data lies on a low dimensional subspace. The KLT gives a global compact representation of this class of images: **Face subspace**.




3D space




2D subspace




Eigenvectors

- In the case of **face recognition**, the usual way to work is:
  - Use face images of the same size N = mxn.
  - Principal Component Analysis (PCA): Use KLT to estimate the **face representation space** of dimension M << N.
    - The M eigenvectors that span the face space are usually referred to as **eigenfaces**.
  - A linear combination of eigenfaces can describe any face image

# Application of biometry: Estimation of the face space

- Face images **do not perfectly lie** on an M-dimensional space.
- All eigenvectors and their associated eigenvalues are studied. The **M vectors with higher associated eigenvalues** are assumed to form the face space (eigenfaces). Those of lower energy are said to describe the noise and rejected.


- A new face image is represented as a linear combination of these eigenfaces :
  - Every person to be recognized is represented by a set of M transformed coefficients values: a point in the face space.

# Summary and Conclusions

- There exist different representations of images in terms of **linear combinations of elementary functions**. Each of these representations implies a given **linear transform**.
- Each transform has different **properties and characteristics** that makes it useful for specific applications. Common properties are: **completeness, separability and orthogonality**.
- The **Discrete Cosine Transform** is a complete, separable and orthogonal transform whose **transformed coefficients are real** and presents good compaction characteristics.
- The **Karhunen-Loeve Transform** is a data-dependent transform which is optimal in the sense of energy compactness and therefore allows space dimensionality reduction. The **basis is data-dependent** and it has to be computed for each data collection.