Basic Image Processing

PPKE-ITK

Lecture 4.

Spatial Filtering

- **Recap: Gaussian Smoothing – efficient for Gaussian noise, but…**

Image with salt and pepper noise

Gaussian blur with convolution

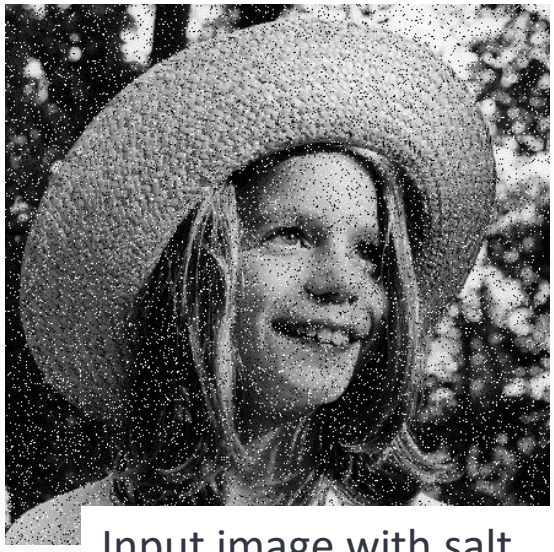
Spatial Filtering

- **Rank filter:**
  1. Consider the actual pixel and its neighborhood (e.g. 3×3=9 pixel sized window),
  2. Sort the observed pixel values according to gray level,
  3. Take the *k-th* value from this row as the new pixel value
- **Median filter**: k is the middle pixel value in the row:

  $k=[(2W+1)^2-1]/2$, if W is the half side size of the neighborhood
- Non-Linear filter

Spatial Filtering

- **Median filter:** replaces each pixel with the *median value* of its analyzed neighborhood. (Median value: the center element of sorted values)
  - Very effective against impulse („salt and pepper”) noise:



Input image with salt and pepper noise

Blur with convolution

Median filter

Spatial Filtering

- **Median filter:** replaces each pixel with the *median value* of its analyzed neighborhood. (Median value: the center element of sorted values)
  - Very effective against impulse („salt and pepper”) noise:


Original image

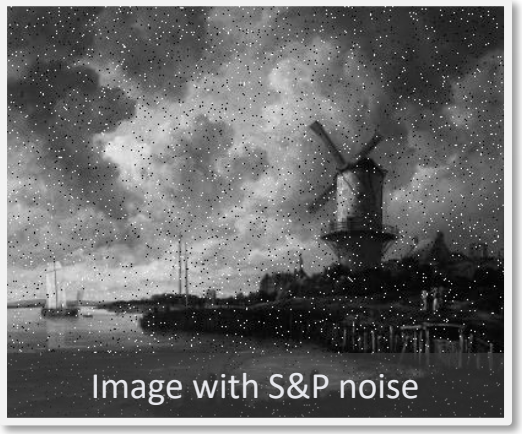


Image with S&P noise


Median filtered S&P noise

  - Not so effective against Gaussian noise.

Spatial Filtering

- **Order statistic filtering:**
  - Based on the sorted pixel intensity levels in the analyzed neighborhood.
  - If after sorting…
    - we take the middle element, we get back the median filter.
    - We take the maximum element to filter „pepper” and min to filter „salt” noise.

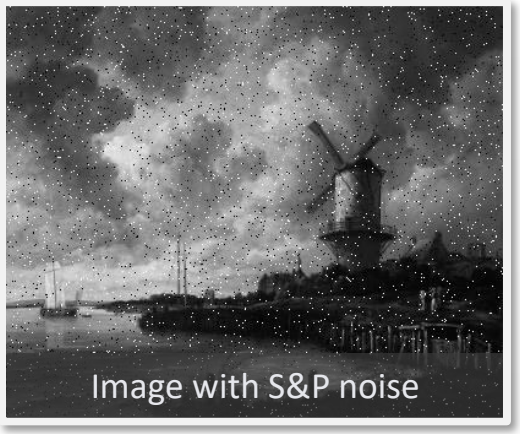
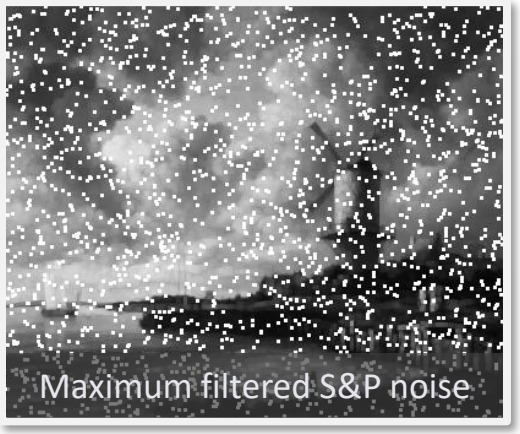


Image with S&P noise


Maximum filtered S&P noise


Minimum filtered S&P noise

- But max filter will highlight „salt”, while min filter will highlight „pepper”.

Spatial Filtering

- **Order statistic filtering:**
  - Mid-point filtering:
    - works well on Gaussian or uniform noise

$$y(n_1, n_2) = \frac{1}{2}\left(\max_{(m_1,m_2)\in N} \{x(m_1, m_2)\} + \min_{(m_1,m_2)\in N} \{x(m_1, m_2)\} \right)$$

  - Alpha-trimmed mean filter:

$$y(n_1, n_2) = \frac{1}{|N| - \alpha} \sum_{(m_1,m_2)\in N_r} x(m_1, m_2)$$

    - Where N_r is a reduced neighborhood, not containing the lowest and highest α element of N.
    - If α = 0, we get back the arithmetic mean.
    - If α = |N|-1, we get back the median filter.

Wallis Operator

- The Wallis operator can help to adjust local contrast:

$$y(n_1, n_2) = \left[x(n_1, n_2) - \bar{x}(n_1, n_2)\right] \frac{A_{\max}\sigma_d}{A_{\max}\sigma_l(n_1, n_2) + \sigma_d} + \left[p\bar{x}_d + (1-p)\bar{x}(n_1, n_2)\right]$$

  - where σ_l the local contrast: $\sigma_l(n_1, n_2) = \frac{1}{|N|}\sqrt{\sum\sum_{(n_1,n_2)\in N}\left(x(n_1, n_2) - \bar{x}(n_1, n_2)\right)^2}$
  - $\bar{x}$ is the local average: $\bar{x}(n_1, n_2) = \frac{1}{|N|}\sum\sum_{(n_1,n_2)\in N} x(n_1, n_2)$
  - σ_d is the desired local contrast, $\bar{x}_d$ is the desired mean value of all pixels, p is a weighting factor of the mean compensation, while A_{max} is maximizing the local contrast modification.

Wallis Operator

- We can describe the image the following way:

$$x(n_1, n_2) = \underbrace{\left[x(n_1, n_2) - \bar{x}(n_1, n_2)\right]}_{(1)} + \underbrace{\bar{x}(n_1, n_2)}_{(2)}$$

where (2) is the local mean and (1) is the deviation from the local mean.

- With the transformation we want to „push” the local mean and standard deviation to a predefined desired value:

$$y(n_1, n_2) = \underbrace{\left[x(n_1, n_2) - \bar{x}(n_1, n_2)\right] \frac{\sigma_d}{\sigma_l(n_1, n_2)}}_{(1)} + \underbrace{\left[p\bar{x}_d + (1-p)\bar{x}(n_1, n_2)\right]}_{(2)}$$

- We are almost there, but if the local contrast is too low, the weighting in (1) may get too high, this is why we maximize it with A_{max}:

$$\frac{\sigma_d}{\sigma_l(n_1, n_2)} \Rightarrow \frac{A_{\max}\sigma_d}{A_{\max}\sigma_l(n_1, n_2) + \sigma_d}$$

Wallis Operator

Original Image

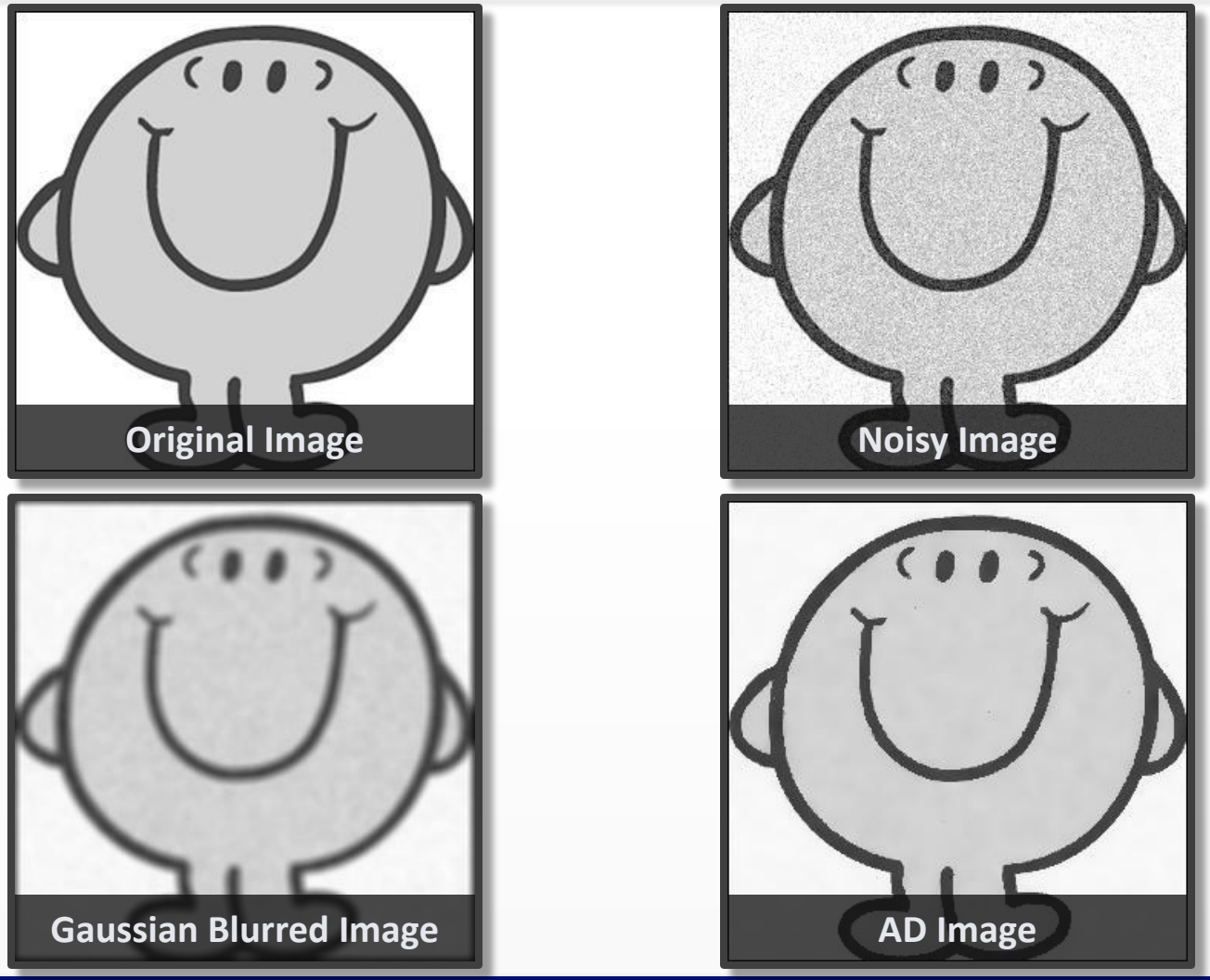
Image after applying Wallis operator

Anisotropic Diffusion

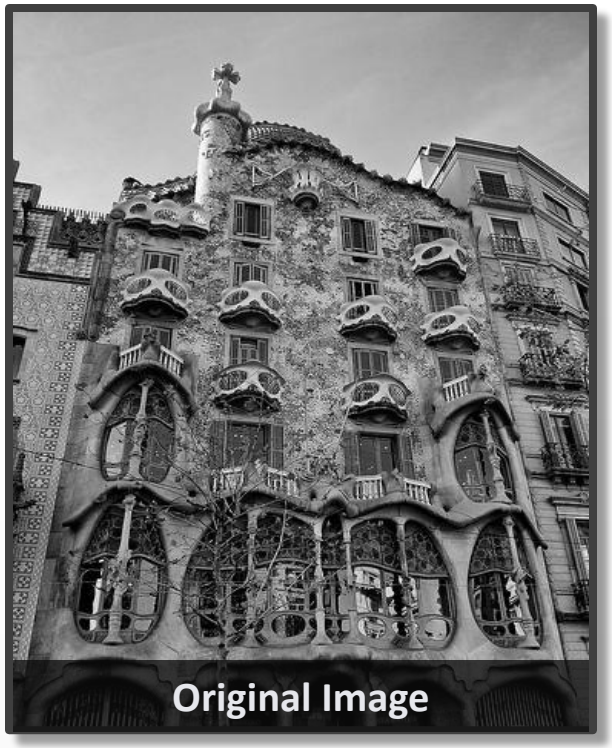
- The anisotropic diffusion is a technique aiming at reducing image noise without blurring significant parts of the image content.
- It was first proposed by Dénes Gábor in 1965 and later by Perona and Malik around 1990.
- ***Non-linear*** and ***space-variant*** transformation.
- The main idea is that the effect of blurring in each direction is inversely proportional to the gradient value in that direction:
  - allows diffusion along the edges or in edge-free territories, but penalizes diffusion orthogonal to the edge direction.
- AD is an iterative process

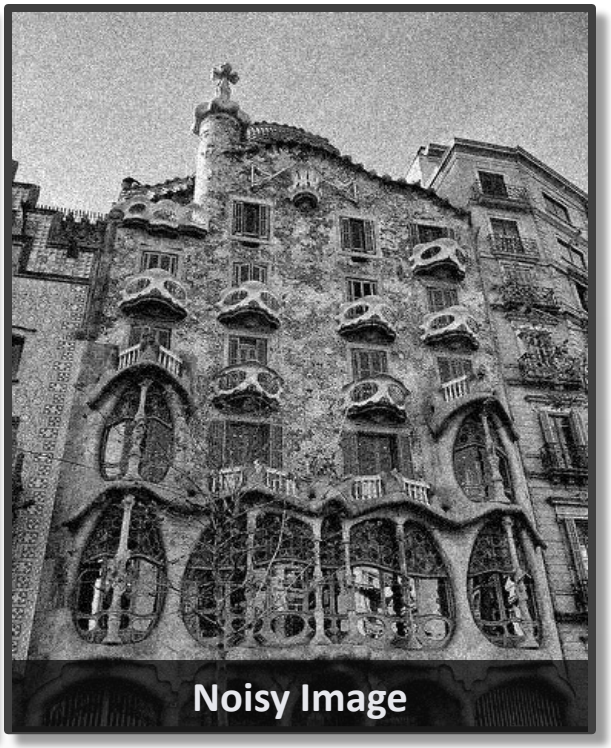
P. Perona, J Malik (July 1990). "Scale-space and edge detection using anisotropic diffusion". IEEE Tr. PAMI, 12 (7): 629–639.
D. Gabor, “Information theory in electron microscopy,” Laboratory Investigation, vol. 14/6, pp. 801–807, 1965.

Anisotropic Diffusion



Anisotropic Diffusion





Total Variation Regularization

- Assumption:
  - The image is smooth inside the objects, with jumps across the boundaries.
  - The noise component has high variation.
- The goal of ***Total Variation based noise removal*** is to minimize the total variation of the image while keep the result as close to the original input image as possible.
- It was introduced by Rudin, Osher and Fatemi in 1992.

Rudin, L. I.; Osher, S.; Fatemi, E. (1992). "Nonlinear total variation based noise removal algorithms". *Physica D* **60**: 259–268

Total Variation Regularization

- TV of the output image y is defined as the integral of the absolute gradient of the signal:

$$\mathrm{V}(y)=\sum_{n_1}\sum_{n_2}\sqrt{\left|y(n_1+1,n_2)-y(n_1,n_2)\right|^2+\left|y(n_1,n_2+1)-y(n_1,n_2)\right|^2}$$

- On the other hand, we also measure the difference between the original image x and the output image y by L_2 norm E:

$$\mathrm{E}(\mathrm{x},\mathrm{y})=\sum_{n_1,n_2}\left(x(n_1,n_2)-y(n_1,n_2)\right)^2$$

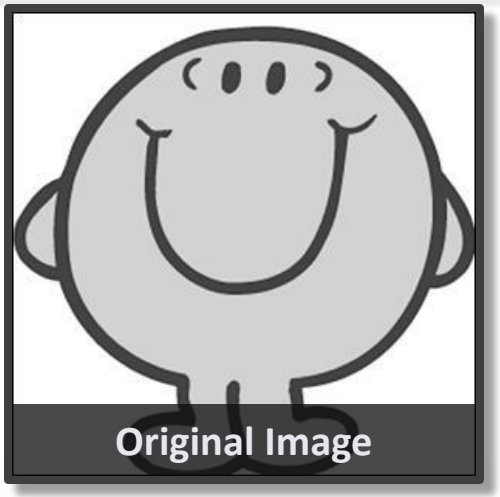
- The goal function for Total Variation based regularization:

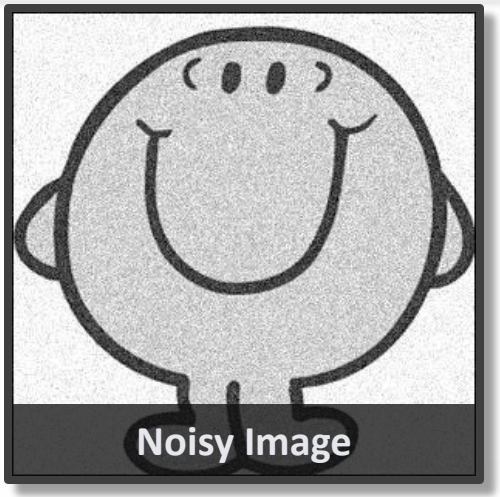
$$\hat{\mathrm{y}}=\arg\min_{y}\left[\mathrm{E}(\mathrm{x},\mathrm{y})+\lambda\mathrm{V}(y)\right]$$

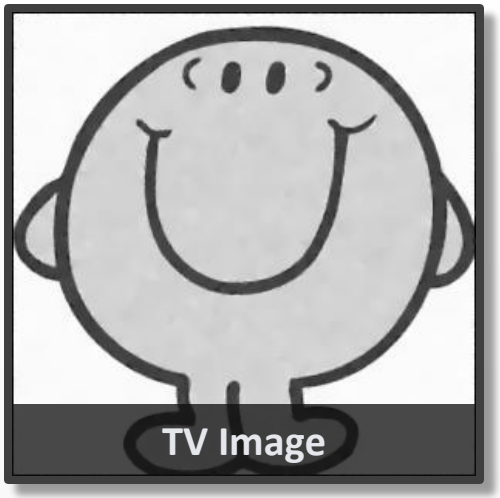
where λ is the regularization parameter.

Rudin, L. I.; Osher, S.; Fatemi, E. (1992). "Nonlinear total variation based noise removal algorithms". *Physica D* **60**: 259–268

Total Variation Regularization







Matlab Code: http://visl.technion.ac.il/~gilboa/PDE-filt/tv_denoising.html

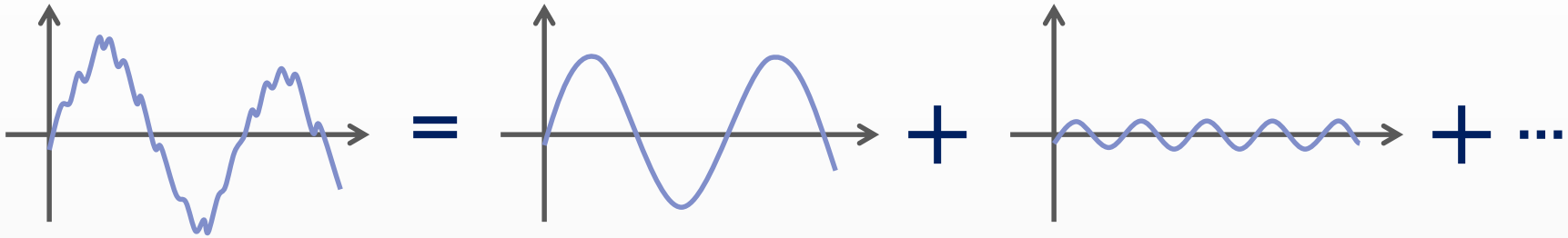
Total Variation Regularization

Matlab Code: http://visl.technion.ac.il/~gilboa/PDE-filt/tv_denoising.html

Fourier Transformation
Image filtering in the frequency domain

What is Fourier Transform?

- A transformation maps data between (different) domains.
- The Fourier Transform changes between the representation in the **time domain** and in the **frequency domain**.
- The information is the same in both domains, only the representation is different.
- It is a reversible transform.
- It builds on the fact that any function can be represented as a weighted sum of sinusoid functions:



- If we can describe sinusoids we can describe every function.

2D Fourier transform

- Forward transform: map an image $x(n_1, n_2)$ of size $N_1 \times N_2$ from the **spatial domain** into the $X(\omega_1, \omega_2)$ **frequency domain**

$$X(\omega_1, \omega_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\omega_1 n_1} e^{-j\omega_2 n_2}$$

  - even if the image is real the spectrum is complex due to the complex exponential factors
  - ω_1, ω_2 frequencies: continuous variables
  - $X(\omega_1, \omega_2)$ continuous Fourier transform or spectrum of the discrete image
  - Drawback: no computable representation of $X(\omega_1, \omega_2)$
  - Solution: **Discrete Fourier Transform (DFT)**: sample the continuous spectrum with equally spaced frequencies

Discrete Fourier Transform

- We sample one period of the Fourier transform in evenly spaced frequencies:

$$X(\omega_1, \omega_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\omega_1 n_1} e^{-j\omega_2 n_2}$$

The size of the image in the spatial domain is $N_1 x N_2$

The size of the image in the frequency domain will be the same: $N_1 x N_2$

$$X(k_1, k_2) = X(\omega_1, \omega_2)\Big|_{\omega_1 = \frac{2\pi}{N_1} k_1, \omega_2 = \frac{2\pi}{N_2} k_2}$$

$$k_1 = 0,1..., N_1 - 1$$
$$k_2 = 0,1..., N_2 - 1$$

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\frac{2\pi}{N_1} k_1 n_1} e^{-j\frac{2\pi}{N_2} k_2 n_2}$$

Only one period is kept

Discrete Fourier Transform

- Forward formula: gives the description of the image in the discrete frequency domain

$$X(k_1, k_2) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} x(n_1, n_2) e^{-j\frac{2\pi}{N_1}k_1 n_1} e^{-j\frac{2\pi}{N_2}k_2 n_2}$$

- Inverse Fourier transform: maps from the discrete frequency domain back to the discrete spatial domain

$$x(n_1, n_2) = \frac{1}{N_1 N_2} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} X(k_1, k_2) e^{j\frac{2\pi}{N_1}k_1 n_1} e^{j\frac{2\pi}{N_2}k_2 n_2}$$

  - algorithmically it has the same structure as the forward transform,

Discrete Fourier Transform

- DFT is an exact transform, there is no transformation error.
  - Not surprising, since we use the same image size for the representation of both $x(n_1, n_2)$ and $X(\omega_1, \omega_2)$.
- Most of the properties of continuous FT hold for DFT
  - Except linear shift of FT becomes circular shift for DFT.
- DFT and inverse DFT are computable transformations
- There are fast ways to compute the DFT: ***Fast Fourier Transform***
  - If the size of the image is *N*x*N*, then the **naive implementation** requires N^4 multiplications: N^2 for each (k_1,k_2) point.
  - The **FFT** with row/column decomposition requires only $N^2 \log_2 N$ multiplications.
- **FFT makes the Fourier transformation applicable in many practical cases.**

Interpretation of Fourier coefficients

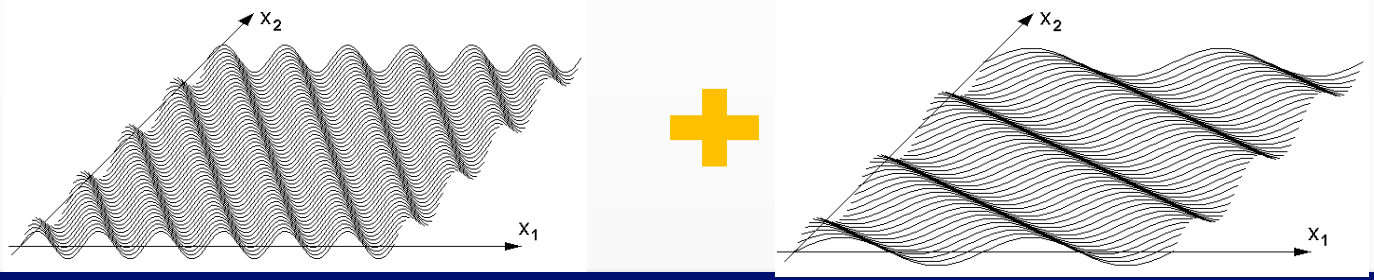
- Analogy of Fourier coefficient based representation:
  - Consider the image as a superposition of sinusoid/cosine waves with different amplitudes, frequencies and directions
  - 1D case in formulas: (x_n: signal, X_k Fourier coefficient)

$$x_n = \sum_{k=0}^{N-1} X_k \exp(j \frac{2\pi}{N} k\, n)$$

$$x_n = X_0 + X_{N/2} \cos(\pi n) + \sum_{k=1}^{N/2-1} 2\left(\mathrm{Re}(X_k) \cos\left(\frac{2\pi}{N} kn \right) - \mathrm{Im}(X_k) \sin\left(\frac{2\pi}{N} kn \right) \right)$$

X_0: real number, average of the function

  - 2D case (e.g. image) visualization:

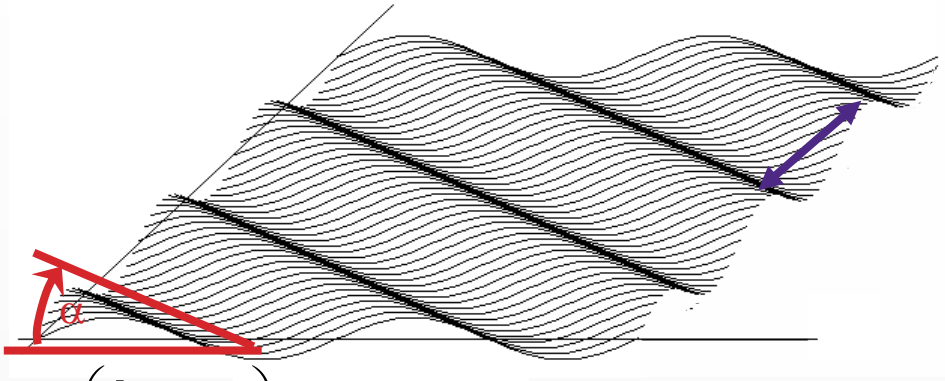


Interpretation of 2D Fourier coefficients

$$X(k_1,k_2)\cdot\exp\left(j2\pi\left(k_1\frac{n_1}{N_1}+k_2\frac{n_2}{N_2}\right)\right)+X(-k_1,-k_2)\cdot\exp\left(-j2\pi\left(k_1\frac{n_1}{N_1}+k_2\frac{n_2}{N_2}\right)\right)=$$

$$\mathrm{Re}\{X(k_1,k_2)\}\cdot 2\cdot\cos\left(2\pi\left(k_1\frac{n_1}{N_1}+k_2\frac{n_2}{N_2}\right)\right)-\mathrm{Im}\{X(k_1,k_2)\}\cdot 2\cdot\sin\left(2\pi\left(k_1\frac{n_1}{N_1}+k_2\frac{n_2}{N_2}\right)\right)$$

- real part of an $X(k_1,k_2)$ Fourier coefficient is the **amplitude** of a cos-wave, while the imaginary part is the amplitude of a sinusoid wave
- **wavelength** and **orientation** of the waves are encoded in the (k_1,k_2) *position* coordinates of the coefficients in the 2D Fourier map



wavelength:

$$\frac{1}{\lambda}=\sqrt{\left(\frac{k_1}{N_1}\right)^2+\left(\frac{k_2}{N_2}\right)^2}$$

orientation:

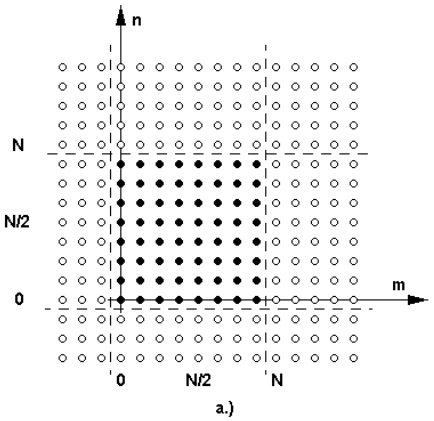
$$\alpha=\arctan\left(\frac{k_2/N_2}{k_1/N_1}\right)$$

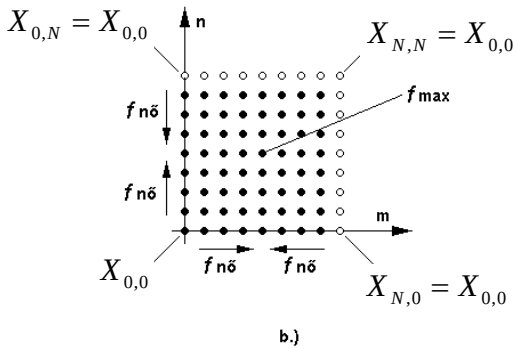
k_1=3 k_2=2

Spatial frequency:

$$f=1/\lambda$$

Illustration of the periodicity of coefficients Centered DFT: better visualization





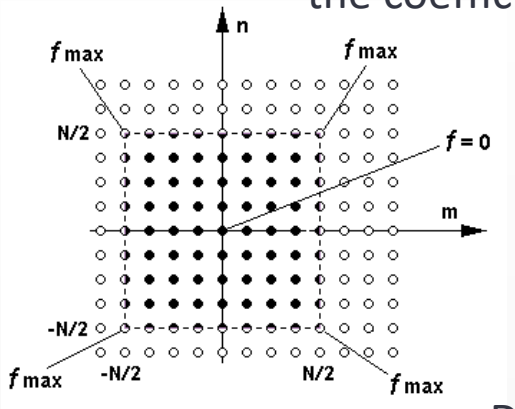
- In the center of the DFT array X_{00} is the *zero-frequency coefficient (DC component)*
- Distance from the center
  - Frequency of the corresponding sin/cos wave

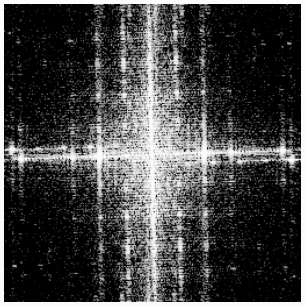
$$f = \sqrt{(k_1 / N_1)^2 + (k_2 / N_2)^2}$$

- Orientation
  - Direction perpendicular to the wavefront

$$\alpha = \arctan\left((k_2 / N_2)/(k_1 / N_1)\right)$$

Re-arrangement of the coefficient matrix



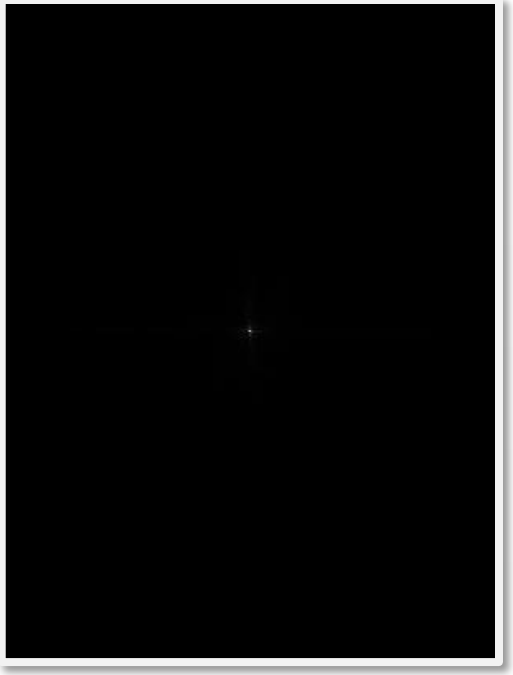


DFT abs. value image

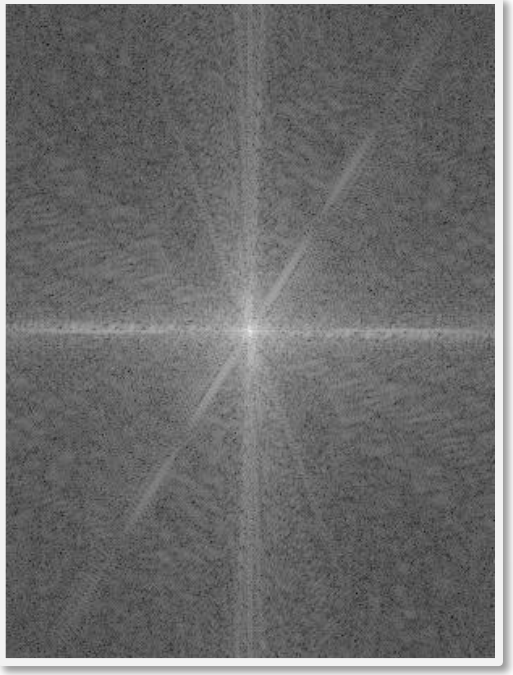
Point-wise Intensity Transformation

- **Log transformation**:
  - Commonly used to visualize the Fourier transform of an image

Original Image*



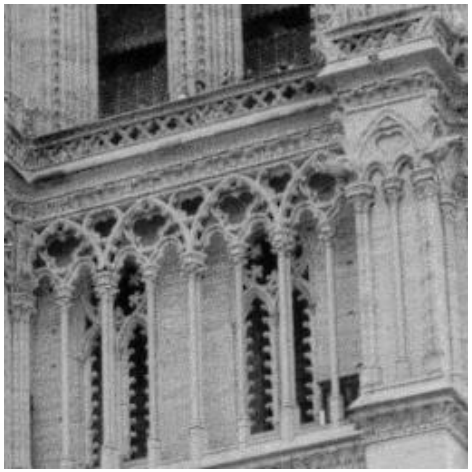
The magnitude of the DFT

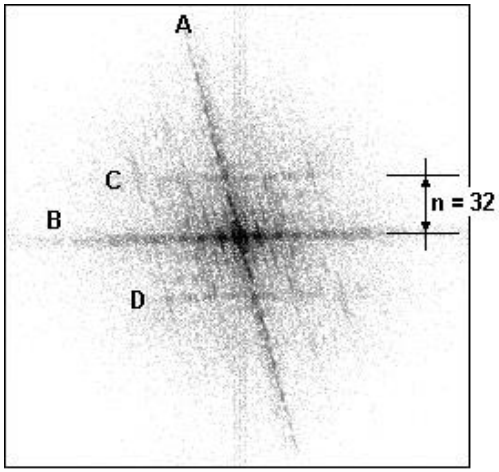


Log of the magnitude of the DFT

*Chez Mondrian by András Kertész (1926)

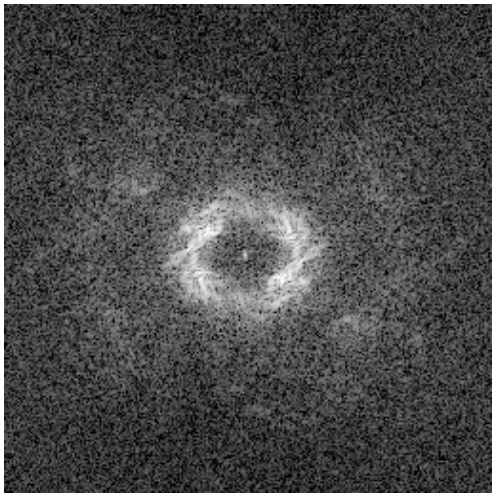
Imaged absolute values of DFT coefficients – facade of the Notre Dame Paris





- In the transformed map, directions of strong lines are <u>perpendicular</u> to the major contours in the image:
  - **A** line- horizontal ledges (párkányok)
  - **B** line- slim vertical columns.
  - **C** and **D** lines – periodic vertical patterns with the frequency „$n = \pm 32$” : decoration of the windows behind the columns

Imaged absolute values of DFT coefficients– analysing fingerprint images



- No characteristic lines in the transform
  - ← ridges of fingerprints run in any directions
- At *d* distance from the center a significant ring shaped maximum
  - ← the average spatial frequency of the fingerprint ridges is *d*-times the basic frequency f_b, and there exist no nominant directions

Filtering in the DFT space

- Low-Pass Filter (LPF):
  - Filtering out the large spatial frequencies

a) b)

Result of filtering out large frequencies. Erased all coeff. a.) above 16 f_b, b.) above 8 f_b

Slide credit ® Prof. Vladimir Székely, BME

Filtering in the DFT space

- High-Pass Filter (HPF)
  - Filtering out the low spatial frequencies

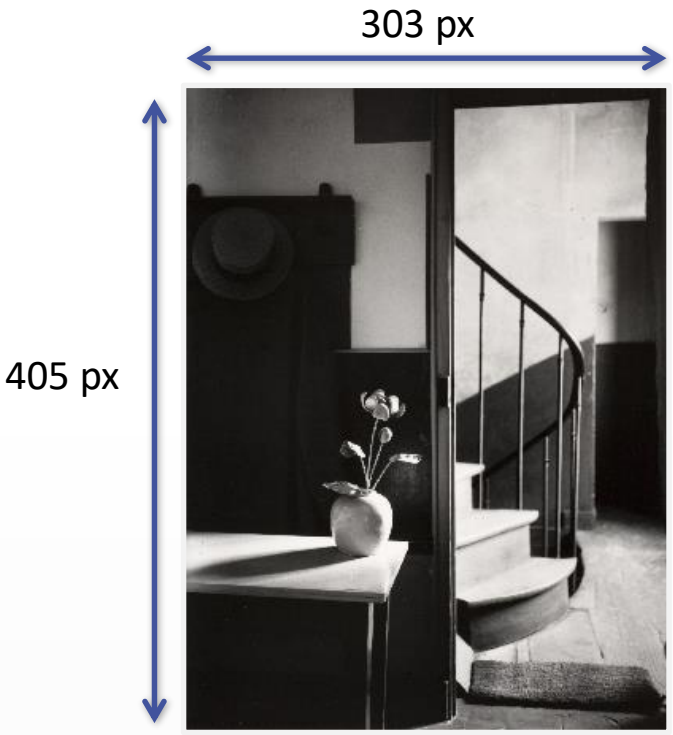
a)

b)

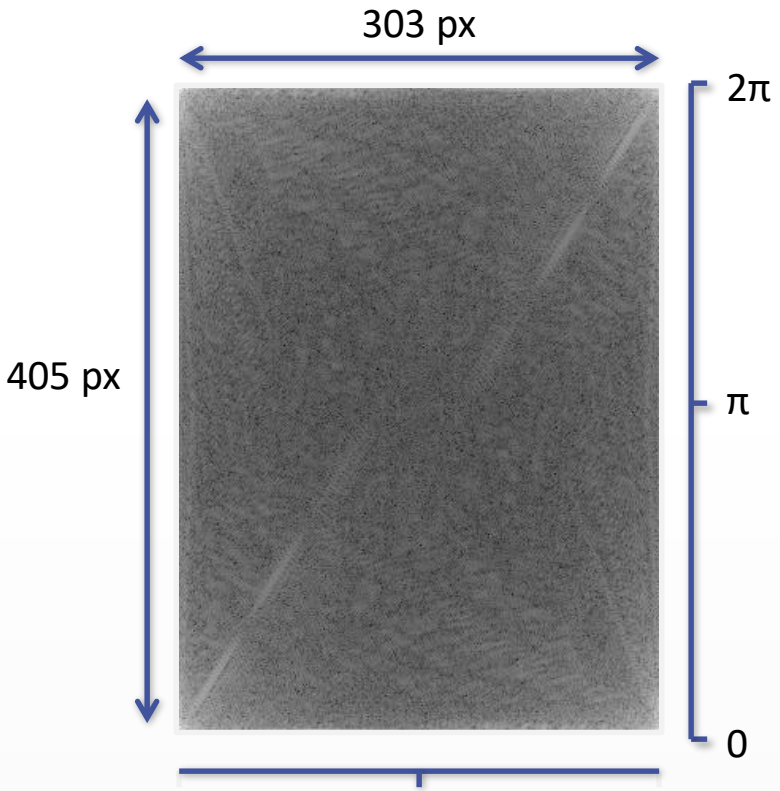
Result of filtering out large frequencies. Erased all coeff. a.) below 4 f_b, b.) below 10 f_b

Slide credit ® Prof. Vladimir Székely, BME

Discrete Fourier Transform



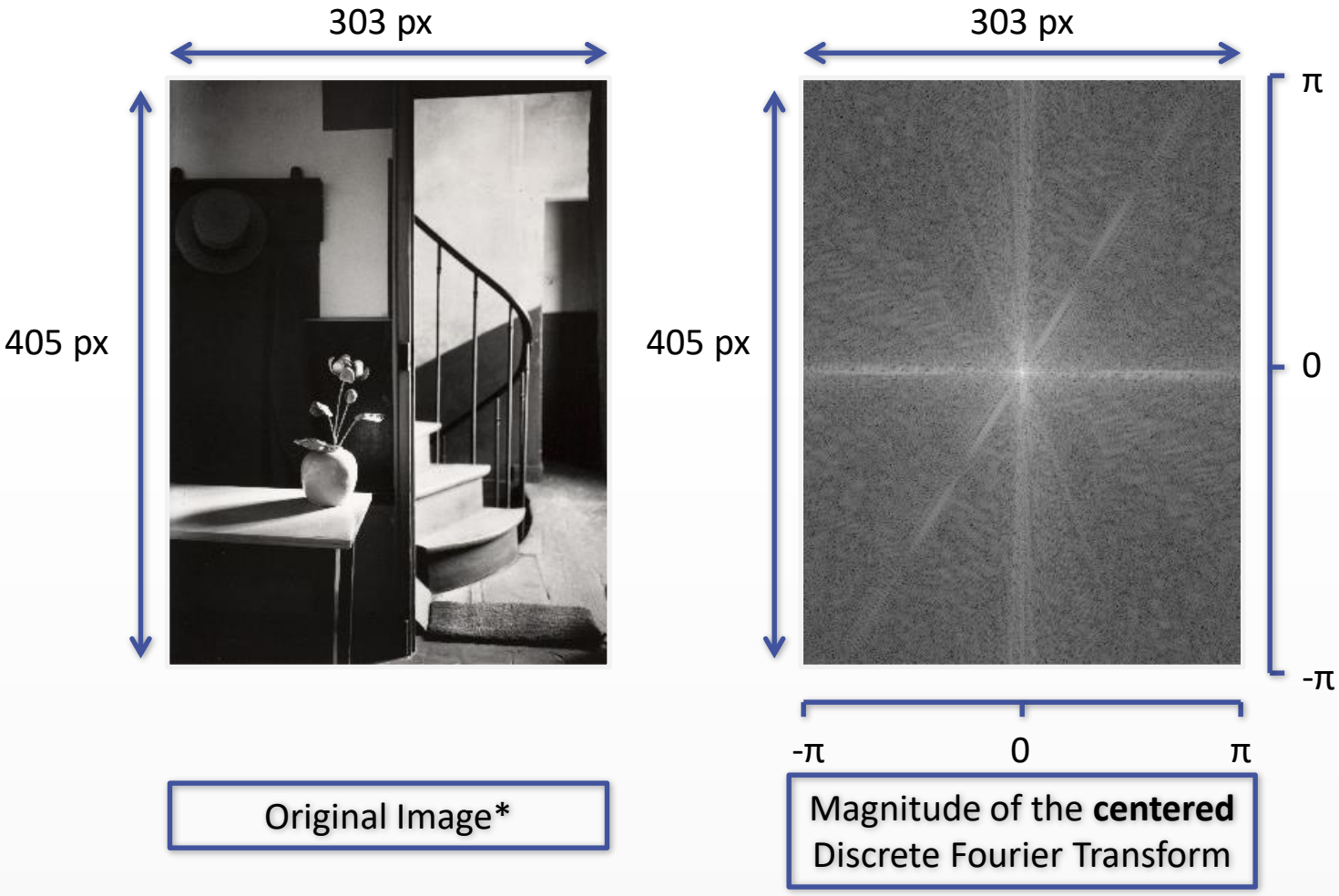
Original Image*



Magnitude of the Discrete Fourier Transform

*Chez Mondrian by András Kertész (1926)

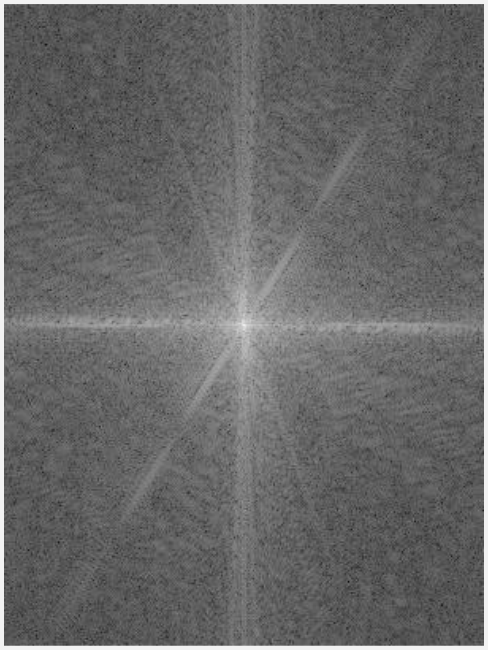
Discrete Fourier Transform



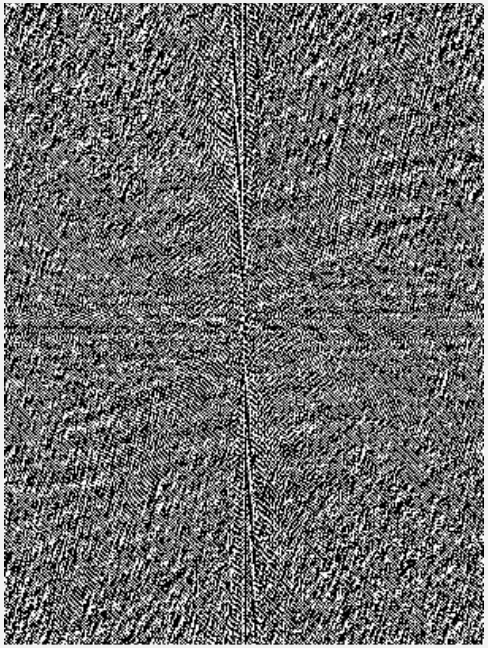
*Chez Mondrian by András Kertész (1926)

Discrete Fourier Transform

Original Image*



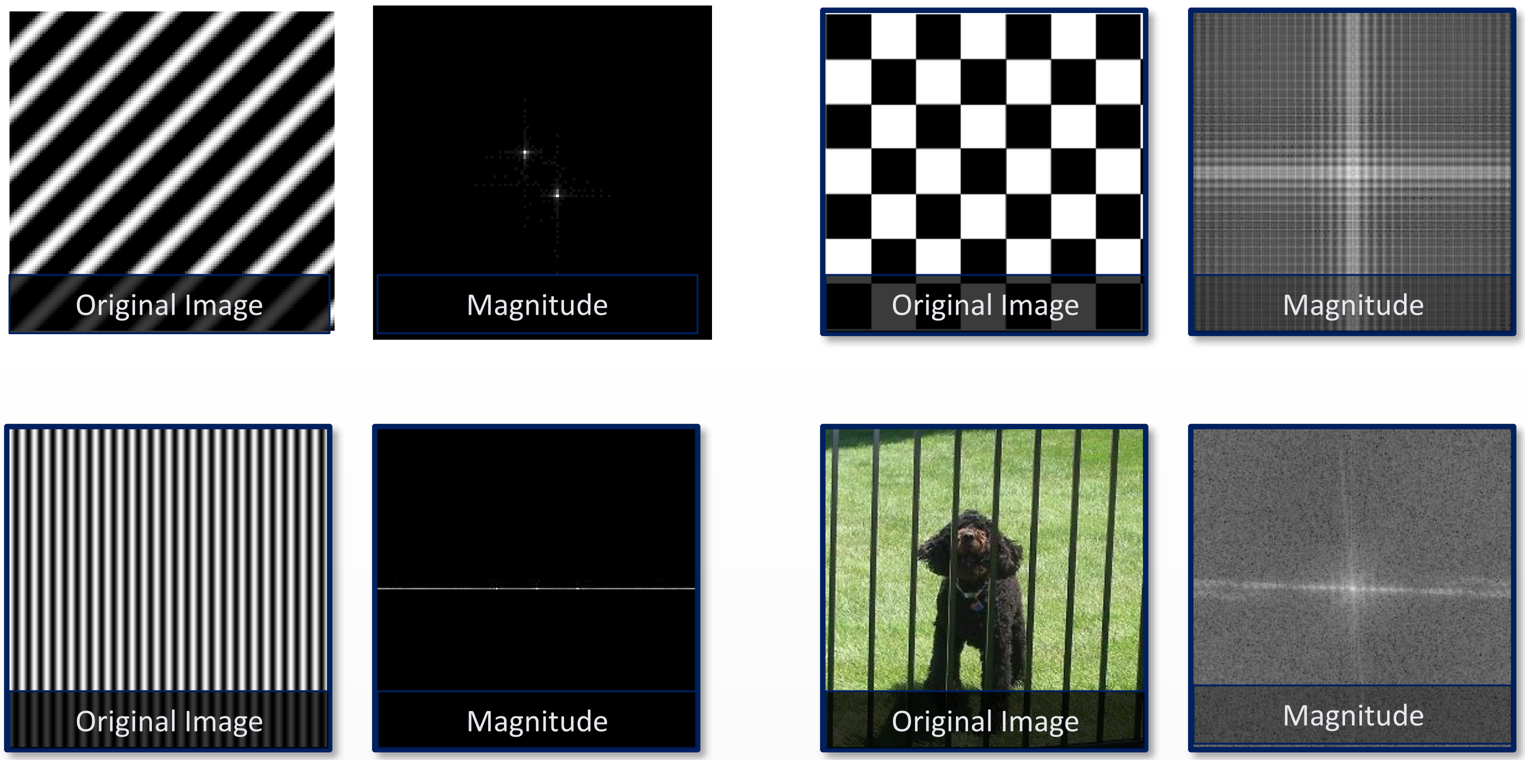
Magnitude of the **DFT**



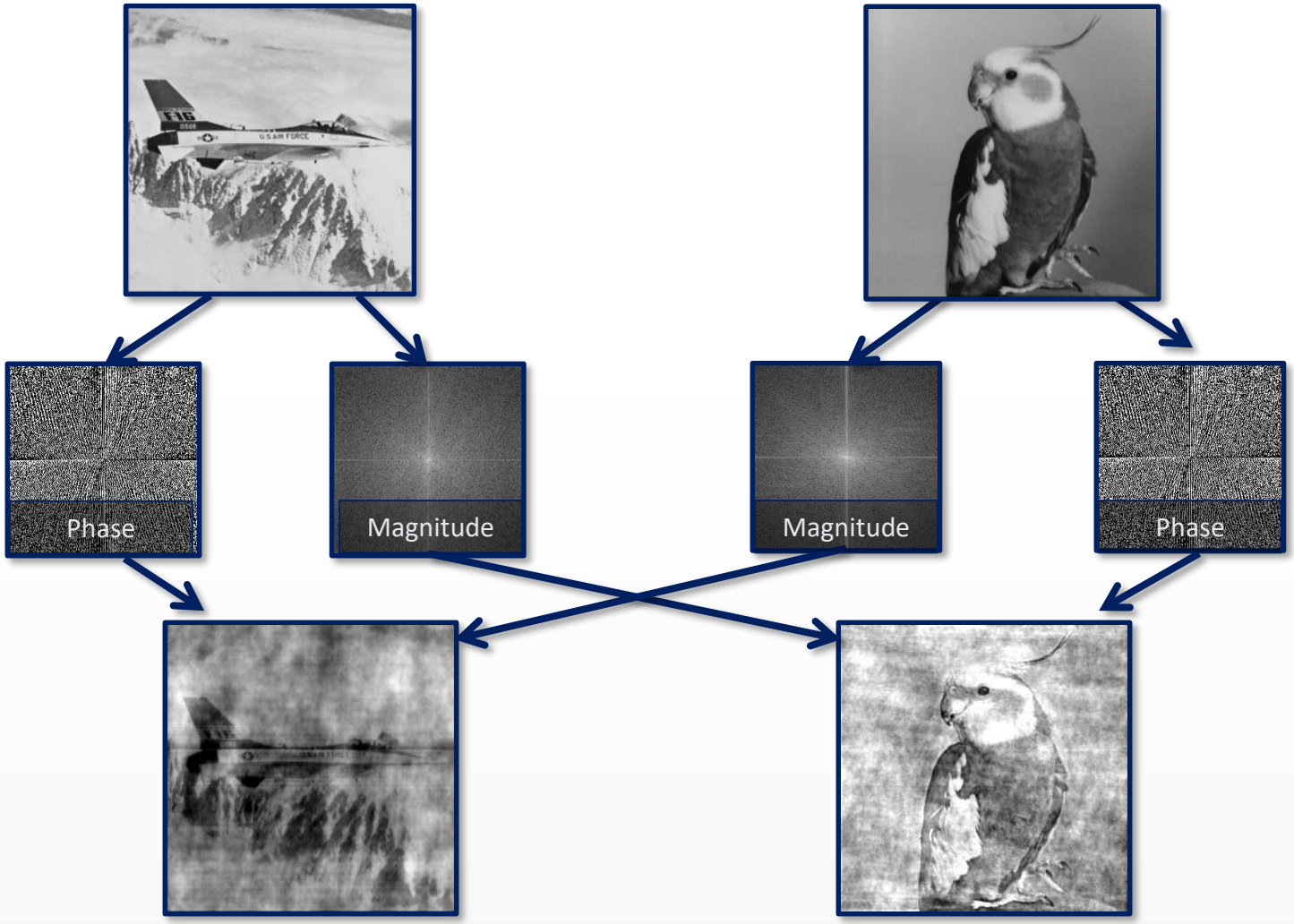
Phase of the **DFT**

*Chez Mondrian by András Kertész (1926)

Discrete Fourier Transform - Examples



Discrete Fourier Transform

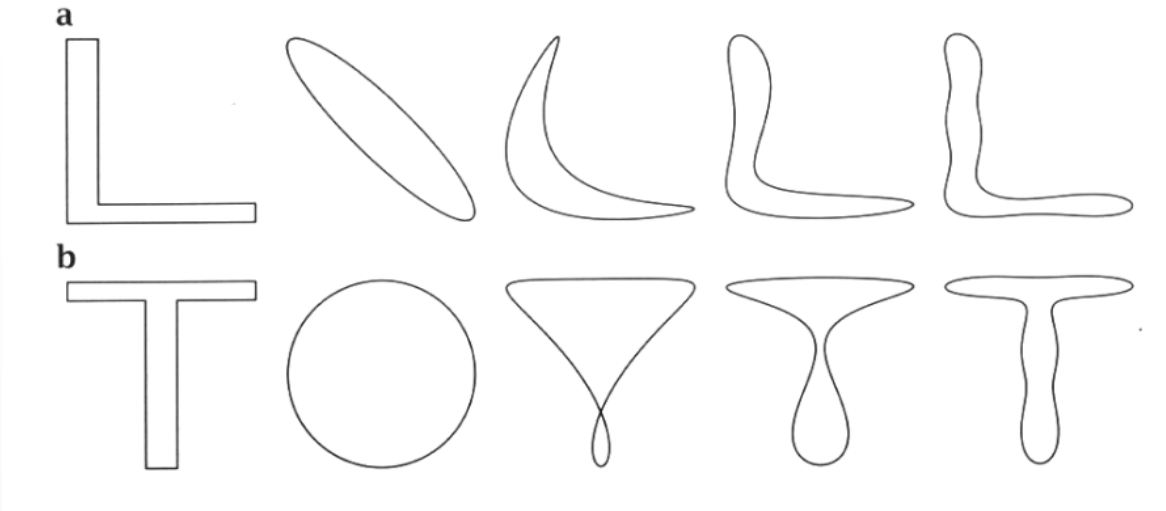


Representing curves with Fourier-descriptors

1. Complex numbers from the 2D curve points: $z_k = x_k + j \cdot y_k$
2. 1D DFT transform calculated for the complete closed curve

$$C_n = \sum_{k=0}^{K-1} z_k \exp(j \frac{2\pi}{K} n \cdot k)$$

3. Setting high frequency C_n coefficient to zero, then recovering of the approximate contour points by inverse transform



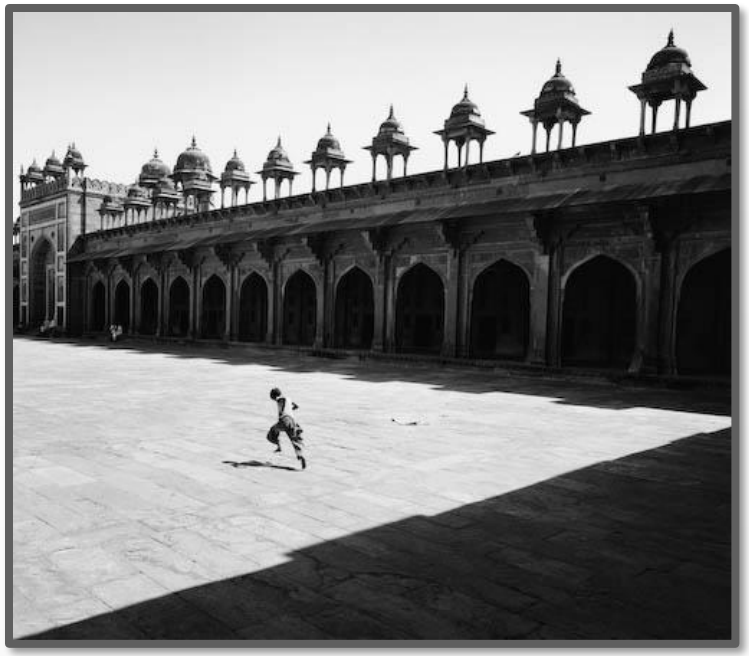
Reconstruction of letters „L” and „T” with 2, 3, 4 and 8 Fourier coefficients

Homomorphic filtering

- ***Motivation*: image** with **large dynamic range**, e.g. natural scene on brightly sunny day, recorded on a **medium** with **small dynamic range** results in image contrast significantly reduced especially in dark and bright regions

- ***Goal*:** reduce the dynamic range, increase contrast

- Example for a spatial filter also using Fourier-based steps



Homomorphic filtering

- It simultaneously ***normalizes the brightness*** across an image and ***increases contrast***.
- Assumes the following image model: the image is formed by recording the light reflected from the objects illuminated by a light source.

$$x(n_1, n_2) = i(n_1, n_2) \cdot r(n_1, n_2)$$

Illumination: slowly varying, main contributor to dynamic range

Reflectance: rapidly varying, main contributor to local contrast

- We want to reduce the illumination component, and increase the reflectance component.

Homomorphic filtering

- The main steps of homomorphic filtering:
  1. To separate the two components we first use log transformation:

$$\log\big(x(n_1,n_2)\big)=\log\big(i(n_1,n_2)\big)+\log\big(r(n_1,n_2)\big)$$

  2. Since we assume that the illumination component varies slowly and the reflectance varies rapidly, we can get the two component by using (Fourier-based) low and high pass filters:

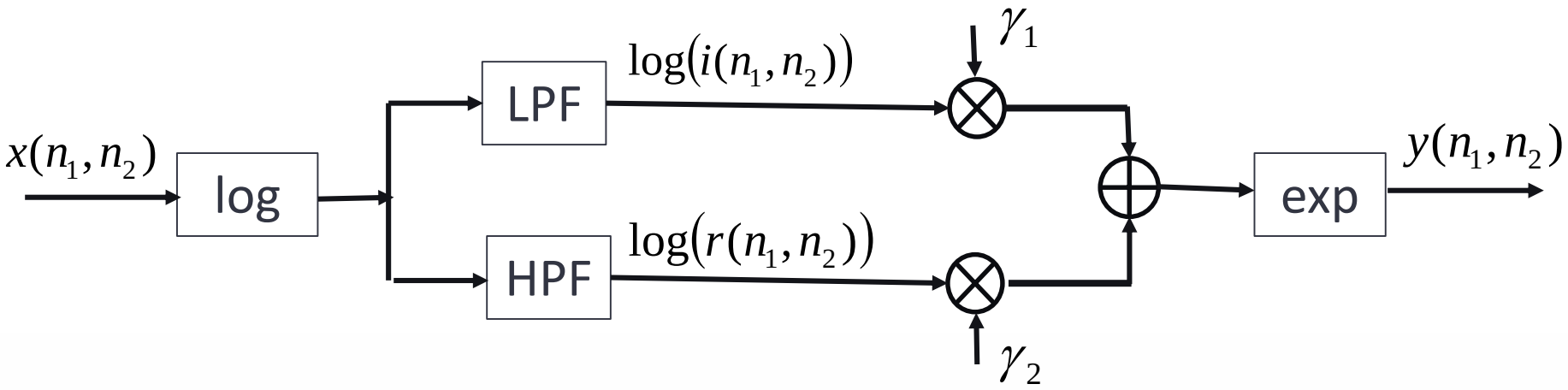
$$\log\big(i(n_1,n_2)\big)=LPF\big[\log\big(x(n_1,n_2)\big)\big]$$
$$\log\big(r(n_1,n_2)\big)=HPF\big[\log\big(x(n_1,n_2)\big)\big]$$

  3. Weight the two component:

$$\log\big(y(n_1,n_2)\big)=\gamma_1\log\big(i(n_1,n_2)\big)+\gamma_2\log\big(r(n_1,n_2)\big),\ \text{where}\ \ \gamma_1<1,\gamma_2>1$$

  4. Transform back to the original range, using the exponential transform.

Homomorphic filtering

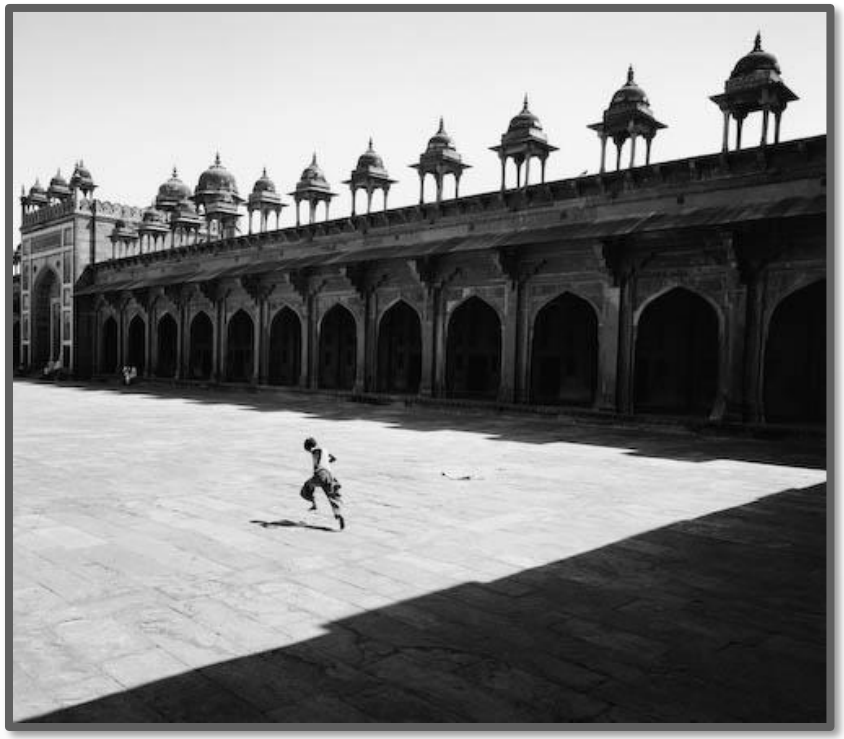


$$x(n_1, n_2) = i(n_1, n_2) r(n_1, n_2)$$

$$\log(y(n_1, n_2)) = \gamma_1 \log(i(n_1, n_2)) + \gamma_2 \log(r(n_1, n_2))$$

$$y(n_1, n_2) = [i(n_1, n_2)]^{\gamma_1} [r(n_1, n_2)]^{\gamma_2}$$

Homomorphic filtering



Original Image

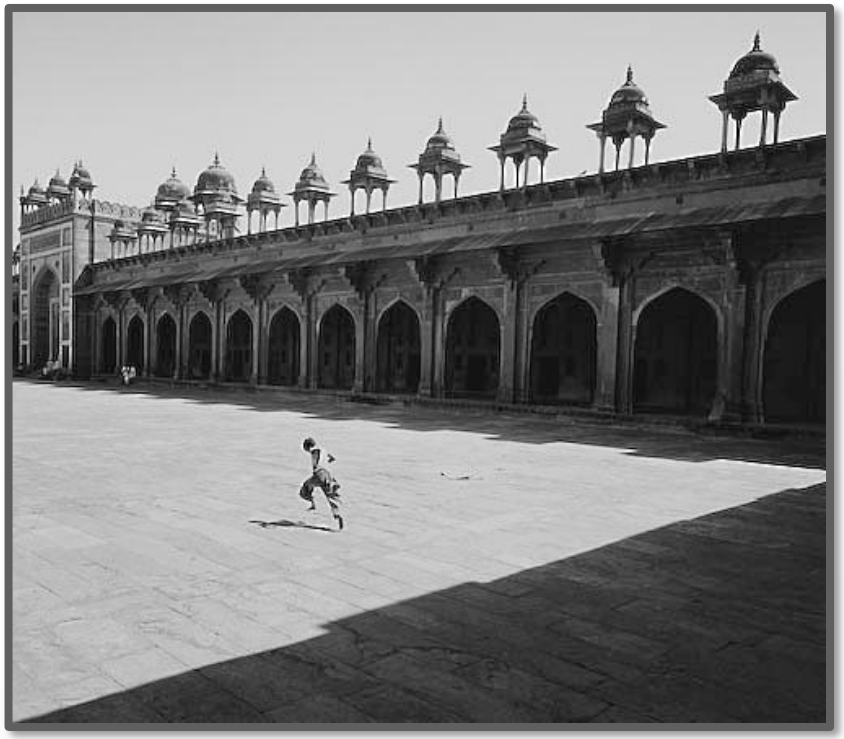


Image after homomorphic filtering

Main Sources

Fundamentals of Digital Image and Video Processing lectures by Aggelos K. Katsaggelos

Introduction to Fourier Transform (https://www.youtube.com/watch?v=1JnayXHhjlg)

Introduction to Compex Exponential Function (https://www.youtube.com/watch?v=qjT3XvS7Qno)