Computer Vision

Module 3:
Image processing

Krzysztof Krawiec
http://www.cs.put.poznan.pl/kkrawiec/

Poznan University of Technology, 2021-2023

Module outline

1. Introduction
2. Local image processing
3. Convolution
4. Nonlinear image processing
5. Morphological image processing
6. Processing in spectral domain

Image processing algorithms: introduction

Image processing: formulation

An algorithm (function) F that, given an input image f, produces an image g:

$$F : f \rightarrow g$$

Remarks:

- g does not need to have the same dimensions as f
- f and g are usually spatially coregistered, i.e. their pixels correspond spatially to each other
  - Exception: geometric transforms.
- Most image processing algorithms are covariant with respect to geometric transforms, i.e. they commute with such transforms:

$$F \circ T \equiv T \circ F$$

Image processing algorithms

Image processing

- has a longer history than computer vision,
- can be often very effectively implemented in hardware, including analog hardware,
- started already with analog photography and analog TV.

Groups of popular image processing methods:

1. Local image processing (point operations)
2. Convolution
3. Nonlinear
4. Morphological
5. Processing in spectral domain

Local image processing

Local image processing

A class of image processing algorithms in which the brightness g(x,y) of a point in the resulting image depends only on the brightness f(x,y) of a point with the same coordinates (x,y) in the source image.

$$g(x, y) = F(f(x, y))$$

Note: we mean strictly local processing (each pixel is processed independently).

Advantages:

- easy to implement,
- fast,
- can be implemented 'in place',
- easy to parallelize,

Disadvantages:

- Ignores the spatial characteristics of the image

Local image processing: implementation

Since F has a finite domain (in discrete realization), it is technically convenient to implement it as a look-up table (LUT)

$$g(x,y) = LUT[f(x,y)]$$

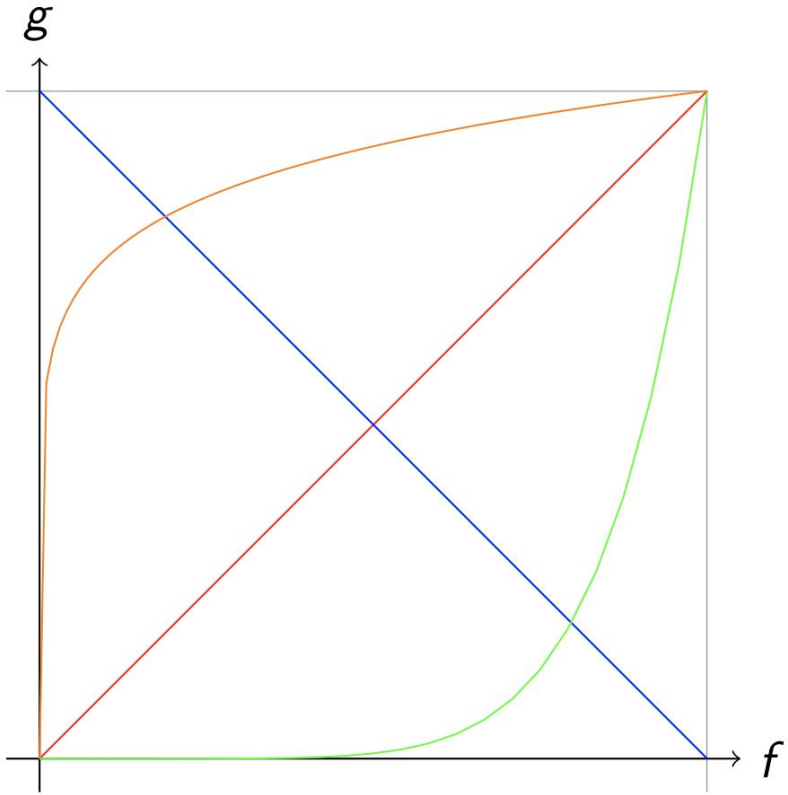
- For instance: LUT = np.array(range(256))
  Recall: every array implements a function.
- Thus: there is only one algorithm for single-point processing, parameterized by the LUT.
- If the purpose of processing is only visualization, single-point processing can be effectively implemented as manipulations on the color palette of the device (widget/graphic window, graphics card, etc.)

LUT as a function

LUT can be conveniently represented as a plot of a function mapping the input value to the output values.

When is a local processing operation reversible?

- The processing is reversible only if the LUT is an injective function.
- For example: strictly monotonic (common in practice)
- Otherwise: loss of information



Gamma correction

The purpose: to provide perceptual linearity between successive brightness levels in the image (and match it to bit resolution).

Power law formula: for brightness v in [0,1]:

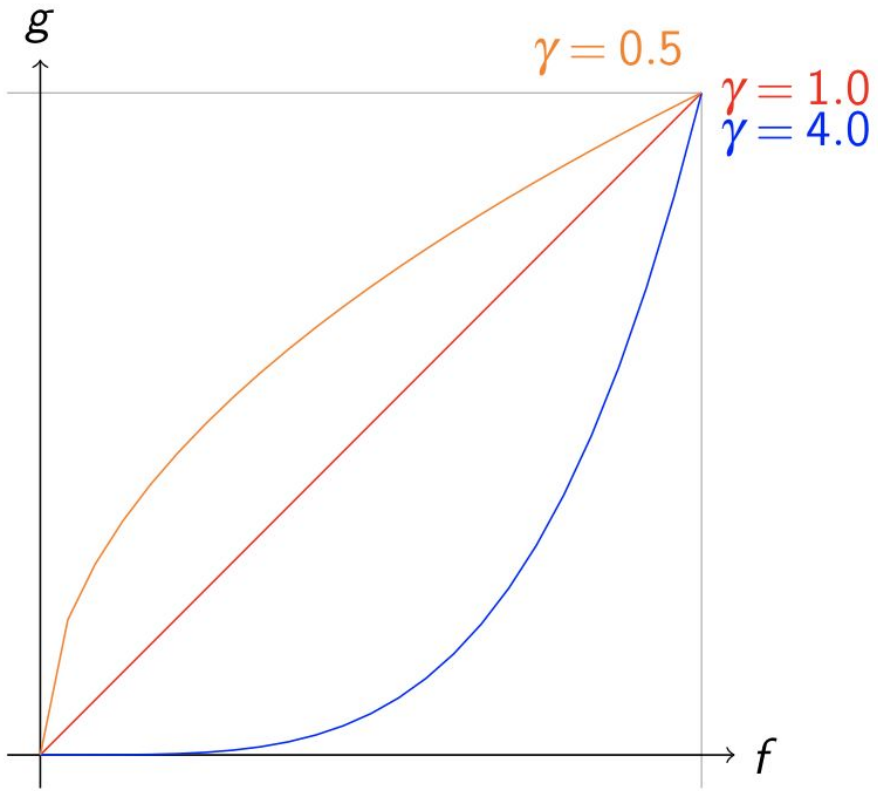
$$LUT(v) = v^{\gamma}$$

Or for images with brightness scale in [0,255]:

$$LUT(v) = 255(\frac{v}{255})^{\gamma}$$

- ɣ < 1: compression (encoding, gamma compression).
- ɣ > 1: expansion (decoding, gamma expansion)

Gamma correction



Gamma correction

- Used in projection devices.
- Necessary to compensate for nonlinear brightness perception: under normal conditions, the eye is more sensitive to differences between darker brightnesses (Steven's power law).

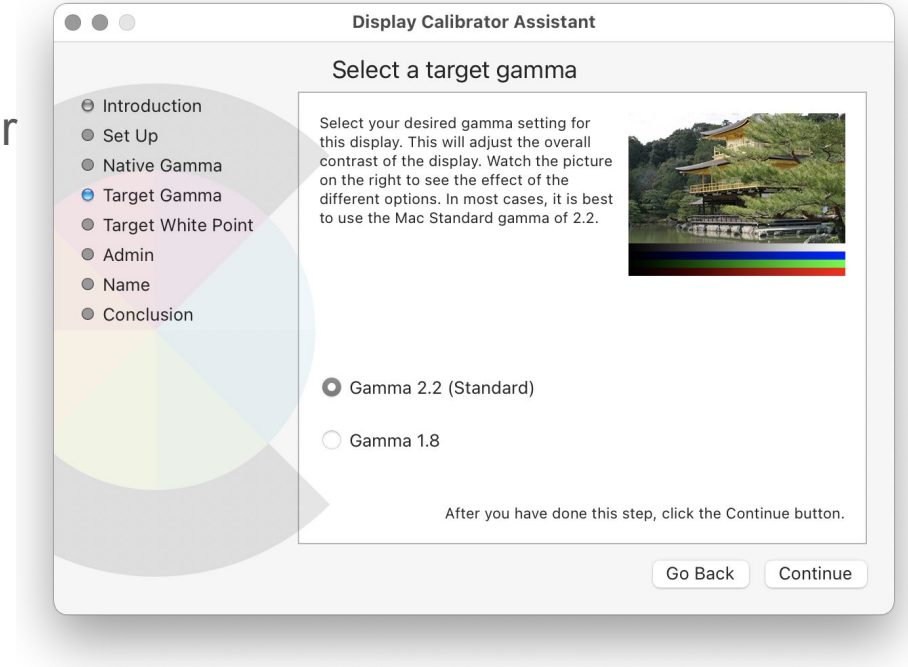


Illustration of linear and nonlinear brightness representation:

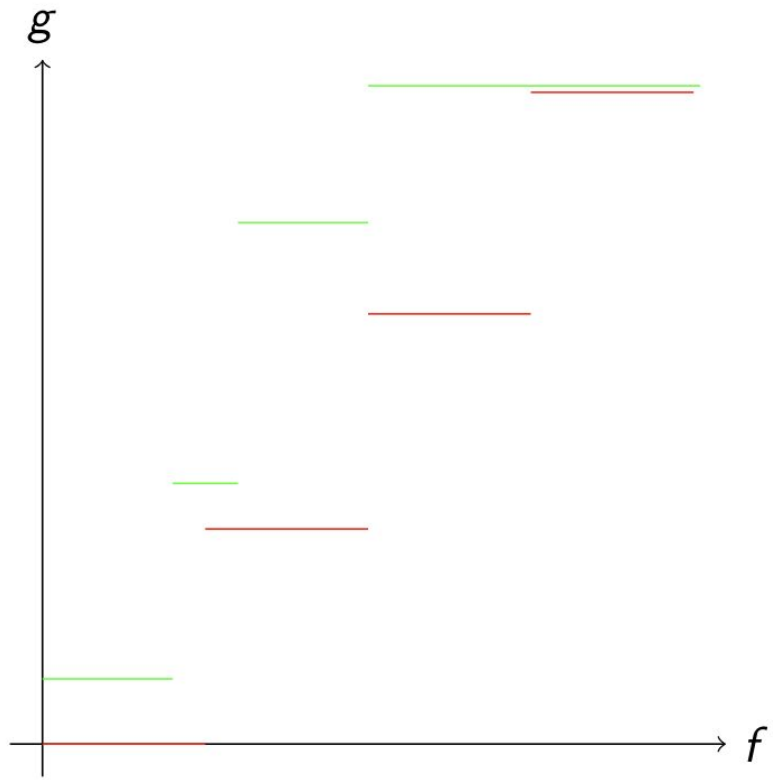
Source: https://en.wikipedia.org/wiki/Gamma_correction

Other applications of local processing

- Image Binarization
- Gray level reduction (signal quantization)

The graph on the right:
implementation of gray level reduction:

- red: uniform:
  - 4 equidistant brightness intervals mapped to 4 equidistant brightness levels)
- green: nonuniform:
  - 4 brightness intervals mapped to 4 brightness levels.



Histogram

A statistic reflecting the brightness distribution of points in an image

- An estimate of the brightness distribution of the original analog image.

Absolute histogram:

$$h_i(f) = |\{(x, y) : f(x, y) = i\}|$$

A relative histogram is a probability distribution:

$$\Pr(f(x, y) = i) = \frac{h_i(f)}{|\{(x,y)\}|}$$

Observation:

- Single point operations correspond to manipulations on the histogram.
  - A LUT can map multiple arguments to a single brightness.
- This corresponds to 'stacking' the bars of the histogram on top of each other.

Histogram stretching

Scale the brightness range to the maximum acceptable range:

$$g(x,y) = \frac{f(x,y) - \min f}{\max f - \min f}$$

- min and max calculated over the entire image
- the resulting image has a brightness scale of [0,1]
- Weaknesses:
  - susceptibility to outlier observations,
  - for large images it is highly probable that min and max are already close to the range limits anyway.

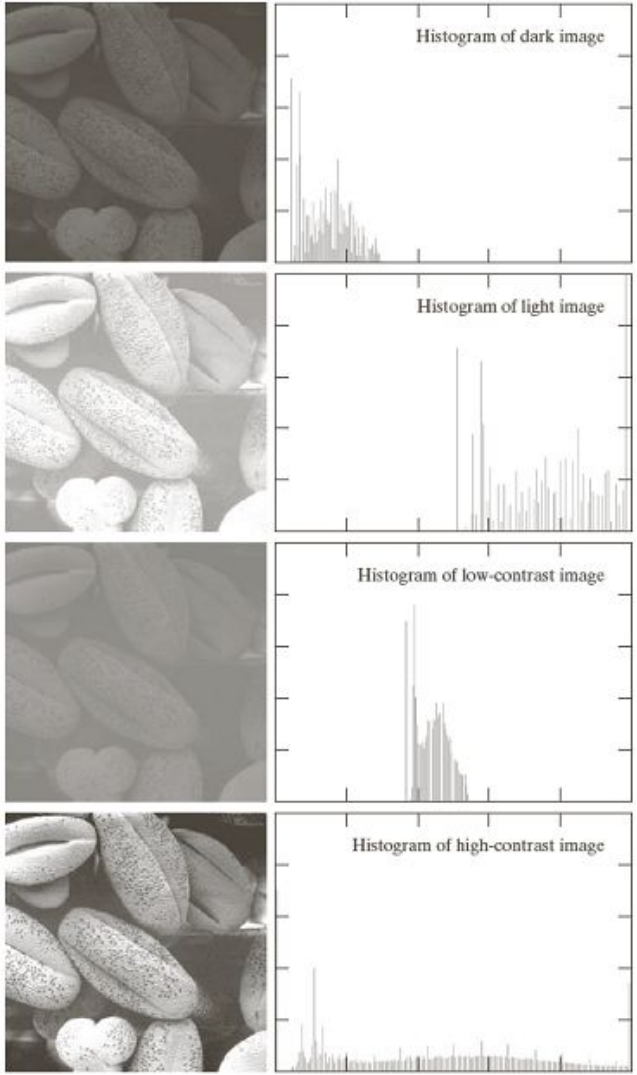
Solution:

$$g(x,y) = \frac{f(x,y) - \overline{f}}{\sigma_f}$$

- Standardization
- Note: unconstrained result and thus requires clipping (a.k.a. clamping), e.g:

$$g'(x,y) = \max(-3, \min(3, g(x,y))$$

Histogram stretching: Example



Histogram equalization

Using the integral of a histogram (the CDF, cumulative distribution function) to determine target (output) brightnesses.

The continuous case:

$$LUT_f(n) = \int_0^n h_i(f)di$$

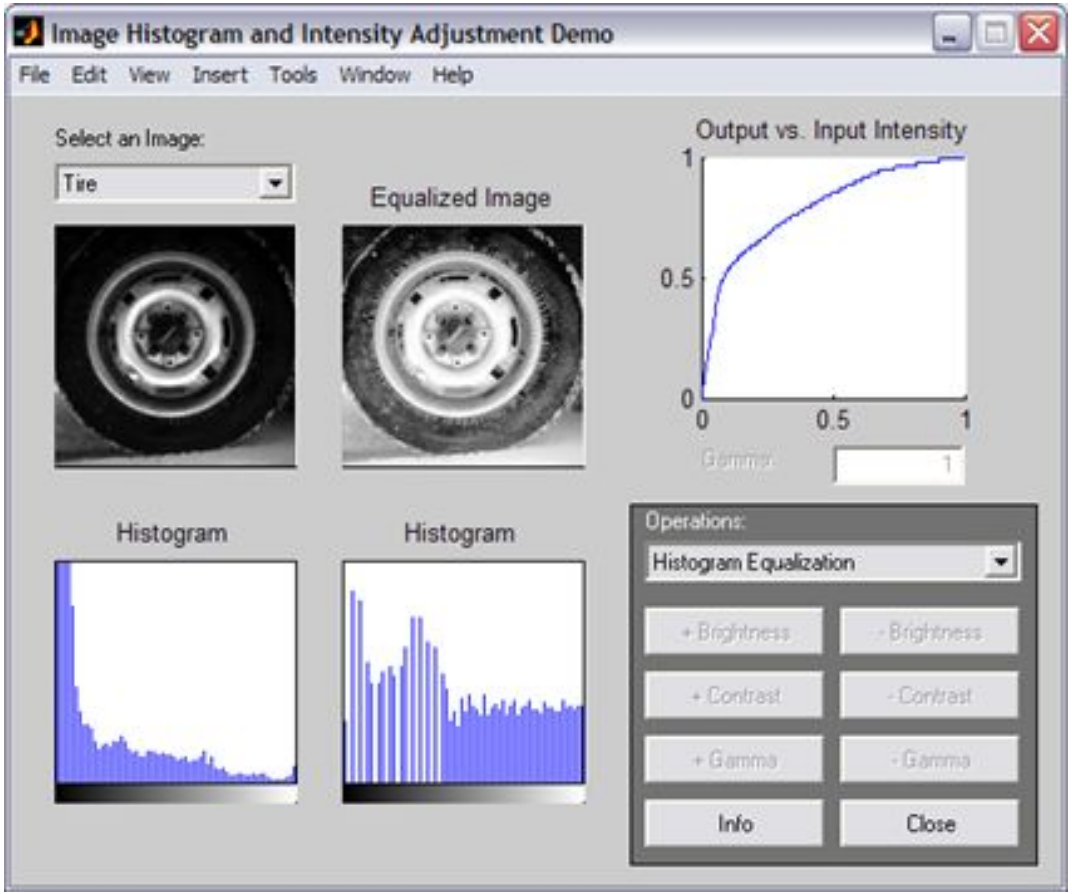
Discrete case:

$$LUT_f(n) = \sum_0^n h_i(f)$$

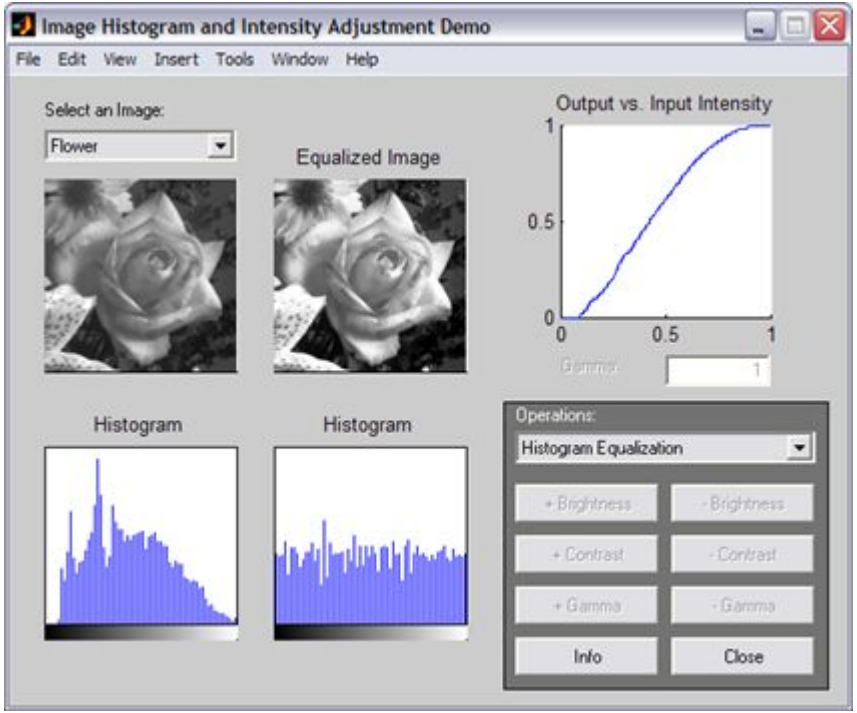
Note: LUTs depend on f; adaptive operation.

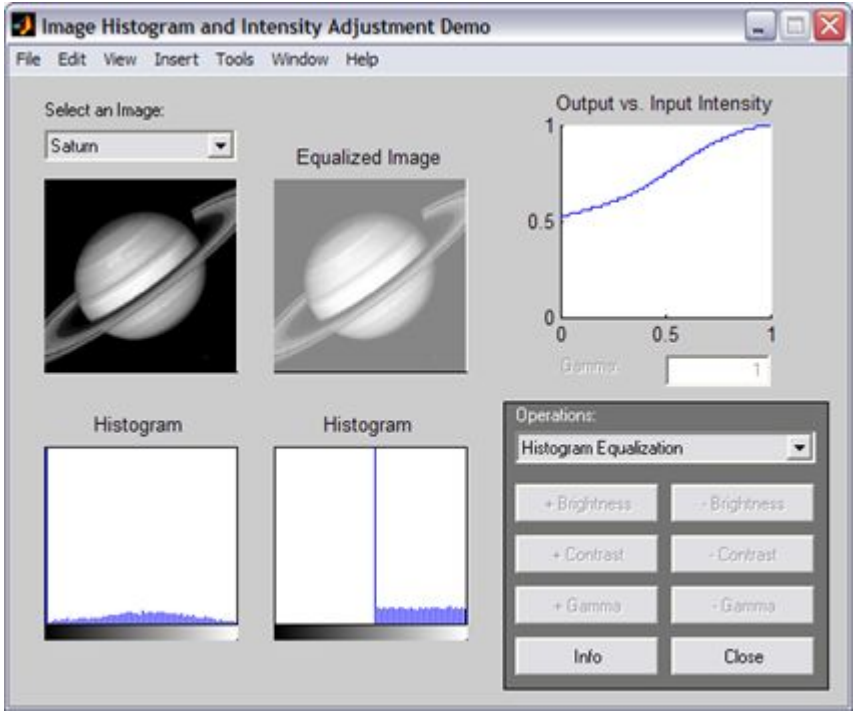
Related topic: histogram matching. Objective: to transform a histogram to a given shape/distribution (not necessarily uniform)

Histogram equalization: Examples

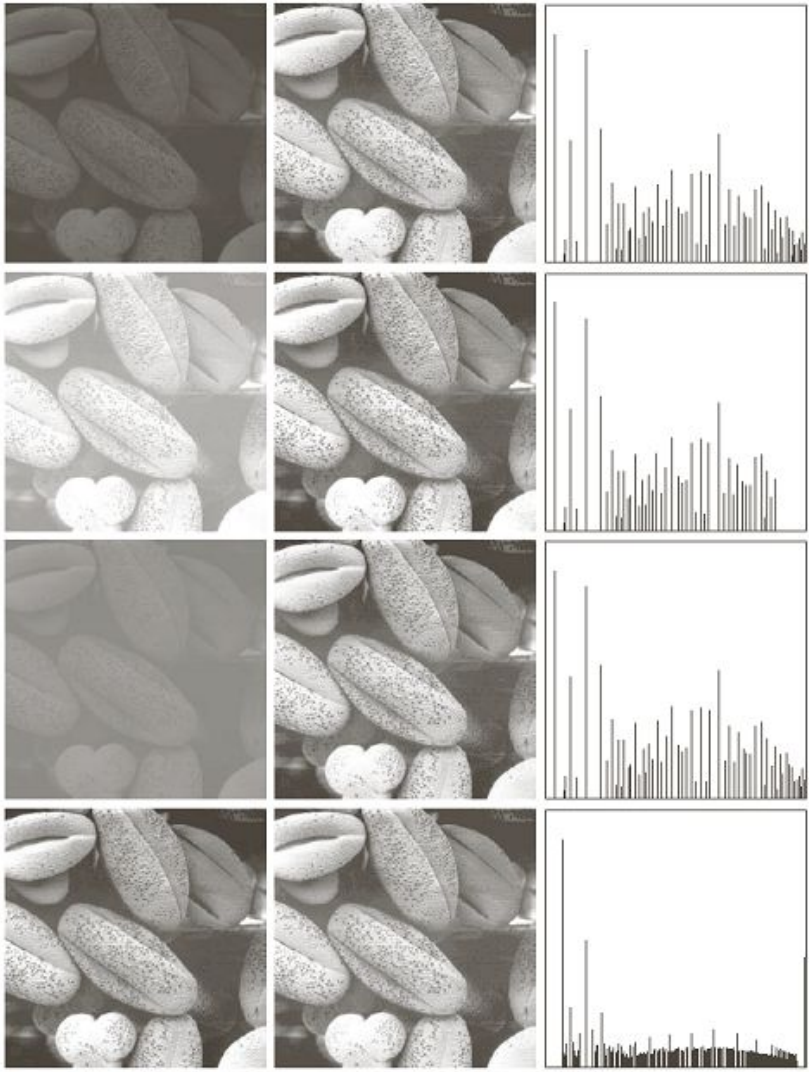


Histogram equalization: Examples





Histogram equalization: Example



Local processing of multiple images

Image arithmetic: addition, subtraction, multiplication, division, linear mixing, ...

$$g(x,y) = \alpha f_1(x,y) + (1-\alpha) f_2(x,y),\ \alpha \in (0,1)$$

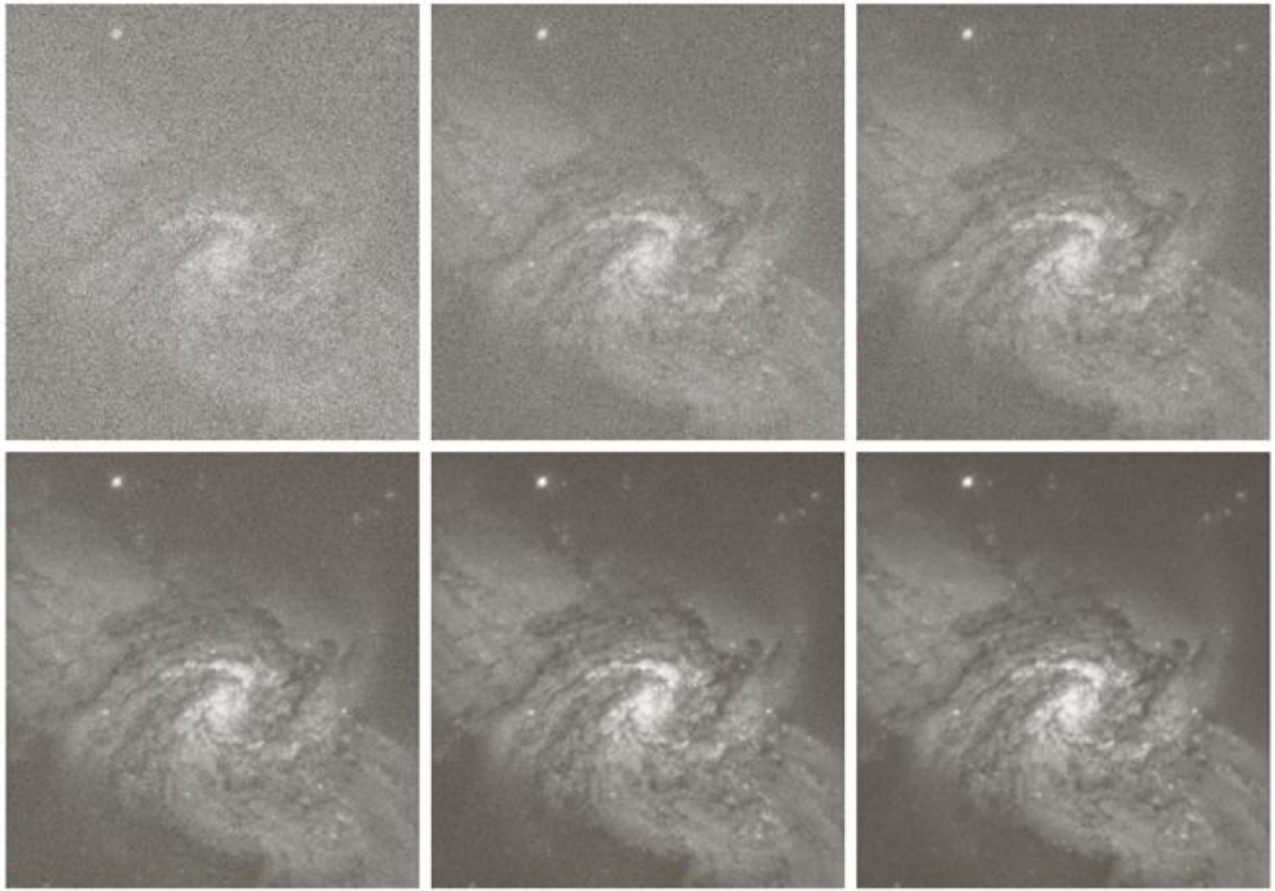
Remark:

- Watch the range of output image values.
  - Clamping may be required.
- Good spatial correlation of aggregated images is essential for high quality of output.

Applications of local image processing

- Image de-noising by averaging
  - Often used in medicine and astronomical imaging,
- Providing uniform lighting
- Noise reduction by creating device-specific frames:
  - baseline: by short exposure with shutter closed => recording instrument noise,
  - uniformity: taking a picture of an empty screen (field of view) => registration of variation in CCD structure.
  - Frames obtained in this way are subtracted from the input image (or the image recorded by the frames is divided, depending on the model).
- Logical operations on images
  - Image masking
    - realized by min/max aggregation with a binary masking image
    - image cropping
    - min/max

Image denoising by averaging



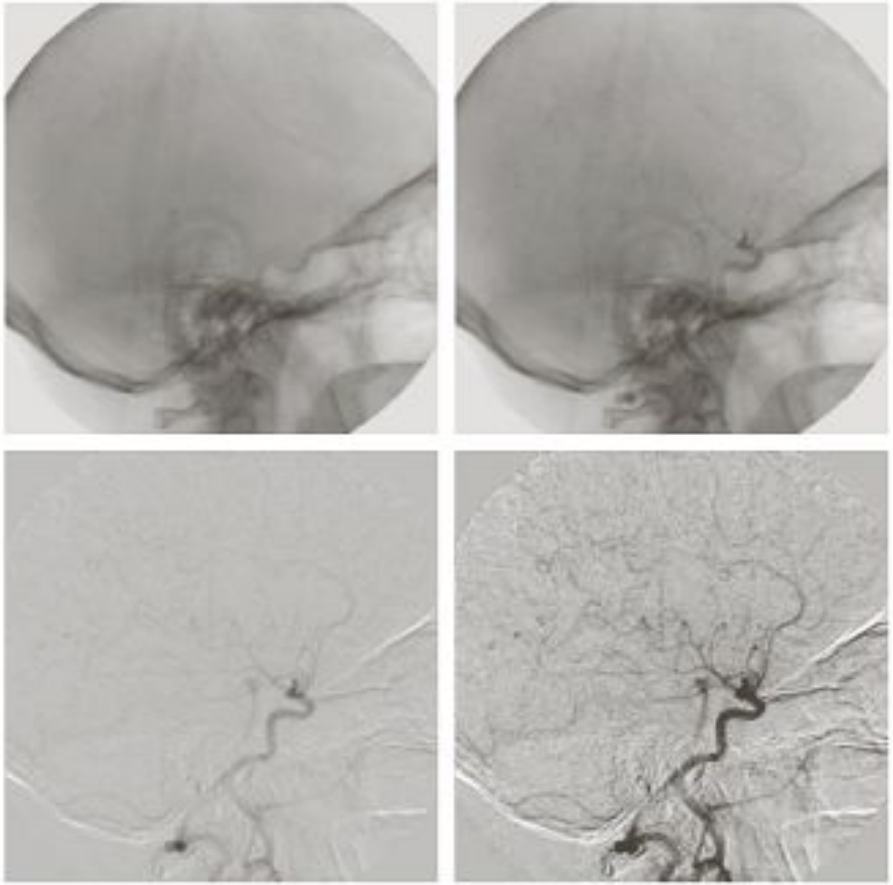
Example of ‘bias frame’

Image created by calculating the median of a series of images taken with a digital camera (Canon EOS Rebel T3i) in total darkness.

The brightness in the obtained image defines the bias of individual detectors (pixels).

The resulting image can then be e.g. subtracted from the acquired images to reduce the load.

Application of image arithmetic



Application of image arithmetic

a b c

FIGURE 2.29 Shading correction. (a) Shaded SEM image of a tungsten filament and support, magnified approximately 130 times. (b) The shading pattern. (c) Product of (a) by the reciprocal of (b). (Original image courtesy of Mr. Michael Shaffer, Department of Geological Sciences, University of Oregon, Eugene.)

Geometric transforms

Geometric transforms

In practice: mostly affine transformations. Basic cases:
Translation (T), scaling (S), rotation (R), sheer

1		tx
	1	ty
		1

sx		
	sy	
		1

cos	sin	
-sin	cos	
		1

1	sh	
sv	1	
		1

Usage:

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = M \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- (x,y) - coordinates of a pixel in the source image
- (x',y') - coordinates of the pixel in the output/target image

Geometric transforms

Essential properties:

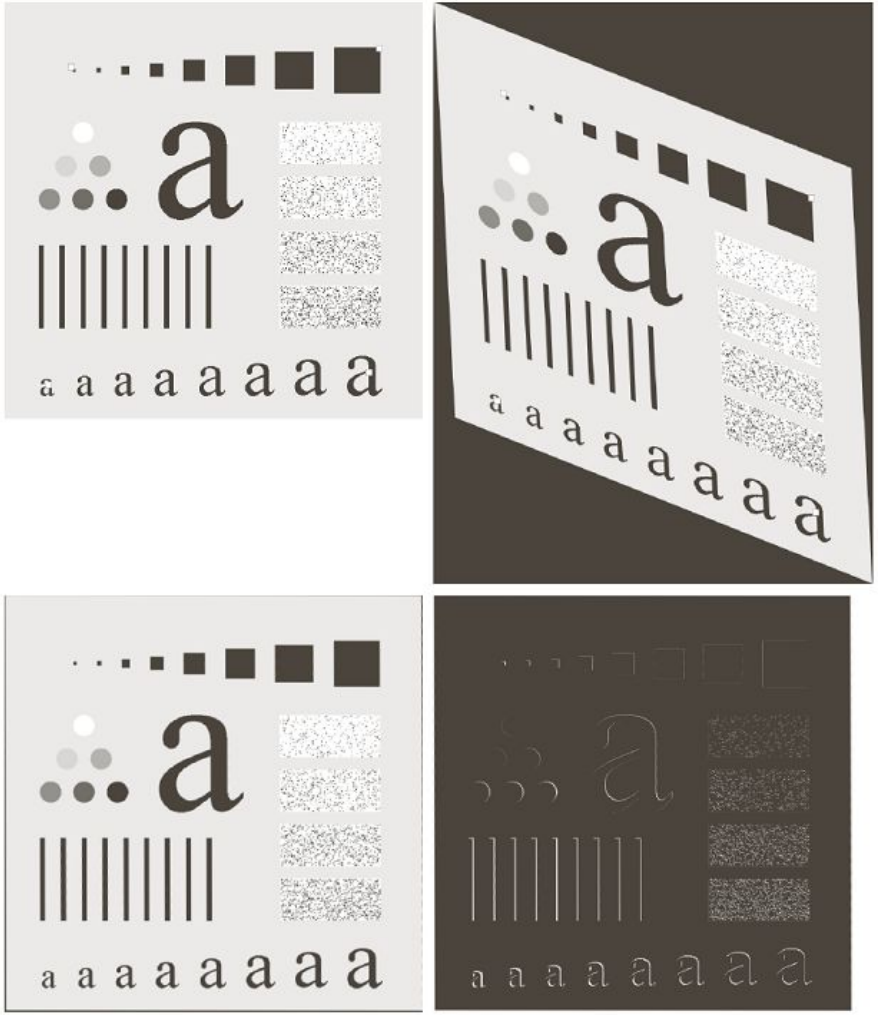
- An inverse matrix implements an inverse transformation
- Composing operations by matrix multiplication, e.g.

$$M_{rot+scale} = M_{rot} M_{scale}$$

Remarks:

- Need to allocate a larger image ('canvas') for the target
- Resulting coordinates (x',y') are in general continuous (non-integer), thus require interpolation. Known interpolation methods:
  - nearest neighbor,
  - linear (linear, bi-linear),
  - cubic (bi-cubic),
  - area,
  - Lanczos method.

Artefacts of interpolation



Multi-channel local operations

Operations on multiple channels

Usage scenario: multi-channel image input or output

Examples:

- decomposing a multichannel image into single-channel images
  - known as channel split, or slicing,
- conversion of a multichannel image to a single-channel image
  - e.g. from RGB to grayscale (e.g. luminance, brightness)
- pseudocoloring: mapping of a single-channel scale to the space of the selected color representation (e.g. HSV).

Channel split

Conversion from RGB to grayscale

The correct calculation of luminance (brightness) requires a weighted sum reflecting the relative sensitivity of the human eye to RGB components.

$$g(x,y) = 0.30 f_r(x,y) + 0.59 f_g(x,y) + 0.11 f_b(x,y)$$

Pseudocoloring

The goal: Improved perceptual differentiation

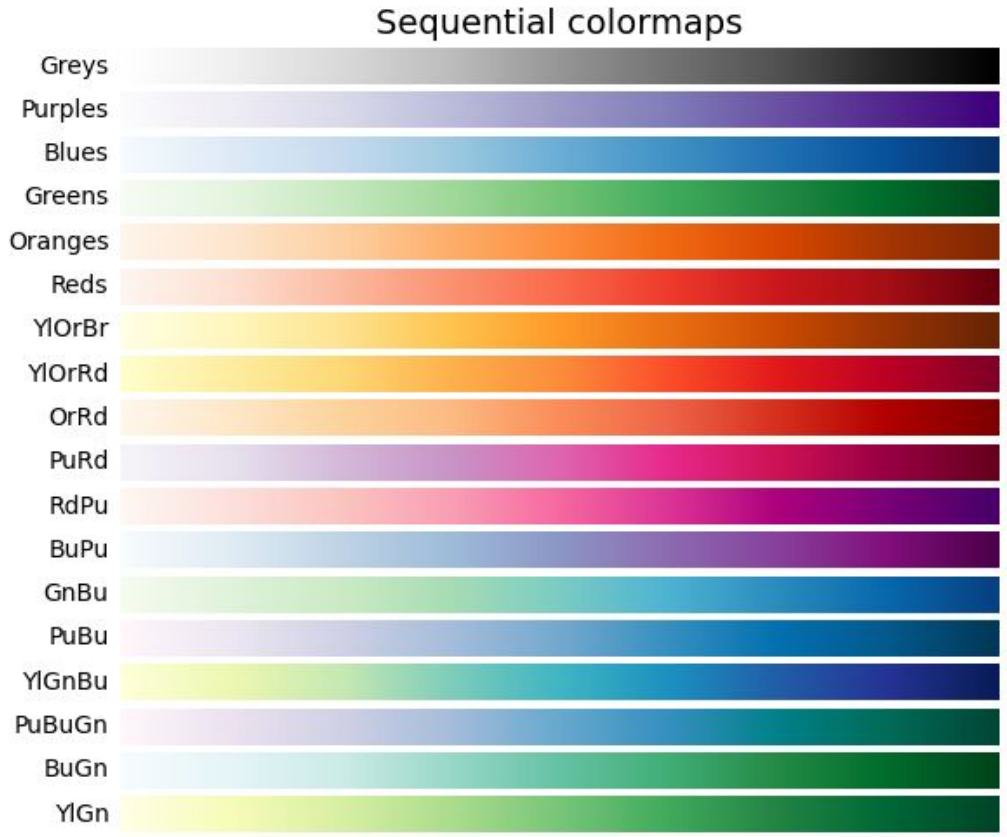
- The human eye can distinguish only about 100 degrees of gray
- The number of distinguishable colors is even on the order of thousands.

Types of color maps:

- sequential
- divergent
  - two dominant colors correspond to the extremes, the middle value of the scale is neutral
- cyclic
  - convenient for quantities defined on cyclic/modulo scales, e.g. angle
- qualitative
  - unordered: assume that the input variable (for input pixels) is nominal.

Examples from the Matplotlib library

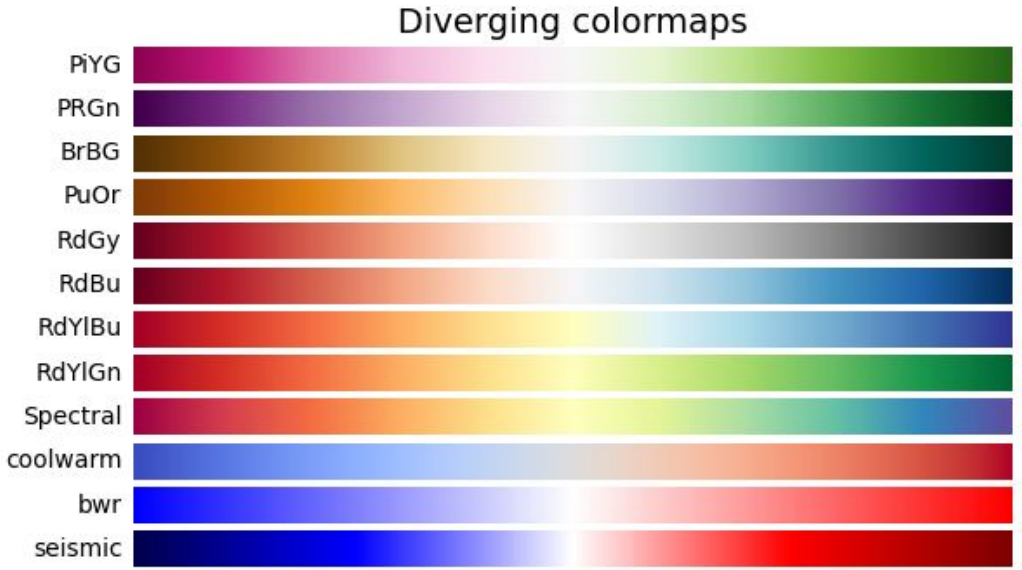
Sequential



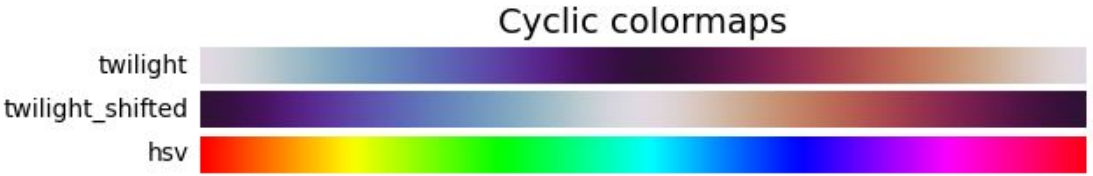
Source: https://matplotlib.org/stable/tutorials/colors/colormaps.html

Examples from the Matplotlib library

Divergent



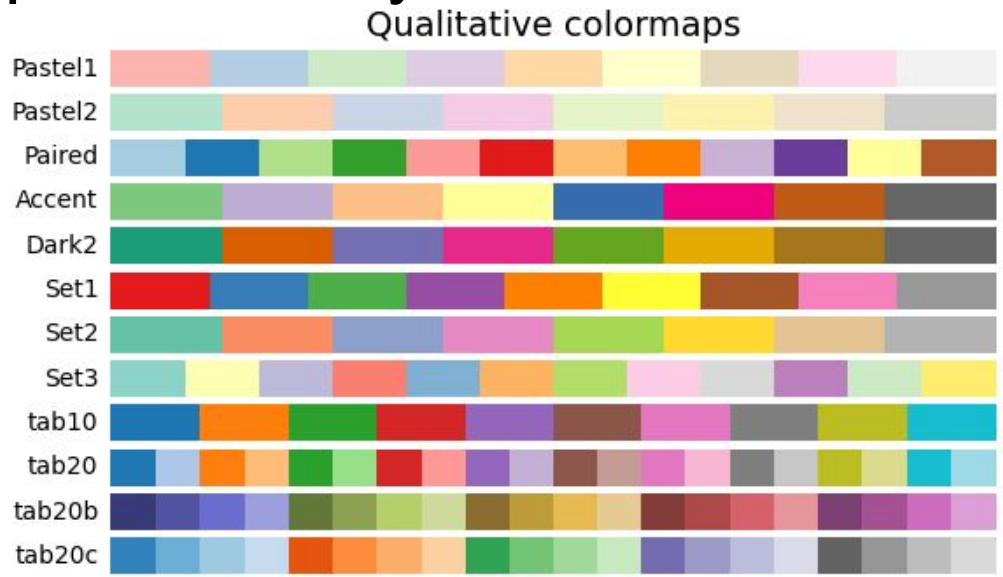
Cyclic



Source: https://matplotlib.org/stable/tutorials/colors/colormaps.html

Examples from the Matplotlib library

Qualitative



Źródło: https://matplotlib.org/stable/tutorials/colors/colormaps.html

Pseudocoloring

Notes:

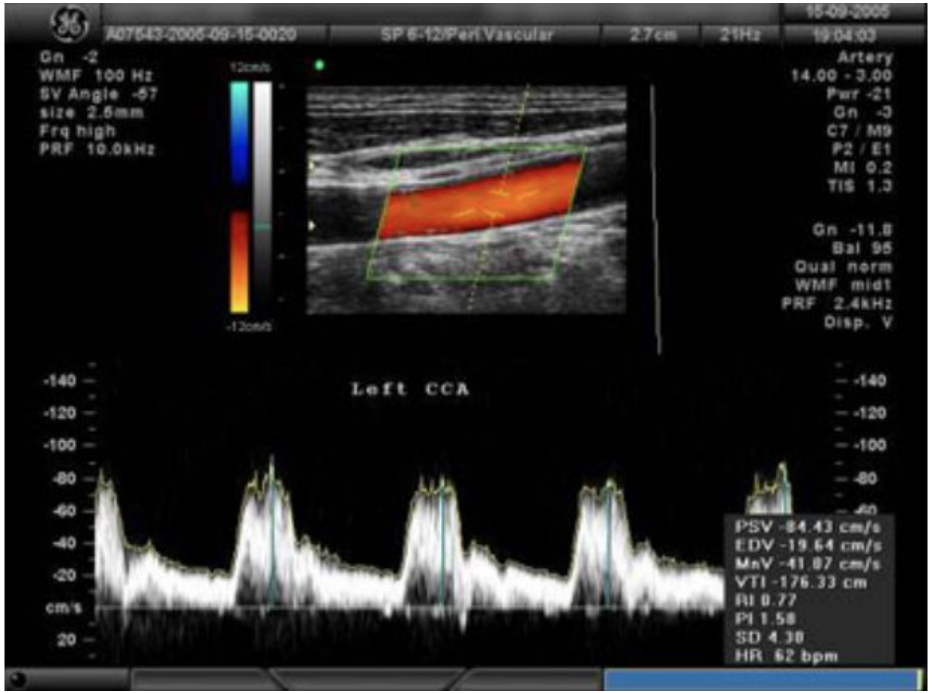
- Some formerly proposed color maps are now going out of use
  - E.g. because they can be misleading when presented in black and white
  - Example: the *jet* colormap from Matplotlib

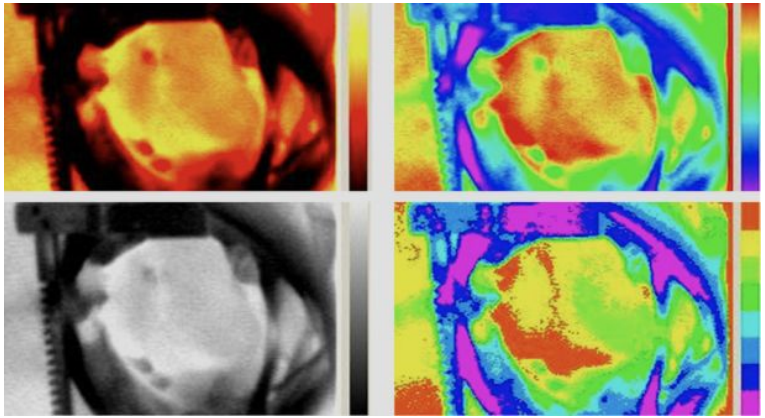
Other scales used in industrial applications:

- RAINBOW - color scale with green inversion,
- SPECTRUM - color scale corresponding to the colors of the visible spectrum,
- BLACKBODY - color scale emphasizing yellow and red,
- IRON - color scale emphasizing red.

Pseudocoloring: applications

Example of using pseudocoloring in: ultrasound (visualization of blood flow velocity) and thermography (intraoperative cardiac image).

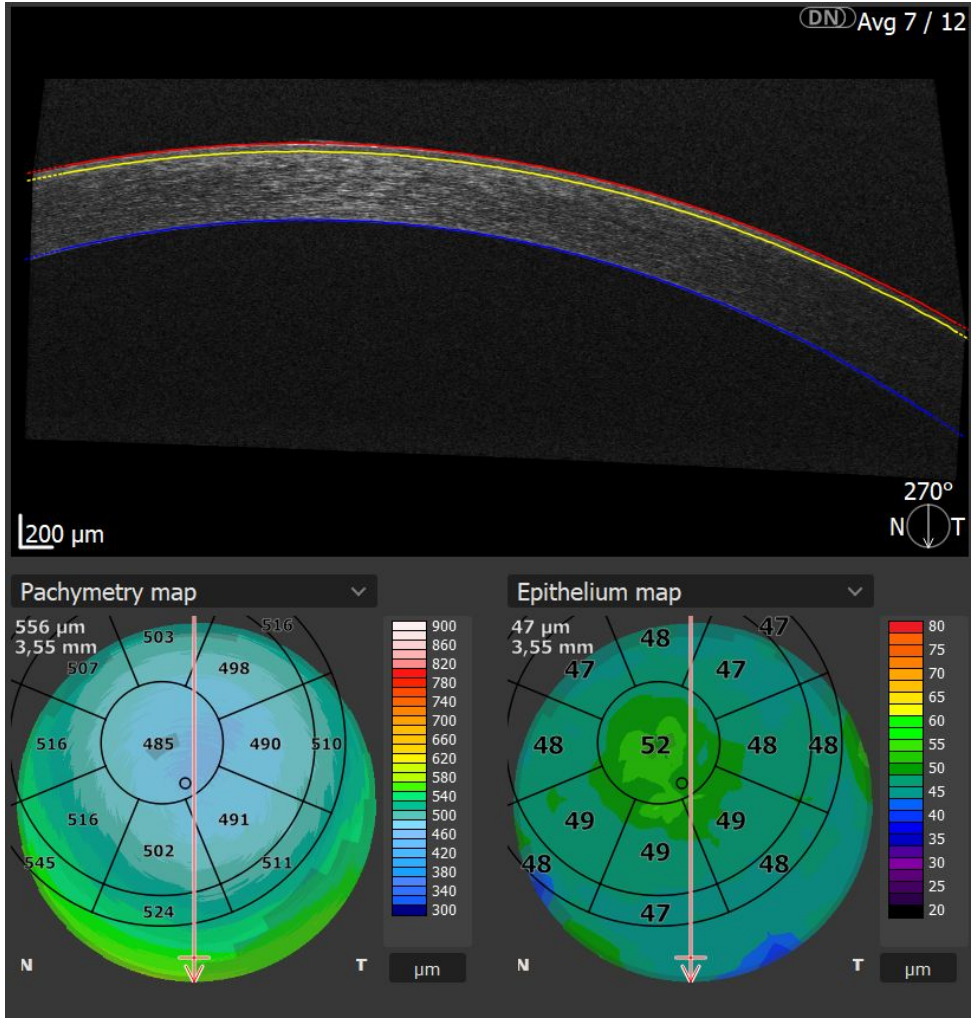




Pseudocoloring: applications

Pseudocoloring of thickness of anatomical structures in human cornea, in optical coherence tomography (OCT) imaging.

Note: the property being visualized here is of anatomical relevance.



Histograms of multi-channel images

Basic option: histograms for individual components

- I.e., marginal probability distribution

Joint histogram = joint probability distribution, e.g.

$$h_i([r,g,b]) = \Pr(f(x,y) = [r,g,b])$$

Notes:

- In general more sparse than a traditional histogram.
- Consequently, it may be less representative.

Right: joint histogram (distribution) of the components R and G of an example image

Convolutional filtering

Convolution

Alternative name: linear filtering.

Discrete one-dimensional convolution of a single-argument function *f*:

$$g(x) = (f * M)(x) = \sum_{h=-\left\lfloor \frac{m}{2} \right\rfloor}^{\left\lfloor \frac{m}{2} \right\rfloor} M(h) f(x-h)$$

- * - the convolution operation
- M - mask, kernel (vector of arbitrary real-valued coefficients)
  - It is the basic ‘parameter’ of the operation (and the only one, actually).
- m - mask size (small odd number > 1),
  - Mask elements indexed with signed indexes.

Interpretation: convolution replaces a function value at a point with a weighted sum of values.

- Note: the sum iterates over M and f in opposite directions.
- Usually neighboring values, because M is usually small.

Example

```
[36] i1 = [[3,1,3,0, 0,3,1,2, 1,3,2,1]]
     show_table(i1)
```

3	1	3	0	0	3	1	2	1	3	2	1

```
     m = [[1,1,1]]
     show_table(m)
```

1	1	1

```
[38] r = np.convolve(i1[0],m[0])
     show_table([r.transpose()])
```

3	4	7	4	3	3	4	6	4	6	6	6	3	1

2D convolution

$$g(x,y) = (f * M)(x,y) = \sum_{k=-\left\lfloor \frac{m}{2} \right\rfloor}^{\left\lfloor \frac{m}{2} \right\rfloor} \sum_{h=-\left\lfloor \frac{m}{2} \right\rfloor}^{\left\lfloor \frac{m}{2} \right\rfloor} M(h,k) f(x-h, y-k)$$

- M – matrix of coefficients (the mask)

Practical implications:

- Significantly higher computational cost than 1D convolution (although this can be avoided in some special cases).
- Convolution can be easily generalized to an arbitrary number of dimensions.

Convolution: remarks

Linear operation:

- quick to implement,
- fast,
- easy to parallelize,
- however: limited capabilities (due to linear nature)
    - paradoxically, in a sense, less versatile than single-point processing
    - on the other hand: takes neighborhood into account.

In technical implementation:

- need to handle boundary cases/effects
    - e.g. in Numpy: mode {'full', 'valid', 'same'},

Example: *full* mode vs. *same* mode

```
[50] i1 = [[3,1,3,0, 0,3,1,2, 1,3,2,1]]
     show_table(i1)
```

3	1	3	0	0	3	1	2	1	3	2	1

```
[51] m = [[1,1,1]]
     show_table(m)
```

1	1	1

```
[54] r = np.convolve(i1[0],m[0], mode='full')
     show_table([r.transpose()])
```

3	4	7	4	3	3	4	6	4	6	6	6	3	1

```
[55] r = np.convolve(i1[0],m[0], mode='same')
     show_table([r.transpose()])
```

4	7	4	3	3	4	6	4	6	6	6	3

Example: *full* mode vs. *valid* mode

```
[50] i1 = [[3,1,3,0, 0,3,1,2, 1,3,2,1]]
     show_table(i1)
```

3	1	3	0	0	3	1	2	1	3	2	1

```
[51] m = [[1,1,1]]
     show_table(m)
```

1	1	1

```
[54] r = np.convolve(i1[0],m[0], mode='full')
     show_table([r.transpose()])
```

3	4	7	4	3	3	4	6	4	6	6	6	3	1

```
     r = np.convolve(i1[0],m[0], mode='valid')
     show_table([r.transpose()])
```

7	4	3	3	4	6	4	6	6	6

Example

```
i1 = [[3,1,3,0, 0,3,1,2, 1,3,2,1]]
show_table(i1)
```

3	1	3	0	0	3	1	2	1	3	2	1

```
[57] m = [[1,2,1]]
     show_table(m)
```

1	2	1

```
[60] r = np.convolve(i1[0],m[0], mode='full')
     show_table([r.transpose()])
```

3	7	8	7	3	3	7	7	6	7	9	8	4	1

```
[59] r = np.convolve(i1[0],m[0], mode='valid')
     show_table([r.transpose()])
```

8	7	3	3	7	7	6	7	9	8

Border effects (edge cases)

The ‘full’ mode assumes that the ‘virtual’ pixels beyond the image raster have zero values.

```
[50] i1 = [[3,1,3,0, 0,3,1,2, 1,3,2,1]]
     show_table(i1)
```

3	1	3	0	0	3	1	2	1	3	2	1

```
[57] m = [[1,2,1]]
     show_table(m)
```

1	2	1

```
[60] r = np.convolve(i1[0],m[0], mode='full')
     show_table([r.transpose()])
```

3	7	8	7	3	3	7	7	6	7	9	8	4	1

```
[61] i1 = [[0,0,3,1,3,0, 0,3,1,2, 1,3,2,1,0,0]]
     show_table(i1)
```

0	0	3	1	3	0	0	3	1	2	1	3	2	1	0	0

```
[62] m = [[1,2,1]]
     show_table(m)
```

1	2	1

```
[63] r = np.convolve(i1[0],m[0], mode='valid')
     show_table([r.transpose()])
```

3	7	8	7	3	3	7	7	6	7	9	8	4	1

Border effects

Is the full mode always desirable?

- Not necessarily: the ‘virtual’ pixels may lead to border artefacts.

Other options:

- Fill with values other than 0 (e.g., the mean/median of the image).
- Duplicate edge pixels
- Mirror a strip of border pixels.

See, e.g., cv2.copyMakeBorder()

- Border effects are very important in practice, especially where masks are small, which is very common.
- Practical implication: it is desirable for the edge elements of a mask to be close to 0.

Relevance of convolution

Why is convolution filtering given so much importance in image analysis?

The convolution theorem

Fourier transform of convolution *g*h* is equivalent to the elementwise (Hadamard) product of the transform G of g (image) and the transform H of h (the mask).

$$r(x) = \{g * h\}(x) = \mathcal{F}^{-1}\{G \cdot H\}$$

- Thus: the mask with appropriately chosen characteristics can attenuate (or completely suppress) selected (spatial) frequencies in the image.
- Practical implication: under certain conditions, convolution can be realized more efficiently in the frequency domain.

Image smoothing with convolution

Averaging filter

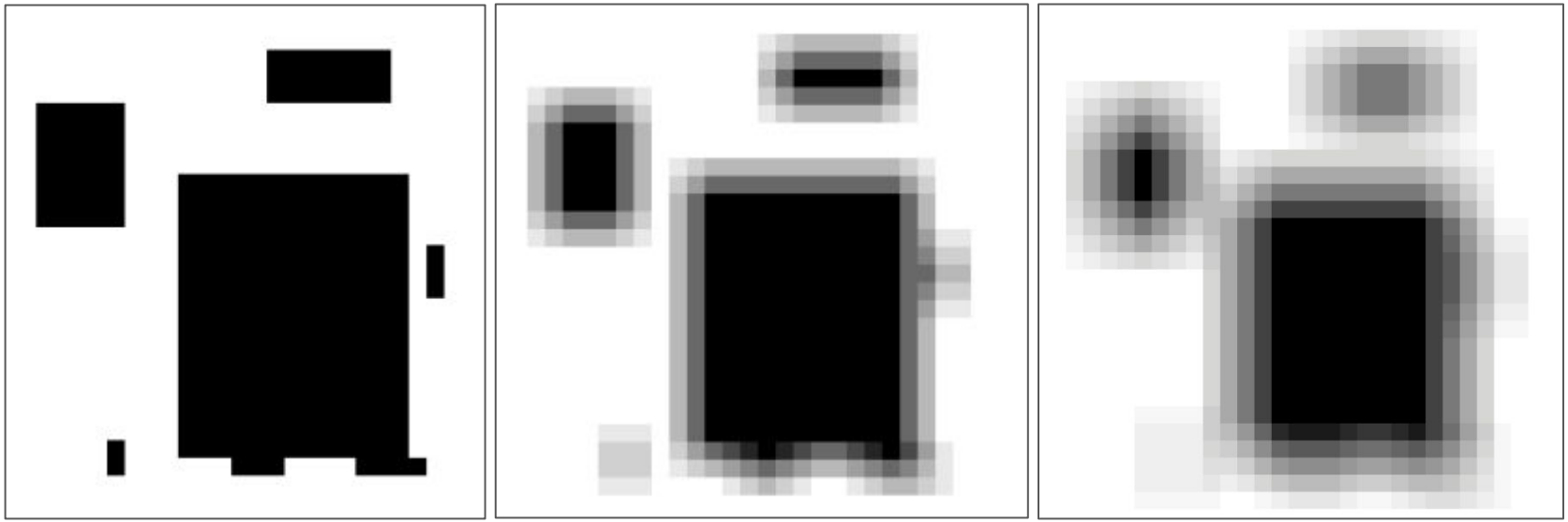
1	1	1
1	1	1
1	1	1

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

- Not to be confused with arithmetic averaging of *multiple* images.
- When the sum of the mask elements is 1, convolution preserves the ‘mass’ of the signal.
- Averaging compensates local brightness fluctuations.
- Averaging m^2 of noisy pixel values reduces the noise standard deviation m times.
- Many other convolution filters have averaging properties.

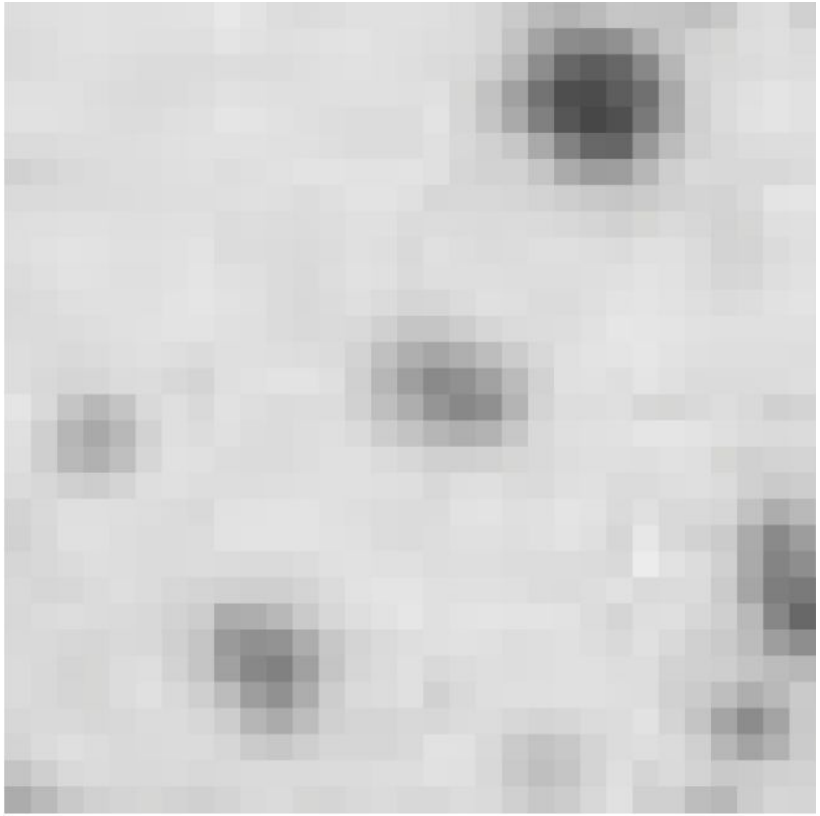
Example

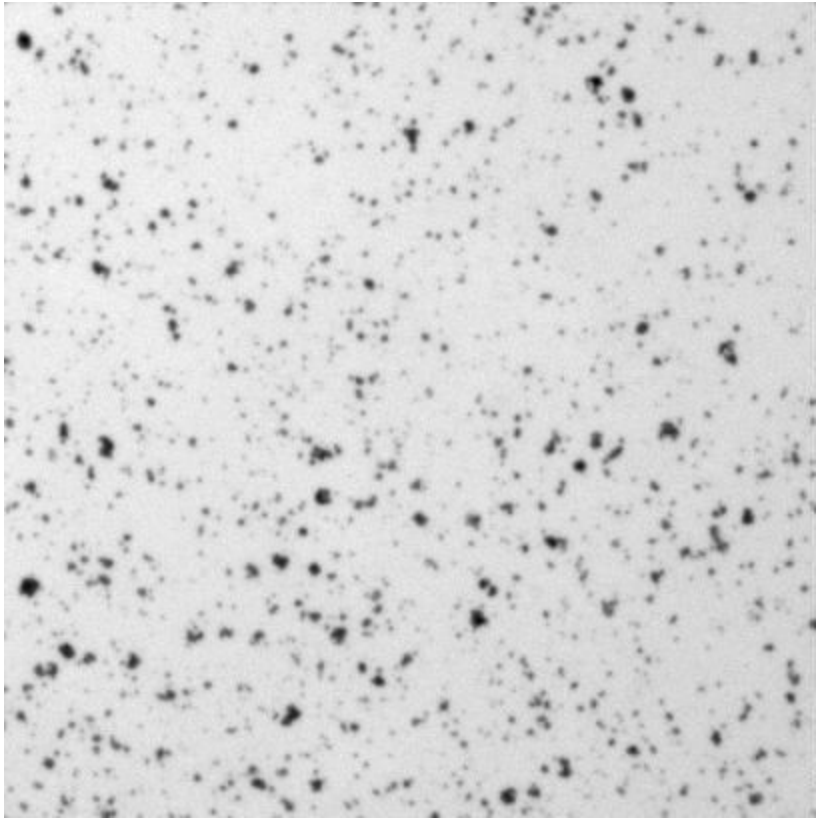
An 32x32 image convolved with averaging 3x3 and 5x5 masks



Example: Input image

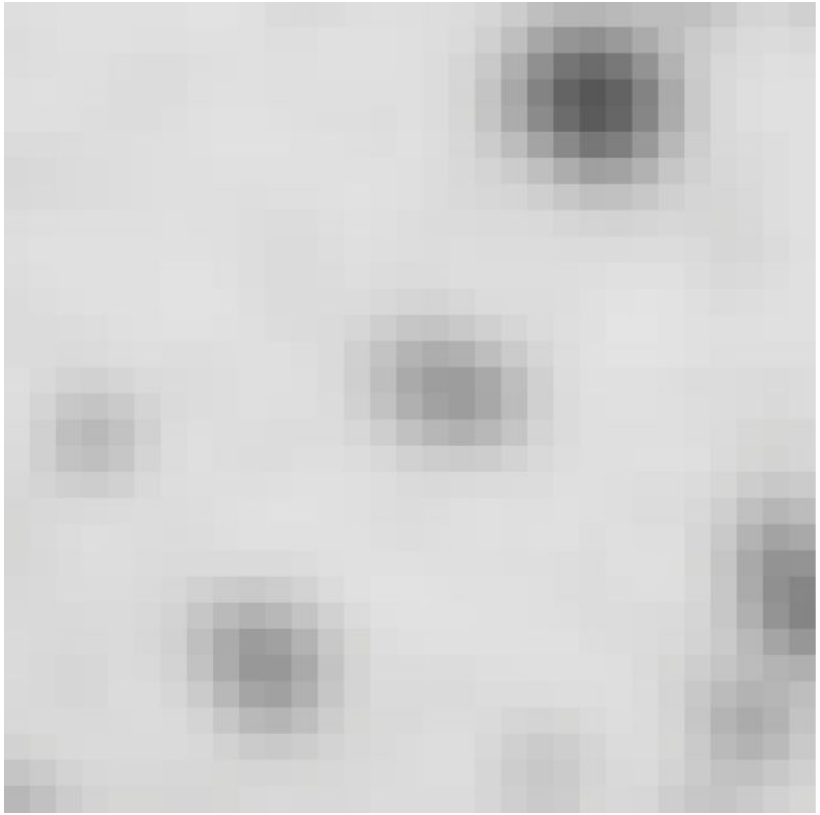
(a cell culture in microscopic imaging)

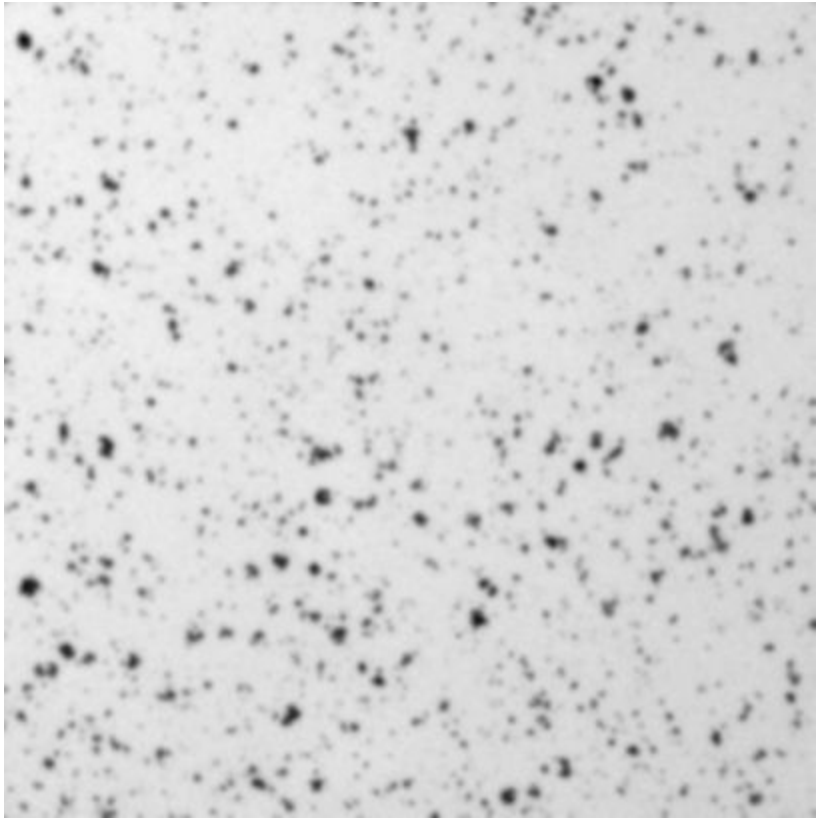




Smoothed image

(convolved with an 5x5 averaging mask)





Gaussian filtering

Convolution with a mask with values determined by the normal distribution:

$$M(h,k) = \exp(-\frac{h^2+k^2}{2\sigma^2})$$

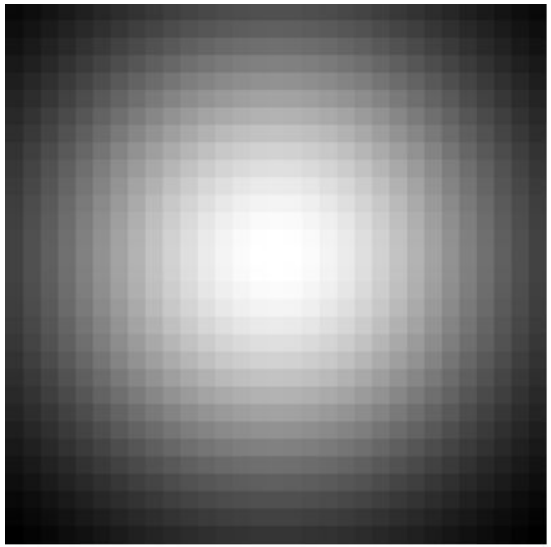
where h,k iterate over the mask elements (with sign, i.e., e.g., h, k={-1,0,1}).

Properties:

- Circular, center-symmetric characteristics: isotropic filter.
- Selection of filter variance to size:
  - Suggested relationship m=5σ, gives an area under the curve of 0.9876
- Suggested use of m ≥ 7; may cause significant distortion for smaller m.
- Better frequency response than the averaging filter: Fourier transform of the mask does not include higher frequency components

Example

The mask of 32x32 Gaussian filter, with σ =10.



Notice: Masks of convolutional filters can be rendered as monochrome raster images.

Separability of the Gaussian filter

Two-dimensional Gaussian filtering can be realized with two consecutive one-dimensional Gaussian filterings (vertical and horizontal)

$$
\begin{aligned}
g(x,y) &= \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} M(h,k) f(x \\
&- h, y-k) \\
&= \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \exp\left(-\frac{h^2+k^2}{2\sigma^2}\right) f(x-h, \\
&y-k) \\
&= \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \exp\left(-\frac{h^2}{2\sigma^2}\right) \exp \\
&\left(-\frac{k^2}{2\sigma^2}\right) f(x-h, y-k) \\
&= \sum_{k=-\frac{m}{2}}^{\frac{m}{2}} \exp\left(-\frac{k^2}{2\sigma^2}\right) \sum_{h=-\frac{m}{2}}^{\frac{m}{2}} \exp \\
&\left(-\frac{h^2}{2\sigma^2}\right) f(x-h, y-k)
\end{aligned}
$$

Separability of the Gaussian filter

Filtering with this integer-valued approximation of the 5x5 Gaussian filter:

$$\frac{1}{159} \times \begin{array}{|c|c|c|c|c|} \hline 2 & 4 & 5 & 4 & 2 \\ \hline 4 & 9 & 12 & 9 & 4 \\ \hline 5 & 12 & 15 & 12 & 5 \\ \hline 4 & 9 & 12 & 9 & 4 \\ \hline 2 & 4 & 5 & 4 & 2 \\ \hline \end{array}$$

is equivalent to successive filtering with the following two 1D masks:

$$\exp\left(-\frac{h^2}{2\sigma^2}\right) = \begin{array}{|c|c|c|c|c|} \hline 5 & 12 & 15 & 12 & 5 \\ \hline \end{array} \qquad \exp\left(-\frac{k^2}{2\sigma^2}\right) = \begin{array}{|c|} \hline 5 \\ \hline 12 \\ \hline 15 \\ \hline 12 \\ \hline 5 \\ \hline \end{array}$$

Gradient

The vector of partial derivatives of a the brightness function f w.r.t. spatial coordinates. In 2D case, f is f(x,y), so the gradient is:

$$\nabla f = [\ \frac{\partial f}{\partial x} \quad \frac{\partial f}{\partial y}\ ]$$

Result: an n-dimensional vector, where n is the dimensionality of the space (n=2).

How to effectively approximate a gradient on a raster using convolution filtering?
Naive implementation: use these masks for horizontal and vertical component:

1	-1

1
-1

Disadvantages:

- susceptibility to noise
- lack of invariability due to rotation

Sobel filter

Masks for estimating the horizontal and vertical gradient components:

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

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

Filter response can be negative.

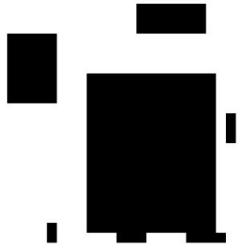
The length of gradient vector (‘total gradient’):

$$g(x,y) = \sqrt{[M_h * f(x,y)]^2 + [M_v * f(x,y)]^2}$$

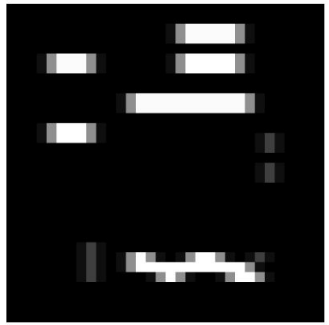
(the square root often dropped in practice if not necessary)

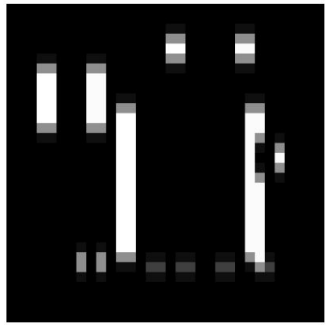
Example

Input image and the results of its convolution with the vertical and horizontal Sobel mask:

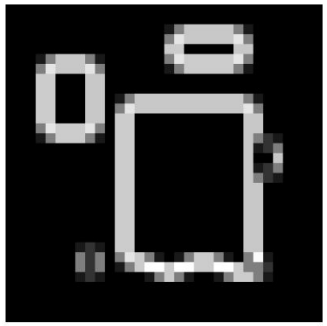


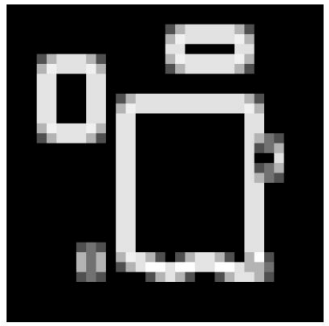
After squaring:





After addition and applying square root:

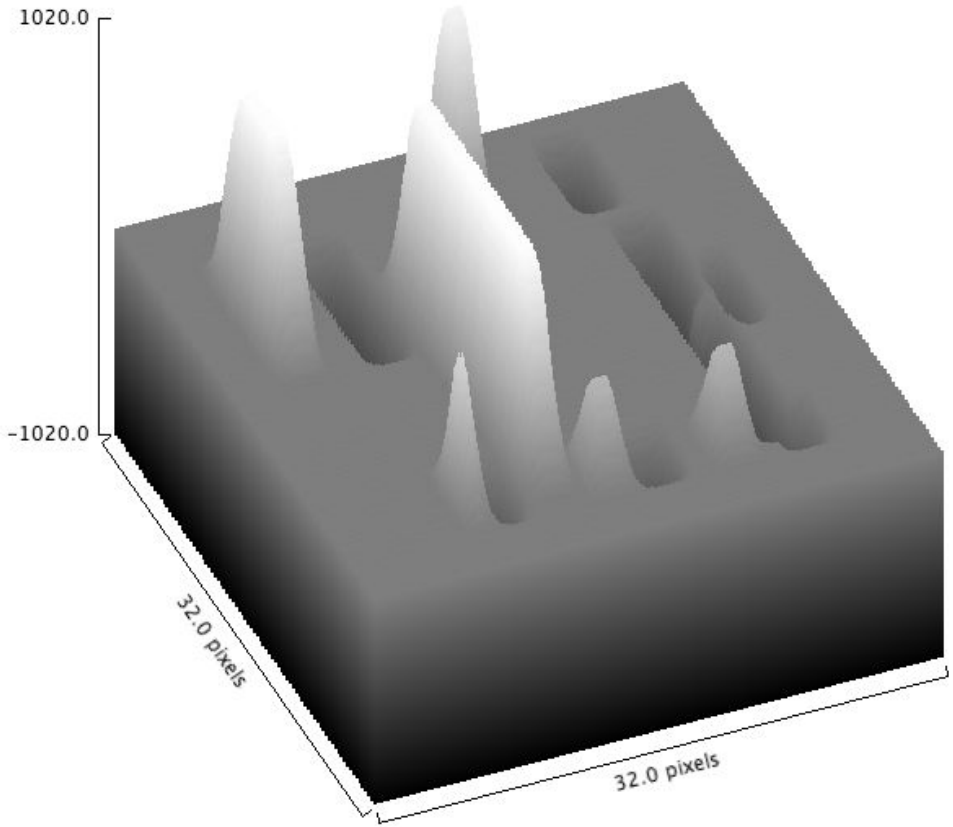




Example

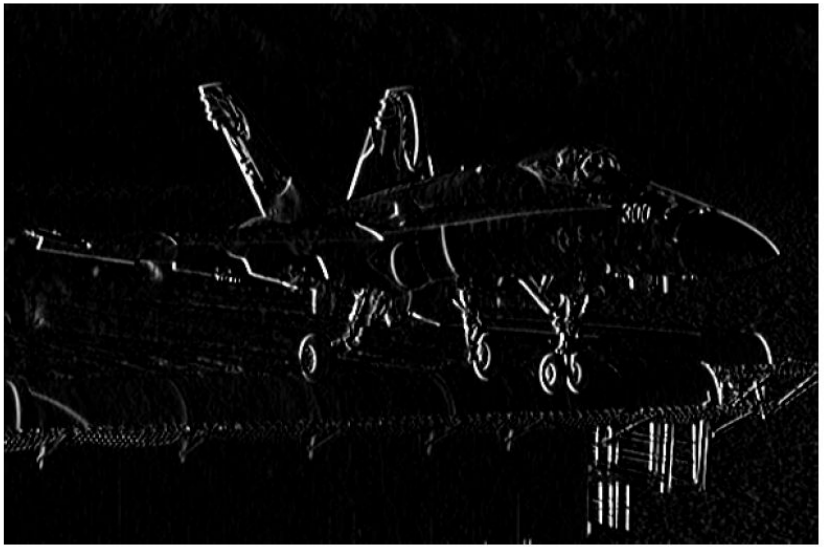
The result of convolution is a signed image:

(3D rendering of the horizontal gradient component for the image from the previous slide)

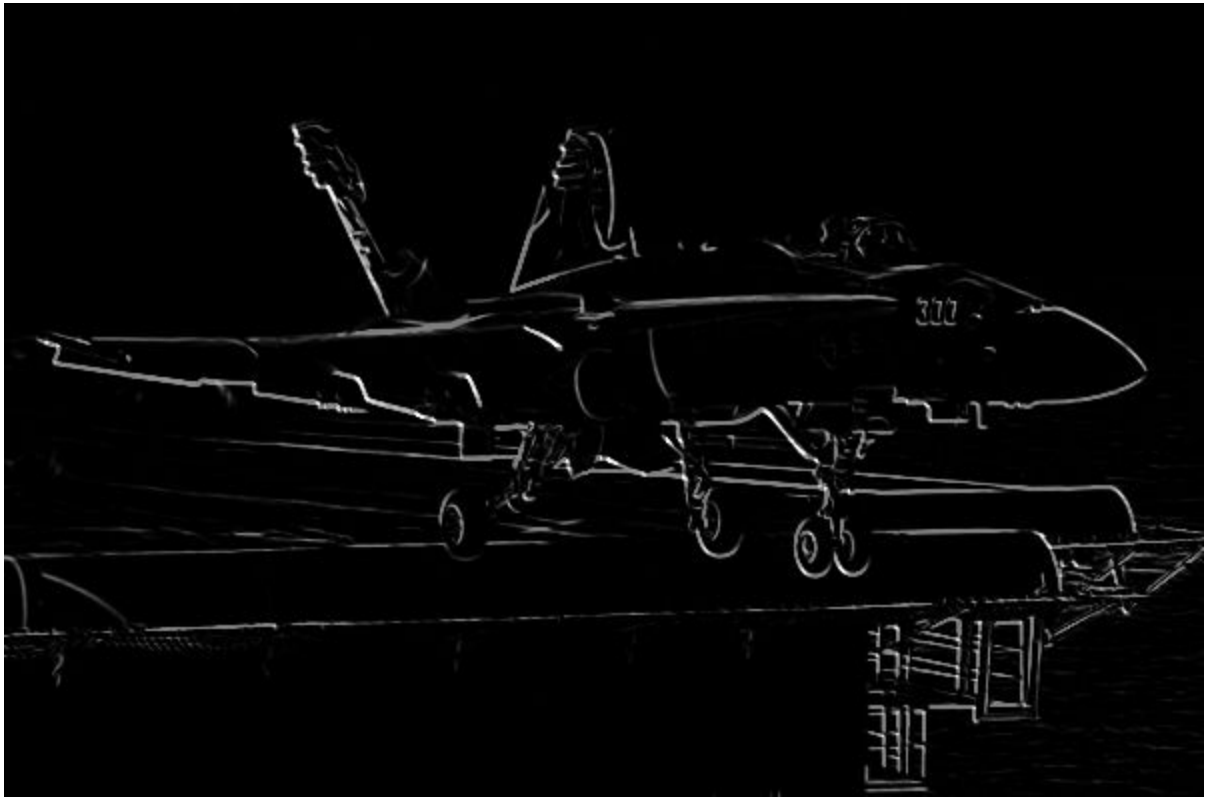



300


300



Sum of squares of gradient components



Scharr filter

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

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

- Better approximation of gradient
- Offers better invariance with respect to rotation (‘more isotropic’)
- However: still integer-valued
  - More precise approximations can be obtained with floating point weights.
  - The differences are, however, negligible for most usage scenarios.

Laplacian

A second order differential operator that measures the local variation of a gradient.
Definition for continuous and doubly differentiable functions f:

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

Most common discrete realization:

0	1	0
1	-4	1
0	1	0

Brief explanation of intuition for one dimension (difference of differences):

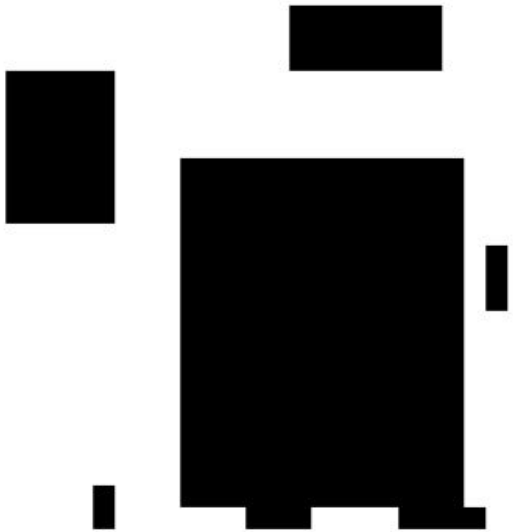
$$\Delta^2 f(x) = \Delta f(x + 0.5) - \Delta f(x - 0.5) = (f(x + 1) - f(x)) - (f(x) - f(x - 1)) = 1 \cdot f(x + 1) - 2 \cdot f(x) + 1 \cdot f(x - 1)$$

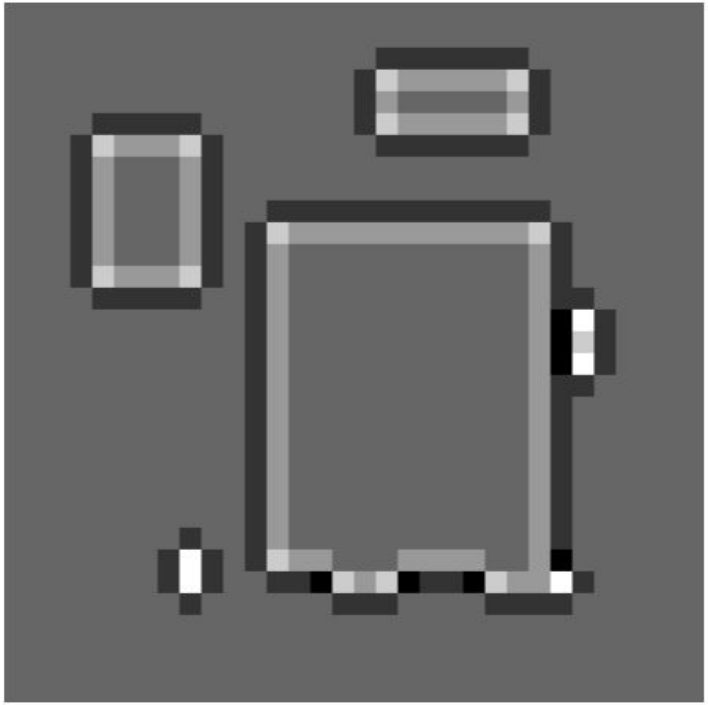
So the weights are (1,-2,1).

- Composing for X and Y axis gives the above matrix of weights.
- (This is a short version of the derivation based on Taylor series)

Example

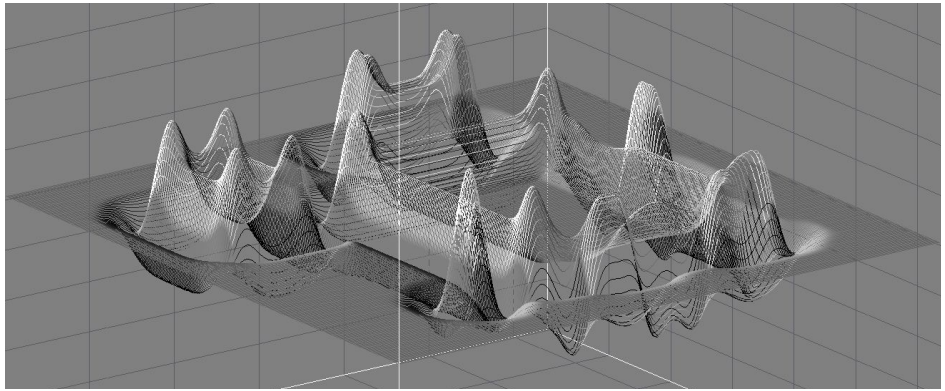
The input image and the outcome of its convolution with 3x3 Laplacian filter:





Laplacian: properties

- Each edge is represented by the zero crossing of the Laplacian.
  - This transition precisely locates the 'center' of the edge (in terms of thickness).
  - 1D interpretation: the inflection point of the brightness curve
- The Laplacian sign reflects the object-background relationship.
- Works often better than the Sobel filter when edges are blurred.



Example

Theoretical motivations

Some of the filters introduced earlier can be ‘inferred’ from the Taylor series expansion of a function:

$$f(x) = \sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x-a)^n = f(a) + \frac{f'(a)}{1}(x-a) + \frac{f''(a)}{2}(x-a)^2 + \frac{f'''(a)}{6}(x-a)^3 + \dots$$

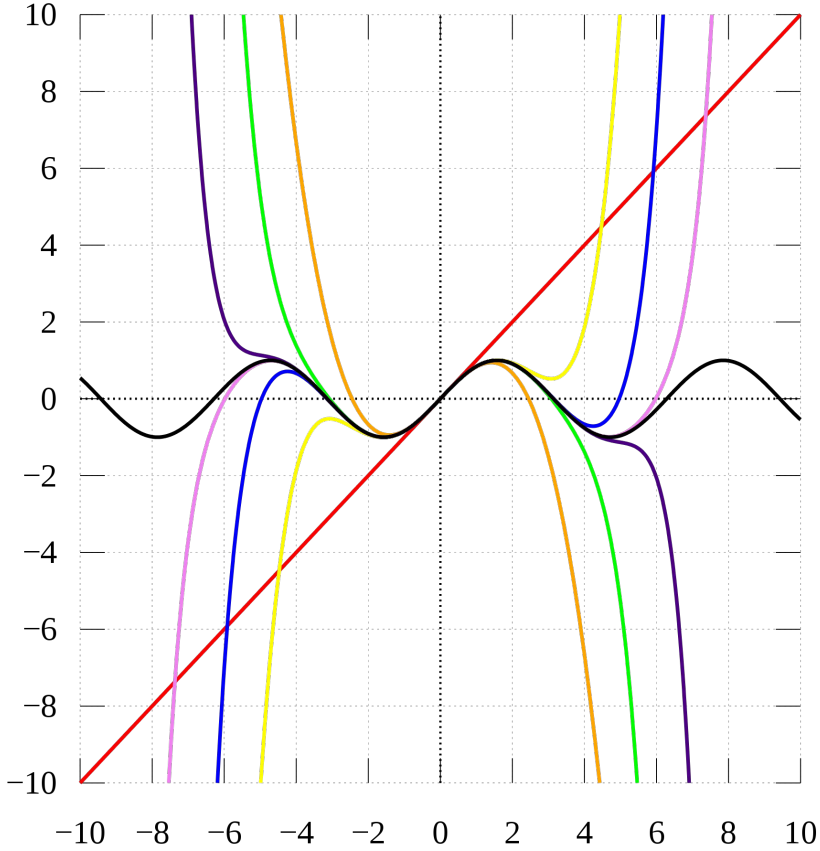
This expansion allows us to express the value of a function at a given point x using the value of that function and its derivatives at another point a.

Dropping higher order terms allows us to obtain practical approximations

- (and free us from the calculating higher-order derivatives of the image)

Example

Successive approximations of the sine function with the (truncated) Taylor series.



Theoretical motivations

Sketch of the derivation for 1D Laplacian (O() - higher order expressions):

$$f(x+h) = f(x) + hf'(x) + \tfrac{1}{2}h^2 f''(x) + O^+(h^3)$$

$$f(x-h) = f(x) - hf'(x) + \tfrac{1}{2}h^2 f''(x) + O^-(h^3)$$

After subtracting the second equation from the first one:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2)$$

For the second derivative:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h)$$

We obtain the vector of weights (1,-2,1) in 1D. For functions of two variables, we would get the mask of the 2D filter.

Nonlinear filtering

Definition

- Similarly to convolution, not a *strictly* local operation.
  - The result for a given pixel depends on more than one pixel in the input image.
  - The set of input pixels (neighbors) determined by a mask (usually).
  - The mask usually does not define weights - it only defines the spatial extent of the dependency.
    - Most common: a n×n square
- Note: Convolution is a linear operation.
  - *All* linear operations can be realized with convolution.
- Therefore: nonlinear filtering comprises operations which cannot be realized via convolution, in particular non-linear statistical operators:
  - median
  - maximum
  - minimum
  - others, e.g. Ordered Weighted Averaging (OWA) operators: $F(a_1, \ldots, a_n) = \sum_{j=1}^{n} w_j b_j$ where b_js are a_is sorted in some manner.
    - Maximum, minimum and median are special cases of the OWA.

Median filter

Very effective for removal of ‘salt and pepper’ noise.

Example: Application of 5x5 median filter.

Median filtering does not preserve mass (brightness)

- In contrast, for instance, to the Gaussian filter.

Averaging (linear) filter vs. median filtering

Other nonlinear filters

- Hampel Filter: removal of outliers
  - Calculate the median *m* of input values x_i
  - Calculate the standard deviation σ from x_is
  - If the $|x_i - m| > \alpha\sigma$, replace x_i with m *(α* - parameter of the method)
  - The x_is modified in above way form the output of the filter (which can be then filtered using another filter, e.g. median)
- Convolutional networks (layers) with nonlinear activation functions.
  - A single layer of this type can be represented as a composite of a convolution and a single-point operation.
  - For more sophisticated nonlinearities, multiple layers with nonlinearities have to be used.
- Manually designed filters,
- Filters optimized with metaheuristics.
  - Example: genetic programming

Case study 1

Evolutionary synthesis of nonlinear feature detectors (filters) for segmentation of anatomical structures in human retina.

Based on:
Genetic Programming with Alternative Search Drivers for Detection of Retinal Blood Vessels
Krzysztof Krawiec, Mikołaj Pawlak, Evo* 2015.

Morphological filtering

Morphological filtering: Introduction

- A class of nonlinear filtering operations.
- Assumption: binary images.
  - E.g. white objects on black background
- Idea:
  - We assume some structural/structuring element (parameter of image processing operation):
    - in practice a small object, e.g. a 3x3 pixel square.
  - We define operations by testing the inclusion of displacements of this object in objects visible in the image.
- Motivation (one of the motivations):
  - Images are sets of points.
  - The inclusion relation is the most elementary relation between sets.
- Image morphology (mathematical morphology) is one of the best formalized and most elegant branches of image analysis and processing.

Morphological filtering: Introduction

The domain: sets of points in d-dimensional space.

- Points: a, b, c, ... $\in \mathbb{R}^d$
  - The space can be continuous
  - Dimensionality d can be arbitrary
  - We identify points with vectors
- Sets of points (images, objects, structural elements): A, B, C, …
- Common convention for raster images:
  - 0 - background, i.e. a point that does not belong to an object (set of pixels)
  - 1 - a point that belongs to an object (set of points).

Notice: in general, set of points ≠ connected region.

The elementary morphological operations:

- dilation,
- erosion

Erosion

For image A and structural element B:

$$A \ominus B = \bigcap_{b \in B} A_{-b}$$

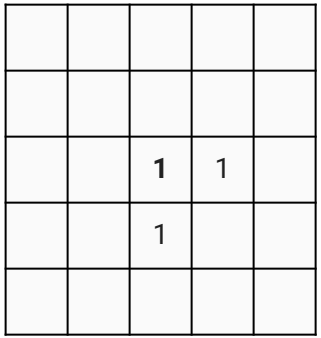
Interpretation:

- The intersection of the displacements of image (object) A by vectors/points of B.
- Negation of b motivated by theory – not always used in practice.
  - Almost equivalent to Minkowski addition,
  - Difference: symmetric reflection of the structural element
  - Equivalence preserved when B is symmetric
    - This is very common in practice: one typically wants processing to be isotropic.
- Important: vectors in B are relative to the origin of the coordinate system in B.
  - In practice, B always contains the (0,0) element, i.e. zero displacement vector.

Mutual correspondence of rasters and points sets

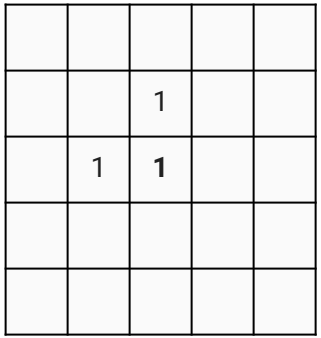
Image A

A = {(0,0), (1,0), (0,1)}



Structural element B

B = {(0,0), (-1,0), (0,-1)}



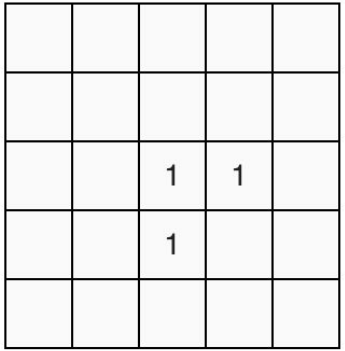
(Location of the origin marker in bold)

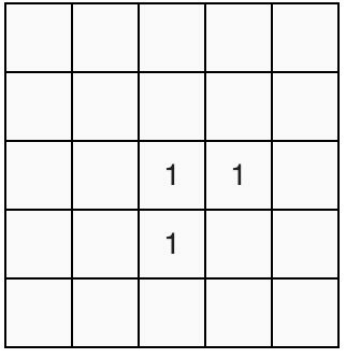
Erosion

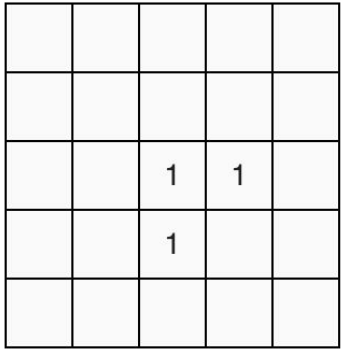
Image A

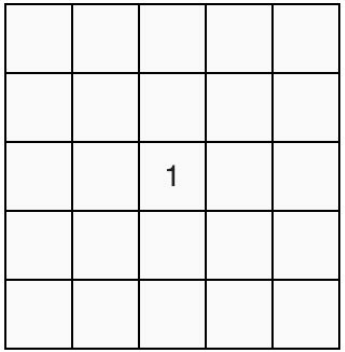
Structural element B (origin in the middle of the inset)

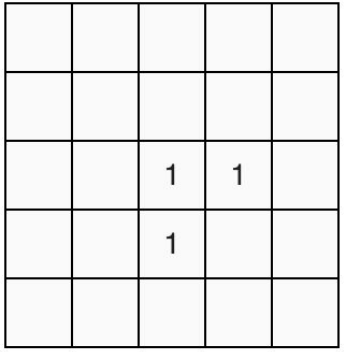
Result

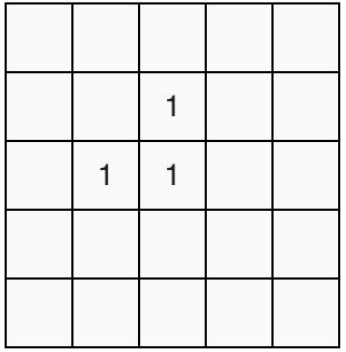


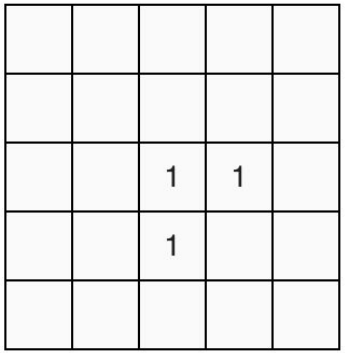


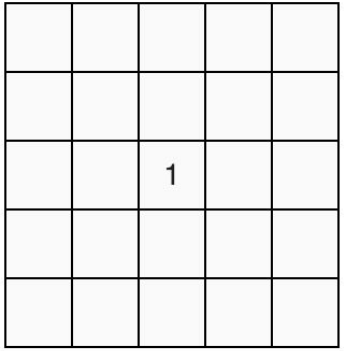


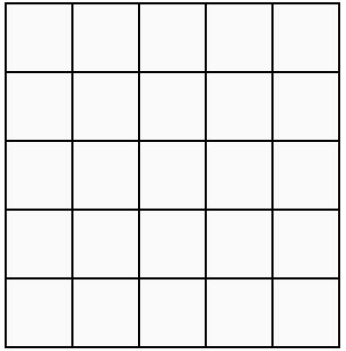








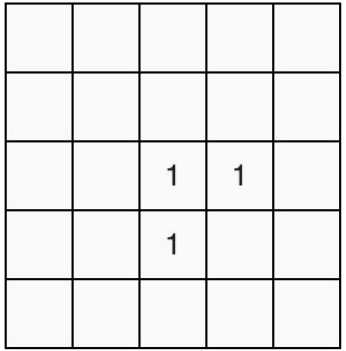


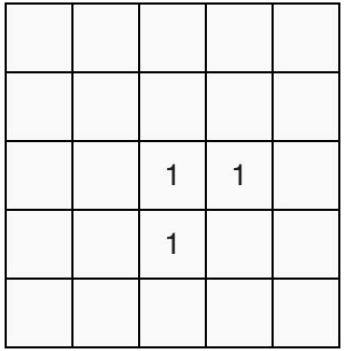


Erosion

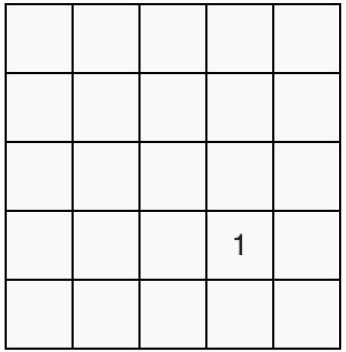
If the structural element is shifted with respect to the origin, so will be the result.

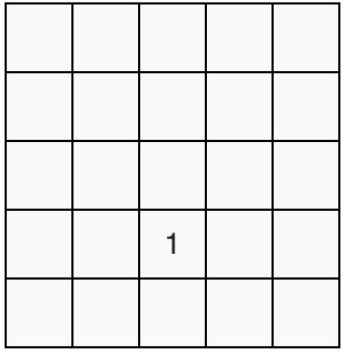
A



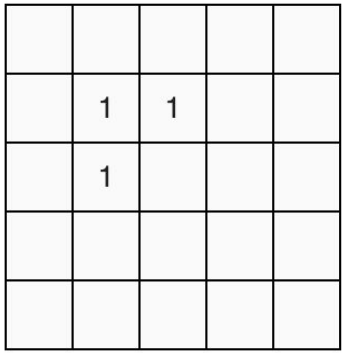


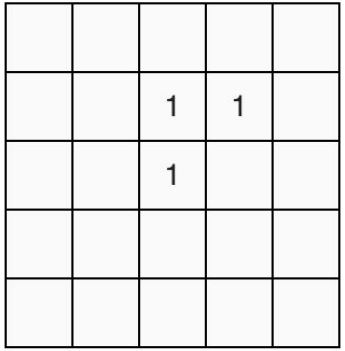
B





Result





Dilation

For image A and structural element B:

$$A \oplus B = \bigcup_{b \in B} A_{-b}$$

Interpretation:

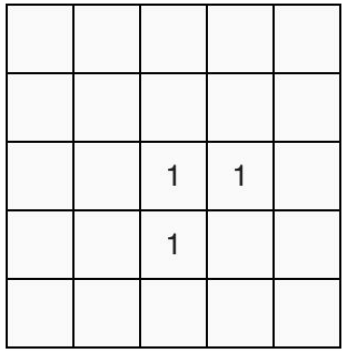
- The sum of the displacements of image (object) A by vectors/points of B.
- (other comments like for erosion)

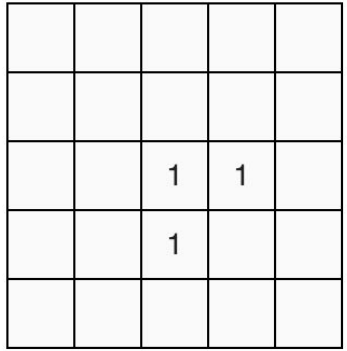
Dilation

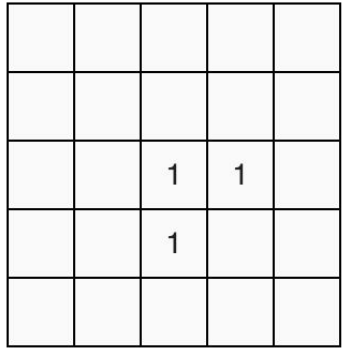
A

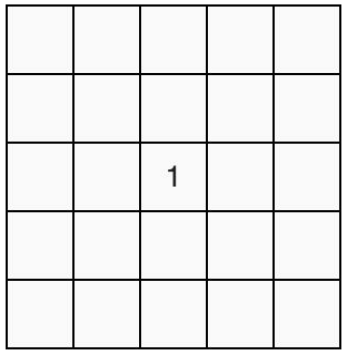
B

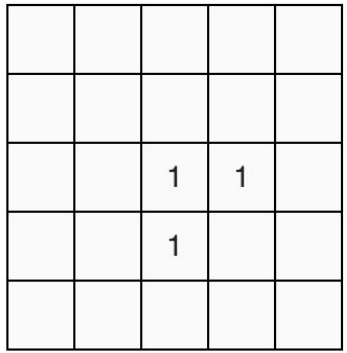
Result

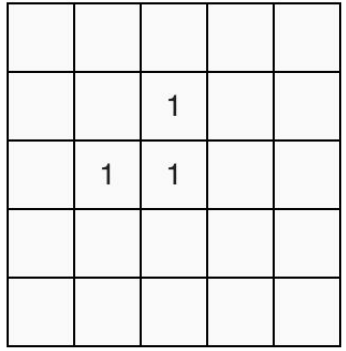


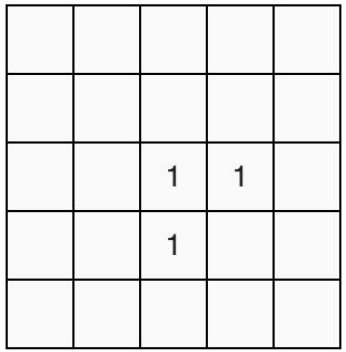


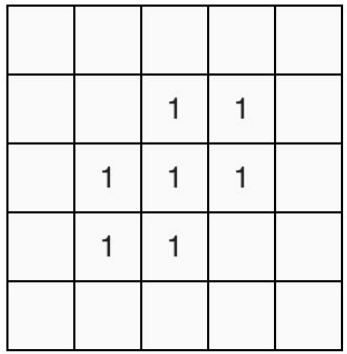


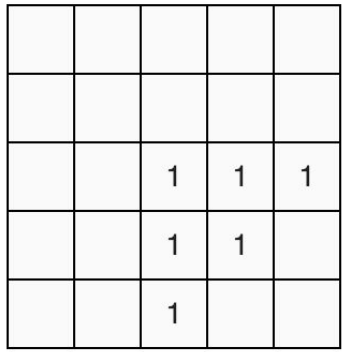






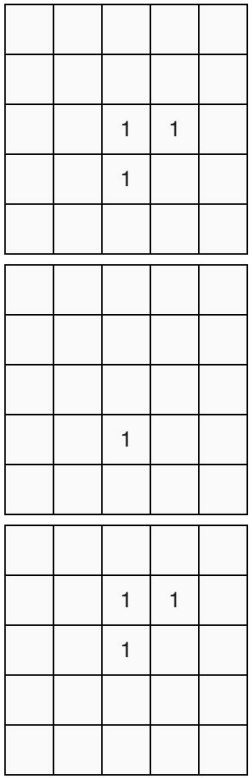




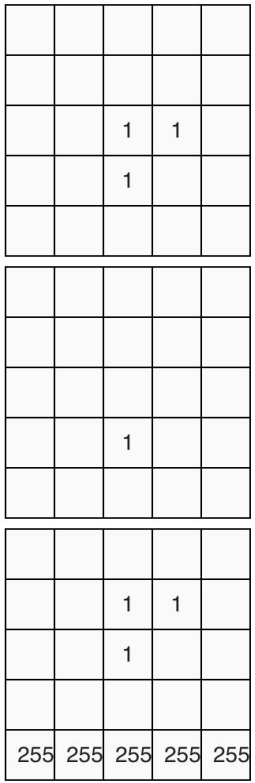


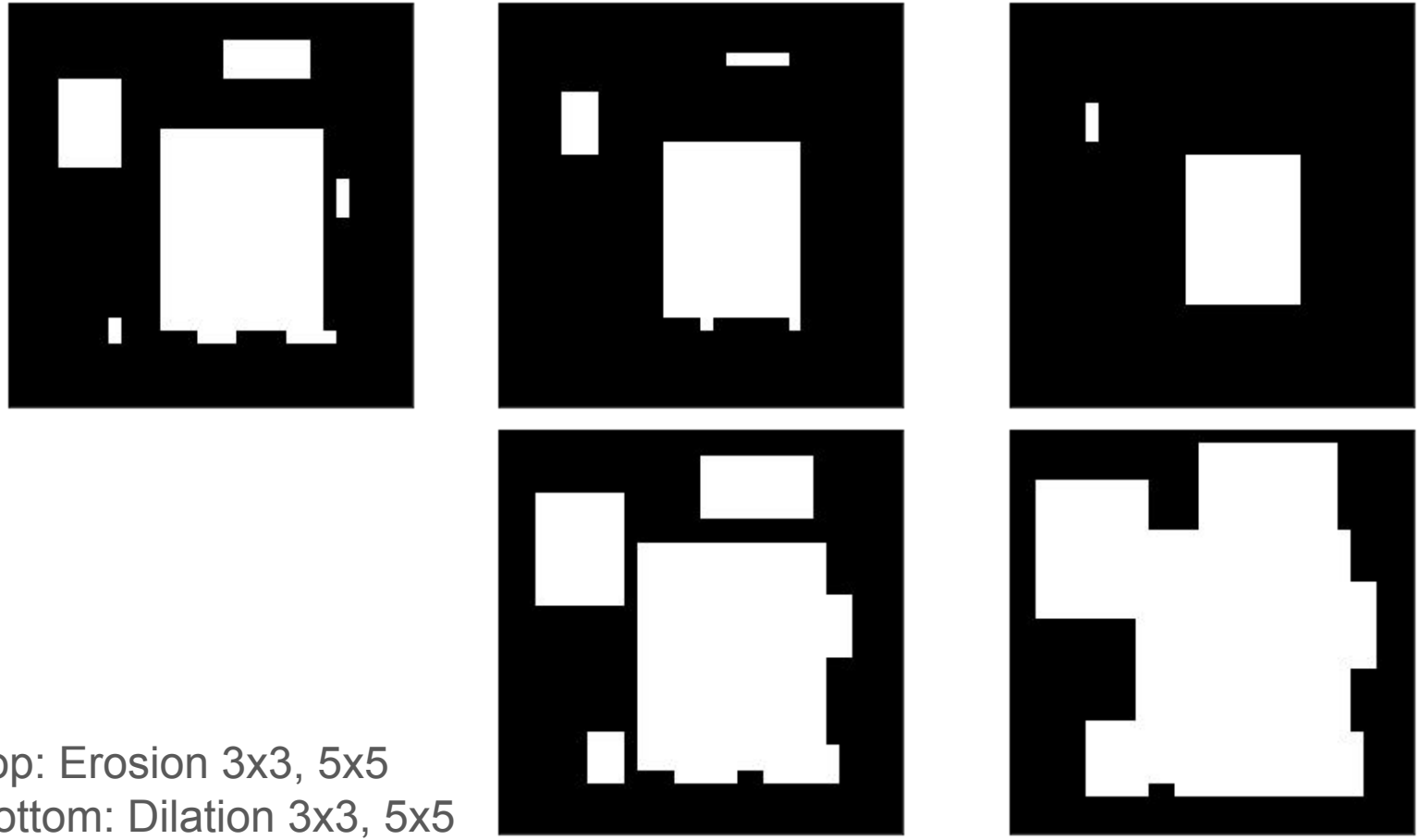
Warning: border effects in some implementations

`cv2.erode(A,B, borderValue= 0)`

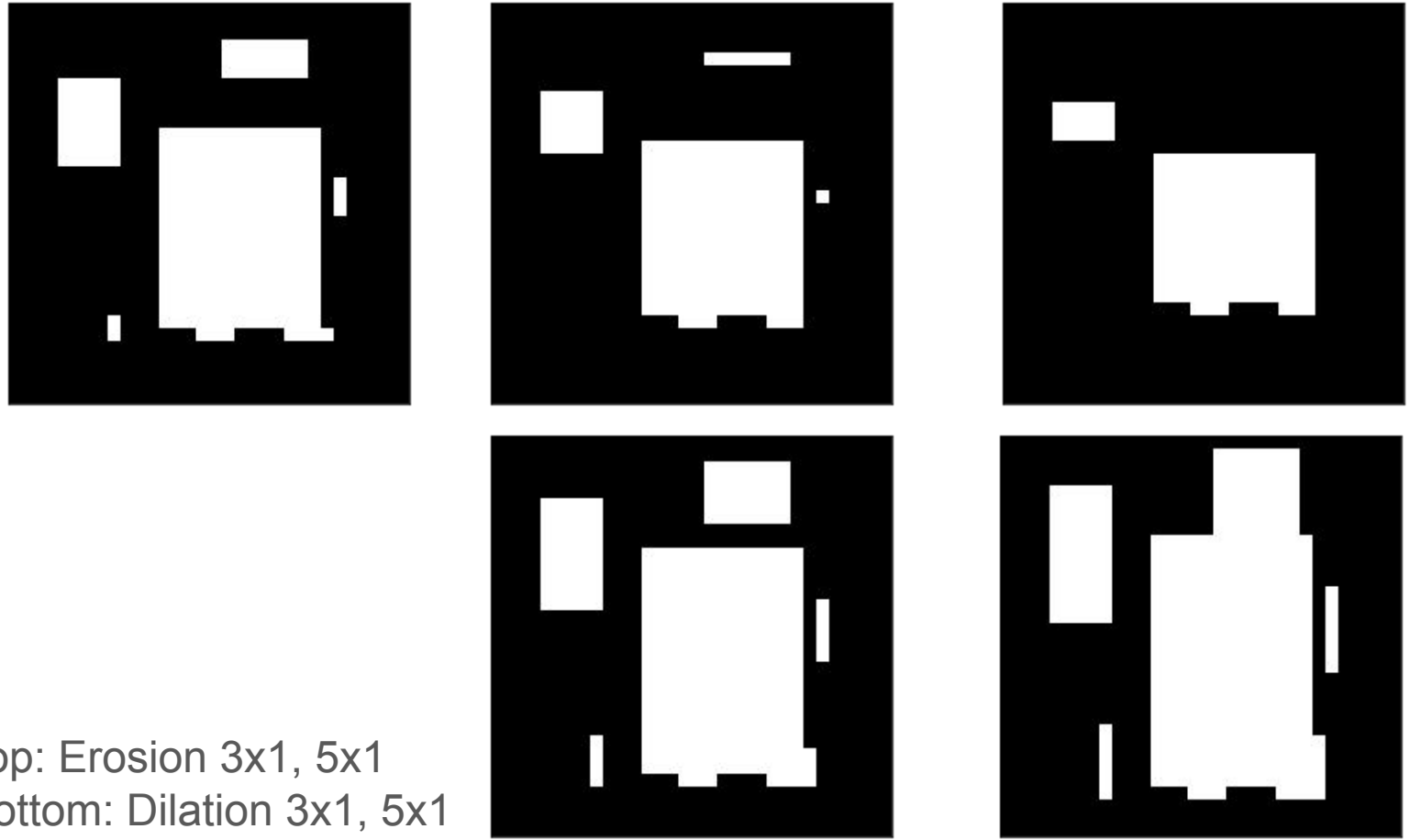


`cv2.erode(A,B)`





Top: Erosion 3x3, 5x5
Bottom: Dilation 3x3, 5x5



Top: Erosion 3x1, 5x1
Bottom: Dilation 3x1, 5x1

Properties of dilation and erosion

Translation invariance

$$A_d \oplus B = (A \oplus B)_d$$
$$A_d \ominus B = (A \ominus B)_d$$

Monotonicity (weak monotonicity)

$$A \subseteq B \Rightarrow (A \oplus C) \subseteq (B \oplus C)$$

Dilation is also extensive

$$A \subseteq A \oplus B$$

Erosion is anti-extensive

$$A \ominus B \subseteq A$$

(requires (0,0) ∈ B)

Structural element

Typical shapes of structural elements

- squares
- rectangles
- diamonds
- approximations of discs.

Desirable property:

- radial symmetry, as it makes morphological operations rotation-invariant
  - radial symmetry at least with respect to some angles of rotation
    - e.g. 0, 90, 180, and 270 degrees for squared structural elements.
- (translation invariance provided by definition)

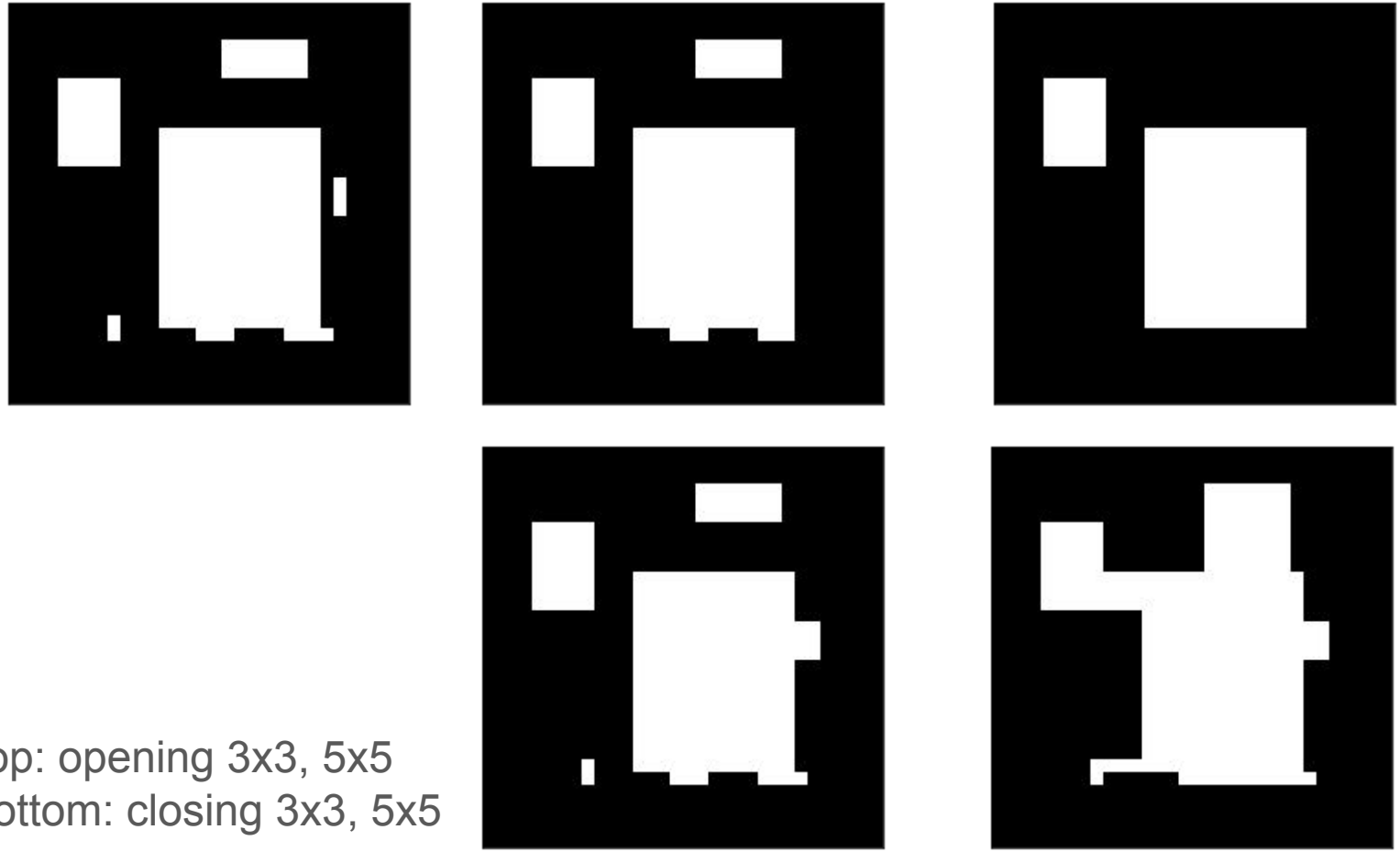
Opening and closing

Opening of A with B: composition of erosion and dilation of A with B, where the first of these operations uses mirrored B:

$$A \circ B \equiv (A \ominus \hat{B}) \oplus B$$

Closing of A with B: composition of dilation and erosion of A with B, where the first of these operations uses mirrored B:

$$A \bullet B \equiv (A \oplus \hat{B}) \ominus B$$



Top: opening 3x3, 5x5
Bottom: closing 3x3, 5x5

Properties of opening and closing

- Translation invariance
- Weak monotonicity
- Opening: antiextensitivity
- Closing: extensitivity
- Idempotence: multiple applications of the same operation do not change the result anymore:

$$(A \circ B) \circ B = A \circ B, \quad (A \bullet B) \bullet B = A \bullet B$$

Morphological filtering

Morphological filter is any operation that is monotonic and translation-invariant.

The Representation Theorem (by Matheron, 1975):

1. Every morphological filter can be represented as a sum of erosions:

$$\Psi(A) = \bigcup_{B \in Ker[\Psi]} A \ominus B$$

2. Every morphological filter can be represented as an intersection of dilations:

$$\Psi(A) = \bigcap_{B \in Ker[\Psi^*]} A \oplus B$$

Top Hat and Black Hat

Top hat

- Pixel-wise* difference between the image and its opening:

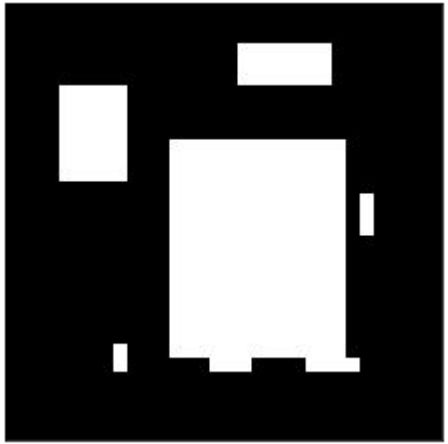
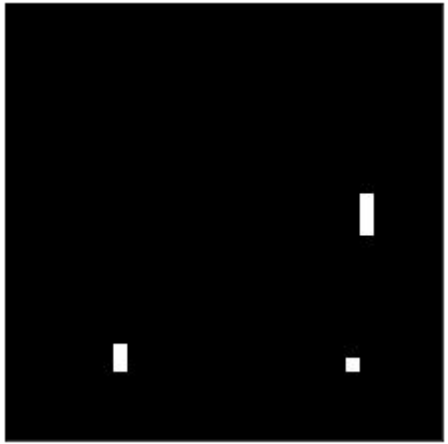
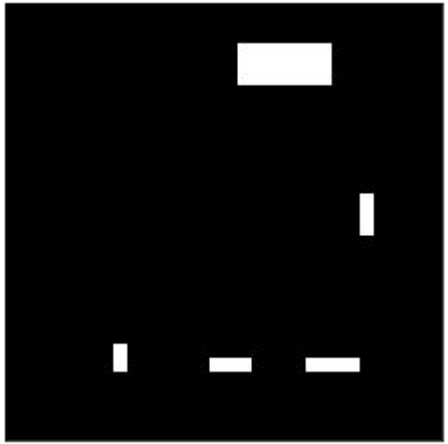
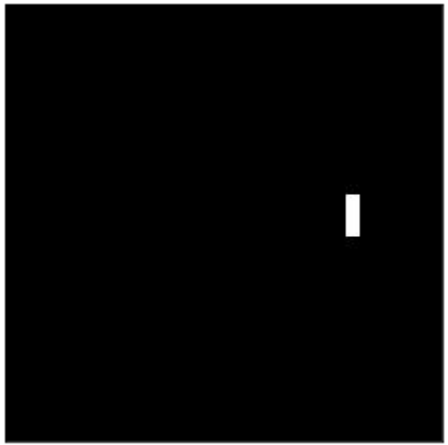
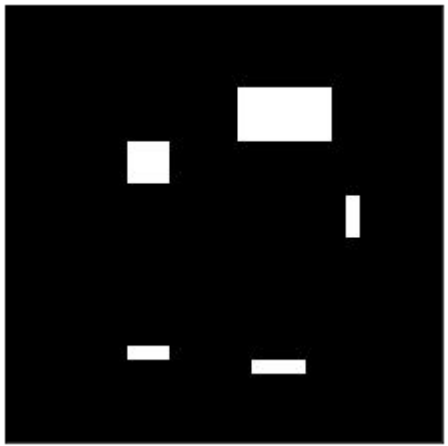
$$A - (A \ominus \hat{B}) \oplus B$$

Black Hat

- Pixel-wise* difference between the closing of an image and that image:

$$((A \oplus \hat{B}) \ominus B) - A$$

* That is, set-theoretic difference in the context of morphology, as images are treated as sets of points.

Top: Top Hat 3x3, 5x5
Bottom: Black Hat 3x3, 5x5

Efficient implementation of morphological operations

Implementation based directly on the definitions of morphological operations is highly ineffective, as it involves shifting/translating the input image multiple times.

- This becomes particularly costly for large structuring elements.

However, it turns out that erosion and dilation are equivalent to known operations of local nonlinear filtering:

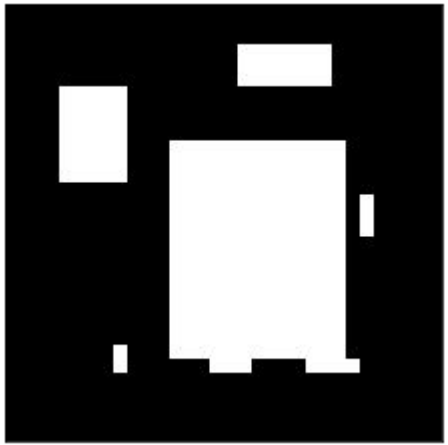
- Erozja = min
  - *min pooling* with stride 1
- Dylatacja = max
  - *max pooling* with stride 1

This observation makes morphological filtering even more attractive.

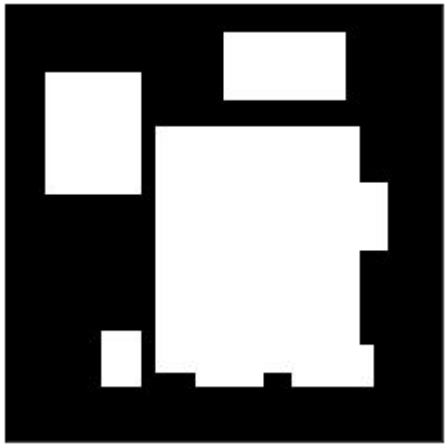
- Min/max pooling are very efficient on most computing platforms.

Morphological operations and max/min pooling

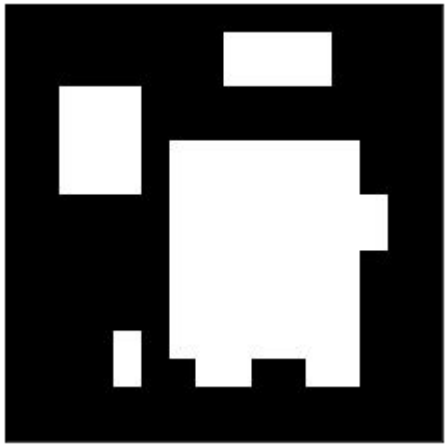
`i32`



`cv2.dilate(i32,np.ones([3,3]))`



`block_reduce(i32,(3,3),np.max)`



Explanation of observed difference in results:

```
r = skimage.measure.block_reduce(i32, (3,3), np.max)
r.shape
```

(11, 11)

Morphological operations for monochrome (non-binary) images

Motivations

A naive approach to make morphological operators applicable to non-binary images:

1. Binarize (threshold) the image
2. Use the operations of binary (conventional) morphology

However, step 1 incurs significant information loss.

Q: Can we generalize the morphological operations to non-binary images in a more principled way?

A: Yes; a range of approaches have been proposed.

1. Keep using min/max operators (see next slides)
2. Generalization with *functions*
3. Generalization with *umbras*

Using the max operator for dilation

Top: Input image A and its local max operator (max pooling) with 5x5 and 11x11 masks.

Bottom: The differences between the results and A.

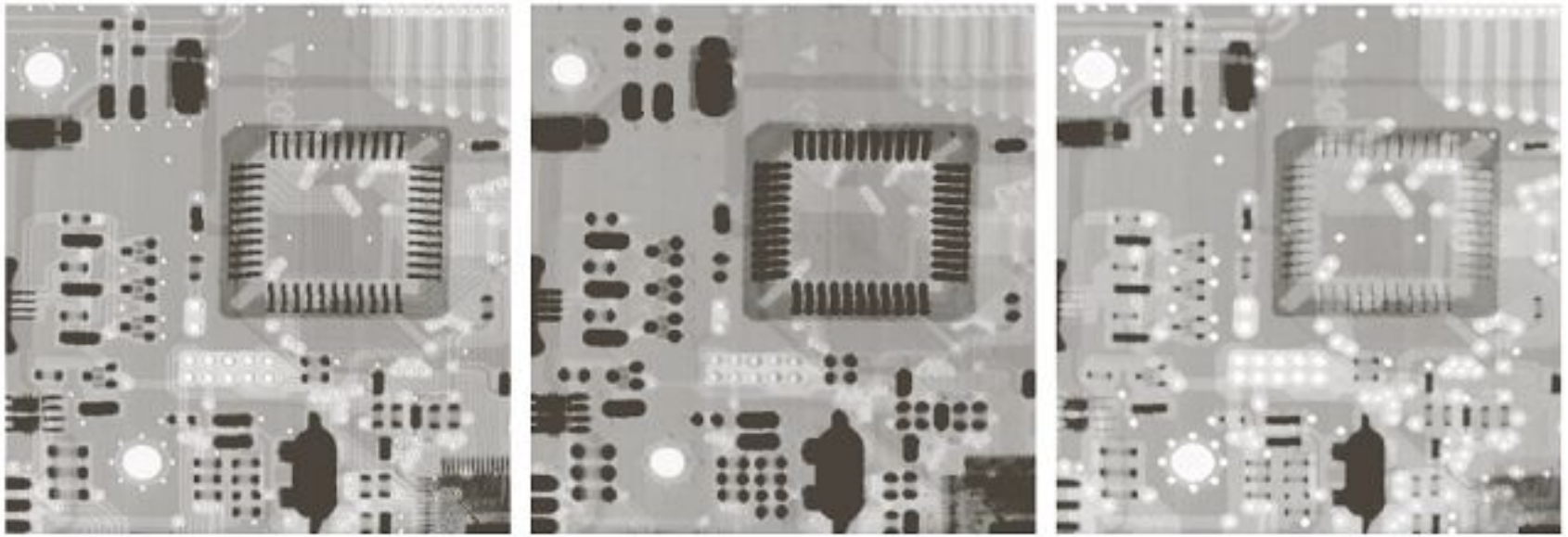
Using the min operator for erosion

Top: Input image A and its local min operator (min pooling) with 5x5 and 11x11 masks.

Bottom: The differences between the A and the results.

Real-world example

Erosion and dilation with a structural element being a flat disc with 2 pixel radius:



Limitations of min/max pooling

Using min/max pooling does not offer a ‘fully-fledged’ generalization of morphological operations to monochrome images, because it’s ‘asymmetric’:

- while the input image (A) is a monochrome image,
- the structuring element (B) is just a mask (e.g. 3x3 mask) – there are no values associated with individual pixels in B.

The other two extensions (via functions and umbras) are more ‘symmetric’ in that respect (see next slides).

The functional take on morphology

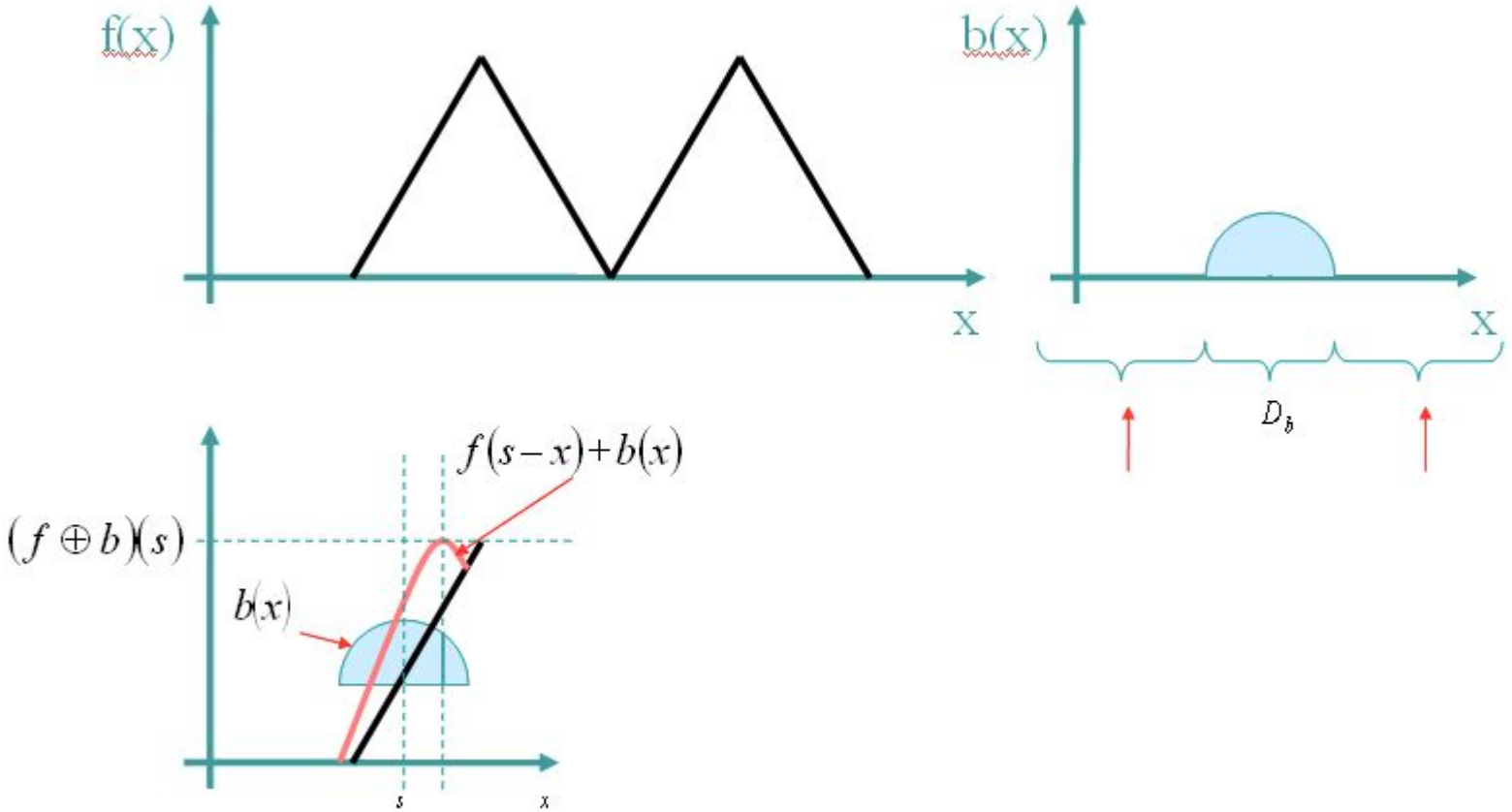
- We consider real-valued functions instead of sets.
- Functions have support, e.g. D_f is the support of function f.
- For 1D functions: dilation of image f with structural element b:

$$(f \oplus b)(s) = \max\{f(s-x) + b(x) : (s-x) \in D_f, x \in D_b\}$$

Notice:

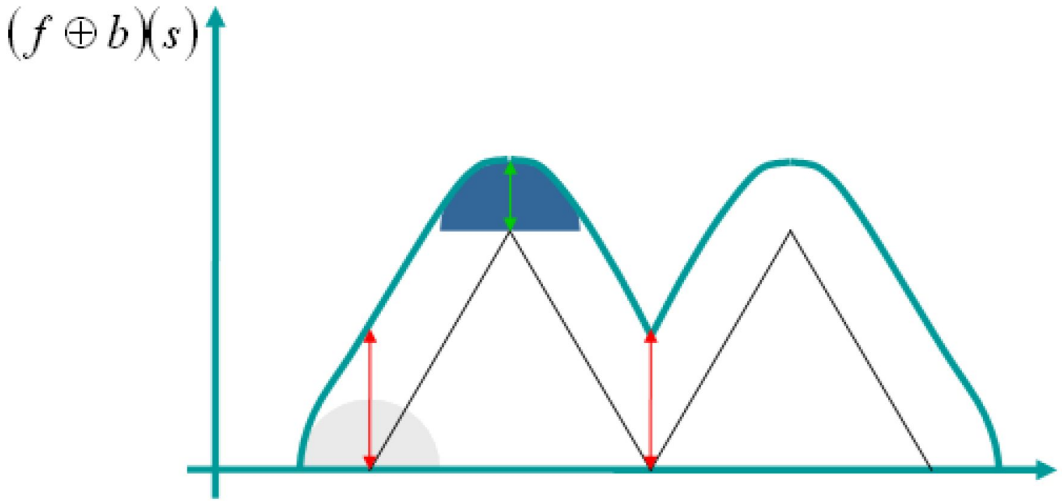
- The maximum operator – as in the ‘regular’ dilation operator.
- The ‘member of’ condition requires f and b to overlap at at least one point.

Morphology with functions: Example



Effects of dilation

1. Greater extent/dimensions of objects.
2. Increased brightness
    - Notice: brightness difference is not trivially the same for all affected points!



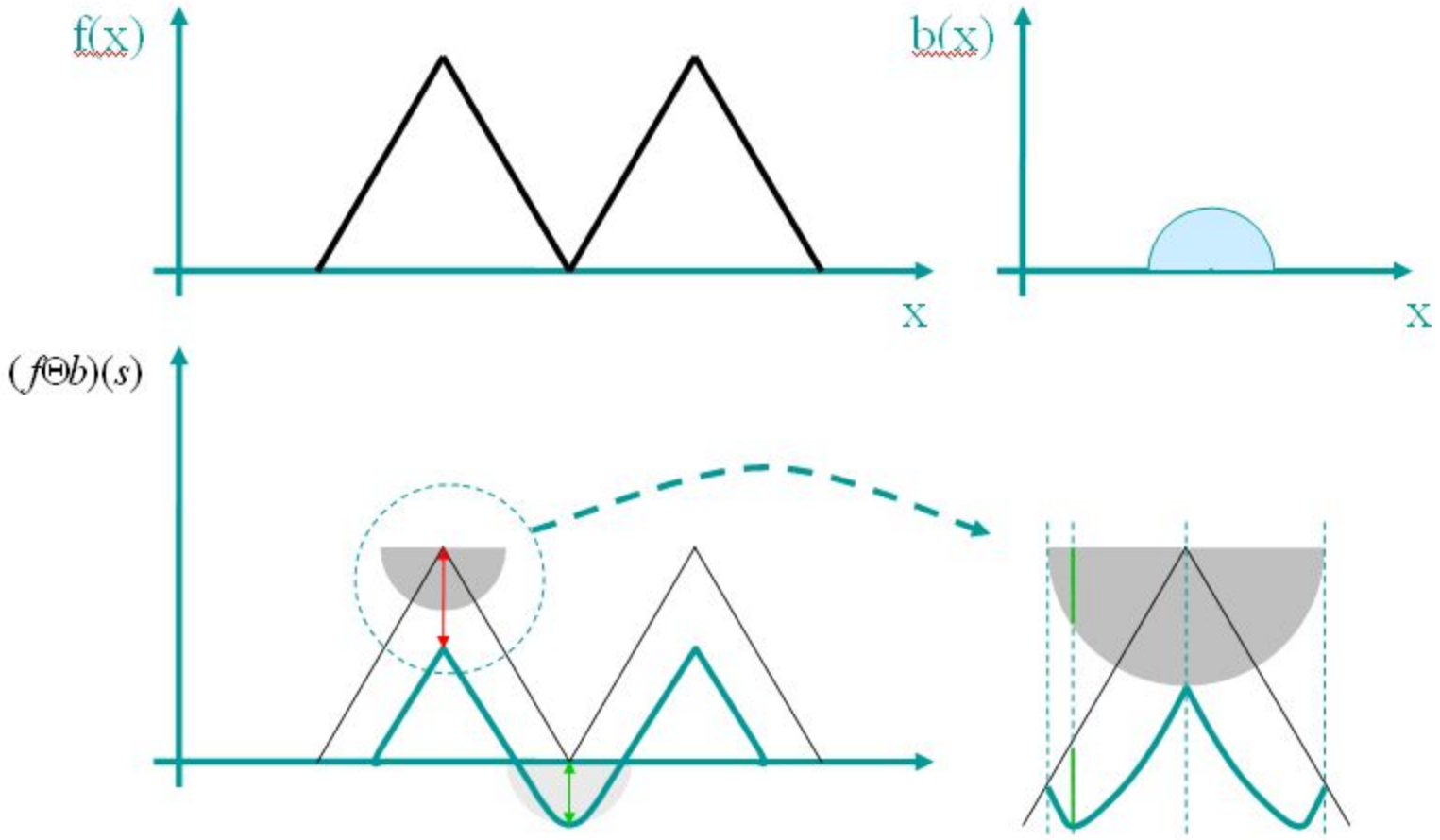
Erosion with functions

Definition:

$$(f \ominus b)(s) = \min\{f(s+x) - b(x) : (s+x) \in D_f, x \in D_b\}$$

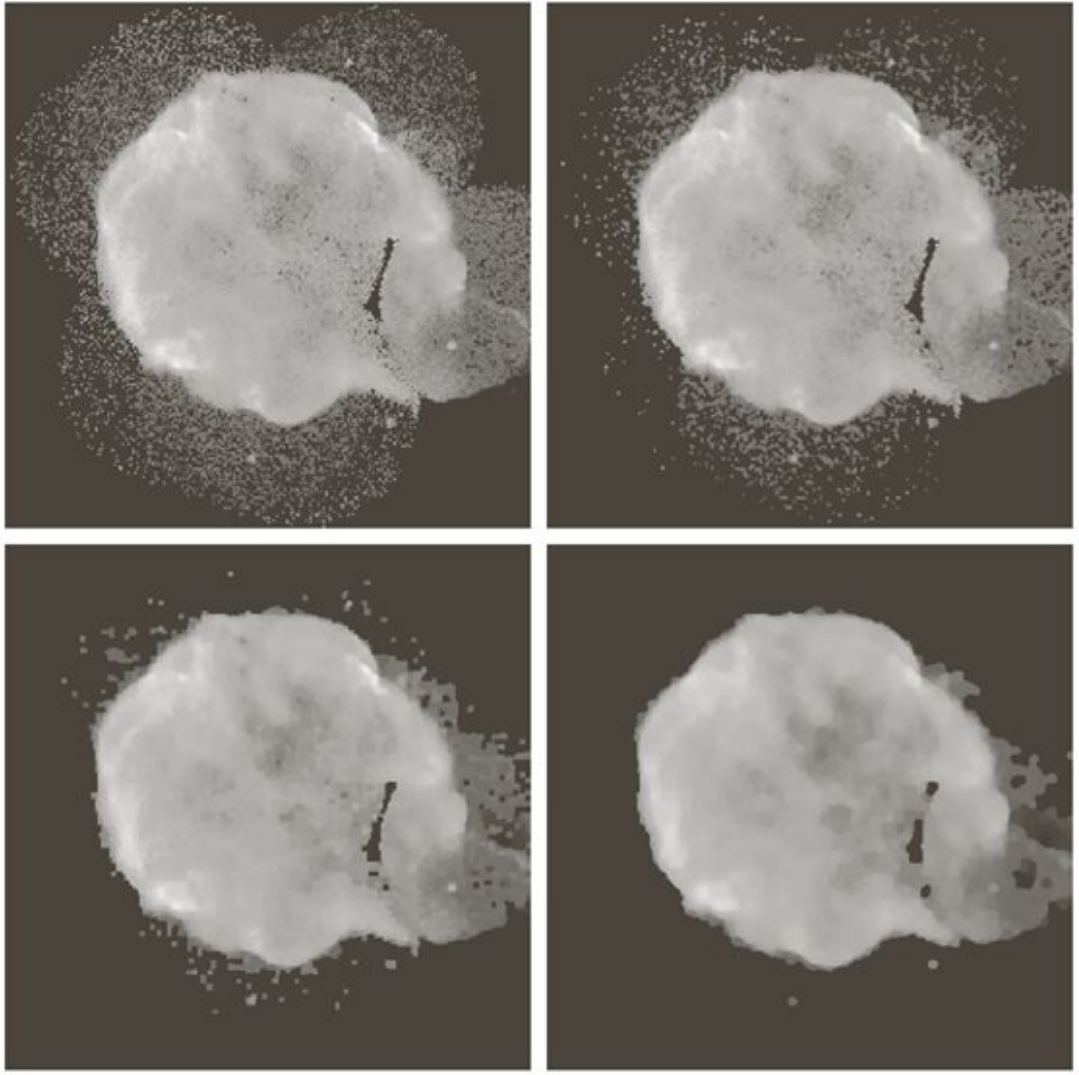
Analogous effects:

1. Decreased dimensions/extent of objects.
2. Lower brightness.


f(x)
x
b(x)
x
$(f \ominus b)(s)$

Example

Supernova Cygnus Loop after opening and closing with discs of radius, respectively, 1, 3 and 5 pixels.

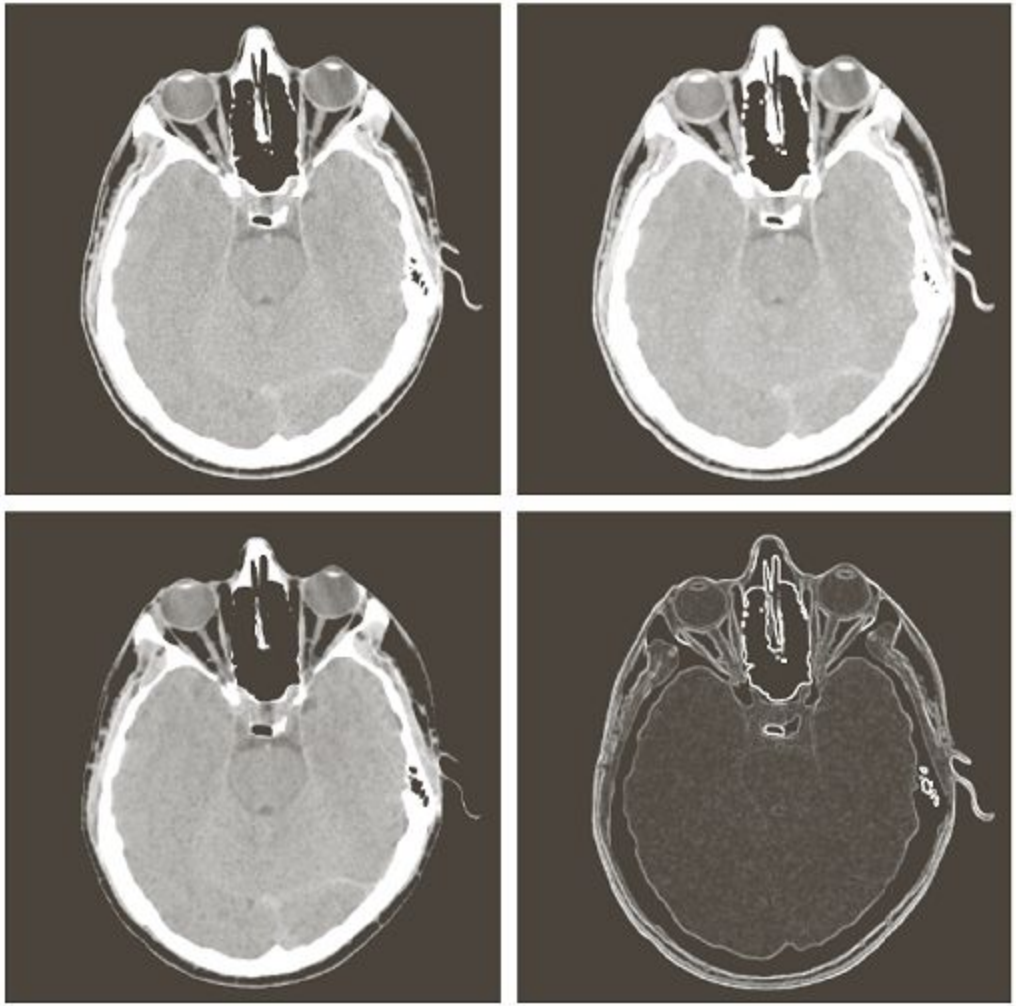


Morphological gradient

Erosion, dilation,
and the difference between
dilation and erosion

$$(f \oplus b) - (f \ominus b)$$

(the minus sign stands here
for pixel-wise subtraction)

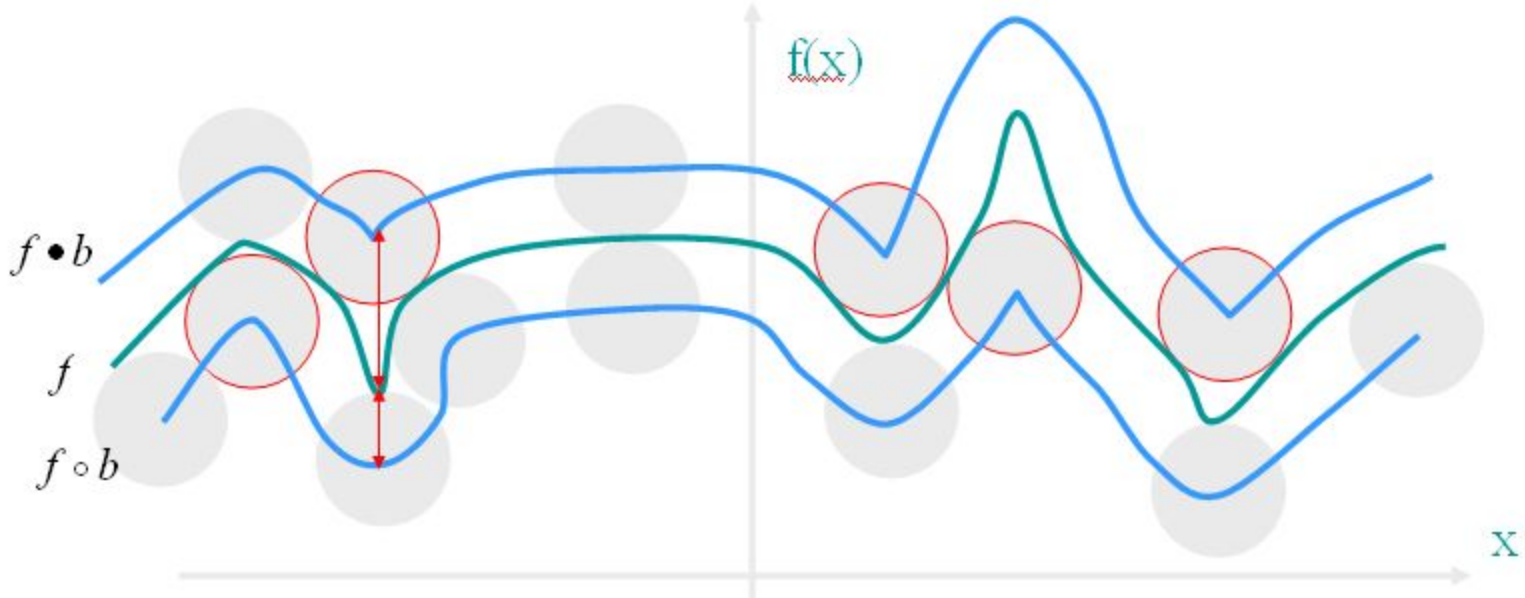


Opening and closing

Analogously to binary morphology:

$$f \circ b = (f \ominus b) \oplus b,\ f \bullet b = (f \oplus b) \ominus b$$

Elegant geometric ‘rolling ball’ interpretation (for spherical structural elements):



Generalization with umbras

Key idea:

- transform the 2D monochrome input image f into 3D binary image U(f),
- process U(f) using operations of 3D binary morphology, obtaining U’,
- transform the resulting 3D binary image U’ back to the space of 2D monochrome images, obtaining f’

$$\begin{array}{ccc} f & & f' \\ \downarrow & & \uparrow \\ U(f) & \xrightarrow{operacje\ na\ zbiorach} & U' \end{array}$$

Umbras: formalization for 1D images

Level set of 1D image f at level t ∈ R (‘thresholding’)

$$X_t(f) = \{x \in \mathbb{R} : f(x) \geq t\}$$

Theorem (Serra): For upper semicontinuous functions f, the level sets X_t are closed sets that have the following property of monotonicity:

$$t_1 < t_2 \Rightarrow X_{t_1}(f) \supseteq X_{t_2}(f)$$

Umbra is a stack of level sets. More formally:
Umbra U(f) of f(x), x ∈ $\mathbb{R}$, is a closed set (if it exists) comprising all points in $\mathbb{R}^2$ located on the surface defined by f or below it.

Example

The 1D monochrome image:

```
f = [2, 0, 4, 6, 6, 2, 4, 5]
```

Its umbra U(f):

7								
6				1	1			
5				1	1			1
4			1	1	1		1	1
3			1	1	1		1	1
2	1		1	1	1	1	1	1
1	1		1	1	1	1	1	1
0	1	1	1	1	1	1	1	1

The left column shows the threshold *t*.

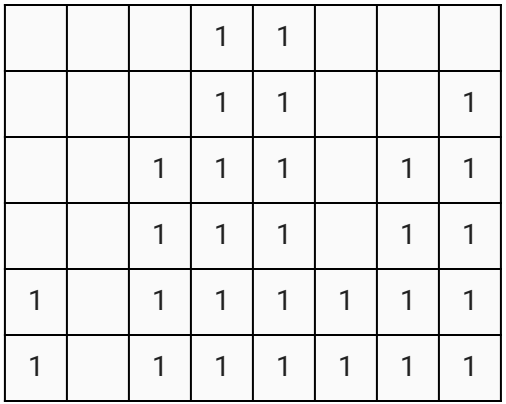
Example

The space of 1D monochrome images

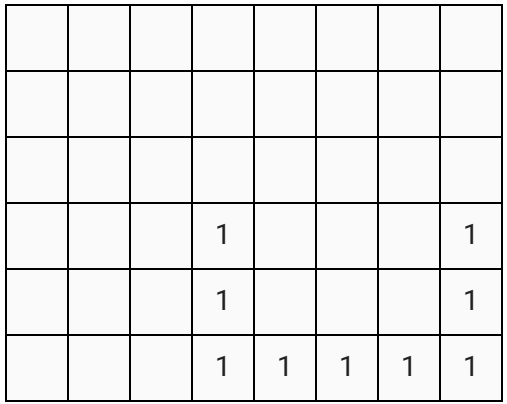
```
f = [2, 0, 4, 6, 6, 2, 4, 5]
```

```
g = [0, 0, 0, 3, 1, 1, 1, 3]
```

The space of umbras:

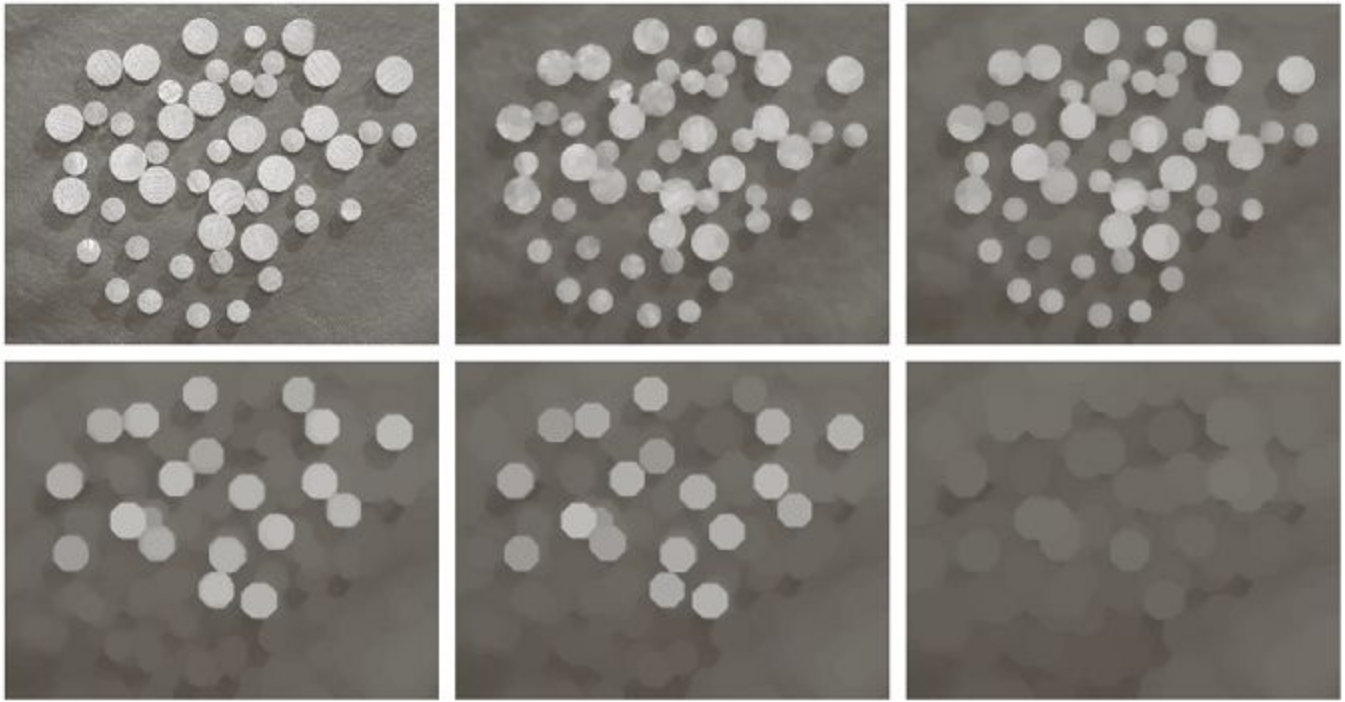


erosion with a square 3x3 element



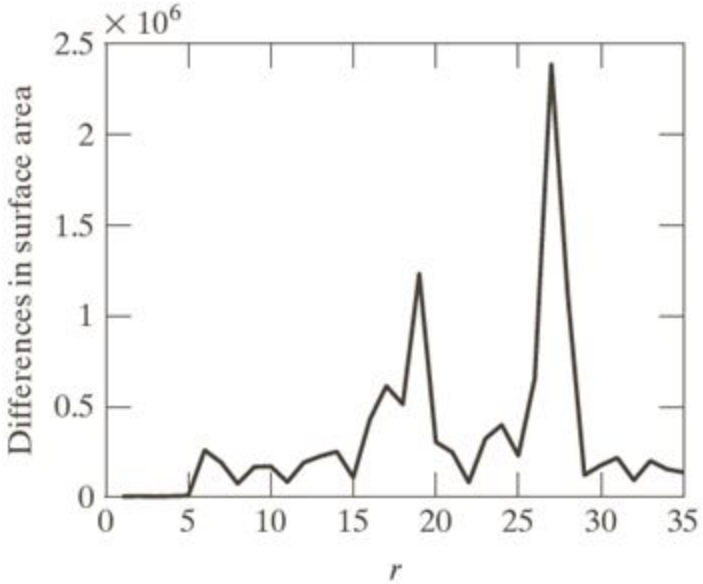
Feature extraction with morphological operators

In order: the input image, the smoothed input image, and the result of opening the image with discs of radius 10, 20, 25 and 30 pixels.



Feature extraction with morphological operators

- Increasing the size of the morphological element causes gradual decrease of brightness.
- We can characterize this process in quantitative manner:



Examples

Impressive capabilities of morphological processing

Other operations often defined in morphology

- contour extraction
- contour filling
- determining connected components
- convex hull
- thinning
- thickening
- skeletonization
- pruning
- ….

Caveat: a lot can be achieved with morphological operators; however, the resulting implementations are not always most efficient.

Image processing in the frequency domain

Fourier series

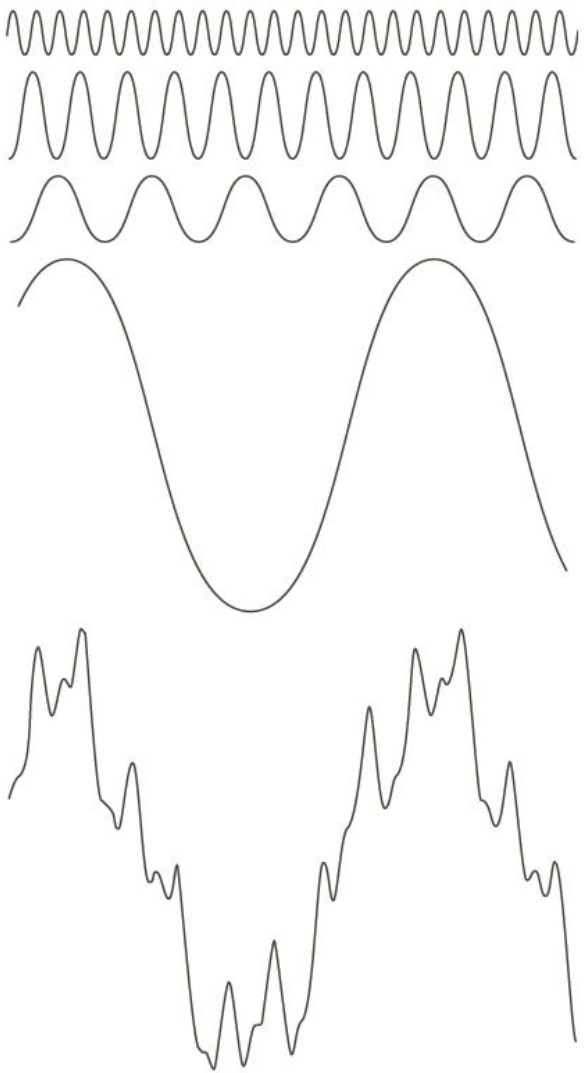
Any complex-valued periodic function with period T can be represented as a sum of sine and cosine components:

$$f(t) = \sum_{n=-\infty}^{\infty} c_n e^{j\frac{2\pi n}{T}t}$$

where

$$c_n = \frac{1}{T}\int_{-T/2}^{T/2} f(t) e^{-j\frac{2\pi n}{T}t} dt$$

for n = 0, ±1, ±2, ±3, …



A few reminders

Polar representation of complex numbers:

$$C = |C|(\cos\theta + j\sin\theta)$$

Exponential form:

$$e^{j\theta} = \cos\theta + j\sin\theta$$

Sketch of proof:

$$\begin{aligned}\frac{df(\theta)}{d\theta} &= \frac{d}{d\theta}(\cos\theta + j\sin\theta) = -\sin\theta + j\cos\theta \\ &= j(j\sin\theta + \cos\theta) = j(\cos\theta + j\sin\theta) = jf(\theta)\end{aligned}$$

Continuous FT of 1D function/signal

FT: For continuous time t:

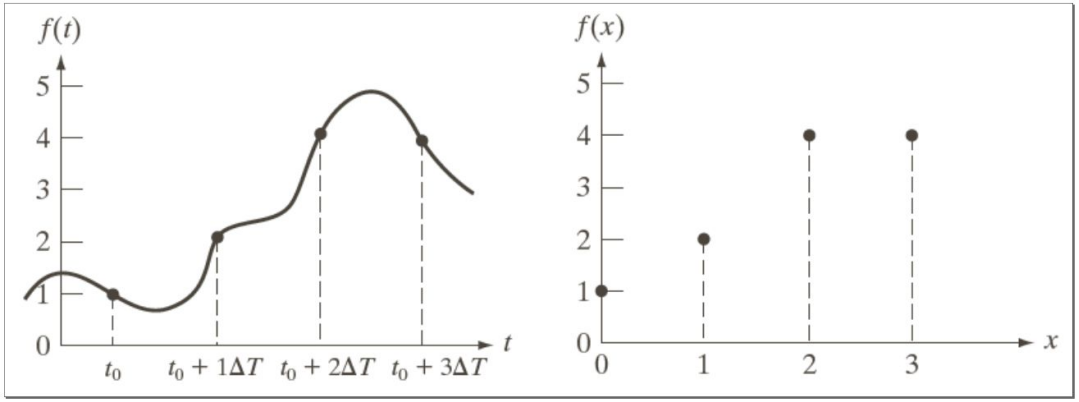
$$F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-j2\pi\mu t} dt$$

where μ is a continuous variable representing frequency. F(μ) measures the 'amount' of frequency μ in f(t).
The inverse FT, i.e. **FT^{-1}**:

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

Transform implies that: FT^{-1}(FT(f)) $\stackrel{\text{def}}{=}$ f

Example: The DFT of a 1D signal



$$F(0) = \sum_{x=0}^{3} f(x) = f(0) + f(1) + f(2) + f(3) = 1 + 2 + 4 + 4 = 11$$

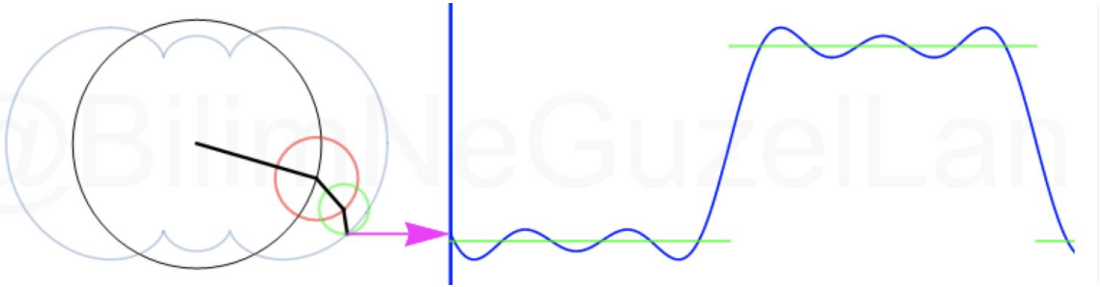
$$F(1) = \sum_{x=0}^{3} f(x) e^{-j2\pi(1)x/4} = 1e^{0} + 2e^{-j\pi/2} + 4e^{-j\pi} + 4e^{-j3\pi/2} = -3 + 2j$$

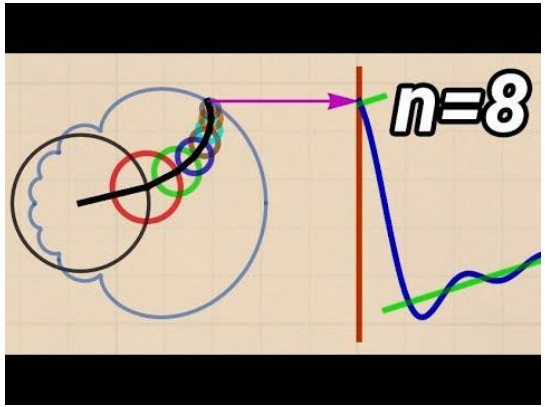
$$F(2) = -(1 + 0j)$$

$$F(3) = -(3 + 2j)$$

Visualization

- Excellent explanation of the essence of Fourier series
  https://bilimneguzellan.net/fuyye-serisi/
  https://youtu.be/ds0cmAV-Yek?t=153





Discrete Fourier transform of 2D signals

For an MxN image f(x,y):

$$F(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(ux/M+vy/N)}$$

where u∈[0,M), v∈[0,N).

Inverse FT:

$$f(x,y) = \frac{1}{MN} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} F(u,v) e^{j2\pi(ux/M+vy/N)}$$

where x∈[0,M), y∈[0,N).

Notes on DFT

- The coefficients F(u,v) are in general complex, even if the signal is real-valued.
  - We can apply DFT to complex images ‘for free’.
- What is the interpretation of F(0,0)?

Technical implementation

- Implementing the DFT directly according to the definition leads to quadratic complexity $O(n^2)$, where n is the number of samples of the signal
  - A signal of length n leads to spectrum with n coefficients.
    - Calculation of each coefficient involves all elements of the signal.
  - This holds no matter what signal's dimensionality: 1D, 2D, or kD
- Fast Fourier Transform (FFT): A *class* of efficient algorithms that allow calculating the DFT in $O(n\log n)$ time.
  - All based on the idea of factorization of n.
- The best-known variant: Cooley–Tukey algorithm
  - The principle proposed by C.F. Gauss in 1805!
  - Factor n into a product $n=n_1n_2$
  - Typically, $n_1=n_2=n$

Things worth remembering about the FT

- Complex coefficients allow capturing both frequency and phase.
- Frequencies can be negative; the signs encode the ‘direction’ of change.
- If the signal is symmetric, i.e. f(x) = f(-x), its FT is real-valued.
- FT is additive: $FT(x_1+x_2) = FT(x_1) + FT(x_2)$
- Uncertainty principle:
  - Functions that are localized in the time domain (short support) have Fourier transforms that are spread out across the frequency domain
  - and vice versa.
  - Critical case: FT of a Gaussian hat is a Gaussian hat.
- Convolution theorem:
  - Convolution in the primary (e.g. time) domain corresponds to element-wise multiplication of transforms (spectra). In other words, if the following holds in the time domain:

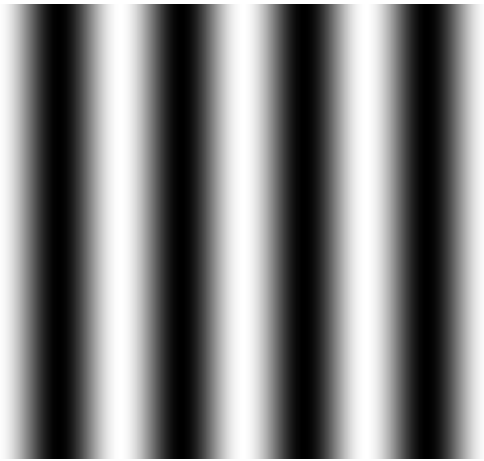
$$h(x) = (f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)\,dy,$$

    then the spectrum of h is the elementwise product of the spectra of f and g.

Examples

In the slides that follow, we apply FT to synthetic images where pixel brightness is determined by a [co]sinusoidal periodicity:

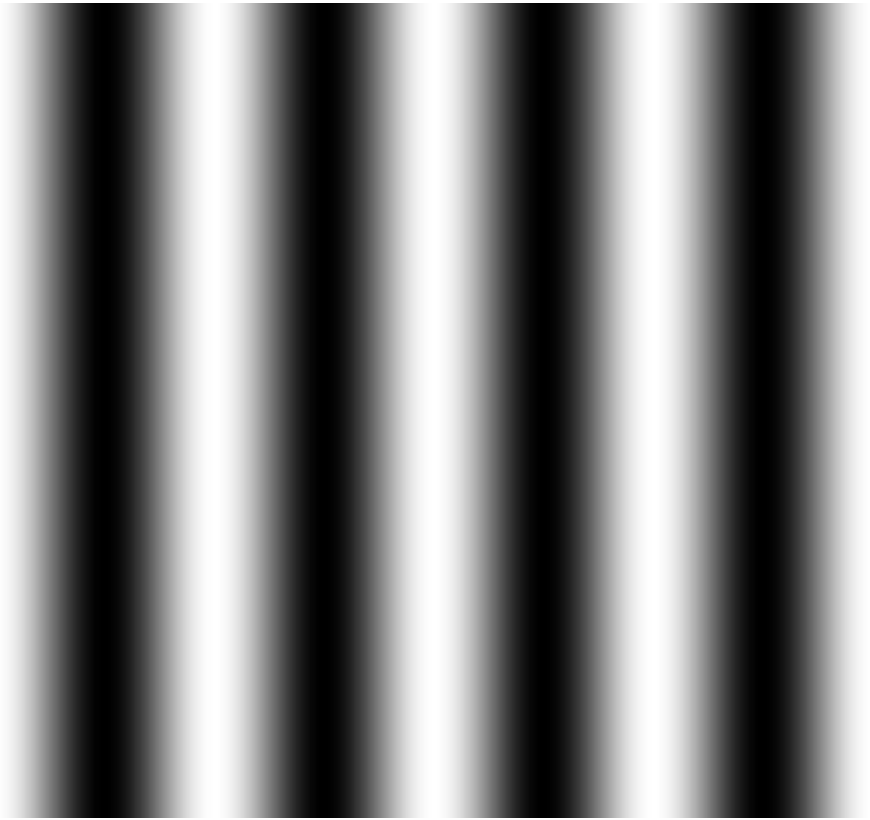
- Range: [-1, 1]
  - Implication: average image brightness = 0
- Number of periods, e.g. 4

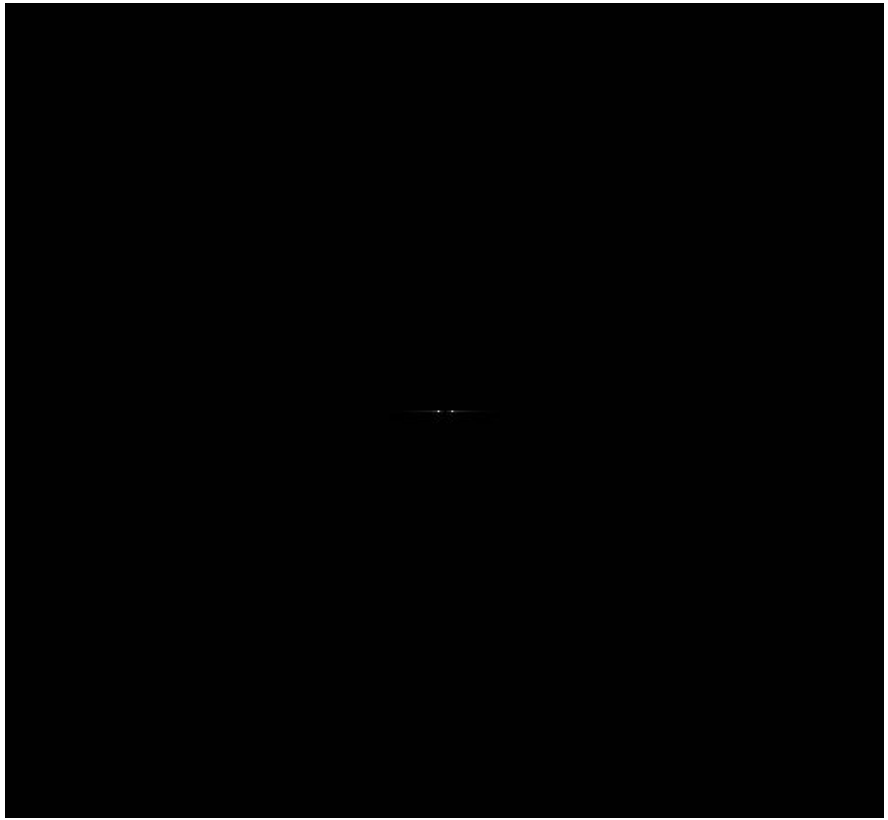


Notice:

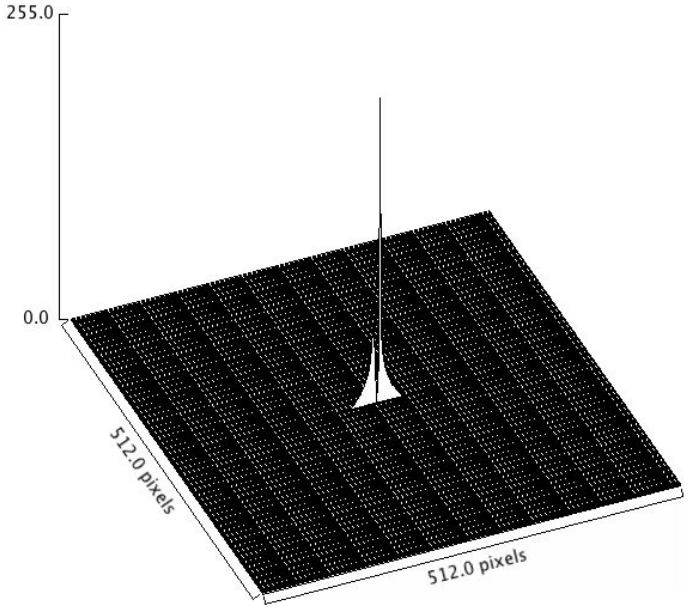
- The outcome of the DFT is a matrix of coefficients, which can be also presented as an image.
- The coefficients are in general complex.
  - We usually present the power of the spectrum, i.e. the module of the complex number (lengths of vectors).

Left: image, Right: FT



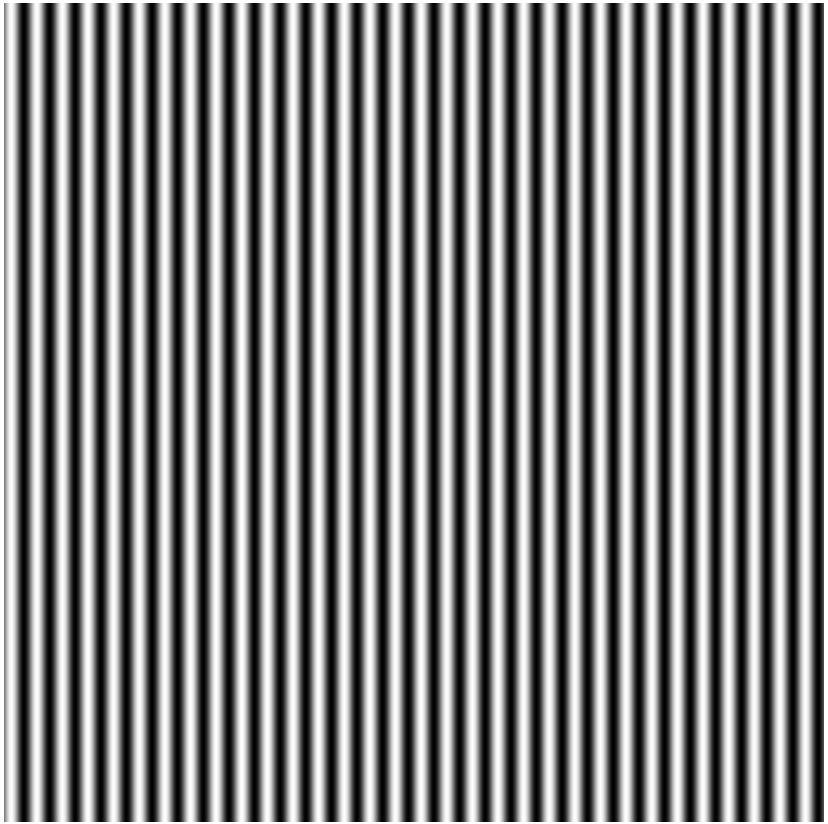


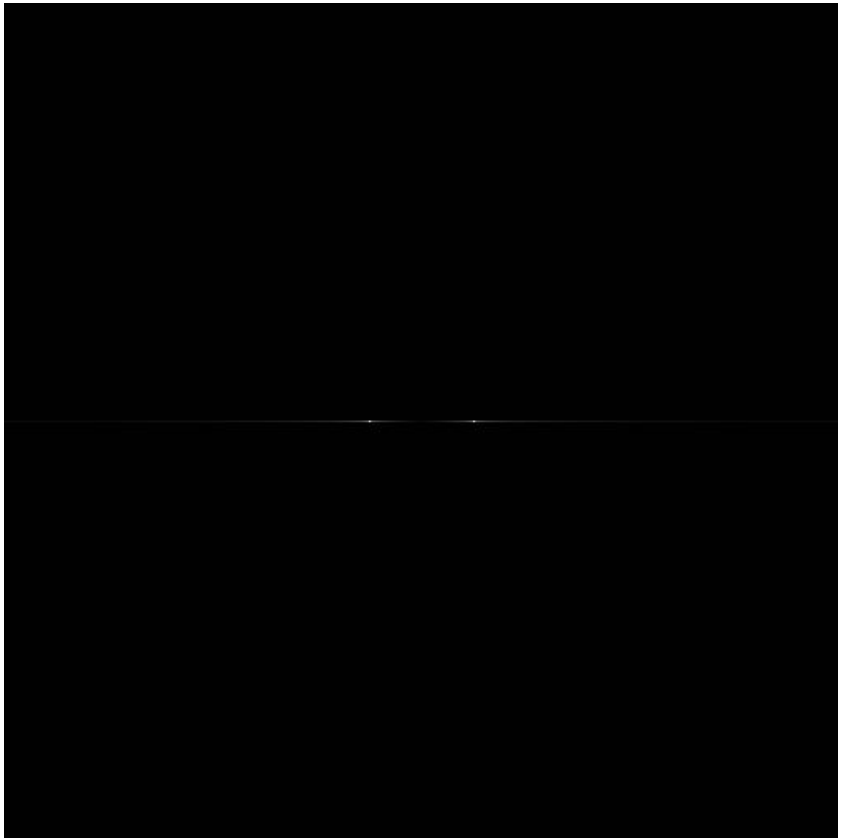
A closer look at the spectrum



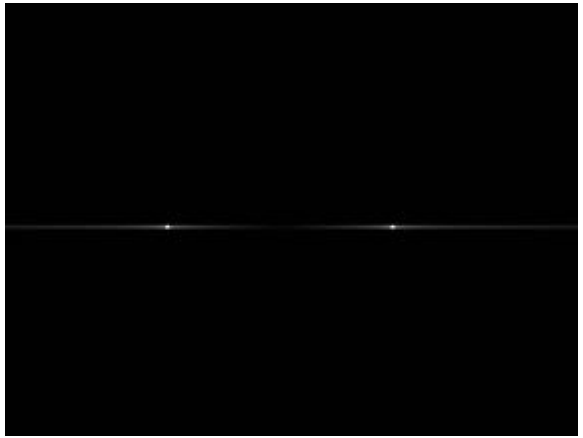
- In perfect, continuous realization, there would be only two peaks in the spectrum.
- The origin of the spectrum corresponds to the constant component F(0,0)

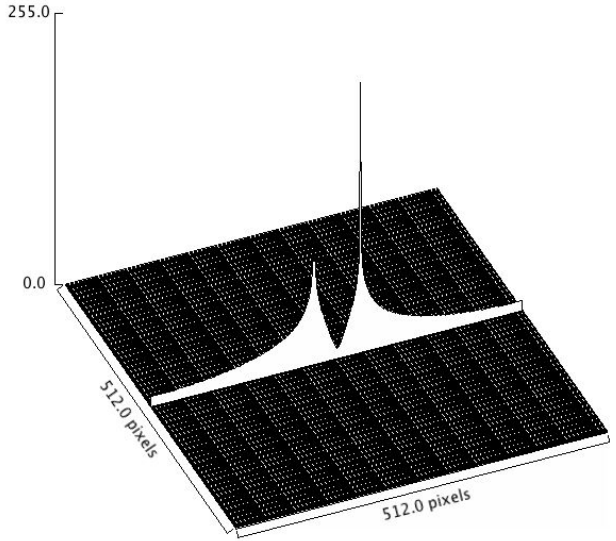
Left: image, Right: FT



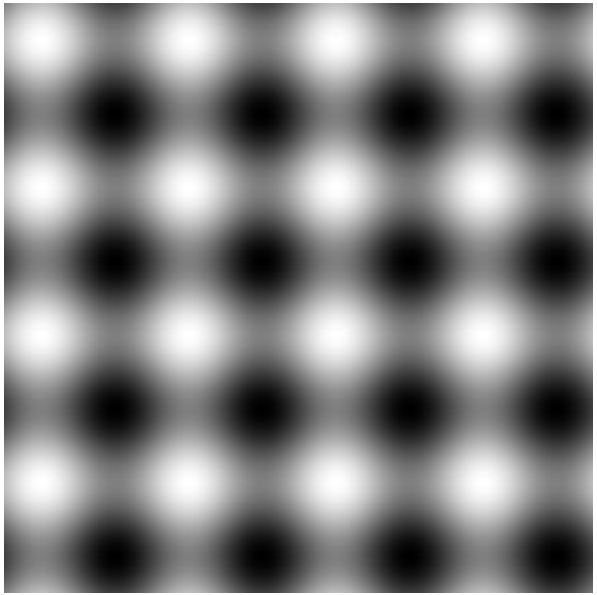


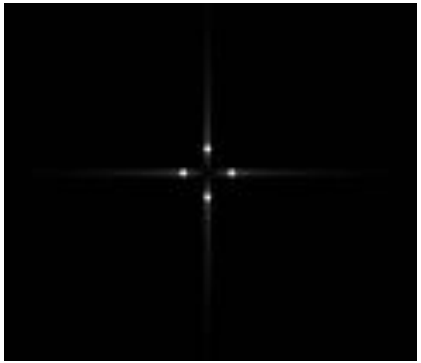
A closer look





For a composite signal:

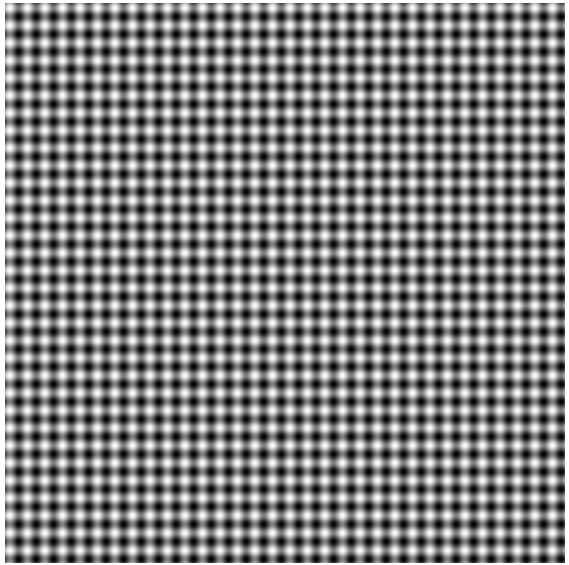


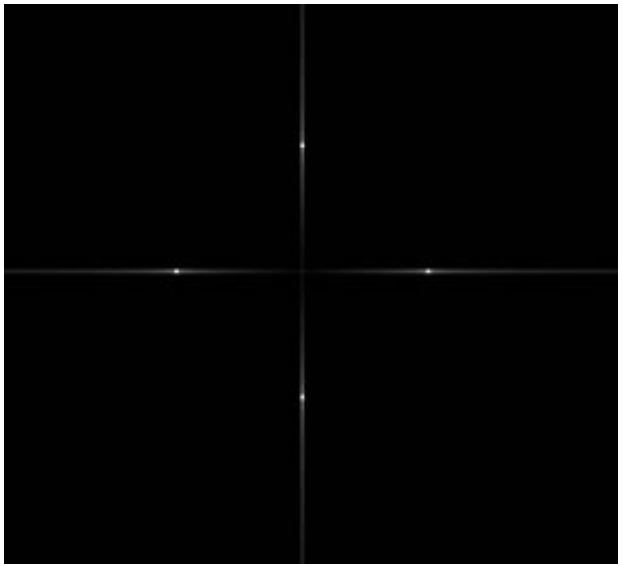


This is due to FT being additive: the FT of a sum of signal is the sum of FTs of those signals.

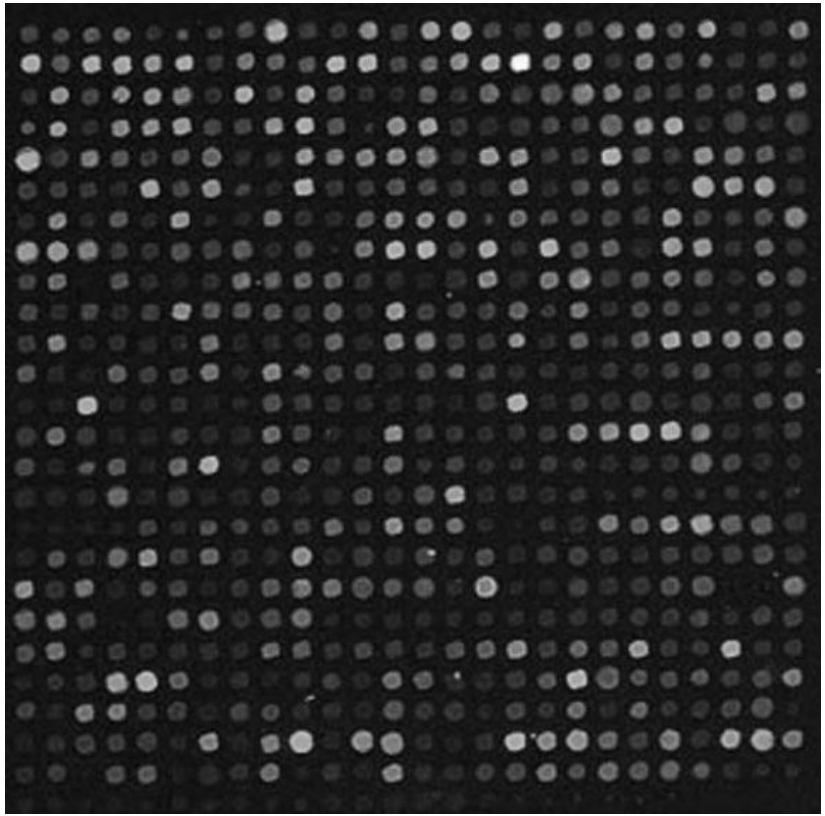
$$F(\mu) = \int_{-\infty}^{\infty} f(t) e^{-j2\pi\mu t} dt$$

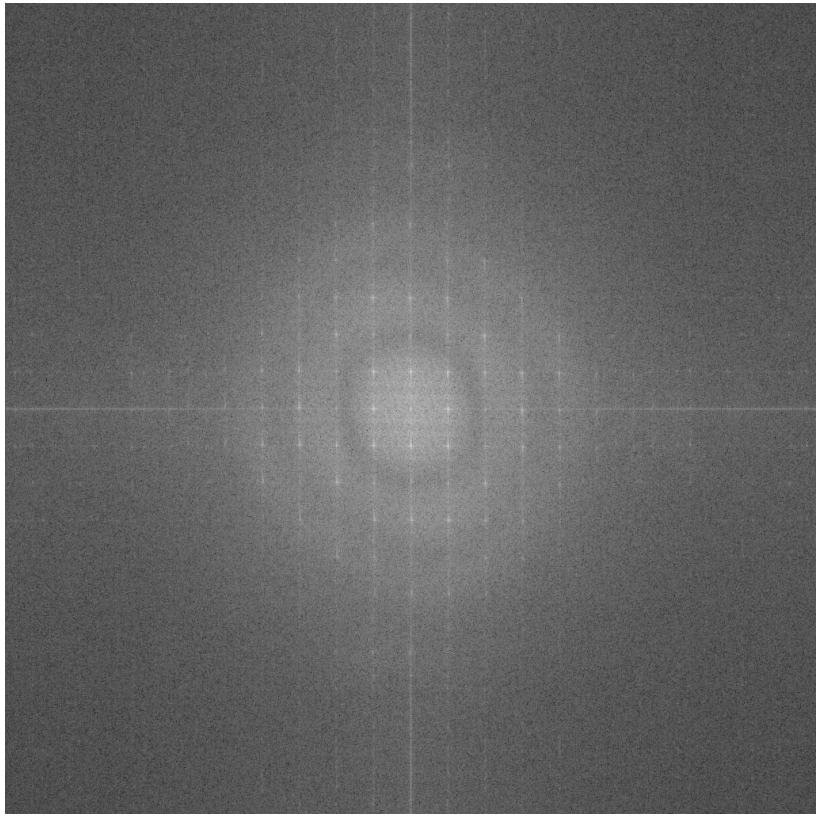
Superposition of higher frequencies



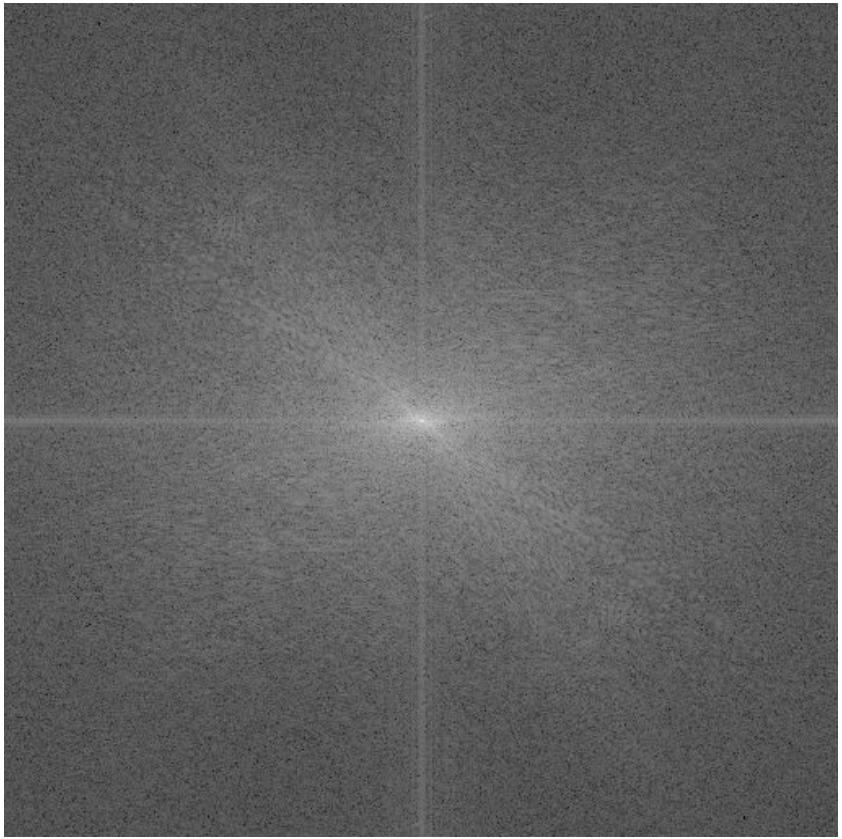


Pseudo-periodic image

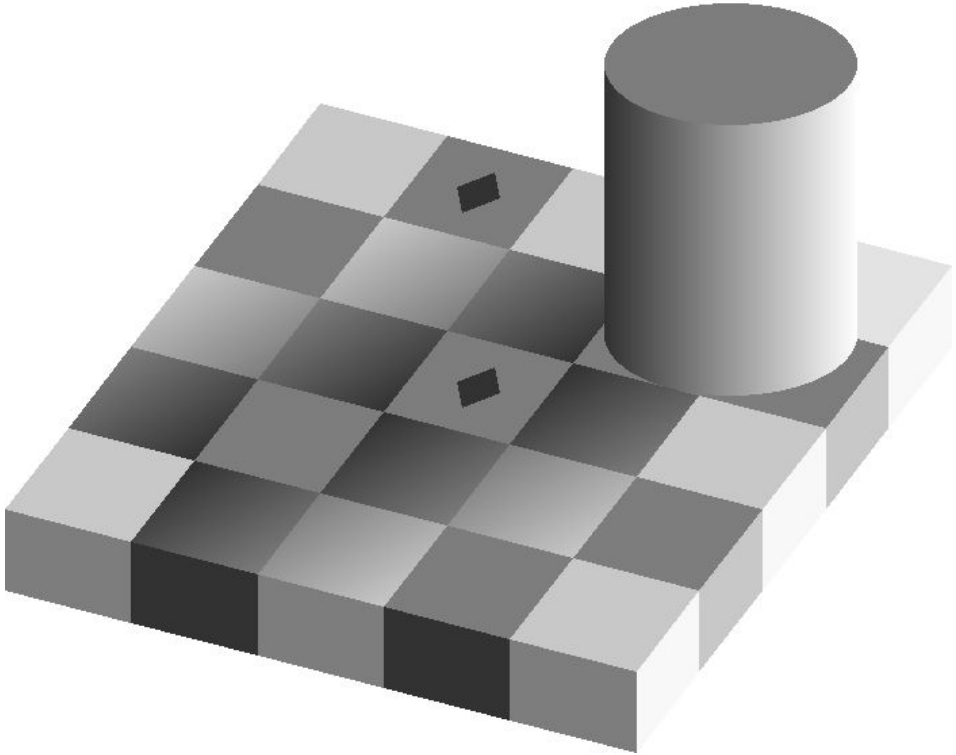


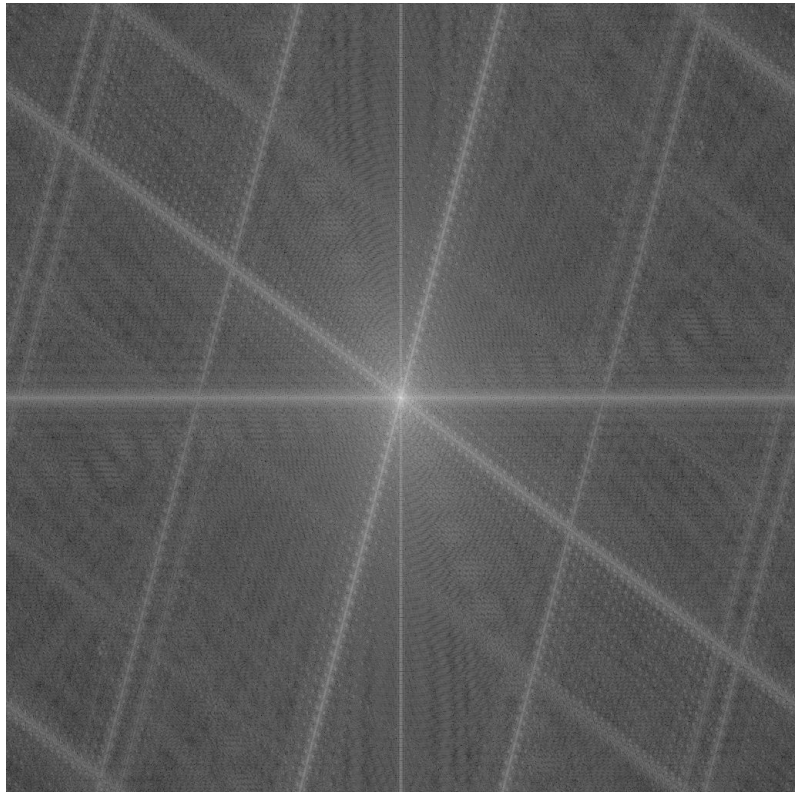


FT applied to a ‘natural’ image

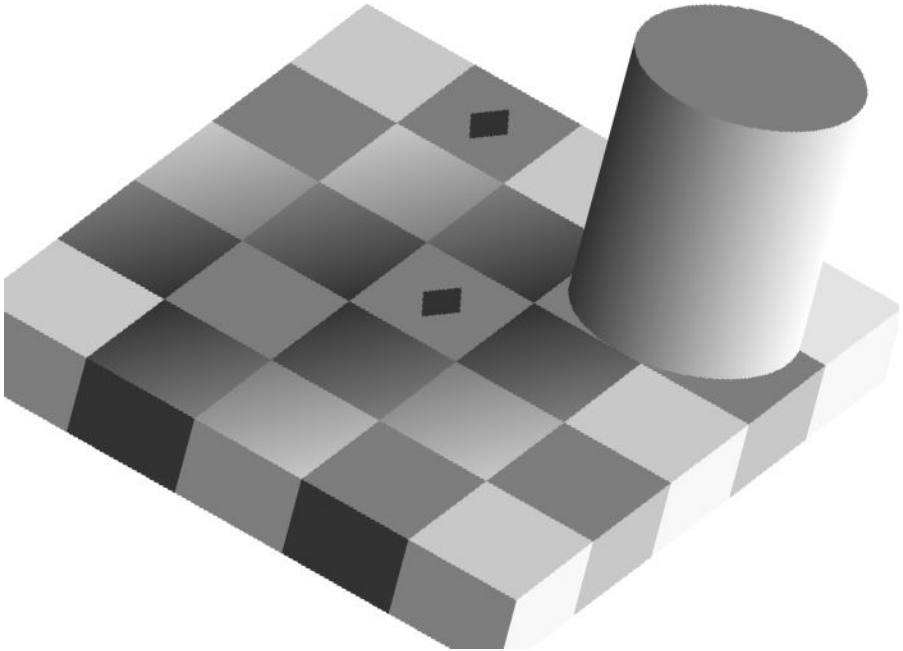


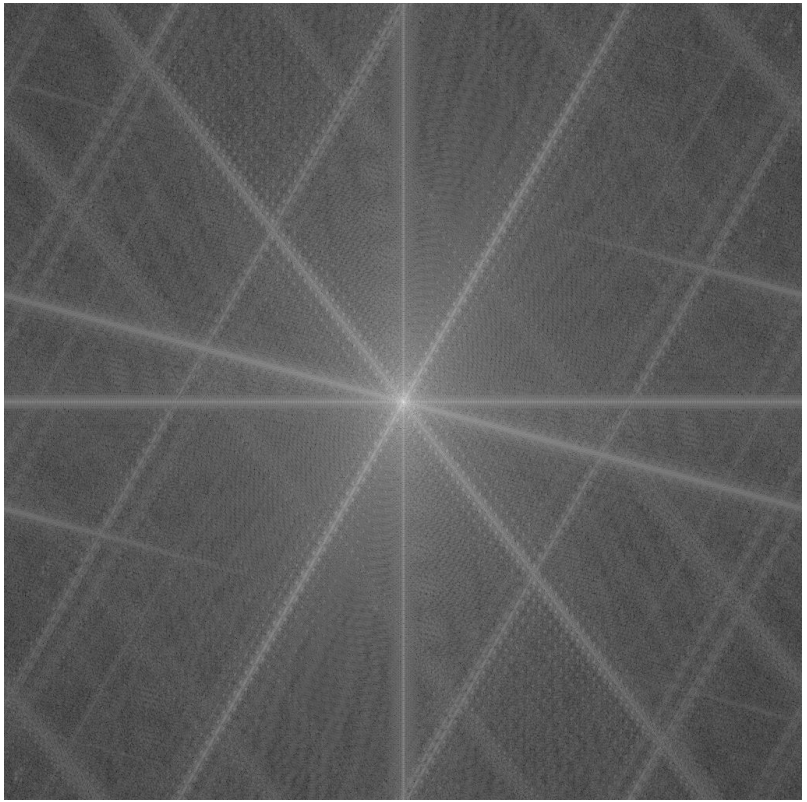
Response of FT to linear features



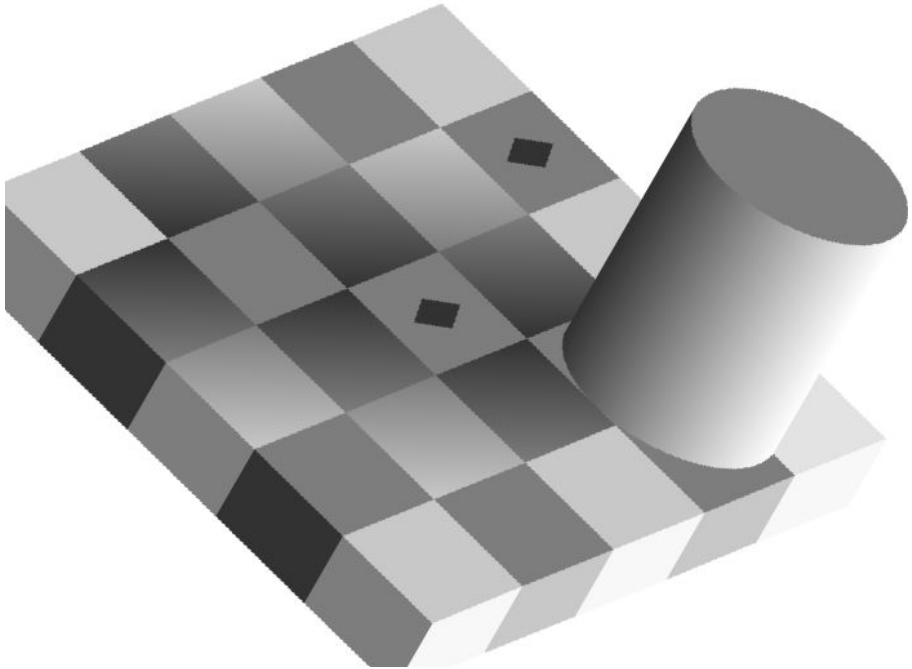


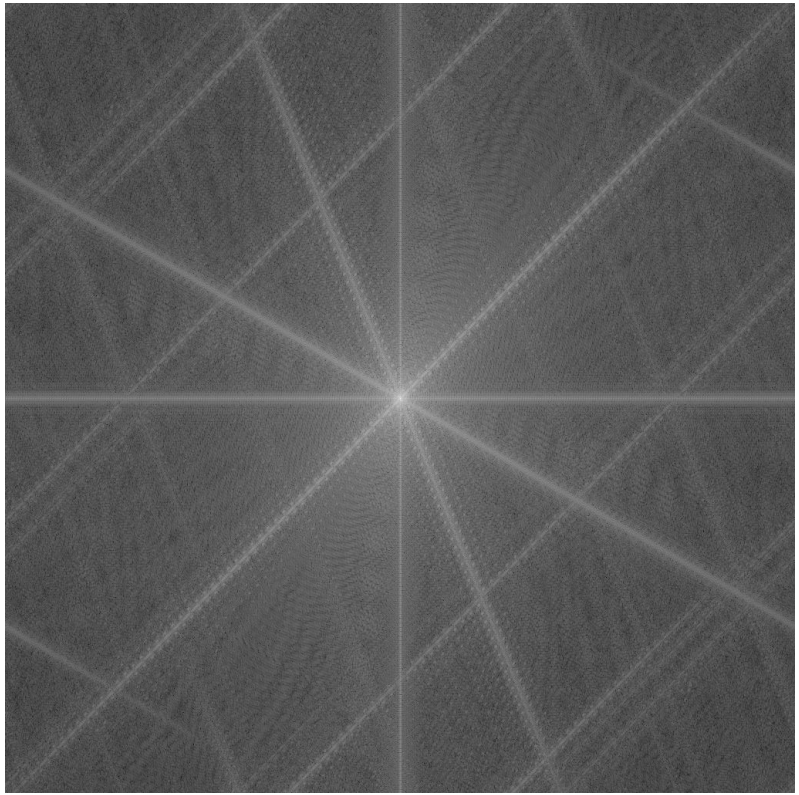
Impact of rotation (15 degrees clockwise)



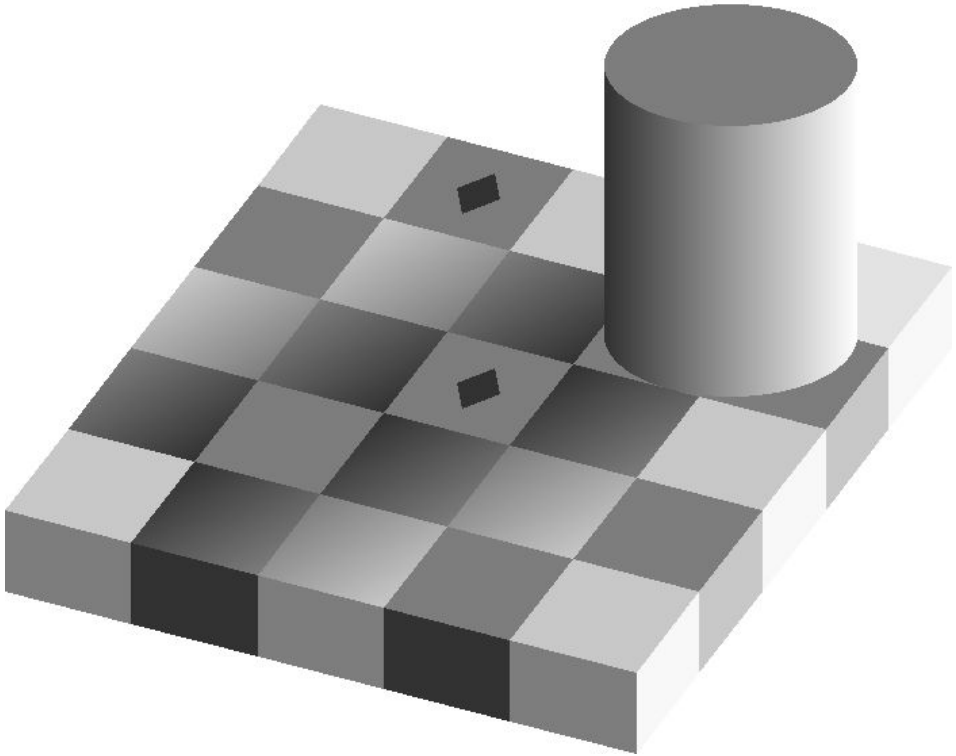


Impact of rotation (30 degrees clockwise)

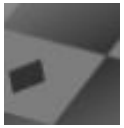




Interlude



Which of the two squares marked with diamonds is brighter?

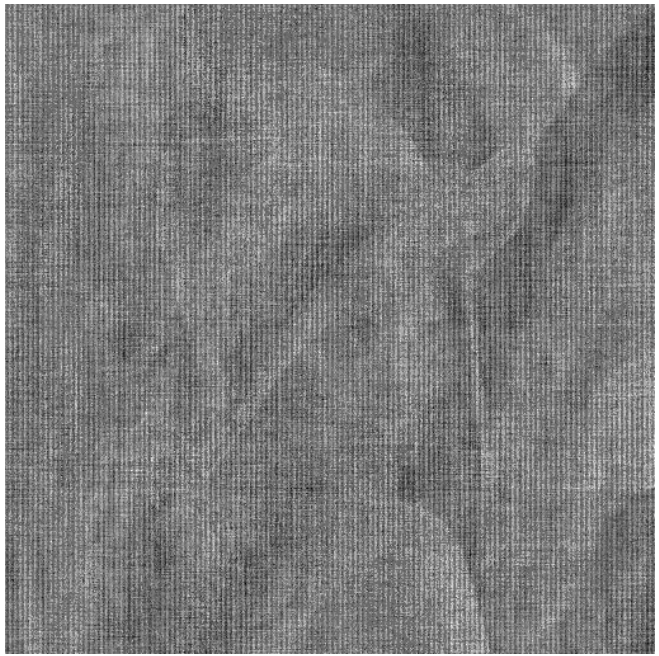


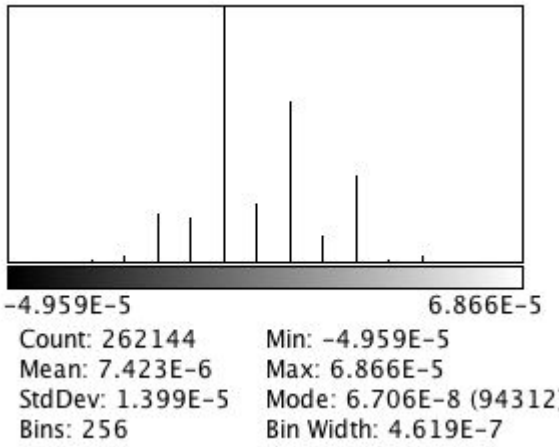
Quality of reconstruction

Recall that in theory $FT^{-1}(FT(f)) \stackrel{def}{=} f$

How does this hold in practice?

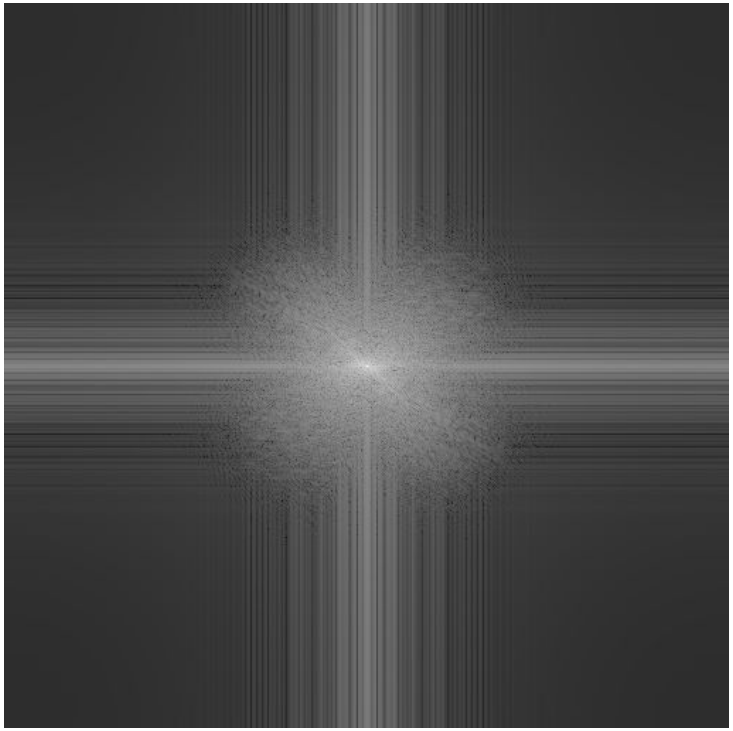
This image presents $|FT^{-1}(FT(f)) - f|$ for f=Lena; its histogram on the right.





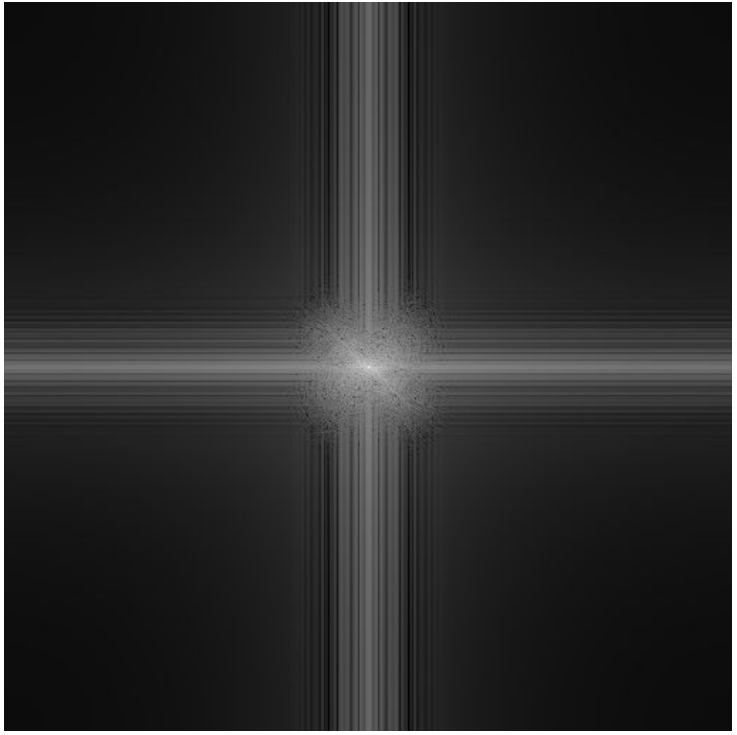
Relation between image sharpness and its spectrum

Lena image blurred with Gaussian filter (σ=2.0)



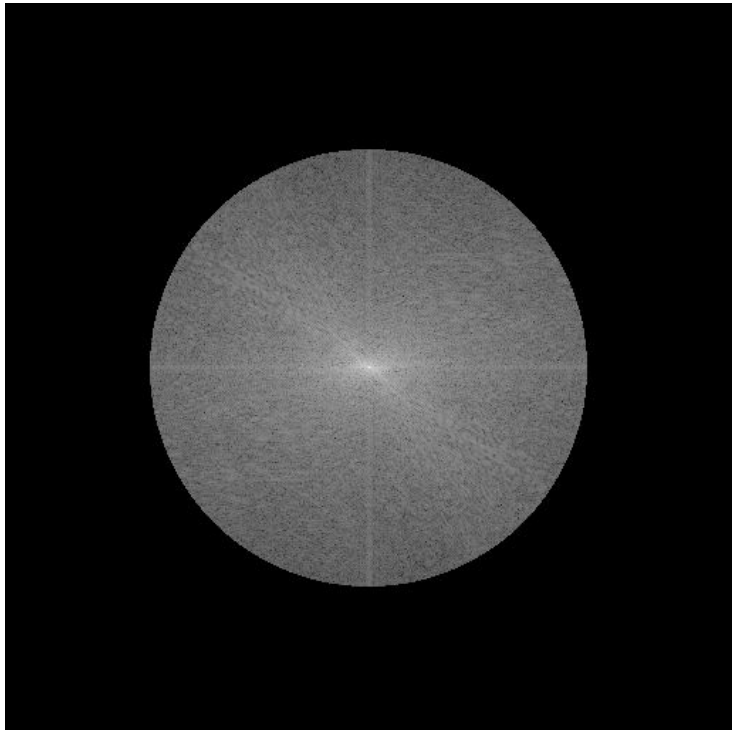
Relation between image sharpness and its spectrum

Lena image blurred with Gaussian filter (σ=2.0)

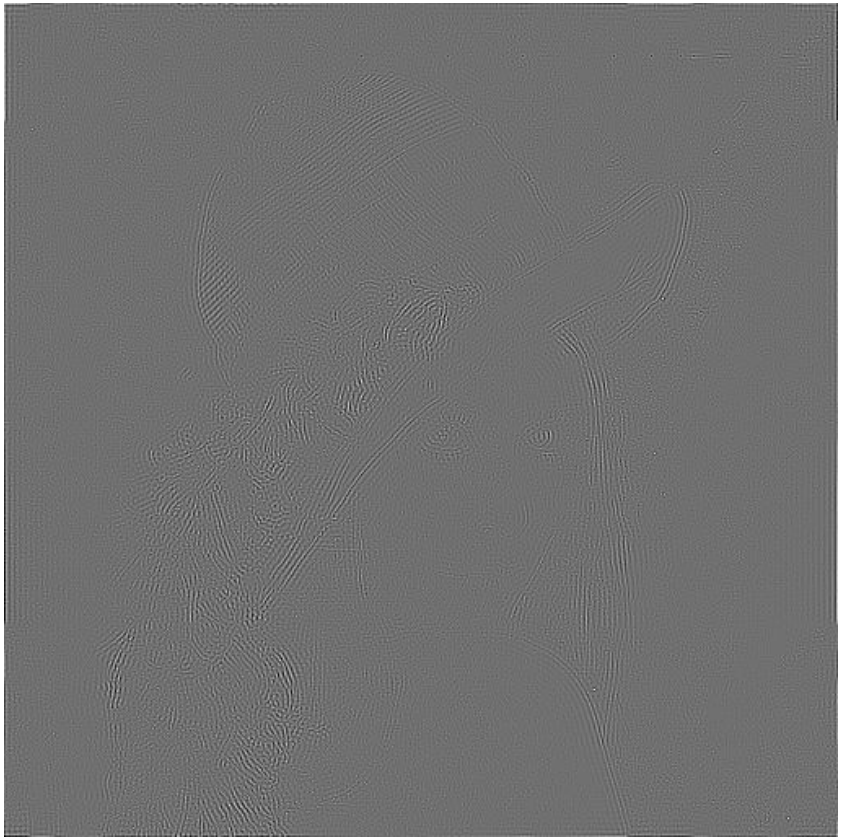


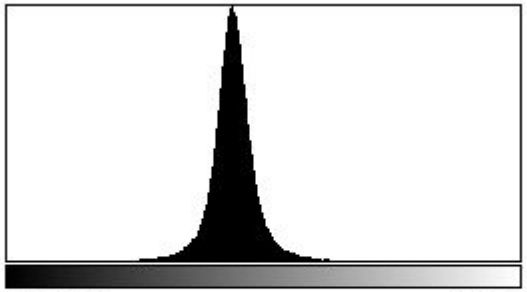
Reconstruction from truncated spectrum

r=150



Quality of reconstruction

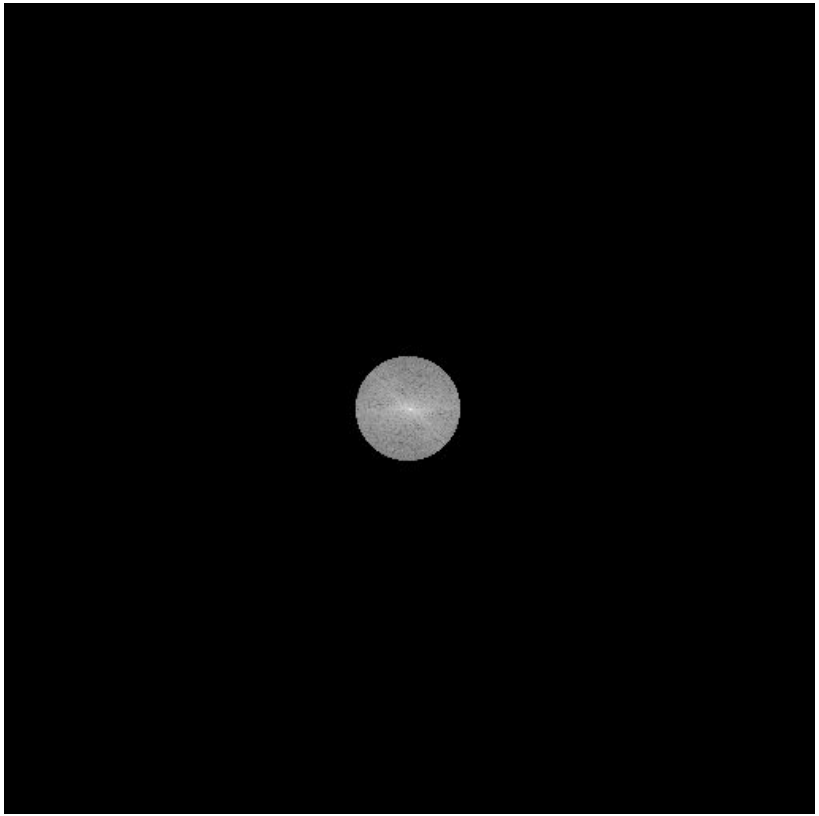




Count: 262144	Min: -34.279
Mean: 6.934E-6	Max: 43.460
StdDev: 3.839	Mode: -0.116 (12703)
Bins: 256	Bin Width: 0.304

Reconstruction from truncated spectrum

r=64



Additional materials

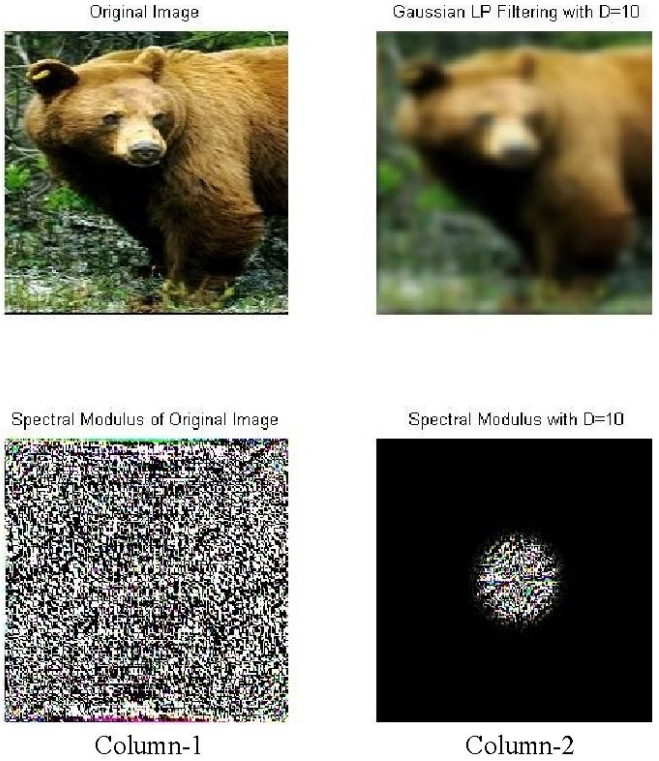
- Similar illustration to that provided in the previous slides: https://homepages.inf.ed.ac.uk/rbf/HIPR2/freqfilt.htm

Fourier Transform for color images

One option: use hyper-complex numbers (a.k.a. quaternions).

- Generalize the FT to quaternions.
- Represent the RGB channels and brightness as quaternions.
- Perform transform

On the right: the effect of lowpass (LP) filtering of the image on the resulting spectrum.



Vikas R. Dubey, Quaternion Fourier Transform for Colour Images, International Journal of Computer Science and Information Technologies, Vol. 5 (3) , 2014, 4411-4416

End of module