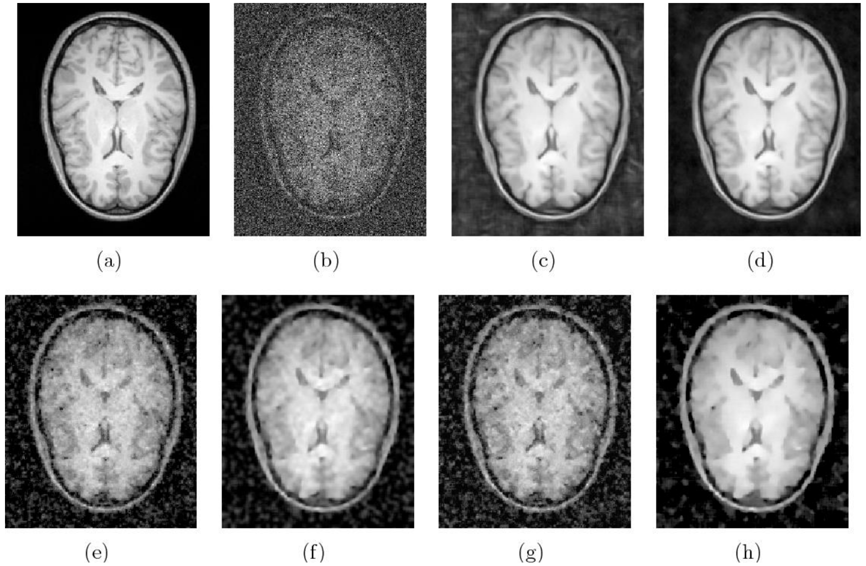
# Image processing

# References

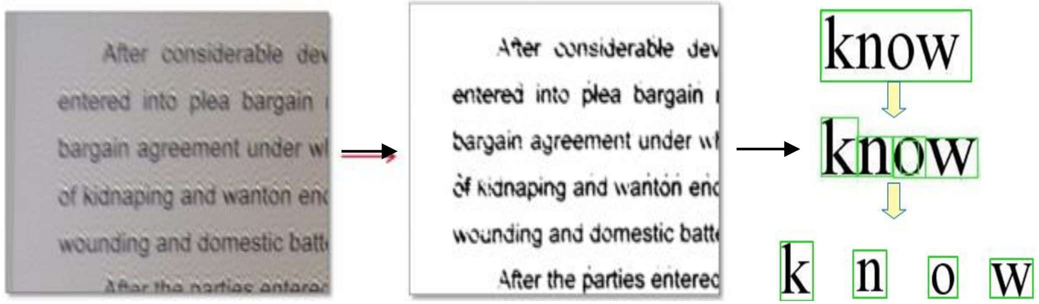
- http://szeliski.org/Book/
- http://www.cs.cornell.edu/courses/cs5670/2019sp/lectures/lectures.html
- http://www.cs.cmu.edu/~16385/

# Some motivation


Art
(Photoshop image correction)




Medicine
(MRI denoising)




Robotics
(OCR – optical character recognition)

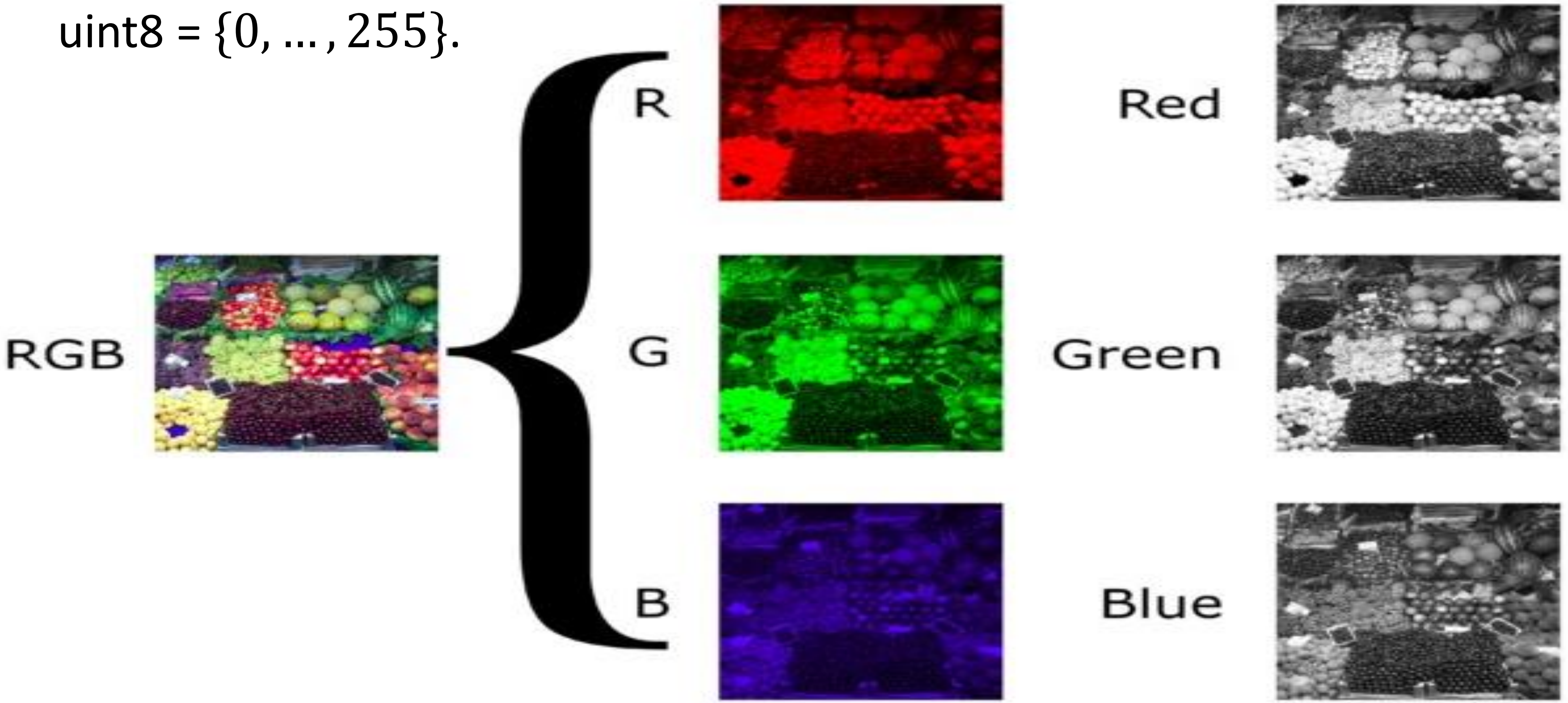
Agriculture
(color ripeness detection)

# contents

- **Image representation**
- Pixel-wise operations
- Noise and filtering
- Frequency representation
- Decimation
- Interpolation
- Morphology operators
- Connected components

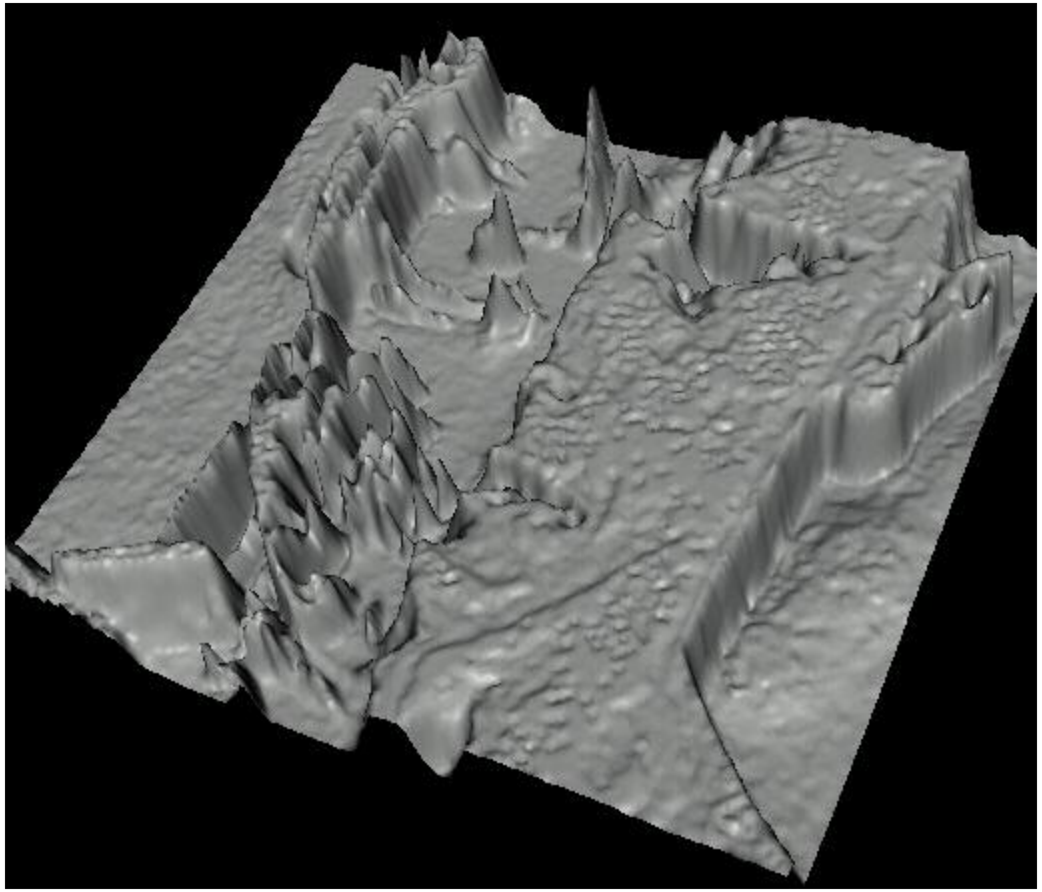
# Image representation

- We cane think of image as a 3d matrix of discrete RGB values.
- The values mark the intensity of each color channel and are usually of type uint8 = $\{0, \ldots, 255\}$.



# Image representation

- We can also think of an image as a function $f(x, y)$ .



# contents

- Image representation
- **Pixel-wise operations**
- Noise and filtering
- Frequency representation
- Decimation
- Interpolation
- Morphology operators
- Connected components

# Pixel-wise operators

- Pixel-wise operators, or point operators, are defined as such that each output pixel’s value depends on only the corresponding input pixel value.

# Pixel-wise operators

original

$x$

darken

lower contrast

Gamma compression

invert

lighten

raise contrast (?)

Gamma expansion

# Pixel-wise operators

original

$x$

darken

lower contrast

Gamma compression

invert

$255 - x$

lighten

raise contrast (?)

Gamma expansion

# Pixel-wise operators

original

$x$

darken

$x - 128$

lower contrast

Gamma compression

invert

$255 - x$

lighten

raise contrast (?)

Gamma expansion

# Pixel-wise operators

original

$x$

darken

$x - 128$

lower contrast

Gamma compression

invert

$255 - x$

lighten

$x + 128$

raise contrast (?)

Gamma expansion

# Pixel-wise operators

original

$$x$$

darken

$$x - 128$$

lower contrast

$$\frac{x}{2}$$

Gamma compression

invert

$$255 - x$$

lighten

$$x + 128$$

raise contrast (?)

Gamma expansion

# Pixel-wise operators

original

$x$

darken

$x - 128$

lower contrast

$\frac{x}{2}$

Gamma compression

invert

$255 - x$

lighten

$x + 128$

raise contrast (?)

$x \times 2$

Gamma expansion

# Contrast

- **Contrast** in visual perception is the difference in appearance of two or more parts of a field seen.
- The human visual system is more sensitive to contrast than absolute luminance;
- **Contrast ratio**, or **dynamic range**, is the ratio between the largest and smallest values of the image or :

$$CR = \frac{V_{max}}{V_{min}}$$

# Pixel-wise operators

original

$x$

darken

$x - 128$

lower contrast

$\frac{x}{2}$

Gamma compression

invert

$255 - x$

lighten

$x + 128$

raise contrast (?)

$x \times 2$

Gamma expansion

# Pixel-wise operators

original

$$x$$

darken

$$x - 128$$

lower contrast

$$\frac{x}{2}$$

Gamma compression

$$\left(\frac{x}{255}\right)^{1/3} \times 255$$

invert

$$255 - x$$

lighten

$$x + 128$$

raise contrast (?)

$$x \times 2$$

Gamma expansion

# Pixel-wise operators

original

$$x$$

darken

$$x - 128$$

lower contrast

$$\frac{x}{2}$$

Gamma compression

$$\left(\frac{x}{255}\right)^{1/3} \times 255$$

invert

$$255 - x$$

lighten

$$x + 128$$

raise contrast (?)

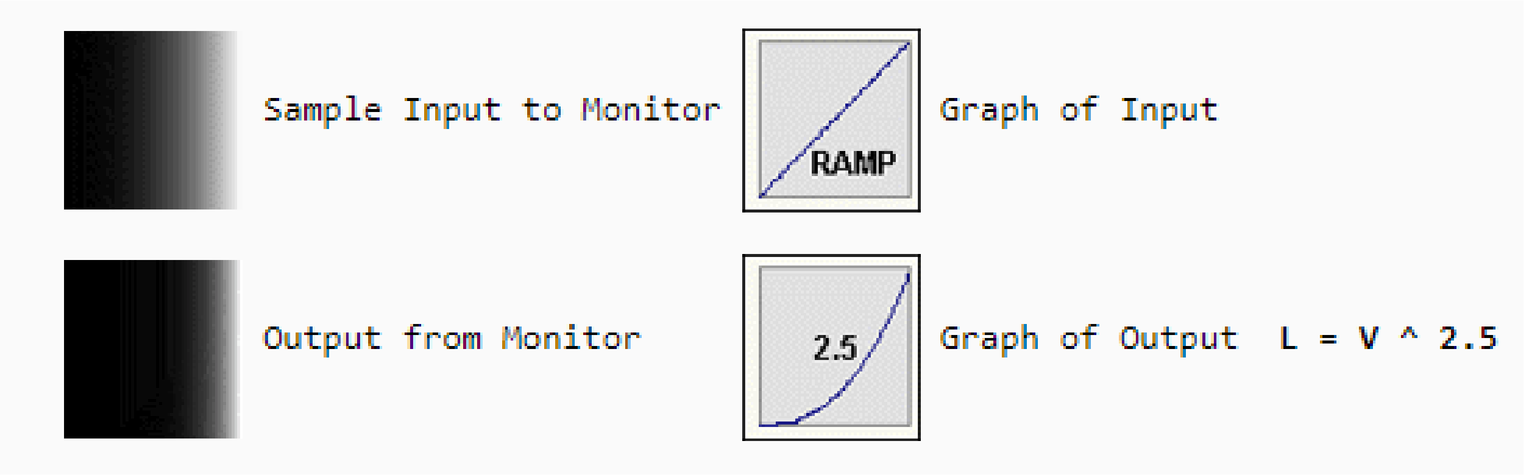
$$x \times 2$$

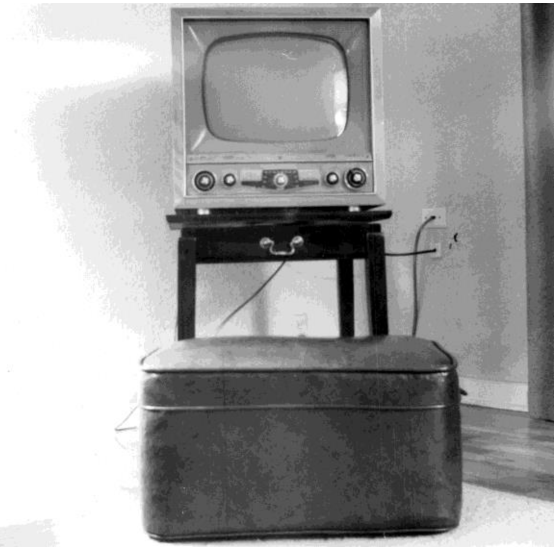
Gamma expansion

$$\left(\frac{x}{255}\right)^{2} \times 255$$

# Gamma correction

- Originally, Due to non-linearities in the old CRT televisions, intensities was seen different then they are.



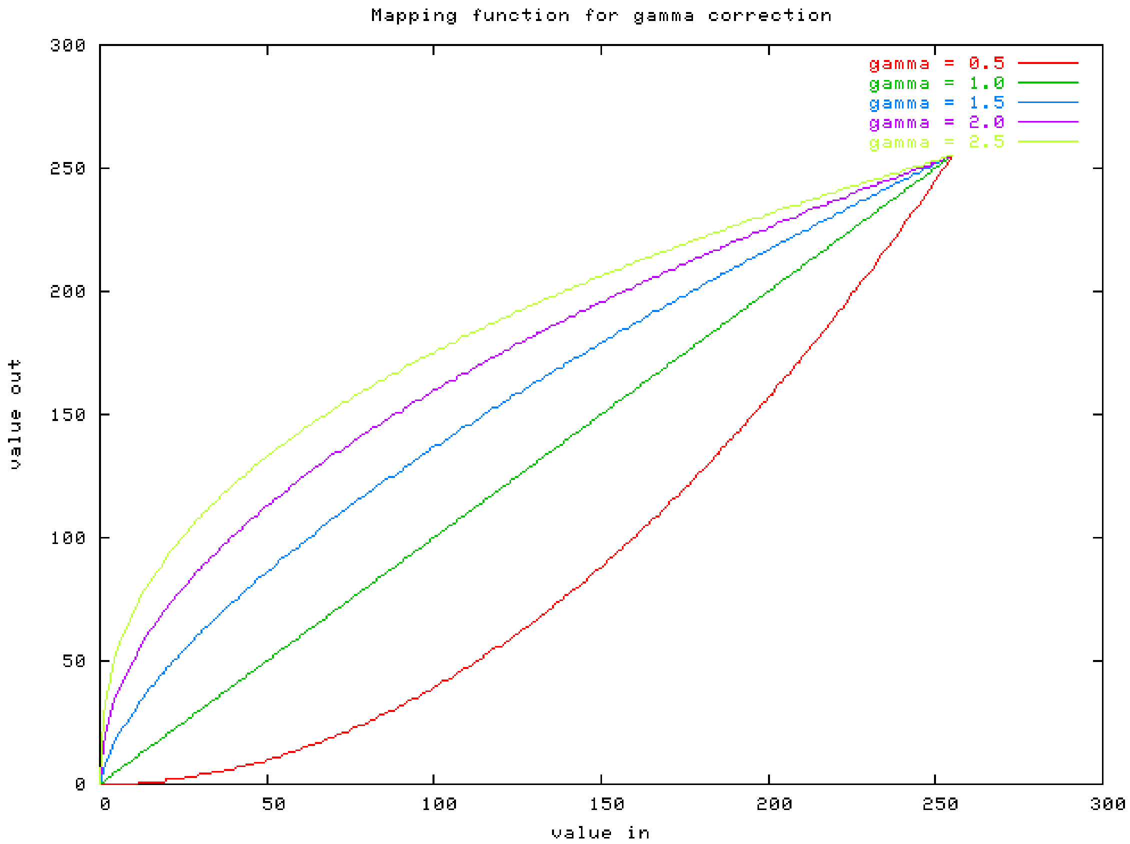



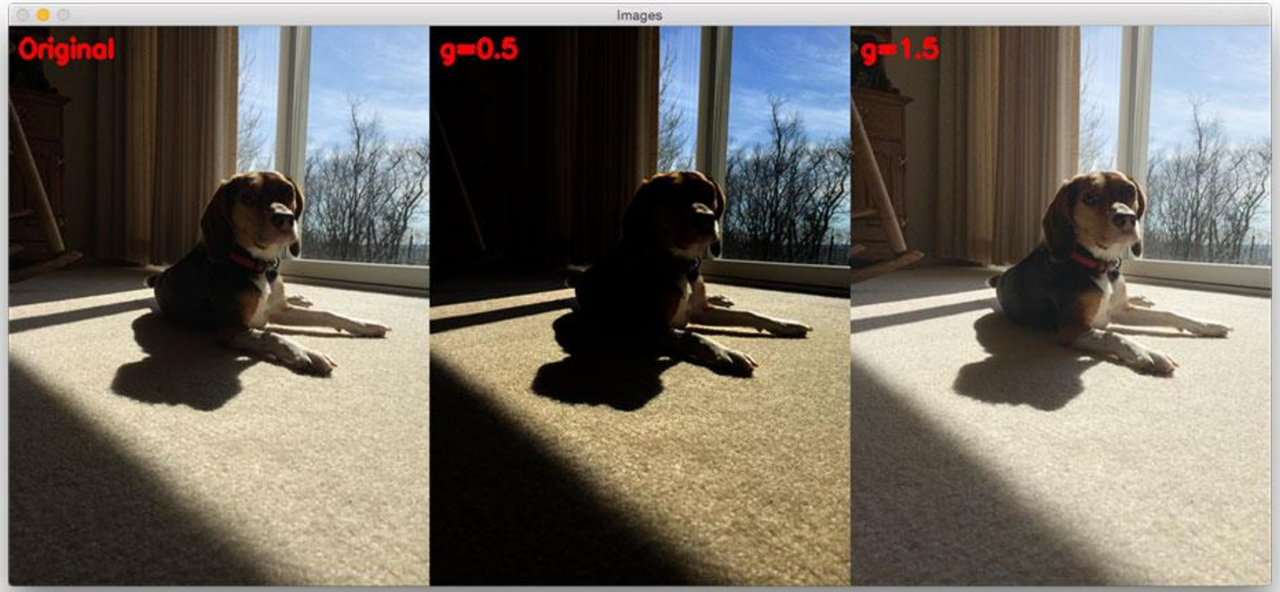
# Gamma correction

- To correct this non linear transformation, gamma correction was done:

$$V_{out} = V_{in}^{\gamma}$$

- This is, of course, also applicable for image enhancements.



# contents

- Image representation
- Pixel-wise operations
- **Noise and filtering**
- Frequency representation
- Decimation
- Interpolation
- Morphology operators
- Connected components

# Gaussian Noise

- Gaussian noise is an additive noise that can appear in images due to the system electrical circuitry.
- This noise is independent of signal strength and independent at each pixel (IID).

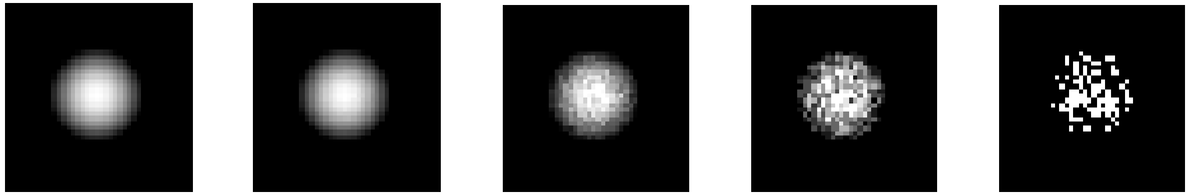
$$p(z) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

# Salt & Pepper noise

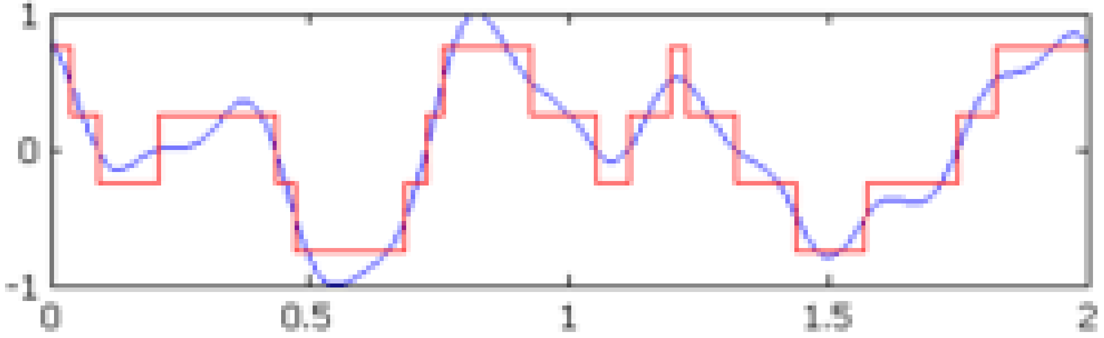
- Noise that can be caused by analog-to-digital converter errors, bit errors in transmission, etc.
- This noise is **not** additive to the signal strength (a replacement of original value with noise value).
- This noise is independent of signal strength and independent at each pixel.

# And some more noise

- **Shot noise** - caused by statistical quantum fluctuations, that is, variation in the number of photons sensed at a given exposure level in the darker parts of an image (where there are just few photons that enter each pixel “bin”). Modeled as Poisson noise.



- **Quantization noise** – caused by quantizing the pixels of a sensed image to several discrete levels (analog to digital conversion).



# Noise reduction with LTI filters

- **linear filtering operators**, which involve weighted combinations of pixels in small neighborhood.
- The combination is determined by the filter's *kernel*.
- The same kernel is *shifted* to all pixel locations so that all pixels use the same linear combination of their neighbors.
- That's why it's called **linear shift-invariance** (LTI) filter.

# Convolution

- Works on LTI filters.
- Let $h$ be the image, $f$ be the kernel (of size 2*k*+1 x 2*k*+1), and $g$ be the output image:

$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] f[i-u, j-v]$$

  – **Note:** by definition, this operator flips the kernel both horizontally and vertically.
- This operation is called **convolution operator** and is more compactly noted as:

$$g = h * f$$

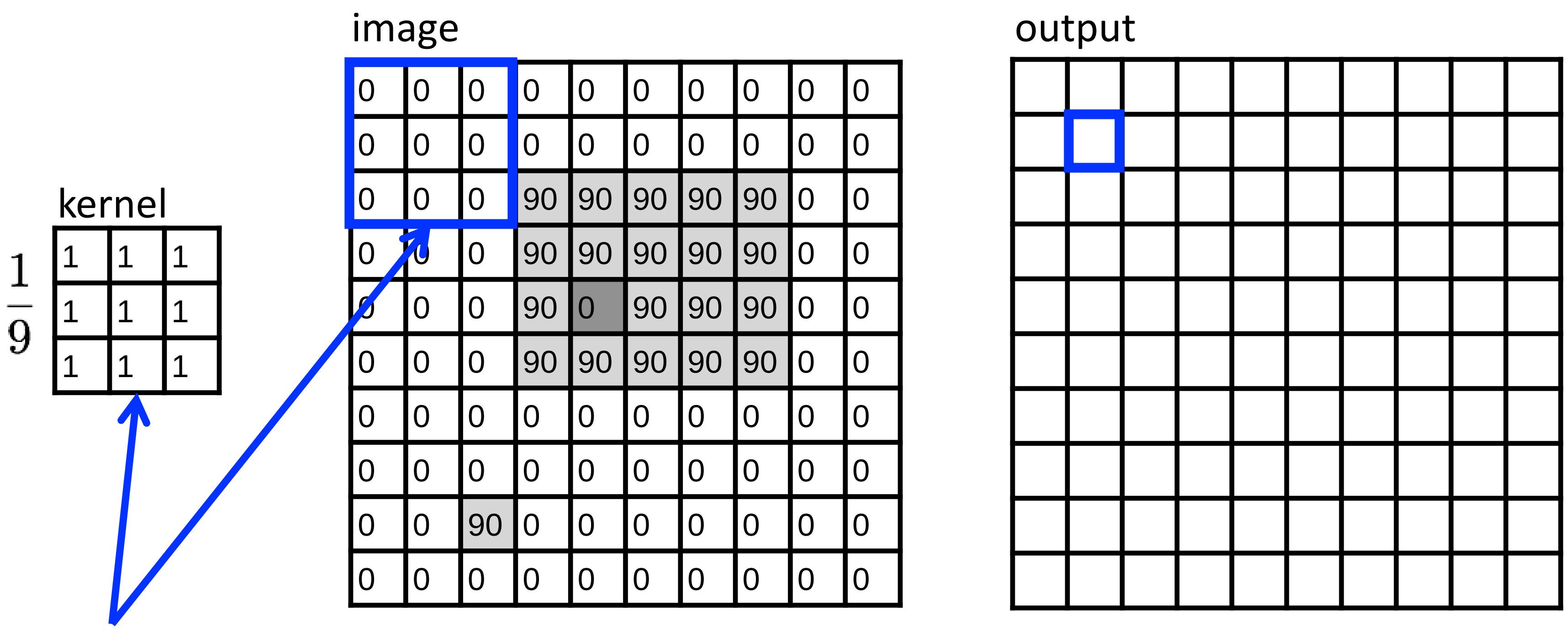
# Example: mean filter

- The kernel is:

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

- Replaces pixel with local average.
- Has smoothing (blurring) effect
- The kernel can be in any other size as well (see .ipynb).

# Run the filter




Note that we assume that the kernel coordinates are centered.
Here the kernel is symmetric horizontally and vertically, so the flipping is not noticeable.

# Run the filter

kernel

$\frac{1}{9}$

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

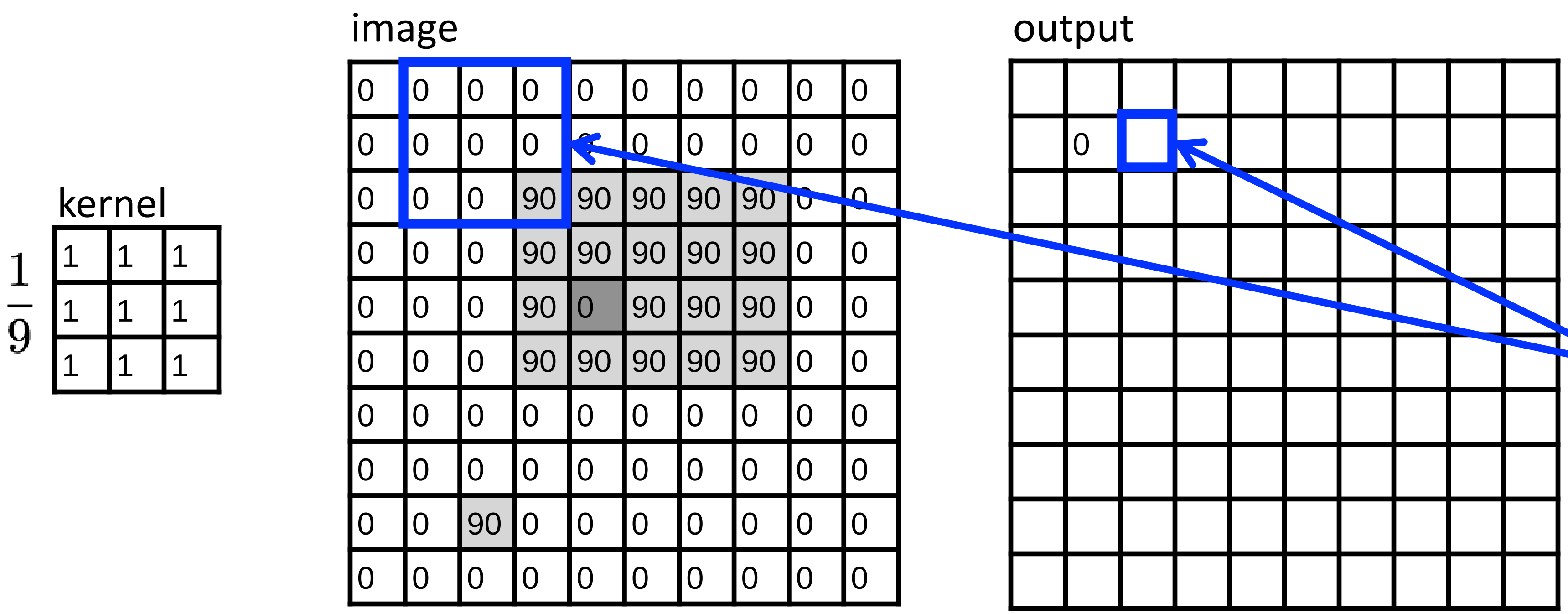
image

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

output

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

# Run the filter




shift-invariant: as the pixel shifts, so does the kernel

# Run the filter

kernel

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

image

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

output

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

# Run the filter

kernel

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

image

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

output

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

# Run the filter

kernel

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

image

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

output

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

# Run the filter

kernel

$\frac{1}{9}$

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

image

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

output

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | 30 | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

# Run the filter

kernel

$\frac{1}{9}$

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

image

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

output

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | | |
| | 0 | 10 | 20 | 30 | 30 | 30 | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

# … and the result is

kernel

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

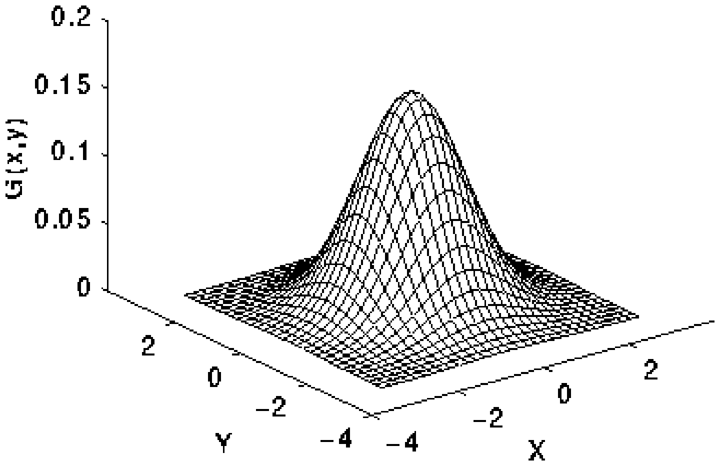
image

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 0 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 90 | 90 | 90 | 90 | 90 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 90 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

output

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

# Gaussian filter



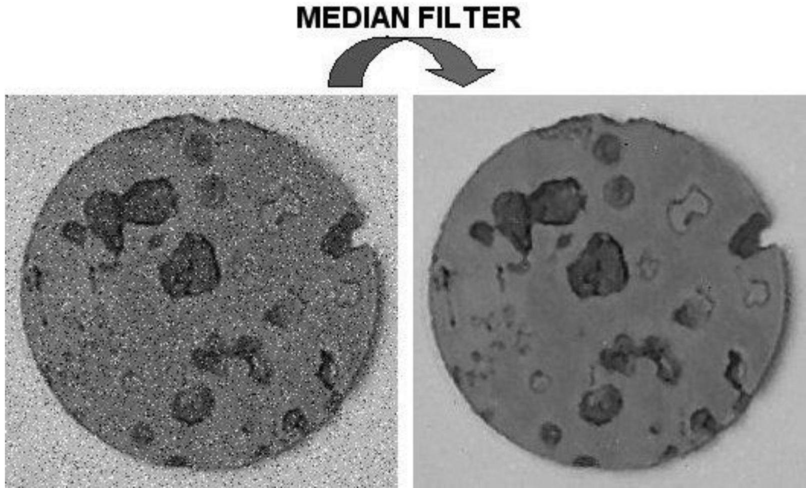

- Another kind of blur filter.
- this filter can be controlled by its size and STD.

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

- The kernel is discretized to bins according to the wanted kernel size.
- Isn't Gaussian function infinite?
  - Most often, the kernel cuts out the remaining lower bins (usually at 2-3 $\sigma$).
- **Both Gaussian and mean filters are good against Gaussian noise, but not effective against S&P noise.**

# Median filter

- Takes the median value from the given neighbors (and hence can also be considered as a “blurring effect”).
- For example:
  - The median of [1, 0, 100] is 1.
- **Median filter is good against salt and pepper noise, and against Gaussian noise (but not as effective).**
- **Median filter is also more computationally expansive.**
- This filter is not LTI because it’s not linear on its weights.



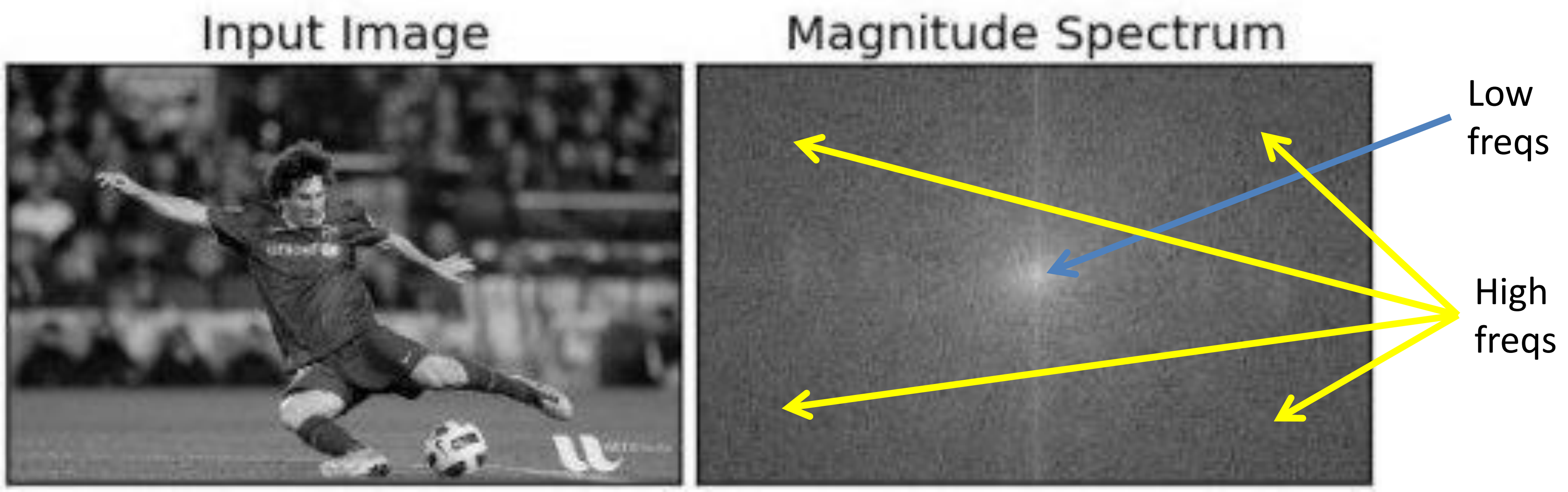
# Example in .ipynb

# contents

- Image representation
- Pixel-wise operations
- Noise and filtering
- **Frequency representation**
- Decimation
- Interpolation
- Morphology operators
- Connected components

# FFT of an image

- Frequency analysis is not in the scope of the course and is not needed for the rest of the course, but here it can give some intuition.
- In audio signals (or any other 1D signals) Lower frequencies change less over time than higher frequencies.
  - In images, the change is represented in change in distance, so images that changes slowly from pixel to pixel has more lower frequencies then others.
- Natural images are mainly built from low frequencies.



# Convolution in frequency domain

- Recall: in time (space) domain:

$$g(i,j) = \sum_{u=-k}^{k} \sum_{v=-k}^{k} h[u,v] f[i-u, j-v]$$

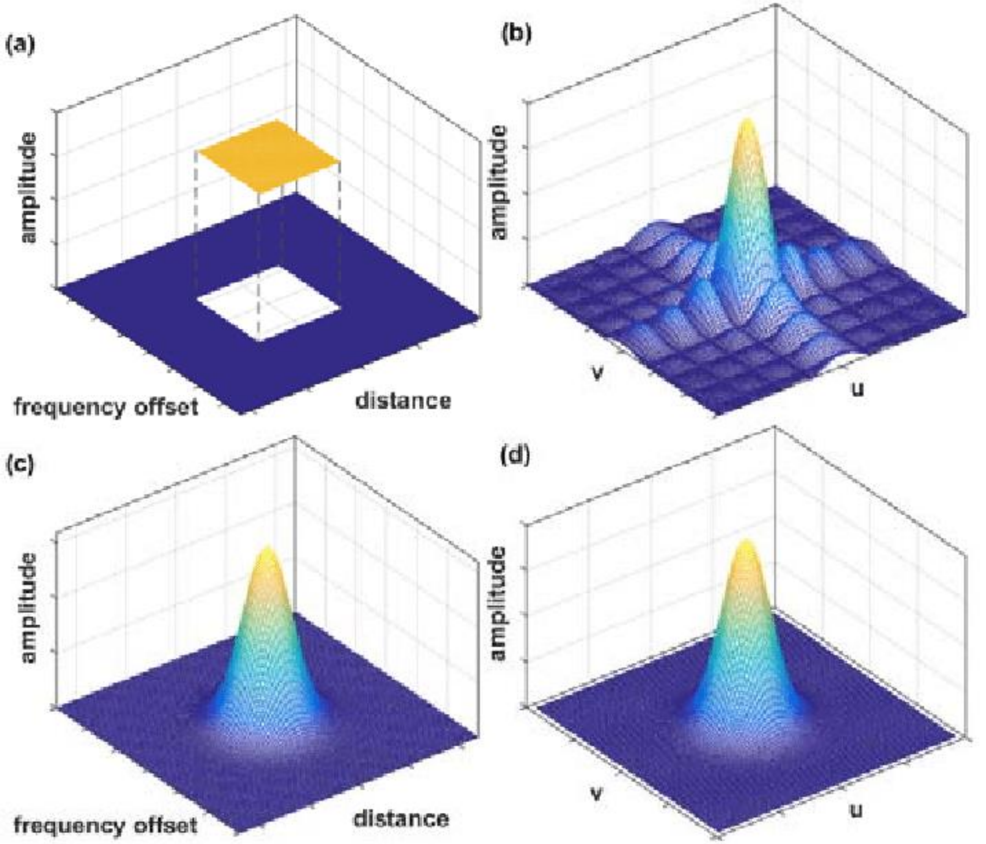
$$g = h * f$$

- In frequency domain- simple multiplication:

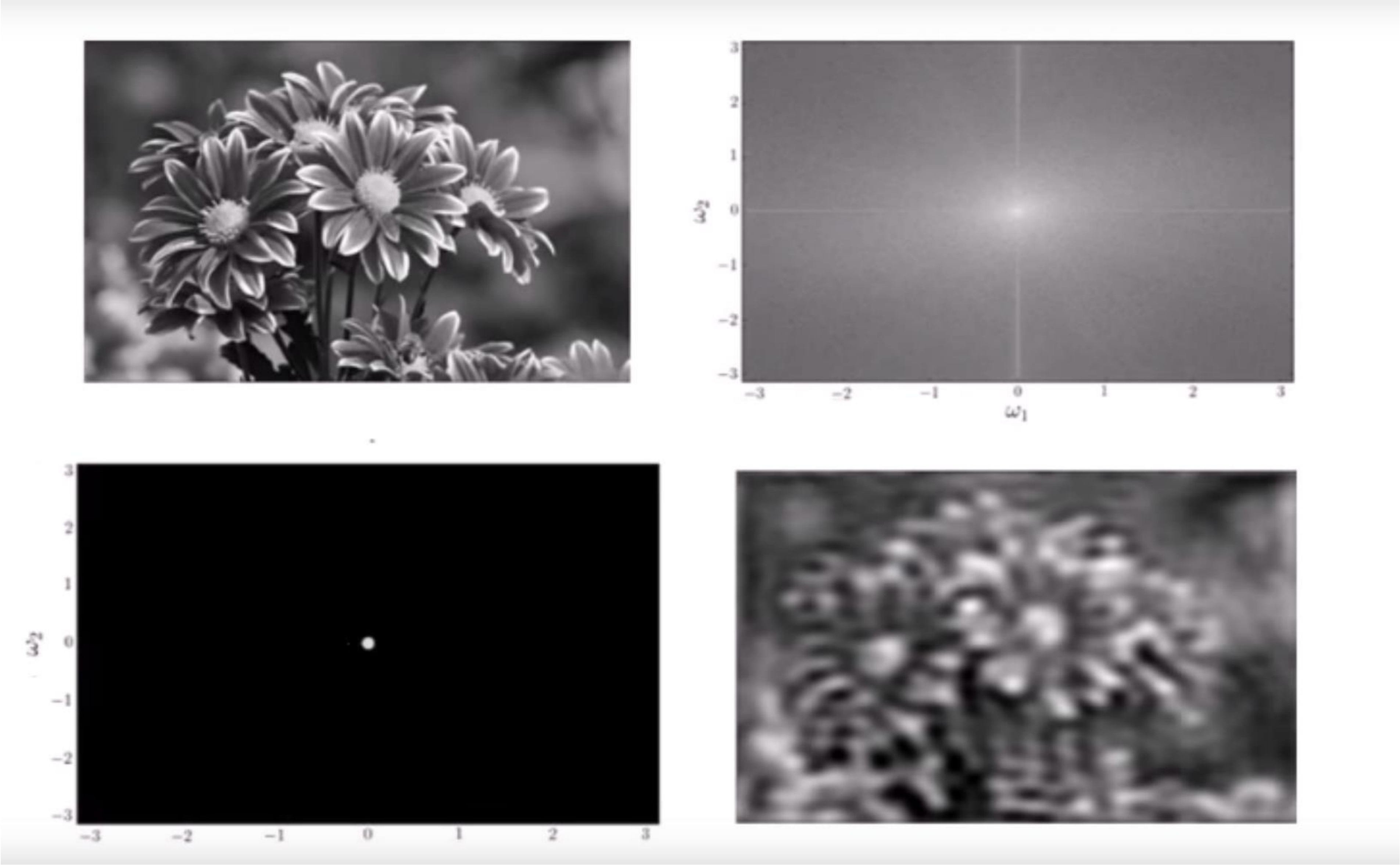
$$G = H \cdot F$$

# Low-pass filters

- Both mean and Gaussian filters are considered low-pass filters because in the frequency domain, they have higher values in the lower frequencies- and when multiplied with frequency spectrums, the high frequencies get smaller.
- When image is left only with the lower frequencies, the rapidly changes parts of the image (e.g.: edges, noise) are smoothen.

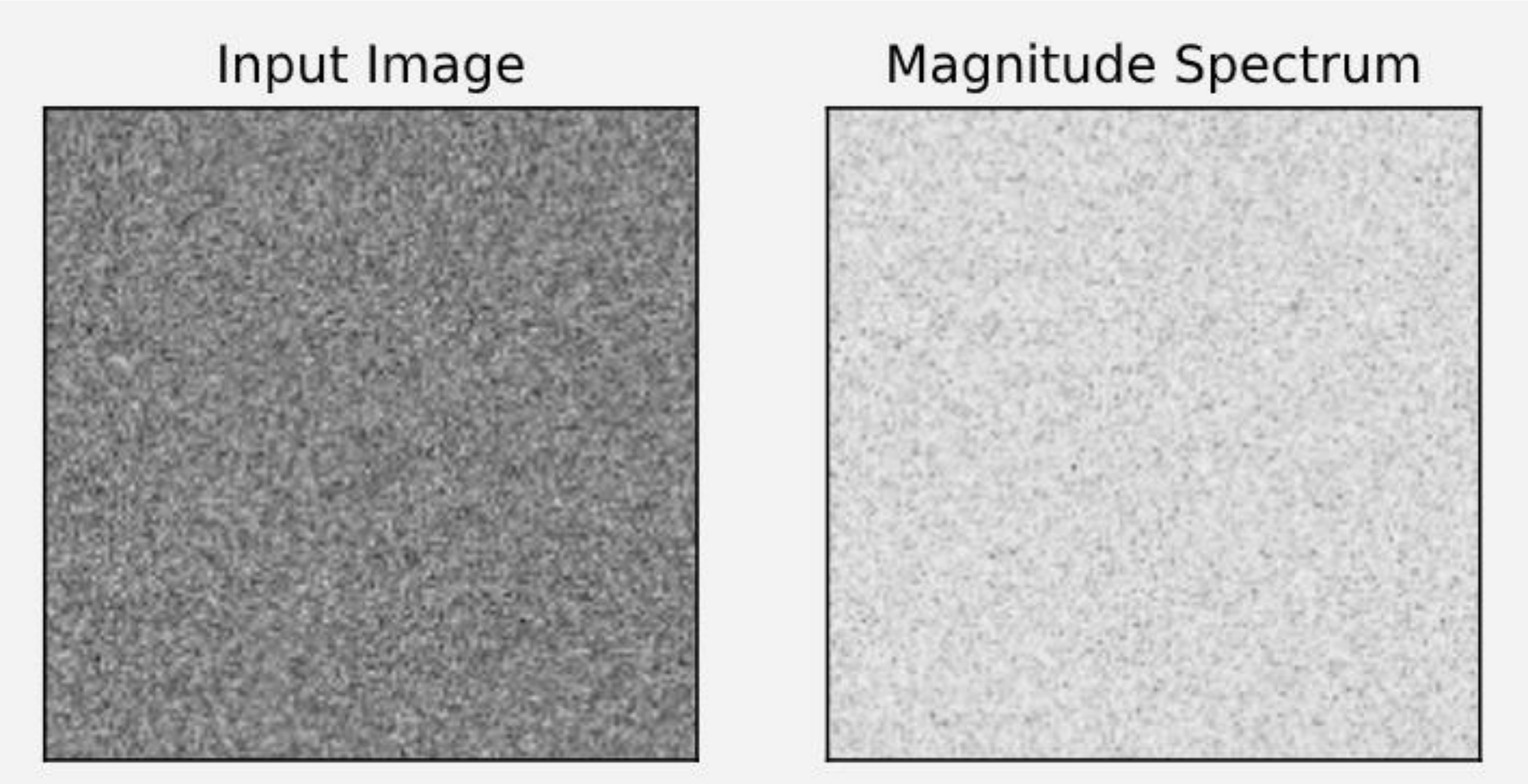


# LP example



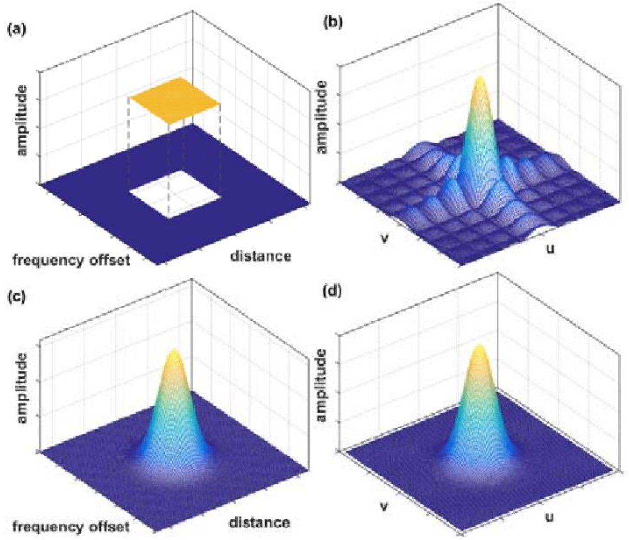
# FFT of gaussian noise

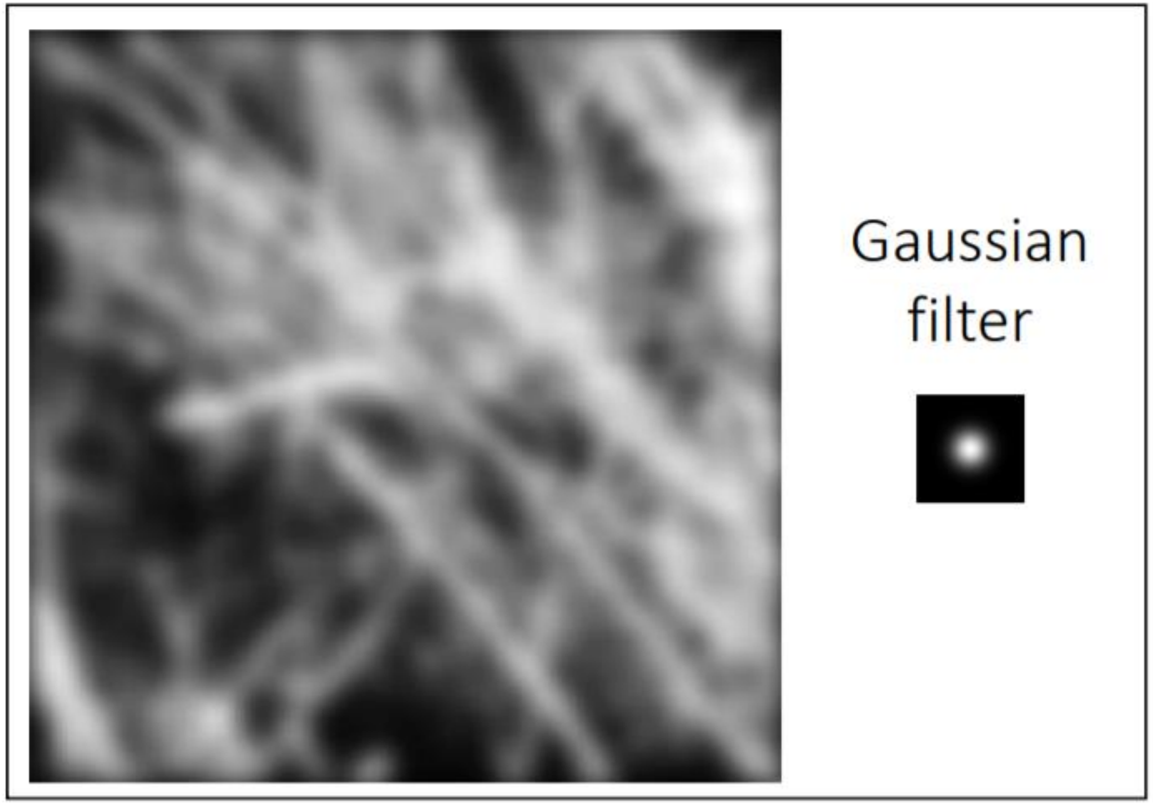
- Since gaussian noise (AWGN) is distributed along all frequencies, LP filter reduce this kind of noise significantly.



# Mean vs. Gaussian filter

- Since Mean filter has higher values in the higher frequencies, edge artifacts sometimes remains.



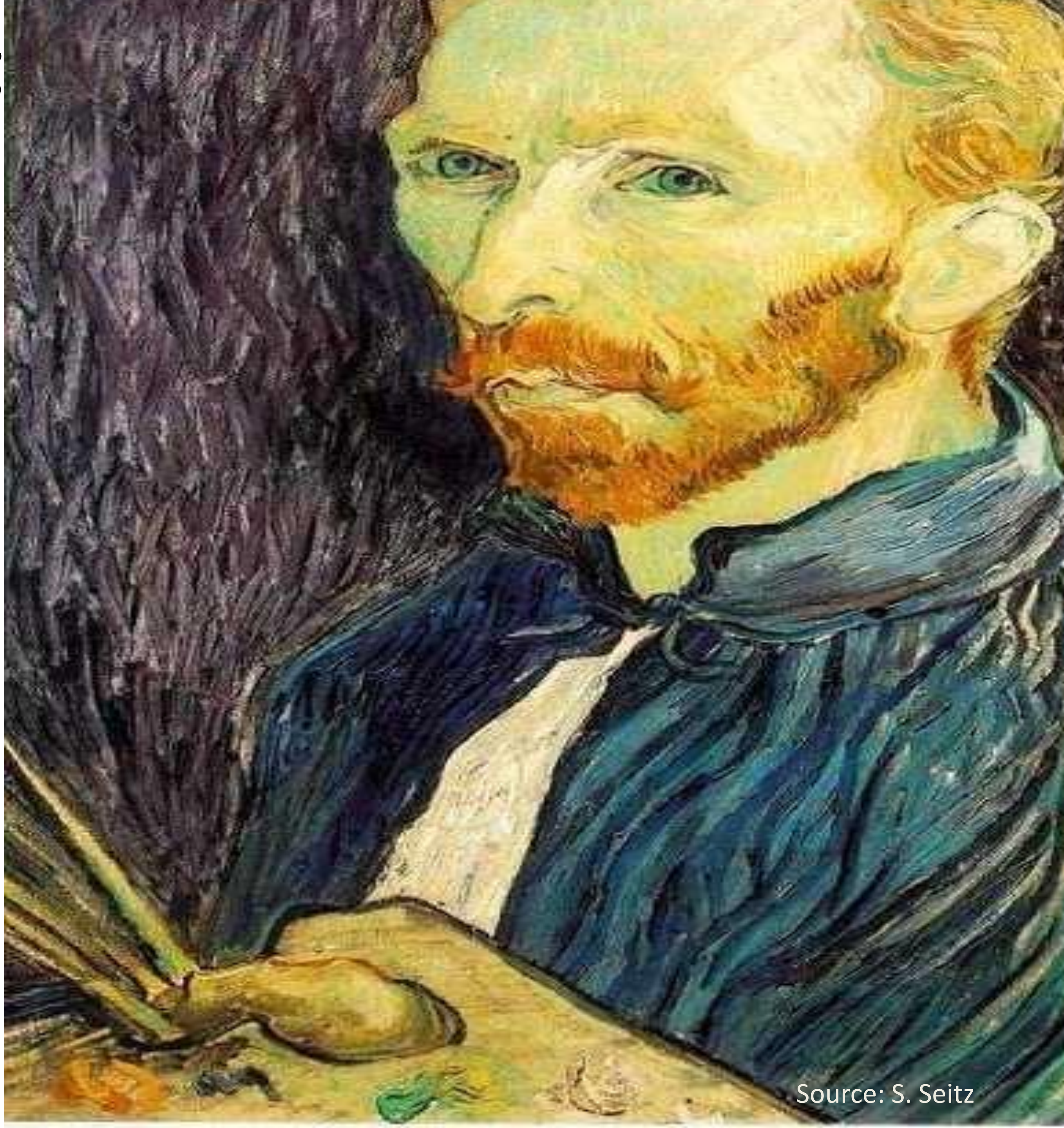




# contents

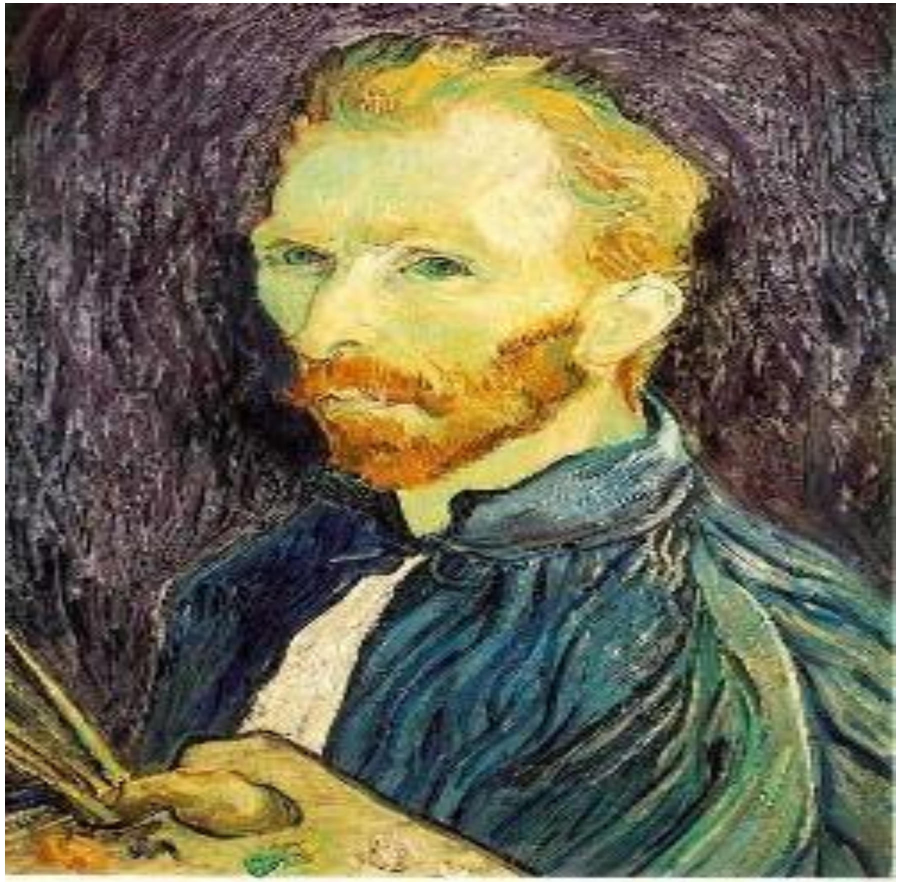
- Image representation
- Pixel-wise operations
- Noise and filtering
- Frequency representation
- **Decimation**
- Interpolation
- Morphology operators
- Connected components

# Imag

This image is too big to fit on the screen.  How can we generate a half-sized version?



# Image sub-sampling



1/4

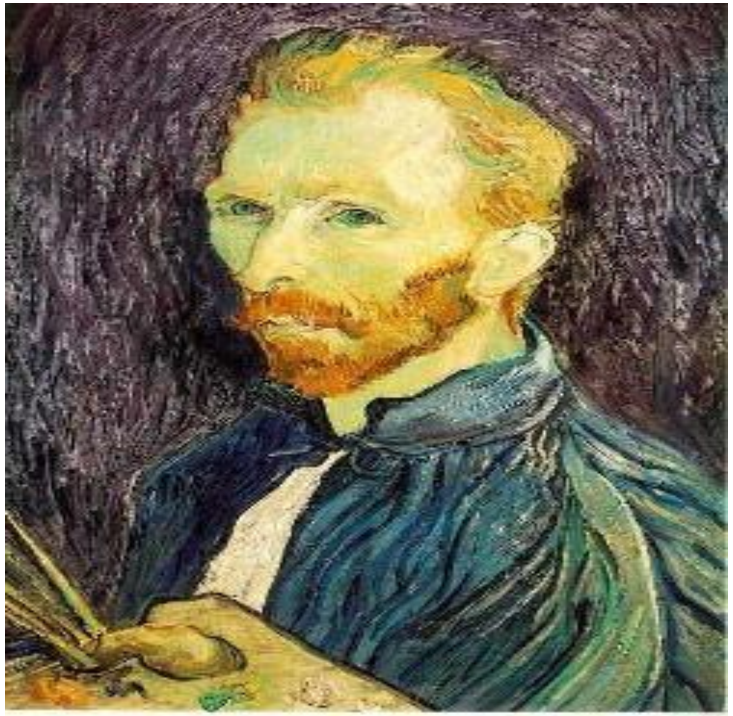
1/8

Throw away every other row and column to create a 1/2 size image

- called **image sub-sampling** or **decimation**

Source: S. Seitz

# Image sub-sampling



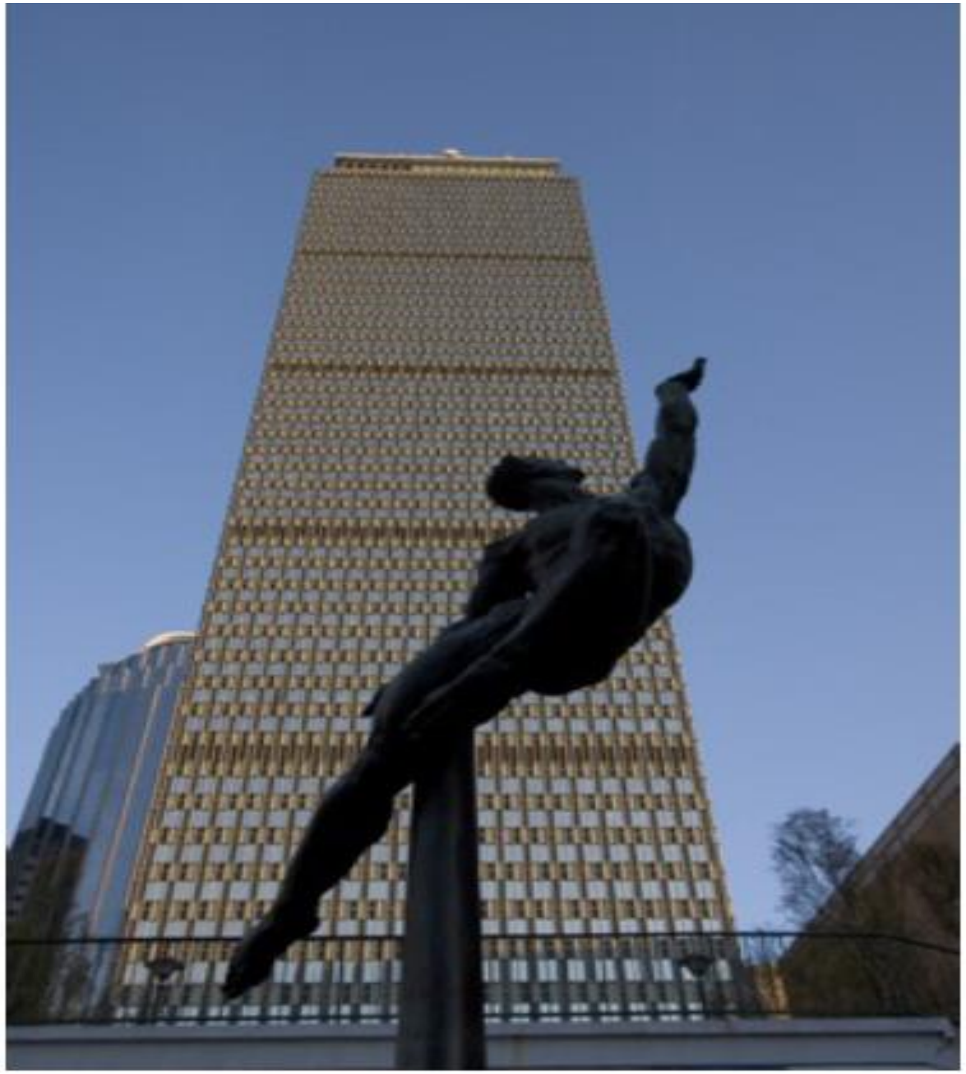
1/2

1/4  (2x zoom)

1/8  (4x zoom)

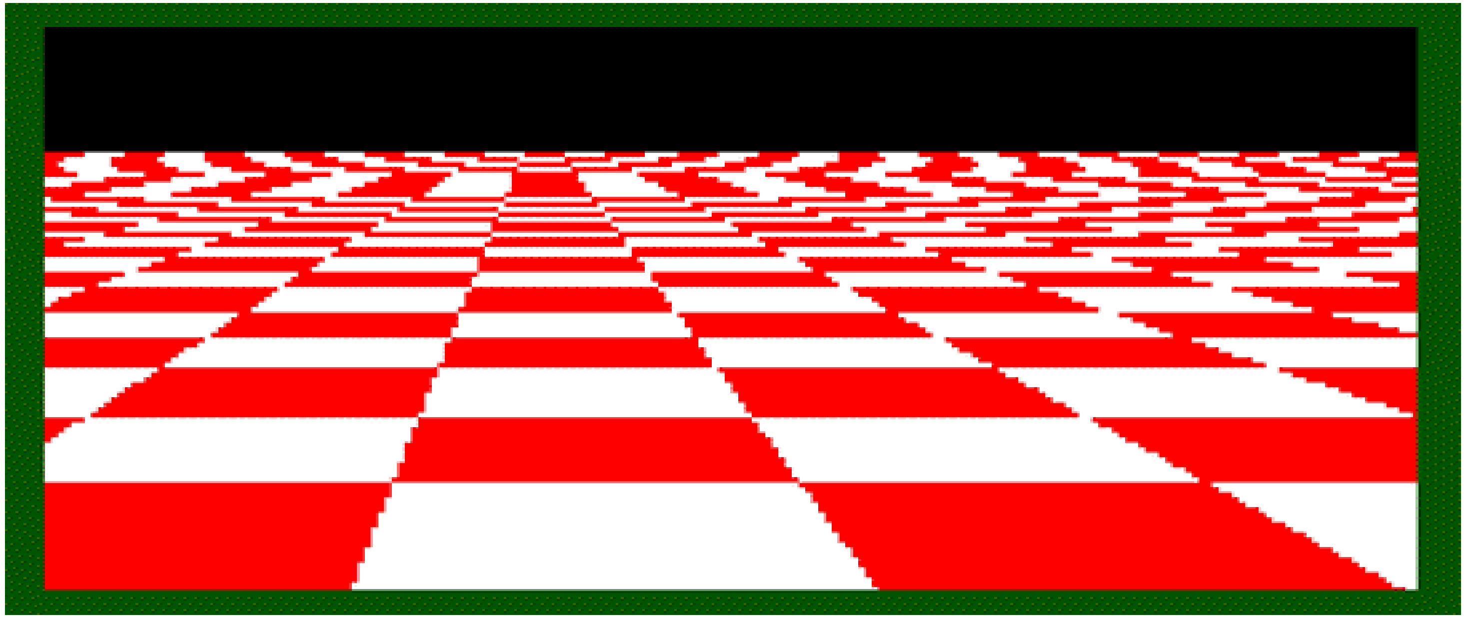
Source: S. Seitz

# Image sub-sampling



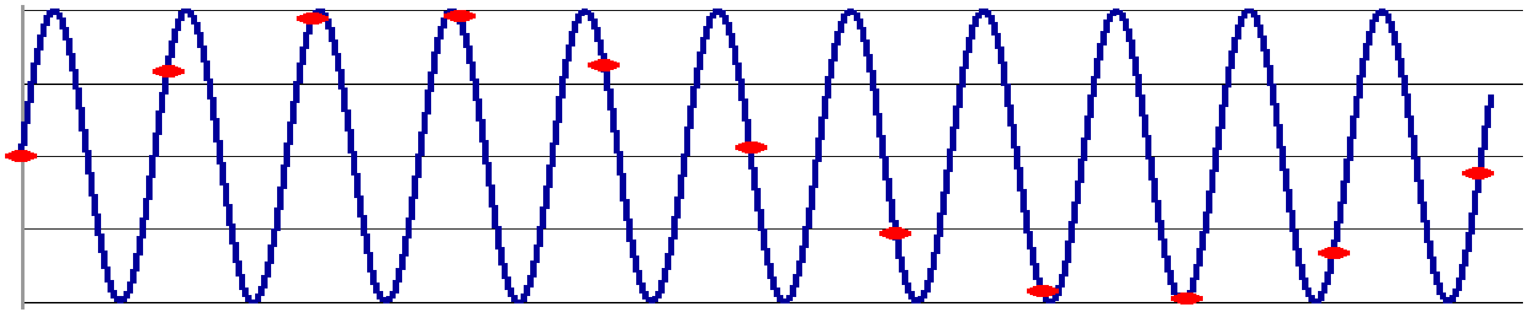
Source: F. Durand

# Even worse for synthetic images



Source: L. Zhang

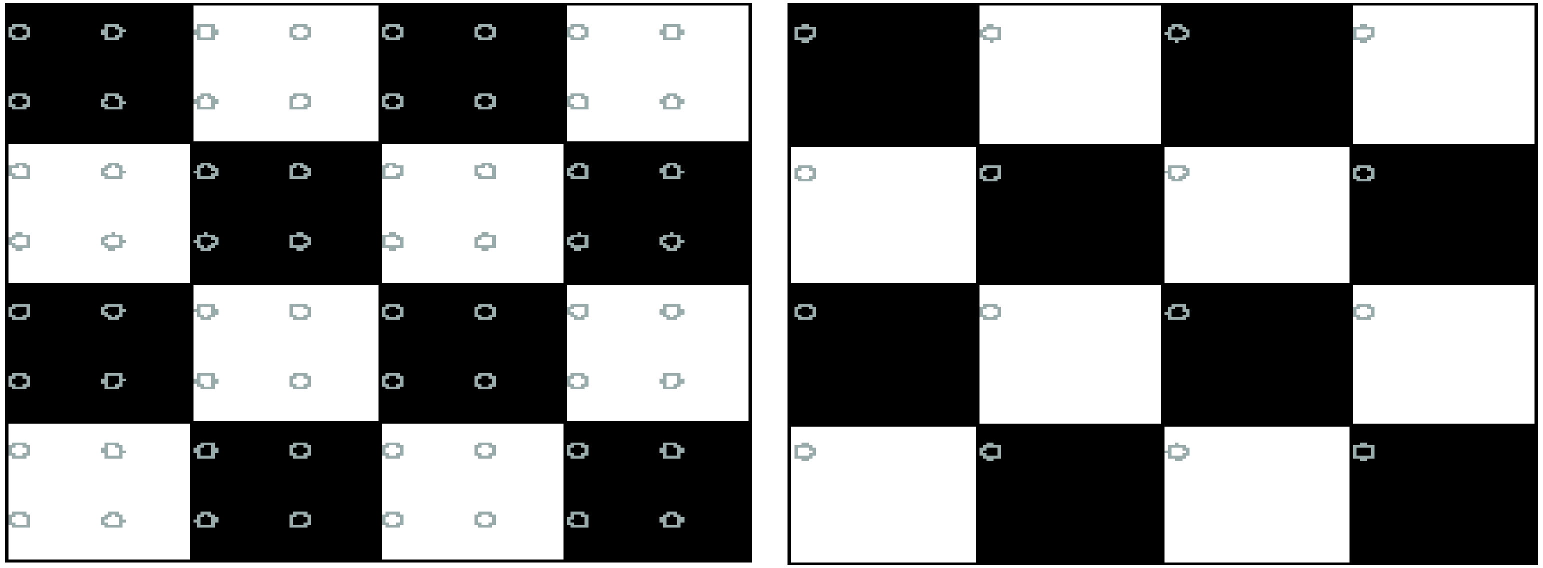
# Aliasing



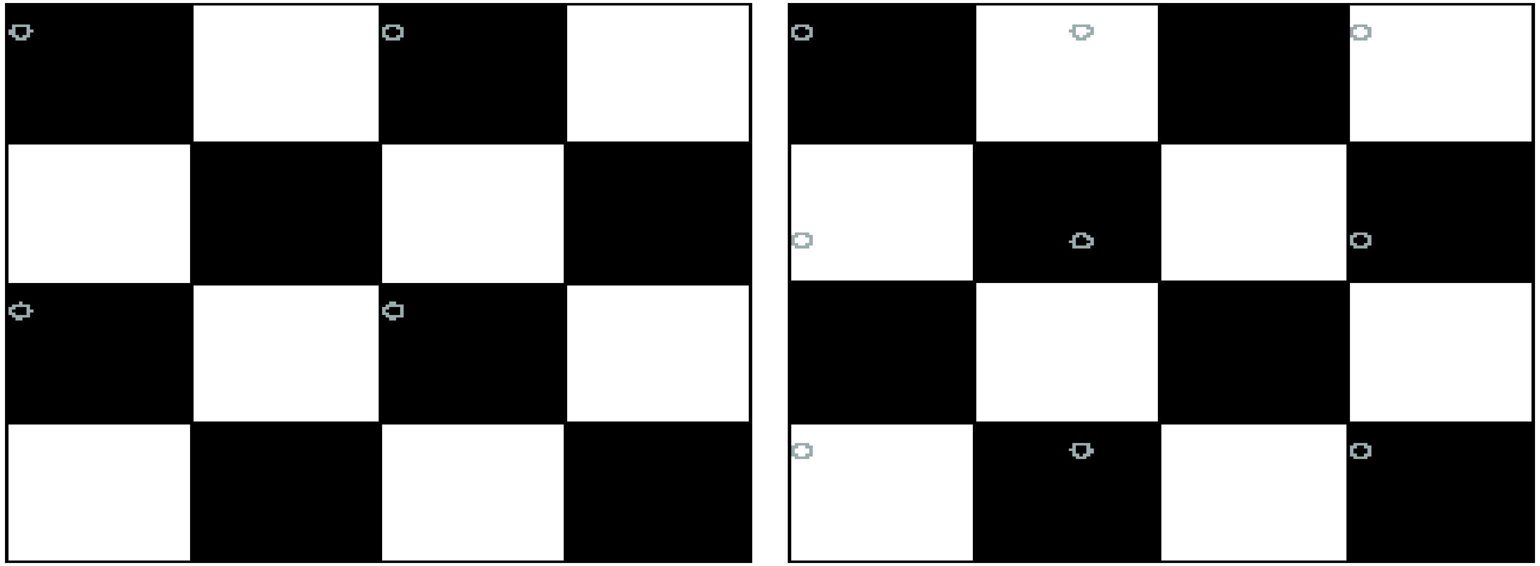
- Occurs when your sampling rate is not high enough to capture the amount of detail in your image
- Can give you the wrong signal/image—an *alias*
- To do sampling right, need to understand the structure of your signal/image
- To avoid aliasing:
  - **sampling rate ≥ 2 * max frequency** in the image
  - This minimum sampling rate is called the **Nyquist rate**

Source: L. Zhang

# Nyquist limit – 2D example



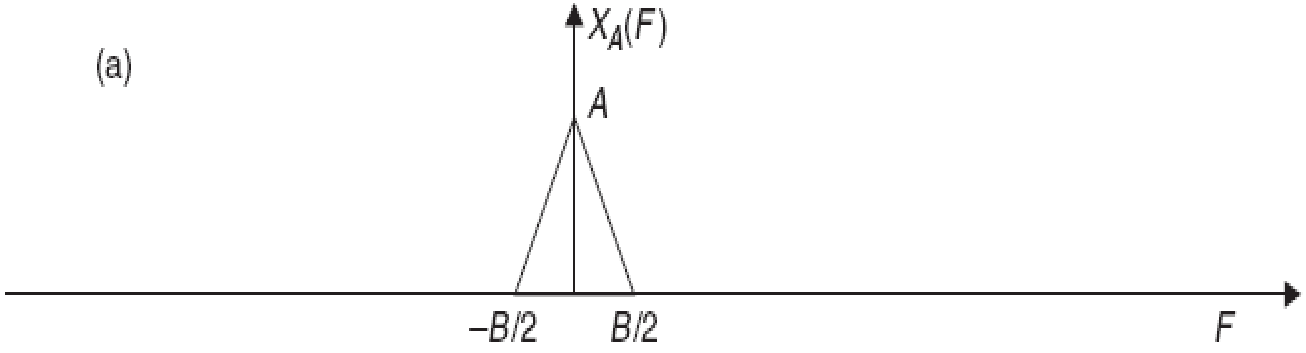
Good sampling



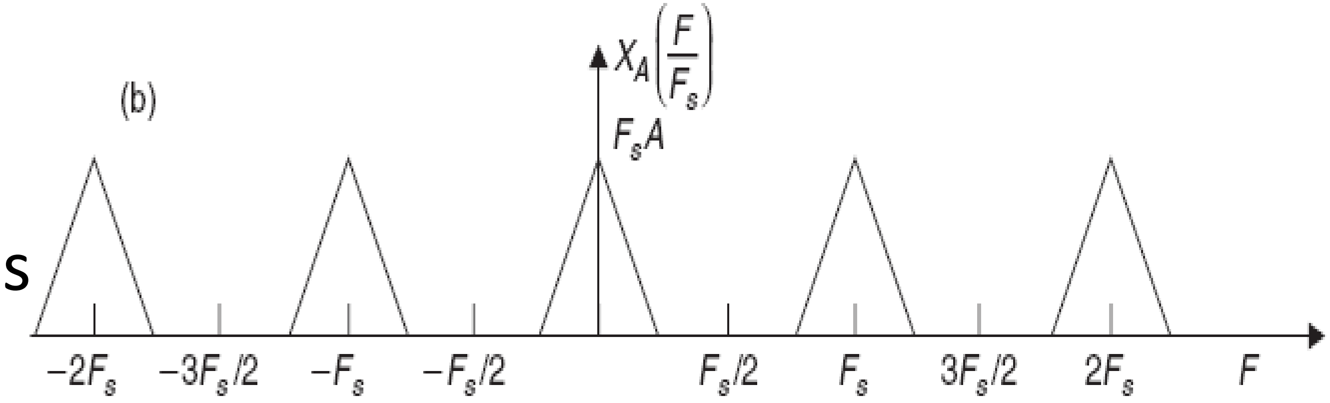
Bad sampling

# Nyquist limit- frequency response

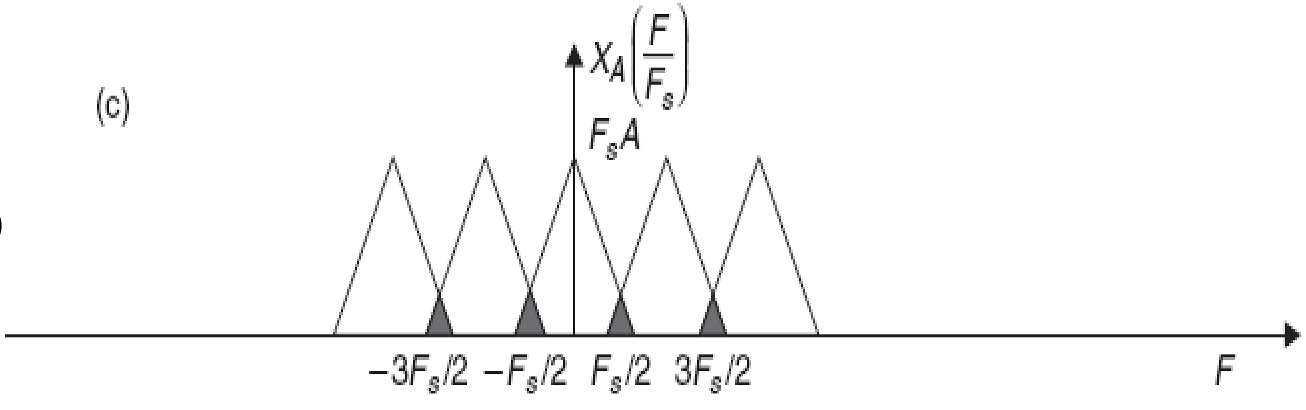
- Original freq representation of signal.




- Regular sampling above nyquist rate- can recreate the original frequencies of the image. (copies are from sampling).

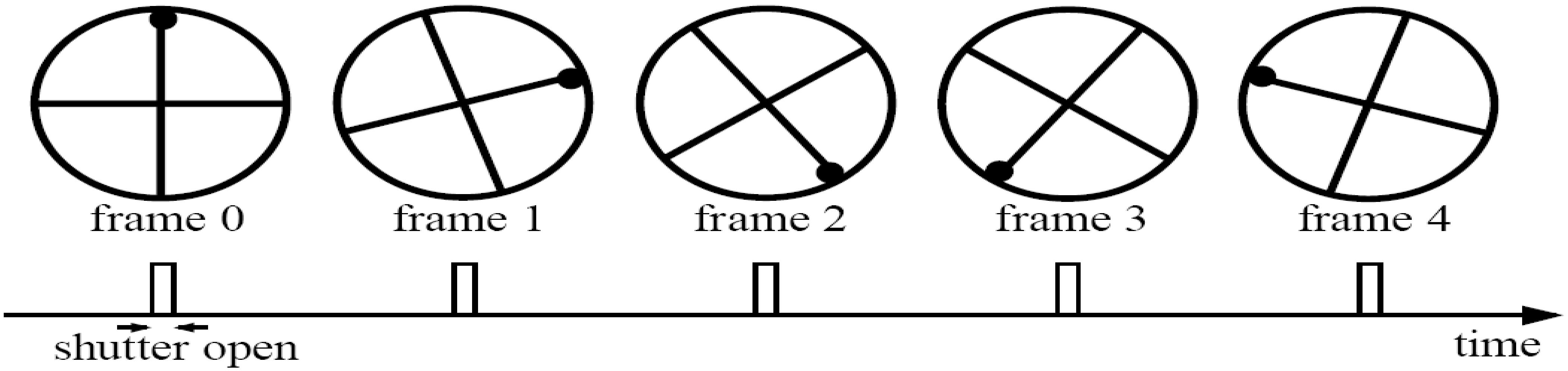



- Sampling below nyquist- original frequencies are destroyed due to the copies overlap- **this is the aliasing**.



# Example: wagon-wheel effect

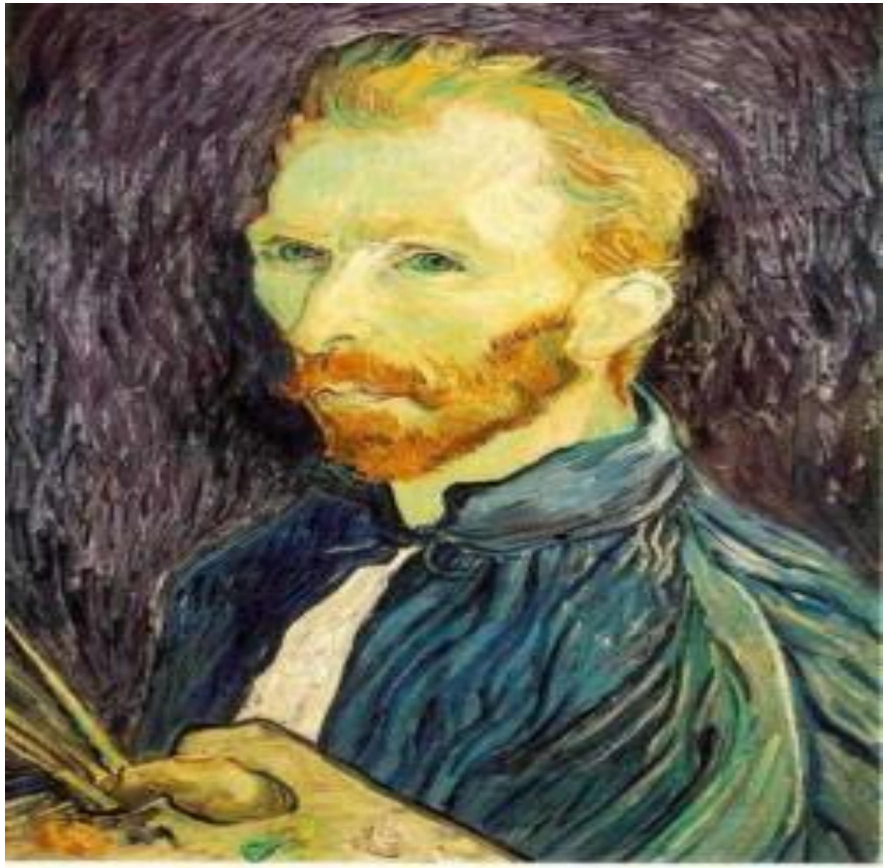
- An example of sub sampling in time domain (instead of spatially like before).




Without dot, wheel appears to be rotating slowly backwards!
(counterclockwise)

- https://en.wikipedia.org/wiki/Wagon-wheel_effect

# Gaussian pre-filtering



Gaussian 1/2

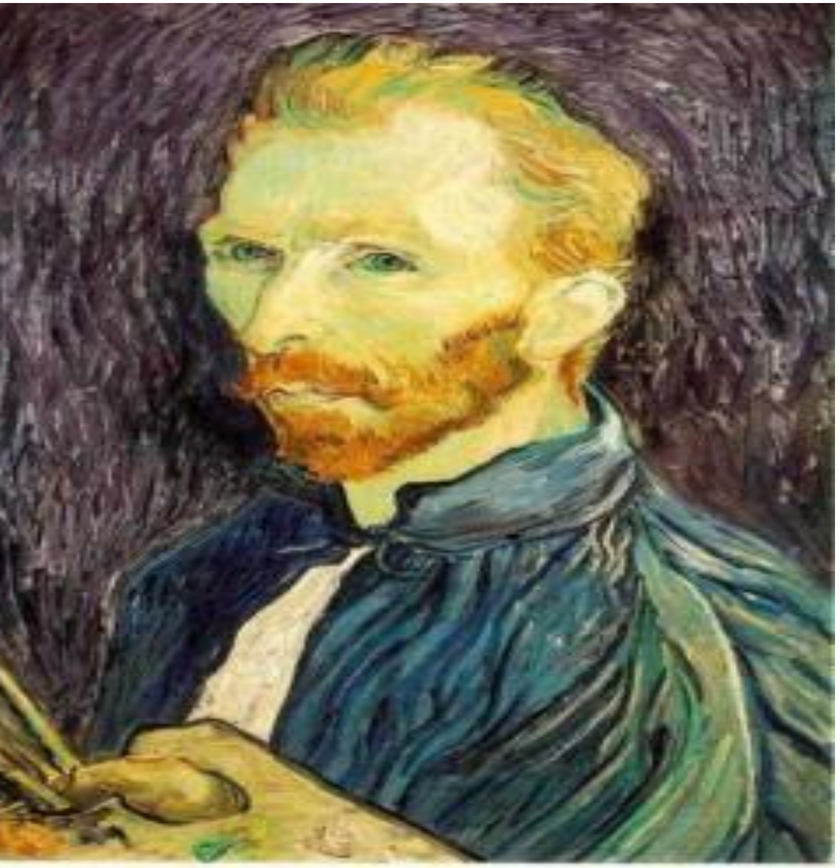
G 1/4

G 1/8

- Solution: filter the image, *then* subsample

Source: S. Seitz

# Subsampling with Gaussian pre-filtering



Gaussian 1/2

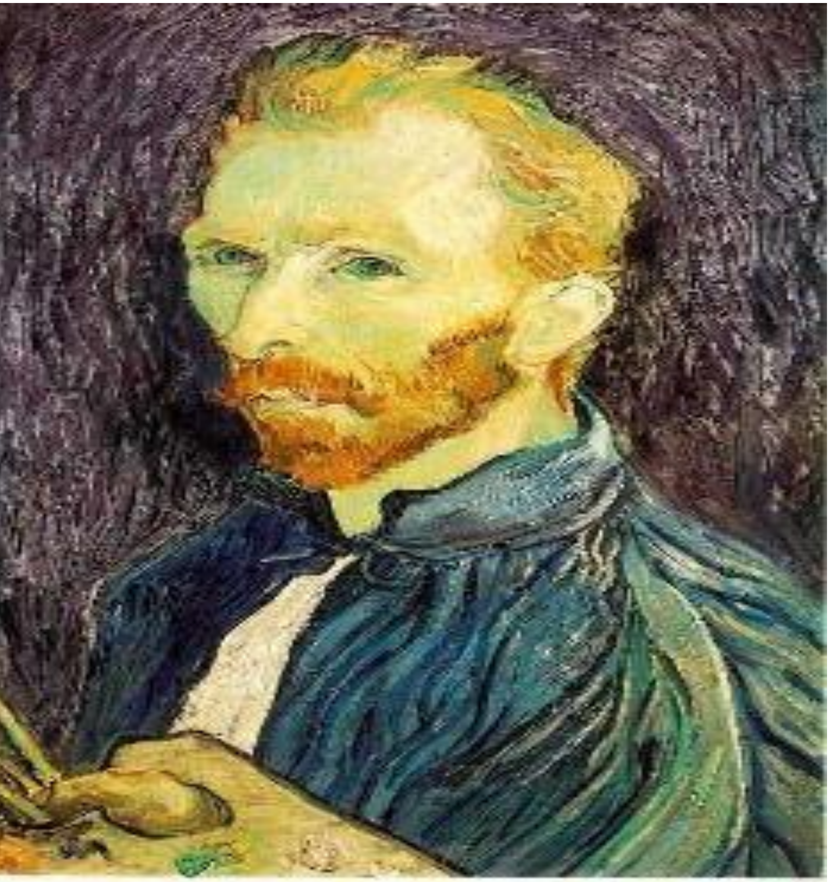
G 1/4

G 1/8

- Solution: filter the image, *then* subsample

Source: S. Seitz

# Compare with...



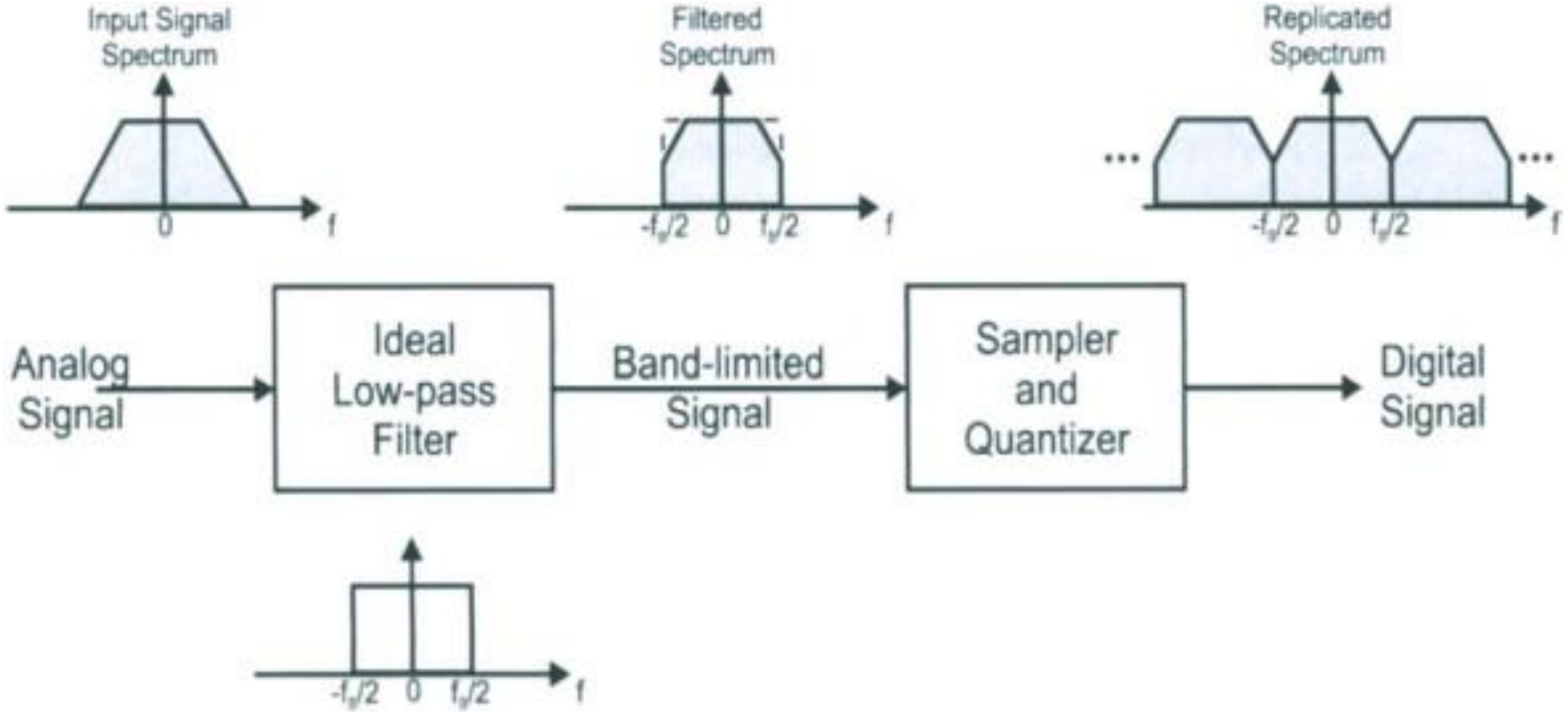
1/2

1/4 (2x zoom)

1/8 (4x zoom)

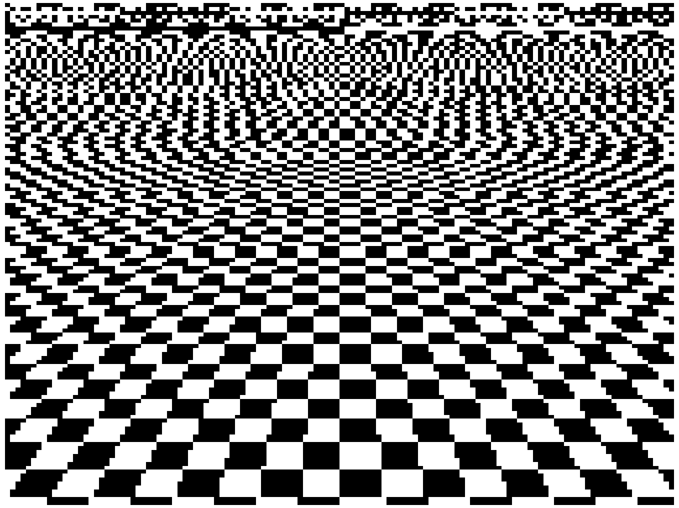
Source: S. Seitz

# Low pass filtering- frequency response

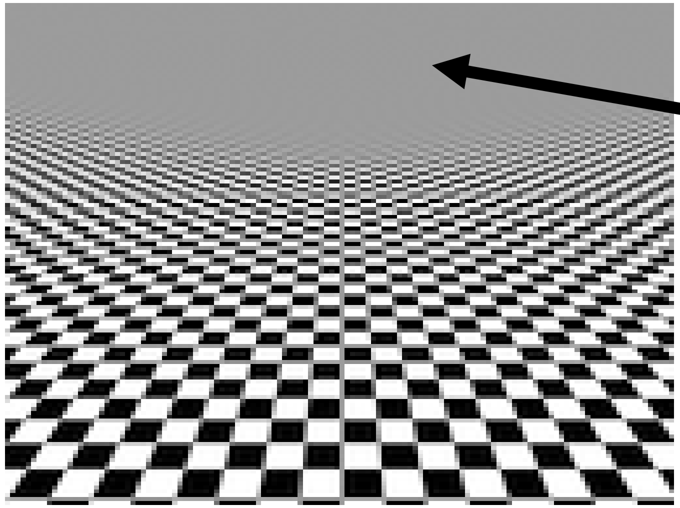


# Back to the checkerboard

- What should happen when you make the checkerboard smaller and smaller?



Naïve subsampling



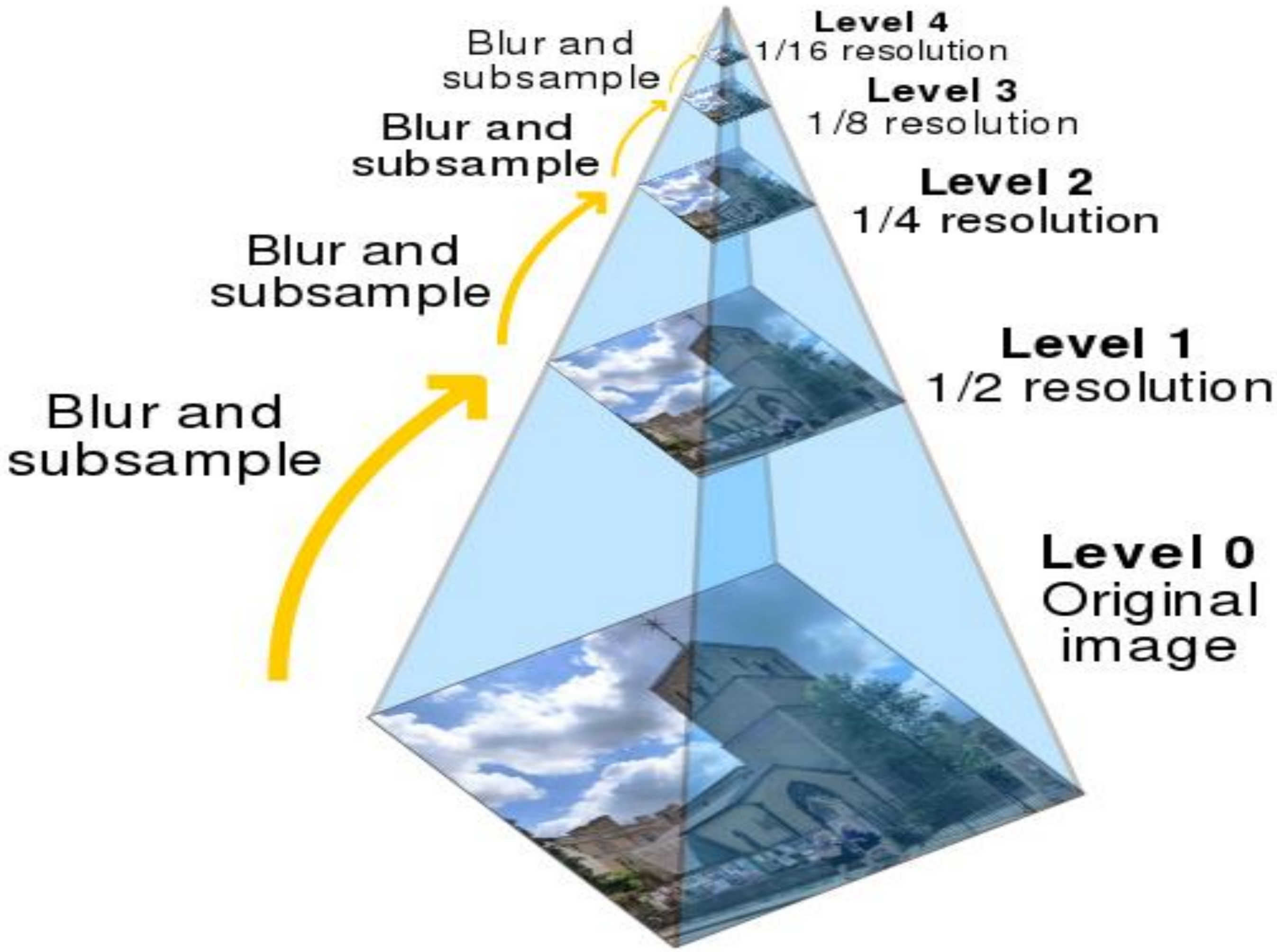
Proper prefiltering (“antialiasing”)

Image turns grey! (Average of black and white squares, because each pixel contains both.)

# Gaussian pre-filtering

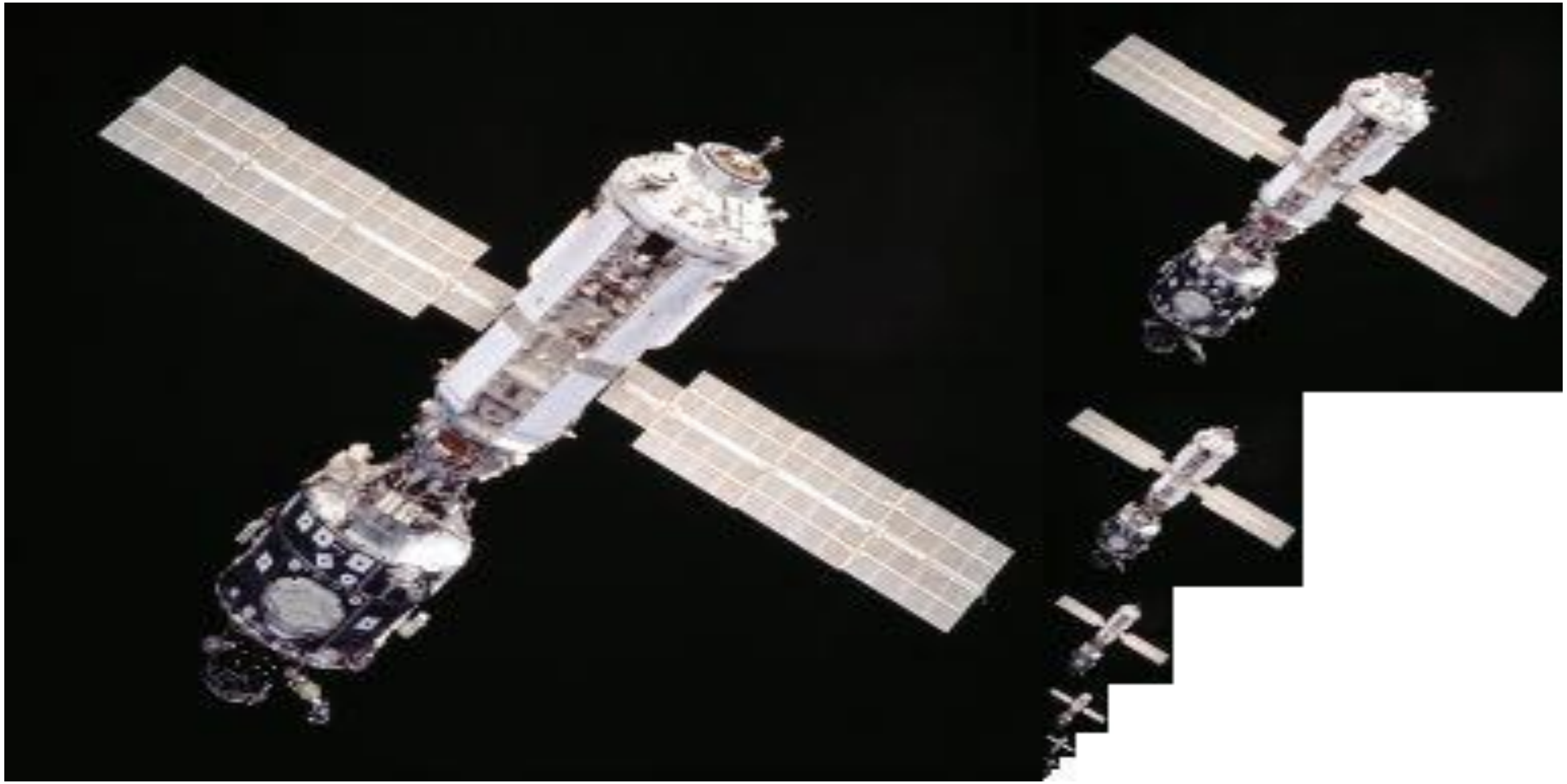
- Solution: filter the image, *then* subsample


Gaussian pyramid
F0
F1
F2
blur
subsample
blur
subsample
...
F0 *H
F1 *H



Level 4
1/16 resolution
Blur and
subsample
Level 3
1/8 resolution
Blur and
subsample
Level 2
1/4 resolution
Blur and
subsample
Level 1
1/2 resolution
Blur and
subsample
Level 0
Original
image

# Gaussian Pyramid



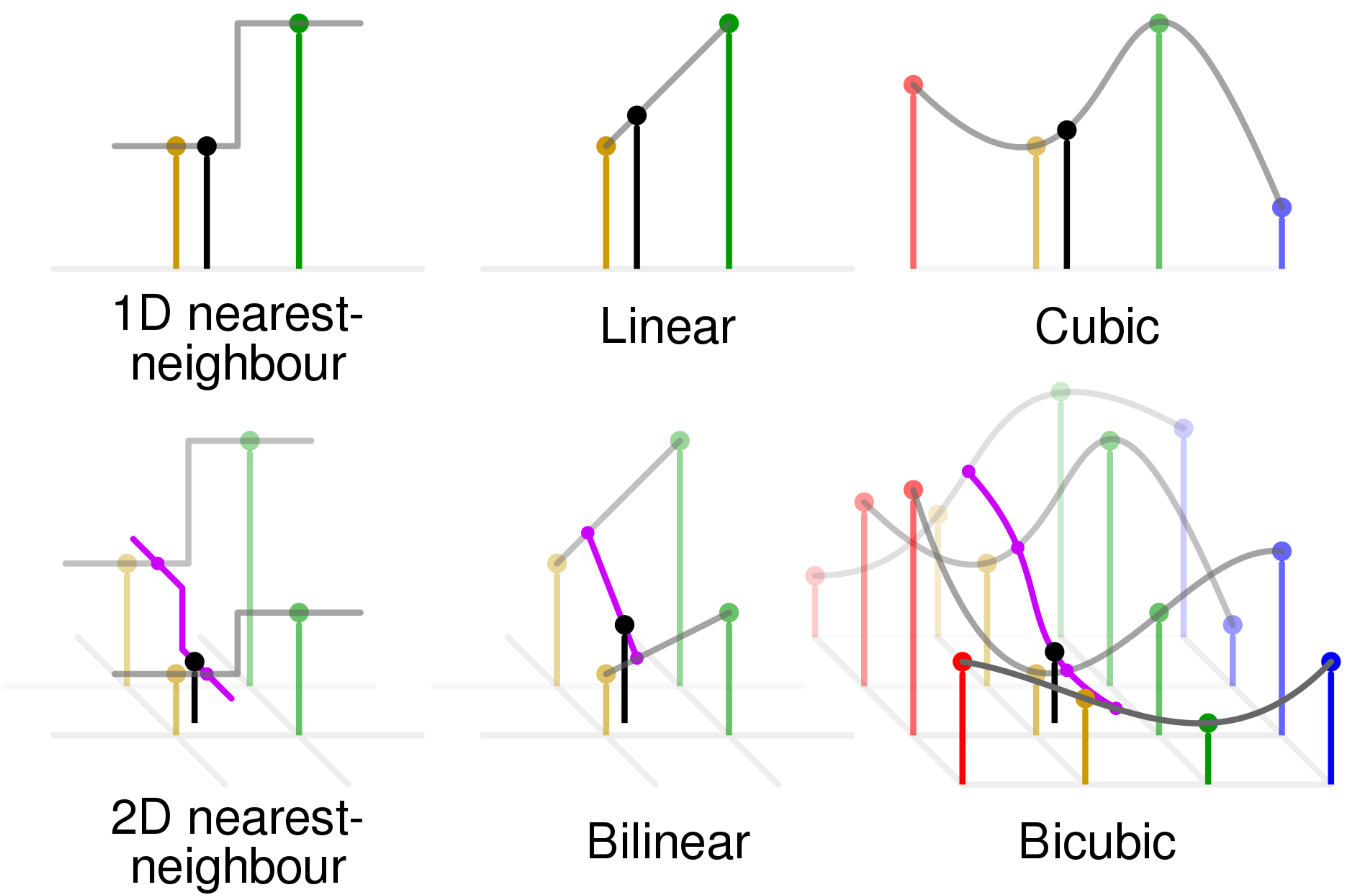
# contents

- Image representation
- Pixel-wise operations
- Noise and filtering
- Frequency representation
- Decimation
- **Interpolation**
- Morphology operators
- Connected components

# Upsampling

- This image is too small for this screen:
- How can we make it 10 times as big?
- Simplest approach: repeat each row and column 10 times ("**Nearest neighbor interpolation**")
- This operation is known as **upsampling** or **interpolation.**

# Upsampling



1D nearest-neighbour
Linear
Cubic
2D nearest-neighbour
Bilinear
Bicubic

# Image interpolation

Original image:  x 10

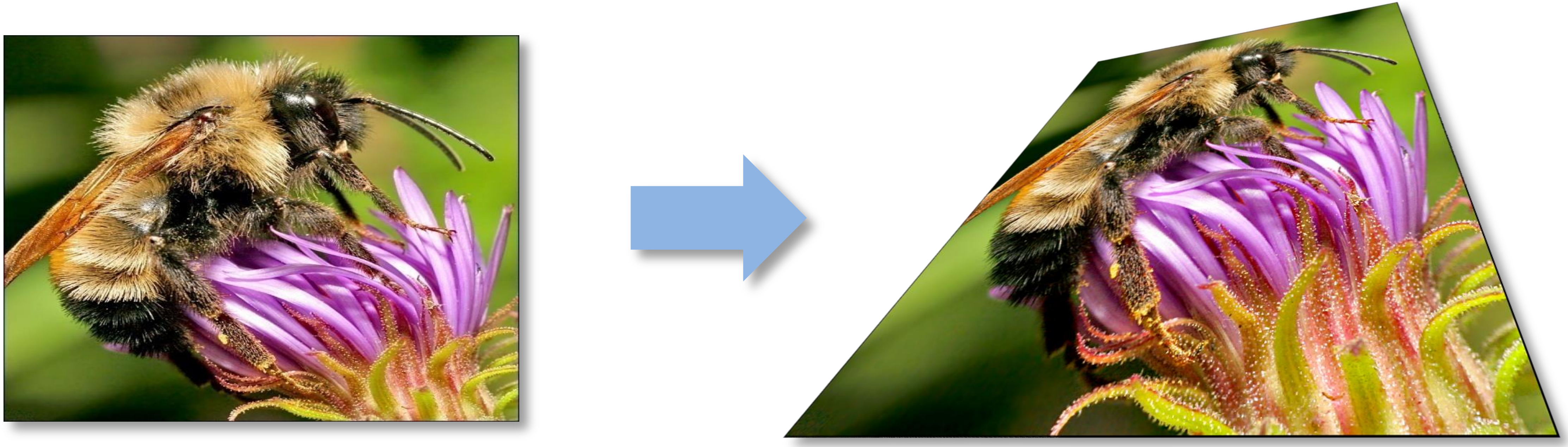
Nearest-neighbor interpolation

Bilinear interpolation

Bicubic interpolation

# Image interpolation

Also used for image warping



# contents

- Image representation
- Pixel-wise operations
- Noise and filtering
- Frequency representation
- Decimation
- Interpolation
- **Morphology operators**
- Connected components

# Morphology

- Handy tool whenever needed to clean up binary images.
- Each morphology operator is constructed as such:
    1. Select some kind of structure element (binary kernel)

$$s = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

# Morphology

- Handy tool whenever needed to clean up binary images.
- Each morphology operator is constructed as such:
  1. Select some kind of structure element (binary kernel)
  2. Convolve with input binary image $g = f * s$

$$s = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

# Morphology

- Handy tool whenever needed to clean up binary images.
- Each morphology operator is constructed as such:
  1. Select some kind of structure element (binary kernel)
  2. Convolve with input binary image $g = f * s$
  3. Threshold the output

$$s = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\theta_{TH}(x,t) = \begin{cases} 1 & if \quad x \geq t, \\ 0 & else \end{cases}$$

# Morphology

- Handy tool whenever needed to clean up binary images.
- Each morphology operator is constructed as such:
  1. Select some kind of structure element (binary kernel)
  2. Convolve with input binary image $g = f * s$
  3. Threshold the output

$$s = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\theta_{TH}(x,t) = \begin{cases} 1 & if \quad x \geq t, \\ 0 & else \end{cases}$$

- Overall morphologic operation should look like so:

$$k = \theta_{TH}(f * s, t)$$

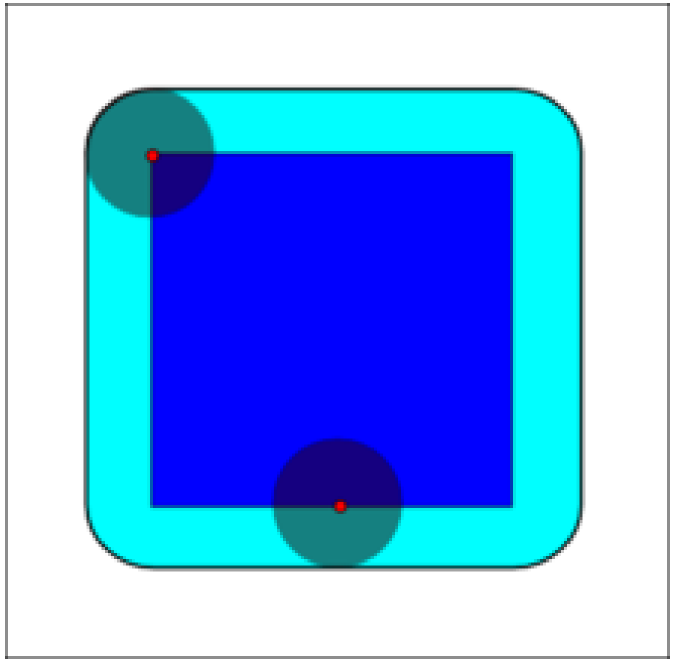
# Dilation

$$s = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

1. Select some kind of structure element (binary kernel)
2. Convolve with input binary image $g = f * s$
3. Threshold the output

$$\theta_{TH}(x,t) = \begin{cases} 1 & if \quad x \geq t, \\ 0 & else \end{cases}$$

- Dilation: $t = 1$
  - Skinny lines will get thicker



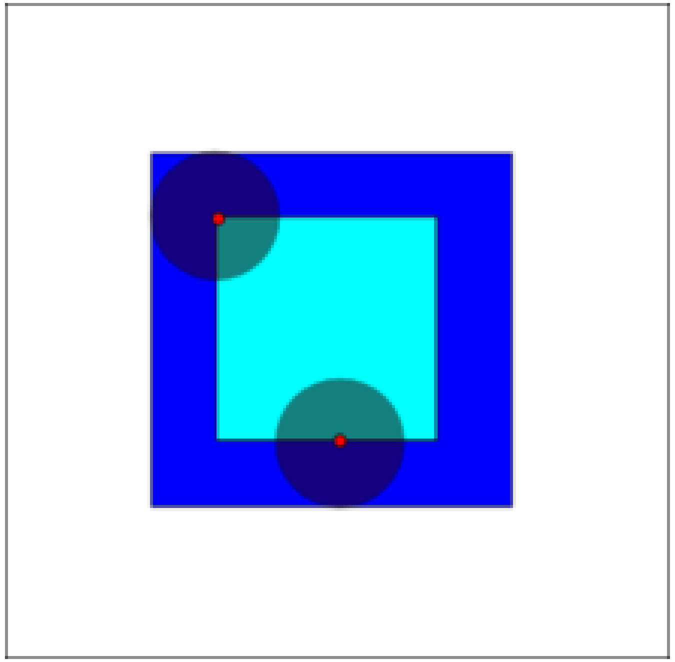
# Erosion

1. Select some kind of structure element (binary kernel)
2. Convolve with input binary image $g = f * s$
3. Threshold the output

$$s = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

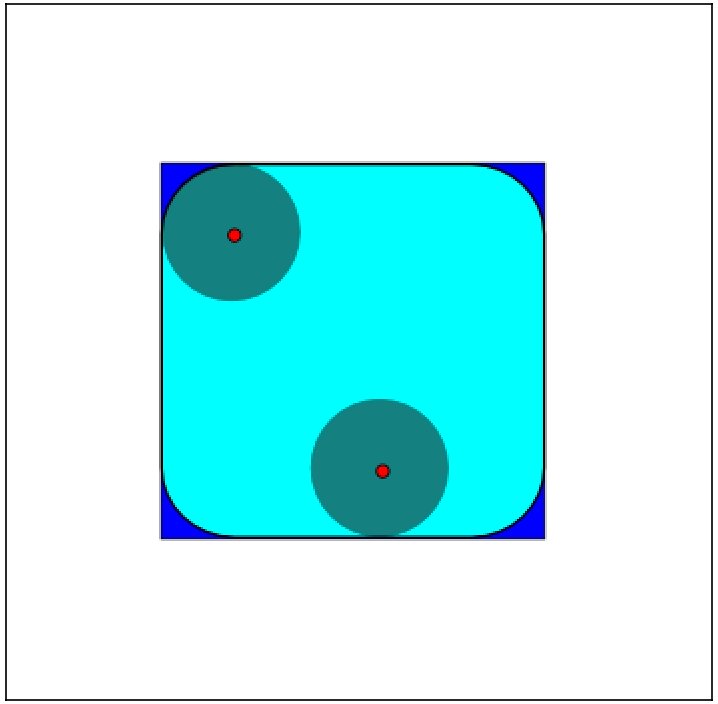
$$\theta_{TH}(x,t) = \begin{cases} 1 & if \quad x \geq t, \\ 0 & else \end{cases}$$

- Erosion: $t = sum(s)$
  - Thicker lines will get skinny



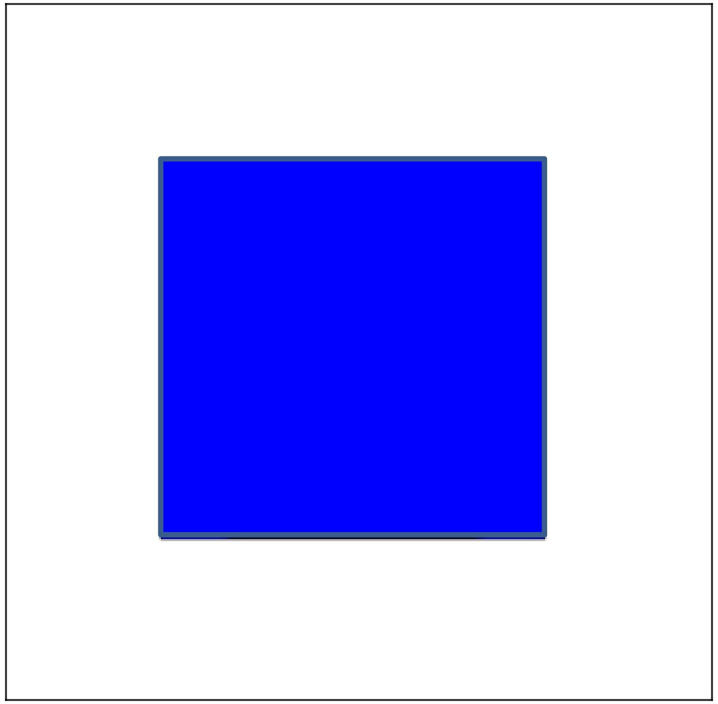
# Opening

- Erosion followed by dilation.
  - The effect is of rounding off sharp edges.



# Closing

- Dilation followed by erosion.
  - The effect is of closing of narrow gaps and holes.



# contents

- Image representation
- Pixel-wise operations
- Noise and filtering
- Frequency representation
- Decimation
- Interpolation
- Morphology operators
- **Connected components**

# Connected components

- Defined as regions of adjacent pixels that have the same value.
- Commonly used with binary images to find stand alone objects.
  - e.g.: letters in a document.