Digital Image Basics

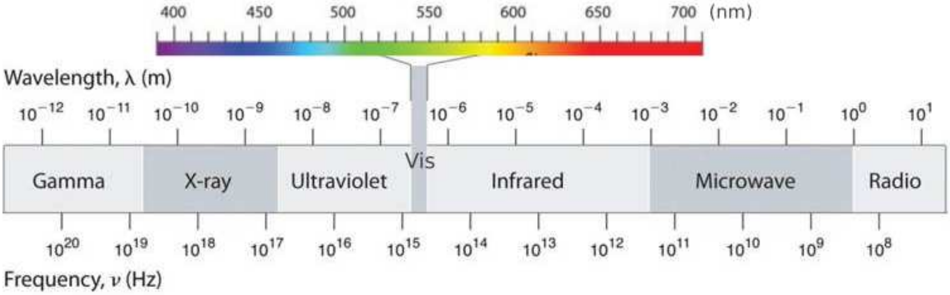
Lecture 02

Computer Vision for Geosciences

2021-03-05

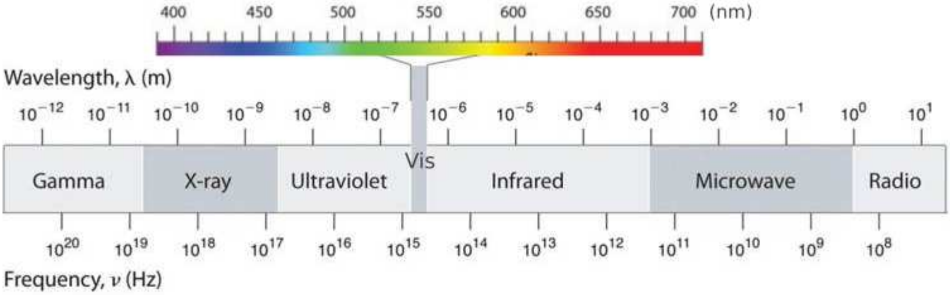
Universidad Nacional Autónoma de México

Images can be constructed using the entire electromagnetic spectra

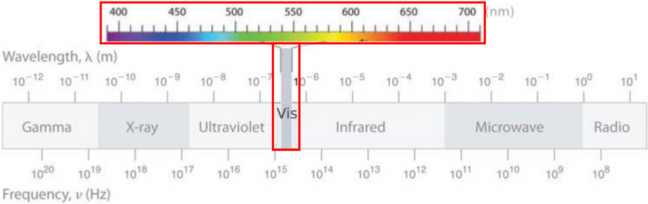


... a few examples in geosciences ...

Images can be constructed using the entire electromagnetic spectra



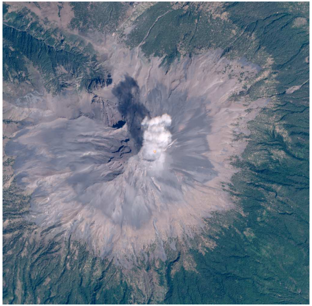
... a few examples in geosciences ...



camera

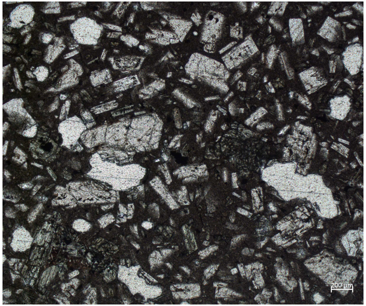
Popocatépetl 2020-04-16

satellite

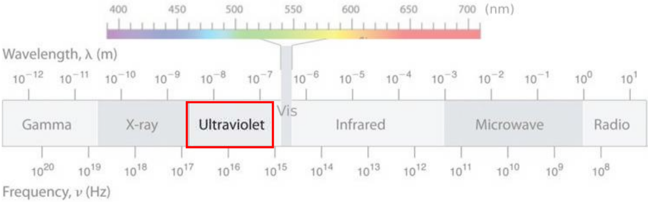


Popocatépetl 2021-02-25 (Sentinel-2, MOUNTS)

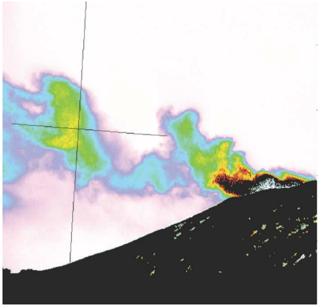
microscope



Popocatépetl 2019-01-22 (andesite, ©T.Boulesteix)

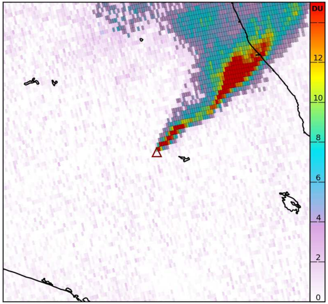


camera

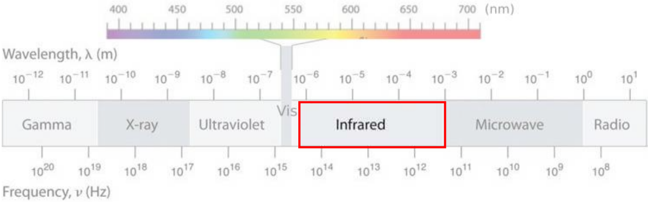


Popocatépetl 2013-01-29 (UV camera, Campion et al. 2018)

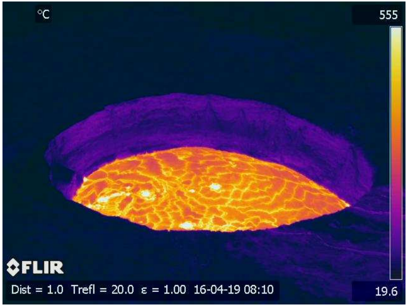
satellite



Popocatépetl 2019-02-17 (Sentinel-5P, MOUNTS)

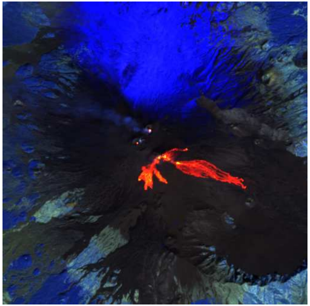


camera

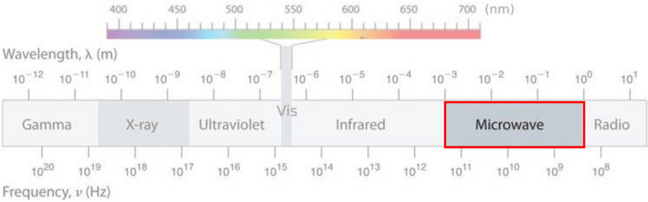


Nyiragongo 2016-04-16 (FLIR image, Valade et al. 2018)

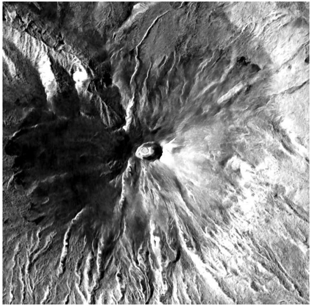
satellite



Etna 2021-02-23 (Sentinel-2 image, MOUNTS)

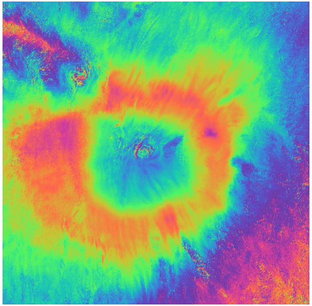


satellite (SAR)



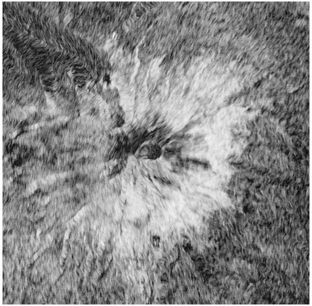
Popocatépetl 2021-02-28 (Sentinel-1, MOUNTS)

satellite (InSAR)

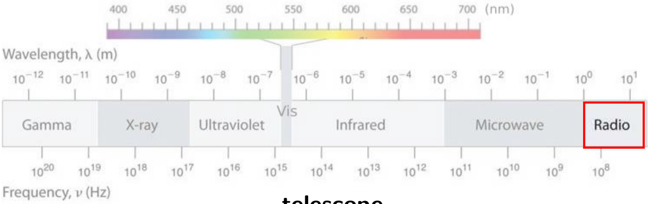


Popocatépetl InSAR interferogram (MOUNTS)

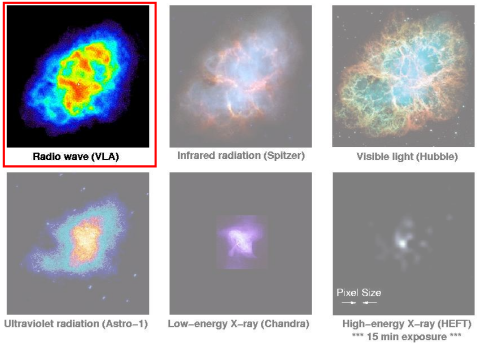
satellite (InSAR)



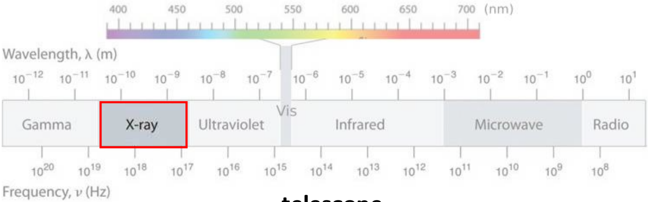
Popocatépetl InSAR coherence (MOUNTS)



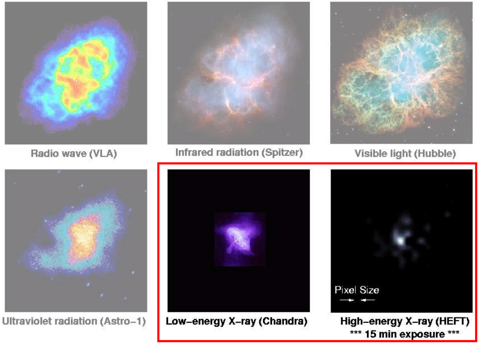
telescope



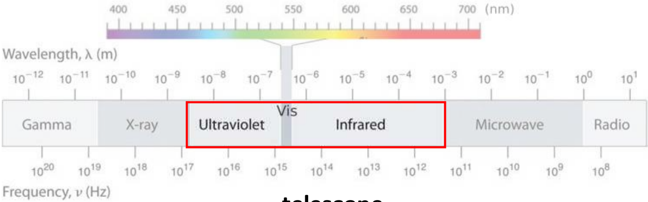
Crab Nebula - remanent of an exploded star (supernova)



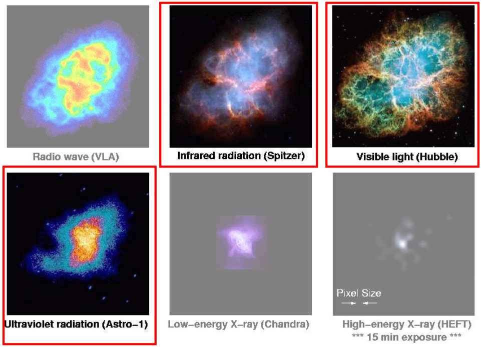
telescope



Crab Nebula - remanent of an exploded star (supernova)



telescope



Crab Nebula - remanent of an exploded star (supernova)

Comparison between human eye and pinhole camera

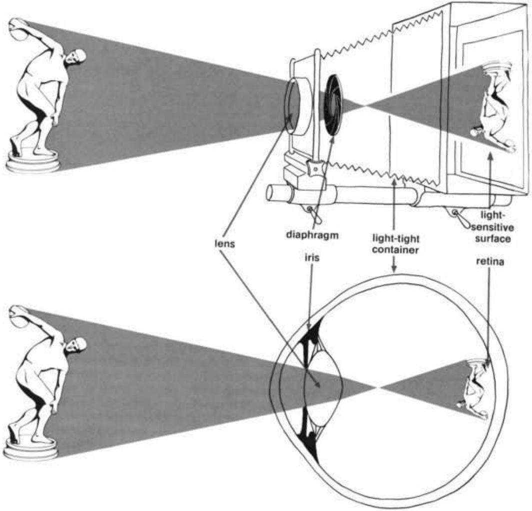
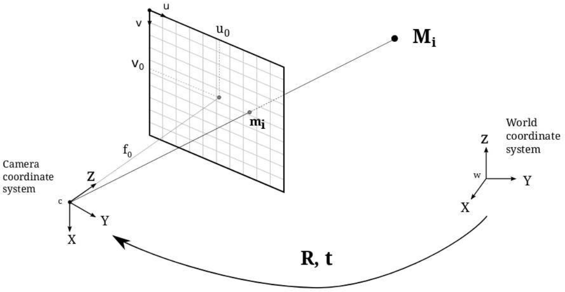


Image $=$ 3D world projection on 2D
$\Rightarrow$ projection using the **pinhole camera** model:



(from PyTorch Geometry)

Perspective transformation:

$$s\ m' = K[R|t]M' \tag{1}$$

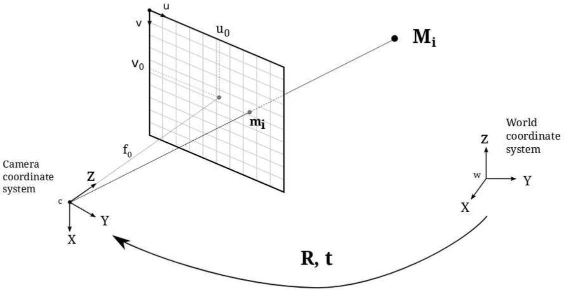
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

where:

- M' $=$ 3D point in space with coordinates $[X, Y, Z]^T$ expressed in Euclidean coordinates
- m' $=$ projection of the 3D point M' onto the image plane with coordinates $[u, v]^T$ expressed in pixel units
- K $=$ camera calibration matrix (a.k.a instrinsics parameters matrix)
  - fx, fy $=$ focal lengths expressed in pixel units
  - u_0, v_0 $=$ coordinates of the optical center (aka principal point), origin in the image plane
- $[R|t]$ $=$ joint rotation-translation matrix (a.k.a. extrinsics parameters matrix), describing the camera pose, and translating from world coordinates to camera coordinates

Image = 3D world projection on 2D

$\Rightarrow$ projection using the **pinhole camera** model:



(from PyTorch Geometry)

Perspective transformation:

$$s\ m' = K[R|t]M' \tag{1}$$

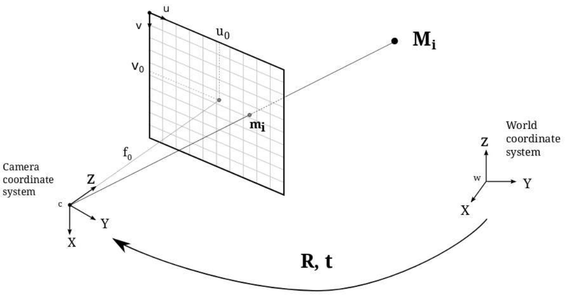
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

where:

- M' = 3D point in space with coordinates $[X, Y, Z]^T$ expressed in Euclidean coordinates
- m' = projection of the 3D point M' onto the image plane with coordinates $[u, v]^T$ expressed in pixel units
- K = camera calibration matrix (a.k.a instrinsics parameters matrix)
  - fx, fy = focal lengths expressed in pixel units
  - u_0, v_0 = coordinates of the optical center (aka principal point), origin in the image plane
- $[R|t]$ = joint rotation-translation matrix (a.k.a. extrinsics parameters matrix), describing the camera pose, and translating from world coordinates to camera coordinates

Image = 3D world projection on 2D

⇒ projection using the **pinhole camera** model:



(from PyTorch Geometry)

Perspective transformation:

$$s\ m' = K[R|t]M' \tag{1}$$

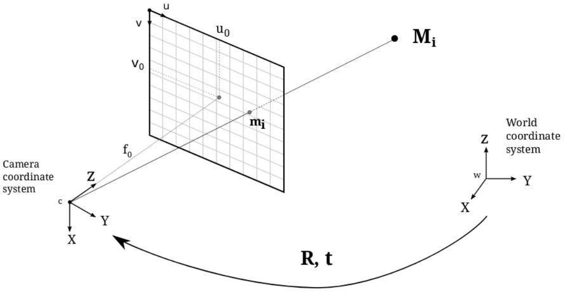
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

where:

- M' = 3D point in space with coordinates $[X, Y, Z]^T$ expressed in Euclidean coordinates
- m' = projection of the 3D point M' onto the image plane with coordinates $[u, v]^T$ expressed in pixel units
- K = camera calibration matrix (a.k.a instrinsics parameters matrix)
  - fx, fy = focal lengths expressed in pixel units
  - u_0, v_0 = coordinates of the optical center (aka principal point), origin in the image plane
- $[R|t]$ = joint rotation-translation matrix (a.k.a. extrinsics parameters matrix), describing the camera pose, and translating from world coordinates to camera coordinates

Image = 3D world projection on 2D

⇒ projection using the **pinhole camera** model:



(from PyTorch Geometry)

Perspective transformation:

$$s\ m' = K[R|t]M' \tag{1}$$

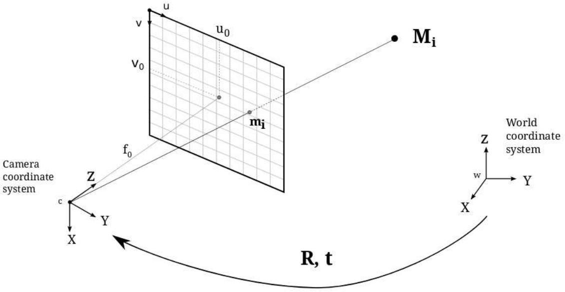
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

where:

- M' = 3D point in space with coordinates $[X, Y, Z]^T$ expressed in Euclidean coordinates
- m' = projection of the 3D point M' onto the image plane with coordinates $[u, v]^T$ expressed in pixel units
- K = camera calibration matrix (a.k.a instrinsics parameters matrix)
  - fx, fy = focal lengths expressed in pixel units
  - u_0, v_0 = coordinates of the optical center (aka principal point), origin in the image plane
- $[R|t]$ = joint rotation-translation matrix (a.k.a. extrinsics parameters matrix), describing the camera pose, and translating from world coordinates to camera coordinates

Image = 3D world projection on 2D

$\Rightarrow$ projection using the **pinhole camera** model:



(from PyTorch Geometry)

Perspective transformation:

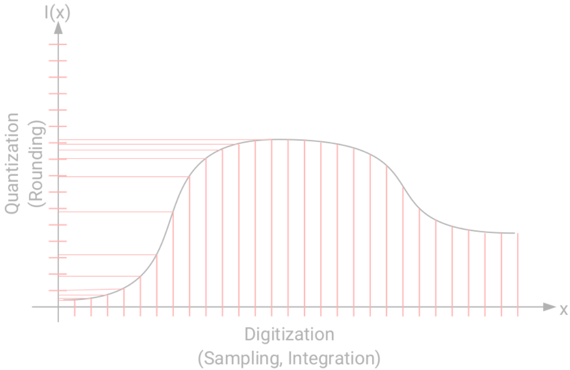
$$s\ m' = K[R|t]M' \tag{1}$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2}$$

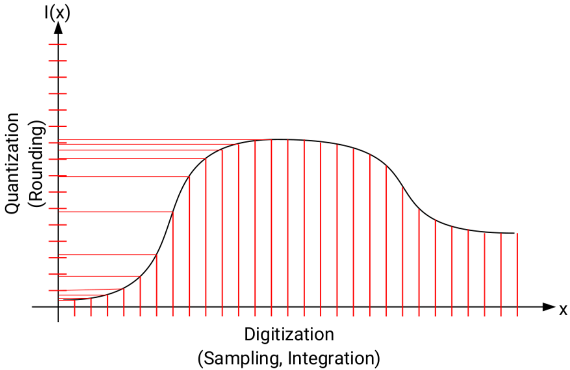
where:

- M' = 3D point in space with coordinates $[X, Y, Z]^T$ expressed in Euclidean coordinates
- m' = projection of the 3D point M' onto the image plane with coordinates $[u, v]^T$ expressed in pixel units
- K = camera calibration matrix (a.k.a instrinsics parameters matrix)
  - fx, fy = focal lengths expressed in pixel units
  - u_0, v_0 = coordinates of the optical center (aka principal point), origin in the image plane
- $[R|t]$ = joint rotation-translation matrix (a.k.a. extrinsics parameters matrix), describing the camera pose, and translating from world coordinates to camera coordinates

- at each point we record incident light
- digitalization of an analog signal involves two operations
  - **spatial sampling** (= discretization of space domain)
  - **intensity quantization** (= discretization of incoming light signal)



- at each point we record incident light
- digitalization of an analog signal involves two operations
  - **spatial sampling** (= discretization of space domain)
  - **intensity quantization** (= discretization of incoming light signal)



spatial sampling (= discretization of space domain)

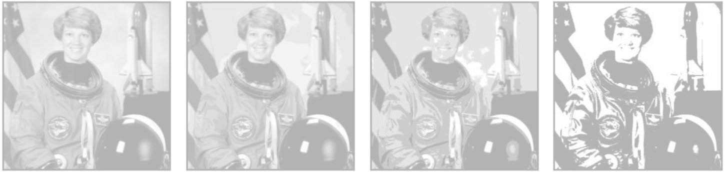
⇒ smallest element resulting from the discretization of the space is called a pixel (=picture element)

(512, 512) (128, 128) (64, 64) (32, 32)

intensity quantization (= discretization of light intensity signal)

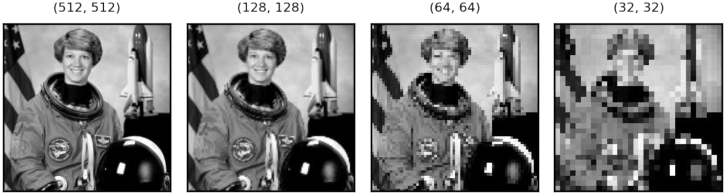
⇒ typically, 256 levels (8 bits/pixel = 2^8 values) suffices to represent the intensity

8-bit resolution $2^8 = 256$ gray levels | 3-bit resolution $2^3 = 8$ gray levels | 2-bit resolution $2^2 = 4$ gray levels | 1-bit resolution $2^1 = 2$ gray levels



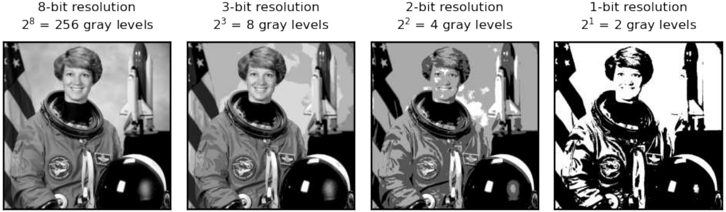
spatial sampling (= discretization of space domain)

⇒ smallest element resulting from the discretization of the space is called a pixel (=picture element)

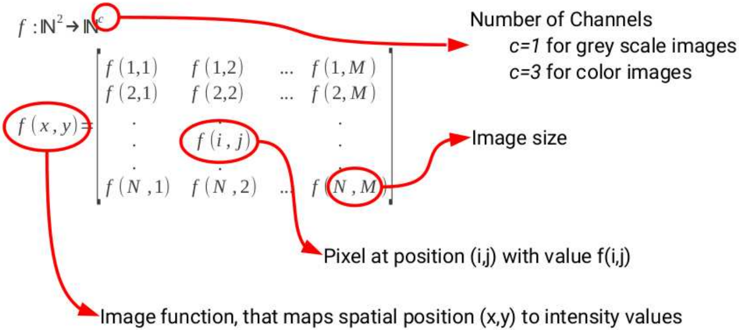


intensity quantization (= discretization of light intensity signal)

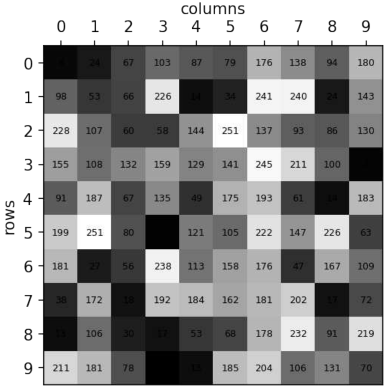
⇒ typically, 256 levels (8 bits/pixel = 2^8 values) suffices to represent the intensity



$\Rightarrow$ digital image function $f(x, y)$



$\Rightarrow$ digital image function $f(x, y)$

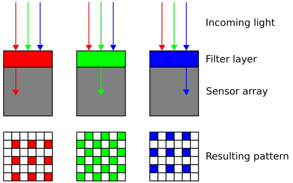


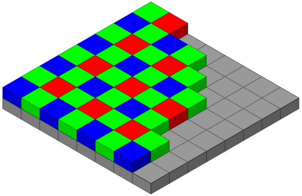
Typical ranges:

- uint8 = [0-255]
  (8 bits = 1 byte = 2^8 = 256 values per pixel)
- float32 = [0-1]
  (32 bits = 4 bytes = 4.3e9 values per pixel)

How do we record colors?

$\Rightarrow$ **Bayer Filter**: color filter array for arranging RGB color filters on a square grid of photosensors





(source wikipedia)

How do we record colors?

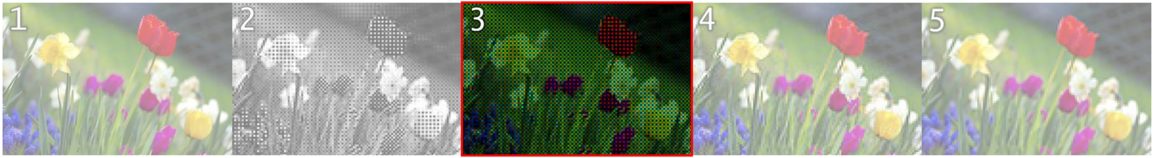
⇒ **Bayer Filter**: color filter array for arranging RGB color filters on a square grid of photosensors

1. Original scene
2. Output of a 120×80-pixel sensor with a Bayer filter
3. Output color-coded with Bayer filter colors
4. Reconstructed image after interpolating missing color information (a.k.a. demosaicing)
5. Full RGB version at 120×80-pixels for comparison

How do we record colors?
⇒ **Bayer Filter**: color filter array for arranging RGB color filters on a square grid of photosensors

1. Original scene
2. Output of a 120×80-pixel sensor with a Bayer filter
3. Output color-coded with Bayer filter colors
4. Reconstructed image after interpolating missing color information (a.k.a. demosaicing)
5. Full RGB version at 120×80-pixels for comparison

How do we record colors?
⇒ **Bayer Filter**: color filter array for arranging RGB color filters on a square grid of photosensors



1. Original scene
2. Output of a 120×80-pixel sensor with a Bayer filter
3. Output color-coded with Bayer filter colors
4. Reconstructed image after interpolating missing color information (a.k.a. demosaicing)
5. Full RGB version at 120×80-pixels for comparison

How do we record colors?

⇒ **Bayer Filter**: color filter array for arranging RGB color filters on a square grid of photosensors

1. Original scene
2. Output of a 120×80-pixel sensor with a Bayer filter
3. Output color-coded with Bayer filter colors
4. Reconstructed image after interpolating missing color information (a.k.a. demosaicing)
5. Full RGB version at 120×80-pixels for comparison

How do we record colors?

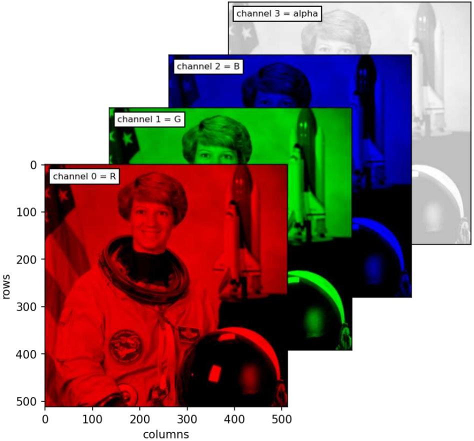
⇒ **Bayer Filter**: color filter array for arranging RGB color filters on a square grid of photosensors

1. Original scene
2. Output of a 120×80-pixel sensor with a Bayer filter
3. Output color-coded with Bayer filter colors
4. Reconstructed image after interpolating missing color information (a.k.a. demosaicing)
5. Full RGB version at 120×80-pixels for comparison

$\Rightarrow$ color image = 3D tensor in colorspace

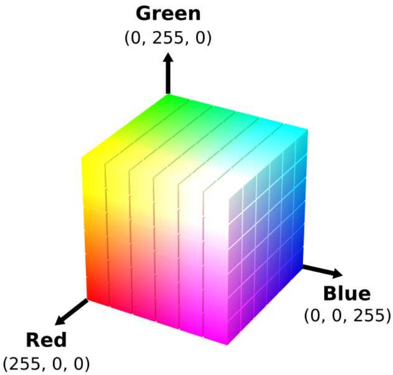
- **RGB** = Red + Green + Blue bands (.JPEG)
- **RGBA** = Red + Green + Blue + Alpha bands (.PNG, .GIF, .BMP, TIFF, .JPEG 2000)

$\longrightarrow$

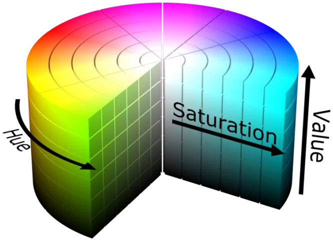


Other ways to represent the color information?

RGB colorspace



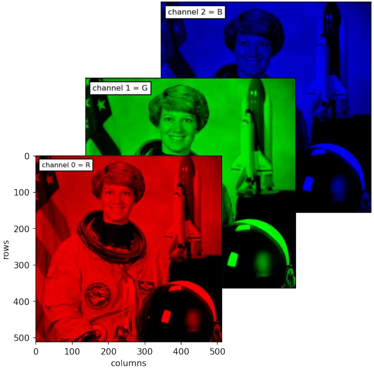
HSV colorspace



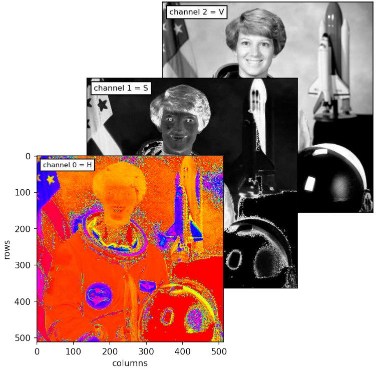
- Hue (H) = [0-360] $\Rightarrow$ shift color
- Saturation (S) = [0-1] $\Rightarrow$ shift intensity
- Value (V) = [0-1] $\Rightarrow$ shift brightness

3D tensor with different information

RGB colorspace



HSV colorspace



- more saturation S

$\Rightarrow$ more intense colors

- more value V

$\Rightarrow$ brighter colors

- shift hue H

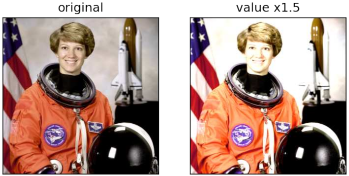
$\Rightarrow$ shift color

- more saturation S

⇒ more intense colors

- more value V

⇒ brighter colors



- shift hue H

⇒ shift color

- more saturation S

⇒ more intense colors

original

saturation x2

- more value V

⇒ brighter colors

original

value x1.5

- shift hue H

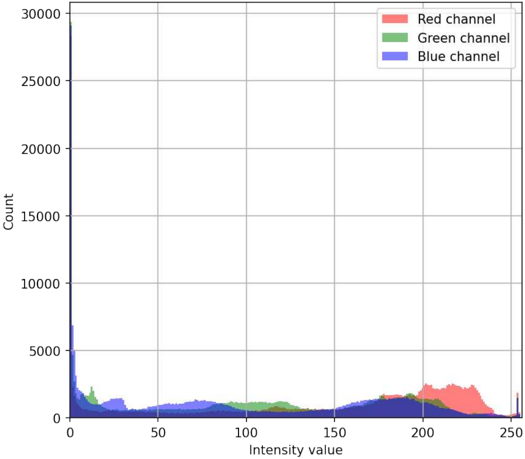
⇒ shift color

original

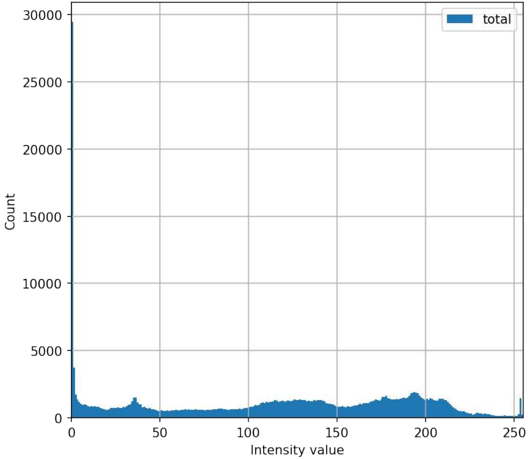
hue x5

Histogram of pixel values in each band:

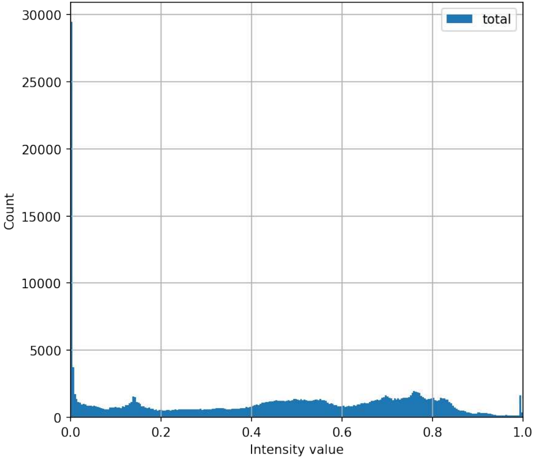
original (uint8)



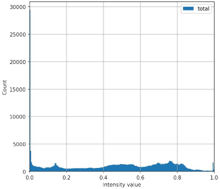
Histogram of pixel values after conversion from RGB (3-bands) to gray-scale (1-band):



Histogram of pixel values after conversion to float values (range [0-1])



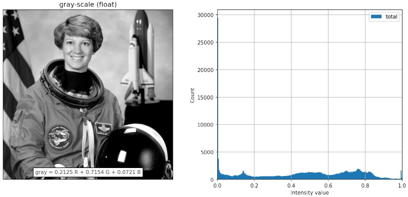
- original gray-scale



- histogram rescale to 10-90 percentiles
$\Rightarrow$ contrast stretching

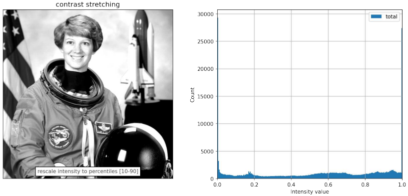
- histogram equalize
$\Rightarrow$ spread out the most frequent intensity values

- original gray-scale



- histogram rescale to 10-90 percentiles

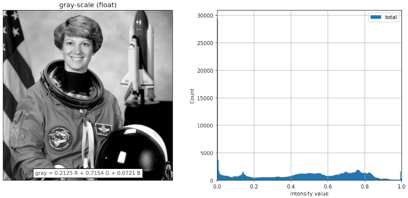
⇒ contrast stretching



- histogram equalize

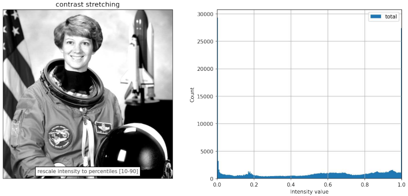
⇒ spread out the most frequent intensity values

- original gray-scale



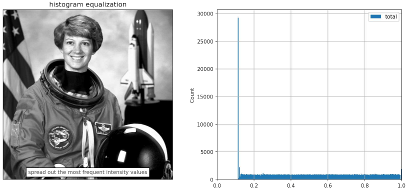
- histogram rescale to 10-90 percentiles

$\Rightarrow$ contrast stretching



- histogram equalize

$\Rightarrow$ spread out the most frequent intensity values



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

Value of original image at position (x,y)

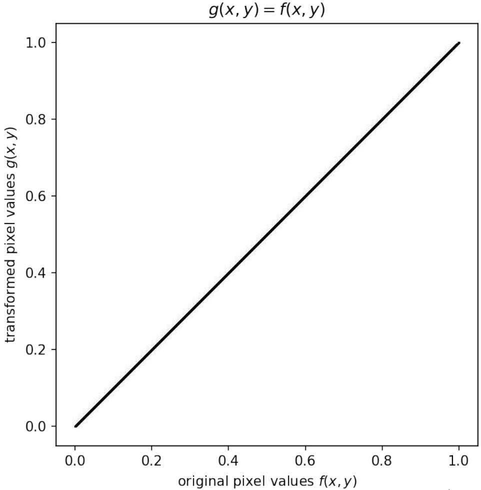
Operator function

Value of output image

Inhomogeneous PO: T is dependent on (x,y)
Homogeneous PO: T is NOT dependent on (x,y)

Homogeneous Point Operations (does not depend on pixel position)

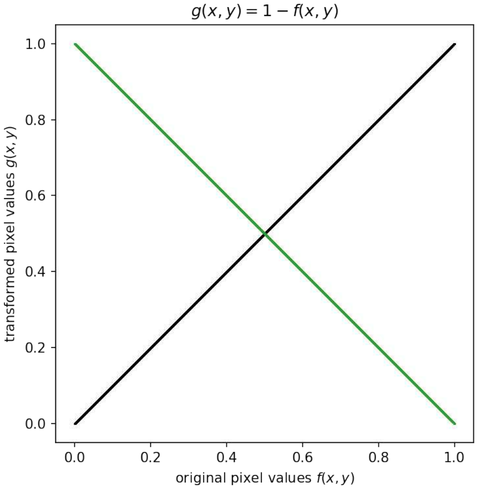
identity



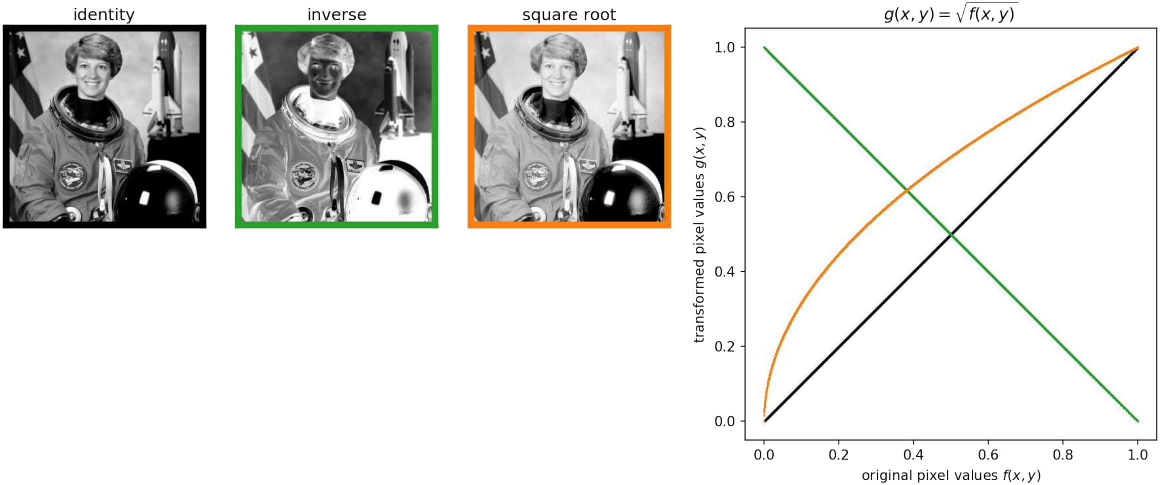
Homogeneous Point Operations (does not depend on pixel position)

identity

inverse



Homogeneous Point Operations (does not depend on pixel position)



Homogeneous Point Operations (does not depend on pixel position)

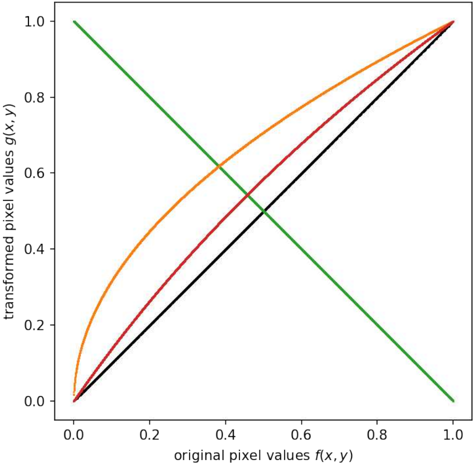
identity

inverse

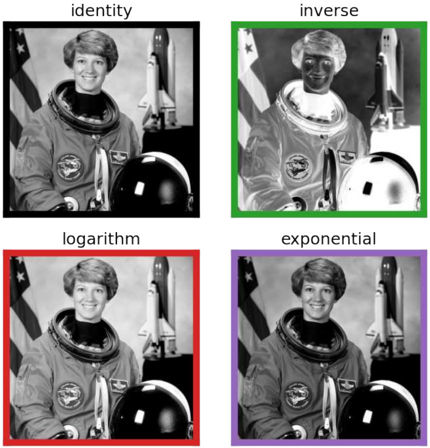
square root

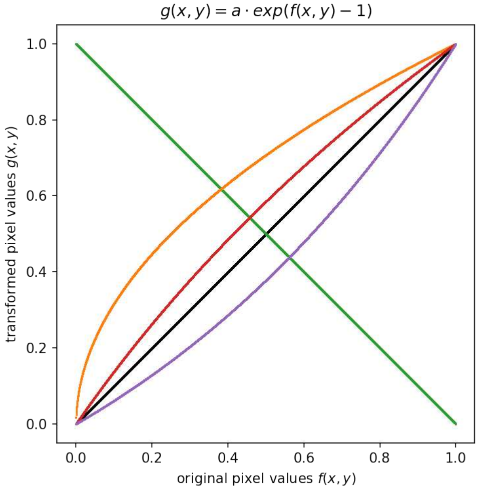
logarithm

$g(x, y) = a \cdot \log(f(x, y) + 1)$

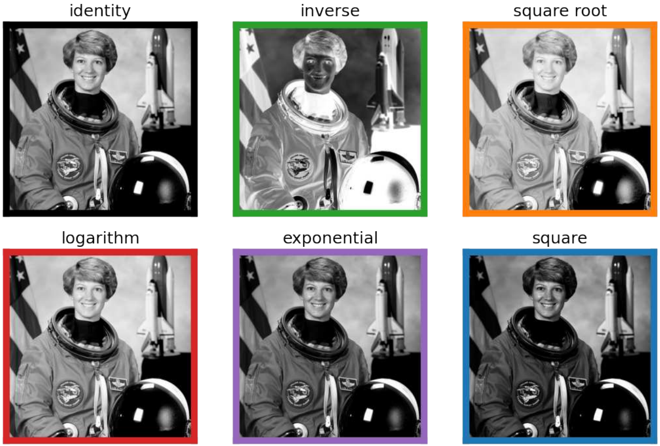


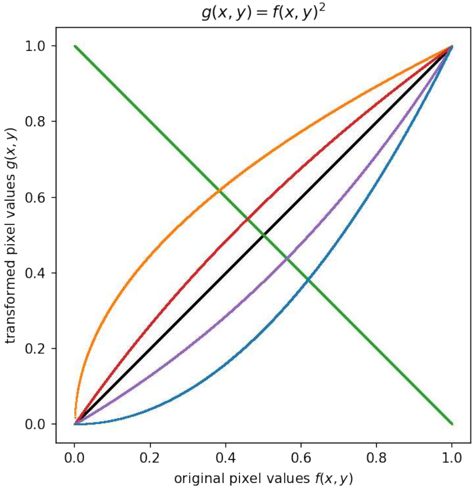
Homogeneous Point Operations (does not depend on pixel position)





Homogeneous Point Operations (does not depend on pixel position)





Inhomogeneous Point Operations (depends on pixel position)

EX: background detection / change detection

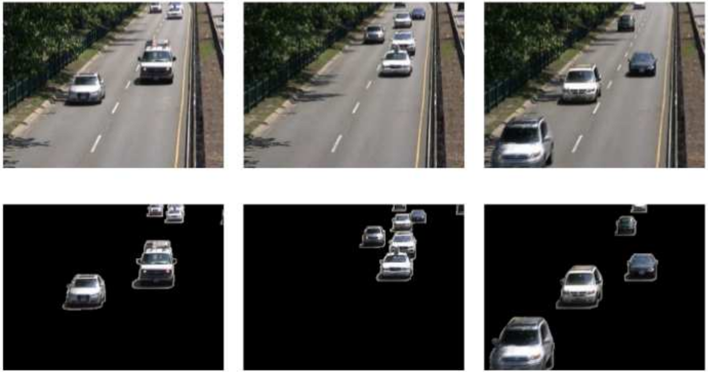
f_1 $\qquad$ f_i $\qquad$ f_N

$$a(x,y)=\frac{1}{N}\sum_{i=0}^{N} f_i(x,y)$$

$a(x,y)$

$$\begin{aligned} g_i(x,y) &= T(f(x,y),x,y) \\ &= f_i(x,y)-a(x,y) \end{aligned}$$

Inhomogeneous Point Operations (depends on pixel position)
EX: background detection / change detection



Computer Vision processing levels:

- Low-level vision
  - image manipulation
    (resizing, color adjustments, ...)
  - feature extraction
    (edges, gradients, ...)
- Mid-level vision
  - panorama stitching
  - Structure from Motion (SfM) $\Rightarrow$ 2D to 3D
  - Optical Flow $\Rightarrow$ velocities
- High-level vision
  - classification: what is in the image?
  - tagging: what are ALL the things in the image?
  - detection: where are they?
  - semantic segmentation $\Rightarrow$ segment image and give names

Computer Vision processing levels:

- Low-level vision
  - image manipulation
    (resizing, color adjustments, ...)
  - feature extraction
    (edges, gradients, ...)
- Mid-level vision
  - panorama stitching
  - Structure from Motion (SfM) $\Rightarrow$ 2D to 3D
  - Optical Flow $\Rightarrow$ velocities
- High-level vision
  - classification: what is in the image?
  - tagging: what are ALL the things in the image?
  - detection: where are they?
  - semantic segmentation $\Rightarrow$ segment image and give names

Computer Vision processing levels:

- Low-level vision
  - image manipulation
    (resizing, color adjustments, ...)
  - feature extraction
    (edges, gradients, ...)
- Mid-level vision
  - panorama stitching
  - Structure from Motion (SfM) $\Rightarrow$ 2D to 3D
  - Optical Flow $\Rightarrow$ velocities
- High-level vision
  - classification: what is in the image?
  - tagging: what are ALL the things in the image?
  - detection: where are they?
  - semantic segmentation $\Rightarrow$ segment image and give names

Computer Vision processing levels:

- Low-level vision
  - image manipulation
    (resizing, color adjustments, ...)
  - feature extraction
    (edges, gradients, ...)
- Mid-level vision
  - panorama stitching
  - Structure from Motion (SfM) $\Rightarrow$ 2D to 3D
  - Optical Flow $\Rightarrow$ velocities
- High-level vision
  - classification: what is in the image?
  - tagging: what are ALL the things in the image?
  - detection: where are they?
  - semantic segmentation $\Rightarrow$ segment image and give names

In Binder:

$\Rightarrow$ Open CV4GS_02_imagebasics/CV4GS_02_numpy-tutorial.ipynb

$\Rightarrow$ Open CV4GS_02_imagebasics/CV4GS_02_exercices.ipynb

In Binder:

⇒ Open CV4GS_02_imagebasics/CV4GS_02_numpy-tutorial.ipynb

⇒ Open CV4GS_02_imagebasics/CV4GS_02_exercices.ipynb