

Lecture 02

Digital Image Basics

2026-02-27

Sébastien Valade



1. Motivation

1. Why Computer Vision for Geosciences?
2. Computer Vision processing levels

2. What is a digital image?

3. Point operations

4. Image manipulation with Python

Computer Vision for Geosciences (CV4GS)

Computer Vision for Geosciences (CV4GS)

- What is Computer Vision?

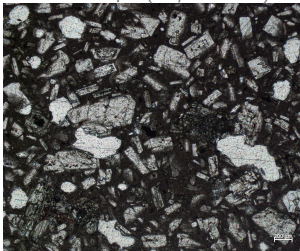
⇒ discipline focused on enabling computers to acquire, process, and interpret visual data, primarily from digital images or video

Computer Vision for Geosciences (CV4GS)

- Sources of images for geosciences applications?

⇒ images can be derived from different imaging techniques at different scales

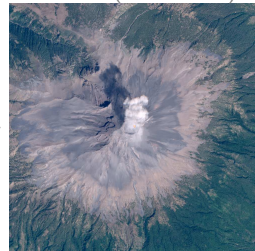
microscope ($\sim \mu m$ -scale)



camera ($\sim m$ -scale)

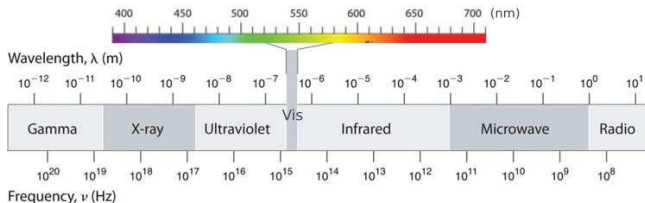


satellite ($\sim km$ -scale)

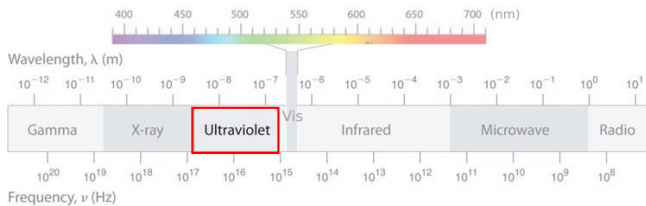


Computer Vision for Geosciences (CV4GS)

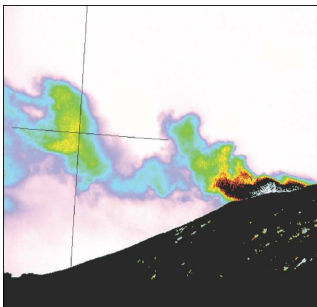
- Sources of images for geosciences applications?
 - ⇒ images can be constructed using different wavelengths spanning the entire electromagnetic spectra



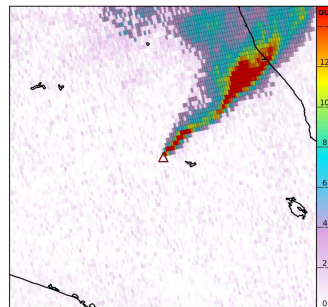
1.1. Why Computer Vision for Geosciences?



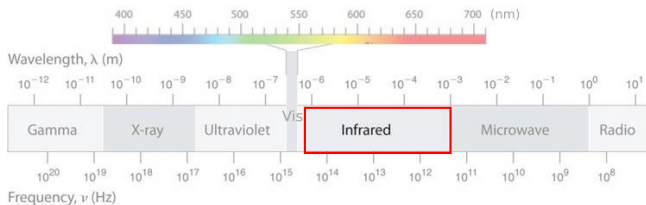
UV camera

Popocatepetl 2013-01-29 (UV camera, [Campion et al. 2018](#))

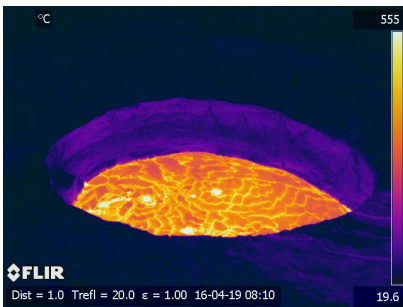
UV satellite

Popocatepetl 2019-02-17 (Sentinel-5P, [MOUNTS](#))

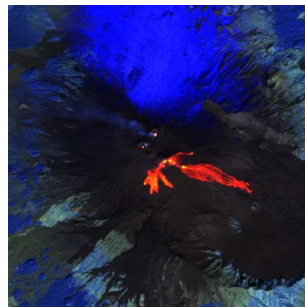
1.1. Why Computer Vision for Geosciences?



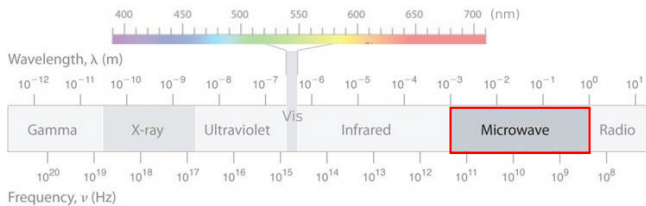
IR camera

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

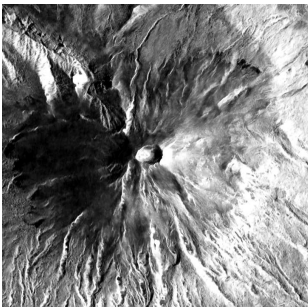
IR satellite

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

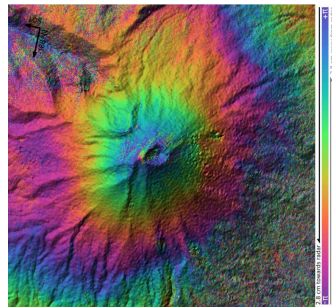
1.1. Why Computer Vision for Geosciences?



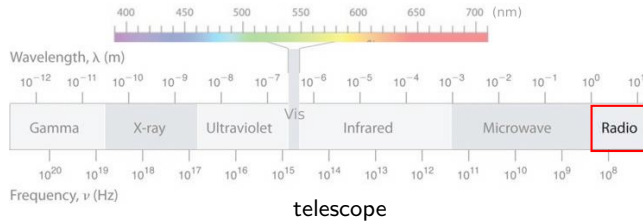
SAR satellite

Popocatepetl SAR 2021-12-25 (Sentinel-1, [MOUNTS](#))

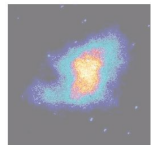
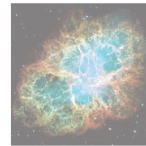
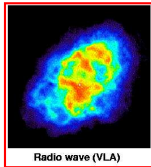
InSAR satellite

Popocatepetl InSAR interferogram 2021-12-25 dt=6 days ([MOUNTS](#))

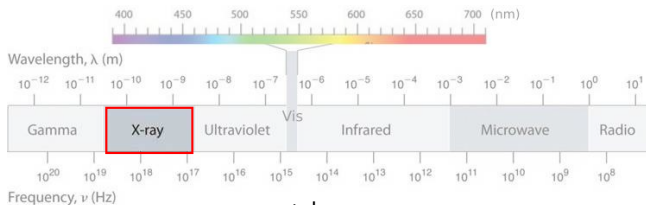
1.1. Why Computer Vision for Geosciences?



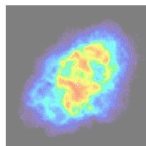
telescope



1.1. Why Computer Vision for Geosciences?



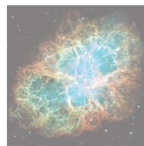
telescope



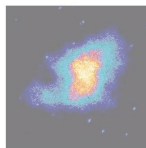
Radio wave (VLA)



Infrared radiation (Spitzer)



Visible light (Hubble)



Ultraviolet radiation (Astro-1)

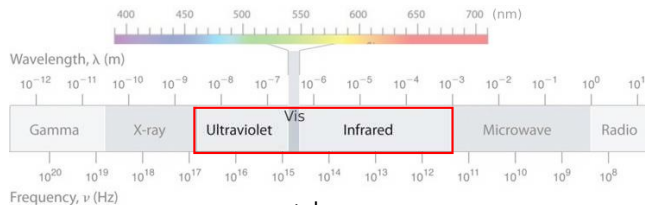


Low-energy X-ray (Chandra)

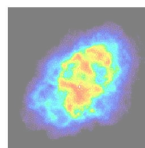
High-energy X-ray (HEFT)
*** 15 min exposure ***

Crab Nebula - remanent of an exploded star (supernova)

1.1. Why Computer Vision for Geosciences?



telescope



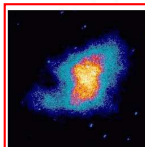
Radio wave (VLA)



Infrared radiation (Spitzer)



Visible light (Hubble)



Ultraviolet radiation (Astro-1)



Low-energy X-ray (Chandra)



High-energy X-ray (HEFT)

Pixel Size

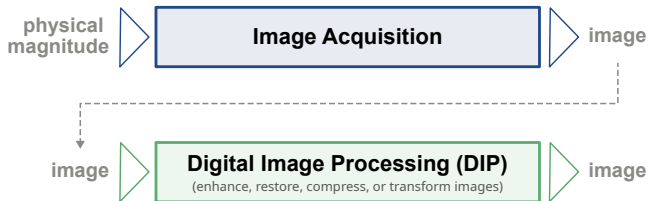
*** 15 min exposure ***

Crab Nebula - remanent of an exploded star (supernova)

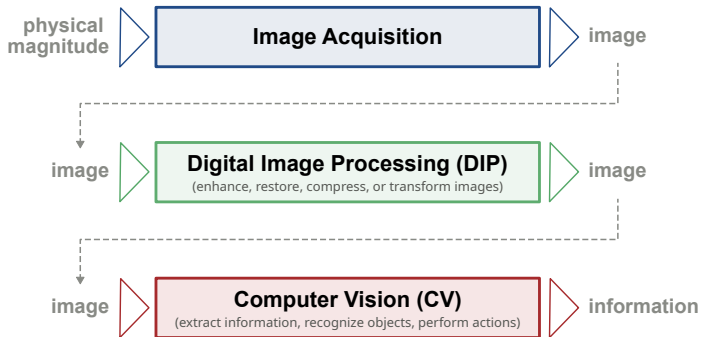
From image acquisition to image processing:



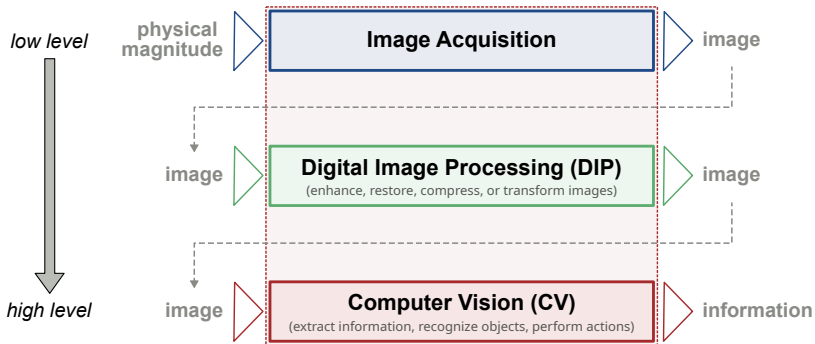
From image acquisition to image processing:



From image acquisition to image processing:



From image acquisition to image processing:



*“**Computer Vision** tasks include methods for **acquiring**, **processing**, **analyzing** and understanding digital images, and extraction of high-dimensional data from the real world in order to produce numerical or symbolic information, e.g., in the forms of decisions”.*

Examples of processing levels:

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

Examples of processing levels:

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



hue x5
→



filter (high pass)
→



Examples of processing levels:

- Low-level processing

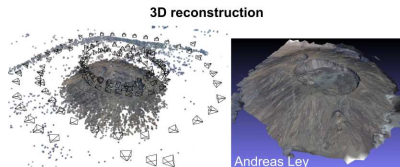
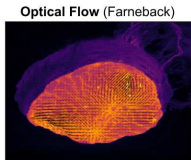
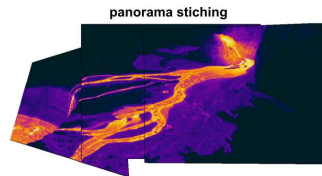
- image manipulation $\Rightarrow$ *resizing, color adjustments, filtering, etc.*
- feature extraction $\Rightarrow$ *edges, gradients, etc.*

- Mid-level processing

- panorama stitching
- Structure from Motion (SfM) $\Rightarrow$ 2D to 3D
- Optical Flow $\Rightarrow$ velocities

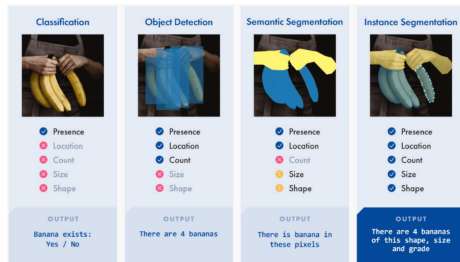
- High-level processing

- classification $\Rightarrow$ *what is in the image?*
- detection $\Rightarrow$ *where are they?*
- segmentation (semantic or instance) $\Rightarrow$ *segment image and give names*



Examples of processing levels:

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



Credit: cloudfactory

Examples of processing levels:

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

1. Motivation

2. What is a digital image?

1. image acquisition
2. sampling and quantization
3. 3D projection on 2D plane
4. color image
5. color spaces
6. image histogram

3. Point operations

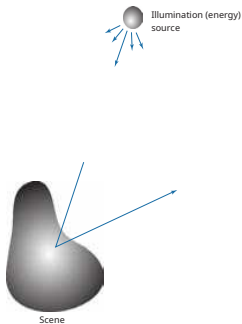
4. Image manipulation with Python

How is an image acquired?

1. energy from an **illumination source** is reflected from a **scene**
2. the **imaging system** collects the incoming energy and focuses it onto an **image plane**

NB: light-sensing instruments typically use 2-D arrays of photosensors to record incoming light intensity $I(x)$: the CCD (Charge-Coupled Device)

3. the image plane is sampled and quantized to produce a **digital image**

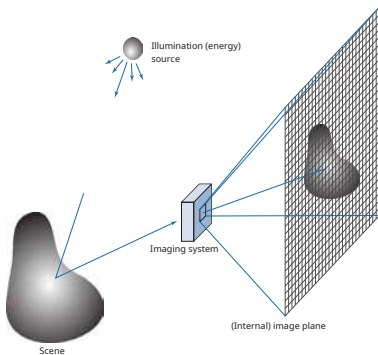


How is an image acquired?

1. energy from an **illumination source** is reflected from a **scene**
2. the **imaging system** collects the incoming energy and focuses it onto an **image plane**

NB: light-sensing instruments typically use 2-D arrays of photosensors to record incoming light intensity $I(x)$: the CCD (Charge-Coupled Device)

3. the image plane is sampled and quantized to produce a **digital image**

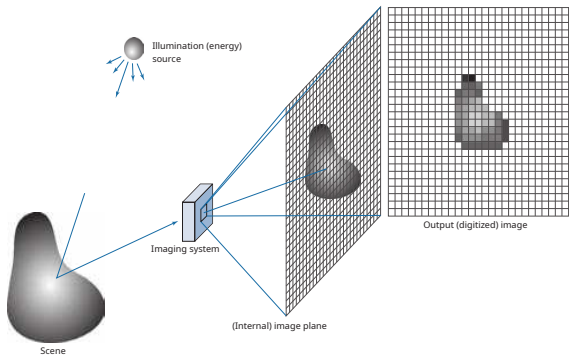


How is an image acquired?

1. energy from an **illumination source** is reflected from a **scene**
2. the **imaging system** collects the incoming energy and focuses it onto an **image plane**

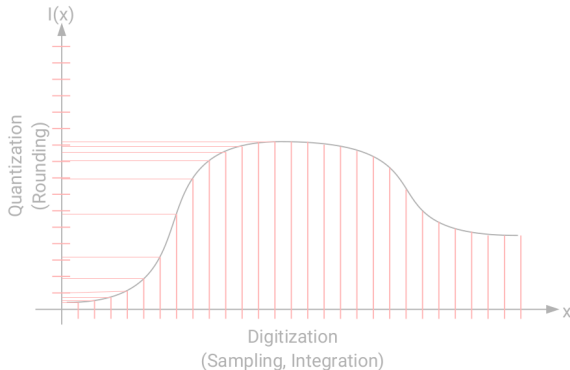
NB: light-sensing instruments typically use 2-D arrays of photosensors to record incoming light intensity $I(x)$: the CCD (Charge-Coupled Device)

3. the image plane is sampled and quantized to produce a **digital image**



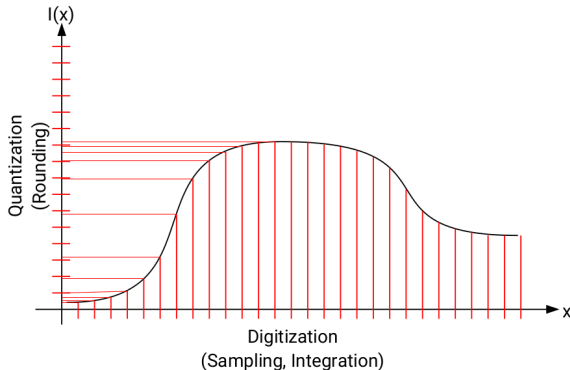
Sampling and quantization of incident light

- each photosensor records incident light
- digitalization of an analog signal involves two operations
 1. spatial sampling (= discretization of space domain)
 2. intensity quantization (= discretization of incoming light signal)



Sampling and quantization of incident light

- each photosensor records incident light
- digitalization of an analog signal involves two operations
 1. spatial sampling (= discretization of space domain)
 2. intensity quantization (= discretization of incoming light signal)



2.2. sampling and quantization

1. 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)

2. 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



1. 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)

2. 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

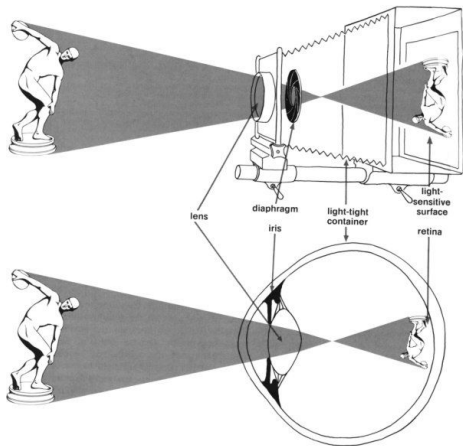


1-bit resolution
 $2^1 = 2$ gray levels



But how is the 3D world projected on a 2D plane?

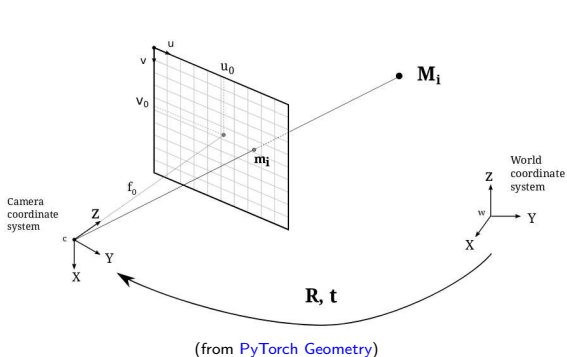
⇒ comparison between human eye and pinhole camera:



In 1514, Leonardo da Vinci explained: "By letting the images of illuminated objects penetrate through a small hole into a very dark room, you will then intercept these images on a white sheet of paper placed in this room. [...] but they will be smaller and reversed".

But how is the 3D world projected on a 2D image plane?

- ⇒ simplest model: projection using the **pinhole camera** model
- ⇒ perspective transformation between 3D world coordinates M' and 2D image coordinates m'



$$s m' = K[R|t]M'$$

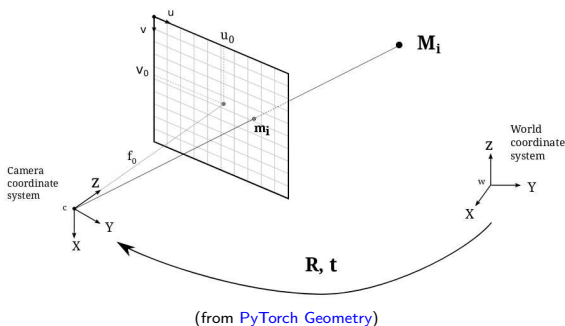
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad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

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
- $s =$ scale factor (depth-related)
- $K =$ camera calibration matrix (a.k.a intrinsics parameters matrix)
 - $f_x, f_y =$ 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

But how is the 3D world projected on a 2D image plane?

- ⇒ simplest model: projection using the **pinhole camera** model
- ⇒ perspective transformation between 3D world coordinates M' and 2D image coordinates m'



$$s m' = K[R|t]M'$$

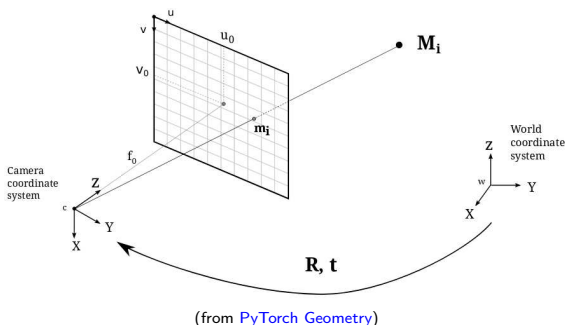
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad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

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
- $s =$ scale factor (depth-related)
- $K =$ camera calibration matrix (a.k.a intrinsics parameters matrix)
 - $f_x, f_y =$ 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

But how is the 3D world projected on a 2D image plane?

- ⇒ simplest model: projection using the **pinhole camera** model
- ⇒ perspective transformation between 3D world coordinates M' and 2D image coordinates m'



$$s m' = K[R|t]M'$$

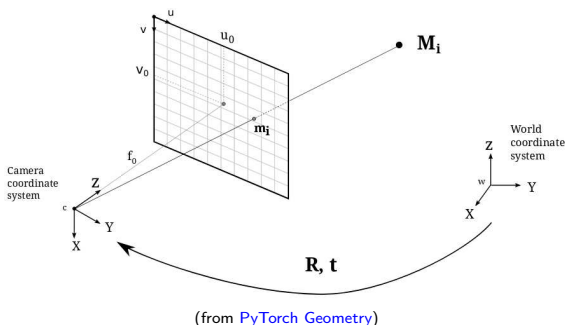
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad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

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
- $s =$ scale factor (depth-related)
- $K =$ camera calibration matrix (a.k.a intrinsics parameters matrix)
 - $f_x, f_y =$ 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

But how is the 3D world projected on a 2D image plane?

- ⇒ simplest model: projection using the **pinhole camera** model
- ⇒ perspective transformation between 3D world coordinates M' and 2D image coordinates m'



$$s m' = K[R|t]M'$$

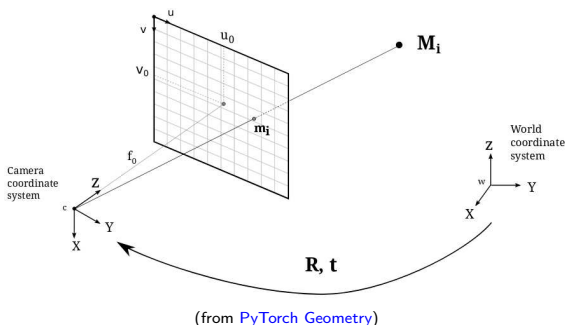
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad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

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
- $s =$ scale factor (depth-related)
- $K =$ camera calibration matrix (a.k.a. intrinsics parameters matrix)
 - $f_x, f_y =$ 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

But how is the 3D world projected on a 2D image plane?

- ⇒ simplest model: projection using the **pinhole camera** model
- ⇒ perspective transformation between 3D world coordinates M' and 2D image coordinates m'



$$s m' = K[R|t]M'$$

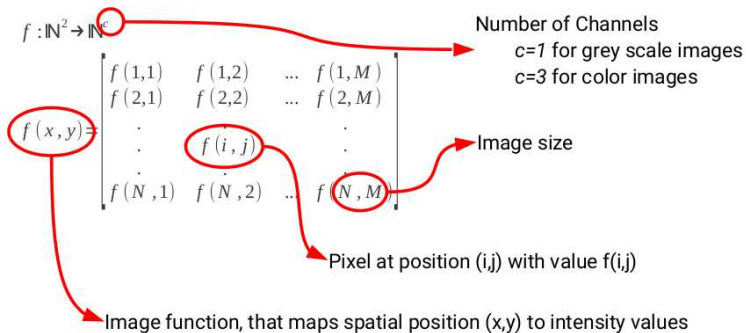
$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad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

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
- $s =$ scale factor (depth-related)
- $K =$ camera calibration matrix (a.k.a. intrinsics parameters matrix)
 - f_x, f_y = 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

Digital image

- ⇒ a digital image is a discrete representation of a continuous scene
- ⇒ it is a tensor of size $(N \times M \times C)$: N and M are the spatial dimensions, and C is the number of channels
- ⇒ the “image function” $f(x, y)$ maps spatial coordinates (x, y) to intensity values



Digital image

	columns									
	0	1	2	3	4	5	6	7	8	9
0	6	34	67	103	87	79	176	138	94	180
1	98	53	66	226	14	34	241	240	24	143
2	228	107	60	58	144	251	137	93	86	130
3	155	108	132	159	129	141	245	211	100	
4	91	187	67	135	49	175	193	61	14	183
5	199	251	80		121	105	222	147	226	63
6	181	27	56	238	113	158	176	47	167	109
7	38	172	18	192	184	162	181	202	57	72
8	11	106	30	17	53	68	178	232	91	219
9	211	181	78		11	185	204	106	131	70

Typical pixel values

⇒ depend on **bit depth** (*intensity quantization*):

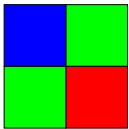
- **uint8** = [0-255] (0=black, 255=white)
bit depth = 8 bits (1 byte) $\Rightarrow 2^8 = 256$ values
- **float32** = [0-1] (0=black, 1=white)
bit depth = 32 bits (4 bytes) $\Rightarrow 2^{32} = 4.3e9$ values

How do we record colors?

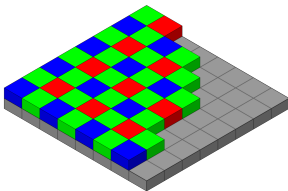
⇒ **Color Filter Array (CFA)**: a mosaic of color filters placed over the photosensors, such that each photosensor measures only a single color channel

⇒ **Bayer filter**: the most common CFA, used in most single-sensor color cameras

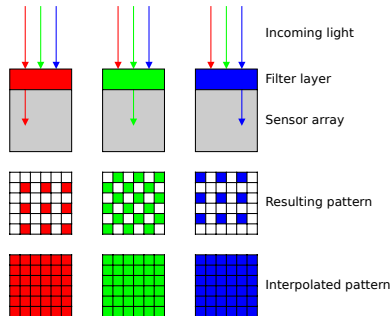
NB: the 2x2 pattern is 50% green, 25% red and 25% blue ⇒ more green photosensors are used in order to mimic the physiology of the human eye, which is more sensitive to green light.



Bayer filter pattern (2x2)

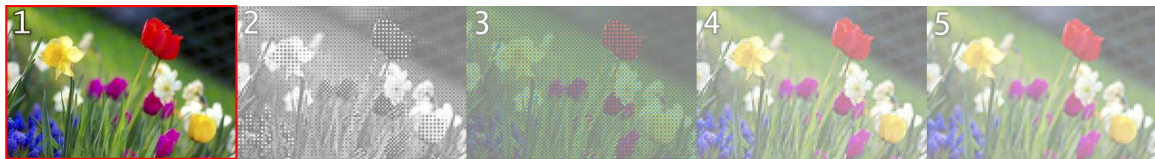


Bayer filter mosaic on an image sensor
(source [wikipedia](#))



How do we record colors?

⇒ **Bayer filter:** recording a real scene with Bayer filter



1. Original scene
2. Output of a 120×80 -pixel sensor with a Bayer filter (i.e. grayscaled light intensity)
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:** recording a real scene with Bayer filter



1. Original scene
2. Output of a 120×80 -pixel sensor with a Bayer filter (i.e. grayscaled light intensity)
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:** recording a real scene with Bayer filter



1. Original scene
2. Output of a 120×80 -pixel sensor with a Bayer filter (i.e. grayscaled light intensity)
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:** recording a real scene with Bayer filter



1. Original scene
2. Output of a 120×80 -pixel sensor with a Bayer filter (i.e. grayscaled light intensity)
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:** recording a real scene with Bayer filter

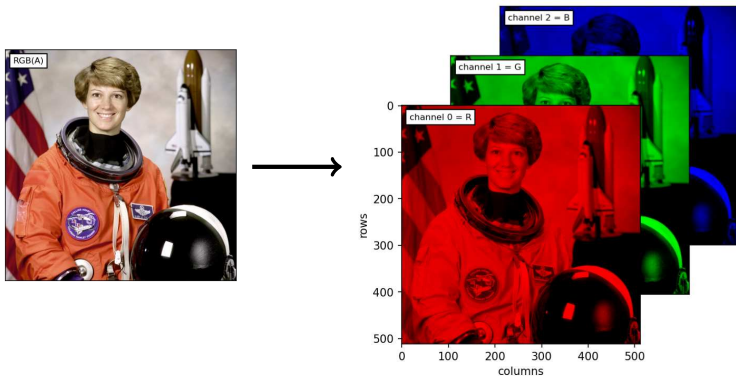


1. Original scene
2. Output of a 120×80 -pixel sensor with a Bayer filter (i.e. grayscaled light intensity)
3. Output color-coded with Bayer filter colors
4. Reconstructed image after interpolating missing color information (a.k.a. demosaicing)
5. Full RGB version at 120×80 -pixels for comparison

How do we record colors?

⇒ a color image is a **3D tensor** ($H \times W \times C$) storing the RGB colors in distinct **channels** (a.k.a. **bands**)

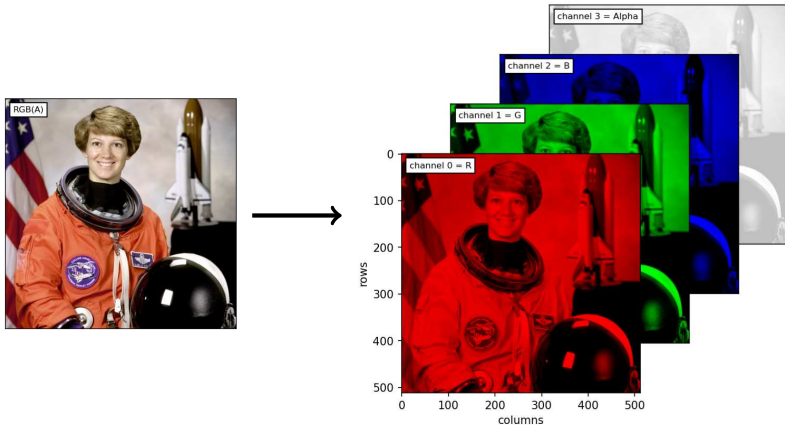
- **RGB** = Red + Green + Blue
- **RGBA** = Red + Green + Blue + Alpha



How do we record colors?

⇒ a color image is a **3D tensor** ($H \times W \times C$) storing the RGB colors in distinct **channels** (a.k.a. **bands**)

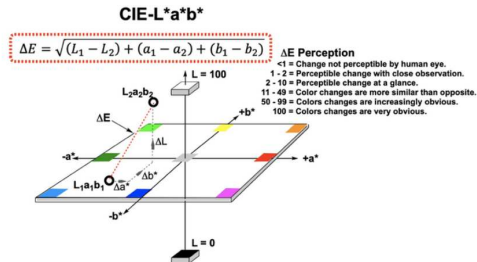
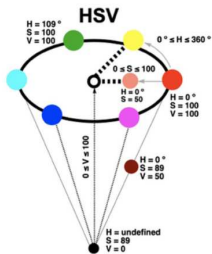
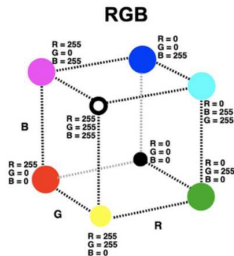
- **RGB** = Red + Green + Blue (*.JPG*)
- **RGBA** = Red + Green + Blue + Alpha (*.PNG, .GIF, .BMP, TIFF, .JPEG 2000*)



Color spaces: other ways to represent the color information

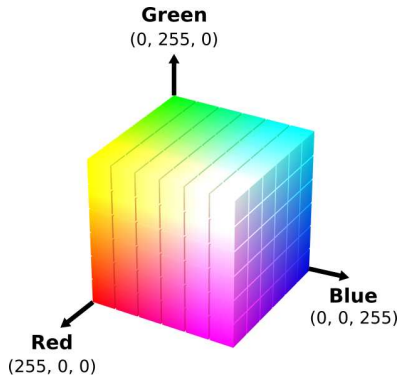
- ⇒ a **color space** is a mathematical coordinate system that represents colors as numerical values (coordinates)
- ⇒ multiple color models exist because different tasks require different physical or perceptual properties:

Model	Key property	Typical use
RGB	Additive light mixing	Cameras, displays
CMYK	Subtractive ink mixing	Printing
LAB	Perceptual uniformity	Color correction, calibration
HSV	Separated hue representation	Intuitive color manipulation

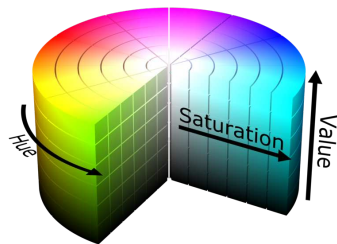


Color spaces: examples in RGB and HSV

RGB colorspace



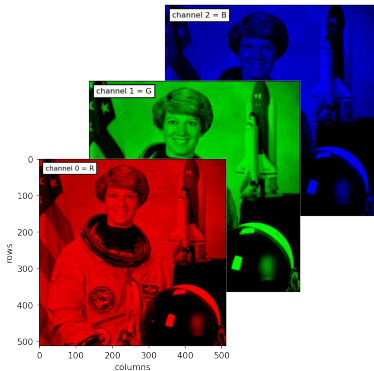
HSV colorspace



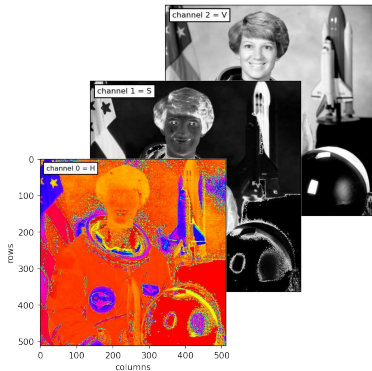
- Hue (H) = [0-360] $\Rightarrow$ shift color
- Saturation (S) = [0-1] $\Rightarrow$ shift intensity
- Value (V) = [0-1] $\Rightarrow$ shift brightness

3D tensor with different information:

RGB colorspace



HSV colorspace



HSV allows for more intuitive color adjustments:

- more saturation S
⇒ more intense colors



- more value V
⇒ brighter colors

- shift hue H
⇒ shift color

HSV allows for more intuitive color adjustments:

- more saturation S
⇒ more intense colors

original



saturation x2



original



value x1.5



- more value V
⇒ brighter colors

- shift hue H
⇒ shift color

HSV allows for more intuitive color adjustments:

- more saturation S
⇒ more intense colors

original



saturation x2



- more value V
⇒ brighter colors

original



value x1.5



- shift hue H
⇒ shift color

original



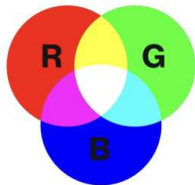
hue x5



Additive vs. Subtractive Color Models:

RGB (Additive Model)

- ⇒ used for devices that emit light (monitors, TVs, smartphones)
- ⇒ additive model: colors are created by combining different intensities of red, green, and blue light
 - combining all three colors at full intensity results in *white* light, absence of all results in *black*



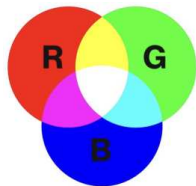
CMYK (Subtractive Model)

- ⇒ used for printing on paper and other physical media
- ⇒ subtractive color model: colors are created by subtracting light reflected off the paper
 - combining all three colors ideally absorb all light, resulting in *black* (NB: in printers black ink is added to achieve deeper blacks and reduce usage of the other inks)

Additive vs. Subtractive Color Models:

RGB (Additive Model)

- ⇒ used for devices that emit light (monitors, TVs, smartphones)
- ⇒ additive model: colors are created by combining different intensities of red, green, and blue light
 - combining all three colors at full intensity results in *white* light, absence of all results in *black*



CMYK (Subtractive Model)

- ⇒ used for printing on paper and other physical media
- ⇒ subtractive color model: colors are created by subtracting light reflected off the paper
 - combining all three colors ideally absorb all light, resulting in *black* (NB: in printers black ink is added to achieve deeper blacks and reduce usage of the other inks)

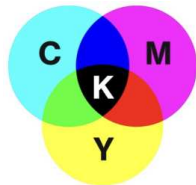


Image histogram

⇒ histogram of pixel values in each band:

original (uint8)

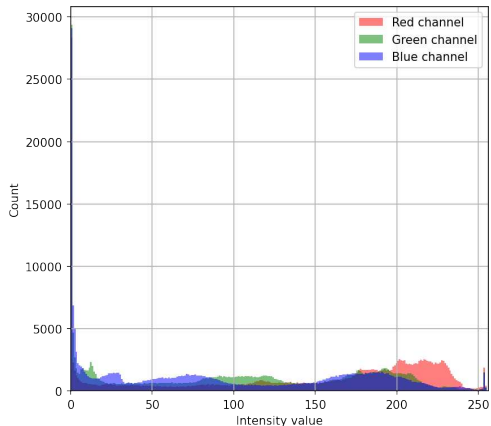
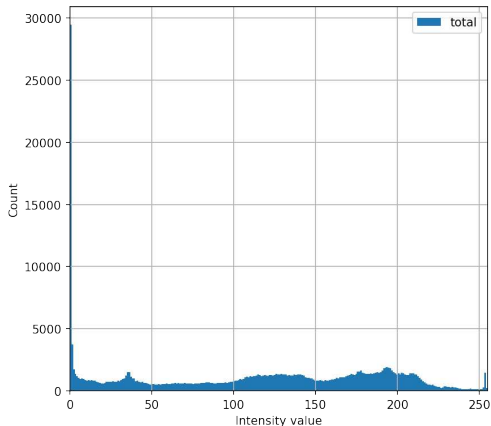


Image histogram

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

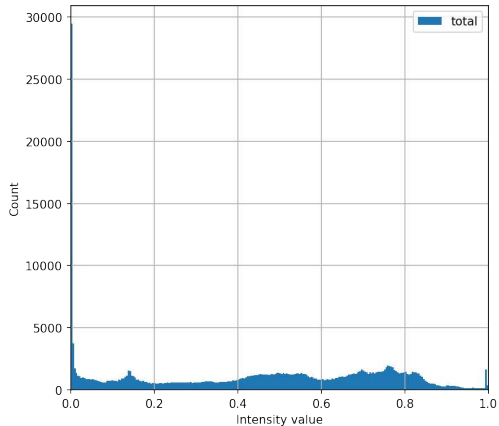


NB: weights are chosen to mimic human perception of red, green and blue: the weight on the green band is larger because the human eye has greater sensitivity to green (the retina contains more photoreceptor cells (cones) that are tuned to detect green light)

Image histogram

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

gray-scale (float)



1. Motivation

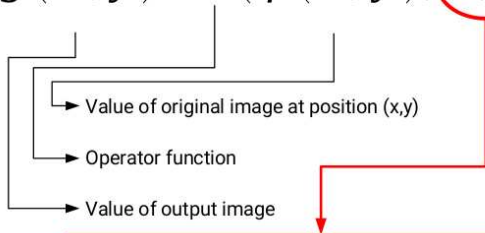
2. What is a digital image?

3. Point operations

1. homogeneous point operations
2. inhomogeneous point operations

4. Image manipulation with Python

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

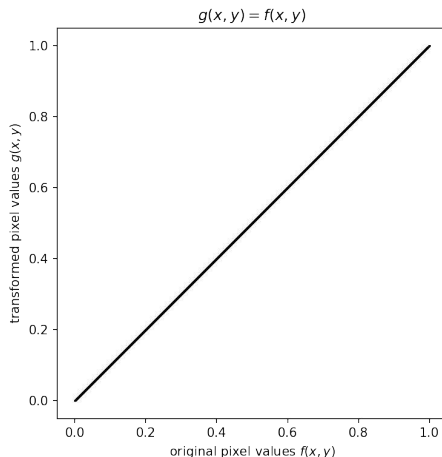


Inhomogeneous PO: T is dependent on (x, y)
Homogeneous PO: T is NOT dependent on (x, y)

Homogeneous point operations (does not depend on pixel position)

1. image intensity transformation using standard mathematical operations ($\Rightarrow$ adjust pixel color 0=black / 1=white)

identity



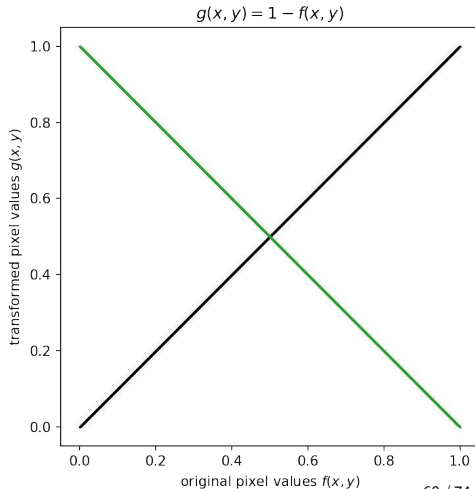
Homogeneous point operations (does not depend on pixel position)

1. image intensity transformation using standard mathematical operations ($\Rightarrow$ adjust pixel color 0=black / 1=white)

identity



inverse



Homogeneous point operations (does not depend on pixel position)

1. image intensity transformation using standard mathematical operations ($\Rightarrow$ adjust pixel color 0=black / 1=white)

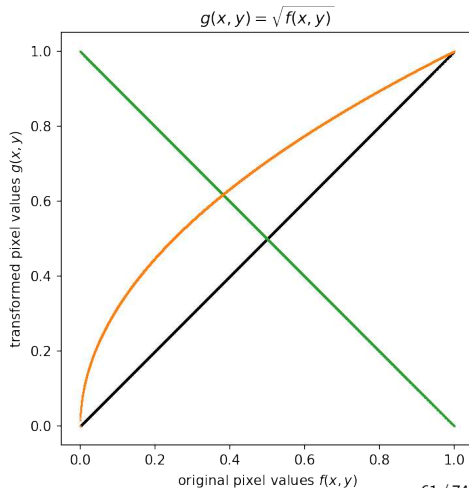
identity



inverse



square root



Homogeneous point operations (does not depend on pixel position)

1. image intensity transformation using standard mathematical operations ($\Rightarrow$ adjust pixel color 0=black / 1=white)

identity



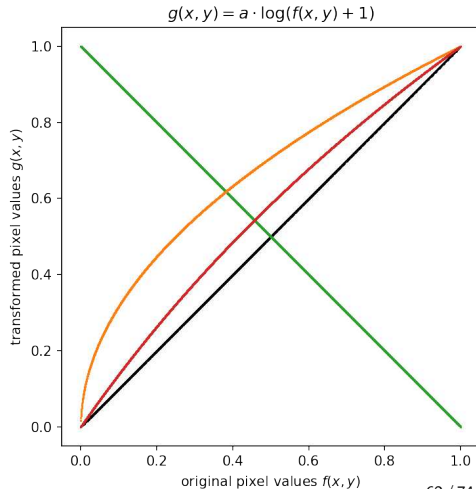
inverse



square root



logarithm



Homogeneous point operations (does not depend on pixel position)

1. image intensity transformation using standard mathematical operations ($\Rightarrow$ adjust pixel color 0=black / 1=white)

identity



inverse



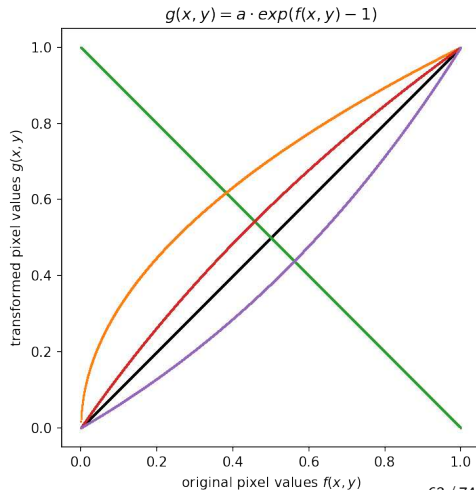
square root



logarithm



exponential



Homogeneous point operations (does not depend on pixel position)

1. image intensity transformation using standard mathematical operations ($\Rightarrow$ adjust pixel color 0=black / 1=white)

identity



inverse



square root



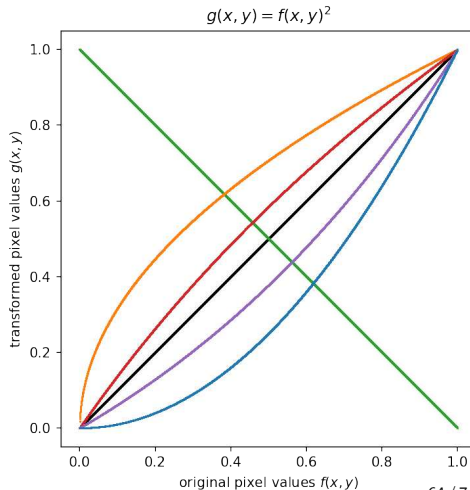
logarithm



exponential



square

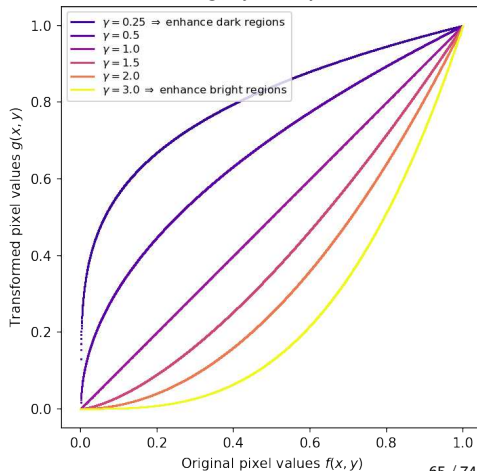


Homogeneous point operations (does not depend on pixel position)

1. image intensity transformation using **Gamma correction** ($\Rightarrow$ power-law transformation)

 $\gamma = 0.25$  $\gamma = 0.5$  $\gamma = 1.0$  $\gamma = 1.5$  $\gamma = 2.0$  $\gamma = 3.0$ 

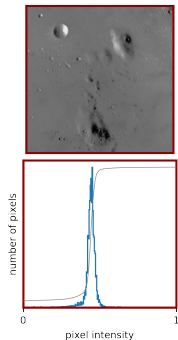
$$g(x, y) = f(x, y)^\gamma$$



Homogeneous point operations (does not depend on pixel position)

2. image contrast adjustment ($\Rightarrow$ adjust image histogram)

ORIGINAL image
low contrast



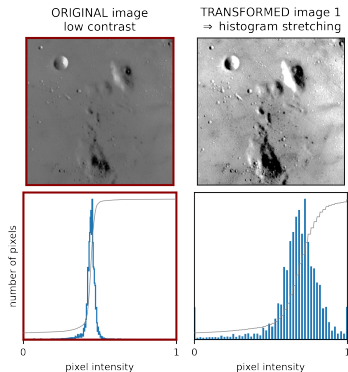
Modified after: [skimage-tutorial](#)

Original image (no stretch)

- $\Rightarrow$ image pixel intensity values are limited to a narrow range
- $\Rightarrow$ without stretch only a small portion of the full range of possible display levels is used
- $\Rightarrow$ results in a low contrast image

Homogeneous point operations (does not depend on pixel position)

2. image contrast adjustment ($\Rightarrow$ adjust image histogram)



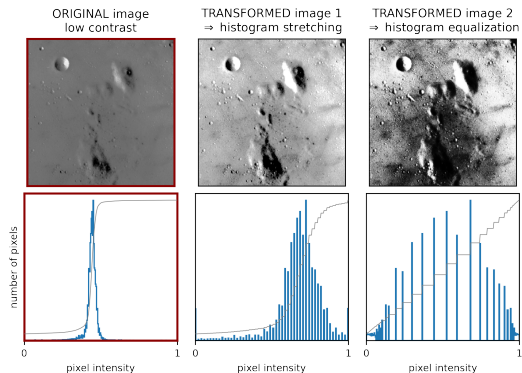
Modified after: [skimage-tutorial](#)

Transformed image #1: linear histogram stretching

- $\Rightarrow$ expand range of pixel intensities to stretch across full range of possible values
- $\Rightarrow$ rescale pixel values to a specific range
EX: rescale pixel intensities between 2nd and 98th percentiles to occupy full 0-1 range

Homogeneous point operations (does not depend on pixel position)

2. image contrast adjustment ($\Rightarrow$ adjust image histogram)



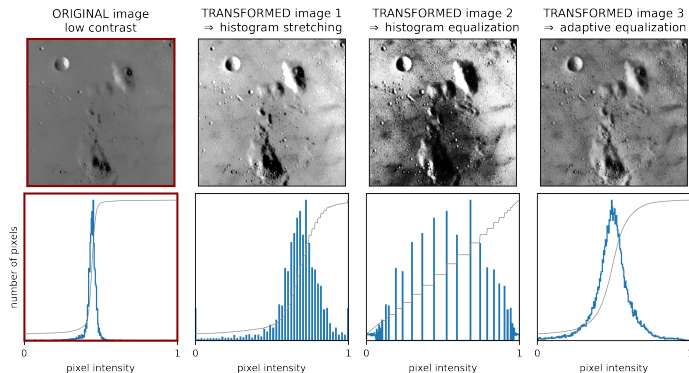
Modified after: [skimage-tutorial](#)

Transformed image #2: histogram equalization

- $\Rightarrow$ expand image pixel values on the basis of their frequency of occurrence (i.e. spreads out the most frequent intensity values)
- $\Rightarrow$ equalized image has a roughly linear cumulative distribution function

Homogeneous point operations (does not depend on pixel position)

2. image contrast adjustment ($\Rightarrow$ adjust image histogram)



Transformed image #3: adaptive histogram equalization

- $\Rightarrow$ algorithm “Contrast Limited Adaptive Histogram Equalization” (CLAHE)
- $\Rightarrow$ computes histograms over different regions of the image for local contrast enhancement
- $\Rightarrow$ local details can be enhanced even in regions that are darker or lighter than most of the image

Modified after: [skimage-tutorial](#)

Inhomogeneous point operations (depends on pixel position)

EX: background detection / change detection

Individual images (f_i)



f_1



f_i



f_N



Average image (a)



a

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

Inhomogeneous point operations (depends on pixel position)

EX: background detection / change detection

Individual images (f_i)



f_1



f_i



f_N



Average image (a)



a

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

Transformed images (g_i): object detection



g_1



g_i



g_N

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

1. Motivation
2. What is a digital image?
3. Point operations
4. Image manipulation with Python
 1. numpy tutorial
 2. exercises

Numpy tutorial:

⇒ Open CV4GS_02_image-basics/[CV4GS_02_numpy-tutorial.ipynb](#)

Exercises:

⇒ Open CV4GS_02_image-basics/[CV4GS_02_exercises.ipynb](#)