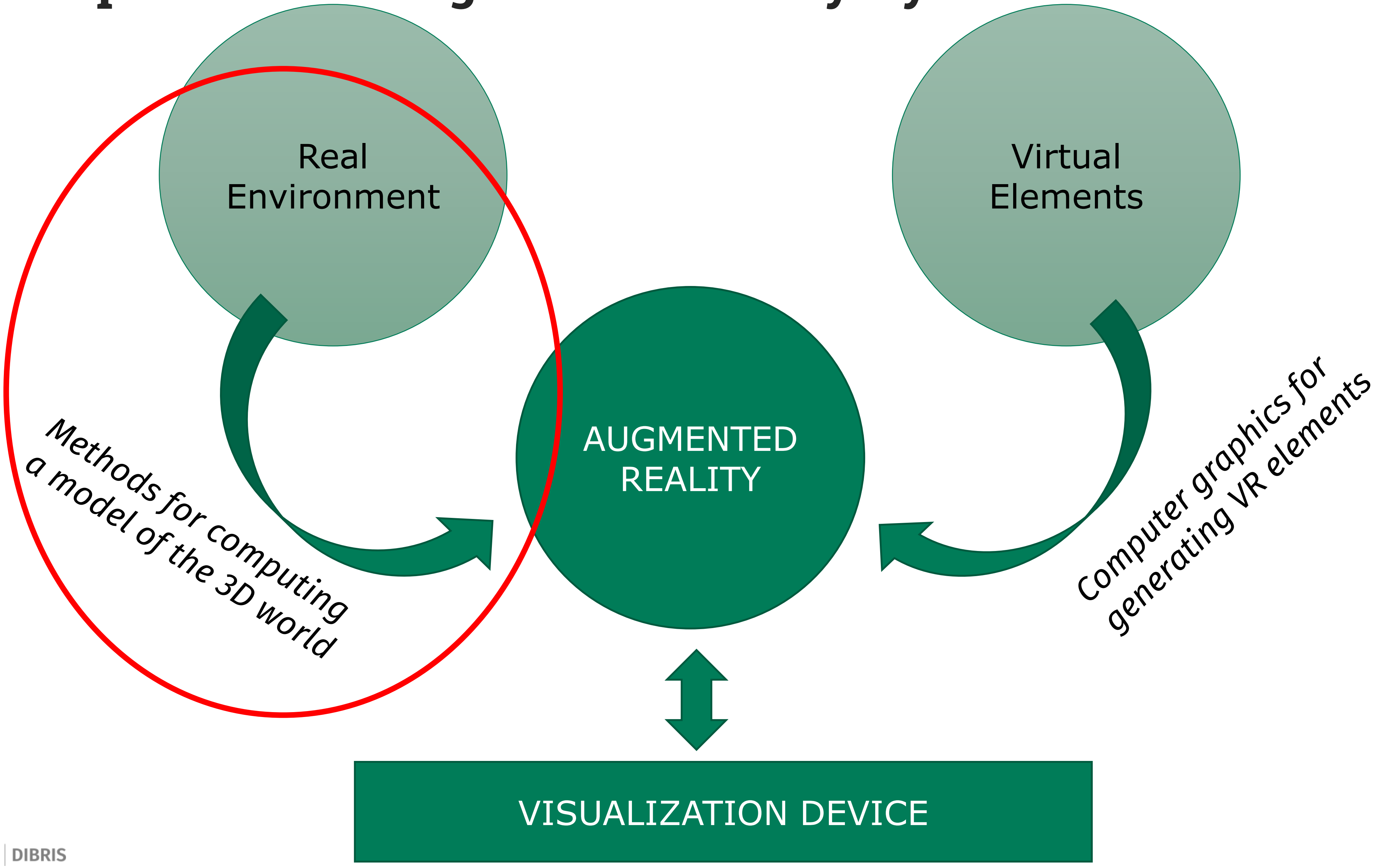
Università di Genova | DIBRIS DIPARTIMENTO DI INFORMATICA, BIOINGEGNERIA, ROBOTICA E INGEGNERIA DEI SISTEMI

Augmented Reality

Lecture 8 – image formation and edge detection

Manuela Chessa – manuela.chessa@unige.it
Fabio Solari – fabio.solari@unige.it

Description of an Augmented Reality System



CG, AR and 3D CV

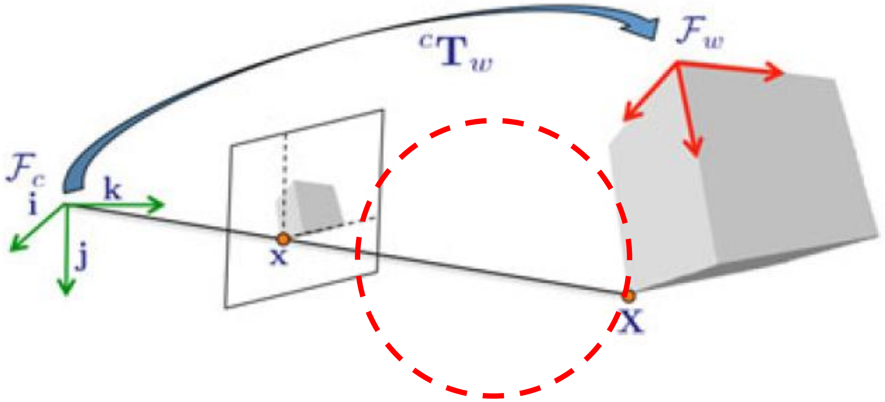
- Computer Graphics (CG): creating an image from scratch by using a 3D model.

- Computer Vision (CV): understanding the “content” of an image, usually for estimating a “3D model” of the depicted scene.

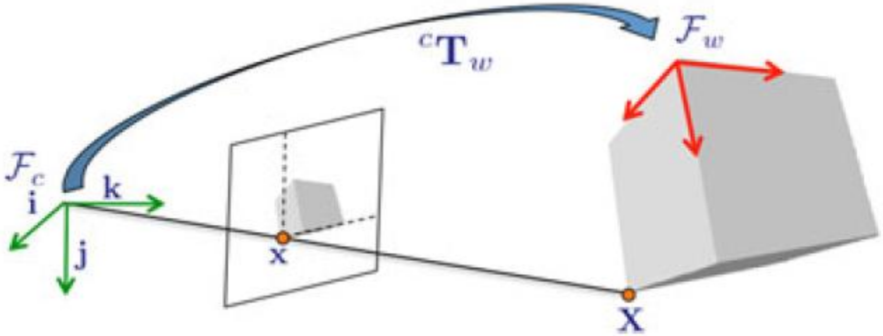
- Augmented Reality (AR): using CG and CV to blend real and virtual contents in a coherent way.

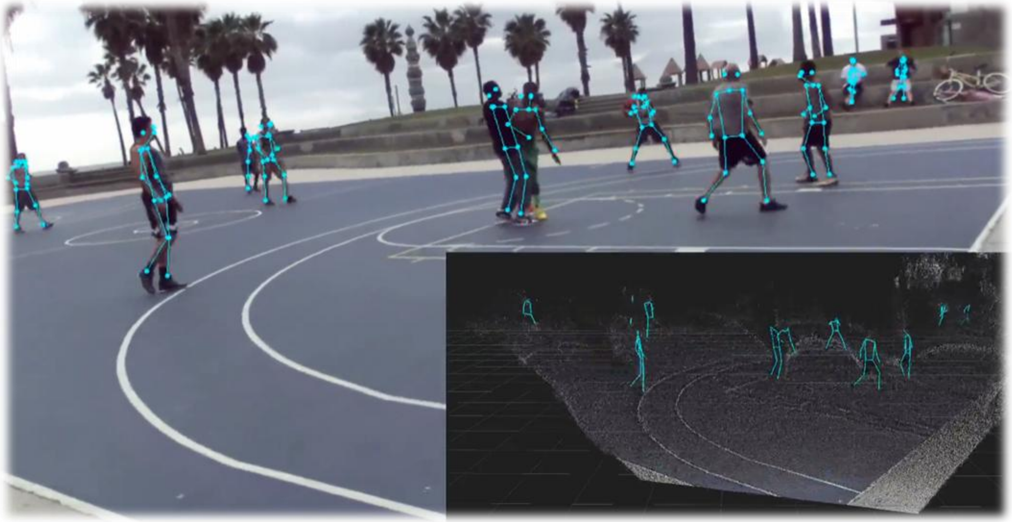
CG, AR and 3D CV

<u>Computer Graphics</u>: the unknown is the 2D image, i.e to create the 2D image(s) from 3D model and camera position



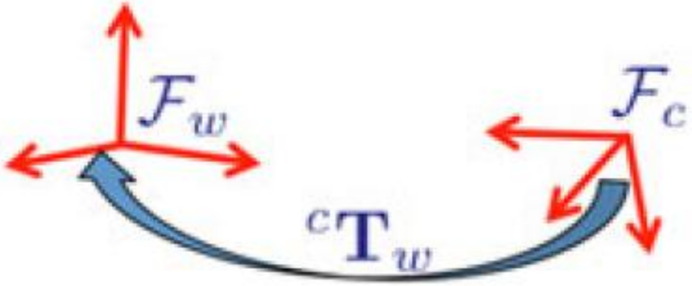
<u>Computer Vision</u>: the unknown is the real camera position in the world, i.e. to estimate the 3D camera pose from 2D image(s), then the 3D scene





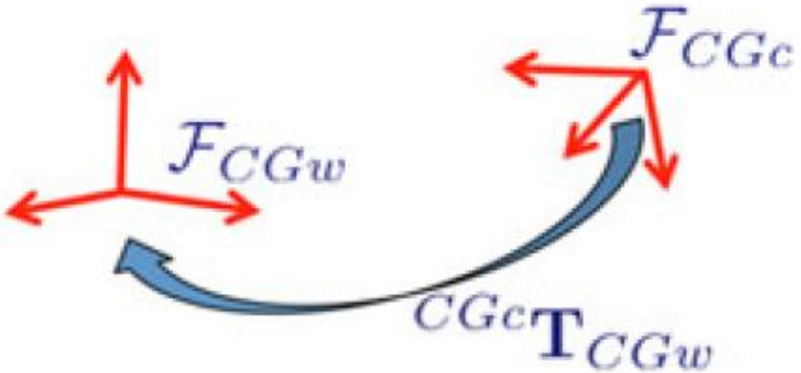
CG, AR and 3D CV

Computer Vision: we infer 3D models

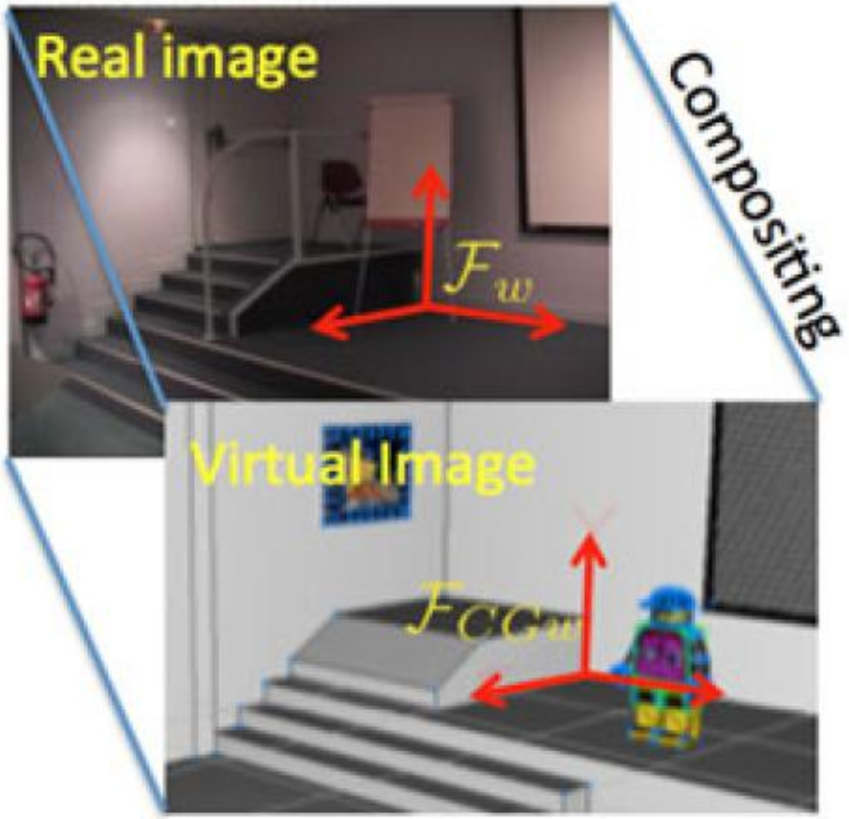


Augmented Reality

Align CG camera with real camera



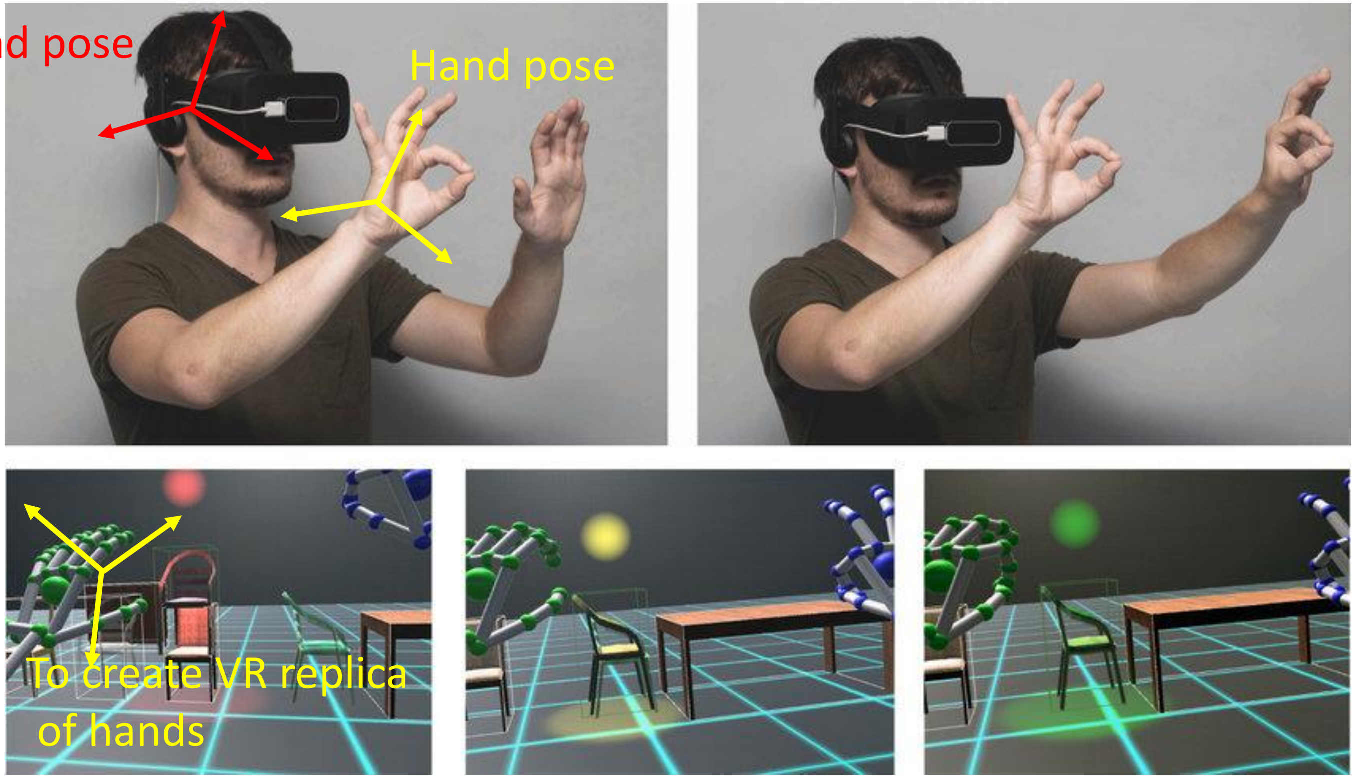
Computer Graphics: we know 3D models



w: world
c: camera
CG: computer graphics

CG, AR and 3D CV

User tracking

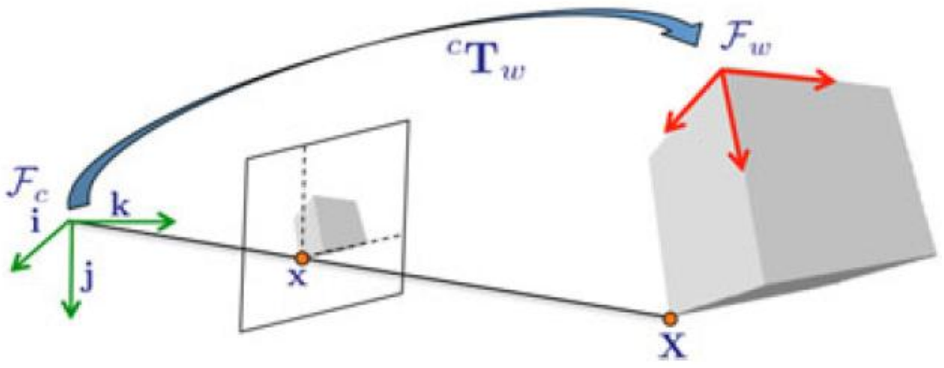


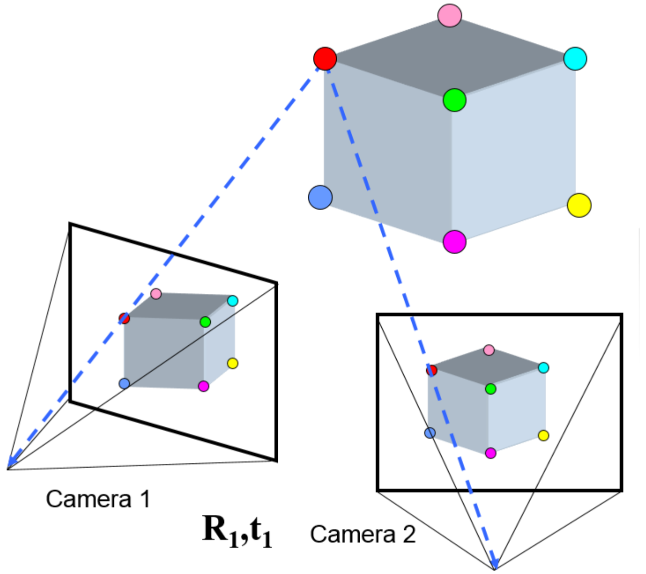
To change the view of VR world

3D Computer Vision for AR

*The **unknown** is the **real camera position** in the world (i.e., pose estimation problem).* There are several approaches:

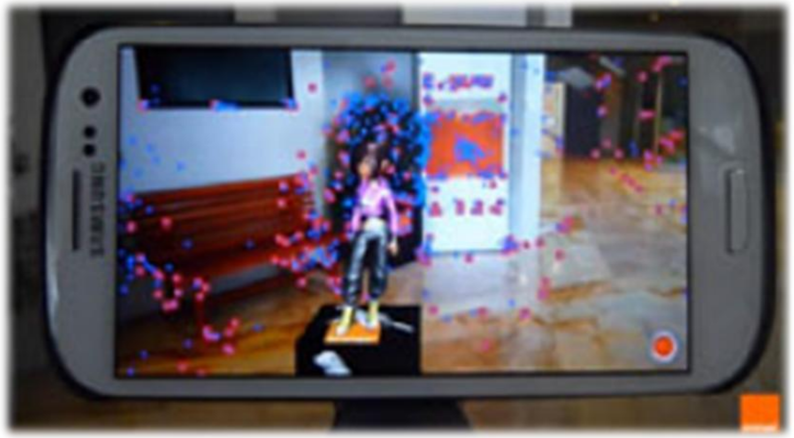
1. Approaches where **3D models are available**:
   - classical pose estimation method (this is an *inverse problem* and uses images)





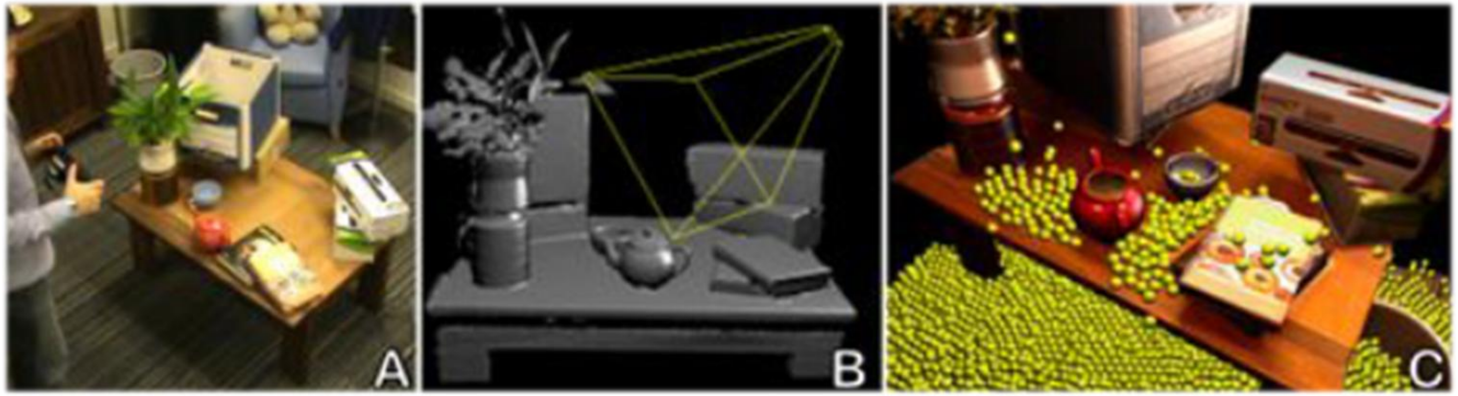
3D Computer Vision for AR

2. Approaches where **3D models are not available** (*by using images*):
   – 3D models can be estimated on-line (by using *RGB cameras*) thanks to Simultaneous Localization and Mapping (SLAM) techniques



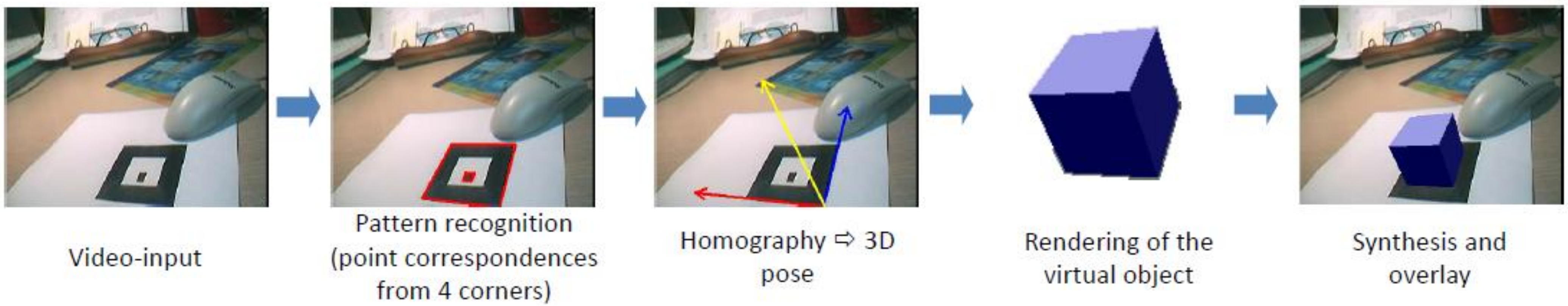
3D Computer Vision for AR

3. Approaches where **3D models are not available** (*by using point cloud*):
   - when 3D data can be directly measured (by *RGB-D cameras*), registration can be done directly in the 3D space, e.g. by Iterative Closest Point (ICP) technique



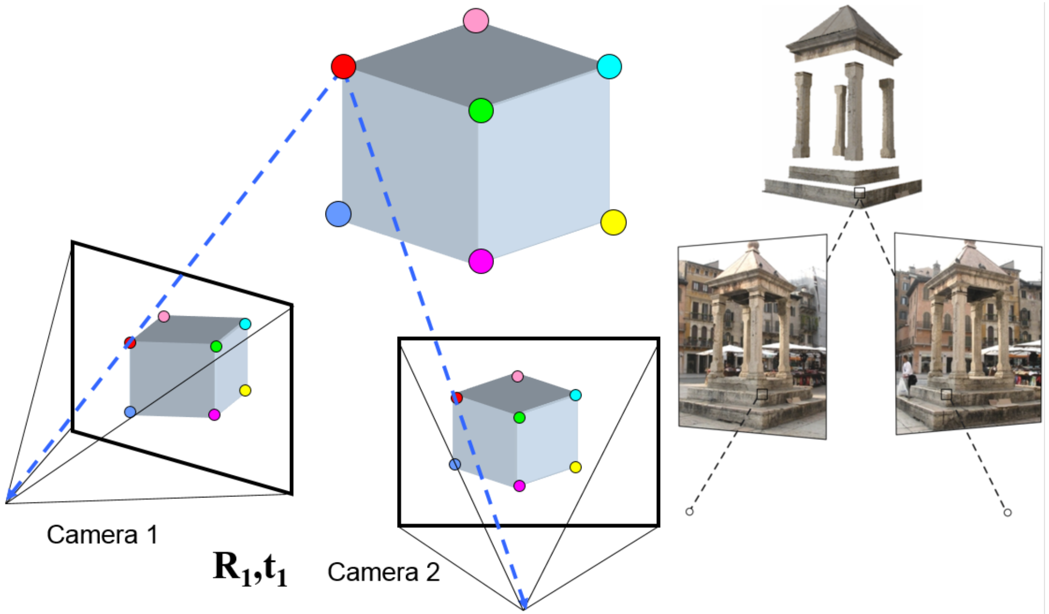
3D Computer Vision for AR

4. Approaches where **2D models are available** (*markers*)**:** the pose estimation problem can be simplified when the **scene is planar** (*homography*)



3D Computer Vision for AR

- From a practical point of view, the development of **actual AR** applications rises the question of the **features extraction and matching** and of the **3D reconstruction**: we consider calibration, registration and tracking.



Summary

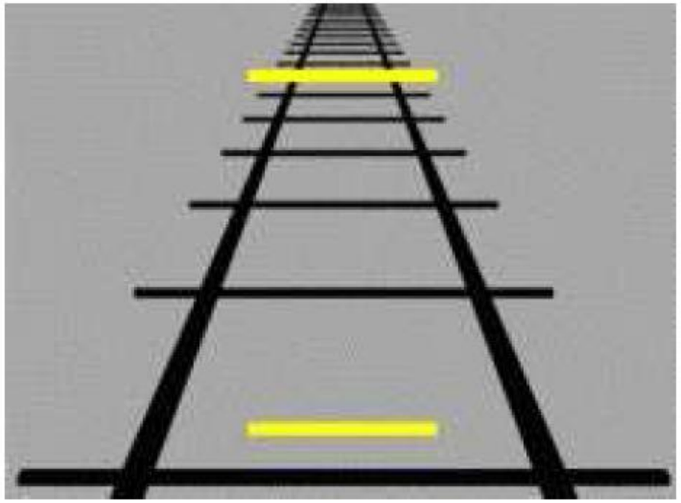
- Image processing:
  - image formation and edge detection
- Computer Vision:
  - image segmentation, keypoints (corners), stereopsis, disparity computation
- 3D Computer Vision for AR:
  - camera calibration, pose estimation, epipolar geometry, RGBD camera
  - camera tracking, homography, SLAM, SPAAM

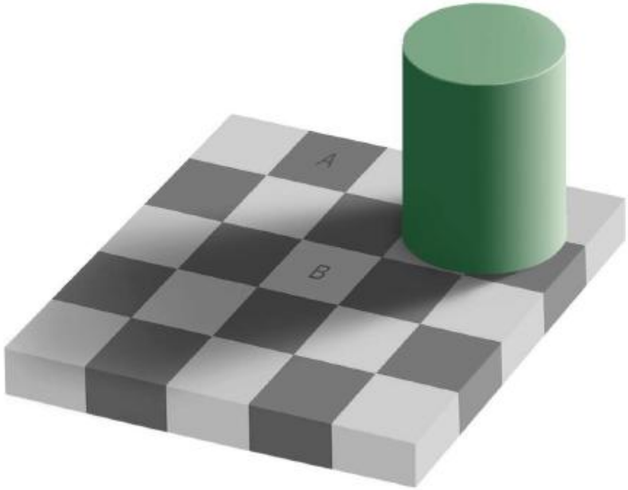
Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

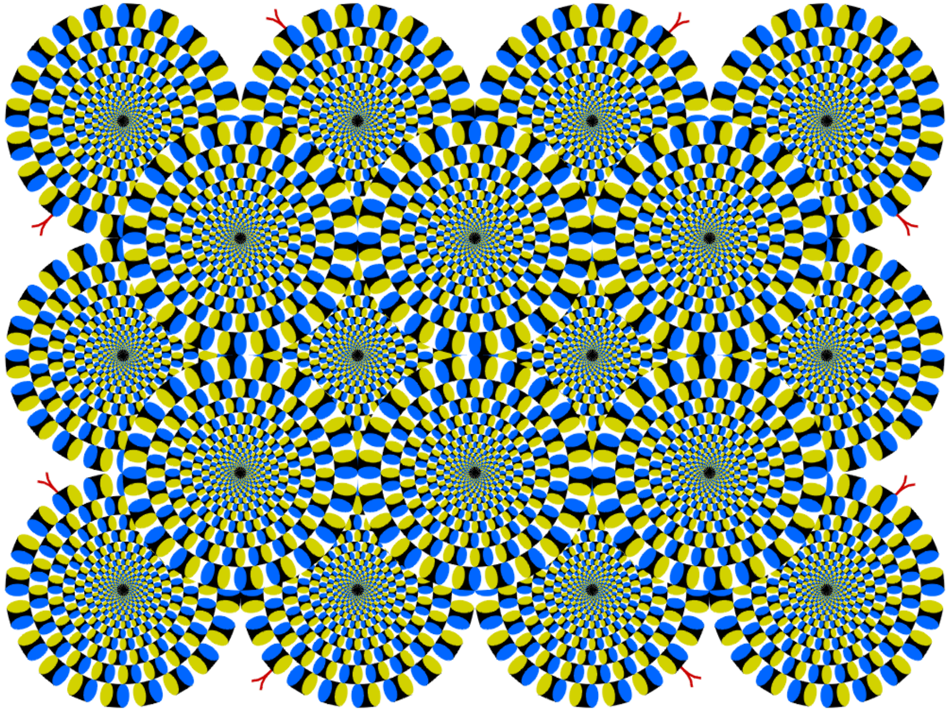
Image formation and edge detection

Introduction

- Vision is deceivingly easy:
  - vision is immediate for us
  - we perceive the visual world as external to ourselves, but it is **a reconstruction within our brain**







- Vision is computationally demanding

Introduction

What do you see?

perceived as a 3D scene

but really just a planar surface (screen) under nonuniform lighting (projector).

- It is something mathematically impossible: **recovering 3D from a single image is a mathematically inverse ill-posed problem**.

- How?
- You used **assumptions** based on prior knowledge/experience about the way the world works

Introduction

The goal of computer vision: **to bridge the gap** between “meaning” and pixels

What we see

0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

What a computer sees

Introduction

- To interact with the real-world, we must tackle the **problem of inferring 3-D information of a scene from a set of 2-D images**.
- In general, this problem falls into the category of so-called *inverse problems*, which are prone to be *ill-conditioned* and difficult to solve in their full generality unless *additional assumptions* are imposed.

- Before we address how to reconstruct 3-D geometry from 2-D images, ***we first need to understand how 2-D images are generated and processed***.

Computer Vision

images

edge detection

image segmentation

object recognition

optic flow

disparity

objects

specific objects

object speed

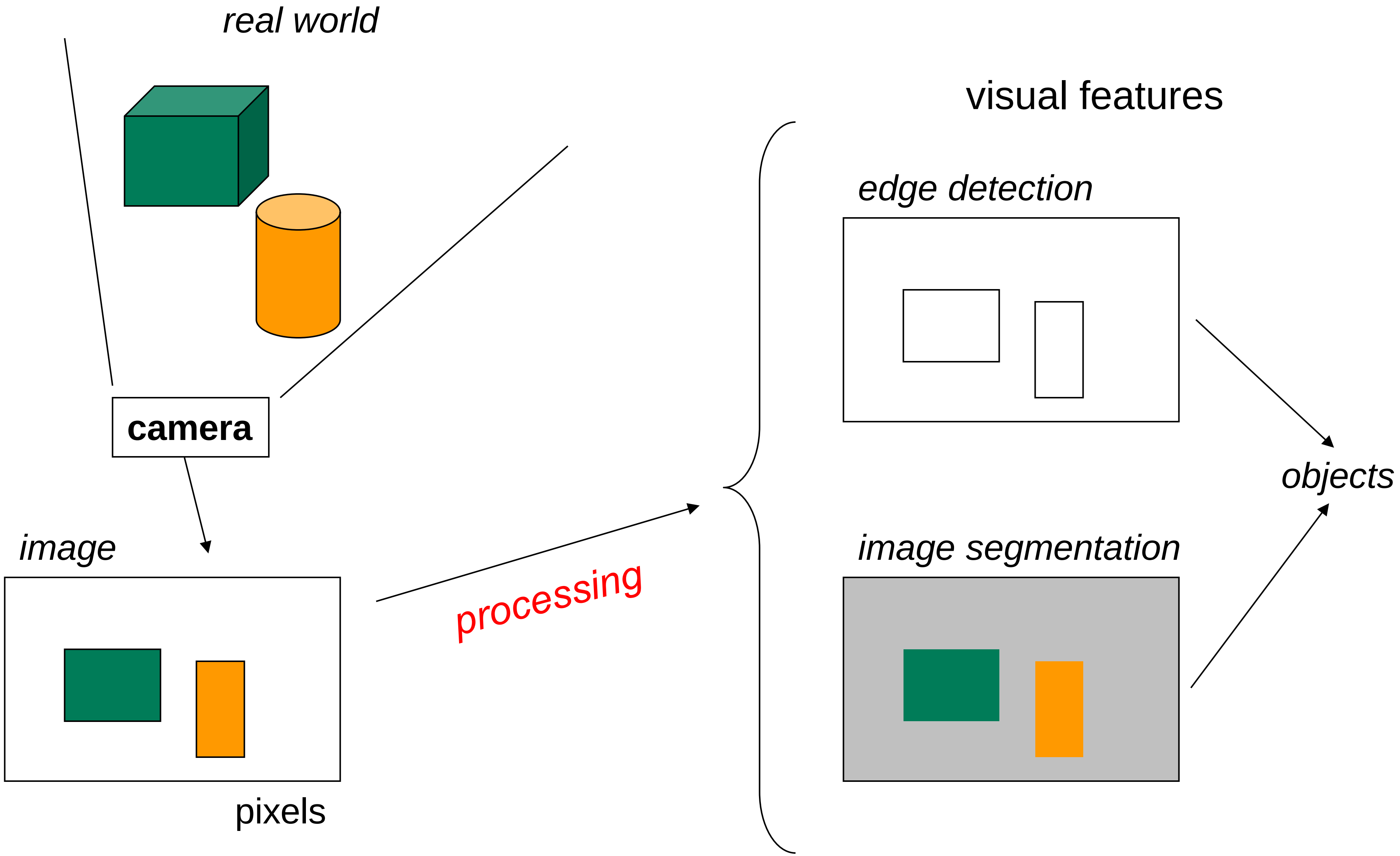
object distance

Matrix of pixels

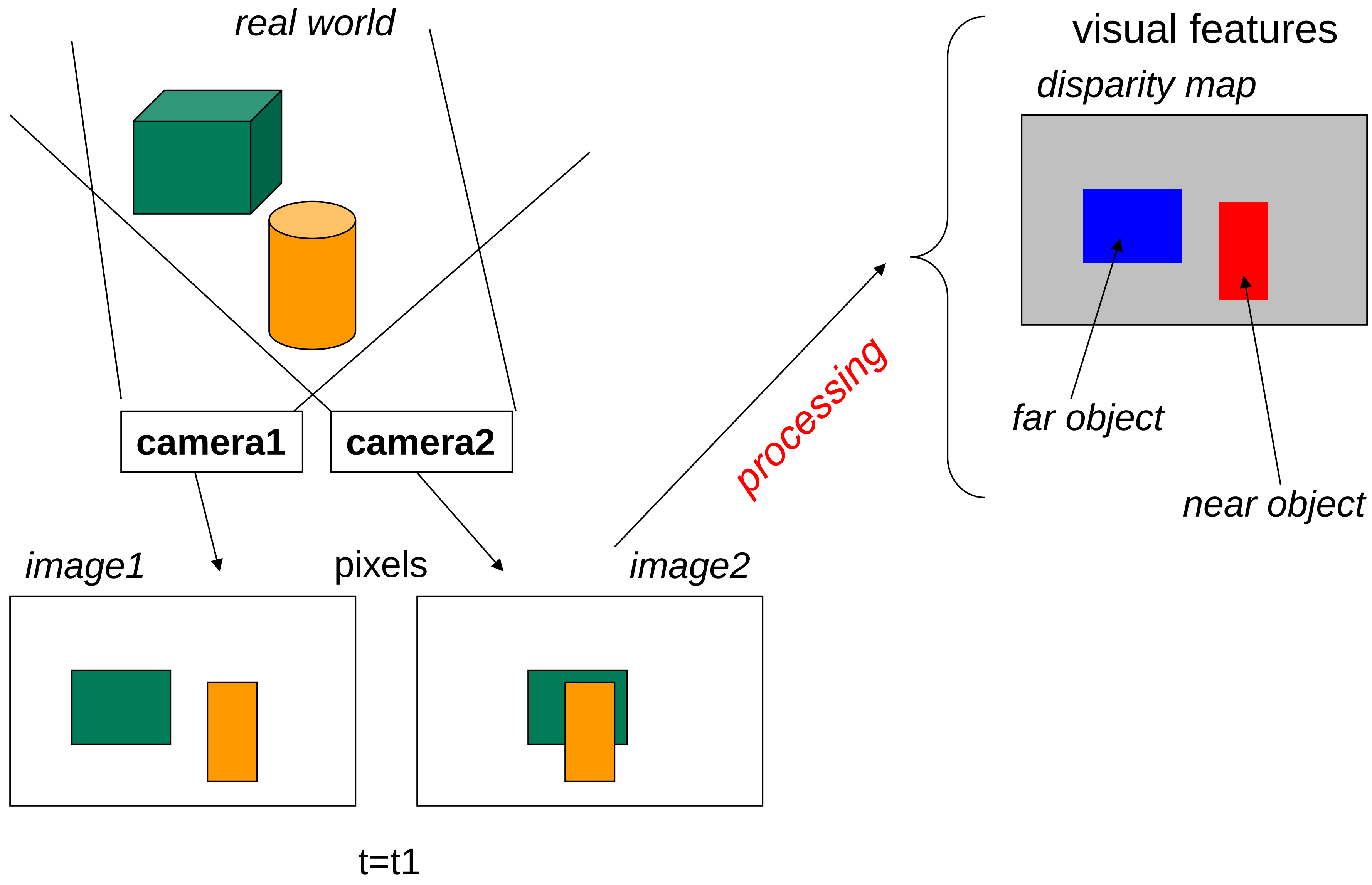
visual features

real-world properties

Computer Vision



Computer Vision



Computer Vision

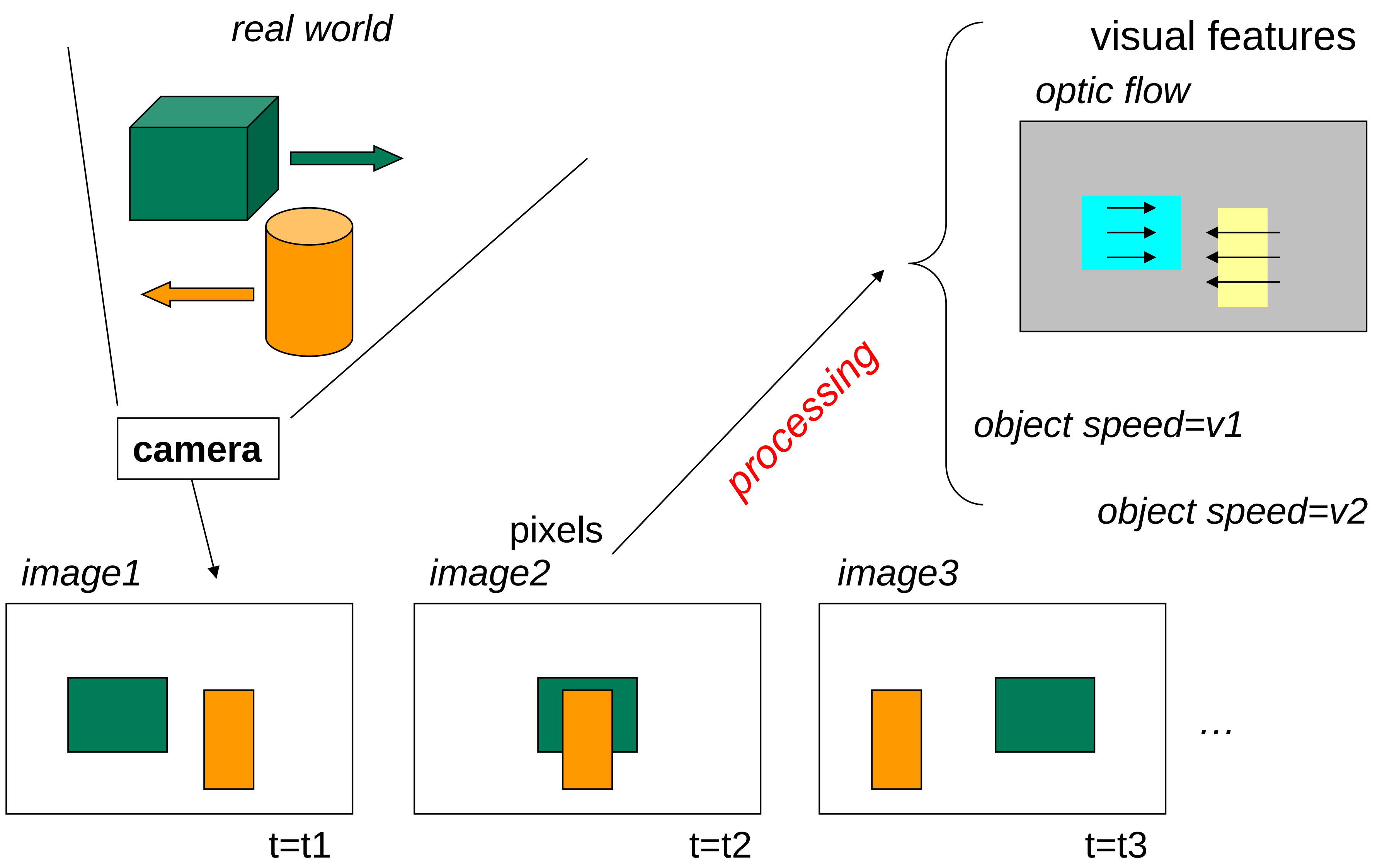


Image formation

Photometry overview

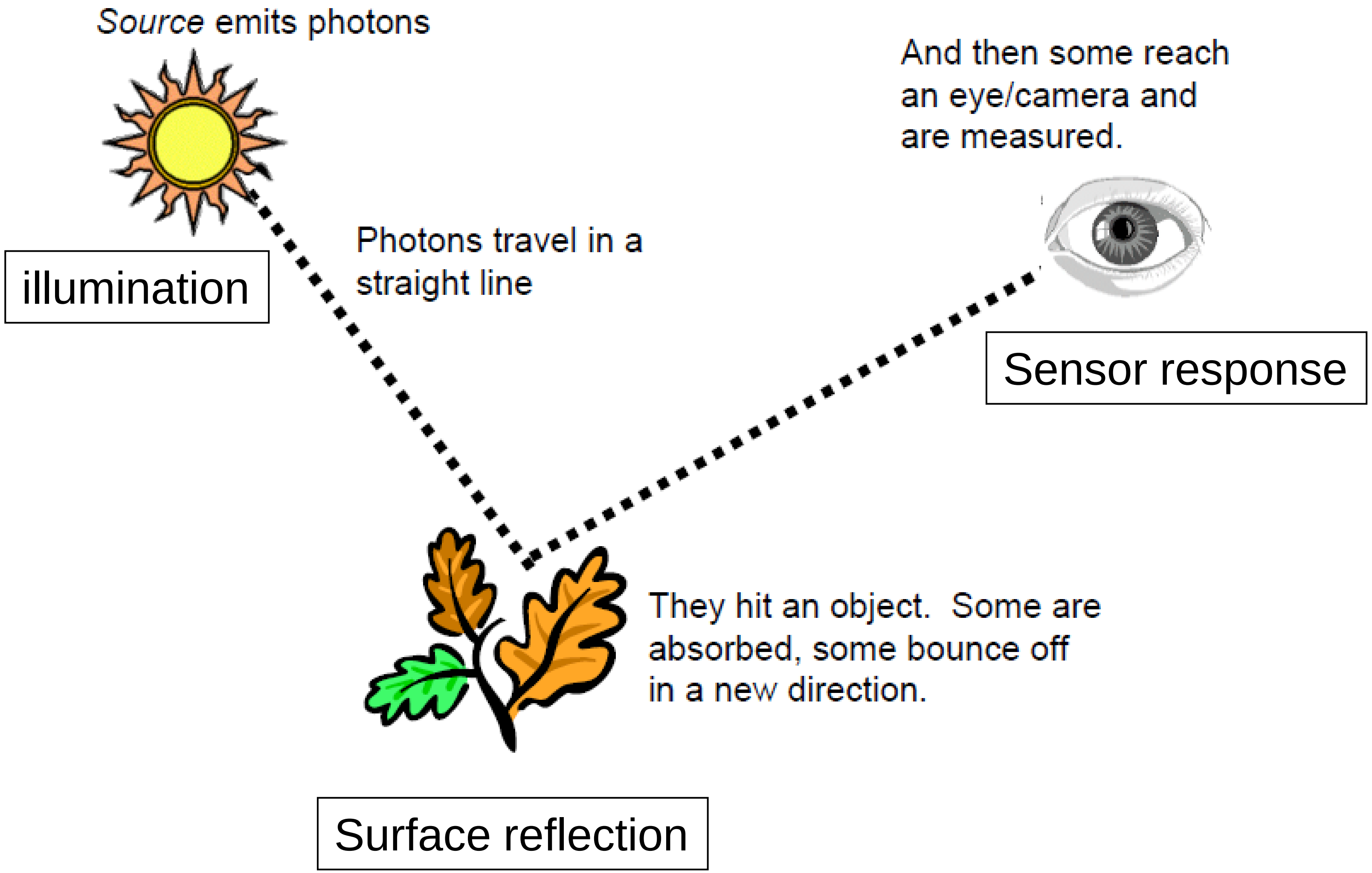


Image formation

Photometry overview

Visual signal s(x,y):

$s: R^2 \rightarrow R$

- s(x,y) is a **two-dimensional function**: x and y are spatial coordinates.
- The **amplitude** of s is called light **intensity** or gray level at the point (x, y).

s(x,y)=il(x,y) r(x,y)

s(x,y): **intensity** at the point (x,y)

il(x,y): **illumination** at the point (x,y) (*the amount of source illumination incident on the objects*)

r(x,y): **reflectance** at the point (x,y) (*the amount of illumination reflected/transmitted by the object*)

Where 0< il(x,y,)<inf and 0<r(x,y)<1

Image formation

Photometry overview

- Illumination

  Lumen: a unit of light flow or luminous flux

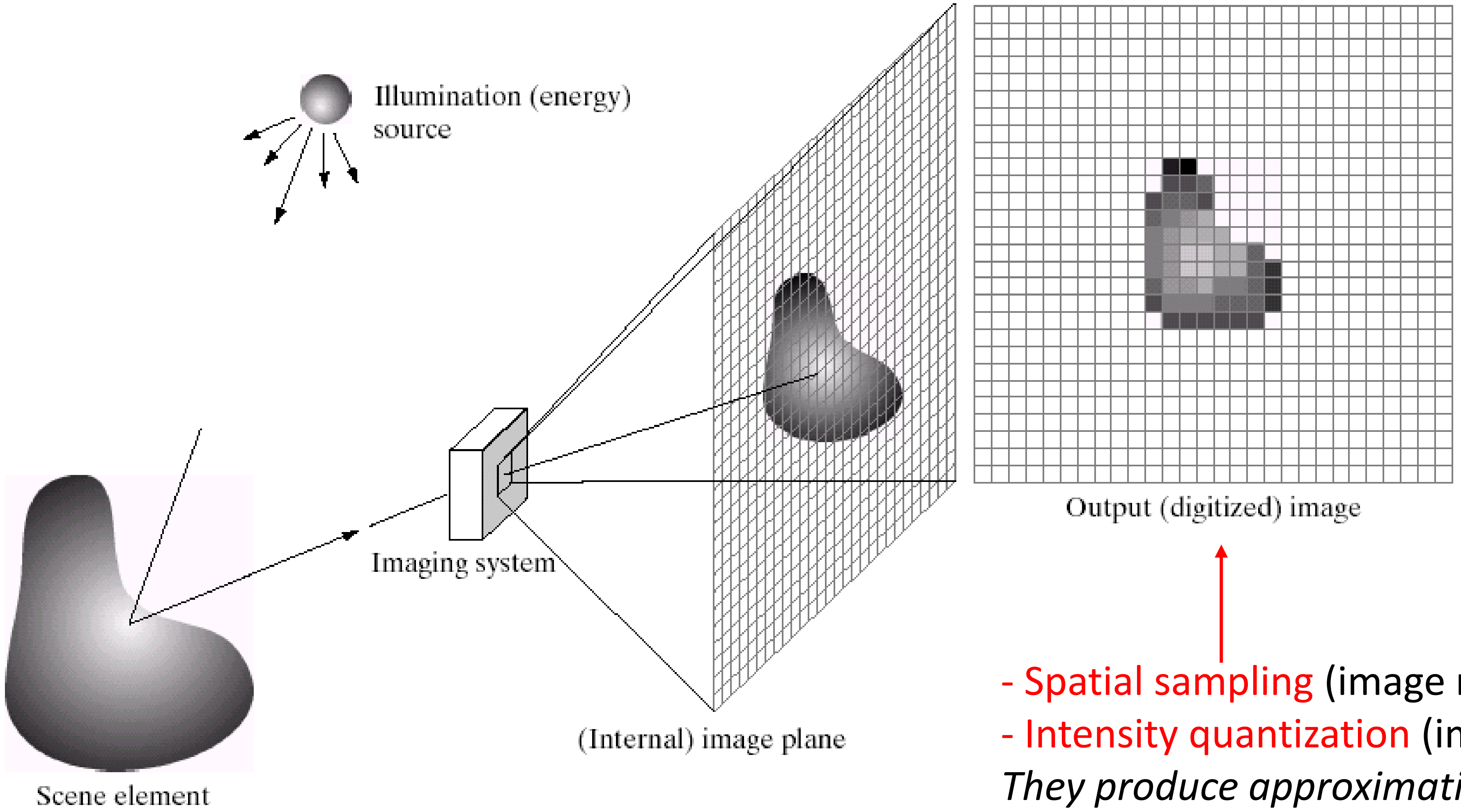
  Lumen per square meter (lm/m^2) — The metric unit of measure for illuminance of a surface

  - On a clear day, the sun may produce in excess of 90,000 lm/m^2 of illumination on the surface of the Earth
  - On a cloudy day, the sun may produce less than 10,000 lm/m^2 of illumination on the surface of the Earth
  - On a clear evening, the moon yields about 0.1 lm/m^2 of illumination
  - The typical illumination level in a commercial office is about 1000 lm/m^2

- Reflectance

  - 0.01 for black velvet
  - 0.65 for stainless steel
  - 0.80 for flat-white wall paint
  - 0.90 for silver-plated metal
  - 0.93 for snow

Image formation

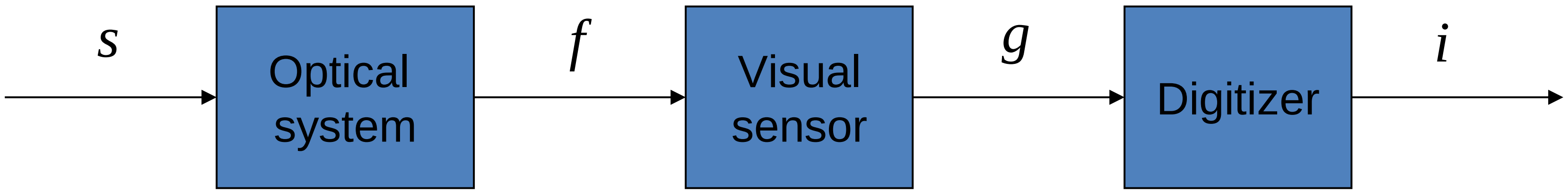


- Spatial sampling (image resolution)
- Intensity quantization (image gray levels)

They produce approximation (<u>errors</u>)

Image formation

- Digital image formation is the first step in any digital image processing application.
- The *digital image formation system* (camera) consists basically of the ***optical system***, the ***sensor*** and the ***digitizer***.



The effect of the recording process is the addition of a noise contribution.
The recorded image *i* is called noisy image.

Optical system

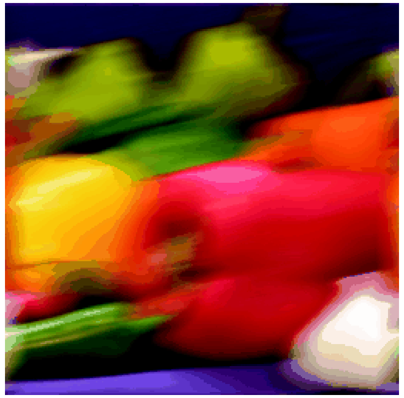
- The *optical system* can be modeled as a ***linear shift invariant system*** having a two-dimensional impulse response $h(x,y)$.
- The **input-output relation** of the optical system is described by a **2D convolution** (both signals *s* and *f* represent optical intensities):

$$f(x,y) = \iint s(\xi,\eta)h(x-\xi, y-\eta)d\xi d\eta$$

Optical system

- The two-dimensional impulse response $h(x,y)$ is also called PSF (***point spread function***), and its Fourier transform is called ***transfer function***.
  - Linear motion blur: it is due to the relative motion, during exposure, between the camera and the object being photographed, $H(\mathbf{w}) \sim exp(jTv\mathbf{w})sinc(Tv\mathbf{w})$.
  - Out-of-focus blur: $H(\mathbf{w}) \sim (1/D|\mathbf{w}|)^{3/2} \cos(D|\mathbf{w}|)$.
  - Atmospheric turbulence blur: $H(\mathbf{w}) \sim exp(-s^2 \, |\mathbf{w}|^2)$.
  - No blur: $h(x, y) = \delta(x, y)$.

Motion blur →



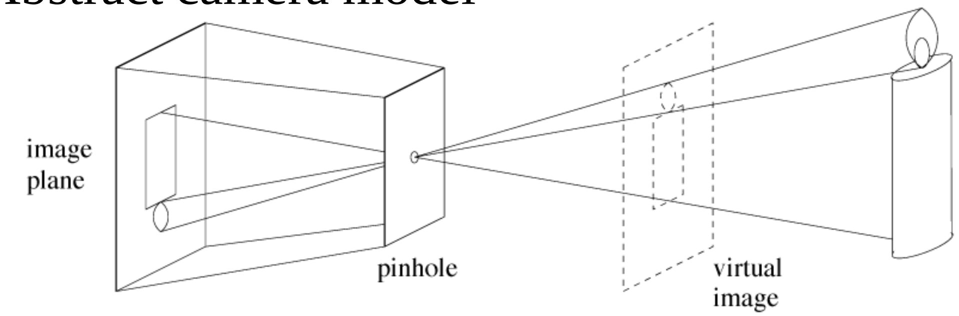
Camera model

- **Images** are **two-dimensional patterns of brightness values**. They are formed by the **projection** of **3D objects** on the camera **image plane**.

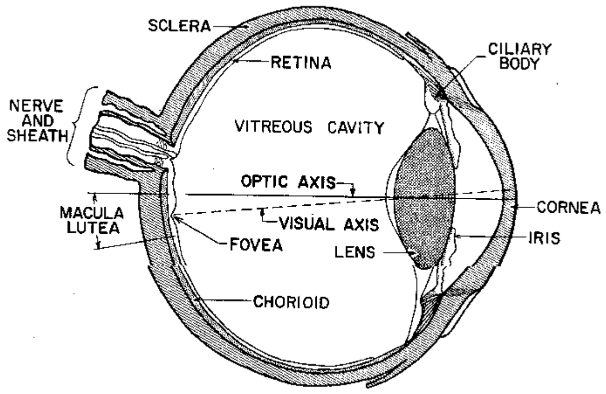
- Basic abstraction is the **pinhole camera**:
  - Lenses required to ensure image is not too dark.
  - Pinhole camera model works in practice.

Pinhole camera

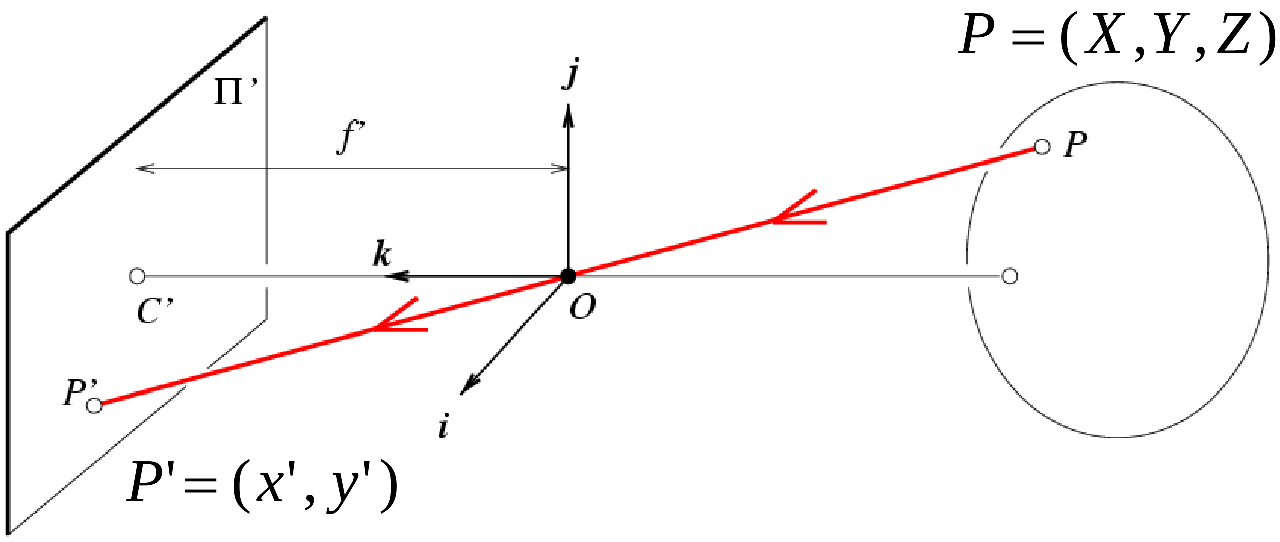
Abstract camera model



Animal eye

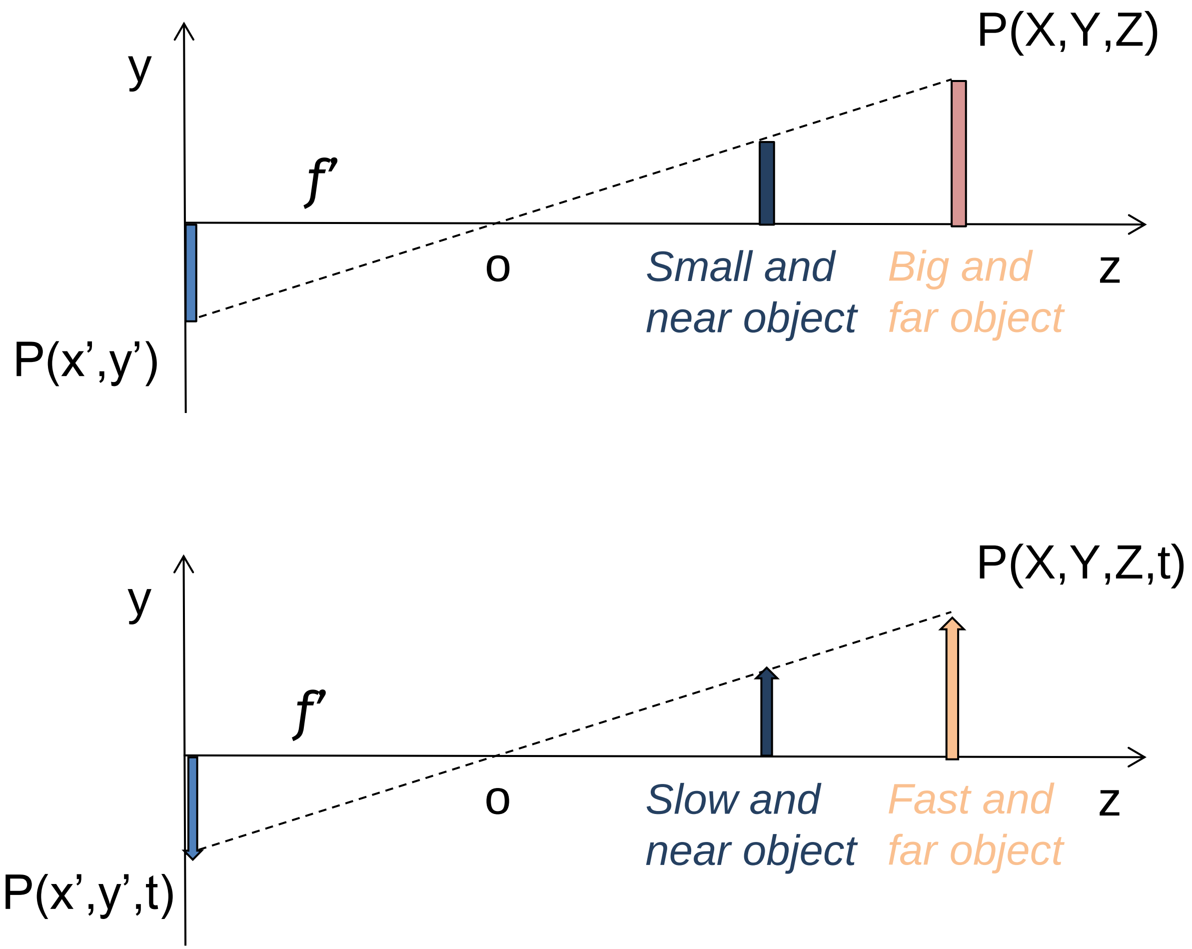


Pinhole Perspective Equation



$$\begin{cases} x' = f' \dfrac{X}{Z} \\ y' = f' \dfrac{Y}{Z} \end{cases}$$

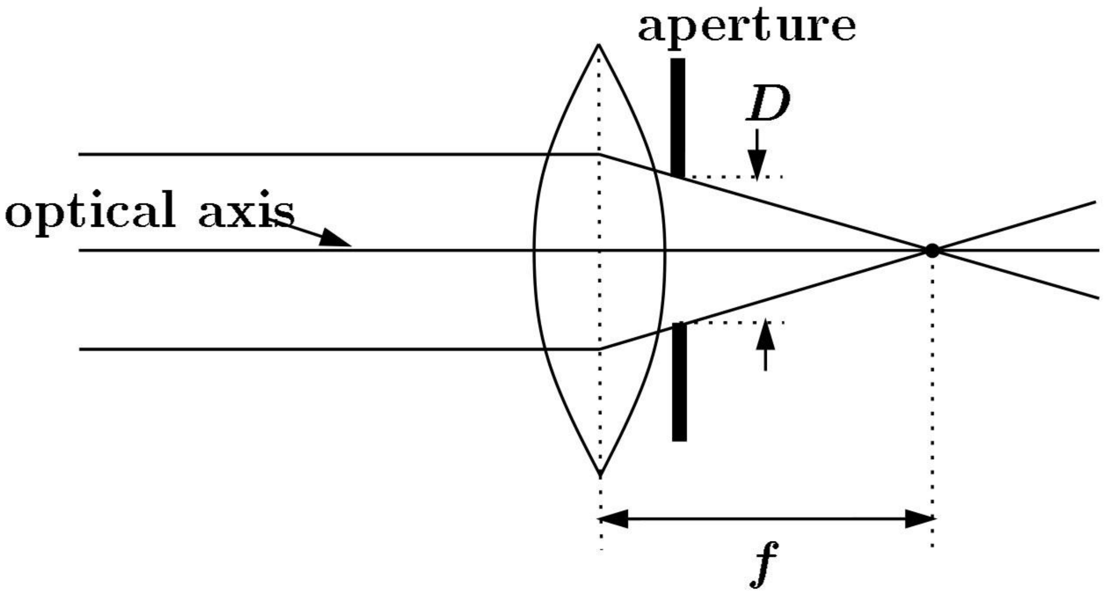
Pinhole camera



***Inverse problems*, which are prone to be ill-conditioned**

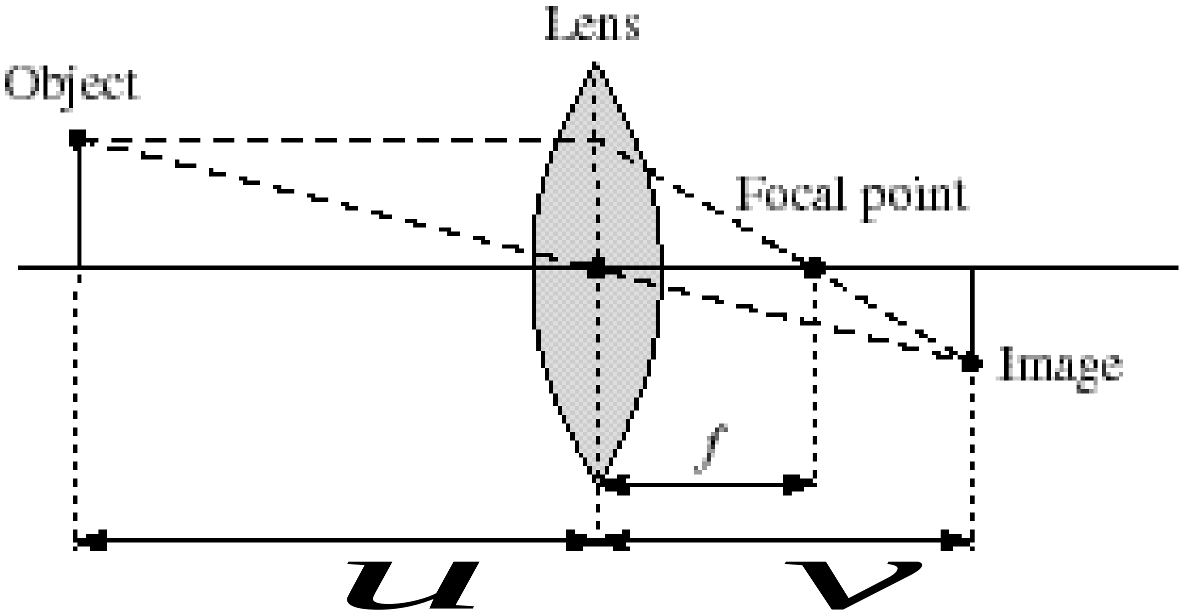
$$\begin{cases} x' = f' \dfrac{X}{Z} \\ y' = f' \dfrac{Y}{Z} \end{cases}$$

Cameras with lenses



- A lens focuses parallel rays onto a single focal point
- Gather more light, while keeping focus; **make pinhole perspective projection practical**

Cameras with lenses



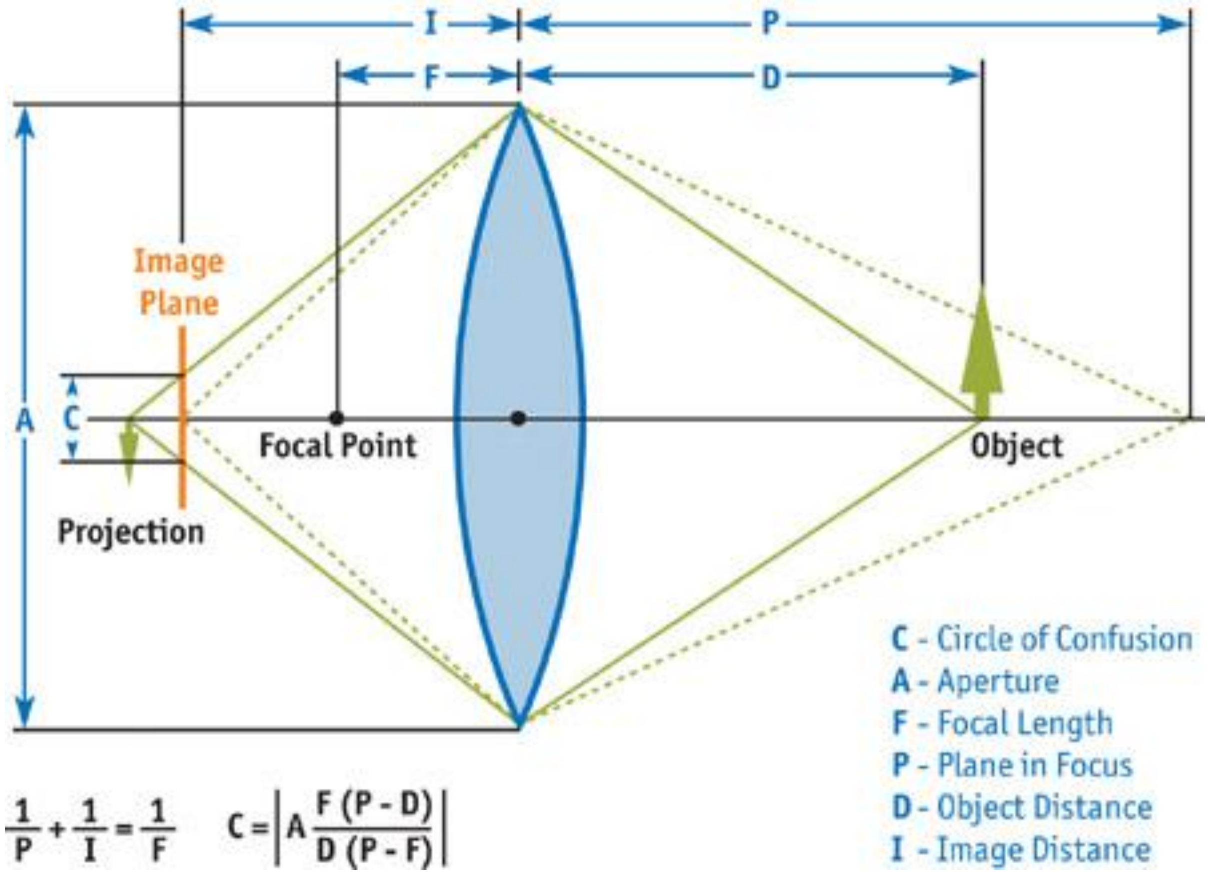
Thin lens equation

$$\frac{1}{f} = \frac{1}{u} + \frac{1}{v}$$

- Any **object** point satisfying this equation is **in focus**

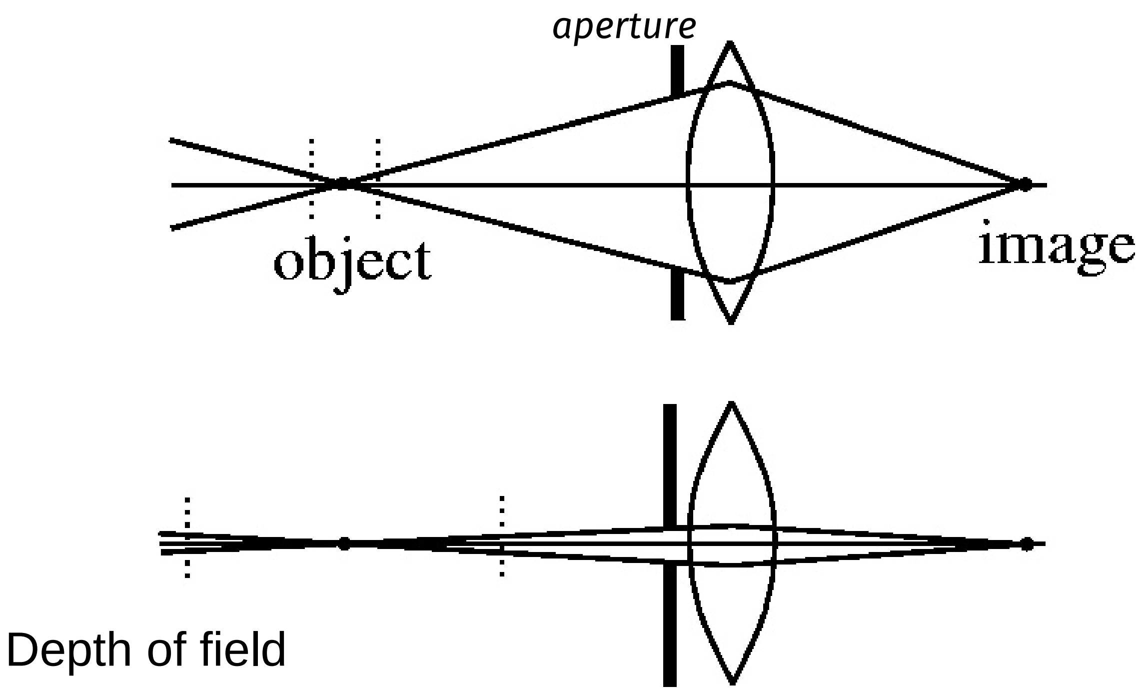
Cameras with lenses

- <u>Depth of field</u>: distance between image planes *where blur is tolerable*



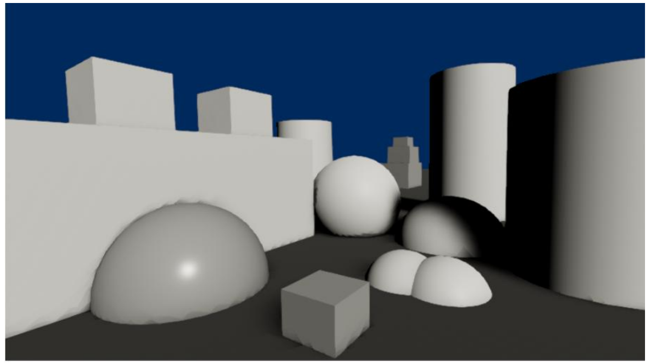
Cameras with lenses

- Depth of field: a *smaller aperture* increases the range in which the object is approximately in focus

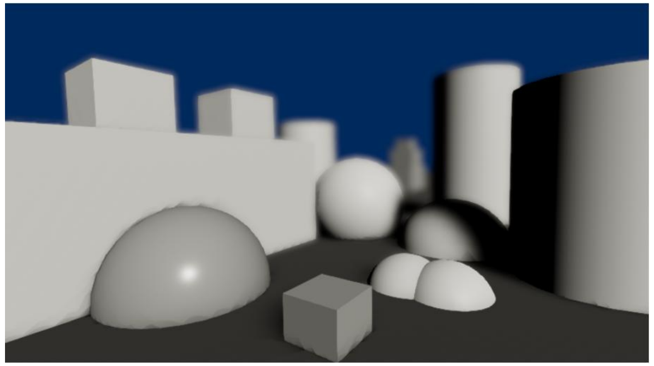


Cameras with lenses

- In VR/AR, the depth of field is a post-processing effect *to be implemented*



Without depth of field



With depth of field (*Unity*)

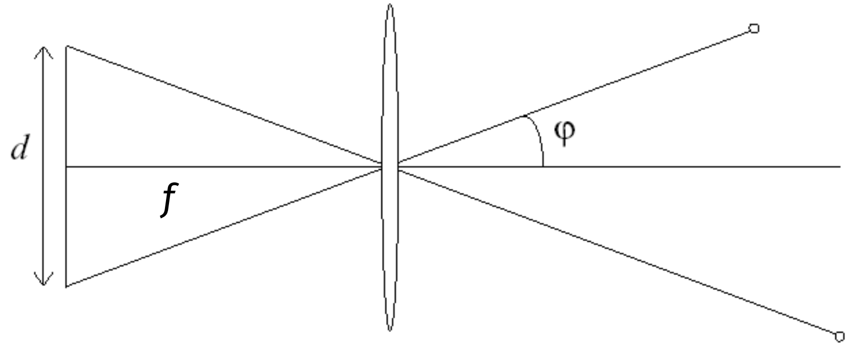
- In SAR, the depth of field is an effect *to be compensated*

Huang, B., & Ling, H. (2021). Deprocams: Simultaneous relighting, compensation and shape reconstruction for projector-camera systems. *IEEE Transactions on Visualization and Computer Graphics*, *27*(5), 2725-2735.

Cameras with lenses

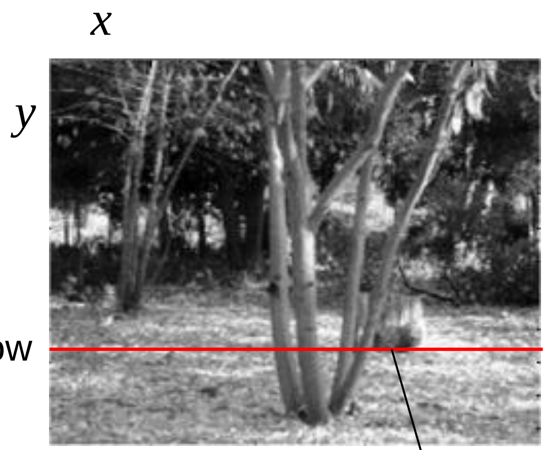
- <u>Field of view</u> (FOV): Angular measure of portion of 3D space seen by the camera

- As **f gets smaller**, image becomes more *wide angle*
  - more world points project onto the finite image plane *d*
- As **f gets larger**, image becomes *more telescopic*
  - smaller part of the world projects onto the finite image *d* plane



Digital images representation

- In Computer Vision we operate on **digital** (**discrete**) images:
  - **To sample** the 2D space on a regular grid (spatial **resolution**)
  - **To quantize** the amplitude of each sample, e.g. round to nearest integer, (gray **levels**)
- Image is represented as a *matrix of integer values* (***intensity***).

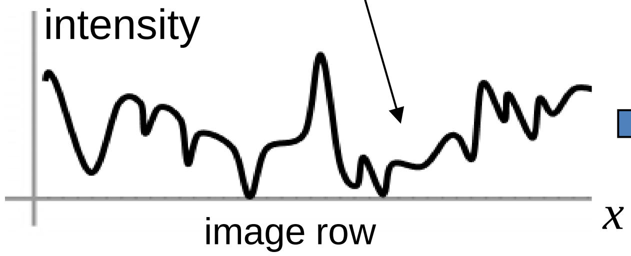


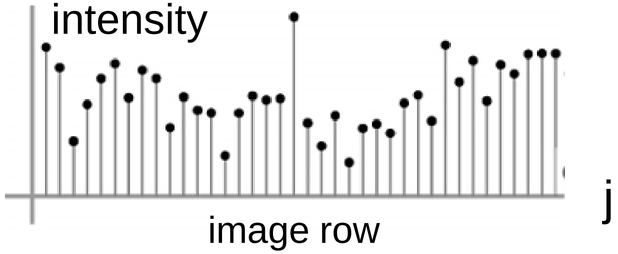
j

i

62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

2D





1D

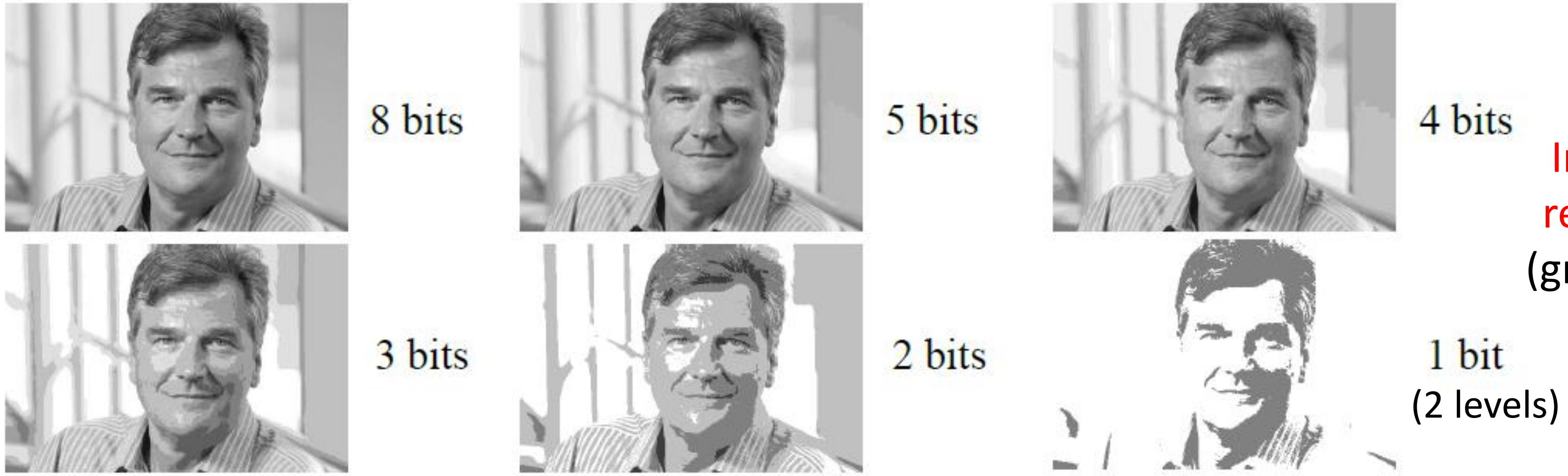
Digital images representation

Image sampling

200x200 100x100 50x50 25x25 (pixels)

Spatial resolution
(pixels)

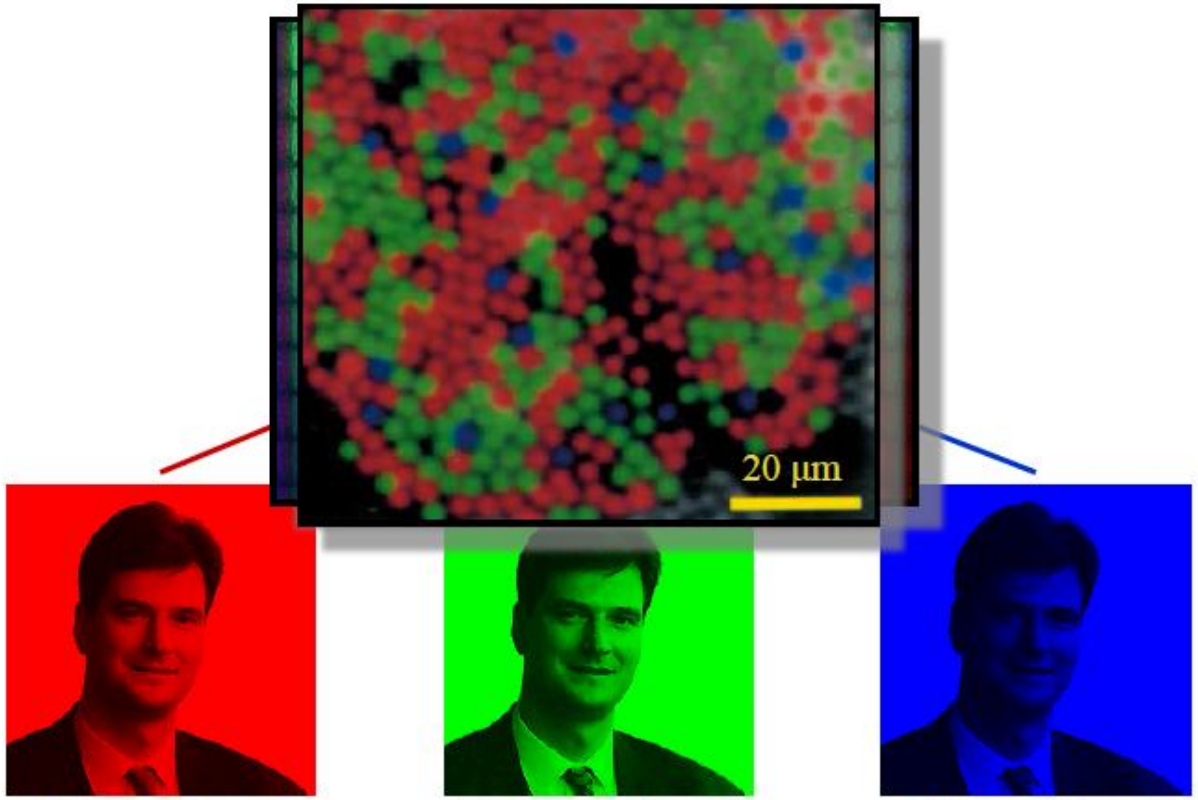
Image quantization



Intensity resolution
(gray levels)

Digital images representation

Human retina: **three different receptors**



Red $R[x,y]$ Green $G[x,y]$ Blue $B[x,y]$

We perceive

Each pixel is specified by three values (RGB)

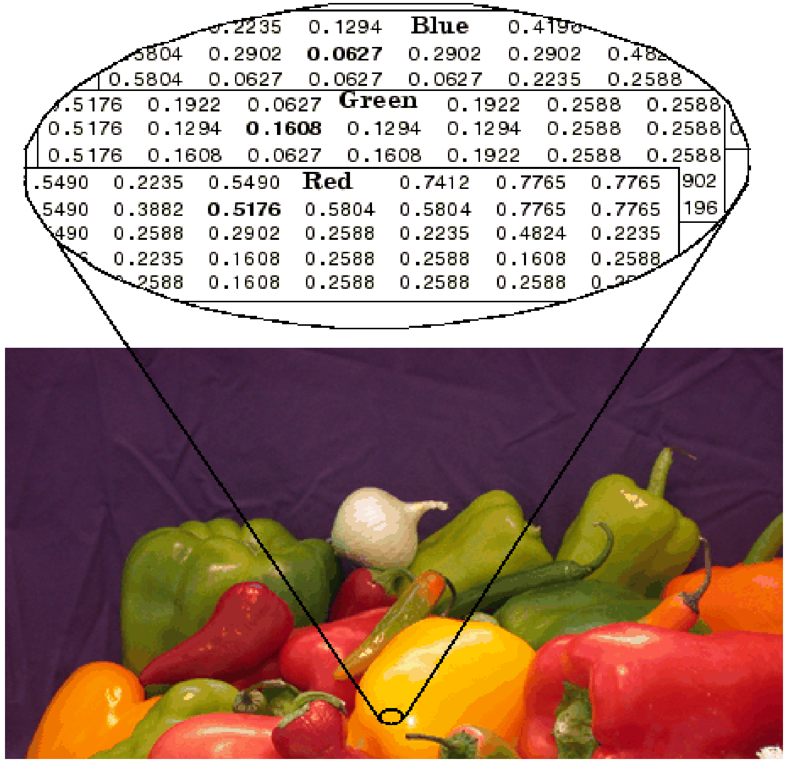
Monochrome image

$R[x,y] = G[x,y] = B[x,y]$

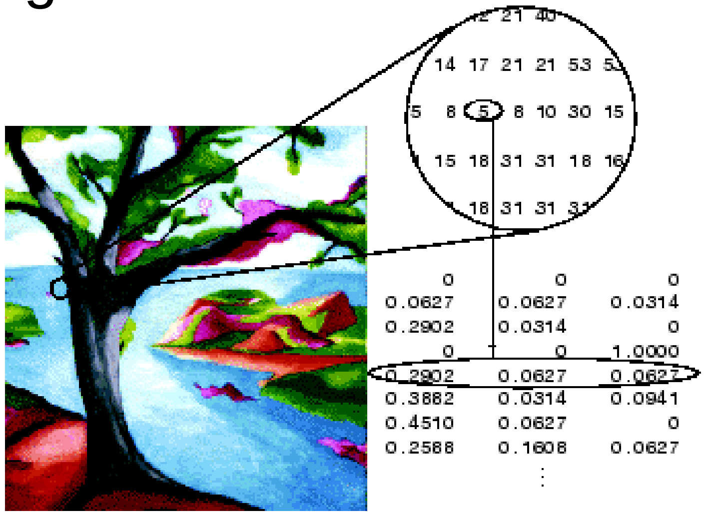
Color image (RGB)

Digital images representation

- A **truecolor image** is an image in which each pixel is specified by <u>three values</u> (RGB), one each for the red, blue, and green components of the pixel's color.



- An **indexed image** consists of an array and a *colormap matrix* (LUT). The pixel values in the array are direct indices into a colormap. The colormap matrix is an m-by-3 array of values in the range [0,1]. Each row of map specifies the red, green, and blue components of a single color.



Digital image formation system

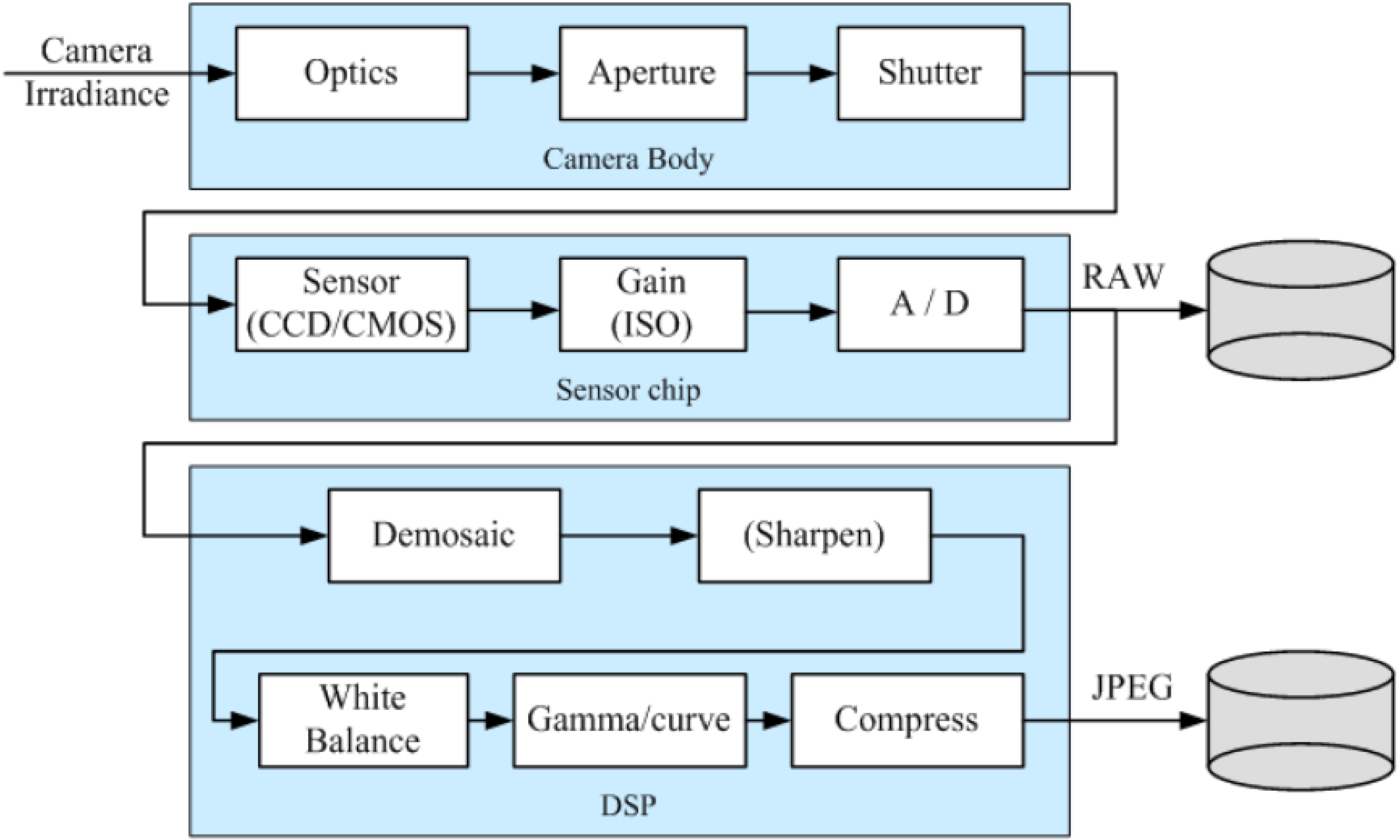
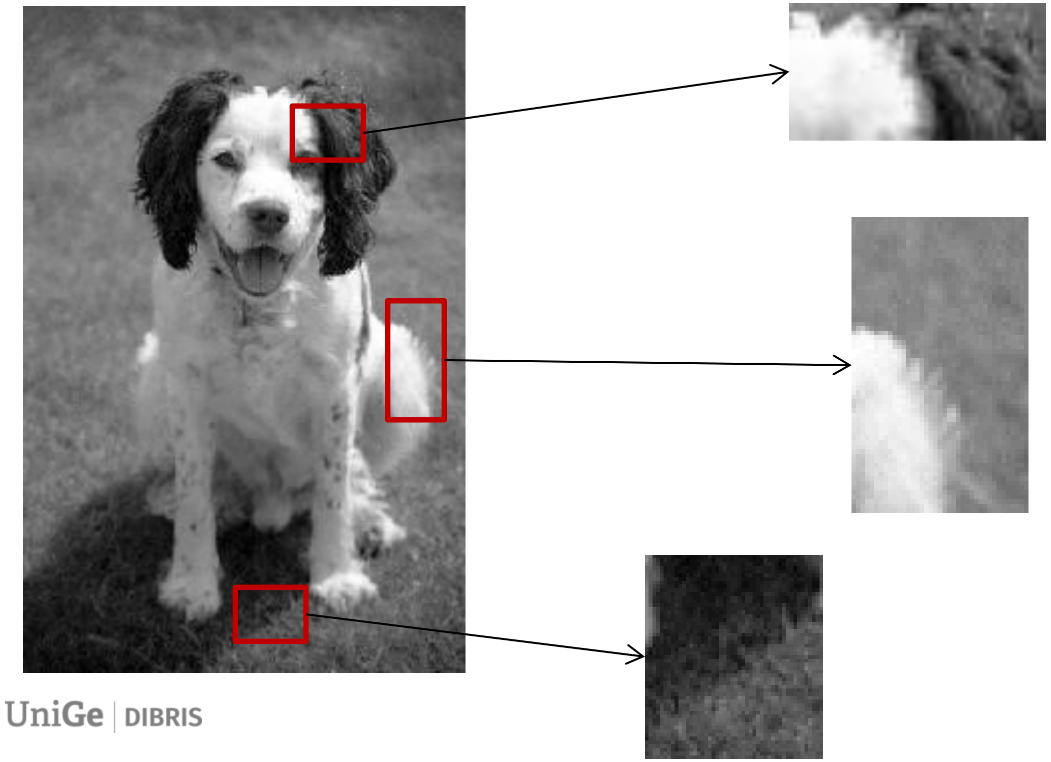


Image features: edges

- Edges: discontinuities in intensity
  - Boundaries of material properties
  - Boundaries of objects
  - Boundaries of lighting



- Edge detection:
- **Goal:** Identify visual changes (discontinuities) in an image.

- **Why?** Intuitively, semantic information is encoded in edges (e.g. to recognize objects and to *recover geometry and viewpoint*).

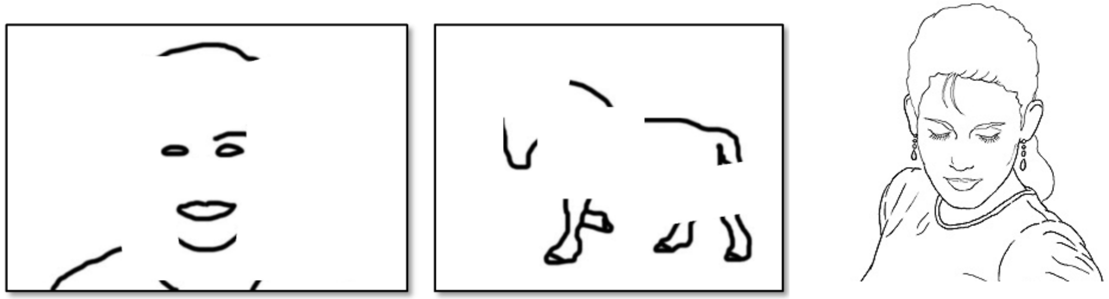
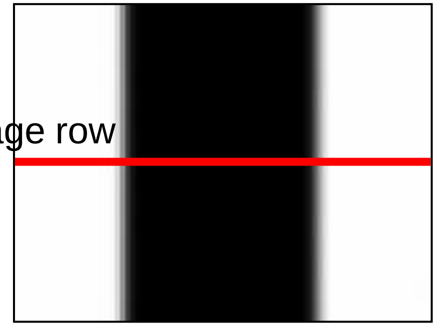
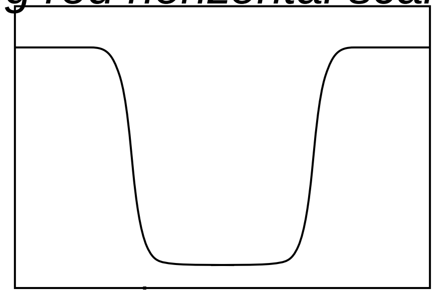
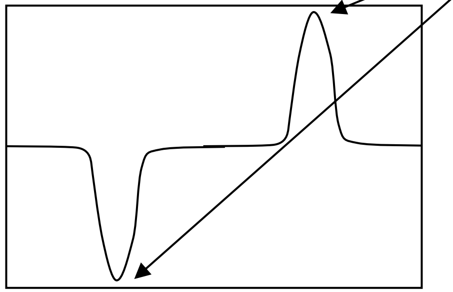


Image features: edges

- <u>Discontinuities</u> in signal can be **detected** by computing the **derivative of the signal**.







- In particular, we compute the **<u>image gradient</u>**:

$$\nabla I(x,y) = [\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}]^T = [I(x,y)_x, I(x,y)_y]^T = [I_x, I_y]^T$$

Plotted as a **vector field**, the *gradient* vector at each pixel *points "uphill".*

The gradient indicates the direction of steepest ascent.

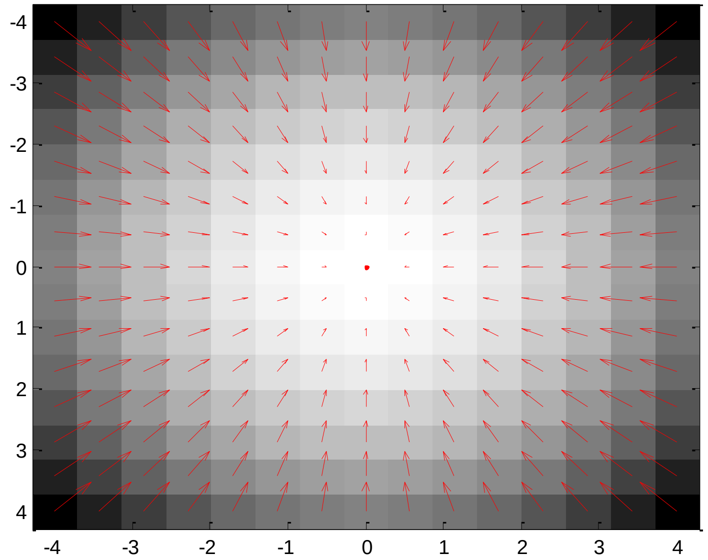
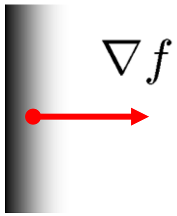


Image features: image gradient

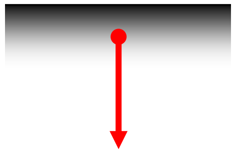
The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

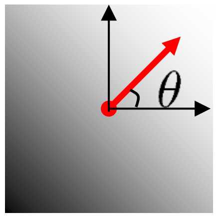
The gradient points in the direction of most rapid change in intensity



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$



$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient direction (*orientation of edge normal*) is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

The *edge strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

Image features: numerical derivatives

- We can compute the gradient vector at each pixel by convolving image with horizontal and vertical derivative filters.

- By considering the *finite forward difference* and h=1 for images, we have:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y)-f(x,y)}{1}$$

the partial derivative with respect to columns as

img(i,j+1) – img(i,j)

- It can be considered as a convolution with the kernel [1 , -1]

1	-1

Image features: numerical derivatives

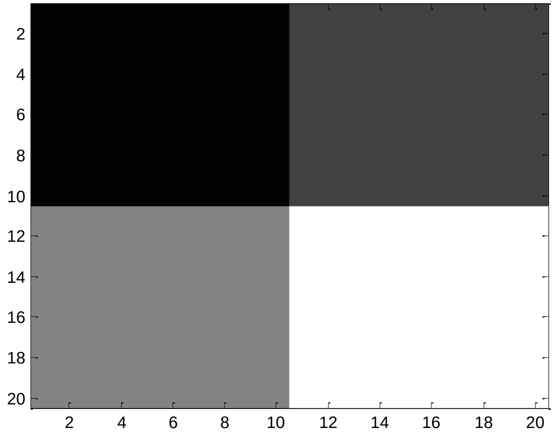
$$I(x,y)$$

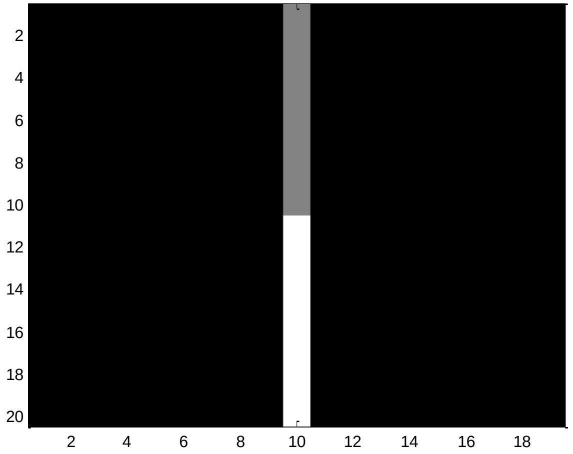
$$I_x = \frac{\partial I(x,y)}{\partial x}$$

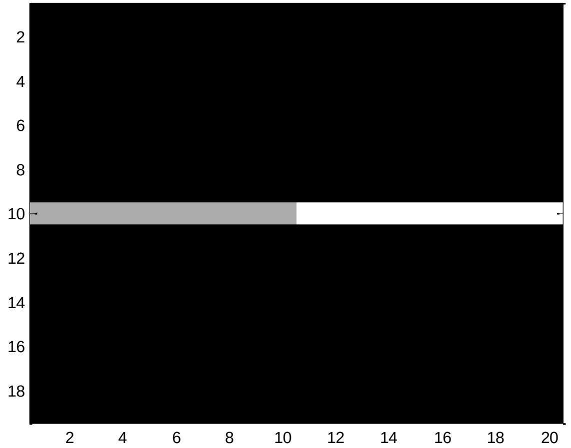
1	-1

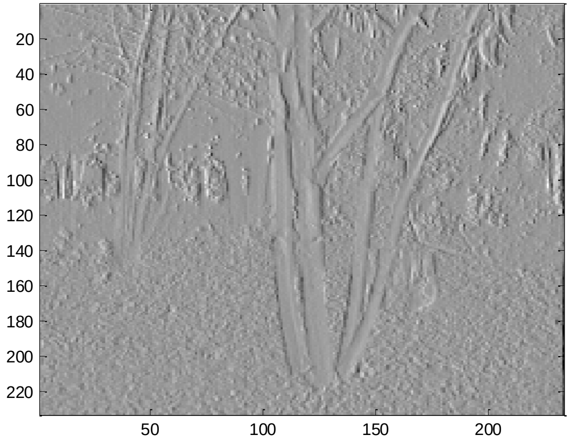
$$I_y = \frac{\partial I(x,y)}{\partial y}$$

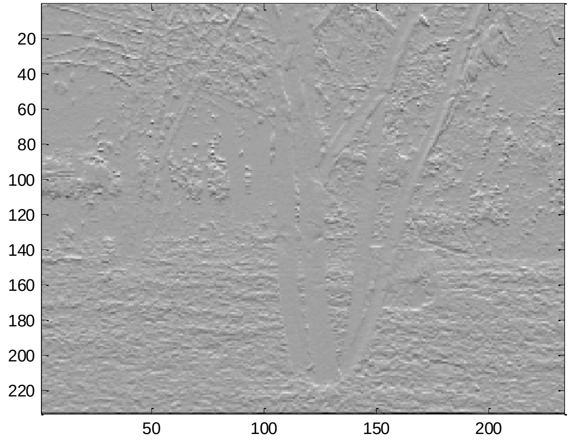
1
-1











Simple Edge Detection Using Gradients

- Issue: the noise
  - **smooth before differentiation**
  - two convolutions: to smooth, then to differentiate
  - actually, we can use the derivative of the kernel → $\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$

For instance, *Prewitt's operator*:

$$\longrightarrow \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & -1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Smooth Differentiate

$$\longrightarrow \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 \\ -1 \end{bmatrix} \longrightarrow \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

Simple Edge Detection Using Gradients

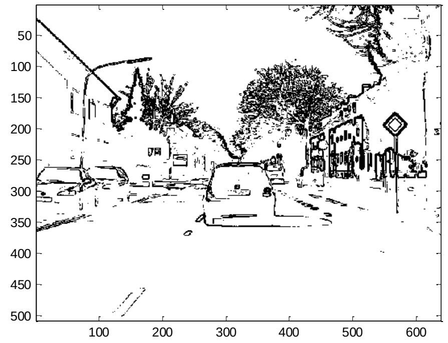
- ***Sobel edge detector***:

Image I

$$* \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \rightarrow \frac{\partial}{\partial x} I$$

$$* \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \rightarrow \frac{\partial}{\partial y} I$$

$$\sqrt{\left(\frac{\partial}{\partial x} I\right)^2 + \left(\frac{\partial}{\partial y} I\right)^2} \rightarrow \text{Threshold} \rightarrow$$



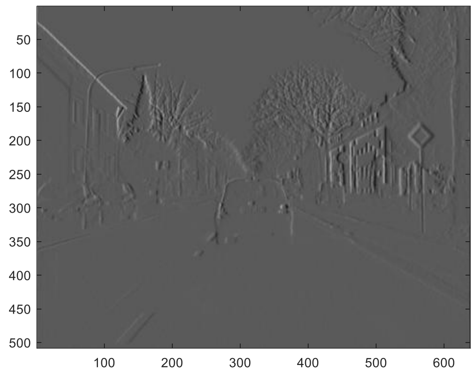
Edge map

Magnitude of gradient measures steepness of slope at each pixel (= edge contrast)

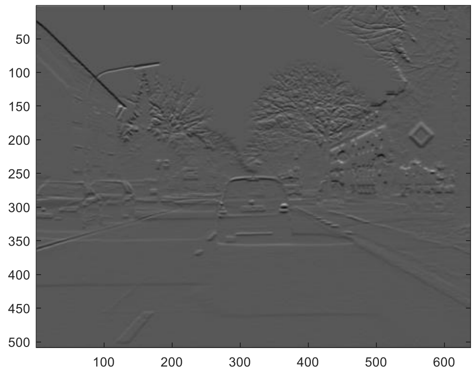
Sobel edge detector: Matlab implementation

```
dx=[1 0 -1; 2 0 -2; 1 0 -1];%mask

dy=[1 2 1; 0  0  0; -1 -2 -1];

tmprgb=imread('left_#290.bmp','bmp');

tmp=rgb2gray(tmprgb);

I=double(tmp);

figure,imagesc(I),colormap gray

Ix=conv2(I,dx,'same');%numerical derivatives

Iy=conv2(I,dy,'same');

figure,imagesc(Ix),colormap gray

figure,imagesc(Iy),colormap gray
```

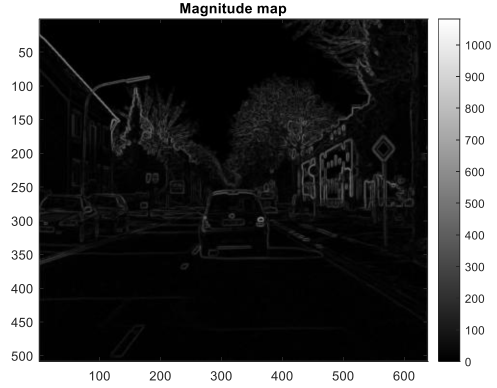
$$\frac{\partial}{\partial x} I$$



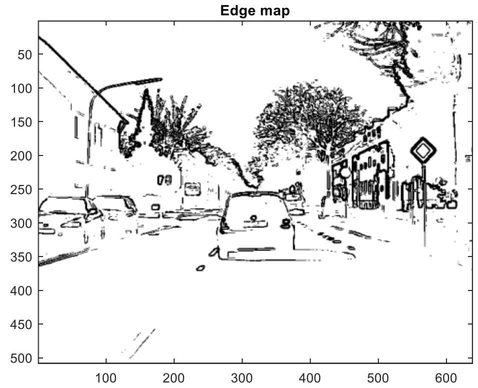
$$\frac{\partial}{\partial y} I$$



```
M=sqrt(Ix.^2 + Iy.^2);%magnitude

figure,imagesc(M),colormap gray, colorbar, title('Magnitude map')
```



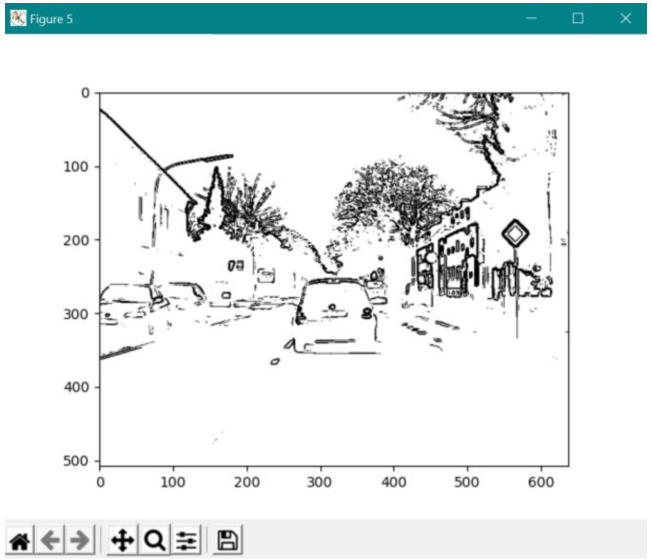

```
I_edge=M>100; %threshold

figure,imagesc(~I_edge),colormap gray,title('Edge map')%binary image
```



Sobel edge detector: Python implementation

```python
import numpy as np

from scipy import signal

from PIL import Image

from matplotlib import pyplot as plt

im = Image.open('left_#290.bmp')

img = im.convert("L")

plt.figure()

plt.imshow(img,cmap='gray')

kh = np.asarray([[1, 0, -1], [2, 0, -2], [1, 0, -1]])

kv = np.asarray([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])

gx = signal.convolve2d(img, kh, mode="same",boundary="symm", fillvalue=0)

gy = signal.convolve2d(img, kv, mode="same",boundary="symm", fillvalue=0)

plt.figure()

plt.imshow(gx,cmap='gray')

plt.figure()

plt.imshow(gy,cmap='gray')
```

```python
g = np.sqrt(gx * gx + gy * gy)

g *= 255.0 / np.max(g)

plt.figure()

plt.imshow(g, cmap='gray')

plt.figure()

plt.imshow(255-g>210, cmap='gray')

plt.show()
```



Then … to use libraries …

```
sobelmap = edge(I,'sobel');
```

```python
import cv2
sobelx = cv2.Sobel(img,cv2.CV_64F,1,0,ksize=3)
```

ARCore SDK for Unity