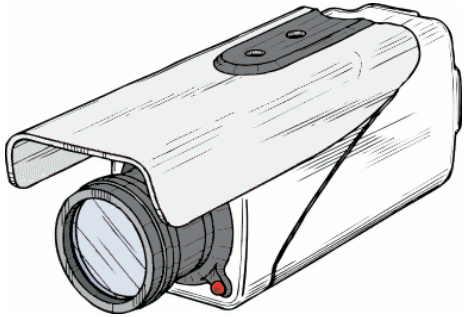
Univerza v Ljubljani



ViCOS visual cognitive systems lab

# Machine perception
# Introduction



Matej Kristan

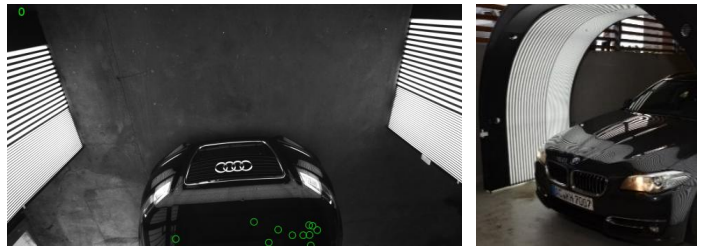

Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

# About the lecturer

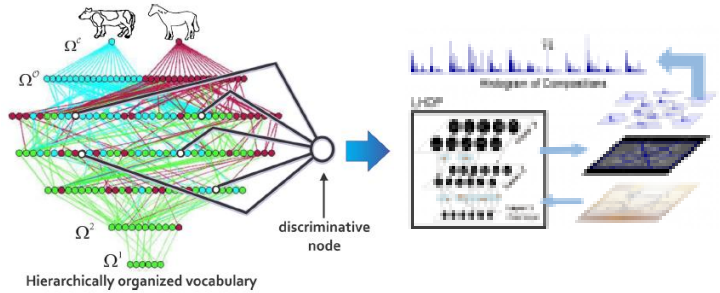
- Name: Matej Kristan
- Where to find me: 2nd floor, ViCoS
  (not in office, in the lab most of time)

- Online contacts and resources:
  - www.vicos.fri.uni-lj.si/matejk
  - ResearchGate
  - Google Scholar
  - eclassroom (https://ucilnica.fri.uni-lj.si/)
  - mail:matej.kristan@fri.uni-lj.si

1. Industrial vision

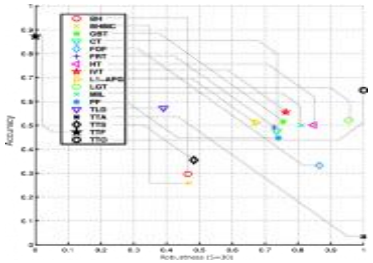


2. Deep structured models




3. Robotic vision


4. Visual object tracking

# Practicum (2-week long guided projects)

- Guided by:
  - mag. Jon Natanel Muhovič *(jon.muhovic@fri.uni-lj.si)*
  - *mag. Jer Pelhan (jer.pelhan@fri.uni-lj.si)*
  - *dr. Josip Šarić (Josip.saric@fri.uni-lj.si) [eng only]*



Start planned:
3rd week Details at the lab.

- Schedule 5 cycles, assistants will send you options to enroll

- Practicum starts next week (13.10. - 17.10.)

# Practicum (2-week long guided projects)

- Practice the theory covered in lectures
- Implementation-oriented
  - *Result is a working source code (Python)*
- Two-week assignments:
  - 1 week consultation (bug the assistants!!)
  - 1 week defense (have to defend in your assigned slot!)
- Attend *ANY* cycle for consultations (except first week), but you can defend only *IN YOUR OWN* cycle.



Start planned:
3rd week Details at the lab.

# Requirements of the course

1. Practicum (programming assignments) > 50% *each* assignment evaluated during the semester (watch out for deadlines!)

2. Written exam > 50%

   Cannot access the written exam without passing the practicum/lab.

   Content: practicum assignments + lectures

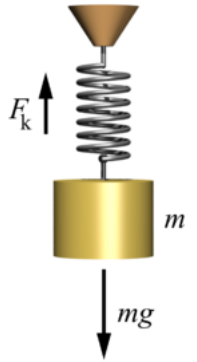
3. Oral exam:

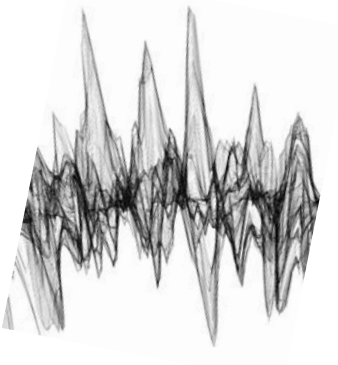
   Not necessary if written >X%

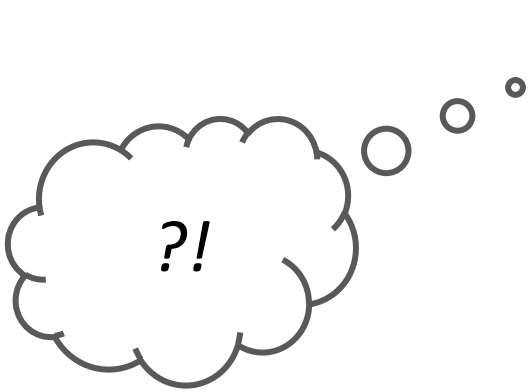
   *(Will depend on class attendance and progress at assignments)*

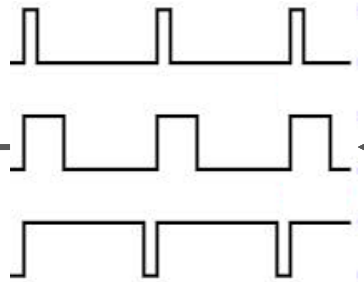
# What is machine perception about?

- Building machines that perceive their environment
- Digitalize environment through sensors
- Image of light, ultrasound, force field, etc.
- Perception through images: Computer vision





# Development of Computer Vision

- Origins: 1950-1965 as side project at MIT:

  *"...building perceiving machines would take about a decade..."*

- Development paced by hardware development (numerical maths)

First multipurpose comps
(UNIVAC ~1951)

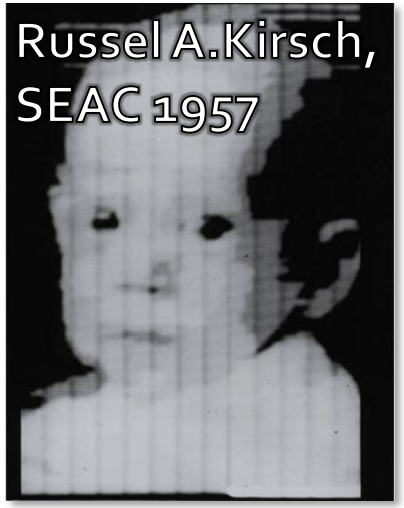



Image digitization

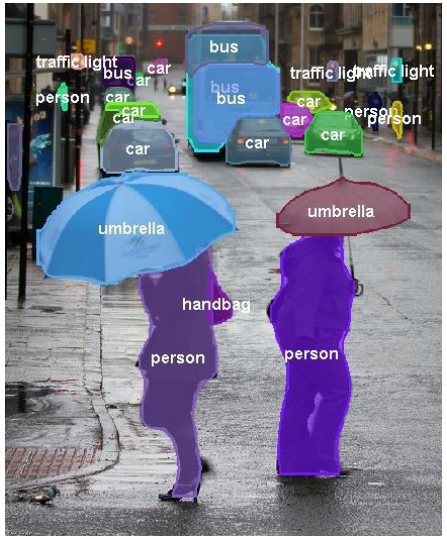
~50 years

Embedded computers
(ARM ~2001)


Face detection

~15 years

Graphic processing units
(GPU ~2016)

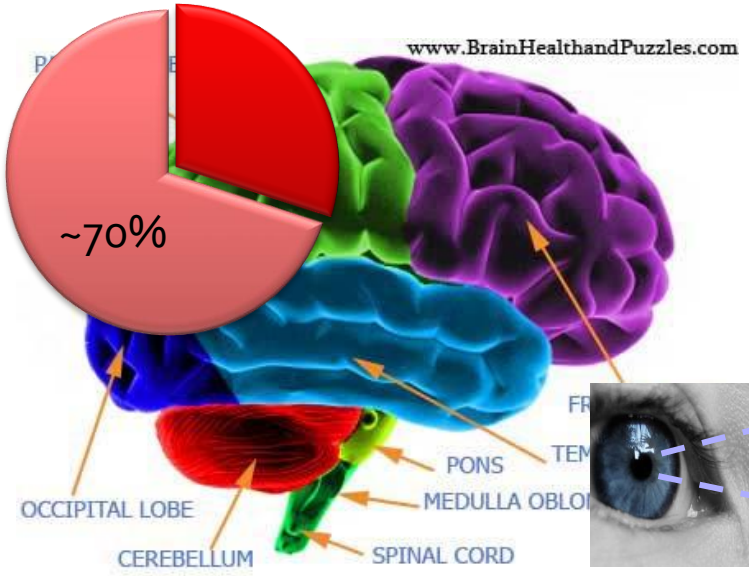



Instance segmentation

# Human vs. Computer vision

- Much harder than it looks…
- Neuroscience: >50% brain dedicated to vision*

*Prof. Cornelia Fermueller ,University of Maryland in College Park

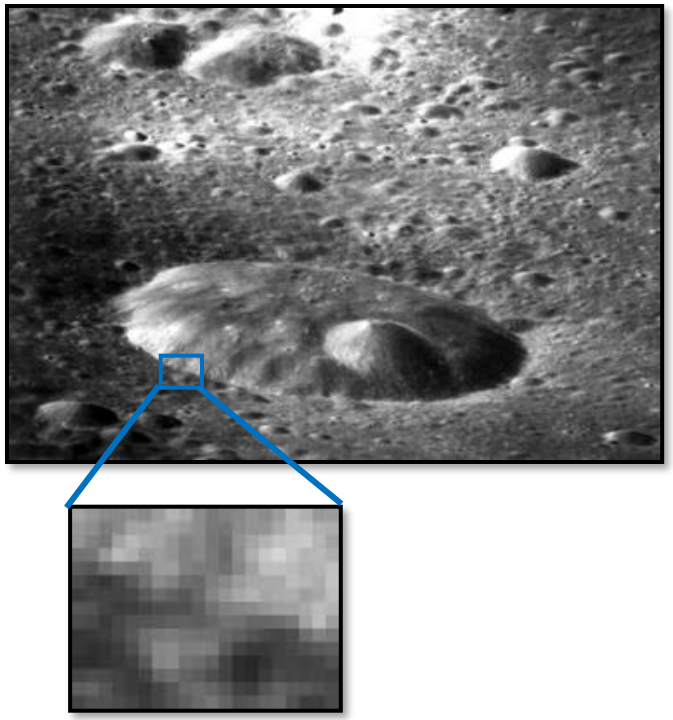





# Human vs. Computer vision

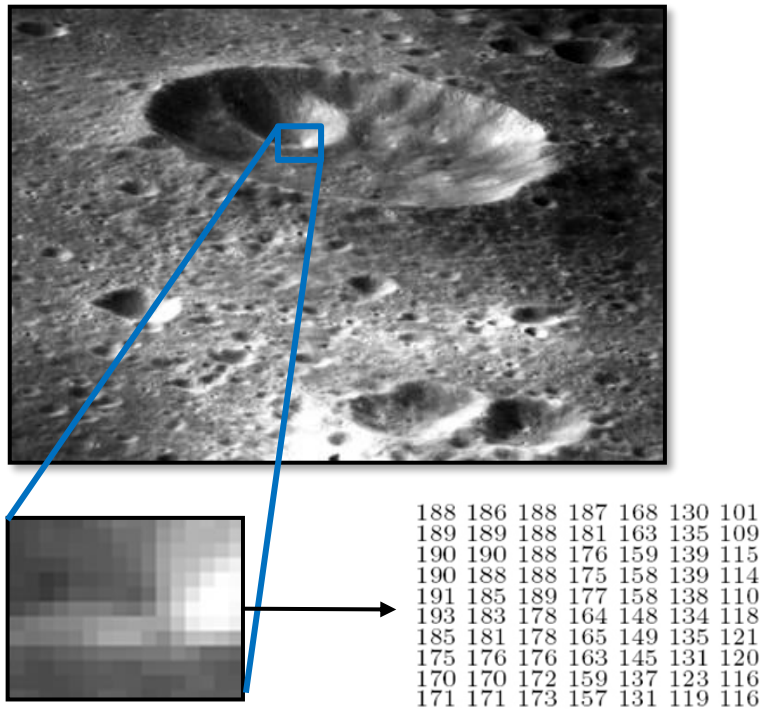
- Much harder than it looks…
- Neuroscience: >50% brain dedicated to vision
- Humans apply experience (prior knowledge)

What do you see?



The CV “tools”:
Algebra, Analysis
Statistics
Signal processing
Machine learning
Algorithms
…

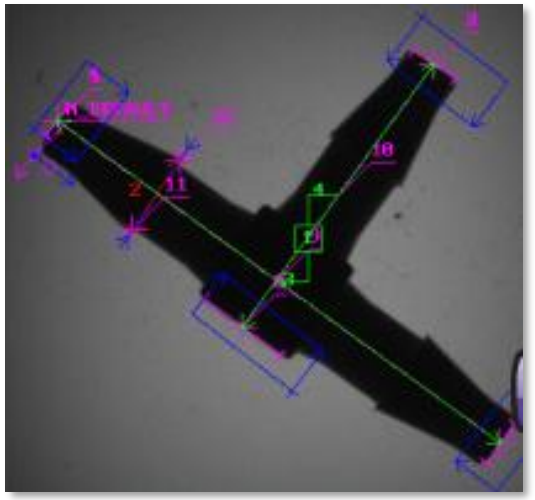
What do you see?



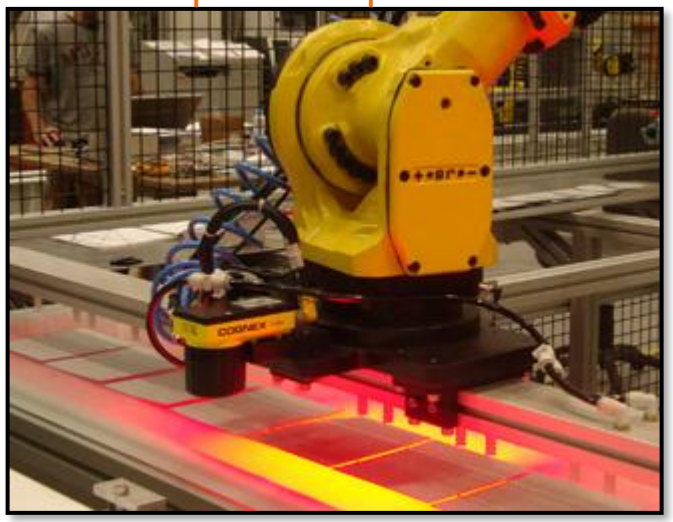
# Modern industrial applications

Industrial applications CONTROL THE SETUP.

Smart cameras http://www.matrox.com

Solar panel inspection



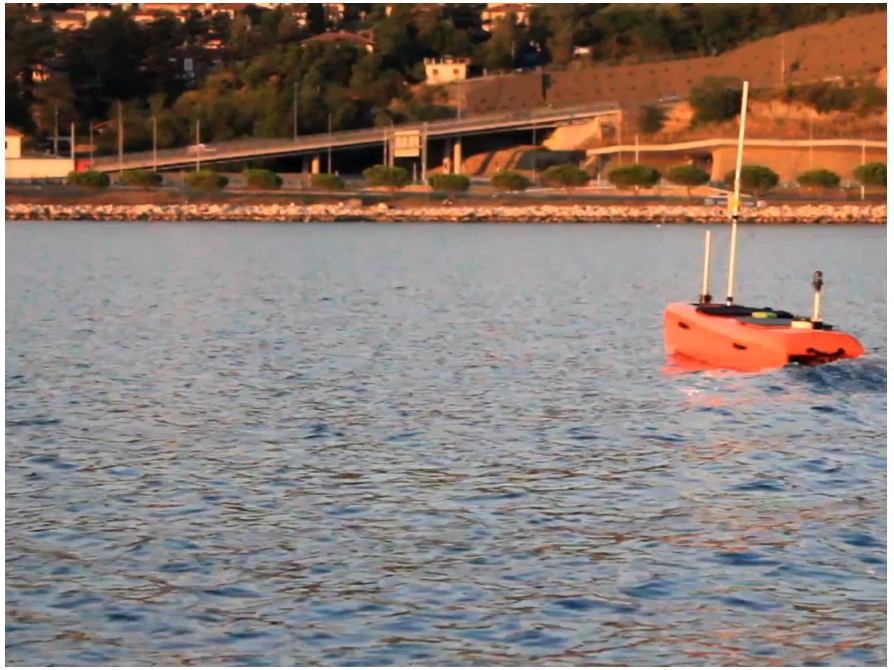
http://www.cognex.com

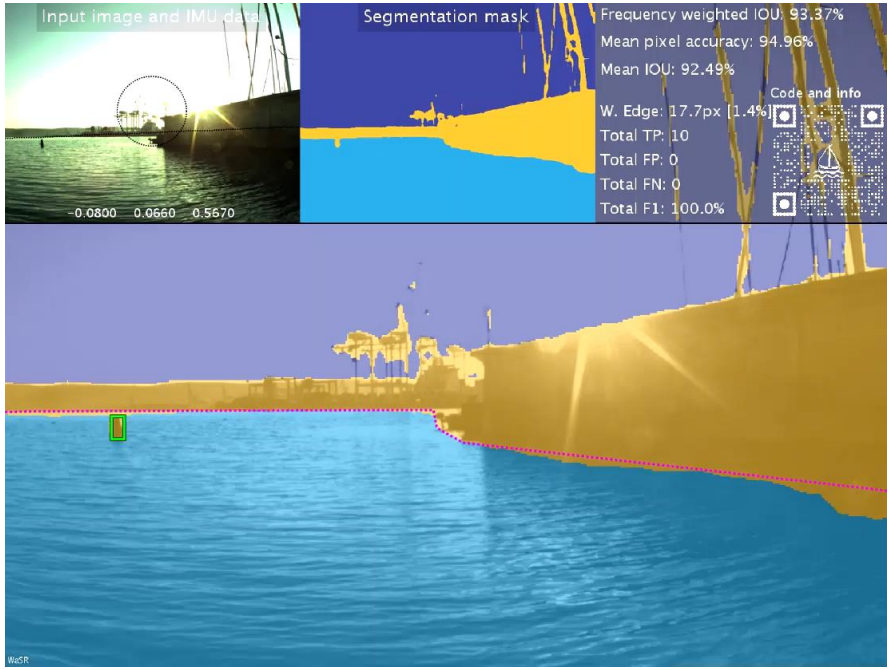
Car damage inspection

Vicos (http://www.vicos.si)

# Modern autonomous vehicles applications

Boats:
(www.vicos.si/Projects/Viamaro)



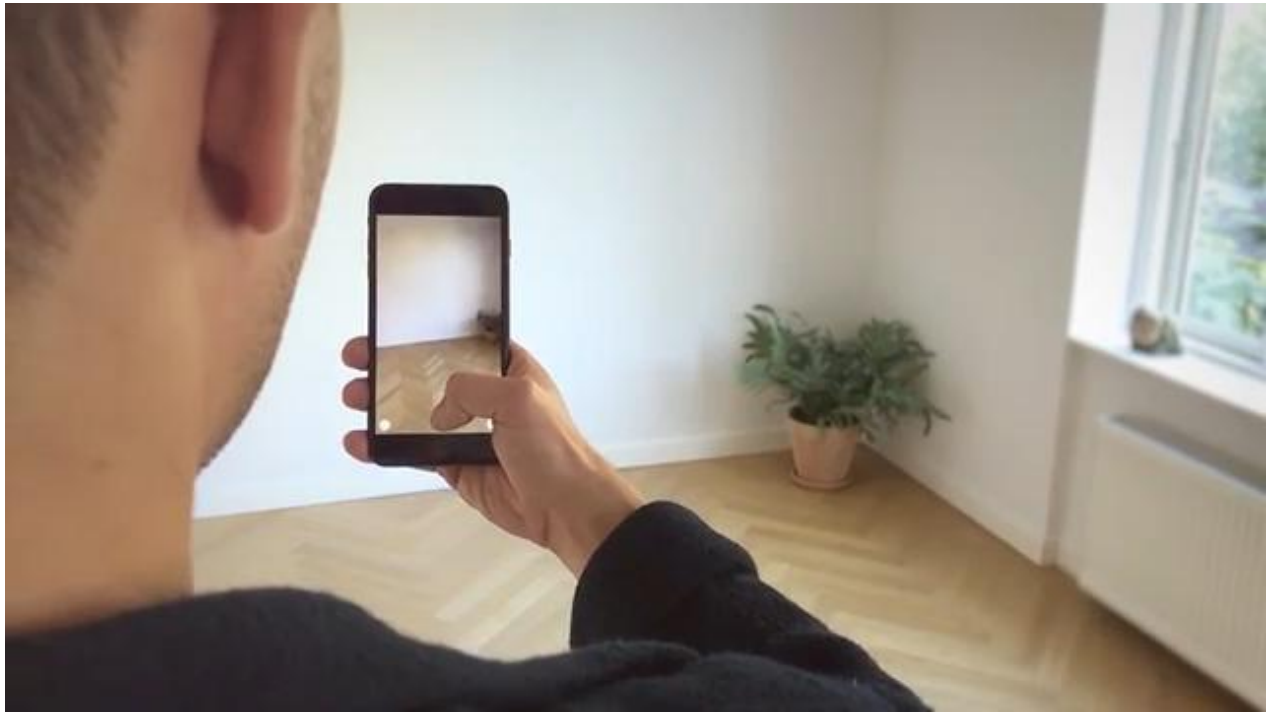



Cars: (https://youtu.be/rPj4T1__gZ4; https://youtu.be/VG68SKoG7vE)

# Modern visual query / AR applications

## MS Hololens https://youtu.be/ihKUoZxNClA


## IKEA AR



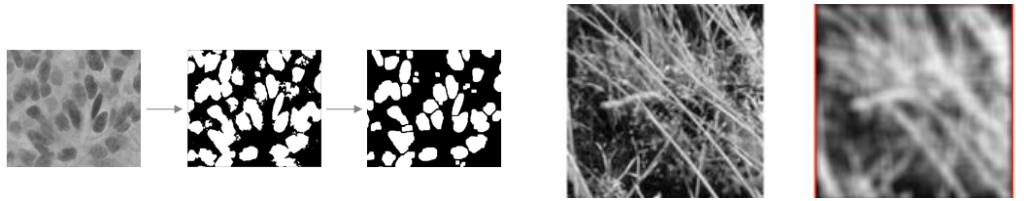
https://youtu.be/ZDWRl9A1p6s

# Topics covered in this course

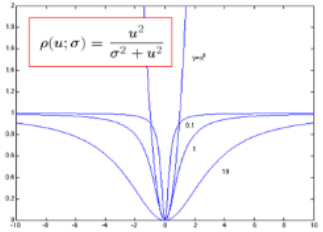
*Might change a bit…*

1. Image processing 1
2. Image processing 2
3. Edge detection
4. Fitting parametric models
5. Local features
6. Camera geometry
7. Multiple-view geometry
8. Recognition & Detection

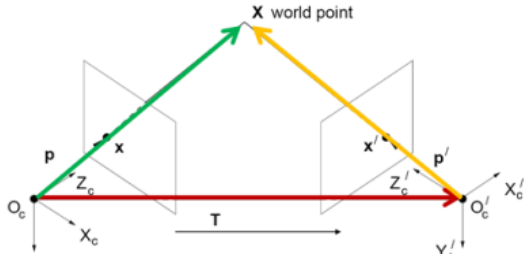
Basic image processing

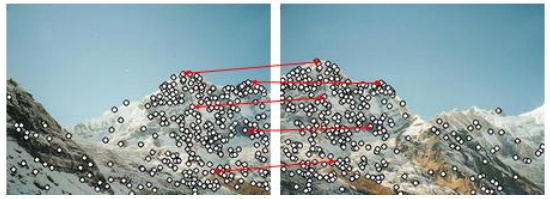


Necessary LS tools

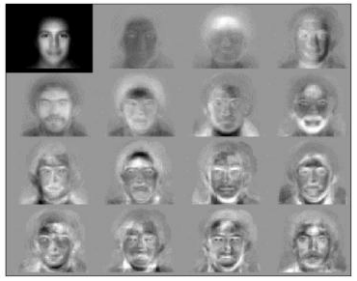


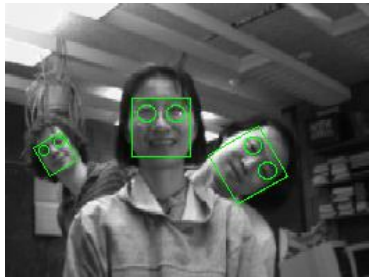

Cameras



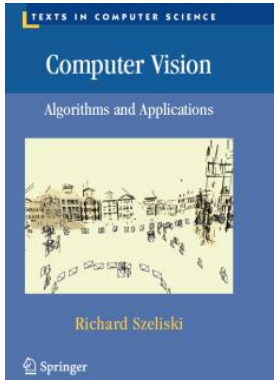
Detection and beyond





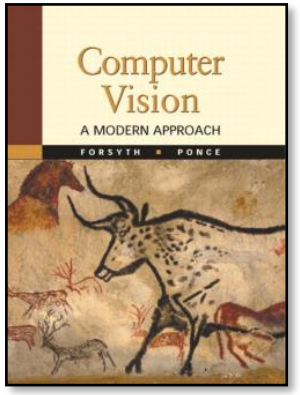
# Literature

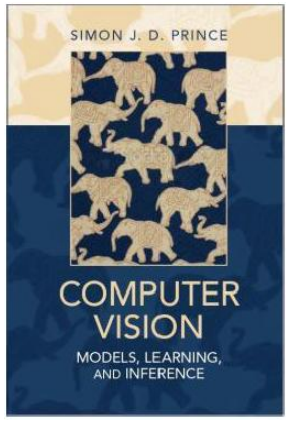
- The topics covered in lectures can be found in the following textbooks:




R. Szeliski, Computer Vision: Algorithms and Applications, 2010
Available online:
http://szeliski.org/Book/

David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition)
(first edition available online)




Simon J.D. Prince, Computer Vision: Models, Learning, and Inference, 2010
Available online:
http://www.computervisionmodels.com/

Considerable book collection:
http://homepages.inf.ed.ac.uk/rbf/CVonline/books.htm

# Literature

- Use the books for studying and solving the practicum assignments
- Lecture slides will be made available from the e-classroom
  - Hopefully a few days before the next lecture
- Slides are not books!
- You will need to make your own notes to properly follow the course

Crucial: *be proactive*

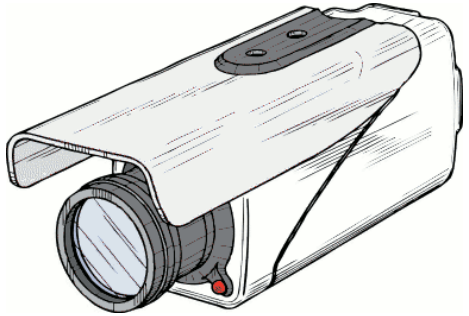
- Attend the lectures and make notes!
- Ask questions (in class and *especially at the practicum – come prepared*)!
- For translation of terms, see: *https://terminoloski.slovenscina.eu/*

Univerza v Ljubljani

ViCOS visual cognitive systems lab

# Machine perception
# Image formation & Image processing 1

Matej Kristan

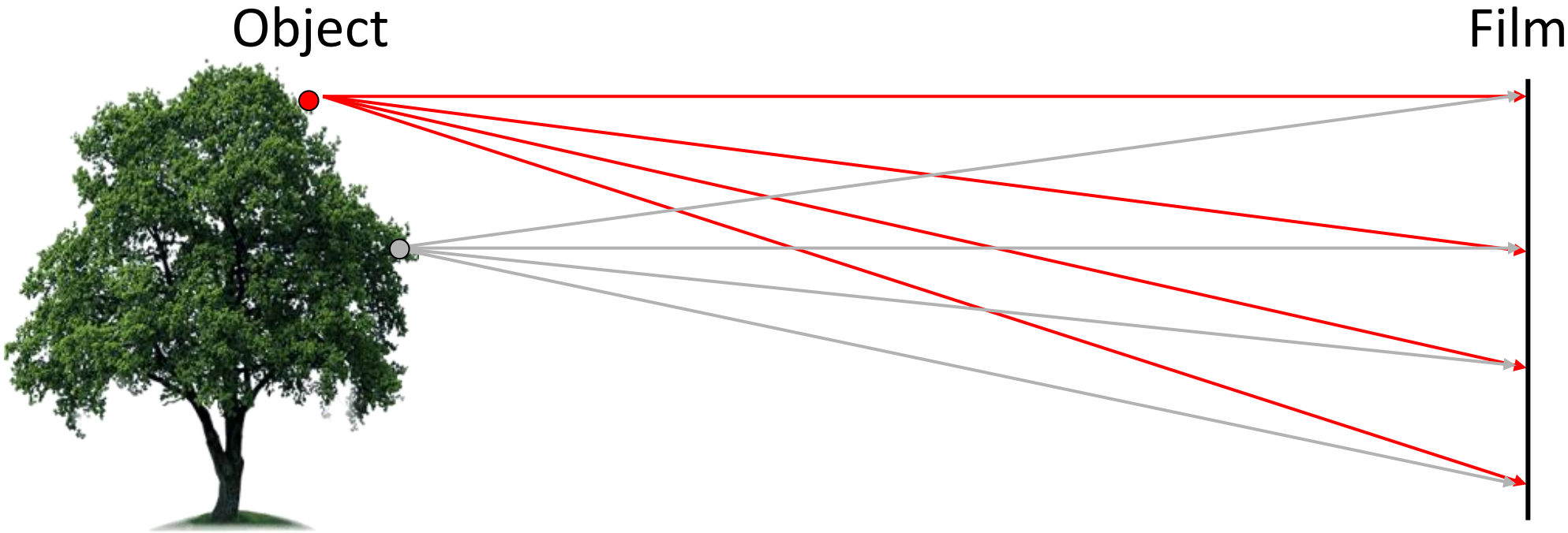



Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

Machine perception

# IMAGE FORMATION

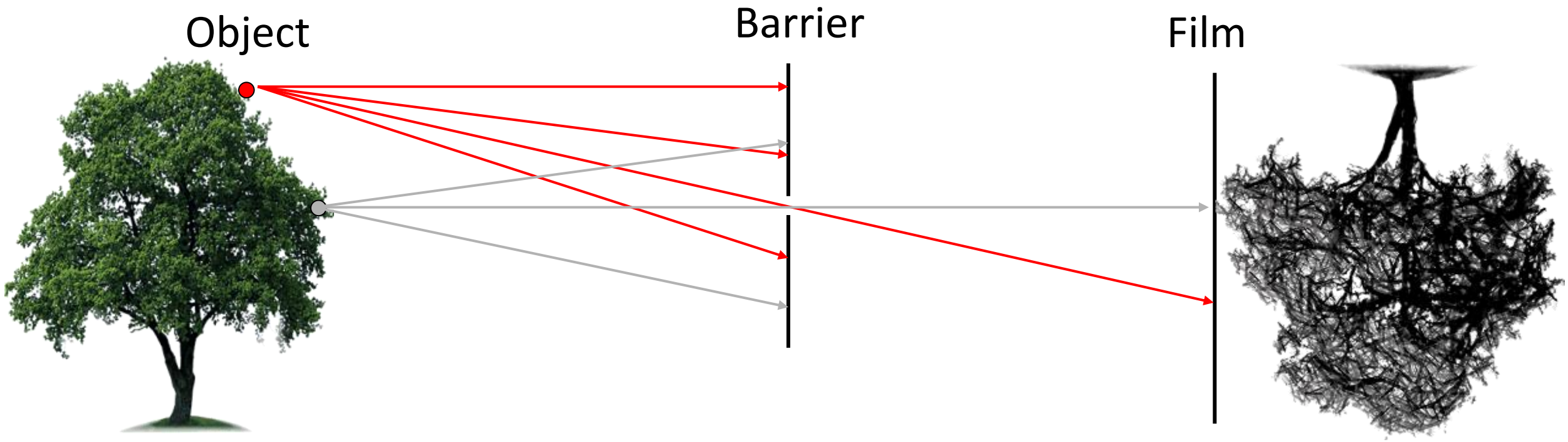
# Let’s design a camera!




- Idea 1:  put an object in front of a film...

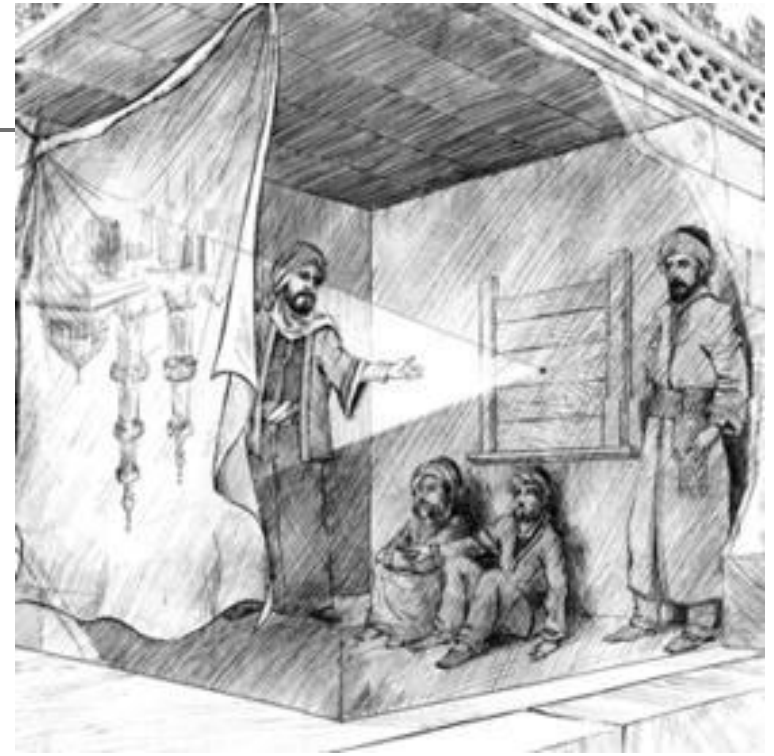
- Do we get a good image of the object?

# Let’s design a camera!

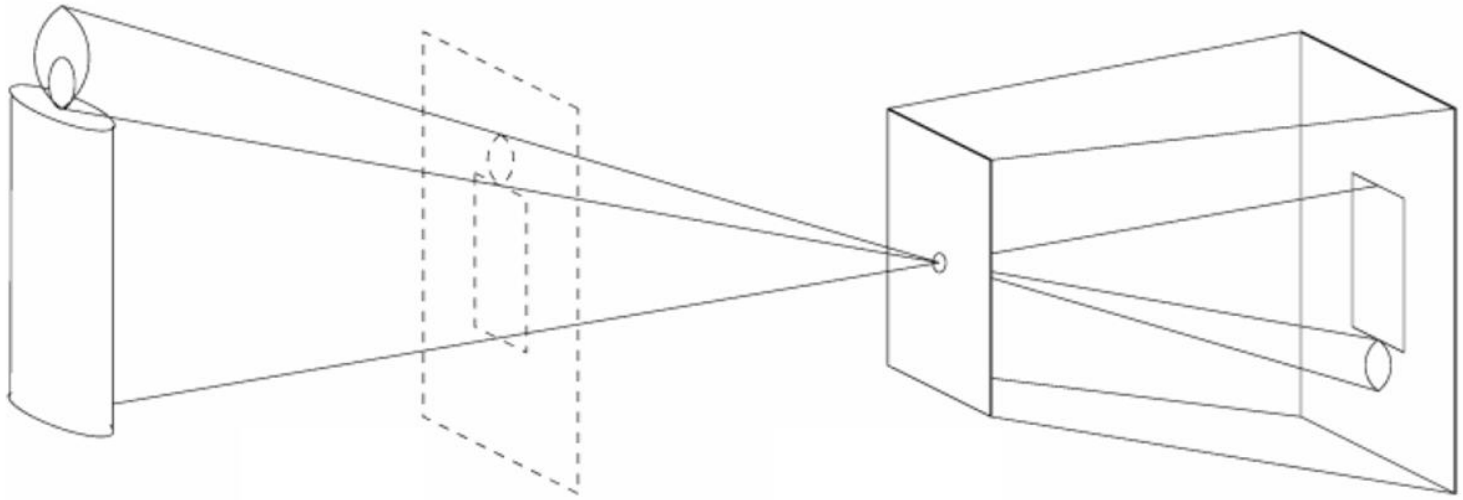



- Add a punctured barrier that blocks most of the rays
  - Significantly reduces blurring
  - The „hole“ is known as aperture

# A pinhole camera

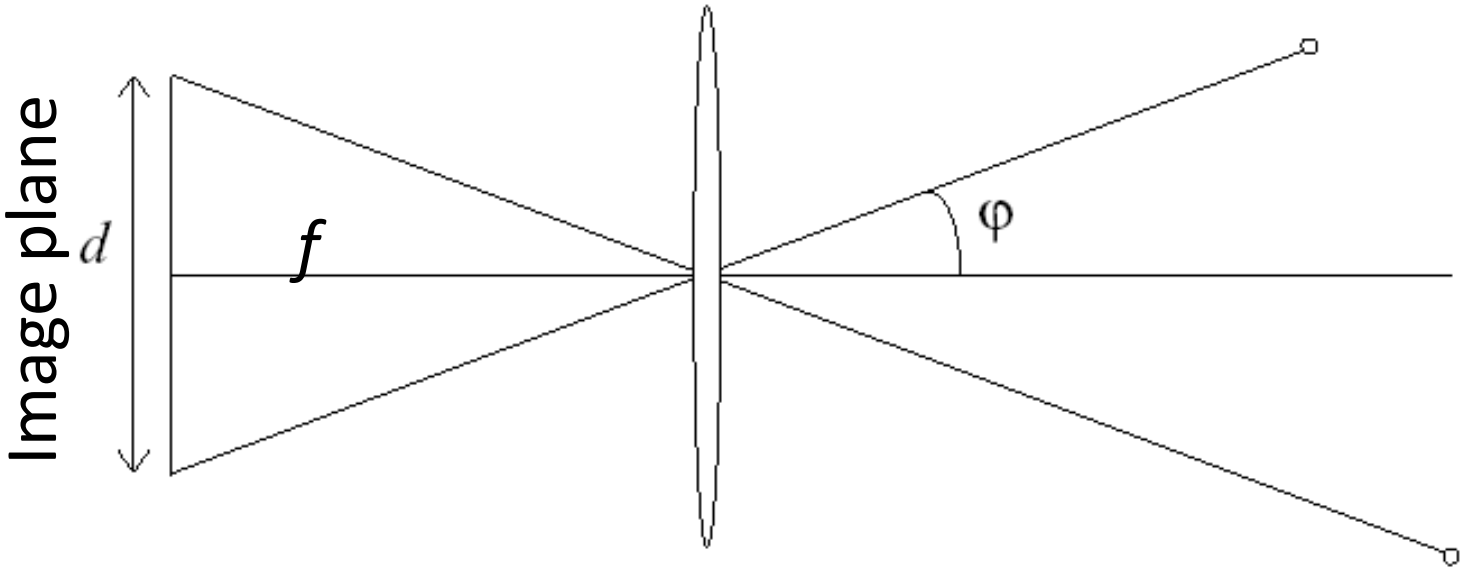


- Earliest and remarkably correct written description:
  ~500 BC Mohist canon [founder Mo-ti]
  (ancient Chinese texts)
- A simple standard camera model
  - A box with a small aperture
  - Works in practice



# Field of view

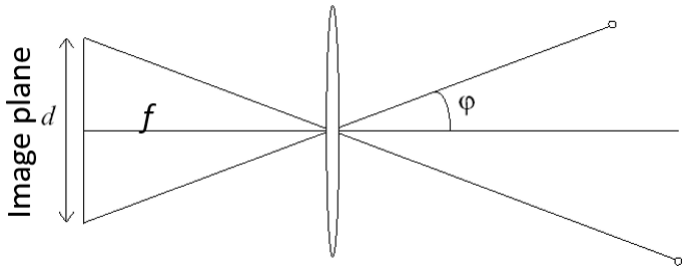
- Field of view (FOV) $(2 \times \varphi)$ is an angular measure of space perceived by the camera.




$$\varphi = \tan^{-1}\left(\frac{d}{2f}\right)$$

# Field of view

- Small $f$ results in wide-angle image (Large field of view) ⟶
  - More 3D points project to the sensor.

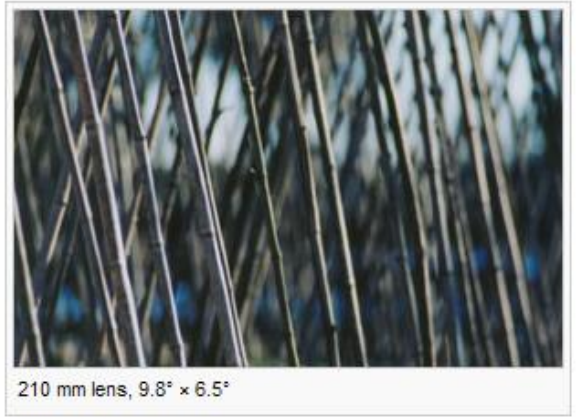



- Large $f$ results in a telescopic image (small FOV) ⟶
  - Smaller portion of 3D scene is projected to the sensor.


28 mm lens, 65.5° × 46.4°

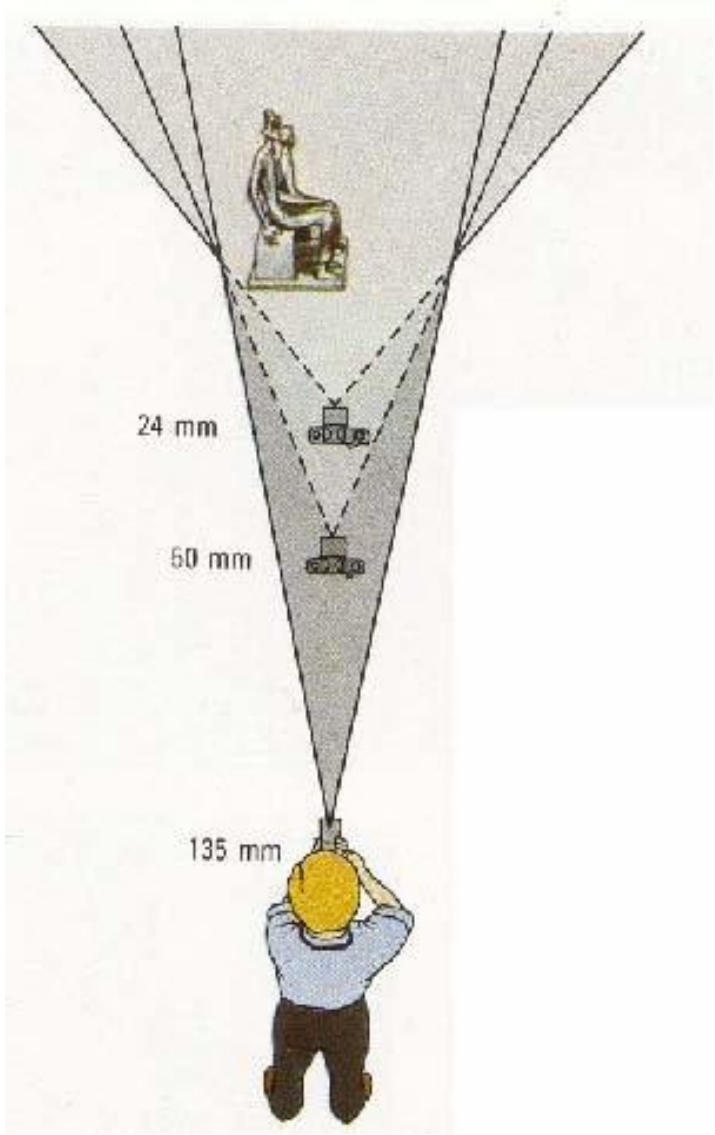

50 mm lens, 39.6° × 27.0°

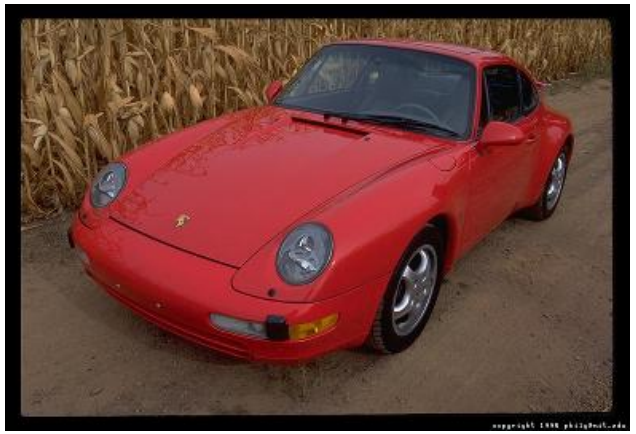

70 mm lens, 28.9° × 19.5°


210 mm lens, 9.8° × 6.5°

$$\varphi = \tan^{-1}\left(\frac{d}{2f}\right)$$

# Field of view and focal length

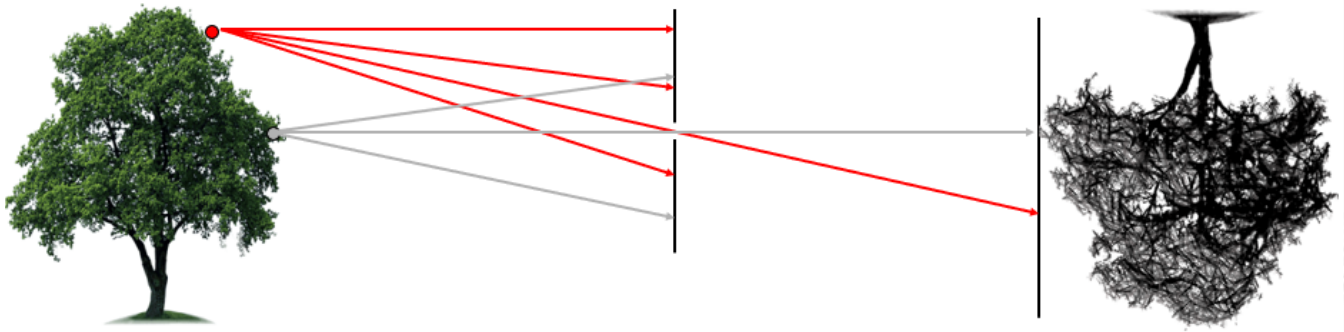


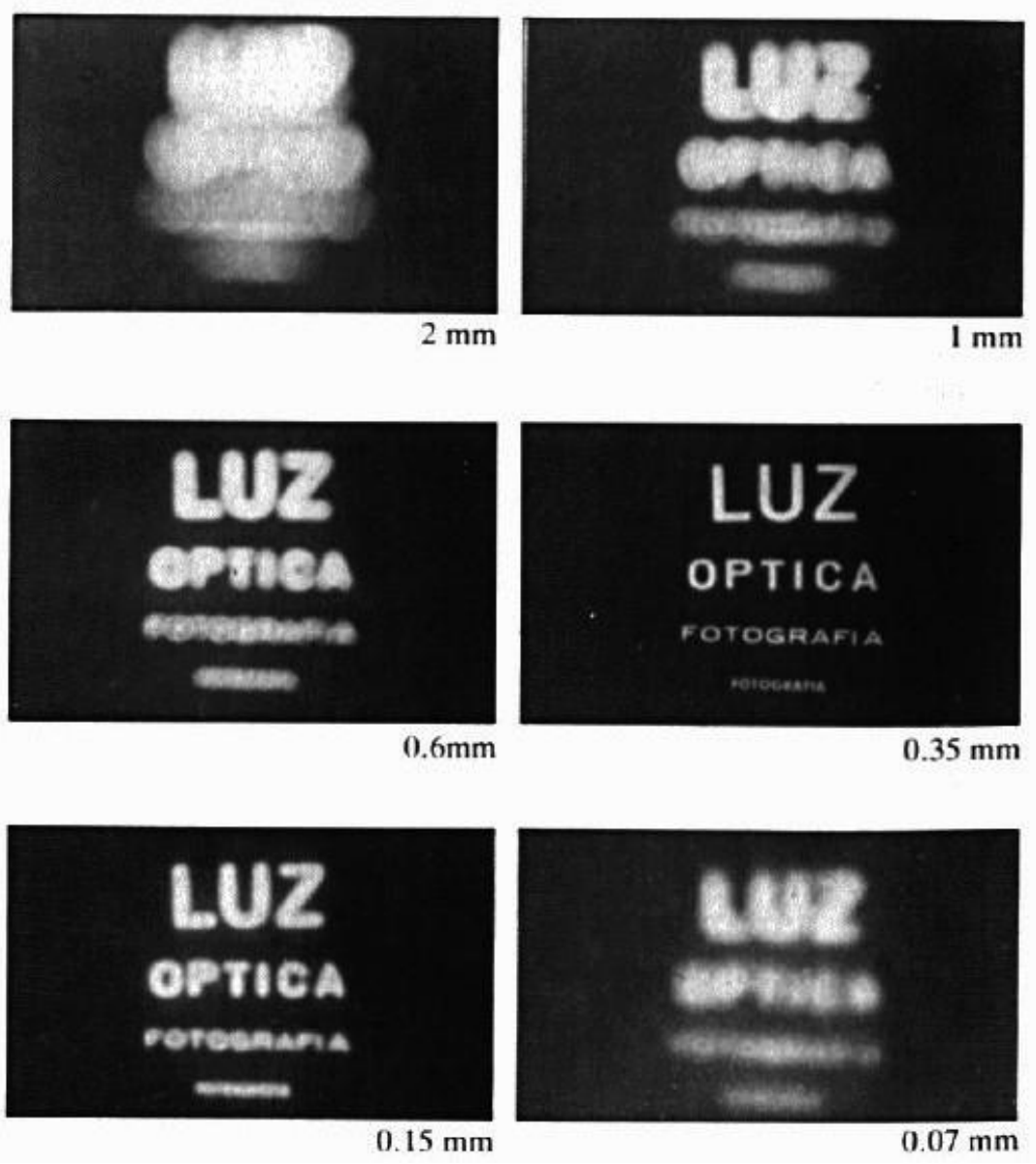



Small FOV, large *f*
Camera far away from the car

Large FOV, small *f*
Camera close to the car

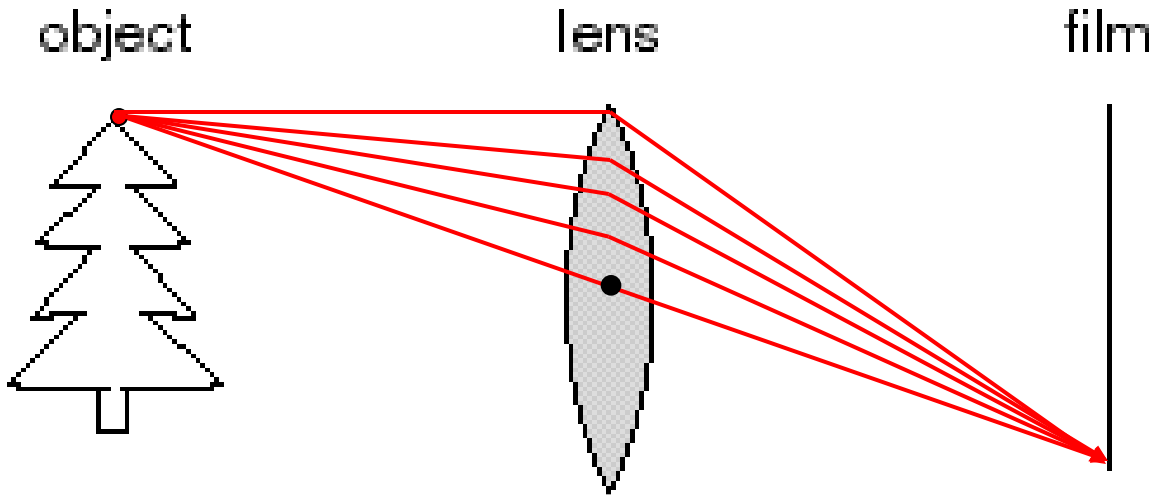
# Effects of the aperture size





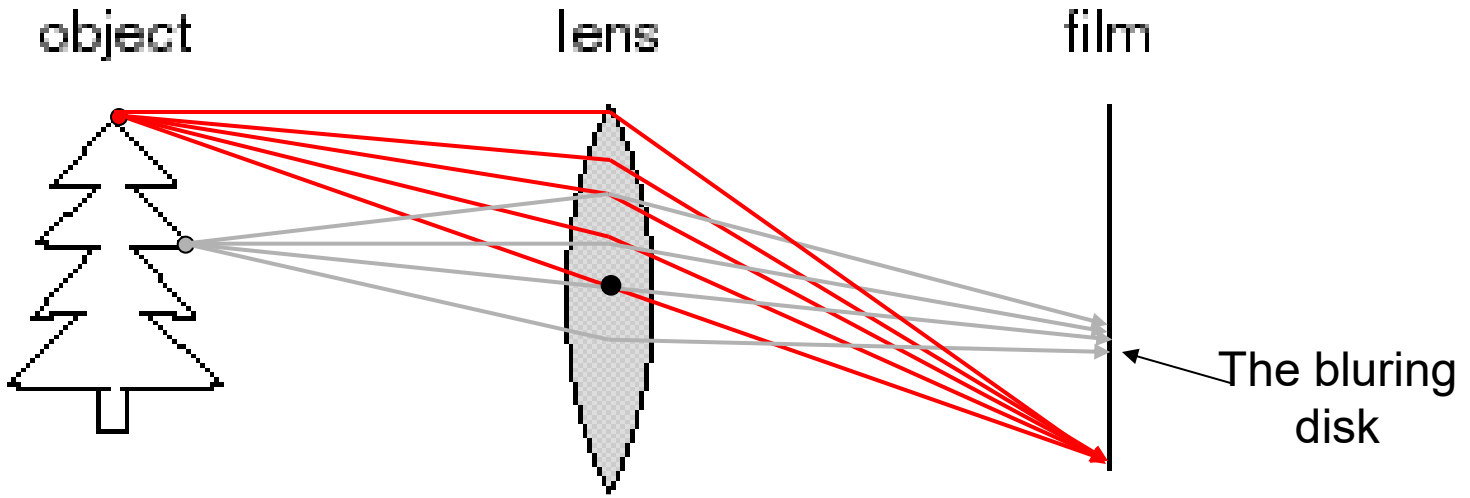

- Too large – multiple directions averaging, resulting in a blurred image.
- Too small – light starts diffracting, causing blurred image.
- In general, small number of rays hit the film, which results in a dark image.
- How do we deal with this?

# Let’s add a lens…




- The lens focuses light to film
  - The rays that travel through the center do not refract.

# Let’s add a lens…

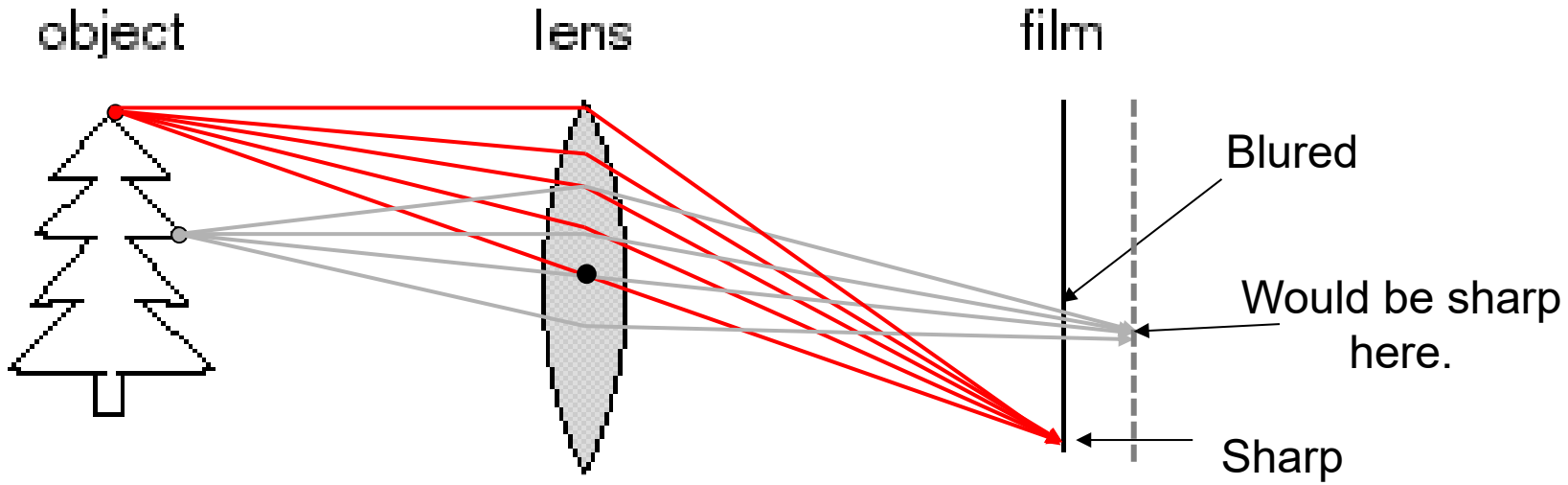



- The lens focuses light to film
  - The rays that travel through the center do not refract.
  - Points at a particular distance remain in-focus.
  - Points at other distances are blurred.

# Focus and the depth-of-field

- Thin lens: Points at different depths get focused on different depths of image plane.

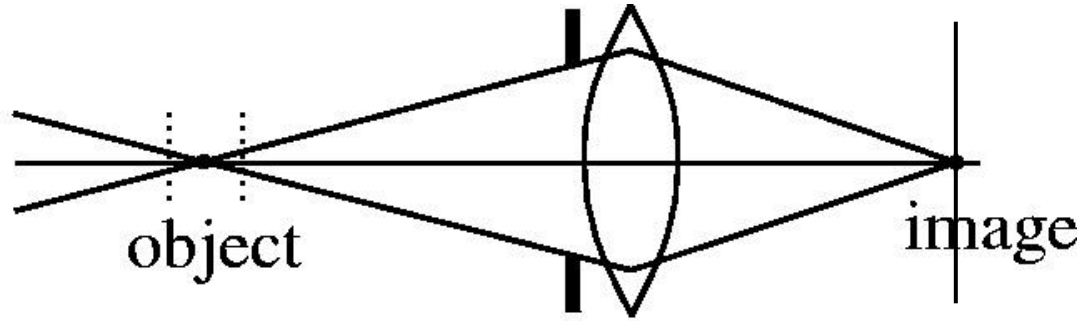
  (Real-world lens have a greater depth of field)

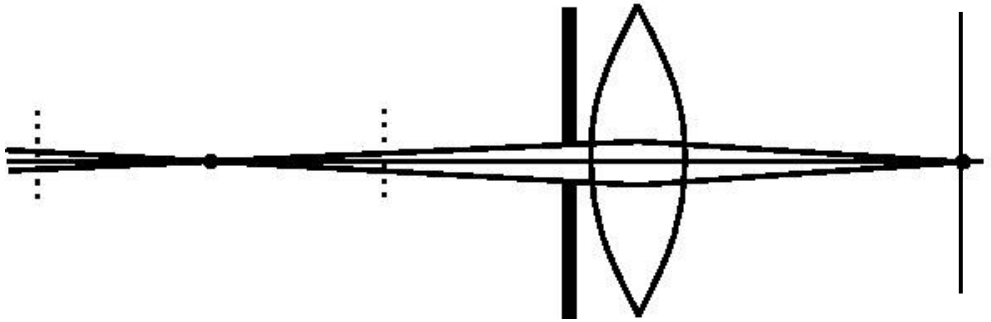



- Depth of field: distance between image planes at which the blurring effect is sufficiently small..

# Focus and the depth-of-field

- Effects of aperture on the depth-of-field

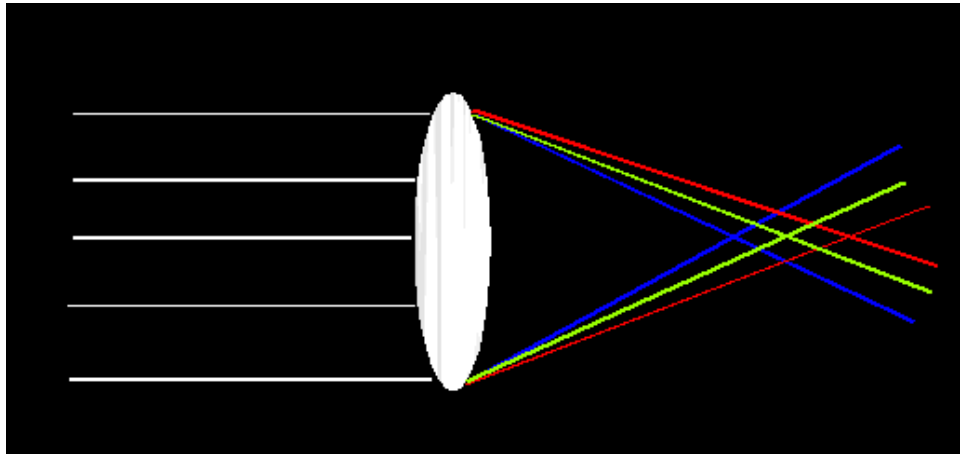


- Small aperture increases the depth-of-field.
- But due to reduced illumination we have to increase the exposure time.

# Chromatic aberration

- Different wave-lengths refract at different angle and focus at slightly different distances:
- The more oblique angle, the greater the effect (consider off-axis, edge points)
- On-axis angles are less oblique + symmetry has a cancelling effect

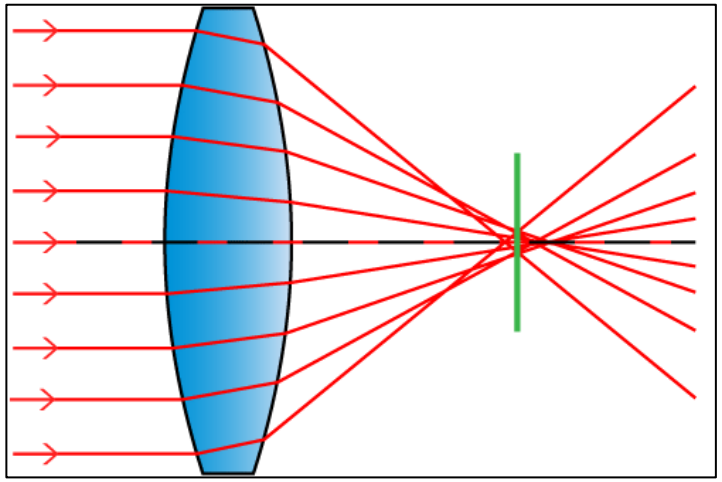


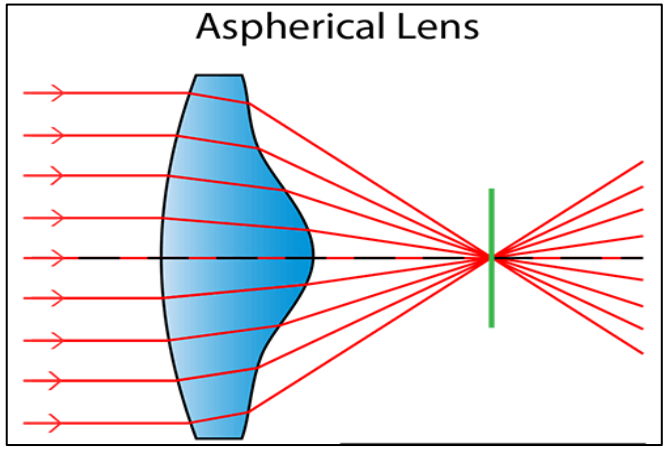
Close to image center

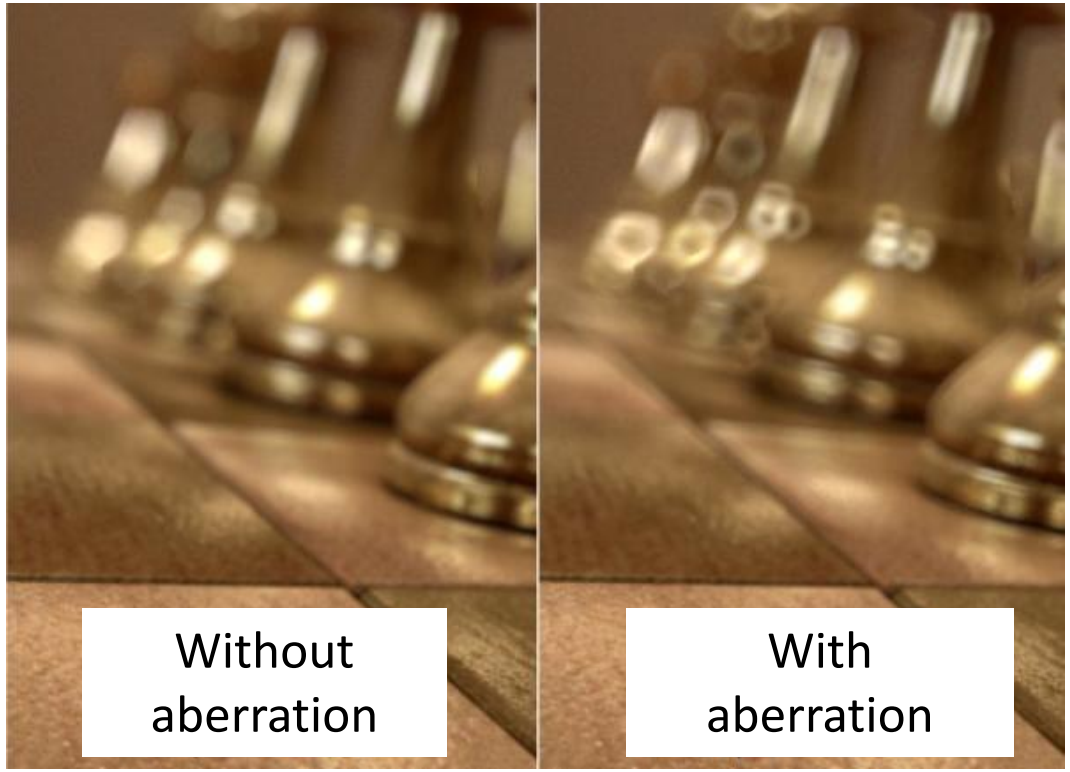
Close to image edge

# Spherical aberration

- Spherical lenses do not focus the light perfectly.
- Rays close to lens edge focus closer than those at the center.

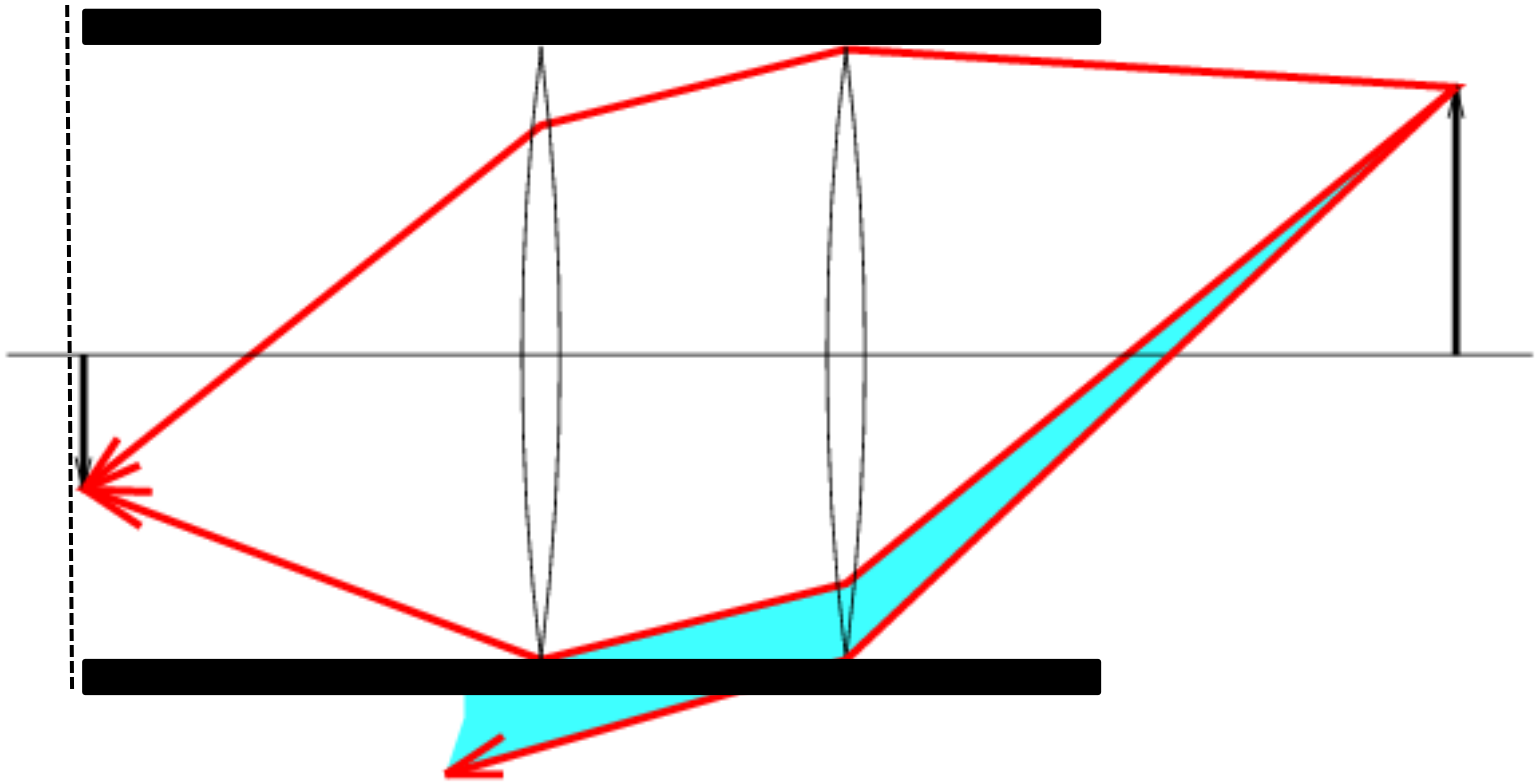




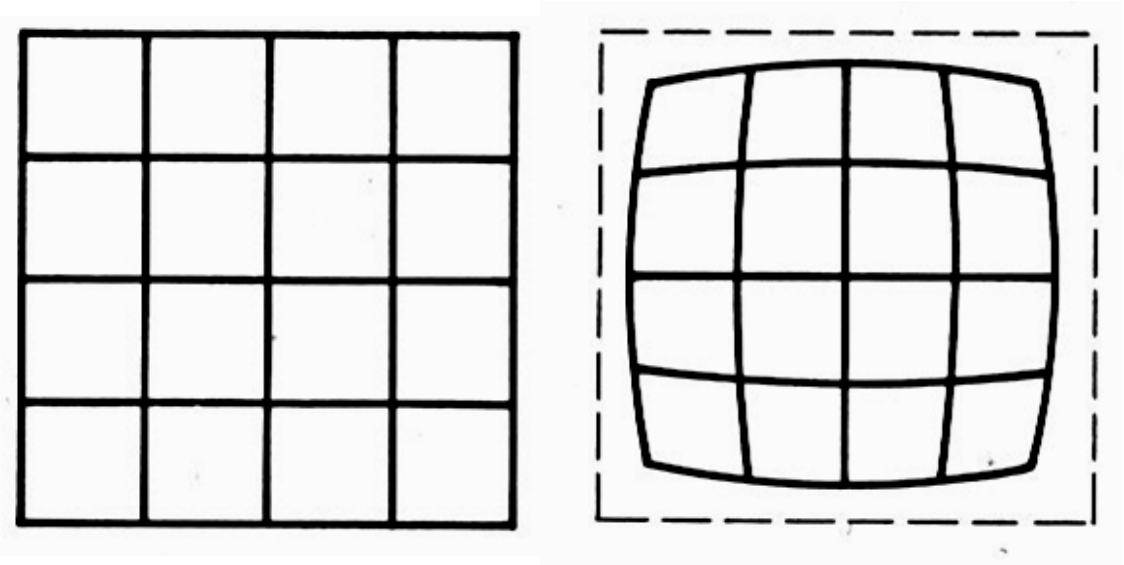




http://photographylife.com/what-is-spherical-aberration

# Vignetting



# Radial distortion

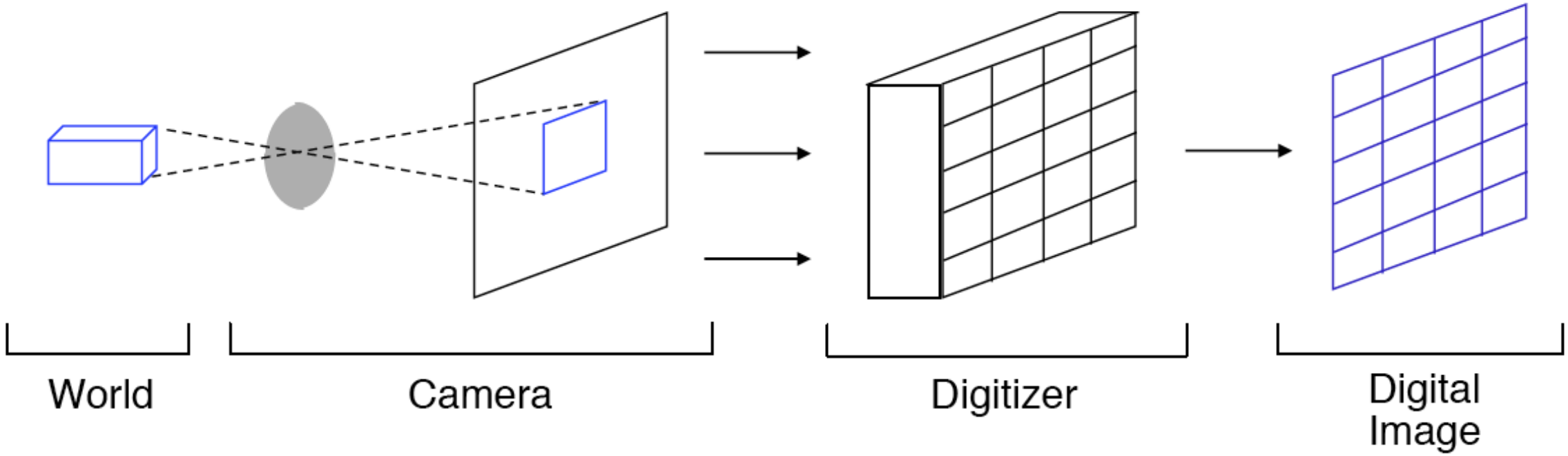


Without distortion

Barrel distortion

- Due to lens imperfections or fisheye.
- Most apparent at the edge of the image.

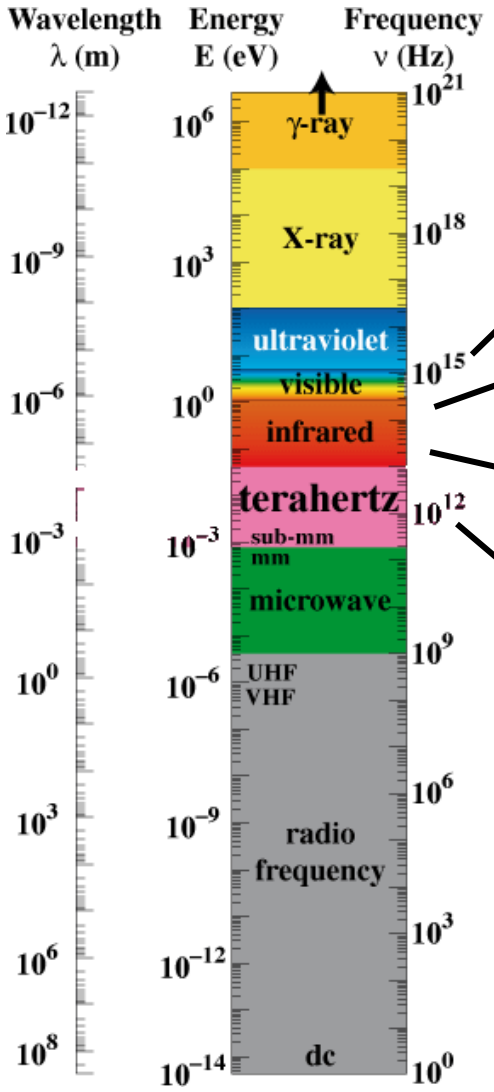
# Digital image




- Instead of film, use matrix (array) of sensors.
- *Discretize* image into pixels.
- *Quantize* light into intensity levels.

# Sensor: Camera

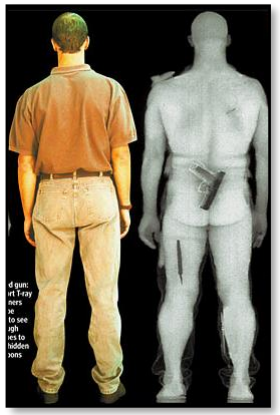
## Electromagnetic spectrum




Visible light

Near-infrared light

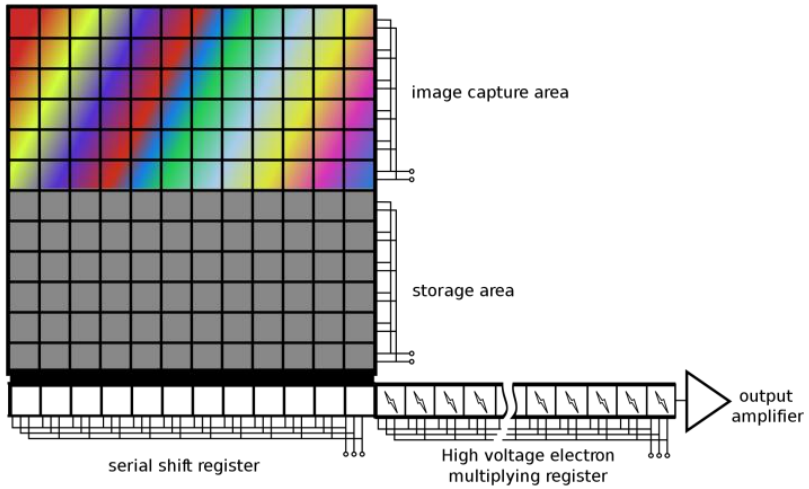
Far-infrared light



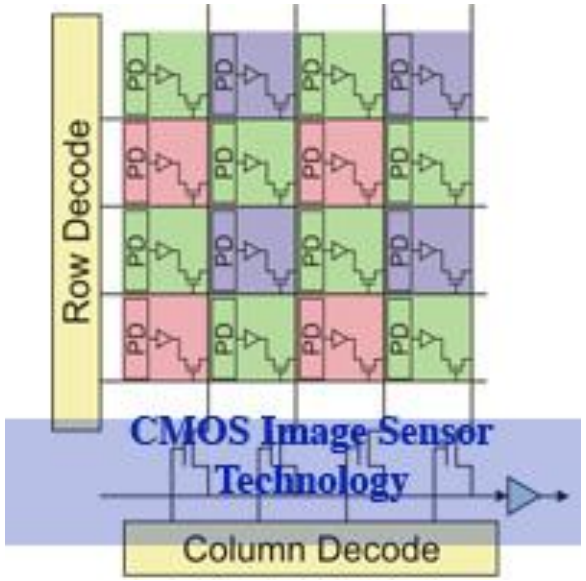
Terahertz light

# Visible light cams: CCD vs CMOS

Charge coupled device (CCD)



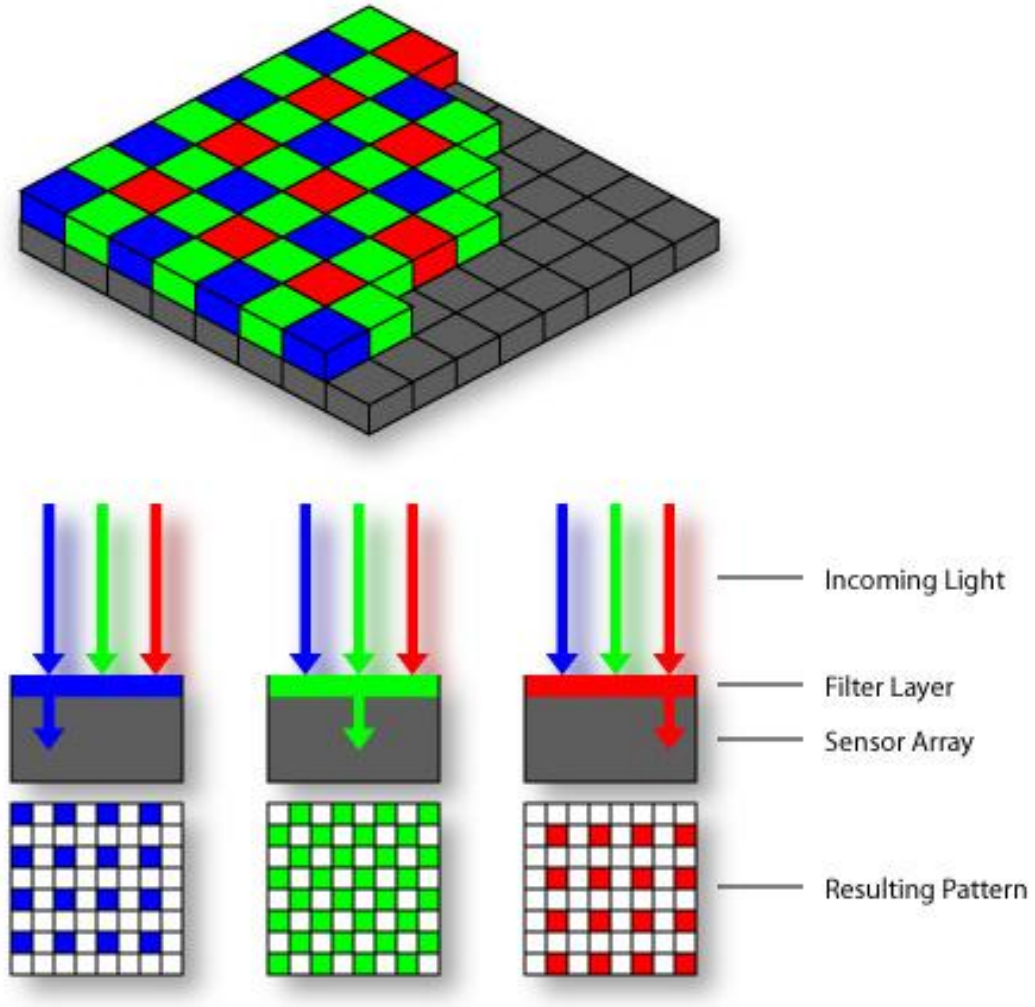

Complementary metal–oxide–semiconductor (CMOS)




- In both: Photons cause charge on each sensor „cell“.
- CCD reads out the charge (FIFO) serially and digitizes.
- CMOS performs digitization on each cell separately.
- CCD used to deliver better images, but CMOS technology has progressed.
- CMOS is cheaper to produce and is thus wide-spread.

# Color sensing in digital cameras

Bayer sensor




In classical design, we cannot read out R, G and B channel at a single pixel.

Why twice as many greens compared to blue and red?

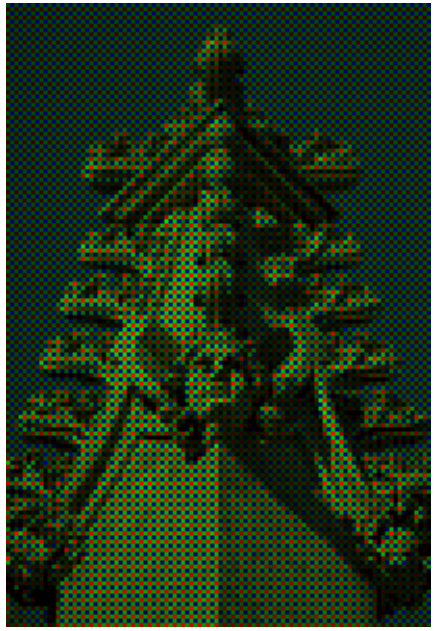
Luminance is mostly determined by the green values.

Human visual system much more sensitive to changes in intensity than in chroma (color).

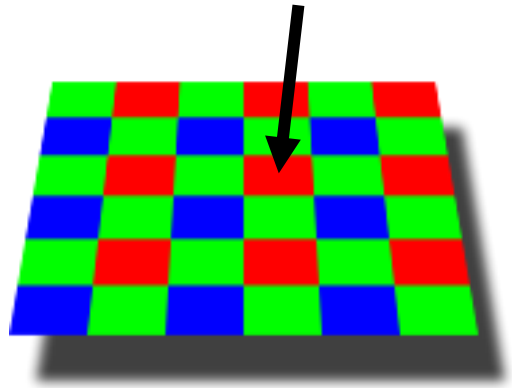
# Color sensing in digital cameras

What you see

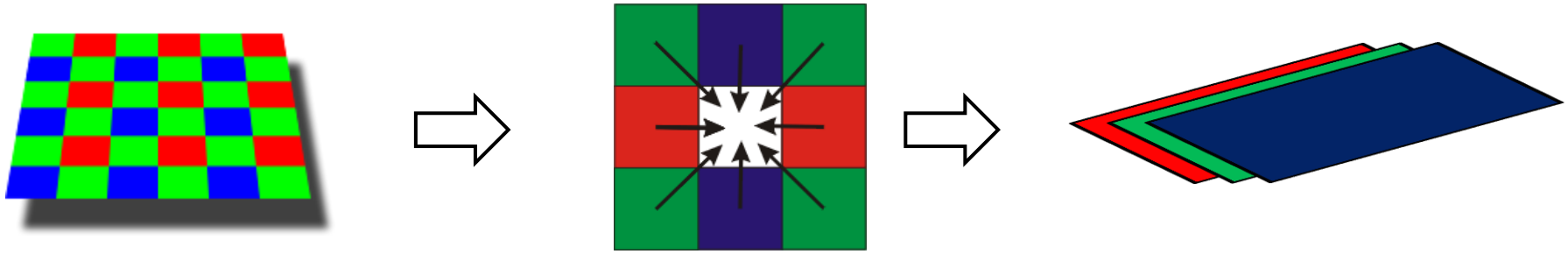
Your camera sees



*Missing green!*



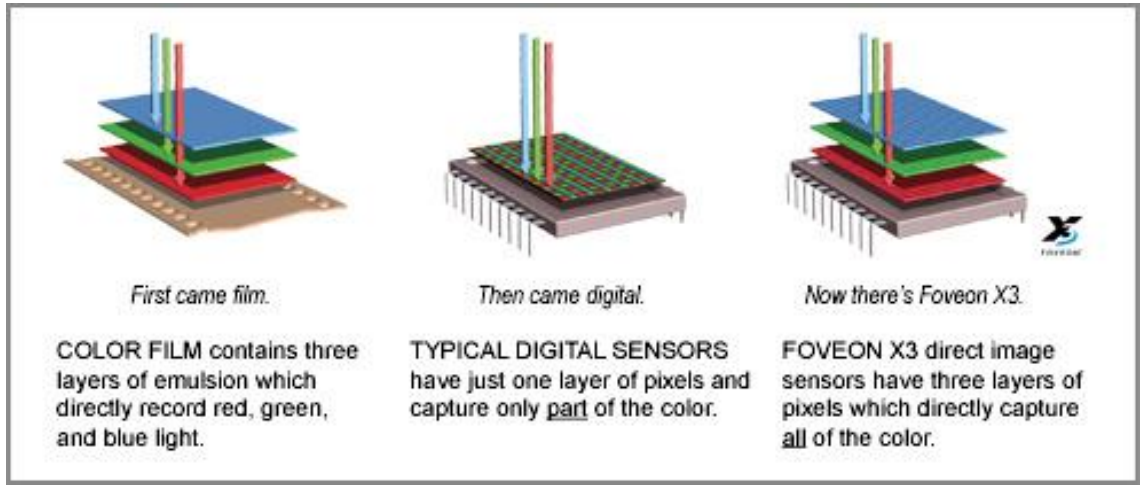
De-mosaicking: The missing color channels at a pixel need to be interpolated!



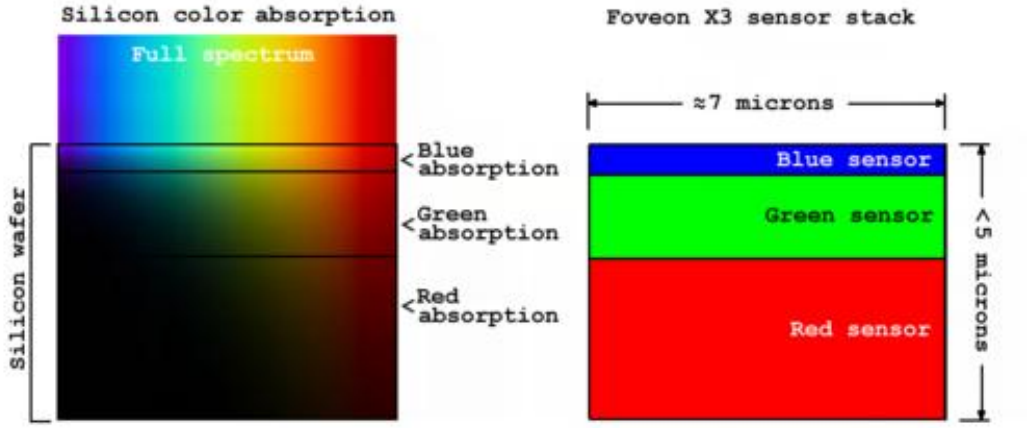
See further info on methods: Wei & Sun, JIG2022; Malvar et al., ICASSP2024

# Color sensing : Foveon X3

- CMOS-based sensor.
- Based on the fact, that red, green and blue color penetrate the silicon at different depths.



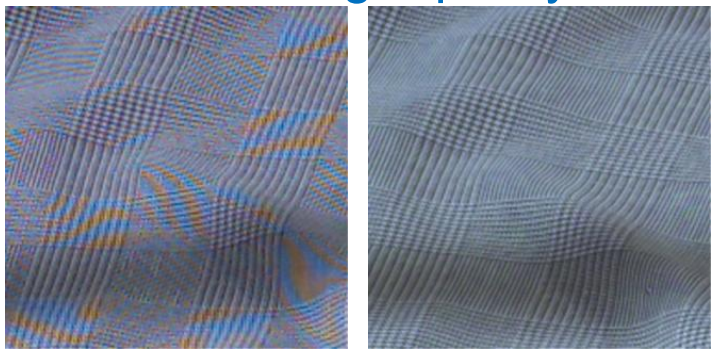

http://www.foveon.com/article.php?a=67




http://en.wikipedia.org/wiki/Foveon_X3_sensor

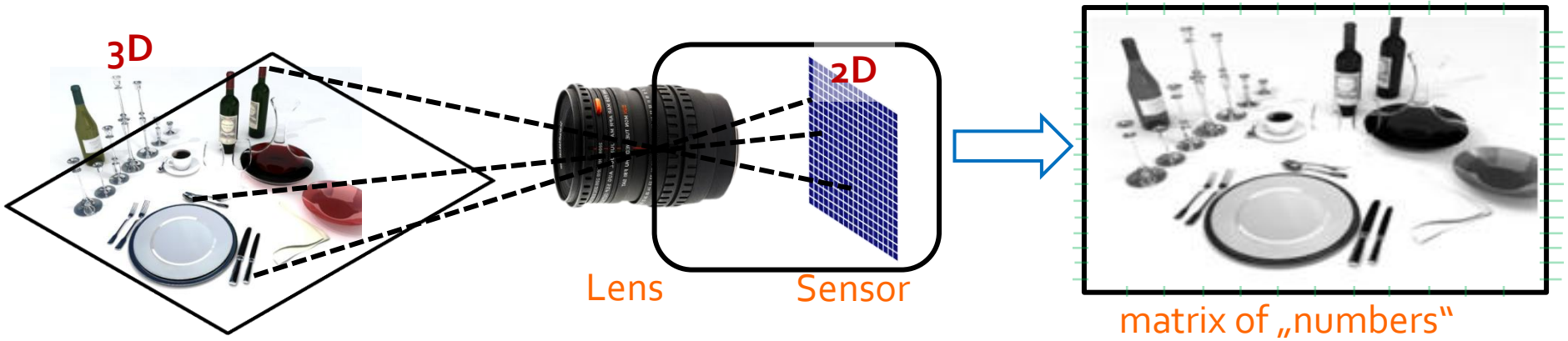
Better image quality

Bayer-like



Foveon X3

# From camera to perception



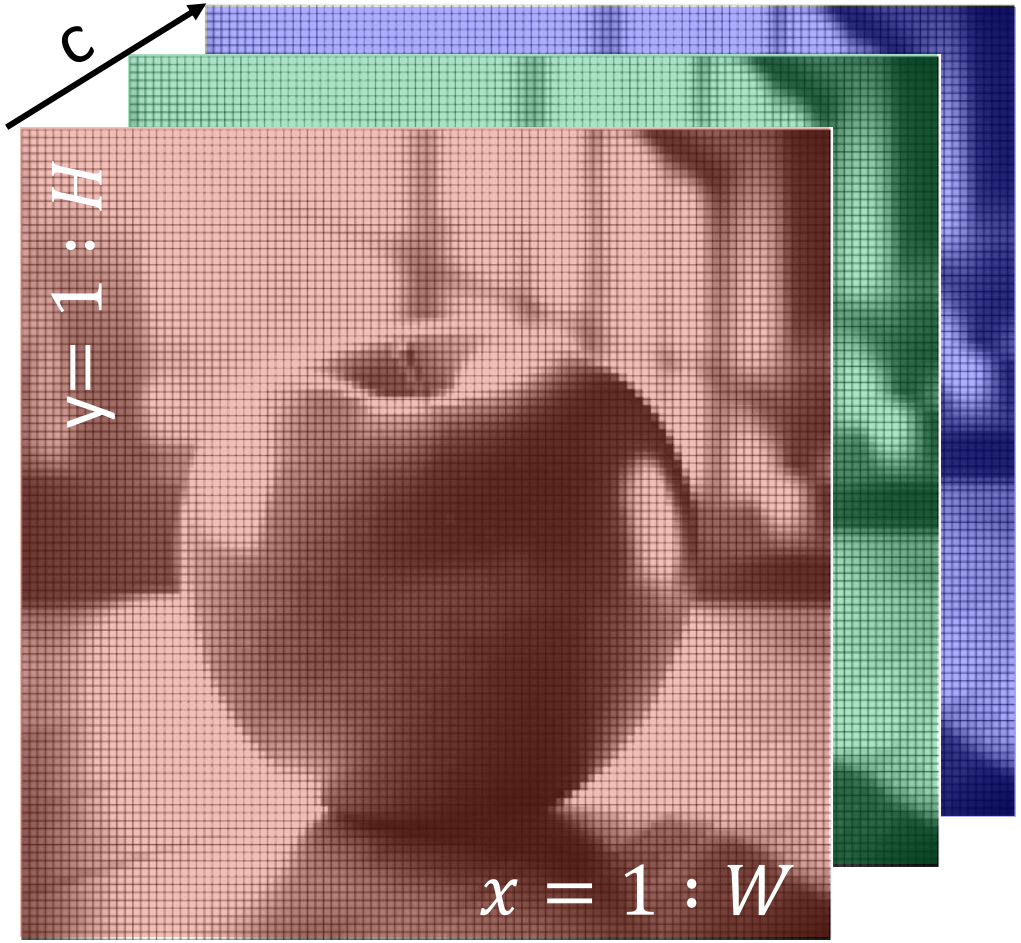

- How does a human perceive the bottles, plates, forks,..., using only brightness?
- How do we perceive depth?
- Can a computer program do that?

Machine perception

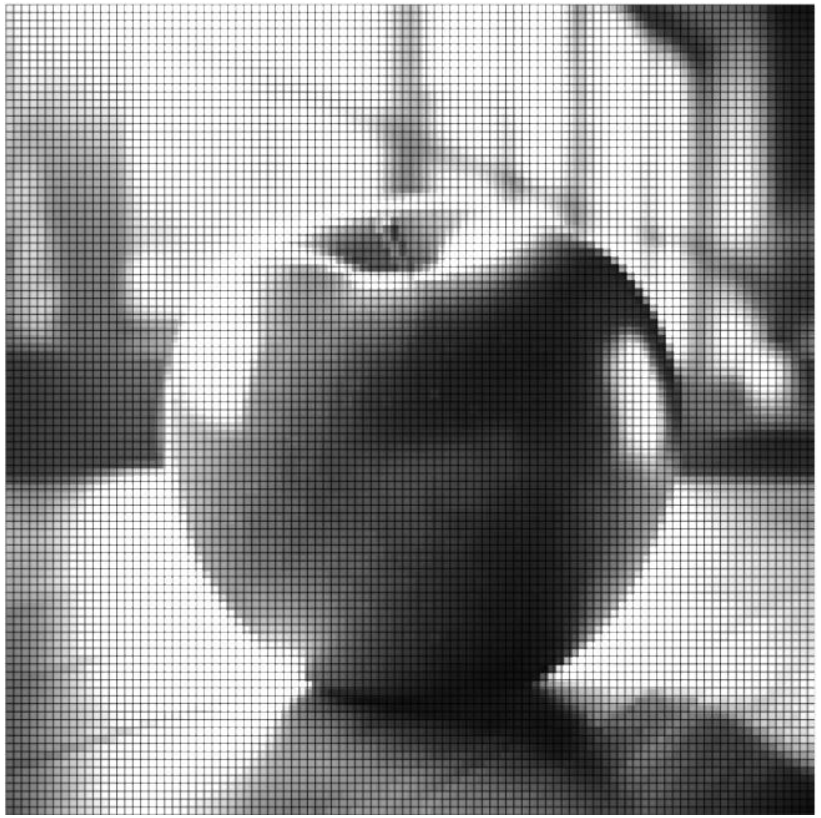
# IMAGE PROCESSING 1

# Images as functions: RGB to grayscale

- Consider a color image I(x,y,c) as a 3D matrix
- At each coordinate, $[x, y]$ we have 3 gray-scale (scalar) values



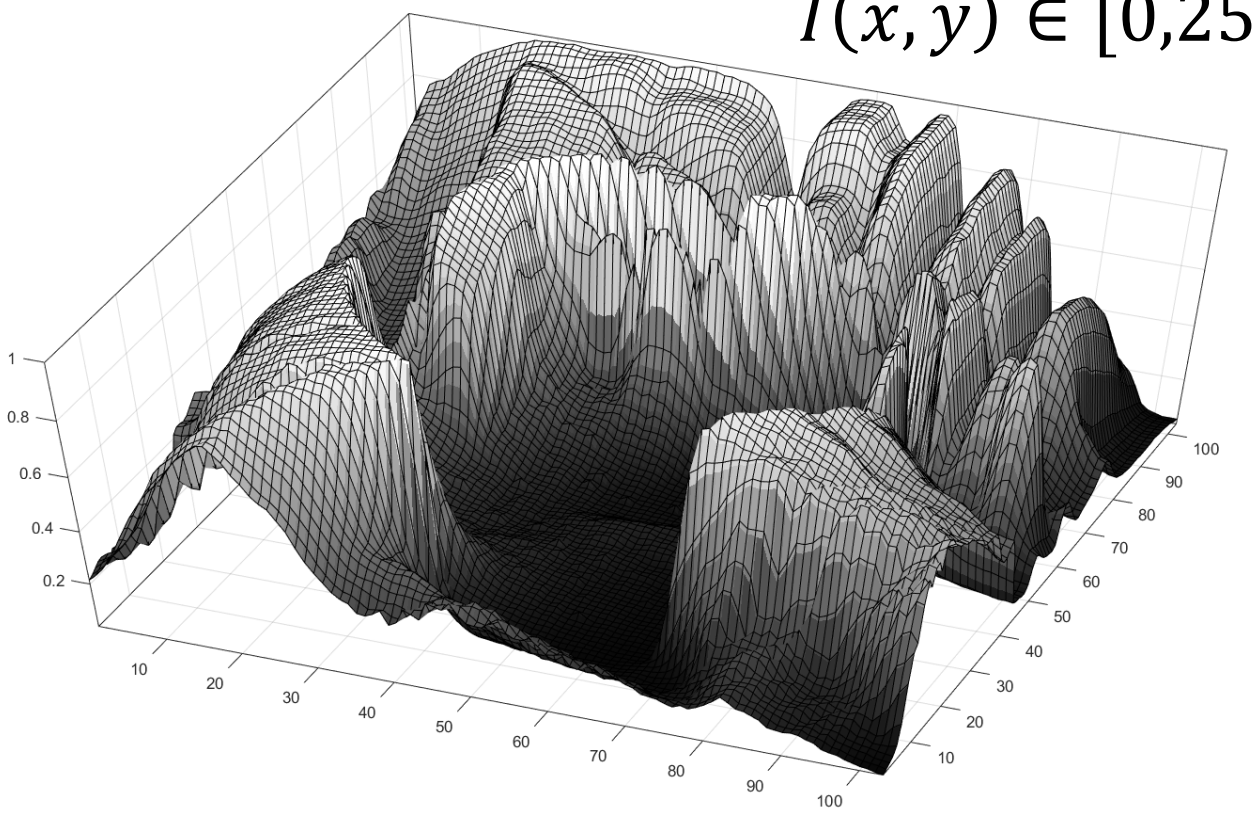

$$\frac{I(x,y,1)+I(x,y,2)+I(x,y,3)}{3}$$



# Images as functions: Grayscale images

- Consider a grayscale image I(x,y) as a matrix of just gray-scale values
- At each coordinate, $[x, y]$ we have a gray-scale value (e.g., 0-255)

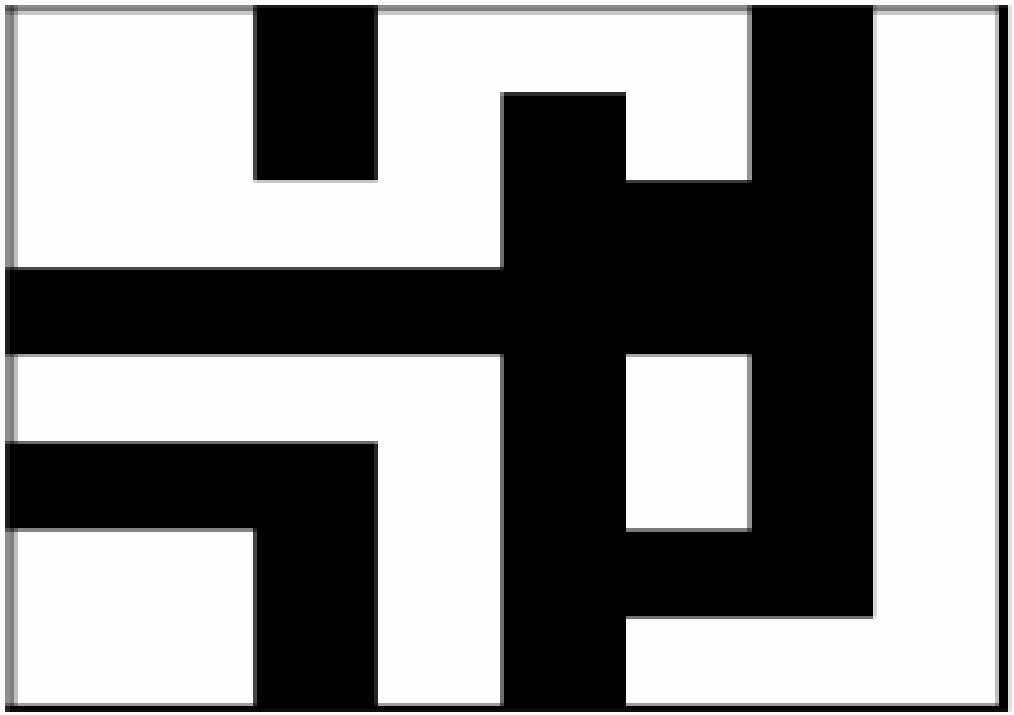

$I(x, y) \in [0,255]$



# Binary images

- Only two possible gray levels
- Foreground vs. background

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |



# Usage: Machine vision, OCR, etc.

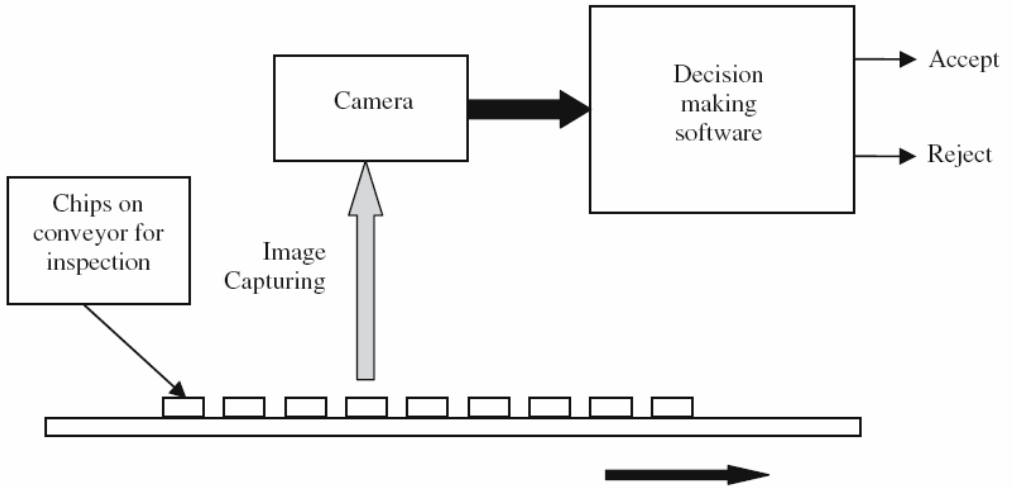



Fig. 3 Schematic diagram of marking inspection setup at Texas Instruments

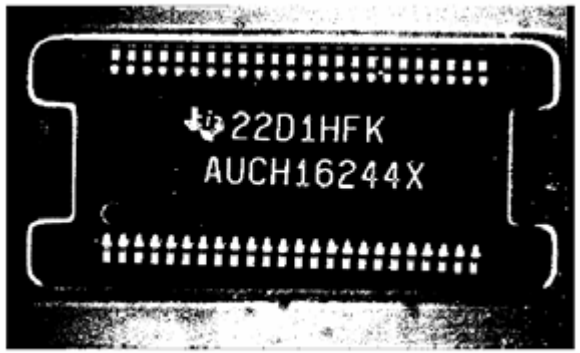
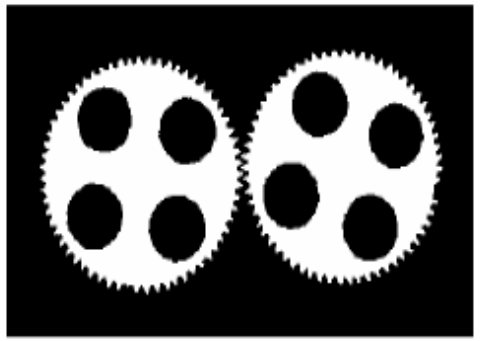
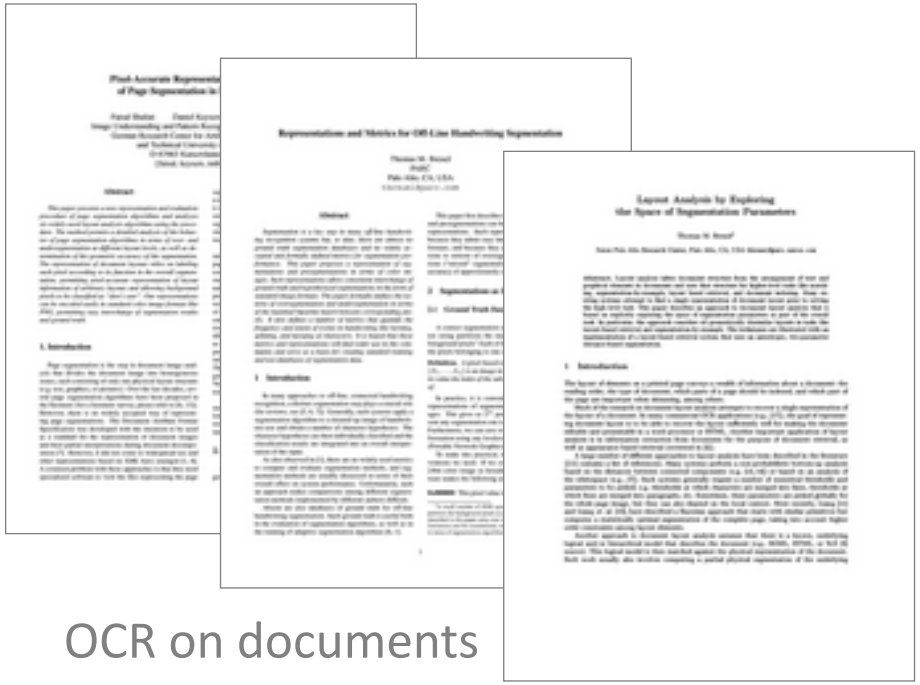



Fig. 7 Binarized image

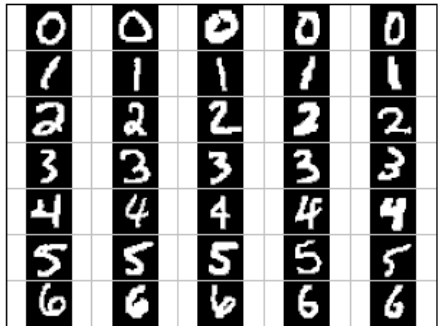


R. Nagarajan et al. “A real time marking inspection scheme for semiconductor industries“, 2006



OCR on documents

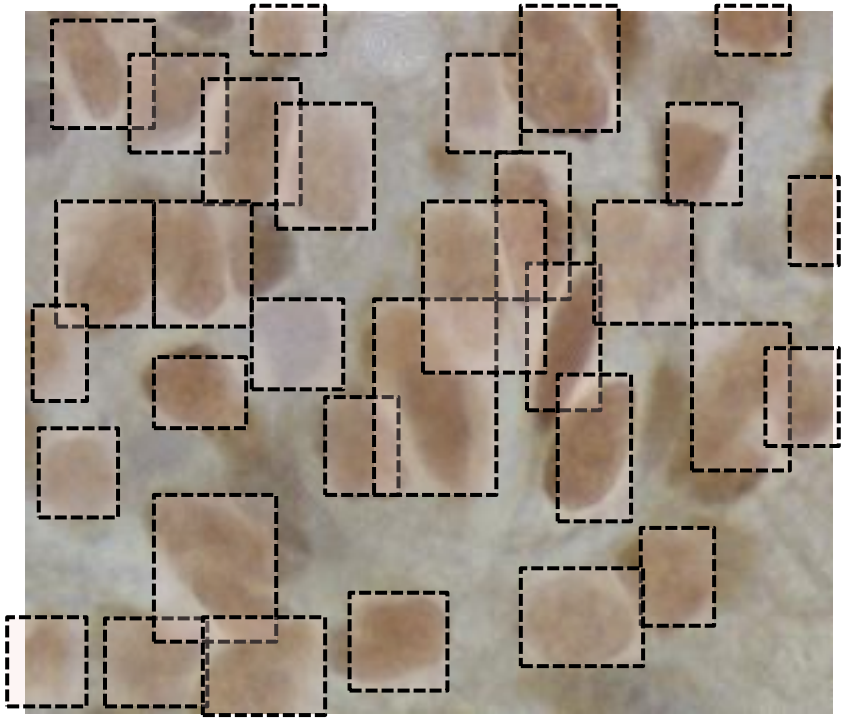
Hand written numbers



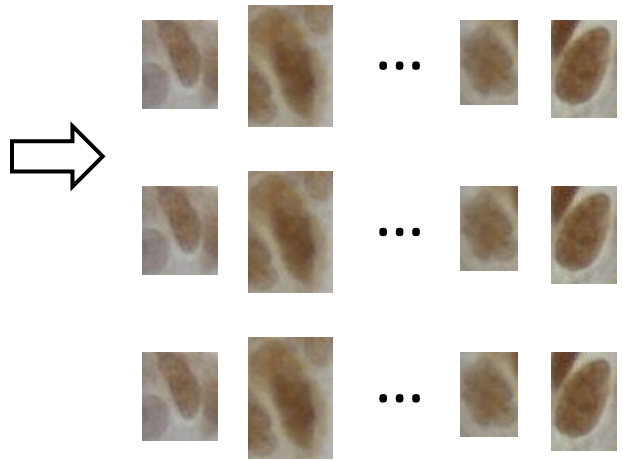
Source: Bastian Leibe

# Use case: Count the “round” cells

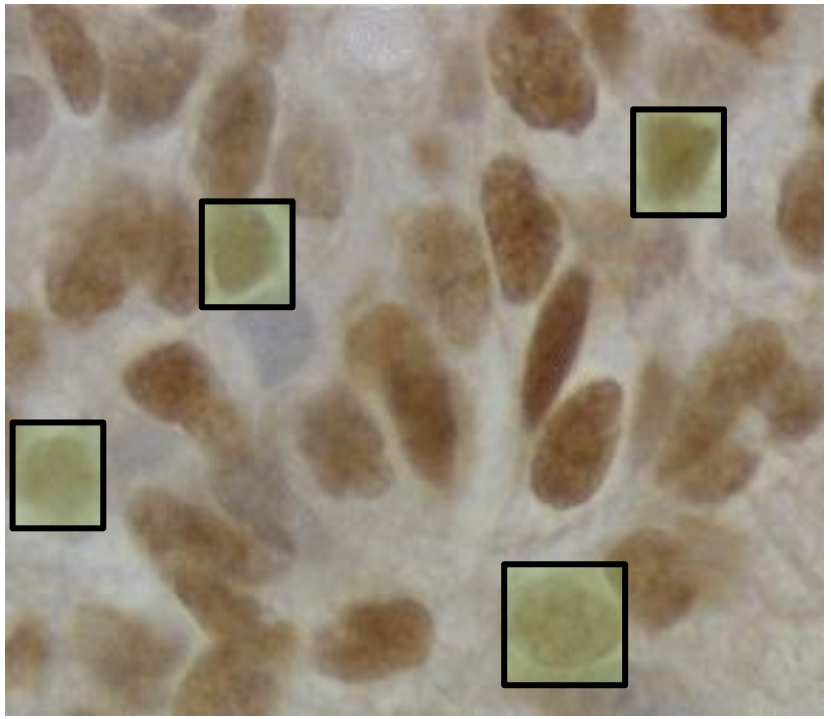
Generate hypotheses



Classify each region into a “round” and “not round”



Keep “round” regions

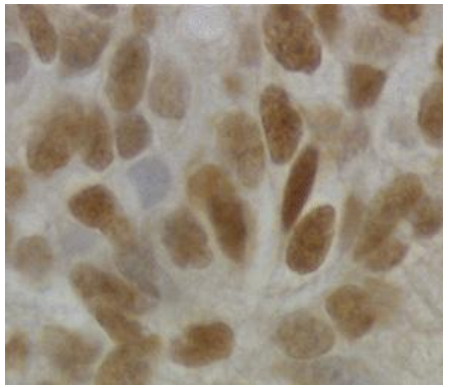


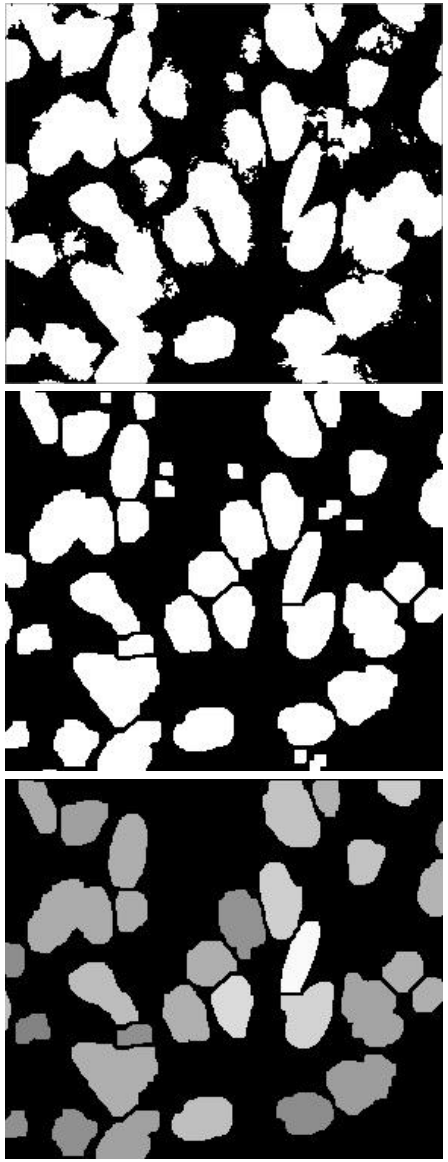
Localize, Describe, Classify

# The “Localize” block: Sequence of processing steps

- Convert gray image to a binary image
  - Thresholding
- Clean binary image
  - Morphologic filtering
- Extract individual regions
  - Connected components

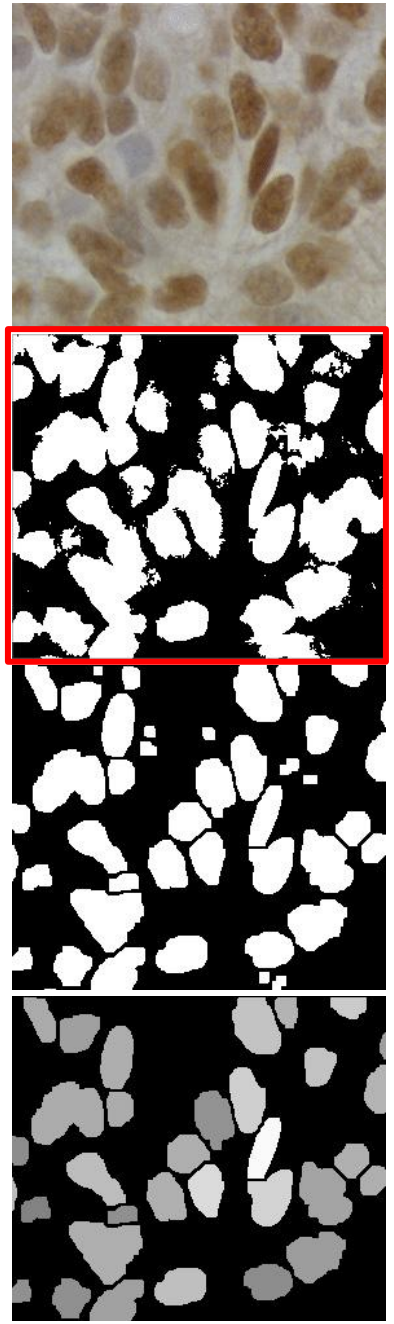
*… then describe each localized region and classify*





Machine perception

# IMAGE THRESHOLDING



# Thresholding

- Transform an image into a Binary Mask
- Various approaches
  - Apply a single threshold

$$F_T[i,j] = \begin{cases} 1, \text{if } F[i,j] \le T \\ 0, \text{otherwise} \end{cases}$$

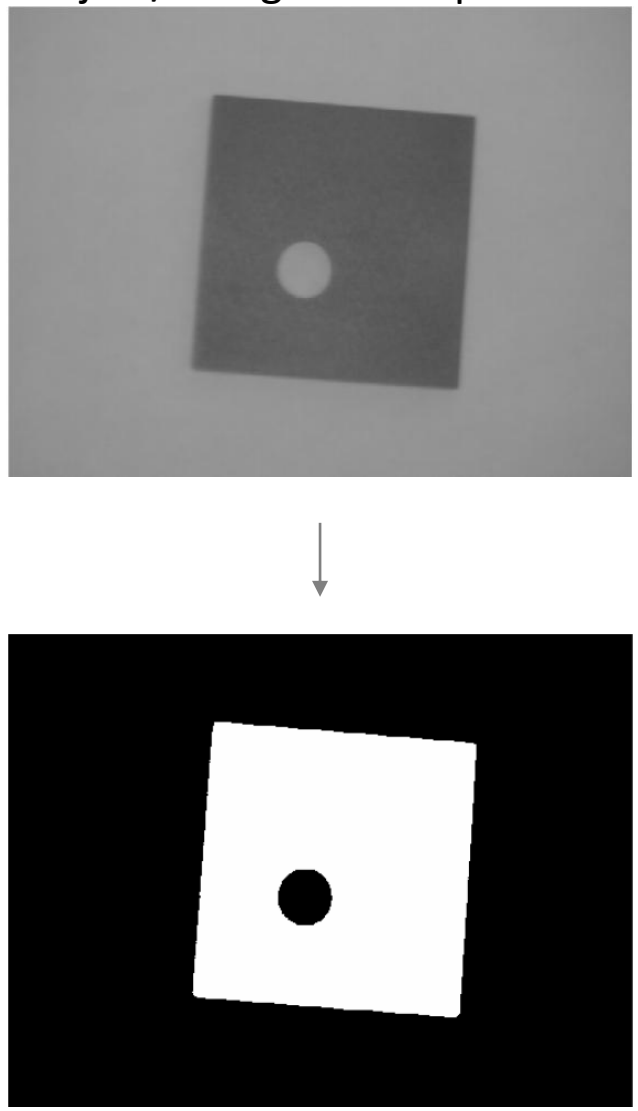
  - Apply two thresholds

$$F_T[i,j] = \begin{cases} 1, & \text{if } T_1 \le F[i,j] \le T_2 \\ 0, & \text{otherwise} \end{cases}$$

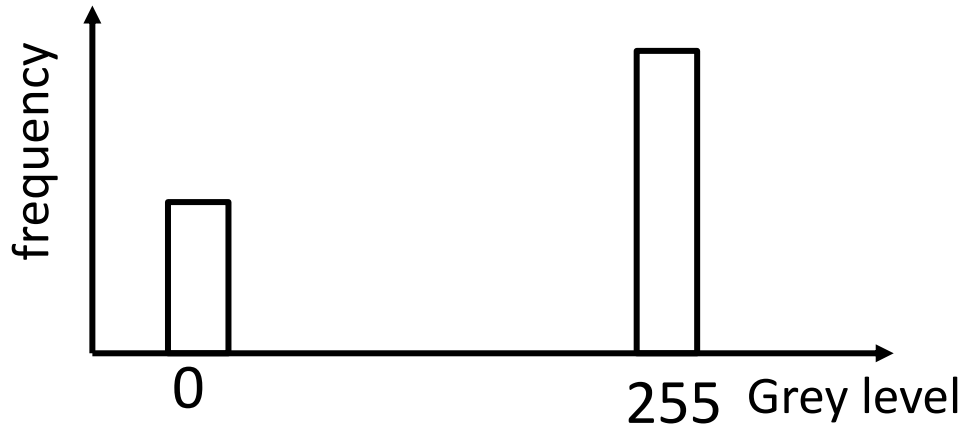
  - A general view: apply a classifier

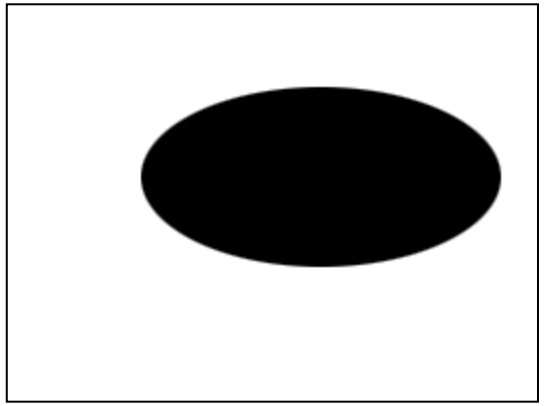
$$F_T[i,j] = \begin{cases} 1, & \text{if } F[i,j] \in Z \\ 0, & \text{otherwise} \end{cases}$$

Object/background separation

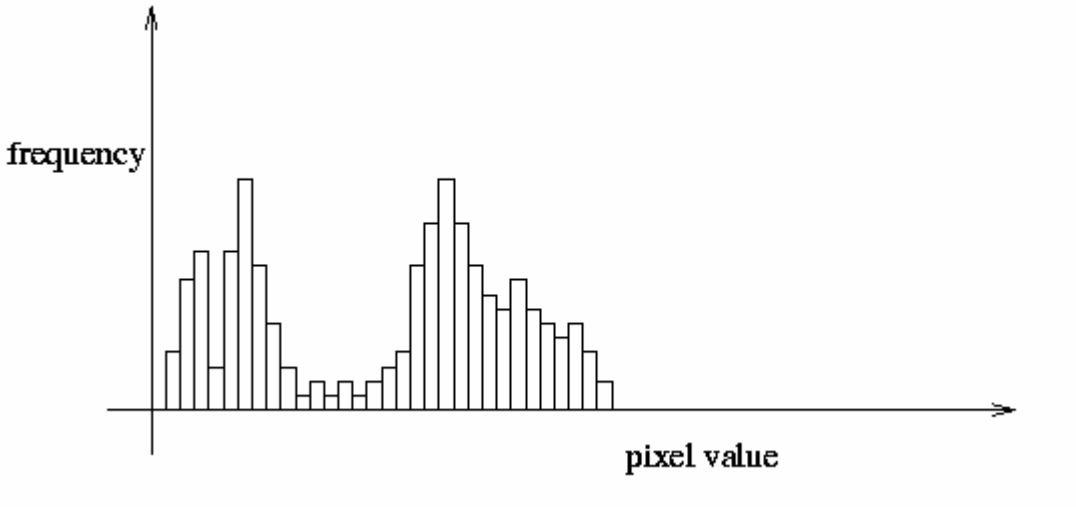


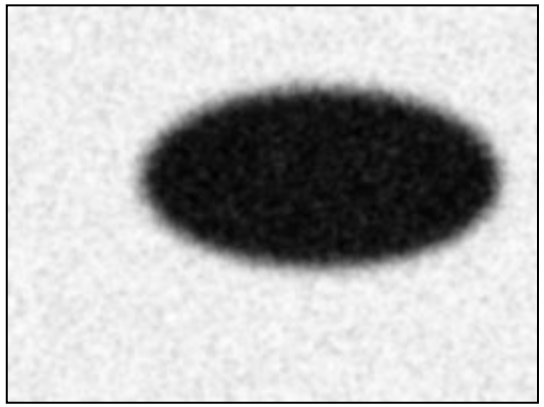
# A simple example: Bimodal histogram



Ideal case:
bright object on
dark background.

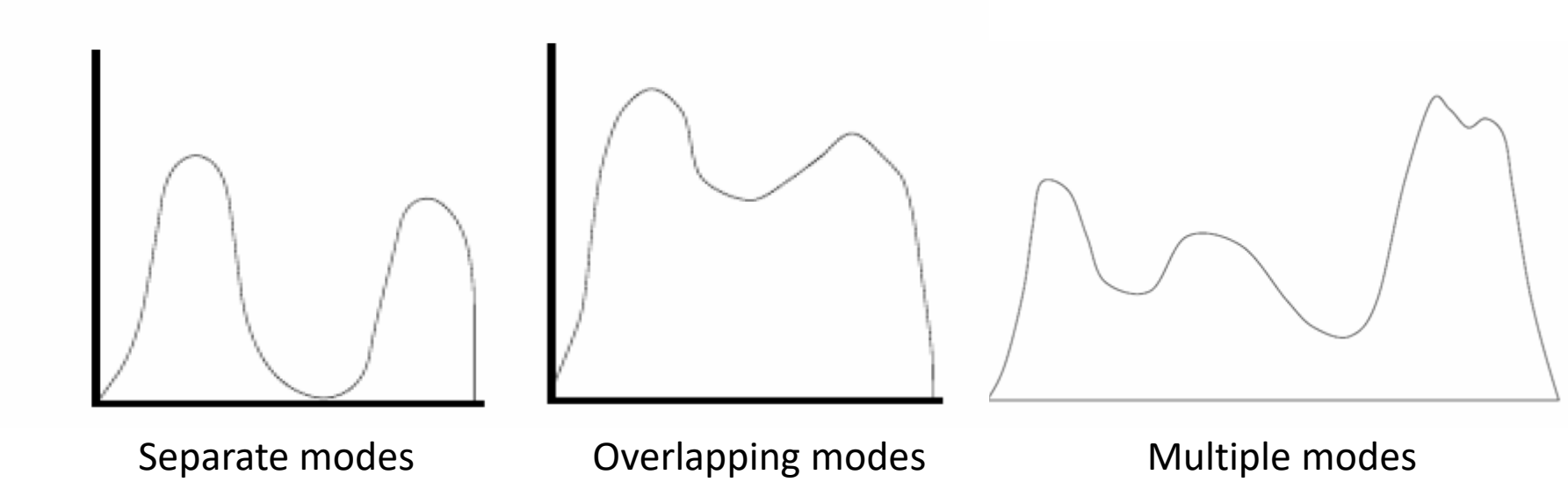


A more realistic noisy
image.

# A not so simple example...

- What to do here?




- Generally thresholding is a difficult problem
  - Domain knowledge helps a great deal.
  - E.g., the portion on letters on a page.
  - E.g., size of the structure we want to detect...

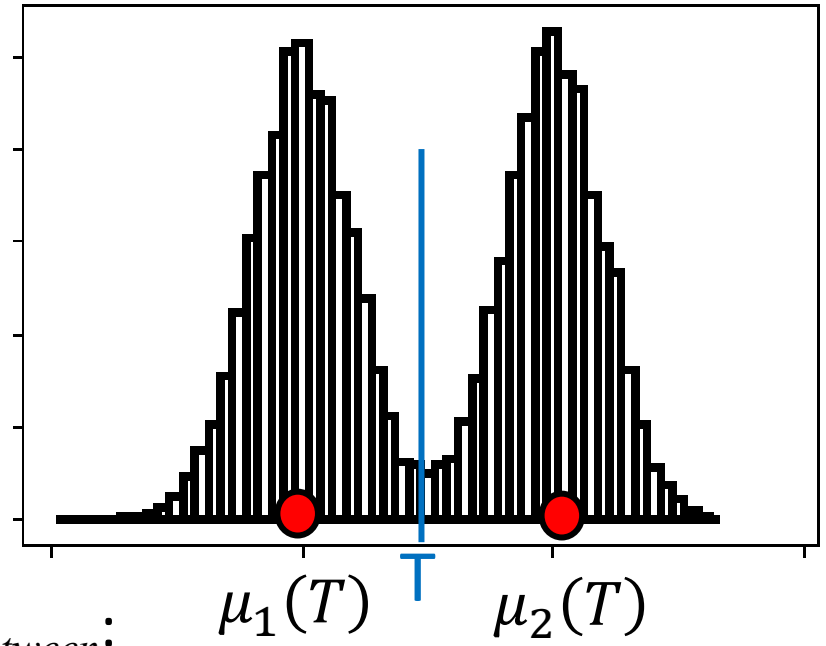
Source: Shapiro & Stockman

# Global binarization [Otsu ’79]

- Find a threshold *T*, that minimizes intensity variances within classes separated by *T*:

$$\sigma^2_{within}(T) = n_1(T)\sigma_1^2(T) + n_2(T)\sigma_2^2(T)$$

$$n_1(T) = \left|\{I_{(x,y)} < T\}\right|, n_2(T) = \left|\{I_{(x,y)} \geq T\}\right|$$




- This equals to maximization of between class variance $\sigma_{between}$:

$$\sigma^2_{between}(T) = \sigma^2 - \sigma^2_{within}(T)$$
$$= n_1(T)n_2(T)[\mu_1(T) - \mu_2(T)]^2$$

$\mu_1(T)$ … *mean value* 1
$\mu_2(T)$ … *mean value* 2

*Otsu, N (1979), "A threshold selection method from gray-level histograms", IEEE SMC*

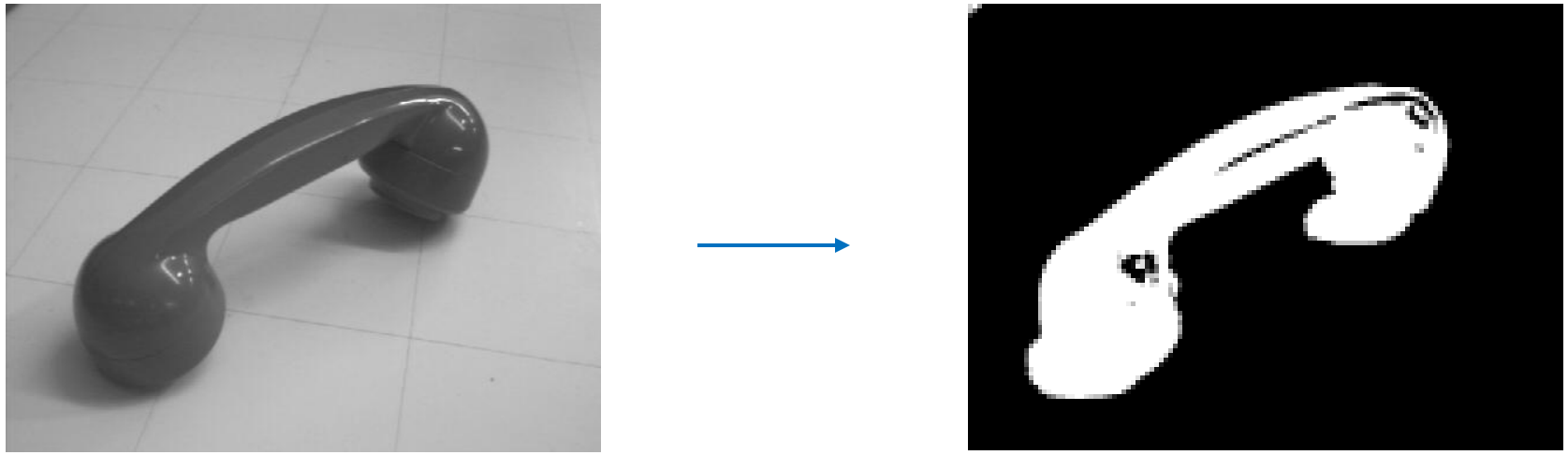
Source: Bastian Leibe

# Otsu's Algorithm

For threshold value *T*

1. Separate the pixels into two groups by intensity threshold *T*
2. For each group get an average intensity and calculate $\sigma^2_{between}$ according to the equation on the previous slide.

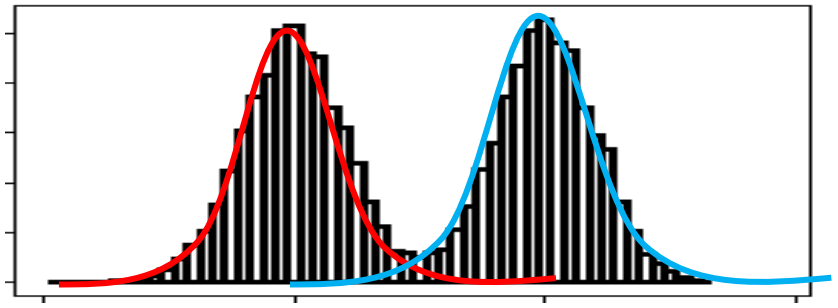
Select the T*, that maximizes the variance: $T^* = \arg\max_T [\sigma^2_{between}(T)]$



Used in several thousand modern algorithms in particular in medical imaging

# State-of-the-art: Generalization of Otsu (CVPR2020)

- "Recently", Otsu's method revisited:
- Formulate the problem as fitting 2 Gaussians to the histogram with priors on mixture weights and variances (Bayesian view)
- Efficiently computed by a single pass through the histogram (like Otsu)
- Outperforms all single-pass algorithms and all deep learning algorithms on the text binarization benchmark



(a) Input Image (b) True Binarization (c) Oracle Binarization

(d) Otsu's Method (e) MET (f) GHT

Barron, J.T., A Generalization of Otsu's Method and Minimum Error Thresholding, CVPR2020 ; link to video

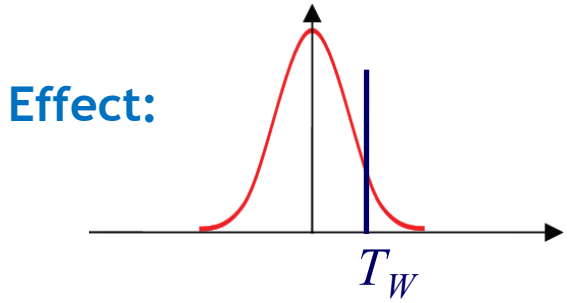
# Local binarization [Niblack'86]

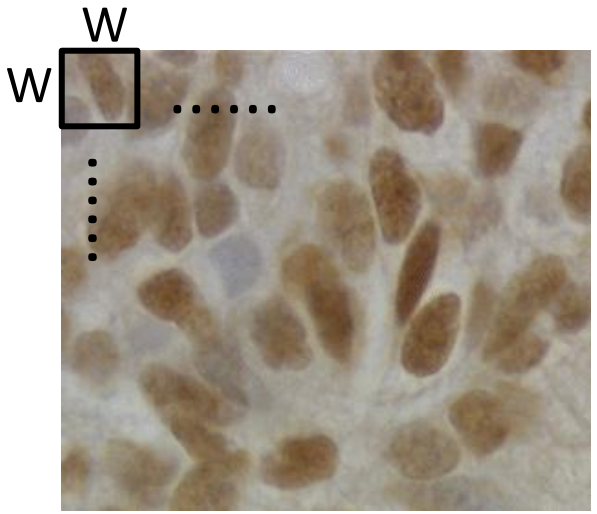
- Estimate a local threshold in neighborhood W:

$$T_W = \mu_W + k \cdot \sigma_W$$

with $k \in [-1,1]$ set by user.

- Calculate the threshold separately for each pixel.

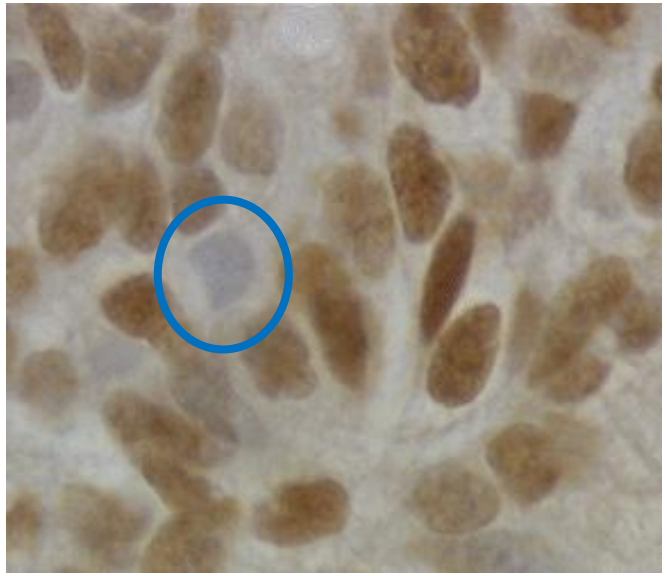


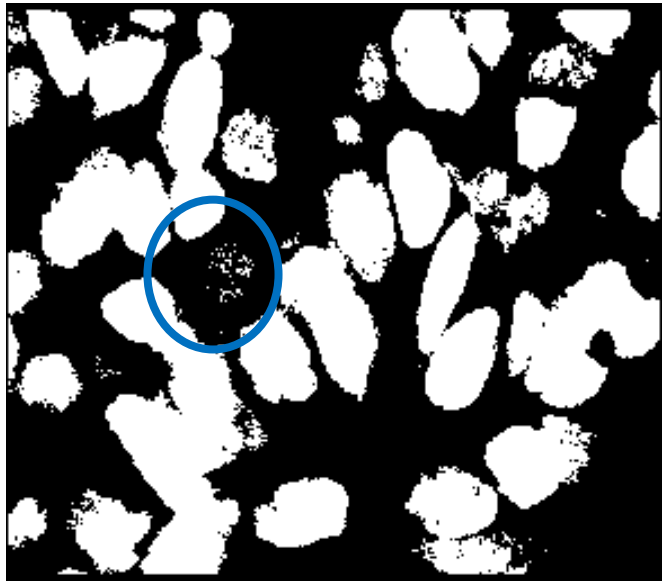

*Niblack, W (1986), An introduction to Digital Image Processing, Prentice-Hall*

Source: Bastian Leibe

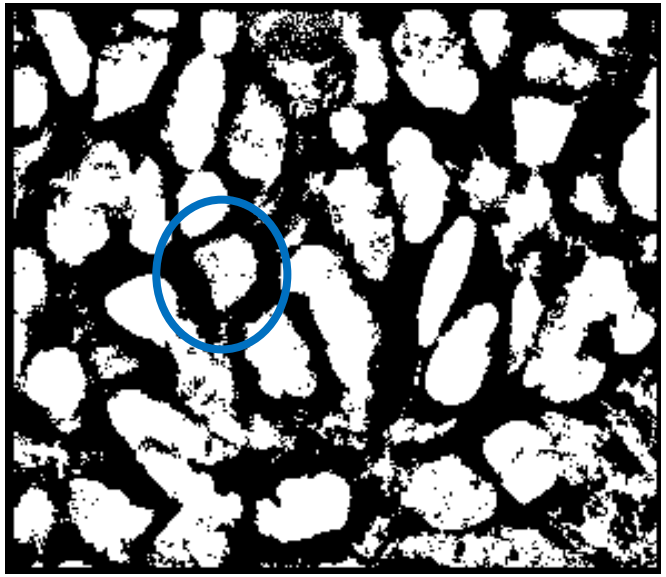
# Examples of thresholding



Original



Global (Otsu)



Local (Niblack)

# Additional improvements

- The shade in documents is often smooth...

⇒*Try to model it by a polynomial!*

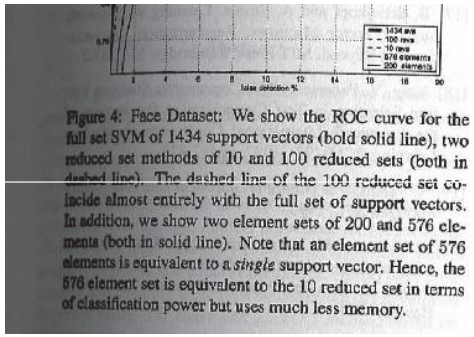
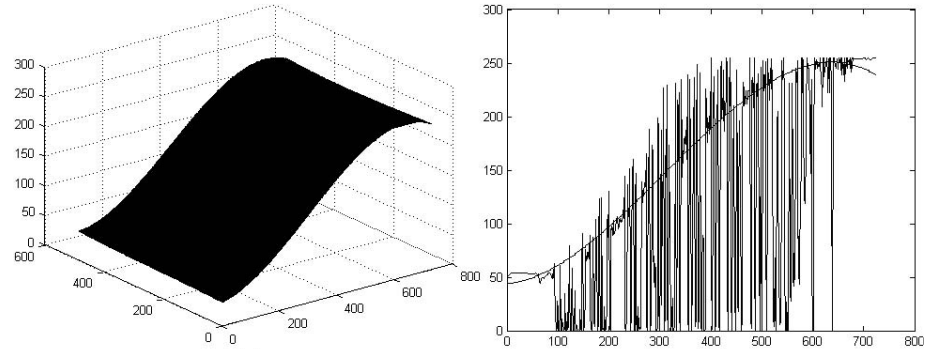


Figure 4: Face Dataset: We show the ROC curve for the full set SVM of 1434 support vectors (bold solid line), two reduced set methods of 10 and 100 reduced sets (both in dashed line). The dashed line of the 100 reduced set coincide almost entirely with the full set of support vectors. In addition, we show two element sets of 200 and 576 elements (both in solid line). Note that an element set of 576 elements is equivalent to a *single* support vector. Hence, the 576 element set is equivalent to the 10 reduced set in terms of classification power but uses much less memory.

Original




Fitted surface

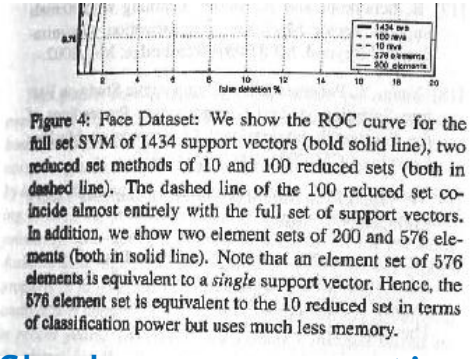


Figure 4: Face Dataset: We show the ROC curve for the full set SVM of 1434 support vectors (bold solid line), two reduced set methods of 10 and 100 reduced sets (both in dashed line). The dashed line of the 100 reduced set coincide almost entirely with the full set of support vectors. In addition, we show two element sets of 200 and 576 elements (both in solid line). Note that an element set of 576 elements is equivalent to a *single* support vector. Hence, the 576 element set is equivalent to the 10 reduced set in terms of classification power but uses much less memory.

Shadow compensation

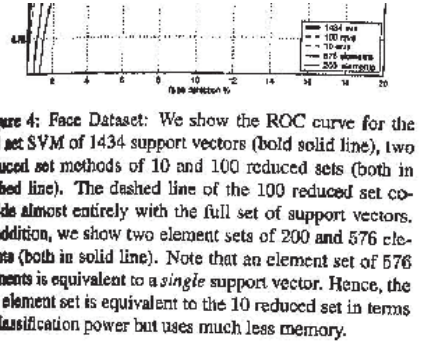
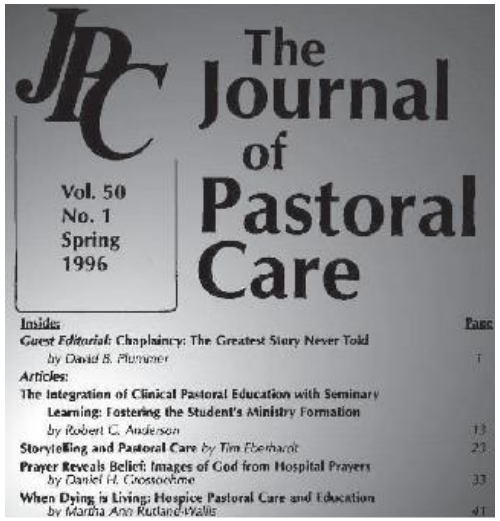


Figure 4: Face Dataset: We show the ROC curve for the full set SVM of 1434 support vectors (bold solid line), two reduced set methods of 10 and 100 reduced sets (both in dashed line). The dashed line of the 100 reduced set coincide almost entirely with the full set of support vectors. In addition, we show two element sets of 200 and 576 elements (both in solid line). Note that an element set of 576 elements is equivalent to a *single* support vector. Hence, the 576 element set is equivalent to the 10 reduced set in terms of classification power but uses much less memory.

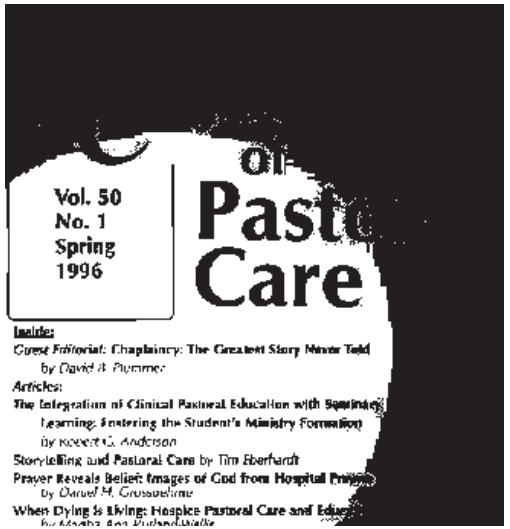
Binarized result

# Comparison of results

Original image

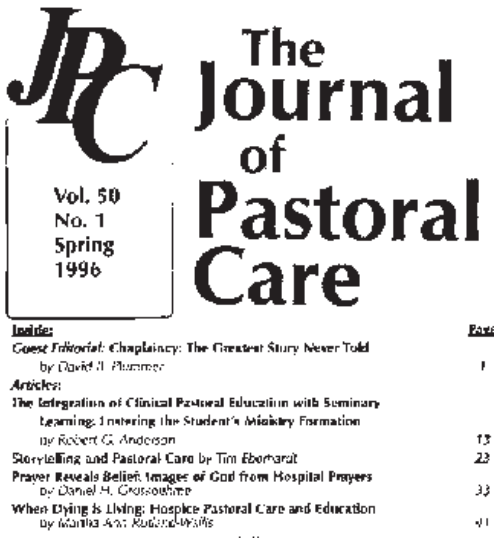



Global (Otsu)



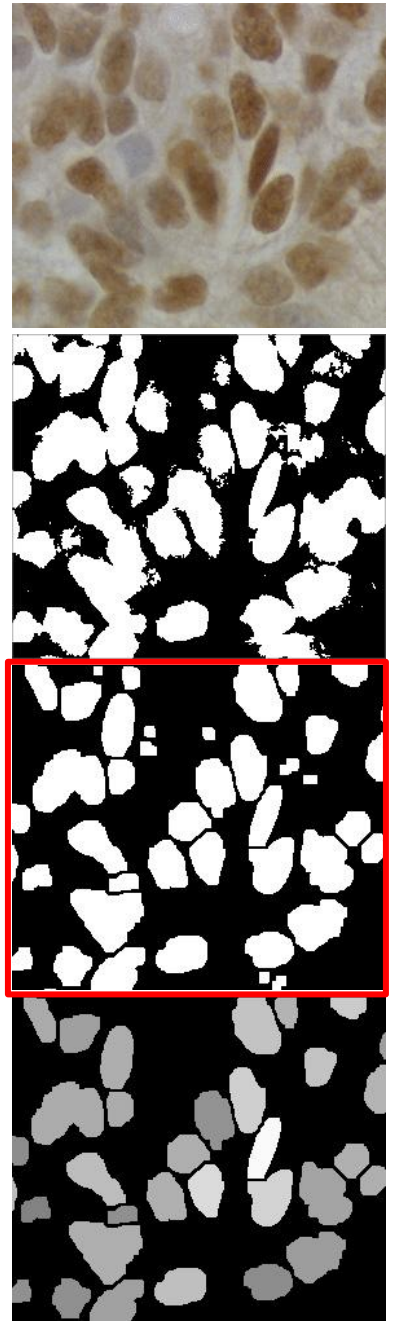

Local (Sauvola)


Polynomial + global



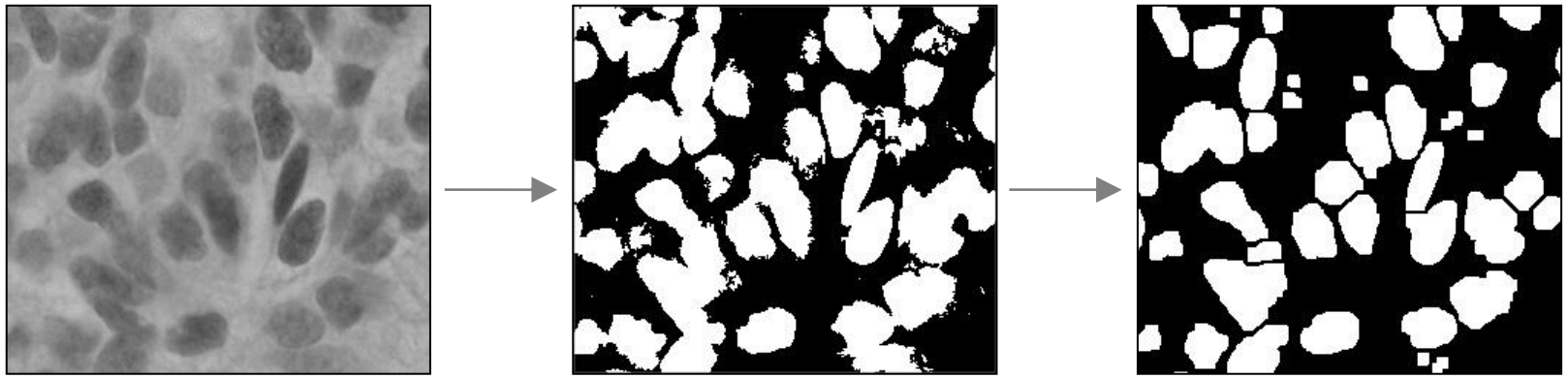
Machine perception

# CLEANING THE IMAGE



# Cleaning the binary image

- Thresholded image still includes noise

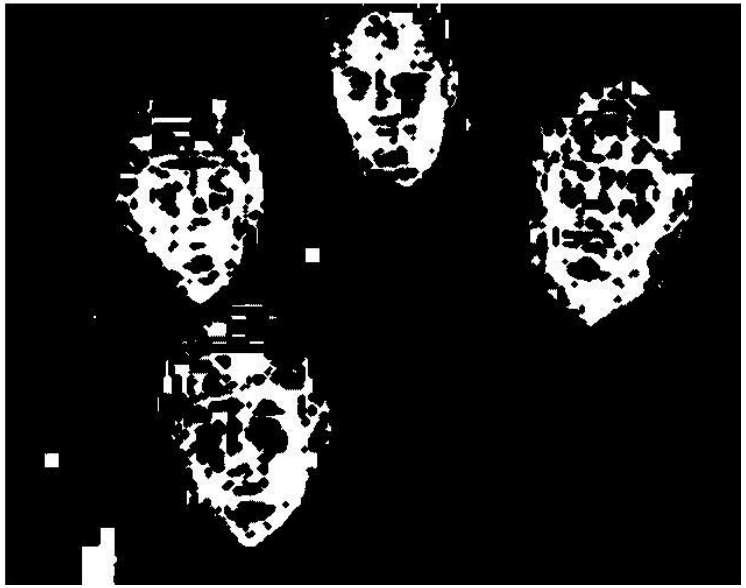


- Require post-processing to remove artefacts
- Morphological operators
  - Remove isolated points and small structures
  - Fill holes

Source: Bastian Leibe

# Dilation: A sneak peak preview

- Dilate the regions of „white“ pixels
- Increases the size of the structures
- Fills holes in regions



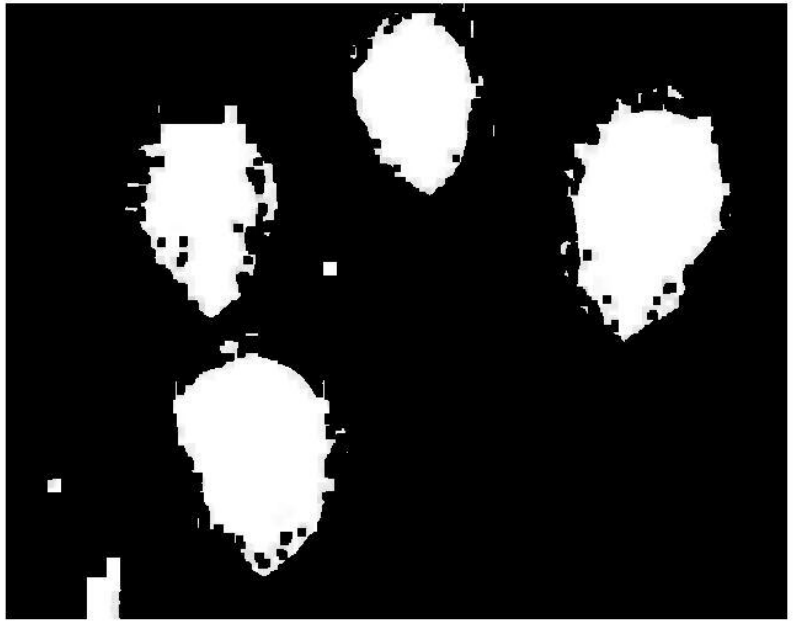
Before dilation

After dilation

# Erosion: A sneak peak preview

- Erode the  regions of „white“ pixels
- Reduce the size of structures
- Remove bridges, branches, noise

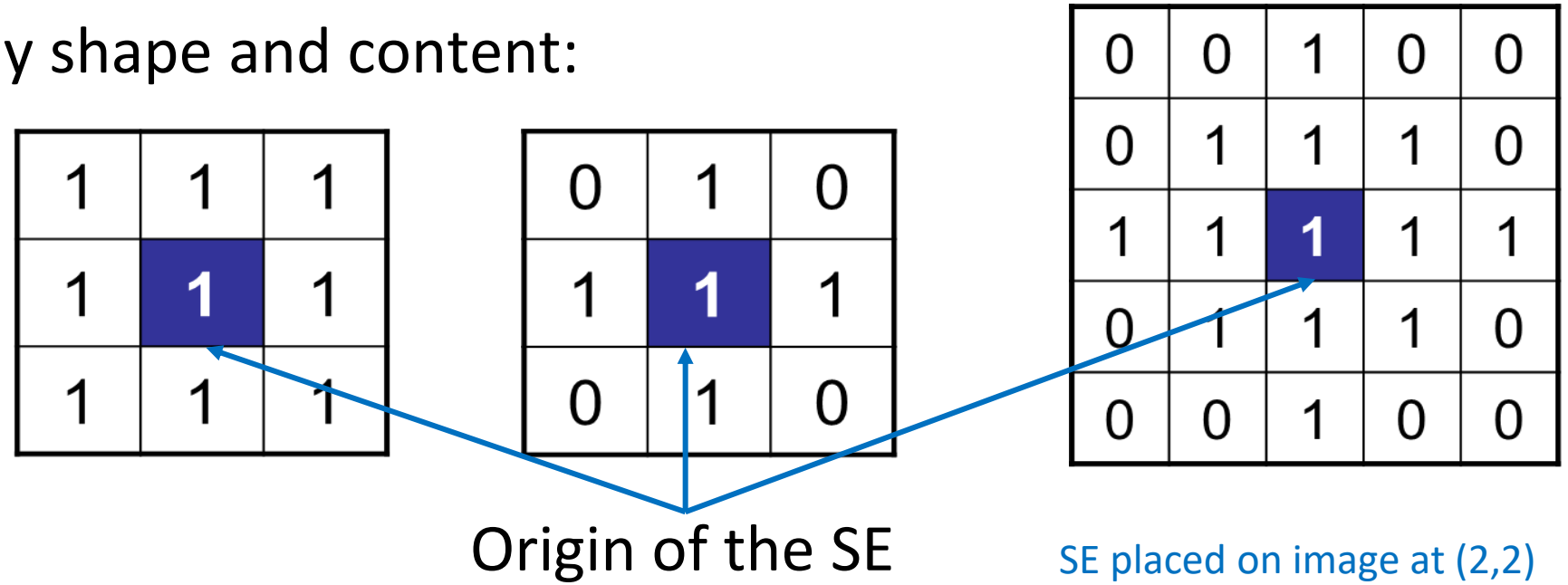
Before erosion



After erosion

# Central to morphology: Structuring element (SE)

- Can be any shape and content:




- Fit: All “1” pixels in SE cover “1” pixels in the image.
- Hit: At least one of “1” pixels in SE cover “1” pixels in the image.

SE placed on image at (2,2)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 … |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 3 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| … | | | | | | | |

# Fitting & Hitting

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | B | 1 | 1 | 1 | 0 | C | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | A | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |

Structuring Element

Fit / Hit?

A:

B:

C:

Fit : All “1” elements in SE cover 1

Hit: Any “1” element in SE cover 1

# Erosion

- Erosion of image $f$ by structuring element $s$ is given by $g = f \ominus s$.
- The structuring element s is positioned with its origin at $(x, y)$ and the new pixel value is determined using the rule:

$$g(x,y) = \begin{cases} 1 \text{ if } s \text{ fits } f \\ 0 \text{ otherwise} \end{cases}$$

$s$

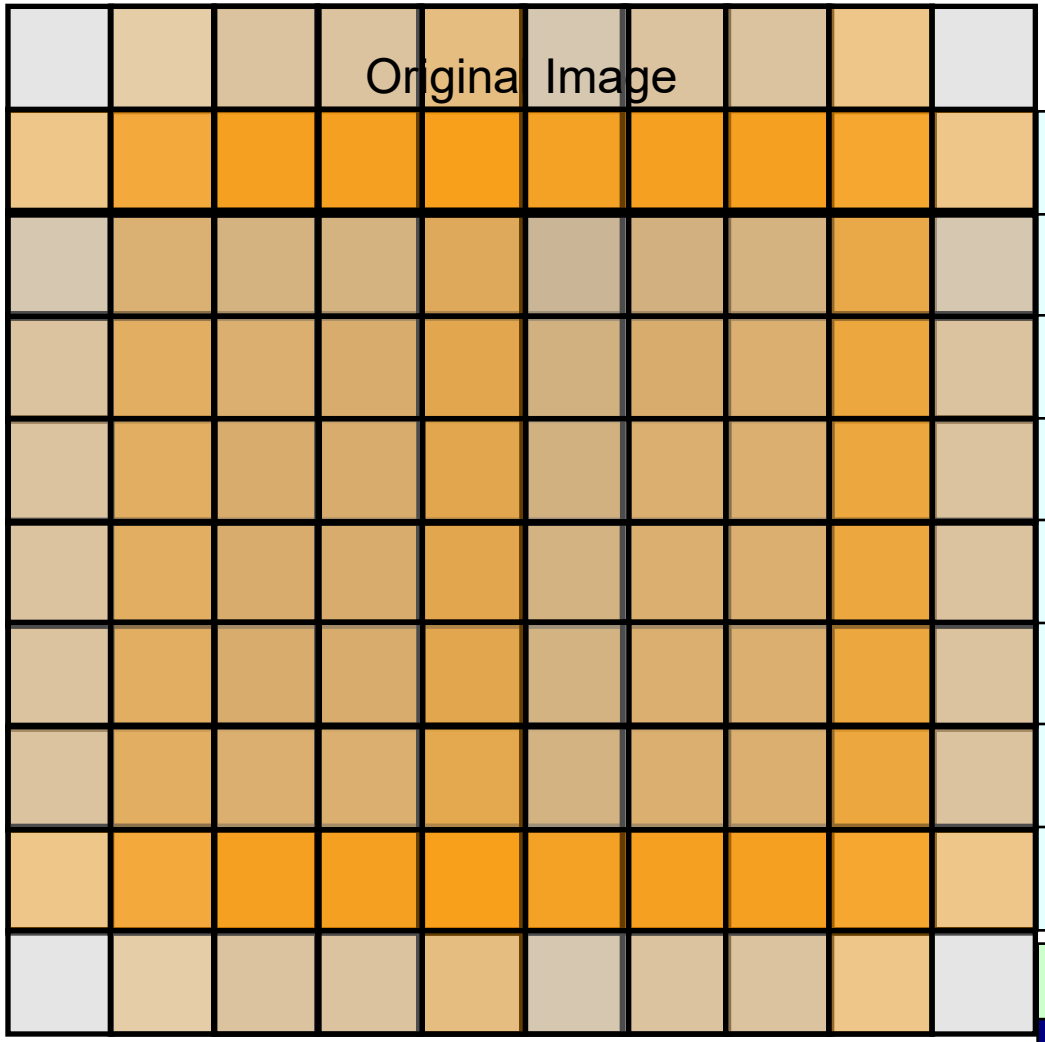
| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |

SE placed on image at (2,2)

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 … |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 3 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 4 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 5 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| … | | | | | | | |

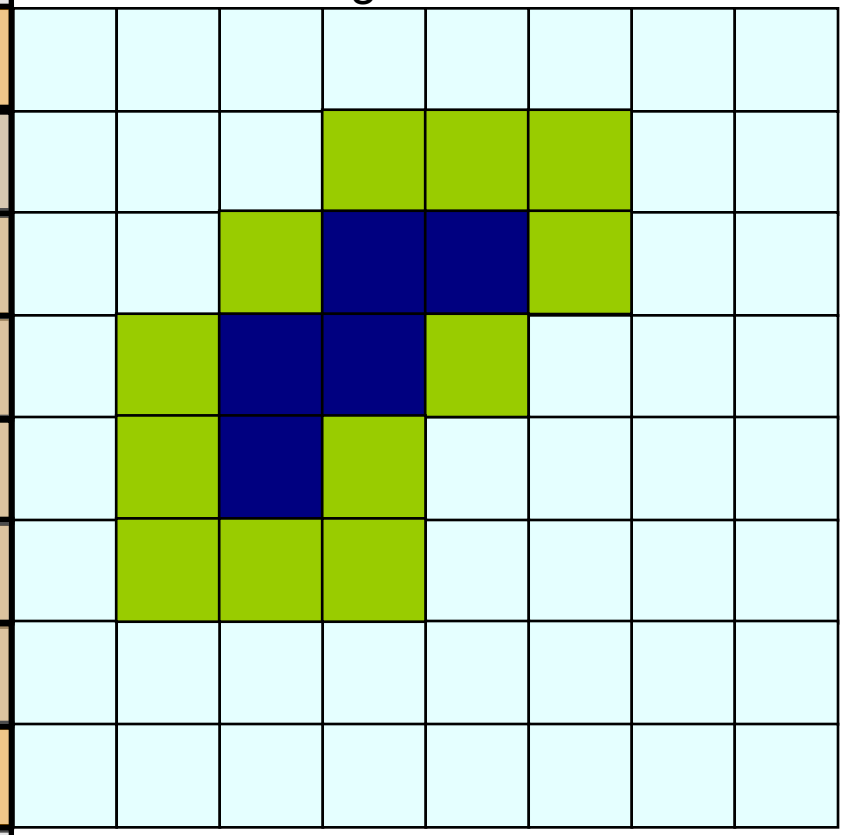
Fit: All “1” pixels in SE cover “1” pixels in the image.

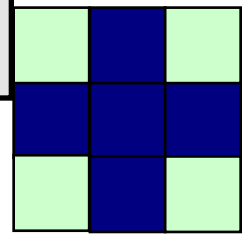
# Erosion Example




Fit :All 1 in SE covered in image

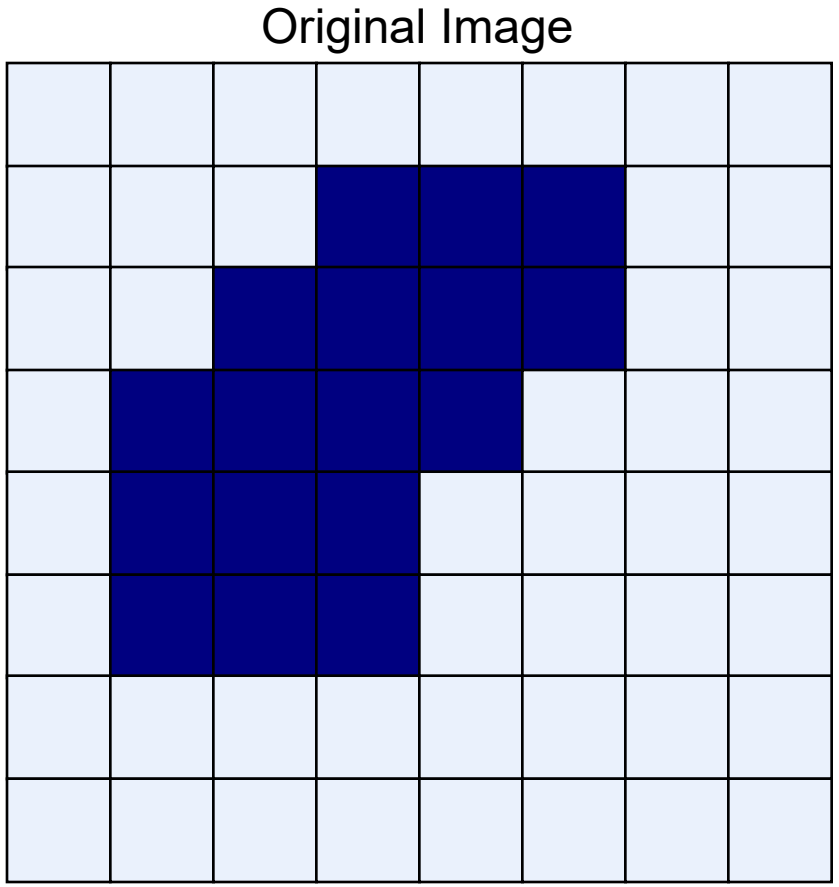
Processed Image With Eroded Pixels

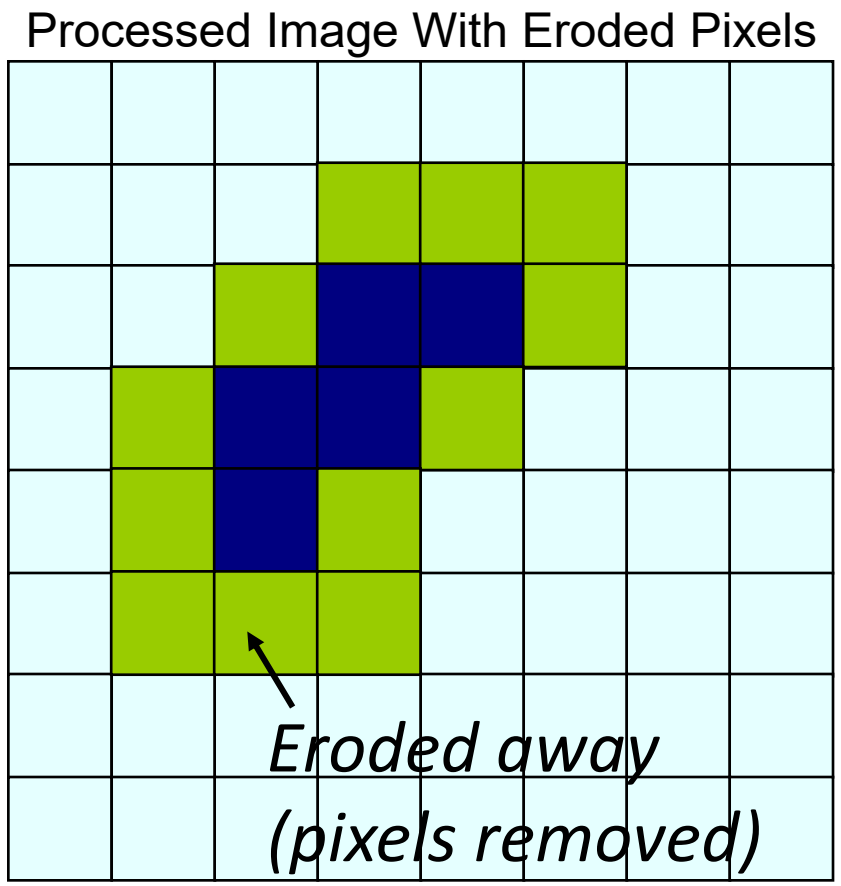


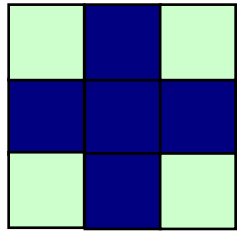


Structuring Element

# Erosion Example



Structuring Element

Fit :All 1 in SE covered in image

# Dilation

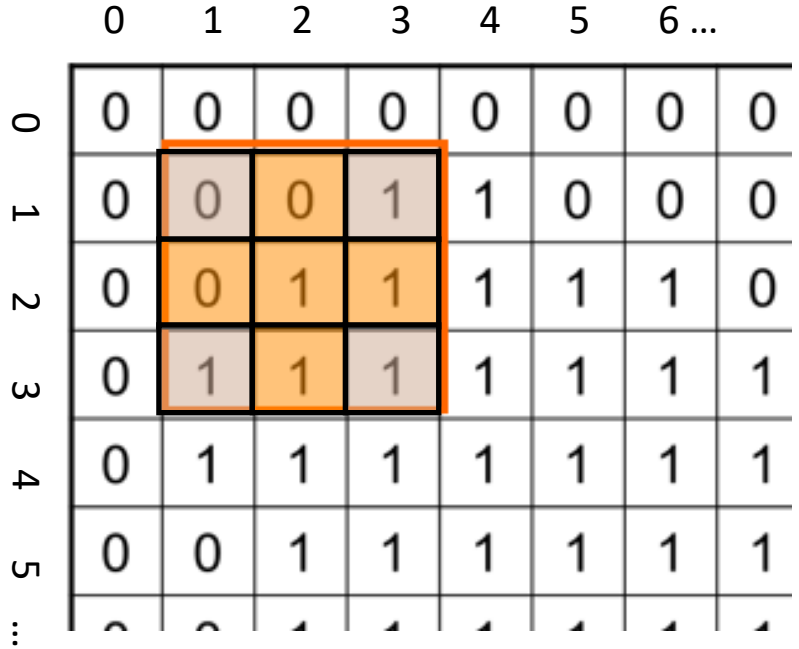
- Dilation of image $f$ by structuring element $s$ is given by $g = f \oplus s$.
- The structuring element s is positioned with its origin at $(x, y)$ and the new pixel value is determined using the rule:

$$g(x,y) = \begin{cases} 1 \text{ if } s \text{ hits } f \\ 0 \text{ otherwise} \end{cases}$$

$s$

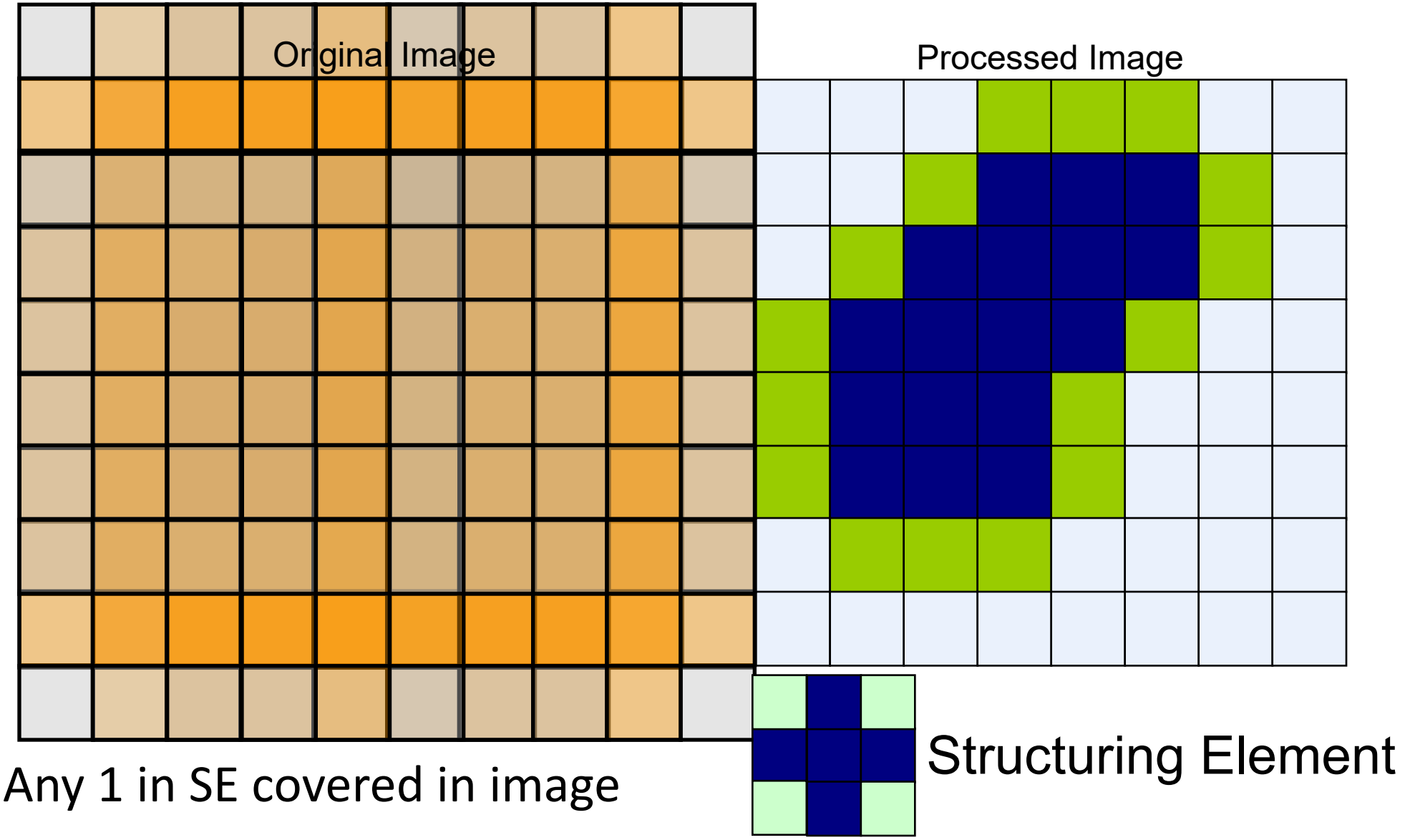
| | | |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 1 | 1 |
| 0 | 1 | 0 |

SE placed on image at (2,2)



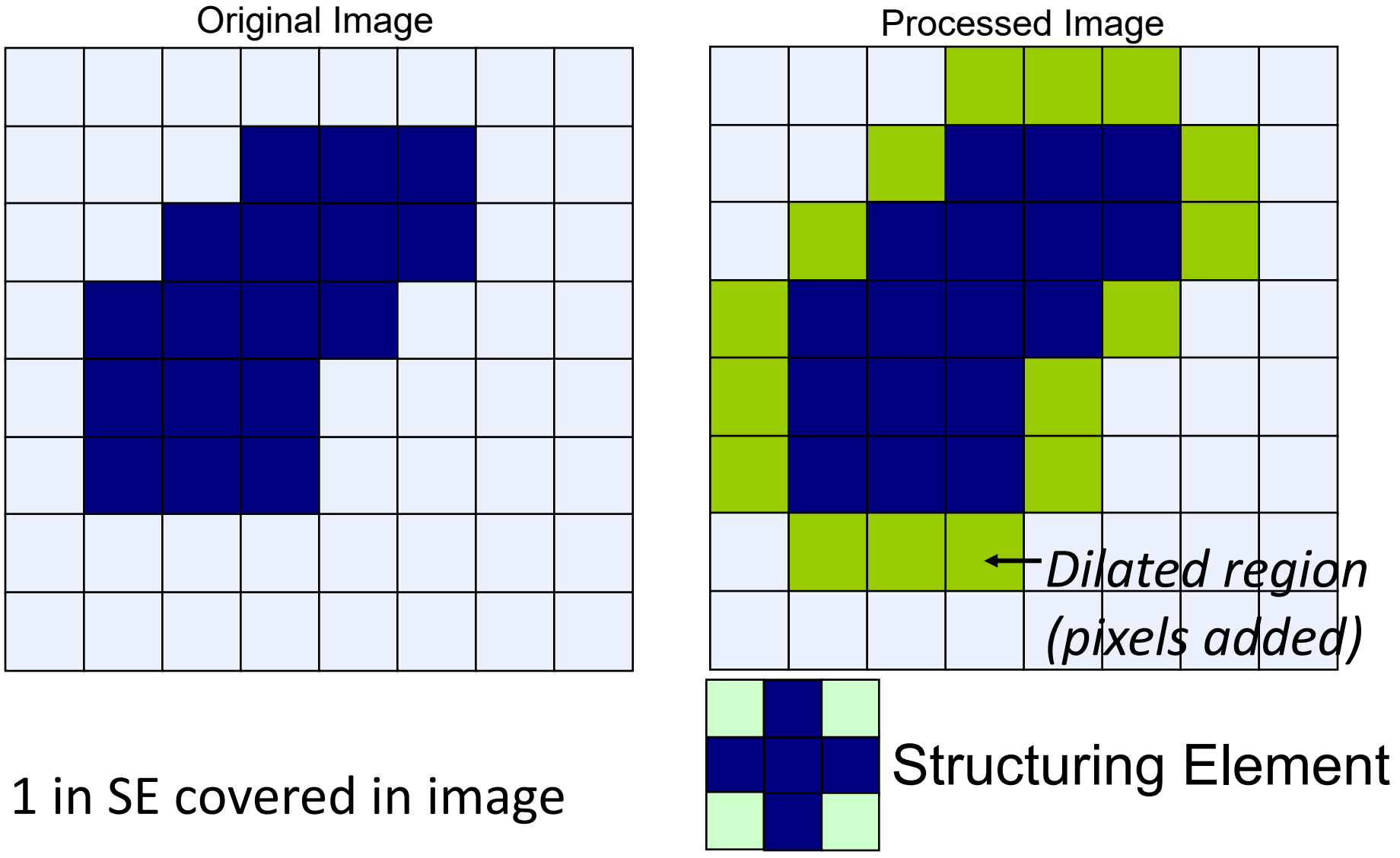

Hit: Any “1” pixels in SE cover “1” pixels in the image.

# Dilation Example



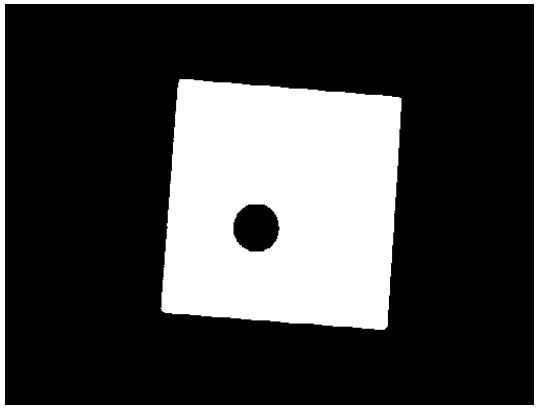

Hit: Any 1 in SE covered in image

# Dilation Example

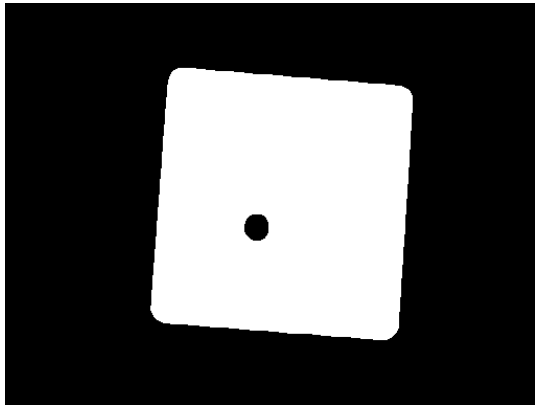



Hit: Any 1 in SE covered in image

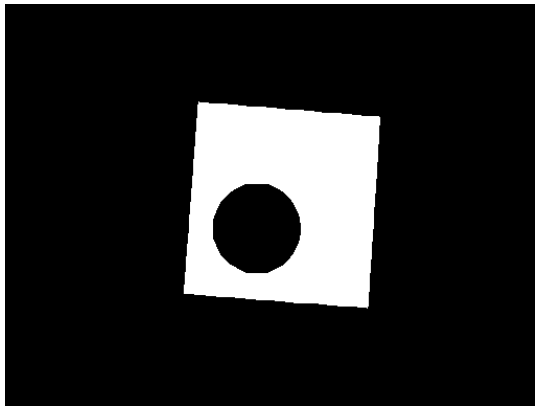
# Effects of erosion and dilation



Original



Dilation by a round structuring element.



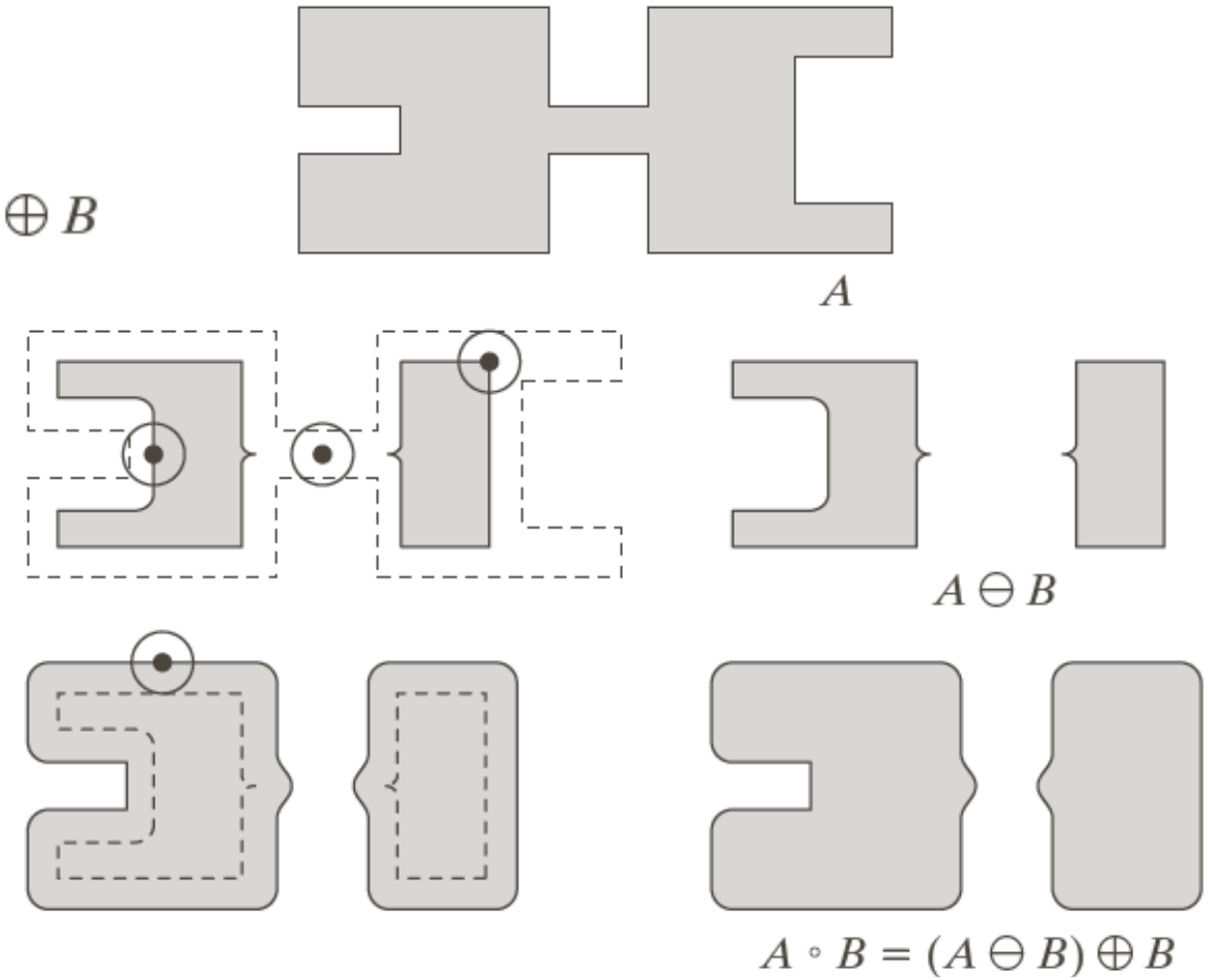
Erosion by a round structuring element.

Source of images: http://homepages.inf.ed.ac.uk/rbf/HIPR2/

# Combined operations: Opening

- Definition
  - Apply erosion then dilation

$$A \circ B = (A \ominus B) \oplus B$$

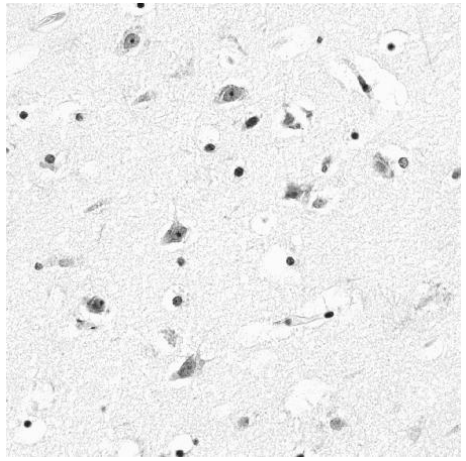



- Effect:

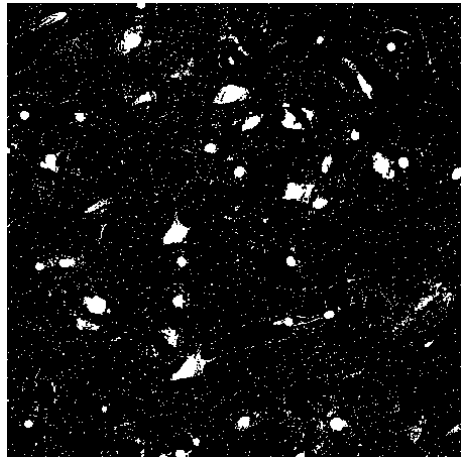
  ⇒ Removes small objects,
  preserves rough shape.

# Effects of opening

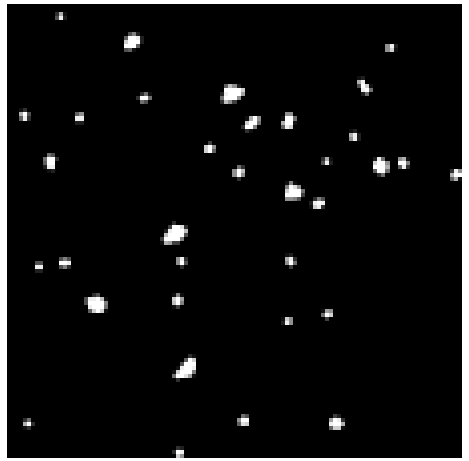
- Can filter out structures by selecting the size of structuring element.



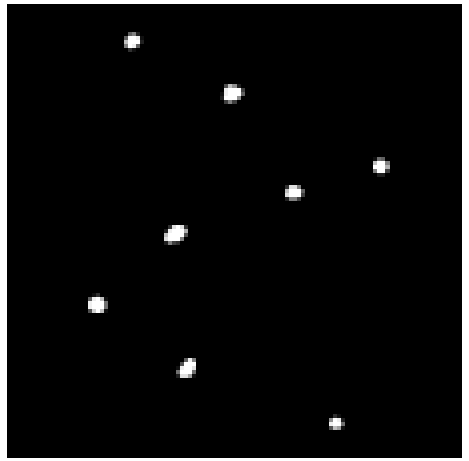
Original



Thresholded



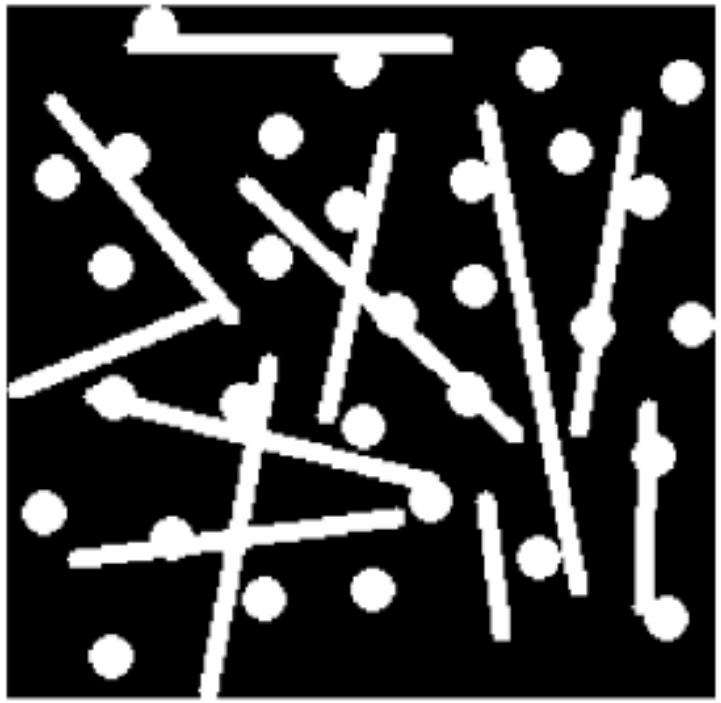
Opening by a small structuring element



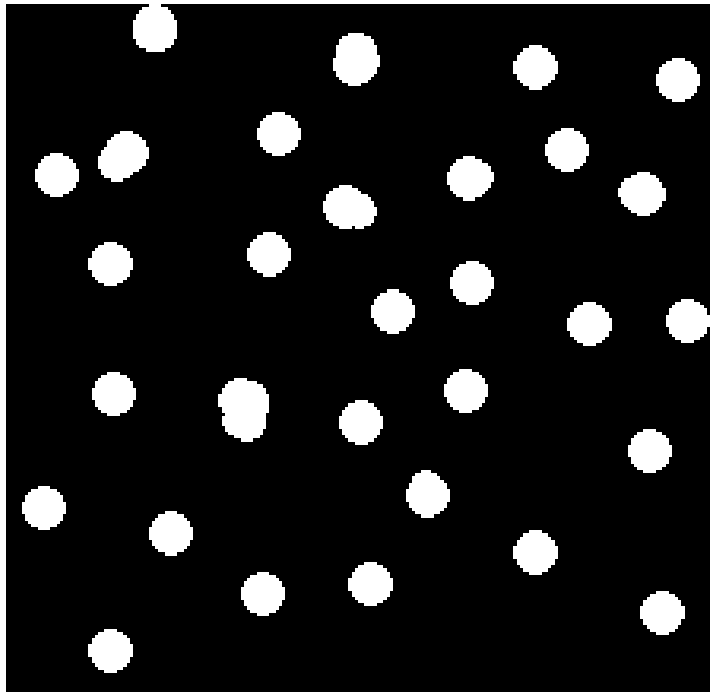
Opening by a large structuring element

# Effects of opening

- Choose the structure in image by choosing the shape of the structuring element.



Original image



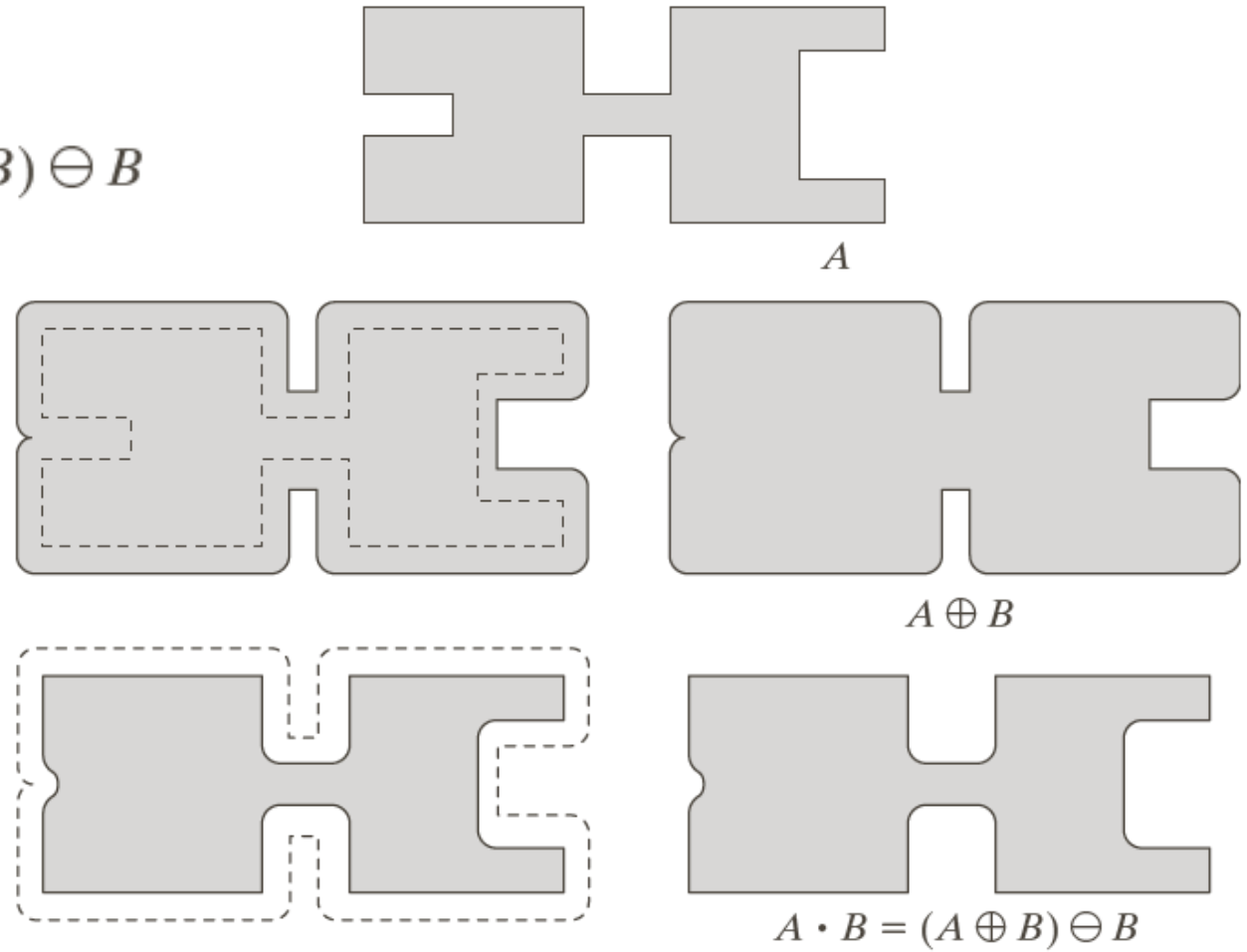
Opening by a round structuring element

Source of images: http://homepages.inf.ed.ac.uk/rbf/HIPR2/

# Combined operations: Closing

- Definition
  - Apply dilation then erosion

$$A \cdot B = (A \oplus B) \ominus B$$




- Effect

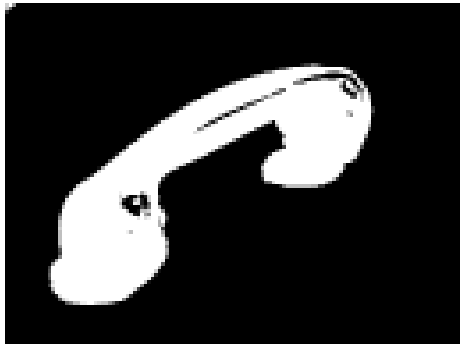
  ⇒ Fill holes, preserves the original shape.

# Effects of closing

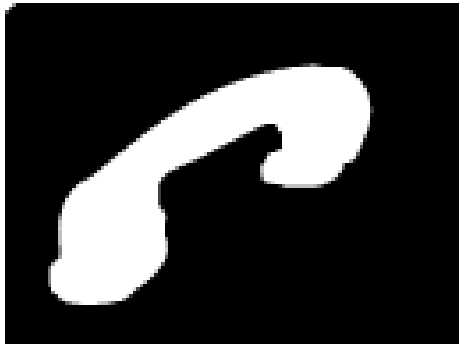
- Fill holes in thresholded image

  (*eg., reflections*)

Original

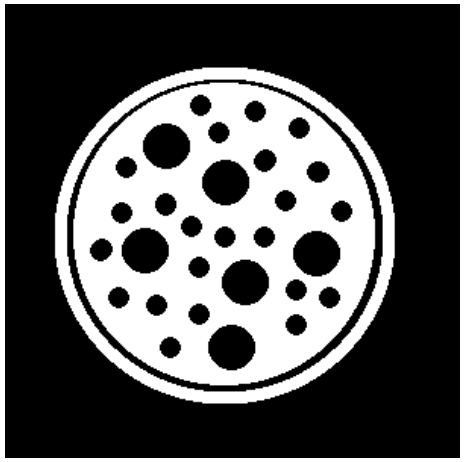


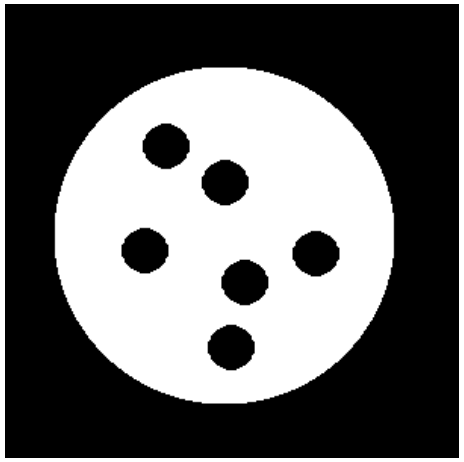
Thresholded



Closing by a round structuring element

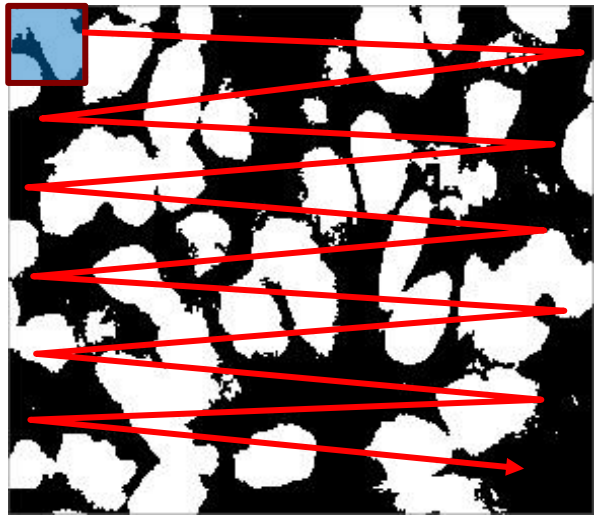
The size of structuring element determines the maximal size of holes that will be filled.





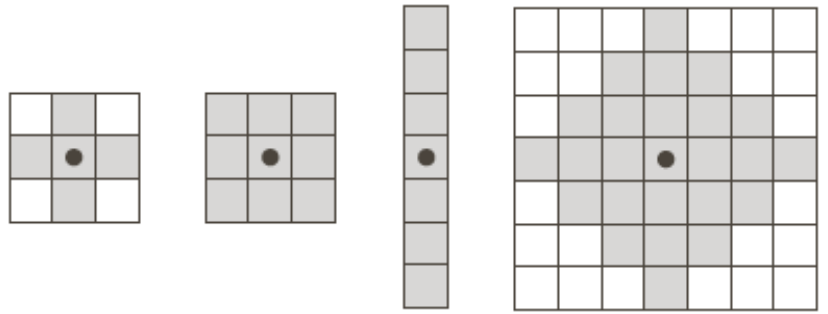
# Example: opening + closing

# Morphological operators in OpenCV



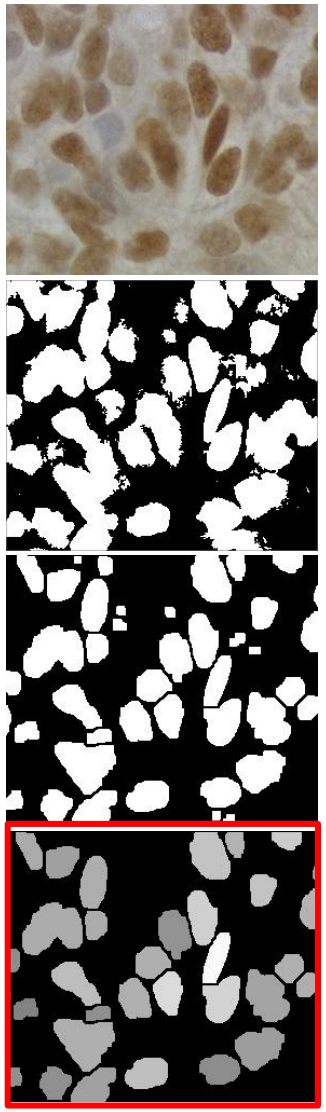
- Main operations
  - Dilation (OpenCV: `cv2.dilate`)
  - Erosion (OpenCV: `cv2.erode`)
- Several important combinations
  - Opening (OpenCV: `cv2.morphologyEx(img,cv2.MORPH_OPEN, kernel)` )
  - Closing (OpenCV: `cv2.morphologyEx(img,cv2.MORPH_CLOSE, kernel)`)
  - Boundary extraction
- Much more available (see help)

Examples of structuring elements:



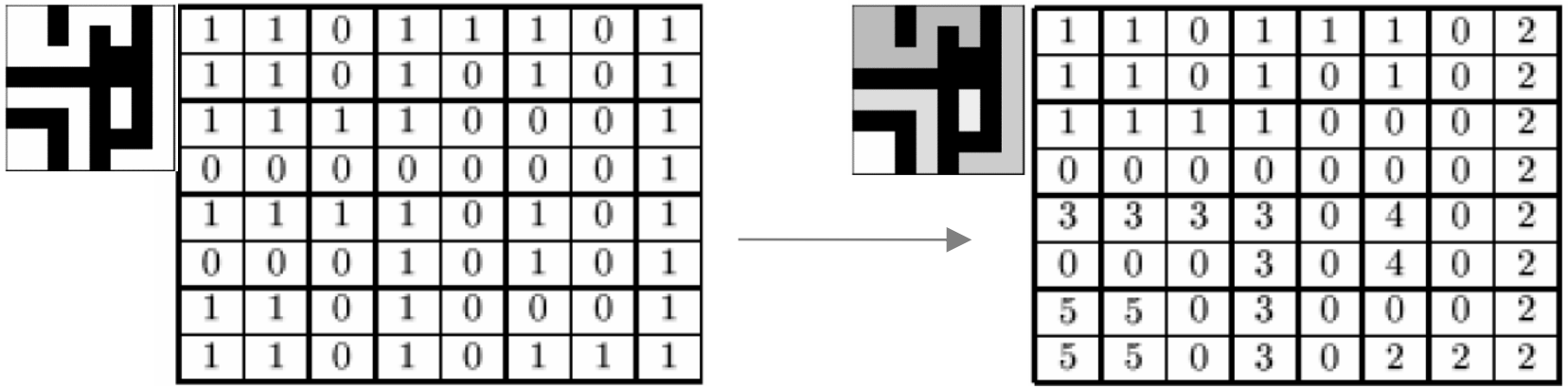
Machine perception

# LABELLING REGIONS



# Connected components for labeling

- Goal: find separate connected regions

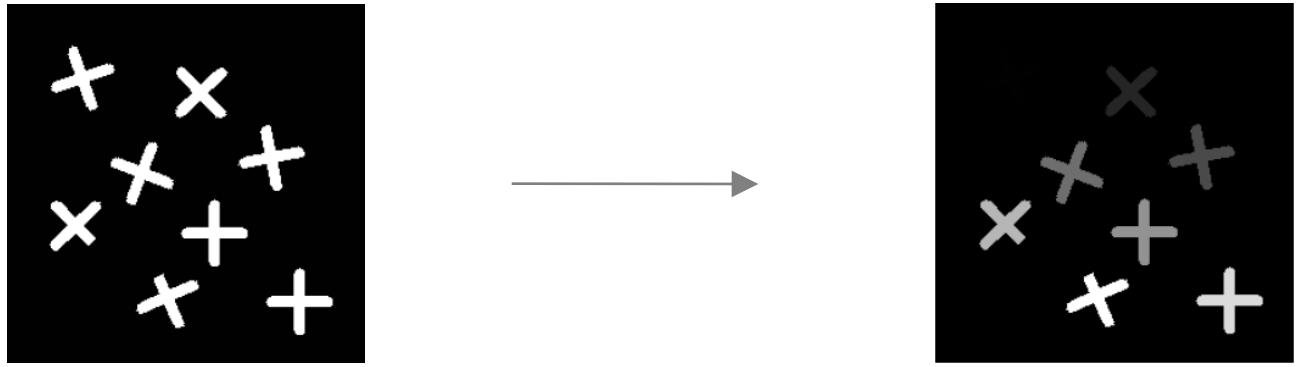


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |

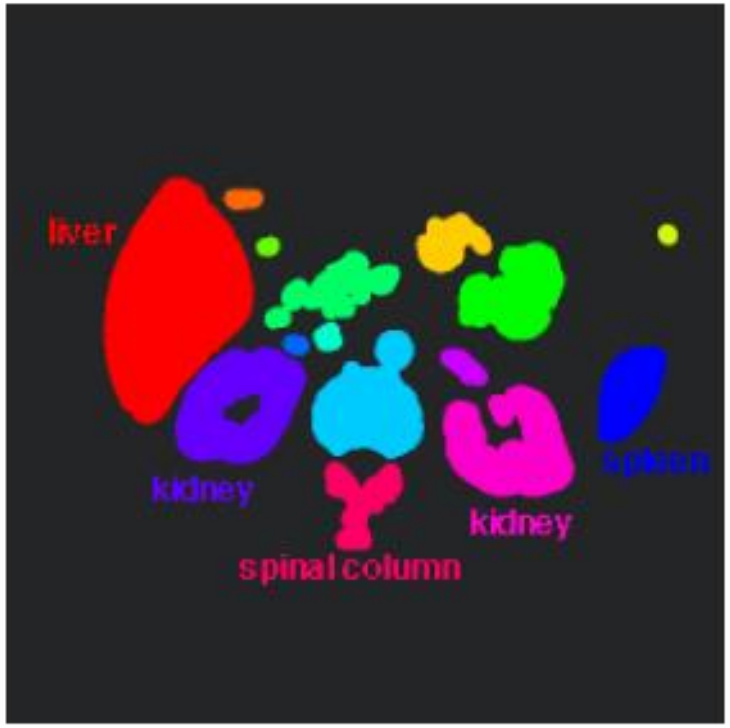
Binary image

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 2 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 2 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 2 |
| 3 | 3 | 3 | 3 | 0 | 4 | 0 | 2 |
| 0 | 0 | 0 | 3 | 0 | 4 | 0 | 2 |
| 5 | 5 | 0 | 3 | 0 | 0 | 0 | 2 |
| 5 | 5 | 0 | 3 | 0 | 2 | 2 | 2 |

connected components



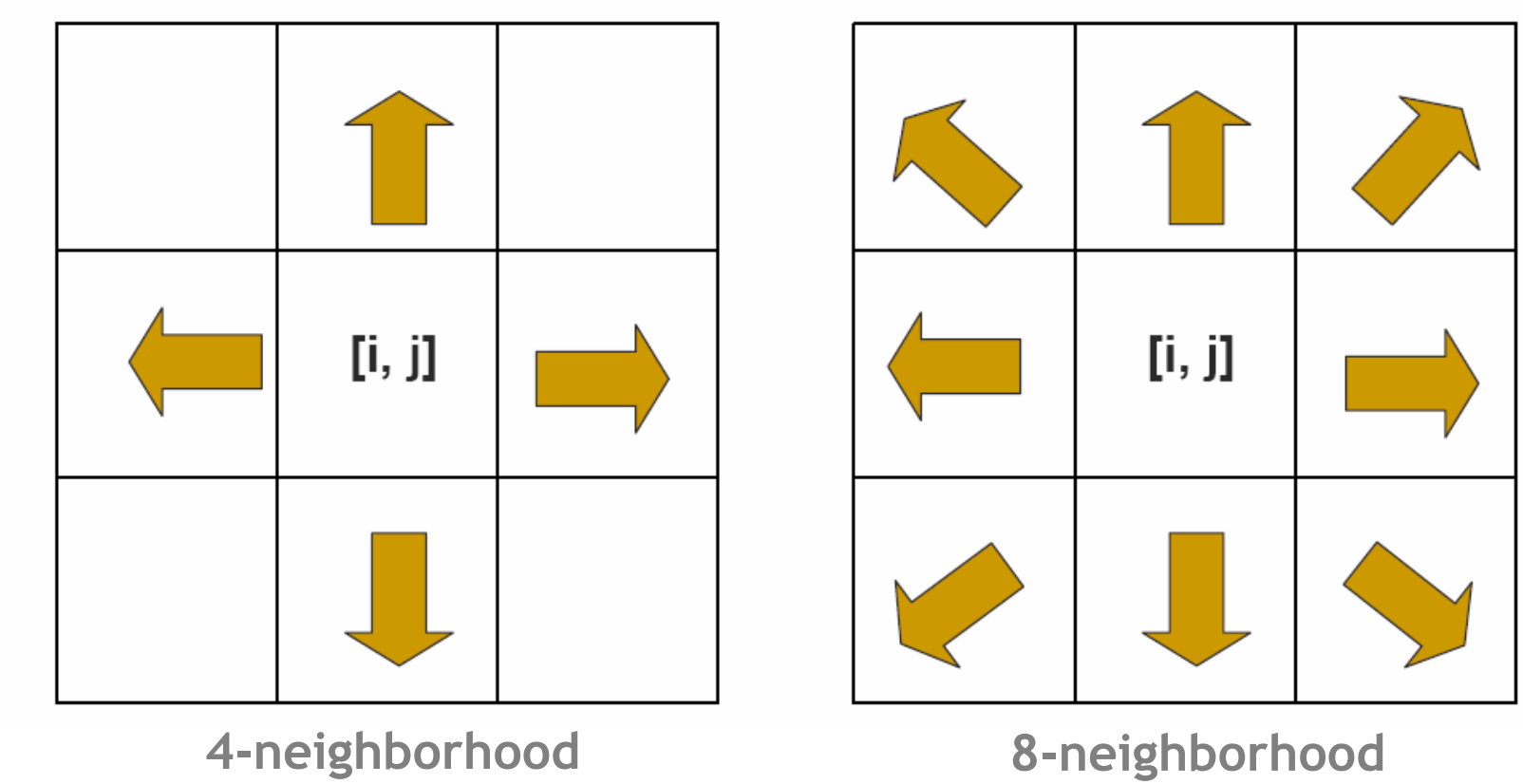
# Examples of connected components




connected components of 1's from thresholded image

# Connectivity

- Determines which pixels are considered neighbors.




4-neighborhood

8-neighborhood

# Sequential connected components

- Process image from left to right, from top to bottom:

  1.) If the current pixel value is 1

  i.) If only one neighbor (left or top) is 1, copy its label.

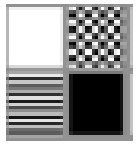
  ii.) If both neighbors are 1 and have same label, copy that label.

  iii.) If they have different labels

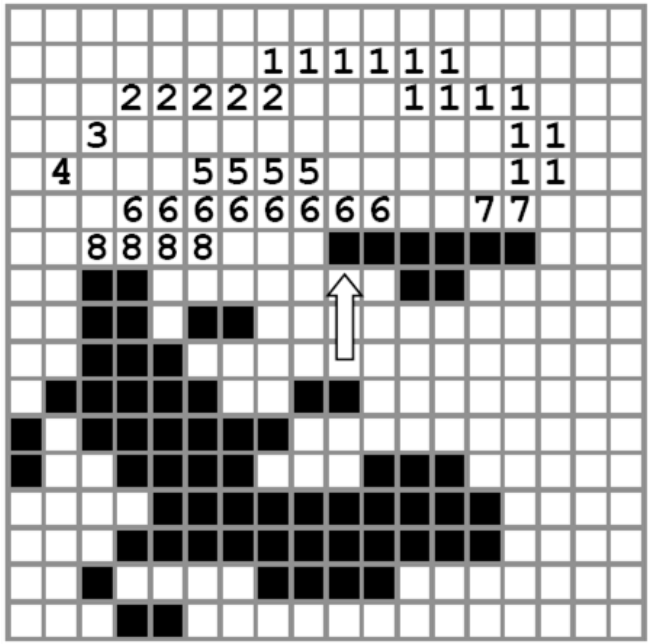
  – Copy label from the left.

  – Update the table of equivalent labels.

  iv.) Otherwise form a new label.




{1, 2, 7}
{3}
{4}
{5, 6, 8}

- Relabel with the smallest equivalent labels.

# Example SCC: 8-connectivity

(a)

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

0 Background

1 Foreground

(b) only background neighbors

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

new label (**2**)

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

# Example SCC: 8-connectivity

(c) exactly one neighbor label

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

neighbor label is propagated

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

(d) two different neighbor labels

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 3 | 3 | 0 | 4 | 0 | |
| 0 | 5 | 5 | 5 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

one of the labels (**2**) is propagated

(Update equivalency table {2,5})

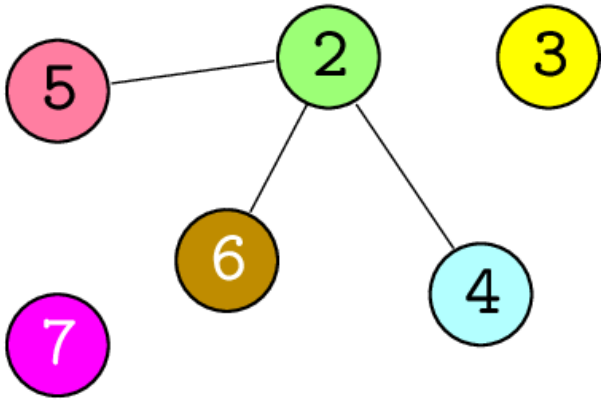
| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 3 | 3 | 0 | 4 | 0 | |
| 0 | 5 | 5 | 5 | 2 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | |
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

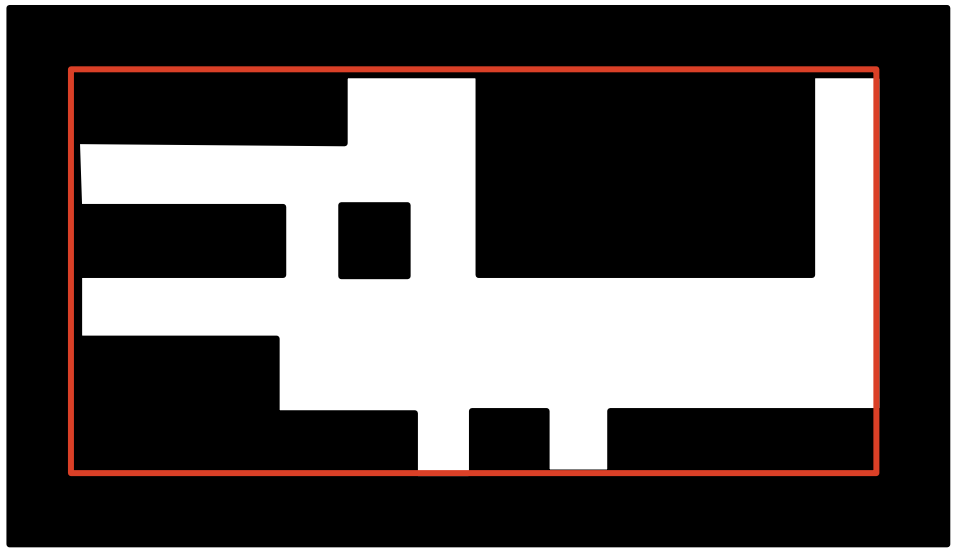
# Example SCC: 8-connectivity

First pass: label

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 3 | 3 | 0 | 4 | 0 |
| 0 | 5 | 5 | 5 | 2 | 2 | 2 | 0 | 0 | 3 | 0 | 0 | 4 | 0 |
| 0 | 0 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 4 | 0 |
| 0 | 6 | 6 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 0 |
| 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 0 |
| 0 | 7 | 7 | 0 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Equivalency table

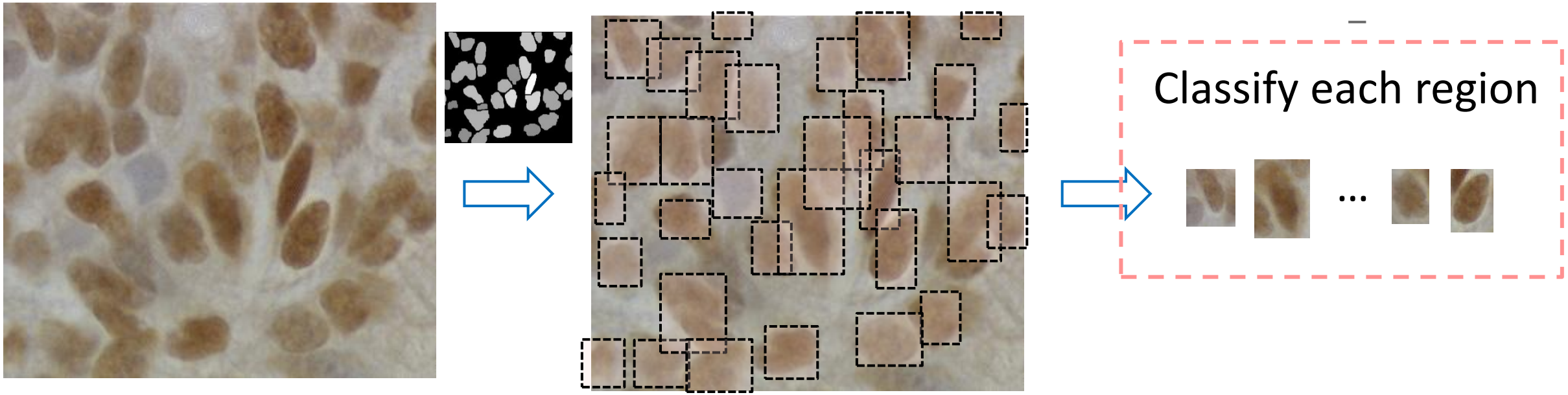


I==2

Second pass: apply equivalences

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 2 | 2 | 0 | 0 | 3 | 3 | 0 | 2 | 0 |
| 0 | 2 | 2 | 2 | 2 | 2 | 2 | 0 | 0 | 3 | 0 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 2 | 0 |
| 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 0 |
| 0 | 0 | 0 | 0 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 0 |
| 0 | 7 | 7 | 0 | 0 | 0 | 2 | 0 | 2 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |



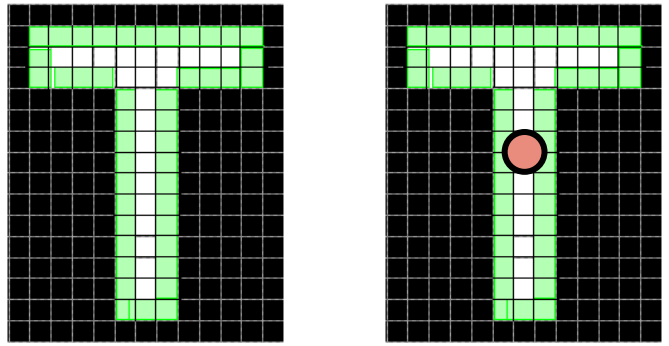

Machine perception

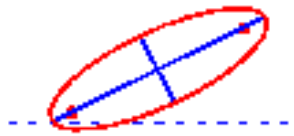
# REGION DESCRIPTORS

# Simple region descriptors

- A region can be detected using the connected components.
- How to describe it?
- Some examples:
  - Area $A$
  - Perimeter $l$
  - Compactness $c=l^2/(4\pi A)$
  - Circularity, roundness $1/c$
  - Centroid (center of mass)
  - Major and minor axes $\lambda_1$, $\lambda_2$
  - Eccentricity $||\lambda_1||/||\lambda_2||$
  - Minimal bounding box area $A_m = h\,b$
  - Rectangularity $A/A_m$

cv2: *connectedComponentsWithStats*

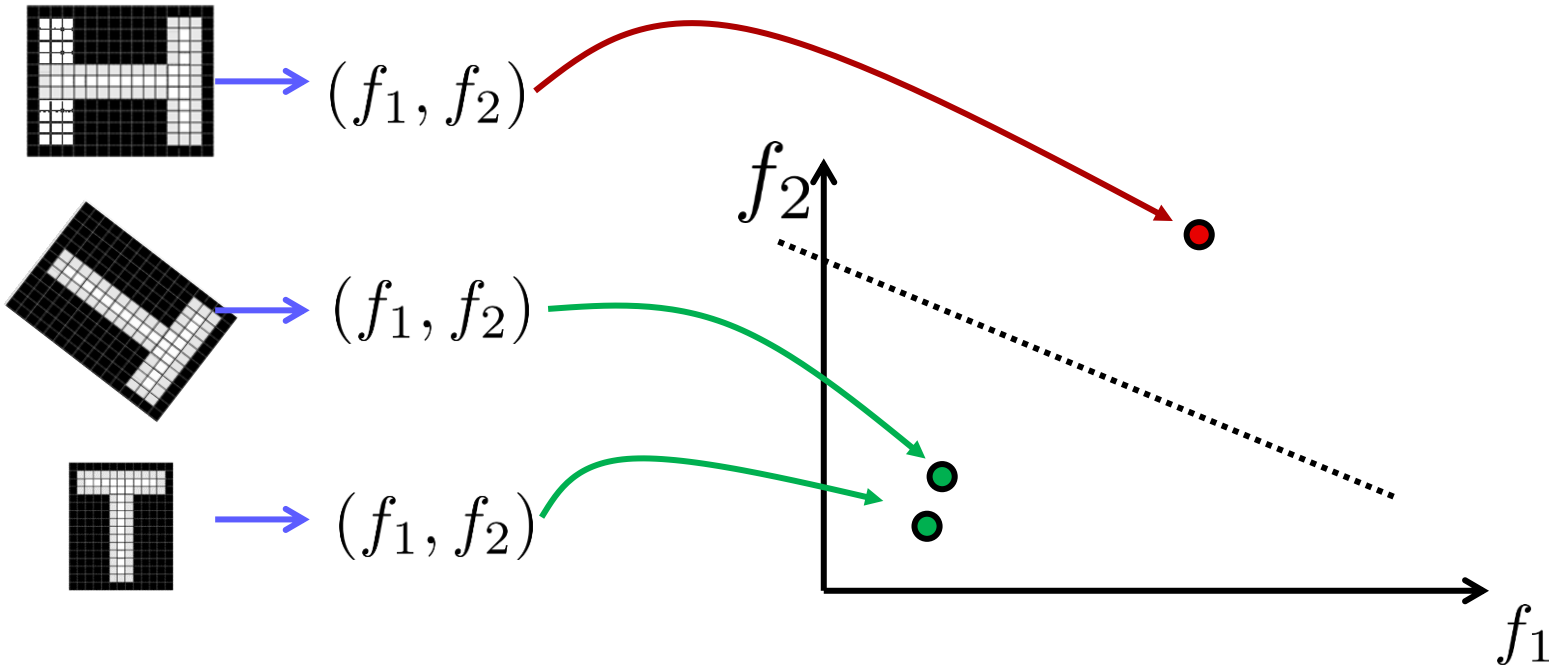




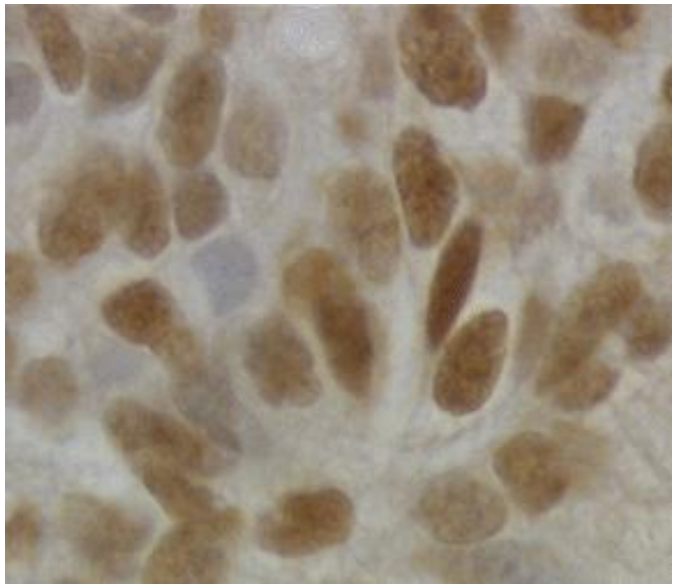
(Easy to come up with your own)

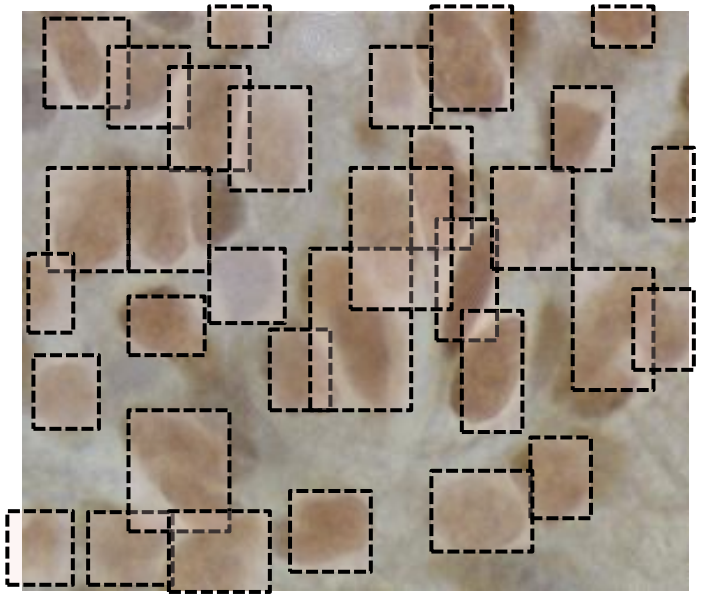
# Require a level of invariance (App dependent)

- Ideal descriptor will map:
  - Two images of the same object close-by in feature space.
  - Two images of different objects to points far between each other.

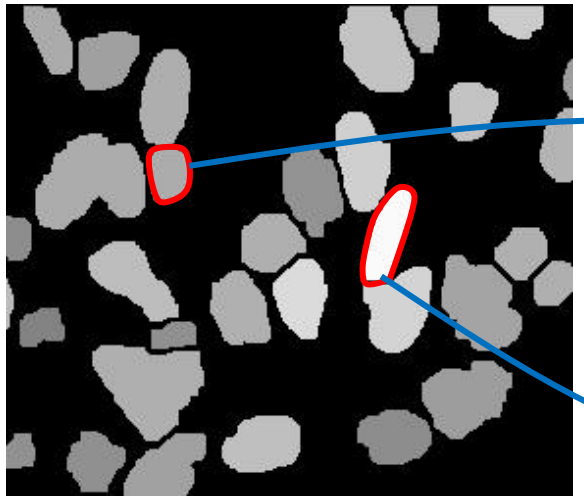


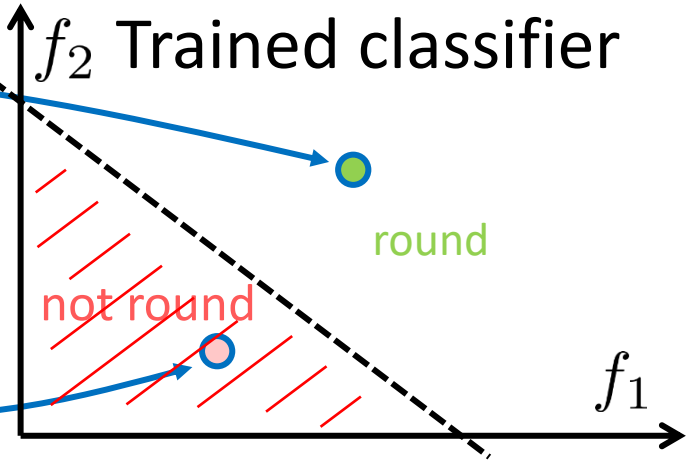
# Task: Detect round cells

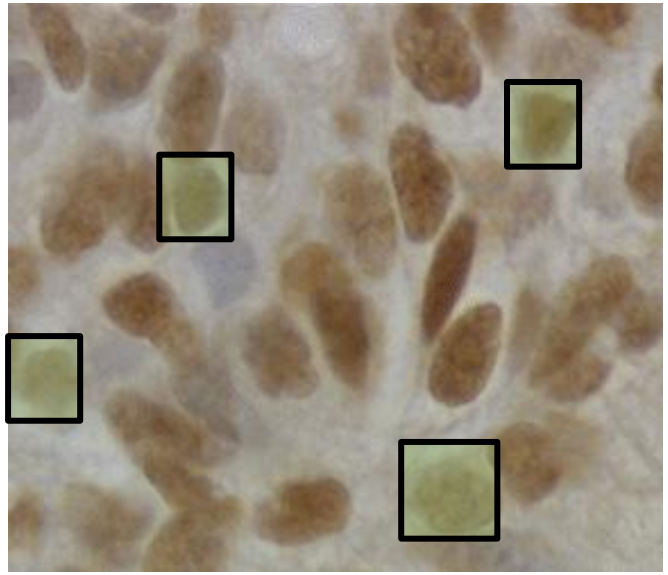




Classify each region





# Summary: Binarization

- Pros
  - Fast, simple to store
  - Simple techniques
  - Works in constrained setups
- Cons
  - Difficult to get „clean“ shapes
  - Many real-world scenarios contain noise
  - Often too coarse representation
  - Not robust in 3D view changes

# References

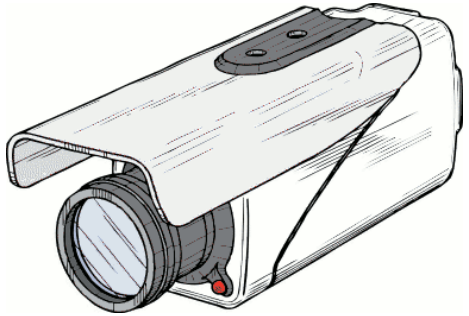

- R.C. Gonzales, R.E. Woods, *Digital Image Processing*. Prentice Hall, 2001
- Morfology – Spletni tutorial: http://homepages.inf.ed.ac.uk/rbf/HIPR2/
- David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition), (prva izdaja dostopna na spletu)
- R. Szeliski,Computer Vision: Algorithms and Applications, 2010
- R. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, 2nd Edition, Cambridge University Press, 2004
- Rob Fegus, „Computer Vision“, lectures
- Bastian Leibe „Computer Vision“, lectures
- Kristen Grauman „Computer Vision“, Lectures
- Otsu, N., "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 9, No. 1, 1979, pp. 62-66.
- Barron, J.T., A Generalization of Otsu’s Method and Minimum Error Thresholding, CVPR2020

Univerza v Ljubljani

ViCOS Visual Cognitive Systems Lab

# Machine perception
# Image processing 2

Matej Kristan




Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

Machine perception

# FILTERING

# Can be applied for…

- Noise reduction and image restoration

noisy

Filtering

Less noisy

- Structure extraction/enhancement (later in the course)

Filtering

# Types of image noise

- Salt and pepper (*sol in poper)*
  - Random black and white dots.
- Impulse noise (Impulzni šum)
  - Random occurrence of white dots.
- Gaussian noise (Gausov šum)
  - The intensity variation sampled from a Gaussian (Normal) distribution.

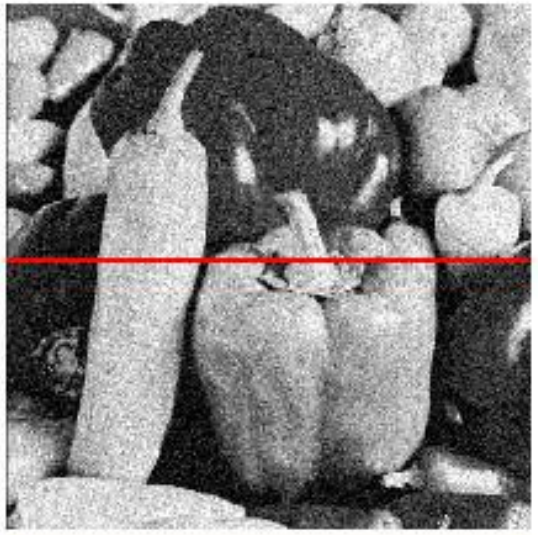
Original

Salt and pepper noise

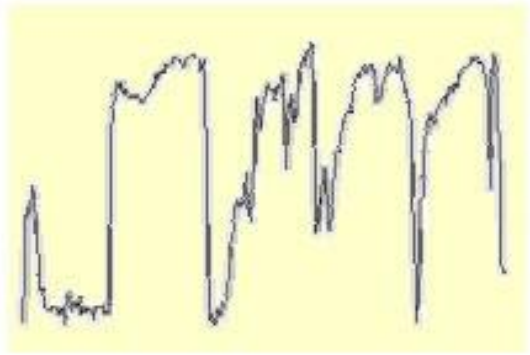
Impulse noise

Gaussian noise

# Gaussian noise

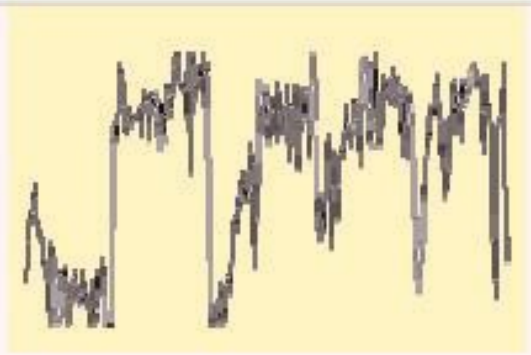


```
Matlab:
>> im = imread("peppers.jpg");
>> noise = randn(size(im)).*5;
>> output = im + noise;
```



Ideal Image

$$\overbrace{\bar{f}(x,y)}$$

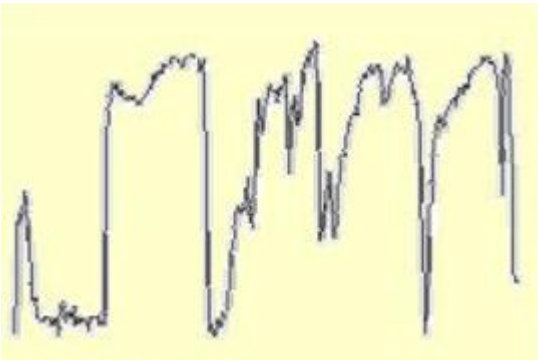


$$f(x,y) = \overbrace{\bar{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$$

Gaussian i.i.d. ("white") noise:

$$\eta(x,y) \sim \mathcal{N}(\mu,\sigma)$$

?

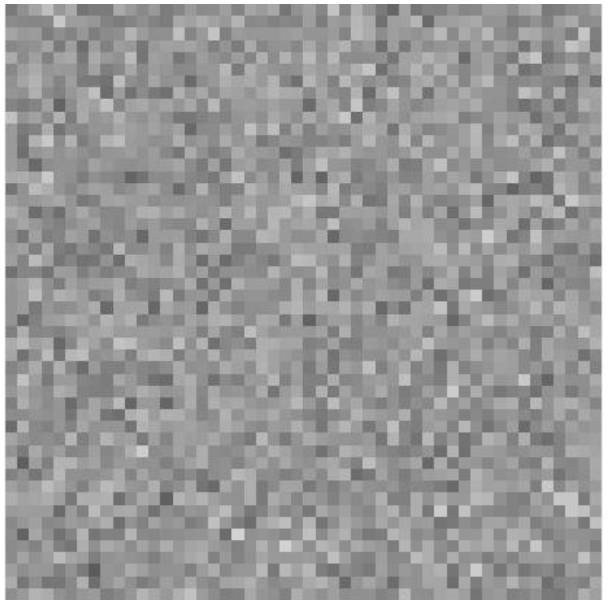


Ideal Image

$$\overbrace{\bar{f}(x,y)}$$

# How to remove a Gaussian noise?

A = ones(100,100)*150



B = A + randn(size(A))*20


C = B*0 + mean(B(:))

- $\langle B \rangle = \langle A \rangle + \langle G_{noise} \rangle = \langle A \rangle = 150$
- Solution: just compute the average value!
- Might it really be this simple?

# Let’s try to remove the noise…

- Assumption:
  - Pixels are similar to their neighboring pixels
  - The noise is independent among pixels (“i.i.d. = independent, identically distributed”)

- So let’s compute an improved estimate of pixel’s intensity by replacing it with an average of pixel intensities in its immediate neighborhood…

# A moving average 2D

$F[x, y]$

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

$G[x, y]$

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

# A moving average 2D

$F[x, y]$

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

$G[x, y]$

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

# A moving average 2D

$F[x, y]$

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

$G[x, y]$

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

# A moving average 2D

$F[x, y]$

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

$G[x, y]$

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

# A moving average 2D

$F[x, y]$

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

$G[x, y]$

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

# A moving average 2D

- Assume the averaging window size is 2k+1 x 2k+1:

$$G[i,j] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Equal weights for all pixels.}} \underbrace{\sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i+u, j+v]}_{\text{A loop over all pixels within the neighbourhood of F[i,j].}}$$

- Now let's generalize this by making a weight depend on relative position from the central element.

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\text{Nonuniform weights}} F[i+u, j+v]$$

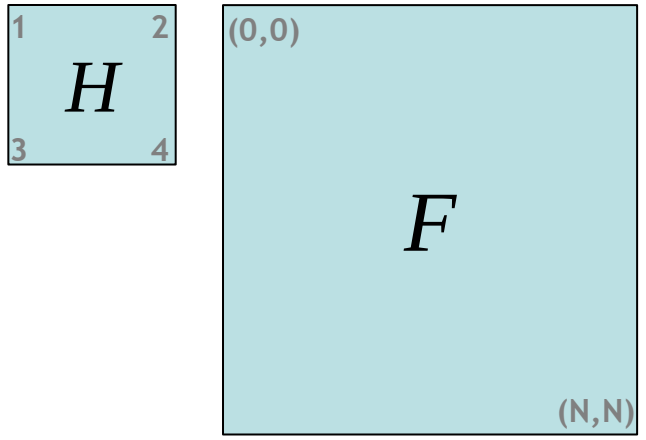
# Correlation filtering

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

- This is called cross-correlation and abbreviate as:

$$G = H \otimes F$$

- Image filtering
  - Replace image intensity with a weighted sum of a window centered at that pixel.
  - The weights in the linear combination are prescribed by the filter's *kernel*.



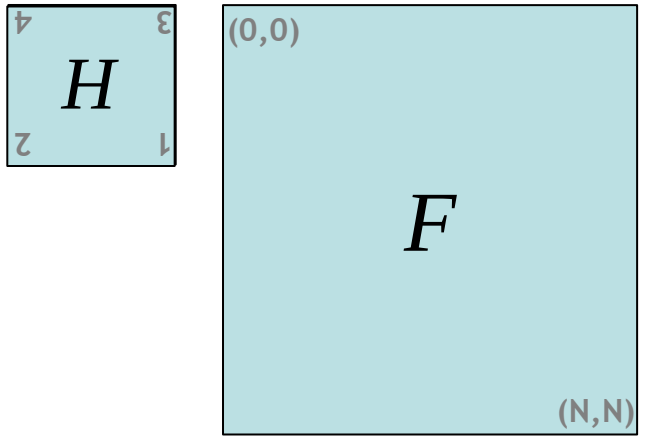
# Convolution as correlation

- Compute convolution by cross-correlation:
  - Flip the filter in both dimensions (horizontal + vertical)
  - Apply cross-correlation

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

$$G = H \star F$$

*convolution operator*



# Convolution vs. Correlation

- Correlation

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u, j+v]$$

$$G = H \otimes F$$

**Notice the difference?**

- Convolution

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u, j-v]$$

$$G = H \star F$$

(we will also use "*" to denote convolution, i.e., $G = H * F$.)

- Comment:
  - For a symmetric filter, $H[-u,-v] = H[u,v]$, correlation ≡ convolution.

# Properties of convolution

- Shift-invariant:

  *The filter weights remain the same, regardless the position.*

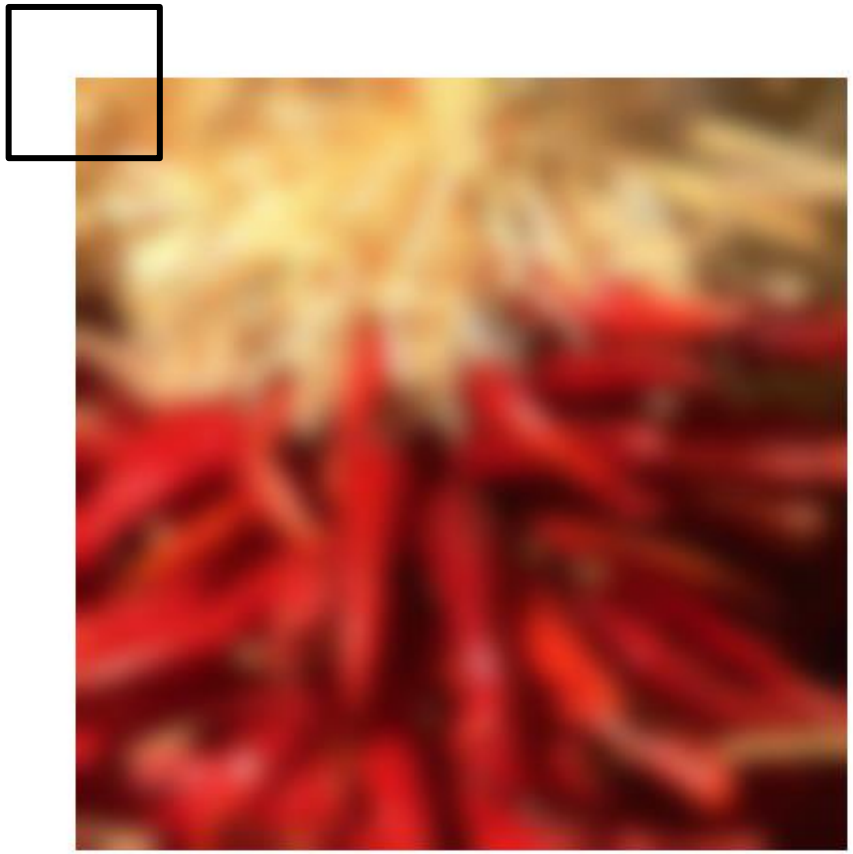
- Linear (superposition & scaling): $h * (\alpha_1 f_1 + \alpha_2 f_2) = \alpha_1 (h * f_1) + \alpha_2 (h * f_2)$
- Commutative: $f * g = g * f$
- Associative: $(f * g) * h = f * (g * h)$
  - As result, application of multiple filters is equal to application of a single filter :

$$\big((f * b_1) * b_2\big) * b_3 = f * (b_1 * b_2 * b_3)$$

- Identity: $f * e = f$ , where $e$= [..., *0, 0, 1, 0, 0,* ...] a unit impulse.
- Derivative: $\frac{\partial}{\partial x}(f * g) = \left(\frac{\partial}{\partial x} g\right) * f = \left(\frac{\partial}{\partial x} f\right) * g$

# Filtering: Boundary conditions

- What to do at the image boundaries?
  - The kernel exceeds image boundaries at the edge
  - Need for extrapolation
  - Methods (assumptions):
    - Crop (black)
    - Bend image around
    - Replicate edges
    - Mirror image



# Filtering: Boundary conditions

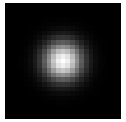
- What to do at the image boundaries?
  - The kernel exceeds image boundaries at the edge
  - Need for extrapolation
  - Methods (Python): `cv2.filter2D( … BorderTypes= )`

| Enumerator | |
|---|---|
| BORDER_CONSTANT<br>Python: cv.BORDER_CONSTANT | `iiiiii\|abcdefgh\|iiiiiii` with some specified `i` |
| BORDER_REPLICATE<br>Python: cv.BORDER_REPLICATE | `aaaaaa\|abcdefgh\|hhhhhhh` |
| BORDER_REFLECT<br>Python: cv.BORDER_REFLECT | `fedcba\|abcdefgh\|hgfedcb` |
| BORDER_WRAP<br>Python: cv.BORDER_WRAP | `cdefgh\|abcdefgh\|abcdefg` |
| BORDER_REFLECT_101<br>Python: cv.BORDER_REFLECT_101 | `gfedcb\|abcdefgh\|gfedcba` |
| BORDER_TRANSPARENT<br>Python: cv.BORDER_TRANSPARENT | `uvwxyz\|abcdefgh\|ijklmno` |
| BORDER_REFLECT101<br>Python: cv.BORDER_REFLECT101 | same as BORDER_REFLECT_101 |
| BORDER_DEFAULT<br>Python: cv.BORDER_DEFAULT | same as BORDER_REFLECT_101 |
| BORDER_ISOLATED<br>Python: cv.BORDER_ISOLATED | do not look outside of ROI |

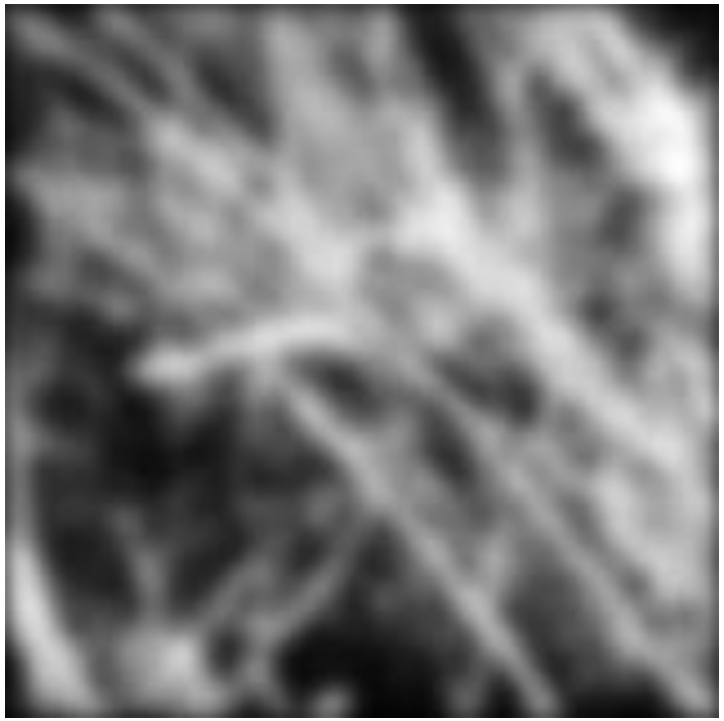
https://docs.opencv.org/master/d2/de8/group__core__array.html#ga209f2f4869e304c82d07739337eae7c5

Caution: the method performs *correlation*, not *convolution*

# Filtering kernels: A Gaussian kernel



*Original*



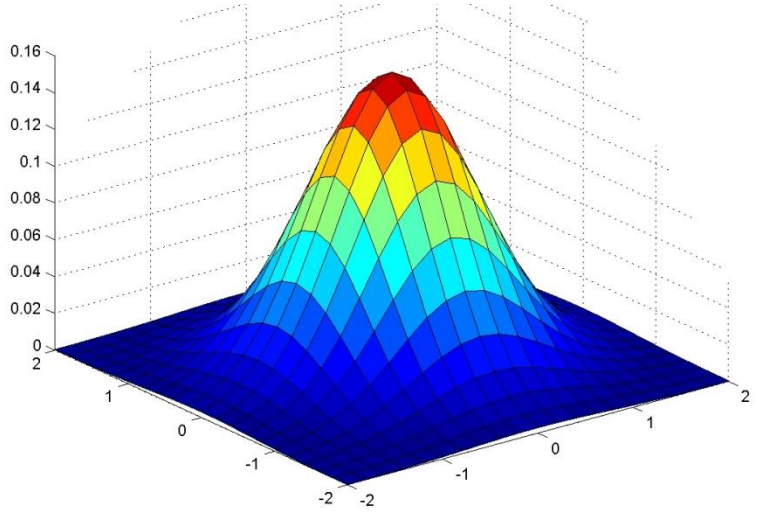
*Filtered*

# Filtering kernels: A Gaussian kernel

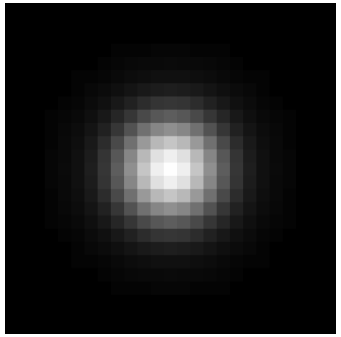
- Instead of using uniform weights, pixels closer to the center should have higher weight.

- A kernel with such property: A Gaussian

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

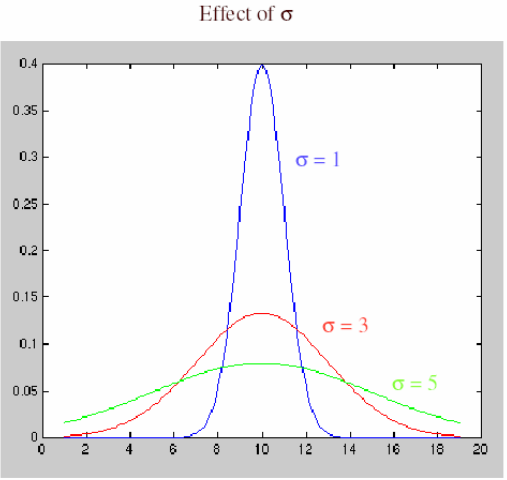


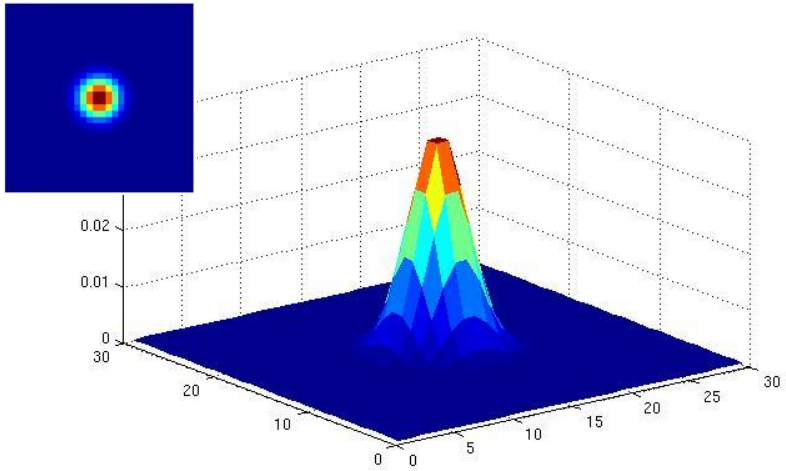

- Rotation symmetric, bell-shaped



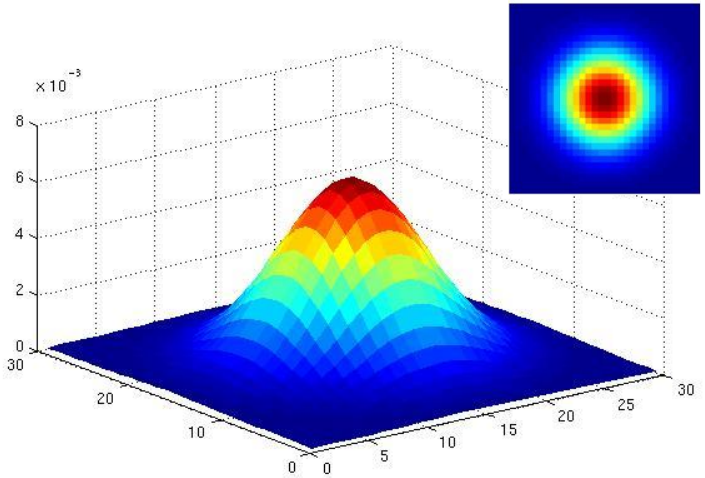
# Filtering kernels: A Gaussian kernel

- How about parameters?
- *Variance* $\sigma^2$ determines the extent of smoothing...



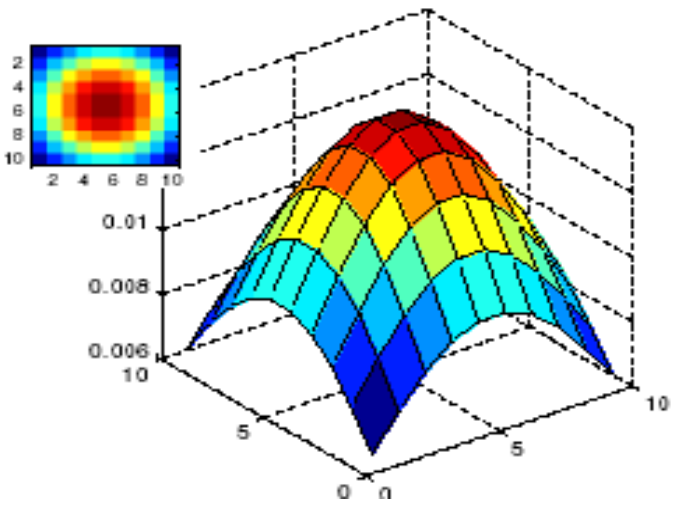
$\sigma = 2$ by kernel
30×30



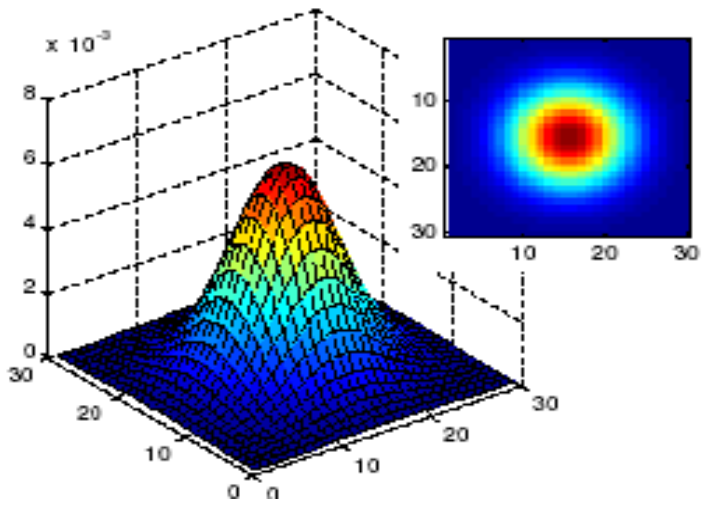
$\sigma = 5$ by kernel
30×30

# Filtering kernels: A Gaussian kernel

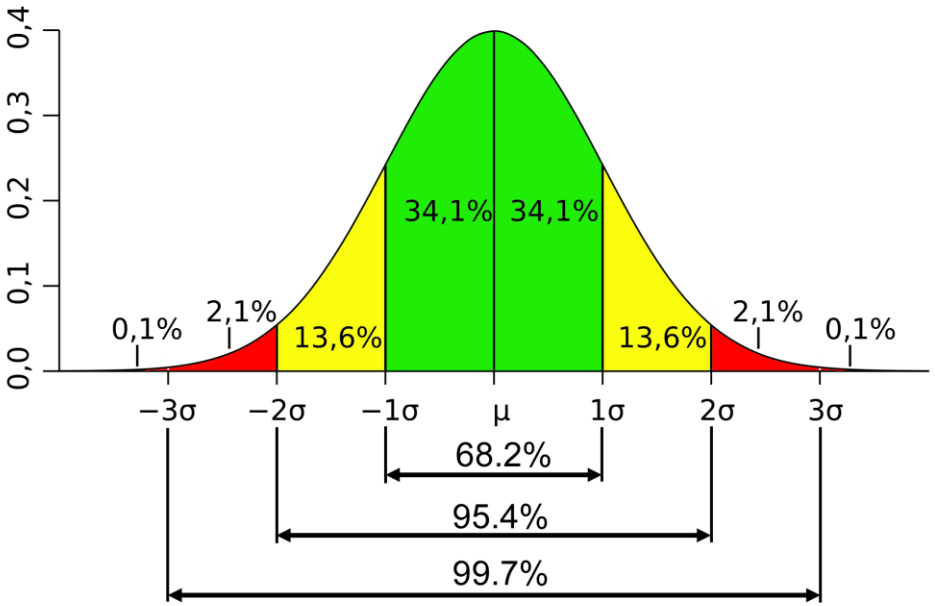
- How about parameters?
- *Kernel size*!
  - Infinite support, but discretization makes it finite.



$\sigma = 5$ with $10 \times 10$ kernel



$\sigma = 5$ with $30 \times 30$ kernel

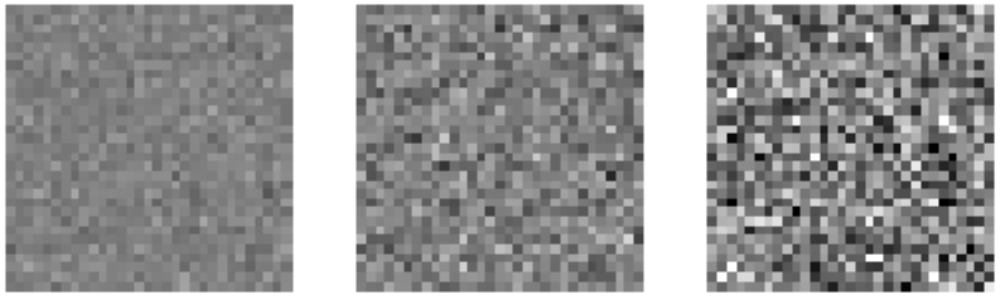



- Rule of thumb: set half size of the kernel to $3\sigma$

# Effects of smoothing

Increasing the noise extent→

σ=0.05 σ=0.1 σ=0.2



no smoothing

Increasing the kernel size →

# Efficient implementation

- In case a filter is separable, we can rewrite it as a convolution of two 1D filters:

$$G = g_x * g_y$$

- Recall: Convolution is linear, associative + commutative

$$I * G = I * (g_x * g_y) = (I * g_x) * g_y$$

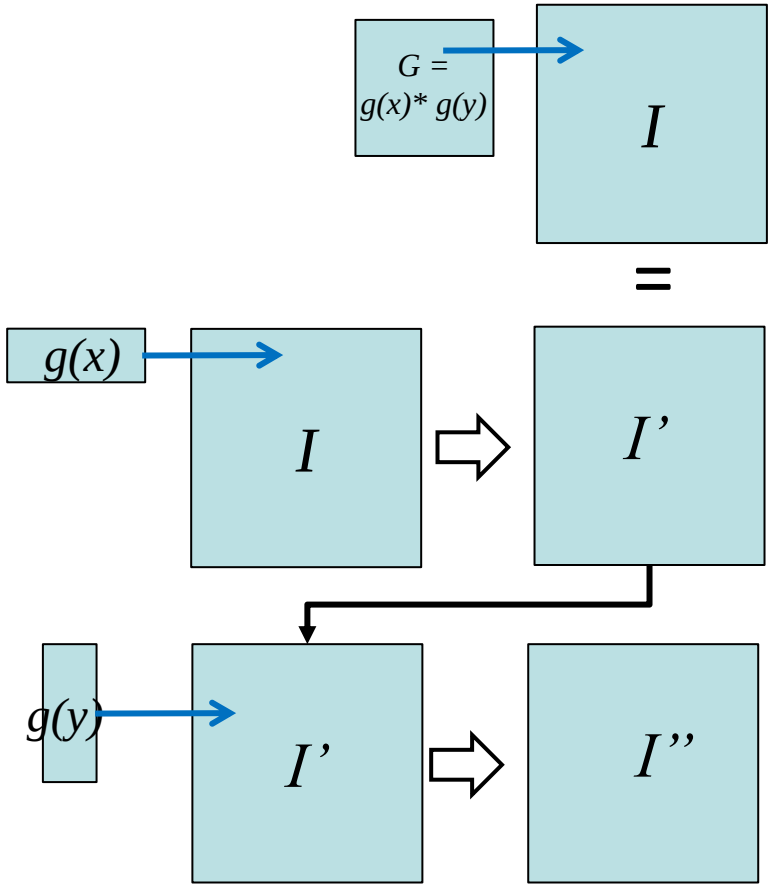
- Apply convolution at each row separately using a 1D kernel:

$$g(y) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-y^2/(2\sigma^2))$$

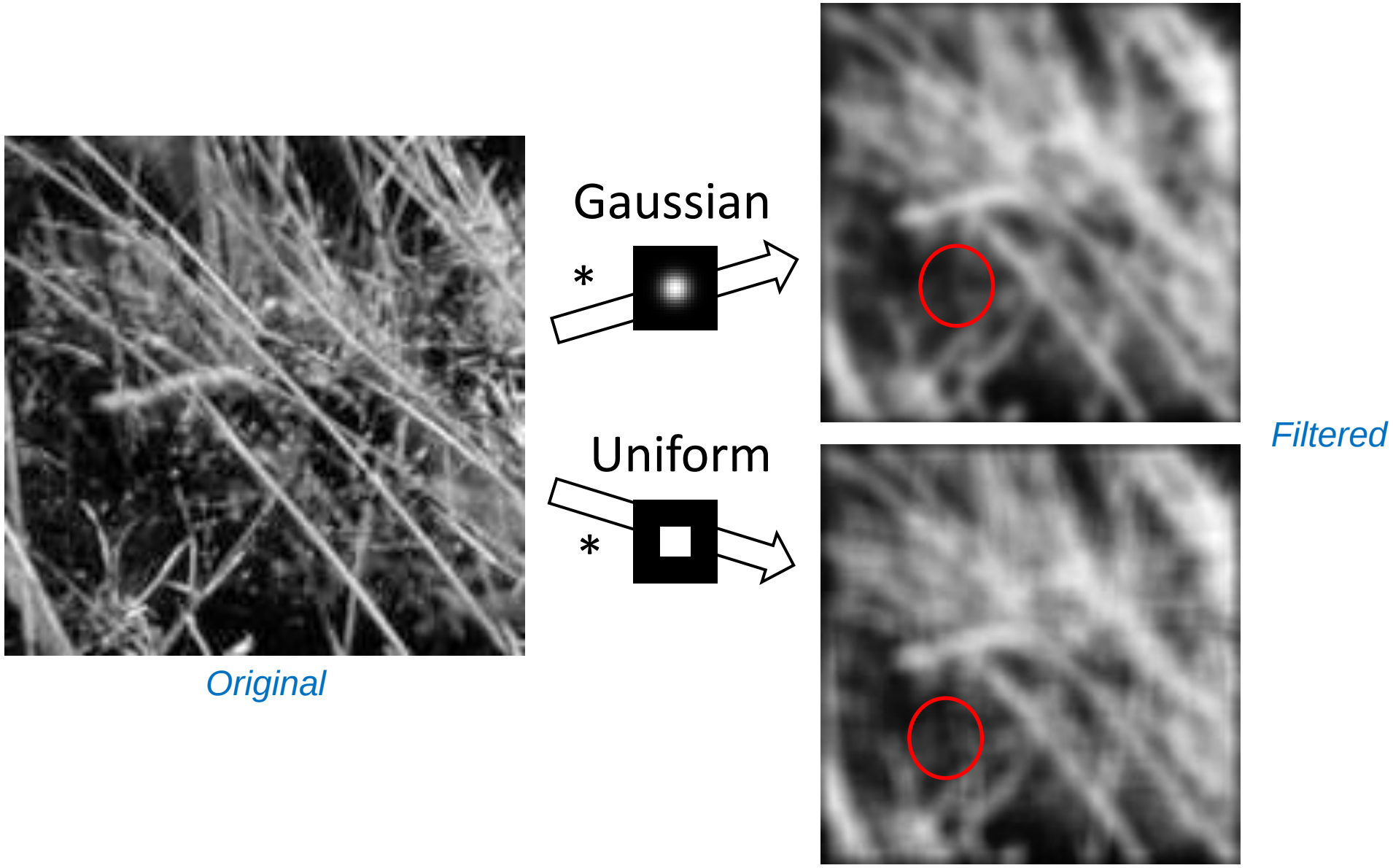
- Next apply a 1D convolution at each column:

$$g(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-x^2/(2\sigma^2))$$

- Both, Uniform as well as Gaussian kernels are separable!



# Strange artefacts in convolution results…



# Convolution and spectrum

- Convolution of two functions in image space is equivalent to the product of their corresponding Fourier transforms (spectra).

$$\mathcal{F}(f * g) = \mathcal{F}(f) \odot \mathcal{F}(g)$$

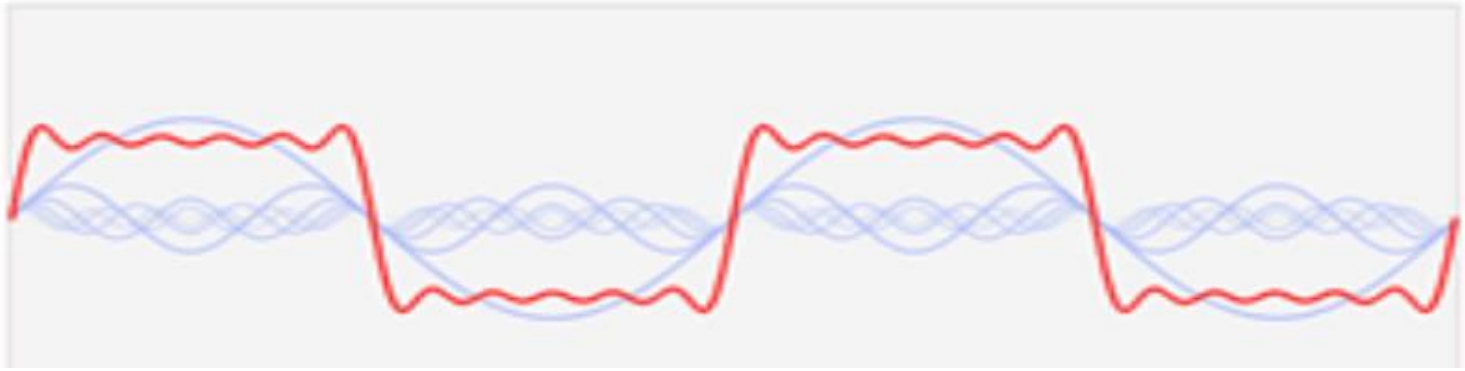
Image $f$ and filter $g$

Fourier transforms of $f$ and $g$.

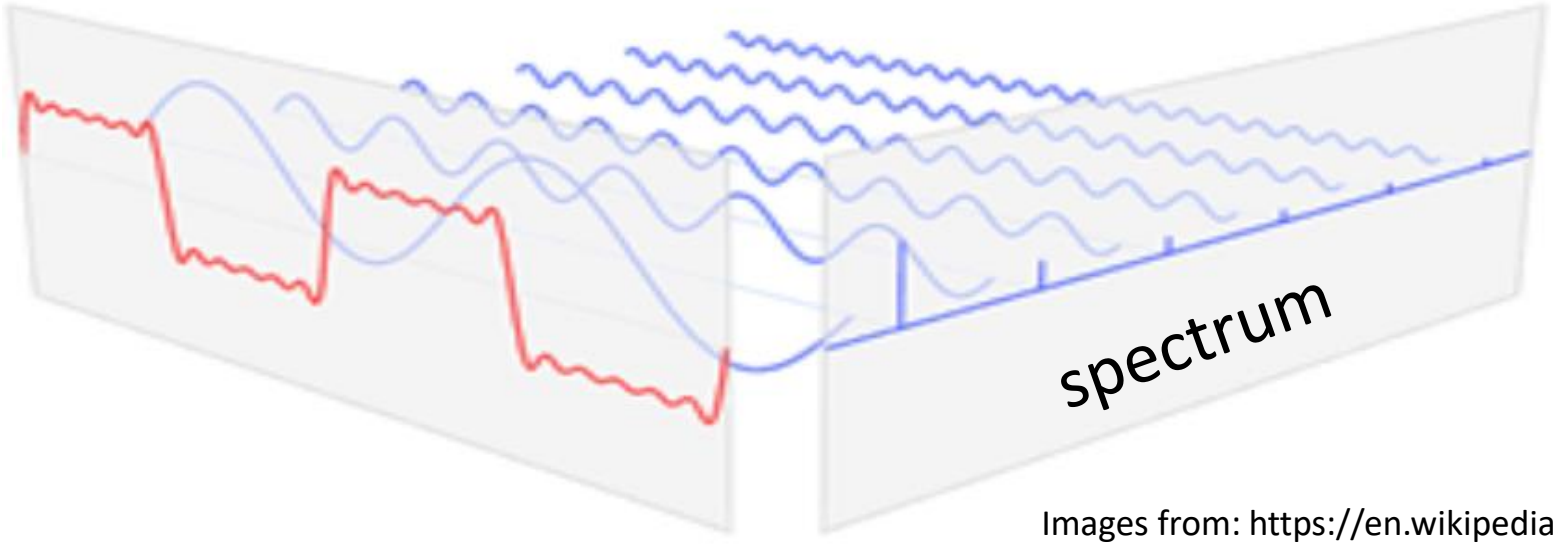
- Convolution manipulates the image spectrum
  - Enhancing/suppressing frequency bands in image.

# Recall the Fourier transform

A signal is represented as a sum of sines/cosines of various frequencies



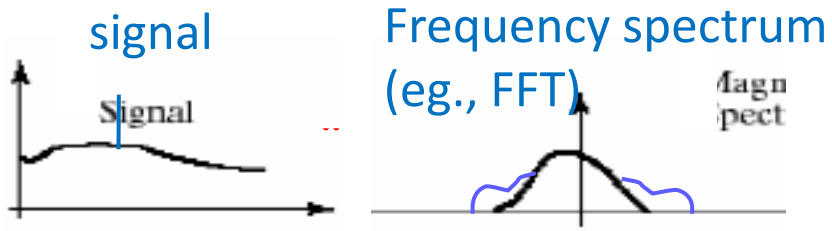
$$f(x) = \sum_n a_n \cos(nx) + b_n \sin(nx)$$




Images from: https://en.wikipedia.org/wiki/Fourier_transform

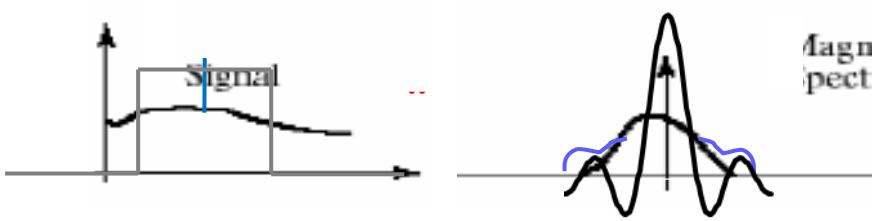
# Convolution: removing noise

- Noise corresponds to adding high frequencies. To remove these, we apply a *low-band pass* filter.

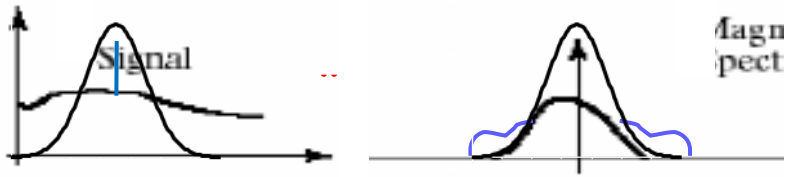



- The spatial box filter transforms to a sinc in frequency space, causing artefacts (side lobes).

$$\mathcal{F}(f * g) = \mathcal{F}(f) \odot \mathcal{F}(g)$$




- A Gaussian maintains a compact support in both image and frequency space. Hence, it's more appropriate as a low-band-pass filter.

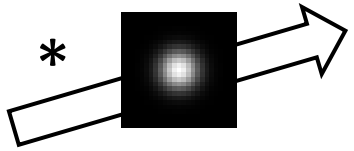



See Forsyth, Ponce: Computer vision, a modern approach.

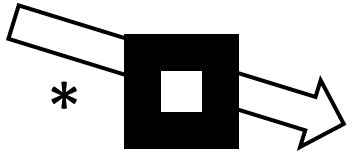
# Strange artefacts in convolution results…

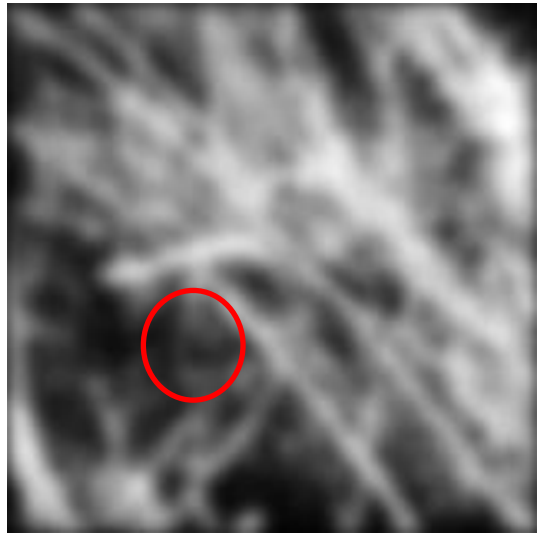
*Original*

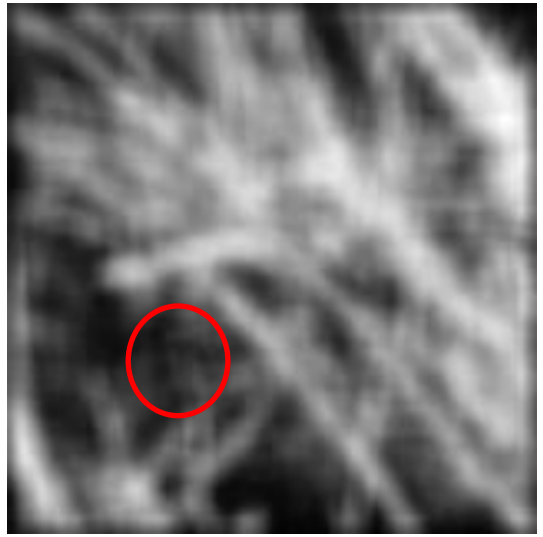
Gaussian



Uniform





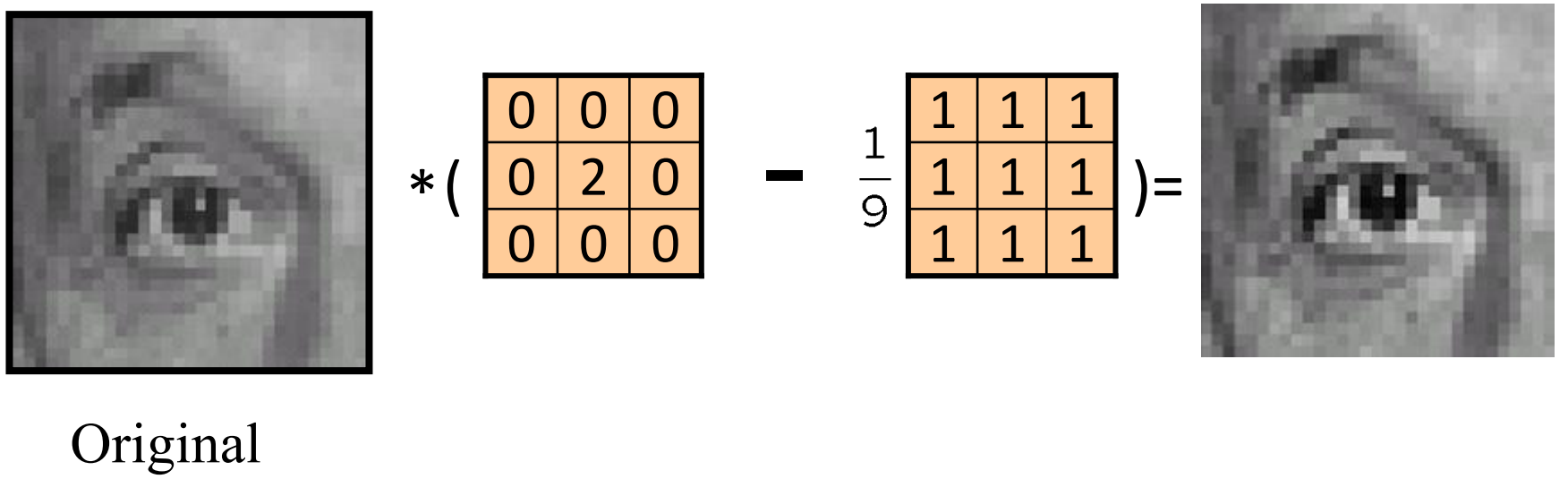


*Filtered*

Filter does not introduce high frequencies

Filter introduces high frequencies

# Linear filters in practice




Original

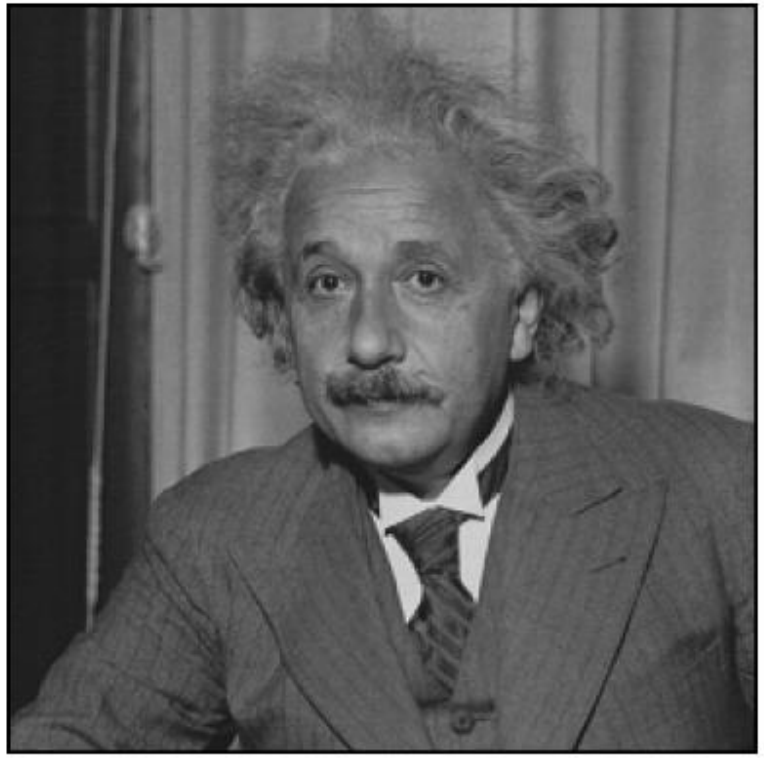
Sharpening filter:

Enhances differences by local averaging.

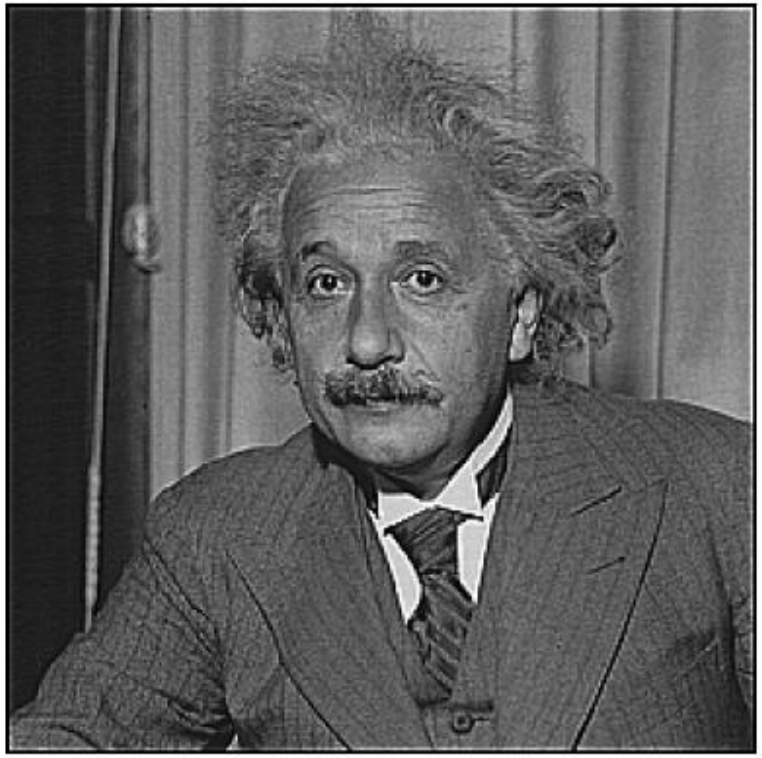
To explain this, think about what happens in te frequency domain.

# Sharpening filter

before



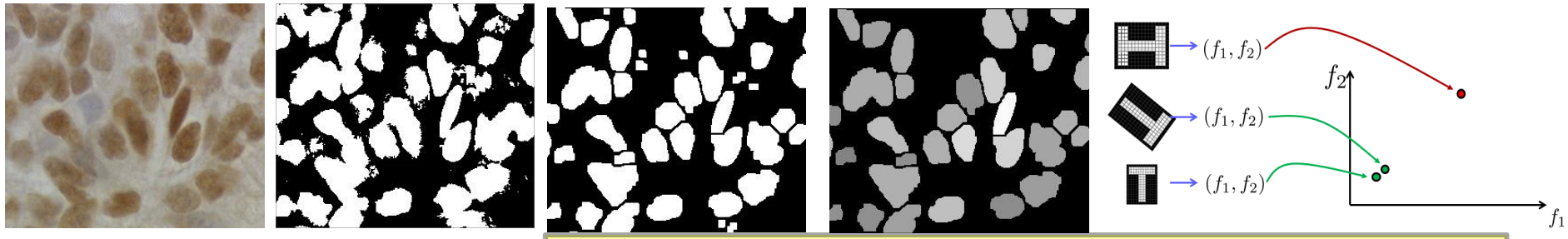
after



To explain this, think about what happens in frequency domain.

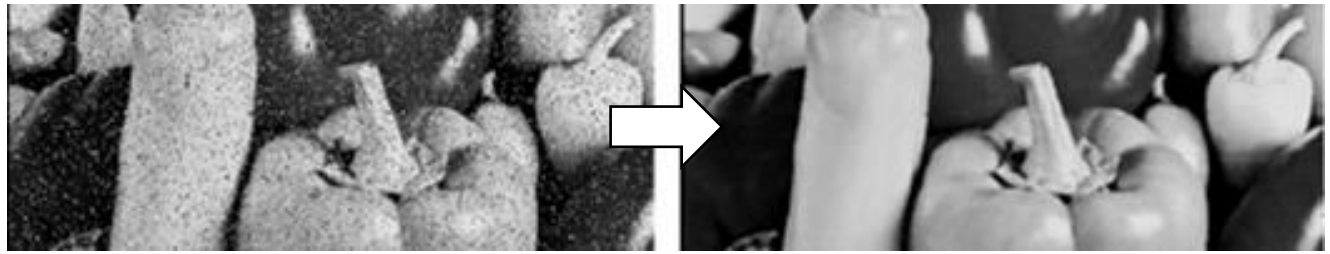
# Previously at MP…

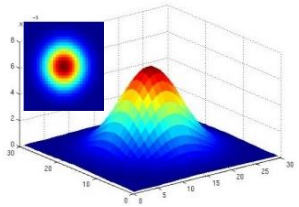
- Basic image processing techniques




Thresholding | Morphology | Region labeling | Region descriptors

- Linear filters: convolution, correlation





$F[x, y]$

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

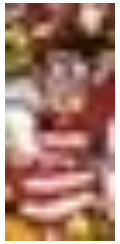
$G[x, y]$

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

Convolution <->Correlation = Dot product

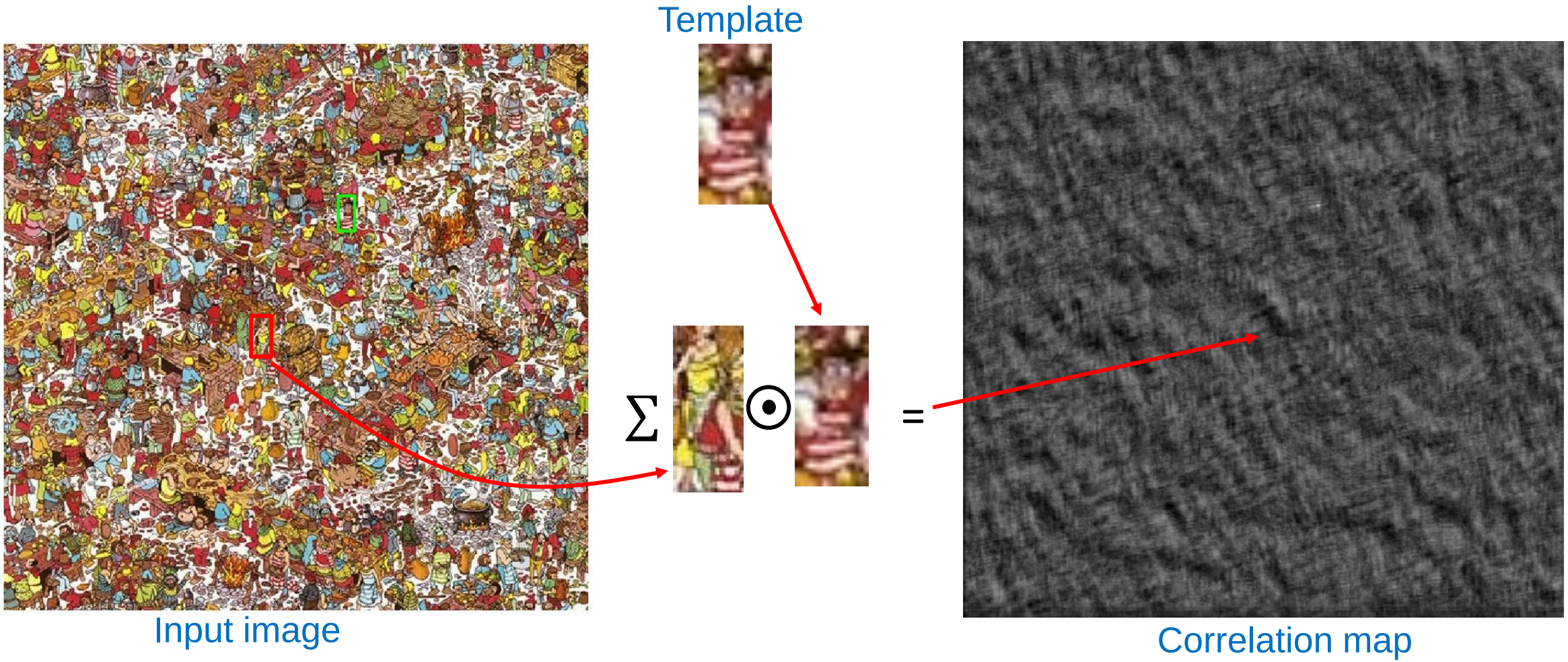
# Filtering as template matching

***Where's waldo?***



Template

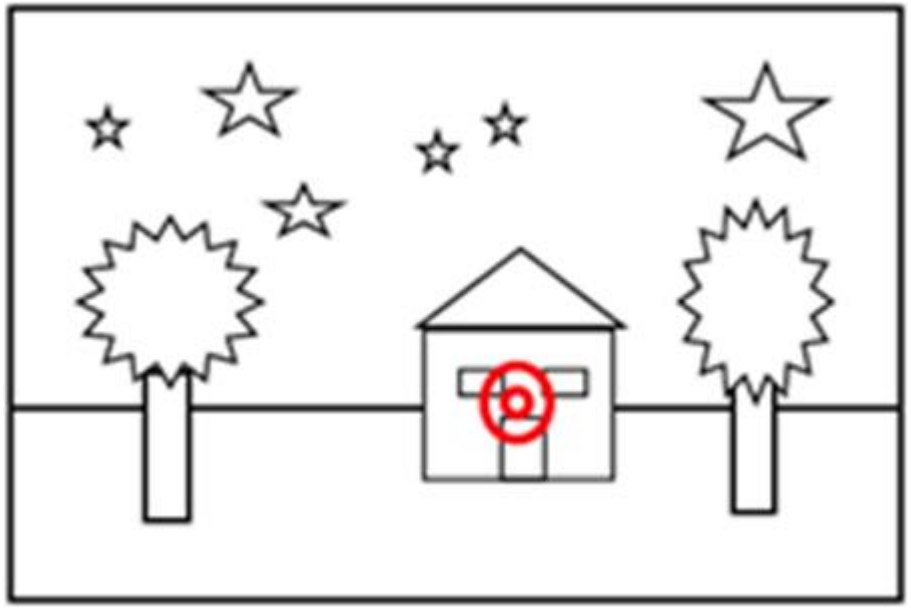
# Apply correlation with template




Convolution <->Correlation = Dot product = Measure of similarity

# Issues with template matching over scales

- But the object may be bigger/smaller in the image!
- Well, we could carry out correlation for different scales of the template...



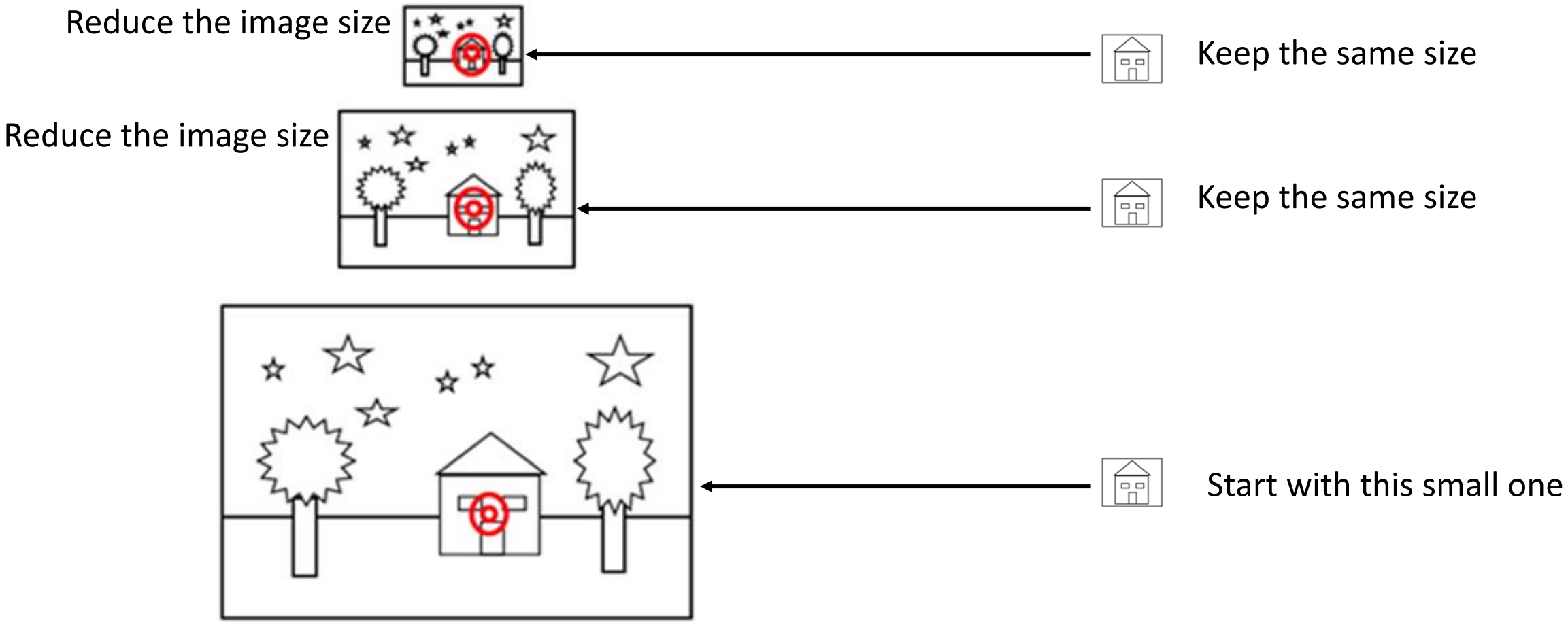
etc. …

Then with this one

Start with this small one

# Template matching in scale space

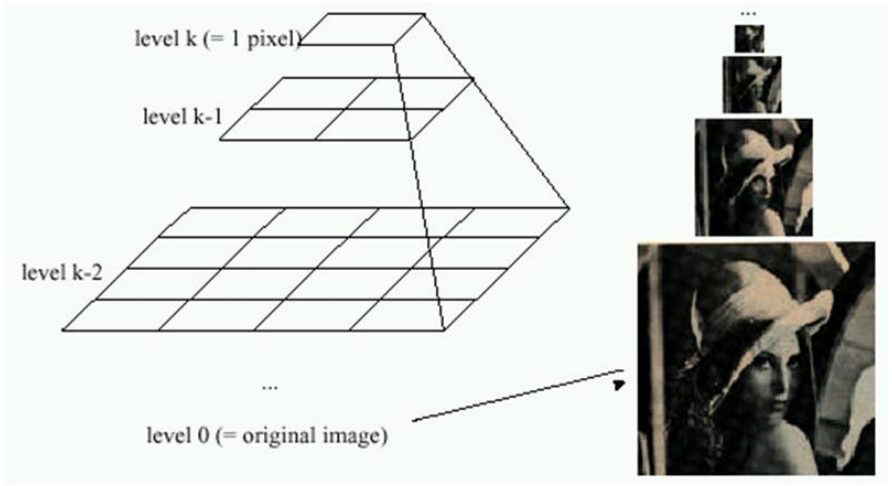
- But rather than template, we scale the input image



# Efficient resizing: Image pyramids

Low resolution

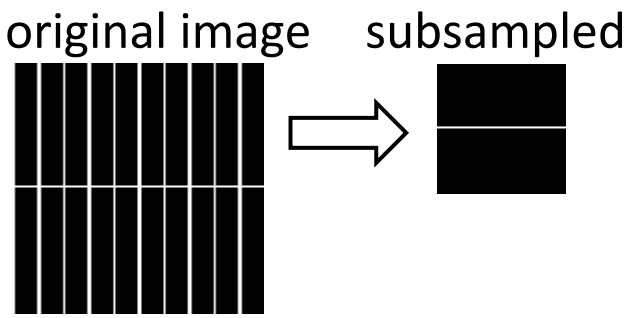
High resolution




*Reduce (resample) the image!*

# How do we reduce an image?

- Naive:
  - Remove every second pixel...

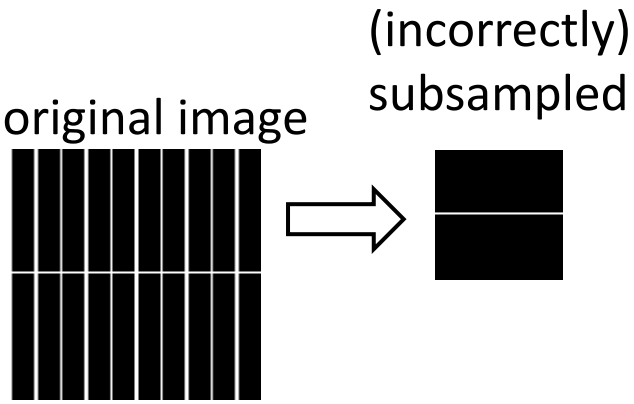



  - Problem: *the structures in image change!*
- This effect is called *Aliasing.*
- *Look into frequency domain* to explain this (Forsyth-Ponce Book)

# Avoiding aliasing

- Nyquist theorem:
  - If we want to reconstruct all frequencies up to *f*, we have to sample the signal by at least a frequency equal to *2f*.

- Meaning: we cannot reconstruct some of the high frequencies when subsampling!



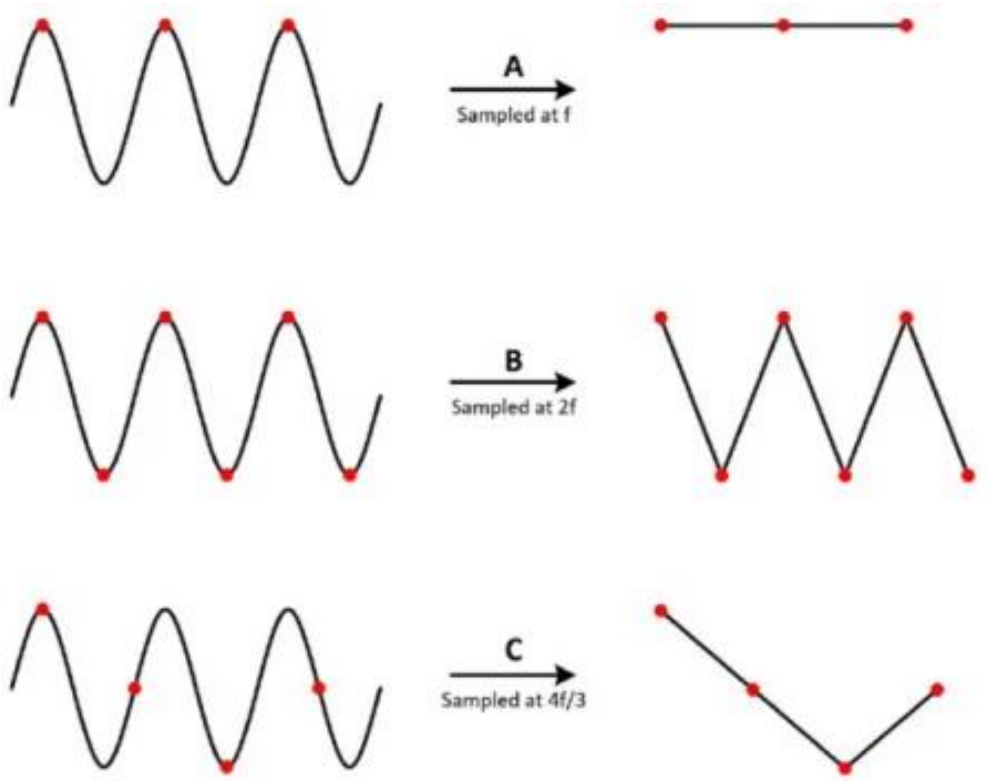
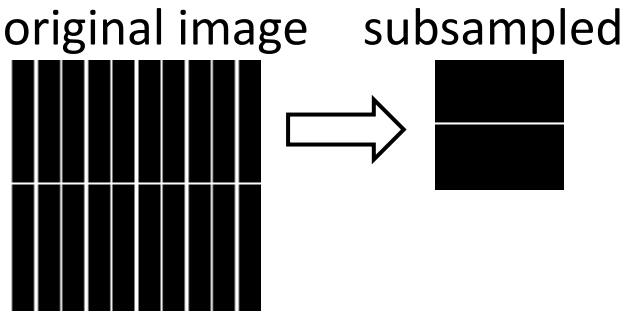




Image source link

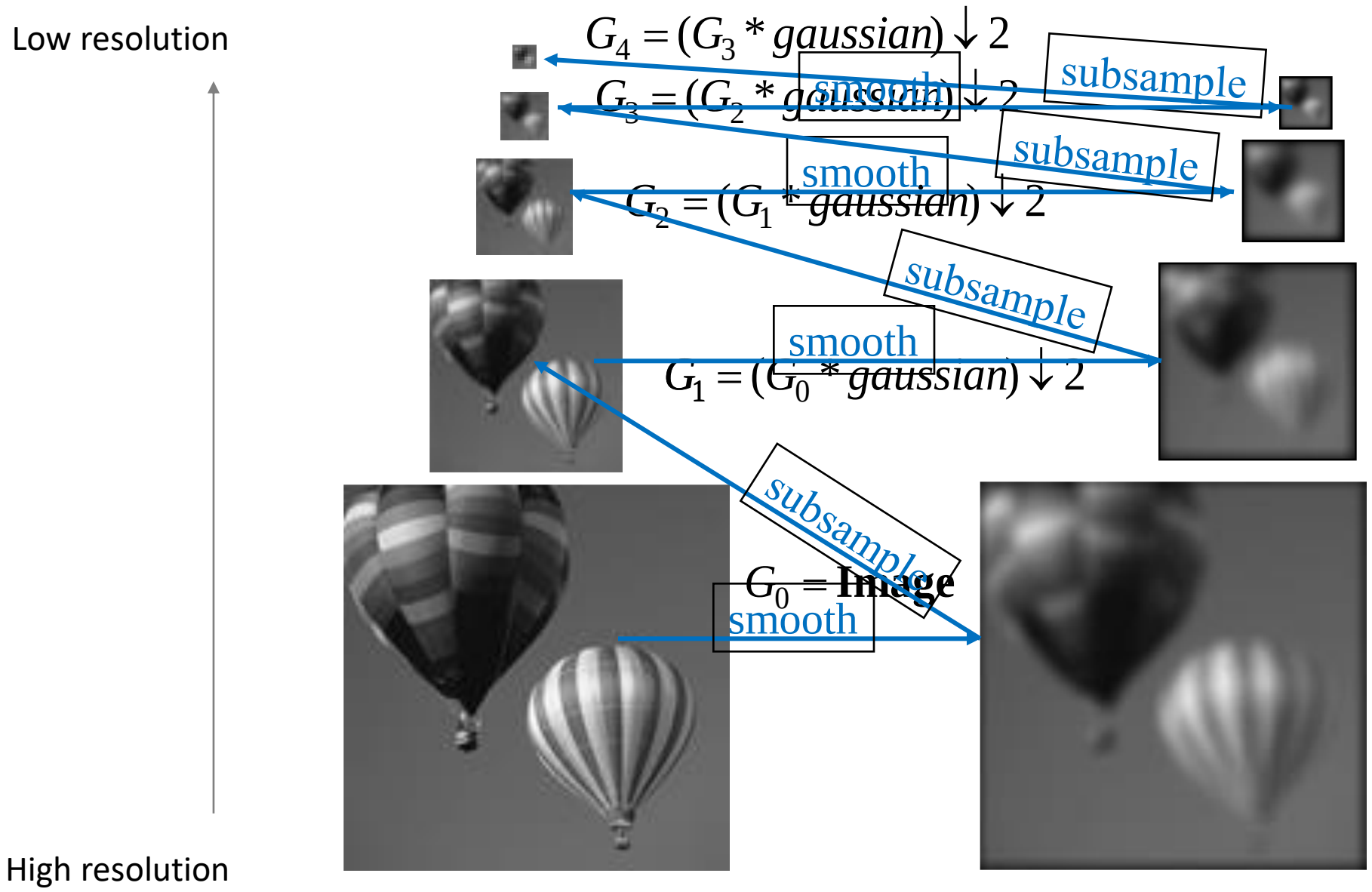
# Avoiding aliasing

- Nyquist theorem:
  - If we want to reconstruct all frequencies up to *f*, we have to sample the signal by at least a frequency equal to *2f*.
- Meaning: we cannot reconstruct some of the high frequencies when subsampling!




- Solution:
  - Remove the high frequencies that cannot be reconstructed, then subsample.
  - How to remove? Blur the image.

# Gaussian pyramid



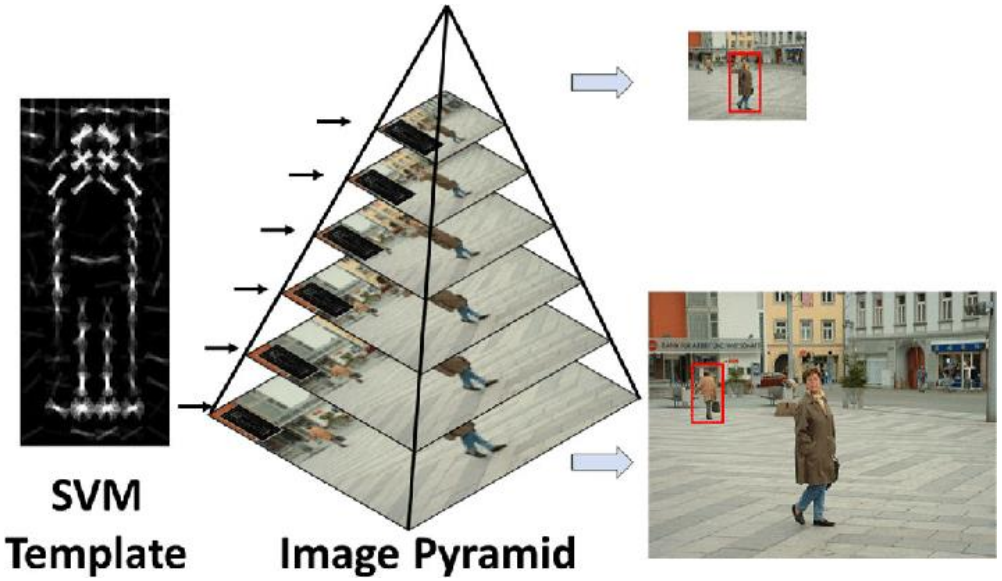
# Summary: Gaussian pyramid

- Construction: get a new level directly from the previous
  - Smooth by a small filter and resample
- Reasons for Gaussian smoothing...
  - Convolution Gauss*Gauss = new Gauss
  - $G(\sigma_1^2) * G(\sigma_2^2) = G(\sigma_1^2 + \sigma_2^2)$
- Reason for size reduction...
  - Gaussian is a low-band-pass filter, so we get a redundant representation of a smoothed image.

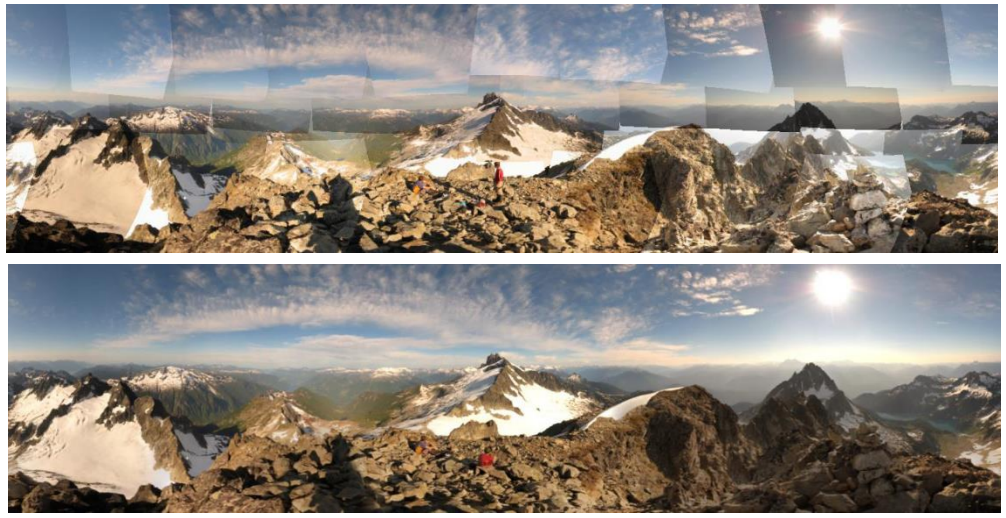
$\Rightarrow$ No need to store a smoothed image in full resolution.

# Image/Feature Pyramids widely applicable

- Enables efficient implementation of many (even) modern methods
  - Multi-scale object detection …
  - Multi-scale edge detection …
  - Multi-scale feature point detection …
- Manipulation of selected frequency bands …

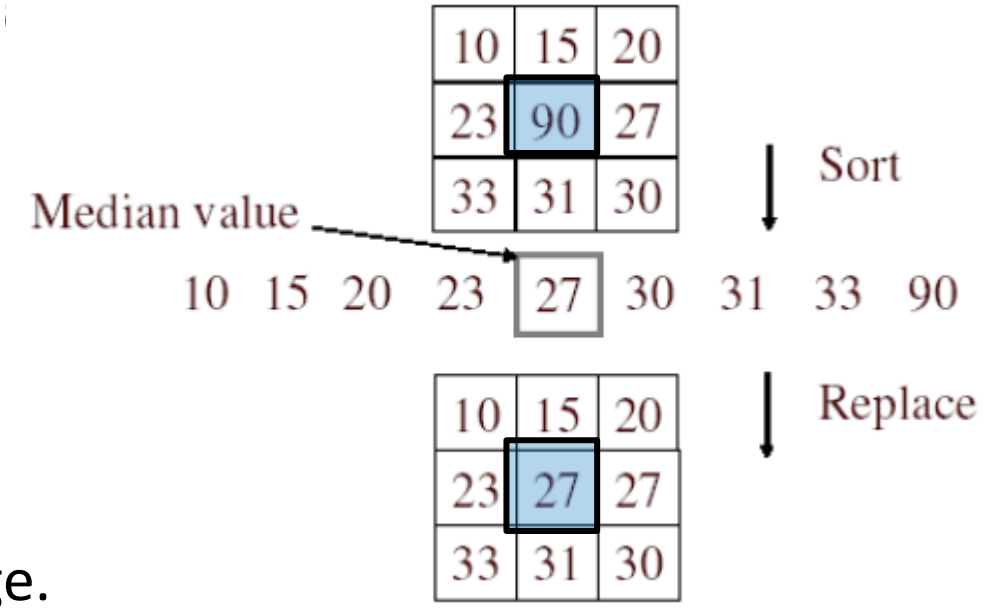



Suleiman and Sze, JSPS2026



# Nonlinear filters: Median filter

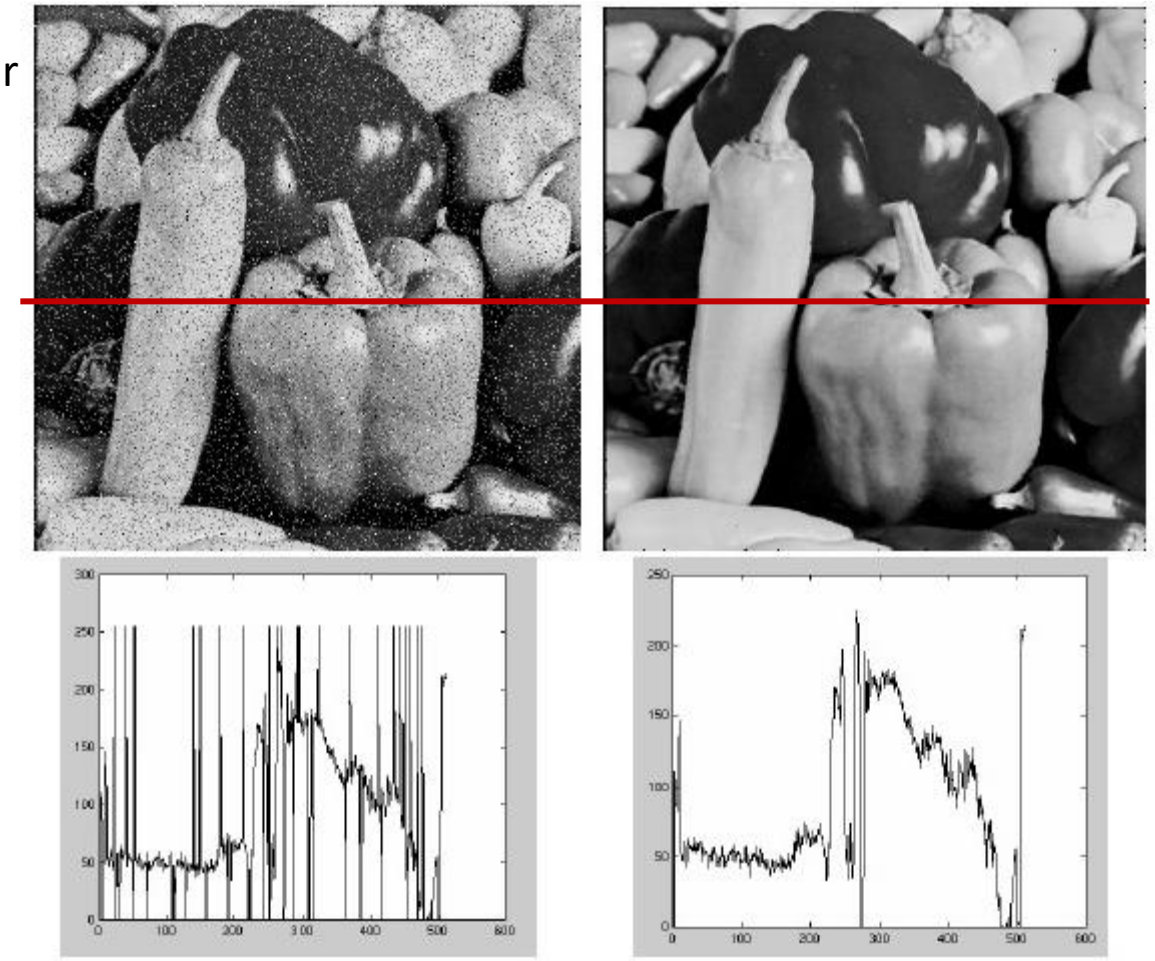
- Basic idea
  - Replace the pixel intensity by a median of intensities within a small patch.




- Properties
  - Does not add new gray-levels into the image.
  - Removes outliers: appropriate for impulse noise and salt&pepper noise removal.

# The Median filter

Salt&pepper noise




After median filtering

Plot of a line in the image

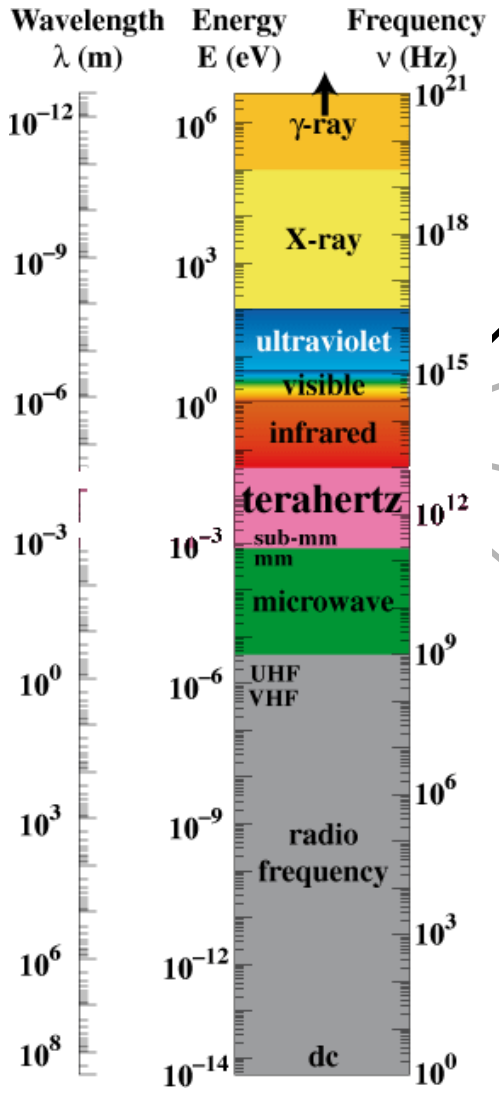
# Median vs. Gaussian

Machine perception

# COLOR

# Sensor: Camera

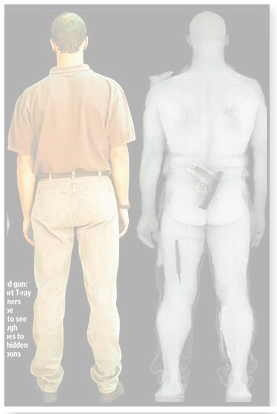
## Electromagnetic spectrum




**Visible light**

Near-infrared light

Far-infrared light



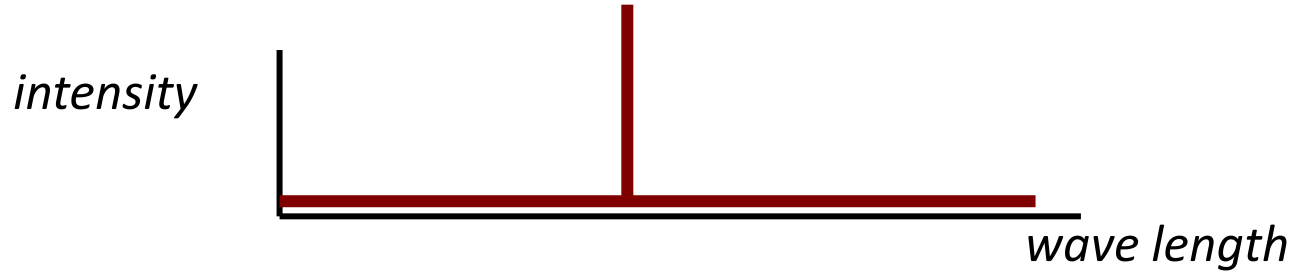
Terahertz light

http://userweb.elec.gla.ac.uk/d/dpaul/terahertz.html

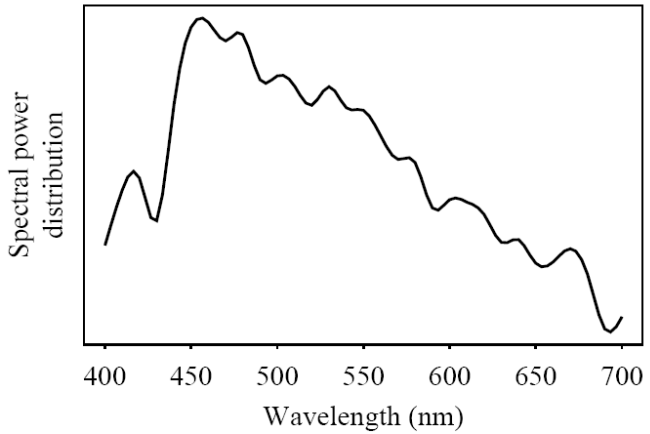
# Light

- *Light is an electromagnetic radiation composed of several frequencies*
- Properties are described by its spectrum (i.e., how much of each frequency is present)

  *Energy radiated in a unit of time w.r.t. wavelength*
- E.g., laser light contains only a narrow band of wavelengths (frequencies)

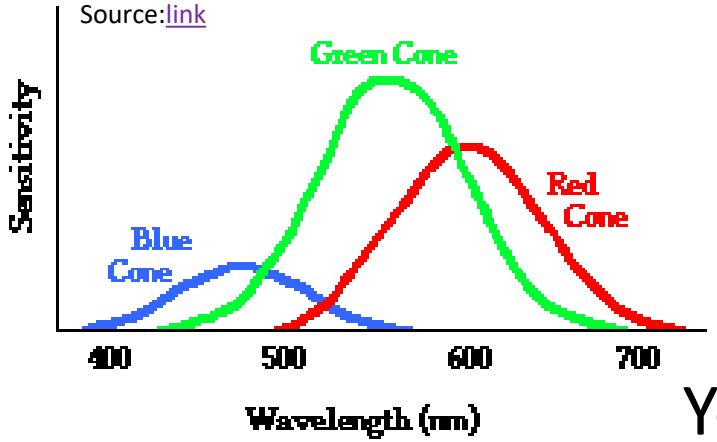



- Visible light contains radiation with wavelengths in interval 400-700nm

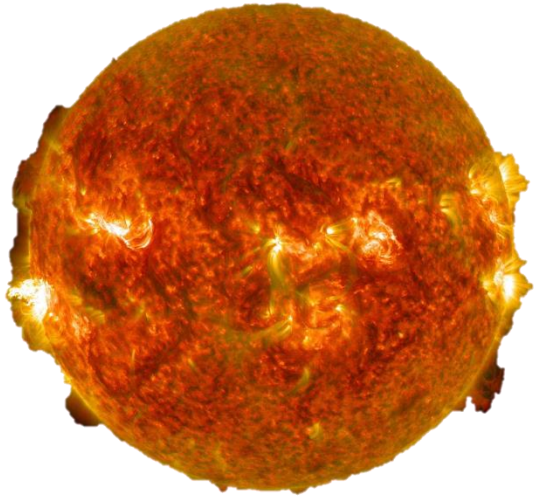


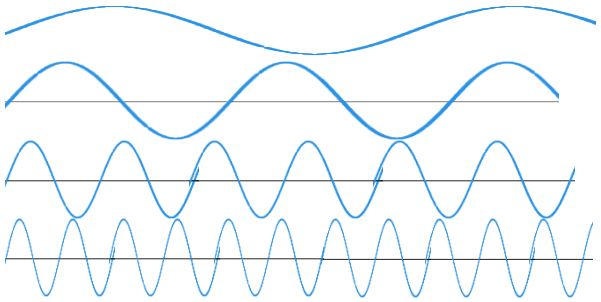
# Human color perception

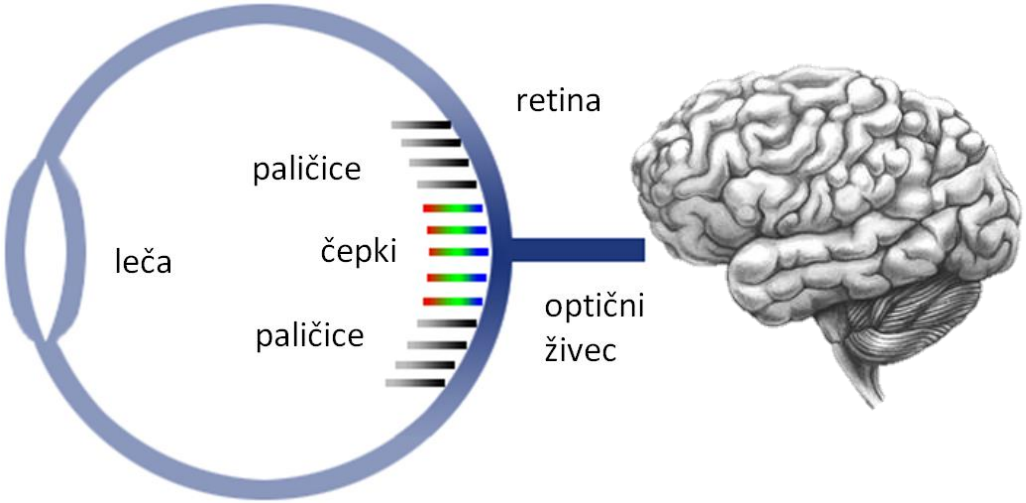
- Human eye (retina) contains specialized cells that react to different wavelengths differently.
- Three types of cells called “cones”: R, G, B
- A type of cells called “rods”: intensity only




Yellow

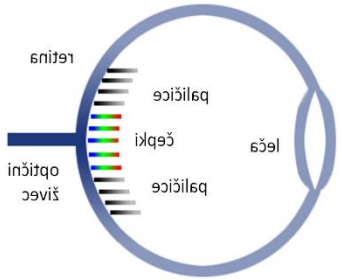




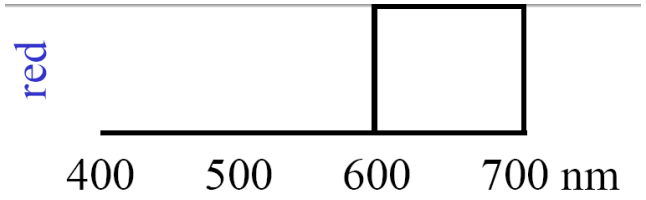


# Additive mixture model

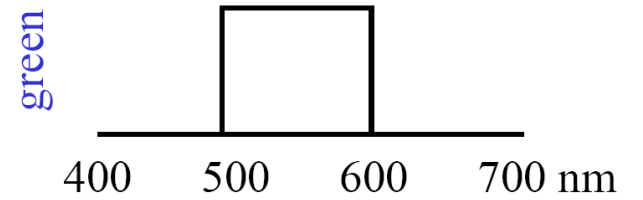
- What color do we get if we shine a red and green light to a white sheet of paper?



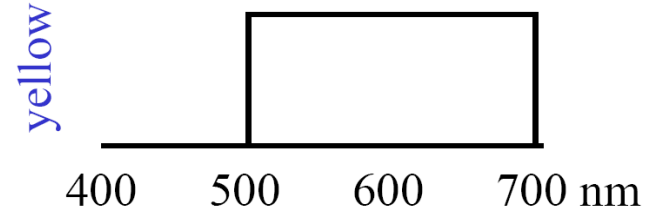
perceive



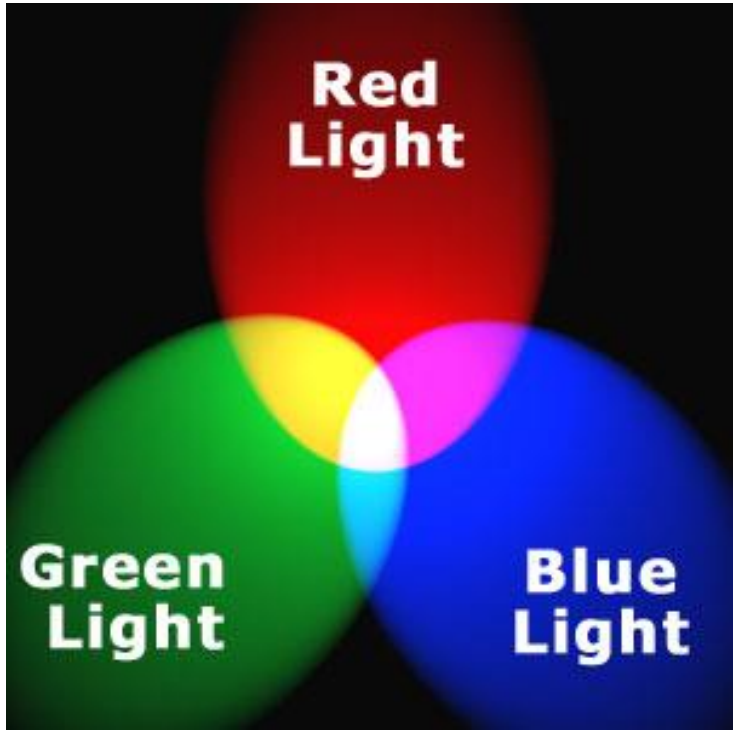

perceive




perceive



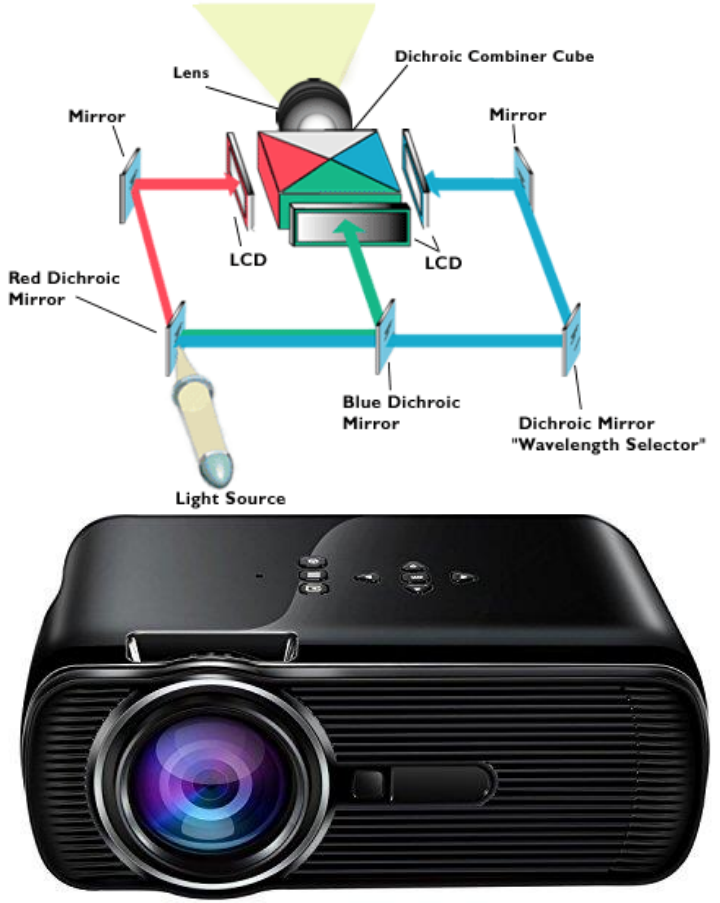

colors mix by summation of their spectra.




colors *added to black*.

# Systems using the additive model

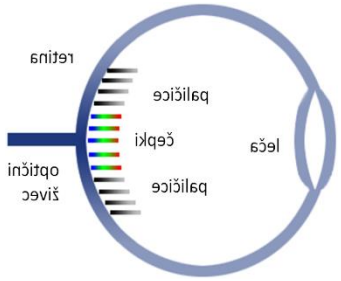
Monitors

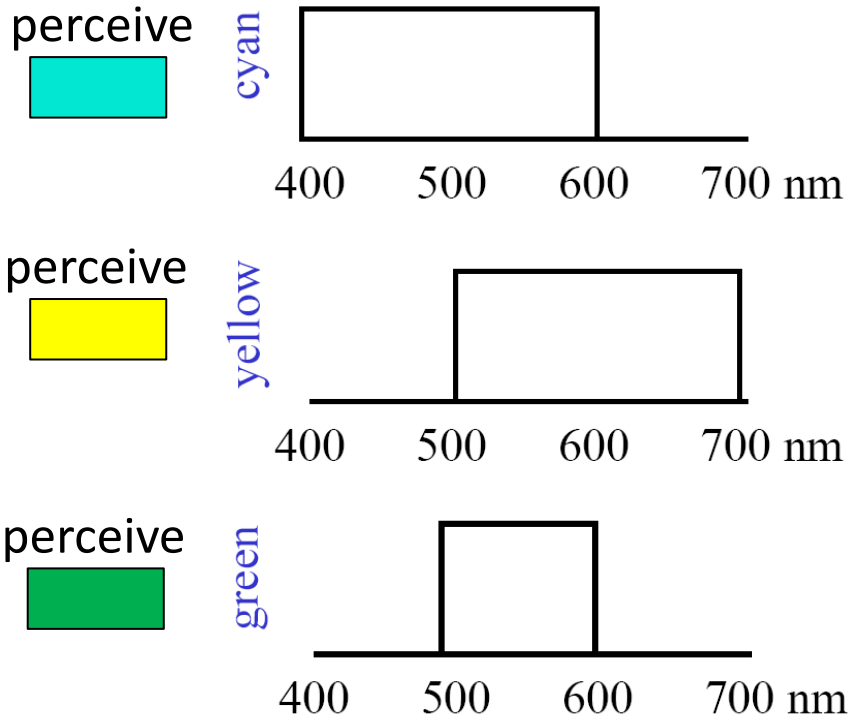



LCD projector

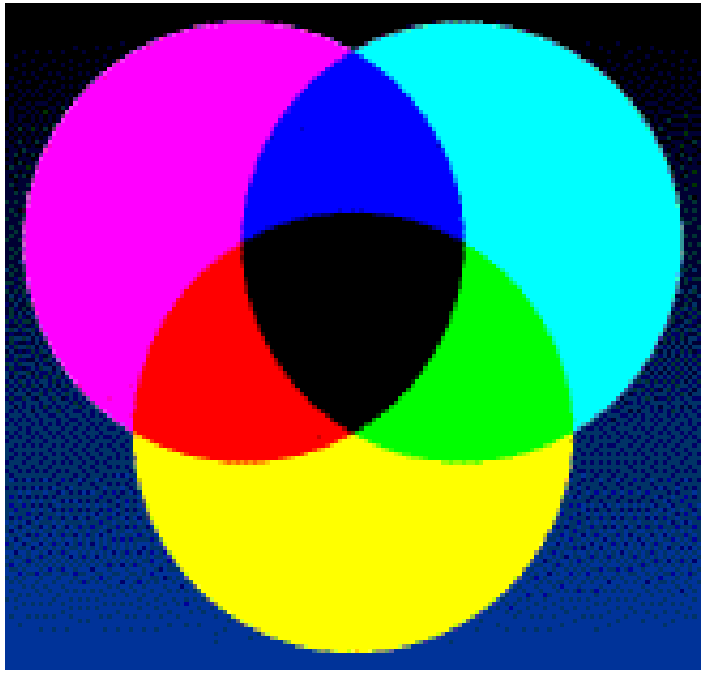
# Subtractive models

- What color do we get if we paint some cyan and yellow pigment to white paper?






colors mix by *spectra intersection*.



Pigments *remove* the color from the incident white light.

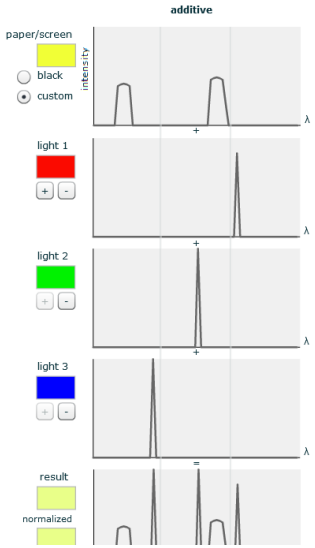
# Systems using a subtractive model

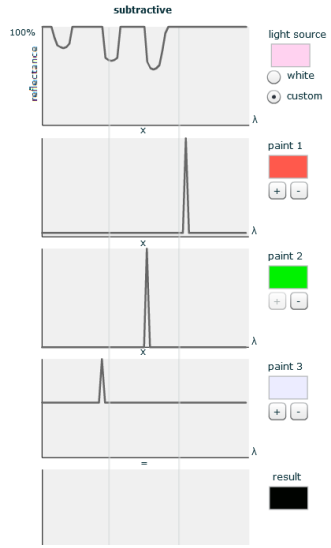
- Printing on paper
- Crayons
- Photographic film




- See this nice app and play with setups:

  https://graphics.stanford.edu/courses/cs178/applets/colormixing.html



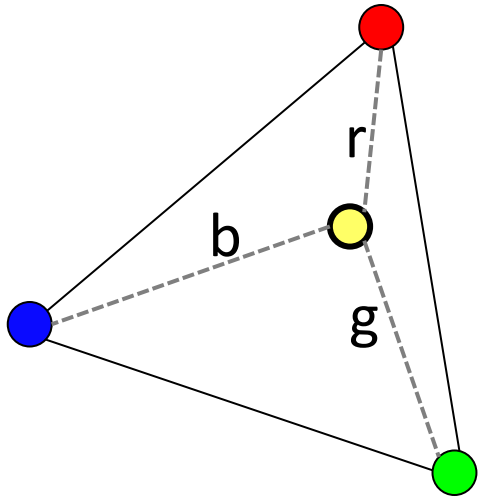



# Color spaces

- Role of color space: Unique color specification (e.g., for reproduction)
  - *Specifying a color in a color space allows accurate color reproduction on various media like photo, print and monitor.*
- Defined by the choice of primary colors *(primaries)*

  Recall: The human eye is equipped with sensory cells for the perception of the three primary colors (RGB)
- A new color is a weighted sum of primaries

*By mixing the colours, we get any colour that lies within the triangle of primaries.*

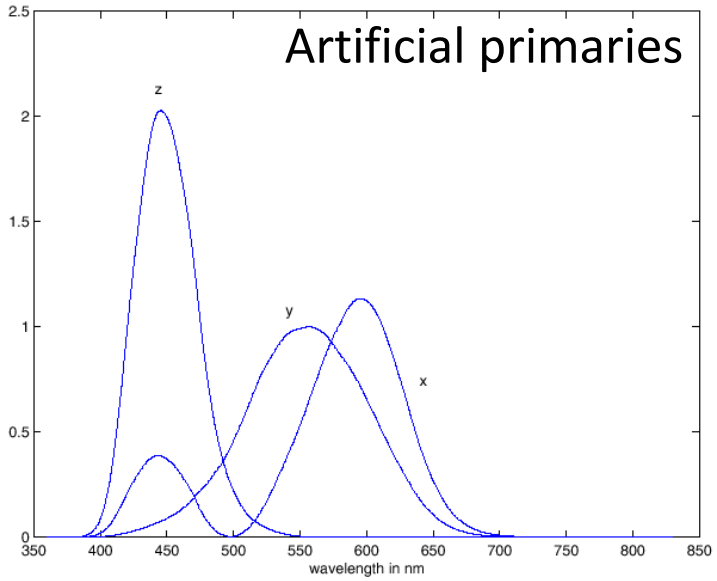


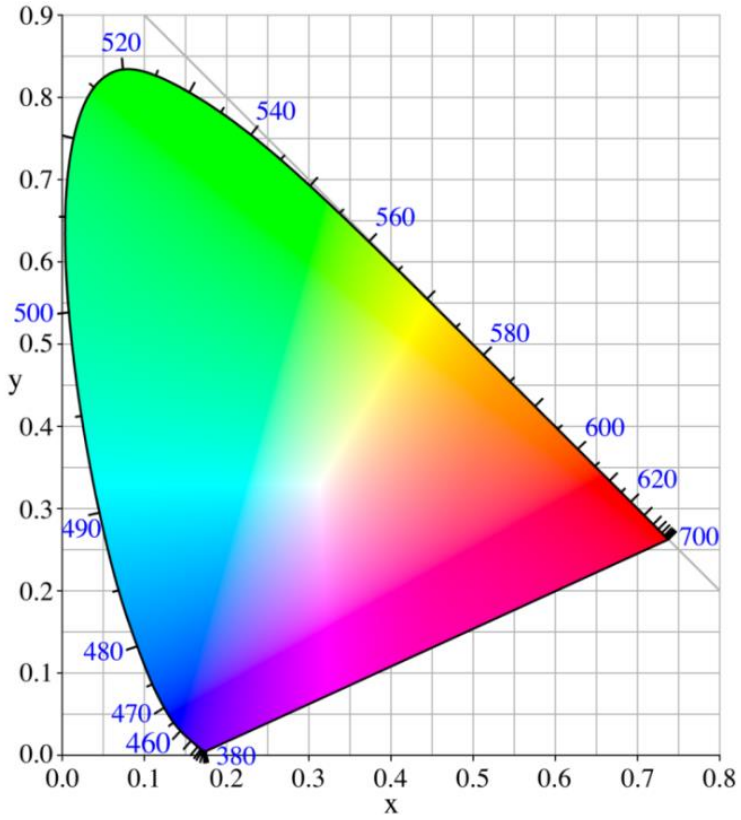

- Mixing weights *r,g,b* to get any color were estimated on human subjects

# Linear color space example: CIE XYZ

- International Commission on Illumination (Commission international d'eclairage -- CIE), 1931

A nice overview available: https://www.sciencedirect.com/topics/engineering/color-matching-function




- Representation by chromaticity only [x,y]:

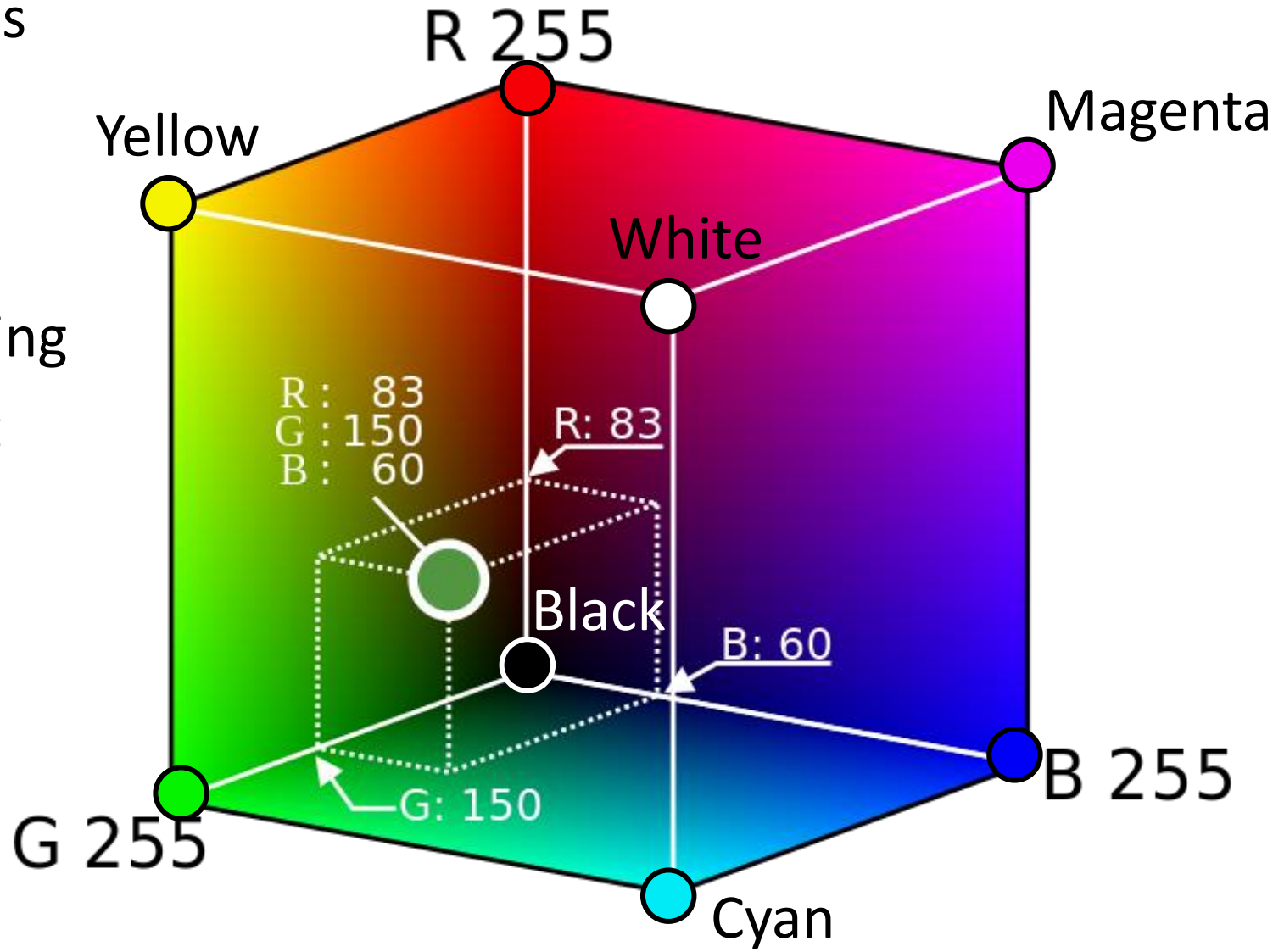
$$x = \frac{X}{X+Y+Z}, \; y = \frac{Y}{X+Y+Z}, \; z = \frac{Z}{X+Y+Z}$$

$$x + y + z = 1$$

# Linear color space example: RGB

- Single wave-length primaries

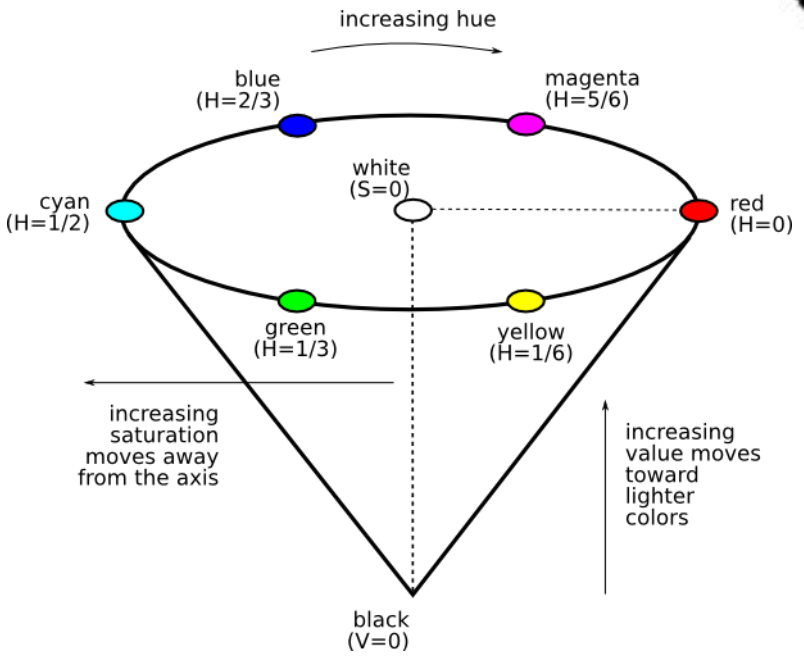
- Appropriate for use in imaging devices (e.g., monitors), but not for human perception

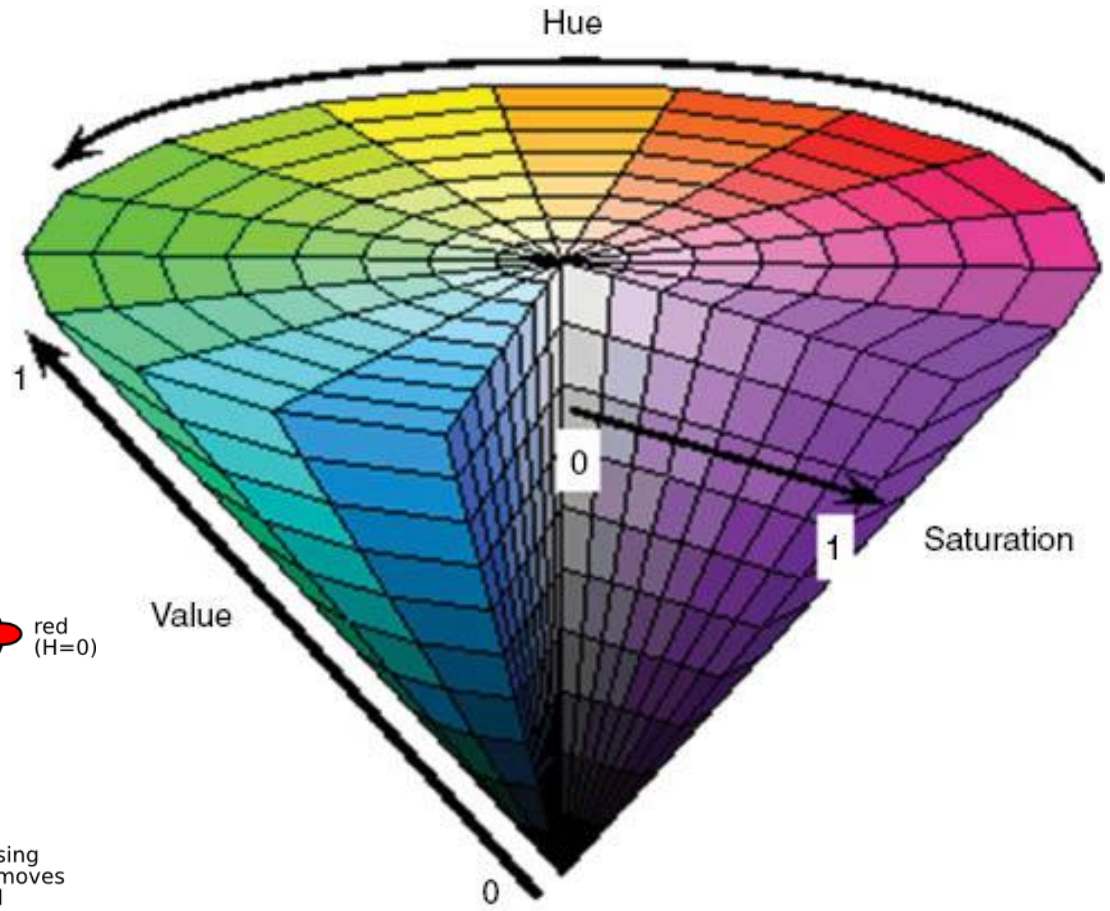


# HSV colorspace

- Hue (barvnost), Saturation (nasičenje), Value (intenziteta)
- Nonlinear – hue coded by angle
- Python: cv2.cvtColor(*I*, *Type*)

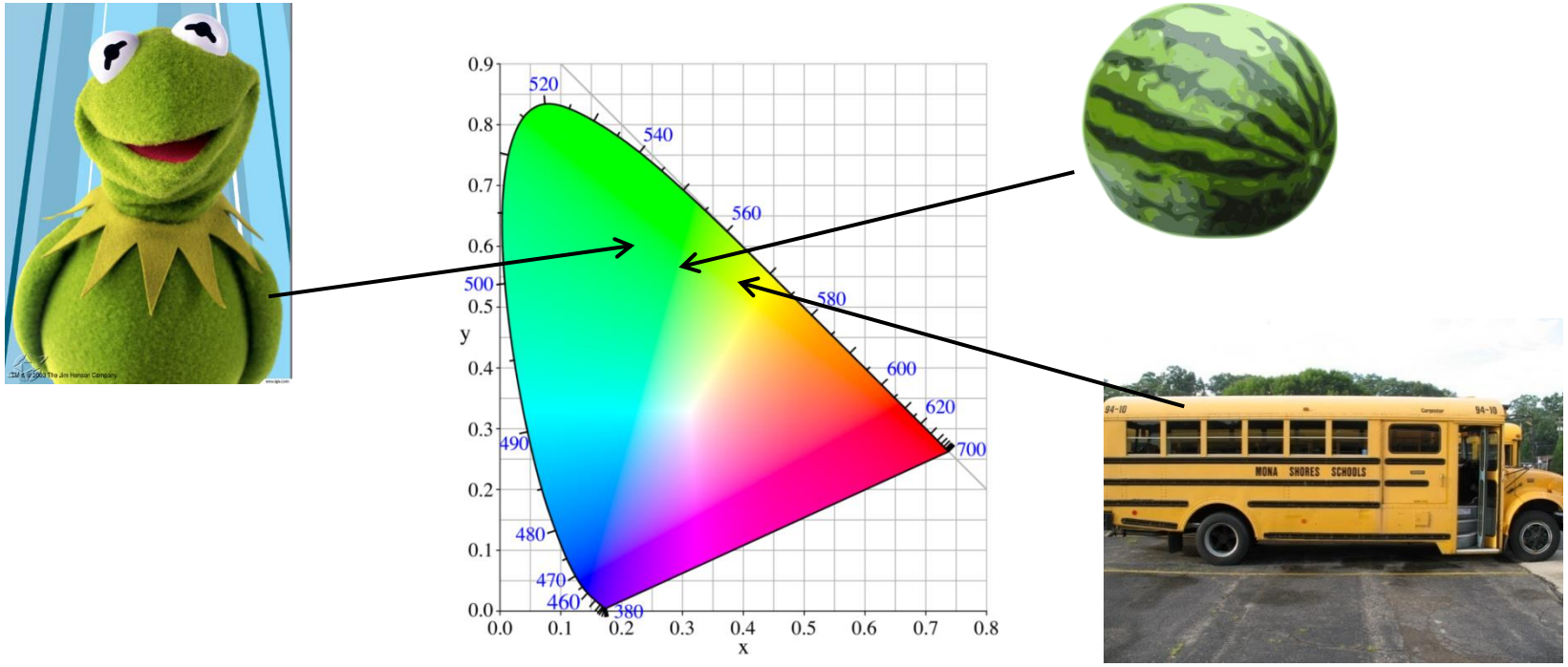
  Type:
  - cv2.COLOR_RGB2HSV
  - cv2.COLOR_HSV2RGB



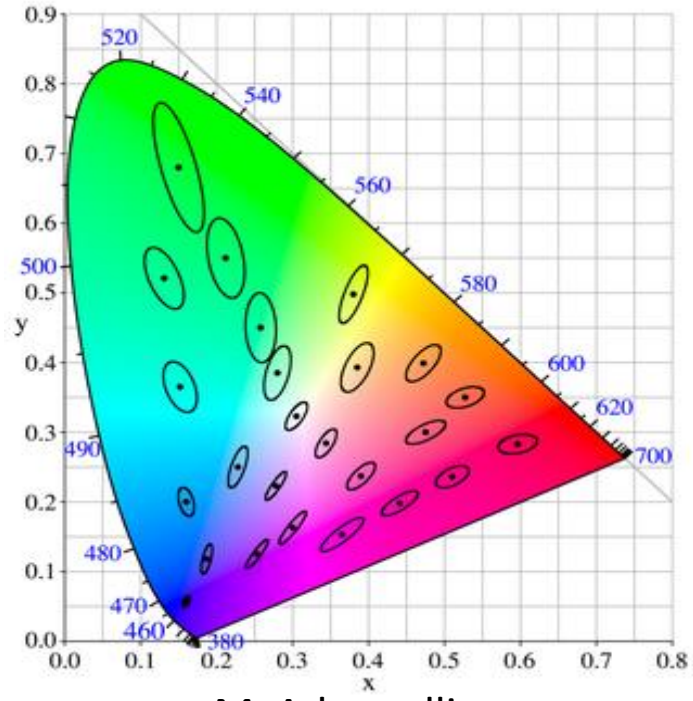
# Distances in colourspaces

- Do distances between points in the colourspace make sense perceptually?



# Distances in color spaces

- Not necessarily: CIE XYZ is nonuniform colorspace – *Euclidean distance between coordinates of colors in colorspace  is not a good indicator of color similarity (in terms of human perception).*



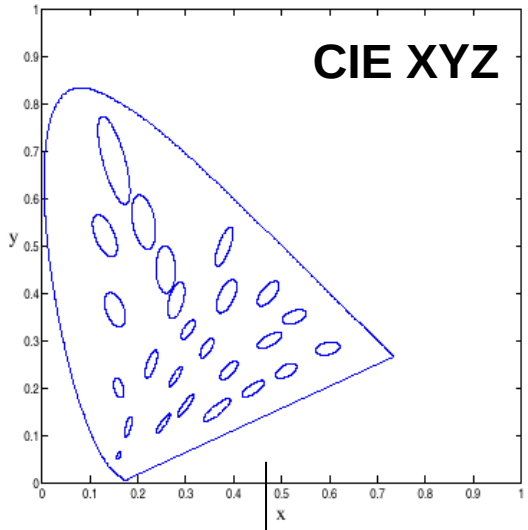

McAdam ellipses:
Just (human) noticeable differences in color

# Uniform color spaces

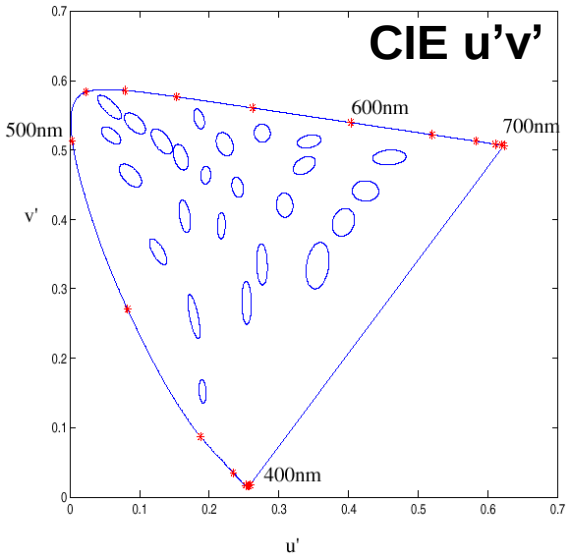
- Transforms such that ellipses are mapped into circles

  → *distances better replicate the human perception of color similarity.*

- Examples of uniform colour spaces:
  - CIE u‘v‘
  - CIE Lab (1976)




Nonuniform colour space




Uniform colour space

# Computing color similarity between objects

- How to summarize the color?
- Idea1: just compute the average (r,g,b)

$\mu_{rgb}$

$$\mu = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$$

$$\mathbf{x}_i = (r_i, g_i, b_i)^T$$

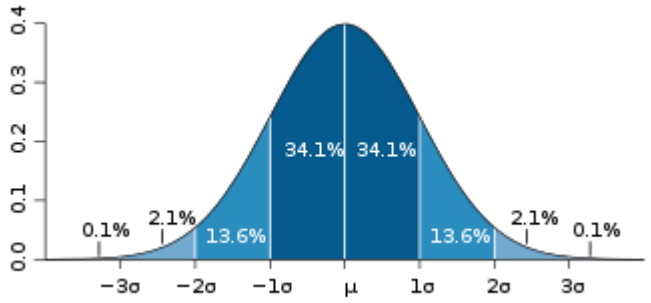
Color of the (r,g,b) at *i*-th pixel

$\mu_{rgb}$

Issue: a single value does not sufficiently capture the color *distribution*

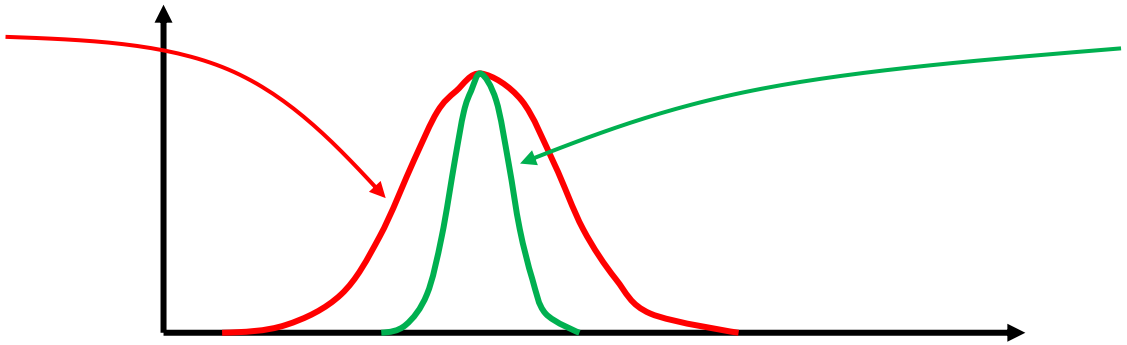
# Describe the color by a Gaussian

- Summarize the color by parameters of a Gaussian distribution




$$\mu = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$$

$$\mathbf{\Sigma} = \frac{1}{N-1}\sum_{i=1}^{N}(\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T$$



But often a more flexible model of color distribution is required!

Machine perception

# COLOR DESCRIPTION BY USING HISTOGRAMS

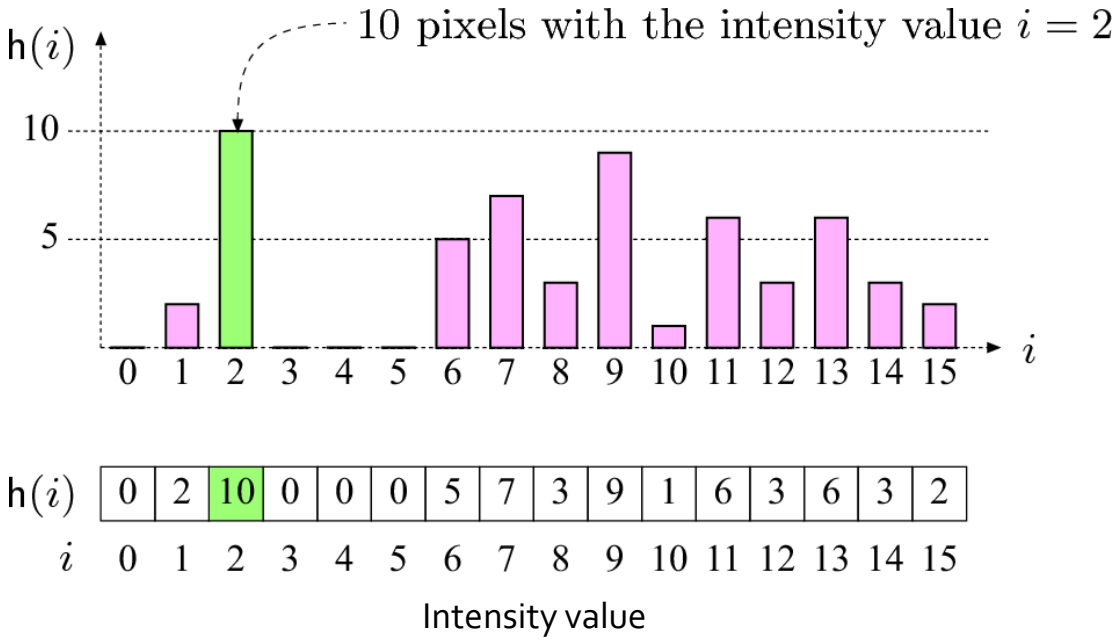
# What is a histogram?

- Image histogram records the frequency of intensity levels

$$\mathrm{h}(i) = \text{ the } \textit{number} \text{ of pixels in } I \text{ with the intensity value } i$$

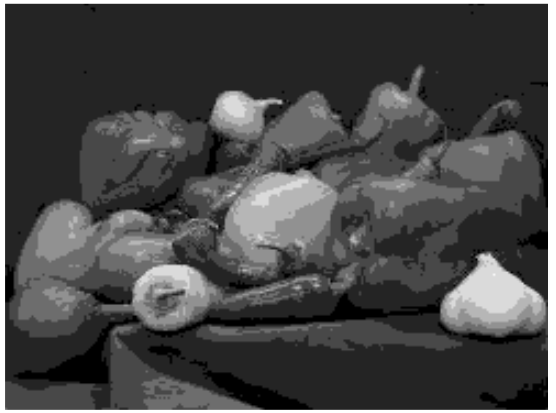
$$\mathrm{h}(i) = \mathrm{card}\{(u,v) \mid I(u,v) = i\}$$

- Example:



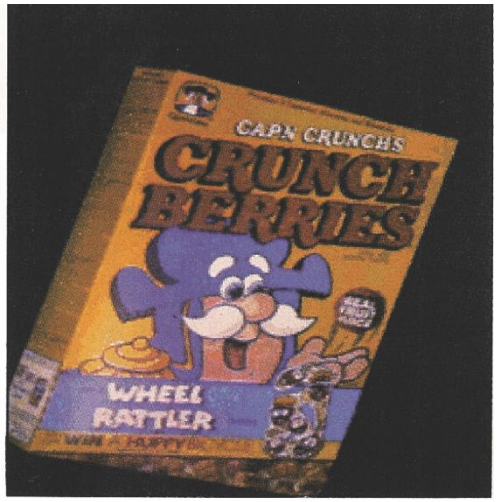

256 intensity levels

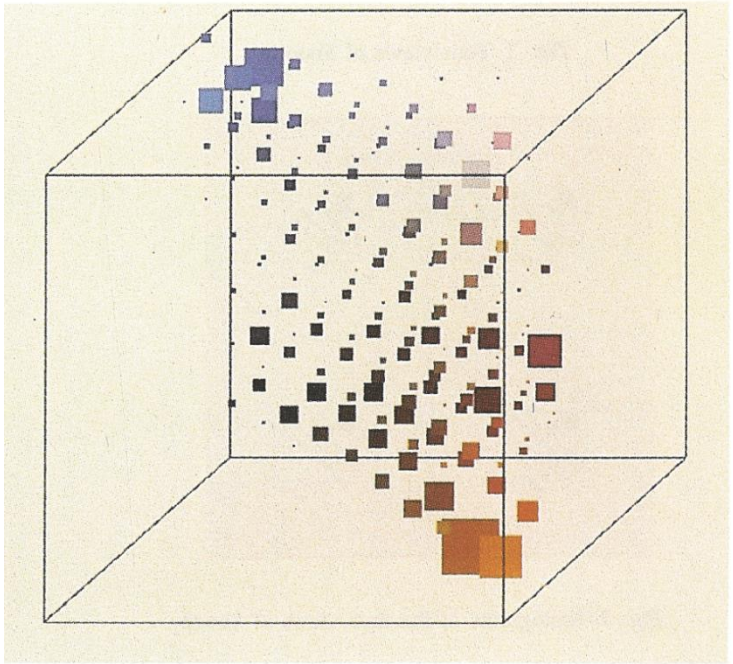
16 intensity levels



# Color histogram

- Color statistic
  - Example of a 3D RGB histogram $H(R, G, B)$ visualization
  - Each pixel color is a point in 3D space (RGB)
  - Calculate the 3D color histogram
    - H(R,G,B) = number of pixels with color [R,G,B]

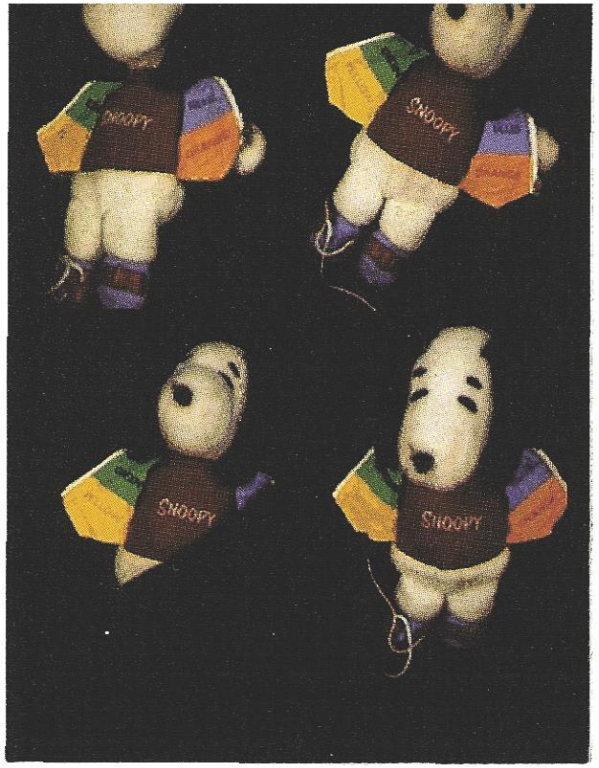


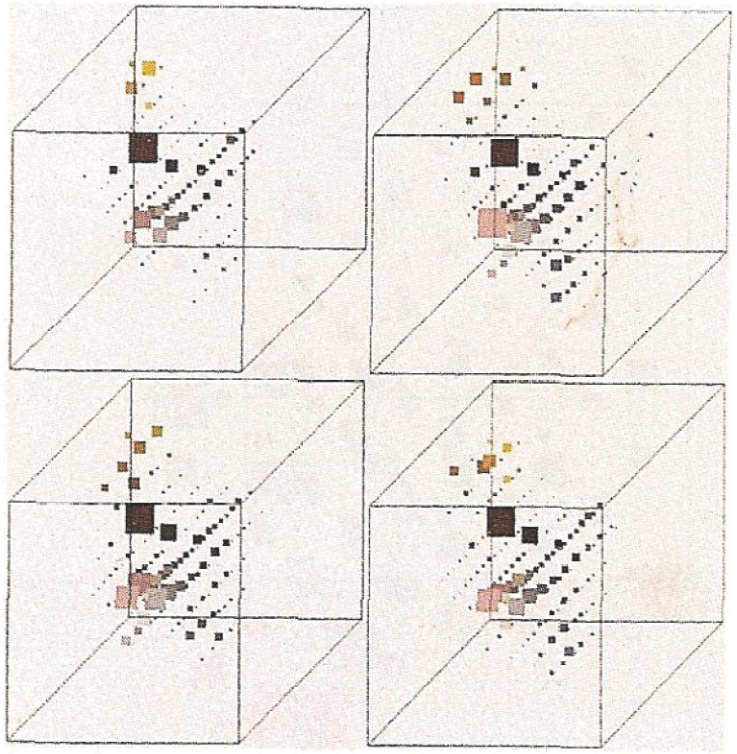



[Swain & Ballard, 1991]

# Color histogram

- Robust representation of images
  - Translation, scale, partial occlusion



[Swain & Ballard, 1991]

# Intensity normalization

- Intensity is contained in each color channel
  - Multiplying a color by a scalar changes the intensity but not the hue („true“ color).
  - This means that we can normalize a color by its intensity.
    - Intensity is defined as: $I = R + G + B$:
  - Chromatic representation:

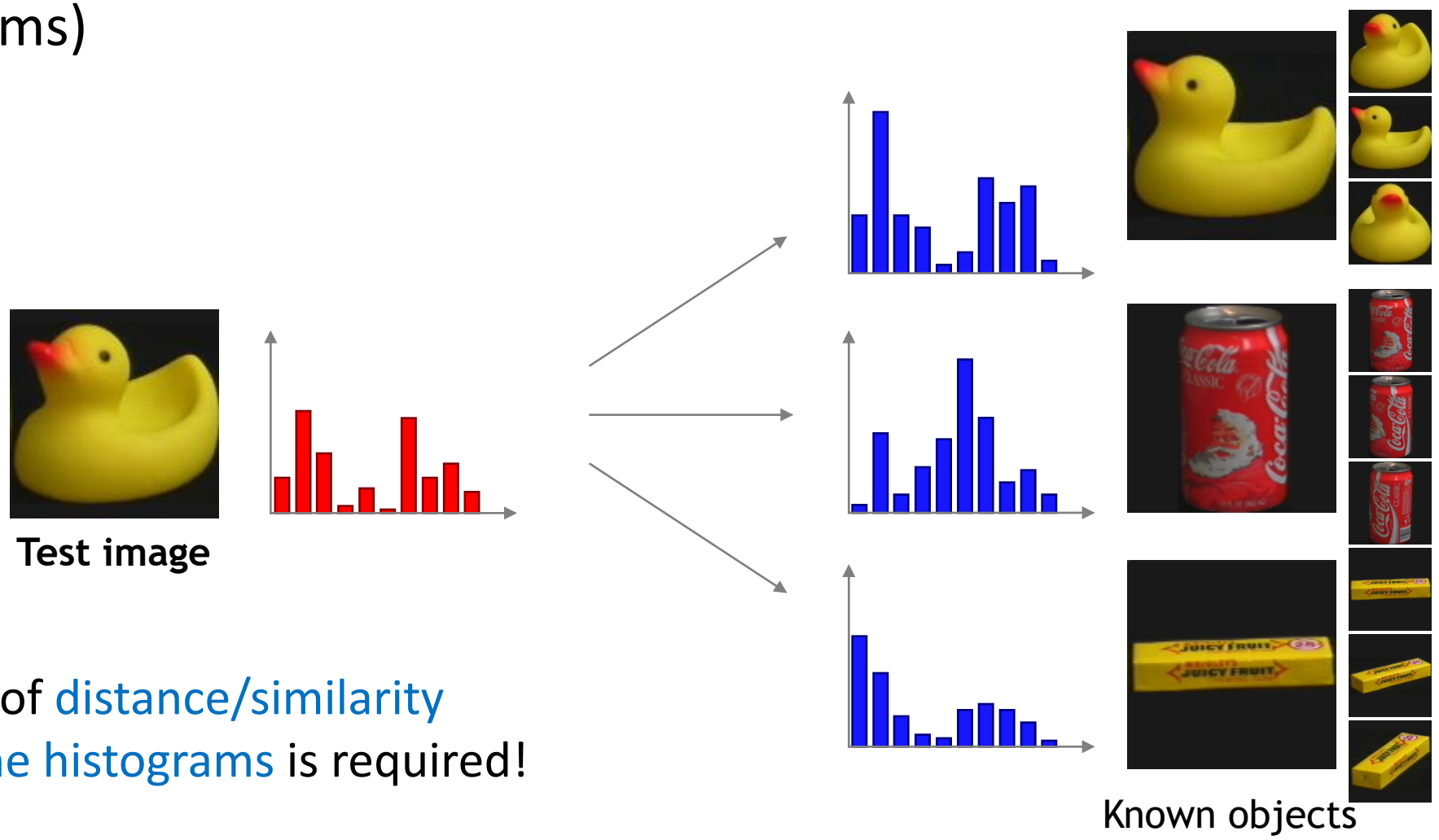
$$r = \frac{R}{R+G+B} \qquad g = \frac{G}{R+G+B} \qquad b = \frac{B}{R+G+B}$$

  - We can now use only a 2D space (rg), since it holds that

$$r + g + b = 1$$

# Color comparison via histograms

- Compare images indirectly – compare only their descriptors (histograms)

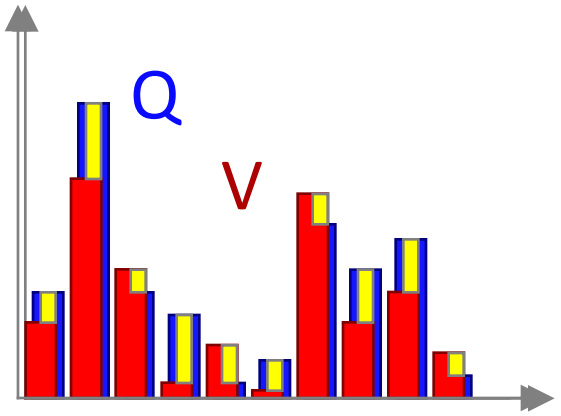



A measure of distance/similarity between the histograms is required!

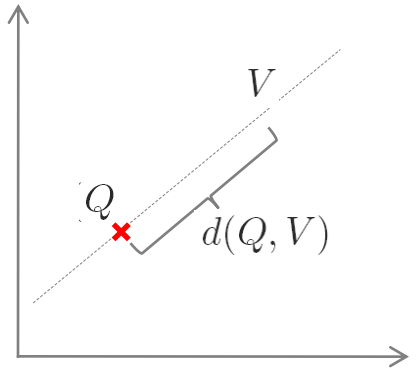
# Popular distances: Euclidean distance

- Definition (=$L_2$ norm)

$$d(Q,V) = \sqrt{\sum_i (q_i - v_i)^2}$$




- Explanation
  - Looks for differences in histogram cells.
  - Interpretation: Distance in feature space.
  - Range of output values: [0,1]
  - All cells receive equal weight.
  - Susceptible to noise!

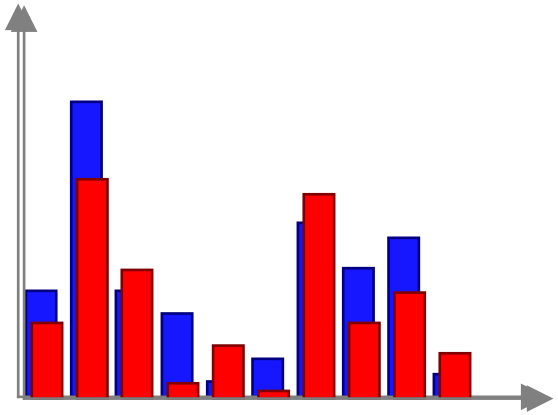


# Popular distances: pdf similarity

- Similarity between two probability density functions
- Chi-squared (slo., hi-kvadrat):

$$\chi^2(Q,V) = \sum_i \frac{(q_i - v_i)^2}{q_i + v_i}$$

WATCH OUT FOR qi=vi=0!!



- Kullback-Leibler divergence:

$$KL(Q,V) = \sum_i q_i \log \frac{q_i}{v_i}$$

Not a proper metric (not symmetric)

Symmetric version (Jeferey's divergence):

$$JD(Q,V) = KL(Q,V) + KL(V,Q)$$

- Hellinger distance:

$$d_{\mathrm{Hell}}(Q,V) = \sqrt{1 - \sum_i \sqrt{q_i v_i}}$$

Proper metric, constrained to interval [0,1]

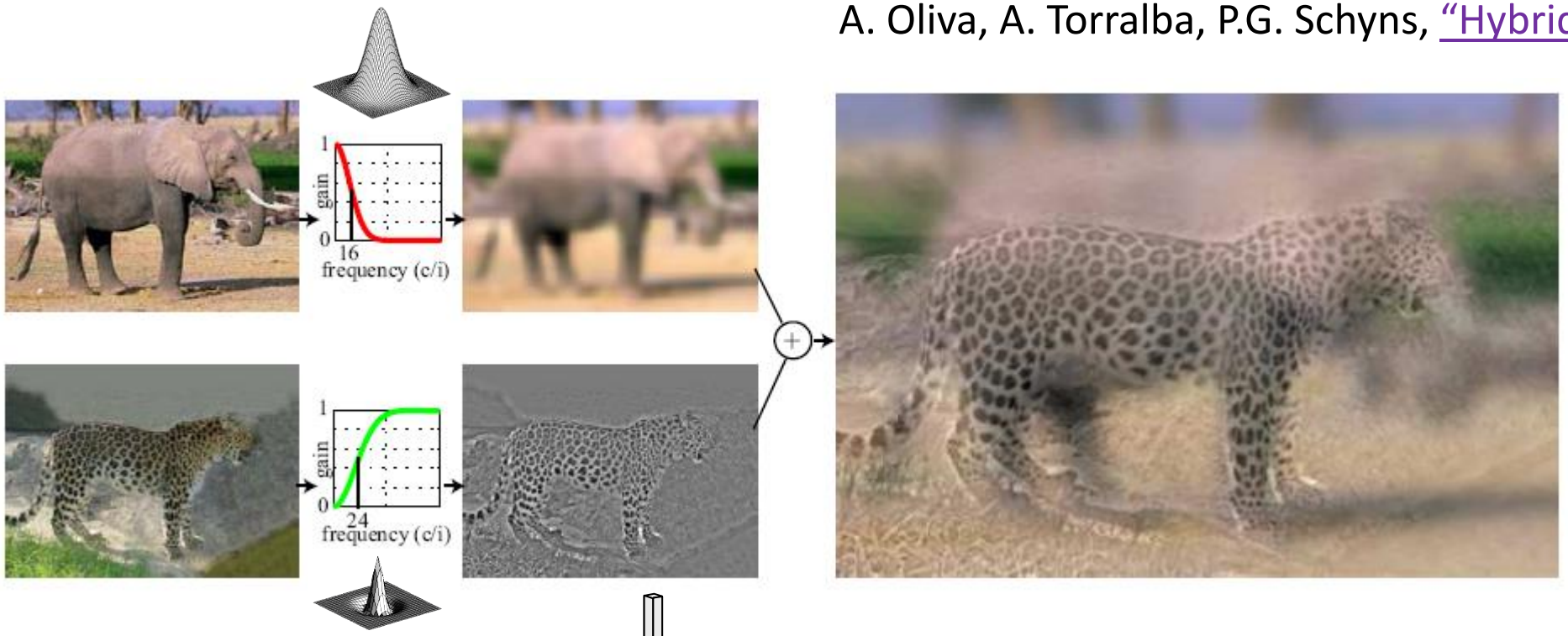
# References


- David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition), (prva izdaja dostopna na spletu)
  (Ozadje linearnih filtrov in povezavo s Fourierjevim transformom najdete v Poglavjih 7 in 8)
- R. Szeliski,Computer Vision: Algorithms and Applications, 2010
- Kristen Grauman, „Computer Vision“, lectures

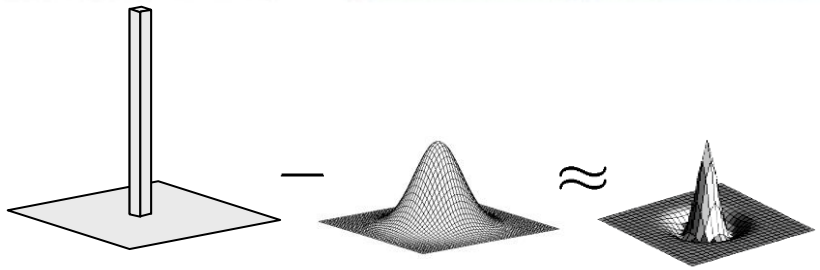
# Appendix: Fun with hybrid images…

Gaussian filter

A. Oliva, A. Torralba, P.G. Schyns, "Hybrid Images," SIGGRAPH 2006



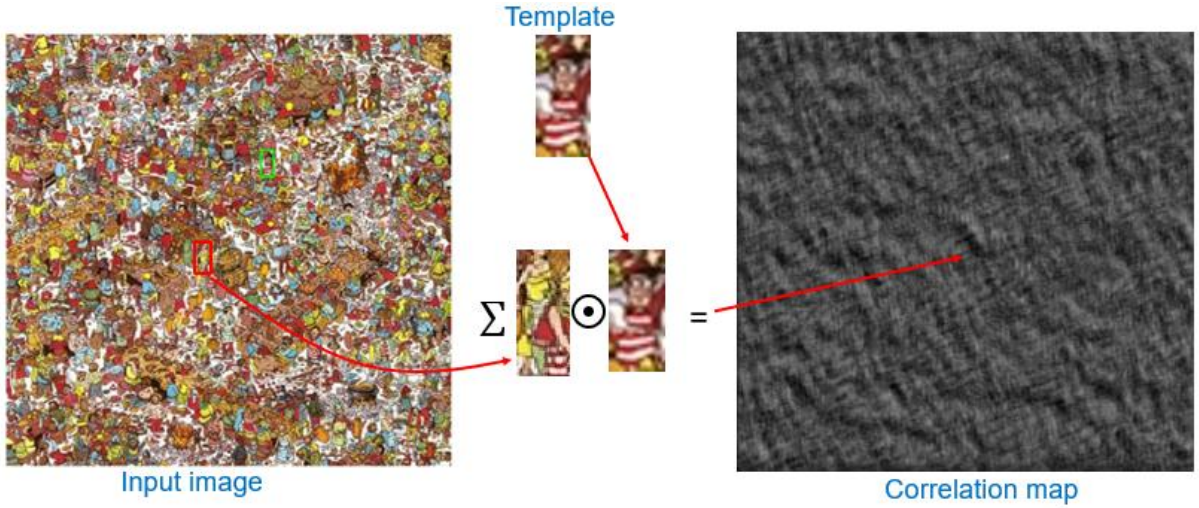

Laplacian filter:

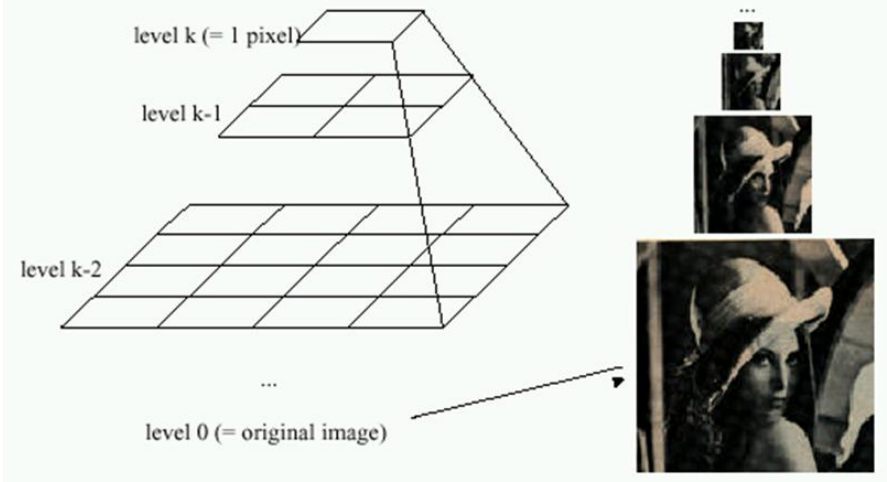


Unit impulse — Gauss ≈ Laplacian of Gaussian

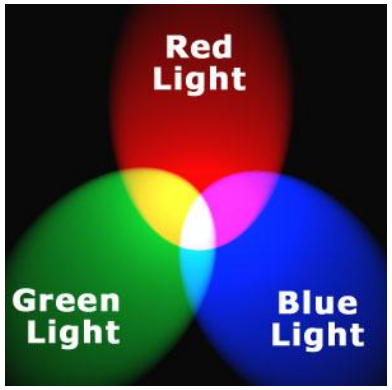
# Previously at MP…

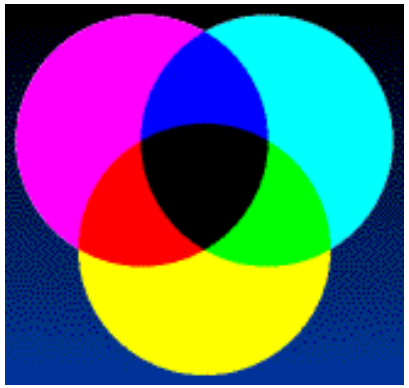
- Correlation, Gaussian pyramids

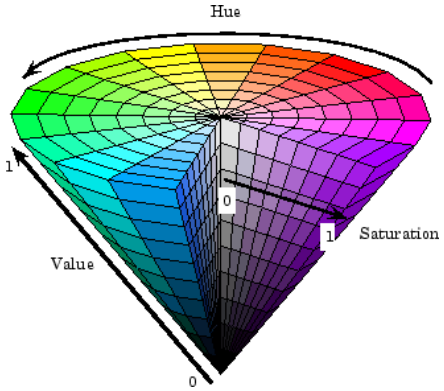


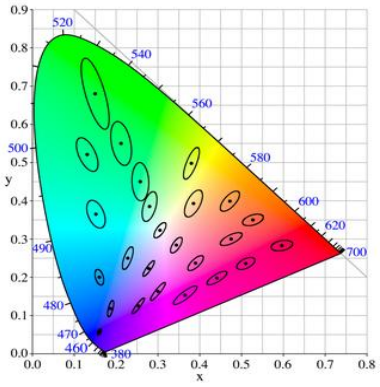



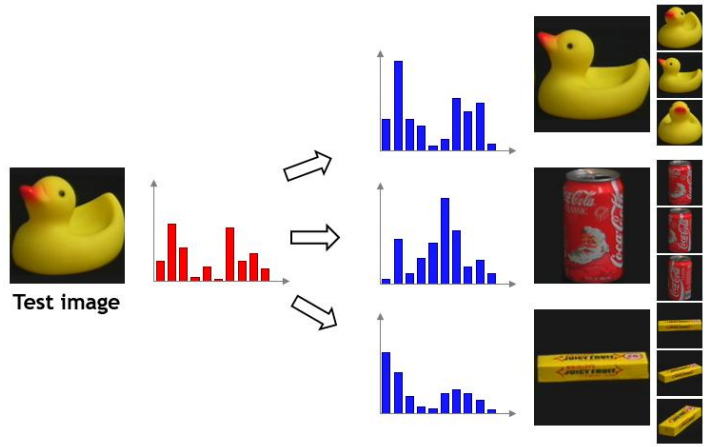

- Color – perception, color spaces and color histograms as image descriptors





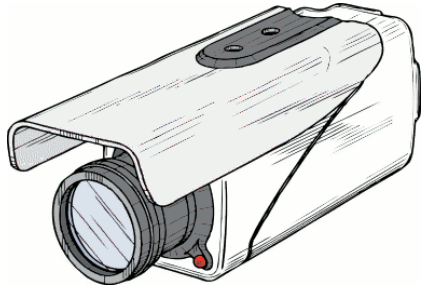





Univerza v Ljubljani

ViCOS visual cognitive systems lab

# Machine perception
# Derivatives and edge detection

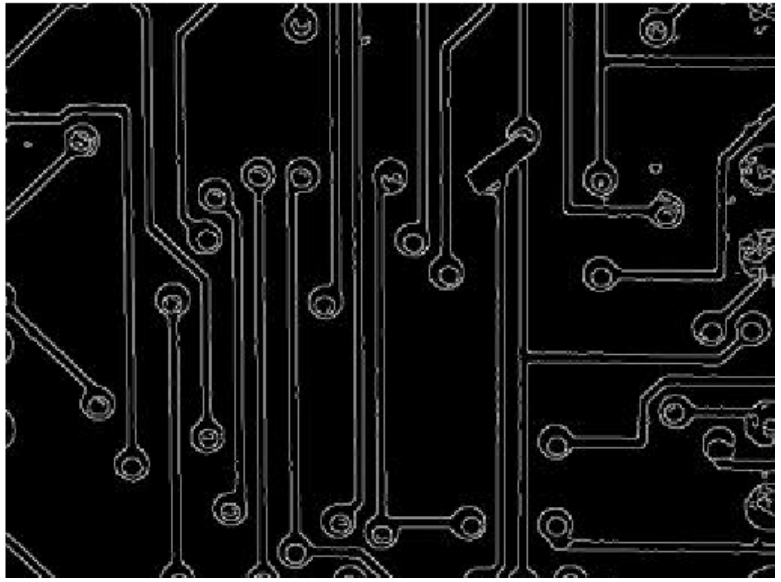


Matej Kristan


Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

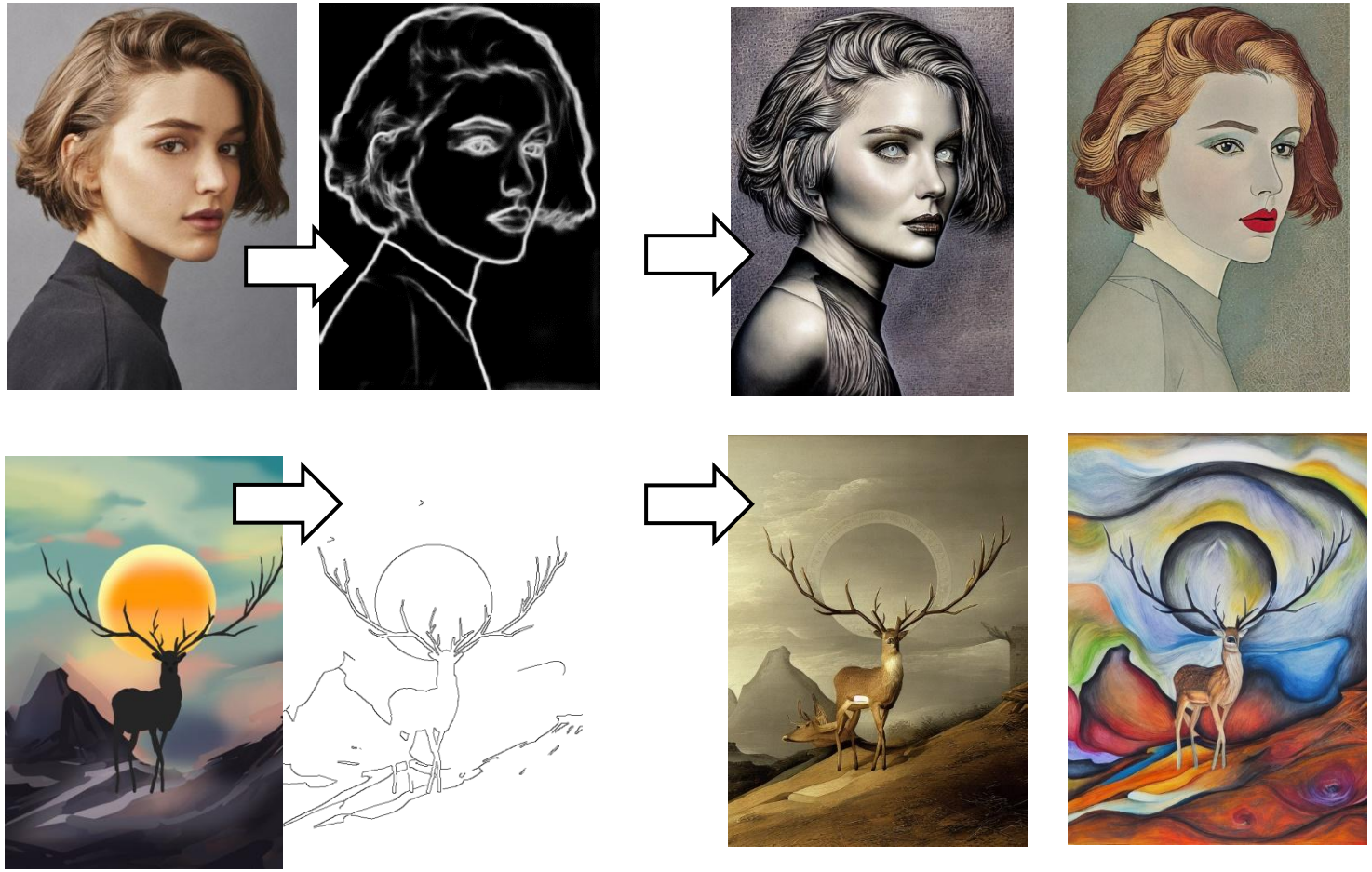
# Edge detection

- Goal: map image from 2D grayscale intensity pixel array into a set of binary curves and lines.
- Why?



Measurement

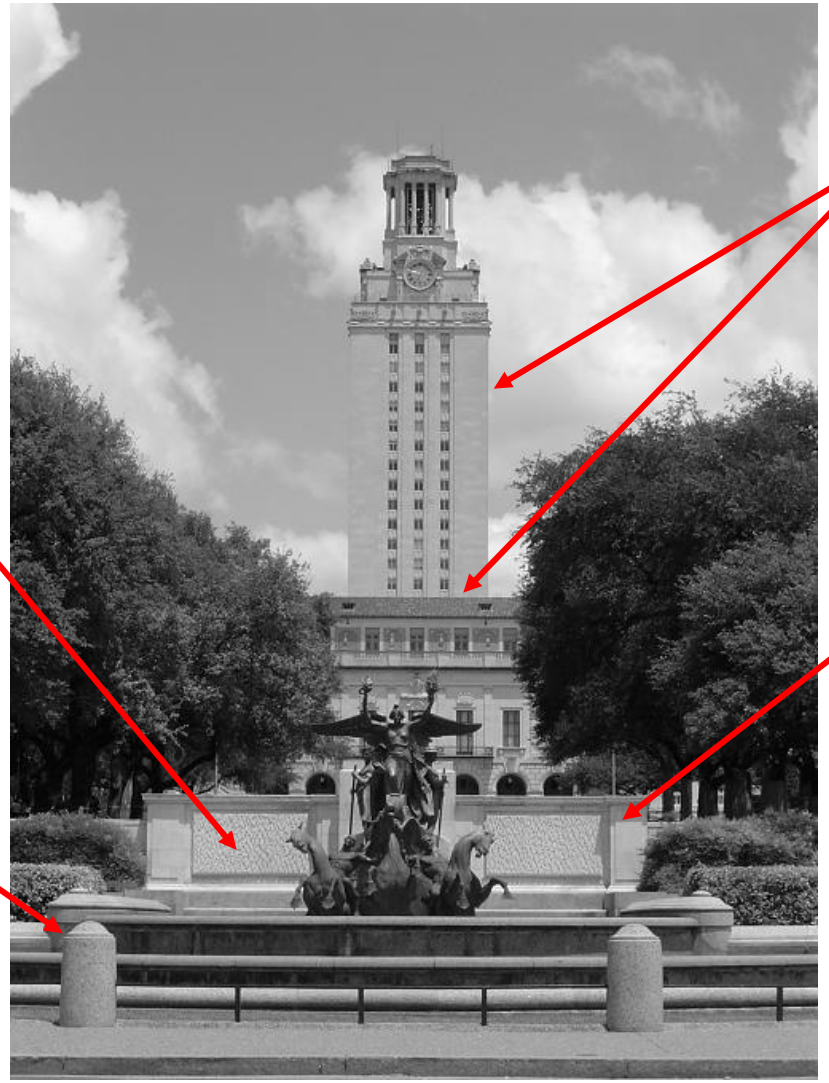
Abstraction



Zhang et al., Adding Conditional Control to Text-to-Image Diffusion Models, ICCV2023

# What constitutes an edge?

Anything that *appears* as an edge…



Discontinuity of depth: object borders

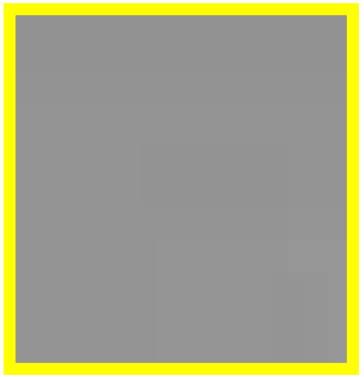
Local texture:

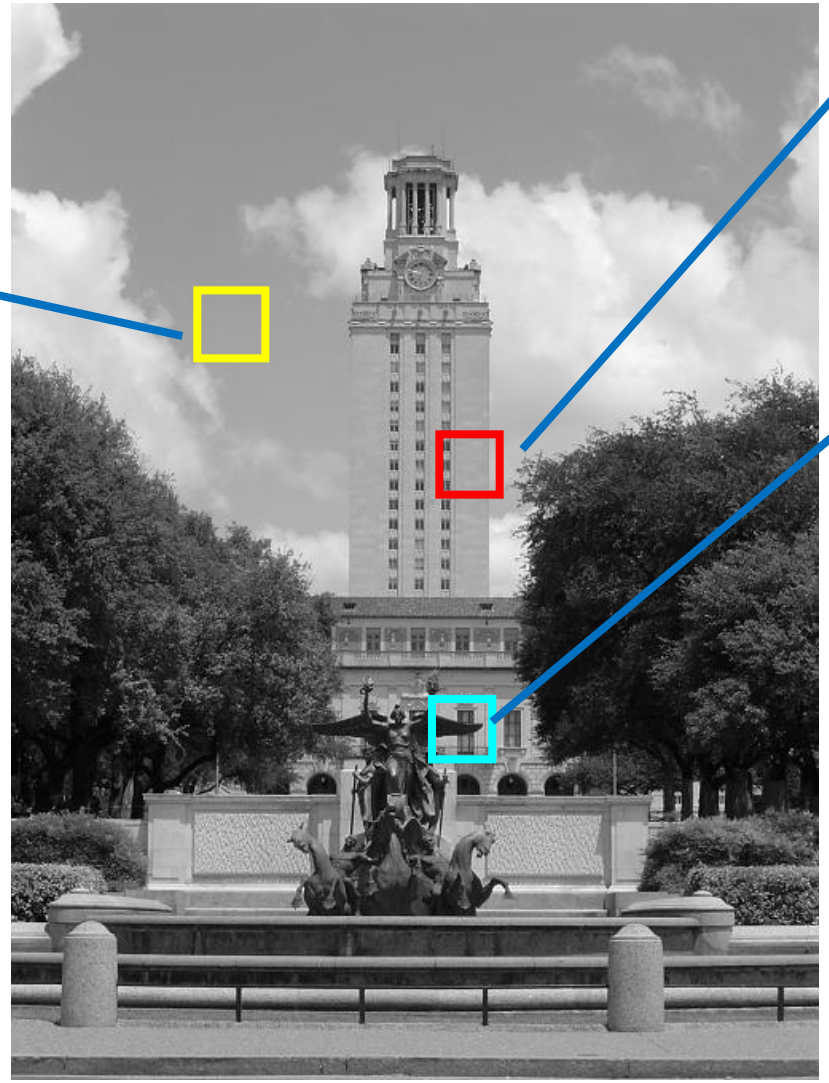
Shadows

Changes in 3D normal orientation caused by shape changes

# What constitutes an edge?

Anything that *appears* as an edge…





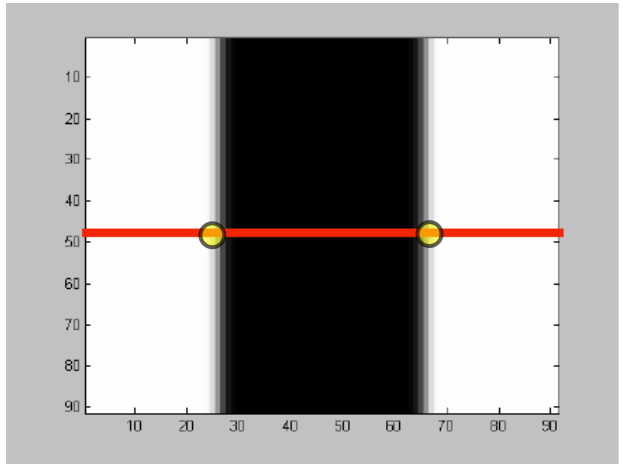
Edge presence is strongly correlated with the local intensity changes.

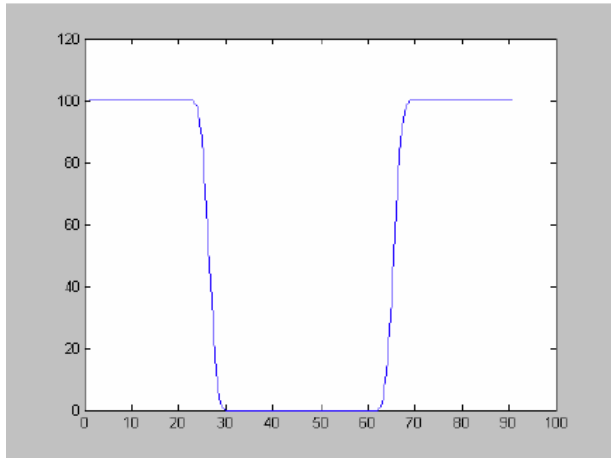
Operator that measures a local intensity change: Derivative

Machine perception

# IMAGE DERIVATIVES

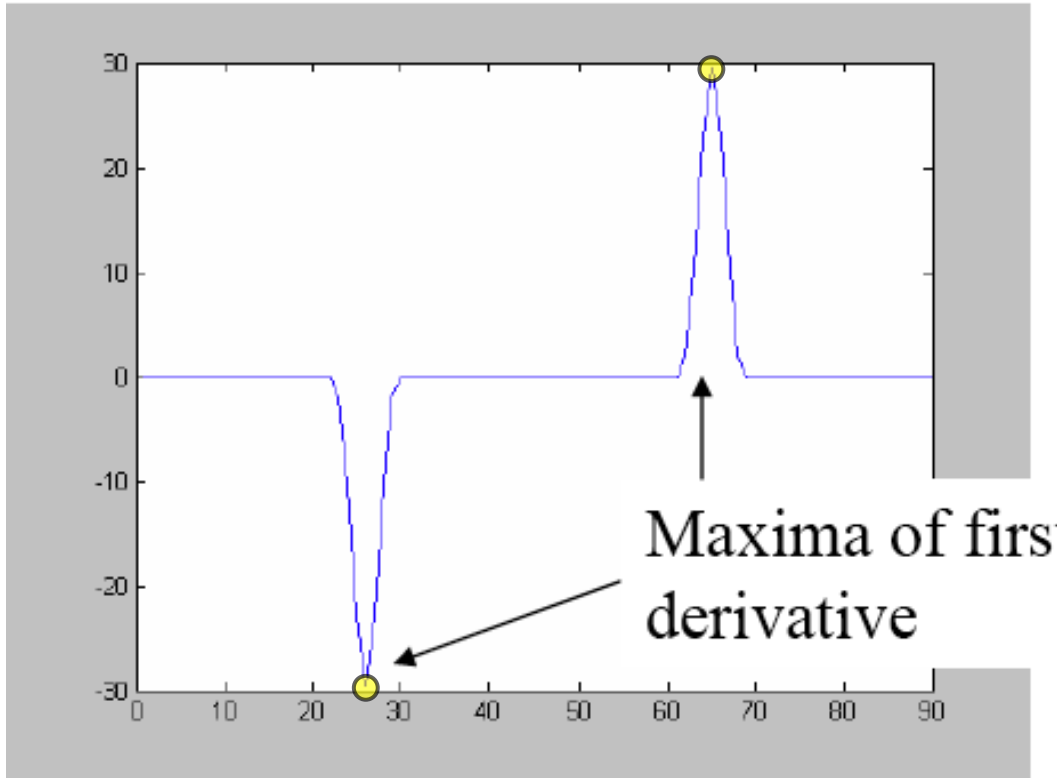
# 1D derivative: Intuition





First order derivative

$$\frac{d}{dx} f(x)$$



# Derivatives and convolution

- A partial derivative of a continuous 2D function $f(x,y)$ :

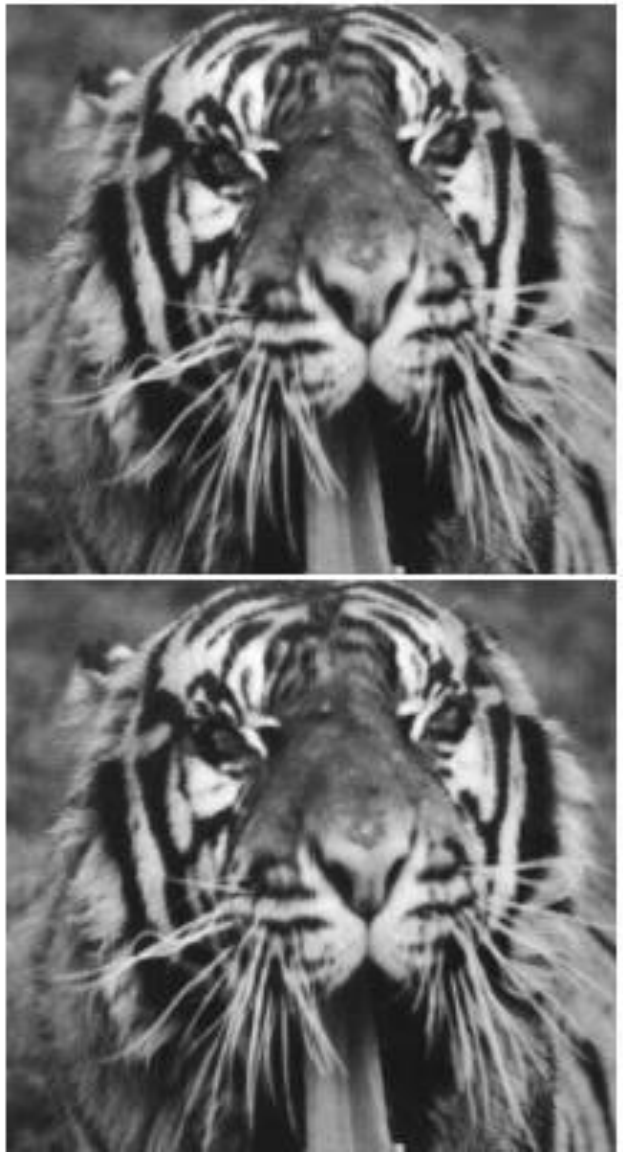
$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$$

- For a discrete case, approximate by using finite differences:

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

- Question: If implemented by convolution, what would the convolution kernel for derivative look like? (Next slide)

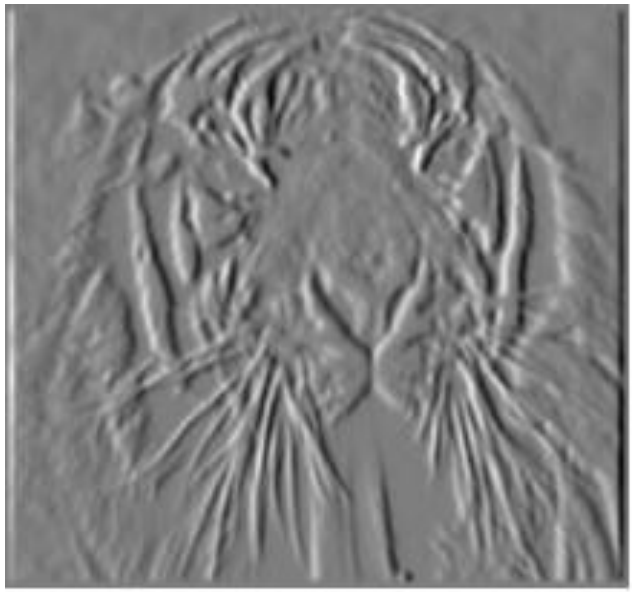
# Partial derivatives: Implementation



Horizontal derivative

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

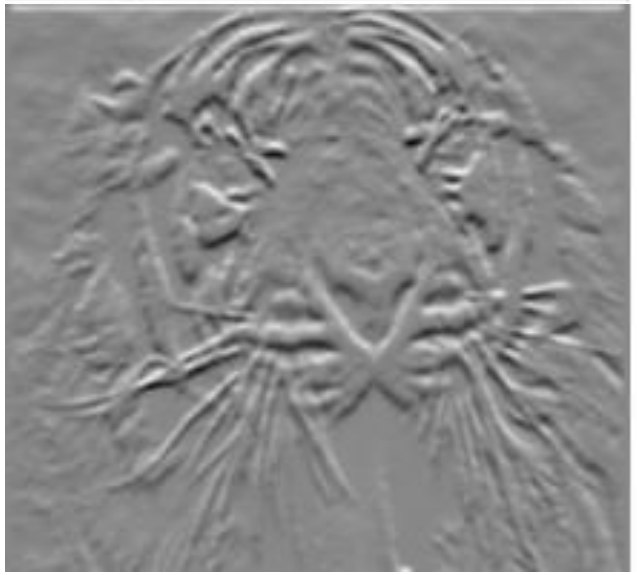
$$* \quad \boxed{-1 \mid 1} =$$



Vertical derivative

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

$$* \quad \begin{array}{|c|}\hline -1 \\ \hline 1 \\ \hline \end{array} =$$



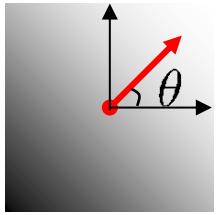
# Partial derivatives: Image gradient

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

- Gradient points in direction of greatest intensity change:

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

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




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

- Gradient direction (orientation of edge normal):

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

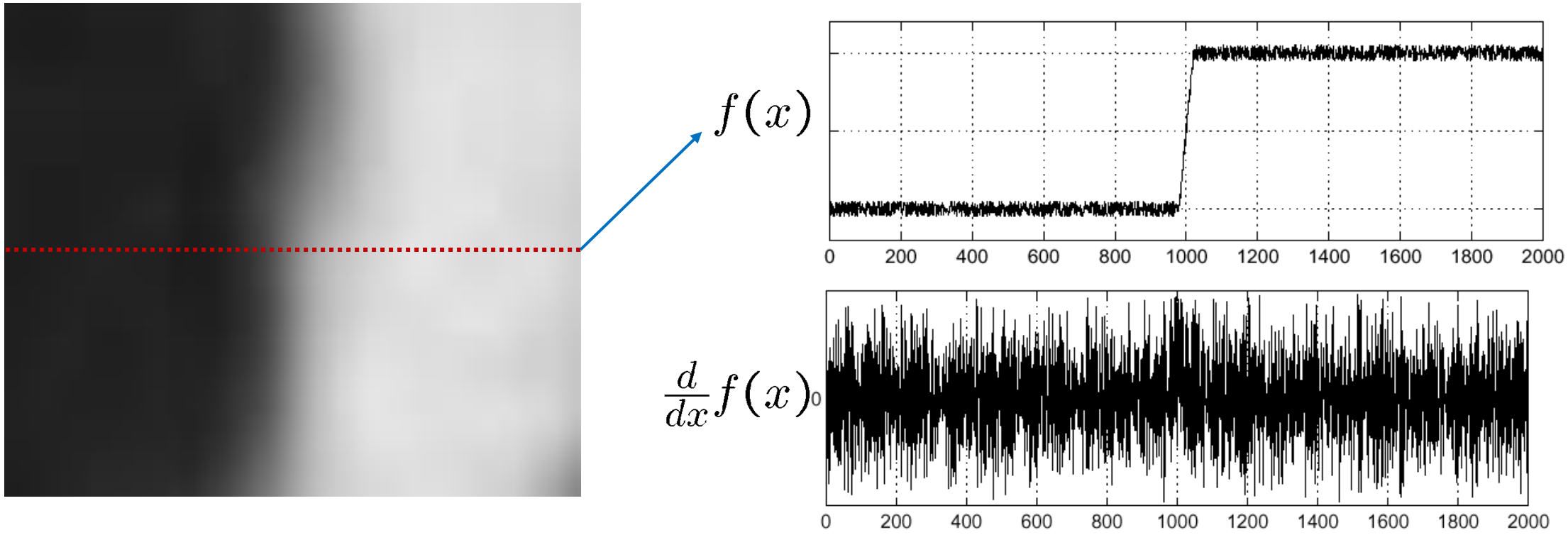
- Gradient strength is defined by its magnitude:

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

Gradient magnitude

# Discrete world is noisy...

- Take a single line in the image:
  - Plot intensities w.r.t. pixels:




*So where did the edge go!? Noise gets amplified by derivation...*

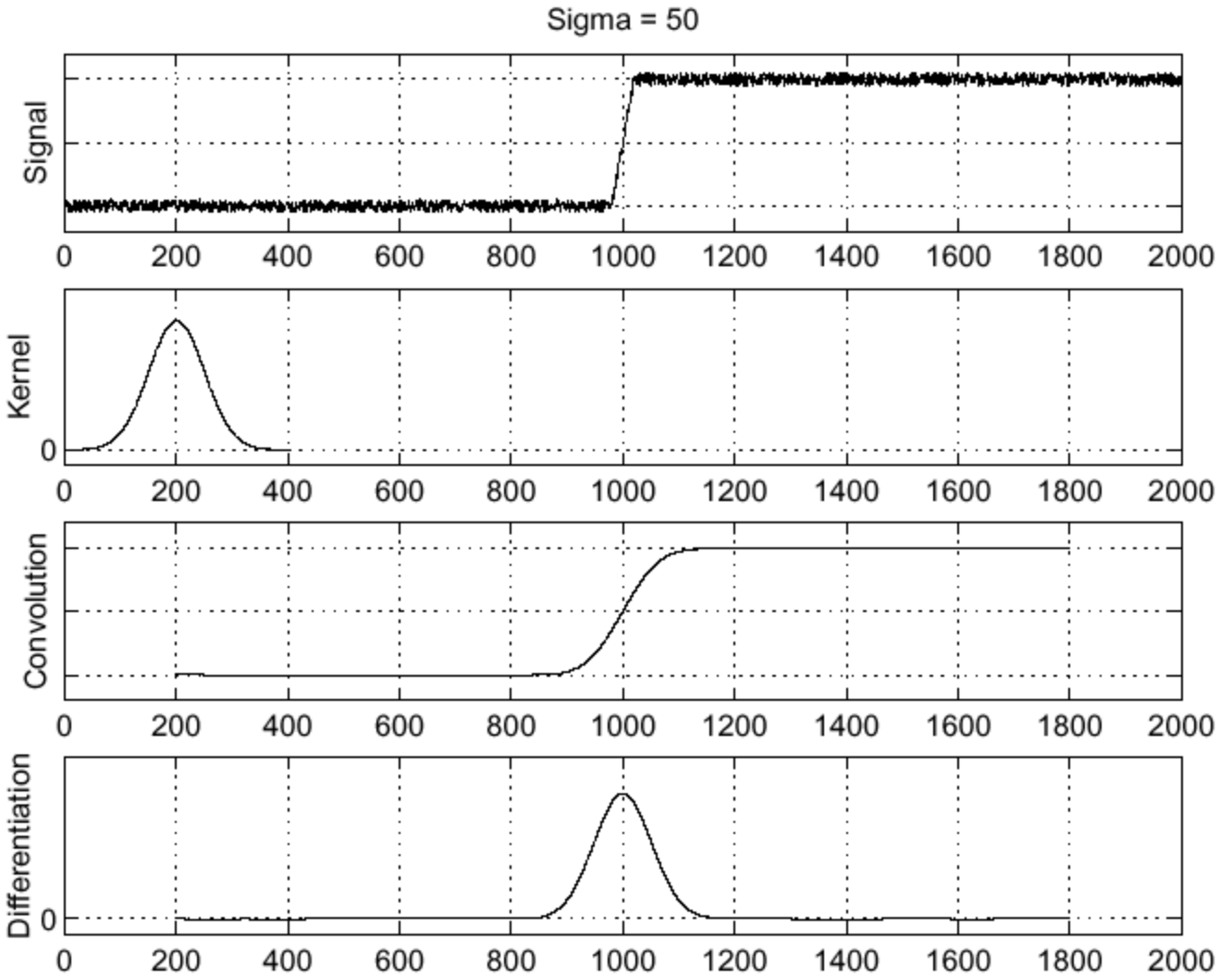
# Solution: Smooth the image first

$f$

$h$

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$




Where's the edge?

Find maxima in map $\frac{\partial}{\partial x}(h \star f)$

# Remember convolution properties

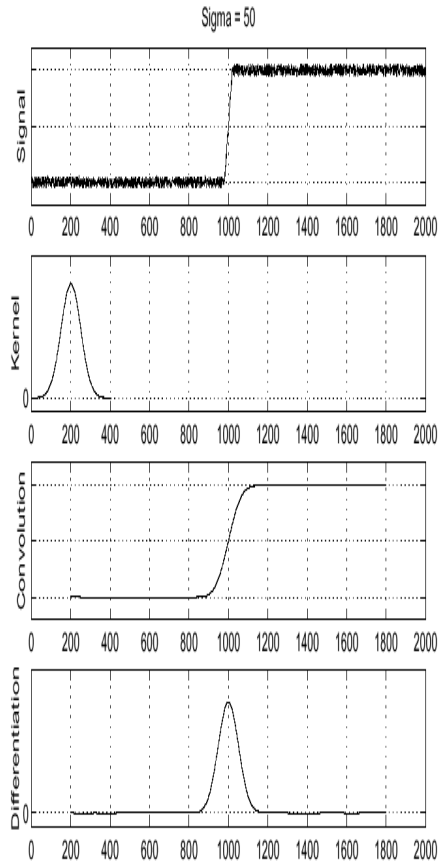
- Derivatives: $\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$

$f$

$h$

$h \star f$

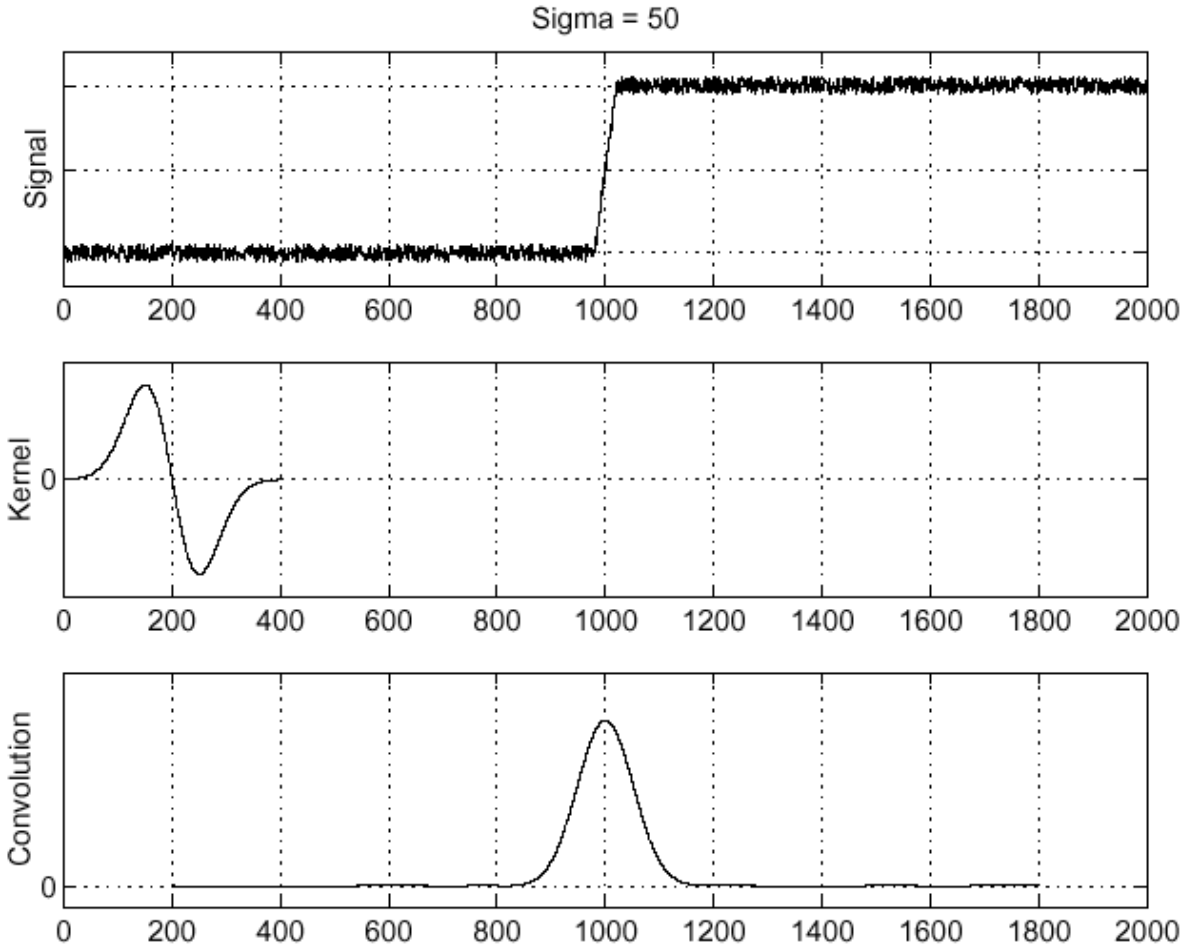
$\frac{\partial}{\partial x}(h \star f)$




$f$

$\frac{\partial}{\partial x}h$

$(\frac{\partial}{\partial x}h) \star f$



# 2D partial derivatives – naiive approach

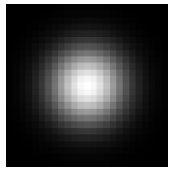
1. Smooth the image by a 2D Gaussian filter
2. Take derivative w.r.t. x

1 Blurring

Image

*

Gaussian kernel



=

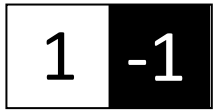
Blurred image

2 Differentiating w.r.t. x

Blurred image

*

Derivative kernel



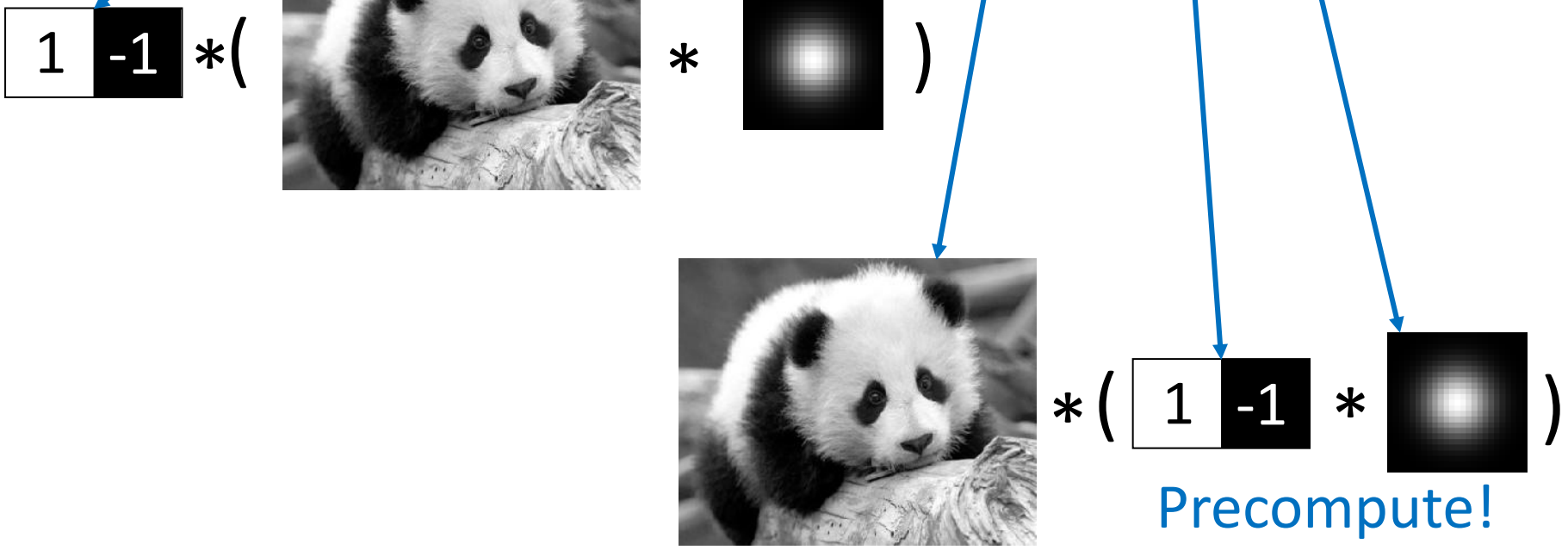

=

Derivative image

# 2D partial derivatives – smarter approach

- Recall the convolution property:

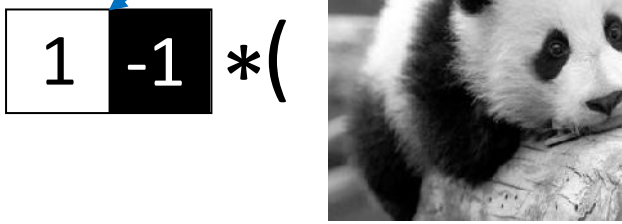
$$\frac{\partial}{\partial x}(I * G) = I * (\frac{\partial}{\partial x} G)$$

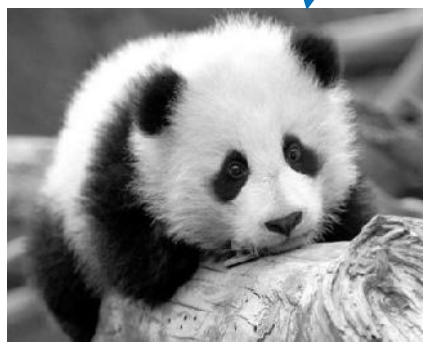


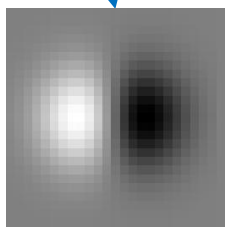
# Smarter way

- Recall the convolution property:

$$\frac{\partial}{\partial x}(I * G) = I * (\frac{\partial}{\partial x} G)$$



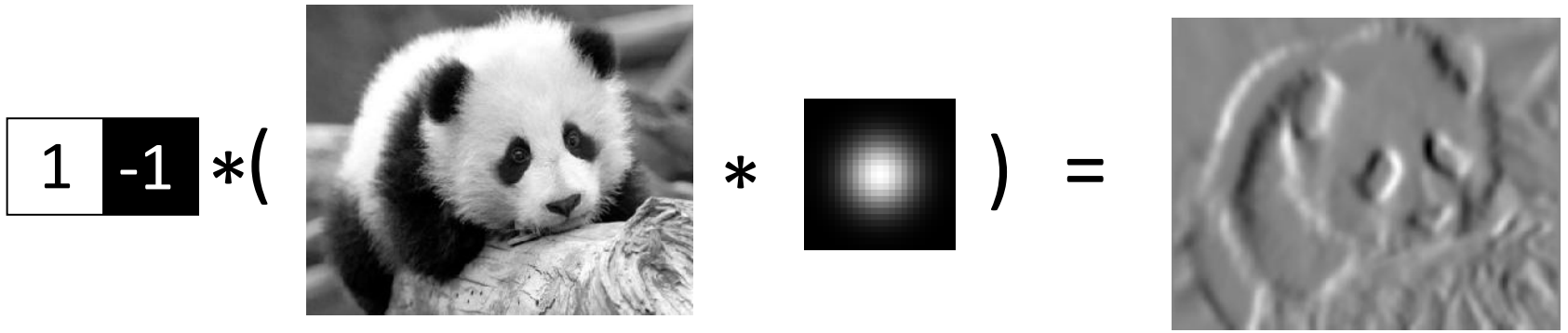
Can precompute analytically and discretize!

# 2D partial derivatives – smarter approach

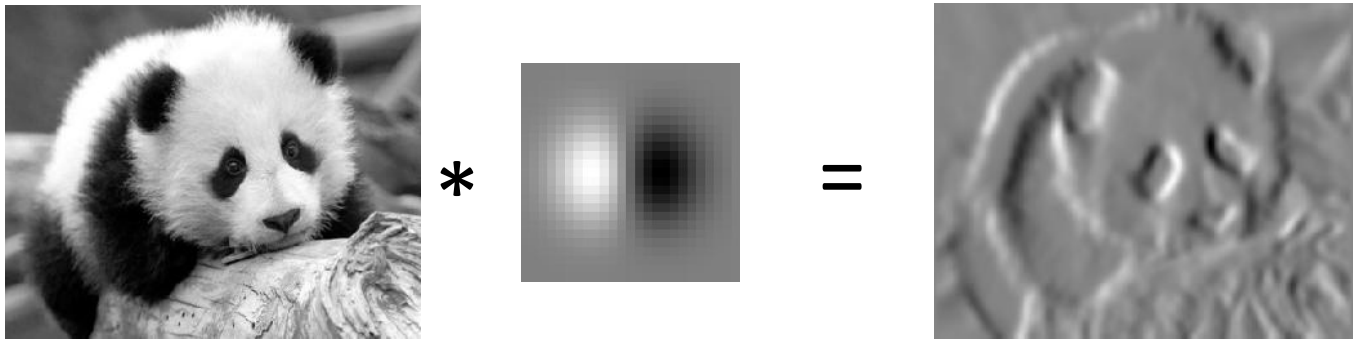
- Recall the convolution property:

$$\frac{\partial}{\partial x}(I * G) = I * (\frac{\partial}{\partial x} G)$$

Naiive:

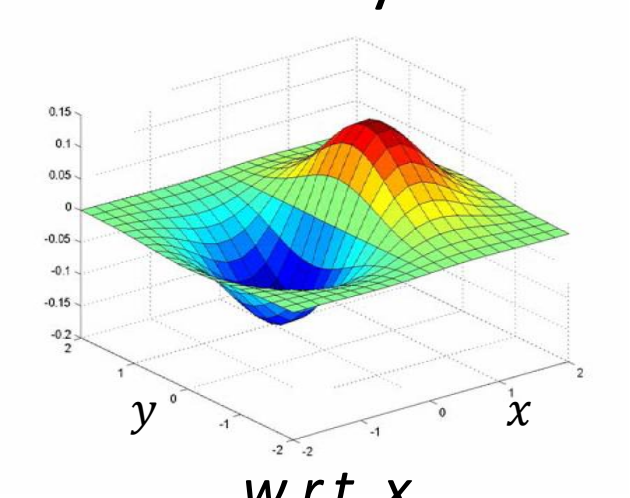



Smarter:

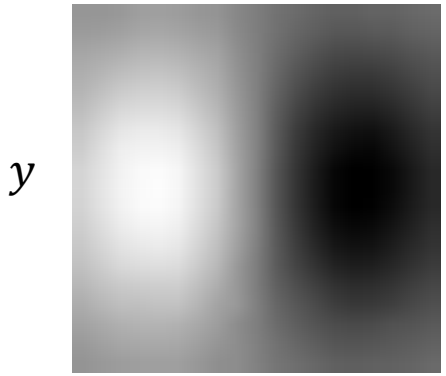


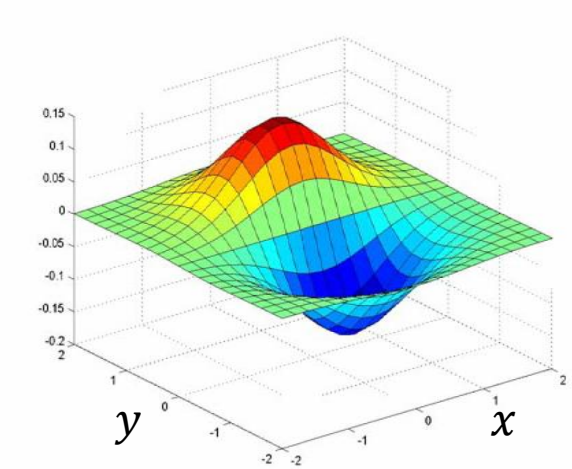
# Gaussian partial derivatives

- Convolution kernels for taking partial derivatives w.r.t. x and y:

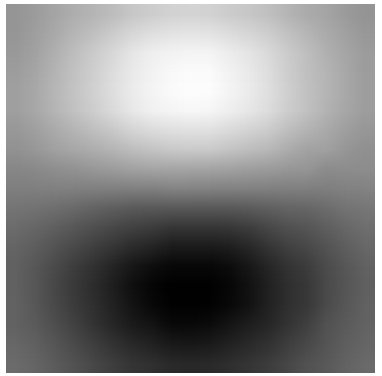



*w.r.t. x*




*w.r.t. y*



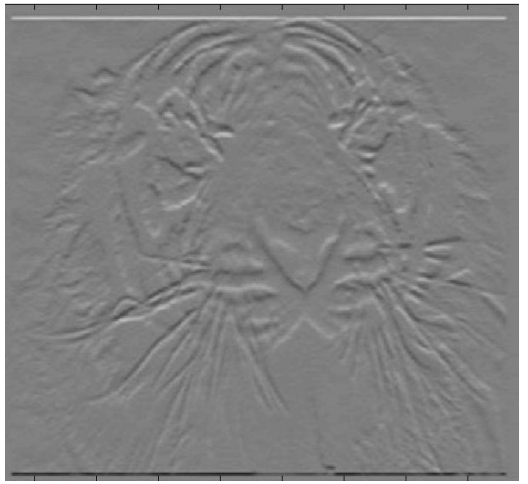
# Some other popular kernels

**Prewitt:** $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$ ; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

**Sobel:** $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$ ; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

**Roberts:** $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$ ; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

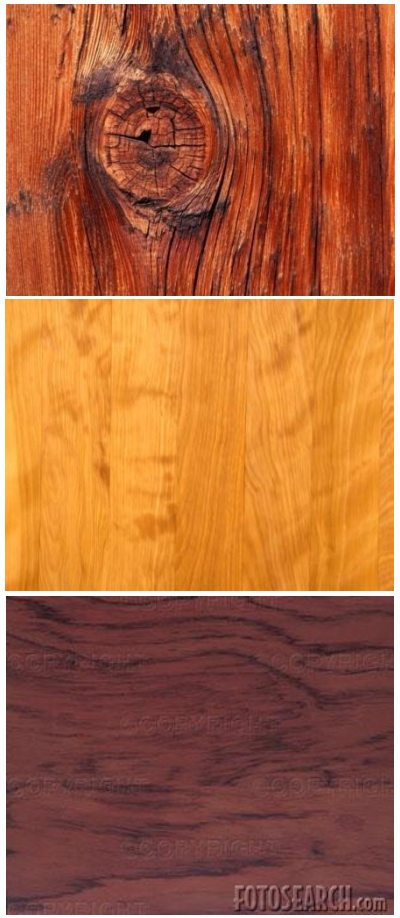
```
>> My = fspecial('sobel');
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```



# Edges exist at different scales

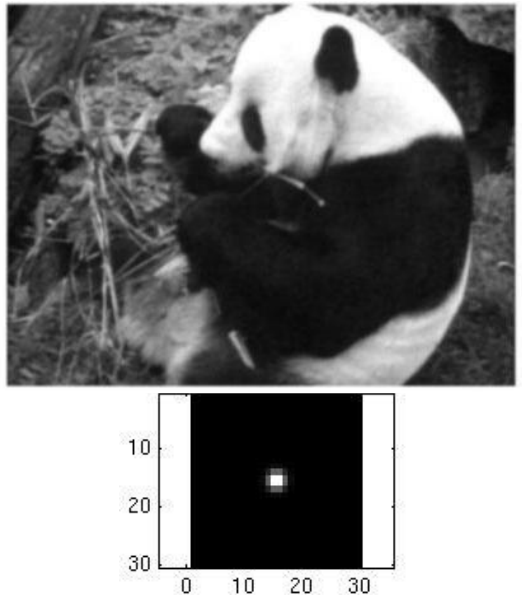
Depends on what we're looking for...

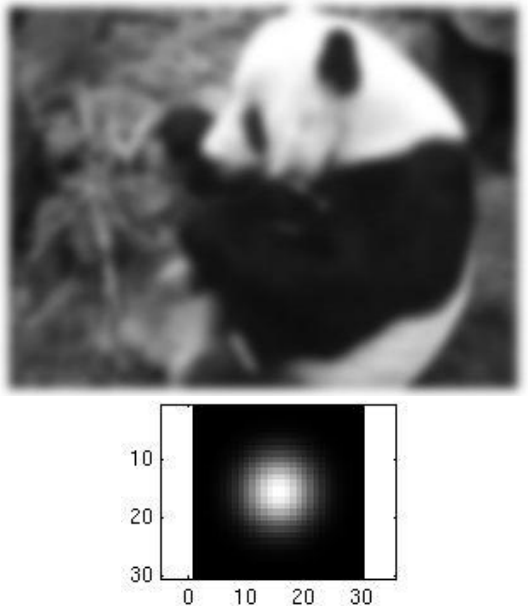
Thin edges or thick edges (leaves, branches, trunks,…)



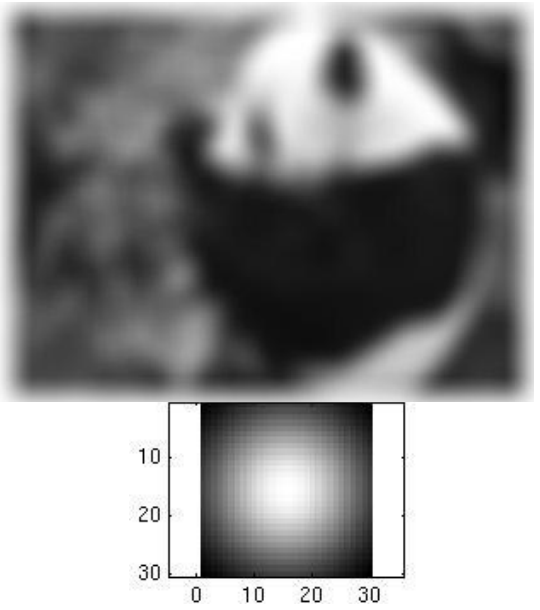

# Tuning the filter to the right scale

Parameter σ is the “scale”/“width” of a Gaussian kernel that determines the extent of smoothing, i.e., determines which *edges will be removed*.



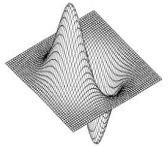




...

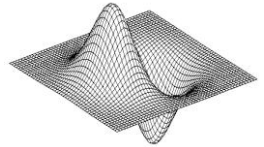


# Tuning the filter to the right scale

How does σ affect the derivative?



σ = 1 pixel



σ = 3 pixels

The enhanced/detected structures depend on the Gaussian kernel size.

Large kernels: emphasize edges on a larger scale.
Small kernels: emphasize edges on a smaller scale.

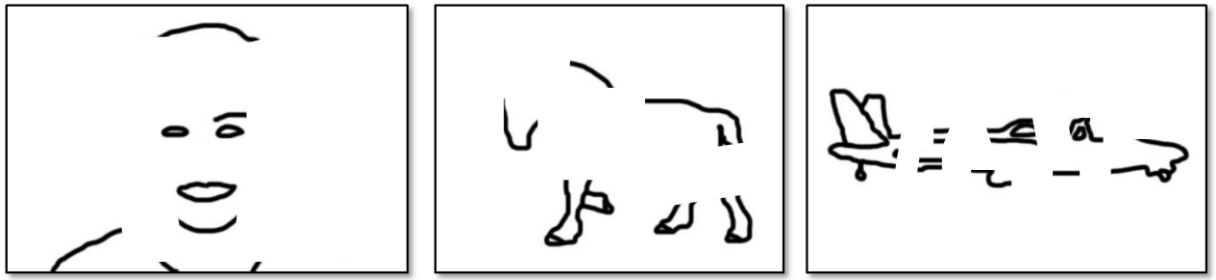
Machine perception

# FROM DERIVATIVES TO EDGE DETECTION

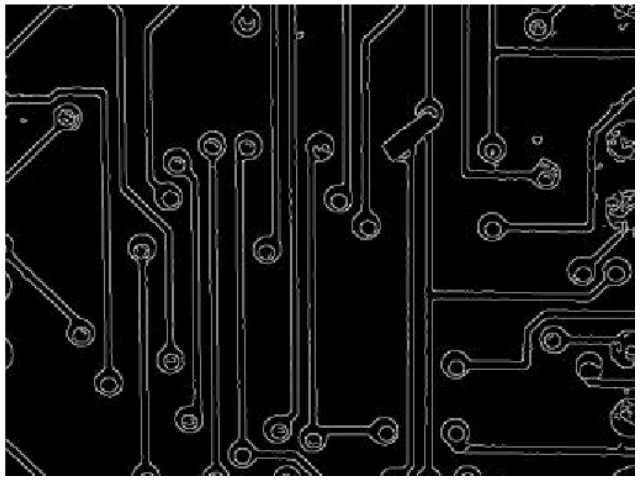
# Recall: The task of edge detection

- Goal: map image from 2D grayscale intensity pixel array into a set of binary curves and lines.

abstraction



Robust, compact representation



Measurement

Derivative enhances the edges, but these are not *binary curves*.

# The task of edge detection

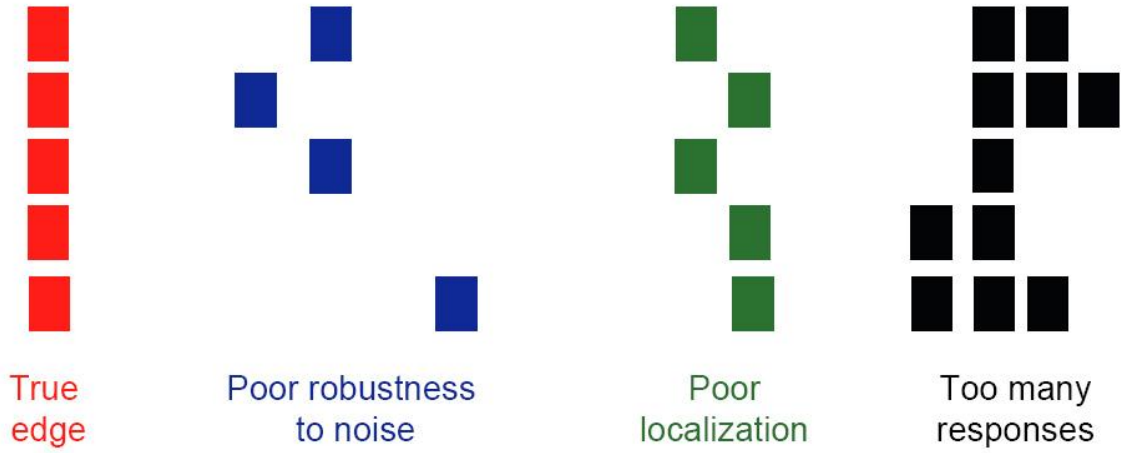
$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- Basic approach:

  find strong gradients + post process

# Designing an edge detector...

- Criteria of "optimal" edge detector:
  1. **Good detection:** optimal detector minimizes probability of false positives (edges caused by noise), and false negatives (missing true edges)
  2. **Good localization:** detected edges should be close to the location of the true edges.
  3. **Specificity:** detector should return only a single point per true edge; minimize number of local maxima around true edge.



# The Canny edge detector [Canny, IEEETPAMI 1986]

- Most popular edge detector in computer vision.
- Theoretical model of *the edge*:

  A step function + Gaussian noise.
- Canny showed that first derivative of a Gaussian well approximates an operator that optimizes a tradeoff between signal-to-noise ratio and localization on the specified theoretical edge model.

Python:

```
>> cv2.Canny(image, Th_lo, Th_hi,…)
```

J. Canny, *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

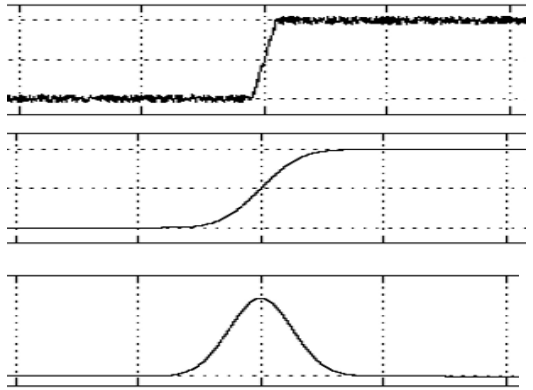
# Canny edge detector

1. Filter image by a derivative of a Gaussian (smooth and enhance)

$$* \longrightarrow$$

$$* \longrightarrow$$

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

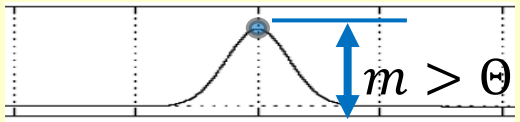


2. Calculate the gradient magnitude and orientation

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

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

3. Thin potential edges to a single pixel thickness




4. Select sequences of connected pixels that are likely an edge

# Canny: enhancing the potential edge pixels

Original image (Lena)

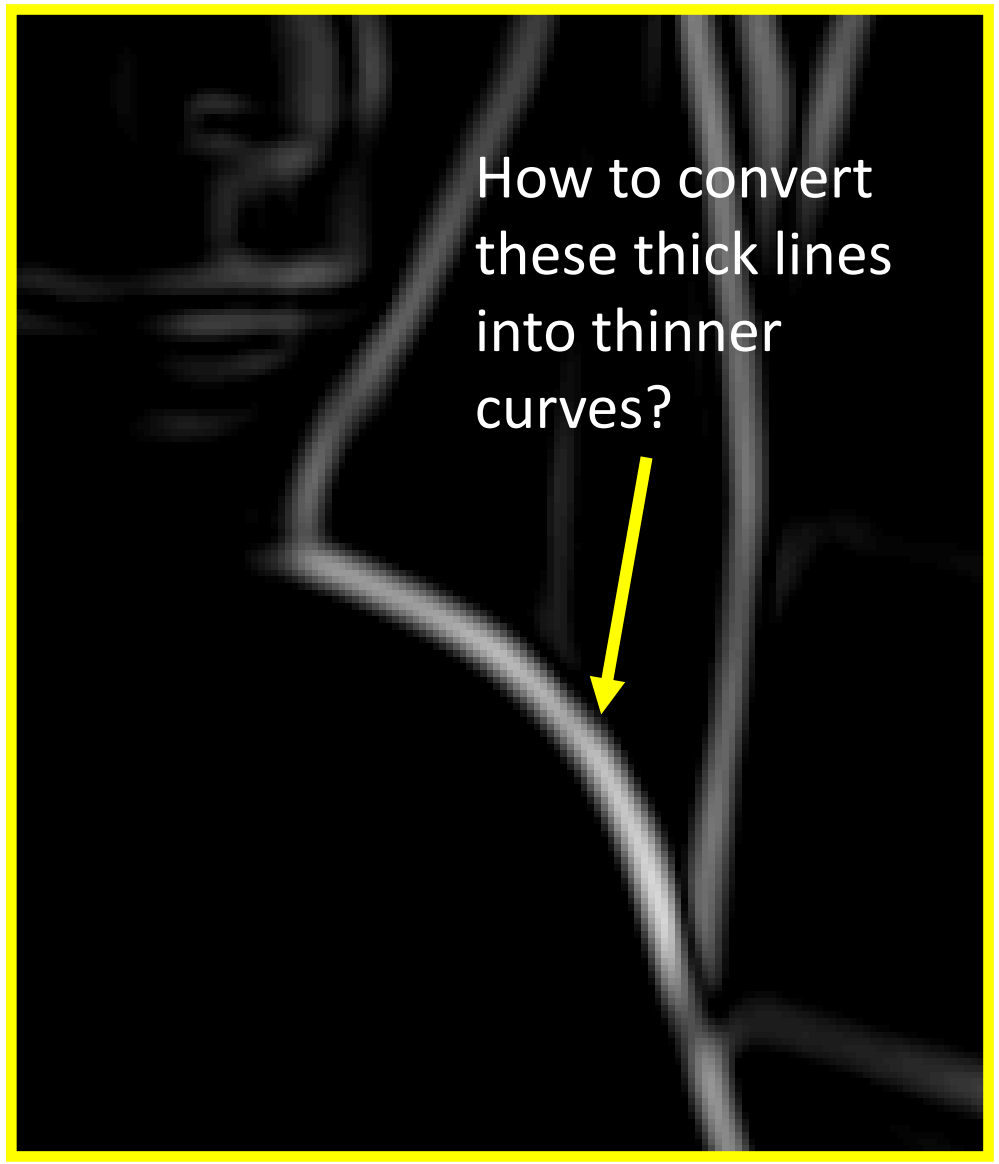
Gradient magnitude

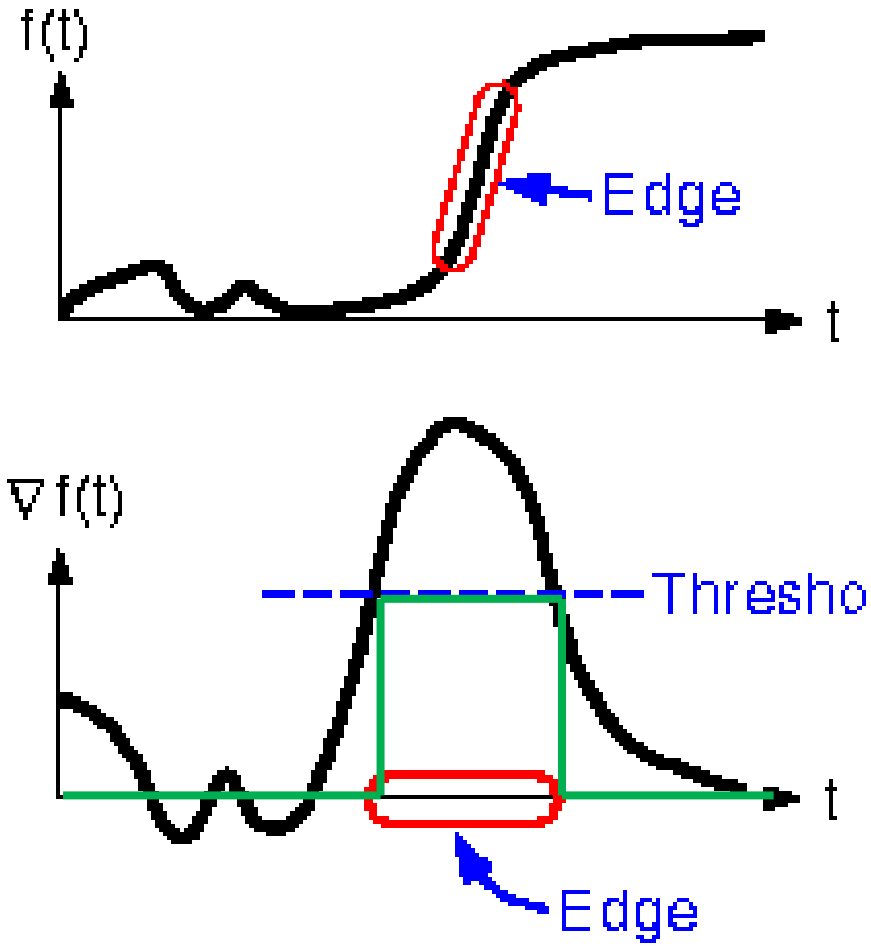
# Canny: enhancing the potential edge pixels

Original image (Lena)

„Thresholding“: Set magnitudes lower than a prescribed threshold to 0.

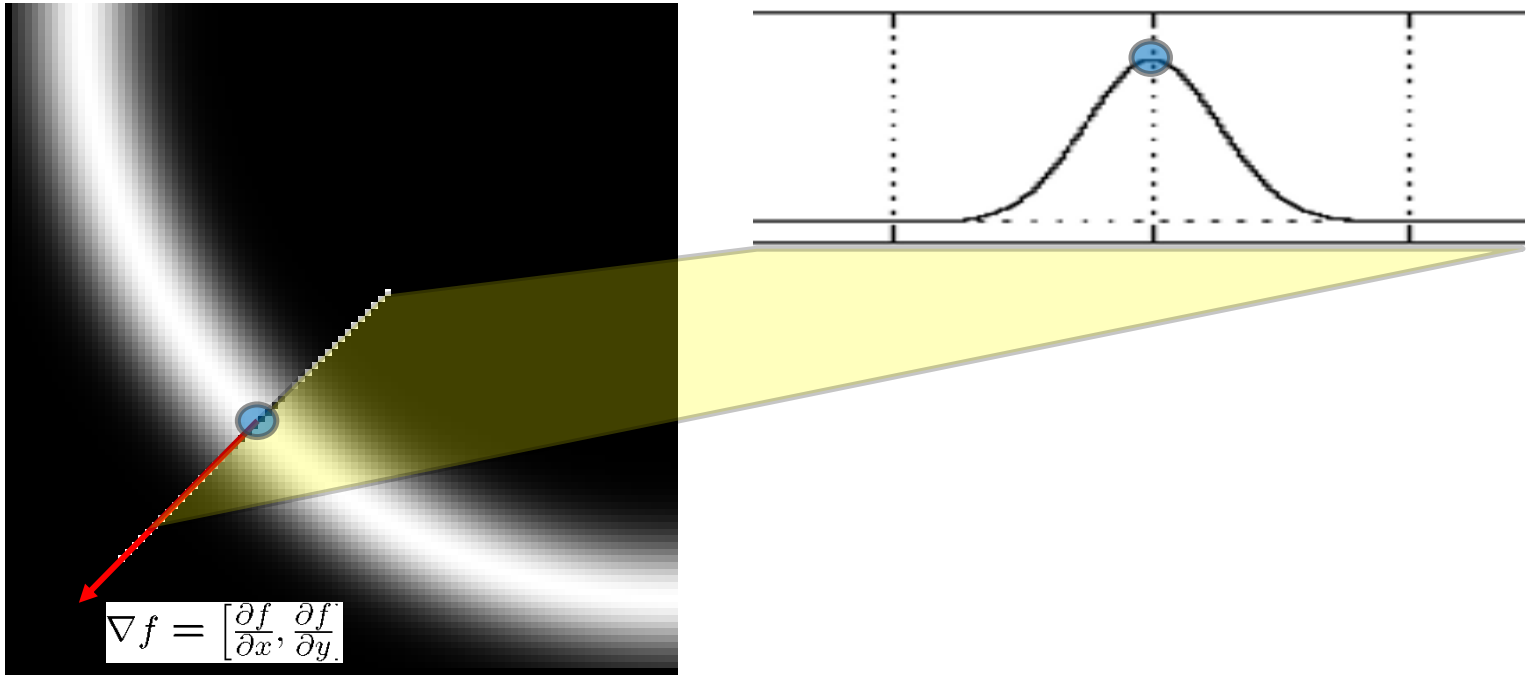
# Canny: thinning the edges

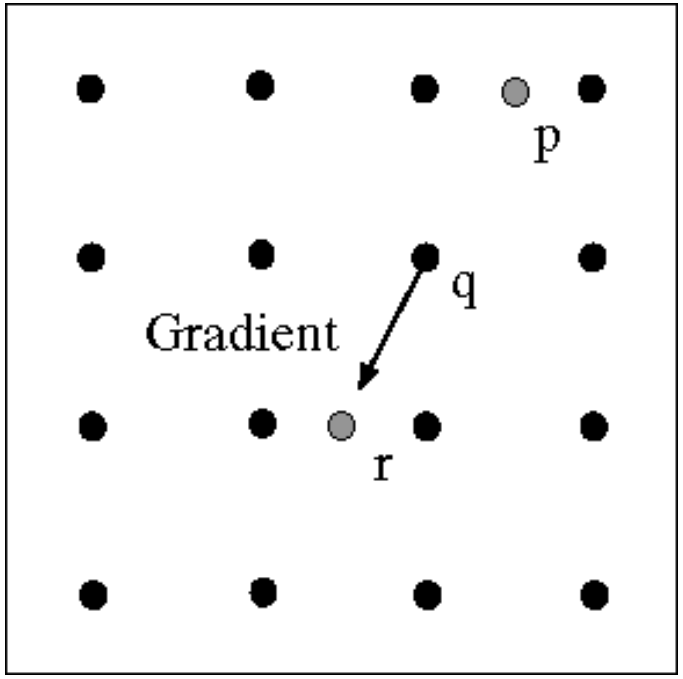



Not by thresholding…

# Thinning by non-maxima suppression



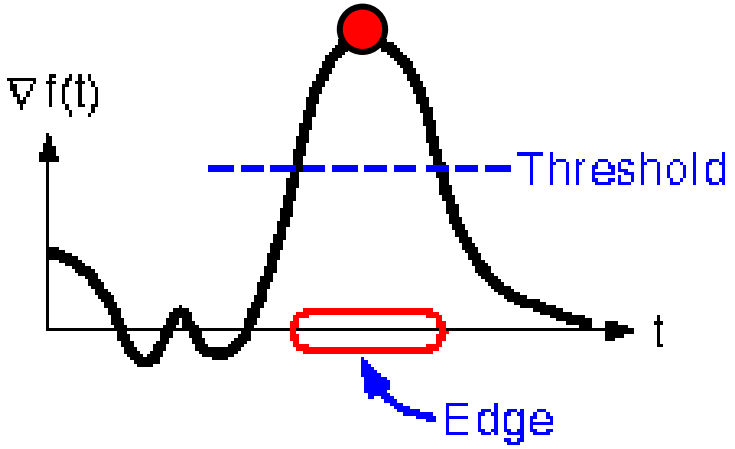

- For each pixel check if it is a local maximum along its gradient direction.
  - Advanced: Actually, for **q**, we should check interpolated pixel values at **p** and **r**.
- Only local maxima should remain.

# Canny: thinning the edges

Thinning

(non-maximum suppression)

# Canny: thinning the edges

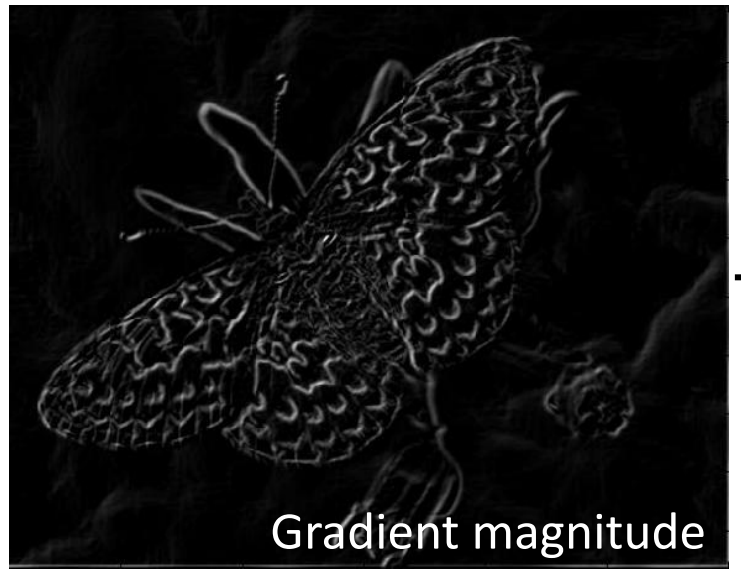



Problem: pixels along this edge did not „survive“ thresholding.

Thinning
(non-maximum suppression)

# How to select a threshold?

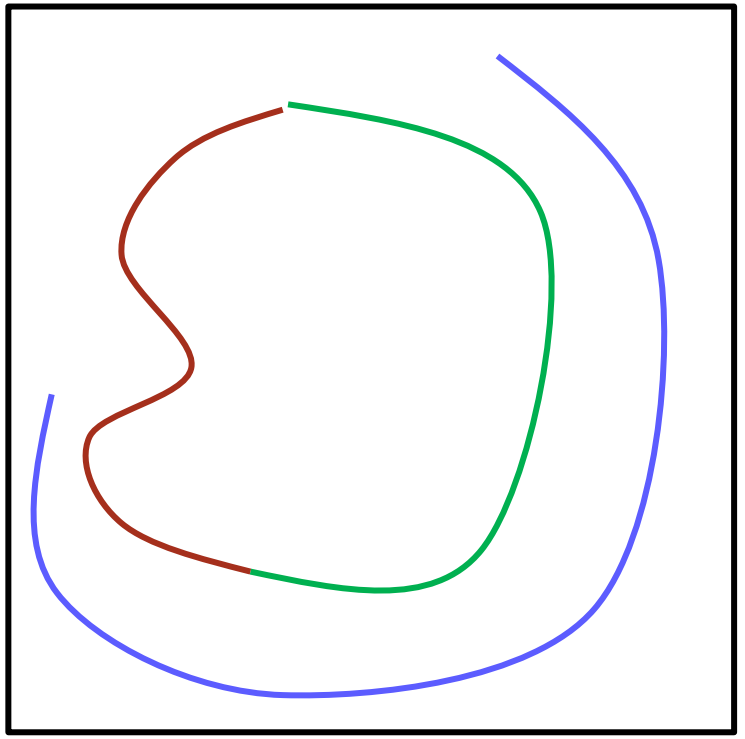
Threshold + Thinning




Threshold + Thinning


# Canny edge detector: Hysteresis thresholding

- Trace each contour separately (e.g., using 4-connectedness).
- Apply two thresholds $k_{high}$ and $k_{low}$
  - Start tracing a line only at pixels that exceed a high threshold $k_{high}$.
  - Continue tracing if the pixels exceed a lower threshold $k_{low}$.
- Typical threshold ratio: $k_{high} / k_{low} = 2$



# Hysteresis thresholding

Original

High threshold
(strong edges)

Low threshold
(weak edges)

Hysteresis thresholding

# The Canny edge detector in a nutshell

1. Convolve the image by a derivative of a Gaussian.
2. Calculate the gradient magnitude and orientation
3. Non-maxima suppression (NMS)
   - Set low gradient magnitudes to zero to reduce the number of candidates in NMS
   - Thin edges to one-pixel width.
4. Trace the edges by hysteresis thresholding
   - Apply a high threshold on the magnitude to initialize a contours and continue tracing the contour until the magnitude falls below a low threshold.

# Canny edge detector in *"action"*

Input image

gradient magnitude

gradient angle

thinned

Thresholded by hysteresis

# Canny edge detector in *"action" ... with ControlNet*

*"hyperrealistic dramatic sky of a dragon"*

Original image

Thresholded by hysteresis

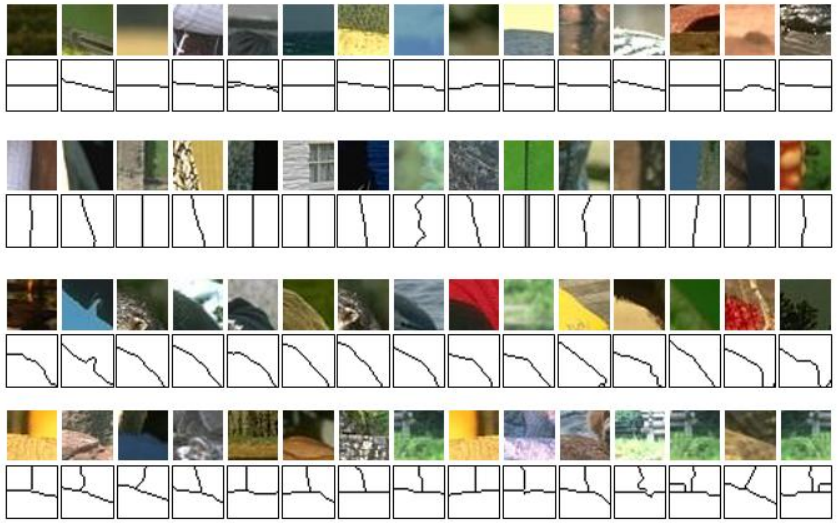
Generated by ControlNet

https://huggingface.co/lllyasviel/sd-controlnet-canny

Zhang et al., Adding Conditional Control to Text-to-Image Diffusion Models, ICCV2023

# Beyond Canny edge detector

- Since Canny's publication, lots of new approaches for edge detection by machine learning.
- Essentially, look at patches and learn what an edge is by inferring the structure from intensities.



Sketch Tokens, CVPR 2013. Joseph Lim, C. Zitnick, and P. Dollár

# Beyond Canny edge detector

- CNNs trained for edge detection


Kung and Fowlkes, Recurrent Pixel Embedding for Instance Grouping, CVPR2018

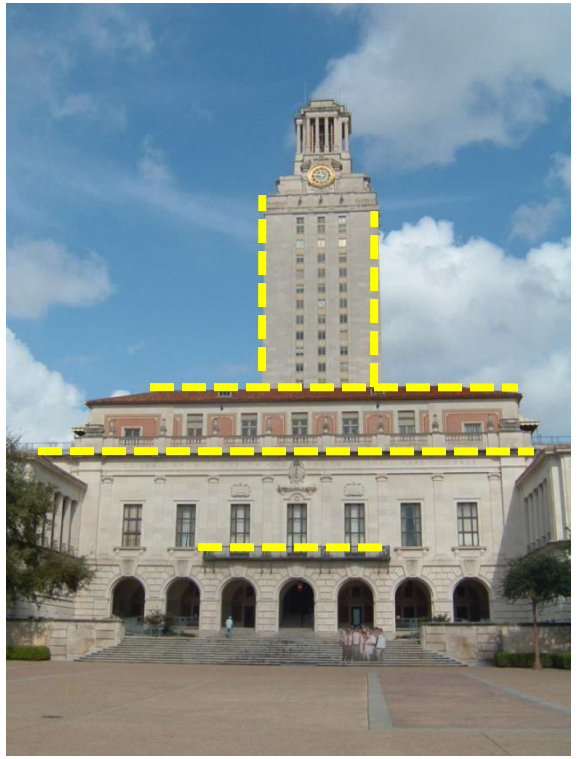
Machine perception

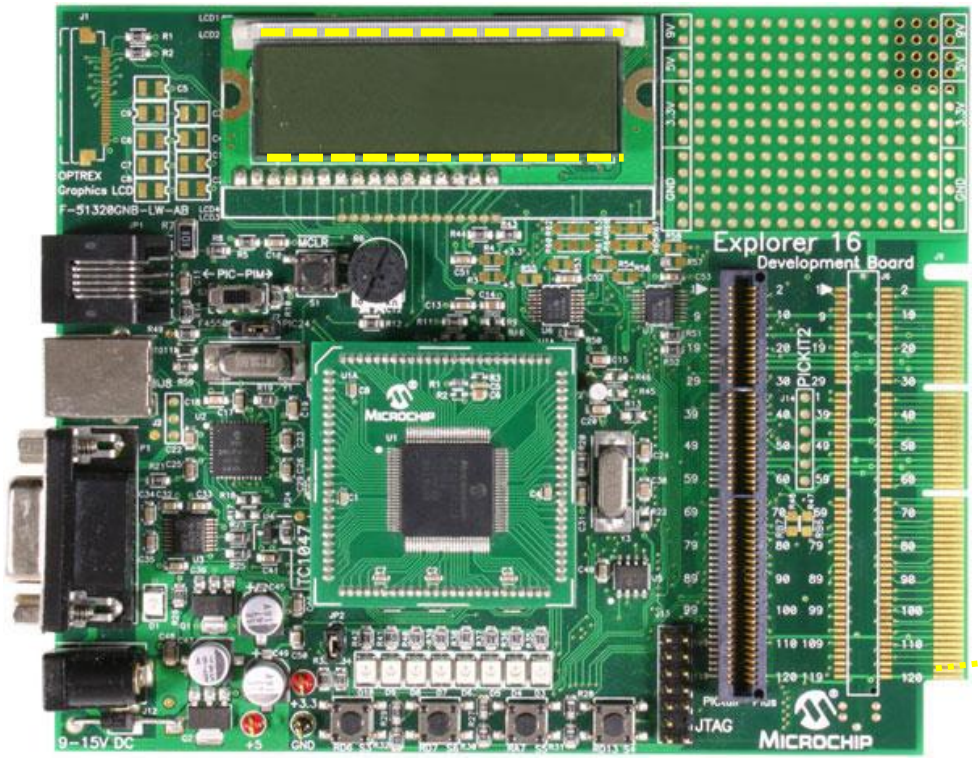
# EDGE DETECTION BY PARAMETRIC MODELS

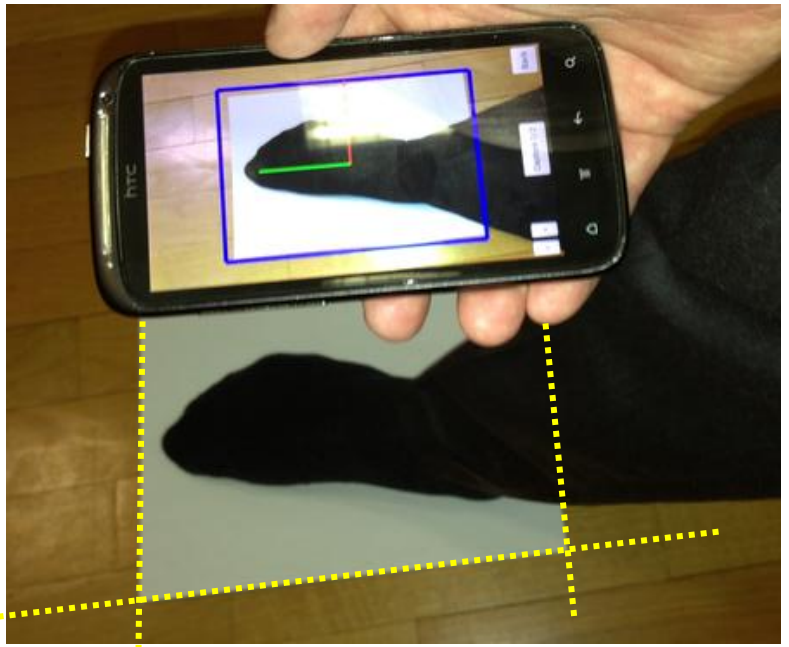
# Example: line fitting

- Why should we fit lines?

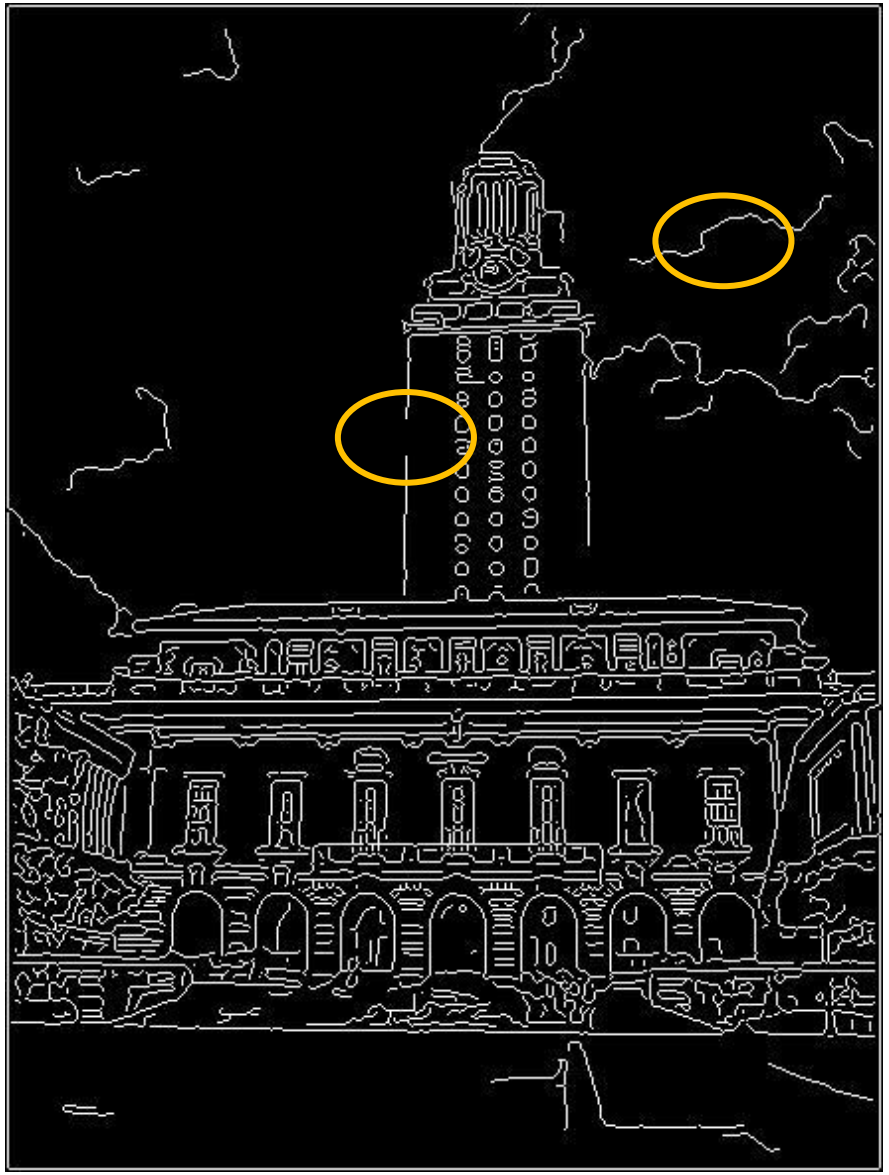
  Many scenes are composed of straight lines



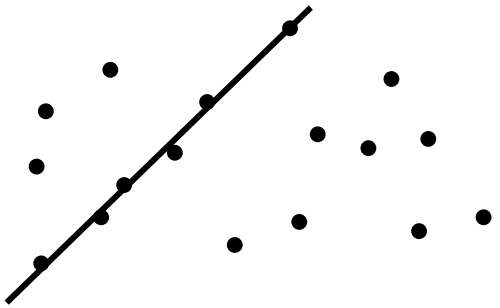


# Challenges of line fitting



- Noisy edges, multiple models:
  - Which points correspond to which line, if at all?
- Some parts of lines are not detected:
  - How to find a line that connects the missing points?
- Noisy orientation:
  - How do we determine the unknown parameters of true lines?

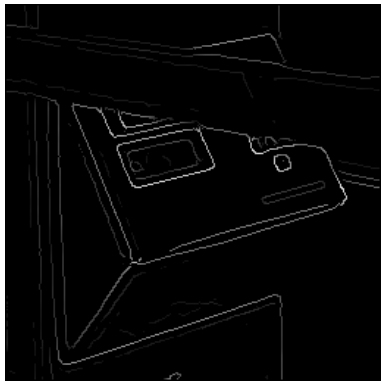


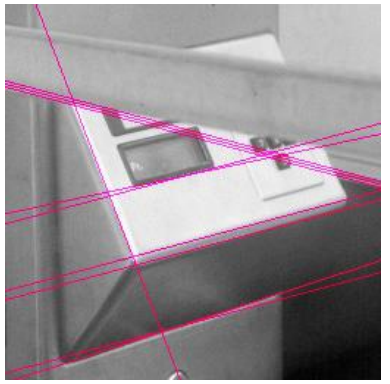
$y = 1.6x + 21.5$

# Line fitting by voting for parameters

- Given a set of points, find the lines.
- How many lines?
- Which points correspond to which lines?

- *Hough Transform* is a voting technique that answers these questions.
- Main idea:
  1. For each edge point compute parameters of all possible lines passing through that point
  2. For each set of parameters cast a vote
  3. Select the lines (parameter combinations) that receive enough votes.





# Hough space: straight lines

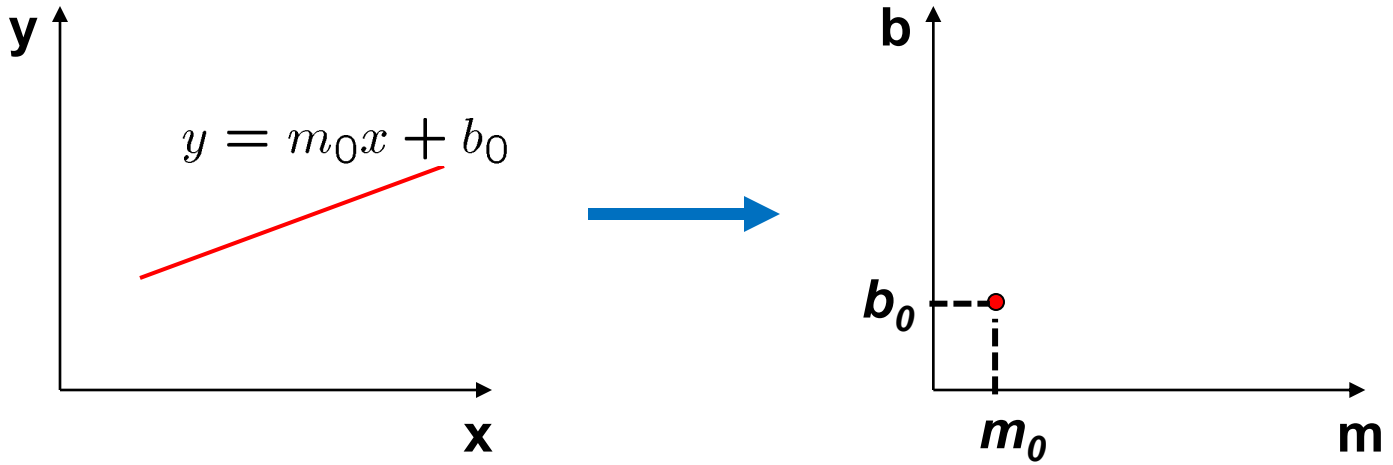
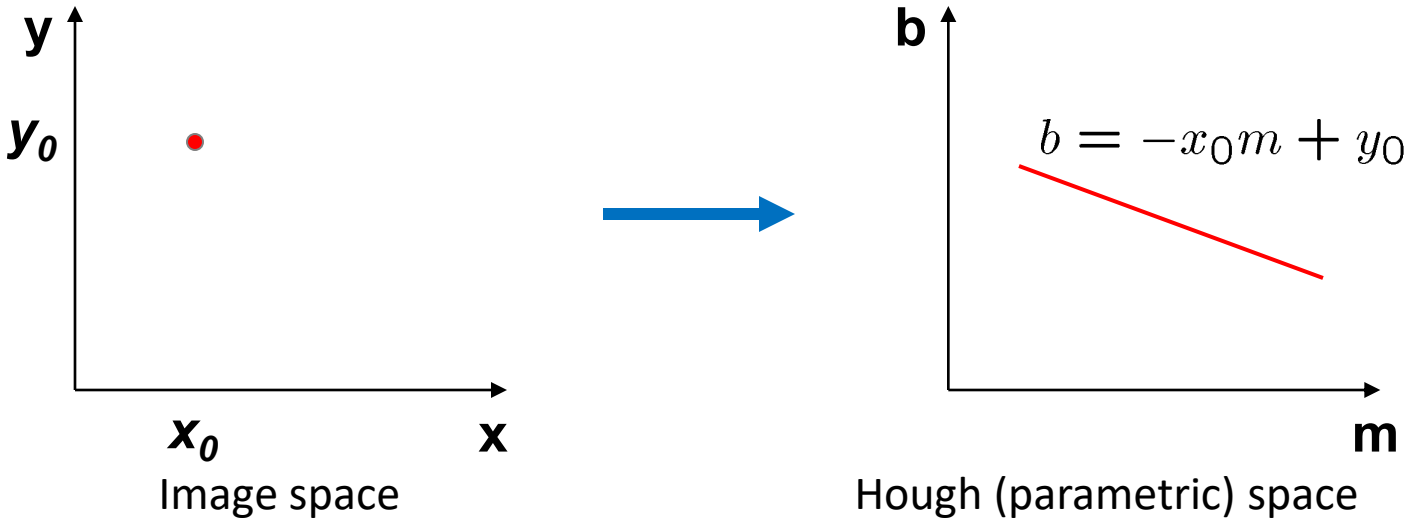



Image space

Hough (parametric) space

- Connection between spatial $(x,y)$ and Hough space $(m,b)$:
  - A line in image corresponds to a point in the Hough space.

# Hough space: straight lines




- Connection between spatial $(x,y)$ and Hough space $(m,b)$:
  - A line in image corresponds to a point in the Hough space.
  - Mapping from image to Hough space:
    - For a point $(x,y)$, find all $(m,b)$ for which this holds : $y = mx + b$

# Hough space: straight lines

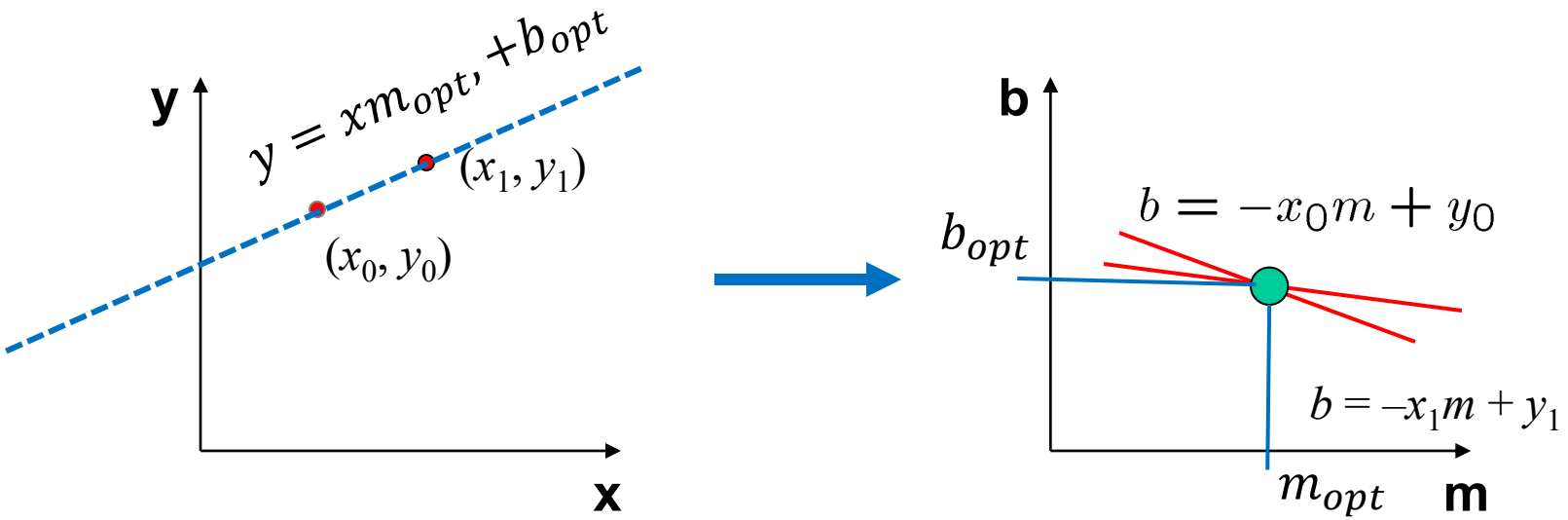



Image space

Hough (parametric) space

- Connection between spatial $(x,y)$ and Hough space $(m,b)$:
  - A line in image corresponds to a point in the Hough space.
  - Mapping from image to Hough space:
    - For a point $(x,y)$, find all $(m,b)$ for which this holds : $y = mx + b$

# Hough space: straight lines

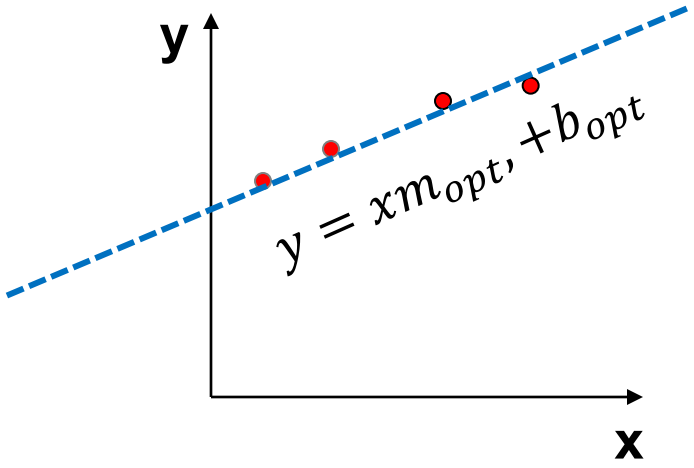
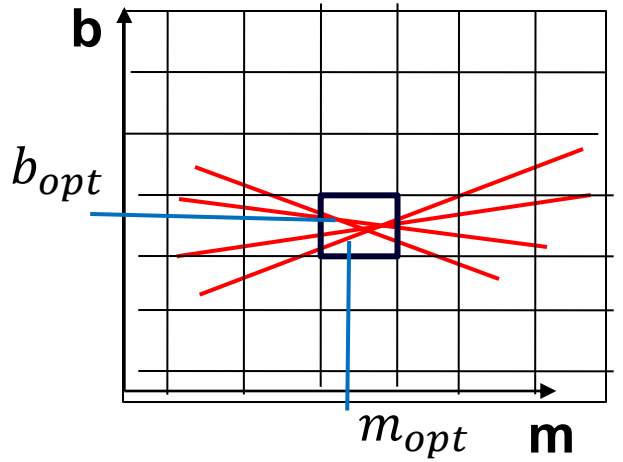



Image space



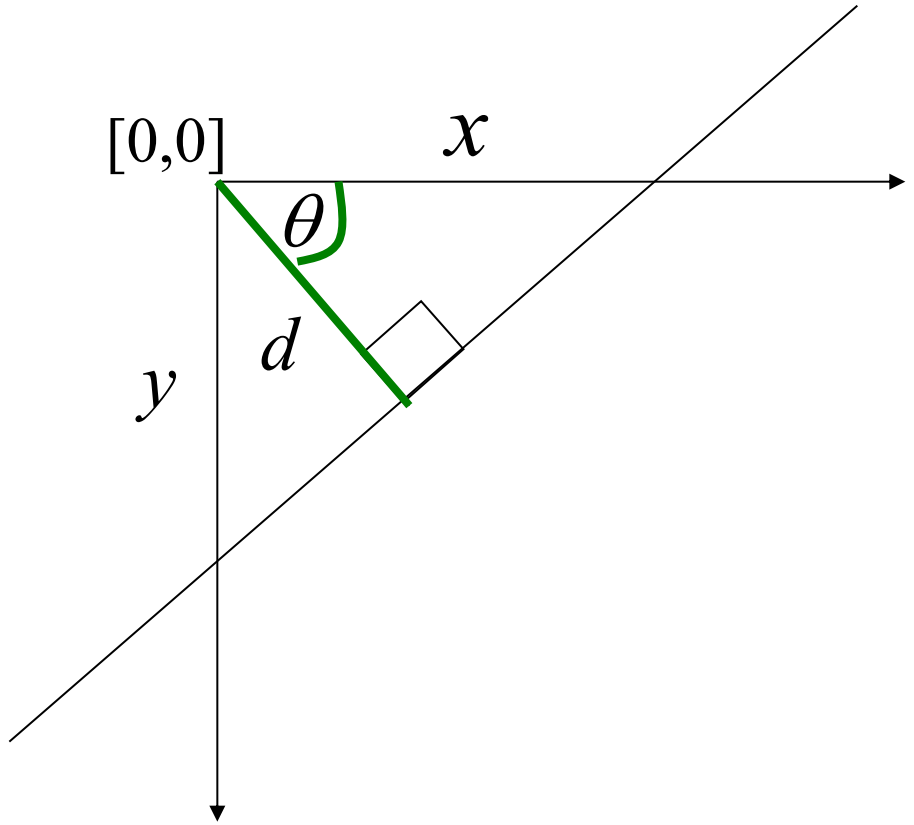

Hough (parametric) space

Discretize the parameter space…

- Connection between spatial $(x,y)$ and Hough space $(m,b)$:
  - A line in image corresponds to a point in the Hough space.
  - Mapping from image to Hough space:
    - For a point $(x,y)$, find all $(m,b)$ for which this holds : $y = mx + b$

# Encode the line in polar coordinates

- Issue with Cartesian $(m,b)$: infinite values for vertical lines!




$d$ : perpendicular distance from the origin

$\theta$: angle of perpendicular line with x axis

$$x\cos\theta - y\sin\theta = d$$

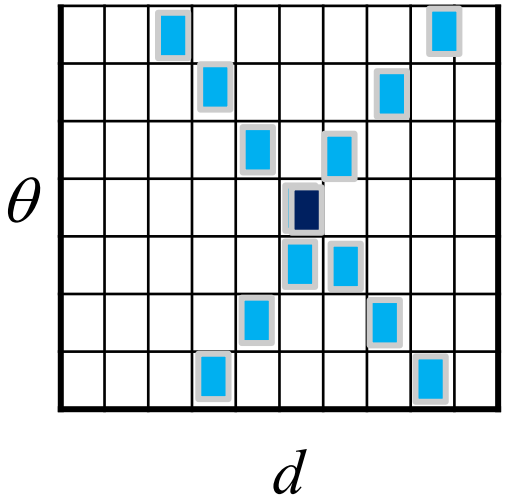
- Point in image $\Rightarrow$ sinusoid in Hough space

# Algorithm: Straight lines

Using polar representation:

$$x\cos\theta - y\sin\theta = d$$

$H$: accumulator array (votes)



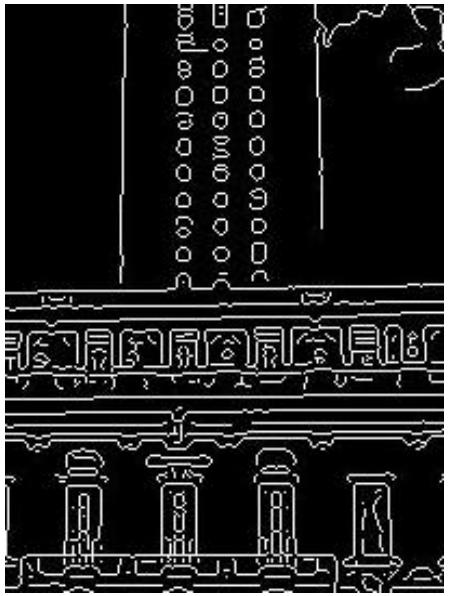

Basic Hough transform:

1. Initialize $H[d,\theta] = 0$.
2. For each edge point $(x,y)$ in image

   For $\theta = 0$ to $180$ // over quantized values!!

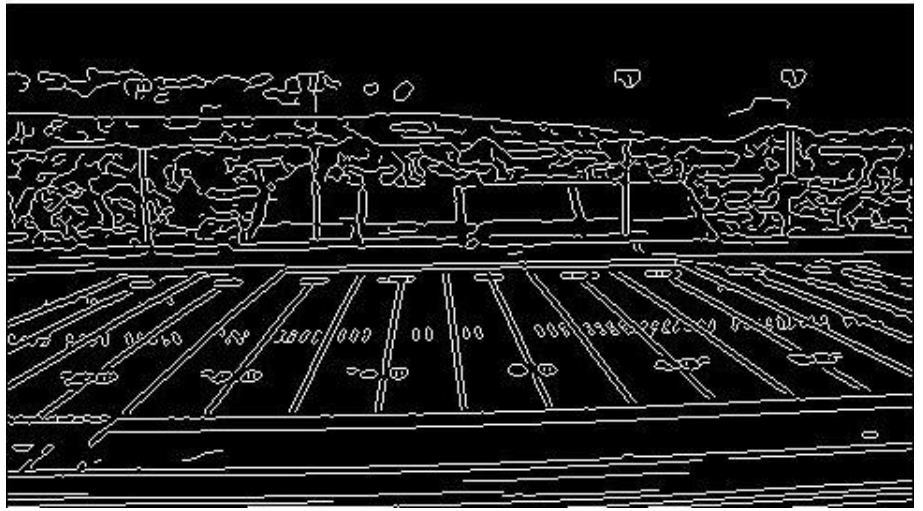
   $$d = x\cos\theta - y\sin\theta$$

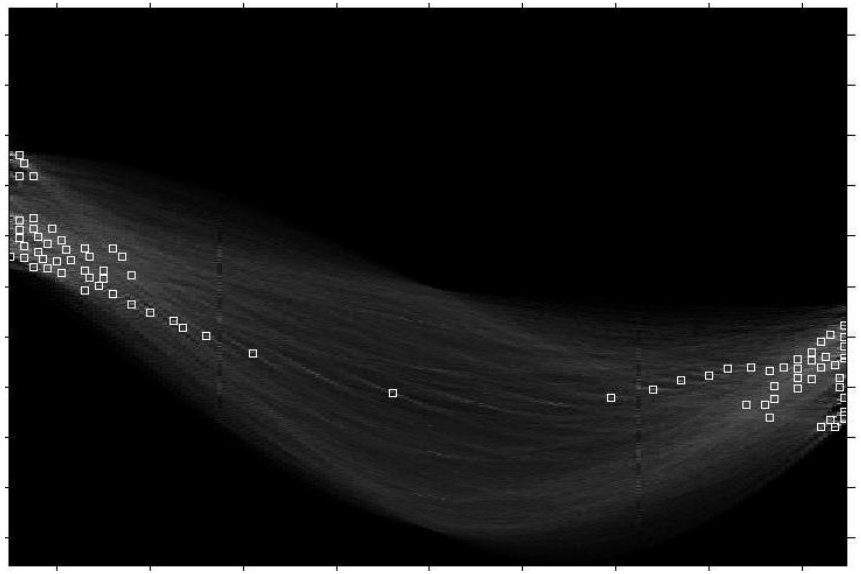
   H[d, θ] += 1
3. Find local maxima $\{d^i_{opt}, \theta^i_{opt}\}_{i=1:N}$ in accumulator array $H[d,\theta]$.
4. Detected line is defined by: $d^i_{opt} = xcos\theta^i_{opt} - sin\theta^i_{opt}$

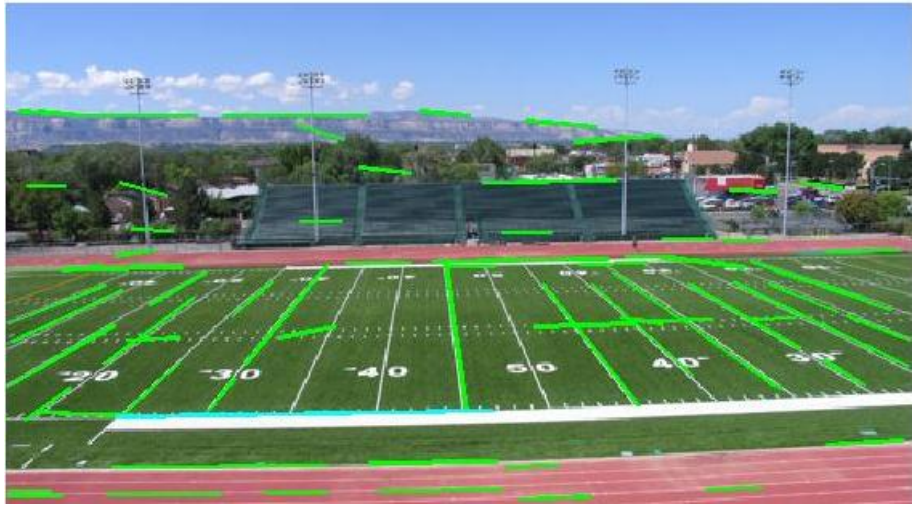


Hough line demo

# Hough transform in action

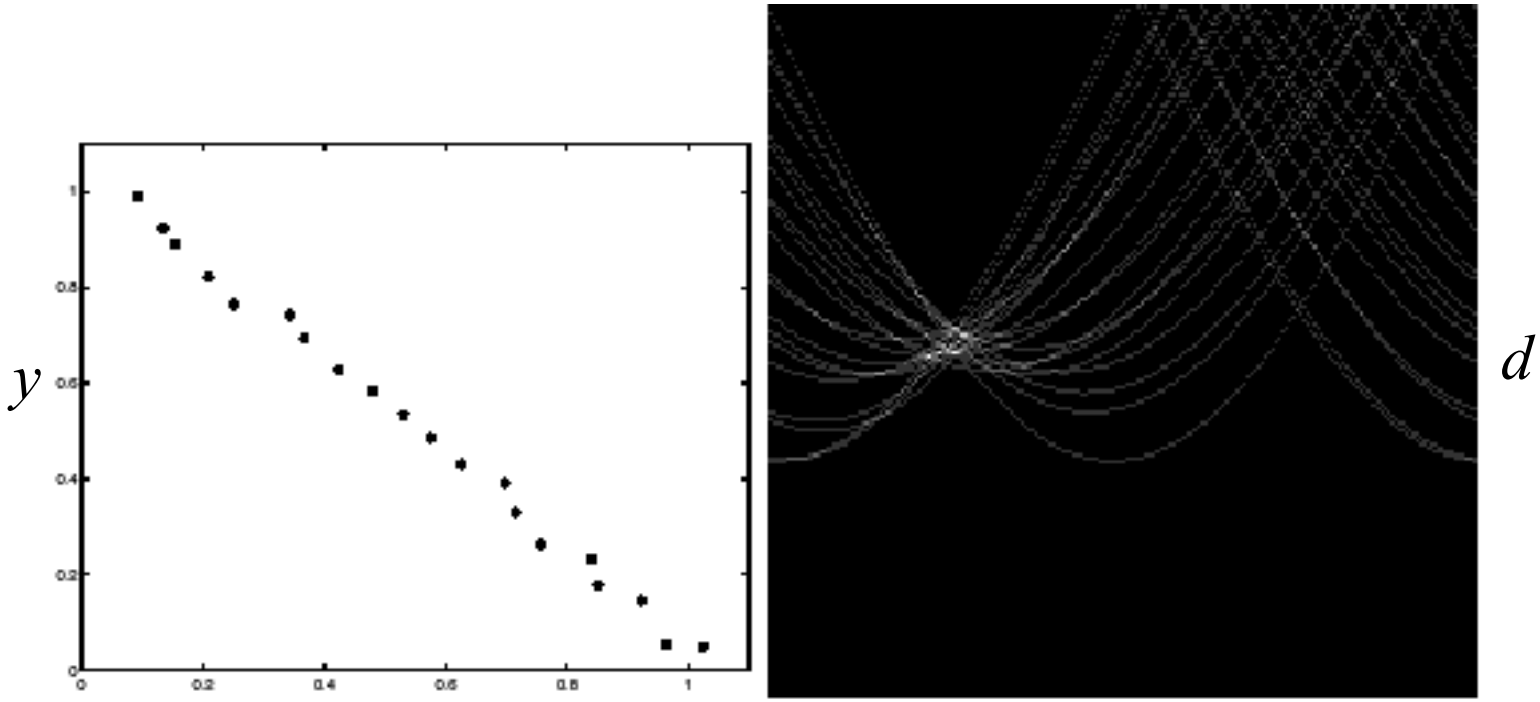






Only the longest segments along each detected line are shown here.

# Hough transform: Noise – binning

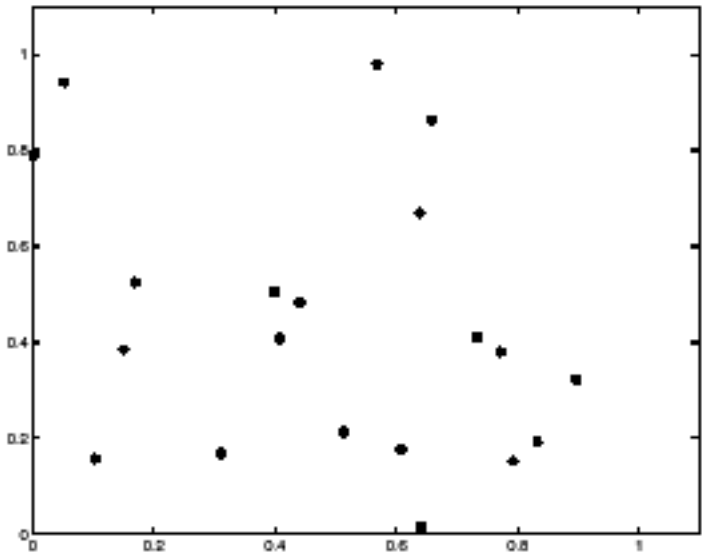



Coordinates of edge points in image

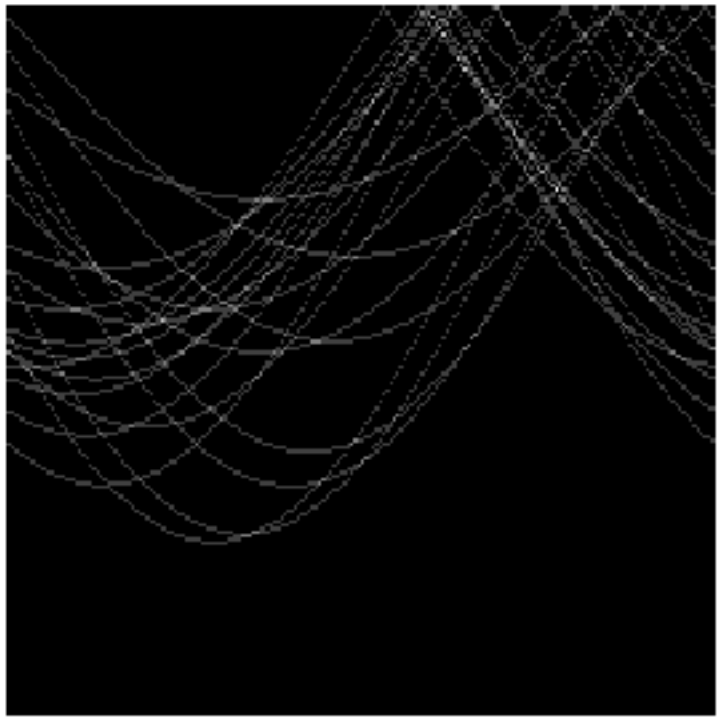
Votes

Are there any significant problems with the noise?

# Hough transform: Noise – amplitude of votes




Coordinates of
edge points in image



Votes

Random points still form some local maxima in the accumulator array!

# Hough transform: Extensions

Extension 1: Use the gradient *direction!*

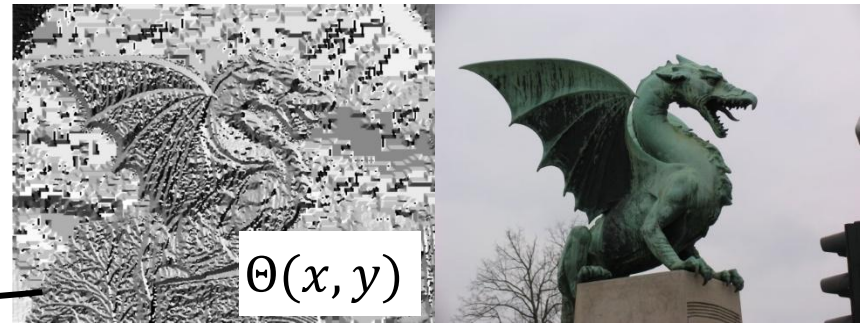
1. *same as standard HT*
2. For each edge point $[x,y]$

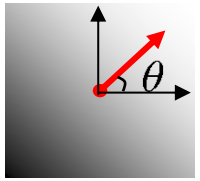
   $\theta$ = gradient direction at $(x,y)$

   $d = x\cos\theta - y\sin\theta$

   $\mathrm{H}[d,\theta]$ += 1
3. *same as standard HT*
4. *same as standard HT*

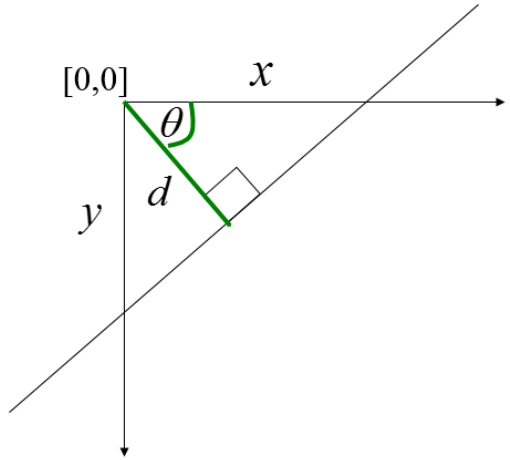
Reduces the number of degrees of freedom (dof)!




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

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



# Hough transform: Extensions

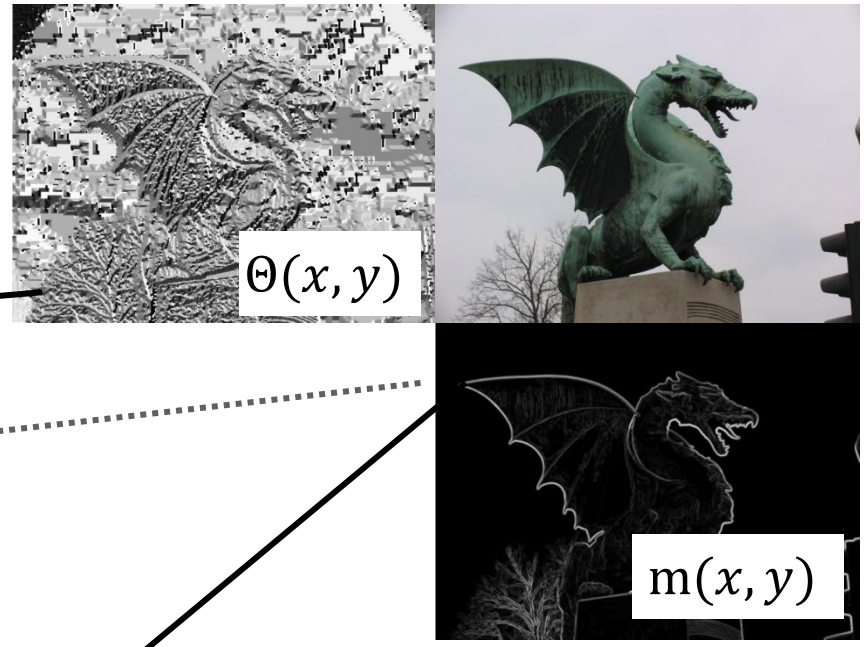
Extension 1: Use the gradient *direction!*

1. *same as standard HT*
2. For each edge point $[x,y]$

   $\theta$ = gradient direction at $(x,y)$

   $d = x\cos\theta - y\sin\theta$

   $\mathrm{H}[d,\theta]$ += 1
3. *same as standard HT*
4. *same as standard HT*




Extension 2:

- Assign higher weight in votes to points with large edge magnitude.
  Instead $\mathrm{H}[d,\theta]$ += 1, use $\mathrm{H}[d,\theta]$ += m(x,y).

- These extensions can be applied in general:
  line, circles, squares, general shapes...

# Hough transform for circles

- Circle parameters: center $(a,b)$ and radius $r$

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- Example of center detection at known radius $r$

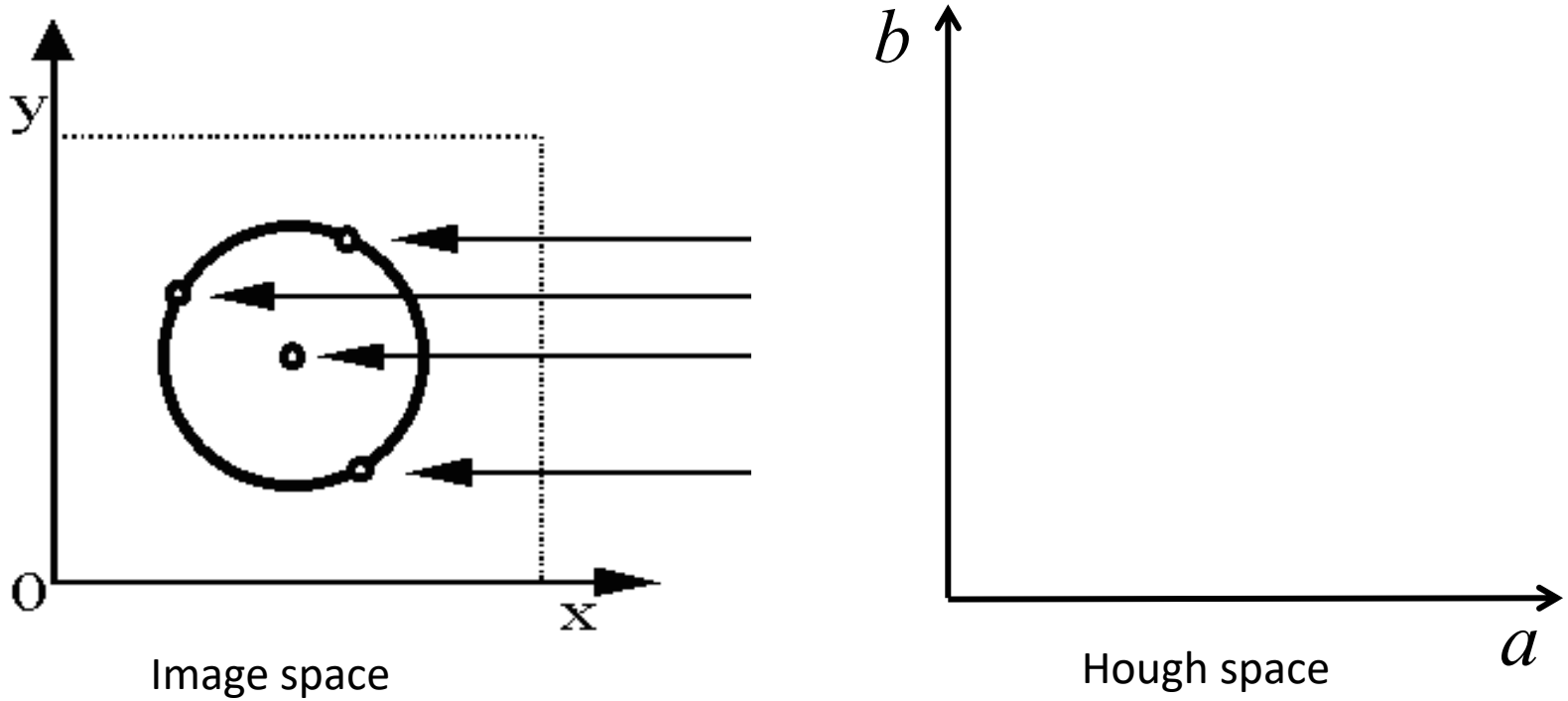



Image space

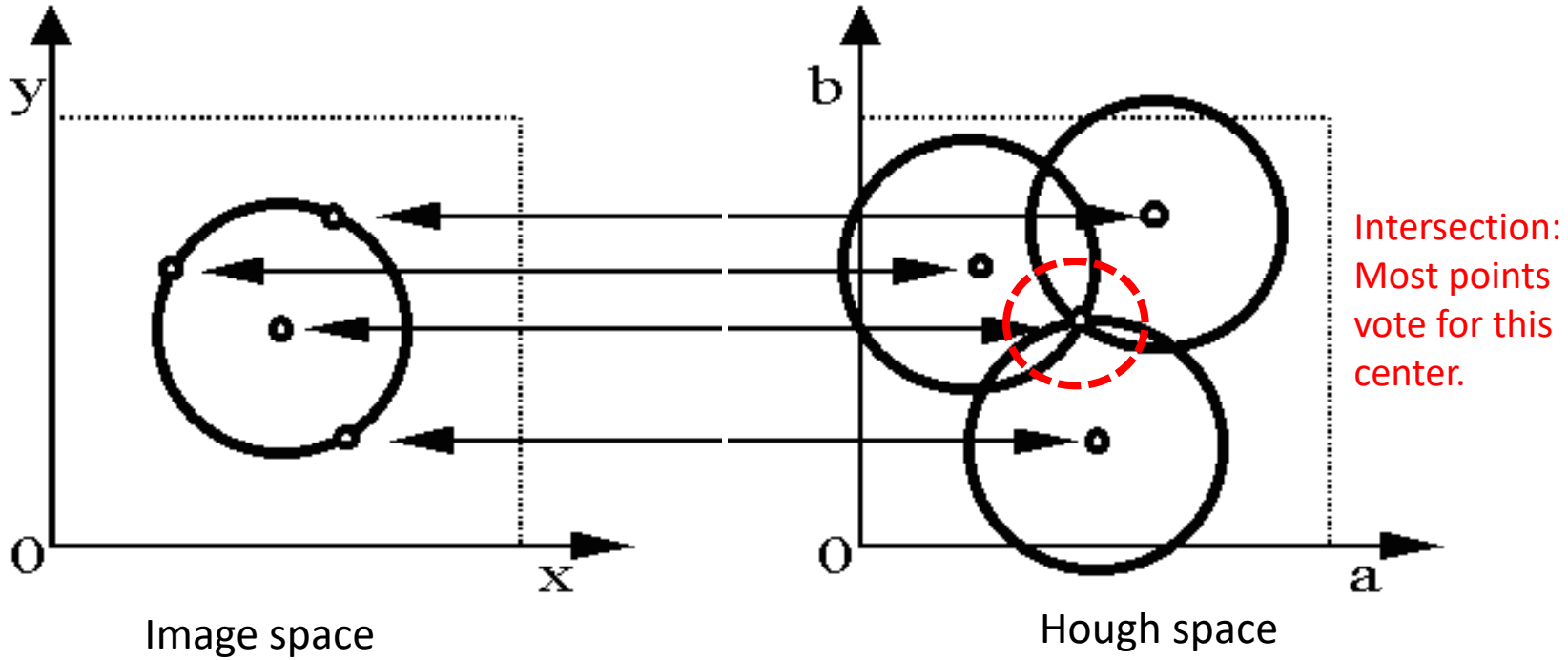
Hough space

# Hough transform for circles

- Circle parameters: center $(a,b)$ and radius $r$

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- Example of center detection at known radius $r$

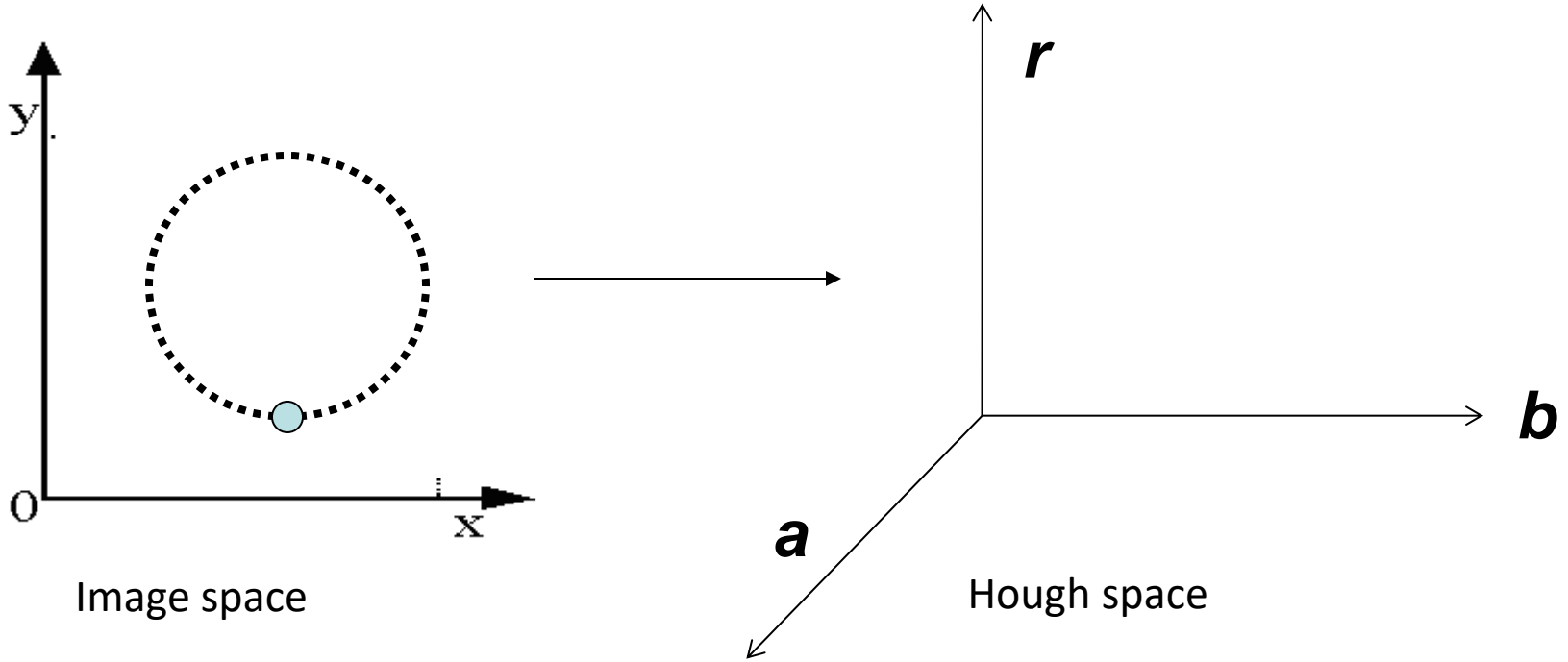


# Hough transform for circles

- Circle parameters: center $(a,b)$ and radius $r$

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- Unknown radius $r$ – *How many dimensions in Hough Space?*



# Hough transform for circles

- Circle parameters: center $(a,b)$ and radius $r$

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- Unknown radius $r$

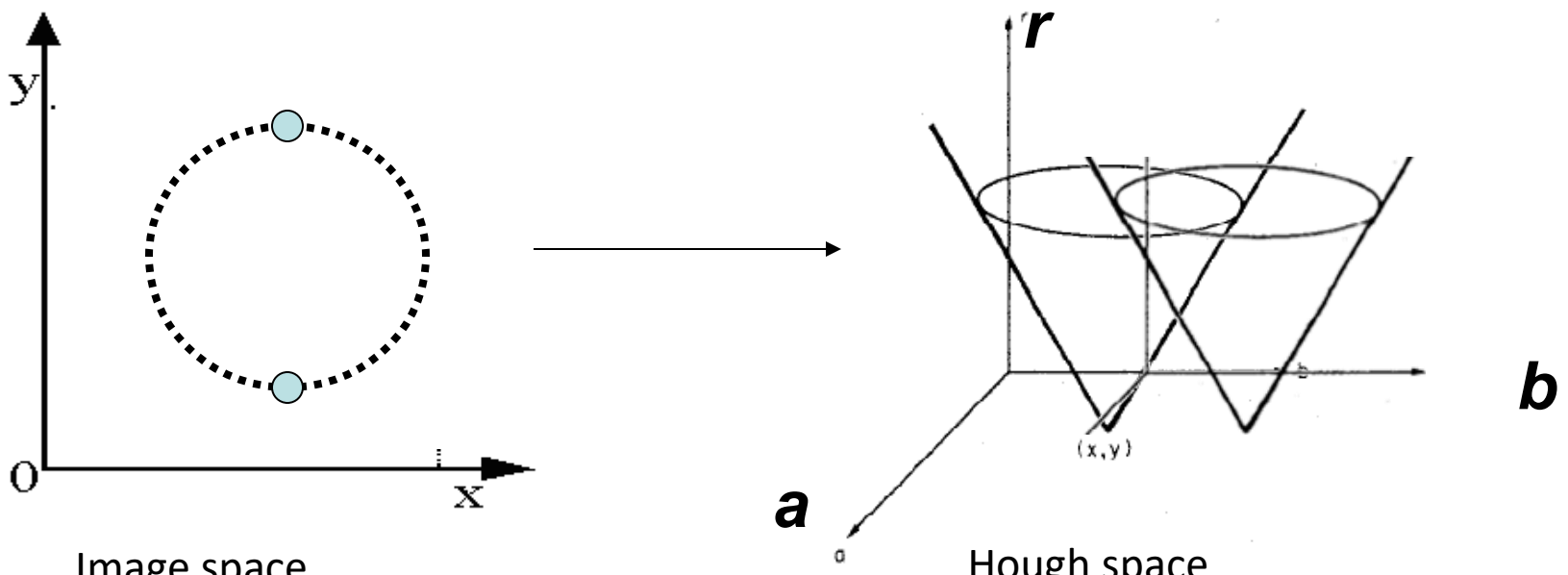



Image space

Hough space

# Hough transform for circles

- Circle parameters: center $(a,b)$ and radius $r$

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- Unknown radius $r$

But assume we know the gradient direction!

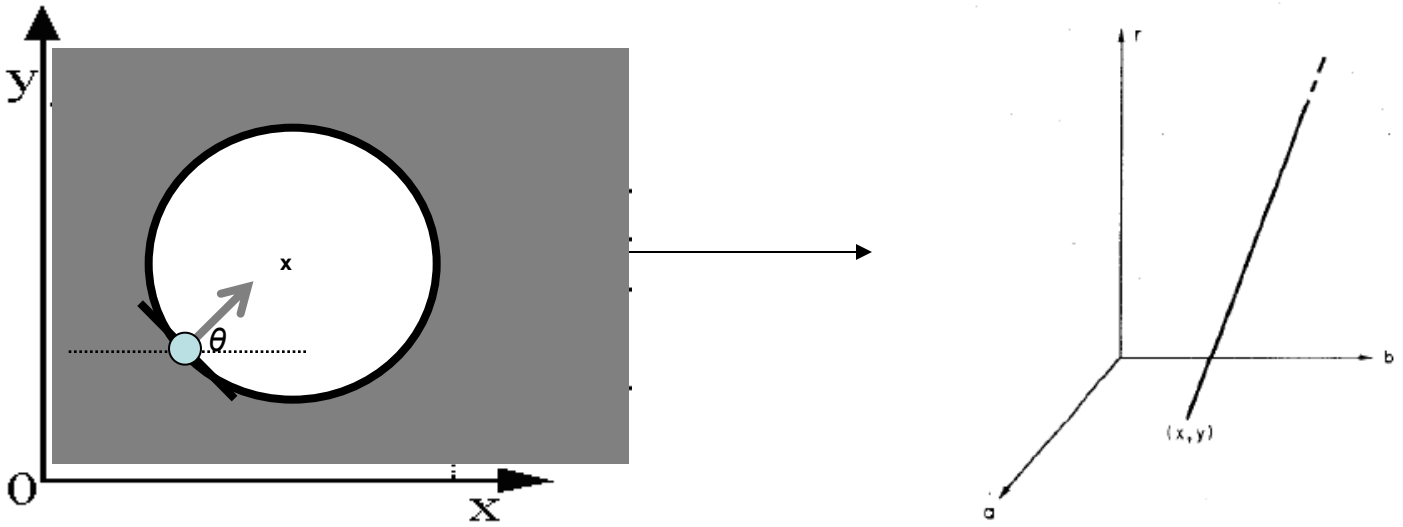



Image space

Hough space

# Hough transform for circles

For each edge pixel $(x,y)$ :

For each radius value $r$:

For each gradient direction $\theta$:

*// or use the estimated direction only*

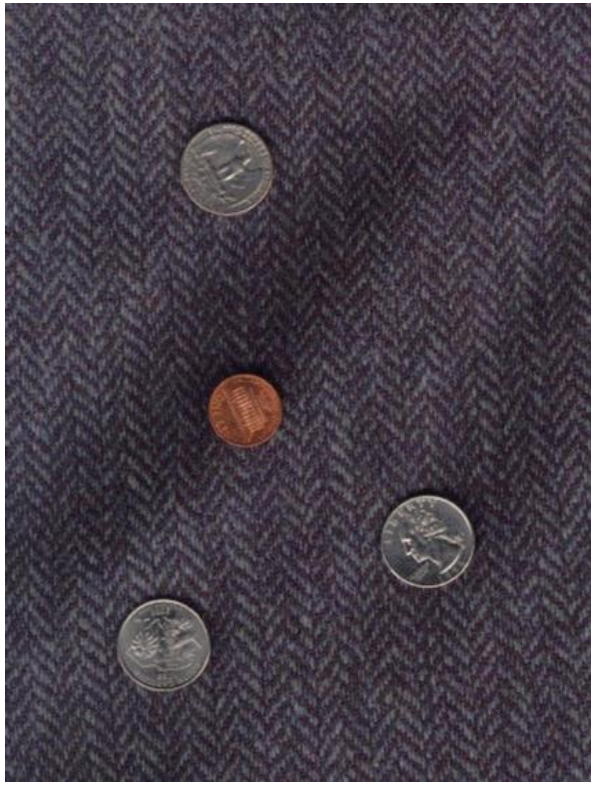
$$a = x - r\cos(\theta)$$

$$b = y + r\sin(\theta)$$

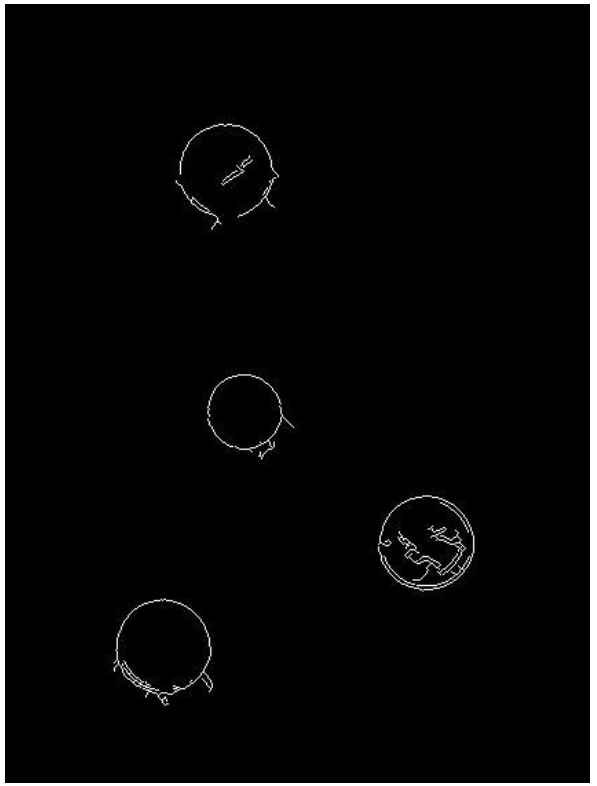
$$H[a,b,r] \mathrel{+}= 1 \text{ (or the magnitude)}$$

# Hough circle detection in action!

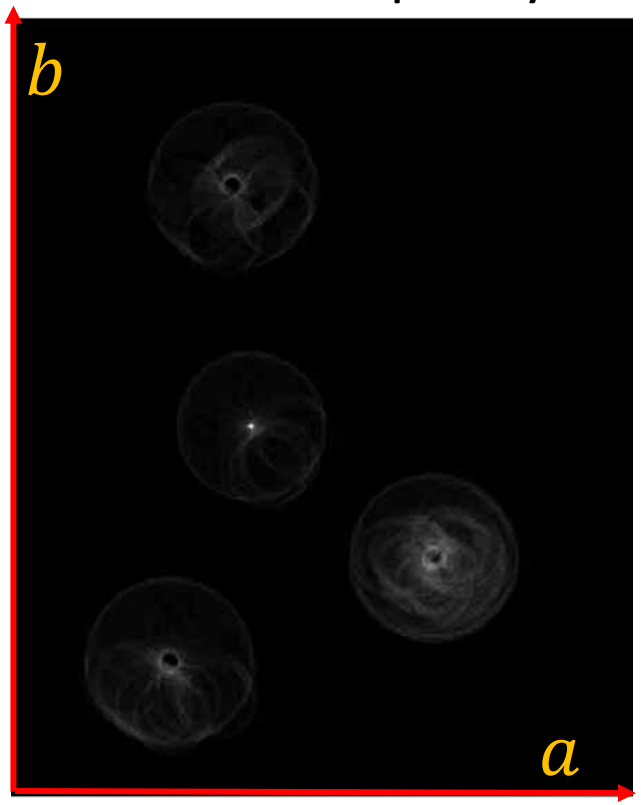
Original



Edges (e.g., Canny)



Votes for penny




$(x_i - a)^2 + (y_i - b)^2 = r^2$

Given a known radius:

$r = 50\ pixels,$

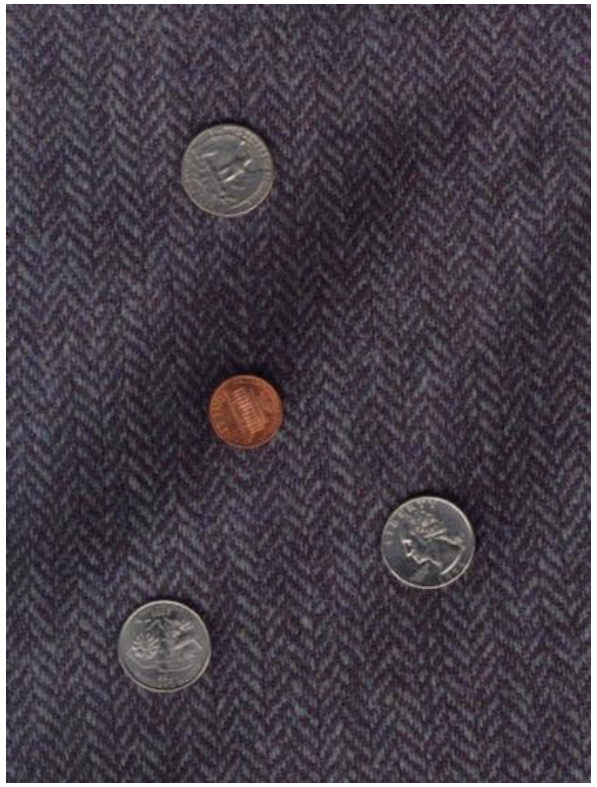
what are the center coordinates, i.e.,

$a = ?\ pixels,$
$b = ?\ pixels\ .$

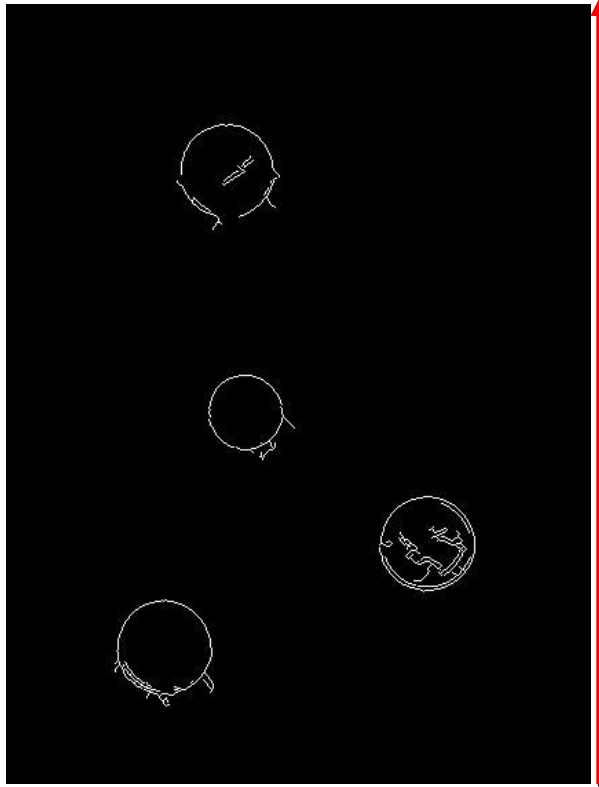
Comment: here we use a separate HT for each coin size.

# Hough circle detection in action!

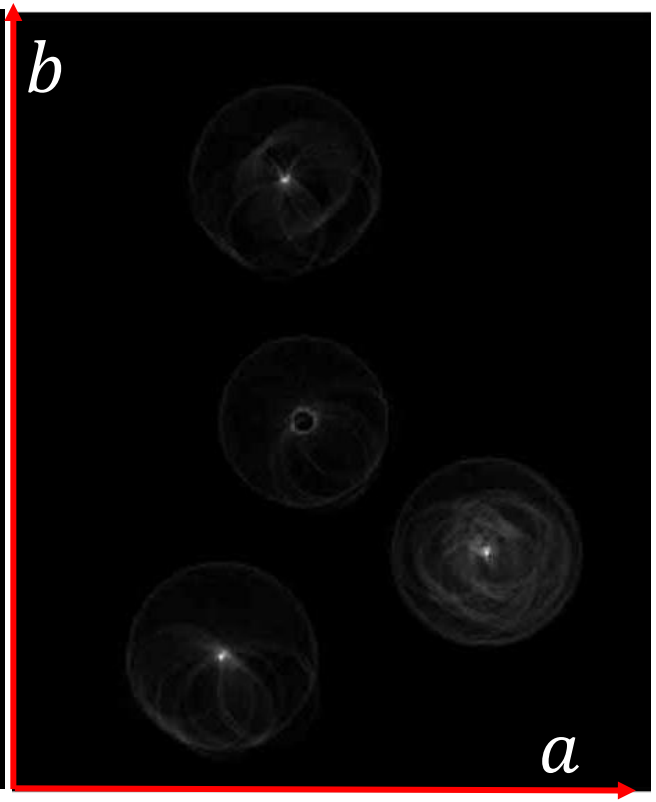
Original



Edges (e.g., Canny)



Votes for 25cent




$(x_i - a)^2 + (y_i - b)^2 = r^2$

Given a known radius:

$r = 80\ pixels,$

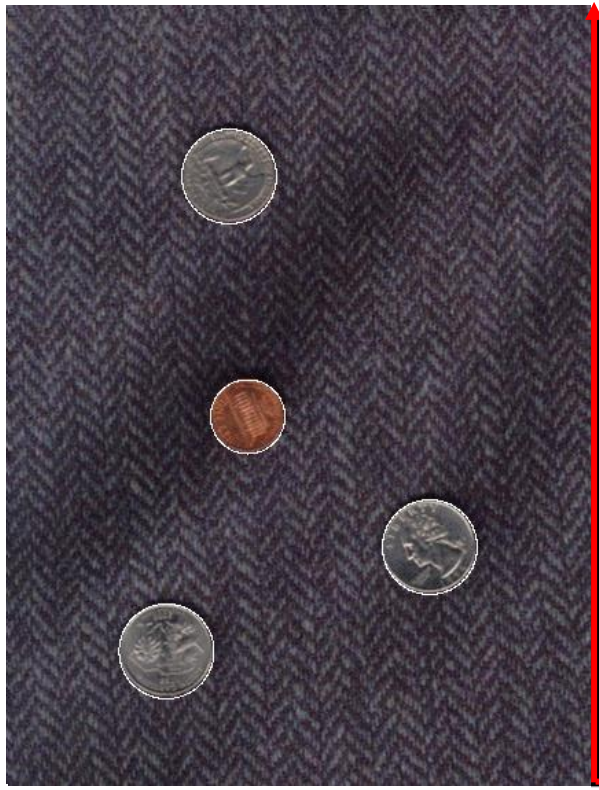
what are the center coordinates, i.e.,

$a = ?\ pixels,$
$b = ?\ pixels\ .$

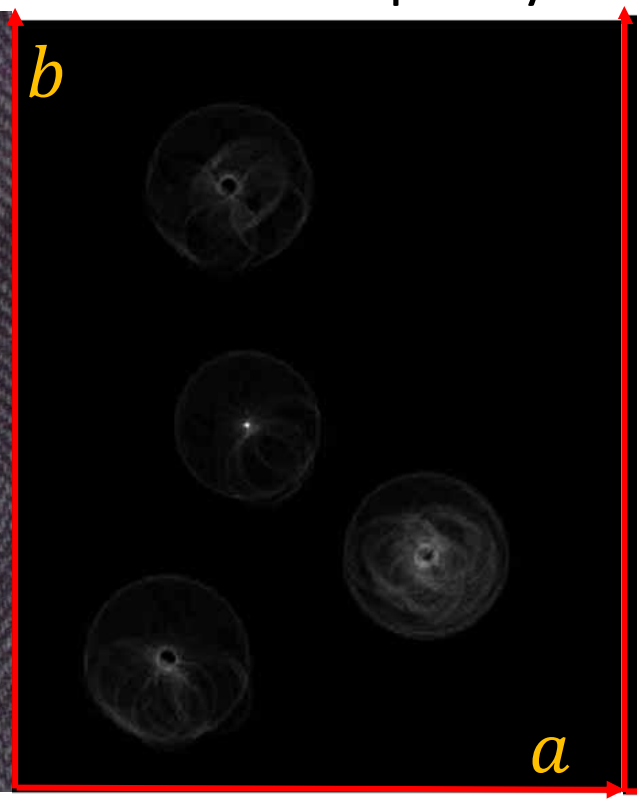
Comment: here we use a separate HT for each coin size.

# Hough circle detection in action!

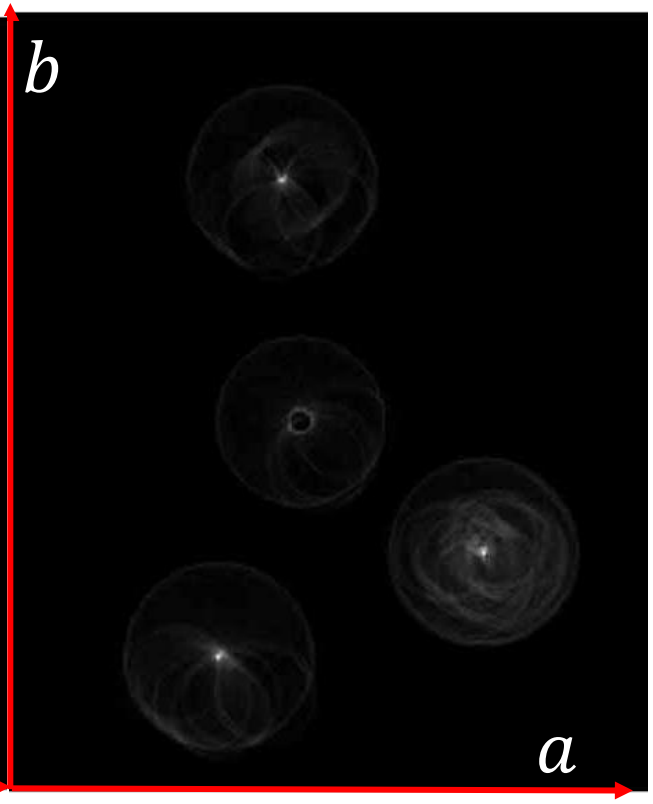
Combined detections



Votes for penny




Votes for 25cent




$(x_i - a)^2 + (y_i - b)^2 = r^2$

Given a known radius:

$r = 50\ or\ 80\ pixels,$

what are the center coordinates, i.e.,

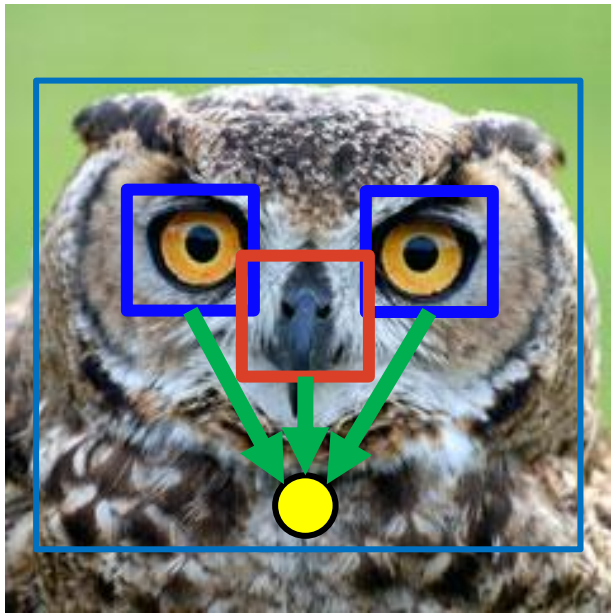
$a = ?\ pixels,$
$b = ?\ pixels\ .$

Comment: here we use a separate HT for each coin size.

# Generalized Hough transform (GHT)

Building a model to detect objects by GHT – intuition:

- Assume we know how to detect parts (recognize+localize), i.e., eyes and beak of an howl. Task: *create an howl head detector.*
- Encode parts by displacements to the neck center.



The owl head model:

*Given a part, where is the neck center?*

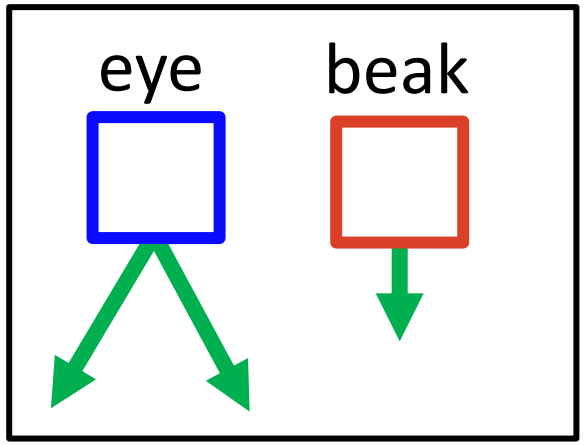
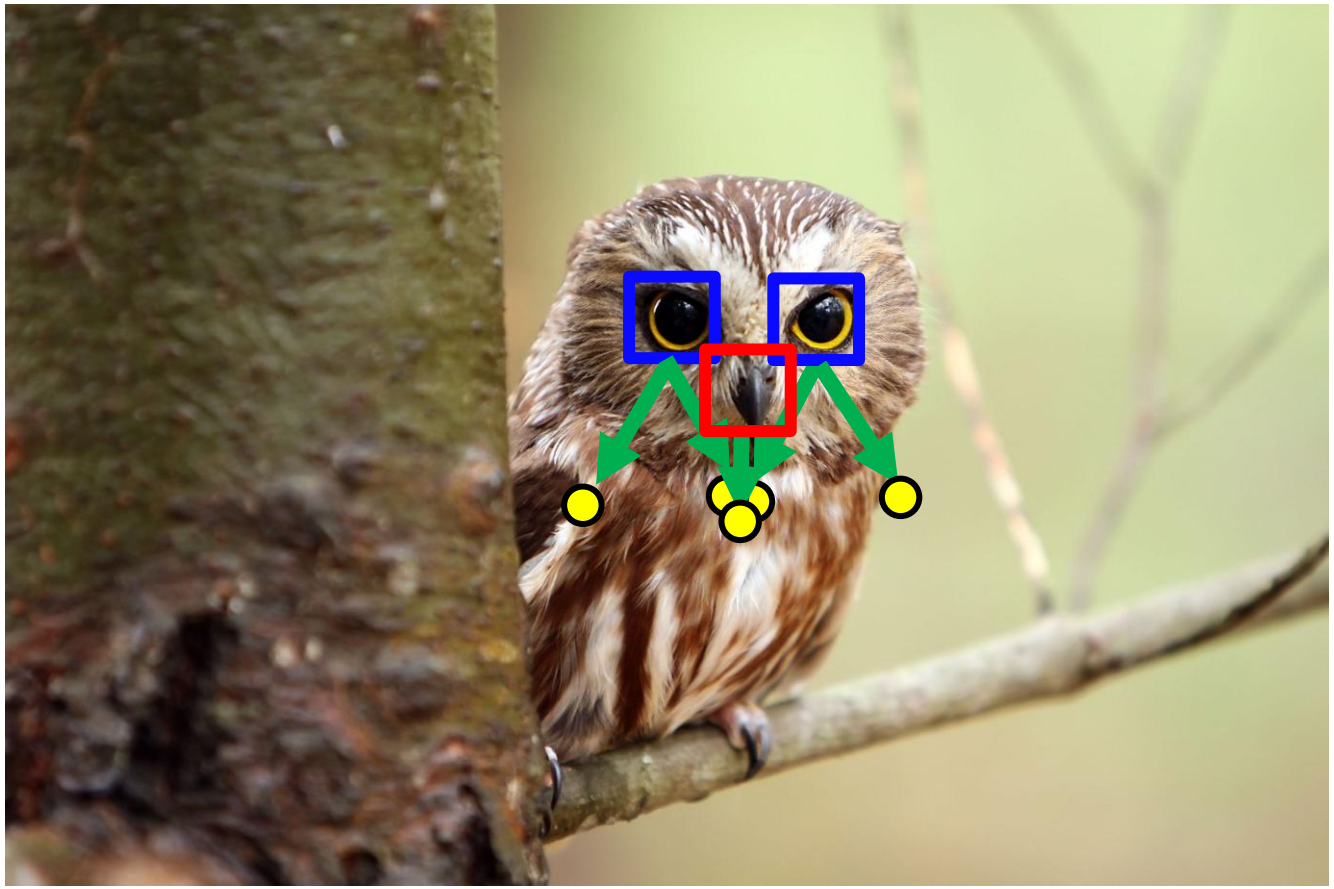


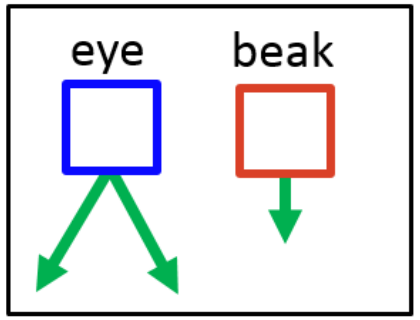

Image from: http://www.exposureguide.com/

# Generalized Hough transform (GHT)

- Detection – intuition



The owl head model:



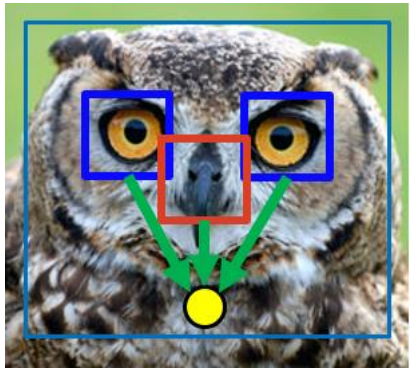
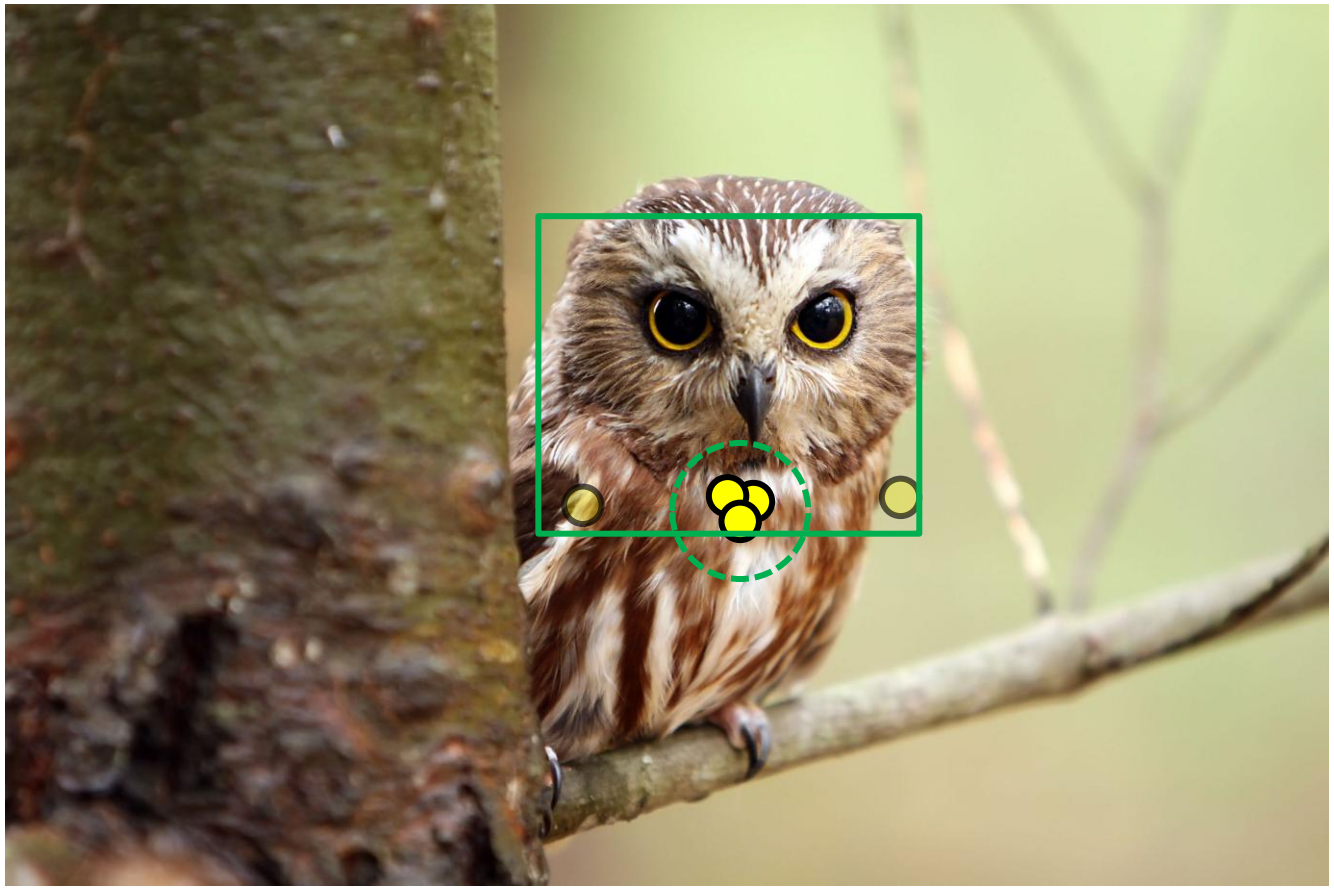
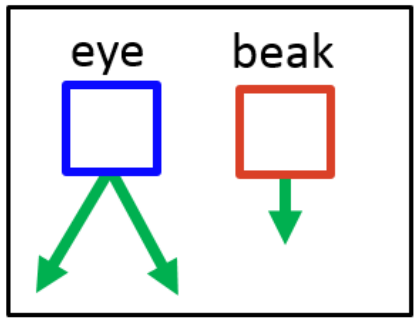



Image from: http://pugetsoundbirds.org/projects/owl-project/

# Generalized Hough transform (GHT)

- Detection – intuition





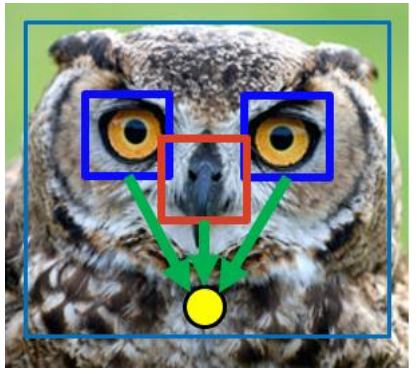
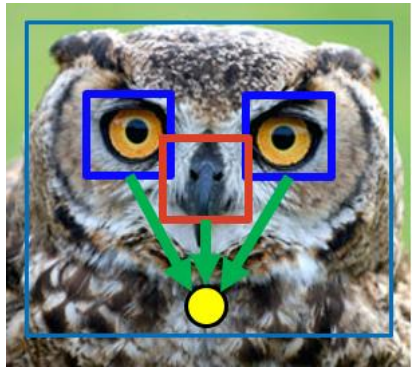
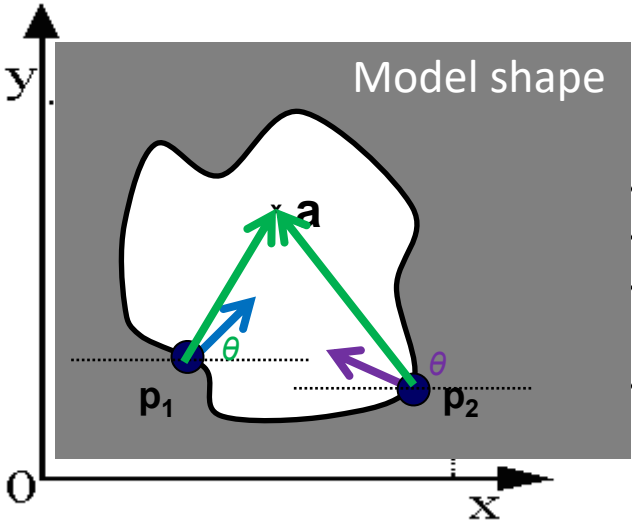



Image from: http://pugetsoundbirds.org/projects/owl-project/

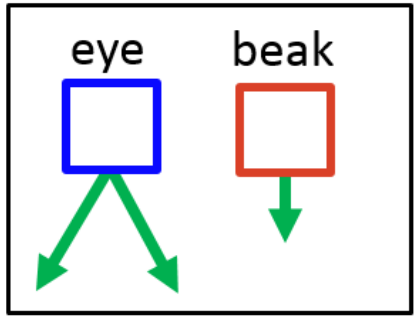
# GHT for shape-based models

- Define the shape model by edge points and a reference point.






The owl head model:




Edge/gradient direction | Voting vector

| | |
|---|---|
| $\theta$ | ... |
| $\theta$ | ... |
| ⋮ | |

Model learning:

For each edge point calculate the displacement vector to the reference point:
$\mathbf{r = a - p_i}$.

Collect displacements in table, indexed by gradient direction $\theta$.

[Dana H. Ballard, Generalizing the Hough Transform to Detect Arbitrary Shapes, 1980]

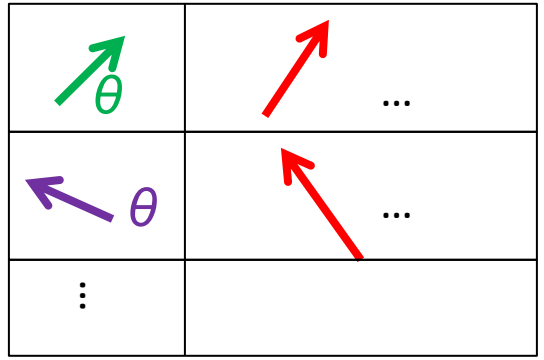
Slide credit: Kristen Grauman

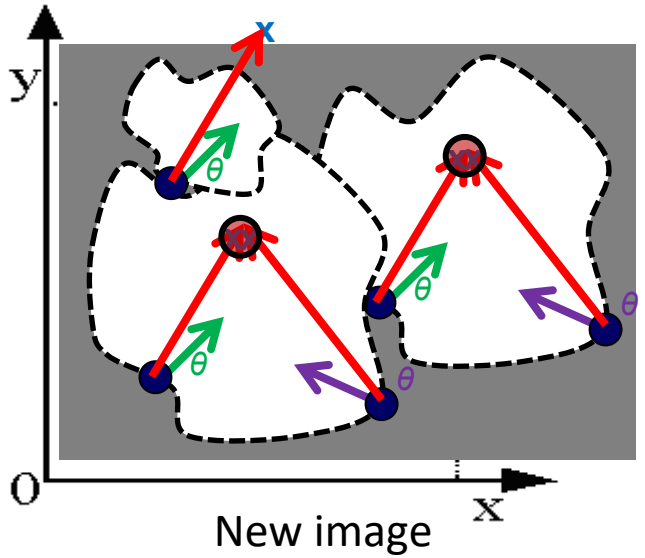
# GHT for shape-based models

## Detection:

For each edge point:

- Use its gradient orientation θ to index into the table.
- Use the displacement vectors **r** to cast a vote for the center.



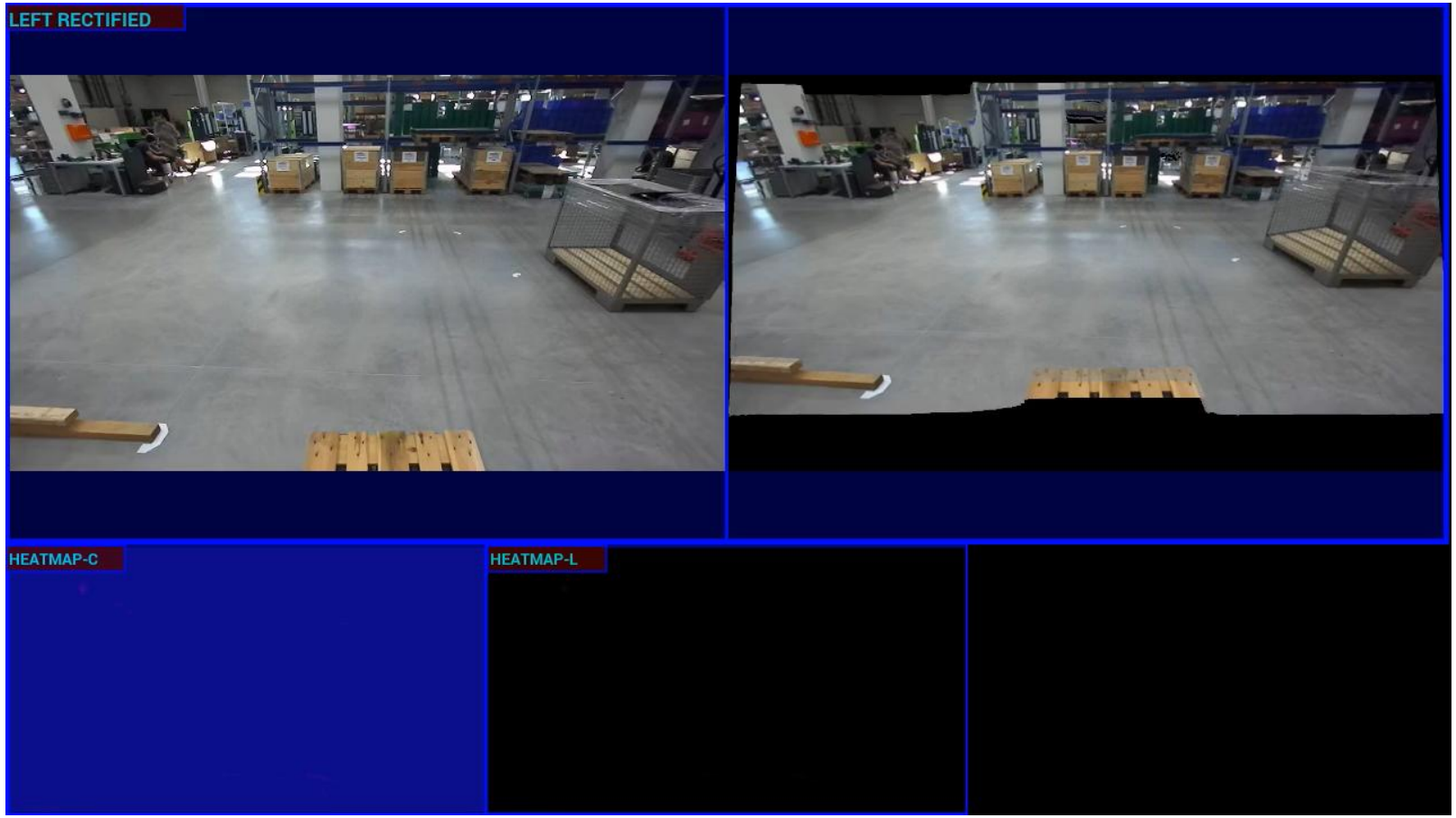
New image

*Assumption: the only transformation is the translation (orientation+scaling are fixed)*

# GHT in practice

Csaba Beleznai, et al., Automated pallet handling via occlusion-robust recognition learned from synthetic data, CAI2023

- Approach: Use a simulated 3D model of an object (automatically annotated) and train a neural network that predicts the 3D object center for each point



# Hough transform line detection: Practical advices

- First minimize irrelevant responses
  (use only edges with significant magnitude of gradient)
- Appropriately discretize the parametric space
  - Too coarse: votes from different lines fall into the same accumulator
  - Too fine: losing lines – due to noise, collinear points cast votes into nearby (BUT DIFFERENT) accumulators.

- Vote for neighboring cells as well
  - Correct: cast a vote by a Gaussian or a bilinear interpolation
  - Approximate: convolve the voting array by a Gaussian
- Use the gradient direction to reduce the number of free parameters

# Hough transform: +/-

Pros

- Each point is processed independently:
  - robustness to partial occlusion,
  - highly parallelizable.
- Robustness to noise: noise will unlikely contribute consistently to a single cell
- Can detect multiple instances of a single model in one pass.

Cons

- Time complexity increases exponentially with the number of free parameters.
- Spurious shapes may generate false local maxima in the parametric space.
- Quantization: Not trivial to choose a proper accumulator cell size – Application dependent!

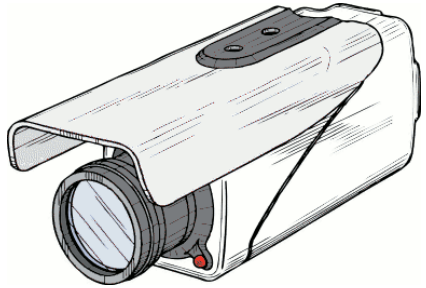
# References


- David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition), (prva izdaja dostopna na spletu)
- R. Szeliski,Computer Vision: Algorithms and Applications, 2010
- R. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, 2nd Edition, Cambridge University Press, 2004
- Kristen Grauman, „Computer Vision“, lectures

Univerza v Ljubljani

ViCOS visual cognitive systems lab

# Machine perception
# Fitting parametric models



Matej Kristan


Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

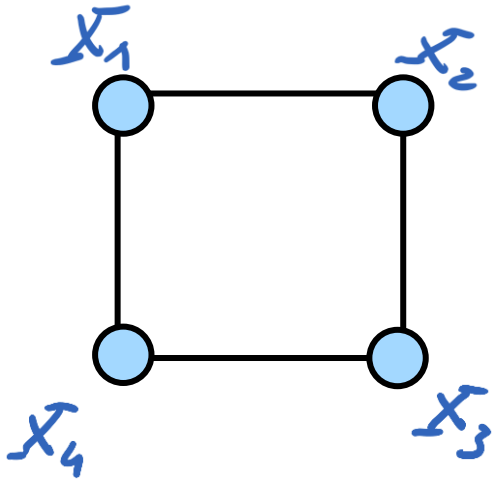
# Parametric models: Forward application

- Transformation parameterized by (many) parameters

$$\mathbf{x}_i' = f(\mathbf{x}_i; \mathbf{p})$$

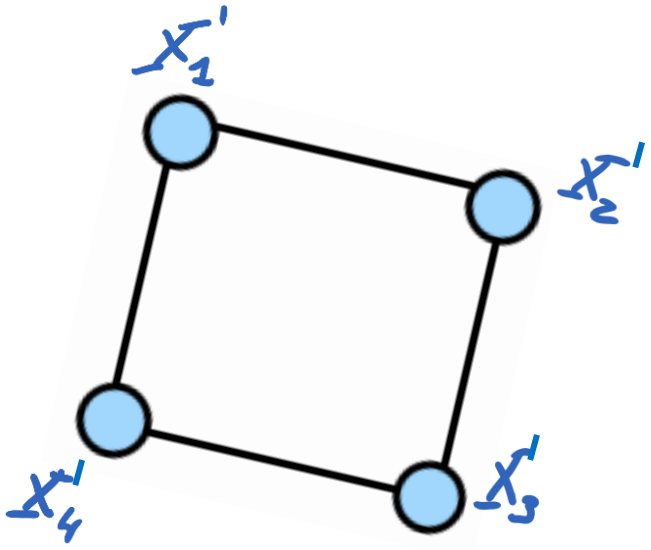
- Example: transform $\boldsymbol{x}_i$ into $\boldsymbol{x}_i'$ by a function $f(\boldsymbol{x}; \boldsymbol{p})$

$x_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix} \in \mathbb{R}^2$ $\quad p = [p_1, p_2, p_3, \cdots, p_M] \in \mathbb{R}^M$



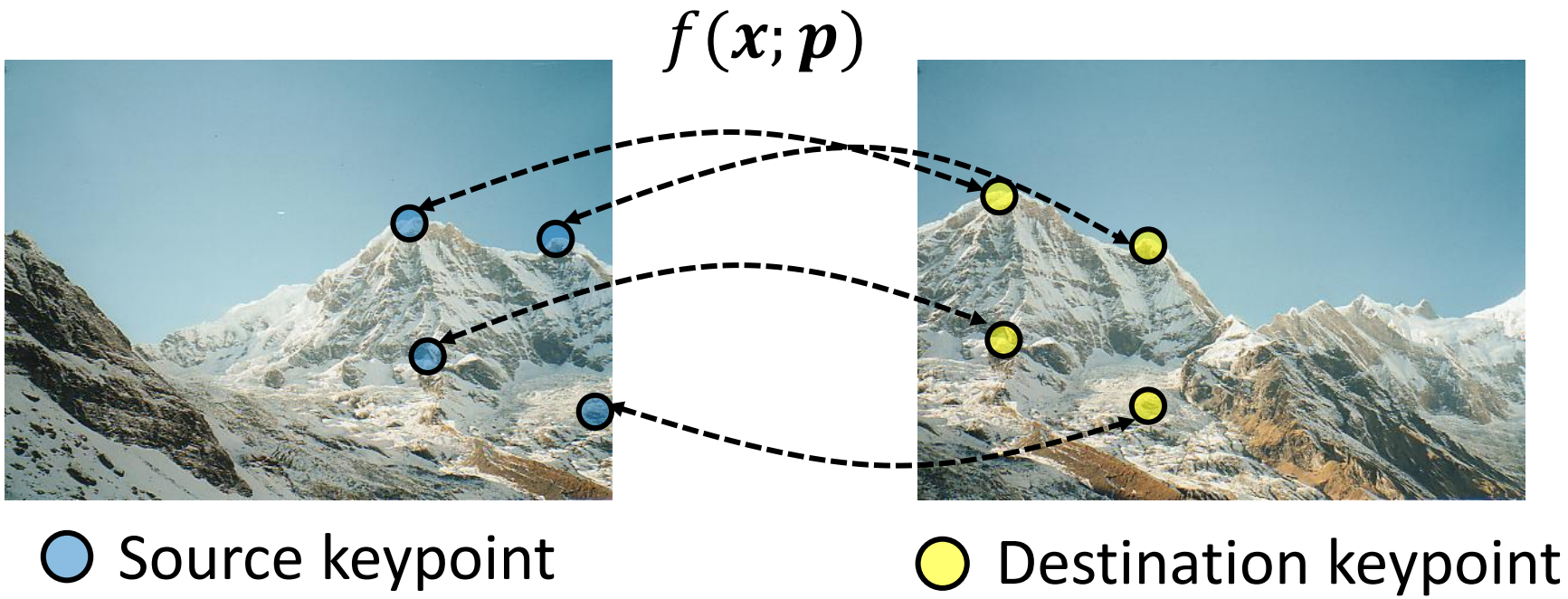

$f(\ ; p)$

$$x_i' = \mathbf{R}x_i + T$$



# Parametric models: Use cases

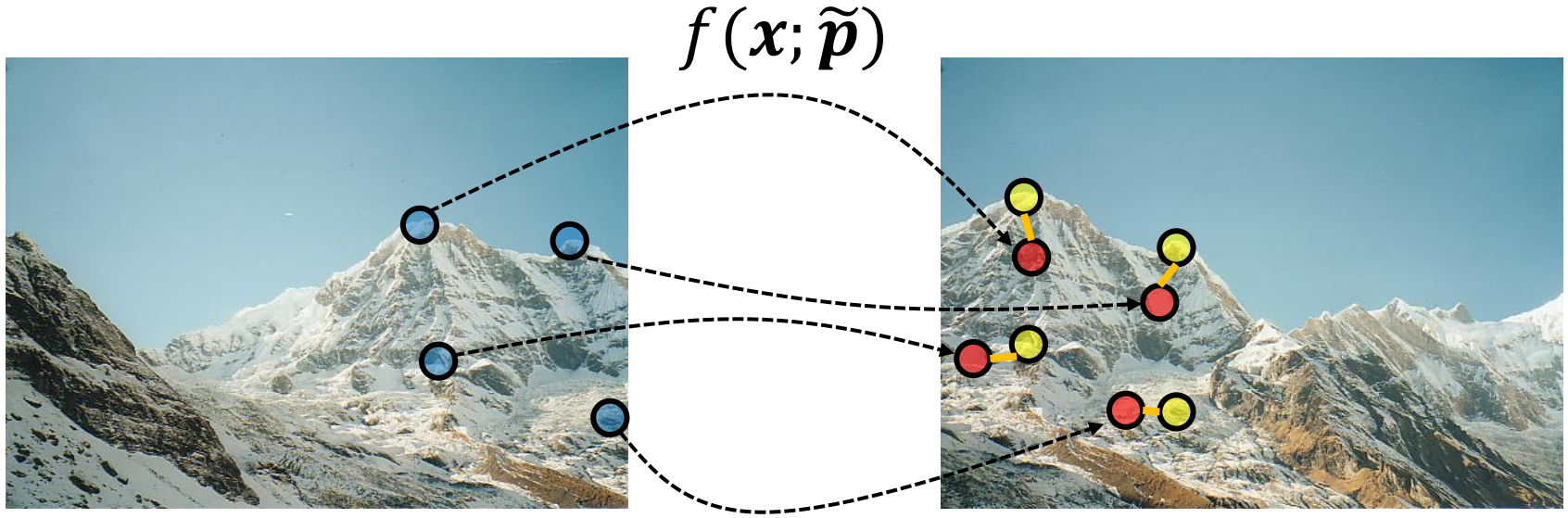
- Inverse problem: ``*Given a set of correspondences, what are the parameters of the transformation?*''




- Assuming the transformation can be well approximated by $f(\boldsymbol{x};\boldsymbol{p})$, what are the best parameter values for $\boldsymbol{p}$?

# Parametric models: Use cases

- Best parameter values: *those that minimize the projection error*



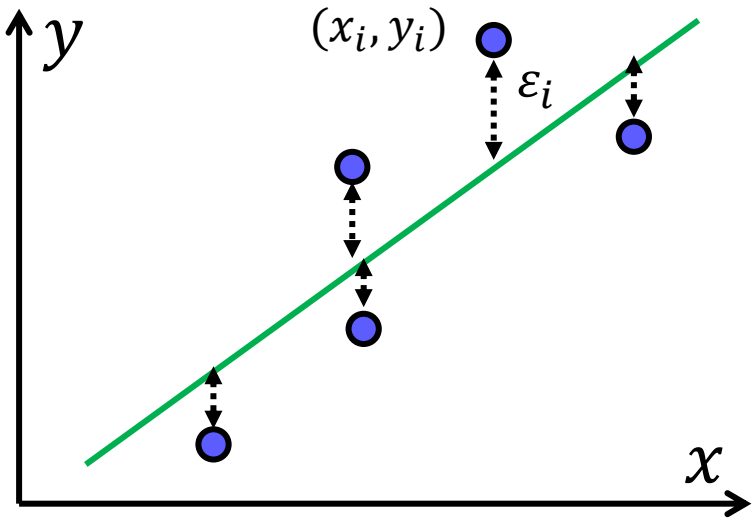

$$E(p) = \sum d_i$$

Stitched images:
Coordinates of all pixels in the left-hand image transformed by $f(x; p_{opt})$

# Least squares: Line fitting

## Problem formulation

- Data: $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- Line equation:
  $y = f(x; \boldsymbol{p}) = xp_1 + p_2$
- Parameters:
  $\boldsymbol{p} = [p_1, p_2]^T$
- Projection error at $i$-th correspondence:
  $\varepsilon_i = f(x_i; \boldsymbol{p}) - y_i$




- The cost function (goodness of fit): $E(\mathbf{p}) = \sum_{i=1}^{N} {\varepsilon_i}^2$

- Best parameters: $\boldsymbol{p}_{opt} = \arg\min_{\mathbf{p}} E(\mathbf{p})$

# A 1D minimization

How do you minimize a 1D continuous error function?
(with respect to one parameter)
(with respect to N parameters)

# Least squares: Line fitting

Strategy:

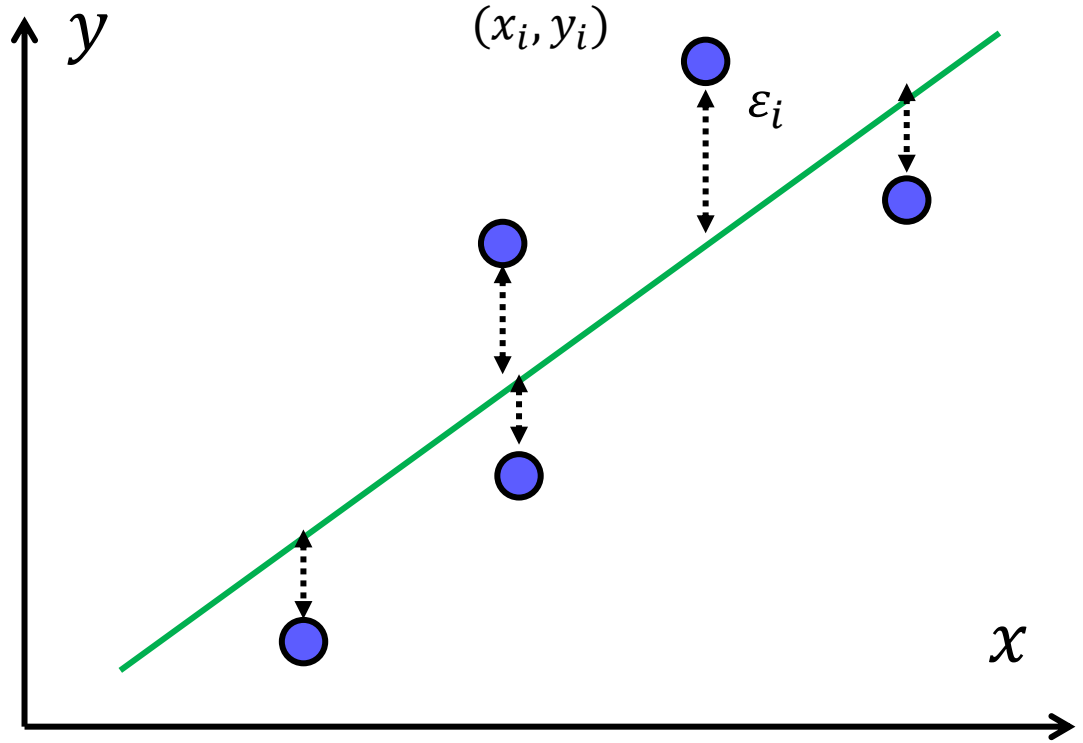
1. Rewrite the cost function $E(\boldsymbol{p})$ into a vector-matrix form
2. Take derivative w.r.t. $\boldsymbol{p}$, set to zero, solve for $\boldsymbol{p}$.

$$f(x;\boldsymbol{p}) = xp_1 + p_2$$

$$\boldsymbol{p} = [p_1, p_2]^T$$

$$\varepsilon_i = f(x_i;\boldsymbol{p}) - y_i$$

$$E(\mathbf{p}) = \sum_{i=1}^{N} {\varepsilon_i}^2$$



# A cookbook for normal equations:

1. Define the set of corresponding points
   $\{\boldsymbol{x}_i\}_{i=1:N}$ , $\{\boldsymbol{x}'_i\}_{i=1:N}$
2. Define the linear transformation
   $f(\boldsymbol{x};\boldsymbol{p}): \boldsymbol{x} \rightarrow \boldsymbol{x}'$
3. Define the per-point error and stack all errors into a single vector $\boldsymbol{\varepsilon}$:

$$E(\mathbf{p}) = \sum_{i=1}^{N} {\varepsilon_i}^2$$

$$\boldsymbol{\varepsilon}_i = f(\boldsymbol{x}_i;\boldsymbol{p}) - \boldsymbol{x}_i{}'$$

$$\varepsilon = \left[\varepsilon_1^T, \ldots, \varepsilon_i^T, \ldots, \varepsilon_N^T\right]^T$$

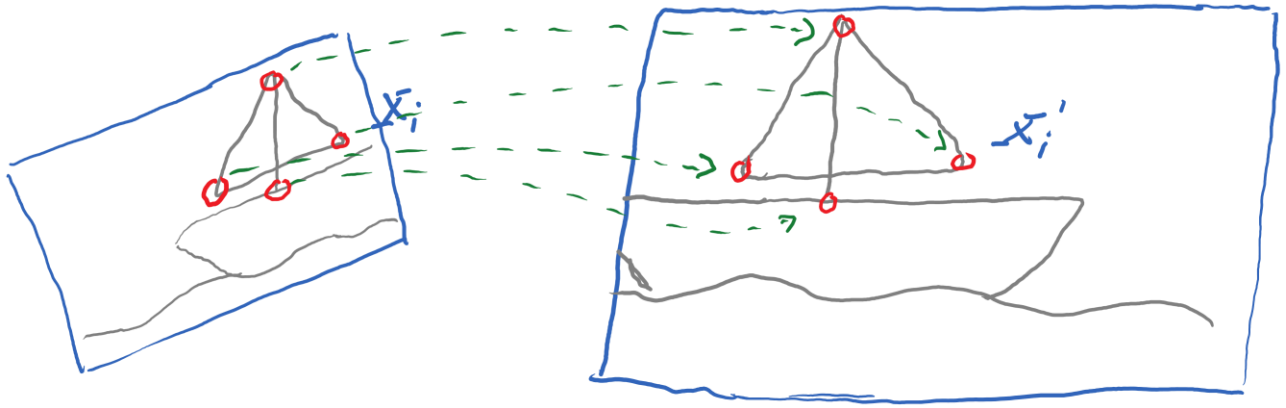
4. Rewrite the error into a form $\boldsymbol{\varepsilon} = \boldsymbol{A}\boldsymbol{p} - \boldsymbol{b}$
5. Solve by pseudoinverse: $\boldsymbol{p} = \boldsymbol{A}^{\dagger}\mathbf{b}$

Note: point errors $\boldsymbol{\varepsilon}_i$ are of same dimensionality as the points $\boldsymbol{x}'$.

Matlab: p = A \ b

# Least squares: A simple image alignment

- Task: Align two images based on correspondences




- Assume a similarity transform (scale, rotation, translation)

$$\boldsymbol{x}' = f(\boldsymbol{x}; \boldsymbol{p})$$

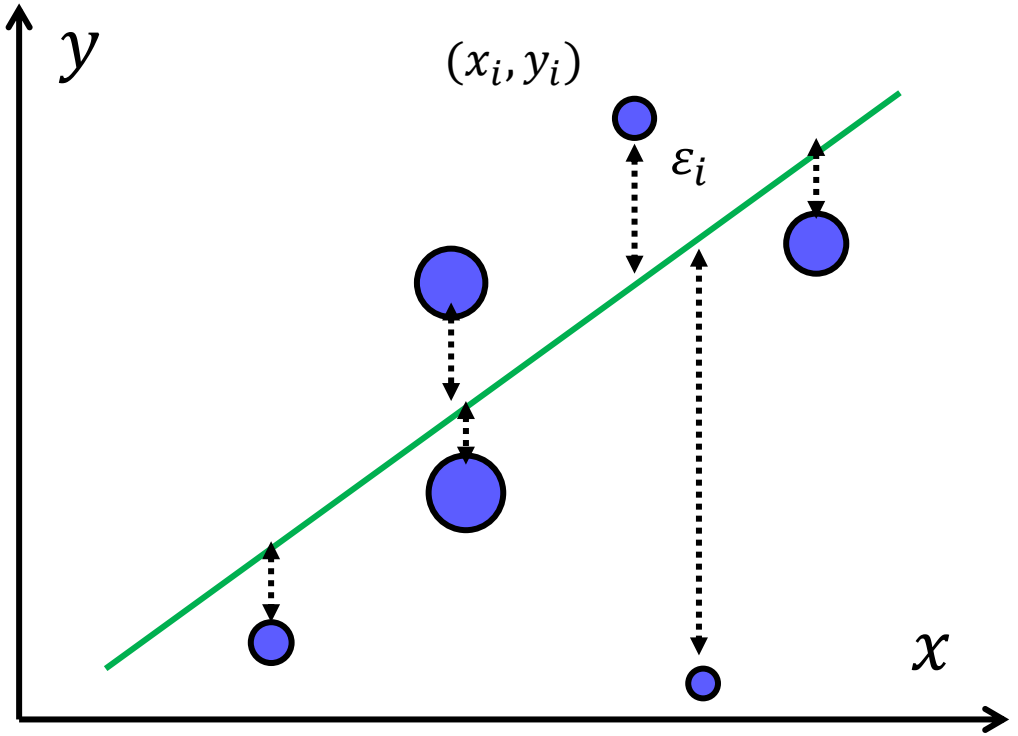
- The similarity transform is parameterized by (See Szeliski, Section 2.1.2):

$$x_i' = \begin{bmatrix} x_i' \\ y_i' \end{bmatrix} = \begin{bmatrix} p_1 x_i - p_2 y_i + p_3 \\ p_2 x_i + p_1 y_i + p_4 \end{bmatrix} \quad ; \; p = [p_1 \; p_2 \; p_3 \; p_4]^T$$

# Weighted least squares: Line fitting

## Problem formulation

- Data: $\{(x_1, y_1), \ldots, (x_N, y_N)\}$
- All points are *not* equally accurately measured!
- Weight at each point: $w_i$
- Projection error at $i$-th correspondence:
  $\varepsilon_i = f(x_i; \boldsymbol{p}) - y_i$
- A weighted cost: $E(\mathbf{p}) = \sum_{i=1}^{N} w_i {\varepsilon_i}^2$




- Best parameters: $\boldsymbol{p}_{opt} = \arg\min_{\mathbf{p}} E(\mathbf{p})$

# Weighted least squares: Line fitting

Strategy:

- Rewrite the cost function $E(\boldsymbol{p})$ into a vector-matrix form
- Take derivative w.r.t. $\boldsymbol{p}$, set to zero, solve for $\boldsymbol{p}$.

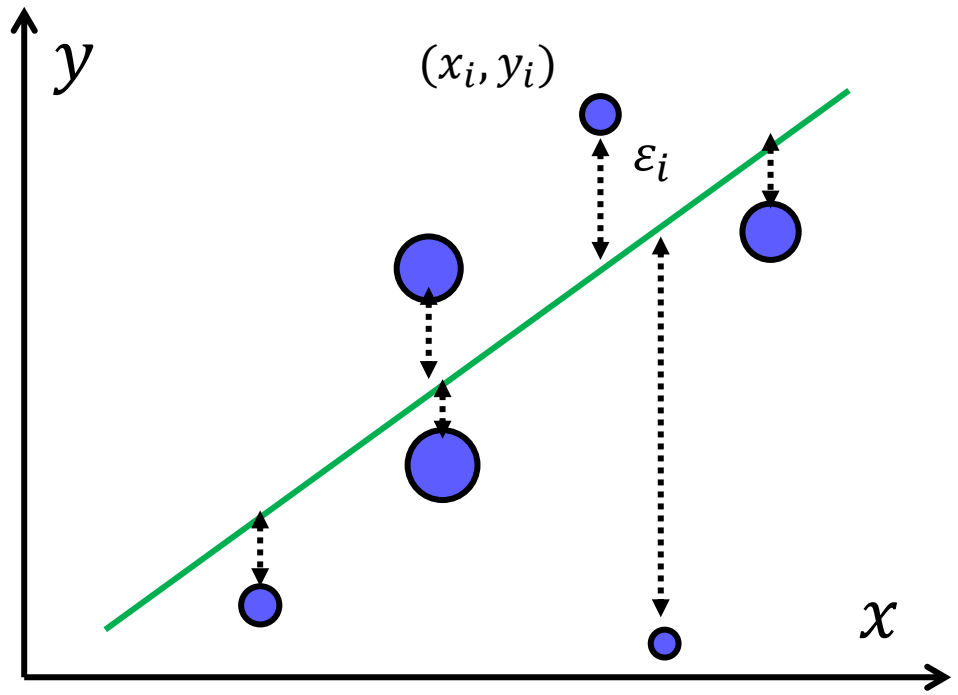
$$f(x;\boldsymbol{p}) = xp_1 + p_2$$

$$\boldsymbol{p} = [p_1, p_2]^T$$

$$\varepsilon_i = f(x_i;\boldsymbol{p}) - y_i$$

$$E(\mathbf{p}) = \sum_{i=1}^{N} w_i {\varepsilon_i}^2$$

$$\tilde{\mathbf{p}} = \arg\min_{\mathbf{p}} E(\mathbf{p})$$



# A cookbook for weighted least squares:

1. Define a weighted set of corresponding points
   $\{\boldsymbol{x}_i\}_{i=1:N}$ , $\{\boldsymbol{x}'_i\}_{i=1:N}$, $\{w_i\}_{i=1:N}$
2. Define the linear transformation
   $f(\boldsymbol{x};\boldsymbol{p}): \boldsymbol{x} \rightarrow \boldsymbol{x}'$
3. Rewrite the error into a form $\boldsymbol{\varepsilon} = \boldsymbol{Ap} - \boldsymbol{b}$
4. Create a weight matrix $\boldsymbol{W}$ as
   $W = diag([\boldsymbol{w}_1^T, \ldots, \boldsymbol{w}_N^T])$
   with $\boldsymbol{w}_i^T = w_i[1, .., 1]_{1\times d}$
5. Solve by : $\mathbf{p} = (\mathbf{A}^T\mathbf{W}\mathbf{A})^{-1}\mathbf{A}^T\mathbf{W}\mathbf{b}$

Note: $\boldsymbol{x}' \in \mathbb{R}^d, w \in \mathbb{R}^1$

Note: think about why are $\boldsymbol{w}_i^T$ vectors of same dimensionality as the points $\boldsymbol{x}'$.

To practice: solve the "sailboat" example

# NOTE

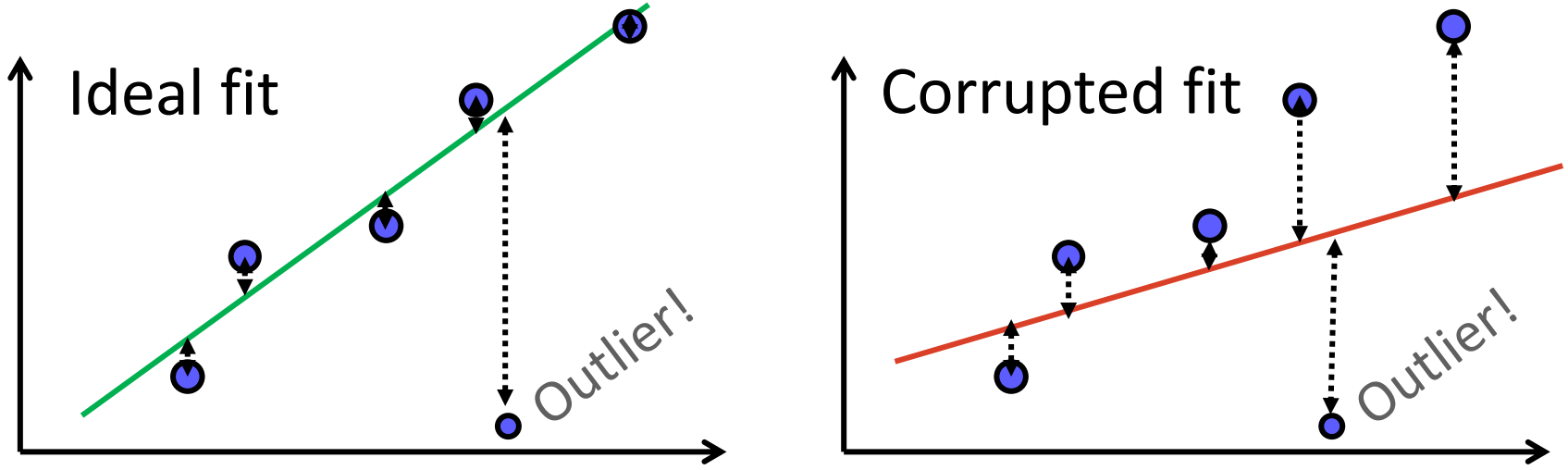
- Weighted least squares can be used for NONLINEAR/ROBUST least-squares problems as well!

- Robust least squares, for example can be implemented by iterative algorithm that applies a weighted least squares solver

- See the slides on e-classroom if you're interested

# Robust least squares

- Quadratic cost function behaves poorly with outliers:




- To see where the problem lies, we will have to rewrite our cost function into a general form.
- The cost can be generally written as: $E(\mathbf{p}) = \sum_{i=1}^{N} h(\varepsilon_i)$
- For ordinary least squares we had: $h(\boldsymbol{\varepsilon}_i) = ||\boldsymbol{\varepsilon}_i||^2$

Not covered in 2025 lectures
– Consider this as additional material

# Robust least squares

- For a cost function with robust error function $h(\varepsilon_i)$

  $E(\mathbf{p}) = \sum_{i=1}^{N} h(\varepsilon_i)$

- It is possible to find an equivalent weighted $L_2$ cost

  $E_W(\mathbf{p}) = \sum_{i=1}^{N} w(\varepsilon_i)||\varepsilon_i||^2$

  with $w = \frac{h'(\varepsilon)}{\varepsilon}$ and $h'(\varepsilon) = \frac{\partial h(\varepsilon)}{\partial \varepsilon}$ .

- Problems:
  1. Weights depend on the errors incurred by the optimal parameters of our model
  2. But the *parameters are unknown* and so are the weights.
- Solution: Can apply an iterative approach

  that will converge as long as $h\left(\sqrt{|\epsilon|}\right)$ is concave[1].

Not covered in 2025 lectures – Consider this as additional material

[1]Aftab, K. and Hartley, R., Convergence of Iteratively Re-weighted Least Squares to Robust M-estimators, WACV 2015

R. Hartley, Robust Optimization Techniques in Computer Vision, Session 3,ECCV2014 tutorials

# Iterative reweighted least squares

1. Set all the weights to $w_i^{t-1} = 1$.
2. Solve for $\boldsymbol{p}^t$ by the weighted least squares problem.
3. Using the estimated parameters $\boldsymbol{p}^t$ re-calculate per-point projection errors $\boldsymbol{\varepsilon}_i^t$.
4. Using the projection errors re-calculate new weights $w_i^t$ from:

$$w = \frac{h'(\varepsilon)}{\varepsilon} \qquad h'(\varepsilon) = \frac{\partial h(\varepsilon)}{\partial \varepsilon}$$

5. Go back to step 2 and continue until the change in parameters is negligibly small (convergence).

Note: $(\cdot)^t$ indicates a step of iteration in the iterative reweighted least squares.

For an instructive discussion on parameters of the Huber cost function from data, please see:
J. Fox, Robust Regression--Appendix to An R and S-PLUS Companion to Applied Regression, 2002, ”1.1 Objective Functions”.

Not covered in 2025 lectures
– Consider this as additional material

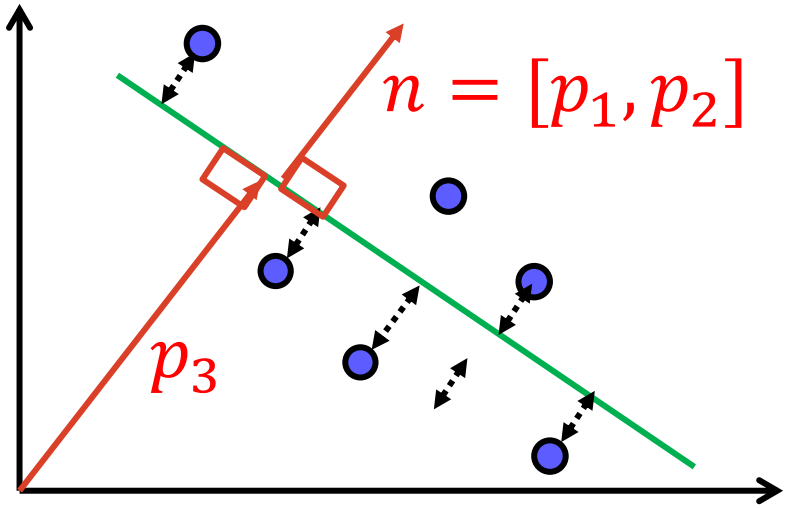
# Constrained least squares

- Often we will seek parameters $\boldsymbol{p}$ that satisfy constraints.
- Reconsider line-fitting example, but this time we'll minimize *perpendicular* distances!

$$E(\mathbf{p}) = \sum_{i=1}^{N} ||\varepsilon_i||^2$$

- Re-parameterize:
  $\boldsymbol{p} = [p_1, p_2, p_3]^T$
- Distance of a point to line:
  $||\boldsymbol{\varepsilon_i}||^2 = (x_i p_1 + y_i p_2 - p_3)^2$
- Let's minimize:

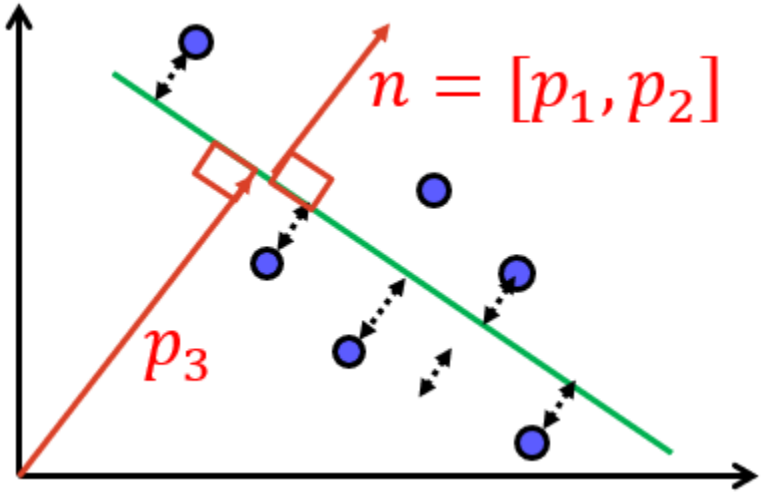
$$E(\mathbf{p}) = \sum_{i=1}^{N} ||\varepsilon_i||^2$$



# Back to line fitting example…

- Distance of a point to line:
  $||\boldsymbol{\varepsilon_i}||^2 = (x_i p_1 + y_i p_2 - p_3)^2$
- Let's minimize:
  $E(\mathbf{p}) = \sum_{i=1}^{N} ||\varepsilon_i||^2$

  $\mathbf{A}^T\mathbf{A}\mathbf{p} = \lambda\mathbf{p}$



# Constrained least squares

- The solution: $\frac{dE(\mathbf{p})}{d\mathbf{p}} = 2\mathbf{A}^T\mathbf{A}\mathbf{p} \equiv \mathbf{0}$
- Trivial solution: $\boldsymbol{p} = \mathbf{0}$

In case you are not confident with Lagrange multipliers, see this excellent tutorial!

- A nontrivial solution is obtained by constraint $\left|\left|\boldsymbol{p}\right|\right|^2 = 1$
- Taking the derivative of a Langrangian and setting to 0:

$$\mathbf{A}^T\mathbf{A}\mathbf{p} = \lambda\mathbf{p} \quad \longleftarrow \text{Homogenous equation!}$$

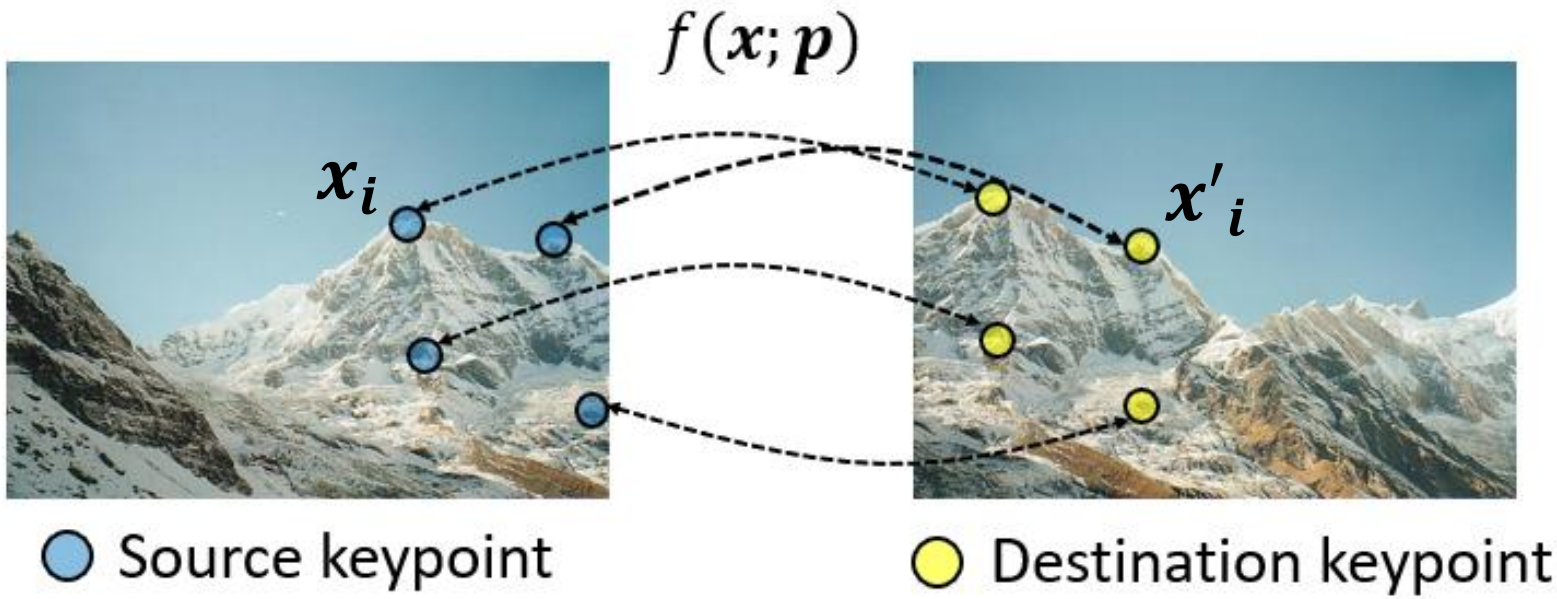
- The solution is the eigenvector of $(\boldsymbol{A}^T\boldsymbol{A})$ corresponding to the smallest eigenvalue.
- Actually, it can be shown that this is also the eigenvector corresponding to the smallest eigenvalue of $\boldsymbol{A}$. (see notes on "*Avoid computing $A^TA$*")

# Previously at MP…

- Least squares fitting

$$\boldsymbol{x'} = \boldsymbol{D}\boldsymbol{x} + \boldsymbol{c}$$
$$\boldsymbol{p} = \{\boldsymbol{D}, \boldsymbol{c}\} = ?$$

$$E(\mathbf{p}) = \sum_{i=1}^{N} w_i {\varepsilon_i}^2$$




Unconstrained least squares:
- Ordinary least squares
- Weighed least squares

Constrained least squares:
- Requires a constraint on the admissible solution

# Recognizing the hammer for your nail!

- Problems that can be written as systems of equations (*normal equations*):

  $\boldsymbol{Ap = b}$

  (if you have weights on equations, then $\boldsymbol{WAp = Wb}$)

  can be solved by ordinary LS or IRWLS

  Matlab: p = A \ b ;

- Problems that result in a homogenous system:

  $\boldsymbol{Ap = 0}$

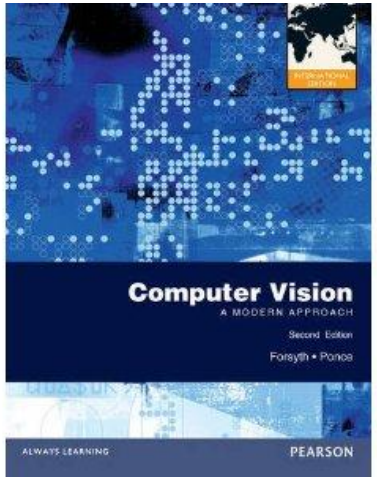
  can be solved by putting the constraint $\left|\left|\boldsymbol{p}\right|\right|^2 = 1$, the solution is the eigenvector corresponding to the smallest eigenvalue.

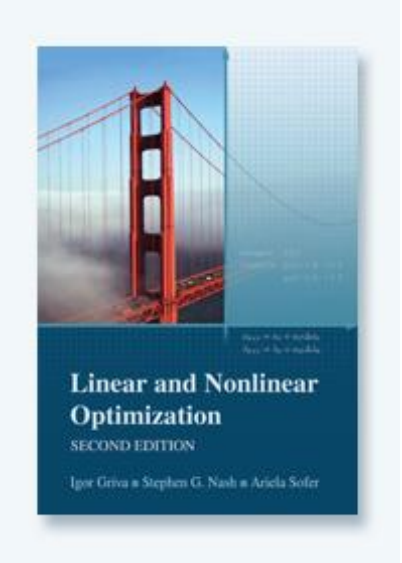
  (If required, rescale the solution for $\boldsymbol{p}$)

  Matlab: [U,S,V] =svd(A) ; p = V(:,end) ;

# For nonlinear cost functions

- Often nonlinear & nonquadratic error functions are used, which cannot be minimized analytically in a closed form.




- Popular approaches:
  - Gradient descend
  - Newton's method
  - Gauss-Newton method
  - Levenberg-Marquardt (very popular in, e.g., camera geometry in CV)
  - Alternate direction method of multipliers (ADMM) [!very powerful & simple]
- More about these:

- Fua and Lepetit: Computer Vision Fundamentals: Robust Non-Linear Least-Squares and their Applications
- Griva et al., Linear and Nonlinear Optimization (See appendix on Matrix Algebra)
- The Matrix Coockbook (List of common vector/matrix solutions)
- Forsyth, Ponce, „Computer Vision – A modern approach", (Appendix in *2nd ed.*)

# Need to deal even better with outliers

- Large disagreements in only a few points (outliers) cause failure of the least-squares-based methods.
- The detection, localization and recognition in CV must work in significantly noisy data.
- In some cases >½ data is expected to be outliers.
- Standard methods for robust estimation can rarely deal with such a large proportion of outliers.

# RANSAC

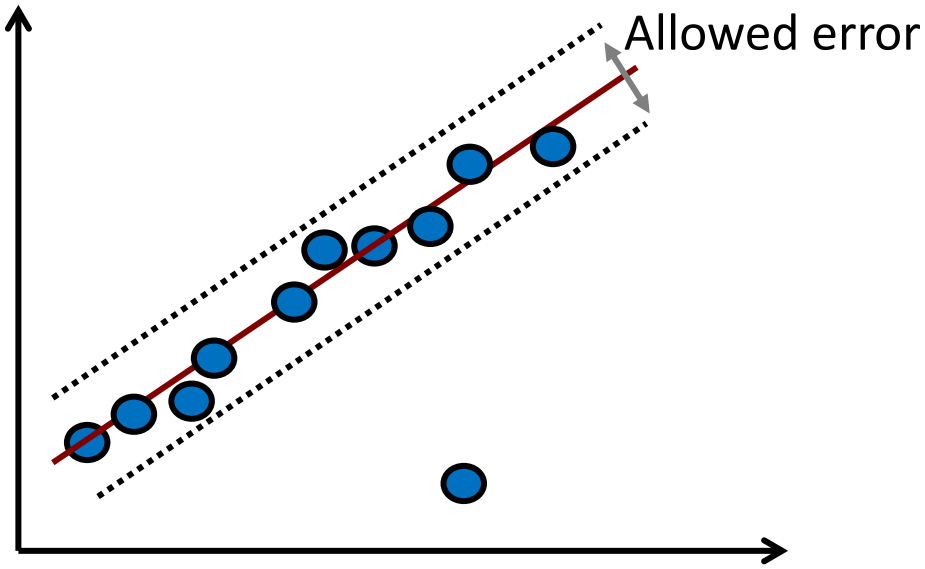
- The RANSAC [1] algorithm (random sample consensus).

- Very popular due to its generality and simplicity.
- Can deal with large portions of outliers.
- Published in 1981 (Fischler in Bolles)
- One of the most cited papers in Computer Vision
- Many improvements proposed since!

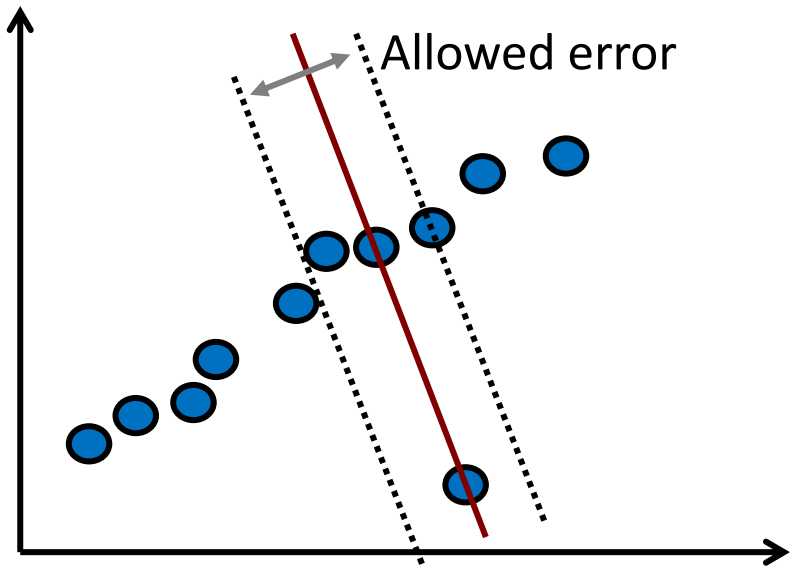
[1] M. A. Fischler, R. C. Bolles. Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. Comm. of the ACM, Vol 24, pp. 381-395, 1981.

# RANSAC: Intuition by line fitting

- A good estimate of our model should have a strong support in data:
  *"recognize a good model when you see it"*




10 point support this line!

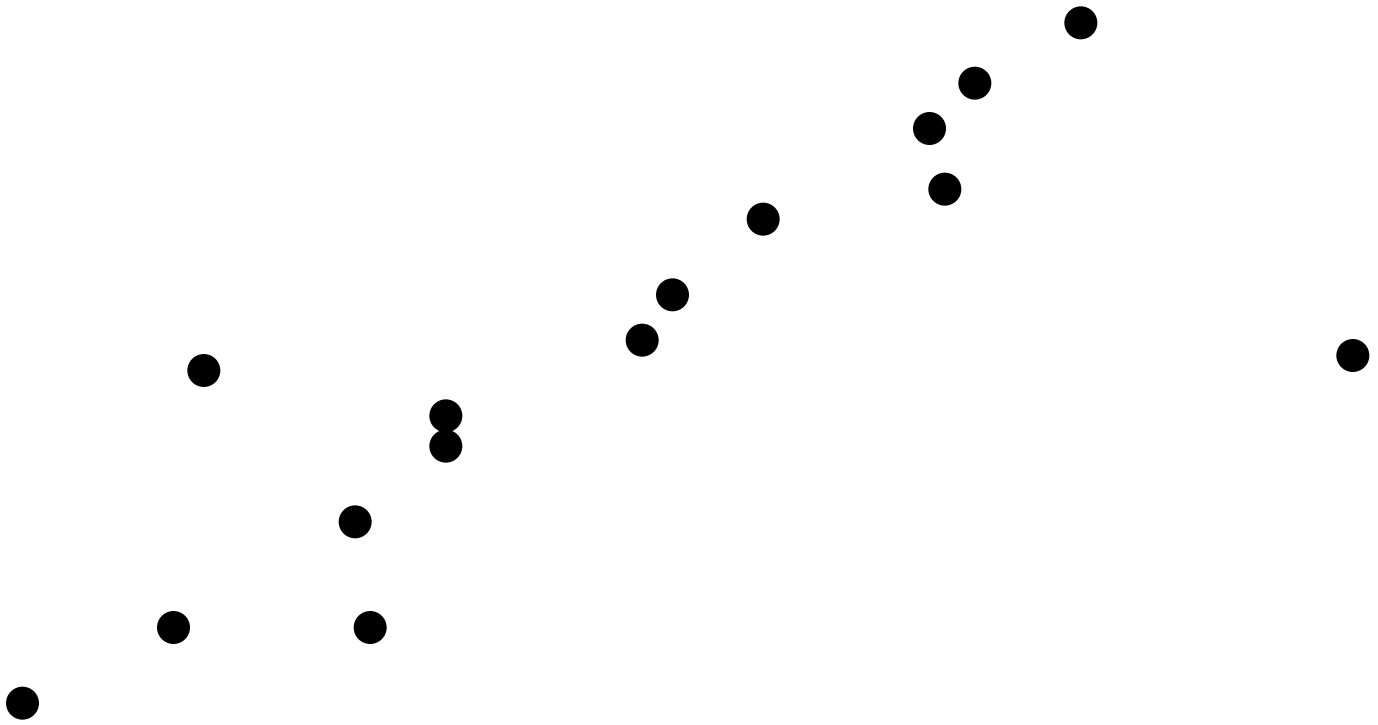



4 point support this line!

- How to find a model with a strong support?
- By randomly sampling potential models.

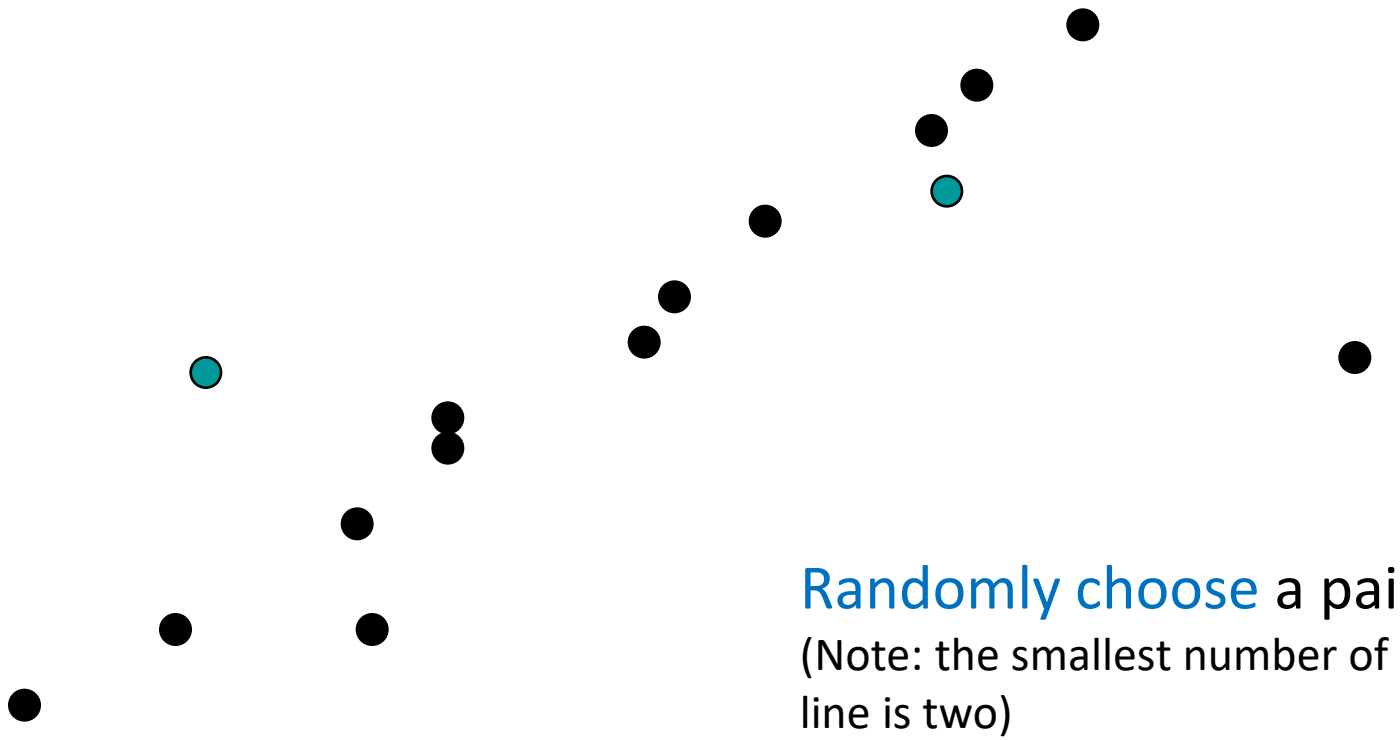
# RANSAC: Intuition by line fitting

- Task: Robustly estimate the most likely line



# RANSAC: Intuition by line fitting

- Task: Robustly estimate the most likely line



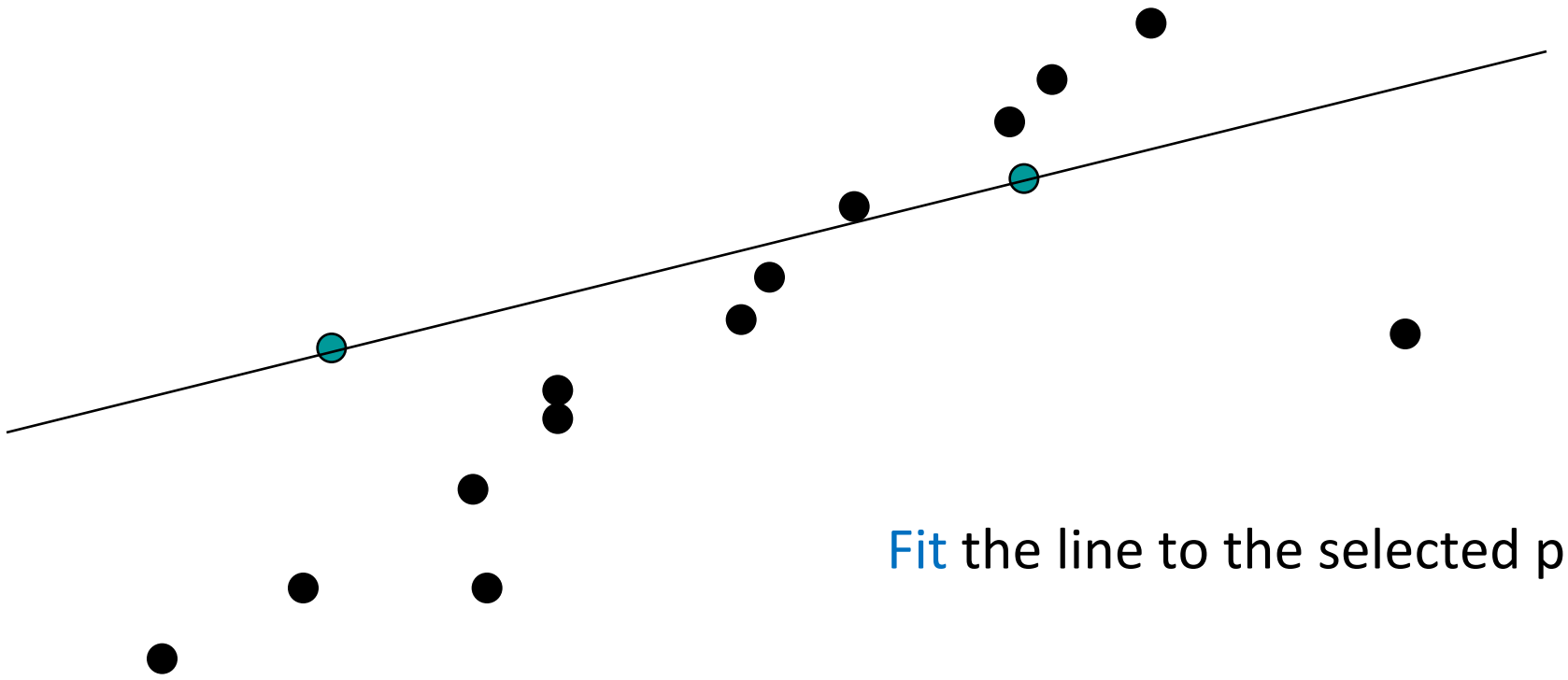
Randomly choose a pair of points

(Note: the smallest number of points to fit a line is two)

Slide credit: Jinxiang Chai

# RANSAC: Intuition by line fitting

- Task: Robustly estimate the most likely line

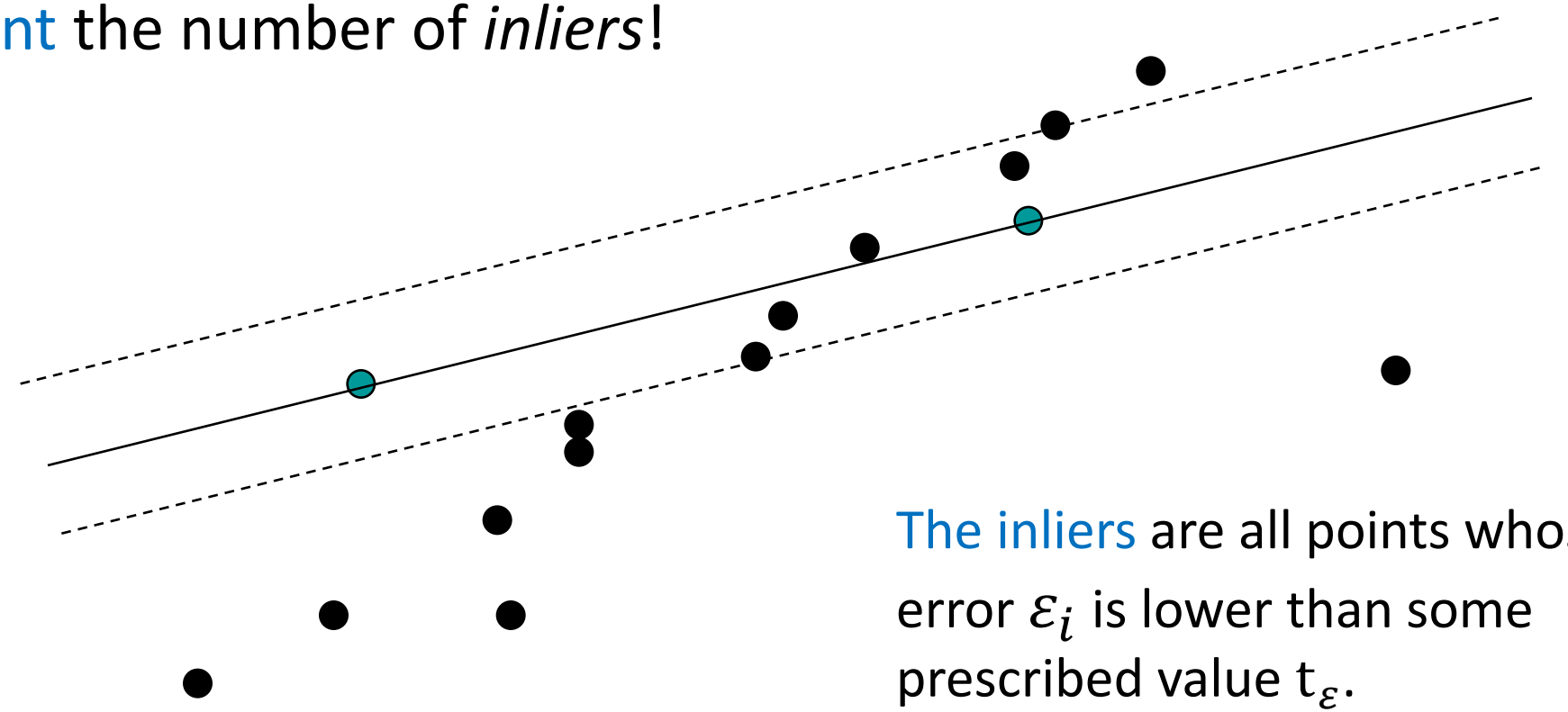


Fit the line to the selected points.

# RANSAC: Intuition by line fitting

- Task: Robustly estimate the most likely line

Count the number of *inliers*!

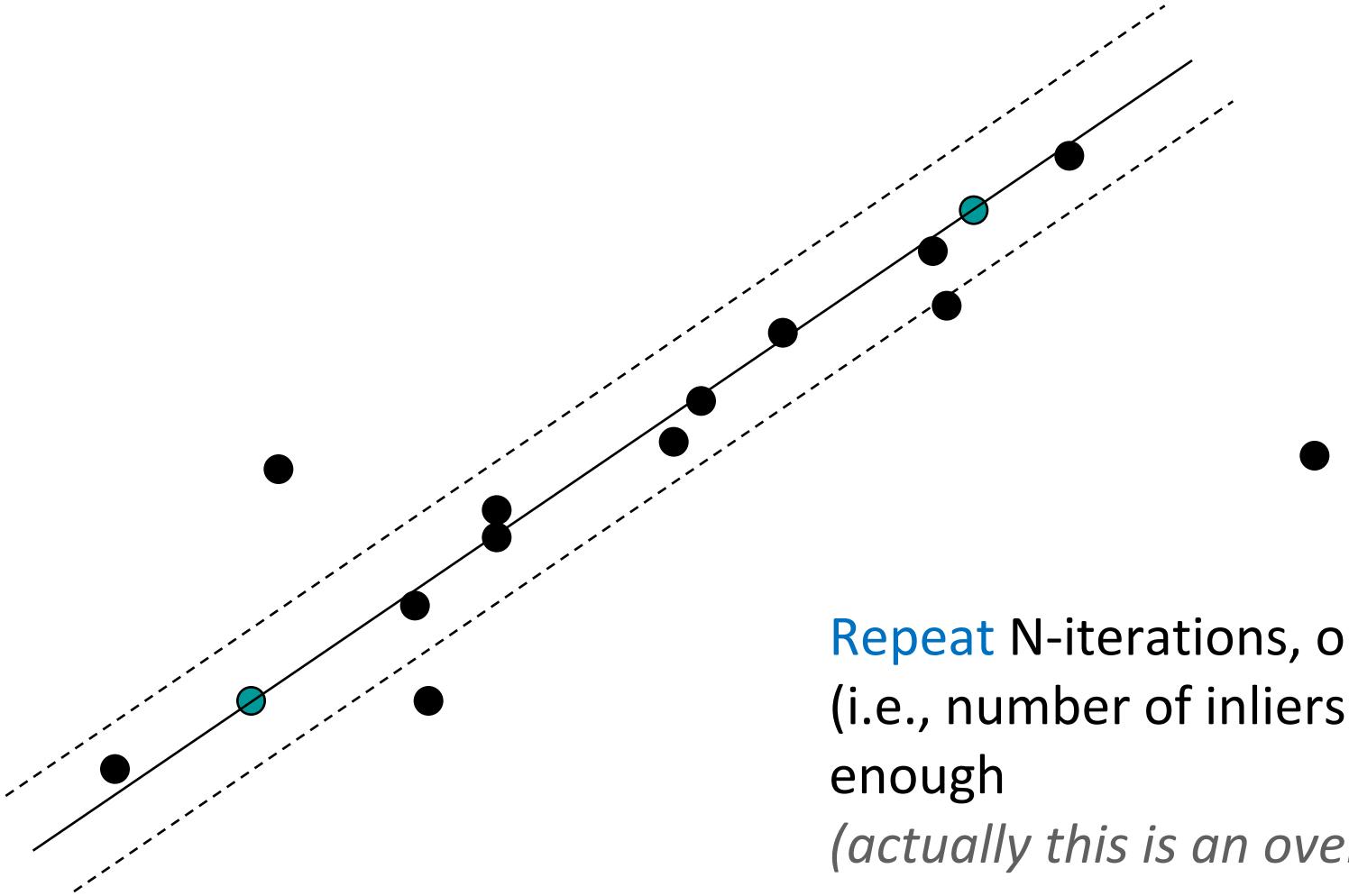


The inliers are all points whose error $\varepsilon_i$ is lower than some prescribed value $t_\varepsilon$.

$$\varepsilon_i = |f(x_i; \boldsymbol{p}) - y_i|$$

# RANSAC: Intuition by line fitting

- Task: Robustly estimate the most likely line

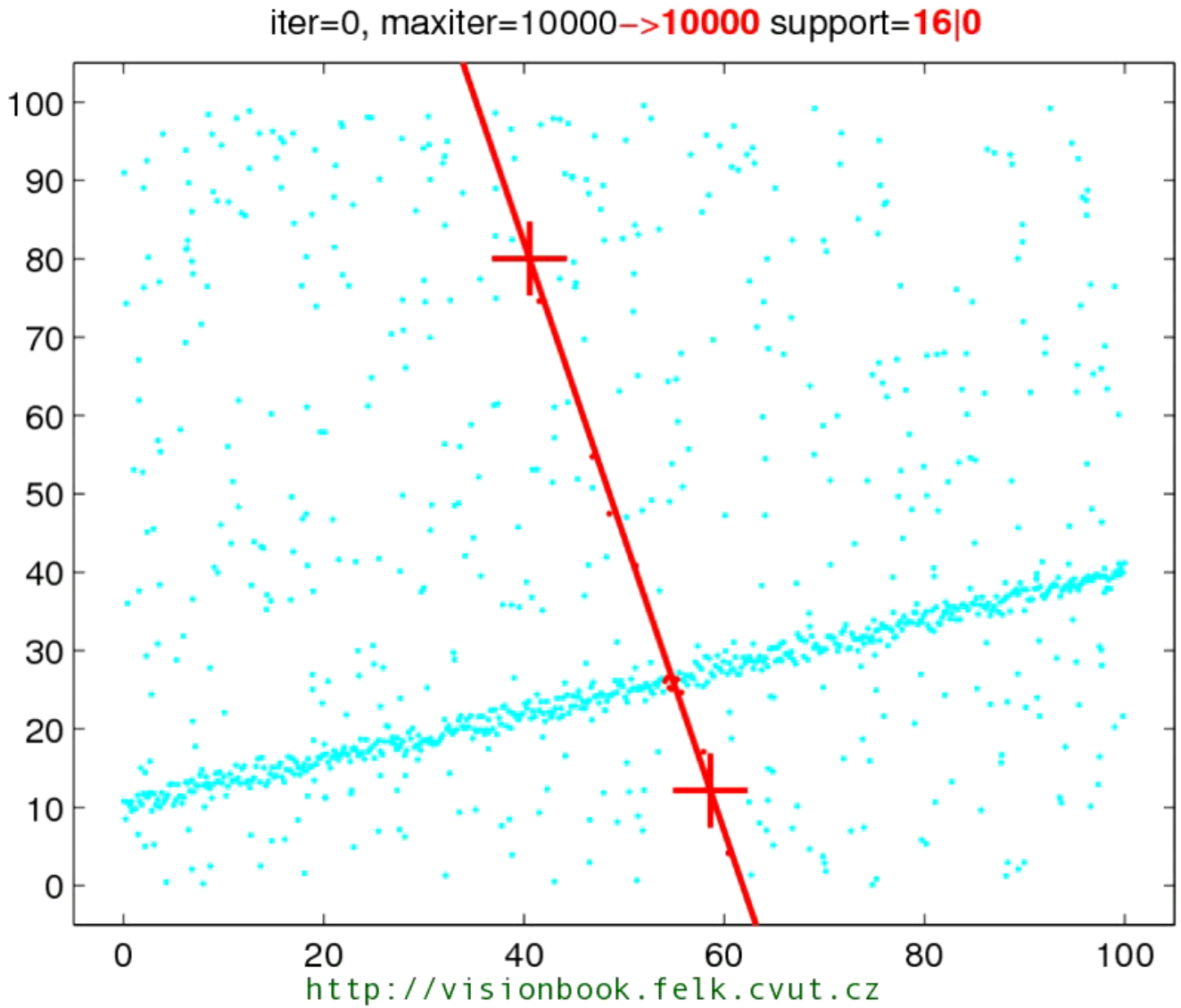


Repeat N-iterations, or, until the support (i.e., number of inliers) becomes strong enough

*(actually this is an oversimplification)*.

# RANSAC: line fitting

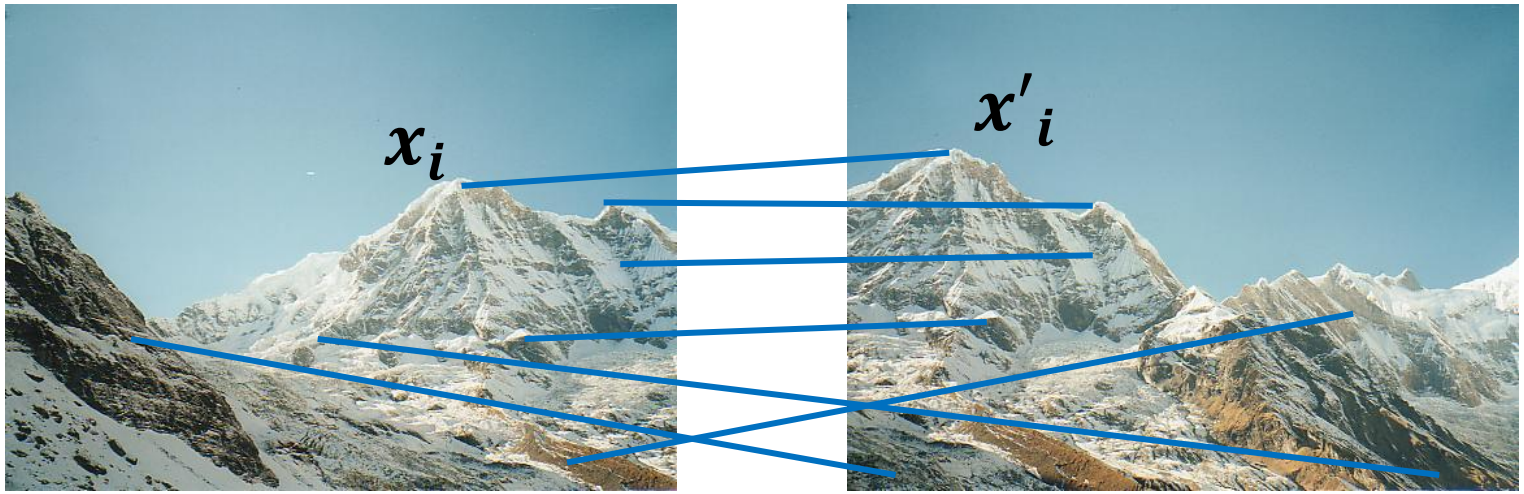
- Another example



# A general setting

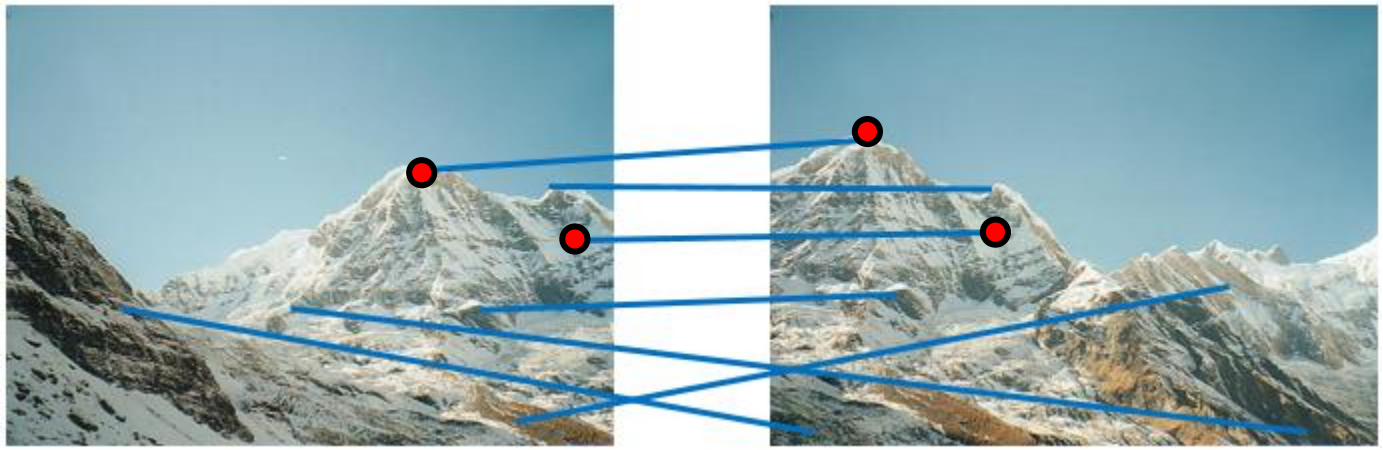
1. Define the set of "potentially" corresponding points:
   $\{\boldsymbol{x}_i\}_{i=1:N}$ , $\{\boldsymbol{x}'_i\}_{i=1:N}$
2. Define the transformation model: $f(\boldsymbol{x};\boldsymbol{p}): \boldsymbol{x} \rightarrow \boldsymbol{x}'$

   In this example, let $f(\boldsymbol{x};\boldsymbol{p})$ be a simple translation + scaling.




Important: Some correspondences are correct and some are NOT!

# A simple RANSAC loop



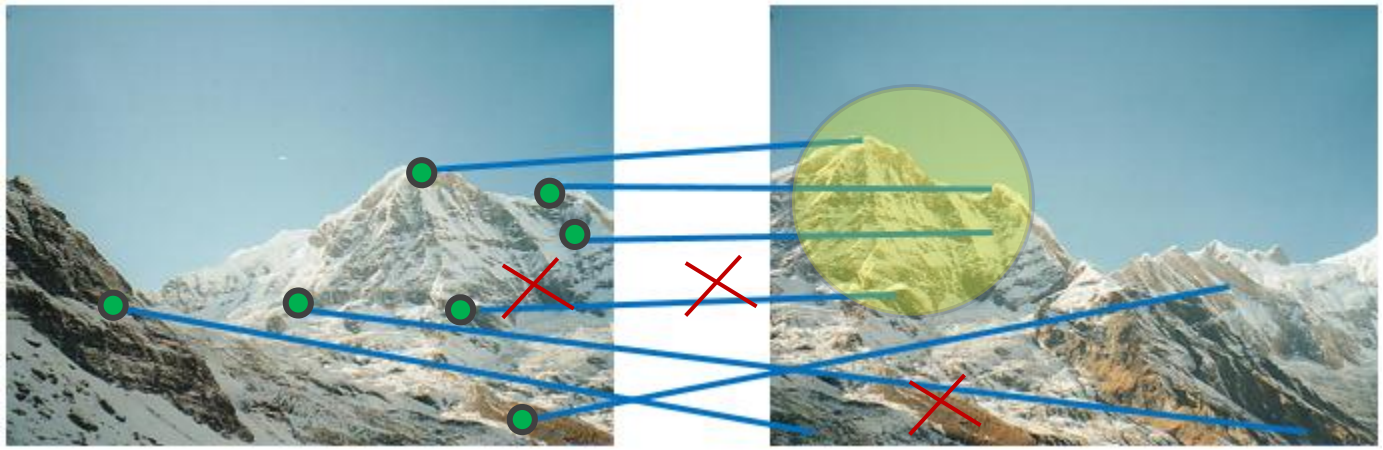
$\{\boldsymbol{x}_i\}_{i=1:N}$ , $\{\boldsymbol{x}'_i\}_{i=1:N}$

$f(\boldsymbol{x};\boldsymbol{p}): \boldsymbol{x} \rightarrow \boldsymbol{x}'$

In this example, let $f(\boldsymbol{x};\boldsymbol{p})$ be a simple translation + scaling.

1. Randomly select the smallest group of correspondences, from which we can estimate the parameters of our model.
2. Fit the parametric model $\tilde{p}$ to the selected correspondences (e.g., by LS).

# A simple RANSAC loop



$\{\boldsymbol{x}_i\}_{i=1:N}$ , $\{\boldsymbol{x}'_i\}_{i=1:N}$

$f(\boldsymbol{x};\boldsymbol{p}): \boldsymbol{x} \rightarrow \boldsymbol{x}'$

In this example, let $f(\boldsymbol{x};\boldsymbol{p})$ be a simple translation + scaling.

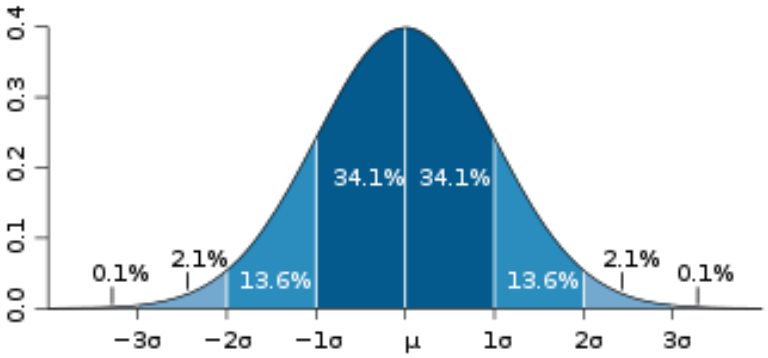
1. Randomly select the smallest group of correspondences, from which we can estimate the parameters of our model.
2. Fit the parametric model $\tilde{p}$ to the selected correspondences (e.g., by LS).
3. Project all other points and count how many of all correspondences are in agreement with the fitted model – number of inliers.

- Remember the model parameters $\tilde{p}_{opt}$ that maximize the number of inliers.

# The choice of parameters

- How many correspondences *''s''* are required*?*
  - Typically the smallest number that allows estimating the model parameters, i.e., as many as the model parameters.
- Threshold distance *t* for identifying the inliers
  - Choose *t*, such, that the probability that an inlier falls below the threshold is equal to $p_w$. For example ($p_w$=0.95)
  - Assuming a Gaussian noise on the measurements.

    The noise standard dev. σ: t=2σ
- Number of sampling iterations *N*
  - Chose *N* such, that the probability *p* of drawing a sample with all inliers at least once is high enough.



# The choice of parameters: N

- Setting the number of sampling iterations *N:*
  - Assume we know the proportion *e* of outliers (probability of selecting an outlier at random).
  - Choose *N* such, that the probability of drawing a sample set with all inliers at least once in N draws is *p*,(e.g., *p*=0.99).
  - Derive the probability of drawing a bad sample in N trials, $1 - \mathrm{p} = {p_{bad}}^{N}$, and expose N
    - Probability of choosing a single inlier: 1 - *e*
    - Probability of an all-inlier sample:
      → *s-times sample an inlier*: $(1 - \mathrm{e})^{s}$
    - Probability, of a bad sample:
      → at least one of *s not an inlier*: $[1- (1 - \mathrm{e})^{s}]$
    - Probability of always drawing a bad sample in N trials: $(1- (1 - \mathrm{e})^{s})^{N}$

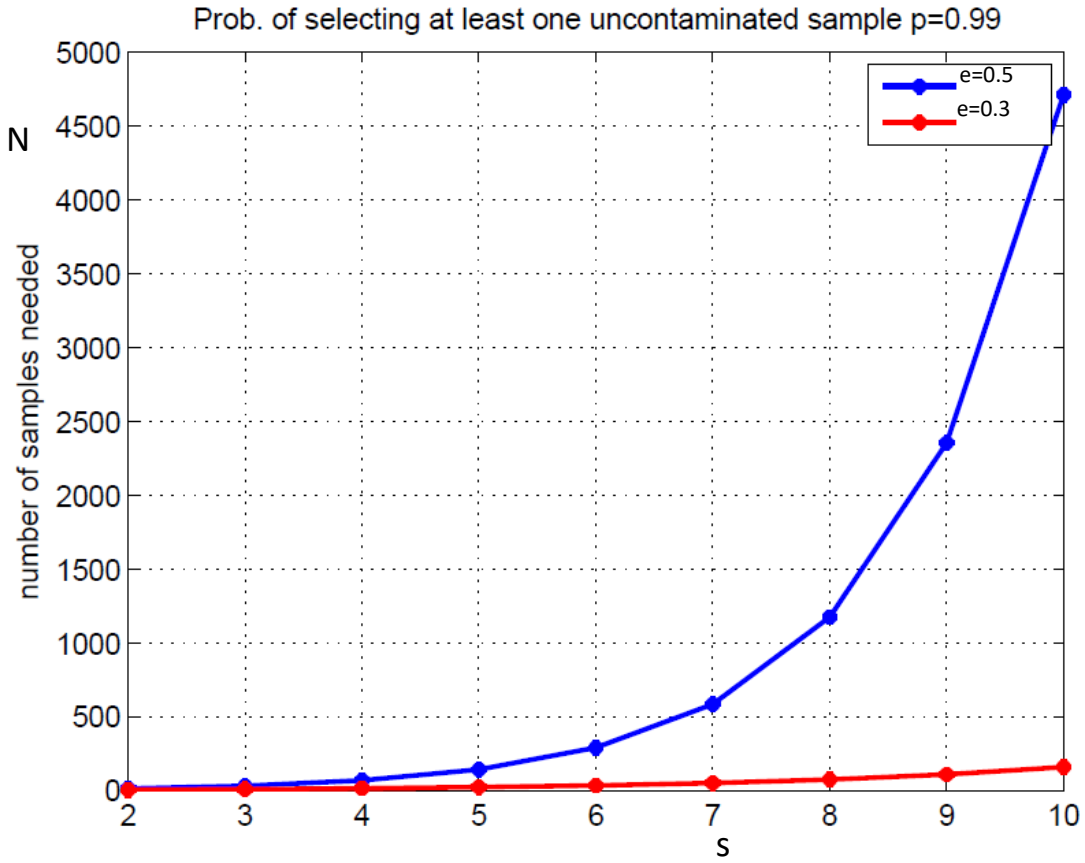
$$1 - p = (1 - (1 - e)^{s})^{N} \quad \Rightarrow \quad N = \frac{\log(1-p)}{\log(1-(1-e)^{s})}$$

# The choice of parameters: N

Number of iterations N required to sample an inlying model with $s$ parameters at least once with probability $p$ if the proportion of outliers is $e$:

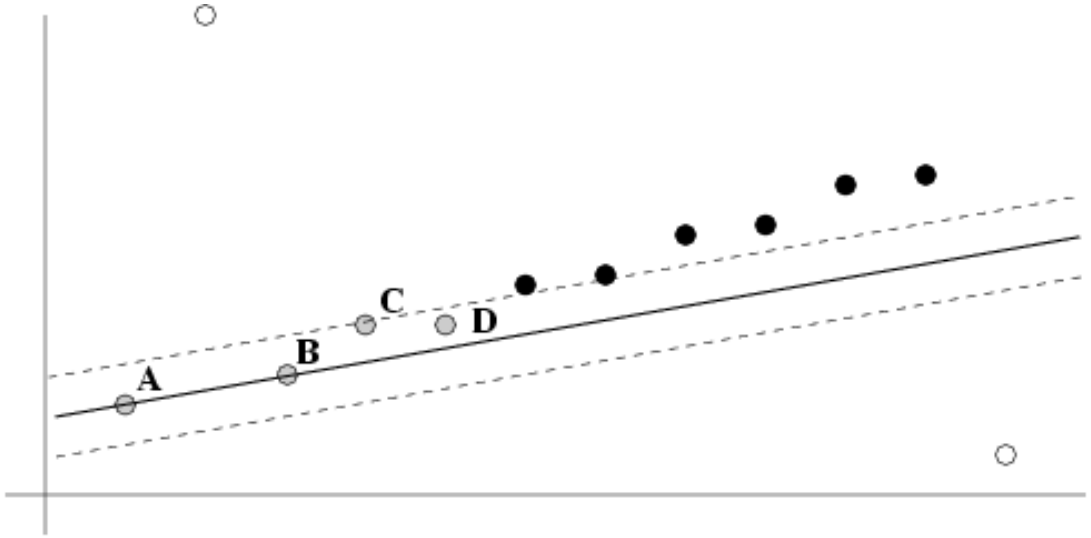
| | portion of outliers: e | | | | | | |
|---|---|---|---|---|---|---|---|
| s | 5% | 10% | 20% | 25% | 30% | 40% | 50% |
| 2 | 2 | 3 | 5 | 6 | 7 | 11 | 17 |
| 3 | 3 | 4 | 7 | 9 | 11 | 19 | 35 |
| 4 | 3 | 5 | 9 | 13 | 17 | 34 | 72 |
| 5 | 4 | 6 | 12 | 17 | 26 | 57 | 146 |
| 6 | 4 | 7 | 16 | 24 | 37 | 97 | 293 |
| 7 | 4 | 8 | 20 | 33 | 54 | 163 | 588 |
| 8 | 5 | 9 | 26 | 44 | 78 | 272 | 1177 |

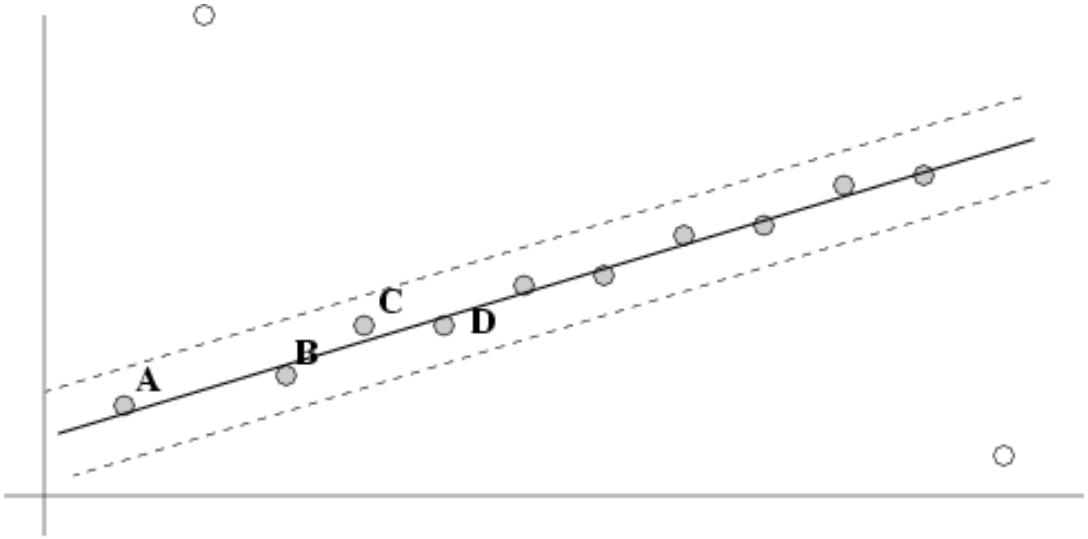
Tabulated values of $N$ for $p = 0.99$



# After RANSAC: Refit by LS

- RANSAC splits the data into inliers and outliers, and calculates the model parameters using a minimal number of correspondences.
- Improve the model parameters by applying least squares to the inliers.



# Beyond the simple RANSAC

- A great deal of research was invested by many researchers into improving RANSAC
  - Finding the right solution faster & with better resiliency to outliers
  - See an excellent tutorial: https://danini.github.io/ransac-2025-tutorial/
- Further reading:
  - PROSAC (state of the art, better chooses the order of samples)
  - MAGSAC++ (best among hand-crafted methods)
  - Excellent tutorial in recent RANSAC developments and toolboxes: RANSAC in 2020: A CVPR Tutorial, CVPR 2020 (Video presentations available!)
- Generalized differentiable RANSAC (ICCV2023): paper, code
  - ∇-RANSAC (learns the entire estimation pipeline)
  - Currently the top performer among all variants

# RANSAC: Summary

- Pros
  - Very simple and general
  - Applicable to many real-life problems
  - Often used in practice

- Cons
  - Requires setting some parameters (modern methods make it simpler)
  - Potentially many iterations required to find the optimum.
  - Fails at very small number of inliers.
  - In some cases more accurate procedures, that do not require brute-force sampling, can be found.

# Fitting: Challenges

- If we know the inliers how to estimate the parameters?
  - Least squares
- What if our data includes outliers?
  - Robust least squares, RANSAC
- What if we have multiple instances of our model (e.g., multiple lines)?
  - Apply voting: sequential RANSAC, Hough transform
- What if we have multiple models (e.g., unknown degree of a polynomial)?
  - Apply model selection (e.g., MDL, BIC, AIC)
- Complicated nonparametric models
  - Generalized Hough (GHT)
  - Iterative Closest Point, (ICP) == iterative local least squares

# Further reading

- A simple and interesting way to iteratively fit a complicated model to data:
  Iterative Closest Point method
  Matlab implementation: ICP

- A very nice and accessible tutorial on nonlinear optimization in computer vision: http://cvlabwww.epfl.ch/~fua/courses/lsq/Intro.htm

- Excellent tutorial in recent RANSAC developments and toolboxes:
  RANSAC in 2020: A CVPR Tutorial
  RANSAC in 2025: An ICCV Tutorial

# References


- R. Szeliski,Computer Vision: Algorithms and Applications, 2010

- David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition), (*second edition!*)
  - See appendix on Normal equations  and Homogeneous systems

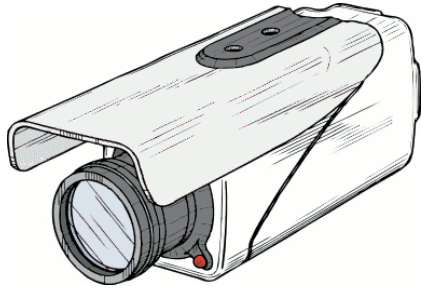
- Igor Griva, Stephen G. Nash, Ariela Sofer ,Linear and Nonlinear Optimization
  - See appendix on Matrix Algebra

- The Matrix Cookbook
  - List of common vector/matrix solutions

Univerza v Ljubljani

ViCOS visual cognitive systems lab

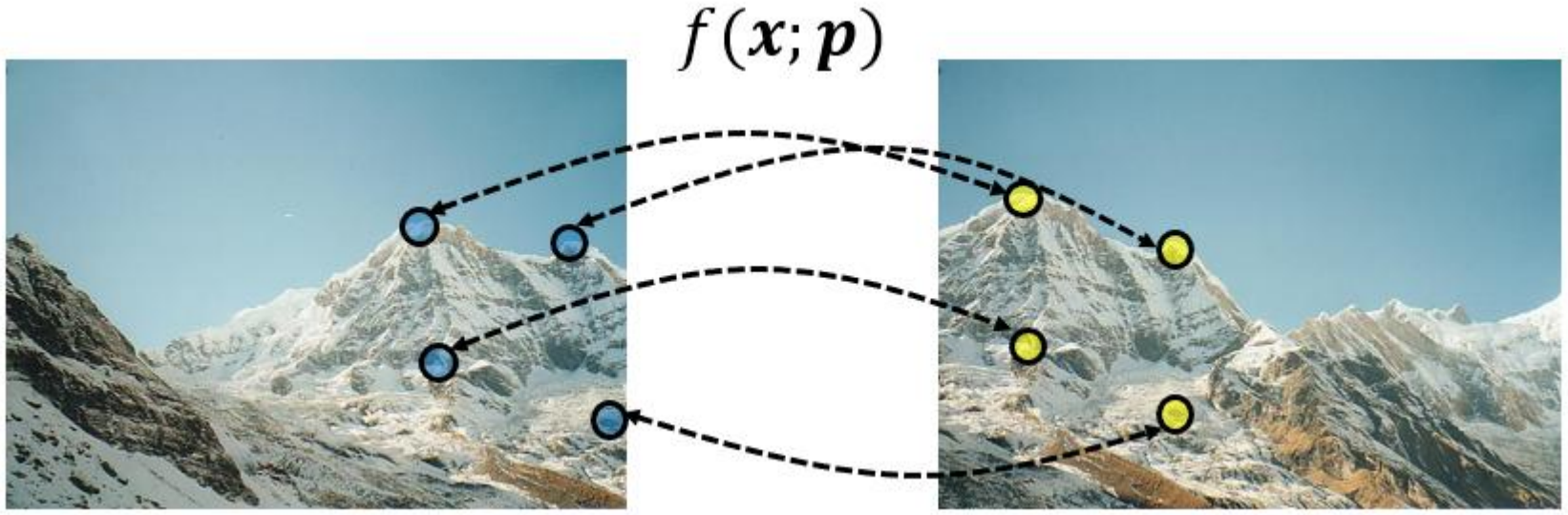
# Machine Perception
# Key-points and matching



Matej Kristan


Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

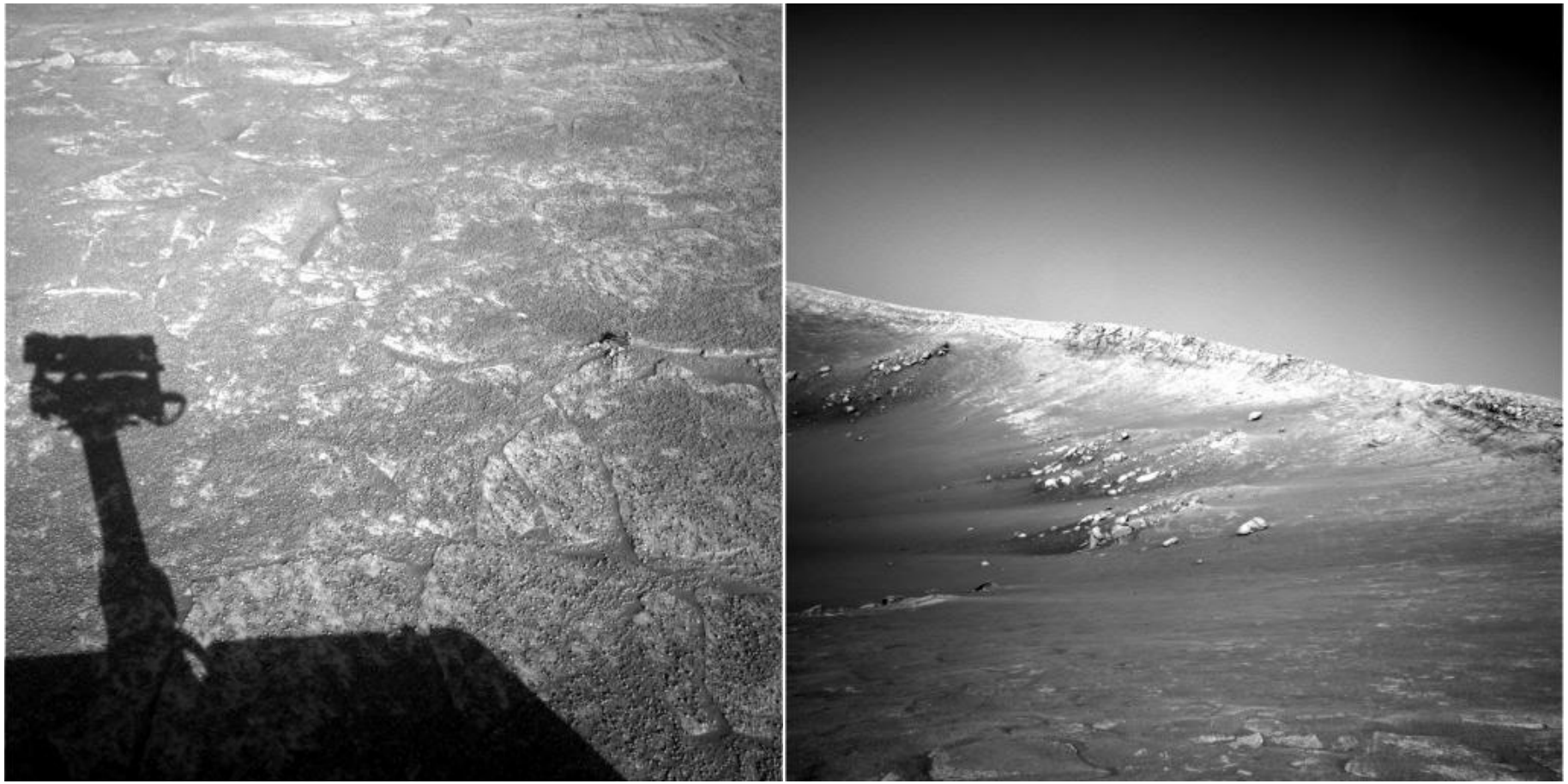
# Recall the panorama creation process




Identification of the corresponding "key points" required!

# Corresponding key points selection

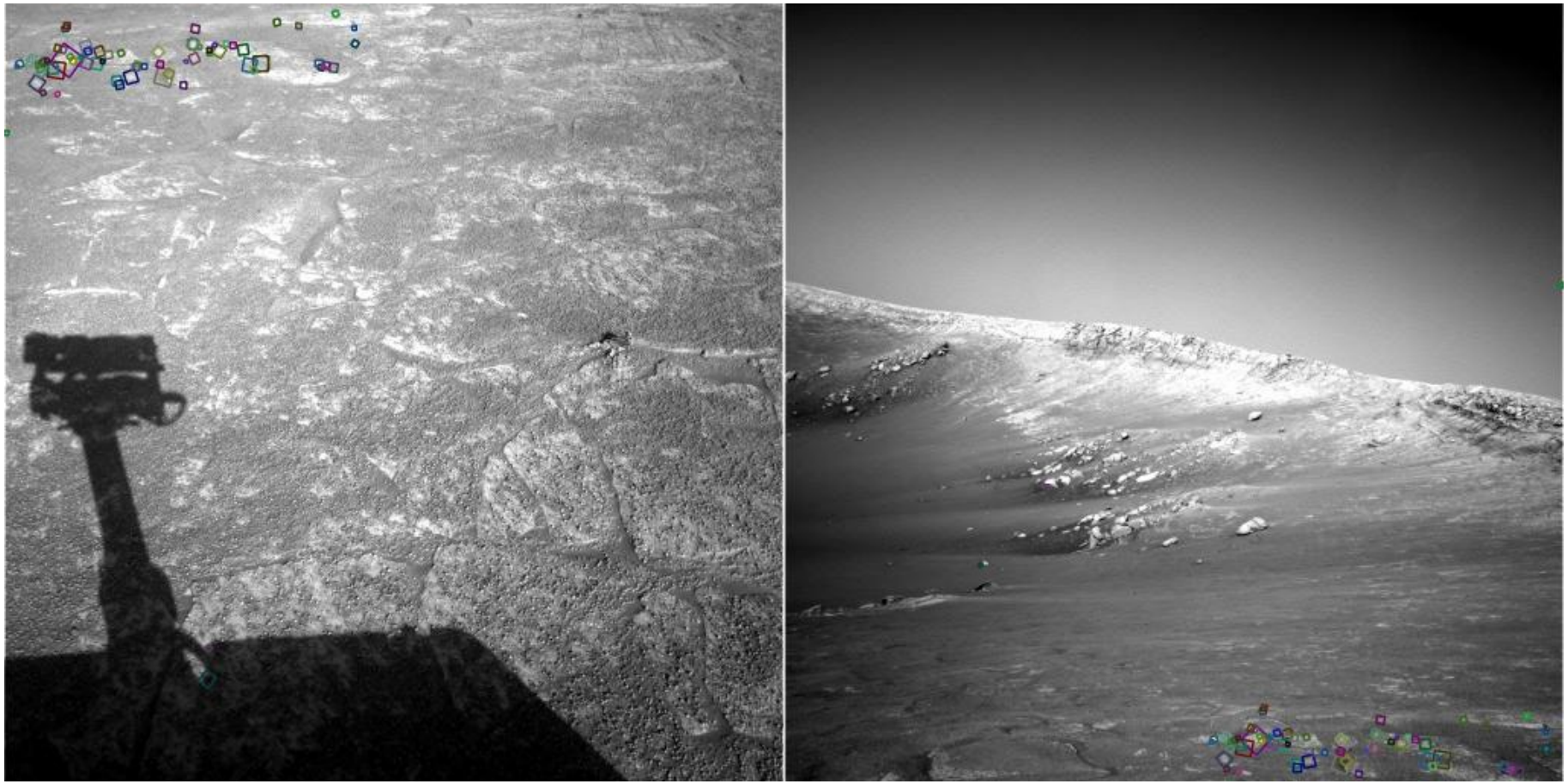
## Manual selection often nontrivial



NASA Mars Rover images
(Figure by Noah Snavely)

# Corresponding key points selection

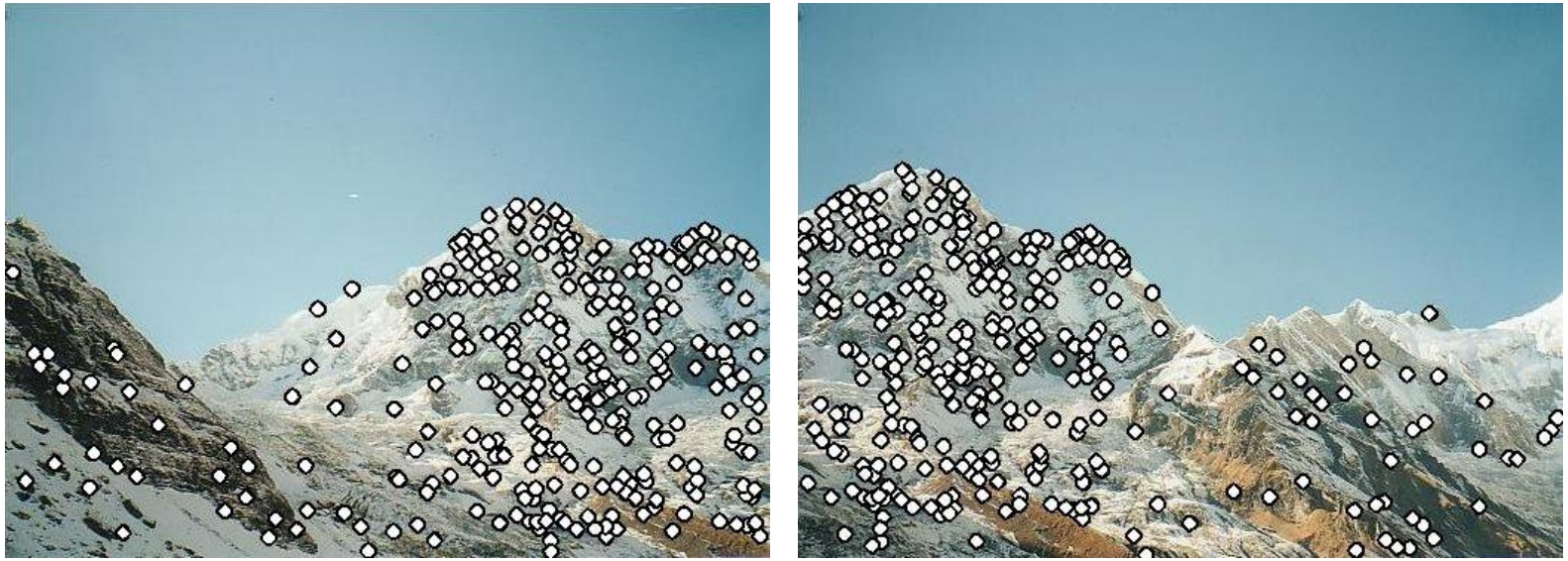
## Manual selection often nontrivial



NASA Mars Rover images
with SIFT feature matches
(Figure by Noah Snavely)

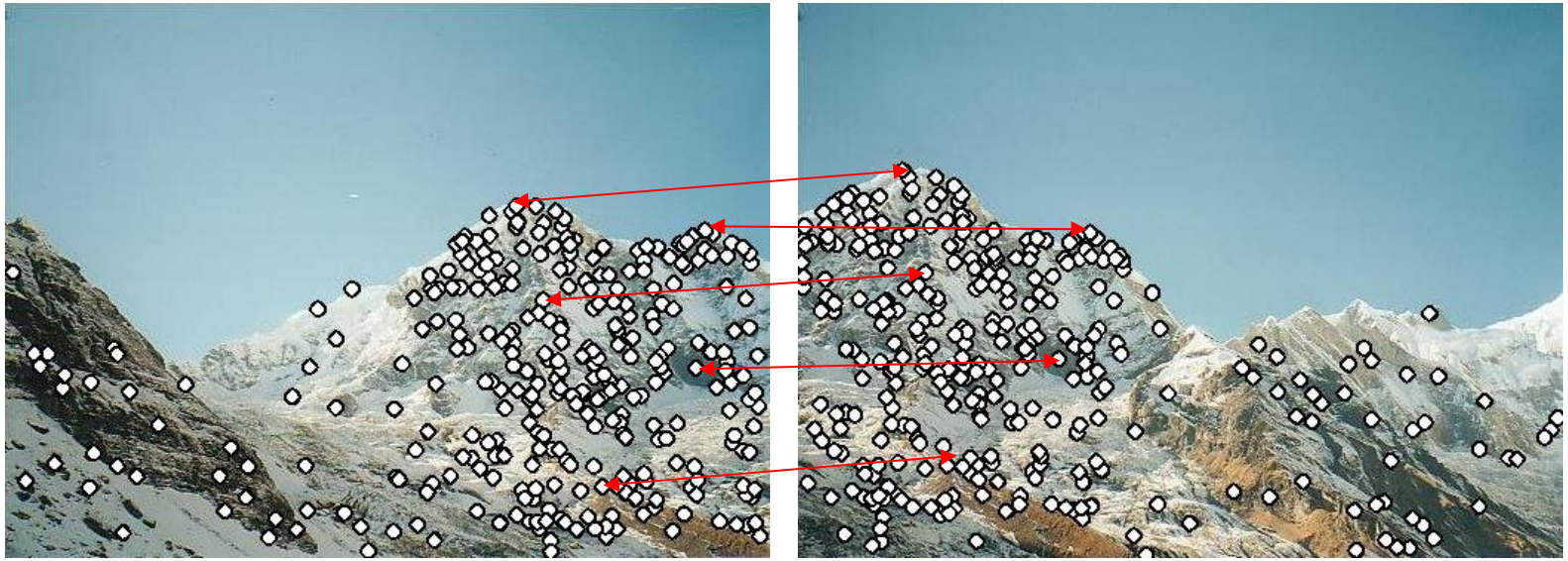
# A case study: Automatic panorama creator

# A case study: Automatic panorama creator



- Standard procedure:
  - Detect interest points (key-points) in both images

# A case study: Automatic panorama creator



- Standard procedure:
  - Detect interest points (key-points) in both images
  - Find pairs of corresponding points

# A case study: Automatic panorama creator

- Standard procedure:
  - Detect interest points (key-points) in both images
  - Find pairs of corresponding points
  - Use these pairs for image registration ← (e.g., RANSAC/least-squares transformation model estimation)

# Efficient keypoint detector requirements

- Requirement 1:
  - Detect the same structure *independently* in each image.

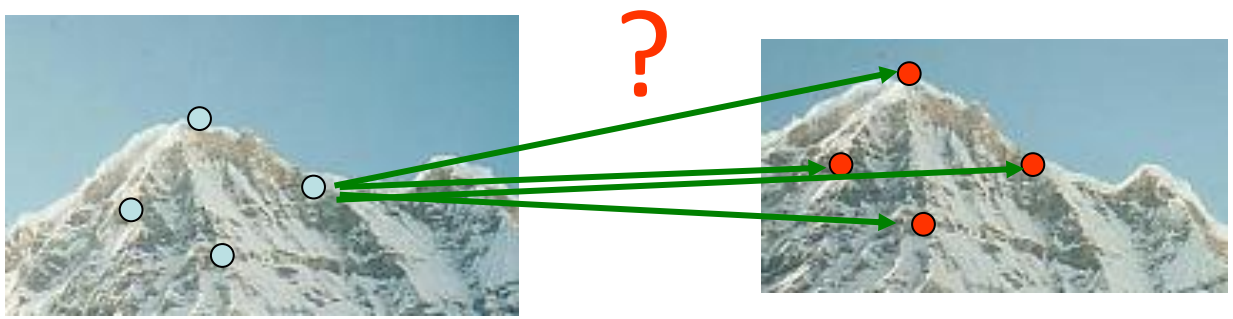
Try random sampling?

*Bad idea: By random sampling in each image, we will not likely detect the same points.*

A *detector* with a high detection repeatability is required!

# Efficient keypoint detector requirements

- Requirement 1:
  - Detect the same structure *independently* in each image.
- Requirement 2:
  - For each point find a corresponding point in the other image.




A reliable and distinctive *descriptor* is required!

# Outline of this lecture

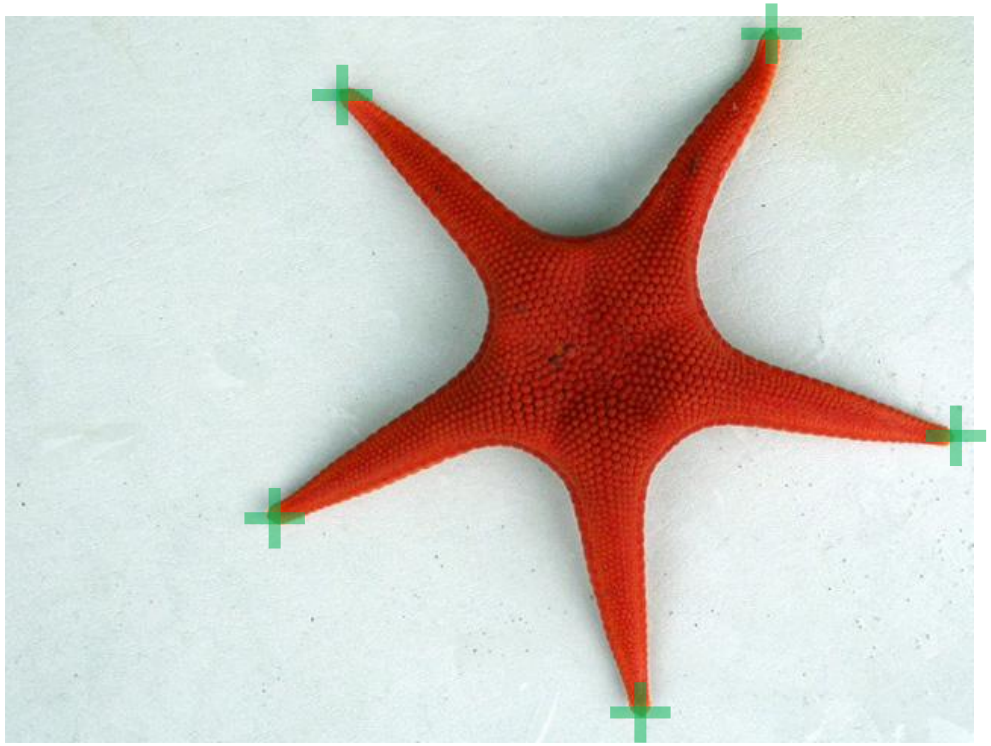
1. Keypoint DETECTION
2. Keypoint DESCRIPTION
3. Keypoint MATCHING

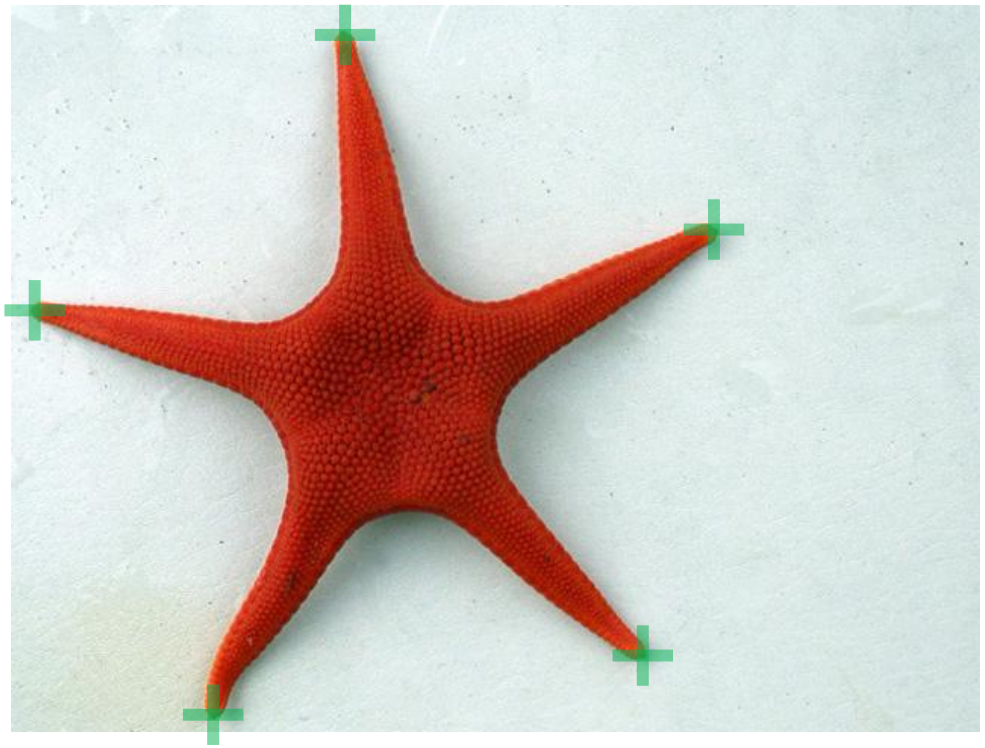
Machine Perception

# SINGLE SCALE KEY-POINT DETECTION

# Corners as keypoints

- Distinctive and repeatedly occurring on the same structures even if the structure changes pose in 3D

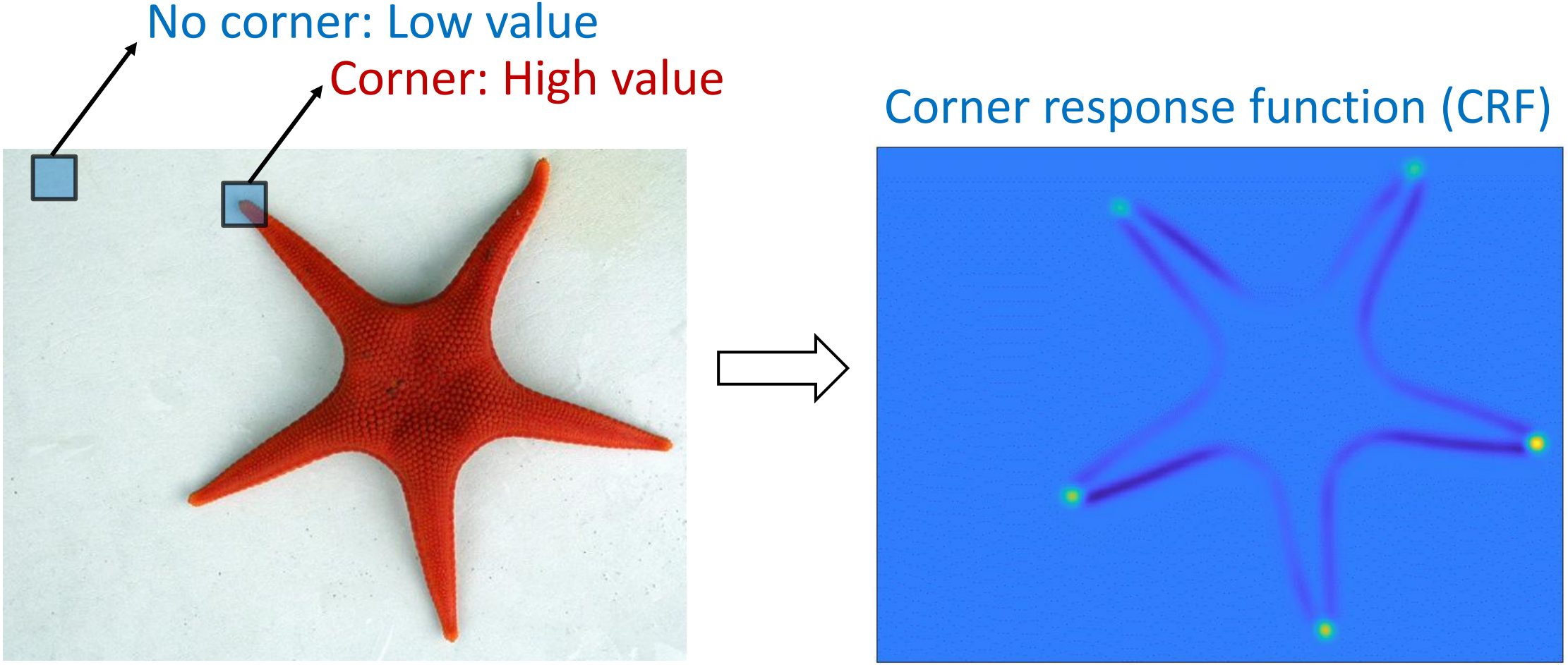




C.Harris and M.Stephens. "A Combined Corner and Edge Detector." *Proceedings of the 4th Alvey Vision Conference*, 1988.

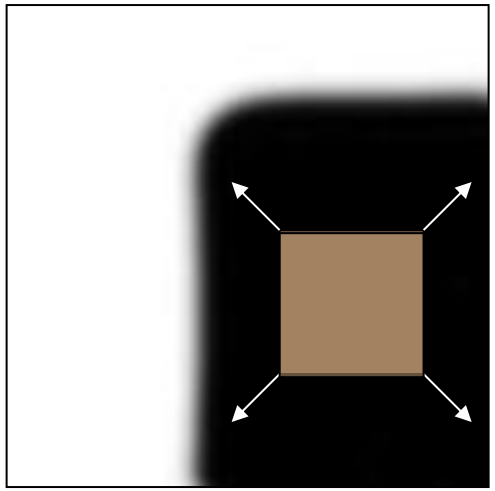
# Require a corner response function – CRF

- An operator that gives a strong response on the corner structure

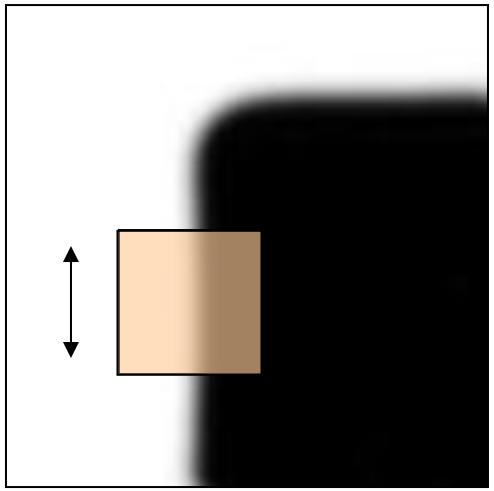


# Corner response function: Intuition

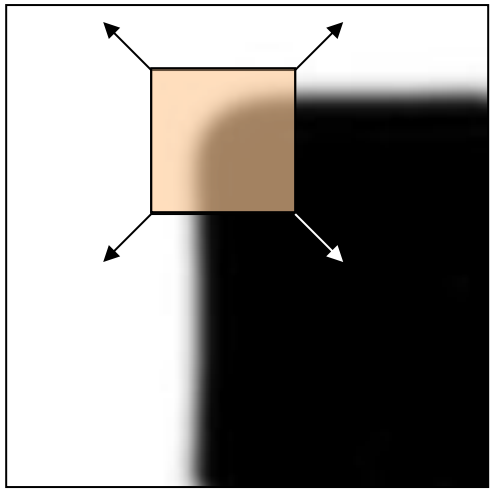
- A good corner detector criteria: Self similarity
  - Observe a small window $R$ around a potential corner (locality).
  - A small shift in window in any direction results in a large intensity change (good localization)



"Flat" region:
A small shift in any direction does not cause an intensity change.



"Edge":
No change when shifting along the edge, otherwise there is a change.



"Corner":
A shift in any direction significantly changes the local intesity.

# Harris corner detector

- The intensity change for a shift [u,v]:

  (weighted self-similarity function)

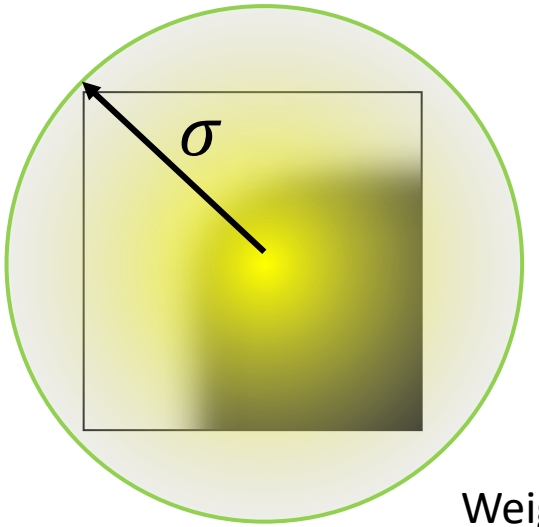
$E_R(u,v)$ should quickly drop for a small displacement (u,v)

$$E_R(u,v) = \sum_{x,y \in R} w(x,y)\left(I(x,y) - I(x+u, x+v)\right)^2$$

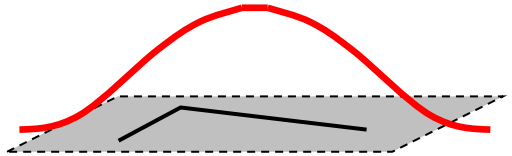
Weight function

Intensity before the shift

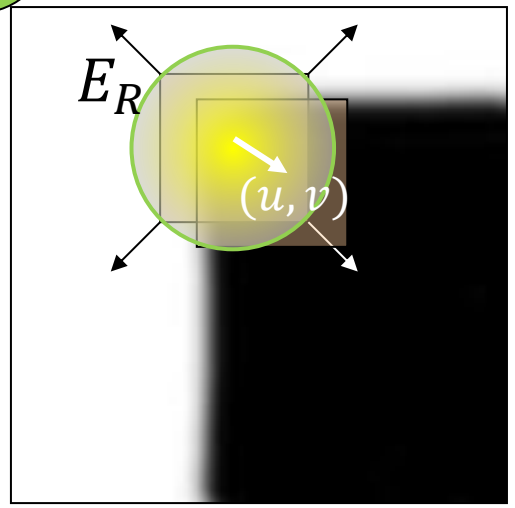
Intensity afer the shift




Weight function $w(x,y)$ =



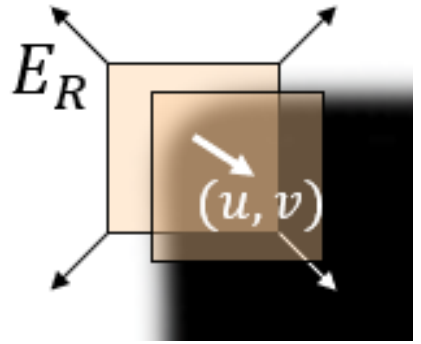
Gaussian kernel $G(\sigma)$



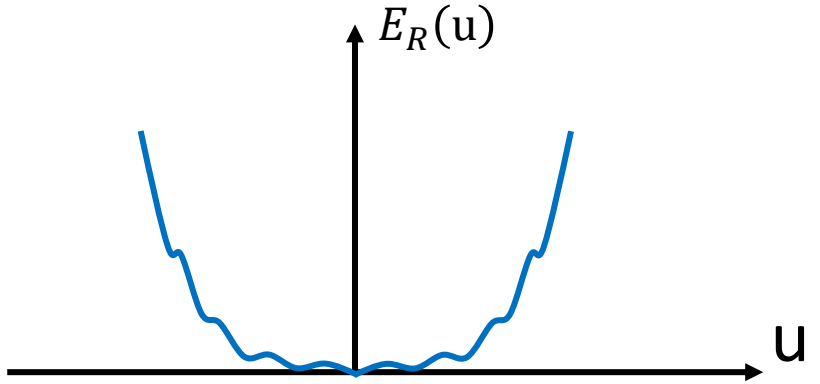

The $\sigma$ specifies the region $R$ size!

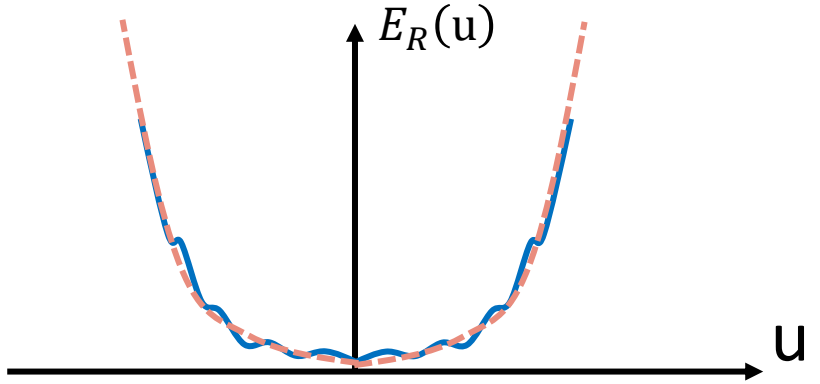
# Harris corner detector

$$E_R(u,v) = \sum_{x,y \in R} w(x,y)\left(I(x,y) - I(x+u, x+v)\right)^2$$




- The function $E_R$ is minimal at (u=0,v=0), i.e., $E_R(0,0) = 0$, and increases as we move away.
- We are interested in analyzing the shape, i.e., *how fast* the difference increases.
- Approach:
  - Approximate $E_R$ at (u=0,v=0) by a quadratic function and analyze its shape.



# Harris corner detector

- Linearize the image values $\mathrm{I}(\mathrm{x}, \mathrm{y})$ for small shifts $(u, v)$:

$$\mathrm{I}(x+u, y+v) \approx I(x,y) + \left[I_x(x,y), I_y(x,y)\right] \begin{bmatrix} u \\ v \end{bmatrix}$$

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

- Plug into the weighted self-similarity:

$$E_R(u,v) = \sum_{x,y \in R} w(x,y)\left(I(x,y) - I(x+u, x+v)\right)^2$$

$$E_{\mathrm{R}}(u,v) \approx \sum_{(x,y)\in R} w(x,y,)\left(\left[I_x(x,y), I_y(x,y)\right]\begin{bmatrix} u \\ v \end{bmatrix}\right)^2$$

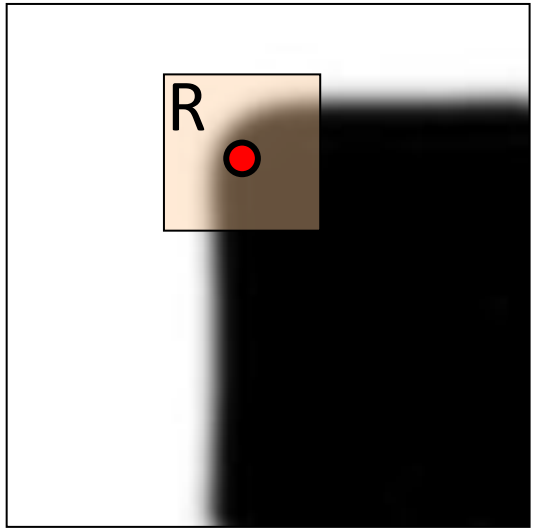
$$\approx [u,v]\left(\sum_{x,y \in R} w(x,y) \begin{bmatrix} I_x(x,y) \\ I_y(x,y) \end{bmatrix} [I_x(x,y), I_y(x,y)]\right)\begin{bmatrix} u \\ v \end{bmatrix}$$

# Harris corner detector

$$E_R(u,v) = \sum_{x,y \in R} w(x,y)\left(I(x,y) - I(x+u, x+v)\right)^2$$

- For small shifts $(u, v)$ *E* can be linearly approximated by:

$$E_R(u,v) \approx [u,v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$




with *M* 2x2 matrix of image derivatives:

$$M = \begin{bmatrix} \sum_{(x,y)\in R} w(x,y) I_x^2(x,y) & \sum_{(x,y)\in R} w(x,y) I_x(x,y) I_y(x,y) \\ \sum_{(x,y)\in R} w(x,y) I_x(x,y) I_y(x,y) & \sum_{(x,y)\in R} w(x,y) I_y^2(x,y) \end{bmatrix}$$

A weighted sum over the region R in which we are verifying a corner (centered at red dot in the sketch above – the red dot is thus treated as the origin of coordinate system)

Construction of $M$ can be made more efficient!

# Construction of M (*for all locations* ●)

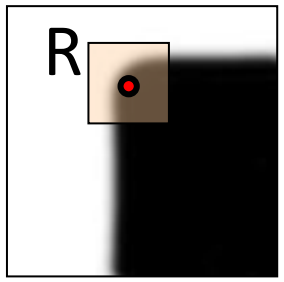
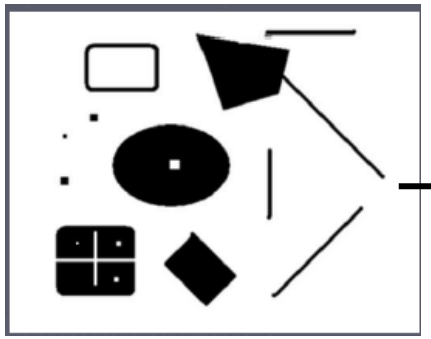



Image $I$

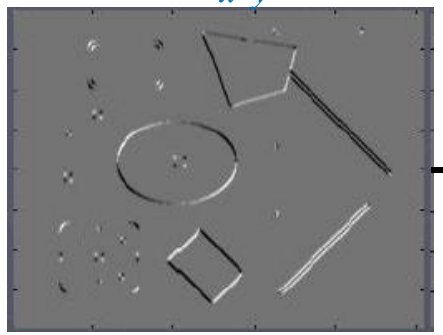


$I_x$

$G(\sigma) * I_x^2$

$I_y$

$G(\sigma) * I_y^2$



$I_x I_y$

$G(\sigma) * I_x I_y$

Note, M is different for each location ●

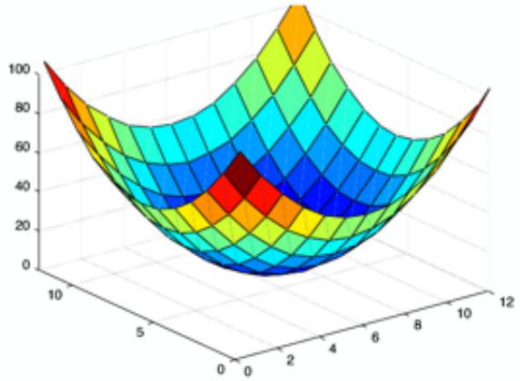
$$M = \begin{bmatrix} \sum_{(x,y)\in R} w(x,y) I_x^2(x,y) & \sum_{(x,y)\in R} w(x,y) I_x(x,y) I_y(x,y) \\ \sum_{(x,y)\in R} w(x,y) I_x(x,y) I_y(x,y) & \sum_{(x,y)\in R} w(x,y) I_y^2(x,y) \end{bmatrix}$$

$$M = \begin{bmatrix} G(\sigma) * I_x^2 & G(\sigma) * I_x I_y \\ G(\sigma) * I_x I_y & G(\sigma) * I_y^2 \end{bmatrix}$$

# Harris corner detection

- We have approximated the autocorrelation function locally by a quadratic form

$$\mathrm{E_R}(u,v) \approx [u,v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$



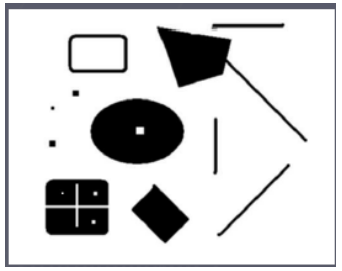



Image $I$

- Matrix $M$ is the covariance matrix of region gradients:

$$M = \begin{bmatrix} G(\sigma) * I_x^2 & G(\sigma) * I_x I_y \\ G(\sigma) * I_x I_y & G(\sigma) * I_y^2 \end{bmatrix}$$

$I_x$

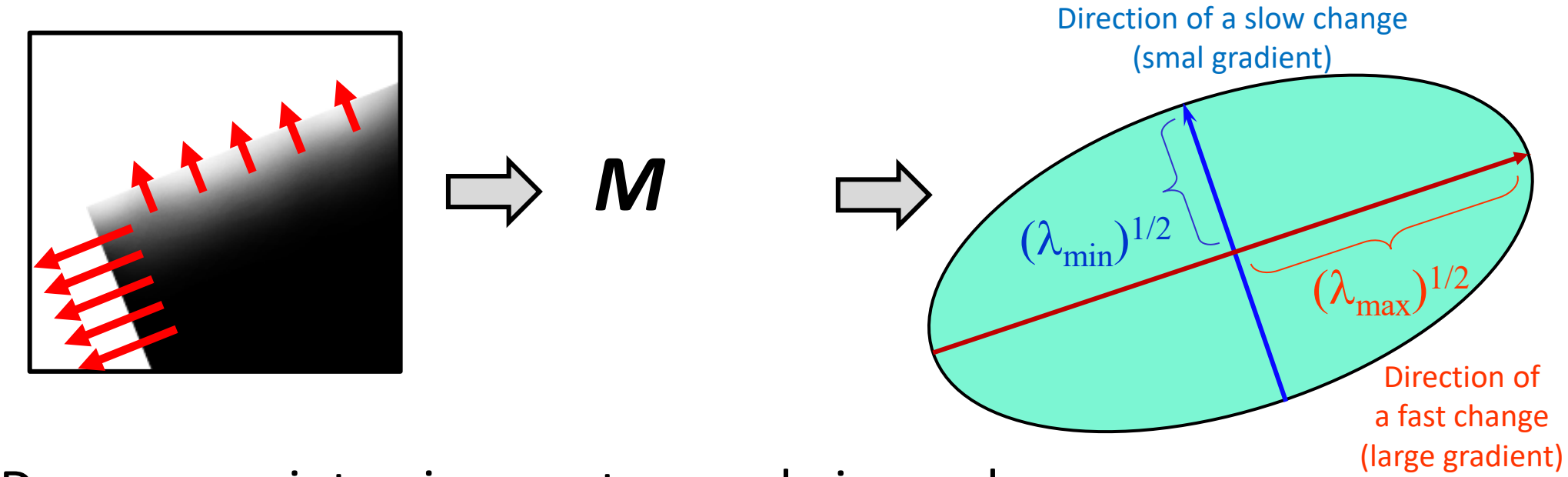
$I_y$

$I_x I_y$

- Quadratic function *"peakiness"* can be quantified by analyzing the matrix $M$

# The Covariance matrix analysis

- Visualize the covariance matrix as an ellipse...




- Decompose into eigenvectors and eigenvalues:

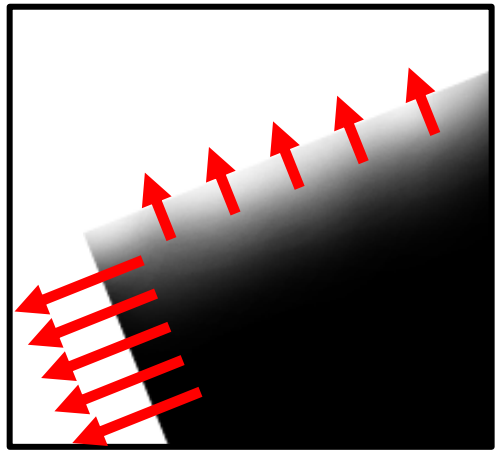
$$M = R \begin{bmatrix} \lambda_{\max} & 0 \\ 0 & \lambda_{\min} \end{bmatrix} R^T$$

eigen values (ellipse SCALING)

eigen vectors
(ellipse ROTATION)

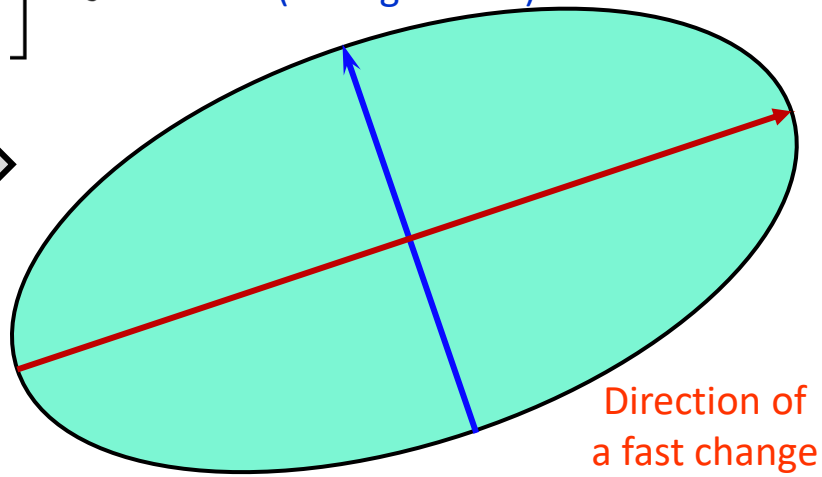
# The Covariance matrix analysis

- Visualize the covariance matrix as an ellipse...



$$M = R \begin{bmatrix} \lambda_{\max} & 0 \\ 0 & \lambda_{\min} \end{bmatrix} R^T$$

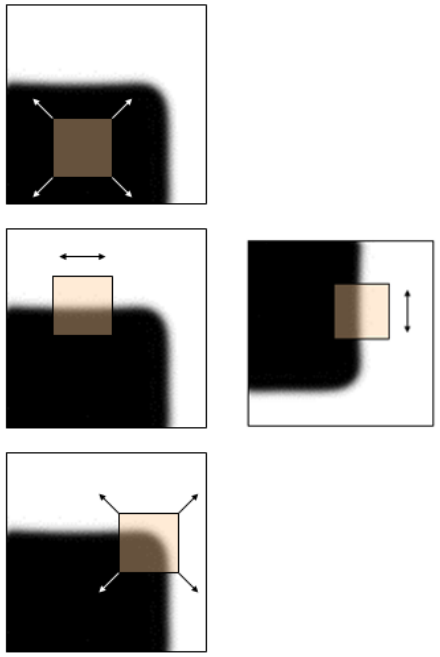
$\Rightarrow$ ***M*** $\Rightarrow$



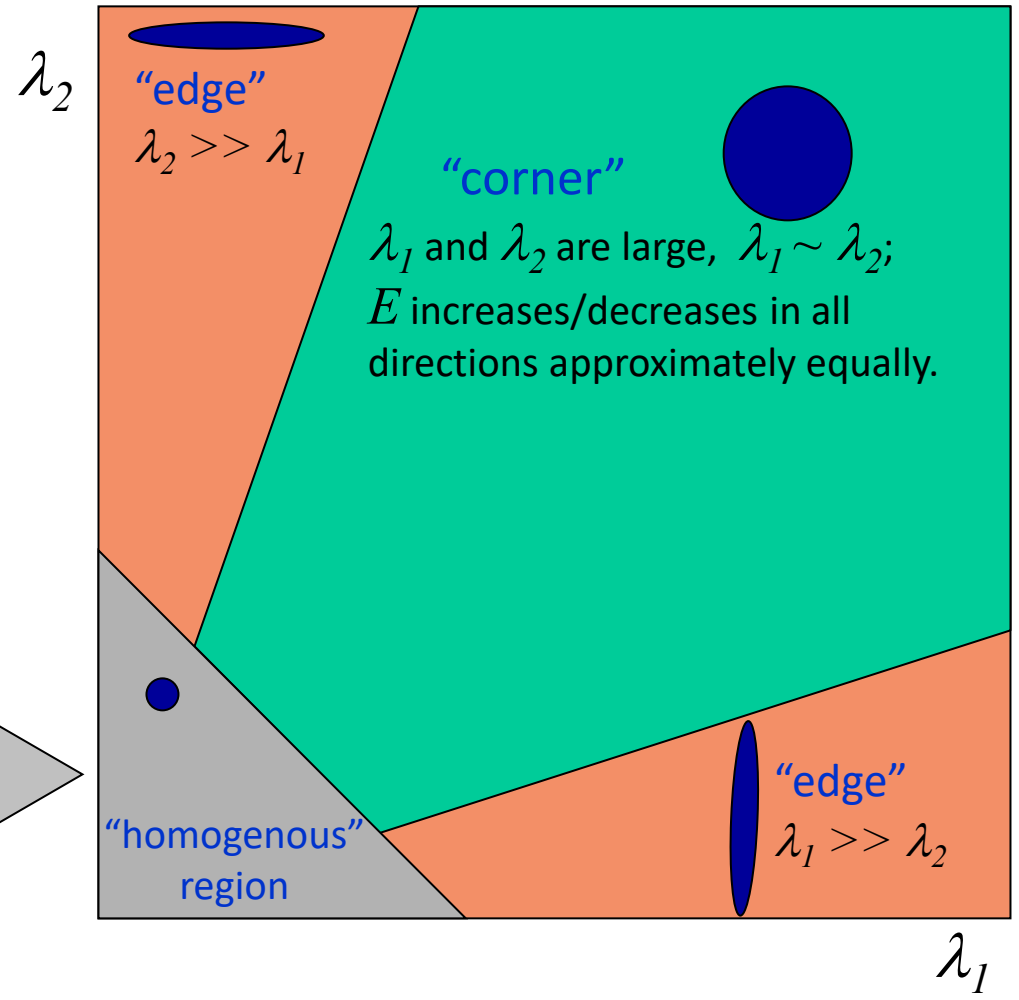

- Decompose into eigenvectors and eigenvalues:
  - A corner has a strong gradient in both major directions!
  - A corner is present when both eigenvalues are large.

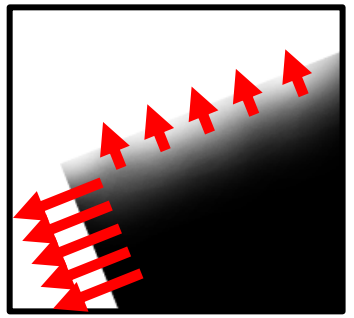
# Eigen values: Interpretation

- Corner detection by eigenvalues of ***M***:

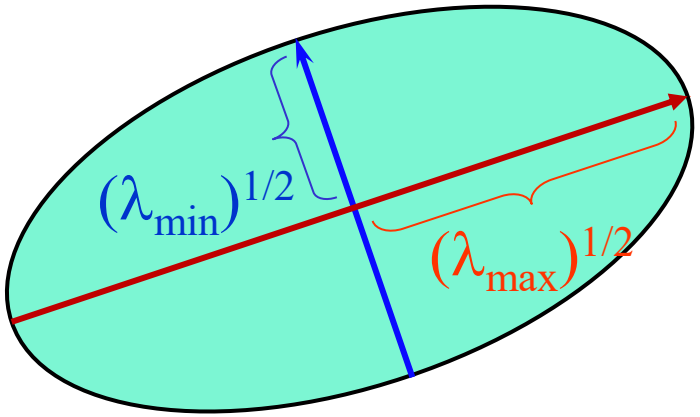


$\lambda_1$ and $\lambda_2$ small;
$E$ almost constant in all directions.



Direction of a slow change (smal gradient)




Direction of a fast change (large gradient)

# The Harris corner response function

- Problem: Calculating the eigenvalues at each pixel is computationally intensive!
- Solution:
  - Possible to compute functions of eigenvalues: $\det(\mathbf{M}) = \lambda_1\lambda_2, \text{trace}(\mathbf{M}) = \lambda_1 + \lambda_2$
  - We are actually not interested necessarily in the individual eigenvalues, but a corner response function that will be large when eigenvalues are comparable and large

$$r = \frac{\lambda_1}{\lambda_2} \quad \frac{\text{trace}^2(\mathbf{M})}{\det(\mathbf{M})} = \frac{(\lambda_1+\lambda_2)^2}{\lambda_1\lambda_2} = \frac{(r+1)^2}{r} = \alpha^{-1}$$

$$\det(\mathbf{M}) - \alpha\text{trace}^2(\mathbf{M}) = 0 \quad \text{(for r = 1, } \alpha = 1/4\text{)}$$

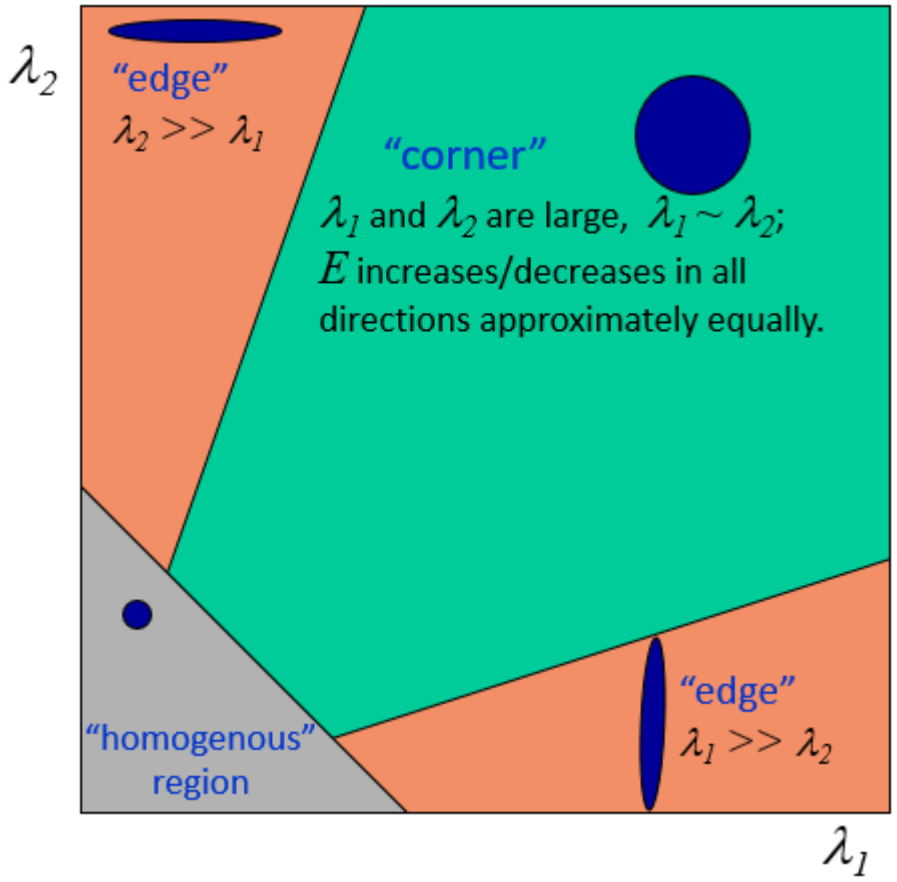
Harris corner response function (with $\alpha$ set to small value):

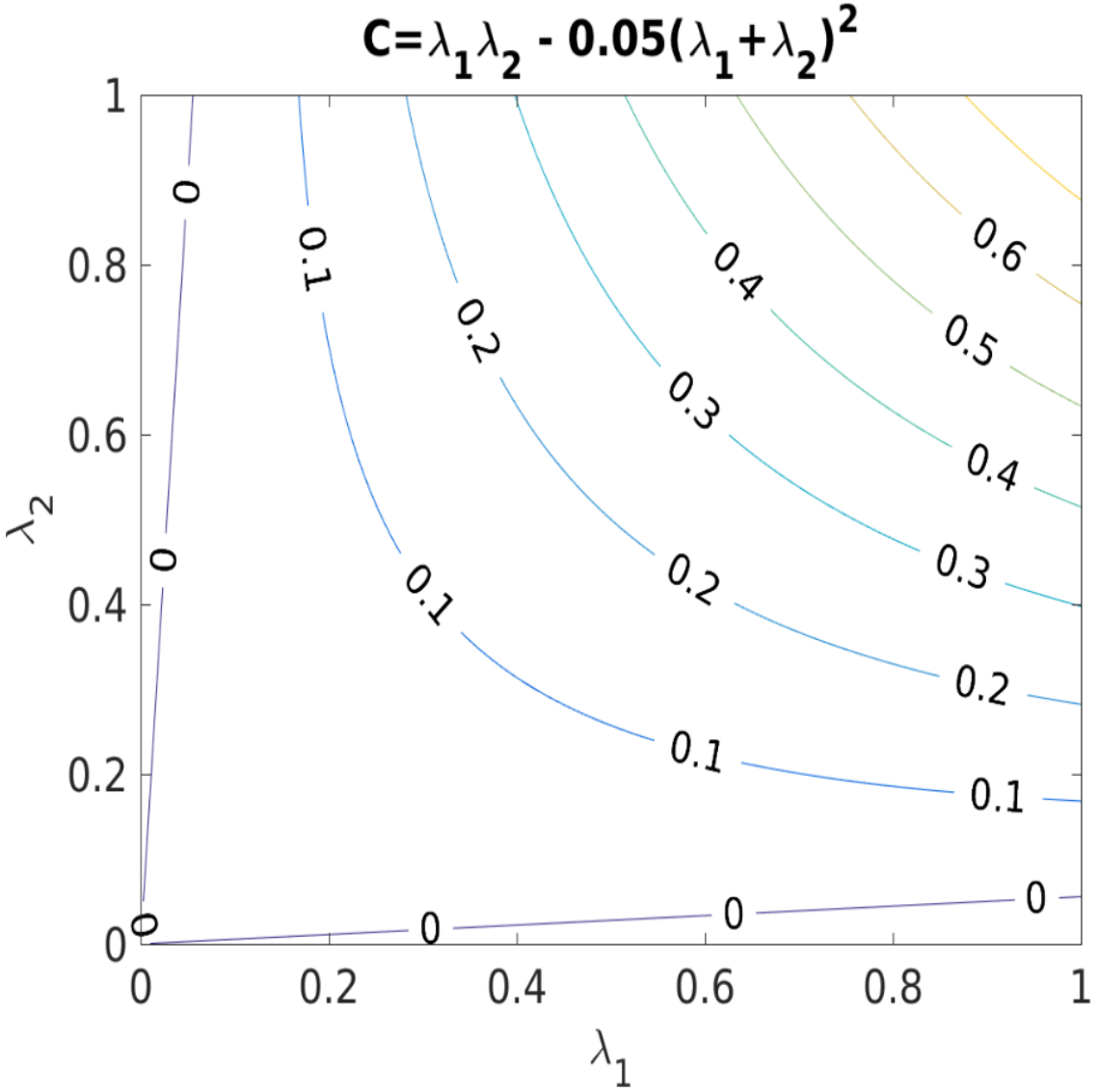
$$c = |M| - \alpha\text{trace}(M)^2$$

# The Harris corner response function

- In practice, a small $\alpha$ (0.04-0.06) is applied

$$c = |M| - \alpha \text{trace}(M)^2$$



# The Harris corner response function

- In practice, fix $\alpha$ (0.04-0.06) and check if corner response function exceeds a threshold

$$\det(\mathbf{M}) - \alpha \mathrm{trace}^2(\mathbf{M}) > t$$

- We can calculate the *Determinant* and *Trace* directly:

$$M = \begin{bmatrix} A & C \\ C & B \end{bmatrix} \qquad \begin{aligned} \det(\mathbf{M}) &= AB - C^2 \\ \mathrm{trace}(\mathbf{M}) &= A + B \end{aligned}$$

$$AB - C^2 - \alpha(A + B)^2 > t$$

$$M = \begin{bmatrix} G(\sigma) * I_x^2 & G(\sigma) * I_x I_y \\ G(\sigma) * I_x I_y & G(\sigma) * I_y^2 \end{bmatrix}$$

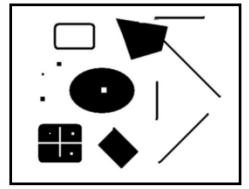
# Harris corner detector: Summary

- Calculate the covariance matrix
  (by virtue of autocorrelation)

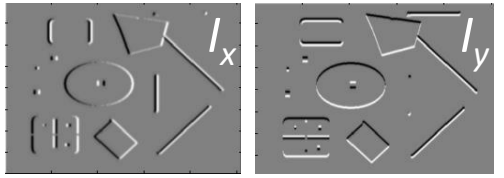
$$M = \begin{bmatrix} G(\sigma) * I_x^2 & G(\sigma) * I_x I_y \\ G(\sigma) * I_x I_y & G(\sigma) * I_y^2 \end{bmatrix}$$

$$\det(\mathbf{M}) - \alpha \mathrm{trace}^2(\mathbf{M}) > t$$

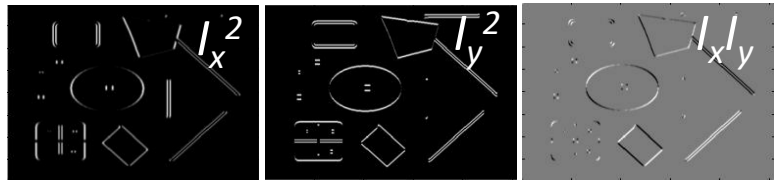
0. The source image



1. Image derivatives




2. Squared derivatives




3. Gaussian filtered squared derivatives $g(s_l)$

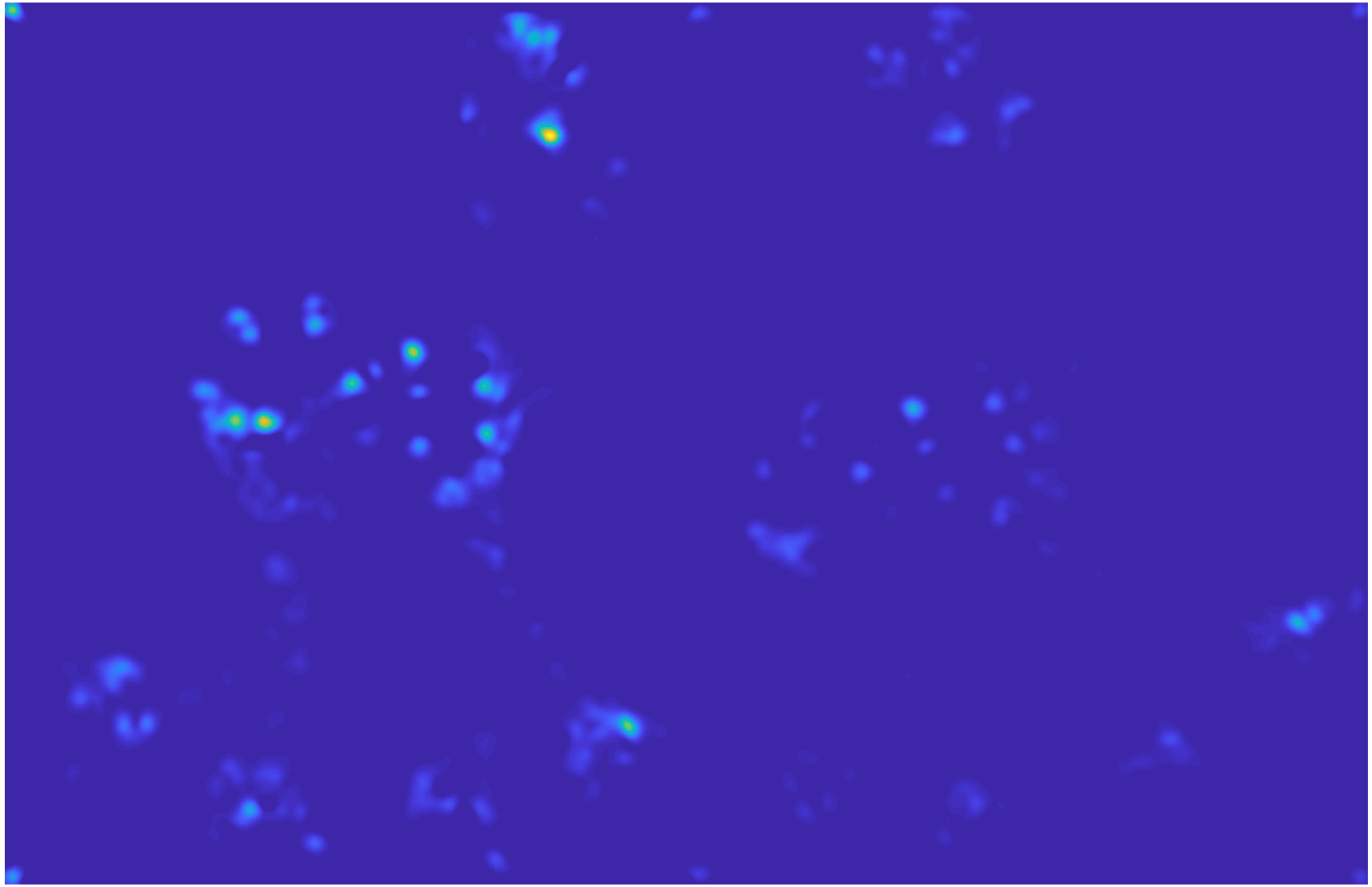

4. Corner response function:

$$\mathrm{c(I)} = \det[M] - \alpha[\mathrm{trace}^2(M)]$$
$$= g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$$


5. Apply a non-maxima suppression

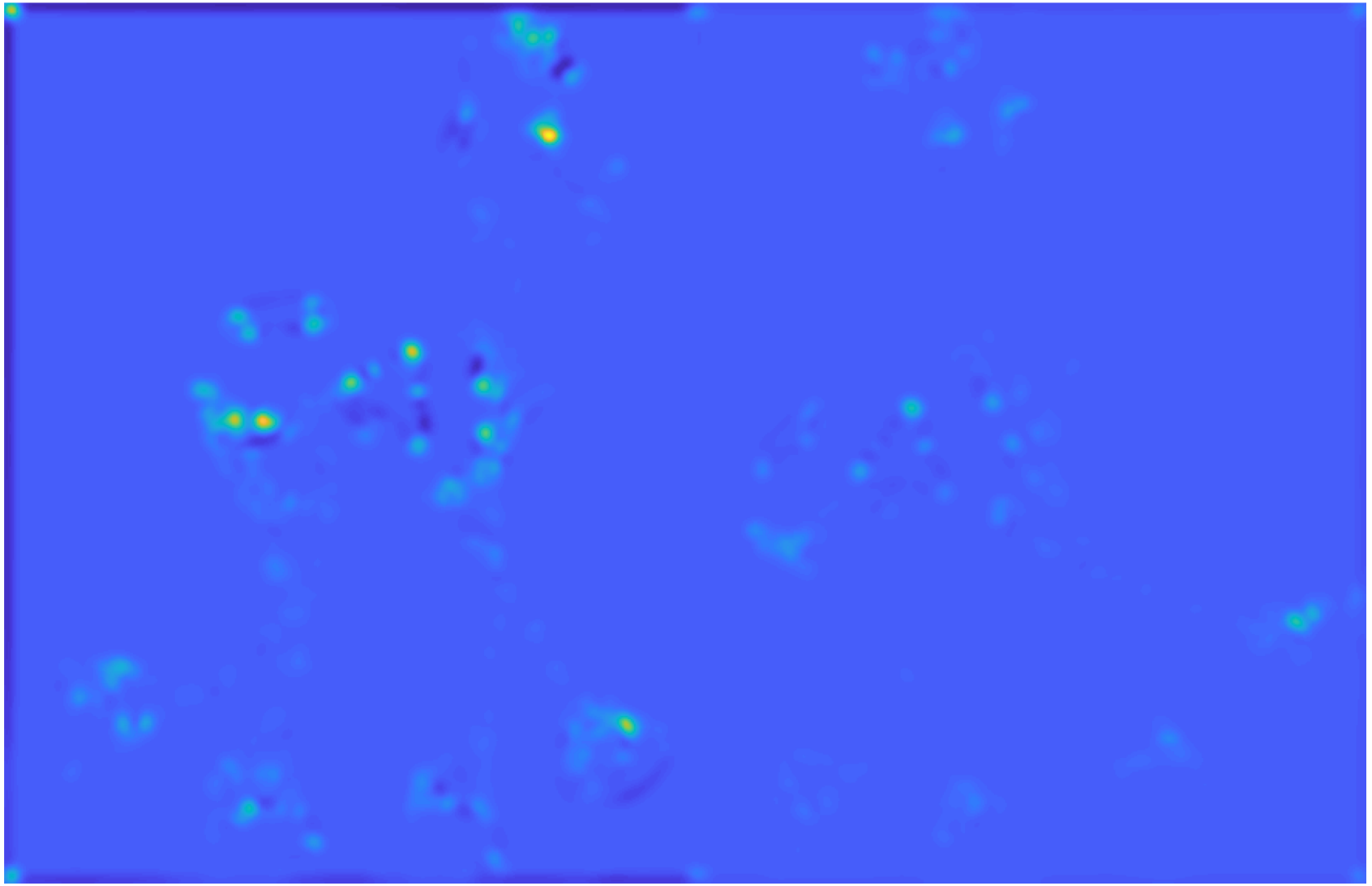
# Harris corner detector: Summary

# Harris corner detector: Summary



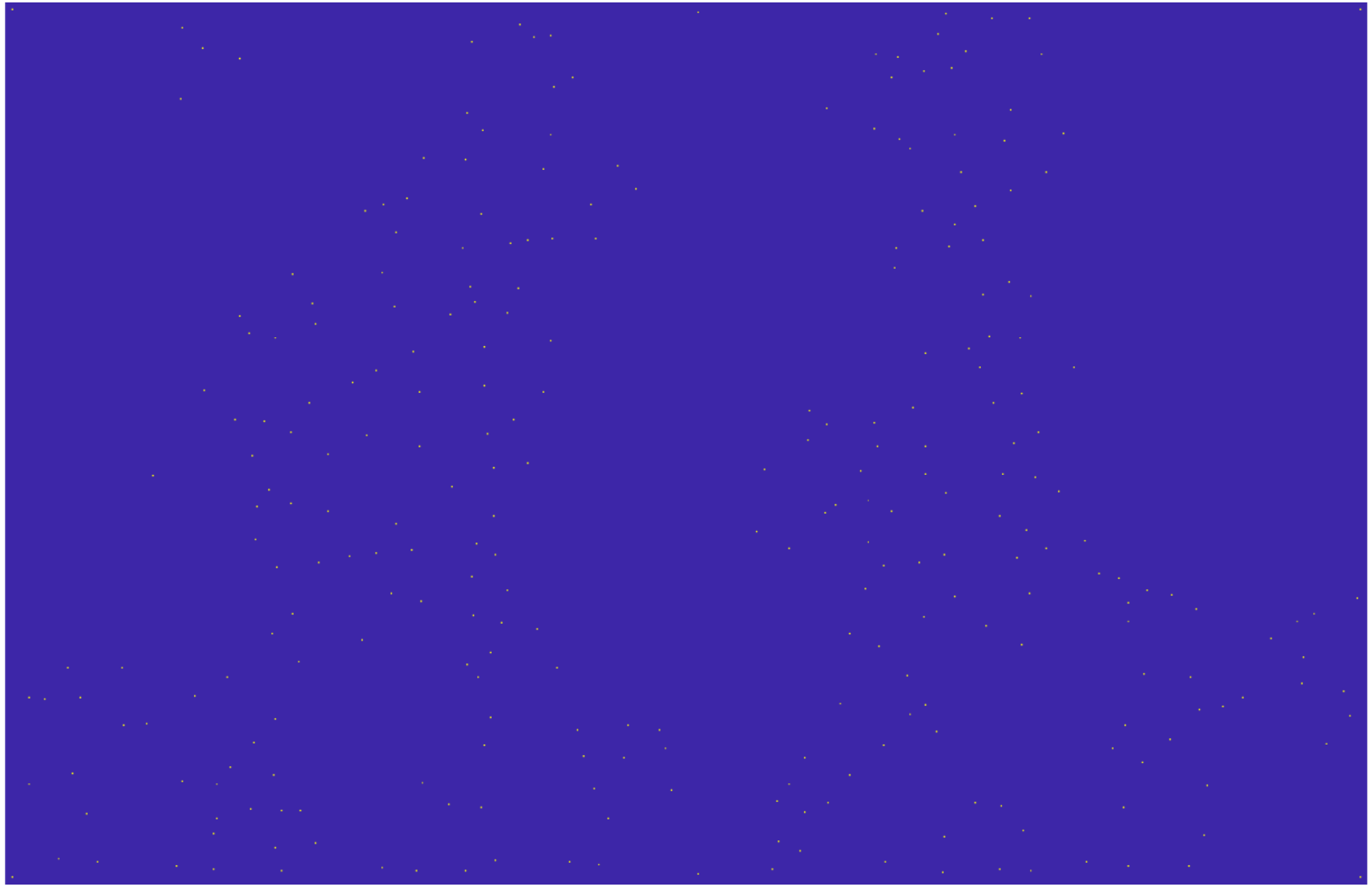
- The Corner response function: $c(I) = g(I_x^2)g(I_y^2) - [g(I_x I_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2$

# Harris corner detector: Summary



- Set values lower than a threshold to zero: $c(c < \text{threshold}) = 0$

# Harris corner detector: Summary

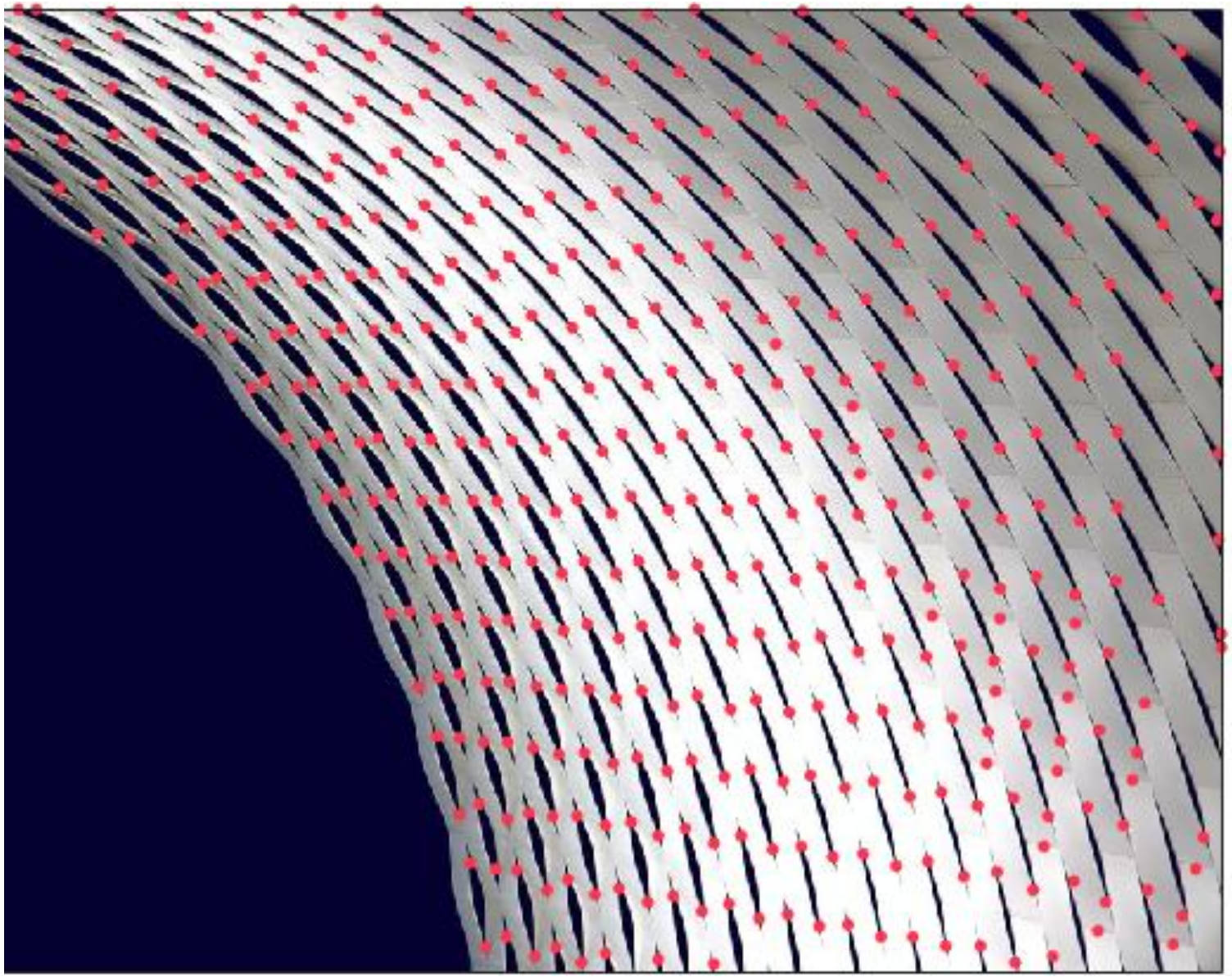


- Find the local maxima in *c(I)*

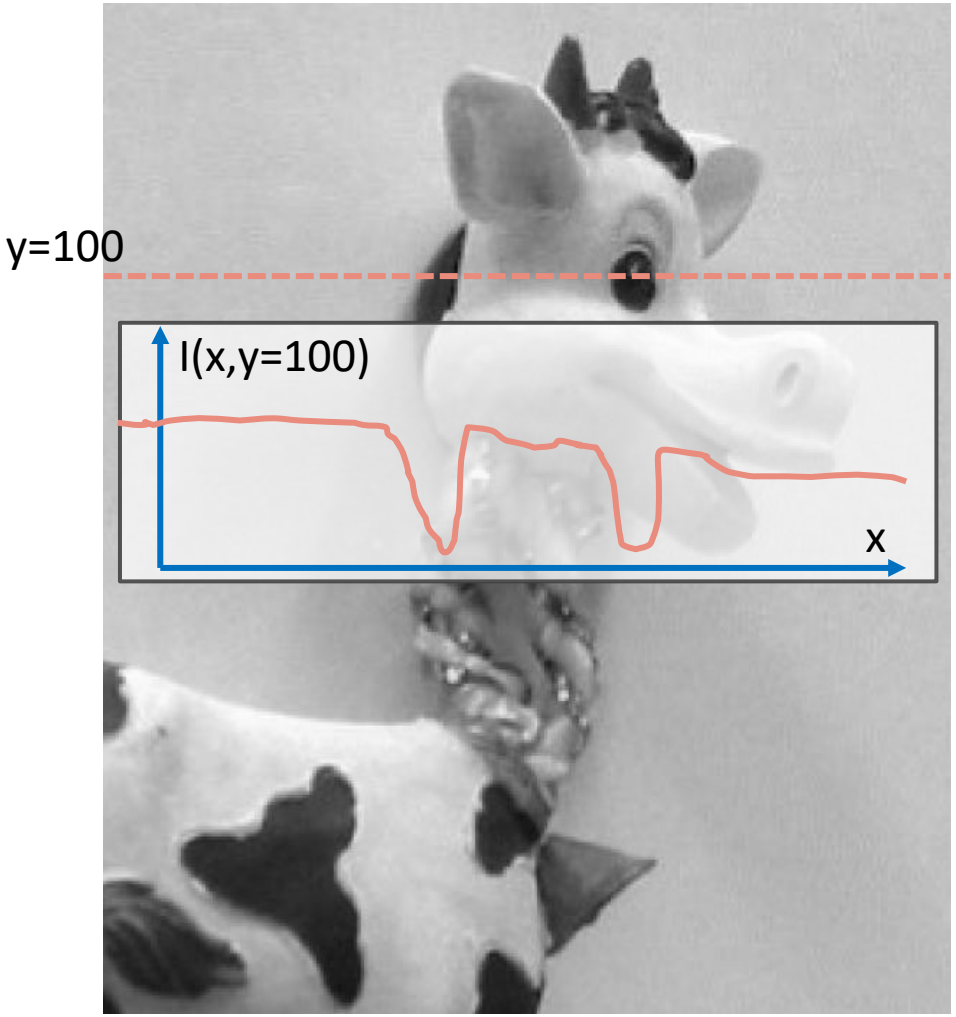
# Harris corner detector: Summary

- Detected Harris corners

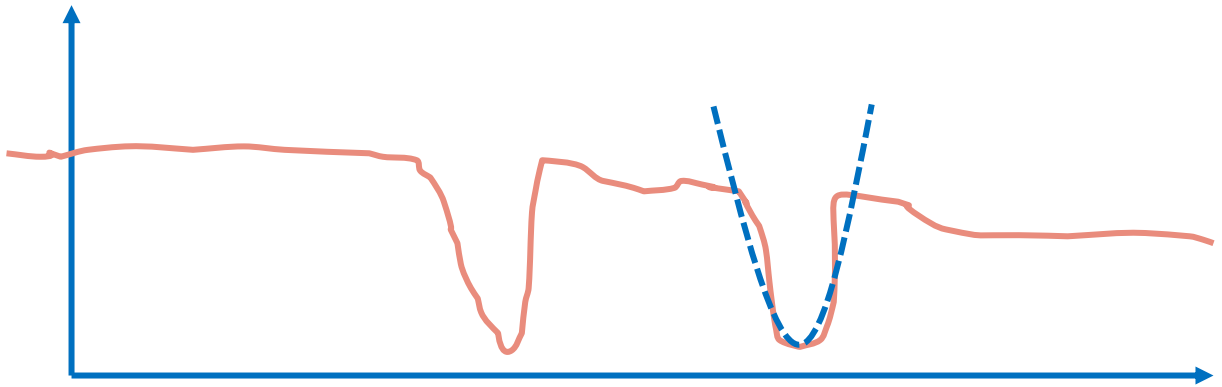
# Harris detector



# Harris corner detector: Motivation

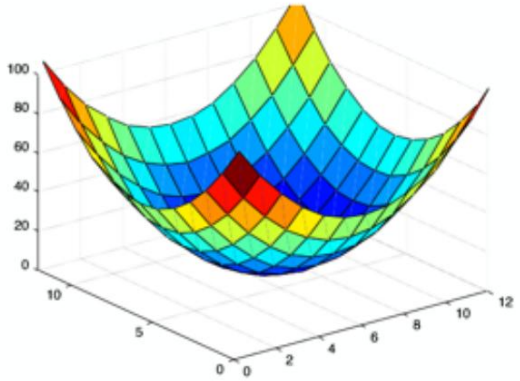



Intensity values extracted along the image row



How about fitting the quadratic function to the intensity values locally?

In 2D image:



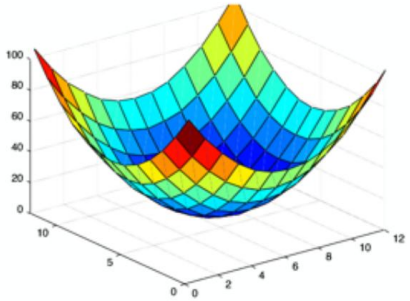

Response function:

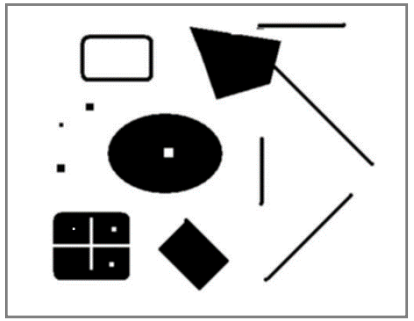
Curvature of the fitted function

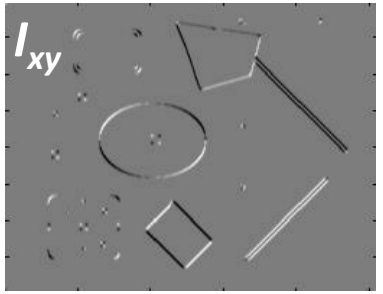
# The Hessian corner detector

- Curvature measure = Determinant of a Hessian

$$\text{Hessian}(I) = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$$







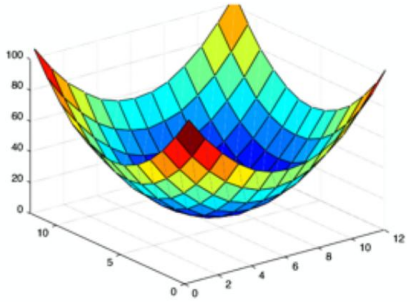
*Note: these are second order derivatives!*
*(Recall what Hessian means → a measure of local curvature)*

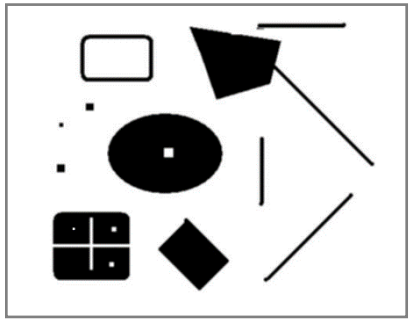
*Intuition:* Find strong gradients in two orthogonal directions

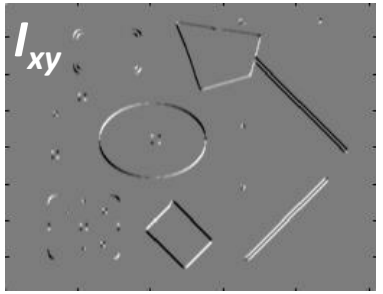
# The Hessian corner detector

- Curvature measure = Determinant of a Hessian

$$\text{Hessian}(I) = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$$







*Note: these are second order derivatives!*
*(Recall what Hessian means*
*→ a measure of local curvature)*

$$\det(\text{Hessian}(I)) = I_{xx}I_{yy} - I_{xy}^2$$

In Matlab: $I_{xx}.*I_{yy} - (I_{xy}).\hat{}\,2$

In Python: $I_{xx} * I_{yy} - I_{xy}{**}2$

# The Hessian corner detector

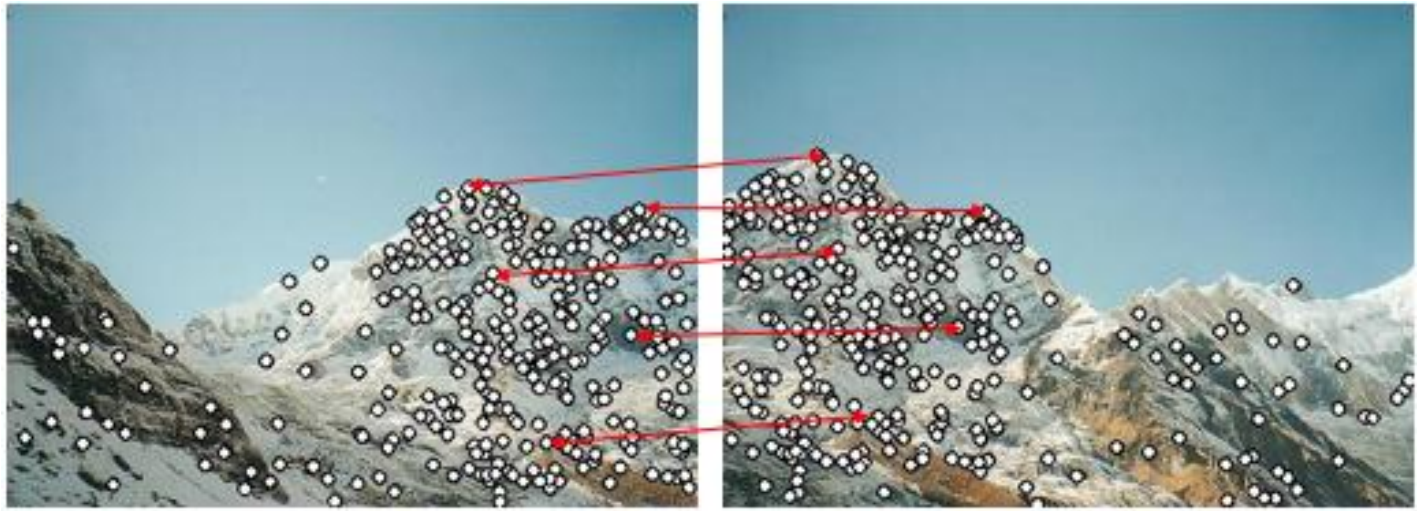
*Result:* responses on corners and blobs.

# A number of keypoint detectors exist

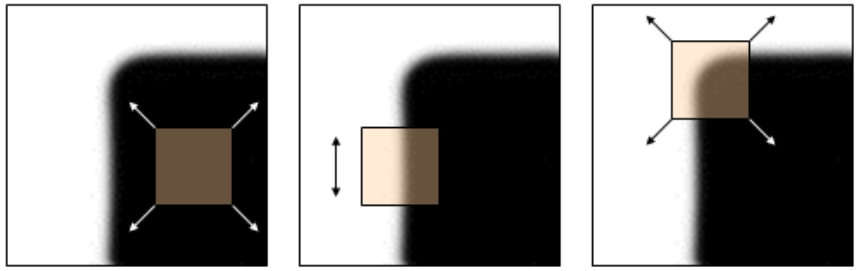
- Hessian & Harris [Beaudet ‘78], [Harris ‘88]
- Laplacian, DoG [Lindeberg ‘98], [Lowe 1999]
- Harris-/Hessian-Laplace [Mikolajczyk & Schmid ‘01]
- Harris-/Hessian-Affine [Mikolajczyk & Schmid ‘04]
- MSER [Matas ‘02]
- FAST , and lots of others
- A very good tutorial ECCV 2012.
- *Learning-based detectors gaining traction.*
- *These detector have become building blocks of numerous computer vision applications!*

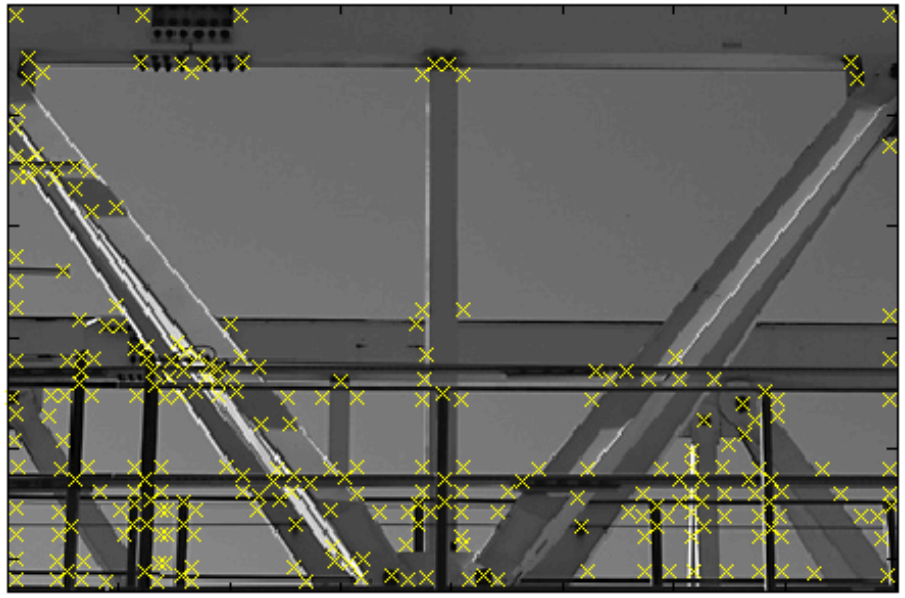
# Previously at MP...

- Automatic image stitching
  - Keypoints detection
  - Keypoint description
  - Keypoint matching



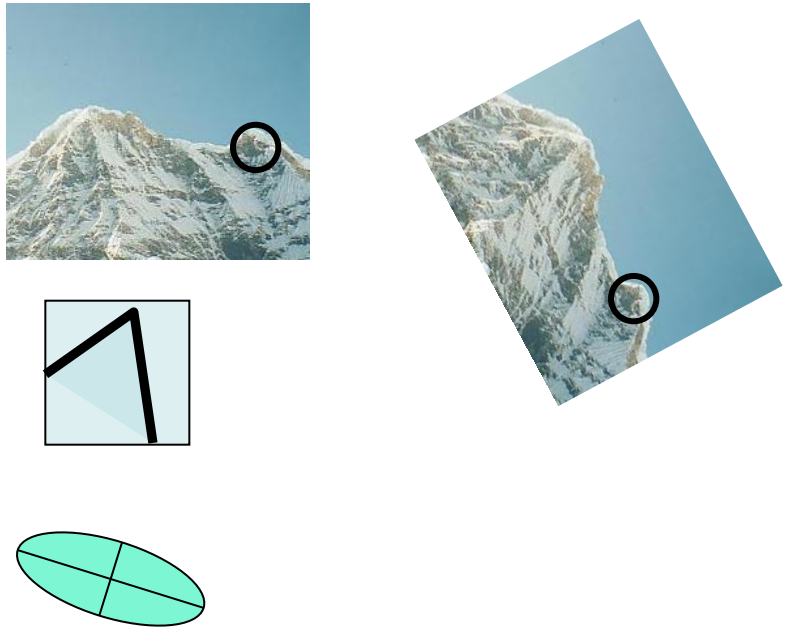
- Key-point detection
  - Analysis of gradient distribution
  - Harris, Hessian





# Harris/Hessian detector: properties

- Is it rotation invariant?

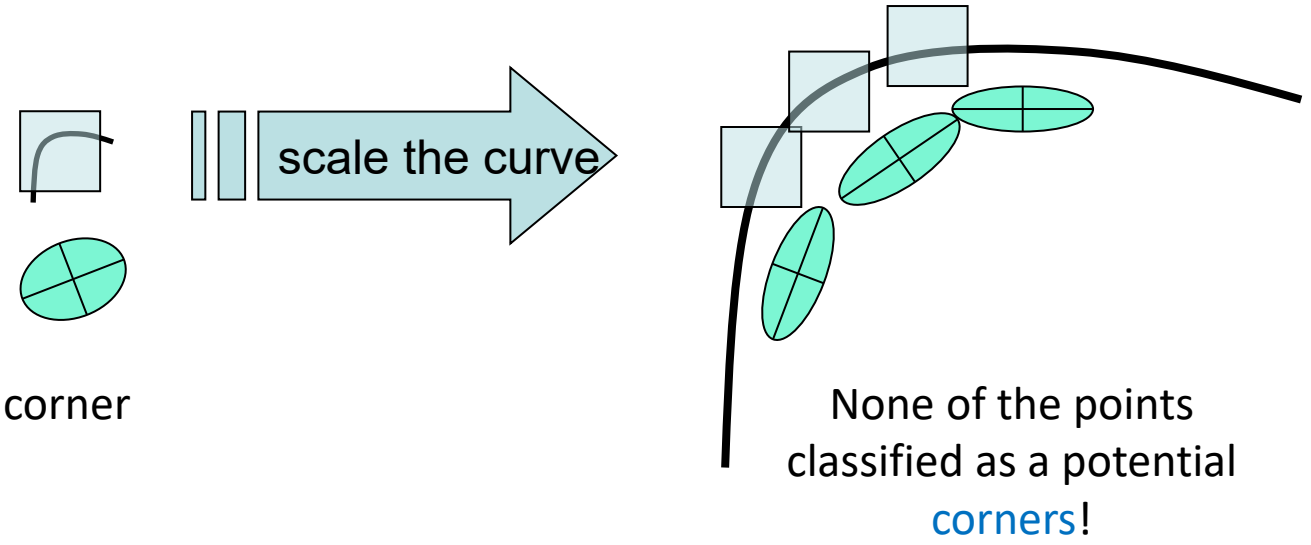


Ellipse rotates, but its shape (e.g., eigenvalues) remains unchanged!

*The corner response function is rotation invariant!*

# Harris/Hessian detector: properties

- Rotation invariance
- Is it invariant to scale change?



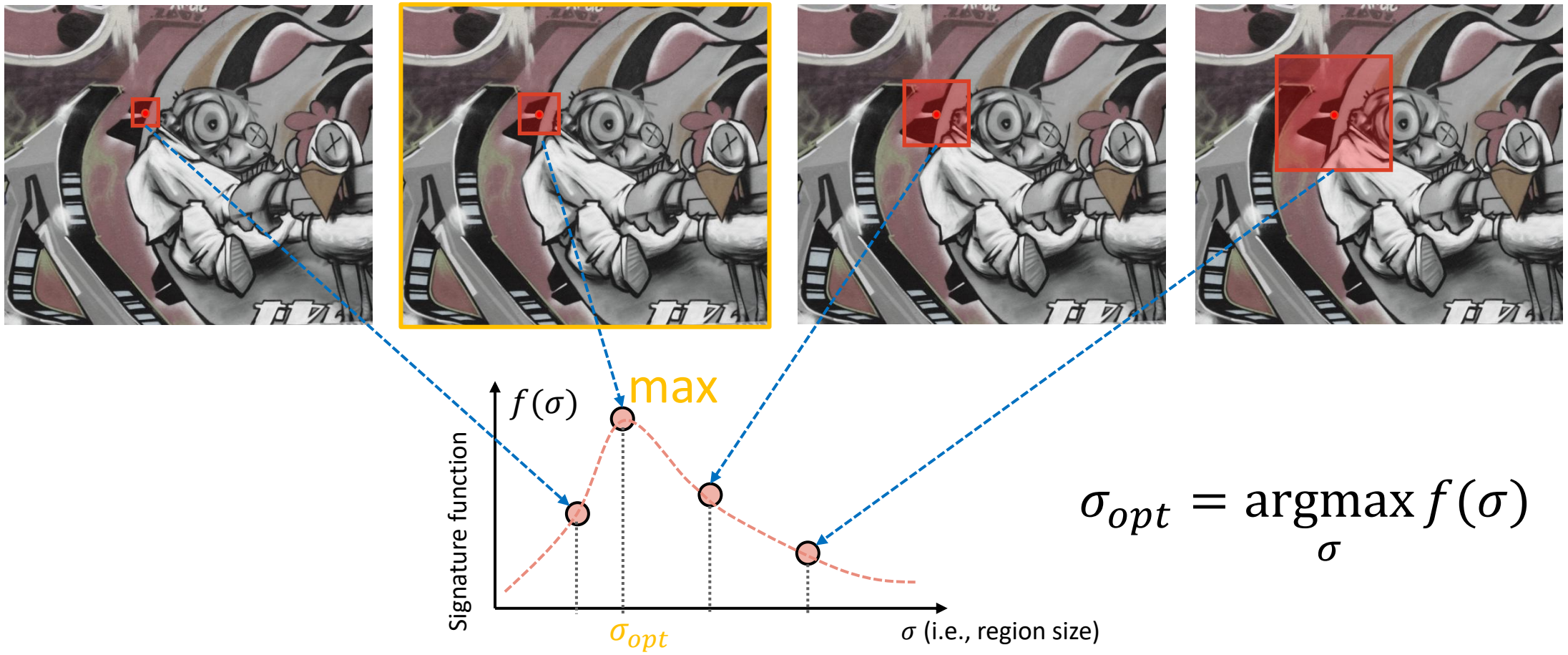

*NOT invariant to scale change!*

Machine Perception

# SCALE SELECTION

# Automatic scale selection

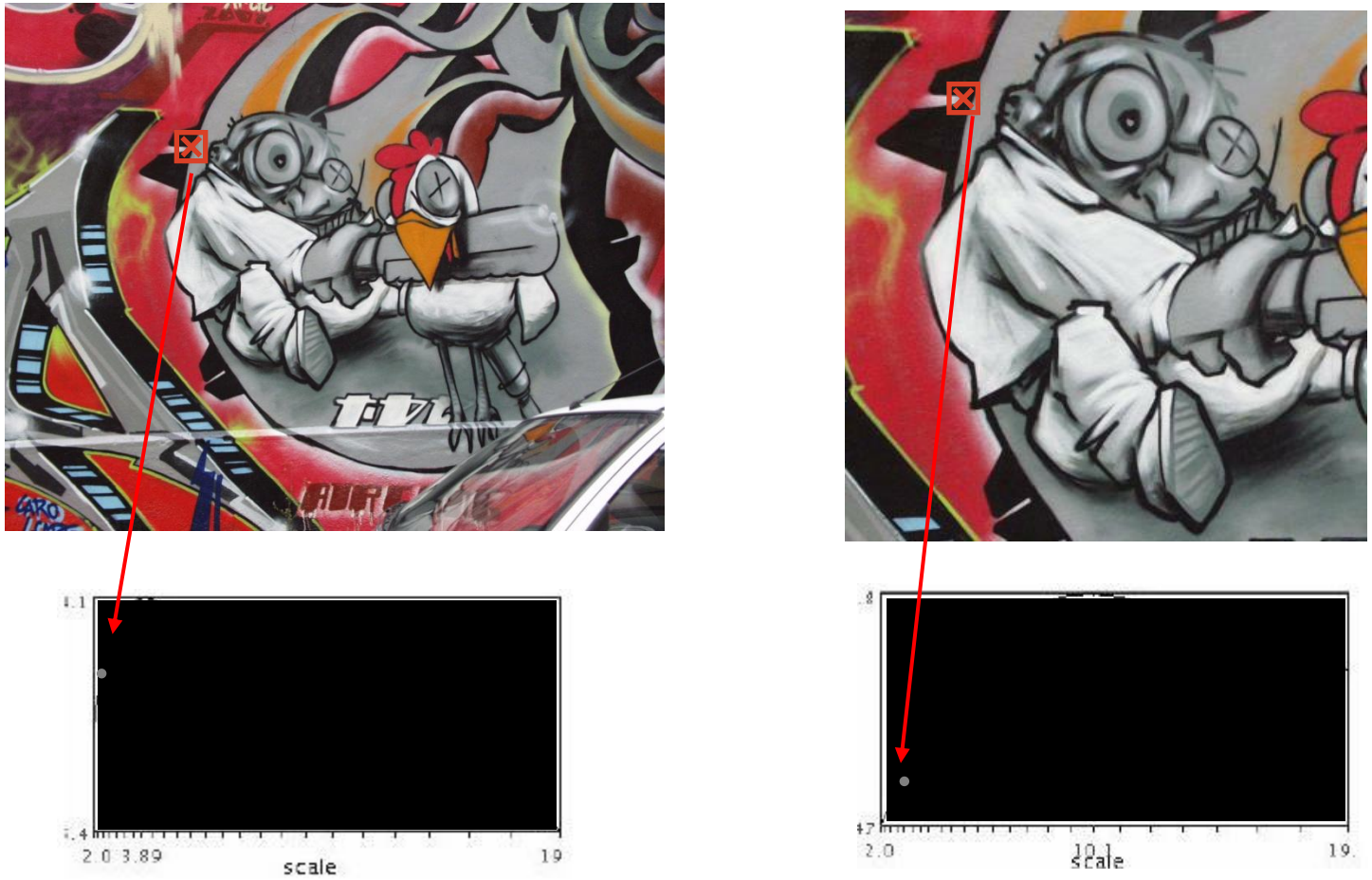
- Requirement: Select the "characteristic" scale independently for each keypoint
- Define a signature function $f(\sigma)$ that reaches maximum at the optimal scale




$$\sigma_{opt} = \underset{\sigma}{\operatorname{argmax}} f(\sigma)$$

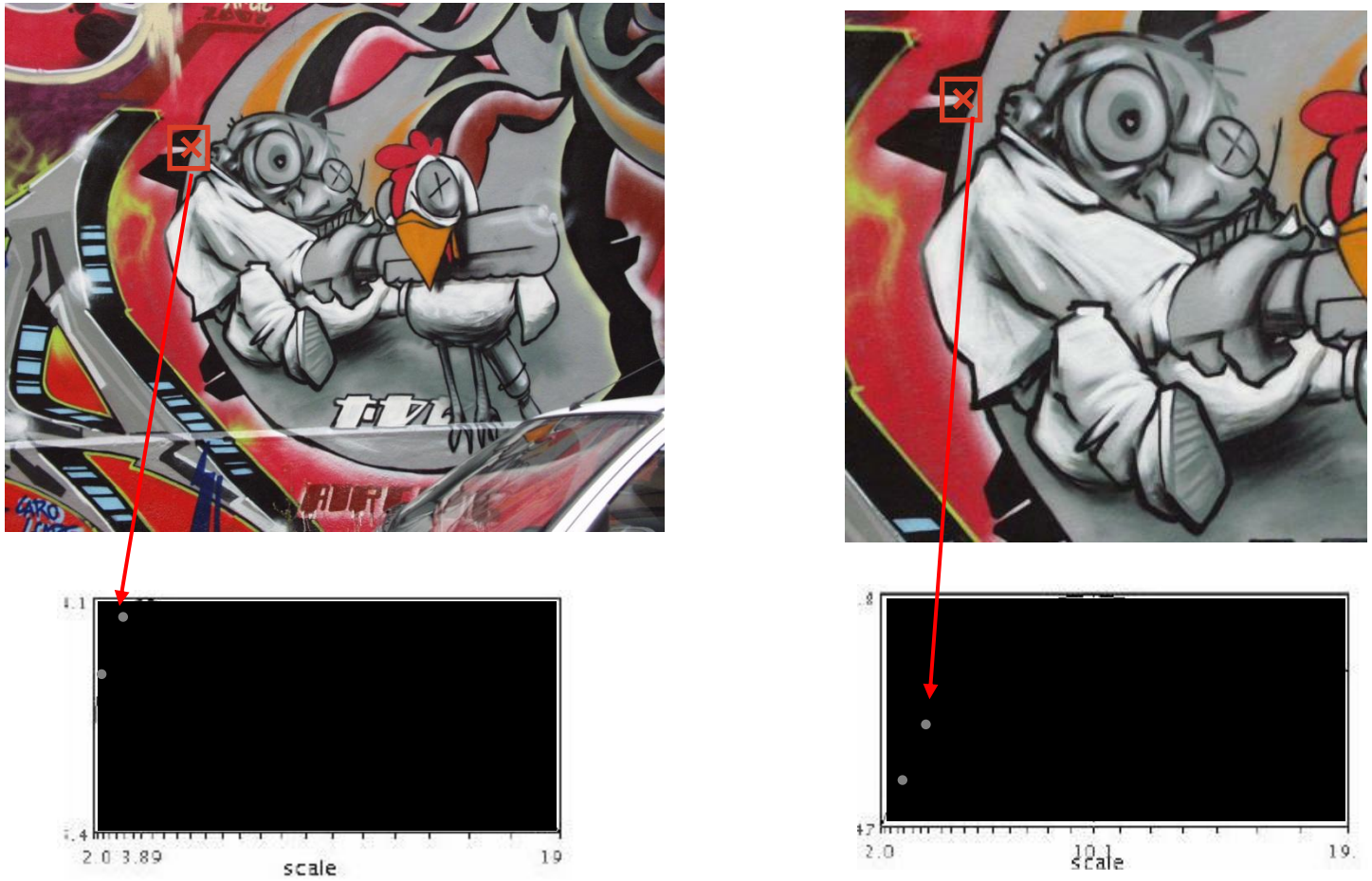
# Automatic scale selection

- Scale evaluated independently at each point and each image



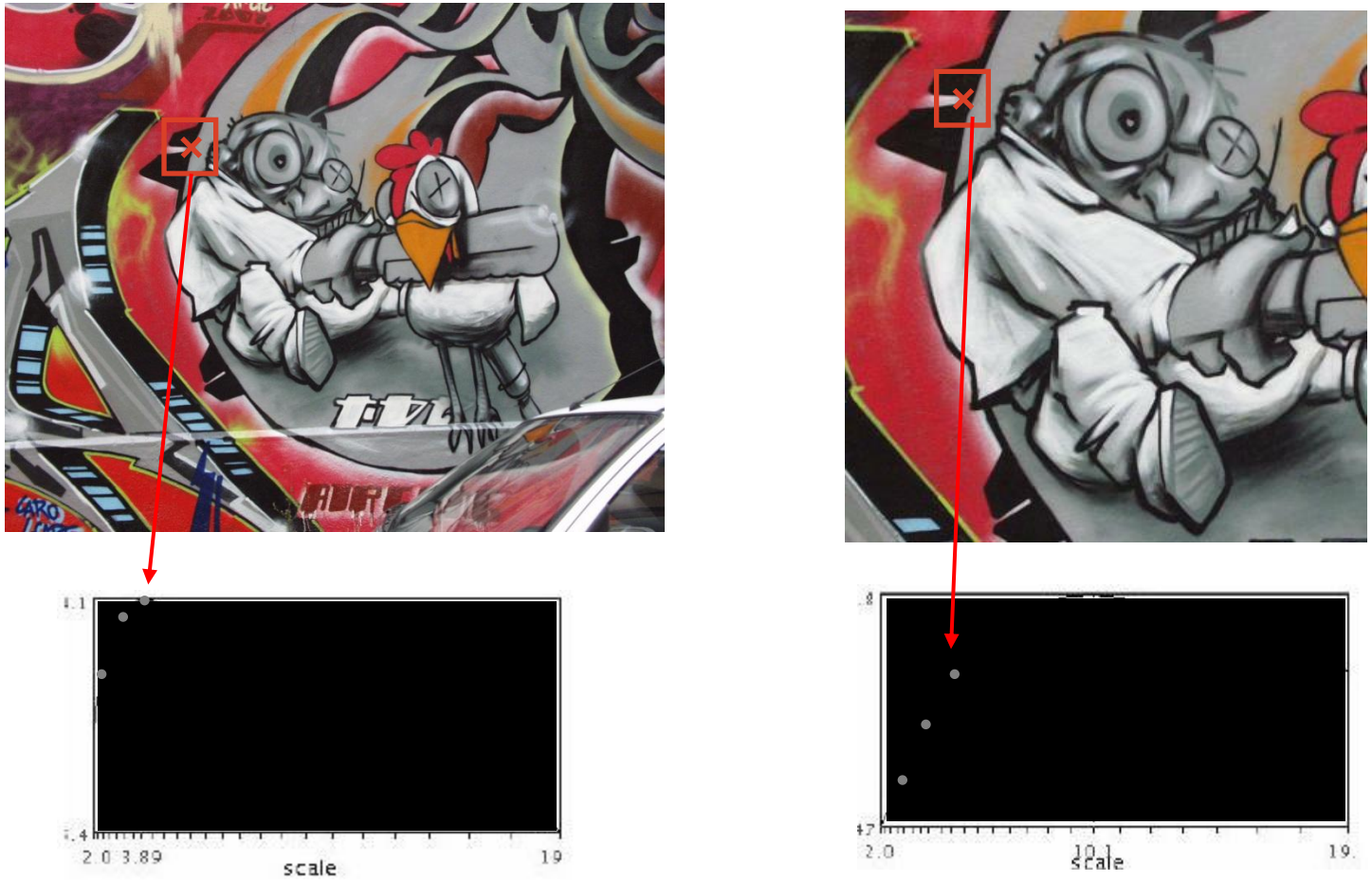
# Automatic scale selection

- Scale evaluated independently at each point and each image



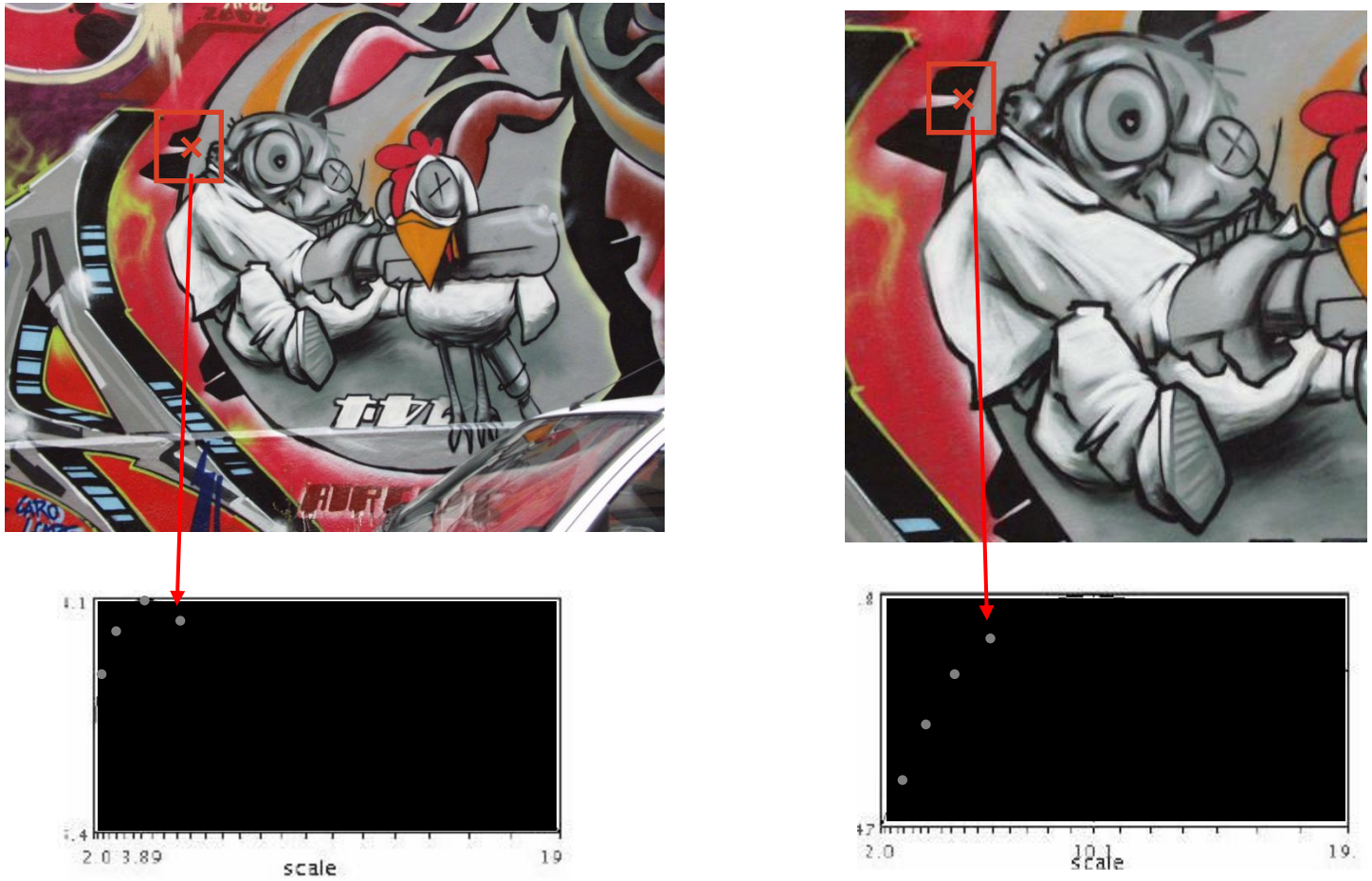
# Automatic scale selection

- Scale evaluated independently at each point and each image



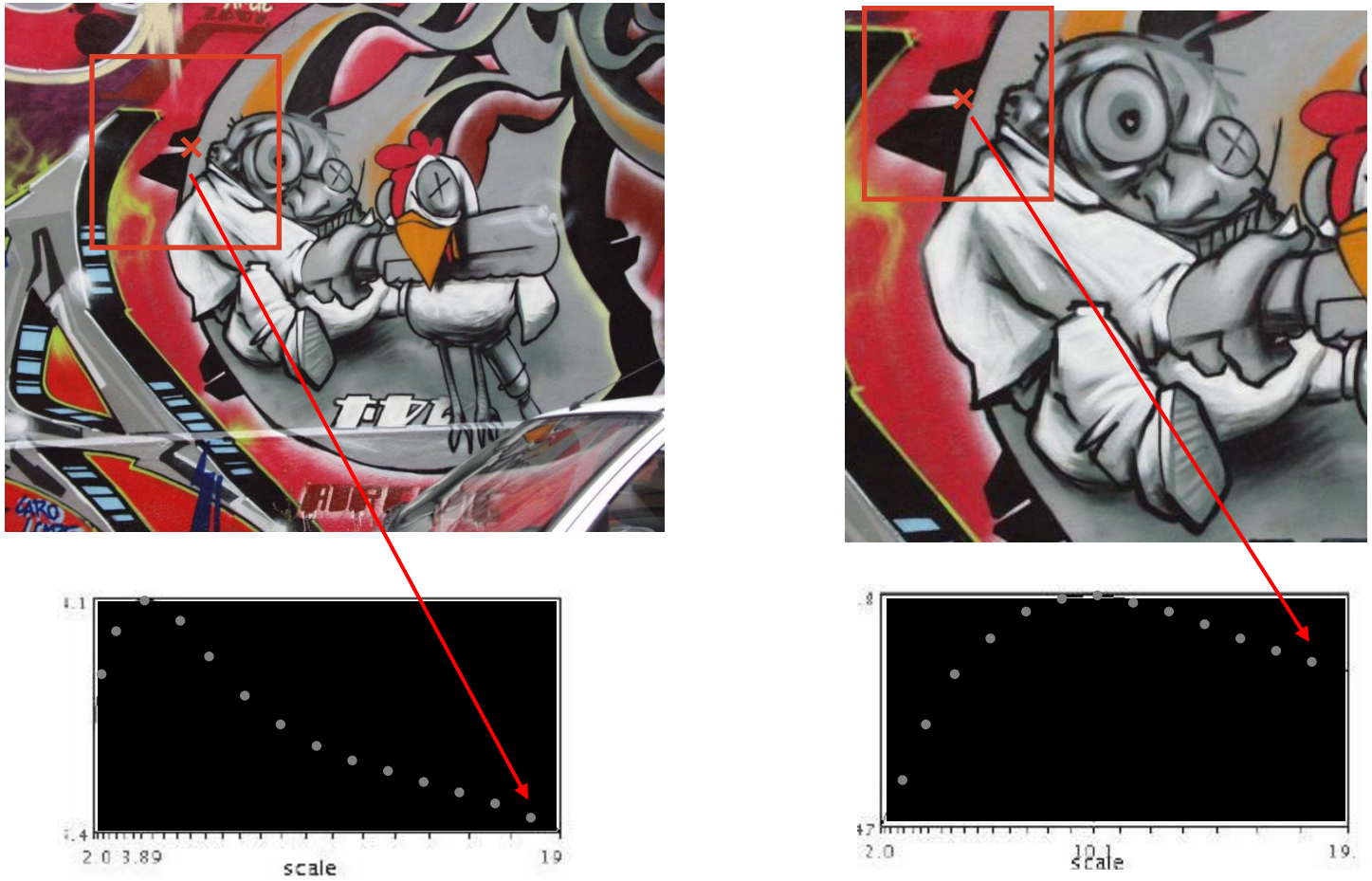
# Automatic scale selection

- Scale evaluated independently at each point and each image



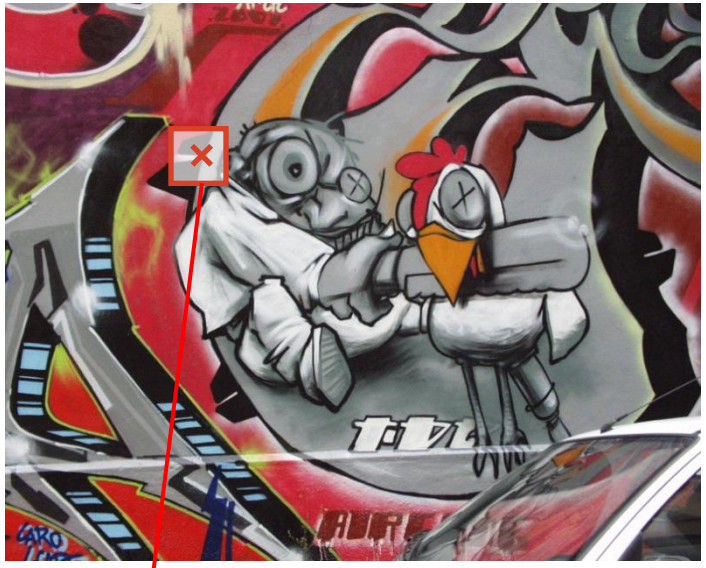
# Automatic scale selection

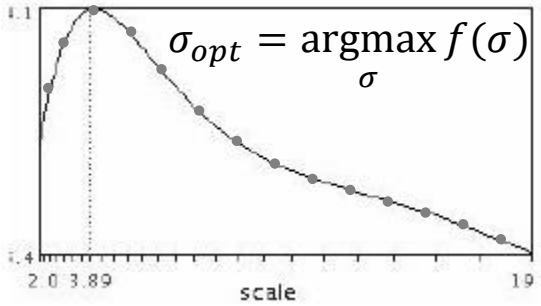
- Scale evaluated independently at each point and each image

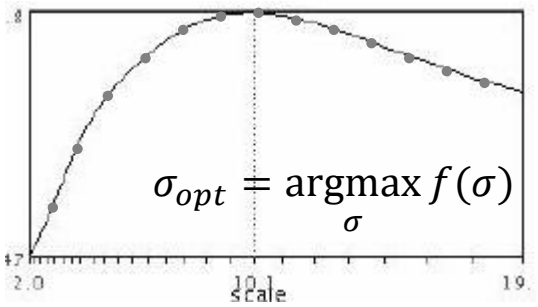


# Automatic scale selection

- Scale evaluated independently at each point and each image

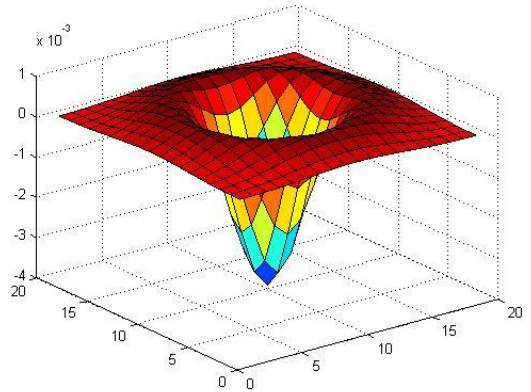




# What is a useful scale signature function?

- Natural images abundantly contain blob-like features
- Blob detection – find regions that locally look like "spots"
- Laplacian of Gaussian: $\nabla^2 G = \frac{\partial^2 G}{\partial x^2} + \frac{\partial^2 G}{\partial y^2}$
- But, for larger scale parameter (larger σ), max response of LoG is reduced – Response is *NOT SCALE INVARIANT*
- Differential operators must be normalized by $\sigma^n$, where $n$ is the derivative factor [1]: → second-order gives $\sigma^2$




- Scale-normalized Laplacian of a Gaussian (LoG*):

$$\text{LoG} = \sigma^2(\nabla^2 \text{G})$$

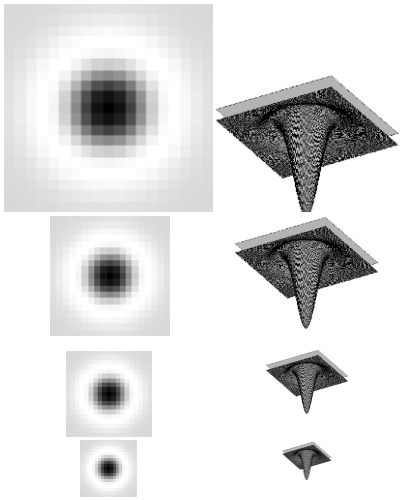
$$\nabla^2 G(x, y; \sigma) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4}\right) \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)$$

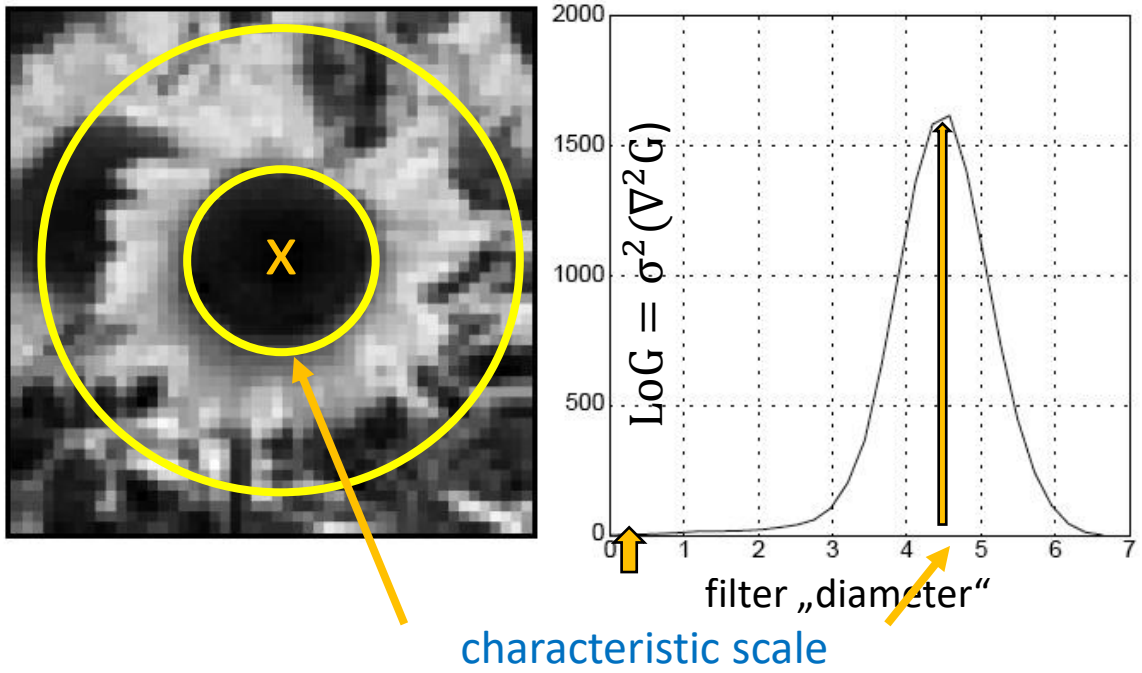
*We'll use LoG to refer to a normalized Laplacian of a Gaussian from here on

[1] T. Lindeberg "Feature detection with automatic scale selection." *International Journal of Computer Vision* 30 (2), 1998.

# Detecting characteristic scale

- The characteristic scale is the scale at which the LoG filter yields a maximum response.



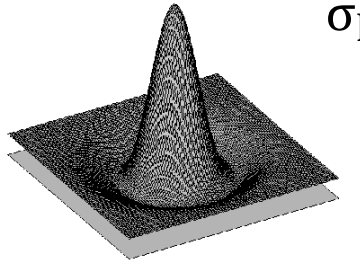



T. Lindeberg "Feature detection with automatic scale selection." *International Journal of Computer Vision* 30 (2), 1998.

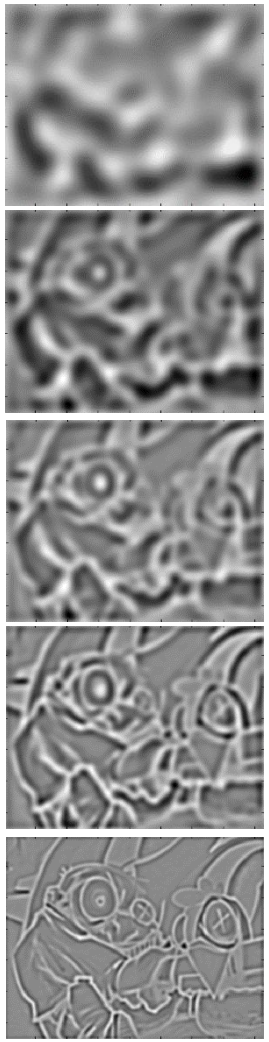
# Detecting keypoints & scales in LoG scale space

- Key-points:
  - Local maxima in scale space of the LoG filter.

$\sigma^2(\nabla^2 G)$



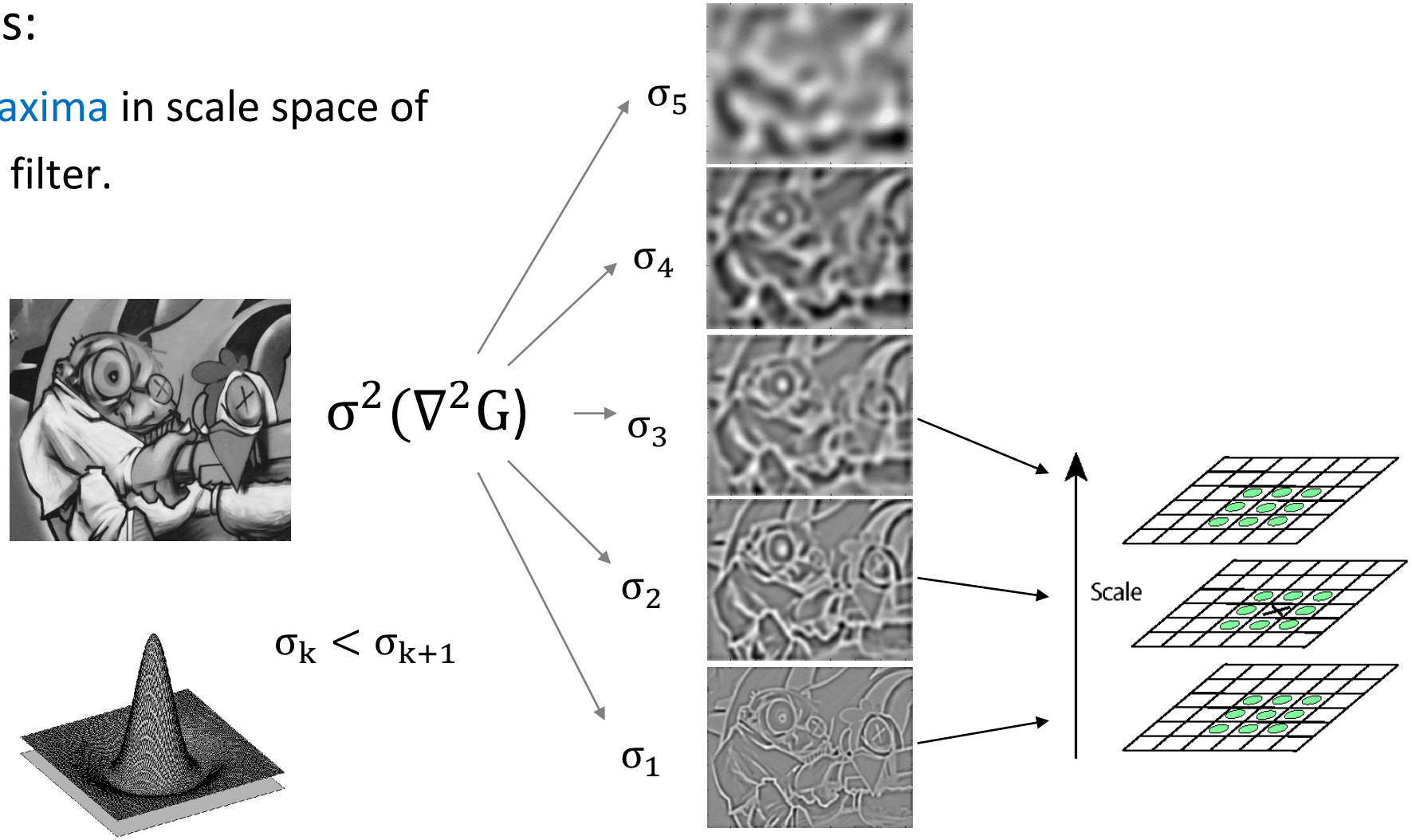
$\sigma_k < \sigma_{k+1}$




Scale-normalized laplacian pyramid

# Laplacian pyramid implementation

- Key-points:
  - Local maxima in scale space of the LoG filter.

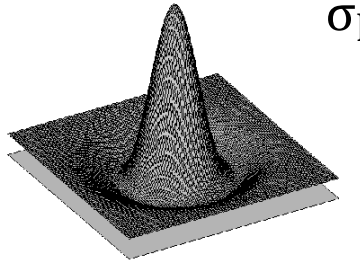



Compare LoG at each point to its $8+9 \times 2$ neighbors (same scale + upper/lower scale.)

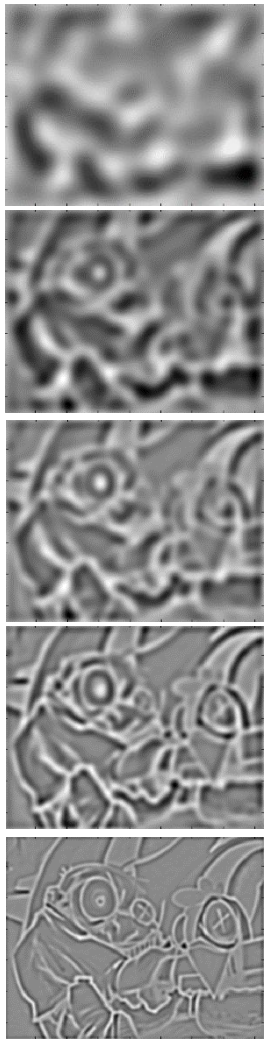
# Laplacian pyramid implementation

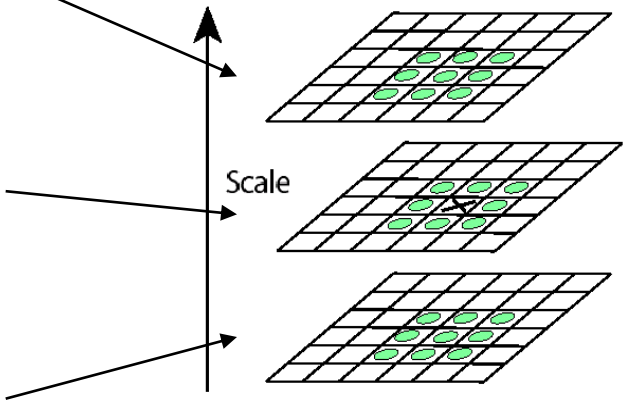
- Key-points:
  - Local maxima in scale space of the LoG filter.

$\sigma^2(\nabla^2 G)$



$\sigma_k < \sigma_{k+1}$



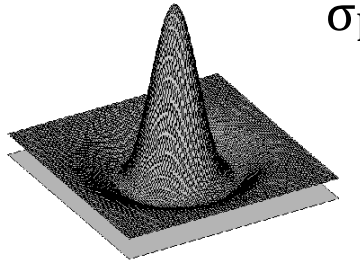




Compare LoG at each point to its 8+9 × 2 neighbors (same scale + upper/lower scale.)

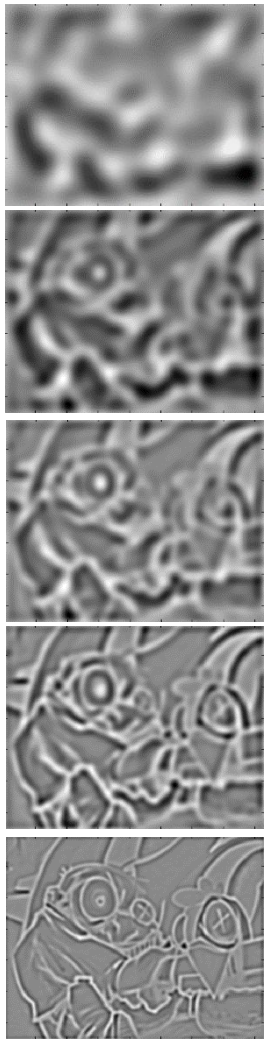
# Laplacian pyramid implementation

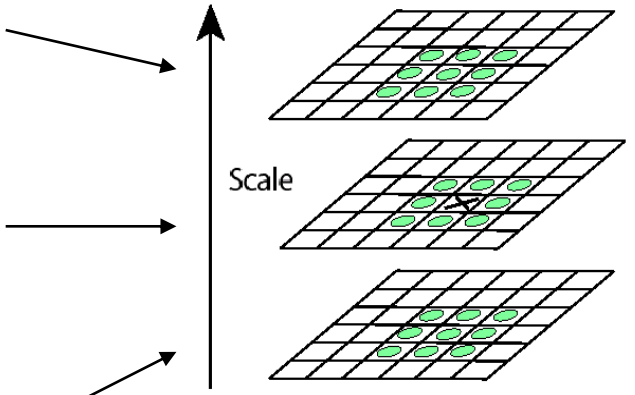
- Key-points:
  - Local maxima in scale space of the LoG filter.

$\sigma^2(\nabla^2 G)$



$\sigma_k < \sigma_{k+1}$




$\Rightarrow$ List *(x, y, σ)*

Let's look at an example…

Compare LoG at each point to its 8+9 × 2 neighbors (same scale + upper/lower scale.)

# LoG detector in action

$$\mathrm{LoG} = \sigma^2(\nabla^2 \mathrm{G}) = \sigma^2\left(\frac{\partial^2 \mathrm{G}}{\partial \mathrm{x}^2} + \frac{\partial^2 \mathrm{G}}{\partial \mathrm{y}^2}\right)$$

Input image

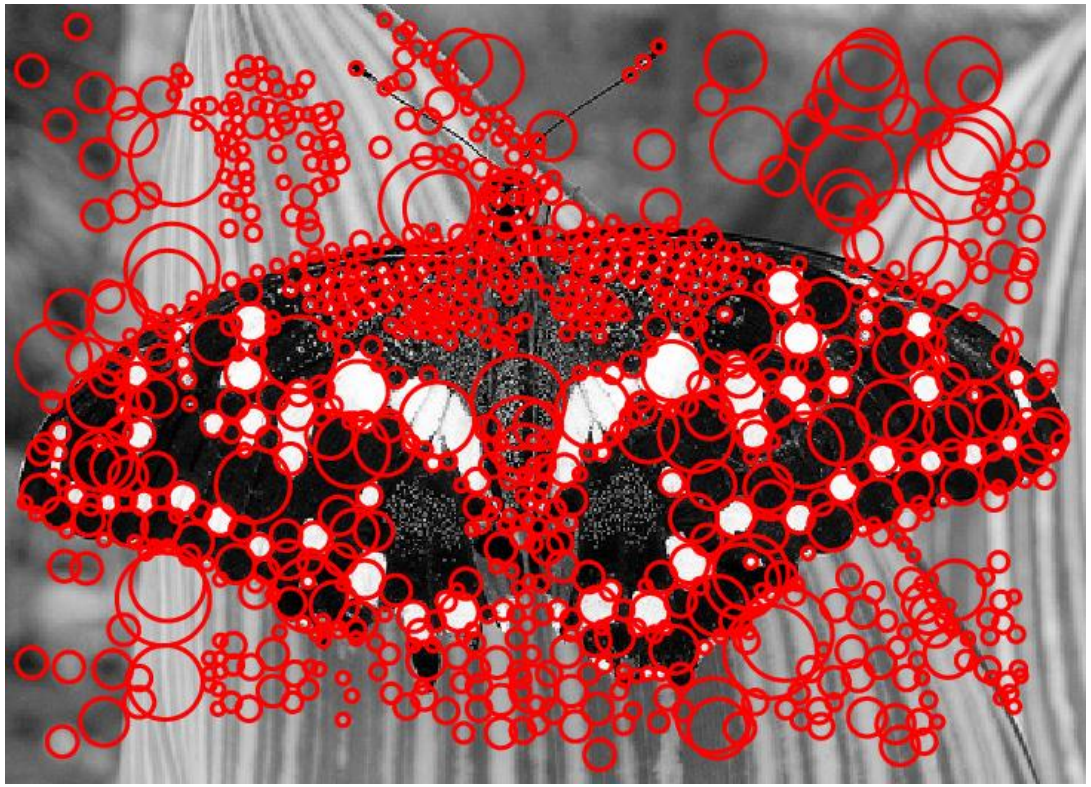
# LoG detector in action

$$\mathrm{LoG} = \sigma^2(\nabla^2 \mathrm{G}) = \sigma^2\left(\frac{\partial^2 \mathrm{G}}{\partial \mathrm{x}^2} + \frac{\partial^2 \mathrm{G}}{\partial \mathrm{y}^2}\right)$$

sigma = 11.9912

LoG filtered image (varying sigma)

# LoG detector in action



Local maxima across scales

# LoG approximation by difference of Gaussians

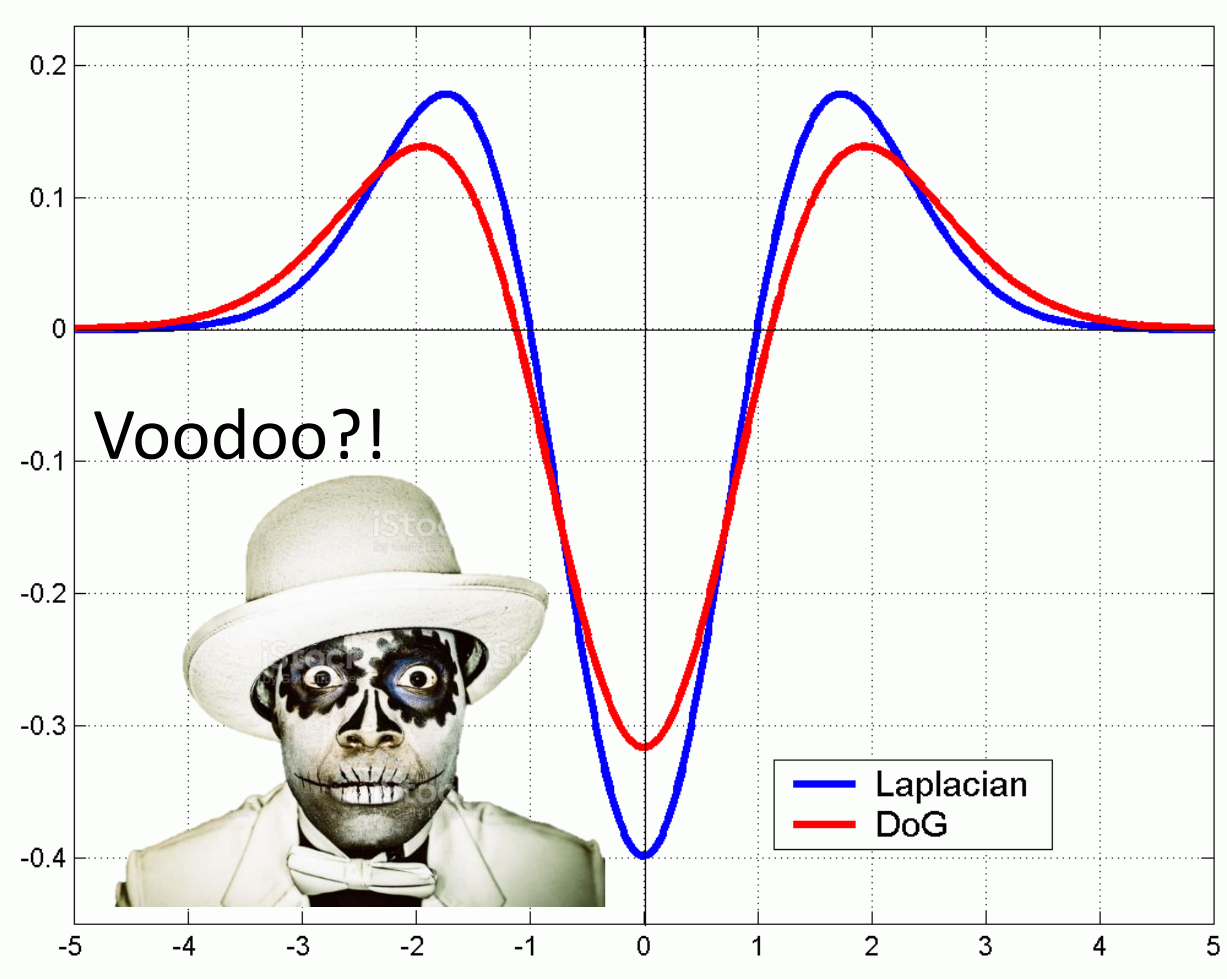
- The LoG can be well approximated with a difference of Gaussians at different values of $\sigma$.

$$L = \sigma^2 \left( G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

(normalized Laplacian of Gaussian)

$$\mathrm{DoG}(x, y, \sigma) = G(x, y, k\sigma) - G(x, y, \sigma)$$

(Difference of Gaussians)



# Not voodoo… $\mathrm{DoG}(x, y, \sigma) = G(x, y, k\sigma) - G(x, y, \sigma)$

- Let's Taylor-expand $G(x, y, k\sigma)$ around $k$, with $\mathrm{k} = 1 + \epsilon$ :

$$G(x, y, (1+\epsilon)\sigma) = G(x, y, \sigma) + \epsilon\sigma\frac{\partial G}{\partial \sigma} + O(\epsilon^2)$$

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx \epsilon\sigma\frac{\partial G}{\partial \sigma}$$

- Then, since $\frac{\partial G}{\partial \sigma} = \sigma\nabla^2 G$ , we have:

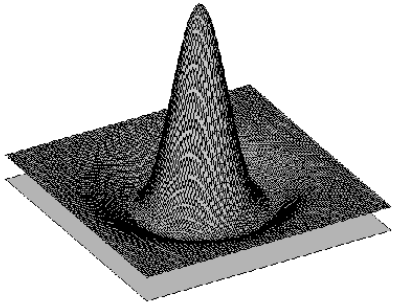
$$G(x, y, k\sigma) - G(x, y, \sigma) \approx \epsilon\sigma(\sigma\nabla^2 G) = \epsilon\sigma^2\nabla^2 G$$

- Recalling that $\epsilon = \mathrm{k} - 1$, we have: $\mathrm{DoG}(x, y, \sigma) \approx (k-1)\sigma^2\nabla^2 G(x, y, \sigma)$

$$\mathrm{DoG}(\mathrm{x}, \mathrm{y}, \sigma) \propto \sigma^2\nabla^2\mathrm{G}(\mathrm{x}, \mathrm{y}, \sigma)$$

# Difference of Gaussians (DoG)

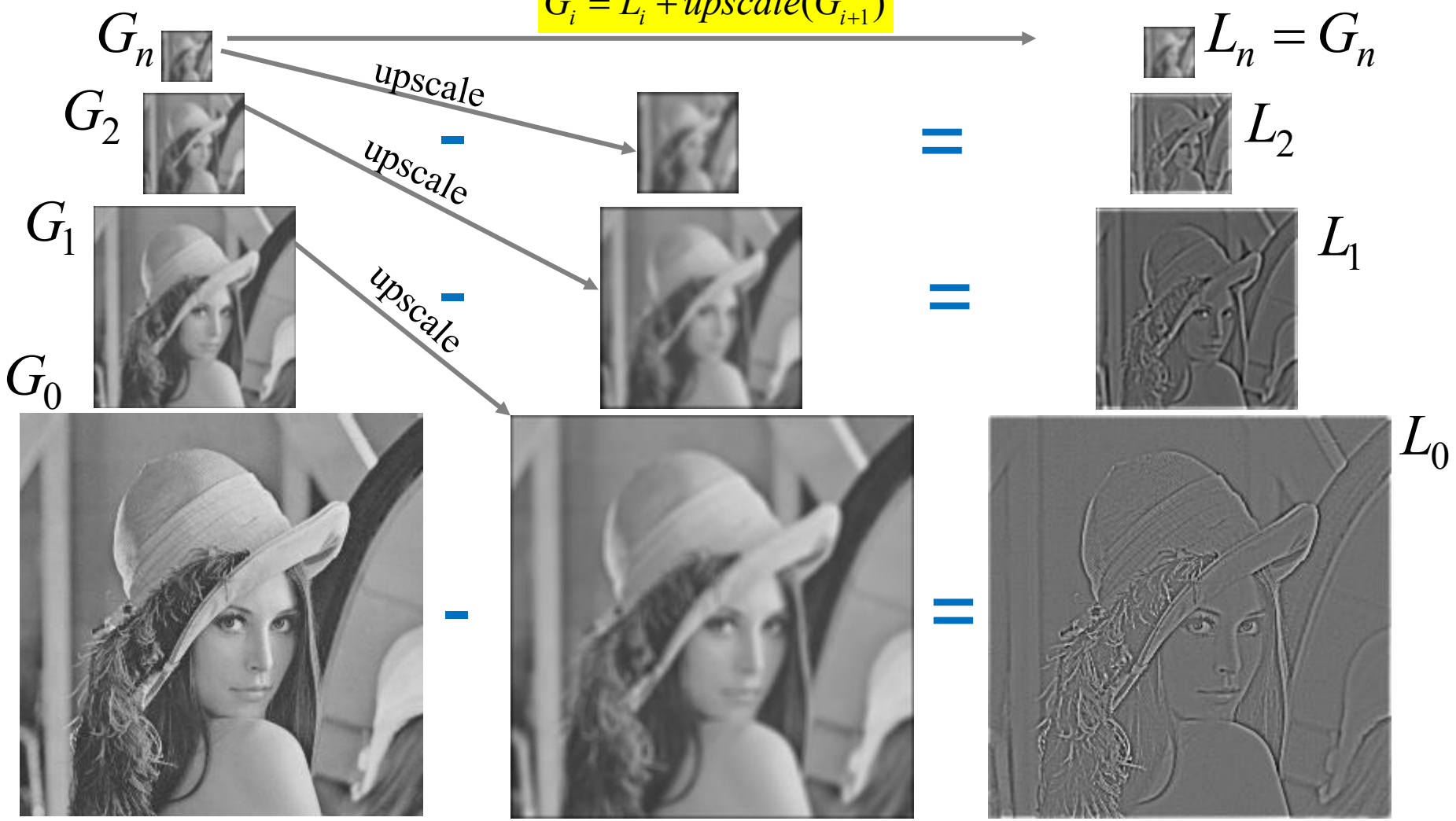
- Difference of Gaussians is an approximation of the LoG
- Advantages
  - Results of Gaussian filtering already calculated during calculation of image resizing (Gaussian Pyramid! – see next slide).



# LoG pyramid approximation

$$L_i = G_i - upscale(G_{i+1})$$

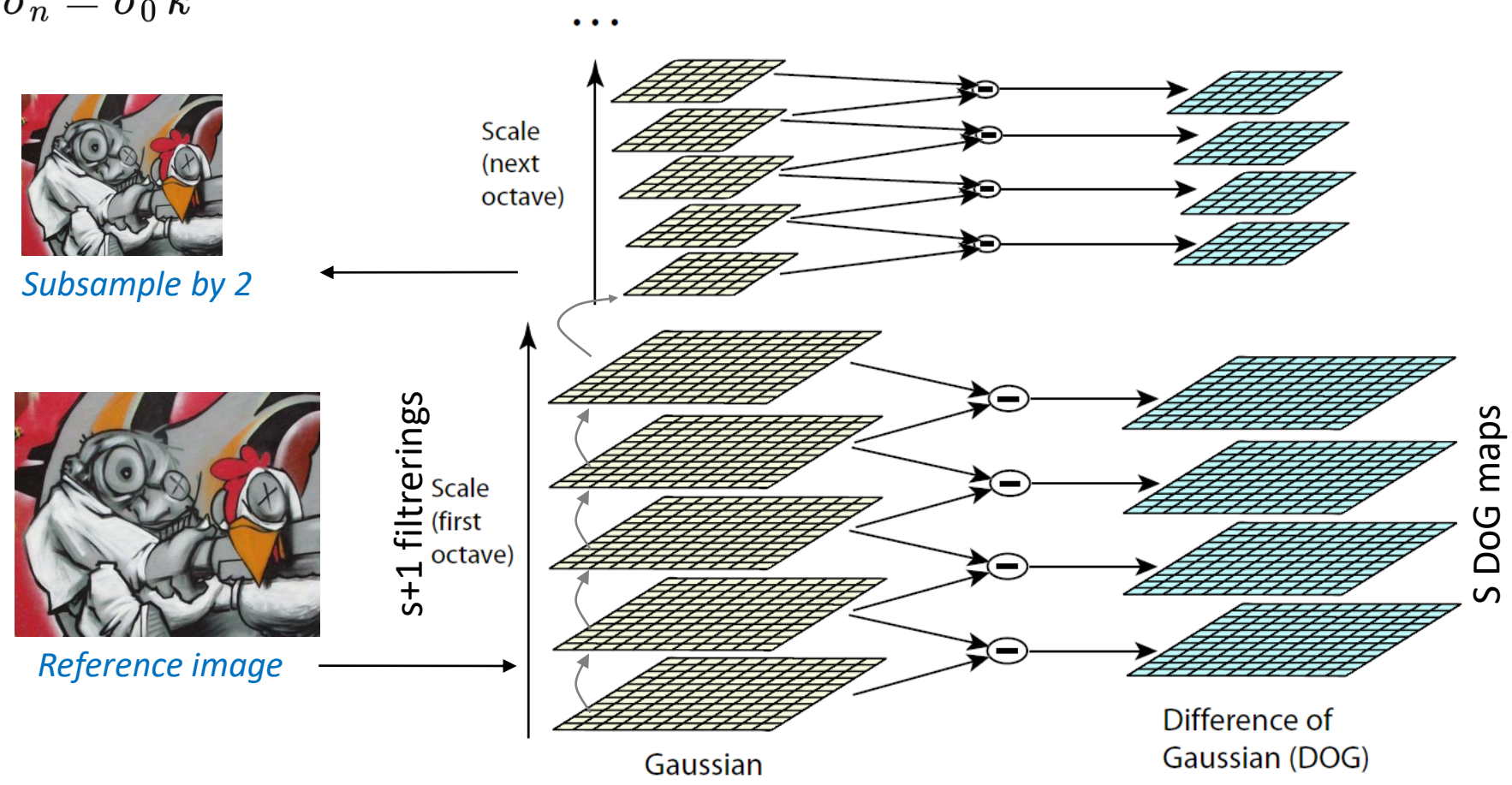
$$G_i = L_i + upscale(G_{i+1})$$



# DoG pyramid– Efficient calculation

- Calculated from a Gaussian pyramid (sequential octaves equivalent to filtering with $\sigma_{next} = 2\sigma_{prev}$ )
- In each octave, apply filtering s times, with $k^s = 2$ , so that $\sigma_{next} = 2\sigma_{prev}$ (e.g., s=4 in example below)

$$\sigma_n = \sigma_0 \, k^n$$

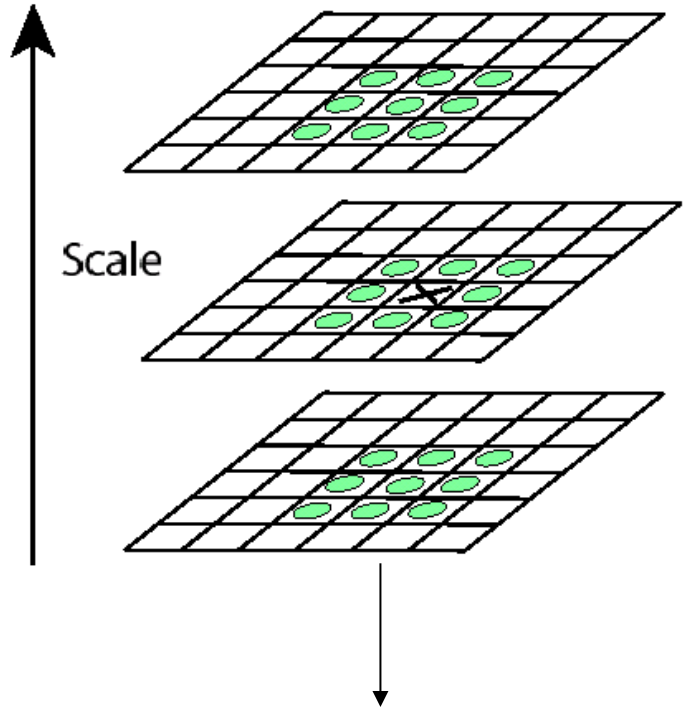



David G. Lowe, IJCV 2004

# Key-point localization using DoG

- Find local maxima of DoG in the scale-space.
  - Check 8+2*9=26 neighbors
- Remove the low contrast points

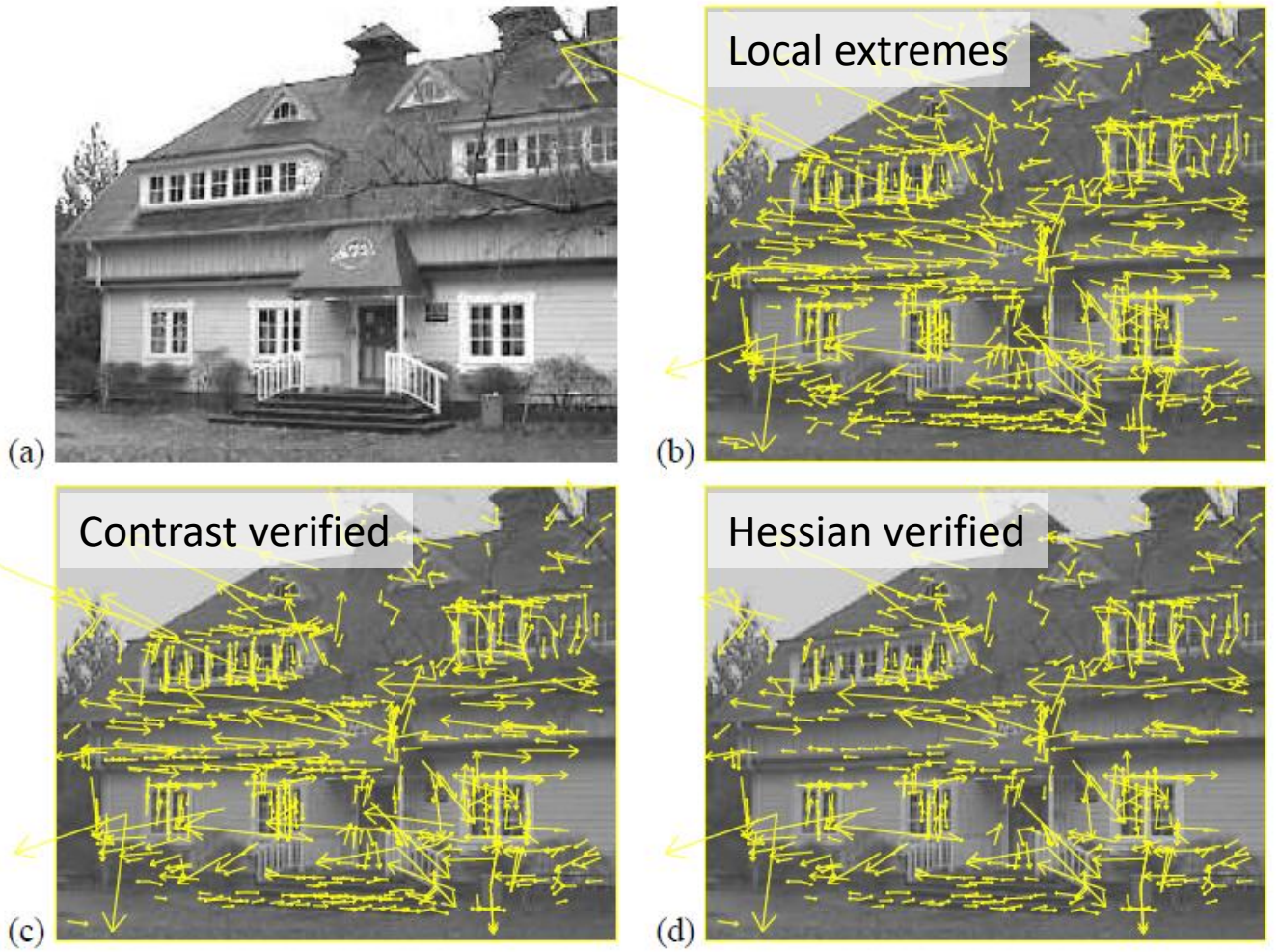
  (*threshold dependent*)
  - If local change in response is small compared to neighbors.
- Remove points detected at the edges
  - Test using the Hessian matrix.




Key-point candidates:
List of triplets (x,y,σ)

Fit a quadratic function to each key-point and its neighbors to improve localization of the maxima (x,y ,σ).

# Results: Lowe's DoG-based detector




(a) 233x189 image

(b) 832 extremes in DoG

(c) 729 remain after contrast verification

(d) 536 remain after verification of the Hessian matrices.

David G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision,* 2004

# Results: Lowe's DoG-based detector

# Summary: scale-invariant key-point detection

- Input: Image of some scene taken at unknown scale.
- Goal: Find key-points and their scales *independently* in each image.
- Solution:
  Find local *maxima* of specialized functions in scale-space and image coordinates.
- Two strategies
  - (normalized) Laplacian of Gaussian (LoG)
  - Difference of Gaussians (DoG) as an efficient approximation

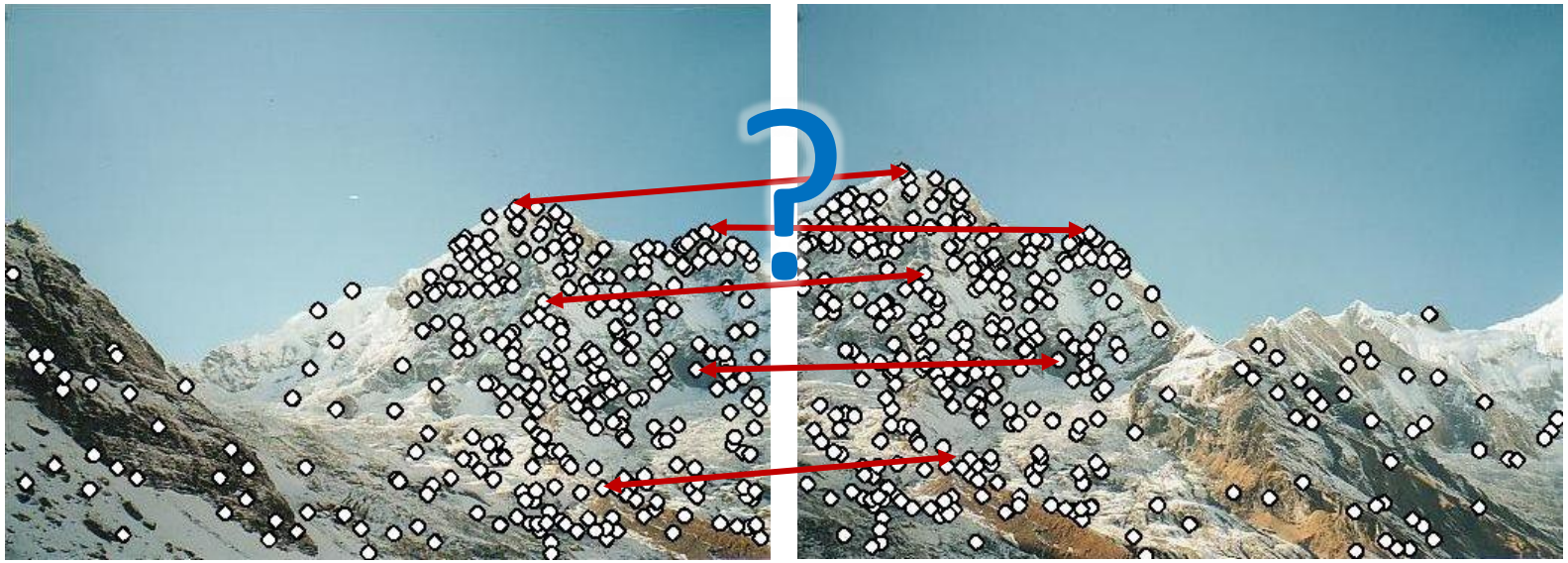
Machine Perception

# LOCAL DESCRIPTORS

# Local descriptors

- Now we know how to detect the key-points
- Next question:

How to describe them?



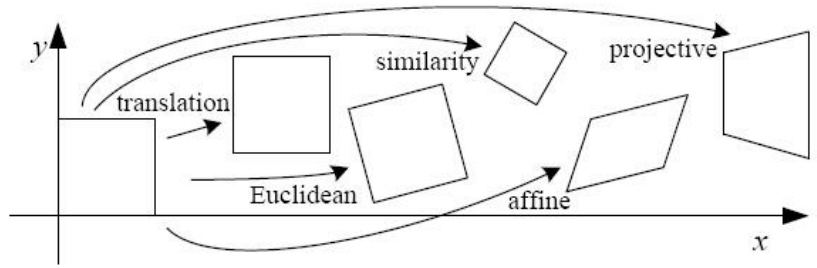

Key-point descriptors should be:

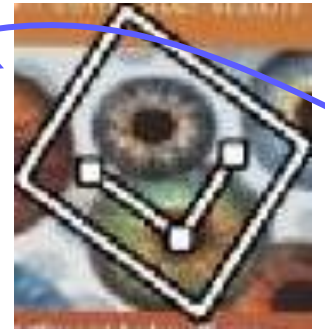
1. Distinctive (be different for keypoints on different structures)
2. Invariant to ambiental changes

# Invariance of descriptor

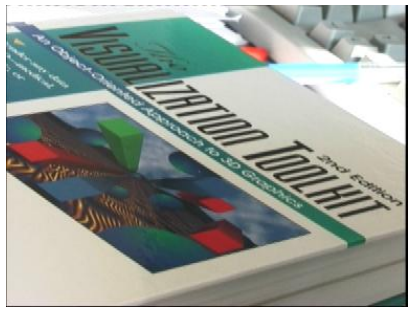
- Geometric transformations

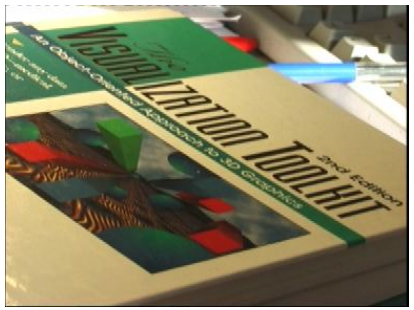


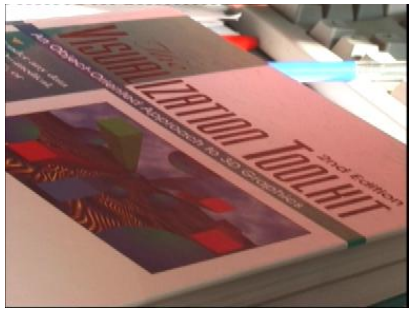



- Photometric transformations
  - Often modeled by intensity scaling and translation



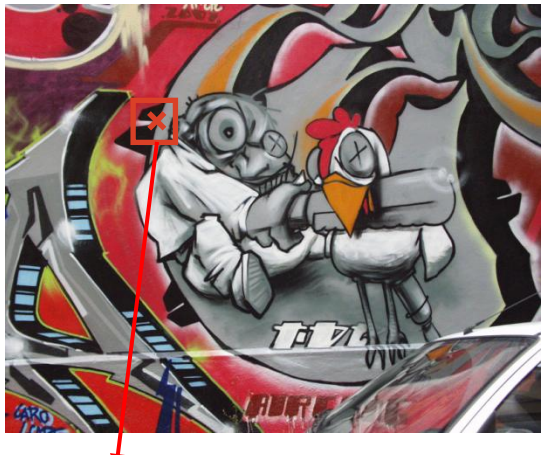



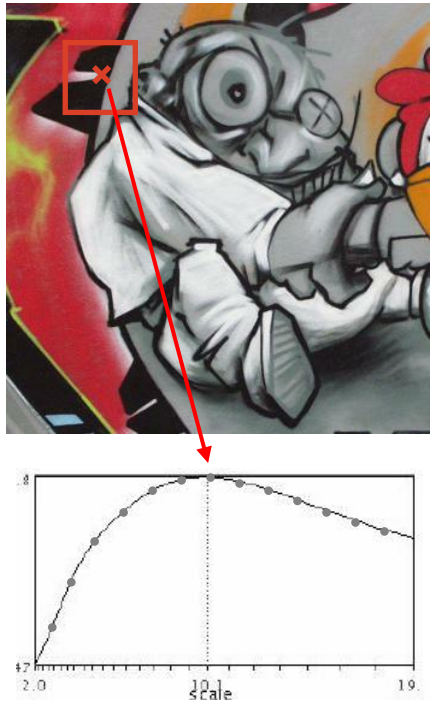
# Normalizing region scale (already covered)

- For comparing regions, normalize: Rescale to a predefined size

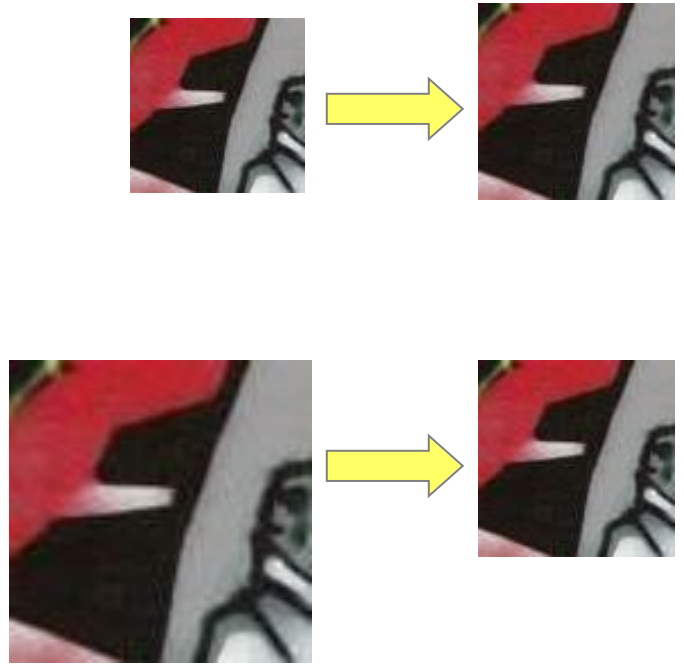
Important: the region location and size (scale) is determined independently in each image for each key-point!




$f(I_{i_1 \ldots i_m}(x,\sigma))$

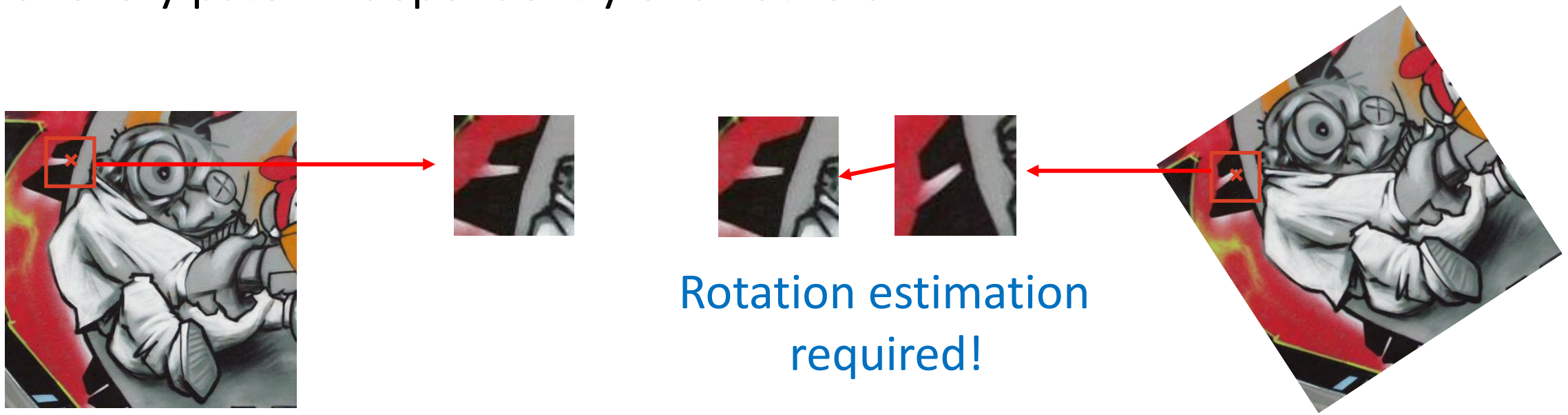



$f(I_{i_1 \ldots i_m}(x',\sigma'))$



# Normalizing region rotation

- Need to automatically determine the inherent orientation of every patch independently of all others!

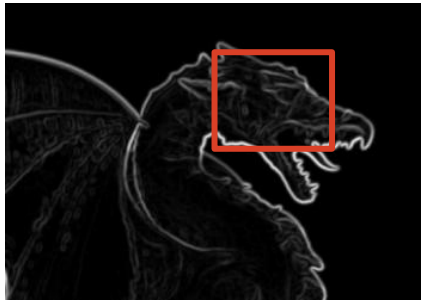


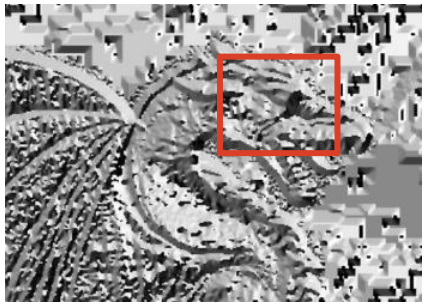
Rotation estimation required!

Per-pixel comparison
will not produce a good match!

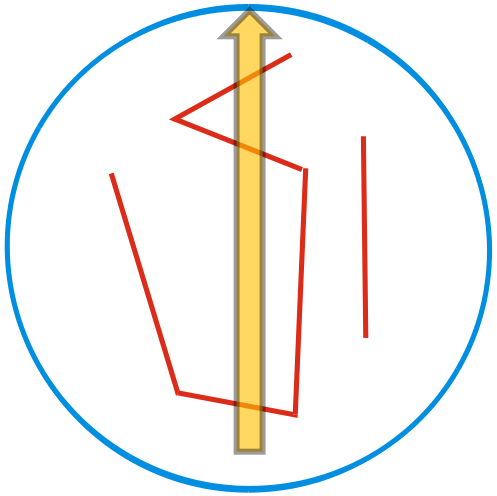
# Normalizing region rotation

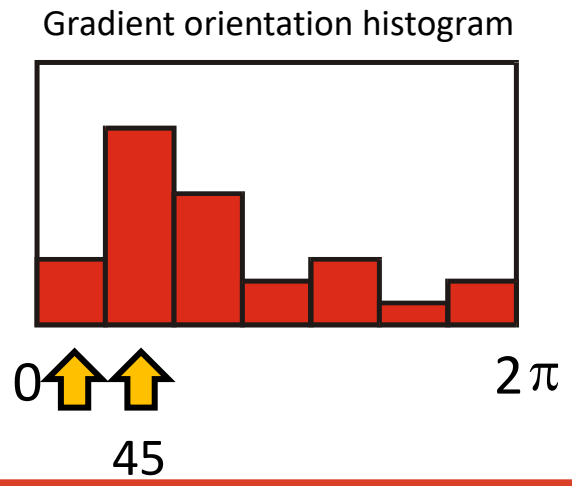
- Calculate gradients in the image patch and look at gradient directions/ pixel orientations
- Calculate the histogram of orientations
  - 36 bins by angle, each point contributes proportionally to its *gradient magnitude* and distance from the center.
- Determine the dominant orientation from histogram
- Normalize: rotate the image into a rectified orientation





The considered reigion:

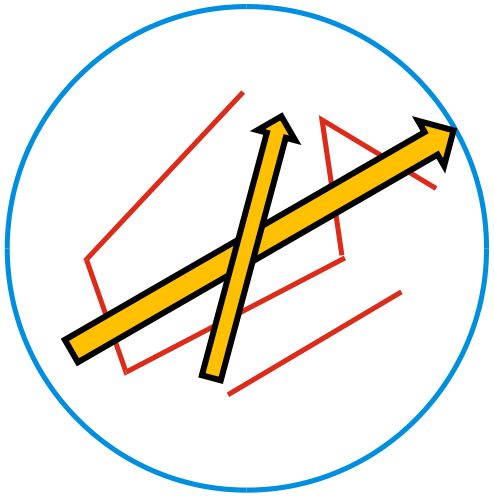




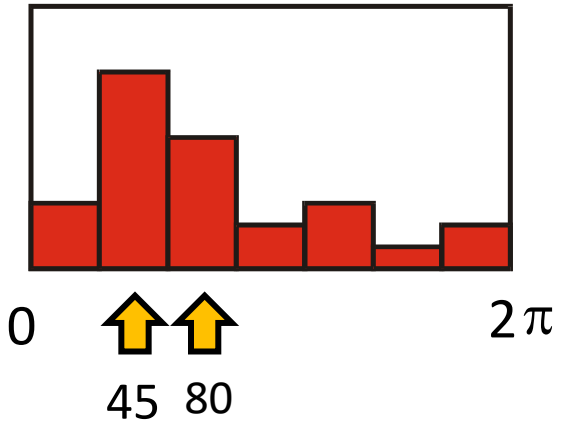
# Normalizing region rotation

- Several directions may hold non-negligible strength.
- Find all orientations in histogram, whose amplitude is, e.g., 80% of the strongest bin.
- Output each detected oriented region as a separate keypoint.

The considered reigion:



Gradient orientation histogram



# Normalizing affine transformation: Affine adaptation

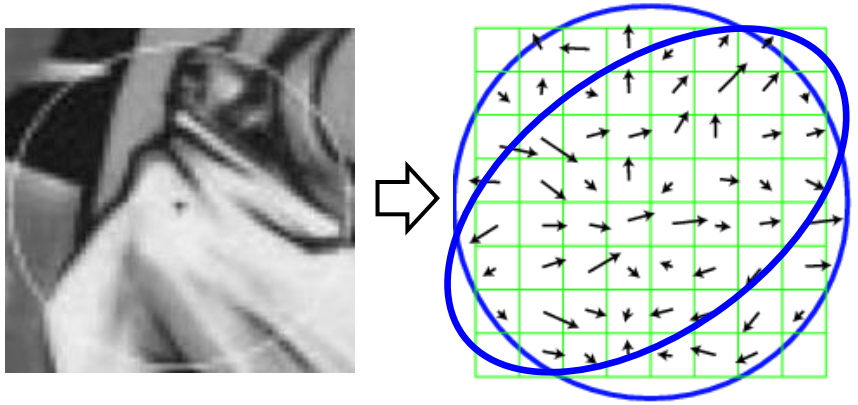


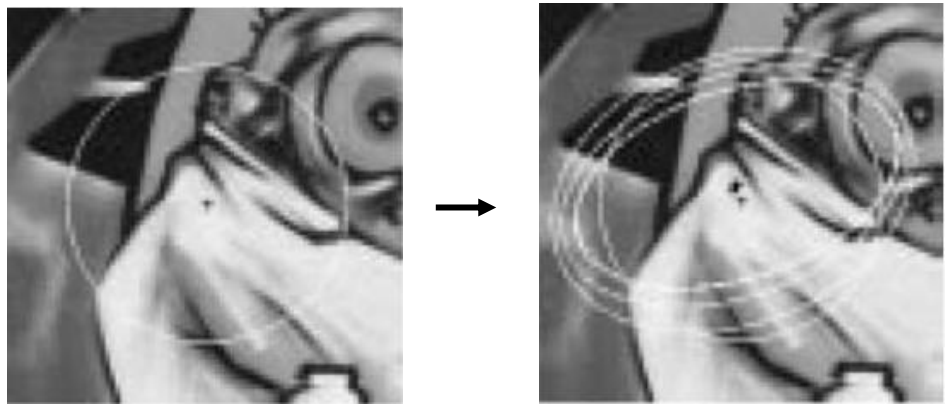
- We have addressed invariance to
  - Translation
  - Scale
  - Rotation
- But that's not enough for very large changes in viewpoint
  - We require an affine adaptation!

# Affine adaptation

Iterative approach:

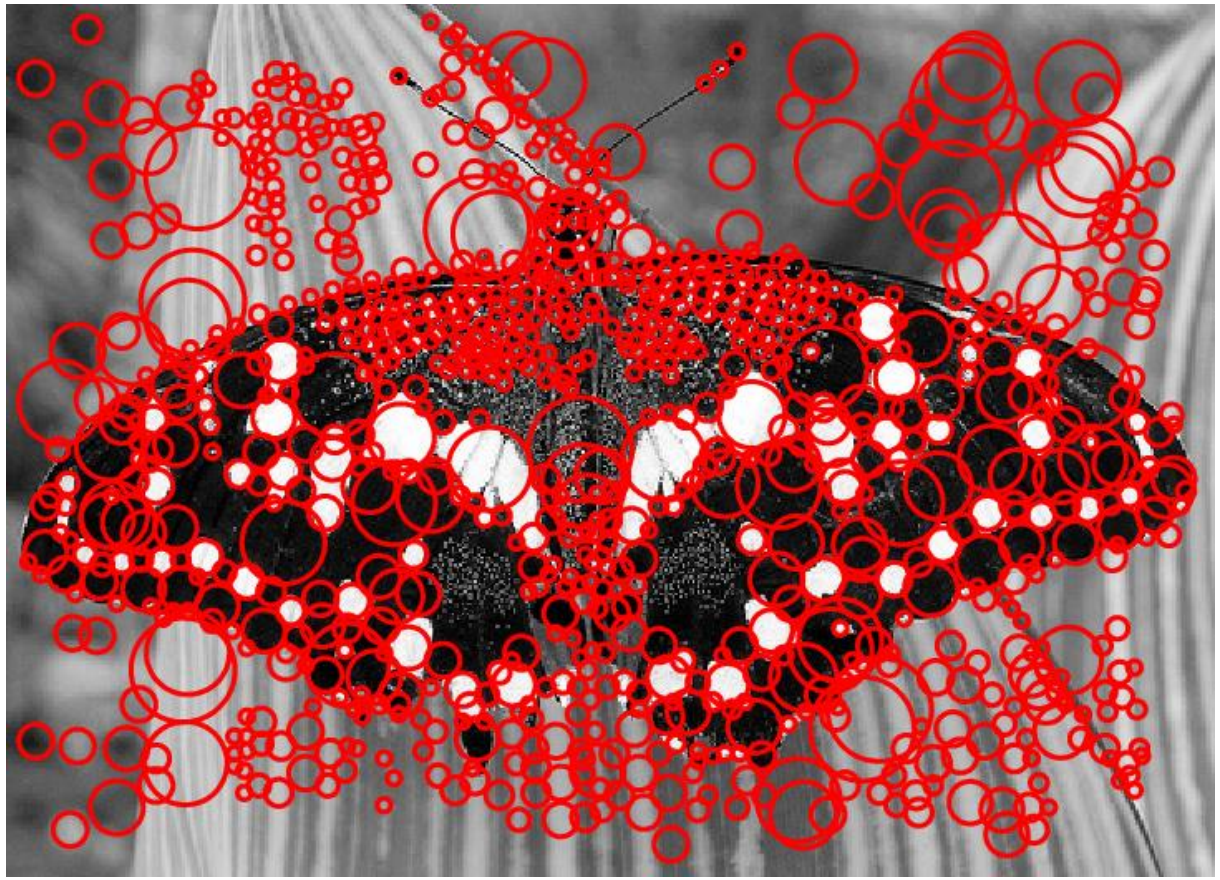
- In circular window calculate a gradient covariance matrix (similar to Harris)
- Estimate an ellipse from the covariance matrix
- Using the new window calculate the new covariance matrix and iterate.





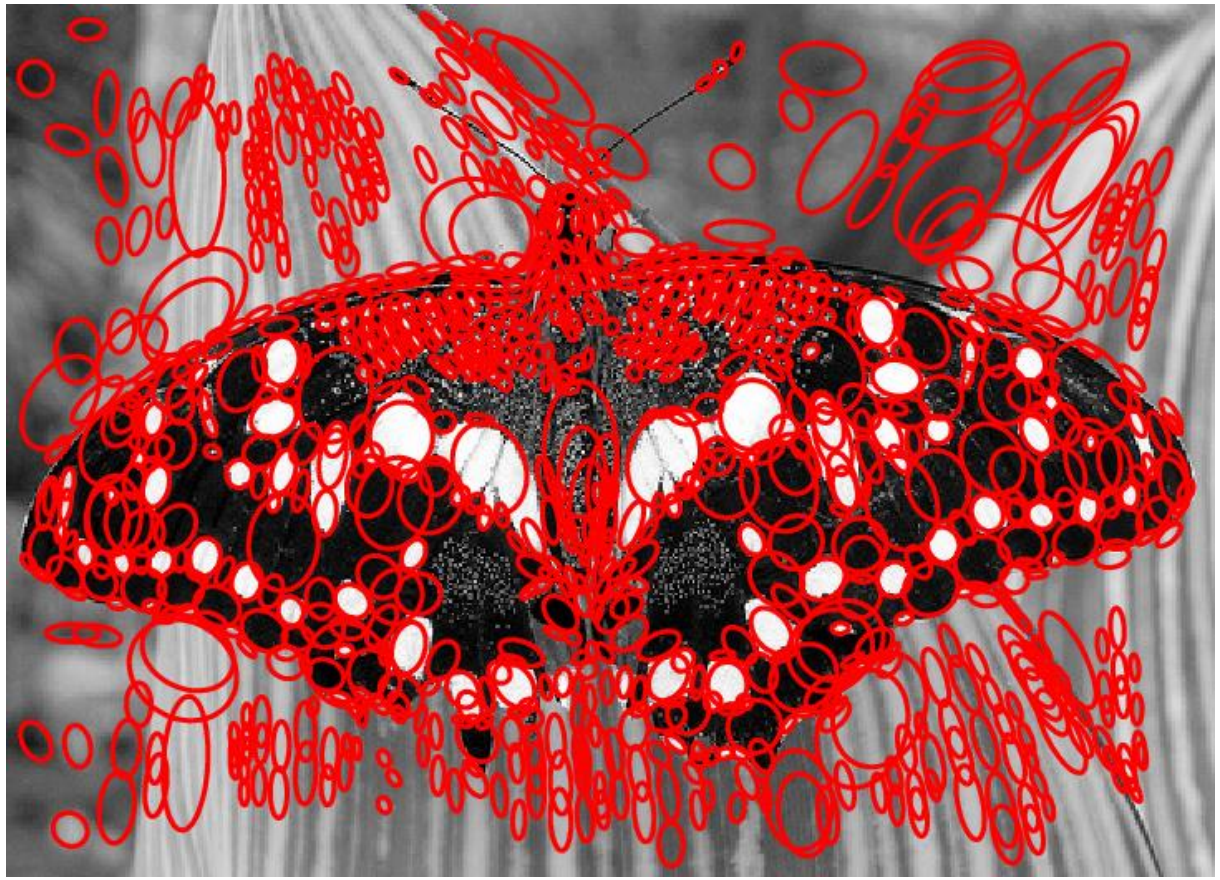
K. Mikolajczyk and C. Schmid, Scale and affine invariant interest point detectors, IJCV 60(1):63-86, 2004.

# Affine adaptation: Example



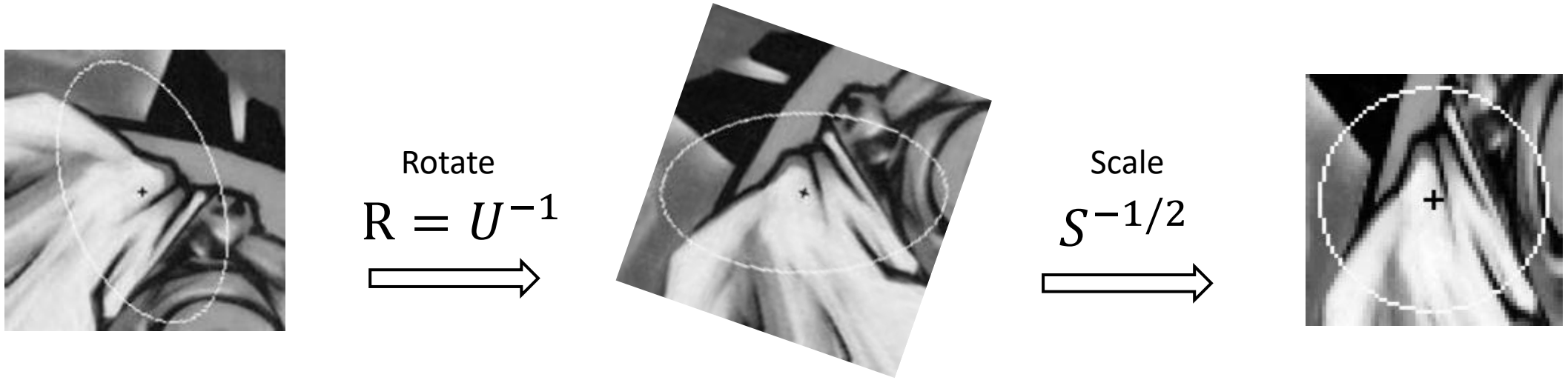
Detect blobs accross scales

# Affine adaptation: Example



Affine-adapted regions

# Affine patch normalization



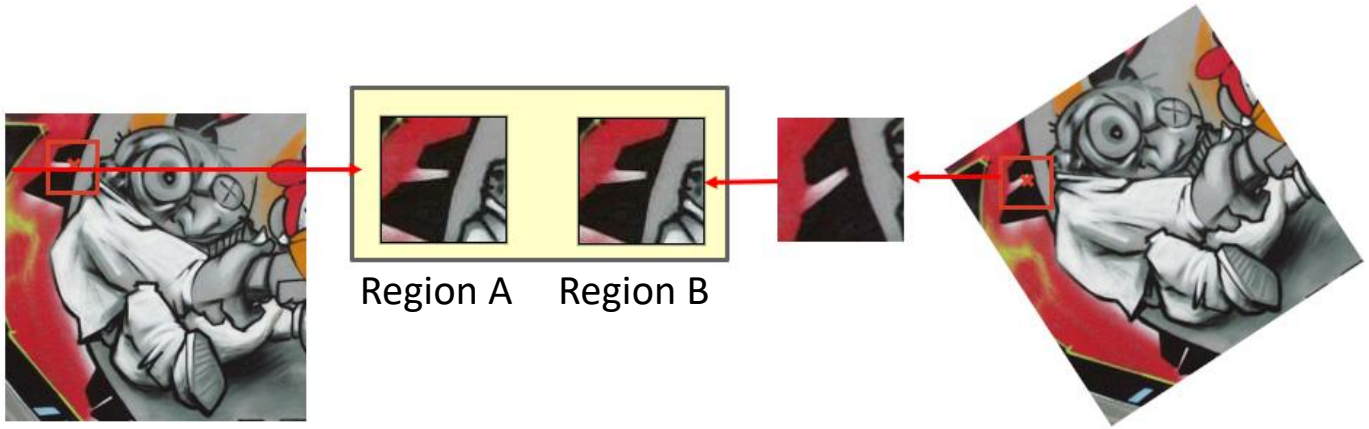

- Transform the patch such that the ellipse becomes as circle.
- Rotate the region such that the ellipse rotates into a horizontal position
- Scale the region such that the ellipse transforms into a circle

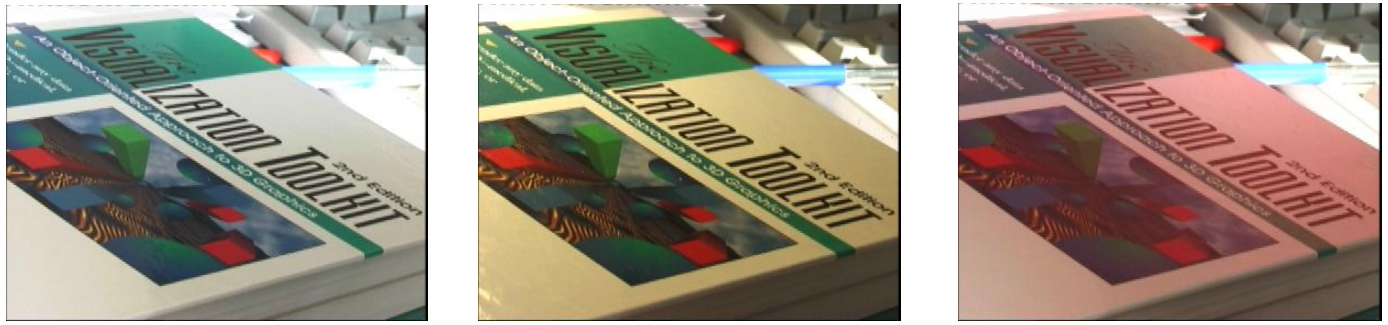
Note: Rotation + Scaling computed from the (ellipse, $\Sigma$) eigen vectors and eigen values

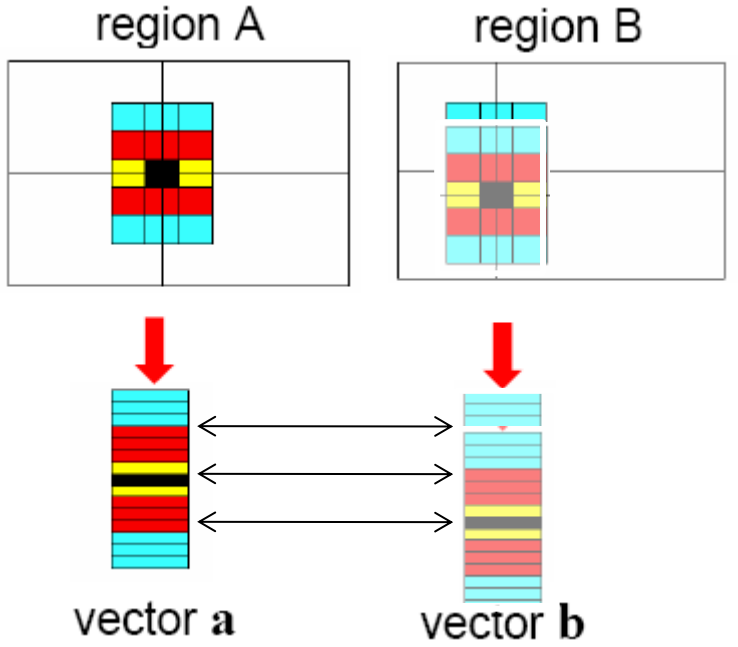
$$\Sigma = USU^T$$

# Local descriptors / Region descriptors

- The simplest descriptor: a vector of region intensities.
- Small shifts due to noise may cause a large change in the descriptor.
- Sensitive to photometric changes.



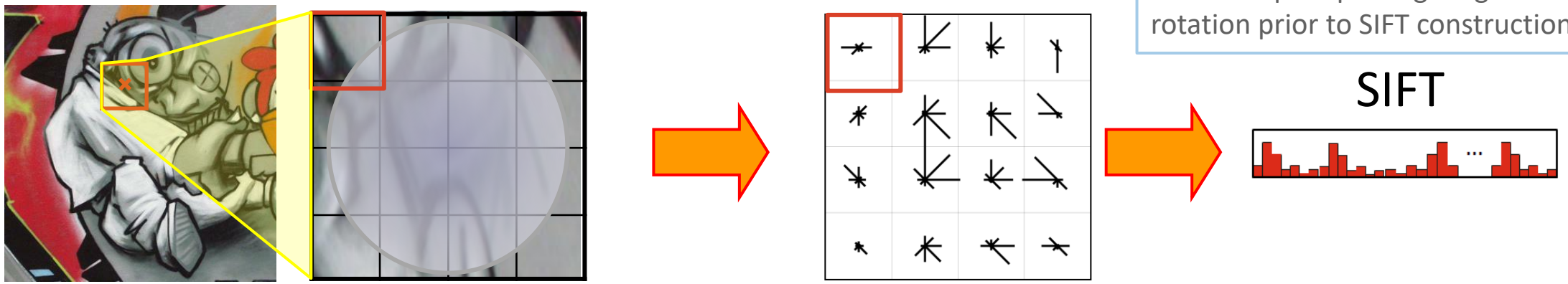




Intensity & Spatial shifts

# Descriptor: SIFT

- Scale Invariant Feature Transform:
  - Split region into 4x4 sub-regions: 16 cells
  - Calculate gradients on each pixel and smooth over a few neighbors.
  - In each cell calculate a histogram of gradient orientations (8 directions)
    - Each point contributes with a weight proportional to its gradient magnitude
    - The contribution is weighted by a Gaussian centered at the region center
  - Descriptor (Stack histograms into a vector and normalize): 4x4x8 = 128 dim



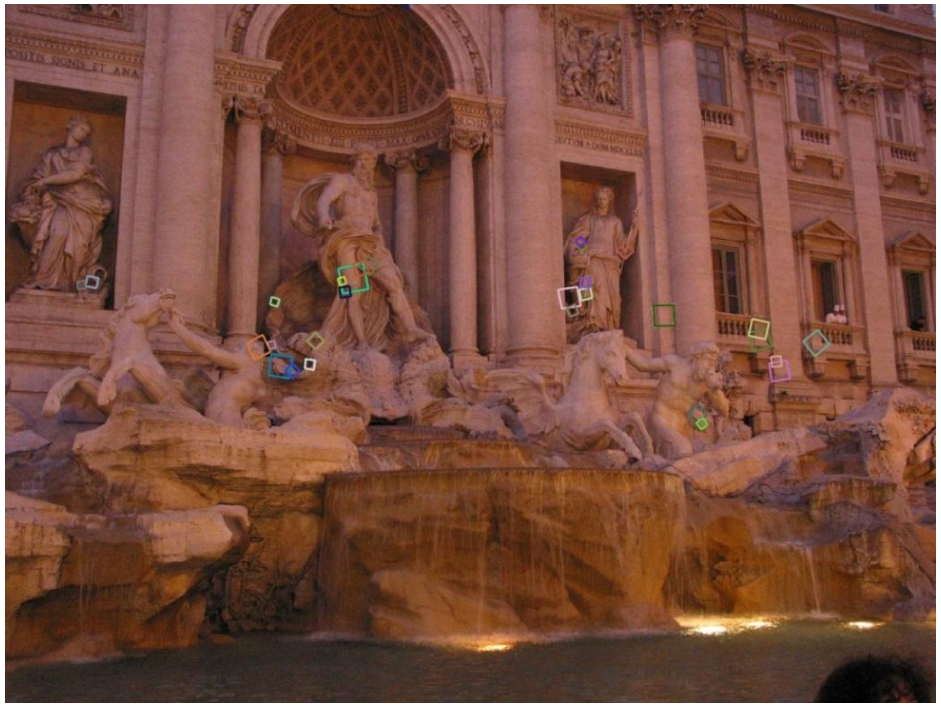

**Note:** for computational efficiency, [Lowe2004] implements region rotation implicitly in the computation of individual sub-region histograms (i.e., for each pixel, compute the sub-regions it might contribute to after rotation, rotate the gradient appropriately and thus calculate the sub-region histogram bin). Thus, classical implementations give **rotation-invariant SIFT** descriptor, and do not require per-region geometric rotation prior to SIFT construction.

Actually, there are a few important suttle details:

David G. Lowe. "Distinctive image features from scale-invariant keypoints." *IJCV* 60 (2), pp. 91-110, 2004.

# Summary: SIFT

- Detect keypoints and their scale/orientation/affine frames and normalize the patches
- Encode by a SIFT descriptor
- A surprisingly robust key-point descriptor
  - Allows ~60 degrees of out-of-plane rotation ; Robust to significant intensity changes
  - Fast (lots of real-time implementations)

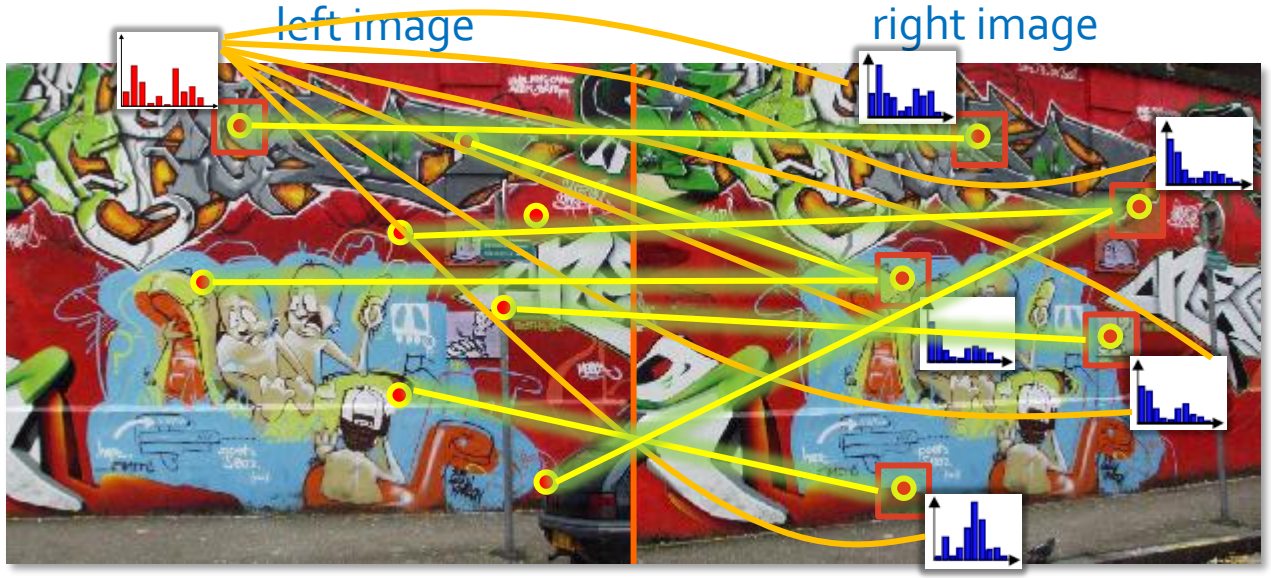


http://people.csail.mit.edu/albert/ladypack/wiki/index.php/Known_implementations_of_SIFT

Slide credit: Steve Seitz

# Correspondences using keypoints

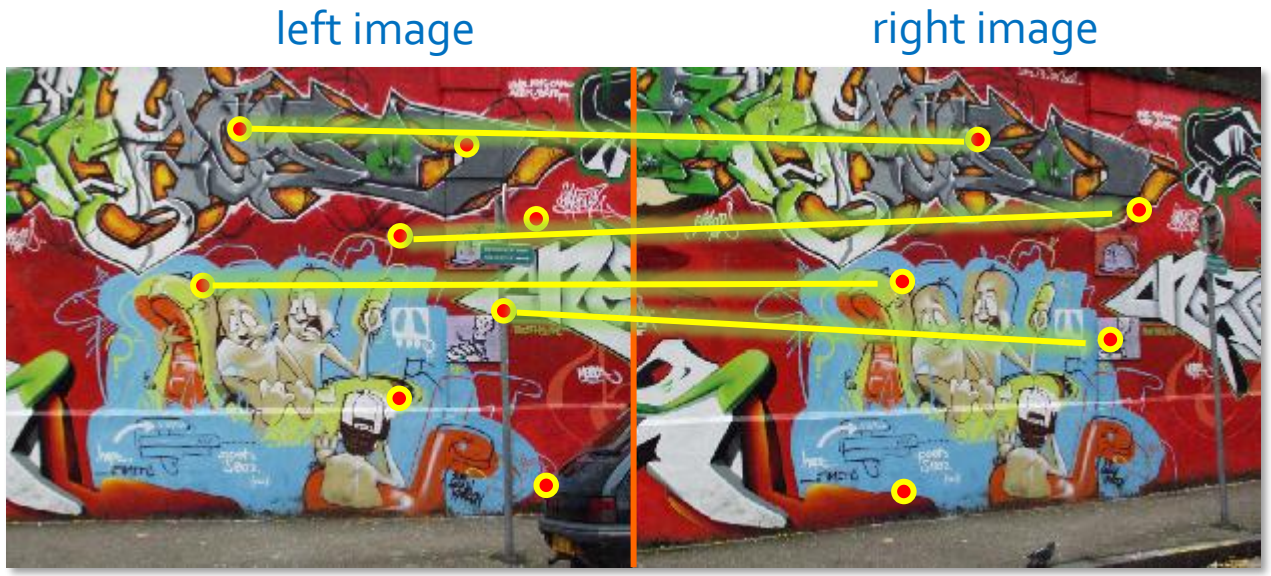
- Compare keypoints by calculating the Euclidean distance ($L_2$ distance) between descriptors.




- Strategy 1: For each keypoint in the left image identify the most similar keypoint in the right image.
- Result: potential (putative) matches/correspondences

# Correspondences using keypoints

- Strategy 2: Keep only symmetric matches



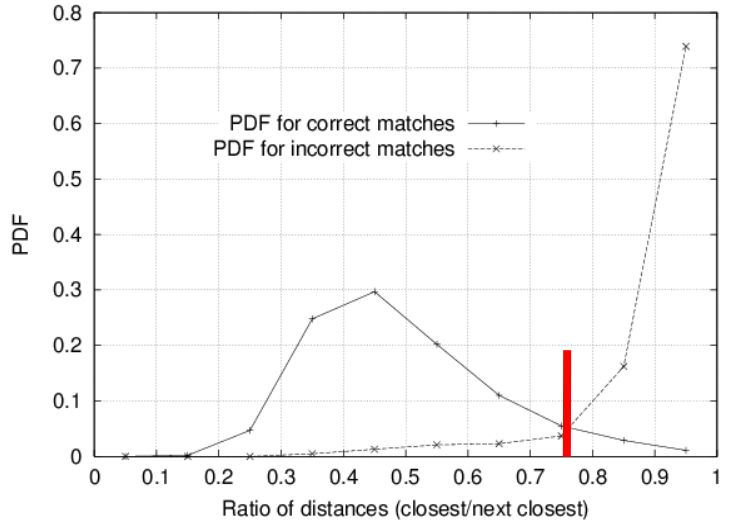

Definition of a symmetric match:

- “Let point A be a point in the left image and point B its match in the right image. If B is most similar to A among all points in the right image and vice versa, they form a symmetric match.”

# Correspondences using keypoints

- Strategy 3: *Calculate the distance between A and the second-most similar keypoint DESCRIPTOR and the most similar keypoint DESCRIPTOR in the other image.*
  - Ratio $R$ of these two distances will be low for distinctive key-points and high for non-distinctive ones.
  - Threshold ~0.8 on *R* gives good results with SIFT.

$$R = \frac{d_{\text{first}}}{d_{\text{second}}}$$


David G. Lowe. "Distinctive image features from scale-invariant keypoints." *IJCV* 60 (2), pp. 91-110, 2004.

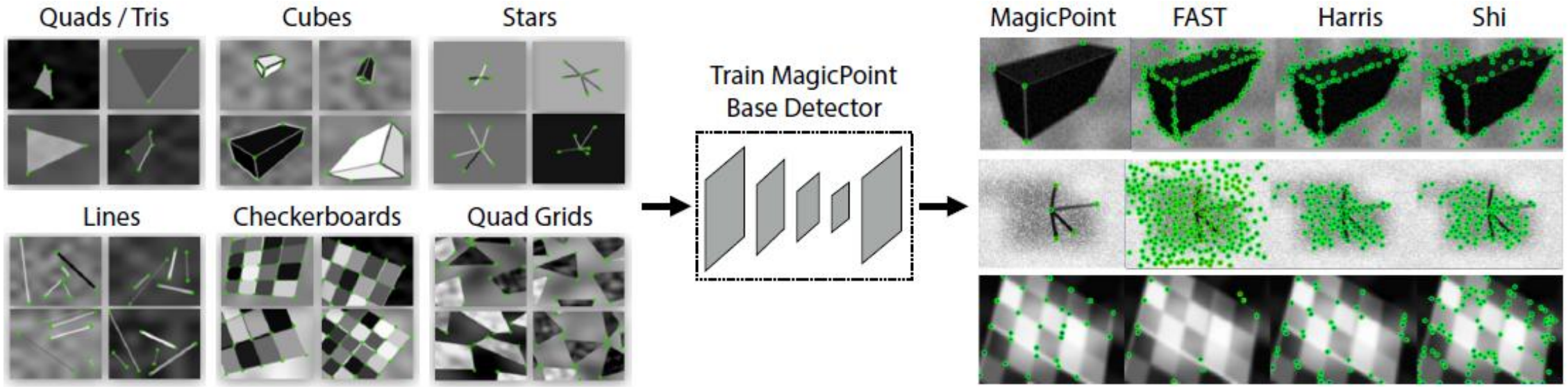
# Finally stitching can be fully automated

- Detect key-points independently in each image
- Determine potential correspondences
- Reject improbable correspondences by strategy 1,2, or 3
- Perform RANSAC to find the inliers and fit the model

All correspondences + filtering by strategy 1,2,3 +RANSAC:

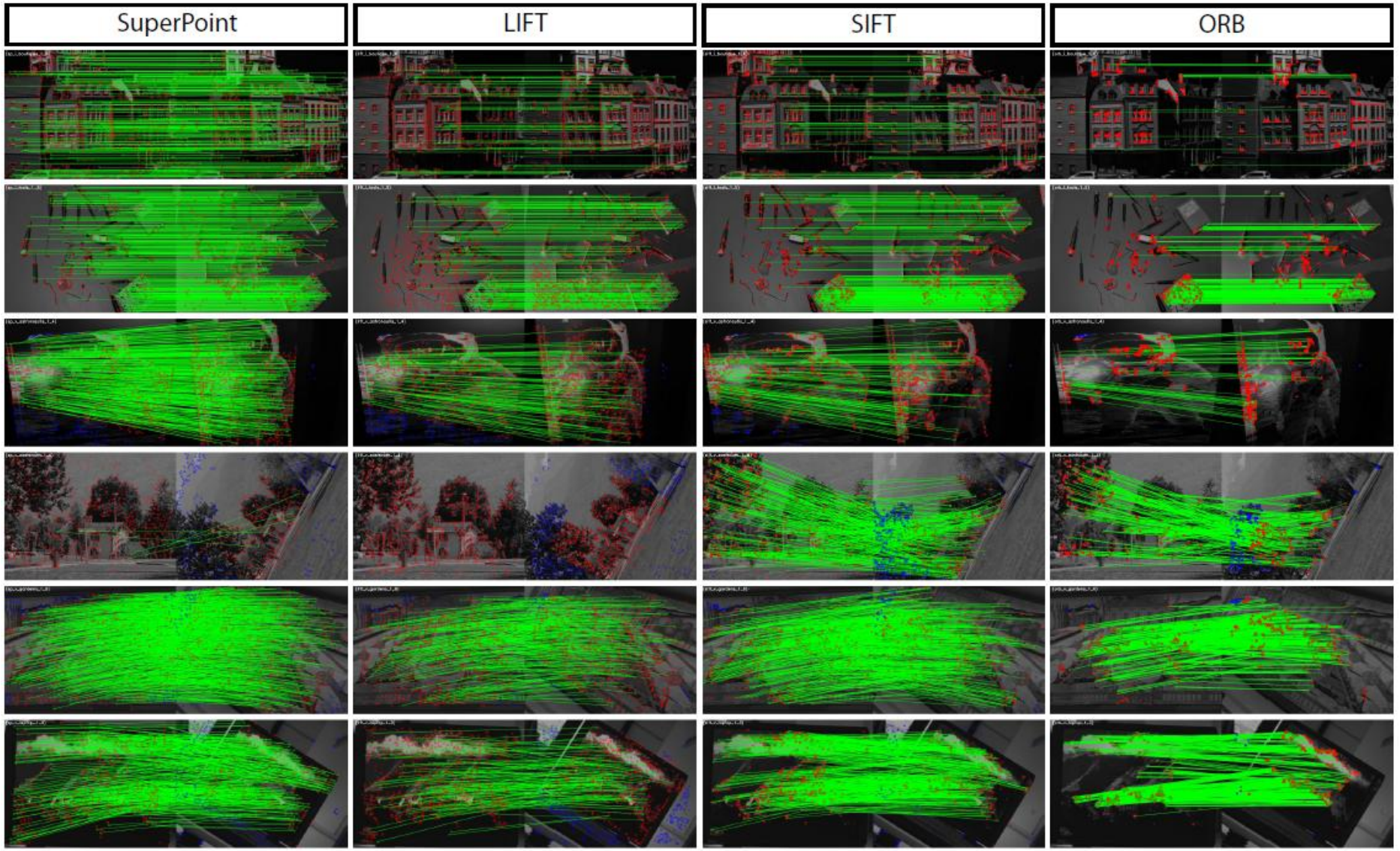
# Trainable keypoint detection (modern approaches)

- SuperPoint – a convolutional neural network trained to “fire” on a key point
- Keypoints trained on simulated data, adapted to real data, re-trained for joint extraction of keypoints and descriptors



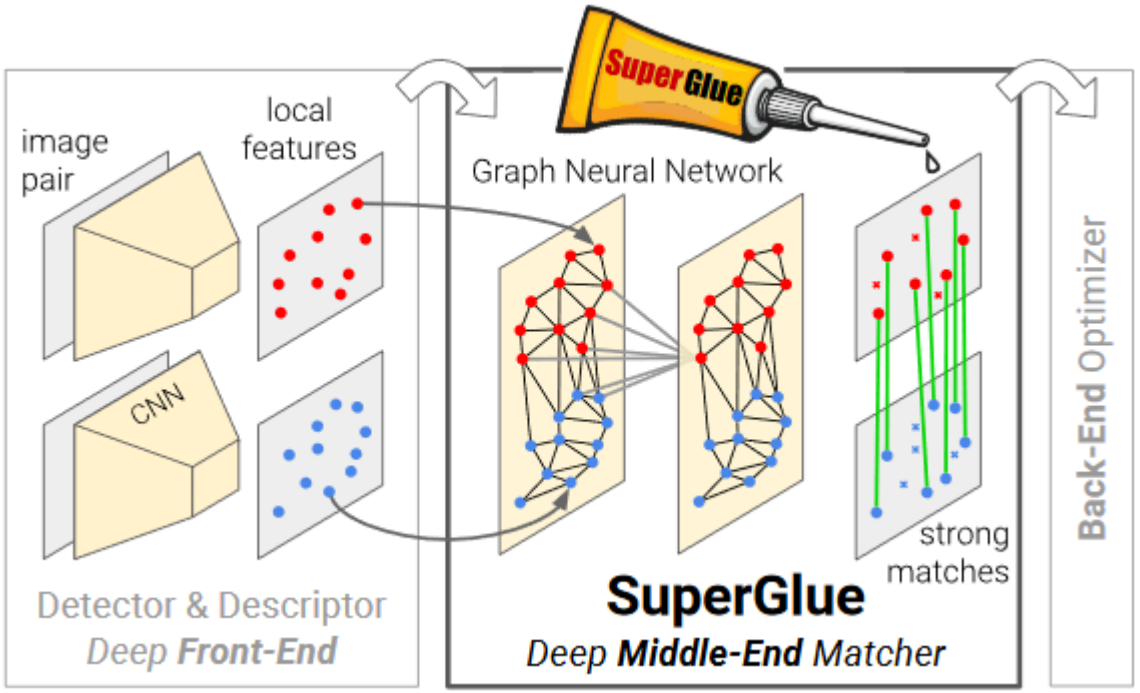

DeTone et al., SuperPoint: Self-Supervised Interest Point Detection and Description, CVPR2018

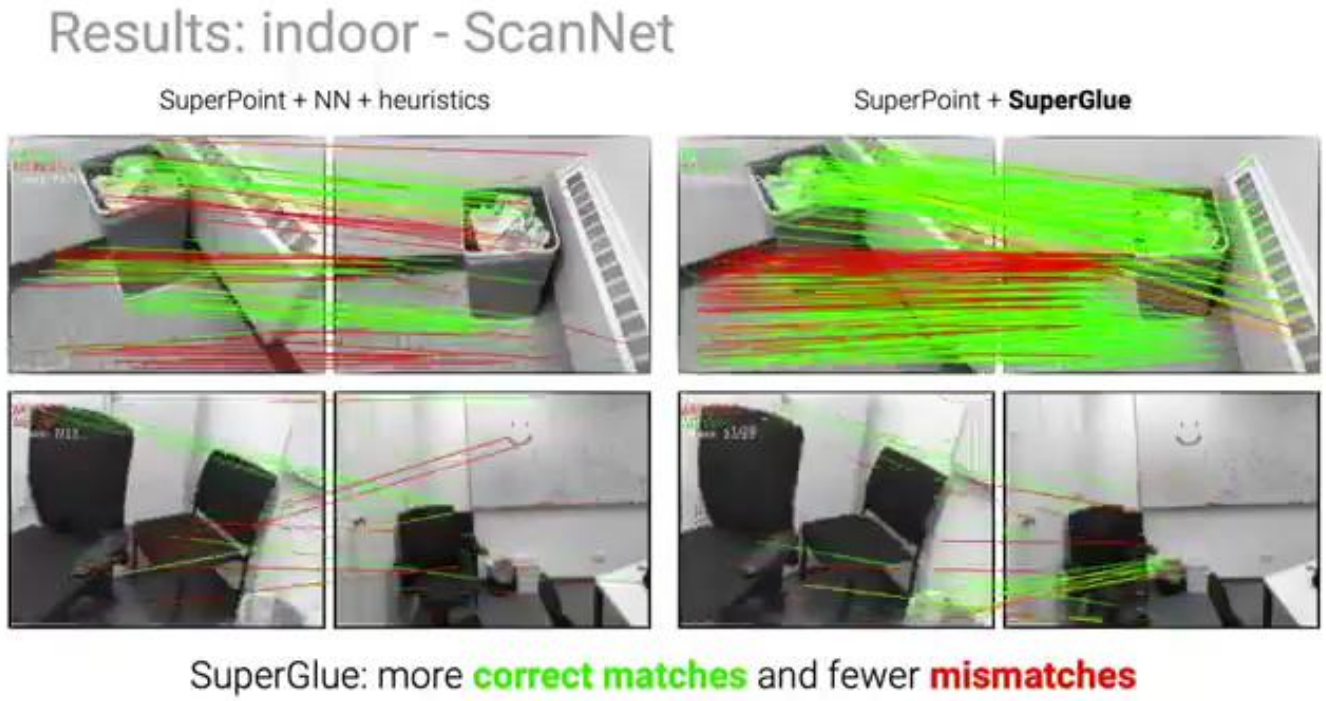
# Recent work on keypoint detection

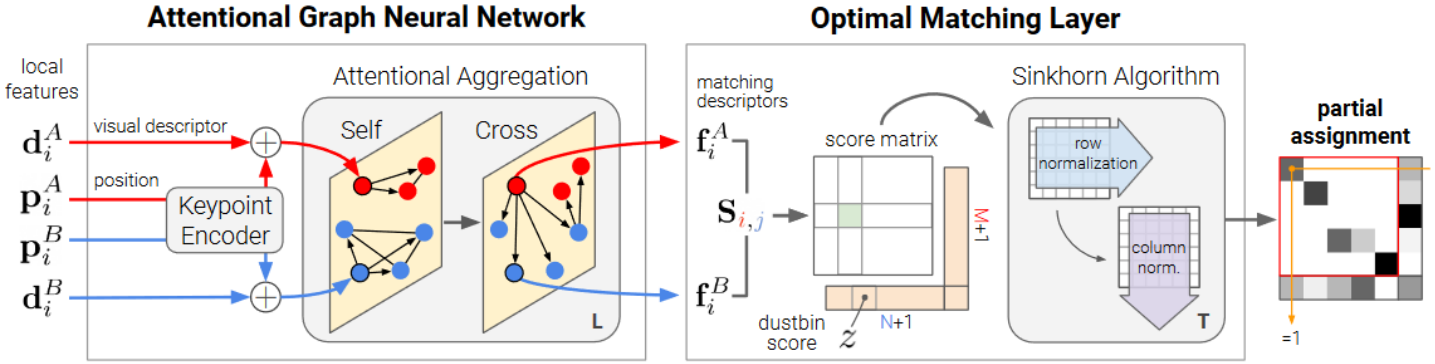



DeTone et al., SuperPoint: Self-Supervised Interest Point Detection and Description, CVPR2018

# “Recent” work on correspondences matching



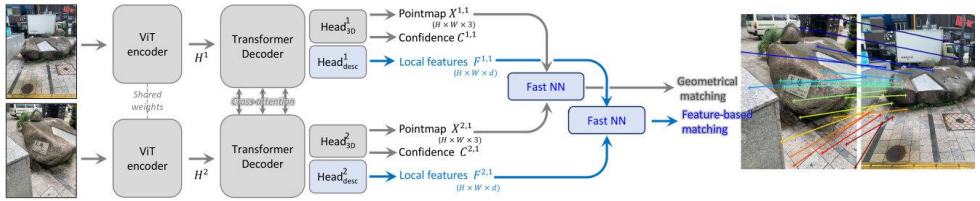



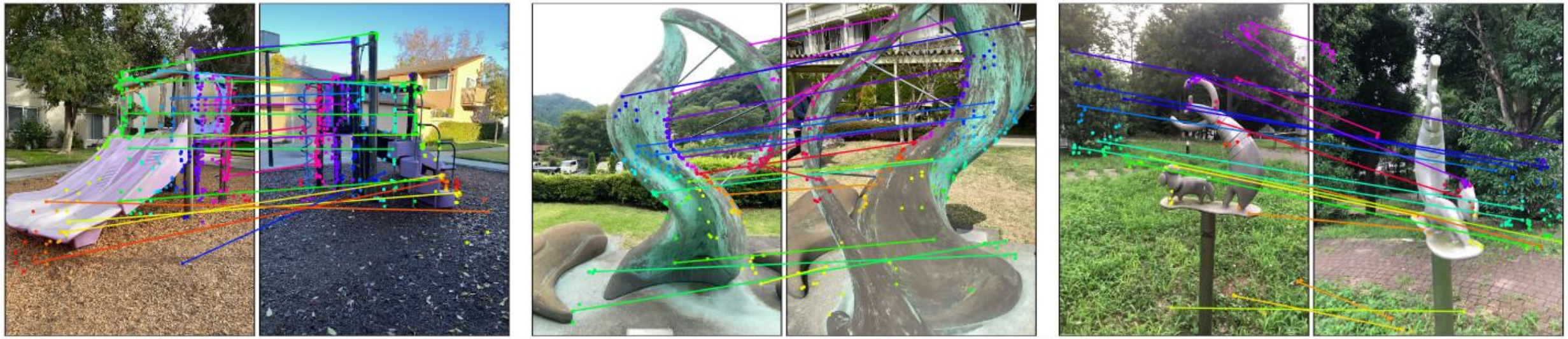

Quote: “The estimated correspondences are so good that a robust estimator is not required…”

Sarlin et al. SuperGlue: Learning Feature Matching with Graph Neural Networks, CVPR2020 (video)

# MaSt3r

- Grounding Image Matching in 3D with MASt3R
- Built upon a 3D foundation model
- Considers keypoints+descriptors+3D jointy
- Remarkable matching capabilities!



https://github.com/naver/mast3r?tab=readme-ov-file

Leroy et al., Grounding Image Matching in 3D with MASt3R, ECCV 2024

# Numerous detectors/descriptors exist

- We have only considered the most popular descriptor (SIFT, Lowe2004)
  - Note that Lowe proposed DoG for keypoint detection and SIFT for descriptor – don't mix these!

- Many efficient and really fast descriptors have been presented since.

- Significant research currently invested into end-to-end learning the keypoint detection, description and matching process by neural nets.
  See Image Matching Challenge (link to the CVPR2023 workshop)

# Back to the panorama stitching

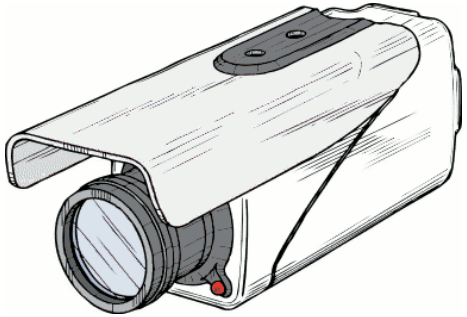
- Keypoints: Detection, Description, Matching
- RANSAC: Robust estimation of transformation model parameters
- But ... What should “the transformation model” be?

*The camera model...*

# References


- David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition), (prva izdaja dostopna na spletu)
- R. Szeliski,Computer Vision: Algorithms and Applications, 2010
- R. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, 2nd Edition, Cambridge University Press, 2004
- D. Lowe, Distinctive image features from scale-invariant keypoints, IJCV 60(2), pp. 91-110, 2004
- T. Tuytelaars, K. Mikolajczyk, Local Invariant Feature Detectors: A Survey, Foundations and Trends in Computer Graphics and Vision, Vol. 3, No. 3, pp 177-280, 2008.

Univerza v Ljubljani



# Machine perception
# Camera geometry

Matej Kristan


Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

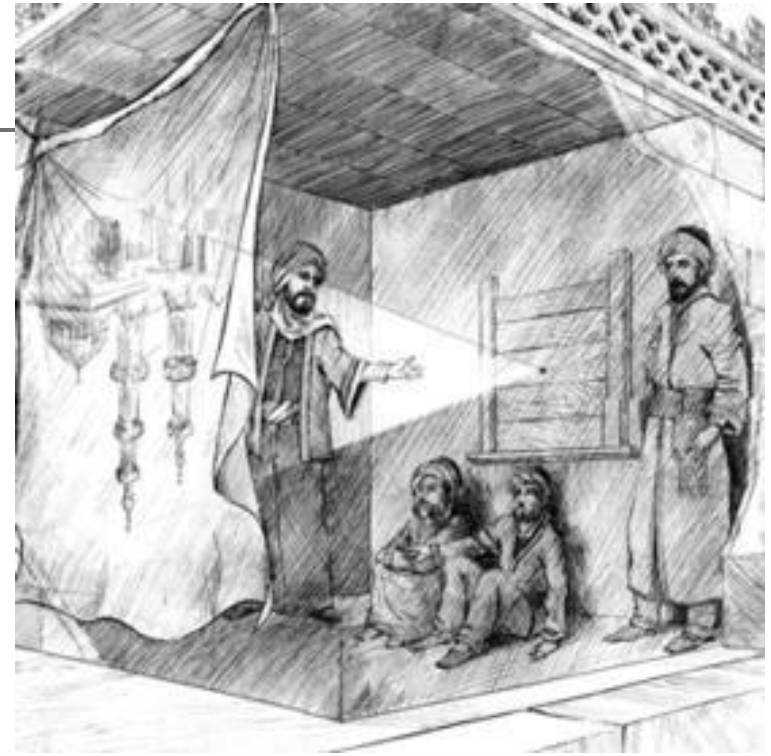
# Extracting 3D information from a 2D image?

- Shading, Texture, Focus, Perspective, …

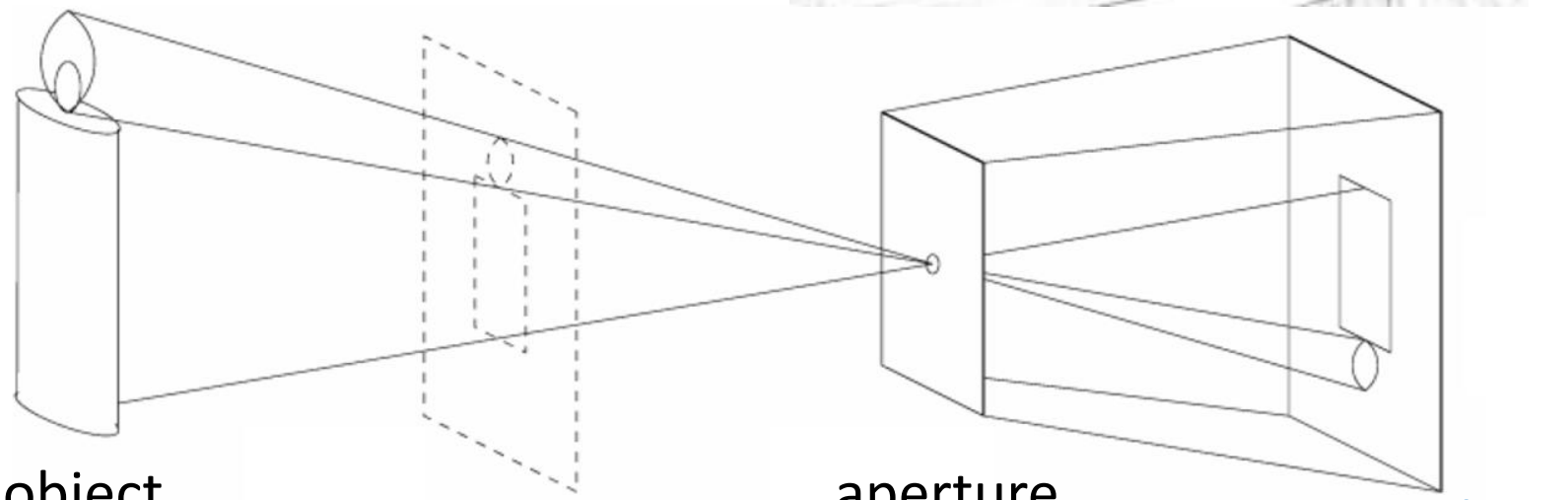
- Humans learn how 3D structure *looks* in a 2D image

- In computer vision, we require a model of 3D-to-2D transform to measure the 3D content

# Recall the pinhole camera

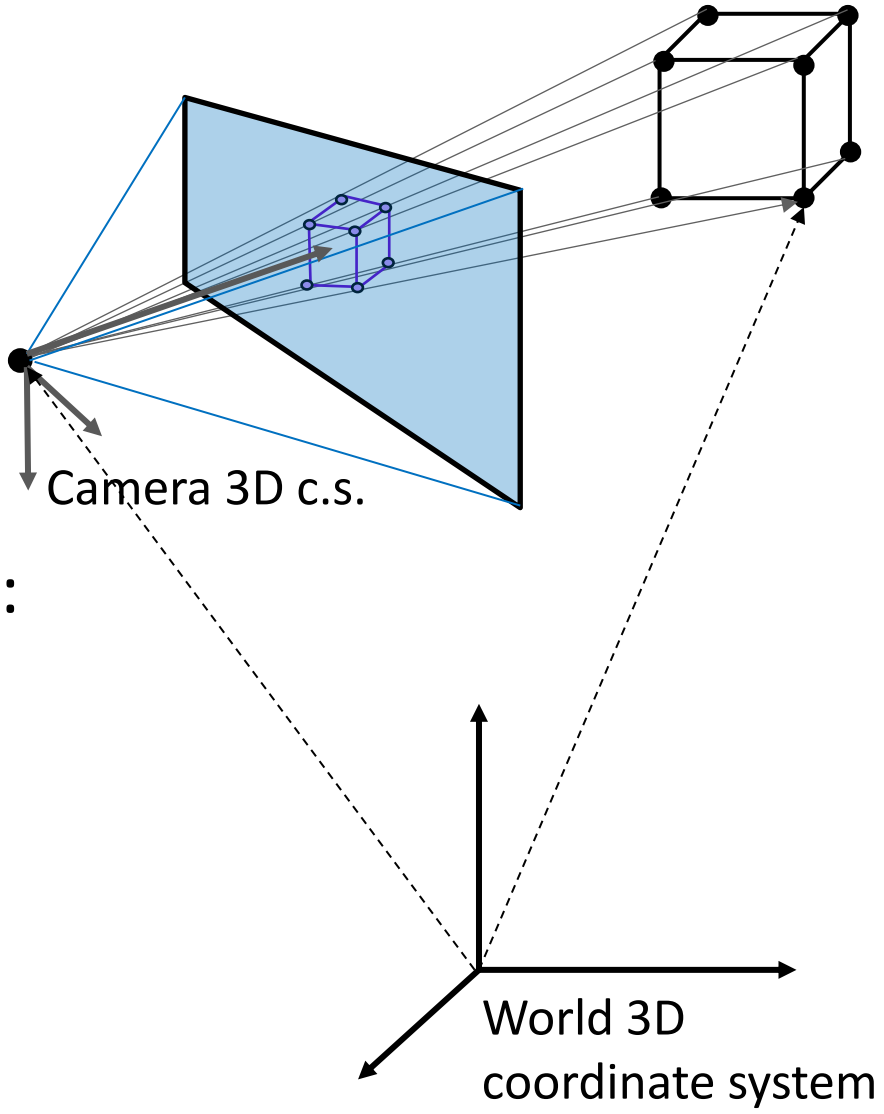


- Earliest and remarkably correct written description:
  ~500 BC Mohist canon [founder Mo-ti]
  (ancient Chinese texts)
- A simple standard camera model
  - A box with a small aperture
  - Image forms on the back
- Virtual vs actual image plane



# Single-view geometry

- Points in a world 3D coordinate system (c.s.)
- Project to image plane, i.e., into 2D pixels

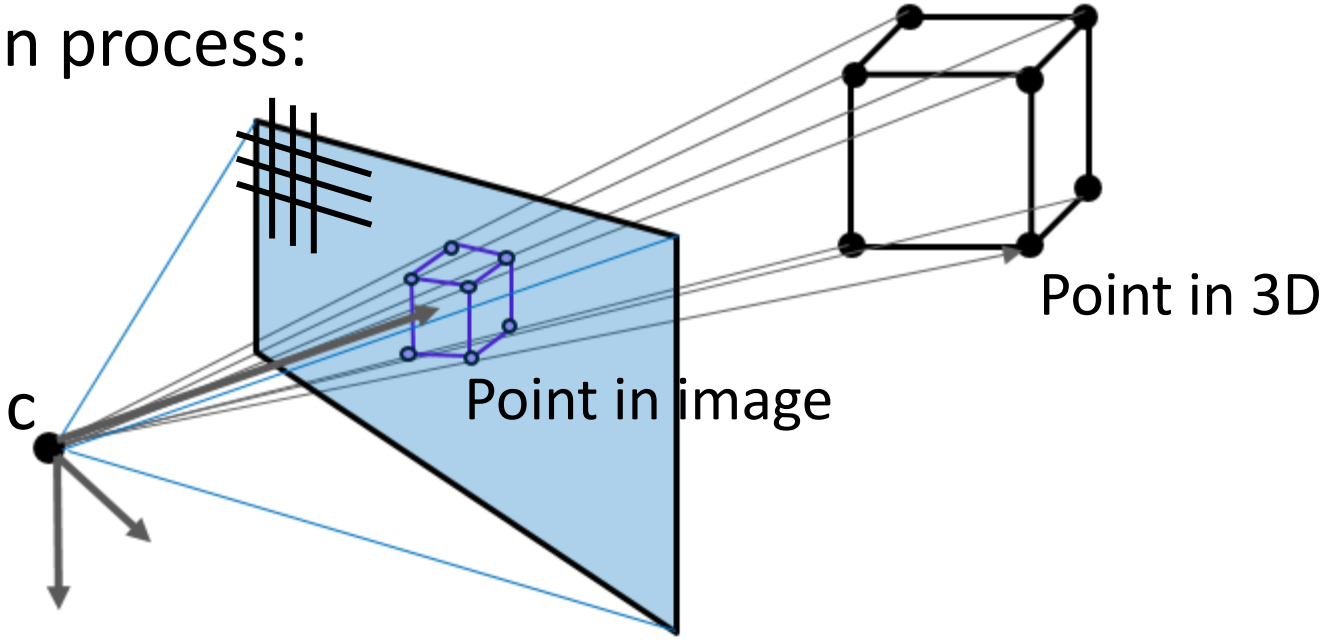



Projection decomposed into two kinds:

1. “Extrinsic” projection
   3D World → 3D Camera
2. “Intrinsic” projection
   3D Camera → 2D Image

# Consider “Intrinsic” projection first

Recall the image formation process:




- A point written in camera 3D coordinate system (meters)
- Projected to camera image plane (meters)
- Projected to discretized image (pixels)
- Let’s derive transformations for a pinhole camera!

# Homogeneous coordinates

- Euclidean geometry uses Cartesian coordinate system
- But for a projective geometry, homogenous coordinates are much more appropriate
- E.g., can easily encode a point in infinity (try that in Euclidean...)

Cartesian form

Homogeneous form

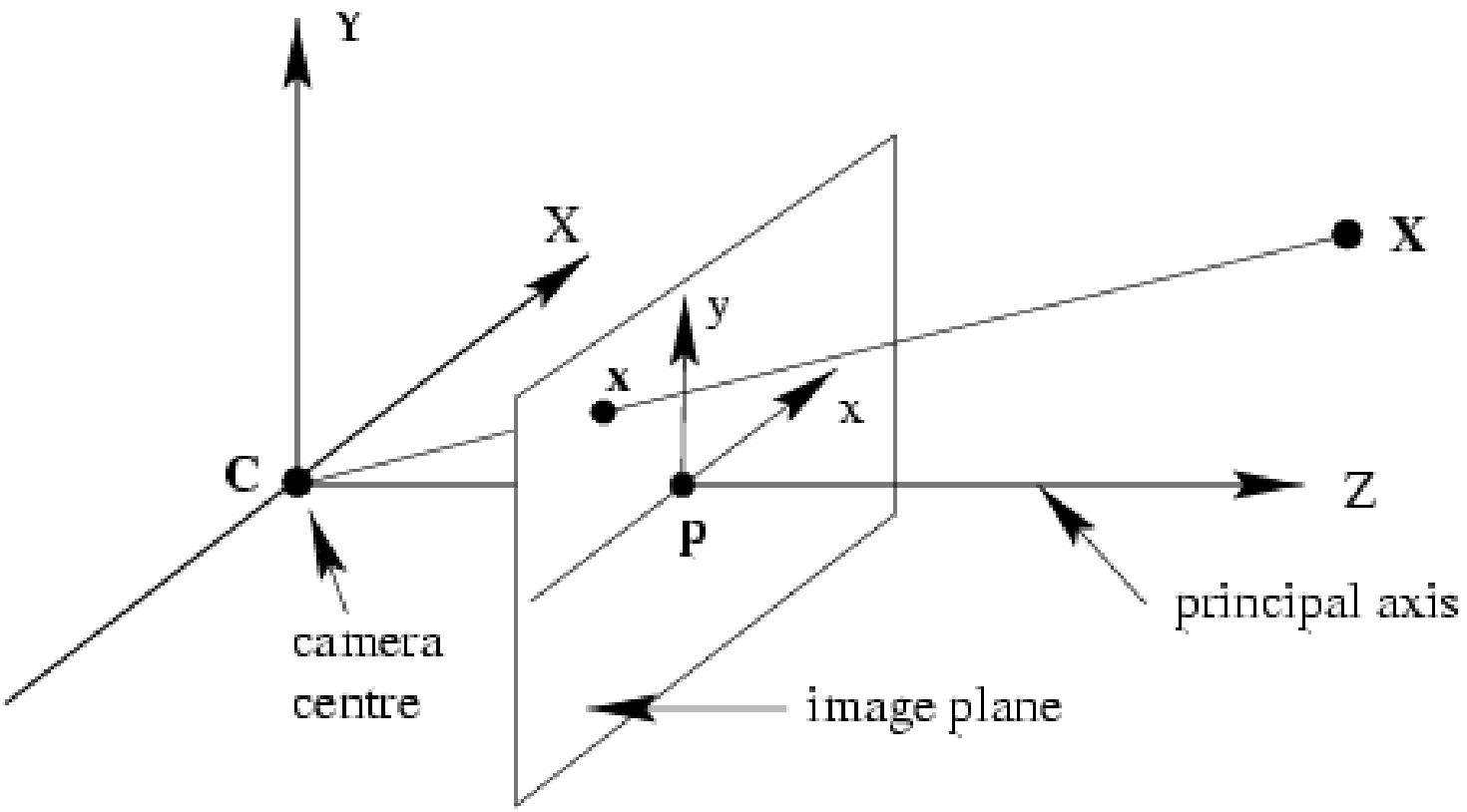
$$\begin{bmatrix} x \\ y \end{bmatrix} \Longrightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Multiplying by a scalar ($\neq 0$) value does not change a point!

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} wx \\ wy \\ w \end{bmatrix}$$

- From homogeneous system to Euclidian:
  Simply divide by the last coordinate to make it 1.

# Camera coordinate system (meters)



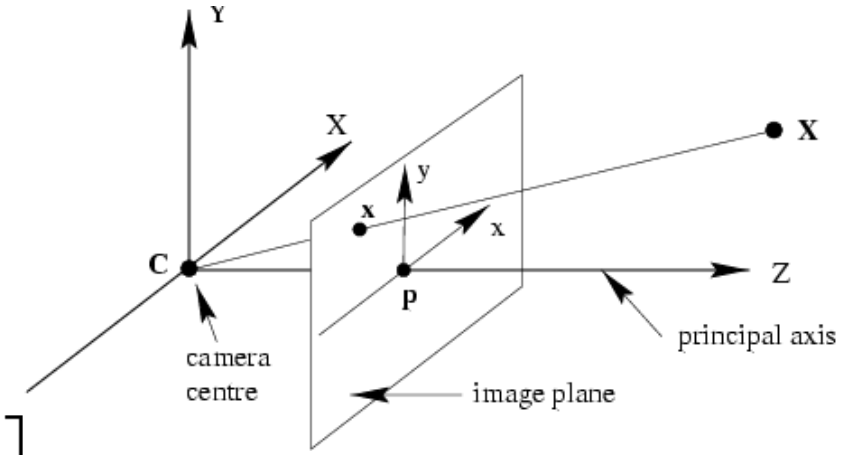

- *Principal axis*:

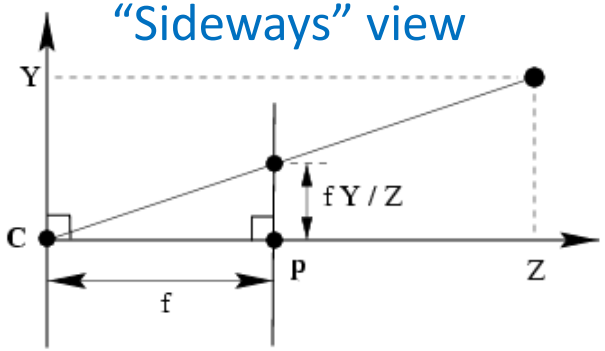
  A line from camera center perpendicular to image plane.

- *Principal point* ($\boldsymbol{p}$): A point where the principal axis punctures the image plane.

- *Normalized (camera) coordinate system*: 2D system with origin at the *principal point*.

# A pinhole camera revisited




$$\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \longleftarrow \text{3D point in world c.s.} \quad [X,Y,Z]^T \mapsto [fX/Z, fY/Z]^T \quad \text{2D projection to image plane} \longrightarrow \begin{bmatrix} fX/Z \\ fY/Z \\ 1 \end{bmatrix}$$

Rewrite in homogeneous coordinates

- Projection as vector-matrix multiplication:

$$\mathbf{x} = \mathrm{P}_0 \mathrm{X}$$

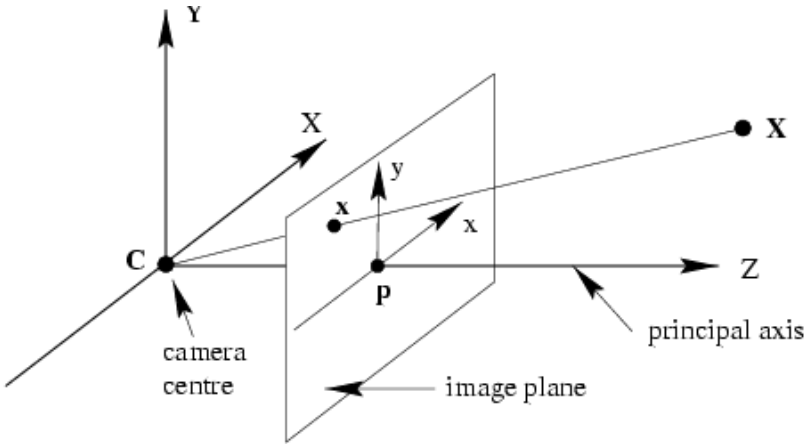
In 3D camera c.s.

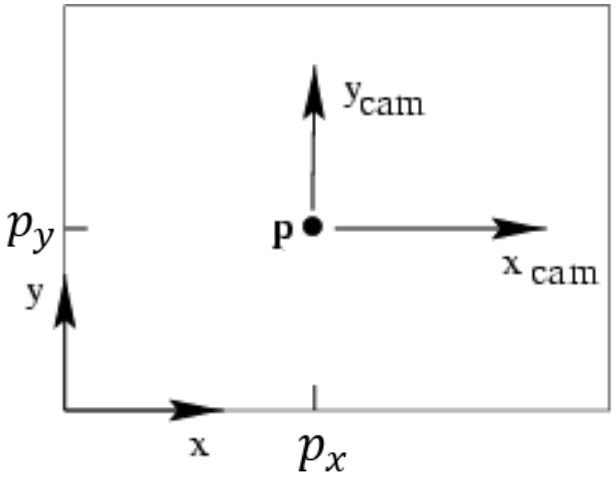
In 2D image plane c.s.

$$\begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \underbrace{\begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix}}_{\boldsymbol{P}_0} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

# From image plane to image pixels

- Change of coordinate system to image corner




- Normalized camera coordinate system:

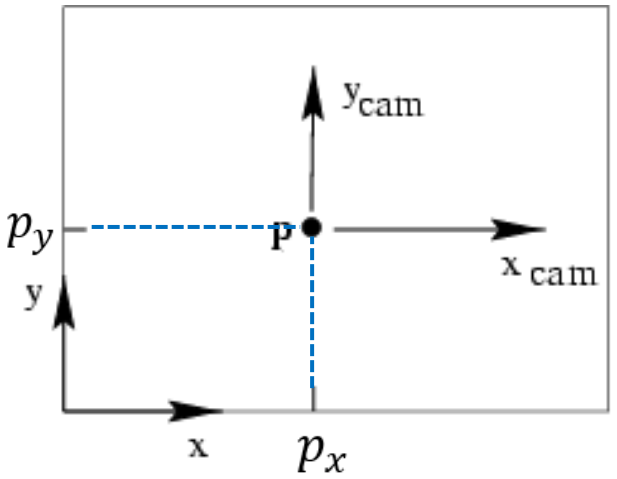
  Origin in principal point $\boldsymbol{p} = [\boldsymbol{p}_x, \boldsymbol{p}_y]^T$.

- Image coordinate system:

  Origin in the corner of the image sensor.

# From image plane to image pixels (1/3)

- Change the c.s. origin by the principal point $\boldsymbol{p}$:




- Write the transformation:

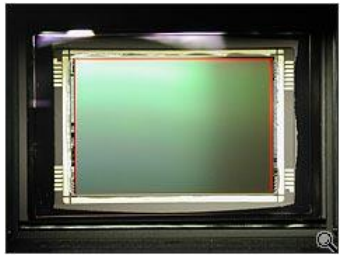
$$(X, Y, Z) \mapsto (fX/Z + p_x, fY/Z + p_y)$$

- Rewrite in vector-matrix multiplication:

$$\begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \underbrace{\begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\boldsymbol{P}_0} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \qquad \mathbf{x} = \mathbf{P}_0 \mathbf{X}$$

# From image plane to image pixels (2/3)

- Projection to a sensor of size $W_S \times H_S$ (in meters).
- Pixels are arranged into a *rectangular* $M_x \times M_y$ pixels matrix.
- Let $m_x = M_x/W_S$ and $m_y = M_y/H_S$.
- Construct projection to pixels:

Just multiply by another matrix:

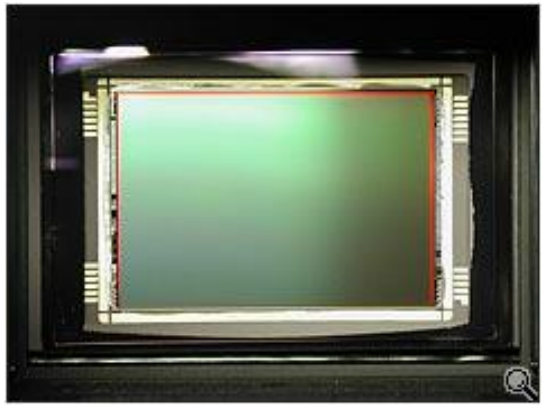
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \underbrace{\begin{bmatrix} m_x & 0 & 0 \\ 0 & m_y & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{pixel/m}} \underbrace{\begin{bmatrix} f & 0 & p_x & 0 \\ 0 & f & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{m}} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

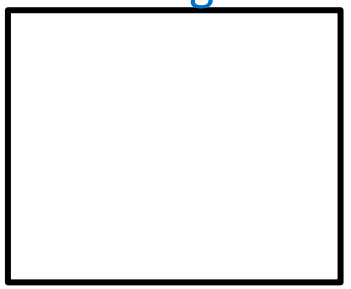
Abbreviated form:

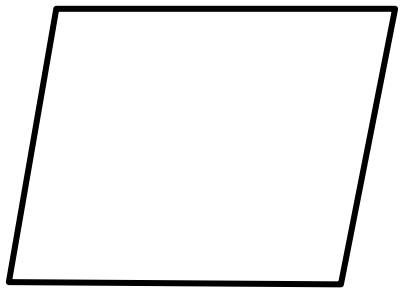
$$\Rightarrow \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} \alpha_x & 0 & x_0 & 0 \\ 0 & \alpha_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

# From image plane to image pixels (3/3)

- In general difficult to guarantee a rectangular sensor.






Rectangular

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} \alpha_x & 0 & x_0 & 0 \\ 0 & \alpha_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Projection matrix $P_0$

Skewed

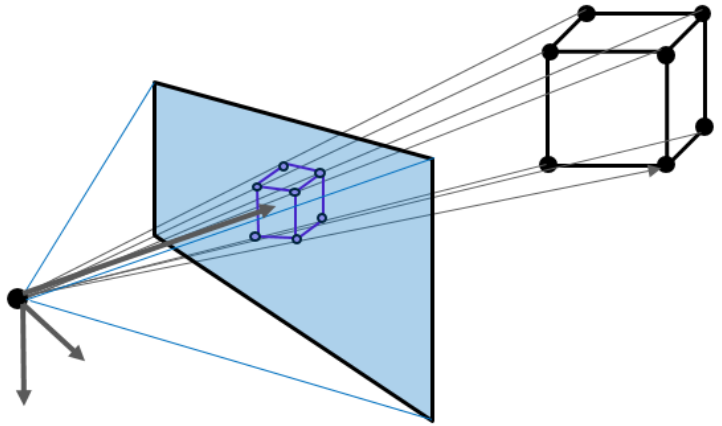
$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} \alpha_x & s & x_0 & 0 \\ 0 & \alpha_y & y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

Projection matrix $P_0$

# Calibration matrix

- Expand the projection matrix $\boldsymbol{P}_0$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \underbrace{\begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathrm{P}_0 = \mathrm{K}[\mathrm{I} \mid 0]} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$



- Calibration matrix $\boldsymbol{K}$:

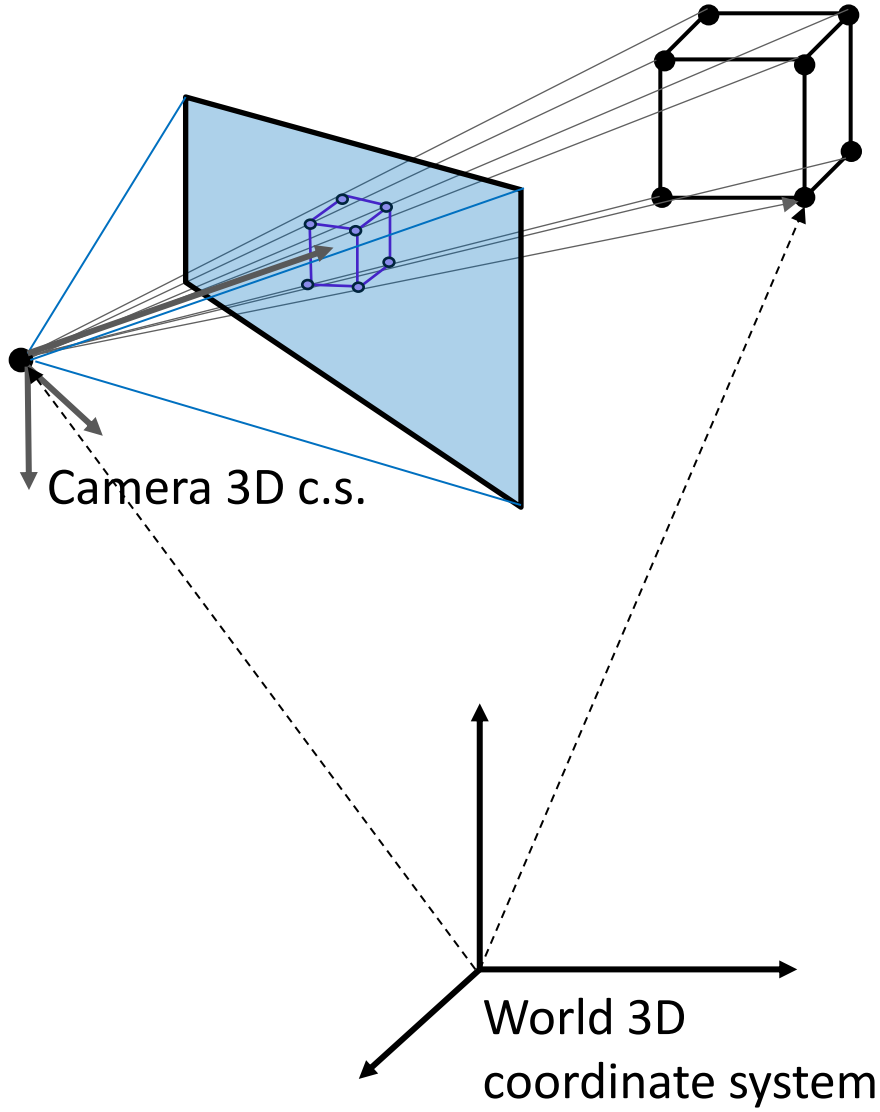
  *"Prescribes projection of 3D point in camera c.s. into pixles!"*

$$\mathbf{K} = \begin{bmatrix} \alpha_x & s & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

Q: What is the meaning of each element of the calibration matrix?

# Single-view geometry

- Points in a world 3D coordinate system (c.s.)
- Project to image plane into 2D pixels

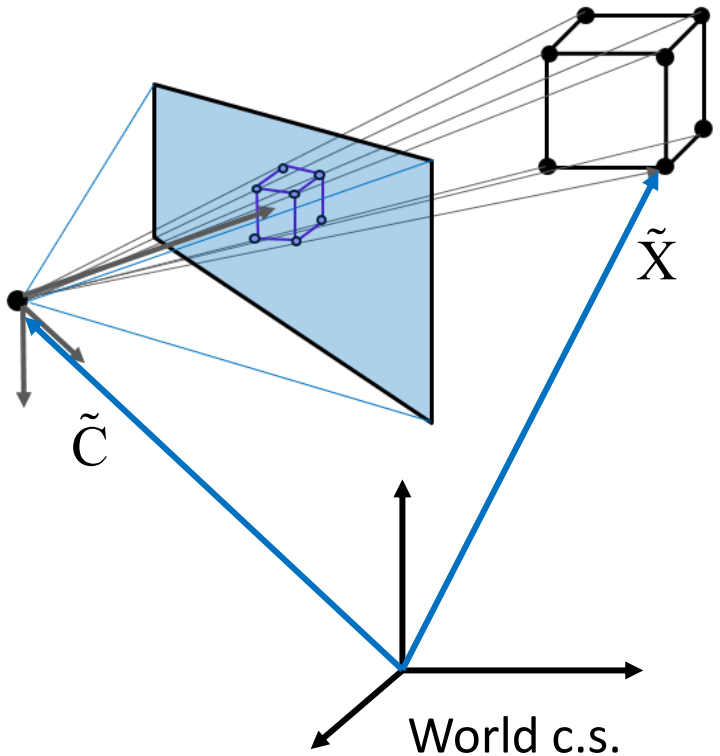



Two kinds of projection:

1. “Extrinsic” projection
   3D World → 3D Camera ?
2. “Intrinsic” projection
   3D Camera → 2D Image ✓

# From world c.s. to 3D camera c.s.

- The 3D camera coordinate system (c.s.) is related to 3D world c.s. by a rotation matrix $\boldsymbol{R}$ and translation $\tilde{\boldsymbol{t}} = \widetilde{\boldsymbol{C}}$.




$\boldsymbol{R}$ ... How to rotate the world c.s. about its own origin to align it with the camera c.s.

$\widetilde{\boldsymbol{C}}$ ... Camera origin in world c.s.

$\widetilde{\boldsymbol{X}}$ ... Point in 3D world c.s.

$\widetilde{\boldsymbol{X}}_{cam}$ ... Same point $\widetilde{\boldsymbol{X}}$, but written in 3D camera c.s.

World-to-camera c.s. transformation:

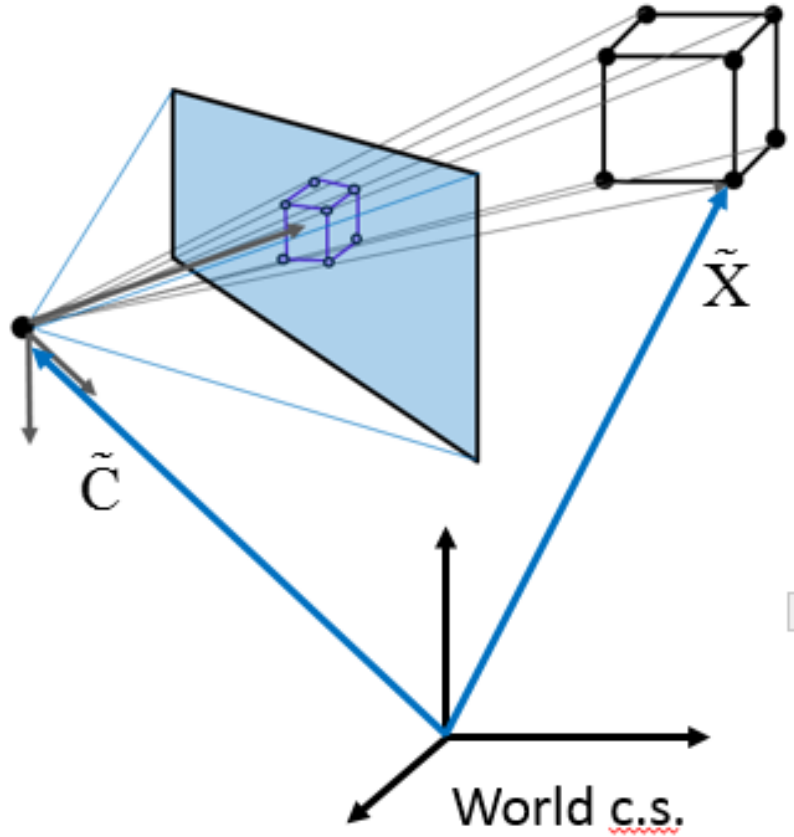
$$\widetilde{\mathrm{X}}_{\mathrm{cam}} = \mathrm{R}\left(\widetilde{\mathrm{X}} - \widetilde{\mathrm{C}}\right)$$

(Euclidean)

# From world c.s. to 3D camera c.s.

- Euclidean form:

$$\tilde{X}_{cam} = R\left(\tilde{X} - \tilde{C}\right)$$




- Rewrite by using homogeneous coordinates:

$$X_{cam} = \begin{bmatrix} \tilde{X}_{cam} \\ 1 \end{bmatrix} \qquad X = \begin{bmatrix} \tilde{X} \\ 1 \end{bmatrix}$$

$$X_{cam} = \begin{pmatrix} \tilde{X}_{cam} \\ 1 \end{pmatrix} = \begin{bmatrix} R & -R\tilde{C} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} \tilde{X} \\ 1 \end{pmatrix} = \begin{bmatrix} R & -R\tilde{C} \\ 0 & 1 \end{bmatrix} X$$

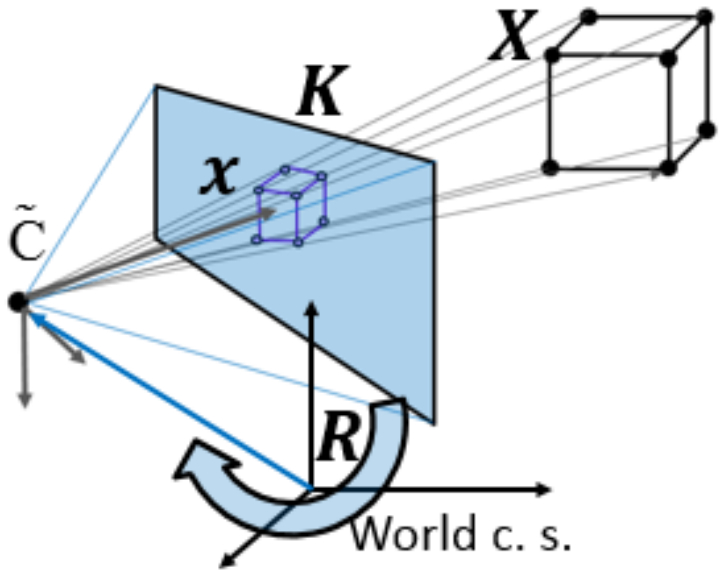
# Putting it all together

- Camera parameters are specified by a calibration matrix $\boldsymbol{K}$, the projection center in world c.s. $\tilde{\boldsymbol{C}}$ and rotation matrix $\boldsymbol{R}$.
- A 3D point in world coordinates (homogeneous) $\boldsymbol{X}$, is projected into pixels $\boldsymbol{x}$ by the following relation:

$$\mathrm{x} = \mathrm{K}\left[\mathrm{I} \mid 0\right]\mathrm{X}_{\mathrm{cam}} = \mathrm{K}\left[\mathrm{R} \mid -\mathrm{R}\tilde{\mathrm{C}}\right]\mathrm{X} = \mathrm{PX}$$

$$\mathrm{P} = \mathrm{K}\left[\mathrm{R} \mid \mathrm{t}\right], \quad \mathrm{t} = -\mathrm{R}\tilde{\mathrm{C}}$$

$$\mathrm{X}_{\mathrm{cam}} = \begin{bmatrix} \mathrm{R} & -\mathrm{R}\tilde{\mathrm{C}} \\ 0 & 1 \end{bmatrix} \mathrm{X}$$




Note the structure of the projection matrix!

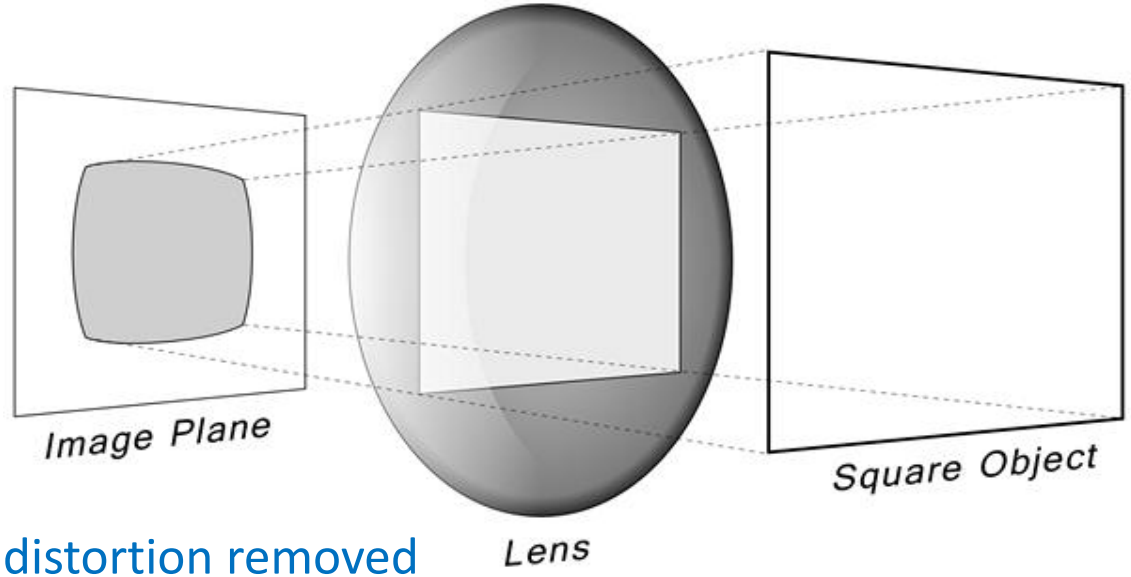
Q: What needs to be known to construct the projection matrix?

# Lens adds a nasty nonlinearity (and other issues)

Straight lines are no longer straight!

Nonlinearity *should be removed*

to apply a pinhole model!

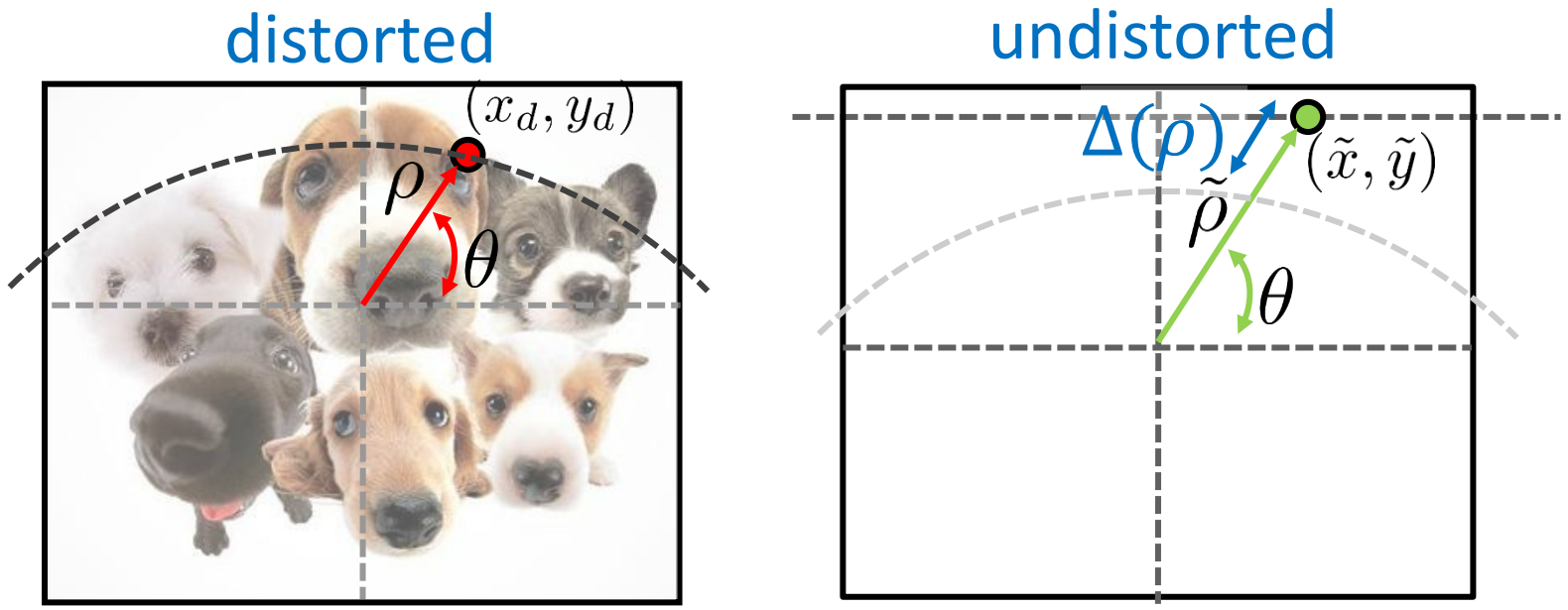



radially distorted image

radial distortion removed

# Lens adds a nonlinearity to the camera model

- Lens distortion assumed radially symmetric
- Radially expand an image to un-distort

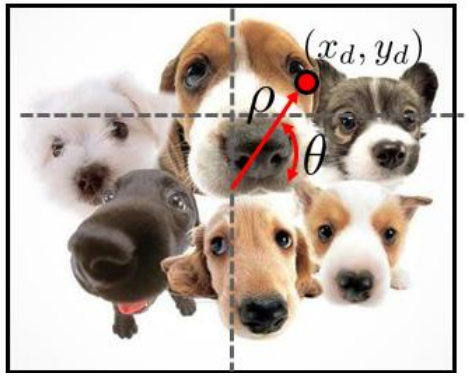


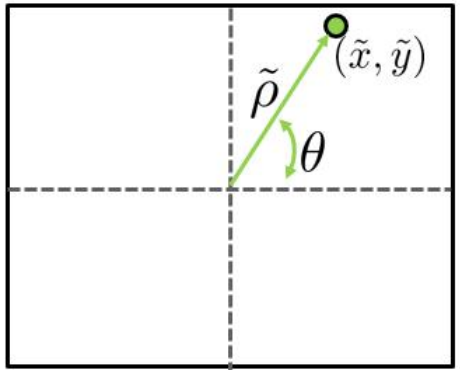

$$(\rho, \theta) \rightarrow (\tilde{\rho}, \theta)$$

- In this transformation, only the radius of transformed point changes, but the angle remains unchanged.

# Lens adds a nasty nonlinearity

- What kind of analytic function to use for transforming ρ?




- Typically, a polynomial is used (low degree used in practice):

$$\tilde{x} = x_d + (x_d - c_x)(K_1\rho^2 + K_2\rho^4 + \dots)$$
$$\tilde{y} = y_d + (y_d - c_y)(K_1\rho^2 + K_2\rho^4 + \dots)$$

- Parameters estimated by adjusting them until straight lines become straight.

[in Matlab use fminsearch (Python, fmin) for optimization method]

# Summary: camera parameters

Degrees of freedom (DoF)

- Intrinsic parameters:
  - Principal point coordinates — DoF: 2
  - Focal length — DoF: 1
  - Pixel scaling factor (rectangular pixels) — DoF: 1
  - *Shear (non-rectangular sensor array)* — DoF: 1
  - *Radial distortion*
- Extrinsic parameters
  - Rotation R — DoF: 3
  - Translation t — DoF: 3
- Camera projection matrix

$$K = \begin{bmatrix} f\alpha_x & s & p_x x_0 \\ & f\alpha_y & p_y y_0 \\ & & 1\ 1 \end{bmatrix}$$

$$P = K[R \mid t]$$

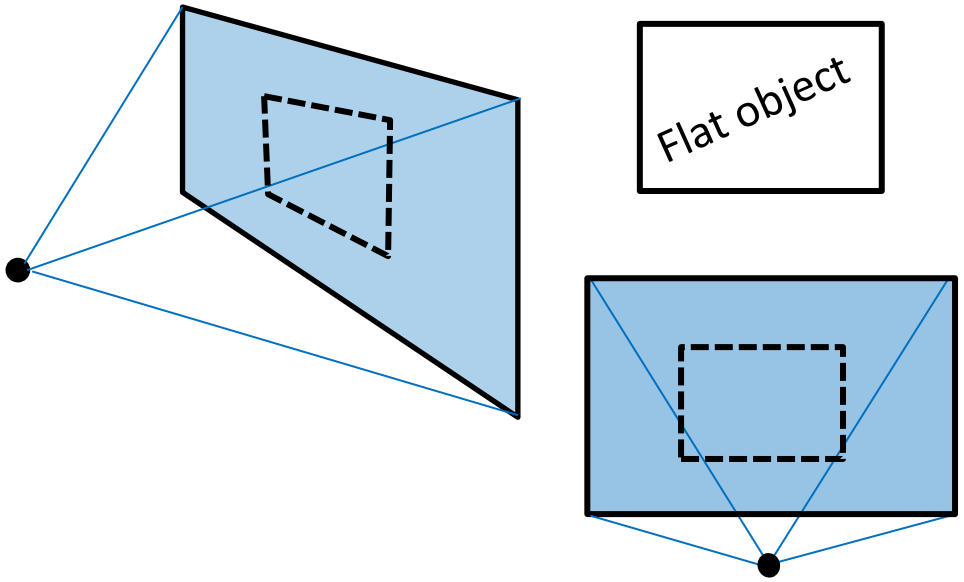
$\Rightarrow$ A pinhole camera: 9 DoF

$\Rightarrow$ Camera with rectangular pixels: 10 DoF

$\Rightarrow$ General camera (skewed sensor): 11 DoF

# Looking at flat objects

- A camera looking at the some planar object
- How would it look if the camera changed position?

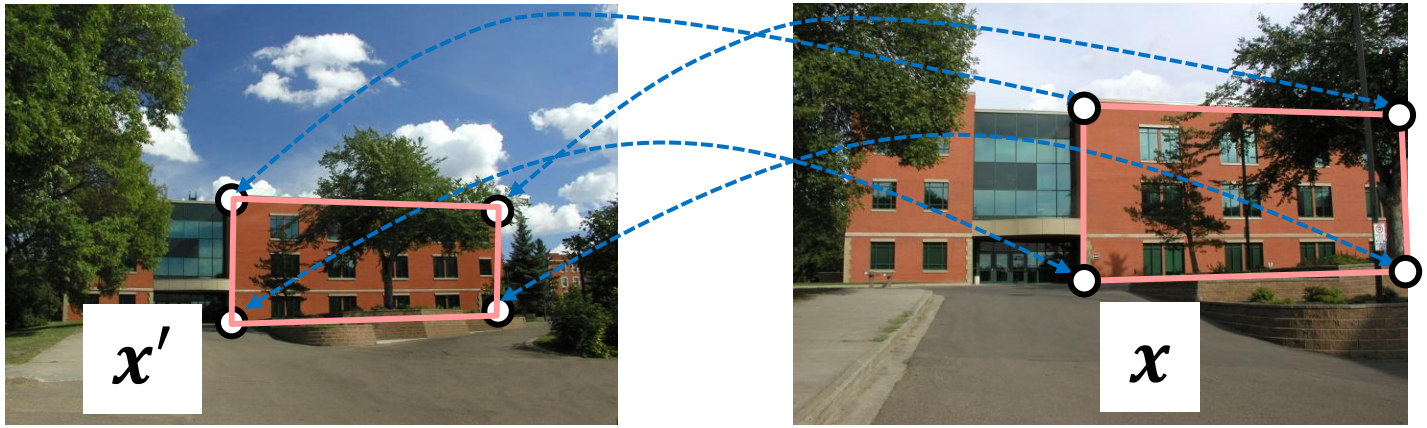



- A plane-to-plane projection is called a *Homography*

# Apps: Panoramas, Augmented reality, etc.

# Homography estimation from correspondences

- Example of four corresponding points




$$wx' = Hx$$

$$w\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

- The elements of $H$ can be estimated by applying a direct linear transform (DLT)!

# Matrix form of a vector product

- Before we continue…

$$\mathbf{c} = \mathbf{a} \times \mathbf{b} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} \equiv \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix}$$

$$\mathbf{c}^T \mathbf{a} = 0$$

$$\mathbf{c}^T \mathbf{b} = 0$$

$$\left[\mathbf{a}_\times\right] \equiv \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix}$$

$$\mathbf{a} \times \mathbf{b} = \left[\mathbf{a}_\times\right] \mathbf{b}$$

# Homography estimation by DLT

$$w\mathbf{x}'_i = \mathbf{H}\mathbf{x}_i$$

$$w\begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & H_{13} \\ H_{21} & H_{22} & H_{23} \\ H_{31} & H_{32} & H_{33} \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} \boldsymbol{h}_1^T \\ \boldsymbol{h}_2^T \\ \boldsymbol{h}_3^T \end{bmatrix} \boldsymbol{x}_i = \begin{bmatrix} \boldsymbol{h}_1^T \boldsymbol{x}_i \\ \boldsymbol{h}_2^T \boldsymbol{x}_i \\ \boldsymbol{h}_3^T \boldsymbol{x}_i \end{bmatrix} = \begin{bmatrix} \boldsymbol{x}_i^T \boldsymbol{h}_1 \\ \boldsymbol{x}_i^T \boldsymbol{h}_2 \\ \boldsymbol{x}_i^T \boldsymbol{h}_3 \end{bmatrix}$$

$$\mathbf{x}'_i \times \mathbf{H}\mathbf{x}_i = 0$$

Change the vector product into vector-matrix:

$$\mathbf{x}'_i \times \begin{bmatrix} \mathbf{x}_i^T \mathbf{h}_1 \\ \mathbf{x}_i^T \mathbf{h}_2 \\ \mathbf{x}_i^T \mathbf{h}_3 \end{bmatrix} = [\mathbf{x}'_{i\times}] \begin{bmatrix} \mathbf{x}_i^T \mathbf{h}_1 \\ \mathbf{x}_i^T \mathbf{h}_2 \\ \mathbf{x}_i^T \mathbf{h}_3 \end{bmatrix} = \begin{bmatrix} 0 & -1 & y'_i \\ 1 & 0 & -x'_i \\ -y'_i & x'_i & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_i^T \mathbf{h}_1 \\ \mathbf{x}_i^T \mathbf{h}_2 \\ \mathbf{x}_i^T \mathbf{h}_3 \end{bmatrix}$$

# Homography estimation by DLT

$$\mathrm{x}'_i \times \mathrm{Hx}_i = 0$$

Multiply in the matrix terms…

$$\mathbf{x}'_i \times \mathbf{H}\mathbf{x}_i = \begin{bmatrix} 0 & -1 & y'_i \\ 1 & 0 & -x'_i \\ -y'_i & x'_i & 0 \end{bmatrix} \begin{bmatrix} \mathbf{x}_i^T \mathbf{h}_1 \\ \mathbf{x}_i^T \mathbf{h}_2 \\ \mathbf{x}_i^T \mathbf{h}_3 \end{bmatrix} = \begin{bmatrix} -\mathbf{x}_i^T \mathbf{h}_2 + y'_i \mathbf{x}_i^T \mathbf{h}_3 \\ \mathbf{x}_i^T \mathbf{h}_1 - x'_i \mathbf{x}_i^T \mathbf{h}_3 \\ -y'_i \mathbf{x}_i^T \mathbf{h}_1 + x'_i \mathbf{x}_i^T \mathbf{h}_2 \end{bmatrix}$$

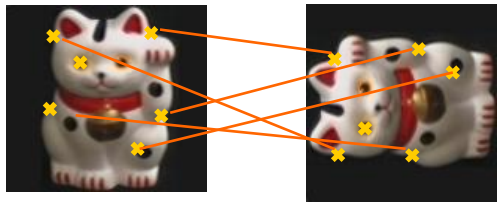
Expose the homography terms $\boldsymbol{h}_1, \boldsymbol{h}_2, \boldsymbol{h}_3$ into a single vector:

$$\mathbf{x}'_i \times \mathbf{H}\mathbf{x}_i = \begin{bmatrix} 0^T & -\mathbf{x}_i^T & y'_i \mathbf{x}_i^T \\ \mathbf{x}_i^T & 0^T & -x'_i \mathbf{x}_i^T \\ -y'_i \mathbf{x}_i^T & x'_i \mathbf{x}_i^T & 0^T \end{bmatrix} \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{bmatrix} = 0$$

A single point contains three coordinates, but gives only two linearly independent equations

# Homography estimation by DLT

*n* Correspondences...



$$\mathbf{x}'_1 \leftrightarrow \mathbf{x}_1$$
$$\mathbf{x}'_2 \leftrightarrow \mathbf{x}_2$$
$$\vdots$$

The *n* points yields a system of equations:

$$\begin{bmatrix} 0^T & -\mathbf{x}'^T_1 & y'_1\,\mathbf{x}^T_1 \\ \mathbf{x}^T_1 & 0^T & -x'_1\,\mathbf{x}^T_1 \\ \cdots & \cdots & \cdots \\ 0^T & -\mathbf{x}^T_n & y'_n\,\mathbf{x}^T_n \\ \mathbf{x}^T_n & 0^T & -x'_n\,\mathbf{x}^T_n \end{bmatrix} \begin{pmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \\ \mathbf{h}_3 \end{pmatrix} = 0$$

Homogeneous system!

$$\mathbf{Ah} = \mathbf{0}$$

**SVD**

$$\mathbf{A} = \mathbf{UDV}^T = \mathbf{U} \begin{bmatrix} d_{11} & & \\ & \ddots & \\ & & d_{99} \end{bmatrix} \begin{bmatrix} v_{11} & \cdots & v_{19} \\ \vdots & \ddots & \vdots \\ v_{91} & \cdots & v_{99} \end{bmatrix}^T$$

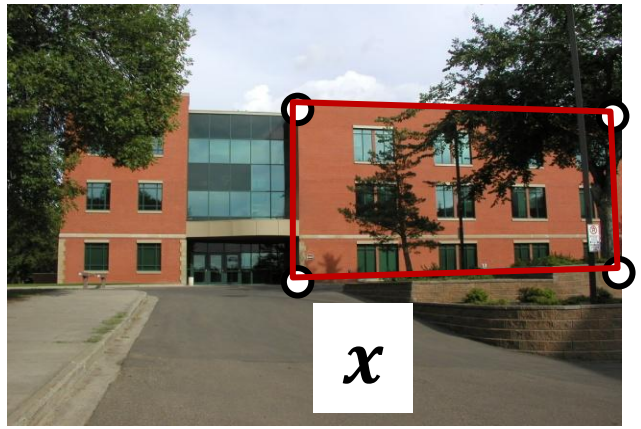
$$\mathbf{h} = \frac{[v_{19}, \cdots, v_{99}]^T}{v_{99}}$$

Minimizes the mean squared error.

Reshape **h** into **H**.

# Preconditioning

- DLT works well if the corresponding points are normalized separately in each view!
- Transformation $T_{pre}$:
  - Subtract the average
  - Scale to average distance 1.




$$\tilde{\mathrm{x}}=\mathrm{T}_{pre}\mathrm{x}$$

$$\mathrm{T}_{\mathrm{pre}}=\begin{bmatrix} a & 0 & c \\ 0 & b & d \\ 0 & 0 & 1 \end{bmatrix}$$

- Set [a,b,c,d] such that the mean of the points $\tilde{\boldsymbol{x}}_i$ is zero and their variance is 1.

# Homography estimation

1. Apply preconditioning (i.e., multiply by the transform matrices) to points in each image separately:

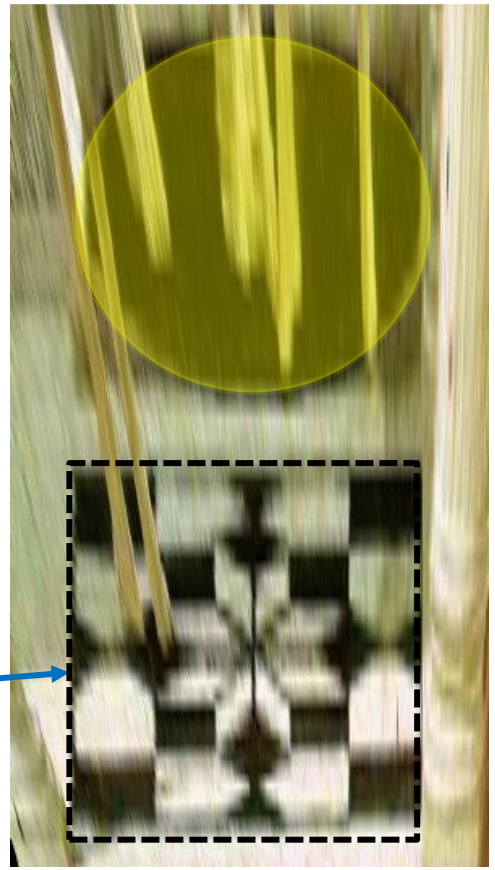
$$\tilde{\mathbf{x}}' = \mathbf{T}'_{pre}\mathbf{x}' \qquad \tilde{\mathbf{x}} = \mathbf{T}_{pre}\mathbf{x}$$

2. Apply DLT to estimate the homography $\tilde{H}$: $\tilde{\mathbf{x}}' = \tilde{H}\tilde{\mathbf{x}}$
3. Transform back the solution to remove preconditioning: $\mathbf{H} = \mathbf{T}'^{-1}_{pre}\tilde{\mathbf{H}}\mathbf{T}_{pre}$

# Secret knowledge

*Flagellation of Christ* (Piero della Francesca, ~1460)

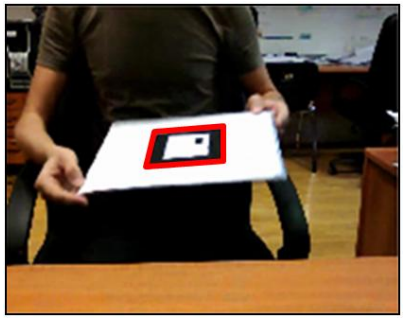
Compute a homography to this rectangle

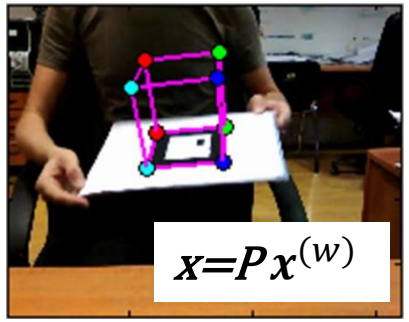


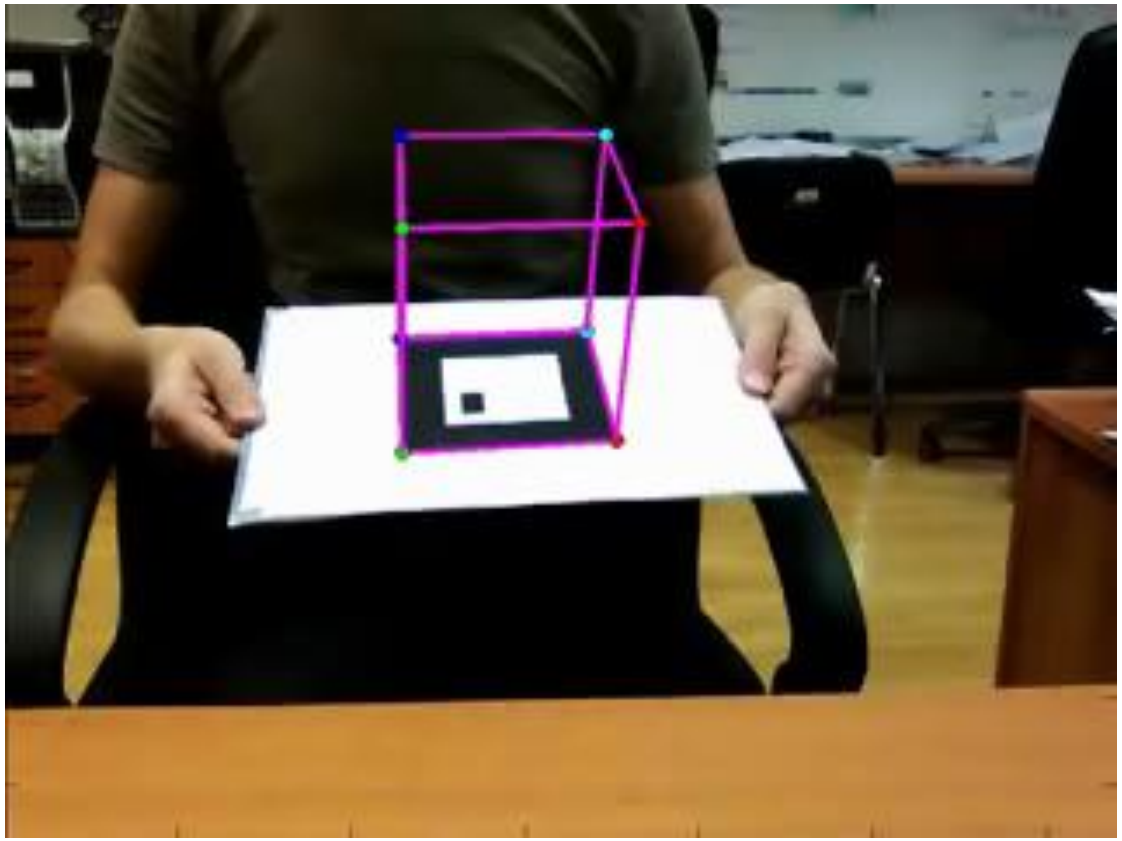
Zoom-in of the floor

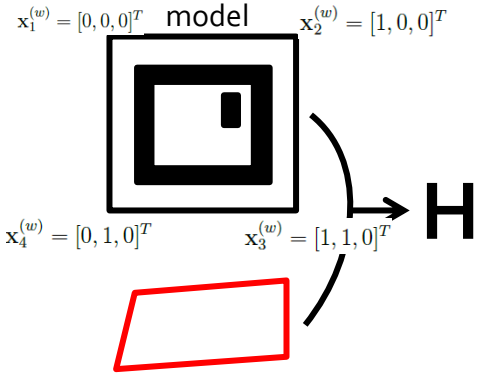
Check out : Secret Knowledge by David Hockney, 2002

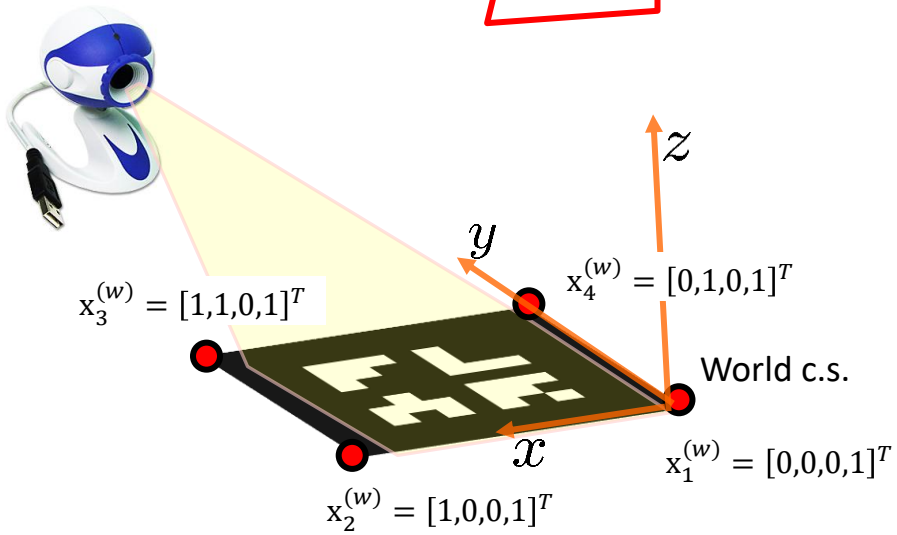
# Marker-based Augmented Reality








$$P = K[r_1, r_2, r_3, t]$$

$$x = K[r_1, r_2, r_3, t]x^{(w)} = K[r_1, r_2, t]x^{(w)\prime}$$

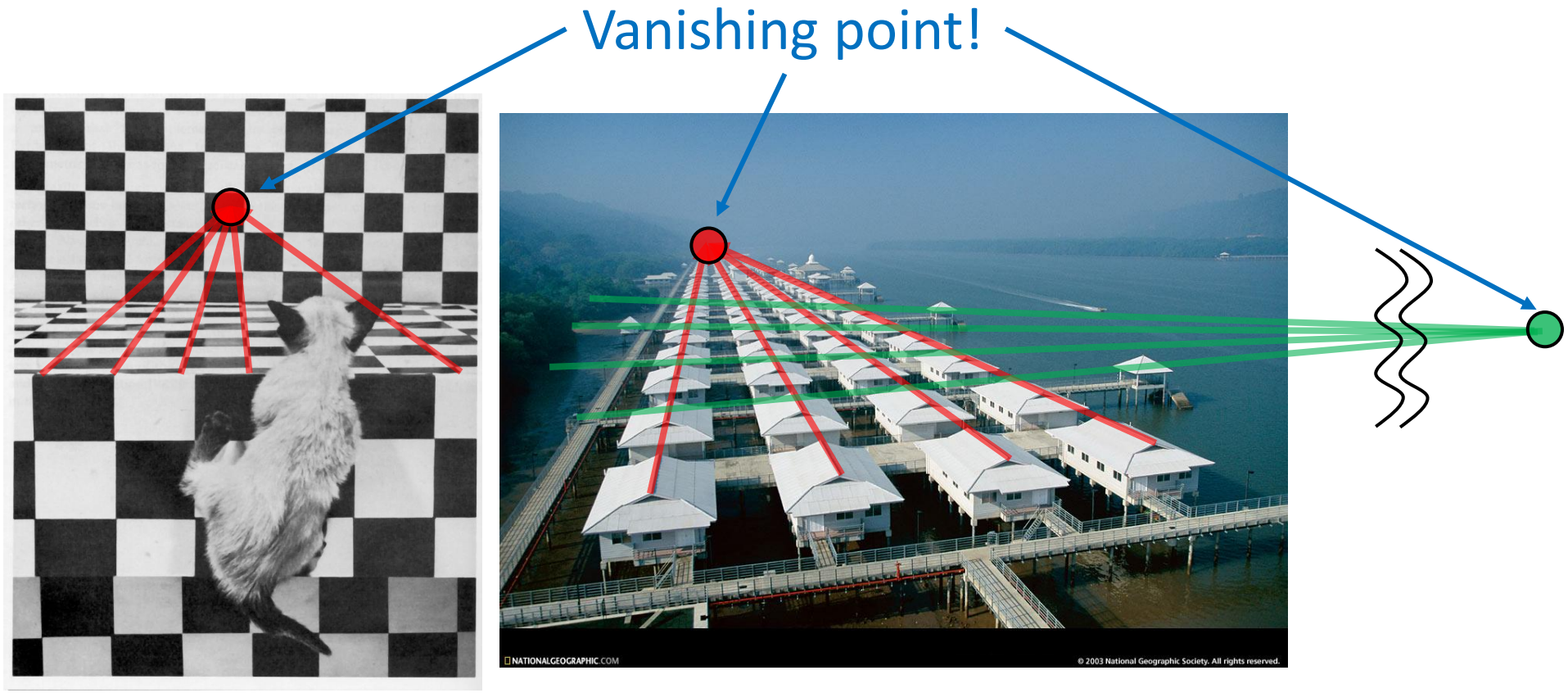
$$x = Hx^{(w)\prime}$$

Note, $x^{(w)\prime}$ are only (x,y) coordinates of $x^{(w)}$ without z=0!

e.g., $x_4^{(w)\prime} = [0, 1, 1]^T$ (the last 1 is due to homogeneus coordinates)

# Vanishing points

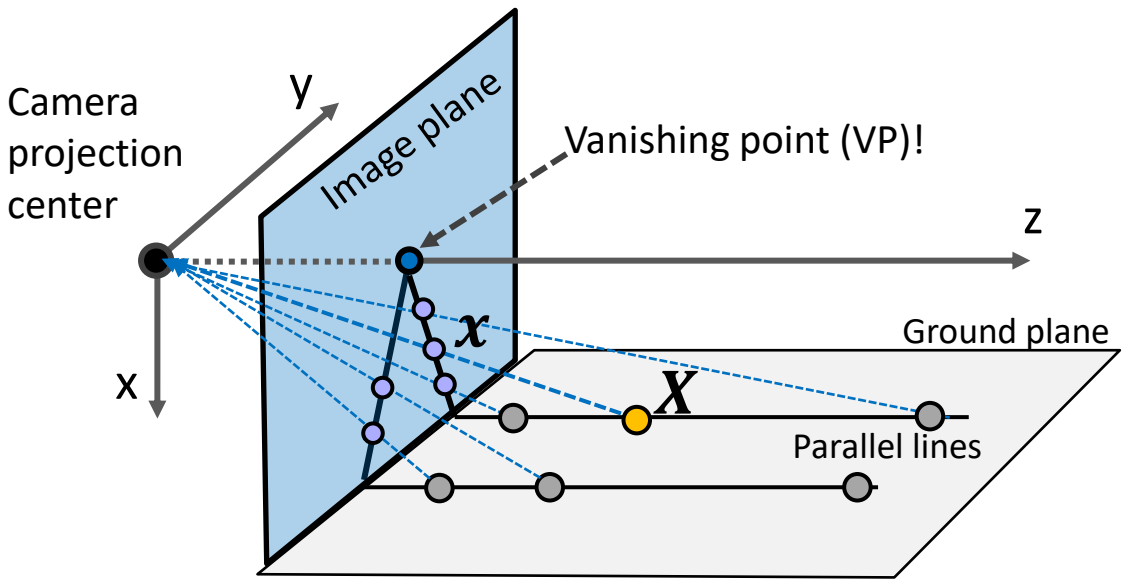
- What happens with projection of parallel lines?




- Sets of 3D parallel lines intersect at a vanishing point!

# Vanishing points

- Where in image do sets of 3D parallel lines *projections* intersect?




A 3D point:

$$\mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}$$

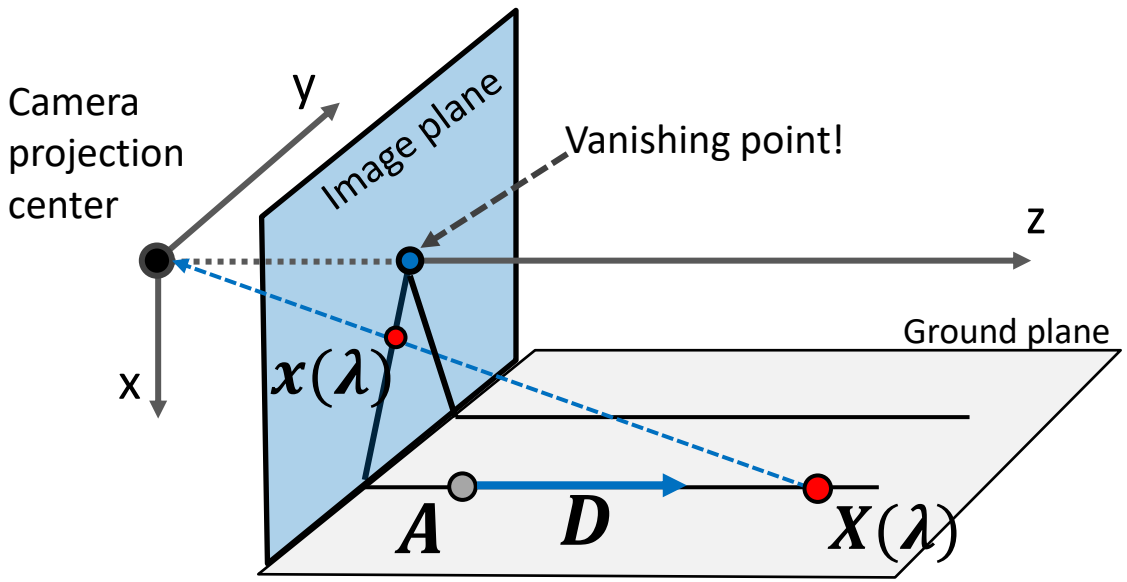
Perspective projection:

$$\mathbf{x} = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \end{bmatrix}$$

- Note that this image shows a special case with lines parallel with principal axis.
- But our derivation of VP will be general.

# Vanishing point: calculation (1/2)

- Consider a point on one of parallel lines




A 3D point **A** and vector **D**:

$$\mathbf{A} = \begin{bmatrix} X_A \\ Y_A \\ Z_A \end{bmatrix} \quad D = \begin{bmatrix} X_D \\ Y_D \\ Z_D \end{bmatrix}$$

A point on a line:

$$\mathbf{X}(\lambda) = \mathbf{A} + \lambda\mathbf{D}$$

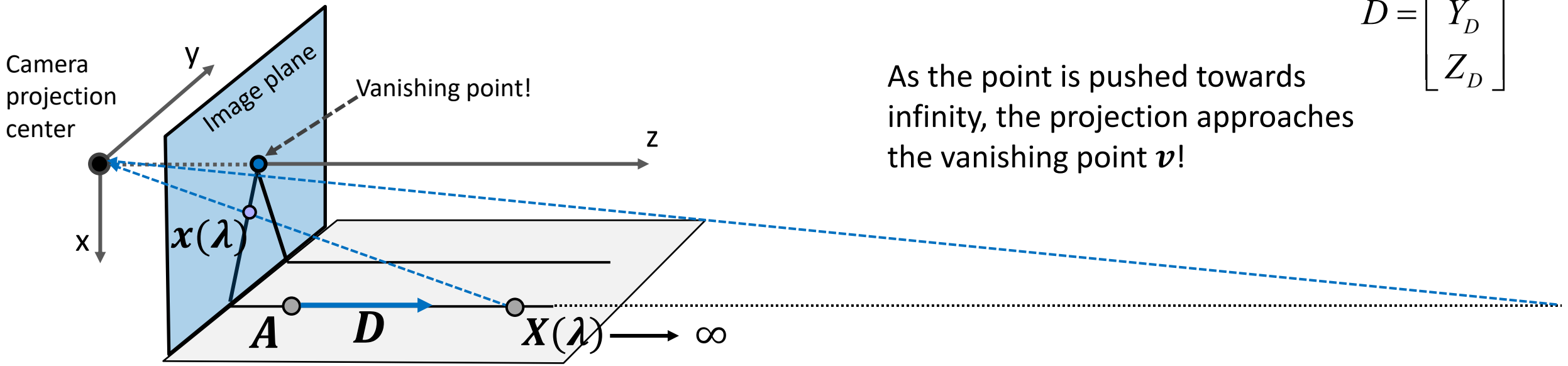
Perspective projection:

$$\mathbf{x}(\lambda) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} fX/Z \\ fY/Z \end{bmatrix} = \begin{bmatrix} \frac{f(X_A+\lambda X_D)}{(Z_A+\lambda Z_D)} \\ \frac{f(Y_A+\lambda Y_D)}{(Z_A+\lambda Z_D)} \end{bmatrix}$$

# Vanishing point: calculation (2/2)

- Now push the point far away from the camera…

$$D = \begin{bmatrix} X_D \\ Y_D \\ Z_D \end{bmatrix}$$




As the point is pushed towards infinity, the projection approaches the vanishing point $\boldsymbol{v}$!

Projection of a point at infinity, i.e., $\boldsymbol{X}(\infty)$:

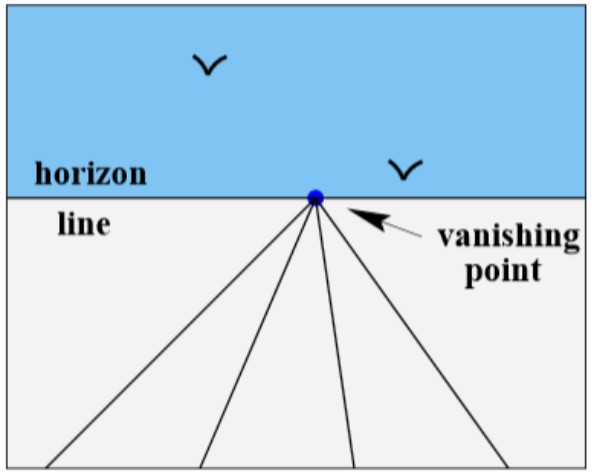
$$\mathbf{v} = \lim_{\lambda \to \infty} x(\lambda) = \lim_{\lambda \to \infty} \begin{bmatrix} f \dfrac{X_A + \lambda X_D}{Z_A + \lambda Z_D} \\ f \dfrac{Y_A + \lambda Y_D}{Z_A + \lambda Z_D} \end{bmatrix} \Rightarrow$$

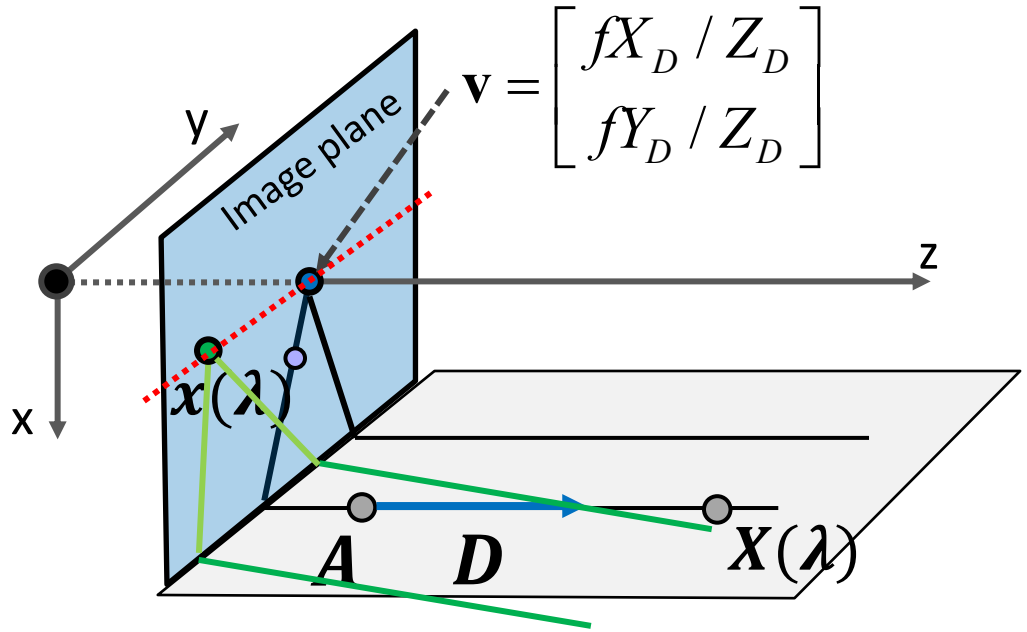
Vanishing point!

$$\mathbf{v} = \begin{bmatrix} fX_D / Z_D \\ fY_D / Z_D \end{bmatrix}$$

# Vanishing points

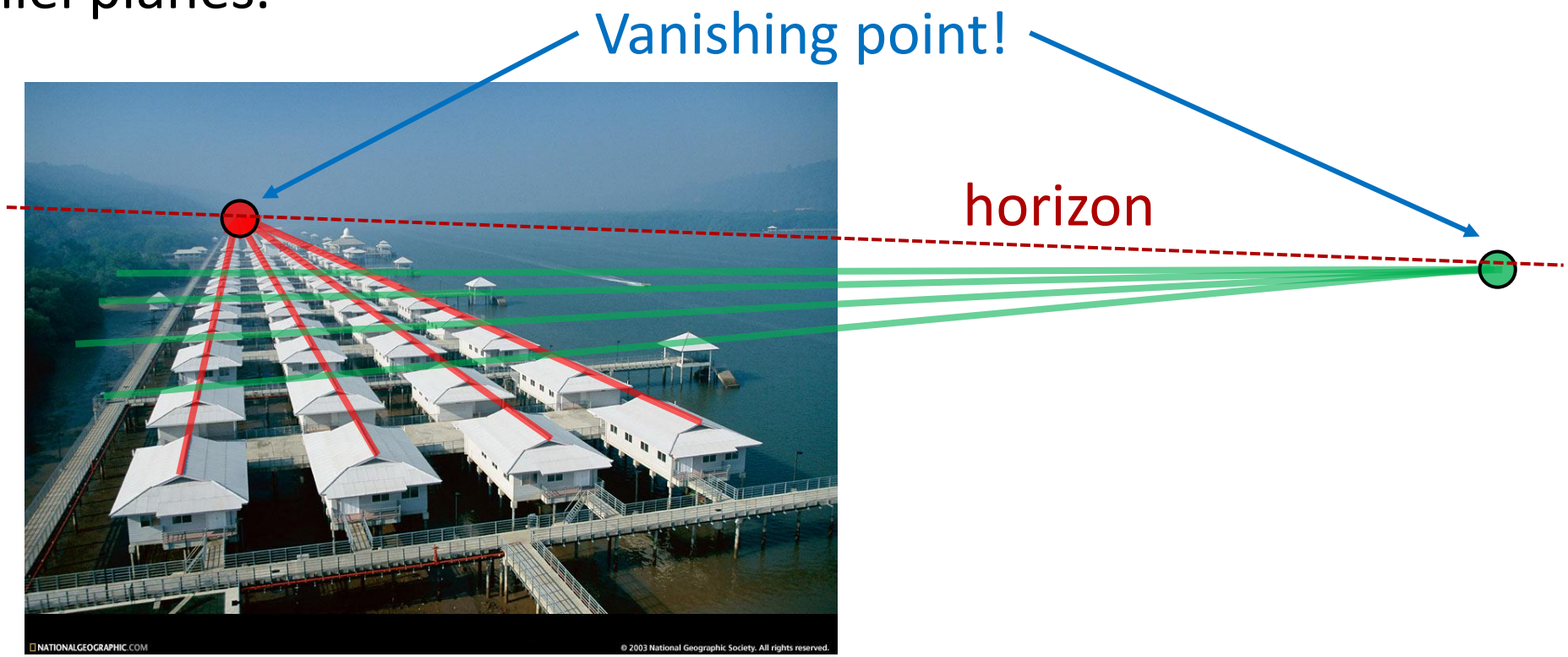
- VP depends on direction $\boldsymbol{D}$, not on point $\boldsymbol{A}$.
- A different set of parallel lines correspond to a different VP!
- Horizon is formed by connecting the vanishing points of a plane



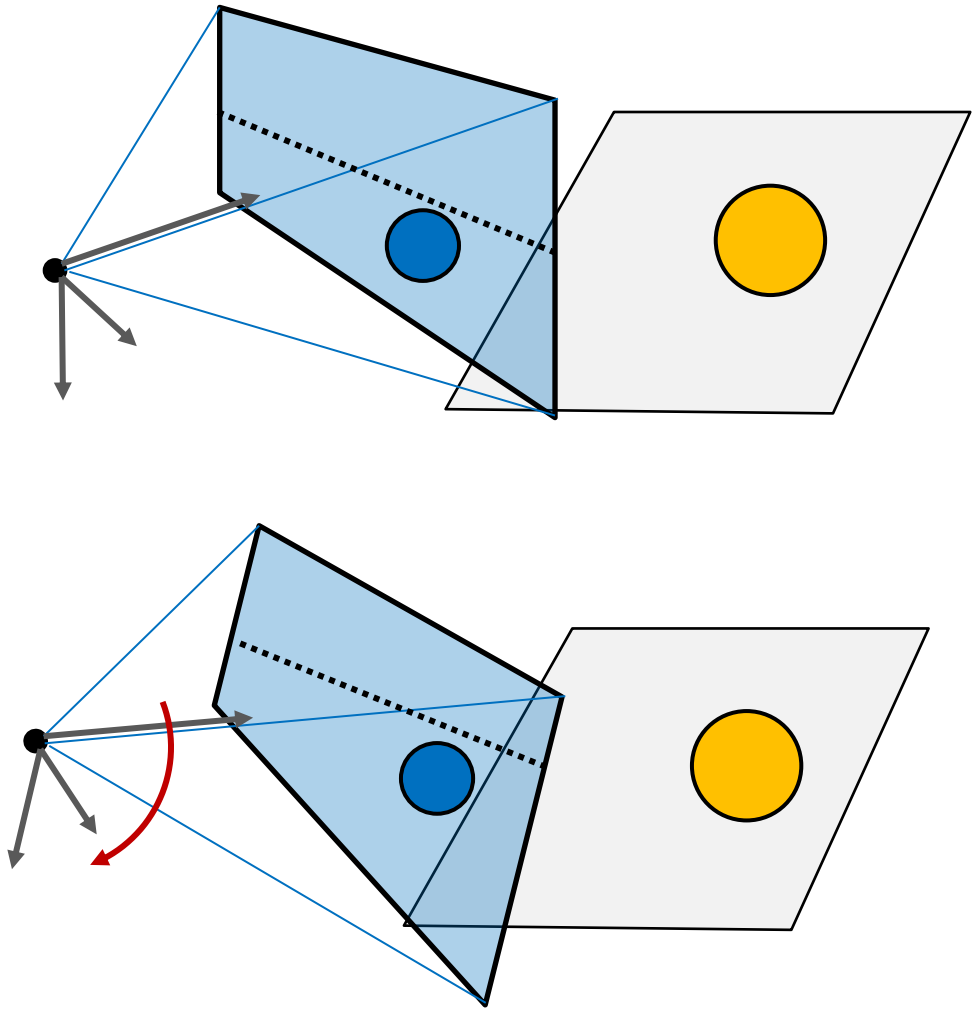
# Vanishing points

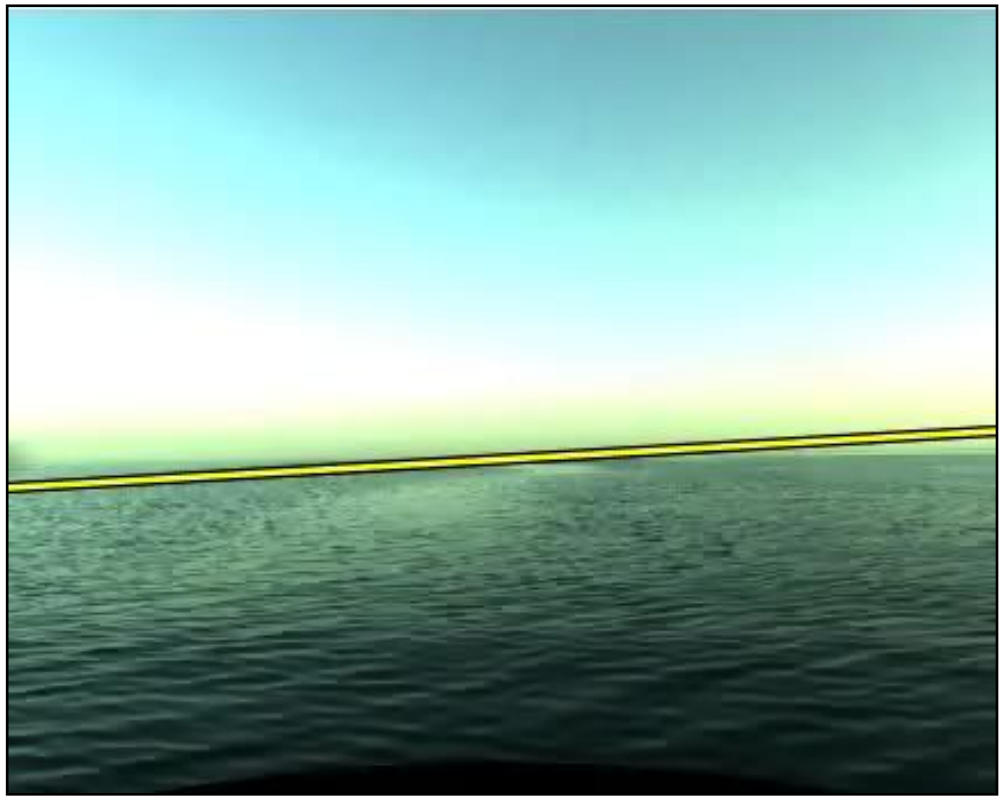
- Horizon is a collection of all the vanishing points corresponding to a set of parallel planes.




- Sets of 3D parallel lines intersect at a vanishing point!

# Example: Use IMU to estimate horizon projection

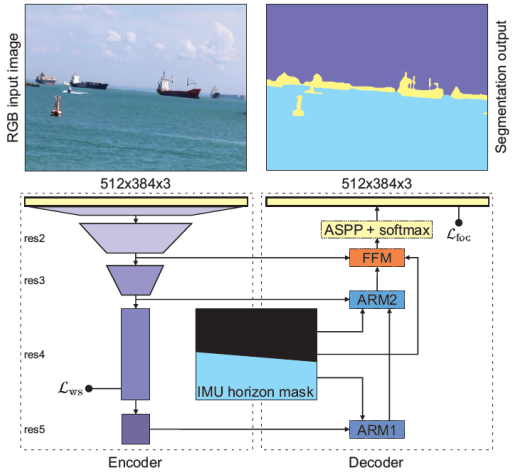


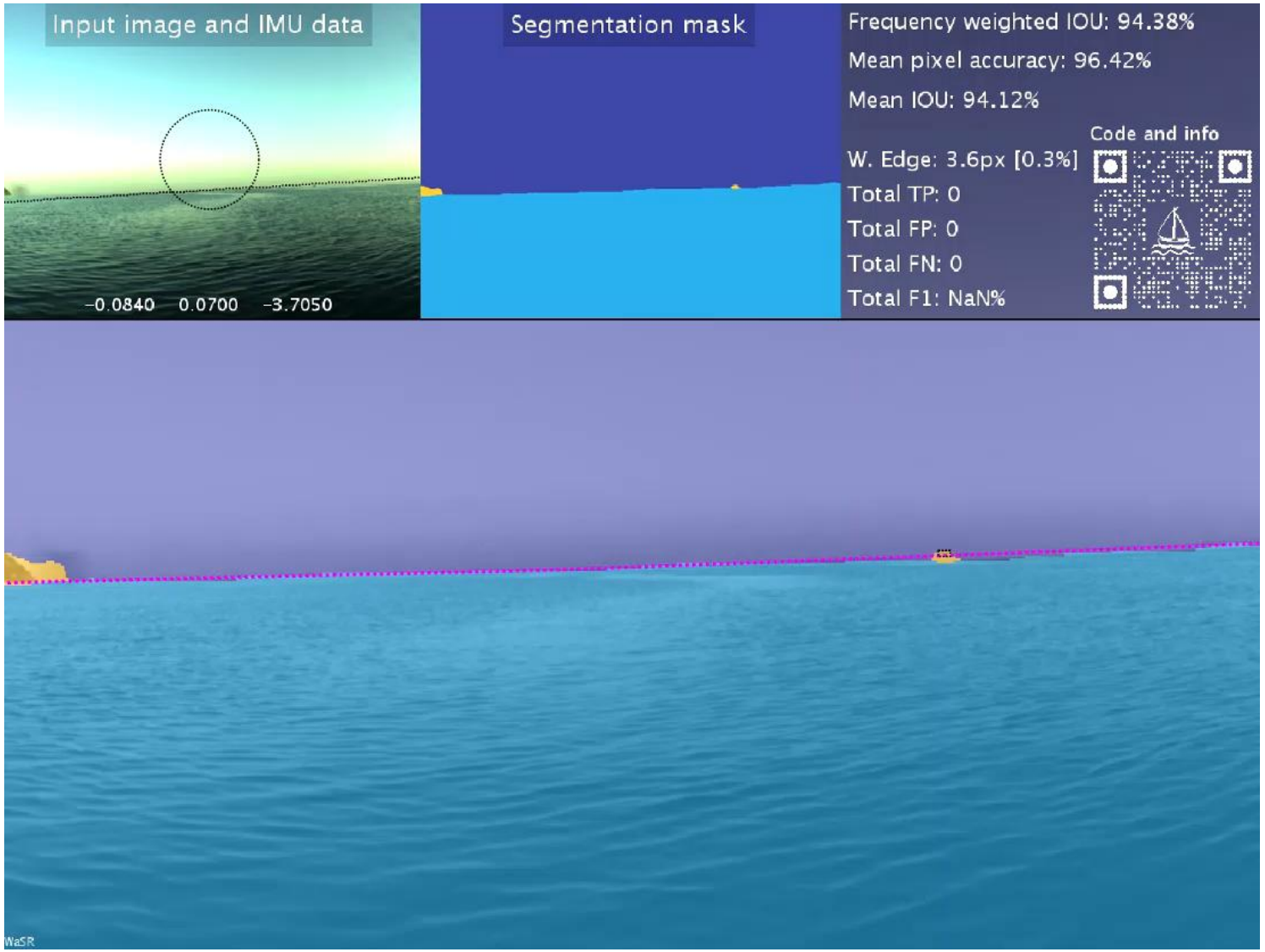
Camera tilt estimated from IMU, horizon projected into image

Bovcon, Perš, Mandeljc, Kristan, Stereo Obstacle Detection for Unmanned Surface Vehicles by IMU-assisted Semantic Segmentation, RAS 2018

# Example: Use IMU for obstacle detection

WaSR: Water Separation and Refinement Network

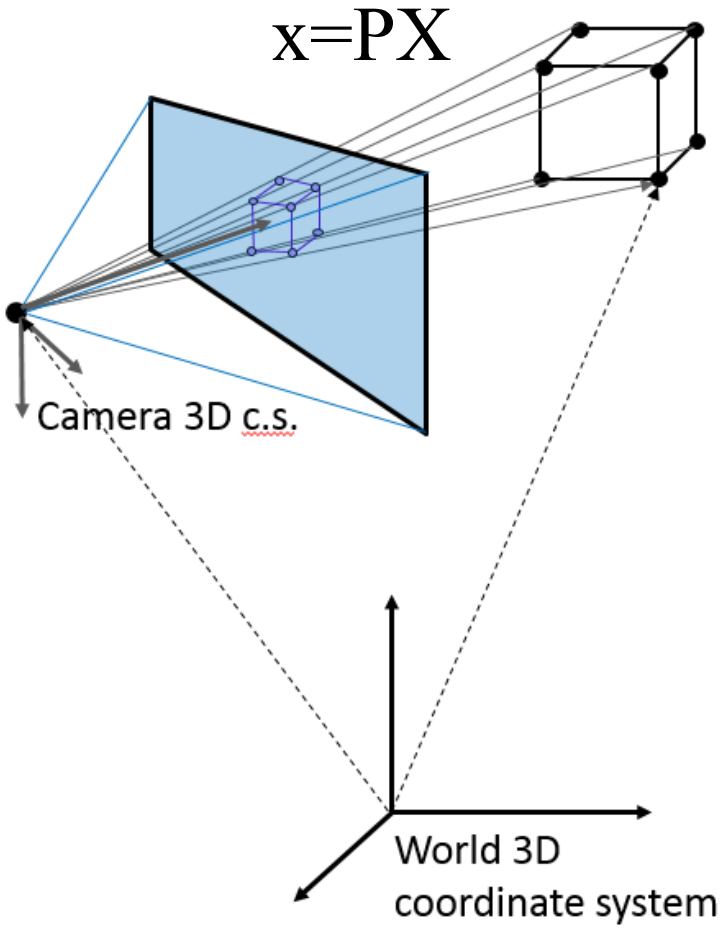


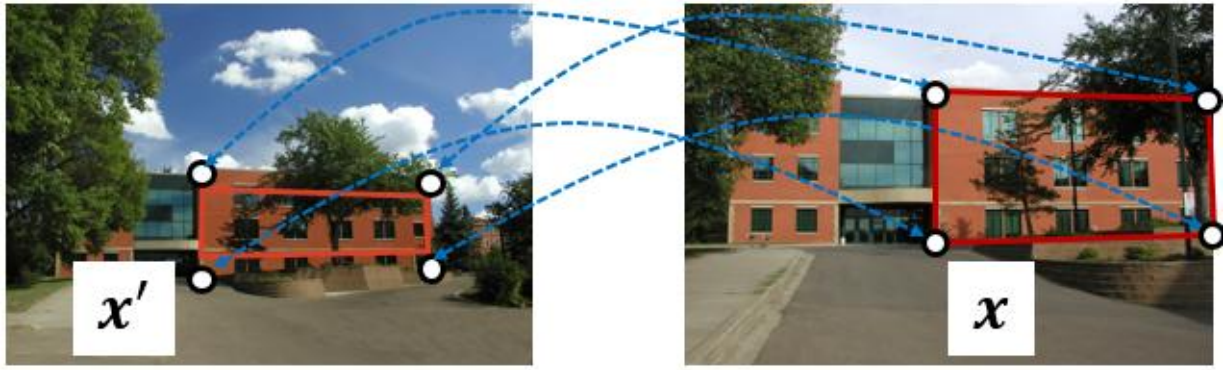

Bovcon, Kristan, A water-obstacle separation and refinement network for unmanned surface vehicles, IEEE TCyb 2021

# Previously at MP…

Single-view geometry:

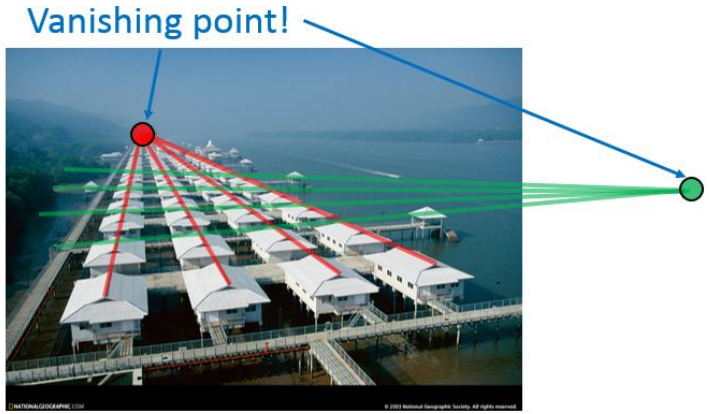


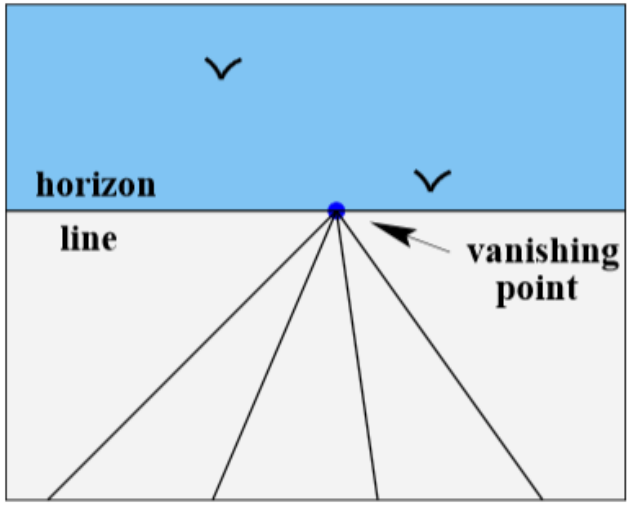

Homography:




$wx' = Hx$ DLT for $H$ estimation.

Vanishing points:



# Camera calibration

- Assume a fixed camera in 3D that you want to use for measuring

$$\lambda \mathrm{x} = P\mathrm{X}$$

What is this distance in $mm$?

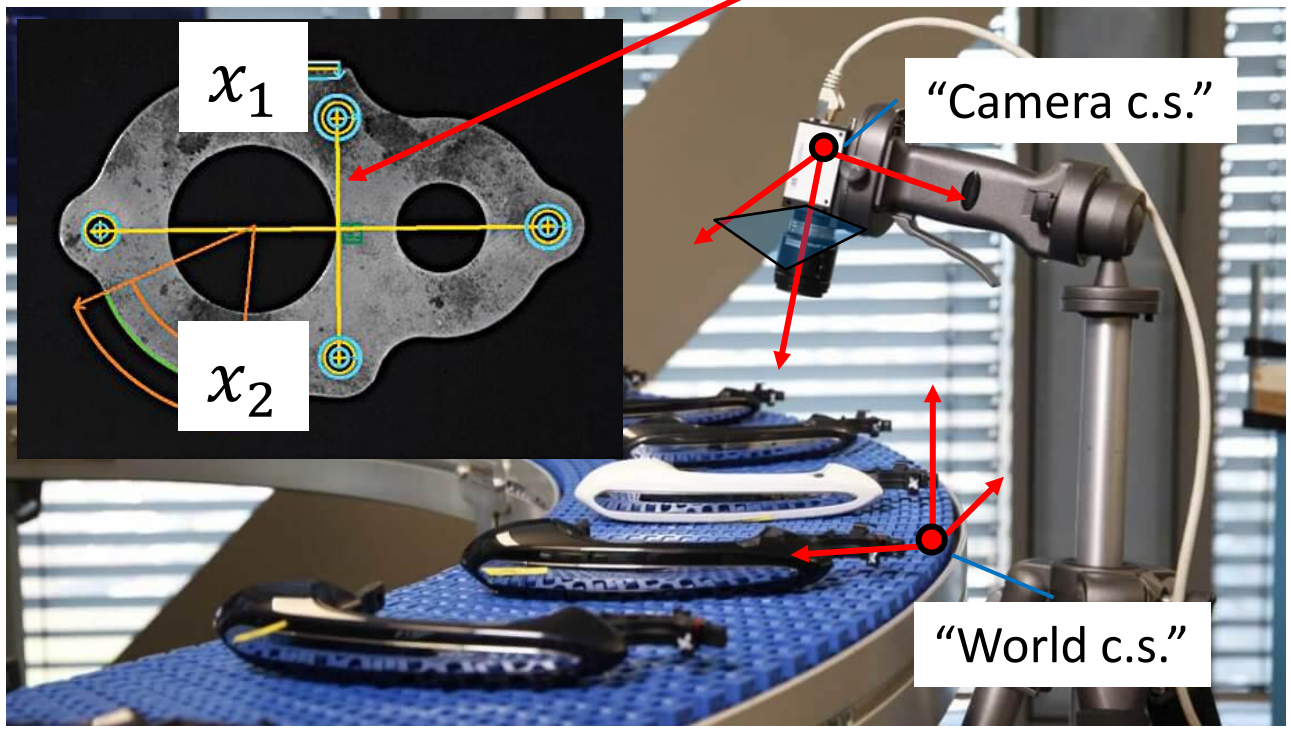
In principle (not really that easy…):

$$X_1 = P^{-1}\mathrm{x}_1, X_2 = P^{-1}\mathrm{x}_2$$

$$d = \| X_1 - X_2 \|$$

What is required to form $P$?

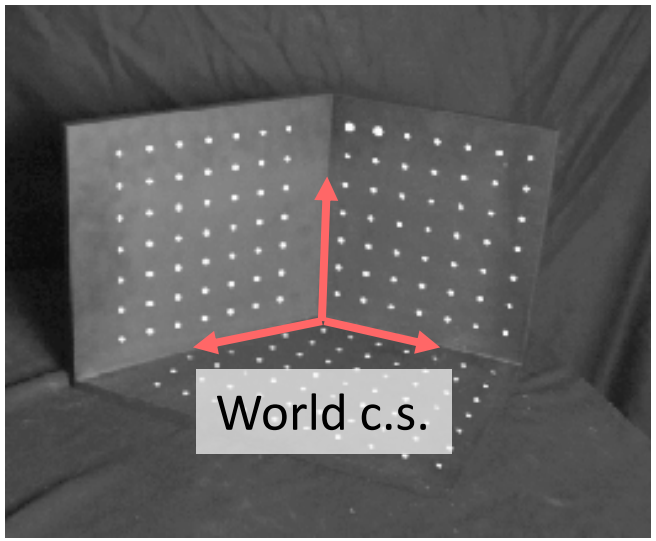
$$\mathrm{x} = \mathrm{K}\left[\mathrm{R} \mid -\mathrm{R}\tilde{\mathrm{C}}\right]\mathrm{X}$$

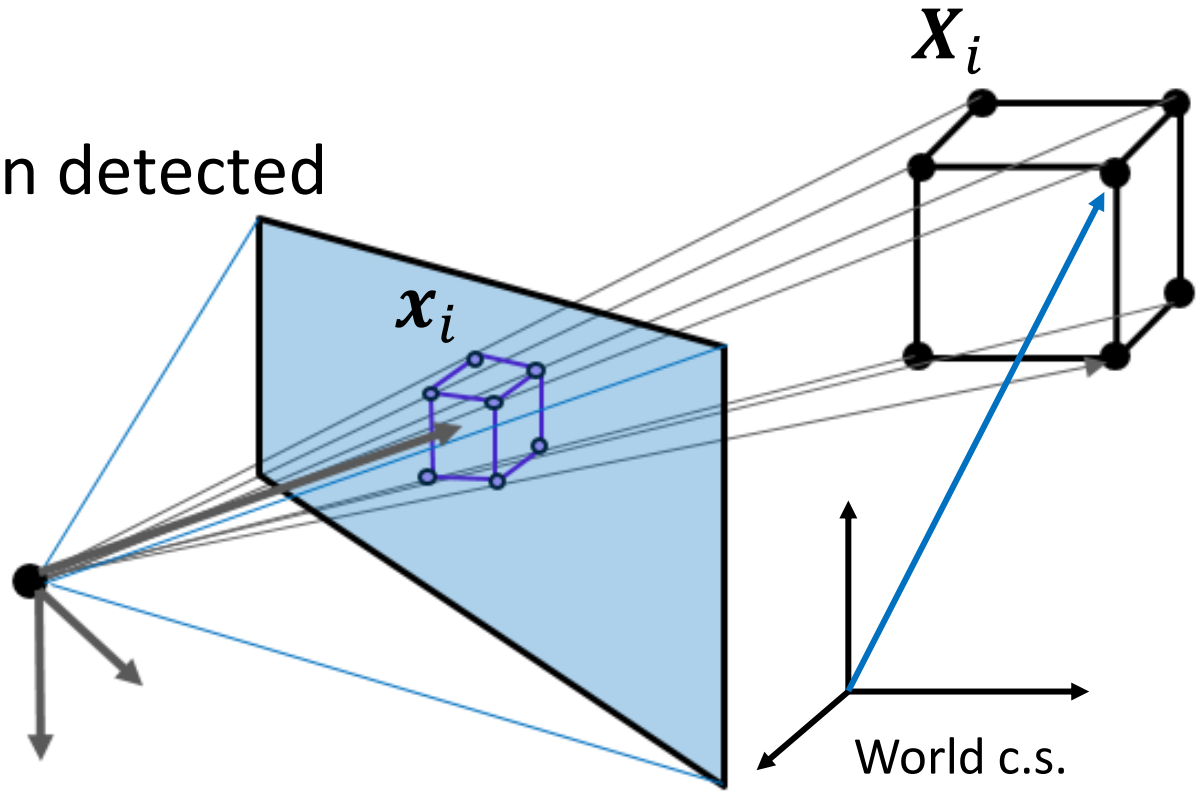



https://vizworld.com/2017/04/watsons-cognitive-visual-inspection-in-lean-manufacturing-processes/

# Camera calibration

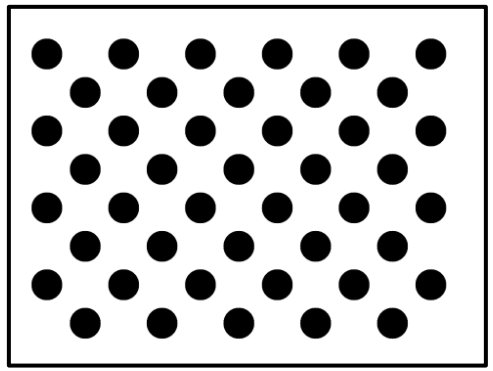
- Camera calibration: *estimate projection matrix* $\boldsymbol{P}$ *from a known calibration object.*
- Corner structures on calibration object for easy and accurate detection
- Coordinates (meters) in 3D known
- Coordinates (pixels) in 2D projection detected



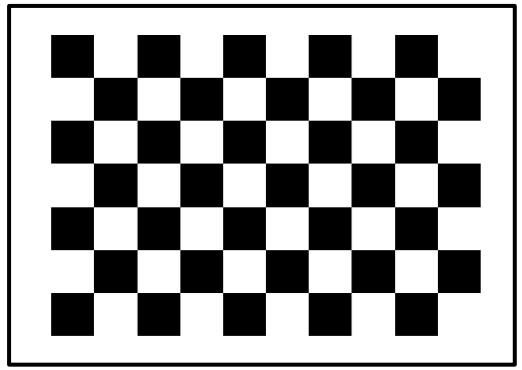



# Camera calibration: point detection

- Proper calibration requires measuring the points at sub-pixel accuracy.
- Highly depends on the calibration pattern.



Gives better results



- How many point correspondences are required?
- A rule of thumb:
  - Number of constraints exceeds the number of unknowns by a factor 5.
  - $\Rightarrow$ For 11 parameters in P, use at least 28 points (2 eqs. per point pair).

# Camera calibration by DLT

- Standard approach for parameter estimation (DLT)

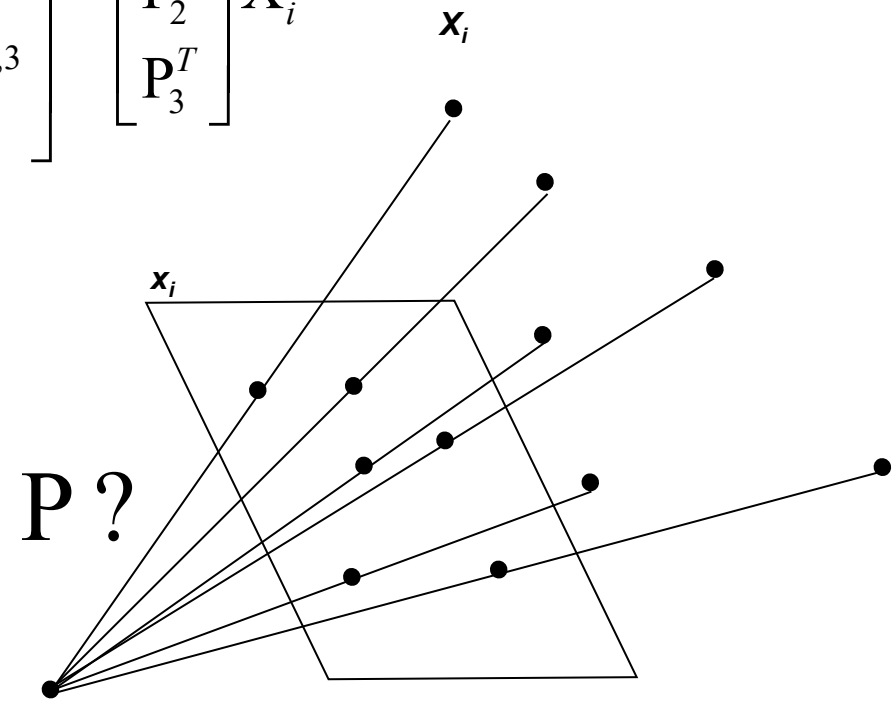
$$\lambda \mathrm{x}_i = \mathrm{P}\mathrm{X}_i$$

$$\lambda \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} P_{11} & P_{12} & P_{13} & P_{14} \\ P_{21} & P_{22} & P_{23} & P_{24} \\ P_{31} & P_{32} & P_{33} & P_{34} \end{bmatrix} \begin{bmatrix} \mathrm{X}_{i,1} \\ \mathrm{X}_{i,2} \\ \mathrm{X}_{i,3} \\ 1 \end{bmatrix} = \begin{bmatrix} \mathrm{P}_1^T \\ \mathrm{P}_2^T \\ \mathrm{P}_3^T \end{bmatrix} \mathrm{X}_i$$

$$\mathrm{x}_i \times \mathrm{P}\mathrm{X}_i = 0$$

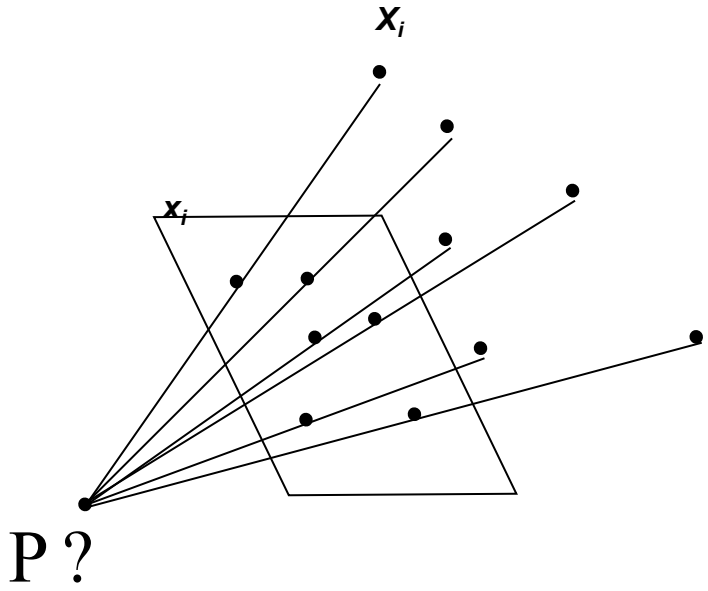
Same approach as with Homography:

$$\begin{bmatrix} 0^T & -\mathrm{X}_i^T & y_i \mathrm{X}_i^T \\ \mathrm{X}_i^T & 0^T & -x_i \mathrm{X}_i^T \\ -y_i \mathrm{X}_i^T & x_i \mathrm{X}_i^T & 0^T \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} = 0$$



# Camera calibration by DLT

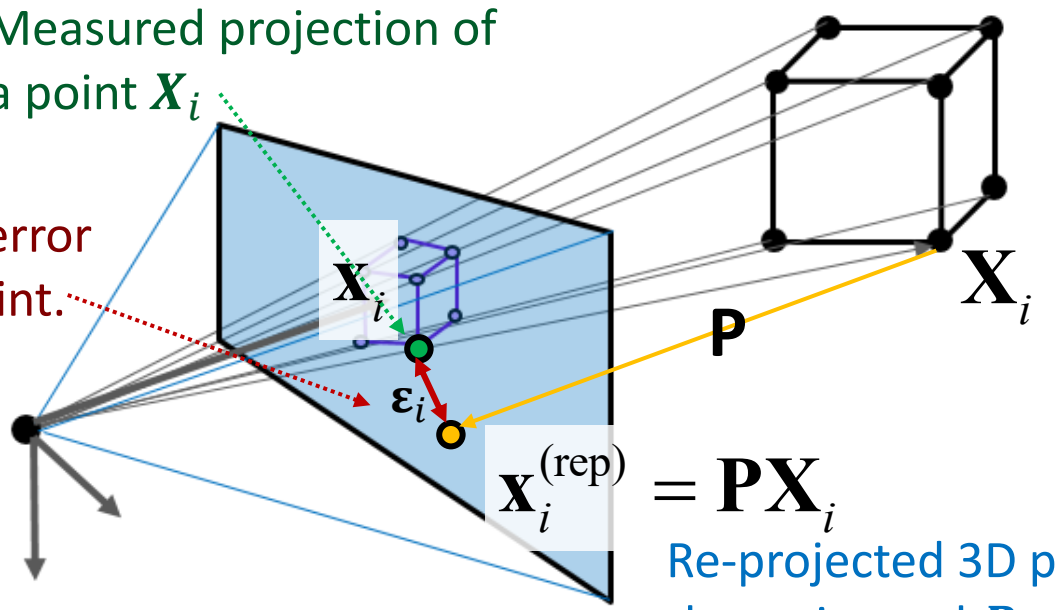
$$\begin{bmatrix} 0^T & \mathrm{X}_1^T & -y_1\mathrm{X}_1^T \\ \mathrm{X}_1^T & 0^T & -x_1\mathrm{X}_1^T \\ \cdots & \cdots & \cdots \\ 0^T & \mathrm{X}_n^T & -y_n\mathrm{X}_n^T \\ \mathrm{X}_n^T & 0^T & -x_n\mathrm{X}_n^T \end{bmatrix} \begin{pmatrix} \mathrm{P}_1 \\ \mathrm{P}_2 \\ \mathrm{P}_3 \end{pmatrix} = 0 \qquad \mathrm{AP} = 0$$




- P has 11 DoF (12 parameters, but the scale is arbitrary).
- A single 2D-3D correspondence gives two linearly independent equations.
- Homogeneous system is solved by SVD of ***A***.
- Solution requires at least 5 ½ correspondences.
- Caution: coplanar points yield degenerate solutions.
- Apply preconditioning as with Homography estimation.

# Camera calibration: practical advices

- The DLT implementation is pretty simple, but it is an algebraic solution.
- In reality we would like to minimize a *re-projection error*:




- The re-projection error:

$$\mathrm{E}(\mathbf{p}) = \sum_{i=1}^{N} \boldsymbol{\varepsilon}_i^T \boldsymbol{\varepsilon}_i$$

# Camera calibration: practical advices

- Nonlinear optimization required (Hartley&Zisserman, Chapter 7.2)
- In practice, initialize by (preconditioned) DLT.
- For practical applications you will need to first remove the radial distortion (H&Z sec. 7.4, or F&P sec. 3.3.).

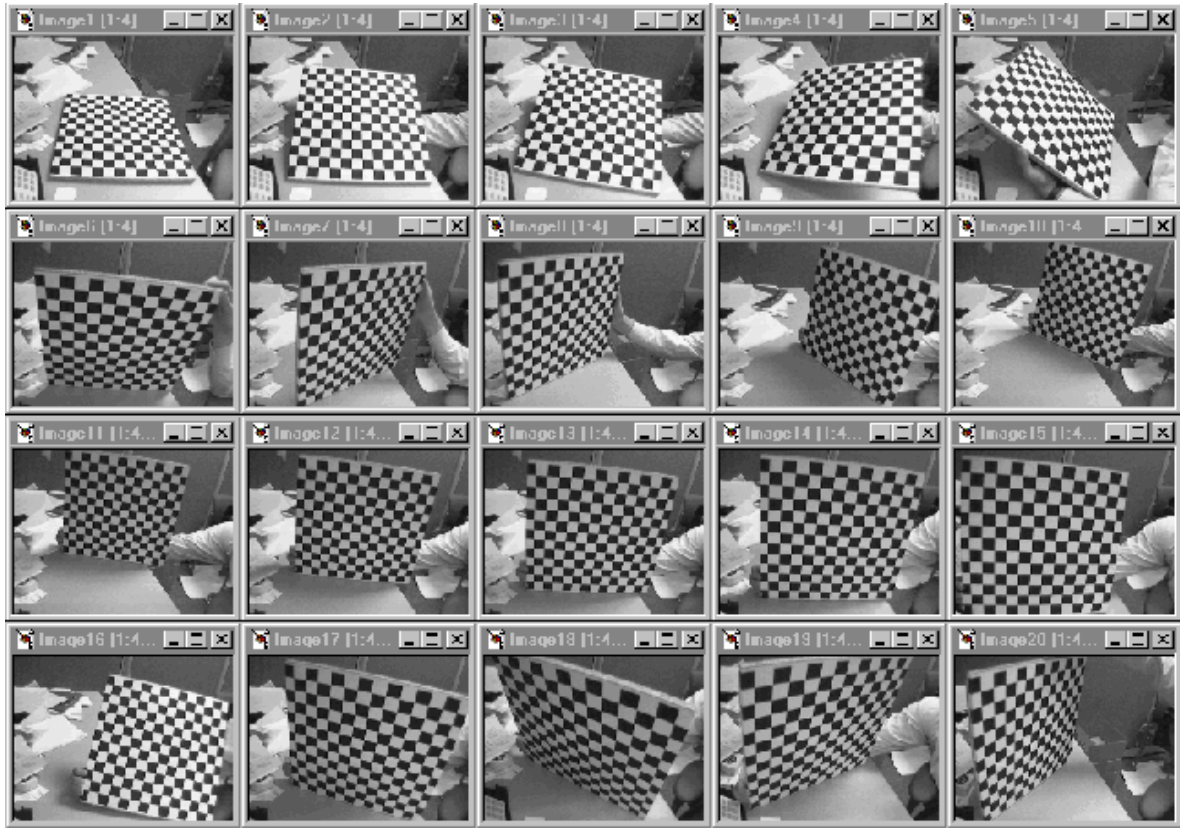
- Fast and accurate approaches for $P$ matrix estimation still an active research topic

# Camera calibration: *P* decomposition

- Once the projection matrix ***P*** is estimated, we need to figure out the camera external and internal parameters, i.e., $P=P_{int}P_{ext}=K[R|t]$.
- This is a matrix decomposition problem.
- Intrinsic and extrinsic matrix have a particular form, that makes such a decomposition possible.
- Solution can be found in Forsyth&Ponce, Chapter 3.2, 3.3. for those who are interested to learn more about camera calibration.

# Standard method: Multiplane camera calibration

- Widely-used approach
- Requires only many images of a single plane
- Does not require knowing positions/orientations



- Solid implementations available online!
  - OpenCV library: http://www.intel.com/research/mrl/research/opencv/
  - Matlab version by Jean-Yves Bouget: http://www.vision.caltech.edu/bouguetj/calib_doc/index.html
  - Zhengyou Zhang's web site: http://research.microsoft.com/~zhang/Calib/

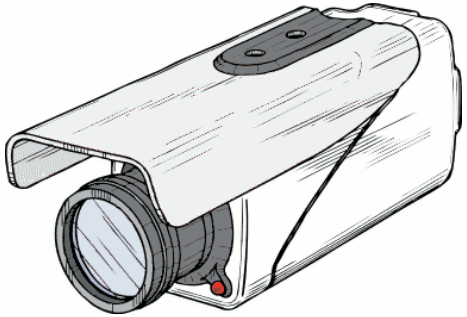
Images courtesy Jean-Yves Bouguet, Intel Corp.

# Thanks.

Univerza v Ljubljani

ViCOS visual cognitive systems lab

# Machine perception
# Multiple-view Geometry



Matej Kristan


Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

# Single-view geometry

- Depth of a point cannot be calculated from a single 2D image

  (without a scene model or other kind of prior information, such as known object sizes)

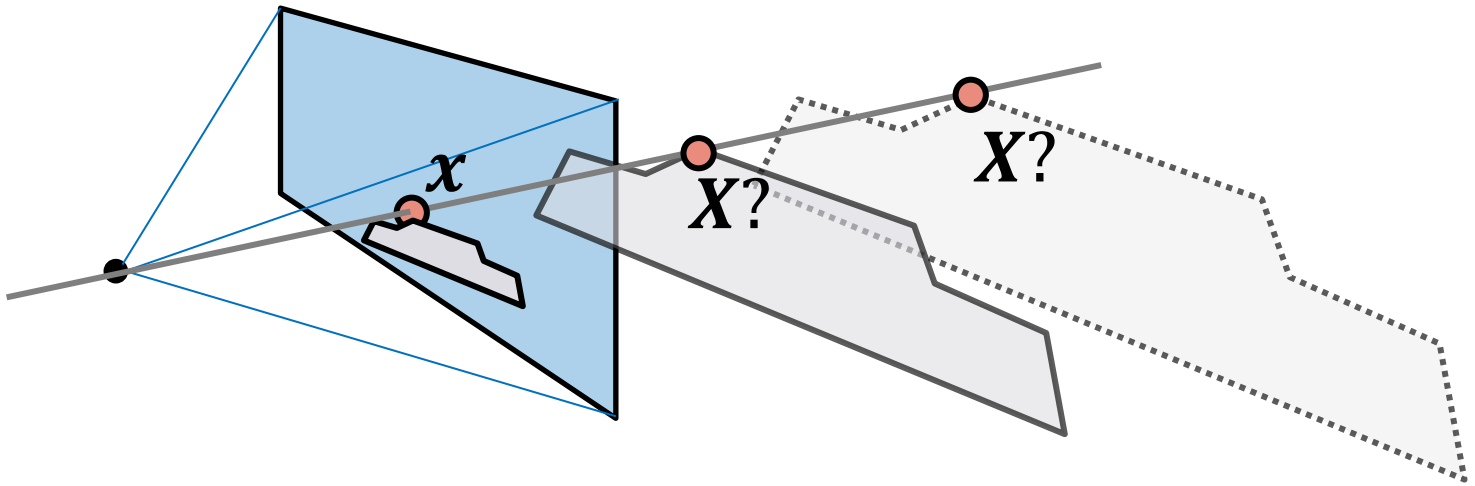
# The reason behind depth ambiguity

- All points along a ray that passes through a camera center are projected into the same point in the image plane.

  $\lambda \mathrm{x}_i = \mathrm{P}\mathrm{X}_i$

- Impossible to calculate a 3D point from a single 2D point

  (without prior on the scene structure, that is)



# Taking advantage of ambiguity

- Anamorphosis (earlier than 15th century)


Author: Julian Beever

# Take two images = Stereo!

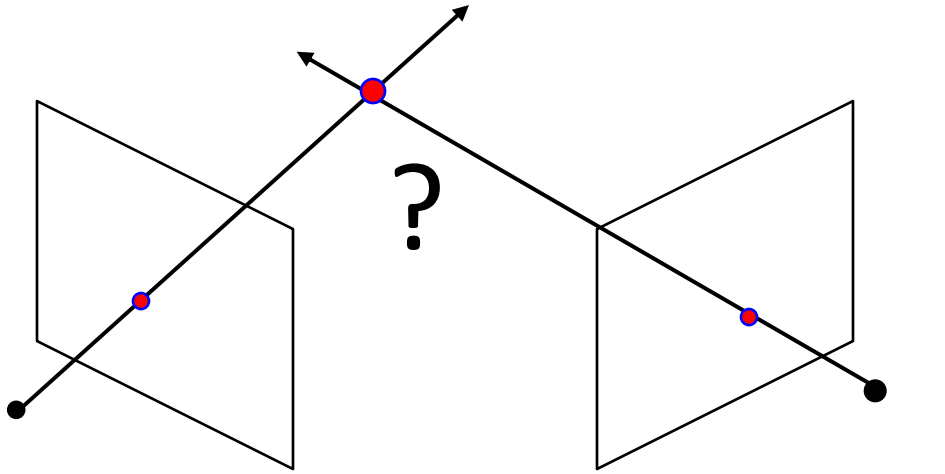
- Much easier using a pair of views...

Photos by: Jim Gasperini

Machine perception

# STEREO GEOMETRY AND SCENE RECONSTRUCTION

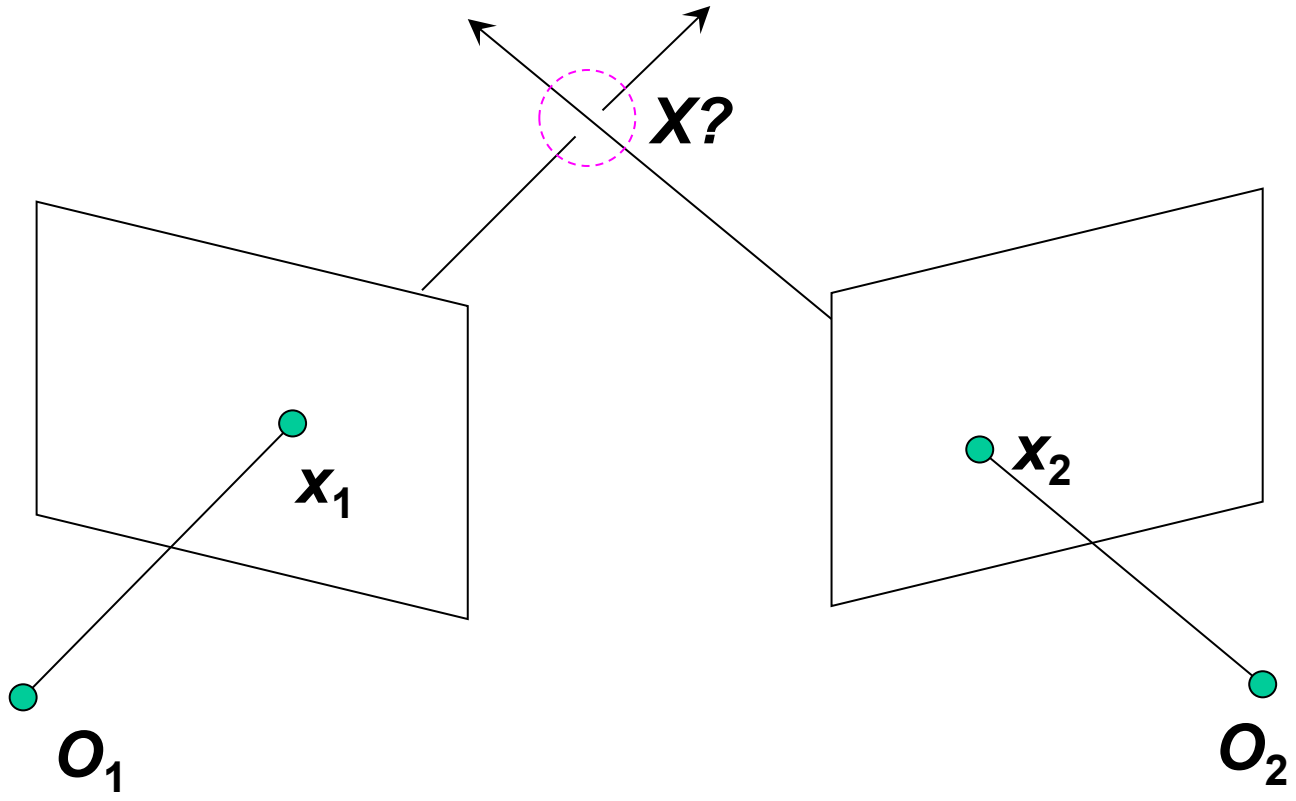
# Depth estimation by triangulation




- The basic principle is triangulation
  - Reconstruction calculated by intersection of two rays
  - Assume:
    - Known camera position in 3D (calibration)
    - Correspondence between points is known

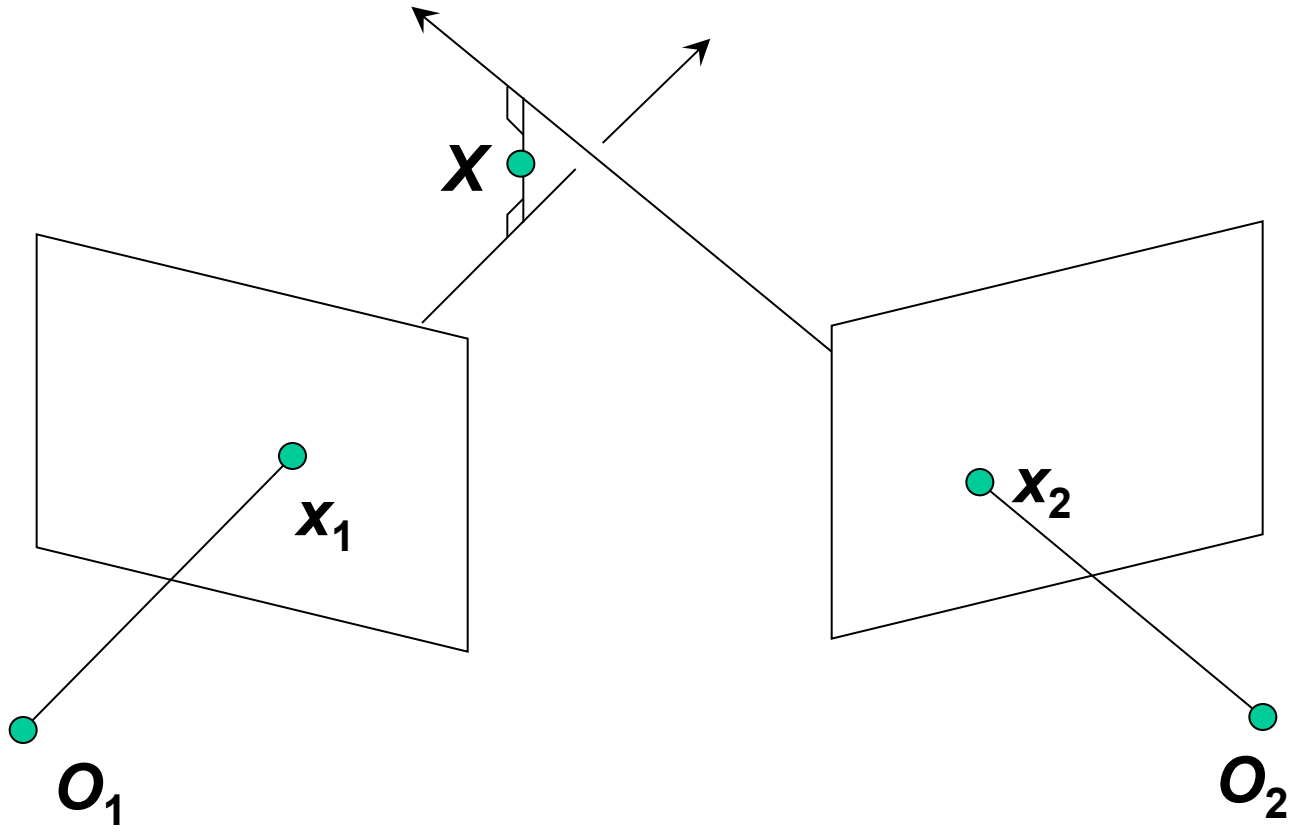
# Triangulation by intersection

- Intersect a pair of visual rays, corresponding to $x_1$ and $x_2$.
- But because of numerical errors and noise, the rays will not intersect in practice!



# Triangulation: Geometric approach

- Find the shortest segment connecting the two rays and take the value $X$ in the middle.
- Not very principled…

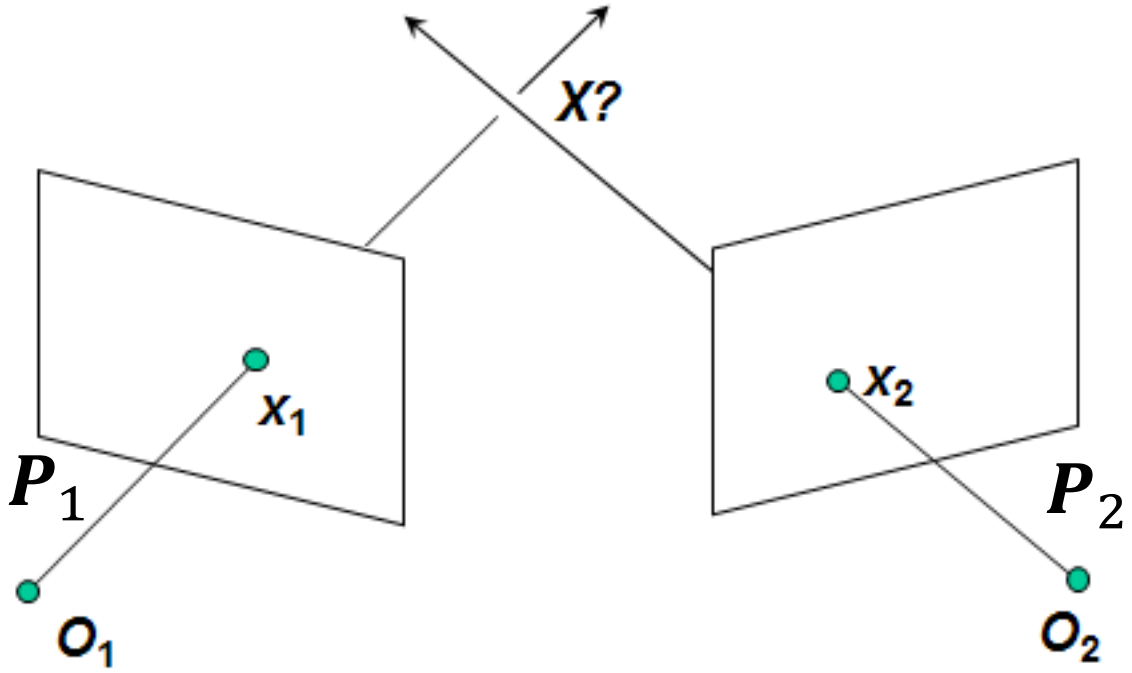


# Triangulation: A linear algebraic approach

$$\lambda_1 \mathrm{x}_1 = \mathrm{P}_1 \mathrm{X} \qquad \mathrm{x}_1 \times \mathrm{P}_1 \mathrm{X} = 0 \qquad [\mathrm{x}_{1\times}]\mathrm{P}_1 \mathrm{X} = 0$$
$$\lambda_2 \mathrm{x}_2 = \mathrm{P}_2 \mathrm{X} \qquad \mathrm{x}_2 \times \mathrm{P}_2 \mathrm{X} = 0 \qquad [\mathrm{x}_{2\times}]\mathrm{P}_2 \mathrm{X} = 0$$

Recall: Vector product written in matrix form:

$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = [\mathbf{a}_\times]\mathbf{b}$$



# Triangulation: Linear algebraic approach

$$\lambda_1 \mathrm{x}_1 = \mathrm{P}_1 \mathrm{X} \qquad \mathrm{x}_1 \times \mathrm{P}_1 \mathrm{X} = 0$$
$$\lambda_2 \mathrm{x}_2 = \mathrm{P}_2 \mathrm{X} \qquad \mathrm{x}_2 \times \mathrm{P}_2 \mathrm{X} = 0$$

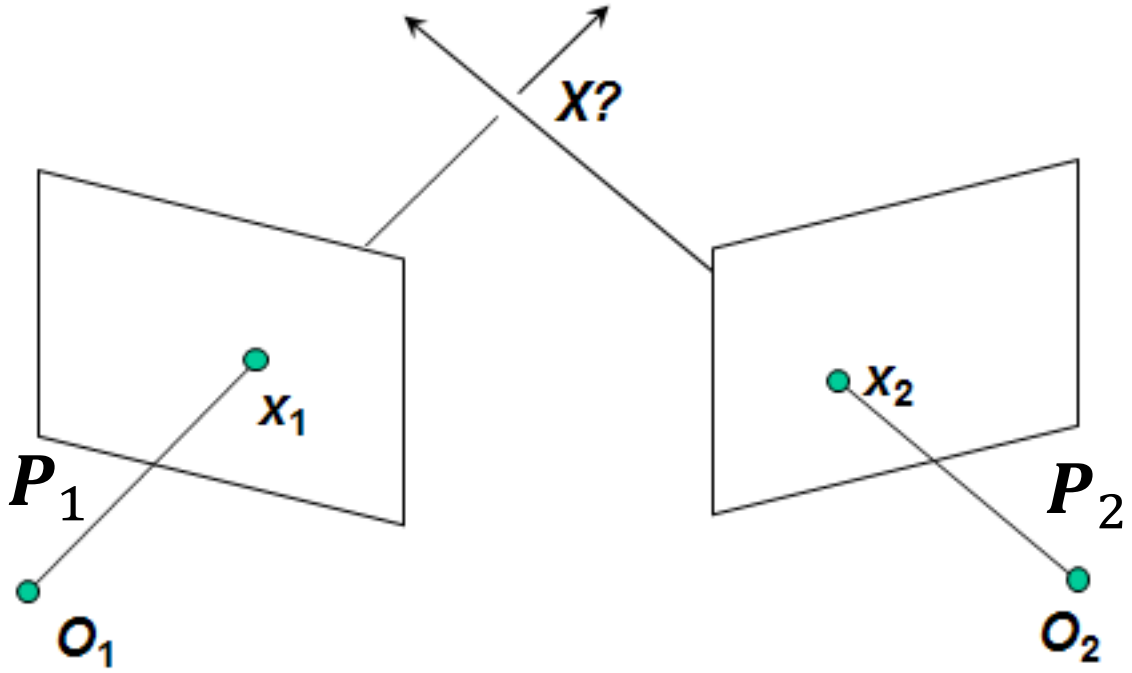
$$[\mathrm{x}_{1\times}]\mathrm{P}_1 \mathrm{X} = 0$$
$$[\mathrm{x}_{2\times}]\mathrm{P}_2 \mathrm{X} = 0$$

Two independent equations each, 3 unknowns in X.

- Write a homogeneous system.

$$\boldsymbol{AX} = \mathbf{0}$$

- Solve by SVD. Solution for $\boldsymbol{X}$ is the eigenvector corresponding to the smallest eigenvalue.
- Better than geometric approach, since it easily generalizes to multiple cameras.



# Triangulation: Nonlinear refinement

- Find $X$ that minimizes a sum of reprojection errors!

$$d^2(x_1, P_1X) + d^2(x_2, P_2X)$$

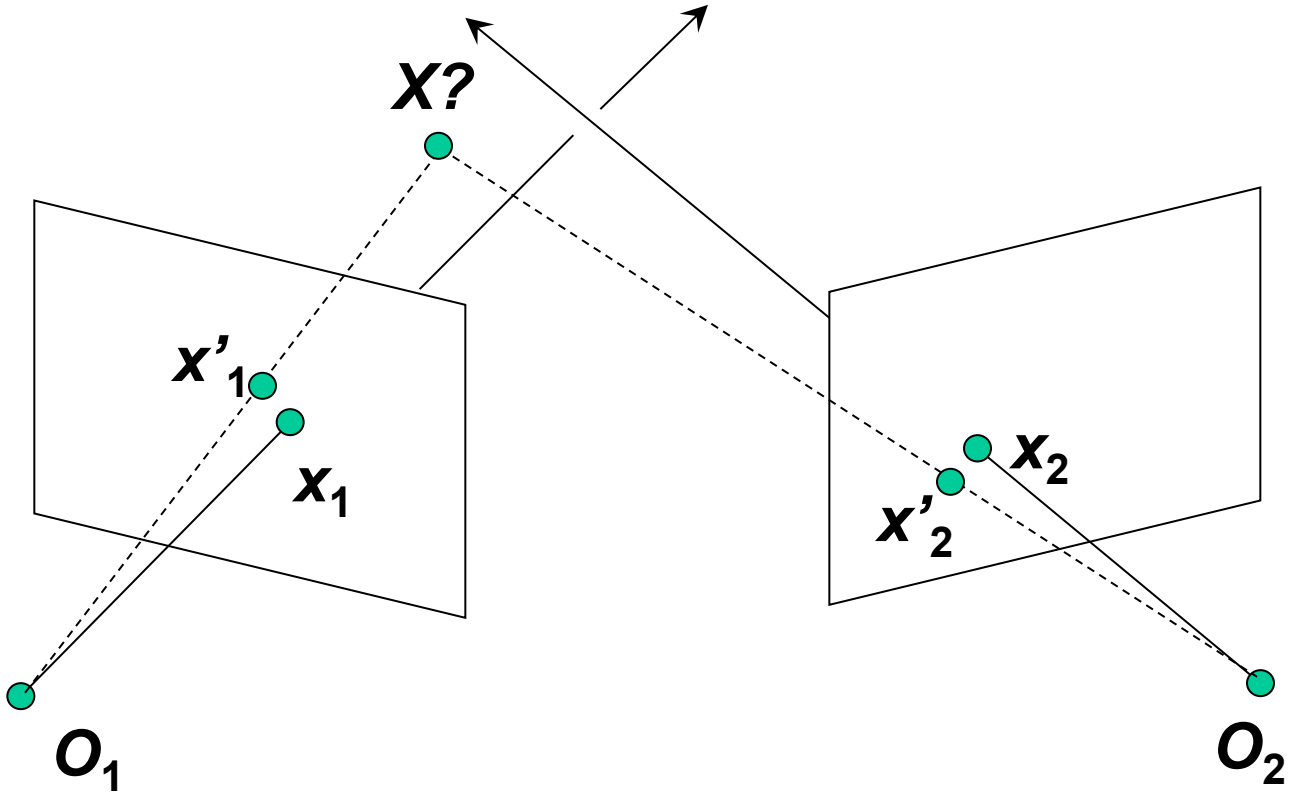



Image credit: Svetlana Lazebnik

# Triangulation: Nonlinear refinement

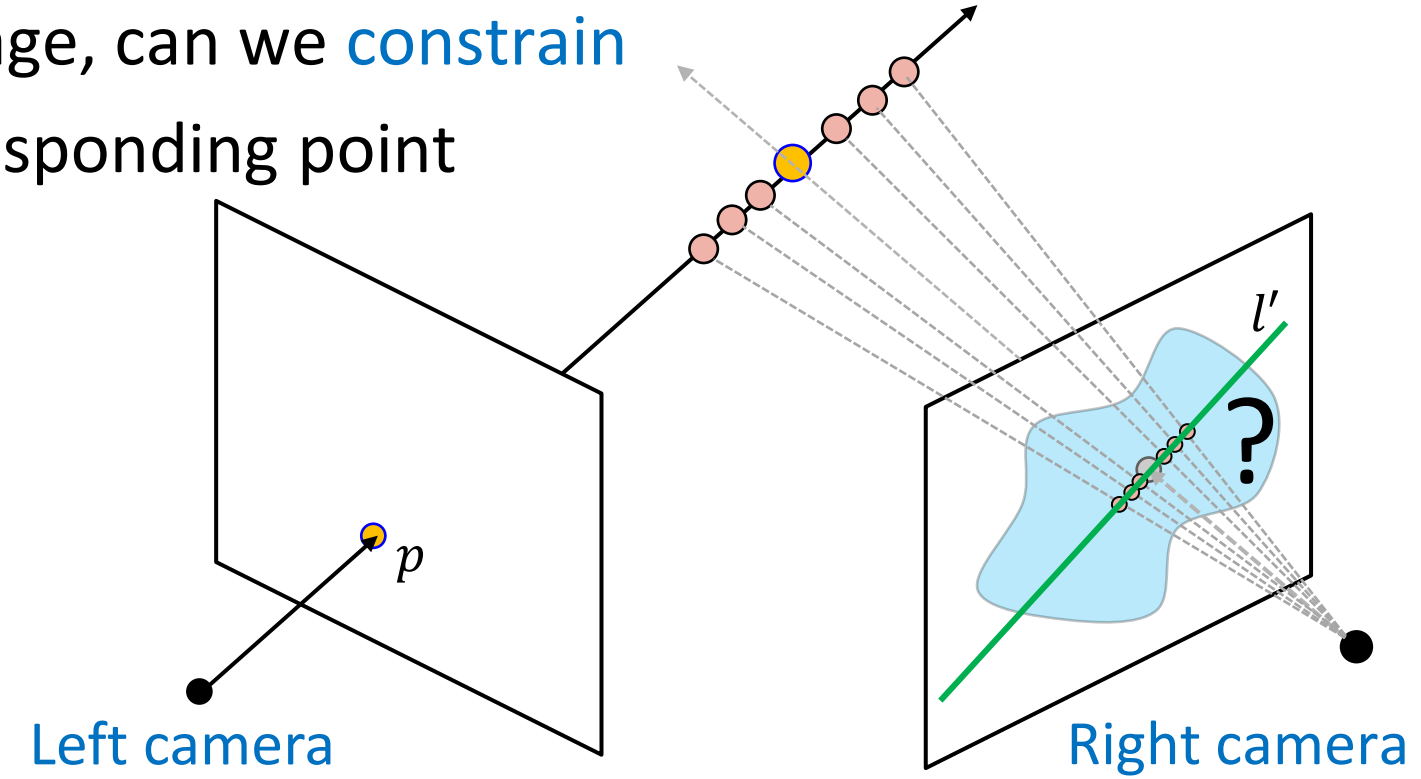
- Find *X* that minimizes a sum of reprojection errors!

$$d^2(x_1, P_1X) + d^2(x_2, P_2X)$$

- Most accurate, but does not have a closed-form solution.
- Requires iterative algorithm (bundle adjustment)
  - Initialize by DLT.
  - Optimize by Gradient descent or Gauss-Newton or Levenberg-Marquardt (see F&P Chapter. 3.1.2 or H&Z Appendix 6).

# But in general correspondences are unknown

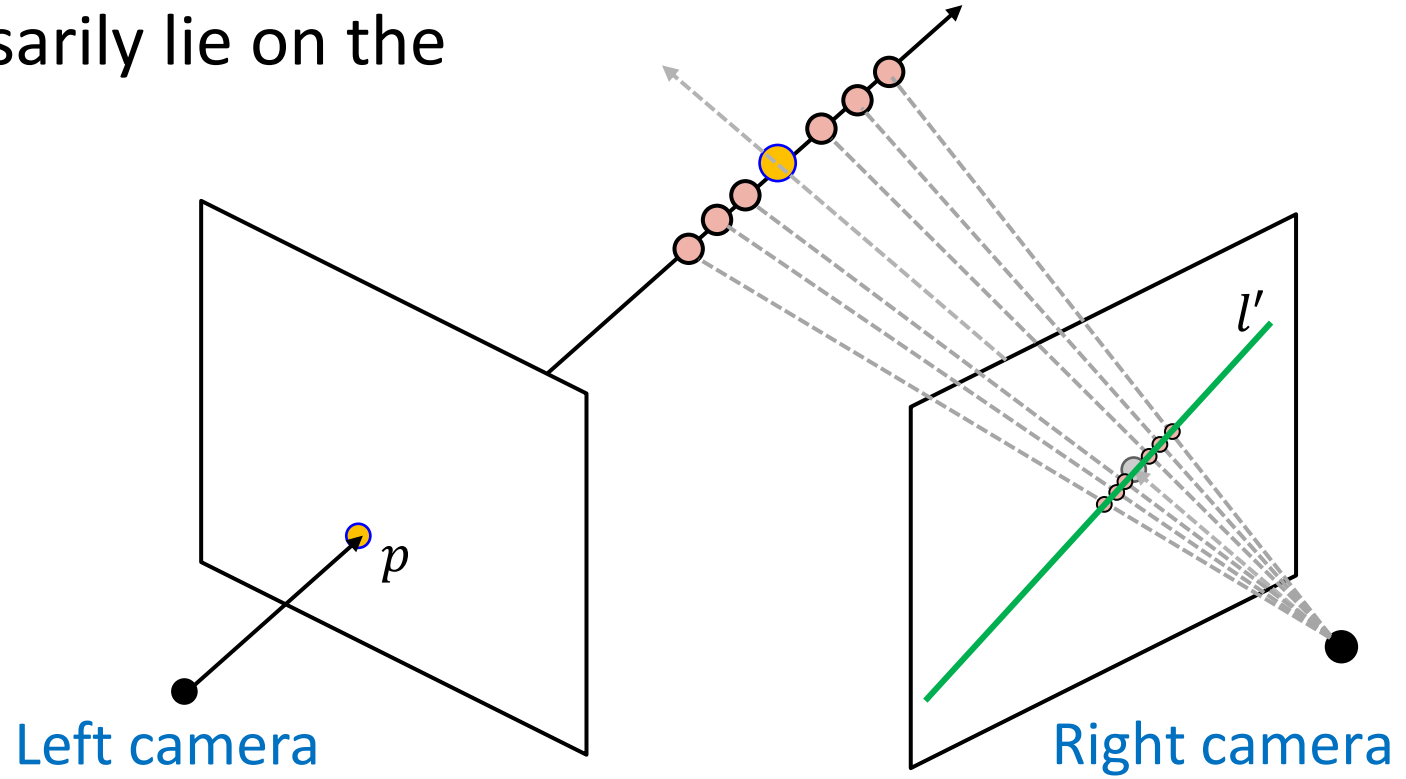
- Correspondences across images are usually not known in advance.
- Assume we know the location of the right camera with respect to the left camera.
- Given a point $\boldsymbol{p}$ in the left image, can we constrain the search region of the corresponding point in the right image?



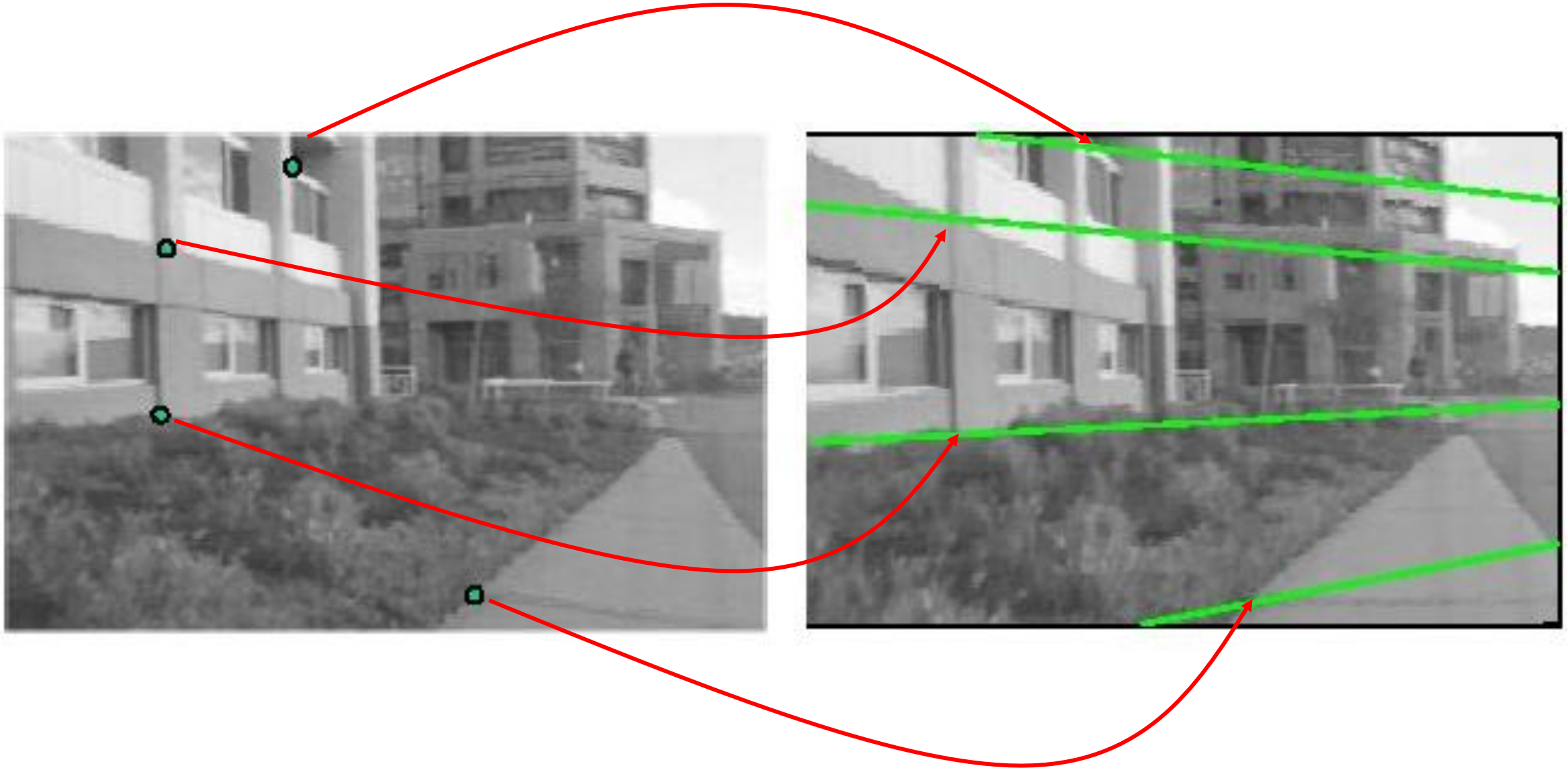
# But in general correspondences are unknown

- Correspondences across images are usually not known in advance.
- Assume we know the location of the right camera with respect to the left camera.
- Potential matches for $\boldsymbol{p}$ necessarily lie on the corresponding *epipolar line* $\boldsymbol{l'}$.

http://www.ai.sri.com/~luong/research/Meta3DViewer/EpipolarGeo.html



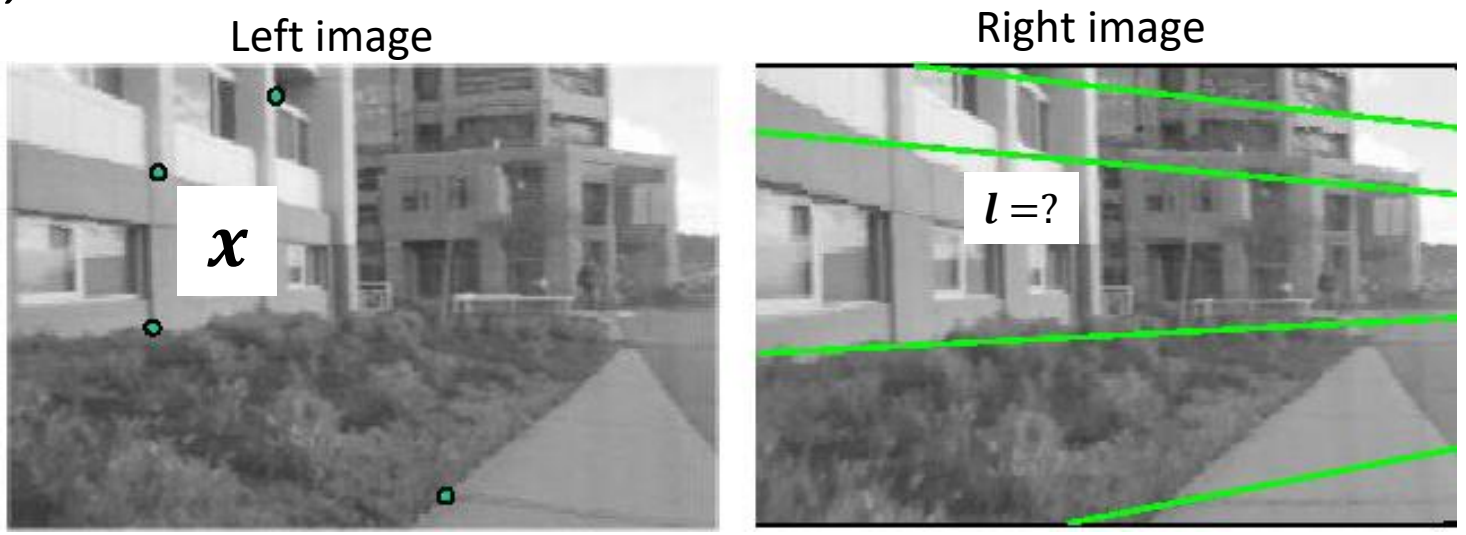
# Example



# Derivation of the epipolar constraint

- The epipolar constraints, for a general stereo system:

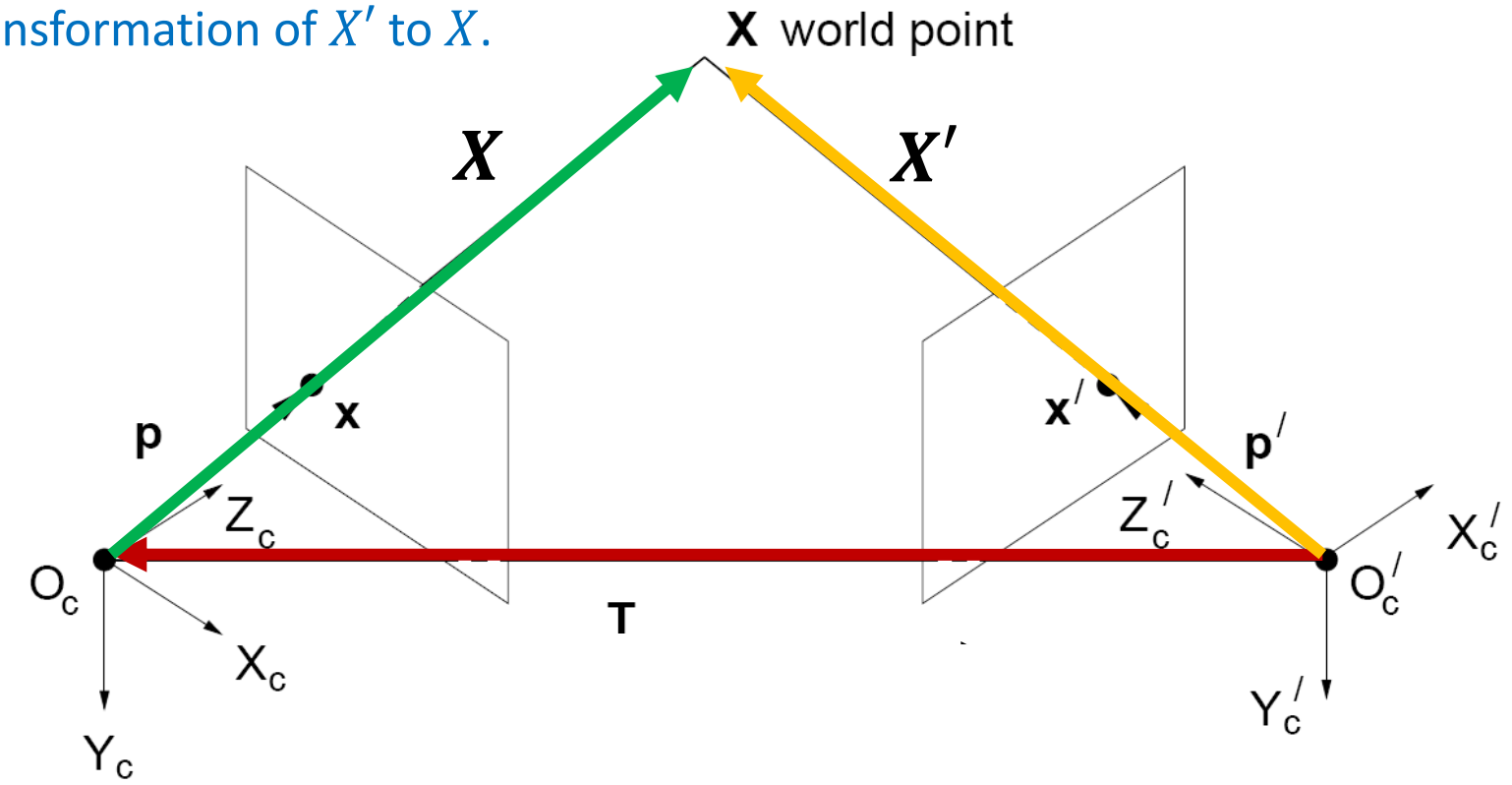
  *"Given a point $\boldsymbol{x}$ in the left image what is the equation of the epipolar line in the right image?"*




- Will look at two cases:
  - Calibrated cameras (known calibration matrices K, K')
  - Noncalibrated cameras (unknown calibration matrices K, K')

# Epipolar constraint: A calibrated system

Note: X‘ is written in c.s. of O‘, while T and X are written in c.s. of O!
Write transformation of $X'$ to $X$.




$$\mathbf{X} = \mathbf{R}\mathbf{X}' + \mathbf{T}$$

$$\mathbf{T} \times \mathbf{X} = \mathbf{T} \times \mathbf{R}\mathbf{X}' + \mathbf{T} \times \mathbf{T}$$

$$= \mathbf{T} \times \mathbf{R}\mathbf{X}'$$

$$\mathbf{X}^T(\mathbf{T} \times \mathbf{X}) = 0$$

$$\mathbf{X}^T(\mathbf{T} \times \mathbf{R}\mathbf{X}') = 0$$

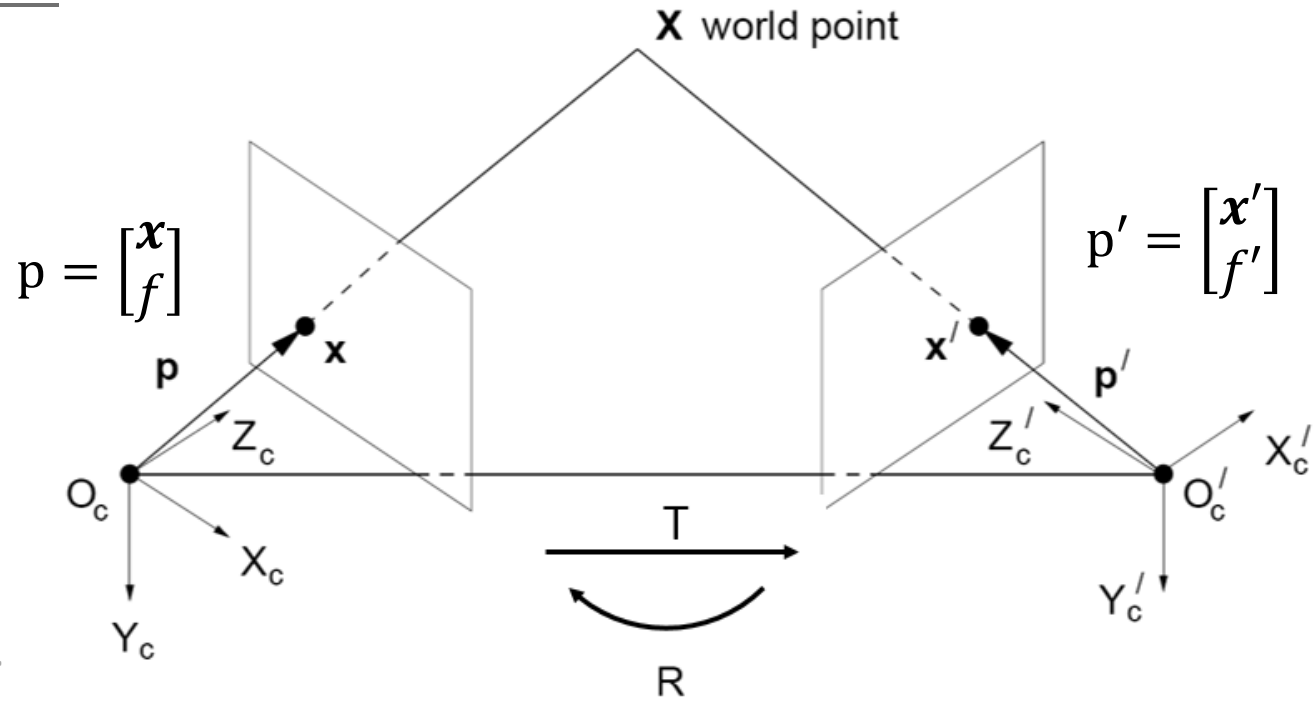
# Epipolar constraint: A calibrated system

$$\mathbf{X}^T(\mathbf{T} \times \mathbf{R}\mathbf{X}') = 0$$

$$\mathbf{X}^T([\mathbf{T}_\times]\mathbf{R}\mathbf{X}') = 0$$

Let $\mathbf{E} = [\mathbf{T}_\times]\mathbf{R}$ , then $\mathbf{X}^T\mathbf{E}\mathbf{X}' = 0$

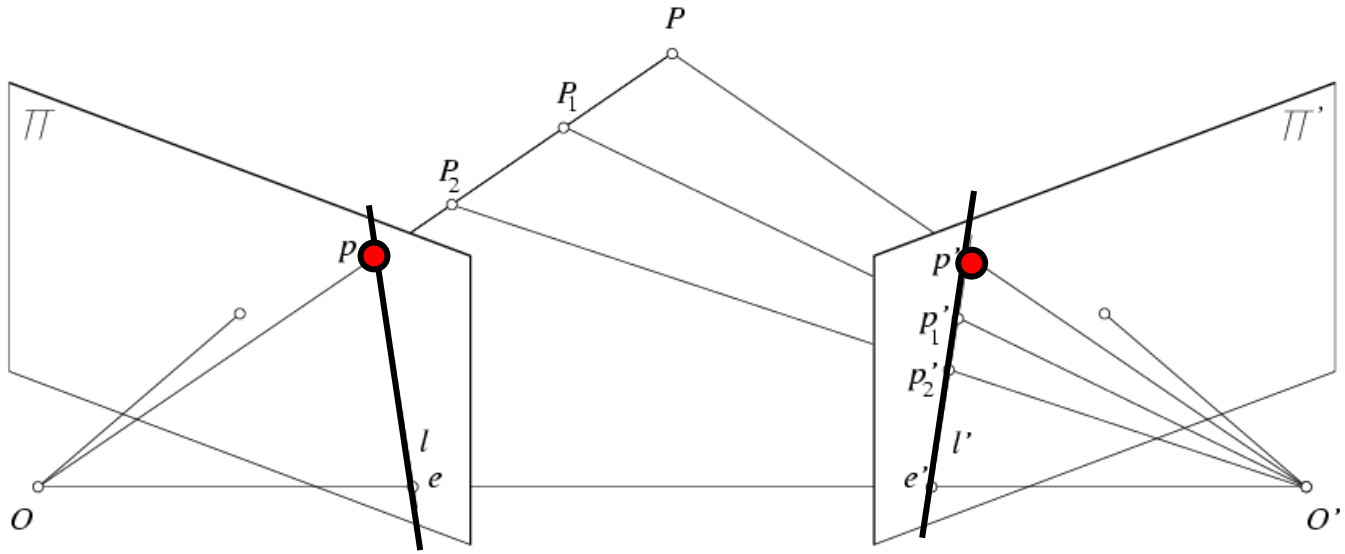
A 3D point written in the left and the right c.s., respectively.




- Points on image plane defined as $\boldsymbol{x} = \lambda_1 \boldsymbol{X}$ and $\boldsymbol{x}' = \lambda_2 \boldsymbol{X}'$, where $\lambda_1$ and $\lambda_2$ are scalars.
- Then this holds: $\boldsymbol{x}^T \boldsymbol{E} \boldsymbol{x}' = \mathbf{0}$
- Matrix $\boldsymbol{E}$ is called an essential matrix, that relates the corresponding image points [Longuet-Higgins 1981]

# Epipolar constraint: Essential matrix

- A 3D point is mapped to points $\boldsymbol{x}$ and $\boldsymbol{x}'$ which are related by $\boldsymbol{x}^T \boldsymbol{E} \boldsymbol{x}' = \boldsymbol{0}$.




*Distance d of point **x** to line **h**:*

$$ax + by + c = 0$$
$$\boldsymbol{h} = [a, b, c]^T$$
$$\boldsymbol{x} = [x_0, y_0, 1]^T$$
$$d = \boldsymbol{h}^T \boldsymbol{x} / \sqrt{a^2 + b^2}$$

https://brilliant.org/wiki/dot-product-distance-between-point-and-a-line/

$\boldsymbol{l} = \boldsymbol{E}\boldsymbol{x}'$ the epipolar line vector $\boldsymbol{l}$, defined in $\Pi$, containing $\boldsymbol{x}$.

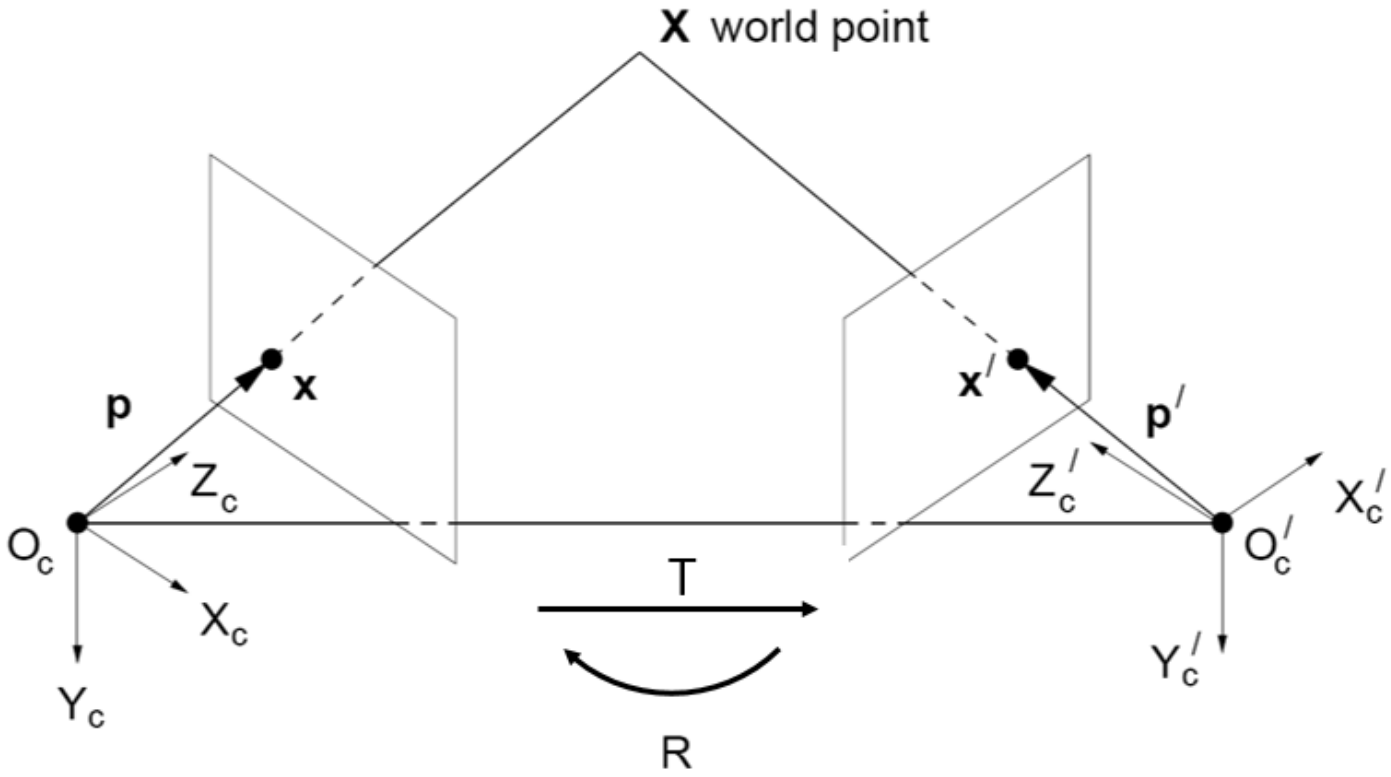
$\boldsymbol{l}' = (\boldsymbol{x}^T \boldsymbol{E})^T$ the epipolar line vector $\boldsymbol{l}'$, defined in $\Pi'$, containing $\boldsymbol{x}'$.

# Epipolar constraint: Essential matrix

- Relates images of corresponding points (meters) in both cameras at a given rotation and translation.
- Can be calculated from known extrinsic parameters:

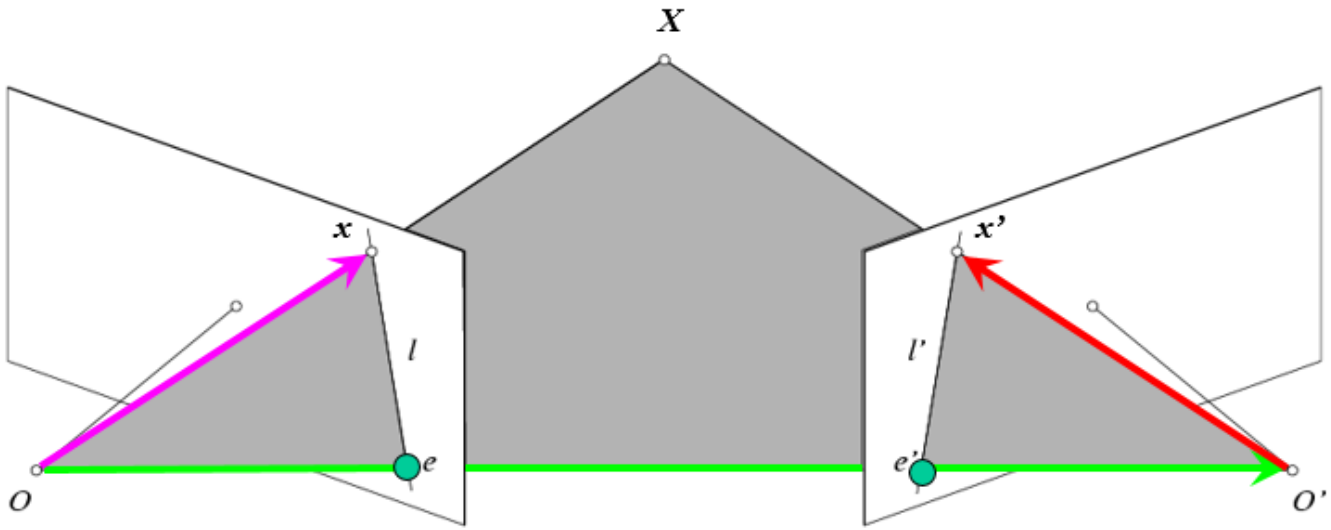
$$\mathbf{E} = [\mathbf{T}_\times]\mathbf{R}$$

Translation and rotation of the second camera with respect (w.r.t.) the first.



# Epipolar constraint: A noncalibrated system

- Now consider image points in pixels!
  - $x'$ & $x$ … image plane coordinates (meters)
  - $\hat{x}'$ & $\hat{x}$… image sensor coordinates (pixels)




- Epipolar constraint for a calibrated system:

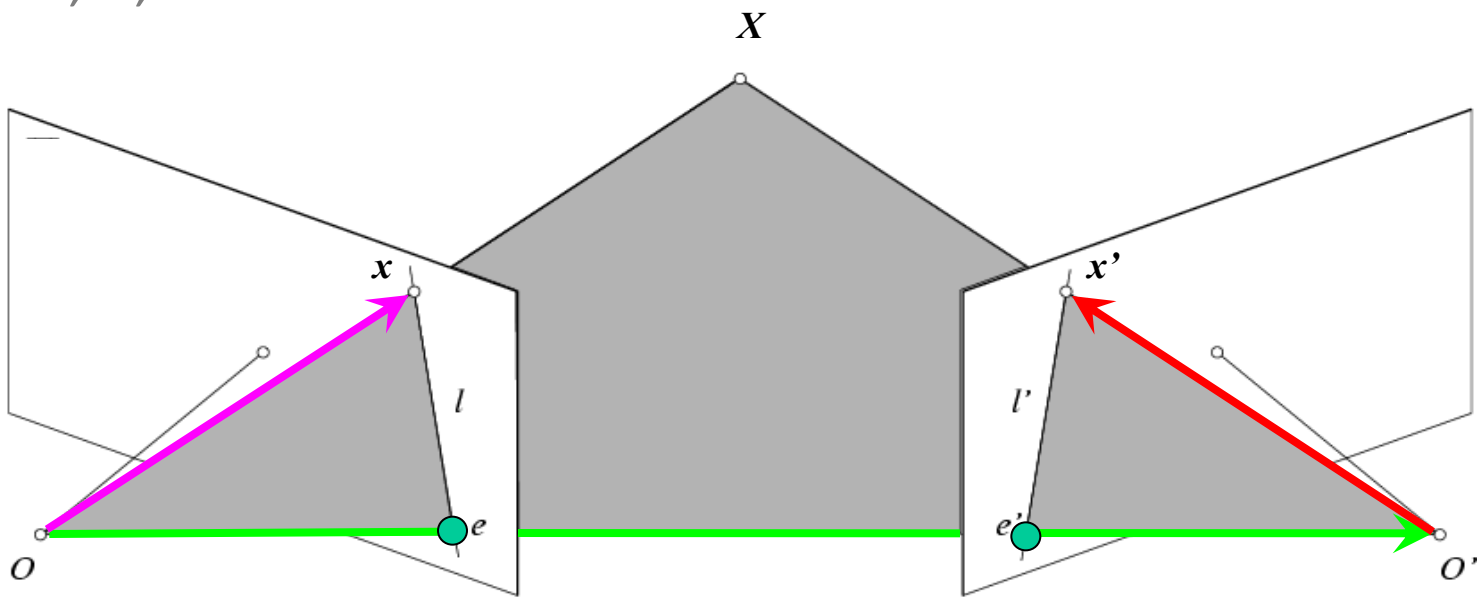
$$x^T E x' = 0$$

- Coordinates related by camera calibration matrix $\boldsymbol{K}$: $\hat{x} = Kx$ (pixels) (meters) $\hat{x}' = K'x'$ (pixels) (meters)

- Camera calibration matrices $\boldsymbol{K}$ and $\boldsymbol{K'}$ unknown → derive the epipolar constraint for the points in pixels

# Epipolar constraint: A noncalibrated system

$\hat{x}, \hat{x}'$ in image pixels ; $x, x'$ in meters




Epipolar constraint: $x^T E x' = 0$ (meters) (meters)

$\Rightarrow \hat{x}^T K^{-T} E K'^{-1} \hat{x}' = 0 \Rightarrow \hat{x}^T F \hat{x}' = 0$

$\hat{x} = Kx$
$\hat{x}' = K'x'$
(pixels) (meters)

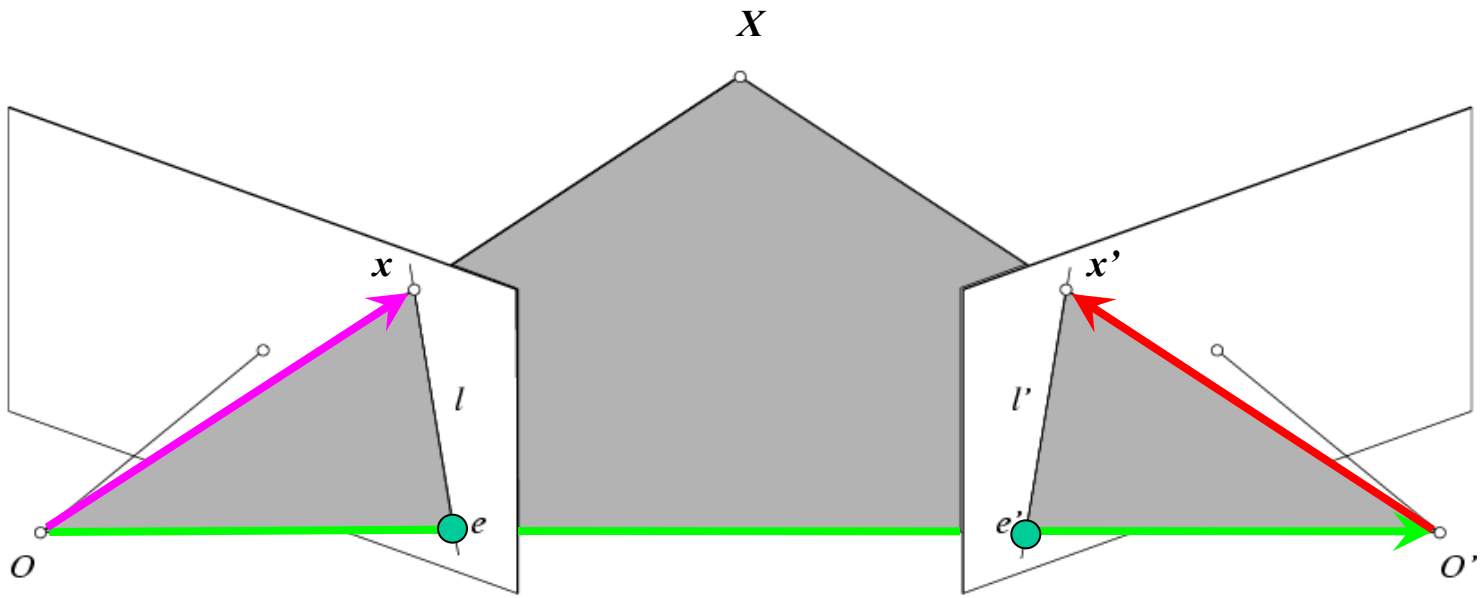
$\Rightarrow x = K^{-1}\hat{x}$
$x' = K'^{-1}\hat{x}'$
(meters) (pixels)

$F = K^{-T} E K'^{-1}$

**Fundamental matrix**

(Faugeras and Luong, 1992)

# Epipolar geometry: Fundamental matrix

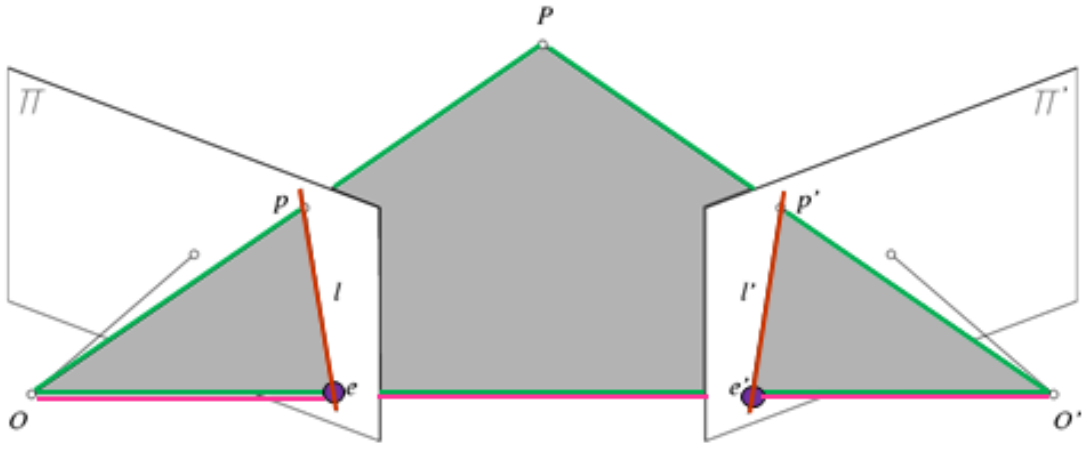



$x^T \mathrm{E} x' = 0 \;;\; \hat{x}' = Kx' \;;\; \hat{x} = Kx$ ⇒ $\hat{x}^T \mathrm{F} \hat{x}' = 0$ with $F = K^{-T} E K'^{-1}$

- $\boldsymbol{F\hat{x}'}$ is epipolar line passing through $\hat{\boldsymbol{x}}$, *i.e.*, $(\boldsymbol{l = F\hat{x}'})$
- $\boldsymbol{F^{\mathrm{T}}\hat{x}}$ is epipolar line passing through $\boldsymbol{x}$, *i.e.*, $(\boldsymbol{l' = F^{\mathrm{T}}\hat{x}})$
- $\boldsymbol{Fe' = 0 \;;\; F^{\mathrm{T}} e = 0}$

# Epipolar geometry: crucial terms

- *Baseline*: a line connecting the camera centers.
- *Epipole*: point where the baseline punctures the image plane.
- *Epipolar plane*: plane connecting two epipoles and a 3D point.
- *Epipolar line*: intersection of epipolar plane and image plane.




- All epipolar lines of a single image intersect at the camera epipole.

# Epipolar matrix vs Fundamental matrix

**Epipolar matrix** (calibrated case):

$$E = [\mathbf{T}_\times]\mathbf{R}$$

Note: in $[\mathbf{T}_\times]$, one column is a linear combination of the other two.

$\text{rank}([\mathbf{T}_\times]) = 2 \rightarrow |[\mathbf{T}_\times]|=0$, thus
$|E|=0$ & rank(E)=2

$$E = U \begin{bmatrix} \sigma & 0 & 0 \\ 0 & \sigma & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

DoF: 9 – 1 – 1 – 2 = 5

**Fundamental matrix** (noncalibrated case):

$$F = K^{-T} E K'^{-1}$$

$|F|=0$ , rank(F)=2

$$F = U \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

DoF: 9 – 1 – 1 = 7

# Special case: Geometry of a simple stereo

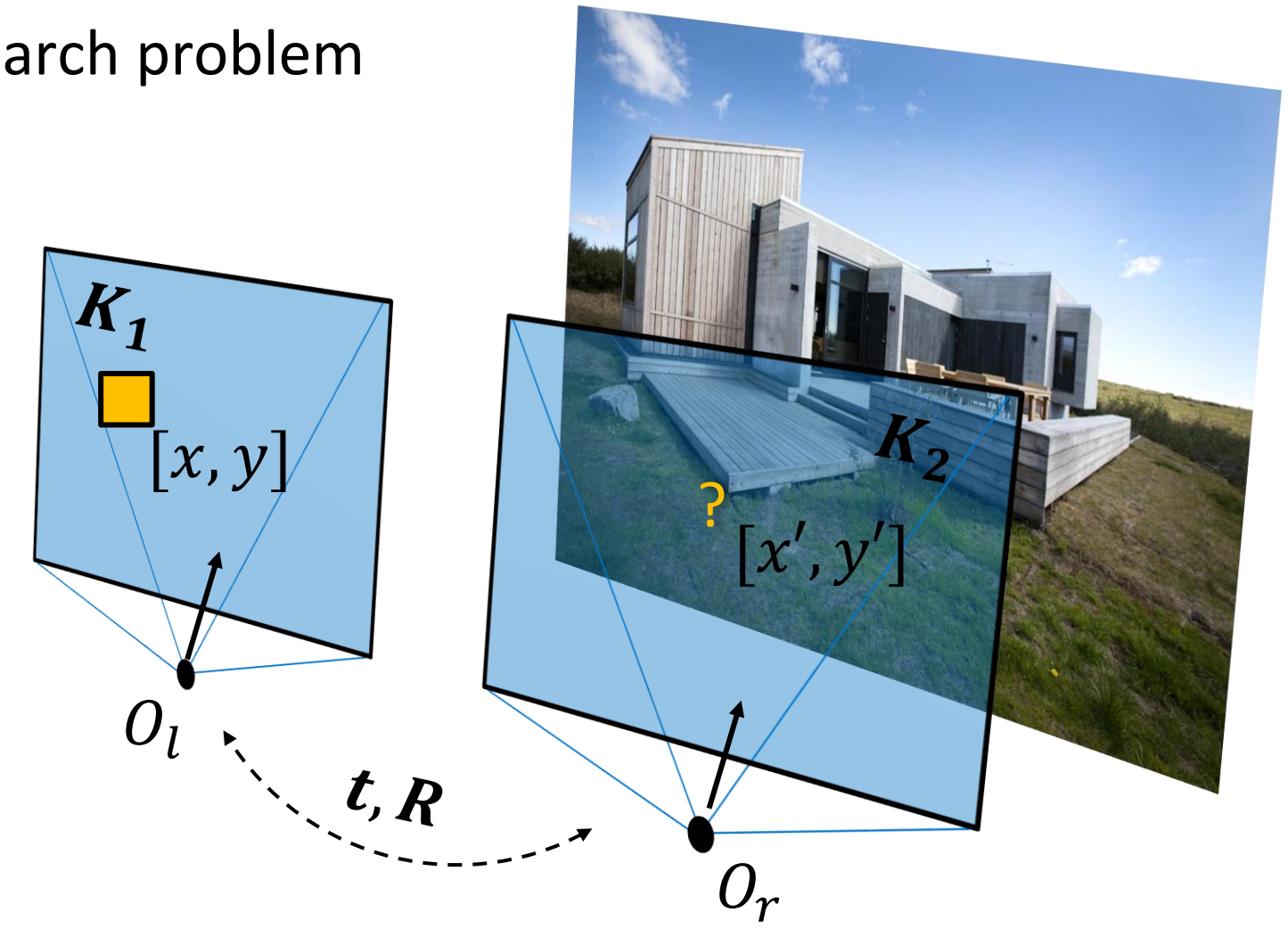
- Now consider a calibrated stereo system with parallel optical axes.
- This will simplify the search problem significantly…

$$\mathbf{E} = \mathbf{T}_{\times}\mathbf{R}$$

$$\boldsymbol{x}^{\mathrm{T}}\mathbf{E}\boldsymbol{x}' = 0$$

$$\boldsymbol{l}' = (\boldsymbol{x}^T \boldsymbol{E})^T$$

*Given $[x, y]$ in the left image, where will the corresponding $[x', y']$ be in the right image?*



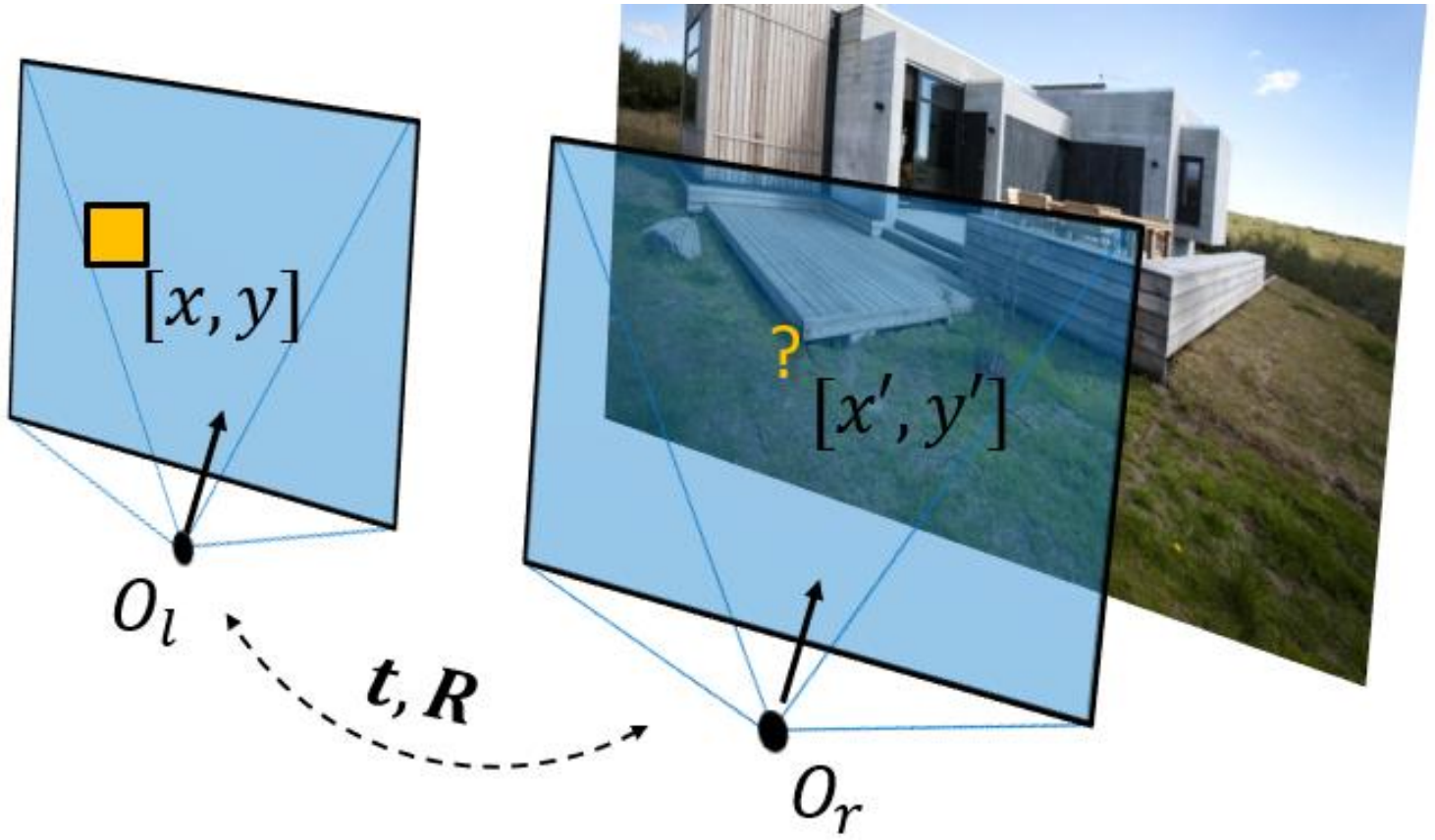
# Special case: Geometry of a simple stereo

$$\mathbf{E} = \mathbf{T}_{\times}\mathbf{R}$$

$$\boldsymbol{x}^{\mathrm{T}}\mathbf{E}\boldsymbol{x}' = 0$$

$$\boldsymbol{l}' = ?$$

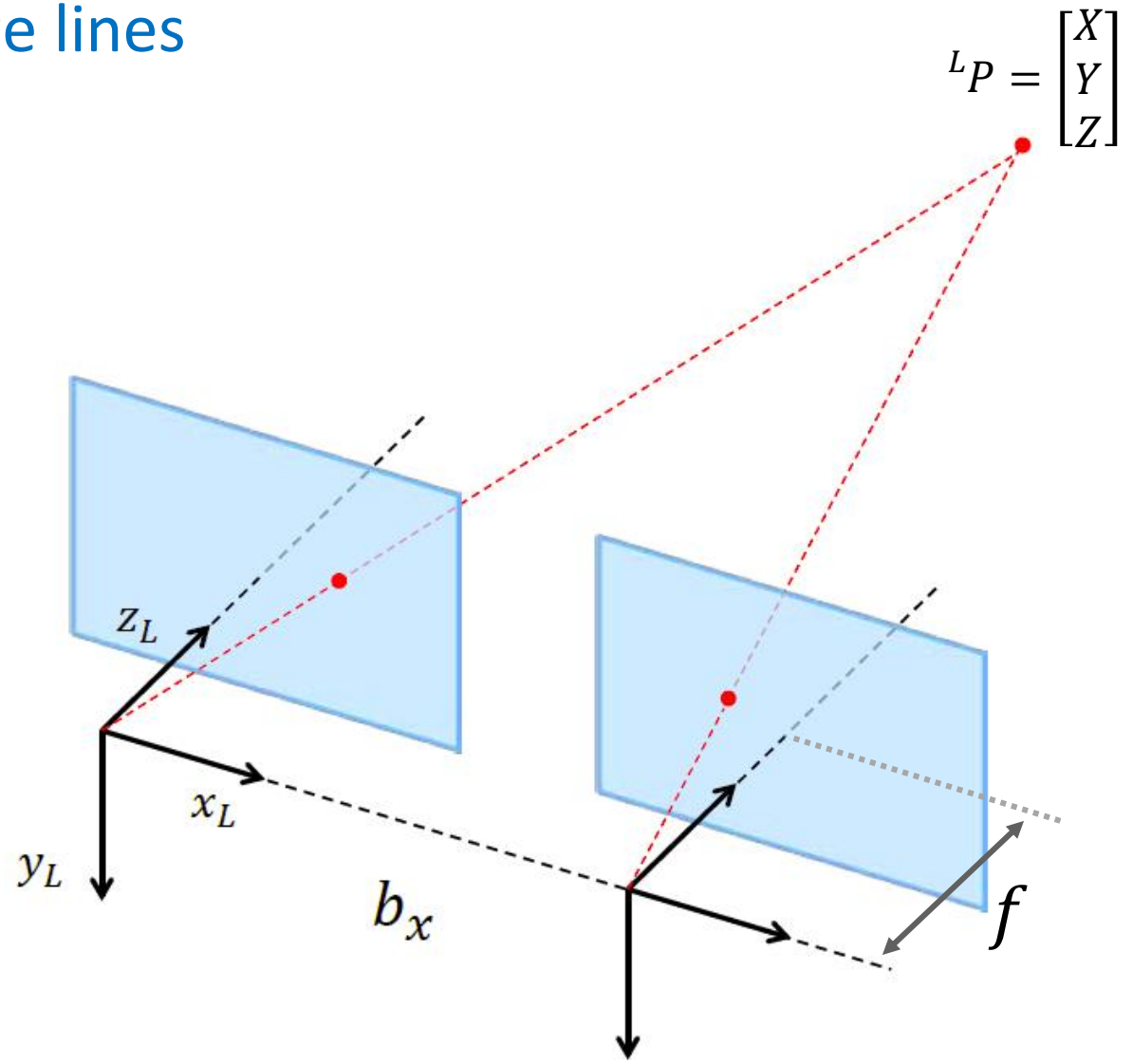
$$\mathbf{a} \times \mathbf{b} = \begin{bmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{bmatrix} \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = [\mathbf{a}_{\times}]\mathbf{b}$$



# Geometry of a simple stereo

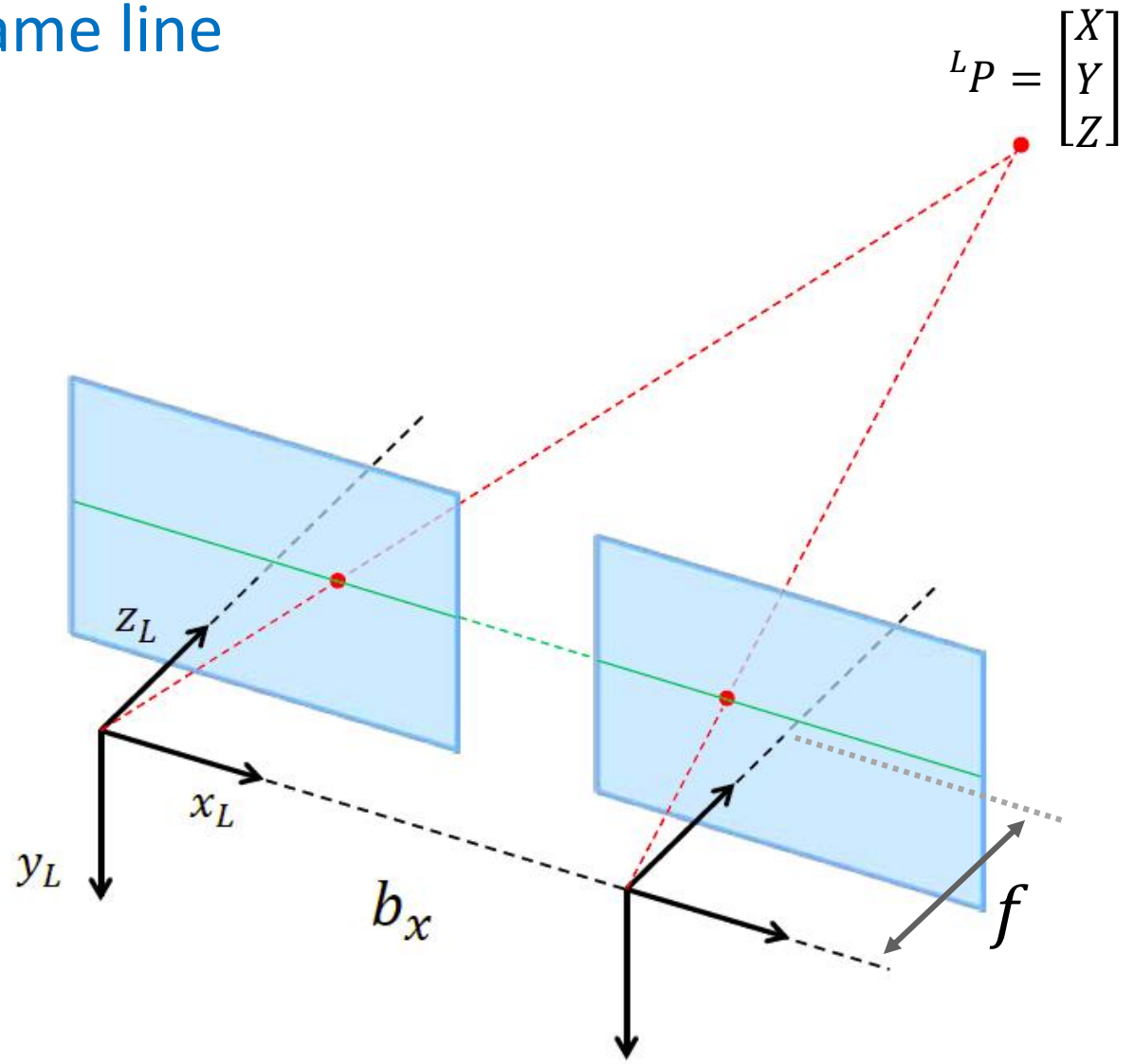
- Parallel optical axes with aligned image lines
- A 3D point written in the coordinate system of the left camera: ${}^{L}P$.
- Baseline $b_x$: displacement of the right camera along $x_L$.
- Focal length $f$: distance of image planes (in both cameras) from their projection centers.

Depth estimation simplifies…



# Geometry of a simple stereo

- The corresponding points lie on the same line of pixels (epipolar line).



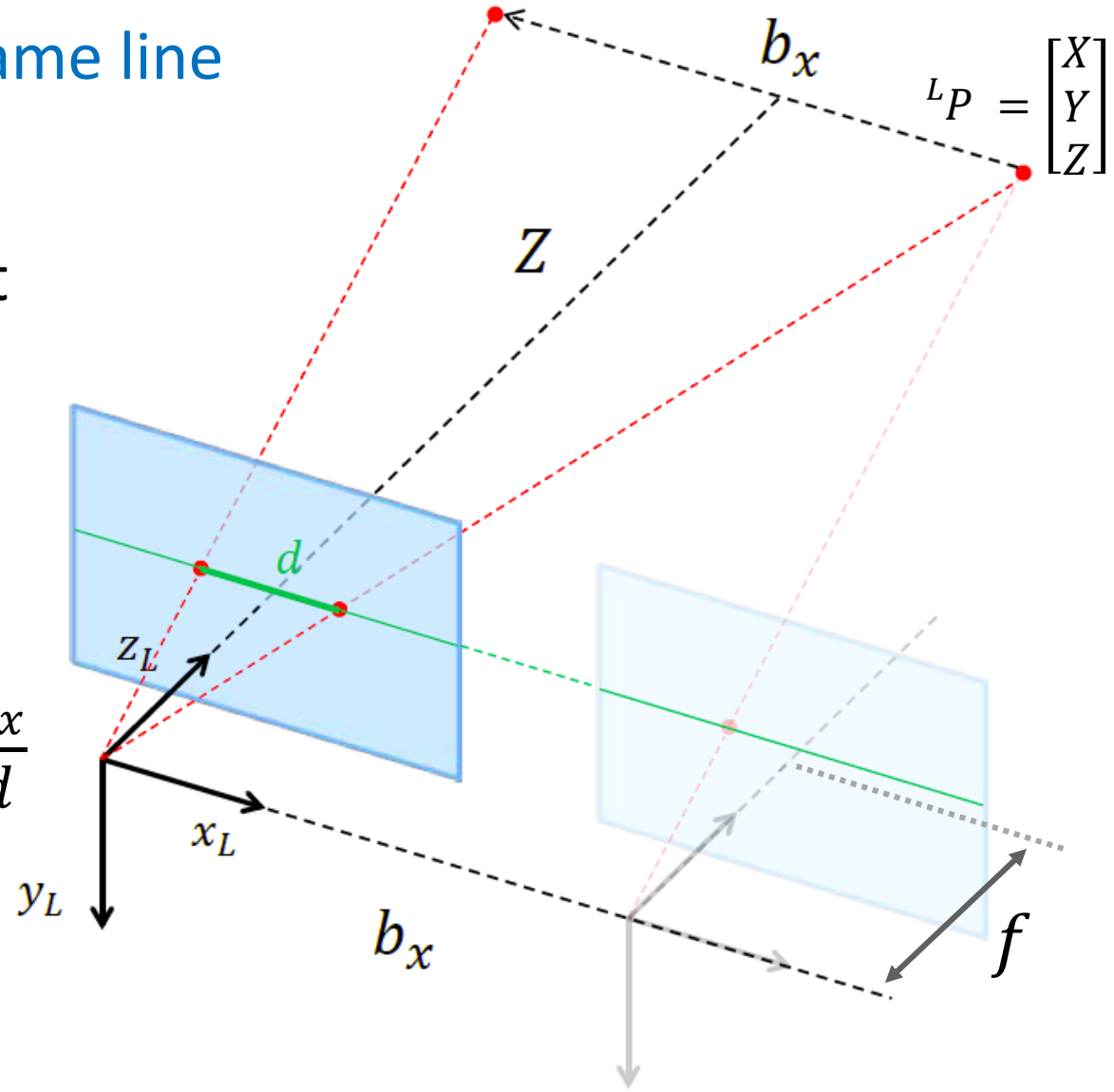
# Geometry of a simple stereo

- The corresponding points lie on the same line of pixels (epipolar line)
- Align the right projection onto the left image (displace coordinates of the right projection by "$-b_x$").
- Depth from disparity:

$$Z = f\frac{b_x}{d}$$

- 3D from disparity

$$X = x_L\frac{b_x}{d} \quad , \quad Y = y_L\frac{b_x}{d} \quad , \quad Z = f\frac{b_x}{d}$$

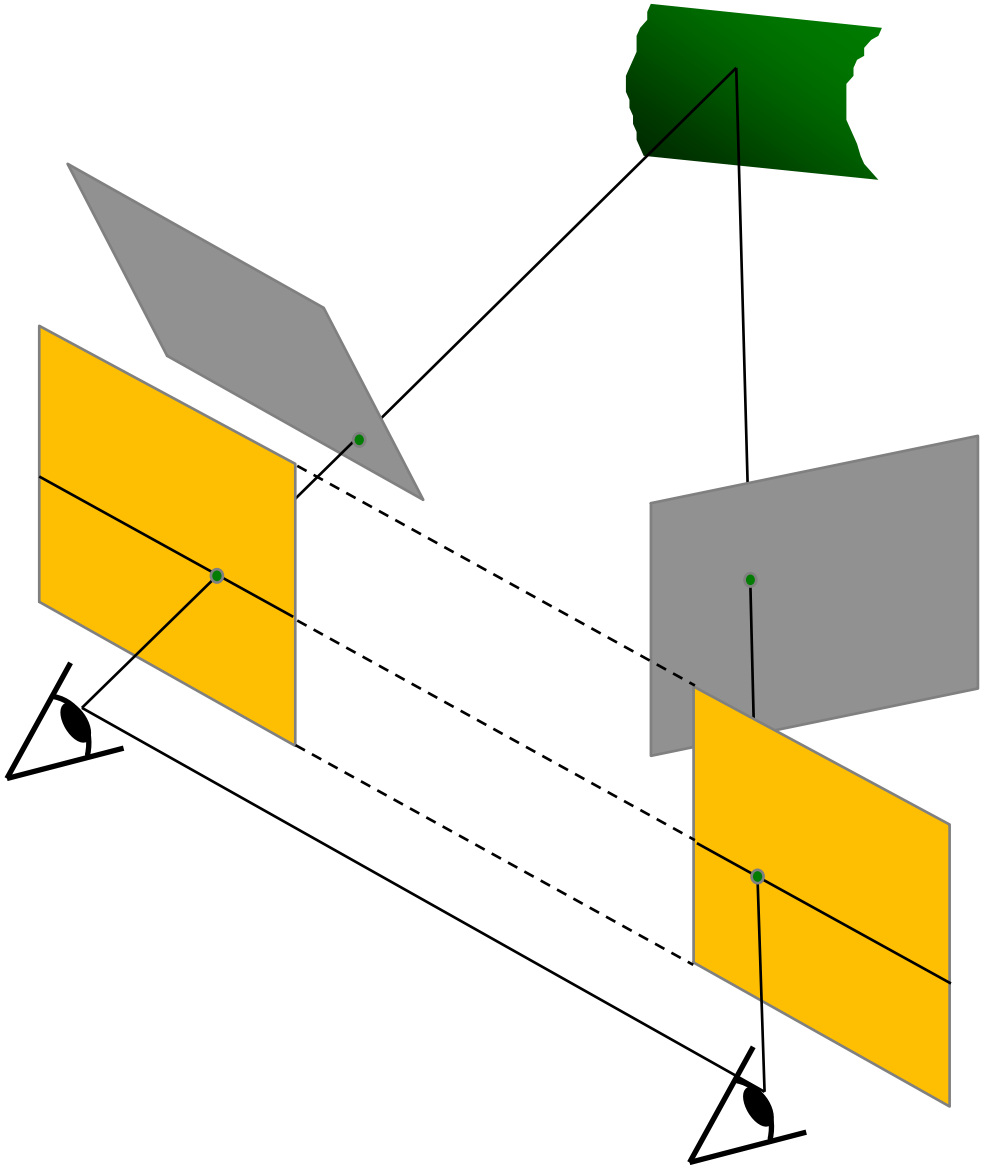


# Stereo image rectification

- It is convenient if the lines for searching the matches correspond to the epipolar lines – as simple as in parallel cameras system

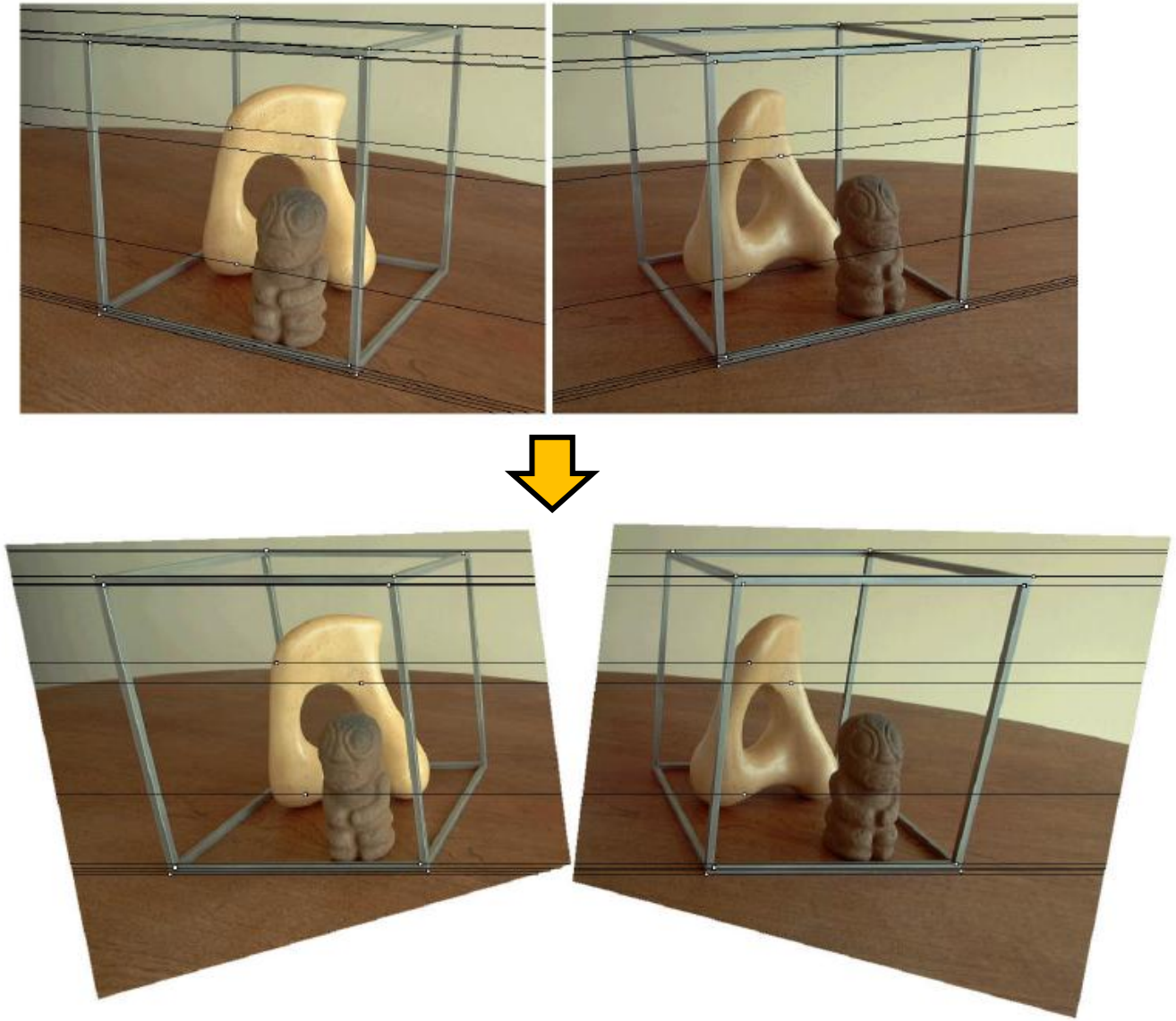
- Reproject image planes into a common plane, parallel to the baseline.

- Two homographies (3x3) – matrix transformation for reprojection of left and right image planes.



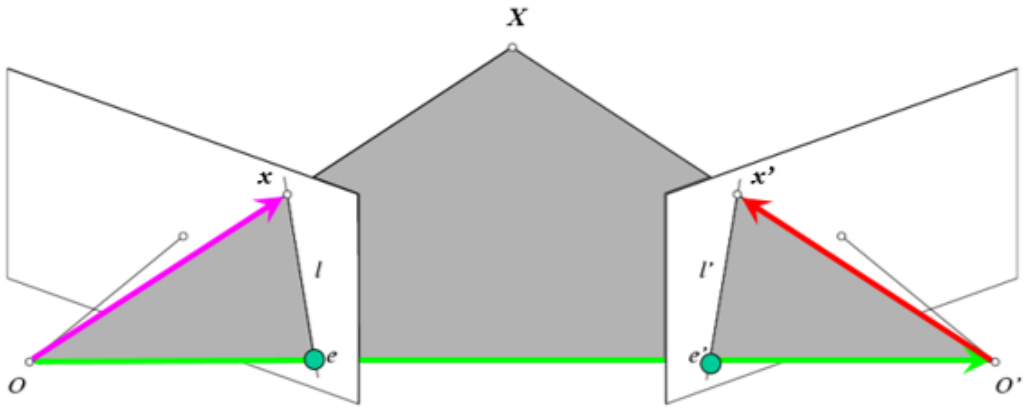
C. Loop & Z. Zhang, Computing Rectifying Homographies for Stereo Vision. CVPR'99

# Stereo image rectification

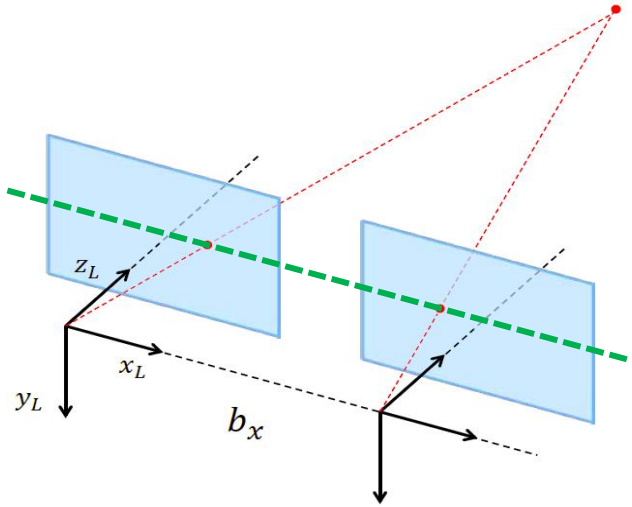


# Previously at MP…

- A system of two or more cameras: triangulation & epipolar geometry

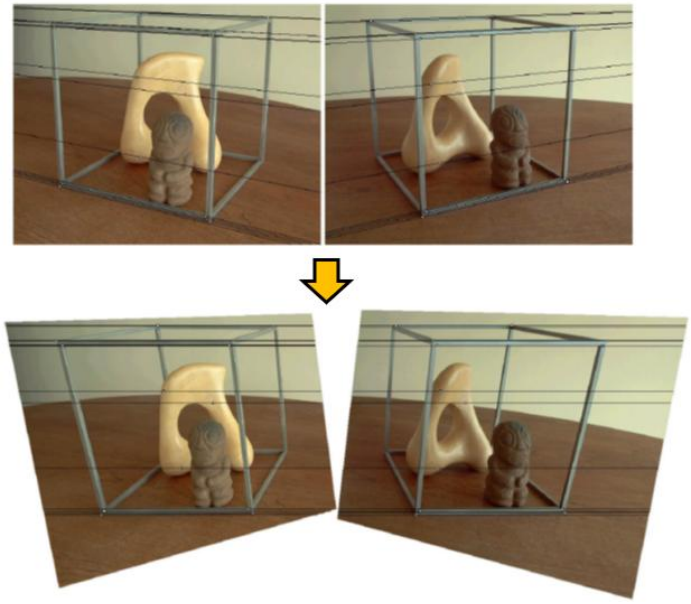



- A system of perfectly aligned cameras:



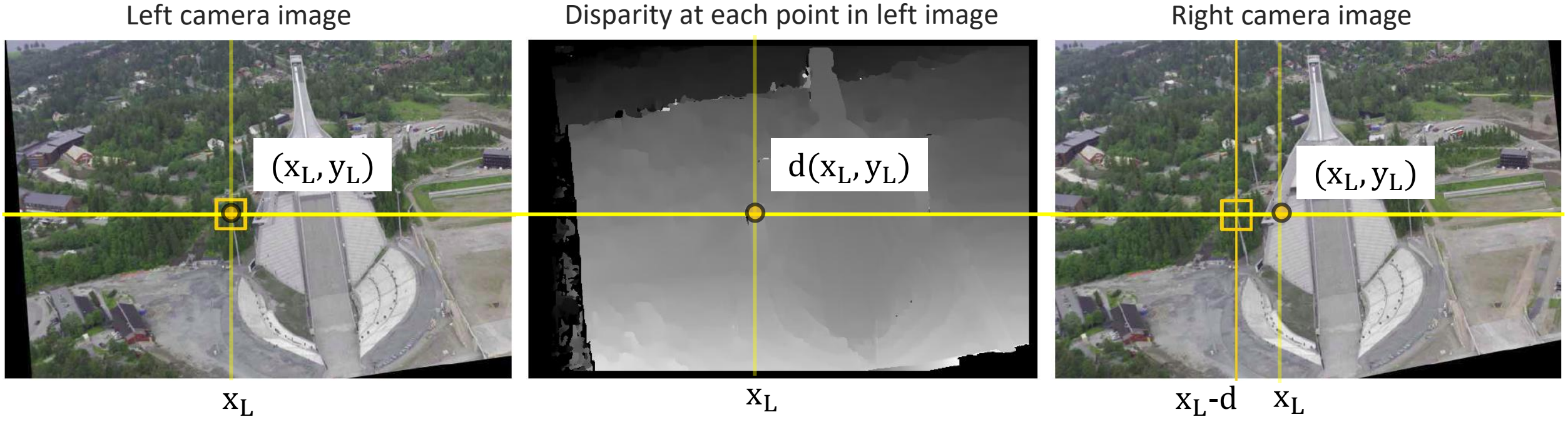

$$X = x_L \frac{b_x}{d}, Y = y_L \frac{b_x}{d}, Z = f \frac{b_x}{d}$$

**Disparity $d$:** the difference in $x$ position of a point in the left and he right camera



# Disparity and depth

- We'll assume a perfectly rectified stereo system: $d = x_L - x_R \;;\; d \geq 0$
- Disparity at point is inversely proportional to its depth: $Z = \frac{1}{d} \cdot f b_x$
- Relates coordinates of points in left and right image

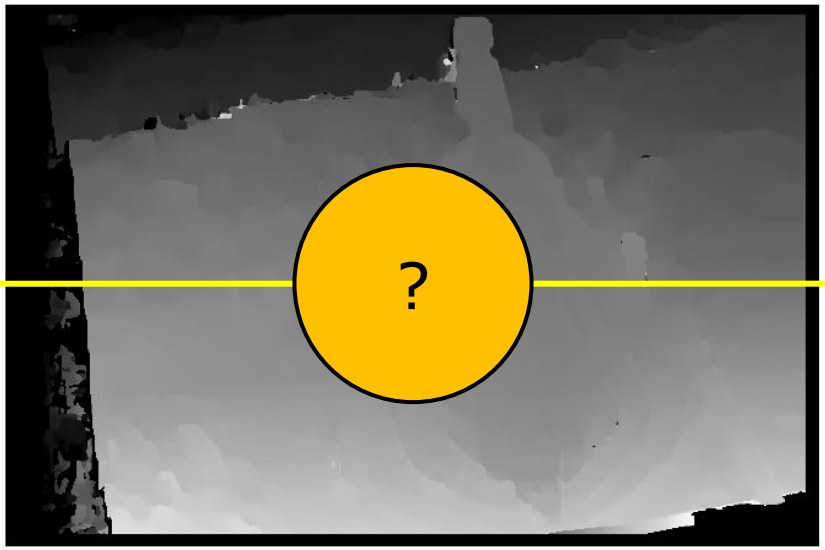


# Disparity estimation

- We'll assume a perfectly rectified stereo system: $\mathrm{d} = \mathrm{x_L} - \mathrm{x_R} \;\; ; \; \mathrm{d} \geq 0$
- Disparity at point is inversely proportional to its depth: $Z = \frac{1}{d} \cdot f b_x$
- How do we compute the disparity for all points in the left image?

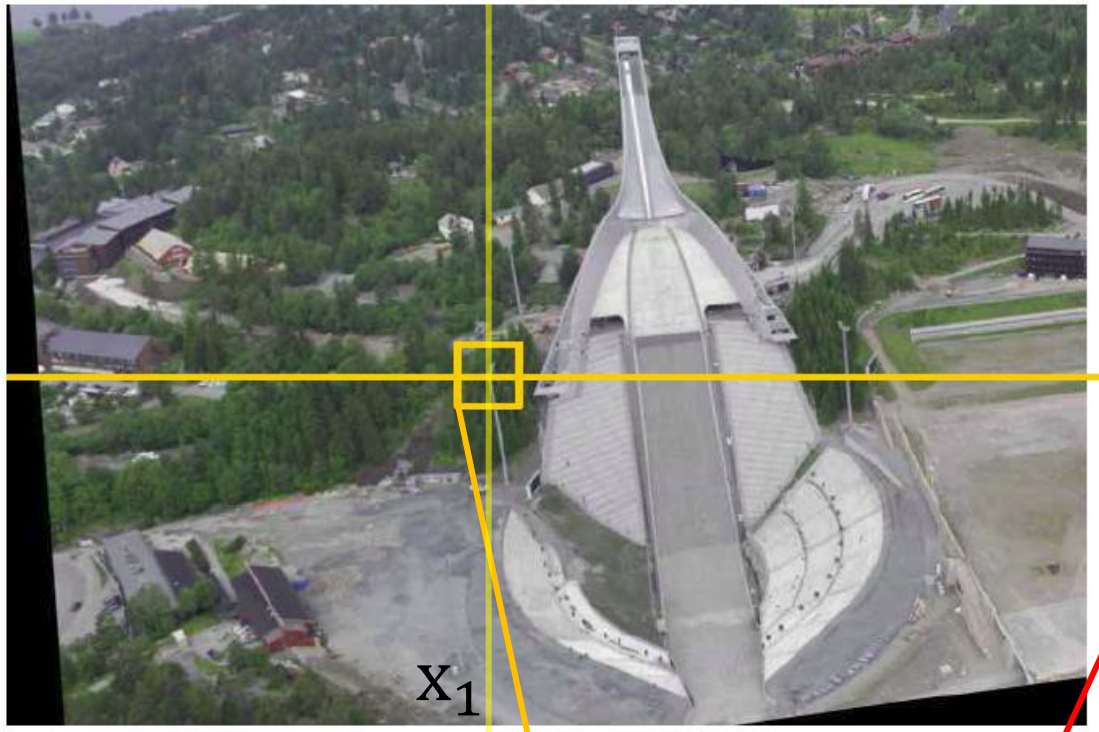
Left camera image

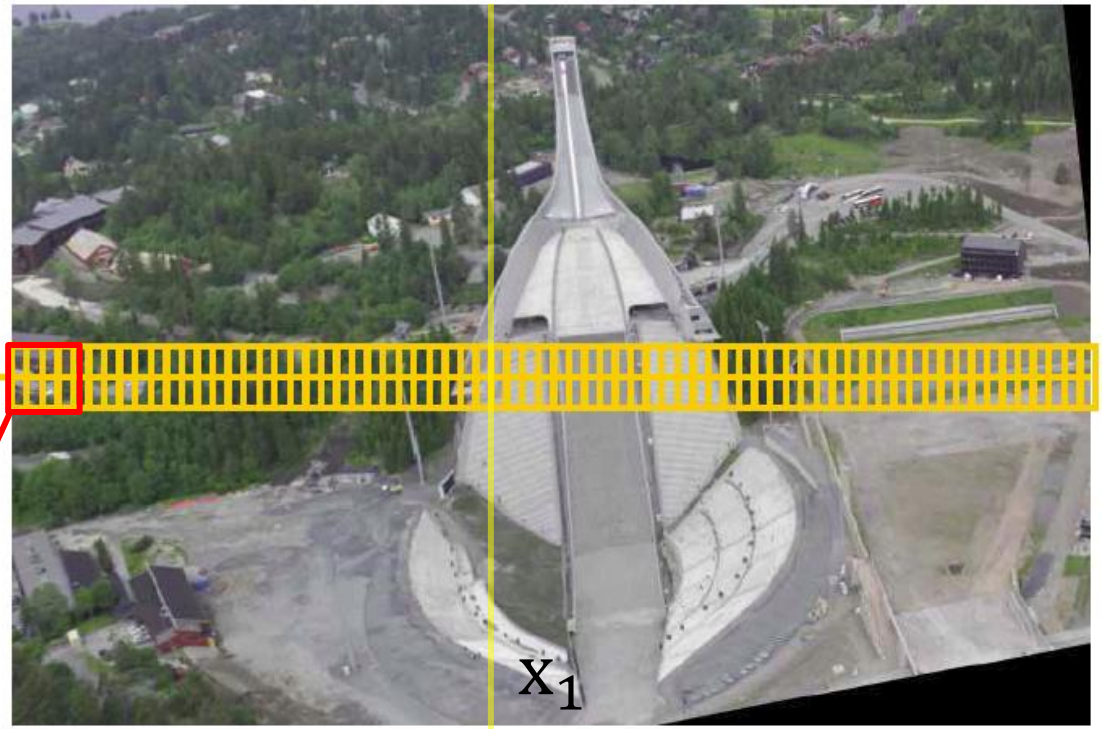
Disparity at each point in left image

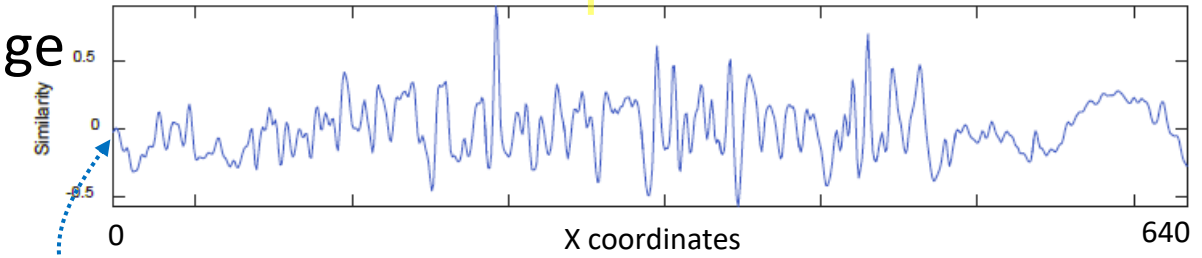



Right camera image

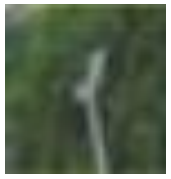
# Disparity estimation



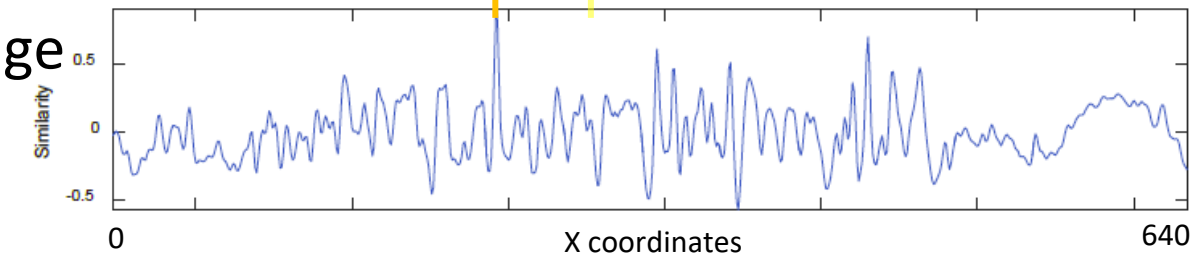




- For a patch centered at a pixel in the left image
- Compare to all patches in the right image along the epipolar line (same line)

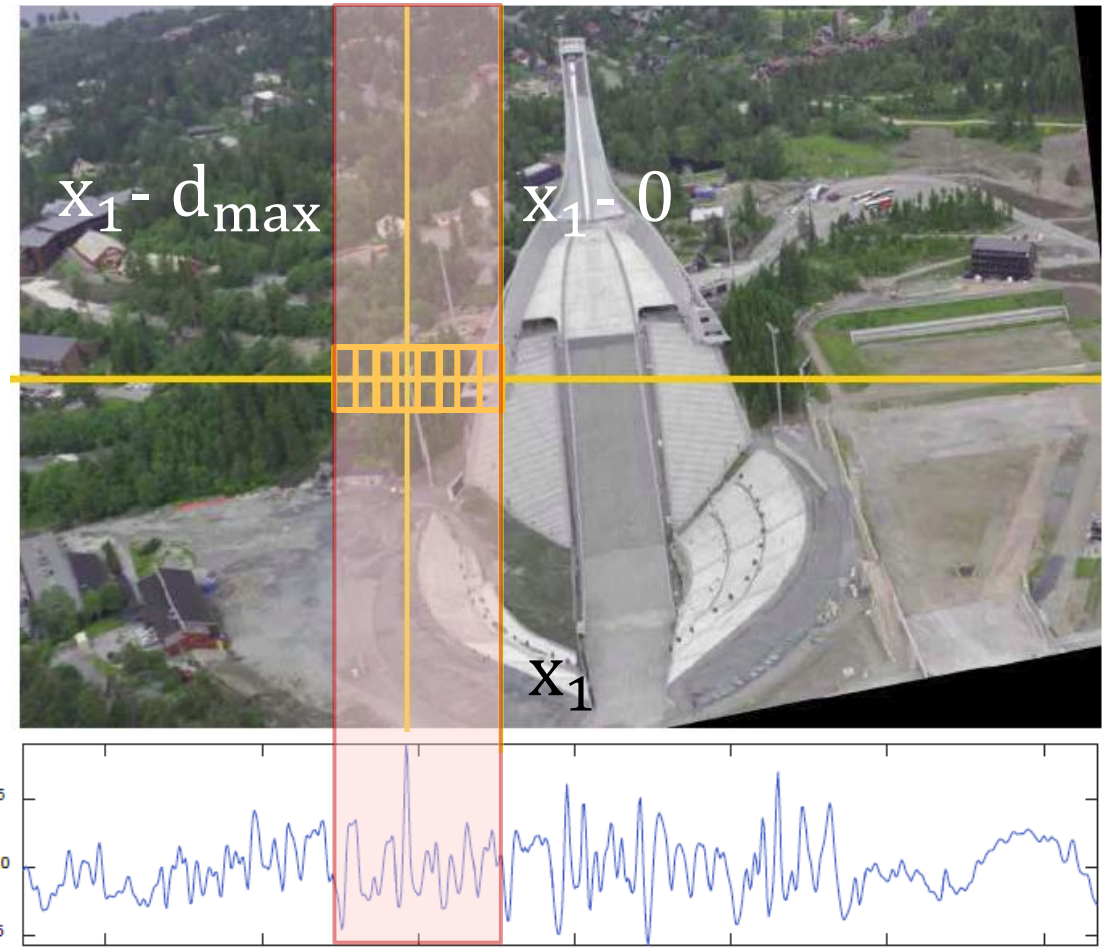
NCC(  ,  ) = 0.01

# Disparity estimation


- For a patch centered at a pixel in the left image
- Compare to all patches in the right image along the epipolar line (same line)
- Select the patch with greatest similarity.
- Difference in position of left patch and right patch is the disparity. $d = x_1 - x_2$

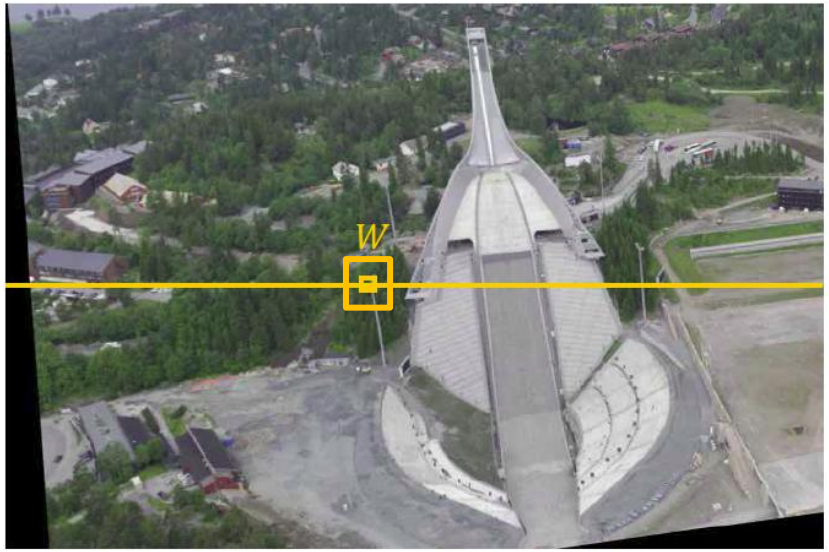
# Disparity estimation



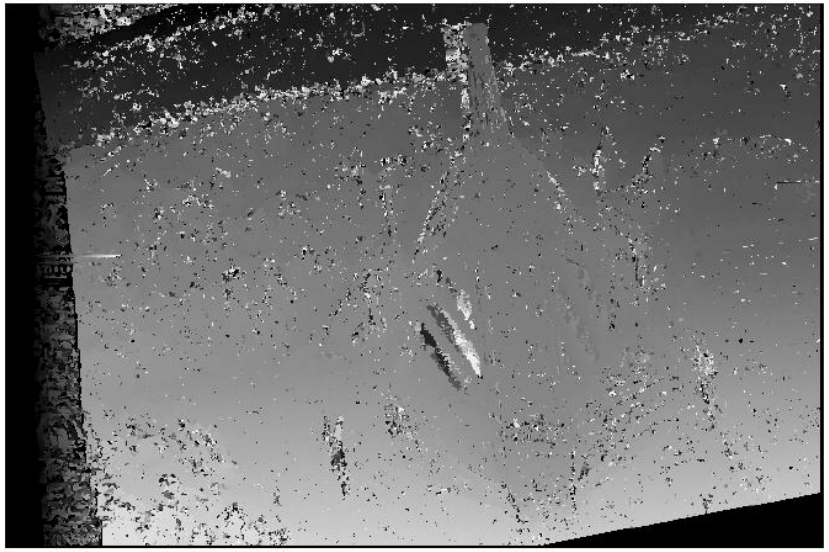

- In practice the disparity values are restricted to a reasonable range of viable disparities.
- E.g.: disparity for an object very far away from the target is 0.
  $d_{max}$ is specified by the minimum distance of an object from the camera
  (see geometrical model of a simple stereo system)

# Disparity: influence of the window size

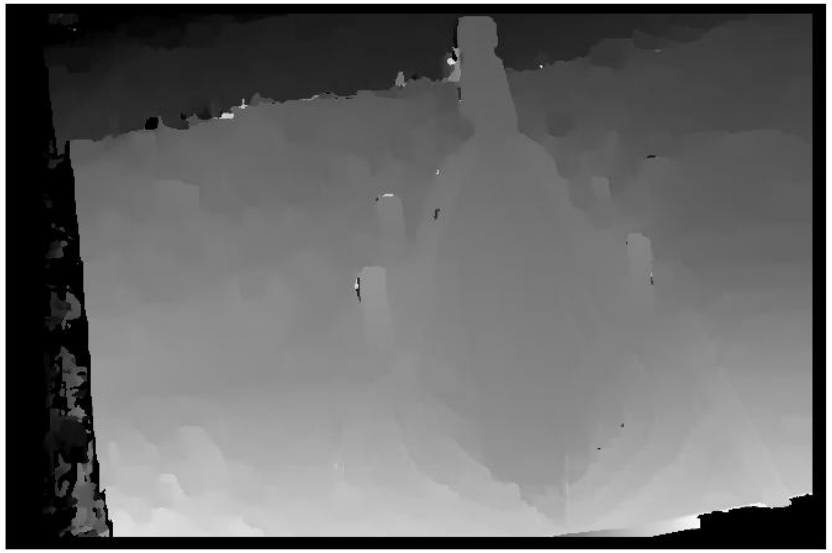
Left image




Small $W$



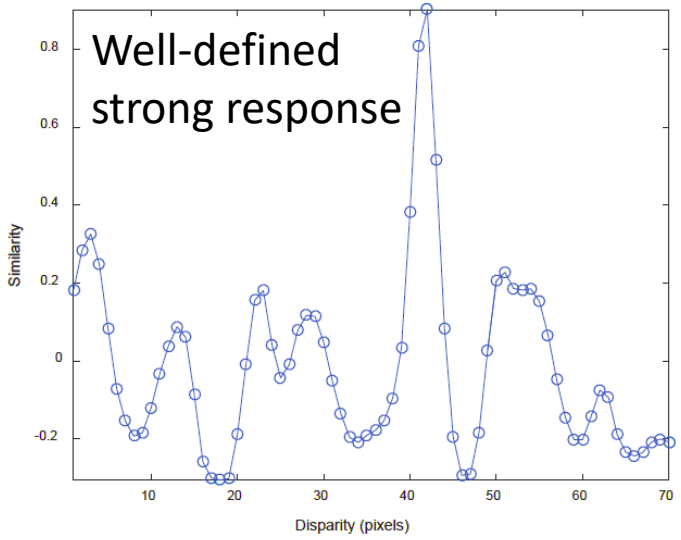
Large $W$

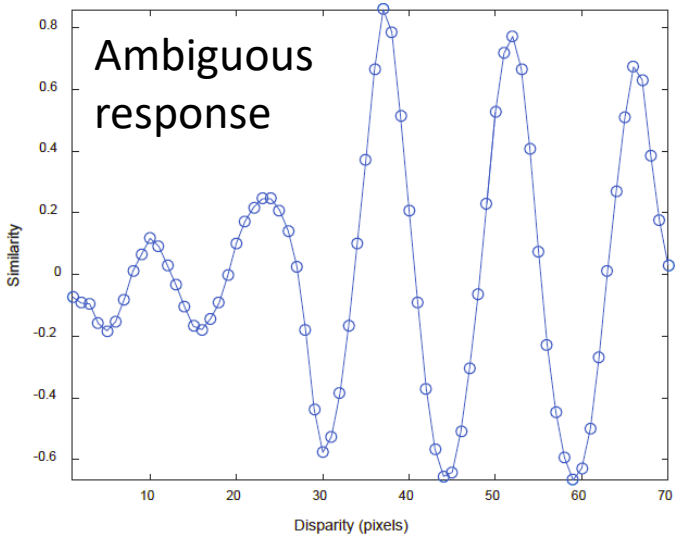


- Small window size $W$:
  - Details potentially better estimated
  - Noisy disparity
  - Fast(er) computation
- Large window size $W$:
  - Details potentially lost
  - (over)Smooth disparity
  - Slow(er) computation

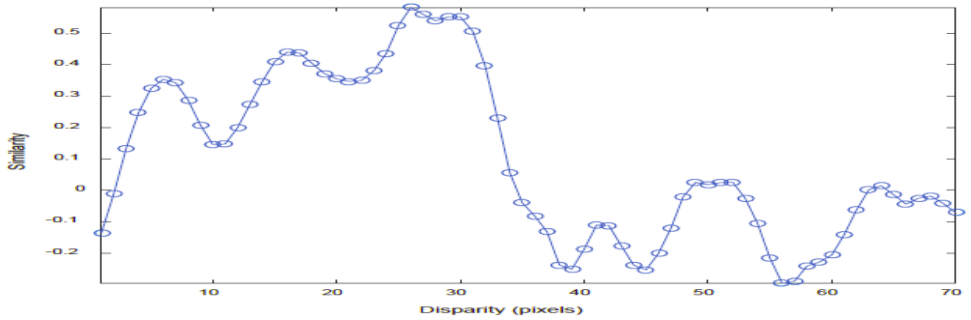
Images credit: Trym Vegard Haavardsholm

# Disparity quality

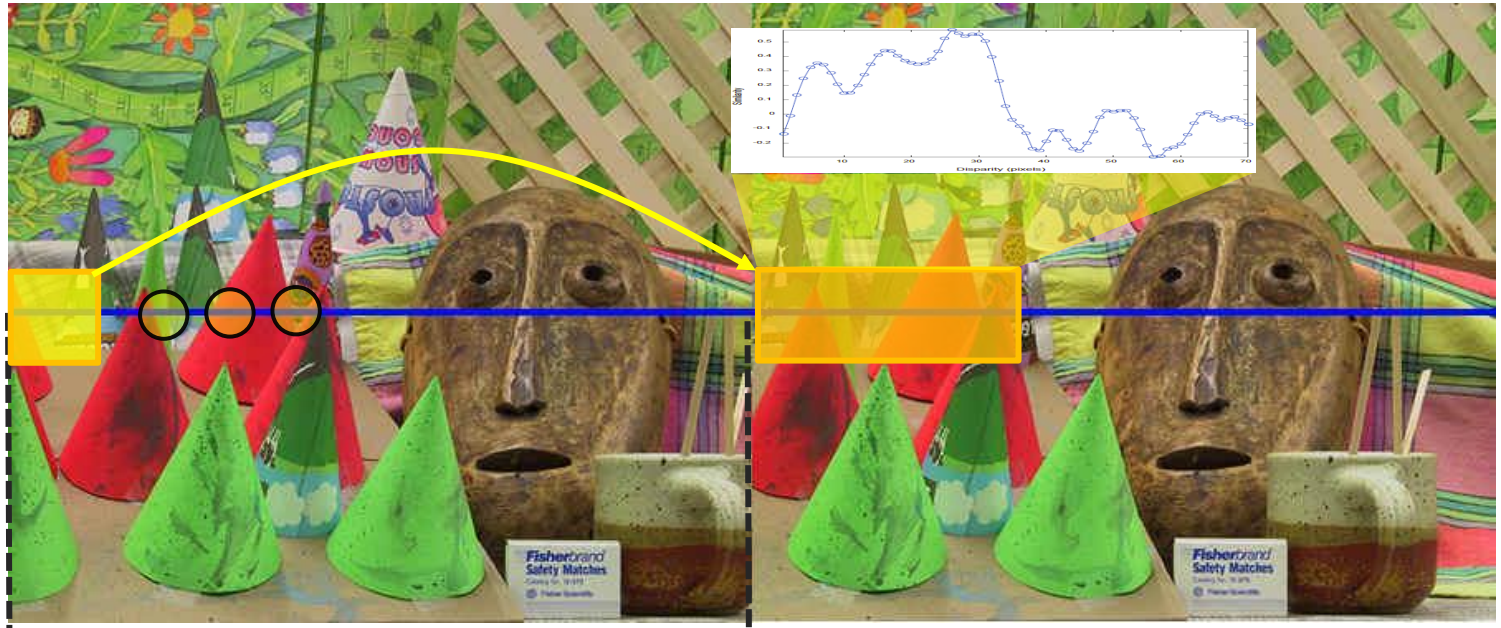


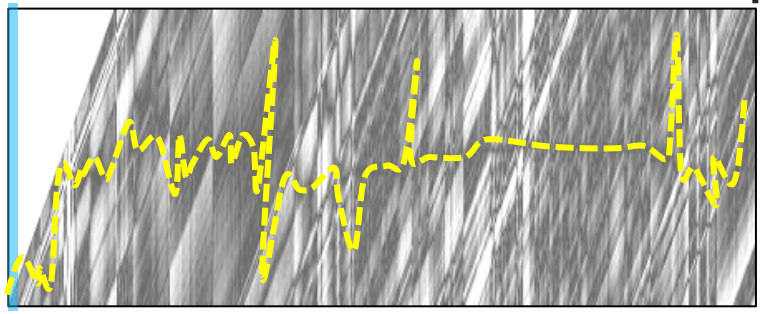

Weak response due to occlusion



# Global disparity optimization




"Energy" output w.r.t. location



$d = 0$

$d_{max}$

$x = 0$

$x = N$

- Consider a single line ($N$ pixels)
- Similarity scores* for different disparities for each pixel.
- Global cost of selecting disparities $d = (d_1 : d_N)$ :

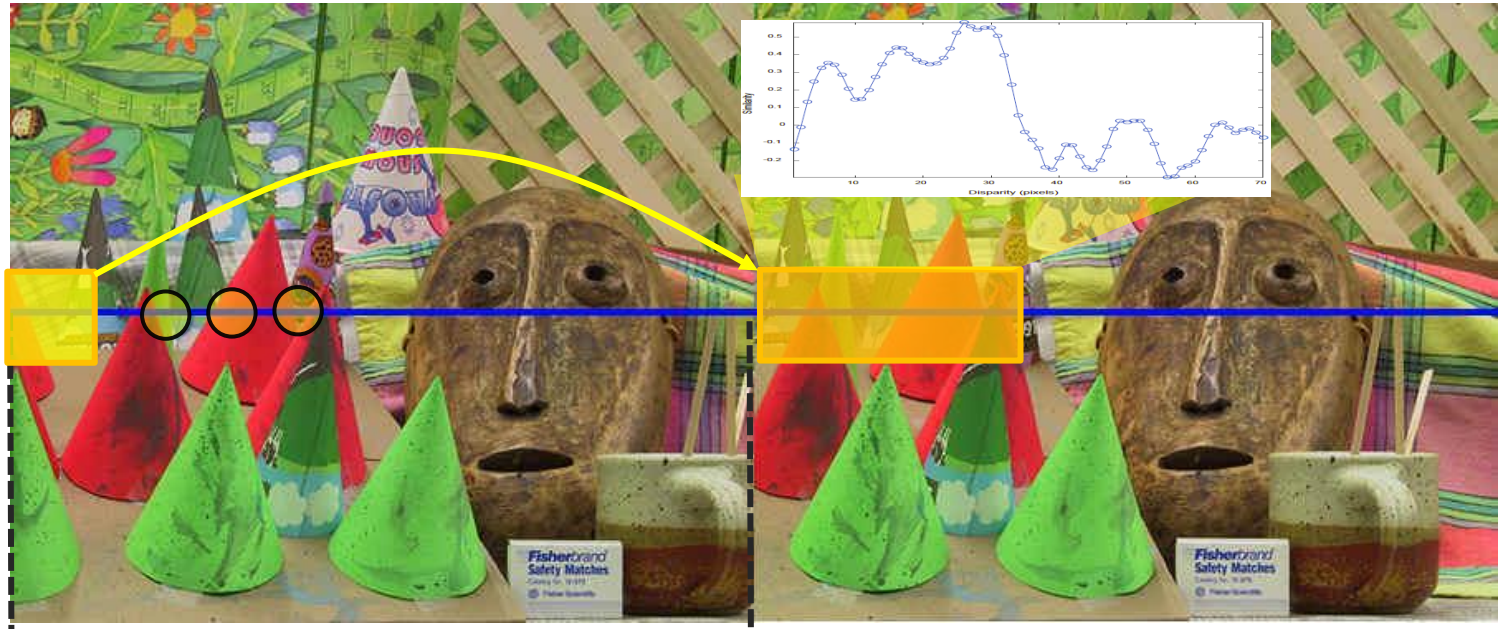
$$E(d_i) = E_{data}(d_i)$$

$$E_{data}(d_i) = e^{-similarity(d_i)}$$

$$E(d) = \sum_i E_{data}(d_i)$$

*Similarity 0 means "no match", 1 means "perfect match"

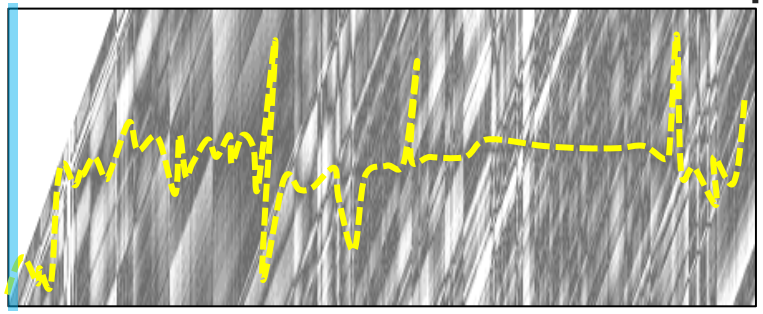
# Global disparity optimization




$$E(d_i) = E_{data}(d_i)$$

$$E_{data}(d_i) = e^{-similarity(d_i)}$$

"Energy" output w.r.t. location



$d = 0$

$d_{max}$

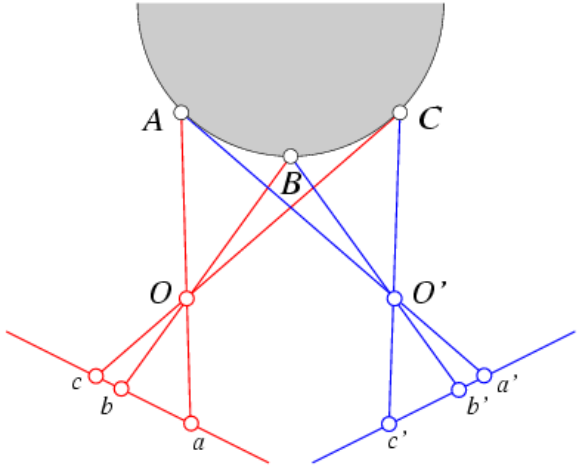
$x = 0$

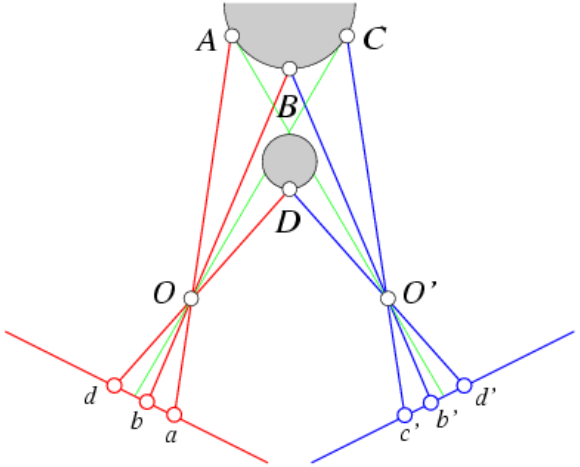
$x = N$

- Disparity calculated independently at each pixel.
- Additional constraints can be imposed on the set of viable disparity estimates.

# Disparity constraints Constraints:

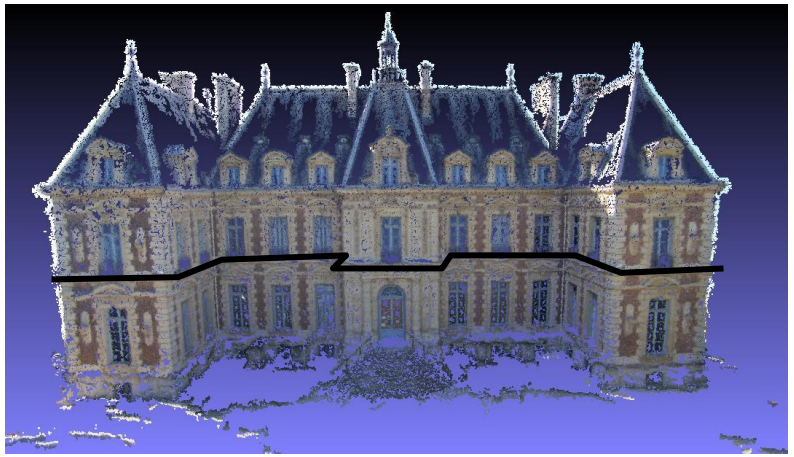
- Order: Points on a single surface appear in the same order in both views.

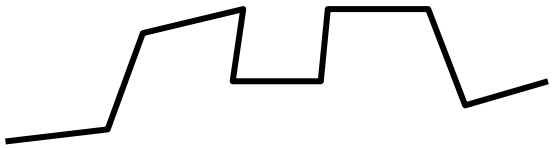




Order of points constraint violated

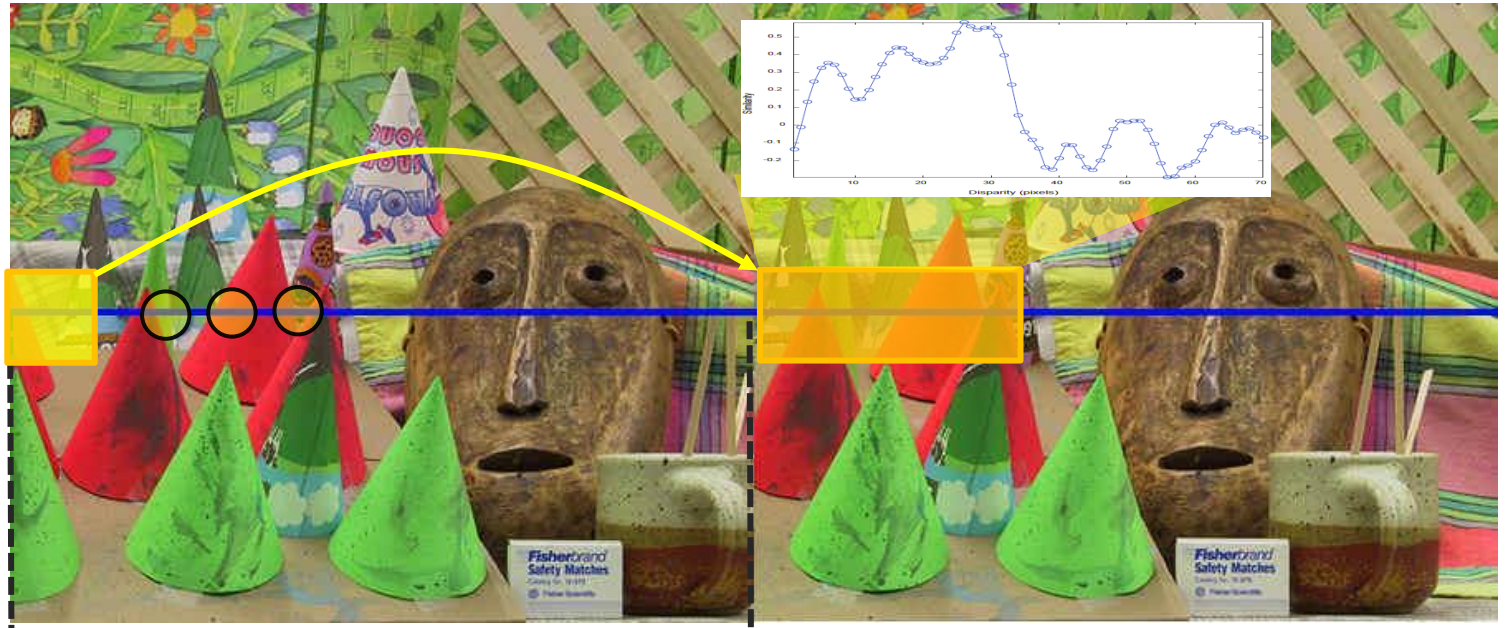
- Slow local depth change: smooth surfaces should result in smooth disparity.



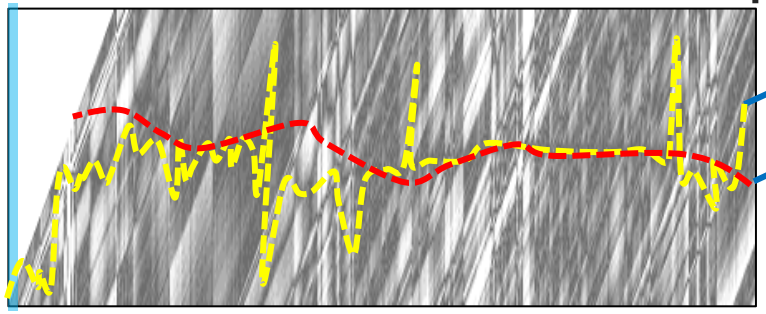


depth

# Global disparity optimization




"Energy" output w.r.t. location



$d = 0$

$d_{max}$

$x = 0$

$x = N$

$E(d)$ = large

$E(d)$ = small

How to find $d$ with globally minimal $E(d)$?

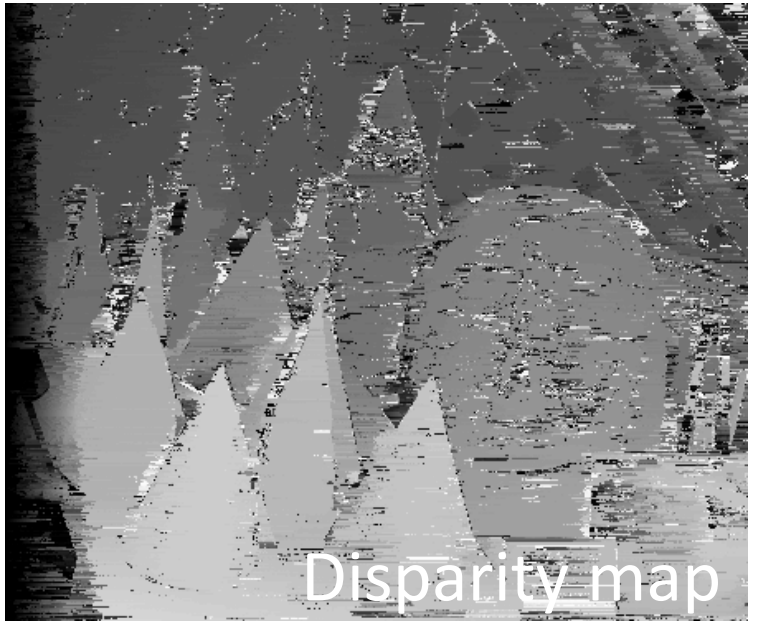
- Consider a single line ($N$ pixels)
- Similarity scores for different disparities for each pixel.
- Global cost of selecting disparities $d = (d_1 : d_N)$ :

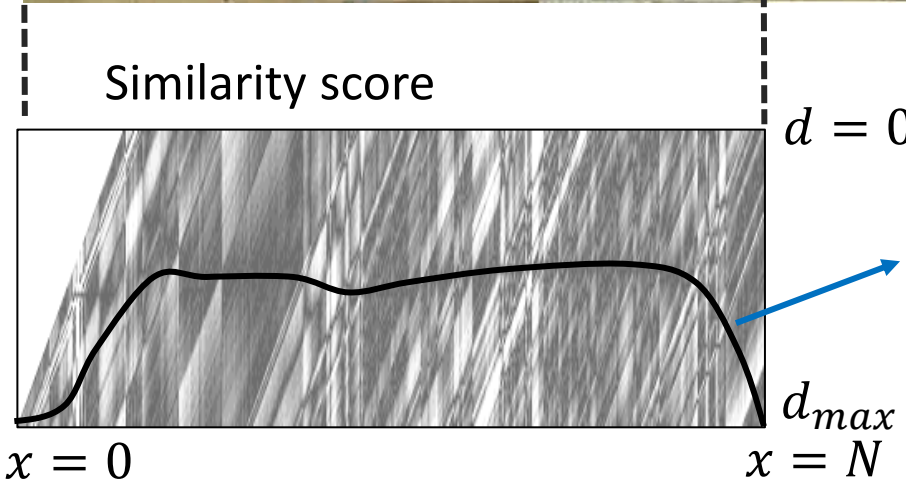
$$E(d_i) = E_{data}(d_i) + \lambda E_S(d_i)$$

$$E_{data}(d_i) = e^{-similarity(d_i)}$$

Smoothness term that assigns a high cost if disparities change significantly between consecutive pixels

# Global disparity optimization



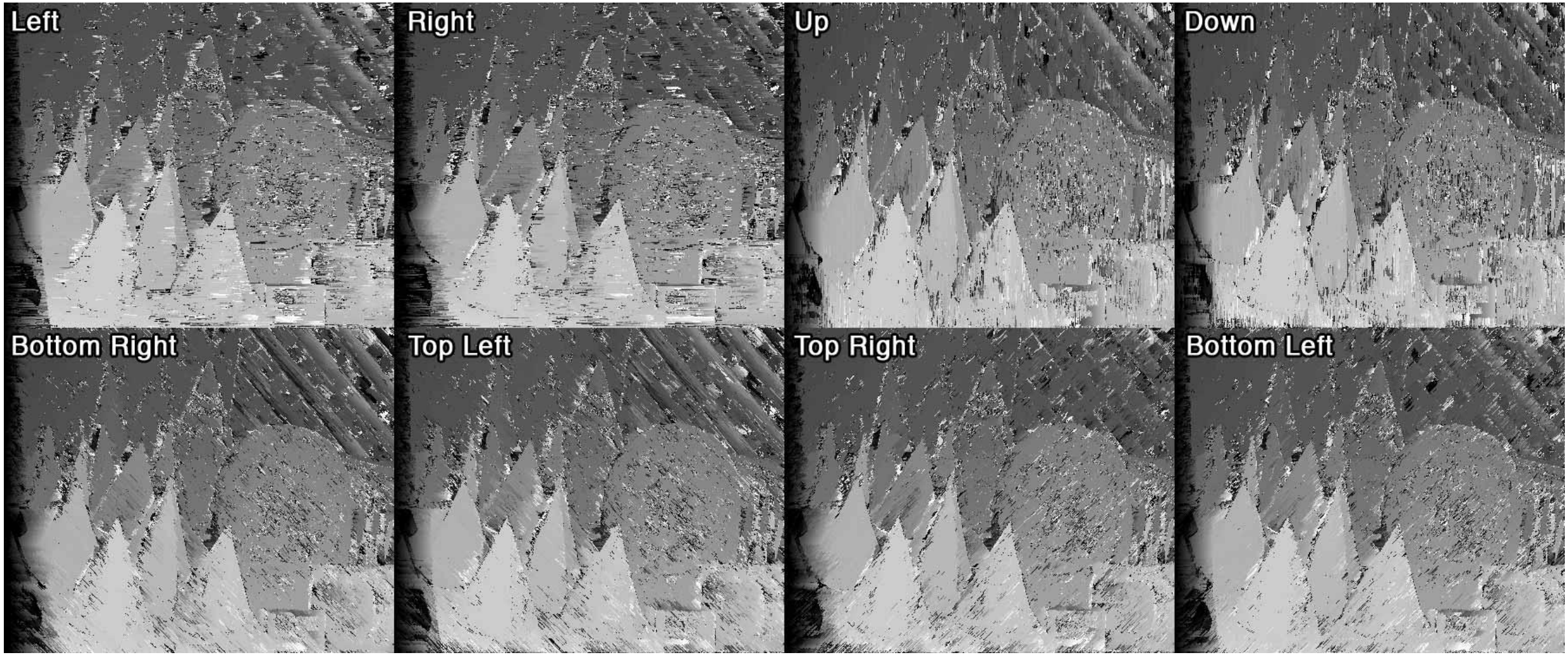




- Optimal sequence of disparities $\mathrm{d}^{\mathrm{opt}} = (d_1 : d_N)$ obtained by dynamic programming (e.g., Viterbi algorithm).
- Apply independently to each line.

Cox, Hingorani, Rao, Maggs, "A Maximum Likelihood Stereo Algorithm," CVIU, 1996.

# Semi global block matching (SGBM)

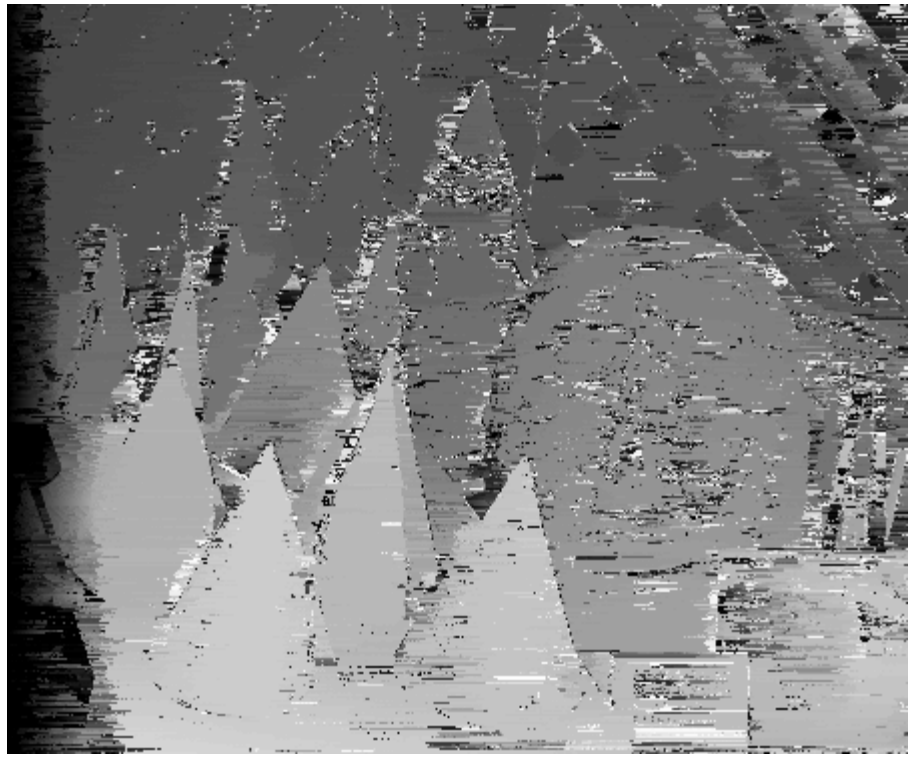
- Apply line-based optimization across several directions in the image …



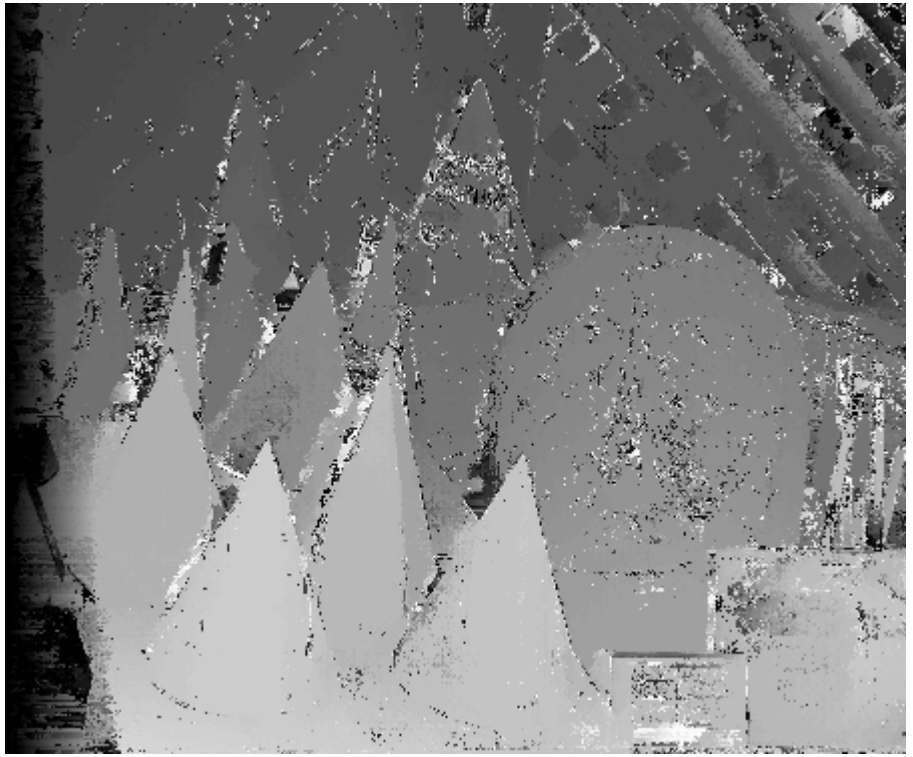
# Semi global block matching (SGBM)

- … aggregate disparity energies from all direction-optimal assignments and take the disparity at each pixel that received a minimum energy.

Left-to-right line optimization



After aggregating 8-direction energies



Heiko Hirschmuller, “Stereo processing by semiglobal matching and mutual information”. *TPAMI*, 2008

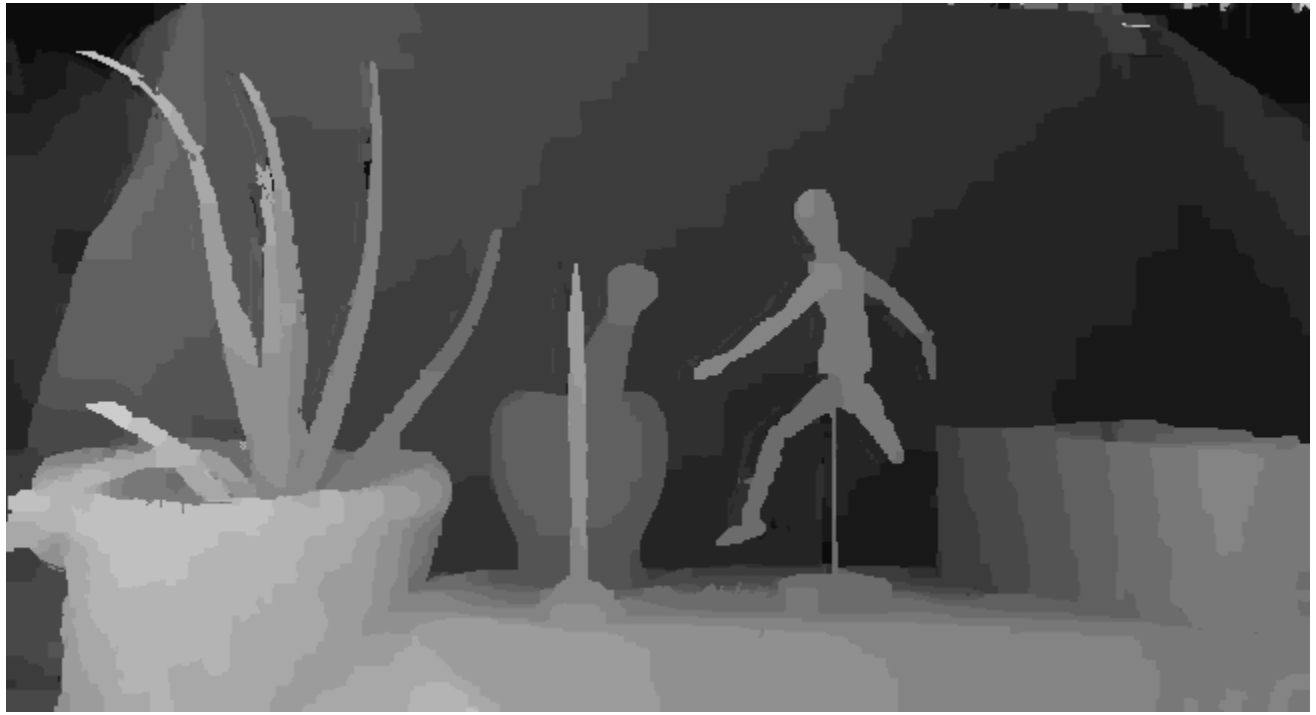
# Application: View interpolation

Right image

# Application: View interpolation

Left image

# Application: View interpolation



Disparity

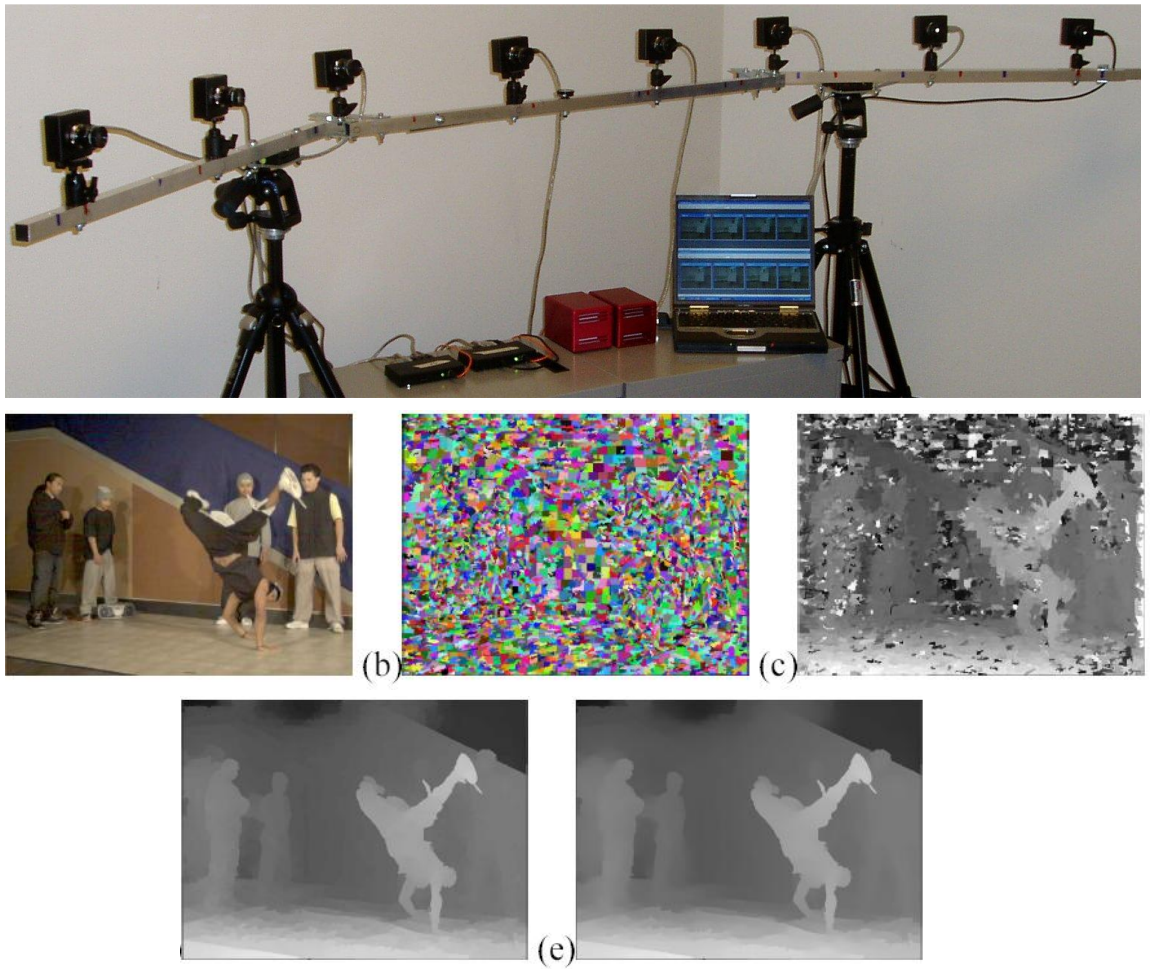
# Application: View interpolation

# Application: view interpolation

- Inpainting required: fill-in the previously occluded scene regions for better realism

Kopf et al., One Shot 3D Photography, Arxiv 2008

# Video view interpolation

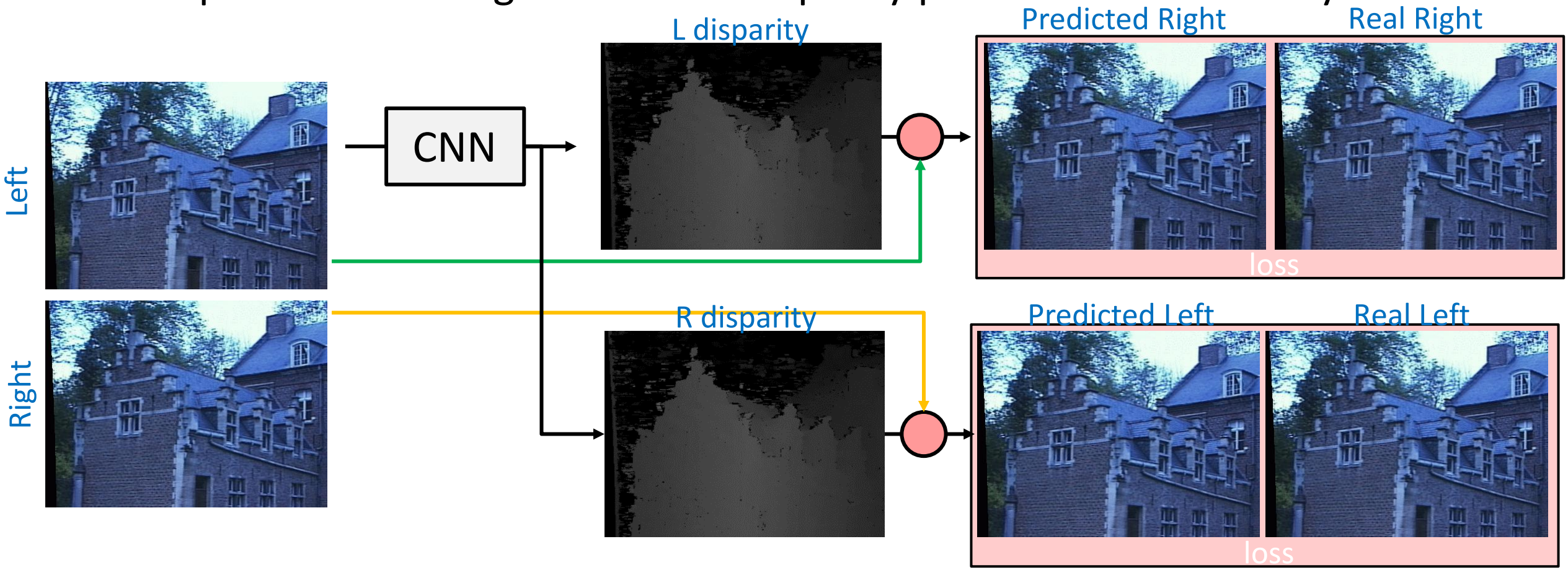



L. Zitnick et al, High-quality video view interpolation using a layered representation, SIGGRAPH 2004

http://research.microsoft.com/IVM/VVV/

# Avoiding stereo: Mono-depth – a basic approch

- Train a CNN to predict depth based on a *single* image
- Unsupervised training: use stereo disparity prediction consistency




Godard et al., Unsupervised Monocular Depth Estimation with Left-Right Consistency, CVPR2017

# Monodepth 2

Godard et al., Digging Into Self-Supervised Monocular Depth Estimation, ICCV2019 [GIT]

# Reconstruction by a moving camera

- If a camera is moving freely, the “stereo system” cannot be pre-calibrated (except from matrix $\boldsymbol{K}$, that is)
- Actually, we are dealing with multiple “cameras”

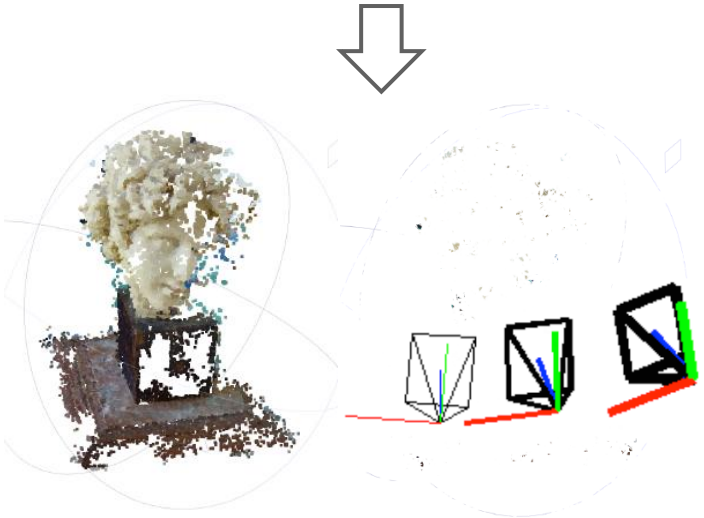
https://www.youtube.com/watch?v=5QLutxstkw4

Machine perception

# RECONSTRUCTION FROM MULTIPLE IMAGES
## A BRIEF OVERVIEW

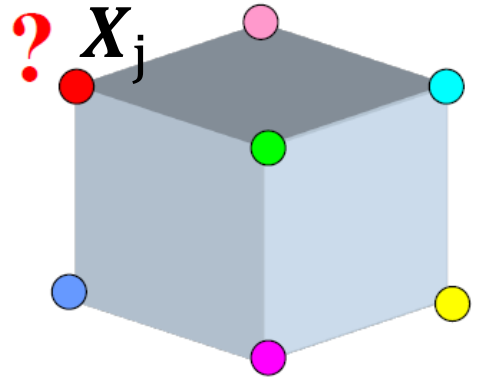
# Problem formulation

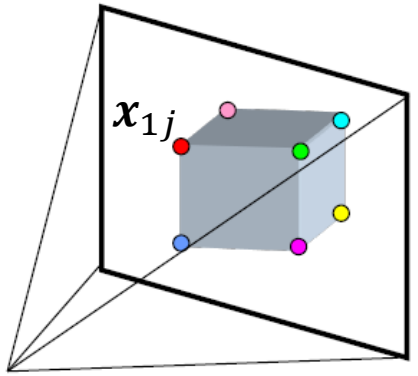
- Given several images of the same scene, reconstruct all camera parameters (poses $\boldsymbol{R}_\mathrm{i}, \boldsymbol{t}_\mathrm{i}$ & calibration matrices $\boldsymbol{K}_\mathrm{i}$) **and** the 3D points $\mathrm{X}_\mathrm{j}$



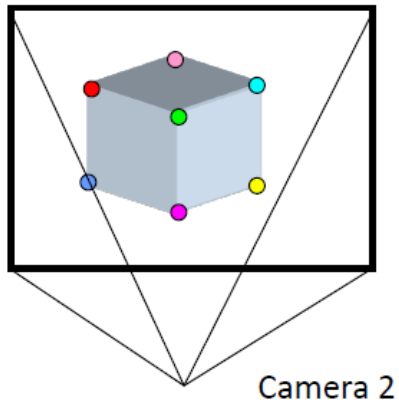
$$\mathrm{P}_\mathrm{i} = \boldsymbol{K}[\boldsymbol{R}_\mathrm{i}|\boldsymbol{t}_\mathrm{i}]$$

$$\boldsymbol{x}_{ij} = \boldsymbol{P}_\mathrm{i}\mathrm{X}_\mathrm{j}$$




Camera 1
$\boldsymbol{K}_1, \boldsymbol{R}_1, \boldsymbol{t}_1$ ?



Camera 2
$\boldsymbol{K}_2, \boldsymbol{R}_2, \boldsymbol{t}_2$

# A simple structure from motion (SFM) pipeline

- Assume a moving camera with known calibration matrix $\boldsymbol{K}$
- Assume a two-view problem (easily extends to multiple views)
- We will consider a pipeline that's easier to explain (but not used in practice)

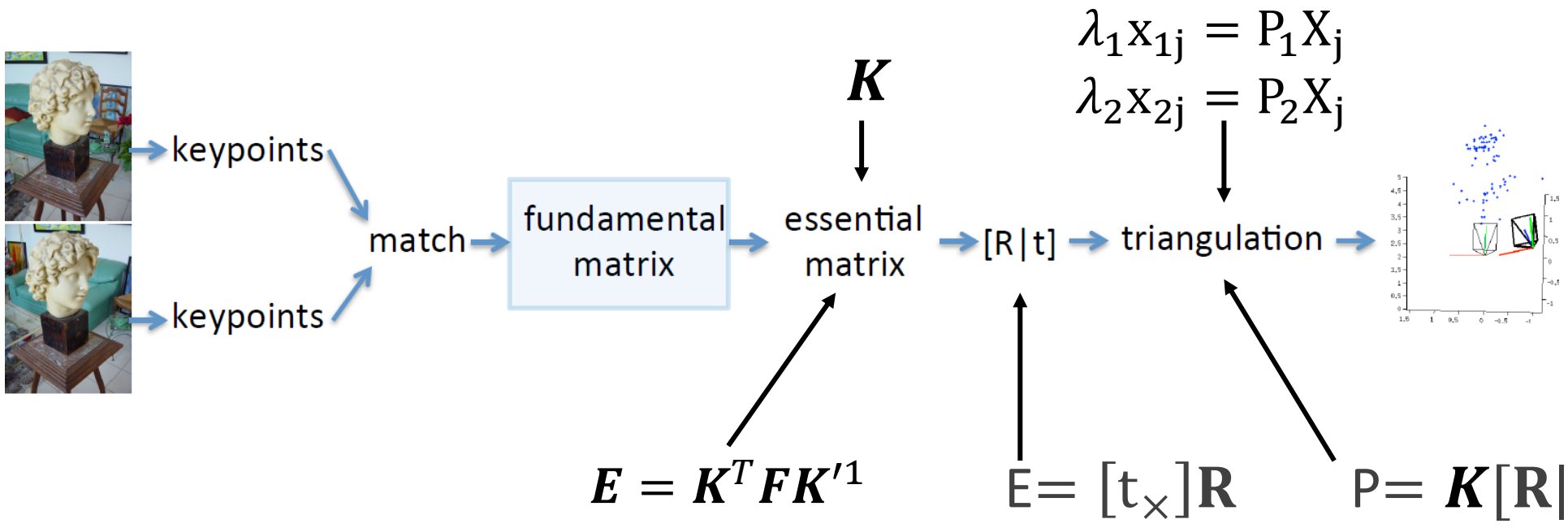
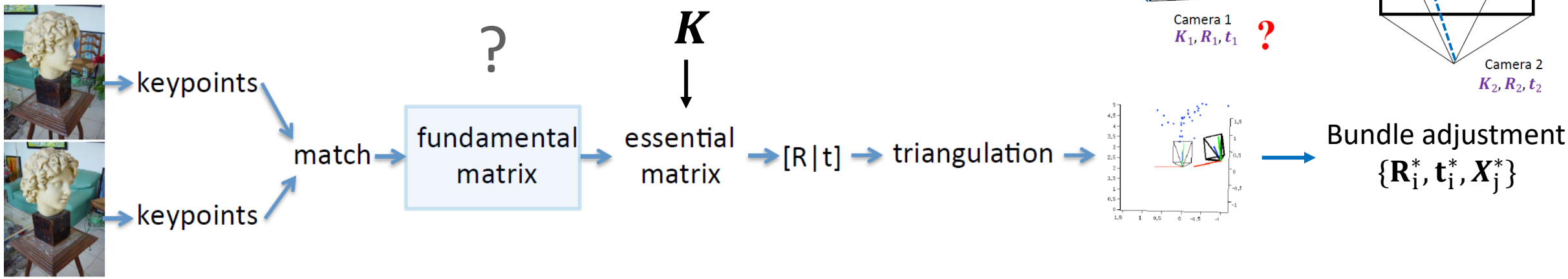
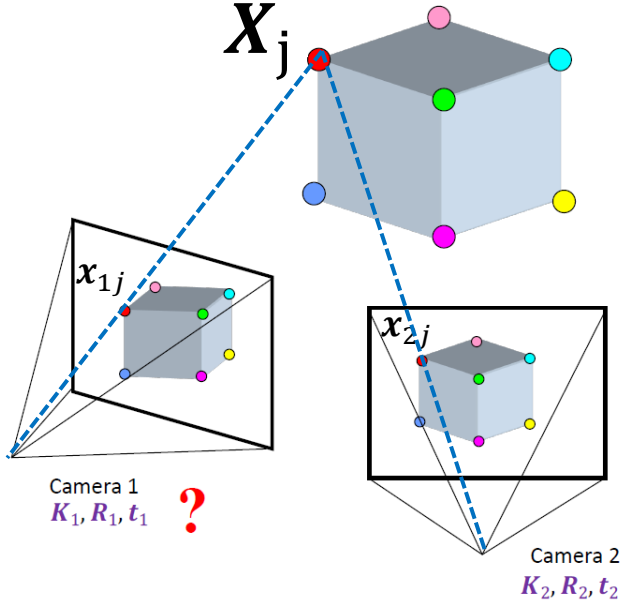



Image from: Xiao, J. Multiview 3D Reconstruction for Dummies

# A simple structure from motion (SFM) pipeline

- Finally, using $\mathrm{P_i} = \boldsymbol{K}[\boldsymbol{R}_\mathrm{i}|\boldsymbol{t}_\mathrm{i}]$, optimize the camera parameteres & 3D point positions by minimizing the re-projection errors: $\epsilon_{\mathrm{ij}} = (\mathbf{x}_{ij} - \boldsymbol{P}_\mathrm{i}\mathrm{X_j})^2$




Bundle adjustment = Minimize re-projection error

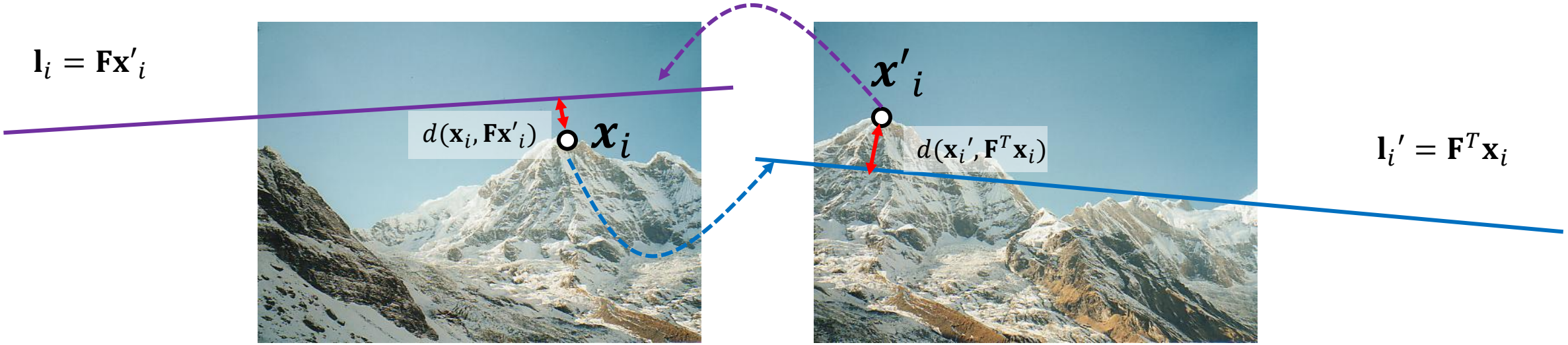
$$\{\mathbf{R}_\mathrm{i}^*, \mathbf{t}_\mathrm{i}^*, \boldsymbol{X}_\mathrm{j}^*\} = \mathrm{argmin} \sum_i \sum_j (\mathbf{x}_{ij} - \mathbf{K}_i[\mathbf{R}_i|\mathbf{t}_i]\mathbf{X}_j)^2$$

$\boldsymbol{x}_{ij}$ should be close to $\boldsymbol{P}_\mathrm{i}\mathrm{X_j}$:

Image from: Xiao, J. Multiview 3D Reconstruction for Dummies

# Fundamental matrix estimation

- Assume known correspondences $\{x_i\}_{i=1:N}, \{x'_i\}_{i=1:N}$ $\quad \mathrm{x}^T \mathrm{F}\mathrm{x}' = 0$
- Estimate $F$ that minimizes reprojection errors $\epsilon(\mathbf{F})$




$$\epsilon(\mathbf{F}) = \frac{1}{N}\sum_{i=1}^{N}(d^2(\mathbf{x}_i, \mathbf{F}\mathbf{x}'_i) + d^2(\mathbf{x}_i', \mathbf{F}^T\mathbf{x}_i))$$

- Nonlinear optimization (Levenberg-Marquardt), requires good initial estimate.
- Usually initialized by 8-point algorithm (described next).

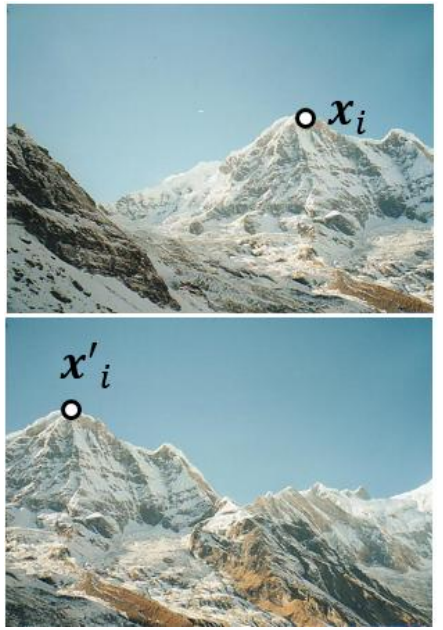
# Fundamental matrix estimation: The Eight-point algorithm

Coordinates of a pair of corresponding points: $\boldsymbol{x} = (u, v, 1)^T$, $\boldsymbol{x'} = (u', v', 1)^T$

Epipolar constraint: $\boldsymbol{x}^T \boldsymbol{F} \boldsymbol{x}' = \boldsymbol{0}$

$$(u, v, 1)\begin{pmatrix} F_{11} & F_{12} & F_{13} \\ F_{21} & F_{22} & F_{23} \\ F_{31} & F_{32} & F_{33} \end{pmatrix}\begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = 0 \quad \Rightarrow \quad (uu', uv', u, vu', vv', v, u', v', 1)\begin{pmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{pmatrix} = 0$$

$$F_{33} + F_{13}u + F_{31}u' + F_{23}v + F_{32}v' + F_{11}uu' + F_{12}uv' + F_{21}u'v + F_{22}vv' = 0$$




(one equation per correspondence – require 8)

$$\begin{bmatrix} u_1u'_1 & u_1v'_1 & u_1 & v_1u'_1 & v_1v'_1 & v_1 & u'_1 & v'_1 & 1 \\ u_2u'_2 & u_2v'_2 & u_2 & v_2u'_2 & v_2v'_2 & v_2 & u'_2 & v'_2 & 1 \\ u_3u'_3 & u_3v'_3 & u_3 & v_3u'_3 & v_3v'_3 & v_3 & u'_3 & v'_3 & 1 \\ u_4u'_4 & u_4v'_4 & u_4 & v_4u'_4 & v_4v'_4 & v_4 & u'_4 & v'_4 & 1 \\ u_5u'_5 & u_5v'_5 & u_5 & v_5u'_5 & v_5v'_5 & v_5 & u'_5 & v'_5 & 1 \\ u_6u'_6 & u_6v'_6 & u_6 & v_6u'_6 & v_6v'_6 & v_6 & u'_6 & v'_6 & 1 \\ u_7u'_7 & u_7v'_7 & u_7 & v_7u'_7 & v_7v'_7 & v_7 & u'_7 & v'_7 & 1 \\ u_8u'_8 & u_8v'_8 & u_8 & v_8u'_8 & v_8v'_8 & v_8 & u'_8 & v'_8 & 1 \end{bmatrix}\begin{bmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Homogeneous system!

$$\mathbf{Af} = \mathbf{0}$$

Minimize:

$$\sum_{i=1}^{N} (x_i^T F x'_i)^2$$

with constraint

$$||F||^2 = 1$$

$F \leftarrow$ last eigenvector(A)

# Normalized 8-point algorithm

1. *Precondition:* Center image points, and scale such that the standard deviation becomes $\sqrt{2}$ pixels.
   - $\tilde{\boldsymbol{x}} = \boldsymbol{T}\boldsymbol{x},\ \tilde{\boldsymbol{x}}' = \boldsymbol{T}'\boldsymbol{x}'$
2. Apply 8-point algorithm to calculate $\widetilde{\boldsymbol{F}}$ from the preconditioned points.
3. Enforce rank=2

   (decompose $\widetilde{\boldsymbol{F}}$, by SVD set the smallest singular value to zero and reconstruct $\widetilde{\boldsymbol{F}}$):

   $$\widetilde{\boldsymbol{F}} \overset{\text{SVD}}{=} \mathbf{U}\mathbf{D}\mathbf{V}^T = \mathbf{U}\begin{bmatrix} d_{11} & & \\ & d_{22} & \\ & & d_{33} \end{bmatrix}\begin{bmatrix} v_{11} & \cdots & v_{13} \\ \vdots & \ddots & \vdots \\ v_{31} & \cdots & v_{33} \end{bmatrix}^T$$

   Set $d_{33}$=0 and reconstruct F: $\widetilde{\boldsymbol{F}} = \mathbf{U}\begin{bmatrix} d_{11} & & \\ & d_{22} & \\ & & 0 \end{bmatrix}V^T$
4. Transform the fundamental matrix back to original units:

   Let T and T' be the transformations used to precondition the points in each image separately. Then the fundamental matrix equals to $\boldsymbol{F} = \boldsymbol{T}'^T\widetilde{\boldsymbol{F}}\boldsymbol{T}$.

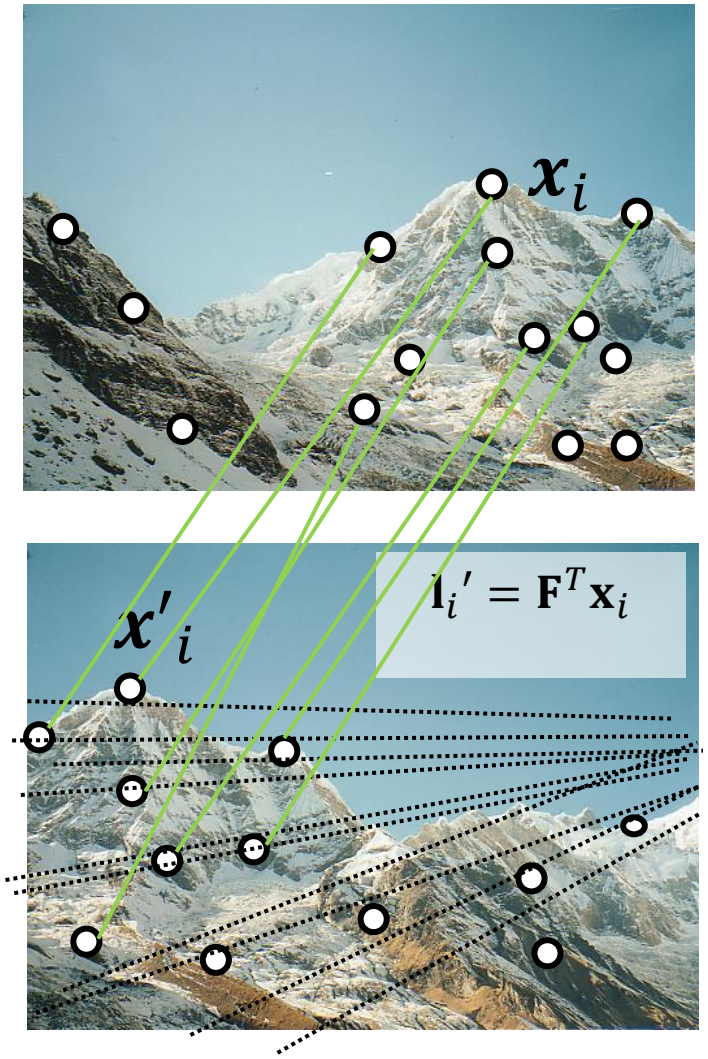
# Fundamental matrix estimation

- In general, the correspondences are unknown
  - Jointly find the fundamental matrix F *AND* the correspondences!
    (pairs across two views (x',y') ↔ (x,y)).

- Approach
  1. Find keypoints in each image
  2. Calculate possible matches (potential matches)
  3. Robustly estimate the epipolar geometry by RANSAC

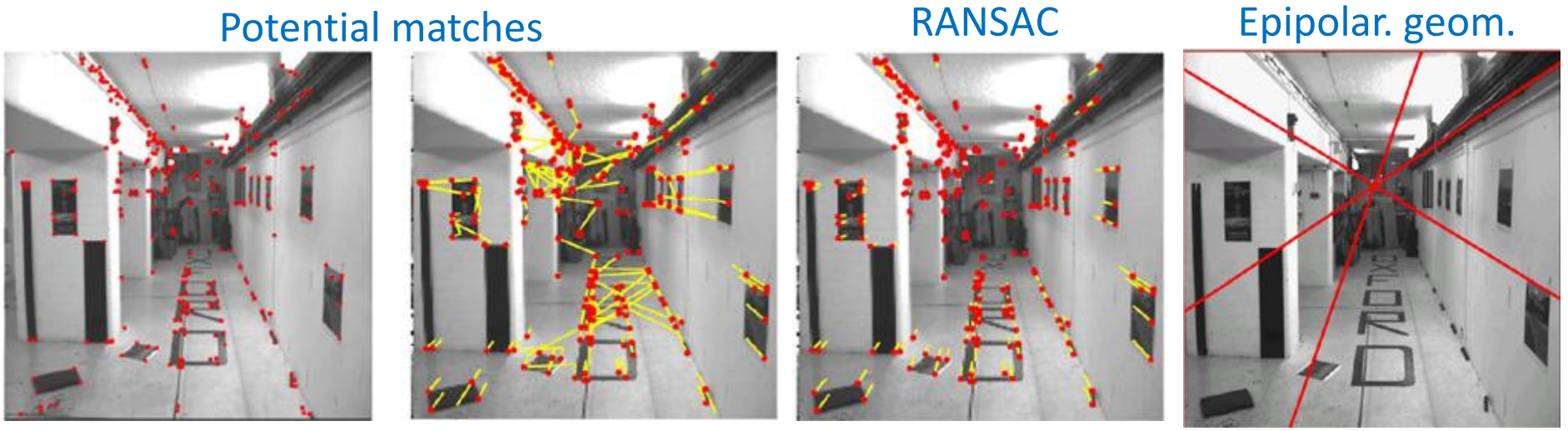
# RANSAC to robustly estimate *F*

- Randomly select a set of 8 correspondences
- Calculate F using these correspondences
  - This gives the epipolar constraint!
- Estimate how many correspondences support F:
  - Apply the estimated fundamental matrix to all points in image $I_1$ and compute their epipolar lines in image $I_2$.
  - Number of inliers: points in $I_2$ that lie close to their epipolar lines calculated from $\boldsymbol{F}$ and corresponding points from $I_1$.
- Choose $\boldsymbol{F}$ with maximal support (#inliers)



# Fundamental matrix estimation summary

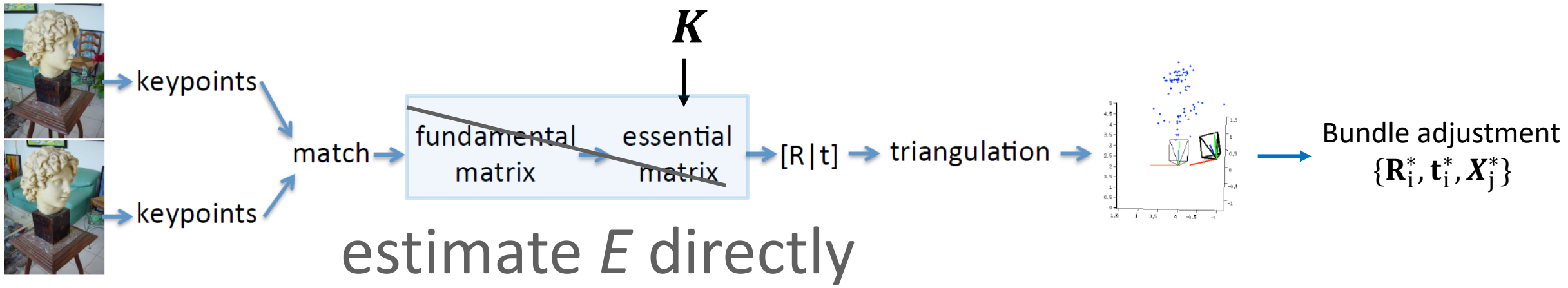
- Robust estimation of F




- Improve by a nonlinear optimization of the cost function w.r.t. $\boldsymbol{F}$ using inliers only:

$$\epsilon(\mathbf{F}) = \frac{1}{N}\sum_{i=1}^{N}(d^2(\mathbf{x}_i, \mathbf{F}\mathbf{x'}_i) + d^2(\mathbf{x}_i', \mathbf{F}^T\mathbf{x}_i))$$

# A simple structure from motion (SFM) pipeline




- But, if $\boldsymbol{K}$ is actually, known, we can estimate $\boldsymbol{E}$ directly in RANSAC, avoiding computation of $\boldsymbol{F}$.
  - Transform pixels to image-plane points: $\boldsymbol{x}_p = \boldsymbol{K}^{-1}\boldsymbol{x}$ , $\boldsymbol{x}'_p = \boldsymbol{K}'^{-1}\boldsymbol{x}'$
  - Calculate $\boldsymbol{E}$ from *5* correspondences using the *5-point algorithm* [Nister, TPAMI 2004]. (solves a 10-degree polinomial)

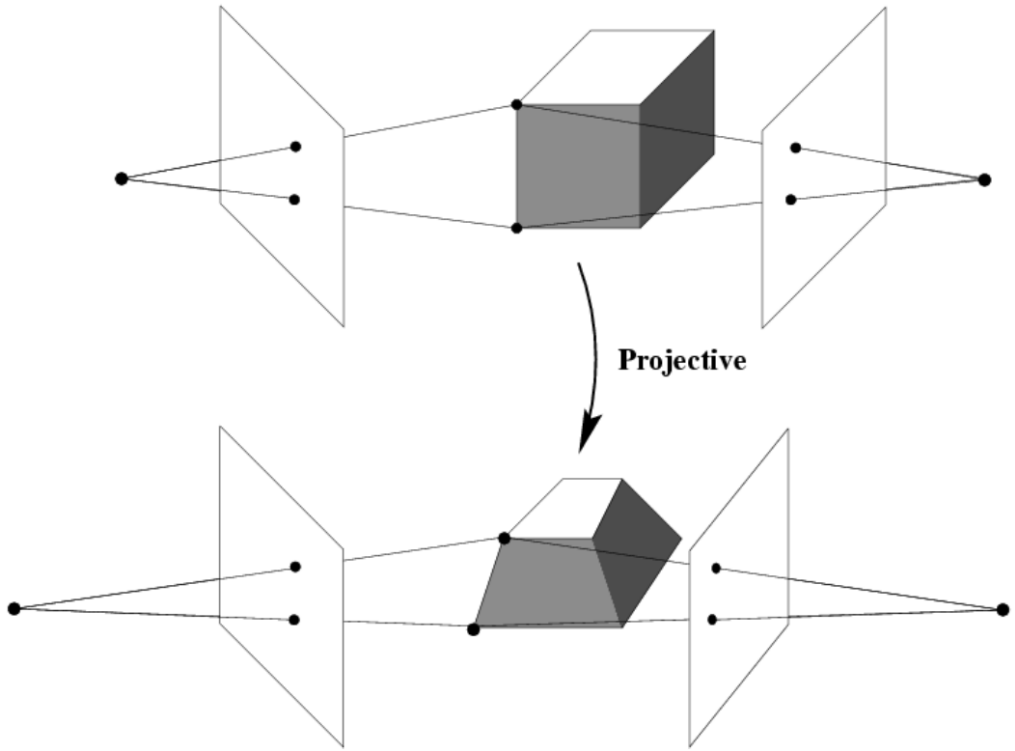
# Uncalibrated structure from motion (SFM)


keypoints → match → Estimate F → Projective cameras $P_1, P_2$

keypoints → match

Without constraints on projection matrices or the scene, the estimation is accurate up to a projective invertible matrix $4 \times 4$ matrix $\boldsymbol{Q}$:

$$\boldsymbol{x}_{ij} = (\boldsymbol{P}_{\mathrm{i}}\boldsymbol{Q}^{-1})(\boldsymbol{Q}\boldsymbol{X}_j)$$

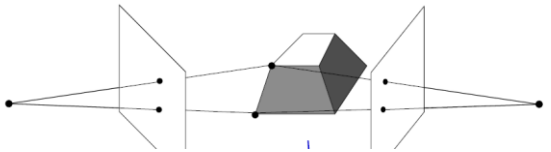


# Uncalibrated structure from motion (SFM)

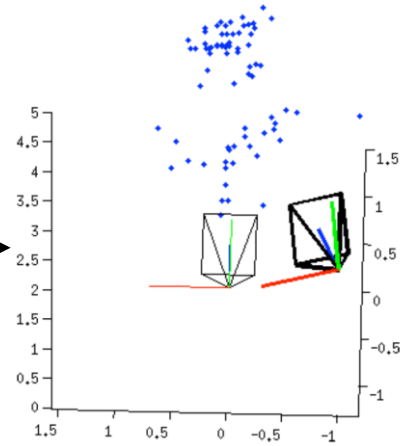
keypoints

keypoints

match → Estimate F →



Projective cameras $P_1, P_2$ → Triangulate →




Bundle adjustment on $\{\boldsymbol{P}_\mathrm{i}\}\{\boldsymbol{X}_\mathrm{j}\}$

Self-calibrate: Estimate $\boldsymbol{K}$

Upgrade projection matrices $\boldsymbol{P}_\mathrm{i} = \boldsymbol{K}[\boldsymbol{R}_\mathrm{i}|\boldsymbol{t}_\mathrm{i}]$

Bundle adjustment $\{\mathbf{R}_\mathrm{i}^*, \mathbf{t}_\mathrm{i}^*, \boldsymbol{X}_\mathrm{j}^*\}$

# Try multi-view reconstruction at home

- Bundler: Structure from Motion (SfM) for Unordered Image Collections

Software written by Noah Snavely
**Download Bundler from the bundler_sfm repository on GitHub**

| What is Bundler? | Downloading Bundler | Documentation | References | Links |

**What is Bundler?**

Bundler is a structure-from-motion (SfM) system for unordered image collections (for instance, images from the Internet) written in C and C++. An earlier version of this SfM system was used in the Photo Tourism project. **For structure-from-motion datasets, please see the BigSFM page.**

Bundler takes a set of images, image features, and image matches as input, and produces a 3D reconstruction of camera and (sparse) scene geometry as output. The system reconstructs the scene incrementally, a few images at a time, using a modified version of the Sparse Bundle Adjustment package of Lourakis and Argyros as the underlying optimization engine. Bundler has been successfully run on many Internet photo collections, as well as more structured collections.

The Bundler source distribution also contains potentially userful implementations of several computer vision algorithms, including:

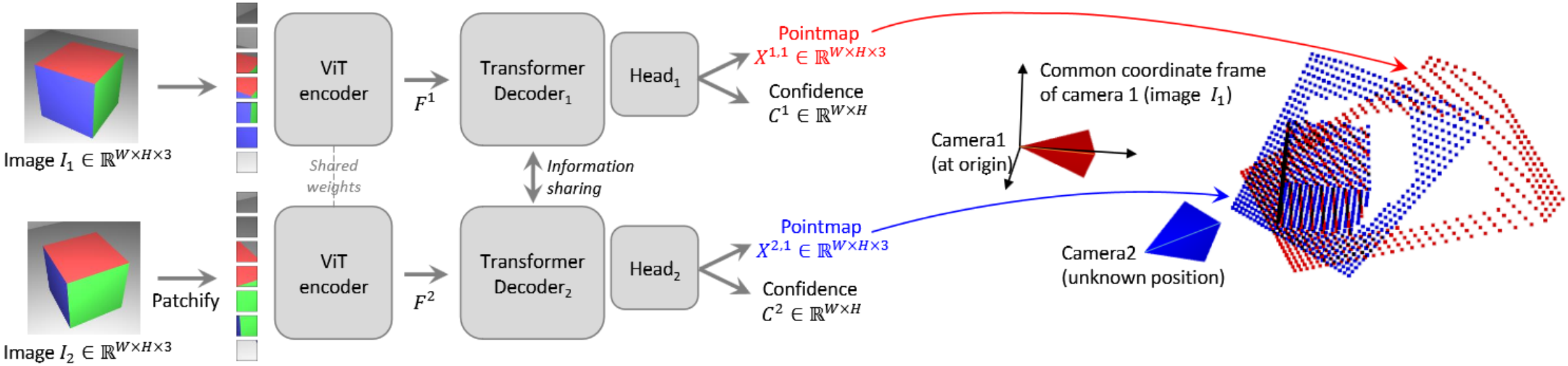
- F-matrix estimation
- Calibrated 5-point relative pose
- Triangulation of multiple rays

PhotoTurism video on YouTube

https://www.youtube.com/watch?v=5rYyB4pKPRo

# Recent deep network DUSt3R: A 3D foundation model

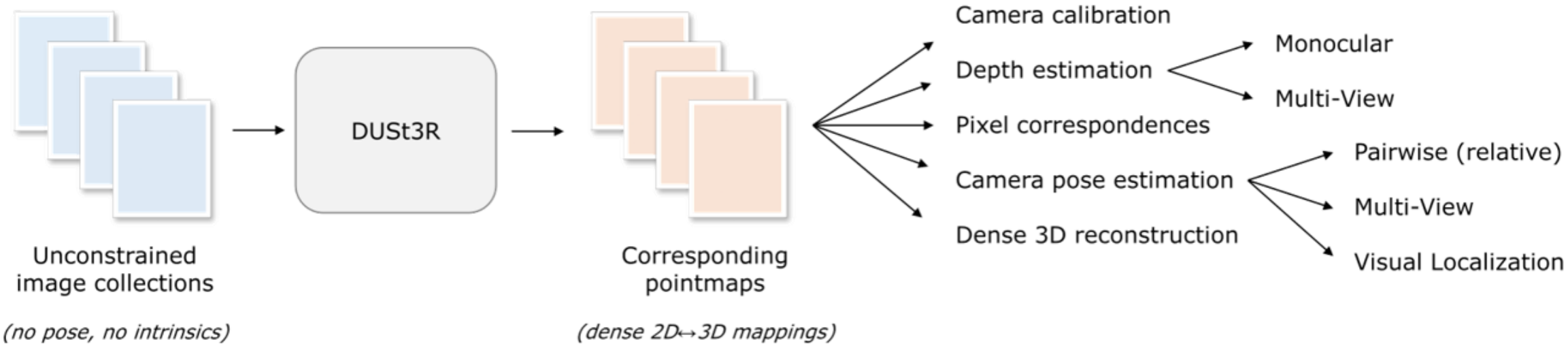
- Two images processed, for each predict the point cloud in the left camera coordinate system

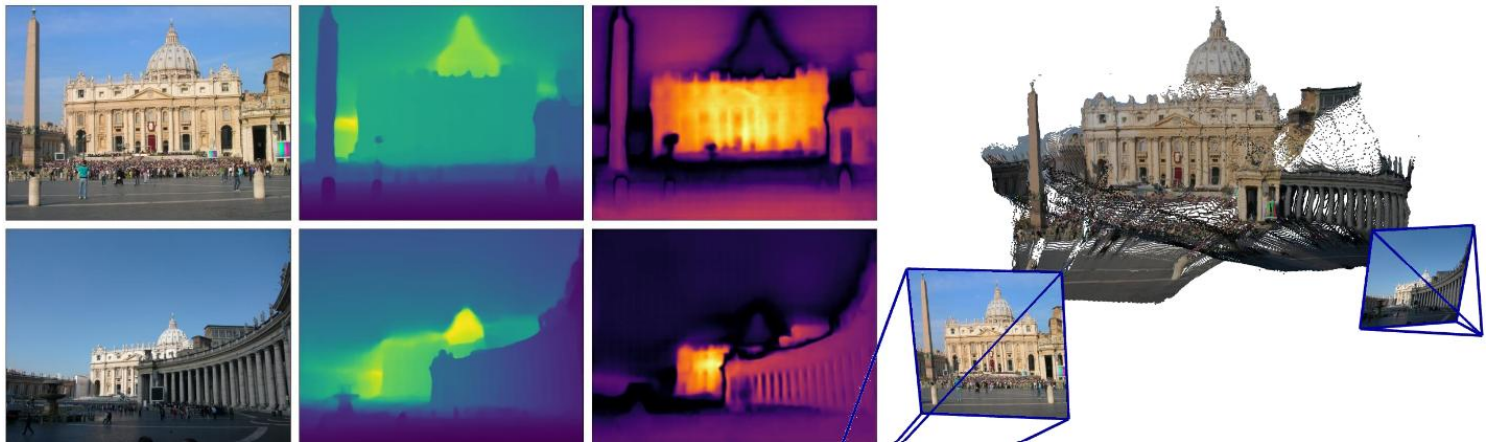



- To get correspondences, just match the points in 3D – i.e., for each point in the left camera 3D point cloud, find the closest point in the right camera 3D point cloud.

# Recent deep network DUSt3R: A 3D foundation model

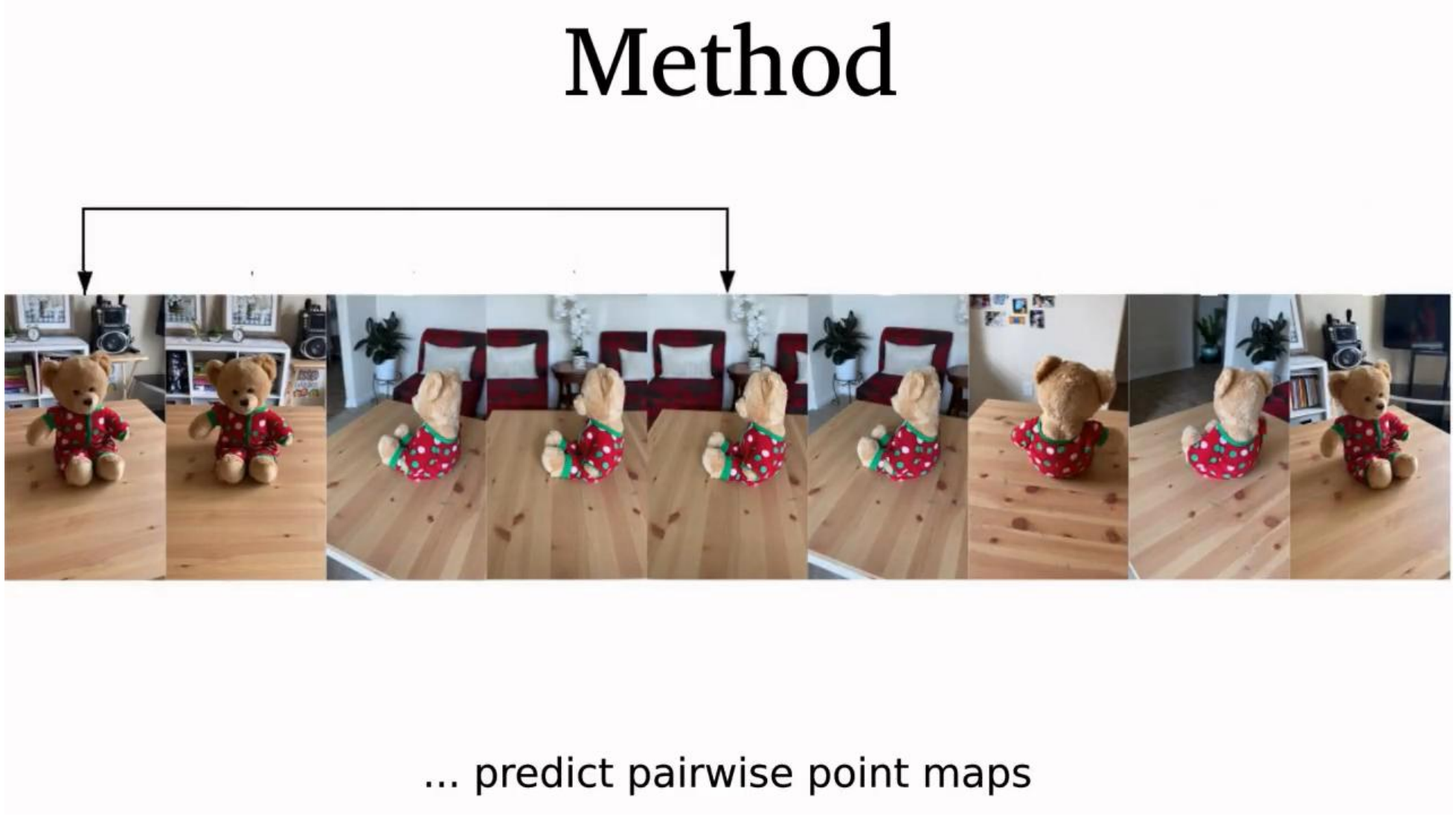
- Pretrained on huge datasets, allows many downstream 3D reconstruction tasks



https://github.com/naver/dust3r

# Recent deep network DUSt3R: A 3D foundation model




Awesome demo: https://europe.naverlabs.com/research/publications/dust3r-geometric-3d-vision-made-easy/

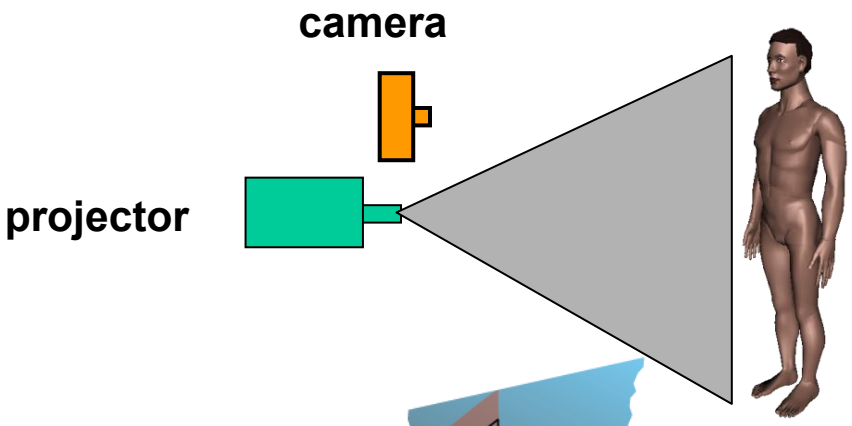
Machine perception

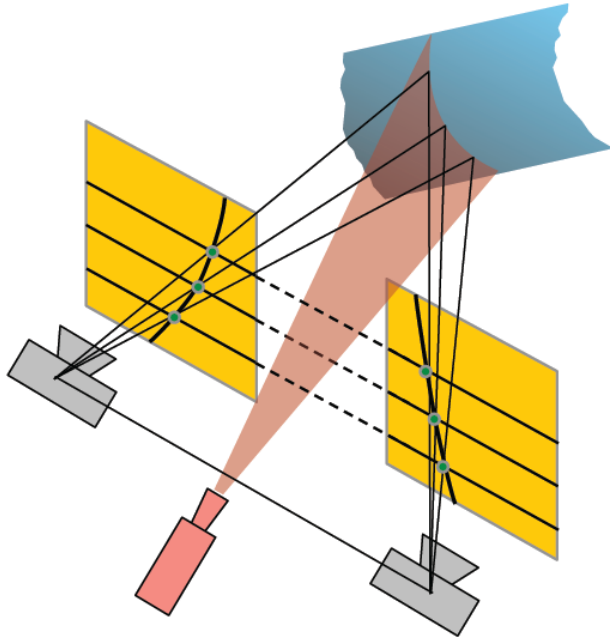
# ACTIVE STEREO

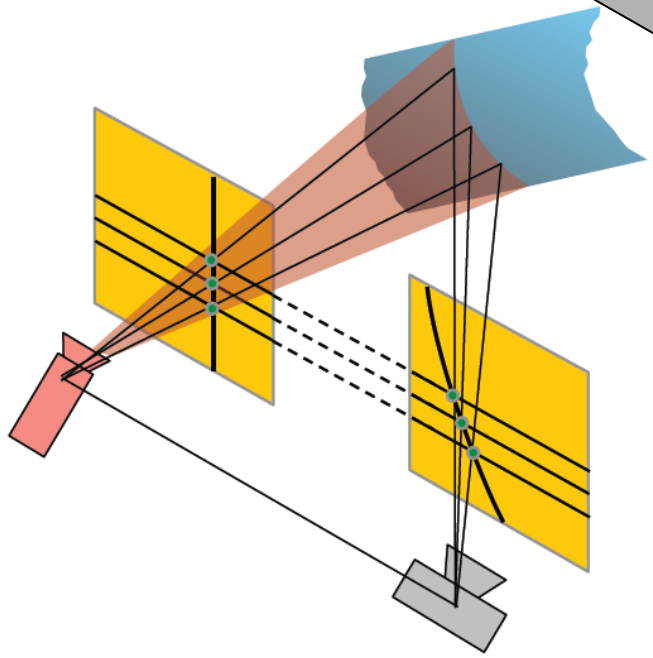
# Structured light stereo

- Idea: project „structured“ light patterns over the object
  - Correspondence problem simplifies
  - Can use only a single camera

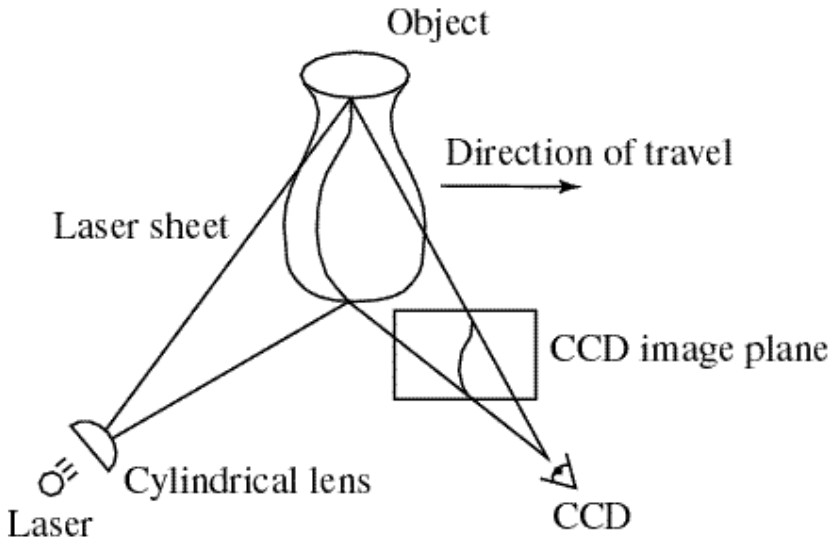
  Epipolar geometry still holds!

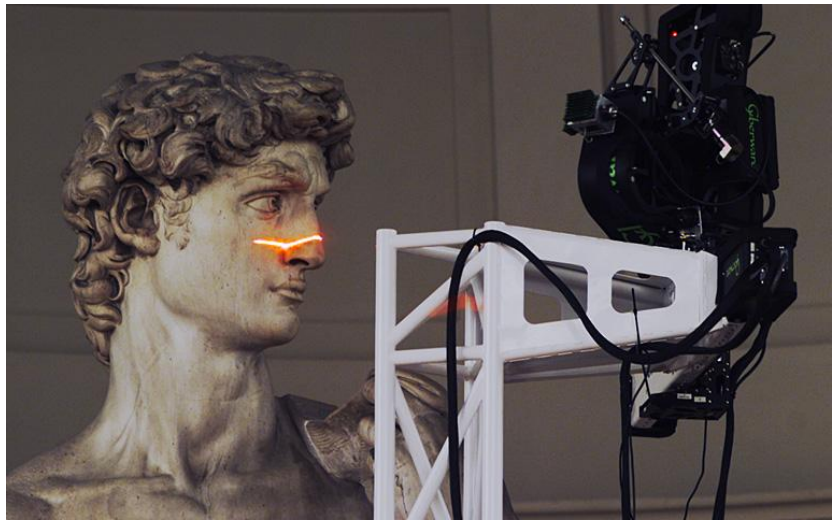




# Laser scanning



Digital Michelangelo Project
http://graphics.stanford.edu/projects/mich/

- Optical triangulation
  - Project a laser light plane
  - Move over an object (the motion has to be accurately measured!)
  - Very precise way to scan using structured light.

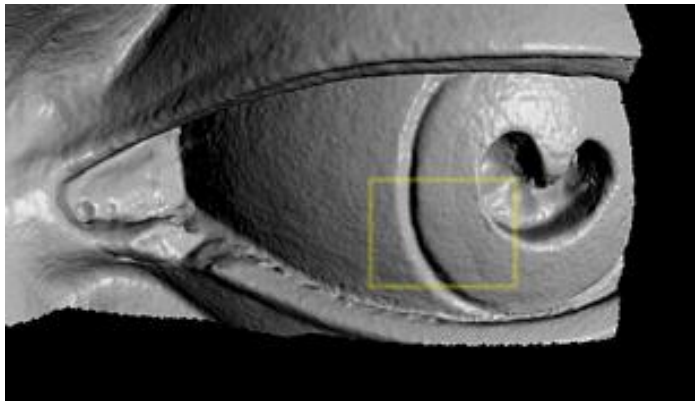
# Obtained models

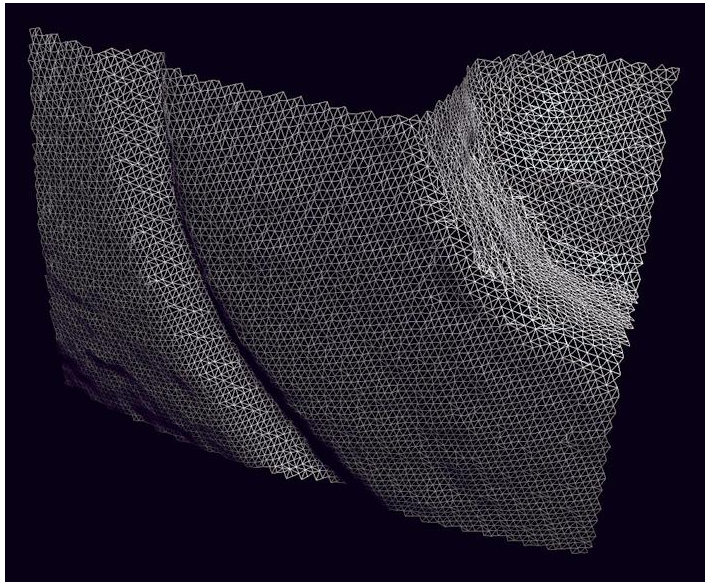
*Michelangelo's David (Florence)*

*The Digital Michelangelo Project*, Levoy et al.

# Obtained models

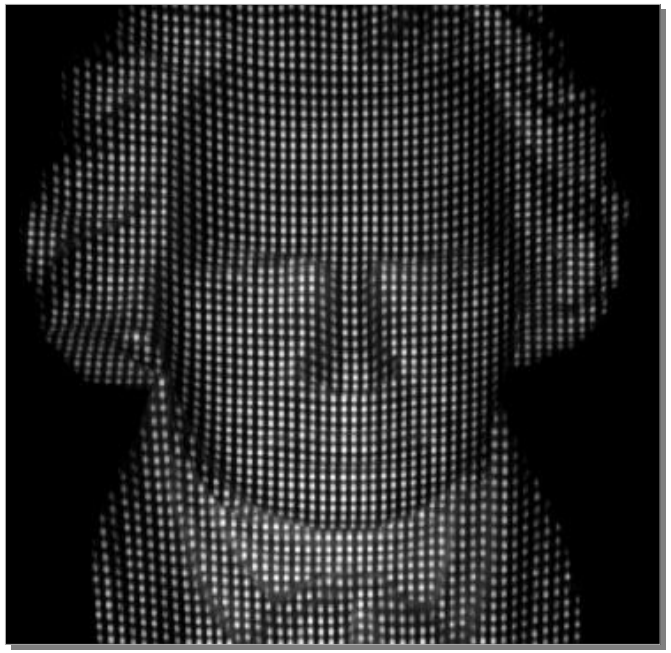
*The Digital Michelangelo Project*, Levoy et al.





# Multi-band triangulation

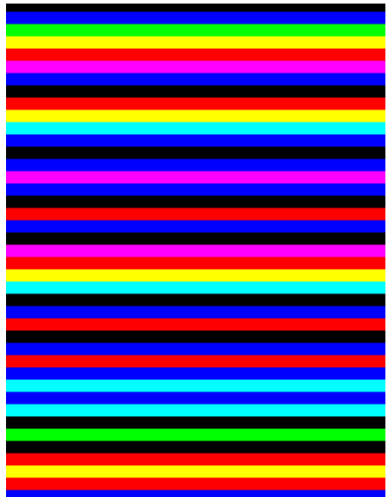
- Project multiple bands to speedup scanning
- But, which pixels belong to which band?
- Answer #1: Assume smooth surface

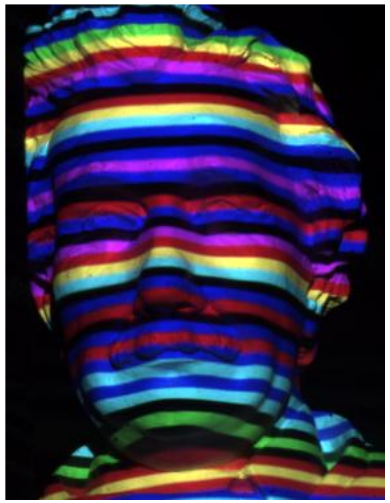


e.g. Eyetronics' ShapeCam

# Multi-band triangulation

- Project multiple bands to speedup scanning
- But, which pixels belong to which band?
- Answer #2: Project color bands (or points)

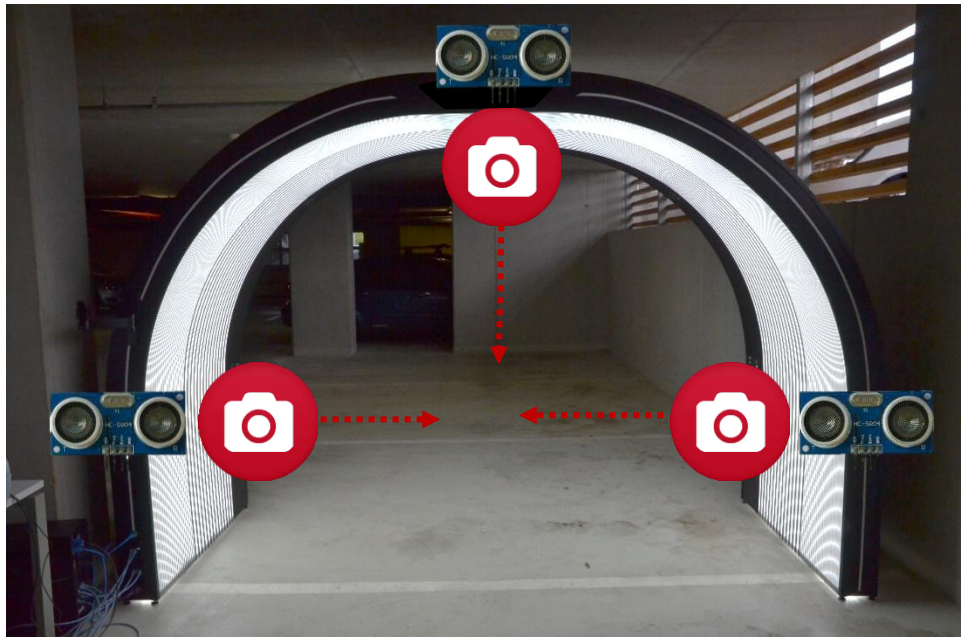


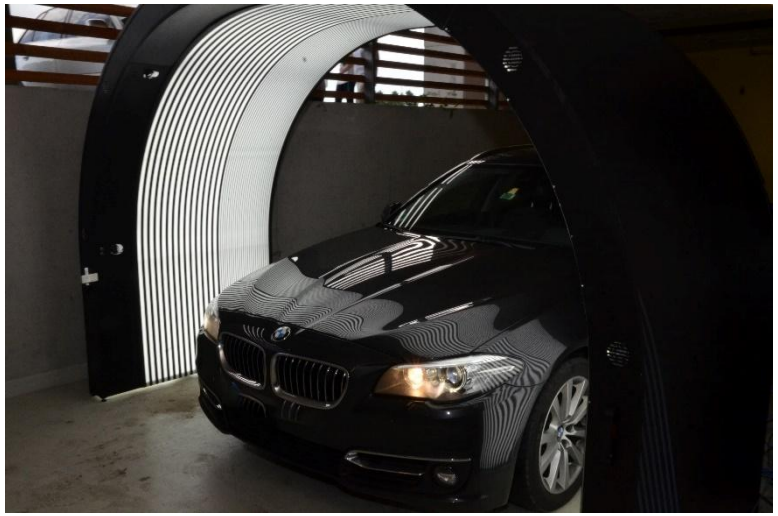


L. Zhang, B. Curless, and S. M. Seitz. Rapid Shape Acquisition Using Color Structured Light and Multi-pass Dynamic Programming. *3DPVT* 2002

# Quality control

- Automatic car inspection
- Industrial project (2016)
- Designed sensory system and software

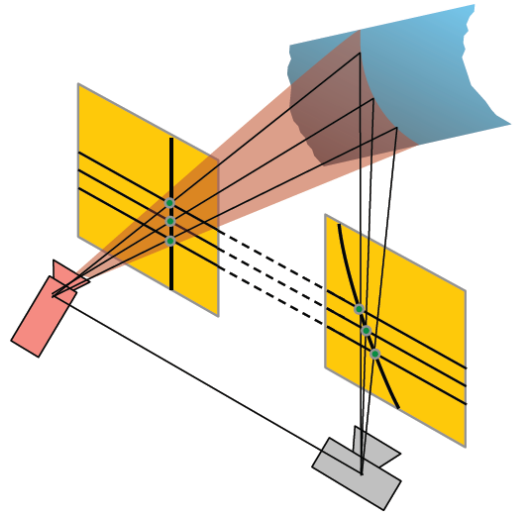


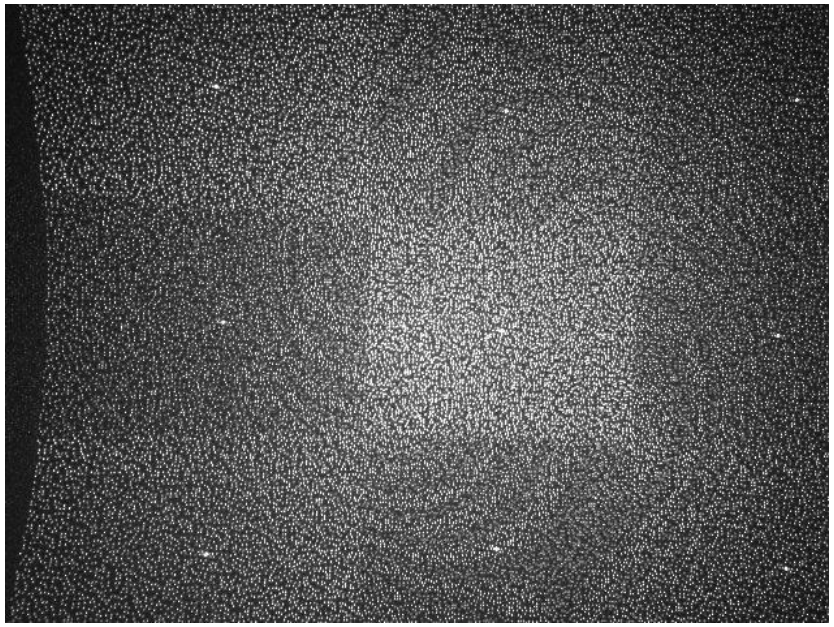


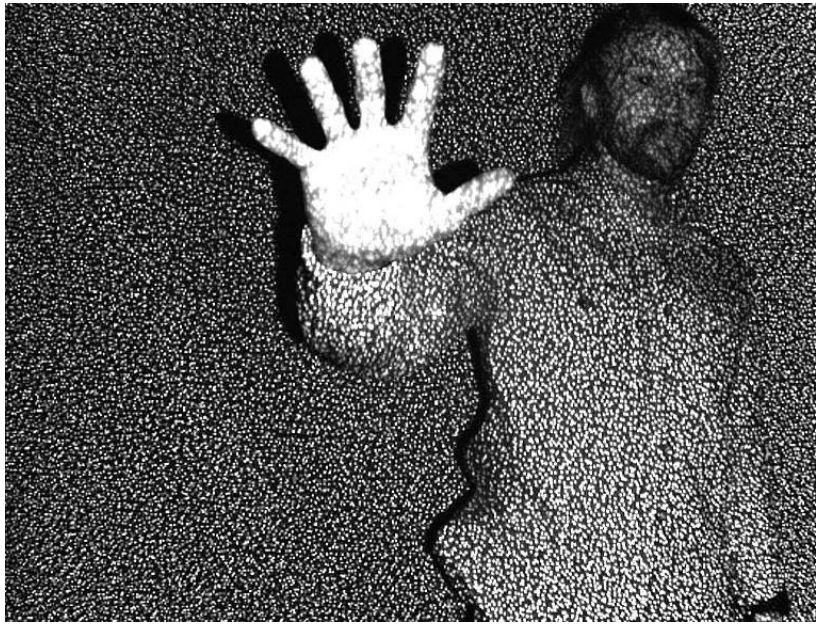
# In gaming industry (2010)

Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications
Kourosh Khoshelham and Sander Oude Elberink

- Project a point pattern for ultra fast triangulation!





# In phones (2017)

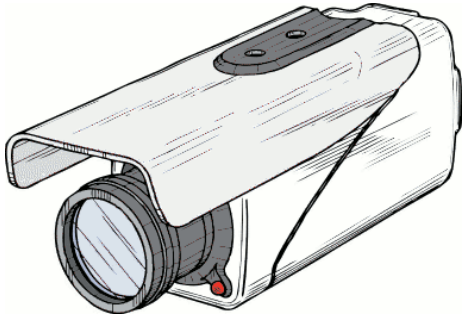
- Project a point pattern for ultra fast triangulation!


https://www.youtube.com/watch?v=OvVKnC6gGtg

# References


- David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition),
  - Stereo: Chapter 7
  - Structure from motion: Section 8.1
- R. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, 2nd Edition, Cambridge University Press, 2004
  - Camera model and calibration (Chapters 6 in 7)
  - Epipolar geometry (Chapter 9)
  - Calculating F (Chapters 11.1-11.6)
- Trym Vegard Haavardsholm: Stereo processing, Lecture 6
- Patent Primesense (Kinect): http://patentscope.wipo.int/search/en/WO2007043036

Univerza v Ljubljani



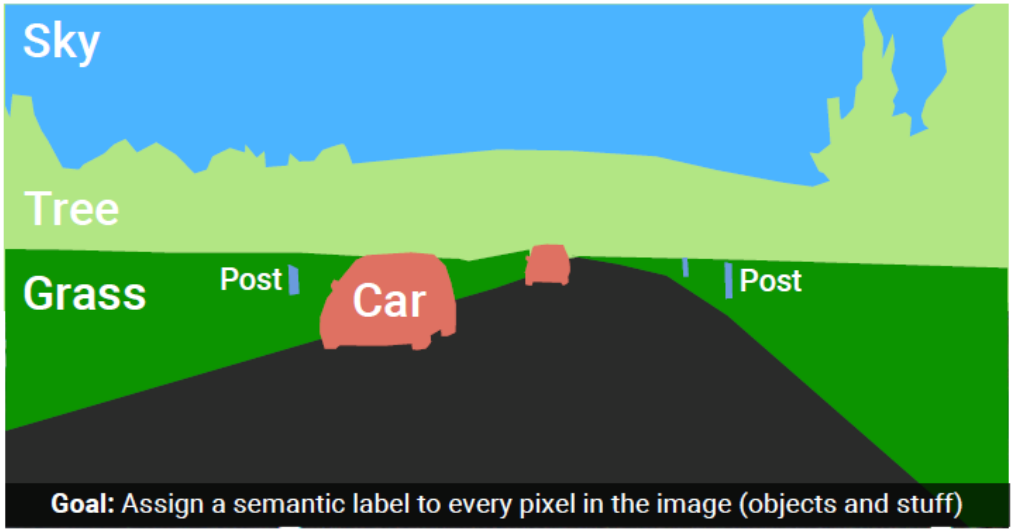
# Machine perception
# Recognition & Detection 1: Classical approaches

Matej Kristan

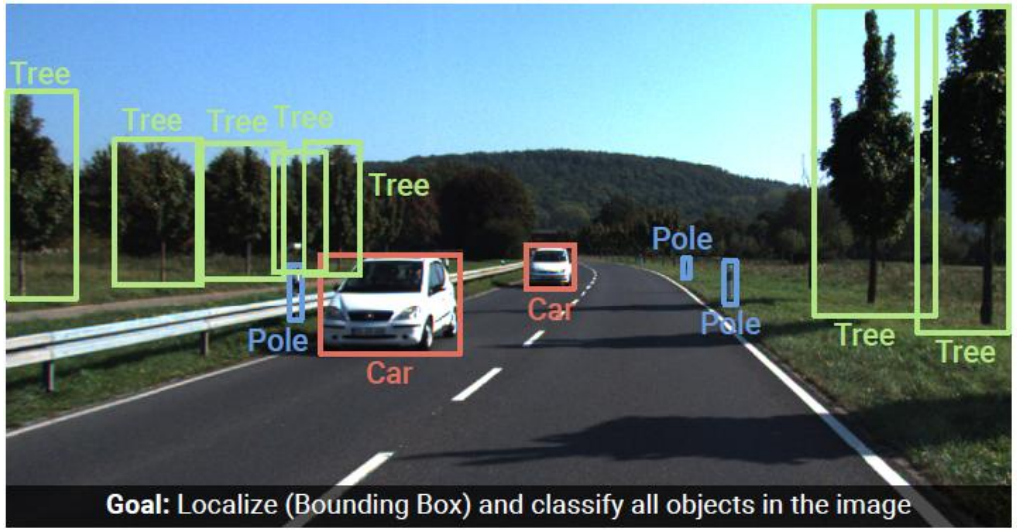

Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

# Recognition, Detection, Segmentation and beyond

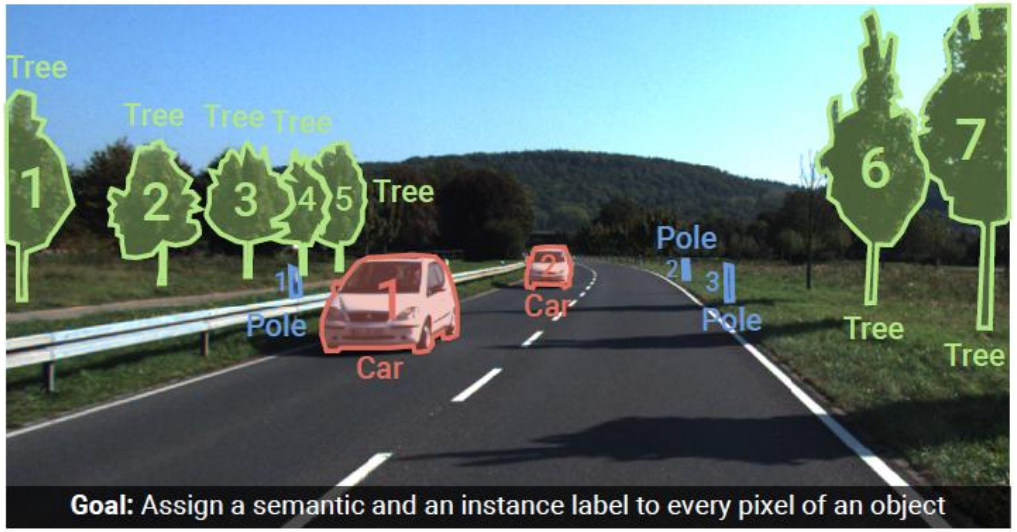

Image Classification




Semantic Segmentation




Object Detection




Instance Segmentation

# Recognition / Image classification

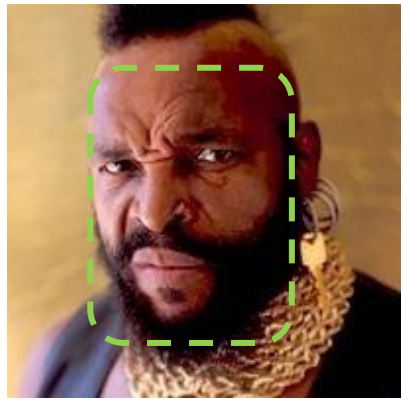
- Assume we have tagged an object with a bounding box

Whose face is it?

Assume it's one of your Instagram friends….

Marilyn Manson?

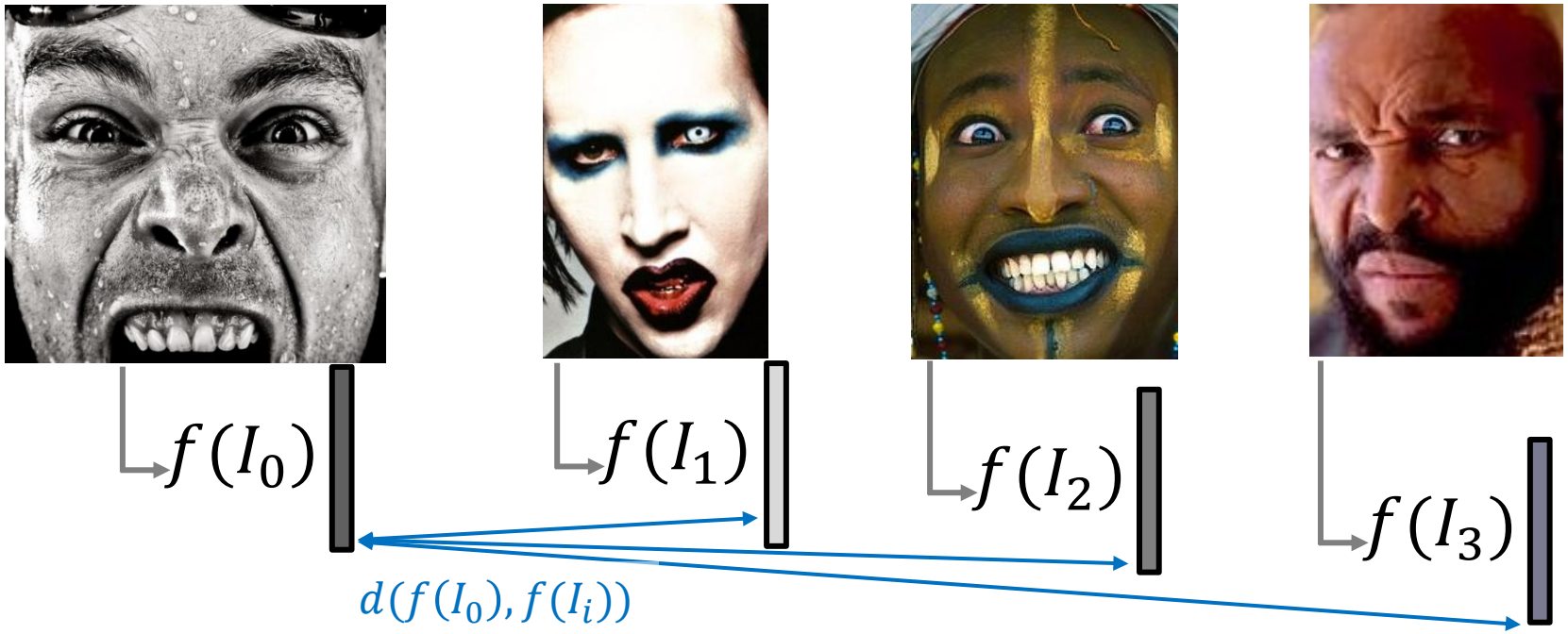
The Woodabe tribe guy?



Mr T?

…

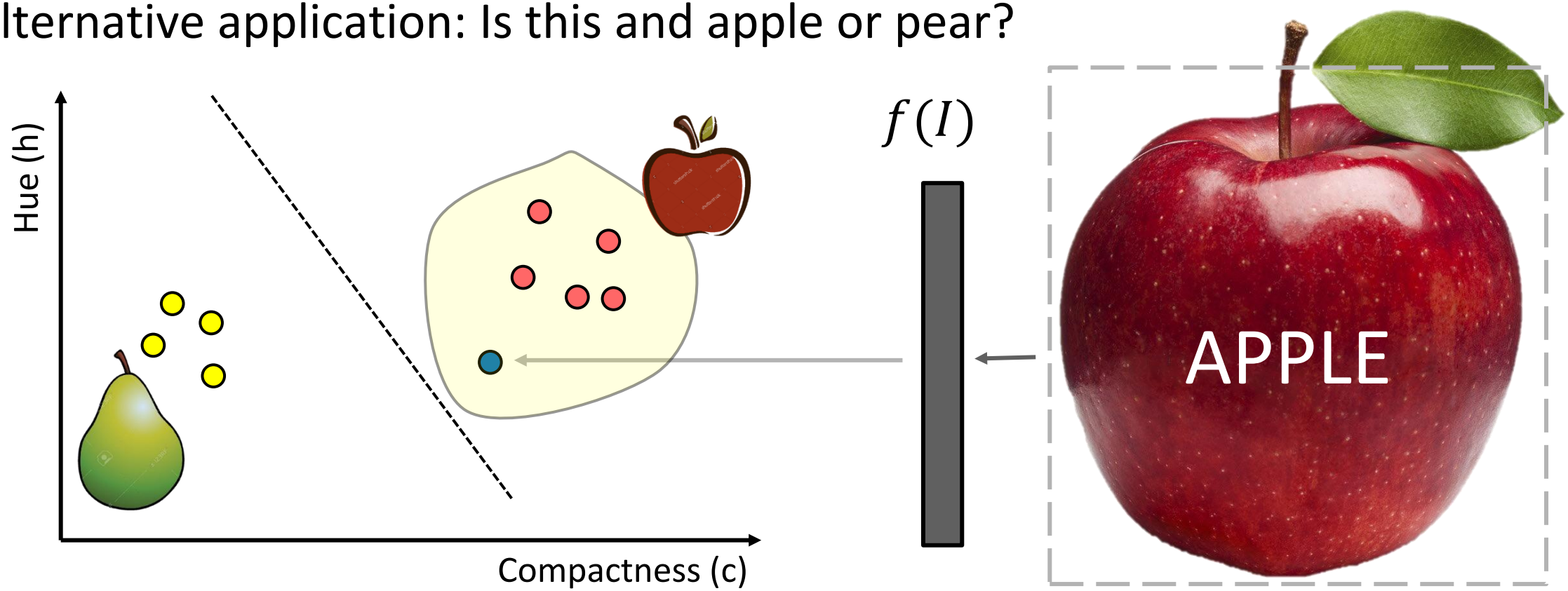
# Recognition by classification




- Select the one with minimal distance?
- Select the one(s) whose distance is sufficiently small?

# Recognition by classification

- Alternative application: Is this and apple or pear?




- Quality of recognition profoundly depends on the quality of image representation – features.

# How to determine the features?

1. Natural (linear) coordinate systems:

   *For some applications, it is enough just to linearly transform the input data.*

2. Feature selection:

   *Machine learning to select optimal features from a pool of several handcrafted transforms.*

3. Handcrafted nonlinear transforms:

   *Nonlinear transforms improve feature robustness.*

4. End-to-end learning of feature transform:

   *Have machine learn entire feature extraction and selection pipeline.*

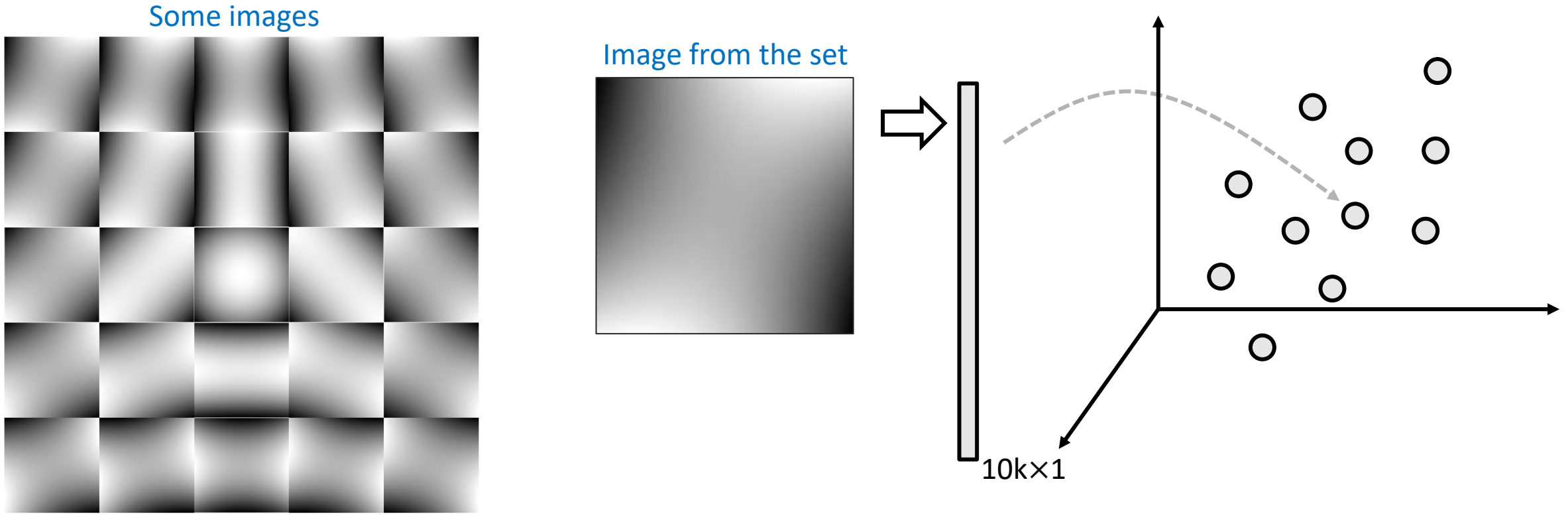
Machine Perception

# LEARNING LINEAR COORDINATE SYSTEMS BY SUBSPACE METHODS

# Motivation

- Imagine image as a high-dimensional gray-level vector

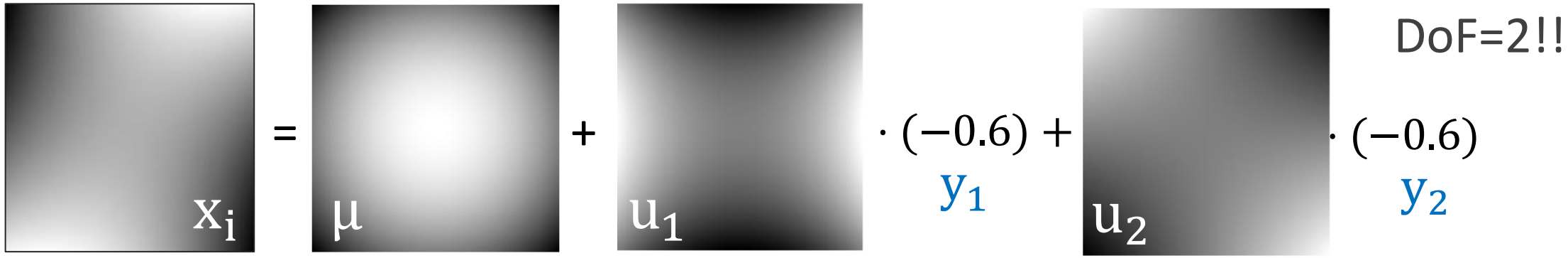
  (e.g., stack columns one on top of the other) → 100x100 image = 10k×1 dim vector



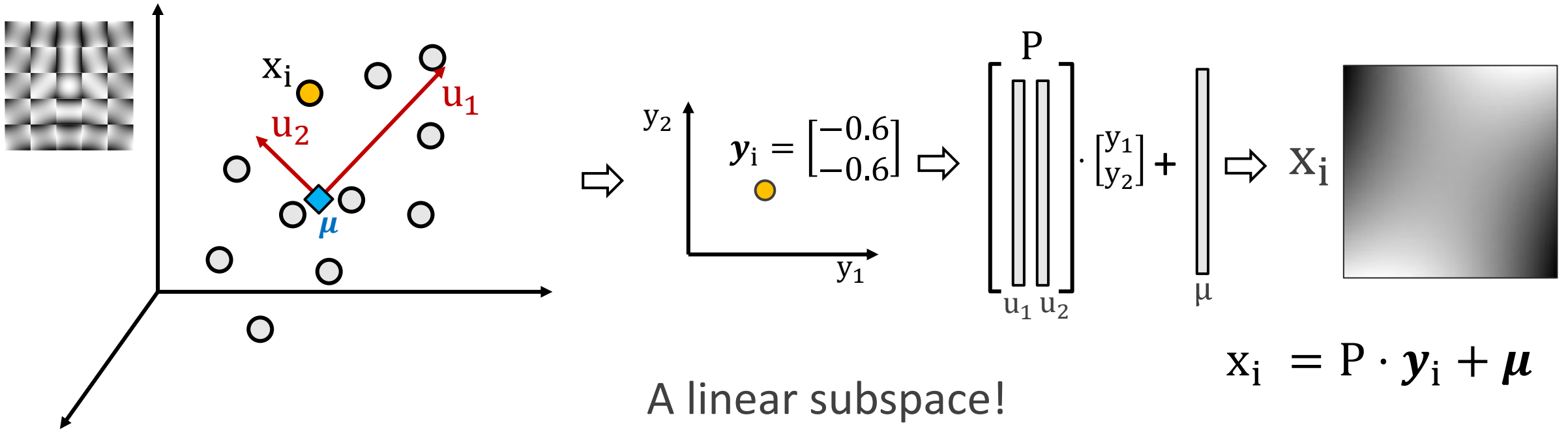

- What is the number of values required to encode the image? … 10k?

# Motivation

- Each image in the dataset has been generated by this equation:




DoF=2!!

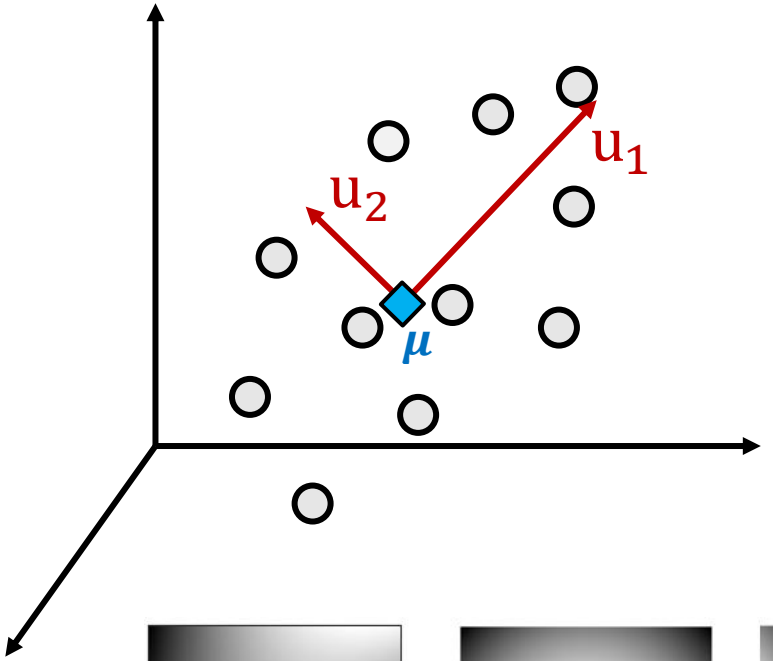



$$x_i = P \cdot y_i + \mu$$

A linear subspace!

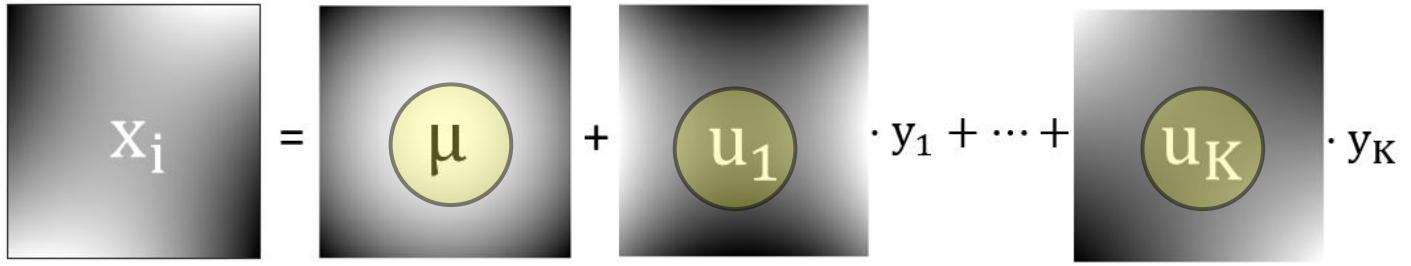
# Principal component analysis PCA (linear subspace)

- Find a low-dimensional linear subspace with orthonormal basis, such that projection of the data onto the subspace minimally distorts the data.




$$\mu = \begin{bmatrix} | \\ | \end{bmatrix} \qquad U = \begin{bmatrix} | & & | \\ u_1 & \cdots & u_K \\ | & & | \end{bmatrix}$$

$$U^{\mathrm{T}}U = \mathrm{I}$$



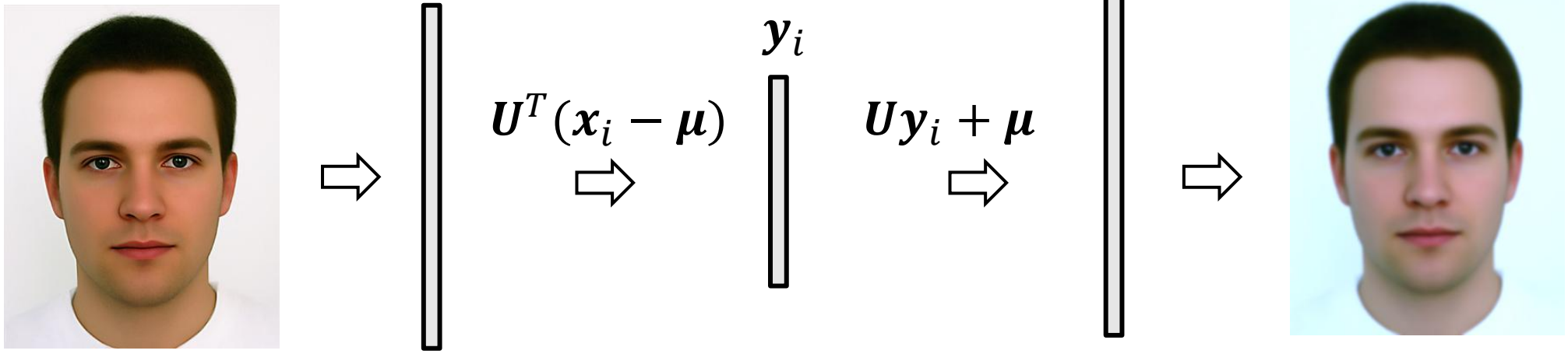
# Principal component analysis PCA (linear subspace)

- Find a low-dimensional linear subspace with orthonormal basis, such that projection of the data onto the subspace minimally distorts the data.

## Reconstruction error minimization

(over training set)!

$$\epsilon_i(\boldsymbol{\mu}, \boldsymbol{U}) = \|\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_i\|^2$$

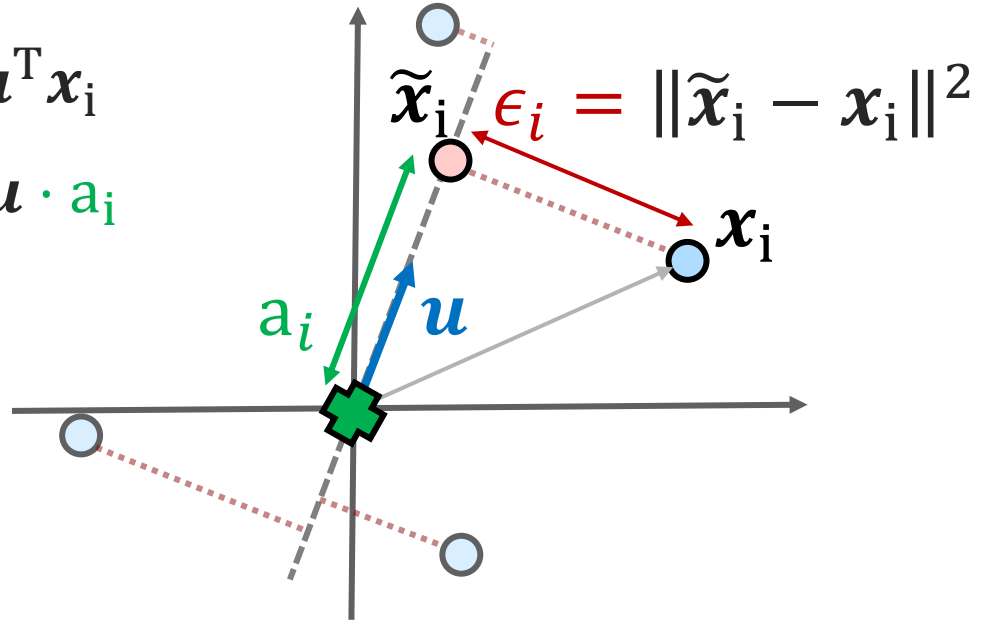


# A 1D subspace derivation

- Assume *N* centered data points: $x_1, \ldots, x_N$ ; $x_i \in R^M$ ,i.e., $\mu = \frac{1}{N}\sum_{i=1}^{N} x_i = 0$
- *Find a unit vector* $\boldsymbol{u} \in R^M$ *such that projection to this vector minimizes the average reconstruction error*

$$a_i = \boldsymbol{u}^T \boldsymbol{x}_i$$

$$\tilde{\boldsymbol{x}}_i = \boldsymbol{u} \cdot a_i$$




$$u_{opt} = \underset{u}{\operatorname{argmin}}\, \mathcal{E}(\boldsymbol{u})$$

$$\mathcal{E}(\boldsymbol{u}) = \frac{1}{N}\sum_{i=1:N} \|\tilde{\boldsymbol{x}}_i - \boldsymbol{x}_i\|^2$$

$$= \frac{1}{N}\sum_{i=1:N} \left\|\boldsymbol{u}\boldsymbol{u}^T\boldsymbol{x}_i - \boldsymbol{x}_i\right\|^2$$

$$= \frac{1}{N}\sum_{i=1:N} \left[-\boldsymbol{x}_i^T\boldsymbol{u}\boldsymbol{u}^T\boldsymbol{x}_i + \boldsymbol{x}_i^T\boldsymbol{x}_i\right]$$

$$u_{opt} = \underset{u}{\operatorname{argmax}} \frac{1}{N}\sum_{i=1:N} \boldsymbol{x}_i^T\boldsymbol{u}\boldsymbol{u}^T\boldsymbol{x}_i$$

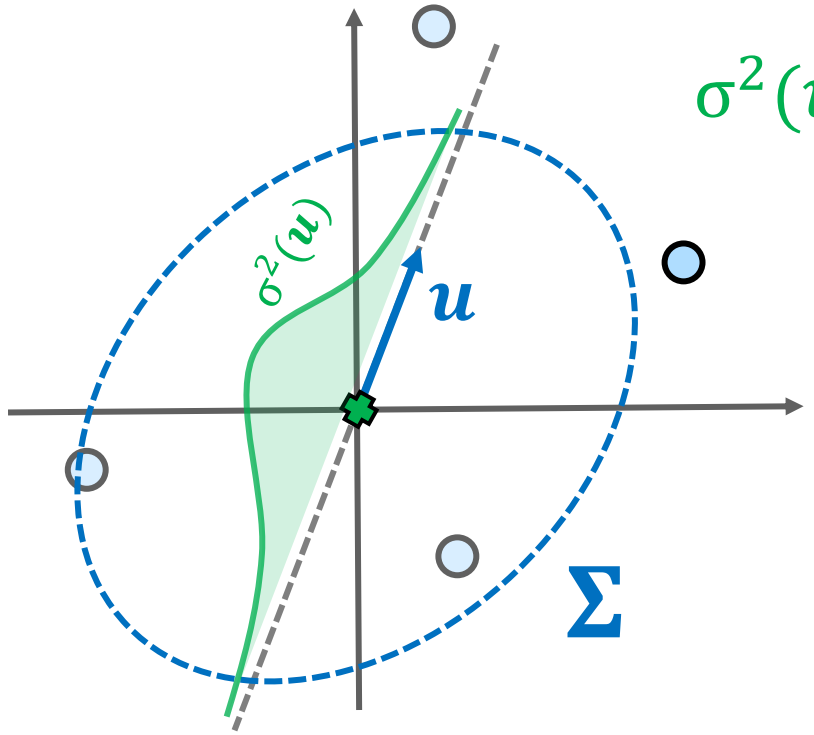
# A 1D subspace derivation

$$u_{opt} = \underset{u}{\operatorname{argmax}}\, \mathcal{E}'(\boldsymbol{u})$$

$$\mathcal{E}'(\boldsymbol{u}) = \frac{1}{N}\sum_{i=1:N} \boldsymbol{x}_i^T \boldsymbol{u}\boldsymbol{u}^T \boldsymbol{x}_i$$

$$= \frac{1}{N}\sum_{i=1:N} \boldsymbol{u}^T \boldsymbol{x}_i \boldsymbol{x}_i^T \boldsymbol{u}$$

$$= \boldsymbol{u}^T \underbrace{\left[\frac{1}{N}\sum_{i=1:N} \boldsymbol{x}_i \boldsymbol{x}_i^T\right]}_{\text{Data covariance matrix } \boldsymbol{\Sigma}} \boldsymbol{u}$$




$$\sigma^2(\boldsymbol{u}) = \boldsymbol{u}^T \boldsymbol{\Sigma} \boldsymbol{u}$$

$$u_{opt} = \underset{u}{\operatorname{argmax}}\, \boldsymbol{u}^T \boldsymbol{\Sigma} \boldsymbol{u}$$

- Find the unit vector that maximizes the projected data variance $\boldsymbol{u}^T \boldsymbol{\Sigma} \boldsymbol{u}$ !

# PCA – projected variance maximization

- Task: Find $\boldsymbol{u}$, that maximizes the following cost function (projected data variance)

$$\mathcal{E}'(\boldsymbol{u}) = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{u}$$

- Under constraint: $\|\boldsymbol{u}\|^2 = 1$
- Write a Lagrangian for constrained optimization:

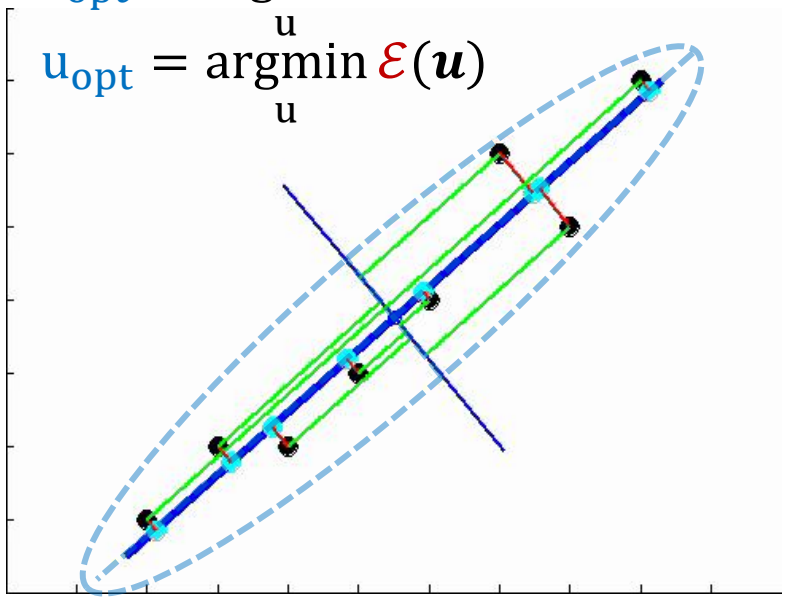
$$\mathcal{L}(\boldsymbol{u}) = \boldsymbol{u}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{u} - \lambda\left(\boldsymbol{u}^{\mathrm{T}}\boldsymbol{u} - \mathbf{1}\right)$$

$$\frac{\partial \mathcal{L}\,(\boldsymbol{u})}{\partial \boldsymbol{u}} = 0$$

$$\boldsymbol{\Sigma}\boldsymbol{u} = \lambda\boldsymbol{u}$$

$$\mathrm{u}_{\mathrm{opt}} = \underset{\mathrm{u}}{\operatorname{argmax}}\, \boldsymbol{u}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{u}$$

$$\mathrm{u}_{\mathrm{opt}} = \underset{\mathrm{u}}{\operatorname{argmin}}\, \mathcal{E}(\boldsymbol{u})$$



- We have obtained a standard equation whose solutions for $\boldsymbol{u}$ are the eigenvectors of $\Sigma$.

# PCA – projected variance maximization

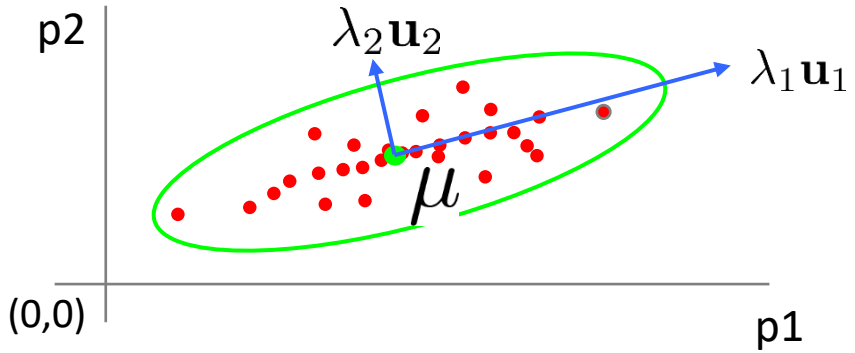
- Equation $\boldsymbol{\Sigma u} = \lambda \boldsymbol{u}$ is solved by eigenvectors & eigenvalues $(\boldsymbol{u}_j, \lambda_j)_{j=1:K}$ of $\boldsymbol{\Sigma}$.
- But which one maximizes the cost function $\mathcal{E}'(\boldsymbol{u}) = \boldsymbol{u}^T \boldsymbol{\Sigma u}$ ?
- Plugging $\boldsymbol{\Sigma u}_j = \lambda_j \boldsymbol{u}_j$ into the cost gives: $\mathcal{E}'(\boldsymbol{u}_j) = \lambda_j \boldsymbol{u}_j{}^T \boldsymbol{u}_j = \lambda_j$
- Since there are finite number of eigenvalues, the largest eigenvalue maximizes the cost! → $(\boldsymbol{u}_{max}, \lambda_{max})$
- *Since eigenvectors are orthogonal, following similar arguments, we can show that the subspace is spanned by the eigenvectors, ordered by their eigenvalues (from largest to smallest)*

# PCA – geometric interpretation

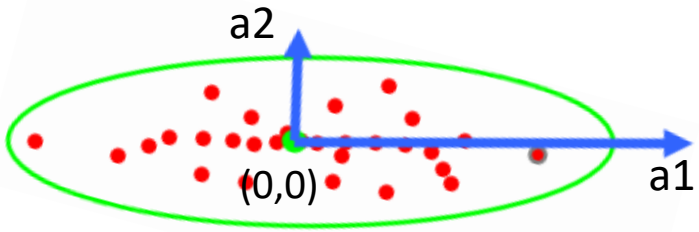
- Calculate eigenvectors and eigenvalues of covariance matrix $\boldsymbol{\Sigma}$
- Eigenvectors: main directions of variance, perpendicular to each other.

$$\boldsymbol{U} = [\boldsymbol{u}_1, \boldsymbol{u}_2]$$

- Eigenvalues: variance of data in direction of eigenvectors




1. Translate to origin by $\mathbf{t} = \boldsymbol{\mu}$
2. Rotate by $\boldsymbol{R} = \boldsymbol{U}$

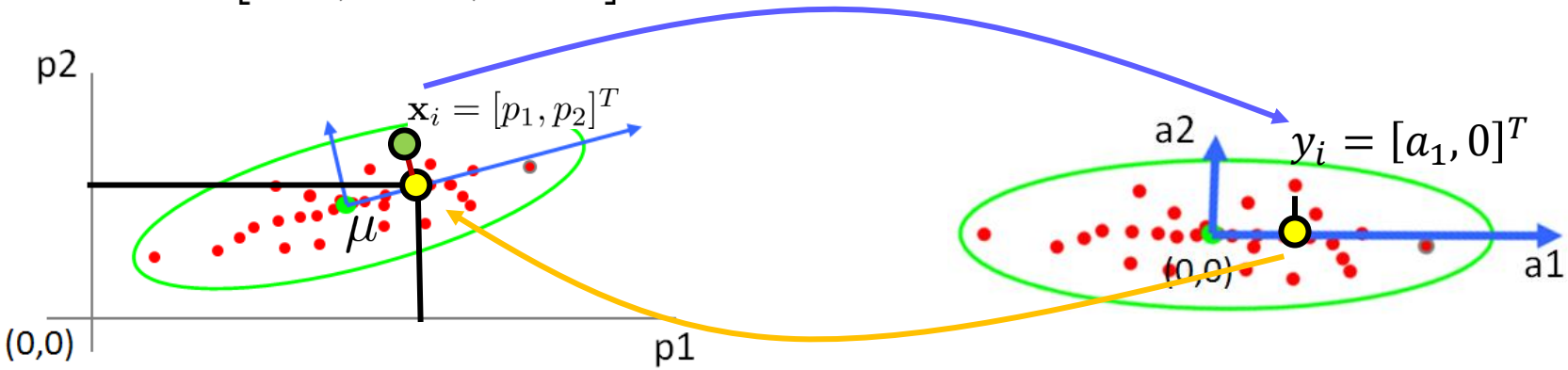



- PCA is actually: change of coordinate system that captures major directions of variance in the data.

# Projection and reconstruction

- We know the covariance matrix $\boldsymbol{\Sigma}$ and the mean value $\boldsymbol{\mu}$
- Concatenate first K eigen vectors into a rotation matrix $\boldsymbol{U}$:

$$\tilde{\mathbf{U}} = [\mathbf{u}_1, \ldots, \mathbf{u}_K]$$




- Projection to subspace:

$$\mathbf{y}_i = \tilde{\mathbf{U}}^T(\mathbf{x}_i - \mu)$$

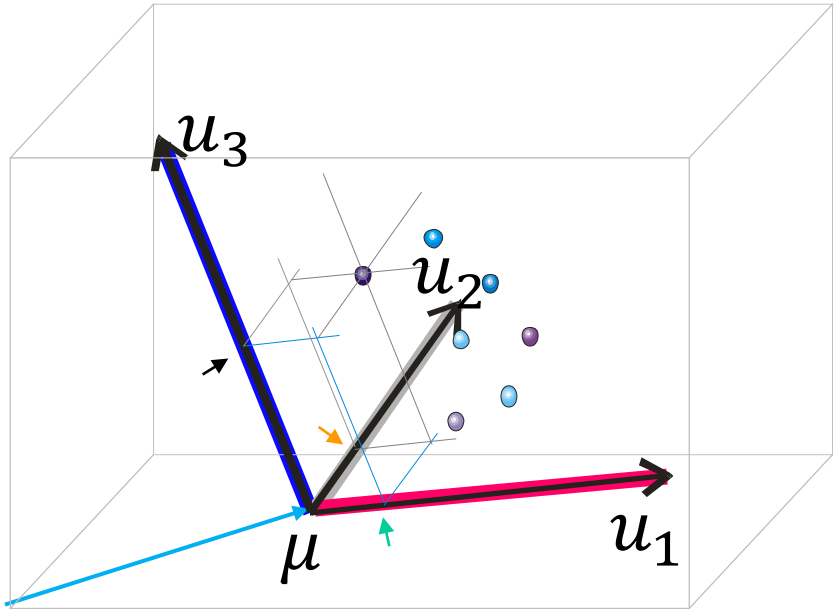
Reconstruction error:

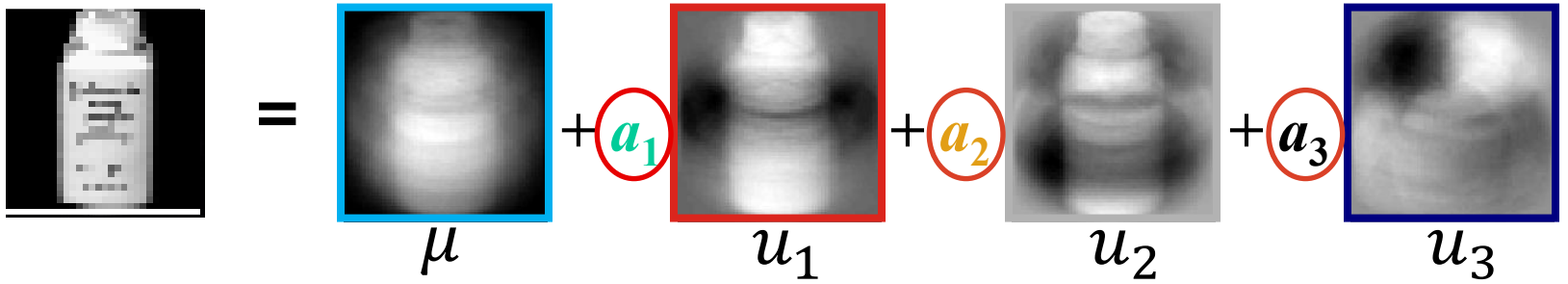
$$\epsilon_i = ||\mathbf{x}_i - \tilde{\mathbf{x}}_i||^2$$

- Reconstruction:

$$\tilde{\mathbf{x}}_i = \tilde{\mathbf{U}}\mathbf{y}_i + \mu$$

# Example: Object representation




Q: How many of $a_i$ should you retain?

# How many eigenvectors for reconstruction?

- Can show that the sum of squared differences $\epsilon(m)$ between training images $\{\boldsymbol{x}_i\}_{i=1:N}$ and their reconstructions using only first $m$ eigen vectors is given by:

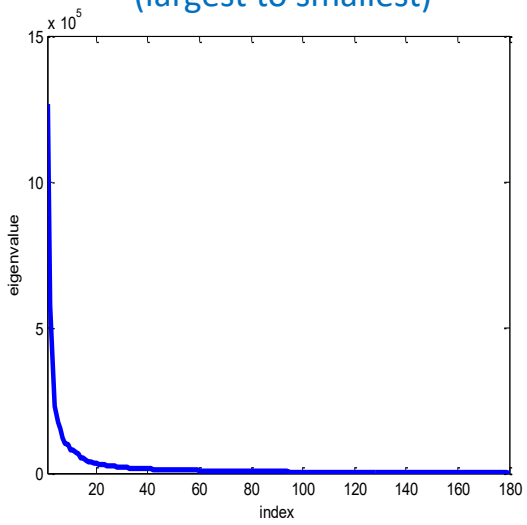
Original

Reconstruction with $m$ eigenvectors

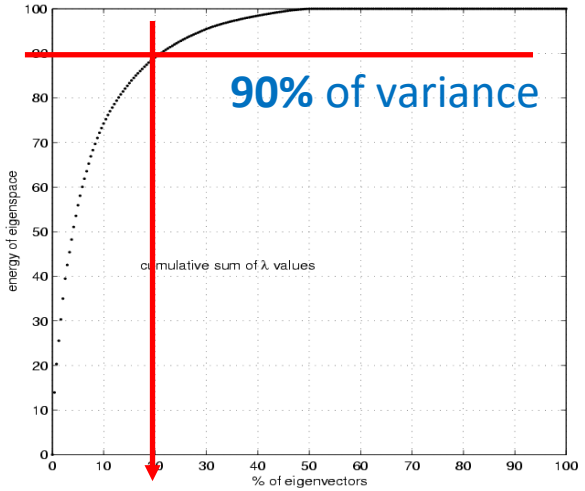
$$\epsilon(m) = \| \quad - \quad \|^2$$

$$\epsilon(m) = \sum_{j=1}^{N} \lambda_j - \sum_{j=1}^{m} \lambda_j = \sum_{j=m+1}^{N} \lambda_j$$

Eigenvalues
(largest to smallest)




Cumulative sum of eigenvalues
(explained variance)




$m$ eigen vectors

# Build you own subspace!

- Reshape all training images into column vectors: $[\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N]$
- Calculate the average image: $\mu = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$
- Center data: $\mathbf{X}_d = [\mathbf{x}_1 - \mu, \mathbf{x}_2 - \mu, \ldots, \mathbf{x}_N - \mu]$
- Calculate the covariance matrix: $\mathbf{C} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x_i} - \mu)(\mathbf{x_i} - \mu)^T = \frac{1}{N}\mathbf{X}_d\mathbf{X}_d^T$
- Calculate eigenvector matrix ***U*** and eigenvalue matrix ***S*** (using, e.g., svd): $\mathbf{C} = \mathbf{U}\mathbf{S}\mathbf{V}^T$
- Construct a matrix using only first K eigen vectors: $\tilde{\mathbf{U}} = [\mathbf{u}_1, \ldots, \mathbf{u}_K]$
- For each test image ***x***:
  - Project to subspace: $\mathbf{y} = \tilde{\mathbf{U}}^T(\mathbf{x} - \mu)$
  - Reconstruct: $\tilde{\mathbf{x}} = \tilde{\mathbf{U}}\mathbf{y} + \mu$

Note: for positive semidefinite matrix C, eigenvalues == singular values

# Important!

- Do not implement PCA as shown in the previous slide!

$$\mathbf{C} = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x_i} - \mu)(\mathbf{x_i} - \mu)^T = \frac{1}{N}\mathbf{X}_d\mathbf{X}_d^T$$

1. Consider the size of the covariance matrix $\mathrm{C}$
   - *The size is M×M*, where *M* is the number of pixels in the image!
   - But, we have only $N$ training examples, typically $N<<M$.

   $\Rightarrow$ So $\mathrm{C}$ will have at most rank $N$!

2. In any case, we need only first $k$ eigen vectors!

# The inner-product matrix

- For *a large* $M$, the SVD of $C$ becomes inefficient.

$$\mathbf{C} = \frac{1}{N}\mathbf{X}_d\mathbf{X}_d^T$$

- For *N* <<*M*, the $N \times N$ inner product matrix $\mathbf{C}'$ will be smaller:

$$\mathbf{C}' = \frac{1}{N}\mathbf{X}_d^T\mathbf{X}_d$$

- Eigenvectors and eigenvalues of matrix $\boldsymbol{C}$ are calculated from the eigenvectors $\boldsymbol{u}_i'$ and eigenvalues $\lambda_i'$ of $\boldsymbol{C}'$:

$$\lambda_i = \lambda_i'$$
$$\mathbf{u}_i = \frac{\mathbf{X}_d\mathbf{u}_i'}{\sqrt{N\lambda_i'}},\ i = 1 \ldots N$$

This is called "the dual PCA"

# A general PCA algorithm

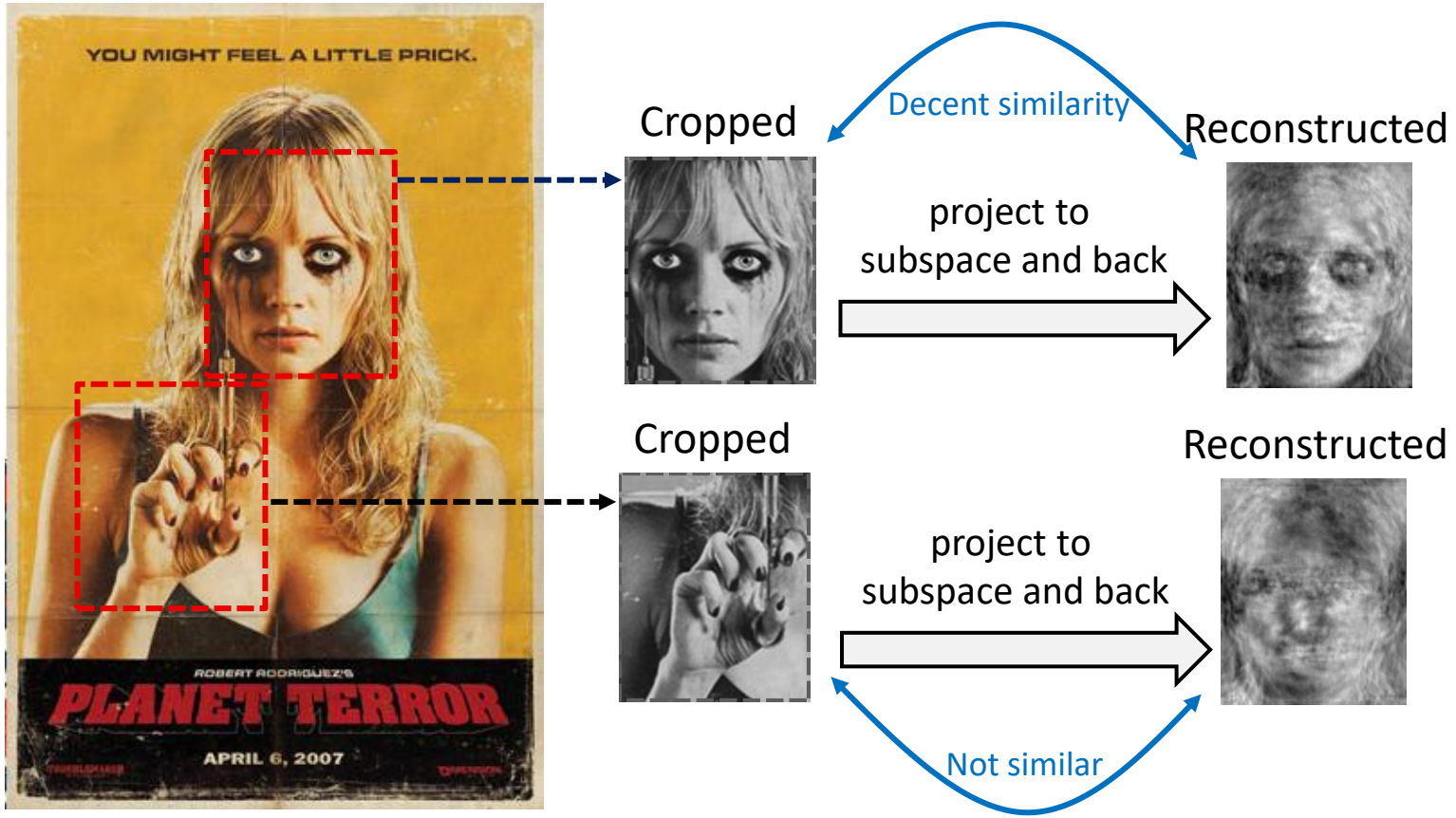
**Input**: data matrix $\mathbf{X}$

**Output**: mean value $\boldsymbol{\mu}$, eigenvectors $\mathbf{U}$, eigenvalues $\boldsymbol{\lambda}$.

1. Estimate the mean vector: $\boldsymbol{\mu} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$
2. Center the input data around the mean: $\hat{\mathbf{X}}_i = \mathbf{X}_i - \mu$
3. **if** $M \leq N$ **then**
4. Estimate the covariance matrix : $\mathbf{C} = \frac{1}{N}\hat{\mathbf{X}}\hat{\mathbf{X}}^\top$
5. Perform SVD on $\mathbf{C}$. Obtain eigenvectors $\mathbf{U}$ and eigenvalues $\boldsymbol{\lambda}$.
6. **else**
7. Estimate the inner product matrix: $\mathbf{C}' = \frac{1}{N}\hat{\mathbf{X}}^\top\hat{\mathbf{X}}$
8. Perform SVD on $\mathbf{C}'$. Obtain eigenvectors $\mathbf{U}'$ and eigenvalues $\boldsymbol{\lambda}'$.
9. Determine the eigenvectors $\mathbf{U}$ : $\mathbf{u}_i = \frac{\hat{\mathbf{X}}\mathbf{u}_i'}{\sqrt{N\lambda_i'}}$ , $i = 1 \ldots N$
10. Determine the eigenvalues $\boldsymbol{\lambda} = \boldsymbol{\lambda}'$
11. **end if**

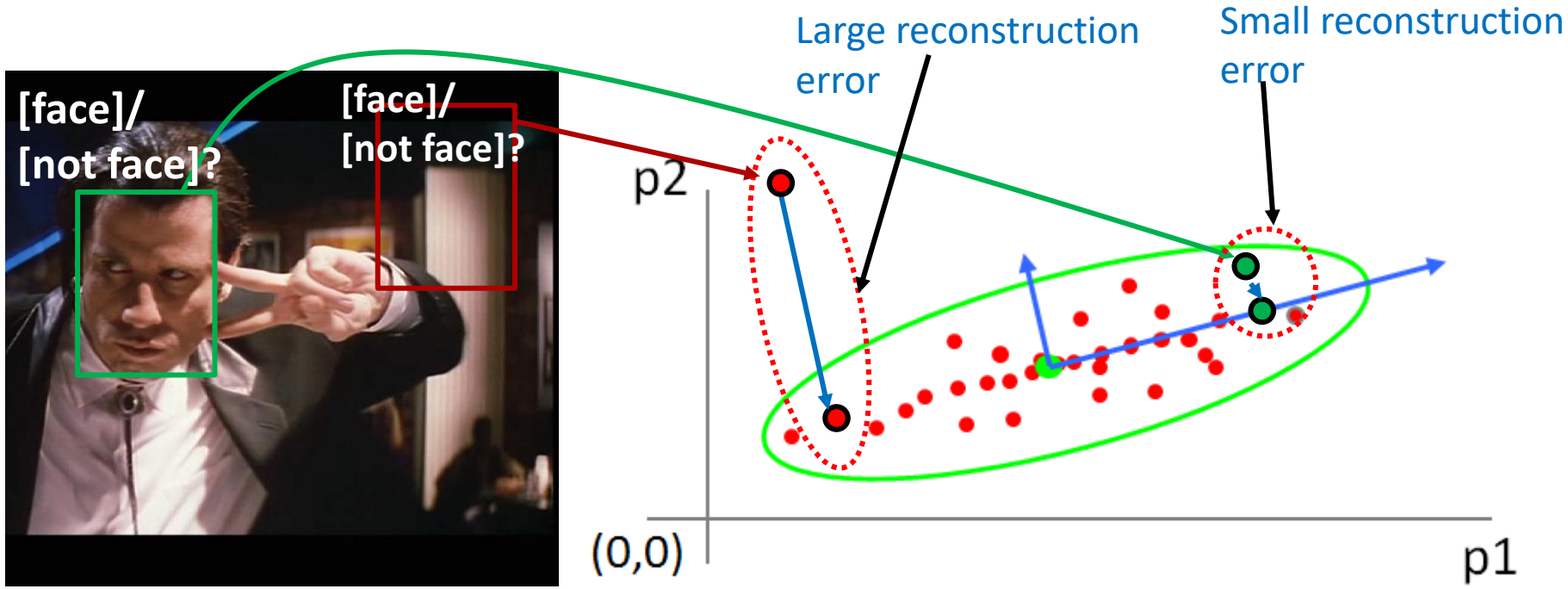
# Classification by subspace reconstruction

- If the window contains a category for which the subspace was constructed, the reconstruction will work well, otherwise not!
- A real-life example



# Classification by subspace reconstruction

- Can exploit this property for category detection/recognition
- Assume we have used a large collection of faces to construct the subspace.
- Fact: Only faces will be well reconstructed by the subspace!



# Detection by distance from subspace

- Use a subspace learned on faces to detect a face.
- Approach: slide a window over each image position and calculate the reconstruction error.
- Repeat for all scales. Makes sense to normalize the window intensity |W|=1.
- Low reconstruction error indicates a face.
  (i.e., apply a threshold)

$$\left\| \tilde{\mathbf{x}}_i - \mathbf{x}_i \right\|^2 < \theta$$

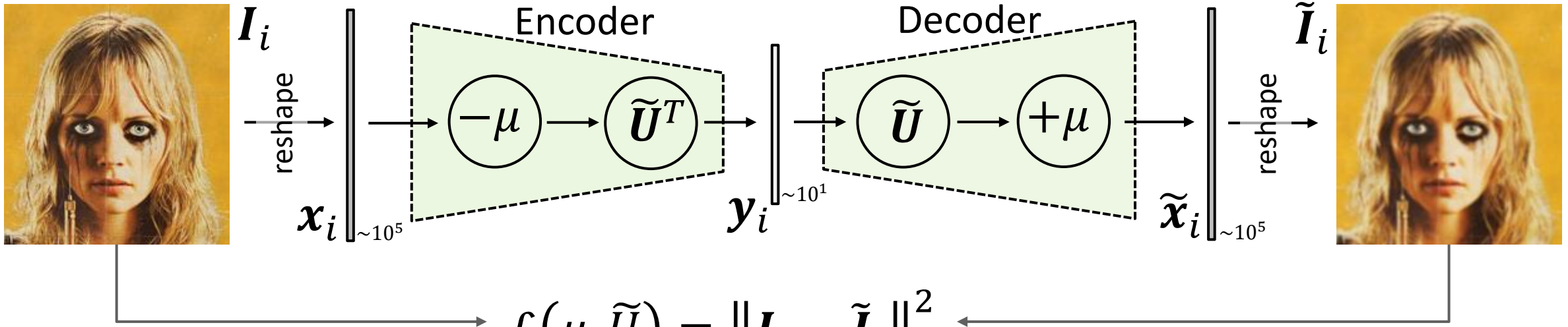
# Home study: Textbooks on PCA

- Szeliski, R., Computer vision – algorithms and applications, 2011, Section14.2.1 (available online)

- Forsyth, Ponce, Computer vision – a modern approach, second edition, 2012, Section 16.1.5

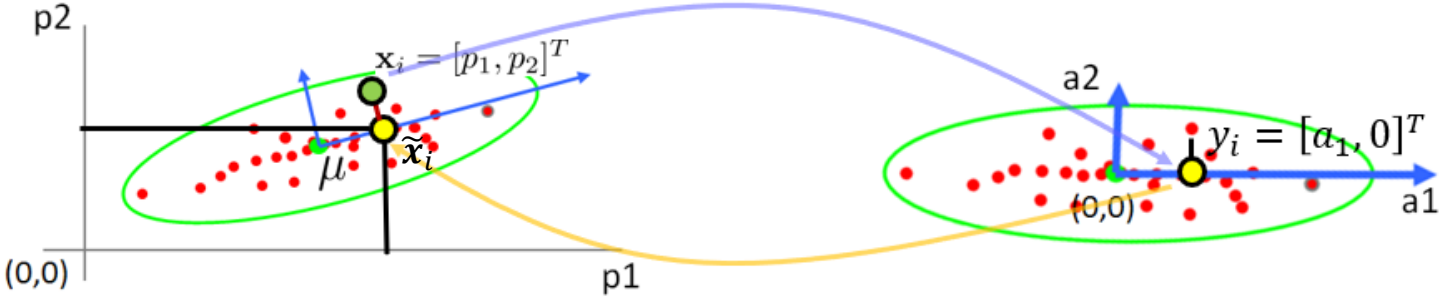
- Prince, S.J.D. Computer vision – modelling learning and inference, 2012, Section 13.4 (to 13.4.3) (available online)

# PCA is a *linear* autoencoder




$$\mathcal{L}(\mu, \tilde{U}) = \left\| I_i - \tilde{I}_i \right\|^2$$

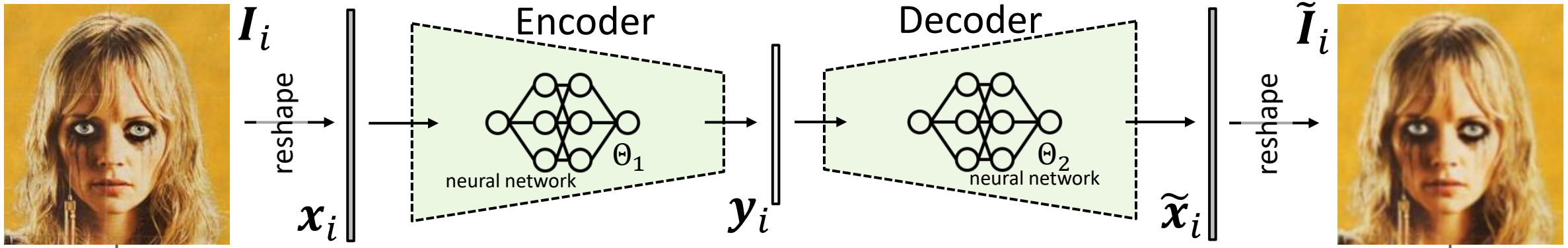
$$\mu, \tilde{U} = \underset{\mu^*, \tilde{U}^*}{\operatorname{argmin}} \mathcal{L}(\mu^*, \tilde{U}^*)$$




$$\mathbf{y}_i = \tilde{\mathbf{U}}^T(\mathbf{x}_i - \mu)$$

$$\tilde{\mathbf{x}}_i = \tilde{\mathbf{U}}\mathbf{y}_i + \mu$$

# Encoder-Decoder does not have to be linear

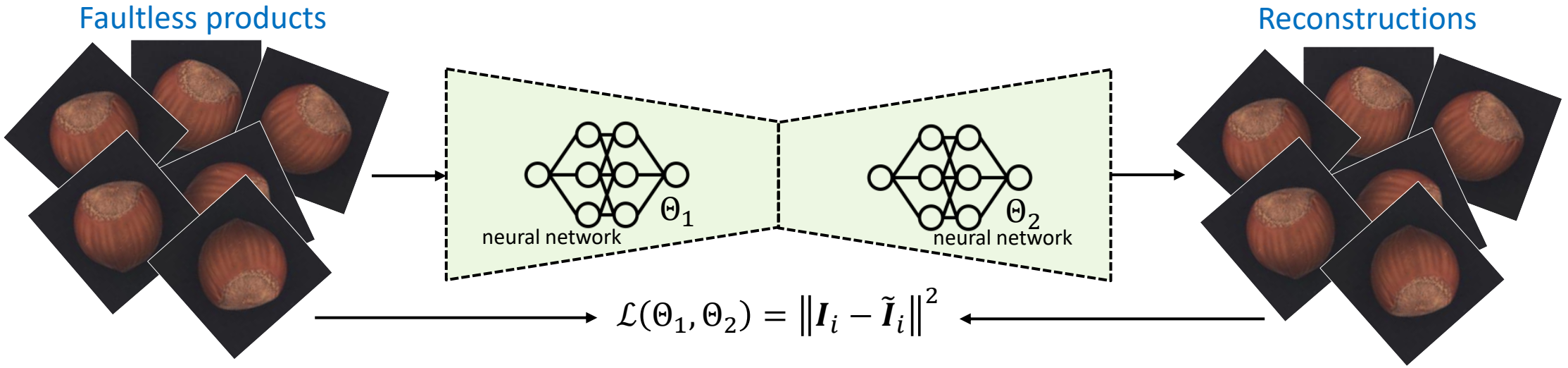



$$\mathcal{L}(\Theta_1, \Theta_2) = \left\| I_i - \tilde{I}_i \right\|^2$$

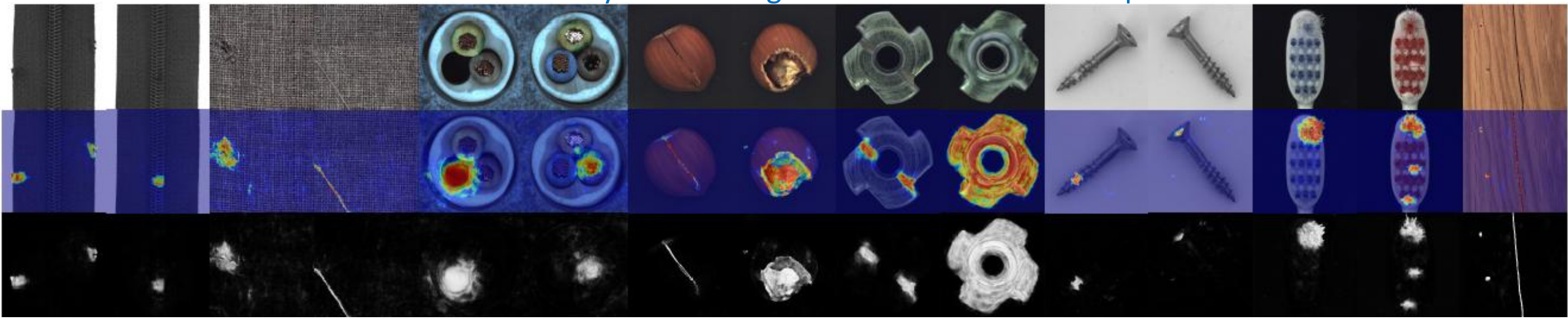
$$\Theta_1, \Theta_2 = \underset{\Theta_1^*, \Theta_2^*}{\operatorname{argmin}} \mathcal{L}(\Theta_1^*, \Theta_2^*)$$

- Modern Autoencoders apply (convolutional/transformer) neural networks to map into a nonlinear subspace (latent space)

# Autoencoders don’t have to just reconstruct

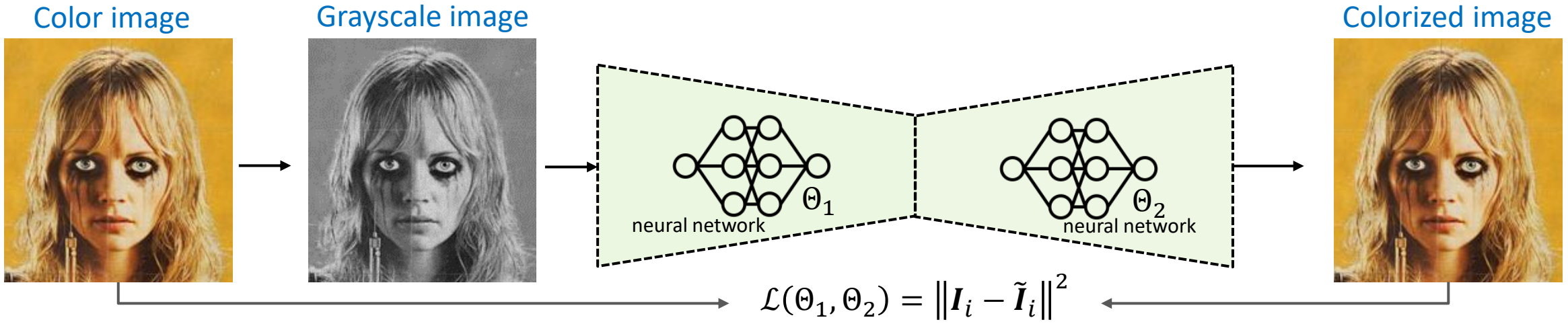


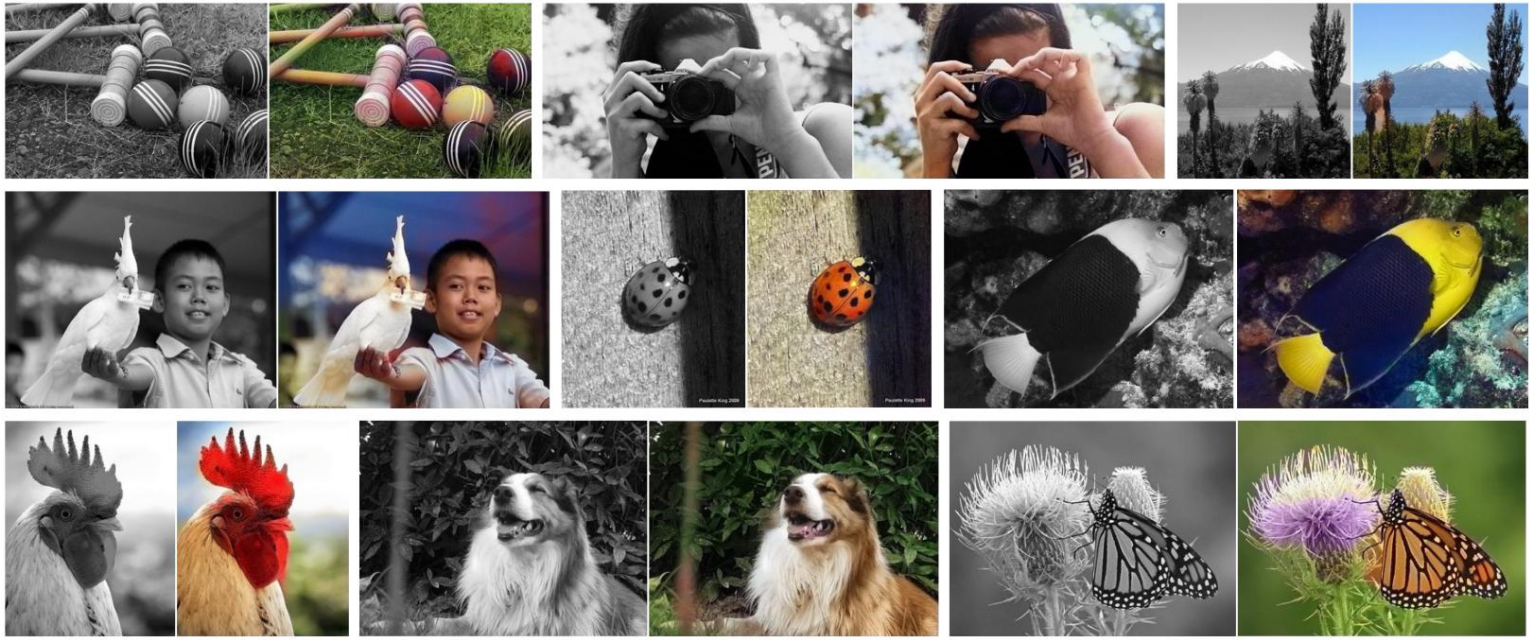

Anomalies detected by differencing the reconstruction and input



Zavrtanik, Kristan, Skočaj, DRÆM – A discriminatively trained reconstruction embedding for surface anomaly detection, ICCV 2021

# Autoencoders don’t have to just reconstruct

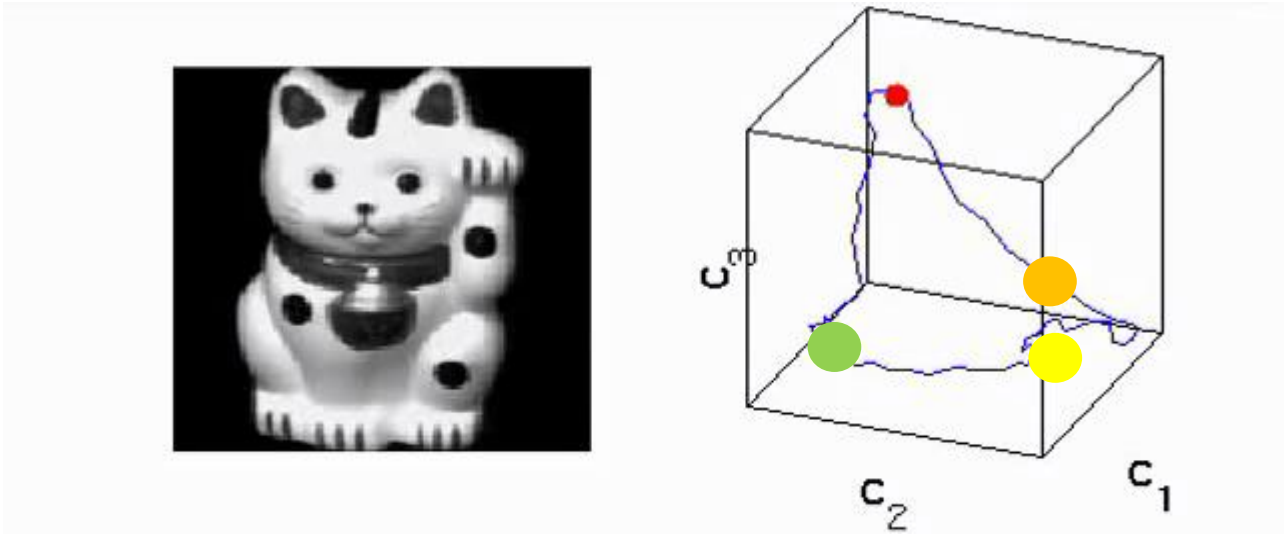


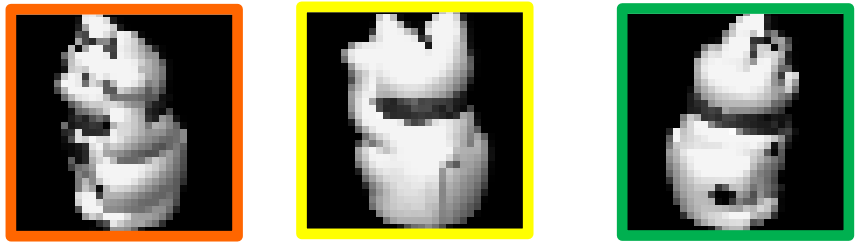



Zhang et al., Colorful Image Colorization, ECCV 2016 [GIT]

# Parametric subspace (PCA)

- Even in PCA, the same object viewed under different conditions (illumination, orientation) projects into different parts of the subspace:



- One could thus in principle generate different appearances by accessing specific part of the subspace

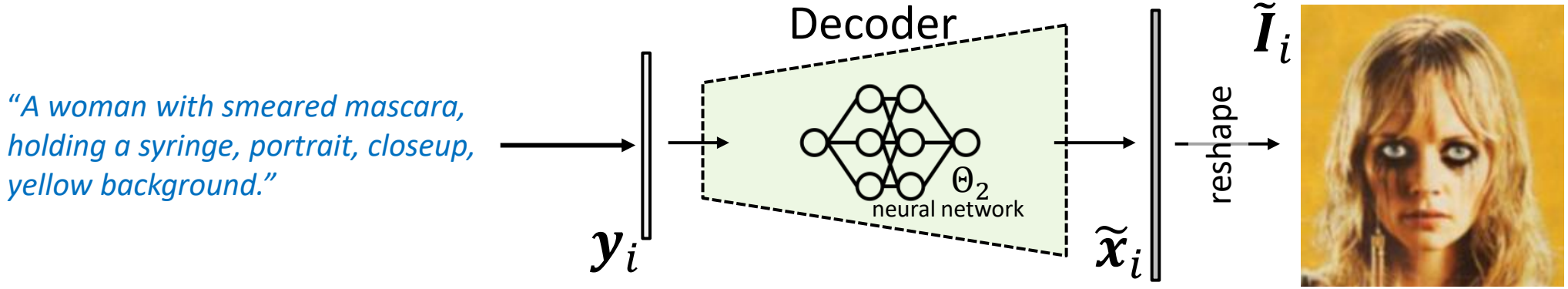
Subspace point (a vector)

Generated image (a vector)

$$\tilde{\mathbf{x}} = \tilde{\mathbf{U}}\mathbf{y} + \mu$$

# Generative promptable image models

- Modern generators like StableDiffusion[1] proceed by first generating an appropriate latent representation, then decode into image.

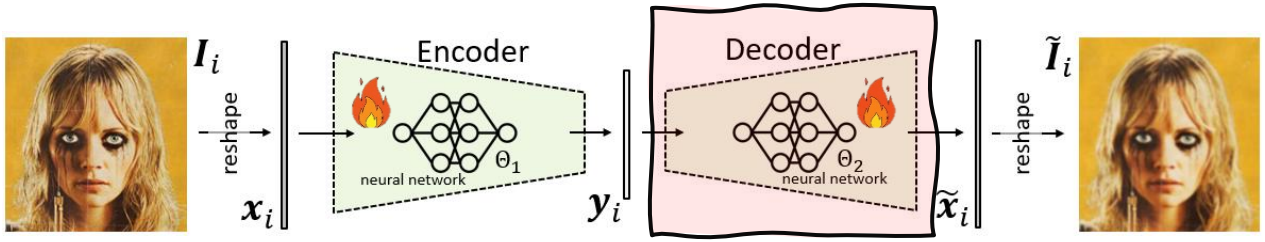



- But, how to come up with the appropriate decoder? And how to find the right latent representation?

[1]Rombach et al., High-Resolution Image Synthesis with Latent Diffusion Models, CVPR 2022

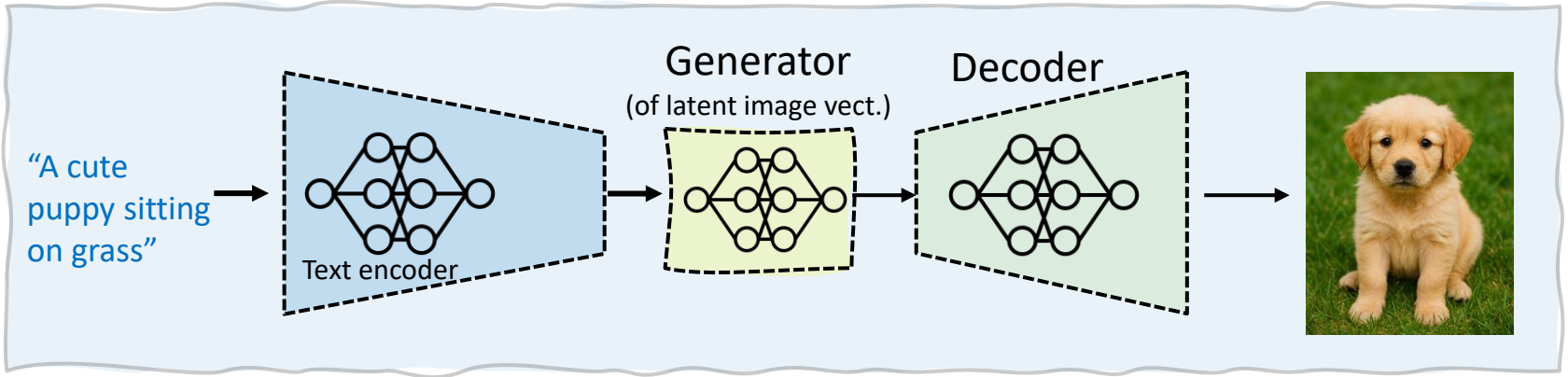
# Generative promptable image models

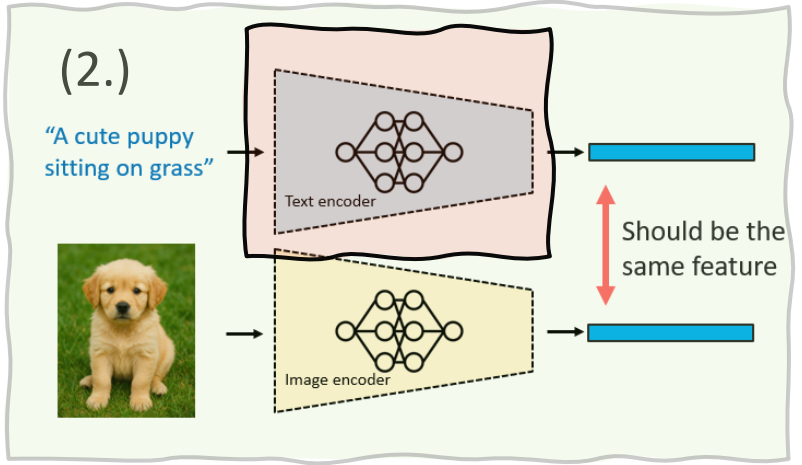
1. Train a (variational) autoencoder on a huge dataset

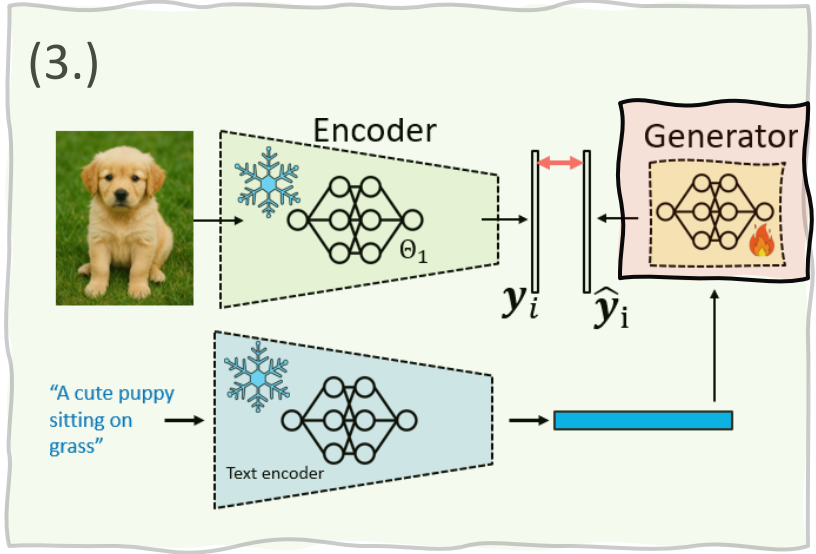



2. Train a text encoder, aligned with image features from large corpus of image/text pairs.

3. Train a diffusion model that uses text prompt to generate the latent representation of an input.



# Examples of image generation

https://flux1.ai/

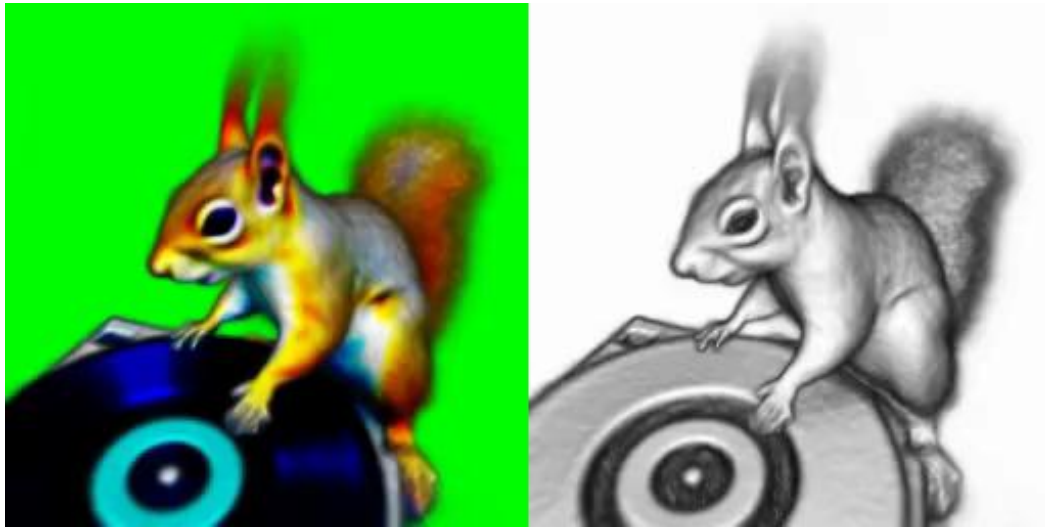
*Prompt: "A close-up shot of a passionate female chef in her 40s with curly black hair tucked under a white chef's hat, wearing a crisp white double-breasted jacket. She's intently focused on plating a colorful dish in a sleek, modern kitchen. Her skilled hands delicately position a vibrant red bell pepper slice on the plate. Stainless steel appliances and hanging copper pots gleam in the background. Steam rises from the freshly cooked food, creating a misty effect around her. Her expression shows a mix of concentration and pride. The scene captures the artistry and precision of haute cuisine, styled like a high-end culinary magazine spread with dramatic lighting emphasizing the textures and colors of the food."*

OpenAI (DALL·E) via ChatGPT

*Prompt: "a guy I met in front of Parlament pub last night, joyful, looked like Yann Le Cun, but younger and thinner and said he's from FDV"*

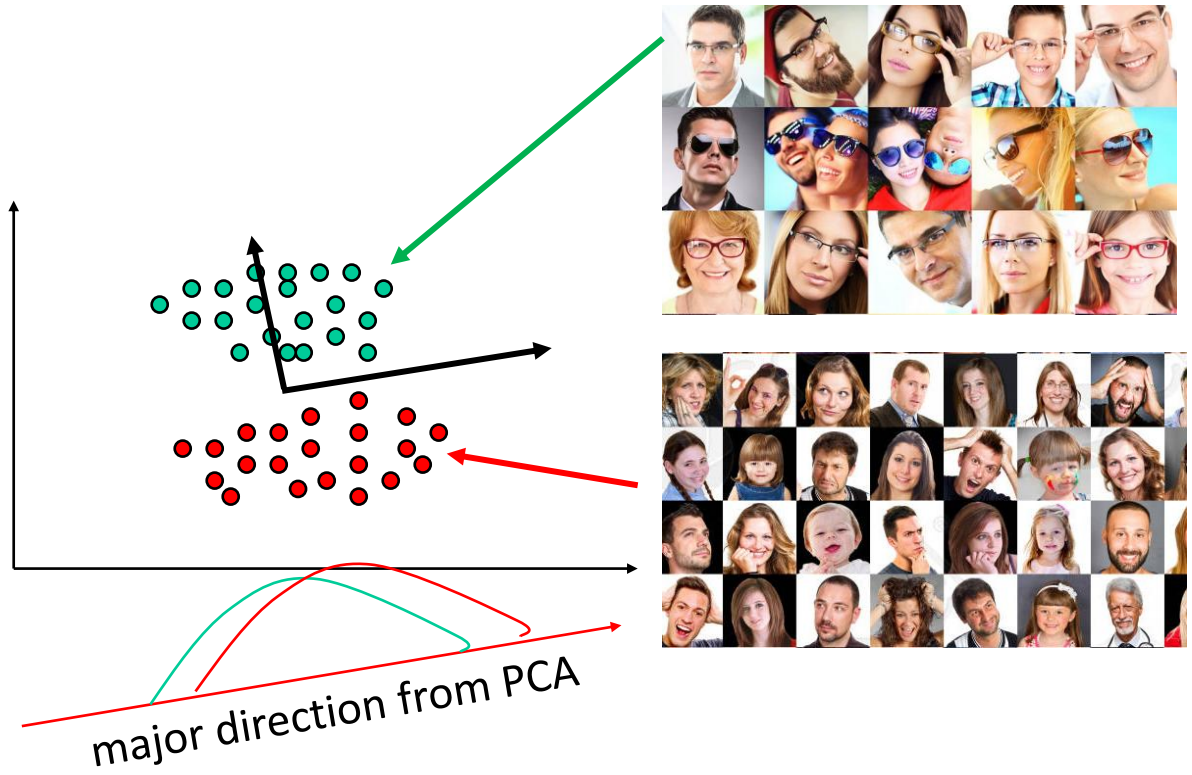

https://dreamfusion3d.github.io/
Poole et al., DreamFusion: Text-to-3D using 2D Diffusion, 2022

*Prompt: "A zoomed out DSLR photo of a squirrel DJing"*



# PCA feature construction issues

- PCA minimizes reprojection (reconstruction) error




Task: Determine whether the person's wearing glasses.

- Reconstruction subspace is not necessarily optimal for *discrimination*
- How to learn potentially "local" discriminative features?

# Time/computation criticality

- A lot of applications are time- and resources-critical
- A case study: Face detection
- Seminal work in realtime detection: Viola&Jones face detector


Viola, M. Jones, Robust Real-Time Face Detection, IJCV, Vol. 57(2), 2004.

# How to come up with features?

1. Natural coordinate systems:

   *For some applications, it is enough just to linearly transform the input data.* PCA

2. Feature selection: ⇦

   *Machine learning to select optimal features from a pool of several handcrafted transforms.*

3. Handcrafted nonlinear transforms:

   *Nonlinear transforms improve feature robustness.*

4. End-to-end learning of feature transform:

   *Have machine learn entire feature extraction and selection pipeline.*

Machine Perception

# LEARNING FEATURES BY FEATURE SELECTION

# Case study: Face detection

- Compact objects → sliding window
- To achieve real-time processing
  1. Feature extraction should be fast
  2. Classifier application should be fast

Viola, M. Jones, Robust Real-Time Face Detection, IJCV, Vol. 57(2), 2004.

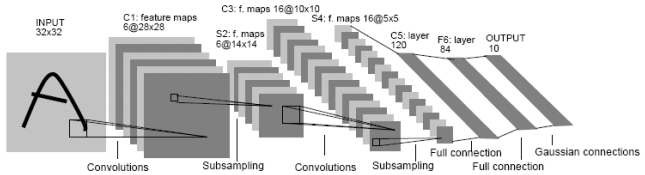
# Choosing the appropriate classifier

## Nearest neighbor

$10^6$ examples

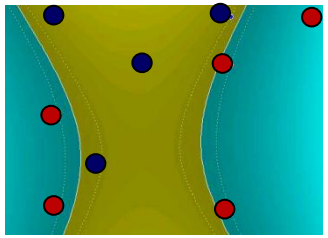
Shakhnarovich, Viola, Darrell 2003
Berg, Berg, Malik 2005...

## Neural networks



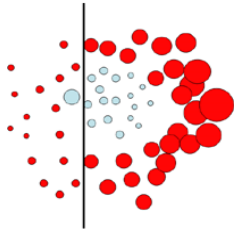

LeCun, Bottou, Bengio, Haffner 1998
Rowley, Baluja, Kanade 1998
…

## Support Vector Machines



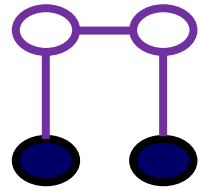
Guyon, Vapnik
Heisele, Serre, Poggio, 2001,…

## Boosting



Viola, Jones 2001,
Torralba et al. 2004,
Opelt et al. 2006,…

## Conditional Random Fields



McCallum, Freitag, Pereira 2000; Kumar, Hebert 2003, …

# AdaBoost: Intuition

- Task: Build a classifier which is a weighted sum of many (weak) classifiers
- Weak == fast to compute (!)

Final (strong) classifier

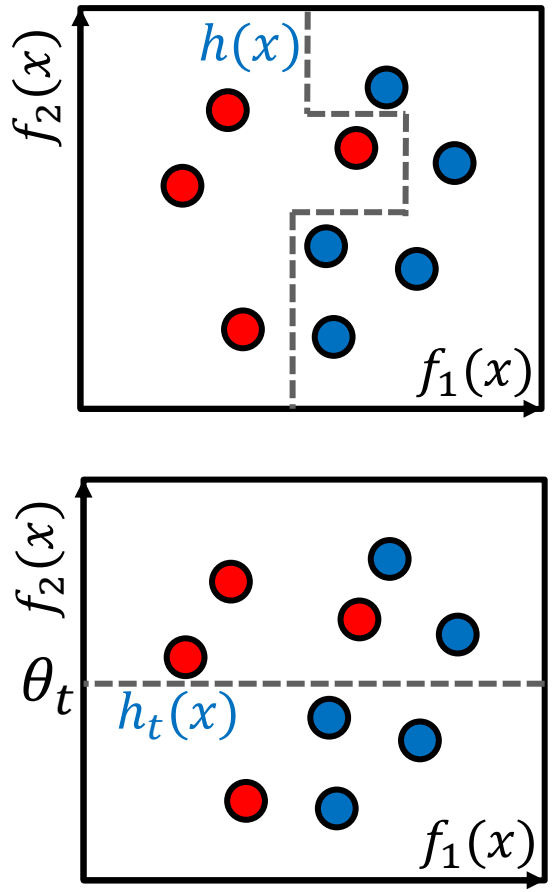
weight

weak classifier

$$h(x) = \text{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

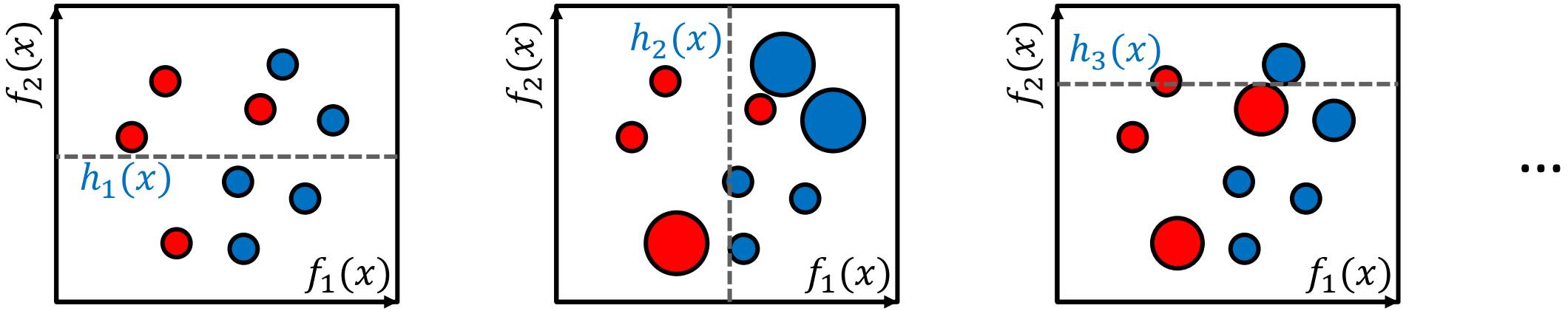
Example of a weak classifier:

$$h_t(x) = \begin{cases} +1 & \text{if } f_t(x) > \theta_t \\ -1 & \text{otherwise} \end{cases}$$




Y. Freund and R. Schapire, A short introduction to boosting, *JSAI*, 1999.

# AdaBoost: Intuition




- Train a sequence of weak classifiers.
- Each weak classifier splits training examples with at least 50% accuracy.
- Those examples that are incorrectly classified by the weak classifier, get more weight in training the next weak classifier.

The final classifier is a combination of many weak classifiers!

$$h(x) = \text{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Y. Freund and R. Schapire, A short introduction to boosting, *JSAI*, 1999.

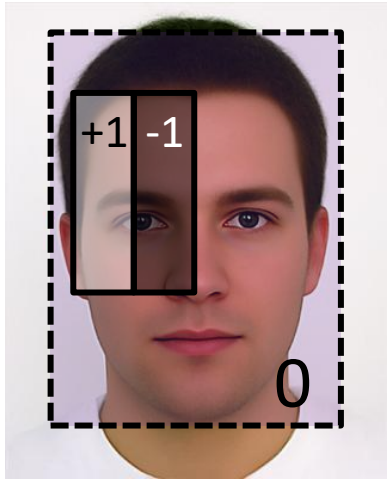
# Case study: Face detection

- Compact objects → sliding window
- To achieve real-time processing
  1. Feature extraction should be fast
     (How to calculate fast/strong features)
  2. Classifier application should be fast
     (ensable of weak classifiers = fast evaluation)

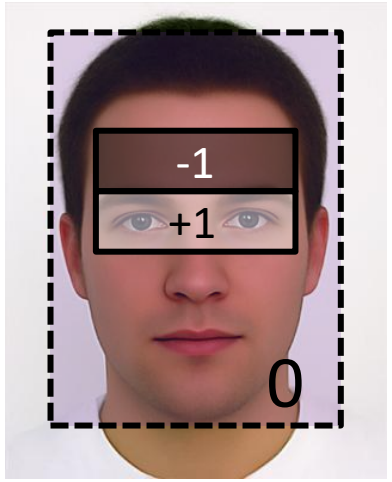
Viola, M. Jones, Robust Real-Time Face Detection, IJCV, Vol. 57(2), 2004

# Computing features

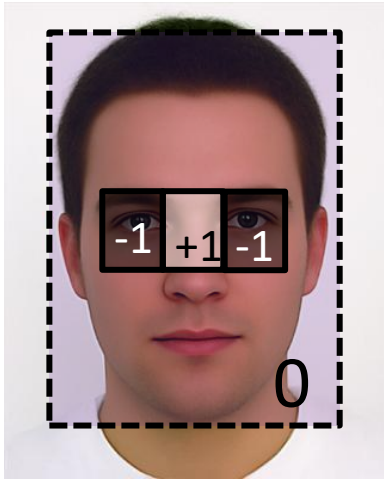
Simple rectangular filters as feature extractors (feature defined by filter type and position)



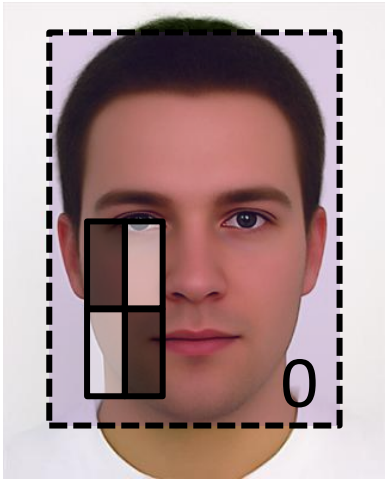

$f_1(x)$




$f_2(x)$

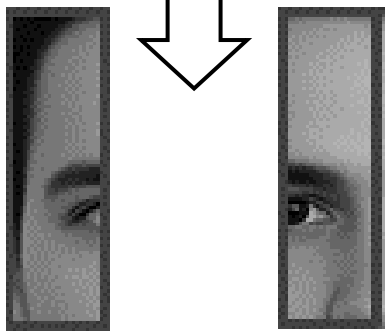



$f_3(x)$




$f_4(x)$

...



Sum=1500 Sum=2000

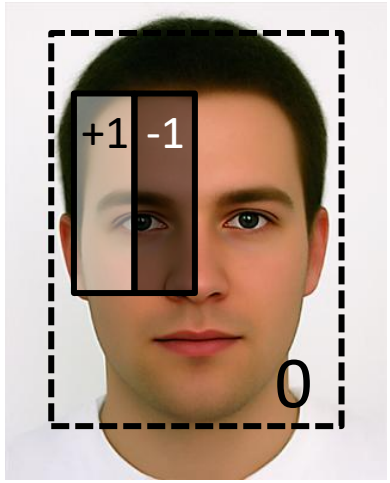
\+ -

$f_1(x) = -500$

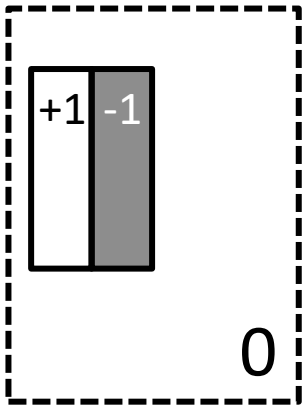
The value of each feature is the difference between the intensity in „black“ and „white“ regions.
Black is weighted as -1, white as +1.

# Computing features

Simple rectangular filters as feature extractors




$f_1(x)$

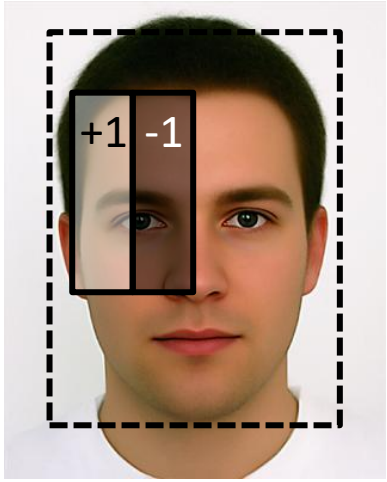


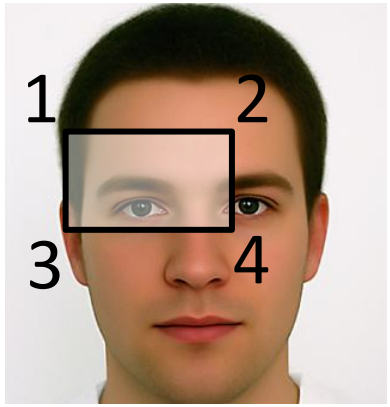

Require evaluation at many displacements and multiple scales!

Possible to evaluate such a simple filter efficiently!

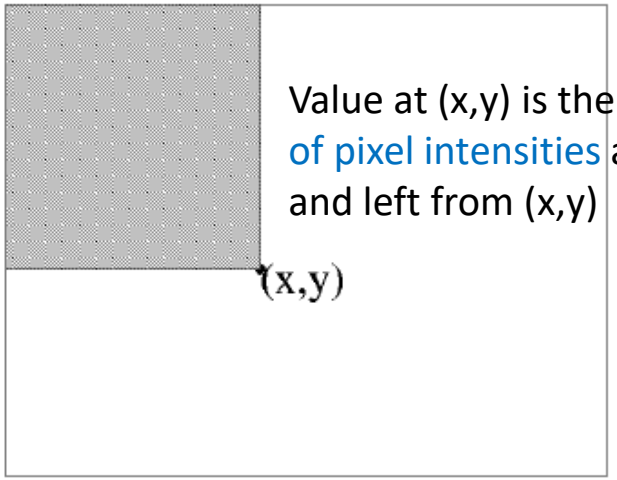
# Efficient computation – Integral images

- Our filters are based on sums of intensities within rectangular regions.
- This can be done in constant time for arbitrary large region!
- Require precomputing the integral image.

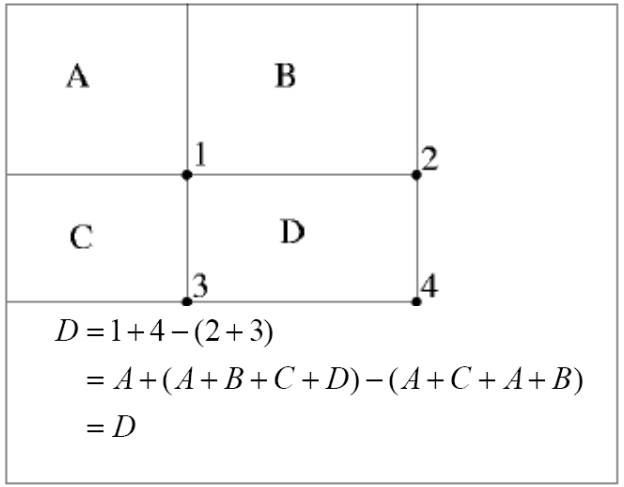



Integral image

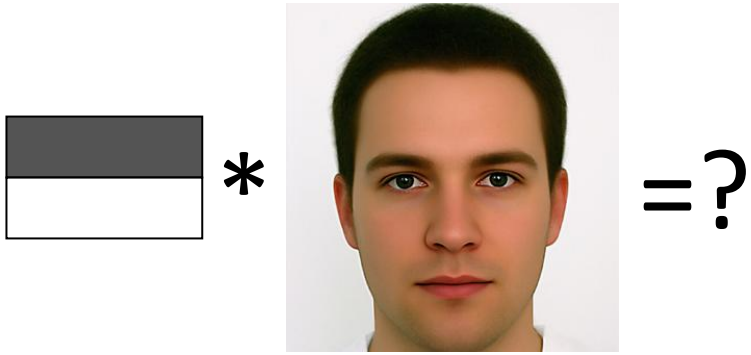


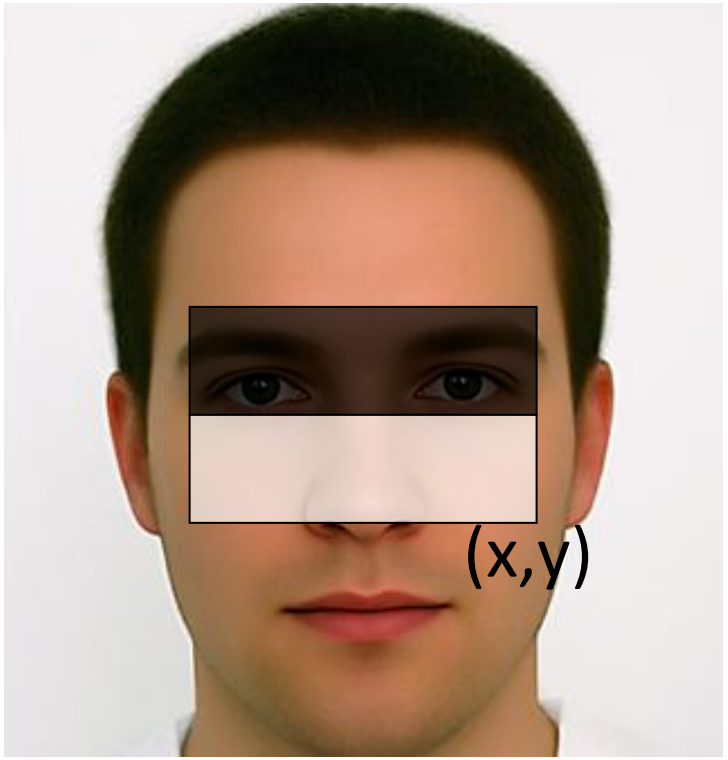

Integral image

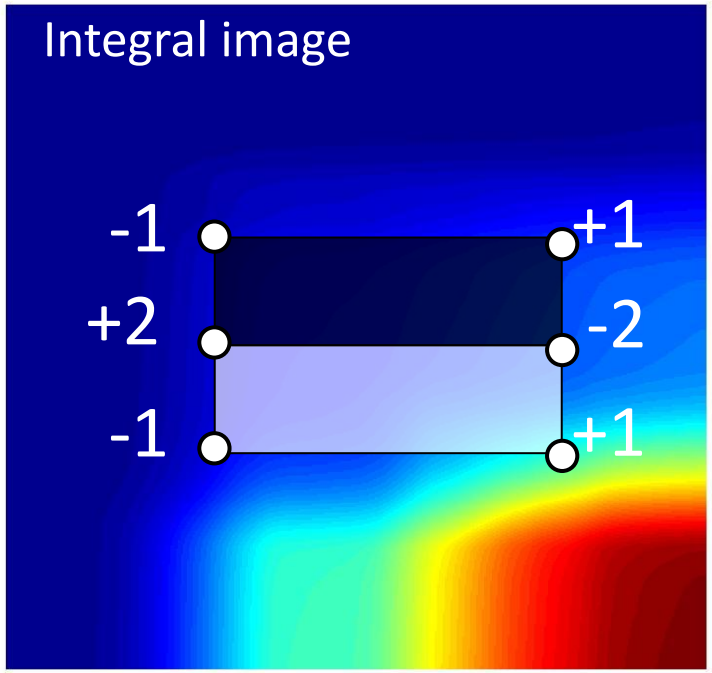


# Efficient computation – Integral images

- Consider a more complex filter

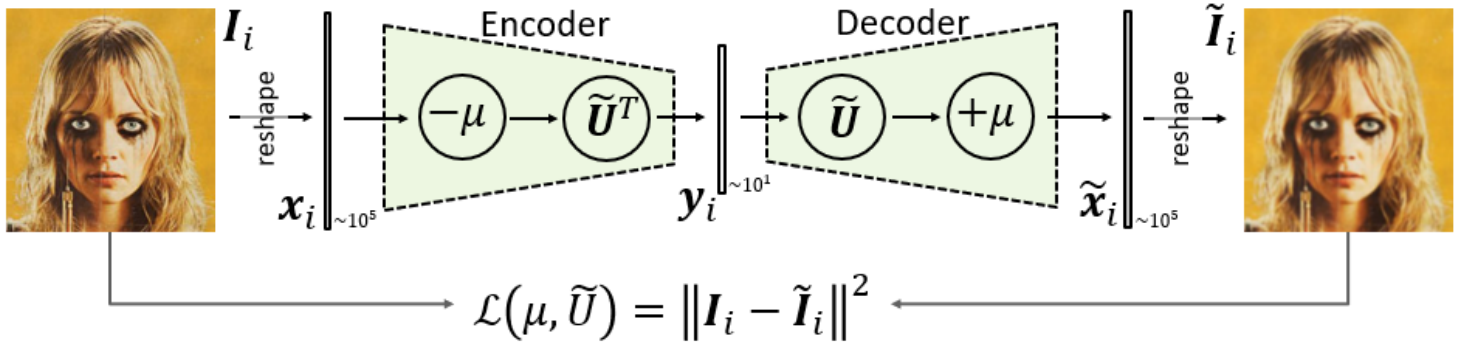


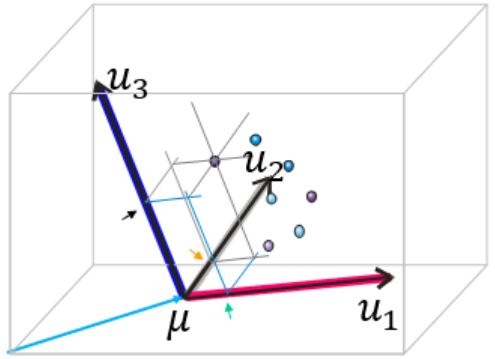



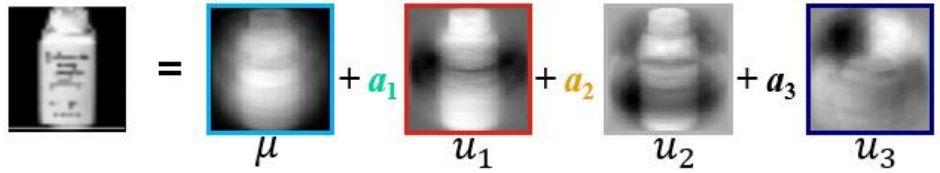



# Previously at MP...

- Learning features by subspace construction: PCA

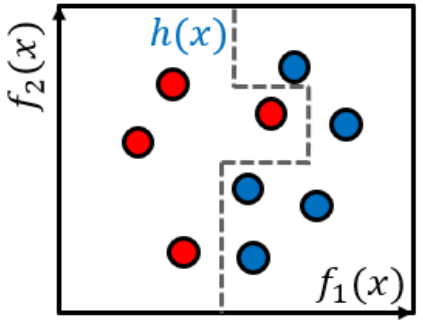


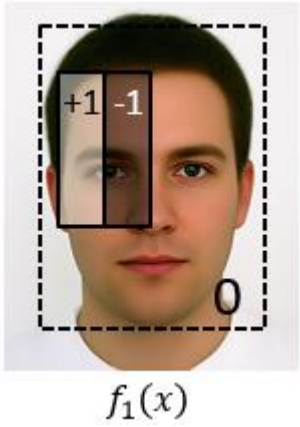




- Learning features by feature selection

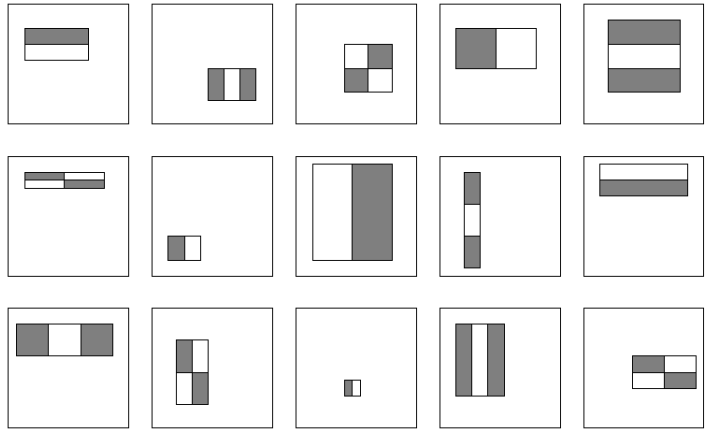
  (A case study: Viola-Jones face detector)



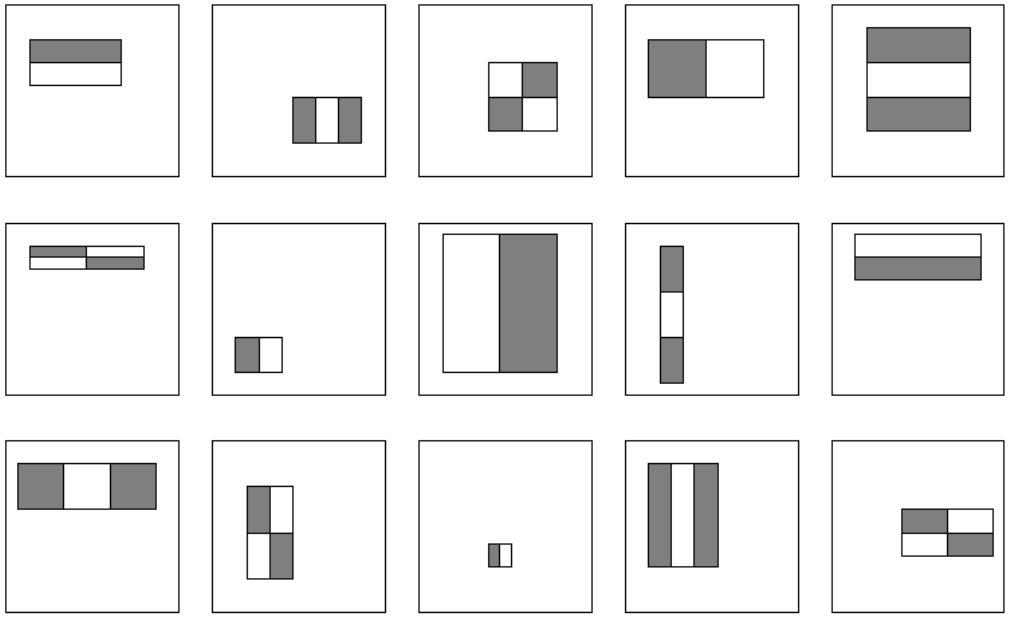

$$h(x) = \text{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$




Fast feature extraction



# Large collection of filters



etc...

Account for all possible parameters:
position, scale, type

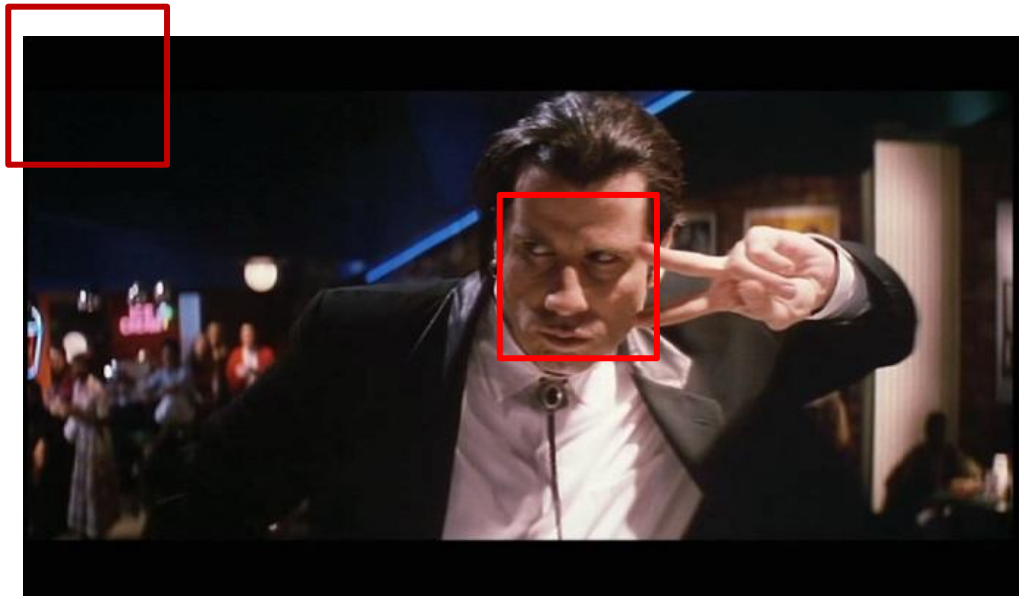
More than 180,000
different features in a 24x24 window.

Apply Adaboost for
(i) selecting most informative features and
(ii) composing a classifier (weights+thresholds).

$$h(x) = \text{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

Viola, M. Jones, Robust Real-Time Face Detection, IJCV, Vol. 57(2), 2004

# Efficiency issues



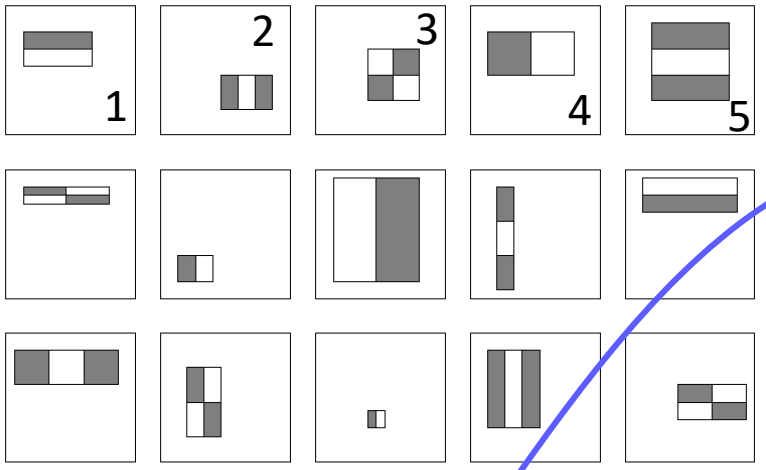
Extract features at each bounding box and apply Adaboost classifier.

$$h(x) = \text{sgn}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

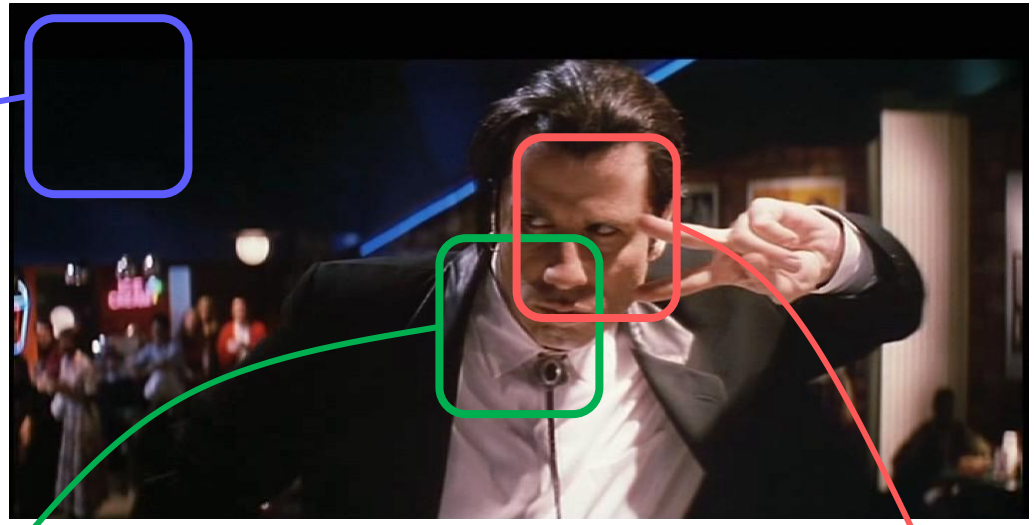
- Filter responses can be evaluated fast.
- But each image contains a lot of windows, that we need to classify – potentially great amount of computation!
- How to make detection efficient?

# Cascade of classifiers

- Efficient: Apply first few classifiers (fast), to reject the windows that obviously do not contain the particular category! Then re-classify the regions that survived, with stronger classifiers.



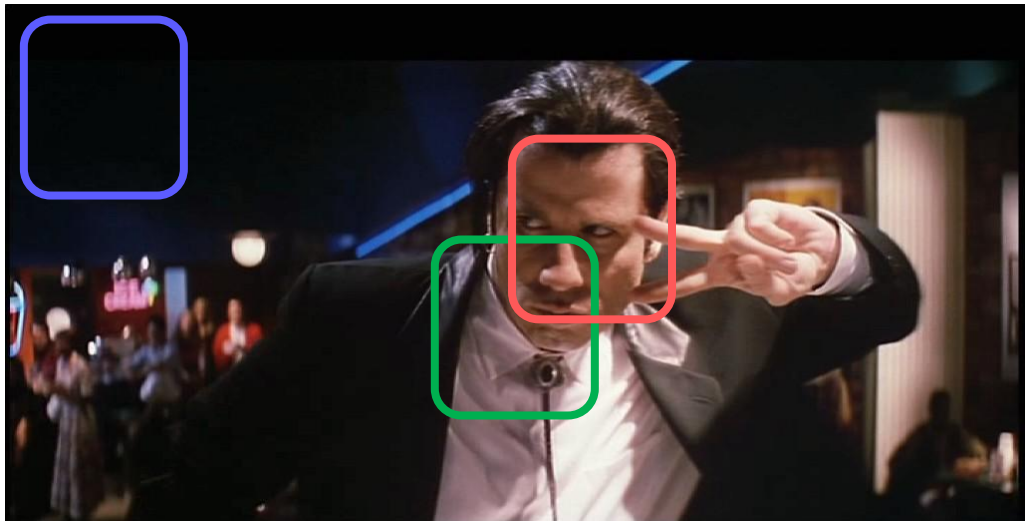

...

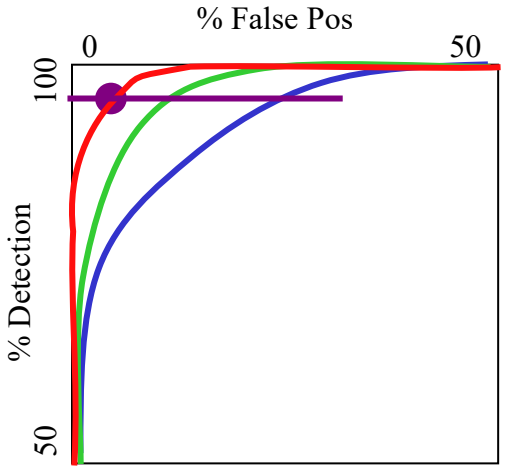


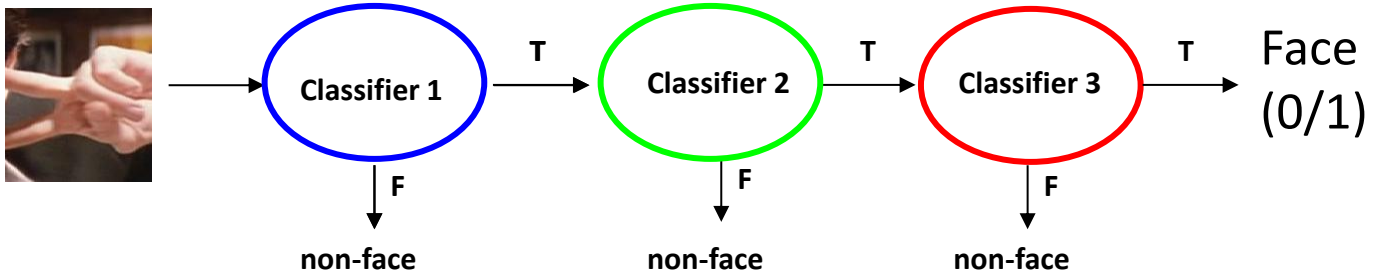
$$h(x) = \text{sgn}(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x) + \alpha_4 h_4(x) + \ldots + \alpha_{T-1} h_{T-1}(x) + \alpha_T h_T(x))$$

# Cascade of classifiers

- Chain classifiers from least complex with low true-positive rejection rate to most complex ones

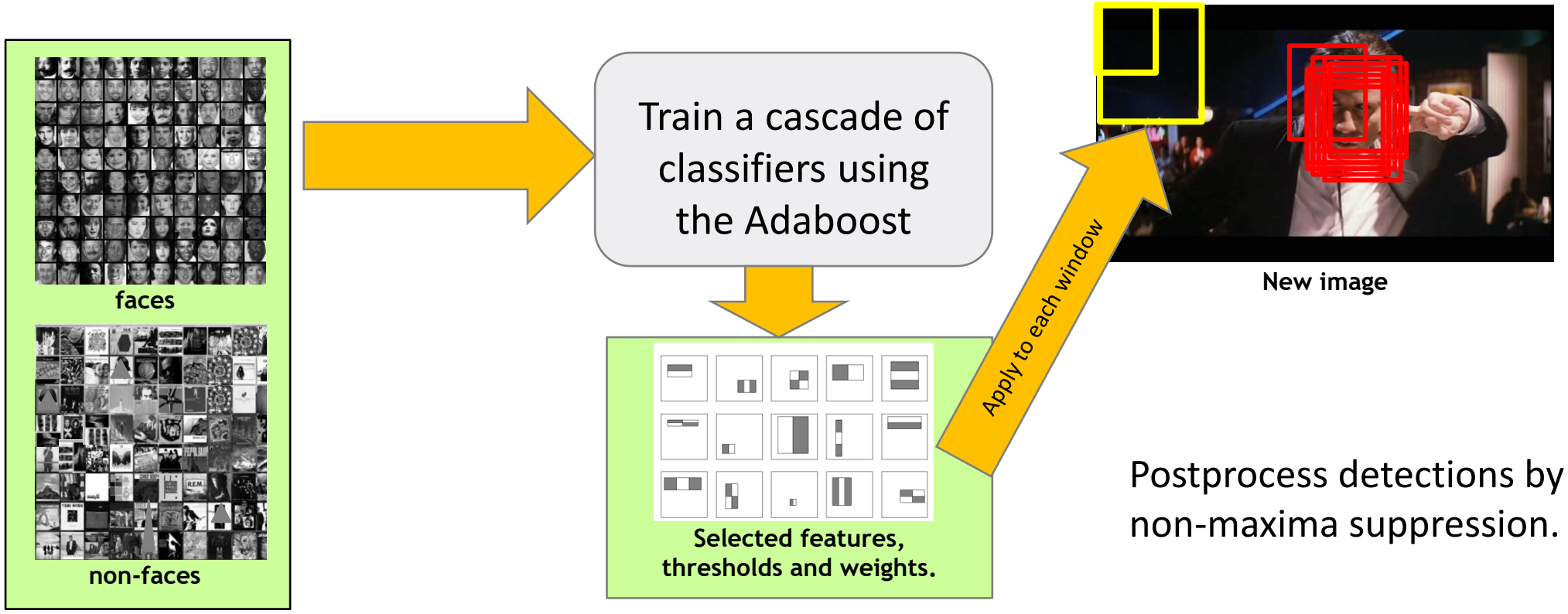





$$h(x) = \text{sgn}(\alpha_1 h_1(x) + \alpha_2 h_2(x) + \alpha_3 h_3(x) + \alpha_4 h_4(x) + \ldots + \alpha_{T-1} h_{T-1}(x) + \alpha_T h_T(x))$$

# Viola-Jones face detector




Postprocess detections by non-maxima suppression.

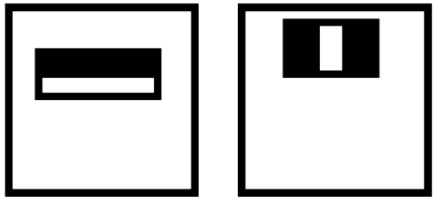
- Train using 5k positives and 350M negatives
- Real-time detector using 38 layers in cascade
- 6061 features in the final layer (classifier)
- [OpenCV implementation:http://sourceforge.net/projects/opencvlibrary/]

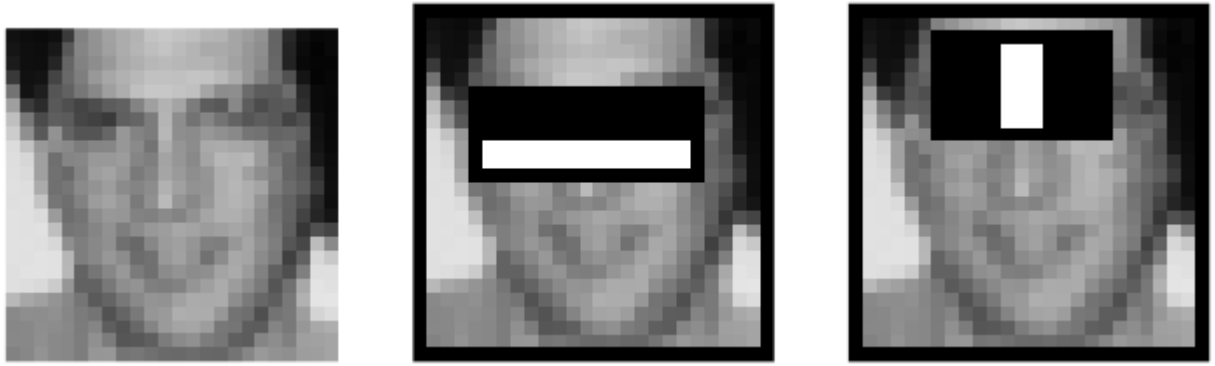
384x288 images, detection 15 fps on 700 MHz Intel Pentium III desktop (2001).
Training time = weeks!

# Viola-Jones face detector

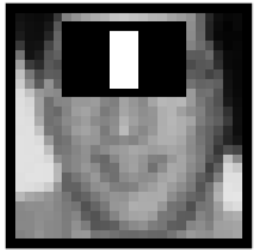

- The video visualizes all the “features”, i.e., filter responses checked in a cascade.
- Observe the increase of cascade once close to face.

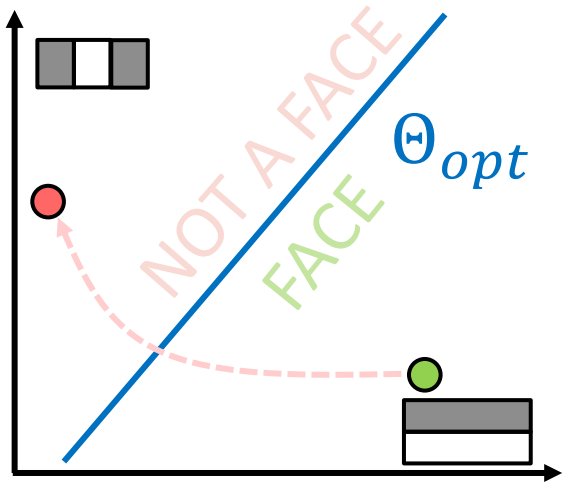
The major two features:

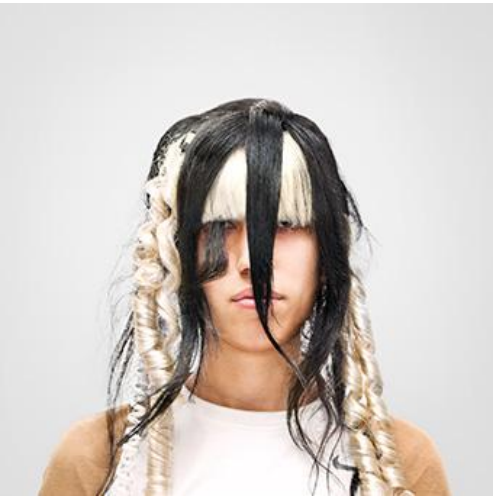




# Becoming invisible to the computer

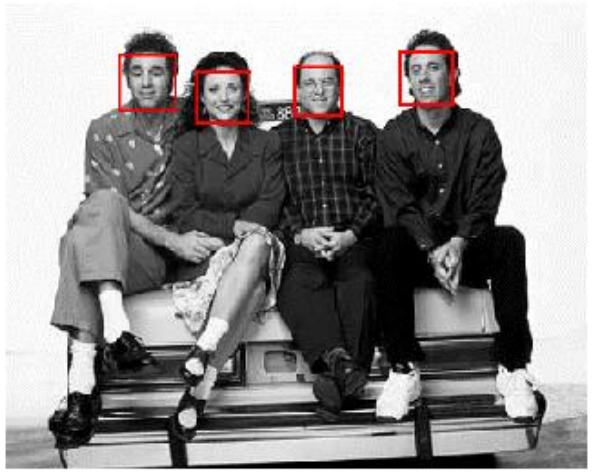


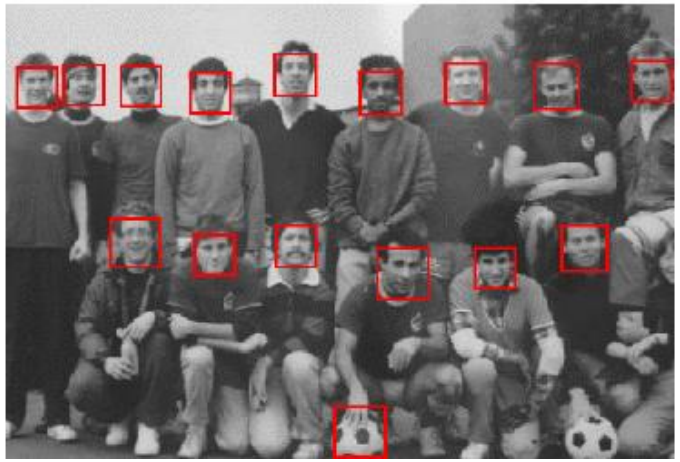


(2010) http://cvdazzle.com

# Viola-Jones: results



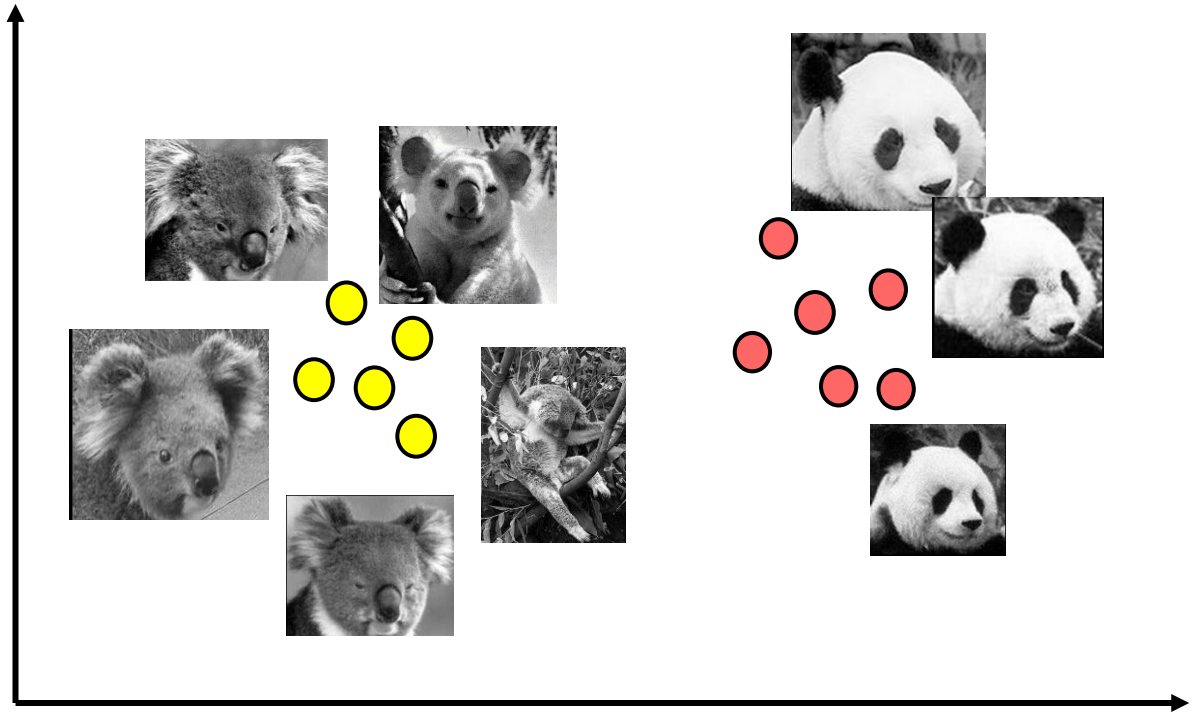



Viola, Jones, *"Rapid Object Detection using a Boosted Cascade of Simple Features"*, CVPR2001

Viola-Jones detection image credit: Žiga Emeršič

# Much stronger features required in general

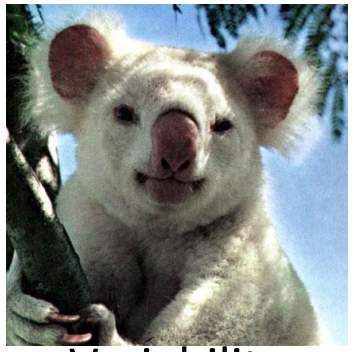
- Require a representation/features that:
  - Accounts for intra-class variation (contrast, illumination, occlusion, etc.)
  - Distinguishes between different classes



Illumination

Occlusion

Aspect



Variability

Pose

# How to come up with features?

1. Natural coordinate systems:

   *For some applications, it is enough just to linearly transform the input data.* PCA

2. Feature selection:

   *Machine learning to select optimal features from a pool of several handcrafted transforms.* Boosting

3. Handcrafted nonlinear transforms: ⇦

   *Nonlinear transforms improve feature robustness.*

4. End-to-end learning of feature transform:

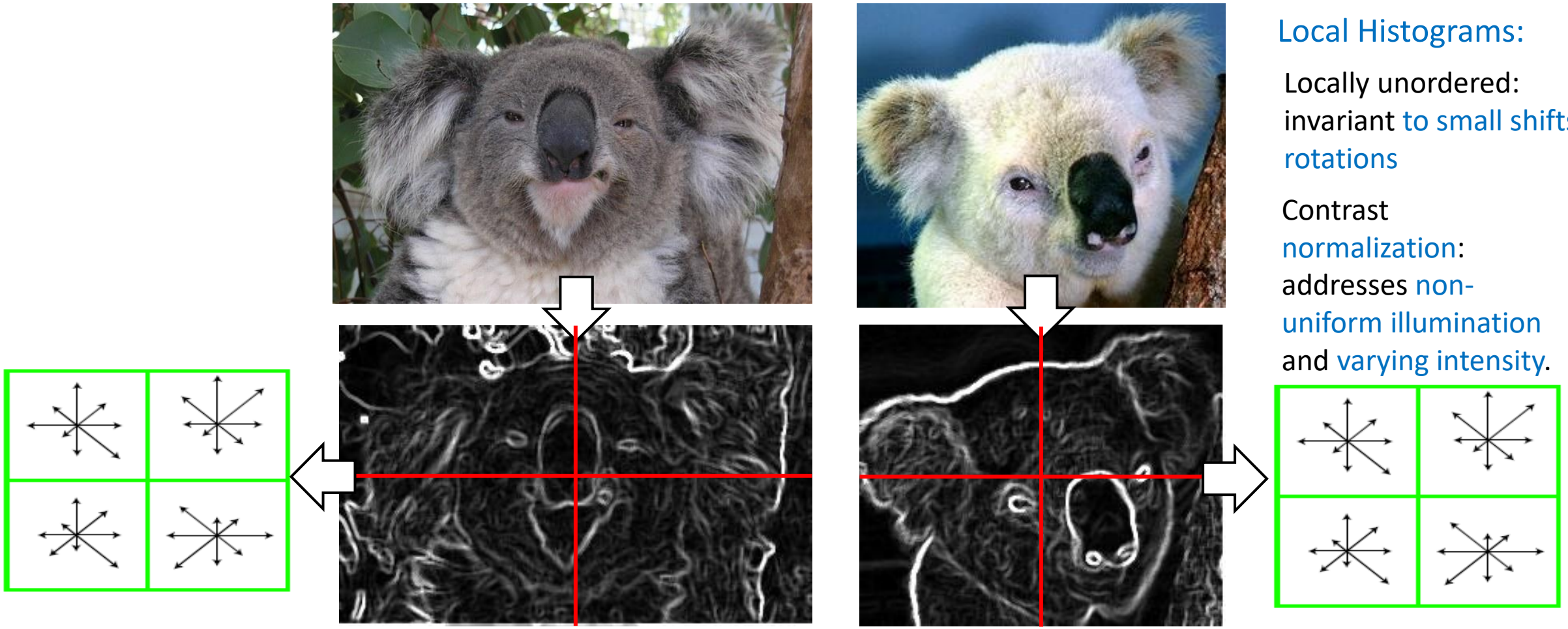
   *Have machine learn entire feature extraction and selection pipeline.*

Machine Perception

# HANDCRAFTED NONLINEAR TRANSFORMS

# Hand-crafting global features

- Problem: Color or gray-level representation is sensitive to illumination changes or within-class color variations.

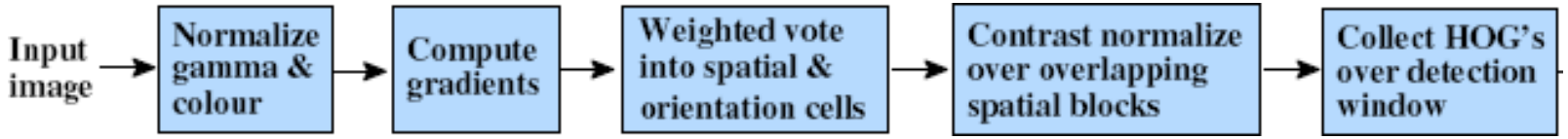


Local Histograms:

Locally unordered: invariant to small shifts and rotations

Contrast normalization: addresses non-uniform illumination and varying intensity.

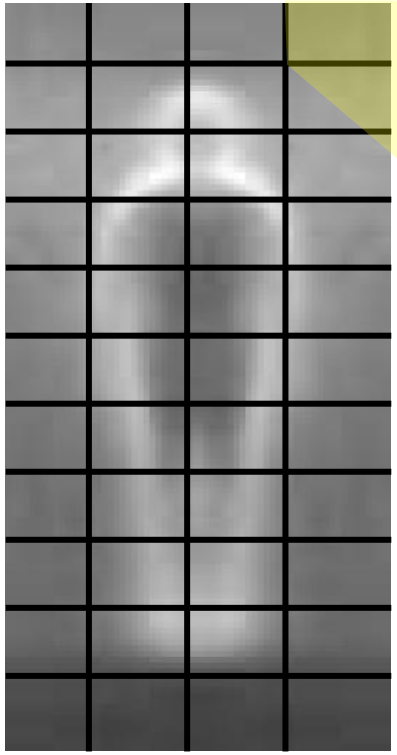
# Gradient-based representation: HOG



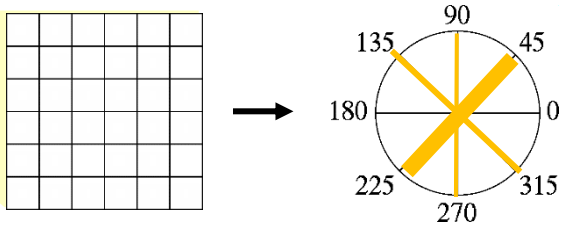

Navneet Dalal and Bill Triggs , Histograms of Oriented Gradients for Human Detection, CVPR 2005

Image

Sobel (gradient)

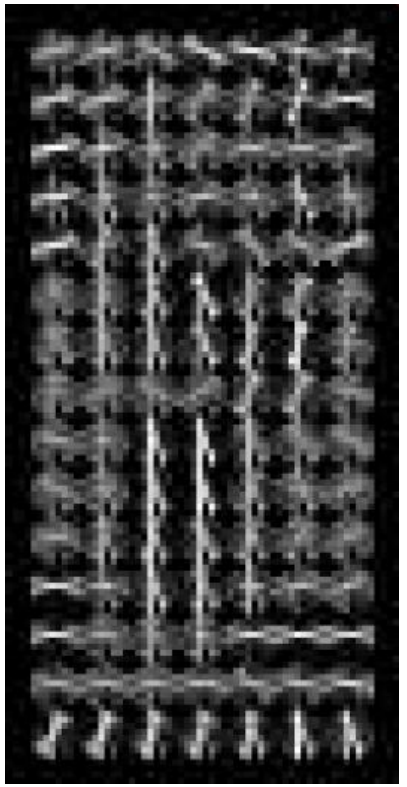


Calculate hog in $8 \times 8$ blocks and normalize




- Histogram of gradient orientations
  - Weighted by magnitude
  - Similar to SIFT

HOG descriptor

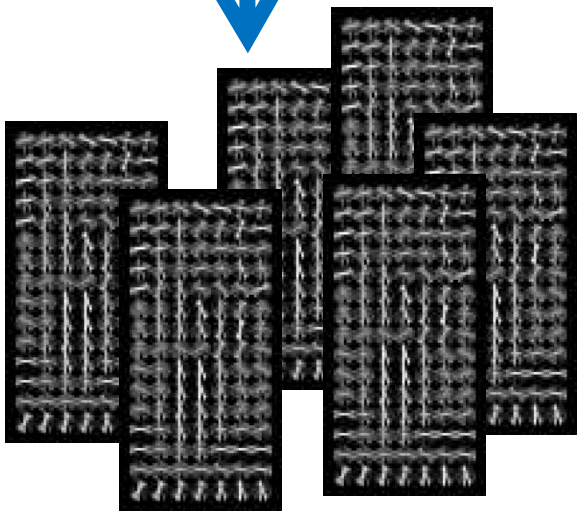


# Practical approach to learning a detector

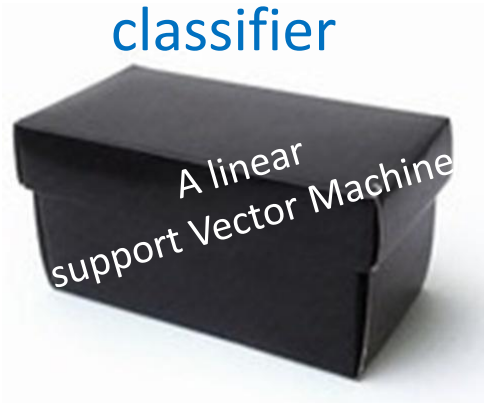
Persons

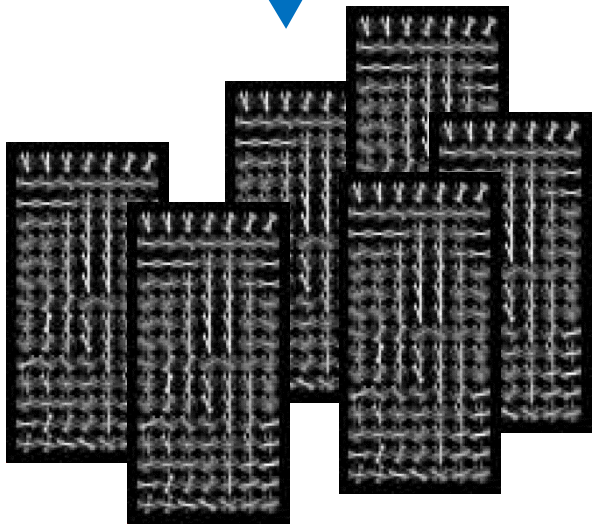
Background

Extract HOGs



Train a person/nonperson classifier



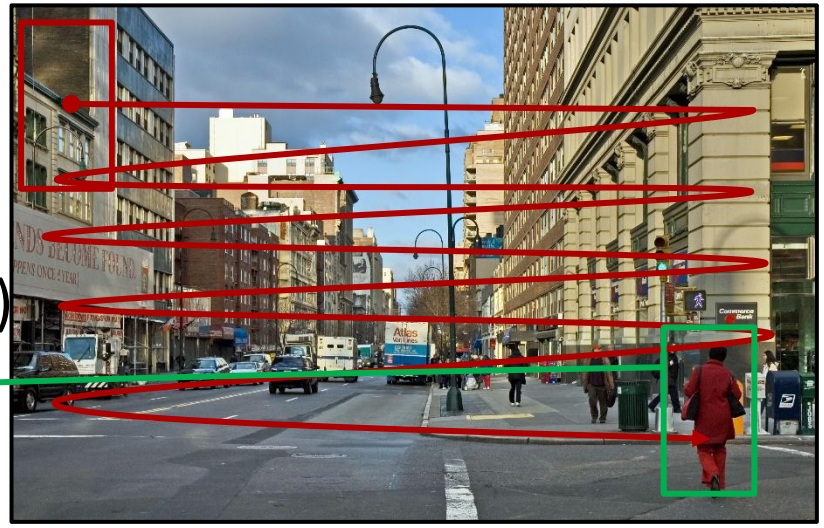
Extract HOGs

# Application: Pedestrian detection

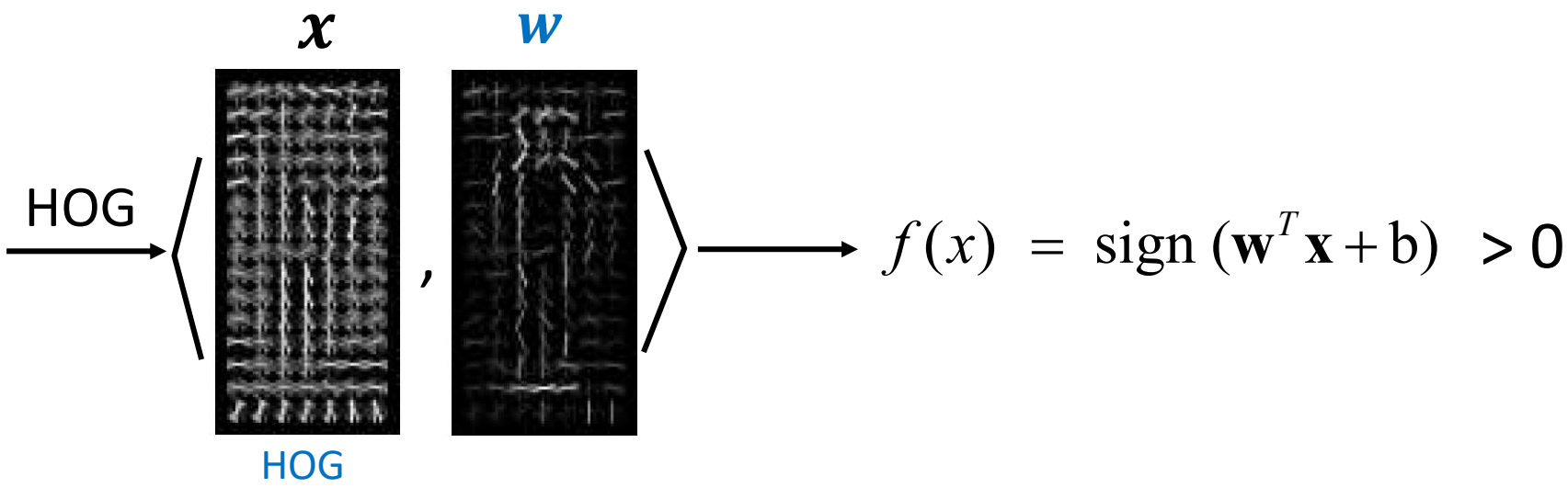
- Sliding window:

1. extract HOG at each displacement
2. classify by a linear support vector machine (SVM)

   (in principle, *convolution*)



Crop




$$f(x) = \text{sign}\,(\mathbf{w}^T\mathbf{x} + \mathrm{b}) > 0$$

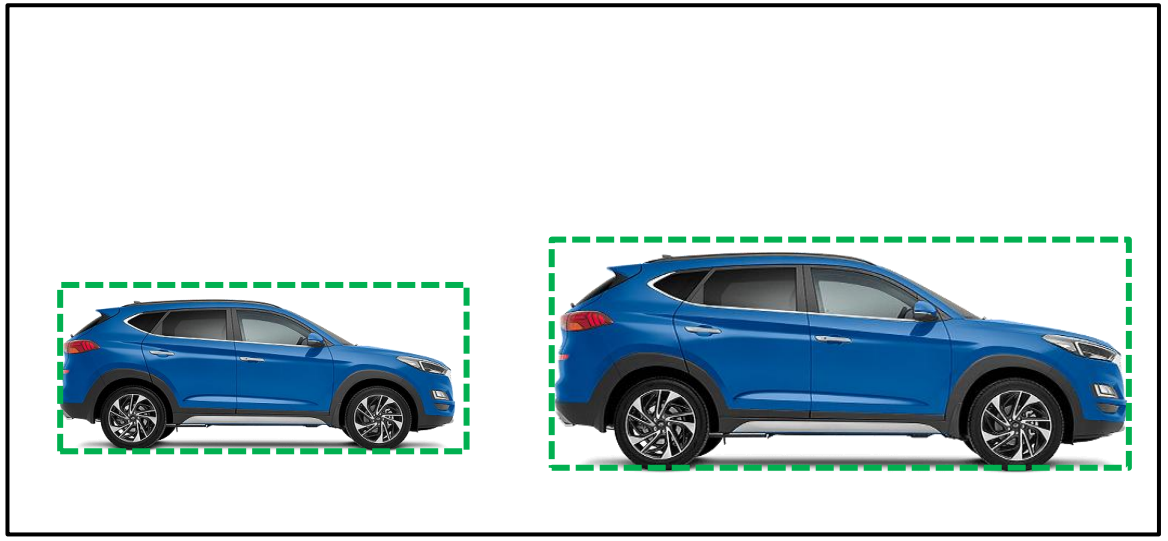
Dalal and Triggs, Histograms of oriented gradients for human detection, CVPR2005

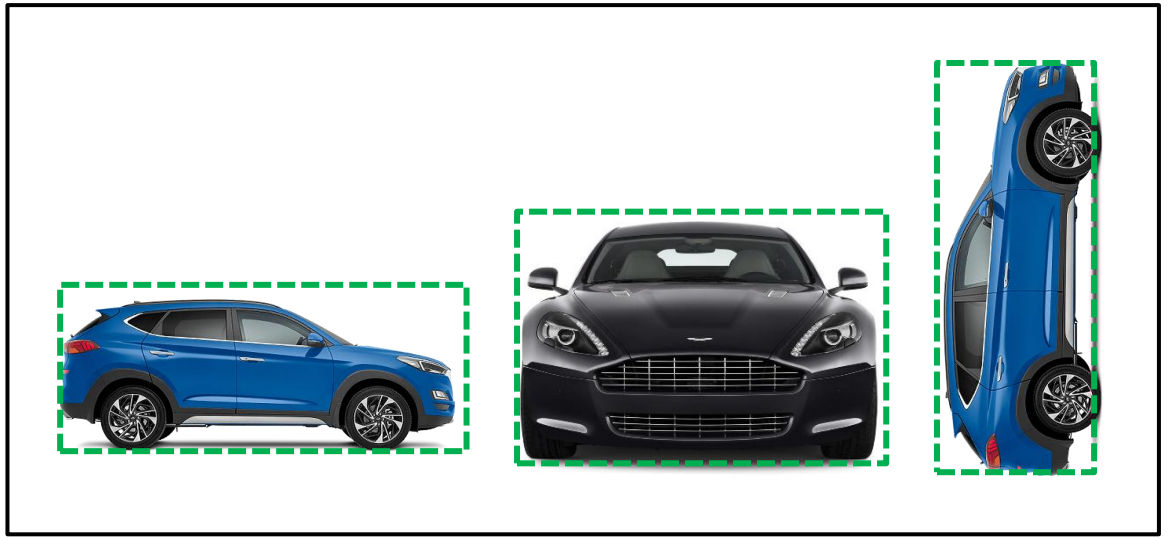
# Pedestrian detection HoG+SVM


Navneet Dalal, Bill Triggs, Histograms of Oriented Gradients for Human Detection, CVPR 2005

# Issues with sliding window object detection

- Strengths:
  - Simple to implement
  - Can deal with scale changes (e.g., by pyramid implementation)
- Weaknesses:
  - Adding aspect change (nonconstant width-to-height ratio) significantly increases the number of bounding boxes to test
  - Feature construction for each box increases computational complexity

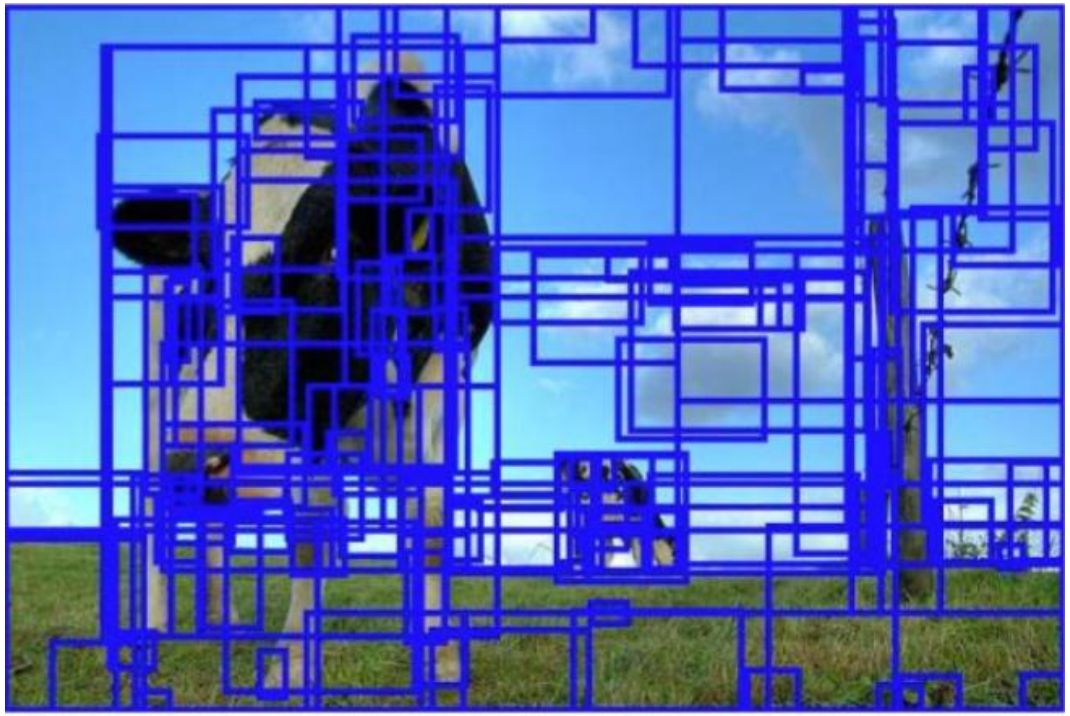




# Region proposals

- Apply a cascade: (i) generate small set of regions (~5000), (ii) evaluate a potentially computationally demanding classifier only on each region

?



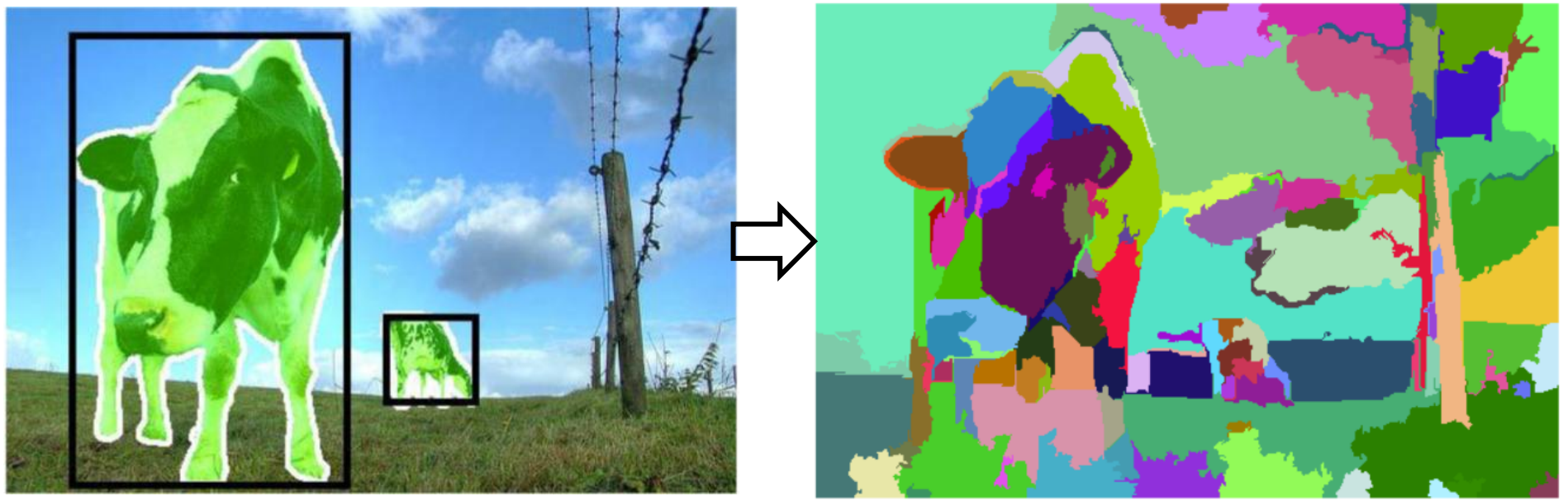
Generated region proposals

After verification with a “cow” classifier

*Sande, et al.,* *Segmentation* as Selective Search for Object Recognition, ICCV 2011

# Selective search

- Insight: Images are intrinsically hierarchical
- Start by over-segmentation into small regions

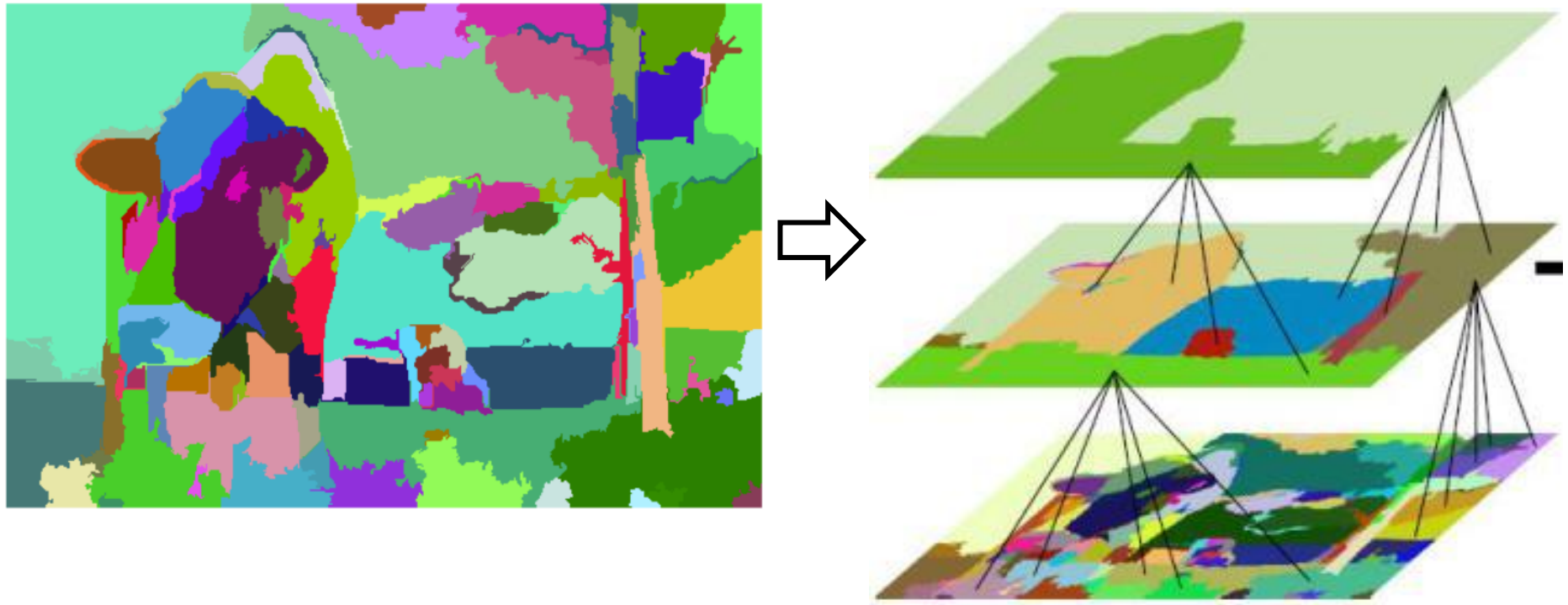


"Efficient graph-based image segmentation" Felzenszwalb and Huttenlocher, IJCV 2004

*Sande, et al.,* *Segmentation* as Selective Search for Object Recognition, ICCV 2011

# Selective search

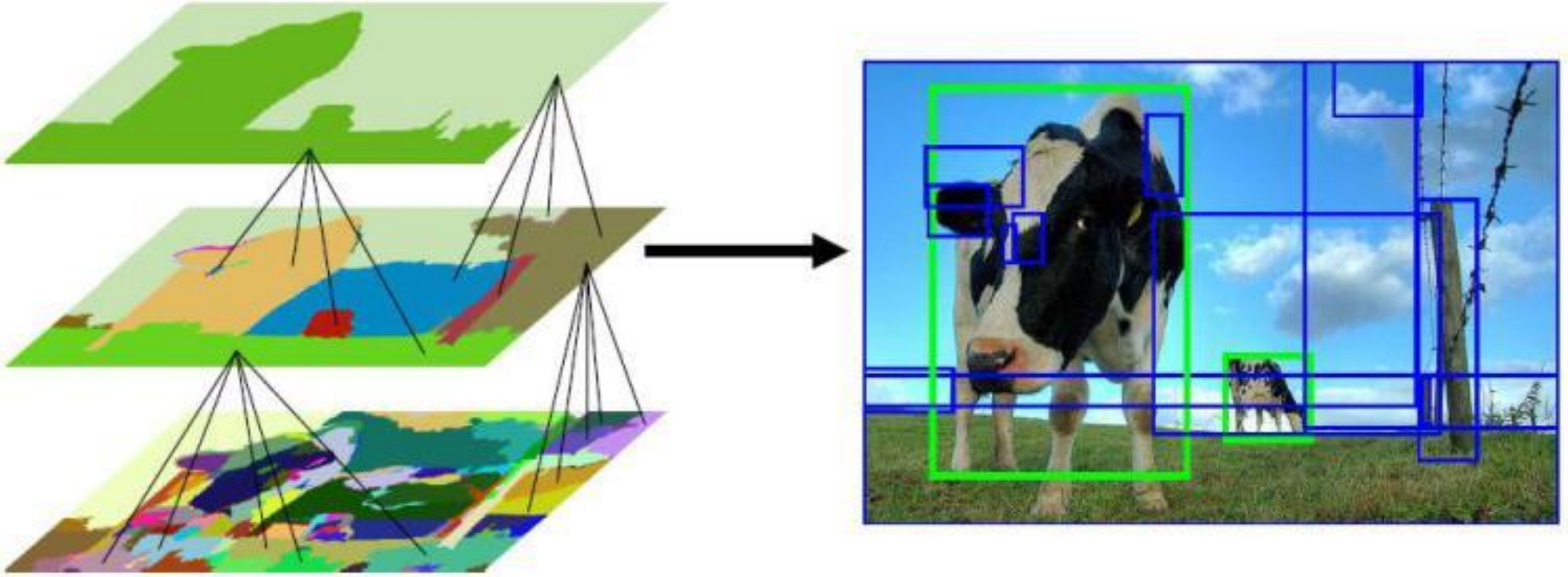
- Merge two most similar regions based on texture similarity and region size
- Continue until a single region remains.



*Sande, et al.,* *Segmentation* as Selective Search for Object Recognition, ICCV 2011

# Selective search

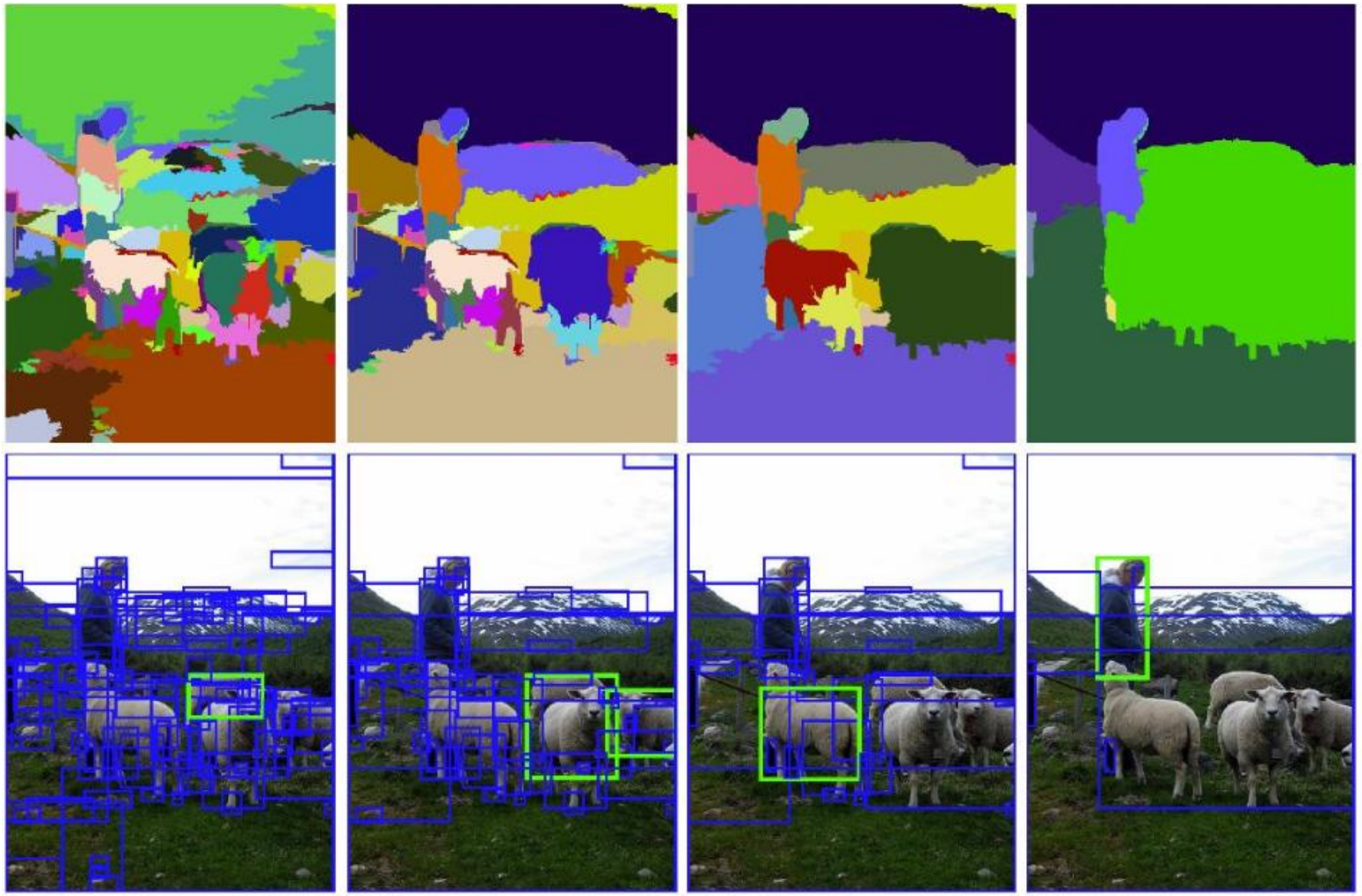
- From each merged region generate a bounding box



*Sande, et al.,* Segmentation as Selective Search for Object Recognition, ICCV 2011

# Selective search

- High recall
- Object-category agnostic!
- Has been the basis for many state-of-the-art computationally-heavy detectors
- *! But still learning powerful feature descriptors & classifiers remained an open Q.*



*Sande, et al., Segmentation as Selective Search for Object Recognition, ICCV 2011*

# References

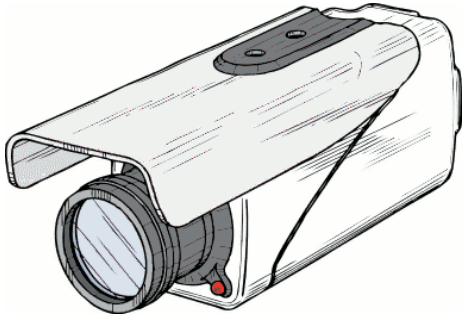

- David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition), (prva izdaja dostopna na spletu)
- R. Szeliski,Computer Vision: Algorithms and Applications, Springer, 2011
- Viola, M. Jones, Robust Real-Time Face Detection, IJCV, Vol. 57(2), 2004.
- Viola-Jones Face Detector
  - C++ implementation in OpenCV [Lienhart, 2002]
    - http://sourceforge.net/projects/opencvlibrary/
  - Matlab wrappers:
    - http://www.mathworks.com/matlabcentral/fileexchange/19912

Univerza v Ljubljani

ViCOS visual cognitive systems lab

# Machine perception
# Recognition & Detection 2:
# A modern approach

Matej Kristan




Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

# How to come up with features?

1. Natural coordinate systems:

   *For some applications, it is enough just to linearly transform the input data.* PCA

2. Feature selection:

   *Machine learning to select optimal features from a pool of several handcrafted transforms.* Adaboost

3. Handcrafted nonlinear transforms:

   *Nonlinear transforms improve feature robustness.* HOG

4. End-to-end learning of feature transform: ⇦

   *Have machine learn entire feature extraction and selection pipeline.*

Machine Perception

# END-TO-END FEATURE (AND CLASSIFIER ) LEARNING

# Modern representation learning


(+linear trans.) Classifier → "panda"?


edges, etc. → (+ feat. sel.) Classifier → "panda"?


HOG, SIFT, etc.

edges → histograms → Classifier → "panda"?

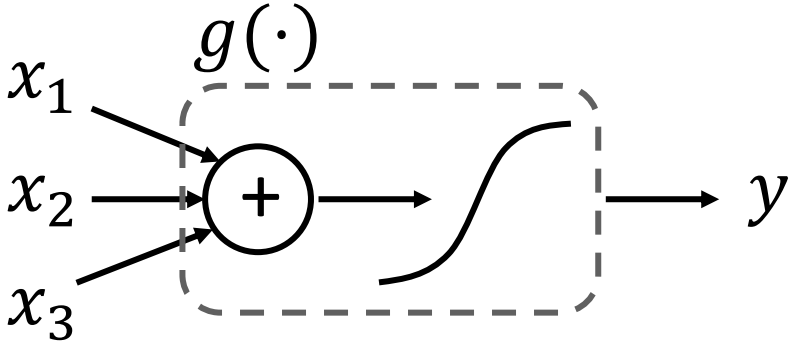

HOG, SIFT, etc.

edges → histograms → K-means sparse code FV/Vlad → Classifier → "panda"?


end-to-end learning of features and classifiers → "panda"?

# Recall a simple neural network

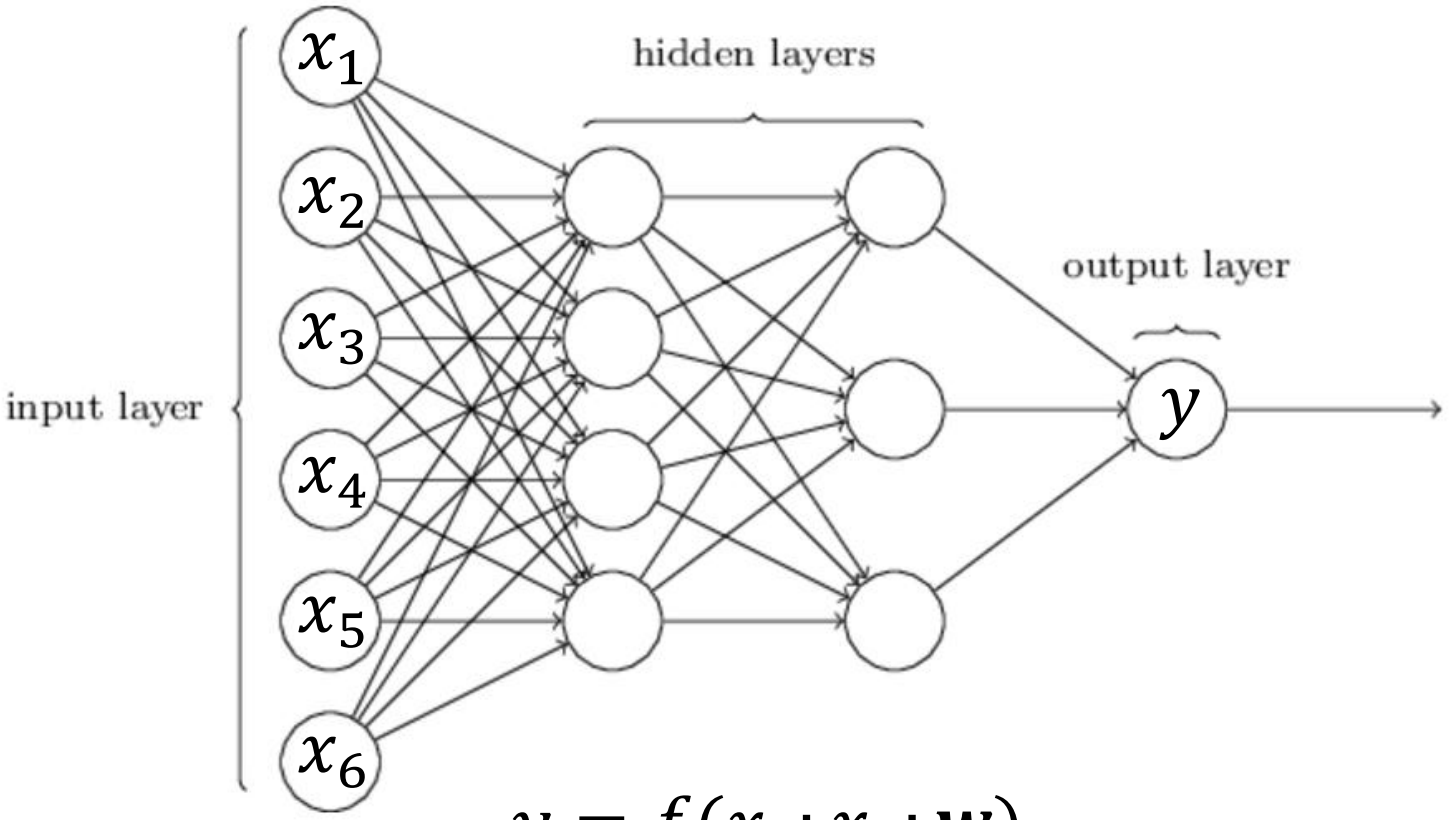
A sigmoid neuron:




A weighted sum of values $x_i$, transformed by a nonlinear function $g(\cdot)$:

$$y = g(\sum_i w_i x_i)$$

A network of neurons:



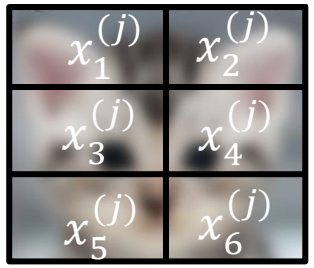

$$y = f(x_1 : x_6; \boldsymbol{w})$$

# Classification problem: training the network

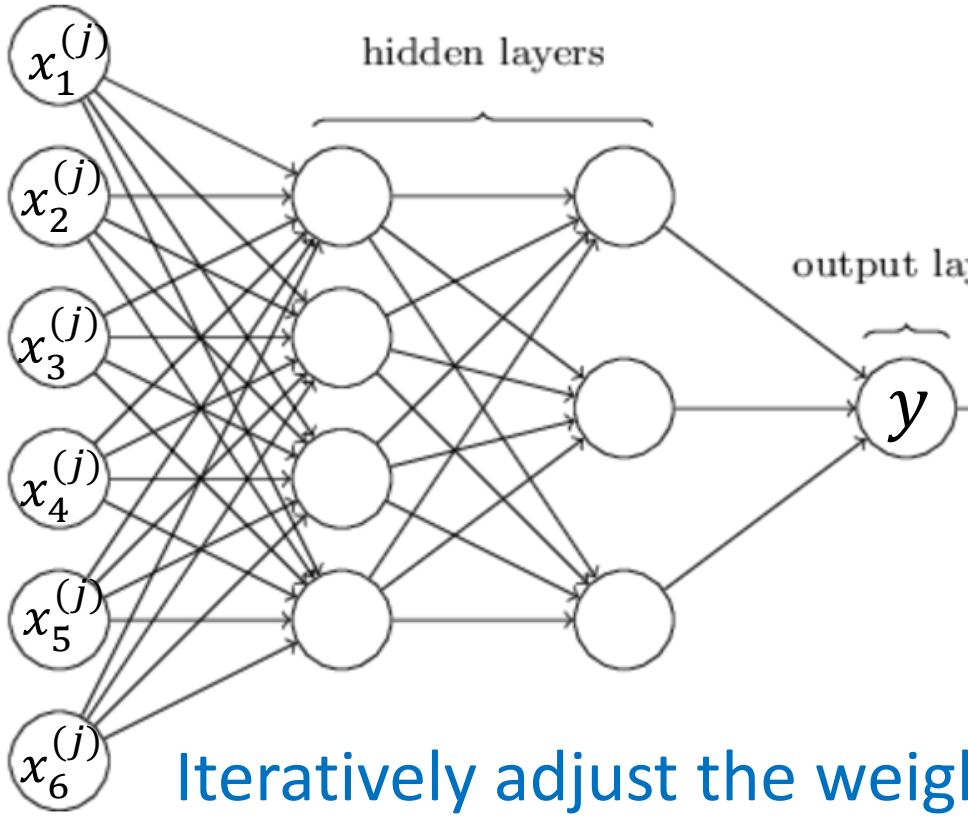
- For an input image $x^{(j)}$, predict probability of cat&dog, i.e., $\mathrm{y}^{(j)} = [p_{cat}, p_{dog}]$

cat dog

$t^{(j)} = [1,0]$




$\boldsymbol{x}^{(j)}$




Cost function (cross entropy):

$$\epsilon(j) = CE(t^{(j)}, f(\boldsymbol{x}^{(j)}; \boldsymbol{w}))$$
$$\epsilon(\boldsymbol{w}) = \sum \epsilon(j)$$

$y^{(j)} = [0.7,0.3]$ (cat dog) $\quad t^{(j)} = [1,0]$

Find optimal parameters:

$$\boldsymbol{w}_{\text{opt}} = \underset{\boldsymbol{w}}{\operatorname{argmin}}\, \epsilon(\boldsymbol{w})$$

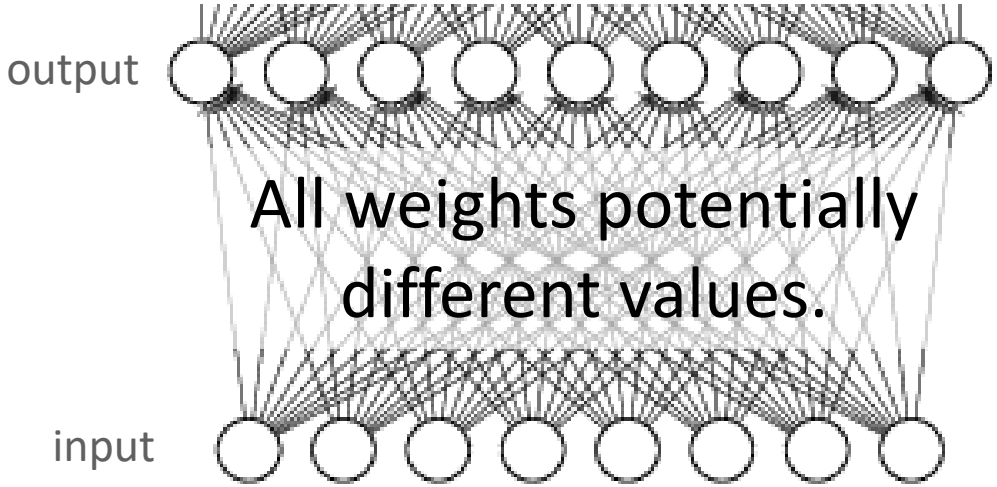
Iteratively adjust the weights to reduce the cost: gradient descent

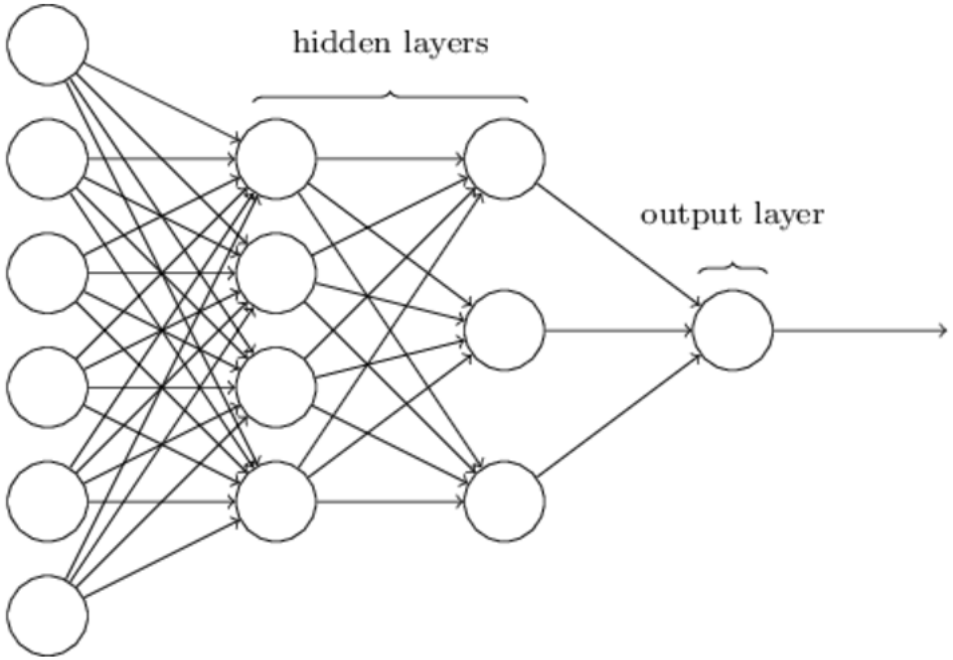
$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \frac{\partial \epsilon(\boldsymbol{w})}{\partial \boldsymbol{w}}$$

Efficient implementation: The backpropagation algorithm

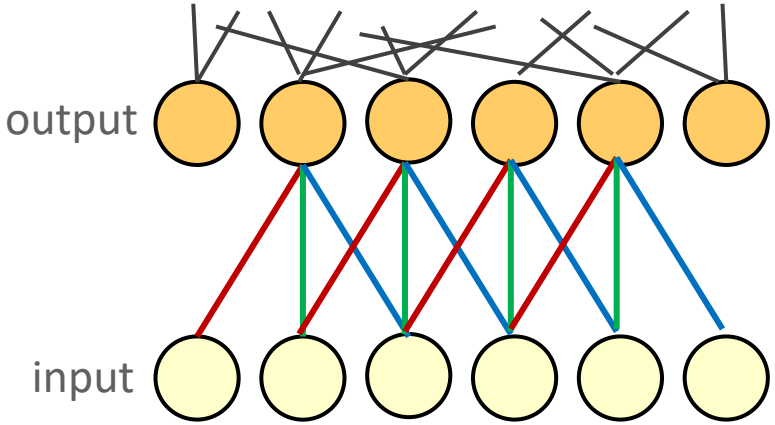
# Put some structure in neural networks: CNN

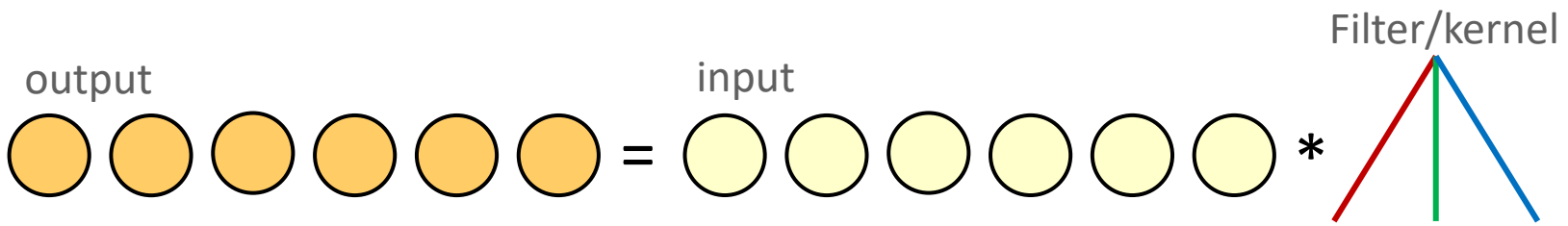
- A fully connected neural network



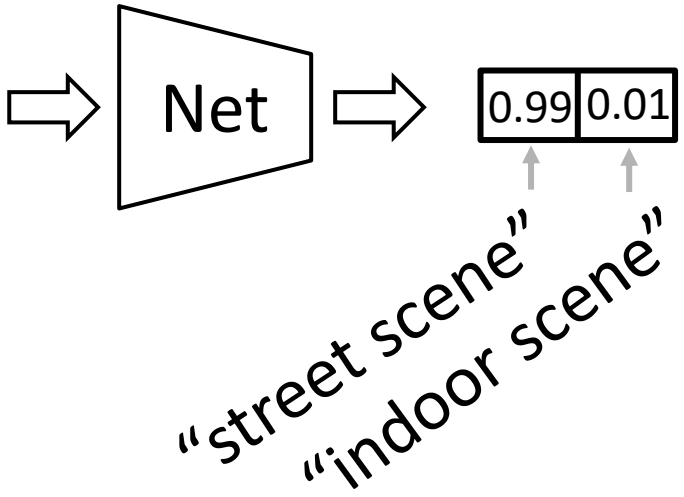




- A convolutional neural network (CNN)

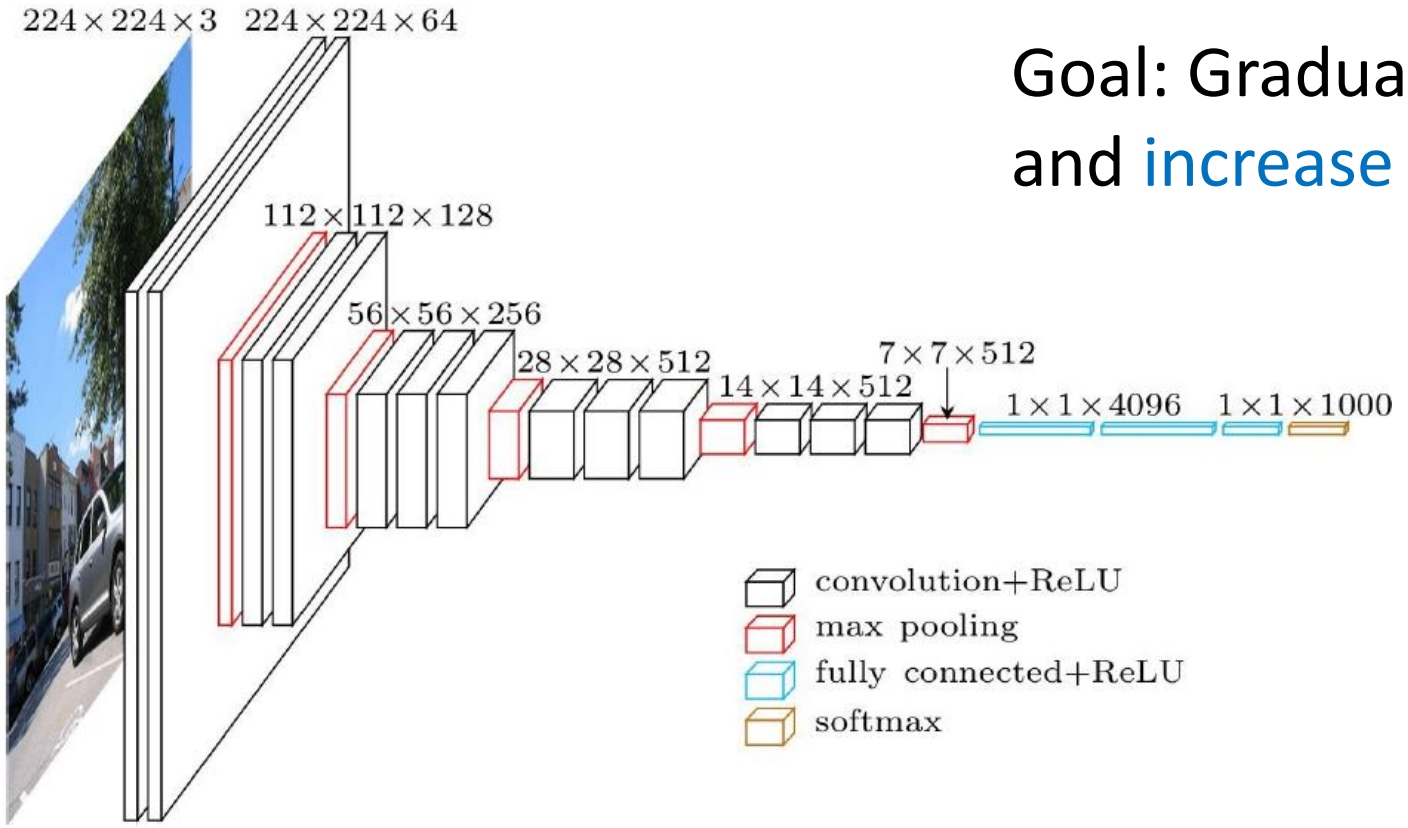


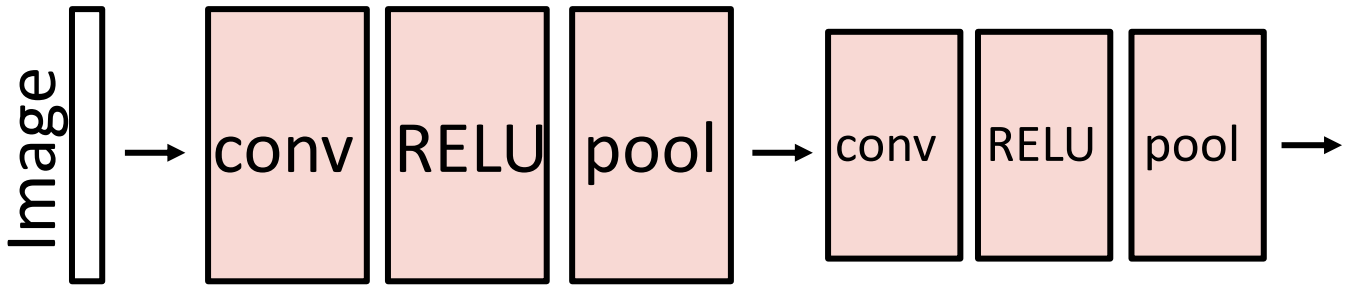
# Consider the image classification problem

# Convolutional neural networks




Goal: Gradually decrease spatial size (WxH) and increase the depth (feature dimension)




The basic building blocks:

- Convolutional layers
- Nonlinearity (RELU)
- Pooling layers

Simonyan and Zisserman: Very Deep Convolutional Networks for Large-Scale Image Recognition. ICLR, 2015

# Convolutional layer

Image
(e.g, 32x32x3)

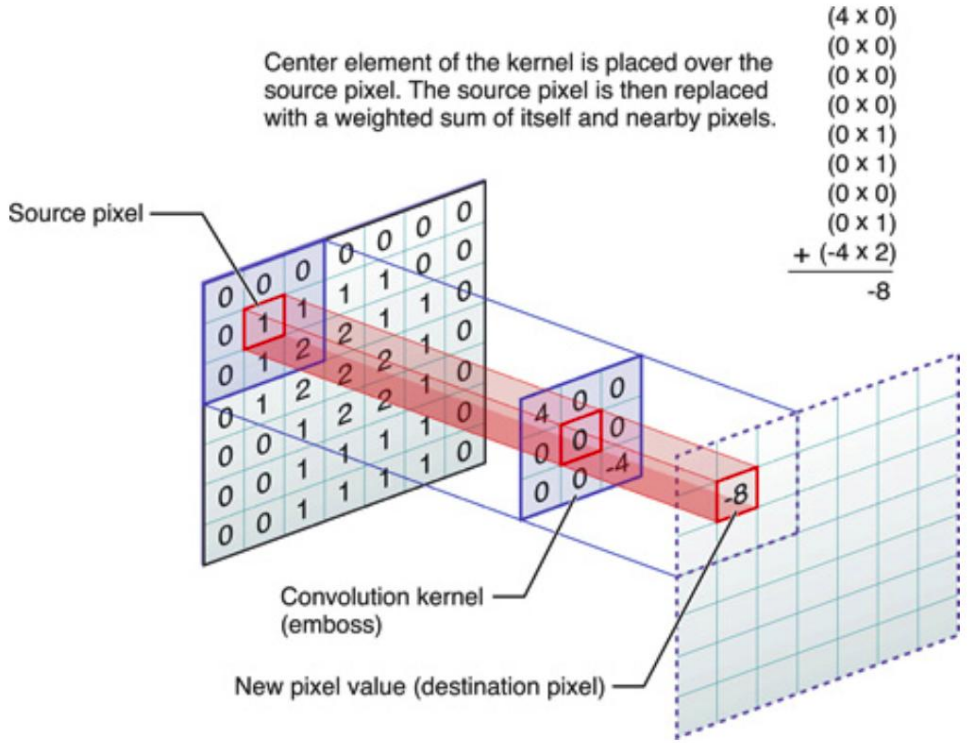
N filters
(e.g, 5x5x3)

...

Each filter
generates a 2D
output

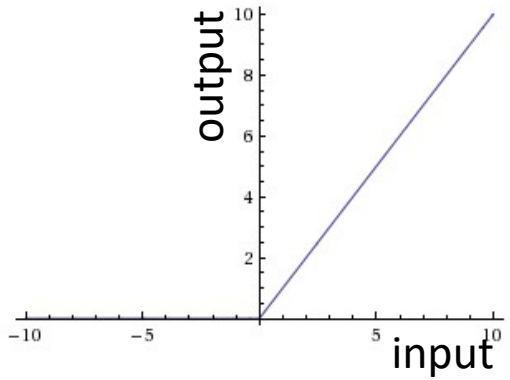
Convolve with
each filter

Output is a 32x32xN
tensor

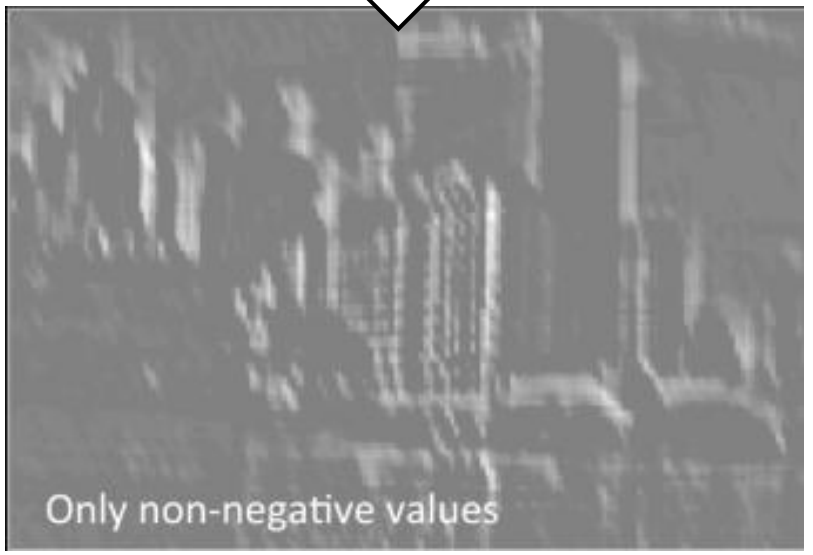


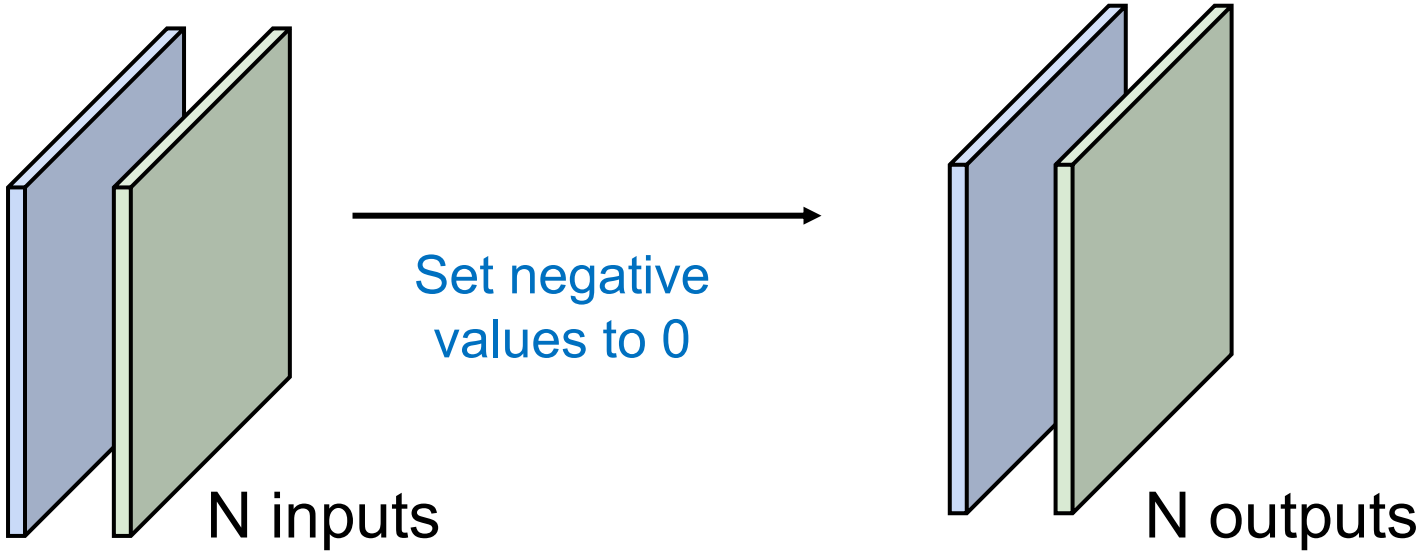
# Nonlinear layer (e.g., RELU)

- Rectified linear unit (RELU)
- Implement nonlinear feature transformations
- Specific form crucial for backpropagation to work!



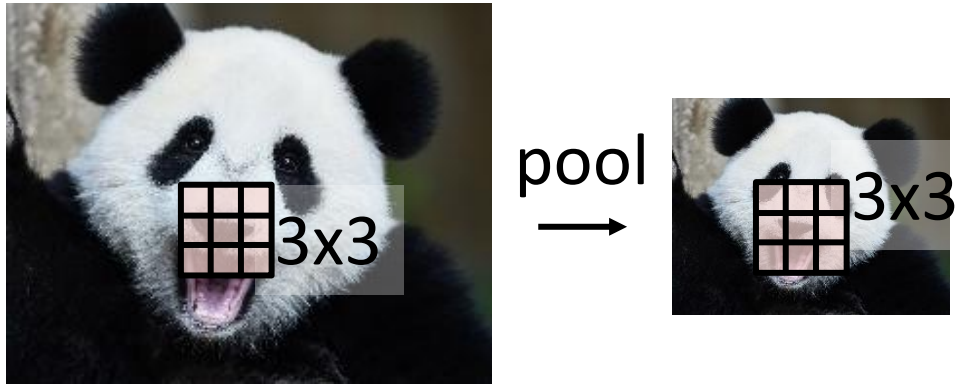


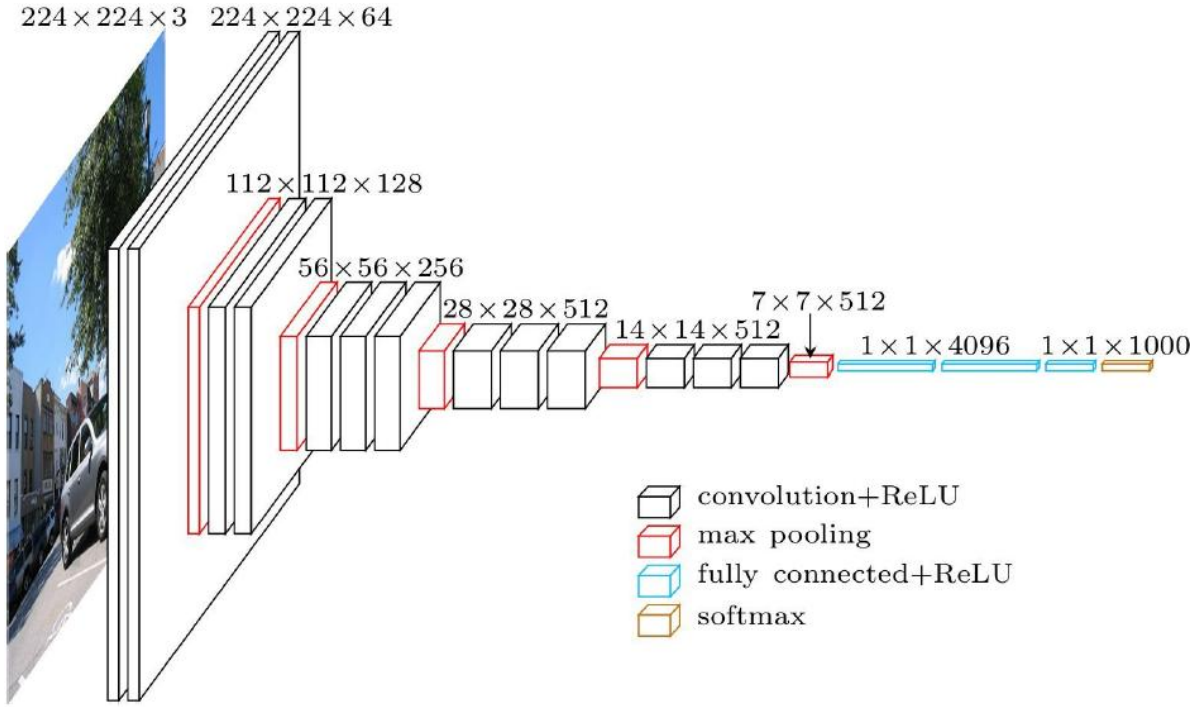


# Pooling layer

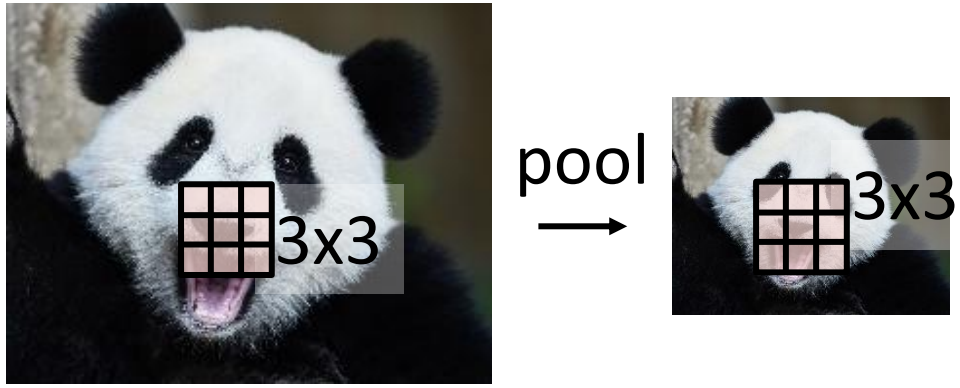
- Implements downsampling:
  - Reduce spatial resolution
  - Increase the receptive field (which pixels influence the neuron)



# Pooling layer

- A popular pooling operation: max pool

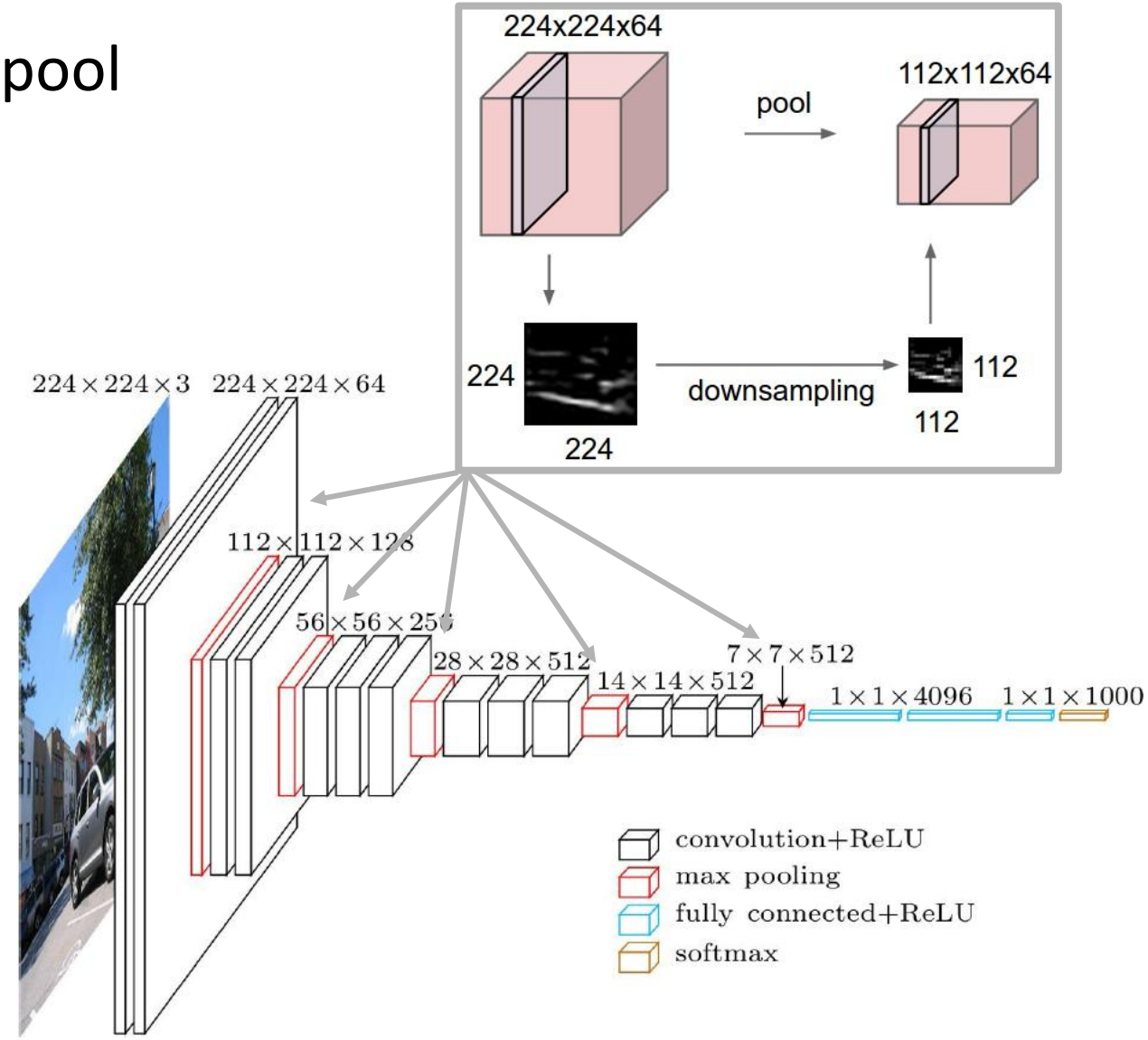



Single depth slice

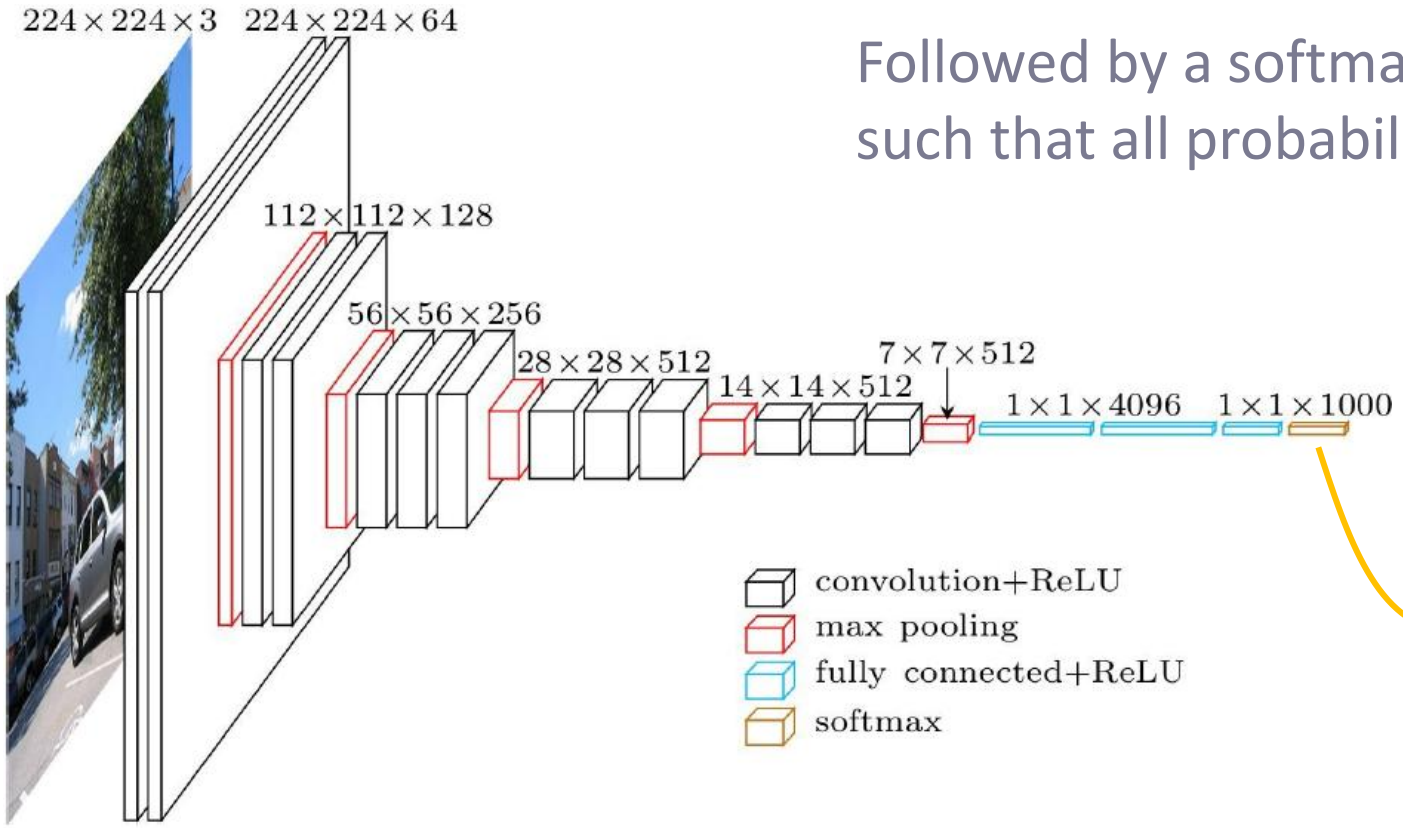
| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 4 |
| 5 | 6 | 7 | 8 |
| 3 | 2 | 1 | 0 |
| 1 | 2 | 3 | 4 |

max pool with 2x2 filters and stride 2

| | |
|---|---|
| 6 | 8 |
| 3 | 4 |



# Fully-connected layer




Followed by a softmax function, such that all probabilities sum to 1.

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$p_{cat}$ ∈[0,1] Cat

$p_{dog}$ ∈[0,1] Dog

$p_{cow}$ ∈[0,1] Cow

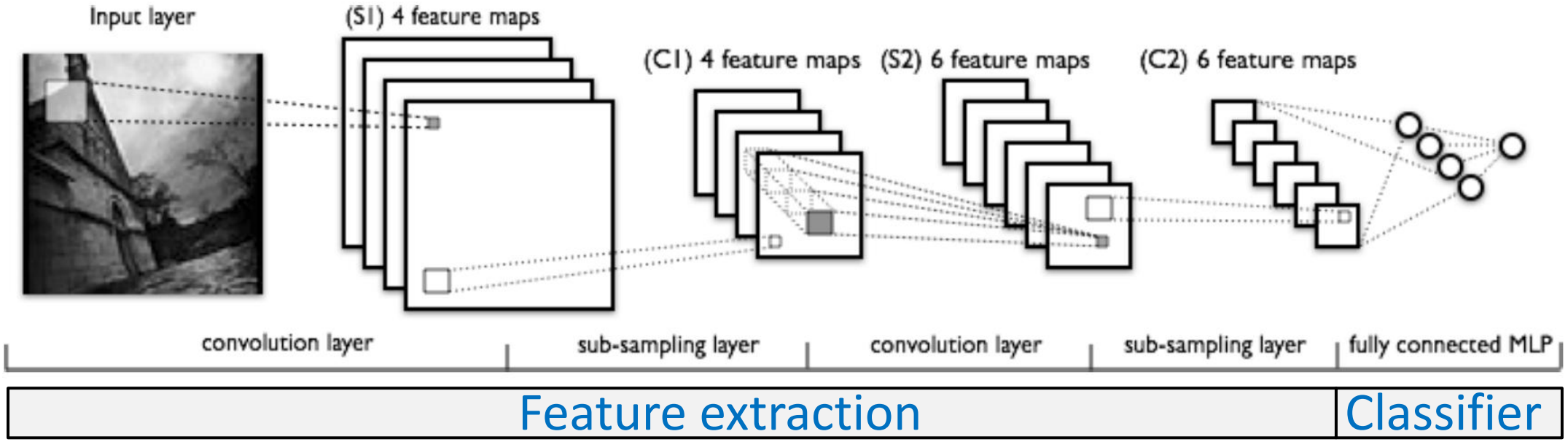
⋮ ⋮

$p_{racoon}$ ∈[0,1] Racoon

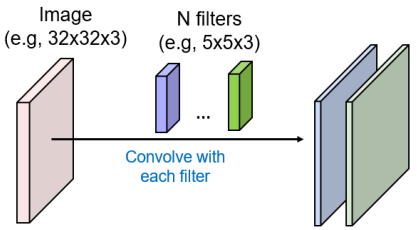
- Maps final 1x1x4096 feature vector into the final prediction one-hot vector of probabilities (in the above example 1000 output categories). Heavy on parameters...

# A conceptual CNN architecture

- Architecture contains feature extraction as well as a classifier
- Learning means:
  - Learn feature extraction (convolution filter kernels)
  - Learn a classifier (e.g., a multi-layer perceptron)



# CNNs attract a significant attention in 2012




- The filters and biases in CNN are the parameters to be learned.
- The breakthrough came with the AlexNet (50-60 million parameters)

[1]Krizhevsky et al., ImageNet Classification with Deep Convolutional Neural Networks, NIPS2012

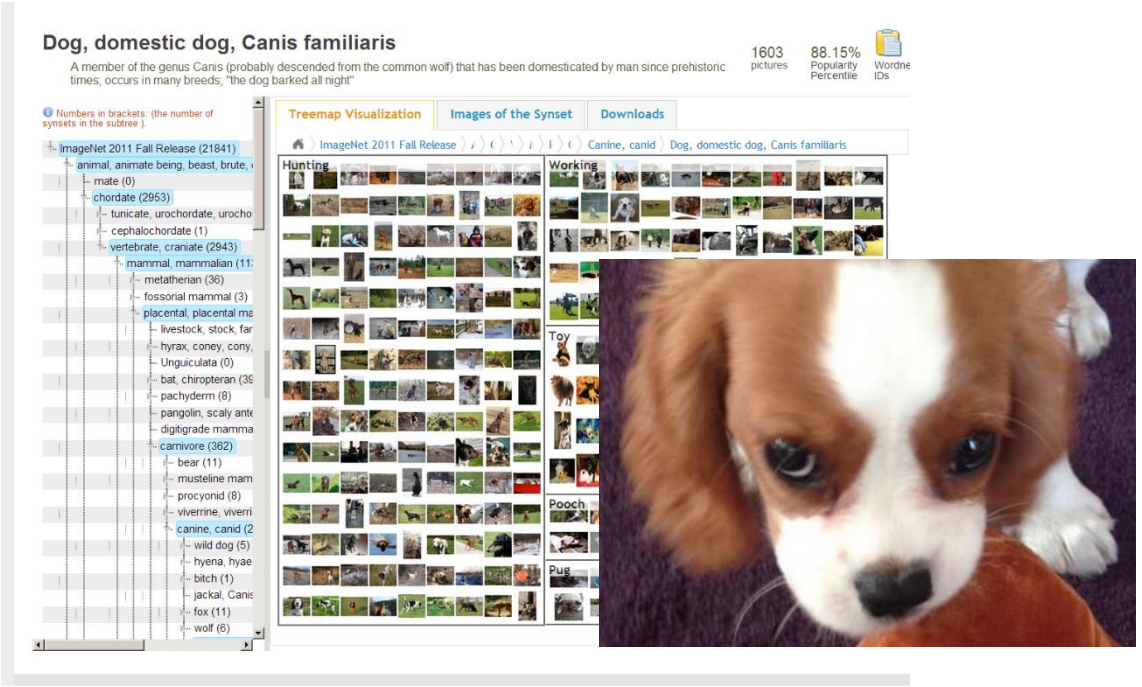
- Became possible due to HUGE labelled datasets (ImageNet )

14 million labeled images, 20K categories

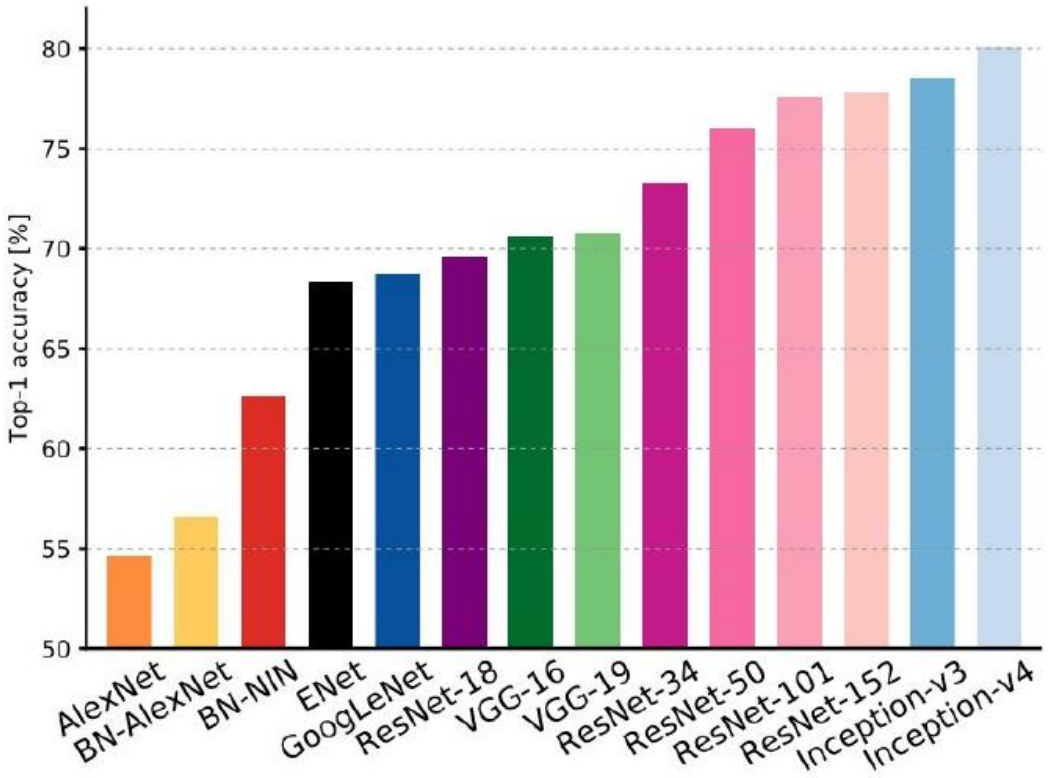
http://www.image-net.org/

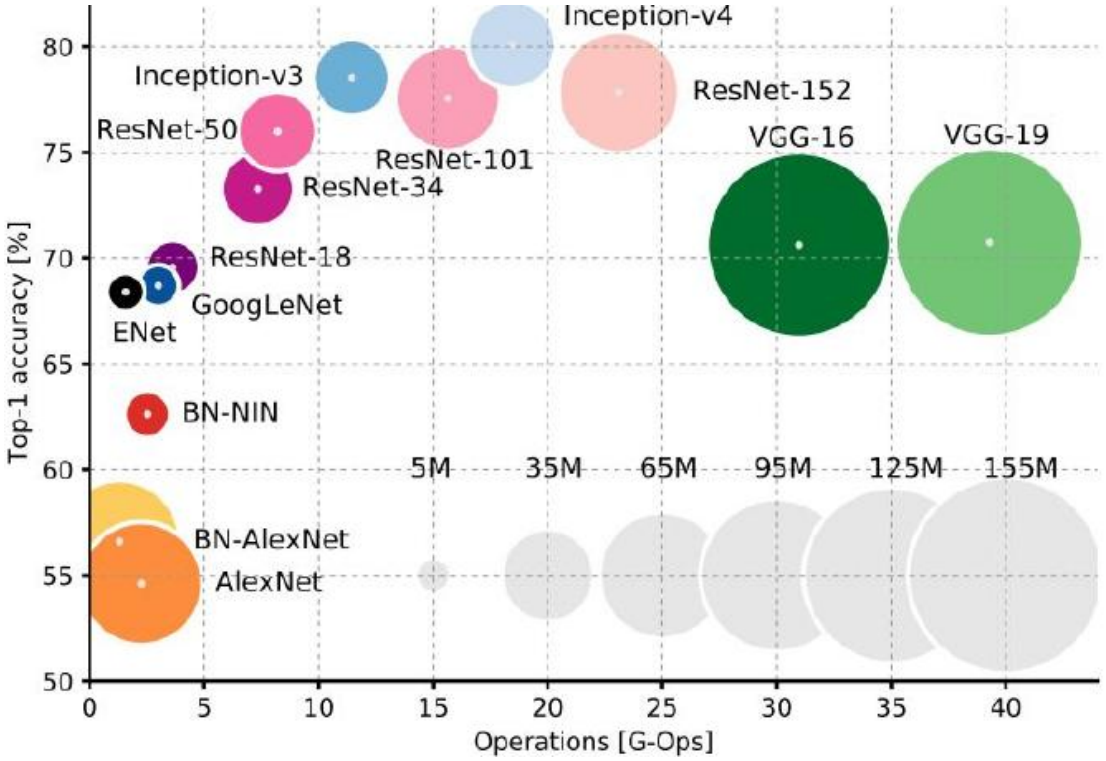
| Model | Top-1 | Top-5 |
|---|---|---|
| *Sparse coding [2]* | *47.1%* | *28.2%* |
| *SIFT + FVs [24]* | *45.7%* | *25.7%* |
| CNN | **37.5%** | **17.0%** |

Huge advances made in subsequent the years...



# CNN architectures

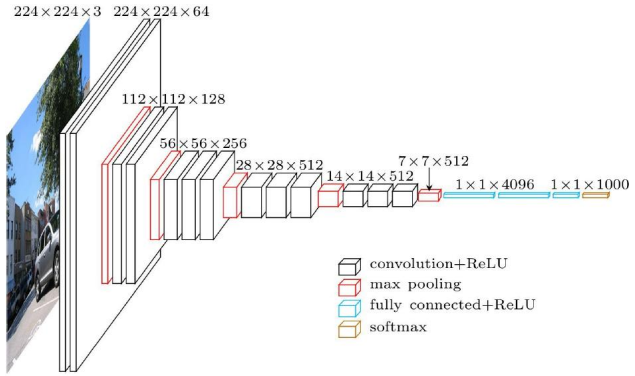



- Speed approximately inversely proportional with the number of parameters
  - Crucial aspect for embedded-ready architectures
- Accuracy (somewhat) related to the network complexity

# From image-wide to pixel-level classification/labelling

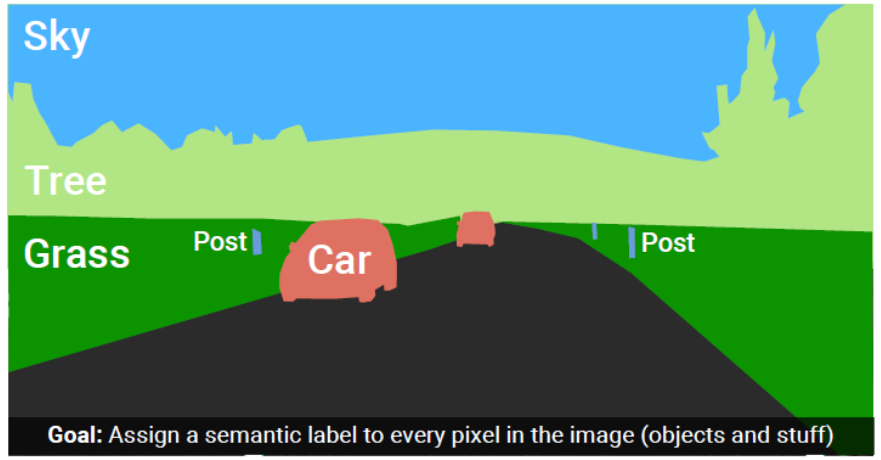
- So far we considered classifying the entire image


-> “Street scene”

- But many applications require more fine-grained classification

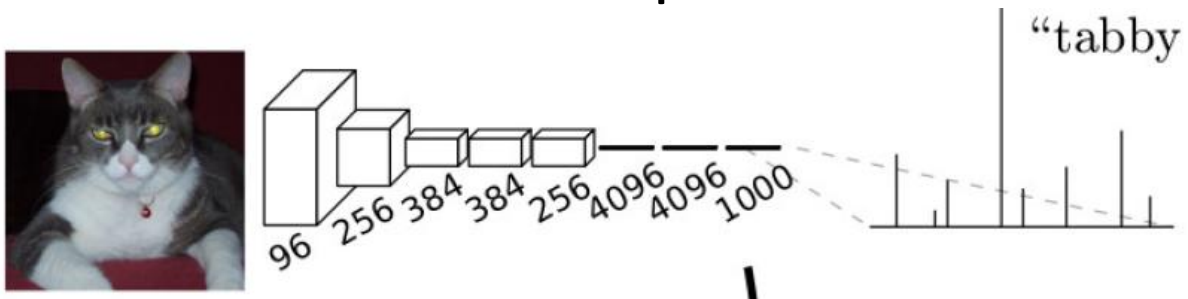

Assign a class label to each pixel



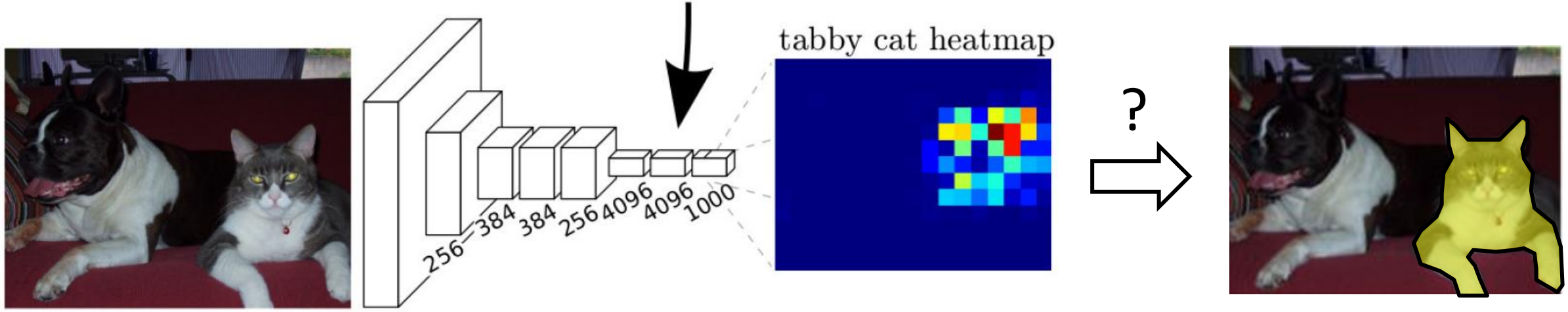
# Semantic segmentation

Long, Shelhamer and Darrell: Fully convolutional networks for semantic segmentation. CVPR, 2015

- Apply CNNs to encode each pixel



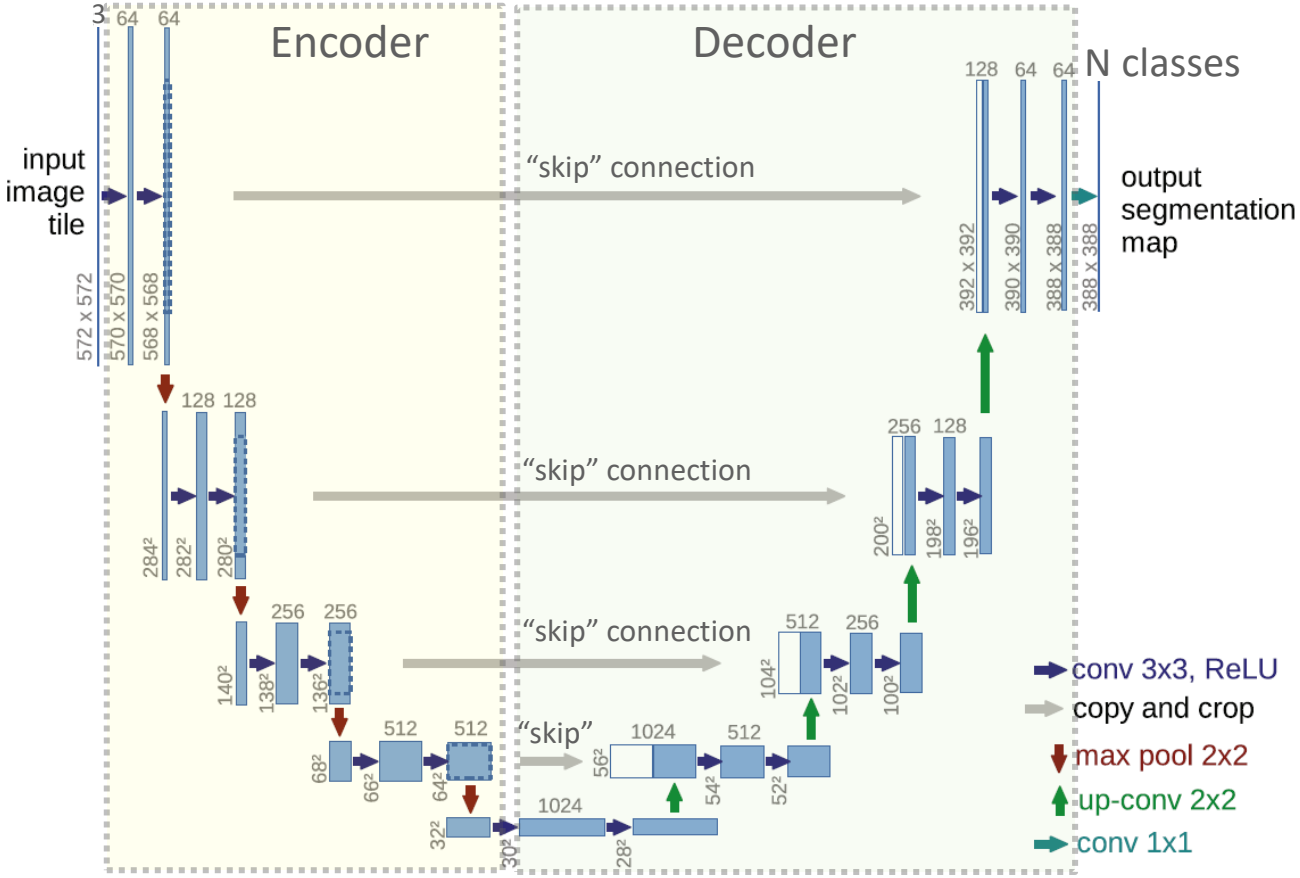

Issue:
Resolution lost as
the encoding becomes deeper

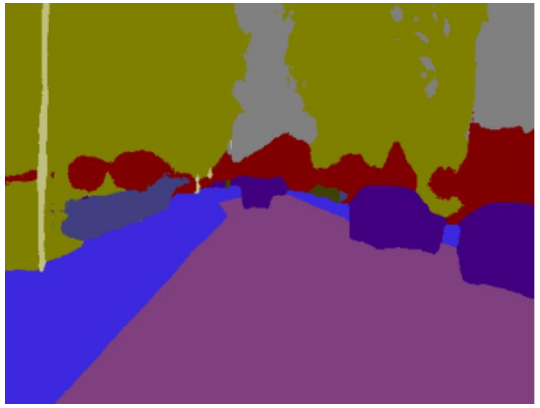



- Encode the pixels by applying a few conv-relu-pool blocks
- Add a final 1x1xN filter to predict the one-hot vector for each pixel

# Semantic segmentation: Unet

- A general approach: apply a “decoder” to gradually upsample the features.



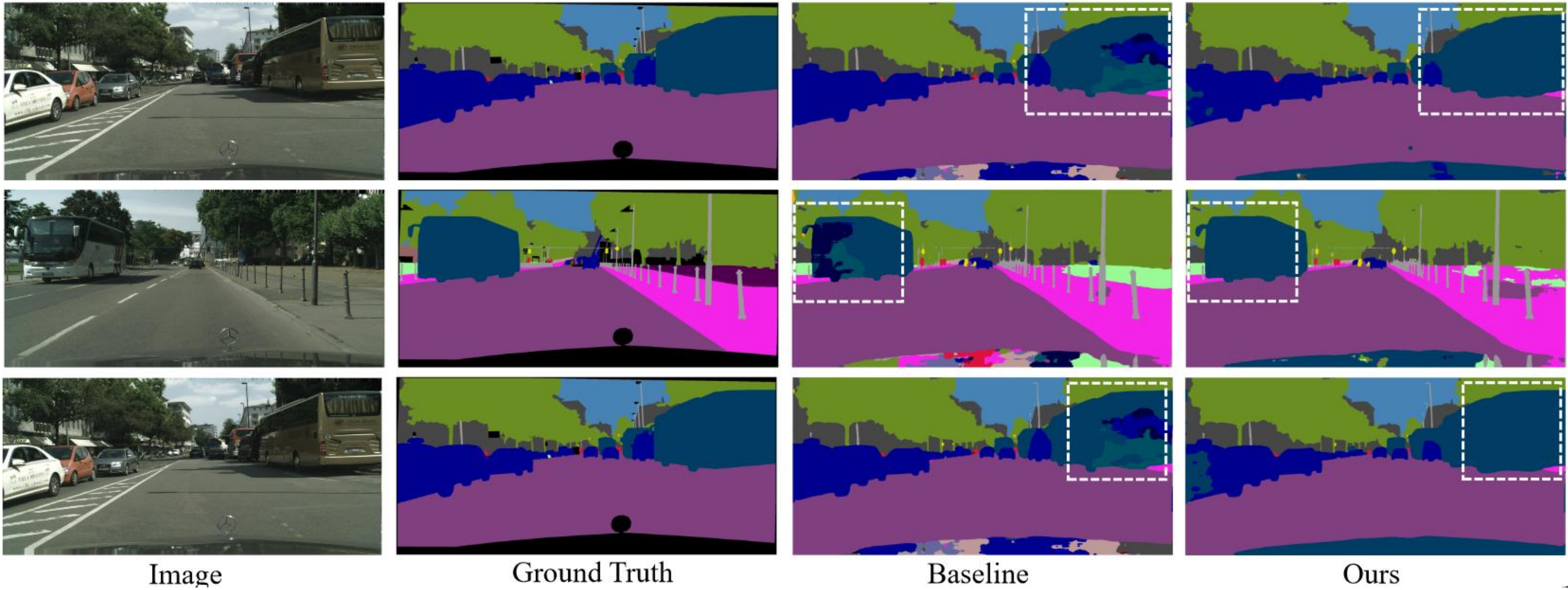



- Skip connections: *attach encoder features to the upsampled decoder features*

>126k citations!

Ronnenberg et al., U-Net: Convolutional Networks for Biomedical Image Segmentation, MICCAI2015

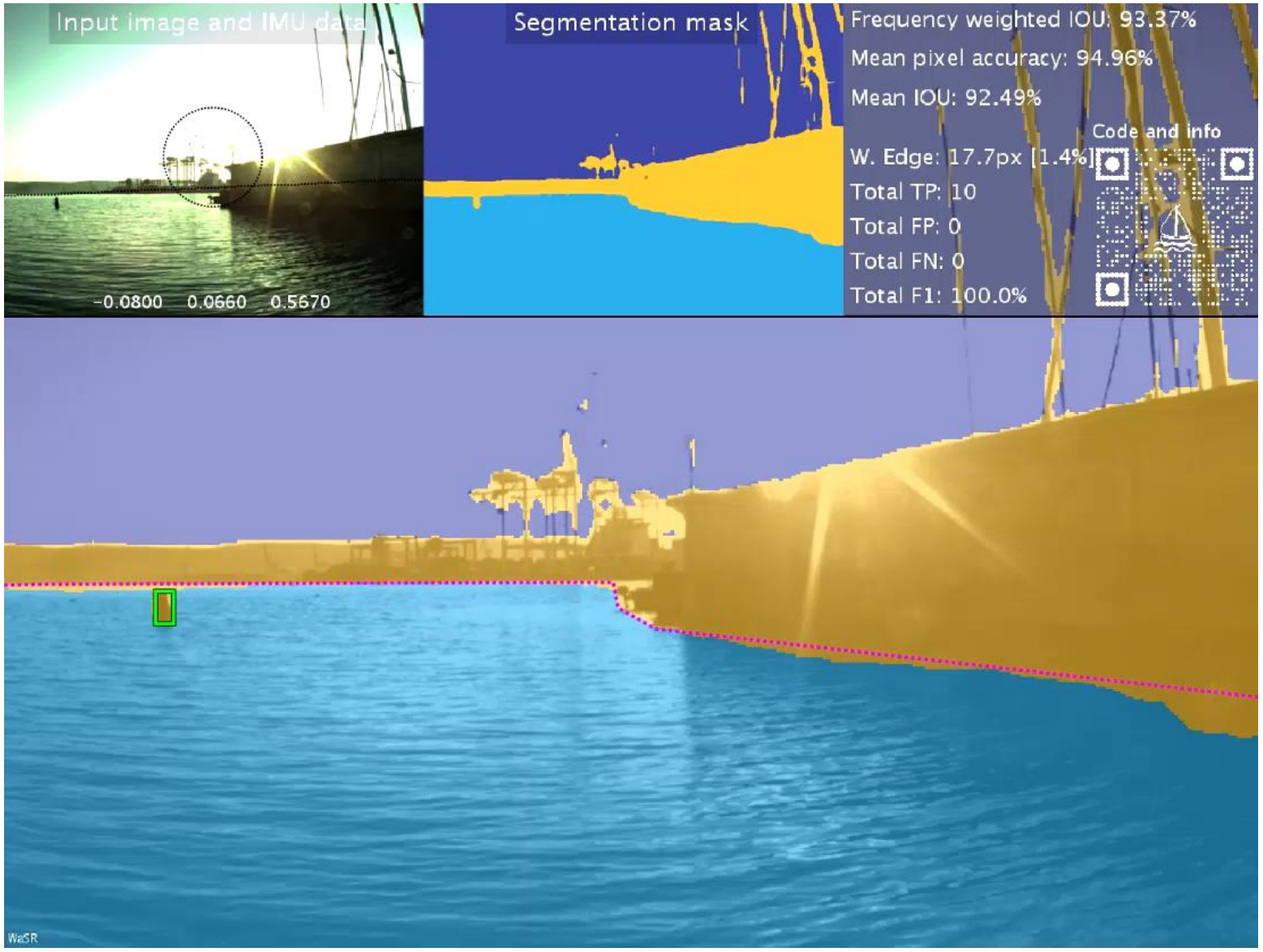
# Autonomous cars




Top performers on the major autonomous cars benchmark Cityscapes in 2019.

Yuhui Yuan, Xilin Chen, Jingdong Wang, Object-Contextual Representations for Semantic Segmentation, Arxiv 2019

# Autonomous boats




Bovcon, Kristan, WaSR -- A Water Segmentation and Refinement Maritime Obstacle Detection Network, TCyb 2021

# Object detection

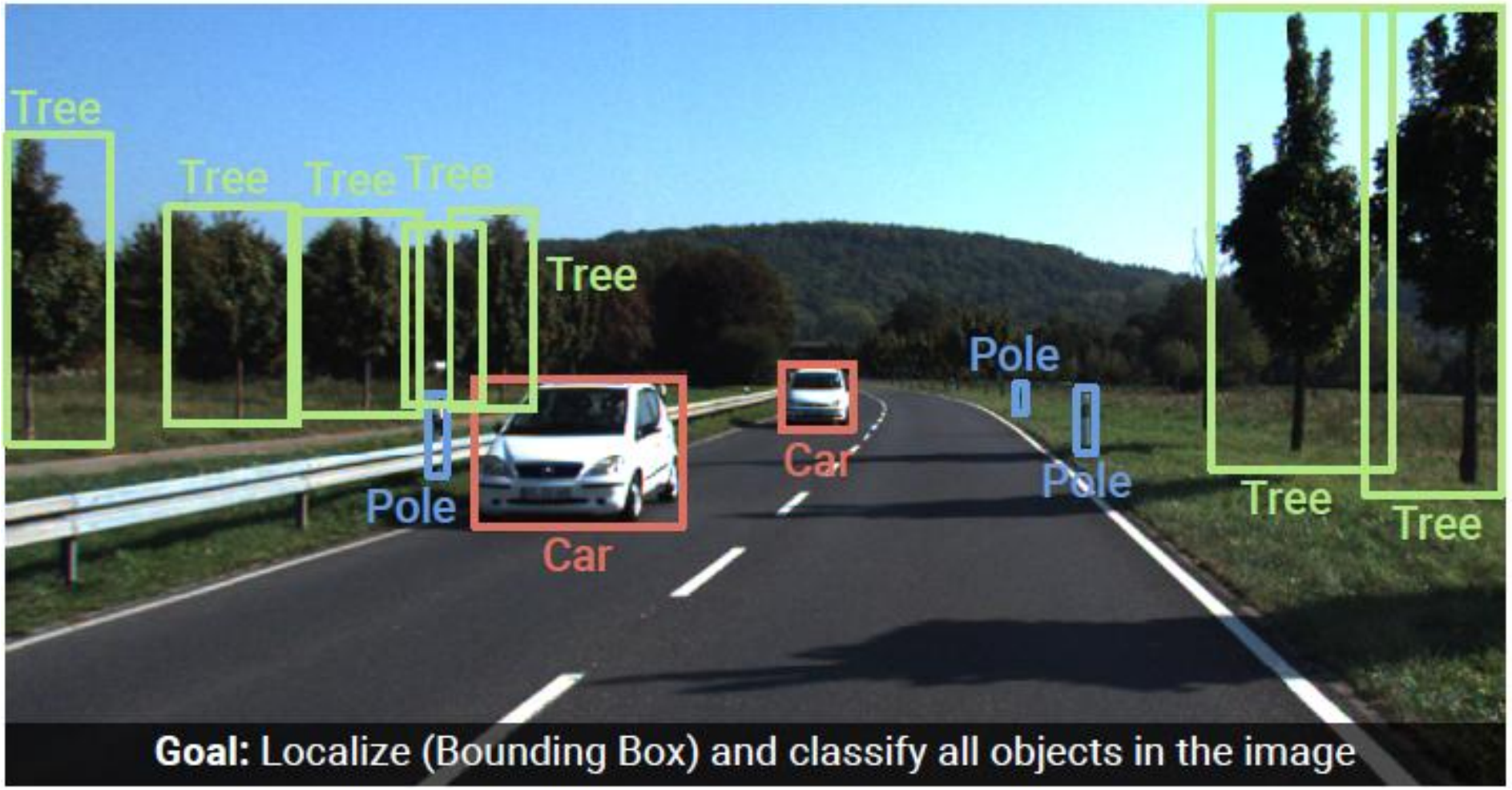



Image curtesy: Andreas Geiger

# Object detection by R-CNN

Per-image computation

Per-region computation for each $r_i \in r(I)$

$f_I = I$

$I$:

Selective search, Edge Boxes, MCG, …

1

Crop & warp

2

ConvNet($r_i$)

3

Linear classifier

4

Box regressor

5

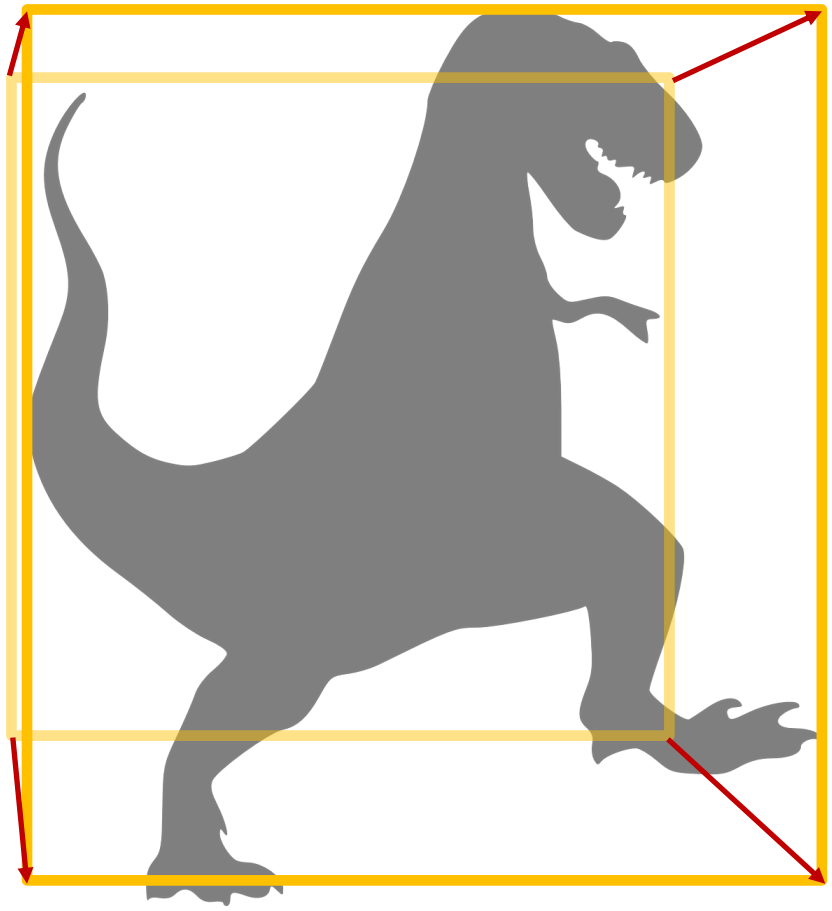
Girshick et. al, Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation, CVPR 2014

Slide credit: Ross Girshick

# Box regression

- Region proposal generates approximate bounding box
- The box regressor refines it

P(object) = 0.94

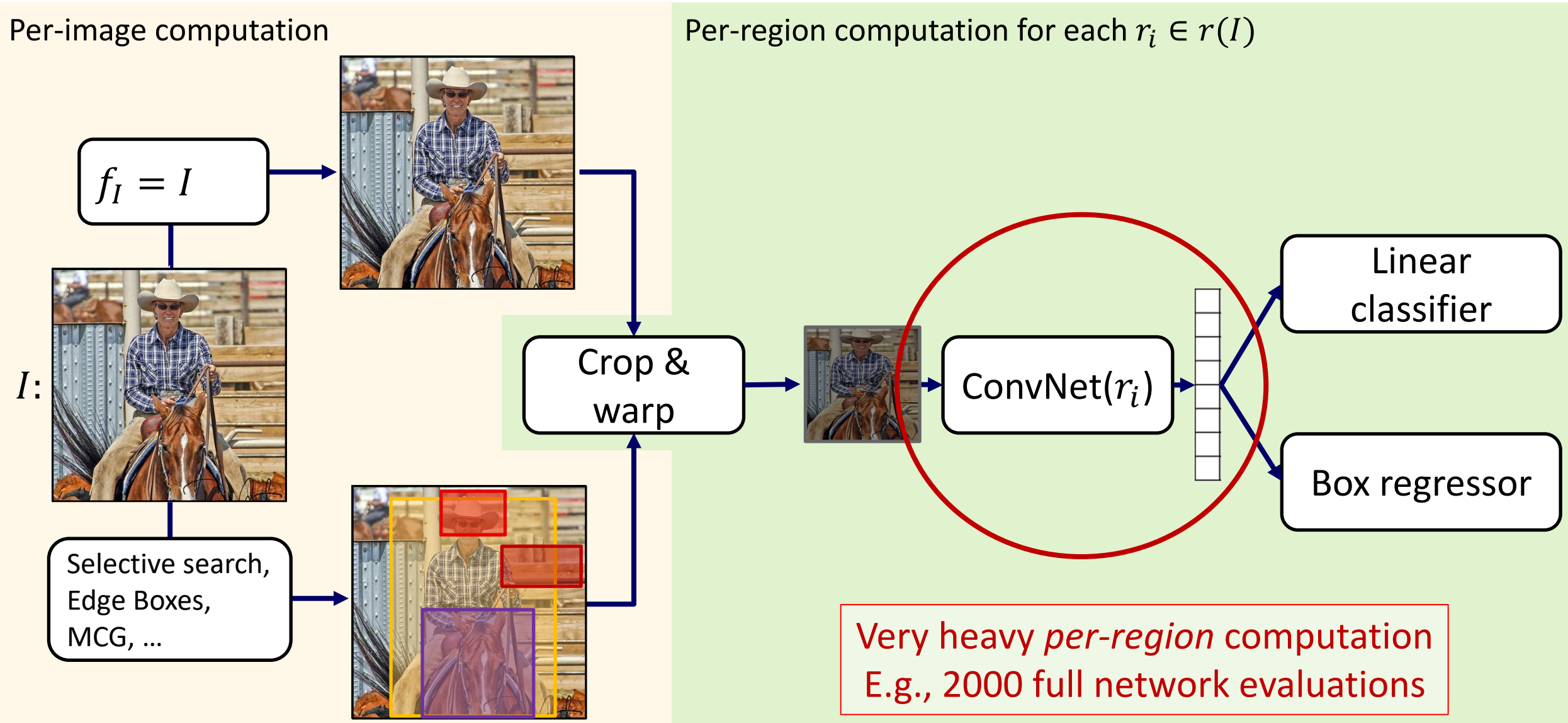


Anchor box:
transformed by
box regressor

# "Slow" R-CNN

Per-image computation

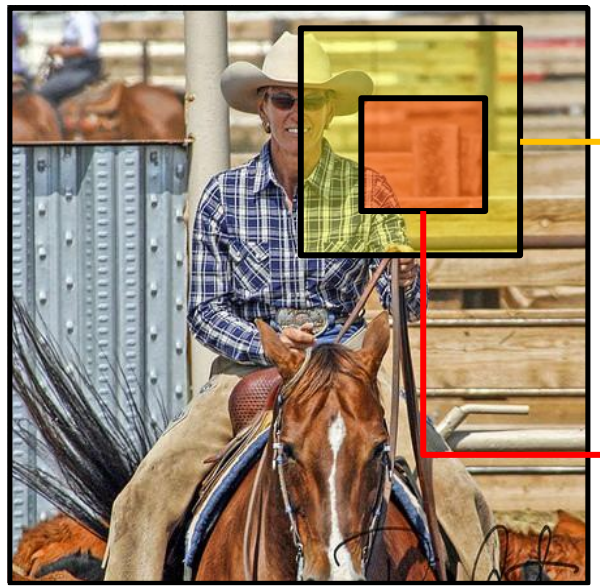
Per-region computation for each $r_i \in r(I)$




Very heavy *per-region* computation
E.g., 2000 full network evaluations

Slide credit: Ross Girshick

# Why is it slow?



Network Θ

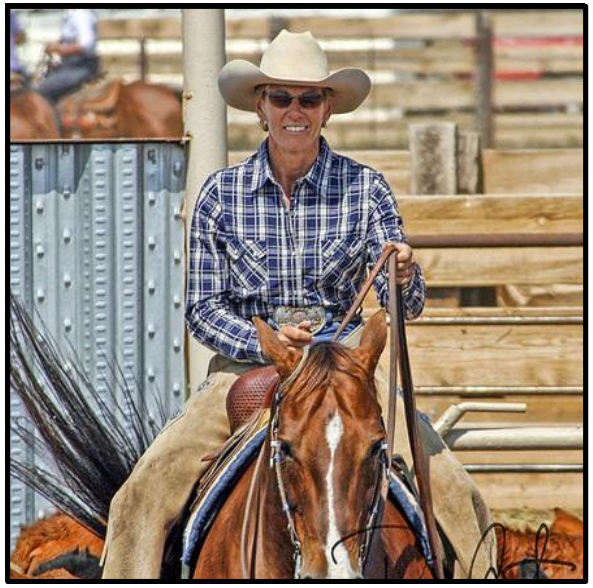
Features

Process (classify)

Network Θ

Features

Process (classify)



Network Θ

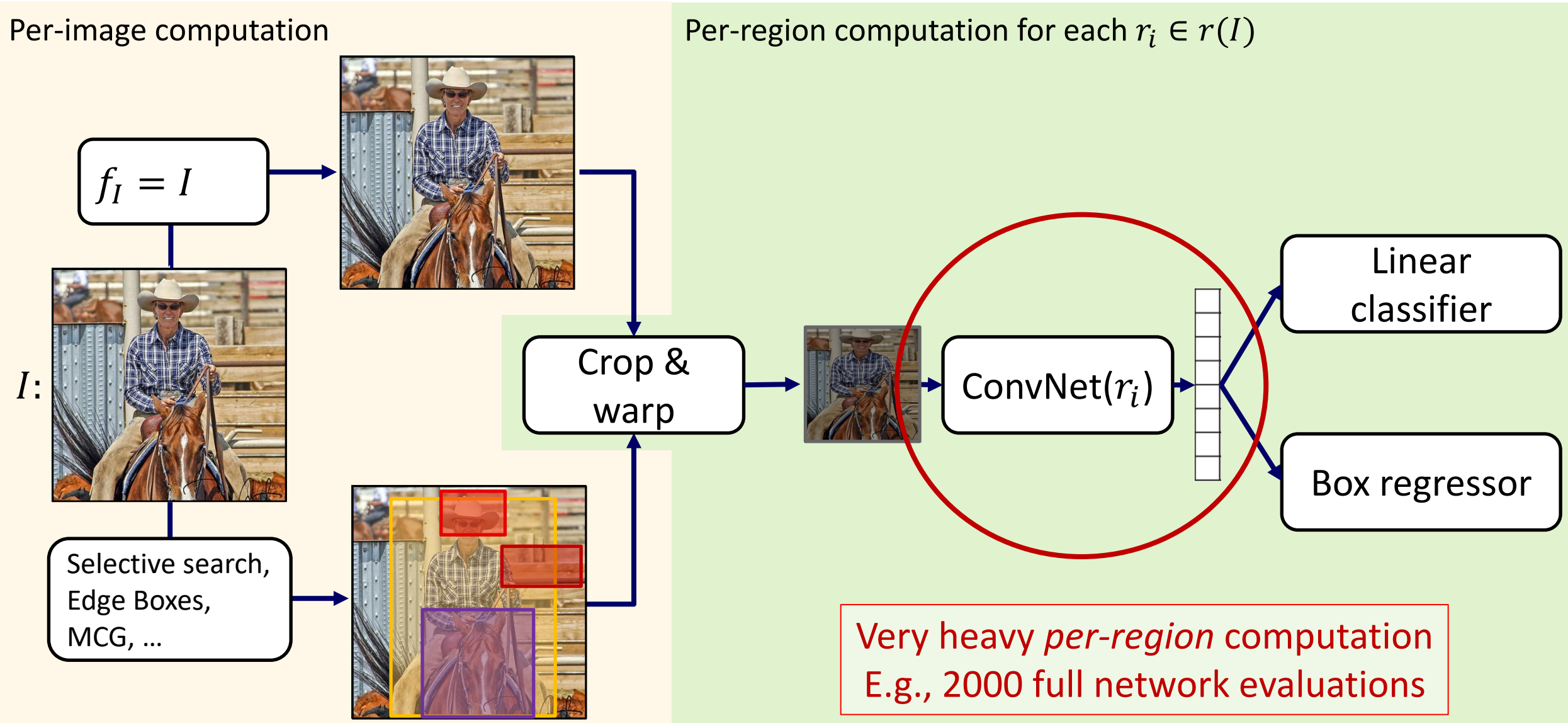
Features

Process (classify)

# "Slow" R-CNN

Per-image computation

Per-region computation for each $r_i \in r(I)$




Very heavy *per-region* computation
E.g., 2000 full network evaluations

Slide credit: Ross Girshick

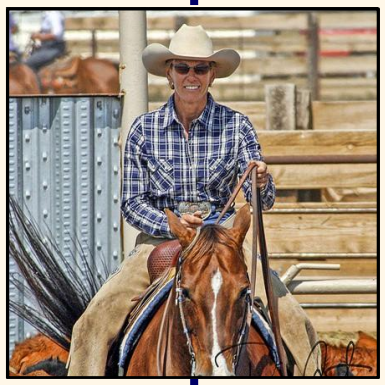
# Generalized R-CNN → Fast R-CNN

Per-image computation

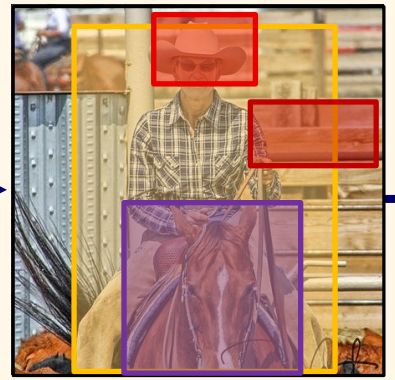
A few CNN layers to extract features

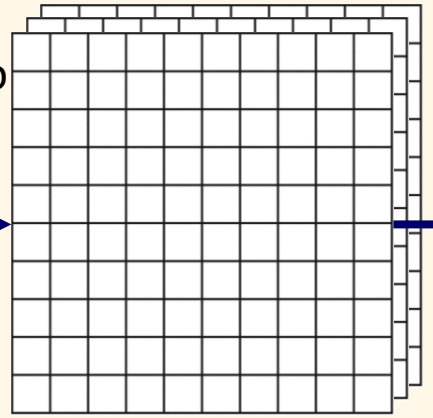
1 FCN($I$)

$I$:



Selective search, Edge Boxes, MCG, …

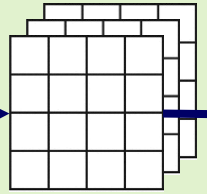




Per-region computation for each $r_i \in r(I)$

*RoIPool: Interpolates extracted features within a proposed region and resizes to a predefined resolution, which is fixed for the input of the multi-layer perceptron (MLP)*

2 RoIPool



3 MLP

Softmax clf.

4

Box regressor

A classification head and box regression head are finally applied

Girshick. Fast R-CNN. ICCV 2015.

Slide credit: Ross Girshick

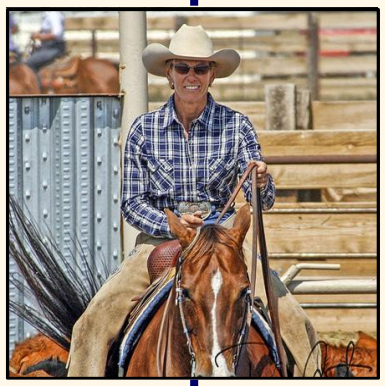
# The Problem with Fast R-CNN

Per-image computation

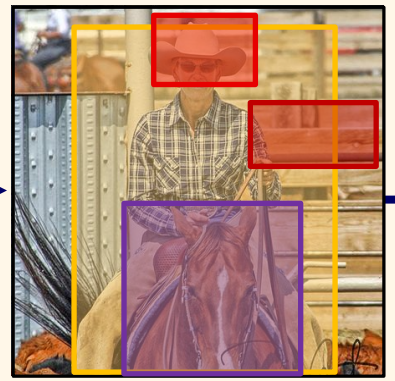
A few CNN layers to extract features

FCN($I$)

$I$:



Selective search, Edge Boxes, MCG, …



Per-region computation for each $r_i \in r(I)$

RoIPool

MLP

Softmax clf.

Box regressor

Region proposals have very poor recall
(ok for PASCAL VOC, major bottleneck for COCO)
Also, they can be slow

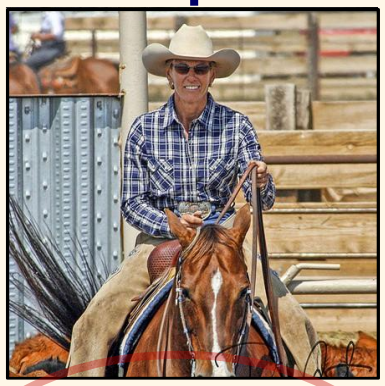
# Faster R-CNN

Per-image computation

A few CNN layers to extract features

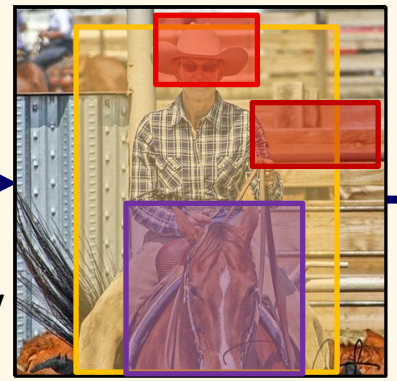
$f_I$ = FCN($I$)

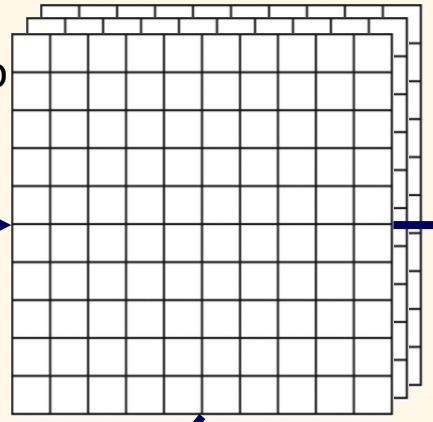
$I$:



RPN($f_I$)

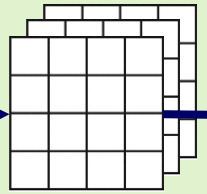
Region proposal network (RPN) generates bboxes by regression from CNN features





Per-region computation for each $r_i \in r(I)$

RoIPool



MLP

Softmax clf.

Box regressor

Learned proposals

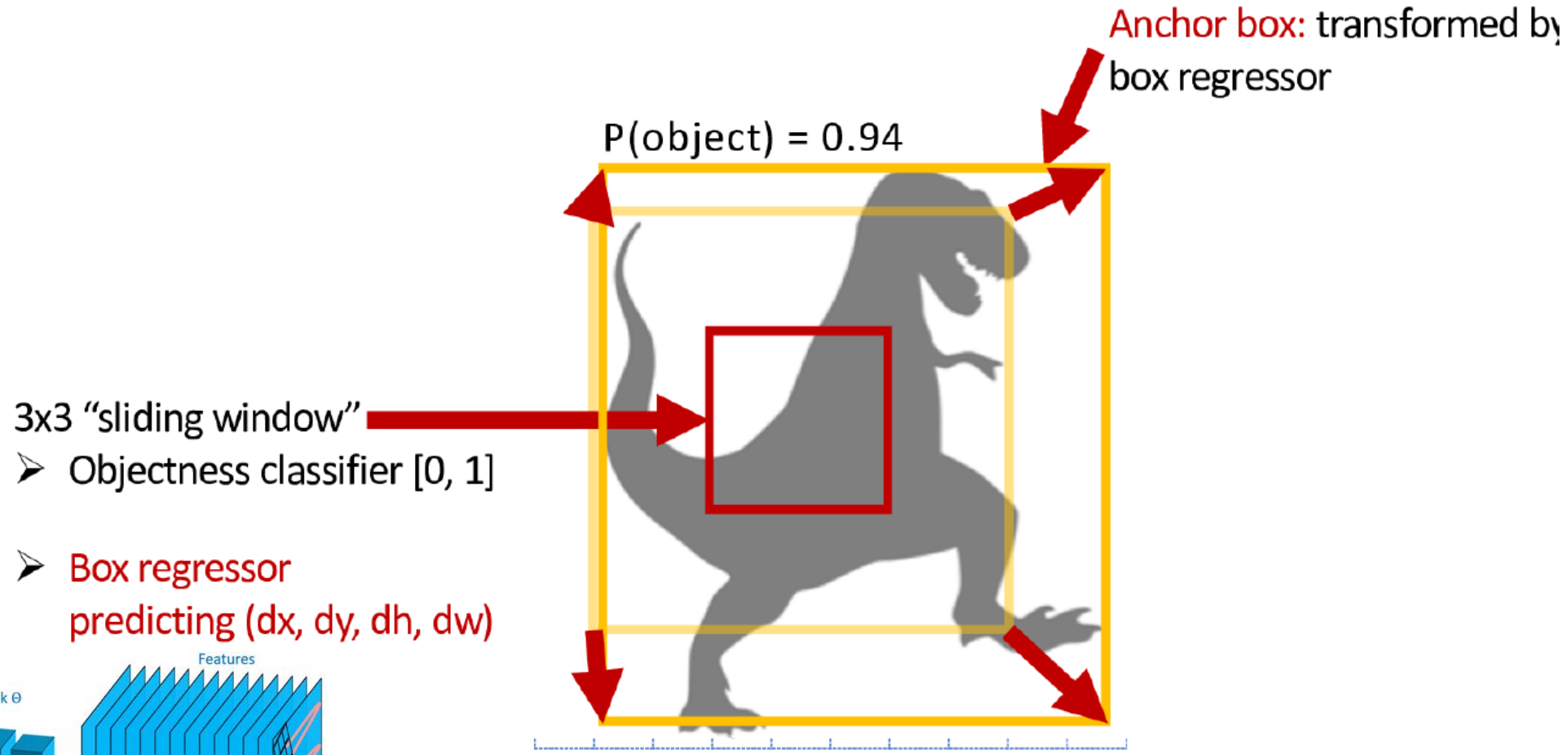
*Shares computation* with whole-image network

Ren, He, Girshick, Sun. Faster R-CNN: Towards Real-Time Object Detection. NIPS 2015.

Slide credit: Ross Girshick

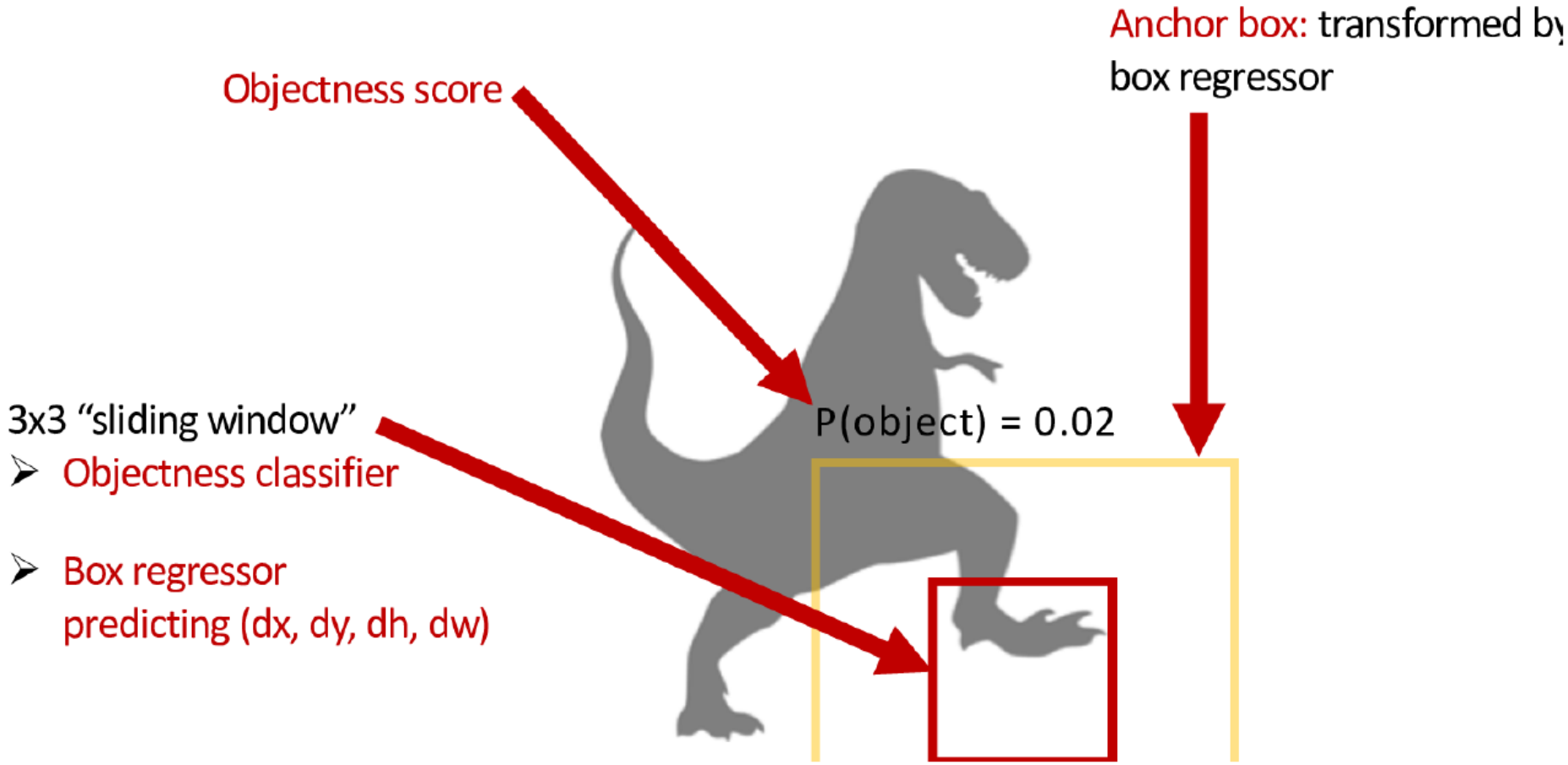
# Region proposal network (RPN)

- Generate object proposals by so-called anchor boxes



# Region proposal network (RPN)

- Low objectness score example



# Faster R-CNN with a Feature Pyramid Network

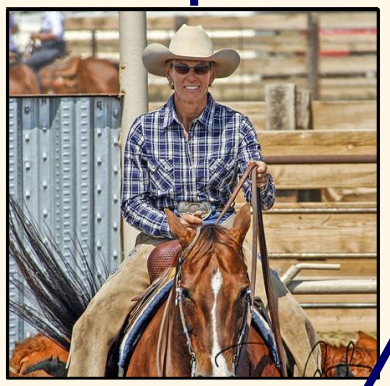
Per-image computation

Per-region computation for each $r_i \in r(I)$

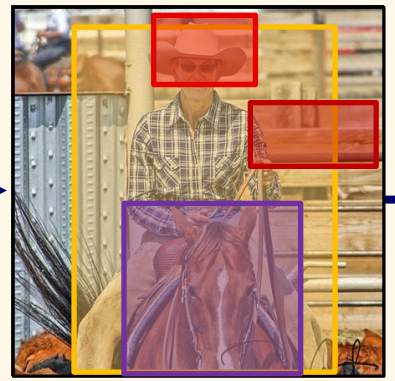
Compute pyramid CNN features

$f_I$ = FPN($I$)

$I$:



RPN($f_I$)



RoIPool

MLP

Softmax clf.

Box regressor

The whole-image feature representation can be improved by making it *multi-scale*

Lin et al. Feature Pyramid Networks for Object Detection. CVPR 2017.

Slide credit: Ross Girshick

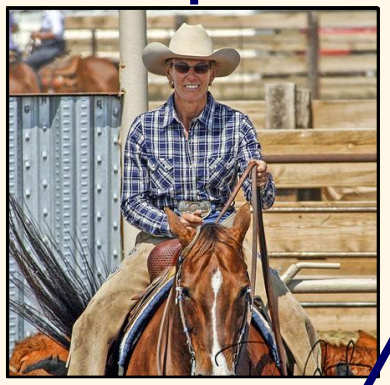
# Mask R-CNN

Per-image computation

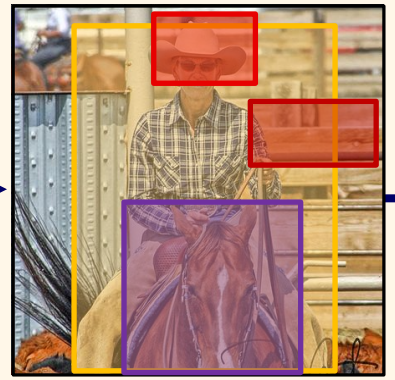
Per-region computation for each $r_i \in r(I)$

$f_I$ = FPN($I$)

$I$:



RPN($f_I$)



RoIAlign

MLP

Softmax clf.

Box regressor

Cascaded heads (inference only)

FCN

Masks

**Training the mask head**

He et al. Mask R-CNN. ICCV 2017.

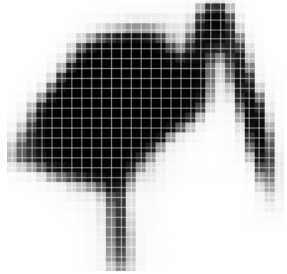
Slide credit: Ross Girshick

# Mask predictor

- Per-pixel occupancy map is predicted at the regressed bounding box

Validation image with box detection shown in red

28x28 soft prediction



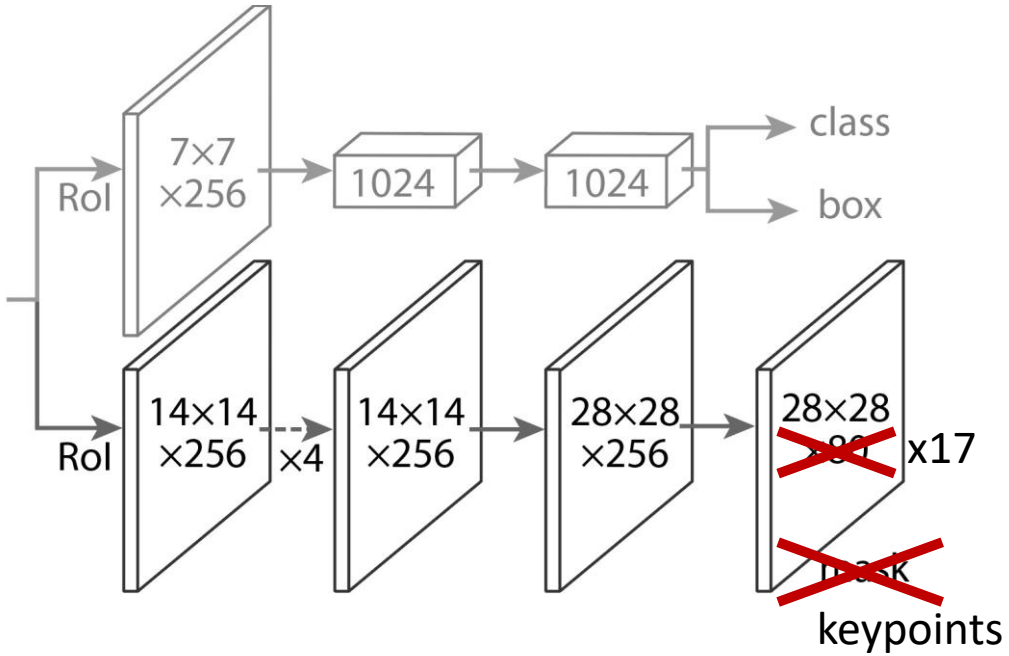
Resized Soft prediction

Final mask

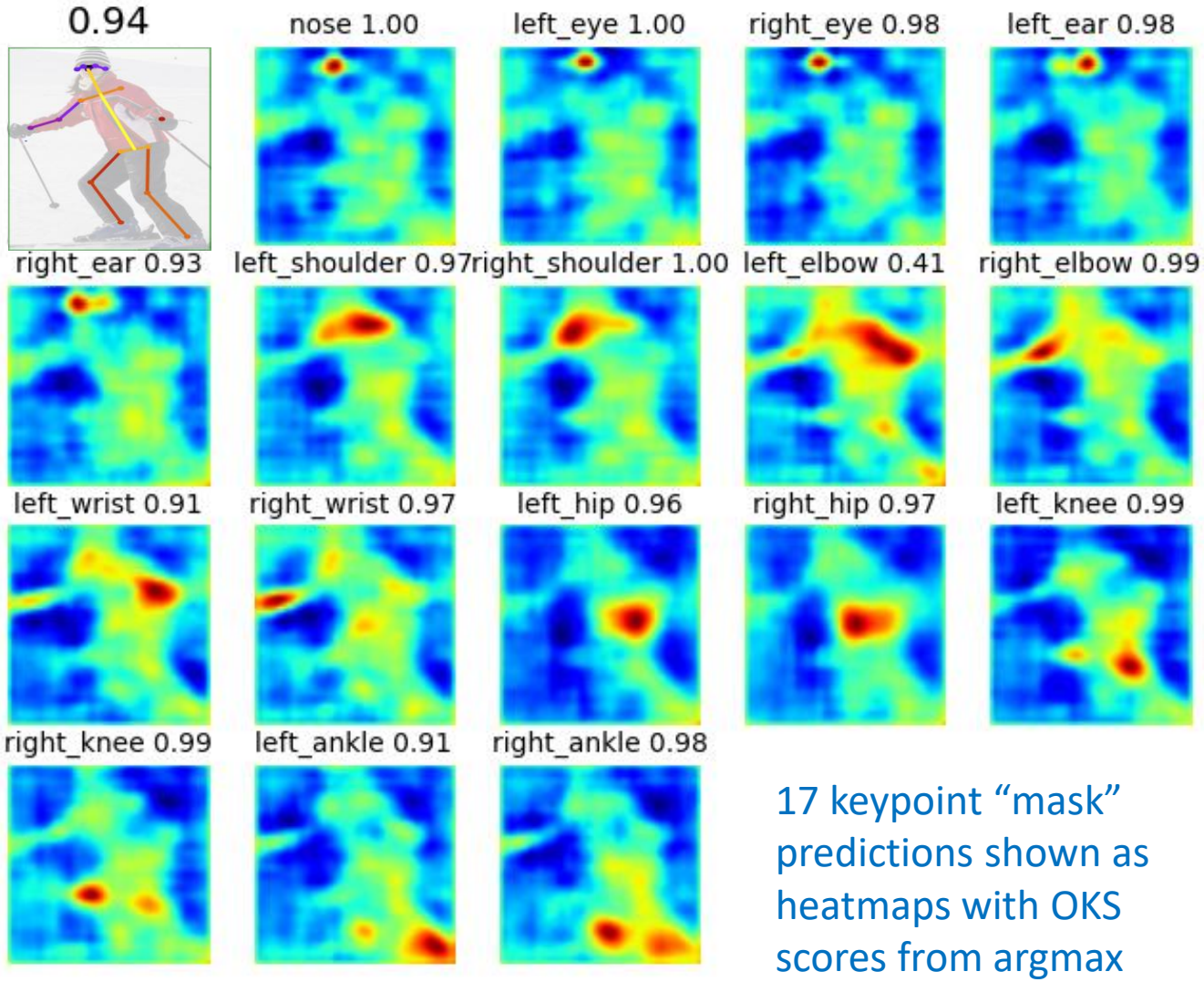
# Mask R-CNN application


Slide credit: Ross Girshick

# Human pose estimation




(Not shown: Head architecture is slightly different for keypoints)




- Add keypoint head (28x28x17)
- Predict one “mask” for each keypoint
- Softmax over spatial locations (encodes one keypoint per mask “prior”)

17 keypoint “mask” predictions shown as heatmaps with OKS scores from argmax positions

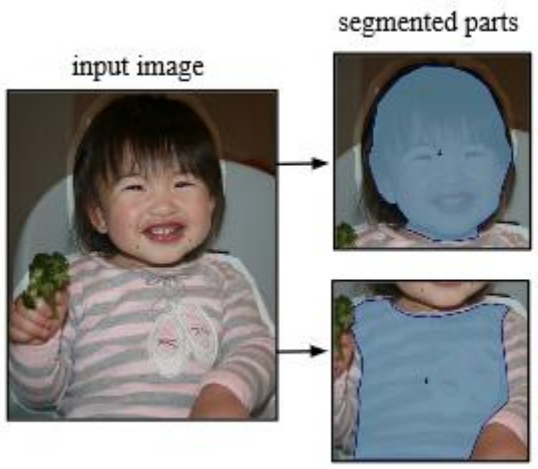
Slide credit: Ross Girshick

# Human pose estimation

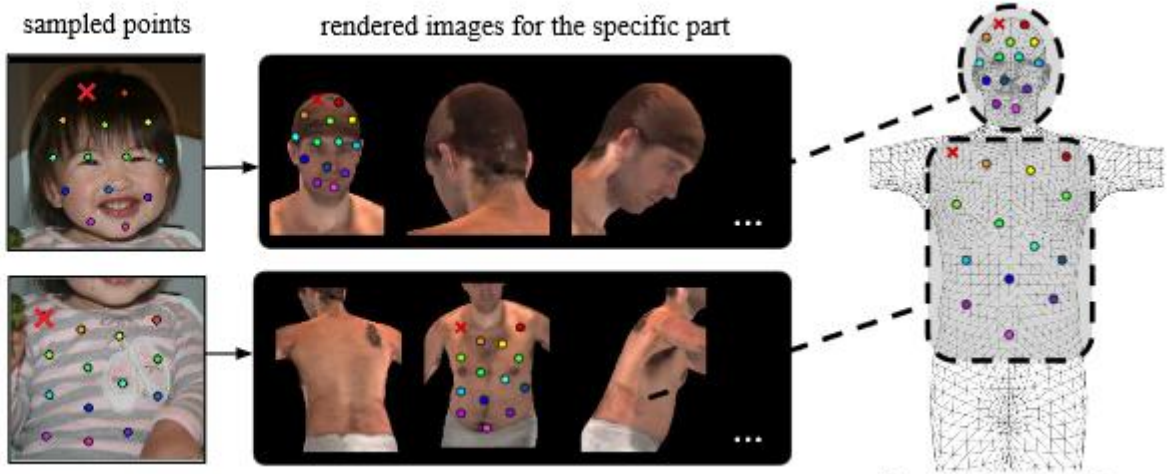
# Human “surface” estimation

Huge effort made to come up with manual ground truth annotations!

**50K humans**, over **5 million** manually annotated correspondences.




TASK 1: Part Segmentation

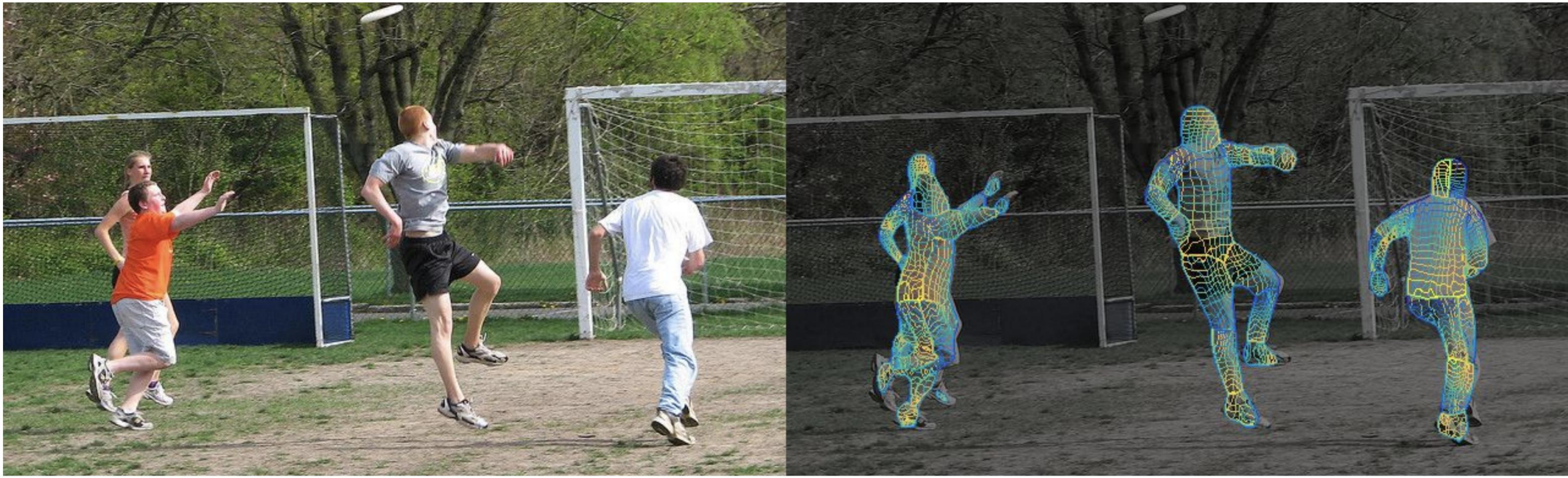



TASK 2: Marking Correspondences



Surface Correspondence

# Human “surface” estimation



The CVPR2018 oral available here: https://www.youtube.com/watch?v=Dhkd_bAwwMc

http://densepose.org/

Güler, Neverova, Kokkinos, **DensePose: Dense Human Pose Estimation In The Wild,** CVPR 2018

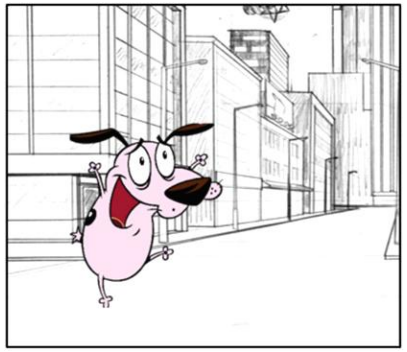
# Toolbox available

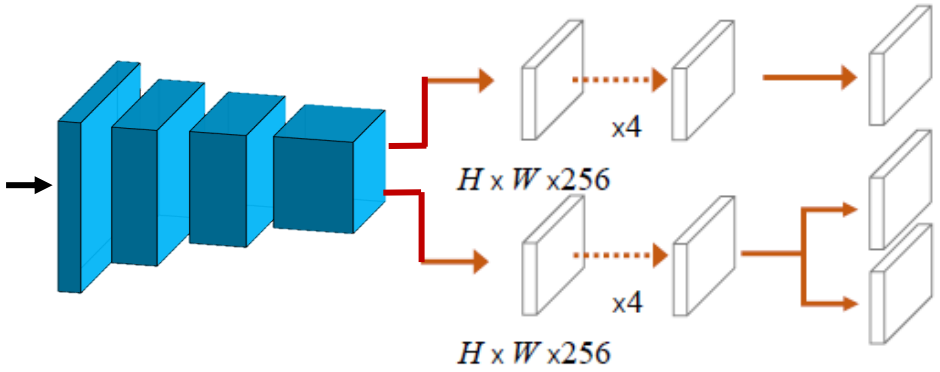
https://github.com/facebookresearch/detectron2



# Single-stage detectors

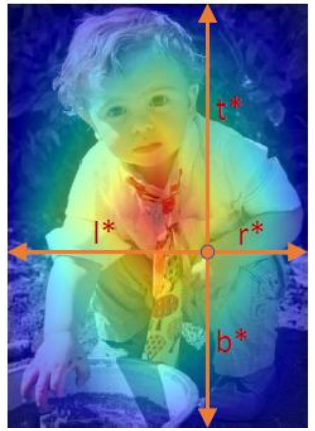
- Frame detection process similar to segmenation: *for each pixel predict the output* e.g., [bounding-box, object class, etc.]
- Early approaches: YOLO[1], SSD[2]
- More recent FCOS[3]:






Regression


Centerness




[1]Redmon, You Only Look Once: Unified, Real-Time Object Detection, CVPR2016
[2]Liu et al., SSD: Single Shot MultiBox Detector, ECCV2016
[3]Tian et al., FCOS: A Simple and Strong Anchor-free Object Detector, TPAMI2020

# Single-stage detectors

- Faster than two-stage detectors

| Method | Backbone | AP | $AP_{50}$ | $AP_{75}$ |
|---|---|---|---|---|
| Two-stage methods: | | | | |
| Faster R-CNN+++ [47] | ResNet-101 | 34.9 | 55.7 | 37.4 |
| Faster R-CNN w/ FPN [9] | ResNet-101-FPN | 36.2 | 59.1 | 39.0 |
| Faster R-CNN by G-RMI [51] | Inception-ResNet-v2 [52] | 34.7 | 55.5 | 36.7 |
| Faster R-CNN w/ TDM [53] | Inception-ResNet-v2-TDM | 36.8 | 57.7 | 39.2 |
| One-stage methods: | | | | |
| YOLOv2 [30] | DarkNet-19 [30] | 21.6 | 44.0 | 19.2 |
| SSD513 [5] | ResNet-101-SSD | 31.2 | 50.4 | 33.3 |
| YOLOv3 $608 \times 608$ [6] | Darknet-53 | 33.0 | 57.9 | 34.4 |
| DSSD513 [54] | ResNet-101-DSSD | 33.2 | 53.3 | 35.2 |
| RetinaNet [7] | ResNet-101-FPN | 39.1 | 59.1 | 42.3 |
| CornerNet [32] | Hourglass-104 | 40.5 | 56.5 | 43.1 |
| FSAF [33] | ResNeXt-64x4d-101-FPN | 42.9 | 63.8 | 46.3 |
| CenterNet511 [55] | Hourglass-104 | 44.9 | 62.4 | 48.1 |
| FCOS w/ deform. conv. v2 | ResNeXt-32x8d-101-BiFPN | **50.4** | **68.9** | **55.0** |

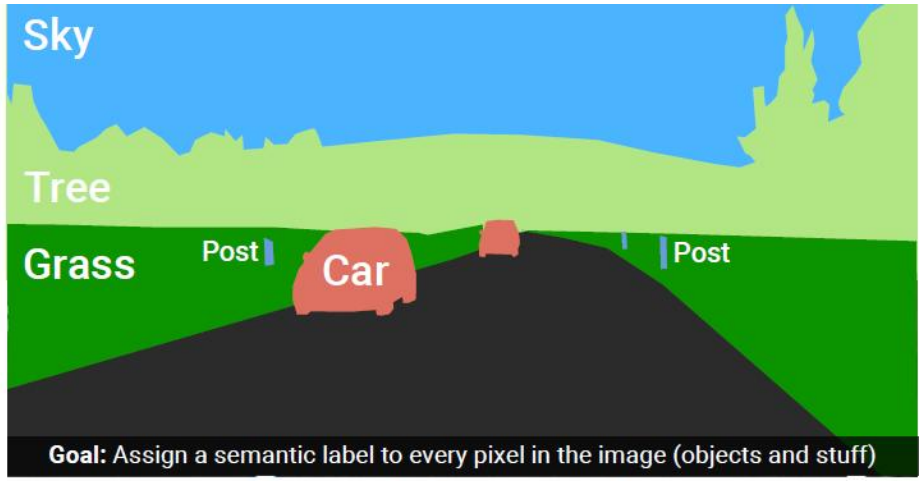
Table & Image from FCOS[1]

- Recent sota: Transformers, e.g., DETR[2]

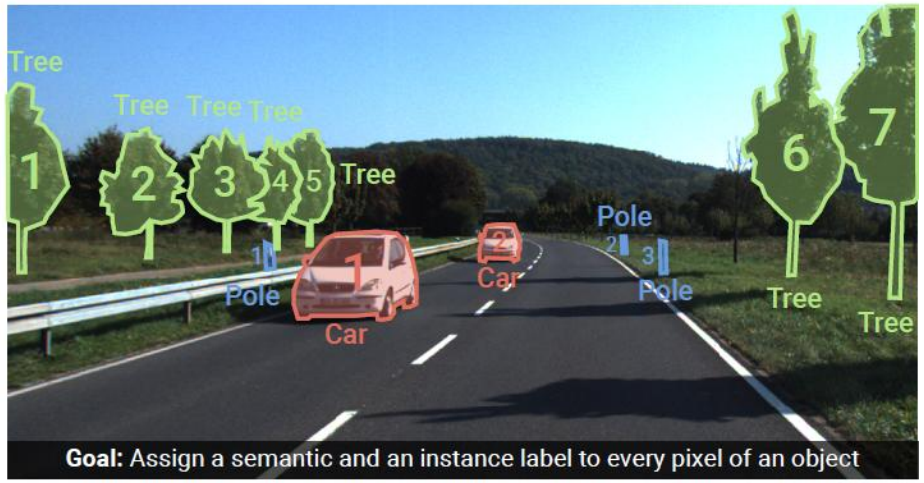
[1] Tian et al., FCOS: A Simple and Strong Anchor-free Object Detector, TPAMI2020

[2] Carion et al., End-to-End Object Detection with Transformers, ECCV2020

# Panoptic segmentation




Semantic Segmentation




Instance Segmentation

Holistic scene understanding requires for each pixel:

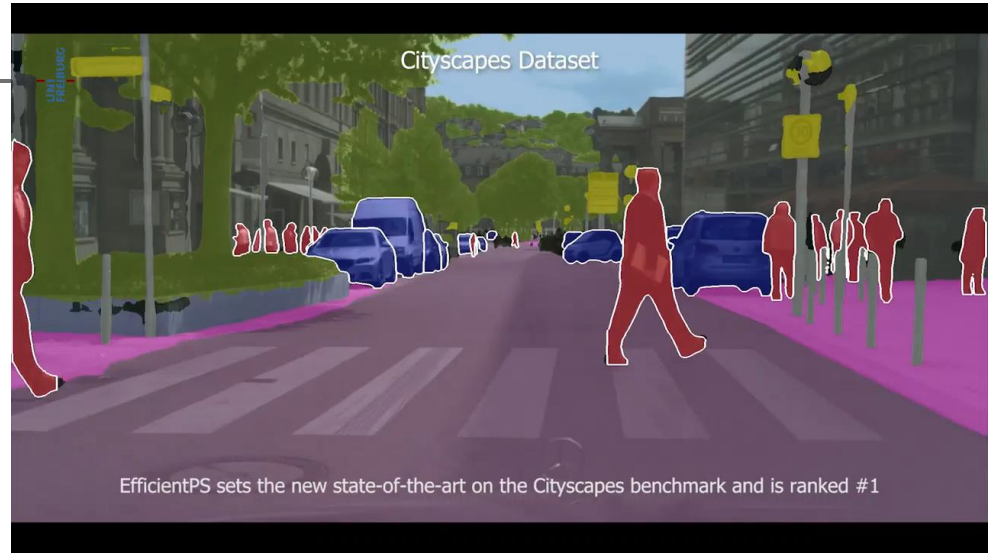
Stuff & instance labels


Panoptic segmentation

https://kharshit.github.io/blog/2019/10/18/introduction-to-panoptic-segmentation-tutorial

# Panoptic segmentation

- Combines object detection and stuff segmentation



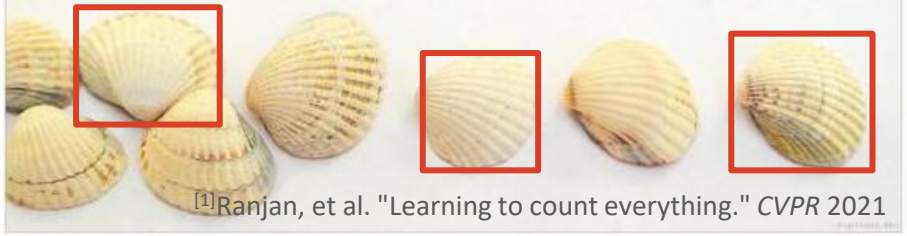

https://www.youtube.com/watch?v=j11mvFqFmfA&t=110s


Žust & Kristan, PanSR: An Object-Centric Mask Transformer for Panoptic Segmentation, Arxiv2024

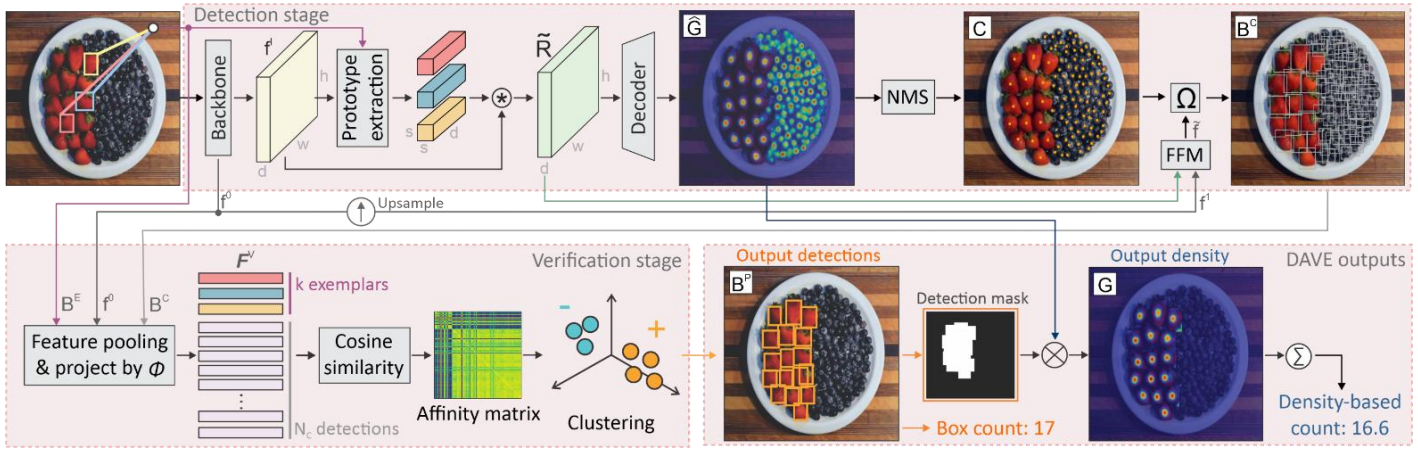
# Few-shot counting[1] … by detection

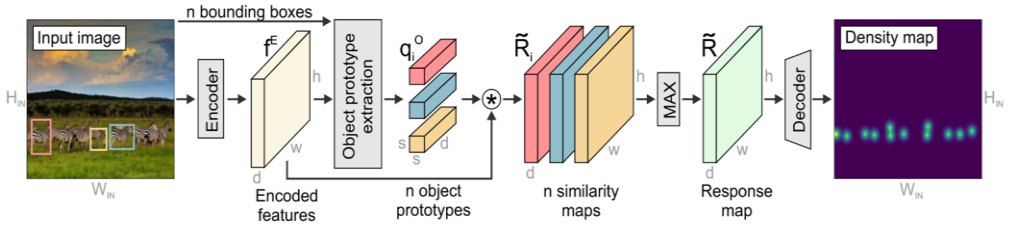
Input: Image + N exemplars

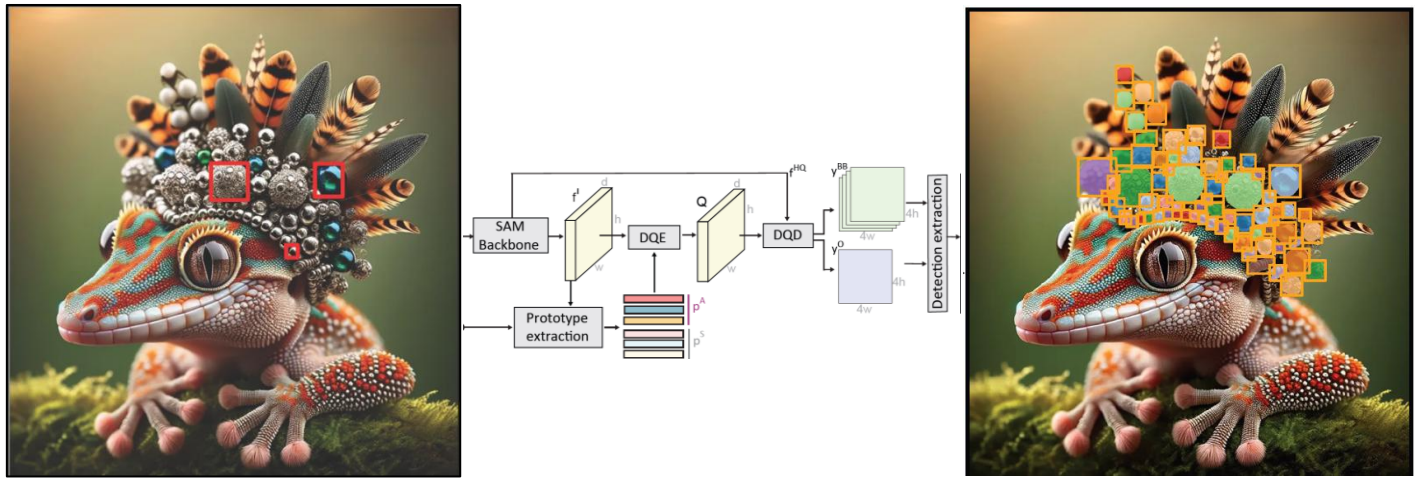
Output: 8



[1]Ranjan, et al. "Learning to count everything." *CVPR* 2021



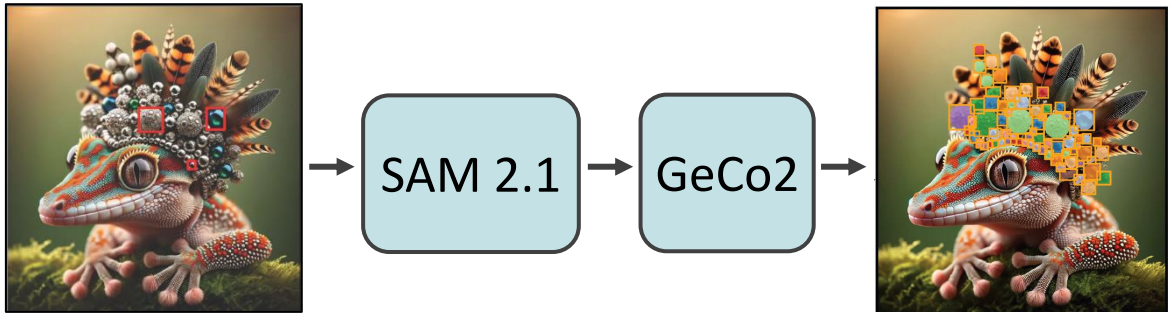

Đukić, Lukežič, Zavrtanik, Kristan. *A Low-Shot Object Counting Network With Iterative Prototype Adaptation*, ICCV2023

Pelhan Lukežič, Zavrtanik, Kristan. *DAVE -- A Detect-and-Verify Paradigm for Low-Shot Counting,* CVPR 2024

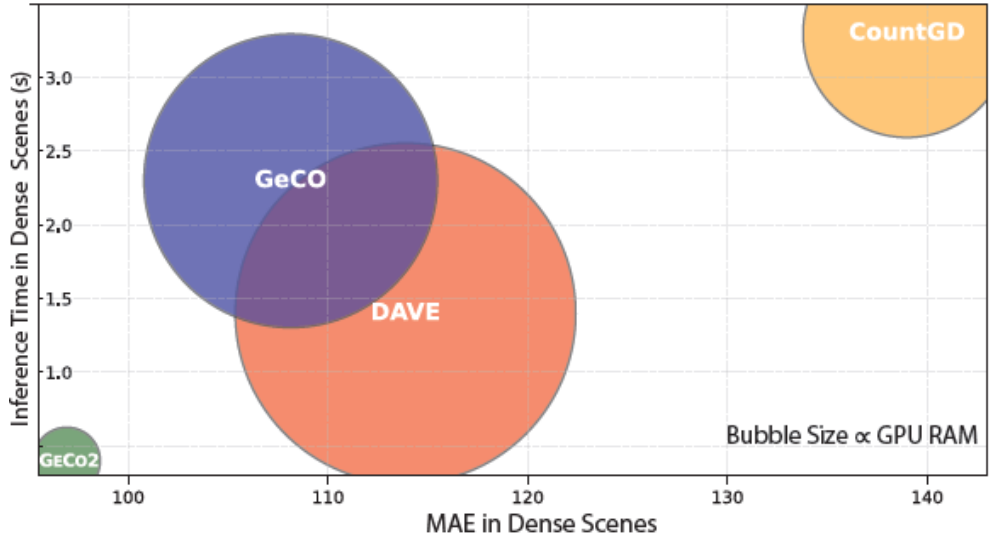
Pelhan, Lukežič, Zavrtanik, Kristan. *A Novel Unified Architecture for Low-Shot Counting by Detection and Segmentation*, NeurIPS2024

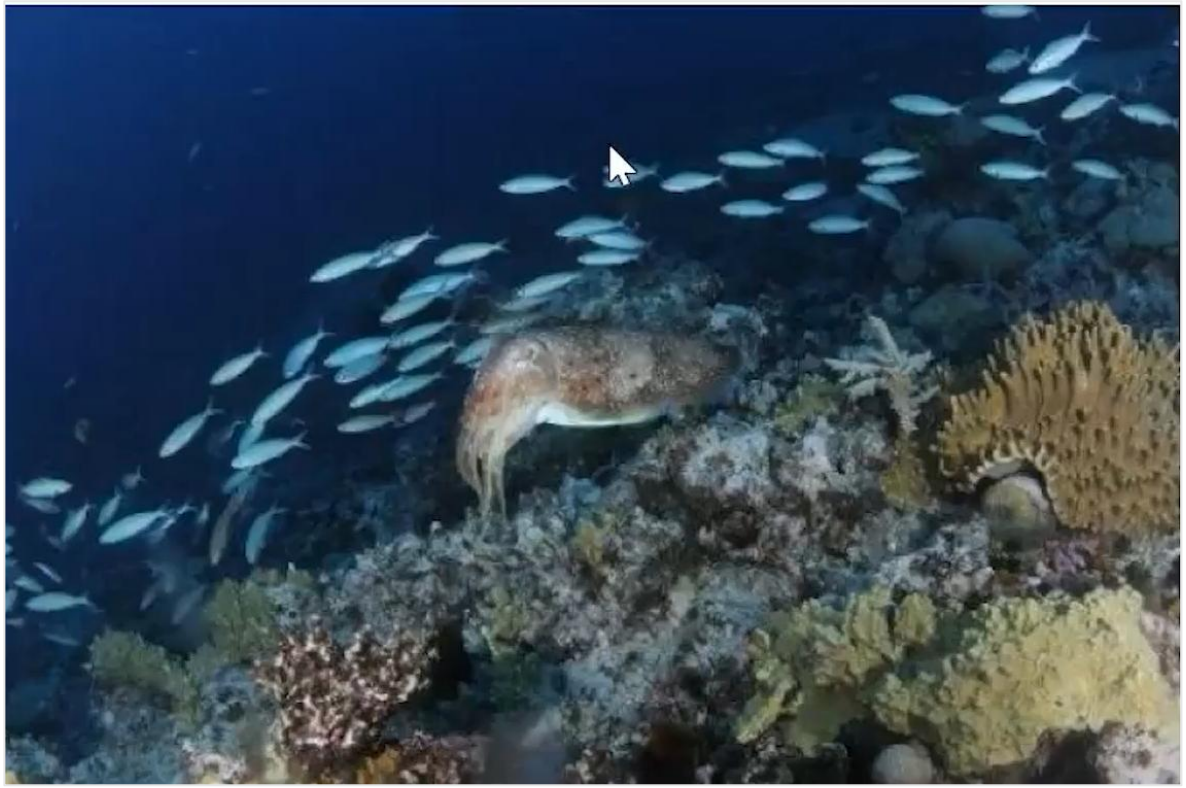
Pelhan, Lukežič, Kristan. *Generalized-Scale Object Counting with Gradual Query Aggregation*, Arxiv 2025

# Few-shot counting[1] … by detection







https://huggingface.co/spaces/jerpelhan/GECO2-demo

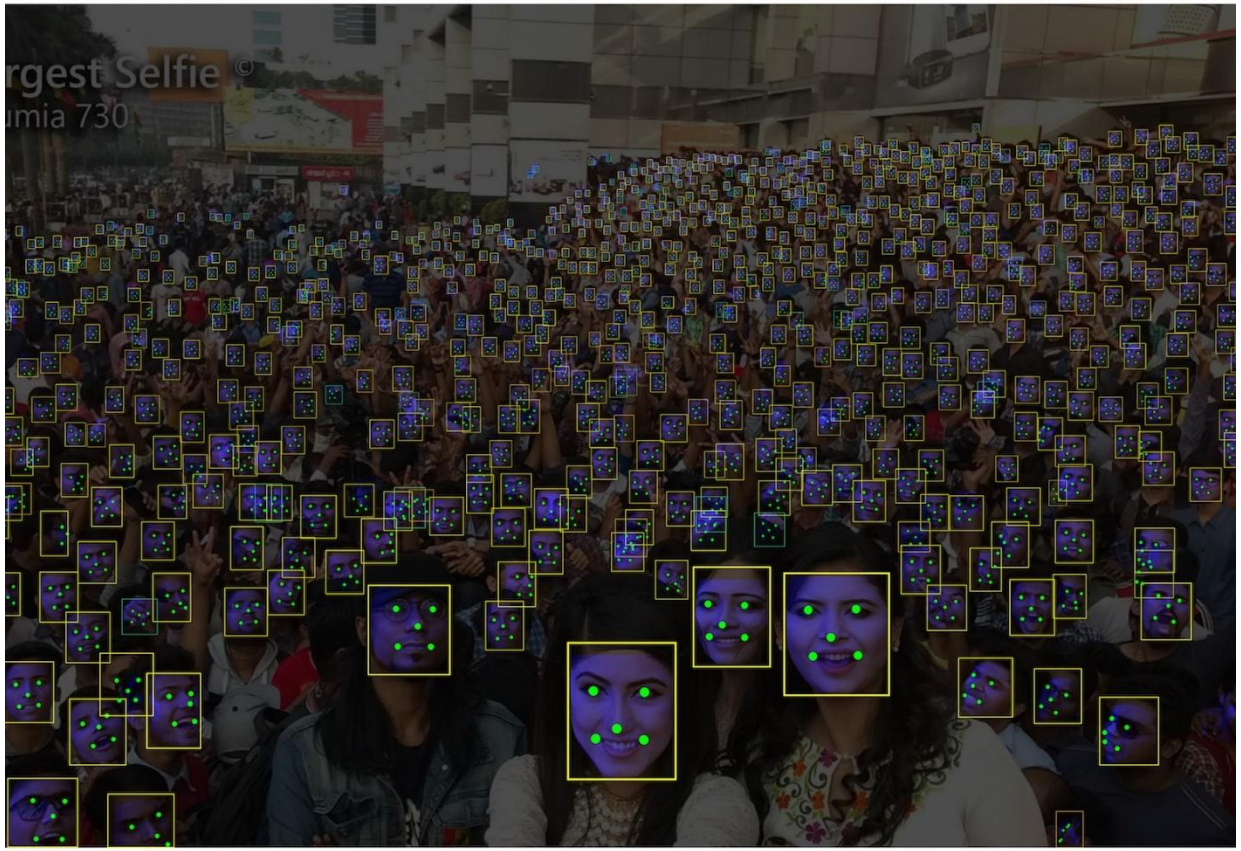
- SAM2 extension
- Fast & reliable
- Excellent in dense regions

Pelhan, Lukežič, Kristan. *Generalized-Scale Object Counting with Gradual Query Aggregation*, AAAI 2026

# Huge performance leaps

## Viola-Jones (2001) vs RetinaFace (2019)


Viola, Jones, *"Rapid Object Detection using a Boosted Cascade of Simple Features"*, CVPR2001

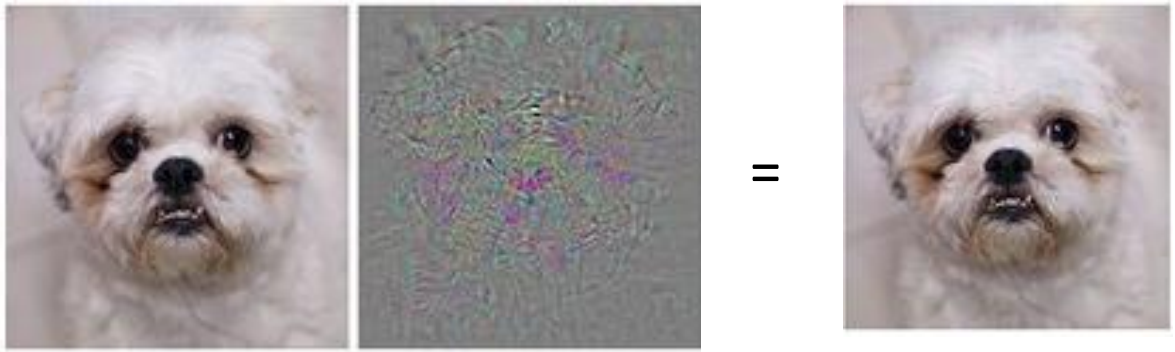



Deng et al., RetinaFace: Single-stage Dense Face Localisation in the Wild, Arxiv2019

Viola-Jones detection image credit: Žiga Emeršič

# CNNs and “human performance” fallacy

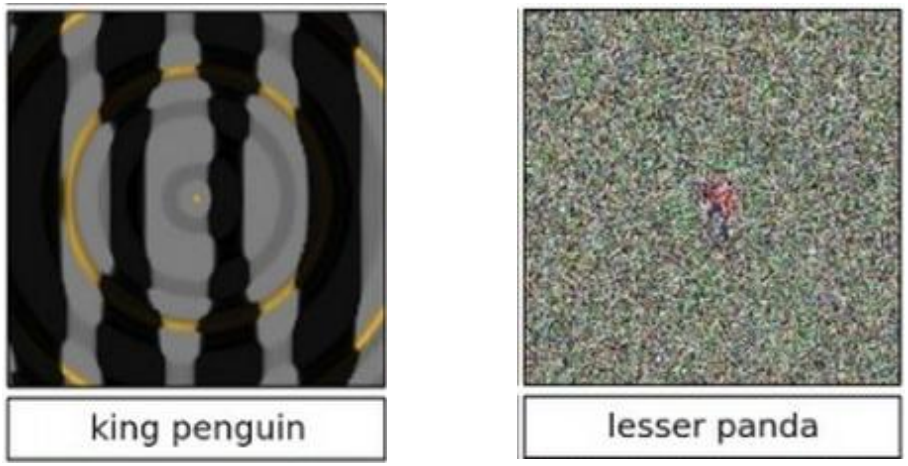
- Adding small (but specific!) perturbations to images



dog + perturbation

Ostrich!

- Generating „adversary“ images





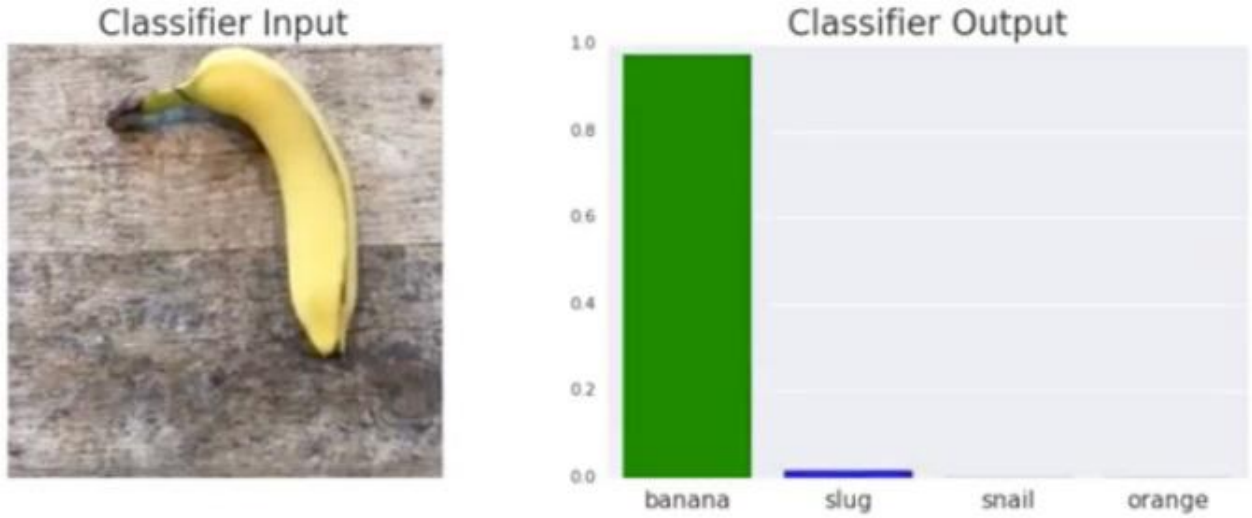


Over 99.6% confidence in decision!

Nguyen et al., Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images, CVPR 2015 (http://www.evolvingai.org/fooling)

https://karpathy.github.io/2015/03/30/breaking-convnets/

# CNNs and “human performance” fallacy




https://techcrunch.com/2018/01/02/these-psychedelic-stickers-blow-ai-minds

# A look forward: Convergence of methodology

- Methodology across different *computer vision subfields* (object detection, segmentation, 3D vision, tracking...) is converging
- AND, the methodology across the *different disciplines* (computer vision, natural language processing, speech processing) is converging
- DALL-E [OpenAI2021] (12 $10^9$ params)

"…an armchair in the shape of an avocado…"

# Look forward: Convergence of methodology

- Current mantra: larger datasets & more parameters = better performance
- Challenges:

  learning from few examples, computational reduction, architectures that reason,...
- Keep in mind the trivia:
  - 15 year methodological cycles since 60's
  - The last one started ~2012
  - Exponential advancements expected
  - Foundation models? Cross-modal learning? Agentic processing? Architecture paradigm shifts?

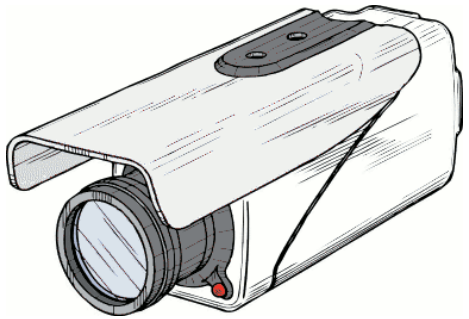
# Conclusion

- This brief overview covered merely some basics in modern CV
- For recent advancements, see proceedings of the major conferences:
  - Computer vision and pattern recognition (CVPR)
  - Neural information processing systems (NeurIPS)
  - International conference on computer vision (ICCV)
  - European conference on computer vision (ECCV)
  - International Conference on Learning Representations (ICLR)
  - International Conference on Machine Learning (ICML)

Univerza v Ljubljani

ViCOS visual cognitive systems lab

# Machine Perception
# Recognition and detection using local features



Matej Kristan

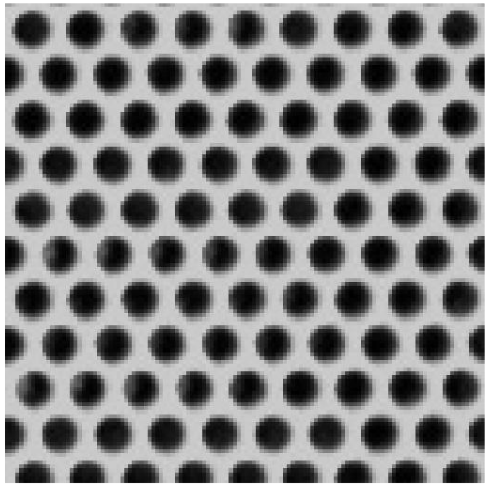

Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

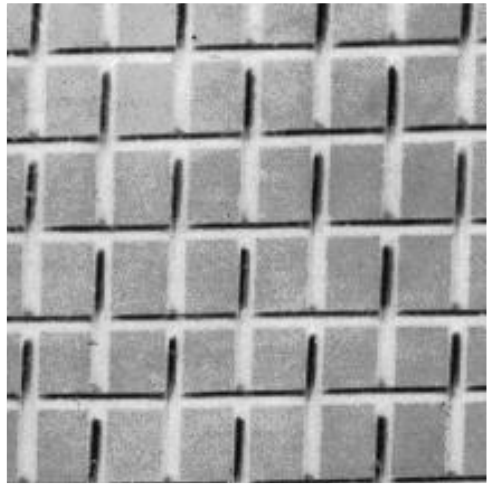
Machine perception

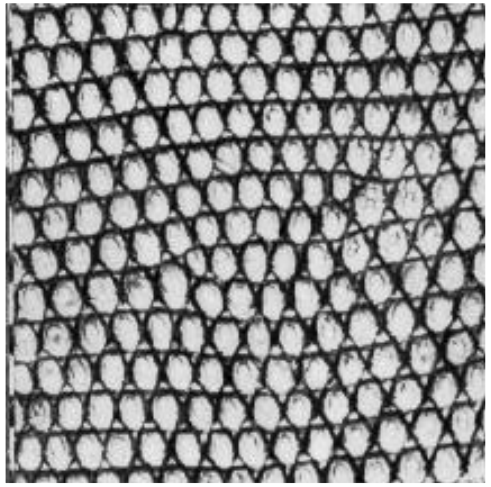
# RECOGNITION USING LOCAL FEATURES: *BAG OF WORDS MODELS*

# Intuition: texture recognition

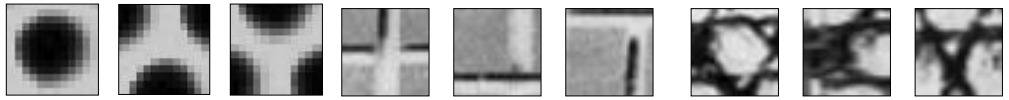
- What is texture? Could say: “spatially organized repeatable images”



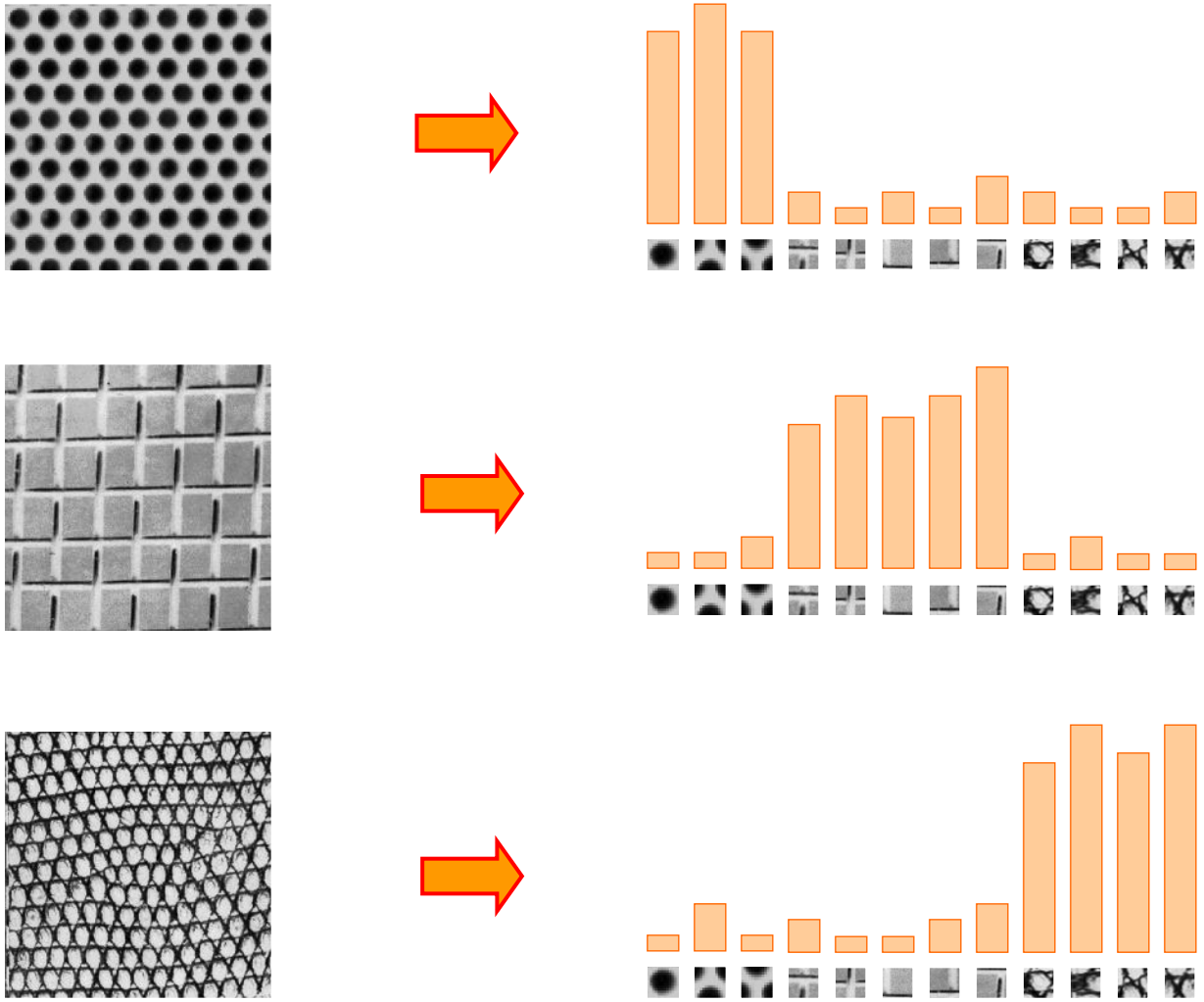




- Texture can be characterized in terms of textons (small „images“)



# Intuition: texture recognition



Note the difference in the histograms!

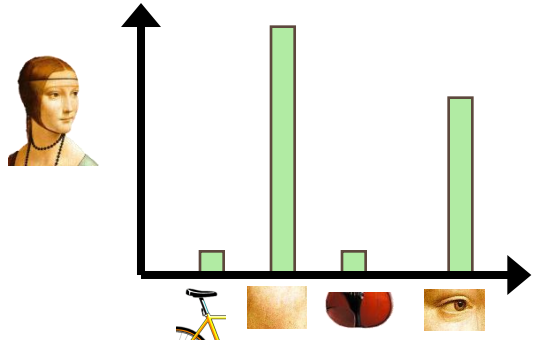
Julesz, 1981; Cula & Dana, 2001; Leung & Malik 2001; Mori, Belongie & Malik, 2001; Schmid 2001; Varma & Zisserman, 2002, 2003; Lazebnik, Schmid & Ponce, 2003

# Bag of words models

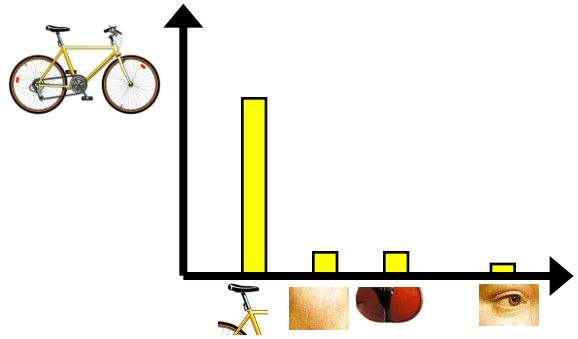
Object → Bag of „words“

# Bag of visual words

- Summarize an image by a distribution (histogram) over visual words.



- Analogous to text-based information retrieval systems – think of Google.



- Except: how to identify the “words”?

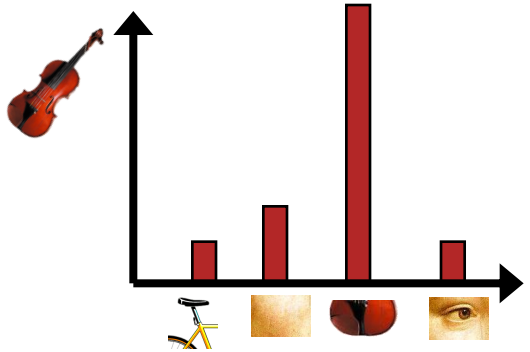
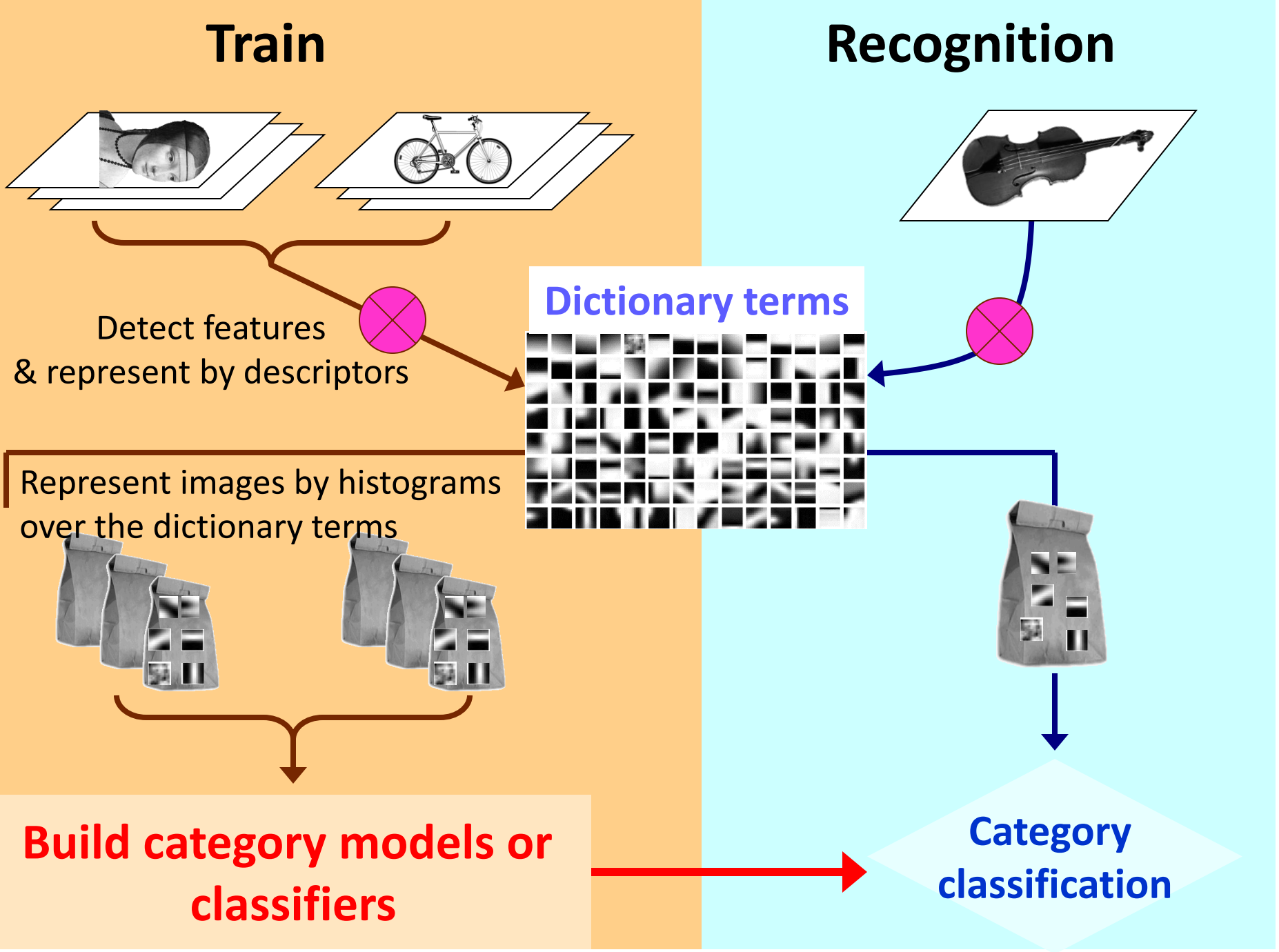
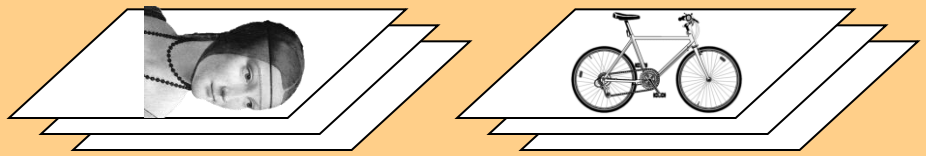


Image credit: Li Fei-Fei

Slide credit: Kristen Grauman

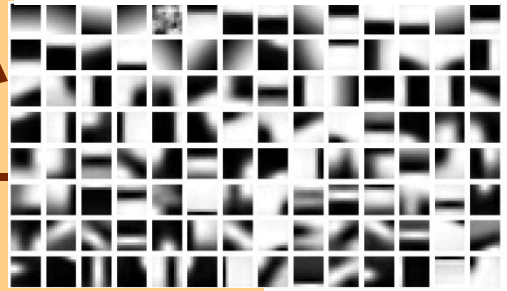

Train
Recognition
Detect features
& represent by descriptors
Dictionary terms
Represent images by histograms
over the dictionary terms
Build category models or
classifiers
Category
classification

# Train



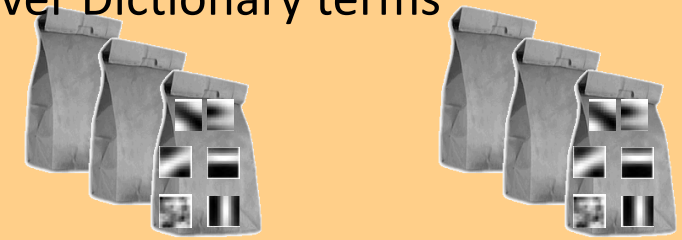
**1.** Detect features
& represent by descriptors

**2.**
**Dictionary terms**



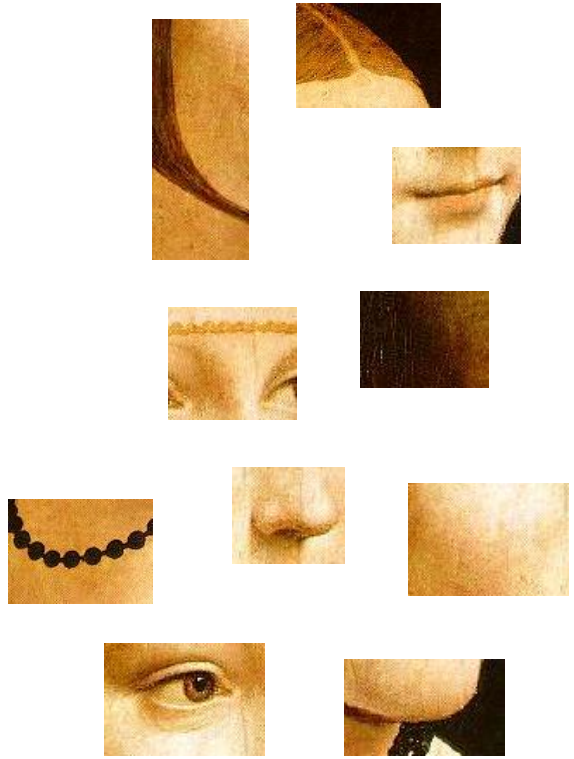
Represent images by histograms
over Dictionary terms

**3.**



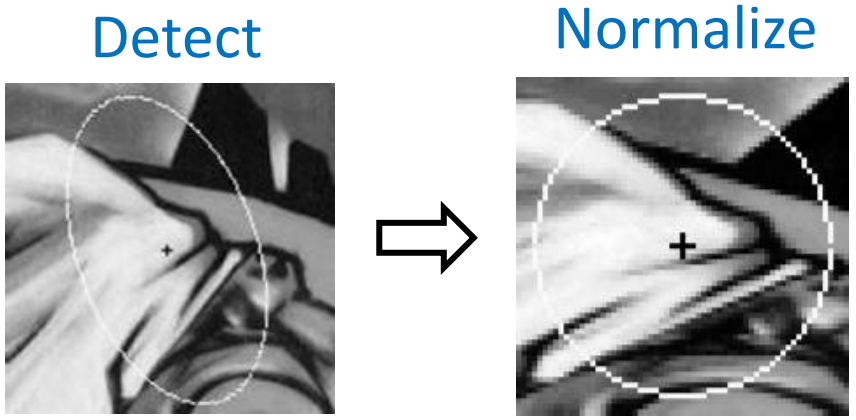
**Build category models or classifiers**

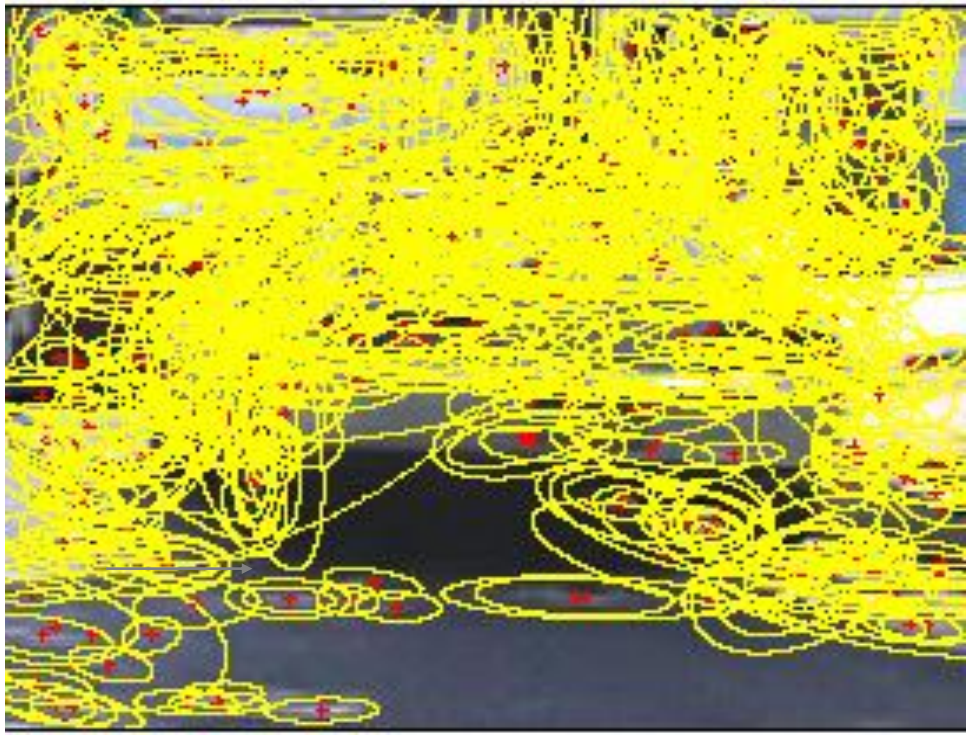
# 1. Feature detection & representation



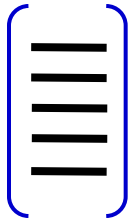
# 1.0 Feature detection & representation

- Use feature point detectors (we have studied quite a few)
  - E.g., SIFT
- Normalize each region to remove local geometric deformation



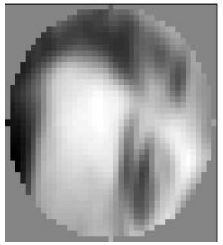



# 1.1 Feature detection & representation

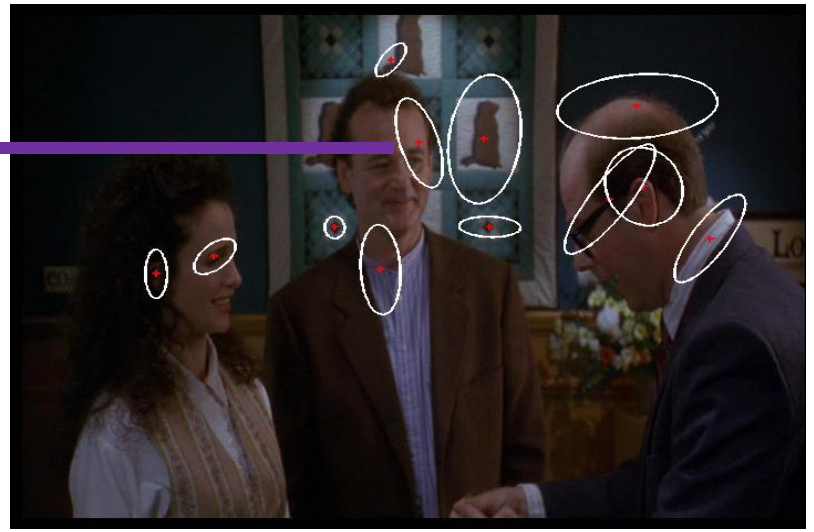


**Calculate the SIFT descriptor**

[Lowe'99]



**Normalize the region**



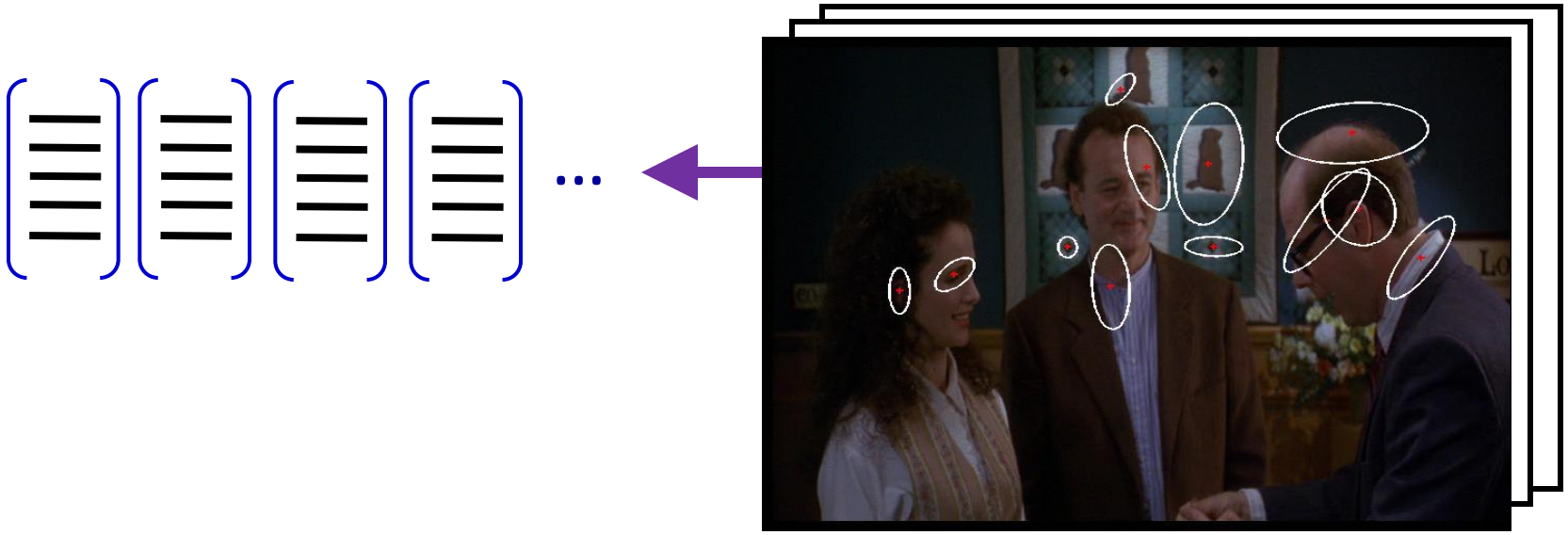
Detect regions

[Mikojaczyk and Schmid '02]

[Matas et al. '02]

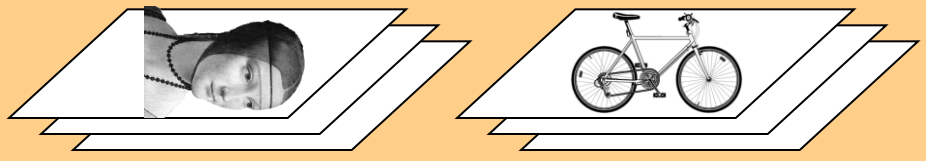
[Sivic et al. '03]

# 1.2 Feature detection & representation



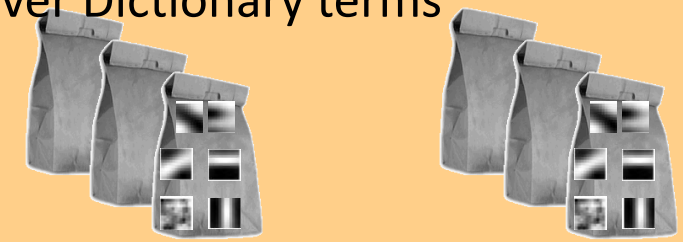
Collect descriptors from all key-points from all training images.

# Train



Detect features
& represent by descriptors

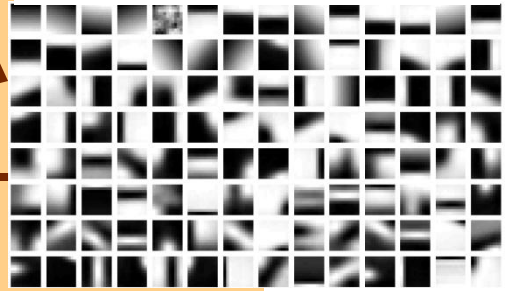
Represent images by histograms over Dictionary terms



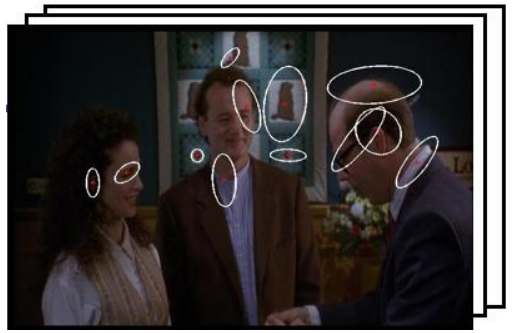
**Build category models or classifiers**

## 2.

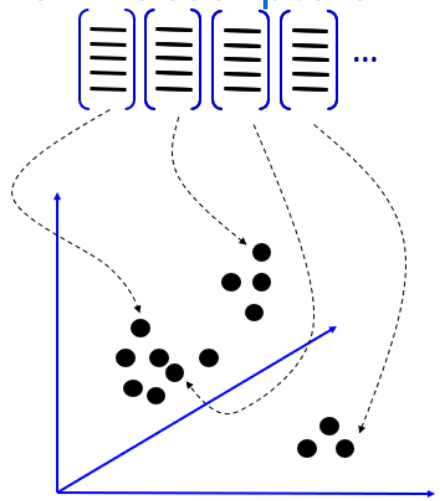
## Dictionary terms



Training images



SIFT descriptors



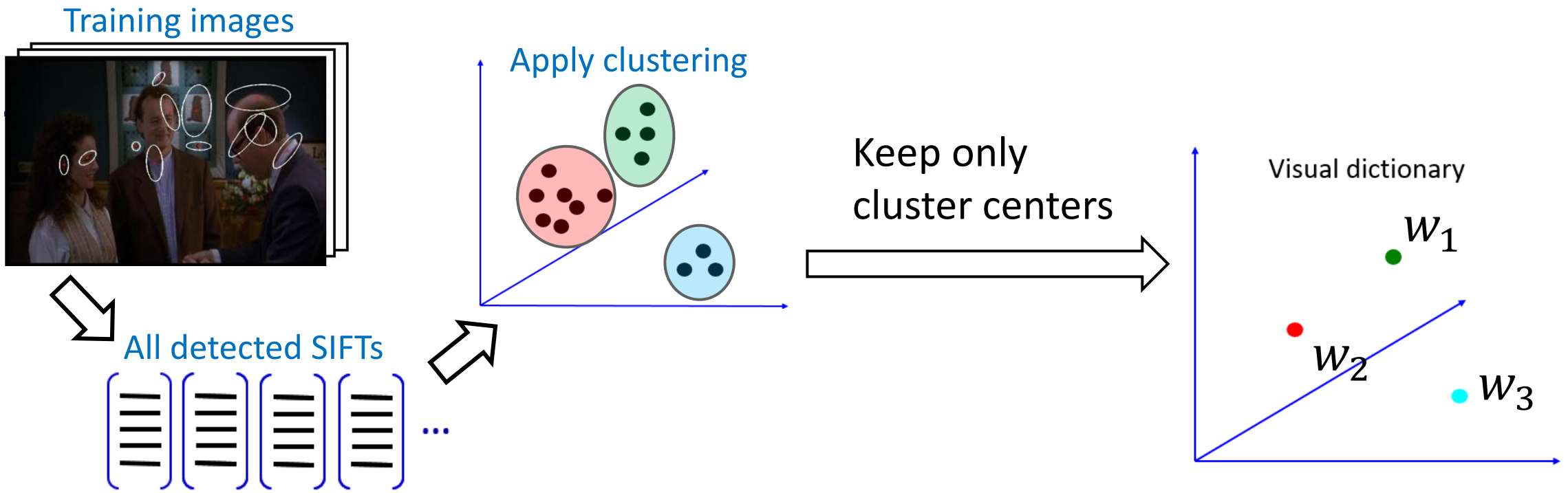
Dictionary:
How to create „word“ labels from the extracted SIFT descriptors?

SIFTs corresponding to the same „visual word“ should be similar.

Similar SIFTs form clusters!

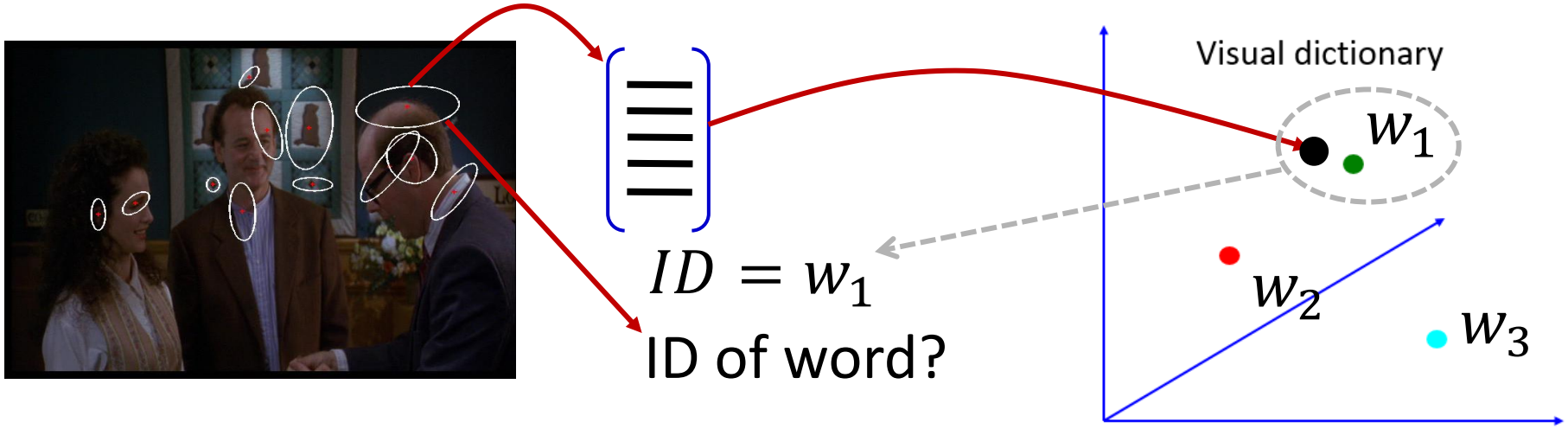
# 2.1 Clustering by vector quantization

- A standard approach to learning the visual codebook
  - Apply K-means clustering to the detected SIFT descriptors
  - Each cluster center is a visual word (for the dictionary)
  - Important: Learn the code-book on separate training data (!!! This is learning stage!)

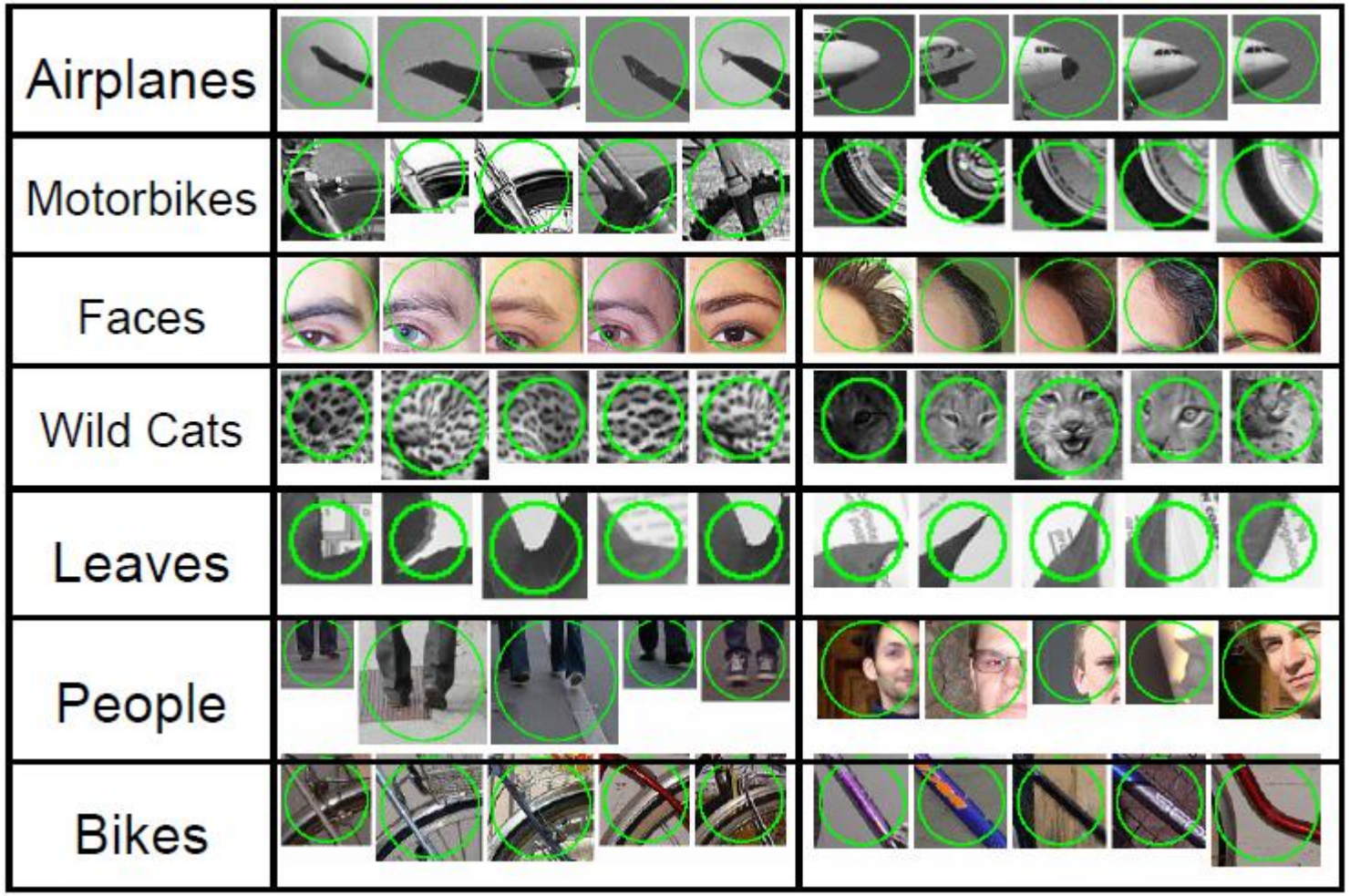


# 2.1 Clustering by vector quantization

- How to assign visual words to the detected SIFTs?
  - For each detected keypoint, assign the word ID closest to the extracted SIFT.
  - Codebook = visual dictionary (vocabulary)



# 2.2 Visual dictionary – example



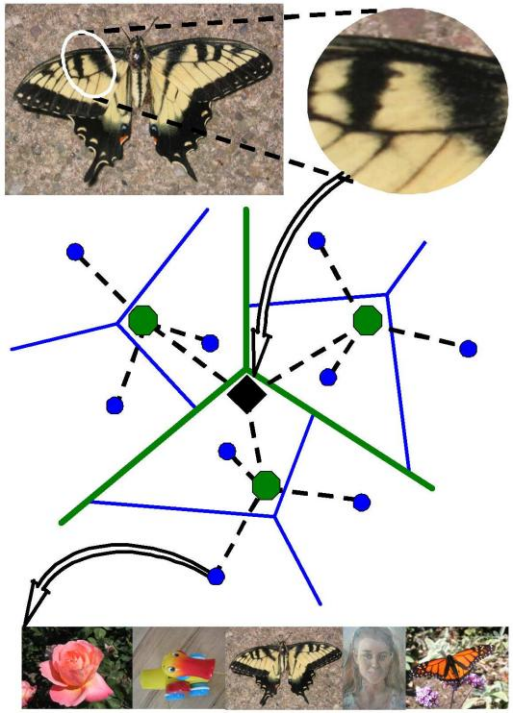

Slide credit: C. Schmidt

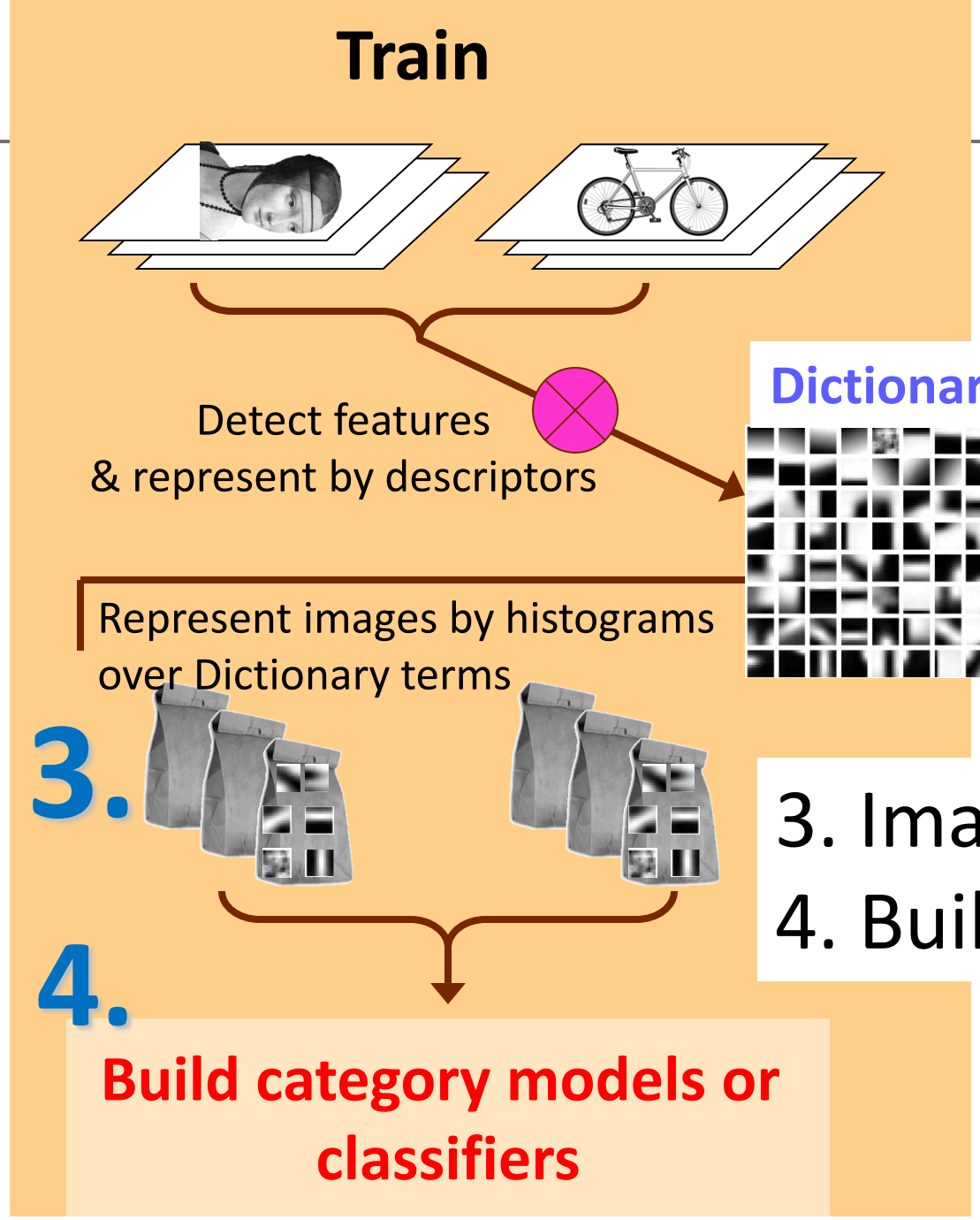
# 2.2 Visual dictionary – issues

- How to choose dictionary size?
  - Too small: visual words not expressive enough to describe all possible patches.
  - Too large: visual words too similar to discriminate well

- Computational efficiency in matching (need to compare many keypoints to many visual words in dictionary)
  - Vocabulary trees

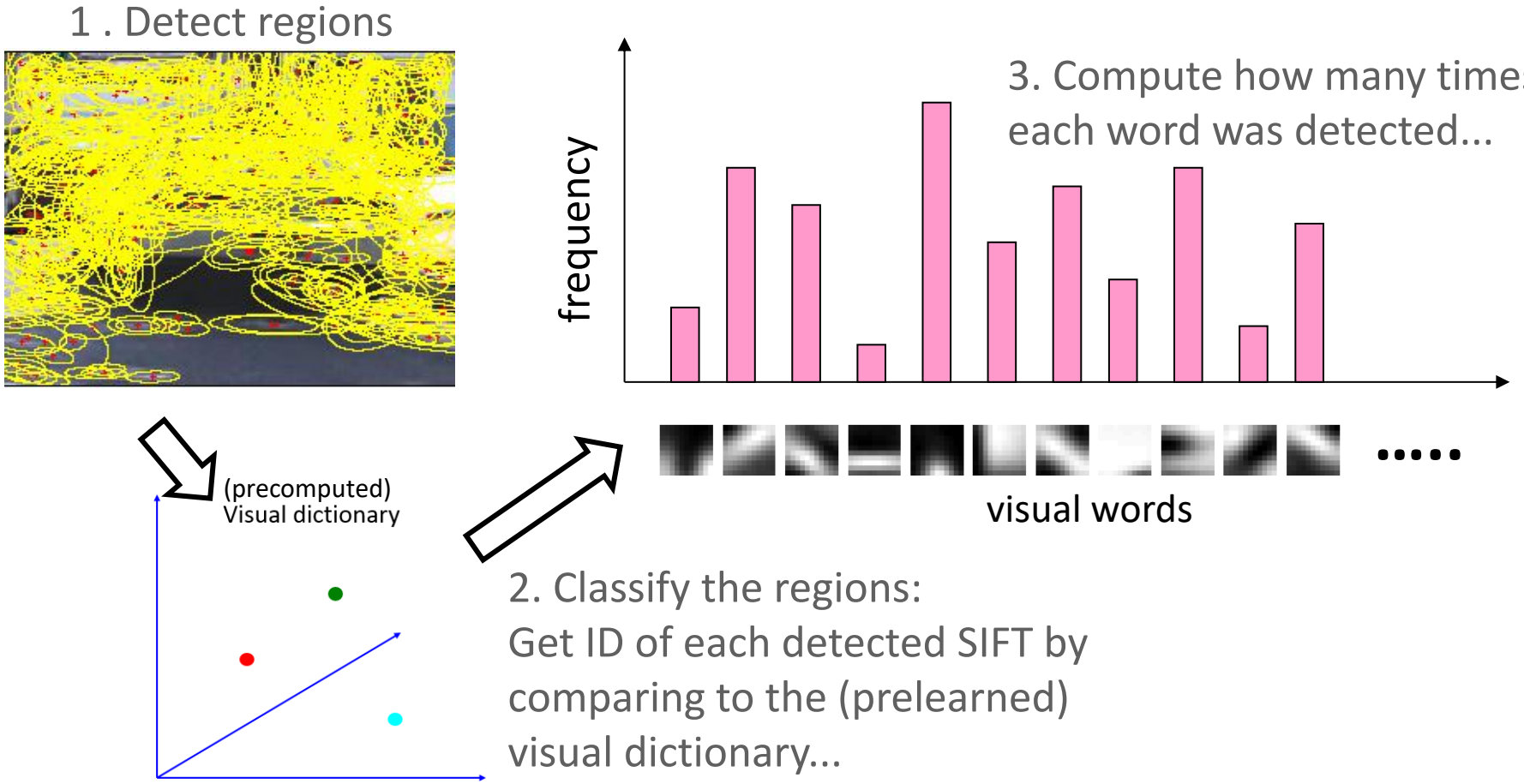
    D. Nistér and H. Stewénius, *"Scalable recognition with a vocabulary tree,"* in Proc. CVPR, 2006






3. Image representation
4. Build a classifier

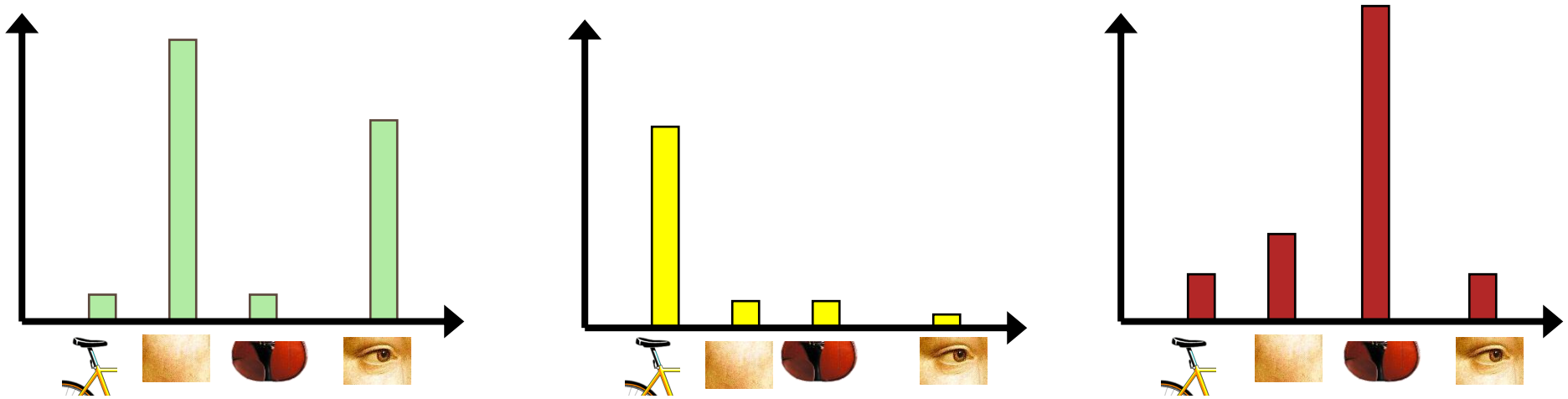
# 3. Image representation




- Each image is represented by a 1000-4000 dimensional histogram, which is then normalized (L1/L2 norm)

# 4. Build a classifier

- Using the training set, we have first built a visual vocabulary.
- The vocabulary can be now used to encode any image with the histogram
- As the final stage of learning, we need to train a classifier that will classify images based on the extracted bag of word histograms.



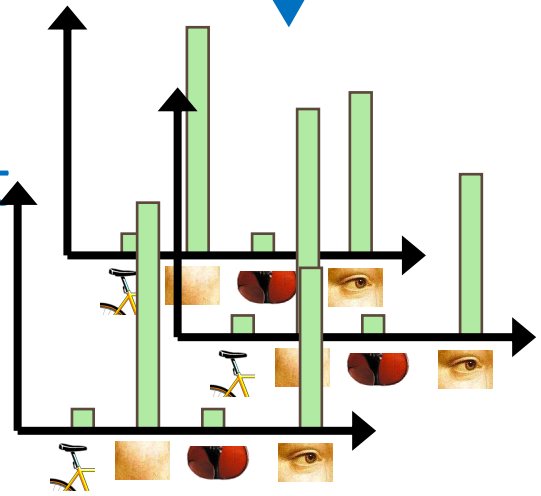
# 4.1 Build a classifier by SVM

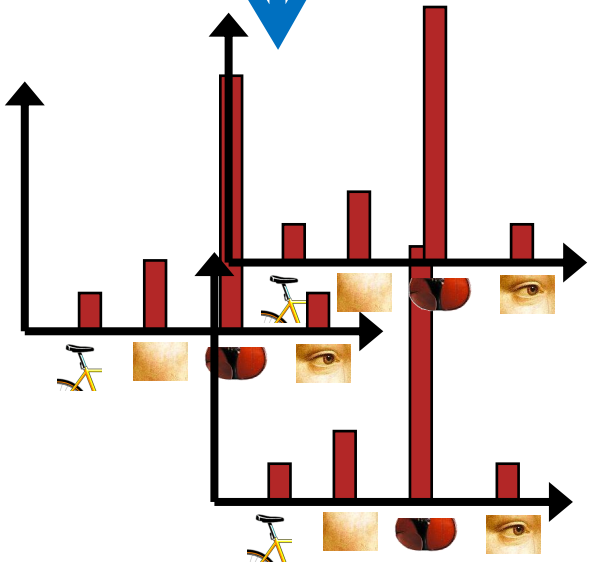
Category 1

Category 2

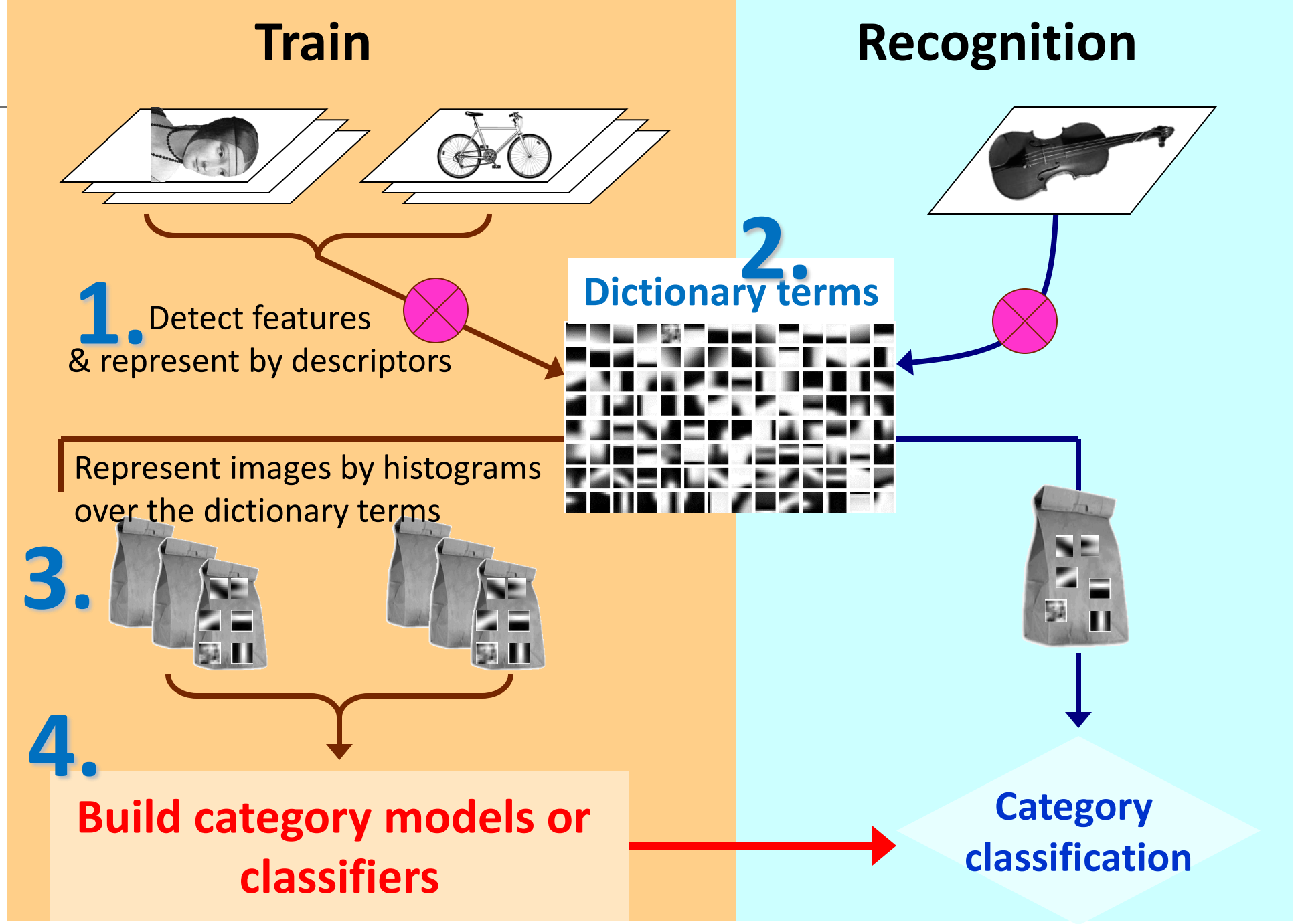
Train a classifier
e.g., SVM

Extract
BOWs





Extract
BOWs

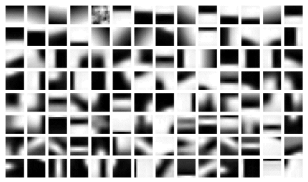

Train
Recognition
1. Detect features & represent by descriptors
2. Dictionary terms
Represent images by histograms over the dictionary terms
3.
4.
Build category models or classifiers
Category classification

# 5. Recognition

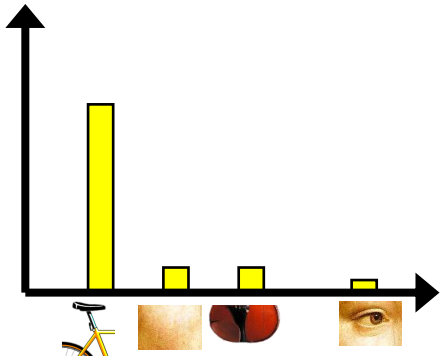
- How to classify a new image?
- Encode the image with the dictionary learned in the training stage
- Feed to a classifier trained at training stage

New image

Pre-learned dictionary



Encode by Bow



Apply a pre-trained SVM

Classified as: motorbike

# 6. BoW application in practice

- Performs very well in image classification despite the background clutter...

# 6.1 Examples of false classification

Books classified as faces and buildings

Buildings classified as faces and trees

Cars classified as buildings and phones

# 6.2 Bags of words: Summary

- Strengths:
  - Fixed descriptor length.
  - Robust to object position and orientation

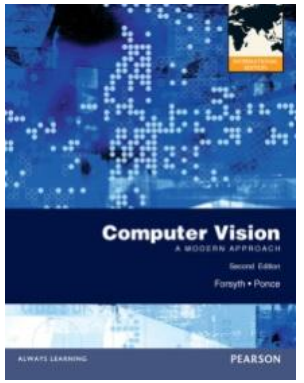
- Weaknesses:
  - Does not account for spatial relations among visual words.
  - Does not localize objects in the image.

Machine perception

# OBJECT DETECTION BY FEATURE CONSTELLATIONS

# Detection as a recognition problem

- How to detect an object in arbitrary pose and estimate that pose?
- Brute force sliding windows with exhaustive testing of all deformation parameters is not always a good option*.




scale

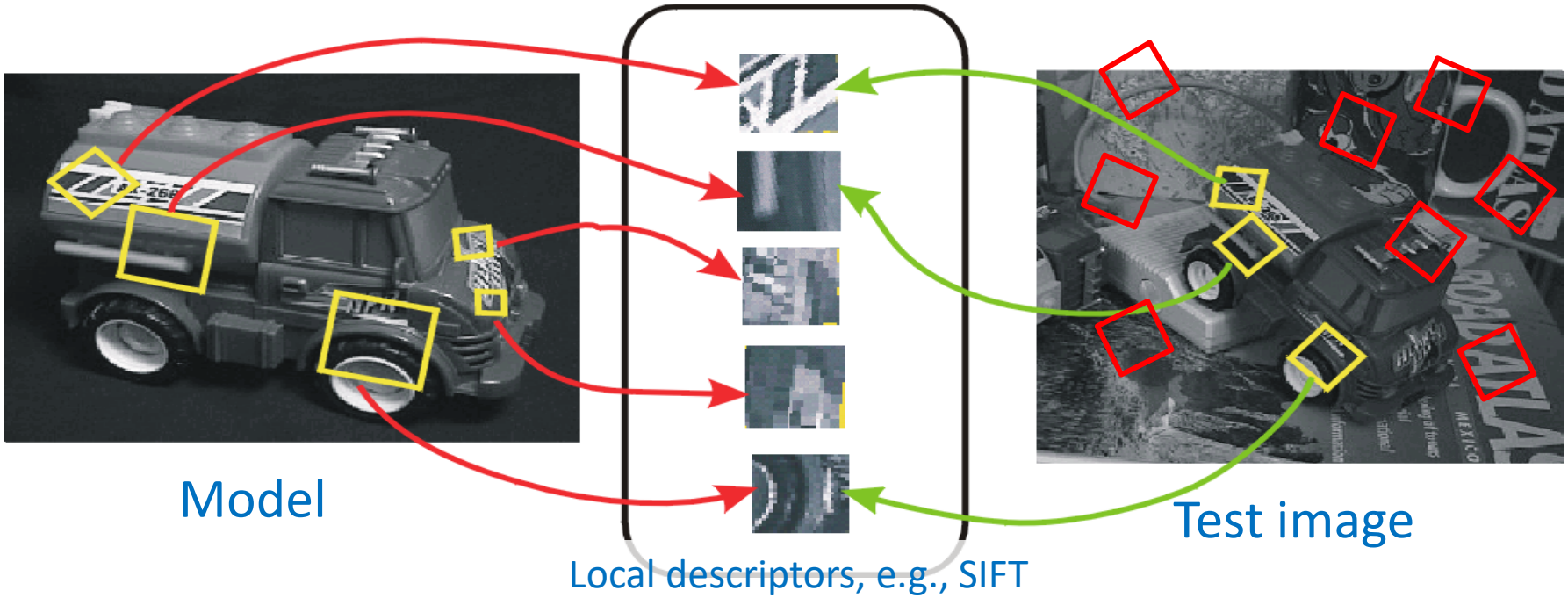
rotation


perspective

*deep learning detectors do apply sliding windows, but avoid testing all parameters…

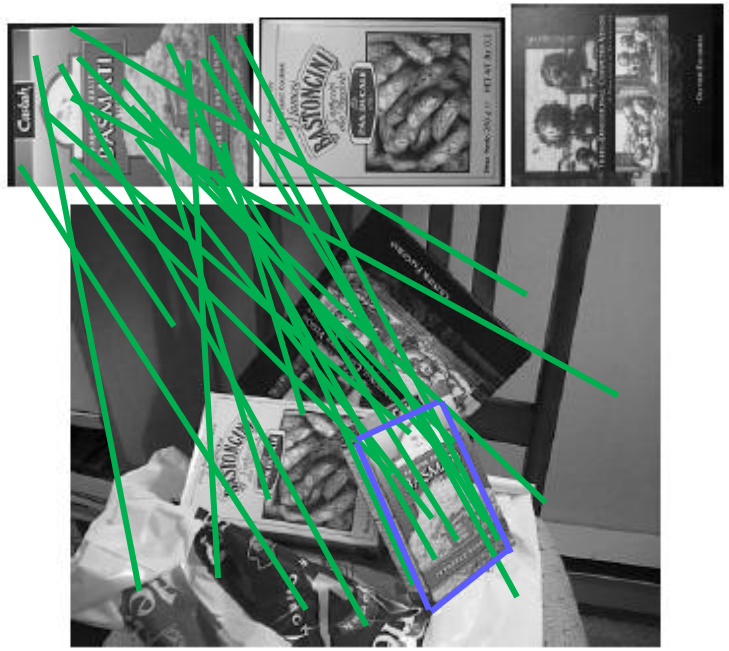
# Detection as a recognition problem

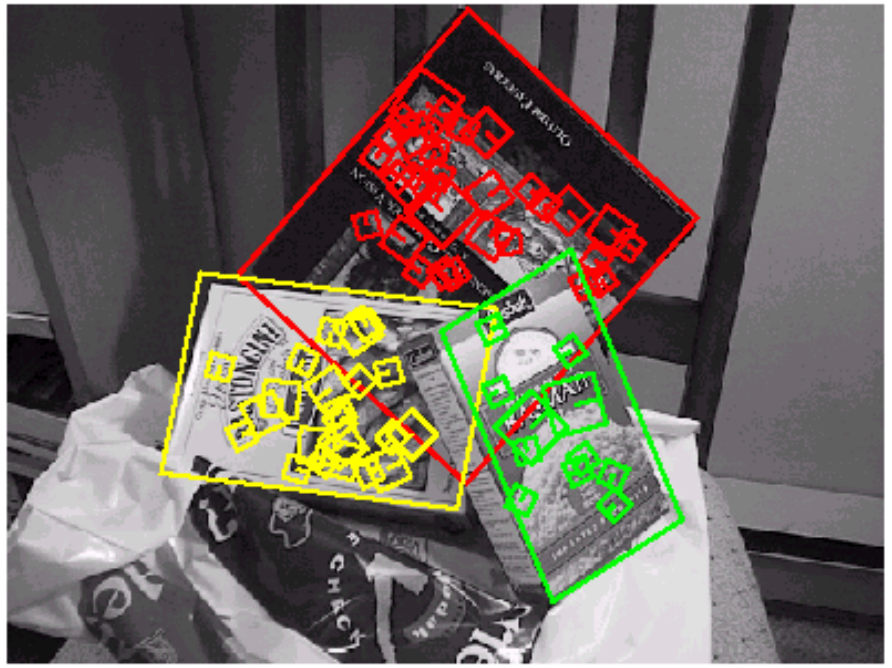
- Represent the target model in terms of small “parts” that can be detected even under an affine deformation (Keypoints!)
- Detection: (i) Detect “parts” in image ; (ii) Verify consistency of geometric configurations



# Detection by fitting an affine deformation

- Affine model approximates perspective transform of planar objects.
- Apply RANSAC to get a globally-valid correspondence.





# Detection by Generalized Hough Transform

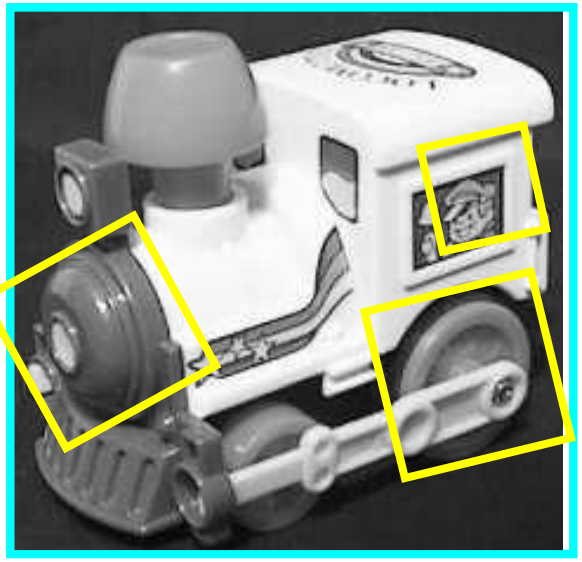
- Assume features are invariant to scale and rotation
  - Then each detected feature becomes a hypothesis of fitting (translation, rotation, scale)
- Each feature casts a vote into the Hough translation/rotation/scale space

model

# Detection by Generalized Hough Transform

- Assume features are invariant to scale and rotation
  - Then each detected feature becomes a hypothesis of fitting (translation, rotation, scale)
- Each feature casts a vote into the Hough translation/rotation/scale space

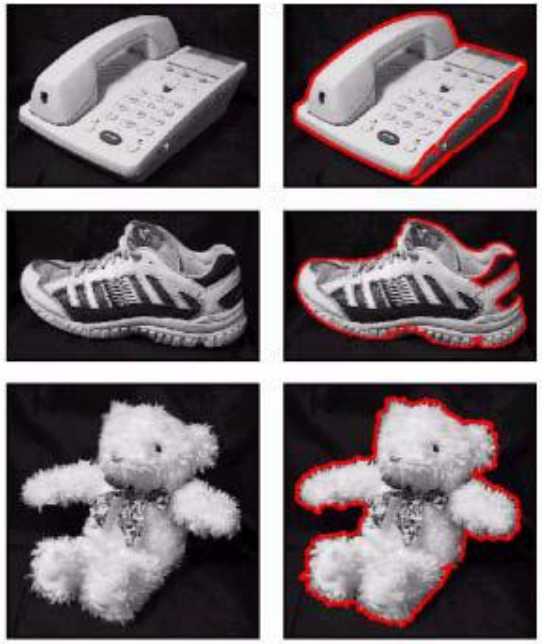
model



# Detection by GHT – summary

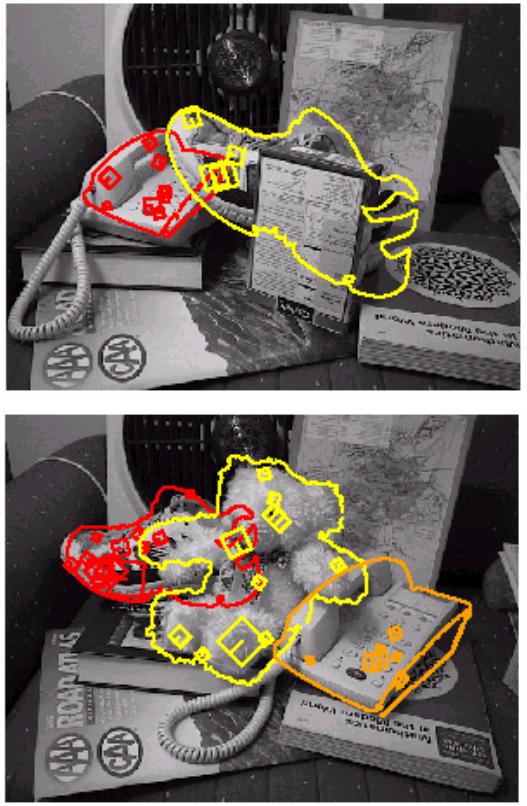

1. Index descriptors
   - Distinctive descriptors reduce the search space
2. Apply a generalized Hough transform (GHT) to obtain approximate detections
   - Key-points associated with local transformation, relative to coordinate frame of the object.
3. Refine each detection by fitting affine transform between the points on the object and the detected points from HGT
   - Fit and verify using features, which vote for the same cell in the Hough space (at least 3 votes)

# Detection results



Background subtraction
to remove background clutter
in training phase

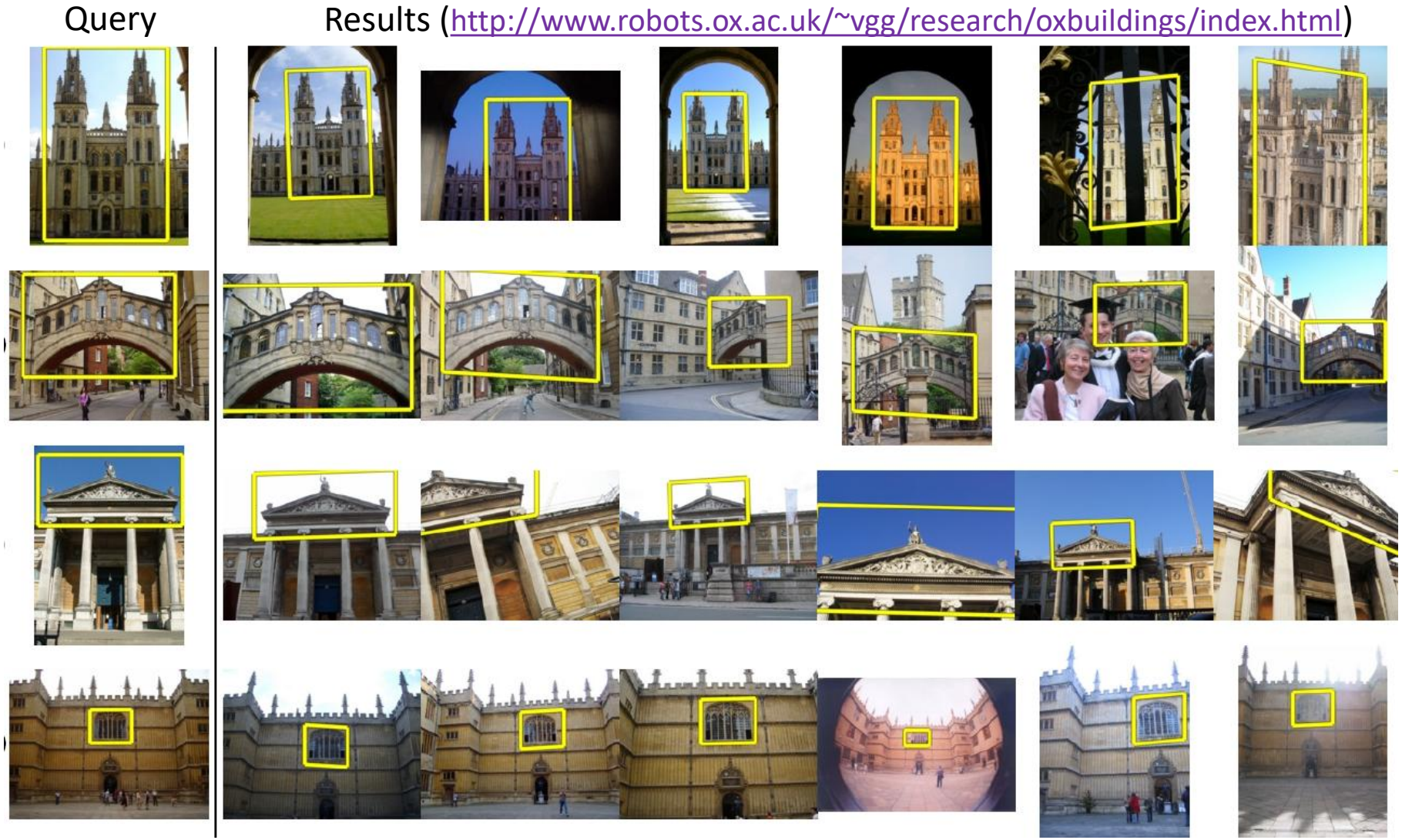
Detected objects



Detection despite
partial occlusion

Lowe, "Distinctive image features from scale-invariant keypoints." *IJCV* 2004.

# Applications: retrieval systems




Interesting work in retrieval: Radenovic, Tolias, and Chum: CNN Image Retrieval Learns from BoW: Unsupervised Fine-Tuning with Hard Examples , ECCV 2016

*Philbin et al.,*. Object retrieval with large vocabularies and fast spatial matching CVPR2007

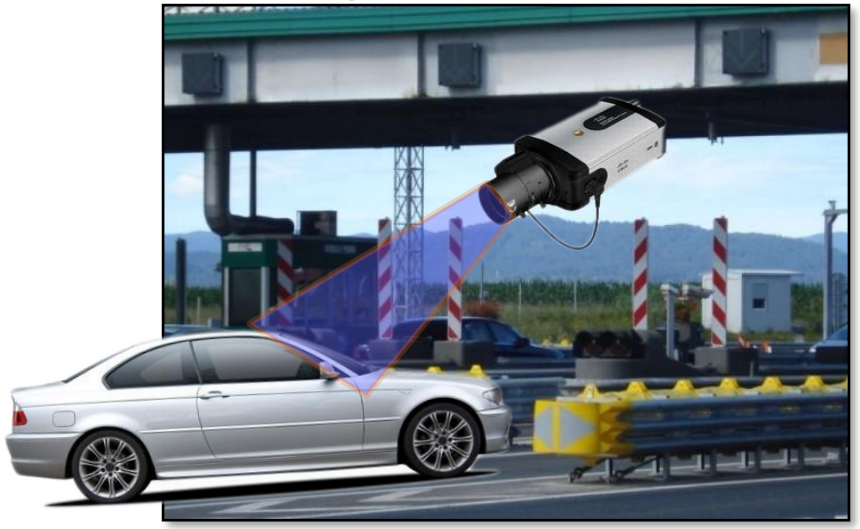
# Applications: specific object recognition

- Sony Aibo
  (Evolution Robotics)

- Application of SIFT
  - Recognition of the charging station
  - Comunication using visual cards

# Applications: Highway vignette verification

Highway checkpoint



# Applications: Augmented reality

- Match keypoints to template, estimate camera pose, project 3D object

# References


- David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition), (prva izdaja dostopna na spletu)
- Li Fei-Fei (Stanford), Rob Fergus (NYU), Antonio Torralba (MIT) , Recognizing and Learning Object Categories, (na spletu)
- Cordelia Schmid, Bag-of-features for category classification, lecture
- Lazebnik, Schmid, Ponce, Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories, CVPR, 2006
- Lowe, "Distinctive image features from scale-invariant keypoints." *IJCV* 2004

Machine perception

# SUMMARY AND OUTLOOK

# What did we learn?

- (1,2) Basic image processing
  - Thresholding, Morphology, Region descriptors
  - Linear/nonlinear filter – convolution, Image pyramids.
- (3) Edge detection and image gradients
  - Image derivatives, Canny edge detector
- (4) Fitting models
  - Least-squares fitting (iterative, robust), Normal equations, Homogenous systems, RANSAC, Hough transform
- (5) Key-points and correspondences between images
  - Key-point detection in scale-space, local descriptors, SIFT

# What did we learn?

- (6,7) Cameras and stereo systems
  - Pinhole camera model, Calibration, Epipolar geometry, Dense correspondence, Triangulation, Active stereo
- (8a-d) Feature learning for recognition and detection:
  - Natural linear coordinate systems: PCA(face recognition)
  - Feature selection: Adaboost+integral images (face detection)
  - Nonlinear hand-crafted transforms: HoG+SVM (pedestrian detection)
  - End-to-end feature & classifier learning: Convolutional neural nets (CNNs)
- (9) Key-point-based recognition
  - Bag-of-words models.
  - Detection/recognition by RANSAC and Generalized Hough transform.

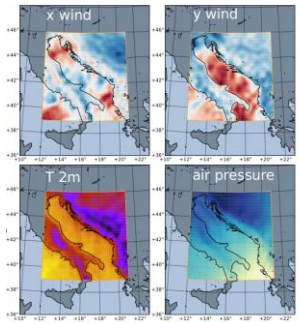
# The Next Big Thing on Your List...

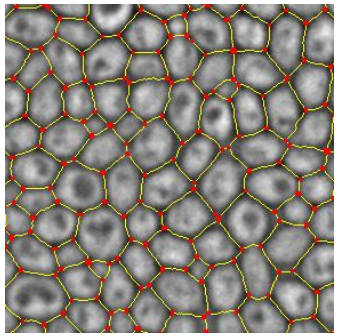
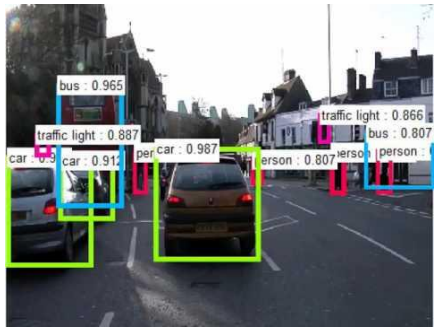
- The written exam (see Studis for dates)
  - Approx. two hours -- Covers the entire course
  - Theoretical as well as analytical assignments (see the lab exercises for examples of analytical parts)
- Oral exam potentially required for low scores (X = ~50%-60% )
  - Need to know *all that you got wrong* on written exam
  - + ~2 random questions
- If >X% do not have to come to oral
    - Can if you would like to increase/decrease grade by 1 (or fail?)
- Please fill-out the poll at *Studis*
  - Constructive suggestions towards improving the course

# Writing a Computer Vision thesis

- ViCoS lab offers a number of topics: segmentation, detection, tracking, industrial inspection, fundamental deep learning topics, climate prediction



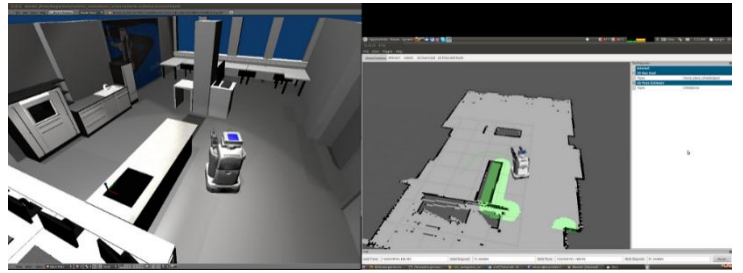


- I will be accepting candidates after the last MP exam in this semester (6.2)
  - 1 week to collect your applications, then will let you know
  - Will announce the call for applications on the eclassroom
- Caution: CV theses are typically challenging (~180 hours)
- See CVPR/ECCV/ICCV/NeurIPS for potential topics (your own ideas also welcome)

# Other Computer-vision-related courses at FRI

- Bachelor's level:
  - Multimedia Systems (Luka Čehovin, Vicos)
  - Development of Intelligent Systems (Danijel Skočaj, Vicos)



- Master's level
  - Advanced computer vision methods (Matej Kristan, Vicos)
  - Deep learning (Danijel Skočaj, Vicos)
  - Image-based biometry (Peter Peer)
  - Biomedical Signal and image Processing (Franc Jager)
  - Potentially another course on 3D perception (prepared by Vicos)

# Other courses on the web

- Check out similar courses at other Universities:
  - Aachen: https://www.vision.rwth-aachen.de/course/6/
  - Stanford: http://vision.stanford.edu/teaching/cs131_fall1617/schedule.html
  - Illinois: http://slazebni.cs.illinois.edu/spring18/
  - … many more can be found on the net

- ***Slovene terminology*** *(see translations here)**:***
  *https://terminoloski.slovenscina.eu/*

# Thanks!

Good luck with the exam(s)!