



# Introduction to Computer Vision

## Lecture 2 - Classic Vision I

Prof. He Wang

# Computer Vision (Cont'd)

# What Else in Computer Vision?

- Computer vision deals with
  - acquiring
  - processing and analyzing
  - understanding
  - generating or imagining
- visual data, and, what's more,
- joint tasks with language

# Applications: Text-to-Image Diffusion Model



A brain riding a rocketship heading towards the moon.



A dragon fruit wearing karate belt in the snow.



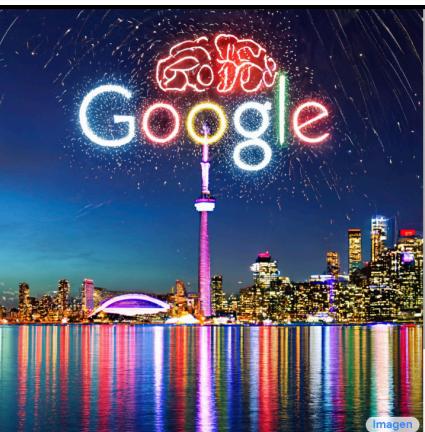
A small cactus wearing a straw hat and neon sunglasses in the Sahara desert.



A photo of a Corgi dog riding a bike in Times Square. It is wearing sunglasses and a beach hat.



A blue jay standing on a large basket of rainbow macarons.



The Toronto skyline with Google brain logo written in fireworks.



A bucket bag made of blue suede. The bag is decorated with intricate golden paisley patterns. The handle of the bag is made of rubies and pearls.



A single beam of light enter the room from the ceiling. The beam of light is illuminating an easel. On the easel there is a Rembrandt painting of a raccoon.

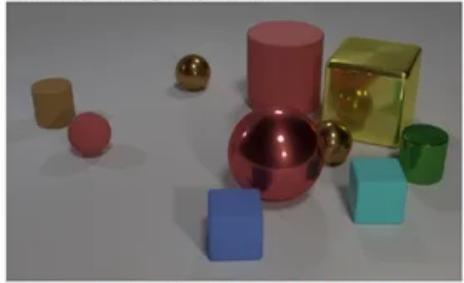
Imagen: <https://imagen.research.google/>

# More Vision Language Tasks

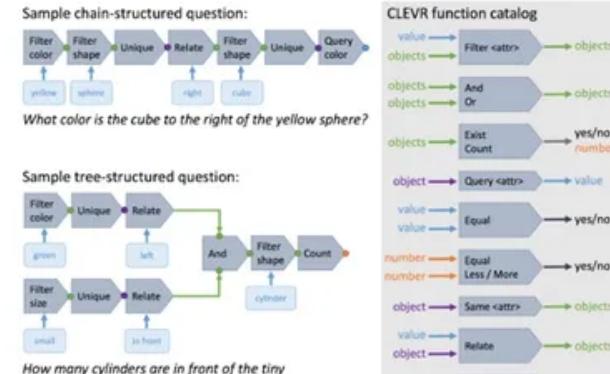
- Referring Expression
  - ReferIt Game, *EMNLP 2014*
  - RefCOCO, *ECCV 2016*
  - GuessWhat?!, *CVPR 2017*
- Visual Dialog
  - VisDial, *CVPR 2017*
  - Image Grounded Conversation, *ACL 2017*
  - Dialog-based Image Retrieval, *NIPS 2018*
- Text 2 image/video



Questions in CLEVR test various aspects of visual reasoning including **attribute identification**, **counting**, **comparison**, **spatial relationships**, and **logical operations**.



- Q: Are there an **equal number** of **large things** and **metal spheres**?  
 Q: **What size** is the **cylinder** that is **left** of the **brown metal** thing that is **left** of the **big sphere**?  
 Q: There is a **sphere** with the **same size** as the **metal cube**; is it **made of the same material** as the **small red sphere**?  
 Q: **How many** objects are **either small cylinders** or **red things**?



Johnson, Justin, Bharath Hariharan, Laurens van der Maaten, Fei Fei Li, C. Lawrence Zitnick, and Ross Girshick. "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning." In *CVPR*. 2017.

## Intersection between computer vision and natural language processing (NLP)

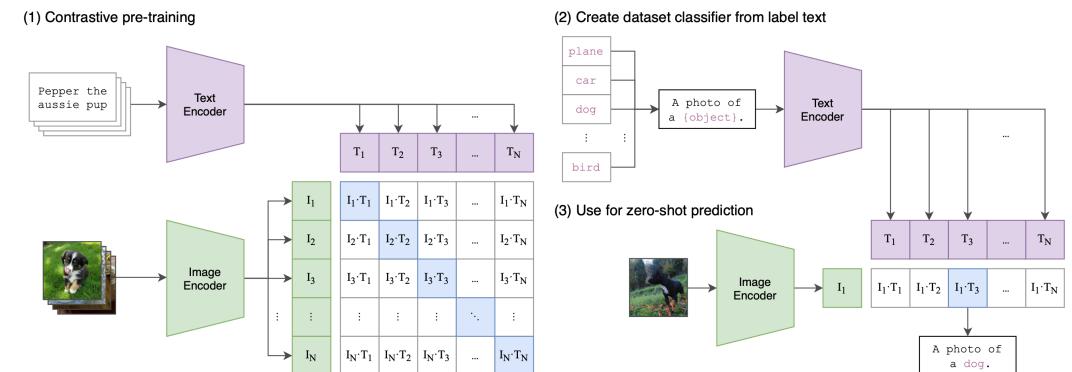


Figure 1. Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

# Large Vision-Language Model

## Sec. 2.1 Time understanding throughout the day

### Prompt:

This is the front-view image of your car. Please describe the current traffic scenario you're in.



### GPT-4V:

You're on a city street with multiple lanes. There are parked cars on both sides of the road. Ahead, there's moderate traffic with vehicles moving in both directions. There's also a pedestrian crossing zone visible. Buildings and trees line the street, and there are various street signs along the way.

### Prompt:

This is the front-view image of your car. Please describe the current traffic scenario you're in.



### GPT-4V:

You are on a road during twilight or early evening. The road has a pedestrian crossing ahead. There's a vehicle with its tail lights on, indicating it's either stationary or moving away from you. There are traffic signs on both sides of the road, and the area appears to be moderately lit. Trees and other structures are visible in the background.

# What Else in Computer Vision?

- Computer vision deals with
  - acquiring
  - processing and analyzing
  - understanding
  - generating or imagining visual data, and, what's more,
    - joint tasks with language
    - providing visual feedbacks for body motions for **embodied** agents.

# Embodied AI and Embodied Vision

- An embodied intelligent agent has its own physical body and can actively perceive and interact with the physical world.

- 👁️ **See:** perceive their environment through vision or other senses.
- 🎤 **Talk:** hold a natural language dialog grounded in their environment.
- 🎧 **Listen:** understand and react to audio input anywhere in a scene.
- 🤖 **Act:** navigate and interact with their environment to accomplish goals.
- 🤔 **Reason:** consider and plan for the long-term consequences of their actions.

Embodied AI is the field for solving AI problems for virtual robots that can move, see, speak, and interact in the virtual world and with other virtual robots — these simulated robot solutions are then transferred to real world robots.

--- Luis Bermudez, Overview of Embodied Artificial Intelligence



Home robots

# Applications: Autonomous Driving



Tesla: pure vision solution  
No LiDAR but multi-cameras

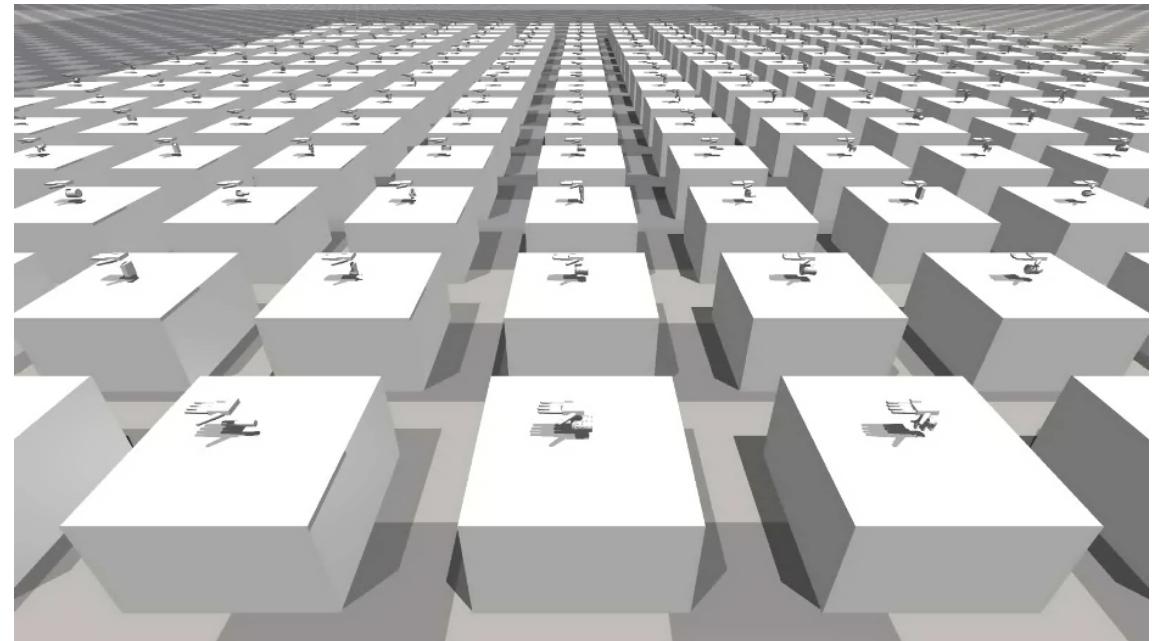


Waymo  
LiDAR-based Solution

# Applications: Embodied Vision

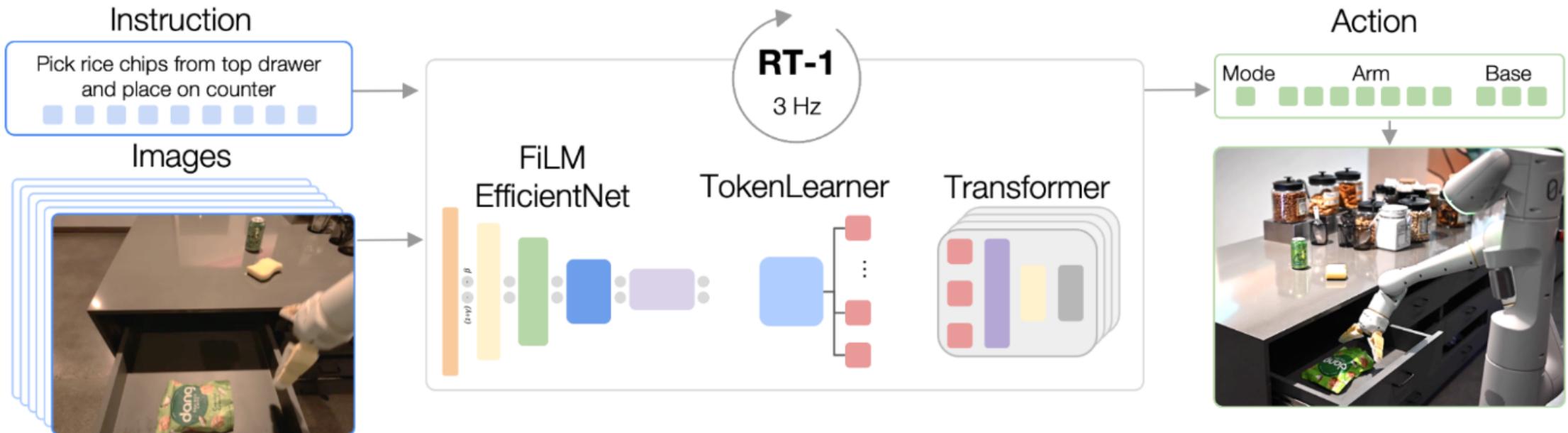


ASGrasp (ICRA 2024)



Dexterous hand grasping (ICCV 2023  
Best paper finalist)

# Embodied Vision-Language Model: RT-1



*RT-1's architecture: The model takes a text instruction and set of images as inputs, encodes them as tokens via a pre-trained FiLM EfficientNet model and compresses them via TokenLearner. These are then fed into the Transformer, which outputs action tokens.*

# Embodied Artificial General Intelligence (AGI)



Everyday robots from Google



Tesla Bot

# Summary of Computer Vision

- Compared to human vision, computer vision deals with the following tasks:
  - visual **data acquisition** (similar to human eyes but comes with many more choices)
  - signal processing and feature extraction (mostly **low-level**)
  - analyze local structures and then 3D reconstruct the original scene (**mid-level**)
  - understanding (mostly **high-level**)
  - generation
  - vision-language tasks
  - and further serving **embodied agents** to make decisions and take actions.

# A Brief History to Computer Vision

# The Birth of Artificial Intelligence



Alan Turing and Turing test

1950, Turing wrote the article “*Computing machinery and intelligence*”, in which he described what would become known as the “**Turing Test**”.

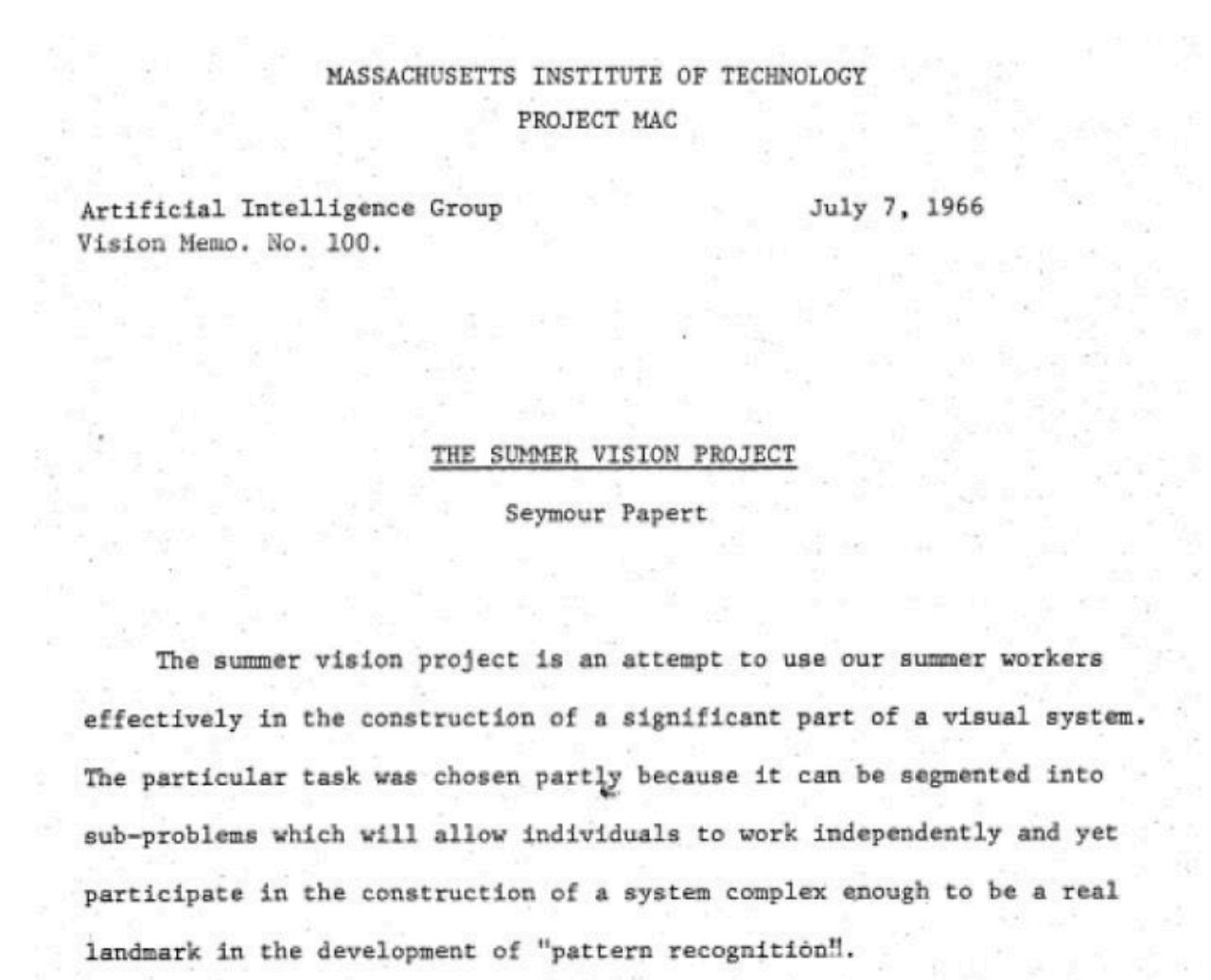


The Dartmouth Conference

August 1956. From left to right: Oliver Selfridge, Nathaniel Rochester, Ray Solomonoff, Marvin Minsky, Trenchard More, John McCarthy, Claude Shannon.

# Early in 1960s: CV as a Summer Project

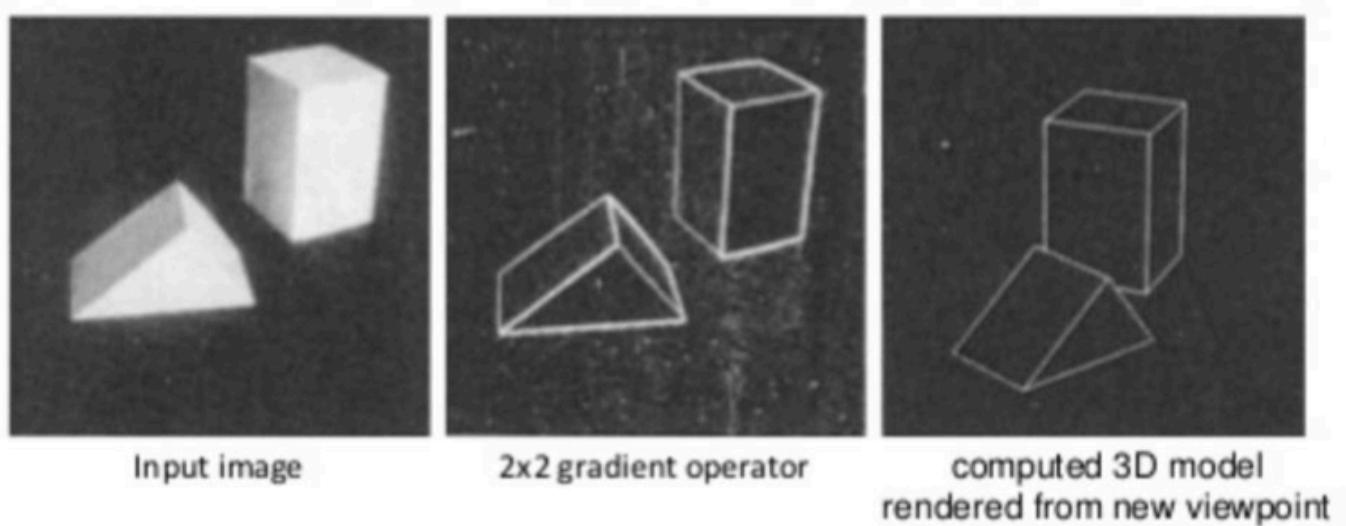
- A visual perception component of an ambitious agenda to mimic human intelligence.
- AI pioneers believed that solving the “visual input” problem would be easier than solving higher-level reasoning and planning.
- Marvin Minsky at MIT asked his undergrad Gerald Jay Sussman to “spend the summer linking a camera to a computer and getting the computer to describe what it saw”. *However, we know this is not that easy.*



# Early in 1960s: Interpretation of Synthetic Objects



Ph.D. thesis "Machine Perception of Three-Dimensional Solids"

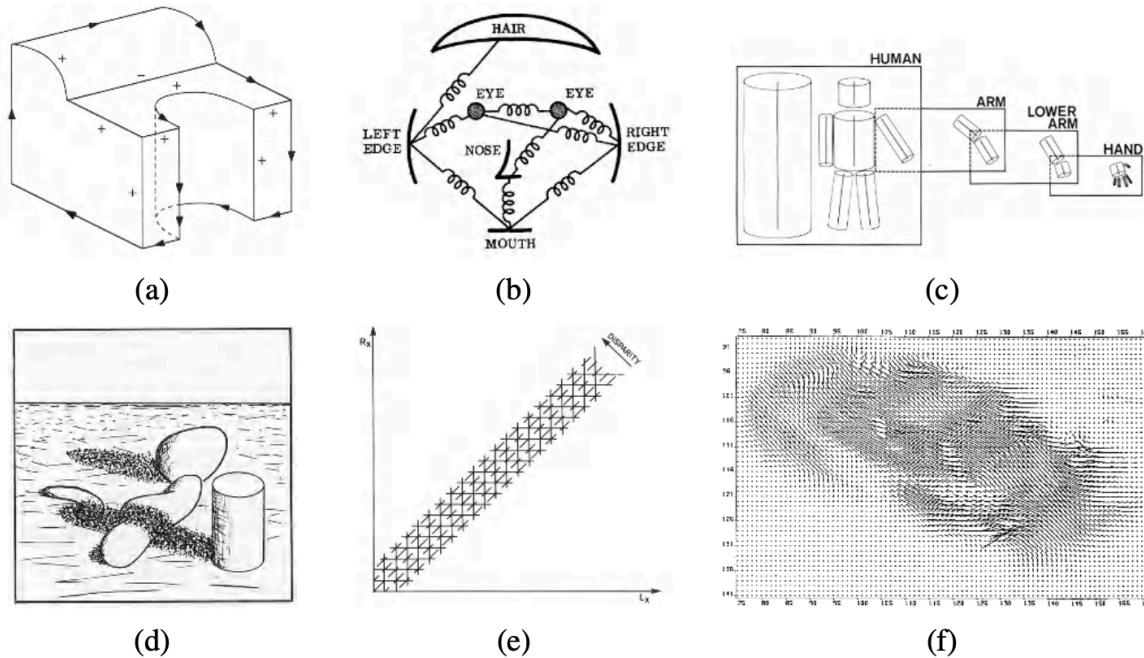


Larry Roberts  
1963, 1<sup>st</sup> thesis of Computer Vision

Borrowed from Stanford CS231N Lecture 01.

# 1970s/1980s: reconstruction as the first step

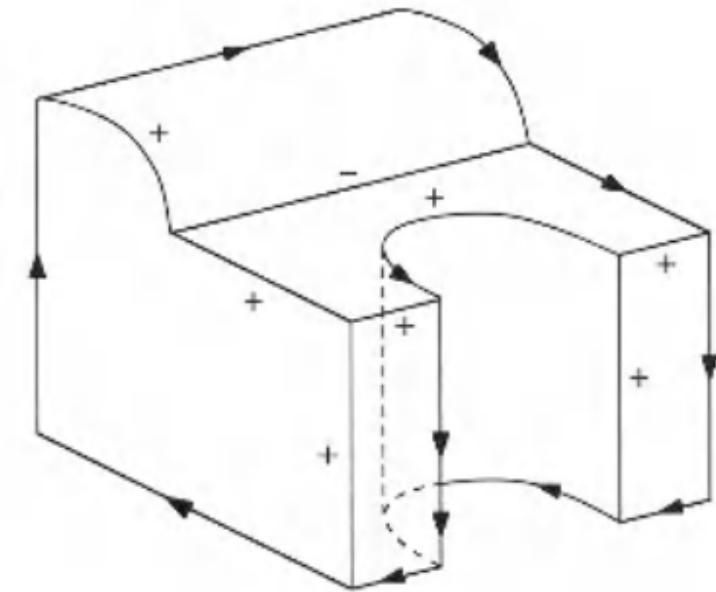
- What distinguished computer vision from the already existing field of digital image processing:
  - the desire to recover the three-dimensional structure of the world from images
  - And use this as a stepping stone to- wards full scene understanding



**Figure 1.7** Some early (1970s) examples of computer vision algorithms: (a) line labeling (Nalwa 1993) © 1993 Addison-Wesley, (b) pictorial structures (Fischler and Elschlager 1973) © 1973 IEEE, (c) articulated body model (Marr 1982) © 1982 David Marr, (d) intrinsic images (Barrow and Tenenbaum 1981) © 1973 IEEE, (e) stereo correspondence (Marr 1982) © 1982 David Marr, (f) optical flow (Nagel and Enkelmann 1986) © 1986 IEEE.

# Basic Ideas

- Extracting edges and then inferring the 3D structure of an object or a “blocks world” from the topological structure of the 2D lines

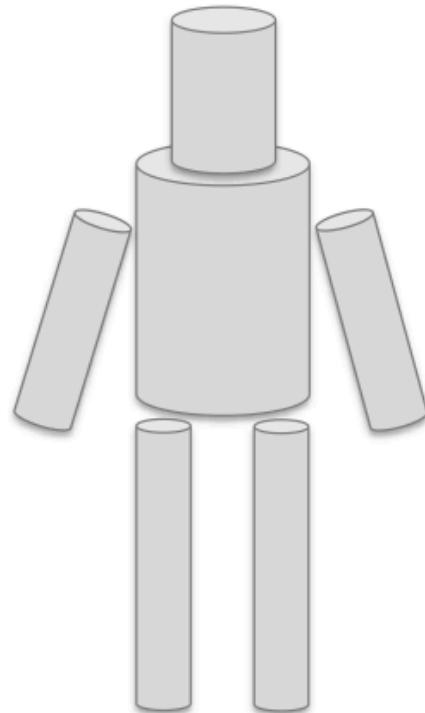


Line labeling (Nalwa 1993)

# Three-dimensional Modeling of Non-polyhedral Objects

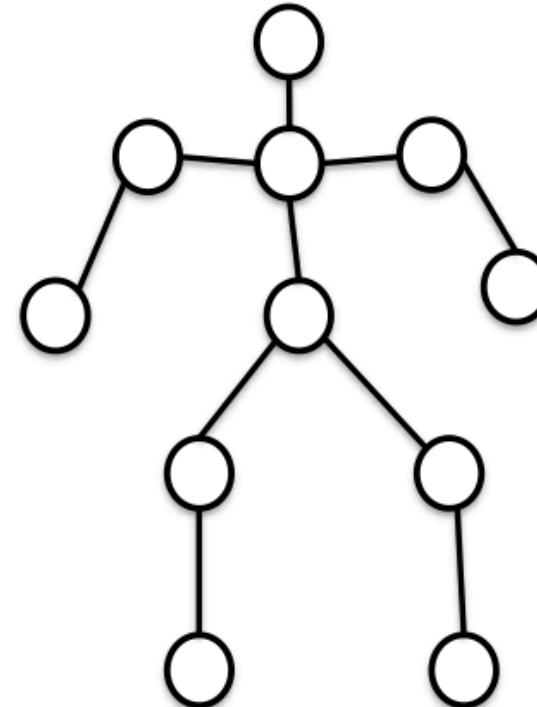
- Generalized Cylinder

Brooks & Binford, 1979



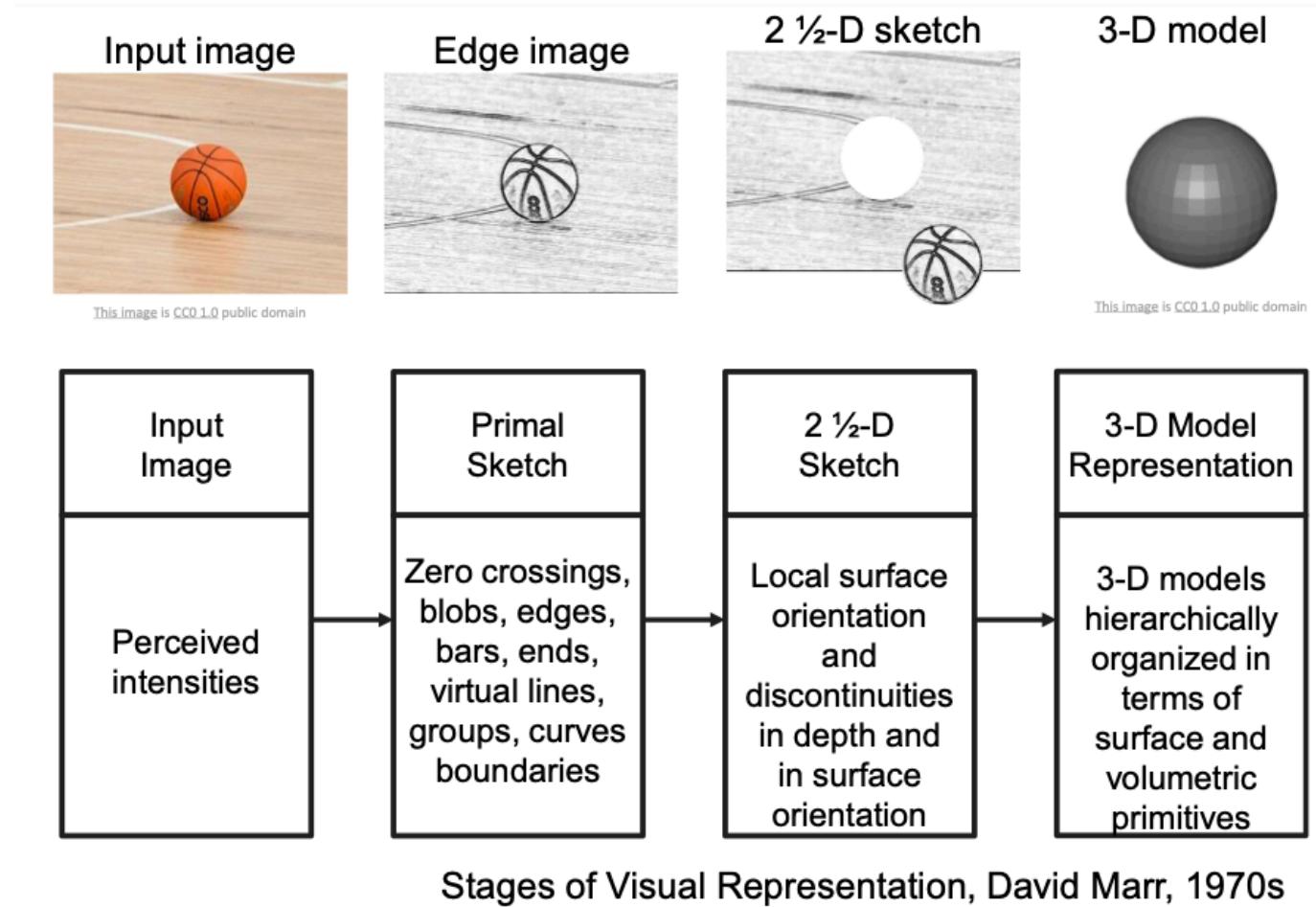
- Pictorial Structure

Fischler and Elschlager, 1973



# David Marr's 2.5-D Sketch

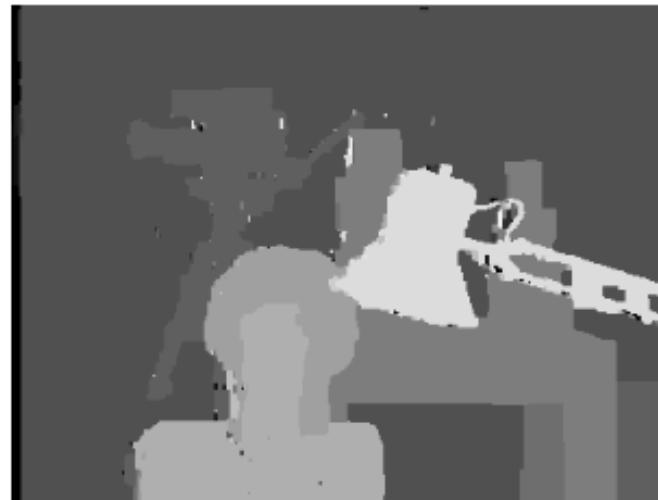
- 2.5-D Sketch:
  - A surface based representation that bridges 2D and 3D
  - Depth-from-X: computed from a 2-D image-based representation (primal sketch) via extracting information about
    - surface orientation
    - depth from a variety of sources, such as shading, stereo, and motion.



# 3D Reconstruction



Structure from Motion  
(Tomasi and Kanade 1992 )

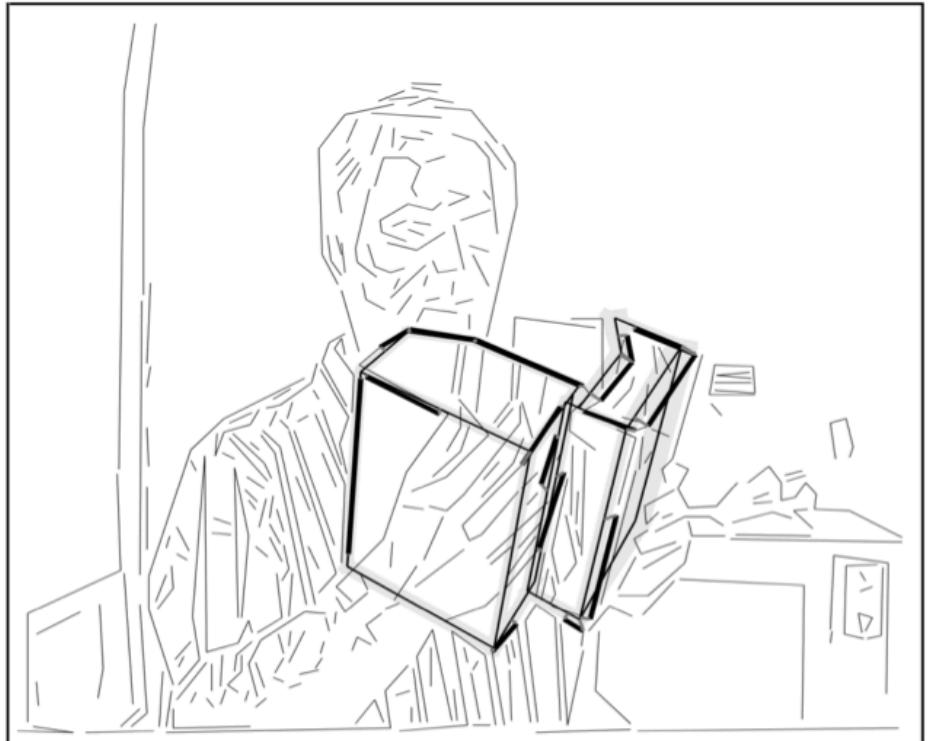


Dense stereo matching  
(Boykov, Veksler, and Zabih  
2001)



Multi-view reconstruction  
(Seitz and Dyer 1999)

# Recognition and Segmentation



D. Lowe. IJCV, 1992



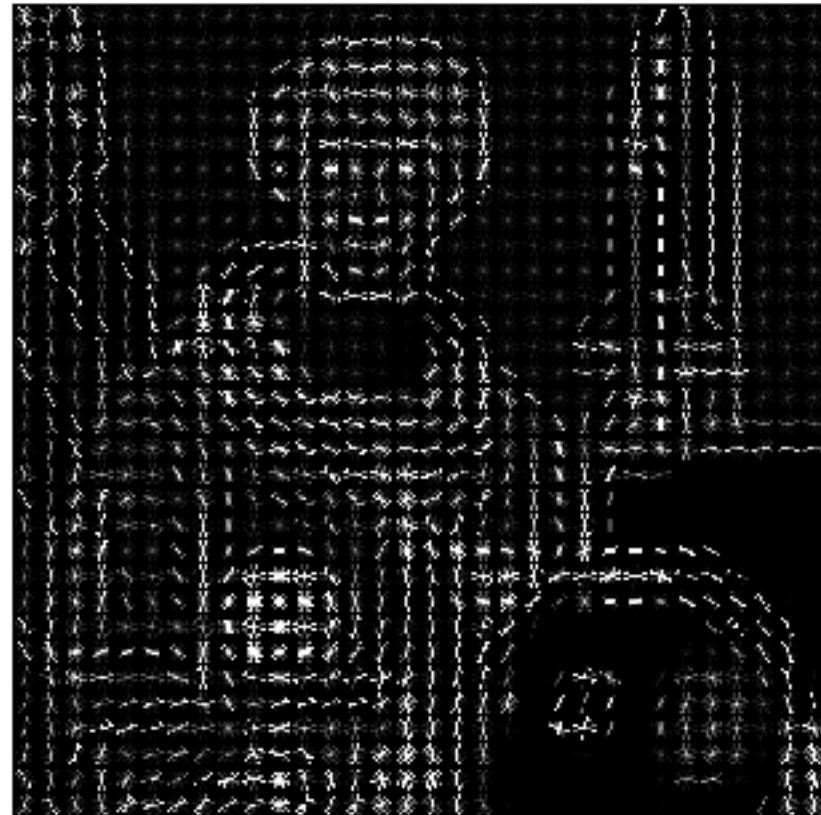
Normalized Cut (Shi & Malik, 1997)

# Descriptors

Input image



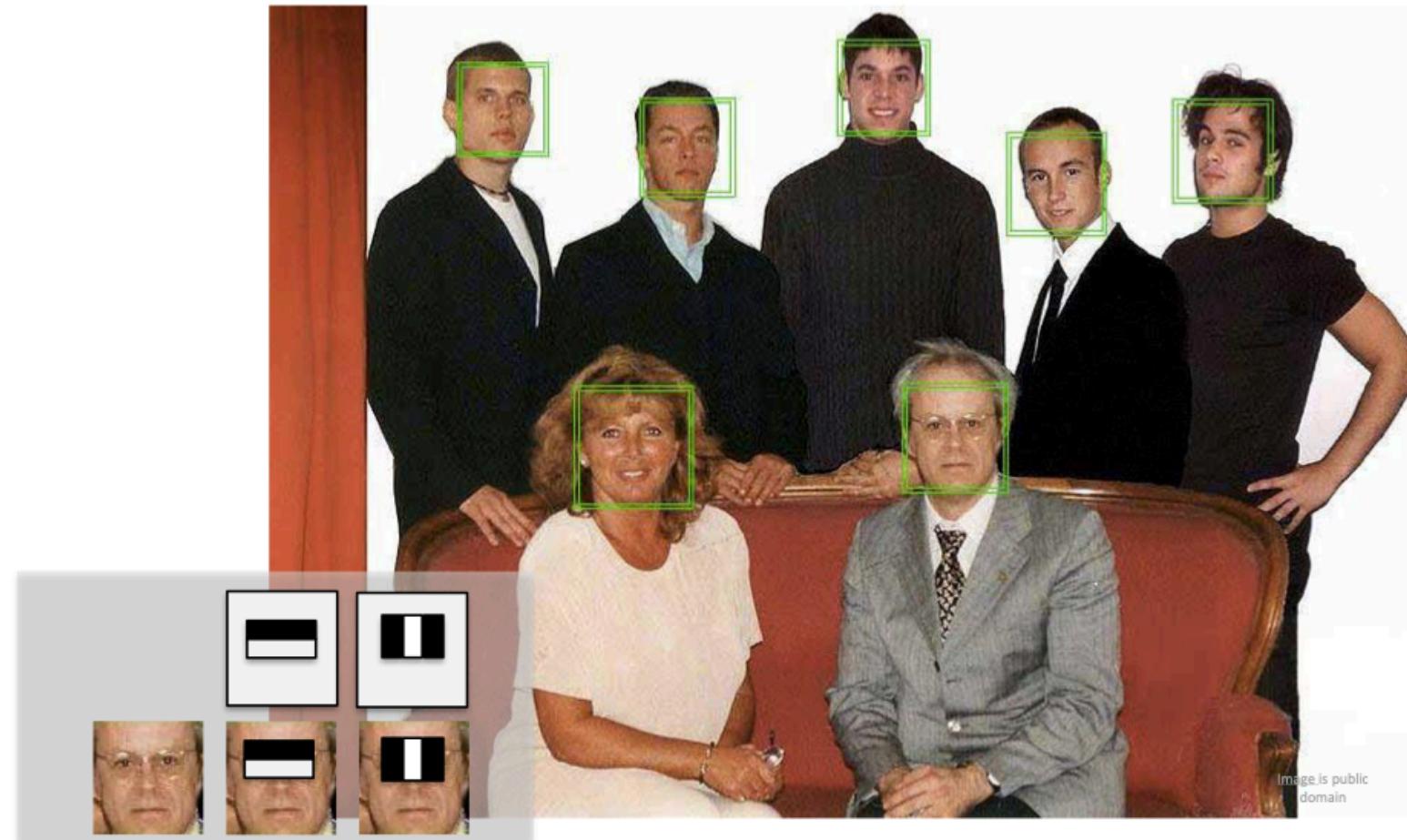
Histogram of Oriented Gradients



Histogram of Gradients (HoG) Dalal & Triggs, 2005

Credit: <https://iq.opengenus.org/object-detection-with-histogram-of-oriented-gradients-hog/>

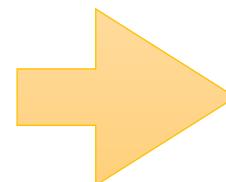
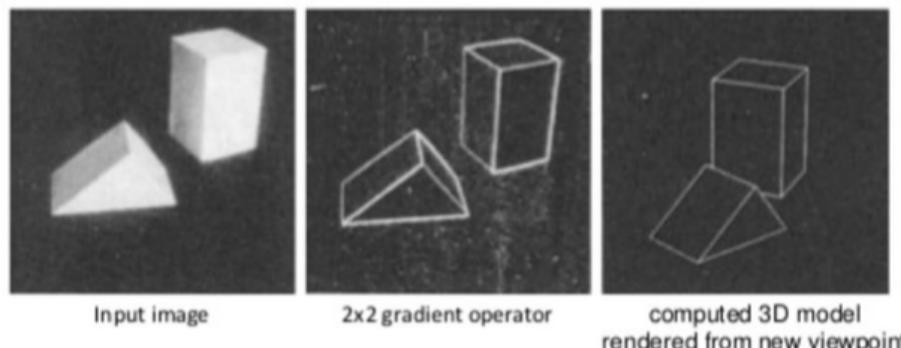
# Detection



Face Detection, Viola & Jones, 2001

# CV from the Classic Era to the Deep Learning Era

- Previous works don't leverage learning.
- However, many techniques and concepts proposed by them are still foundations for modern computer vision.
- Current trend:
  - From non-learning based method to **learning-based method**
  - Rely on **big data**
  - Requires more **computation resources**.



# Algorithm: Deep Learning



2018 Turing Awards: Geoffrey Hinton, Yann LeCun, and Yoshua Bengio

# Data: ImageNet and Its Benchmark



IMAGENET

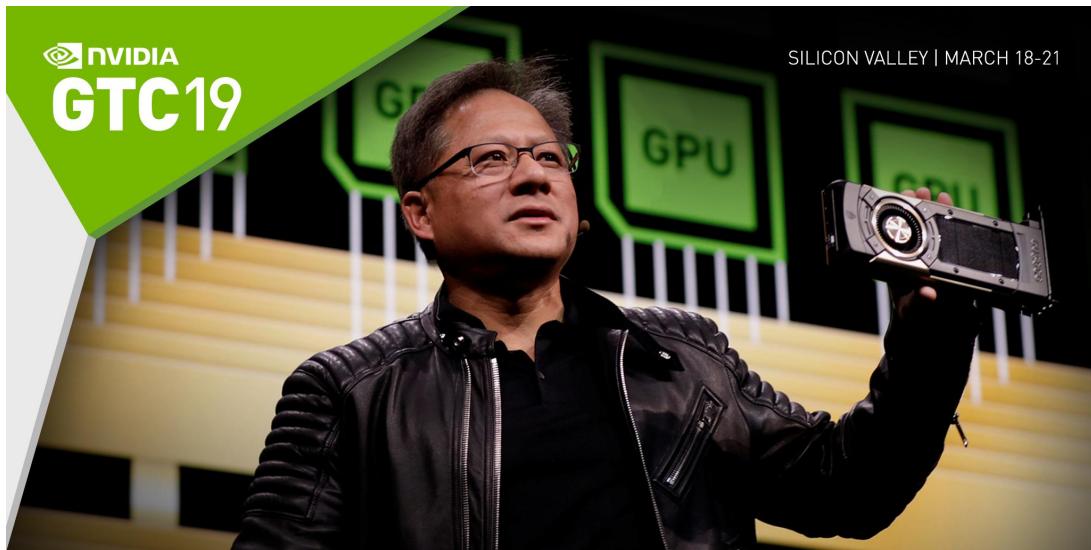
22,000 categories

15,000,000 images

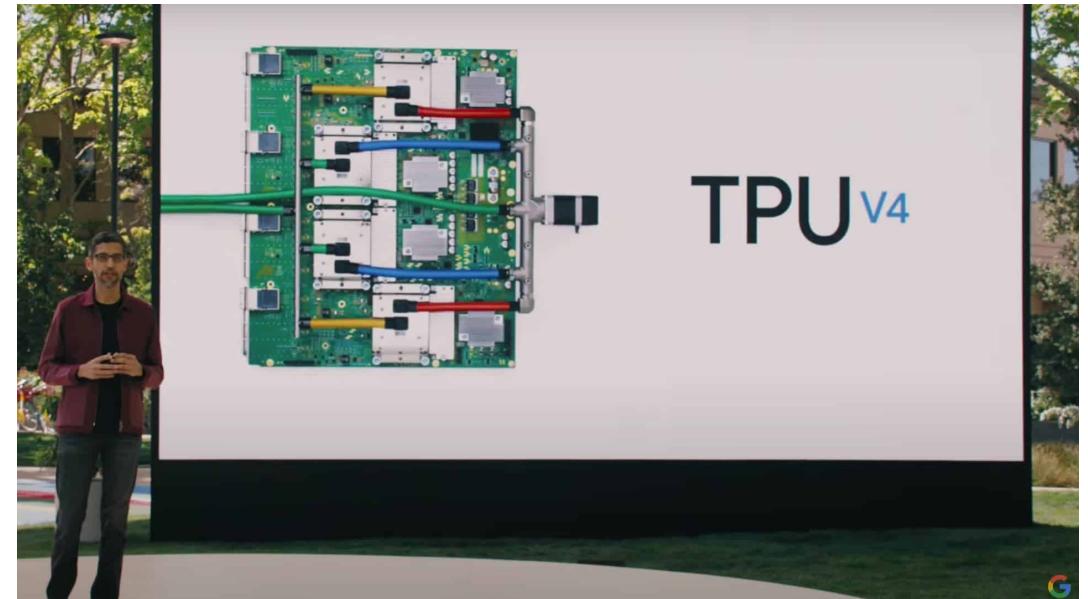


J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li & L. Fei-Fei. CVPR, 2009.

# Computational Resources: GPU



NVIDIA and its GPU



Google and its TPU

# Deep Learning Frameworks and Opensource

PyTorch



facebookresearch / Detectron Public

Code Issues 305 Pull requests 22 Actions Projects Security Insights

main 2 branches 0 tags Go to file Add file Code About

generatedunixname89002005287564 and facebook-github-bot fbcde//v... e8ebaf7 18 hours ago 142 commits

.github Add template for github issues 5 years ago

FAIR's research platform for object detection research, implementing popular algorithms like Mask R-CNN and RetinaNet.

OpenMMLab Codebase Ecosystem Open Platform Community About Us Log in

Officially Endorsed Projects Projects that are officially endorsed by OpenMMLab with high quality.

**MMEngine** NEW A foundational library for training deep learning models

- Universal and powerful runner
- Open architecture with unified interfaces
- Customizable training process

Deep Learning Computer Vision

**MMCV** A foundational library for computer vision.

- Universal IO APIs
- Image/Video/OpticalFlow processing
- High-quality implementation of CUDA ops
- High-level training APIs for PyTorch

Computer Vision

**MMDetection** Object detection toolbox and benchmark.

- Modular Design
- Support of multiple frameworks out of the box
- High efficiency

Object Detection Instance Segmentation

# Course Organization

# Course Organization: Low-Level Vision and Deep Learning

Lecture	<b>02/21/2024 15:10 Wednesday</b>	Course Introduction	Logistic, Introduction, History
Lecture	<b>02/28/2024 15:10 Wednesday</b>	Classic Vision I	Image processing: image gradient, filter, convolution. Classic edge/corner/line detection methods: Canny edge detection, Harris corner detection, line fitting.
Lecture	<b>03/06/2024 15:10 Wednesday</b>	Classic Vision II and Deep Learning I	Edge/line detection: Canny edge detection, line fitting(RANSAC, Hough transform). Corner detection and feature descriptors: Harris corner detection, feature. Outline of deep learning.
Lecture	<b>03/13/2024 15:10 Wednesday</b>	Deep Learning II	Outline of deep learning. Network structure: Multi-Layer Perceptron (MLP) and Convolutional Neural Network (CNN).
Lecture	<b>03/20/2024 15:10 Wednesday</b>	Deep Learning III	CNN Training: weight initialization, loss, optimizer (gradient descent, SGD, Adam), learning rate. Underfitting and overfitting: batch normalization, skip link, augmentation, regularization
Lecture	<b>03/27/2024 15:10 Wednesday</b>	Deep Learning IV	Classification task: KNN, SoftMax, cross entropy loss, receptive field, architecture(VGG, ResNet...) Segmentation task: K-Means, upsampling, architecture (FCN, UNet), IoU

(Tentative schedule)

For the complete and up-to-date schedule, see [https://pku-epic.github.io/Intro2CV\\_2024/schedule/](https://pku-epic.github.io/Intro2CV_2024/schedule/).

# Course Organization: Mid-Level and 3D Vision

Lecture	<b>04/03/2024 15:10 Wednesday</b>	3D Vision I	Camera model: pinhole camera, lense, intrinsic, extrinsic, calibration Depth image: sensor, stereo
Lecture	<b>04/10/2024 15:10 Wednesday</b>	3D Vision II (3D Deep Learning)	3D data (voxel, mesh); 3D Deep Learning: Point, Mesh; 3D Deep Learning: Sparse Voxel Conv
<b>Exam</b>	<b>04/17/2024 15:10 Wednesday</b>	<b>Midterm Exam</b>	
Lecture	<b>04/17/2024 17:10 Wednesday</b>	Temporal Analysis I	Motion and optical flow
Lecture	<b>04/24/2024 15:10 Wednesday</b>	Temporal Data Analysis II	RNN, LSTM, GRU; Video Analysis: 3D CNN, Two-stream network, ConvRNN
Peking University Anniversary	05/01/2024 Wednesday	One Week Break (Labor Day Holiday)	
Lecture	<b>05/08/2024 15:10 Wednesday</b>	Object Detection and Instance Segmentation	2D Object detector (SSD, RCNN series, YOLO); Instance Segmentation, Panoptic Segmentation; 3D object detection and instance segmentation

(Tentative schedule)

For the complete and up-to-date schedule, see [https://pku-epic.github.io/Intro2CV\\_2024/schedule/](https://pku-epic.github.io/Intro2CV_2024/schedule/).

# Course Organization: High-Level Vision

Lecture	<b>05/15/2024 15:10 Wednesday</b>	Guest Lecture by Dr. Zhizheng Zhang	Attention mechanism, transformer, visual transformer, vision language pretraining, large language model, vision language model.
Lecture	<b>05/22/2024 15:10 Wednesday</b>	Generative Model I	PixelRNN/CNN, VAE, GAN.
Lecture	<b>05/29/2024 15:10 Wednesday</b>	Generative Model II (Diffusion model)	text-to-image generation, diffusion-based vision applications
Lecture	<b>06/05/2024 15:10 Wednesday</b>	Embodied AI	Object grasping, object manipulation, locomotion and navigation, embodied vision-language models.
<b>Exam</b>	<b>06/19/2024 13:59 Wednesday</b>	<b>Finalterm Exam</b>	

(Tentative schedule)

For the complete and up-to-date schedule, see [https://pku-epic.github.io/Intro2CV\\_2024/schedule/](https://pku-epic.github.io/Intro2CV_2024/schedule/).

# What Does Computer Vision?

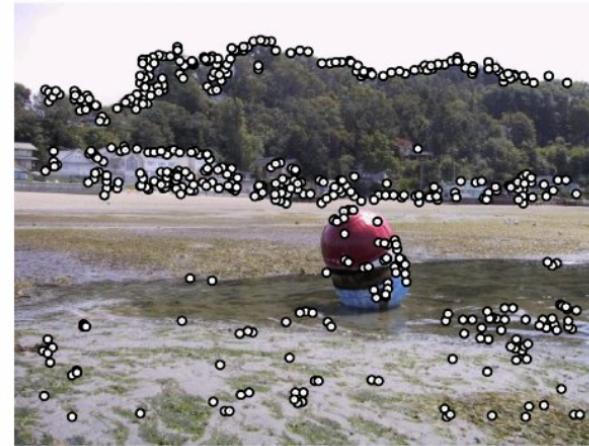
- Computer vision deals with
    - acquiring
    - **processing (today's topic)**
    - analyzing
    - understanding
    - generating or imagining
- visual data**, and, what's more,
- providing visual feedbacks for body motions
  - helping making decisions
- for embodied agents.

# Today's Topic

- **Low-level vision**
  - Image processing
  - Edge/corner detection
  - Feature extraction
- Mid-level vision
  - Grouping
  - Inferring scene geometry (3D reconstruction)
  - Inferring camera and object motion
- High-level vision
  - Object recognition
  - Scene understanding
  - Activity understanding

# Outline of Today's Lecture

- Images as functions
- Classic (non-learning) methods
  - Edge detectors
  - Corner detectors
  - Line fitting



Adaptive non-maximal suppression (ANMS) (Brown, Szeliski, and Winder 2005)

<https://medium.com/@realderektan/self-driving-car-project-part-1-lane-lines-detector-6d960e2b023>

# Images as Functions

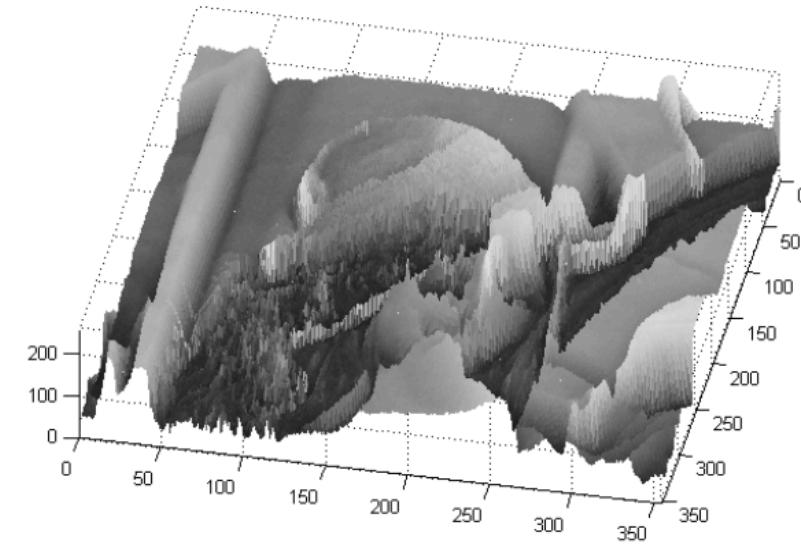
# Images as Functions

- An **Image** as a function  $f$  from  $\mathbb{R}^2$  to  $\mathbb{R}^M$ :
  - $f(x, y)$  gives the **intensity** at position  $(x, y)$
  - Defined over a rectangle, with a finite range:

$$f: [a,b] \times [c,d] \rightarrow [0,255]$$

Domain  
support

range



# Images as Functions

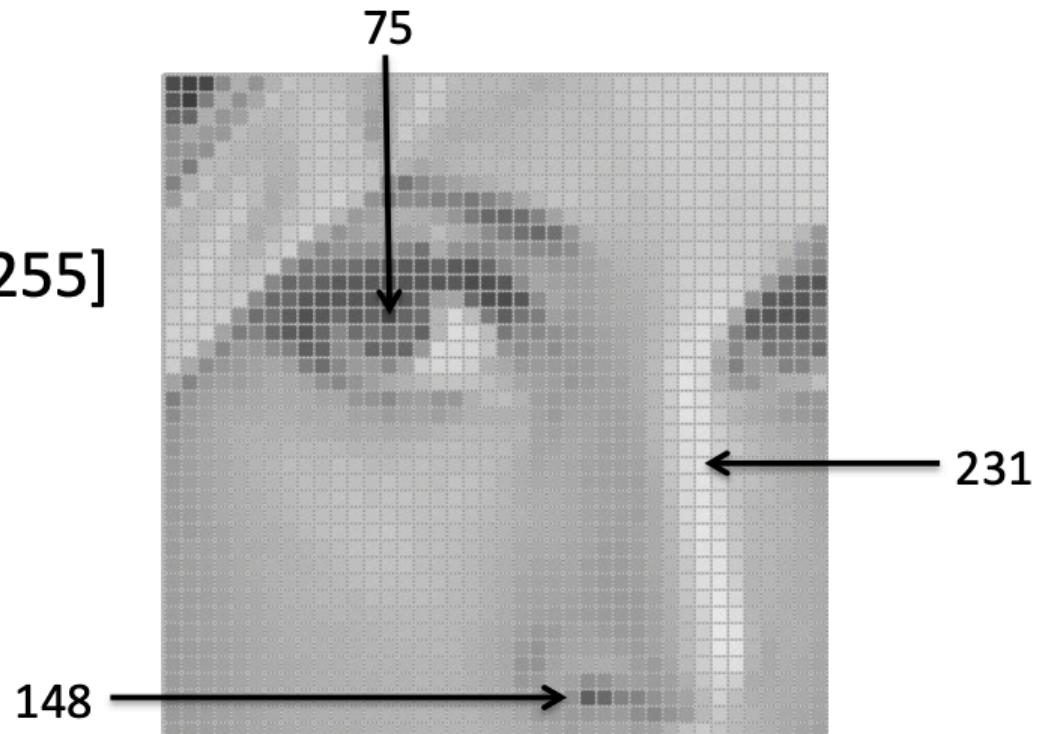
- An Image as a function  $f$  from  $\mathbb{R}^2$  to  $\mathbb{R}^M$ :
  - $f(x, y)$  gives the **intensity** at position  $(x, y)$
  - Defined over a rectangle, with a finite range:

$$f: \underbrace{[a,b] \times [c,d]}_{\text{Domain support}} \rightarrow \underbrace{[0,255]}_{\text{range}}$$

- A color image:  $f(x, y) = \begin{bmatrix} r(x, y) \\ g(x, y) \\ b(x, y) \end{bmatrix}$

# Images as Functions

- An image contains discrete number of pixels
  - A simple example
  - Pixel value:
    - “grayscale”  
(or “intensity”):  $[0, 255]$
    - “color”
      - RGB:  $[R, G, B]$



# Images as Functions

- Images are usually **digital (discrete)**:
  - Sample the 2D space on a regular grid
- Represented as a matrix of integer values

pixel

62	79	23	119	120	05	4	0	
10	10	9	62	12	78	34	0	
10	58	197	46	46	0	0	48	
176	135	5	188	191	68	0	49	
2	1	1	29	26	37	0	77	
0	89	144	147	187	102	62	208	
255	252	0	166	123	62	0	31	
166	63	127	17	1	0	99	30	

*i*      *j*

A 9x9 grid of integer values representing pixels. The value at position (i, j) is highlighted with a red box and labeled 'pixel'. A red arrow points from the text 'pixel' to this highlighted cell.

Cartesian coordinates

$$f[n, m] = \begin{bmatrix} \ddots & & & & \\ & f[-1, 1] & f[0, 1] & f[1, 1] & \\ \dots & f[-1, 0] & \underline{f[0, 0]} & f[1, 0] & \dots \\ & f[-1, -1] & f[0, -1] & f[1, -1] & \\ & & \vdots & & \ddots \end{bmatrix}$$

Notation for discrete functions

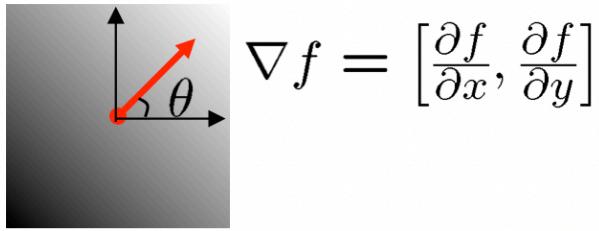
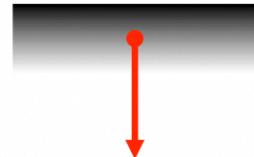
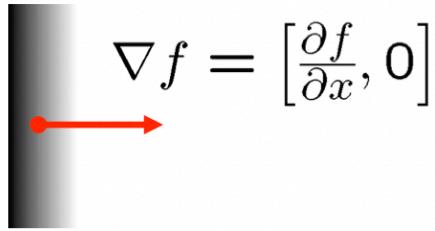
The diagram illustrates the mapping of a 2D image to a function  $f[n, m]$ . It shows a 3x3 grid of values with indices  $n$  (vertical) and  $m$  (horizontal). The value at index  $(0, 0)$  is underlined to indicate it is the current point of interest. The grid is enclosed in a large bracket, representing the function's domain. Arrows show the correspondence between the grid elements and the function notation.

# Image Gradient

- Image as a function:
- Image gradient:

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

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

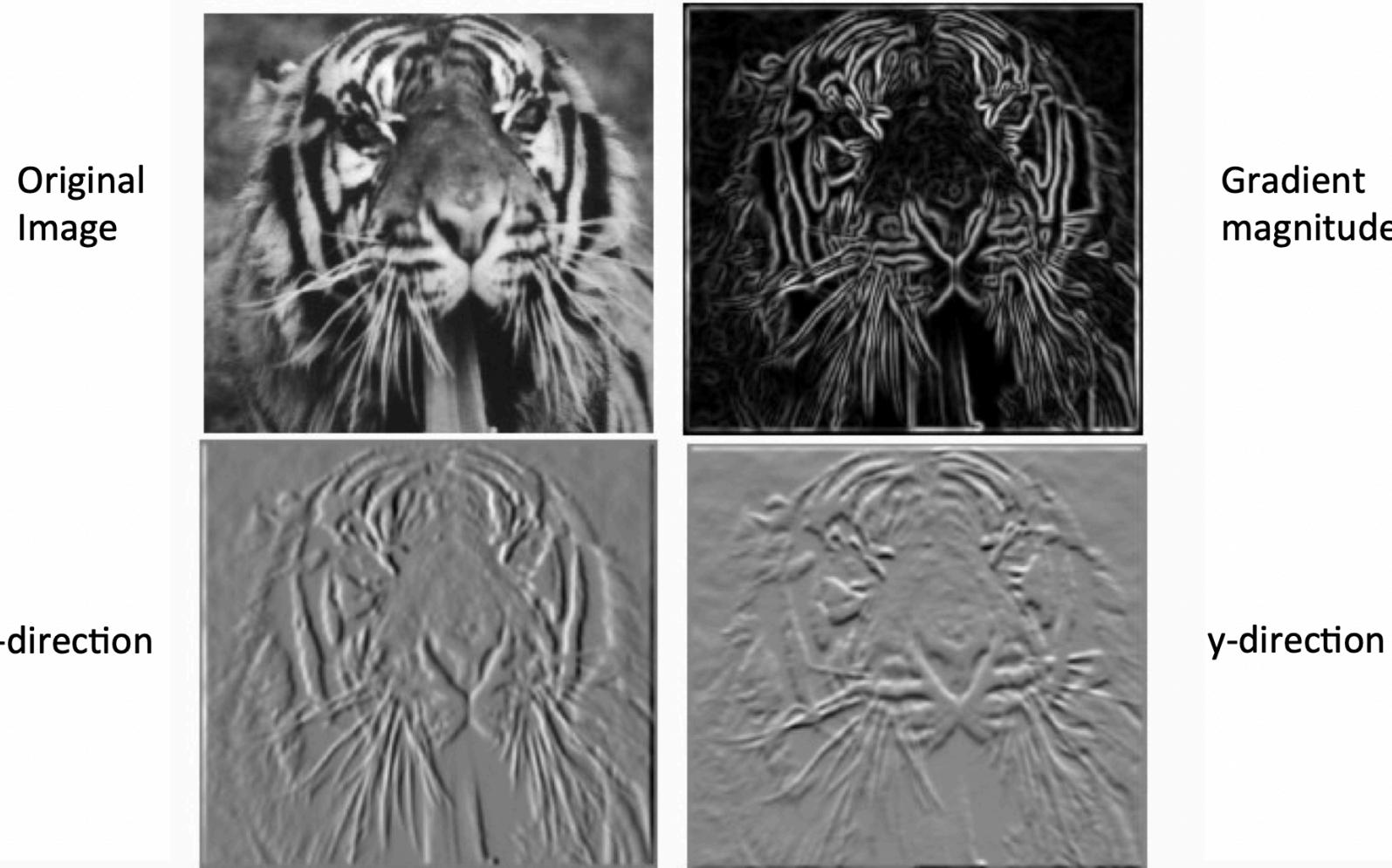


- In practice, use **finite difference** to replace gradient.

- $\frac{\partial f}{\partial x} \Big|_{x=x_0} \approx \frac{f(x_0 + 1, y_0) - f(x_0 - 1, y_0)}{2}$

- The image gradient points in the direction of the most rapid change in intensity.

# Visualizing Image Gradient



Source: Feifei Li

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

# Filters

- **Filtering:**
  - Form a new image whose pixels are a combination original pixel values

## Goals:

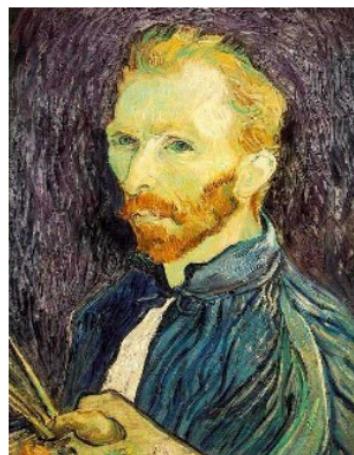
- Extract useful information from the images
  - Features (edges, corners, blobs...)
- Modify or enhance image properties:
  - super-resolution; in-painting; de-noising

De-noising



Salt and pepper noise

Super-resolution



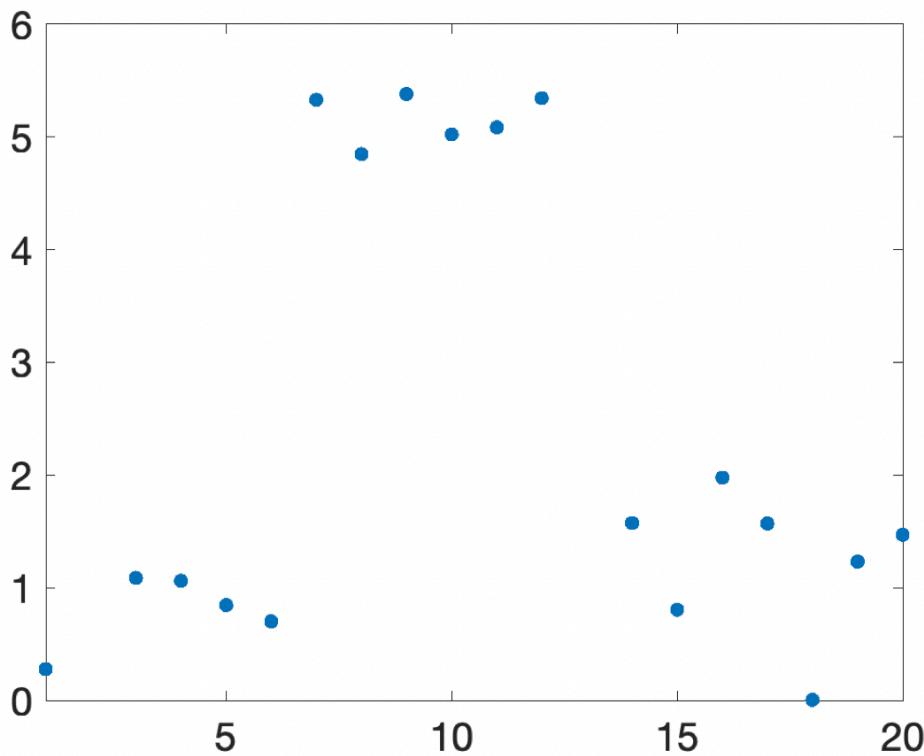
# 1D Discrete-Space Systems (Filters)

$$f[n] \rightarrow \boxed{\text{System } \mathcal{G}} \rightarrow h[n]$$

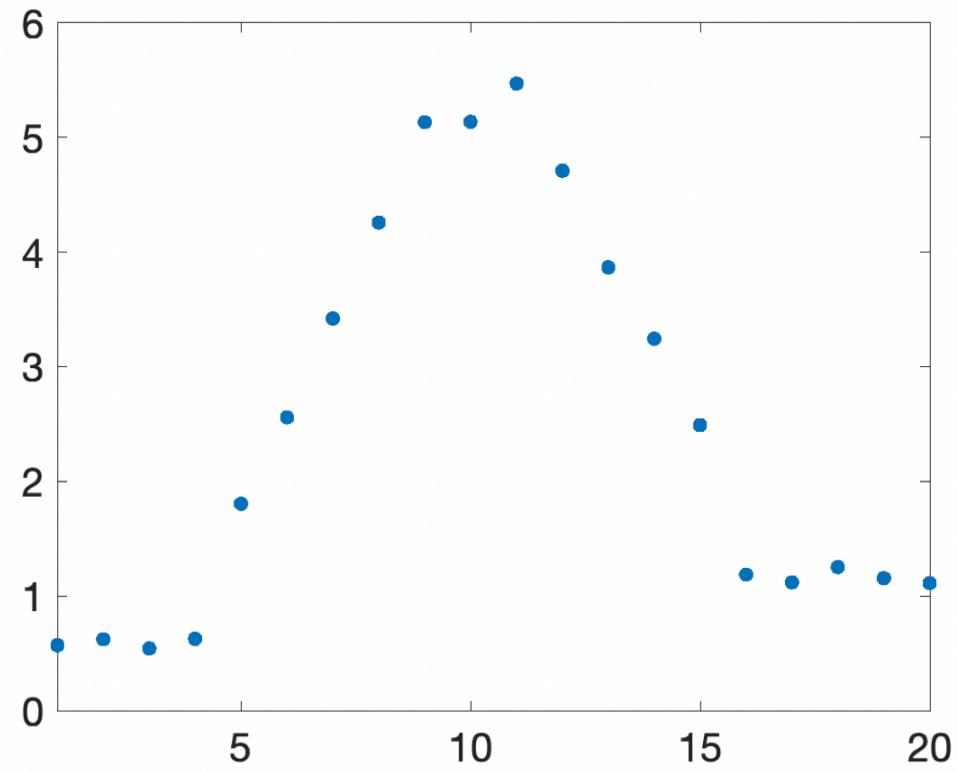
$$h = \mathcal{G}(f), \quad h[n] = \mathcal{G}(f)[n]$$

# 1D Filter Example: Moving Average

Original data  $f[n]$

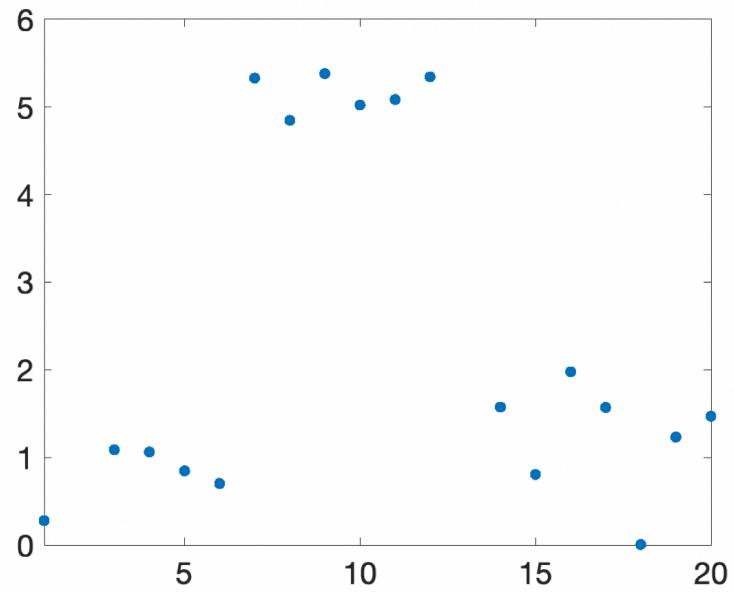


After moving average with window size = 5,  
 $h[n]$

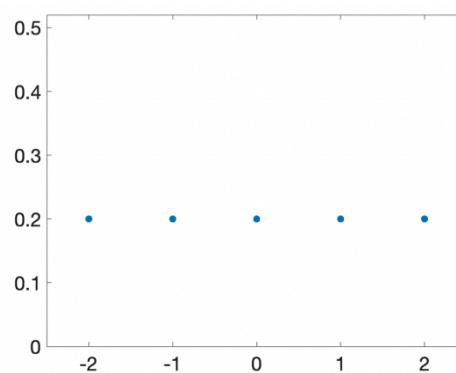


# 1D Filter Example: Moving Average

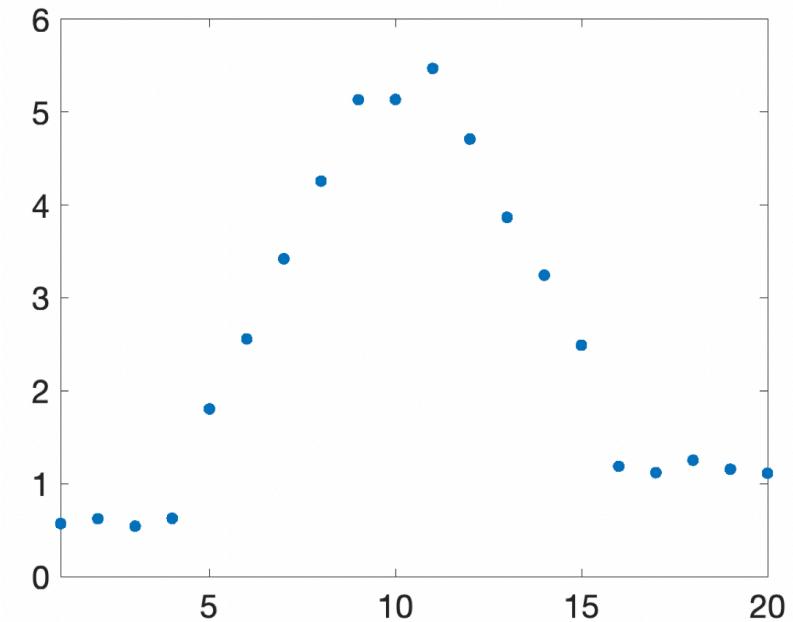
Original data  $f[n]$



Weight function  
(equal weight)



After moving average,  
 $h[n]$

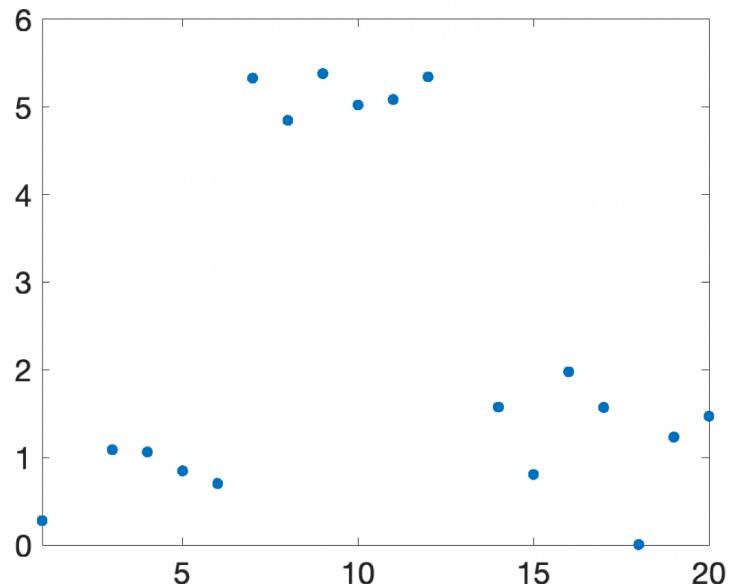


Let's use the language of image or signal processing!

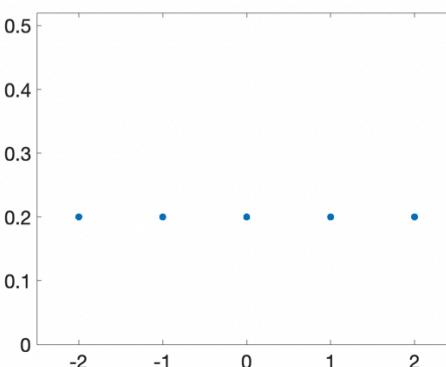
# 1D Discrete Convolution (\*)

We can express this moving averaging using convolution!

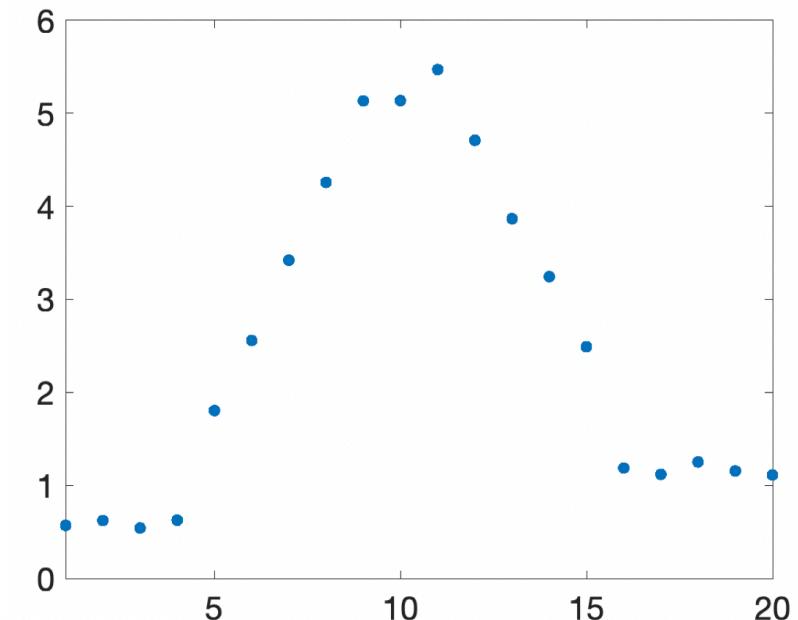
Original data  $f[n]$



Filter  $g[n]$



After moving average,  
 $h[n]$



\*

=

$$h[n] = (f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

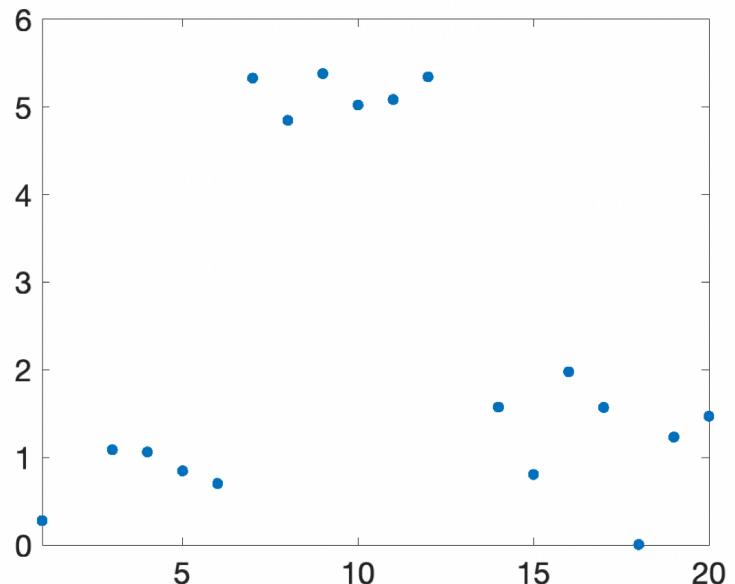
# Quick Facts of Convolution

	Discrete signal	Continuous signal
Convolution	$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$	$(f * g)(x) = \int_{t=-\infty}^{\infty} f(t)g(x-t)dt$
Fourier Transform	$\mathcal{F}(f)[n] = \sum_{m=0}^{M-1} f[m] \exp\left(-\frac{i2\pi}{M} m n t\right)$	$\mathcal{F}(f) = \int_{t=-\infty}^{\infty} f(t) \exp(-i2\pi\omega t) dt$
• Derivative Theorem	$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$	
• Convolution Theorem	$\mathcal{F}(f * g) = \mathcal{F}(f)\mathcal{F}(g)$ $\therefore h = \mathcal{F}^{-1}(\mathcal{F}(f)\mathcal{F}(g))$	

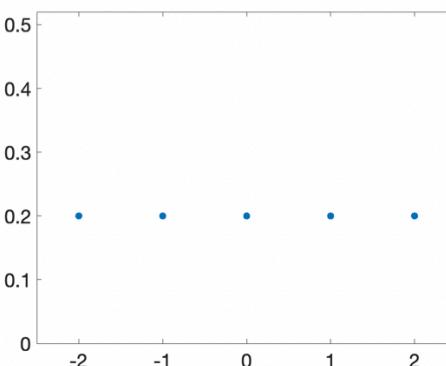
# Discrete Convolution: \*

Our filter is indeed a rectangular function. What is its Fourier transform?

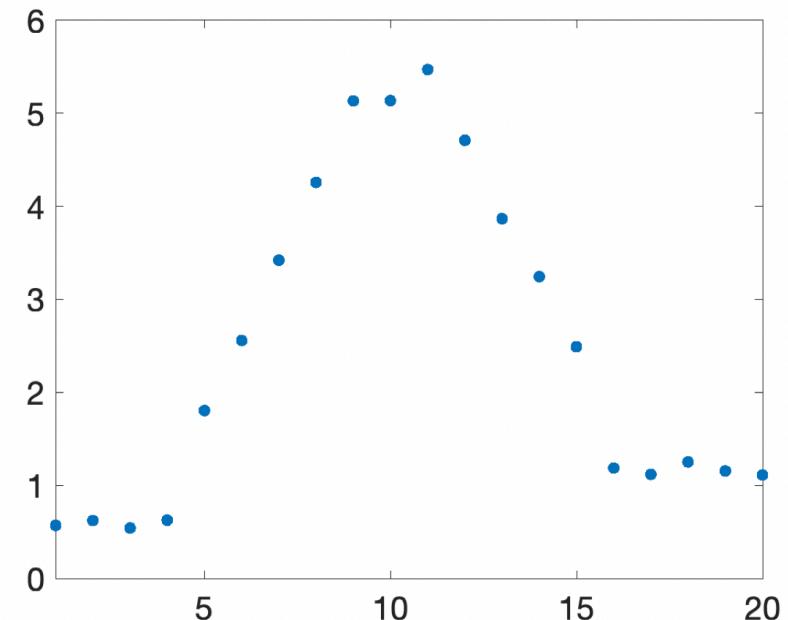
Original data  $f[n]$



Filter  $g[n]$



After moving average,  
 $h[n]$



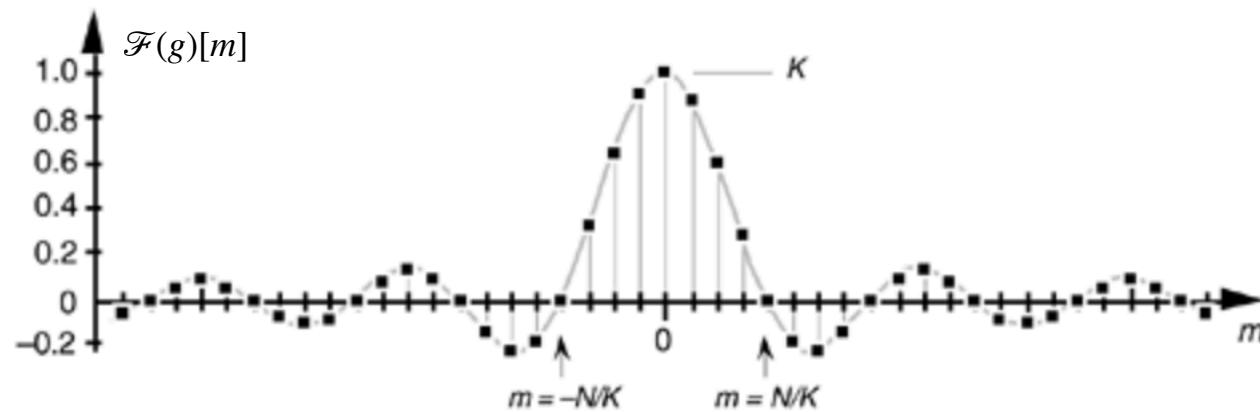
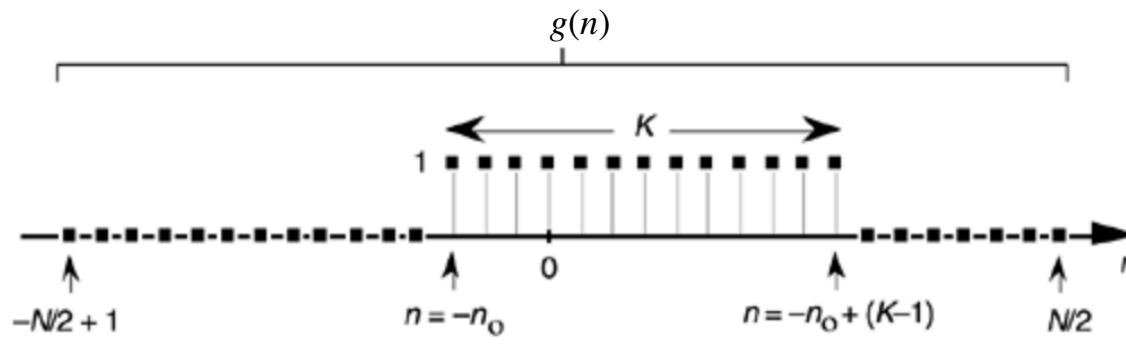
\*

=

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

# Rectangular Function and its Fourier Transform

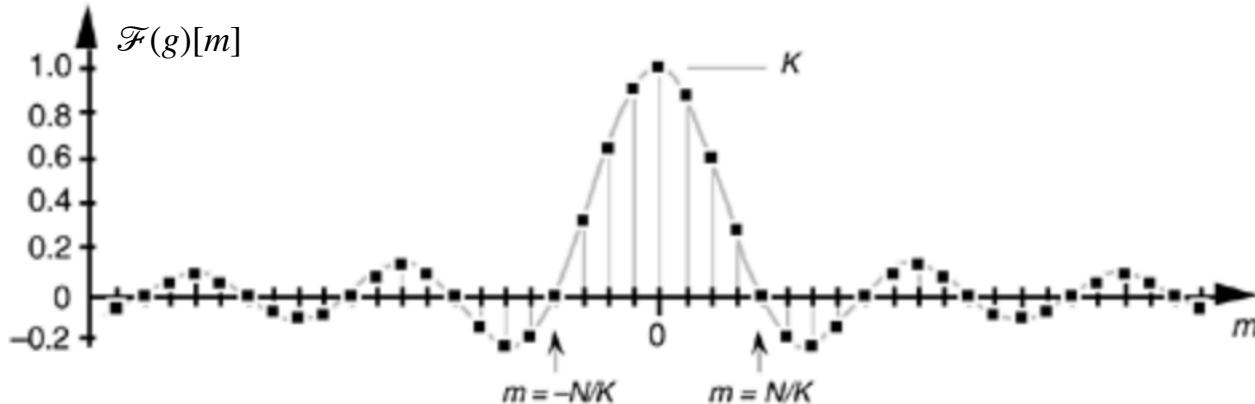
Figure 3-24. Rectangular function of width K samples defined over N samples where  $K < N$ .



$\mathcal{F}(g)[m]$  mainly  
concentrates around 0  
 $\Rightarrow g$  is a low-pass filter.

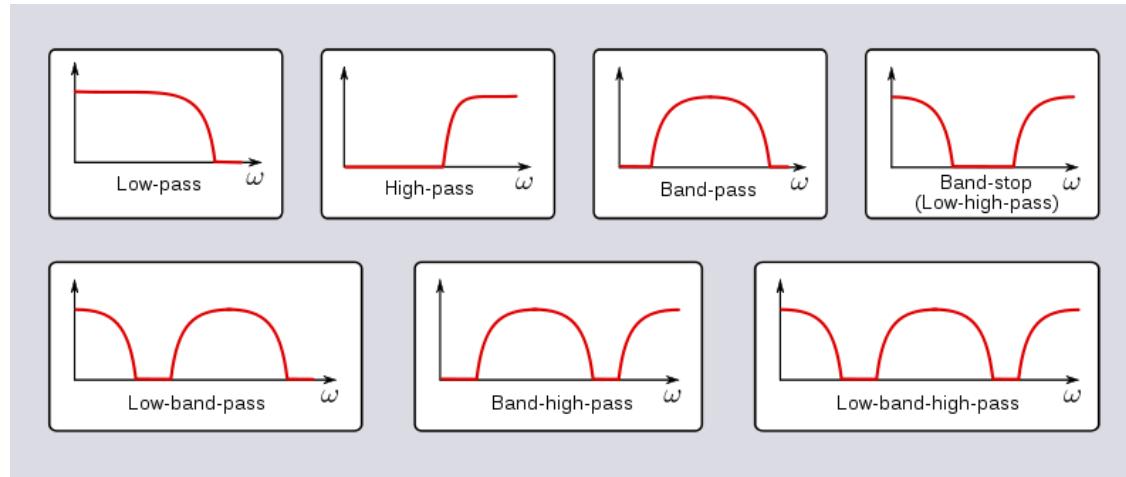
For more information about this discrete Fourier transform, please see  
[https://flylib.com/books/en/2.729.1/the\\_dft\\_of\\_rectangular\\_functions.html](https://flylib.com/books/en/2.729.1/the_dft_of_rectangular_functions.html)

# From a Low-Pass Filter Perspective



$\mathcal{F}(g)[m]$  mainly  
concentrates around 0  
 $\Rightarrow g$  is a low-pass filter.

According to Convolution theorem,  
 $\mathcal{F}(f * g) = \mathcal{F}(f)\mathcal{F}(g)$   
 $\mathcal{F}(f)\mathcal{F}(g)$  means the high frequency part of  
 $\mathcal{F}(f)$  turns to 0 in  $\mathcal{F}(f * g)$ .



When you removes high frequency parts in the signal, the signal becomes smooth. That's how Fourier transform explains the smoothing effect by moving average.

# Linear System $\leftrightarrow$ Linear Filters $\leftrightarrow$ Convolution

$$f[n] \rightarrow \boxed{\text{System } \mathcal{G}} \rightarrow h[n]$$

$$h = \mathcal{G}(f), \quad h[n] = \mathcal{G}(f)[n]$$

- Linear filtering  $\mathcal{G}$ :

- $h[n]$  is a linear combination of values from  $f[n]$
  - The weight of this linear combination is the same at each point  $n$

- Then  $\mathcal{G}$  is a linear system (function) iff  $\mathcal{G}$  satisfies

$$\mathcal{G}(\alpha f_1 + \beta f_2) = \alpha \mathcal{G}(f_1) + \beta \mathcal{G}(f_2)$$

- It can be proved that linear filters can also be expressed using convolutions.

# 2D Discrete-Space Systems (Filters)

$$f[n, m] \rightarrow \boxed{\text{System } \mathcal{G}} \rightarrow h[n, m]$$

$$h = \mathcal{G}(f), \quad h[n, m] = \mathcal{G}(f)[n, m]$$

# 2D Discrete Filter Example: Moving Average

- 2D DS moving average over a  $3 \times 3$  window of neighborhood

$$h[m, n] = \frac{1}{9} \sum_{k=n-1}^{n+1} \sum_{l=m-1}^{m+1} f[k, l]$$

$$= \frac{1}{9} \sum_{k=-1}^1 \sum_{l=-1}^1 f[n - k, m - l]$$

$$= (f * g)[m, n] = \sum_{k,l} f[k, l] g[m - k, n - l]$$

$g =$

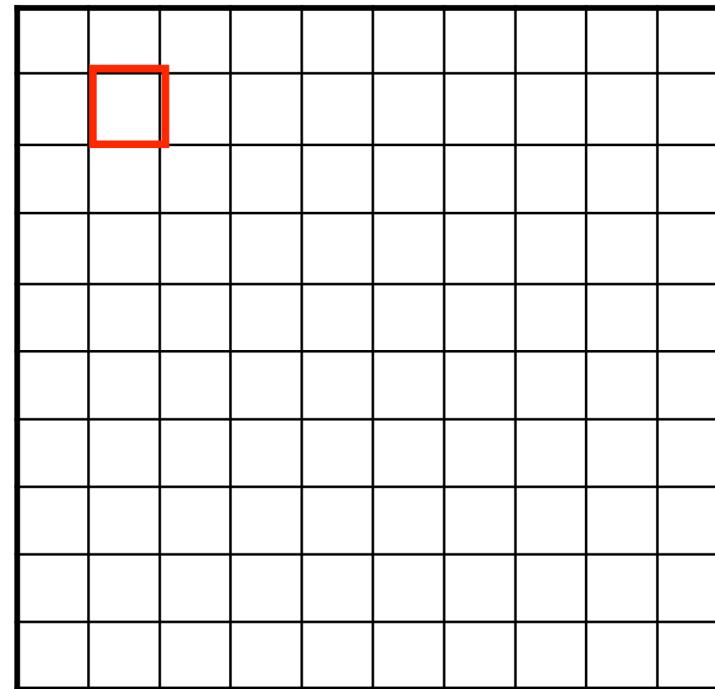
$$\frac{1}{9} \begin{array}{|c|c|c|} \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

# 2D Discrete Filter Example: Moving Average

$f[m, n]$

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	90	90	90	90	90	90	0	0
0	0	0	0	90	90	90	90	90	90	0	0
0	0	0	0	90	90	90	90	90	90	0	0
0	0	0	0	90	0	90	90	90	90	0	0
0	0	0	0	90	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

$h[m, n]$



$$(f * g)[m, n] = \sum_{k,l} f[k, l] g[m - k, n - l]$$

Image borrowed from Stanford CS131

# 2D Discrete Filter Example: Moving Average

$f[m, n]$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$h[m, n]$

0	10									

$$(f * g)[m, n] = \sum_{k,l} f[k, l] g[m - k, n - l]$$

Image borrowed from Stanford CS131

# 2D Discrete Filter Example: Moving Average

$f[m, n]$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	90	0	0
0	0	0	90	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$h[m, n]$


$$(f * g)[m, n] = \sum_{k,l} f[k, l] g[m - k, n - l]$$

Image borrowed from Stanford CS131

# 2D Discrete Filter Example: Moving Average

$f[m, n]$

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	90	0	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

$h[m, n]$


$$(f * g)[m, n] = \sum_{k,l} f[k, l] g[m - k, n - l]$$

Image borrowed from Stanford CS131

# 2D Discrete Filter Example: Moving Average

$f[m, n]$

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	90	0	90	90	90	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

$h[m, n]$

			0	10	20	30	30				

$$(f * g)[m, n] = \sum_{k,l} f[k, l] g[m - k, n - l]$$

Image borrowed from Stanford CS131

# 2D Discrete Filter Example: Moving Average

$f[m, n]$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$h[m, n]$

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		

$$(f * g)[m, n] = \sum_{k,l} f[k, l] g[m - k, n - l]$$

Image borrowed from Stanford CS131

Source: S. Seitz

# Summary of Moving Average

- Replaces each pixel with an average of its neighborhood.
- Achieve smoothing effect (remove sharp features)



$$\frac{1}{9} \begin{matrix} 1 & 1 & 1 \\ 1 & \boxed{1} & 1 \\ 1 & 1 & 1 \end{matrix}$$

# Non-linear Filtering Example: Binarization via Thresholding

Define a threshold  $\tau$ , e.g.,  $\tau = 100$ .

$$h[m, n] = \begin{cases} 1, & f[n, m] > \tau \\ 0, & \text{otherwise.} \end{cases}$$



Is thresholding a linear system?

- $f1[n,m] + f2[n,m] > T$
- $f1[n,m] < T$
- $f2[n,m] < T$       No!

# Edge Detection

# Start with A Task: Lane Detection



How to detect the lane?

<https://medium.com/@realderektan/self-driving-car-project-part-1-lane-lines-detector-6d960e2b023>

# Start with Detecting Edges

- Edge detector

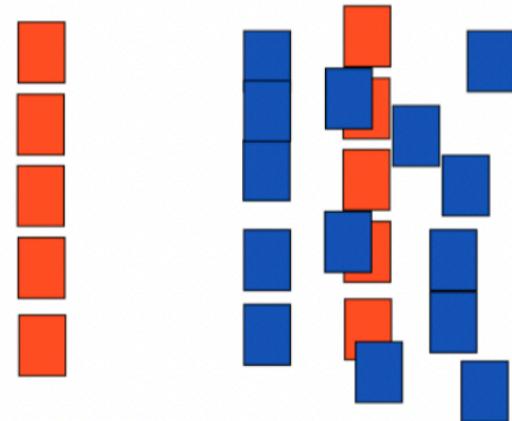


# What is an Edge?

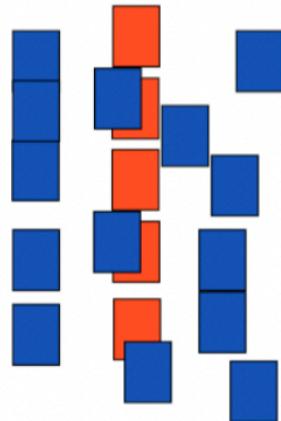
- An edge is defined as a region in the image where there is a “significant” change in the pixel intensity values (or having high contrast) along one direction in the image, and almost no changes in the pixel intensity values (or low contrast) along its orthogonal direction.



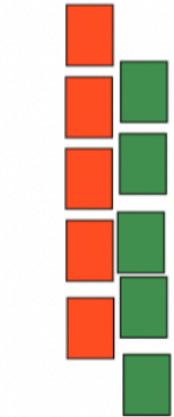
# Criteria for Optimal Edge Detection



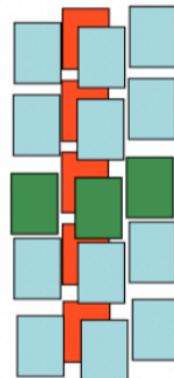
True  
edge



Low precision



Poor  
localization



Too many  
responses

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

TP: true positives, FP: false positives  
TN: true negatives, FN: false negatives

- High precision: make sure all detected edges are true edges (via minimizing FP).
- High recall: make sure all edges can be detected (via minimizing FN).
- Good localization: minimize the distance between the detected edge and the ground truth edge
- Single response constraint: minimize redundant responses

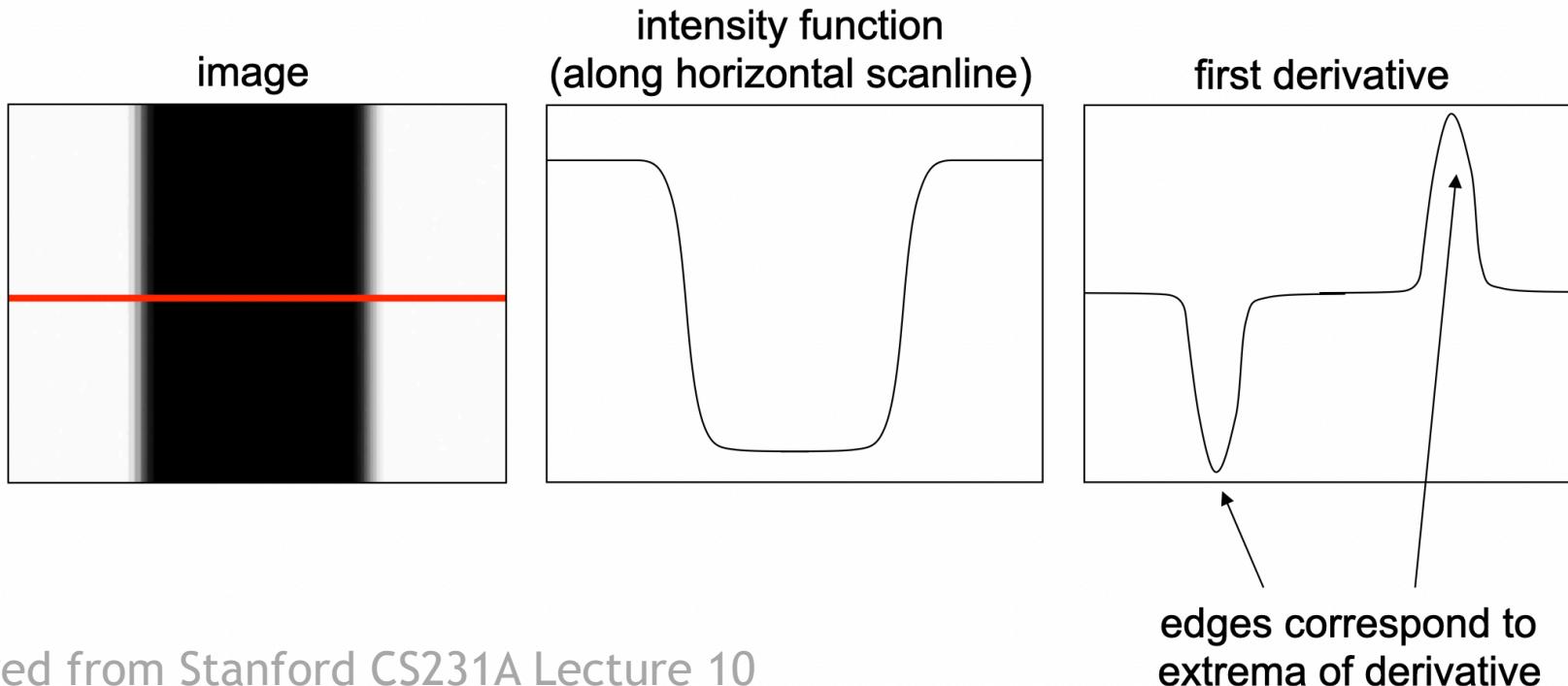
# What Causes An Edge?

- Depth discontinuity
- Surface orientation discontinuity
- Surface color discontinuity
- Illumination discontinuity

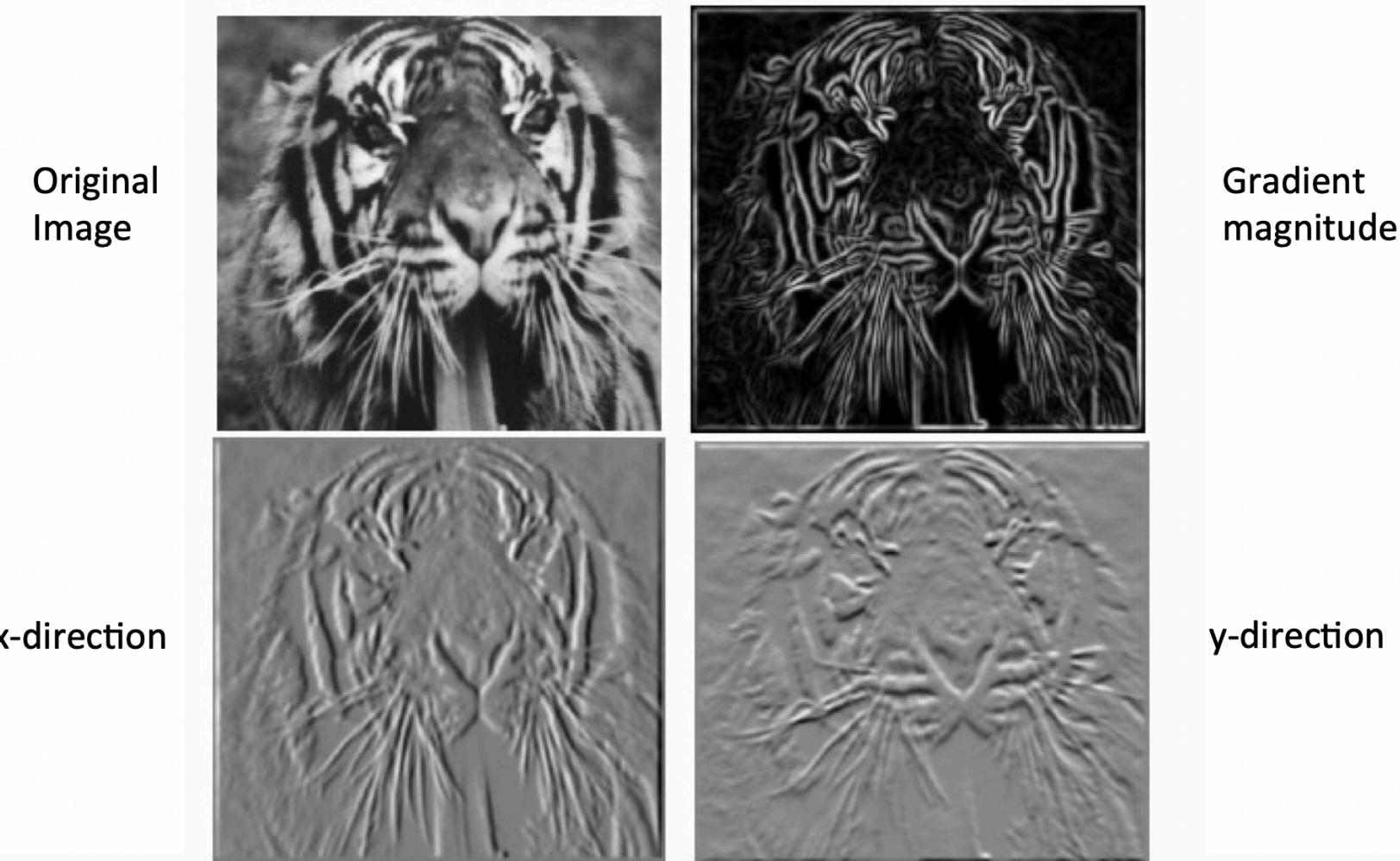


# Characterizing Edges

- An edge is defined as a region in the image where there is a “significant” **change in the pixel intensity values** along one direction in the image, and almost no changes in the pixel intensity values along its orthogonal direction.



# Visualizing Image Gradient



Source: Feifei Li

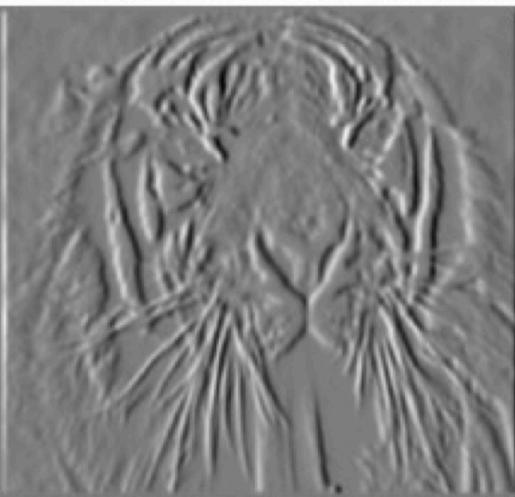
Gradient magnitude:  $\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$

# Problem

Original  
Image

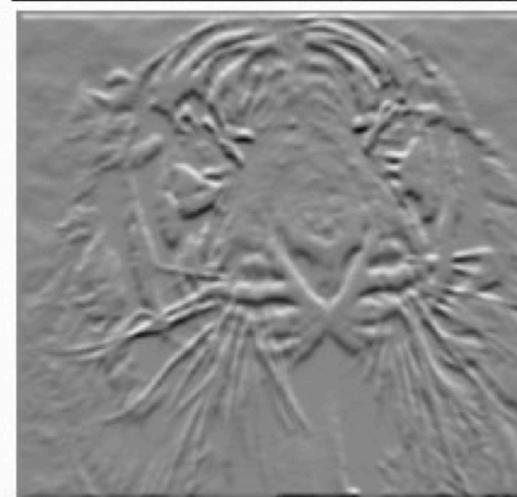


x-direction



Gradient  
magnitude

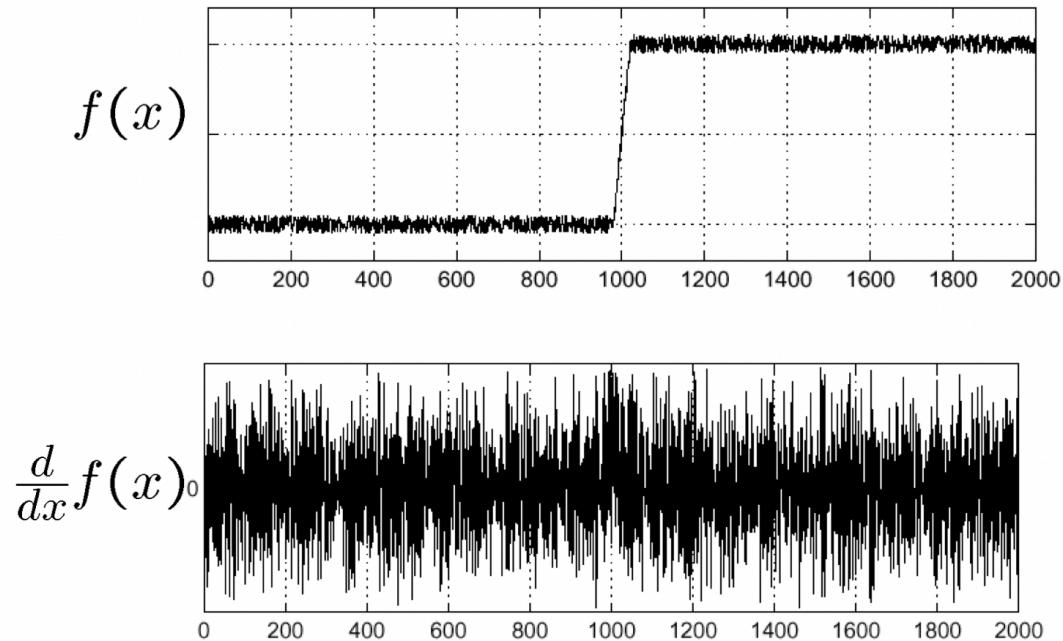
y-direction



- Gradient is non-zero everywhere. Where is the edges?

# Effects of Noises

- Consider one row in the image

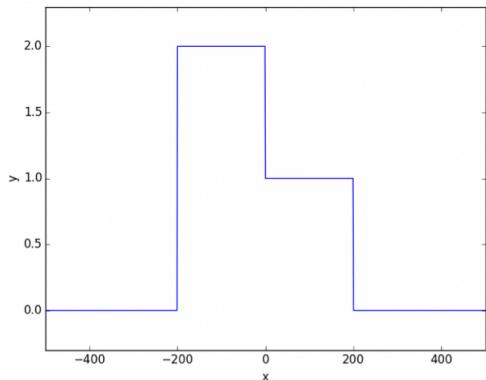


- Image gradients are too sensitive to noise.
- Gradients of the true edge is overwhelmed by noises.
- **We need smoothing!**

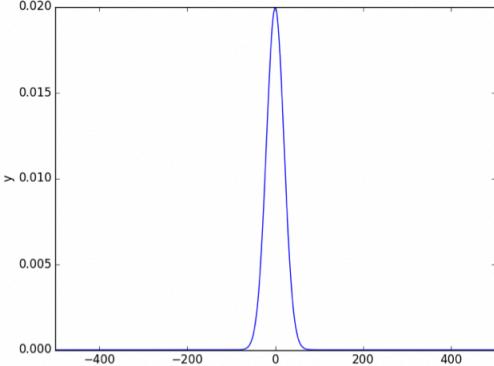
Source: Steven Seitz

# Smoothing by Gaussian Filter

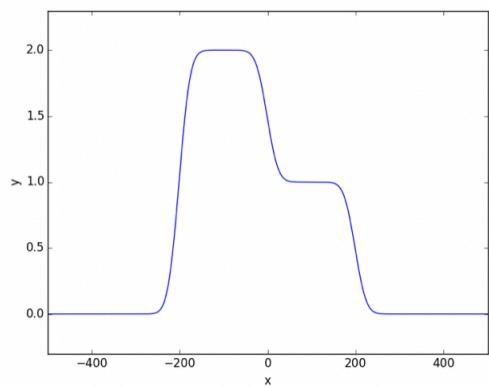
$f$  = function of measurement



$g$  = weighting function



$f$  convolved with  $g$  (written  $f*g$ )

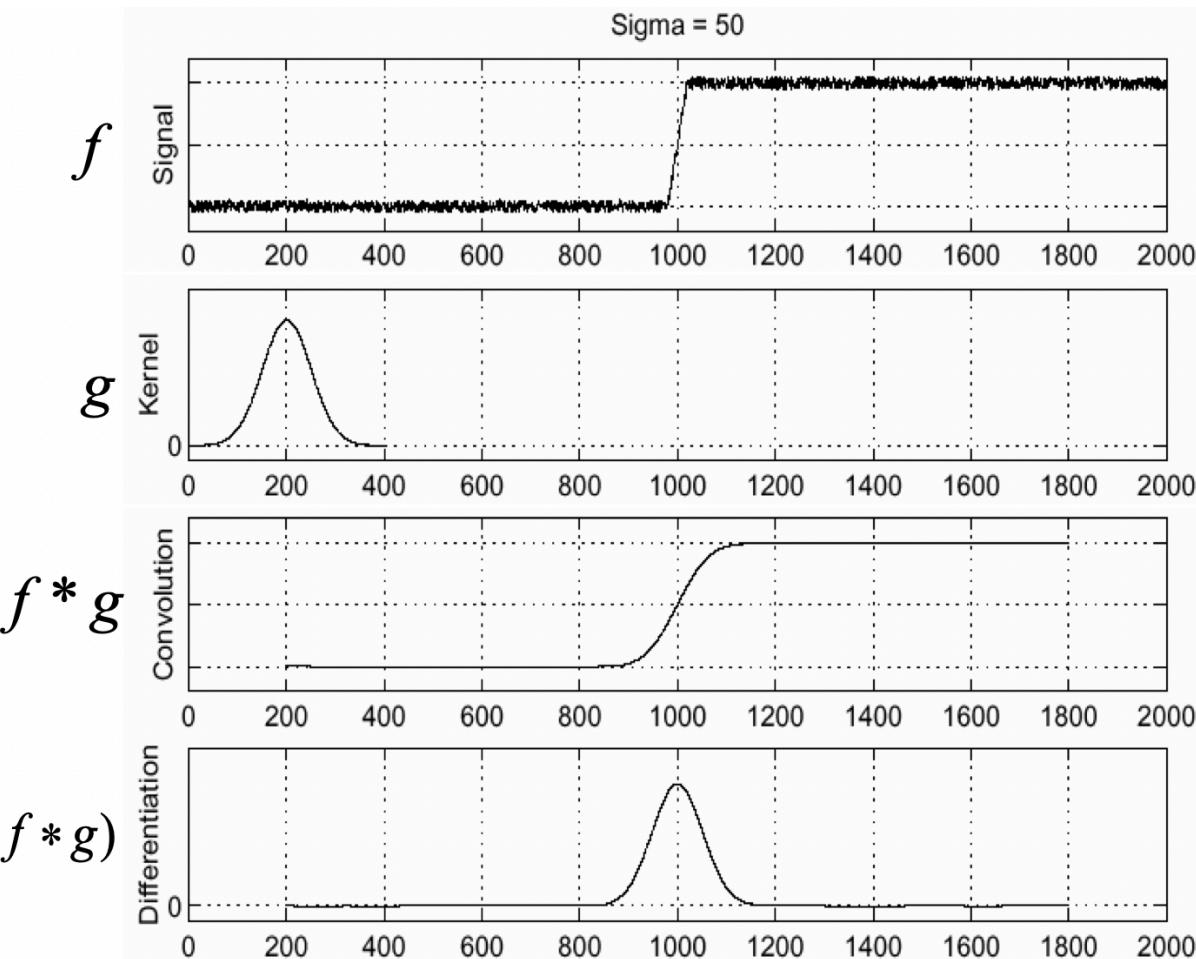


- Gaussian transforms to another Gaussian,
- **low-pass filter!**

$$g = \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{x^2}{2\sigma^2} \quad \mathcal{F}(g) = \exp\left(-\frac{\sigma^2\omega^2}{2}\right)$$

- The bigger  $\sigma$  is, the sharper  $\mathcal{F}(g)$  is. When  $\sigma \rightarrow +\infty$ , filter all high-frequency parts and then the signal becomes a constant.
- The smaller  $\sigma$  is, the boarder  $\mathcal{F}(g)$  is. When  $\sigma \rightarrow 0$ ,  $\mathcal{F}(g) = 1$ , no filtering at all.

# Smoothing by a Low-Pass Filter



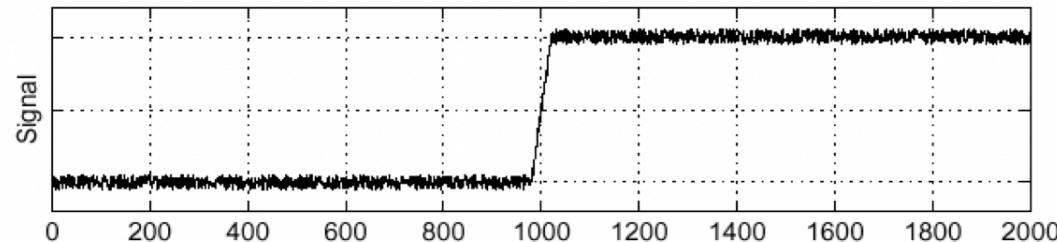
Source: Steven Seitz

# Derivative Theorem of Convolution

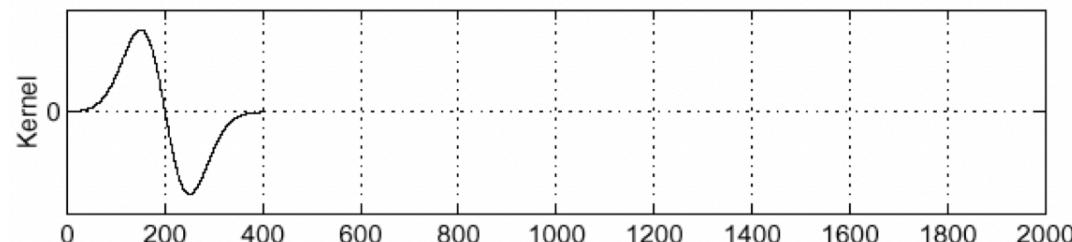
- Theorem:

$$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$$

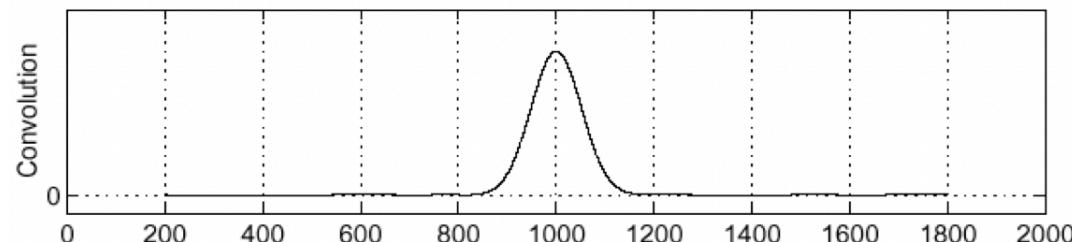
$f$



$$\frac{d}{dx}g$$



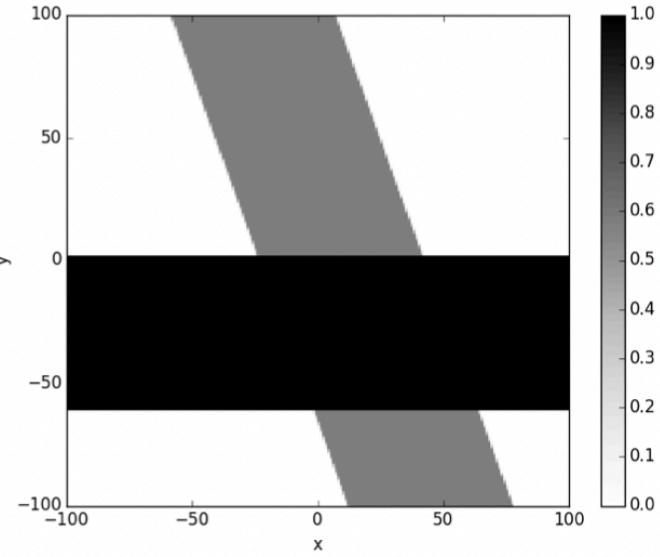
$$f * \frac{d}{dx}g$$



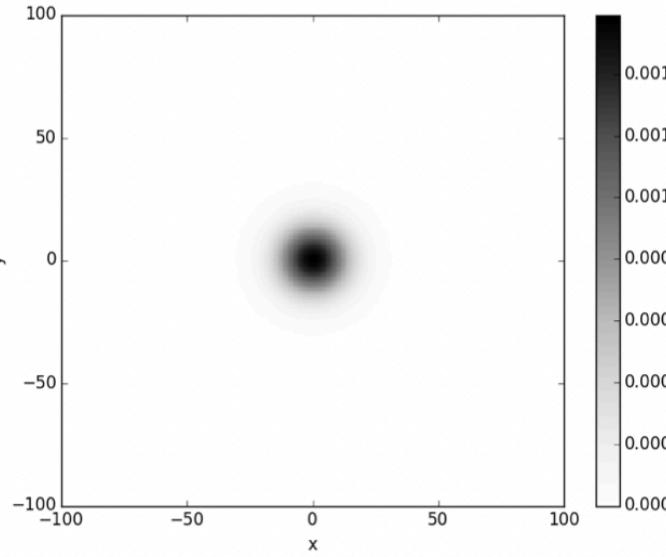
- Saves us one operation.

# Two-Dimensional Convolution

$f$

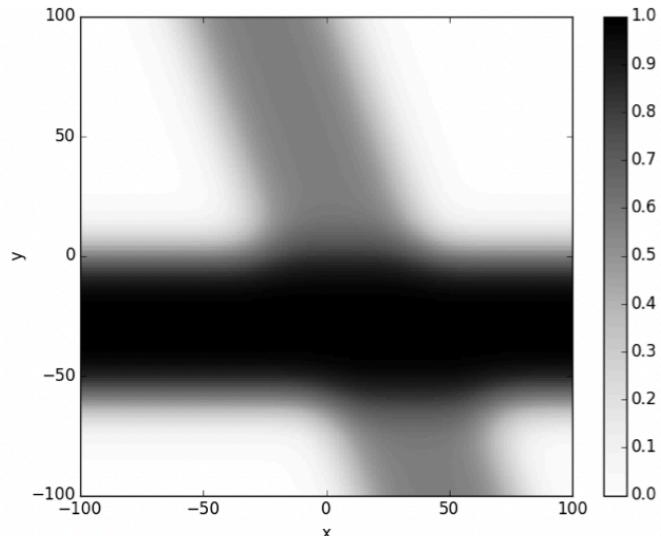


\*



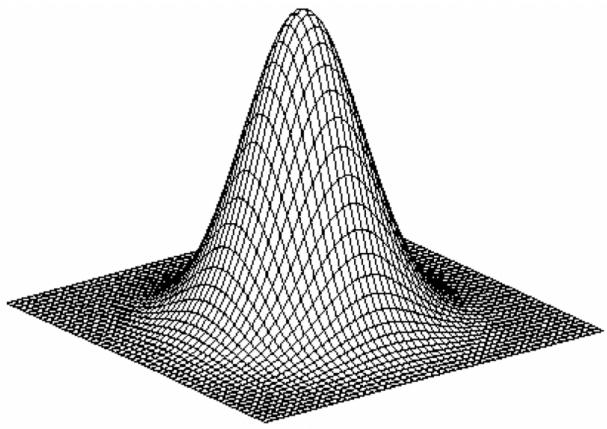
$$g = \frac{1}{2\pi\sigma^2} \exp - \frac{x^2 + y^2}{2\sigma^2}$$

=

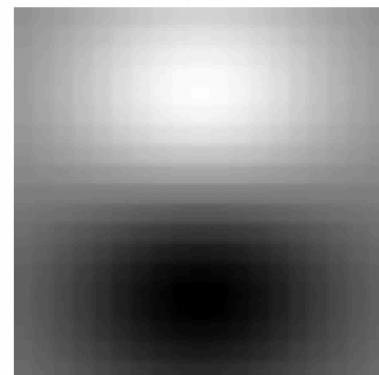
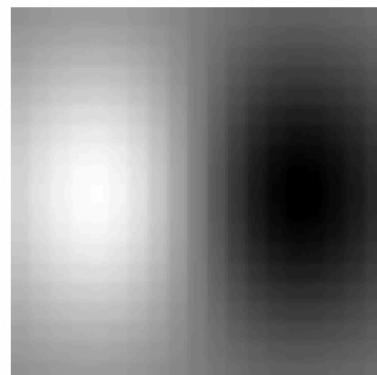
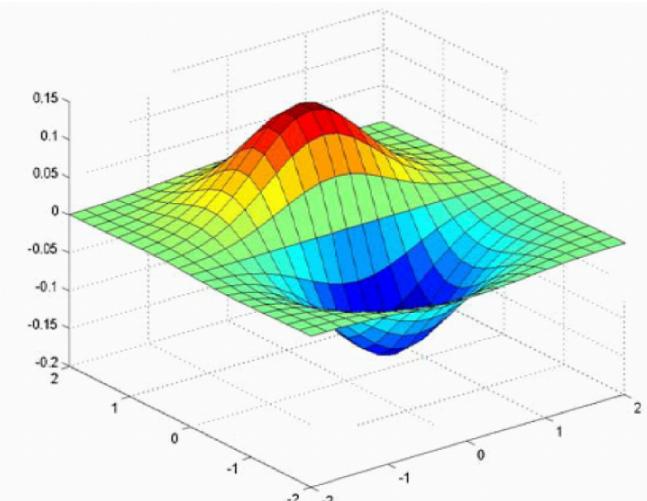
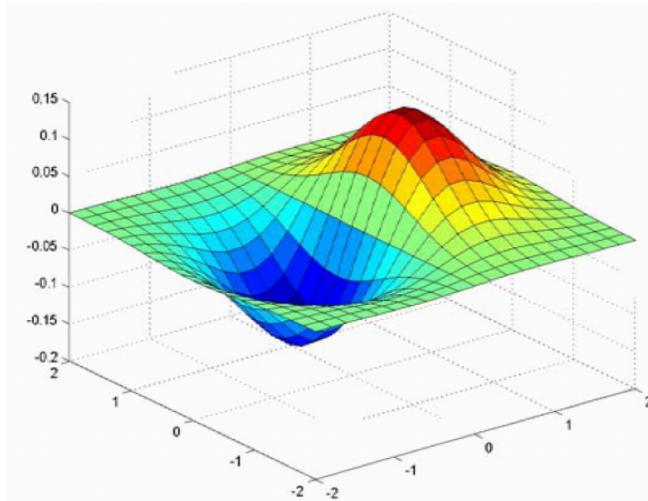


$$(f * g)[m, n] = \sum_{k=-\infty}^{\infty} \sum_{l=-\infty}^{\infty} f[k, l]g[m - k, n - l]$$

# Derivative of 2D Gaussian Filter



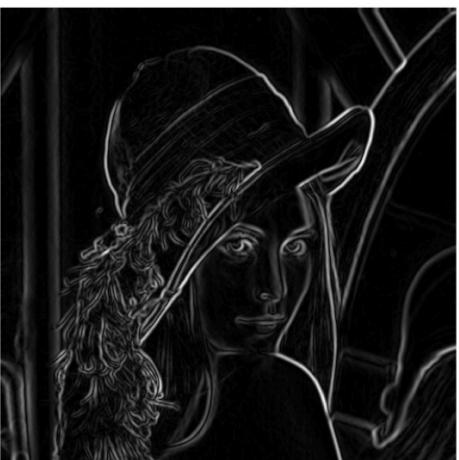
$\ast [1 \ -1]$  =



# Compute Gradient



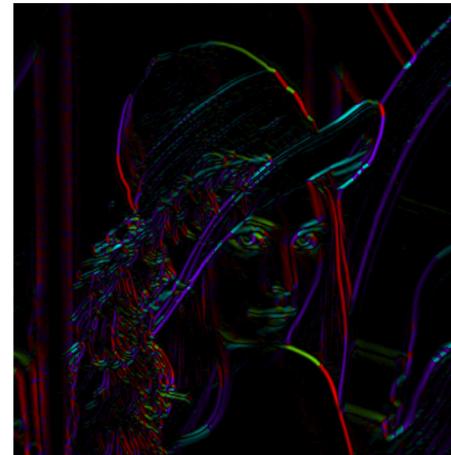
x-derivative of Gaussian



Gradient magnitude

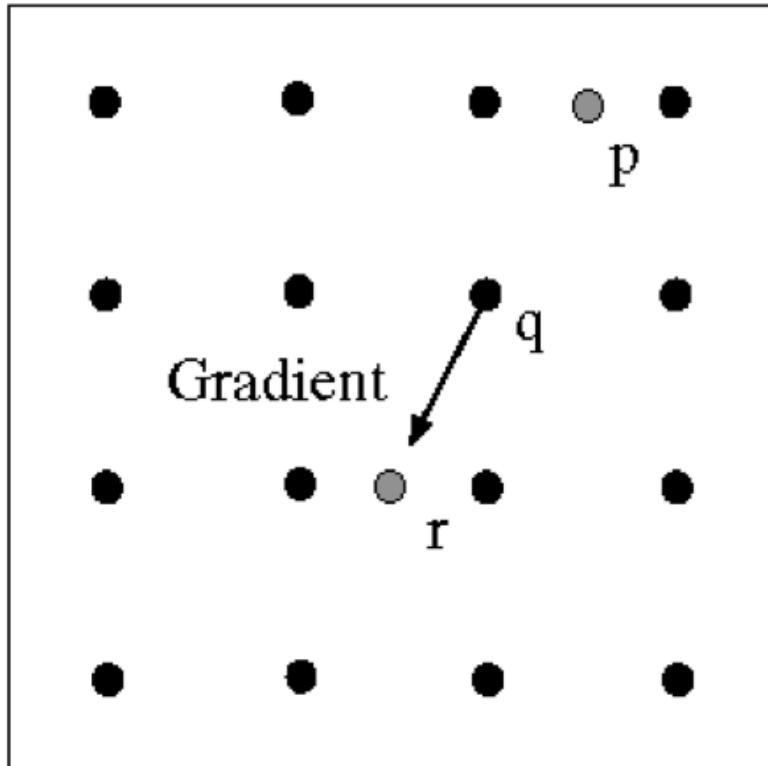


y-derivative of Gaussian



Thresholding and Gradient orientation

# Non-Maximal Suppression (NMS)



- For each point  $q$  on grids, compute the gradient  $g(q)$ .
- Move along the gradient to get two neighbors:  
 $r = q + g(q), p = q - g(q)$
- Perform **bilinear interpolation** to get  $g(p)$  and  $g(r)$ .
- If the magnitude of  $g(q)$  is larger than  $g(p)$  and  $g(r)$ ,  $q$  is a maximum that should be kept.

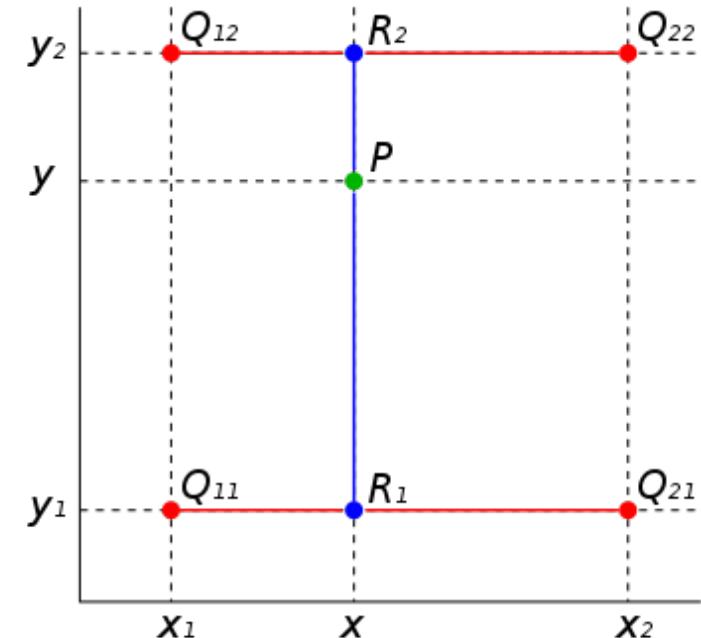
# Bilinear Interpolation

For  $P(x, y)$ , given its four surrounding grid points  $f(Q_{11}), f(Q_{12}), f(Q_{21})$  and  $f(Q_{22})$ ,  
how to obtain  $f(P)$  via *bilinear interpolation*?

First, linear interpolate to obtain  $f(R_1)$  and  $f(R_2)$

$$R_1 : f(x, y_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}),$$

$$R_2 : f(x, y_2) = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}).$$



# Bilinear Interpolation

For  $P(x, y)$ , given its four surrounding grid points  $f(Q_{11}), f(Q_{12}), f(Q_{21})$  and  $f(Q_{22})$ ,  
how to obtain  $f(P)$  via *bilinear interpolation*?

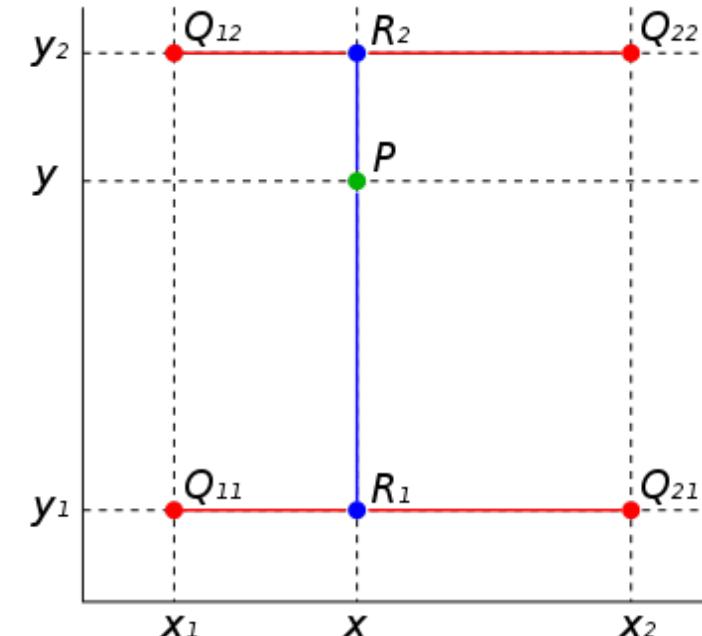
First, linear interpolate to obtain  $f(R_1)$  and  $f(R_2)$

$$R_1 : f(x, y_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}),$$

$$R_2 : f(x, y_2) = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}).$$

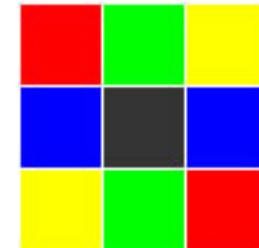
Then, linear interpolate between  $f(R_1)$  and  $f(R_2)$  to obtain  $f(P)$ :

$$\begin{aligned} P : f(x, y) &= \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y - y_1}{y_2 - y_1} f(x, y_2) \\ &= \frac{y_2 - y}{y_2 - y_1} \left( \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \right) + \frac{y - y_1}{y_2 - y_1} \left( \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \right) \end{aligned}$$

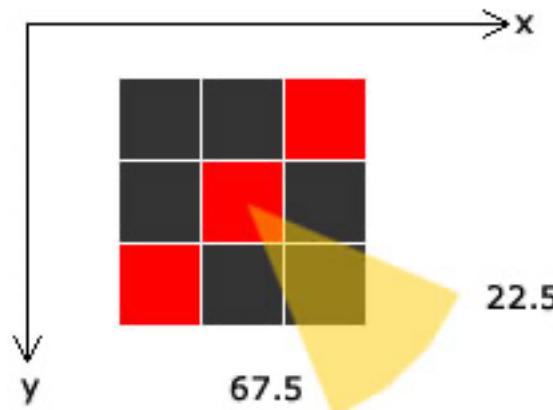


# A Simplified Version of NMS

The orientation of each pixel is put into one of the four bins.



Example: gradient orientation from 22.5 to 67.5 degrees



To check if the central red pixel belongs to an edge, you need to check if the gradient is maximum at this point. You do this by comparing its magnitude with the top left pixel and the bottom right pixel.

# Before and After NMS



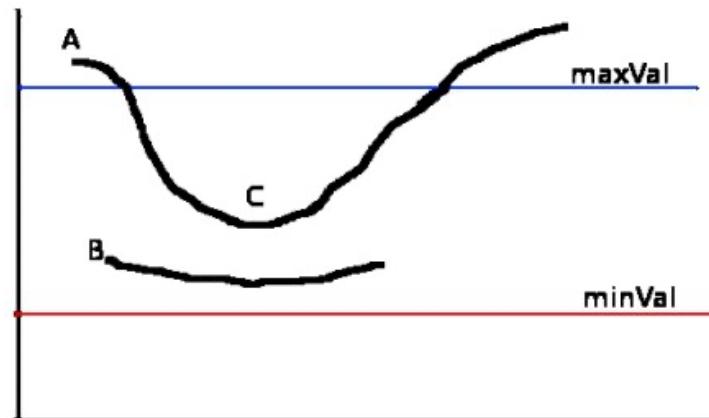
Thin multi-pixel wide “ridges” down to single pixel width

# Hysteresis Thresholding

- Use a high threshold (`maxVal`) to start edge curves and a low threshold (`minVal`) to continue them.
  - Pixels with gradient magnitudes  $> \text{maxVal}$  should be reserved
  - Pixels with gradient magnitudes  $< \text{minVal}$  should be removed.
  - How to decide `maxVal` and `minVal`? Examples:
    - $\text{maxVal} = 0.3 \times \text{average magnitude of the pixels that pass NMS}$
    - $\text{minVal} = 0.1 \times \text{average magnitude of the pixels that pass NMS}$

# Edge Linking

- Drop-outs?
- Now using the direction information and the lower threshold, we'll "grow" these edges.
  - If the current pixel is not an edge, check the next one.
  - If it is an edge, check the two pixels in the direction of the edge (ie, perpendicular to the gradient direction). If either of them (or both)



# Edge Linking

- Drop-outs?
- Now using the direction information and the lower threshold, we'll "grow" these edges.
  - If the current pixel is not an edge, check the next one.
  - If it is an edge, check the two pixels in the direction of the edge (ie, perpendicular to the gradient direction). If either of them (or both)
    - have the direction in the same bin as the central pixel
    - gradient magnitude is greater than  $\text{minVal}$
    - they are the maximum compared to their neighbors (NMS for these pixels), then you can mark these pixels as an edge pixel



# Edge Linking

- Drop-outs?
- Now using the direction information and the lower threshold, we'll "grow" these edges.
  - If the current pixel is not an edge, check the next one.
  - If it is an edge, check the two pixels in the direction of the edge (ie, perpendicular to the gradient direction). If either of them (or both)
    - have the direction in the same bin as the central pixel
    - gradient magnitude is greater than  $\text{minVal}$
    - they are the maximum compared to their neighbors (NMS for these pixels), then you can mark these pixels as an edge pixel
  - Loop until there are no changes in the image Once the image stops changing, you've got your canny edges! That's it! You're done!

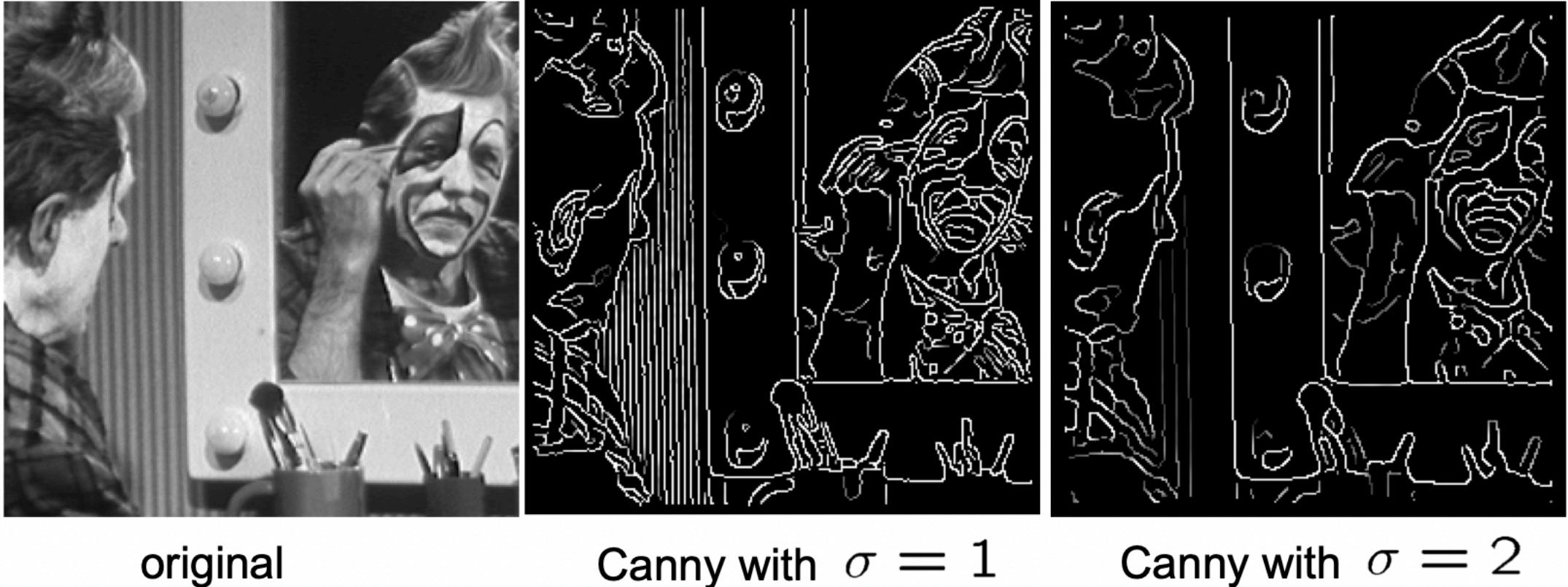
# Canny Edge Detector

- The most widely used edge detector in computer vision
- Canny shows that the first derivative of the Gaussian closely approximates the operator that optimizes the product of signal-to-noise ratio and localization.



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

# Tradeoff between Smoothing and Localization



original

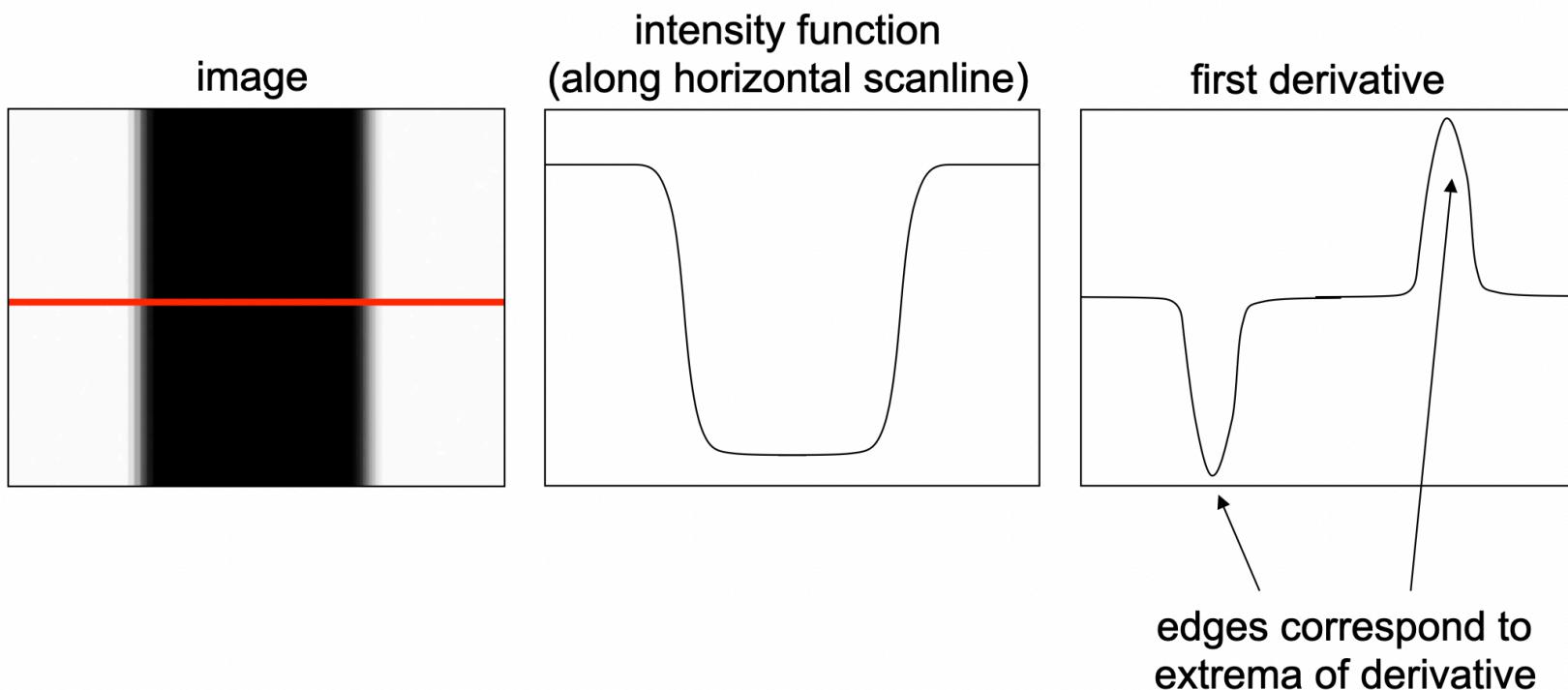
Canny with  $\sigma = 1$

Canny with  $\sigma = 2$

- Note a larger  $\sigma$  corresponds to stronger smoothing.
- Smoothed derivative reduces noises but blurs edges.
- Find edges at different scales.

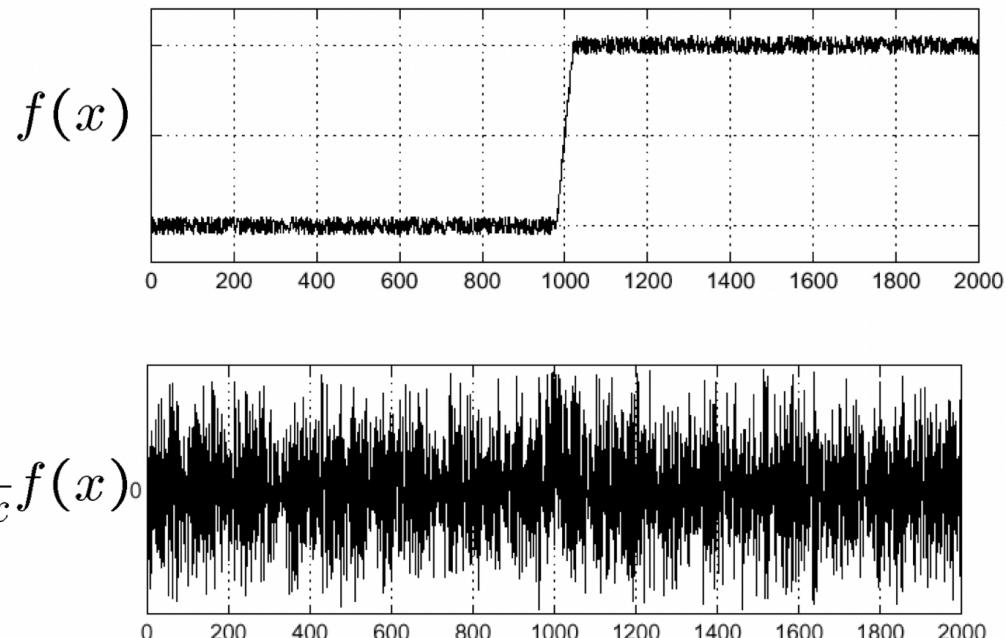
# Summary of Edge Detection

- What is an edge?



# Summary of Edge Detection

- Edge: where pixel intensity changes drastically
- Compute image gradient to find edge, however noises can be overwhelming and fail the detection



# Summary of Canny Edge Detection

- Edge: where pixel intensity changes drastically
- Jointly detecting edge and smoothing by convolving with the derivative of a Gaussian filter
- Non-maximal suppression
- Thresholding and linking (hysteresis):



# Keypoint Detection

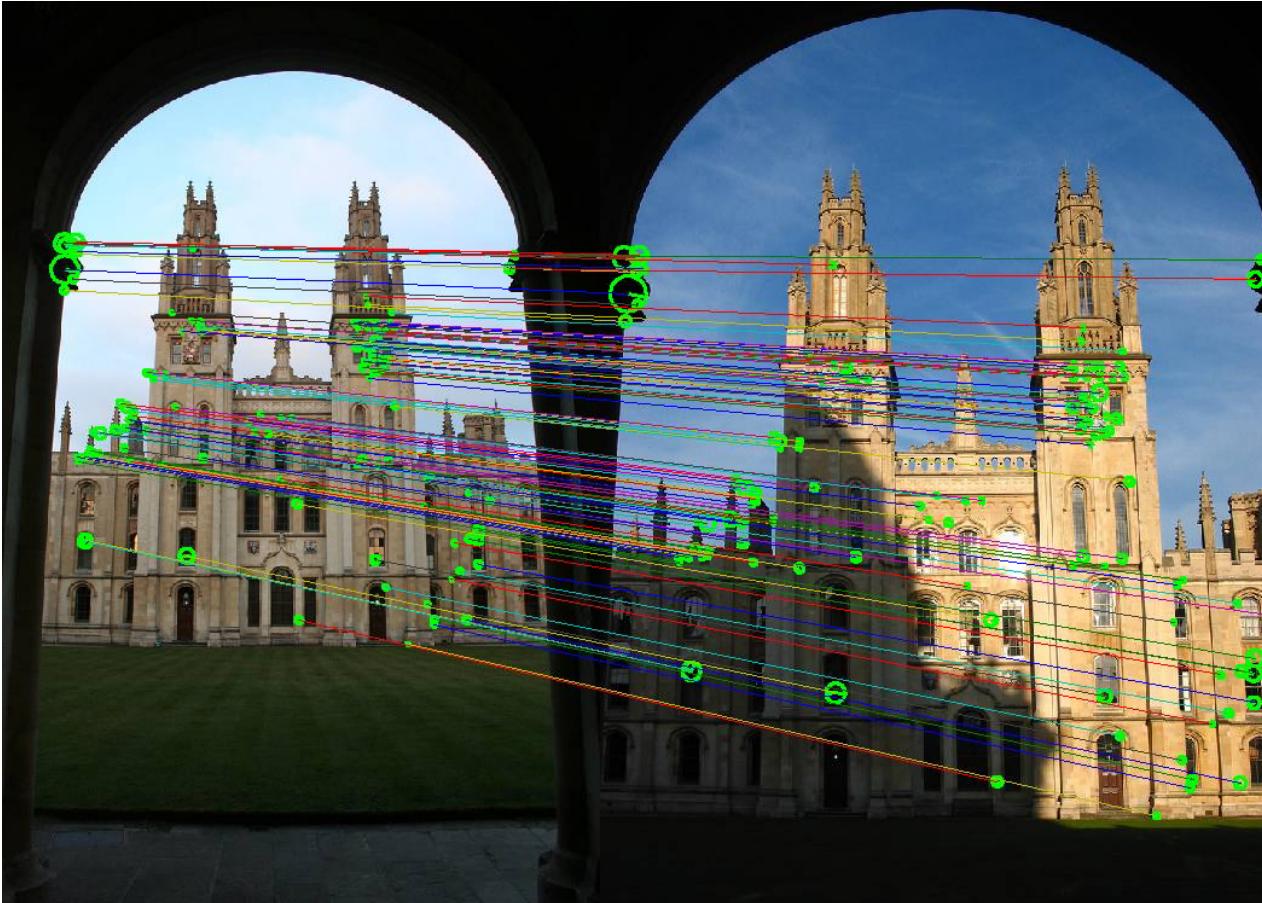
Some slides are borrowed from Stanford CS131.

# Keypoint Localization



- In addition to edges, keypoints are also important to detect.

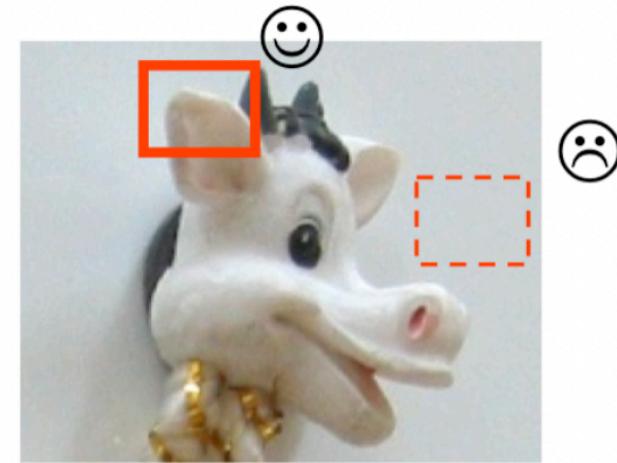
# Applications: Image Matching



Separately detect keypoints and then find matching.

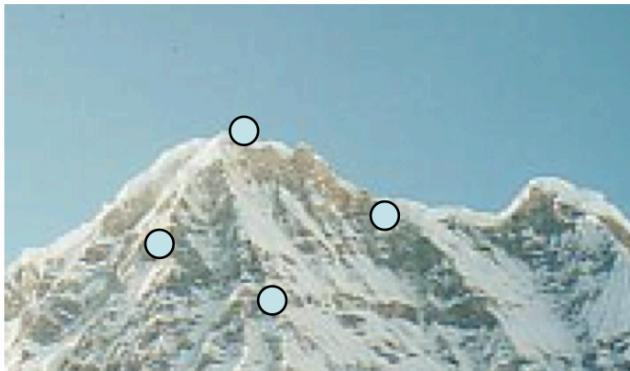
# What Points are Keypoints?

- Saliency: interesting points



# More Requirements

- Saliency: interesting points
- Repeatability: detect the same point independently in both images



**No chance to match!**

Image borrowed from Stanford CS131

# More Requirements

- Repeatability: detect the same point independently in both images
- Saliency: interesting points
- Accurate localization

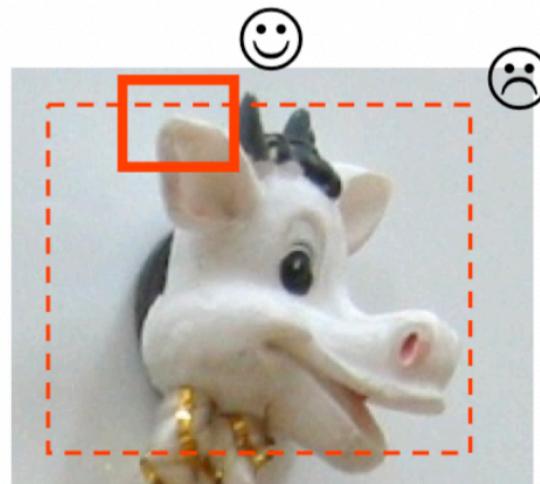
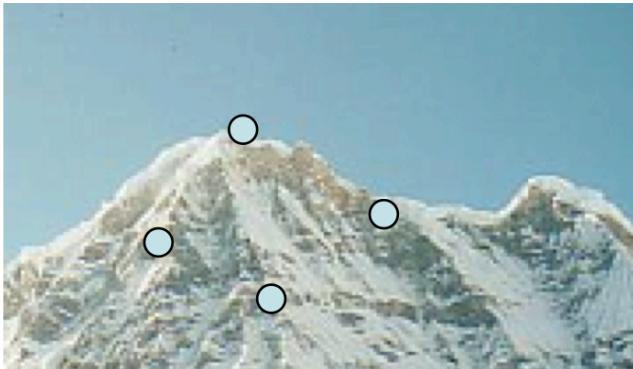


Image borrowed from Stanford CS131

# More Requirements

- Repeatability: detect the same point independently in both images
- Saliency: interesting points
- Accurate localization
- Quantity: sufficient number



No chance to match!

# Repeatability and Invariance

- For a keypoint detector to be repeatable, it has to be invariant to:
  - Illumination
  - Image scale
  - Viewpoint



Scale  
invariance

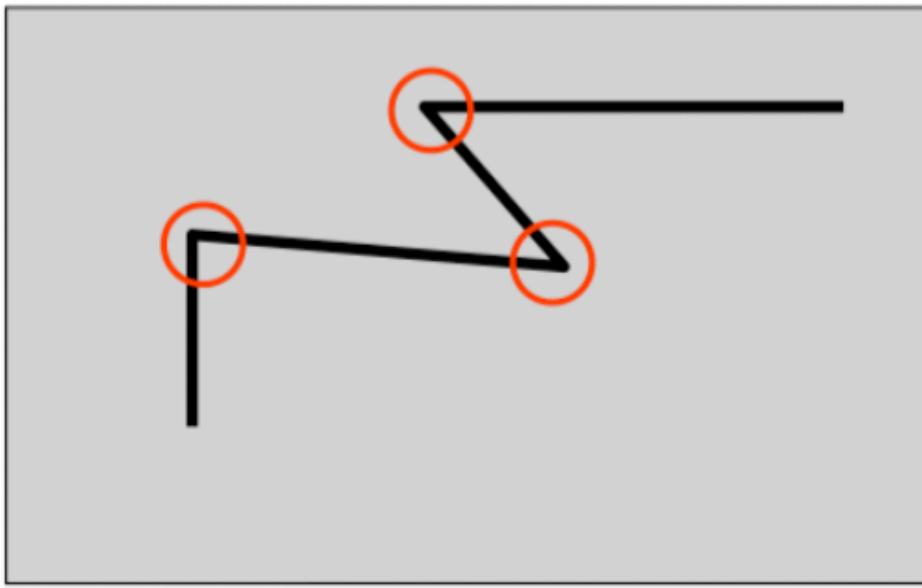


Illumination  
invariance



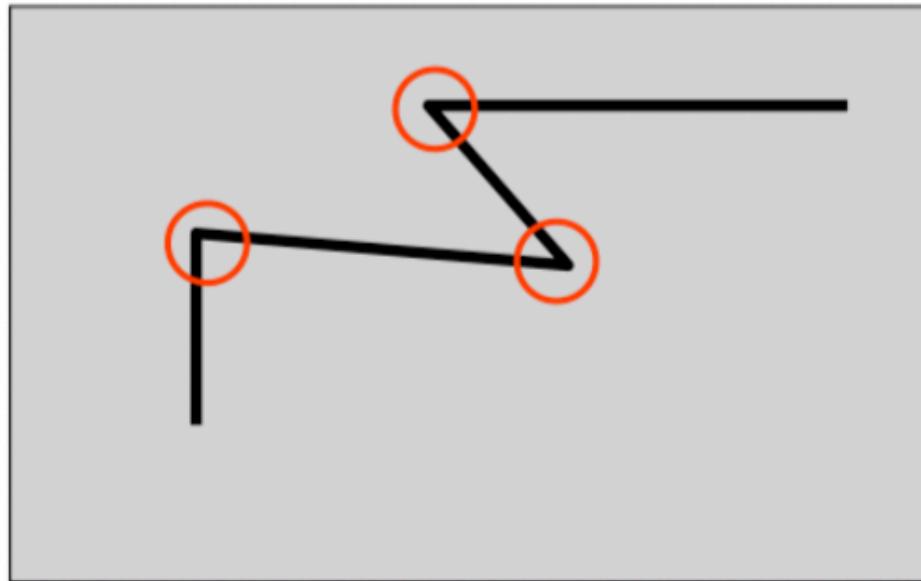
Pose invariance  
•Rotation  
•Affine

# Corners as Keypoints



- Corners are such kind of keypoints, because they are
  - Salient;
  - Repeatable (one corner would still be a corner from another viewpoint);
  - Sufficient (usually an image comes with a lot of corners);
  - Easy to localize.

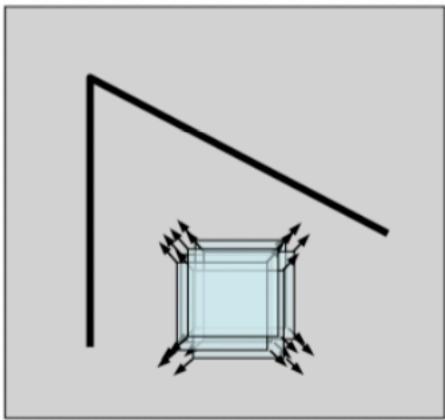
# The Properties of a Corner



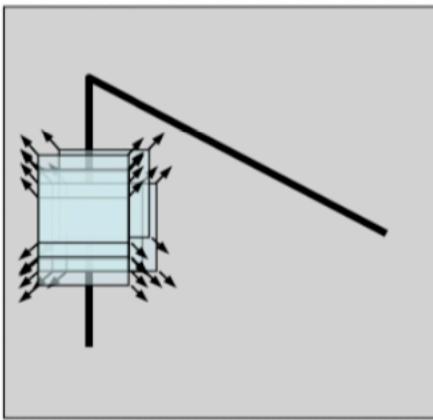
- The key property of a corner: In the region around a corner, image gradient has two or more dominant directions

# The Basic Idea of Harris Corner

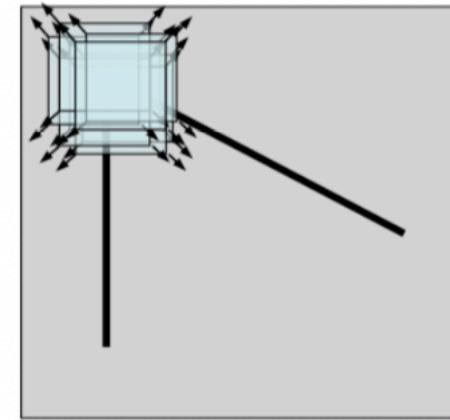
- Move a window and explore intensity changes within the window



Flat region: no  
change in all  
directions

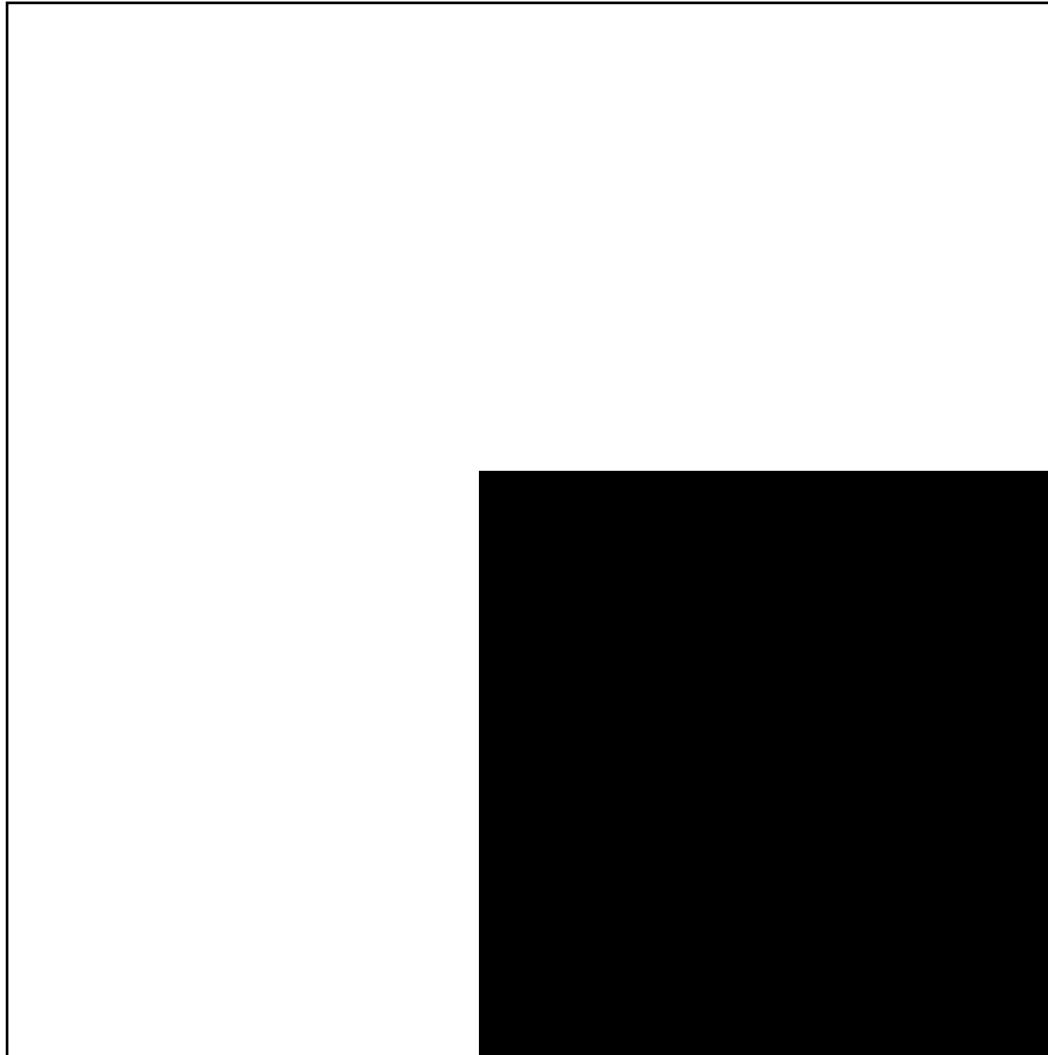


Edge: no change  
along the edge  
direction



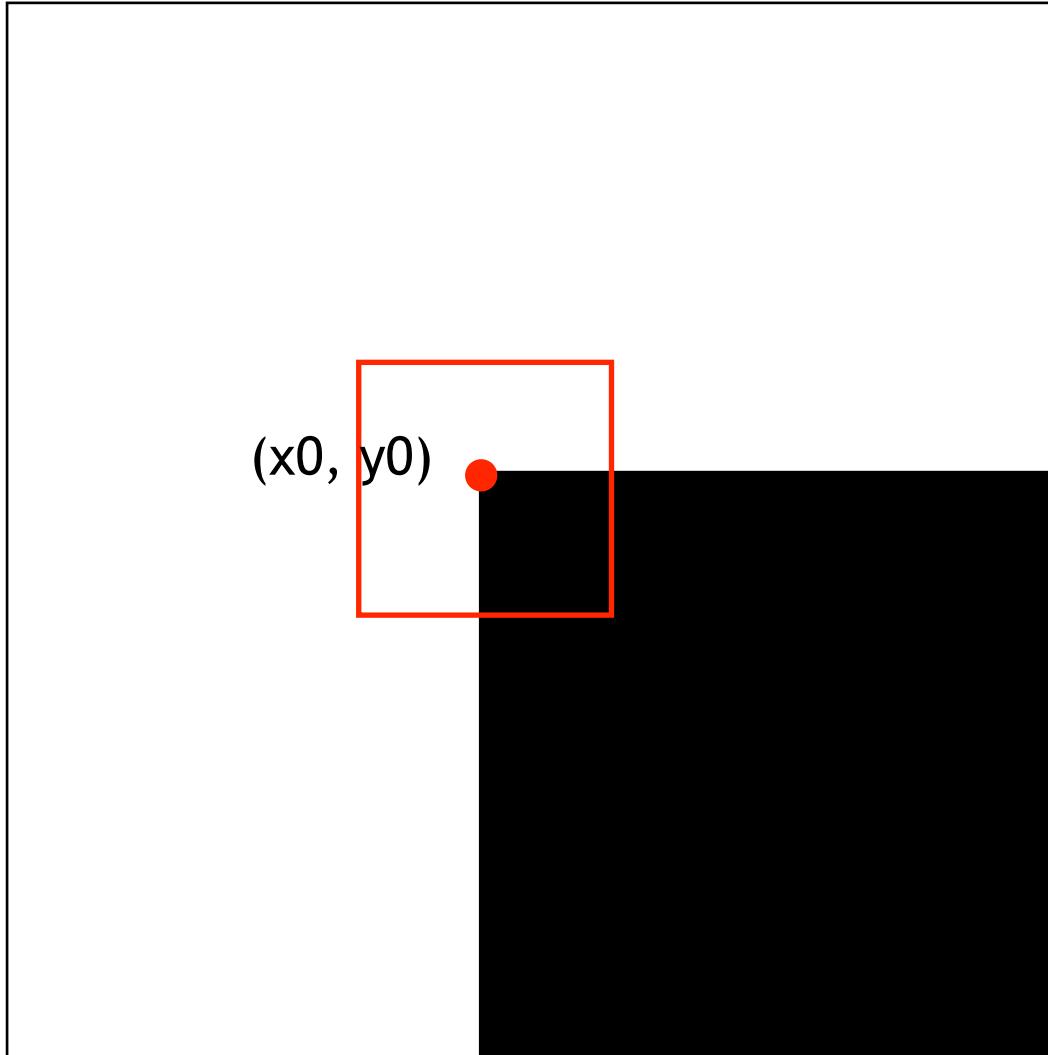
Corner:  
significant  
change in all  
directions

# The Basic Idea of Harris Corner



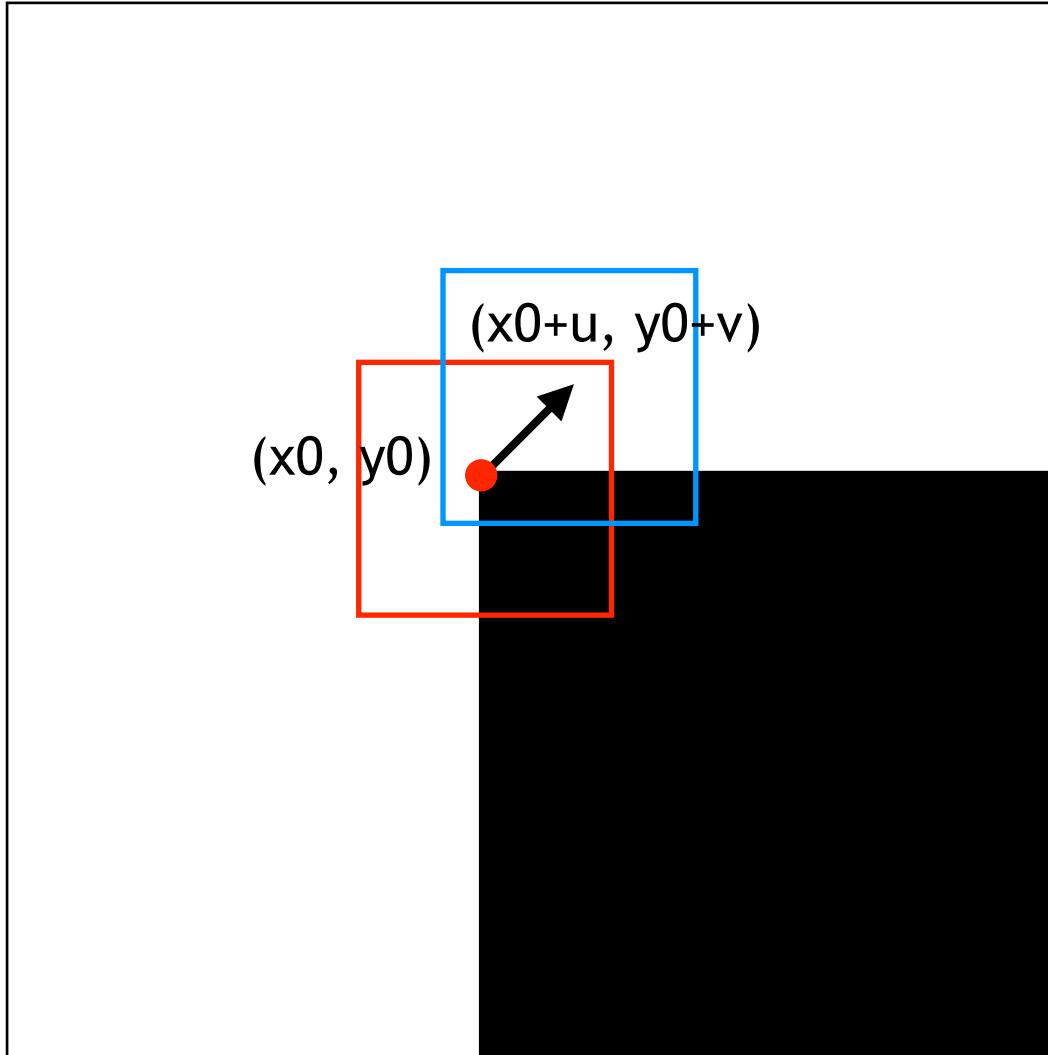
Original image

# The Basic Idea of Harris Corner



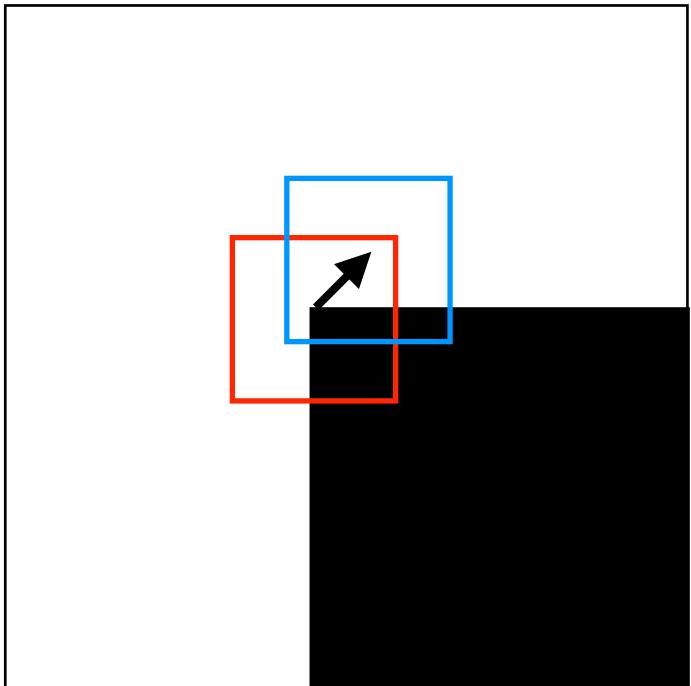
Local neighborhood of  
a corner point  $(x_0, y_0)$

# The Basic Idea of Harris Corner

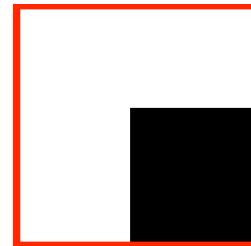


Move along direction  
( $u, v$ )

# The Basic Idea of Harris Corner

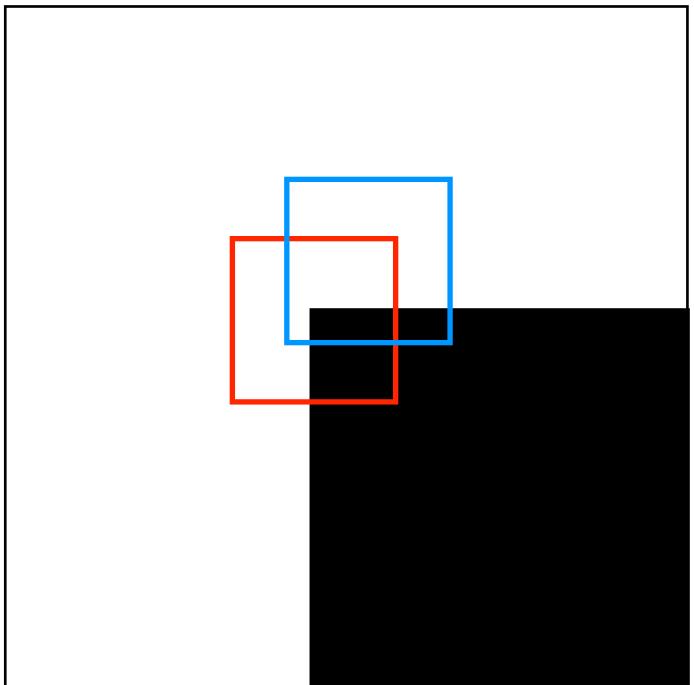


Local neighborhood of  
a corner point  $(x_0, y_0)$



Local neighborhood of  
point  $(x_0+u, y_0+v)$

# The Basic Idea of Harris Corner



Change along direction  $(u, v) =$

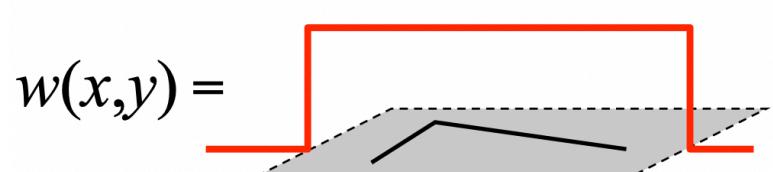
$$= \sum_{(x,y) \in N} [I(x+u, y+v) - I(x, y)]^2$$

2

Where  $N$  is the neighborhood of  $(x_0, y_0)$

# Notation

Rectangle Window Function



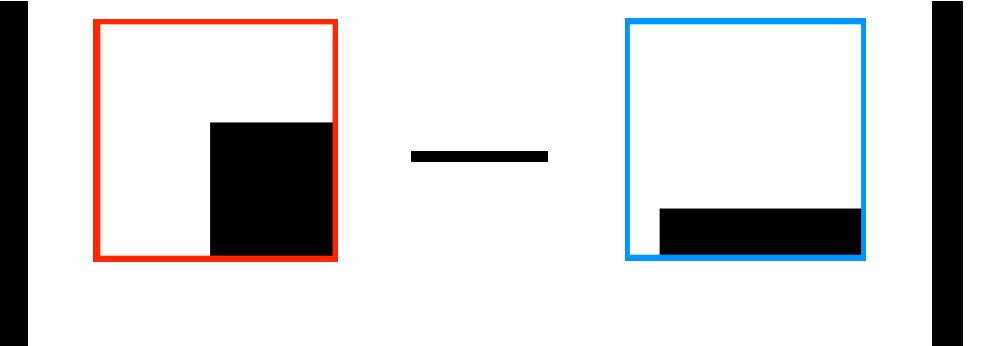
**1 in window, 0 outside**

$$w'(x, y) = w(x - x_0, y - y_0)$$

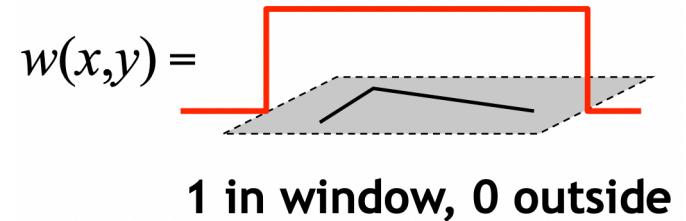
Square intensity difference

$$D(x, y) = [I(x + u, y + v) - I(x, y)]^2$$

# The Basic Idea of Harris Corner


$$= \sum_{(x,y) \in N} [I(x+u, y+v) - I(x, y)]^2$$
$$= \sum_{x,y} w'(x, y) [I(x+u, y+v) - I(x, y)]^2$$
$$= \sum_{x,y} w'(x, y) D(x, y)$$
$$= w' * D$$

Rectangle Window Function



$$w'(x, y) = w(x - x_0, y - y_0)$$

Square intensity difference

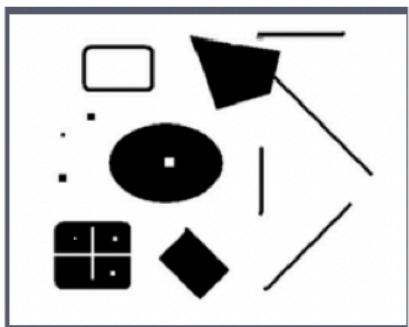
$$D(x, y) = [I(x+u, y+v) - I(x, y)]^2$$

# Harris Detector

First-order Taylor expansion:  $I[x + u, y + v] - I[x, y] \approx I_x u + I_y v$

$$\therefore D(x, y) = (I[x + u, y + v] - I[x, y])^2 \approx (I_x u + I_y v)^2 = [u, v] \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

$$\therefore E_{(x_0, y_0)}(u, v) = w' * D = [u, v] w' * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$



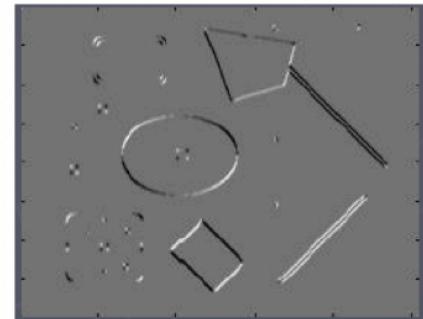
**Image  $I$**



$I_x$



$I_y$



$I_x I_y$

# Harris Detector

If we are checking the corner at  $(x_0, y_0)$ , then the change along direction  $(u_0, v_0)$  is:

$$E_{(x_0, y_0)}(u, v) \approx [u, v] M(x_0, y_0) \begin{bmatrix} u \\ v \end{bmatrix}$$

where  $M(x, y) = w' * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = \begin{bmatrix} w' * I_x^2 & w' * (I_x I_y) \\ w' * (I_x I_y) & w' * I_y^2 \end{bmatrix}$

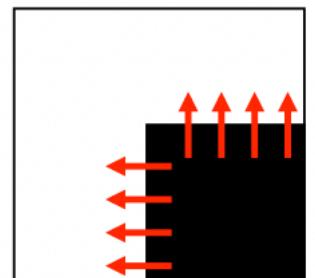
# Harris Detector

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

- $M$  is a symmetric matrix.
- $M$  is a positive semi-definite matrix. (since all its principle minors  $\geq 0$ .)
- Simple case:  $M$  is diagonal at  $(x_0, y_0)$  :  $M(x_0, y_0) = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$  ( $\lambda_1 \geq 0$ ,  $\lambda_2 \geq 0$ )

$$\therefore E_{(x_0, y_0)}(u, v) \approx [u, v] M(x_0, y_0) \begin{bmatrix} u \\ v \end{bmatrix} = \lambda_1 u^2 + \lambda_2 v^2$$

- This corresponds to an axis-aligned corner.
- If either  $\lambda \approx 0$ , this is not a corner.



# Harris Detector

- General case:
  - since  $M$  is a symmetric matrix, perform eigendecomposition:

$$M(x, y) = w' * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R \quad (\lambda_1 \geq 0, \lambda_2 \geq 0)$$

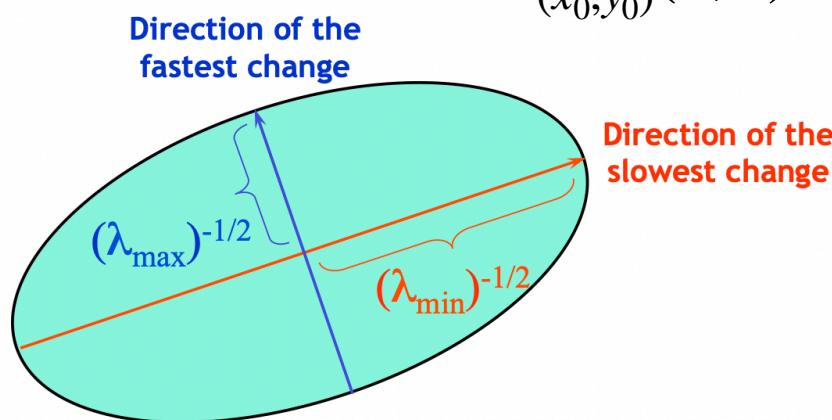
R is an orthogonal matrix,  $\lambda$ s are the eigenvalues of M!

# Harris Detector

- General case: since  $M$  is a symmetric matrix, perform eigen-decomposition:

$$M(x, y) = w'^* \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R \quad (\lambda_1 \geq 0, \lambda_2 \geq 0)$$

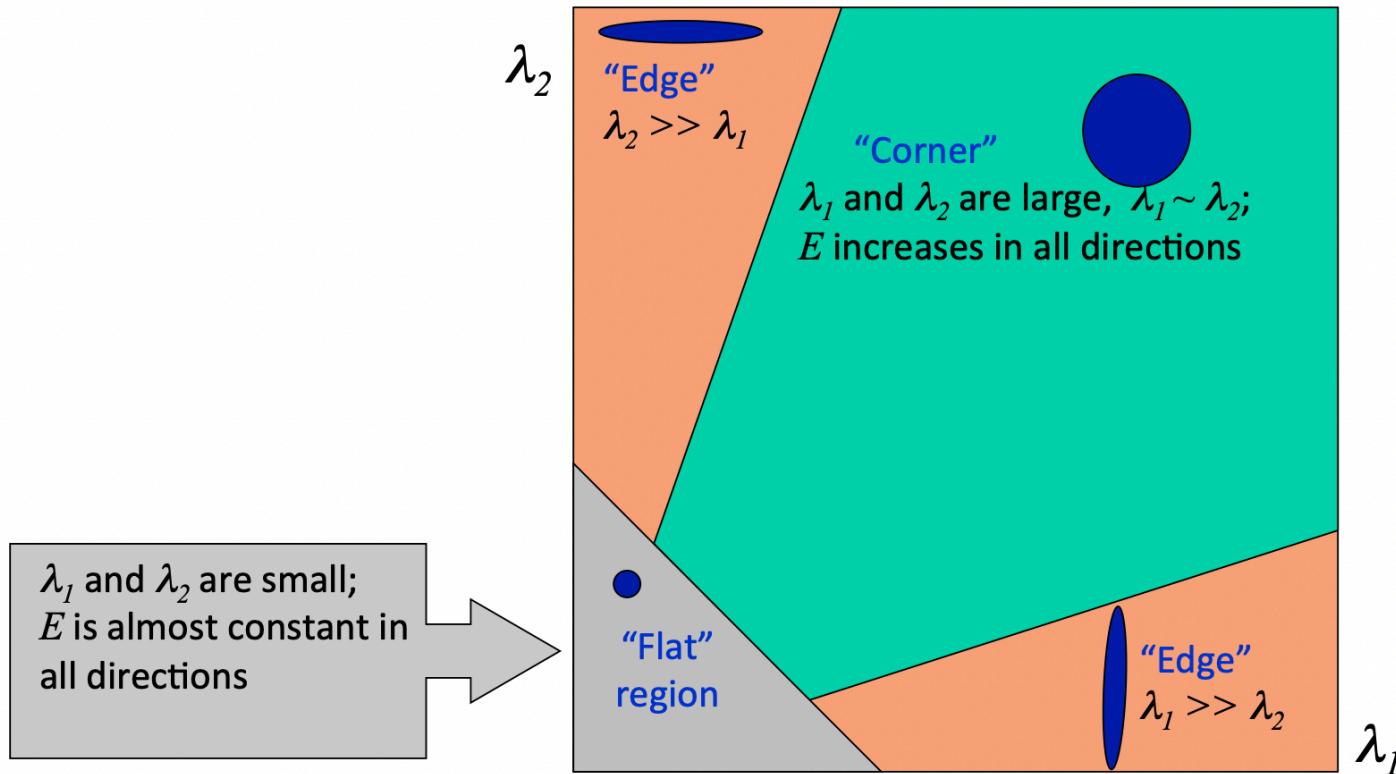
$$\therefore E_{(x_0, y_0)}(u, v) \approx \lambda_1 u_R^2 + \lambda_2 v_R^2 \quad \text{where } \begin{bmatrix} u_R \\ v_R \end{bmatrix} = R \begin{bmatrix} u \\ v \end{bmatrix}$$



The energy landscape is a paraboloid!

# Eigenvalues

- Classification of the type of the image point according to the eigenvalues of  $M$ .



Two conditions must be satisfied:

$$\lambda_1, \lambda_2 > b$$

$$\frac{1}{k} < \frac{\lambda_1}{\lambda_2} < k$$

# Corner Response Function $\theta$

- Fast approximation:

$$\theta = \frac{1}{2}(\lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2) + \frac{1}{2}(\lambda_1\lambda_2 - 2t)$$

$$\frac{1}{k} < \frac{\lambda_1}{\lambda_2} < k$$

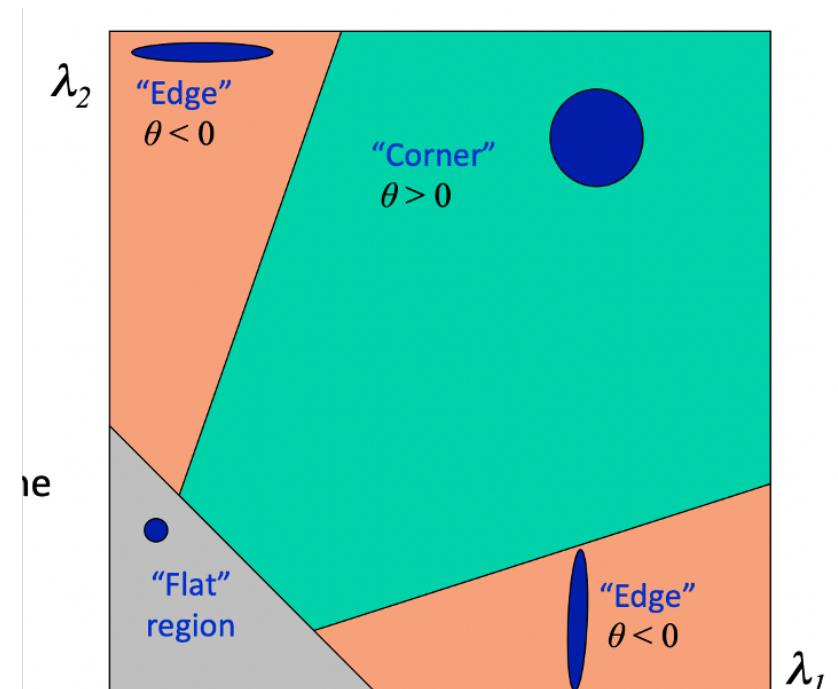
$$\lambda_1, \lambda_2 > b$$

$$= \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2 - t$$

$$= \underline{\det(M)} - \underline{\alpha \text{Trace}(M)^2} - t$$

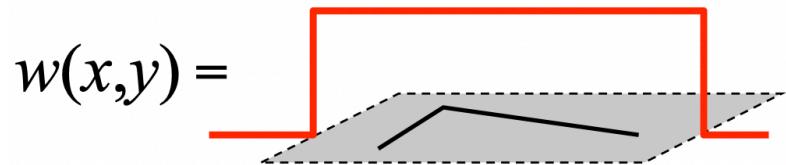
Orthogonal transformation won't change  
the determinant and trace of a matrix

$\alpha$  in [0.04,0.06]



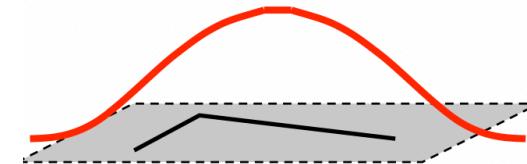
# Choices of Window Functions

Rectangle window



1 in window, 0 outside

or



Gaussian

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

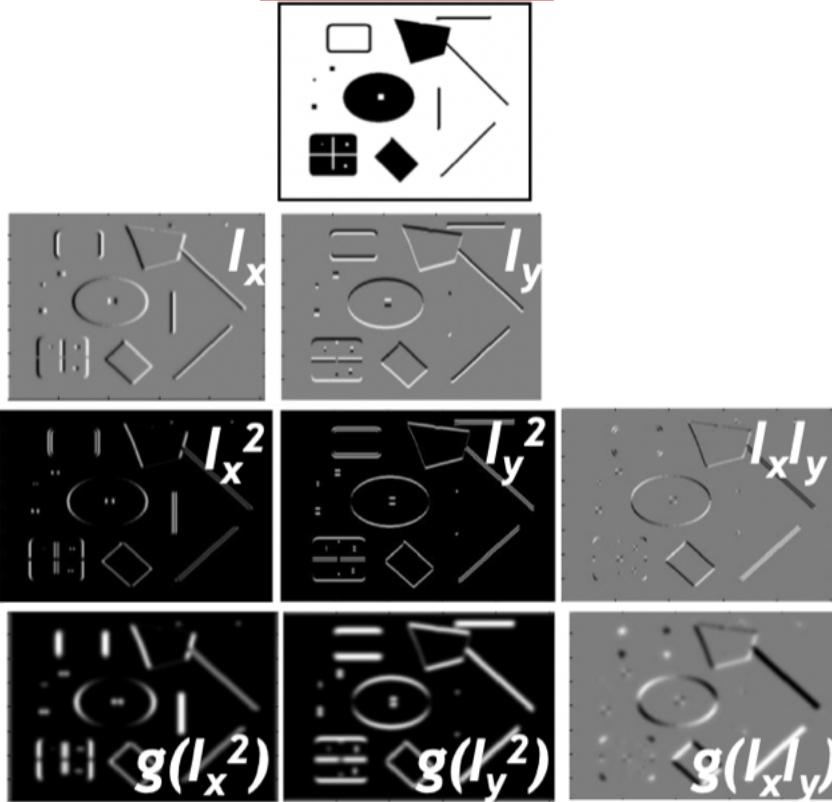
Not rotation-invariant.

$$M(x, y) = g(\sigma) * \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Rotation-invariant.

# Summary of Harris Detector

1. Image derivatives



2. Square of derivatives

3. Rectangle window or Gaussian filter

4. Corner response function

$$\theta = g(I_x^2)g(I_y^2) - [g(I_xI_y)]^2 - \alpha[g(I_x^2) + g(I_y^2)]^2 - t$$

5. Non-maximum suppression



# Step-by-Step Harris Detector

- Input: two images



Image borrowed from Stanford CS131

# Step-by-Step Harris Detector

- Compute corner response  $\theta$

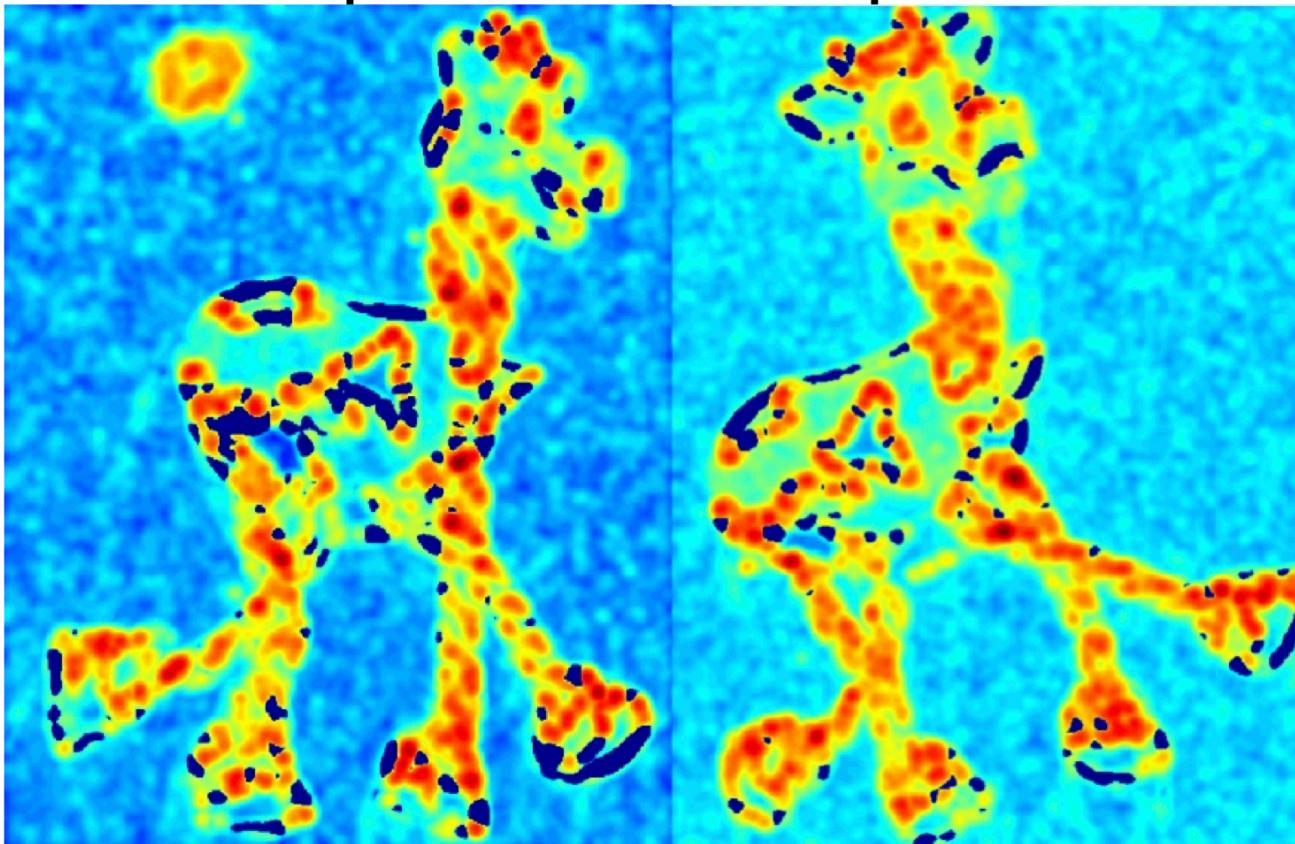


Image borrowed from Stanford CS131

# Step-by-Step Harris Detector

- Thresholding and perform non-maximal suppression



# Step-by-Step Harris Detector

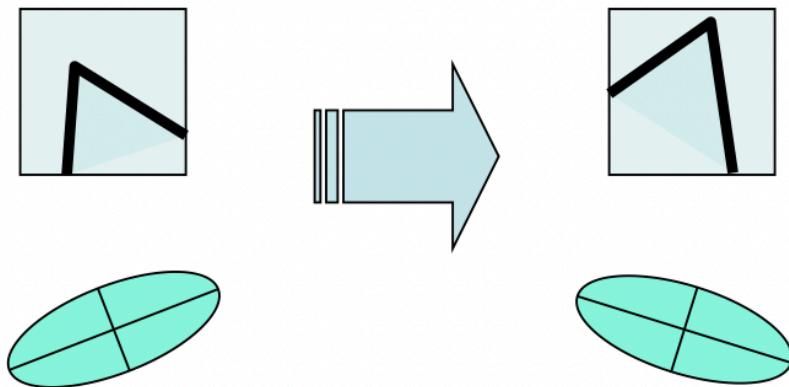
- Results



Image borrowed from Stanford CS131

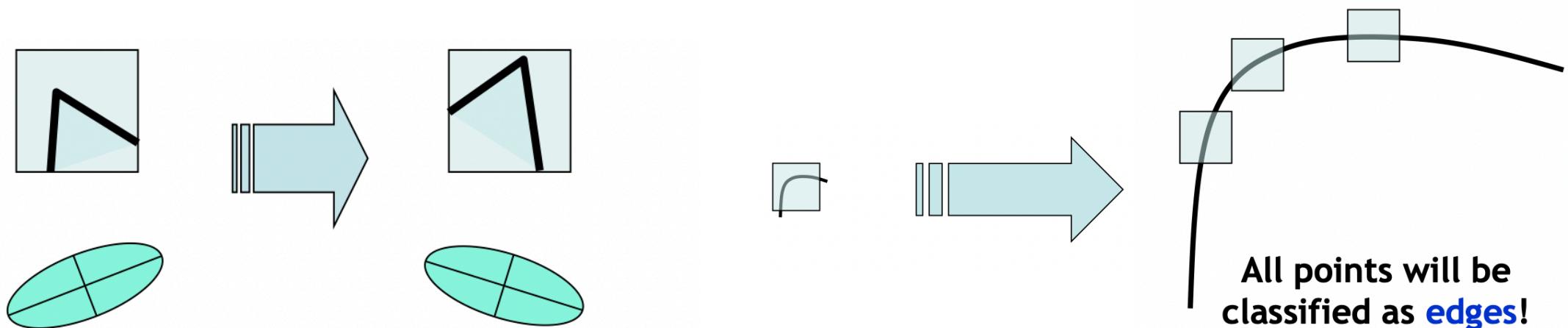
# Properties of Harris Detector

- Corner response is equivariant with both translation and image rotation.



# Properties of Harris Detector

- Corner response is equivariant with both translation and image rotation.
- Not invariant to scale.

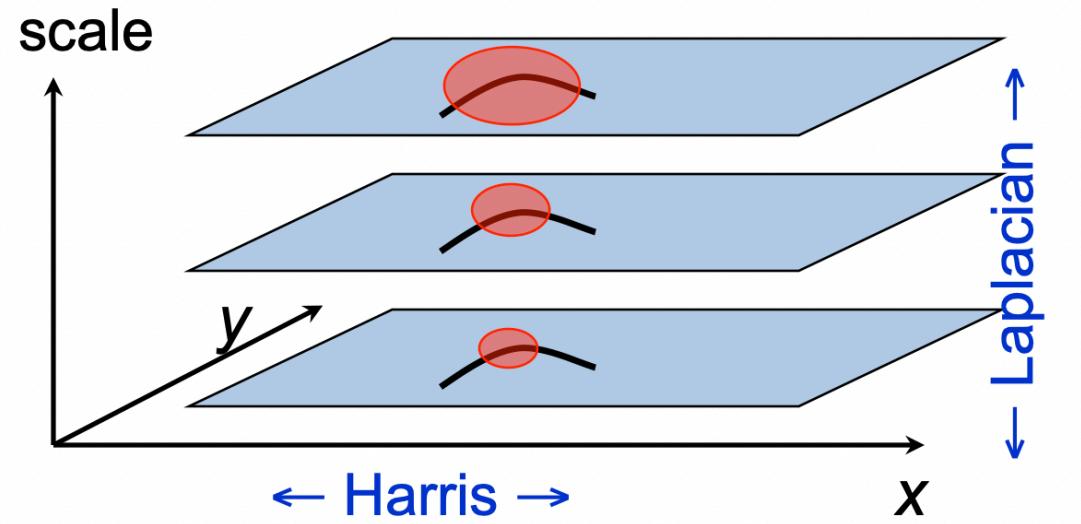


# Scale Invariant Detectors

- **Harris-Laplacian<sup>1</sup>**

*Find local maximum of:*

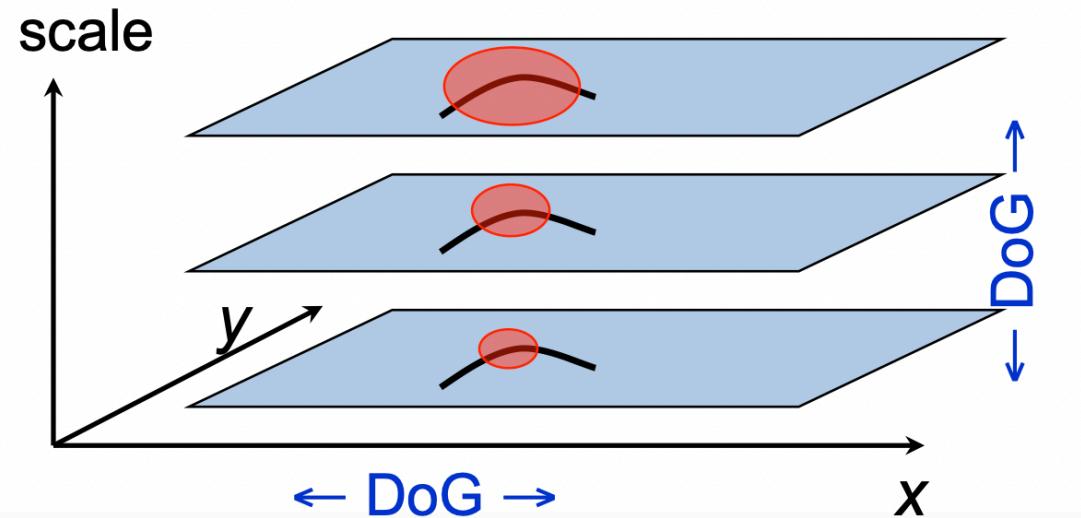
- Harris corner detector in space (image coordinates)
- Laplacian in scale



- **SIFT (Lowe)<sup>2</sup>**

*Find local maximum of:*

- Difference of Gaussians in space and scale





# Introduction to Computer Vision

Next week: Lecture 3,  
Classic Vision Methods II