

# **Computer Vision**

**Kumar Bipin**

**BE, MS, PhD (MMTU, IISc, IIIT-Hyderabad) (Robotics Control and Computer Vision)**

**Motorola, STMicroelectronics, Tata Elxsi (Technical Manager)**

# Edge Detection

---

Convert a 2D image into a set of points where image intensity changes rapidly.

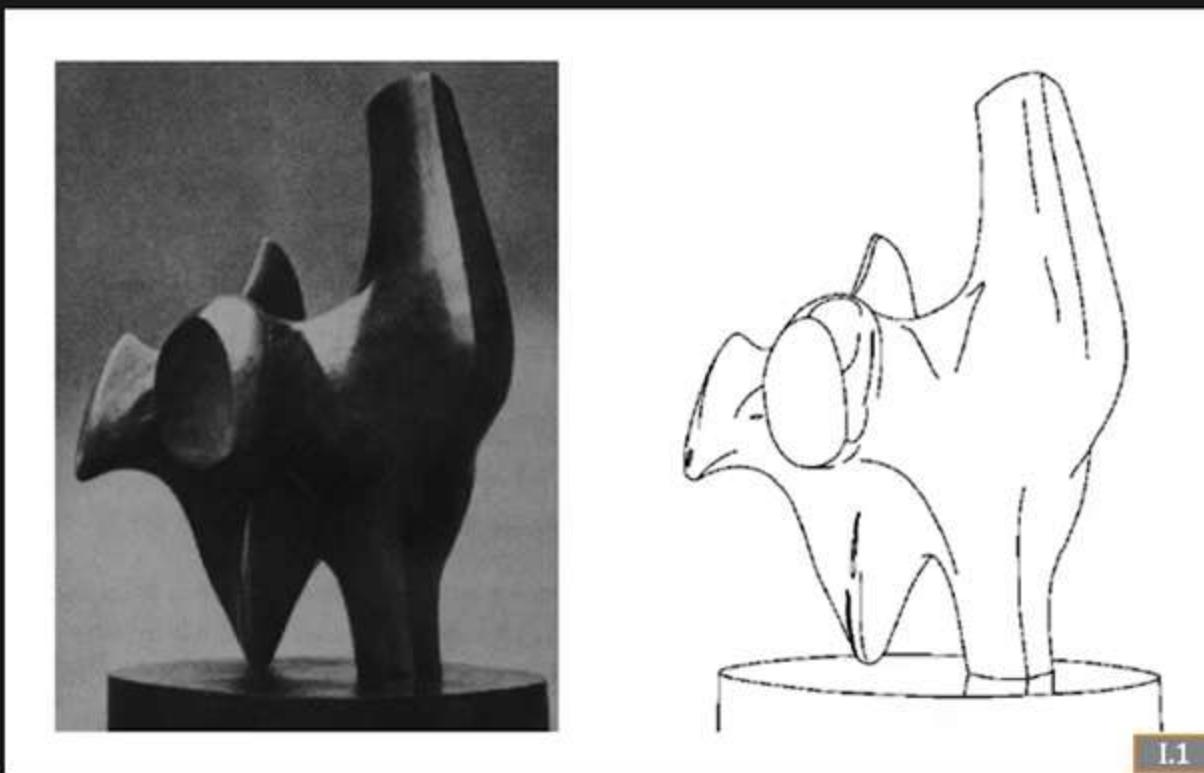
## Topics:

- (1) What is an Edge?
- (2) Edge Detection Using Gradients
- (3) Edge Detection Using Laplacian
- (4) Canny Edge Detector
- (5) Corner Detection

# What is an Edge?

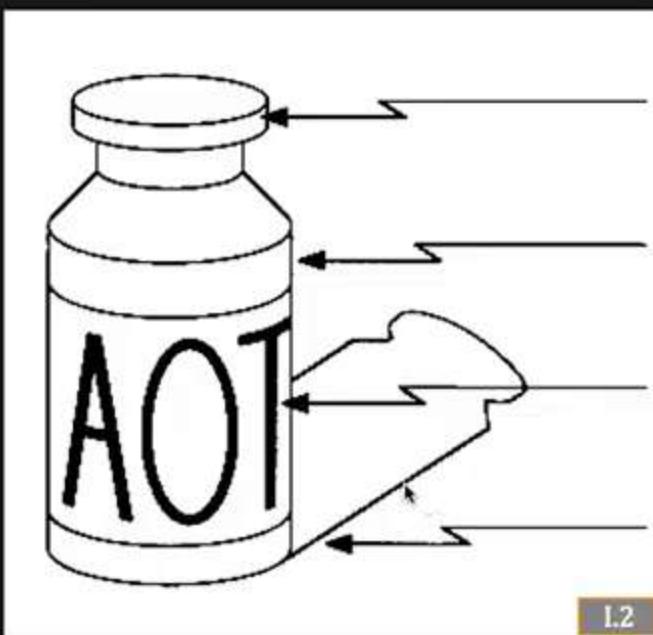
---

Rapid change in image intensity within small region



# Causes of Edges

Rapid changes in image intensity are caused by various physical phenomena.



Surface Normal Discontinuity

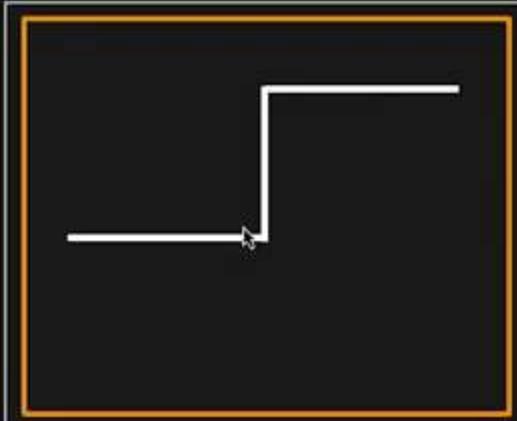
Depth Discontinuity

Surface Reflectance Discontinuity

Illumination Discontinuity

# Types of Edges

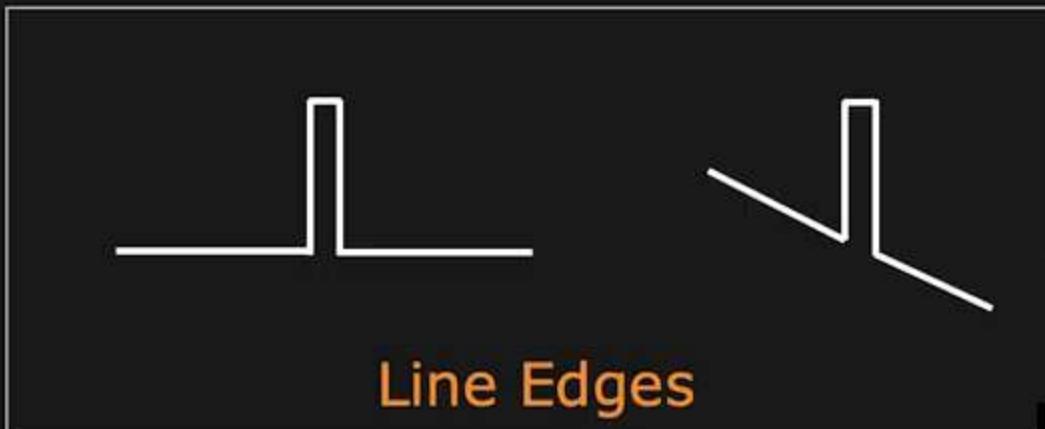
---



Step Edges



Roof Edge



Line Edges

# Real Edges

---



Problems: Noisy Images and Discrete Images

# Edge Detector

---

We want an **Edge Operator** that produces:

- Edge **Position**
- Edge **Magnitude** (Strength)
- Edge **Orientation** (Direction)

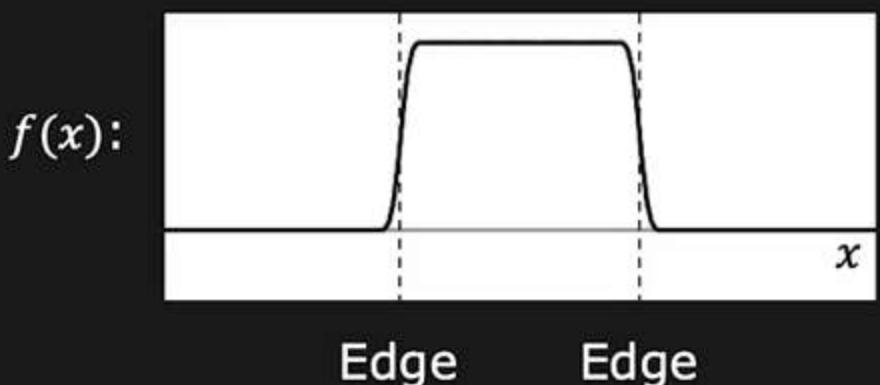
Performance Requirements:

- High **Detection Rate**
- Good **Localization**
- Low **Noise Sensitivity**

# 1D Edge Detection

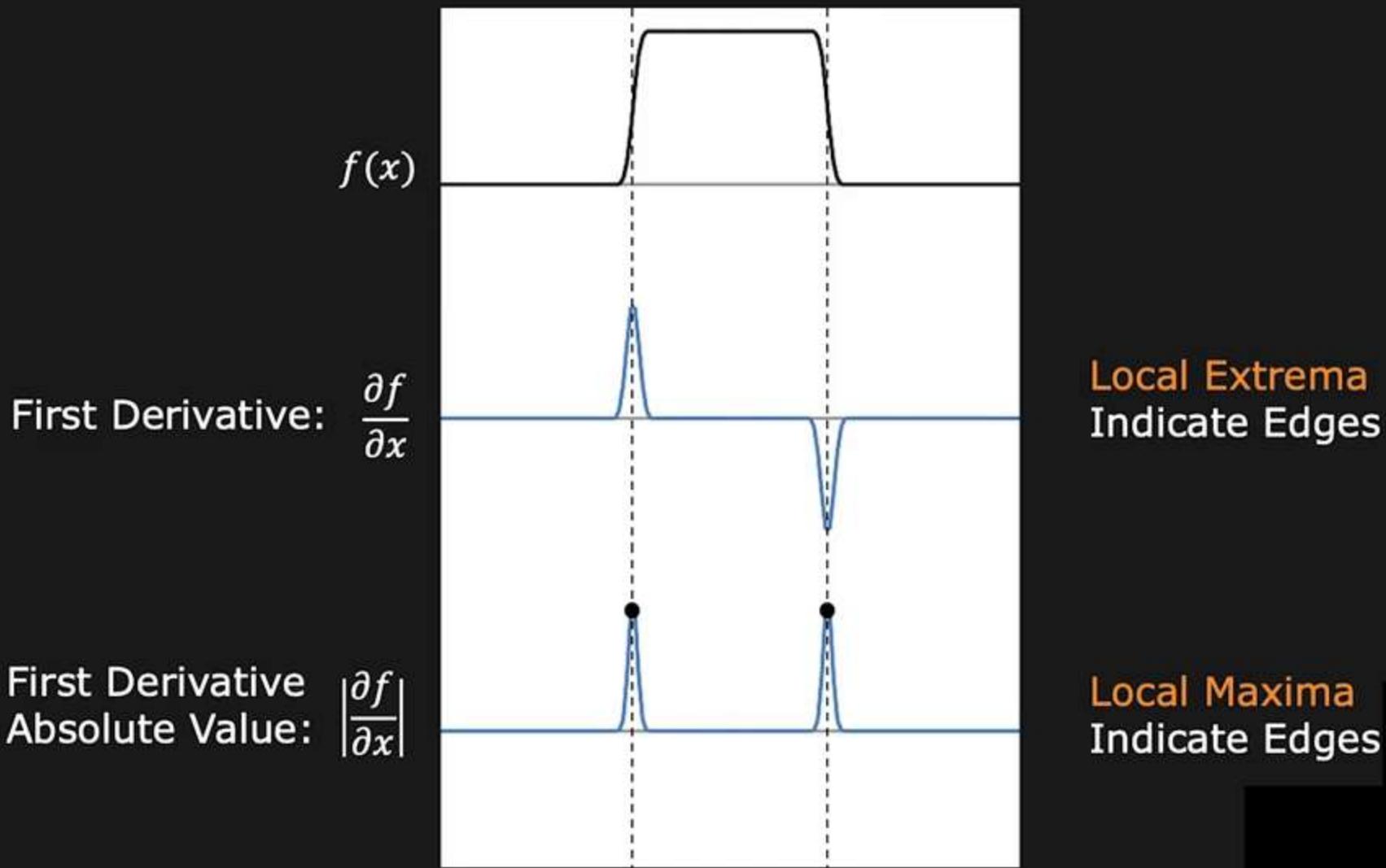
---

Edge is a rapid change in image intensity in a small region.



Basic Calculus: **Derivative** of a continuous function represents the amount of change in the function.

# Edge Detection Using 1<sup>st</sup> Derivative



Provides Both Location and Strength of an Edge

# 2D Edge Detection

---



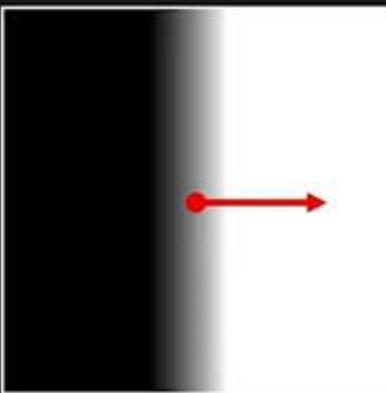
Basic Calculus: **Partial Derivatives** of a 2D continuous function represents the amount of change along each dimension.

# Gradient ( $\nabla$ )

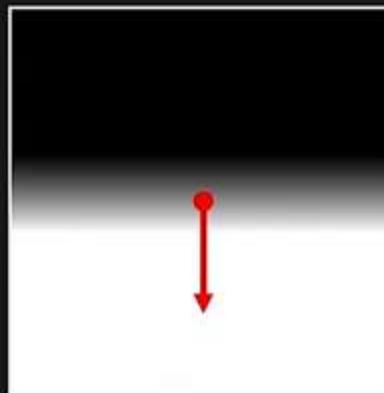
Gradient (Partial Derivatives) represents the direction of most rapid change in intensity

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

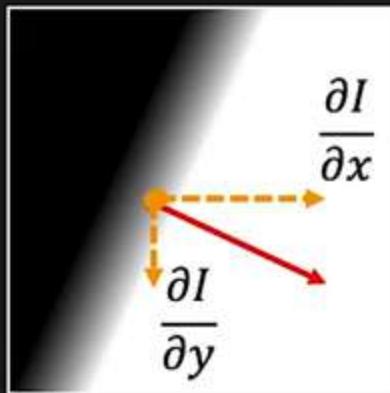
Pronounced as "Del I"



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



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



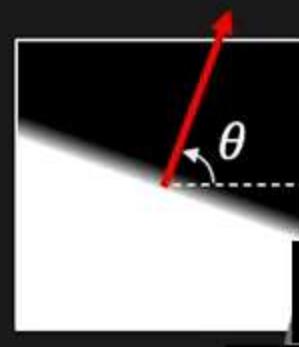
$$\nabla I = \left[ \frac{\partial I}{\partial x}, \frac{\partial I}{\partial y} \right]$$

# Gradient ( $\nabla$ ) as Edge Detector

---

Gradient Magnitude  $S = \|\nabla I\| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2}$

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

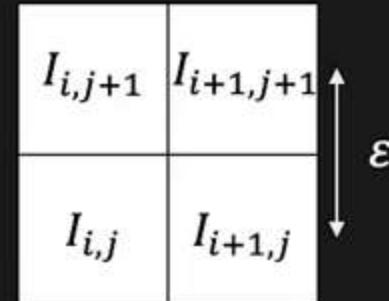


# Discrete Gradient ( $\nabla$ ) Operator

Finite difference approximations:

$$\frac{\partial I}{\partial x} \approx \frac{1}{2\varepsilon} ((I_{i+1,j+1} - I_{i,j+1}) + (I_{i+1,j} - I_{i,j}))$$

$$\frac{\partial I}{\partial y} \approx \frac{1}{2\varepsilon} ((I_{i+1,j+1} - I_{i+1,j}) + (I_{i,j+1} - I_{i,j}))$$



Can be implemented as Convolution!

$$\frac{\partial}{\partial x} \approx \frac{1}{2\varepsilon} \begin{array}{|c|c|} \hline -1 & 1 \\ \hline -1 & 1 \\ \hline \end{array}$$

$$\frac{\partial}{\partial y} \approx \frac{1}{2\varepsilon} \begin{array}{|c|c|} \hline 1 & 1 \\ \hline -1 & -1 \\ \hline \end{array}$$

Note: Convolution flips have been applied

# Comparing Gradient ( $\nabla$ ) Operators

Gradient	Roberts	Prewitt	Sobel (3x3)	Sobel (5x5)
$\frac{\partial I}{\partial x}$	$\begin{matrix} 0 & 1 \\ -1 & 0 \end{matrix}$	$\begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{matrix}$	$\begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix}$	$\begin{matrix} -1 & -2 & 0 & 2 & 1 \\ -2 & -3 & 0 & 3 & 2 \\ -3 & -5 & 0 & 5 & 3 \\ -2 & -3 & 0 & 3 & 2 \\ -1 & -2 & 0 & 2 & 1 \end{matrix}$
$\frac{\partial I}{\partial y}$	$\begin{matrix} 1 & 0 \\ 0 & -1 \end{matrix}$	$\begin{matrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{matrix}$	$\begin{matrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{matrix}$	$\begin{matrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 3 & 5 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 \\ -2 & -3 & -5 & -3 & -2 \\ -1 & -2 & -3 & -2 & -1 \end{matrix}$

Good Localization

Noise Sensitive

Poor Detection

Poor Localization

Less Noise Sensitive

Good Detection



# Gradient ( $\nabla$ ) Using Sobel Filter



Image ( $I$ )



$\partial I / \partial x$



$\partial I / \partial y$



Gradient Magnitude

# Edge Thresholding

## Standard: (Single Threshold $T$ )

$\|\nabla I(x, y)\| < T$       Definitely Not an Edge

$\|\nabla I(x, y)\| \geq T$       Definitely an Edge

**Hysteresis Based: (Two Thresholds  $T_0 < T_1$ )**

$\|\nabla I(x, y)\| < T_0$     Definitely Not an Edge

$\|\nabla I(x, y)\| \geq T_1$     Definitely an Edge

$T_0 \leq \|\nabla I(x, y)\| < T_1$  Is an Edge if a Neighboring Pixel is Definitely an Edge

# Sobel Edge Detector



Image ( $I$ )



$\partial I / \partial x$



$\partial I / \partial y$



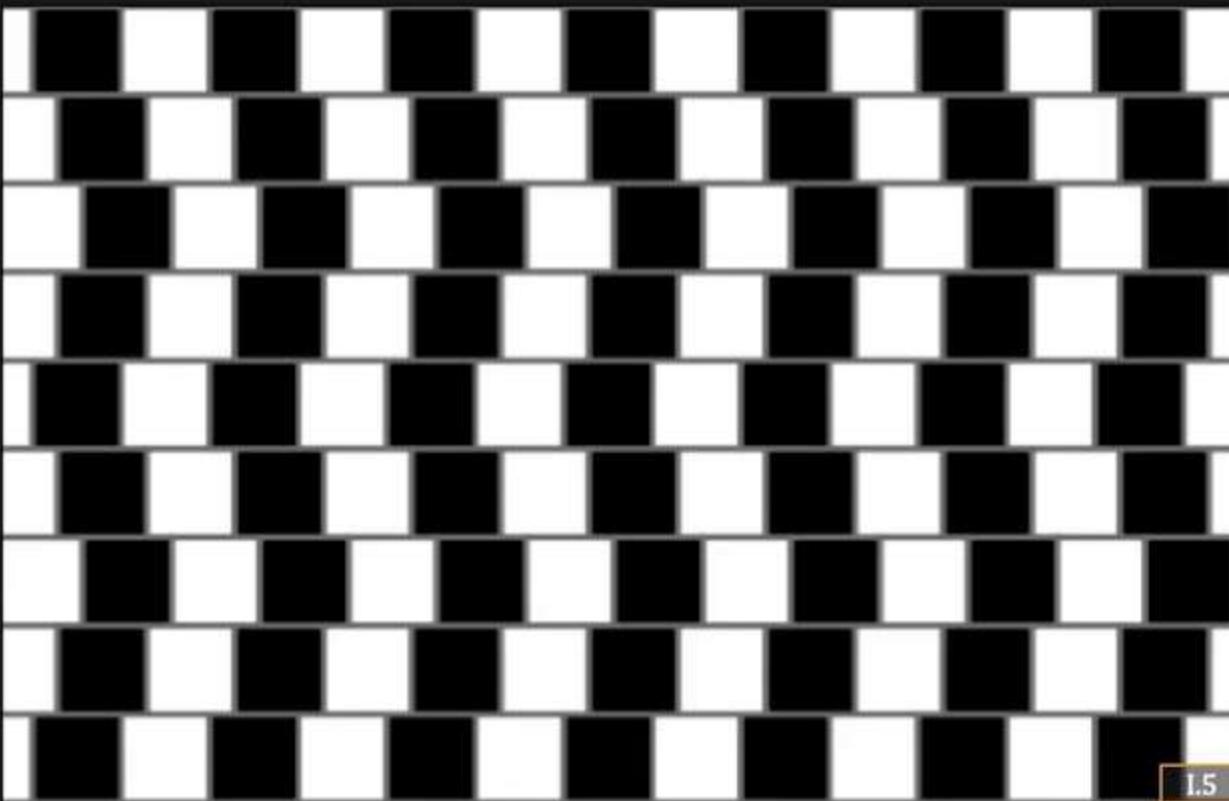
Gradient Magnitude



Thresholded Edge

# Edge Illusions: Café Wall Illusion

---



Gregory and Heard, 1979

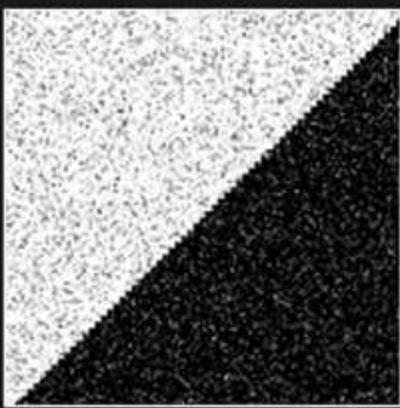
# Corners

---

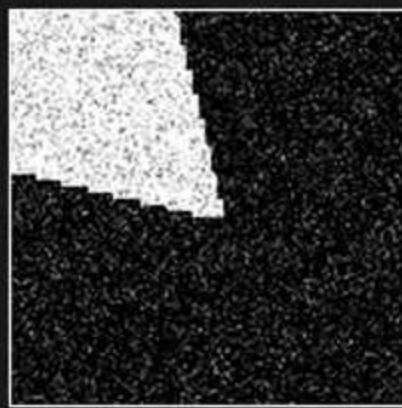
Corner: Point where Two Edges Meet. i.e., Rapid Changes of Image Intensity in Two Directions within a Small Region



"Flat" Region



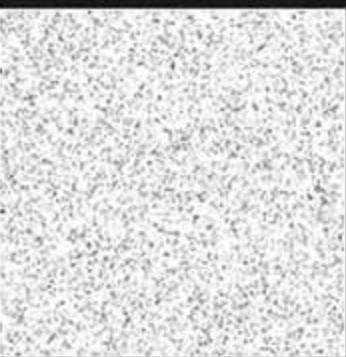
"Edge" Region



"Corner" Region

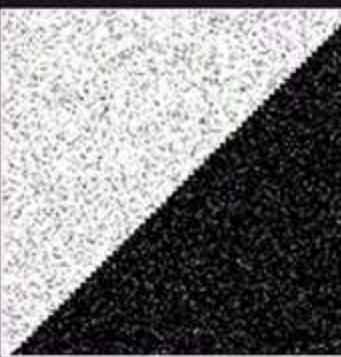
# Image Gradients

Flat Region

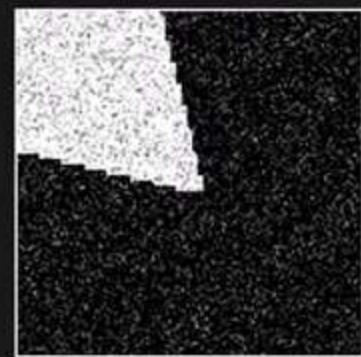


$I$

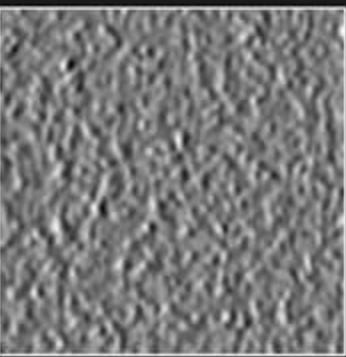
Edge Region



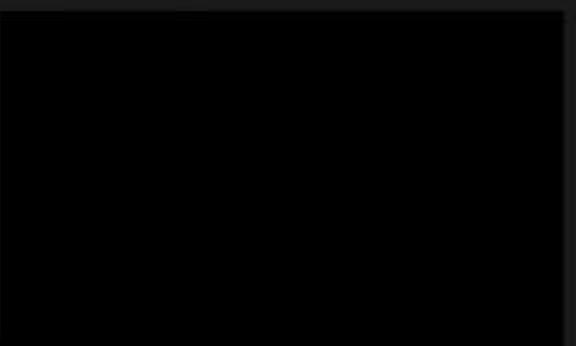
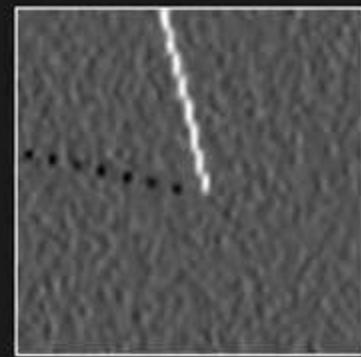
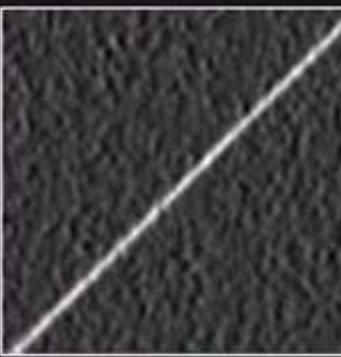
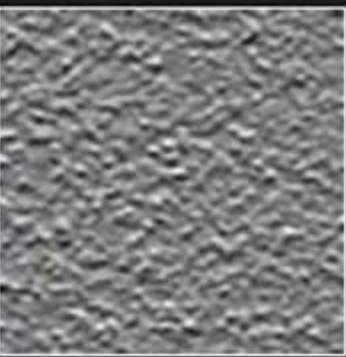
Corner Region



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

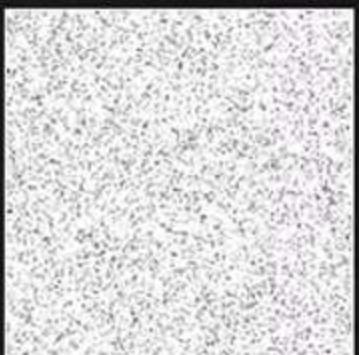


$$I_y = \frac{\partial I}{\partial y}$$

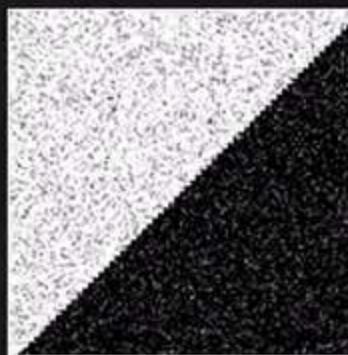


# Distribution of Image Gradients

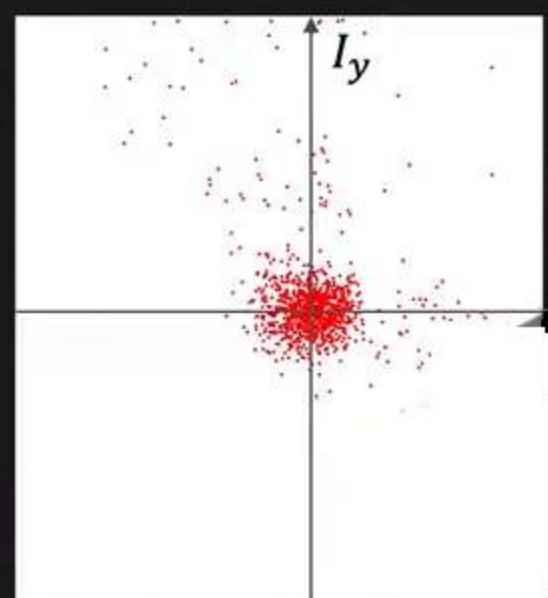
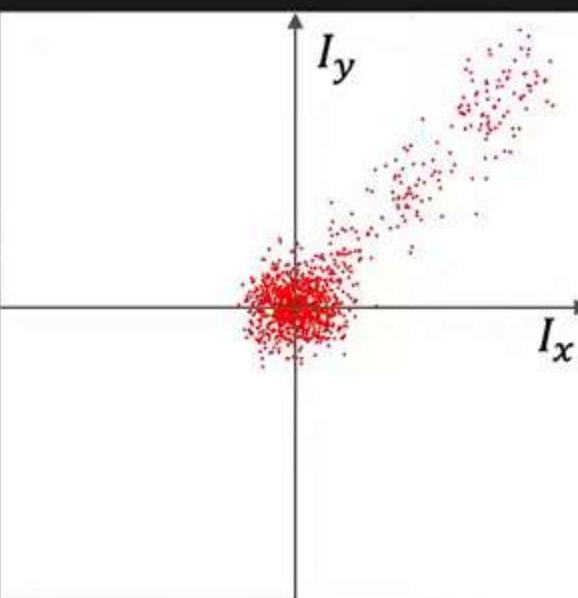
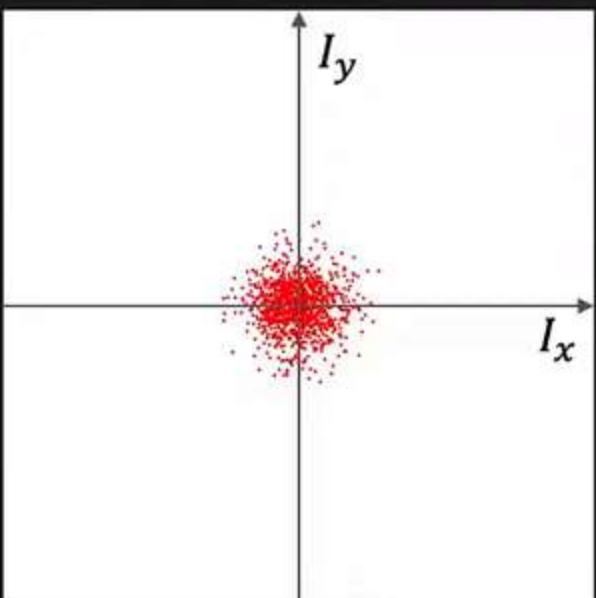
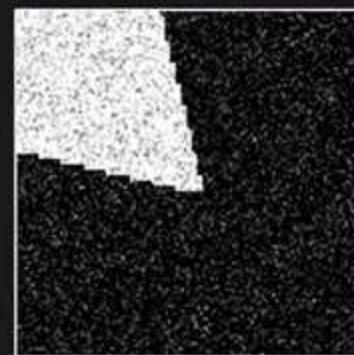
Flat Region



Edge Region



Corner Region



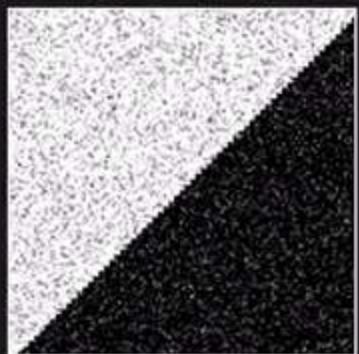
Distribution of  $I_x$  and  $I_y$  is different for all three regions.

# Fitting Elliptical Disk to Distribution

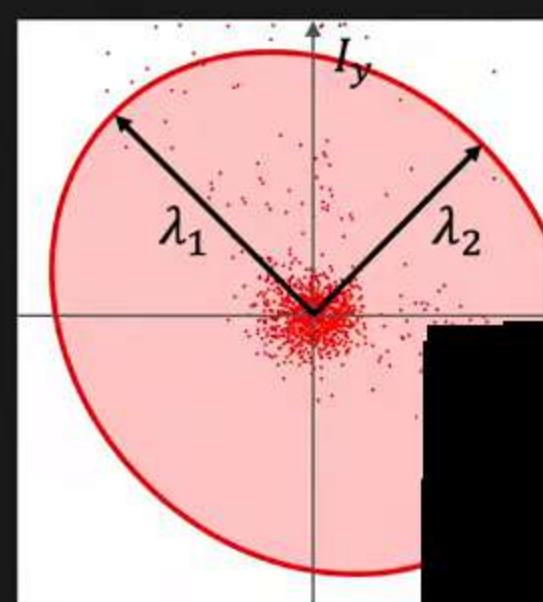
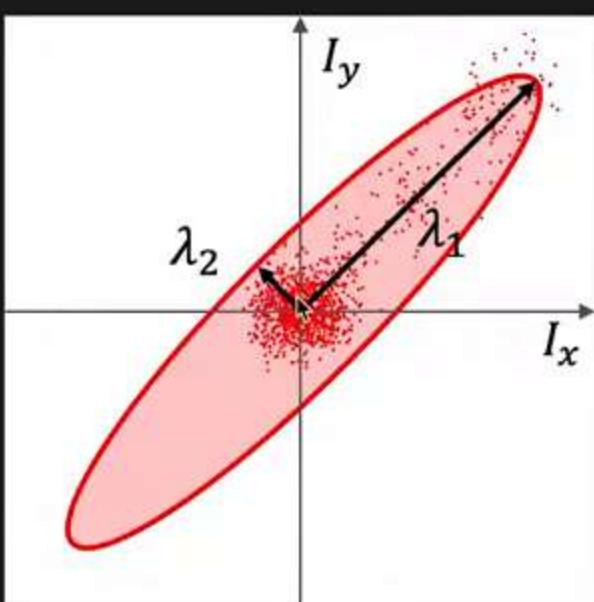
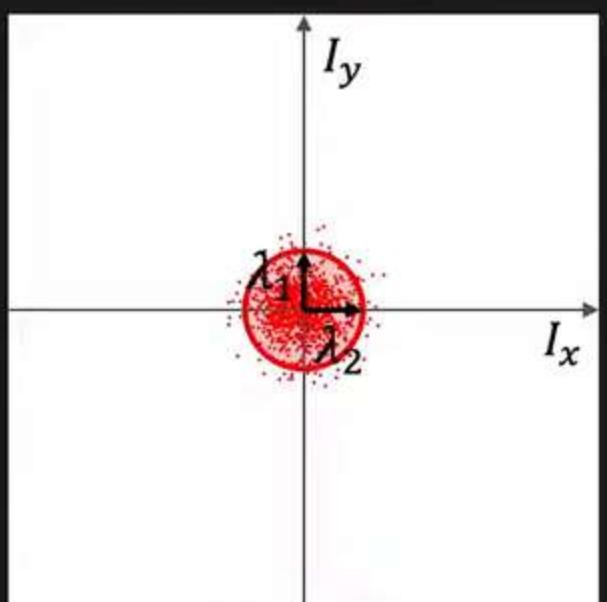
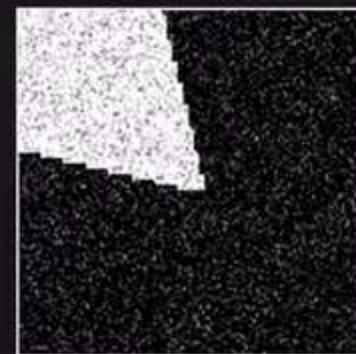
Flat Region



Edge Region



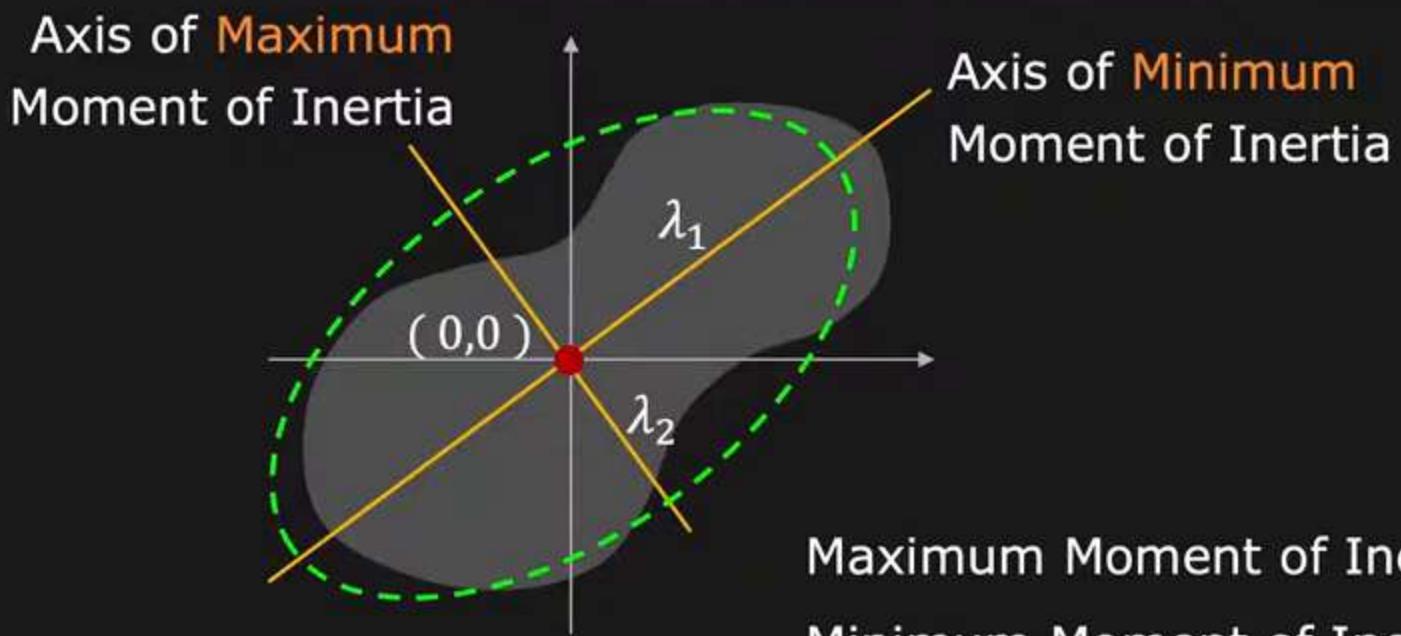
Corner Region



$\lambda_1$ : Length of Semi-Major Axis

$\lambda_2$ : Length of Semi-Minor Axis

# Fitting an Elliptical Disk



Maximum Moment of Inertia =  $E_{max}$   
Minimum Moment of Inertia =  $E_{min}$

(See lecture on Binary Images)

Length of Semi-Major Axis =  $\lambda_1 = E_{max}$

Length of Semi-Minor Axis =  $\lambda_2 = E_{min}$

# Fitting an Elliptical Disk

Second Moments for a Region:

$$a = \sum_{i \in W} (I_{x_i})^2 \quad b = 2 \sum_{i \in W} (I_{x_i} I_{y_i})$$

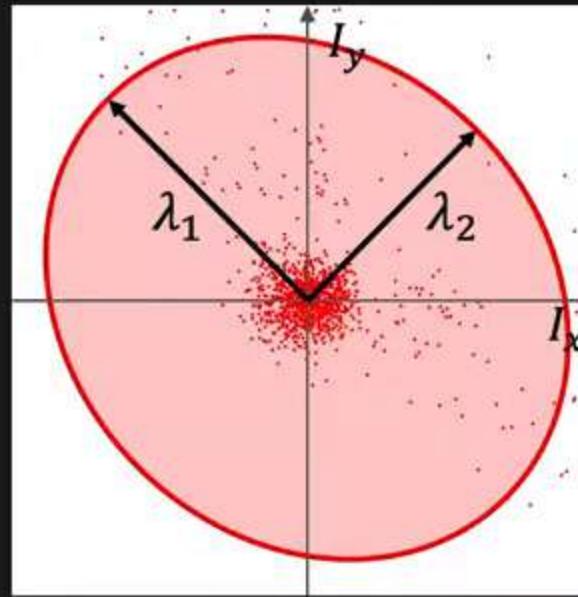
$$c = \sum_{i \in W} (I_{y_i})^2 \quad W: \text{Window centered at pixel}$$

(See lecture on Binary Images)

Ellipse Axes Lengths:

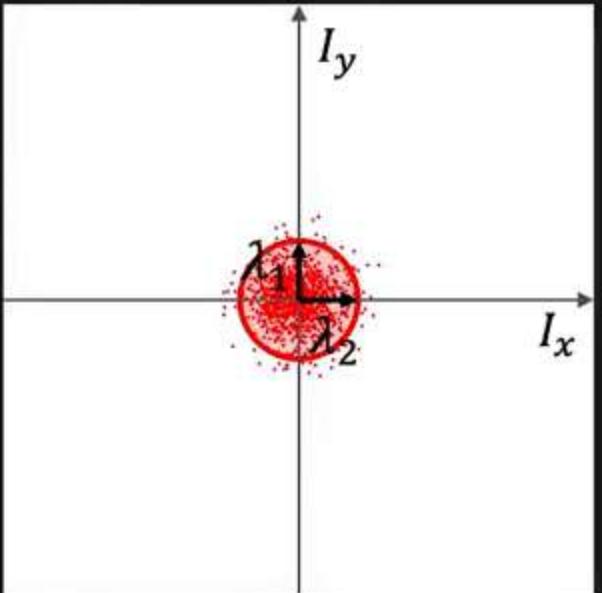
$$\lambda_1 = E_{max} = \frac{1}{2} [a + c + \sqrt{b^2 + (a - c)^2}]$$

$$\lambda_2 = E_{min} = \frac{1}{2} [a + c - \sqrt{b^2 + (a - c)^2}]$$



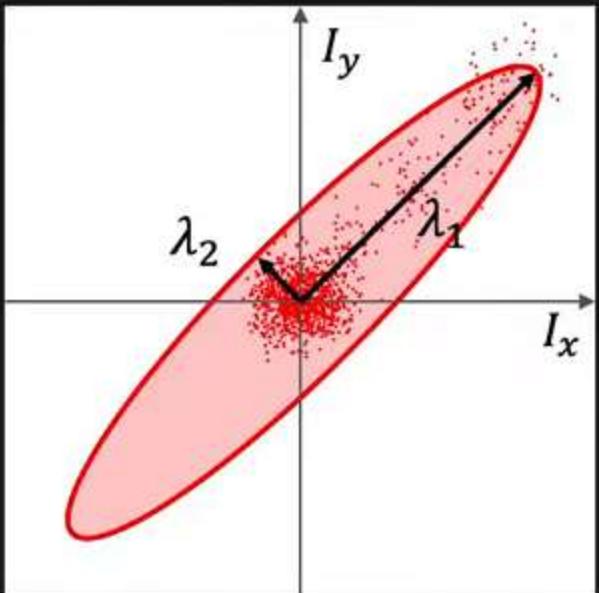
# Interpretation of $\lambda_1$ and $\lambda_2$

Flat Region



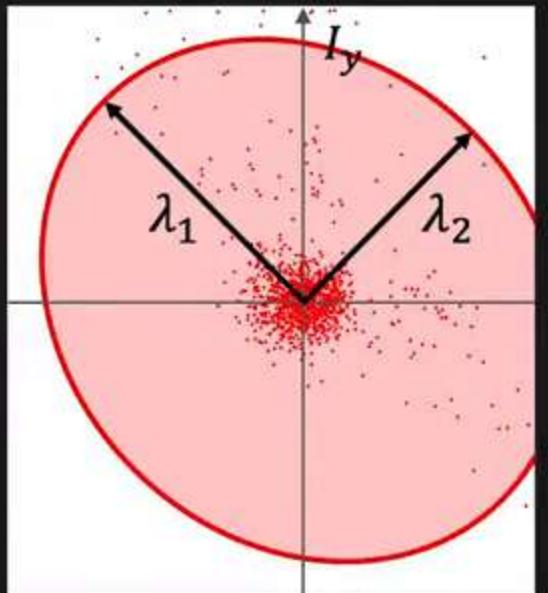
$\lambda_1 \sim \lambda_2$   
Both are Small

Edge Region



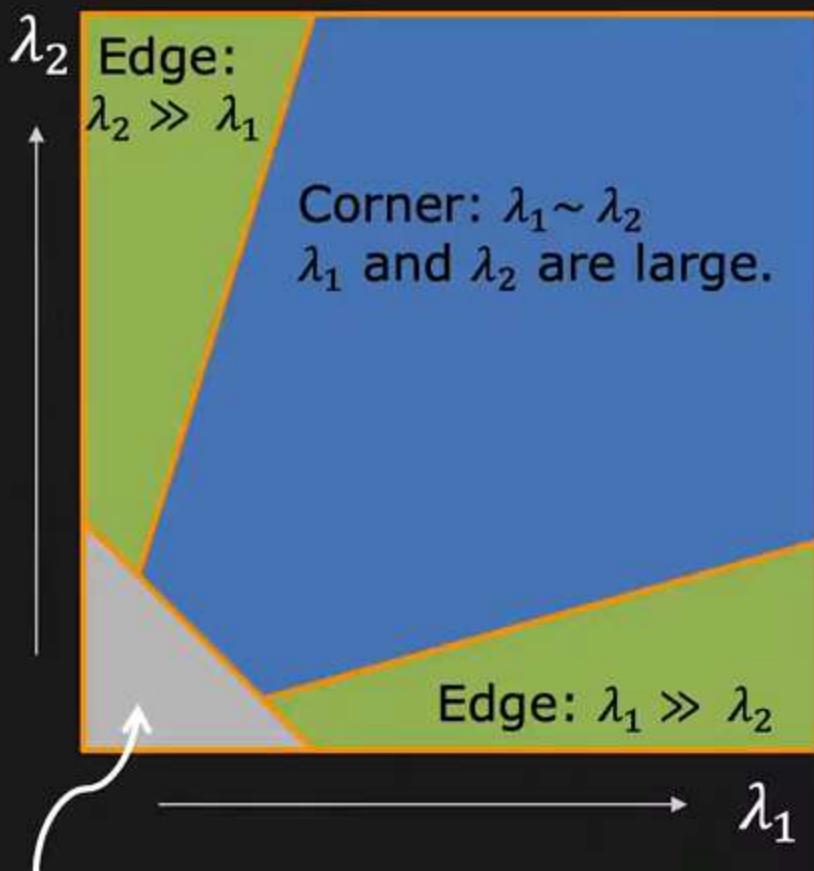
$\lambda_1 \gg \lambda_2$   
 $\lambda_1$  is Large  
 $\lambda_2$  is Small

Corner Region

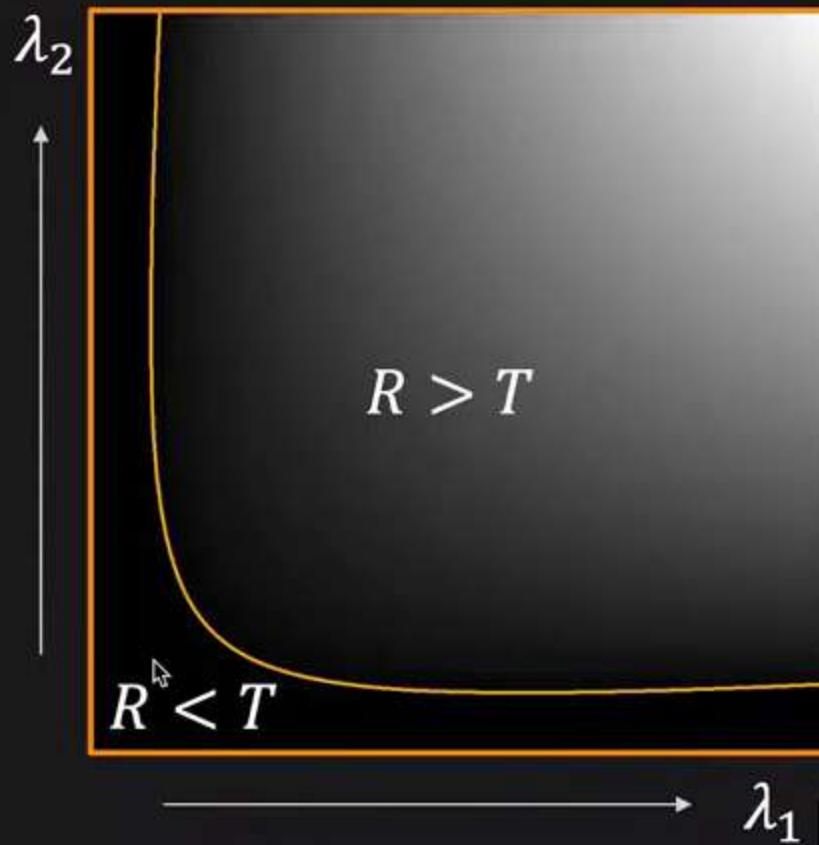


$\lambda_1 \sim \lambda_2$   
Both are La

# Harris Corner Response Function



Flat:  $\lambda_1 \sim \lambda_2$   
 $\lambda_1$  and  $\lambda_2$  are small.



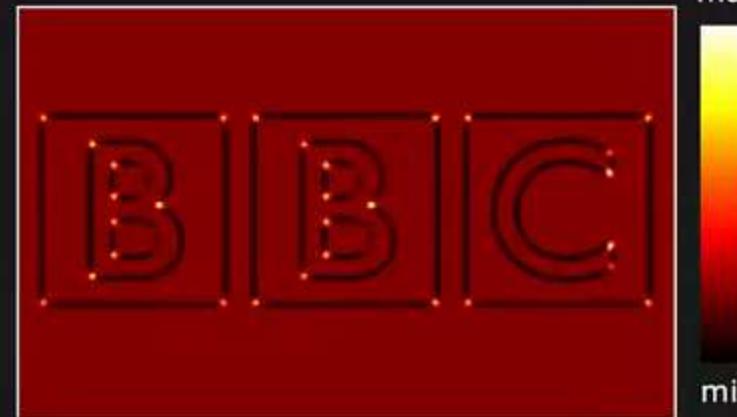
$$R = \lambda_1 \lambda_2 - k(\lambda_1 + \lambda_2)^2$$

where:  $0.04 \leq k \leq 0.5$   
(Designed Empirically)

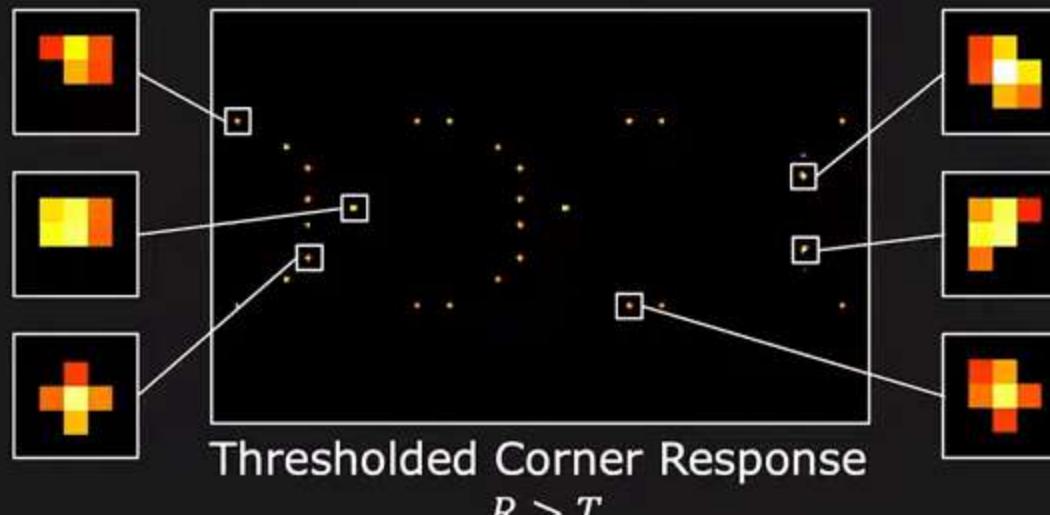
# Harris Corner Detection Example



Image



Corner Response  $R$



Thresholded Corner Response

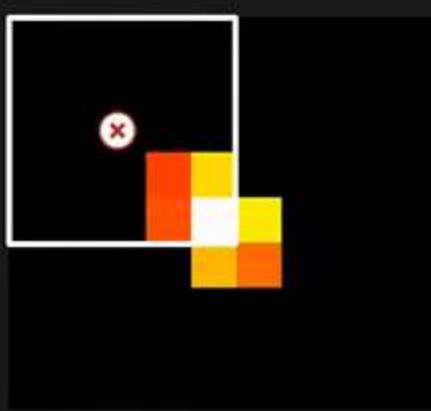
$$R > T$$

How to determine the actual corner pixel?

# Non-Maximal Suppression

---

1. Slide a window of size  $k$  over the image.
2. At each position, if the pixel at the center is the maximum value within the window, label it as positive (retain it). Else label it as negative (suppress it).



Suppress



Suppress



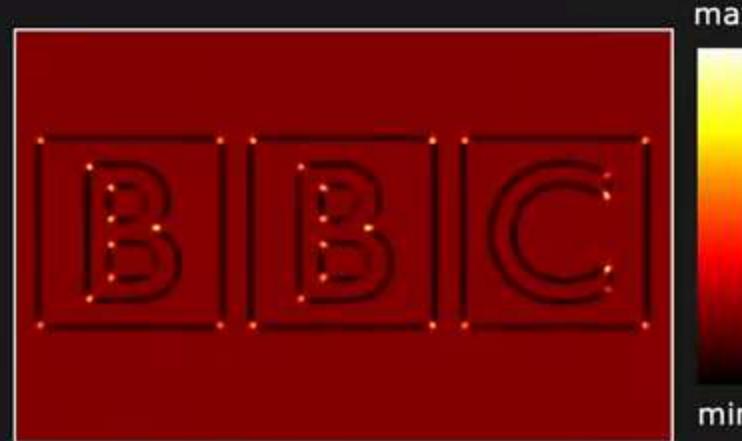
Retain

Used for finding Local Extrema (Maxima/Minima)

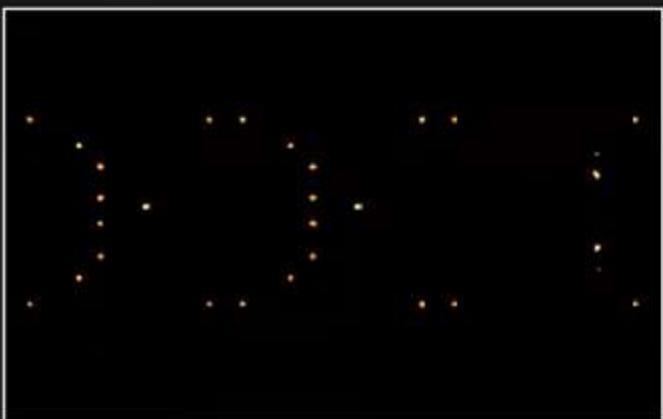
# Harris Corner Detection Example



Image



Corner Response  $R$



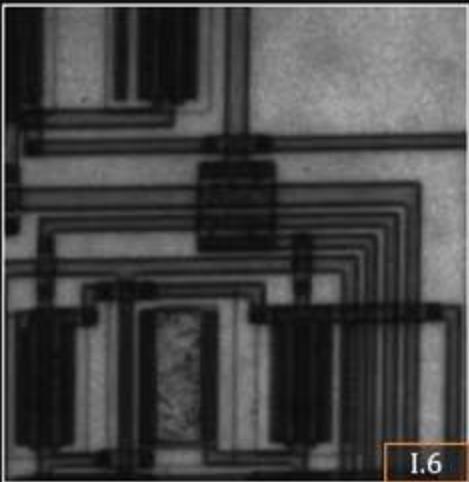
Thresholded Corner Response  
 $R > T$  ( $T = 5.1 \times 10^7$ )



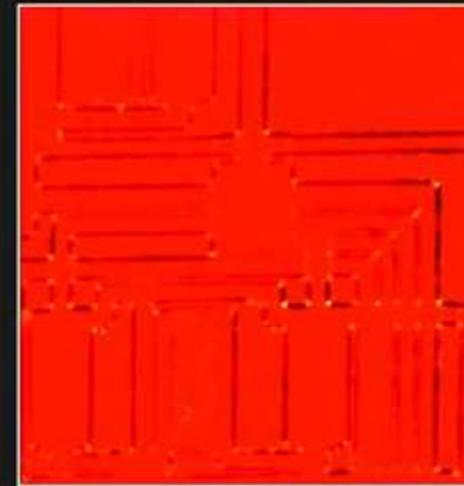
Detected Corners

# Harris Corner Detection Example

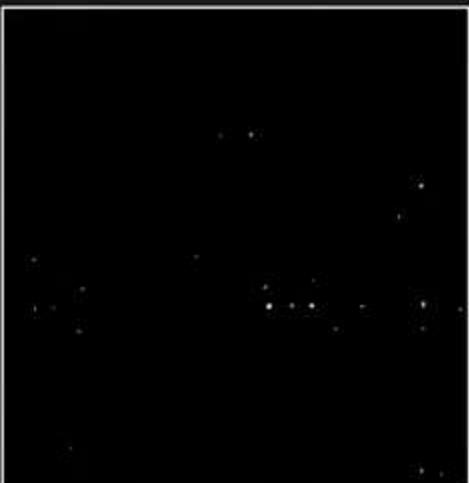
---



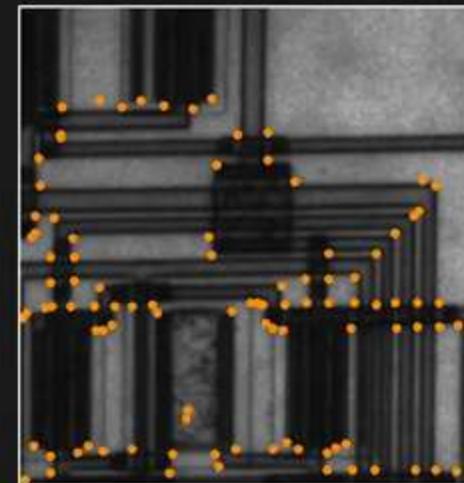
Image



Corner Response  $R$



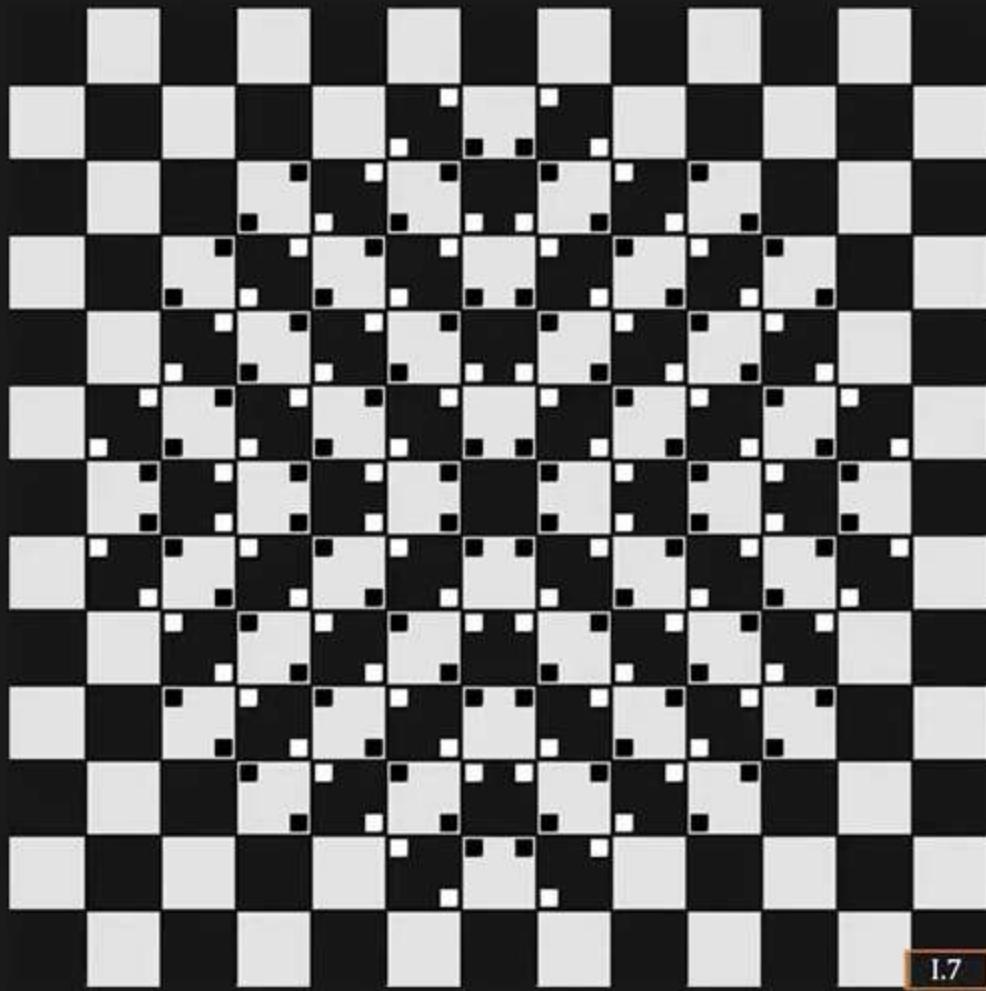
Thresholded Corner Response  
 $R > T$  ( $T = 5.1 \times 10^7$ )



Detected Corners

# Corner Illusion: The Bulge

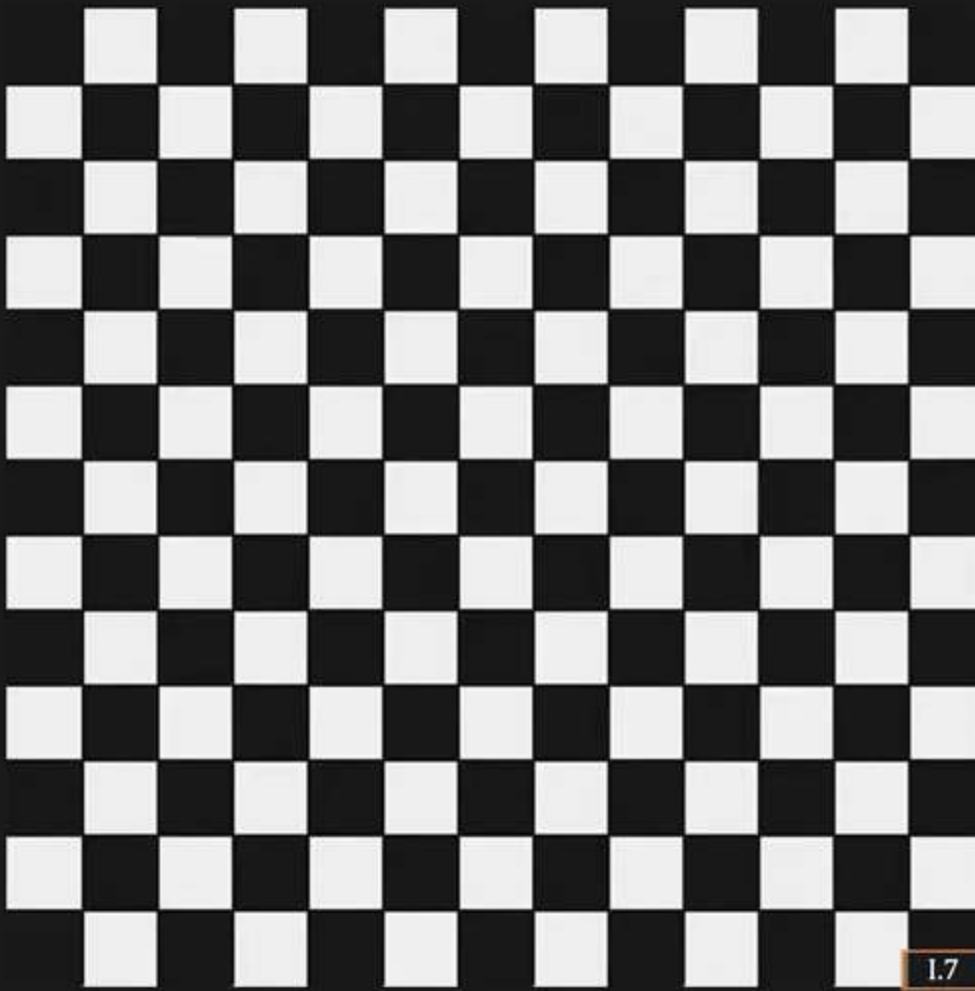
---



Kitaoka, 1998

# Corner Illusion: The Bulge

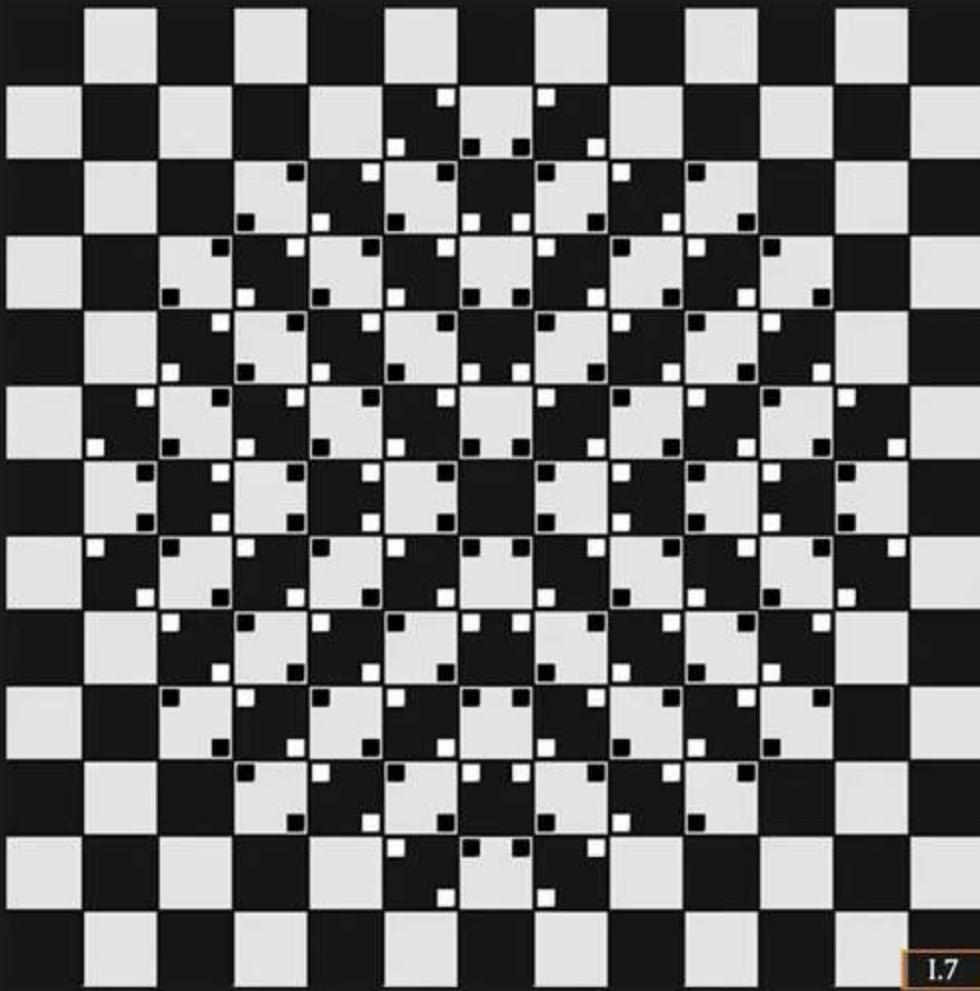
---



Kitaoka, 1998

# Corner Illusion: The Bulge

---



Kitaoka, 1998

# Boundary Detection

---

We need to find Object Boundaries from Edge Pixels.

## Topics:

- (1) Fitting Lines and Curves to Edges
- (2) Active Contours (Snakes)
- (3) The Hough Transform
- (4) Generalized Hough Transform

# Preprocessing Edge Images



Manually Sketched

Edge  
Detection



Thresholding



Shrink  
& Expand



Boundary  
Detection



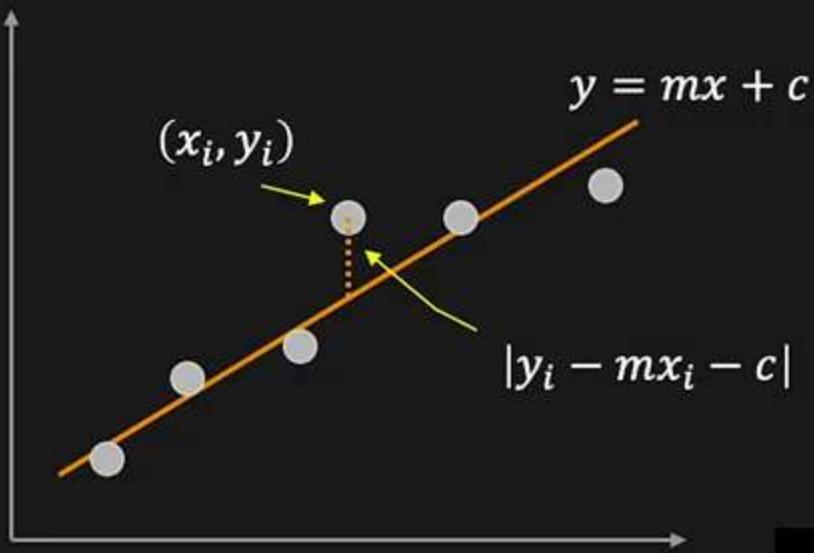
Thinning



# Fitting Lines to Edges

Given: Edge Points  $(x_i, y_i)$

Task: Find  $(m, c)$



Minimize: Average Squared Vertical Distance

$$E = \frac{1}{N} \sum_i (y_i - mx_i - c)^2$$

Least Squares Solution:

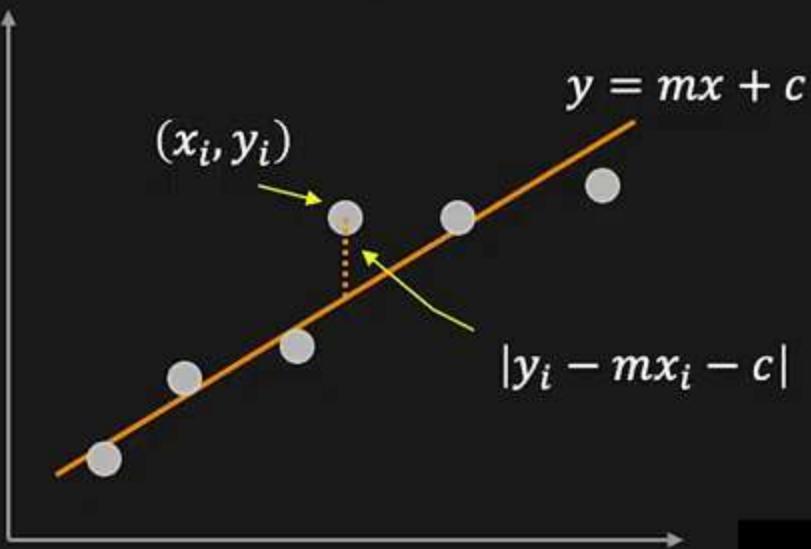
$$\frac{\partial E}{\partial m} = \frac{-2}{N} \sum_i x_i (y_i - mx_i - c) = 0$$

$$\frac{\partial E}{\partial c} = \frac{-2}{N} \sum_i (y_i - mx_i - c) = 0$$

# Fitting Lines to Edges

Given: Edge Points  $(x_i, y_i)$

Task: Find  $(m, c)$



Solution:

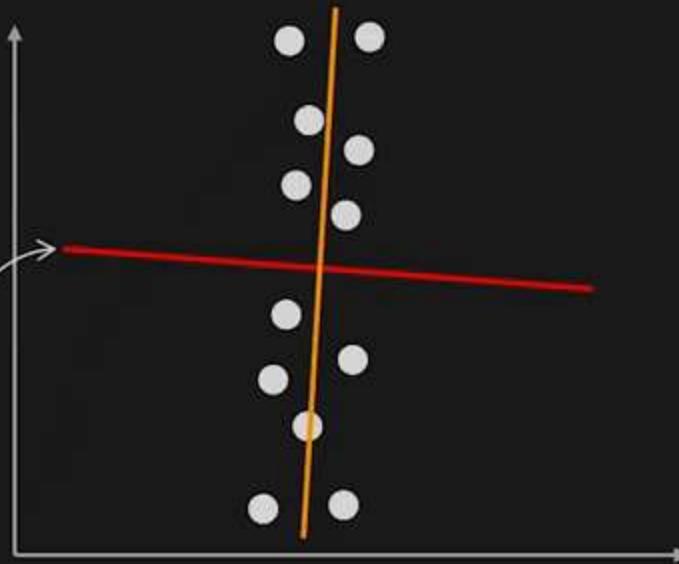
$$m = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} \quad c = \bar{y} - m\bar{x}$$

where:  $\bar{x} = \frac{1}{N} \sum_i x_i$      $\bar{y} = \frac{1}{N} \sum_i y_i$

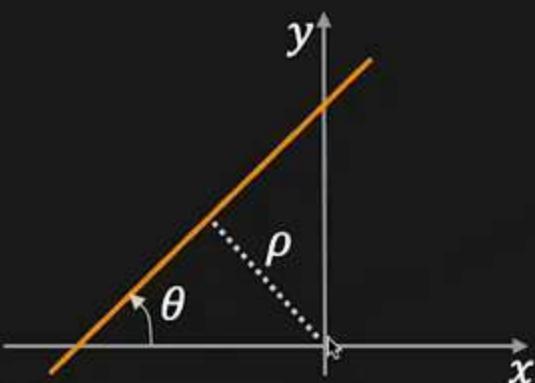
# Fitting Lines to Edges

**Problem:** When the points represent a vertical line.

Line that minimizes E!



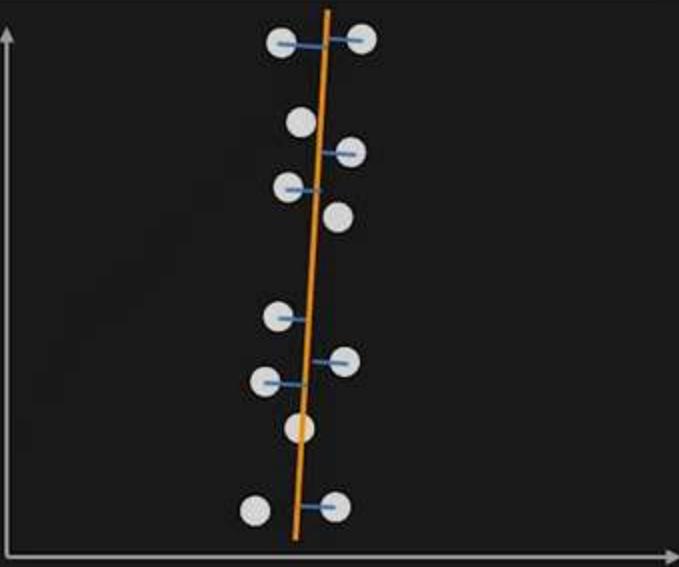
**Solution:** Use a different line equation



$$x \sin \theta - y \cos \theta + \rho = 0$$

# Fitting Lines to Edges

**Problem:** When the points represent a vertical line.



**Minimize:** Average Squared **Perpendicular** Distance

$$E = \frac{1}{N} \sum_i \frac{(x_i \sin \theta - y_i \cos \theta + \rho)^2}{\text{Perpendicular Distance}}$$

(See Binary Img. Processing Lecture)

# Fitting Curves to Edges

Given: Edge Points  $(x_i, y_i)$

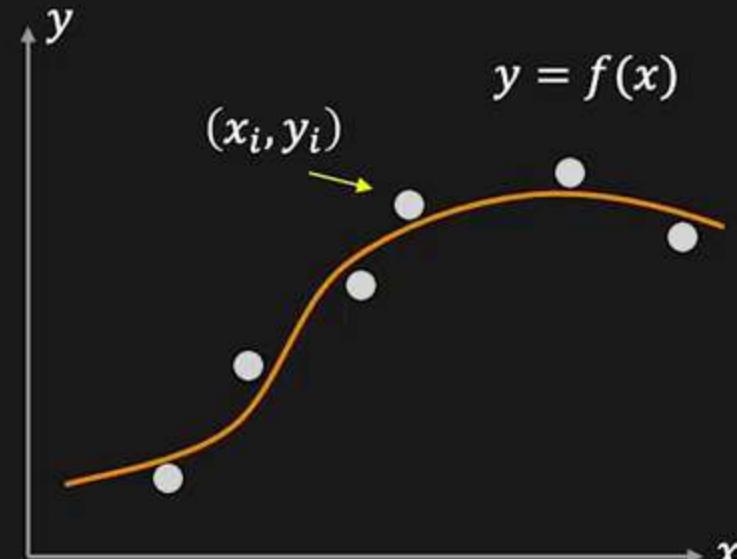
Task: Find polynomial

$$y = f(x) = ax^3 + bx^2 + cx + d$$

that best fits the points

Minimize:

$$E = \frac{1}{N} \sum_i (y_i - ax_i^3 - bx_i^2 - cx_i - d)^2$$



Solve the Linear System Using Least Squares Fit by:

$$\frac{\partial E}{\partial a} = 0 \quad \frac{\partial E}{\partial b} = 0 \quad \frac{\partial E}{\partial c} = 0 \quad \frac{\partial E}{\partial d} = 0$$

Closed-form solution cumbersome when unknowns are many

# Fitting Curves to Edges

Solving as a Linear System:

$$y_0 = ax_0^3 + bx_0^2 + cx_0 + d$$

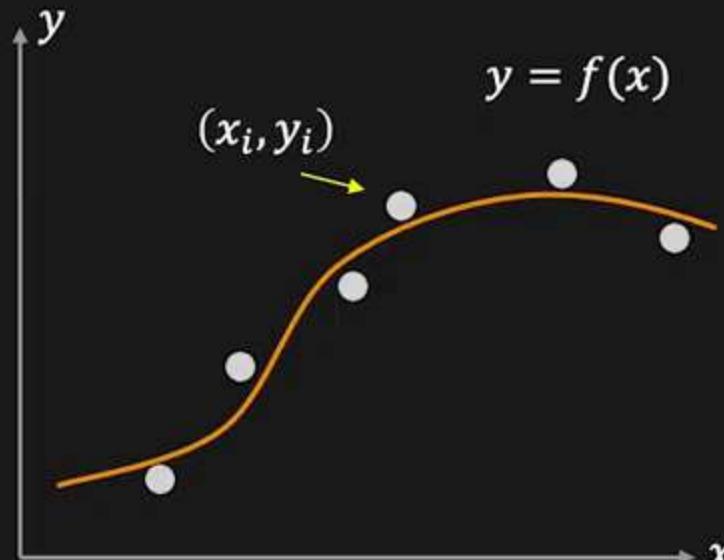
$$y_1 = ax_1^3 + bx_1^2 + cx_1 + d$$

⋮

$$y_i = ax_i^3 + bx_i^2 + cx_i + d$$

⋮

$$y_n = ax_n^3 + bx_n^2 + cx_n + d$$



Given many  $(x_i, y_i)$ 's, this is an over-determined linear system with four unknowns  $(a, b, c, d)$ .

# Solving a Linear System

An over-determined linear system with  $m$  unknowns  $\{a_j\}$  ( $j = 0, \dots, m$ ) and  $n$  observations  $\{(x_{ij}, y_i)\}$  ( $i = 0, \dots, n$ ) ( $n > m$ ) can be written in a matrix form.

$$\left[ \begin{array}{cccc} x_{00} & x_{01} & \dots & x_{0m} \\ x_{10} & x_{11} & \dots & x_{1m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n0} & x_{n1} & \dots & x_{nm} \end{array} \right] \left[ \begin{array}{c} a_0 \\ a_1 \\ \vdots \\ a_m \end{array} \right] = \left[ \begin{array}{c} y_0 \\ y_1 \\ \vdots \\ y_n \end{array} \right] \quad \left. \right\} \begin{array}{l} X\mathbf{a} = \mathbf{y} \\ X_{n \times m} \text{ is not a square matrix and hence not invertible.} \end{array}$$

$X_{n \times m}$        $\mathbf{a}_{m \times 1}$        $\mathbf{y}_{n \times 1}$   
Known                  Unknown      Known

Least Squares Solution:

$$X^T X \mathbf{a} = X^T \mathbf{y} \Rightarrow \mathbf{a} = (X^T X)^{-1} X^T \mathbf{y} \quad X^+ = (X^T X)^{-1} X^T$$

$\boxed{\mathbf{a} = X^+ \mathbf{y}}$       (Pseudo Inverse)

# What is an Active Contour?

**Given:** Approximate boundary (contour) around the object

**Task:** Evolve (move) the contour to fit exact object boundary

**Active Contour:**

Iteratively “deform” the initial contour so that:

- It is near pixels with high gradient (edges)
- It is smooth



Image

Also called **Snakes**

# Power of Deformable Contours

---

Boundaries could deform over time



Boundaries could deform with viewpoint

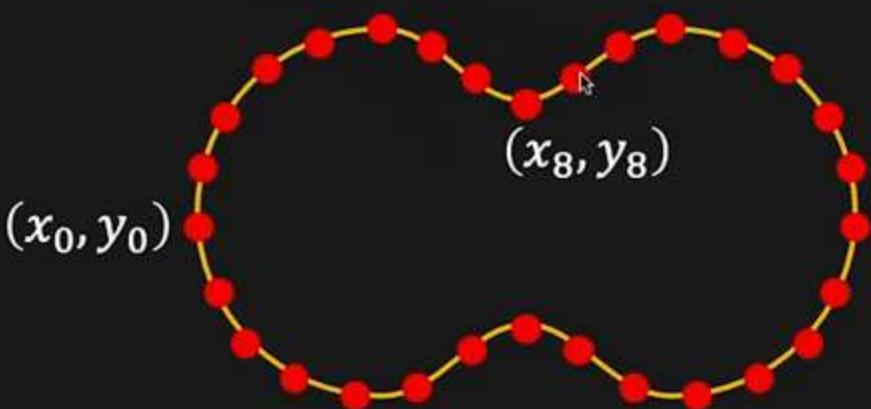


**Boundary Tracking:** Use the boundary from the current image as initial boundary for the next image.

# Representing a Contour

---

Contour  $\mathbf{v}$ : An ordered list of 2D vertices (control points) connected by straight lines of fixed length



$$\mathbf{v} = \{v_i = (x_i, y_i) \mid i = 0, 1, 2, \dots, n - 1\}$$

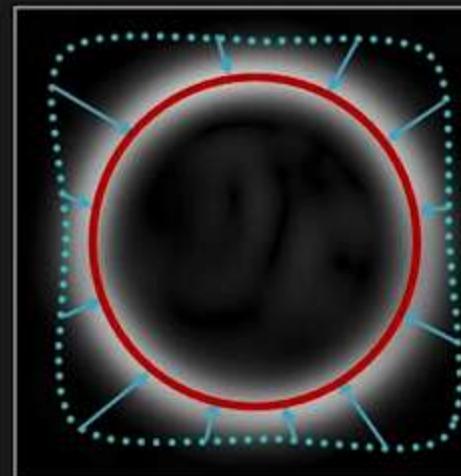
# Attracting Contours to Edges



Image with  
Initial Contour



Gradient Magnitude  
Squared  
 $\|\nabla I\|^2$



Blurred Gradient  
Magnitude Squared  
 $\|\nabla n_\sigma * I\|^2$

Maximize Sum of Gradient Magnitude Square

≡ Minimize -ve (Sum of Gradient Magnitude Square)

≡ Minimize  $E_{image} = - \sum_{i=0}^{n-1} \|\nabla n_\sigma * I(v_i)\|^2$

# Contour Deformation: Greedy Algorithm

---

1. For each contour point  $v_i$  ( $i = 0, \dots, n - 1$ ), move  $v_i$  to a position within a window  $W$  where the energy function  $E_{image}$  for the contour is minimum.

2. If the sum of motions of all the contour points is less than a threshold, stop. Else go to Step 1.



Greedy solution might be suboptimal and slow.

# Sensitivity to Noise and Initialization

---



Contour fitted to  
gradient magnitude



# Sensitivity to Noise and Initialization

---



Contour fitted to  
gradient magnitude

1

# Sensitivity to Noise and Initialization

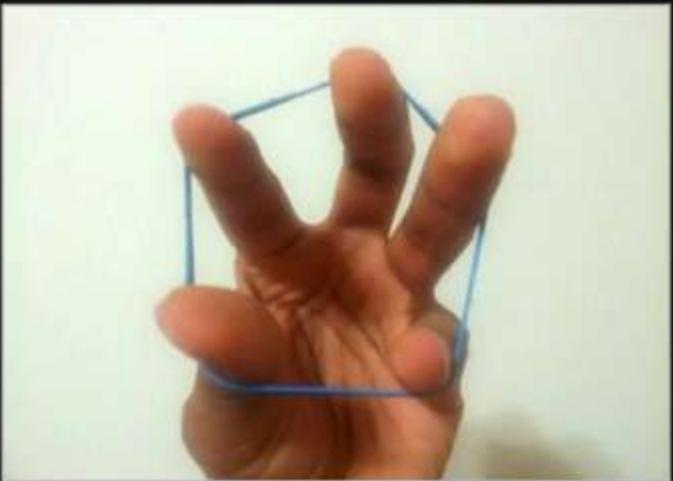
---



Contour fitted to  
gradient magnitude

Solution: Add constraints that make the contour  
**contract and remain smooth**

# Making Contours Elastic and Smooth



Elastic and contracts  
like a rubber band



Smooth  
like a metal strip

Minimize **Internal Bending Energy** of the Contour:

$$E_{contour} = \alpha E_{elastic} + \beta E_{smooth}$$

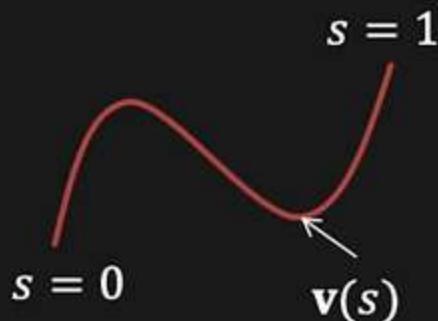
$(\alpha, \beta)$ : Control the influence of elasticity and smoothness

# Elasticity and Smoothness

For point  $0 \leq s \leq 1$  on continuous contour  $\mathbf{v}(s) = (x(s), y(s))$ :

$$E_{elastic} = \left\| \frac{d\mathbf{v}}{ds} \right\|^2$$

$$E_{smooth} = \left\| \frac{d^2\mathbf{v}}{ds^2} \right\|^2$$



Discrete approximations at control point  $\mathbf{v}_i$ :

$$E_{elastic}(\mathbf{v}_i) = \left\| \frac{d\mathbf{v}}{ds} \right\|^2 \approx \|\mathbf{v}_{i+1} - \mathbf{v}_i\|^2 = (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2$$

$$\begin{aligned} E_{smooth}(\mathbf{v}_i) &= \left\| \frac{d^2\mathbf{v}}{ds^2} \right\|^2 \approx \|(\mathbf{v}_{i+1} - \mathbf{v}_i) - (\mathbf{v}_i - \mathbf{v}_{i-1})\|^2 \\ &= (x_{i+1} - 2x_i + x_{i-1})^2 + (y_{i+1} - 2y_i + y_{i-1})^2 \end{aligned}$$

# Elasticity and Smoothness

---

Internal bending energy along the entire contour:

$$E_{contour} = \alpha E_{elastic} + \beta E_{smooth}$$

where:

$$E_{elastic} = \sum_{i=0}^{n-1} [(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2]$$

$$E_{smooth} = \sum_{i=0}^{n-1} [(x_{i+1} - 2x_i + x_{i-1})^2 + (y_{i+1} - 2y_i + y_{i-1})^2]$$

# Combining the Forces

---

**Image Energy,  $E_{image}$ :** Measure of how well the contour latches on to edges

**Internal Energy,  $E_{contour}$ :** Measure of elasticity and smoothness

Total Energy of Active Contour:

$$E_{total} = E_{image} + E_{contour}$$

Minimize the Total Energy

# Contour Deformation: Greedy Algorithm

1. Uniformly sample the contour to get  $n$  contour points.
2. For each contour point  $v_i$  ( $i = 0, \dots, n - 1$ ), move  $v_i$  to a position within a window  $W$  where the energy function  $E_{total}$  for the entire contour is minimum.

$$E_{total} = E_{image} + E_{contour}$$

3. If the sum of motions of all the contour points is less than a threshold, stop. Else go to Step 1.



# Result: Effect of Contour Constraint

---



Without contour constraint

$$E_{total} = E_{image}$$



With contour constraint

$$E_{total} = E_{image} + E_{contour}$$

# Result: Effect of Contour Constraint

---



Without contour constraint

$$E_{total} = E_{image}$$



With contour constraint

$$E_{total} = E_{image} + E_{contour}$$

# Result: Effect of Contour Constraint

---



Without contour constraint

$$E_{total} = E_{image}$$



With contour constraint

$$E_{total} = E_{image} + E_{contour}$$

# Result: Effect of Contour Constraint

---



Without contour constraint

$$E_{total} = E_{image}$$



With contour constraint

$$E_{total} = E_{image} + E_{contour}$$

# Result: Boundary Around Two Objects

---



Large  $\alpha$   
(More like a rubber band)



Small  $\alpha$   
(Less like a rubber band)

# Active Contours: Comments

---

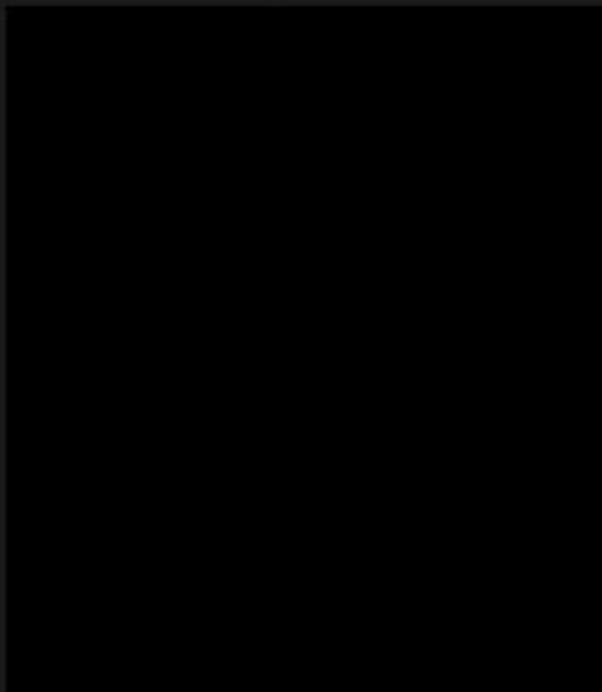
- Additional energy constraints can be added
  - Penalize deviation from prior model of shape
- Requires good initialization
  - Edges cannot attract contours that are far away
- Elasticity makes contour contract
  - Replace contracting force with ballooning force to expand

# Medical Image Segmentation

---



I.1



# Interactive Image Segmentation

---



Magnetic Lasso Tool in Photoshop

# Difficulties for the Fitting Approach



- Extraneous Data: Which points to fit to?
- Incomplete Data: Only part of the model is visible.
- Noise

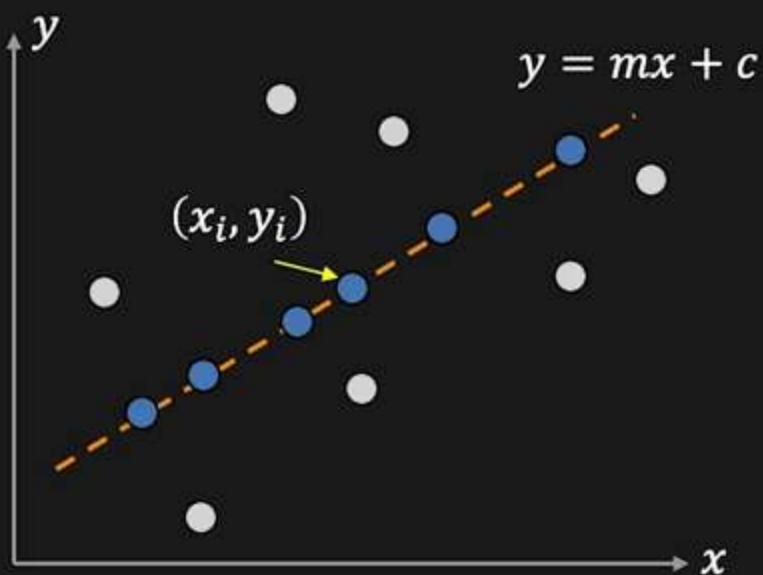
Solution: Hough Transform

# Hough Transform: Line Detection

Given: Edge Points  $(x_i, y_i)$

Task: Detect line

$$y = mx + c$$



Consider point  $(x_i, y_i)$

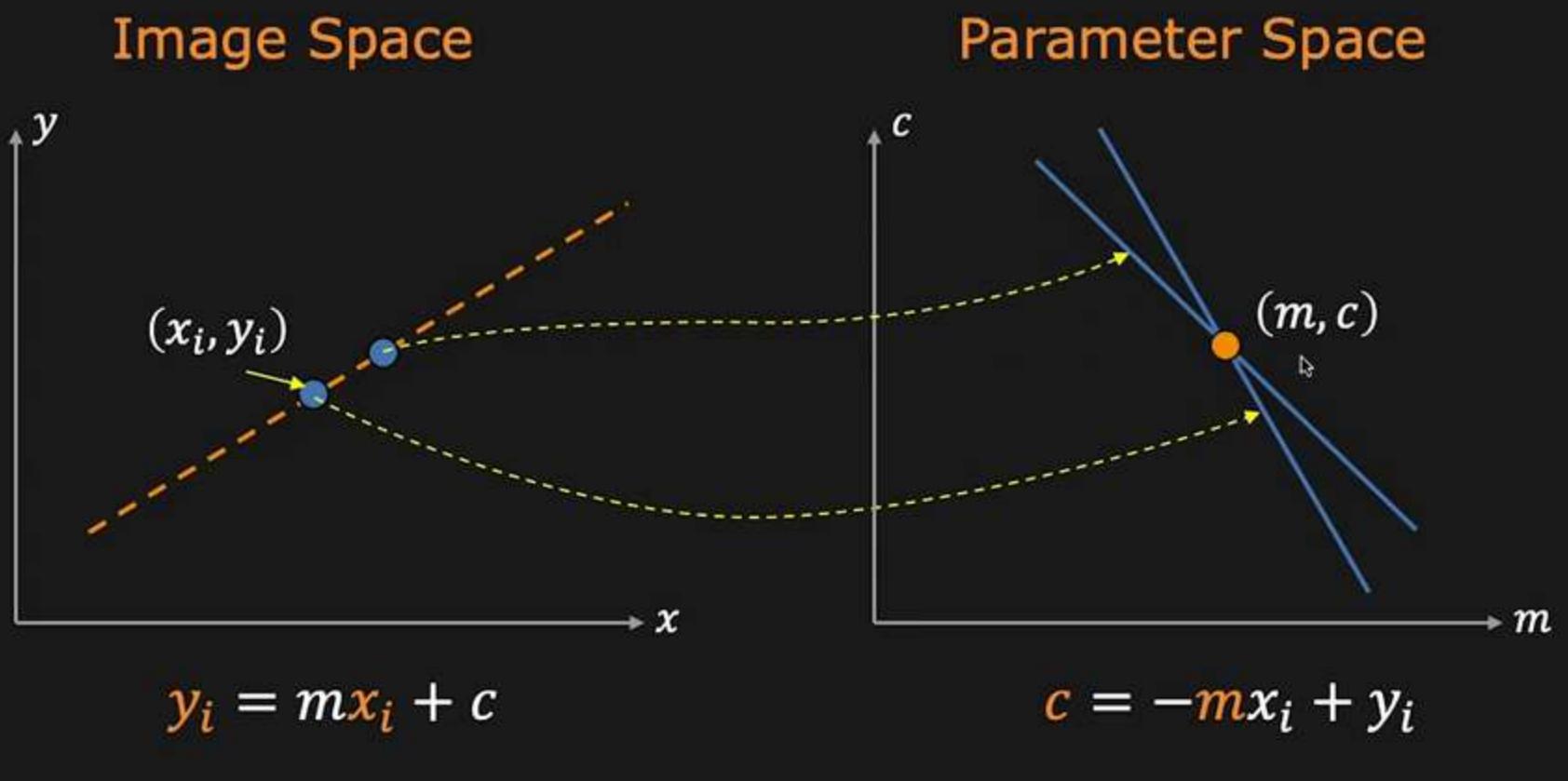
$$y_i = mx_i + c$$



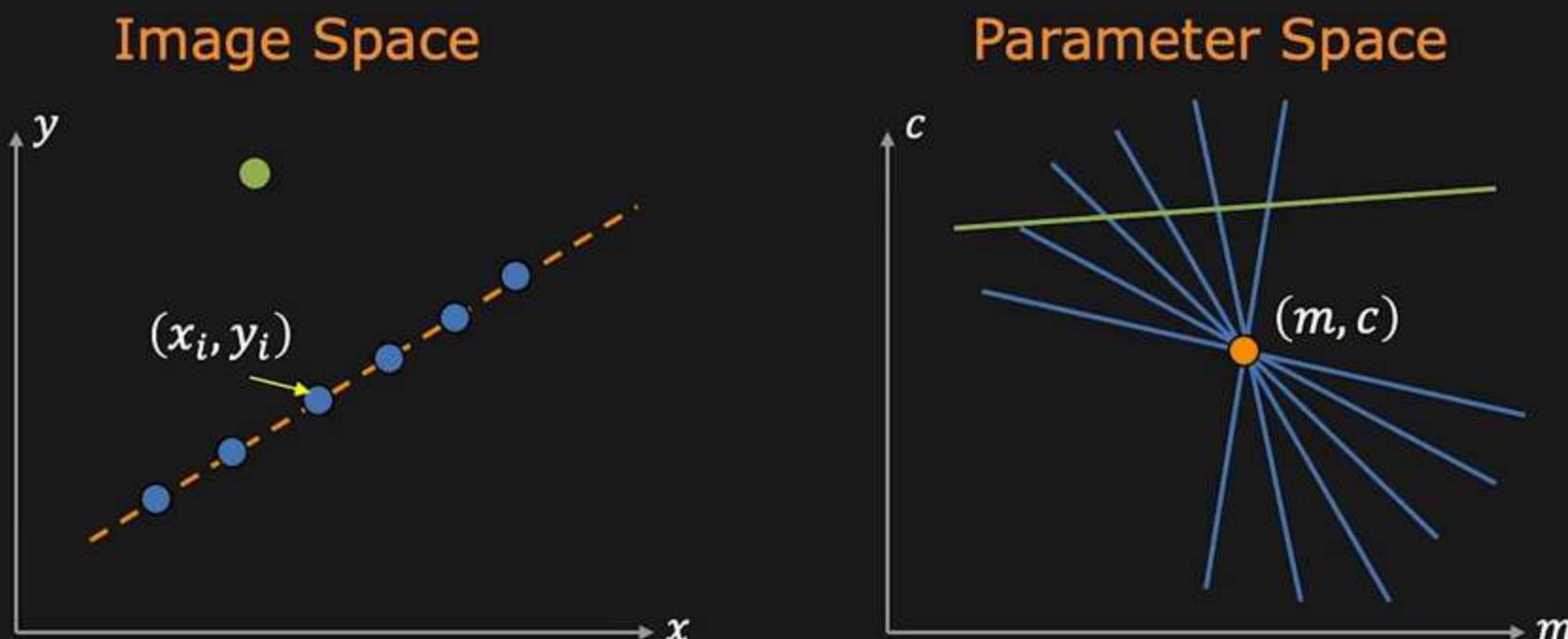
$$c = -mx_i + y_i$$

↓

# Hough Transform: Concept



# Hough Transform: Concept



Point  $\longleftrightarrow$  Line

Line  $\longleftrightarrow$  Point

# Line Detection Algorithm

Step 1. Quantize parameter space  $(m, c)$

Step 2. Create **accumulator array**  $A(m, c)$

Step 3. Set  $A(m, c) = 0$  for all  $(m, c)$

Step 4. For each edge point  $(x_i, y_i)$ ,

$$A(m, c) = A(m, c) + 1$$

if  $(m, c)$  lies on the line:  $c = -mx_i + y_i$



$A(m, c)$

$c$	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	0

# Line Detection Algorithm

Step 1. Quantize parameter space  $(m, c)$

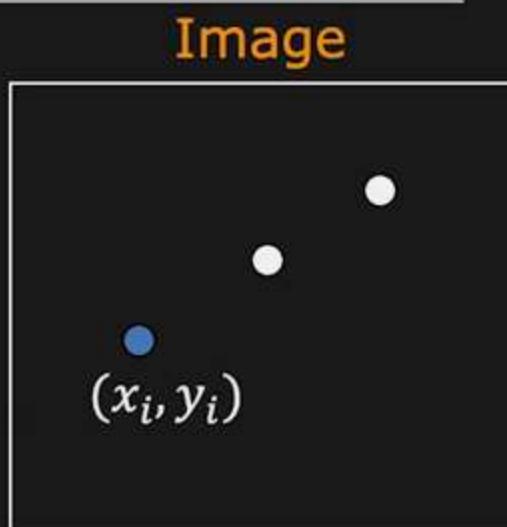
Step 2. Create **accumulator array**  $A(m, c)$

Step 3. Set  $A(m, c) = 0$  for all  $(m, c)$

Step 4. For each edge point  $(x_i, y_i)$ ,

$$A(m, c) = A(m, c) + 1$$

if  $(m, c)$  lies on the line:  $c = -mx_i + y_i$



**$A(m, c)$**

$c$	1	0	0	0	0
1	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	0	1

# Line Detection Algorithm

Step 1. Quantize parameter space  $(m, c)$

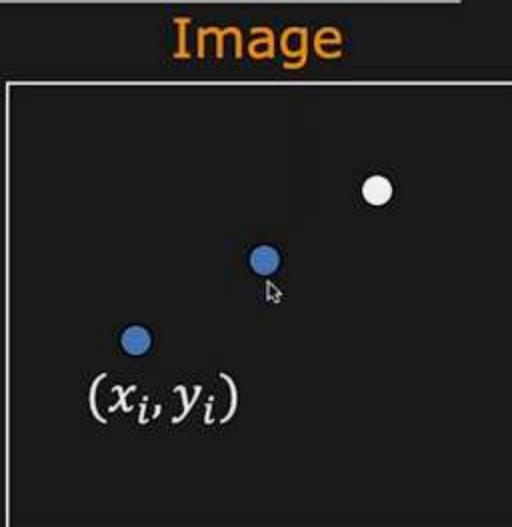
Step 2. Create **accumulator array**  $A(m, c)$

Step 3. Set  $A(m, c) = 0$  for all  $(m, c)$

Step 4. For each edge point  $(x_i, y_i)$ ,

$$A(m, c) = A(m, c) + 1$$

if  $(m, c)$  lies on the line:  $c = -mx_i + y_i$



**$A(m, c)$**

$c$	1	0	0	0	1
1	0	0	1	0	0
0	1	0	0	1	0
0	0	2	0	0	0
0	1	0	1	0	0
1	0	0	0	0	1

# Line Detection Algorithm

Step 1. Quantize parameter space  $(m, c)$

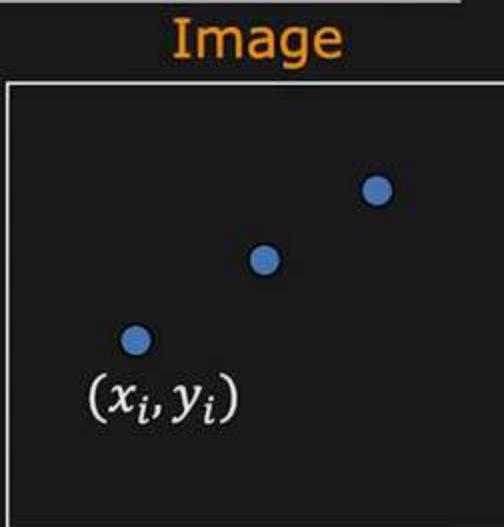
Step 2. Create **accumulator array**  $A(m, c)$

Step 3. Set  $A(m, c) = 0$  for all  $(m, c)$

Step 4. For each edge point  $(x_i, y_i)$ ,

$$A(m, c) = A(m, c) + 1$$

if  $(m, c)$  lies on the line:  $c = -mx_i + y_i$



**$A(m, c)$**

$c$	1	0	0	0	1
1	0	1	0	1	0
0	1	0	1	0	1
1	1	3	1	1	1
0	1	0	1	0	1
1	0	0	0	0	1

# Line Detection Algorithm

Step 1. Quantize parameter space  $(m, c)$

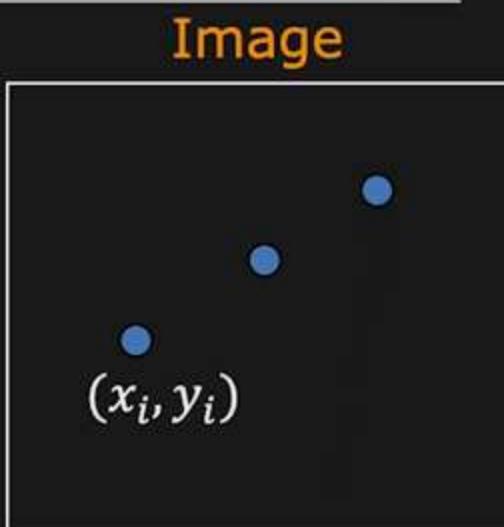
Step 2. Create **accumulator array**  $A(m, c)$

Step 3. Set  $A(m, c) = 0$  for all  $(m, c)$

Step 4. For each edge point  $(x_i, y_i)$ ,

$$A(m, c) = A(m, c) + 1$$

if  $(m, c)$  lies on the line:  $c = -mx_i + y_i$



**$A(m, c)$**

$c$	1	0	0	0	1
1	1	0	1	0	0
0	0	1	0	1	1
1	1	3	1	1	1
0	0	1	0	1	0
1	1	0	0	0	1

# Line Detection Algorithm

Step 1. Quantize parameter space  $(m, c)$

Step 2. Create **accumulator array**  $A(m, c)$

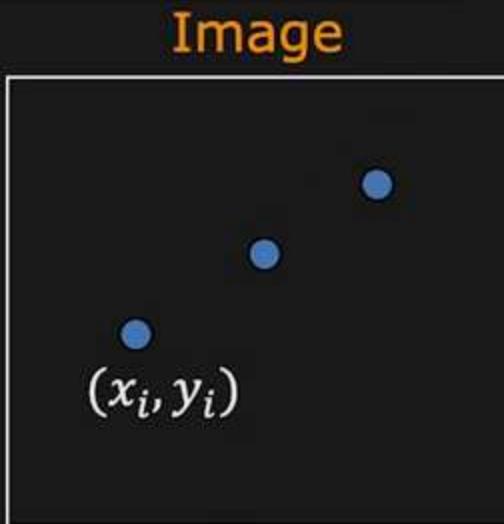
Step 3. Set  $A(m, c) = 0$  for all  $(m, c)$

Step 4. For each edge point  $(x_i, y_i)$ ,

$$A(m, c) = A(m, c) + 1$$

if  $(m, c)$  lies on the line:  $c = -mx_i + y_i$

Step 5. Find local maxima in  $A(m, c)$



**$A(m, c)$**

$c$	1	0	0	0	1
1	0	1	0	1	0
0	1	0	1	0	
1	1	3	1	1	1
0	1	0	1	1	
1	0	0	0	0	

# Multiple Line Detection

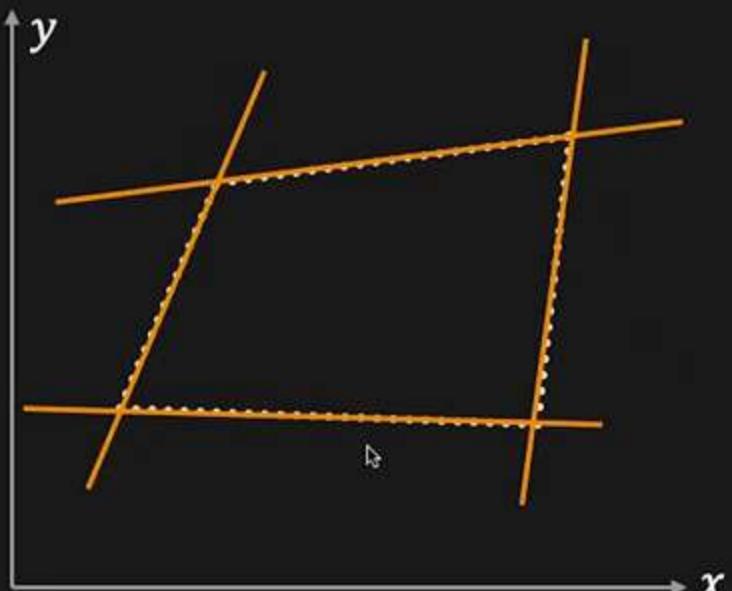
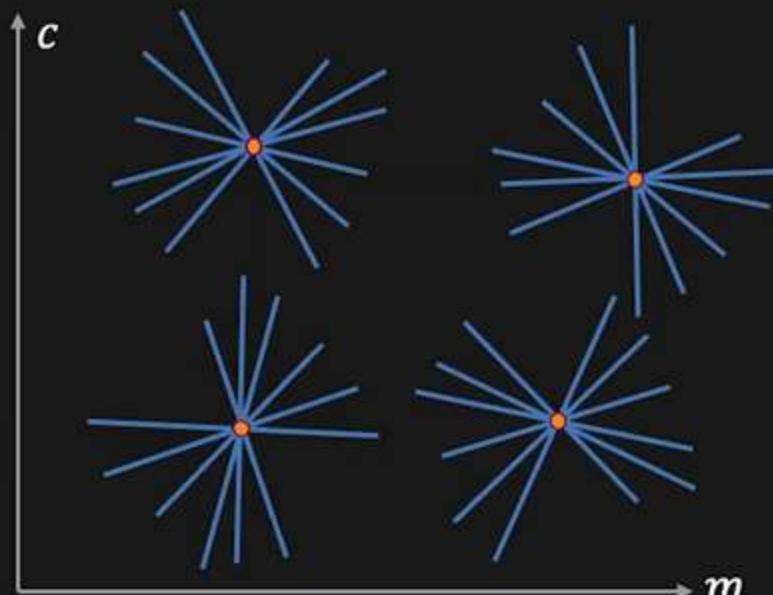


Image Space



Parameter Space

# Better Parameterization

---

**Issue:** Slope of the line  $-\infty \leq m \leq \infty$

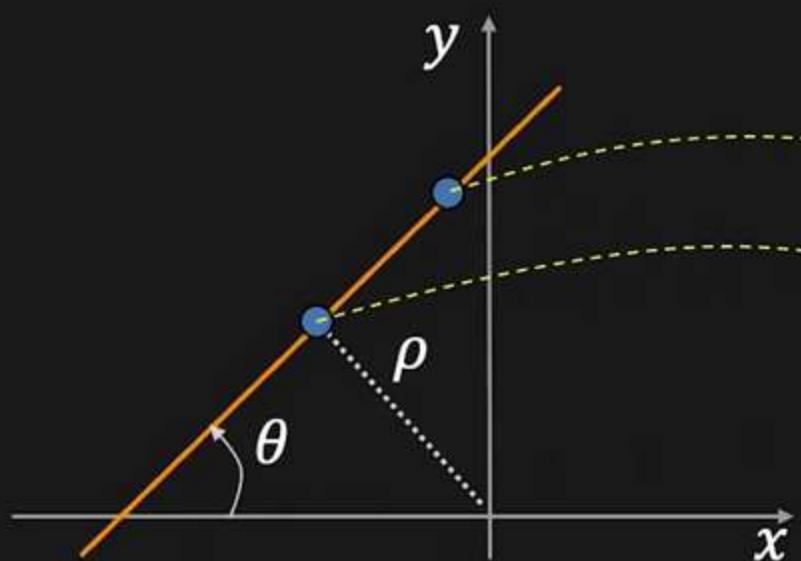
- Large Accumulator
- More Memory and Computation

**Solution:** Use  $x \sin \theta - y \cos \theta + \rho = 0$

- Orientation  $\theta$  is finite:  $0 \leq \theta < \pi$
- Distance  $\rho$  is finite

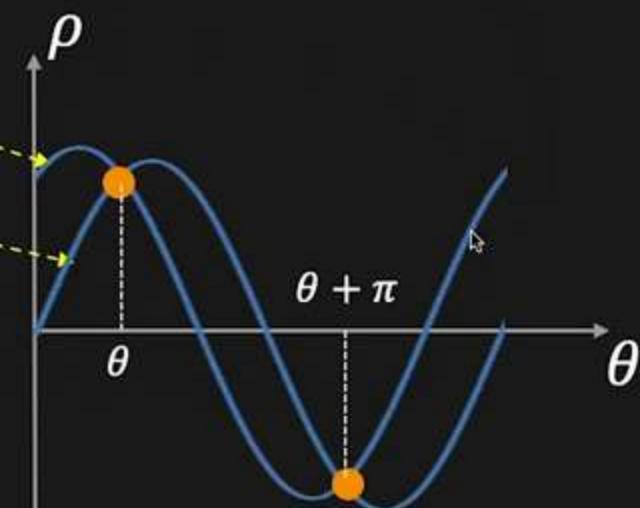
# Better Parameterization

Image Space



$$x \sin \theta - y \cos \theta + \rho = 0$$

Parameter Space



$$x \sin \theta - y \cos \theta + \rho = 0$$

# Hough Transform Mechanics

---

- How big should the accumulator cells be?
  - Too big, and different lines may be merged
  - Too small, and noise causes lines to be missed
- How many lines?
  - Count the peaks in the accumulator array
- Handling inaccurate edge locations:
  - Increment patch in accumulator rather than single point

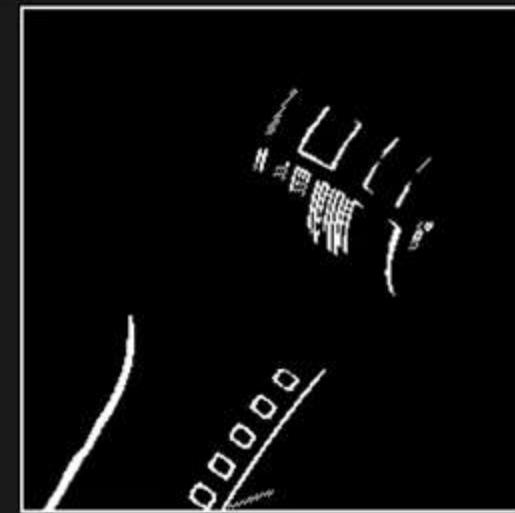
# Line Detection Results



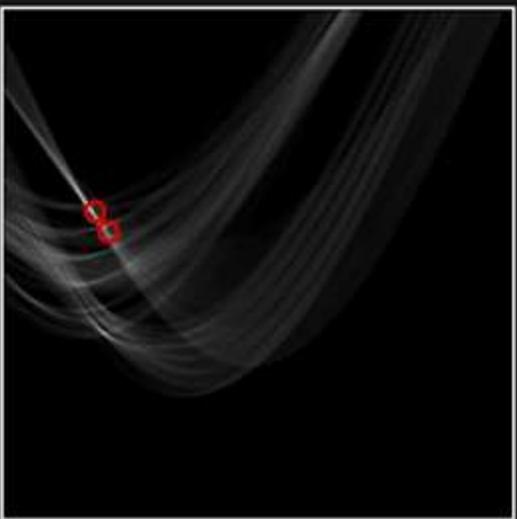
Original Image



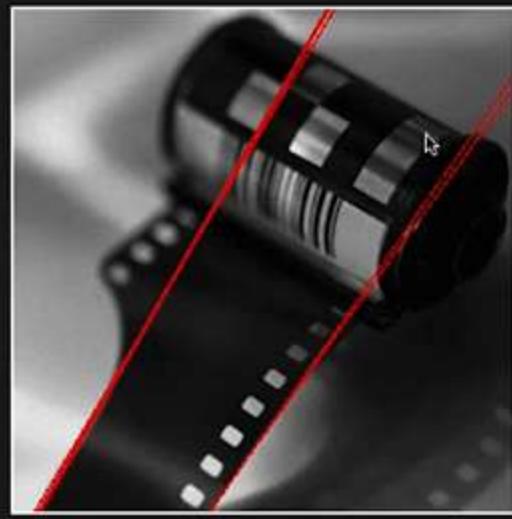
Gradient



Edge (Threshold)



Hough Transform  $A(\rho, \theta)$

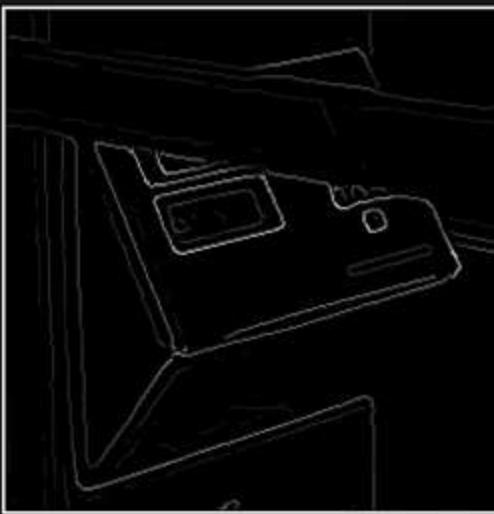


Detected Lines

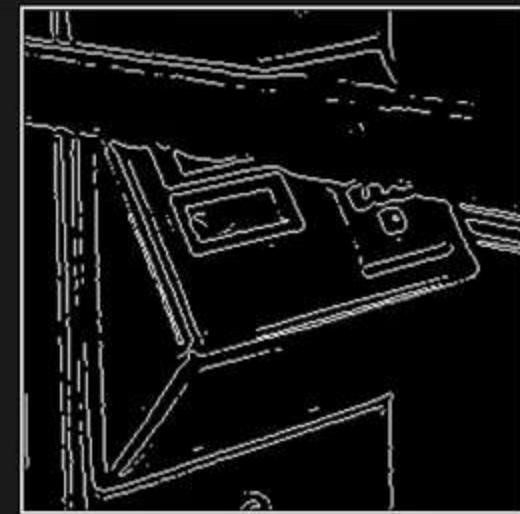
# Line Detection Results



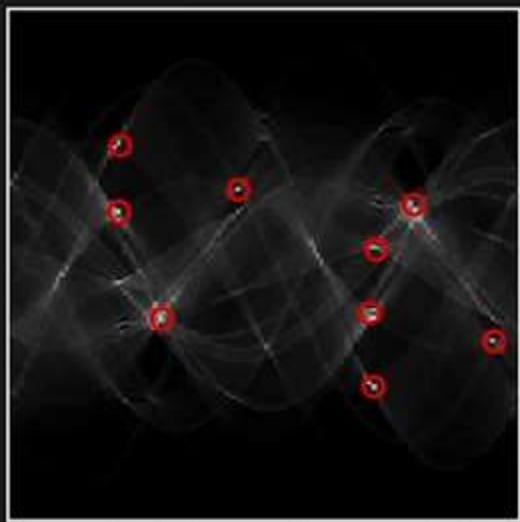
Original Image



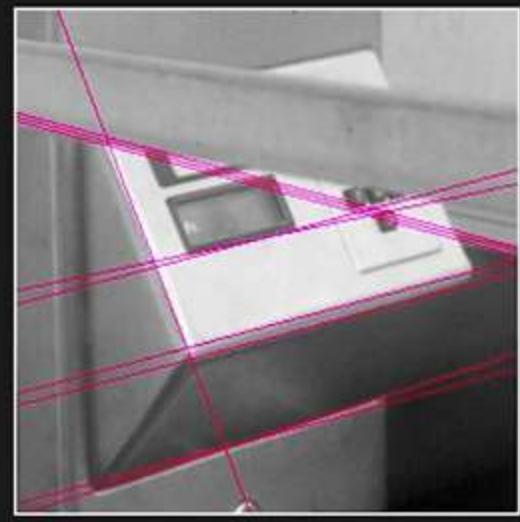
Gradient



Edge (Threshold)



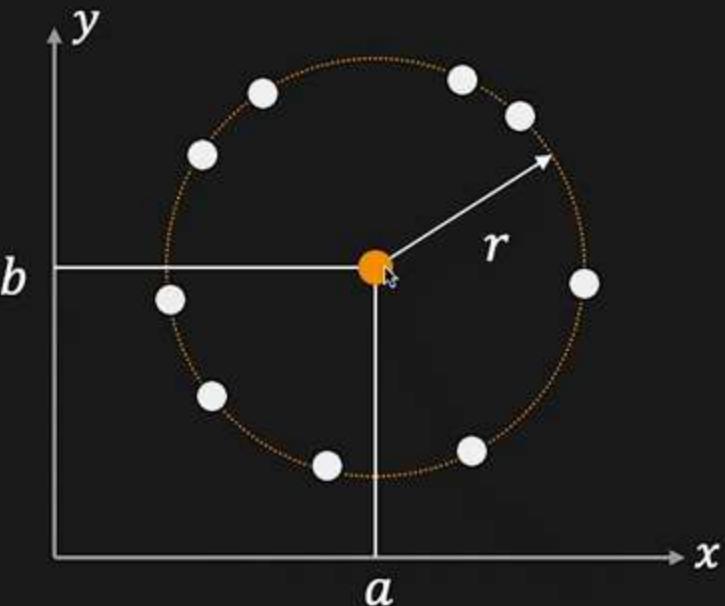
Hough Transform  $A(\rho, \theta)$



Detected Lines

# Hough Transform: Circle Detection

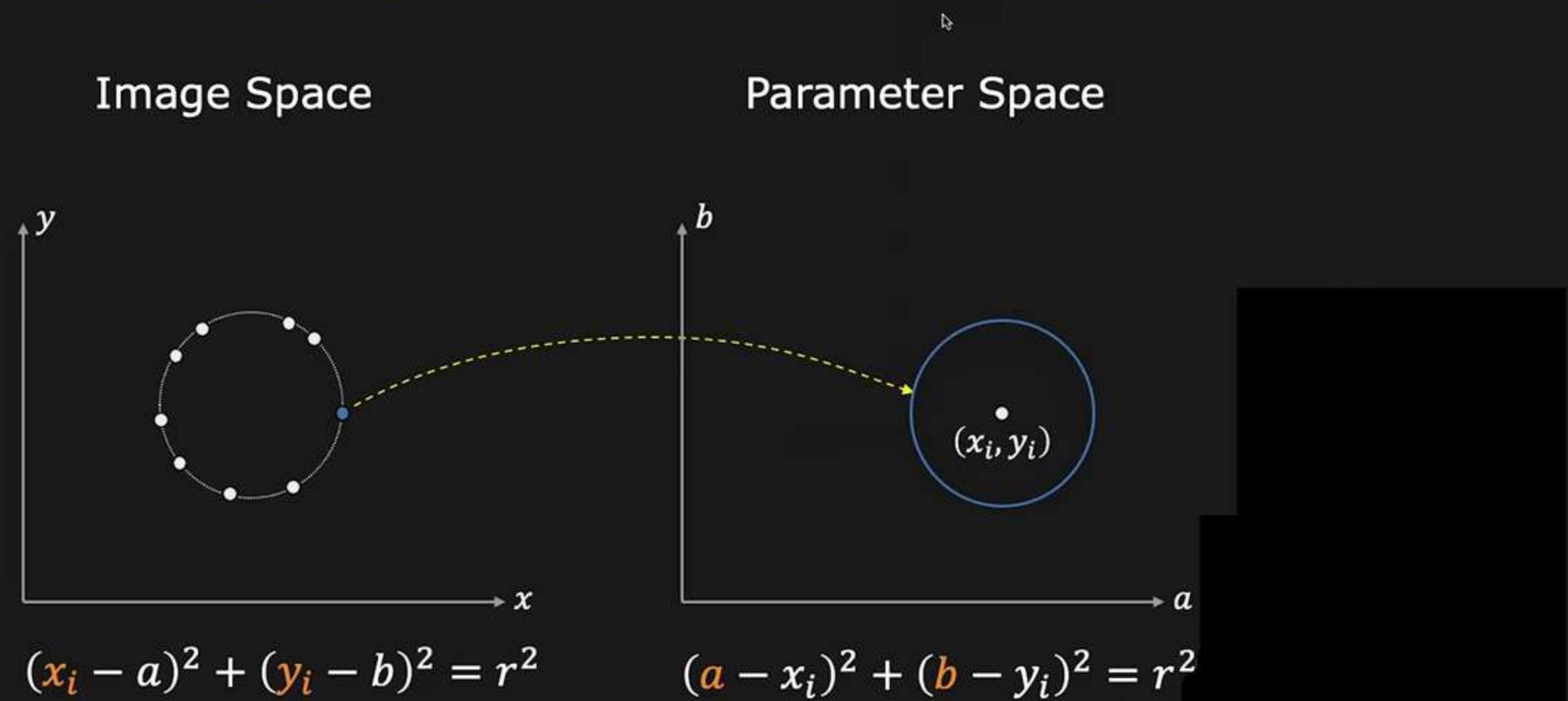
---



$$\text{Equation of Circle: } (x_i - a)^2 + (y_i - b)^2 = r^2$$

# Hough Transform: Circle Detection

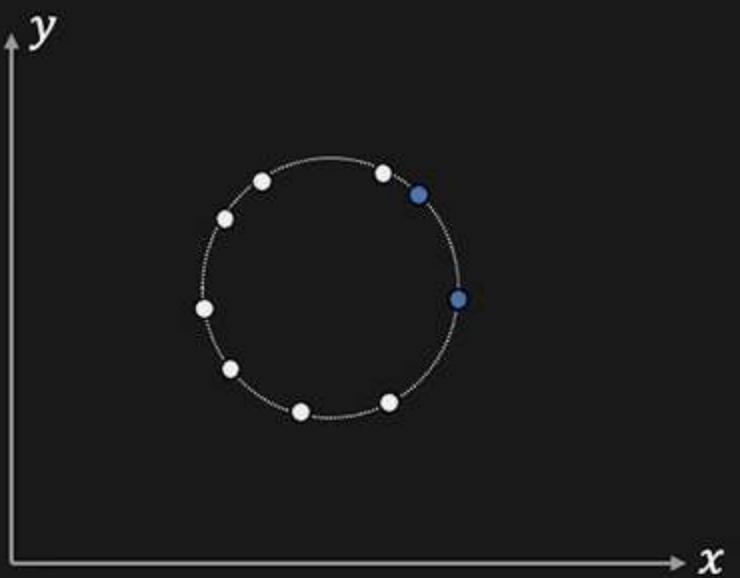
If radius  $r$  is known: Accumulator Array:  $A(a, b)$



# Hough Transform: Circle Detection

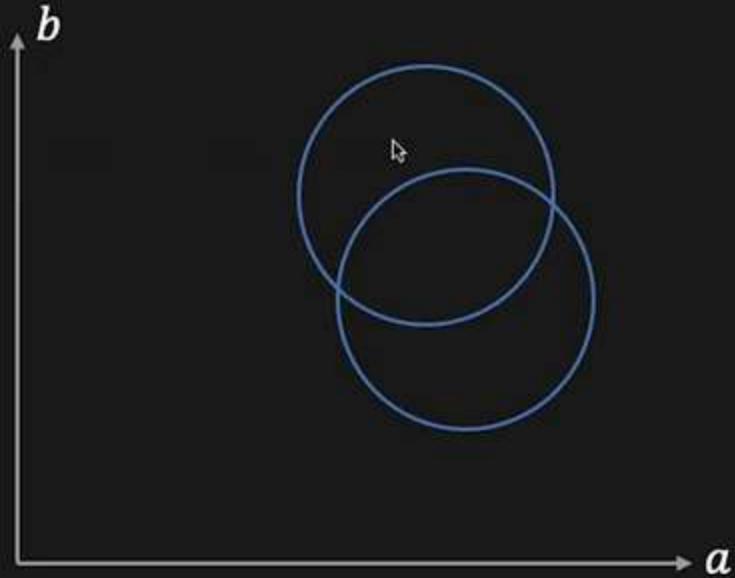
If radius  $r$  is known: Accumulator Array:  $A(a, b)$

Image Space

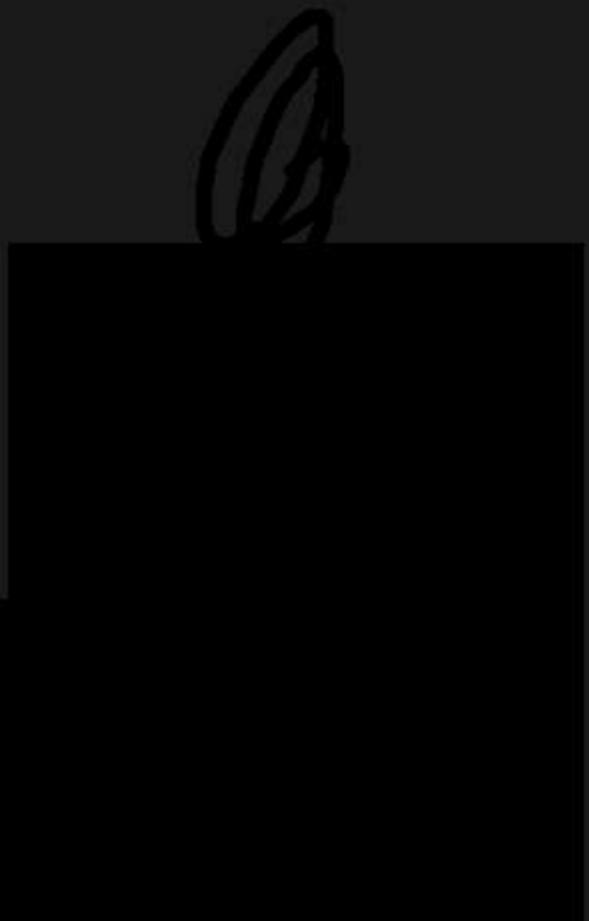


$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

Parameter Space



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



# Hough Transform: Circle Detection

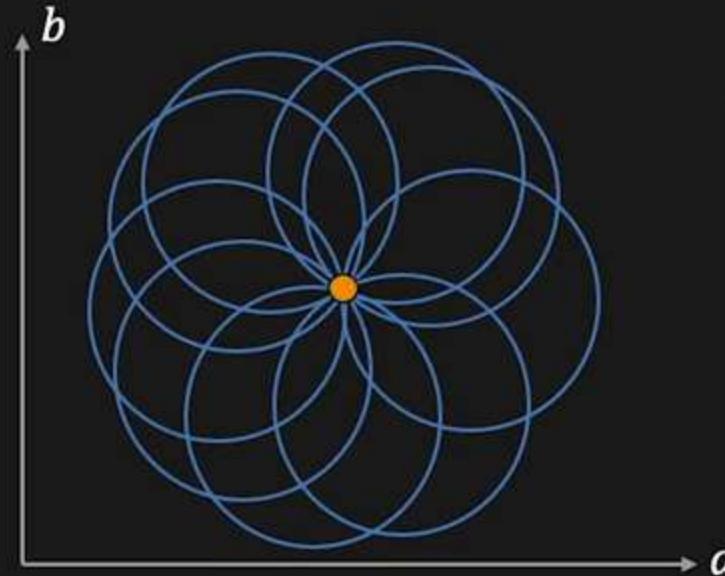
If radius  $r$  is known: Accumulator Array:  $A(a, b)$

Image Space



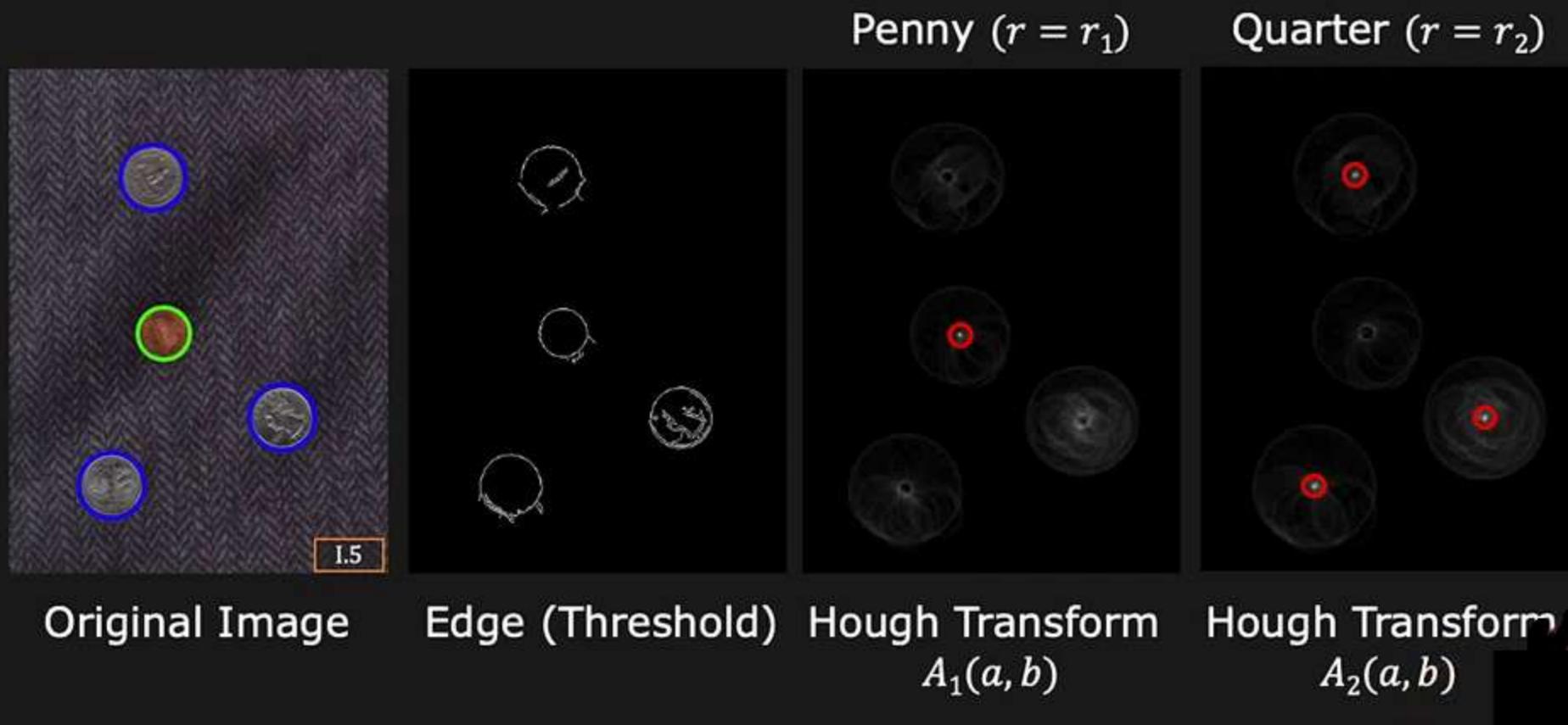
$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

Parameter Space



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

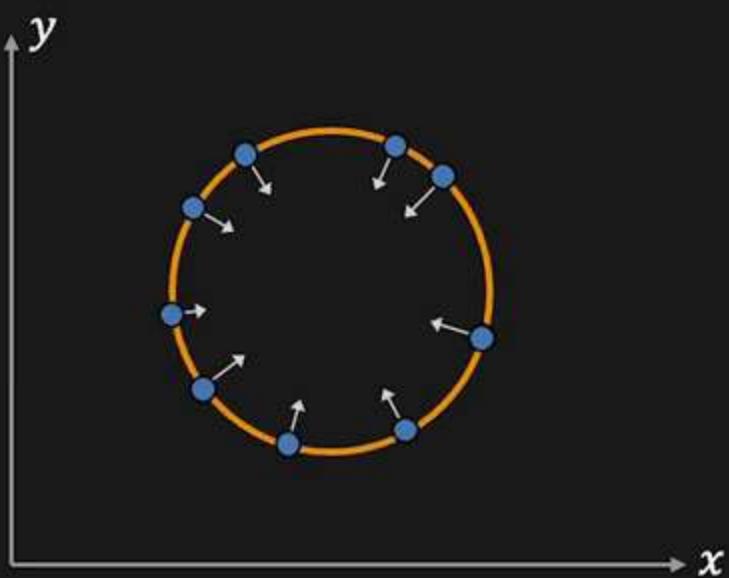
# Circle Detection Results



# Using Gradient Information

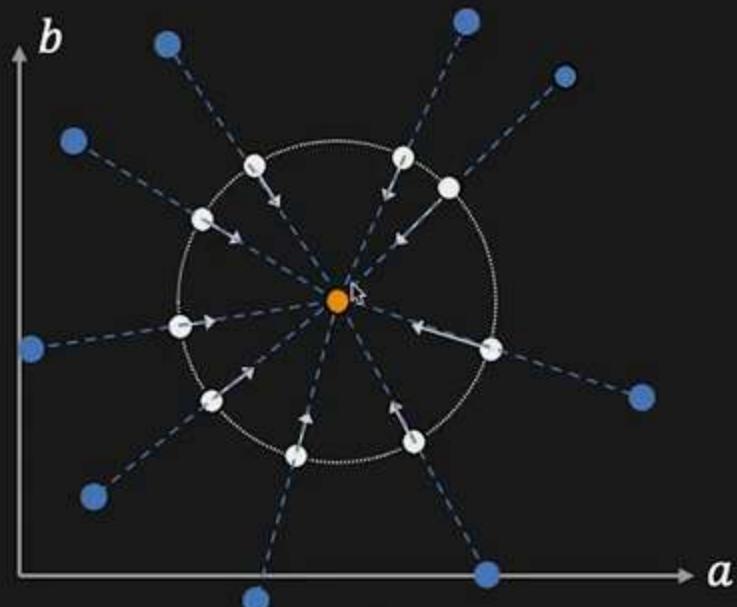
Given: Edge Location  $(x_i, y_i)$ , Edge Direction  $\varphi_i$  and Radius  $r$

Image Space



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

Parameter Space

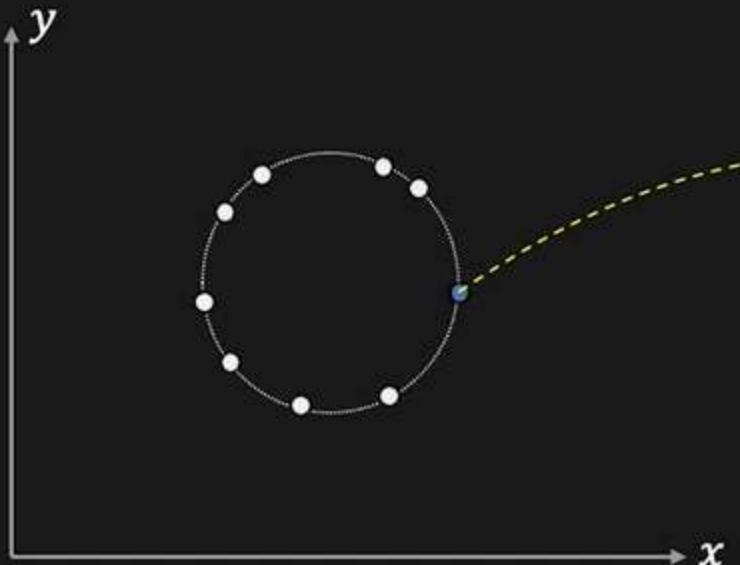


$$\begin{aligned} a &= x_i \pm r \cos \varphi_i \\ b &= y_i \pm r \sin \varphi_i \end{aligned}$$

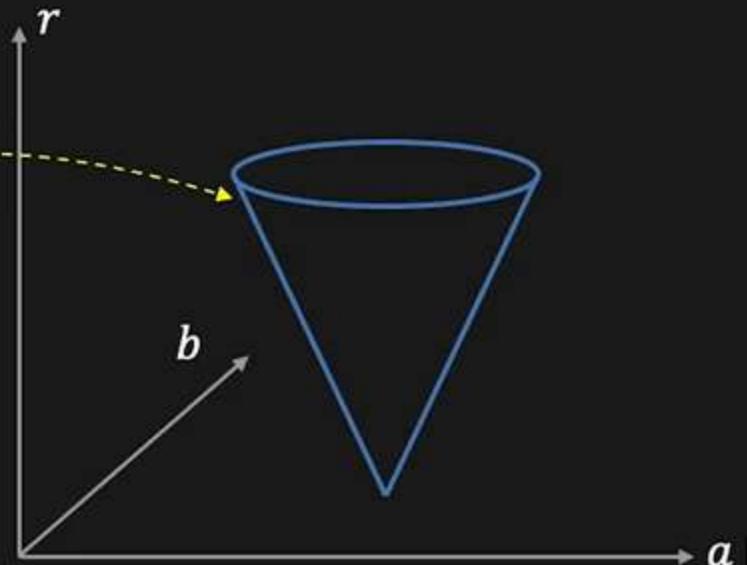
# Hough Transform: Circle Detection

If radius  $r$  is NOT known: Accumulator Array:  $A(a, b, r)$

Image Space



Parameter Space

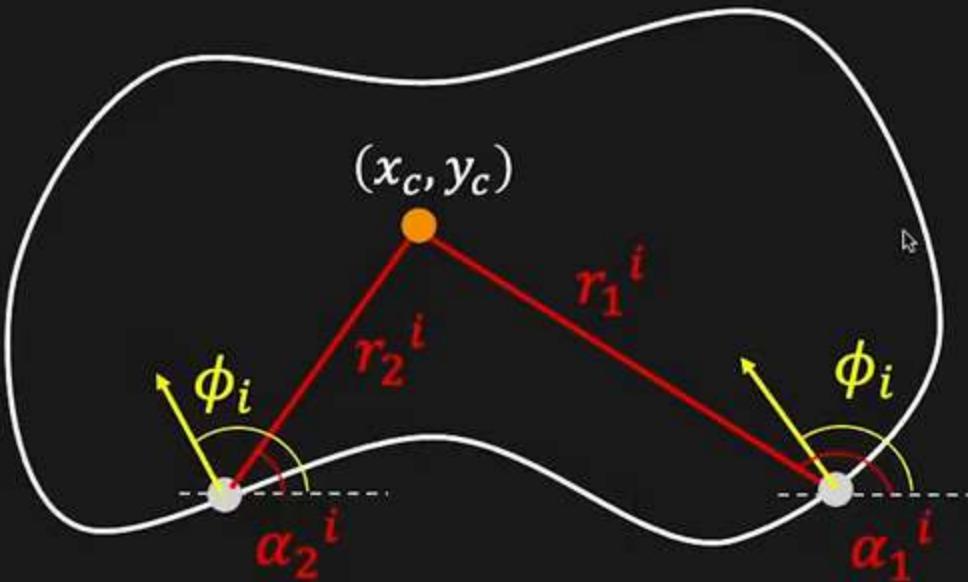


$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

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

# Generalized Hough Transform

Find shapes that cannot be described by equations

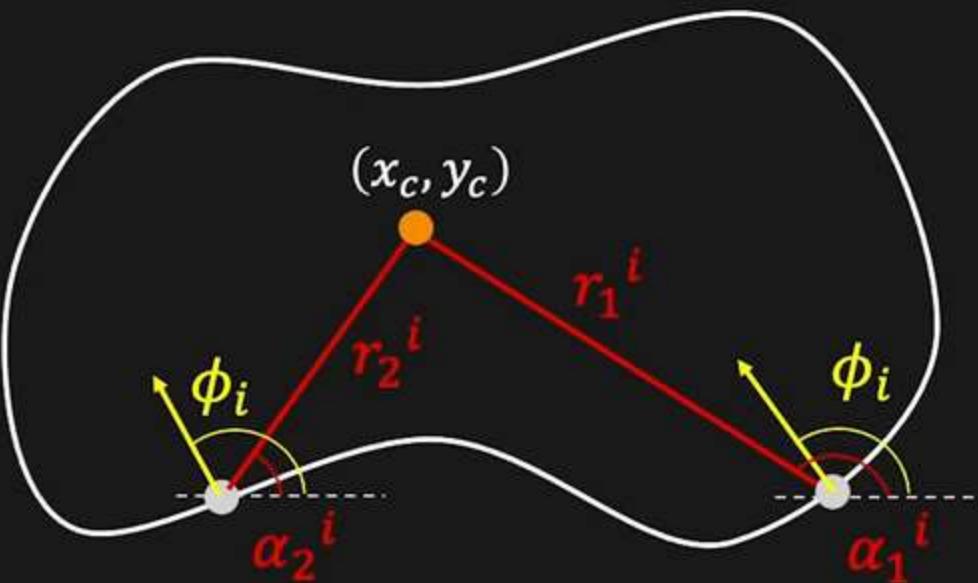


Reference point:  $(x_c, y_c)$

Edge direction:  $\phi_i$        $0 \leq \phi_i < 2\pi$

Edge location:  $\vec{r}_k^i = (r_k^i, \alpha_k^i)$

# Hough Model



$\phi$ -Table:

Edge Direction	$\vec{r} = (r, \alpha)$
$\phi_1$	$\vec{r}_1^1, \vec{r}_2^1, \vec{r}_3^1$
$\phi_2$	$\vec{r}_1^2, \vec{r}_2^2$
$\vdots$	$\vdots$
$\phi_n$	$\vec{r}_1^n, \vec{r}_2^n, \vec{r}_3^n, \vec{r}_4^n$

# Generalized Hough Algorithm

- Create **accumulator array**  $A(x_c, y_c)$
- Set  $A(x_c, y_c) = 0$  for all  $(x_c, y_c)$
- For each edge point  $(x_i, y_i, \phi_i)$ ,

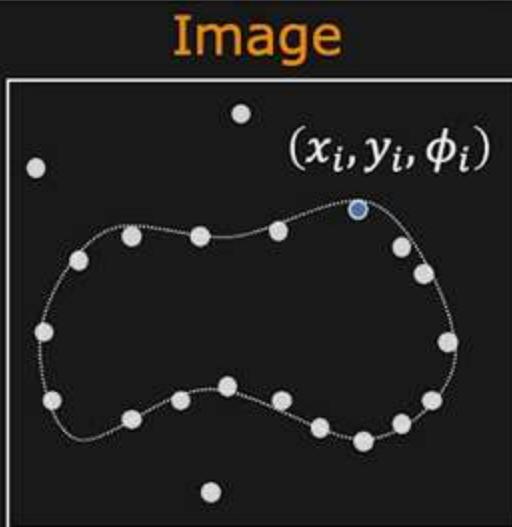
For each entry  $\phi_i \rightarrow \vec{r}_k^i$  in  $\phi$  – table,

$$x_c = x_i \pm r_k^i \cos(\alpha_k^i)$$

$$y_c = y_i \pm r_k^i \sin(\alpha_k^i)$$

$$A(x_c, y_c) = A(x_c, y_c) + 1$$

- Find local maxima in  $A(x_c, y_c)$



$x_c$	$A(x_c, y_c)$				
0	0	0	0	0	0
0	2	0	1	0	0
0	0	4	1	0	0
0	2	0	0	0	0
0	0	0	1	0	0

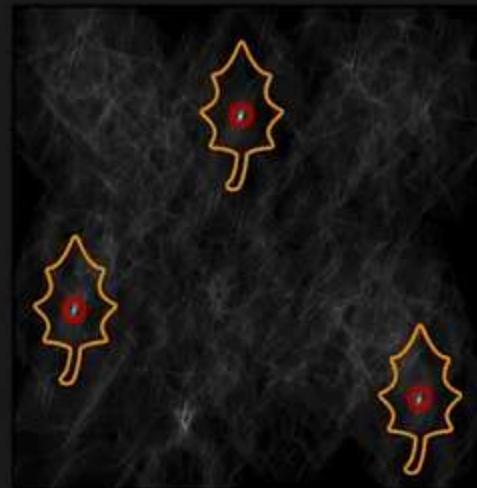
# Results



Model



Model Detected



Hough Transform  $A(x_c, y_c)$



Model



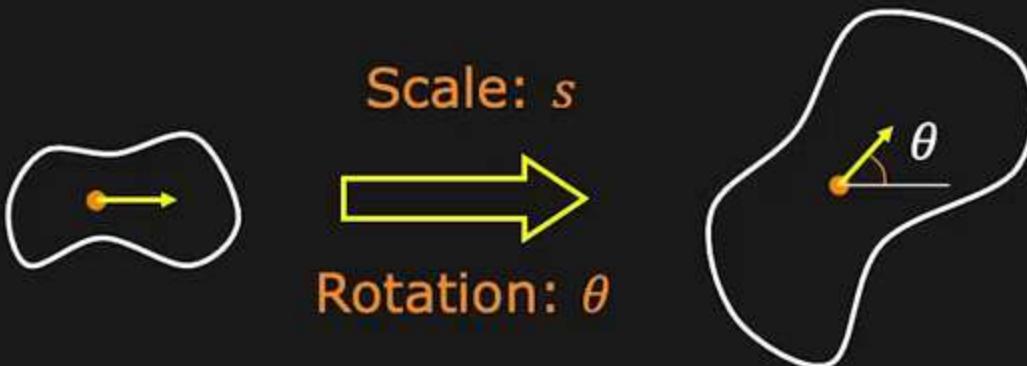
Model Detected



Hough Transform  $A(x_c, y_c)$

# Handling Scale And Rotation

---



Use Accumulation Array:  $A(x_c, y_c, s, \theta)$

$$x_c = x_i \pm r_k^i \cdot s \cos(\alpha_k^i + \theta)$$

$$y_c = y_i \pm r_k^i \cdot s \sin(\alpha_k^i + \theta)$$

$$A(x_c, y_c, s, \theta) = A(x_c, y_c, s, \theta) + 1$$

Huge Memory and Computationally Expensive!

# Hough Transform: Comments

---

- Works on disconnected edges
- Relatively insensitive to occlusion and noise
- Effective for simple shapes (lines, circles, etc.)
- Complex Shapes: Generalized Hough Transform
- Trade-off between work in image space and parameter space

# A Little Quiz

---

How would you recognize the following types of objects?



Objects on an assembly line

# A Little Quiz

---

How would you recognize the following types of objects?

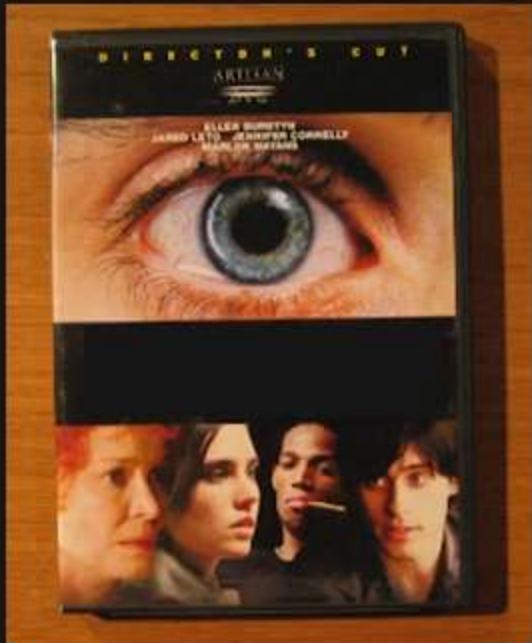


License plates

# A Little Quiz

---

How would you recognize the following types of objects?



Template



Rich 2D image

Find and Match “Interesting Points or Features”

# SIFT Detector

---

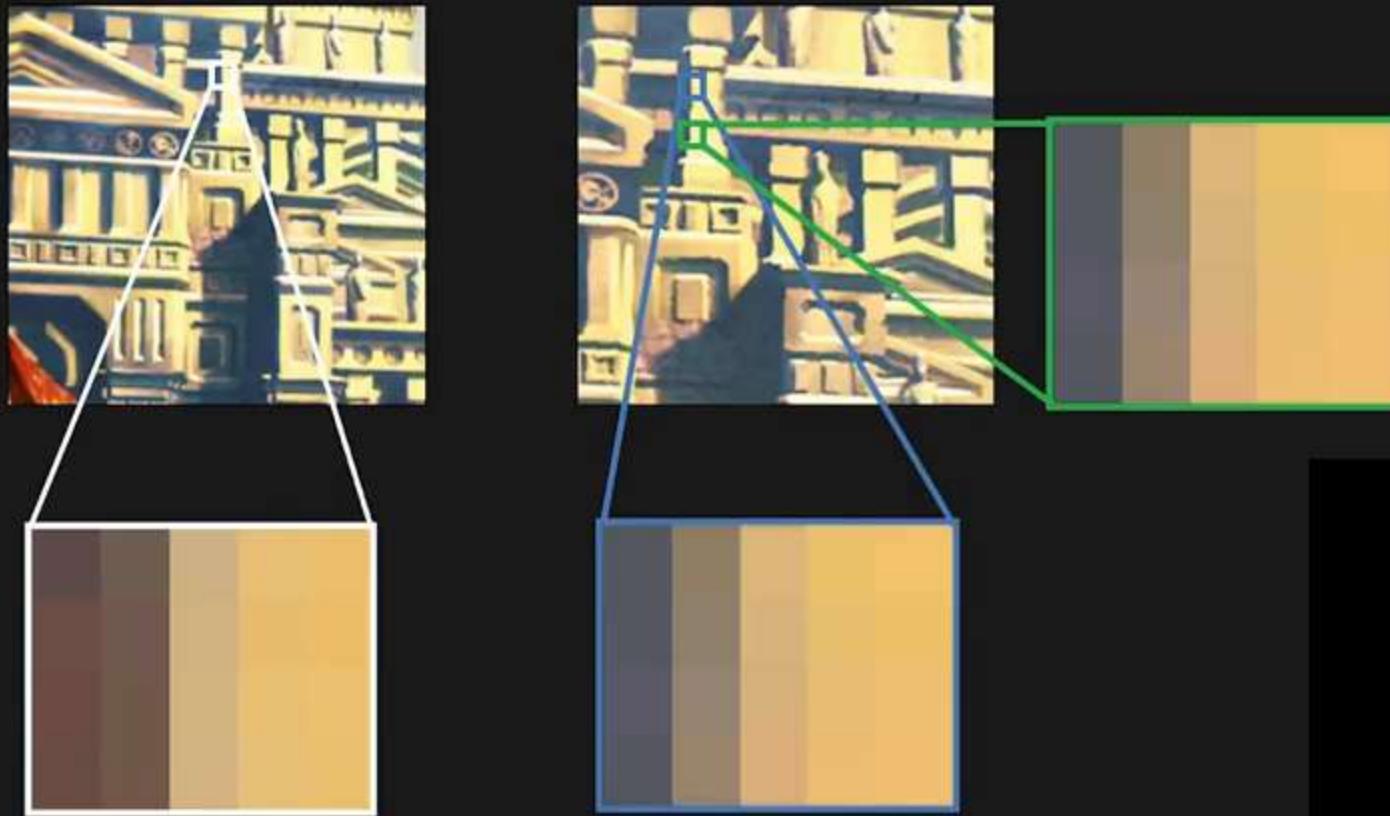
Scale Invariant Feature Transform (SIFT) and its use  
for image alignment and 2D object recognition.

## Topics:

- (1) What is an Interest Point?
- (2) Detecting Blobs
- (3) SIFT Detector
- (4) SIFT Descriptor

# Are Lines/Edges Interesting?

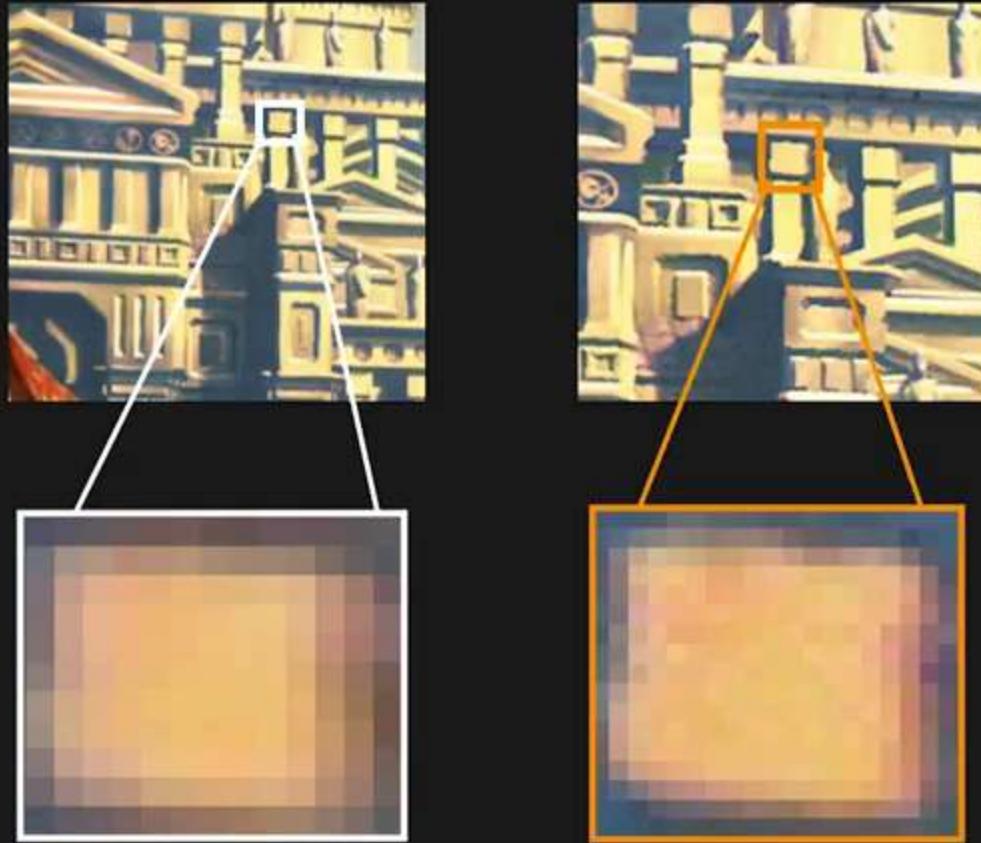
---



Cannot “**Localize**” an Edge

# Are Blobs Interesting?

---



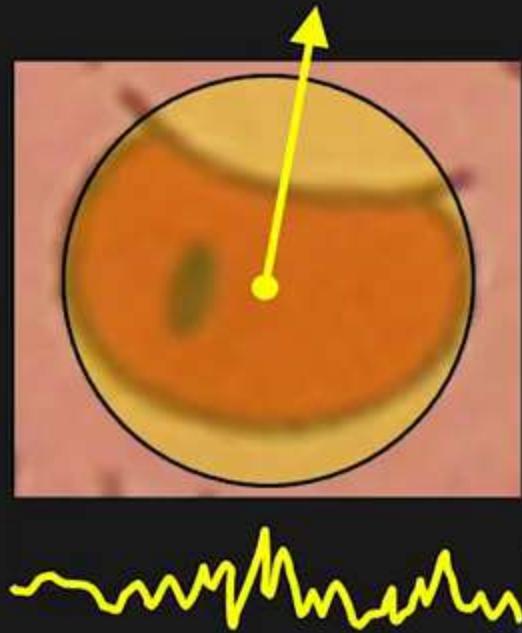
Yes! Blobs have **fixed position** and definite **size**.

# Blobs as Interest Points

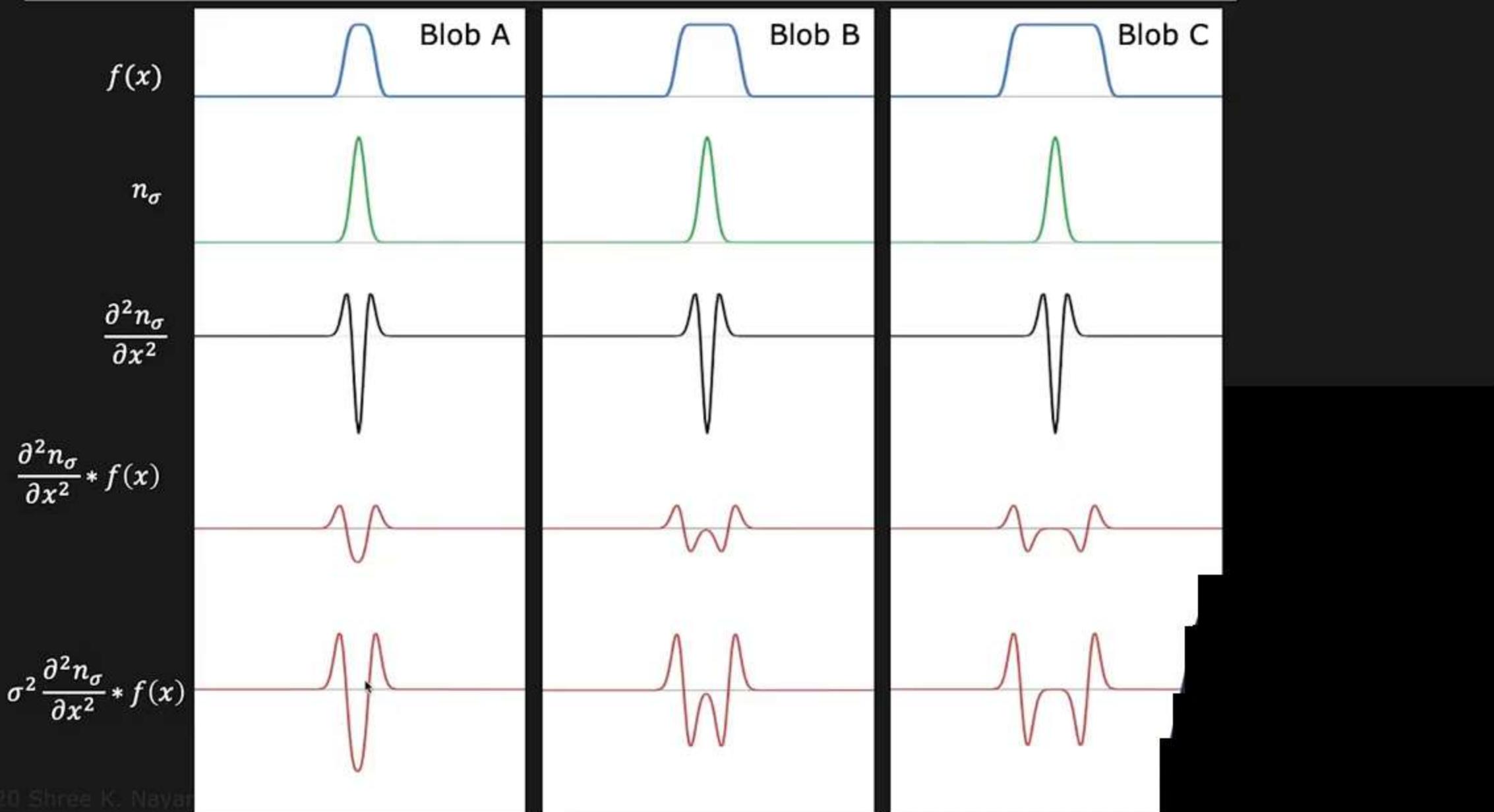
---

For a **Blob-like Feature** to be useful, we need to:

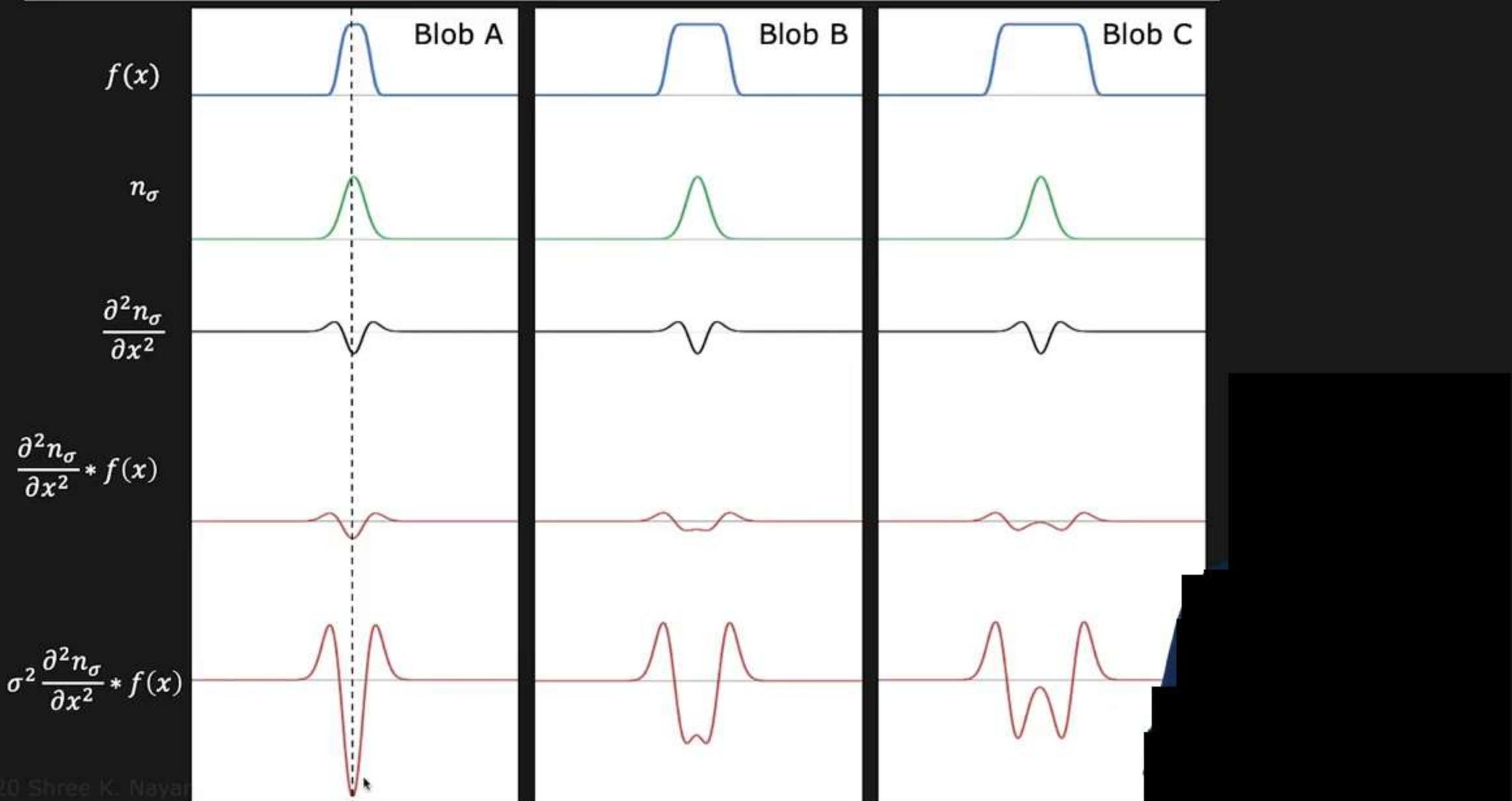
- Locate the blob
- Determine its **size**
- Determine its **orientation**
- Formulate a **description** or  
signature that is independent of  
size and orientation



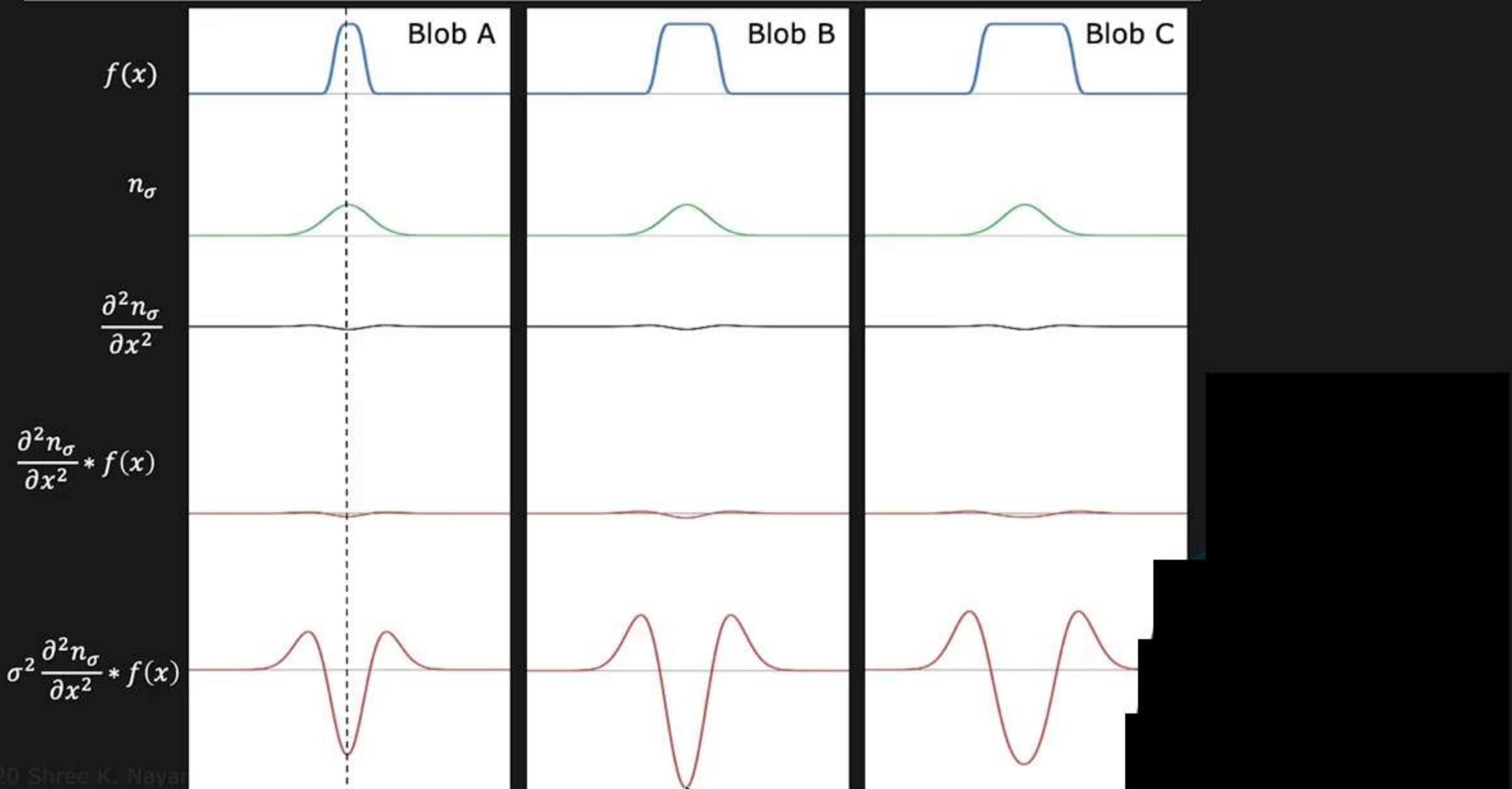
# 1D Blob and 2<sup>nd</sup> Derivative of Gaussian



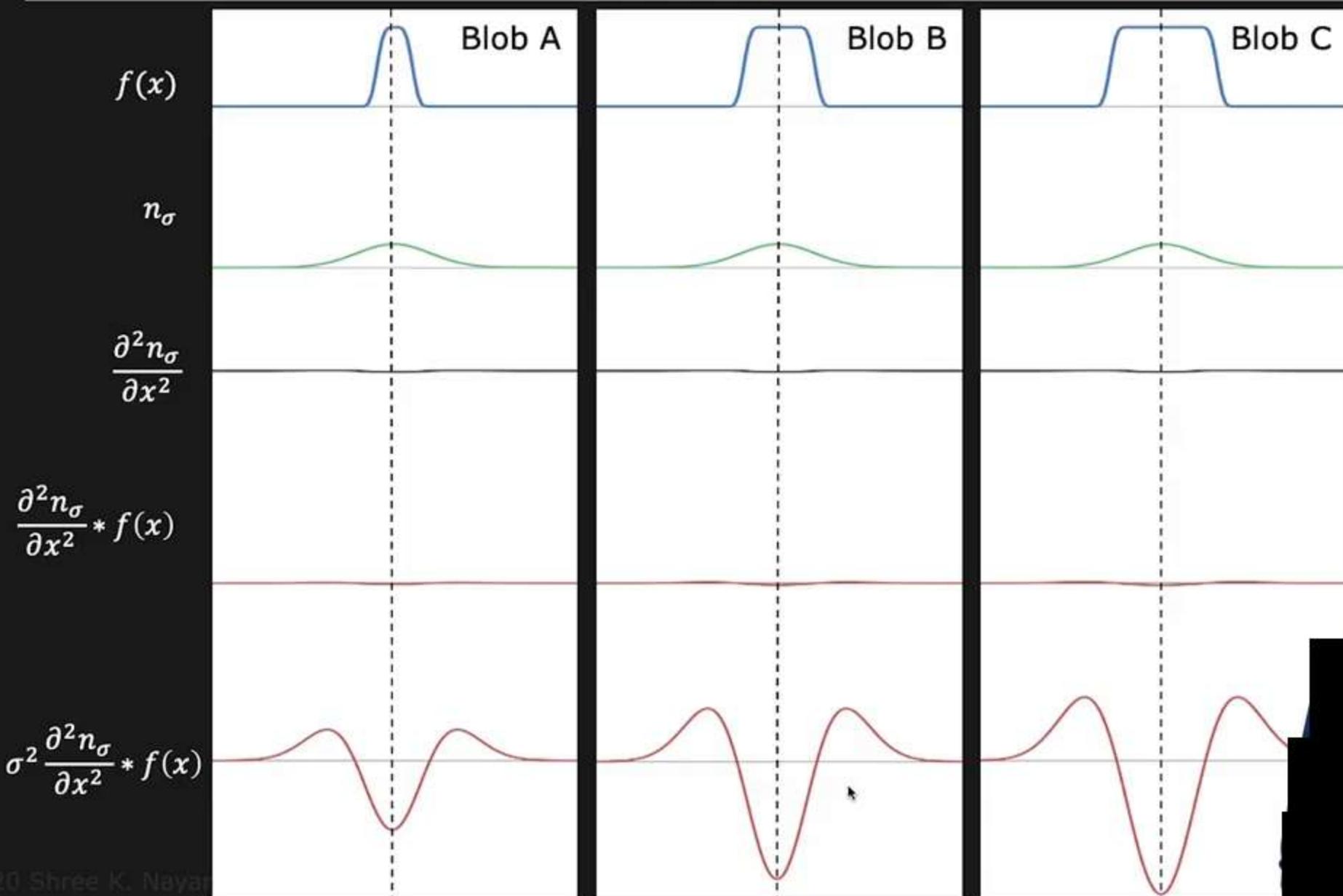
# 1D Blob and 2<sup>nd</sup> Derivative of Gaussian



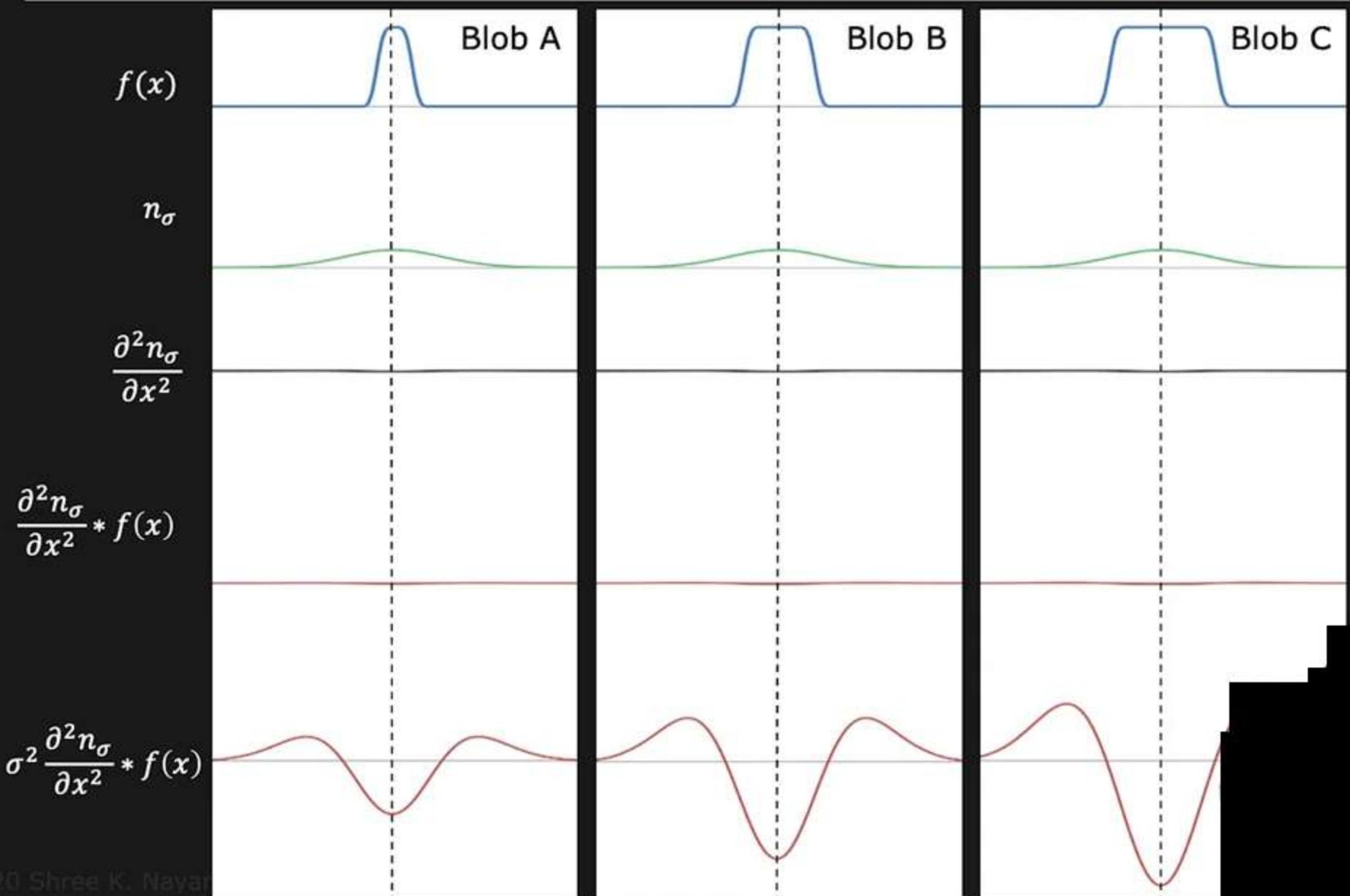
# 1D Blob and 2<sup>nd</sup> Derivative of Gaussian



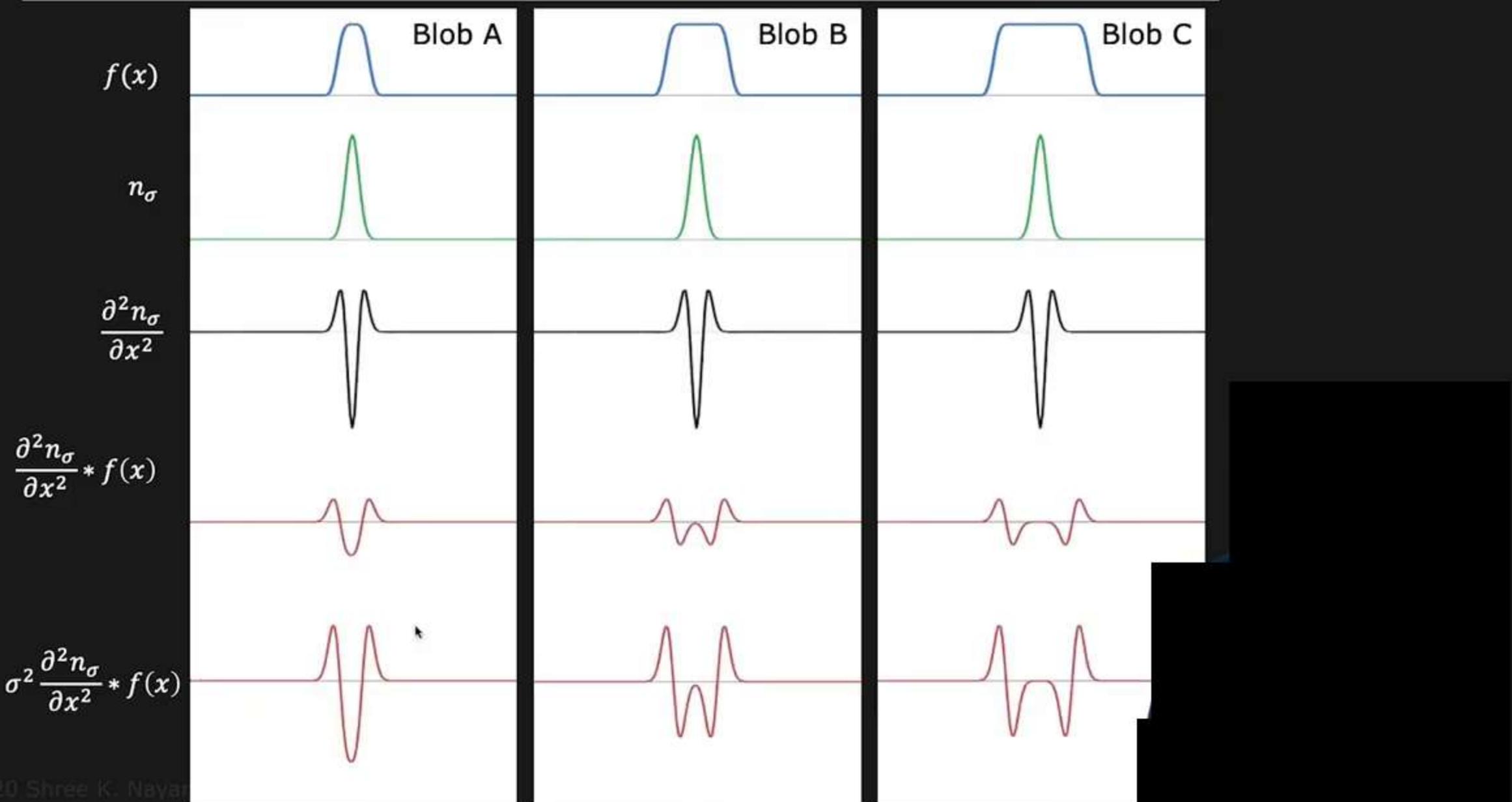
# 1D Blob and 2<sup>nd</sup> Derivative of Gaussian



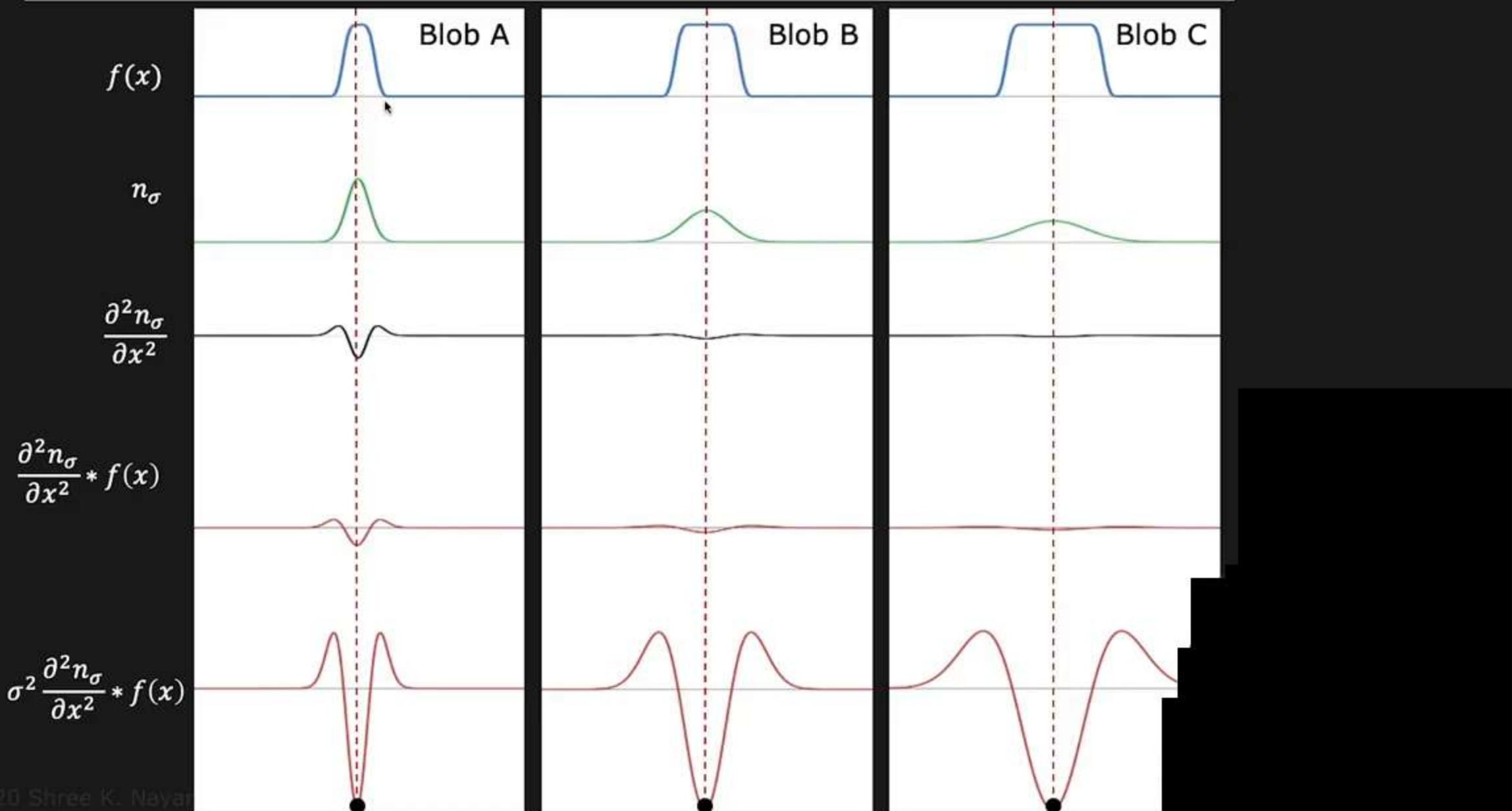
# 1D Blob and 2<sup>nd</sup> Derivative of Gaussian



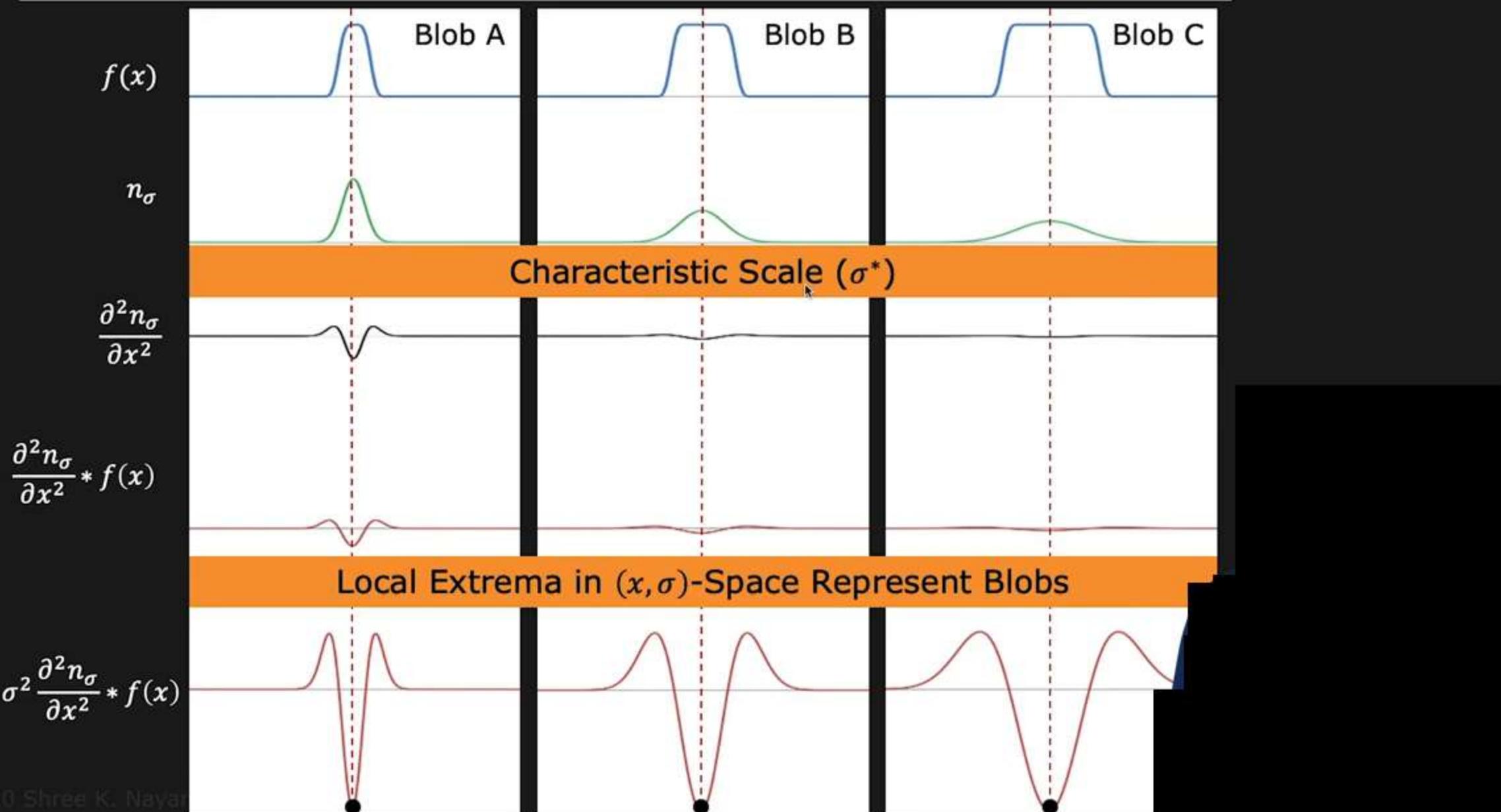
# 1D Blob and 2<sup>nd</sup> Derivative of Gaussian



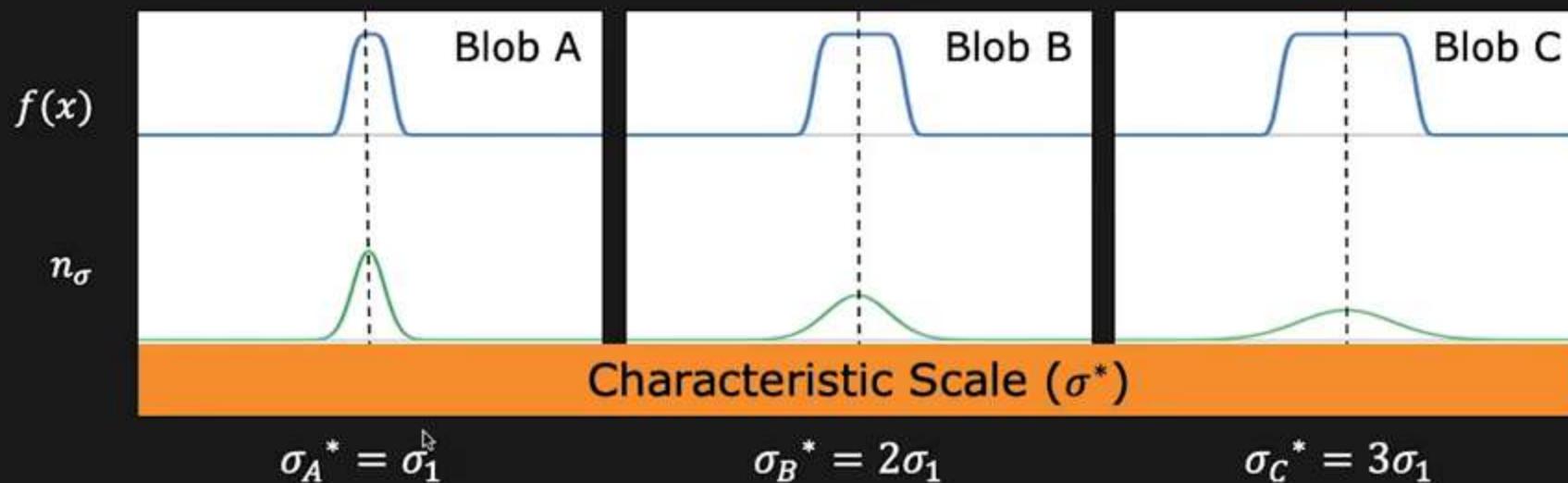
# 1D Blob and 2<sup>nd</sup> Derivative of Gaussian



# 1D Blob and 2<sup>nd</sup> Derivative of Gaussian



# Characteristic Scale and Blob Size



Characteristic Scale: The  $\sigma$  at which  $\sigma$ -normalized 2<sup>nd</sup> derivative attains its extreme value.

Characteristic Scale  $\propto$  Size of Blob

$$\frac{\text{Size of Blob A}}{\text{Size of Blob B}} = \frac{\sigma_A^*}{\sigma_B^*}; \quad \frac{\text{Size of Blob B}}{\text{Size of Blob C}} = \frac{\sigma_B^*}{\sigma_C^*}$$

# 1D Blob Detection Summary

---

Given: 1D signal  $f(x)$

Compute:  $\sigma^2 \frac{\partial^2 n_\sigma}{\partial x^2} * f(x)$  at many scales  $(\sigma_0, \sigma_1, \sigma_2, \dots, \sigma_k)$ .

Find: 
$$(x^*, \sigma^*) = \arg \max_{(x, \sigma)} \left| \sigma^2 \frac{\partial^2 n_\sigma}{\partial x^2} * f(x) \right|$$

$x^*$ : Blob Position

$\sigma^*$ : Characteristic Scale (Blob Size)

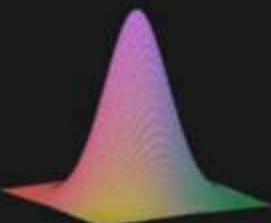
# 2D Blob Detector

Normalized Laplacian of Gaussian (NLoG) is used as the 2D equivalent for Blob Detection.

Laplacian

$$\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$$

Gaussian



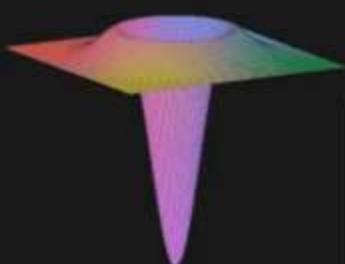
$$n_{\sigma}$$

LoG



$$\nabla^2 n_{\sigma}$$

NLoG



$$\sigma^2 \nabla^2 n_{\sigma}$$

Location of Blobs given by Local Extrema after applying Normalized Laplacian of Gaussian at many scales.

# Scale-Space

 $S(x, y, \sigma_0)$  $S(x, y, \sigma_1)$  $S(x, y, \sigma_2)$  $S(x, y, \sigma_3)$ 

...

**Scale Space:** Stack created by filtering an image with Gaussians of different sigma ( $\sigma$ )

$$S(x, y, \sigma) = n(x, y, \sigma) * I(x, y)$$



# Creating Scale-Space



$S(x, y, \sigma_0)$



$S(x, y, \sigma_1)$



$S(x, y, \sigma_2)$



$S(x, y, \sigma_3)$

...

Increasing  $\sigma$ , Higher Scale, Lower Resolution

Selecting sigmas to generate the scale-space:

$$\sigma_k = \sigma_0 s^k \quad k = 0, 1, 2, 3, \dots$$

$s$ : Constant multiplier

$\sigma_0$ : Initial Scale

# Blob Detection using Local Extrema



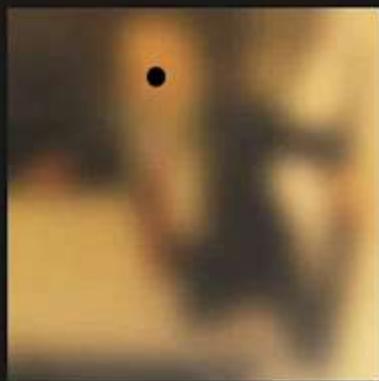
$S(x, y, \sigma_0)$



$S(x, y, \sigma_1)$



$S(x, y, \sigma_2)$



$S(x, y, \sigma_3)$

...

$$\sigma^2 \nabla^2 S(x, y, \sigma) \\ (NLog * I(x, y))$$



# Blob Detection using Local Extrema

---

 $S(x, y, \sigma_0)$  $S(x, y, \sigma_1)$  $S(x, y, \sigma_2)$  $S(x, y, \sigma_3)$ 

...

# 2D Blob Detection Summary

---

Given an image  $I(x, y)$

Convolve the image using NLoG at many scales  $\sigma$

Find:

$$(x^*, y^*, \sigma^*) = \arg \max_{(x, y, \sigma)} |\sigma^2 \nabla^2 n_\sigma * I(x, y)|$$

$(x^*, y^*)$ : Position of the blob

$\sigma^*$ : Size of the blob

# Computing the Principal Orientation

---

Use the histogram of gradient directions

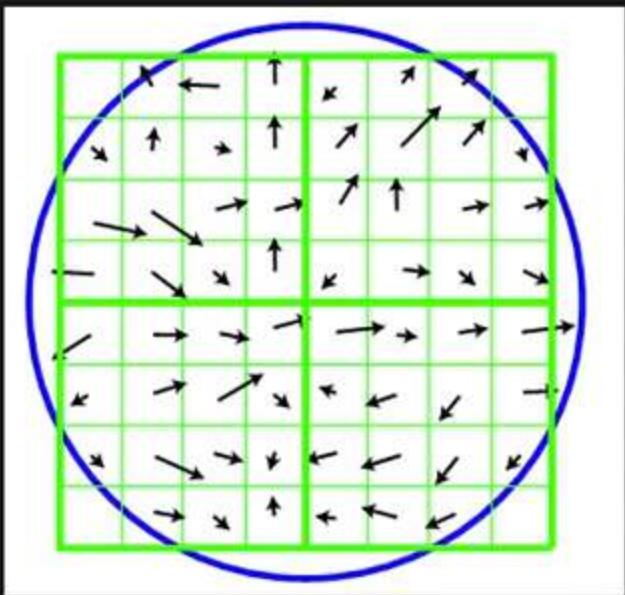
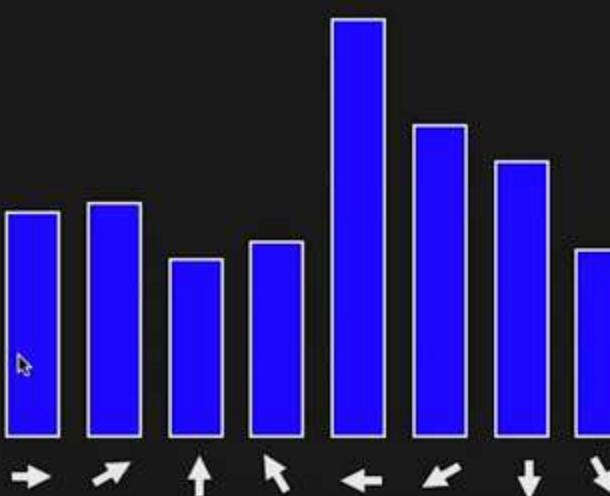


Image gradient directions



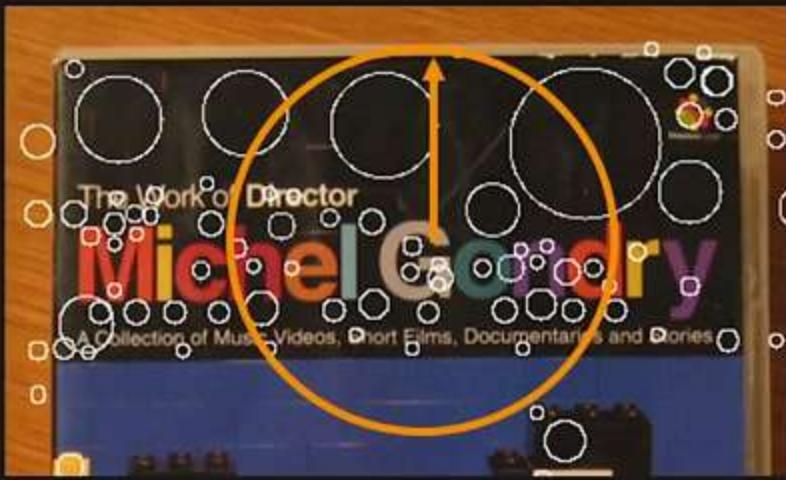
$$\theta = \tan^{-1} \left( \frac{\partial I}{\partial y} / \frac{\partial I}{\partial x} \right)$$



# SIFT Rotation Invariance

---

Use the principal orientation to undo rotation



# SIFT Rotation Invariance

---

Use the principal orientation to undo rotation



# SIFT Descriptor

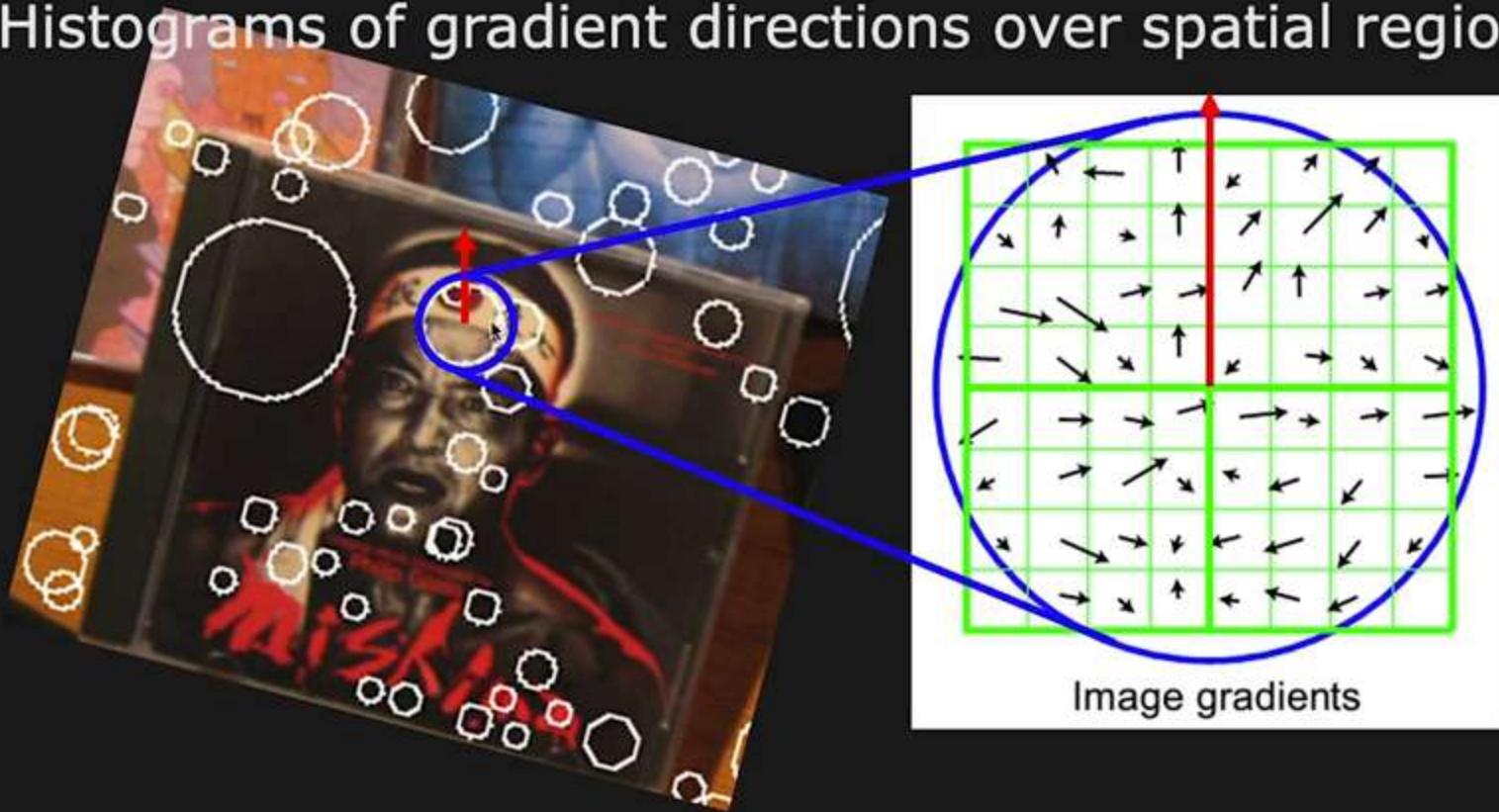
---

Histograms of gradient directions over spatial regions



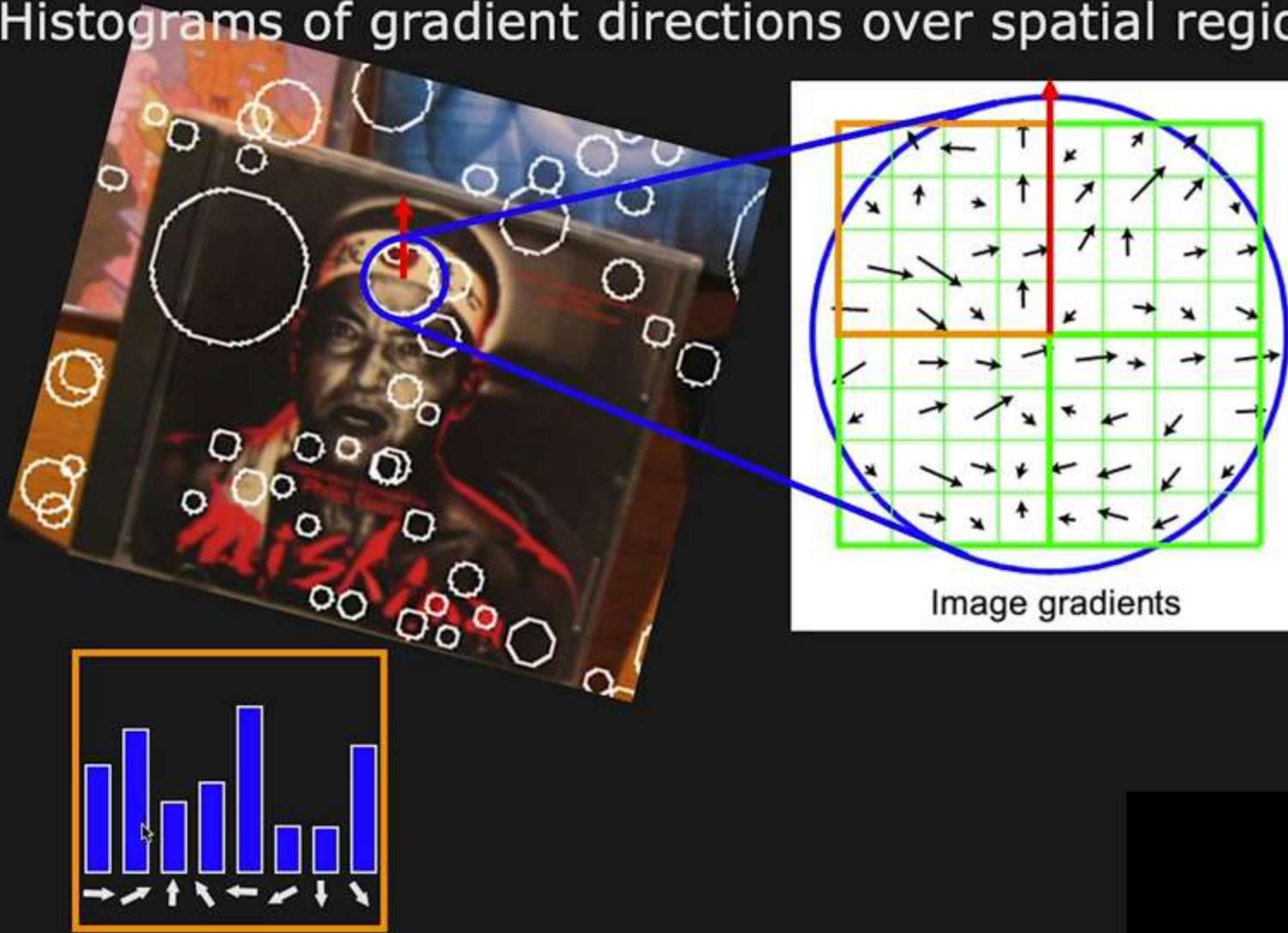
# SIFT Descriptor

Histograms of gradient directions over spatial regions



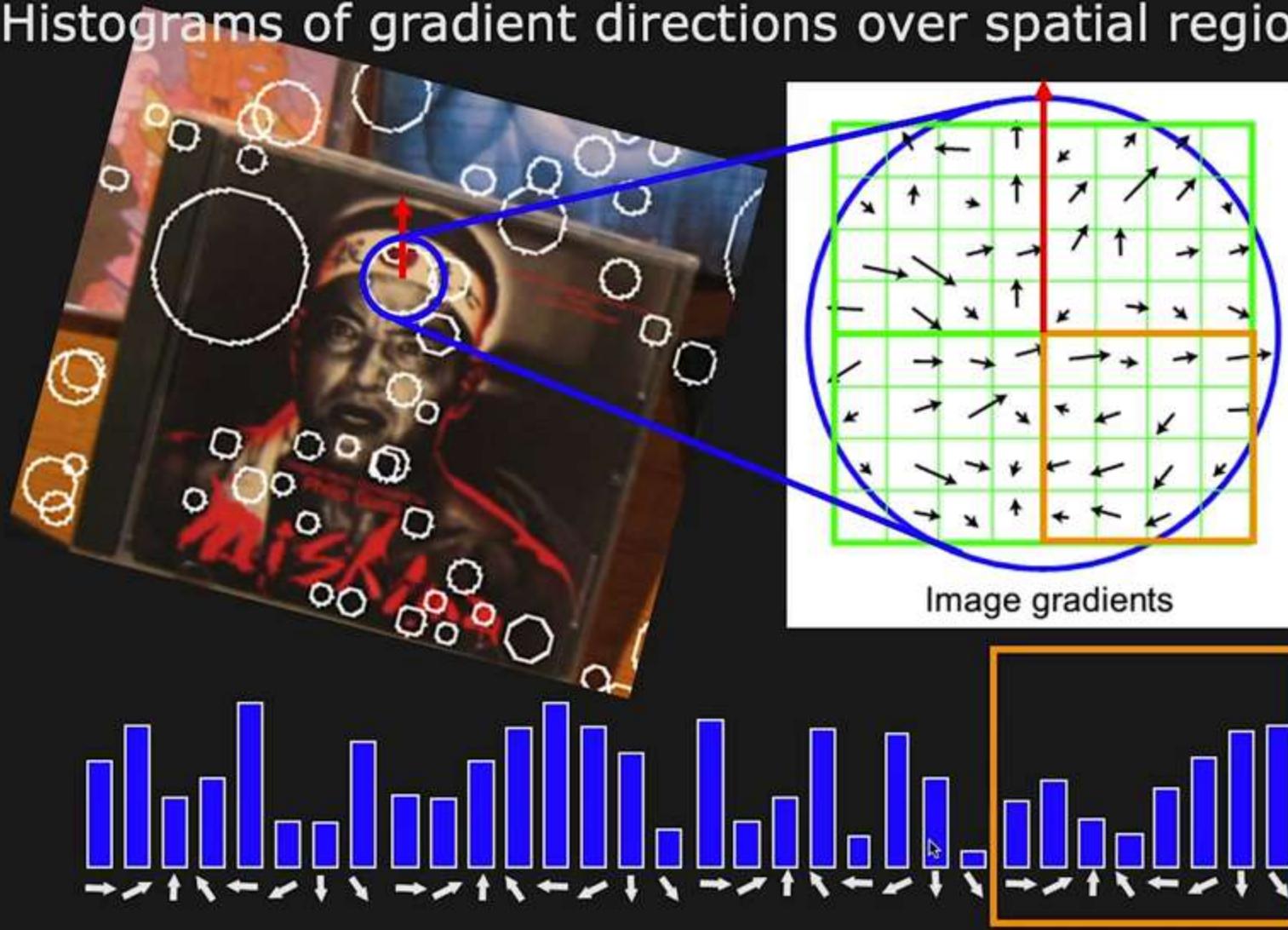
# SIFT Descriptor

Histograms of gradient directions over spatial regions



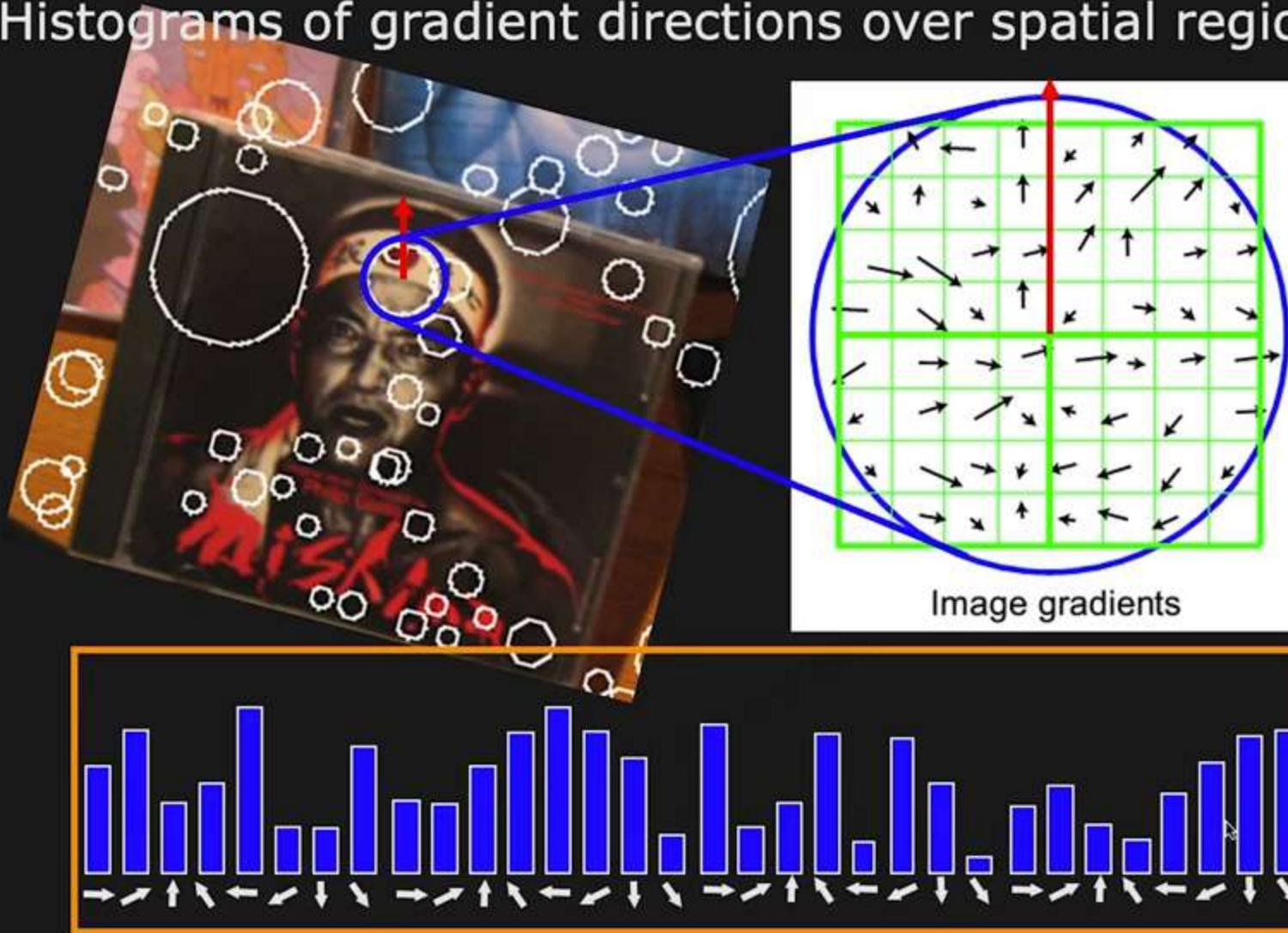
# SIFT Descriptor

Histograms of gradient directions over spatial regions



# SIFT Descriptor

Histograms of gradient directions over spatial regions



Normalized Histogram: Invariant to Rotation, Scale, Brightness

# Comparing SIFT Descriptors

---

Essentially comparing two arrays of data.

Let  $H_1(k)$  and  $H_2(k)$  be two arrays of data of length  $N$ .

L2 Distance:

$$d(H_1, H_2) = \sqrt{\sum_k (H_1(k) - H_2(k))^2}$$

# Comparing SIFT Descriptors

---

Essentially comparing two arrays of data.

Let  $H_1(k)$  and  $H_2(k)$  be two arrays of data of length  $N$ .

Normalized Correlation:

$$d(H_1, H_2) = \frac{\sum_k [(H_1(k) - \bar{H}_1)(H_2(k) - \bar{H}_2)]}{\sqrt{\sum_k (H_1(k) - \bar{H}_1)^2} \sqrt{\sum_k (H_2(k) - \bar{H}_2)^2}}$$

where:  $\bar{H}_i = \frac{1}{N} \sum_{k=1}^N H_i(k)$

Larger the distance metric, better the match.

Perfect match when  $d(H_1, H_2) = 0$

# Comparing SIFT Descriptors

---

Essentially comparing two arrays of data.

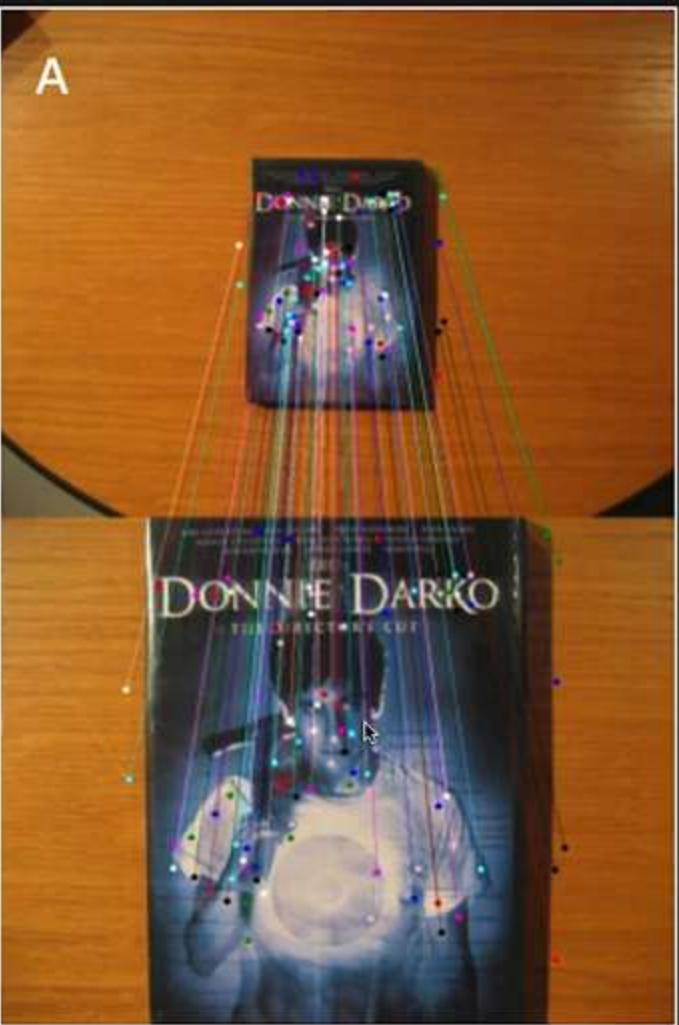
Let  $H_1(k)$  and  $H_2(k)$  be two arrays of data of length  $N$ .

Intersection:

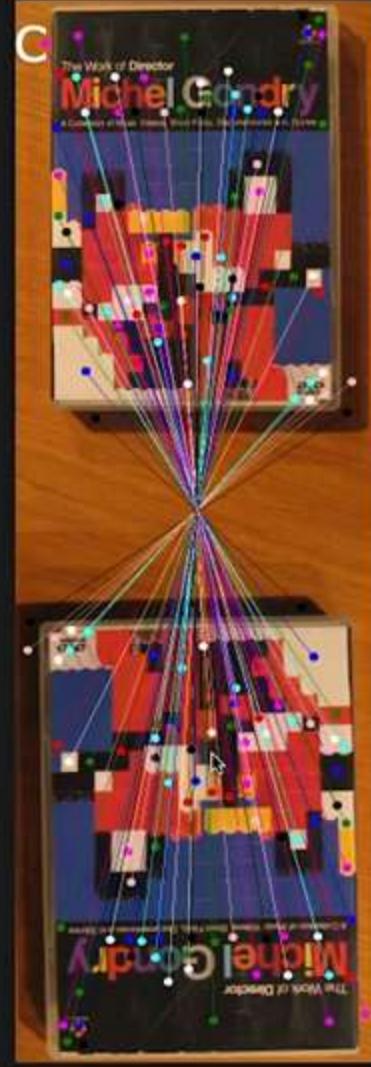
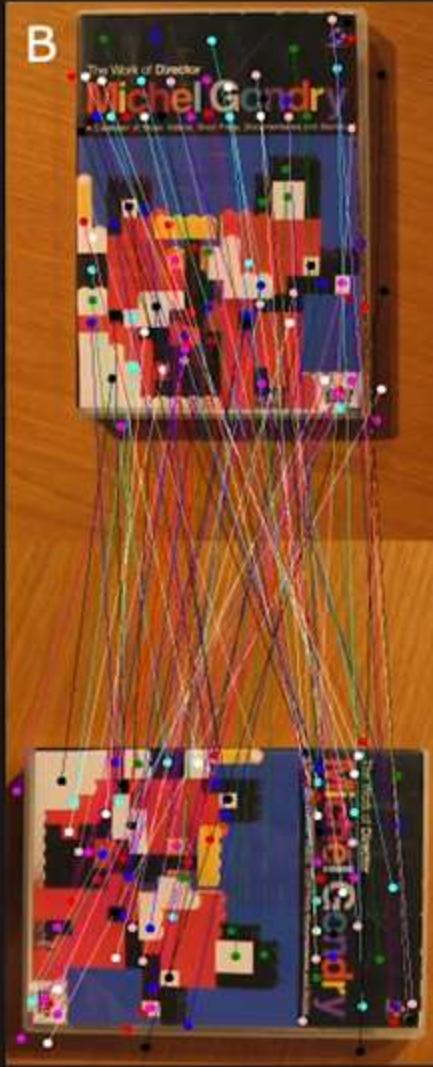
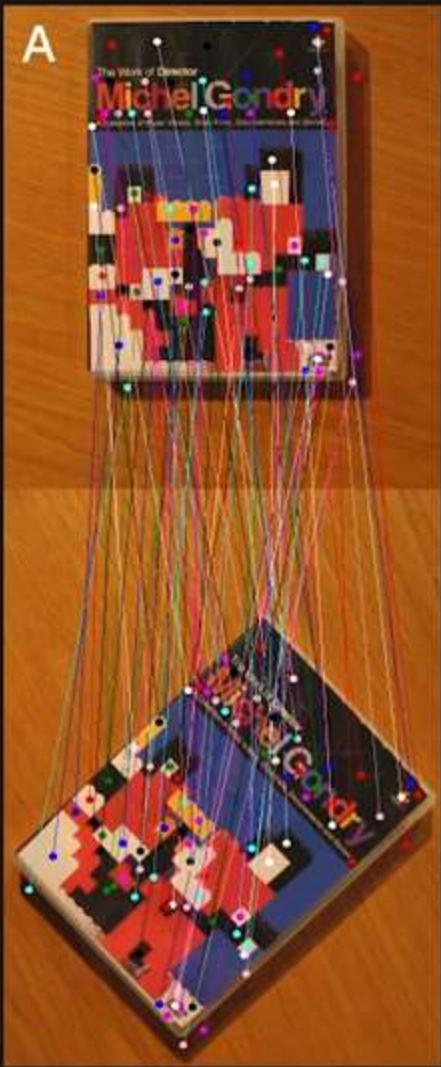
$$\hat{d}(H_1, H_2) = \sum_k \min(H_1(k), H_2(k))$$

Larger the distance metric, better the match.

# SIFT Results: Scale Invariance



# SIFT Results: Rotation Invariance



# SIFT Results: Robustness to Clutter

---



# Panorama Stitching using SIFT

---



Image 1



Image 2

# Panorama Stitching using SIFT

---



Warp and combine images to create a larger image

# Auto Collage using SIFT

---

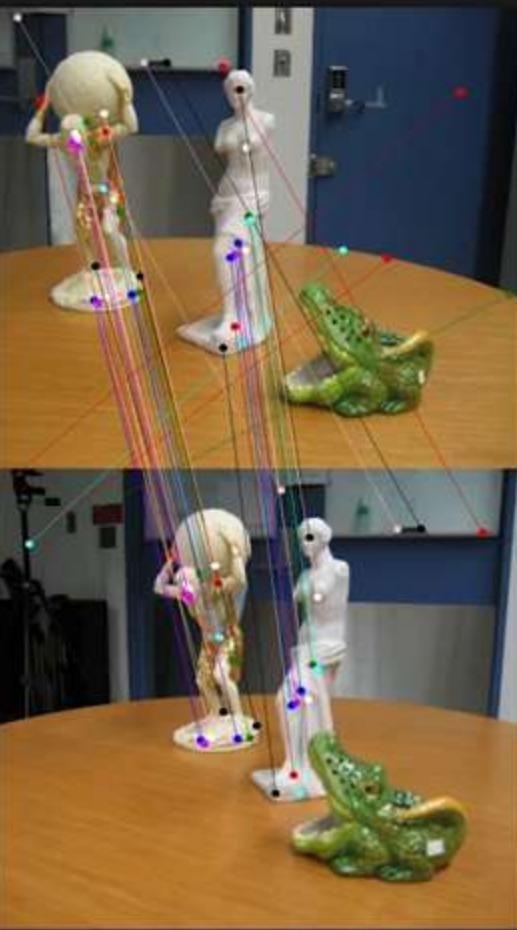
Collage



# SIFT for 3D Objects?



No Change in Viewpoint



30° Change in Viewpoint



90° Change in Viewpoir

SIFT is reliable for only small changes in viewpoint

# Image Stitching

---



Image 1



Image 2



Image 3

How would you align these images?

# Image Stitching



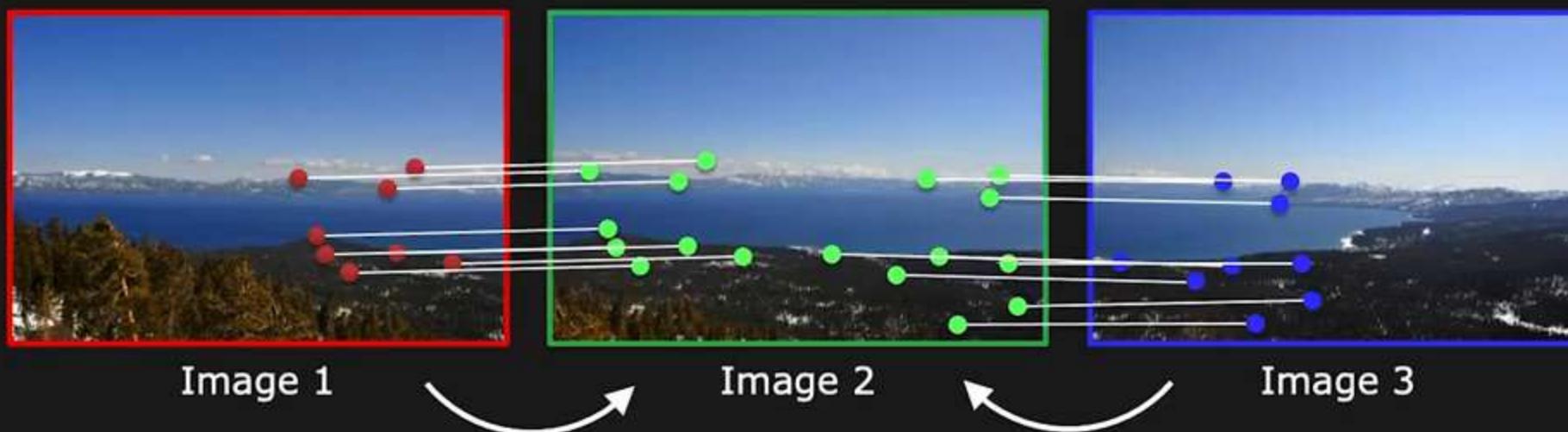
Image 1

Image 2

Image 3

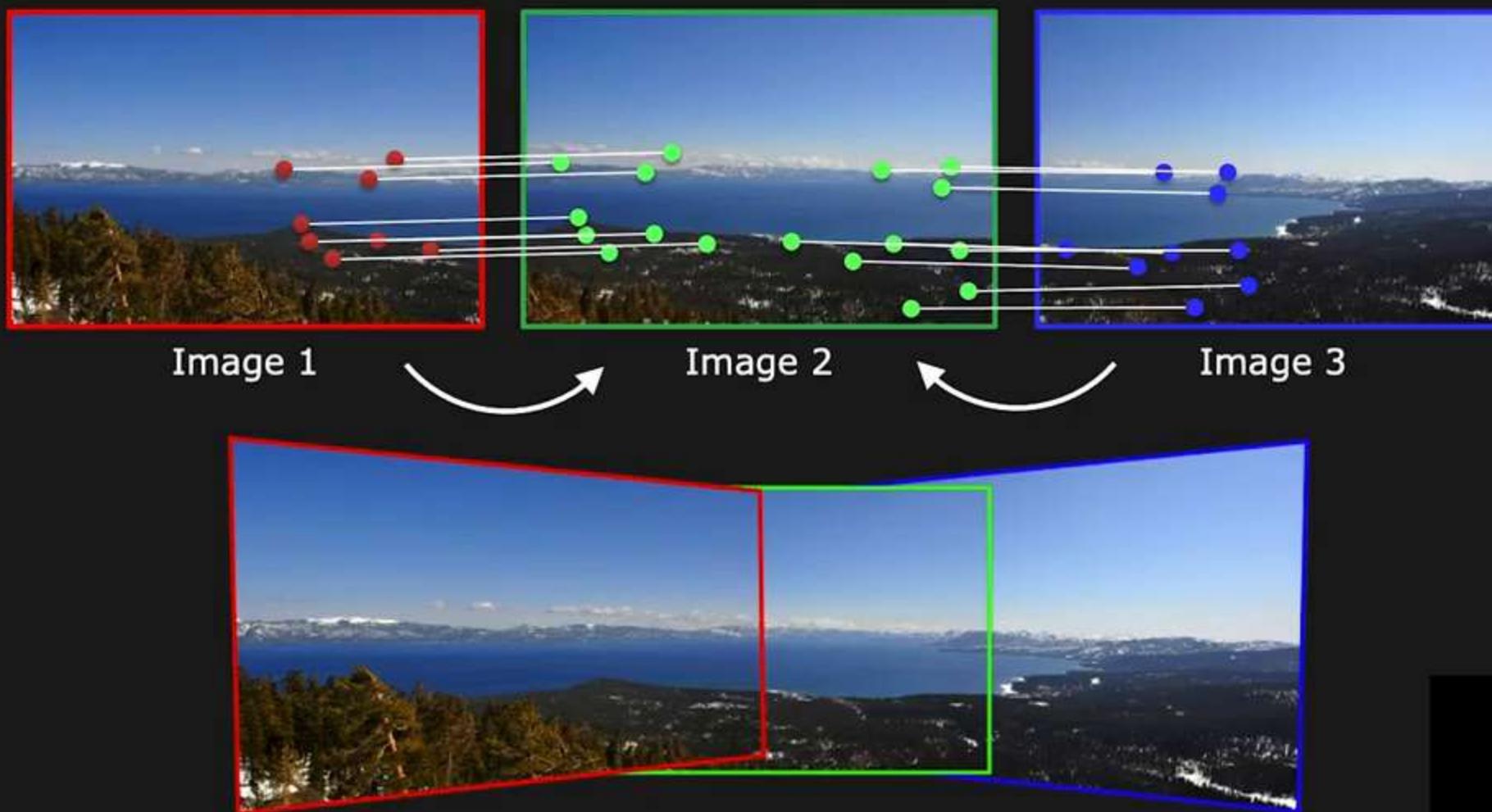
Find corresponding points  
(using feature detectors like SIFT)

# Image Stitching



Find geometric relationship between the images

# Image Stitching



Warp images so that corresponding points align

# Image Stitching

---



Overlaid Aligned Images



Blended Images

Blend images to remove hard seams

# Image Stitching

---

Combine multiple photos to create a larger photo

Topics:

- (1) 2x2 Image Transformations
- (2) 3x3 Image Transformations
- (3) Computing Homography
- (4) Dealing with Outliers: RANSAC
- (5) Warping and Blending Images

# Global Warping/Transformation



Translation



Rotation



Scaling and Aspect

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



Affine



Projective



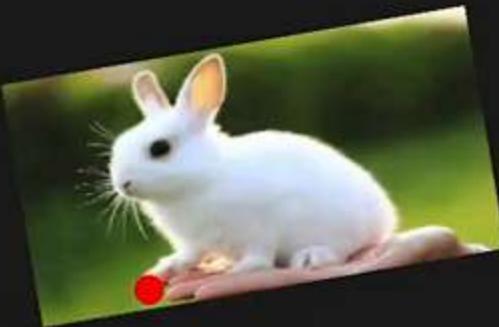
Barrel

Transformation  $T$  is the same over entire domain

Often can be described by just a few parameters

# 2x2 Linear Transformations

---



$$\mathbf{p}_1 = (x_1, y_1)$$

$$\mathbf{p}_2 = (x_2, y_2)$$

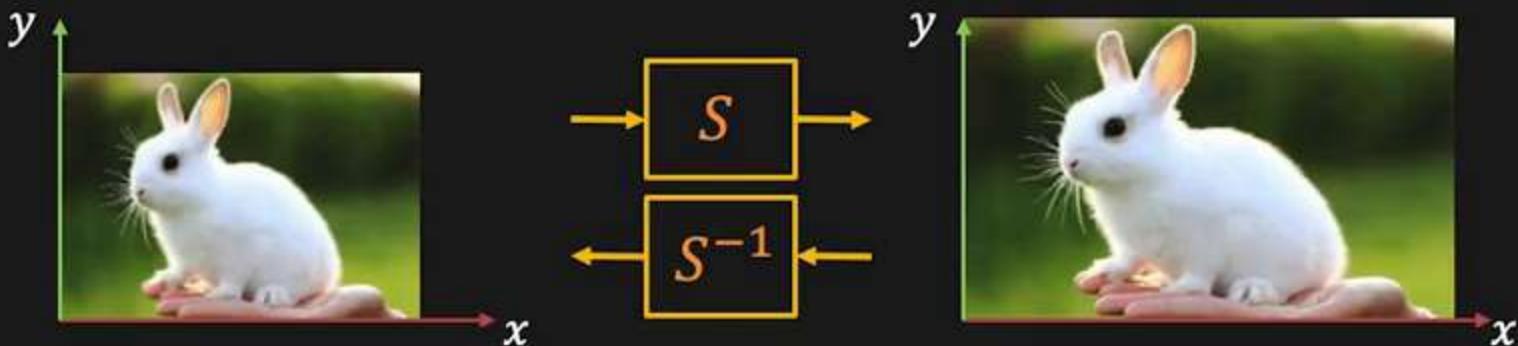
$T$  can be represented by a matrix.

$$\mathbf{p}_2 = T\mathbf{p}_1$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = T \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

# Scaling (Stretching or Squishing)



Forward:

$$x_2 = ax_1 \quad y_2 = by_1$$

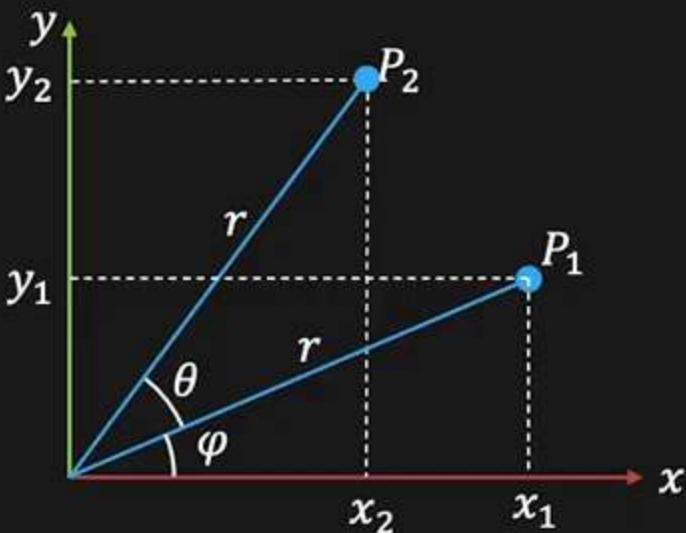
Inverse:

$$x_1 = \frac{1}{a}x_2 \quad y_1 = \frac{1}{b}y_2$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = S^{-1} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1/a & 0 \\ 0 & 1/b \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

# 2D Rotation



$$x_1 = r \cos(\varphi)$$

$$y_1 = r \sin(\varphi)$$

$$x_2 = r \cos(\varphi + \theta)$$

$$x_2 = r \cos \varphi \cos \theta - r \sin \varphi \sin \theta$$

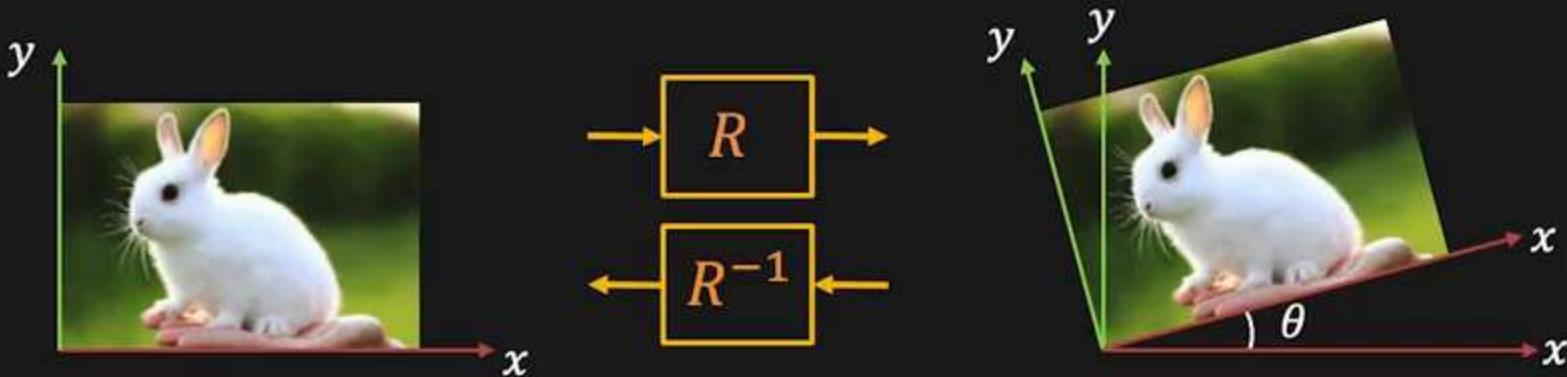
$$x_2 = x_1 \cos \theta - y_1 \sin \theta$$

$$y_2 = r \sin(\varphi + \theta)$$

$$y_2 = r \cos \varphi \sin \theta + r \sin \varphi \cos \theta$$

$$y_2 = x_1 \sin \theta + y_1 \cos \theta$$

# Rotation



Forward:

$$x_2 = x_1 \cos\theta - y_1 \sin\theta$$

$$y_2 = x_1 \sin\theta + y_1 \cos\theta$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = R \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

Inverse:

$$x_1 = x_2 \cos\theta + y_2 \sin\theta$$

$$y_1 = -x_2 \sin\theta + y_2 \cos\theta$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = R^{-1} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

# Skew

---



Horizontal Skew:

$$x_2 = x_1 + m_x y_1$$

$$y_2 = y_1$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S_x \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & m_x \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

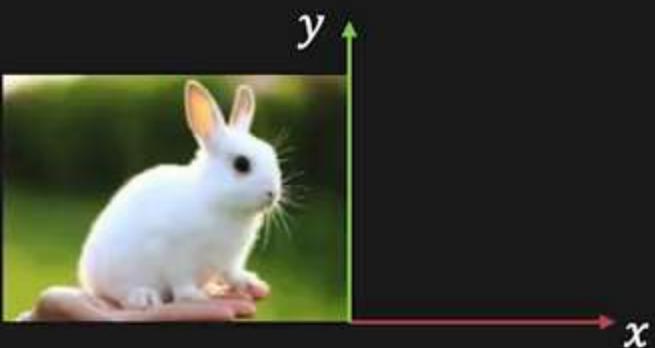
Vertical Skew:

$$x_2 = x_1$$

$$y_2 = m_y x_1 + y_1$$

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = S_x \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ m_y & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

# Mirror



Mirror about Y-axis:

$$x_2 = -x_1$$

$$y_2 = y_1$$

$$M_y = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

Mirror about line  $y = x$ :

$$x_2 = y_1$$

$$y_2 = x_1$$

$$M_{xy} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

# 2x2 Matrix Transformations

---

Any transformation of the form:

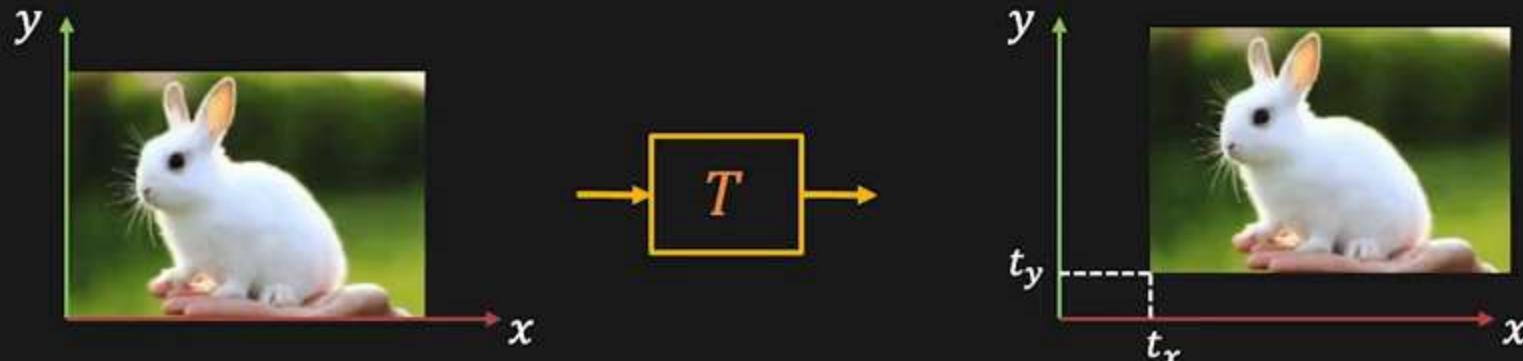
$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

- Origin maps to the origin
- Lines map to lines
- Parallel lines remain parallel
- Closed under composition

$$\left. \begin{array}{l} \mathbf{p}_2 = T_{21}\mathbf{p}_1 \\ \mathbf{p}_3 = T_{32}\mathbf{p}_2 \\ \mathbf{p}_3 = T_{31}\mathbf{p}_1 \end{array} \right\} \quad \mathbf{p}_3 = T_{32}\mathbf{p}_2 = T_{32}T_{21}\mathbf{p}_1 \Rightarrow T_{31} = T_{32}T_{21}$$

# Translation

---



$$x_2 = x_1 + t_x \quad y_2 = y_1 + t_y$$

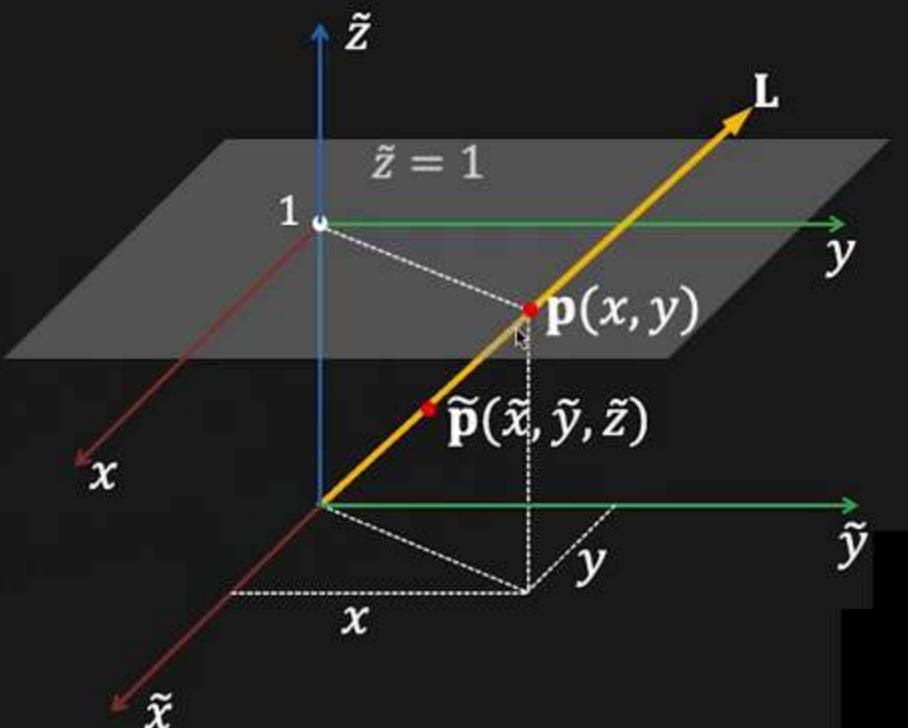
Can translation be expressed as a  $2 \times 2$  matrix? No.

# Homogenous Coordinates

The **homogenous** representation of a 2D point  $\mathbf{p} = (x, y)$  is a 3D point  $\tilde{\mathbf{p}} = (\tilde{x}, \tilde{y}, \tilde{z})$ . The third coordinate  $\tilde{z} \neq 0$  is fictitious such that:

$$x = \frac{\tilde{x}}{\tilde{z}} \quad y = \frac{\tilde{y}}{\tilde{z}}$$

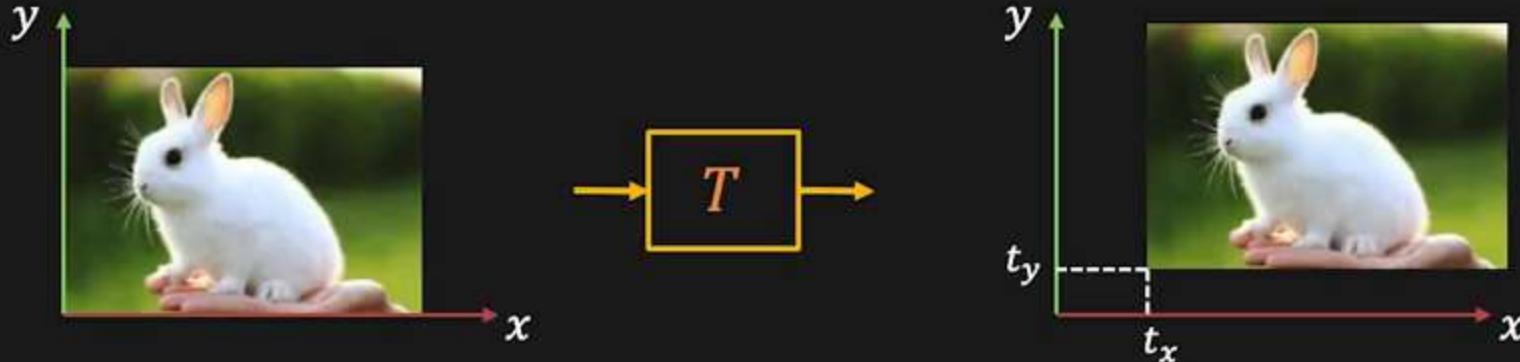
$$\mathbf{p} \equiv \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{z}x \\ \tilde{z}y \\ \tilde{z} \end{bmatrix} \equiv \begin{bmatrix} \tilde{x} \\ \tilde{y} \\ \tilde{z} \end{bmatrix} = \tilde{\mathbf{p}}$$



Every point on line  $\mathbf{L}$  (except origin) represents the homogenous coordinate of  $\mathbf{p}(x, y)$

# Translation

---



$$x_2 = x_1 + t_x \quad y_2 = y_1 + t_y$$

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

# Scaling, Rotation, Skew, Translation

---

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Scaling

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} 1 & m_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Skew

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Translation

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

Rotation

Composition of these transformations?

# Affine Transformation

---

Any transformation of the form:

$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix}$$



# Affine Transformation

---

Any transformation of the form:

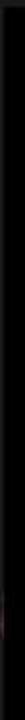
$$\begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix}$$

- Origin does not necessarily map to the origin
- Lines map to lines
- Parallel lines remain parallel
- Closed under composition

# Projective Transformation

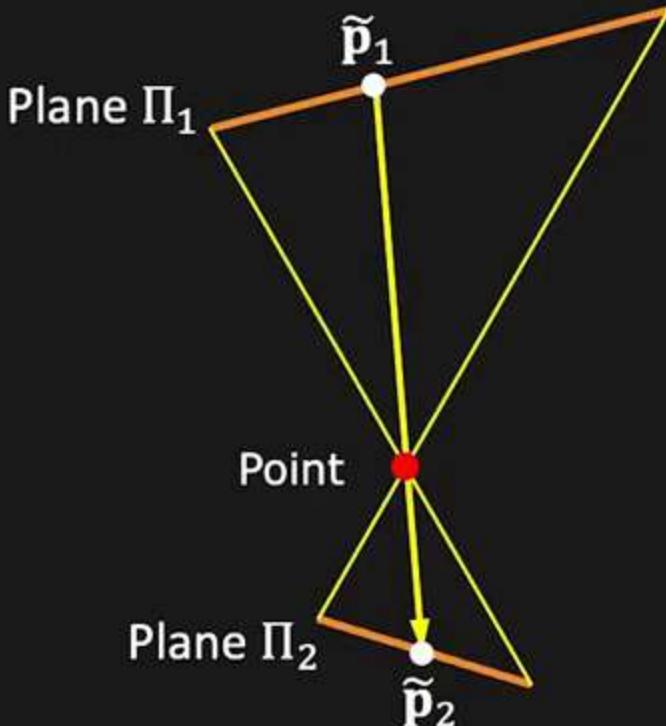
Any transformation of the form:

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix} \quad \tilde{\mathbf{p}}_2 = H\tilde{\mathbf{p}}_1$$



# Projective Transformation

Mapping of one plane to another through a point



$$\tilde{\mathbf{p}}_2 = H\tilde{\mathbf{p}}_1$$

$$\begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix}$$

Same as imaging a plane through a pinhole

# Projective Transformation

---

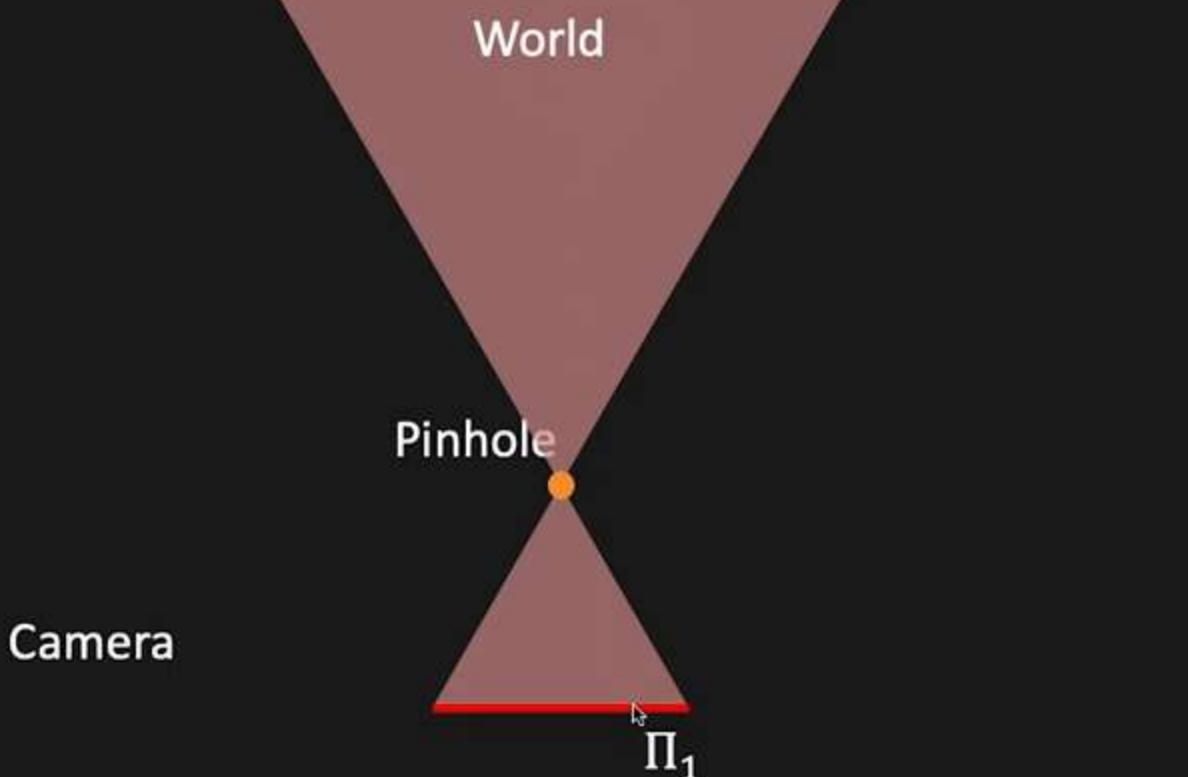
Homography can only be defined up to a scale.

$$\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x}_2 \\ \tilde{y}_2 \\ \tilde{z}_2 \end{bmatrix} \equiv k \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} \tilde{x}_1 \\ \tilde{y}_1 \\ \tilde{z}_1 \end{bmatrix}$$

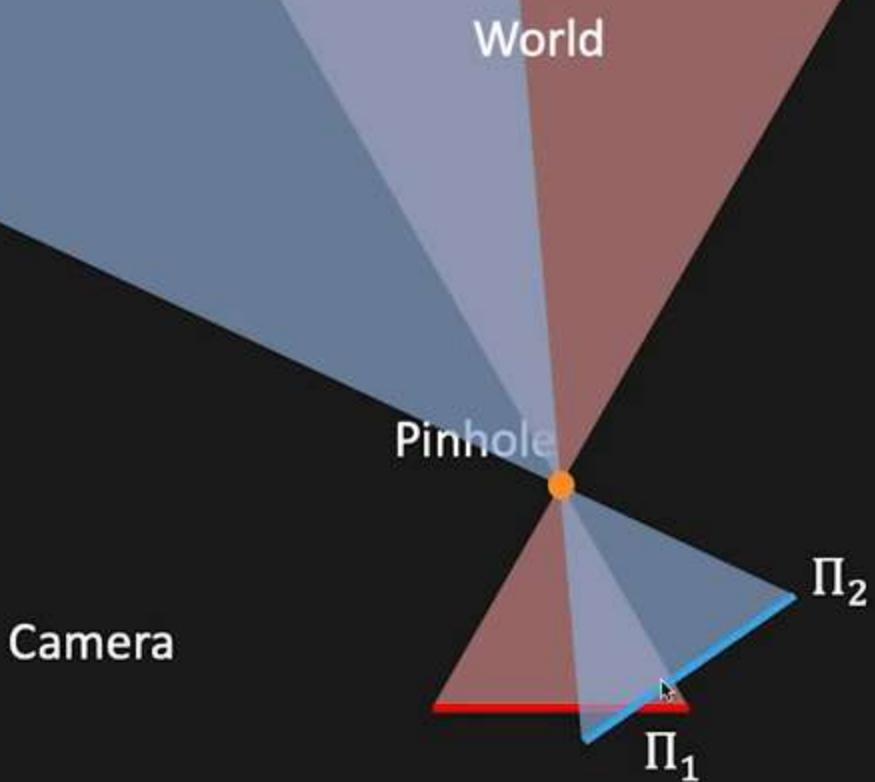
If we fix scale such that  $\sqrt{\sum(h_{ij})^2} = 1$  then 8 free parameters

- Origin does not necessarily map to the origin
- Lines map to lines
- Parallel lines do not necessarily remain parallel
- Closed under composition

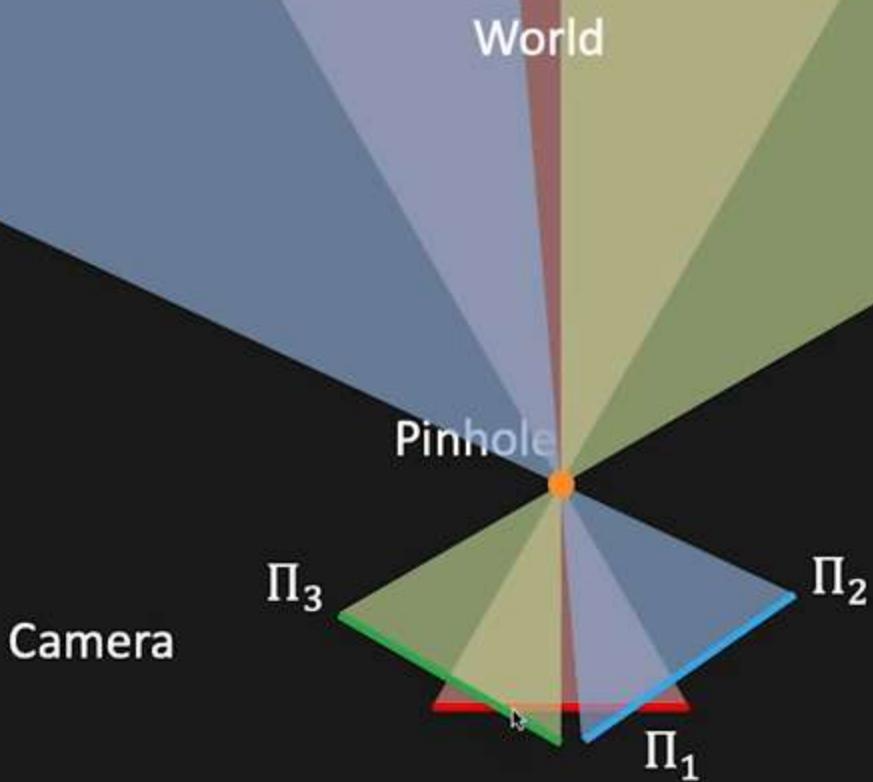
# Homography Composition



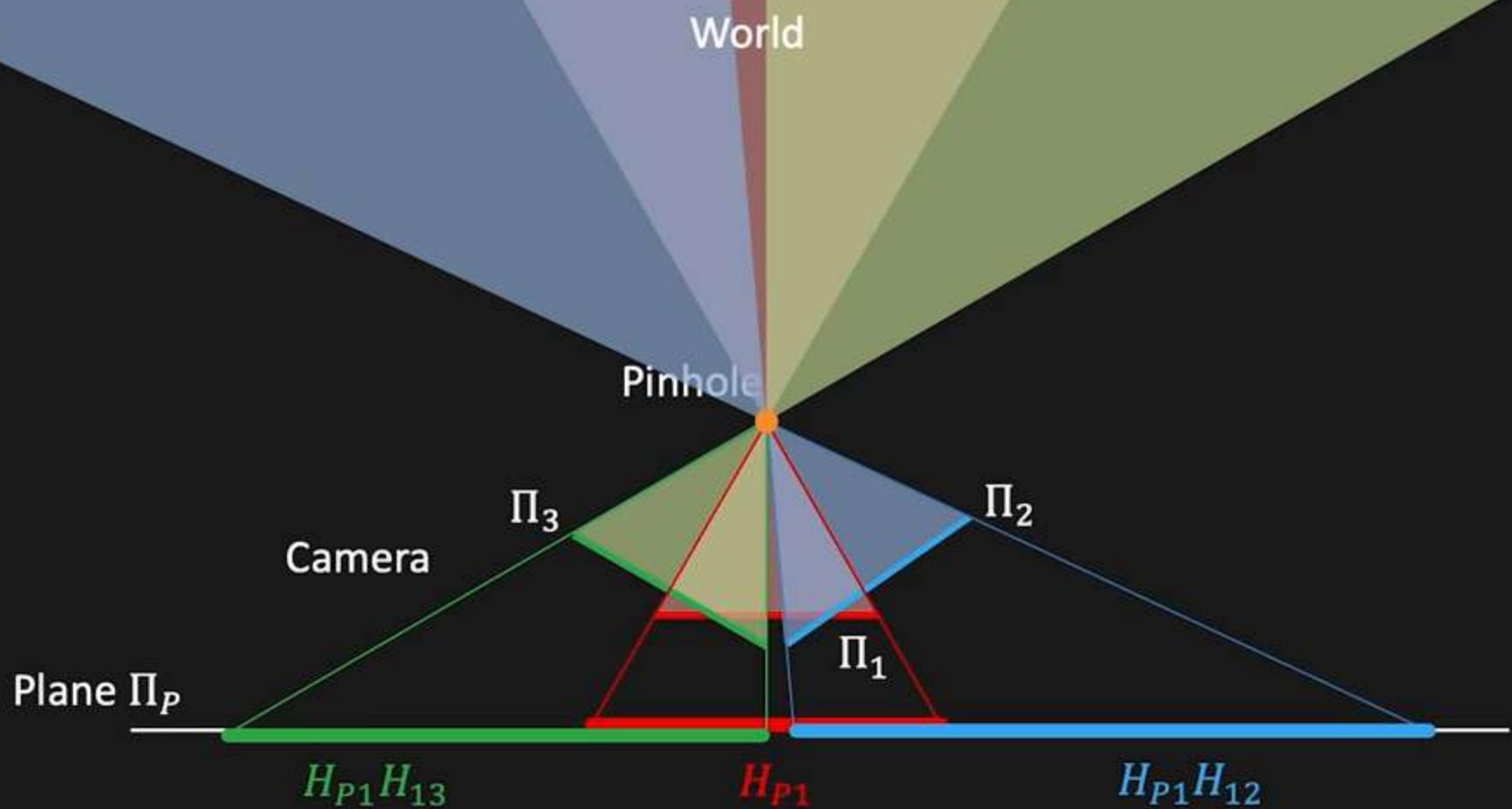
# Homography Composition



# Homography Composition



# Homography Composition



Useful in stitching planar panoramas

# Computing Homography



Image 1

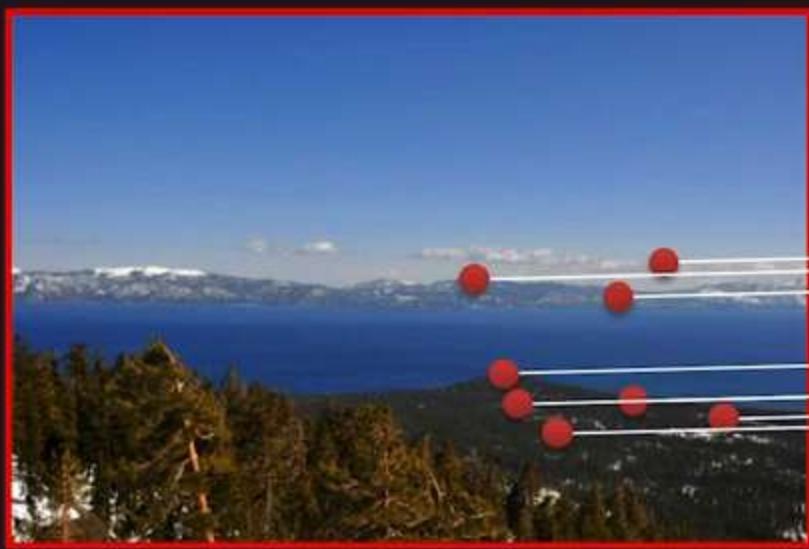


Image 2

Given a set of matching features/points between images 1 and 2, find the **homography  $H$**  that best “agrees” with the matches.

The scene points should lie on a plane, or be distant (plane at infinity), or imaged from the same point.

# Computing Homography



Source Image



Destination Image

$$\begin{bmatrix} x_d \\ y_d \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{x}_d \\ \tilde{y}_d \\ \tilde{z}_d \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_s \\ y_s \\ 1 \end{bmatrix}$$

How many unknowns? 9 ...But 8 degrees of freedom

How many minimum pairs of matching points? 4

# Computing Homography

For a given pair  $i$  of corresponding points:

$$x_d^{(i)} = \frac{\tilde{x}_d^{(i)}}{\tilde{z}_d^{(i)}} = \frac{h_{11}x_s^{(i)} + h_{12}y_s^{(i)} + h_{13}}{h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}}$$



$$y_d^{(i)} = \frac{\tilde{y}_d^{(i)}}{\tilde{z}_d^{(i)}} = \frac{h_{21}x_s^{(i)} + h_{22}y_s^{(i)} + h_{23}}{h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}}$$

Rearranging the terms:

$$x_d^{(i)} (h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}) = h_{11}x_s^{(i)} + h_{12}y_s^{(i)} + h_{13}$$



$$y_d^{(i)} (h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}) = h_{21}x_s^{(i)} + h_{22}y_s^{(i)} + h_{23}$$

# Computing Homography

$$x_d^{(i)} (h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}) = h_{11}x_s^{(i)} + h_{12}y_s^{(i)} + h_{13}$$

$$y_d^{(i)} (h_{31}x_s^{(i)} + h_{32}y_s^{(i)} + h_{33}) = h_{21}x_s^{(i)} + h_{22}y_s^{(i)} + h_{23}$$

Rearranging the terms and writing as linear equation:

$$\begin{bmatrix} x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}x_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

(Known) **h**  
(Unknown)

# Computing Homography

Combining the equations for all corresponding points:

$$\begin{bmatrix} x_s^{(1)} & y_s^{(1)} & 1 & 0 & 0 & 0 & -x_d^{(1)}x_s^{(1)} & -x_d^{(1)}y_s^{(1)} & -x_d^{(1)} \\ 0 & 0 & 0 & x_s^{(1)} & y_s^{(1)} & 1 & -y_d^{(1)}x_s^{(1)} & -y_d^{(1)}y_s^{(1)} & -y_d^{(1)} \\ & & & & & \vdots & & & \\ x_s^{(i)} & y_s^{(i)} & 1 & 0 & 0 & 0 & -x_d^{(i)}x_s^{(i)} & -x_d^{(i)}y_s^{(i)} & -x_d^{(i)} \\ 0 & 0 & 0 & x_s^{(i)} & y_s^{(i)} & 1 & -y_d^{(i)}x_s^{(i)} & -y_d^{(i)}y_s^{(i)} & -y_d^{(i)} \\ & & & & & \vdots & & & \\ x_s^{(n)} & y_s^{(n)} & 1 & 0 & 0 & 0 & -x_d^{(n)}x_s^{(n)} & -x_d^{(n)}y_s^{(n)} & -x_d^{(n)} \\ 0 & 0 & 0 & x_s^{(n)} & y_s^{(n)} & 1 & -y_d^{(n)}x_s^{(n)} & -y_d^{(n)}y_s^{(n)} & -y_d^{(n)} \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$$

$A$   
(Known)       $\mathbf{h}$   
(Unknown)

Solve for  $\mathbf{h}$ :  $A \mathbf{h} = \mathbf{0}$  such that  $\|\mathbf{h}\|^2 = 1$

# Constrained Least Squares

Solve for  $\mathbf{h}$ :  $A \mathbf{h} = \mathbf{0}$  such that  $\|\mathbf{h}\|^2 = 1$

Define least squares problem:

$$\min_{\mathbf{h}} \|A\mathbf{h}\|^2 \text{ such that } \|\mathbf{h}\|^2 = 1$$

We know that:

$$\|A\mathbf{h}\|^2 = (A\mathbf{h})^T(A\mathbf{h}) = \mathbf{h}^T A^T A \mathbf{h} \quad \text{and} \quad \|\mathbf{h}\|^2 = \mathbf{h}^T \mathbf{h} = 1$$

$$\min_{\mathbf{h}} (\mathbf{h}^T A^T A \mathbf{h}) \text{ such that } \mathbf{h}^T \mathbf{h} = 1$$

# Constrained Least Squares

---

$$\min_{\mathbf{h}} (\mathbf{h}^T A^T A \mathbf{h}) \text{ such that } \mathbf{h}^T \mathbf{h} = 1$$

Define Loss function  $L(\mathbf{h}, \lambda)$ :

$$L(\mathbf{h}, \lambda) = \mathbf{h}^T A^T A \mathbf{h} - \lambda(\mathbf{h}^T \mathbf{h} - 1)$$

Taking derivatives of  $L(\mathbf{h}, \lambda)$  w.r.t  $\mathbf{h}$ :  $2A^T A \mathbf{h} - 2\lambda \mathbf{h} = \mathbf{0}$

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

Eigenvalue Problem

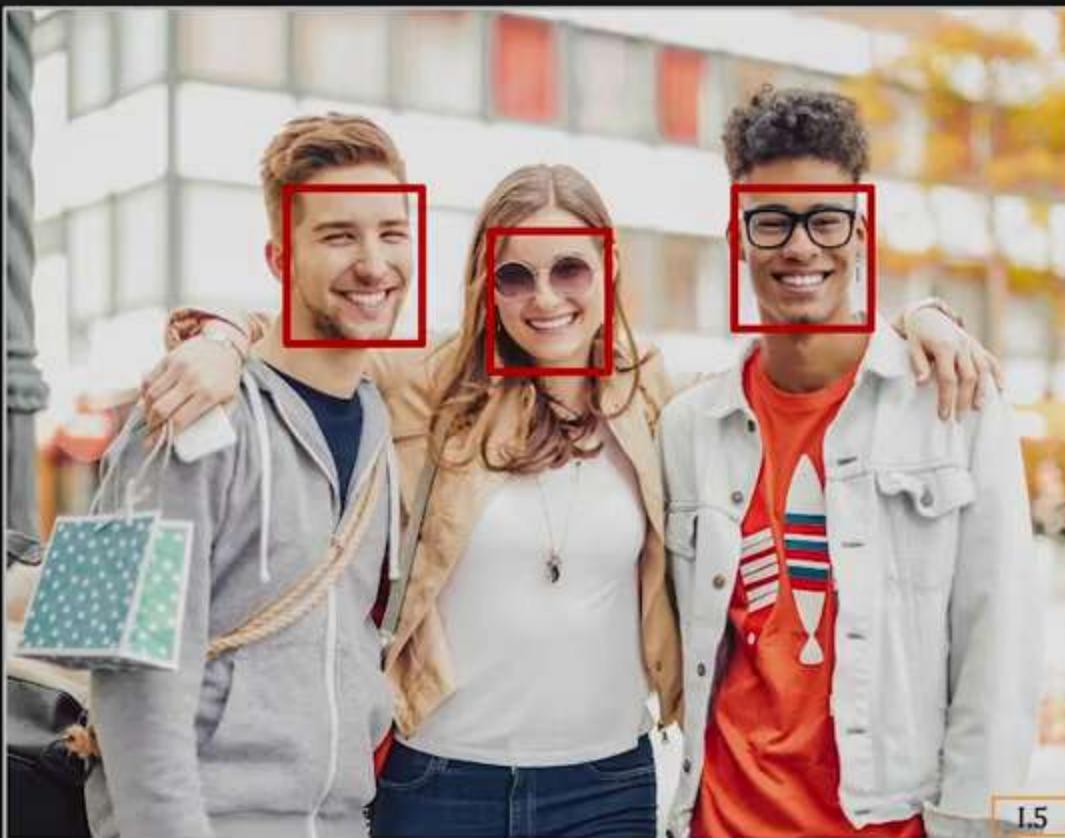
Eigenvector  $\mathbf{h}$  with smallest eigenvalue  $\lambda$  of matrix  $A^T A$  minimizes the loss function  $L(\mathbf{h})$ .

Matlab: `eig(A' * A)` returns eigenvalues and vectors of  $A^T A$

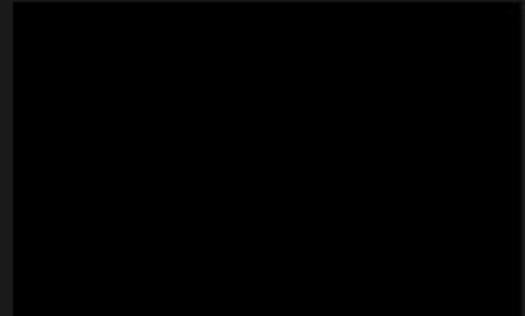
# What is Face Detection?

---

Locate human faces in images



1.5



# Face Detection

---

Locate human faces in images.

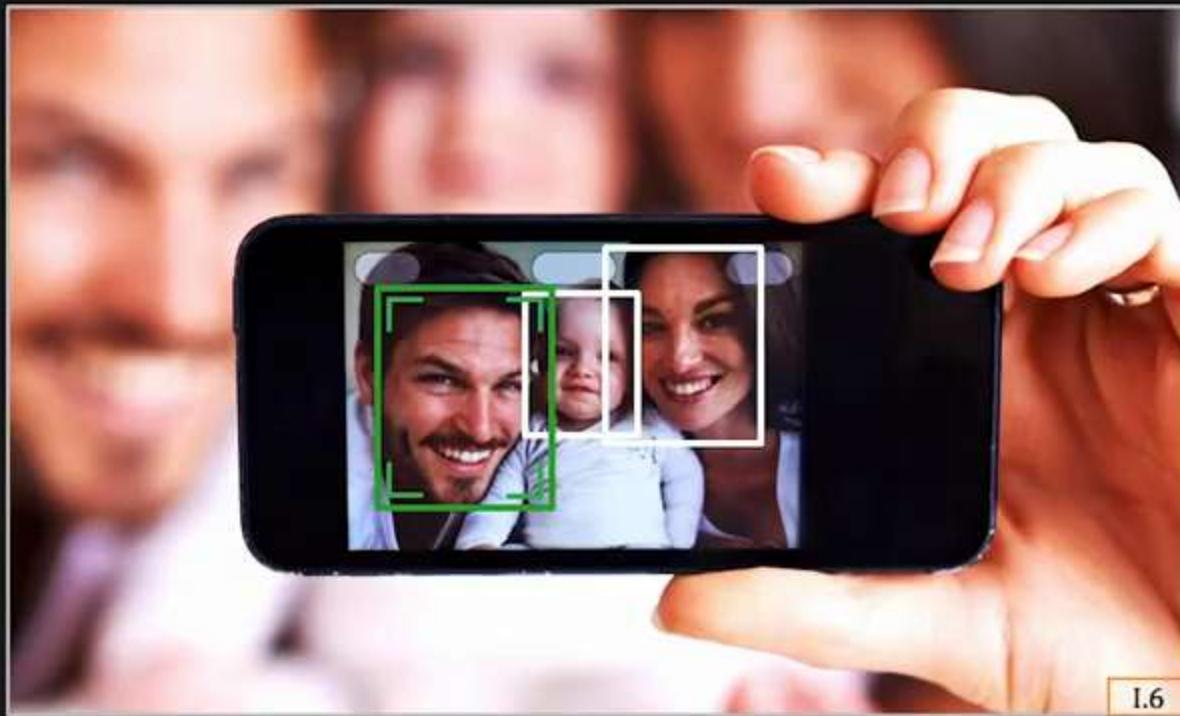
## Topics:

- (1) Uses of Face Detection
- (2) Haar Features for Face Detection
- (3) Integral Image
- (4) Nearest Neighbor Classifier
- (5) Support Vector Machine



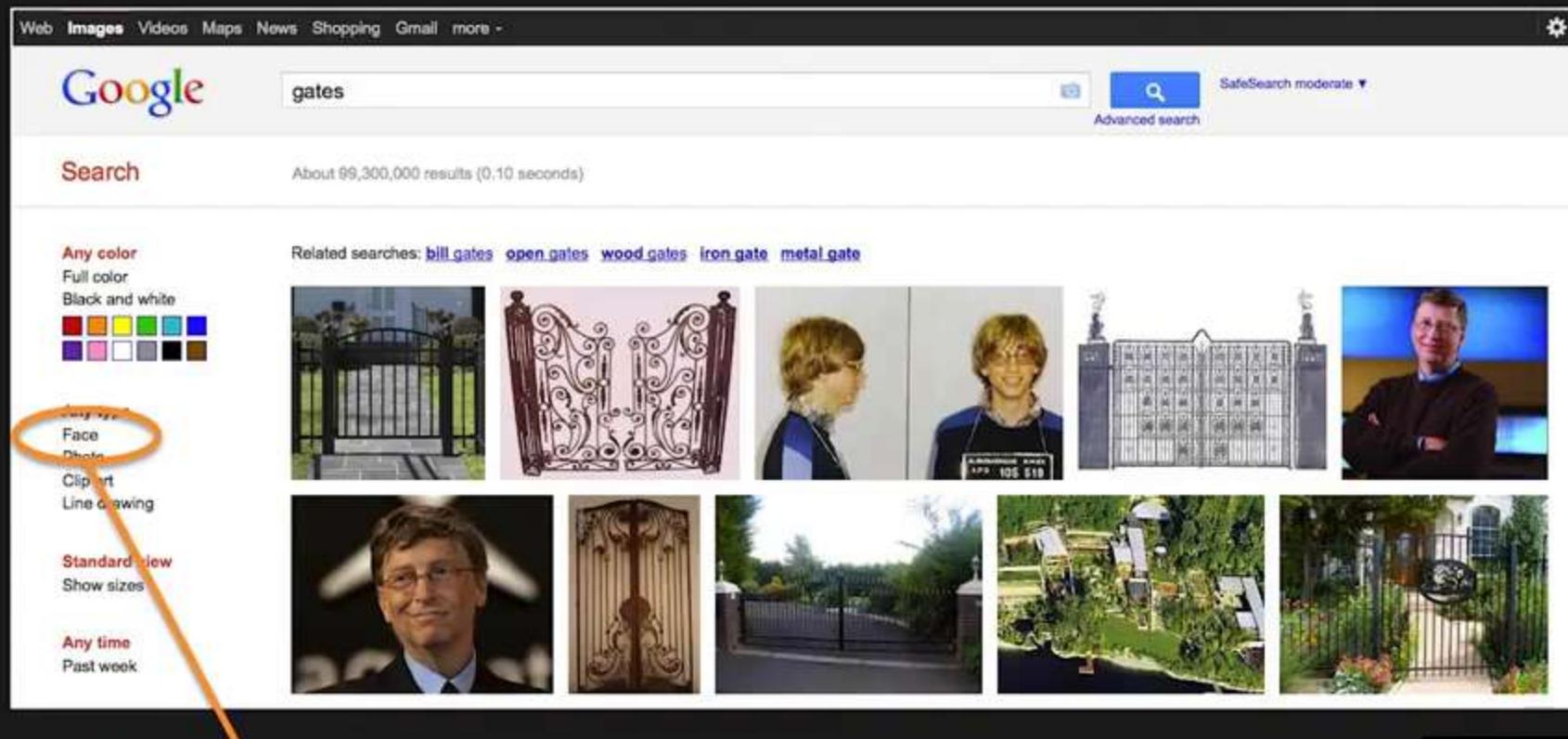
# Where is Face Detection Used?

---



Automatic Selection of Camera Settings  
(Autofocus, Exposure, Color Balance, etc.)

# Where is Face Detection Used?



Face Detection

Finding People using Search Engines

# Where is Face Detection Used?

---



L2

Intelligent Marketing

# Where is Face Detection Used?

---



Biometrics, Surveillance, Monitoring

# Face Detection in Computers

---

Slide windows of different sizes across image.  
At each location match window to face model.



# Face Detection in Computers

---

Slide windows of different sizes across image.  
At each location match window to face model.

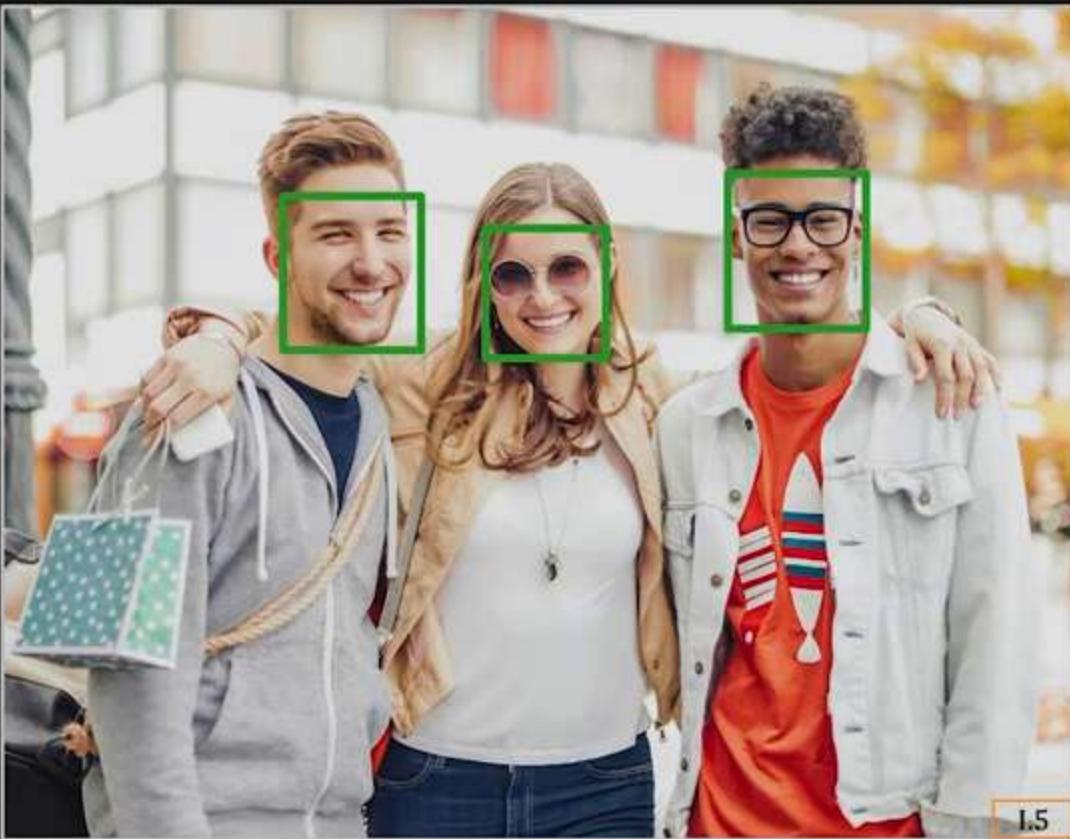


l5

# Face Detection in Computers

---

Slide windows of different sizes across image.  
At each location match window to face model.



# Face Detection Framework

For each window:



Features: Which features represent faces well?

Classifier: How to construct a face model and efficiently classify features as face or not?

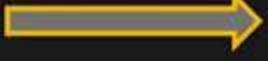
# What are Good Features?

---

Interest Points (Edges, Corners, SIFT)?

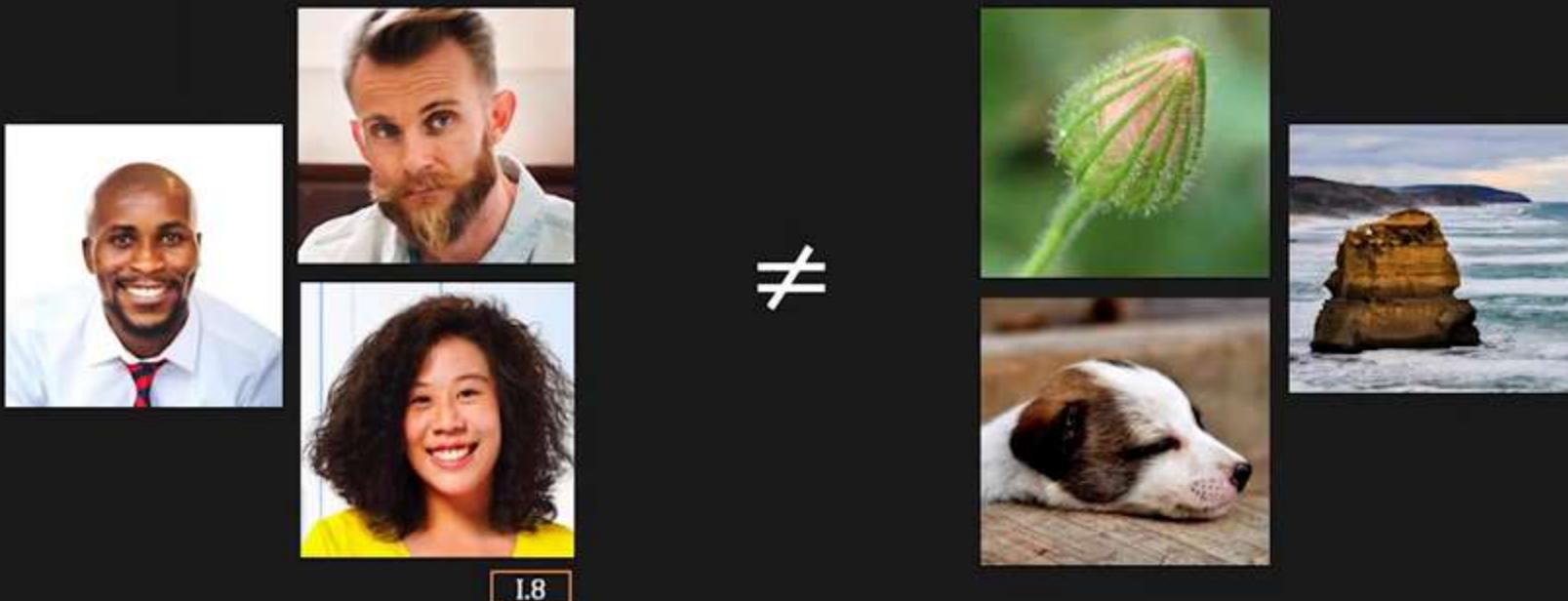


Facial Components (Templates)?



# Characteristics of Good Features

## Discriminate Face/Non-Face



Extremely Fast to Compute

Need to evaluate millions of windows in an image

## Haar Features

## Set of Correlation Responses to Haar Filters



## Input Image

$$\otimes \begin{bmatrix} H_A \\ H_B \\ H_C \\ H_D \\ \vdots \end{bmatrix} = \begin{bmatrix} V_A[i,j] \\ V_B[i,j] \\ V_C[i,j] \\ V_D[i,j] \\ \vdots \end{bmatrix}$$

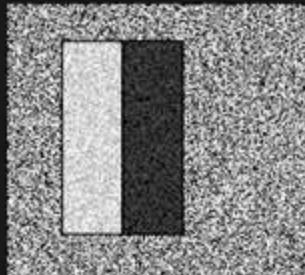
**Haar Filters**      **Haar Features**

# Discriminative Ability of Haar Feature

---



$$V_A = 64$$



$$V_A \approx 0$$



$$V_A = 16$$

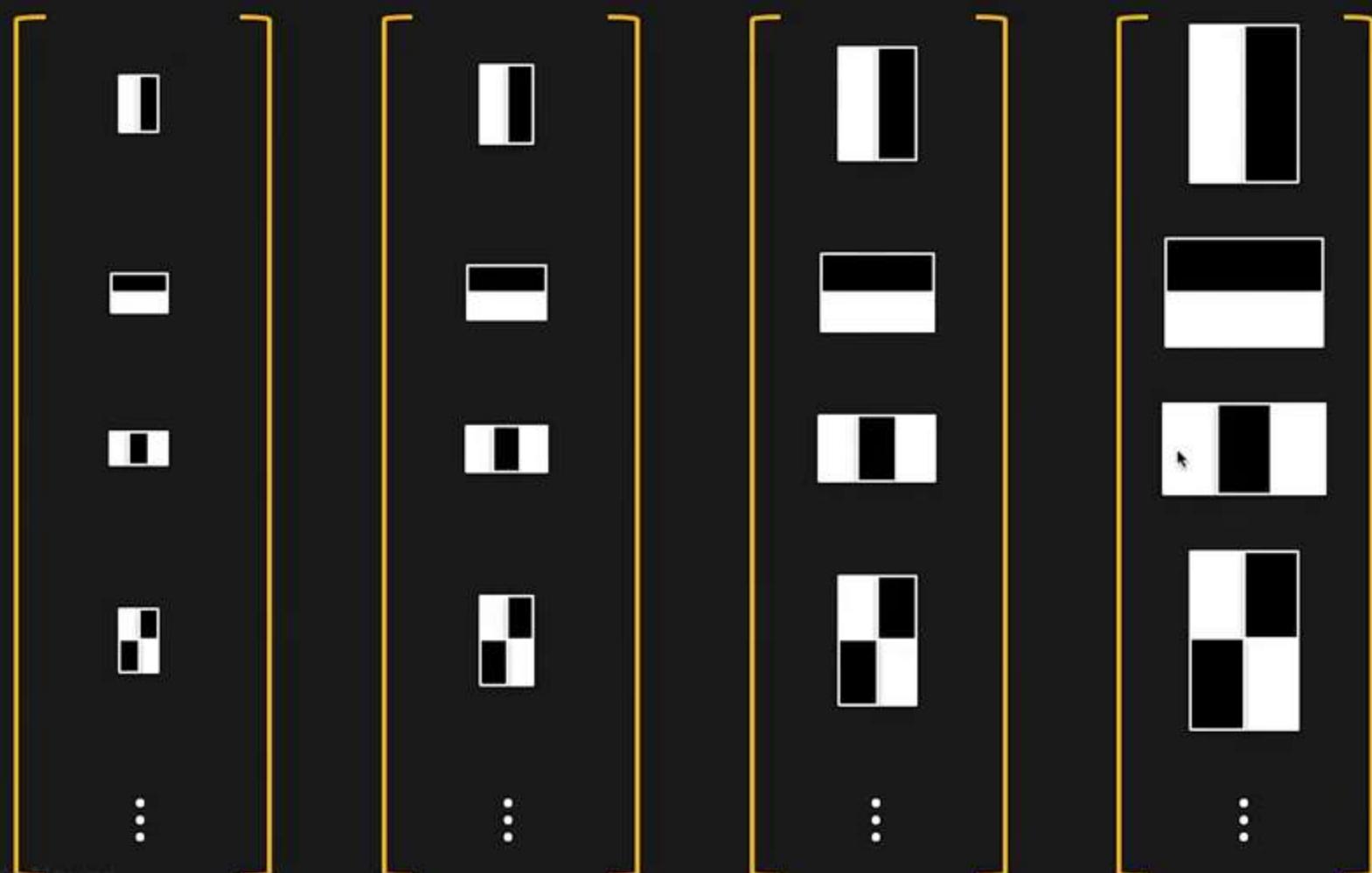


$$V_A = -127$$

Haar Features are Sensitive to Directionality of Patterns

# Detecting Faces of Different Size

Compute Haar Features at different scales to detect faces of different sizes.



# Computing A Haar Feature



$$\otimes \quad \begin{matrix} & \\ & \end{matrix} \quad H_A$$

White = 1, Black = -1

Response to Filter  $H_A$  at location  $(i, j)$ :

$$V_A[i, j] = \sum_m \sum_n I[m - i, n - j] H_A[m, n]$$

$$V_A[i, j] = \sum \text{(pixel intensities in white area)} - \sum \text{(pixels intensities in black area)}$$

# Haar Feature: Computation Cost



$$\text{Value} = \sum(\text{pixel intensities in white}) - \sum(\text{pixel intensities in black})$$

Computation cost =  $(N \times M - 1)$  additions per pixel per filter per scan

Can We Do Better?

# Integral Image

---

A table that holds the sum of all pixel values to the left and top of a given pixel, inclusive.

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image *I*

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image *II*

# Integral Image

---

A table that holds the sum of all pixel values to the left and top of a given pixel, inclusive.

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image *I*

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image *II*



# Summation Within a Rectangle

Fast summations of arbitrary rectangles using integral images

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image  $I$

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image  $II$

$$\text{Sum} = II_P + \dots$$

$$= 3490 + \dots$$

# Summation Within a Rectangle

Fast summations of arbitrary rectangles using integral images

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image  $I$

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image  $II$

$$\begin{aligned} \text{Sum} &= II_P - II_Q - II_S + \dots \\ &= 3490 - 1137 - 1249 + \dots \end{aligned}$$

# Summation Within a Rectangle

Fast summations of arbitrary rectangles using integral images

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

Image *I*

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image *II*

$$\begin{aligned} \text{Sum} &= II_P - II_Q - II_S + II_R \\ &= 3490 - 1137 - 1249 + 417 = 1521 \end{aligned}$$

Computational Cost: Only 3 additions

# Haar Response using Integral Image

---

98	110	121	125	122	129
99	110	120	116	116	129
97	109	124	111	123	134
98	112	132	108	123	133
97	113	147	108	125	142
95	111	168	122	130	137
96	104	172	130	126	130

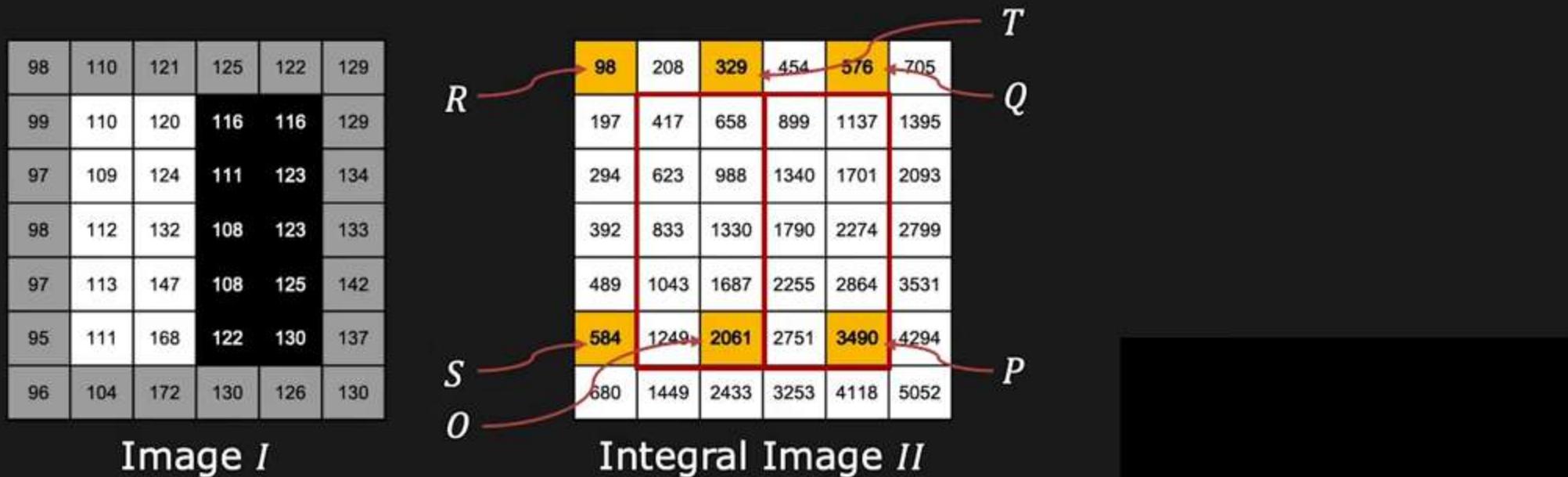
Image I

98	208	329	454	576	705
197	417	658	899	1137	1395
294	623	988	1340	1701	2093
392	833	1330	1790	2274	2799
489	1043	1687	2255	2864	3531
584	1249	2061	2751	3490	4294
680	1449	2433	3253	4118	5052

Integral Image II

$$V_A = \sum(\text{pixels in white}) - \sum(\text{pixels in black})$$

# Haar Response using Integral Image

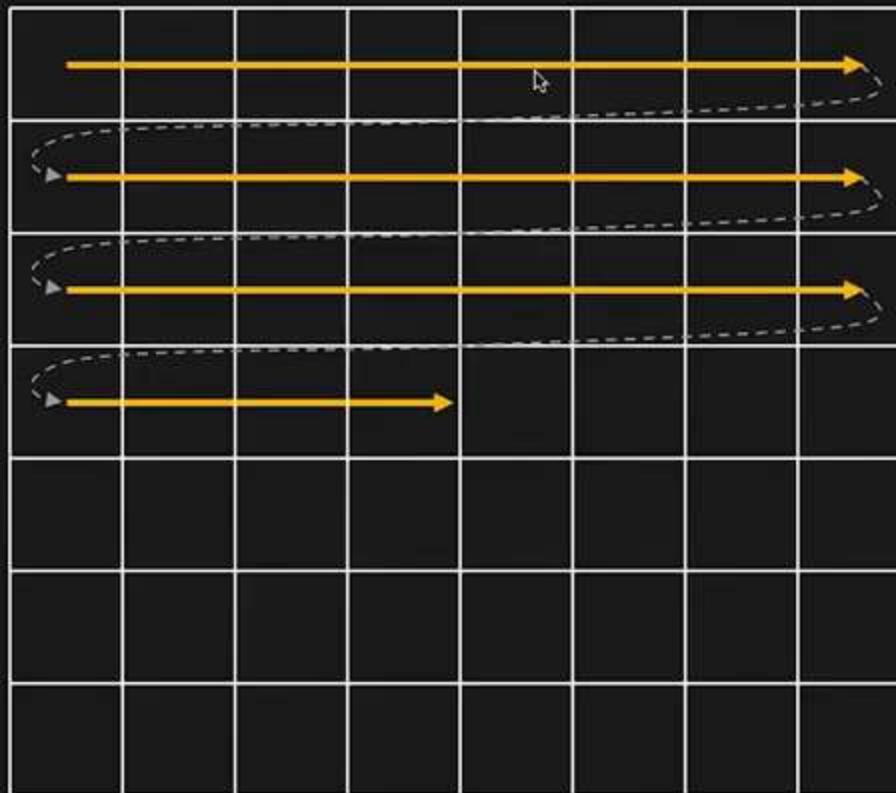


$$\begin{aligned}V_A &= \sum(\text{pixel intensities in white}) - \sum(\text{pixel intensities in black}) \\&= (II_O - II_T + II_R - II_S) - (II_P - II_Q + II_T - II_O) \\&= (2061 - 329 + 98 - 584) - (3490 - 576 + 329 - 2061) = 64\end{aligned}$$

Computational Cost: Only 7 additions

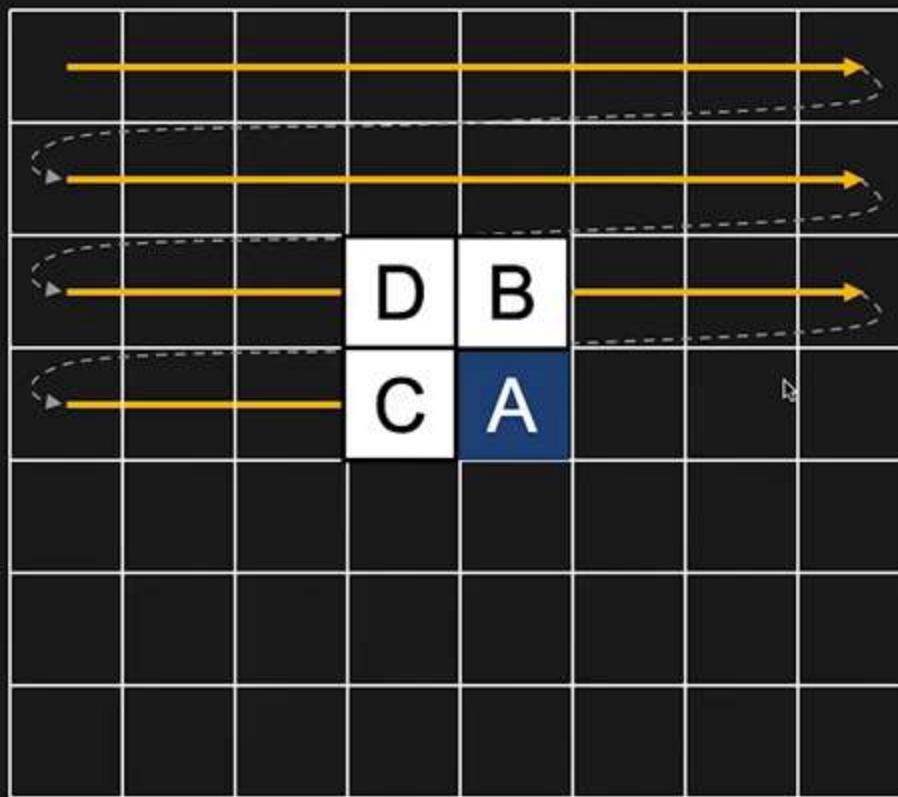
# Computing Integral Image

---



Raster  
Scanning

# Computing Integral Image



Raster  
Scanning

Let  $I_A$  and  $II_A$  be the values of Image and Integral Image, respectively, at pixel A.

$$II_A = II_B + II_C - II_D + I_A$$

# Haar Features Using Integral Images

Integral image needs to be computed once per test image.  
Allows fast computations of Haar features.



Input Image

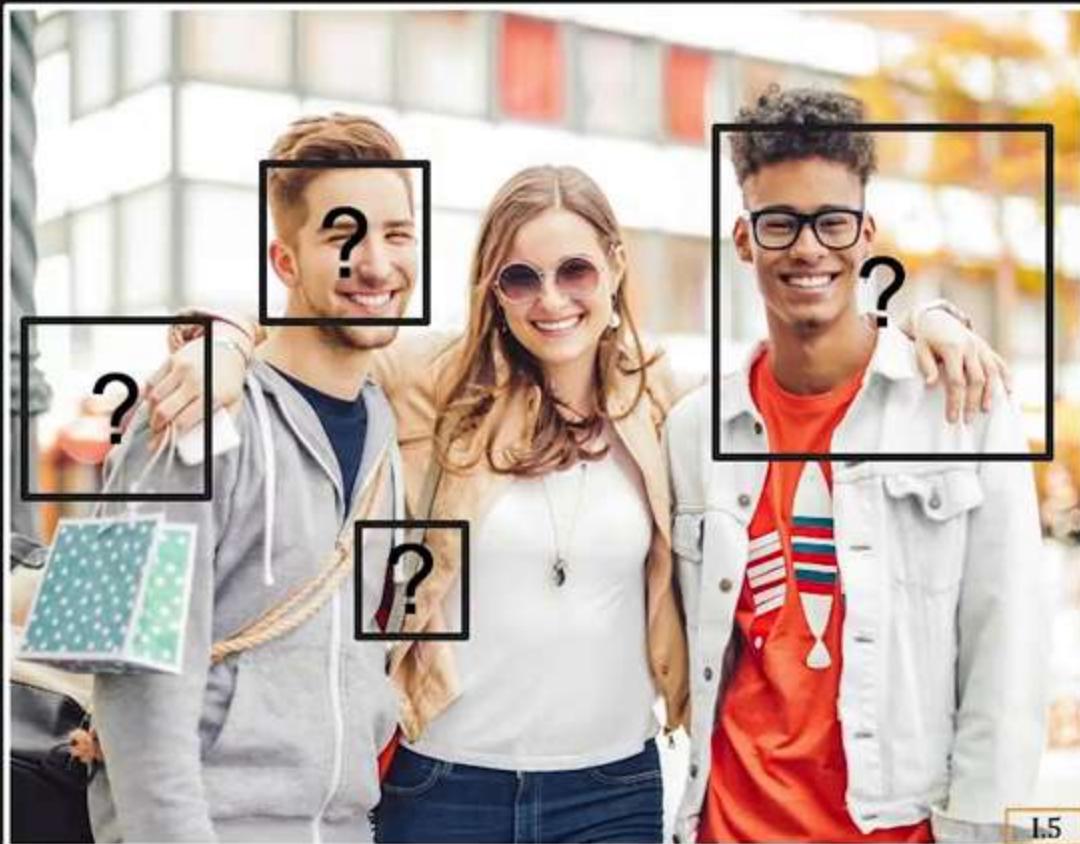
$$\otimes \begin{bmatrix} H_A \\ H_B \\ H_C \\ H_D \\ \vdots \end{bmatrix} = \begin{bmatrix} V_A[i,j] \\ V_B[i,j] \\ V_C[i,j] \\ V_D[i,j] \\ \vdots \end{bmatrix}$$

The diagram illustrates the computation of Haar features using integral images. On the left, an input image of three people is shown. To its right is a vertical column of four Haar filters labeled  $H_A$ ,  $H_B$ ,  $H_C$ , and  $H_D$ . Below these filters is a symbol representing element-wise multiplication ( $\otimes$ ). To the right of the multiplication symbol is a vertical column of four corresponding Haar feature vectors labeled  $V_A[i,j]$ ,  $V_B[i,j]$ ,  $V_C[i,j]$ , and  $V_D[i,j]$ . Ellipses at the bottom indicate that there are more filters and features. The entire equation is preceded by an equals sign.

# Classifier for Face Detection

---

Given the features for a window, how to decide whether it contains a face or not?



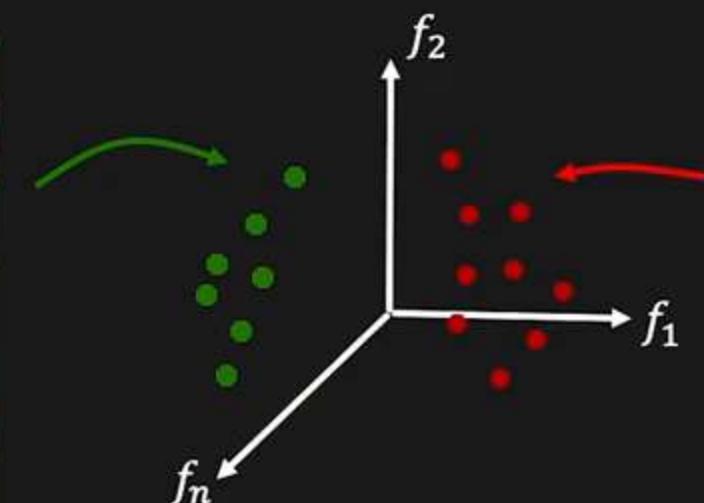
1.5

# Feature Space

Haar features  $\mathbf{f}$  (a vector) at a pixel  
is a point in an n-D space,  $\mathbf{f} \in \mathbb{R}^n$



Training Data  
of Face



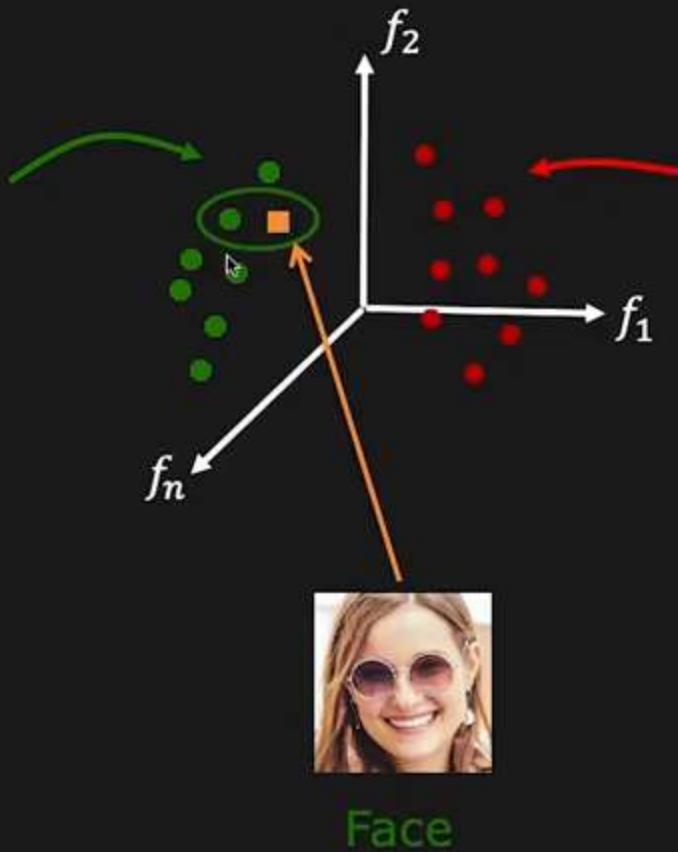
Training Data  
of Non-Face

# Nearest Neighbor Classifier

Find the nearest training sample using  $\mathcal{L}^2$  distance and assign its label.



Training Data  
of Face



Face



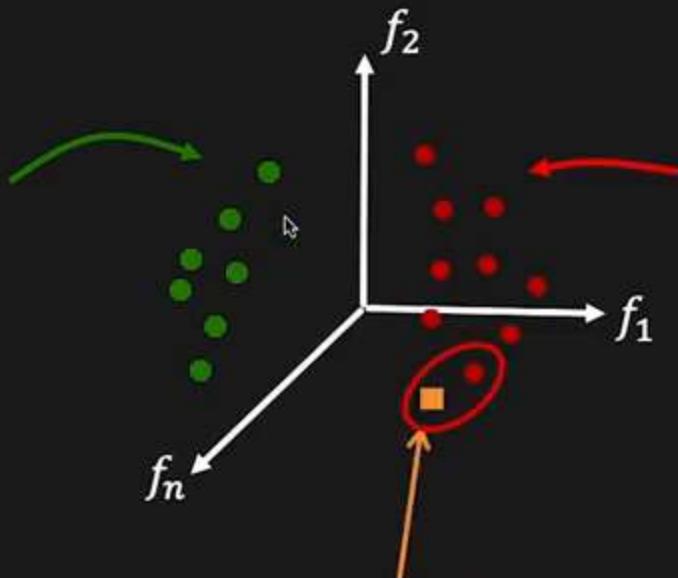
Training Data  
of Non-Face

# Nearest Neighbor Classifier

Find the nearest training sample using  $\mathcal{L}^2$  distance and assign its label.



Training Data  
of Face



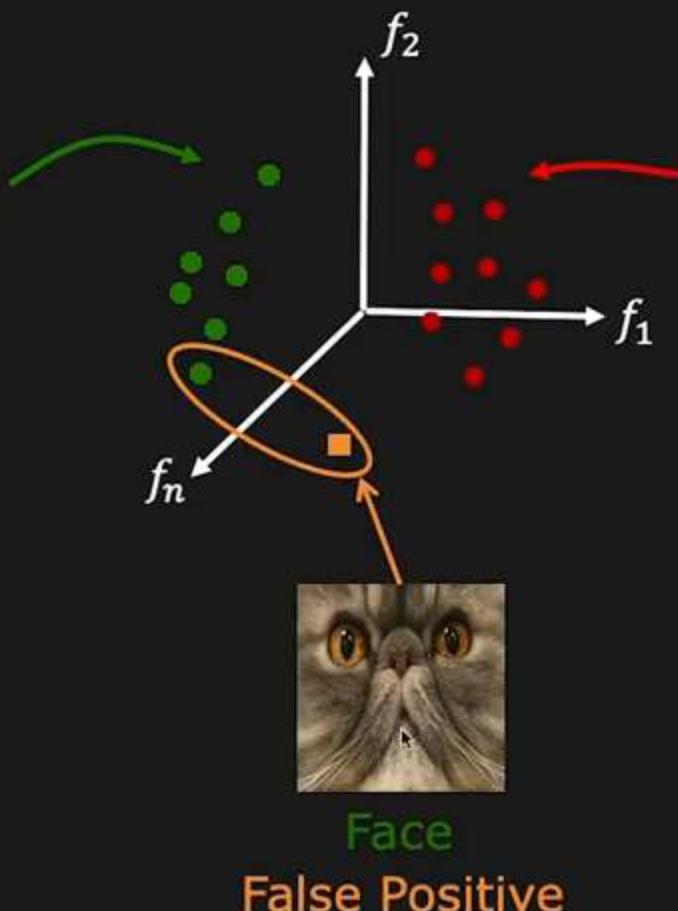
Training Data  
of Non-Face

# Nearest Neighbor Classifier

Find the nearest training sample using  $\mathcal{L}^2$  distance and assign its label.



Training Data  
of Face



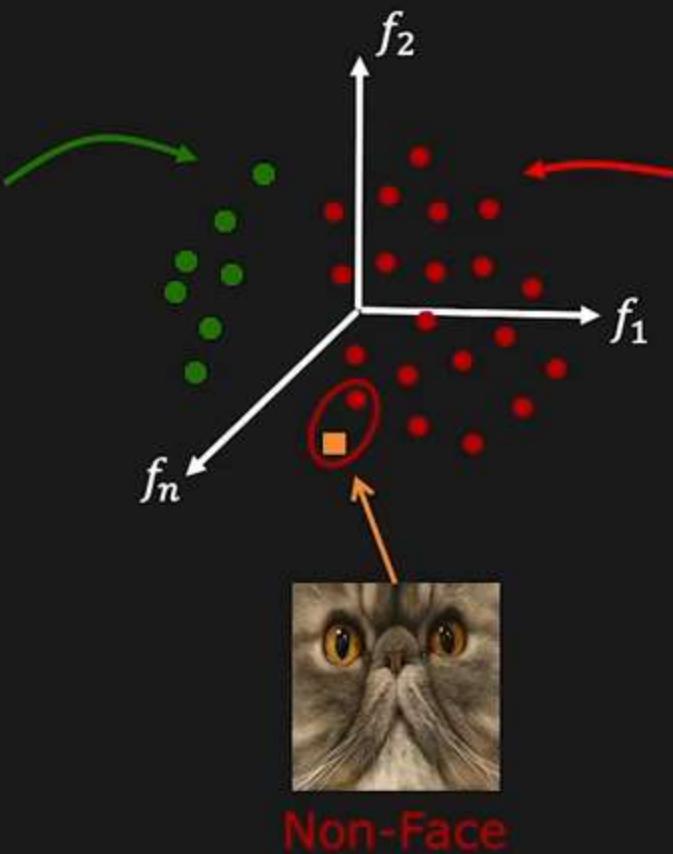
Training Data  
of Non-Face

# Nearest Neighbor Classifier

Larger the training set, more robust the NN classifier



Training Data  
of Face



Non-Face



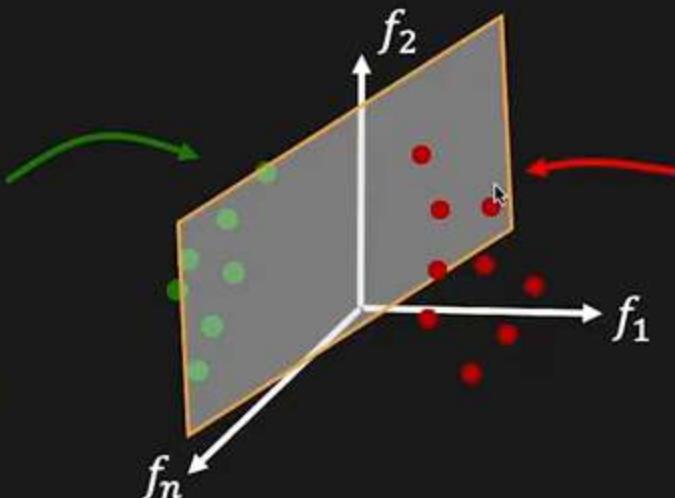
Training Data  
of Non-Face

# Decision Boundary

A simple decision boundary separating face and non-face classes will suffice.



Training Data  
of Face



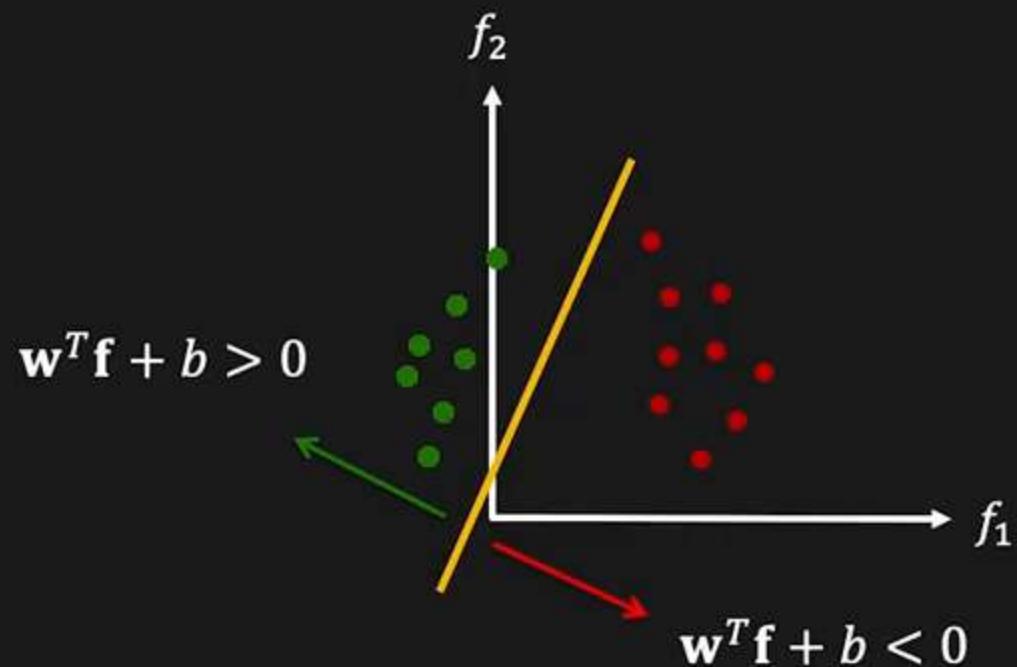
I.8



Training Data  
of Non-Face

# Linear Decision Boundaries

A Linear Decision Boundary in 2-D space is a 1-D Line



Equation of Line:

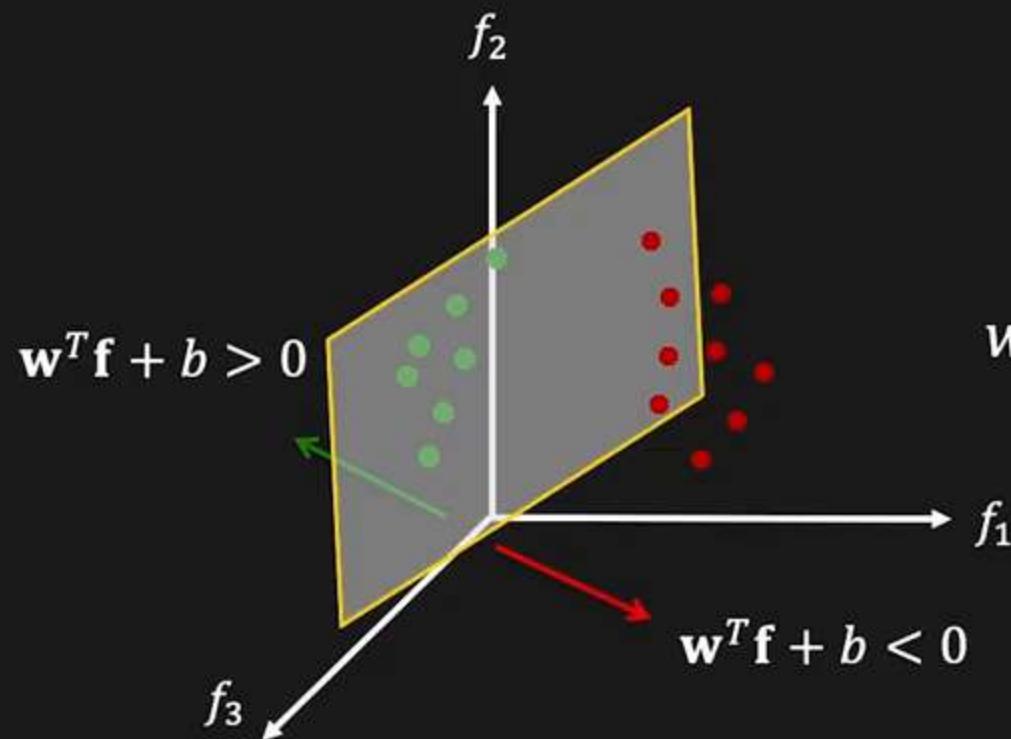
$$w_1 f_1 + w_2 f_2 + b = 0$$

$$\begin{bmatrix} w_1 & w_2 \end{bmatrix} \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} + b = 0$$

$$\boxed{w^T \mathbf{f} + b = 0}$$

# Linear Decision Boundaries

A Linear Decision Boundary in 3-D space is a **2-D Plane**



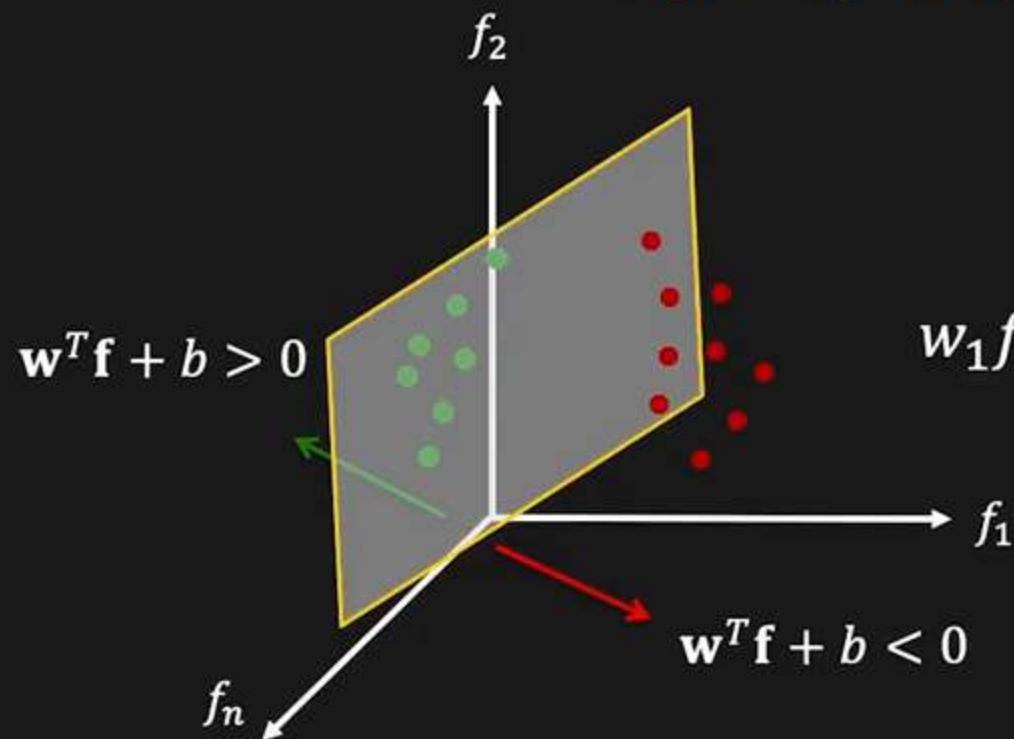
Equation of Plane:

$$w_1 f_1 + w_2 f_2 + w_3 f_3 + b = 0$$

$$\mathbf{w}^T \mathbf{f} + b = 0$$

# Linear Decision Boundaries

A Linear Decision Boundary in  $n$ -D space  
is a  $(n - 1)$ -D Hyperplane



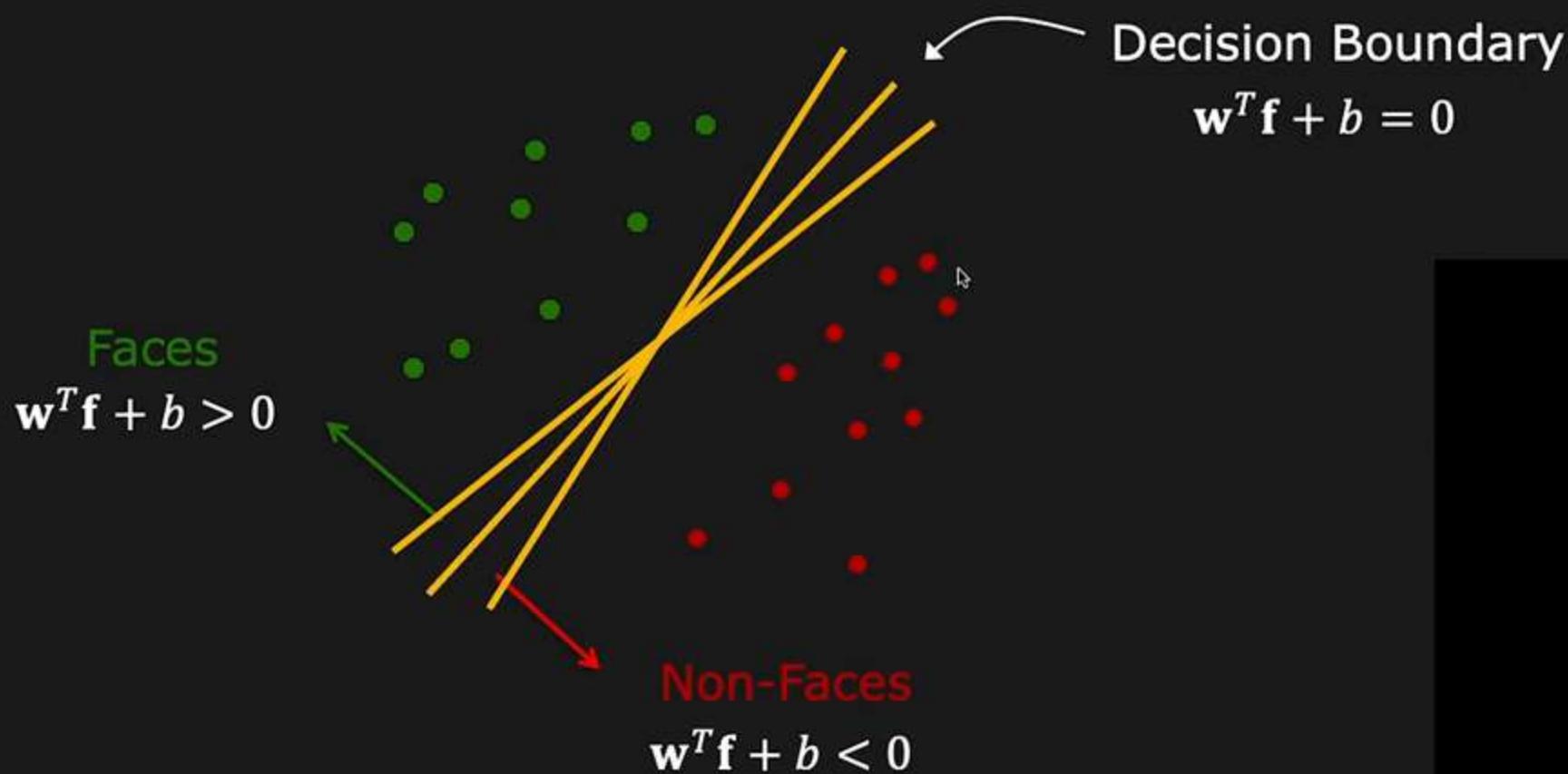
Equation of Hyperplane:

$$\mathbf{w}_1 f_1 + \mathbf{w}_2 f_2 + \cdots + \mathbf{w}_n f_n + b = 0$$

$$\mathbf{w}^T \mathbf{f} + b = 0$$

# Decision Boundary ( $\mathbf{w}, b$ )

What is the **optimal** decision boundary?



# Evaluating a Decision Boundary

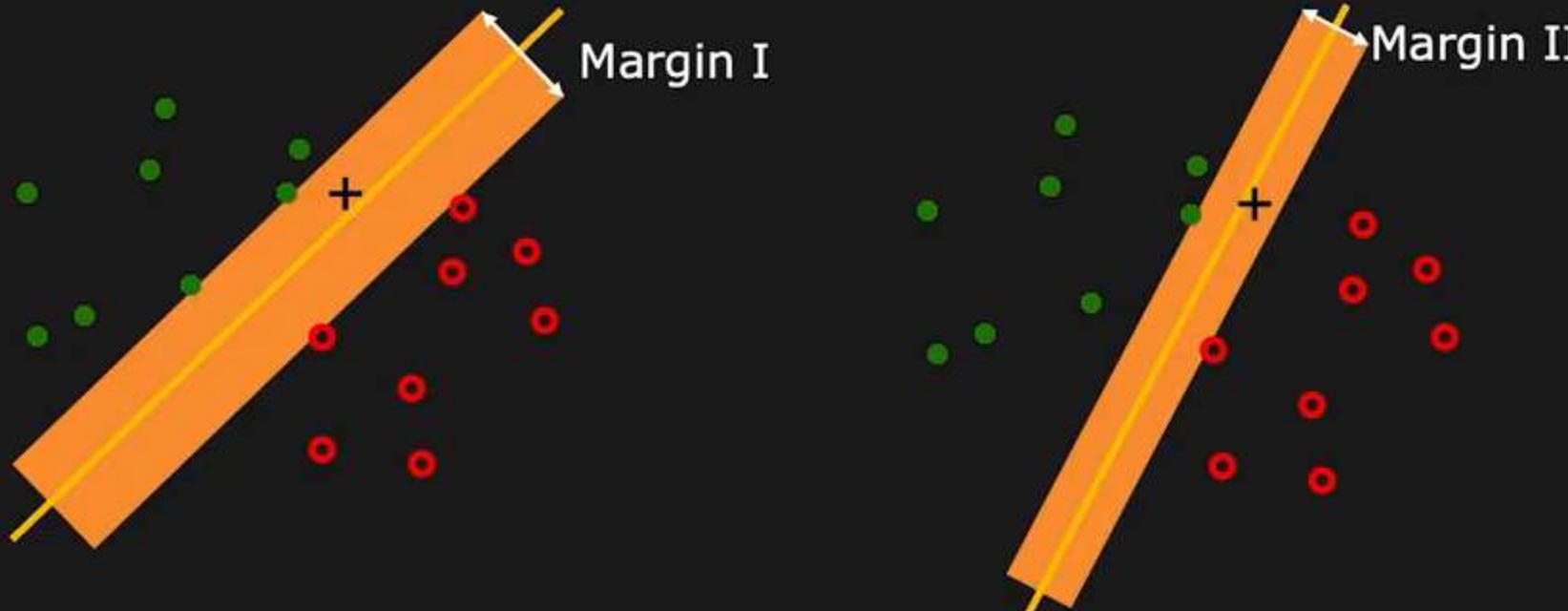
---



**Margin or Safe Zone:** The width that the boundary could be increased by, before hitting a feature point.

# Evaluating a Decision Boundary

---



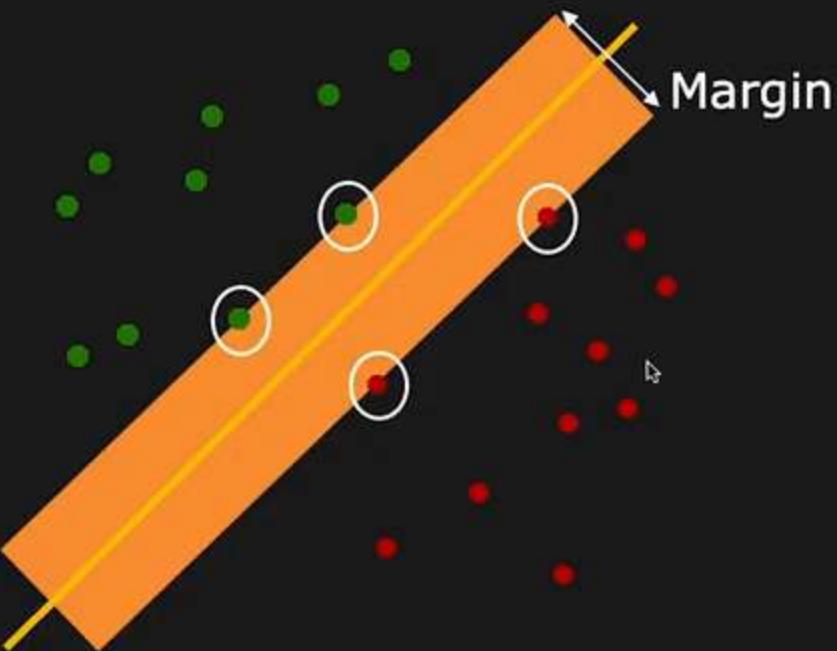
Decision I: Face

Decision II: Non-Face

Choose Decision Boundary with Maximum Margin!

# Support Vector Machine (SVM)

Classifier optimized to Maximize Margin



Support Vectors: Closest data samples to the boundary

Decision Boundary & Margin depend only on Support Vectors

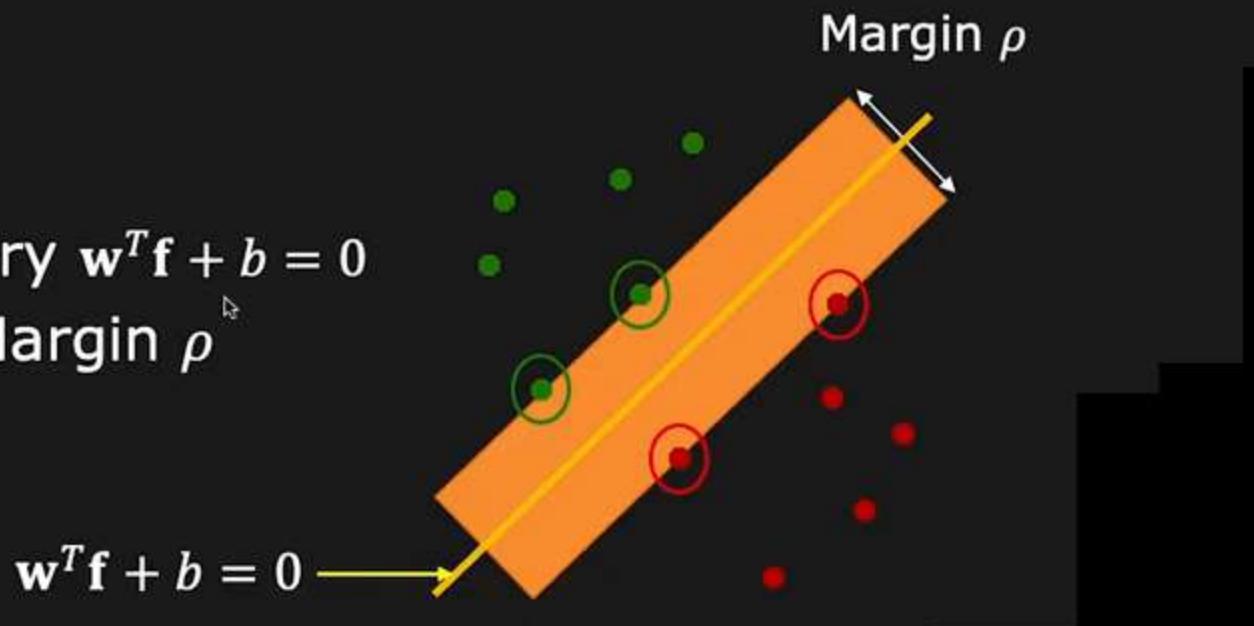
# Support Vector Machine (SVM)

Given:

- $k$  training images  $\{I_1, I_2, \dots, I_k\}$  and their Haar features  $\{\mathbf{f}_1, \mathbf{f}_2, \dots, \mathbf{f}_k\}$ .
- $k$  corresponding labels  $\{\lambda_1, \lambda_2, \dots, \lambda_k\}$ , where  $\lambda_j = +1$  if  $I_j$  is a face and  $\lambda_j = -1$  if  $I_j$  is not a face.

Find:

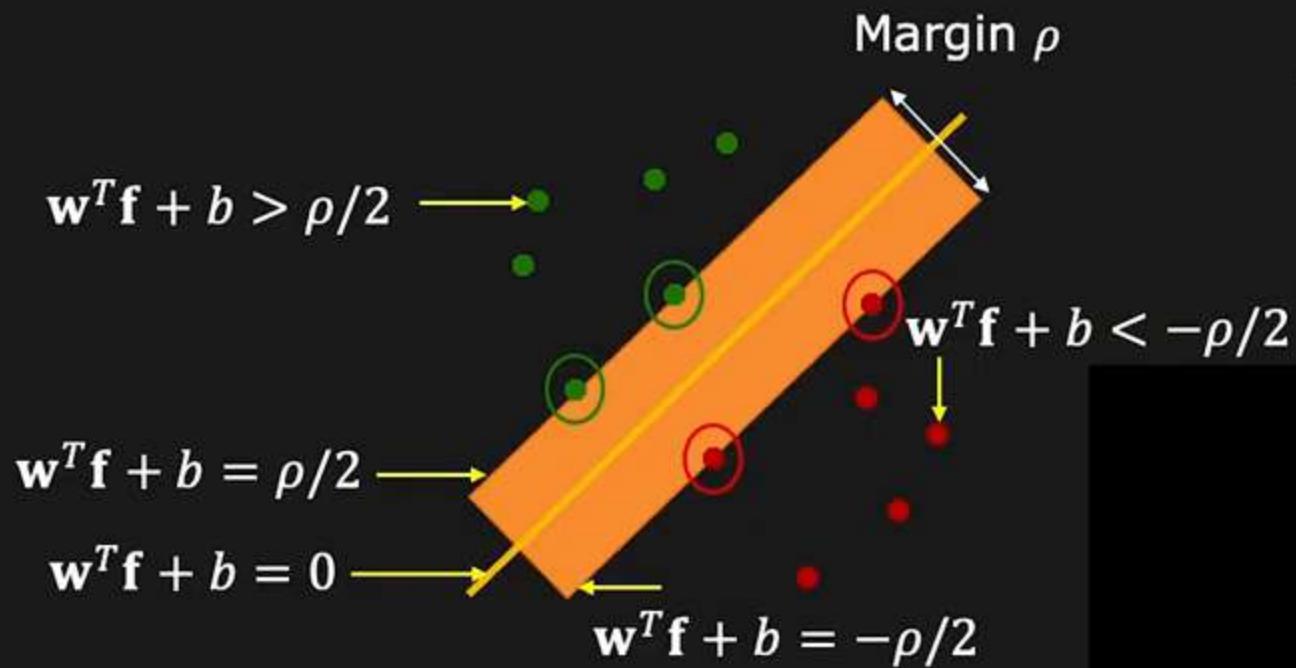
Decision Boundary  $\mathbf{w}^T \mathbf{f} + b = 0$   
with Maximum Margin  $\rho$



# Finding Decision Boundary ( $\mathbf{w}, b$ )

For each training sample  $(\mathbf{f}_i, \lambda_i)$ :

$$\left. \begin{array}{ll} \text{If } \lambda_i = +1: & \mathbf{w}^T \mathbf{f}_i + b \geq \rho/2 \\ \text{If } \lambda_i = -1: & \mathbf{w}^T \mathbf{f}_i + b \leq -\rho/2 \end{array} \right\} \quad \boxed{\lambda_i(\mathbf{w}^T \mathbf{f}_i + b) \geq \rho/2}$$



# Finding Decision Boundary ( $\mathbf{w}, b$ )

---

For each training sample  $(\mathbf{f}_i, \lambda_i)$ :

$$\left. \begin{array}{ll} \text{If } \lambda_i = +1: & \mathbf{w}^T \mathbf{f}_i + b \geq \rho/2 \\ \text{If } \lambda_i = -1: & \mathbf{w}^T \mathbf{f}_i + b \leq -\rho/2 \end{array} \right\} \quad \boxed{\lambda_i(\mathbf{w}^T \mathbf{f}_i + b) \geq \rho/2}$$

If  $\mathcal{S}$  is the set of support vectors,

Then for every support vector  $s \in \mathcal{S}$ : 
$$\boxed{\lambda_s(\mathbf{w}^T \mathbf{f}_s + b) = \rho/2}$$

Numerical methods exist to find  
 $\mathbf{w}, b$  and  $\mathcal{S}$  that maximize  $\rho$

MATLAB: `svmtrain`

# Classification using SVM

---

Given: Haar features  $\mathbf{f}$  for an image window and  
SVM parameters  $\mathbf{w}, b, \rho, \mathcal{S}$

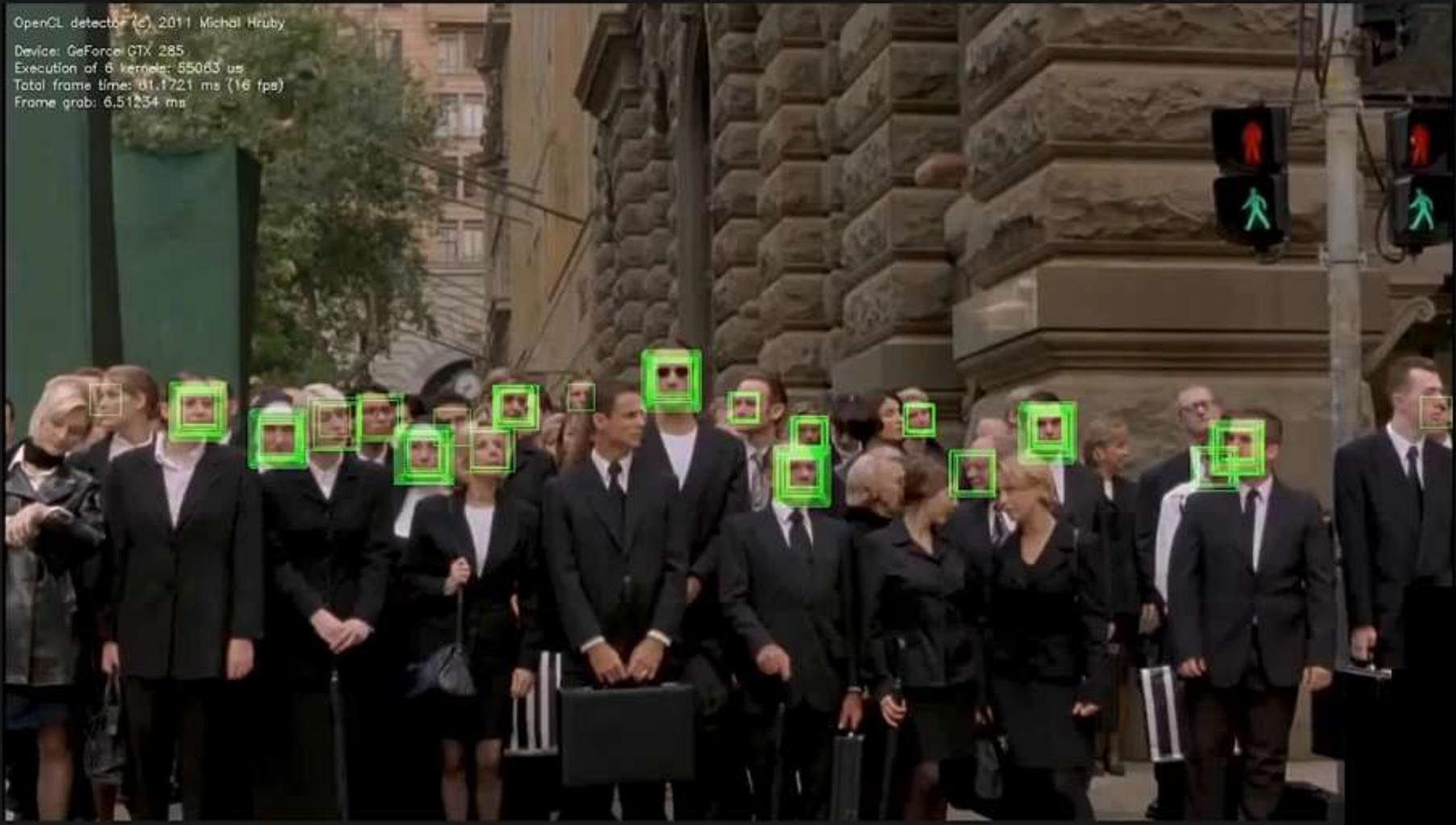
Classification:

Compute  $d = \mathbf{w}^T \mathbf{f} + b$

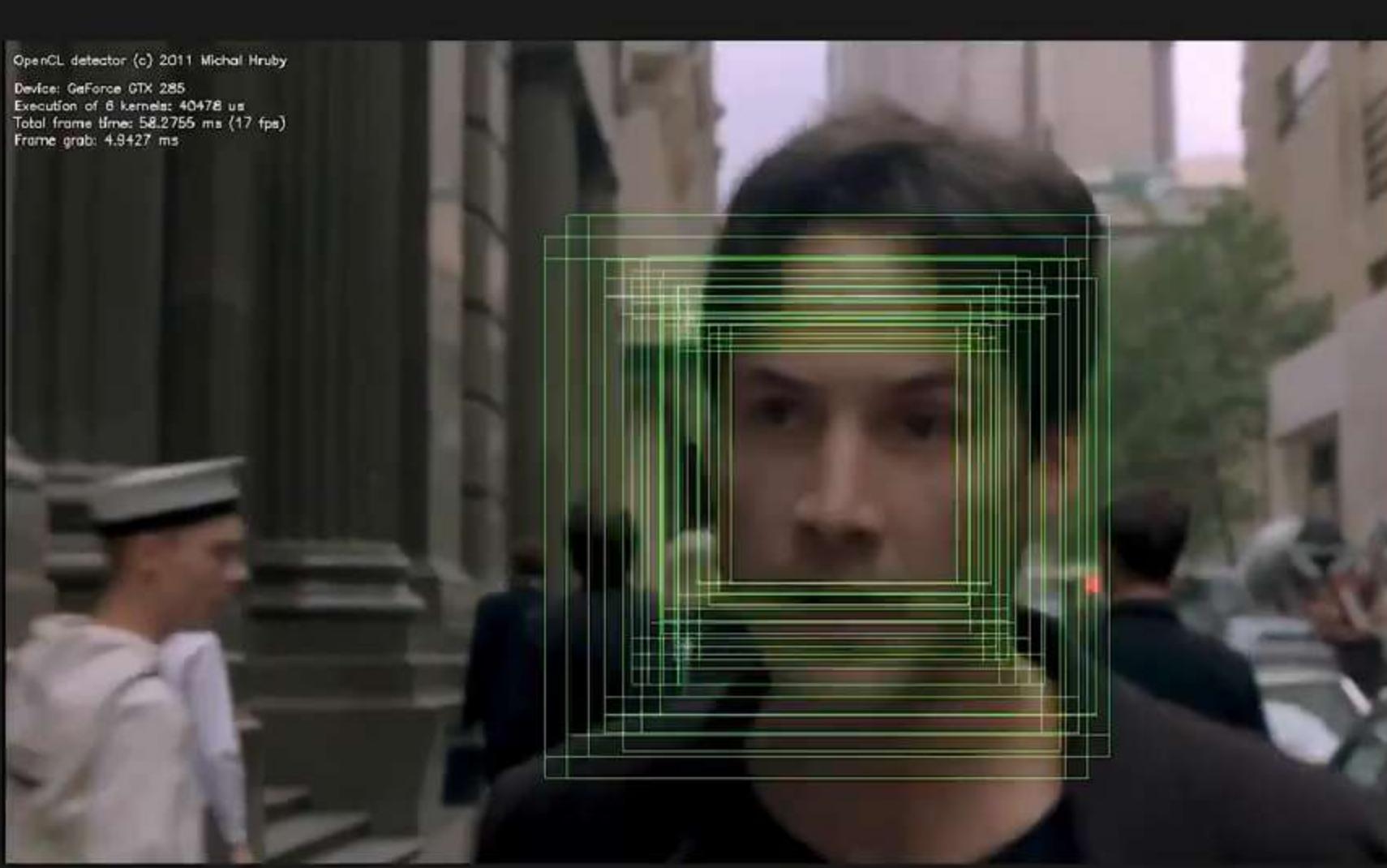
If: 
$$\begin{cases} d \geq \rho/2 & \text{Face} \\ d > 0 \text{ and } d < \rho/2 & \text{Probably Face} \\ d < 0 \text{ and } d > -\rho/2 & \text{Probably Not-Face} \\ d \leq -\rho/2 & \text{Not-Face} \end{cases}$$

# Face Detection Results

---



# Face Detection Results



## Remarks

---

- Current face detection systems are mature but not perfect.
- Frontal and side poses usually require different face models.
- Successful vision technology used in cameras, surveillance, biometrics, search.
- Performance continues to improve.

# Thank You

Kumar Bipin | linux.kbp@gmail.com