Computer Vision-IT416

Dinesh Naik

Department of Information Technology,
National Institute of Technology Karnataka, India

April 5, 2022

Boundary Detection

We need to find Object Boundaries from Edge Pixels.

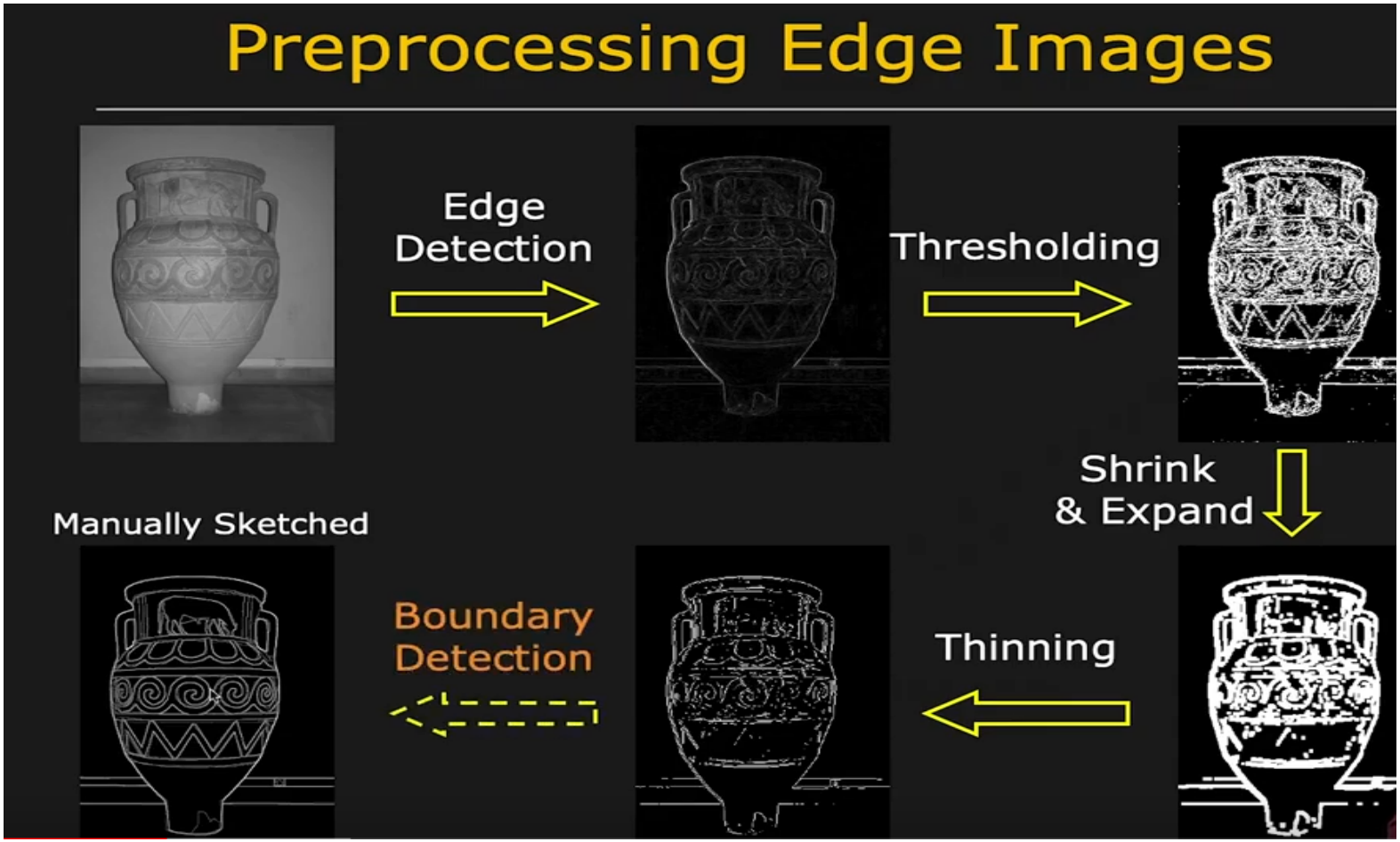
Topics:

(1) Fitting Lines and Curves to Edges

(2) Active Contours (Snakes)

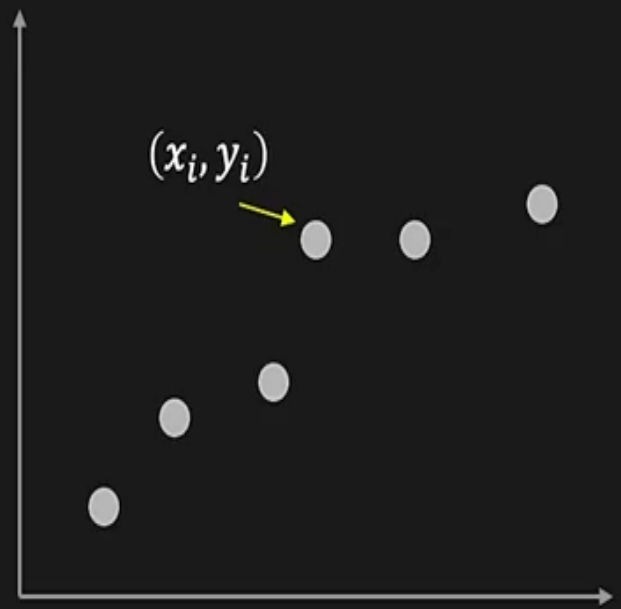
(3) The Hough Transform

(4) Generalized Hough Transform


Preprocessing Edge Images
Edge Detection
Thresholding
Shrink & Expand
Manually Sketched
Boundary Detection
Thinning

Fitting Lines to Edges

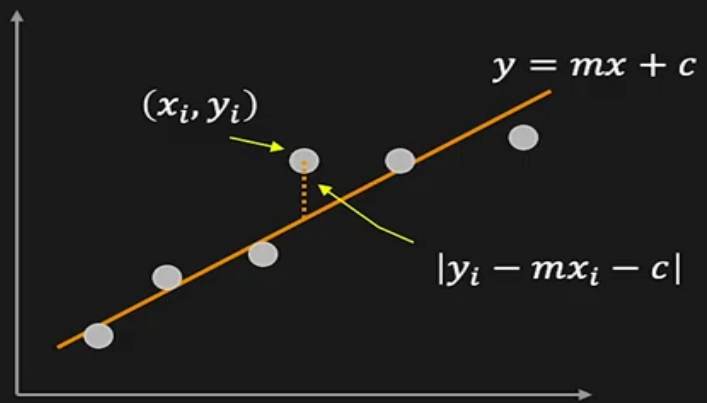
Given: Edge Points (x_i, y_i)



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:

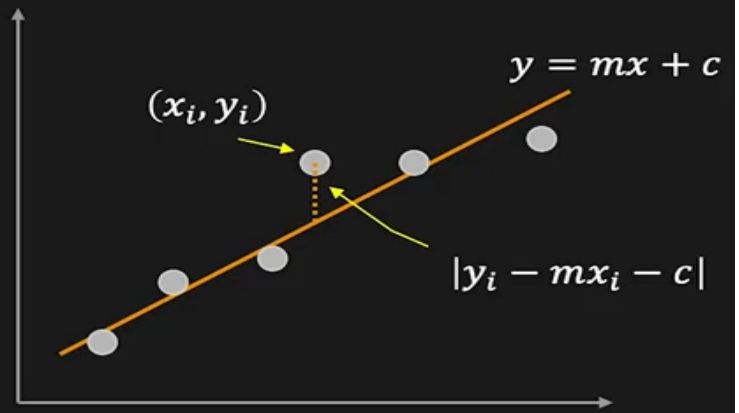
$$\frac{\partial E}{\partial m} = \frac{-2}{N}\sum_i x_i(y_i - mx_i - c) = 0$$

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

Fitting Lines to Edges

Given: Edge Points (x_i, y_i)

Task: Find (m, c)



Solution:

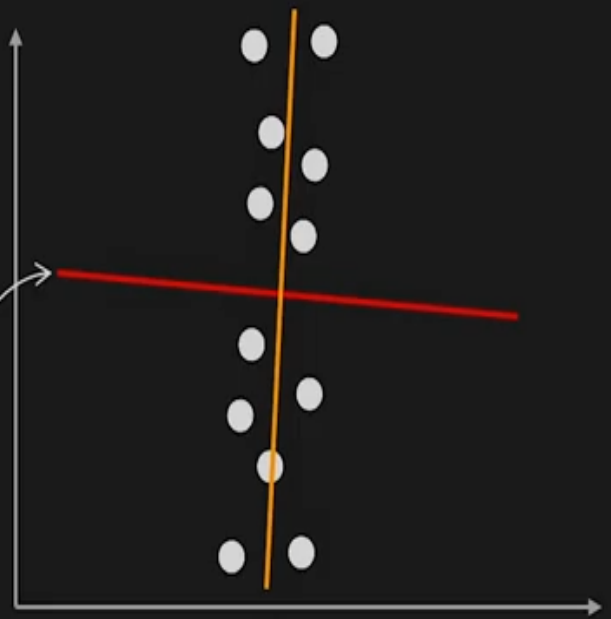
$$m = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2} \qquad c = \bar{y} - m\bar{x}$$

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

Fitting Lines to Edges

Problem: When the points represent a vertical line.

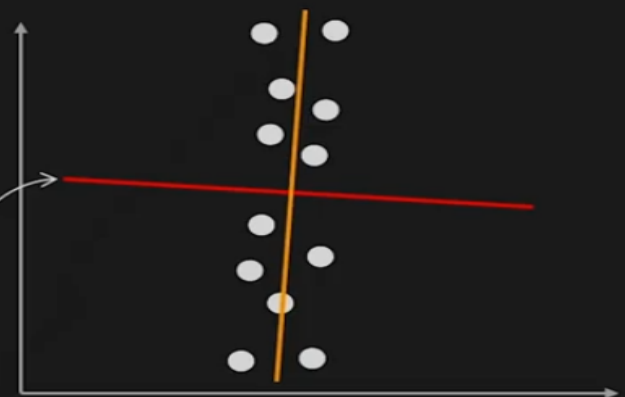
Line that minimizes E!



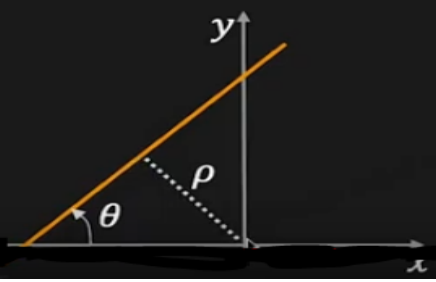
Fitting Lines to Edges

Problem: When the points represent a vertical line.

Line that minimizes E!



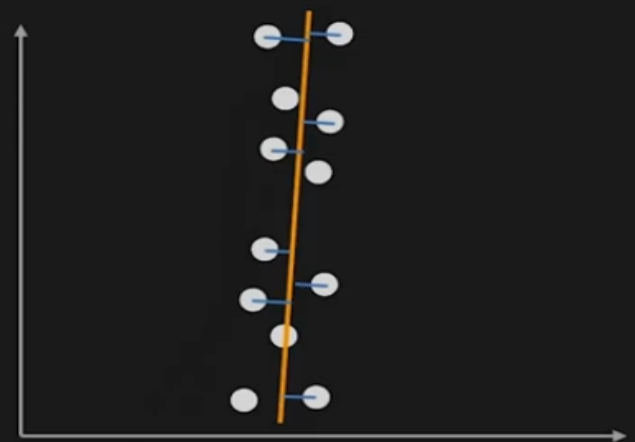
Solution: Use a different line equation



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

Fitting Lines to Edges

Problem: When the points represent a vertical line.



Minimize: Average Squared **Perpendicular** Distance

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

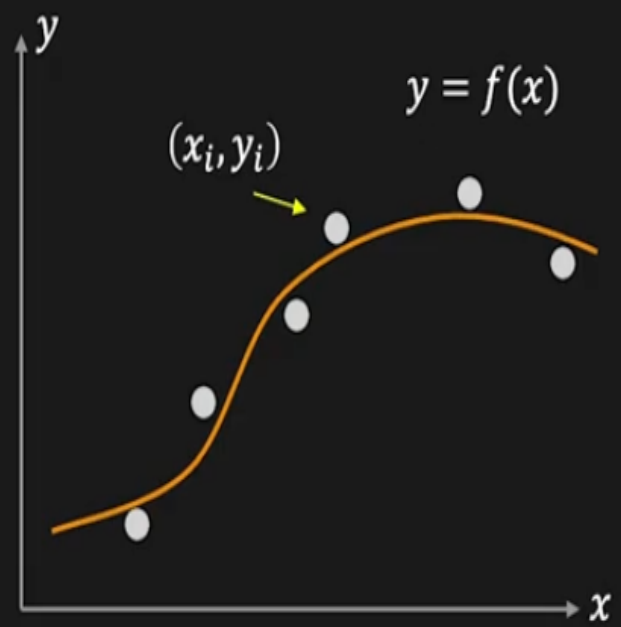
Fitting Curves to Edges

Given: Edge Points (x_i, y_i)

Task: Find polynomial

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

that best fits the points



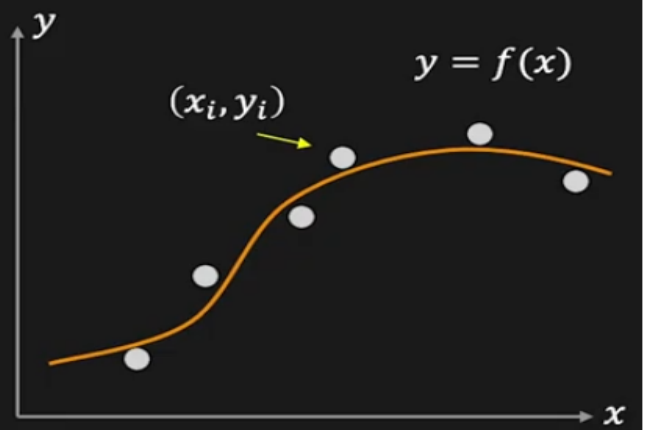
Fitting Curves to Edges

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:

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

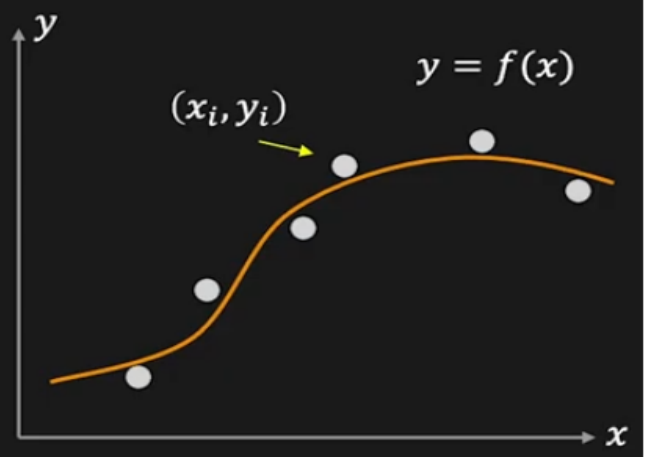
Solve the Linear System Using Least Squares Fit by:

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

Fitting Curves to Edges

Solving as a Linear System:

$$y_0 = ax_0^3 + bx_0^2 + cx_0 + d$$
$$y_1 = ax_1^3 + bx_1^2 + cx_1 + d$$
$$\vdots$$
$$y_i = ax_i^3 + bx_i^2 + cx_i + d$$
$$\vdots$$
$$y_n = ax_n^3 + bx_n^2 + cx_n + d$$



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 $\{a_j\}$ $(j = 0, \ldots, m)$ and n observations $\{(x_{ij}, y_i)\}$ $(i = 0, \ldots, n)$ $(n > m)$ can be written in a matrix form.

$$\begin{bmatrix} x_{00} & x_{01} & \cdots & x_{0m} \\ x_{10} & x_{11} & \cdots & x_{1m} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n0} & x_{n1} & \cdots & x_{nm} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}$$

$$X\mathbf{a} = \mathbf{y}$$

$X_{n\times m}$ is not a square matrix and hence not invertible.

$X_{n\times m}$ Known

$\mathbf{a}_{m\times 1}$ Unknown

$\mathbf{y}_{n\times 1}$ Known

Least Squares Solution:

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

$$X^+ = (X^T X)^{-1} X^T$$ (Pseudo Inverse)

$$\boxed{\mathbf{a} = X^+\mathbf{y}}$$

Difficulties for the Fitting Approach

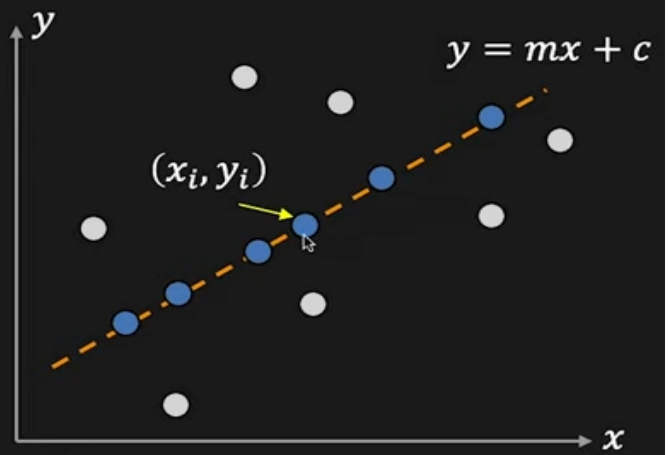
- Extraneous Data: Which points to fit to?
- Incomplete Data: Only part of the model is visible.
- Noise

Hough Transform: Line Detection

Given: Edge Points (x_i, y_i)

Task: Detect line

$$y = mx + c$$



Consider point (x_i, y_i)

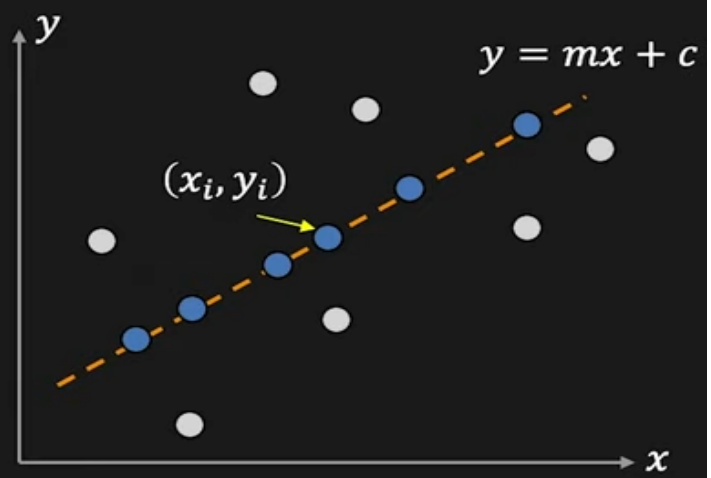
$$y_i = mx_i + c$$

Hough Transform: Line Detection

Given: Edge Points (x_i, y_i)

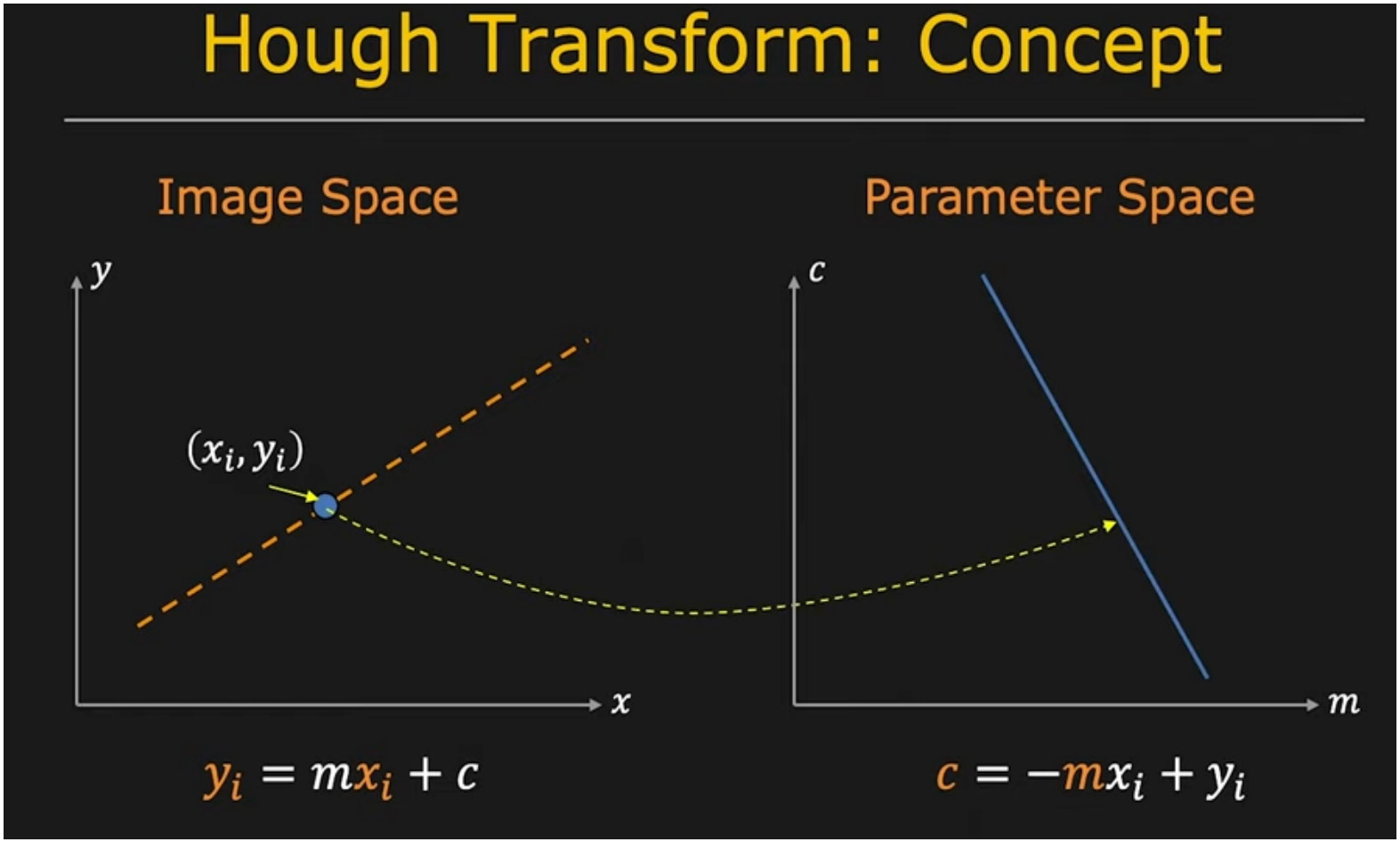
Task: Detect line

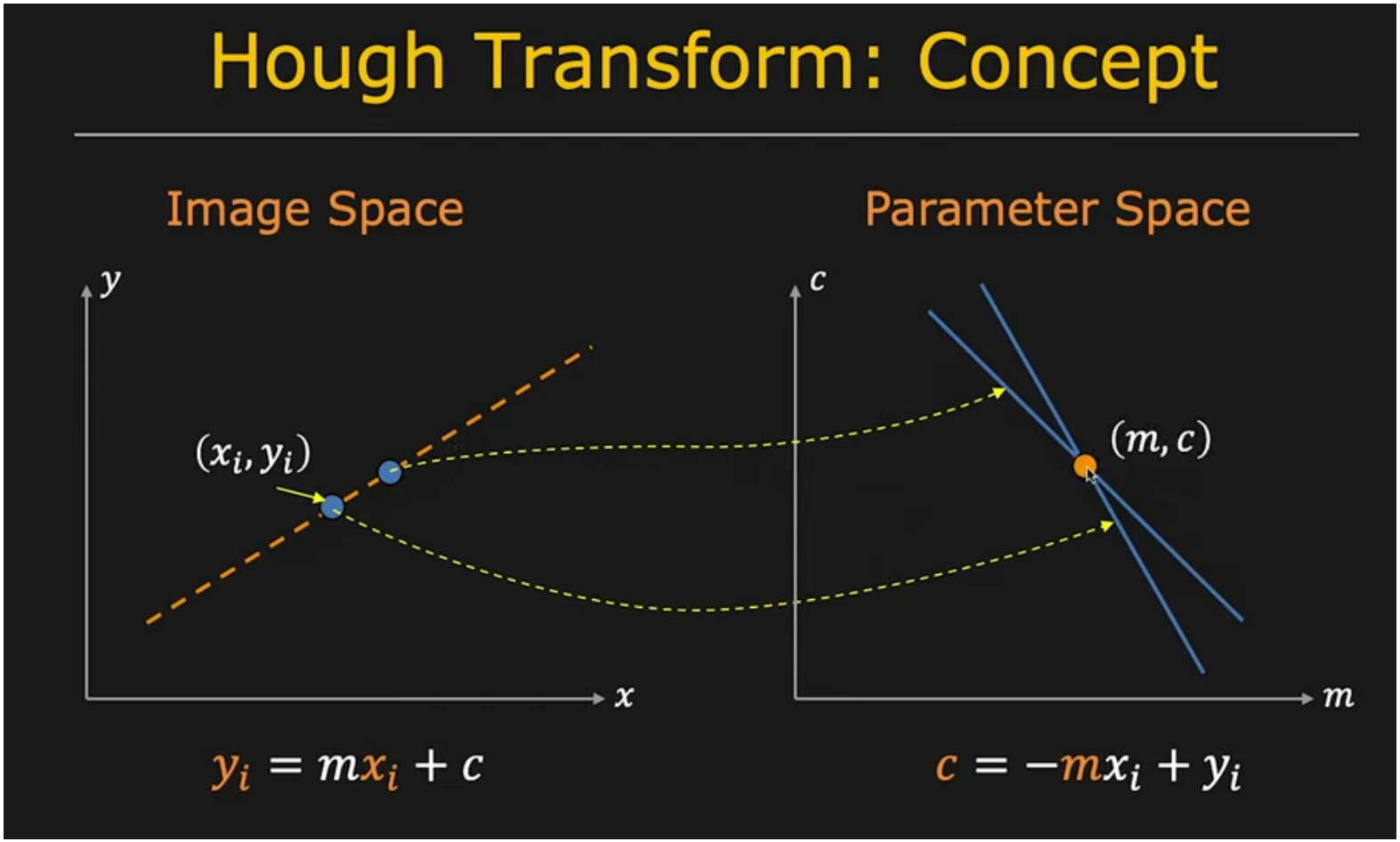
$$y = mx + c$$



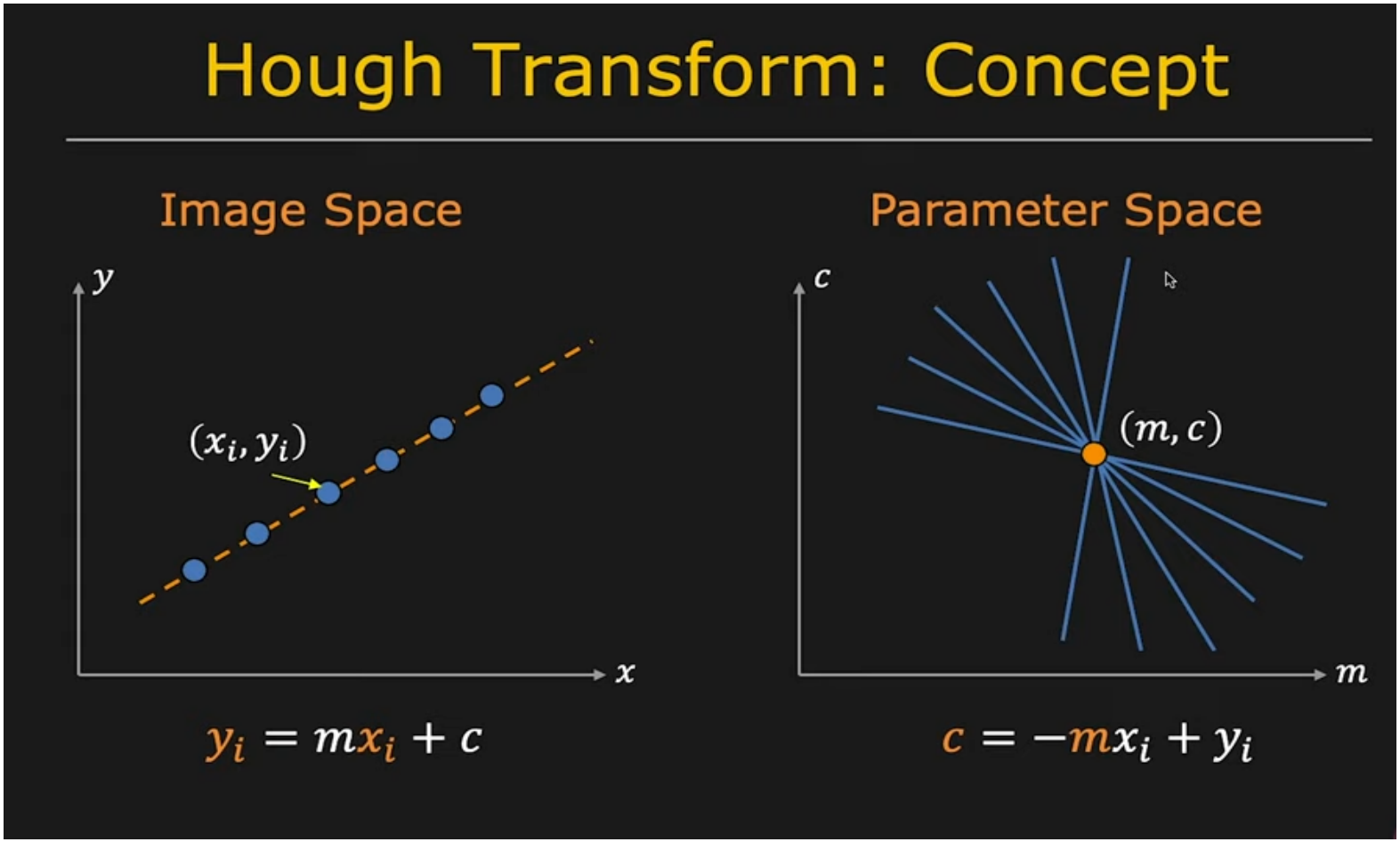
Consider point (x_i, y_i)

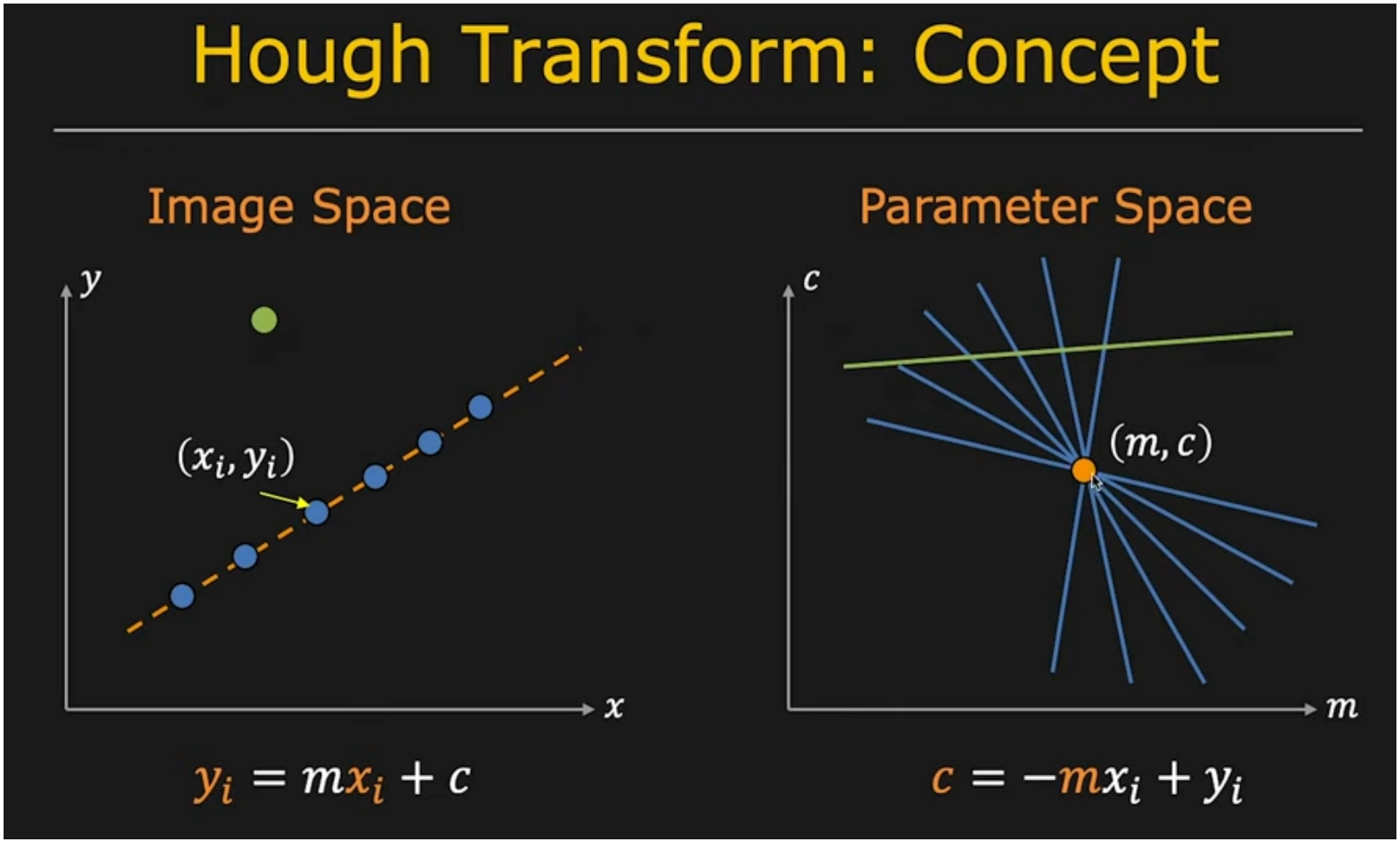
$$y_i = mx_i + c \iff c = -mx_i + y_i$$

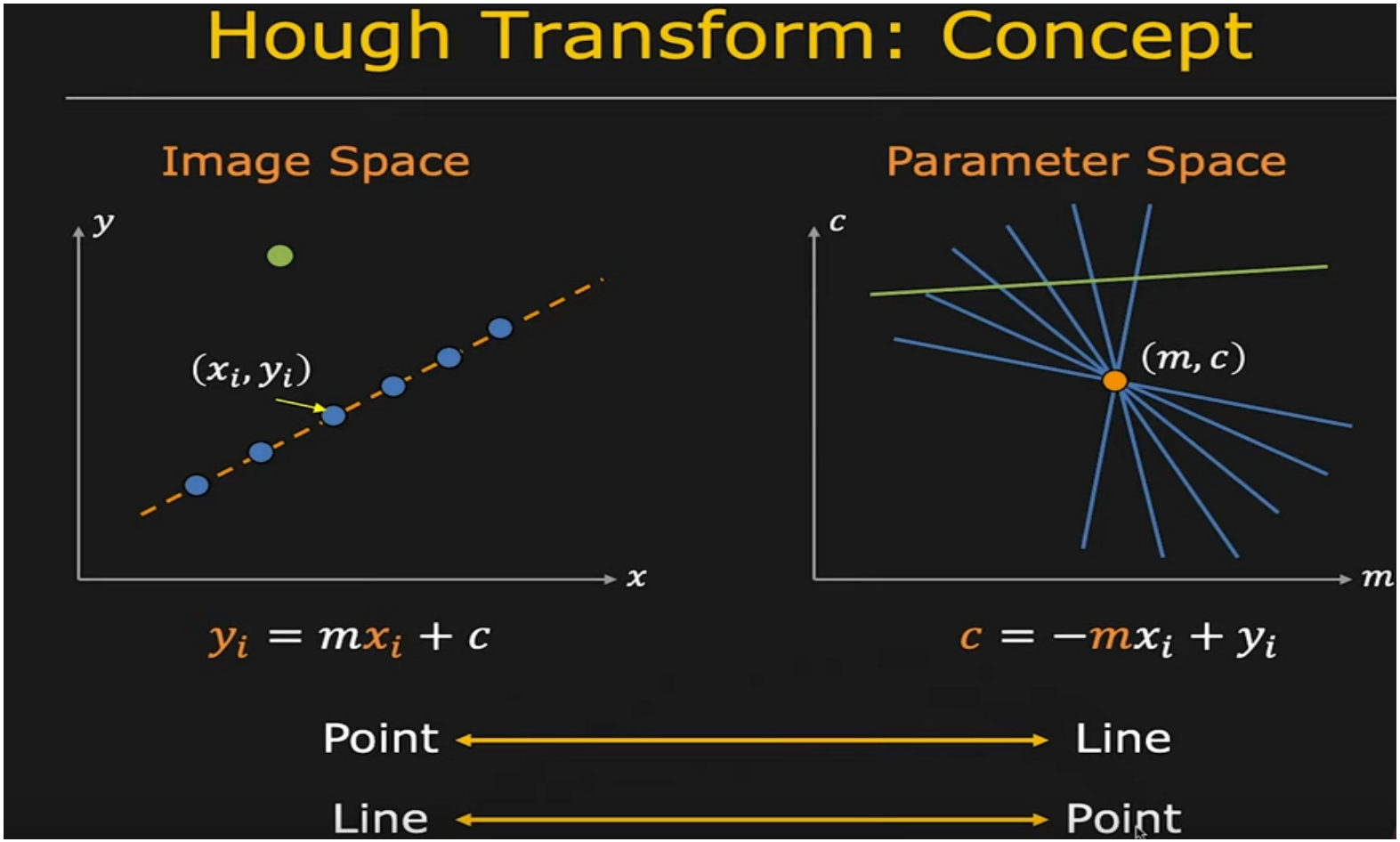

Hough Transform: Concept
Image Space
Parameter Space
y
x
c
m
(x_i, y_i)
$y_i = mx_i + c$
$c = -mx_i + y_i$


Hough Transform: Concept
Image Space
Parameter Space
y
x
c
m
(x_i, y_i)
(m, c)
$y_i = mx_i + c$
$c = -mx_i + y_i$

Hough Transform: Concept




Hough Transform: Concept
Image Space
Parameter Space
y
x
(x_i, y_i)
c
m
(m, c)
$y_i = mx_i + c$
$c = -mx_i + y_i$


Hough Transform: Concept
Image Space
Parameter Space
y
x
c
m
(x_i, y_i)
(m, c)
$y_i = mx_i + c$
$c = -mx_i + y_i$
Point
Line
Line
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: $c = -mx_i + y_i$

Image

$A(m, c)$

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

Line Detection Algorithm

Step 1. Quantize parameter space (m, c)

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

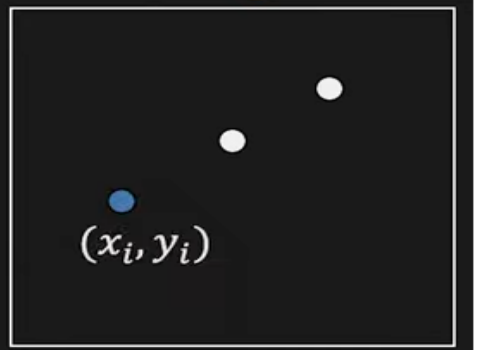
Step 3. Set $A(m, c) = 0$ for all (m, c)

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

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

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

Image



$A(m, c)$

c					
	1	0	0	0	0
	0	1	0	0	0
	0	0	1	0	0
	0	0	0	1	0
	0	0	0	0	1

Line Detection Algorithm

Step 1. Quantize parameter space (m, c)

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

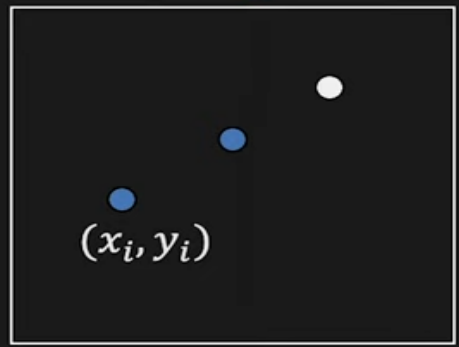
Step 3. Set $A(m, c) = 0$ for all (m, c)

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

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

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

Image



$A(m, c)$

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

Line Detection Algorithm

Step 1. Quantize parameter space (m, c)

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

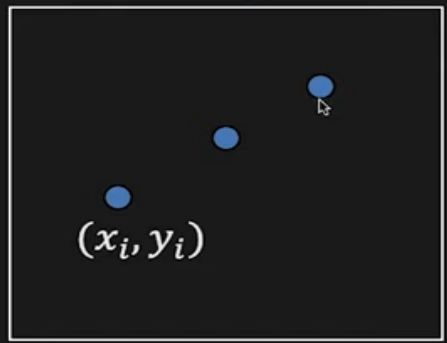
Step 3. Set $A(m, c) = 0$ for all (m, c)

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

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

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

Image



$A(m, c)$

c				
1	0	0	0	1
0	1	0	1	0
1	1	3	1	1
0	1	0	1	0
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)

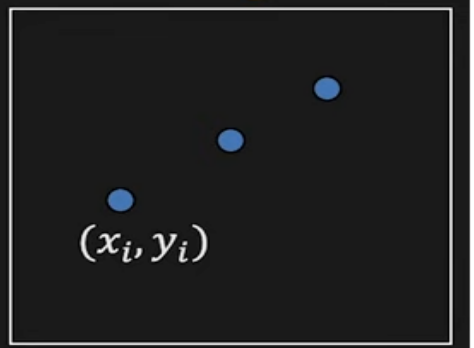
Step 4. For each edge point (x_i, y_i),

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

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

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

Image



$A(m, c)$

c					
	1	0	0	0	1
	0	1	0	1	0
	1	1	3	1	1
	0	1	0	1	0
	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)

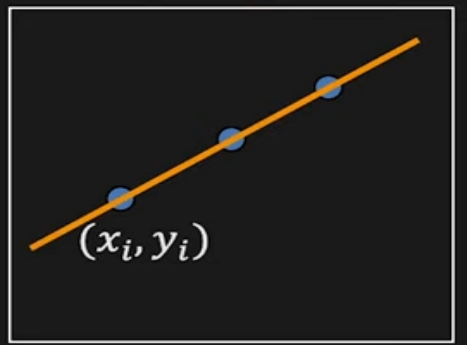
Step 4. For each edge point (x_i, y_i),

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

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

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

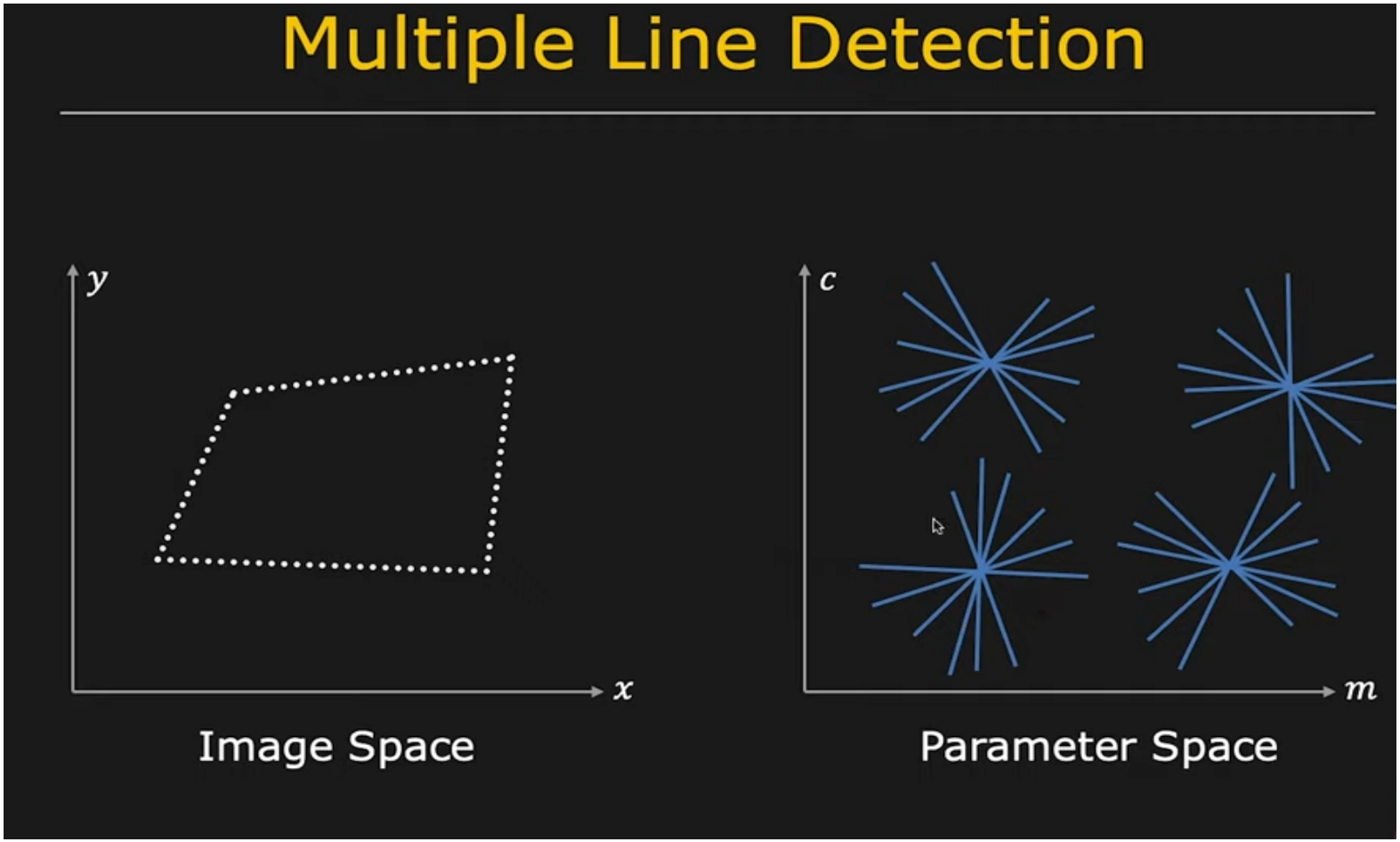
Image

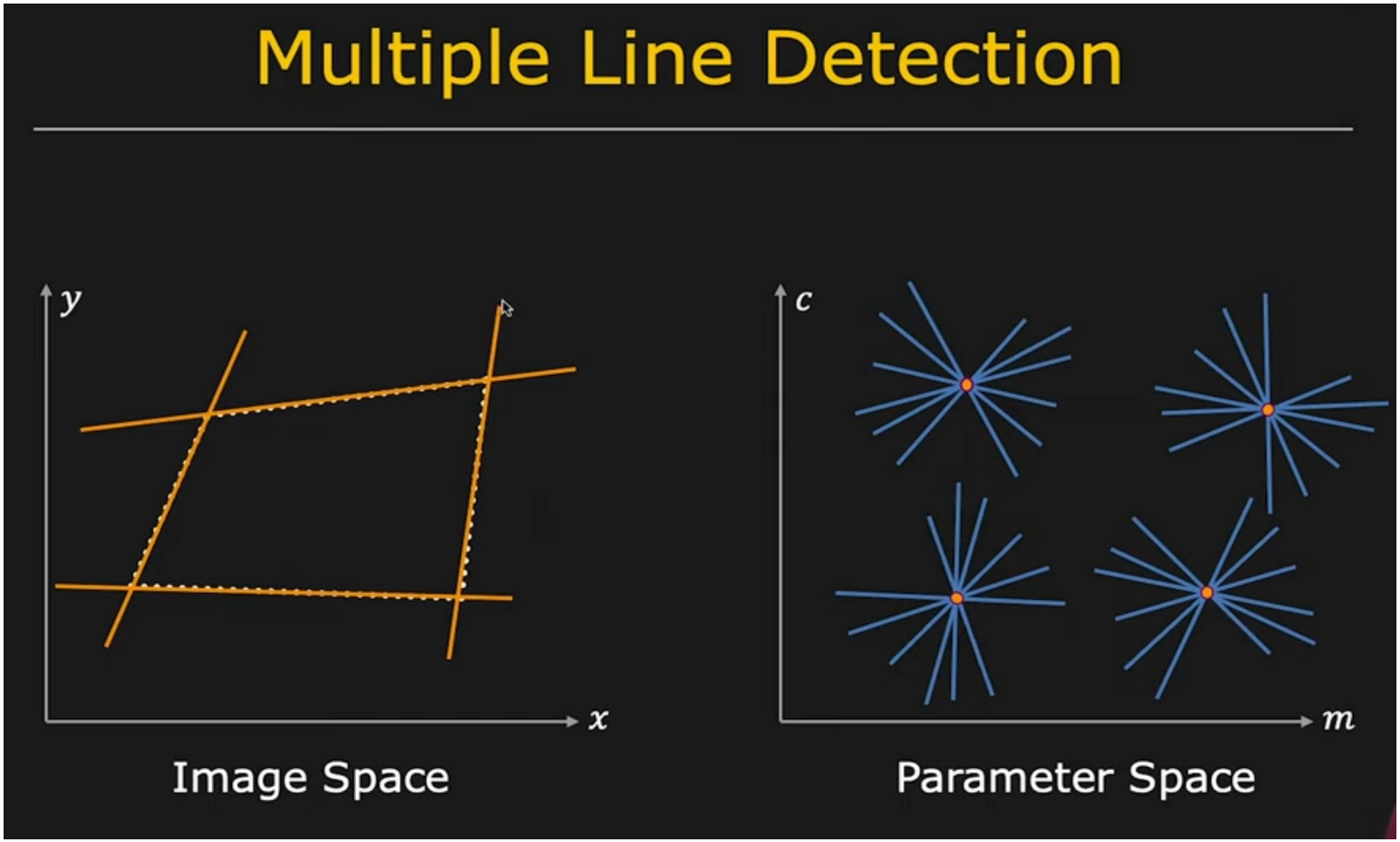


$A(m, c)$

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

m


Multiple Line Detection
y
x
Image Space
c
m
Parameter Space


Multiple Line Detection
y
x
Image Space
c
m
Parameter Space

Better Parameterization

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

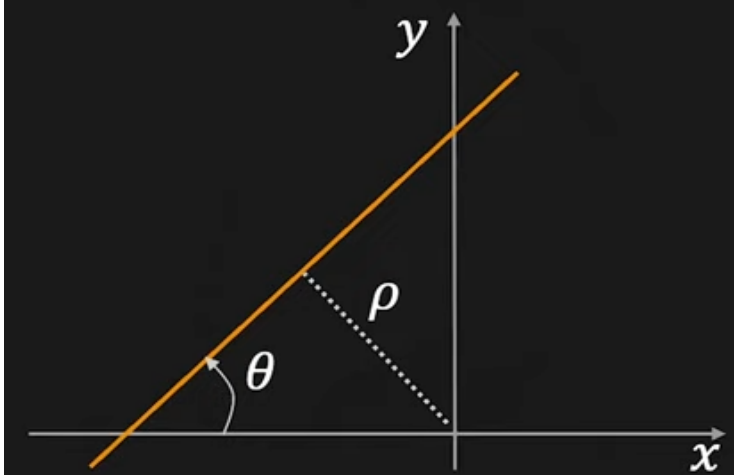
- Large Accumulator
- More Memory and Computation

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

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

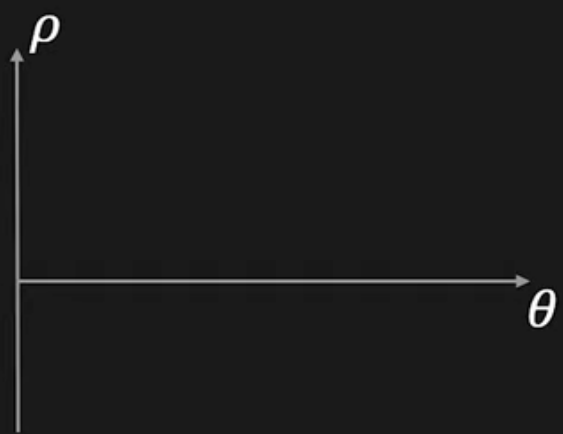
Better Parameterization

Image Space

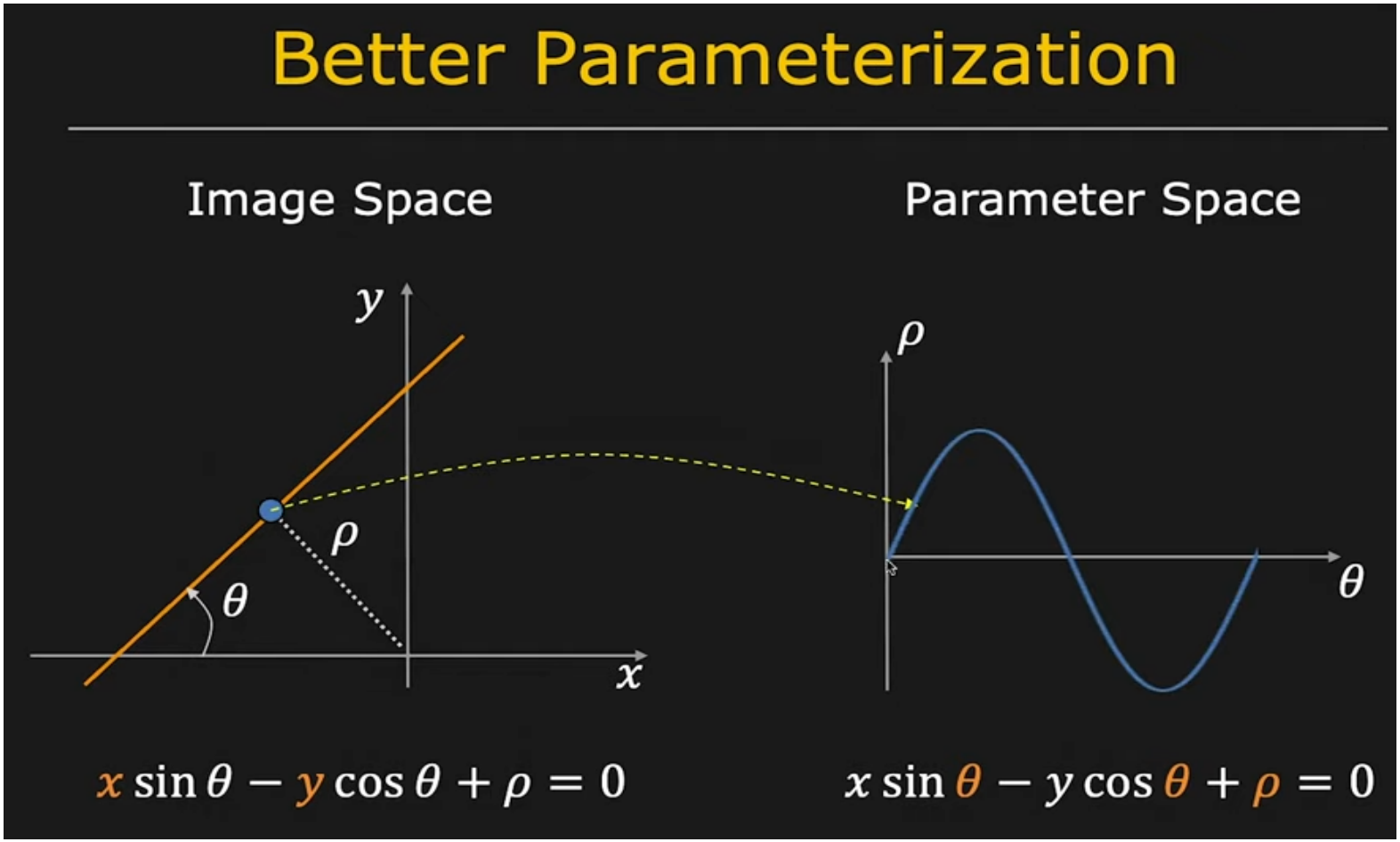


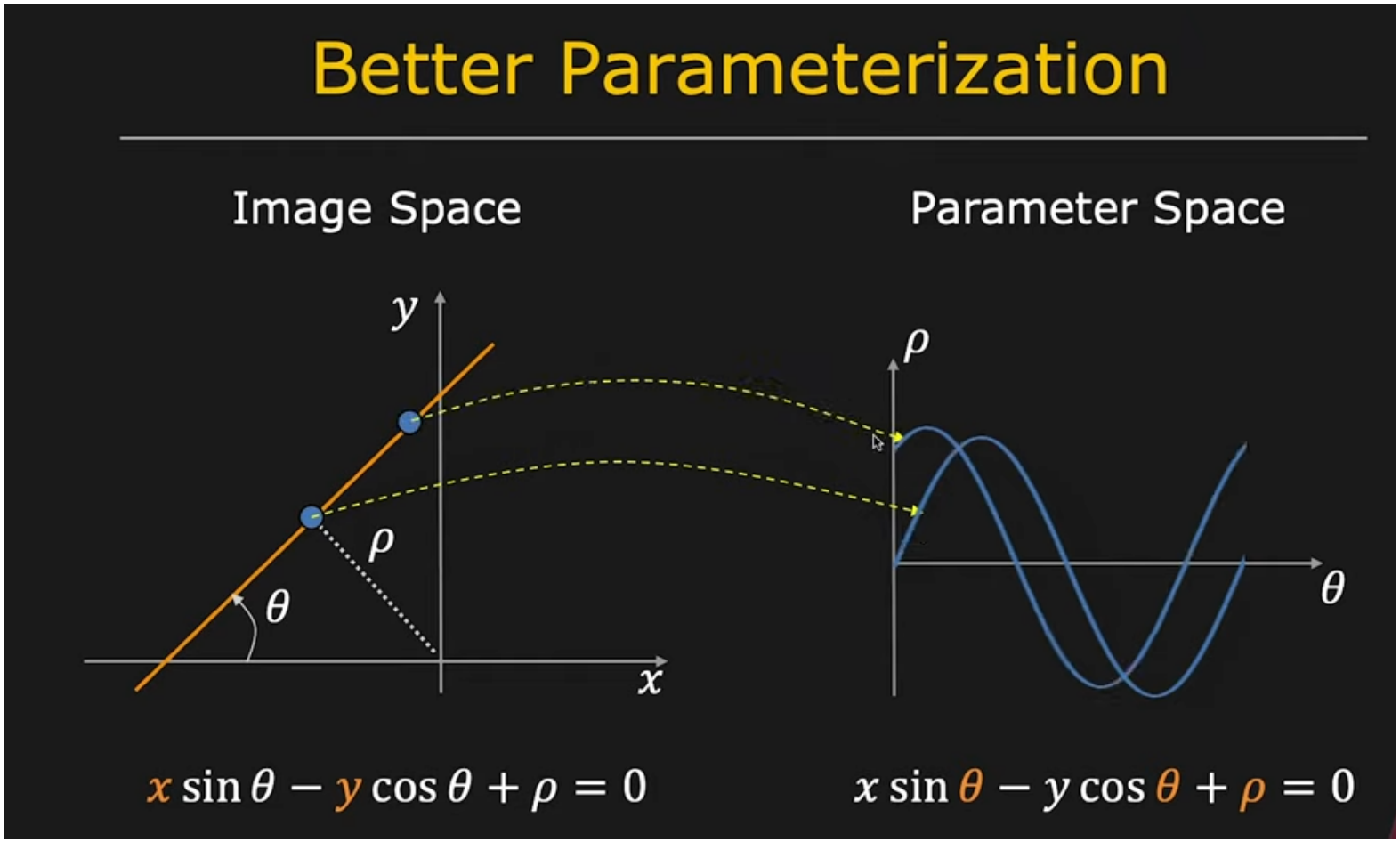
$$x \sin\theta - y \cos\theta + \rho = 0$$

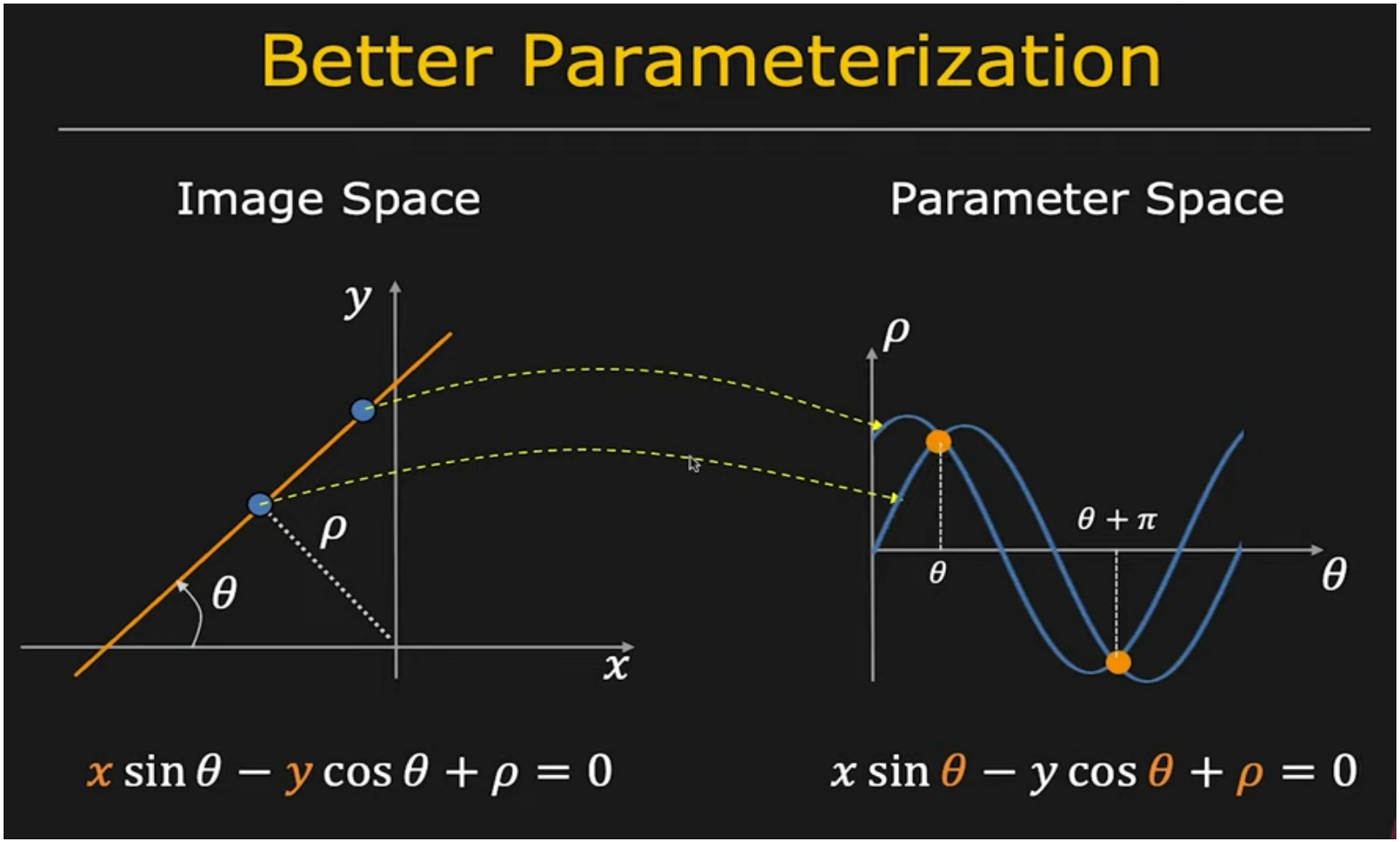
Parameter Space



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

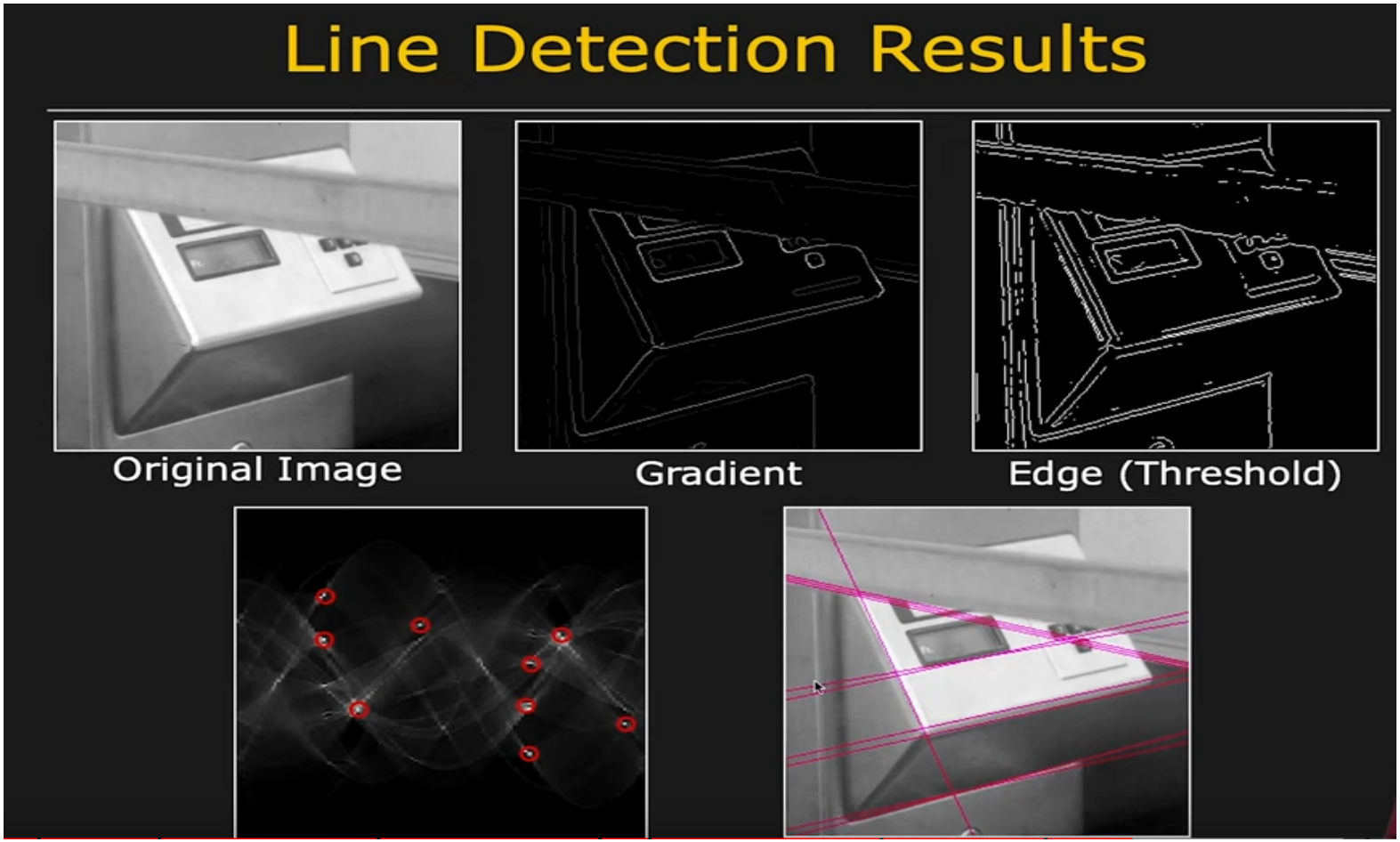

Better Parameterization
Image Space
Parameter Space
y
x
ρ
θ
ρ
θ
$x \sin\theta - y \cos\theta + \rho = 0$
$x \sin\theta - y \cos\theta + \rho = 0$


Better Parameterization
Image Space
Parameter Space
y
x
ρ
θ
ρ
θ
$x \sin\theta - y \cos\theta + \rho = 0$
$x \sin\theta - y \cos\theta + \rho = 0$

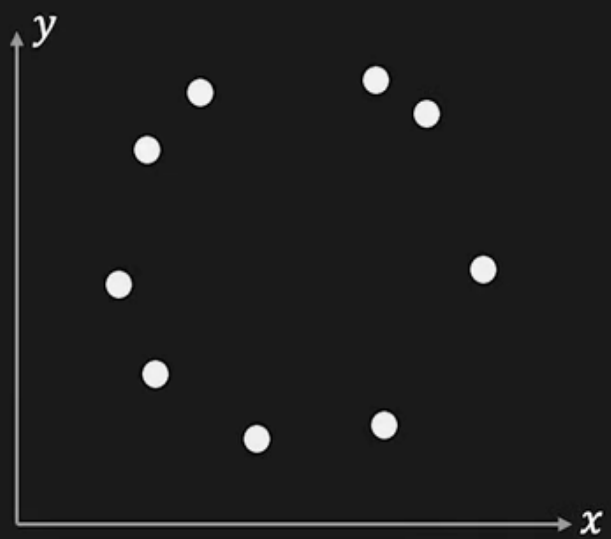

Better Parameterization
Image Space
Parameter Space
y
x
ρ
θ
ρ
θ + π
θ
θ
x sin θ − y cos θ + ρ = 0
x sin θ − y cos θ + ρ = 0

Hough Transform Mechanics

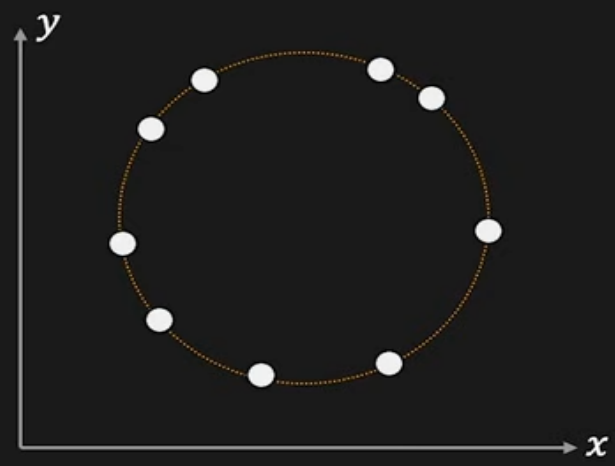
- How big should the accumulator cells be?
  - Too big, and different lines may be merged
  - Too small, and noise causes lines to be missed


Line Detection Results
Original Image
Gradient
Edge (Threshold)

Hough Transform: Circle Detection

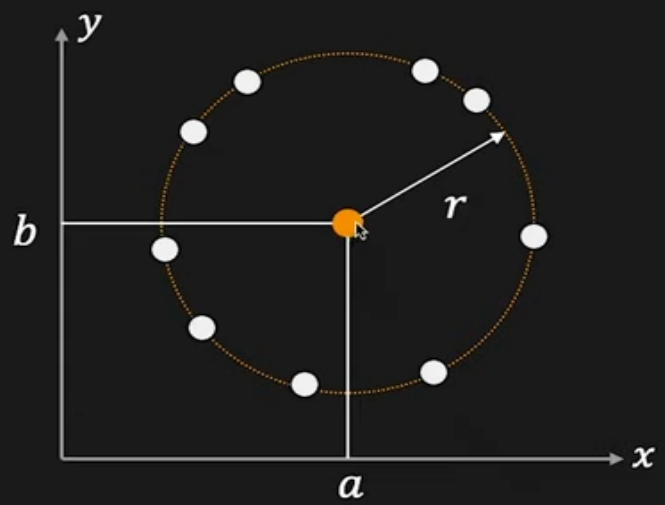


Hough Transform: Circle Detection



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

Hough Transform: Circle Detection

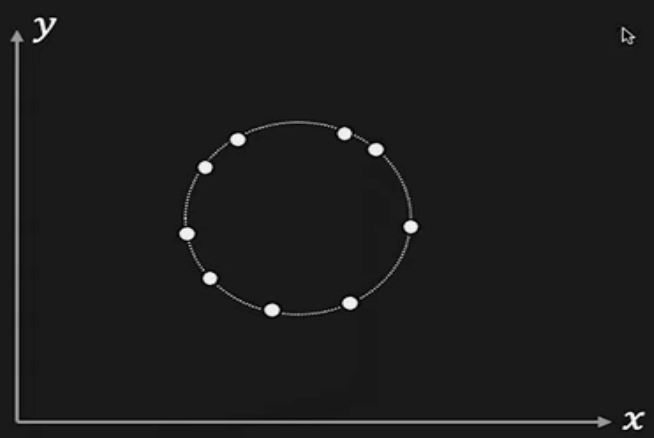


Equation of Circle: $(x_i - a)^2 + (y_i - b)^2 = r^2$

Hough Transform: Circle Detection

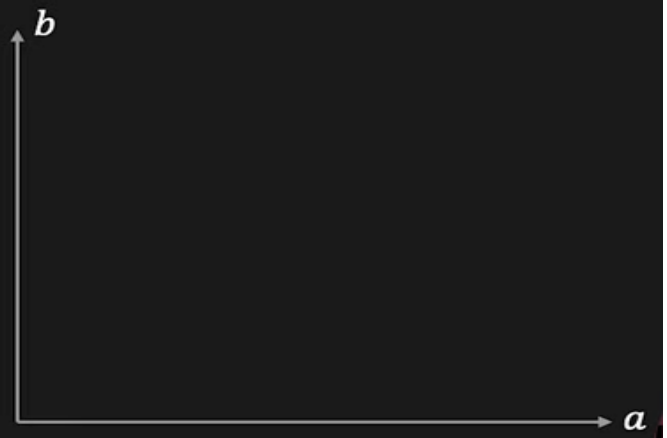
If radius r is known: Accumulator Array: $A(a, b)$

Image Space



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

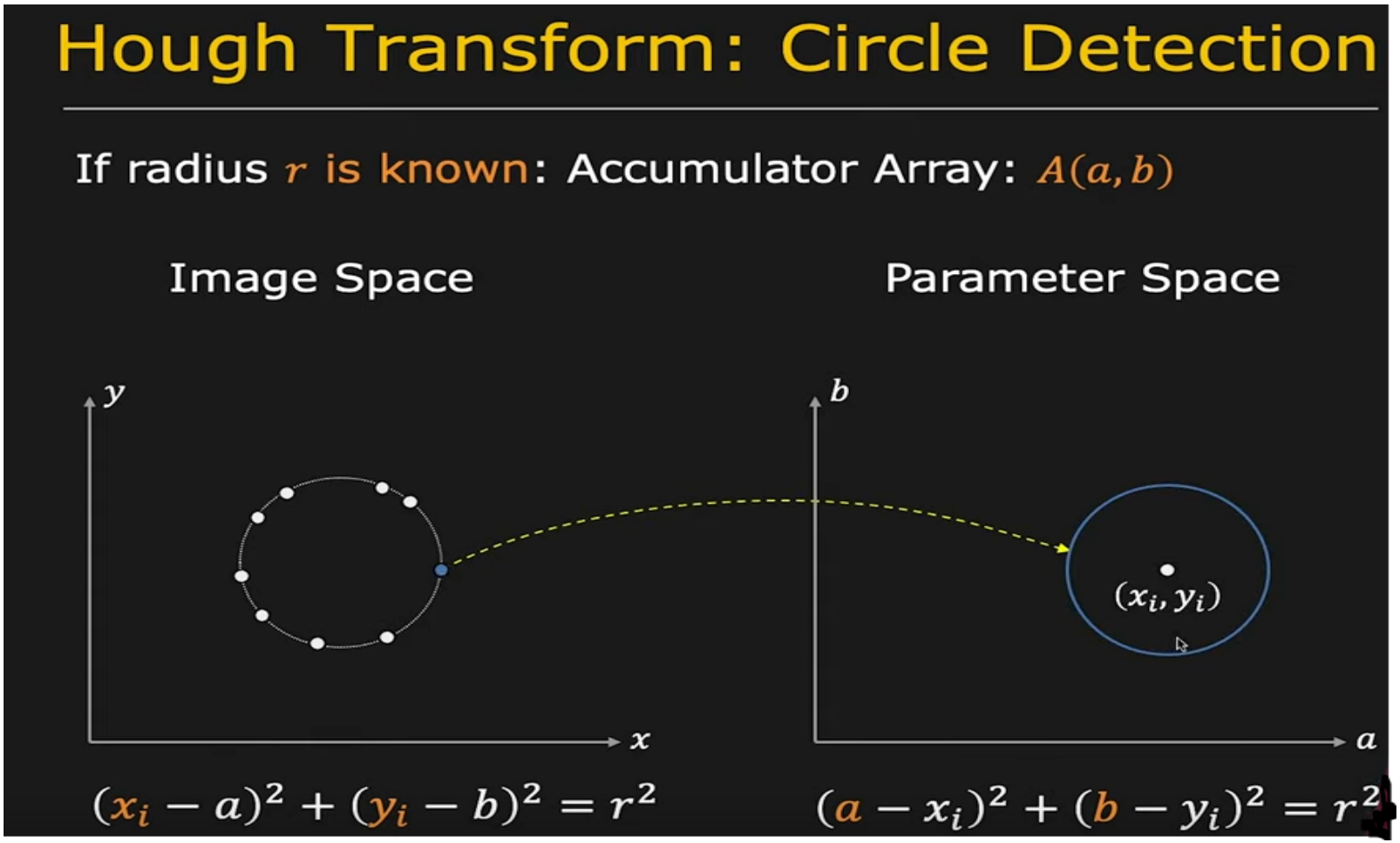
Parameter Space



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

Hough Transform: Circle Detection

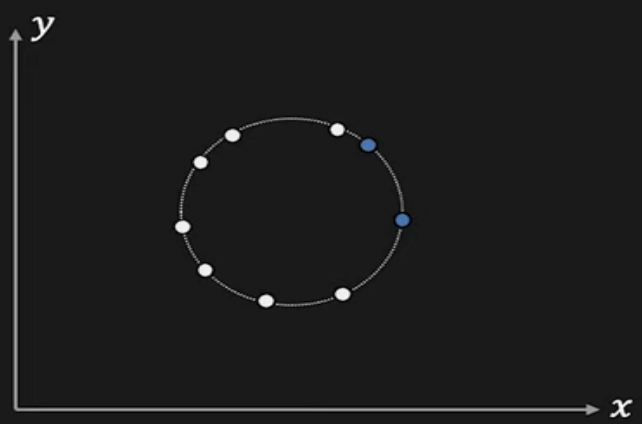
If radius r is known: Accumulator Array: $A(a, b)$



Hough Transform: Circle Detection

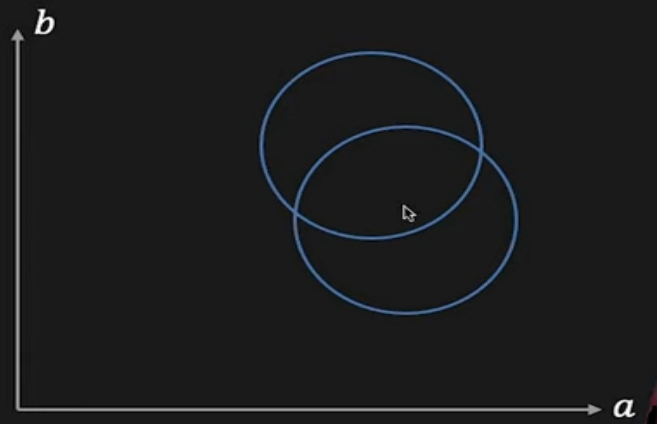
If radius r is known: Accumulator Array: $A(a, b)$

Image Space



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

Parameter Space

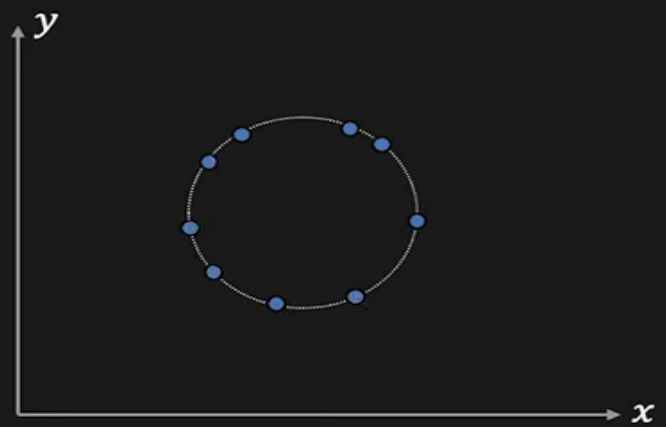


$$(a - x_i)^2 + (b - y_i)^2 = r^2$$

Hough Transform: Circle Detection

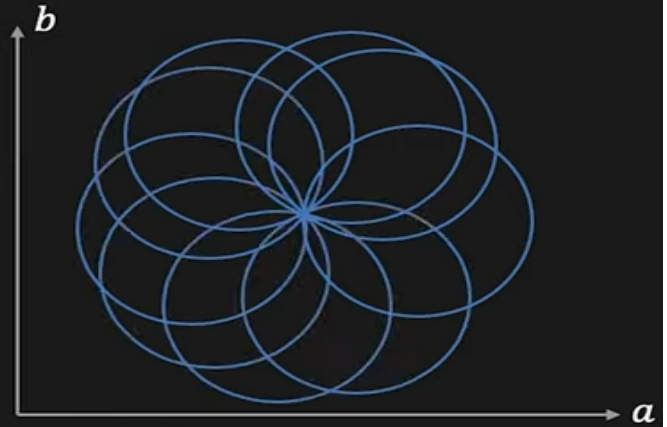
If radius r is known: Accumulator Array: $A(a, b)$

Image Space



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

Parameter Space

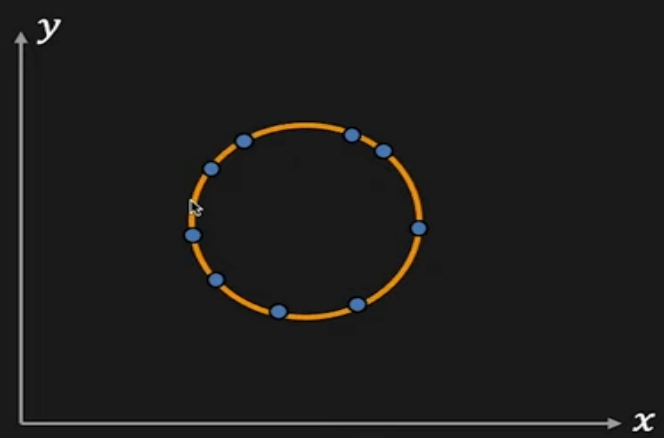


$$(a - x_i)^2 + (b - y_i)^2 = r^2$$

Hough Transform: Circle Detection

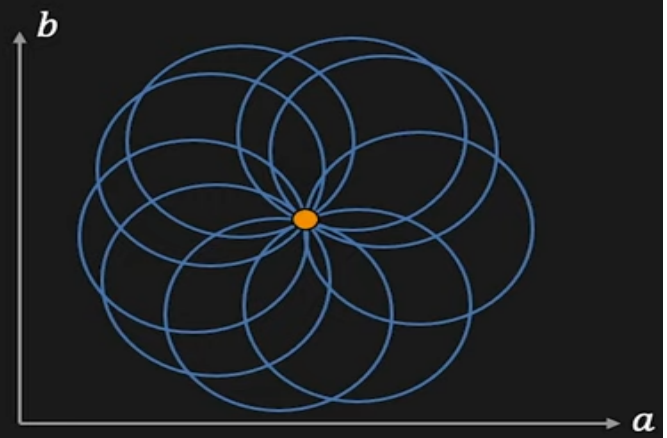
If radius r is known: Accumulator Array: $A(a, b)$

Image Space



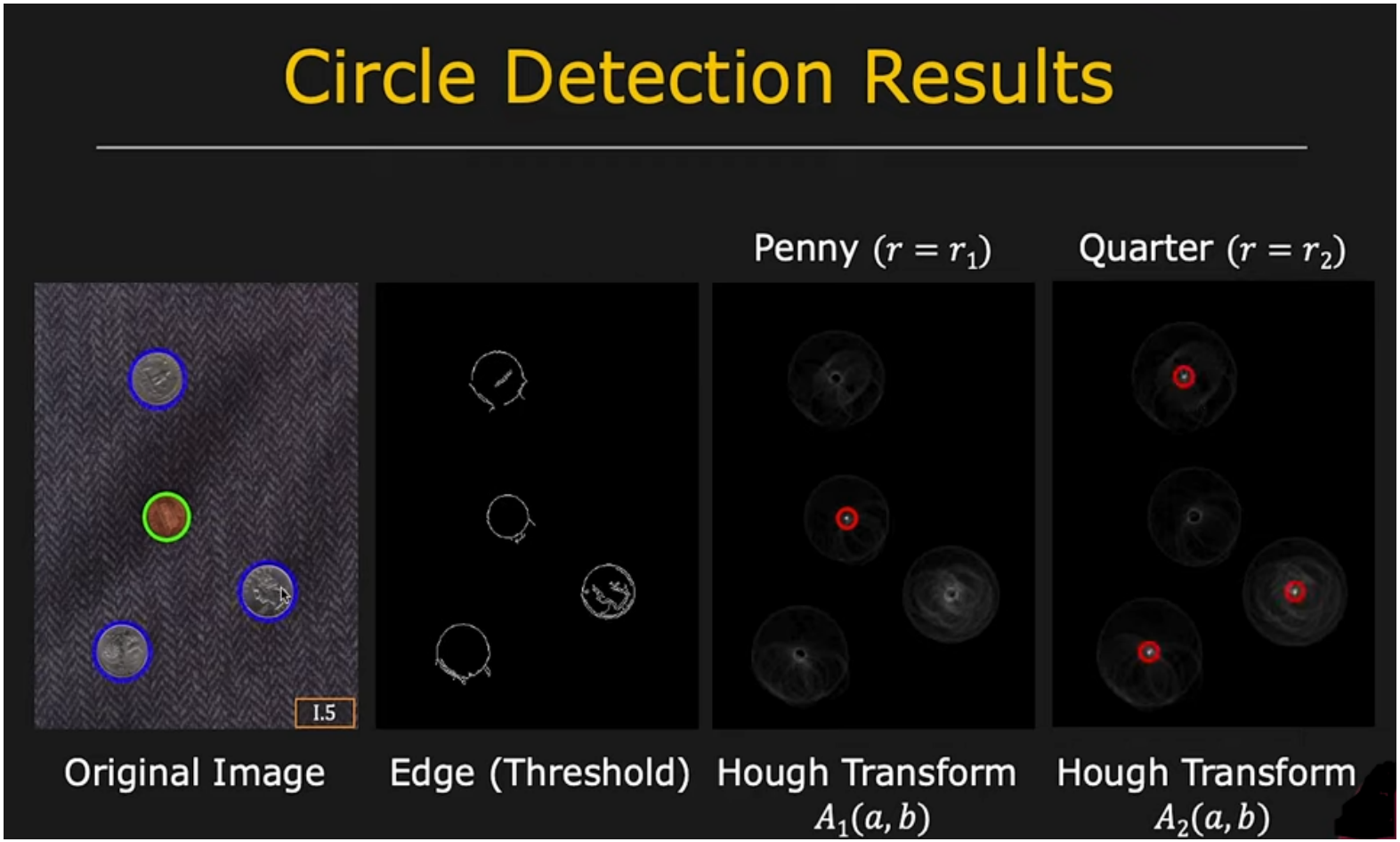
$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

Parameter Space



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

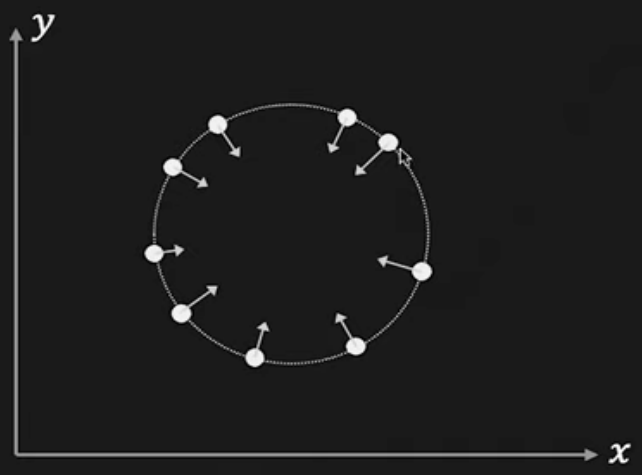
Circle Detection Results



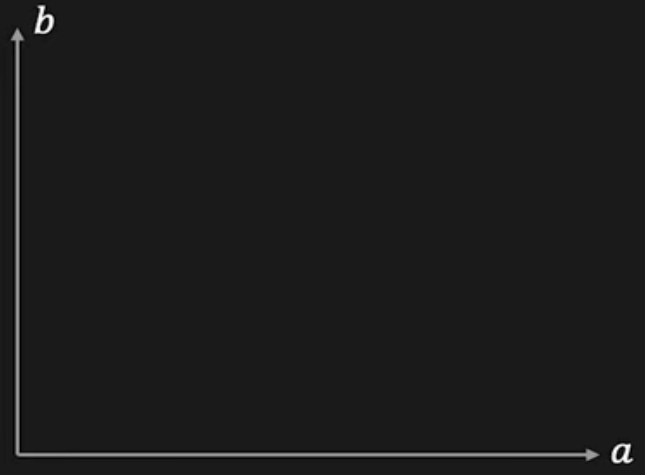
Using Gradient Information

Given: Edge Location (x_i, y_i), Edge Direction φ_i and Radius r

Image Space



Parameter Space

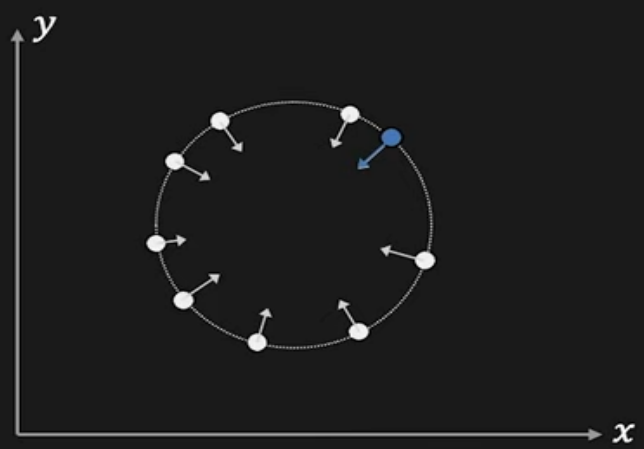


$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

Using Gradient Information

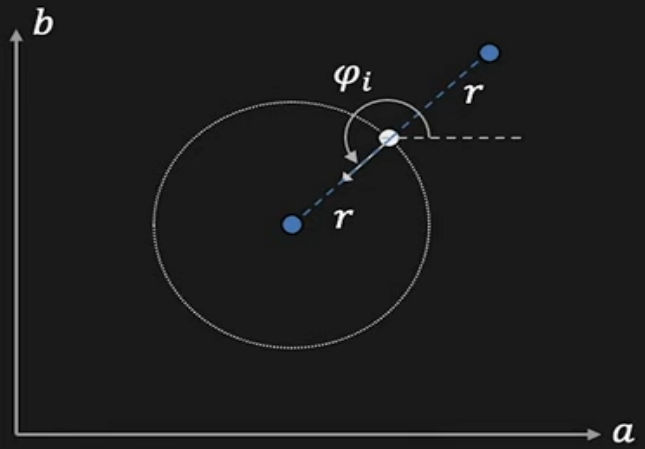
Given: Edge Location (x_i, y_i), Edge Direction φ_i and Radius r

Image Space



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

Parameter Space

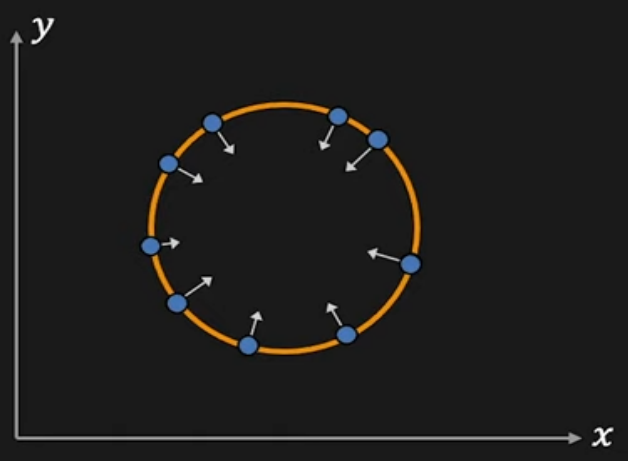


$$a = x_i \pm r\cos\varphi_i$$
$$b = y_i \pm r\sin\varphi_i$$

Using Gradient Information

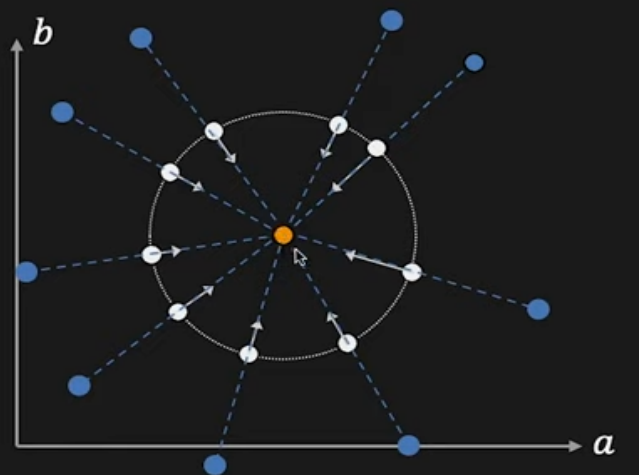
Given: Edge Location (x_i, y_i), Edge Direction φ_i and Radius r

Image Space



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

Parameter Space



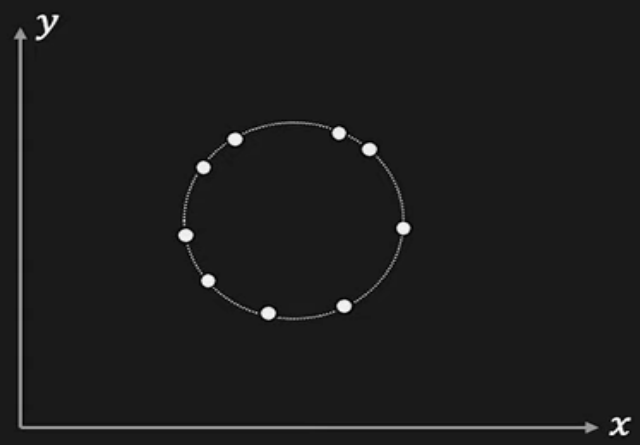
$$a = x_i \pm r\cos\varphi_i$$

$$b = y_i \pm r\sin\varphi_i$$

Hough Transform: Circle Detection

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

Image Space

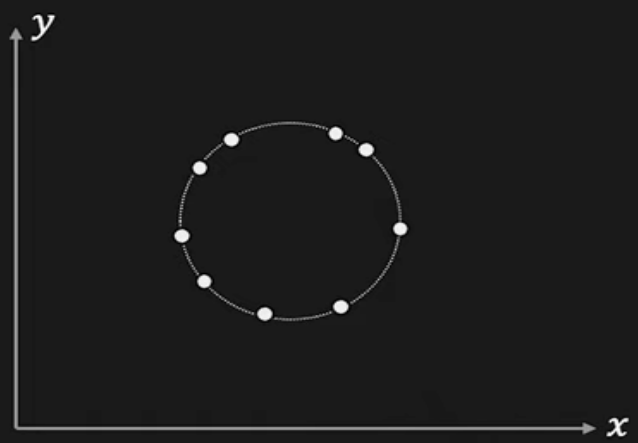


$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

Hough Transform: Circle Detection

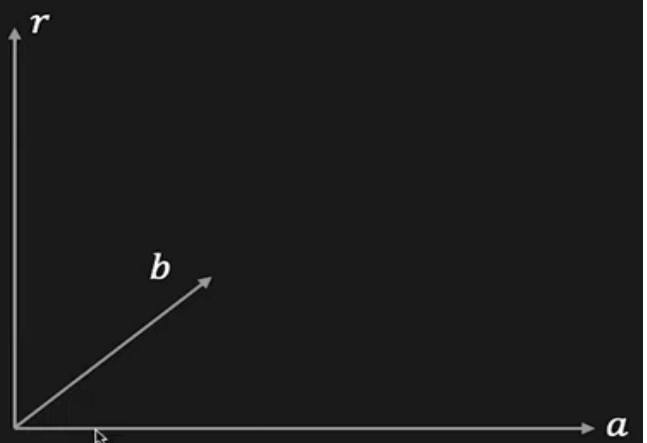
If radius r is NOT known: Accumulator Array: $A(a, b, r)$

Image Space



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

Parameter Space



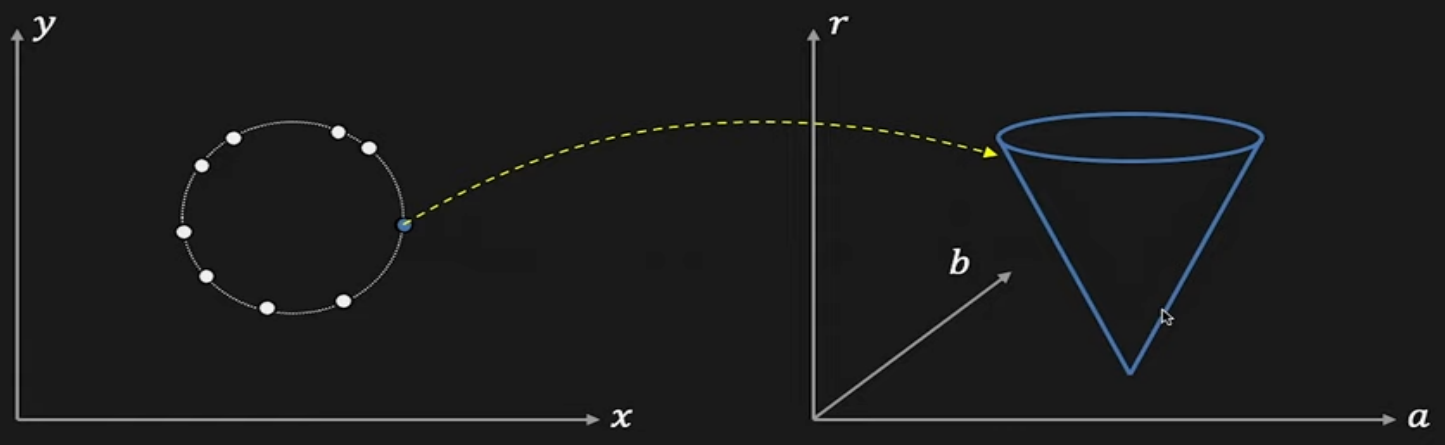
$$(a - x_i)^2 + (b - y_i)^2 = r^2$$

Hough Transform: Circle Detection

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

Image Space

Parameter Space



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

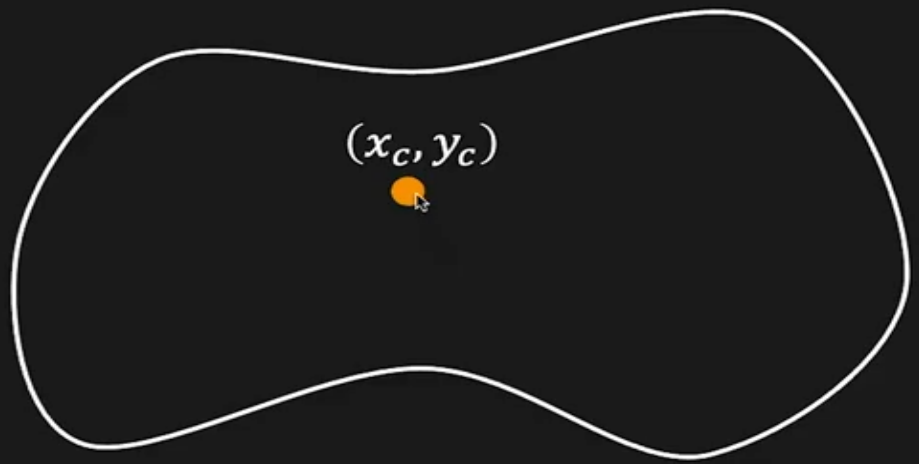
$$(a - x_i)^2 + (b - y_i)^2 = r^2$$

Generalized Hough Transform

Find shapes that cannot be described by equations

Generalized Hough Transform

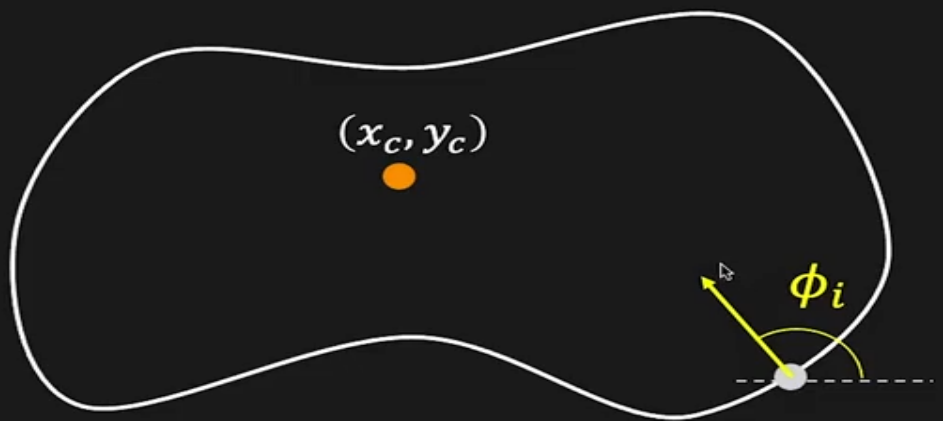
Find shapes that cannot be described by equations



Reference point: (x_c, y_c)

Generalized Hough Transform

Find shapes that cannot be described by equations

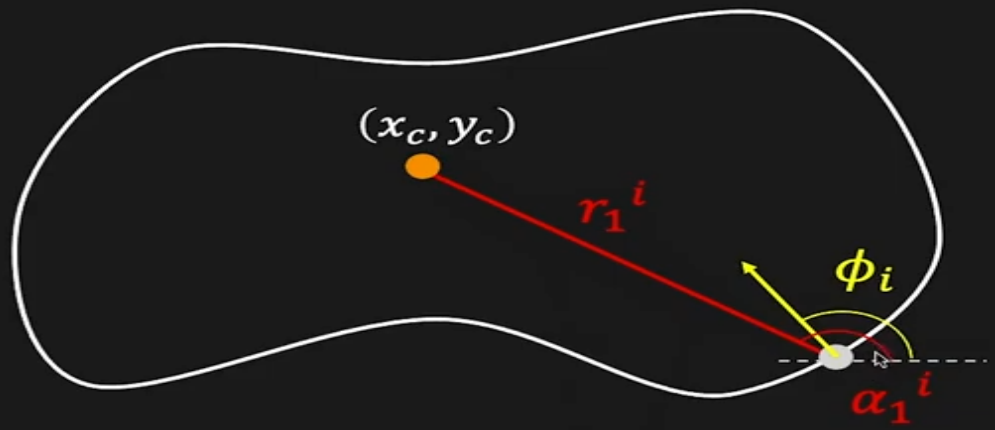


Reference point: (x_c, y_c)

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

Generalized Hough Transform

Find shapes that cannot be described by equations



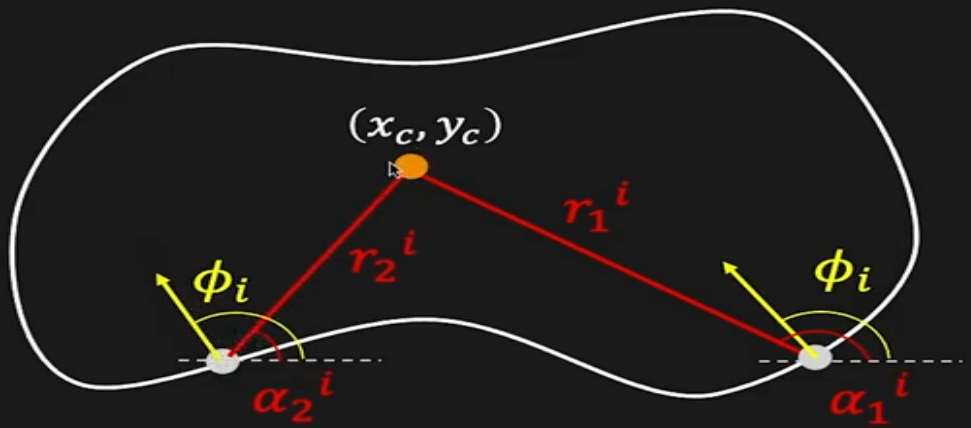
Reference point: (x_c, y_c)

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

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

Generalized Hough Transform

Find shapes that cannot be described by equations

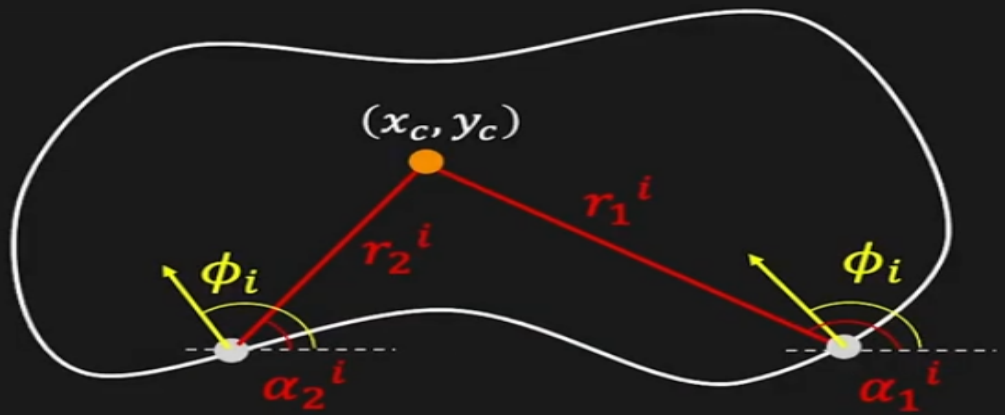


Reference point: (x_c, y_c)

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

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

Hough Model



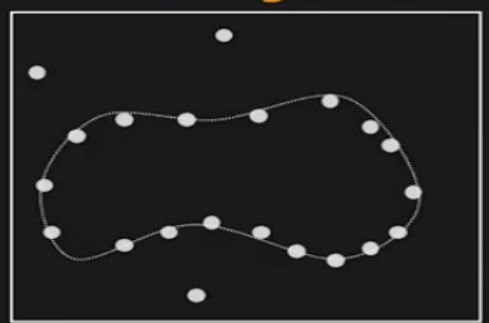
ϕ-Table:

Edge Direction	$\vec{r} = (r, \alpha)$
ϕ_1	$\vec{r}_1^{\,1}, \vec{r}_2^{\,1}, \vec{r}_3^{\,1}$
ϕ_2	$\vec{r}_1^{\,2}, \vec{r}_2^{\,2}$
$\vdots$	$\vdots$
ϕ_n	$\vec{r}_1^{\,n}, \vec{r}_2^{\,n}, \vec{r}_3^{\,n}, \vec{r}_4^{\,n}$

Shree K. Nayar

Generalized Hough Algorithm

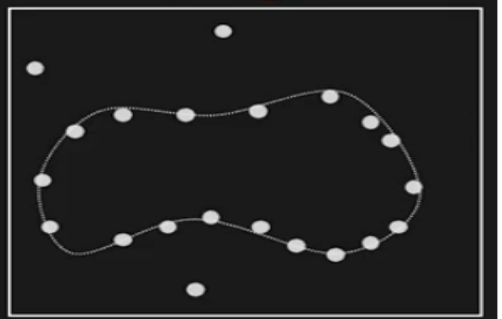
- Create accumulator array $A(x_c, y_c)$



Generalized Hough Algorithm

- Create accumulator array $A(x_c, y_c)$
- Set $A(x_c, y_c) = 0$ for all (x_c, y_c)

Image



$A(x_c, y_c)$

x_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

y_c

Generalized Hough Algorithm

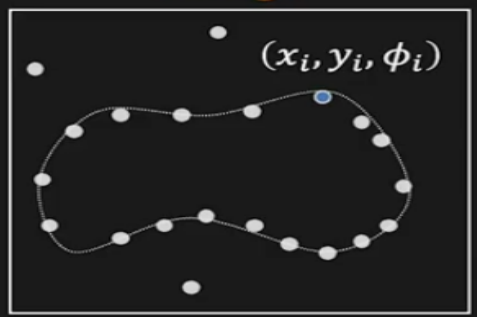
- Create accumulator array $A(x_c, y_c)$
- Set $A(x_c, y_c) = 0$ for all (x_c, y_c)
- For each edge point (x_i, y_i, ϕ_i),

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

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

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

Image



$A(x_c, y_c)$

x_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

y_c

Shree K. Nayar

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, ϕ_i),

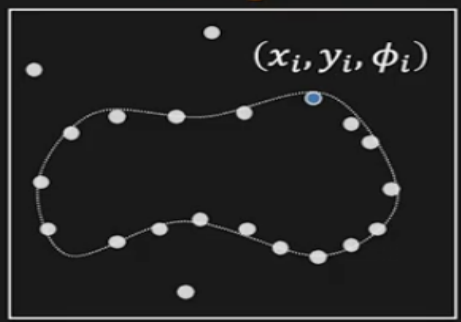
  For each entry $\phi_i \rightarrow \vec{r}_k{}^i$ in ϕ – table,

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

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

  $$A(x_c, y_c) = A(x_c, y_c) + 1$$
- Find local maxima in $A(x_c, y_c)$

Image



$A(x_c, y_c)$

x_c / y_c				
0	0	0	0	0
0	2	0	1	0
0	0	4	1	0
0	2	0	0	0
0	0	0	1	0

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, ϕ_i),

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

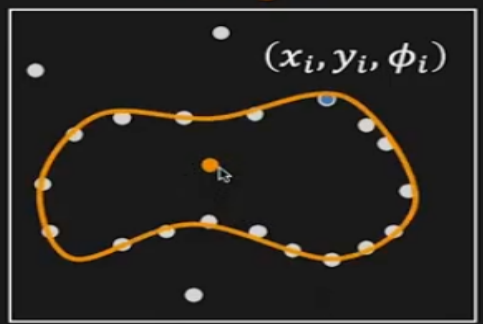
  $$x_c = x_i \pm r_k{}^i \cos(\alpha_k{}^i)$$

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

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

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

Image

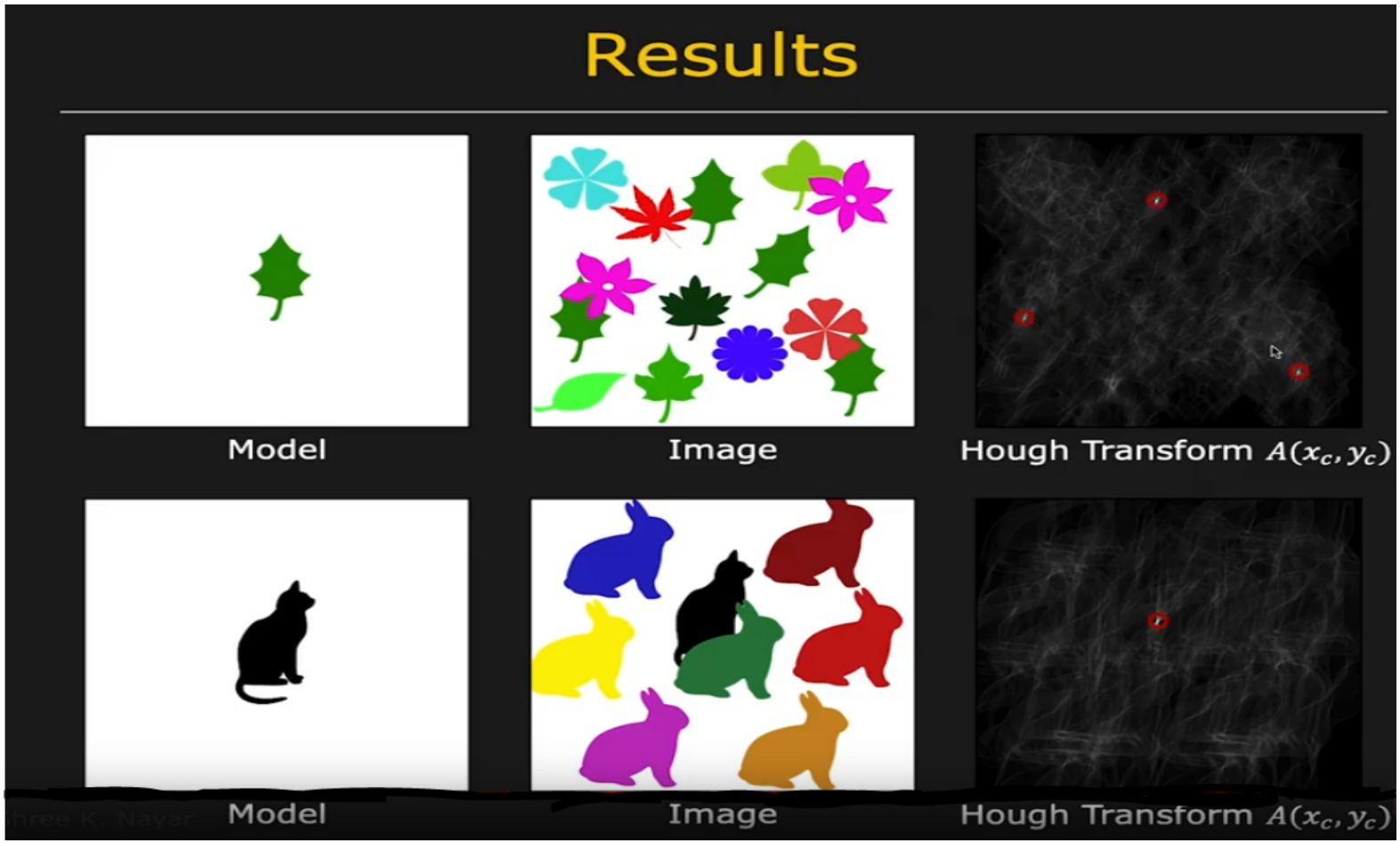


$A(x_c, y_c)$

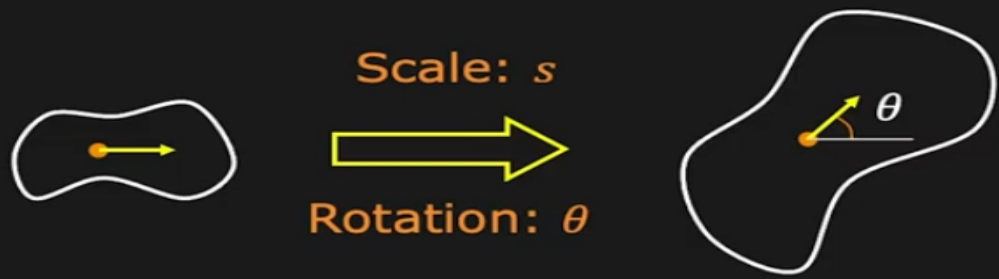
x_c

0	0	0	0	0
0	2	0	1	0
0	0	4	1	0
0	2	0	0	0
0	0	0	1	0

y_c


Results
Model
Image
Hough Transform $A(x_c, y_c)$
Model
Image
Hough Transform $A(x_c, y_c)$

Handling Scale And Rotation



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

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

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

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

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