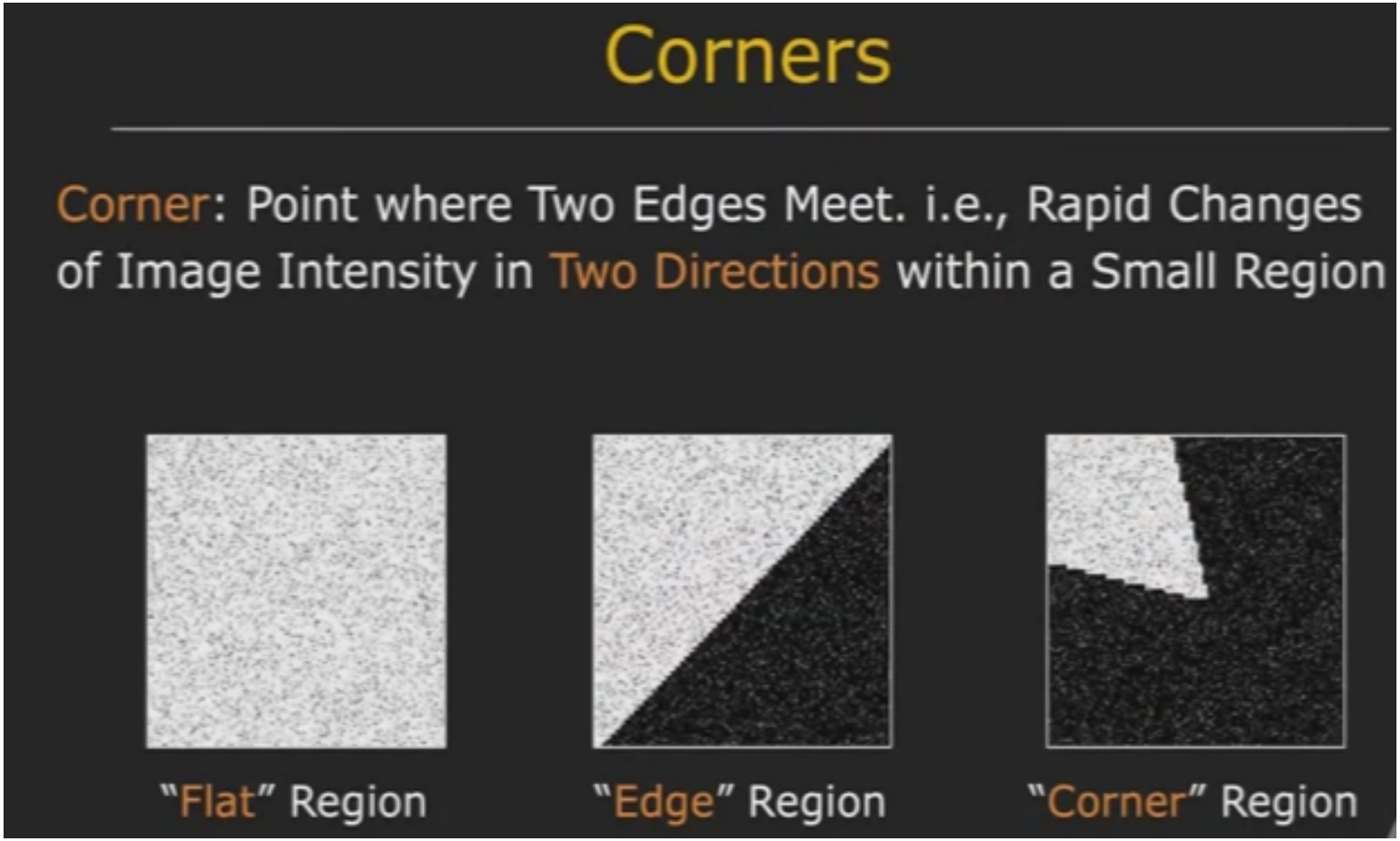
Computer Vision-IT416

Dinesh Naik

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

April 5, 2022

Harris Corner Detection



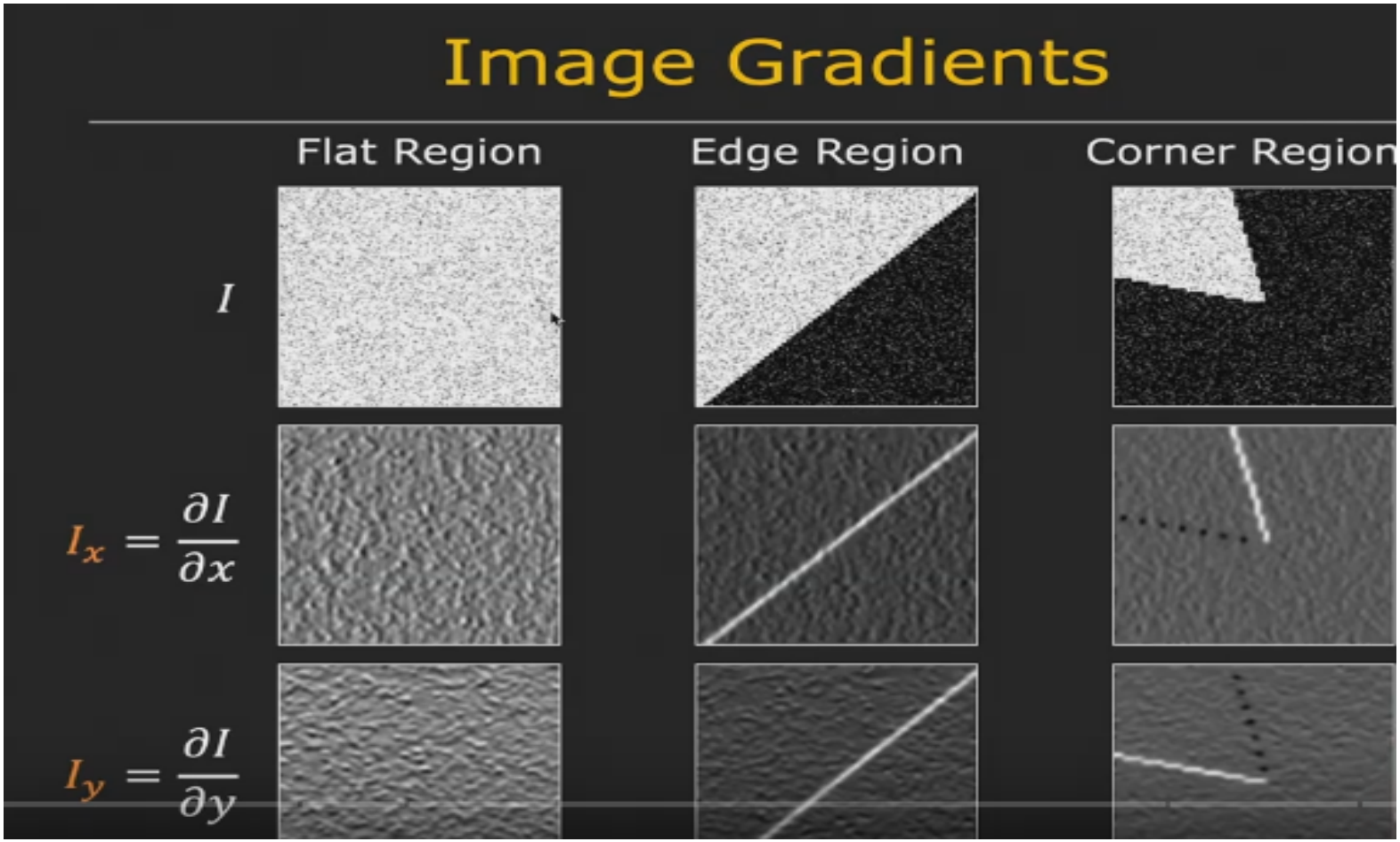
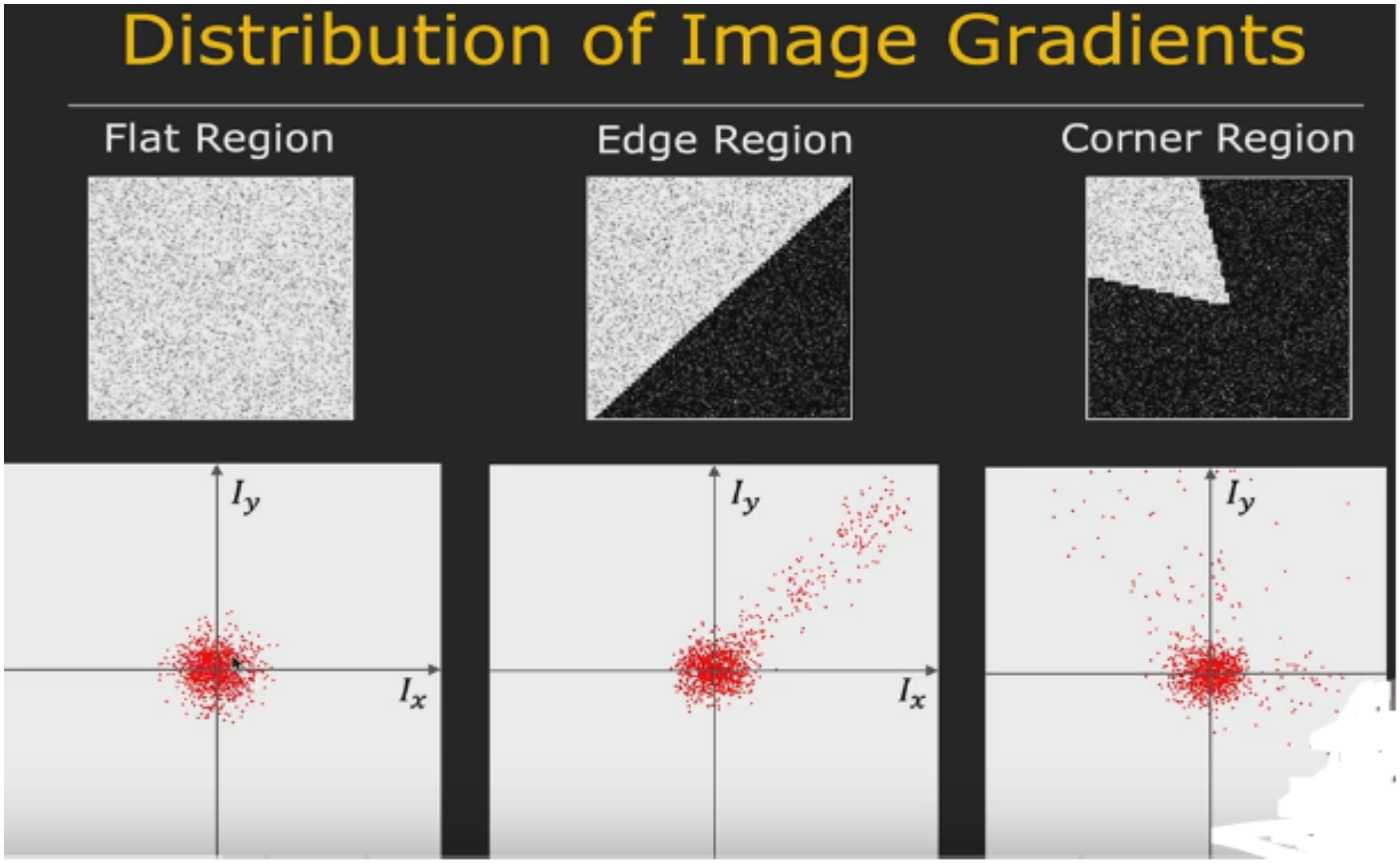
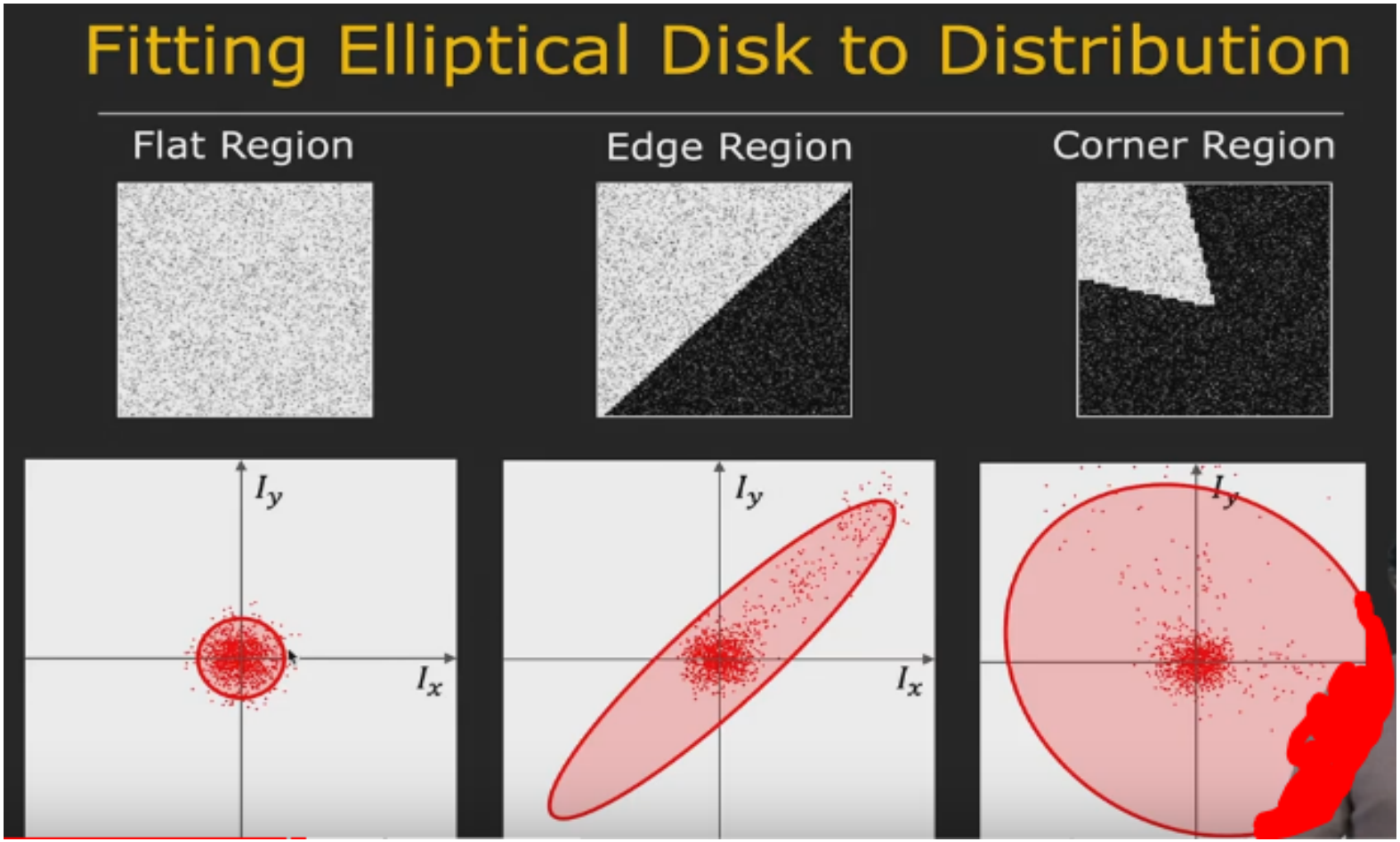
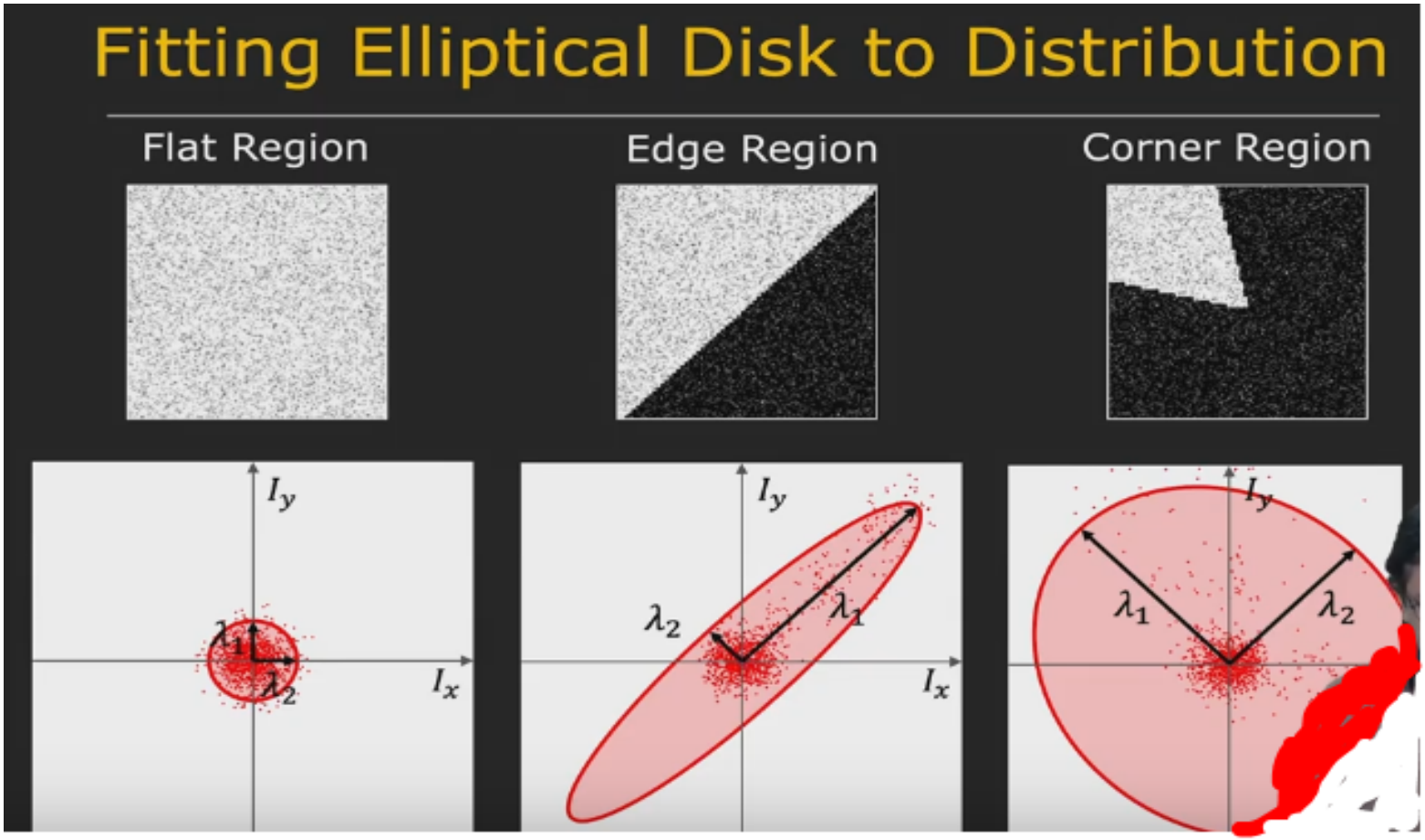
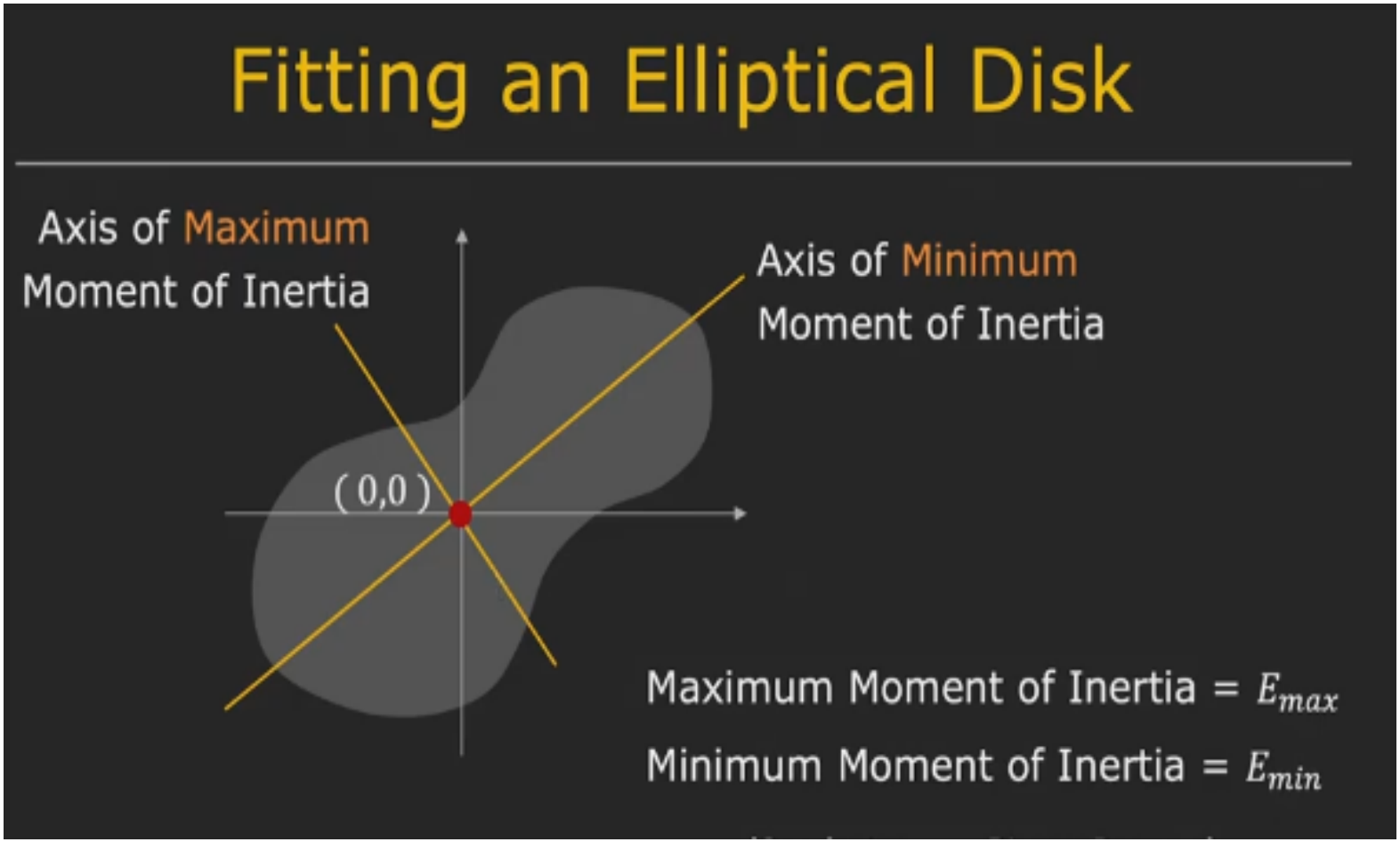


Image Gradients
Flat Region
Edge Region
Corner Region
I
$I_x = \frac{\partial I}{\partial x}$
$I_y = \frac{\partial I}{\partial y}$

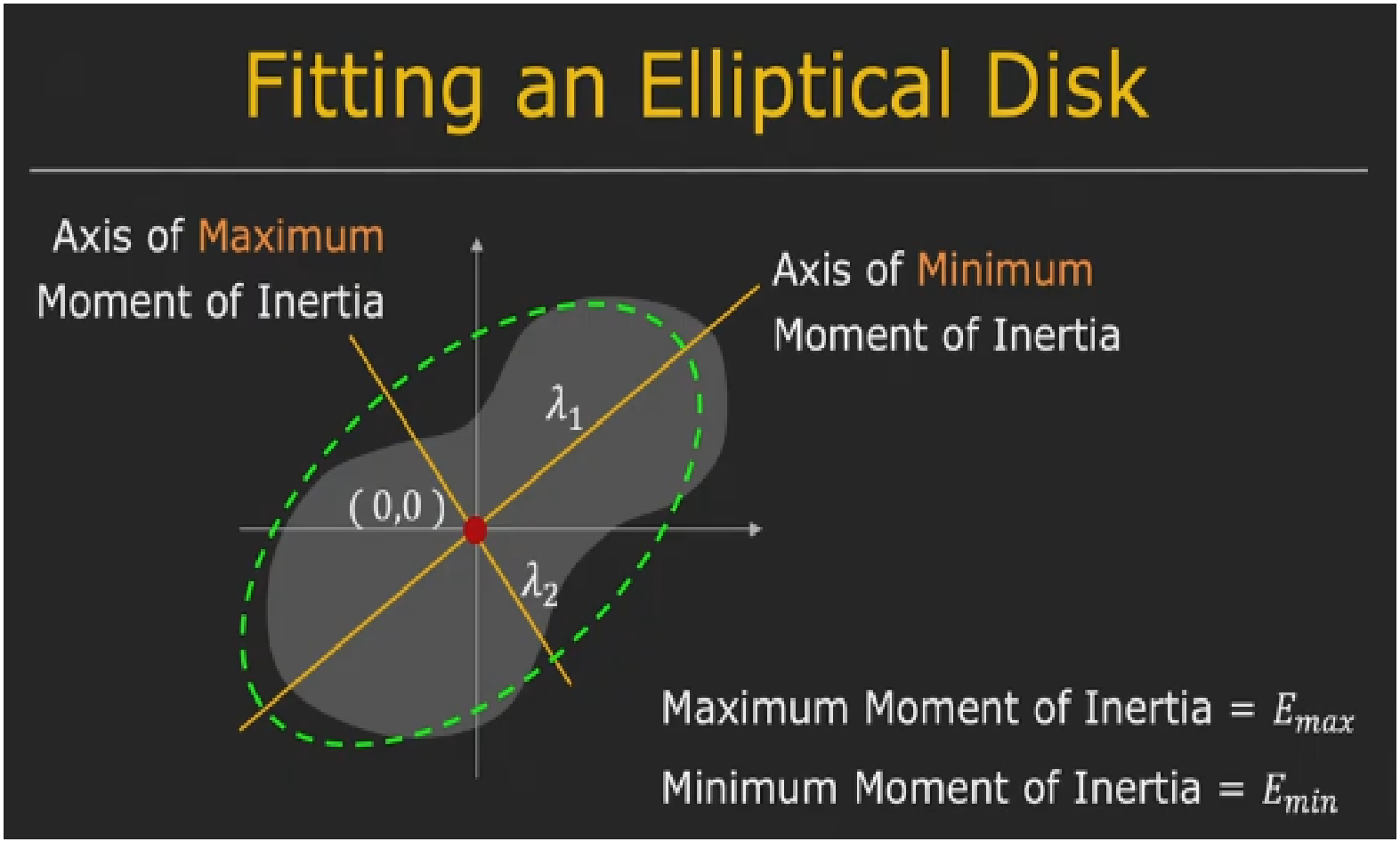

Distribution of Image Gradients
Flat Region
Edge Region
Corner Region
I_y
I_x
I_y
I_x
I_y


Fitting Elliptical Disk to Distribution
Flat Region
Edge Region
Corner Region
I_y
I_x
I_y
I_x
I_y


Fitting Elliptical Disk to Distribution
Flat Region
Edge Region
Corner Region
I_y
I_x
λ_1
λ_2
I_y
I_x
λ_2
λ_1
I_y
λ_1
λ_2


Fitting an Elliptical Disk
Axis of Maximum
Moment of Inertia
Axis of Minimum
Moment of Inertia
(0,0)
Maximum Moment of Inertia = E_{max}
Minimum Moment of Inertia = E_{min}

Fitting an Elliptical Disk



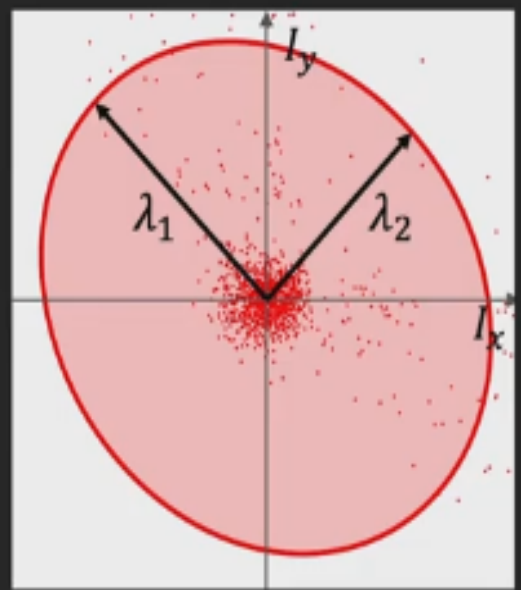
Fitting an Elliptical Disk

Second Moments for a Region:

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

$$c = \sum_{i \in W} \left(I_{y_i}{}^2\right)$$

W: Window centered at pixel



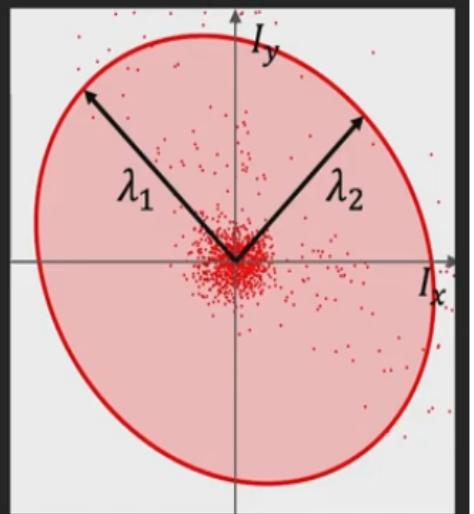
Second Moments for a Region:

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

$$c = \sum_{i \in W} \left(I_{y_i}{}^2\right)$$

W: Window centered at pixel

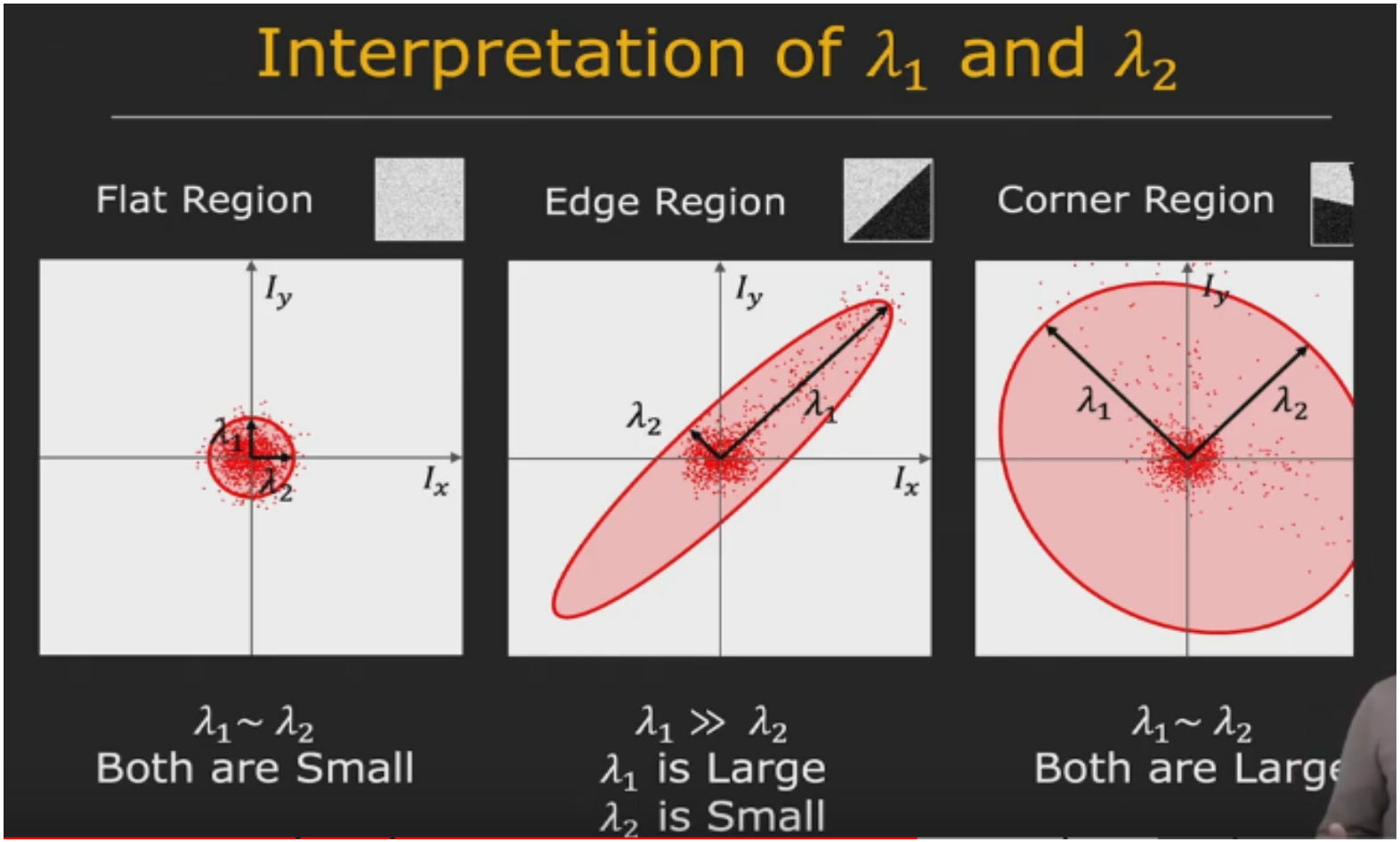
(See lecture on Binary Images)



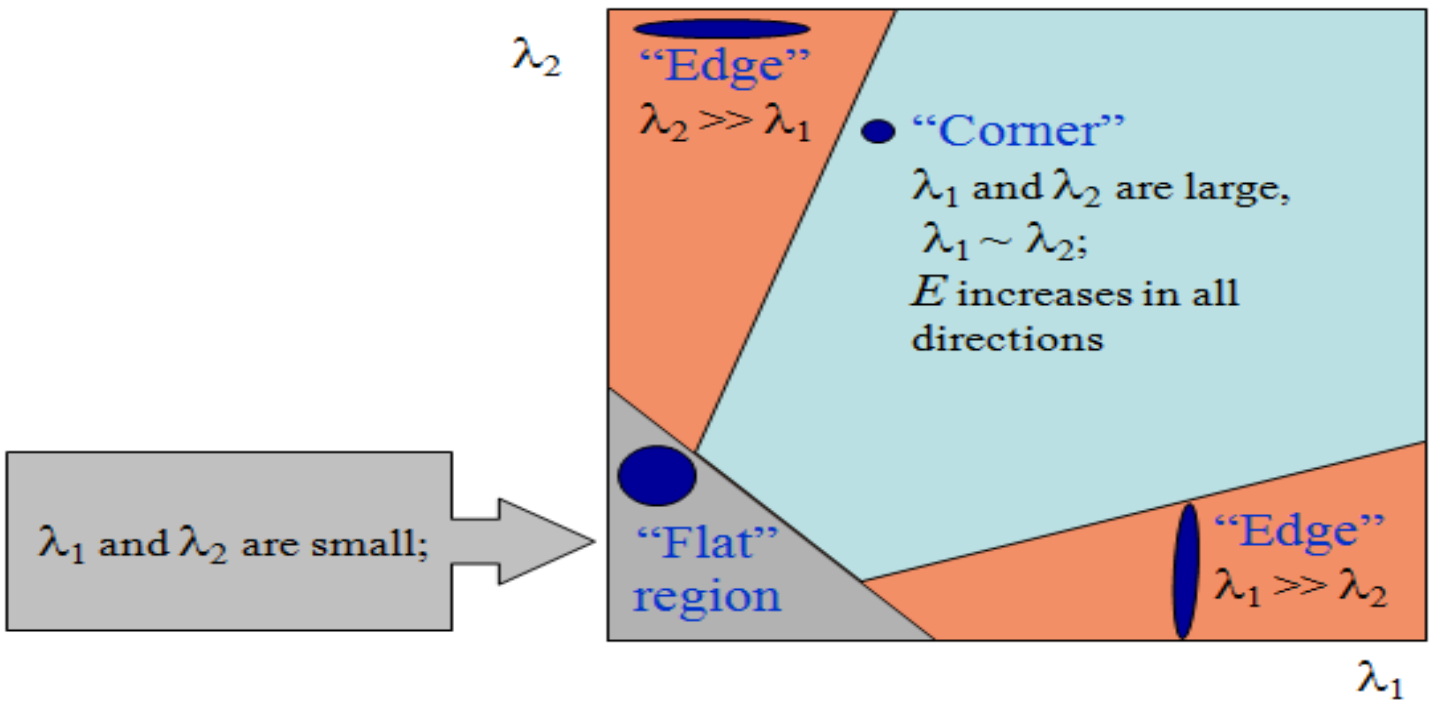
Ellipse Axes Lengths:

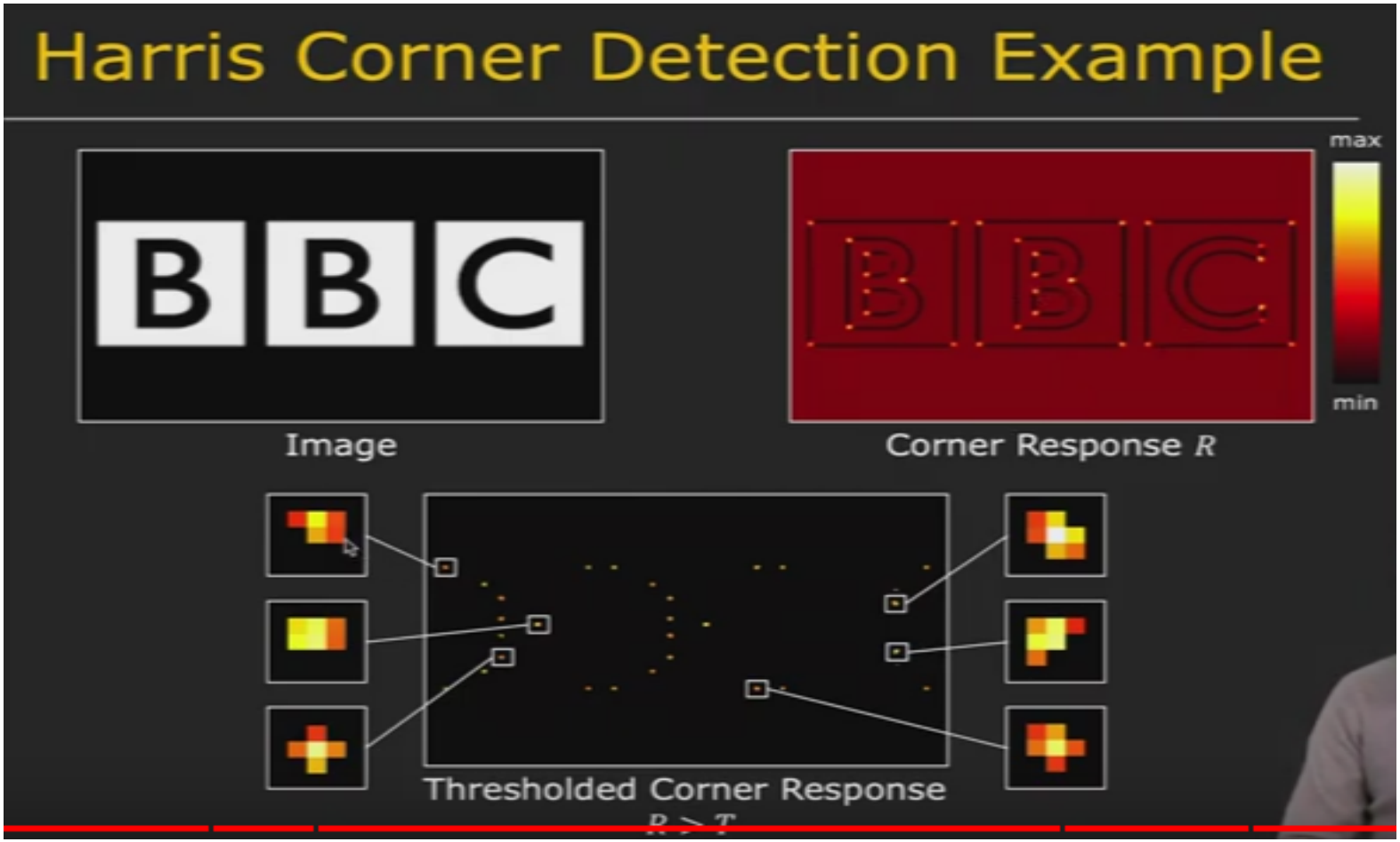
$$\lambda_1 = E_{max} = \frac{1}{2}\left[a + c + \sqrt{b^2 + (a-c)^2}\right]$$

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


Interpretation of λ_1 and λ_2
Flat Region
Edge Region
Corner Region
I_y
I_x
λ_1
λ_2
$\lambda_1 \sim \lambda_2$
Both are Small
$\lambda_1 \gg \lambda_2$
λ_1 is Large
λ_2 is Small
$\lambda_1 \sim \lambda_2$
Both are Large

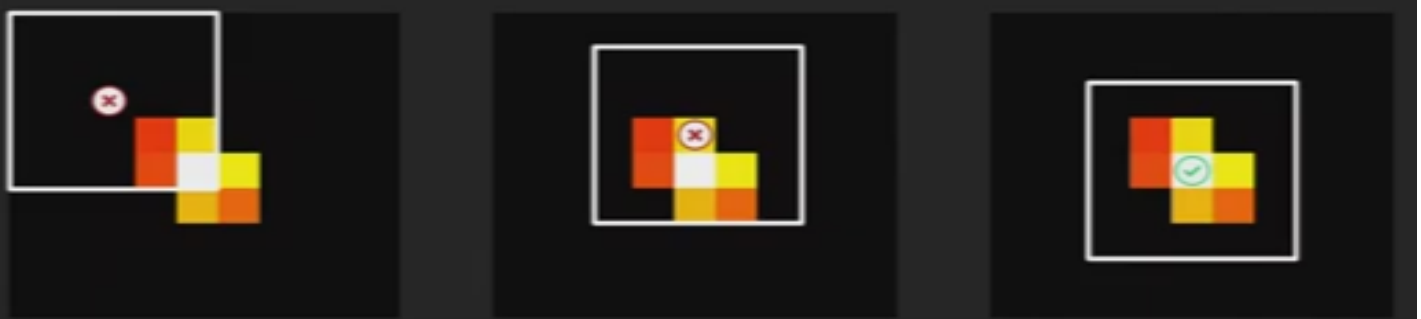
Harris Detector: Mathematics




Harris Corner Detection Example
max
min
Image
Corner Response R
Thresholded Corner Response
$R > T$

Non-Maximal Suppression

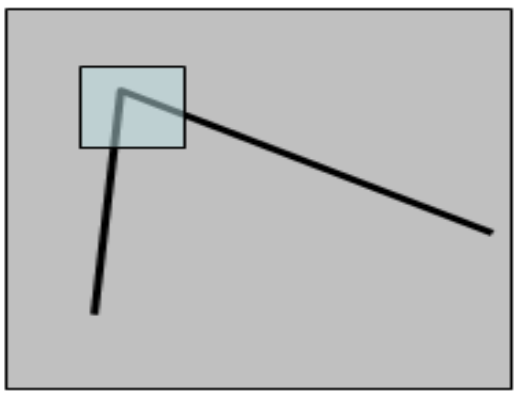
1. Slide a window of size k over the image.
2. At each position, if the pixel at the center is the maximum value within the window, label it as positive (retain it). Else label it as negative (suppress it).



Suppress | Suppress | Retain

The Basic Idea

- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give *a large change* in intensity



Harris Detector: Mathematics

Measure of corner response:

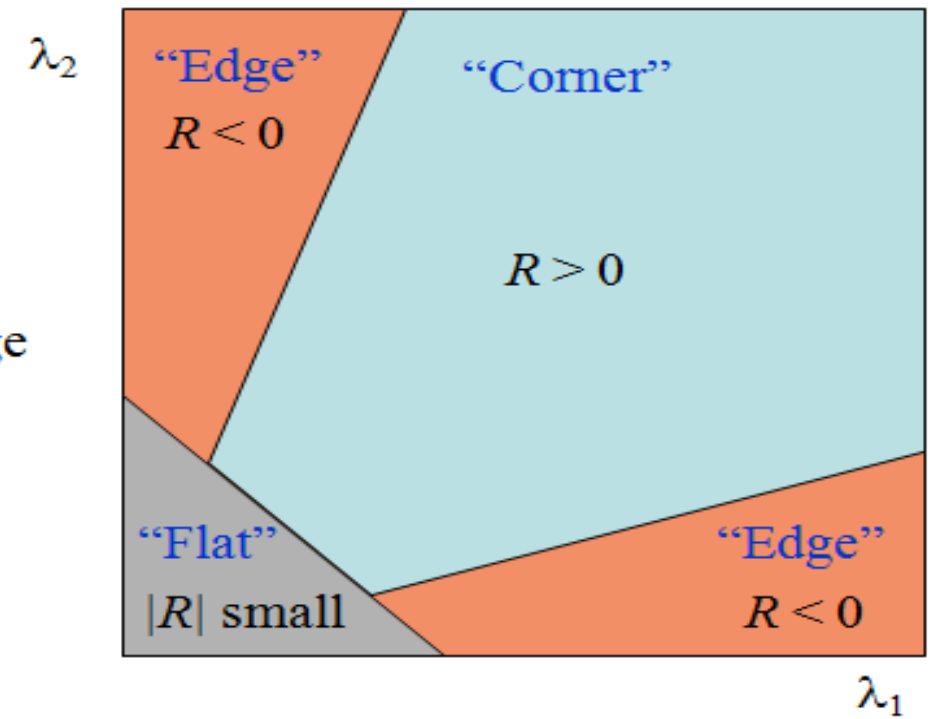
$$R = \det M - k\left(\text{trace}\, M\right)^2$$

$$\det M = \lambda_1 \lambda_2$$
$$\text{trace}\, M = \lambda_1 + \lambda_2$$

(k – empirical constant, $k = 0.04$-0.06)

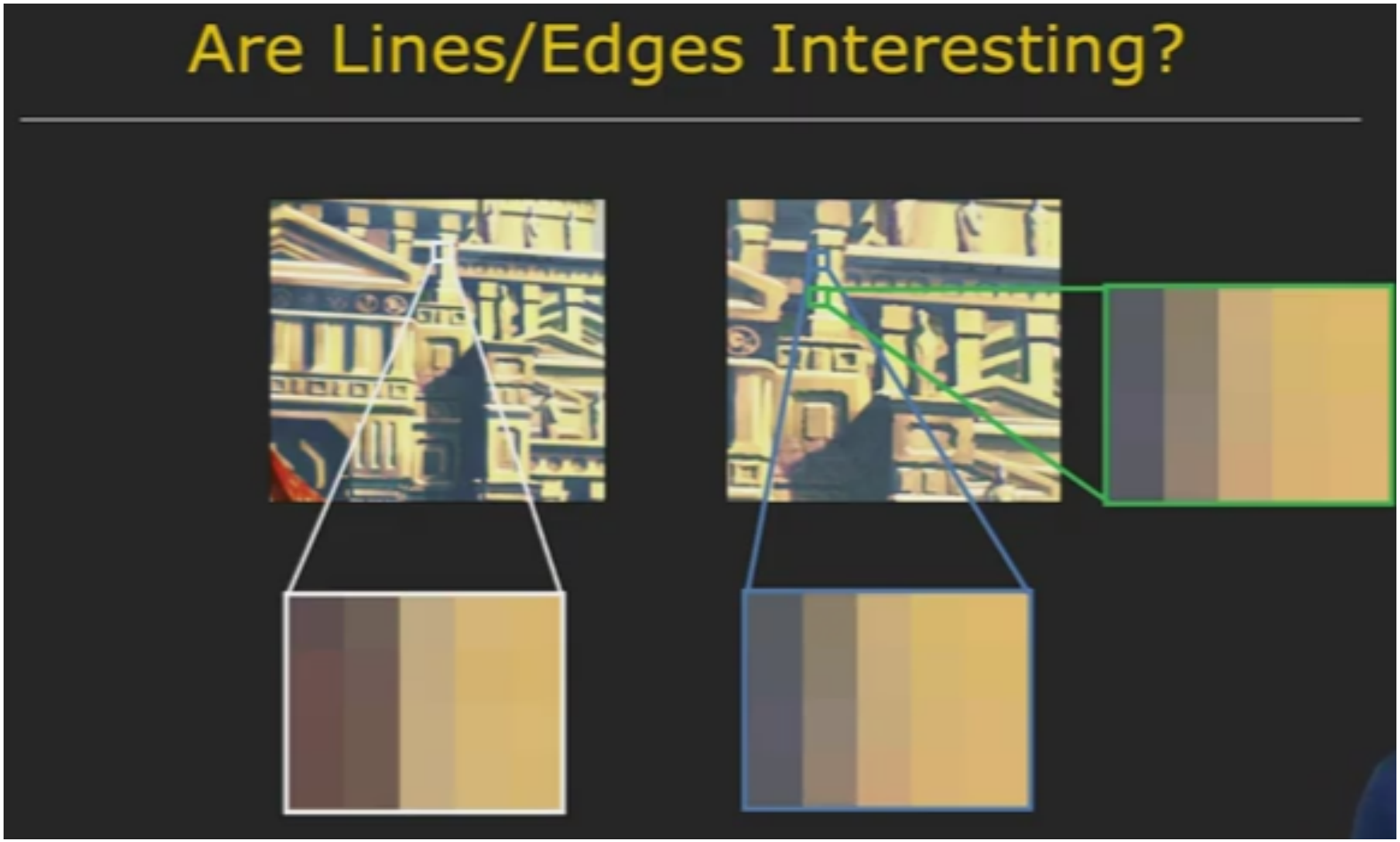
Harris Detector: Mathematics

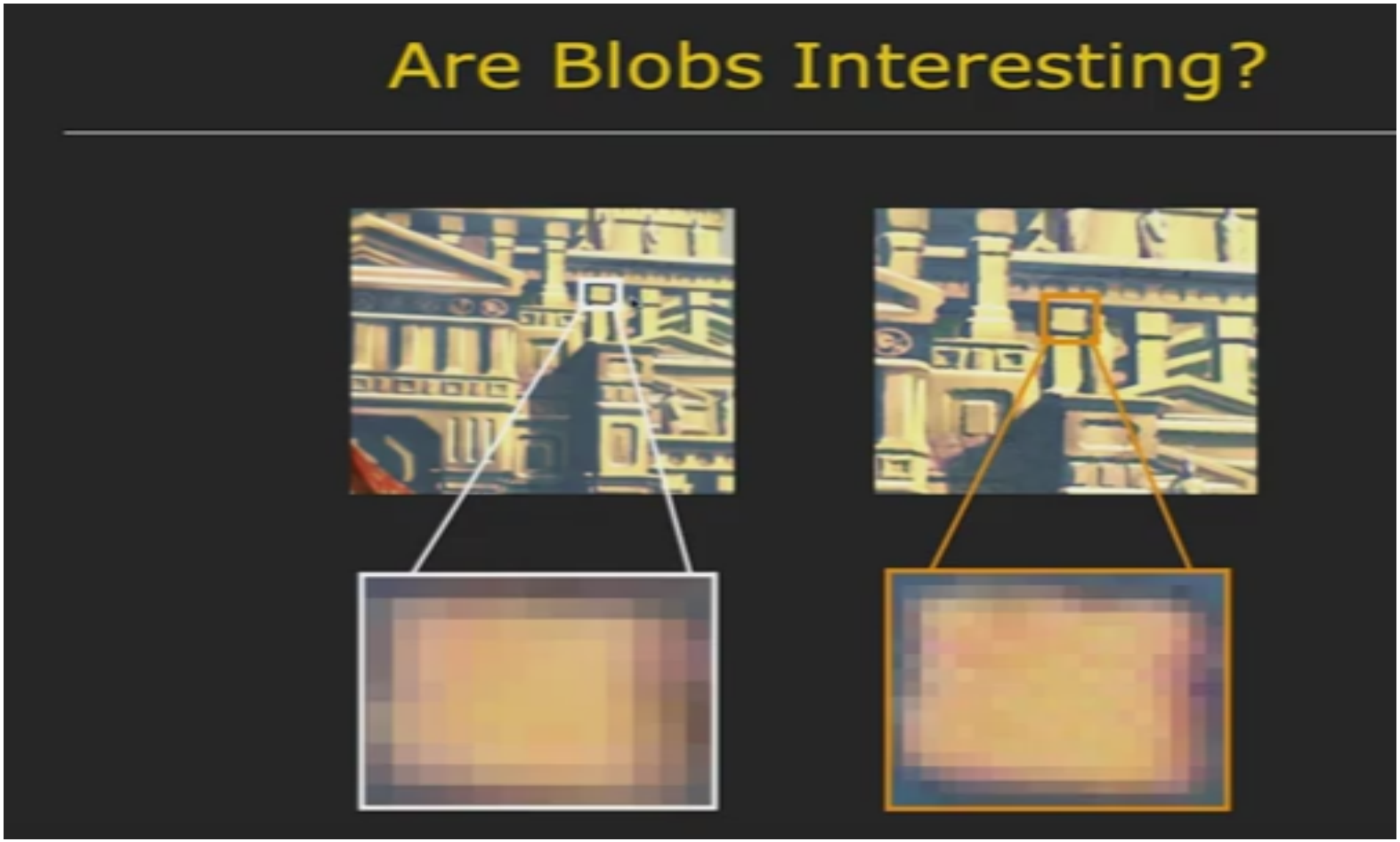
- R depends only on eigenvalues of M
- R is large for a corner
- R is negative with large magnitude for an edge
- $|R|$ is small for a flat region



What is an Interesting Point/Feature?

- Has rich image content (brightness variation, color variation, etc.) within the local window
- Has well-defined representation (signature) for matching/comparing with other points
- Has a well-defined position in the image
- Should be invariant to image rotation and scaling
- Should be insensitive to lighting changes

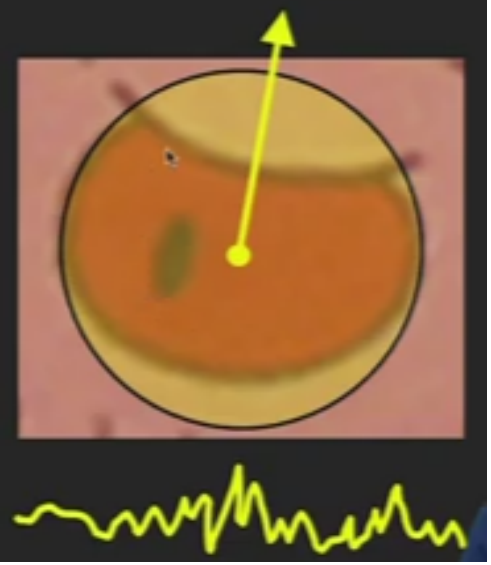

Are Lines/Edges Interesting?


Are Blobs Interesting?

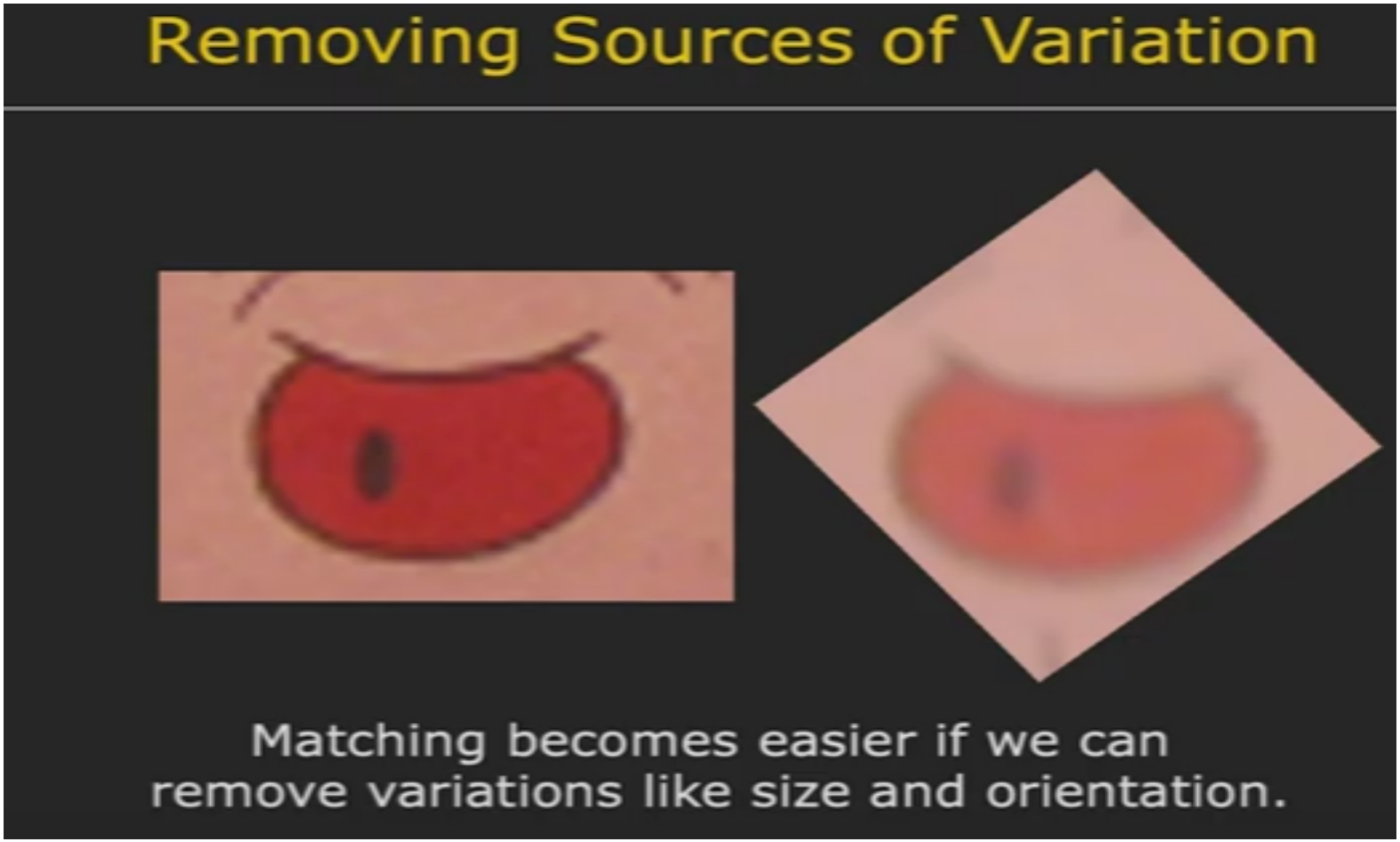
Blobs as Interest Points

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

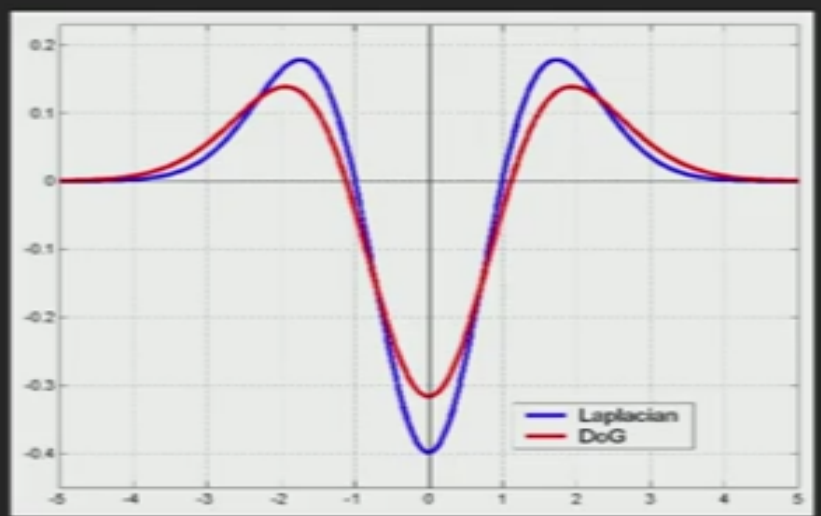



Raw Images are Hard to Match
Different size, orientation, lighting, brightness, etc.

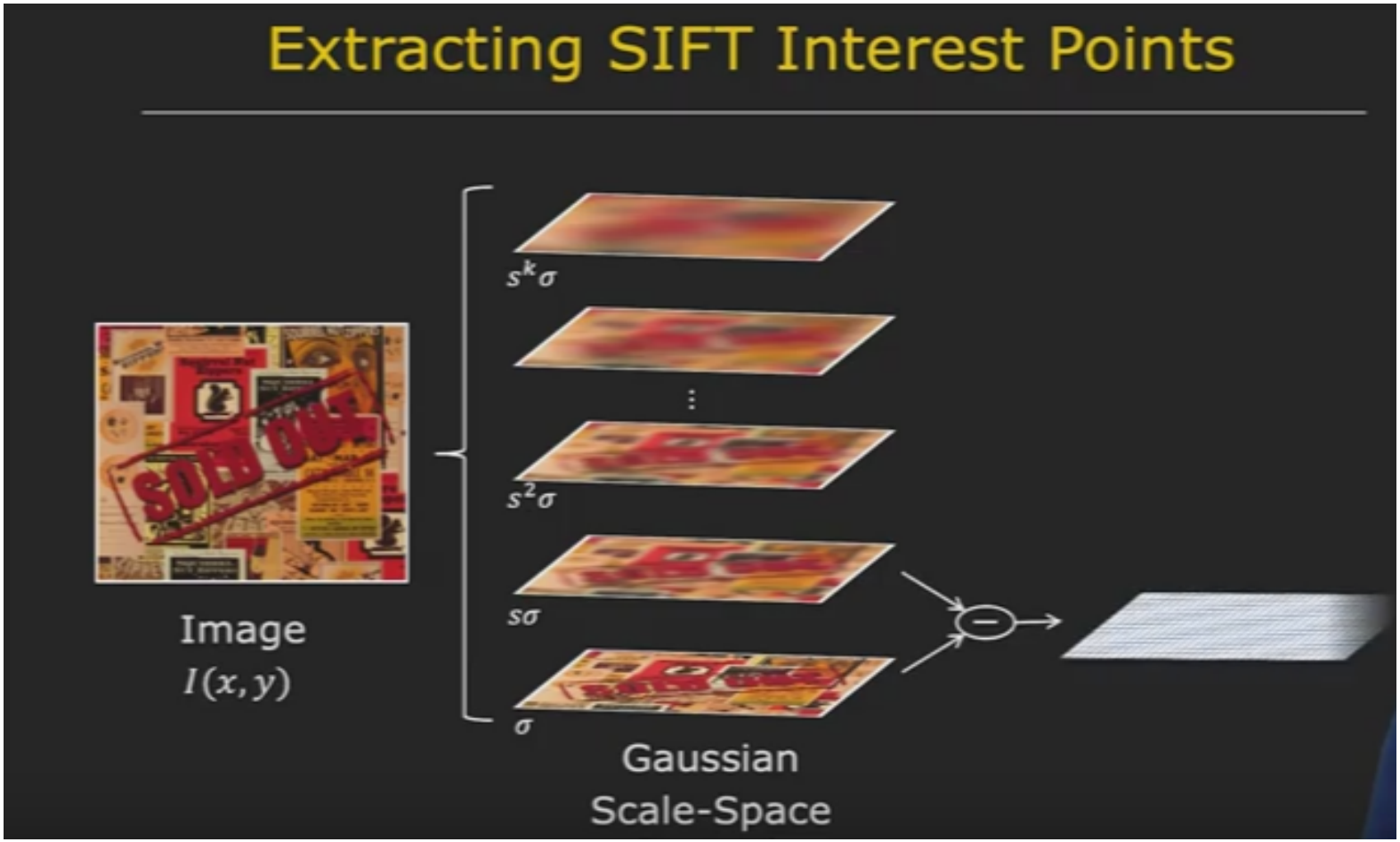

Removing Sources of Variation
Matching becomes easier if we can
remove variations like size and orientation.

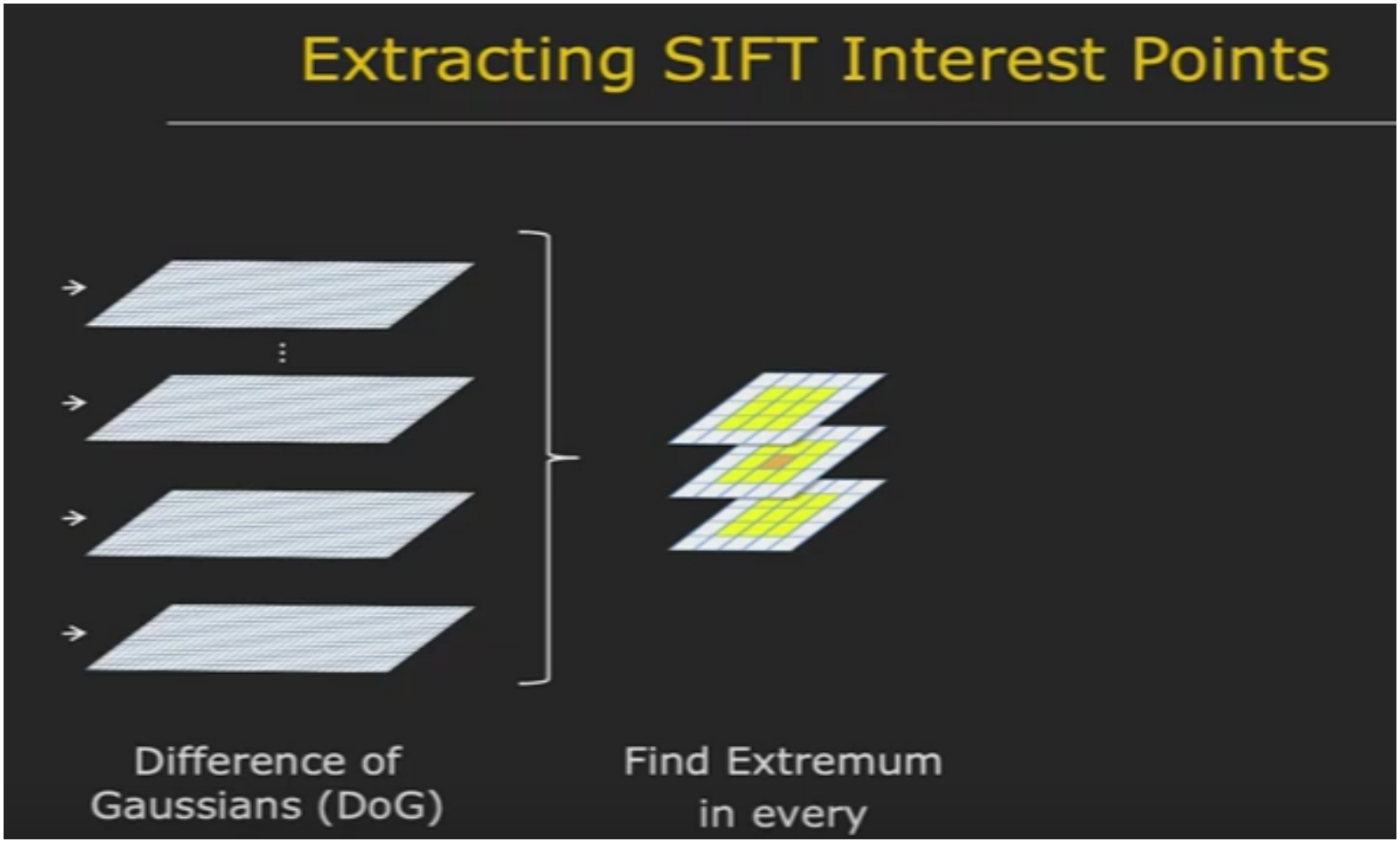
Fast NLoG Approximation: DoG

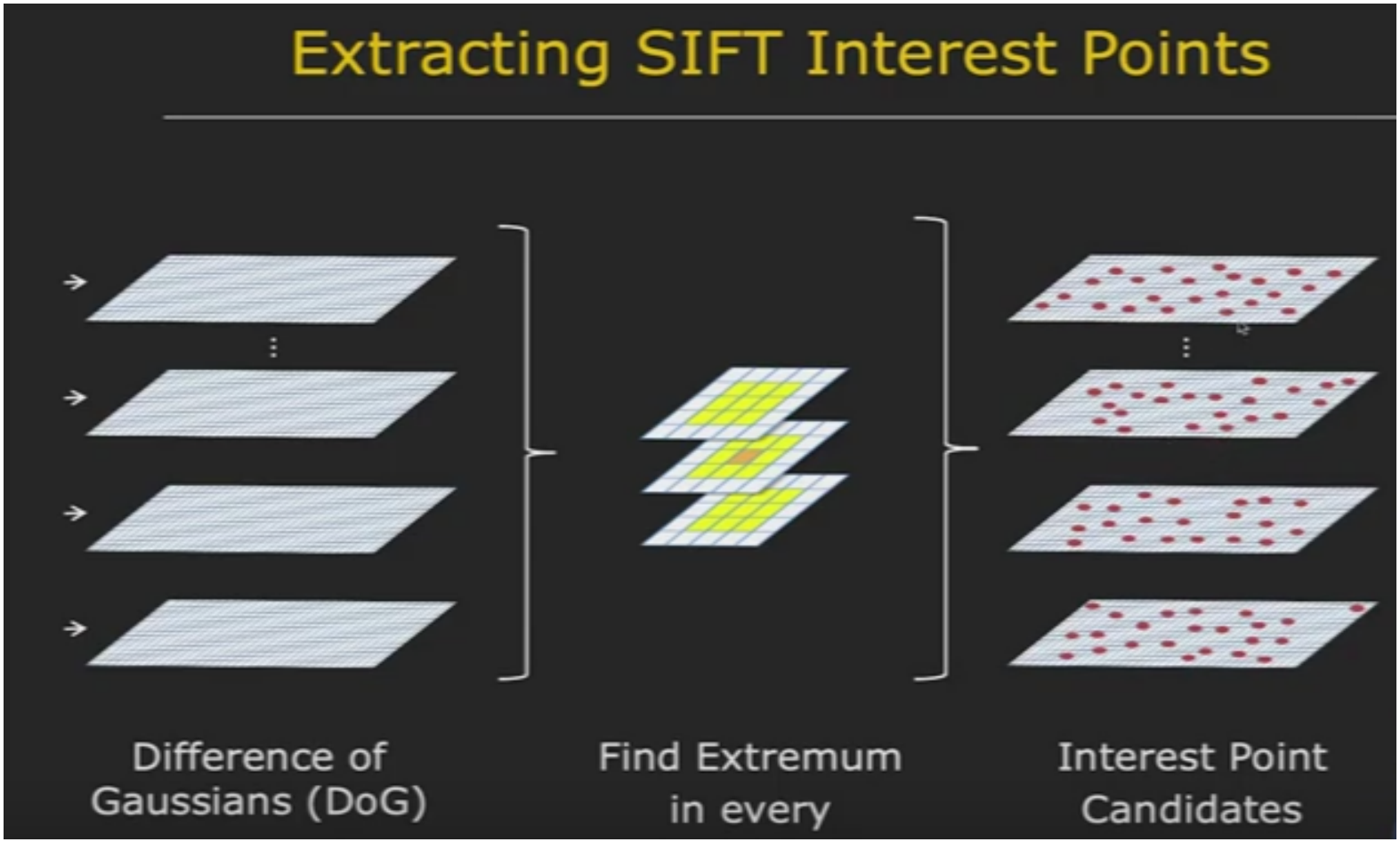
Difference of Gaussian (DoG) $= (n_{s\sigma} - n_{\sigma}) \approx (s-1)\underbrace{\sigma^2 \nabla^2 n_{\sigma}}_{\text{NLoG}}$

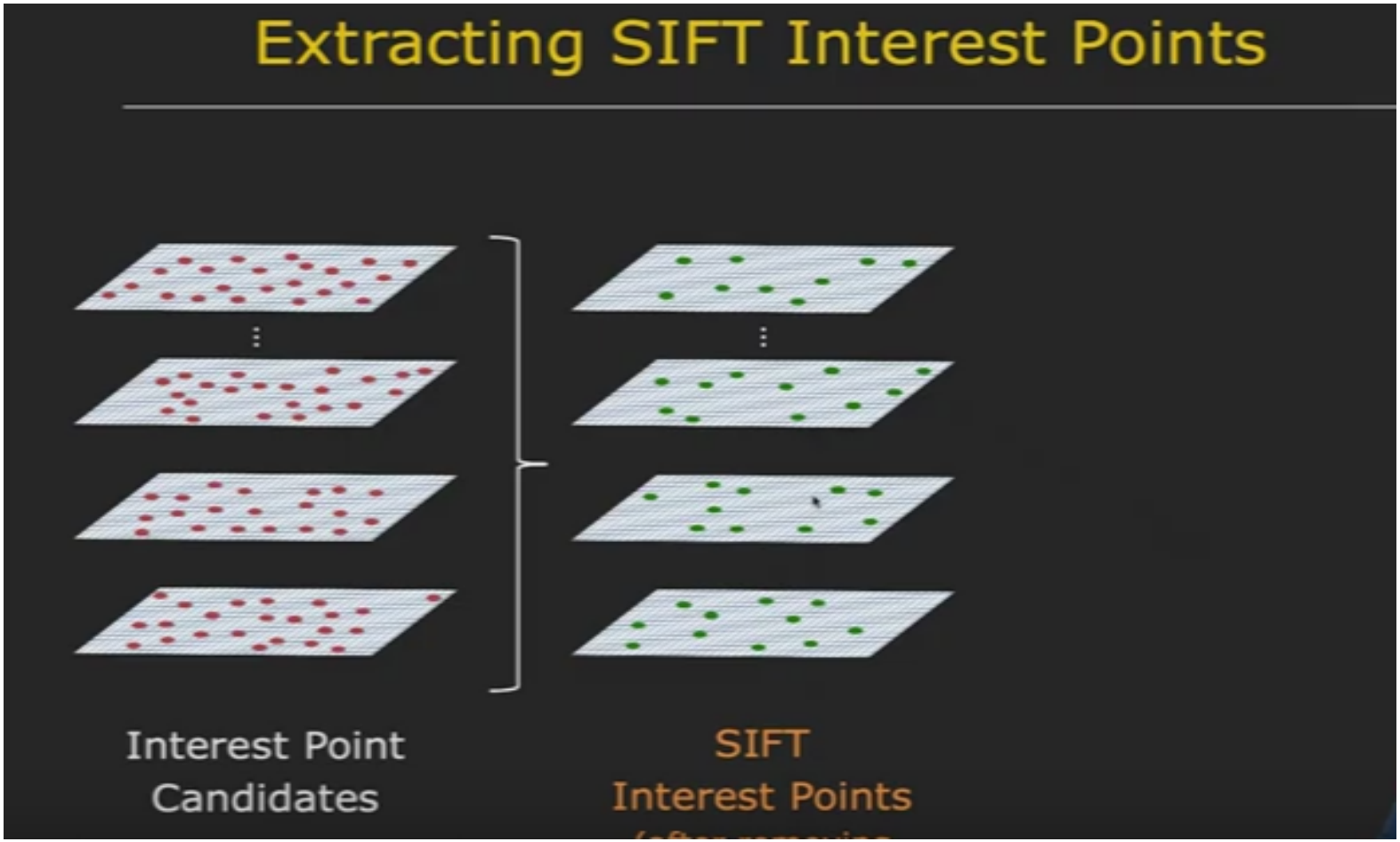


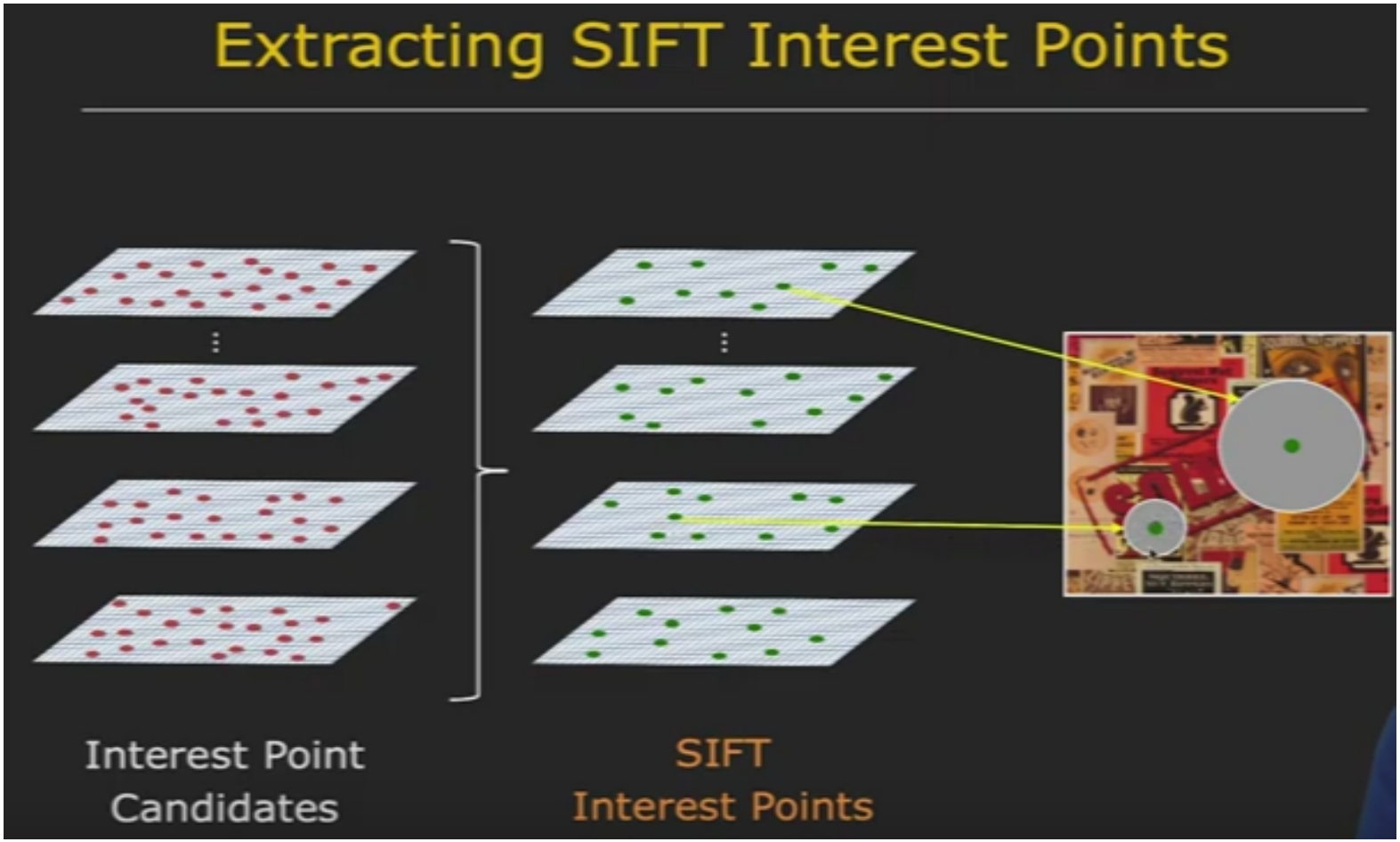
$\text{DoG} \approx (s-1)\ \text{NLoG}$

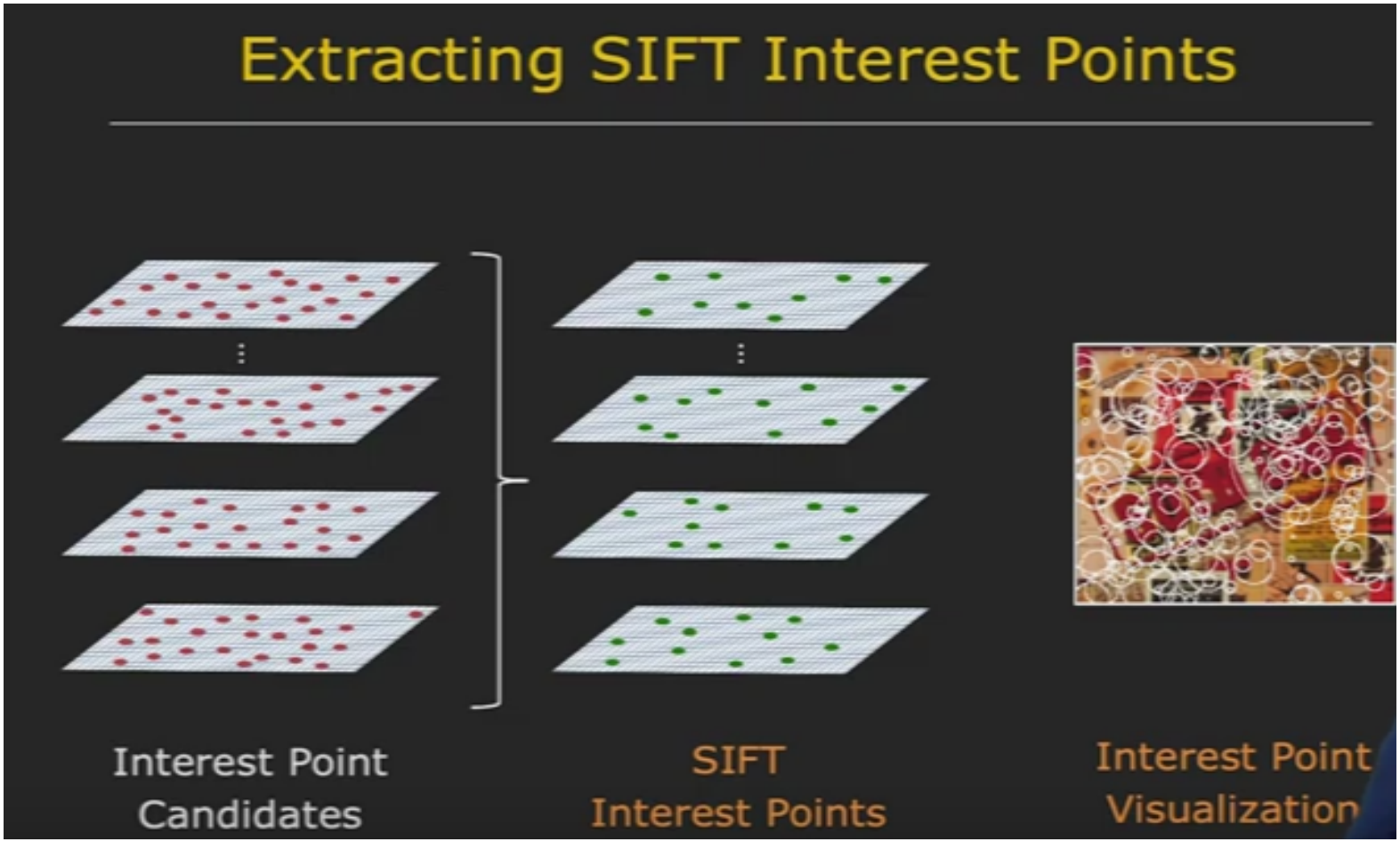

Extracting SIFT Interest Points
$s^k\sigma$
$s^2\sigma$
$s\sigma$
σ
Image
$I(x,y)$
Gaussian
Scale-Space


Extracting SIFT Interest Points
Difference of Gaussians (DoG)
Find Extremum in every


Extracting SIFT Interest Points
Difference of Gaussians (DoG)
Find Extremum in every
Interest Point Candidates


Extracting SIFT Interest Points
Interest Point Candidates
SIFT Interest Points


Extracting SIFT Interest Points
Interest Point Candidates
SIFT Interest Points


Extracting SIFT Interest Points
Interest Point Candidates
SIFT Interest Points
Interest Point Visualization

Computing the Principal Orientation

Use the histogram of gradient directions

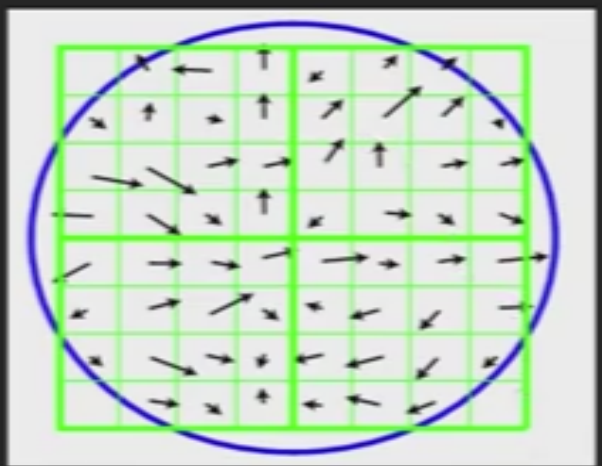
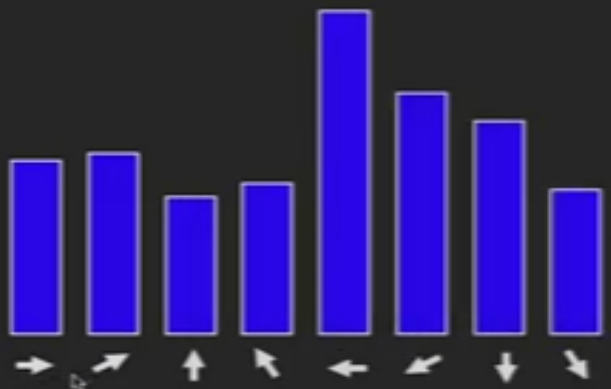


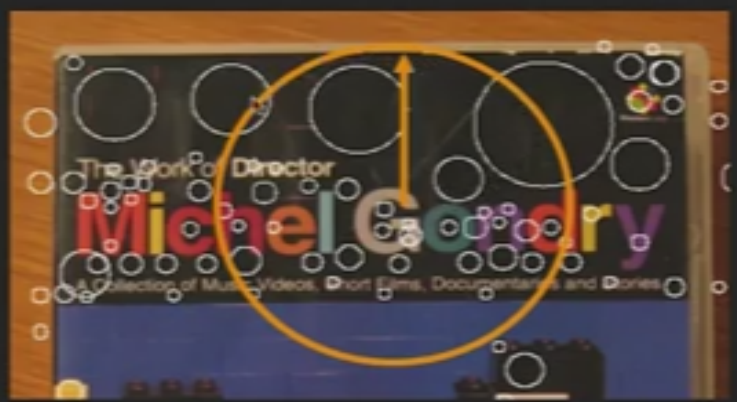
Image gradient directions

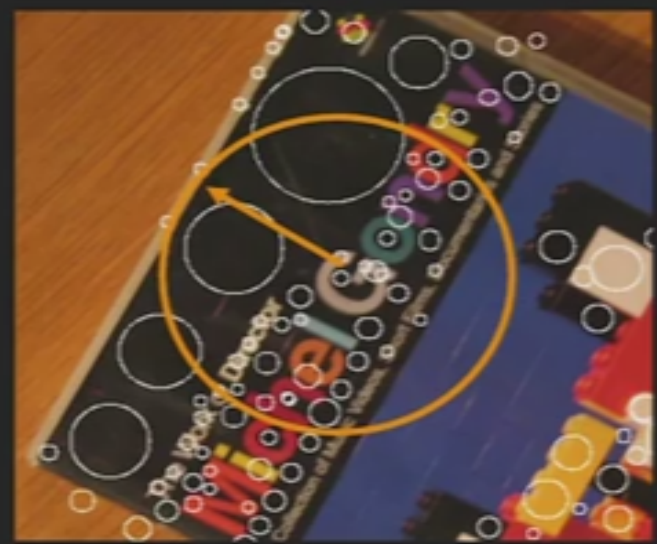


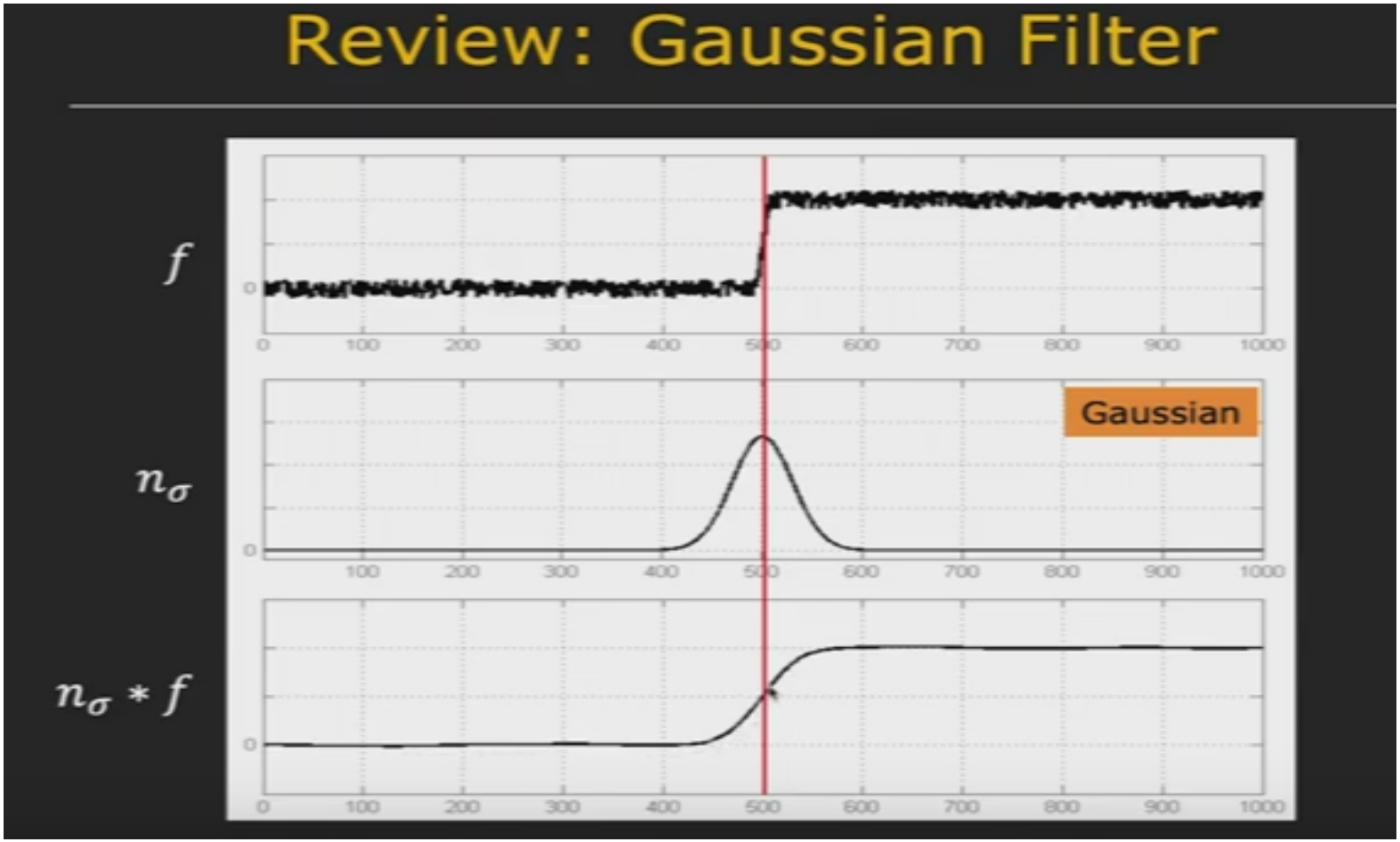
$$\theta = \tan^{-1}\left(\frac{\partial I}{\partial y} / \frac{\partial I}{\partial x}\right)$$

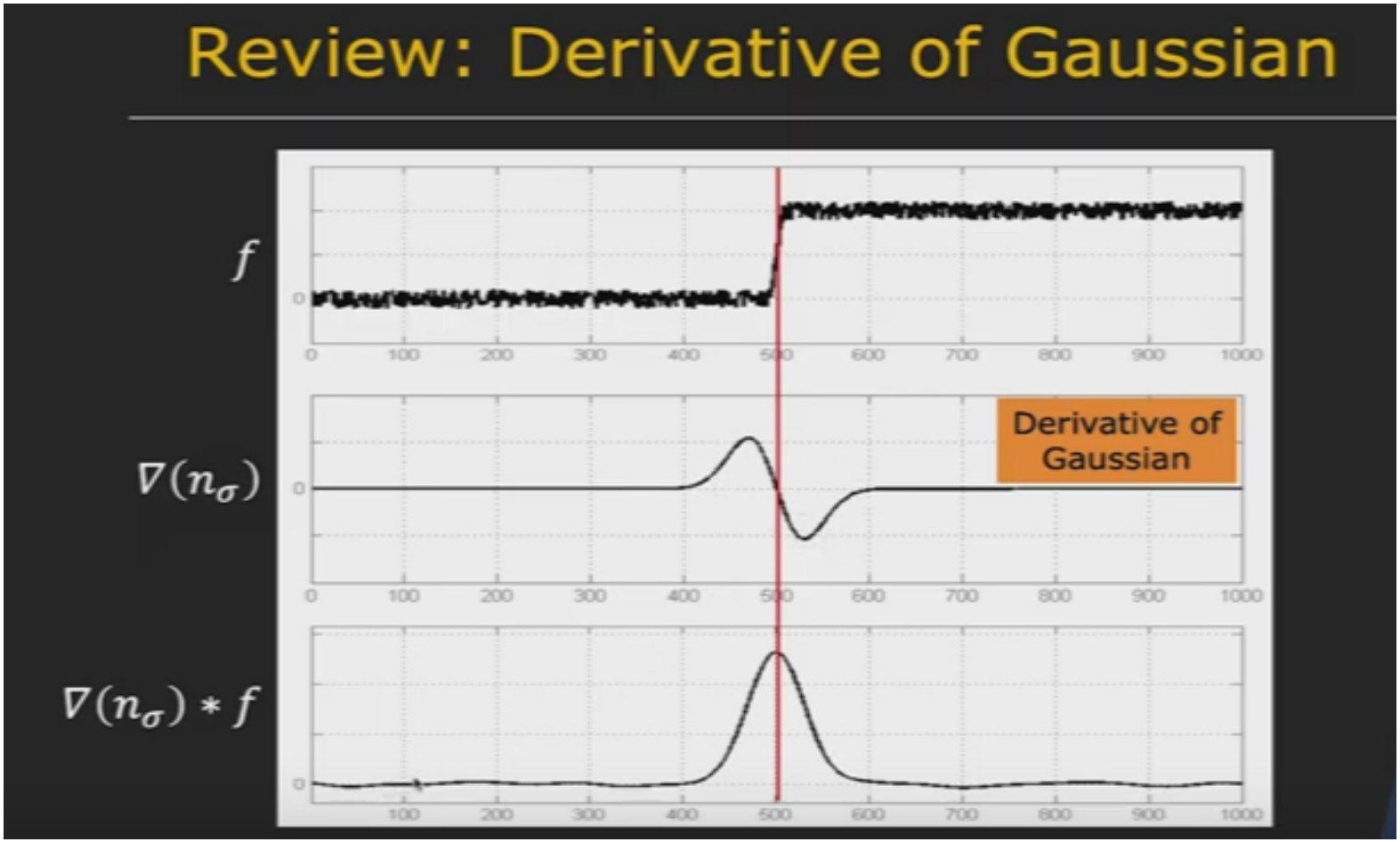
SIFT Rotation Invariance

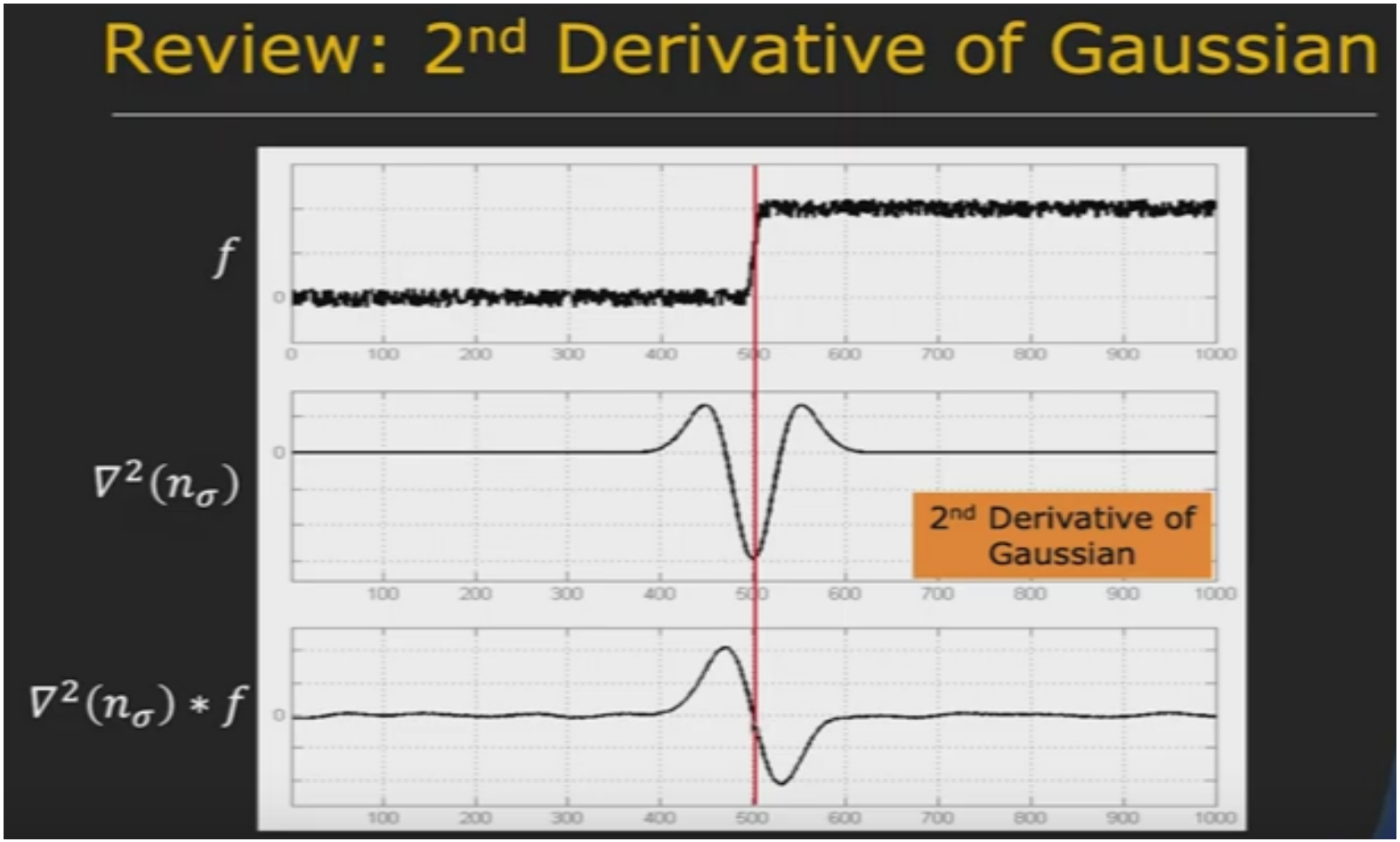
Use the principal orientation to undo rotation

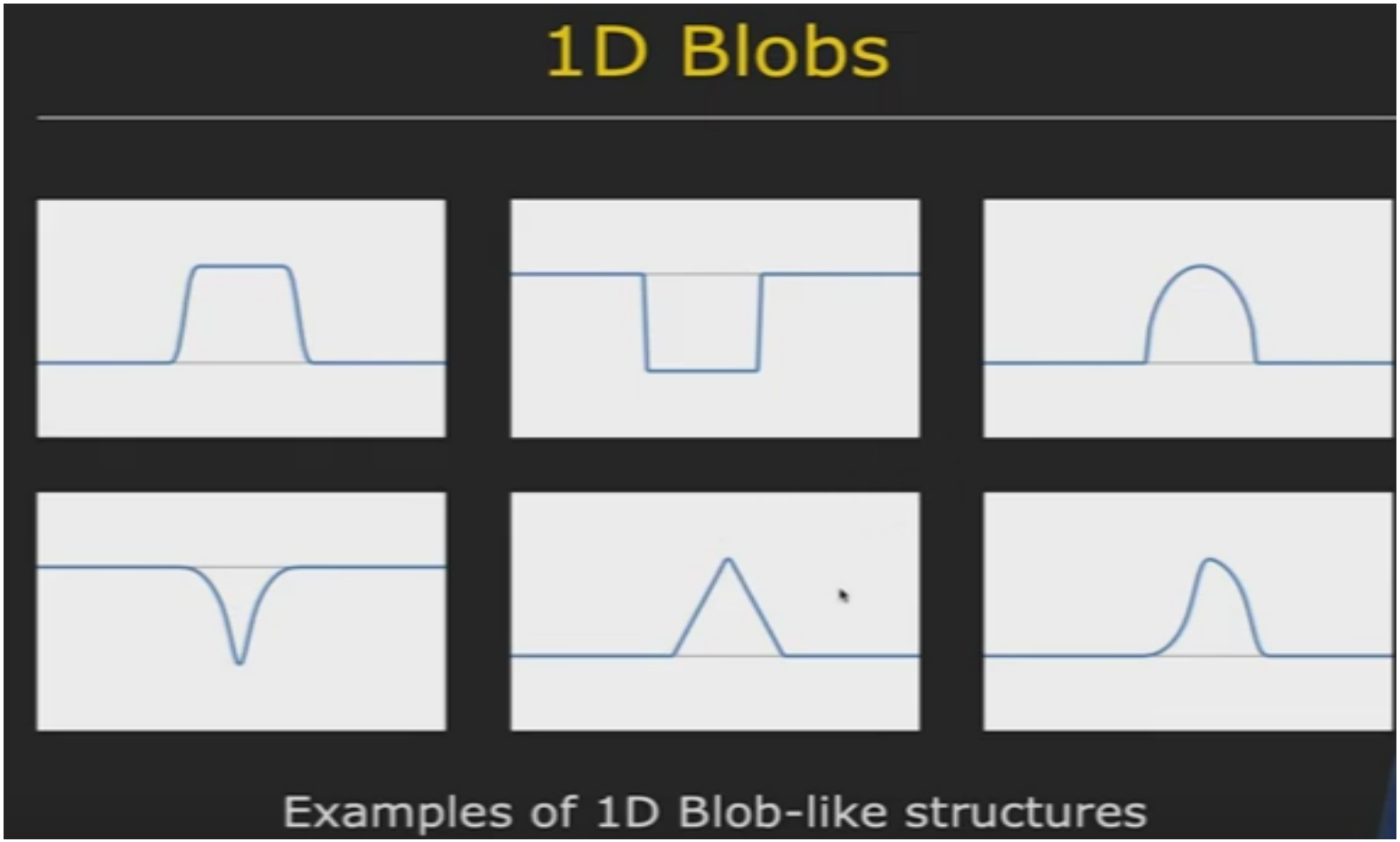


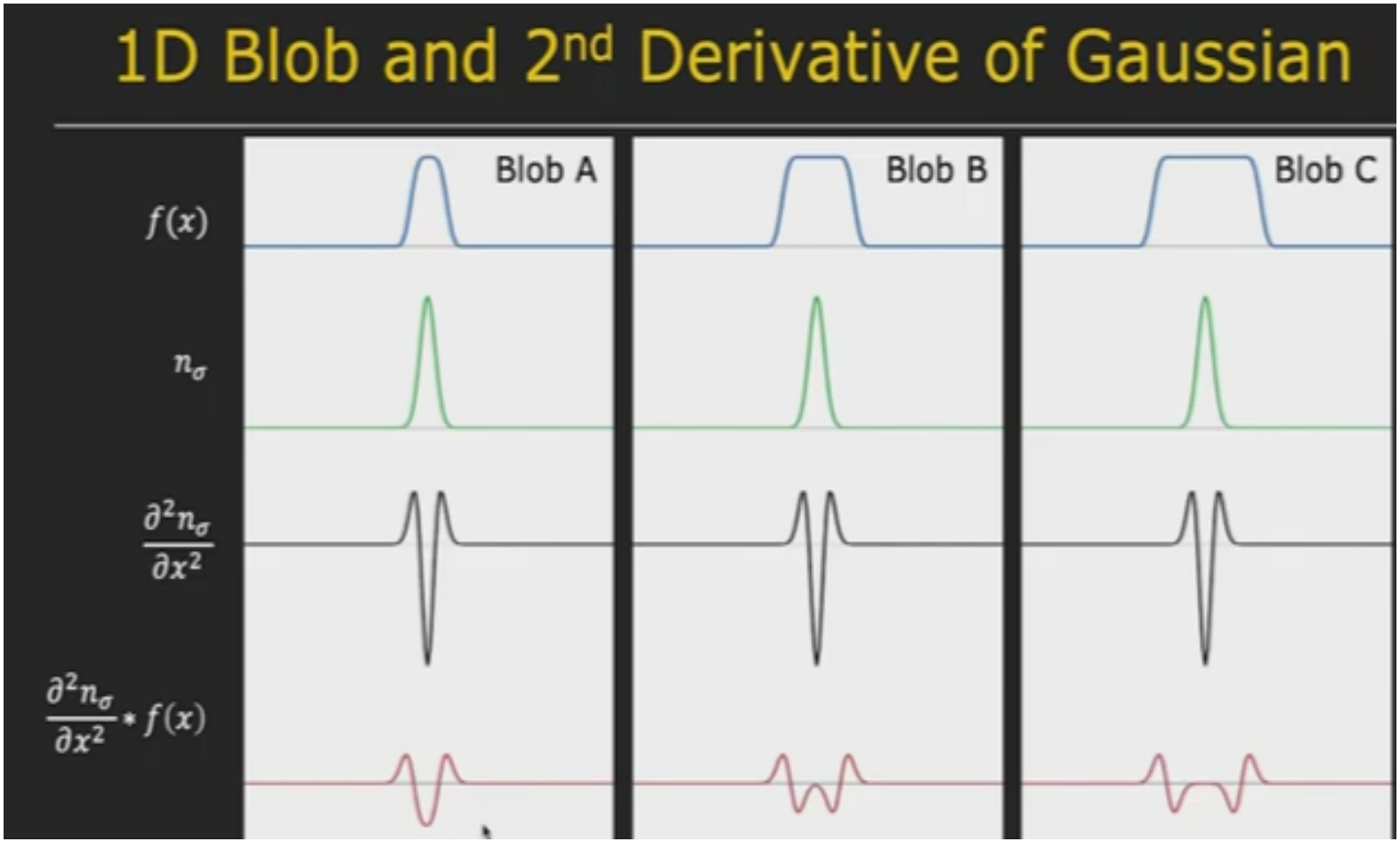


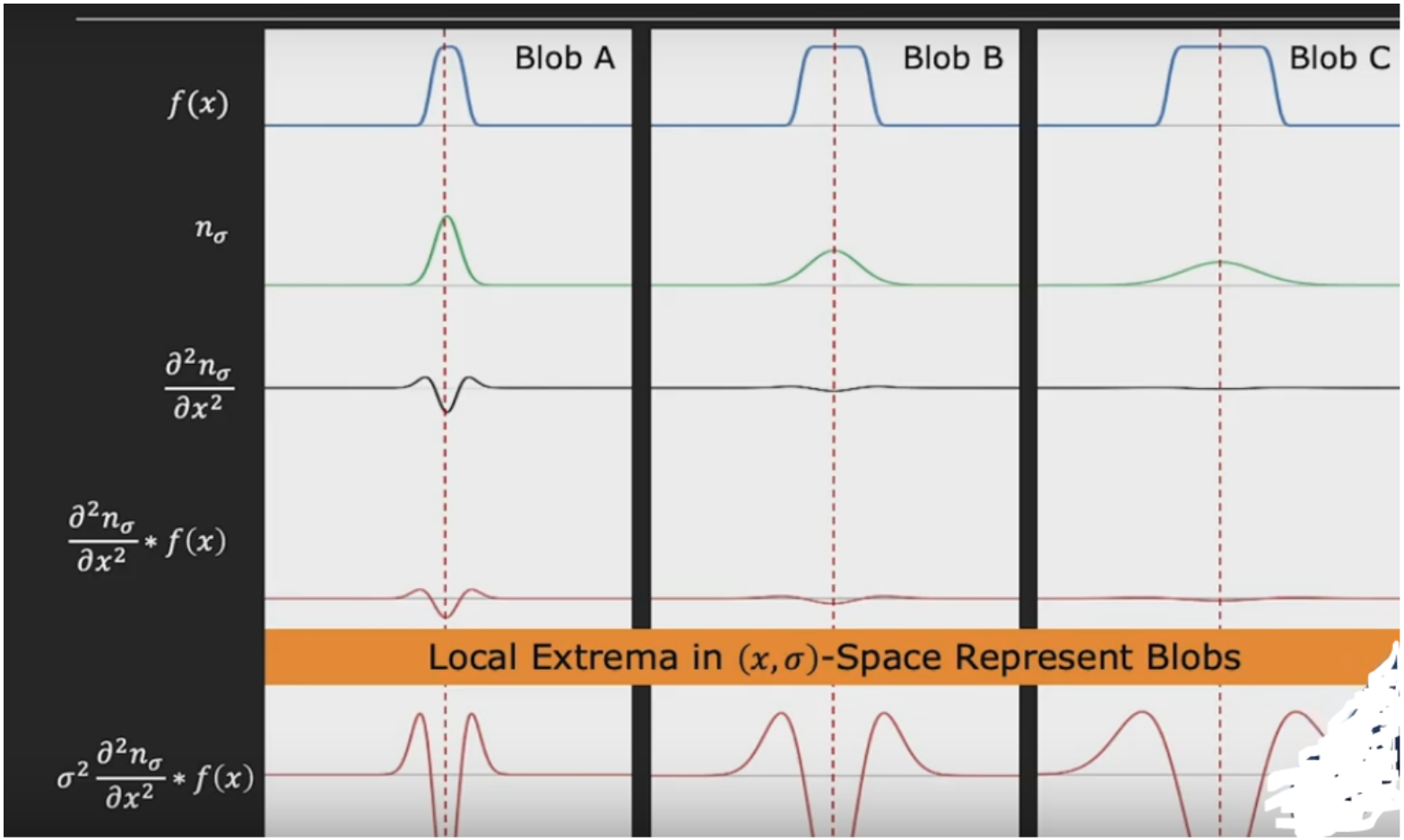

Review: Gaussian Filter
f
n_σ
$n_\sigma * f$
Gaussian
0
100
200
300
400
500
600
700
800
900
1000


Review: Derivative of Gaussian
f
$\nabla(n_\sigma)$
$\nabla(n_\sigma) * f$
Derivative of Gaussian
0
0 100 200 300 400 500 600 700 800 900 1000


Review: 2nd Derivative of Gaussian
f
$\nabla^2(n_\sigma)$
$\nabla^2(n_\sigma) * f$
2nd Derivative of Gaussian
0
100
200
300
400
500
600
700
800
900
1000


1D Blobs
Examples of 1D Blob-like structures


1D Blob and 2nd Derivative of Gaussian
Blob A
Blob B
Blob C
$f(x)$
n_σ
$\frac{\partial^2 n_\sigma}{\partial x^2}$
$\frac{\partial^2 n_\sigma}{\partial x^2} * f(x)$


Blob A
Blob B
Blob C
$f(x)$
n_σ
$\frac{\partial^2 n_\sigma}{\partial x^2}$
$\frac{\partial^2 n_\sigma}{\partial x^2} * f(x)$
Local Extrema in (x, σ)-Space Represent Blobs
$\sigma^2 \frac{\partial^2 n_\sigma}{\partial x^2} * f(x)$

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 $(\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_k)$.

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

x^*: Blob Position

σ^*: Characteristic Scale (Blob Size)

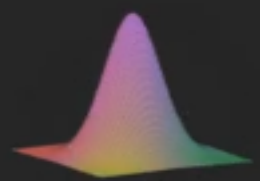
2D Blob Detector

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

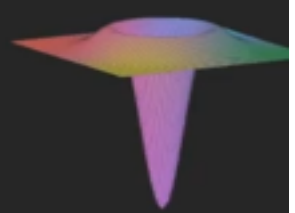


n_σ

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.

2D Blob Detection Summary

Given an image $I(x, y)$

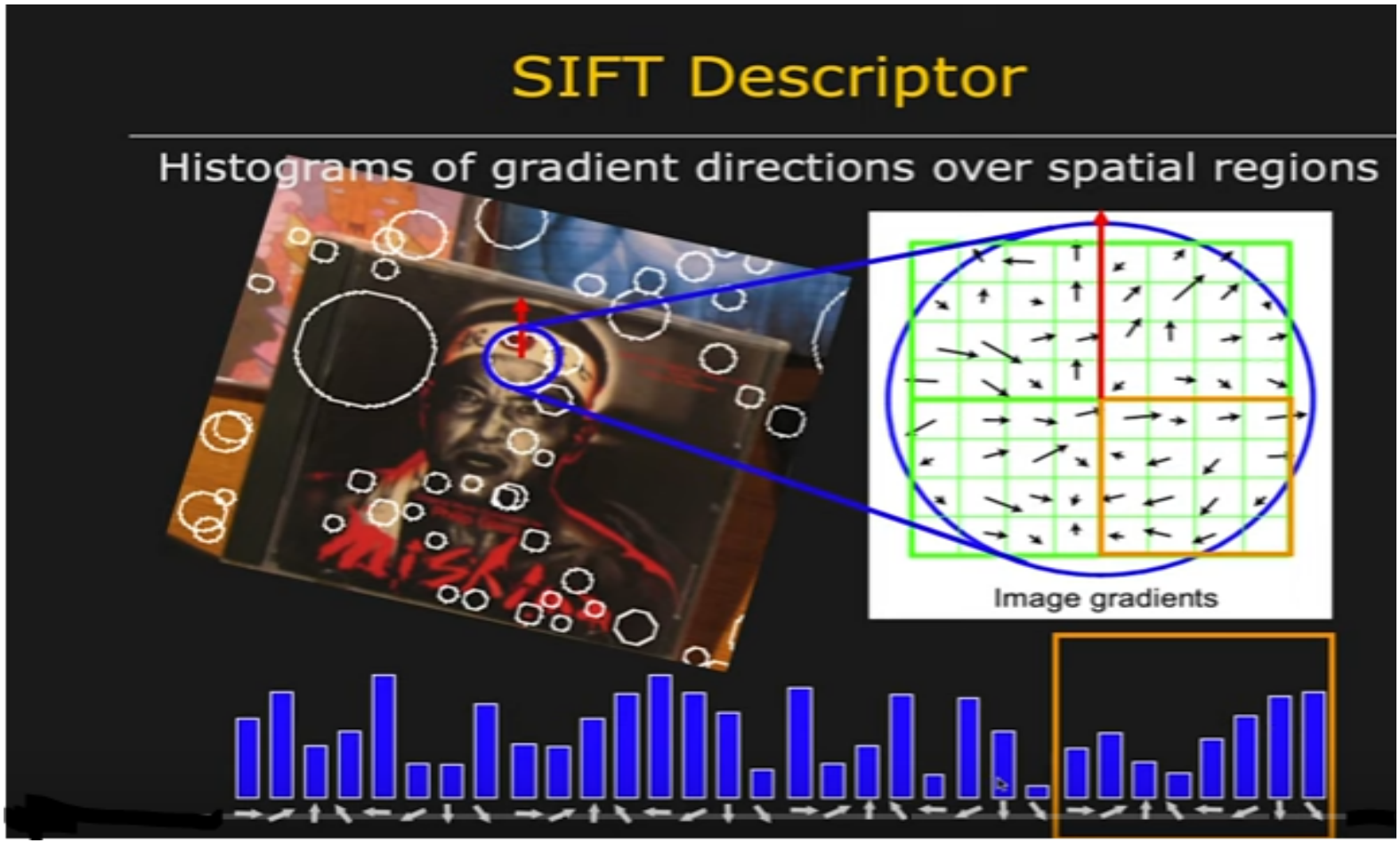
Convolve the image using NLoG at many scales σ

Find:

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

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

σ^*: Size of the blob


SIFT Descriptor
Histograms of gradient directions over spatial regions
Image gradients

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:

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

Smaller 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.

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 $d(H_1, H_2) = 1$

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:

$$d(H_1, H_2) = \sum_k \min\big(H_1(k), H_2(k)\big)$$

Larger the distance metric, better the match.

Speeded Up Robust Features (SURF)

- In computer vision, Speeded Up Robust Features (SURF) is a patented local feature detector and descriptor.
- It can be used for tasks such as object recognition, image registration, classification, or 3D reconstruction.
- It is partly inspired by the scale-invariant feature transform (SIFT) descriptor.
- The SURF method (Speeded Up Robust Features) is a fast and robust algorithm for local, similarity invariant representation and comparison of images.

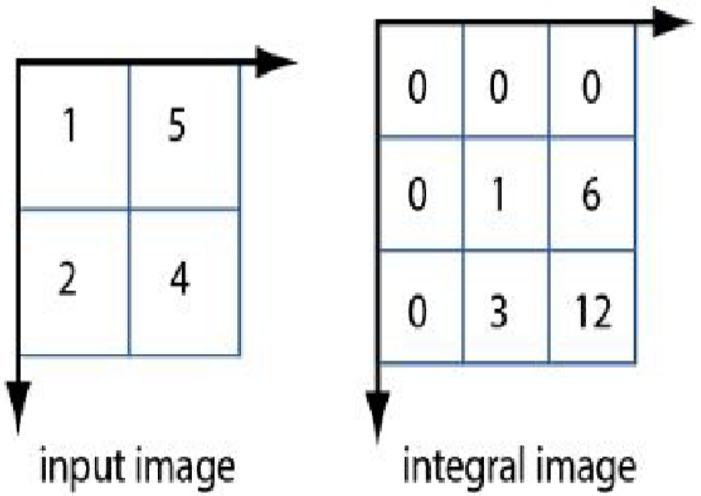
What is SURF?

SURF (**S**peeded-**U**p **R**obust **F**eatures) is a feature detection framework introduced by Herbert Bay and his colleagues at ETH Zurich. SURF interest points are in-plane rotation-invariant, robust to noise, and overall, extremely fast to calculate. This procedure can be divided into three steps:

1. Interest Point Detection
2. Interest Point Description
3. Interest Point Matching

Detection: Integral Images

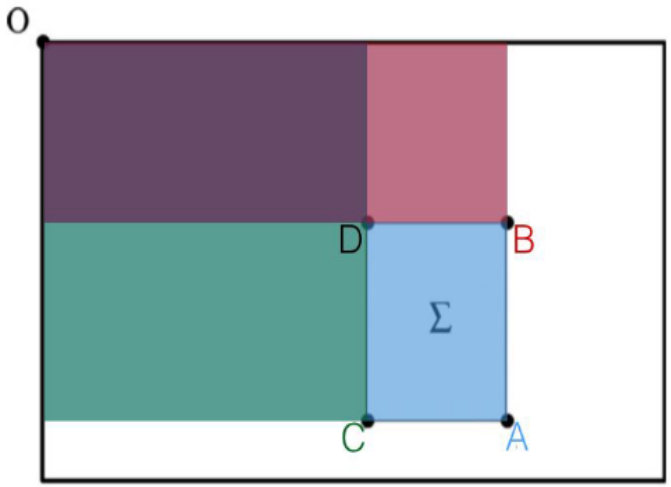
Integral images are an image transform such that any entry of an integral image $I_{\Sigma}(\mathbf{x})$ at a location $\mathbf{x} = (x, y)^T$ represents the sum of all pixels in the input image I within a rectangular region formed by the origin and $\mathbf{x}$.



$$I_{\Sigma}(\mathbf{x}) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(i, j)$$

Detection: Integral Images

Integral images are incredibly efficient. It is possible to characterize a region of the image using four memory accesses and three operations. This makes it very cheap to detect blobs.



$$\Sigma = A - B - C + D$$

Detection: Hessian-Based Interest Points

The detector detects blob-like structures at locations where the determinant of the Hessian is maximum.

The Hessian is defined as such: $$H(\mathbf{x}, \sigma) = \begin{bmatrix} L_{xx}(\mathbf{x}, \sigma) & L_{xy}(\mathbf{x}, \sigma) \\ L_{xy}(\mathbf{x}, \sigma) & L_{yy}(\mathbf{x}, \sigma) \end{bmatrix}$$

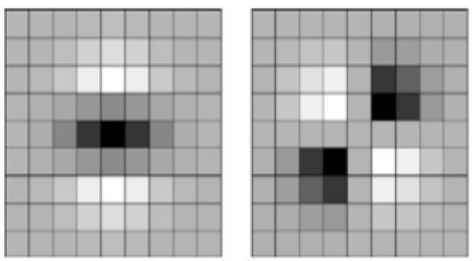
where $L_{xx}(\mathbf{x}, \sigma)$ is the convolution of the Gaussian second order derivative $\frac{\partial^2}{\partial x^2} g(\sigma)$ with the image I in point $\boldsymbol{x}$.

Detection: Hessian Approximation

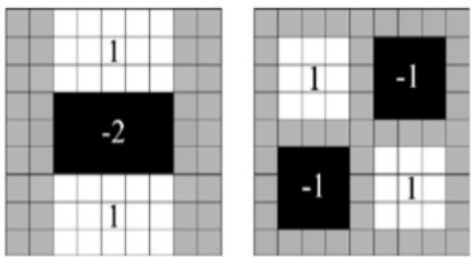
The actual computation of the Hessian matrix is expensive and slow. Instead, the Hessian can be approximated using box filters!

$$det(H_{approx}) = D_{xx}D_{yy} - (wD_{xy})^2$$

where D_{xx} is the approximation of the Gaussian second order partial derivative in the x-direction and $w = 0.9$.



The Gaussian second order partial derivative in y- and xy-direction.



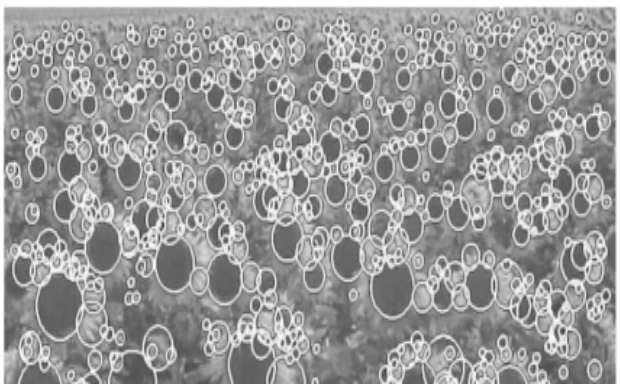
Box filter approximations of the Gaussian second order partial derivatives.

Detection: Scale Space Representation

To match interest points across different scales, a pyramidal scale space is built. Rather than serial downsampling, each successive level of the pyramid is built by upscaling the image in parallel. Each scale is defined as the the response of the image convolved with a box filter of a certain dimension (9x9, 15x15, 27x27 etc.). The scale space is further divided into octaves (sets of filter responses).

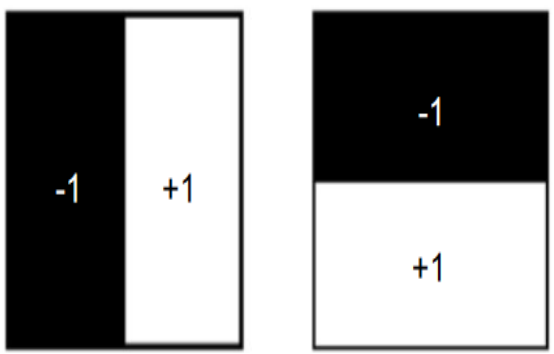
Detection: Interest Point Localization

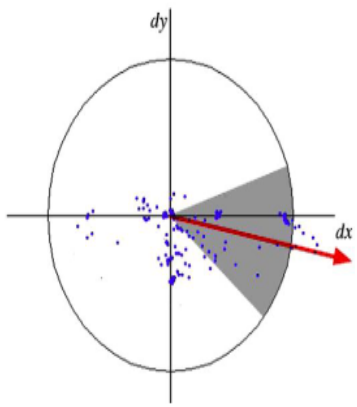
To localize interest points in the image and over scales, a non-maximum suppression (non-maximum pixels are set to 0) in a 3 x 3 x 3 neighborhood is applied. The maxima of the determinant of the Hessian matrix are then interpolated in scale and image space.



Descriptor: Orientation Assignment

The Haar wavelet responses in x- and y-direction within a circular neighborhood with radius 6s is calculated. Responses are weighted with a Gaussian ($\sigma = 2s$) centered at the interest point and then the directional strengths are plotted. These plots are then divided into sliding orientation windows and local orientation vectors are computed as the sum of the x and y responses within each window. The dominant orientation is the largest of all such vectors across all windows.





Descriptor: Feature Vector

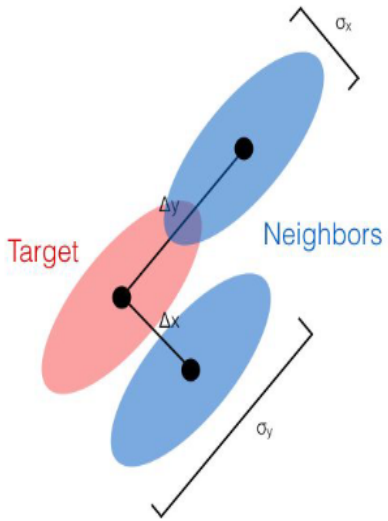
To extract features, an axis-orientated square window of size 20*s* and centered around the interest point is defined. This window is subdivided into a 4 x 4 grid. The "horizontal" and "vertical" Haar wavelet response is calculated over each subdivision and four metrics are extracted from each subdivision using 5 x 5 equally spaced points. These metrics are then summed to produce the local feature vector. These local feature vectors are concatenated to form a 64-element feature vector describing the interest point and surrounding neighborhood.

$$\mathbf{v} = \begin{bmatrix} \Sigma d_x \\ \Sigma d_y \\ \Sigma |d_x| \\ \Sigma |d_y| \end{bmatrix}$$

where d_x is the "horizontal" Haar wavelet response
d_y is the "vertical" Haar wavelet response

Matching: Nearest Neighbors

Features are matched across frames as the nearest neighbor within a distinct feature threshold. Either Euclidean or Mahalanobis distance may be used to determine "nearest". In this implementation, uniform precision was assumed and, therefore, Euclidean distance was sufficient.



$$D_E = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

$$D_M = \sqrt{\frac{(x_2 - x_1)^2}{\sigma_x^2} + \frac{(y_2 - y_1)^2}{\sigma_y^2}}$$

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

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*

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

P

Integral Image II

$$Sum = II_P + \cdots$$
$$= 3490 + \cdots$$

Summation Within a Rectangle

Fast summations of arbitrary rectangles 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

Q

P

Integral Image II

$$Sum = II_P - II_Q + \cdots$$
$$= 3490 - 1137 + \cdots$$

Summation Within a Rectangle

Fast summations of arbitrary rectangles 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

Q S P

Integral Image II

$$Sum = II_P - II_Q - II_S + \cdots$$
$$= 3490 - 1137 - 1249 + \cdots$$

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

R Q S P

Integral Image II

$$Sum = II_P - II_Q - II_S + II_R$$
$$= 3490 - 1137 - 1249 + 417 = 1521$$

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

Integral Image II

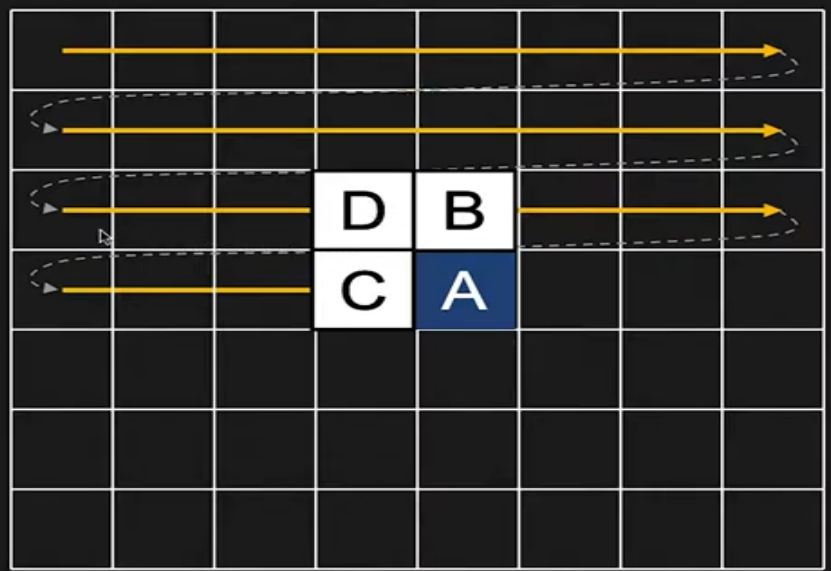
$$V_A = \sum(\text{pixel intensities in white}) - \sum(\text{pixel intensities in black})$$

$$= (II_O - II_T + II_R - II_S) - (II_P - II_Q + II_T - II_O)$$

$$= (2061 - 329 + 98 - 584) - (3490 - 576 + 329 - 2061) = 64$$

Computational Cost: Only 7 additions

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

$\otimes$

H_A

H_B

H_C

H_D

$\vdots$

$=$

$V_A[i,j]$

$V_B[i,j]$

$V_C[i,j]$

$V_D[i,j]$

$\vdots$