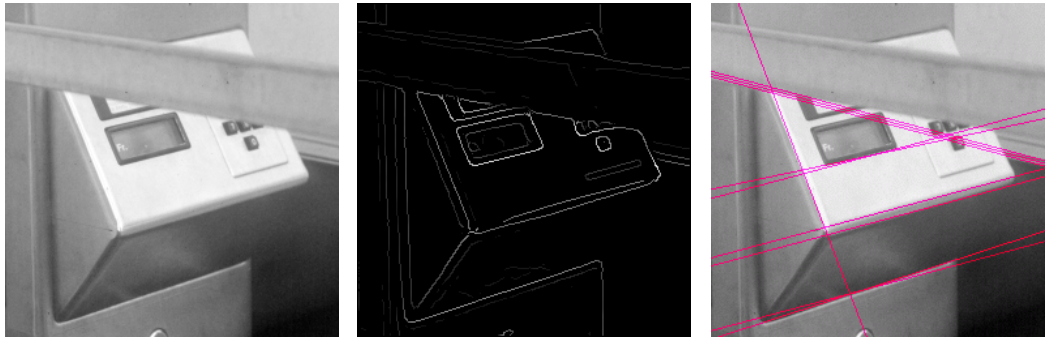
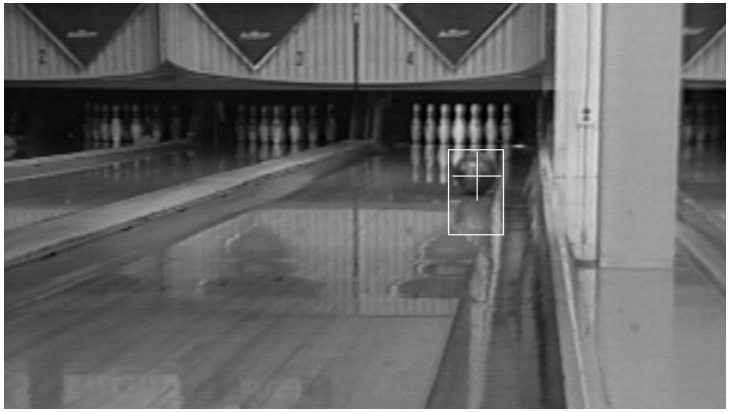
Review: Hough Transform

Fitting Line & Circle

- Want to associate a model with observed features



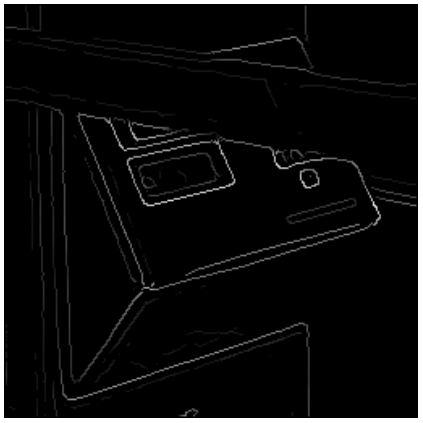
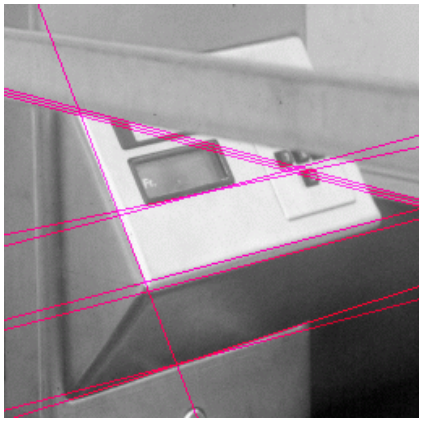


[Fig from Marszalek & Schmid, 2007]

For example, the model could be a line, a circle, or an arbitrary shape.

Ballard, Dana H. "Generalizing the Hough transform to detect arbitrary shapes." Pattern recognition 13.2 (1981): 111-122.

Fitting lines

- **Hough Transform** is a voting technique that can be used to answer all of these
  - 1. Record all possible lines on which each edge point lies.
  - 2. Look for lines that get many votes.

variables

$$y = mx + b$$

parameters

parameters

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

variables

variables

$$y - mx = b$$

parameters

Hough Circle

parameters

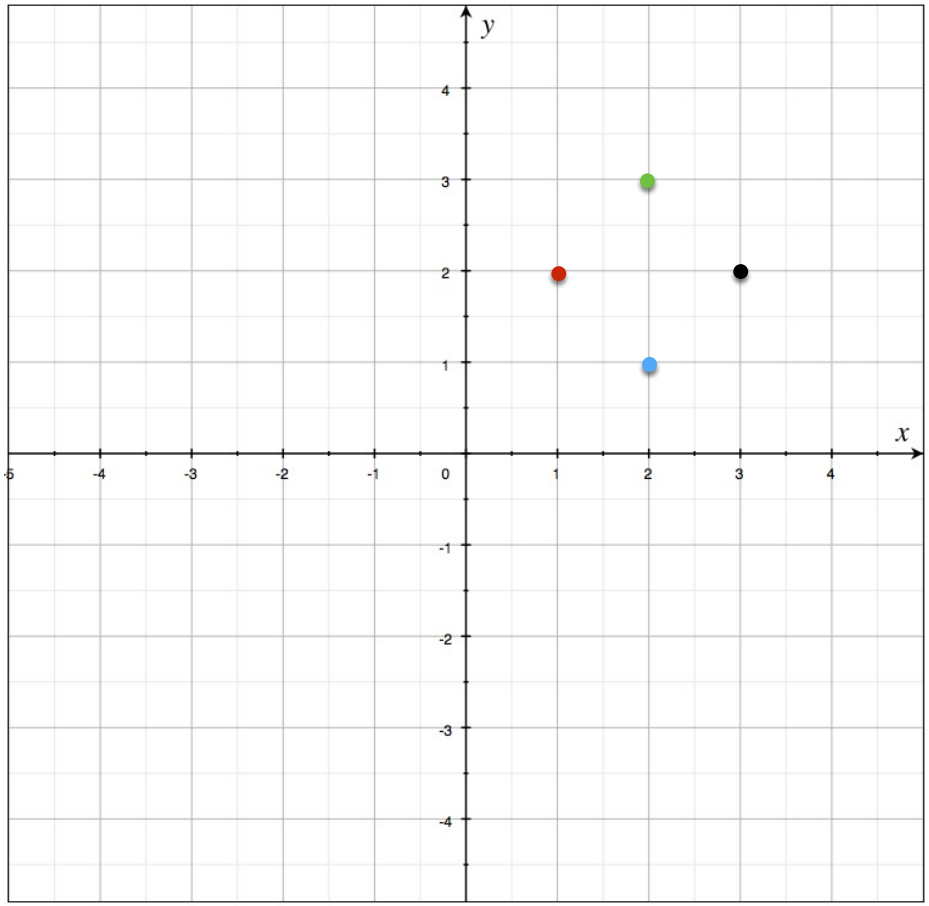
$$(x-a)^2+(y-b)^2=r^2$$

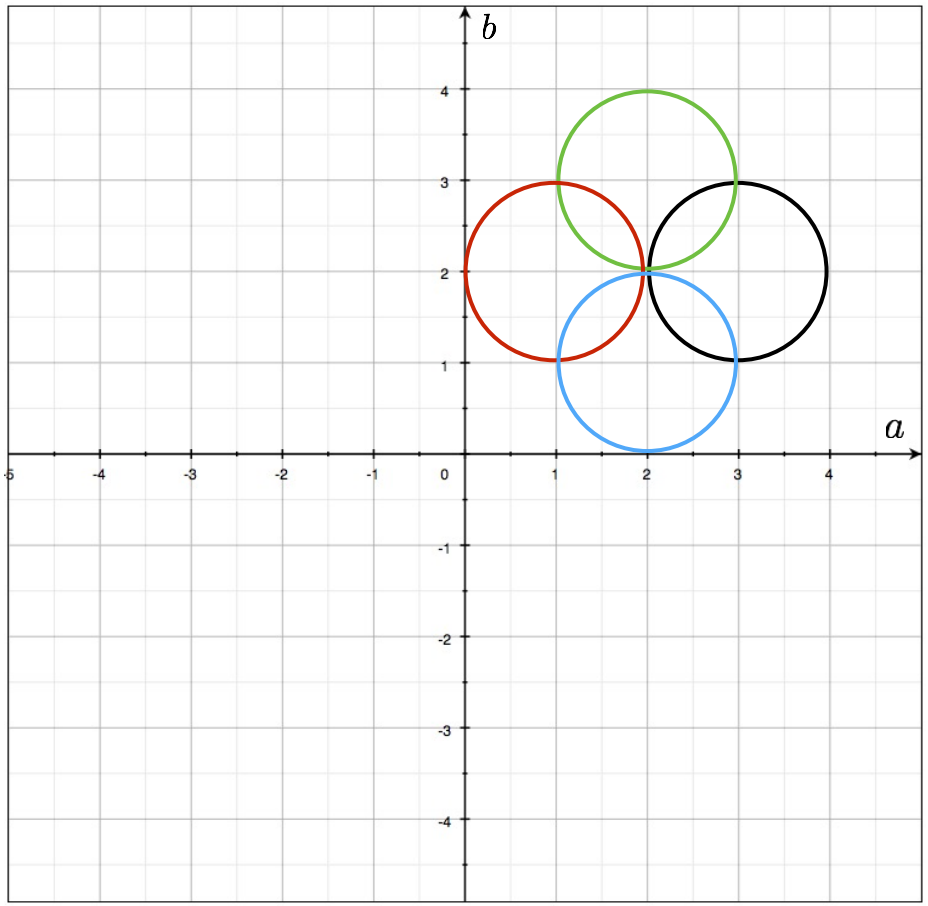
variables

parameters

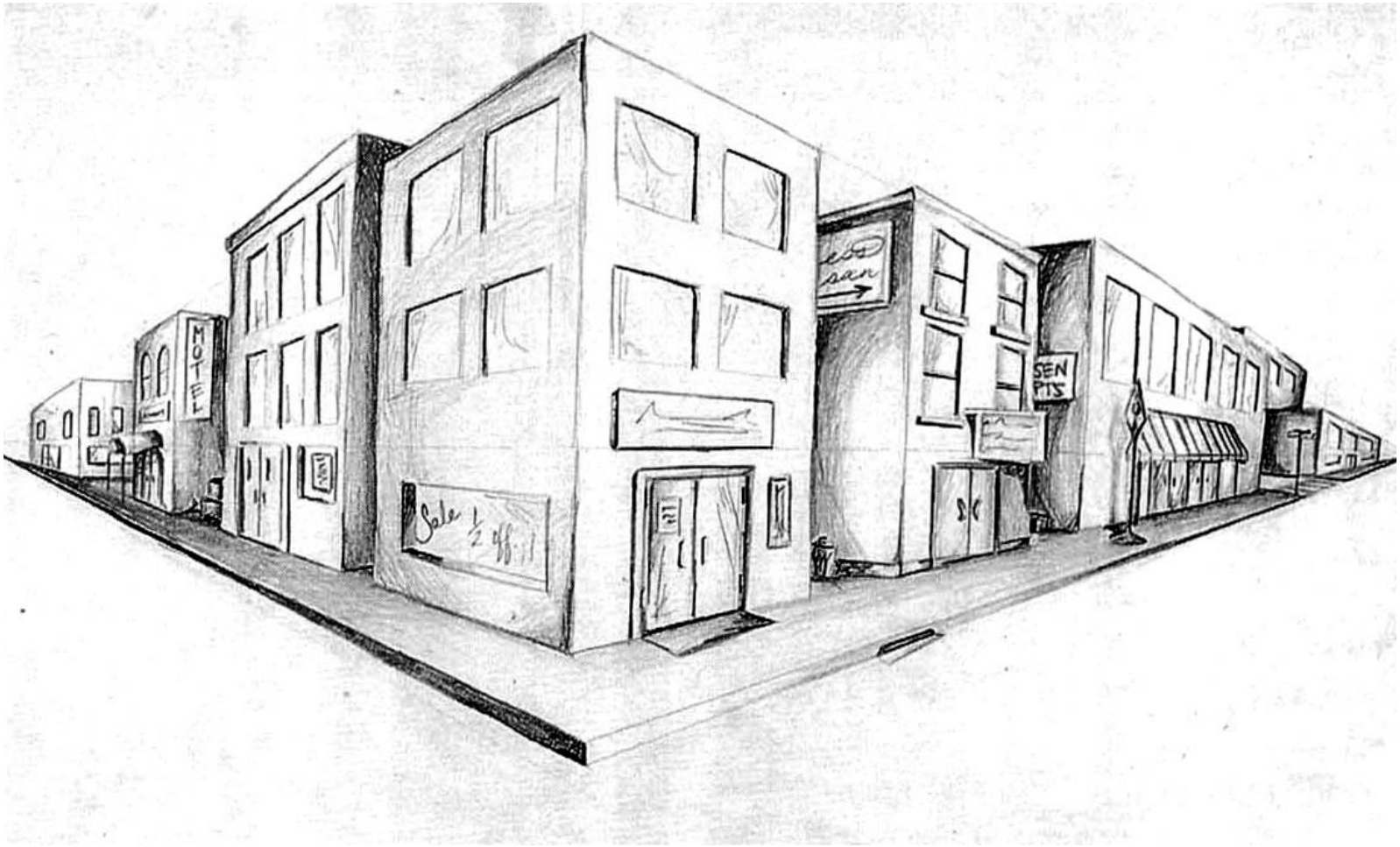
$$(x-a)^2+(y-b)^2=r^2$$

variables



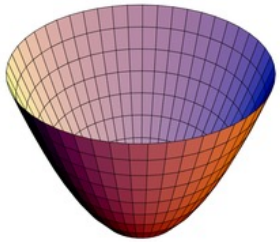


Detecting Corners



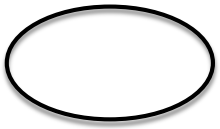
Computer Vision Fall
2022, Lecture 6

Visualizing quadratics



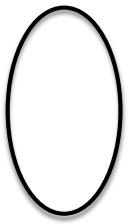
$$f(x,y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

you can smash this bowl in the **y** direction



$$f(x,y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

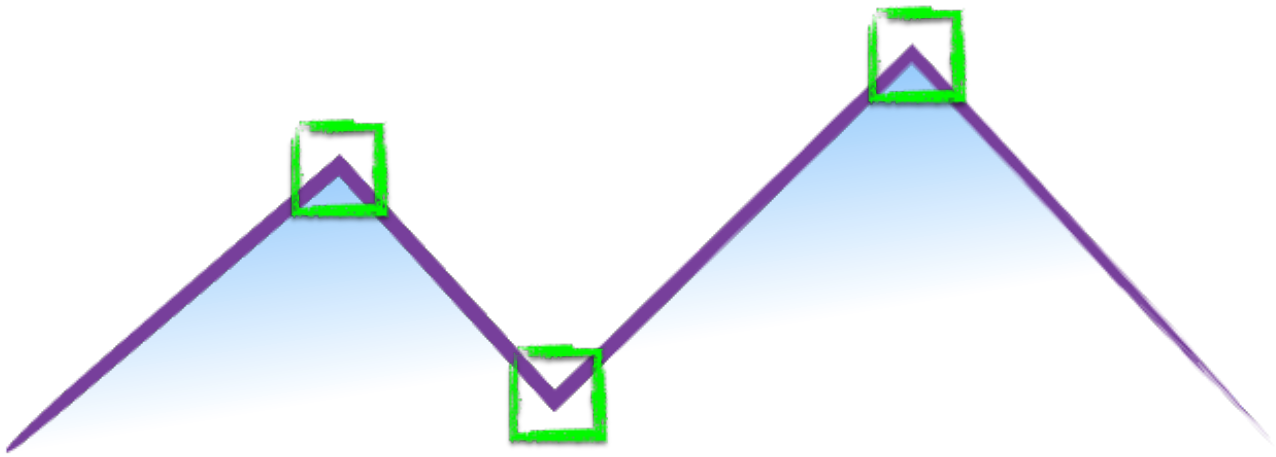
you can smash this bowl in the **x** direction



$$f(x,y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

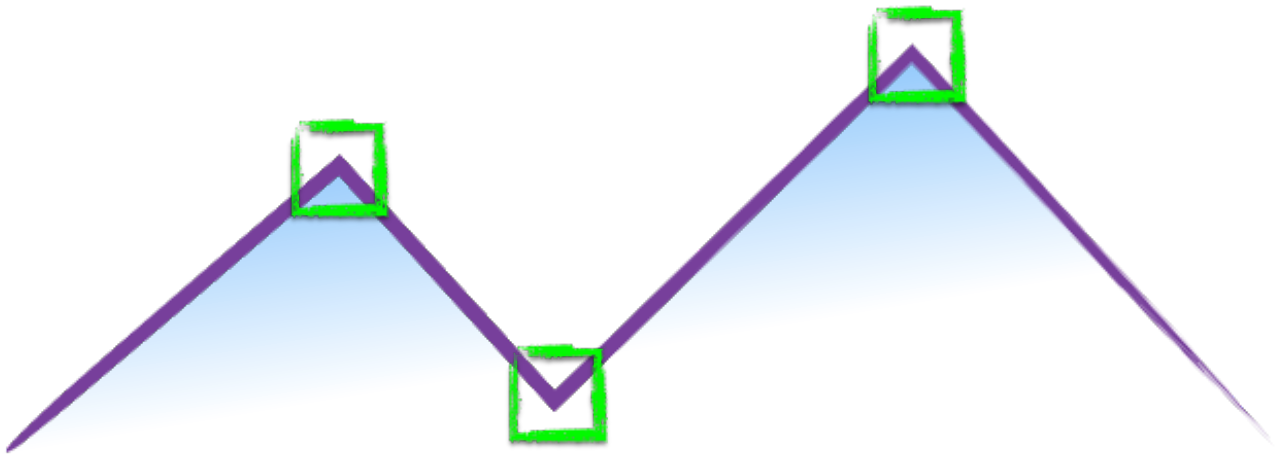
Harris corner detector

How do you find a corner?



How do you find a corner?

[Moravec 1980]

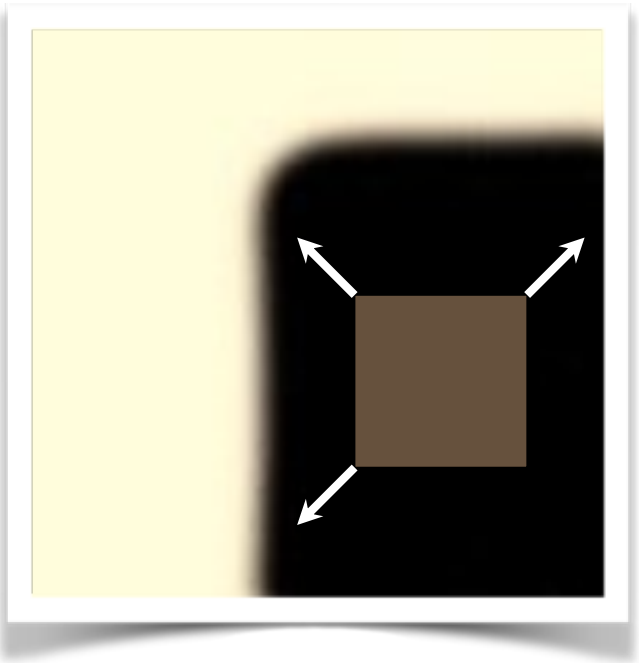


Easily recognized by looking through a small window

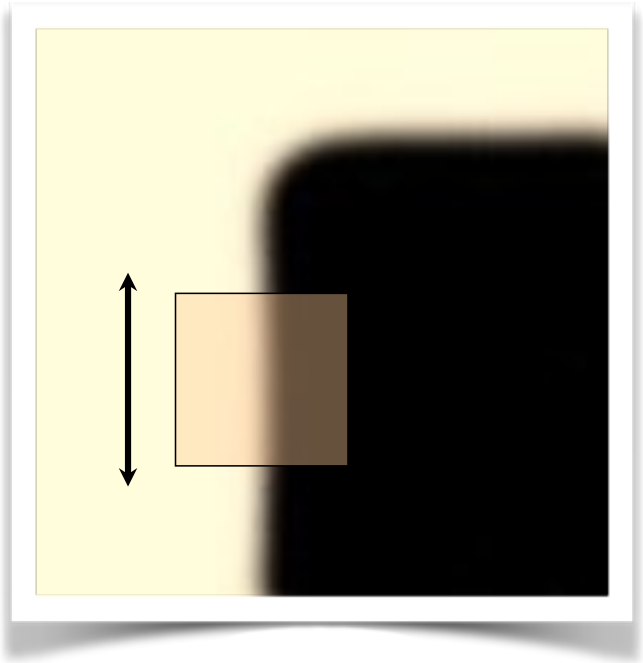
Shifting the window should give large change in intensity

Easily recognized by looking through a small window

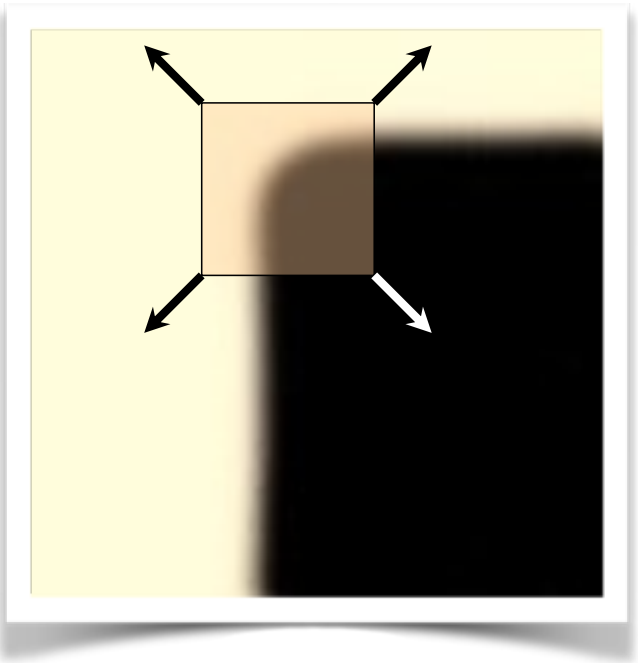
Shifting the window should give large change in intensity



"flat" region:
no change in all directions



"edge":
no change along the edge direction



"corner":
significant change in all directions

[Moravec 1980]

Error function

Change of intensity for the shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

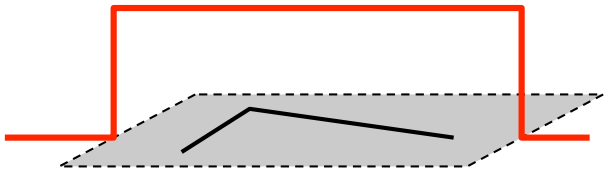
Error function

Window function

Shifted intensity

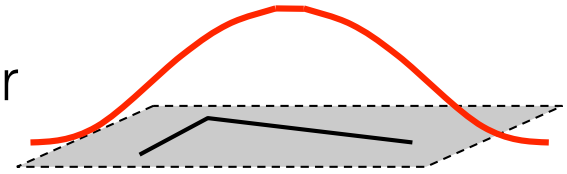
Intensity

Window function $w(x,y)$ =



1 in window, 0 outside

or



Gaussian

Error function approximation

Change of intensity for the shift $[u,v]$:

$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

First-order Taylor expansion of I(x,y) about (0,0)
(bilinear approximation for small shifts)

Error function approximation

$$f(x,y) = f(x_k,y_k) + (x-x_k)f'_x(x_k,y_k) + (y-y_k)f'_y(x_k,y_k)$$
$$+ \frac{1}{2!}(x-x_k)^2 f''_{xx}(x_k,y_k) + \frac{1}{2!}(x-x_k)(y-y_k)f''_{xy}(x_k,y_k)$$
$$+ \frac{1}{2!}(x-x_k)(y-y_k)f''_{yx}(x_k,y_k) + \frac{1}{2!}(y-y_k)^2 f''_{yy}(x_k,y_k)$$
$$+ o^n$$

$$f(x+u, y+v) \approx f(x,y) + uf_x(x,y) + vf_y(x,y)$$

Error function approximation

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

$$\approx \sum_{(x,y)\in W} w(x,y)[I(x,y)+uI_x+vI_y-I(x,y)]^2$$

$$= \sum_{(x,y)\in W} w(x,y)[u^2I_x^2+2uvI_xI_y+v^2I_y^2]$$

$$= \sum_{(x,y)\in W} w(x,y)\begin{bmatrix}u & v\end{bmatrix}\begin{bmatrix}I_x^2 & I_xIy\\ I_xI_y & I_y^2\end{bmatrix}\begin{bmatrix}u\\ v\end{bmatrix}$$

$$= \begin{bmatrix}u & v\end{bmatrix}\left(\sum_{(x,y)\in W} w(x,y)\begin{bmatrix}I_x^2 & I_xIy\\ I_xI_y & I_y^2\end{bmatrix}\right)\begin{bmatrix}u\\ v\end{bmatrix}$$

Bilinear approximation

For small shifts $[u,v]$ we have a 'bilinear approximation':

Change in appearance for a shift [u,v]

$$E(u,v) \cong \begin{bmatrix} u, v \end{bmatrix} \; M \; \begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a 2×2 matrix computed from image derivatives:

'second moment' matrix
'structure tensor'

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

Windowing function - computing a weighted sum (simplest case, w=1)

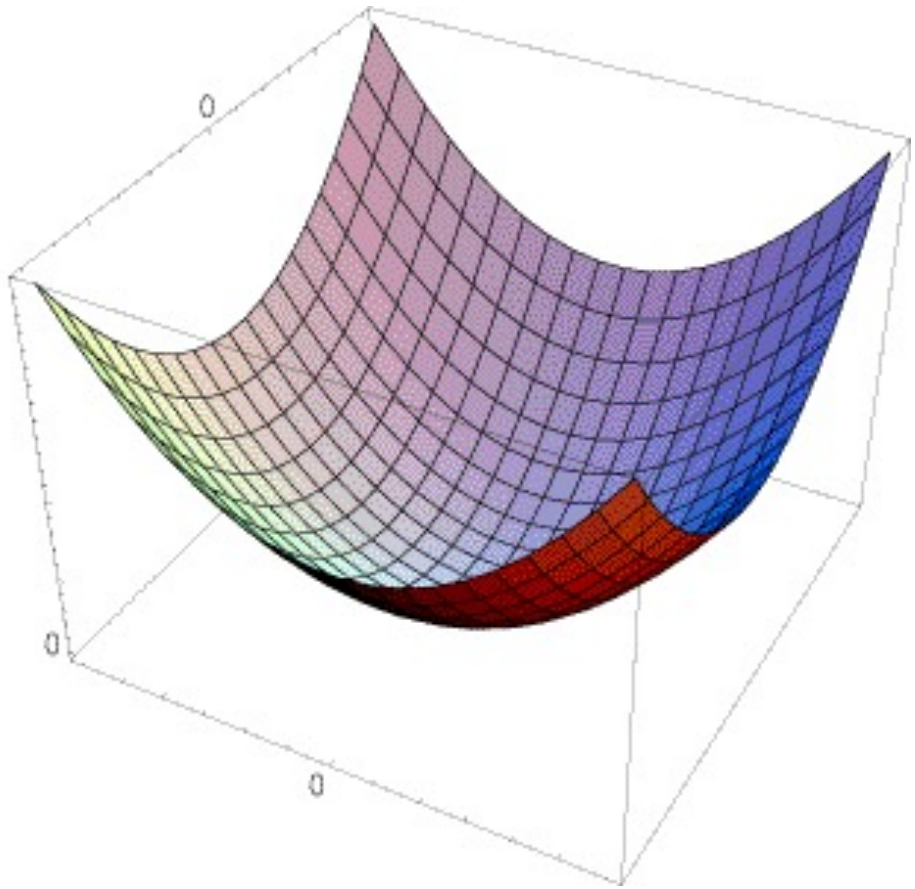
Note: these are just products of components of the gradient, Ix, Iy

Visualization of a quadratic

The surface $E(u,v)$ is locally approximated by a quadratic form

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix}$$

$$M = \sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



Design a program to detect corners

Finding corners

1. Compute image gradients over small region (3x3, 5x5, 7x7)

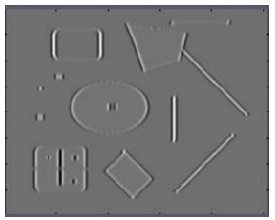
2. Subtract mean from each image gradient

3. Compute the covariance matrix

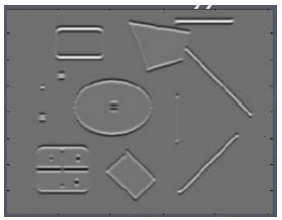
4. Compute eigenvectors and eigenvalues

5. Use threshold on eigenvalues to detect corners

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



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



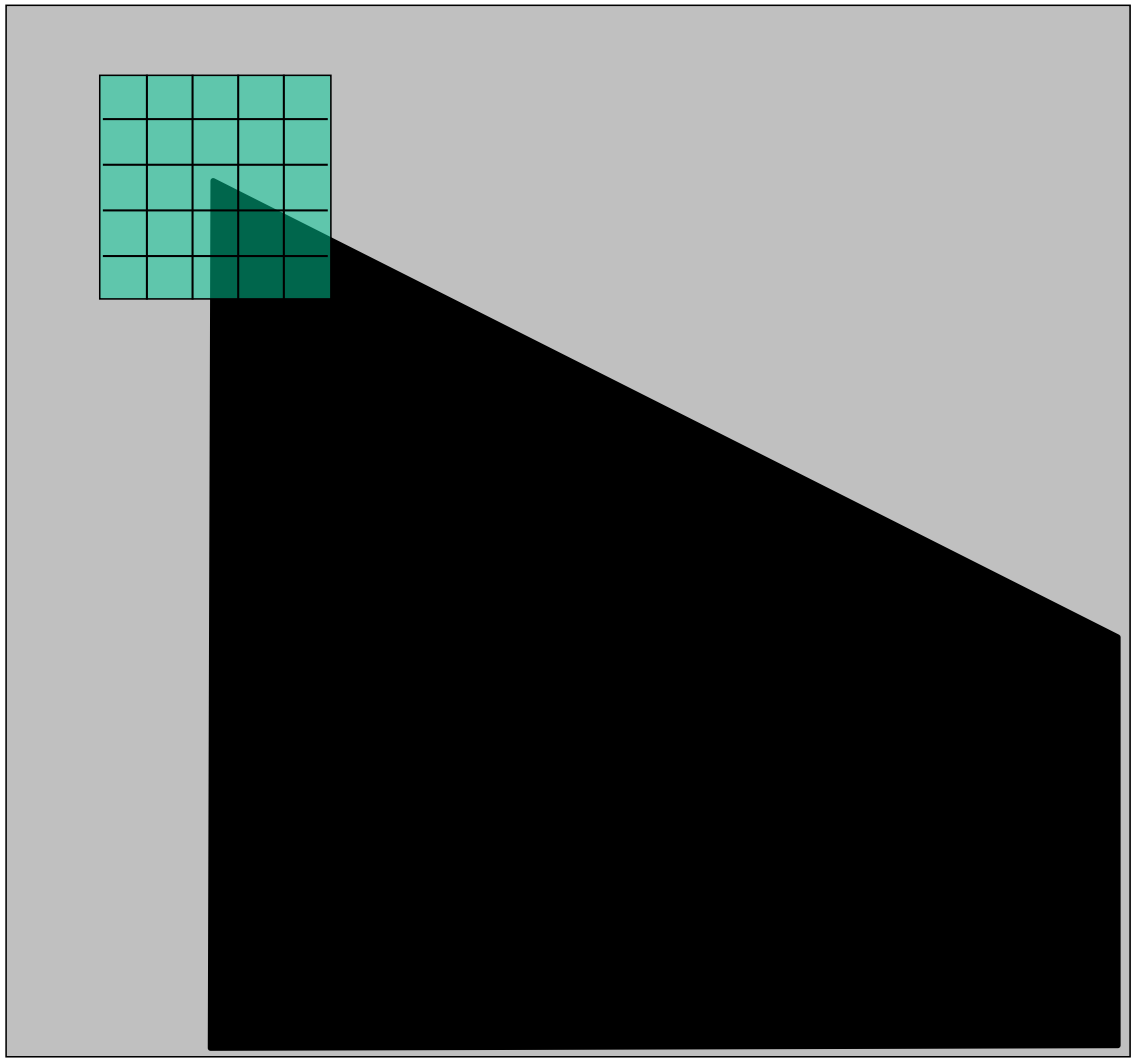
$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_y I_x & \sum I_y I_y \end{bmatrix}$$

1. Compute image gradients over a small region

(not just a single pixel)

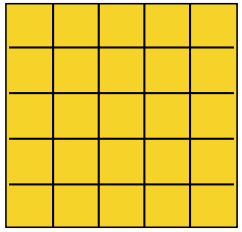
1. Compute image gradients over a small region

(not just a single pixel)



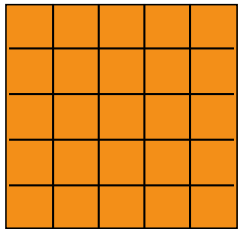
array of x gradients

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



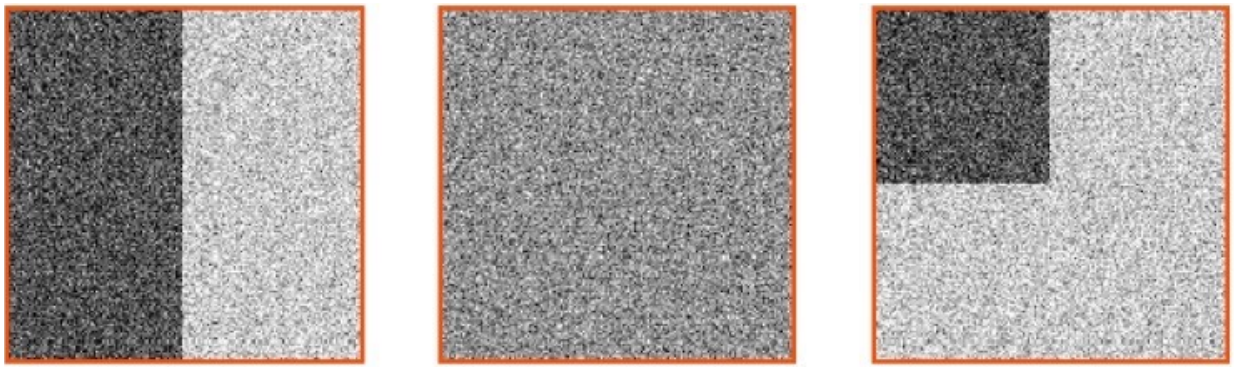
array of y gradients

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



visualization of gradients

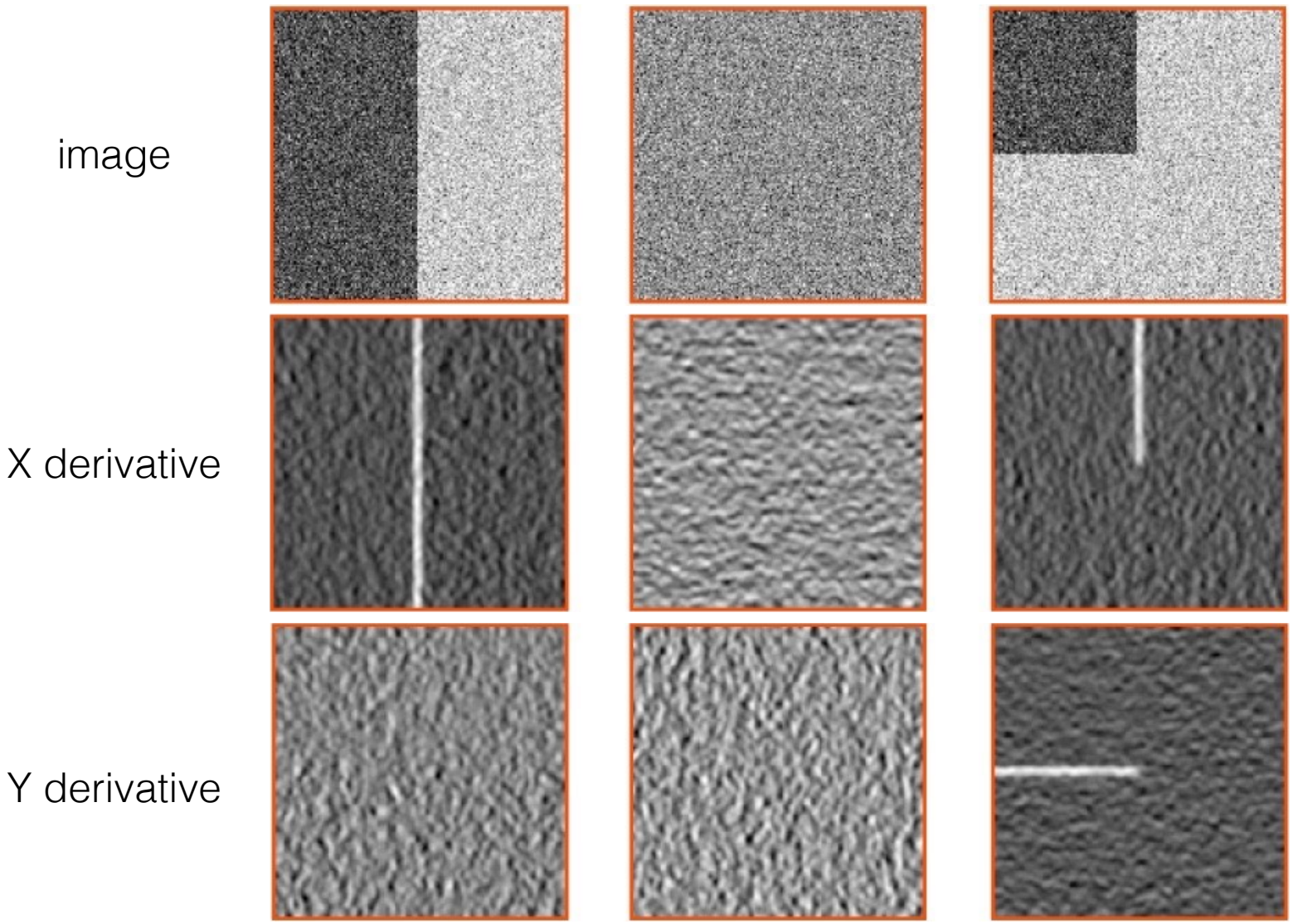
image

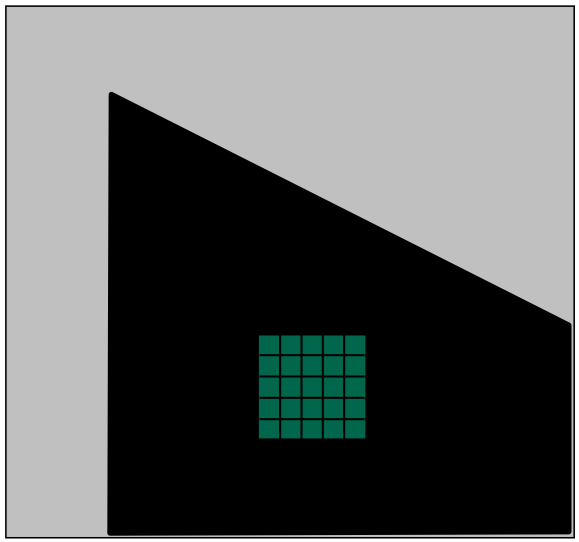


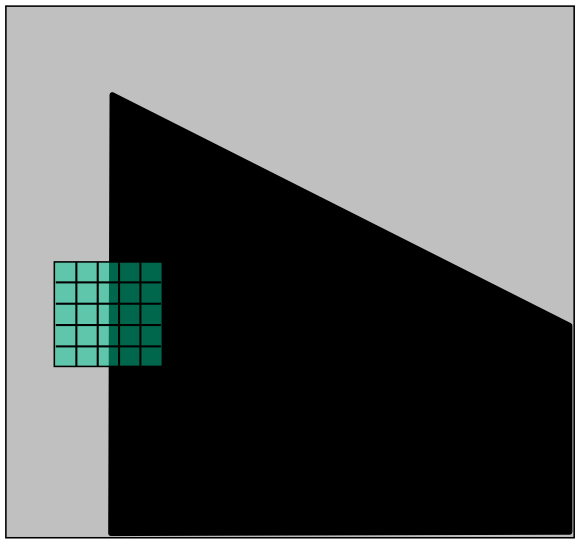
X derivative

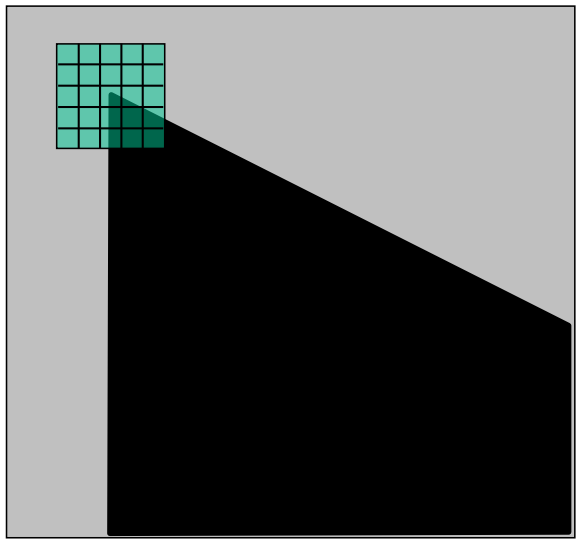
Y derivative

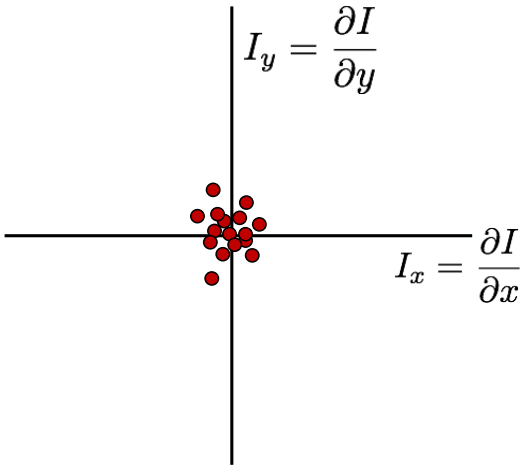
visualization of gradients

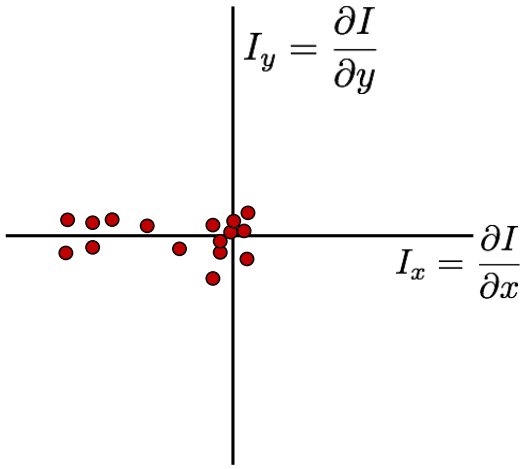


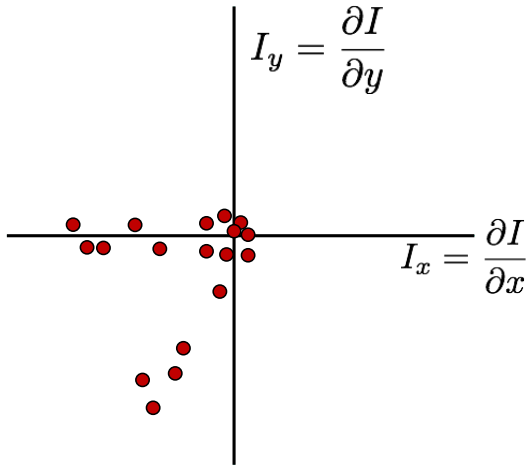




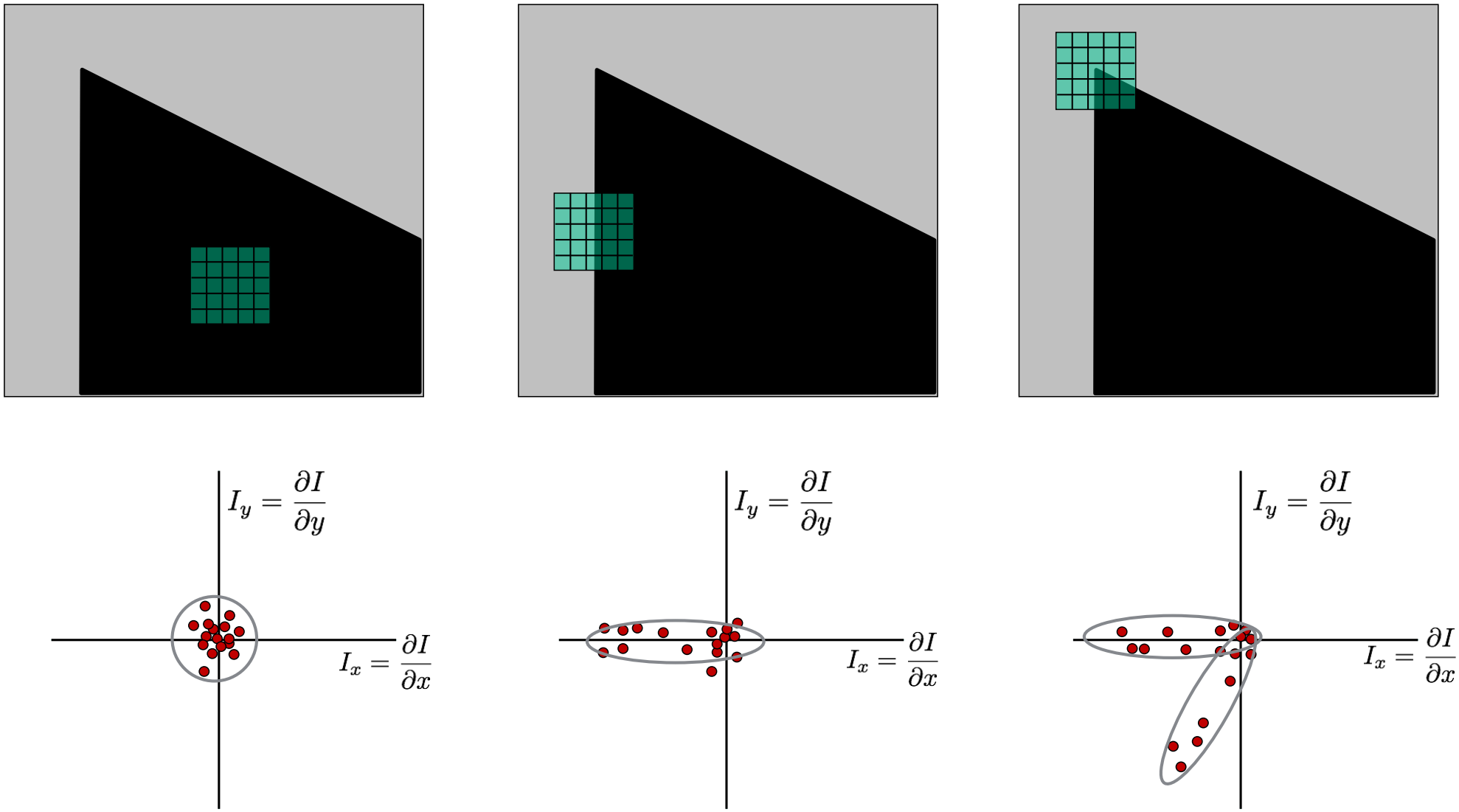




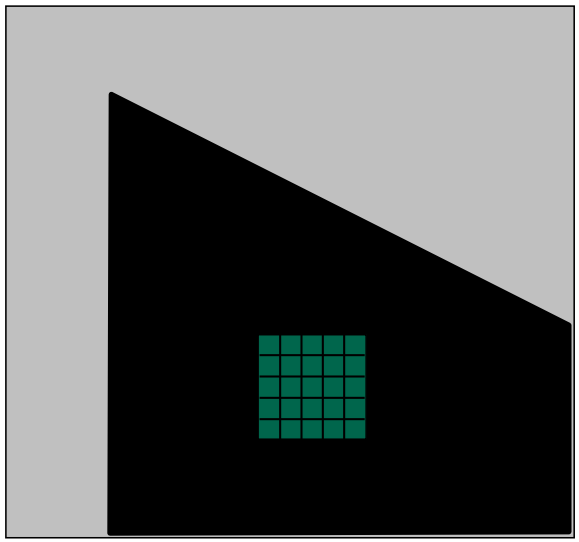


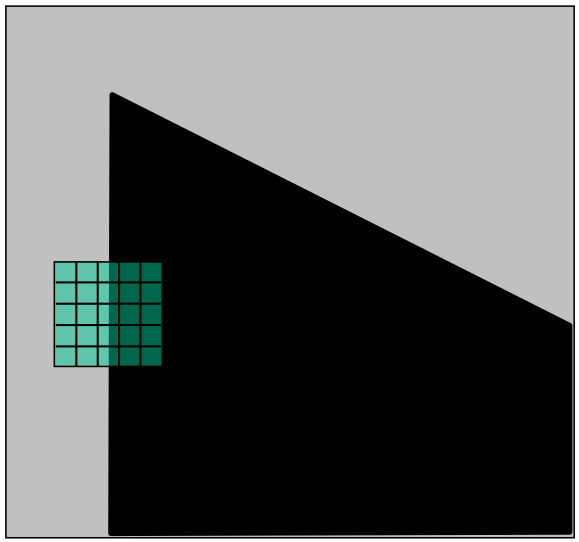


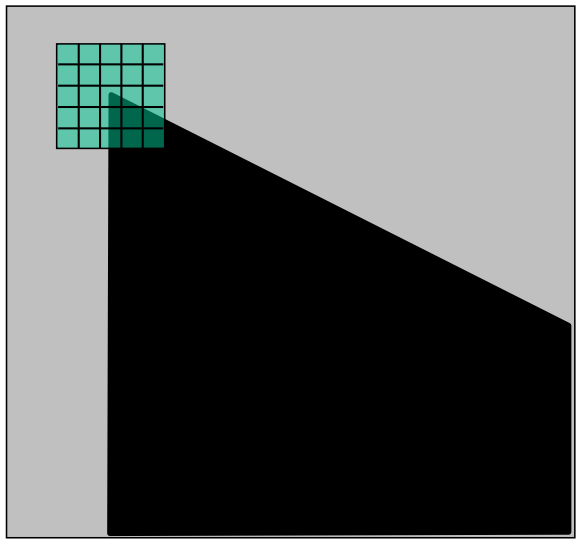
What does the distribution tell you about the region?

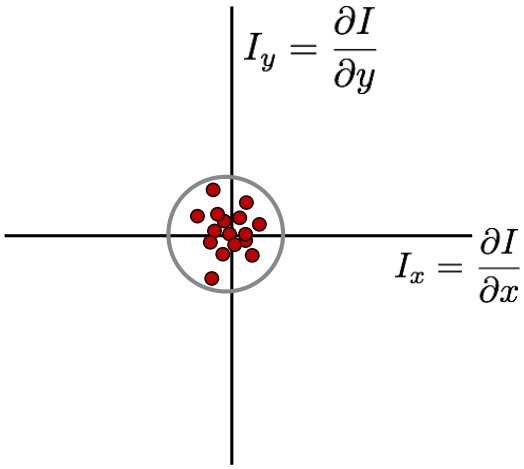


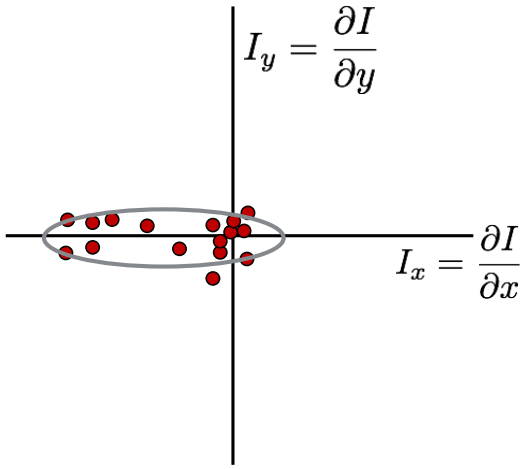
distribution reveals edge orientation and magnitude

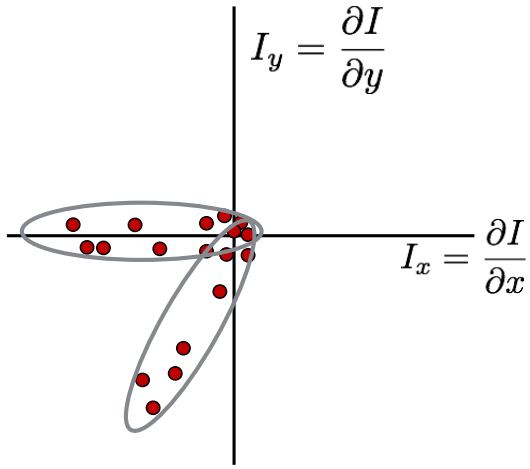










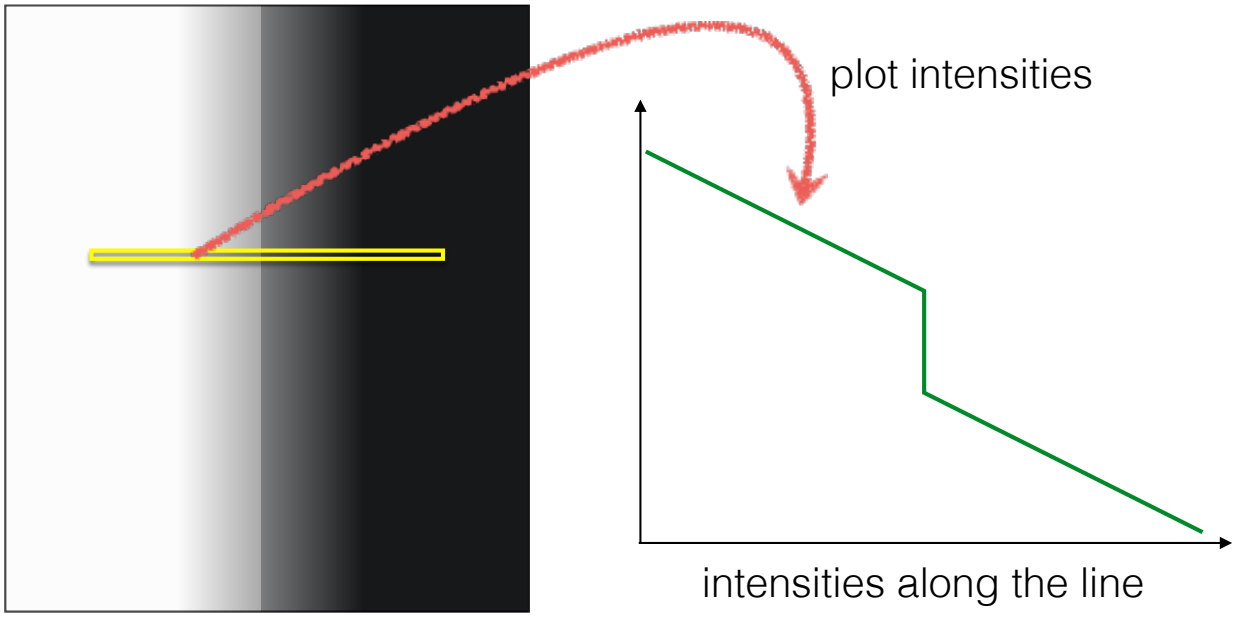


How do you quantify orientation and magnitude?

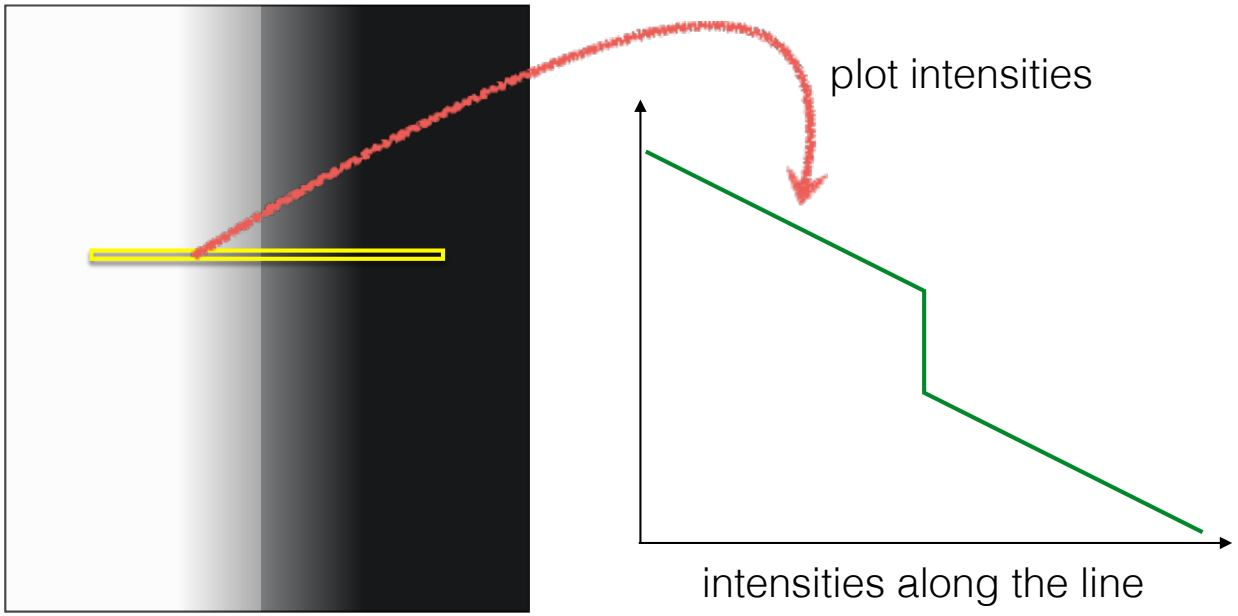
2. Subtract the mean from each image gradient

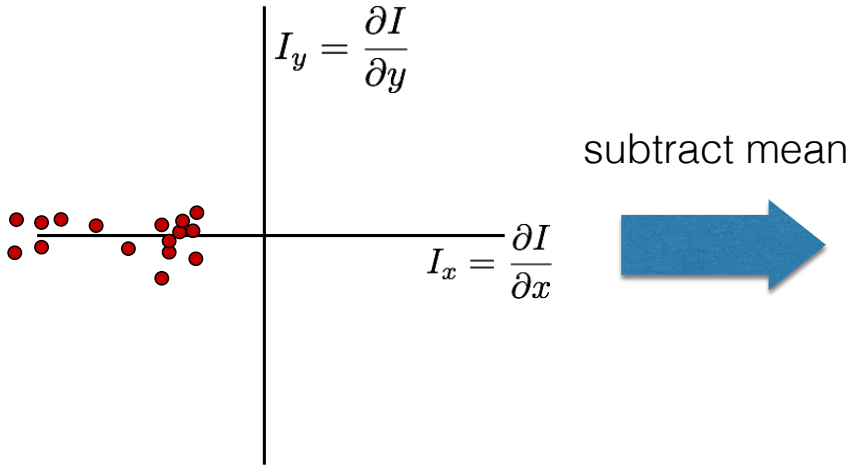
2. Subtract the mean from each image gradient

constant intensity gradient



2. Subtract the mean from each image gradient

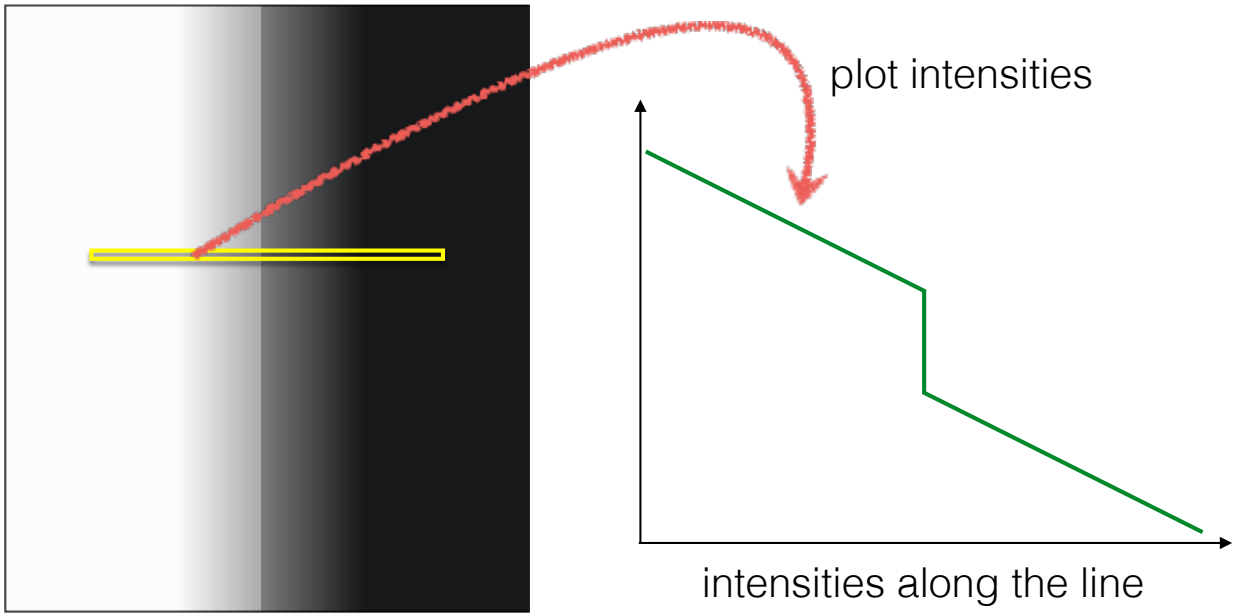




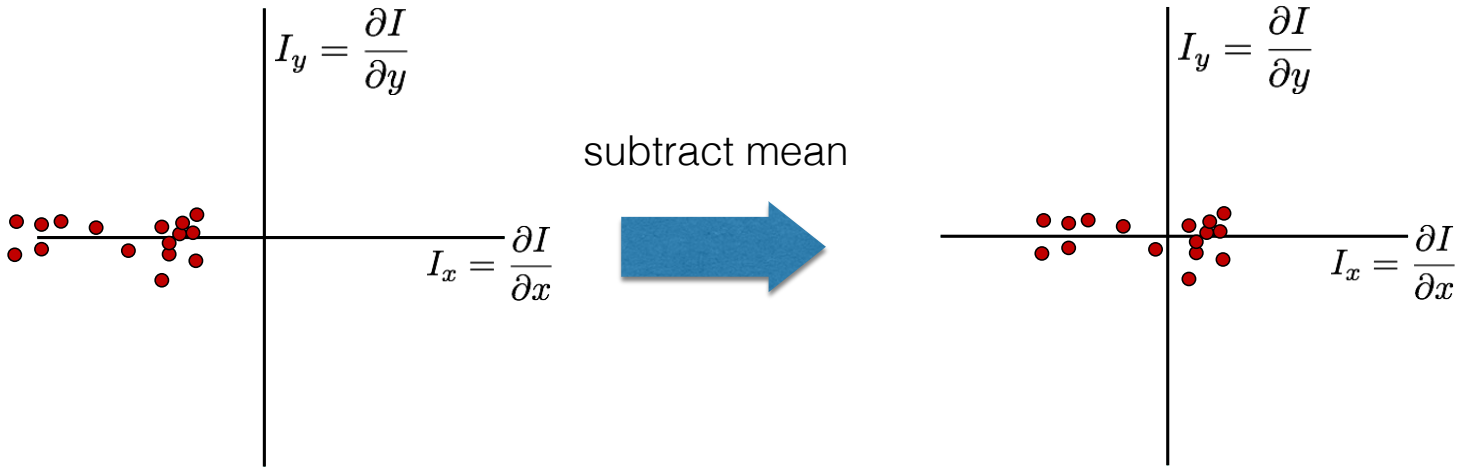
subtract mean

plot of image gradients

2. Subtract the mean from each image gradient



constant intensity gradient



plot of image gradients

data is centered
('DC' offset is removed)

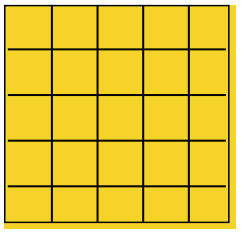
https://en.wikipedia.org/wiki/DC_bias

3. Compute the covariance matrix

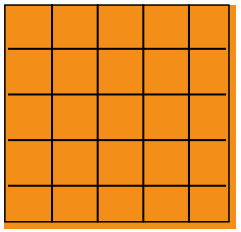
3. Compute the covariance matrix

$$\begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_y I_x & \sum I_y I_y \end{bmatrix}$$

$$\sum_P I_x I_y = \text{sum}\Big(\; I_x = \frac{\partial I}{\partial x} \; * \; I_y = \frac{\partial I}{\partial y} \; \Big)$$

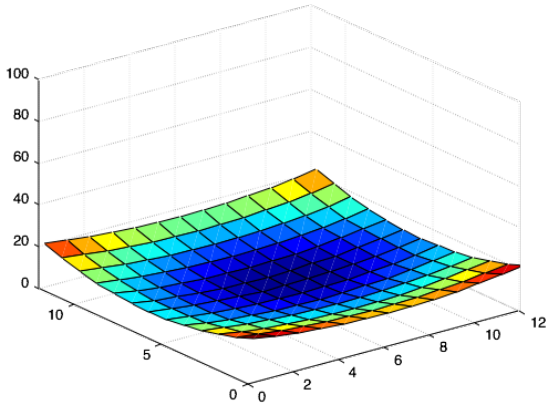


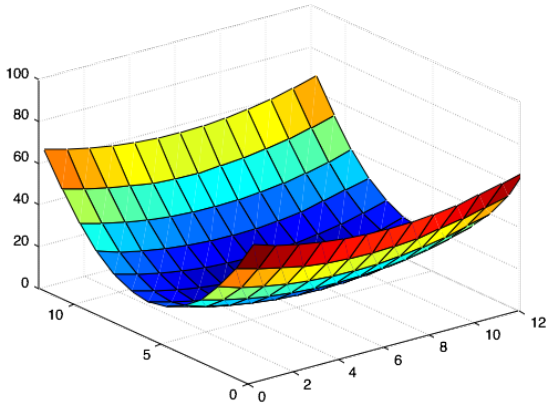
array of x gradients

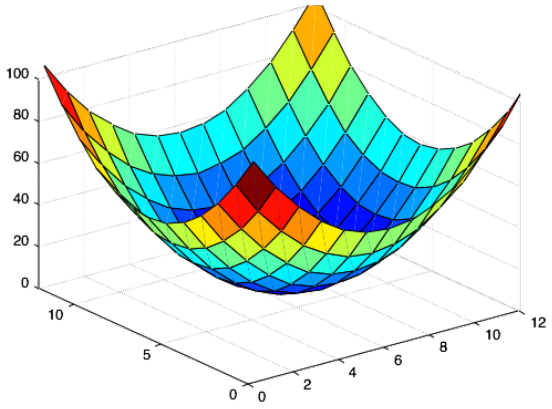


array of y gradients

Which error surface indicates a good image feature?

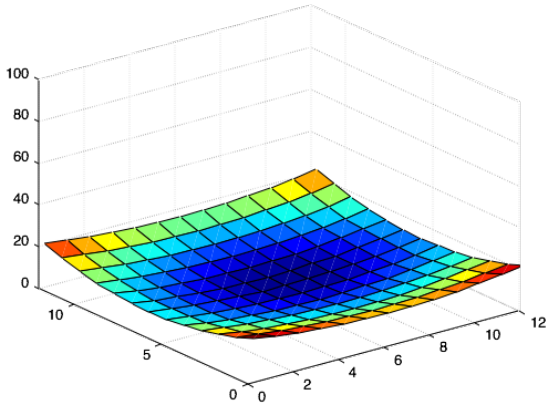




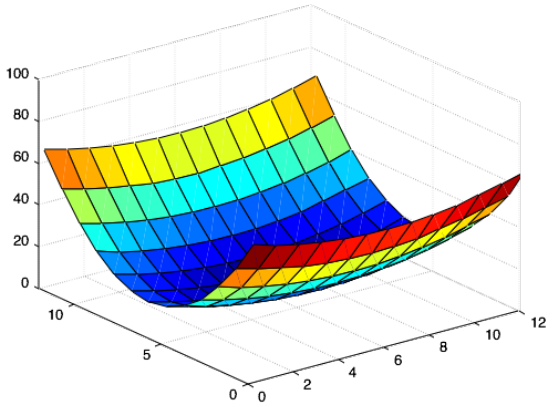


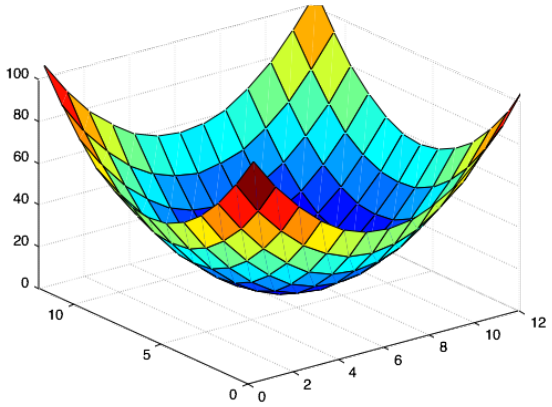
What kind of image patch do these surfaces represent?

Which error surface indicates a good image feature?

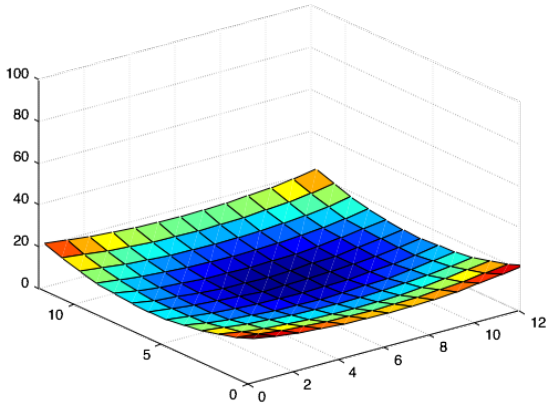


flat

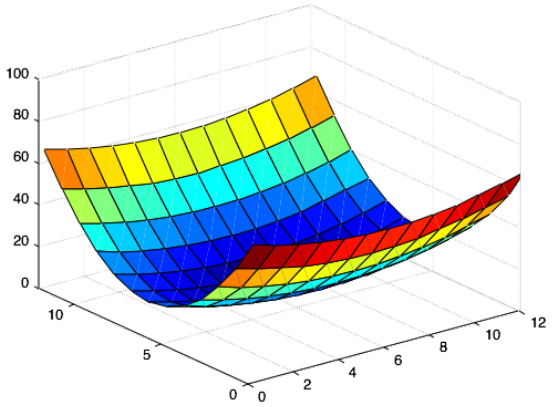




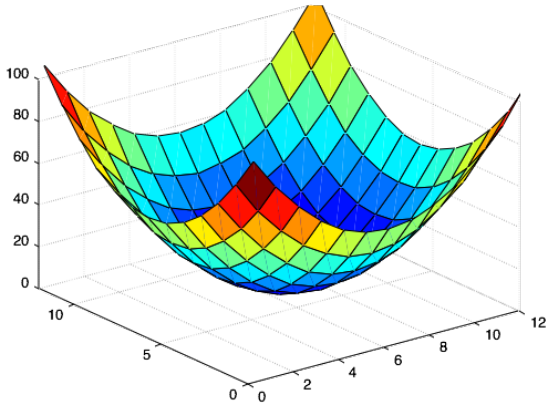
Which error surface indicates a good image feature?



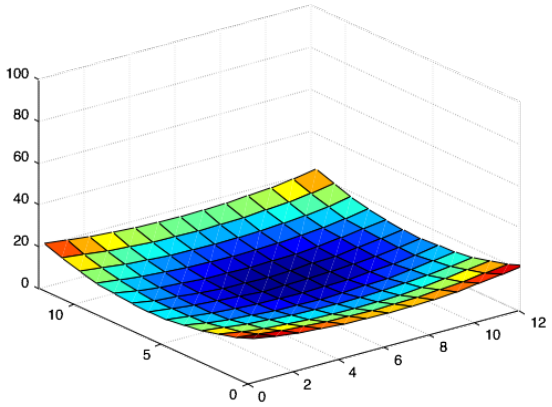
flat



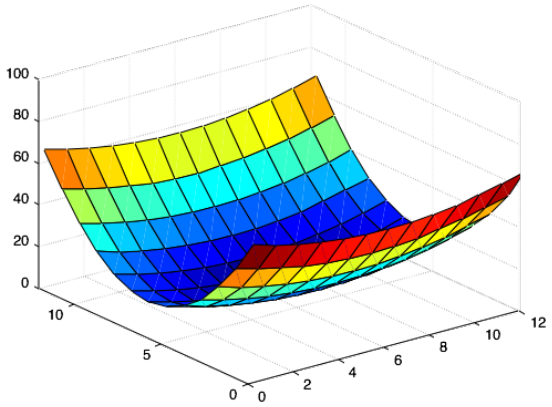
edge
'line'



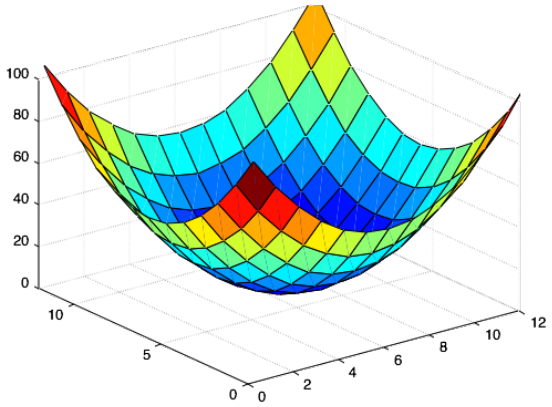
Which error surface indicates a good image feature?



flat



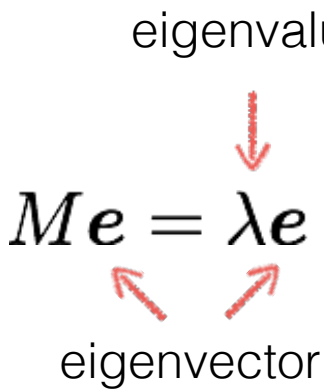
edge
‘line’



corner
‘dot’

4. Compute eigenvalues and eigenvectors

4. Compute eigenvalues and eigenvectors



$$(M - \lambda I)e = 0$$

eig(M)

numpy.linalg.eig(M)

```
import numpy as np
from numpy import linalg as LA

M = np.array([[7.75, 3.90], [3.90, 3.25]])
w, v = LA.eig(M)
print('The eigenvalues are\n', w)
print('The eigenvectors are\n', v)
```

```
The eigenvalues are
 [10.00249931  0.99750069]
The eigenvectors are
 [[ 0.86594528 -0.50013875]
 [ 0.50013875  0.86594528]]
```

https://numpy.org/doc/stable/reference/generated/numpy.linalg.eig.html

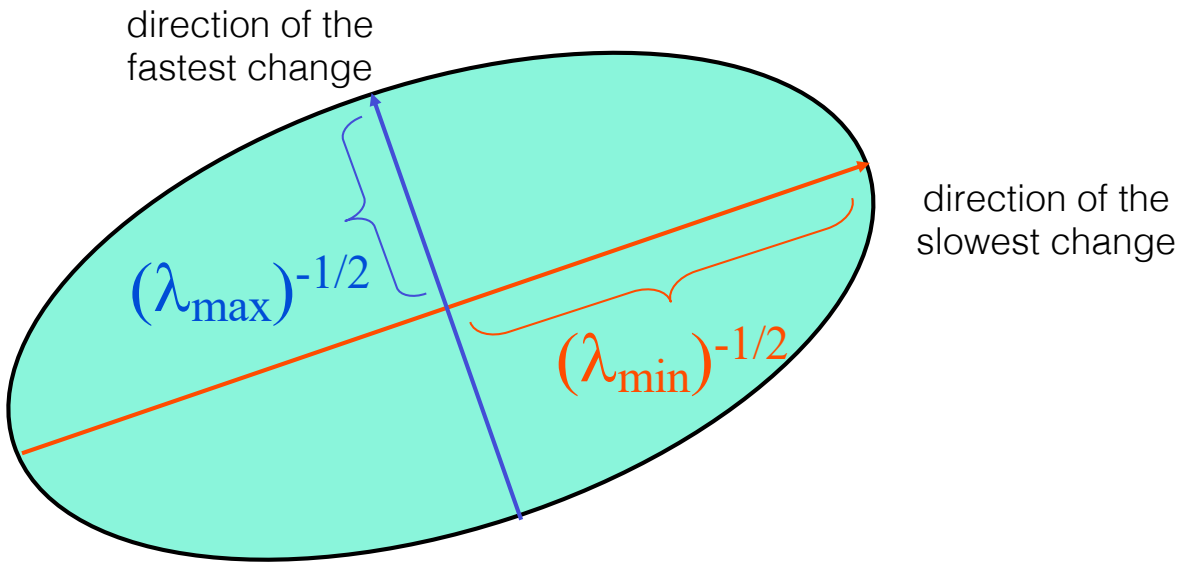
Visualization as an ellipse

Since M is symmetric, we have $M = A^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} A$

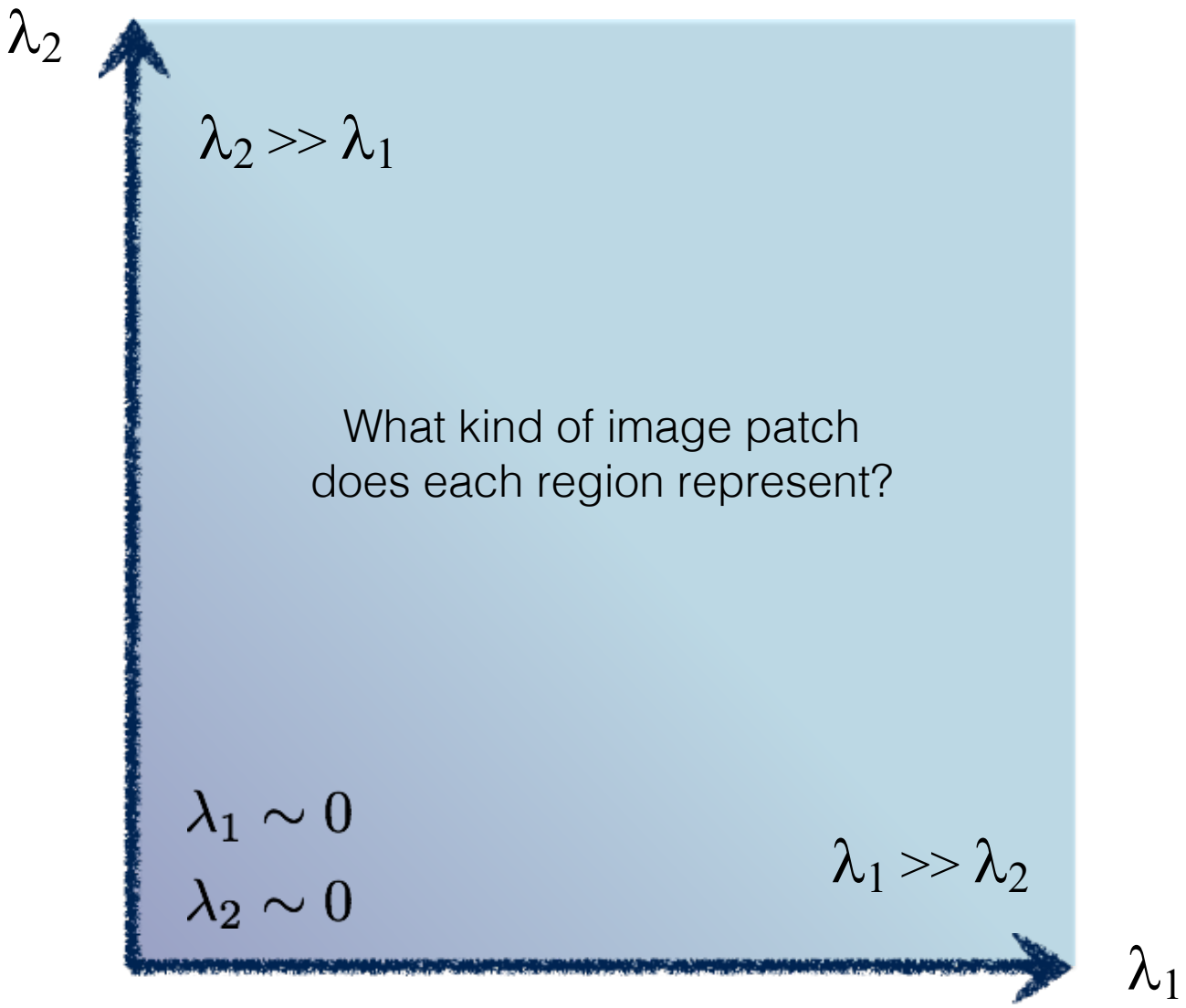
We can visualize ***M*** as an ellipse with axis lengths determined by the eigenvalues and orientation determined by A

Ellipse equation:

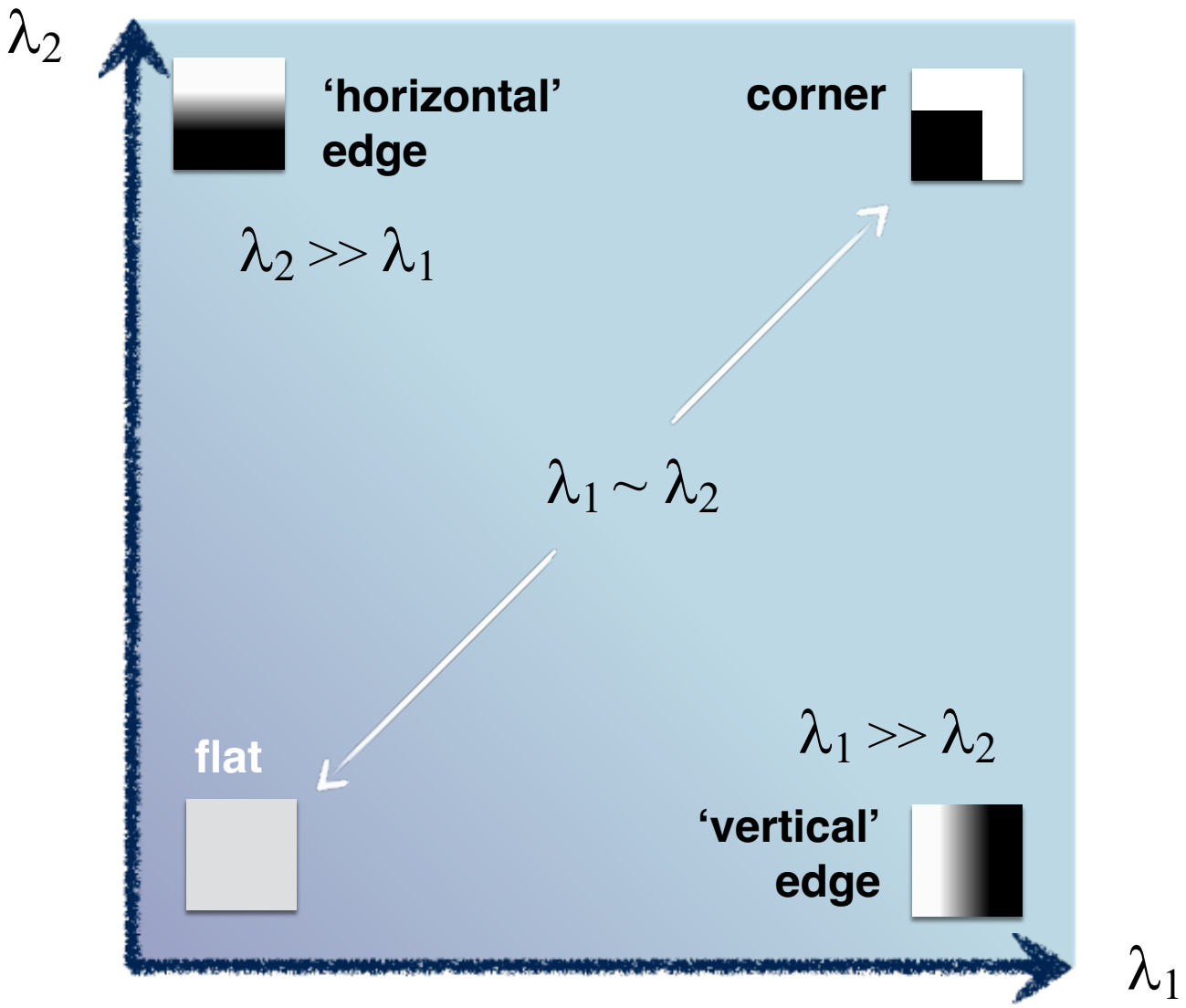
$$[u \quad v] \; M \; \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$



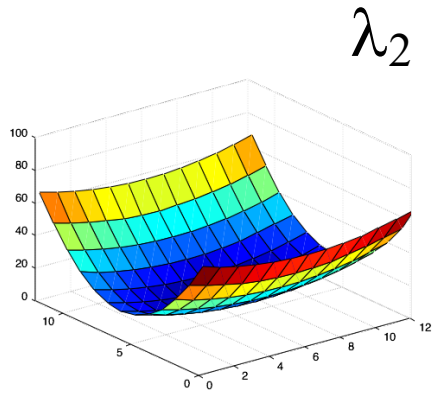
interpreting eigenvalues

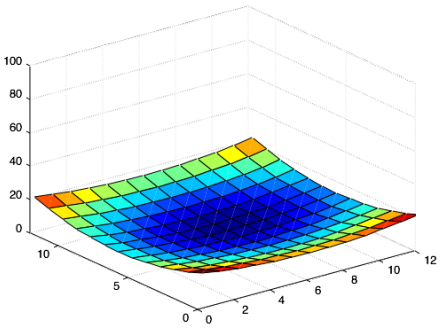


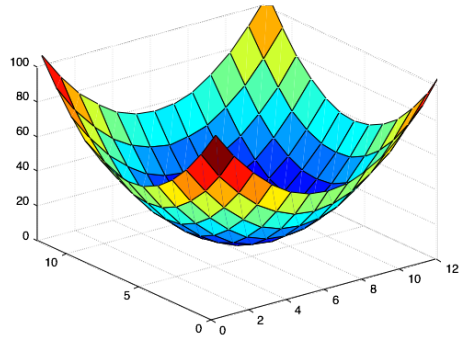
interpreting eigenvalues

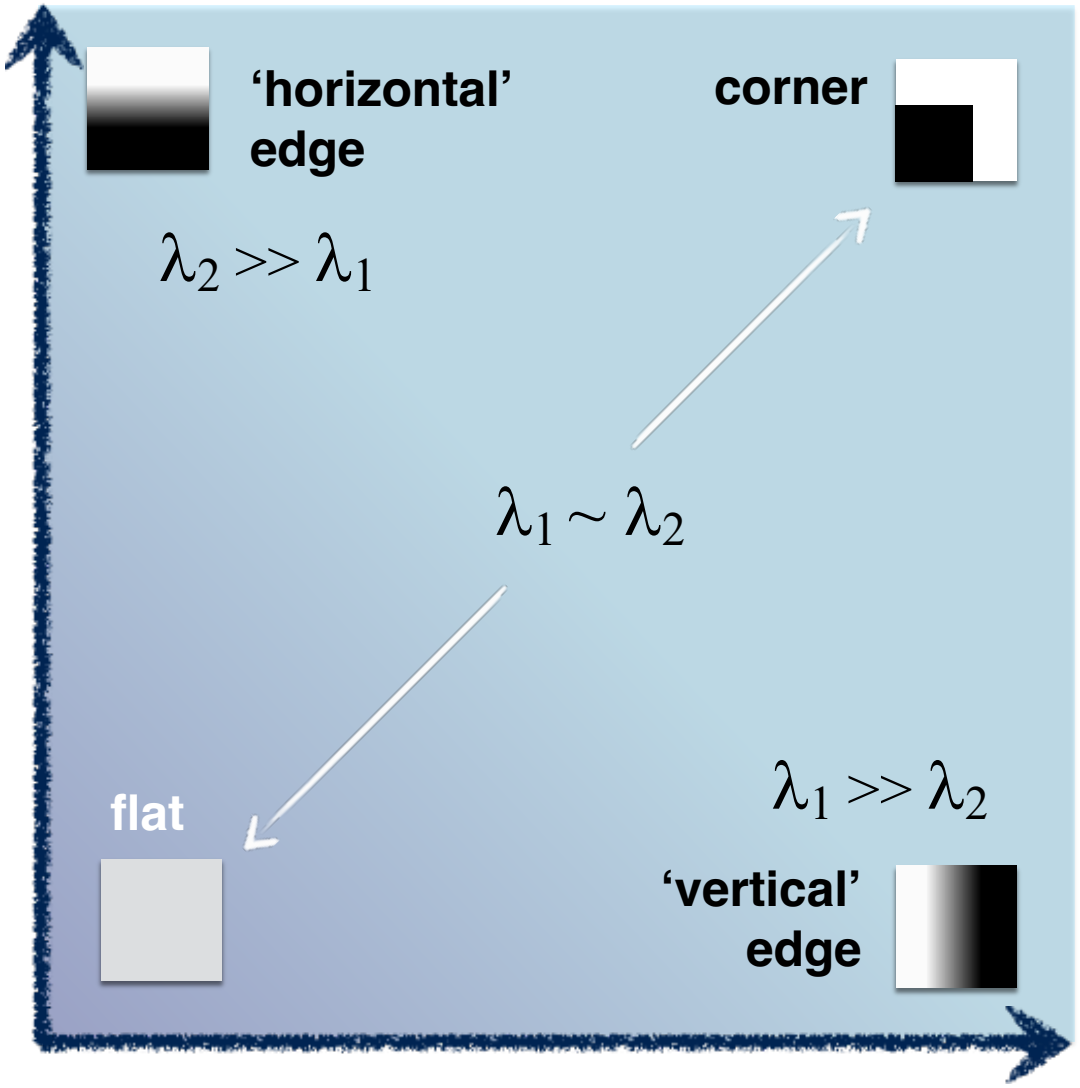


interpreting eigenvalues

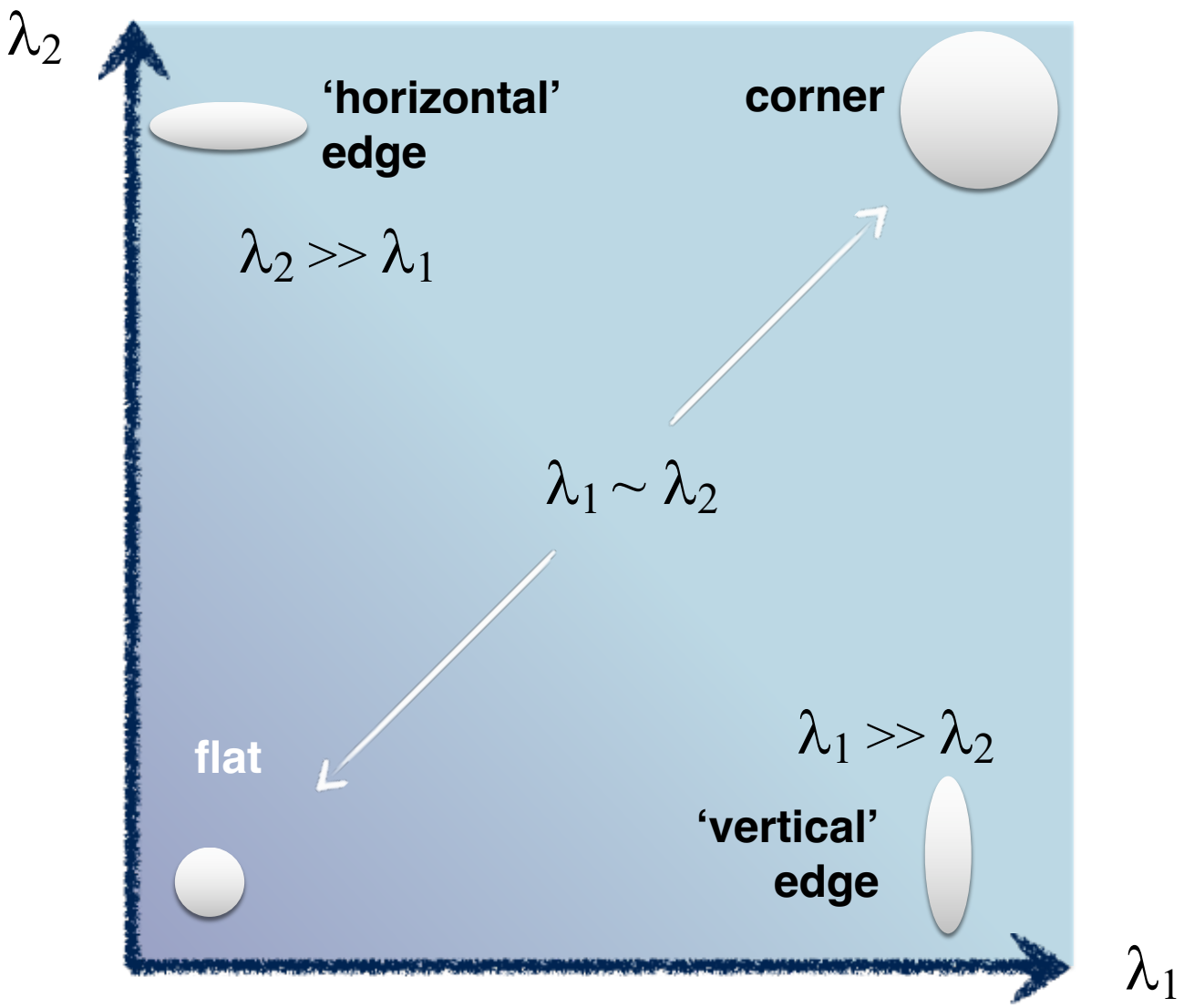




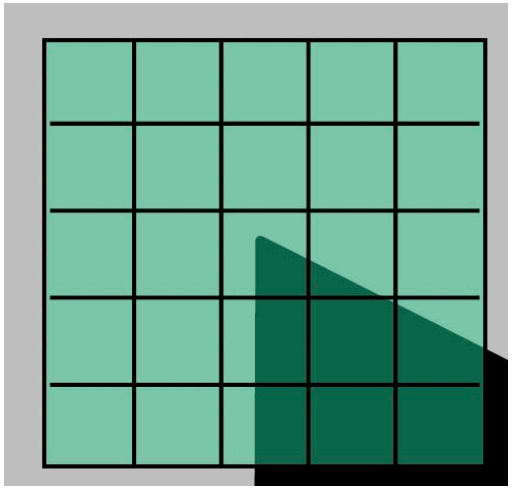




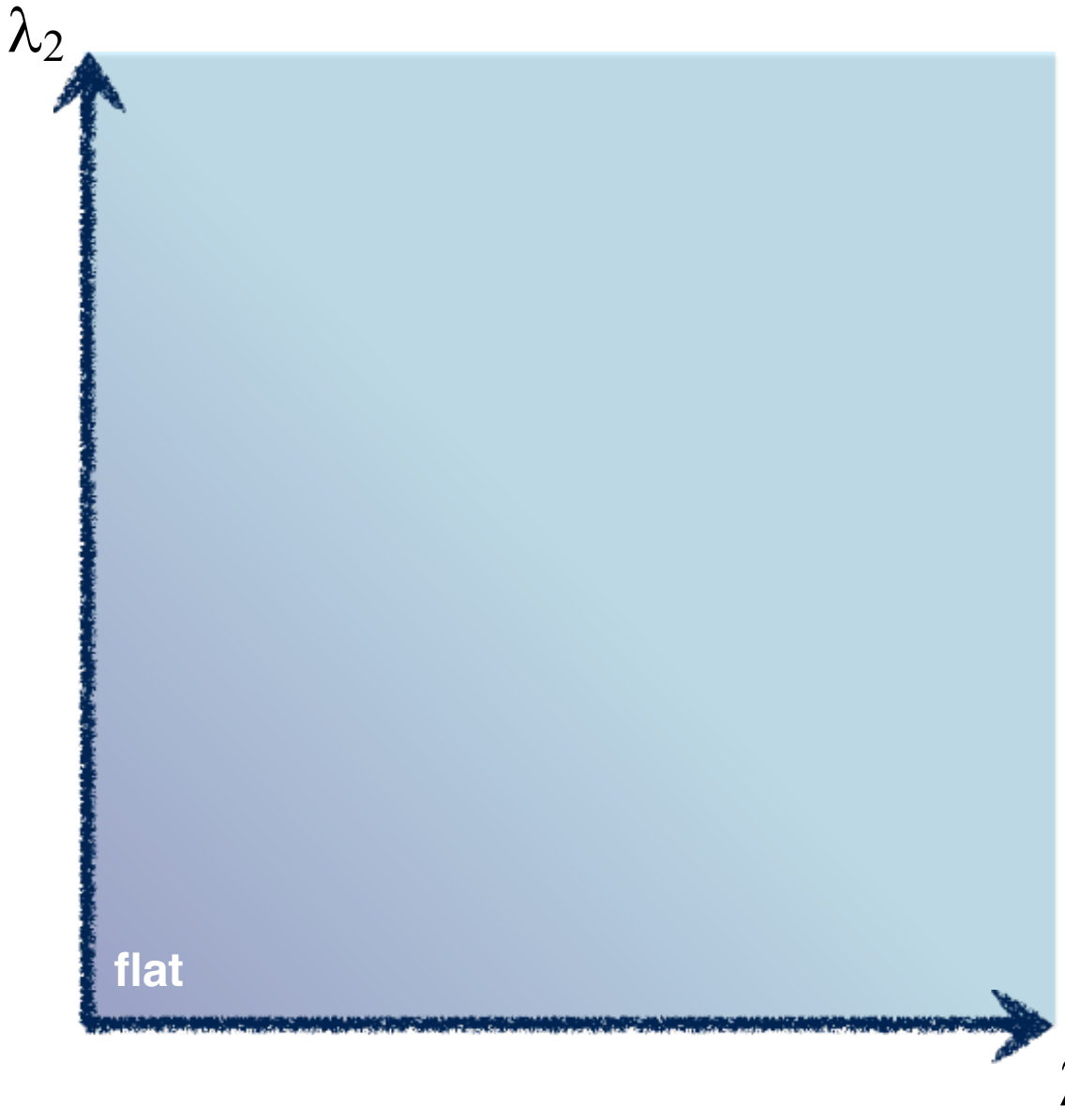
interpreting eigenvalues



5. Use threshold on eigenvalues to detect corners

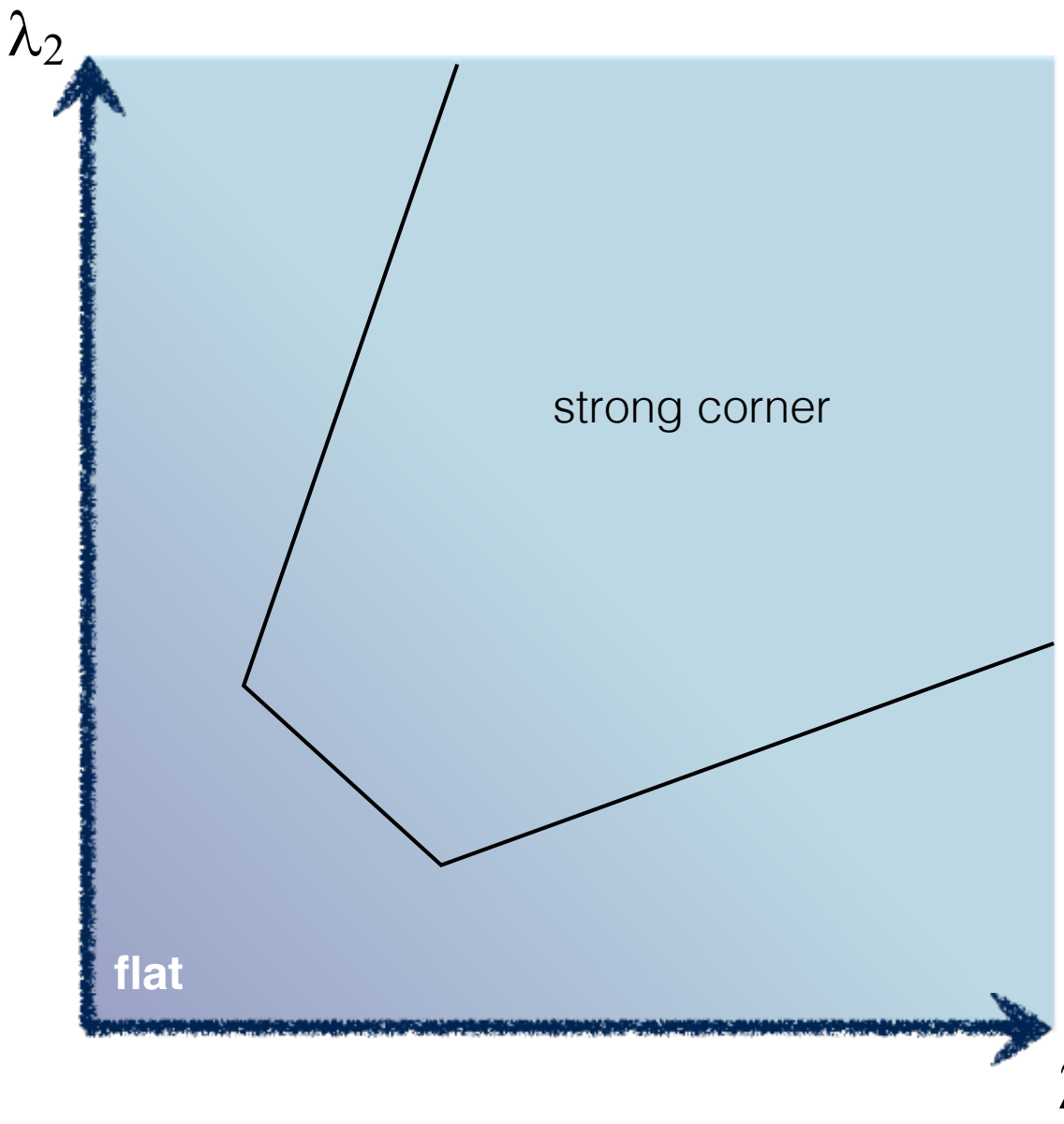


5. Use threshold on eigenvalues to detect corners



Think of a function to score ‘cornerness’

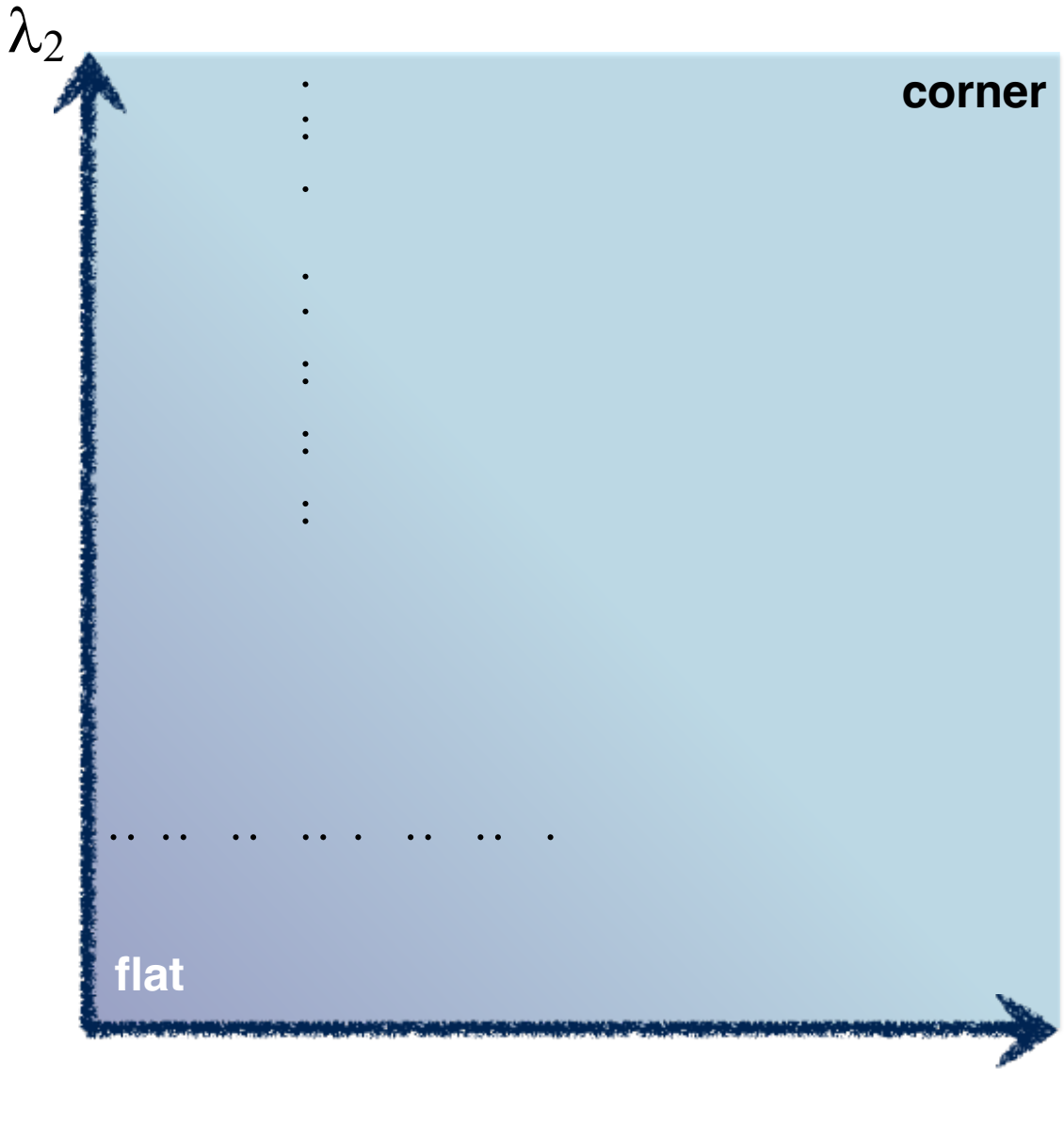
5. Use threshold on eigenvalues to detect corners



Think of a function to score ‘cornerness’

5. Use threshold on eigenvalues to detect corners

(a function of $\hat{}$)



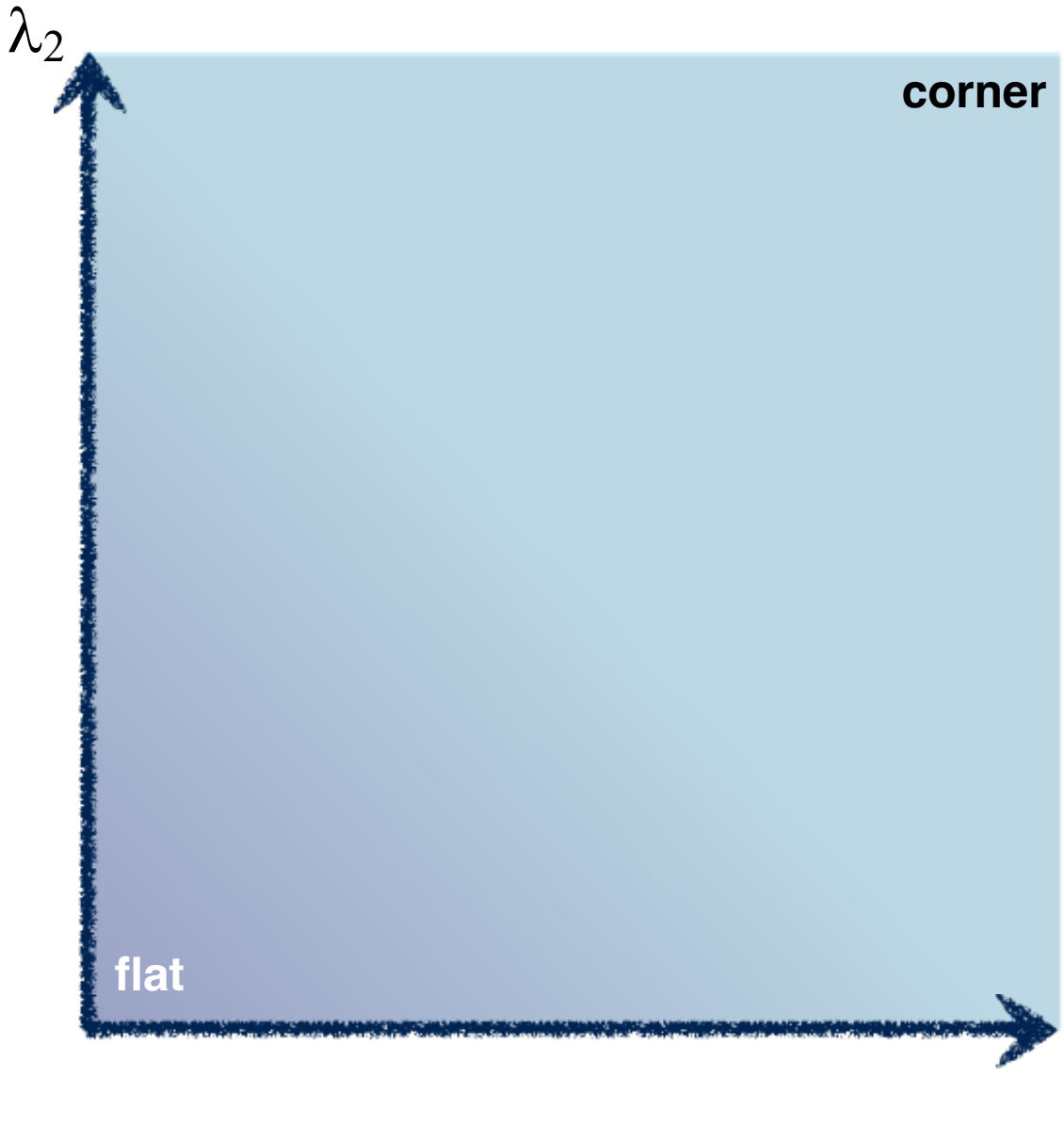
Use the smallest eigenvalue as the response function

$$R = \min(\lambda_1, \lambda_2)$$

5. Use threshold on eigenvalues to detect corners

(a function of)

https://en.wikipedia.org/wiki/Harris_corner_detector



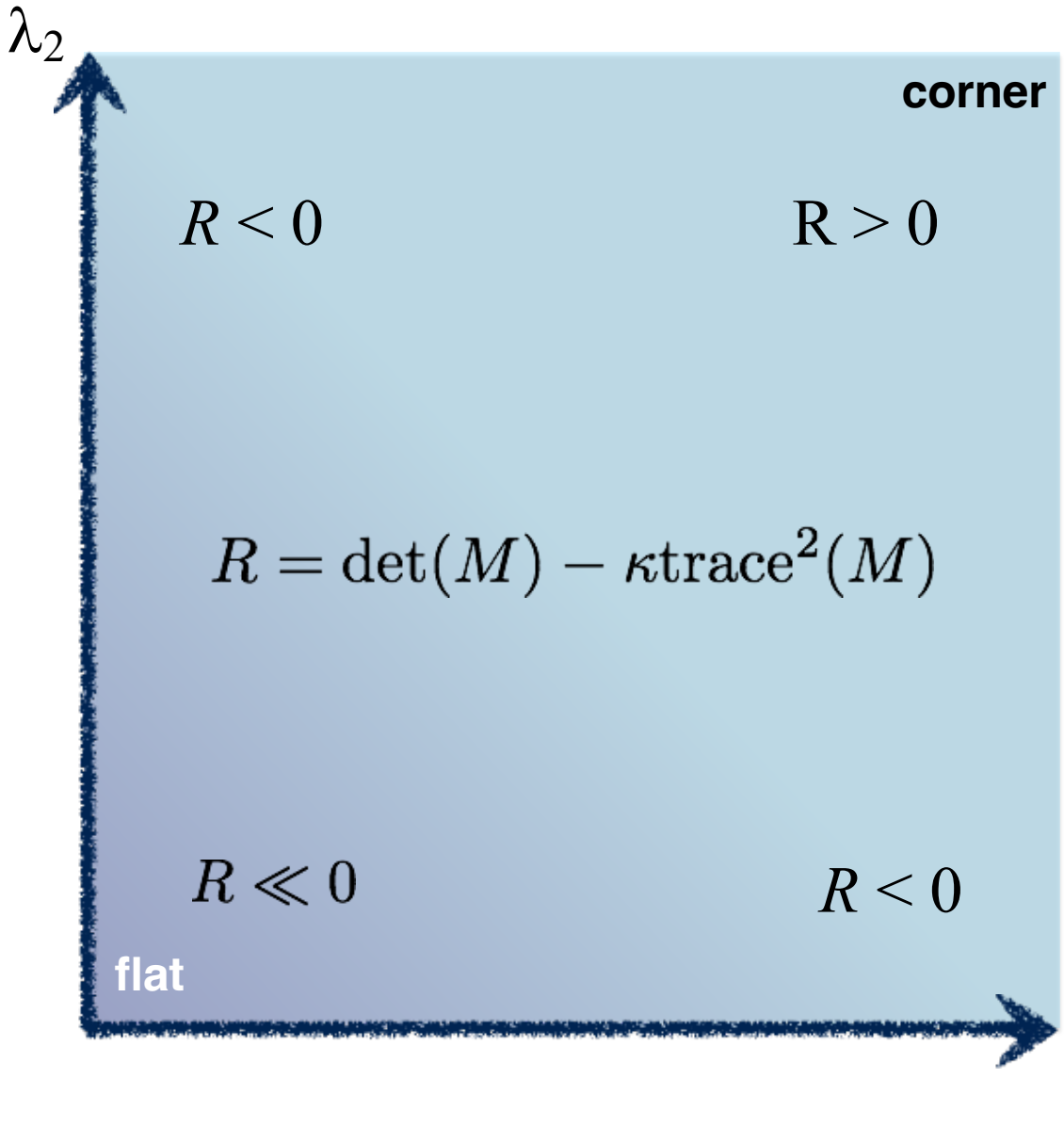
Eigenvalues need to be bigger than one.

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

Can compute this more efficiently…

5. Use threshold on eigenvalues to detect corners

(a function of)



$$\det M = \lambda_1 \lambda_2$$

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

$$det\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = ad - bc$$

$$trace\left(\begin{bmatrix} a & b \\ c & d \end{bmatrix}\right) = a + d$$

Harris & Stephens (1988)

$$R = \det(M) - \kappa \text{trace}^2(M)$$

Kanade & Tomasi (1994)

$$R = \min(\lambda_1, \lambda_2)$$

Nobel (1998)

$$R = \frac{\det(M)}{\text{trace}(M) + \epsilon}$$

Harris Detector

C.Harris and M.Stephens. “A Combined Corner and Edge Detector.”1988.

1. Compute x and y derivatives of image

$$I_x = G_\sigma^x * I \qquad I_y = G_\sigma^y * I$$

2. Compute products of derivatives at every pixel

$$I_{x^2} = I_x \cdot I_x \qquad I_{y^2} = I_y \cdot I_y \qquad I_{xy} = I_x \cdot I_y$$

3. Compute the sums of the products of derivatives at each pixel

$$S_{x^2} = G_{\sigma'} * I_{x^2} \qquad S_{y^2} = G_{\sigma'} * I_{y^2} \qquad S_{xy} = G_{\sigma'} * I_{xy}$$

Harris Detector

C.Harris and M.Stephens. “A Combined Corner and Edge Detector.”1988.

4. Define the matrix at each pixel

$$M(x,y) = \begin{bmatrix} S_{x^2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y^2}(x,y) \end{bmatrix}$$

5. Compute the response of the detector at each pixel

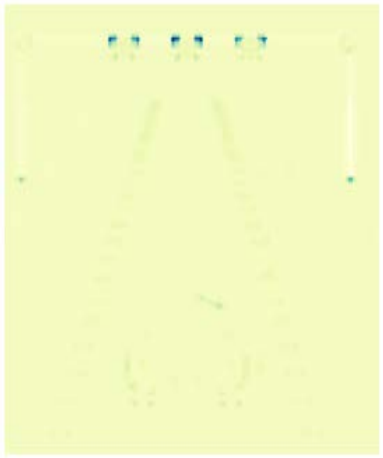
$$R = \det M - k\left(\text{trace}M\right)^2$$

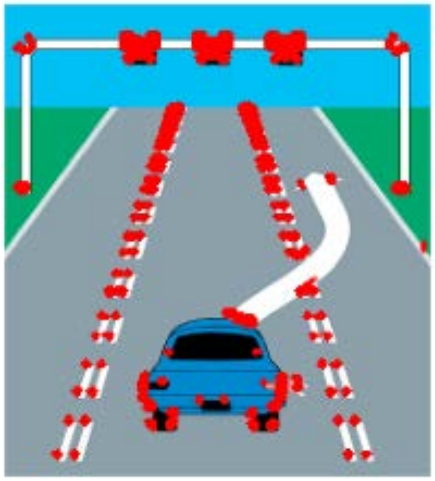
6. Threshold on value of R; compute non-max suppression.

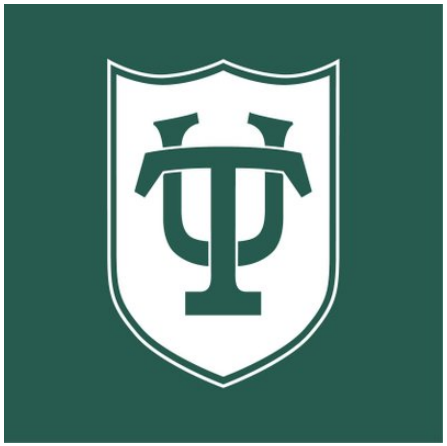
`cv2.cornerHarris`

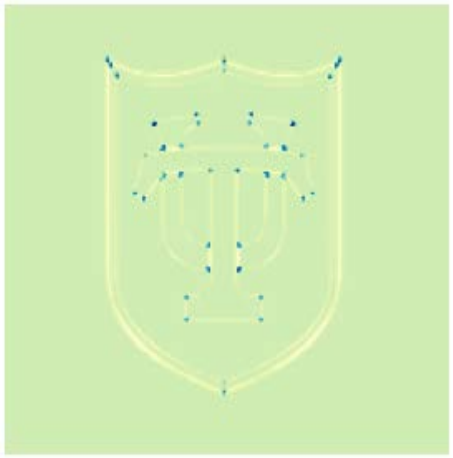
https://docs.opencv.org/3.4/dc/d0d/tutorial_py_features_harris.html

Harris Detector

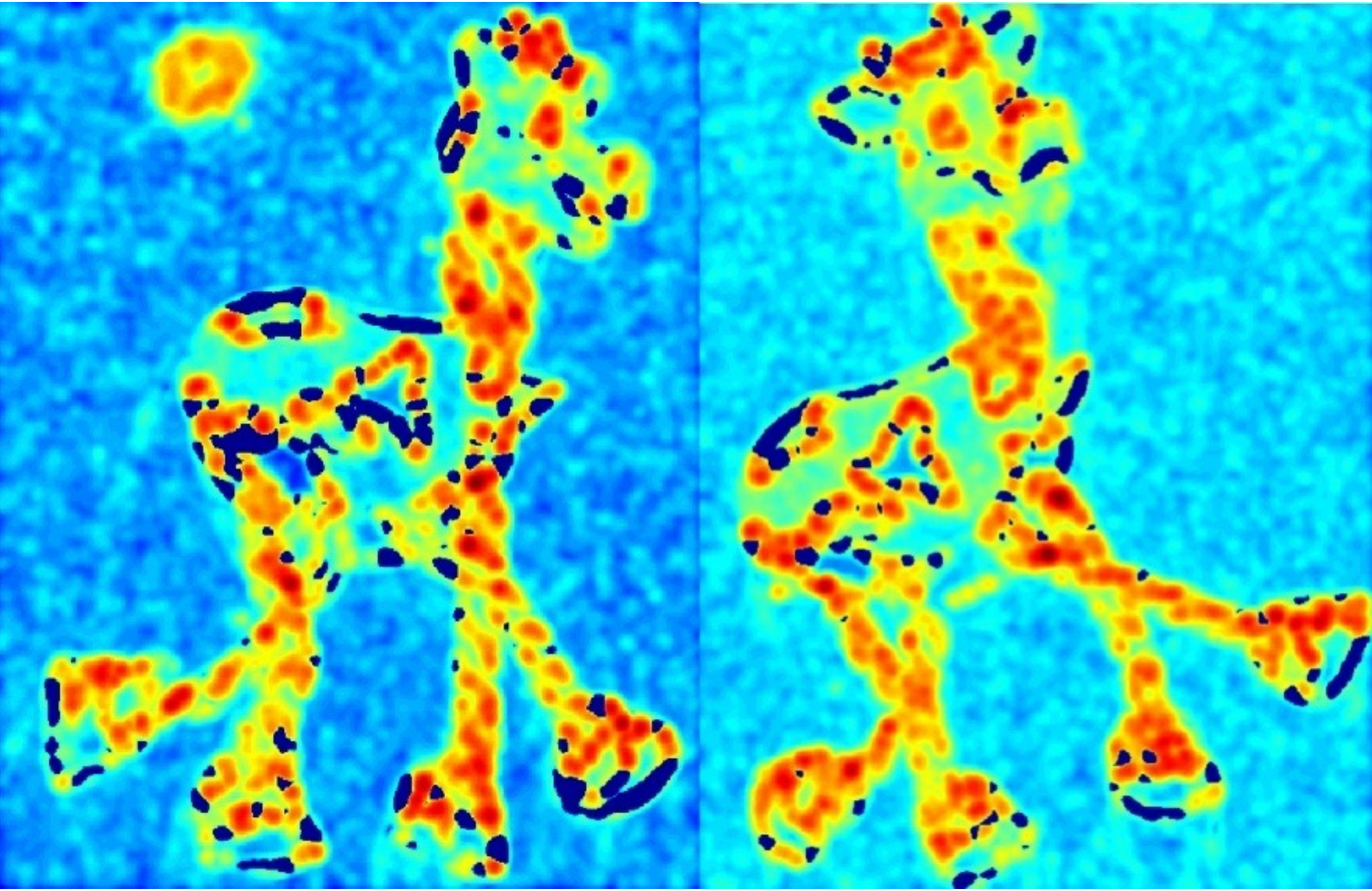




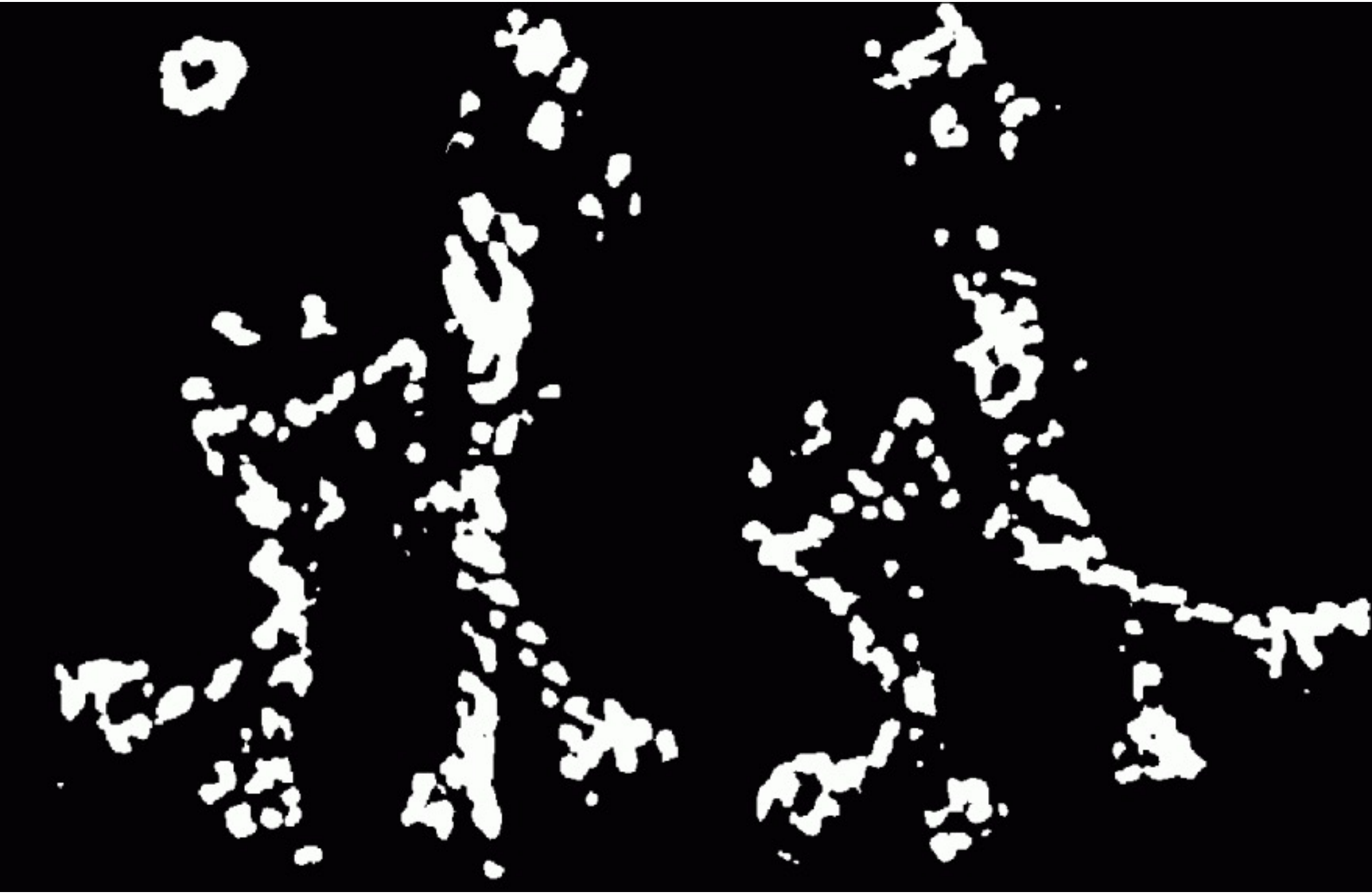




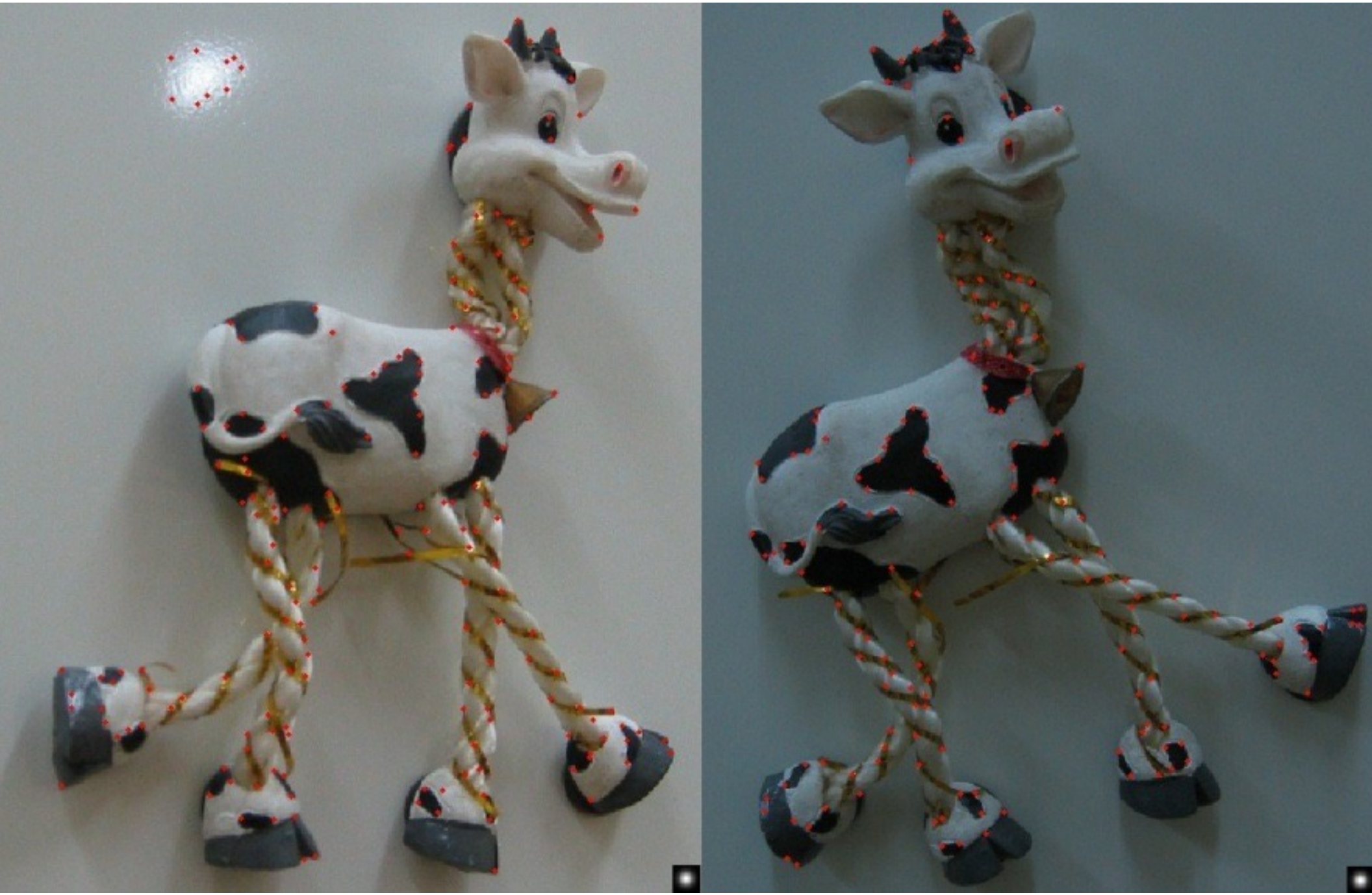
Corner response



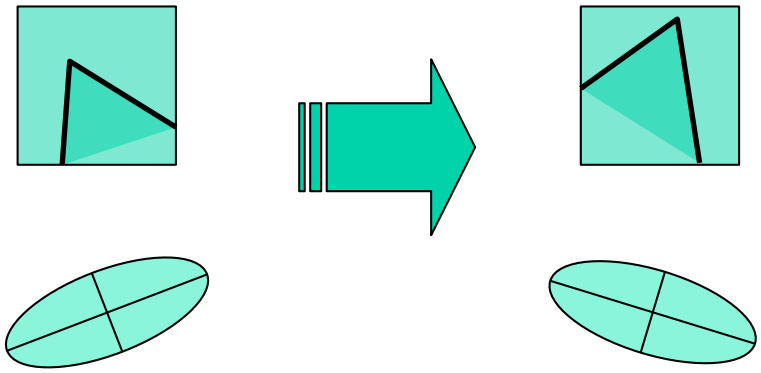
Thresholded corner response



Non-maximal suppression



Harris corner response is invariant to rotation



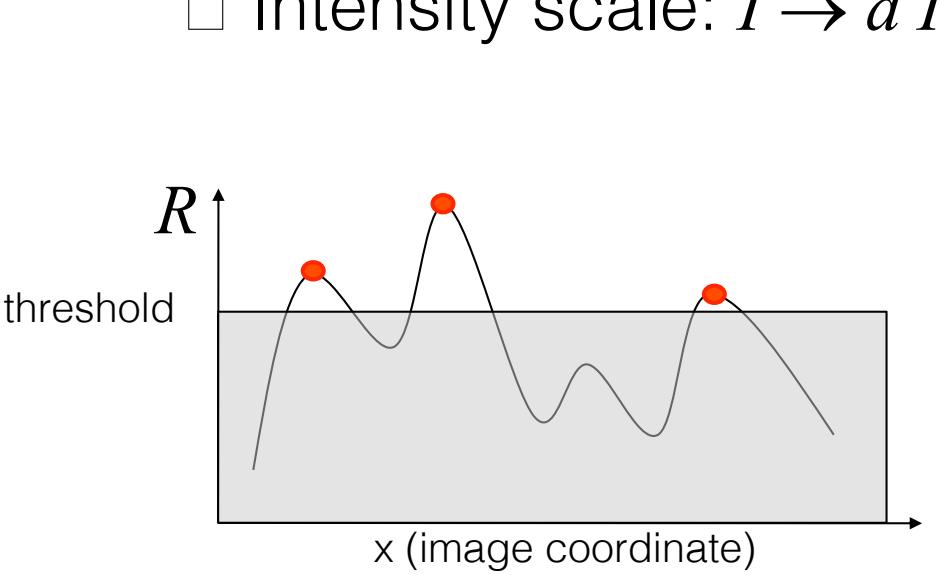
Ellipse rotates but its shape (**eigenvalues**) remains the same

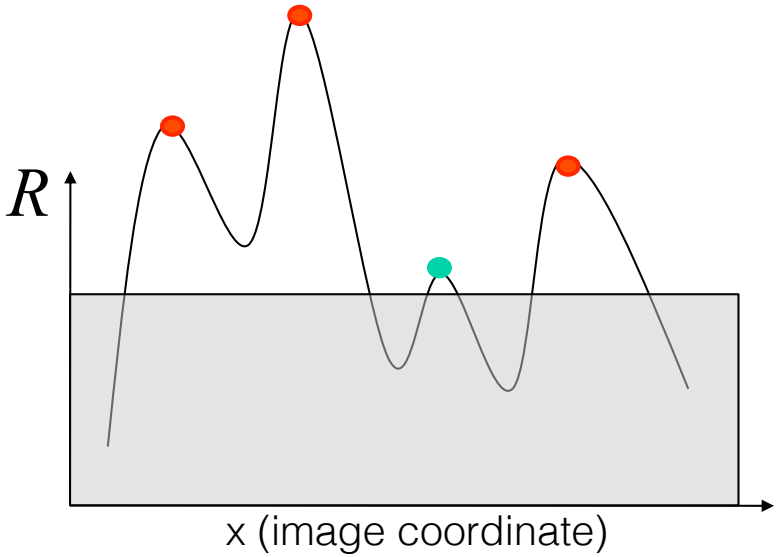
Corner response R is invariant to image rotation

Harris corner response is invariant to intensity changes

Partial invariance to ***affine intensity*** change

- Only derivatives are used => invariance to intensity shift $I \rightarrow I + b$
- Intensity scale: $I \rightarrow a\, I$

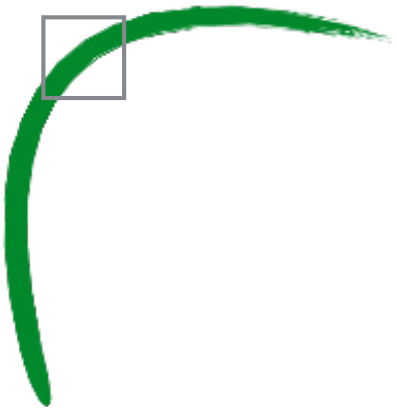




The Harris detector is not invariant to changes in …

The Harris corner detector is not invariant to scale

edge!



corner!