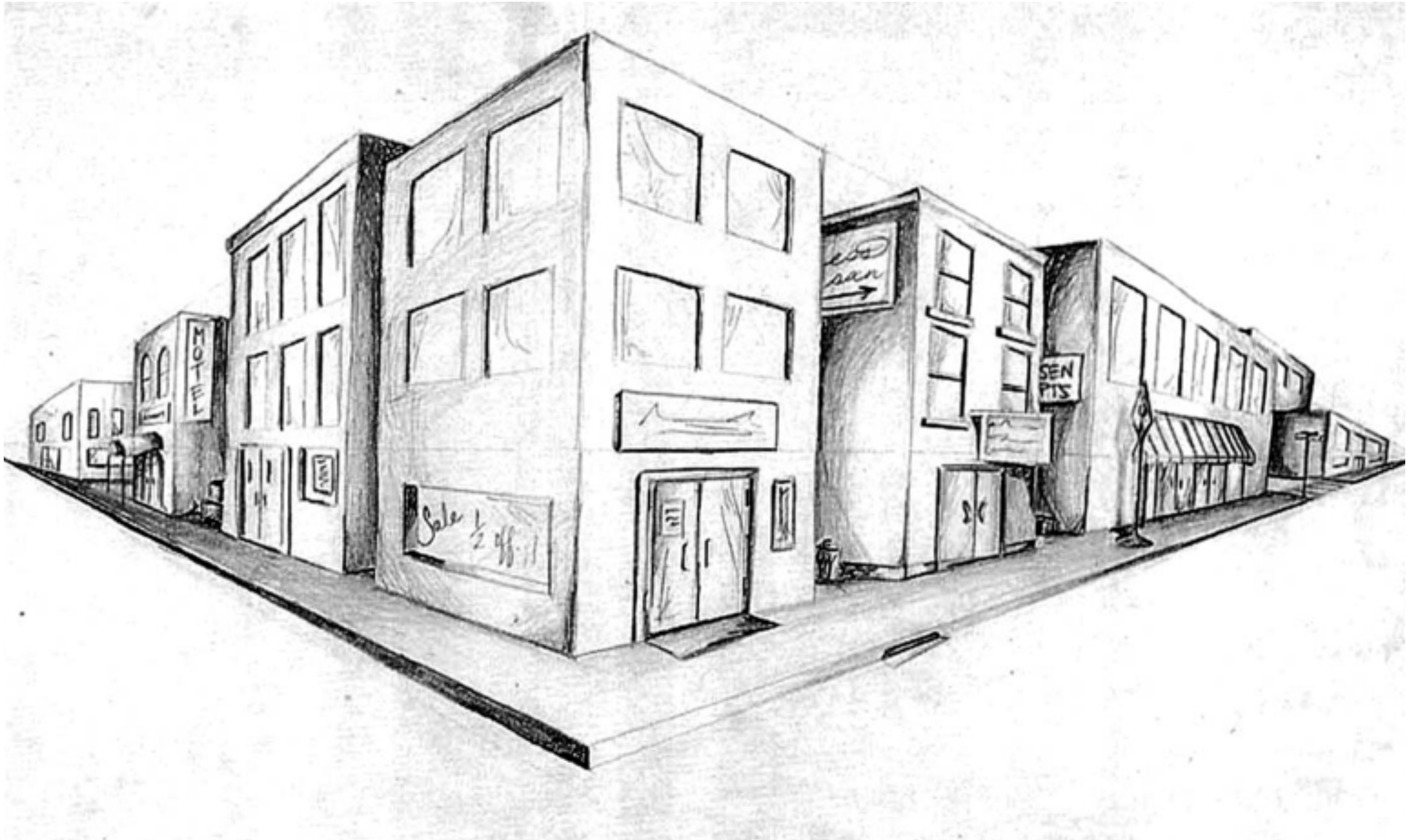


Detecting Corners

16-385 Computer Vision

Carnegie Mellon University (Kris Kitani)

Why detect corners?

Image alignment (homography, fundamental matrix)

3D reconstruction

Motion tracking

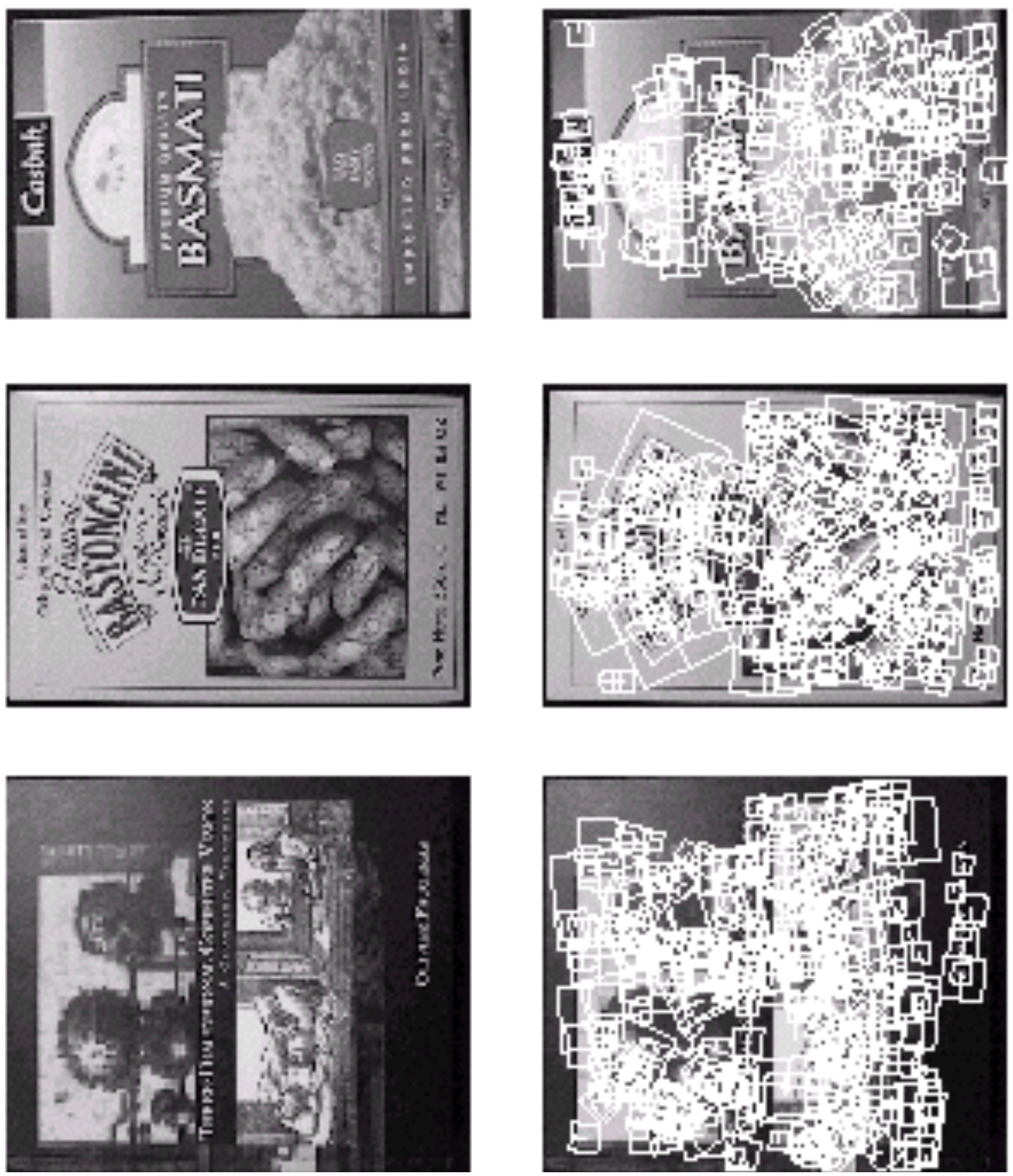
Object recognition

Indexing and database retrieval

Robot navigation

Planar object instance recognition

Database of planar objects

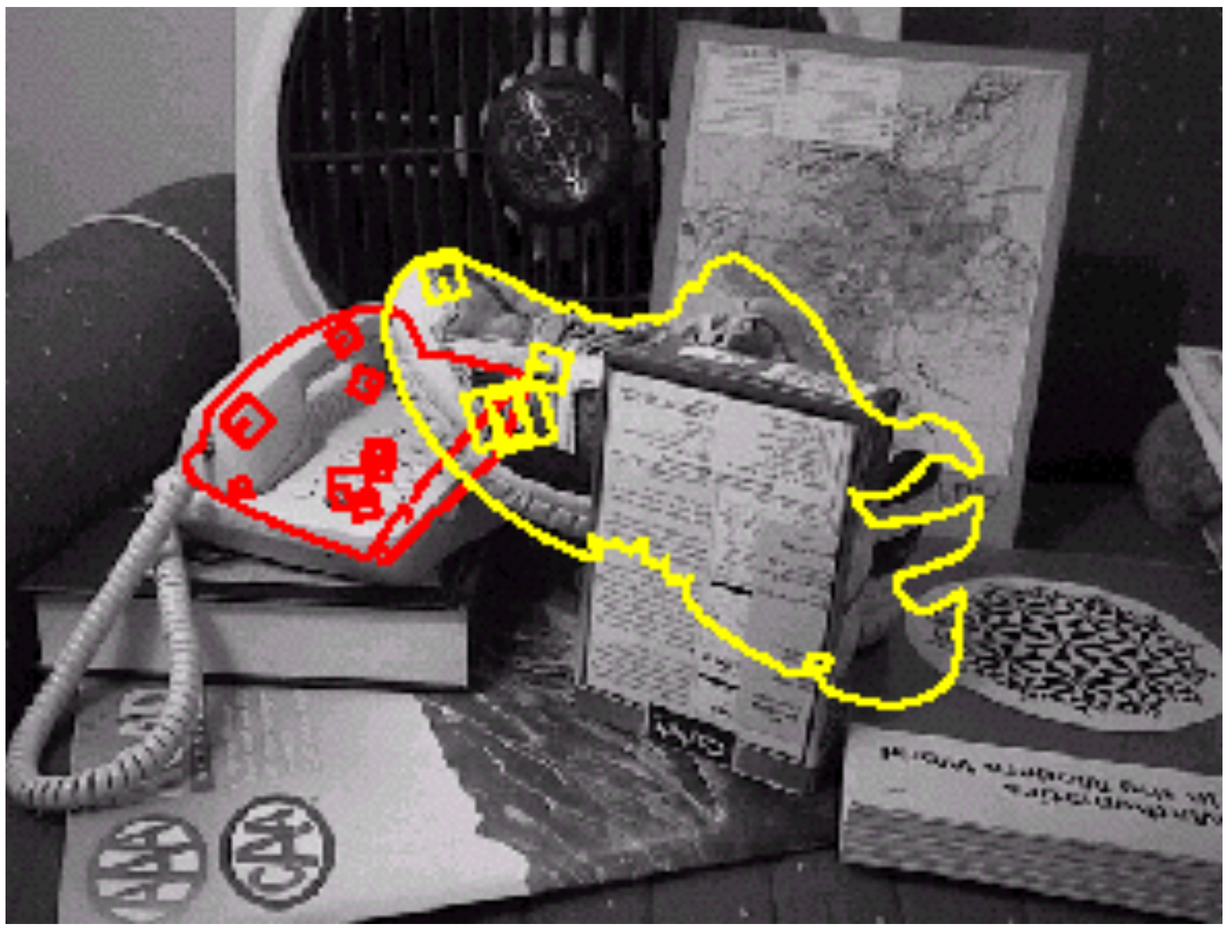


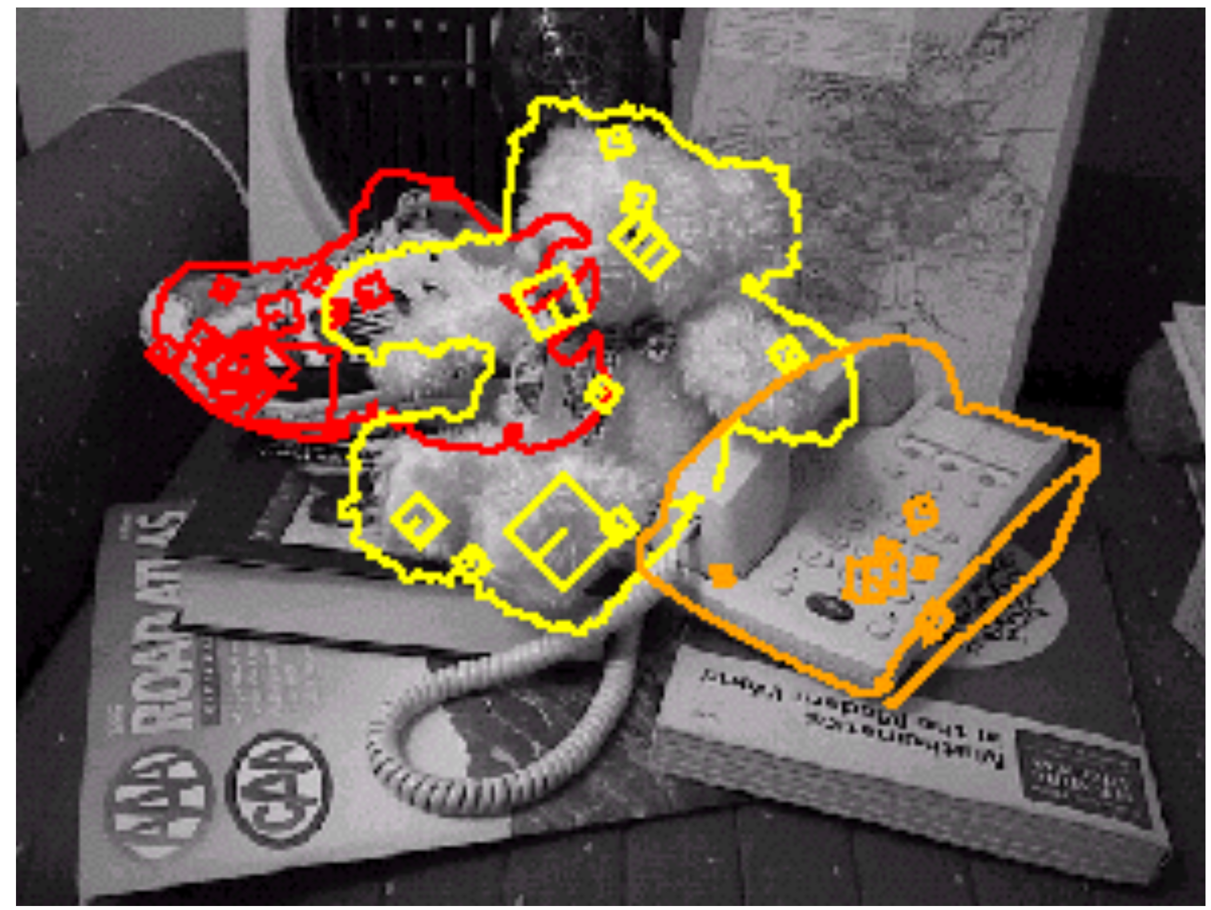
Instance recognition

3D object recognition

Database of 3D objects

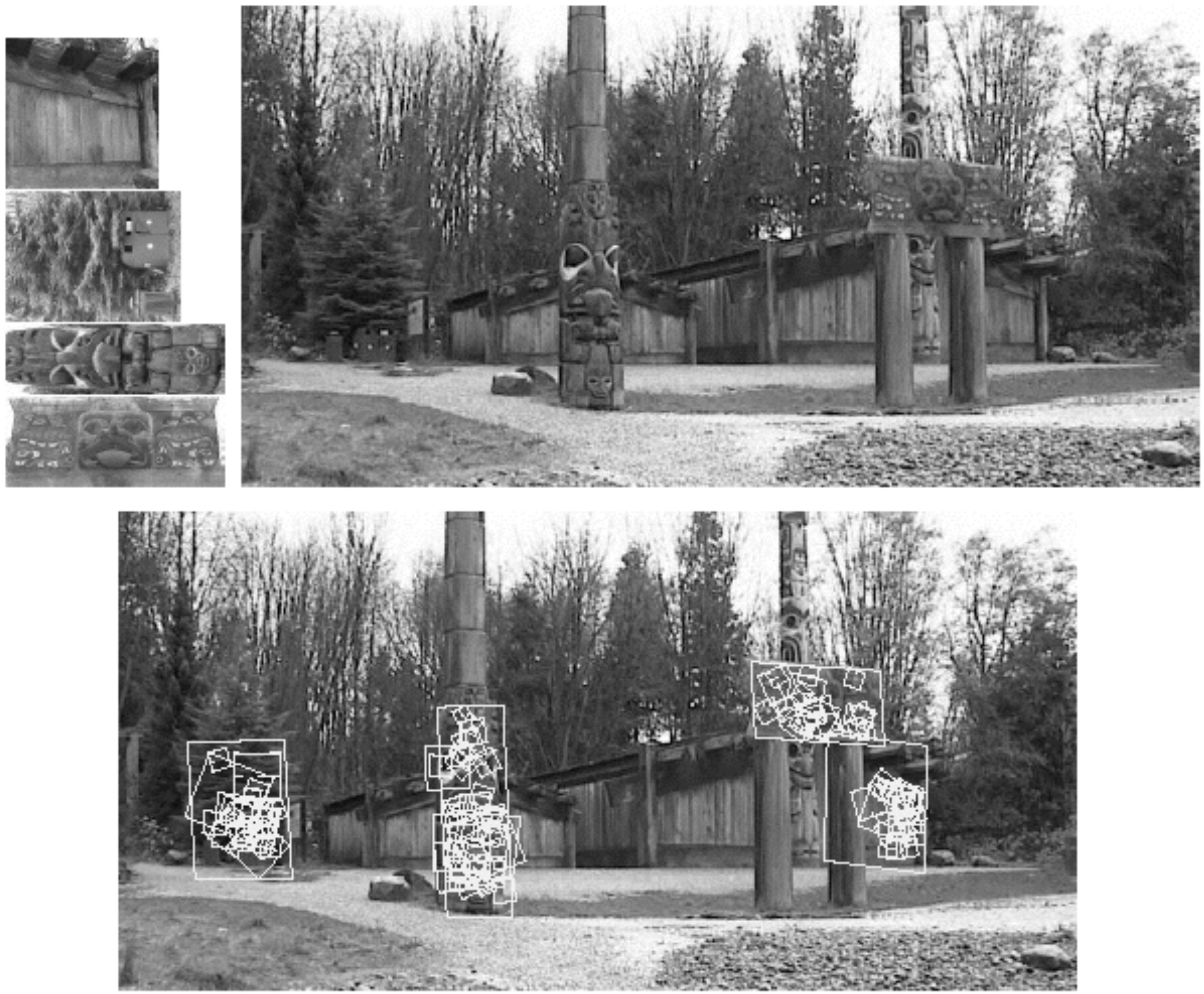
3D objects recognition



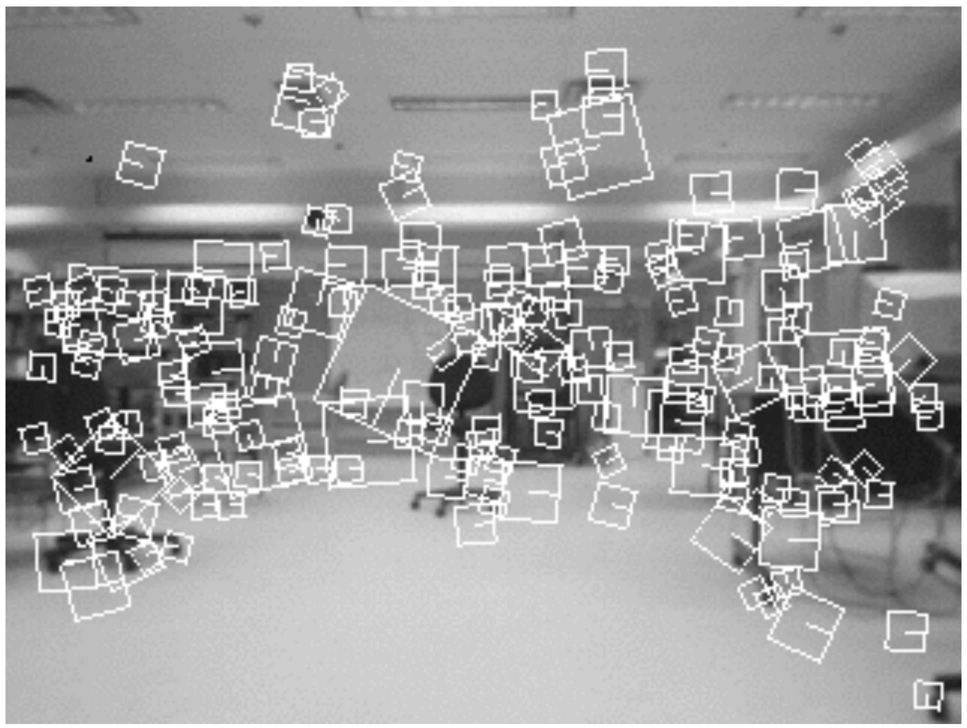


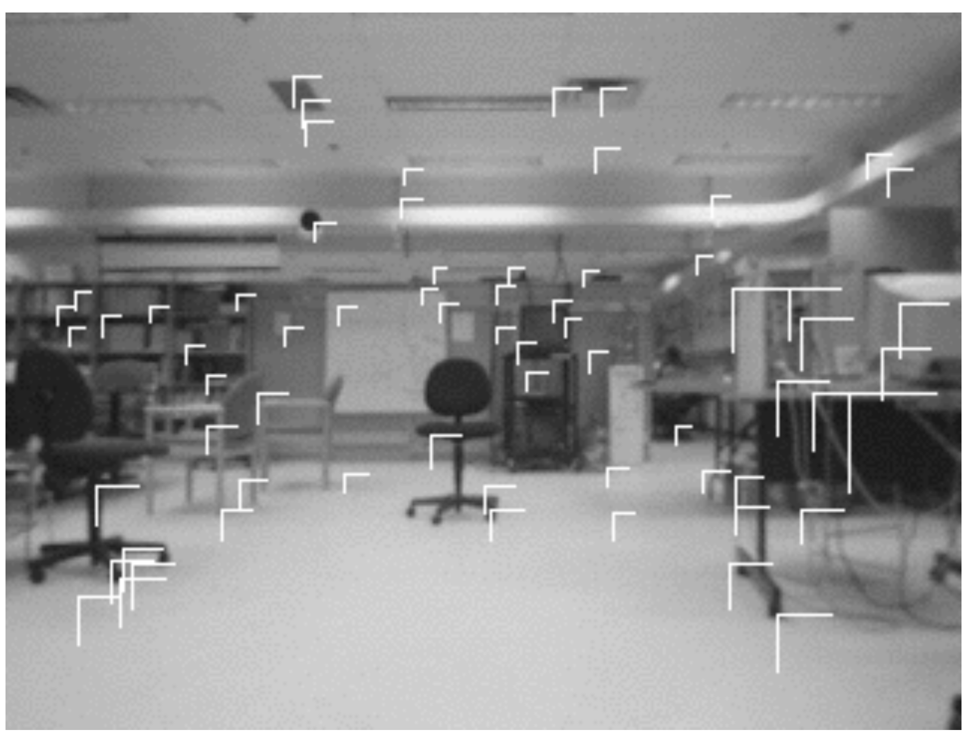
Recognition under occlusion

Location Recognition

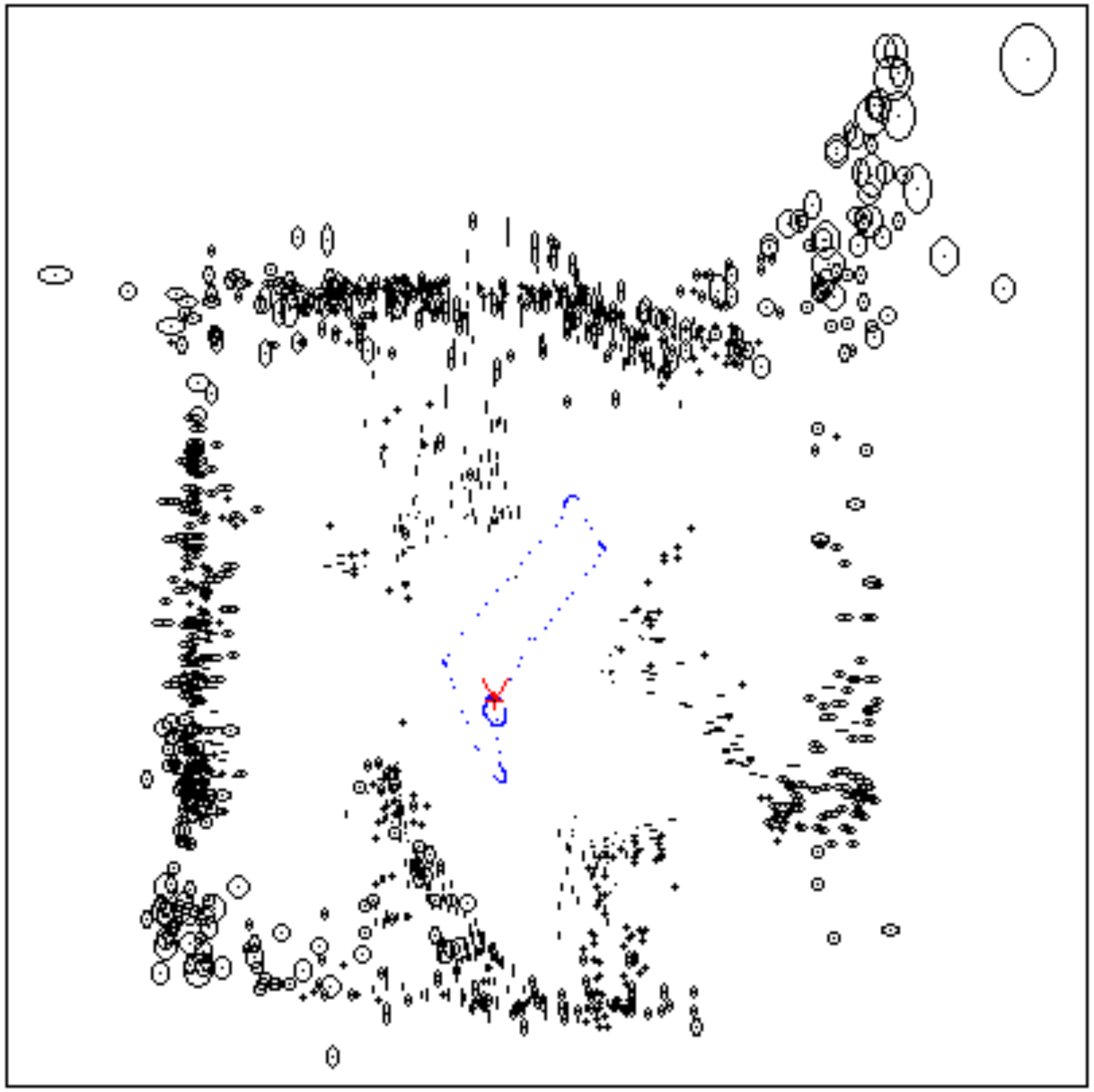


Robot Localization



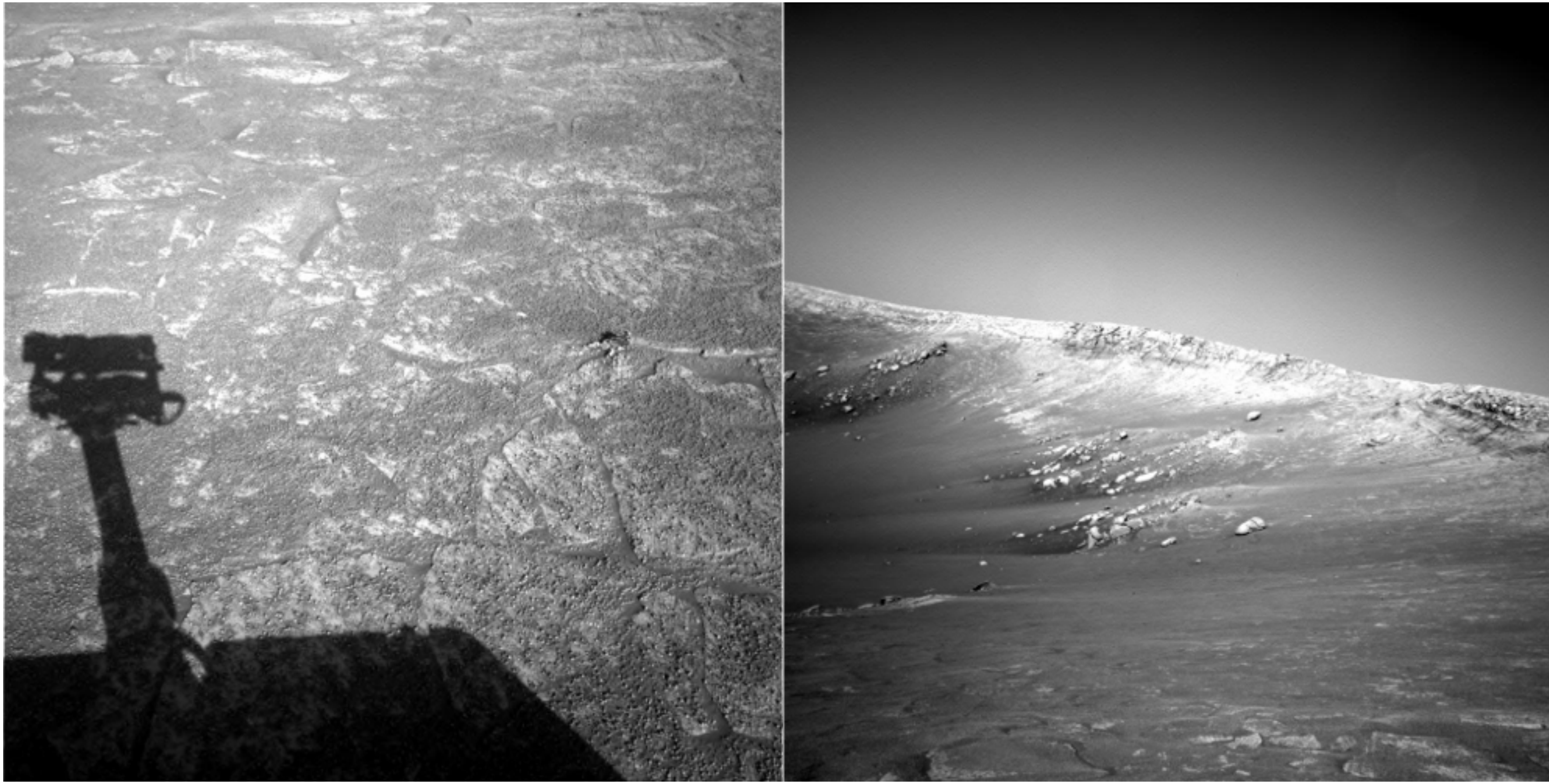


Map built over time



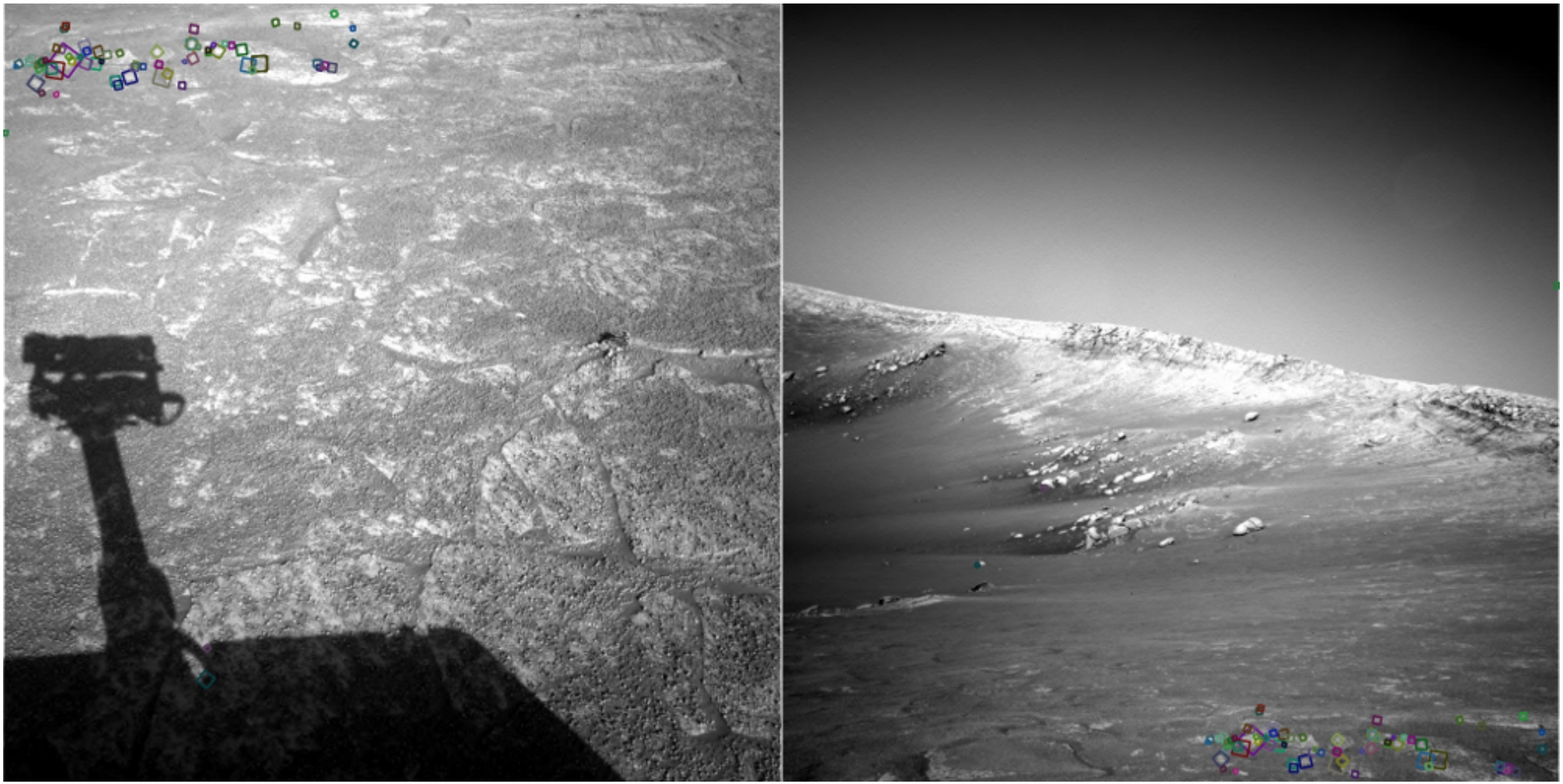
Example: Image Matching

How would you find corresponding points?



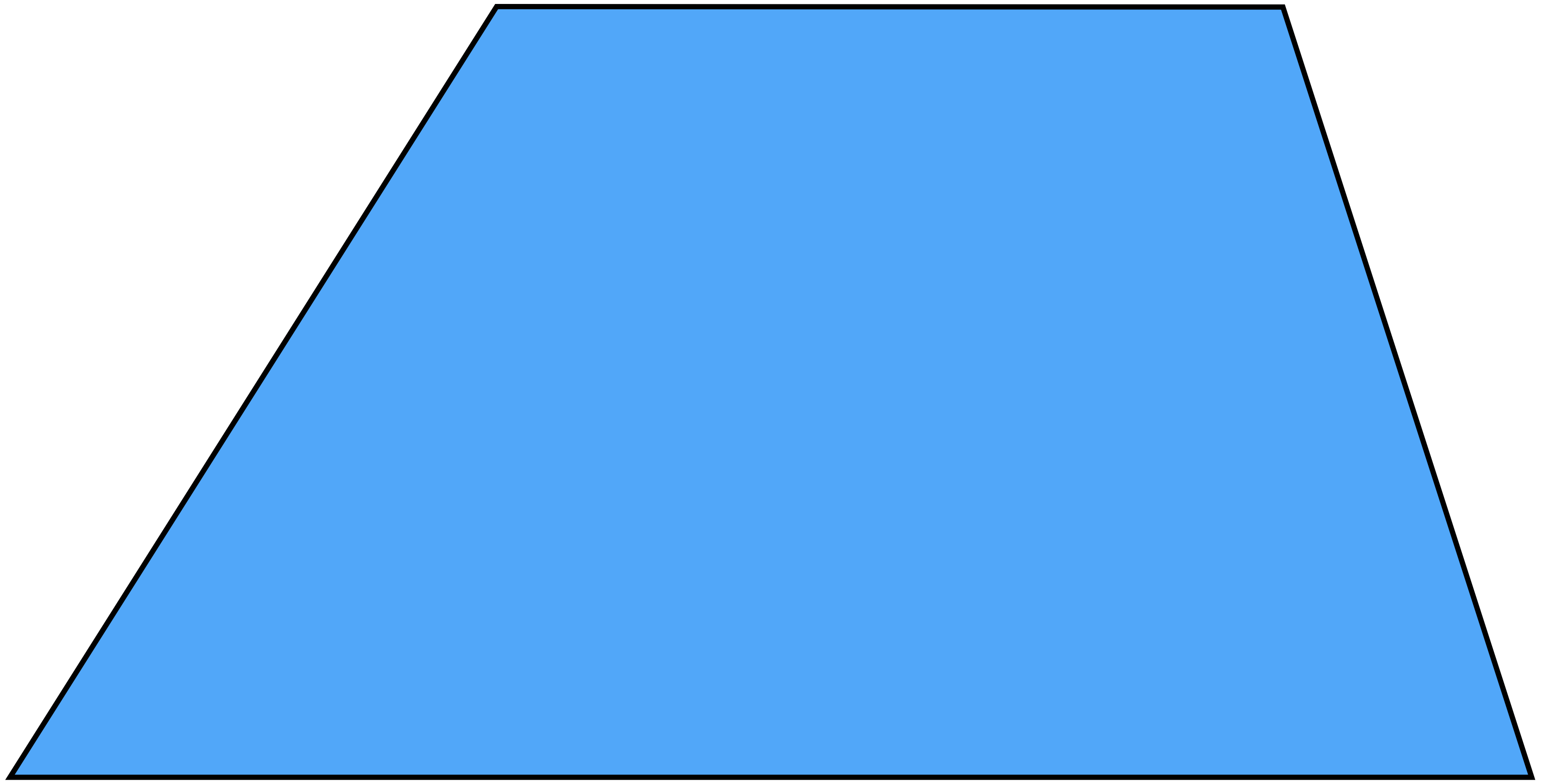
NASA Mars Rover images

Where are the corresponding points?



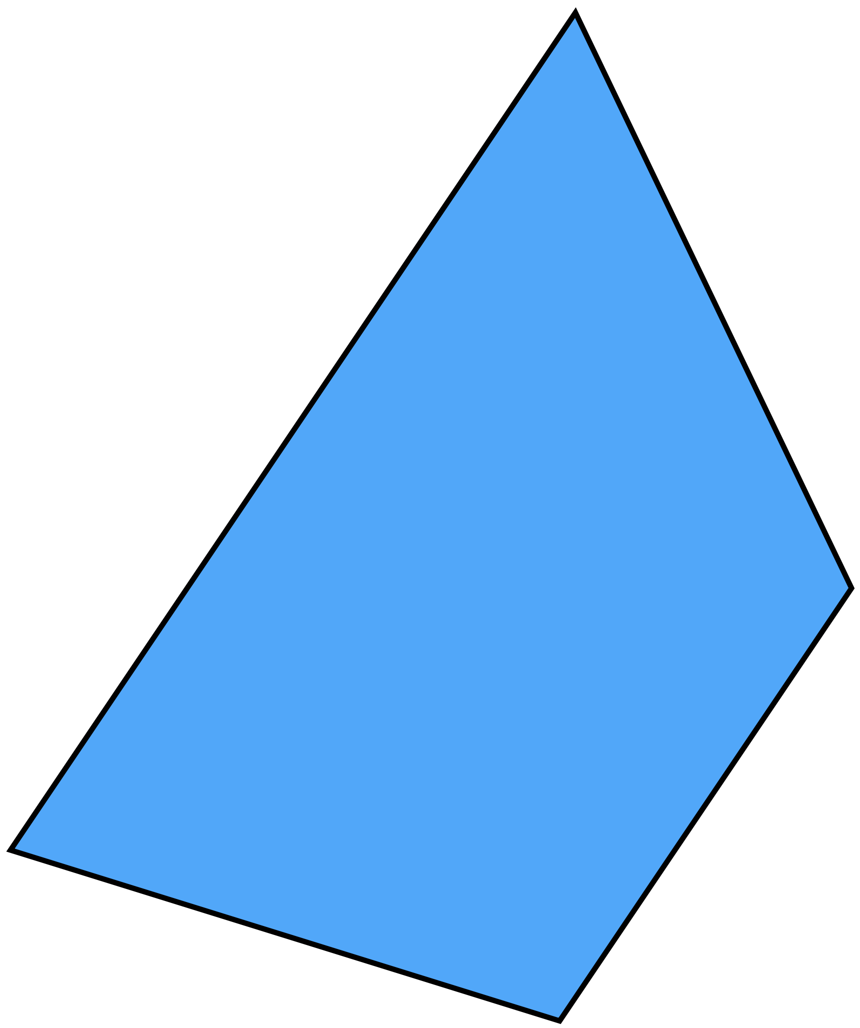
What type of features were you trying to match?

Explain to me your thought process.



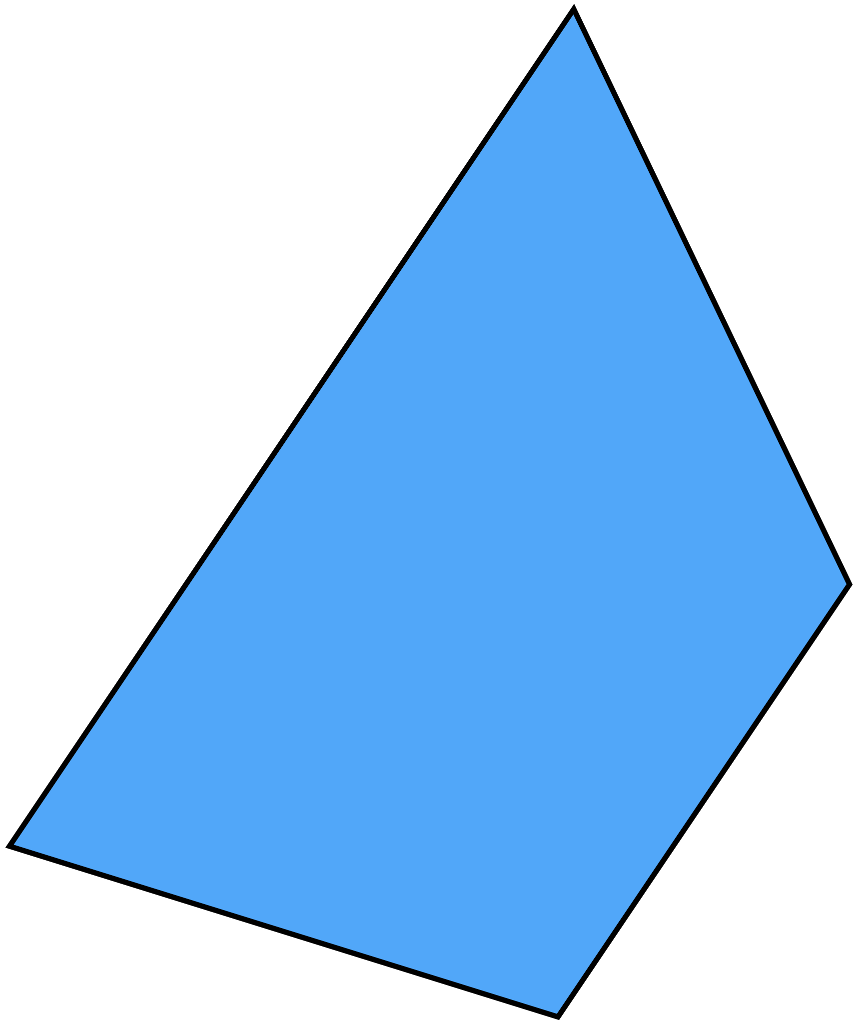
Pick a point in the image.
Find it again in the next image.

What type of feature would you select?



Pick a point in the image.
Find it again in the next image.

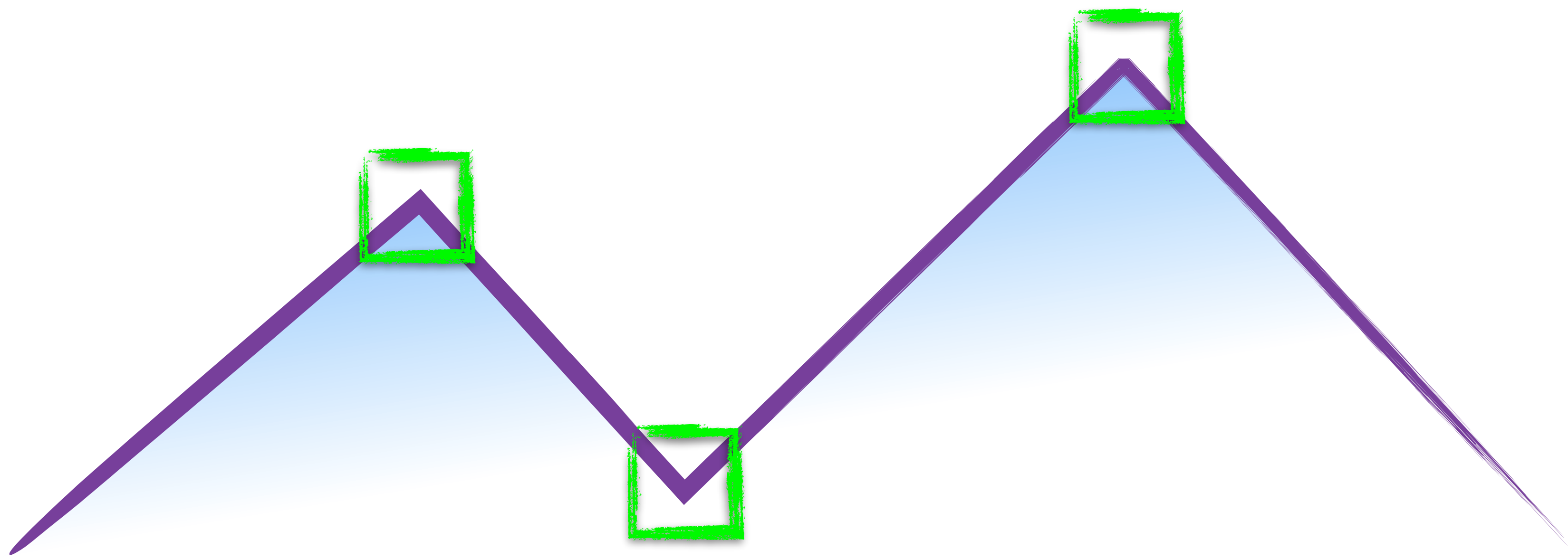
What type of feature would you select?



Pick a point in the image.
Find it again in the next image.

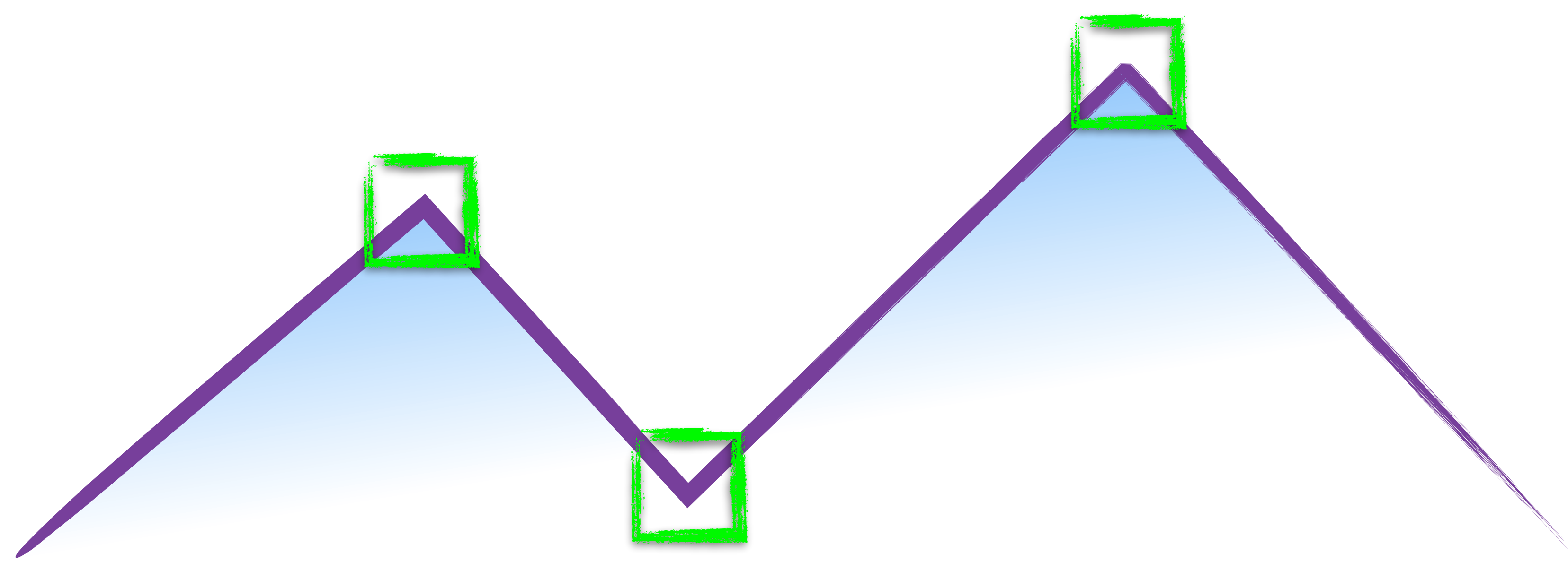
What type of feature would you select?
a corner

How do you find a corner?



How do you find a corner?

[Moravec 1980]

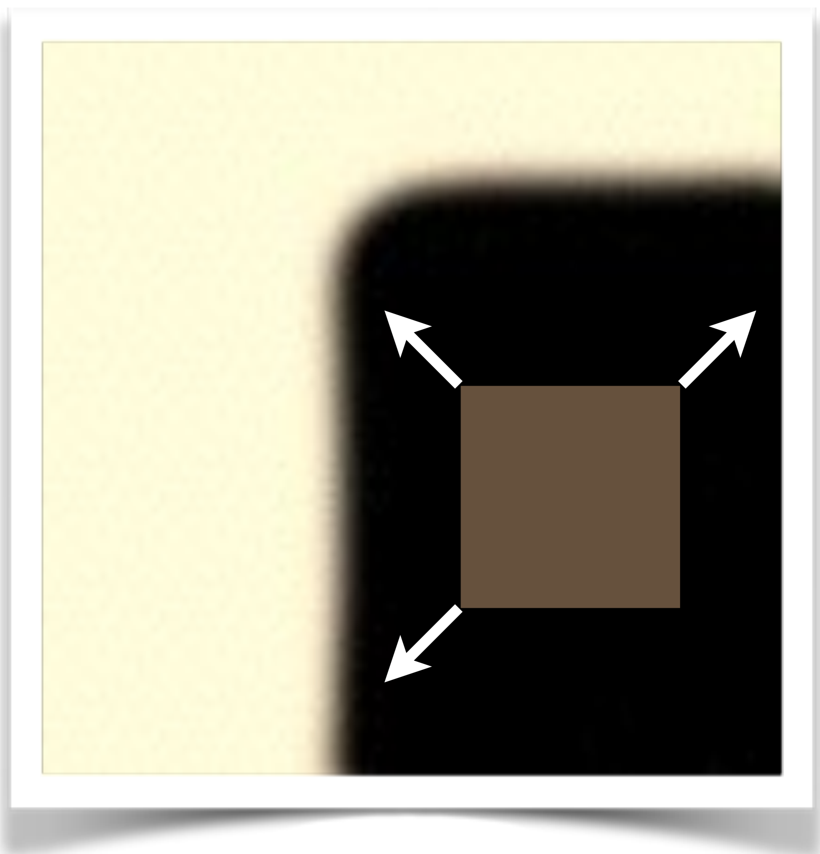


Easily recognized by looking through a small window

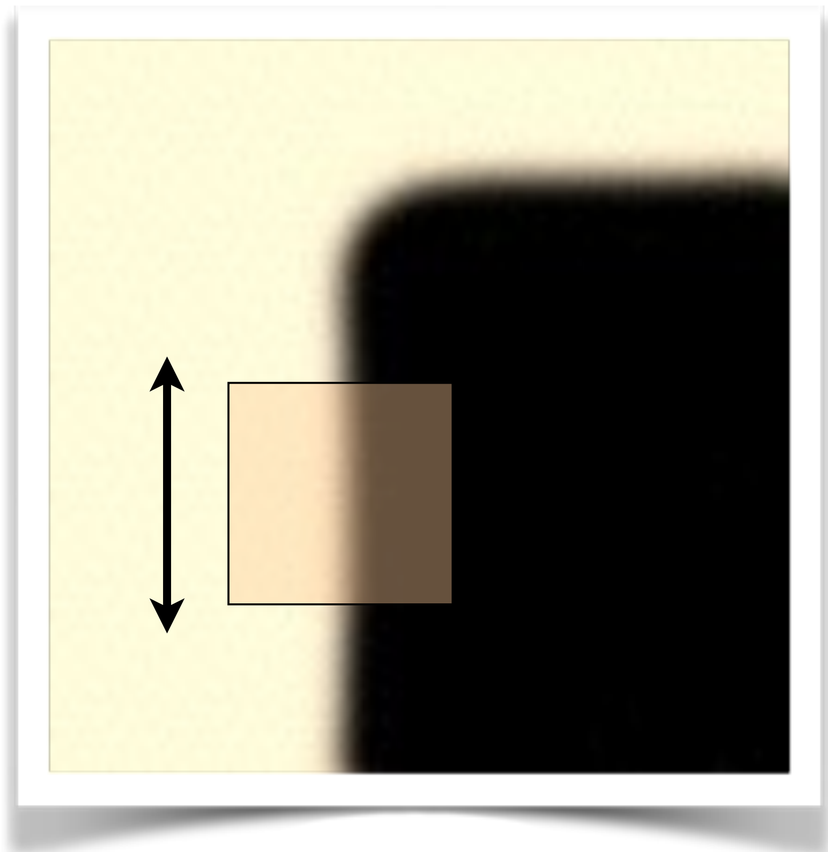
Shifting the window should give large change in intensity

Easily recognized by looking through a small window

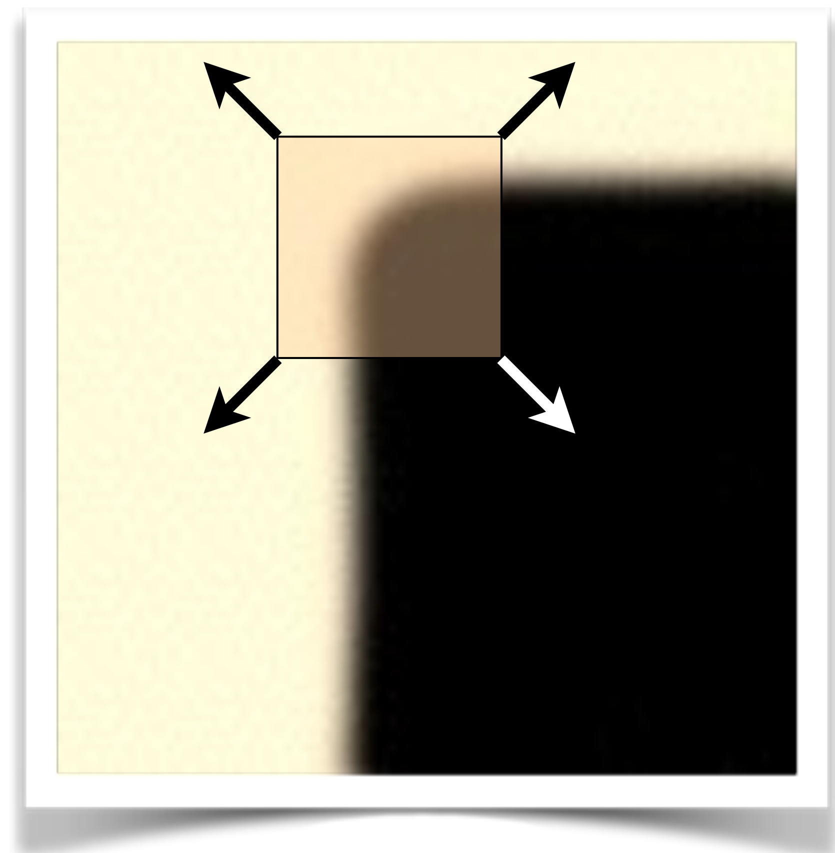
Shifting the window should give large change in intensity



"flat" region:
no change in all directions



"edge":
no change along the edge direction



"corner":
significant change in all directions

[Moravec 1980]

Design a program to detect corners

(hint: use image gradients)

Finding corners
(a.k.a. PCA)

1. Compute image gradients over small region
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

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

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

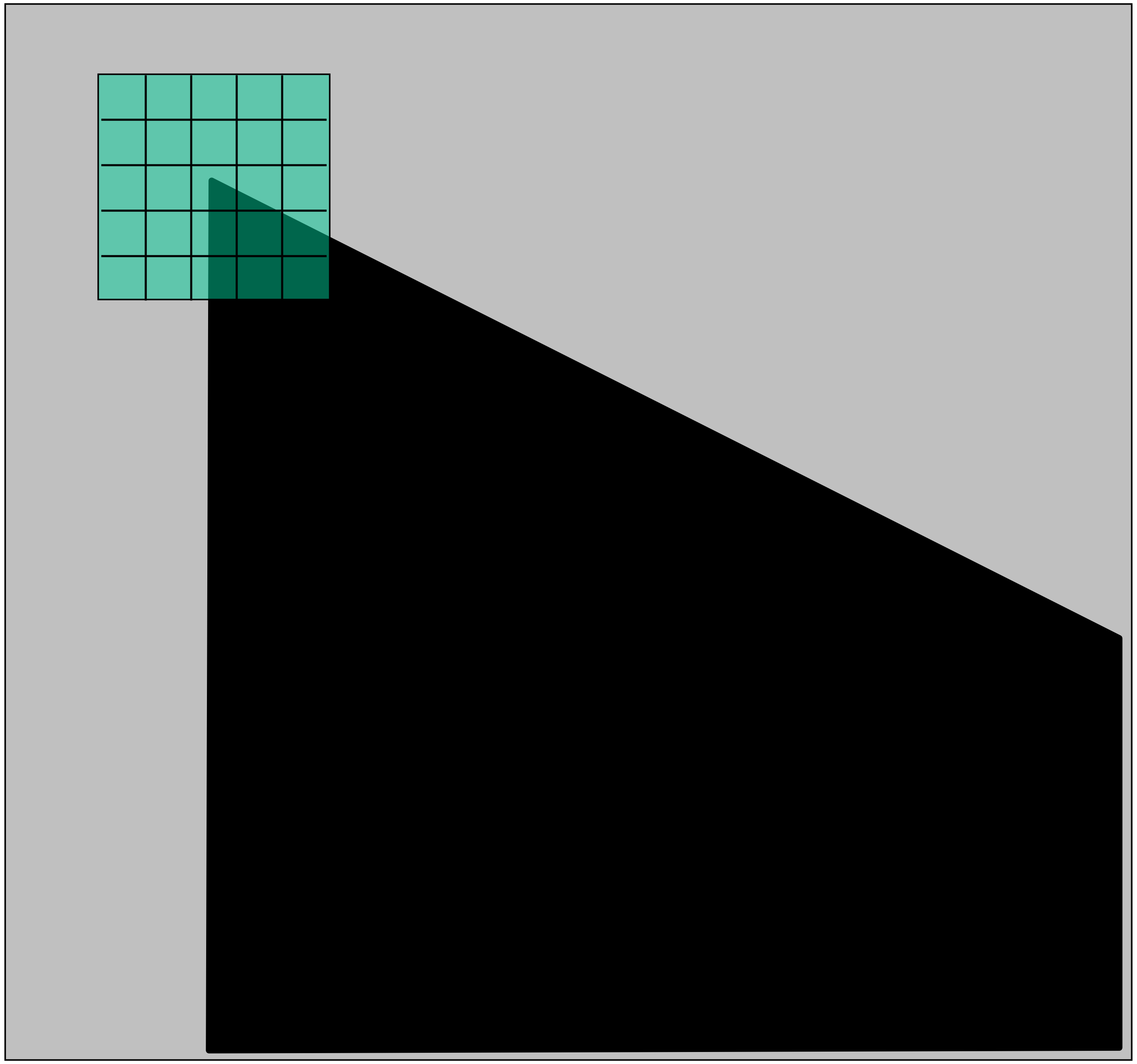
$$\begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

1. Compute image gradients over a small region

(not just a single pixel)

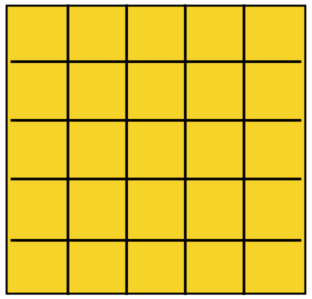
1. Compute image gradients over a small region

(not just a single pixel)



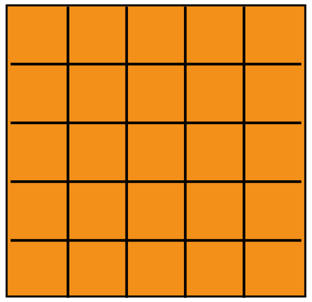
array of x gradients

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



array of y gradients

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

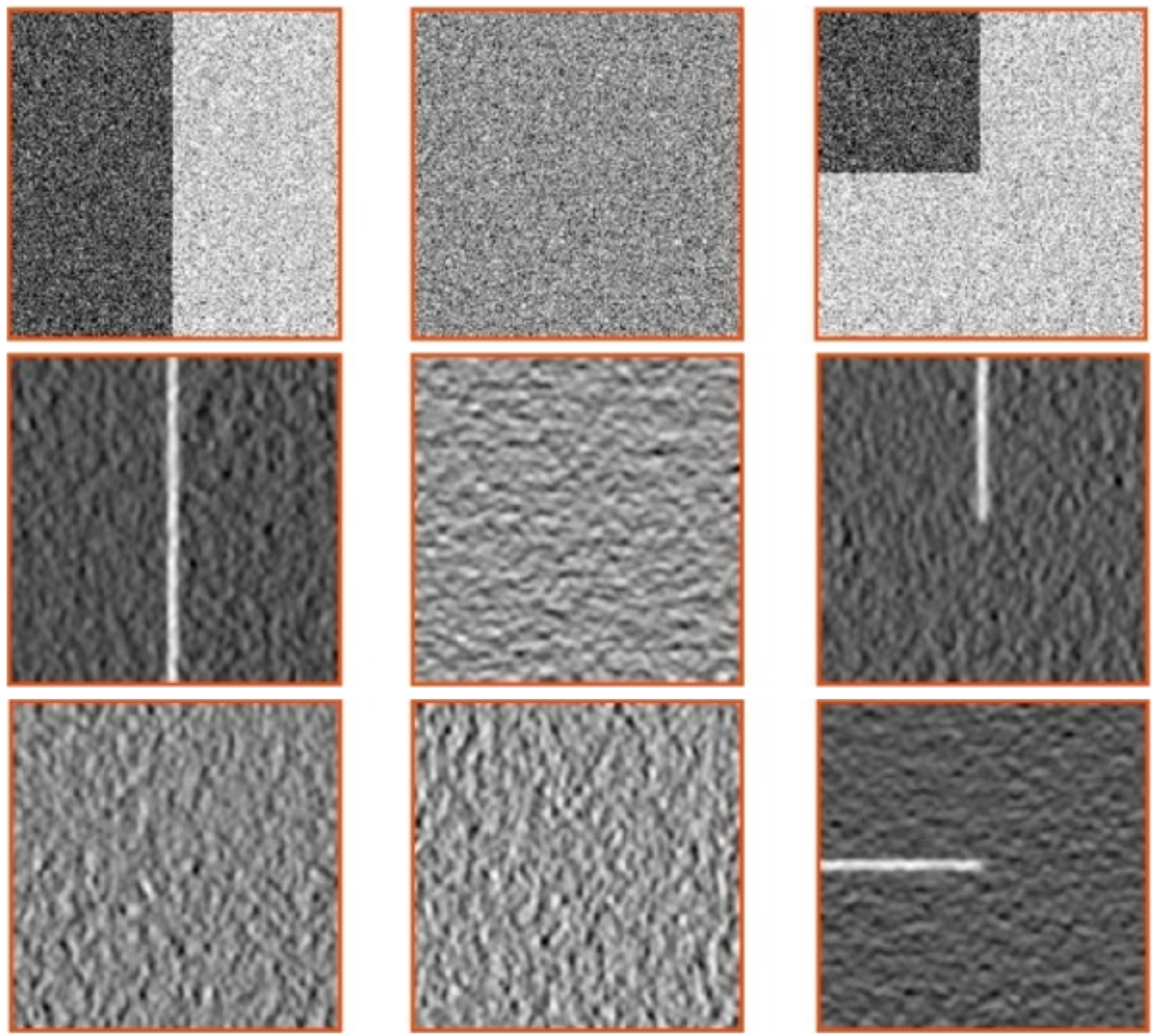


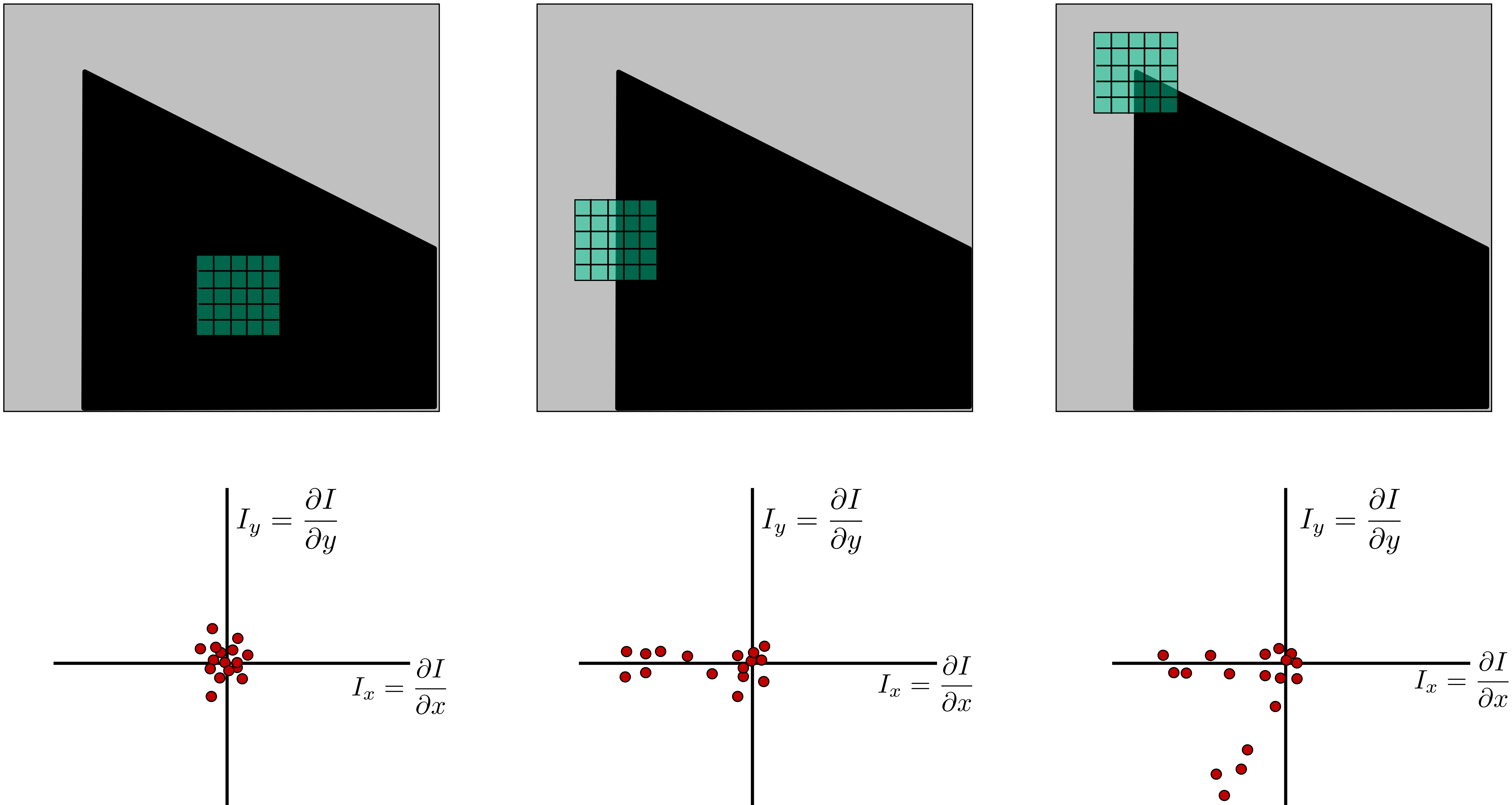
visualization of gradients

image

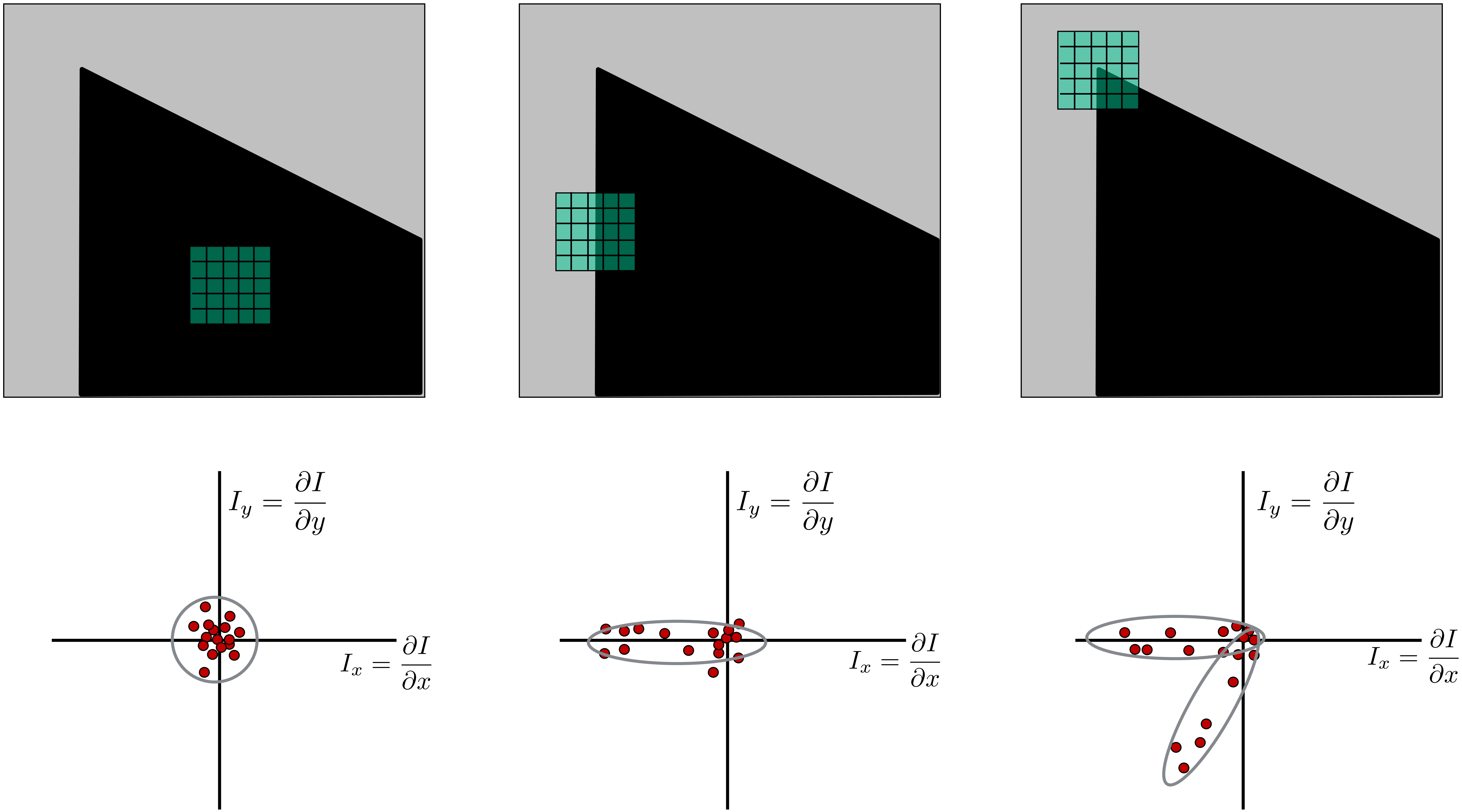
X derivative

Y derivative

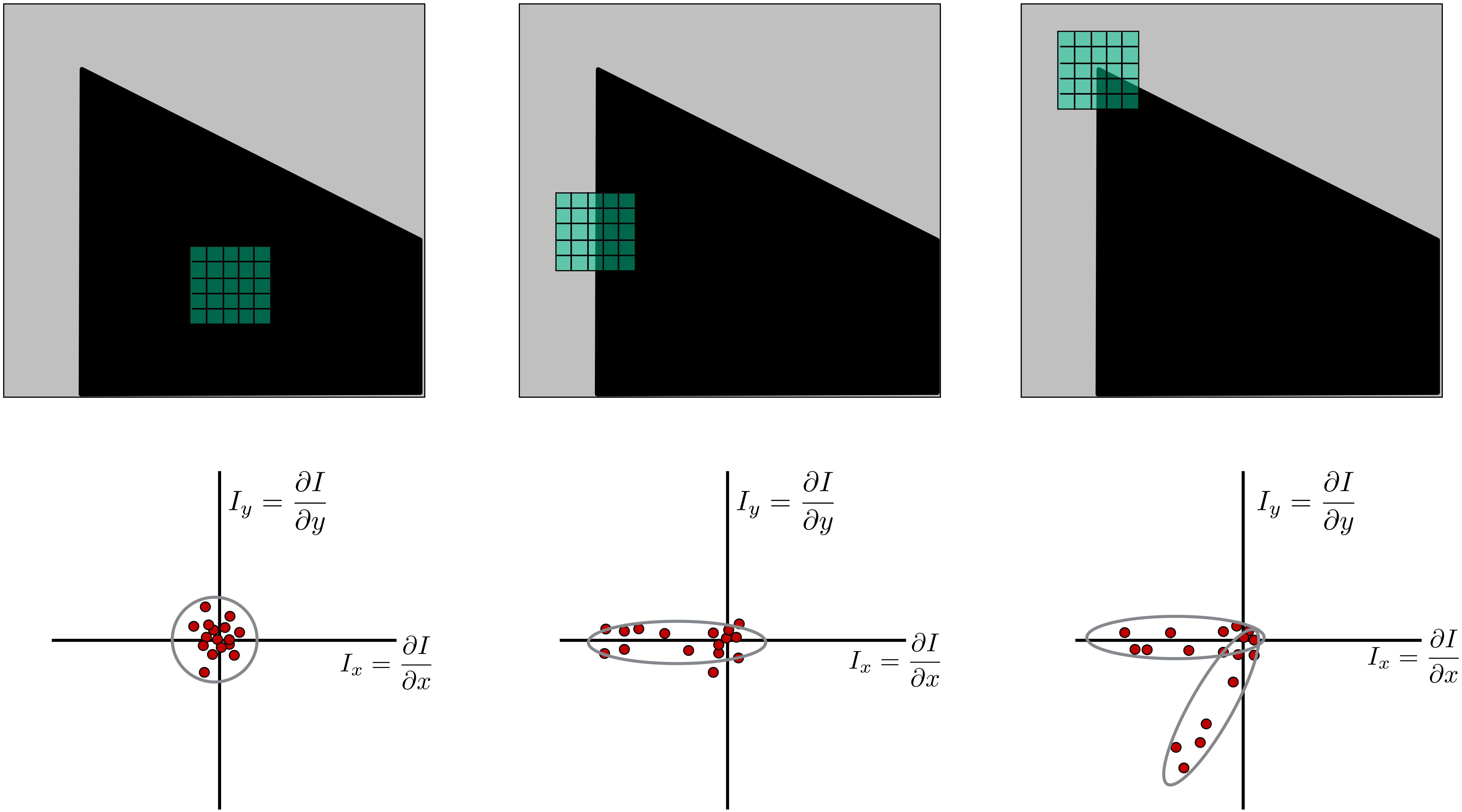




What does the distribution tell you about the region?



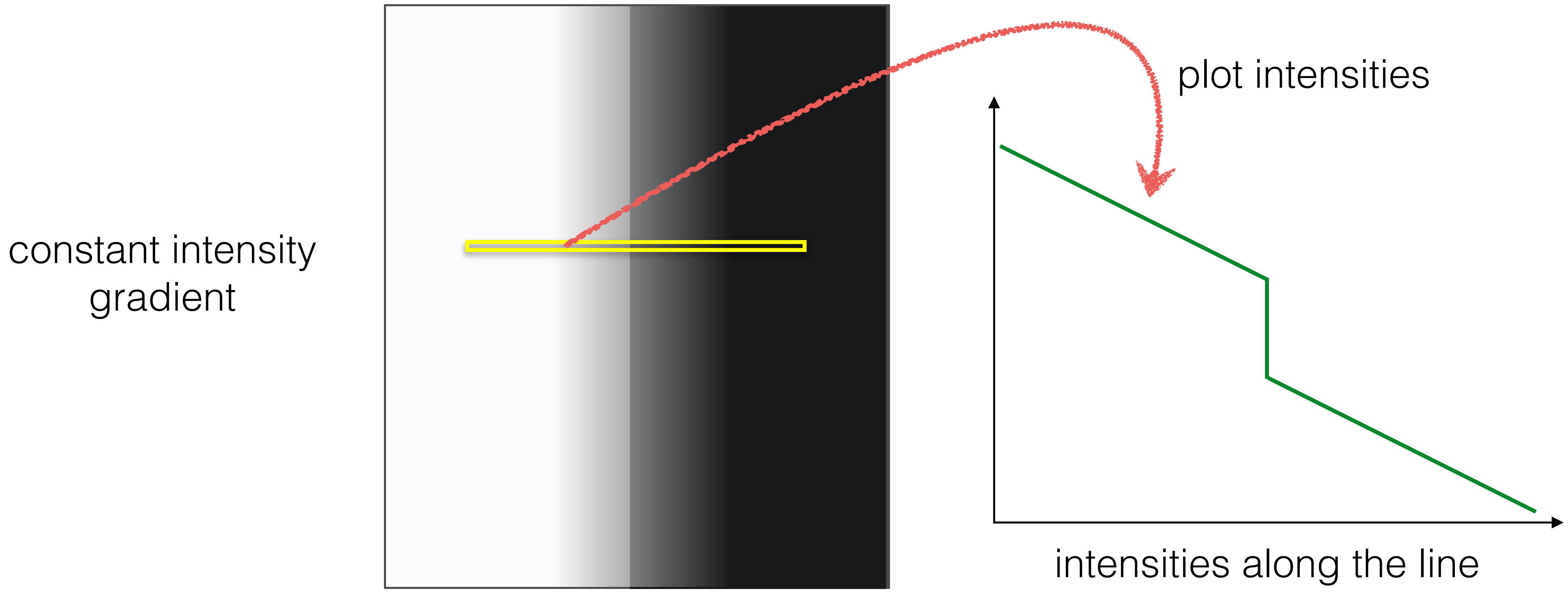
distribution reveals edge orientation and magnitude



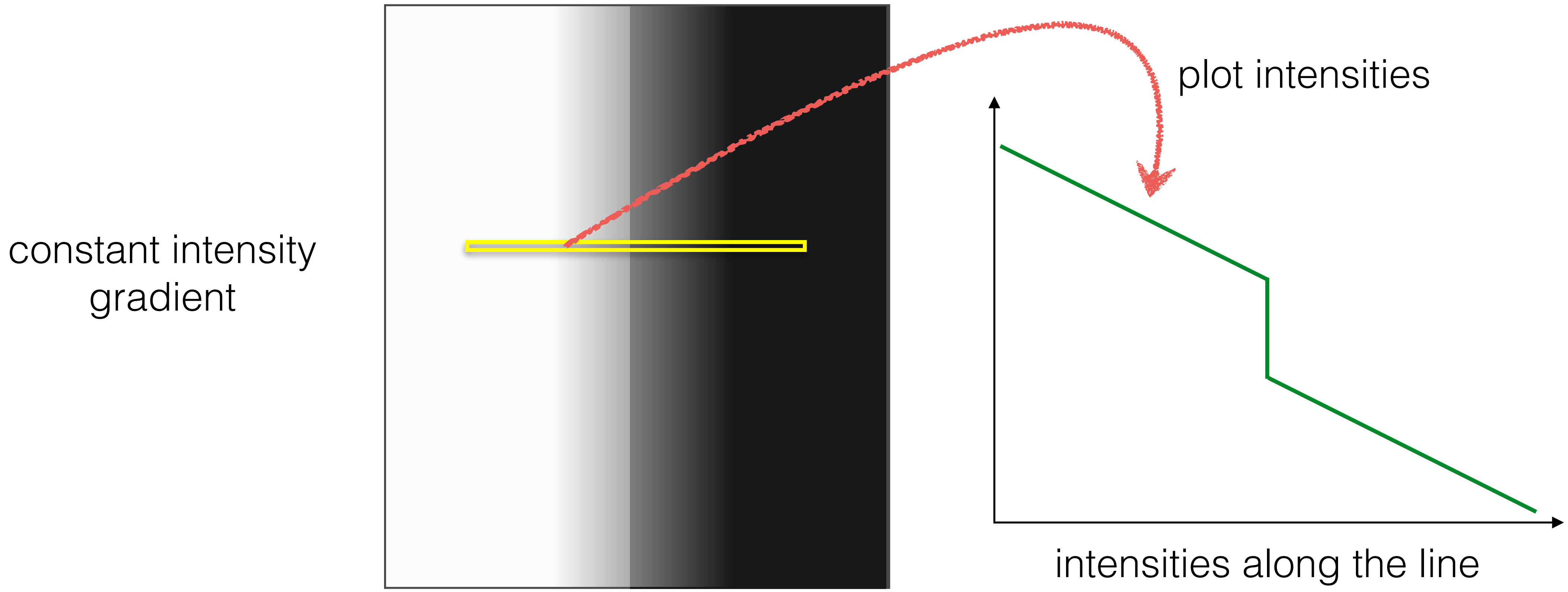
How do you quantify orientation and magnitude?

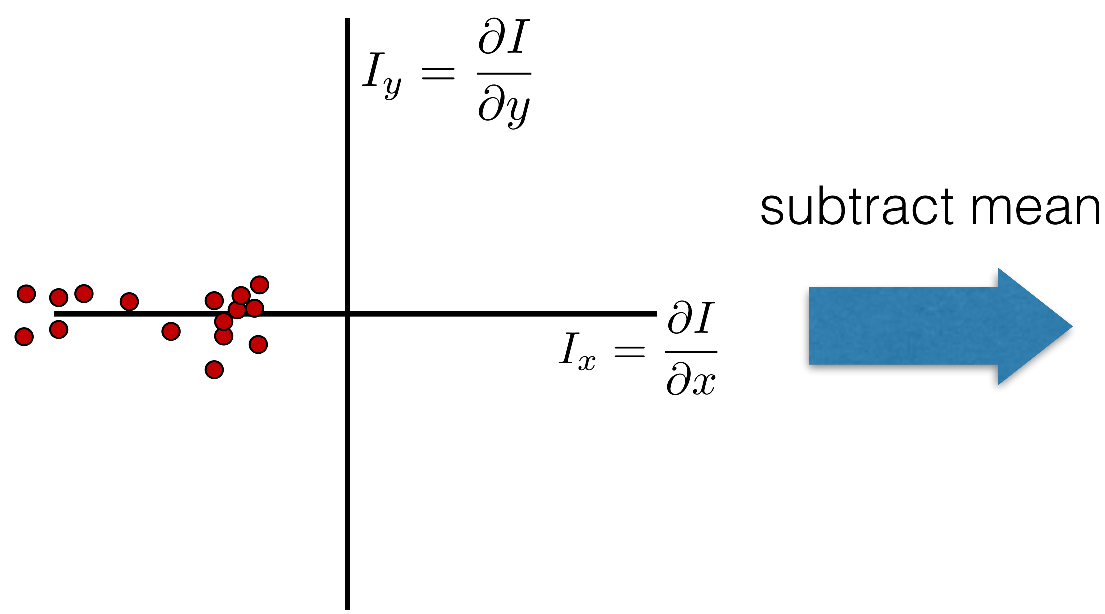
2. Subtract the mean from each image gradient

2. Subtract the mean from each image gradient



2. Subtract the mean from each image gradient

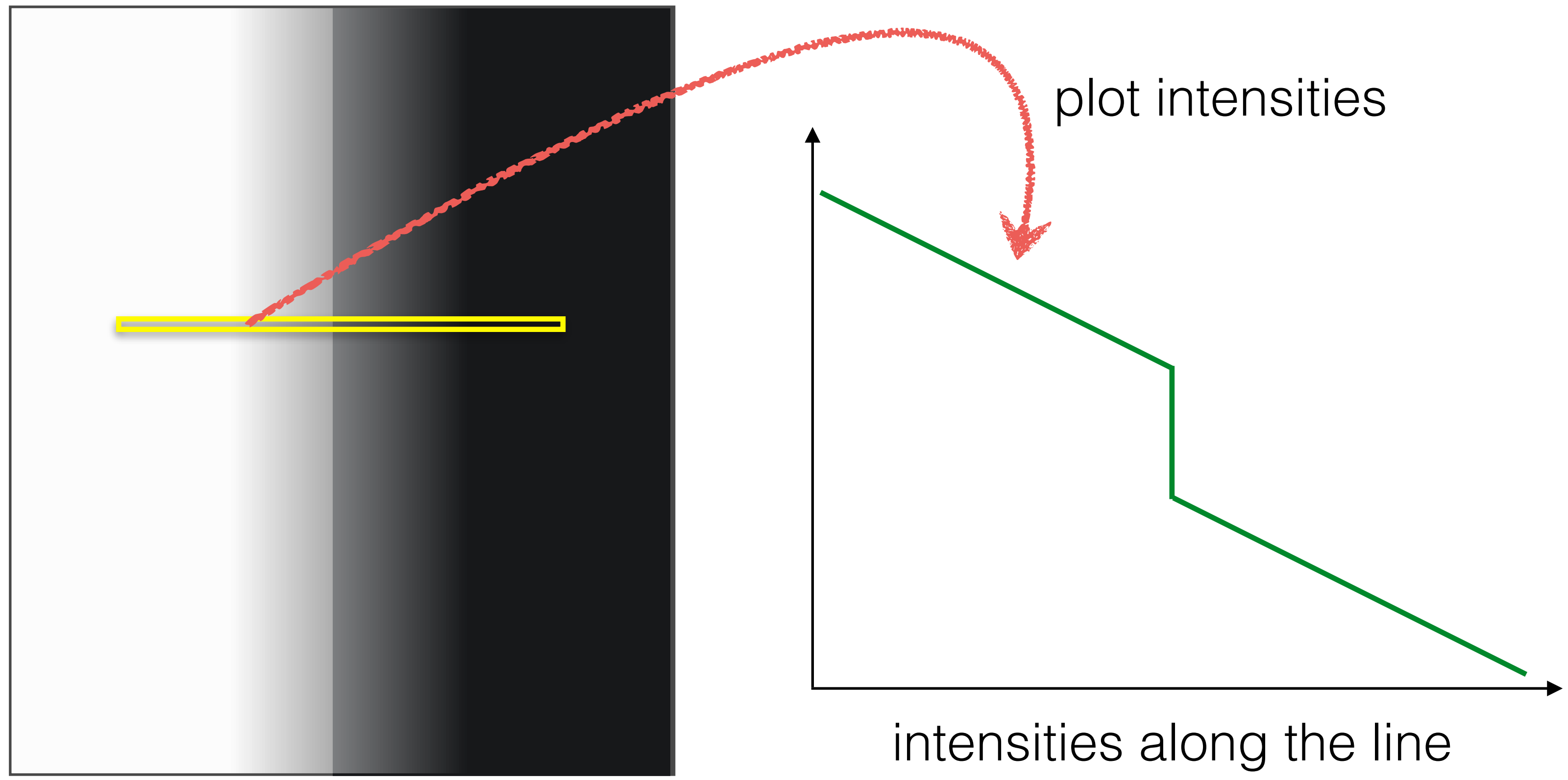


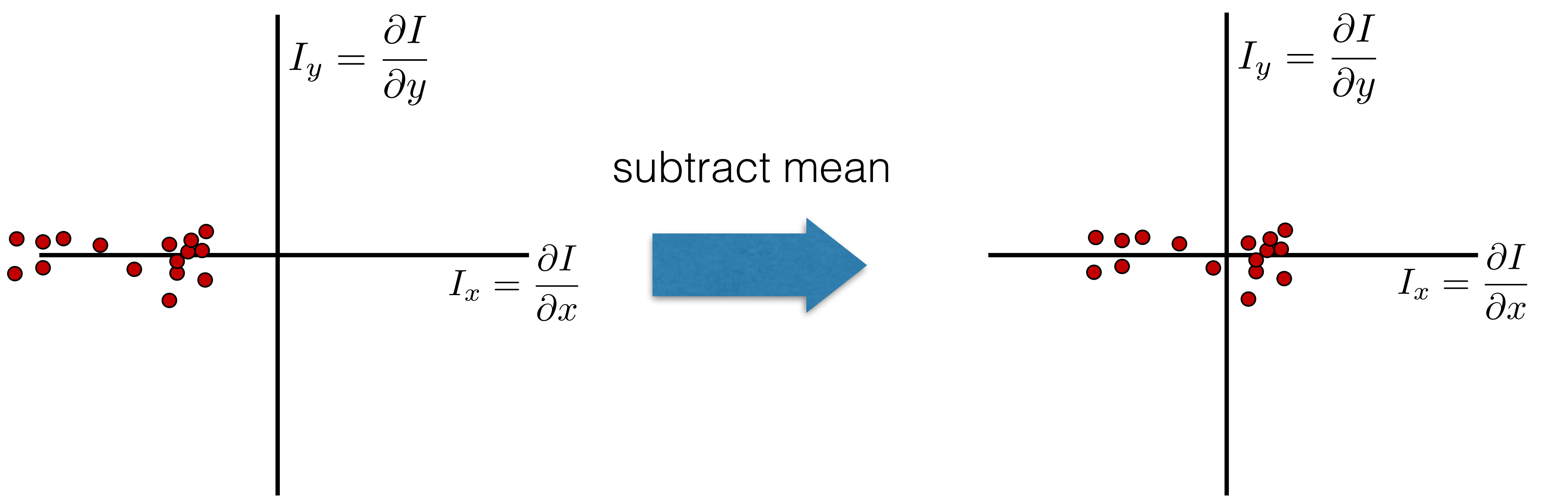


plot of image gradients

2. Subtract the mean from each image gradient

constant intensity gradient





plot of image gradients

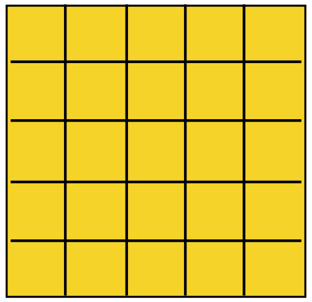
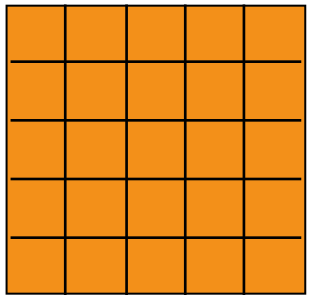
data is centered
('DC' offset is removed)

3. Compute the covariance matrix

3. Compute the covariance matrix

$$\begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

$$I_x = \frac{\partial I}{\partial x} \qquad I_y = \frac{\partial I}{\partial y}$$

$\sum_{p \in P} I_x I_y$ = sum( .* )

array of x gradients

array of y gradients

Where does this covariance matrix come from?

Error function

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

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

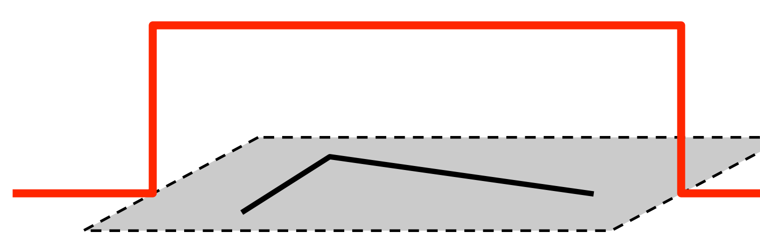
Error function

Window function

Shifted intensity

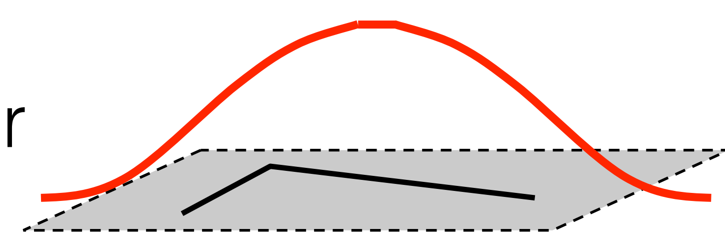
Intensity

Window function $w(x,y) =$



1 in window, 0 outside

or



Gaussian

Error function approximation

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

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

Second-order Taylor expansion of E(u,v) about (0,0)
(bilinear approximation for small shifts):

$$E(u,v) \approx E(0,0) + [u \quad v]\begin{bmatrix} E_u(0,0) \\ E_v(0,0) \end{bmatrix} + \frac{1}{2}[u \quad v]\begin{bmatrix} E_{uu}(0,0) & E_{uv}(0,0) \\ E_{uv}(0,0) & E_{vv}(0,0) \end{bmatrix}\begin{bmatrix} u \\ v \end{bmatrix}$$

first derivative

second derivative

Bilinear approximation

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

Change in appearance for a shift [u,v]

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

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

'second moment' matrix
'structure tensor'

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

By computing the gradient covariance matrix…

$$\begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

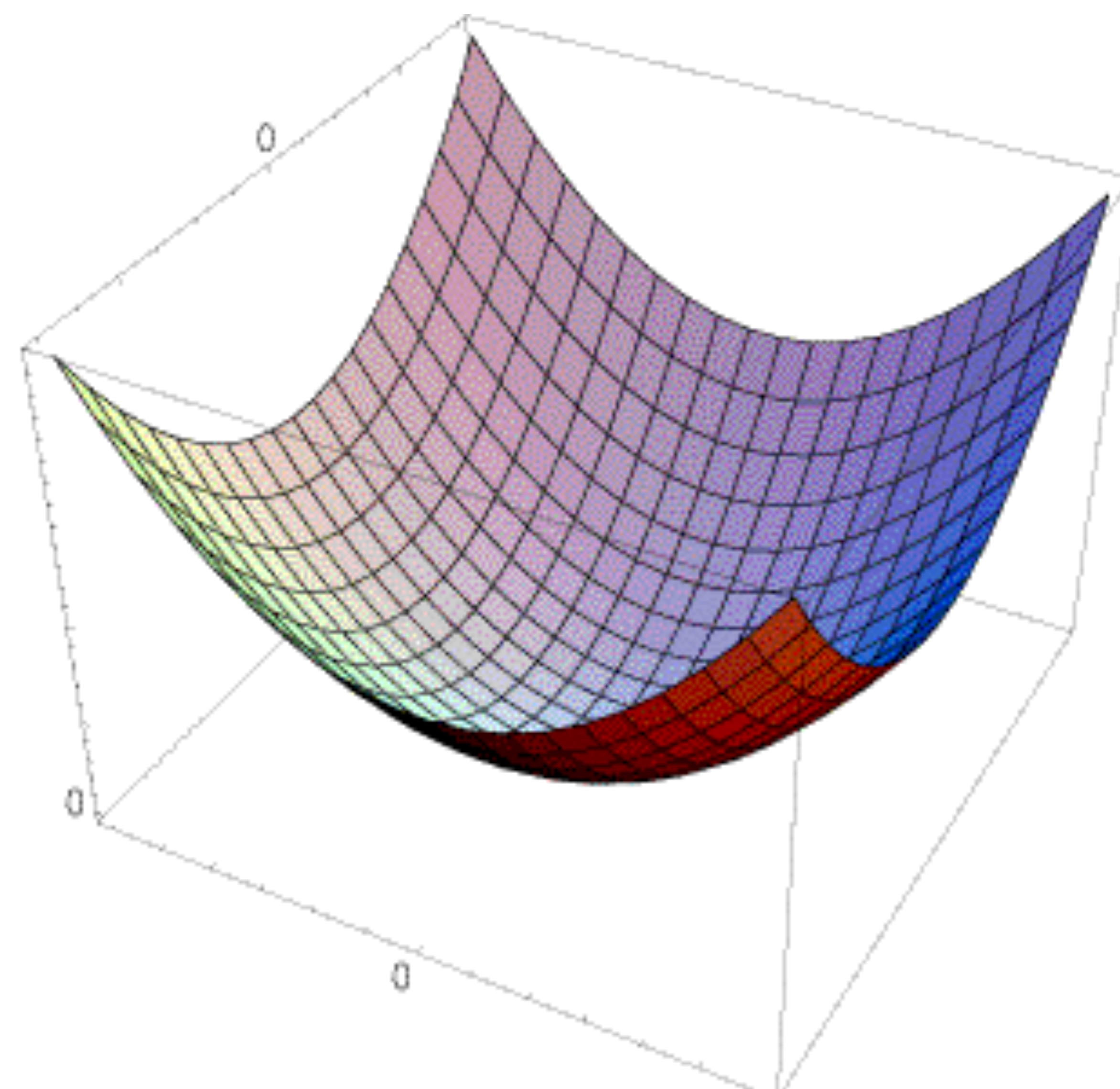
we are fitting a quadratic to the gradients over a small image region

Visualization of a quadratic

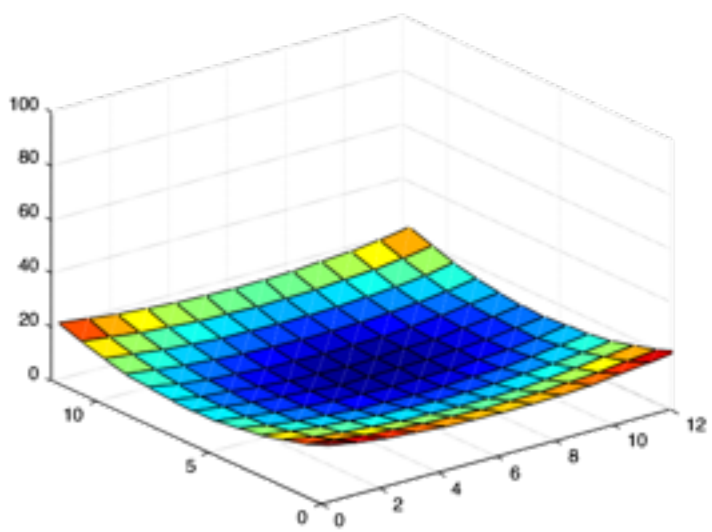
The surface *E*(*u*,*v*) is locally approximated by a quadratic form

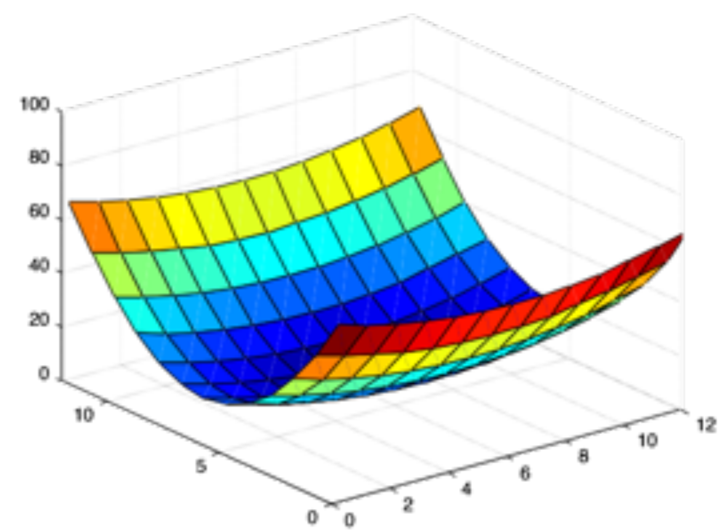
$$E(u,v) \approx [u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

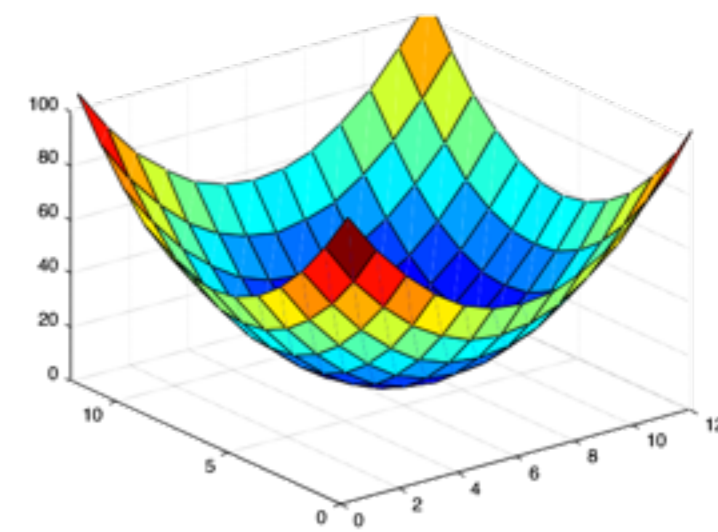
$$M = \sum \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$



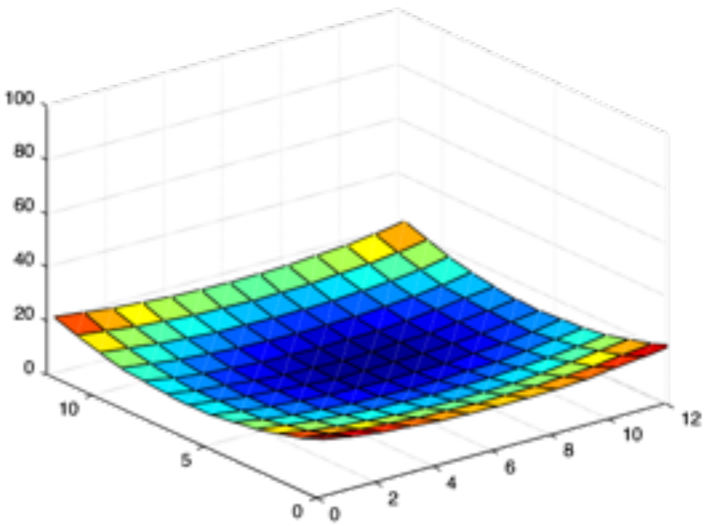
Which error surface indicates a good image feature?



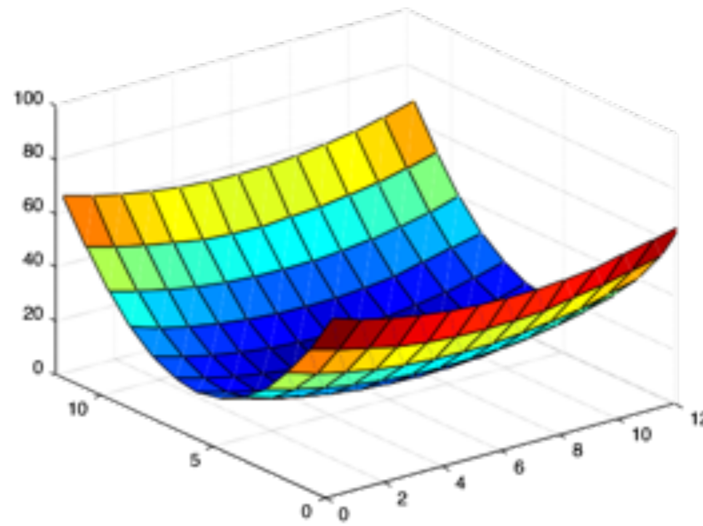




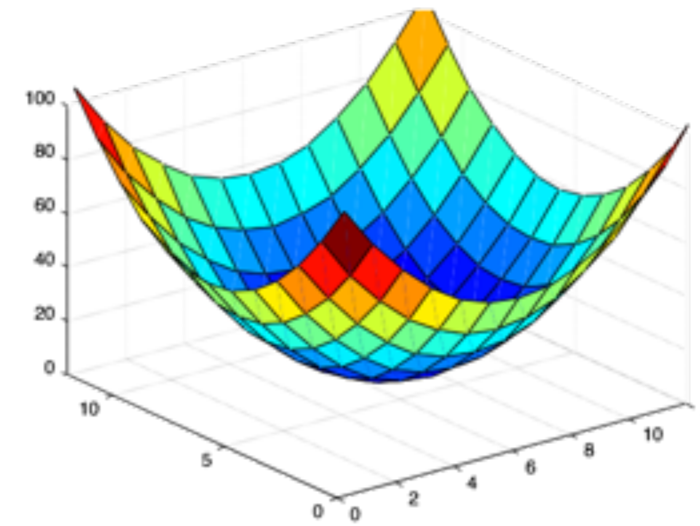
What kind of image patch do these surfaces represent?



flat



edge



corner

4. Compute eigenvalues and eigenvectors

```
eig(M)
```

4. Compute eigenvalues and eigenvectors

eigenvalue

$$\boldsymbol{M}\boldsymbol{e} = \lambda \boldsymbol{e}$$

eigenvector

$$(M - \lambda I)\boldsymbol{e} = 0$$

4. Compute eigenvalues and eigenvectors

eigenvalue

$$M\boldsymbol{e} = \lambda \boldsymbol{e}$$

eigenvector

$$(M - \lambda I)\boldsymbol{e} = 0$$

1. Compute the determinant of $M - \lambda I$

(returns a polynomial)

4. Compute eigenvalues and eigenvectors

eigenvalue

$$M\boldsymbol{e} = \lambda \boldsymbol{e}$$

eigenvector

$$(M - \lambda I)\boldsymbol{e} = 0$$

1. Compute the determinant of $\quad M - \lambda I$

(returns a polynomial)

2. Find the roots of polynomial $\quad \det(M - \lambda I) = 0$

(returns eigenvalues)

4. Compute eigenvalues and eigenvectors

eigenvalue

$$M\boldsymbol{e} = \lambda\boldsymbol{e}$$

eigenvector

$$(M - \lambda I)\boldsymbol{e} = 0$$

1. Compute the determinant of $M - \lambda I$
(returns a polynomial)

2. Find the roots of polynomial $\det(M - \lambda I) = 0$
(returns eigenvalues)

3. For each eigenvalue, solve $(M - \lambda I)\boldsymbol{e} = 0$
(returns eigenvectors)

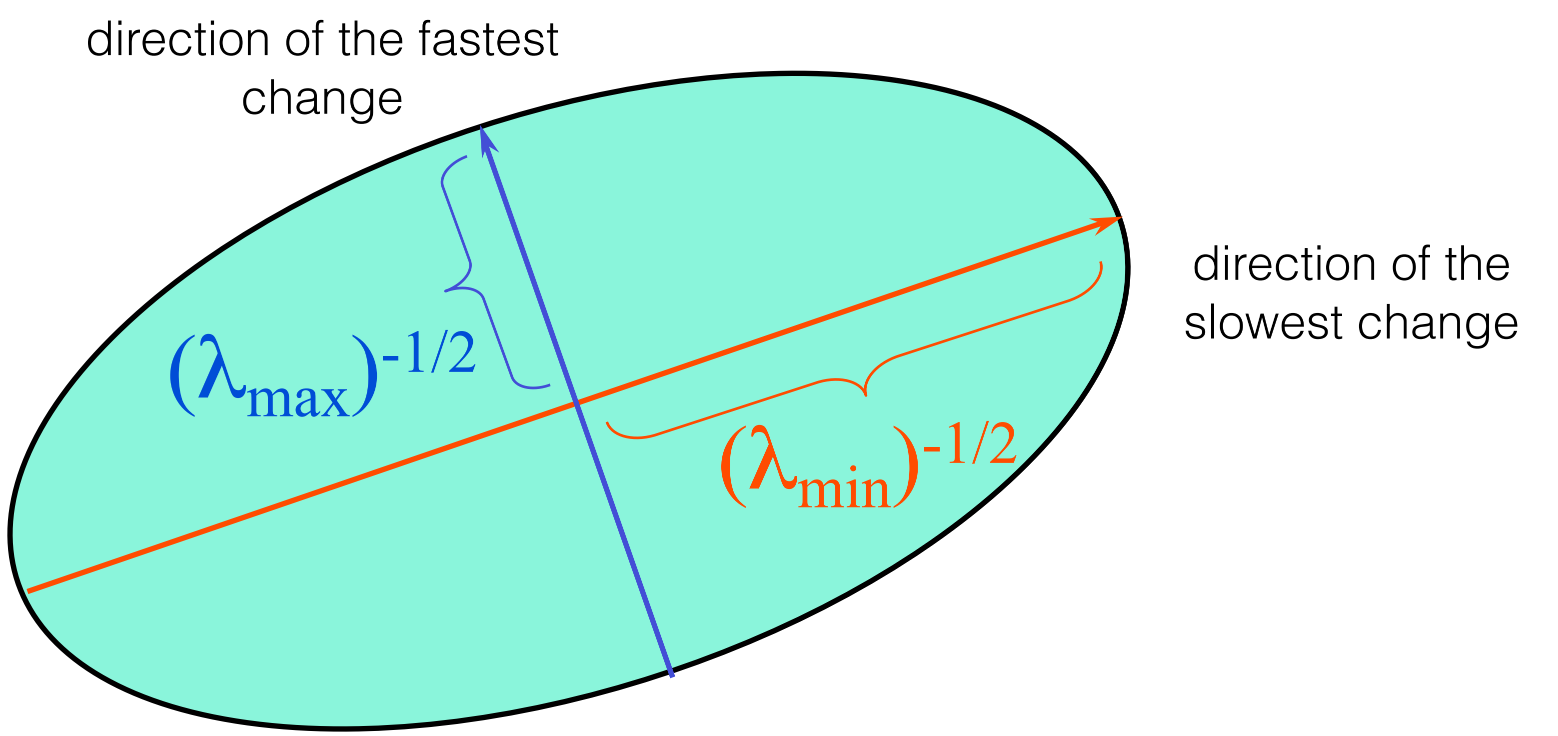
Visualization as an ellipse

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

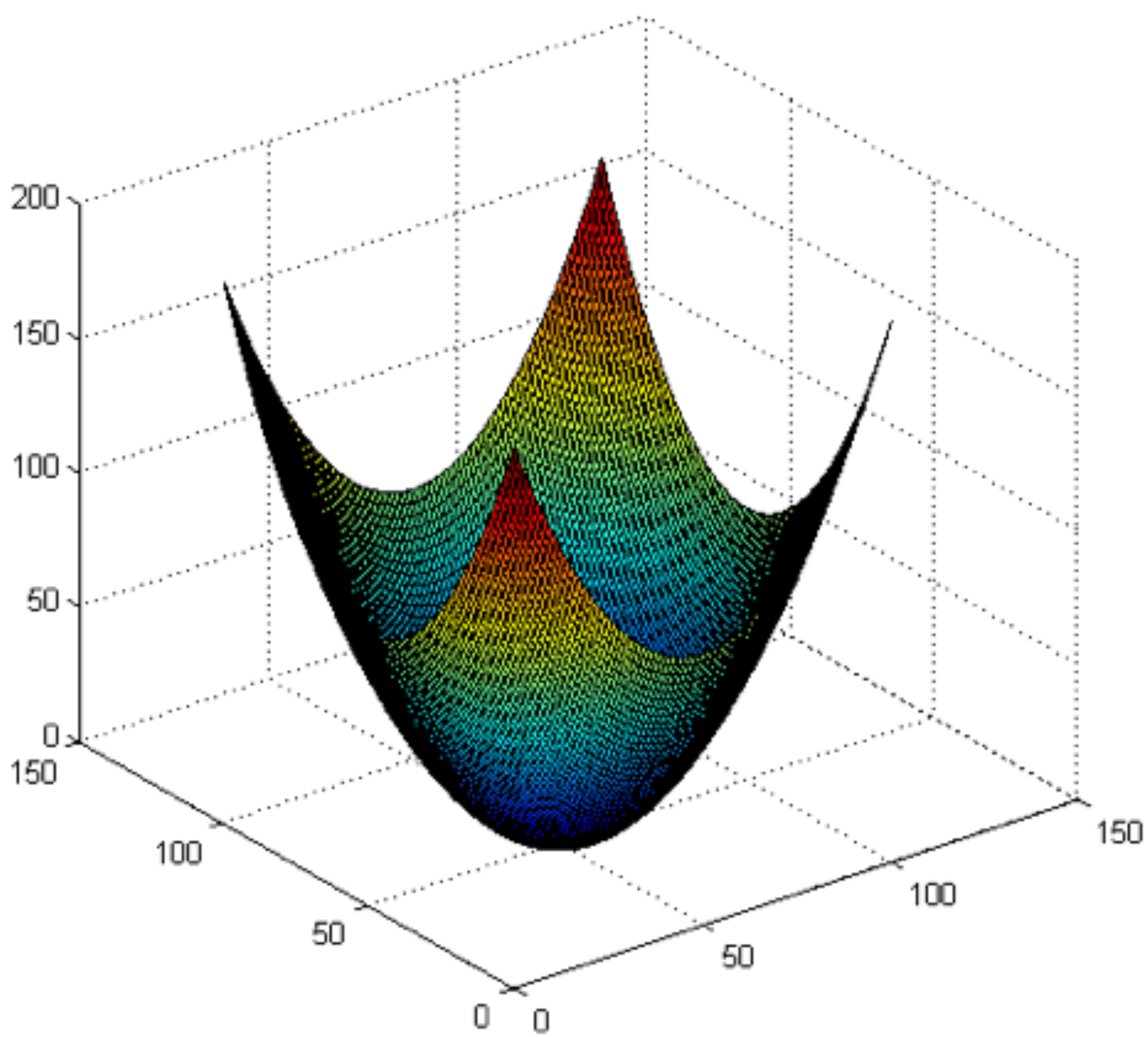
We can visualize ***M*** as an ellipse with axis lengths determined by the eigenvalues and orientation determined by ***R***

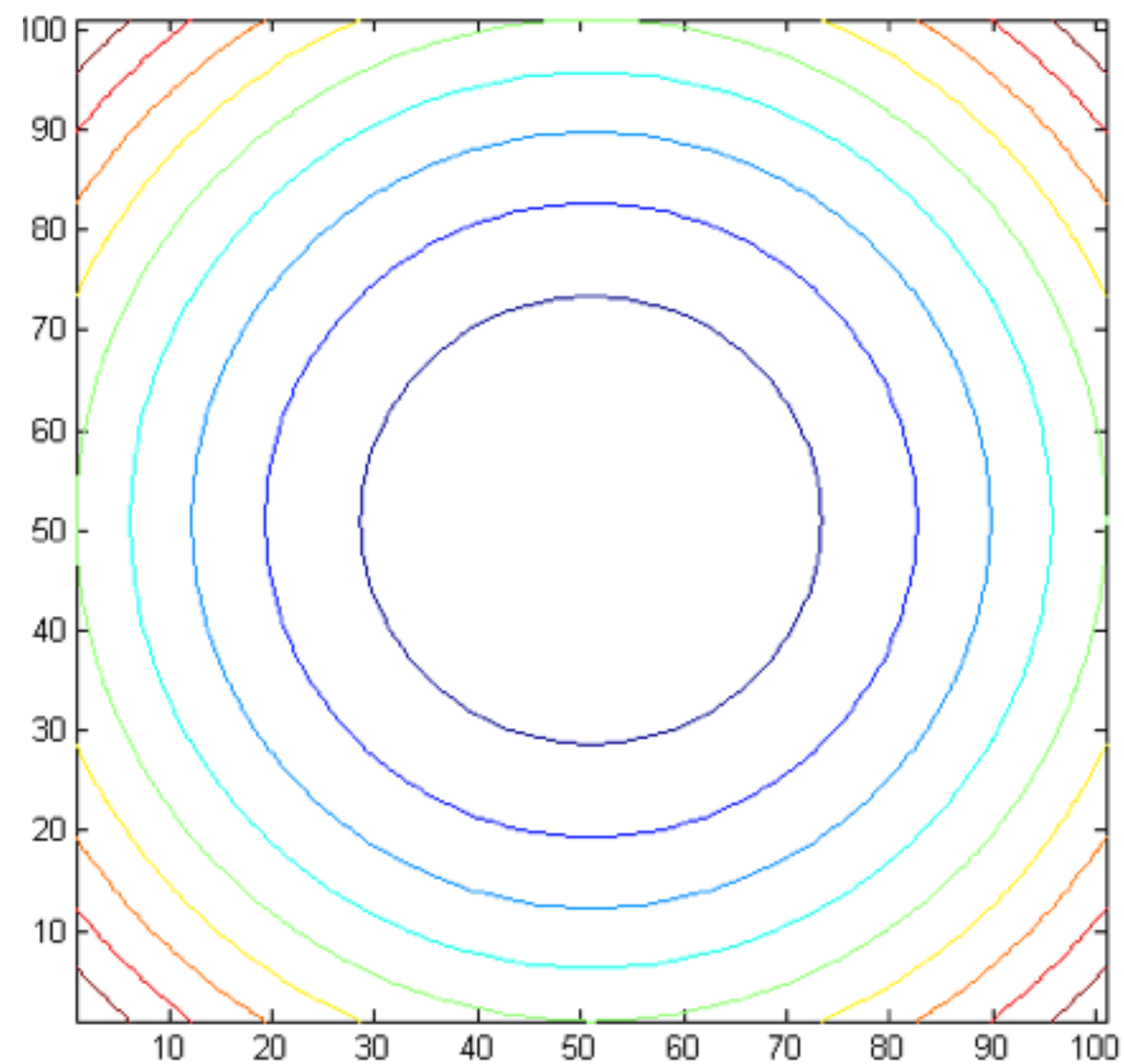
Ellipse equation:

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

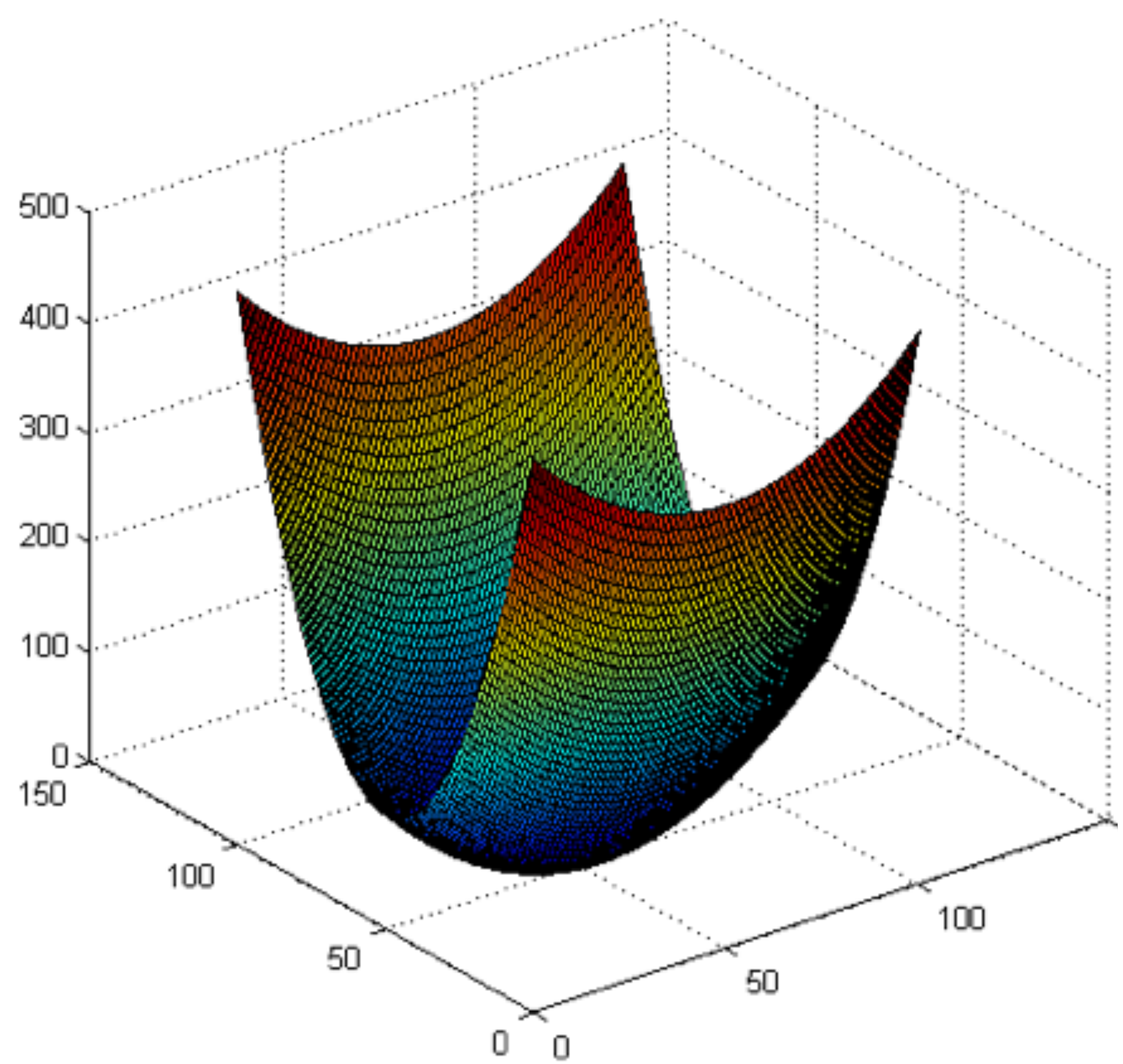


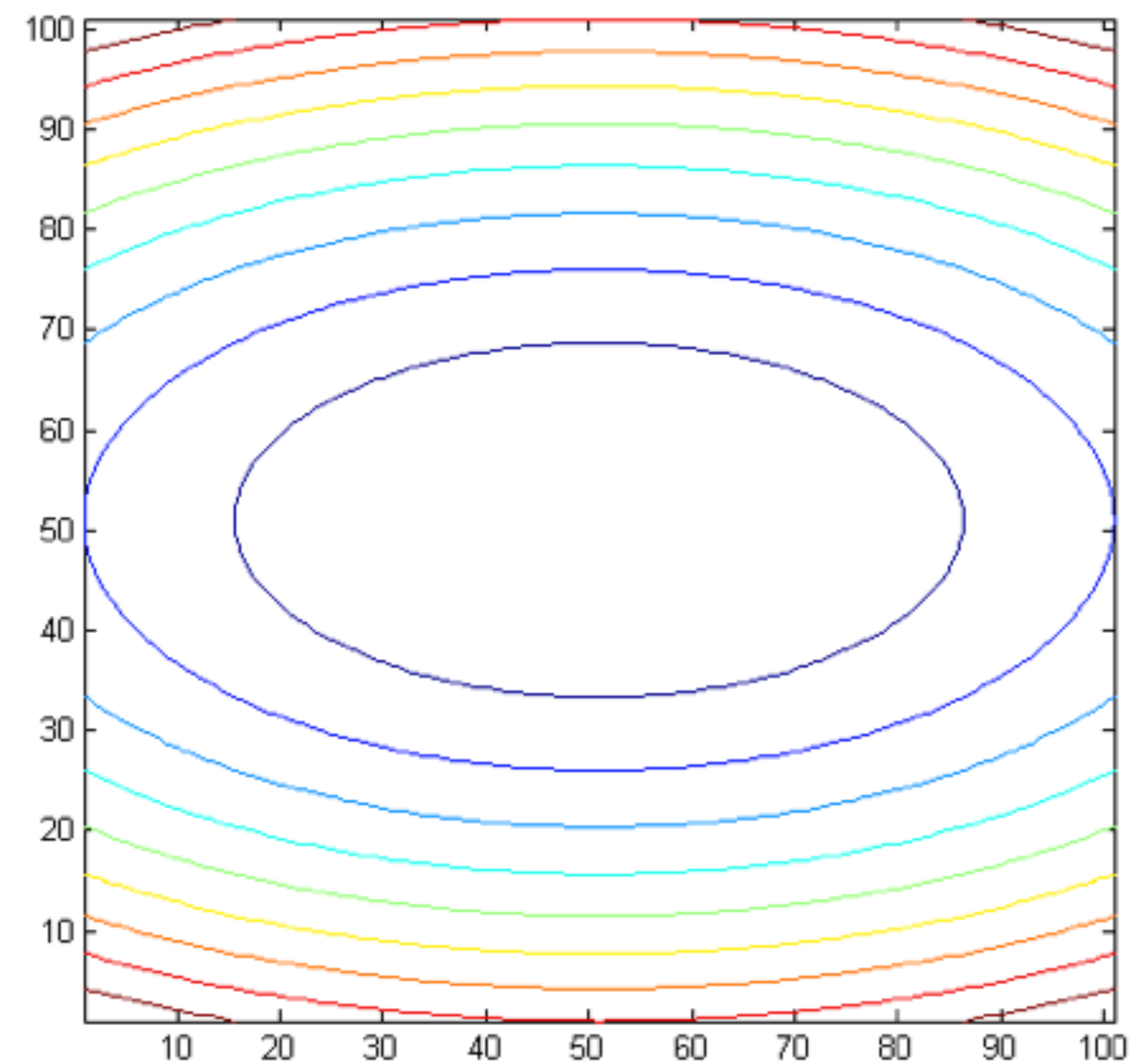
$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\text{Eigenvectors}} \overbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}^{\text{Eigenvalues}} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T}_{\text{Eigenvectors}}$$



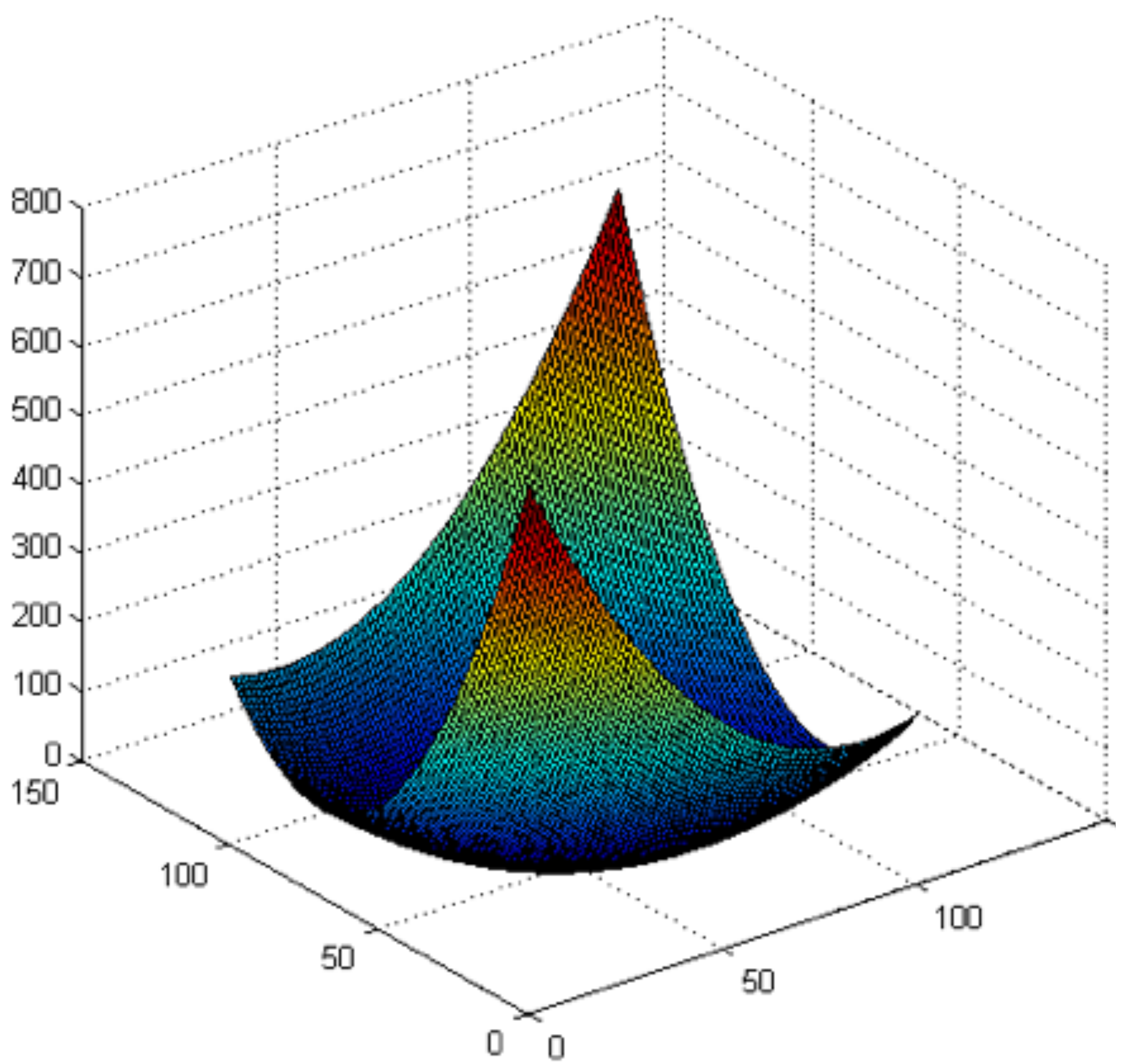


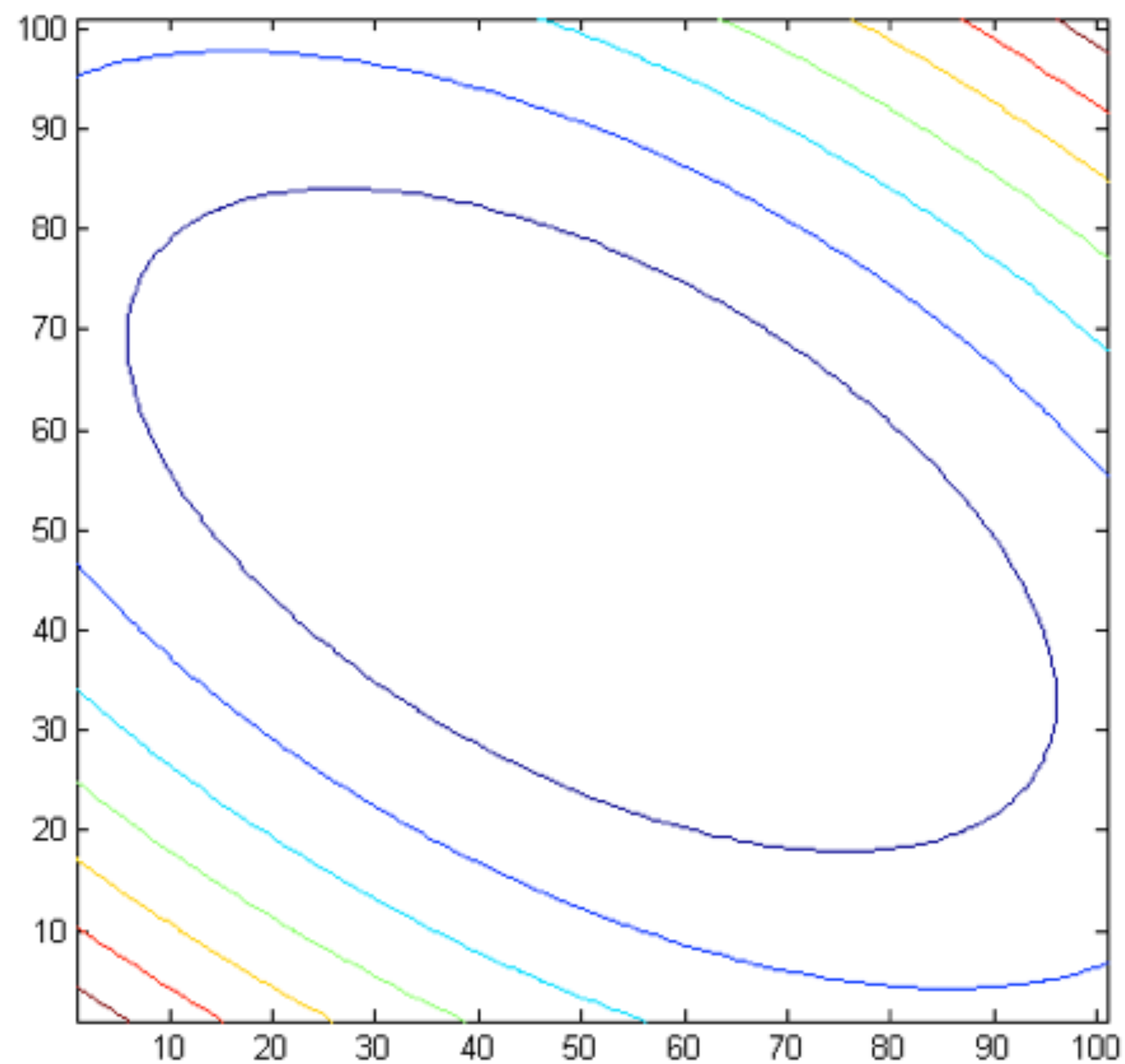
$$\mathbf{A} = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\text{Eigenvectors}} \overbrace{\begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix}}^{\text{Eigenvalues}} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T}_{\text{Eigenvectors}}$$





$$\mathbf{A} = \begin{bmatrix} 3.25 & 1.30 \\ 1.30 & 1.75 \end{bmatrix} = \underbrace{\begin{bmatrix} 0.50 & -0.87 \\ -0.87 & -0.50 \end{bmatrix}}_{\text{Eigenvectors}} \overbrace{\begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix}}^{\text{Eigenvalues}} \underbrace{\begin{bmatrix} 0.50 & -0.87 \\ -0.87 & -0.50 \end{bmatrix}^T}_{\text{Eigenvectors}}$$

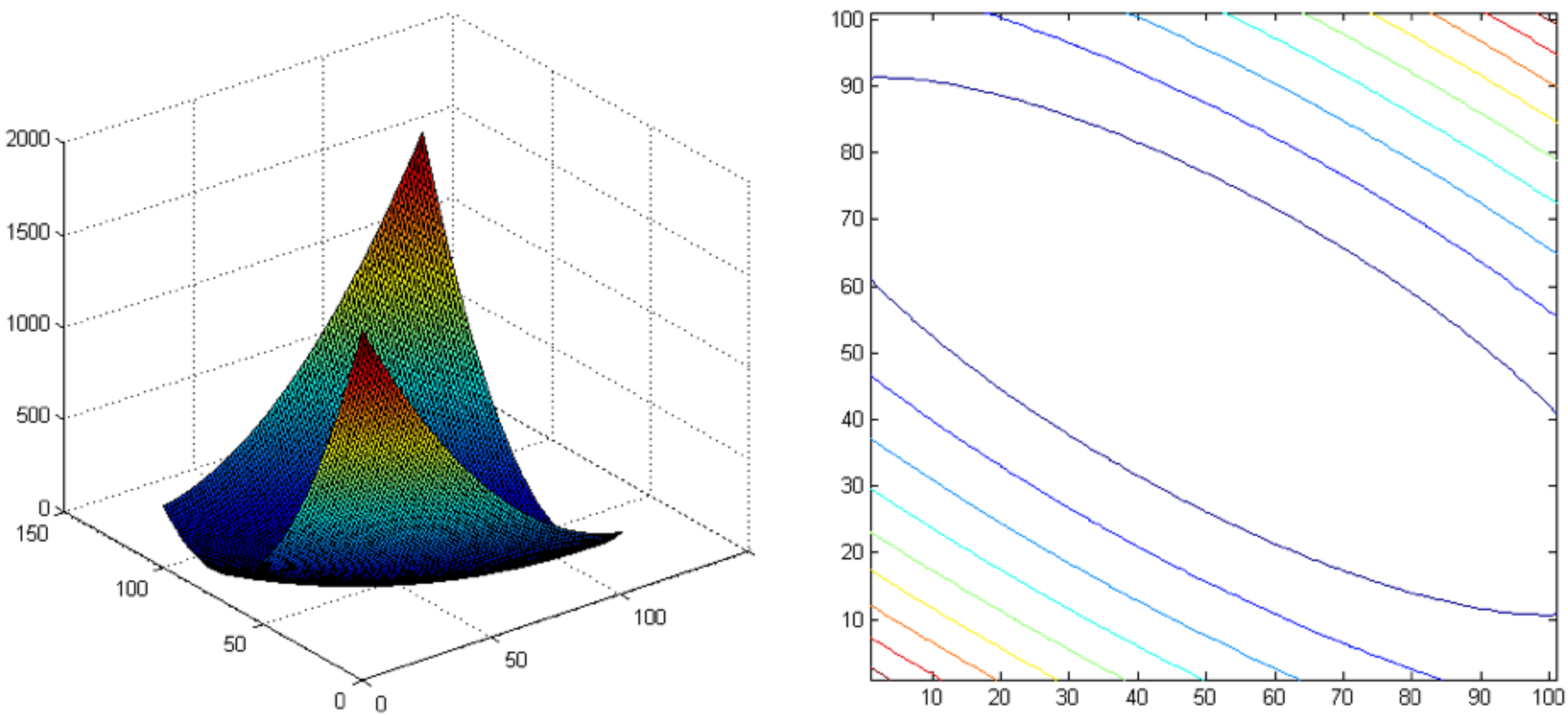




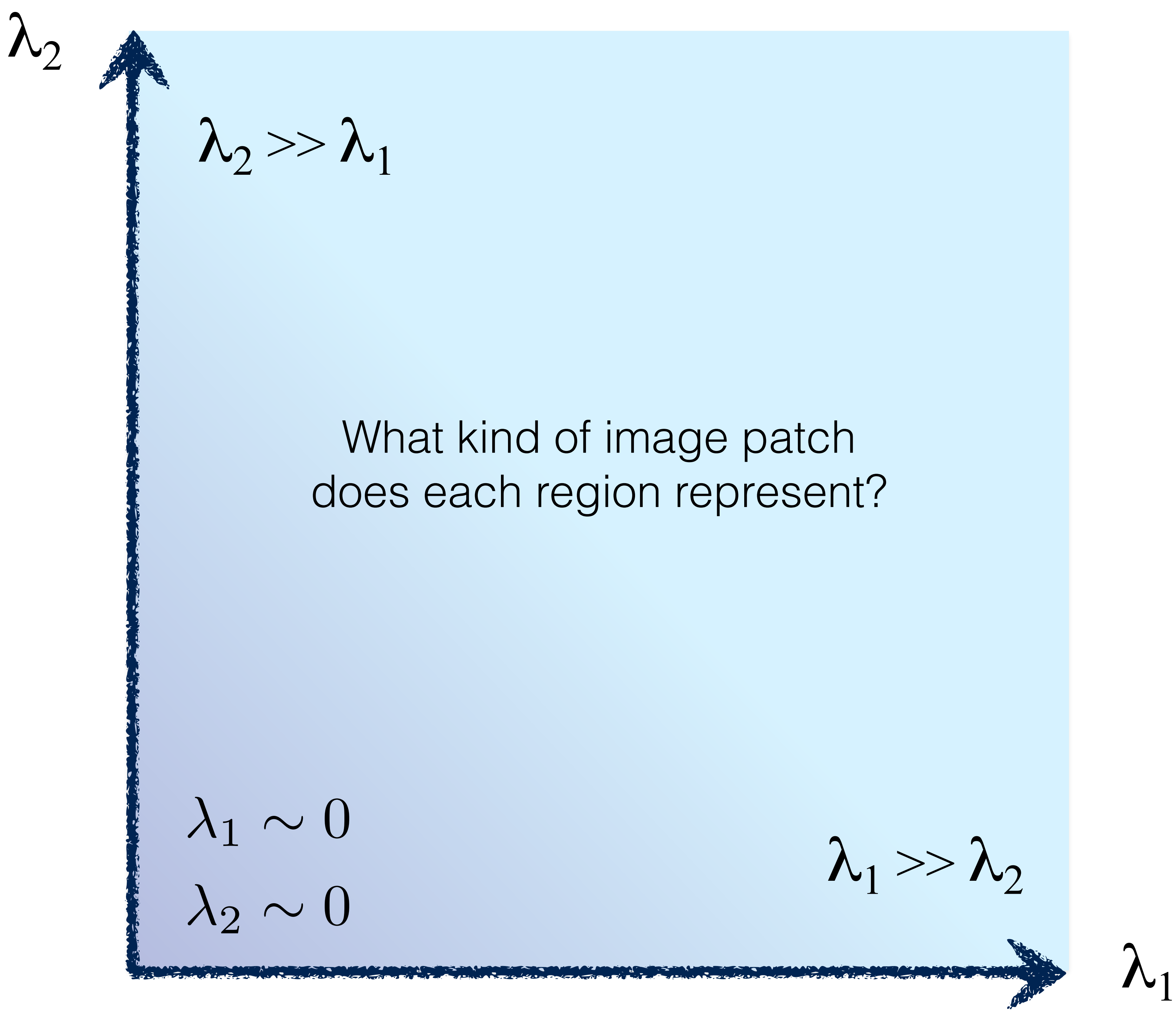
Eigenvalues

$$\mathbf{A} = \begin{bmatrix} 7.75 & 3.90 \\ 3.90 & 3.25 \end{bmatrix} = \begin{bmatrix} 0.50 & -0.87 \\ -0.87 & -0.50 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 10 \end{bmatrix} \begin{bmatrix} 0.50 & -0.87 \\ -0.87 & -0.50 \end{bmatrix}^T$$

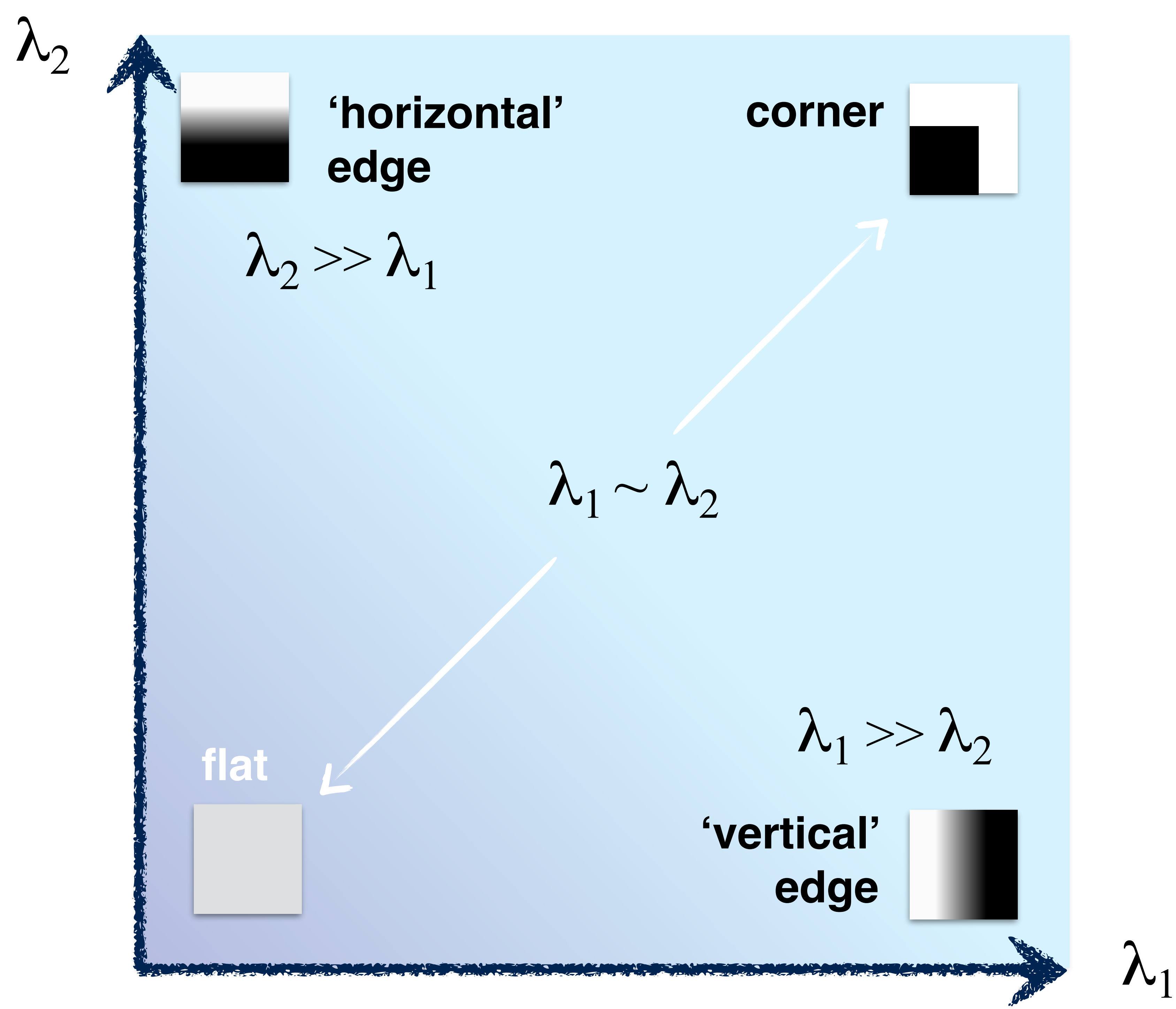
Eigenvectors Eigenvectors



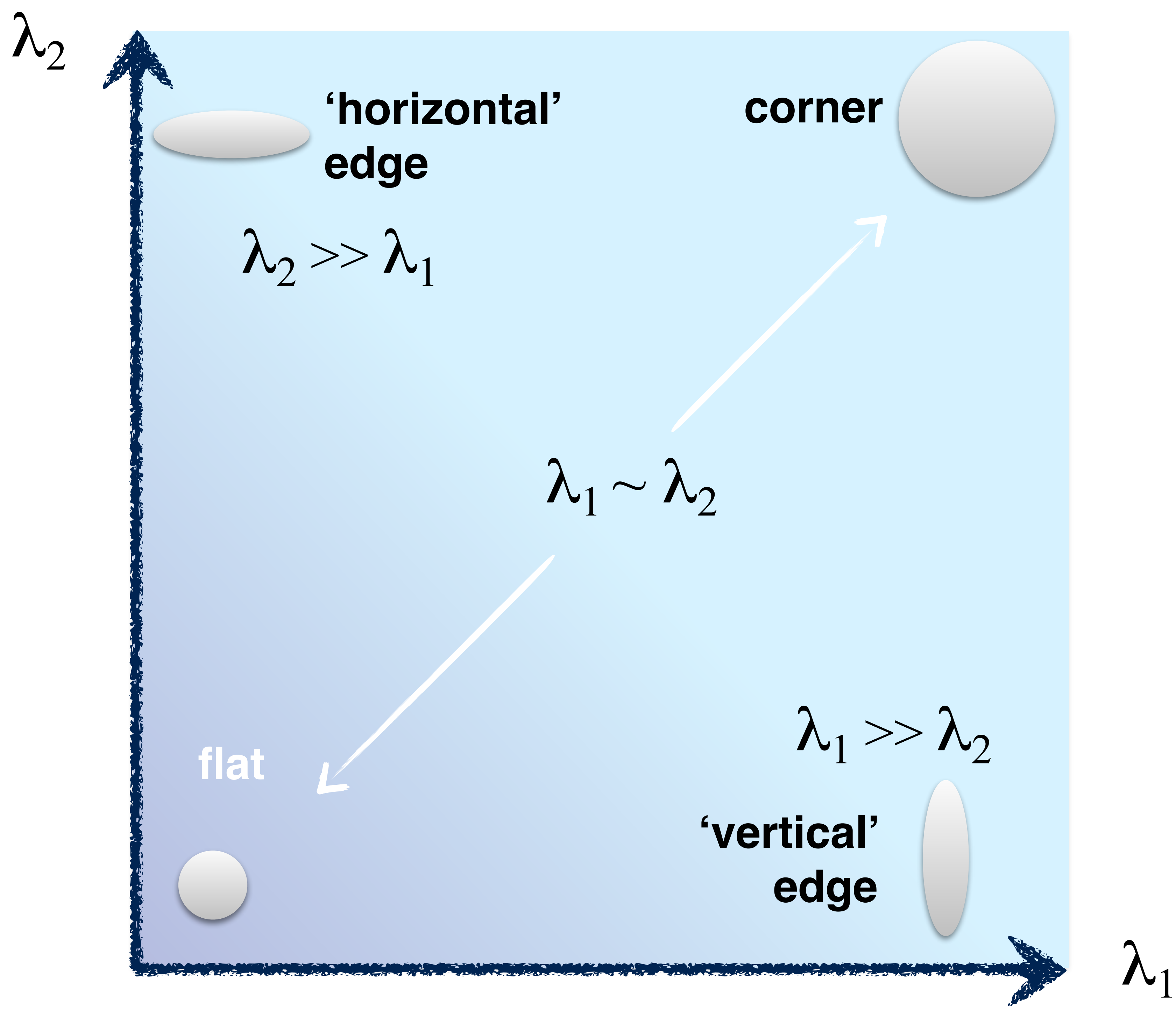
interpreting eigenvalues



interpreting eigenvalues

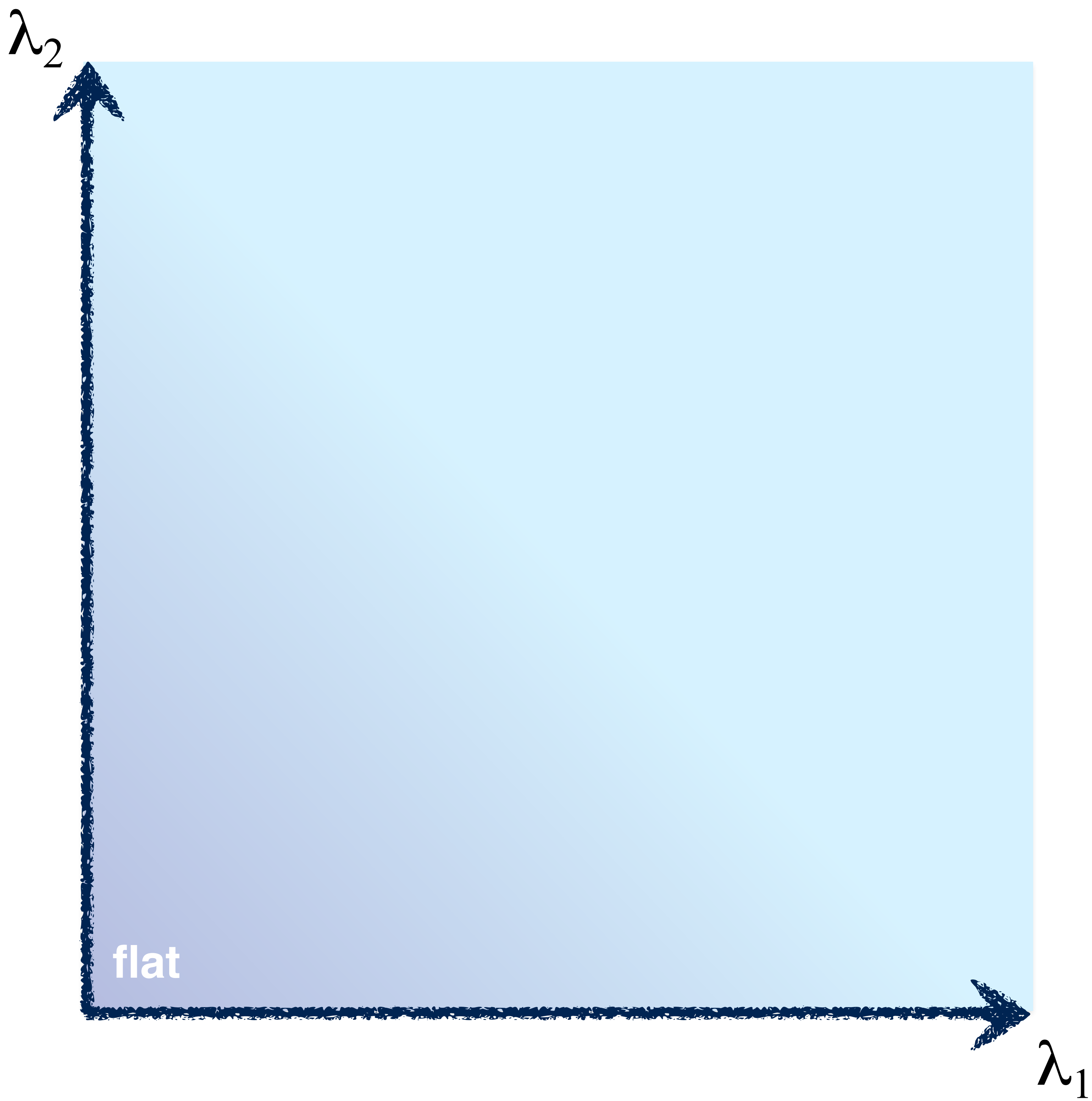


interpreting eigenvalues



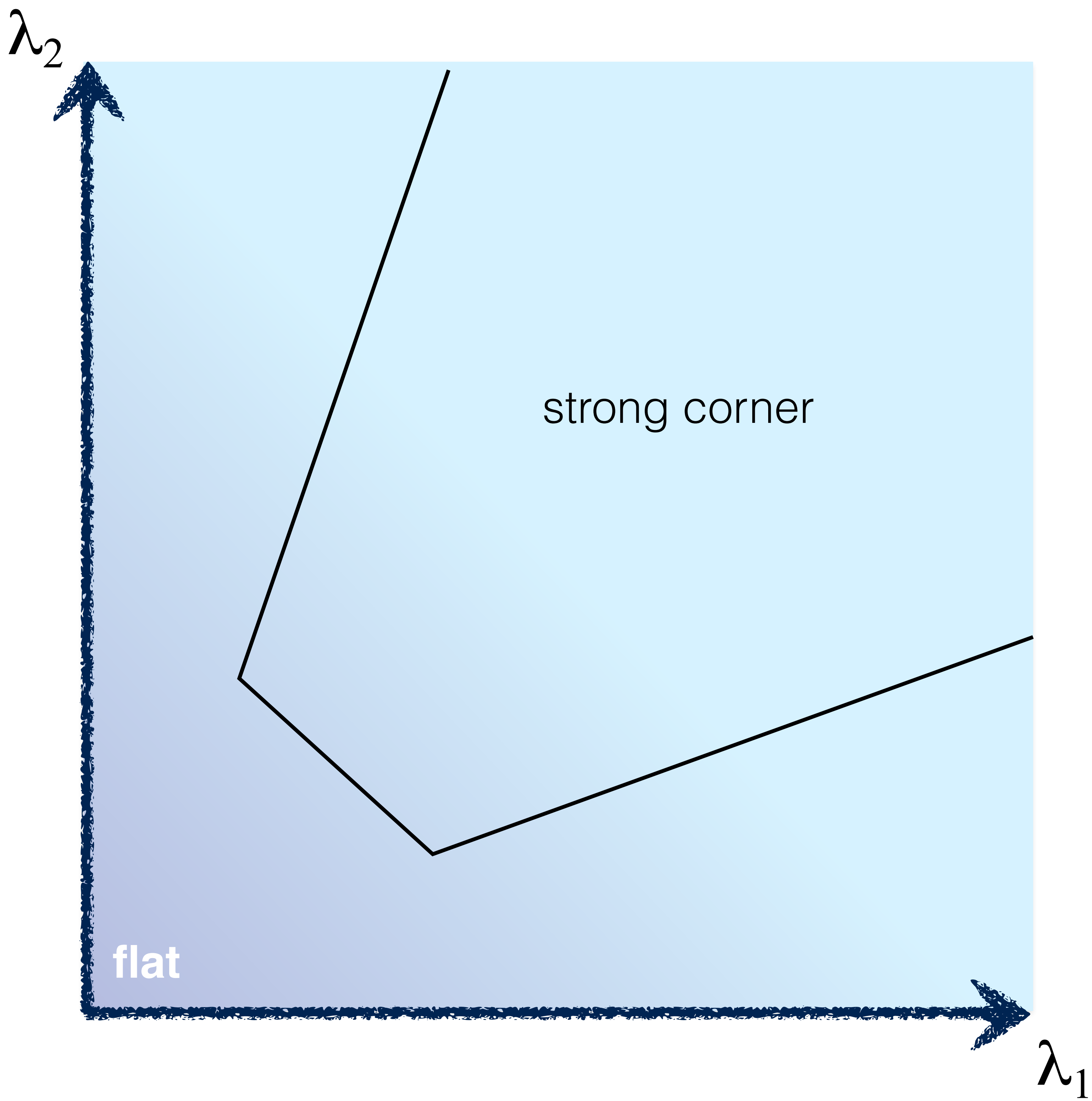
5. Use threshold on eigenvalues to detect corners

5. Use threshold on eigenvalues to detect corners



Think of a function to score ‘cornerness’

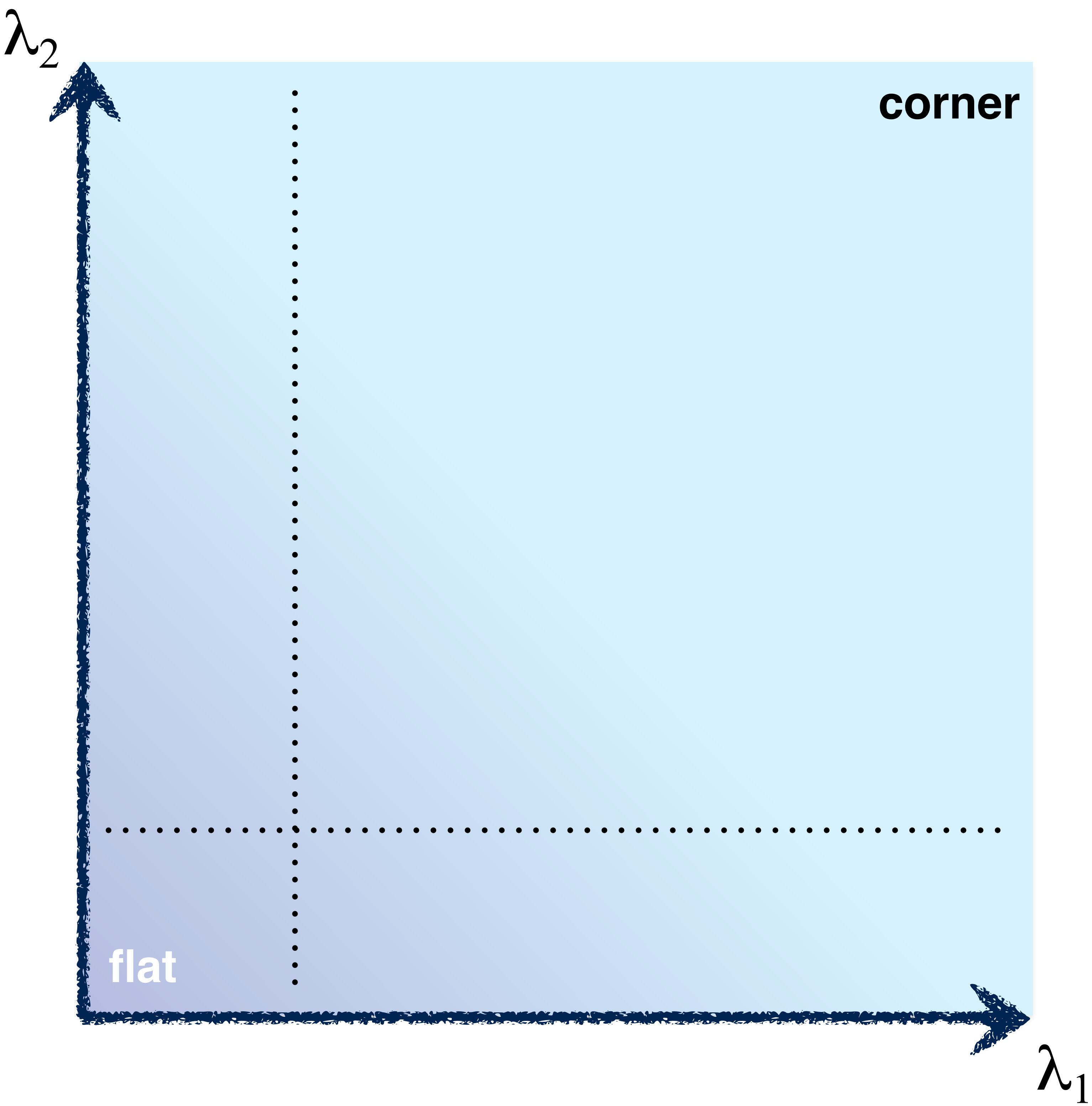
5. Use threshold on eigenvalues to detect corners



Think of a function to score ‘cornerness’

5. Use threshold on eigenvalues to detect corners

(a function of)

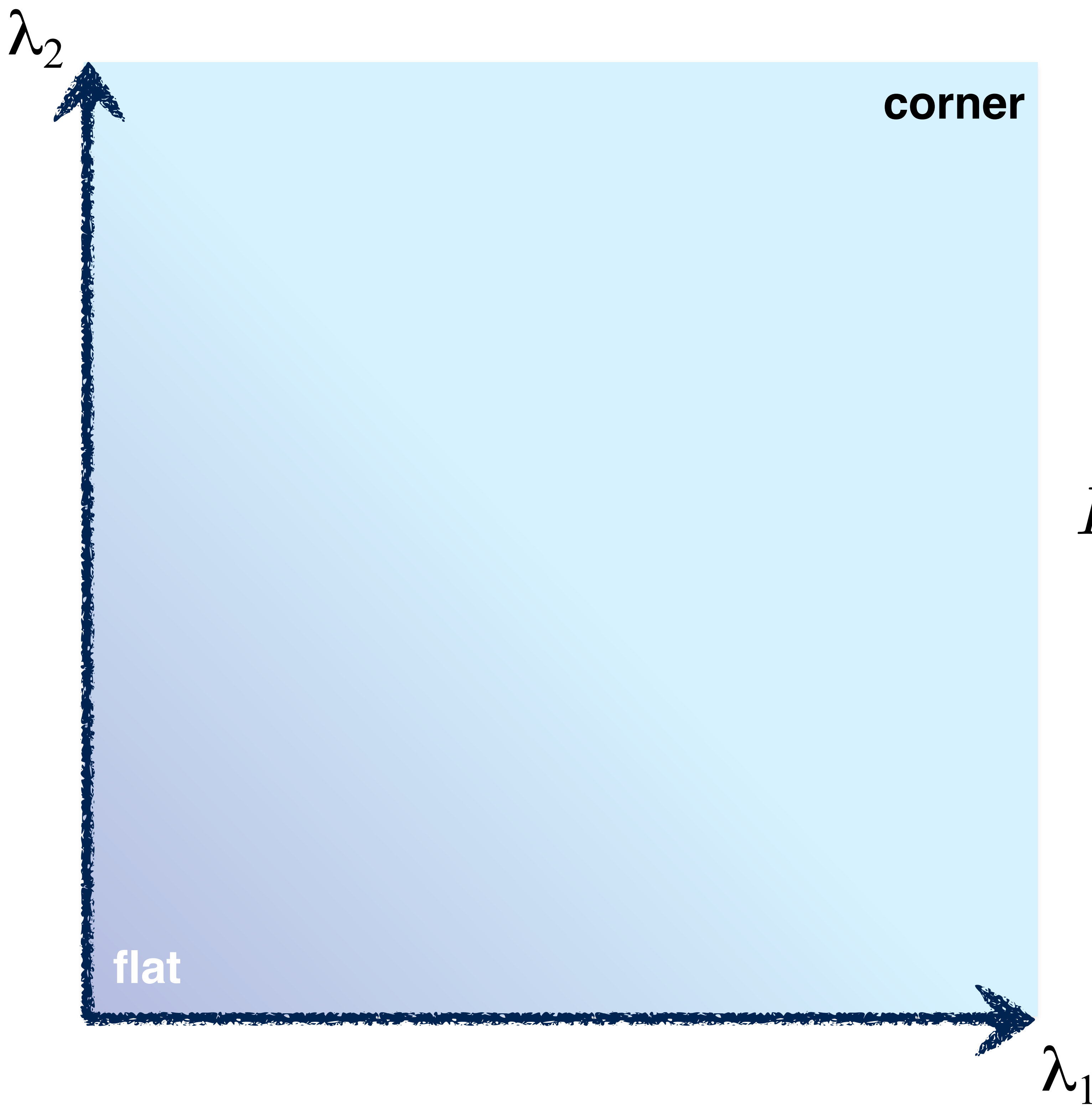


Use the smallest eigenvalue as the response function

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

5. Use threshold on eigenvalues to detect corners

(a function of)



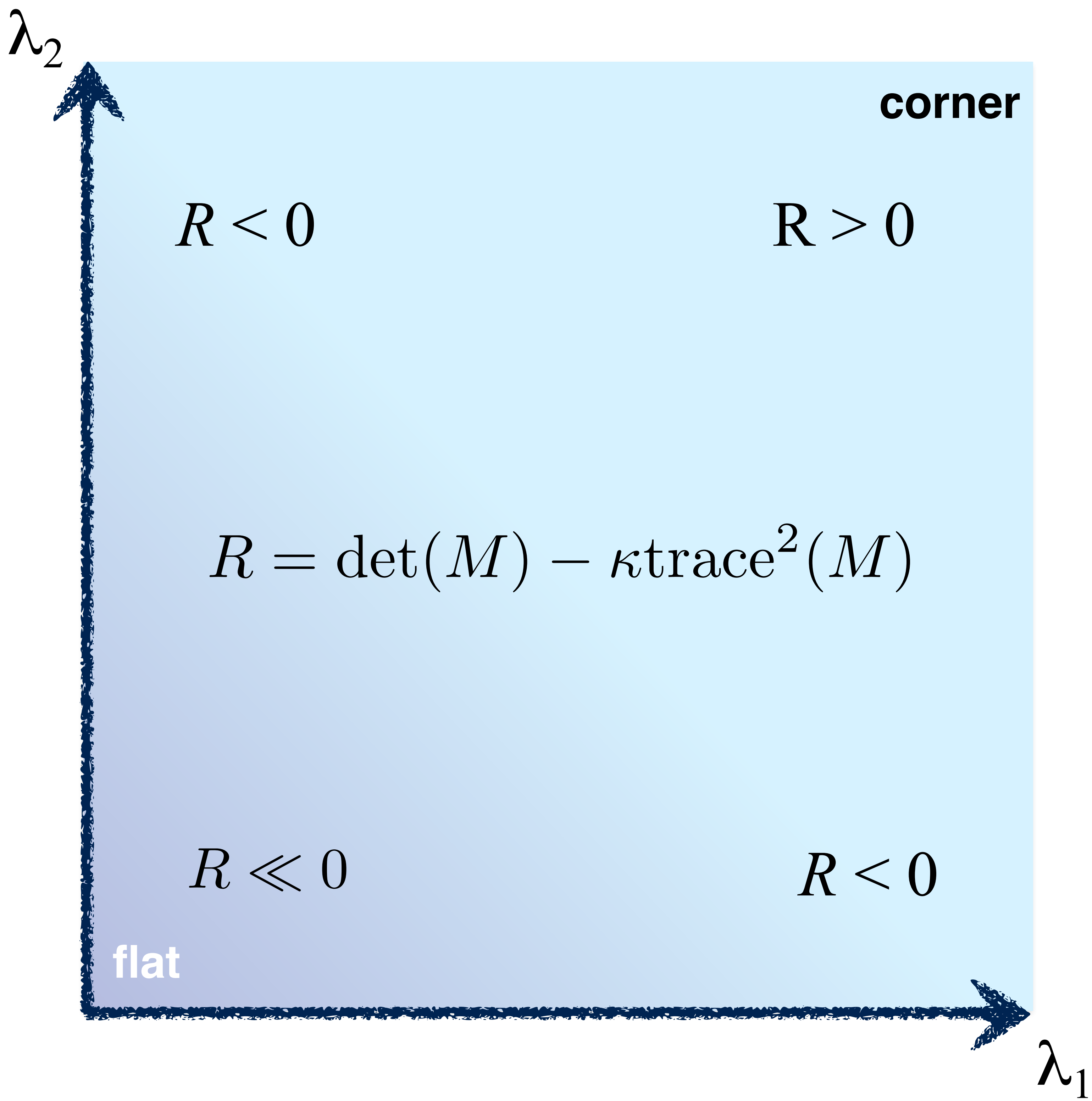
Eigenvalues need to be bigger than one.

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

Can compute this more efficiently…

5. Use threshold on eigenvalues to detect corners

(a function of)



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

$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

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

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

Harris & Stephens (1988)

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

Kanade & Tomasi (1994)

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

Nobel (1998)

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

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

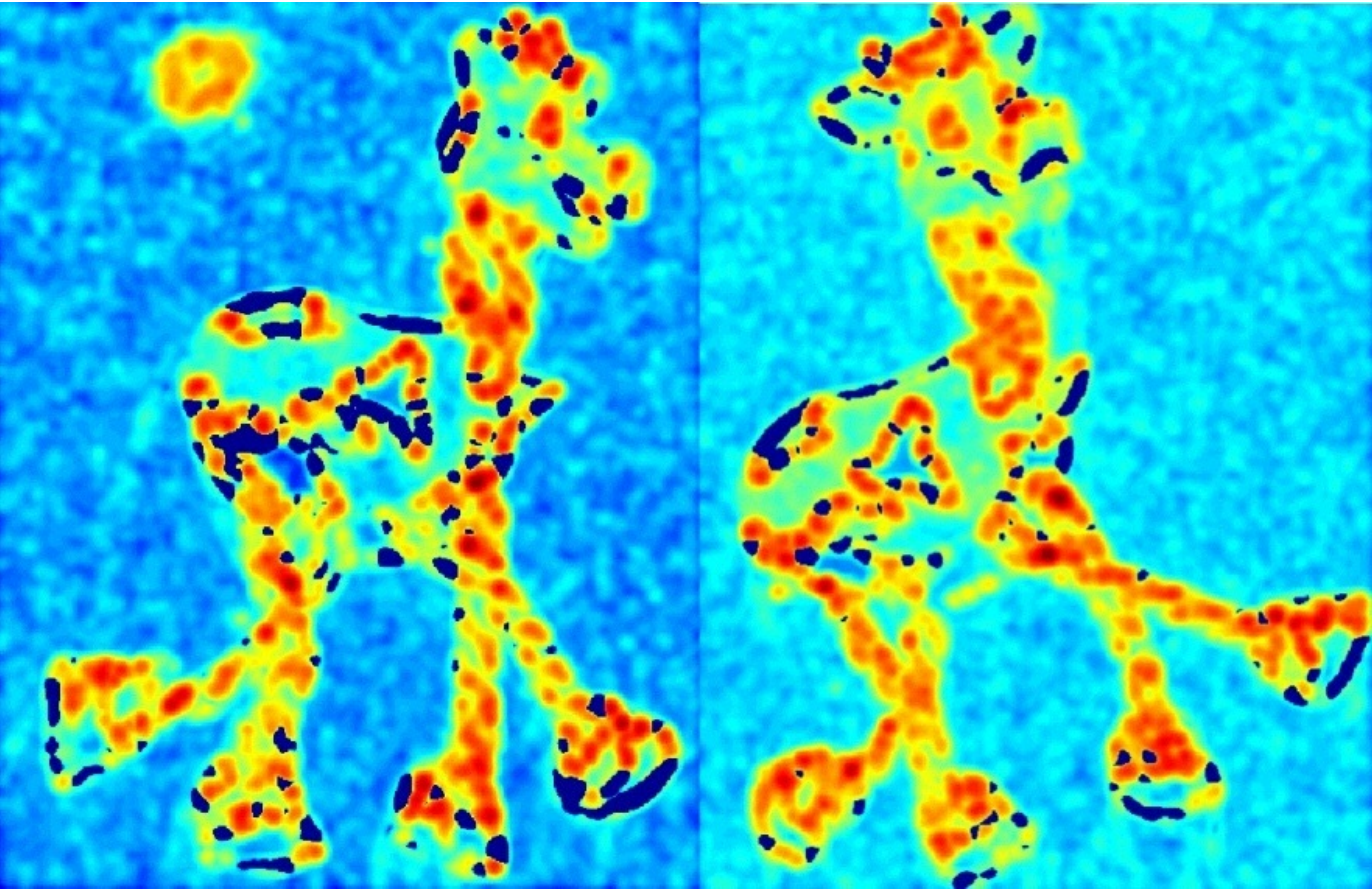
$$M(x,y) = \begin{bmatrix} S_{x^2}(x,y) & S_{xy}(x,y) \\ S_{xy}(x,y) & S_{y^2}(x,y) \end{bmatrix}$$

5. Compute the response of the detector at each pixel

$$R = \det M - k(\mathrm{trace} M)^2$$

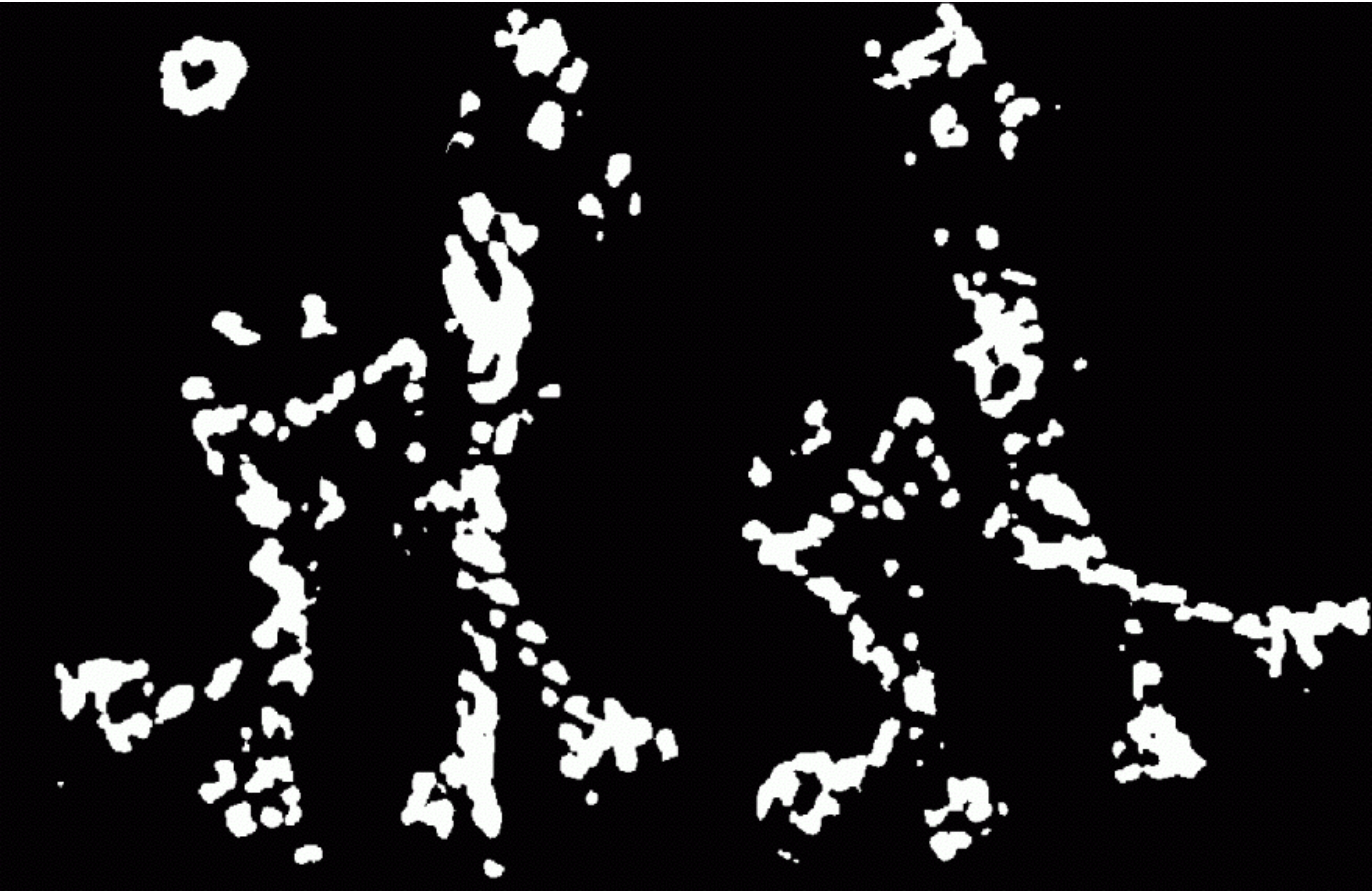
6. Threshold on value of R; compute non-max suppression.


Corner response

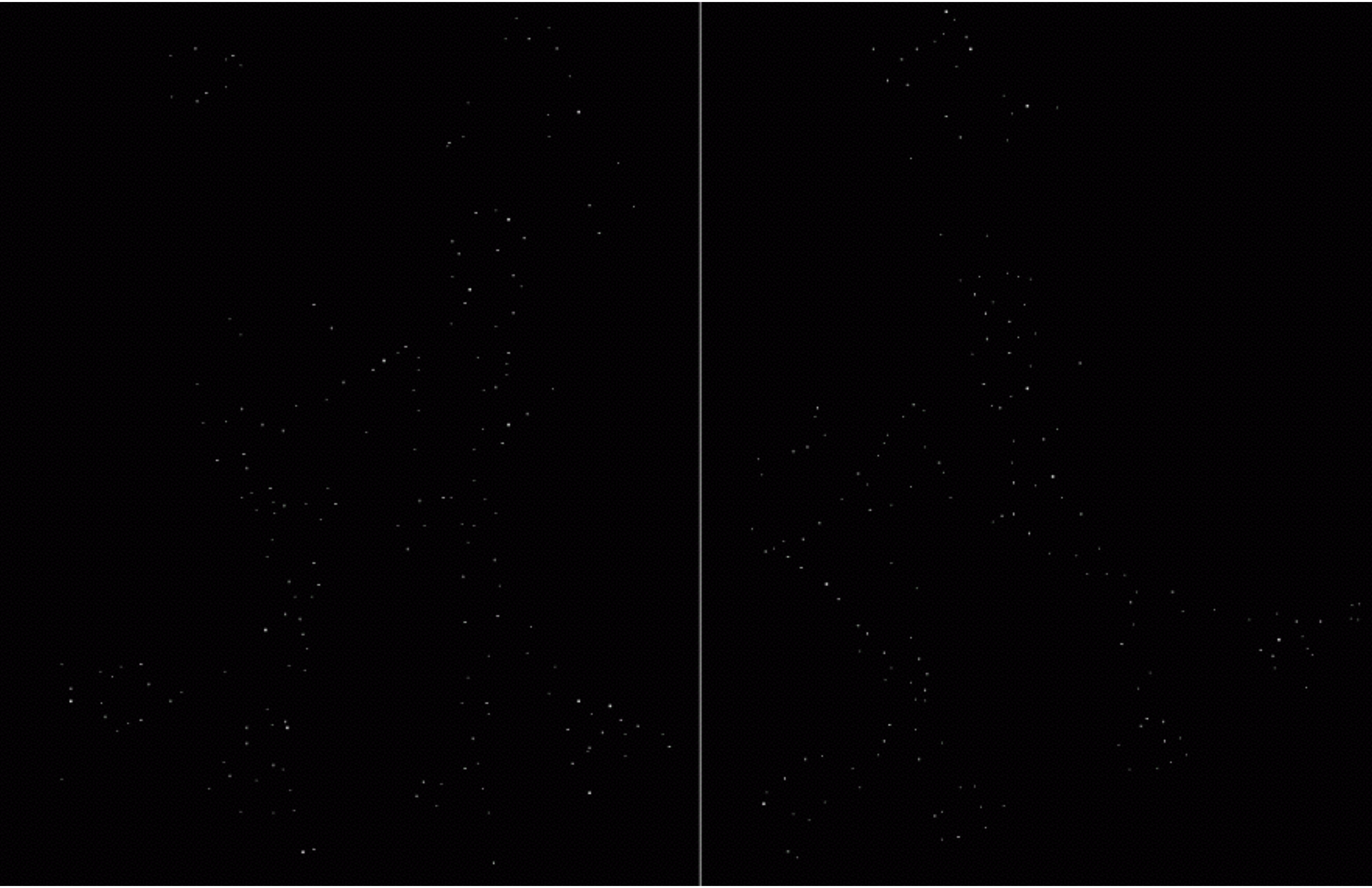


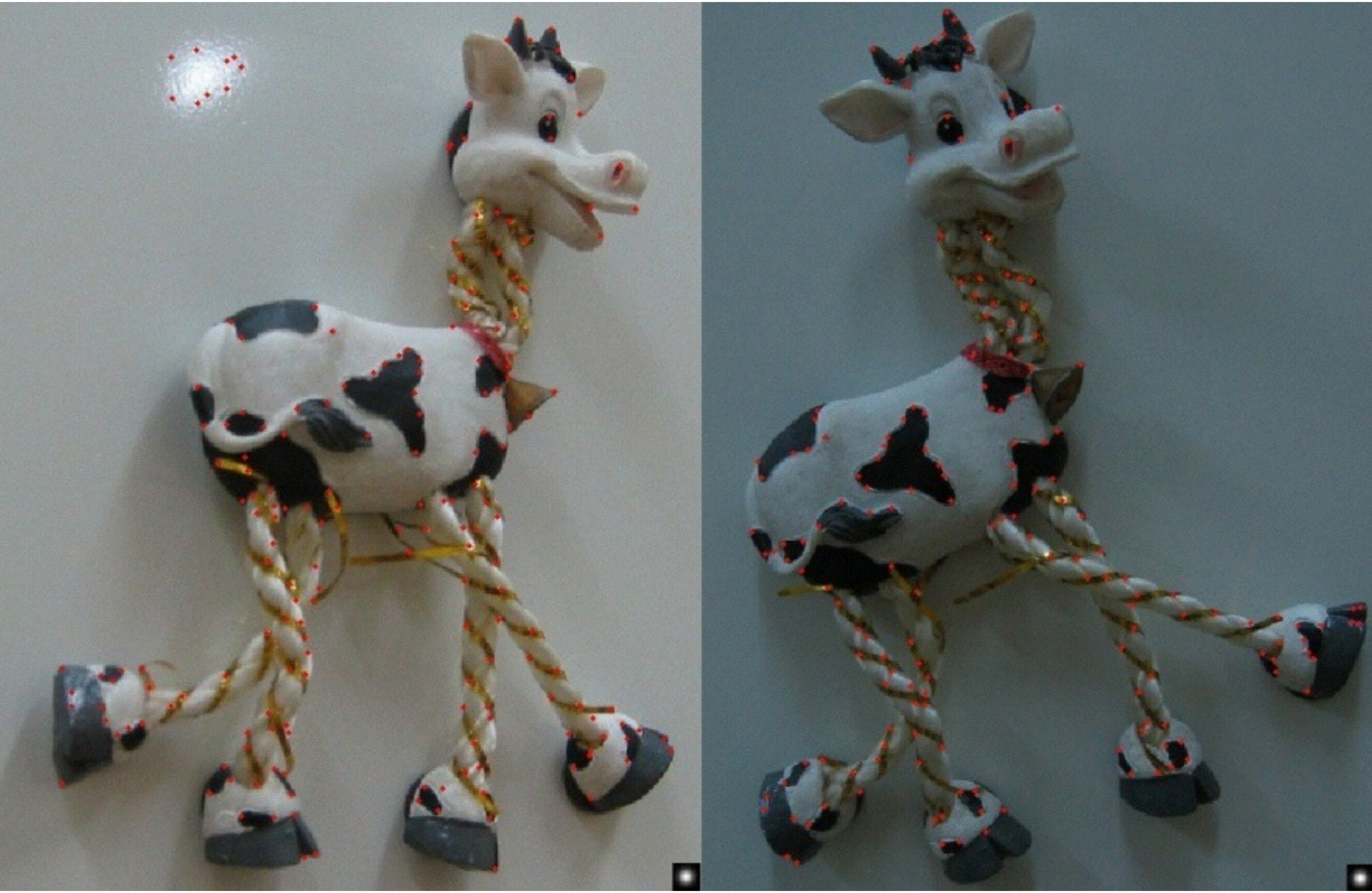
 

Thresholded corner response

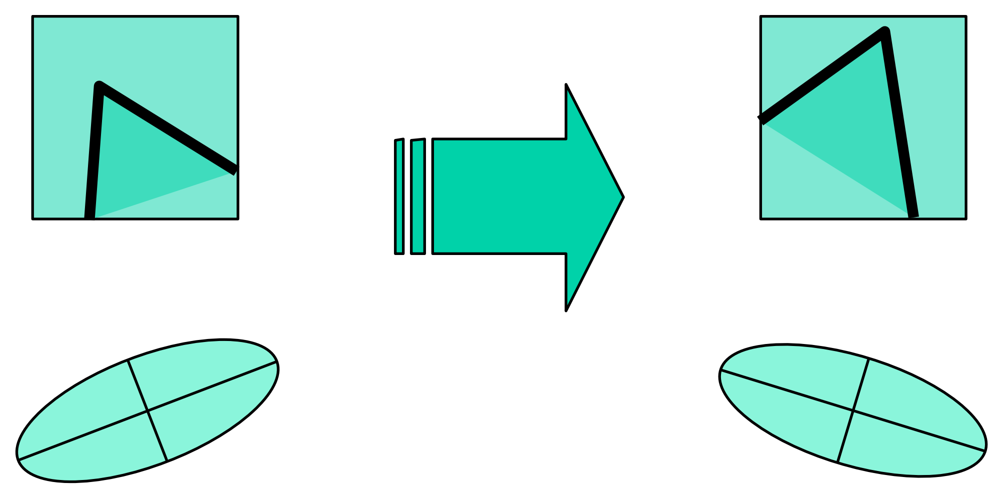


Non-maximal suppression





rotation invariance



Ellipse rotates but its shape
(i.e. eigenvalues) remains the same

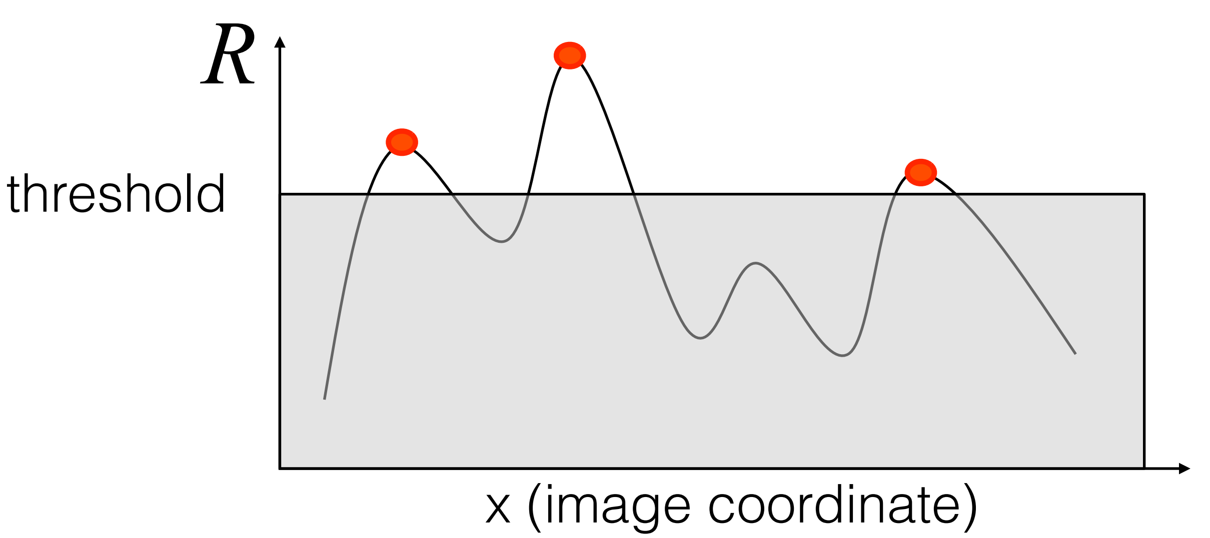
Corner response R is invariant to image rotation

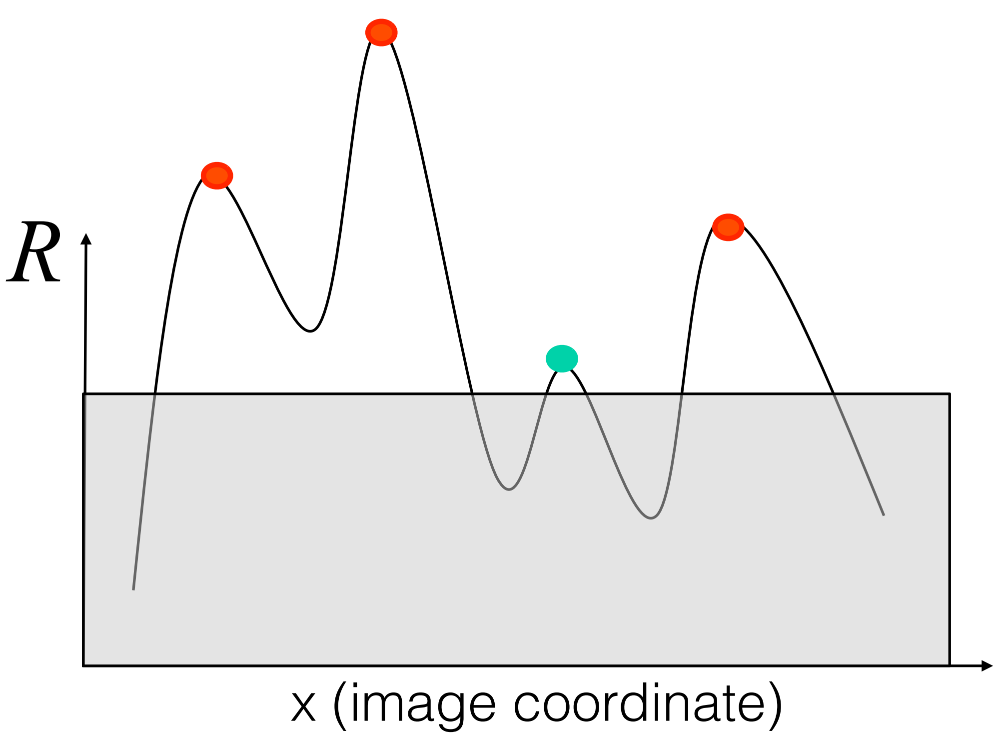
intensity changes

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 not invariant to changes in …

Multi-scale Detection

16-385 Computer Vision

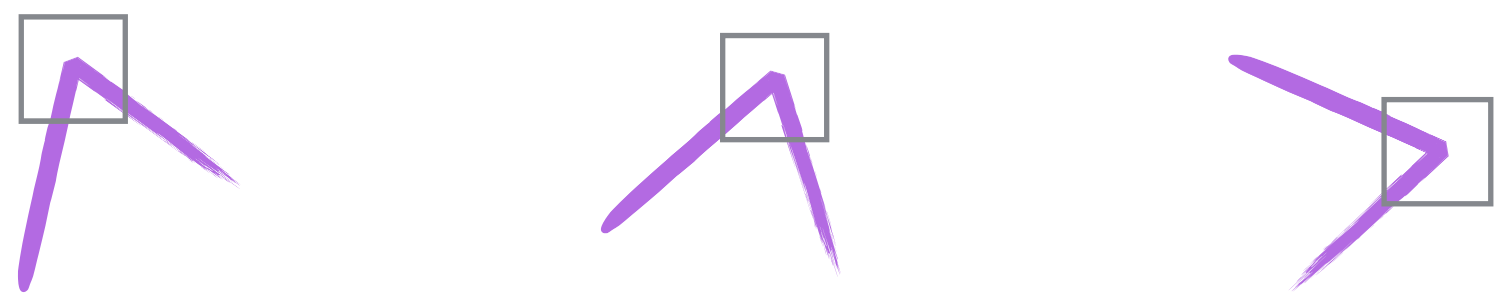
Properties of the Harris corner detector

Rotation invariant?

Scale invariant?

Properties of the Harris corner detector

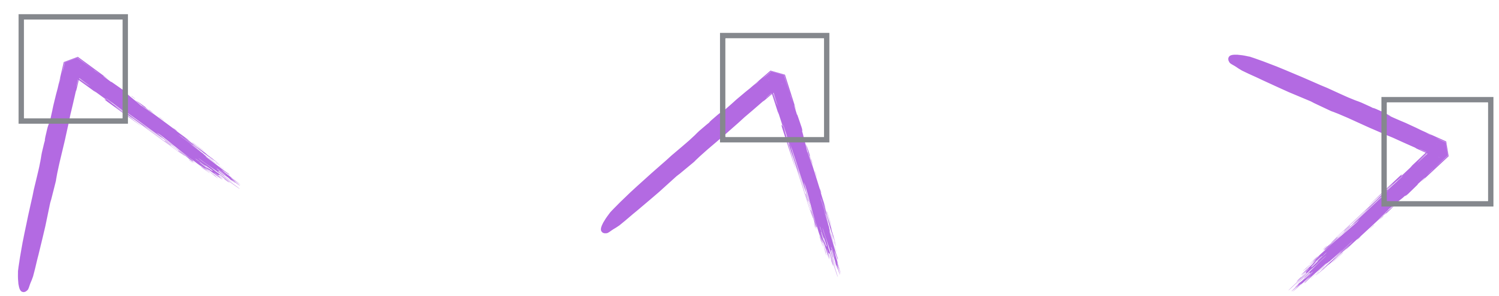
Rotation invariant? 



Scale invariant?

Properties of the Harris corner detector

Rotation invariant?

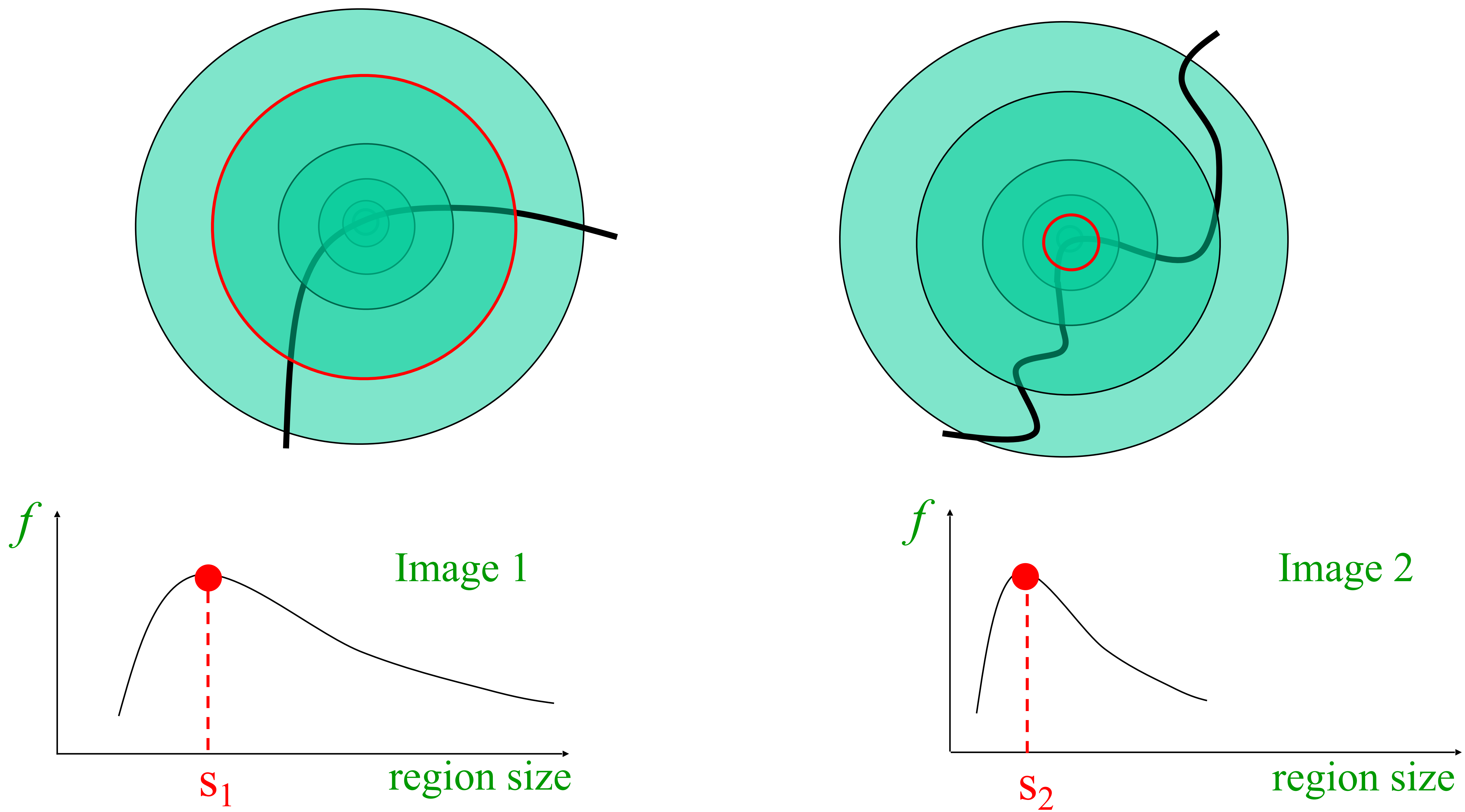


Scale invariant?

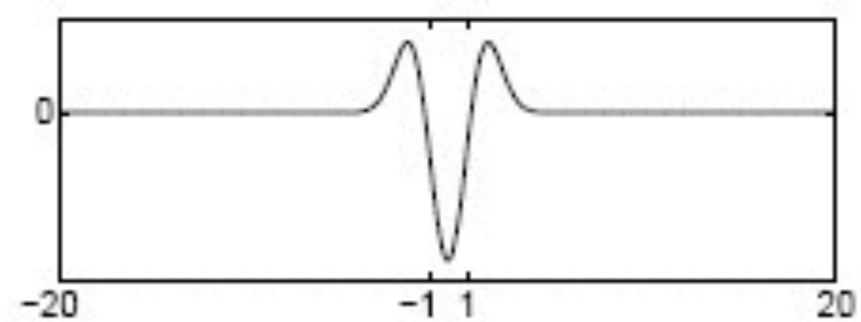
How can we make a feature detector scale-invariant?

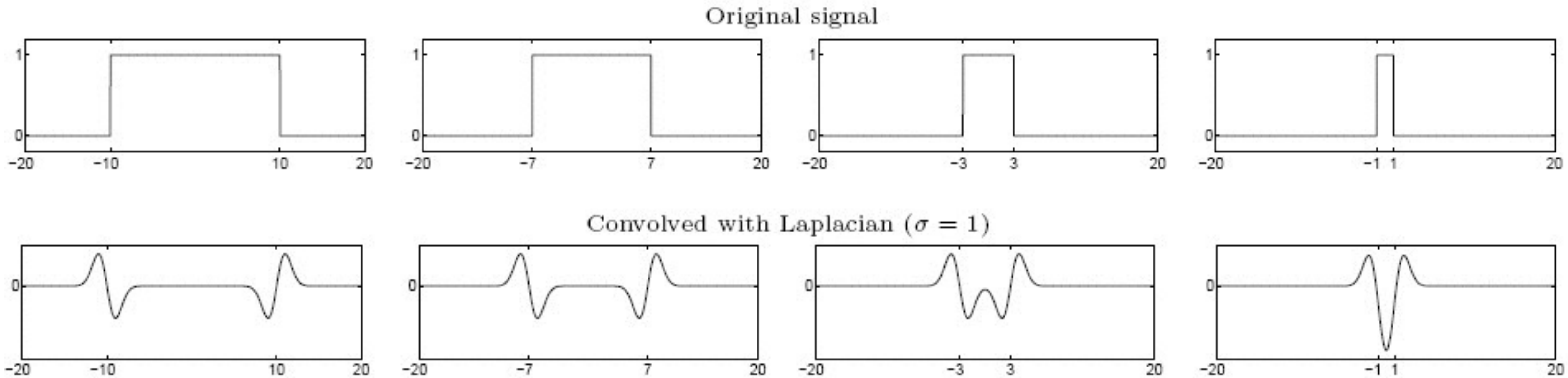
How can we automatically select the scale?

Find local maxima in both **position** and **scale**

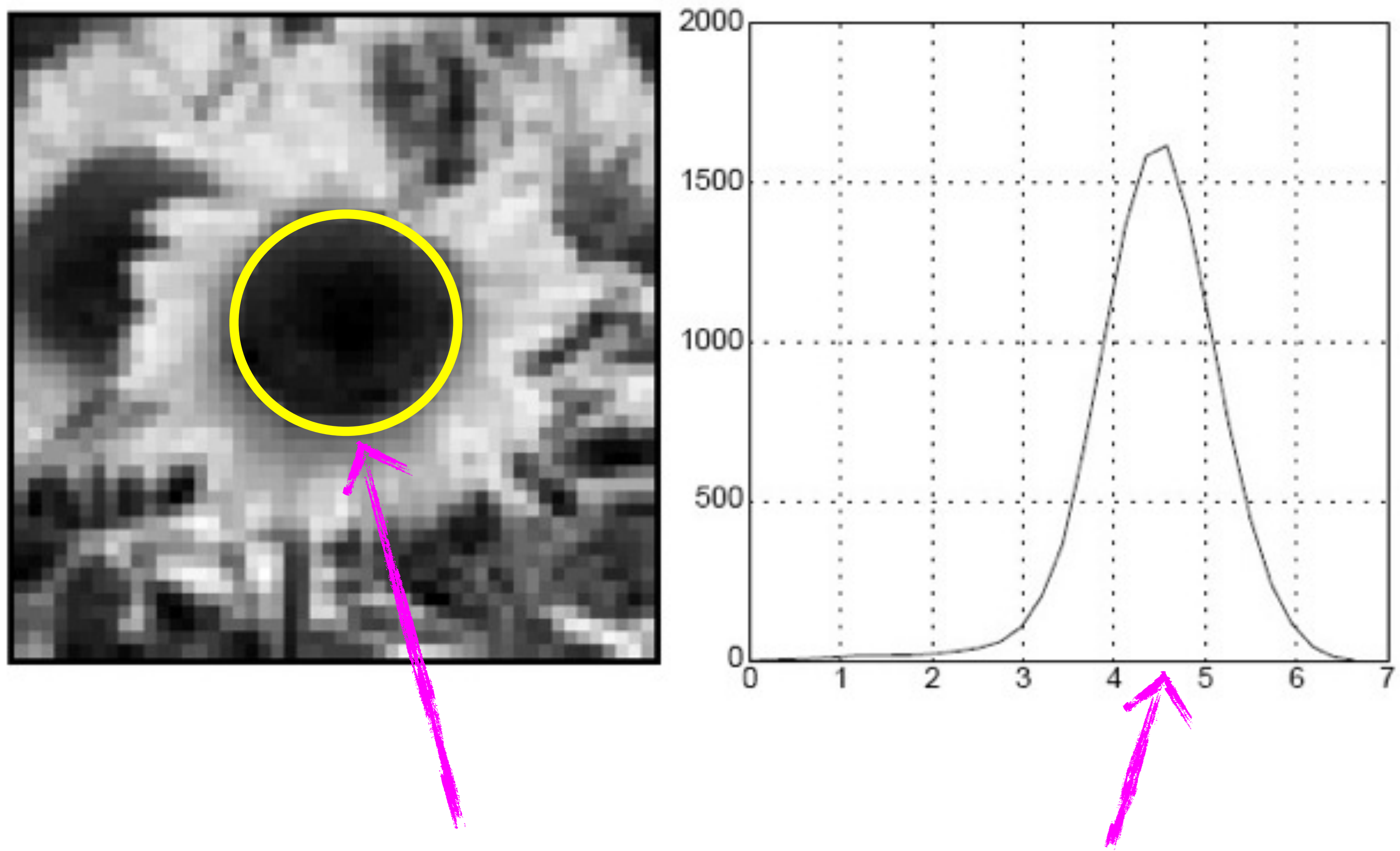


Laplacian filter





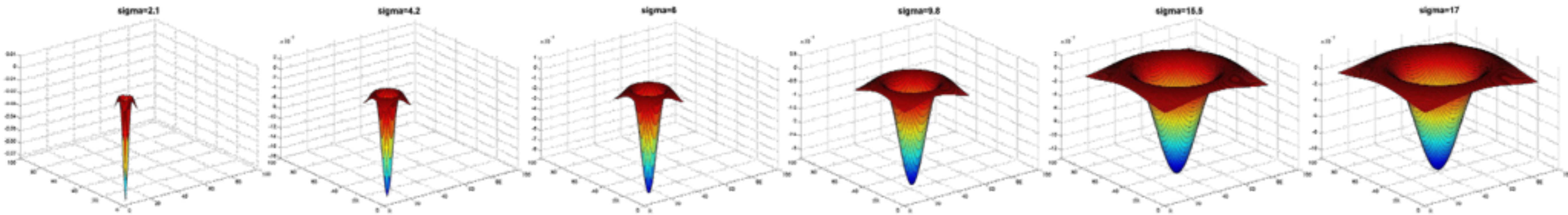
Highest response when the signal has the same **characteristic scale** as the filter characteristic scale - the scale that produces peak filter response



characteristic scale

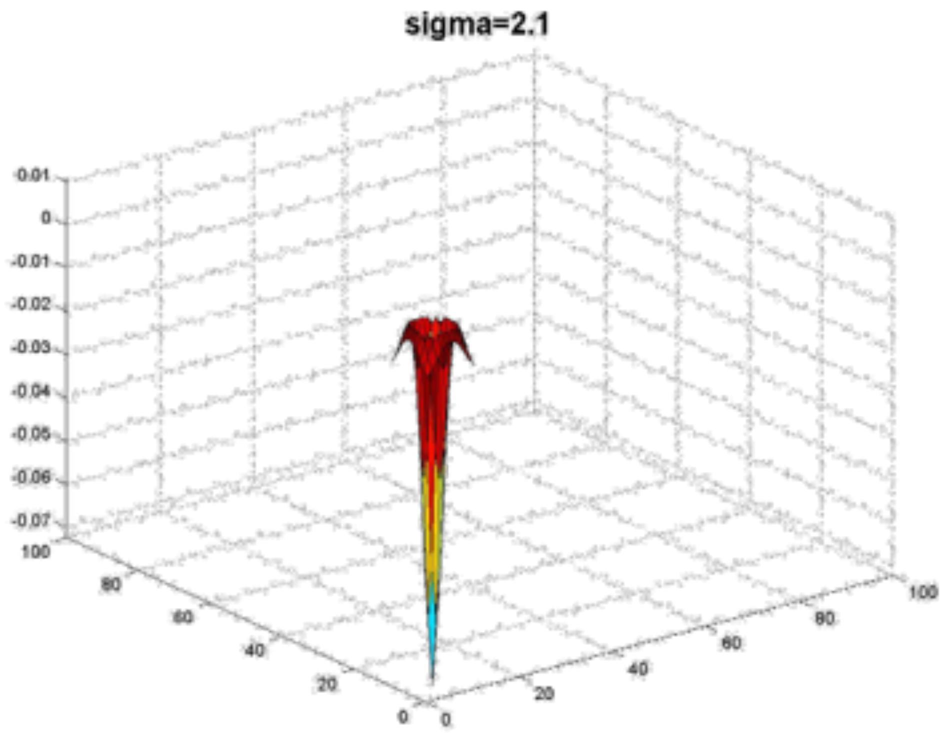
Multi-scale
2D Blob detection

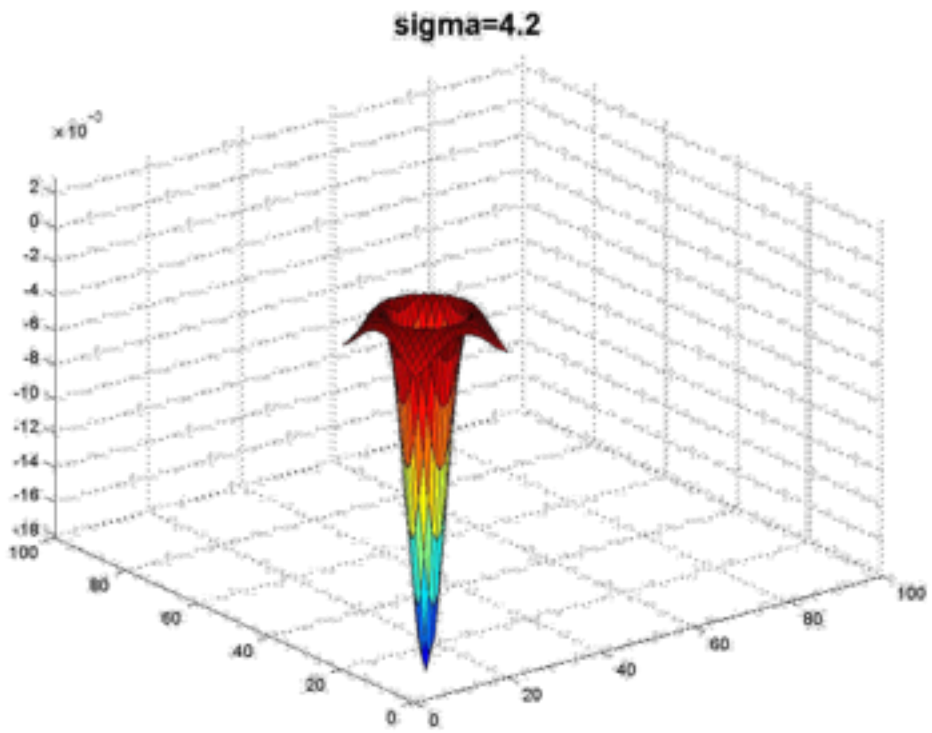
What happens if you apply different Laplacian filters?



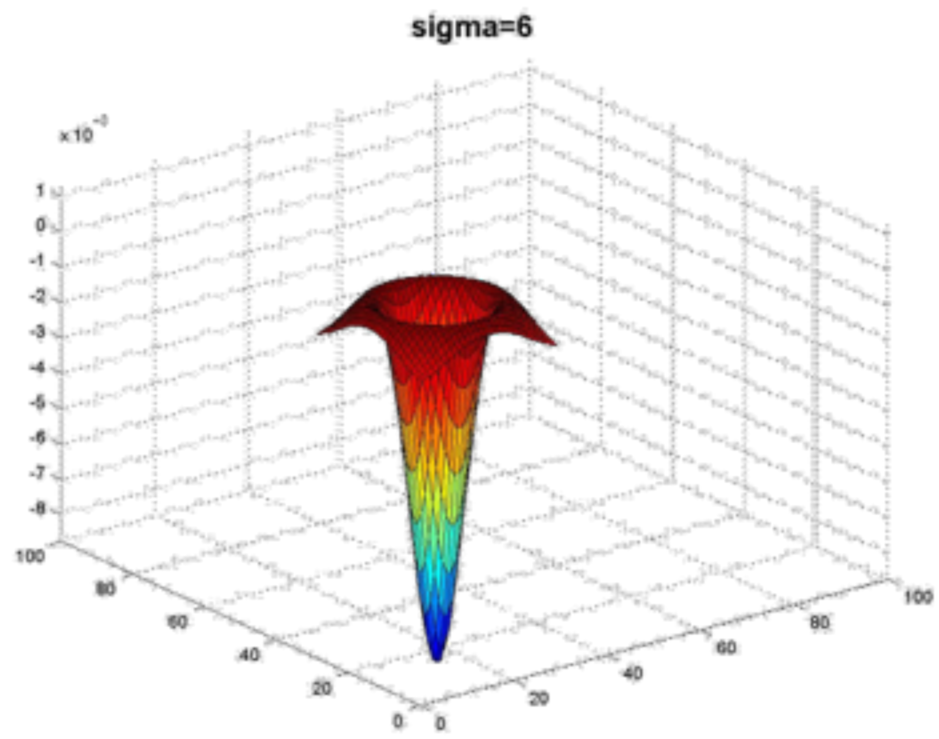
Full size

3/4 size

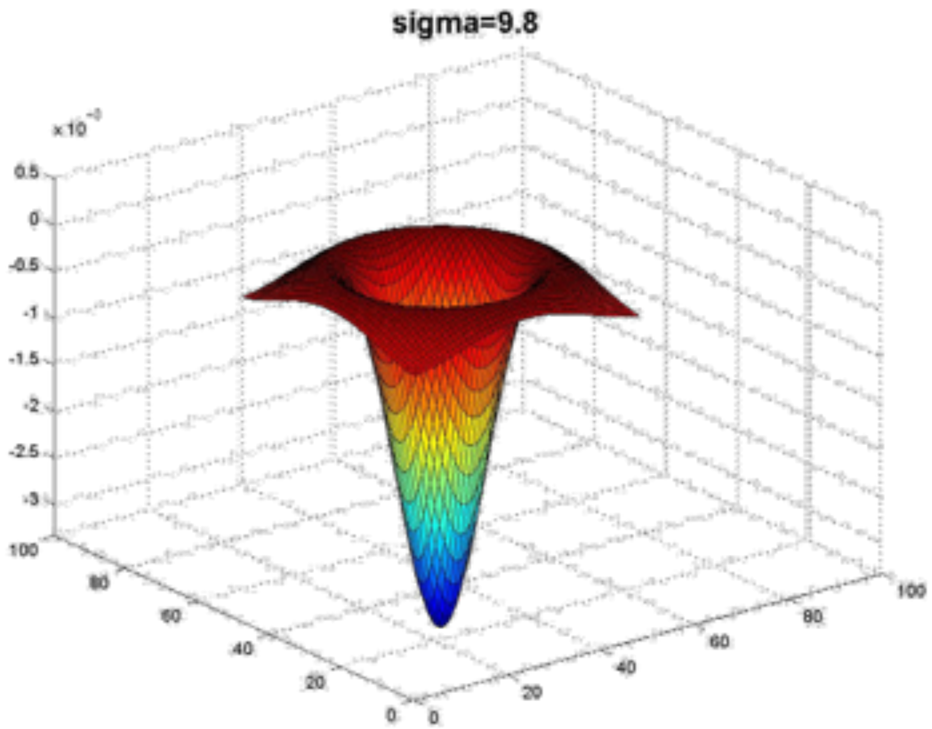

sigma=2.1

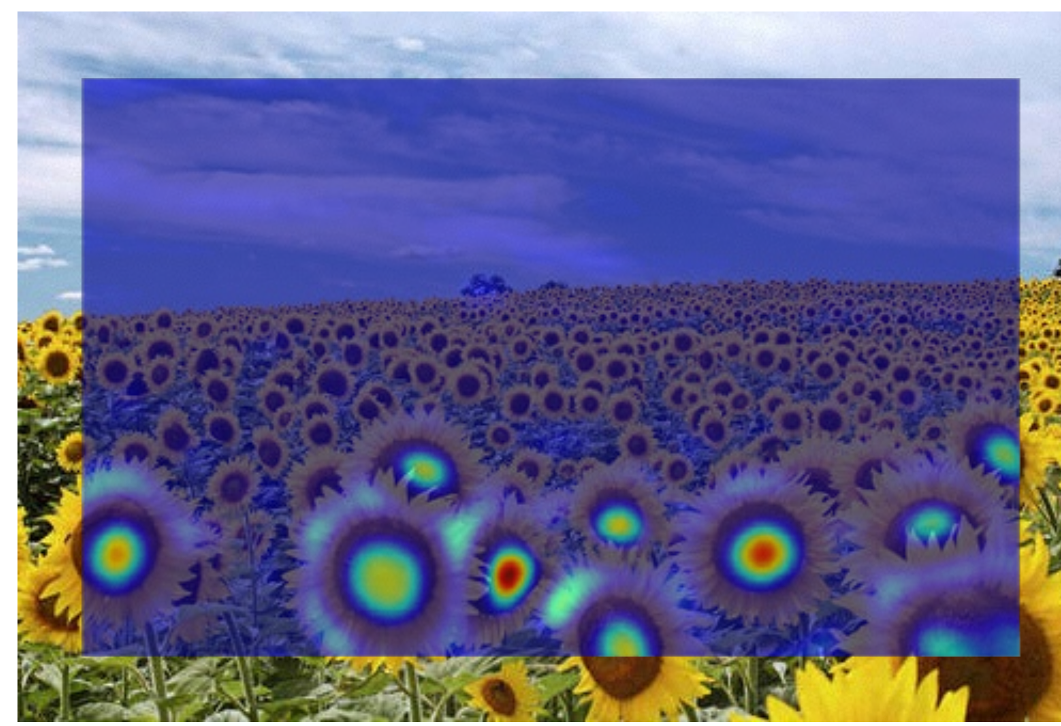


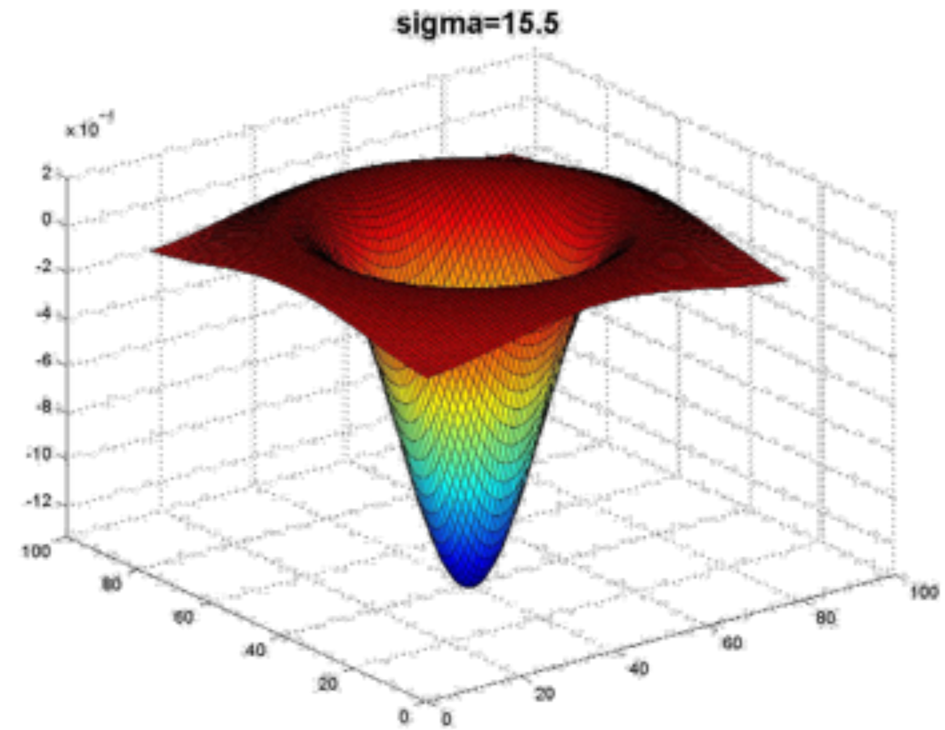
sigma=4.2

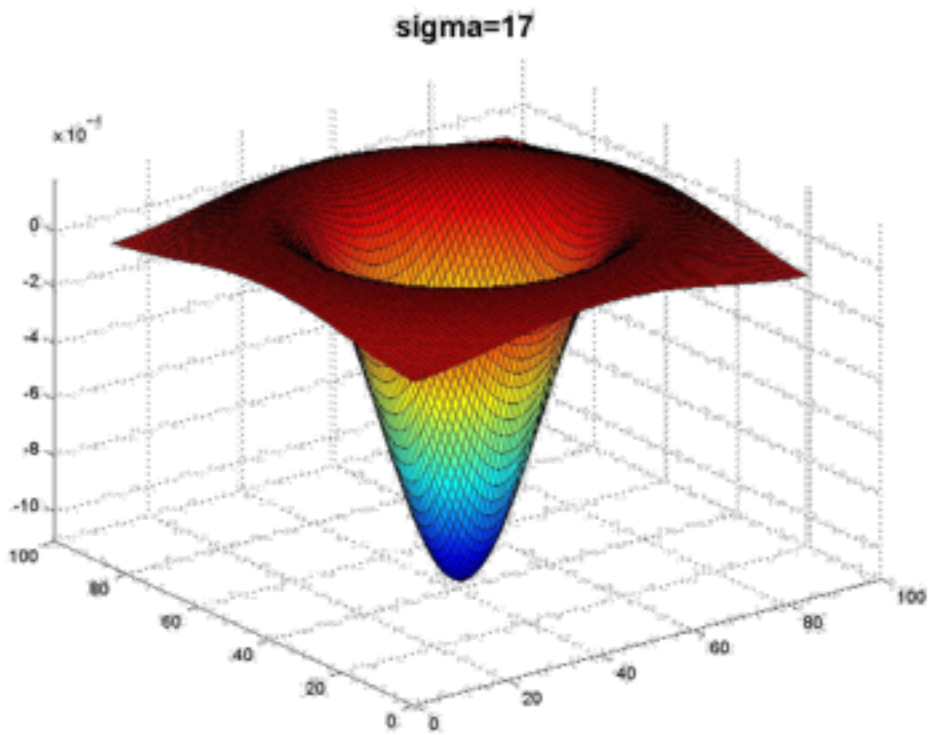


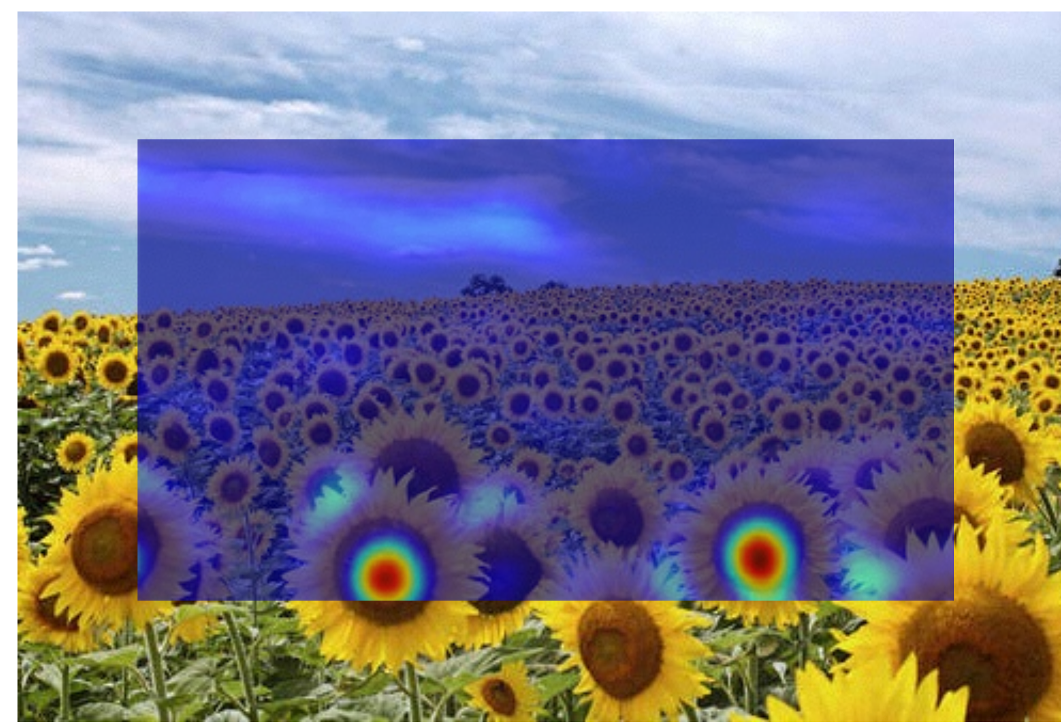
sigma=6


sigma=9.8




sigma=15.5

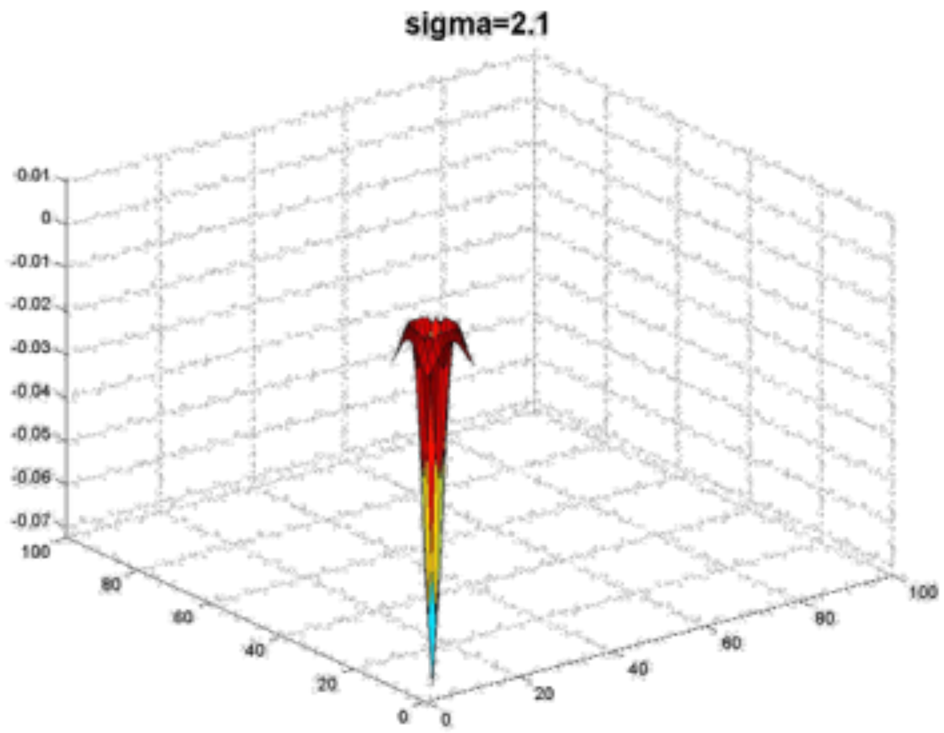

sigma=17
x 10^-4
0
-2
-4
-6
-8
-10
100
80
60
40
20
0
0
20
40
60
80
100

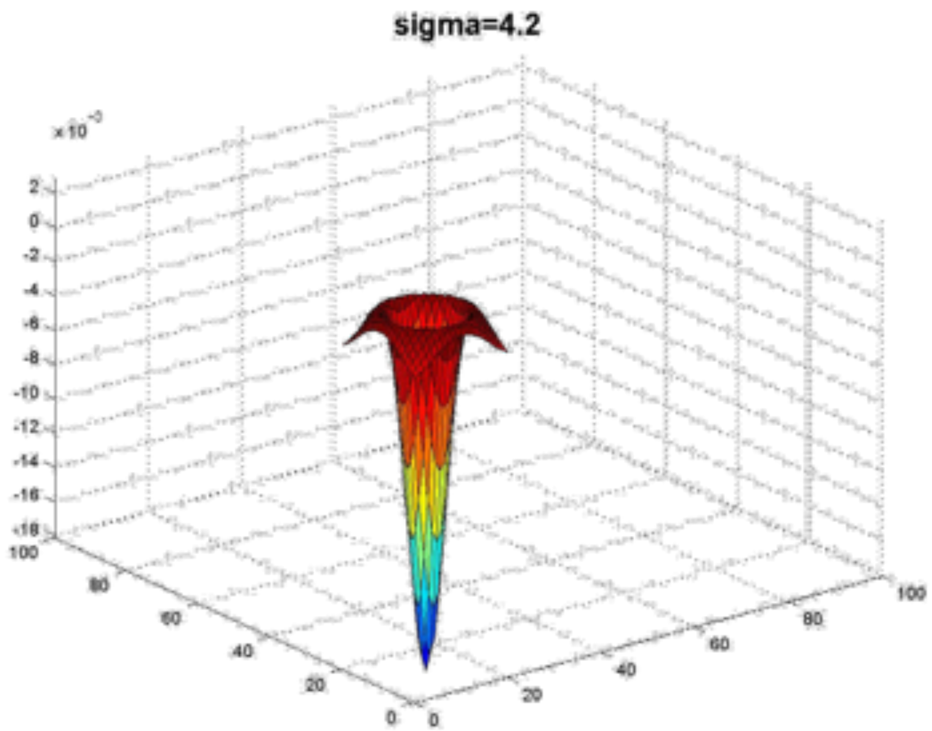


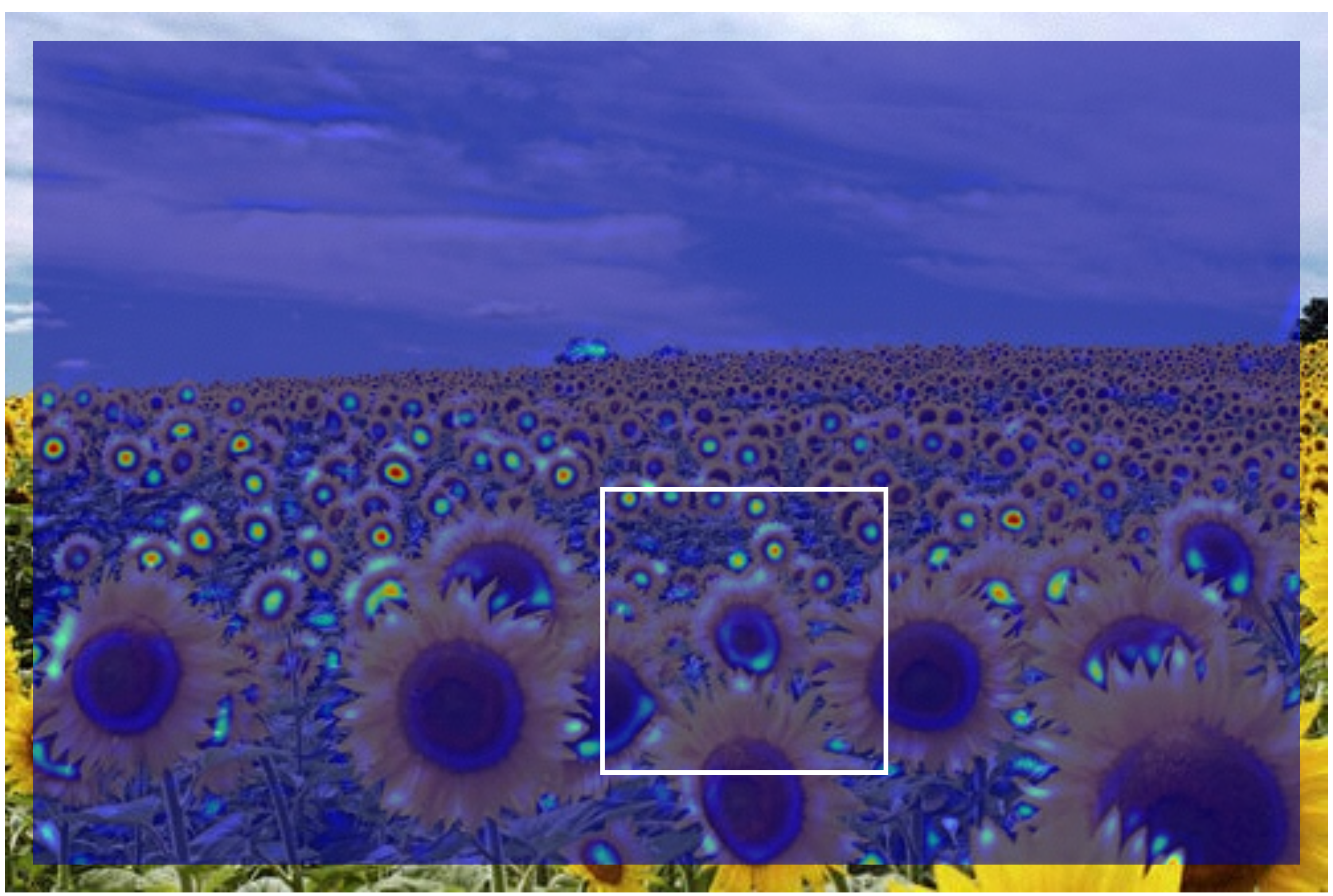
What happened when you applied different Laplacian filters?

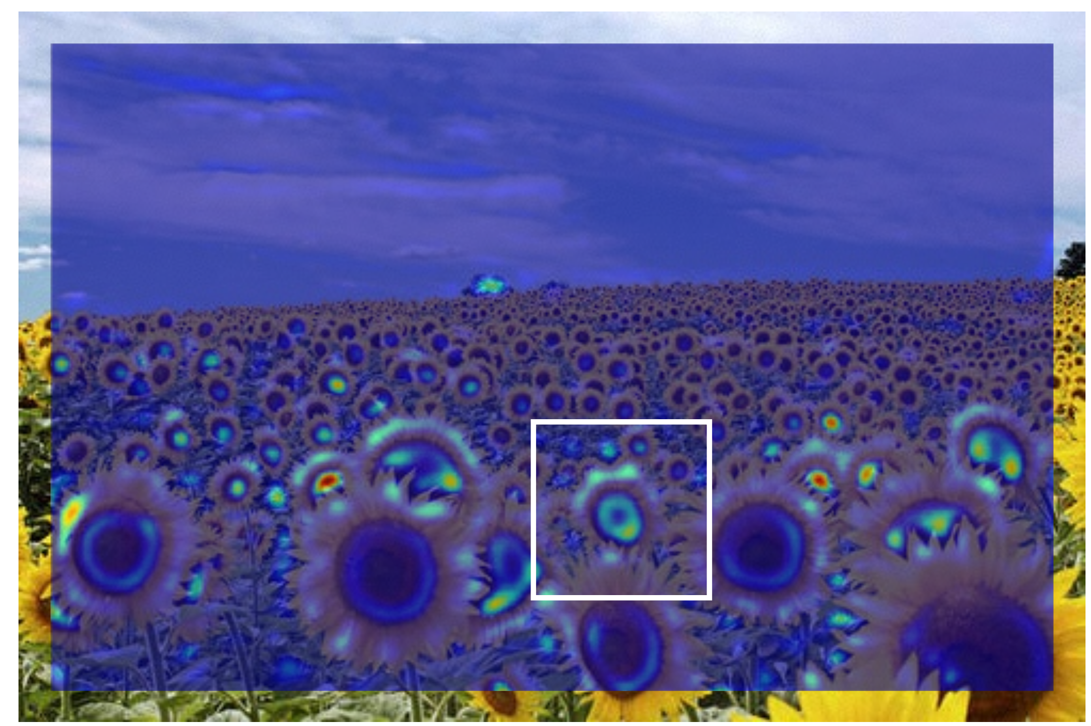
Full size

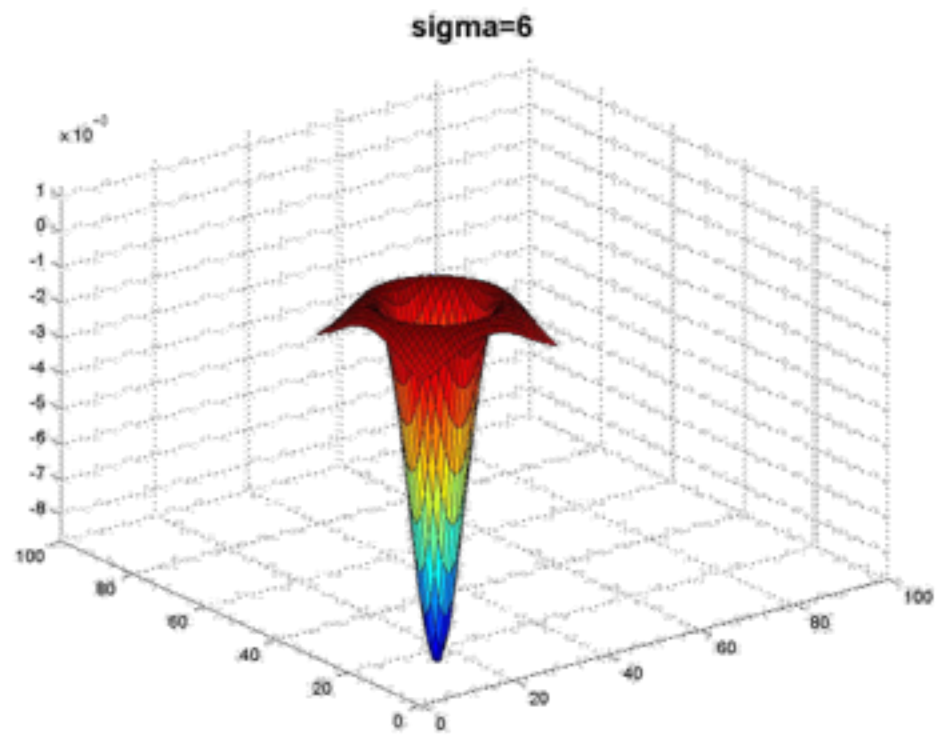
3/4 size

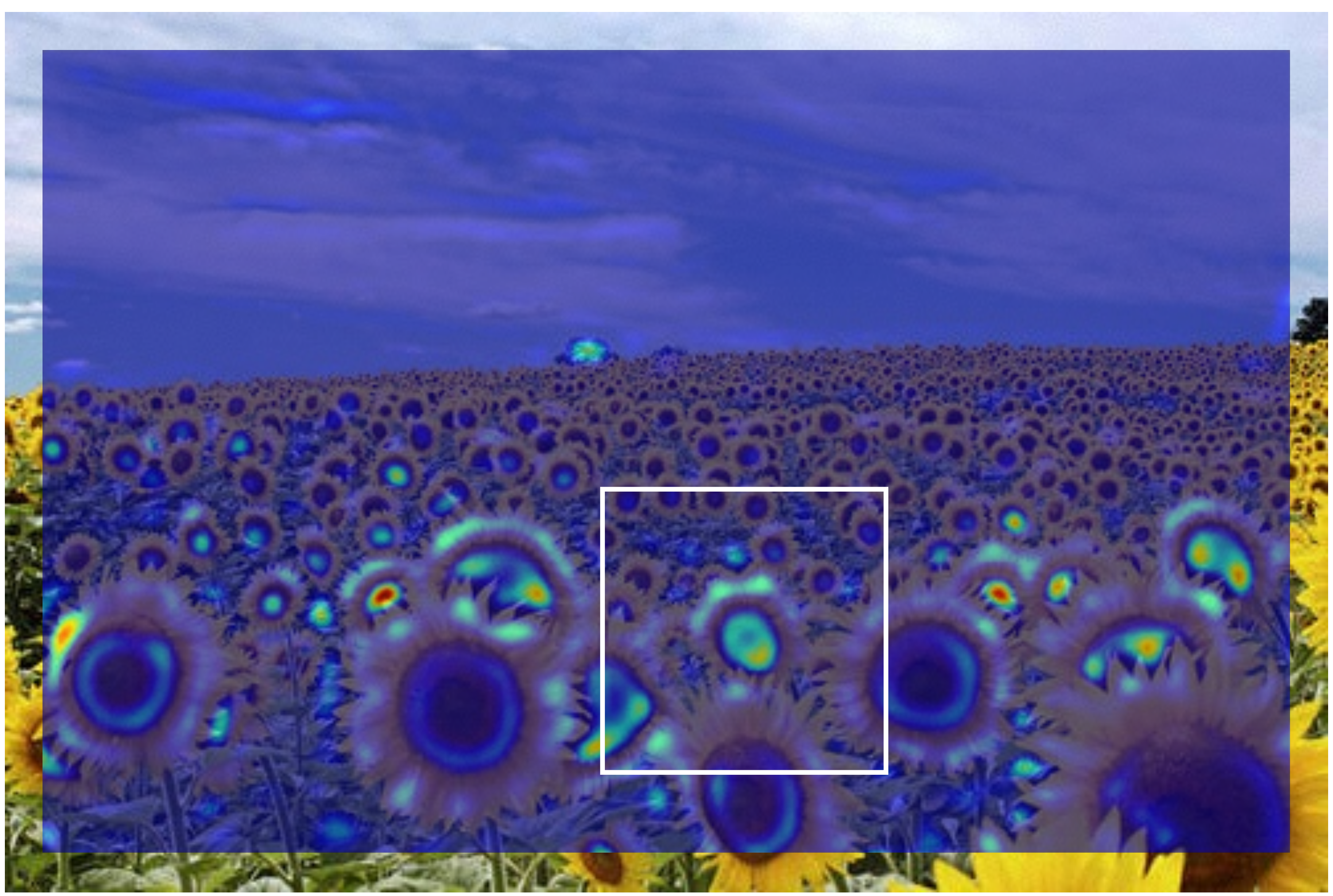

sigma=2.1

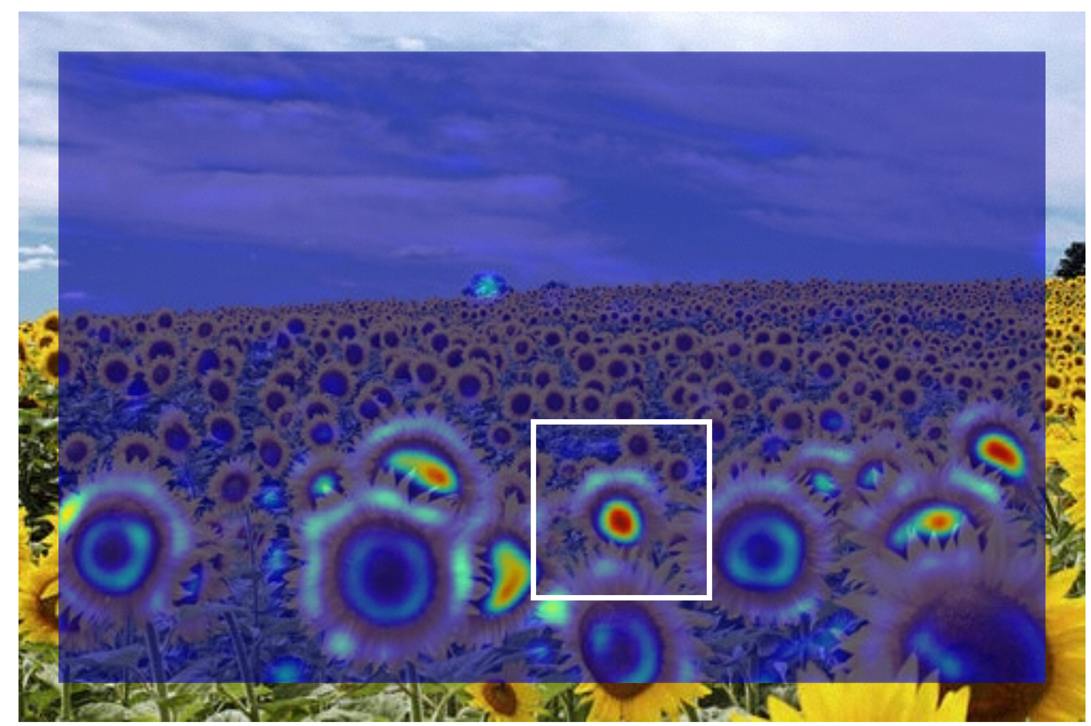

sigma=4.2
x 10-3
2
0
-2
-4
-6
-8
-10
-12
-14
-16
-18
100
80
60
40
20
0
0
20
40
60
80
100

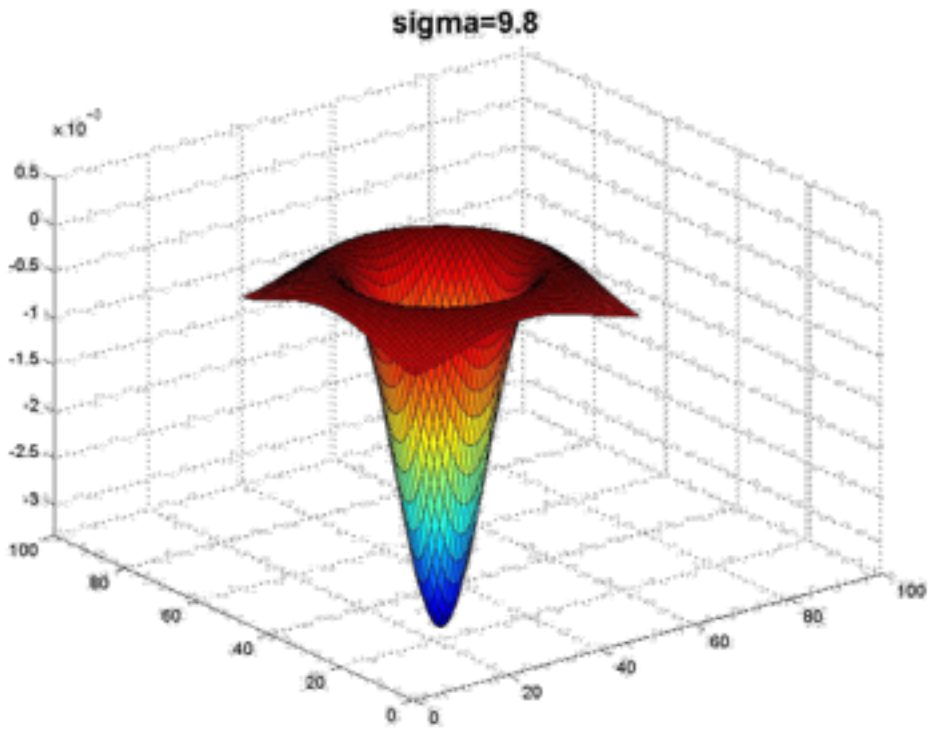


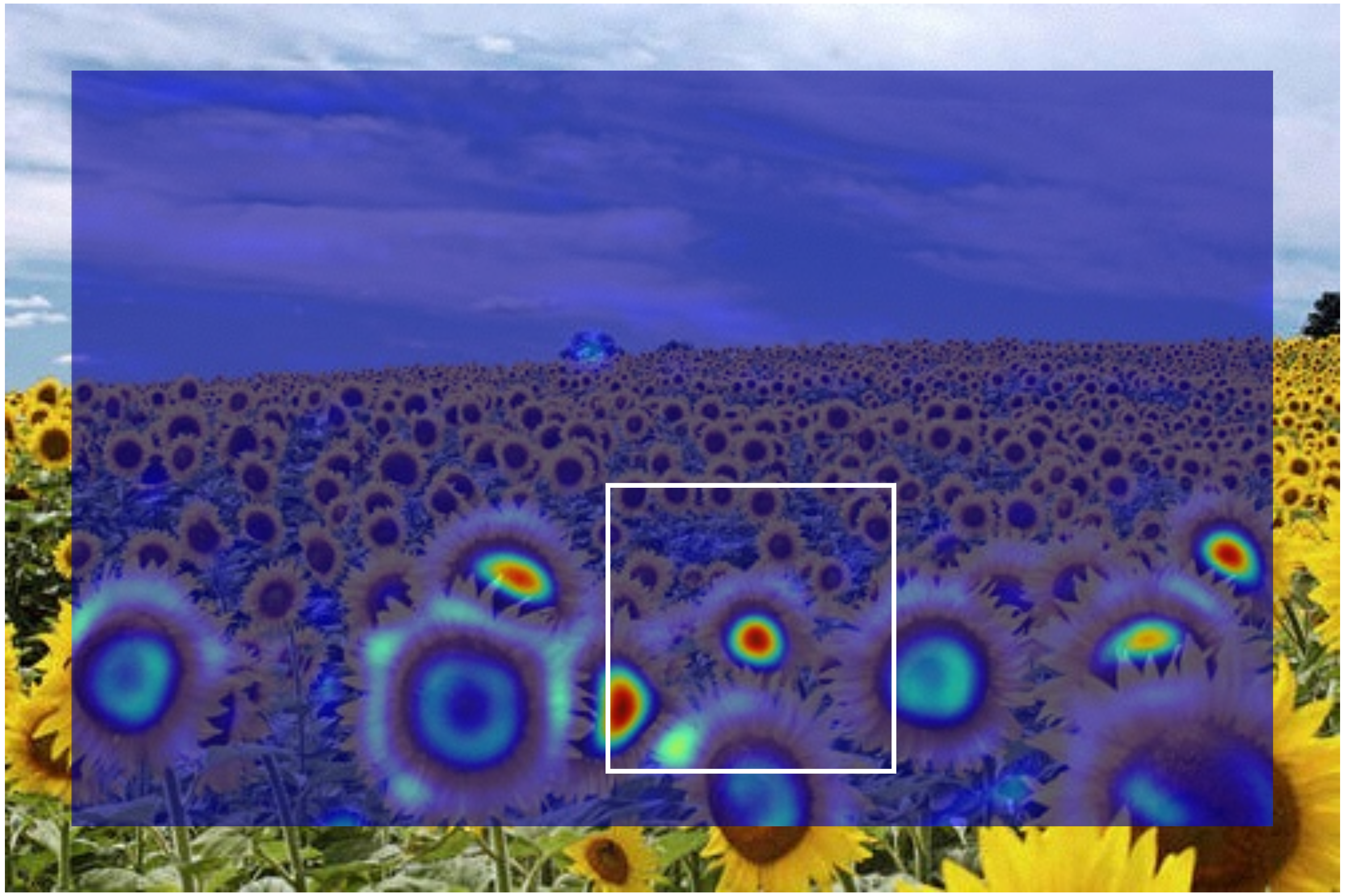


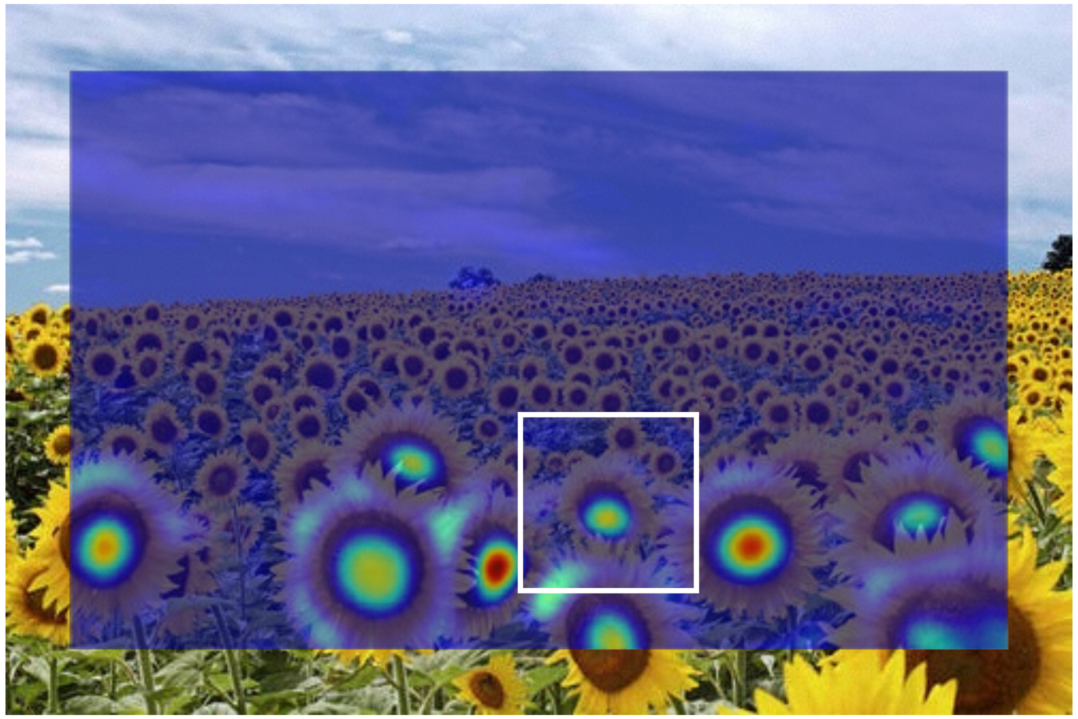

sigma=6

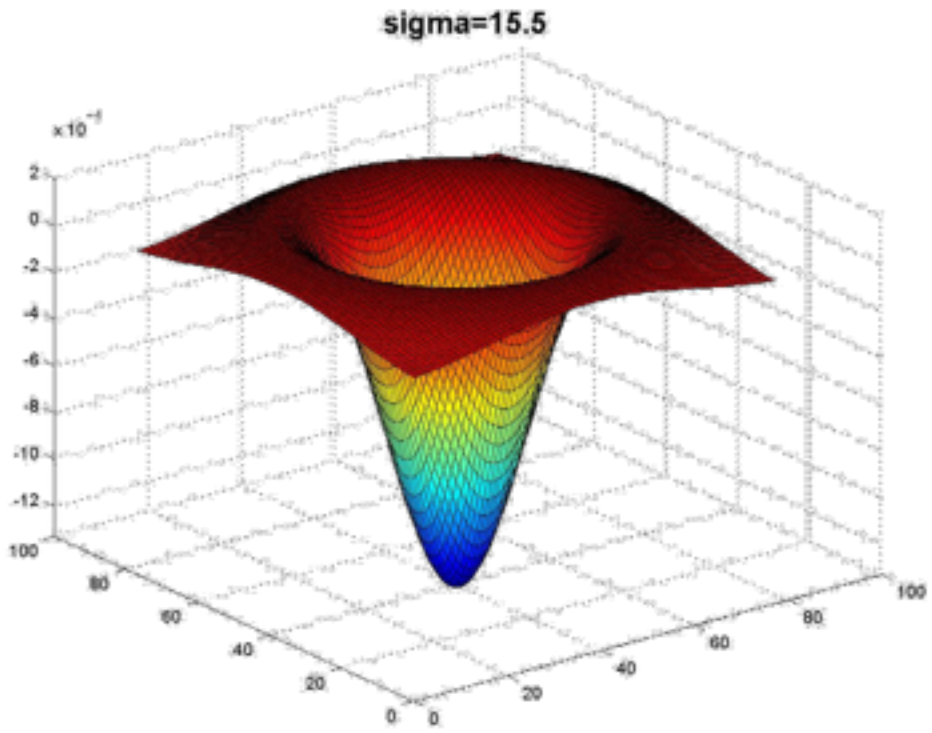




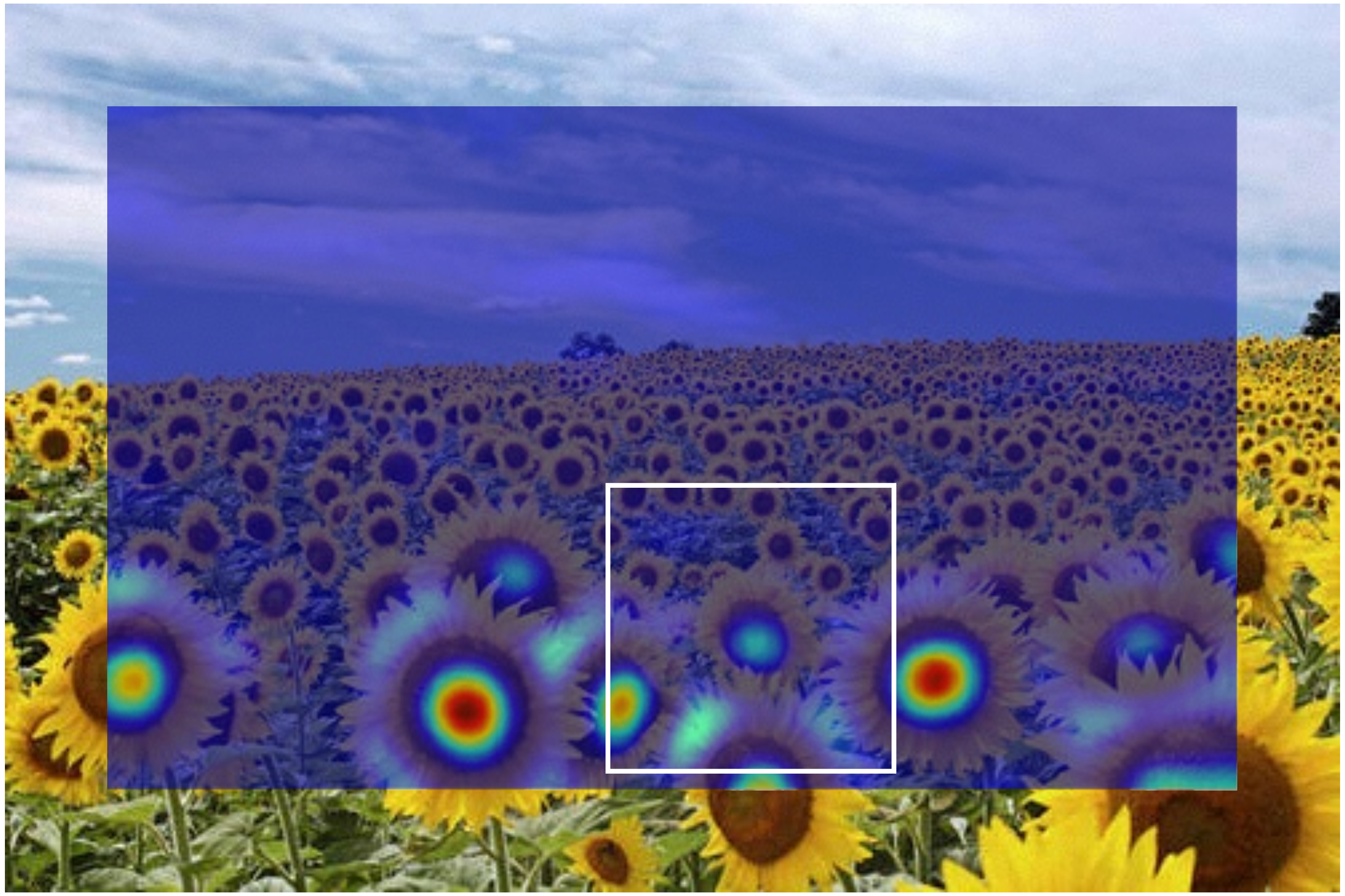

sigma=9.8

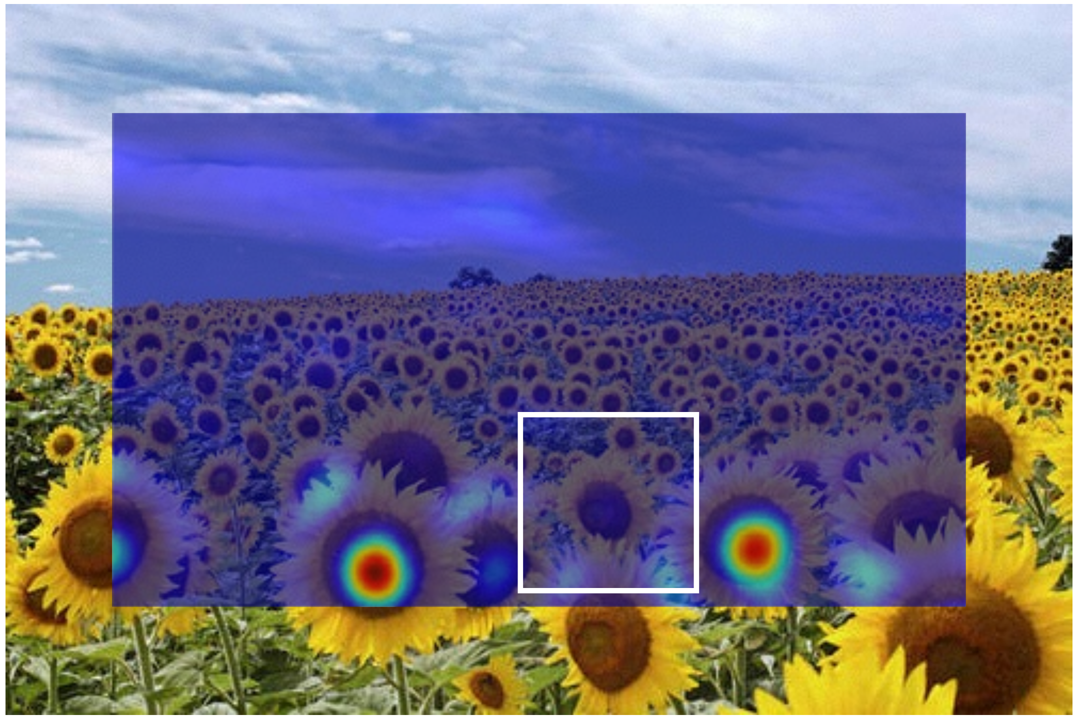


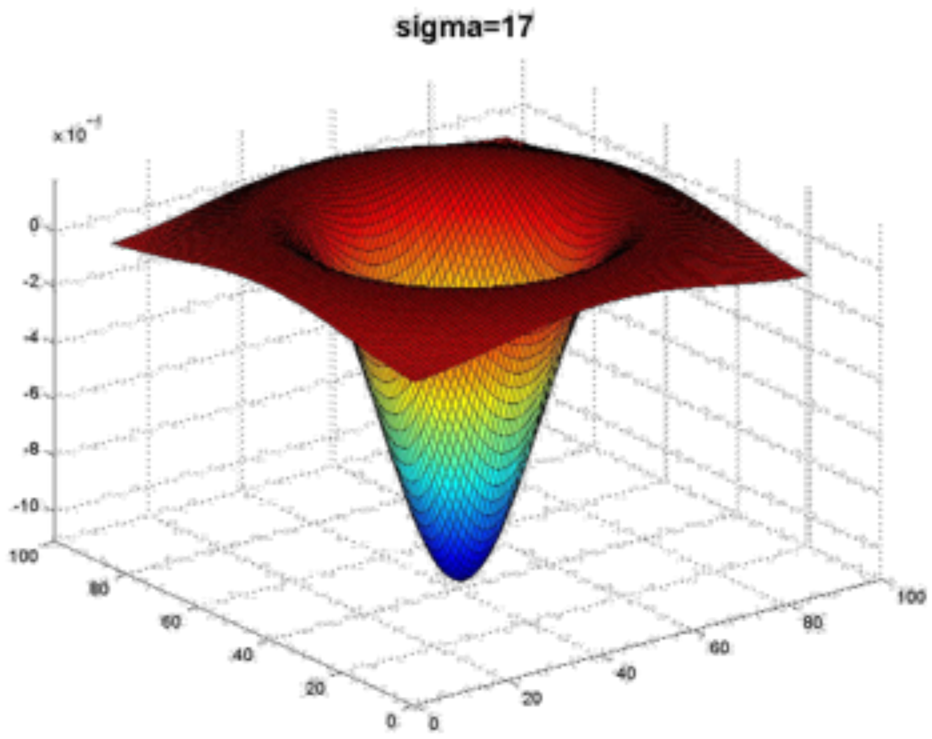




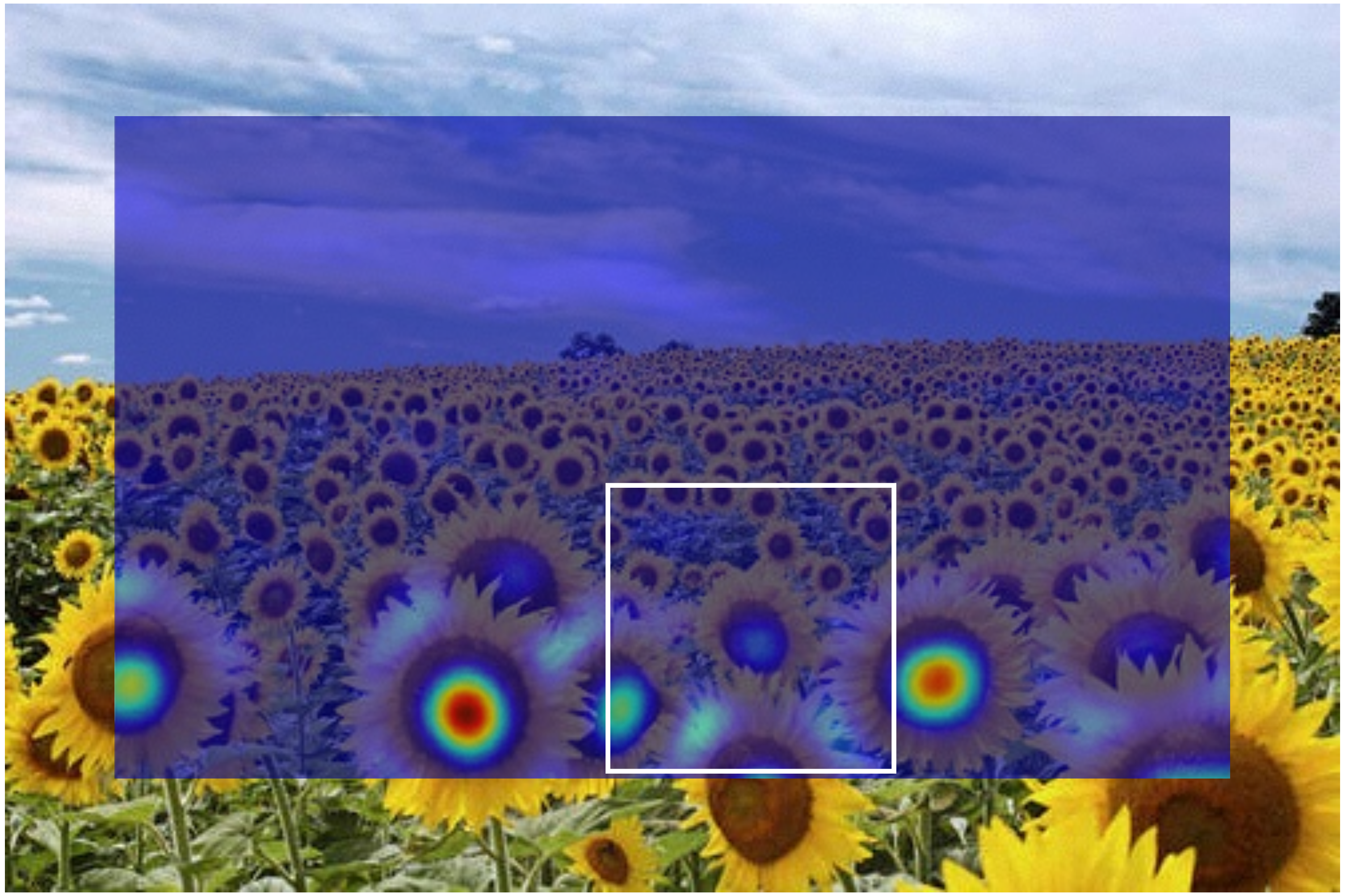
sigma=15.5
x 10^-4
2
0
-2
-4
-6
-8
-10
-12
100
80
60
40
20
0
0
20
40
60
80
100

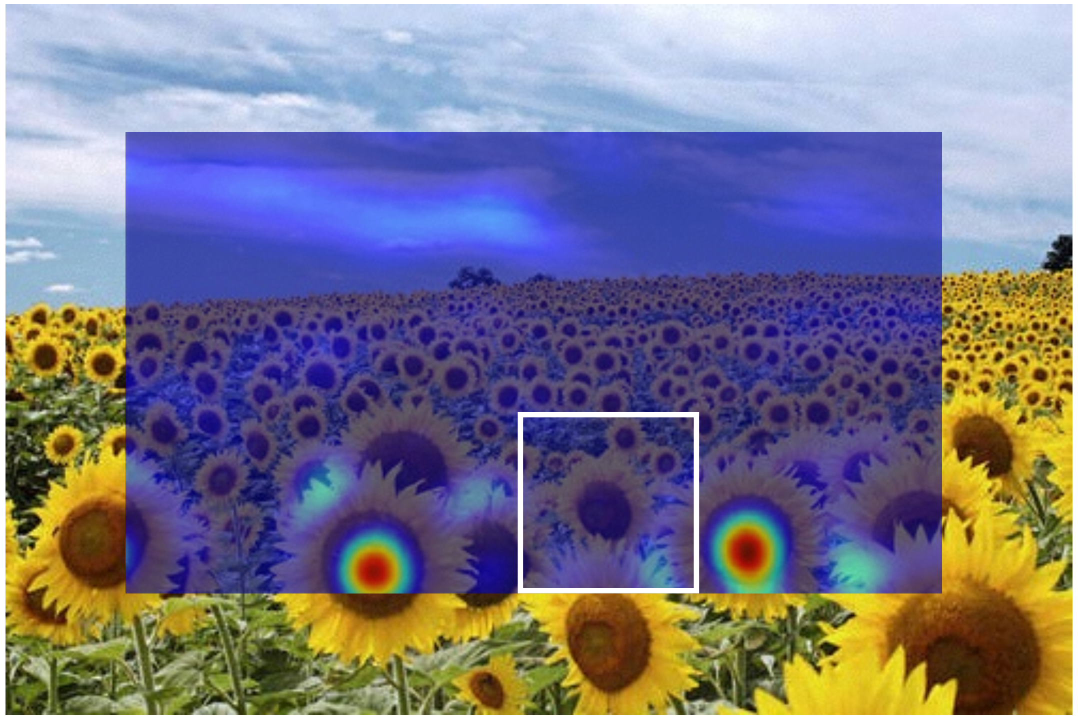






sigma=17

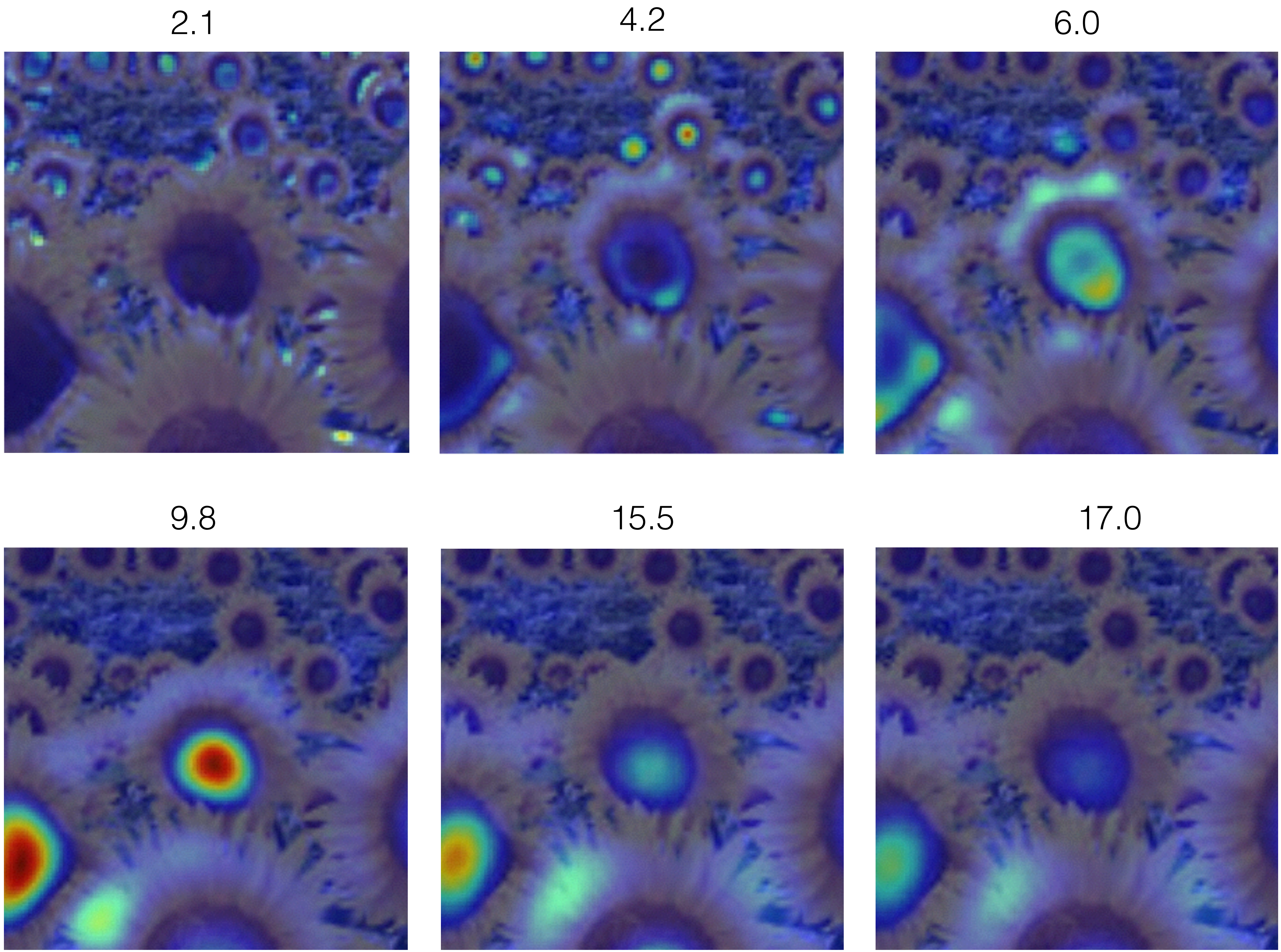


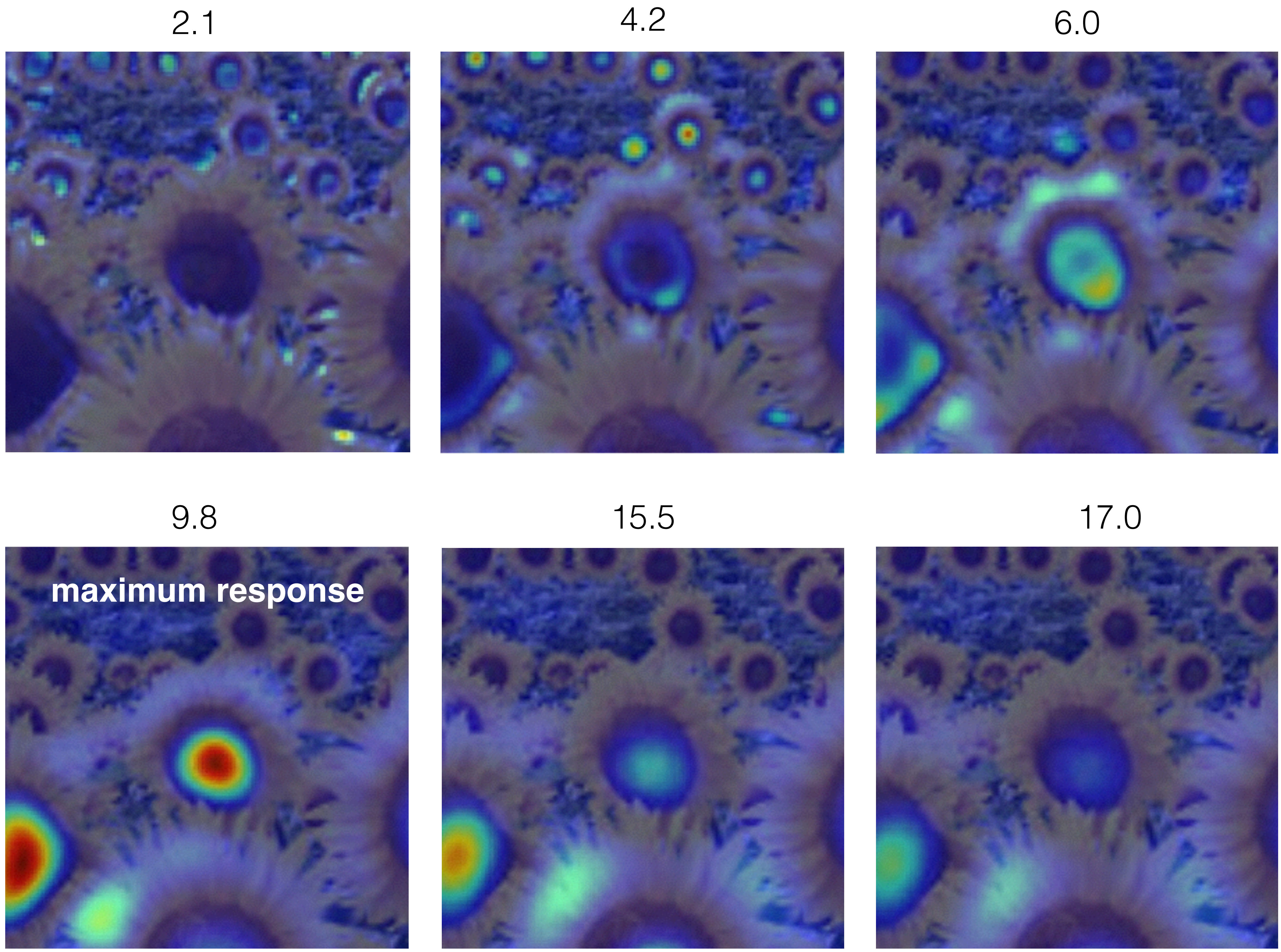


What happened when you applied different Laplacian filters?

Full size

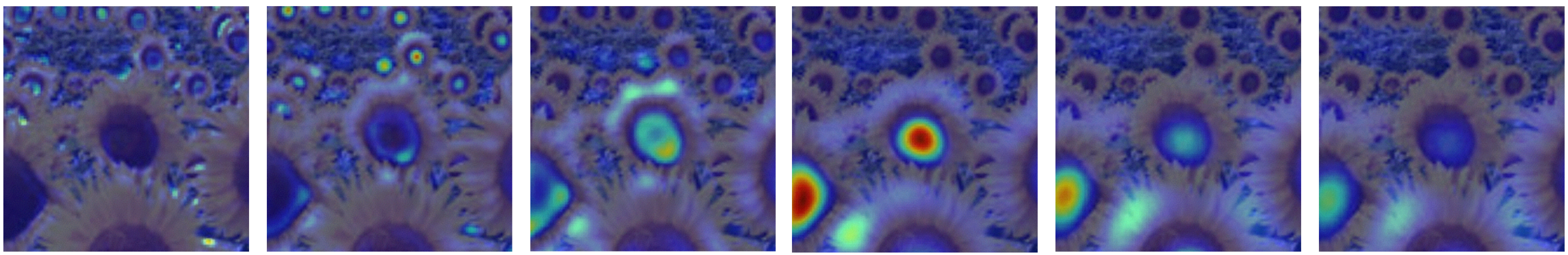
3/4 size


2.1
4.2
6.0
9.8
15.5
17.0


2.1
4.2
6.0
9.8
maximum response
15.5
17.0

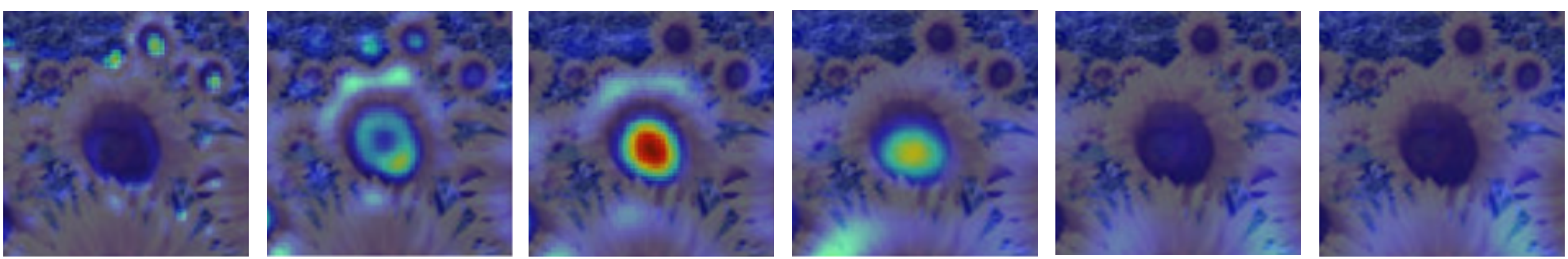
optimal scale

2.1 4.2 6.0 9.8 15.5 17.0



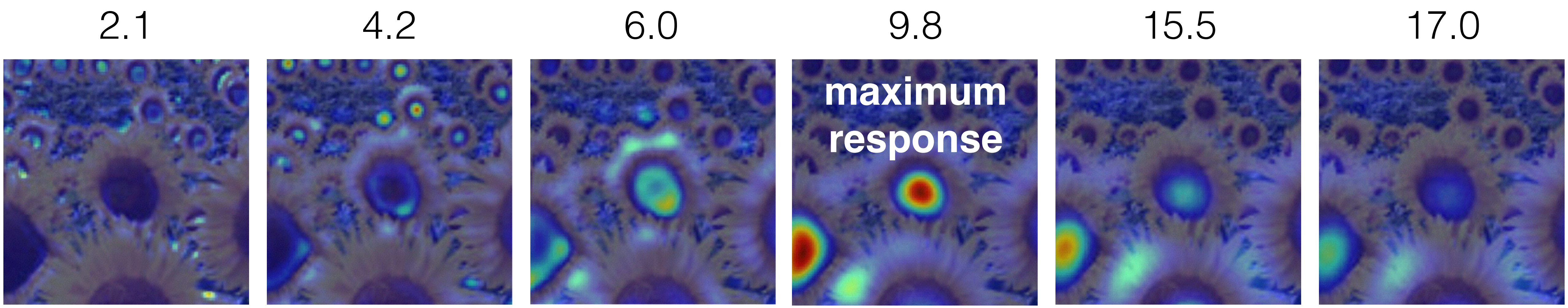
Full size image

2.1 4.2 6.0 9.8 15.5 17.0

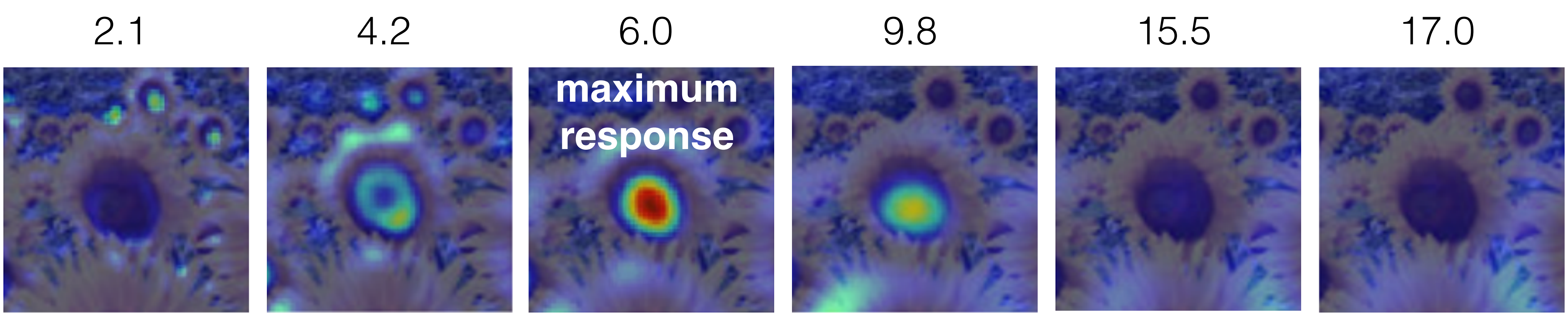


3/4 size image

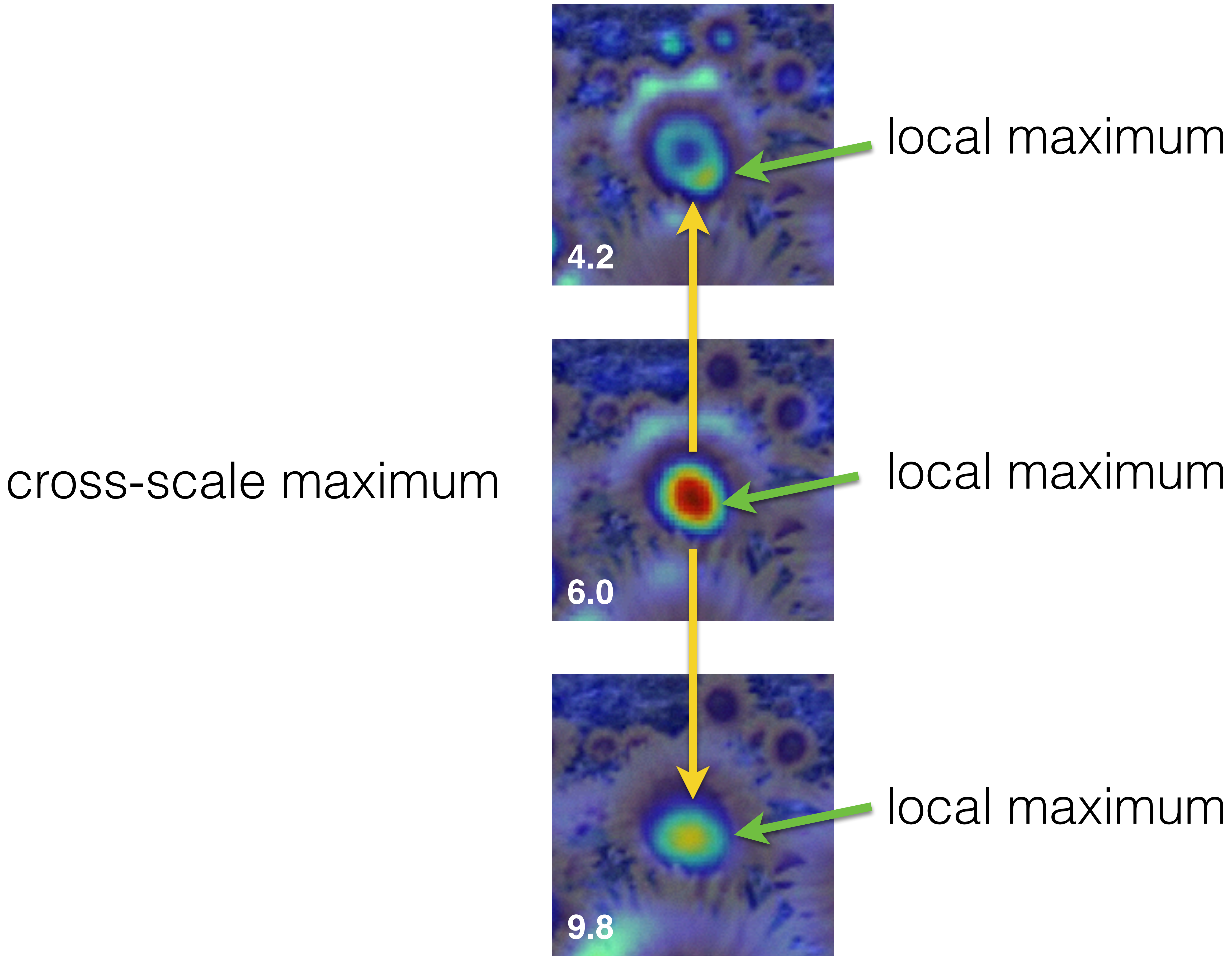
optimal scale



Full size image



3/4 size image


local maximum
4.2
cross-scale maximum
local maximum
6.0
local maximum
9.8

implementation

```
For each level of the Gaussian pyramid
      compute feature response (e.g. Harris, Laplacian)


For each level of the Gaussian pyramid
     if local maximum and cross-scale
     save scale and location of feature
```