

# Why detect corners?

Image alignment (homography, fundamental matrix)

3D reconstruction

Motion tracking

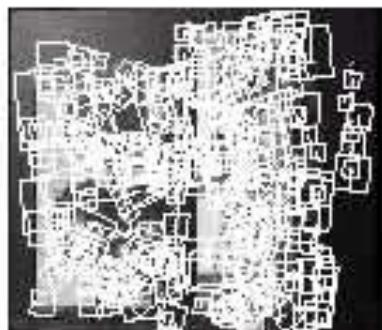
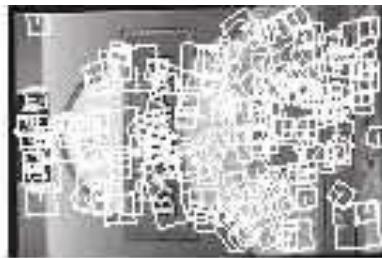
Object recognition

Indexing and database retrieval

Robot navigation

# Planar object instance recognition

Database of planar objects



Instance recognition



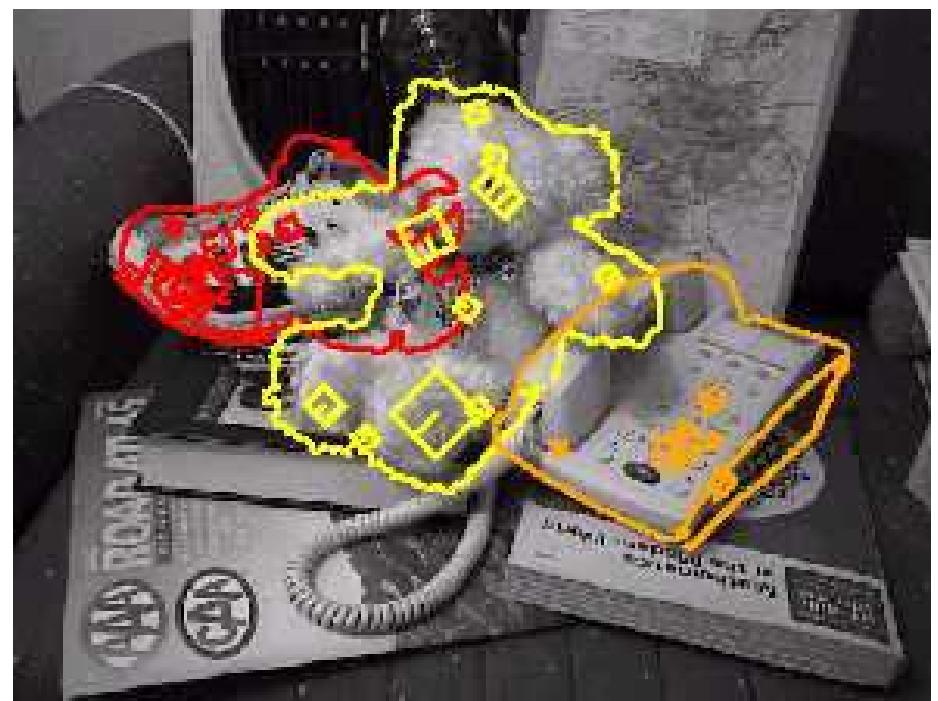
# 3D object recognition

Database of 3D objects



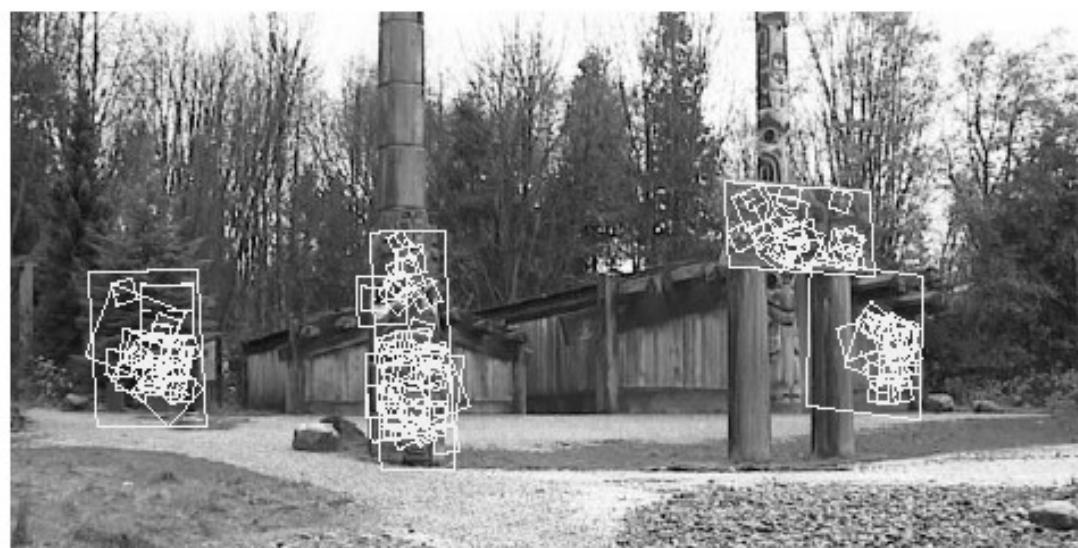
3D objects recognition



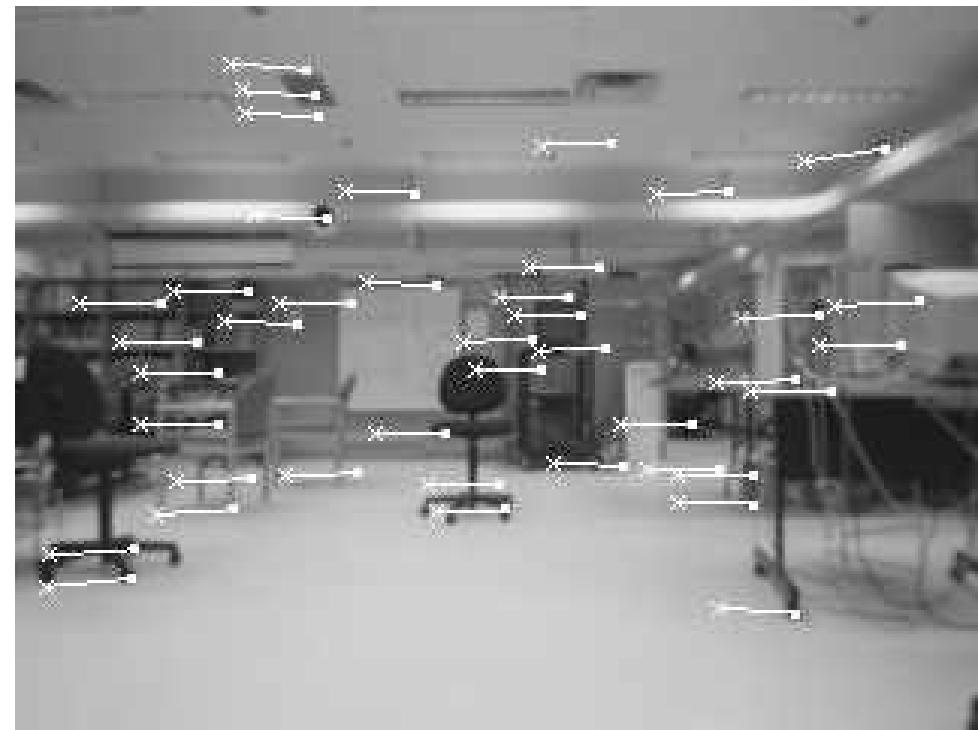
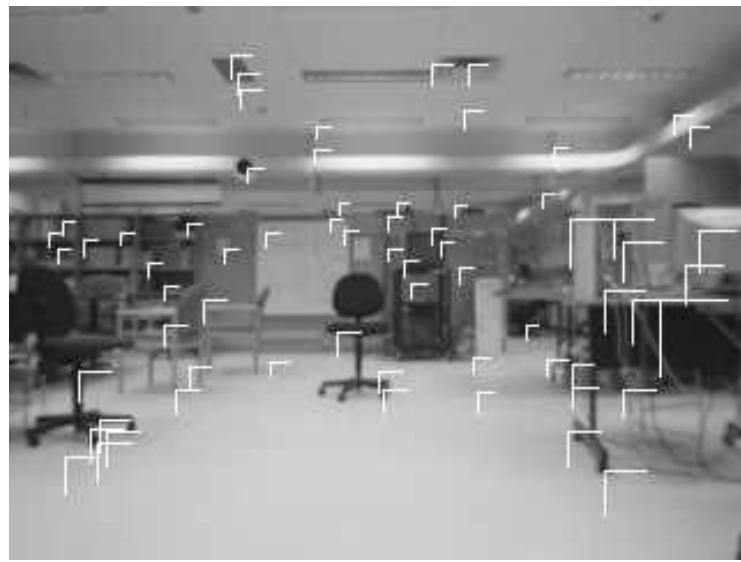
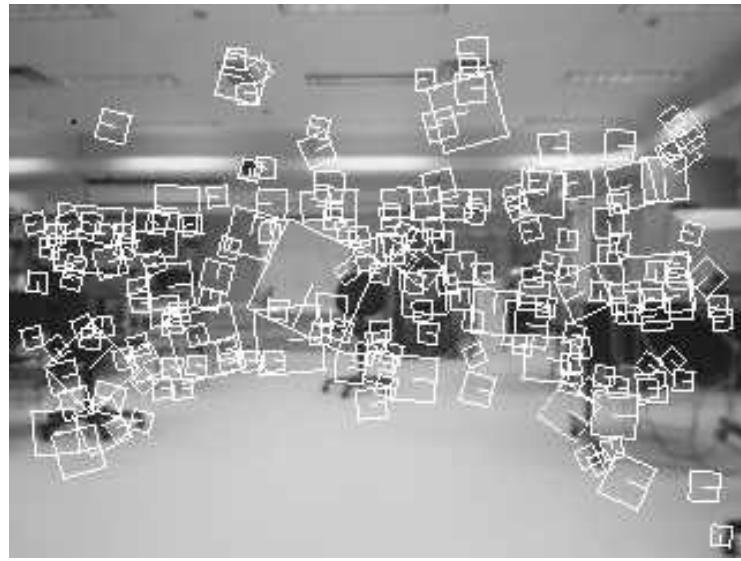


Recognition under occlusion

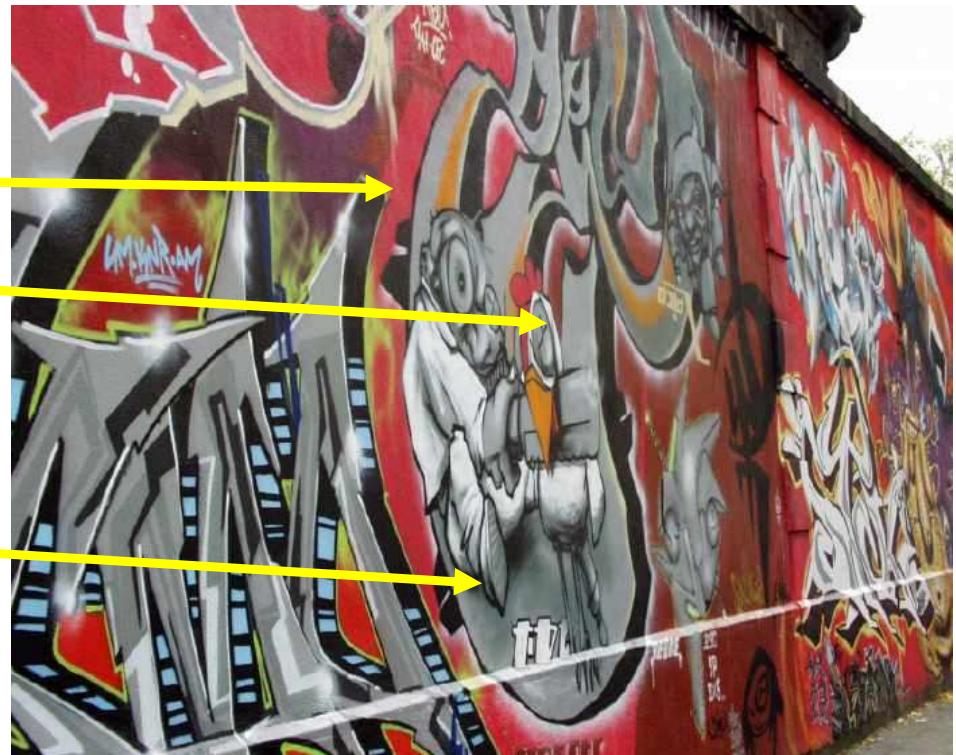
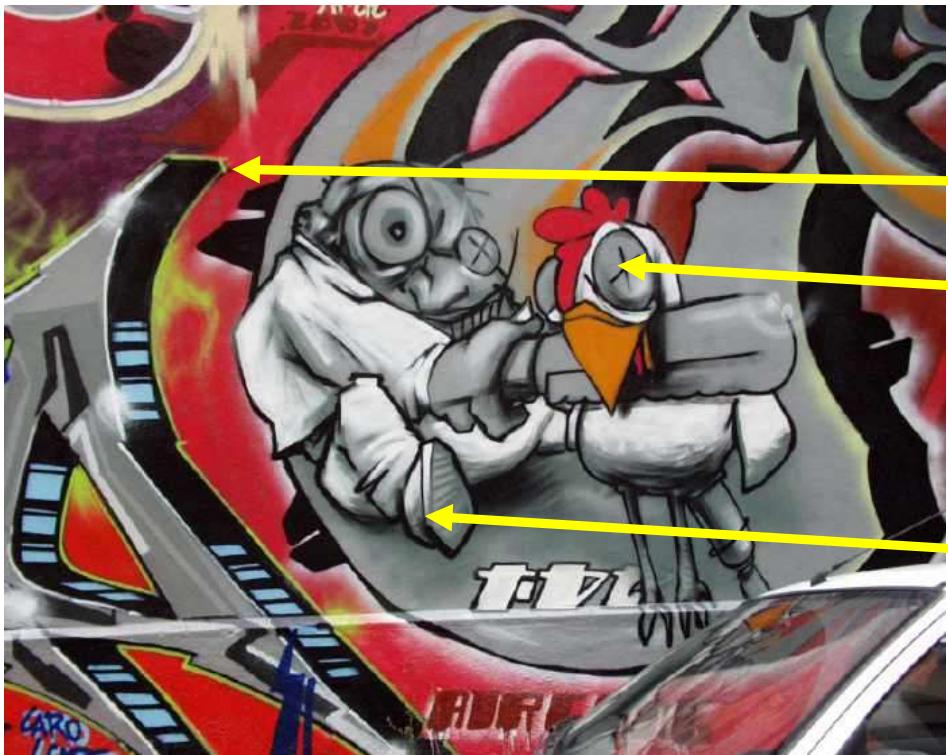
# Location Recognition

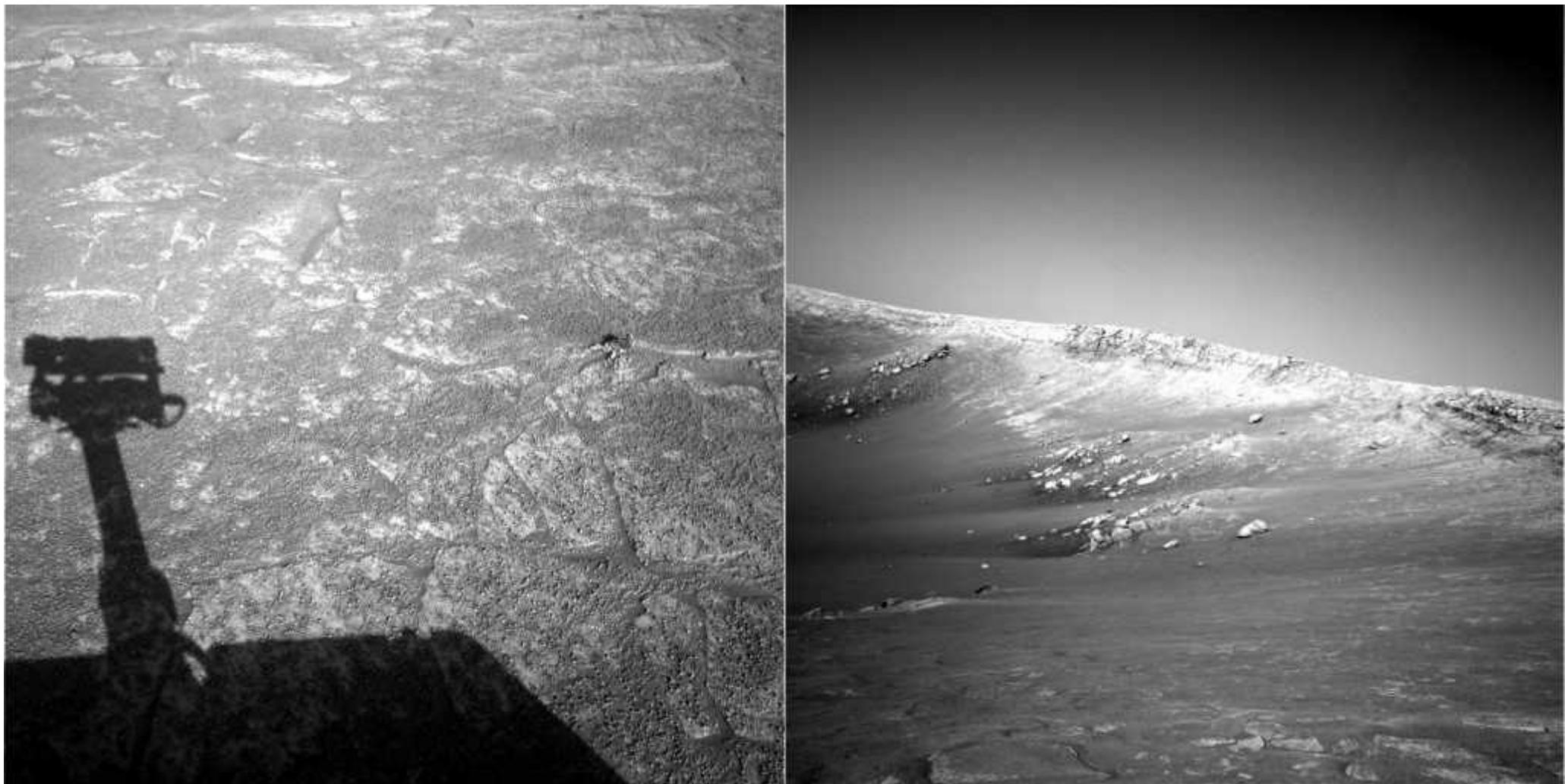


# Robot Localization



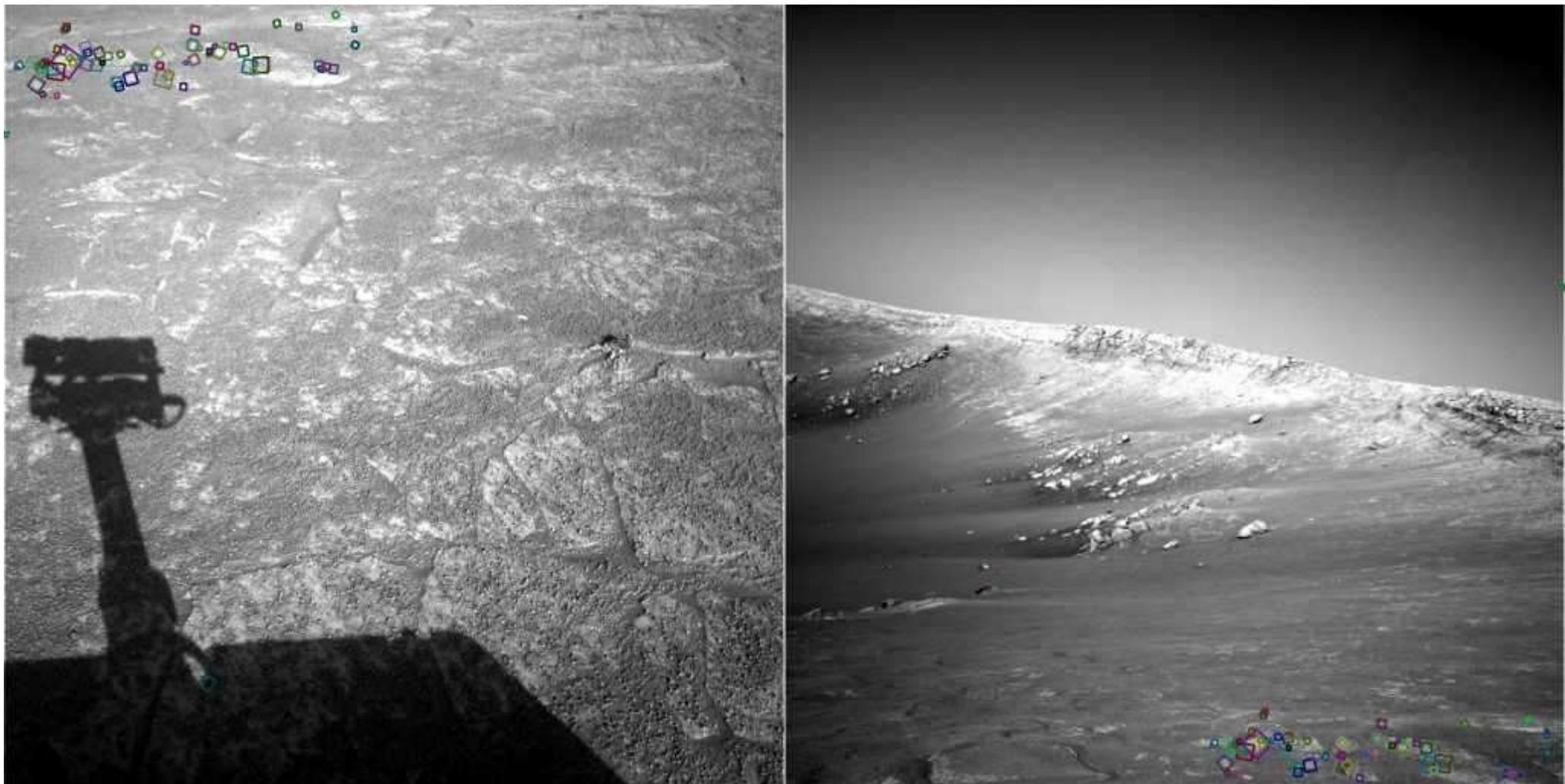
# Image matching



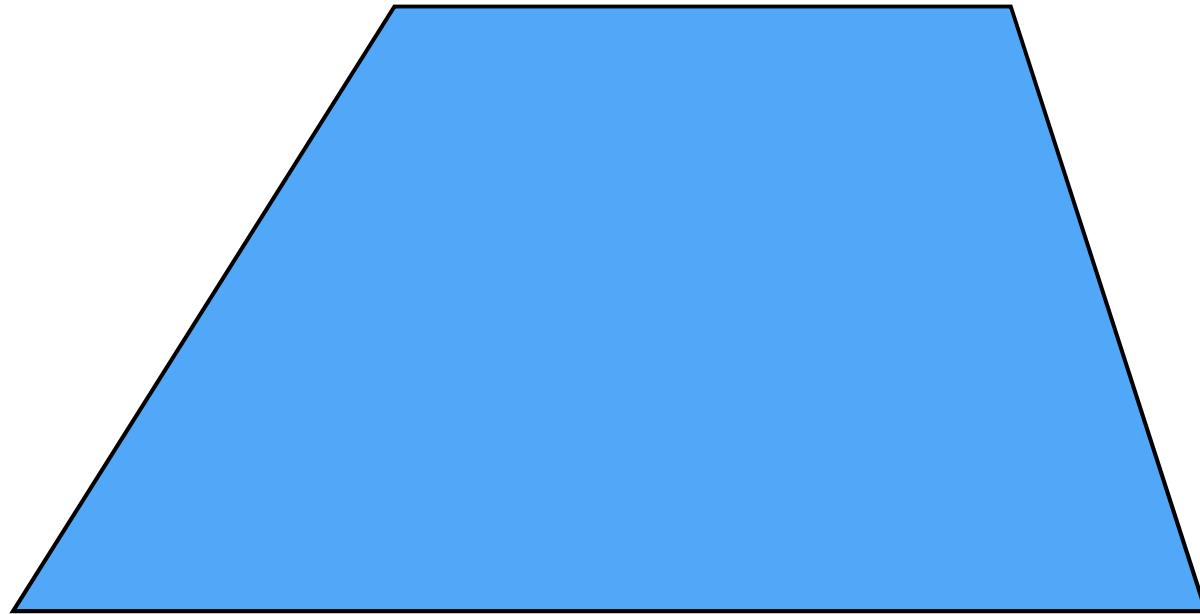


NASA Mars Rover images

*Where are the corresponding points?*

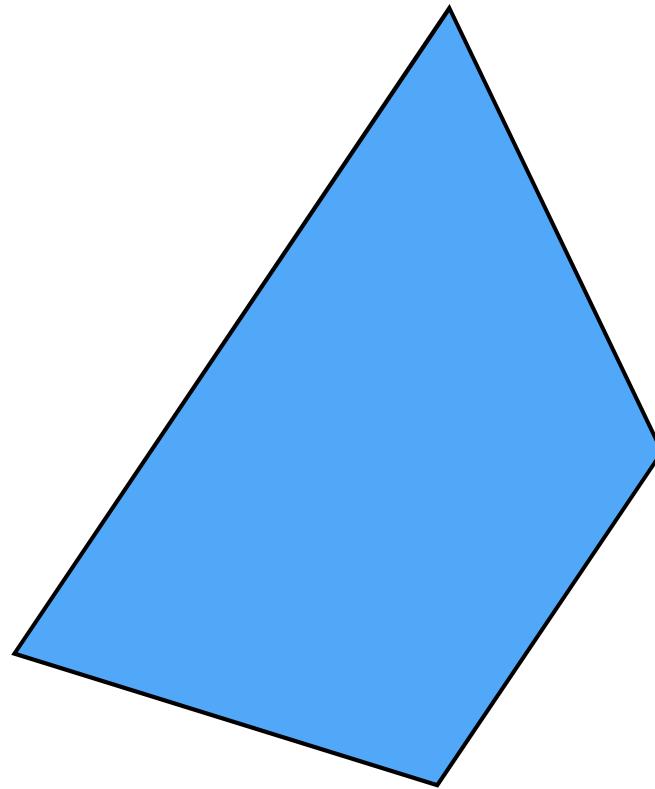


NASA Mars Rover images



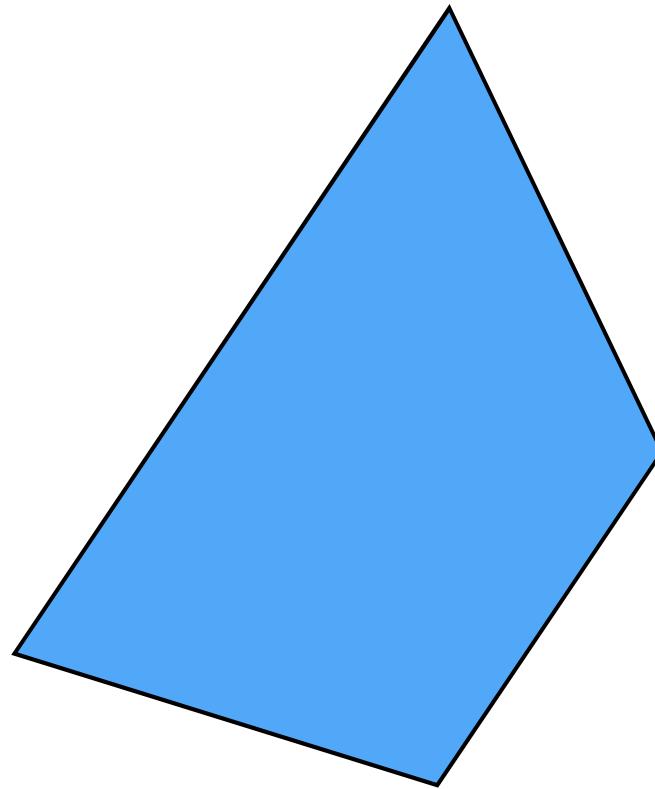
Pick a point in the image.  
Find it again in the next image.

*What type of feature would you select?*



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

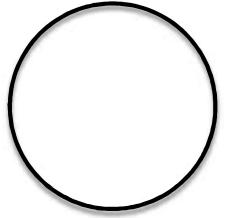
*What type of feature would you select?*



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

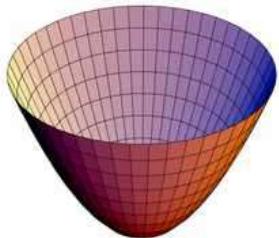
*What type of feature would you select?*  
a corner

# Visualizing quadratics



Equation of a circle

$$1 = x^2 + y^2$$



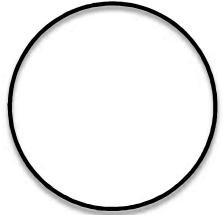
Equation of a ‘bowl’ (paraboloid)

$$f(x, y) = x^2 + y^2$$

*If you slice the bowl at*

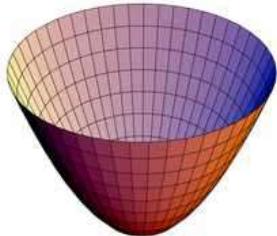
$$f(x, y) = 1$$

*what do you get?*



Equation of a circle

$$1 = x^2 + y^2$$



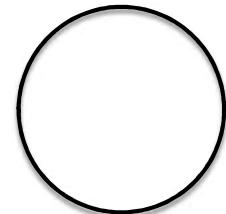
Equation of a ‘bowl’ (paraboloid)

$$f(x, y) = x^2 + y^2$$

*If you slice the bowl at*

$$f(x, y) = 1$$

*what do you get?*



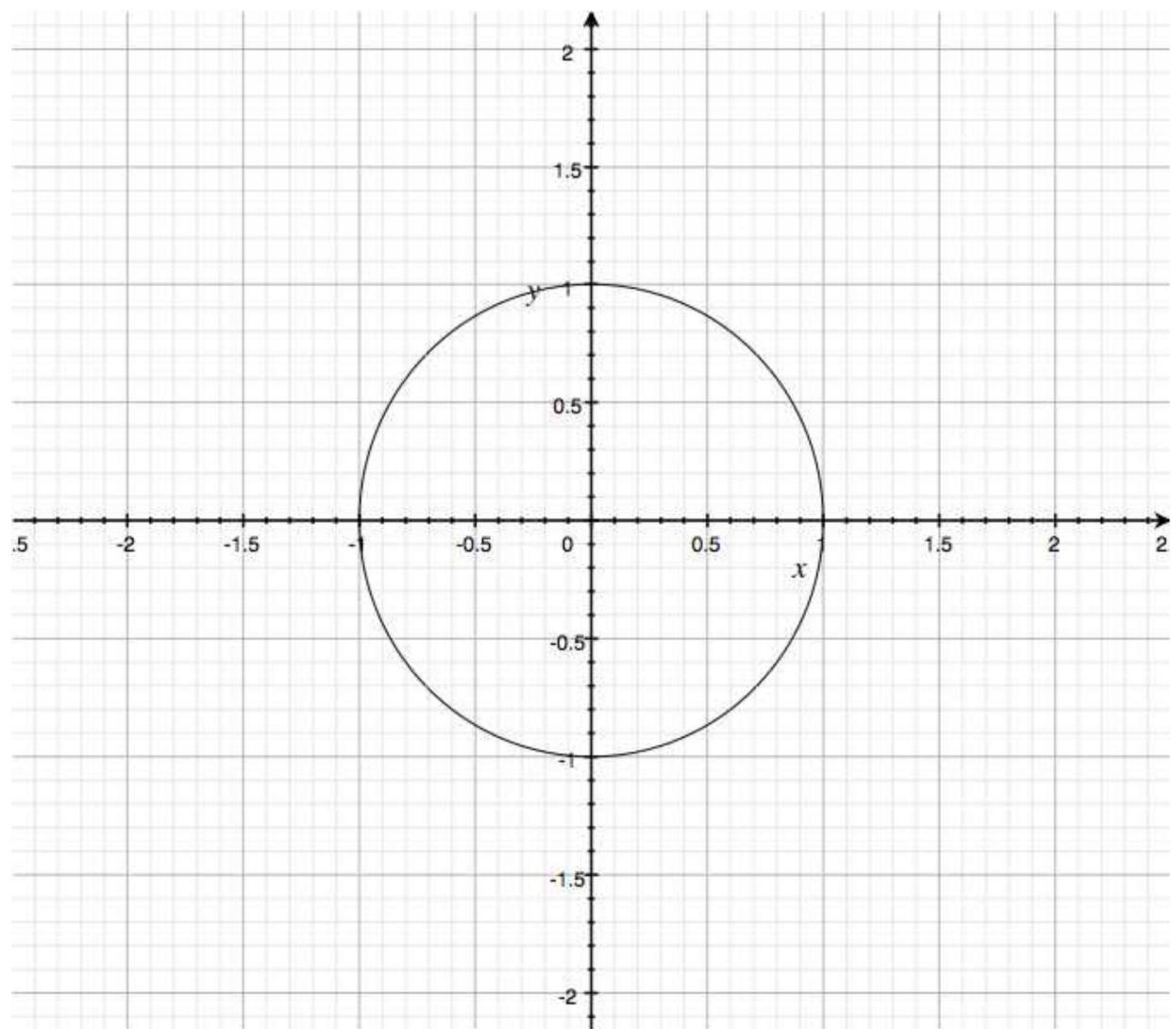
$$f(x, y) = x^2 + y^2$$

can be written in matrix form like this...

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

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

'sliced at 1'



*What happens if you **increase**  
coefficient on **x**?*

$$f(x, y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

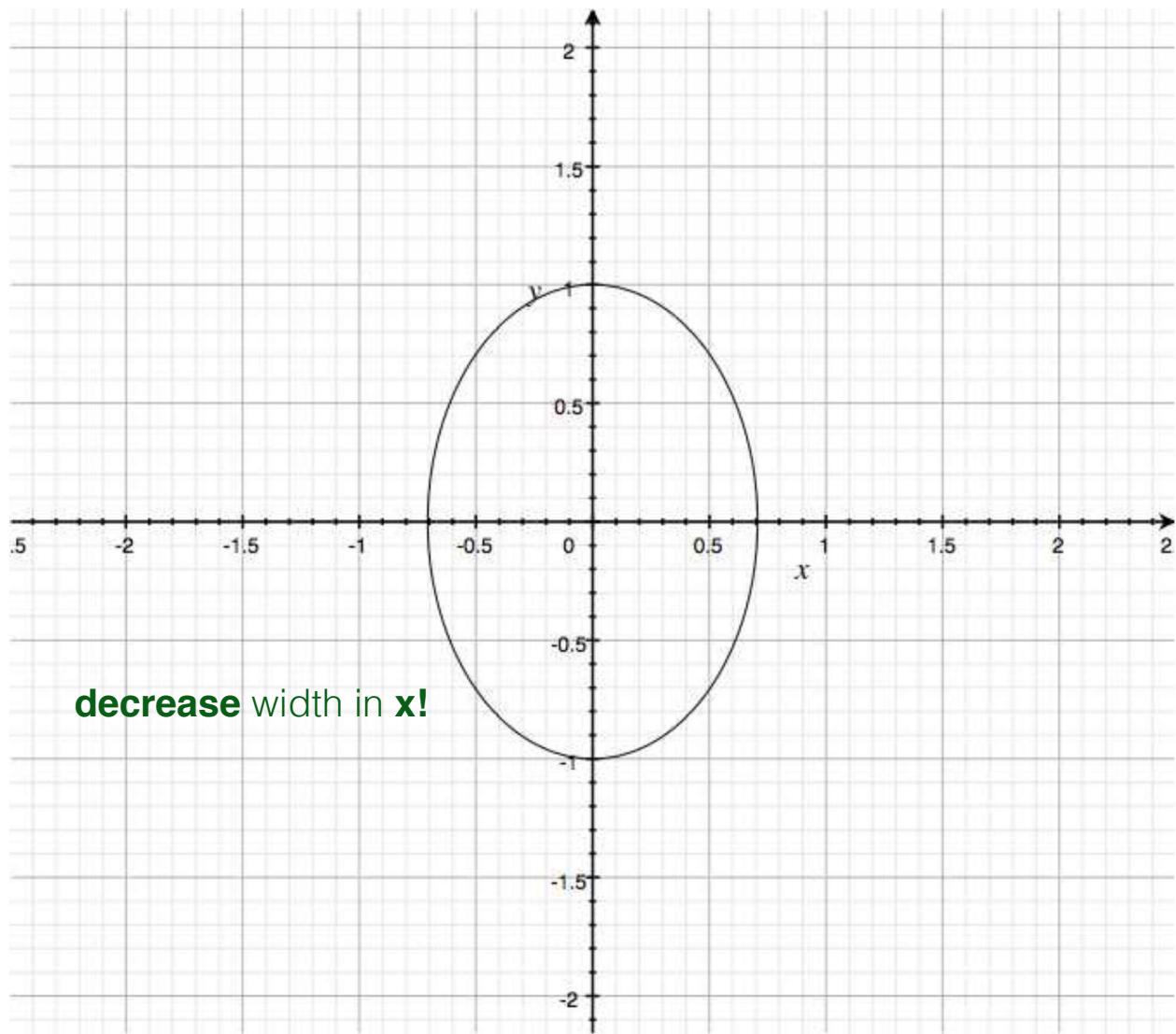
and slice at 1

*What happens if you **increase** coefficient on **x**?*

$$f(x, y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

and slice at 1

**decrease width in x!**



*What happens if you **increase**  
coefficient on **y**?*

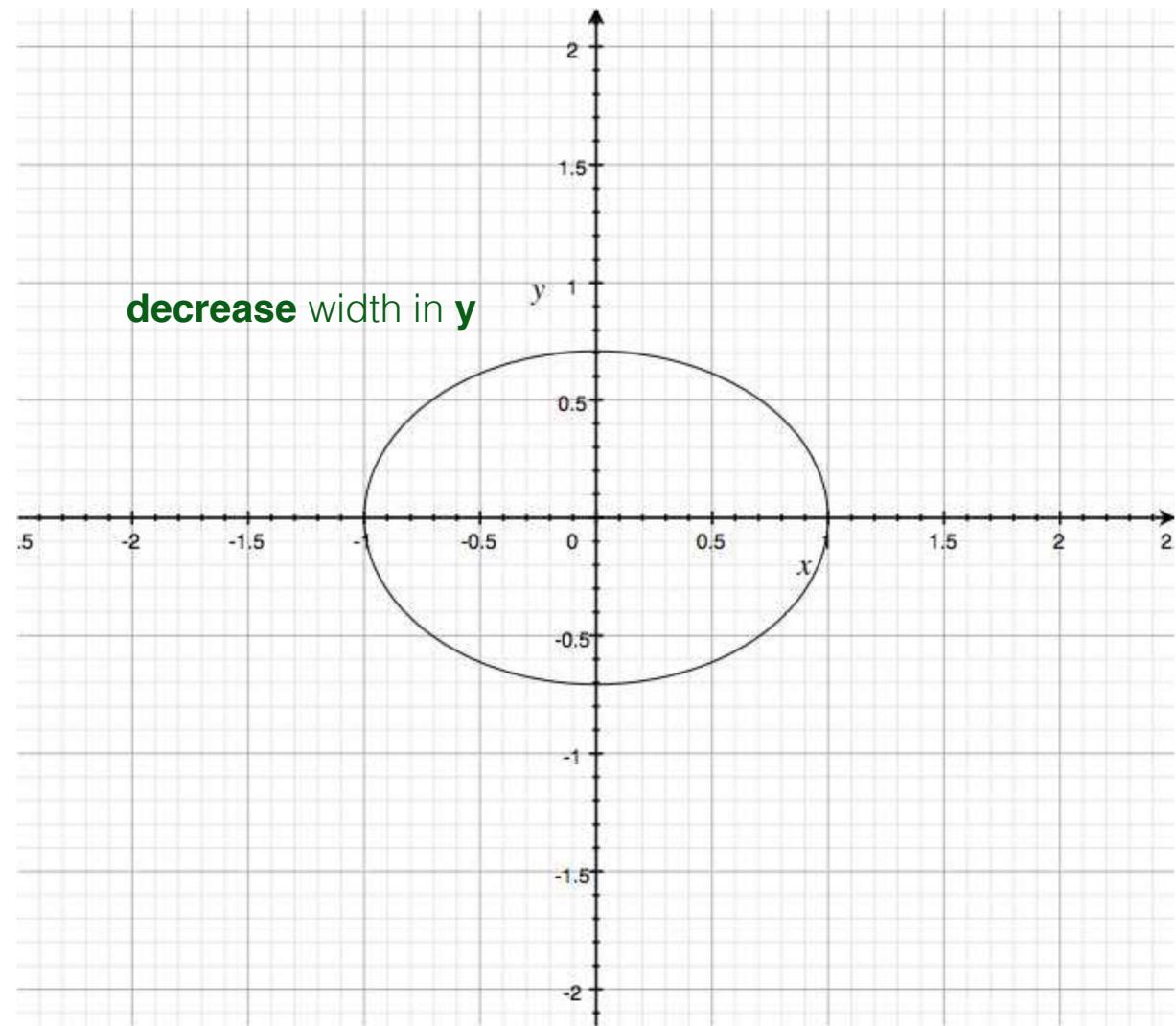
$$f(x, y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

and slice at 1

*What happens if you **increase** coefficient on **y**?*

$$f(x, y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

and slice at 1



$$f(x, y) = x^2 + y^2$$

can be written in matrix form like this...

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

*What's the shape?*

*What are the eigenvectors?*

*What are the eigenvalues?*

$$f(x, y) = x^2 + y^2$$

can be written in matrix form like this...

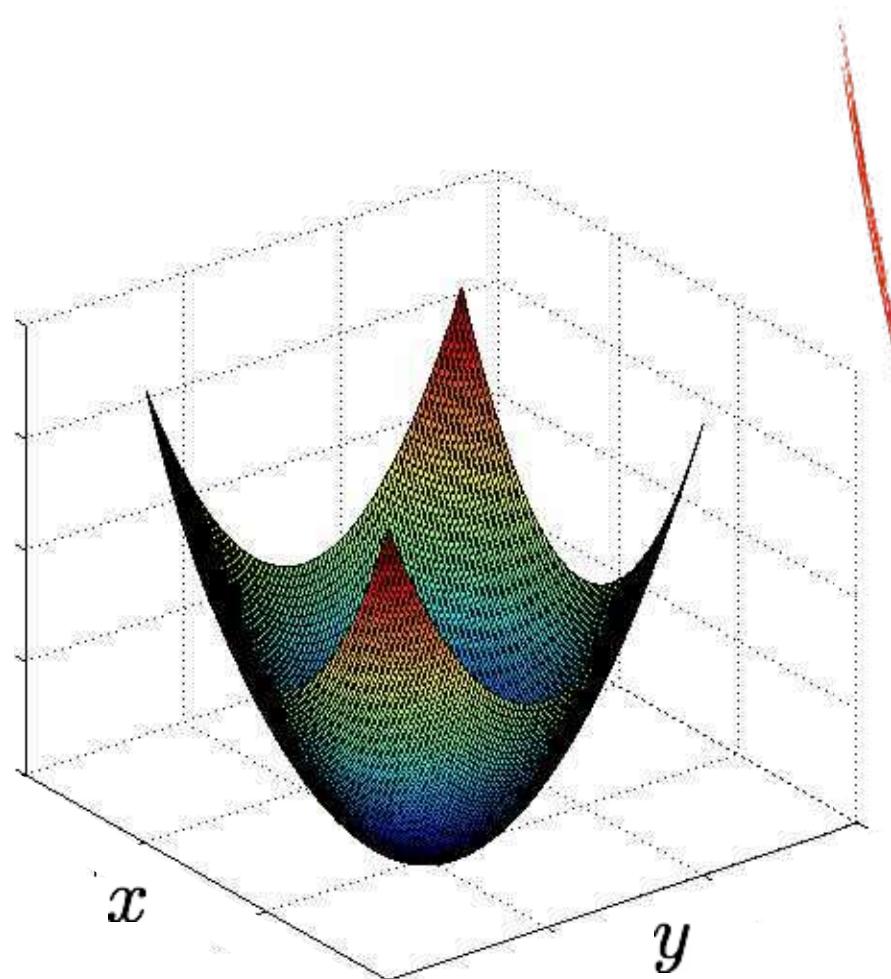
$$f(x, y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

### Result of Singular Value Decomposition (SVD)

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \text{eigenvectors} \\ \boxed{1} & \boxed{0} \\ \boxed{0} & \boxed{1} \end{bmatrix} \begin{bmatrix} \text{eigenvalues along diagonal} \\ \boxed{1} & 0 \\ 0 & \boxed{1} \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^\top$$

axis of the  
'ellipse slice'

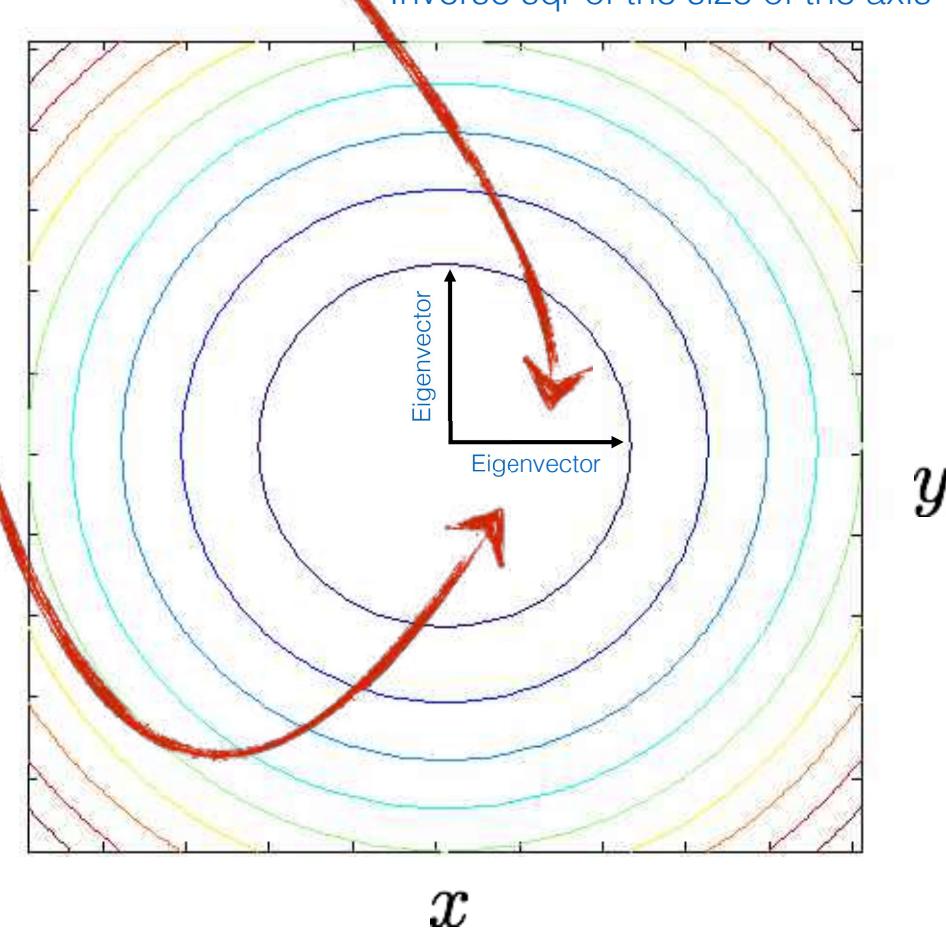
Inverse sqrt of  
length of the  
quadratic along  
the axis



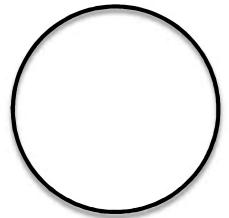
$$\mathbf{A} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T$$

Eigenvectors   Eigenvalues

Eigenvectors

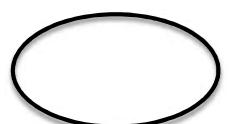


Recall:



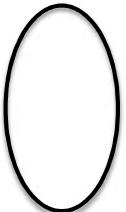
$$f(x, y) = \begin{bmatrix} x & y \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

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



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

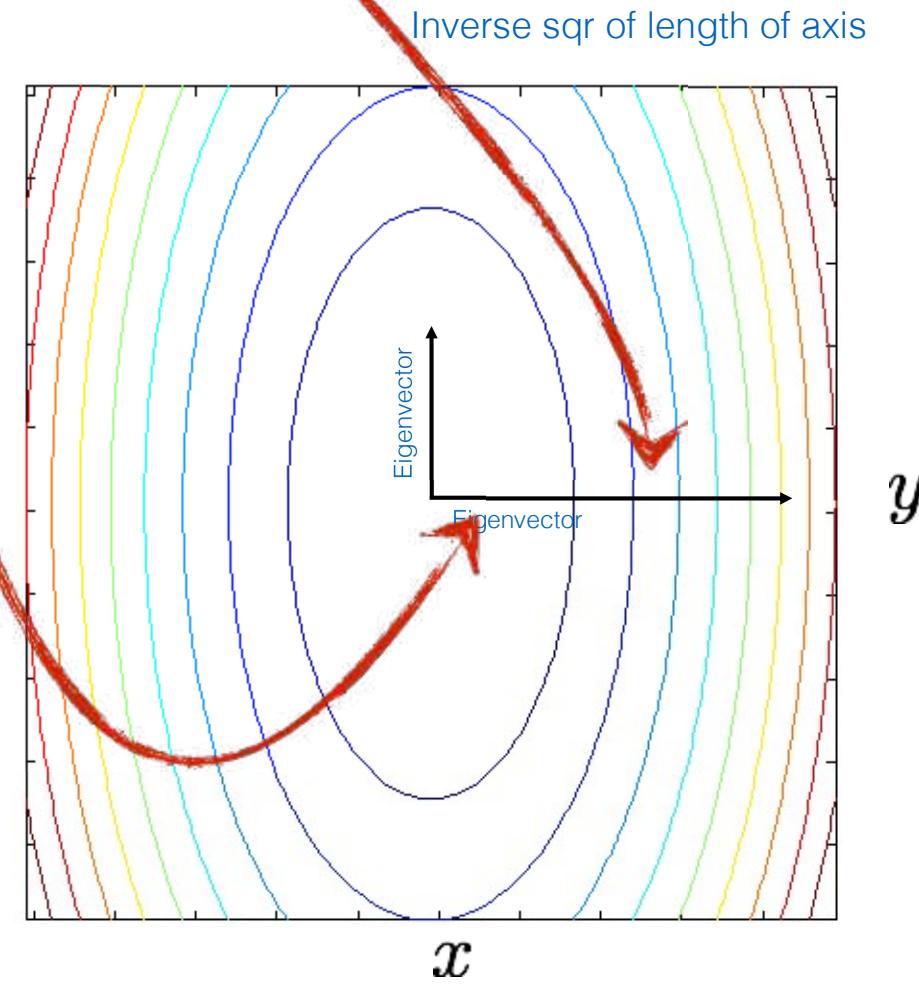
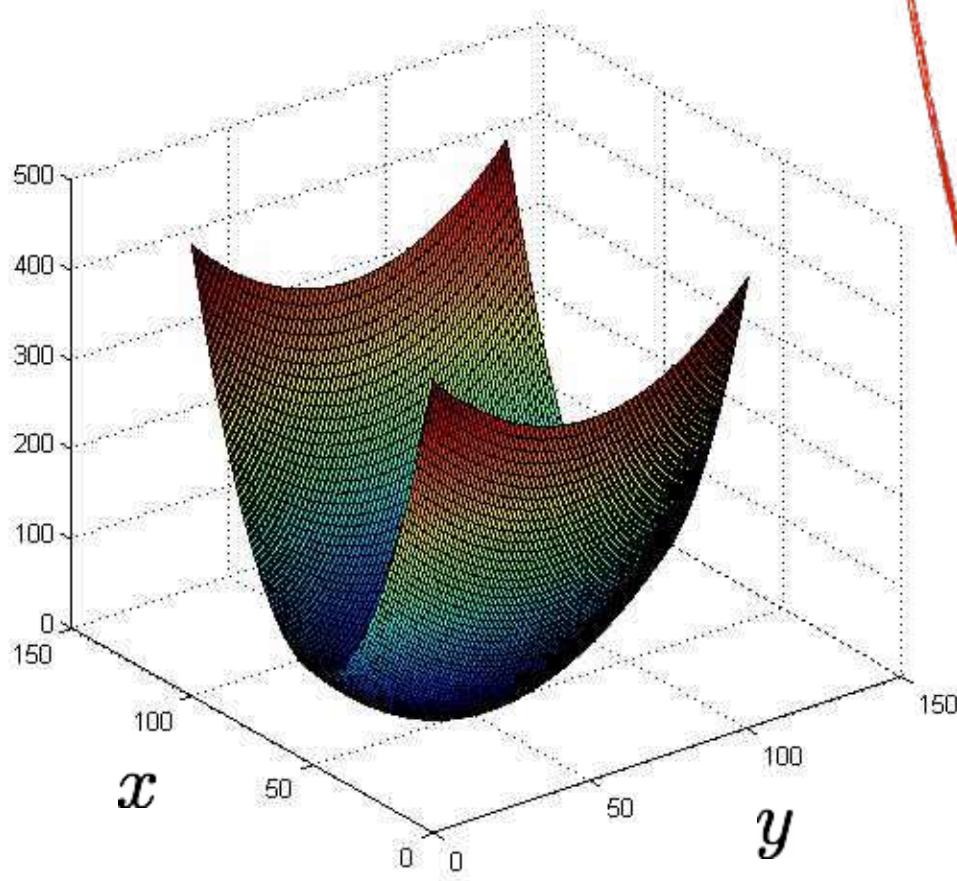
you can smash this bowl in the **x** direction



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

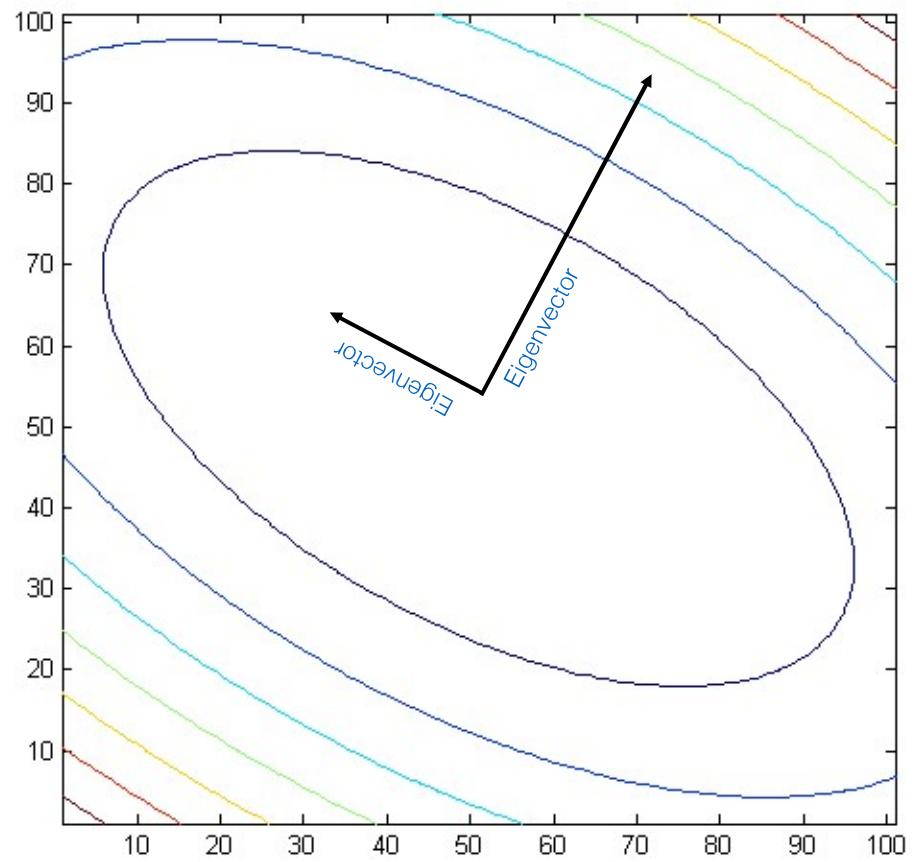
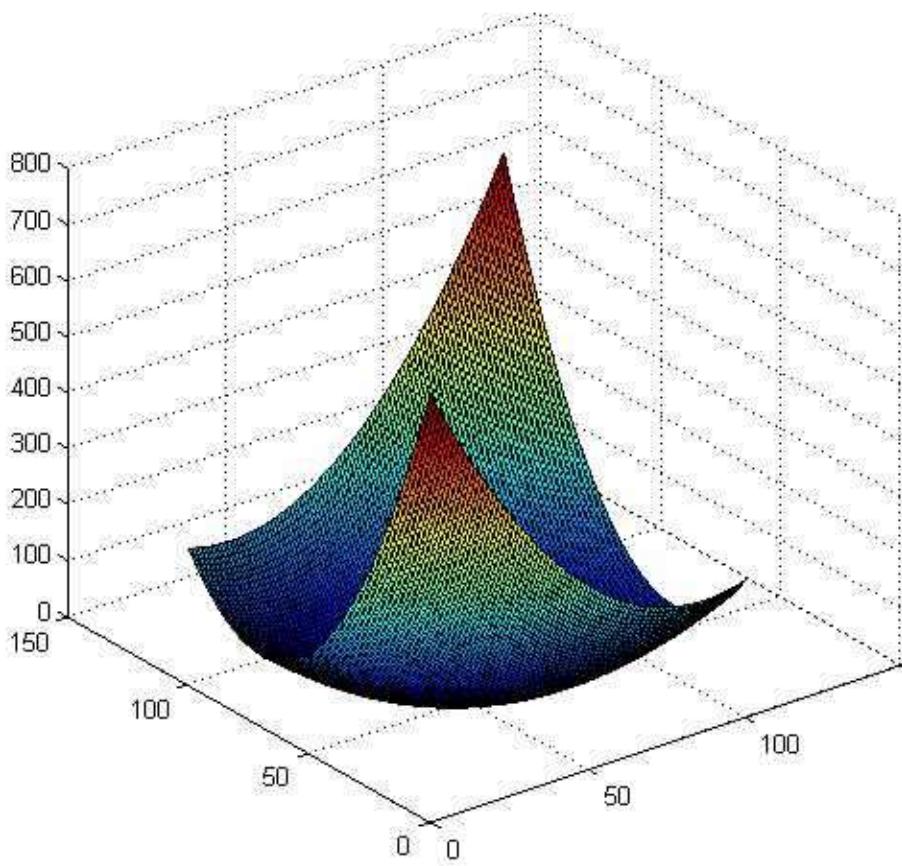
$$A = \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}^T$$

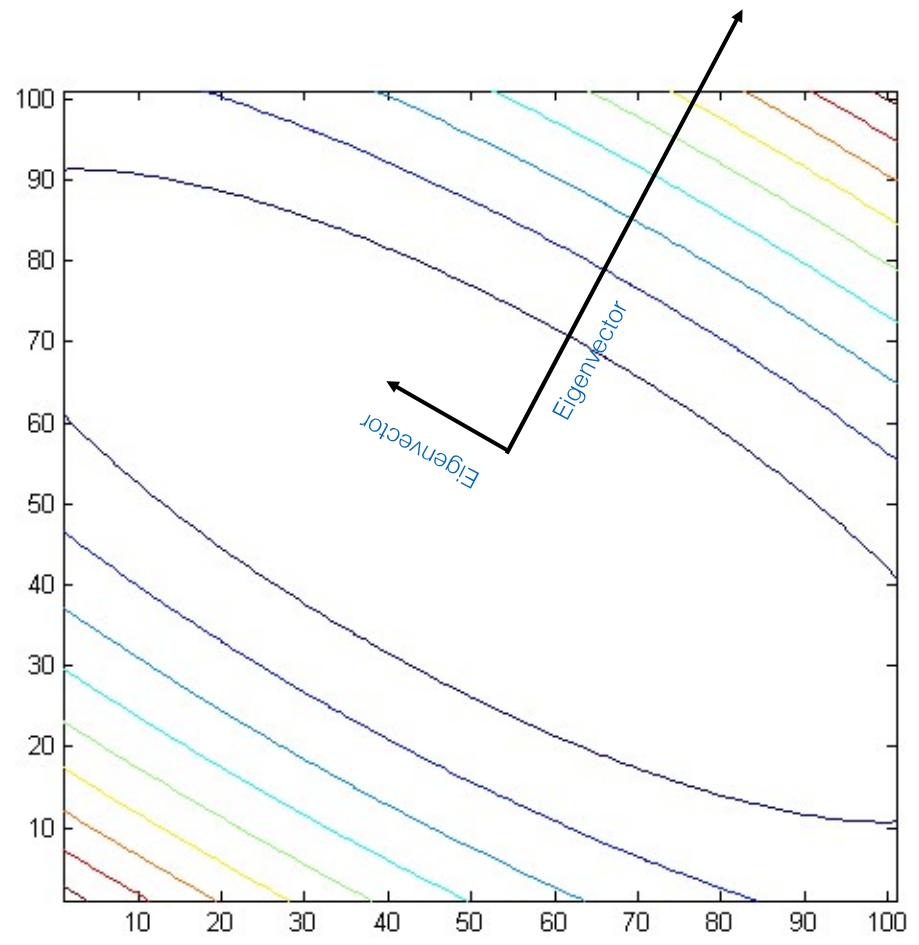
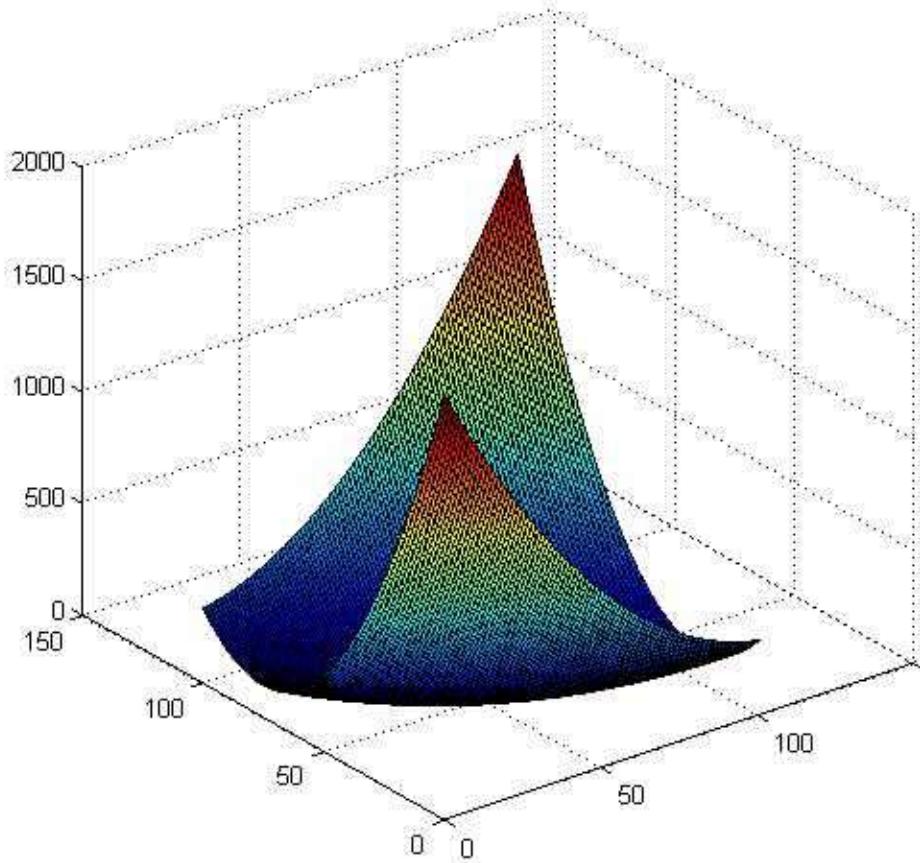
Eigenvalues  
Eigenvectors  
Eigenvectors



$$A = \begin{bmatrix} 3.25 & 1.30 \\ 1.30 & 1.75 \end{bmatrix} = \begin{bmatrix} 0.50 & -0.87 \\ -0.87 & -0.50 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} 0.50 & -0.87 \\ -0.87 & -0.50 \end{bmatrix}^T$$

Eigenvalues  
Eigenvectors      Eigenvalues  
Eigenvectors





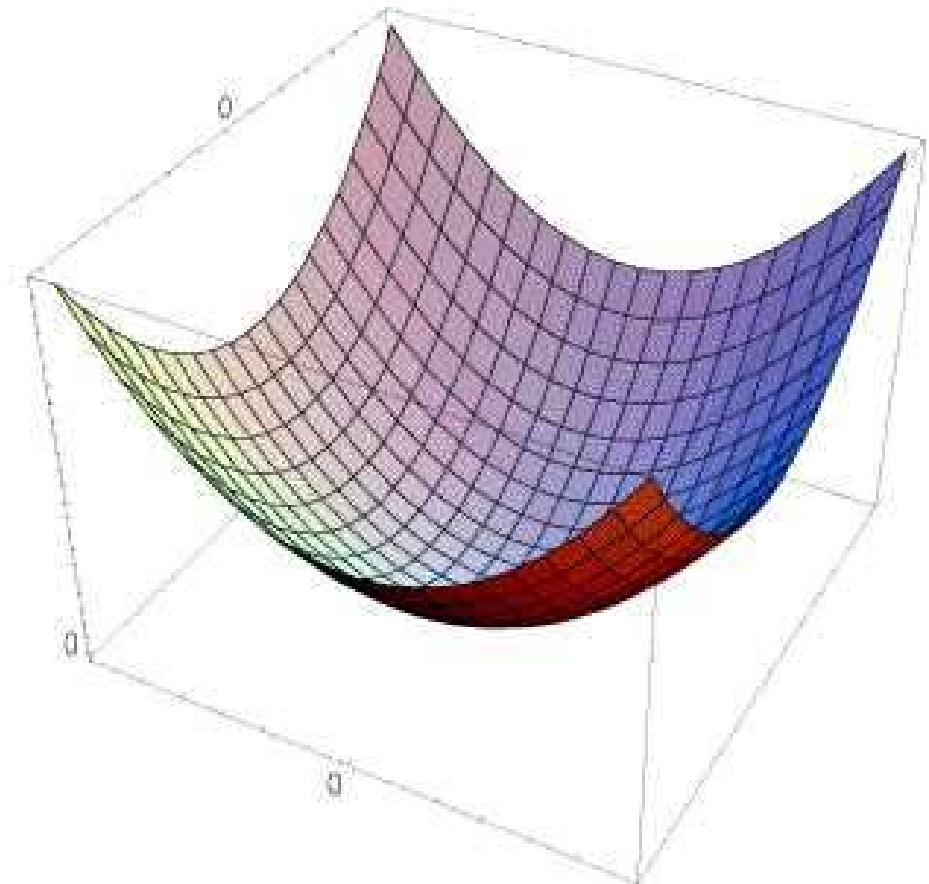
We will need this to understand the...

# Error function for Harris Corners

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

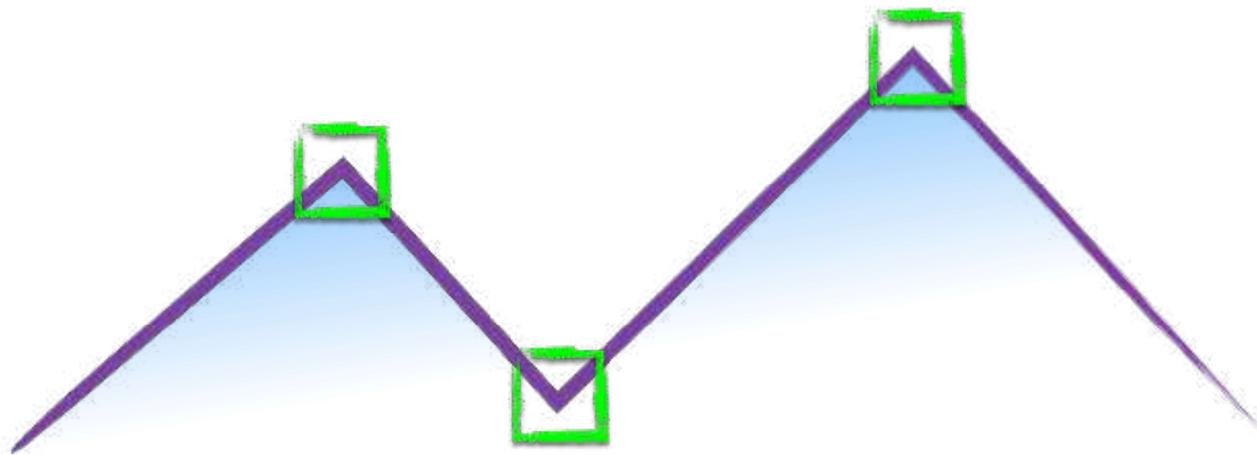
$$E(u,v) \approx [u \ v] M \begin{bmatrix} u \\ v \end{bmatrix}$$

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



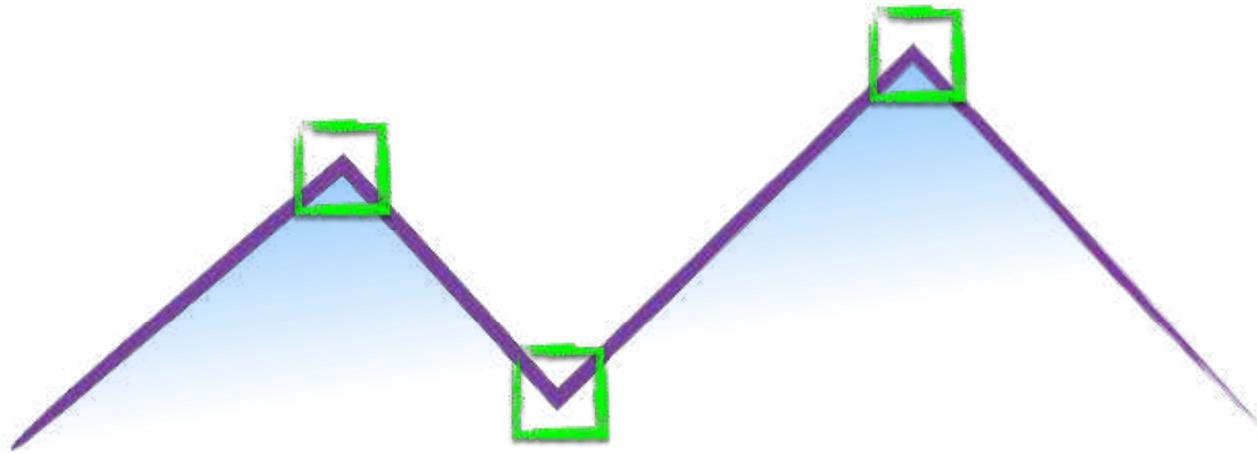
# Harris corner detector

# How do you find a corner?



# How do you find a corner?

[Moravec 1980]

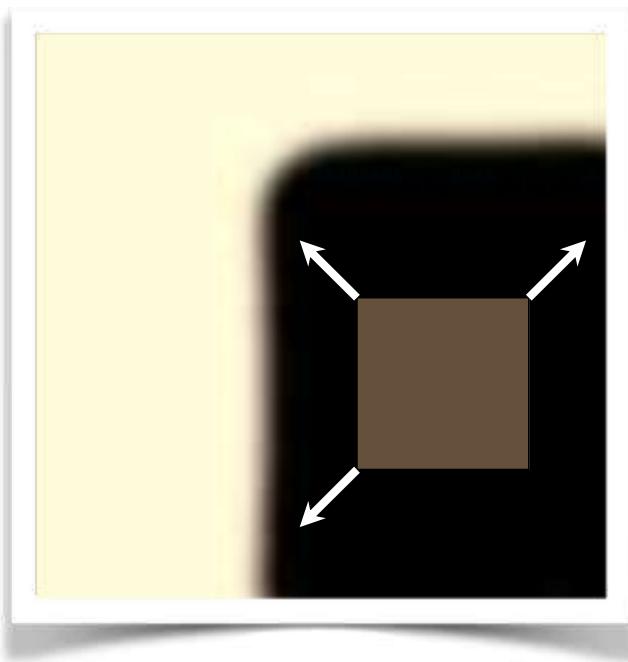


Easily recognized by looking through a small window

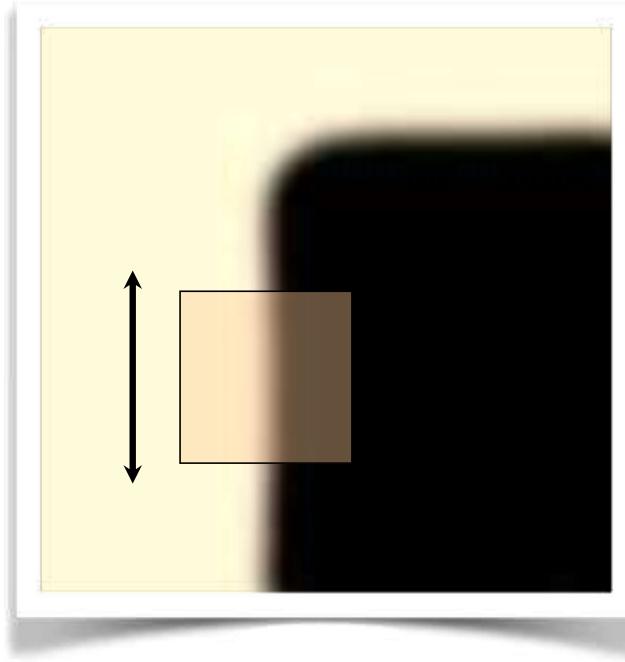
Shifting the window should give large change in intensity

Easily recognized by looking through a small window

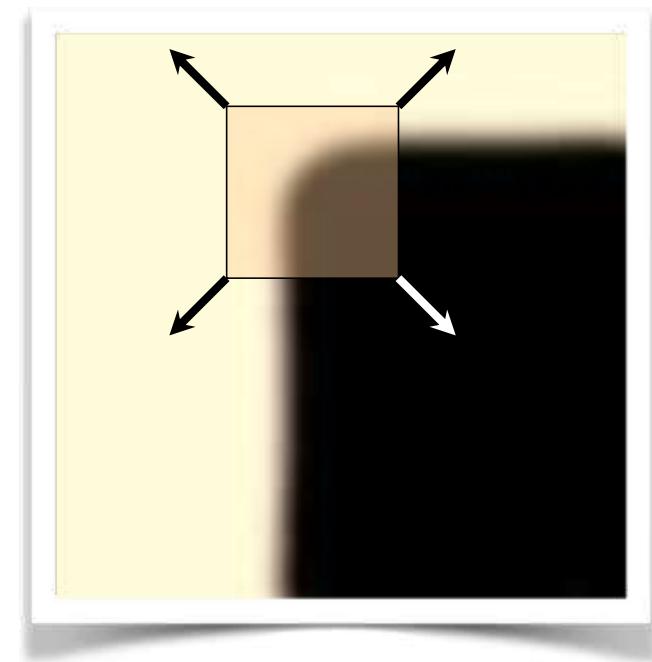
Shifting the window should give large change in intensity



“flat” region:  
no change in all  
directions



“edge”:  
no change along the edge  
direction



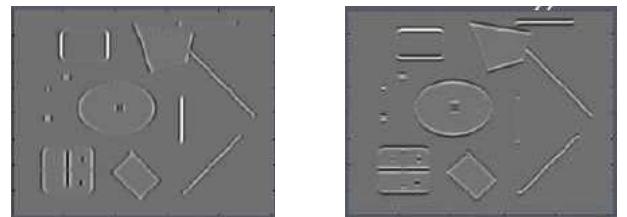
“corner”:  
significant change in all  
directions

Design a program to detect corners  
(hint: use image gradients)

# Finding corners

## (a.k.a. PCA)

1. Compute image gradients over small region

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


2. Subtract mean from each image gradient

3. Compute the covariance matrix

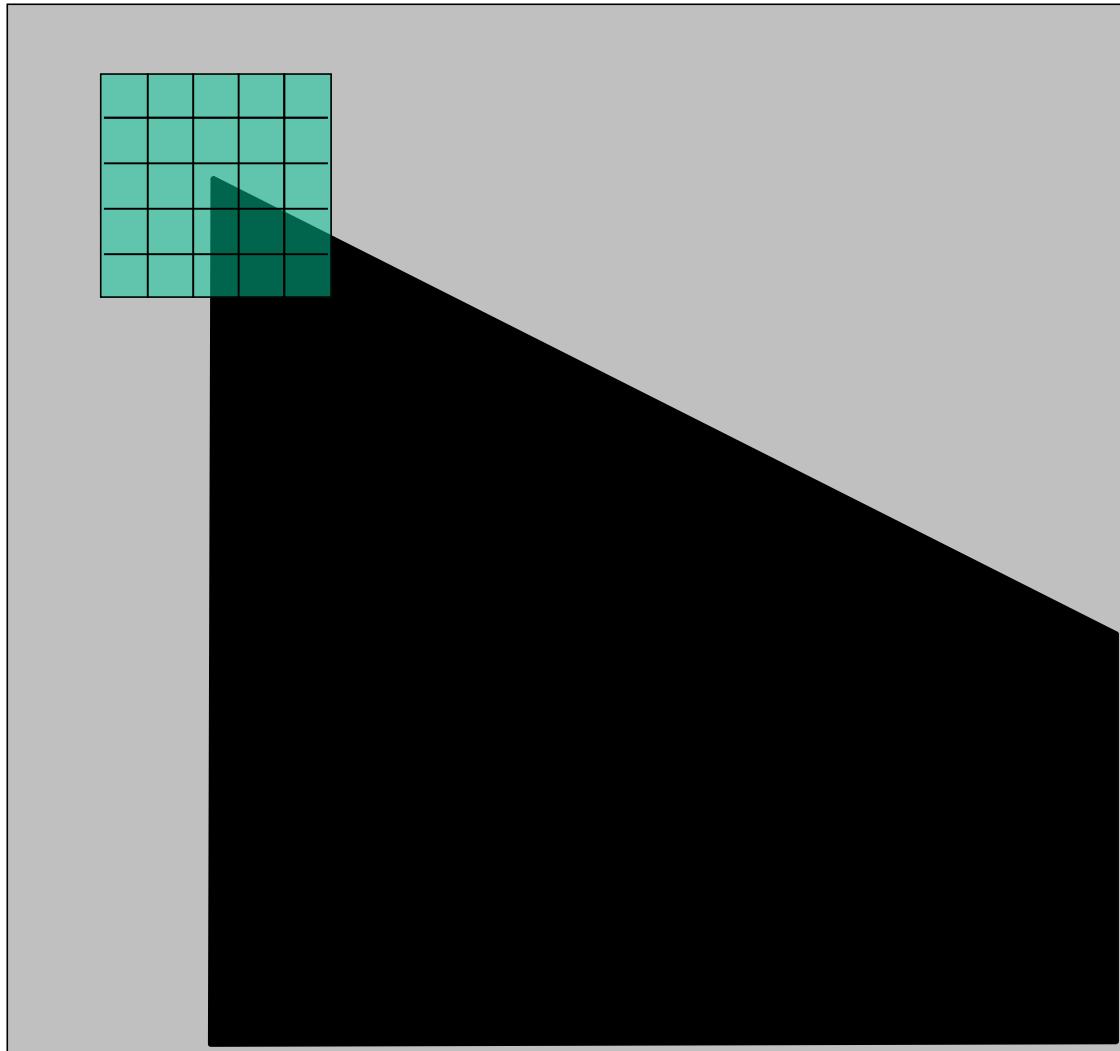
4. Compute eigenvectors and eigenvalues

5. Use threshold on eigenvalues to detect corners

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

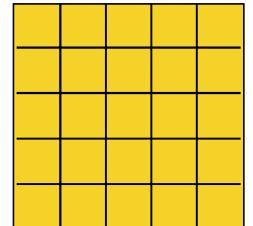
1. Compute image gradients over a small region  
(not just a single pixel)

# 1. Compute image gradients over a small region (not just a single pixel)



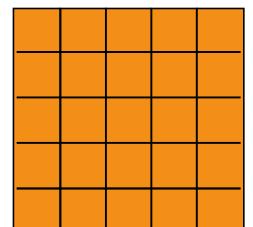
array of x gradients

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



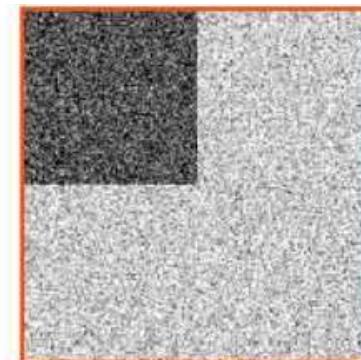
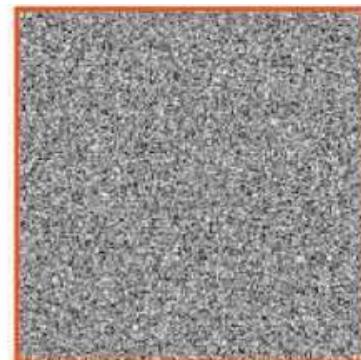
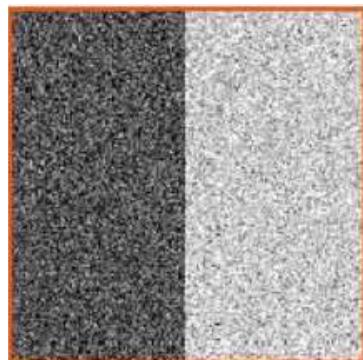
array of y gradients

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

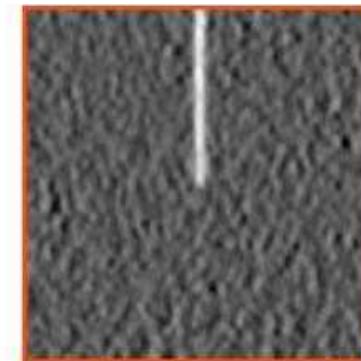
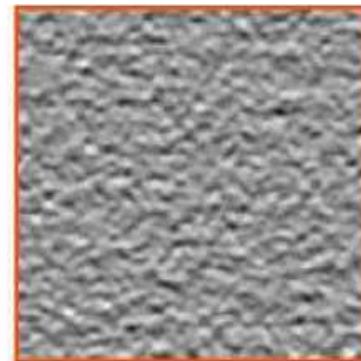
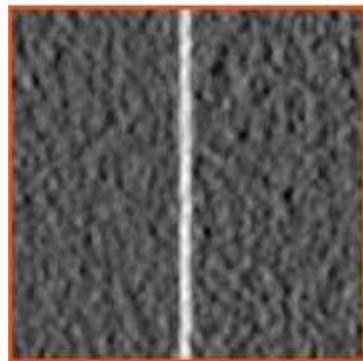


# visualization of gradients

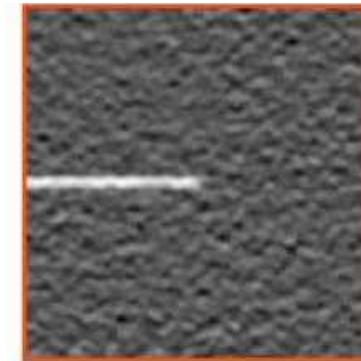
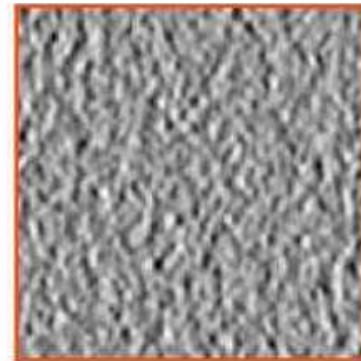
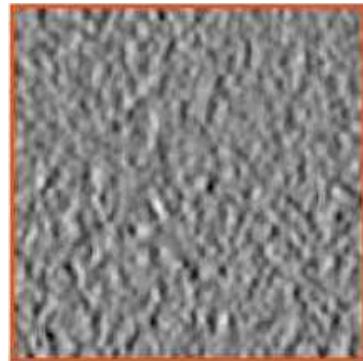
image

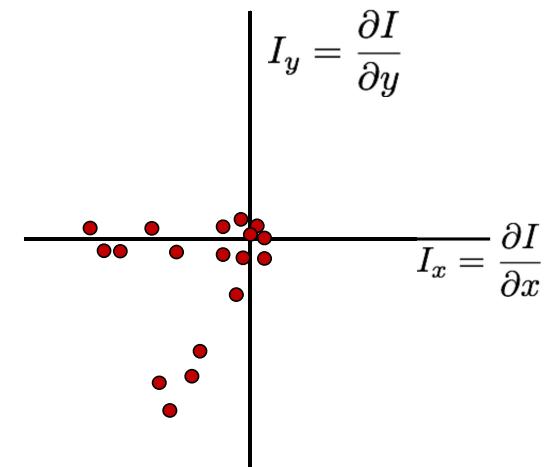
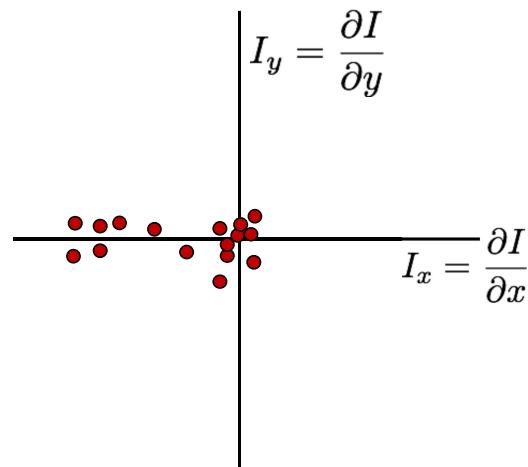
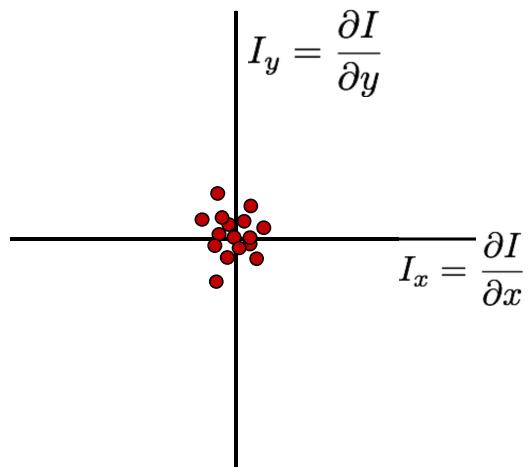
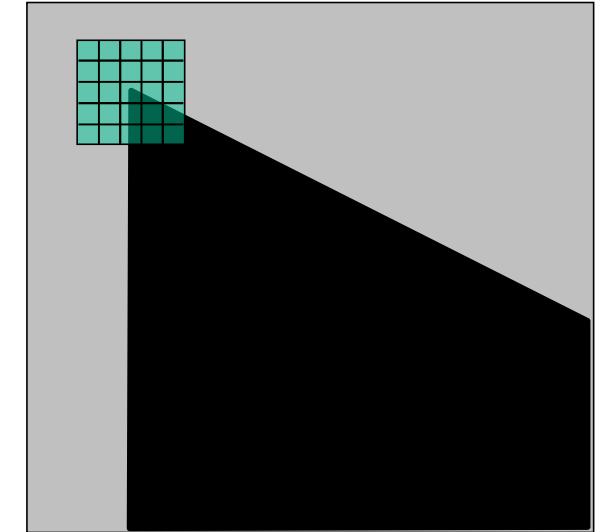
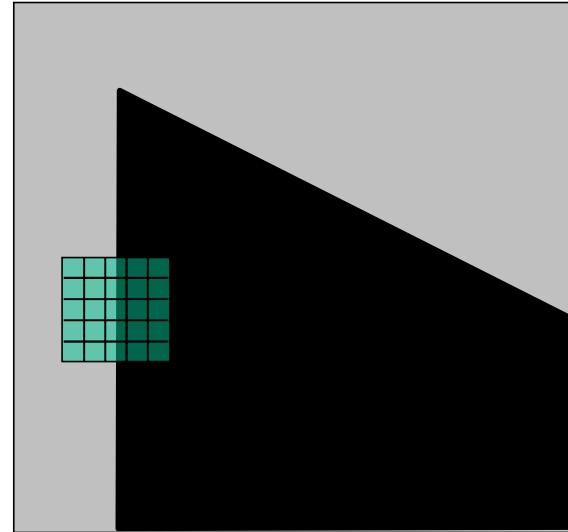
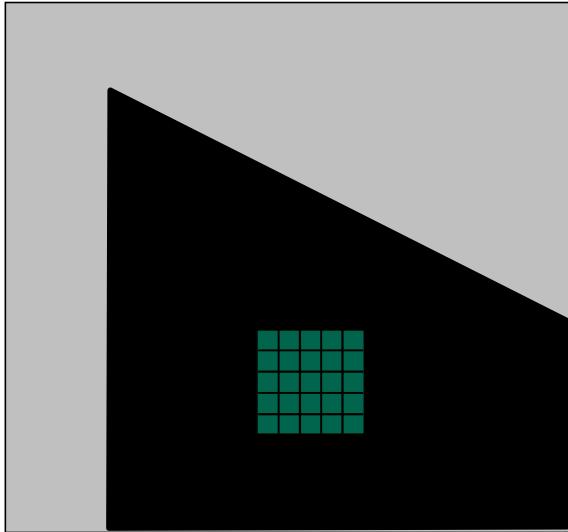


X derivative

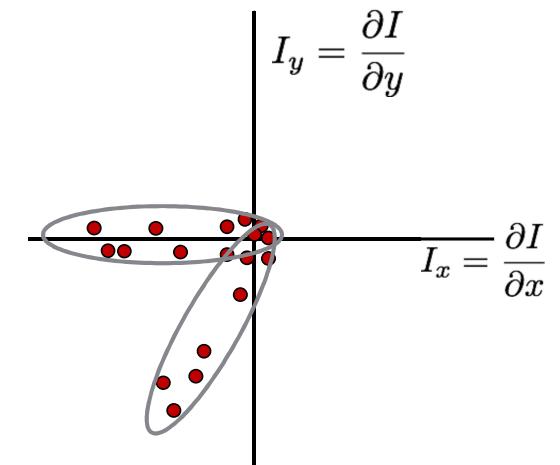
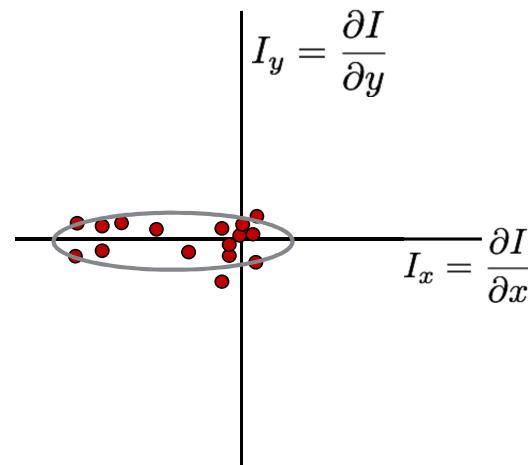
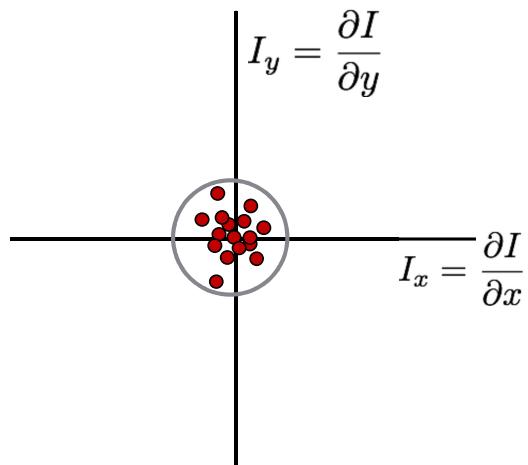
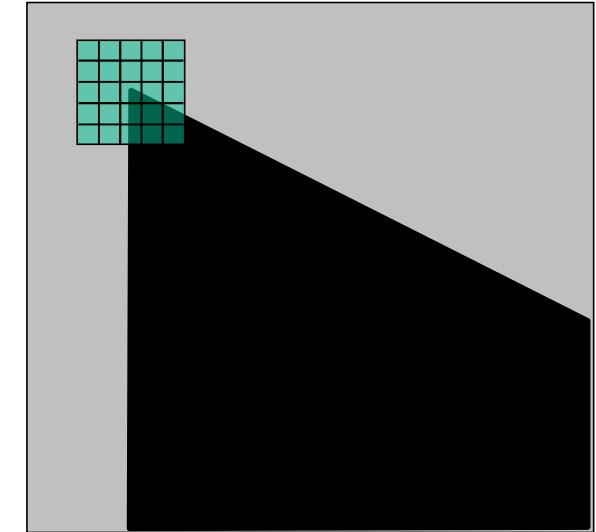
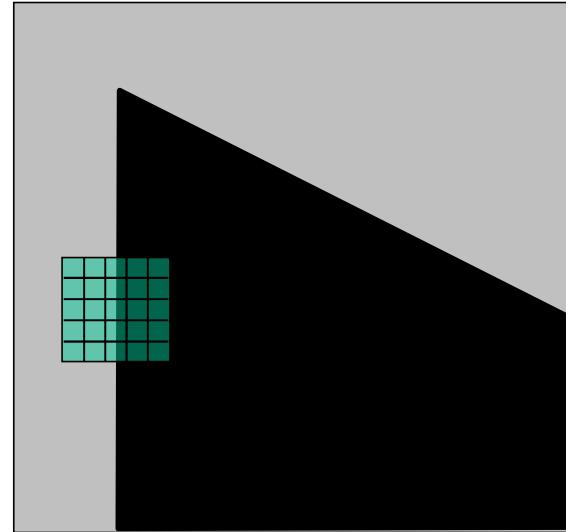
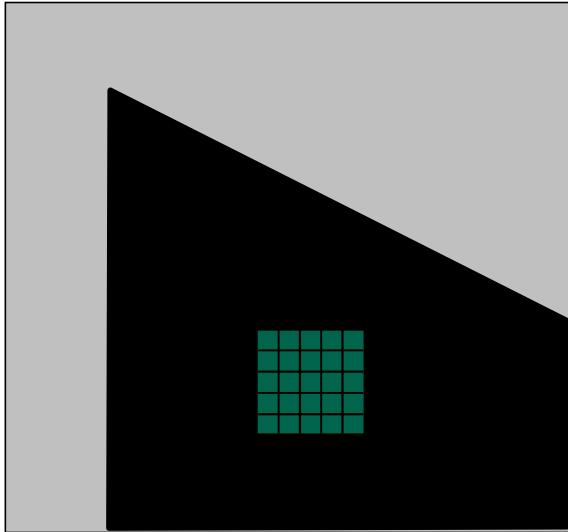


Y derivative

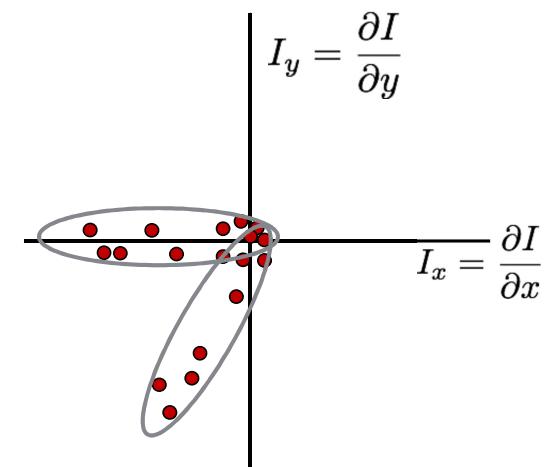
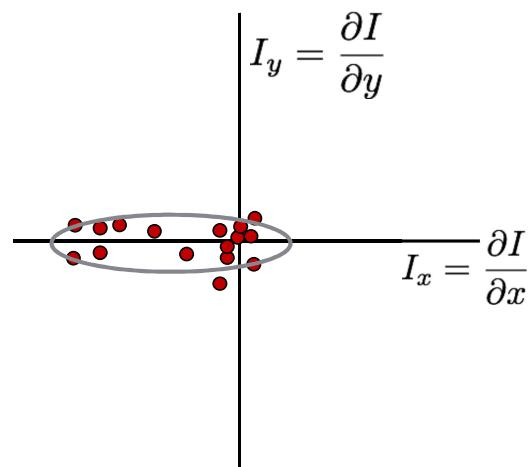
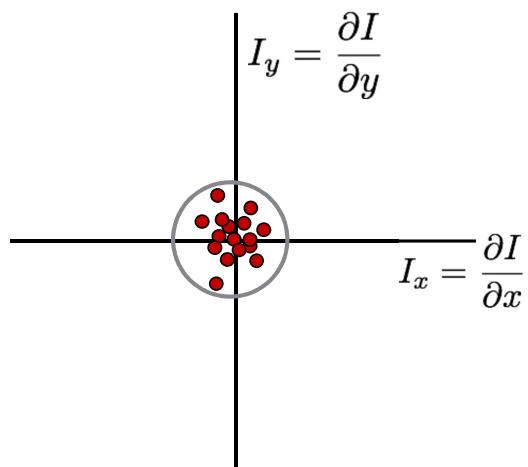
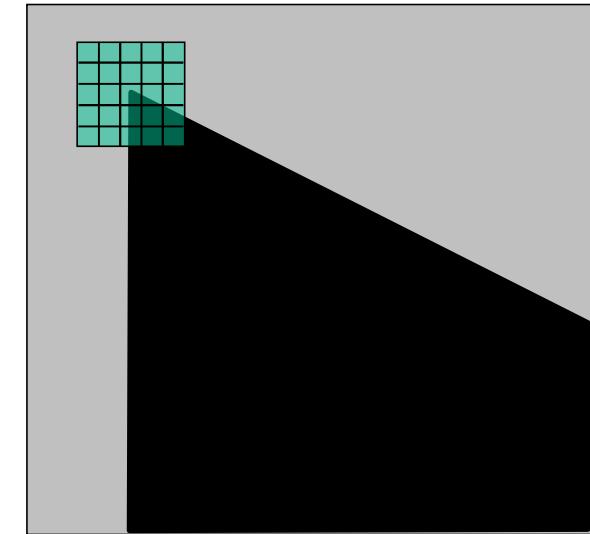
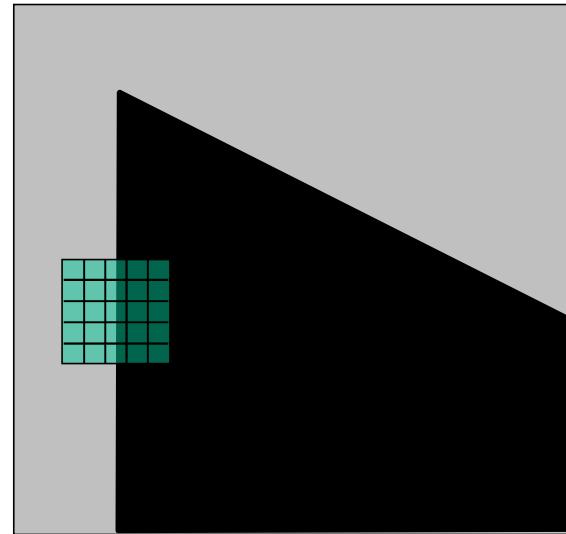
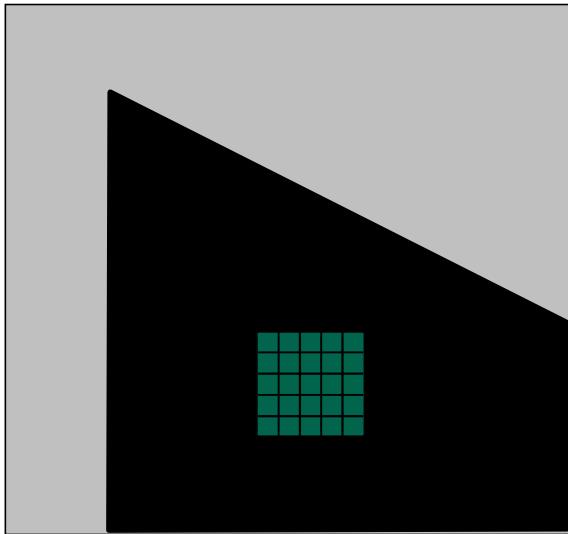




*What does the distribution tell you about the region?*



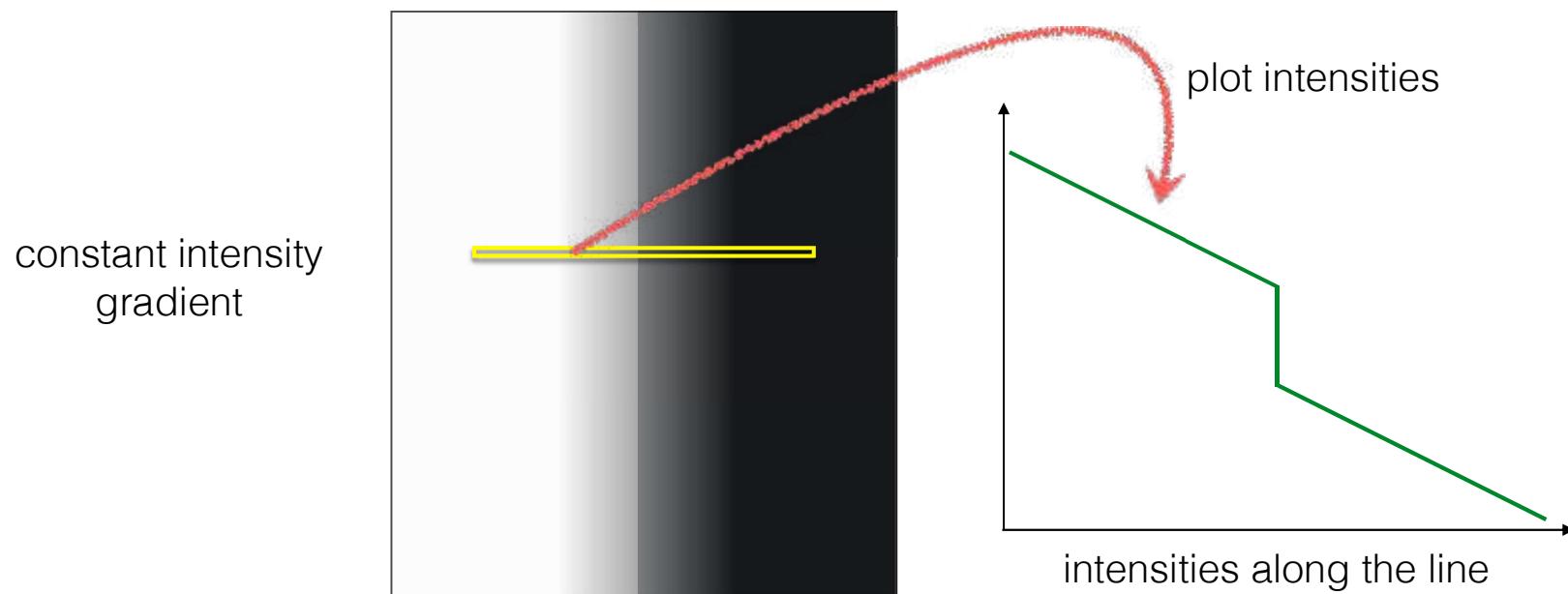
distribution reveals edge orientation and magnitude



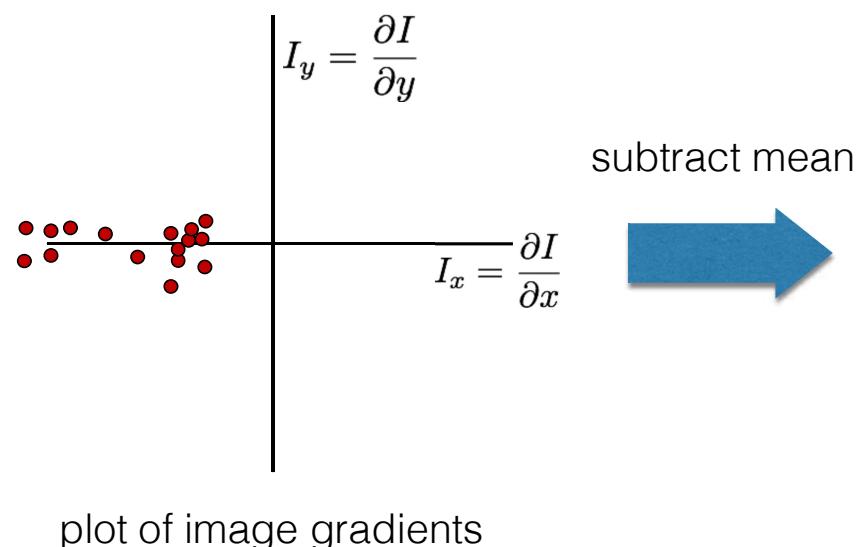
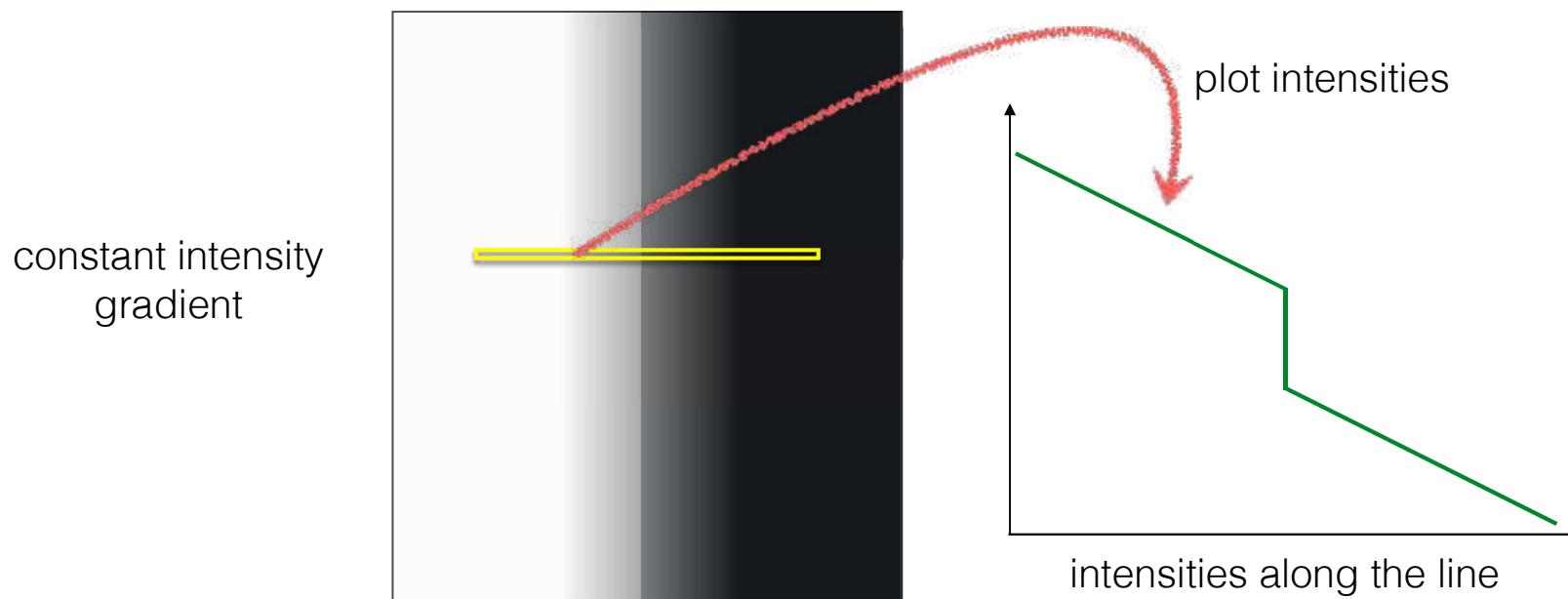
*How do you quantify orientation and magnitude?*

2. Subtract the mean from each image gradient

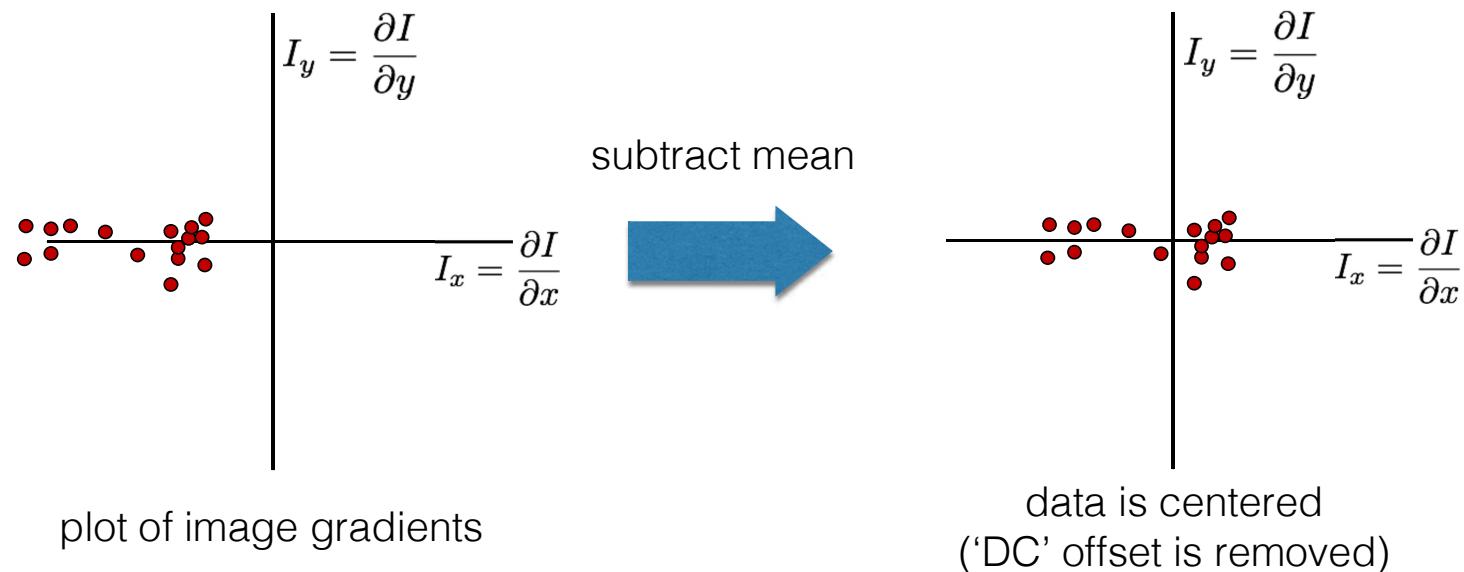
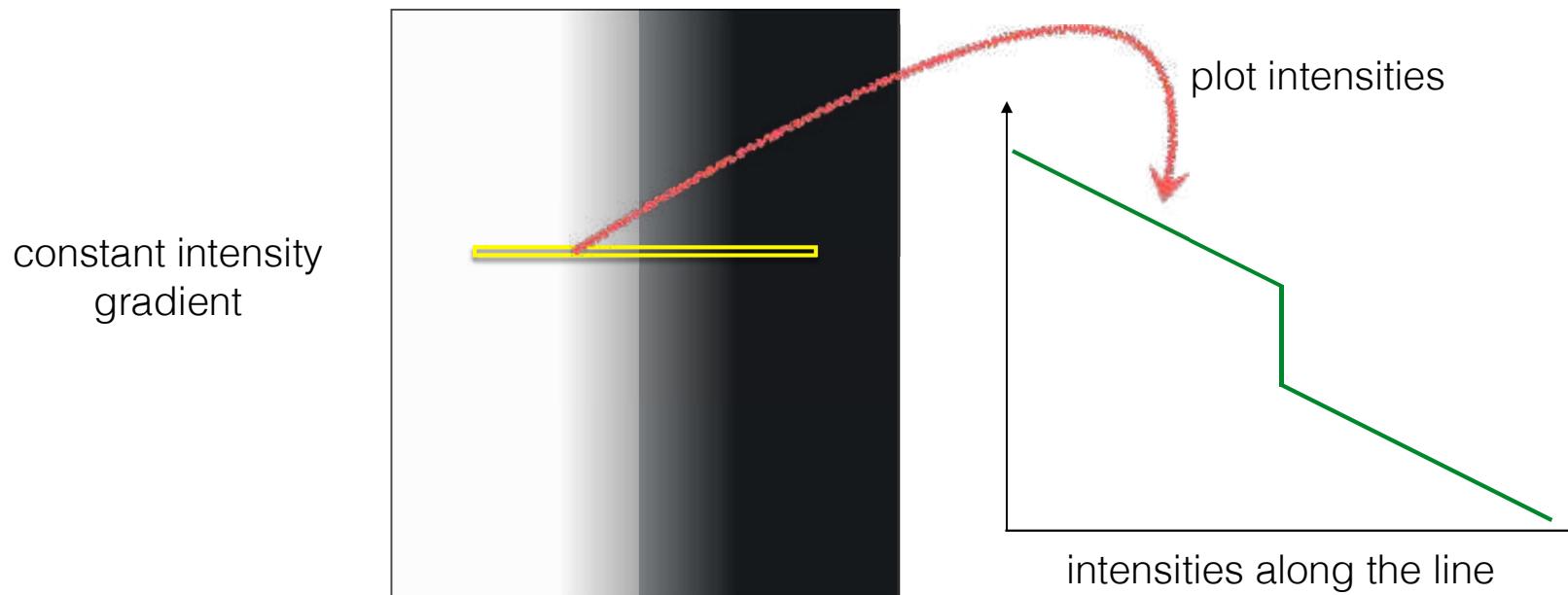
## 2. Subtract the mean from each image gradient



## 2. Subtract the mean from each image gradient



## 2. Subtract the mean from each image gradient



3. Compute the covariance matrix

### 3. Compute the covariance matrix

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

$$\sum_{p \in P} I_x I_y = \text{sum}\left( \begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} \right) *$$

array of x gradients

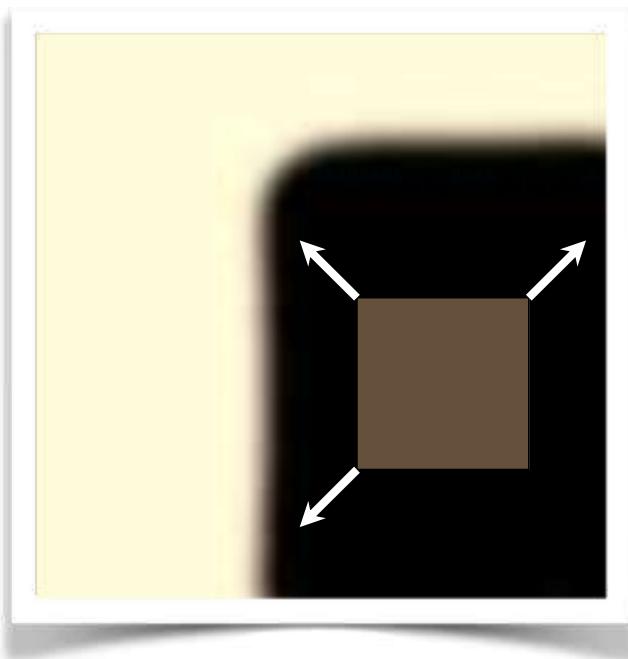
$$I_x = \frac{\partial I}{\partial x} \qquad I_y = \frac{\partial I}{\partial y}$$
$$\begin{array}{|c|c|c|c|} \hline & & & \\ \hline \end{array} )$$

array of y gradients

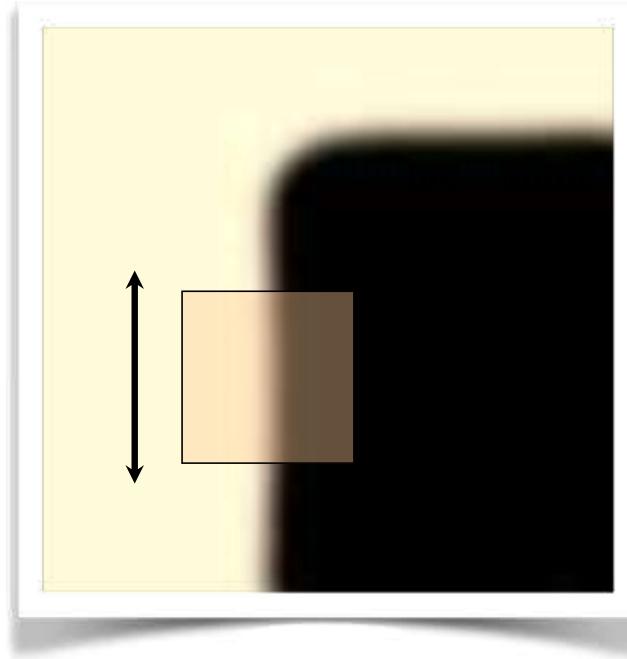
*Where does this covariance matrix come from?*

Easily recognized by looking through a small window

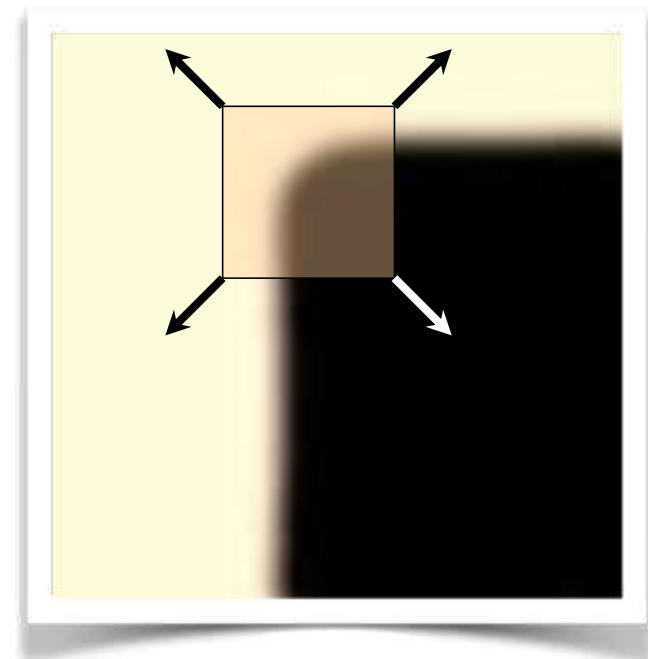
Shifting the window should give large change in intensity



“flat” region:  
no change in all  
directions



“edge”:  
no change along the edge  
direction



“corner”:  
significant change in all  
directions

Some mathematical background...

# Error function

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

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

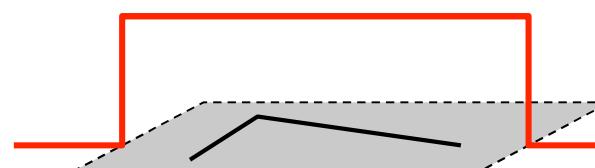
Error  
function

Window  
function

Shifted  
intensity

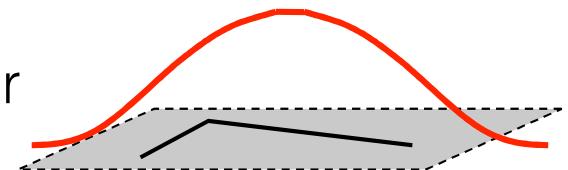
Intensity

Window function  $w(x, y) =$



1 in window, 0 outside

or



Gaussian

# Error function approximation

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

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

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

# 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  $2 \times 2$  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...

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

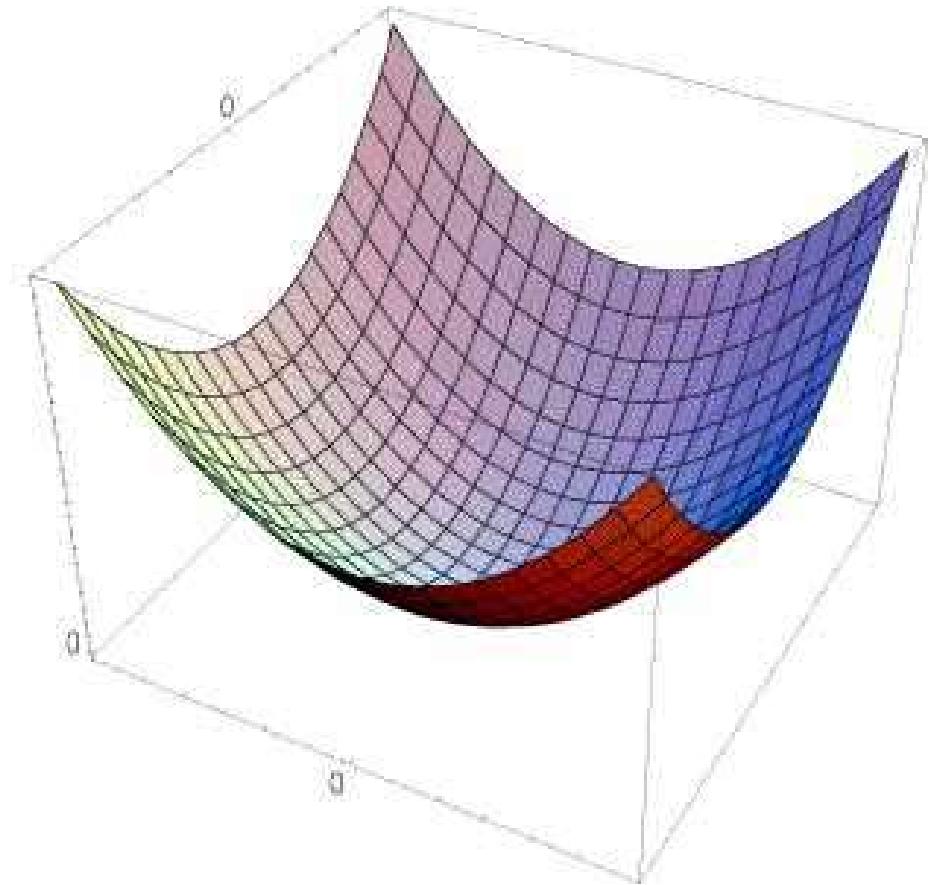
we are fitting a quadratic to the gradients over a  
small image region

# Visualization of a quadratic

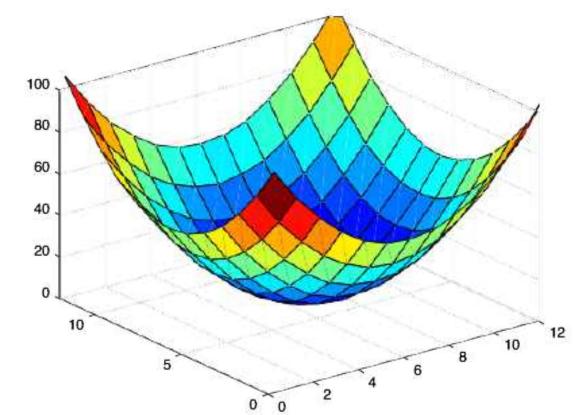
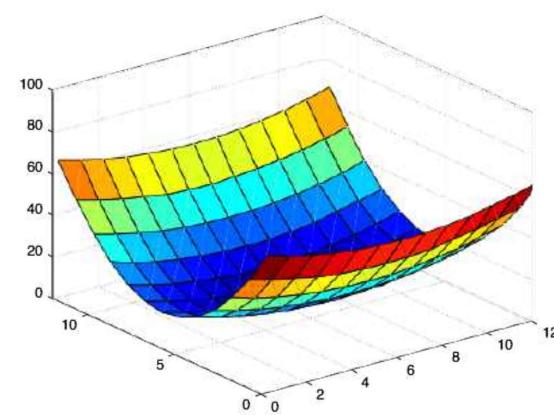
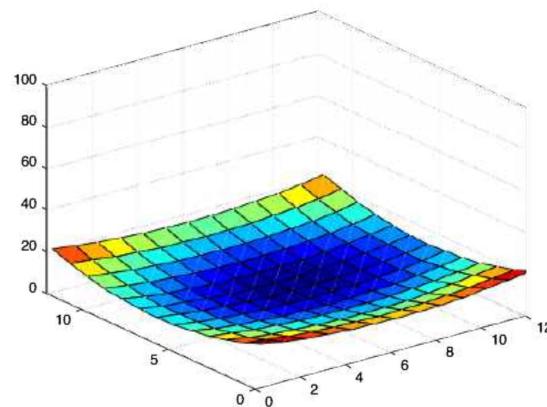
The surface  $E(u,v)$  is locally approximated by a quadratic form

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

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

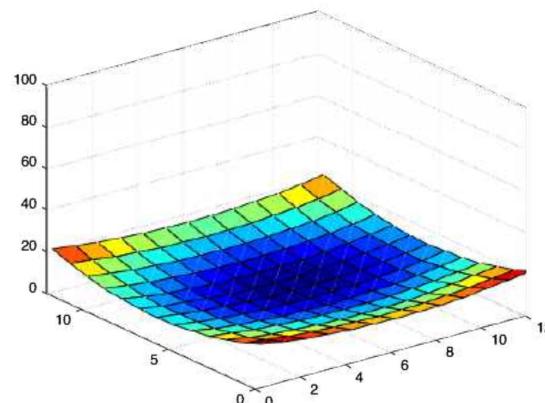


*Which error surface indicates a good image feature?*

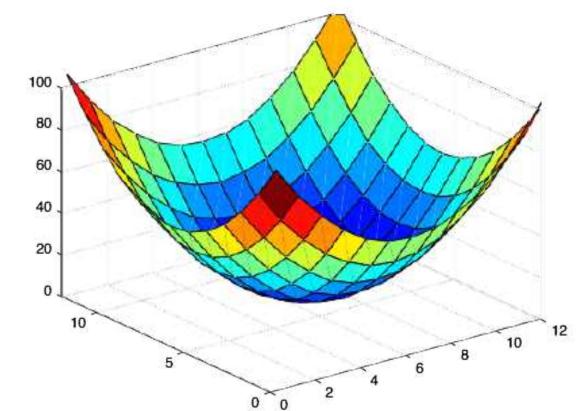
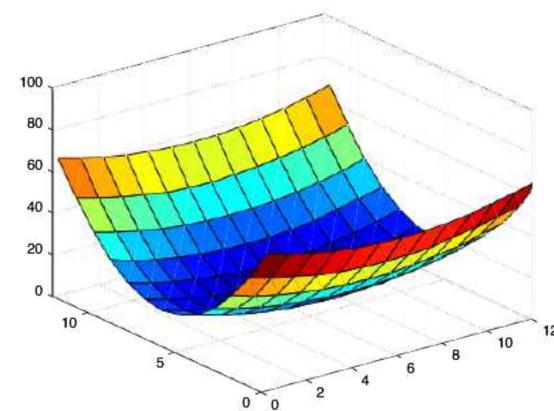


*What kind of image patch do these surfaces represent?*

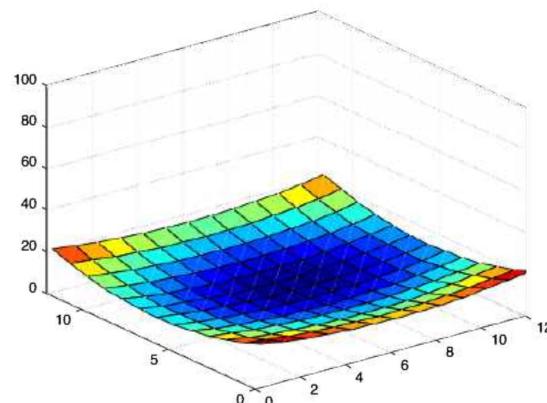
*Which error surface indicates a good image feature?*



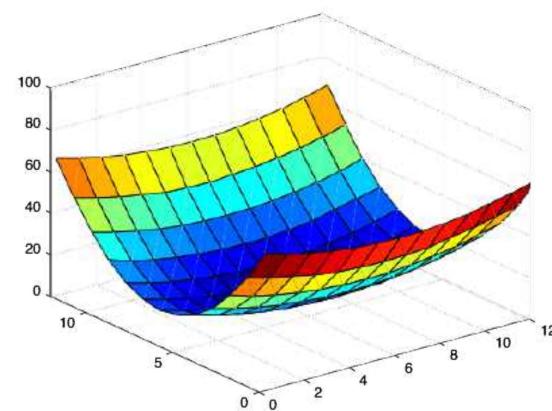
flat



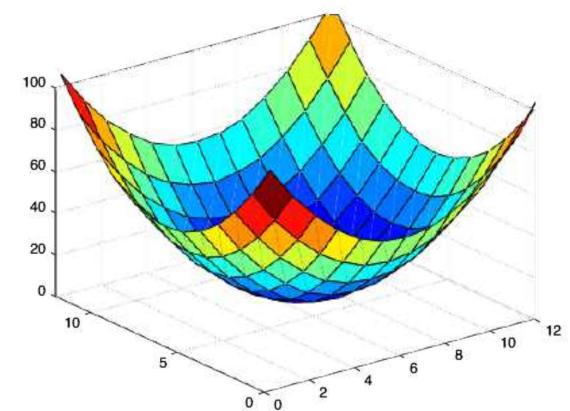
*Which error surface indicates a good image feature?*



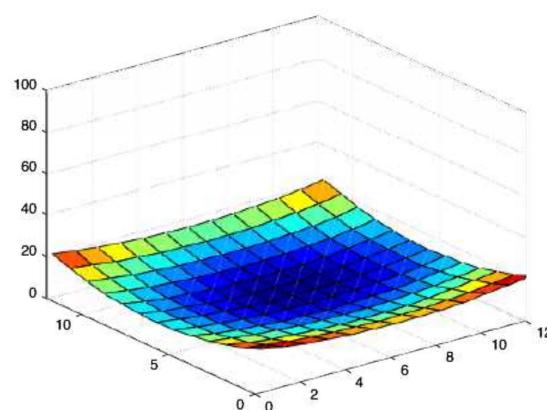
flat



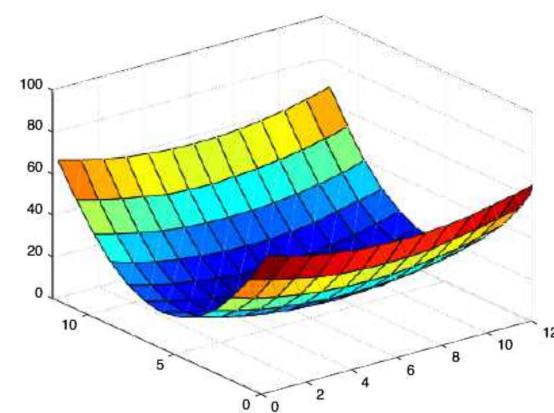
edge  
'line'



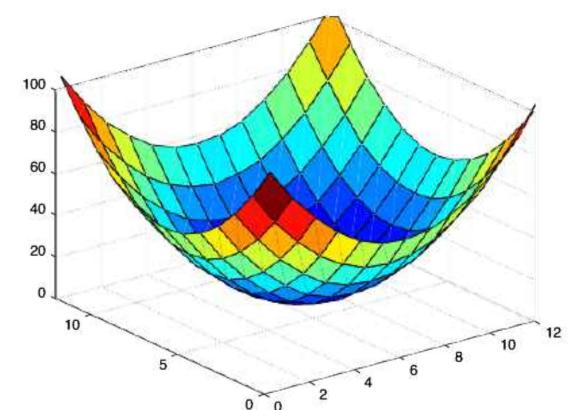
*Which error surface indicates a good image feature?*



flat



edge  
'line'



corner  
'dot'

4. Compute eigenvalues and eigenvectors

#### 4. Compute eigenvalues and eigenvectors

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

↓  
eigenvalue  
↑↑  
eigenvector

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

## 4. Compute eigenvalues and eigenvectors

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

eigenvalue  
↓  
eigenvector

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

1. Compute the determinant of  
(returns a polynomial)

$$M - \lambda I$$

## 4. Compute eigenvalues and eigenvectors

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

eigenvalue  
↓  
eigenvector

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

1. Compute the determinant of  
(returns a polynomial)

$$M - \lambda I$$

2. Find the roots of polynomial  
(returns eigenvalues)

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

## 4. Compute eigenvalues and eigenvectors

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

eigenvalue  
↓  
 $(M - \lambda I)\mathbf{e} = 0$   
eigenvector

1. Compute the determinant of  
(returns a polynomial)

$$M - \lambda I$$

2. Find the roots of polynomial  
(returns eigenvalues)

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

3. For each eigenvalue, solve  
(returns eigenvectors)

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

eig(M)

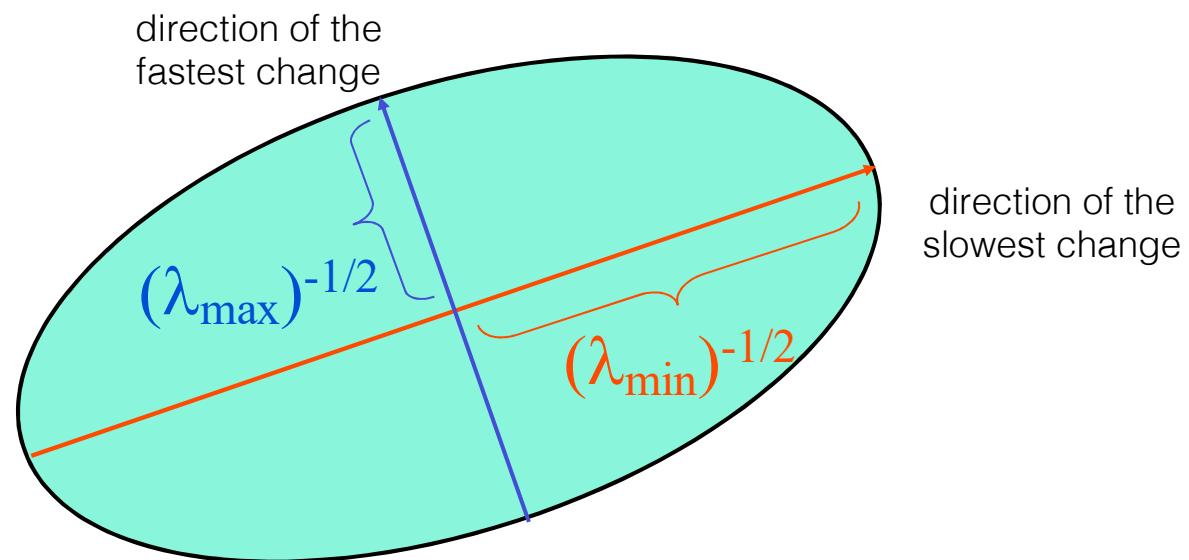
# Visualization as an ellipse

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

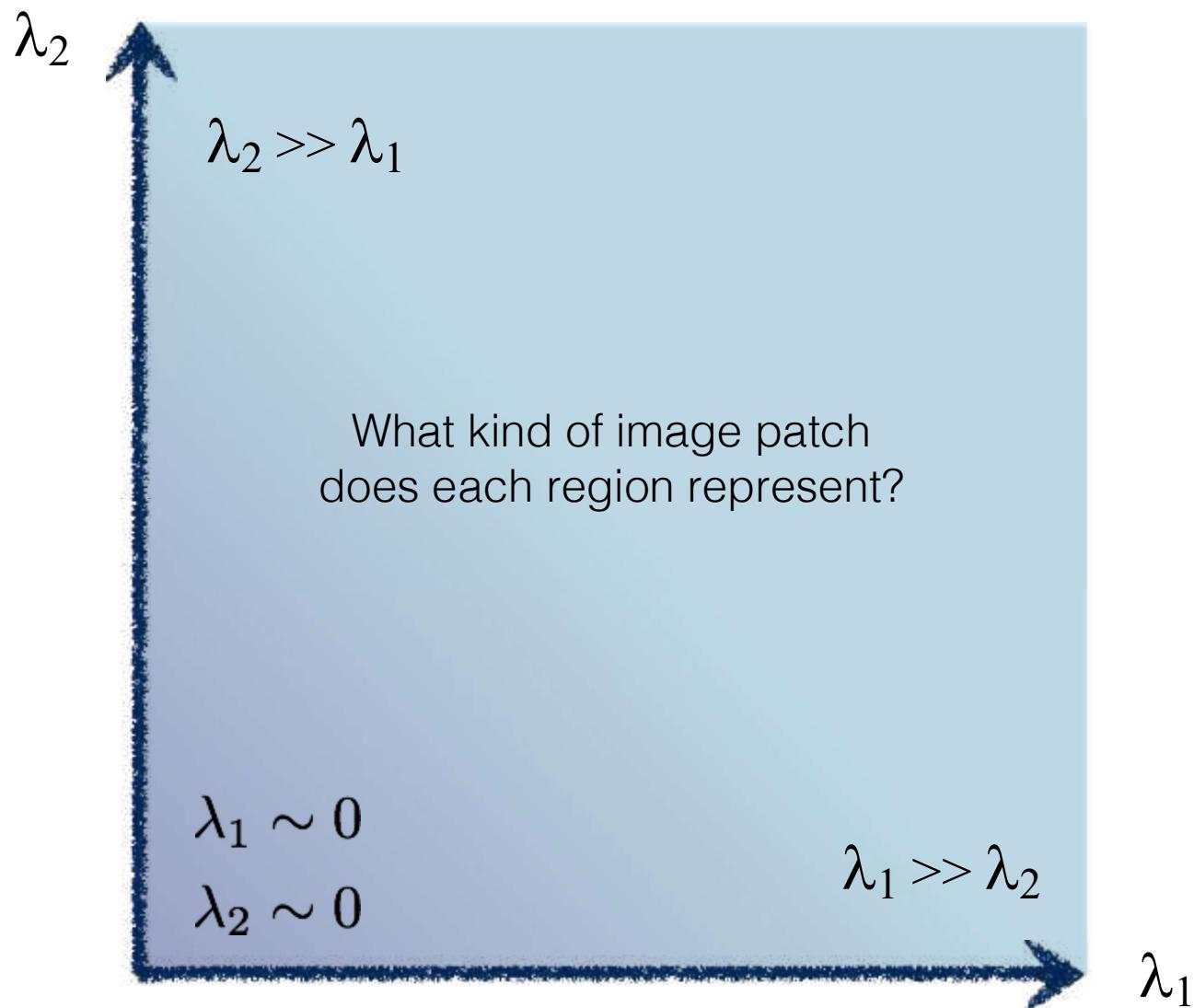
We can visualize  $M$  as an ellipse with axis lengths determined by the eigenvalues and orientation determined by  $R$

Ellipse equation:

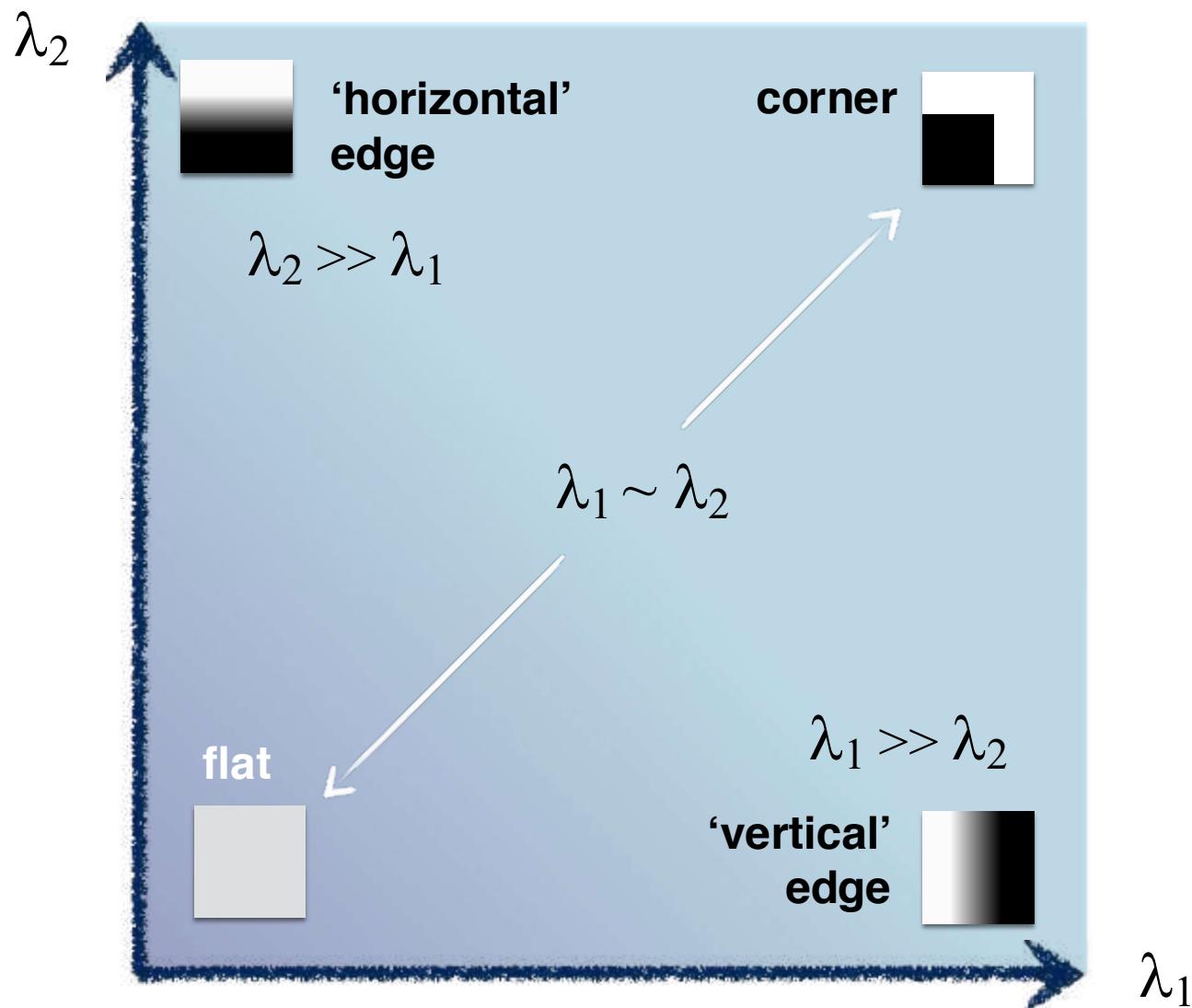
$$[u \ v] M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$



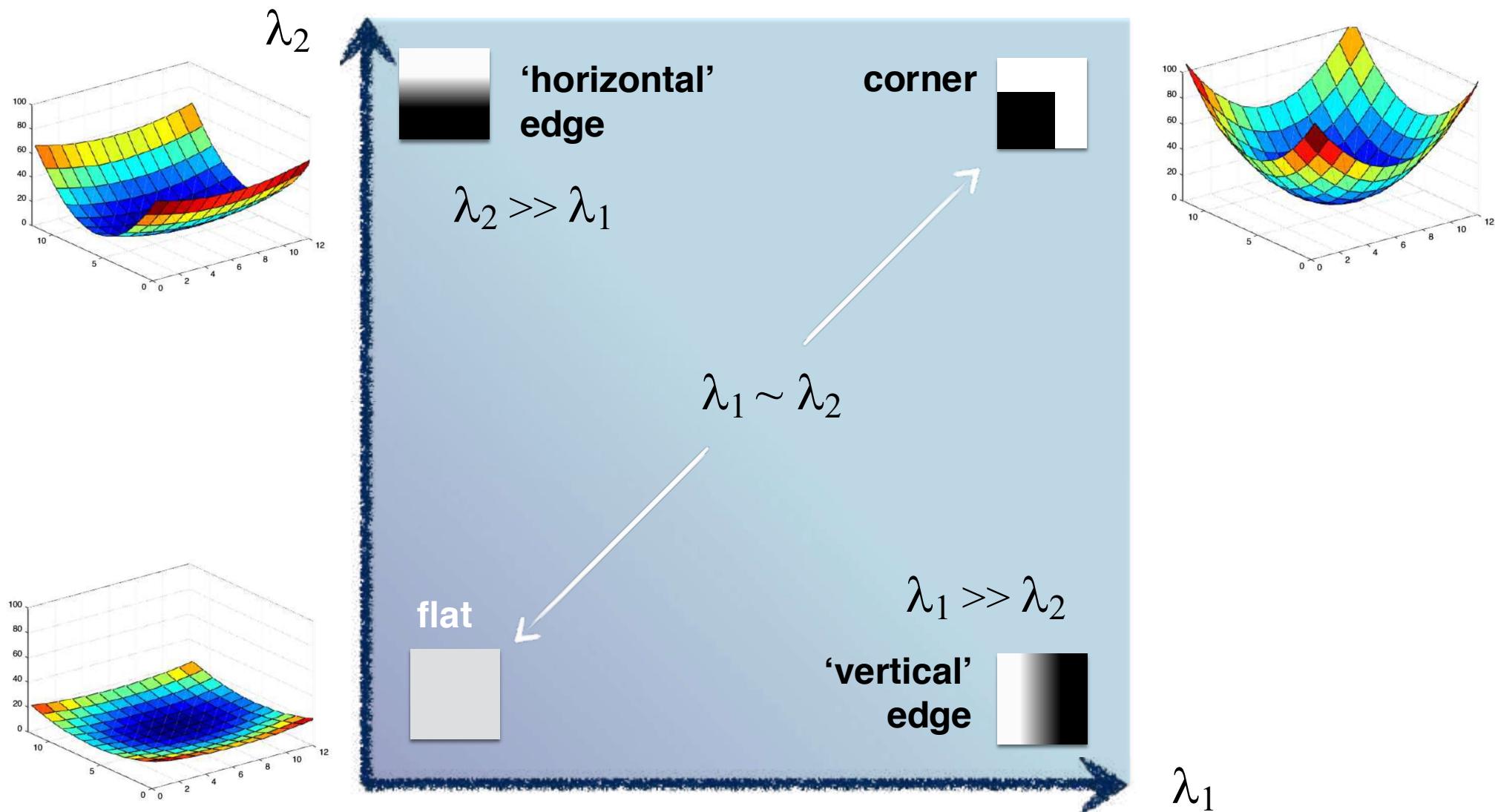
# interpreting eigenvalues



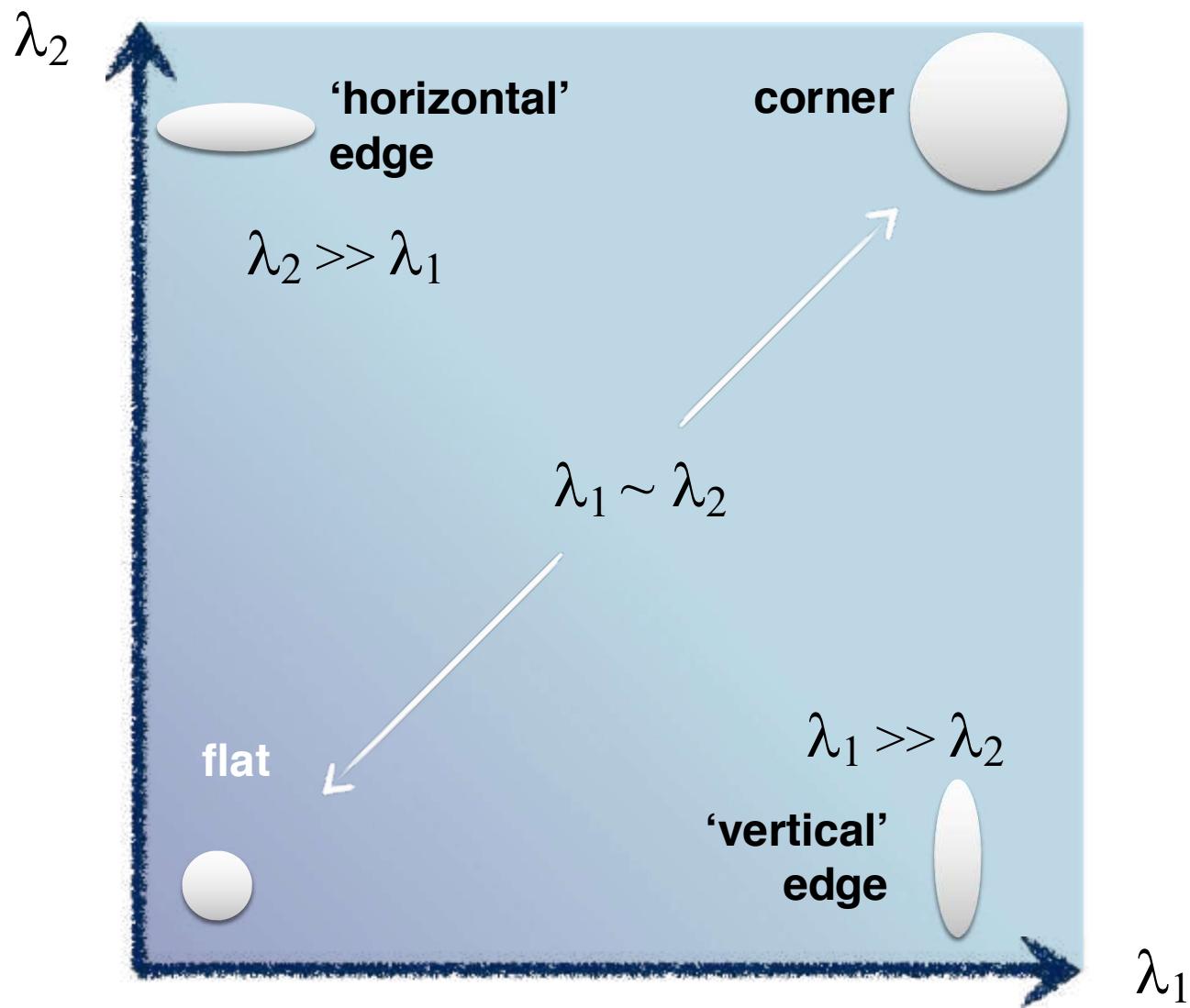
# interpreting eigenvalues



# interpreting eigenvalues

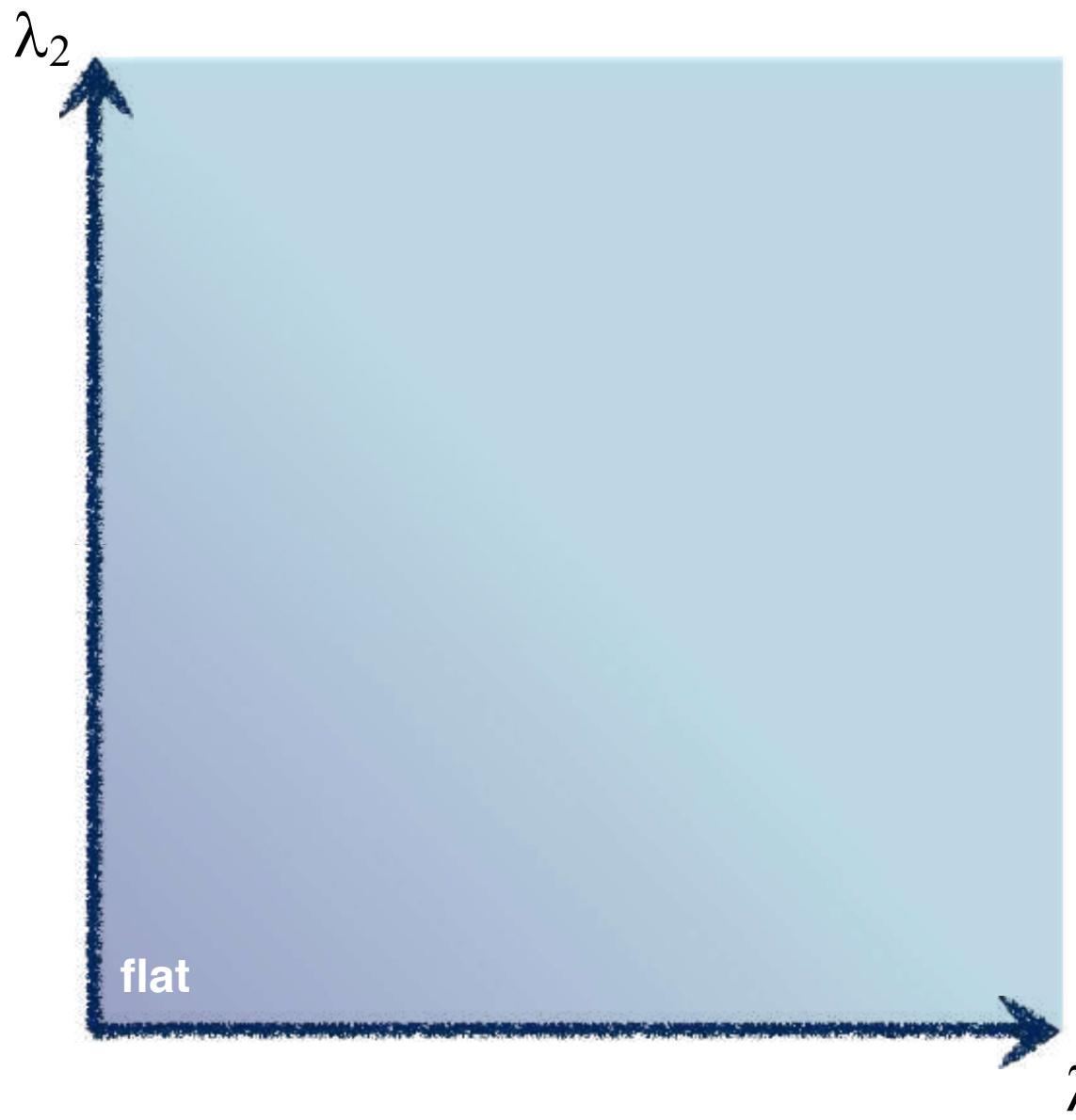


# interpreting eigenvalues



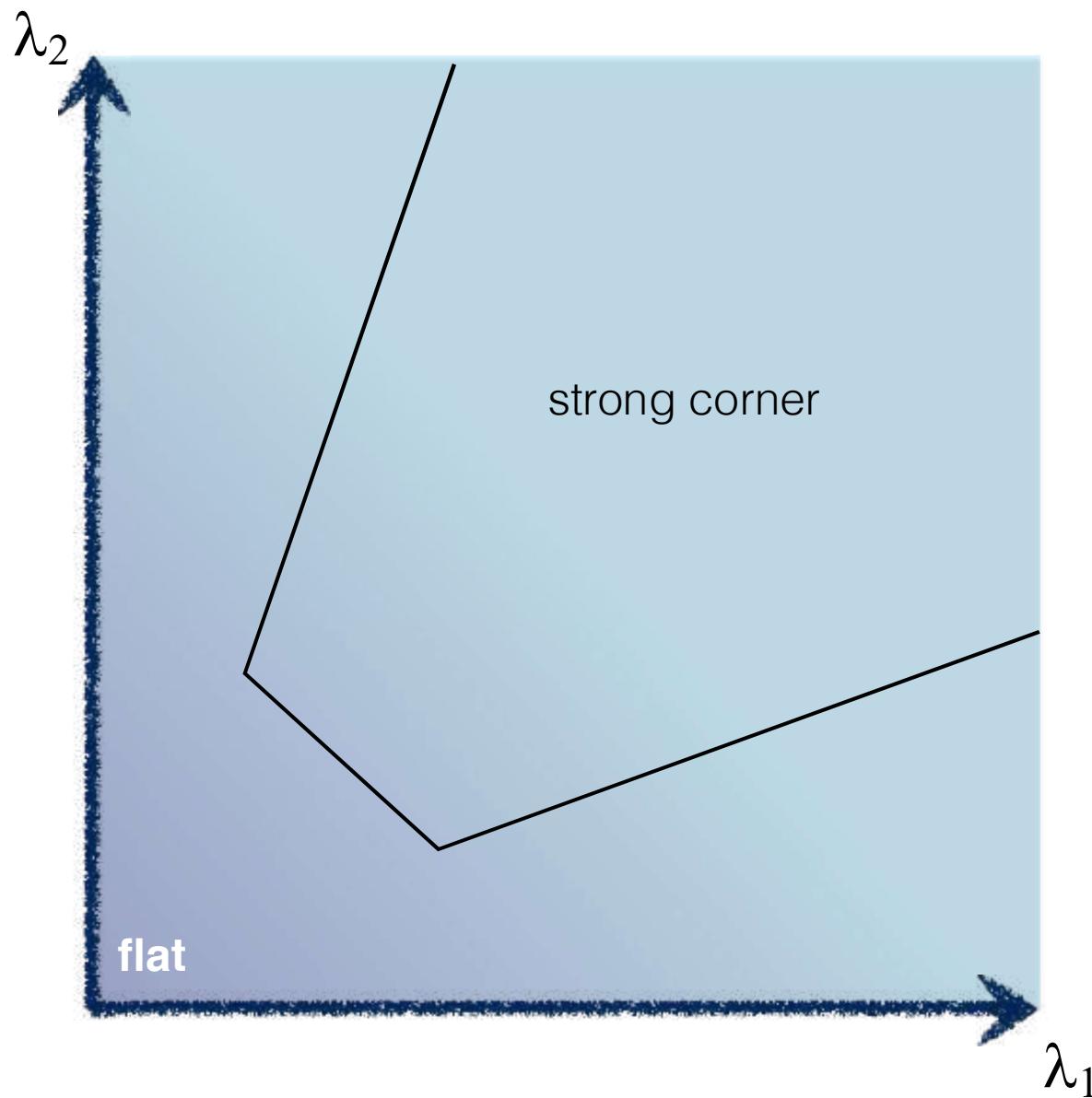
5. Use threshold on eigenvalues to detect corners

## 5. Use threshold on eigenvalues to detect corners



Think of a function to score 'cornerness'

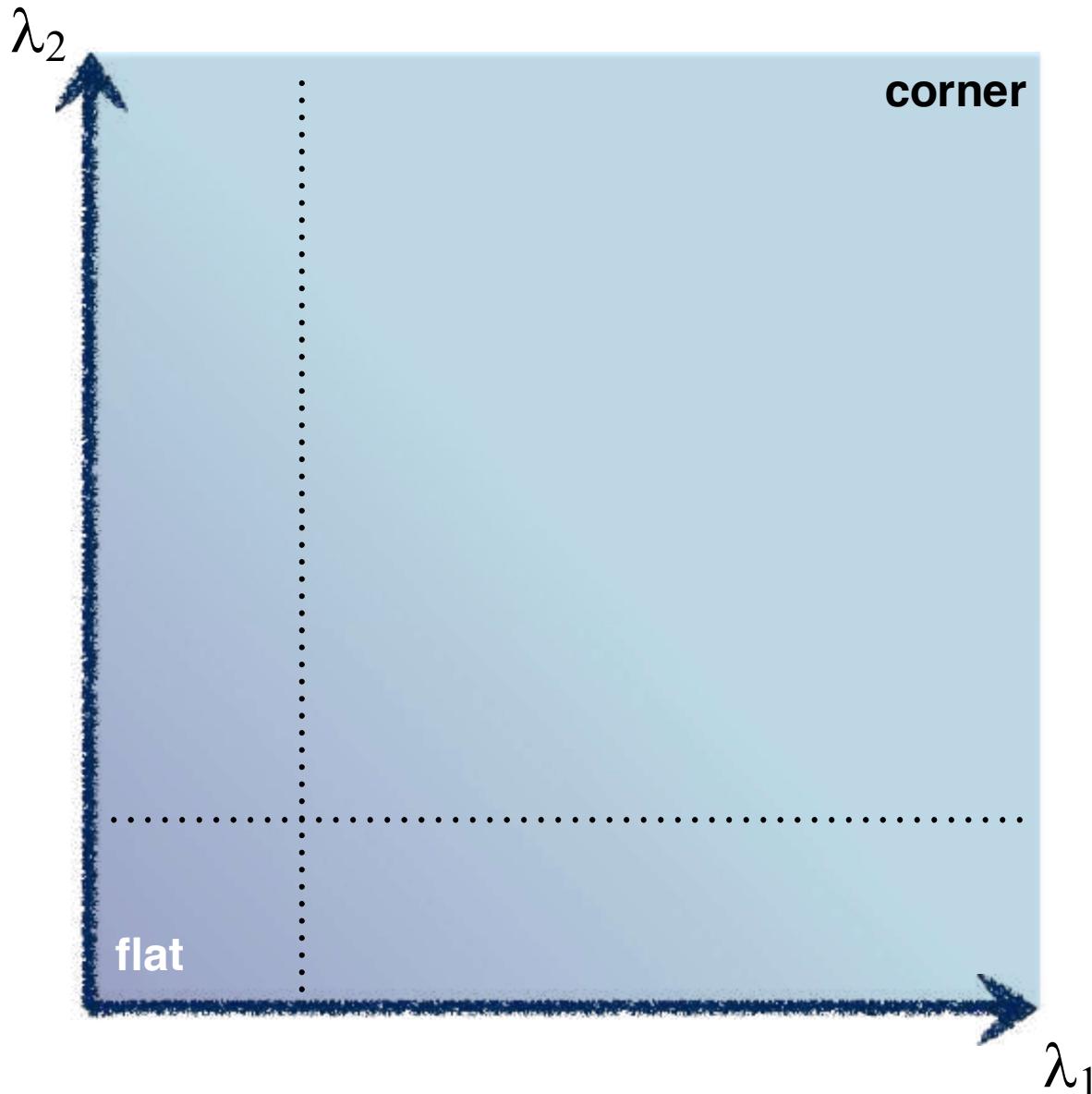
## 5. Use threshold on eigenvalues to detect corners



Think of a function to score 'cornerness'

## 5. Use threshold on eigenvalues to detect corners

^  
(a function of )

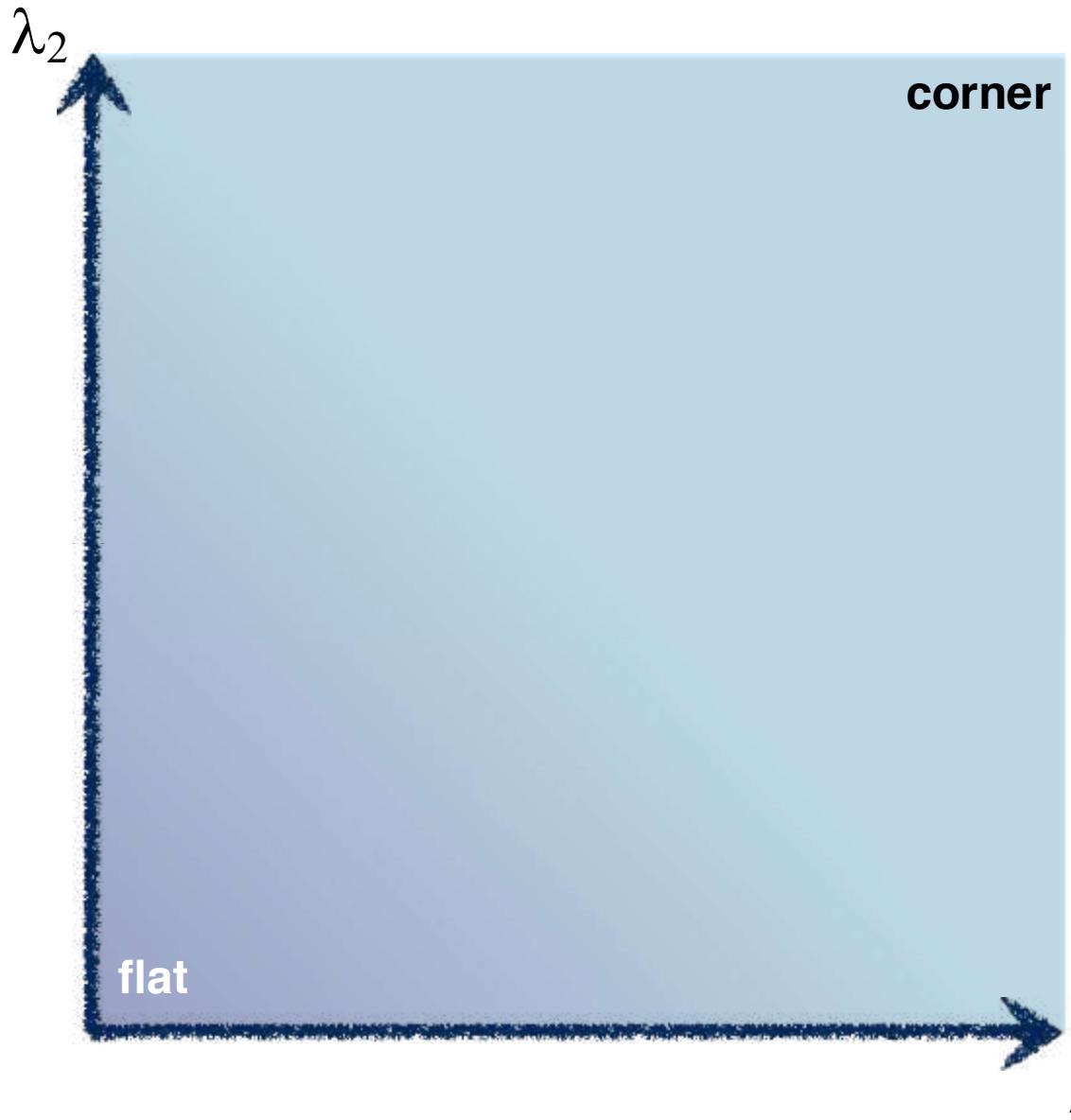


Use the smallest eigenvalue  
as the response function

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

## 5. Use threshold on eigenvalues to detect corners

^  
(a function of )

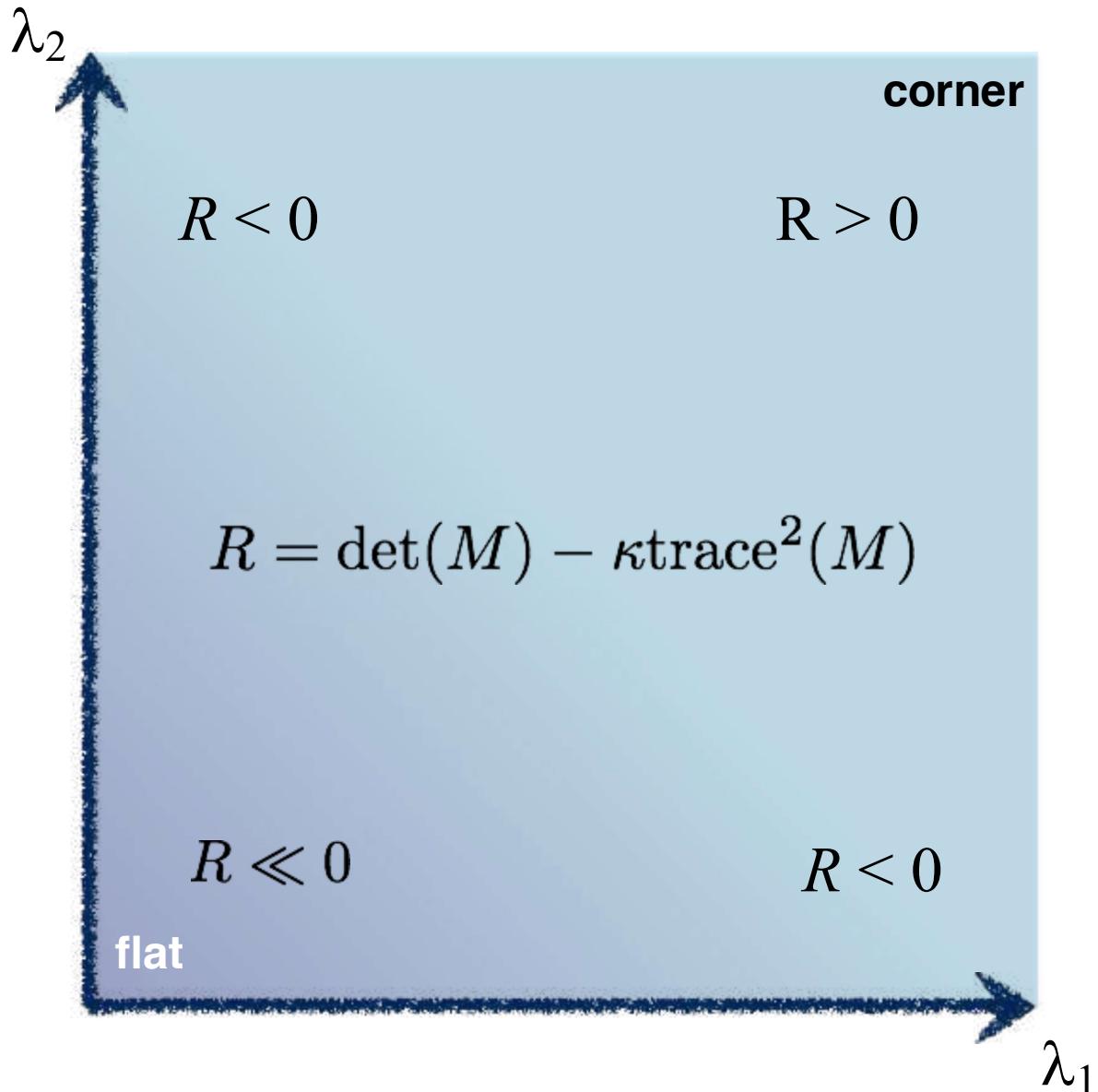


Eigenvalues need to be  
bigger than one.

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

Can compute this more efficiently...

## 5. Use threshold on $\lambda_1 \wedge$ (a function of )



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

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

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

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc$$

$$\text{trace} \begin{pmatrix} a & b \\ c & d \end{pmatrix} = a + d$$

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

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

2. Compute products of derivatives at every pixel

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

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

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

# Harris Detector

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

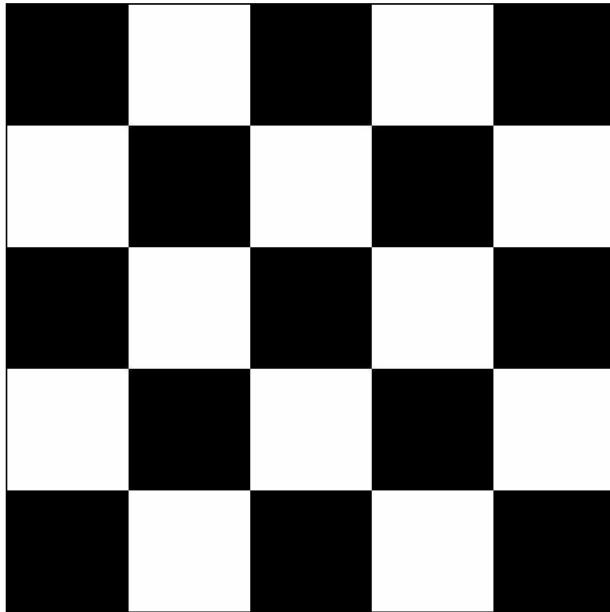
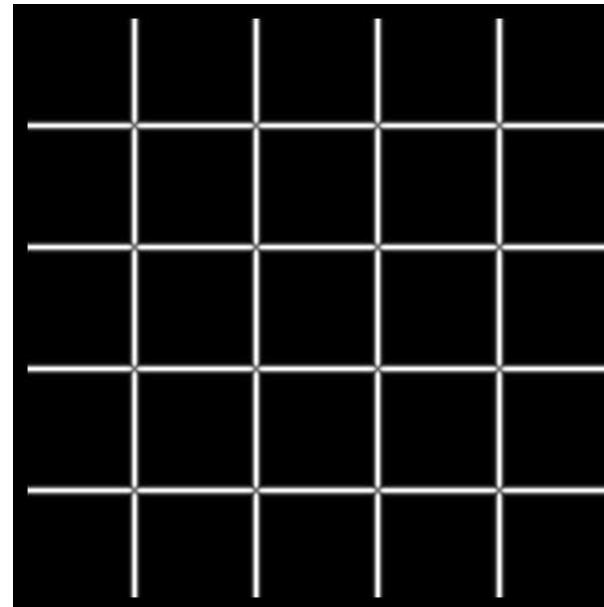
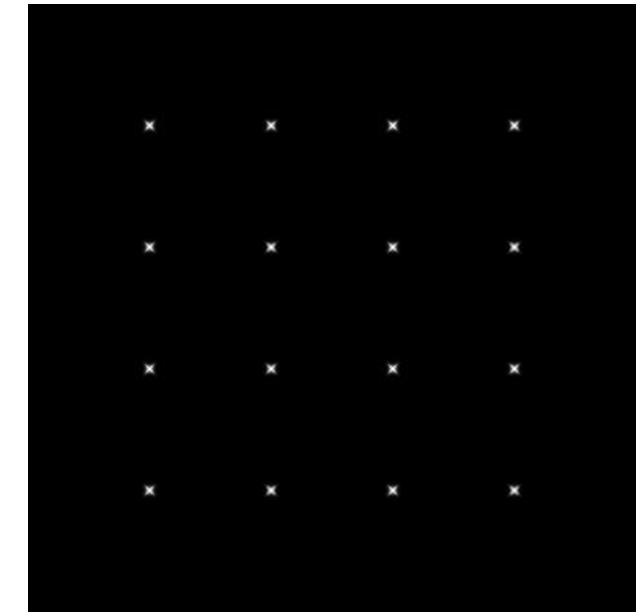
4. Define the matrix at each pixel

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

5. Compute the response of the detector at each pixel

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

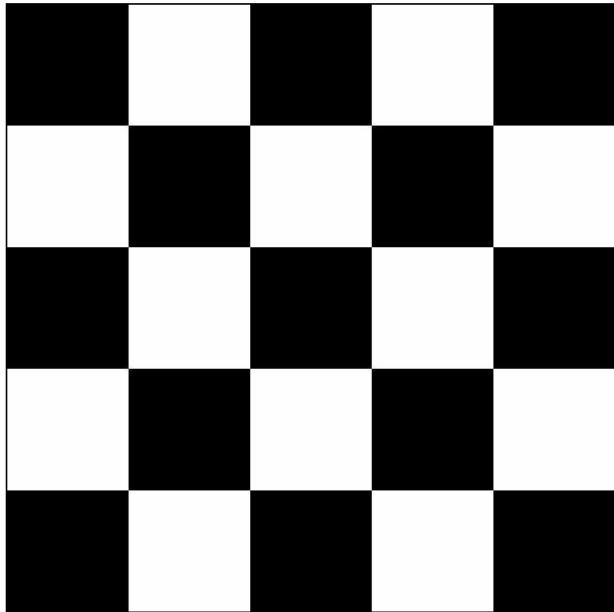
6. Threshold on value of R; compute non-max suppression.

 $I$  $\lambda_{\max}$  $\lambda_{\min}$ 

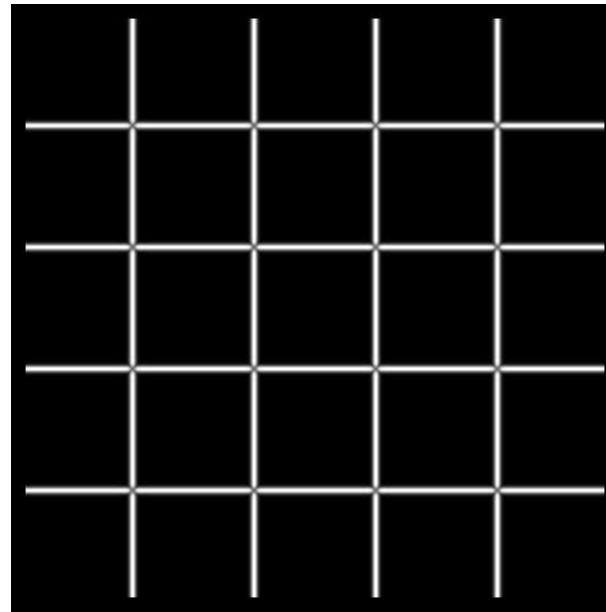
Yet another option:

$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2}$$

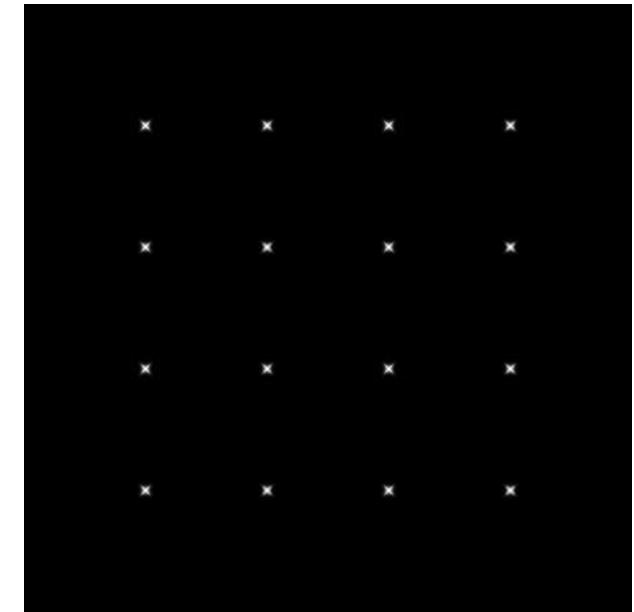
How do you write this equivalently  
using determinant and trace?



$I$



$\lambda_{\max}$

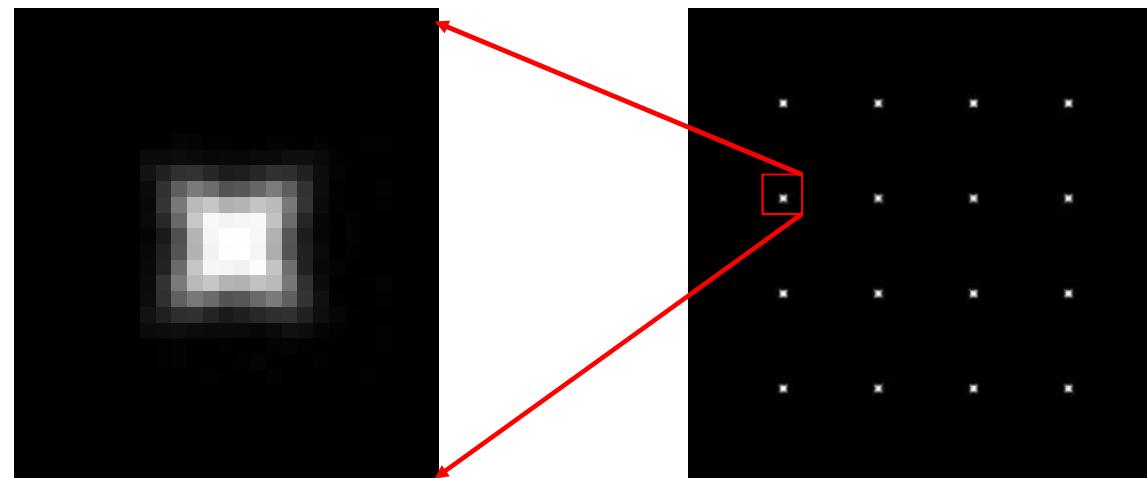


$\lambda_{\min}$

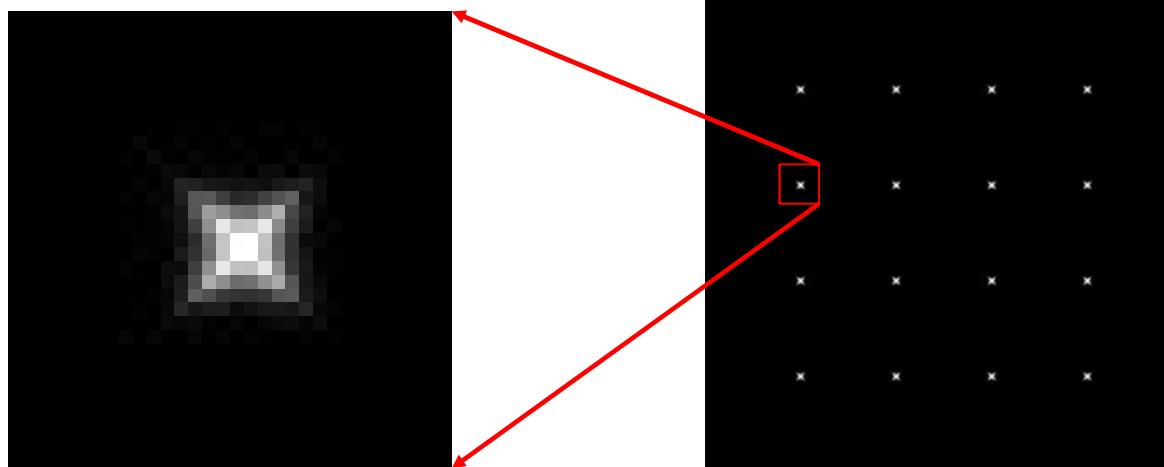
Yet another option:

$$f = \frac{\lambda_1 \lambda_2}{\lambda_1 + \lambda_2} = \frac{\text{determinant}(H)}{\text{trace}(H)}$$

# Different criteria



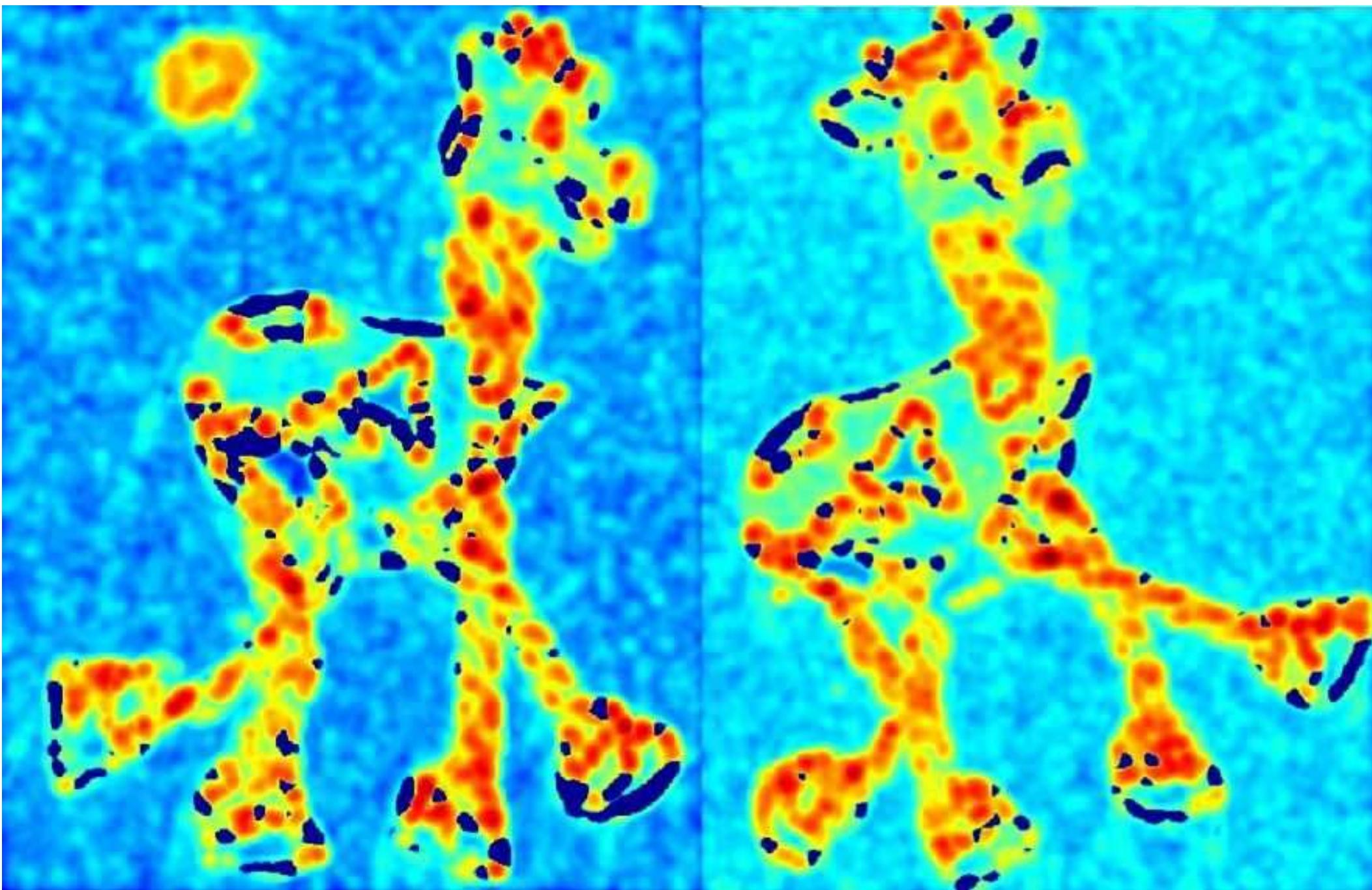
Harris criterion



$\lambda_{\min}$

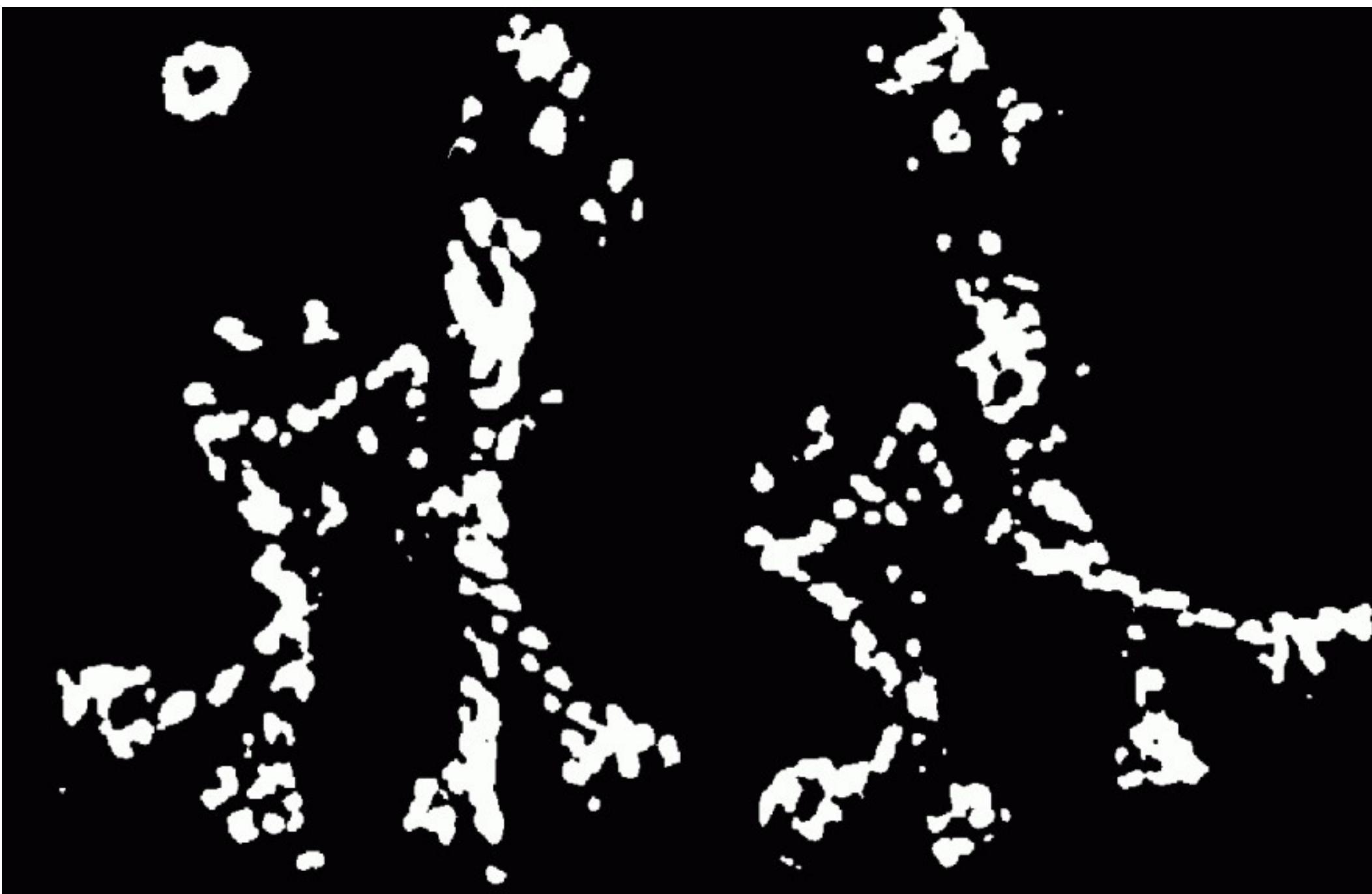


# Corner response

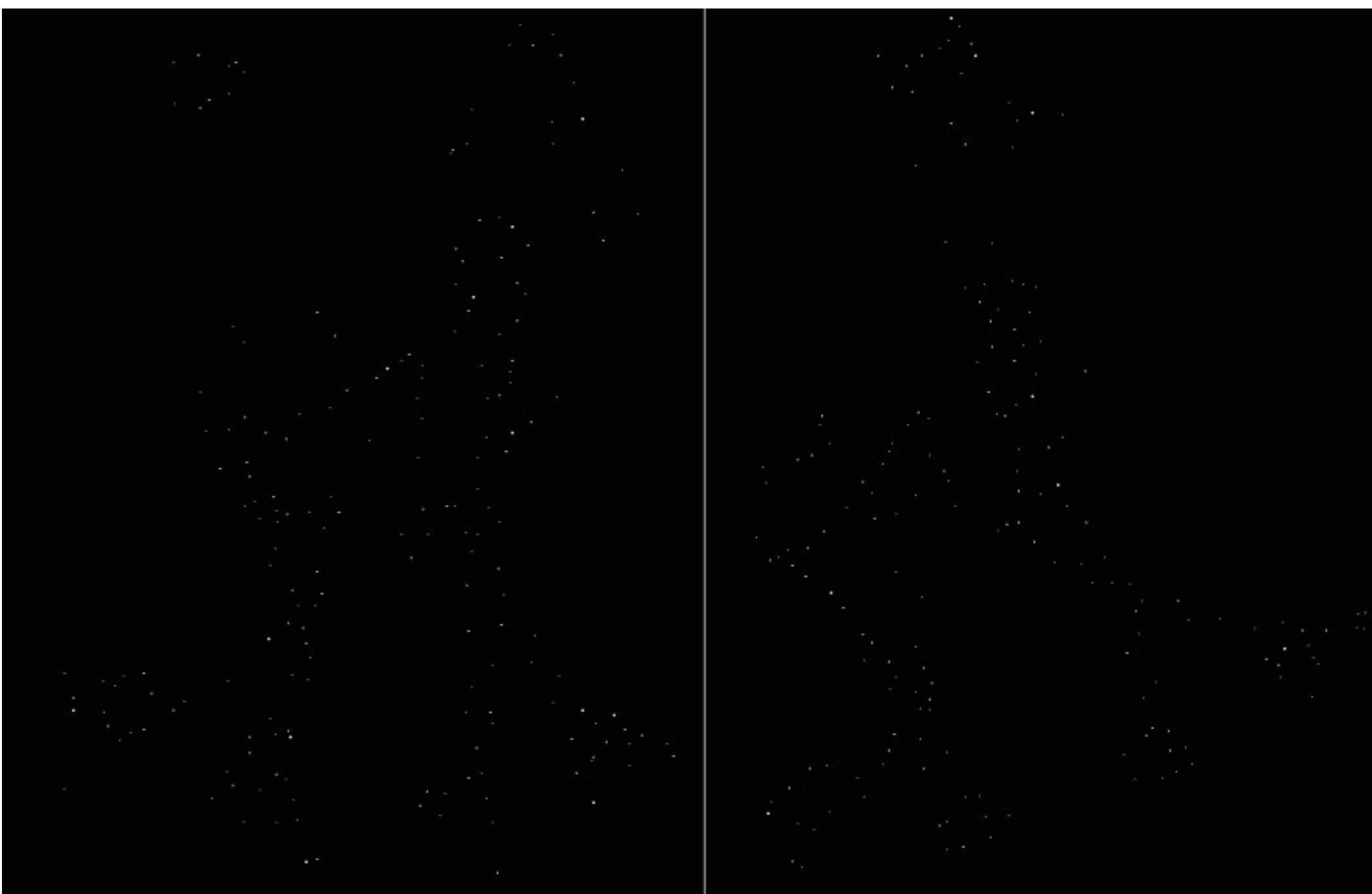




Thresholded corner response

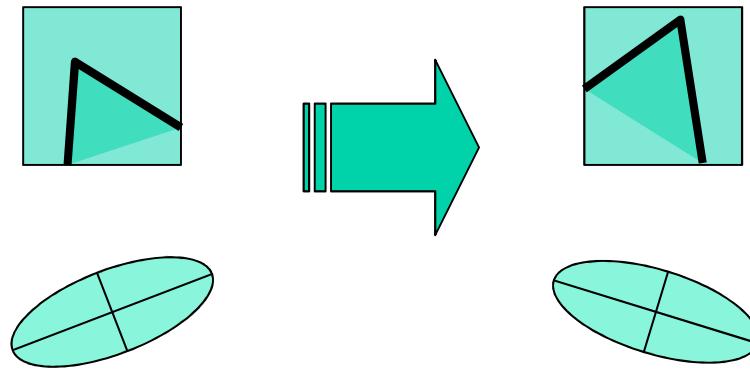


## Non-maximal suppression





Harris corner response is invariant to rotation



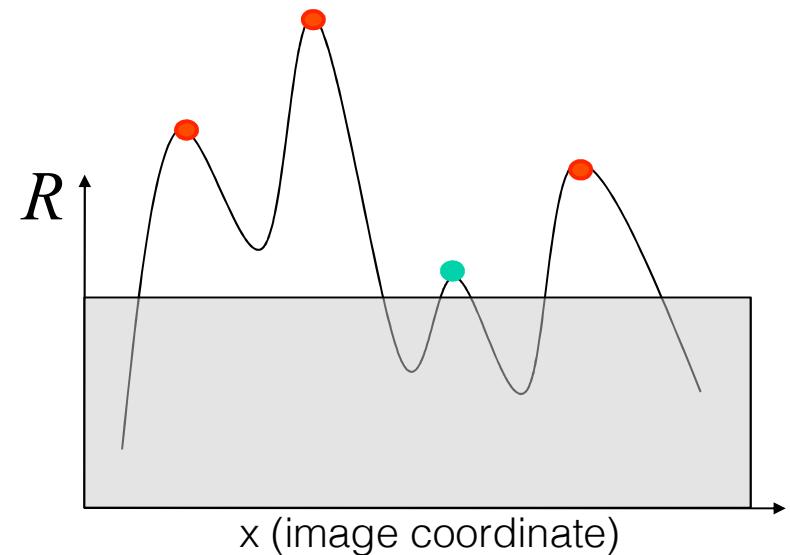
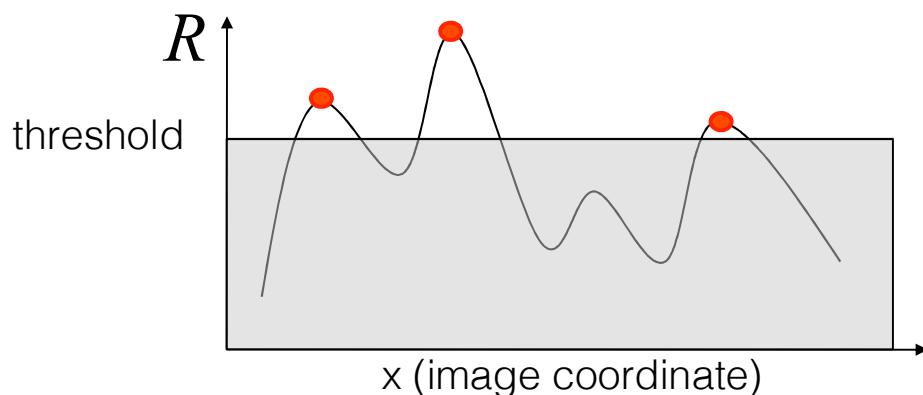
Ellipse rotates but its shape  
**(eigenvalues)** remains the same

**Corner response R is invariant to image rotation**

# Harris corner response is invariant to intensity changes

Partial invariance to *affine intensity* change

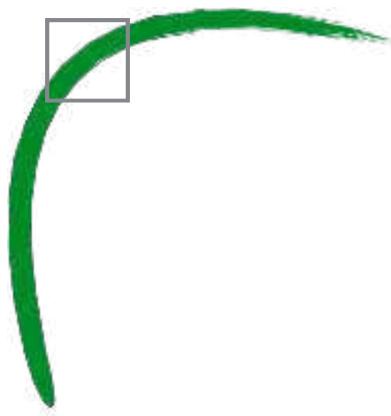
- Only derivatives are used => invariance to intensity shift  $I \rightarrow I + b$
- Intensity scale:  $I \rightarrow a I$



The Harris detector is not invariant to changes in ...

The Harris corner detector is not  
invariant to scale

edge!



corner!



# Multi-scale detection

How can we make a feature detector scale-invariant?

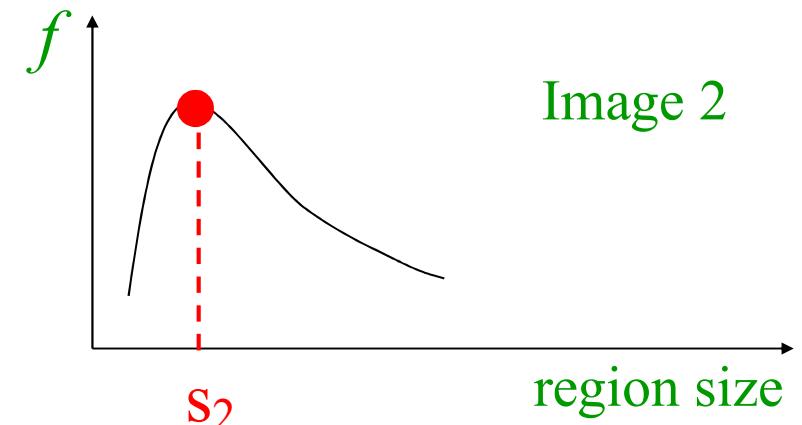
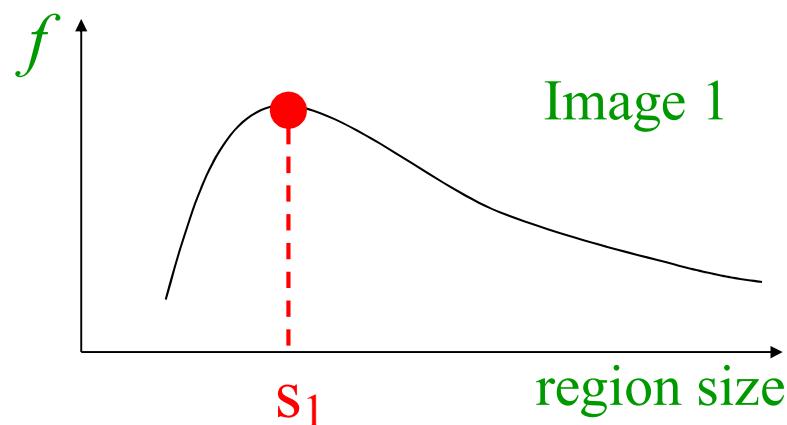
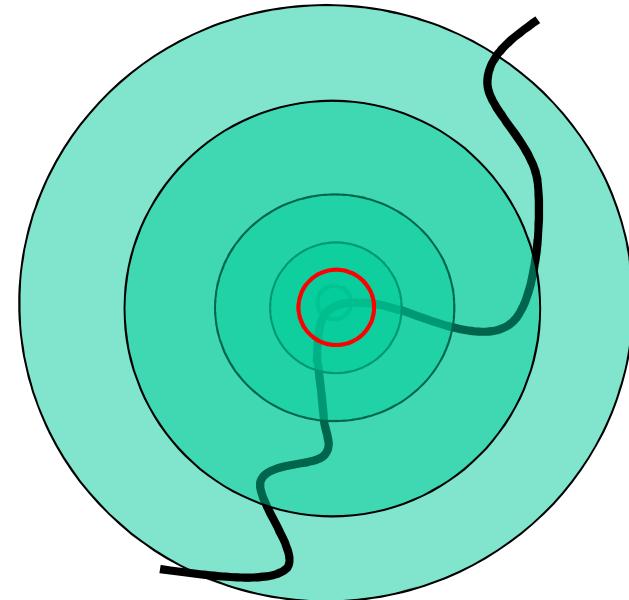
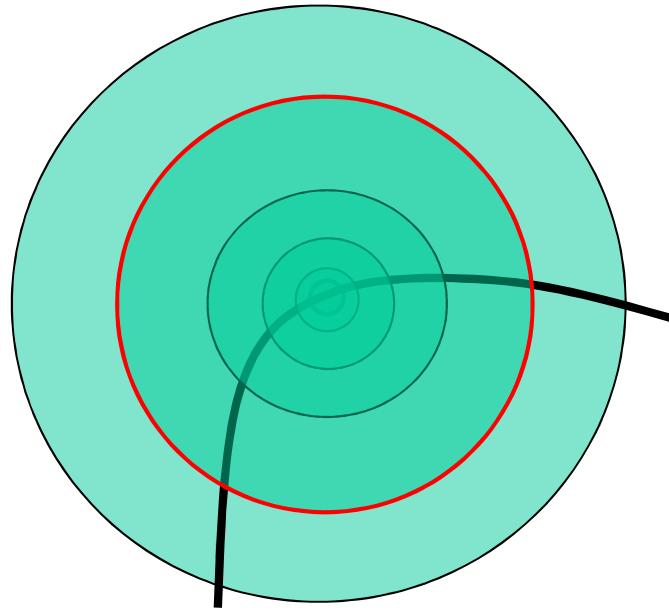
How can we automatically select the scale?

# Multi-scale blob detection



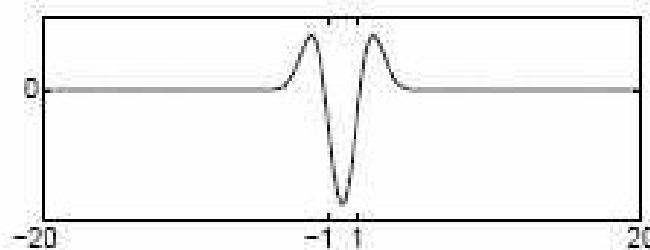
Intuitively...

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

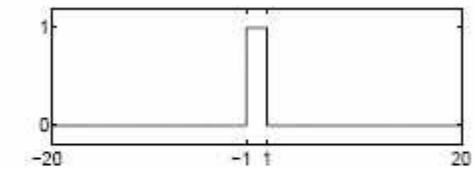
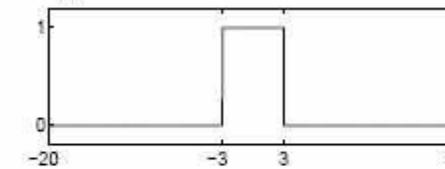
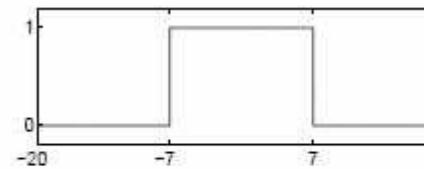
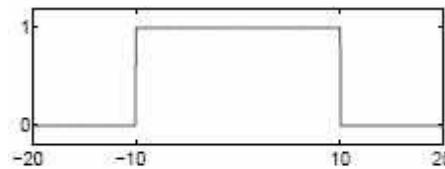


Formally...

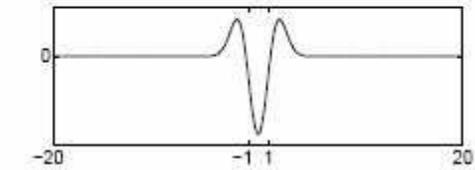
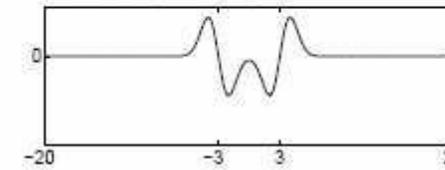
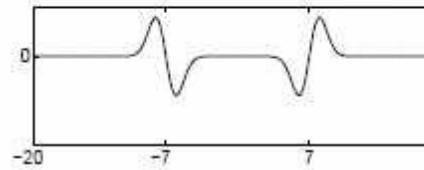
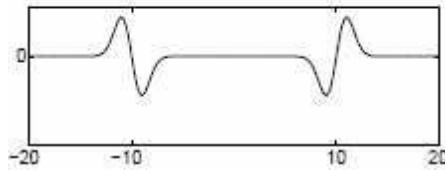
Laplacian filter



Original signal

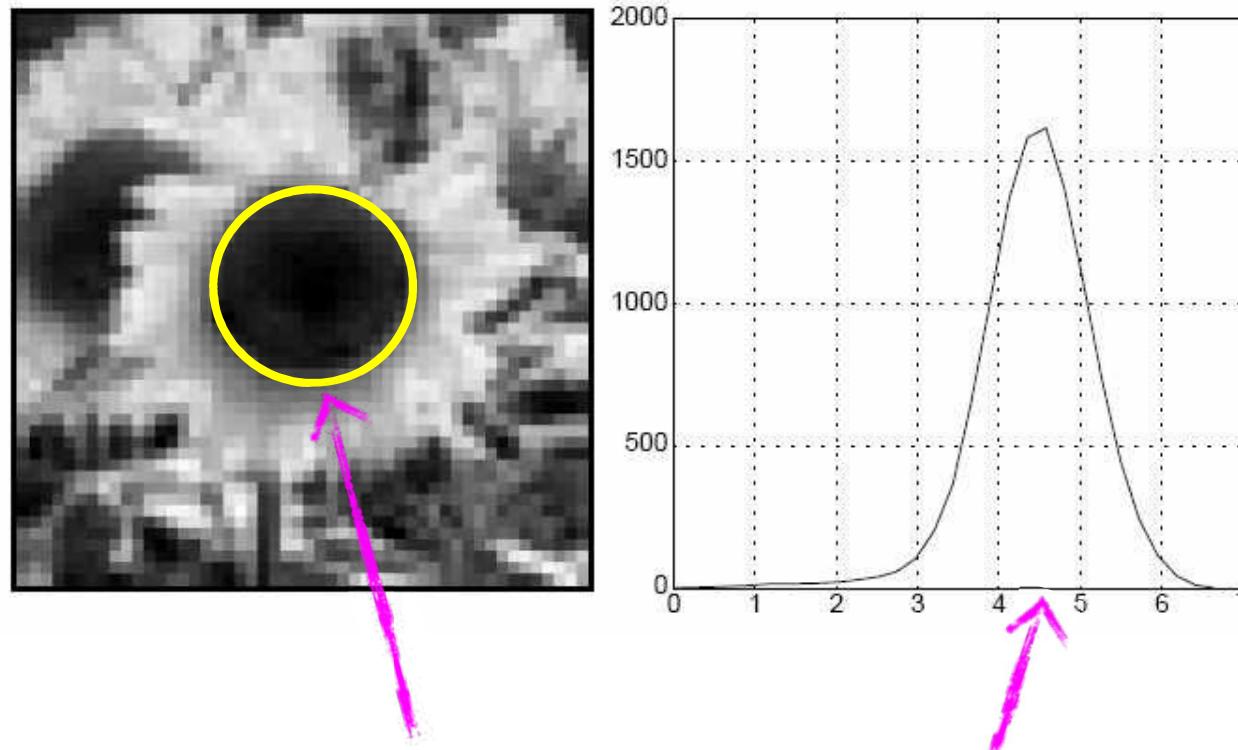


Convolved with Laplacian ( $\sigma = 1$ )



Highest response when the signal has the same **characteristic scale** as the filter

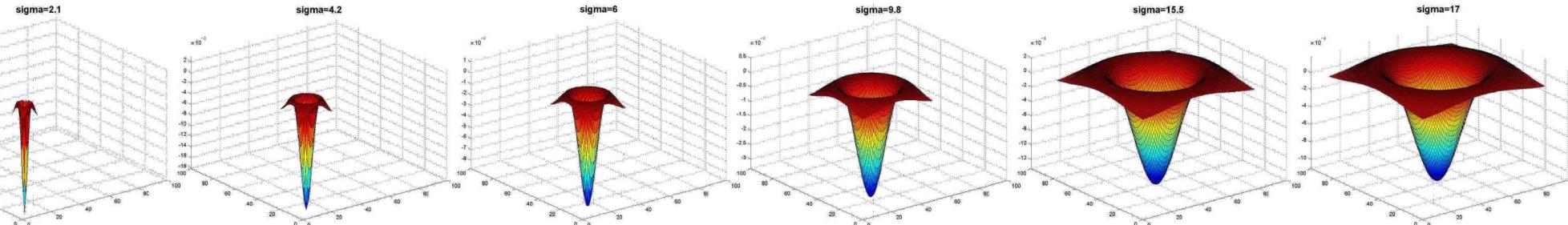
characteristic scale - the scale that produces peak filter response



characteristic scale

**we need to search over characteristic scales**

# What happens if you apply different Laplacian filters?

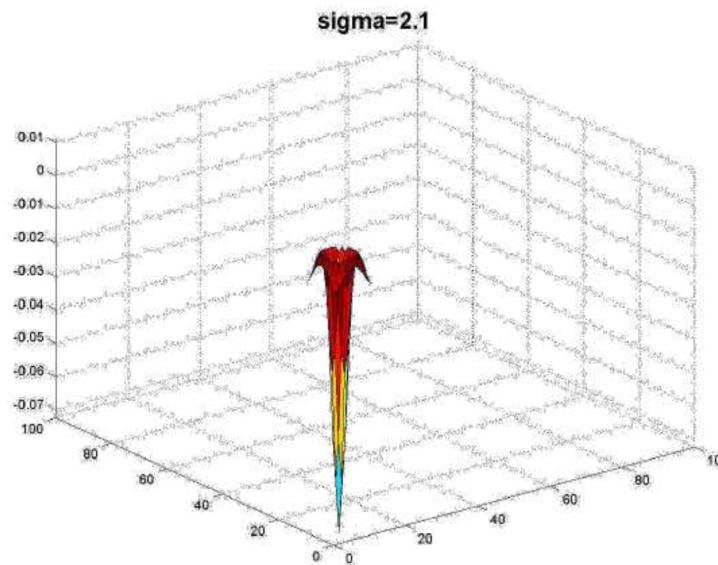


Full size

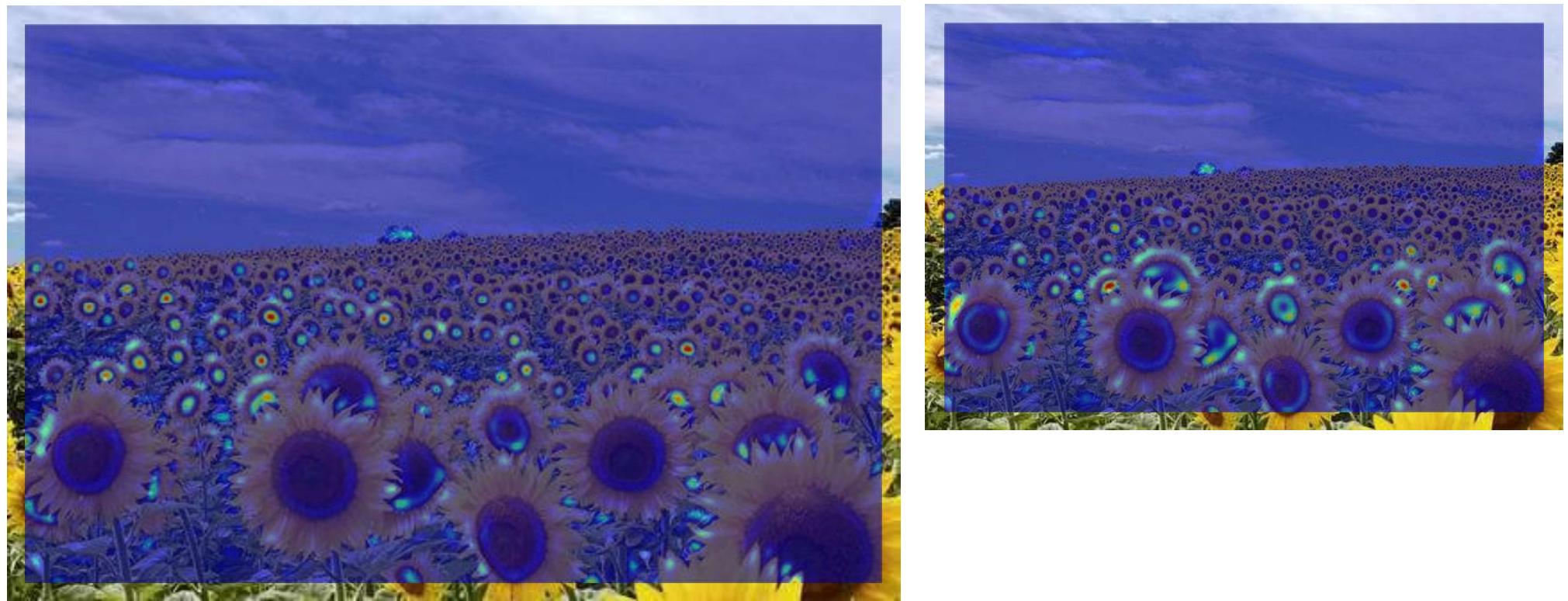
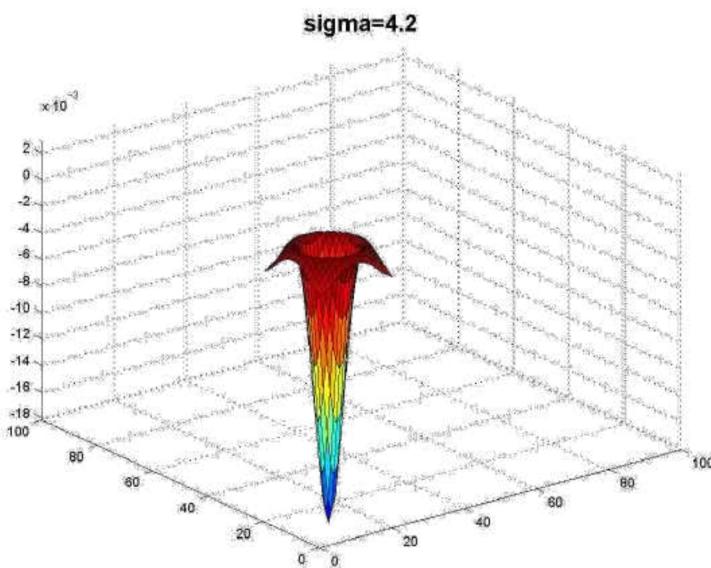


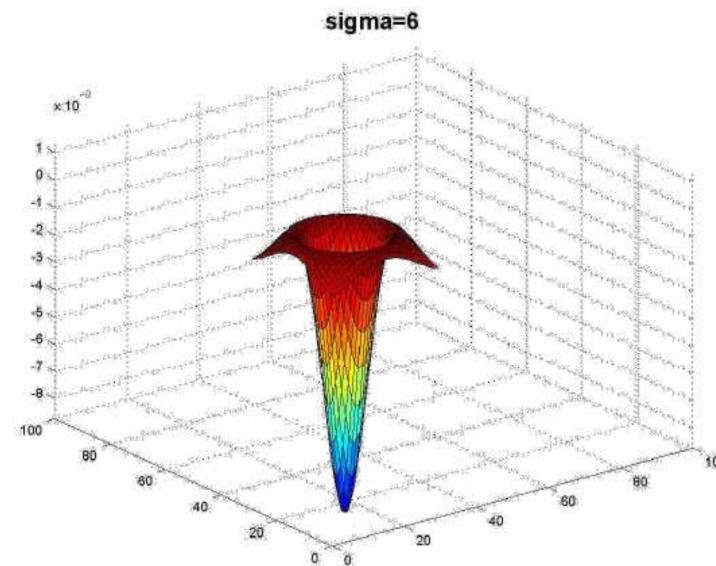
3/4 size

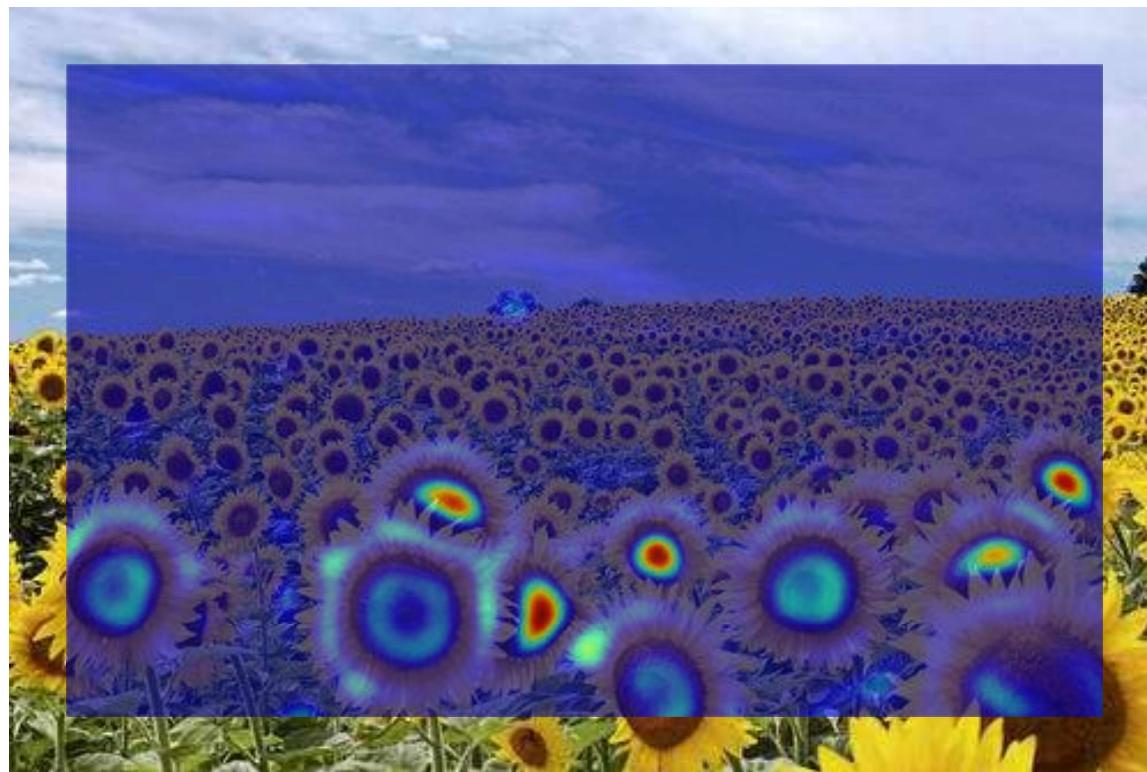
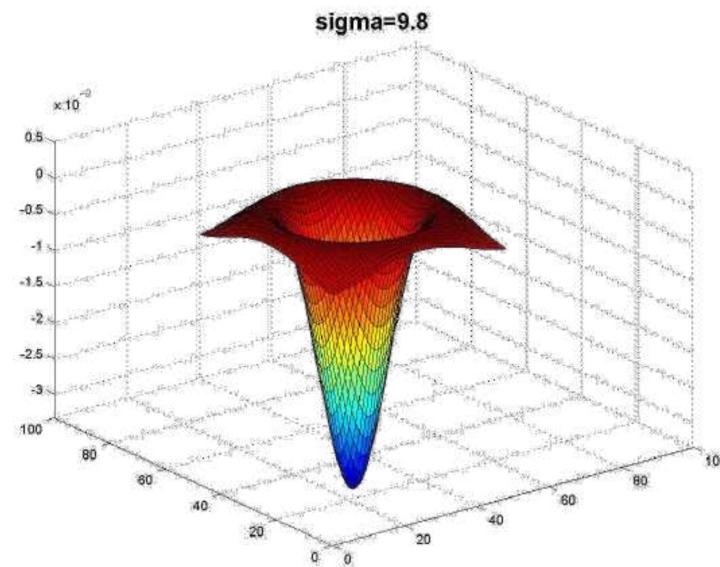


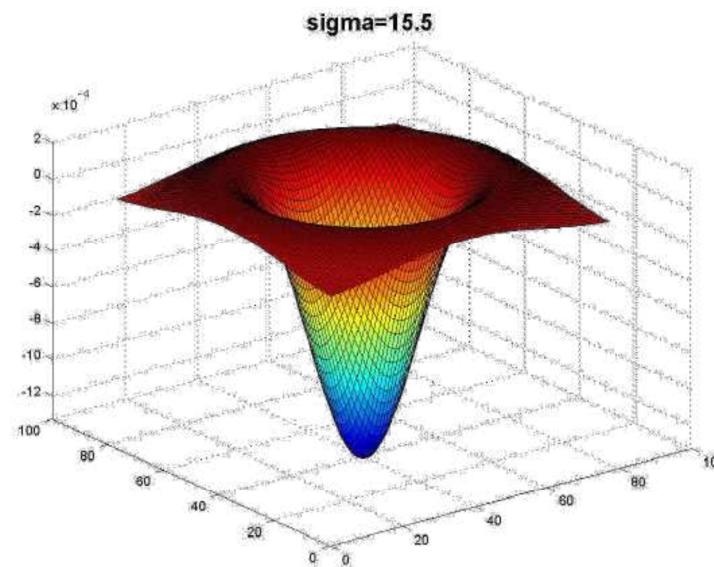


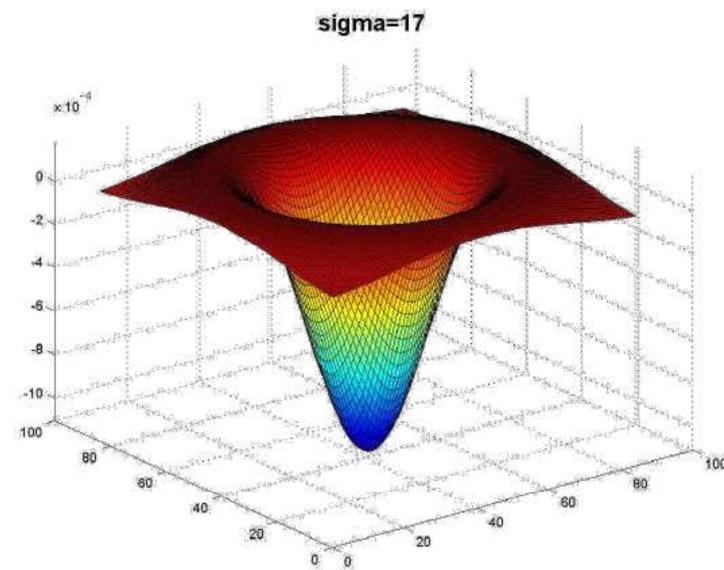
**jet** color scale  
blue: low, red: high











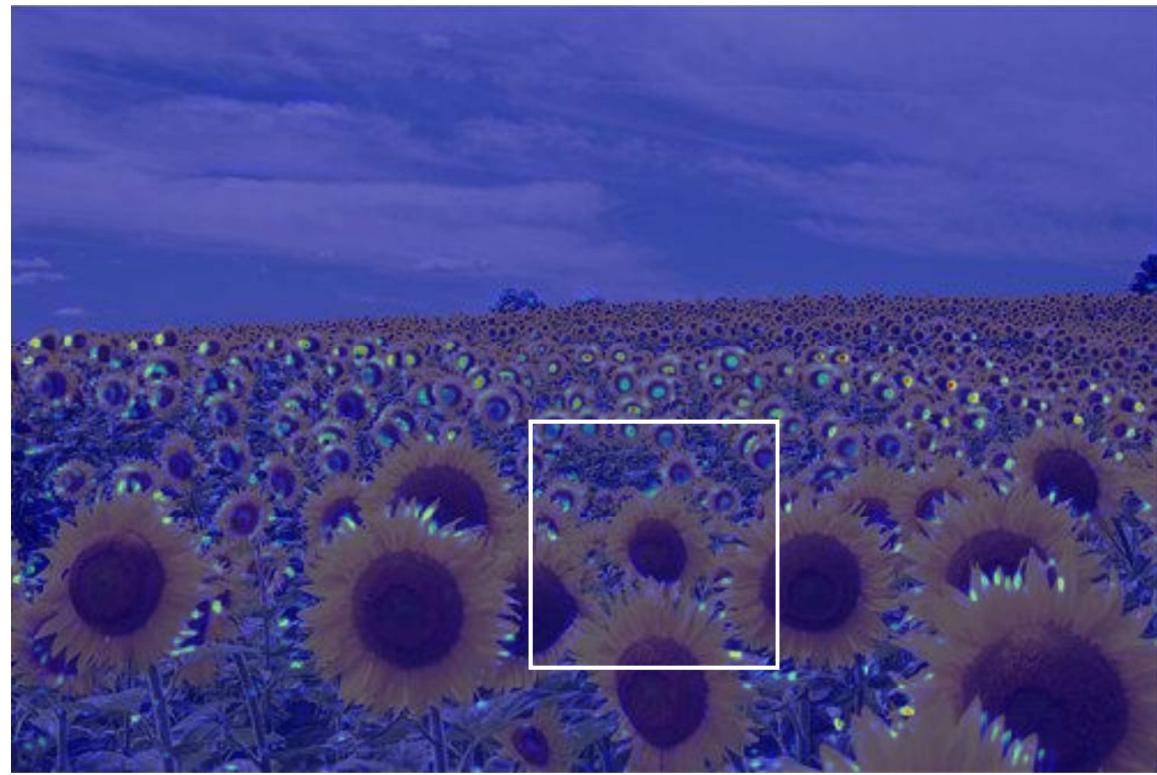
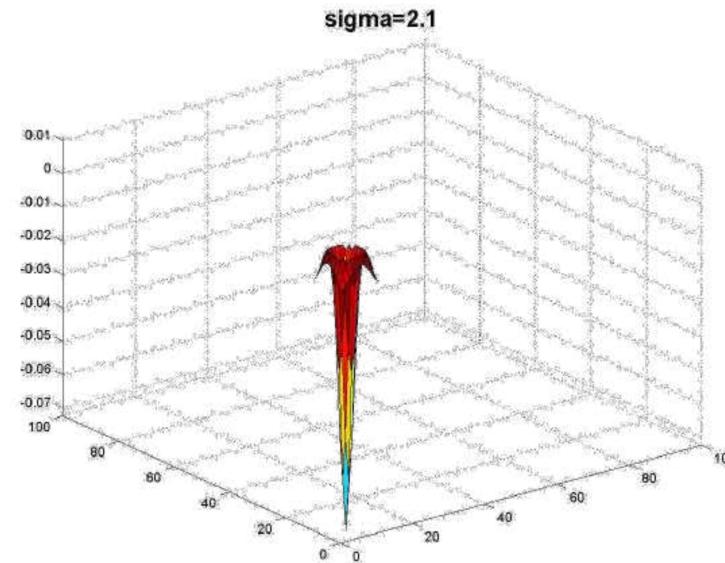
What happened when you applied different Laplacian filters?

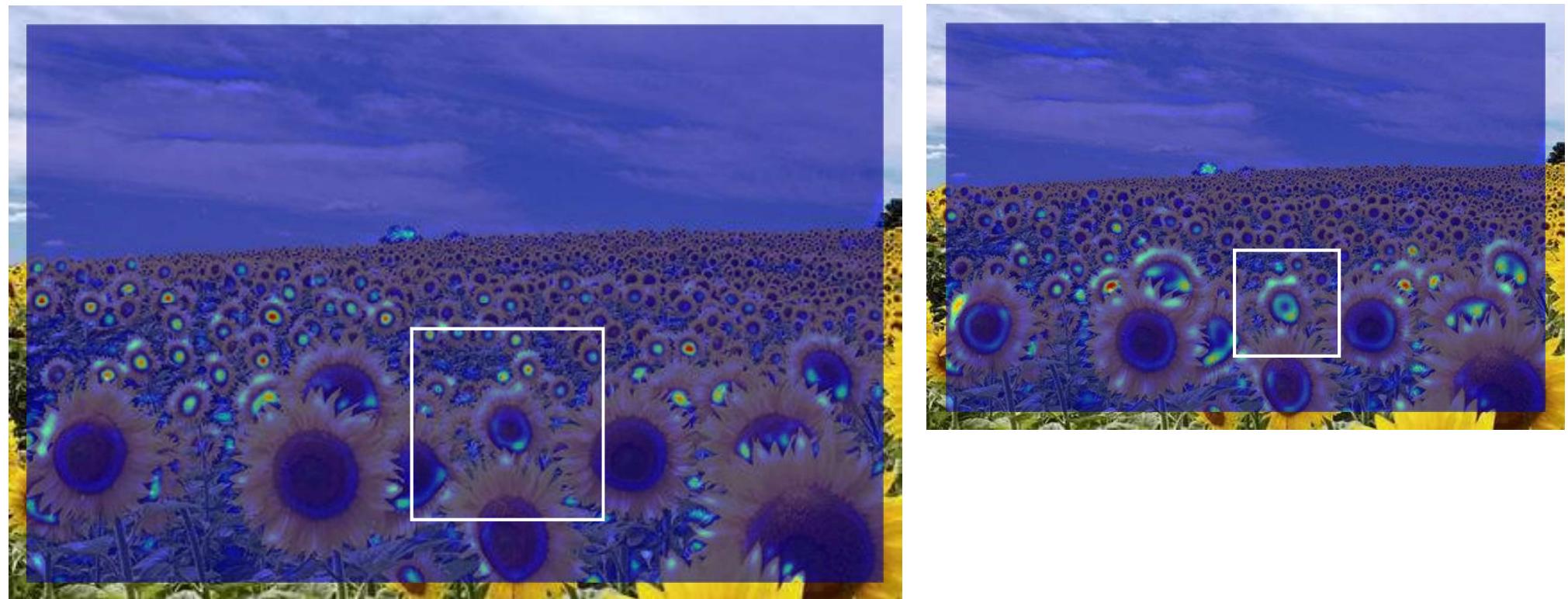
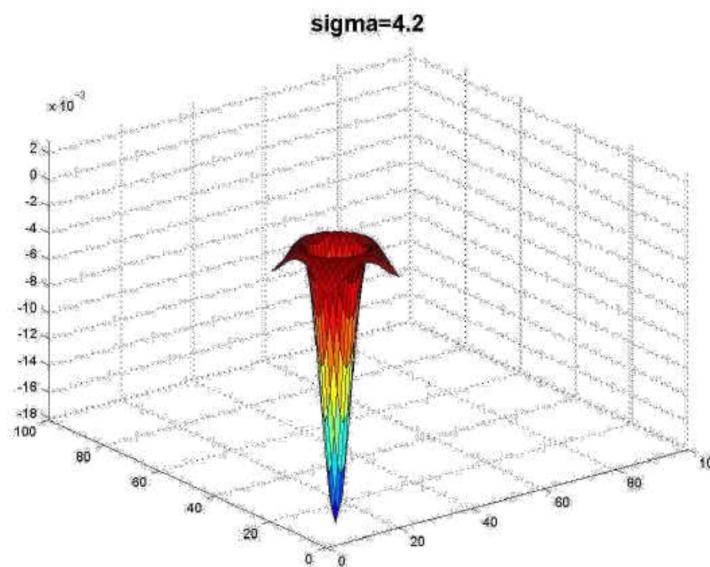
Full size

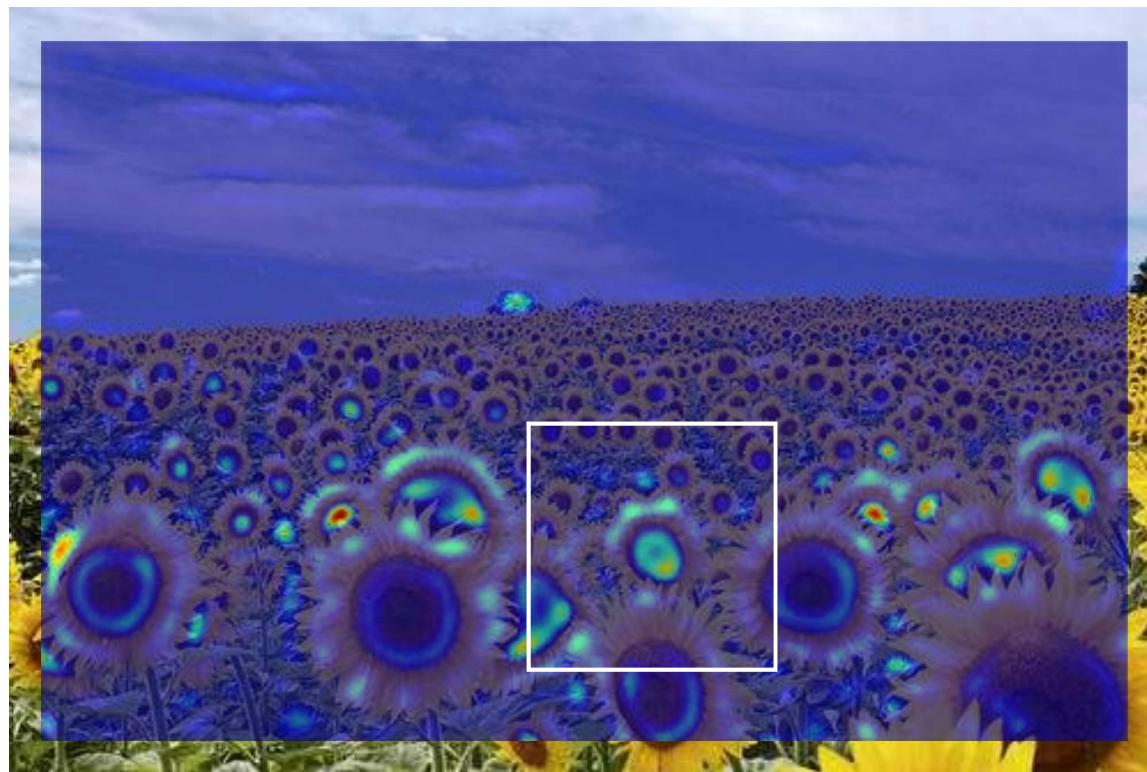
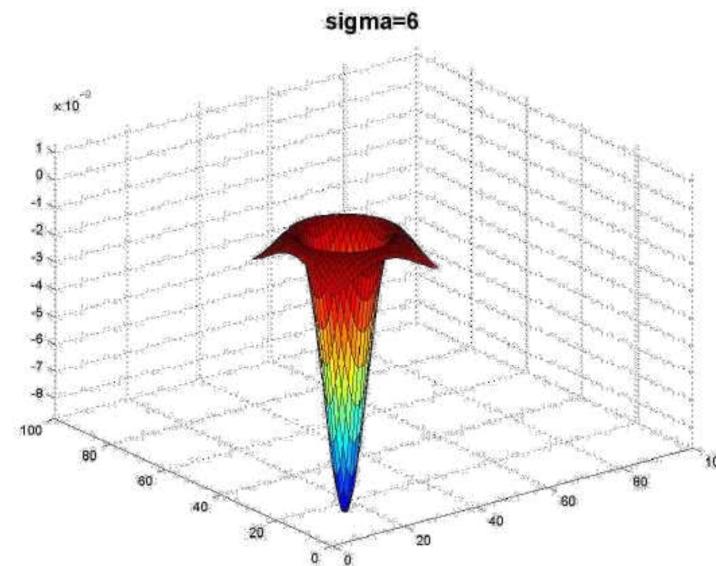


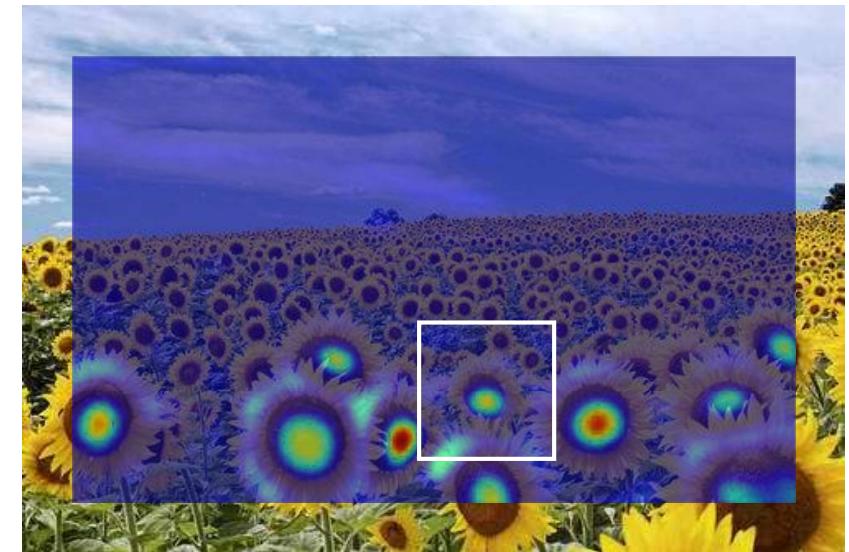
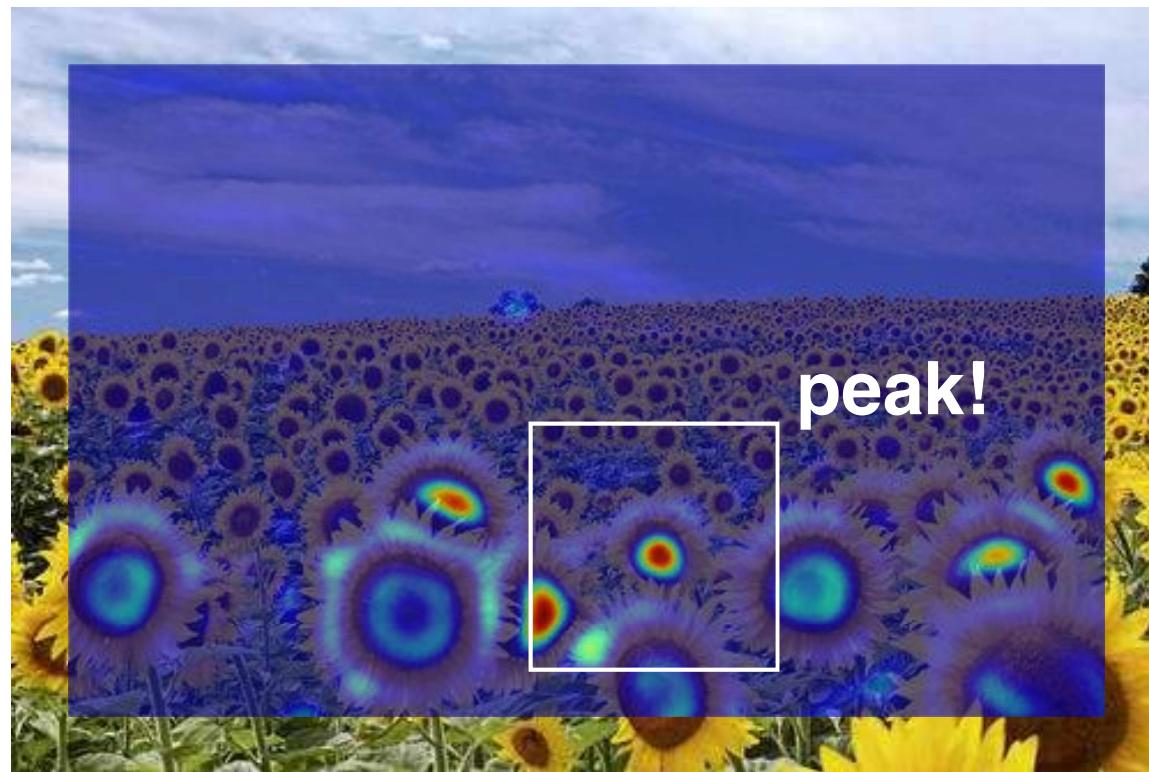
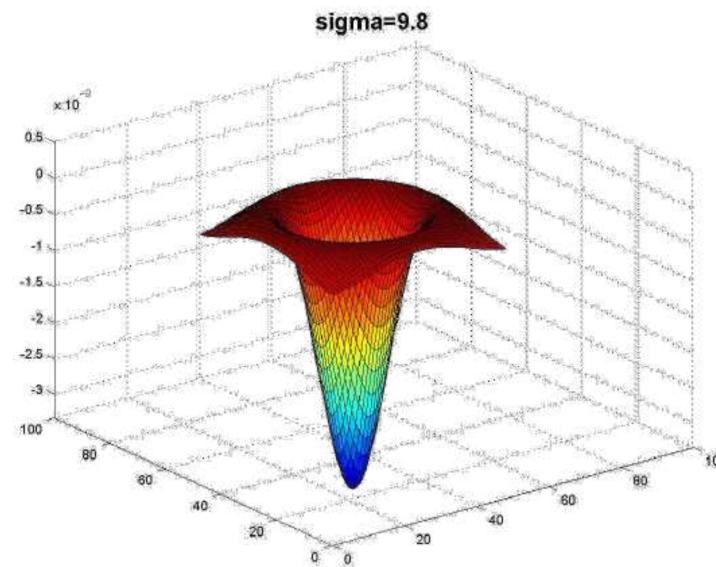
3/4 size

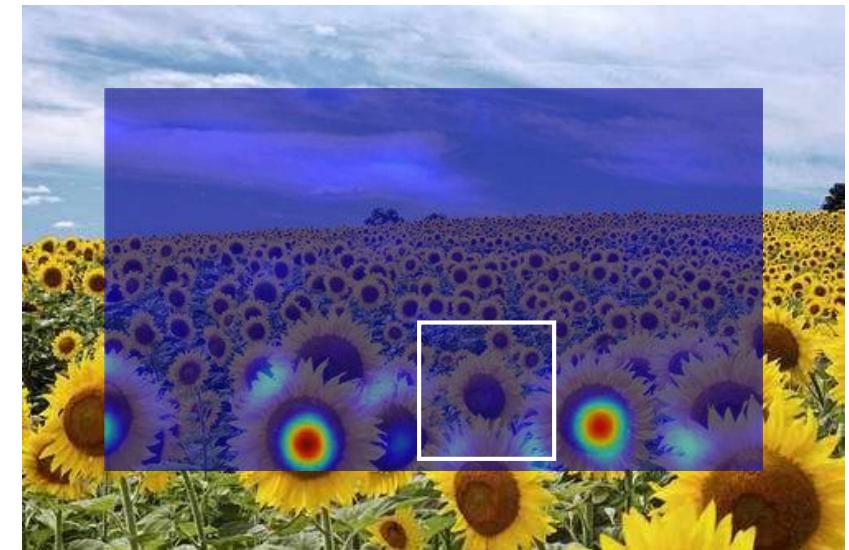
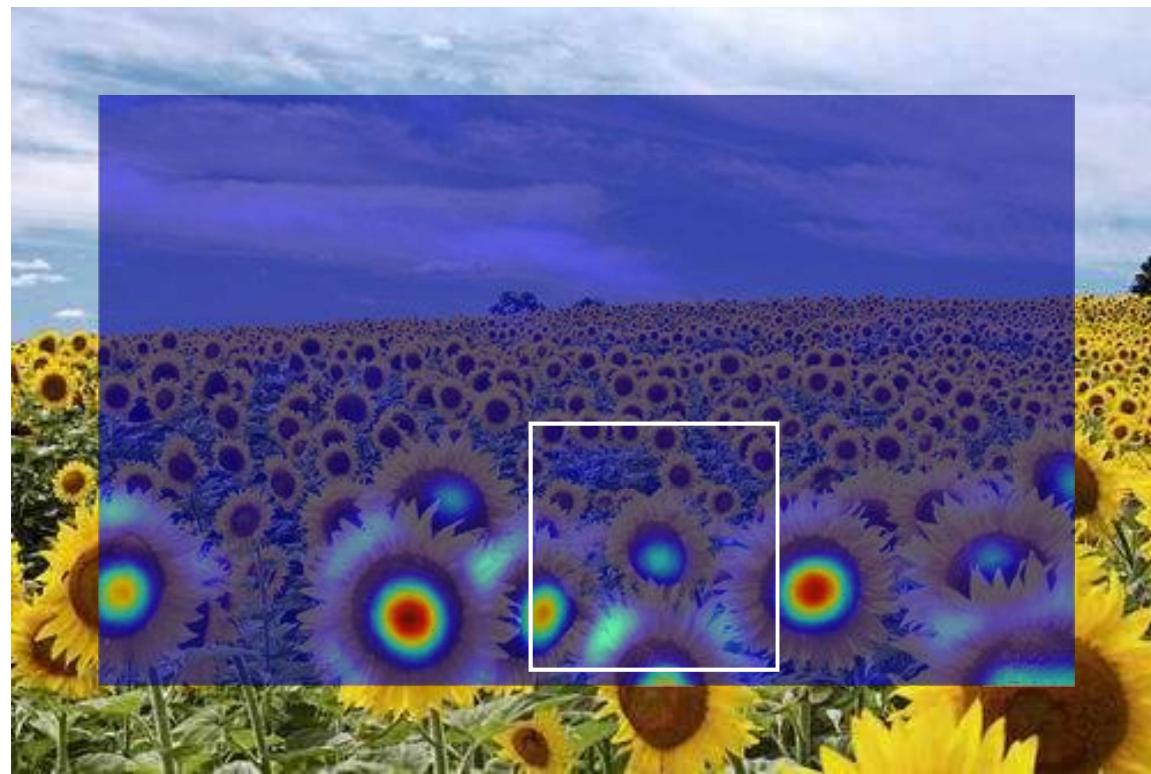
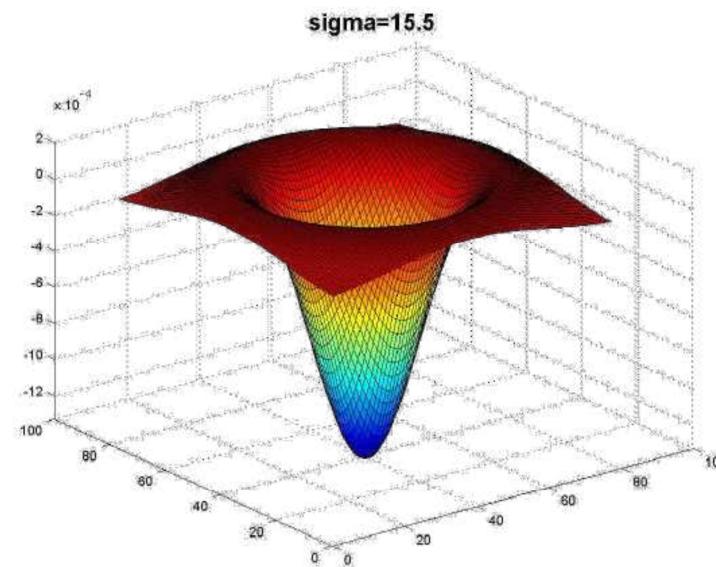




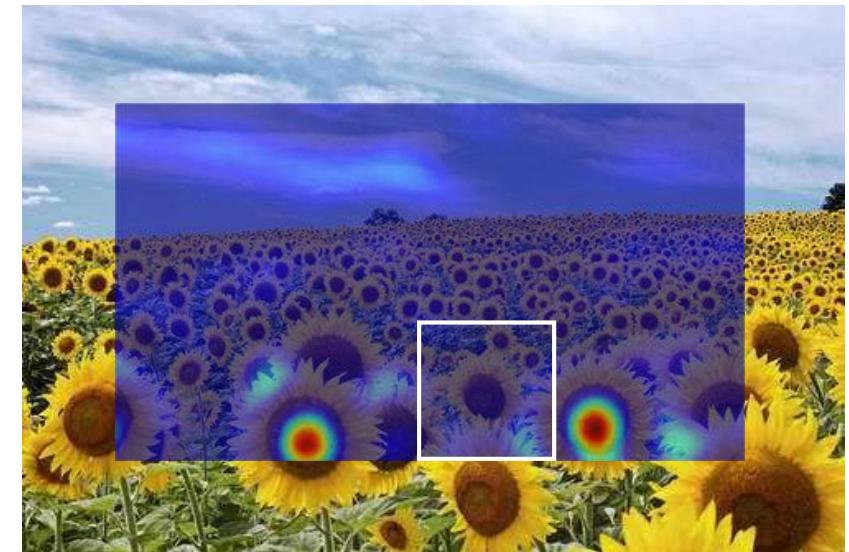
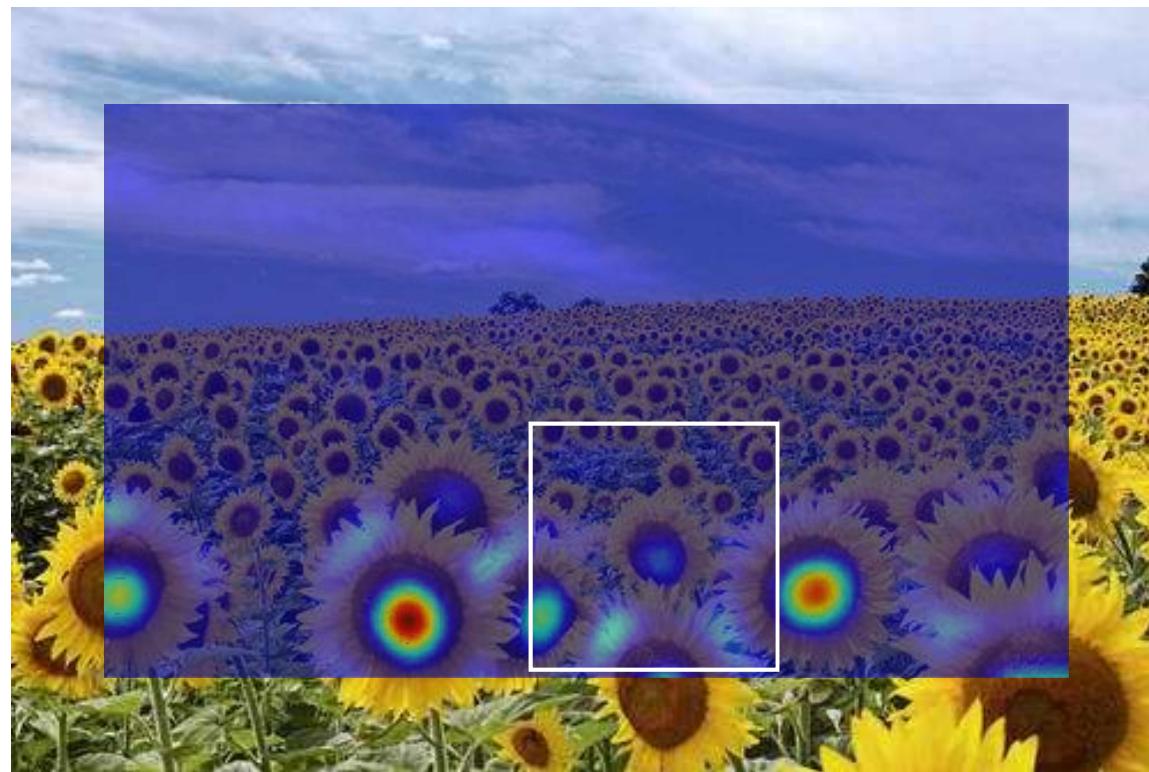
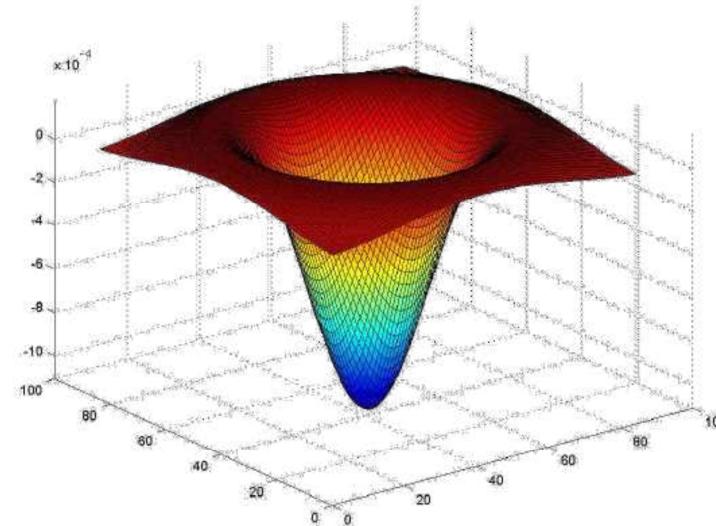








**sigma=17**



What happened when you applied different Laplacian filters?

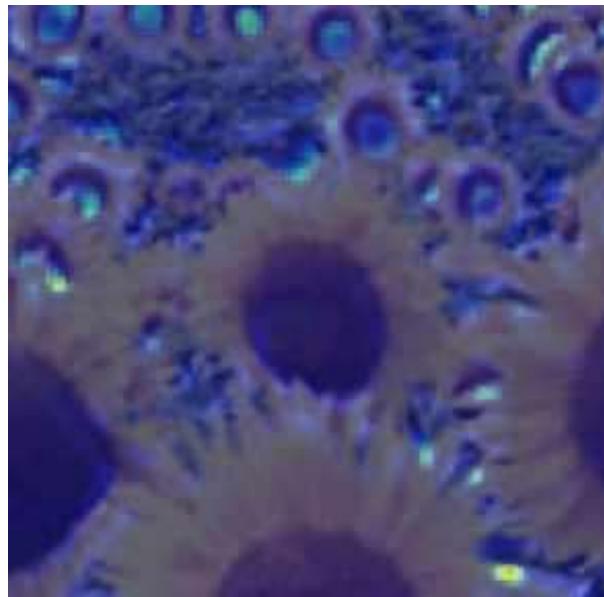
Full size



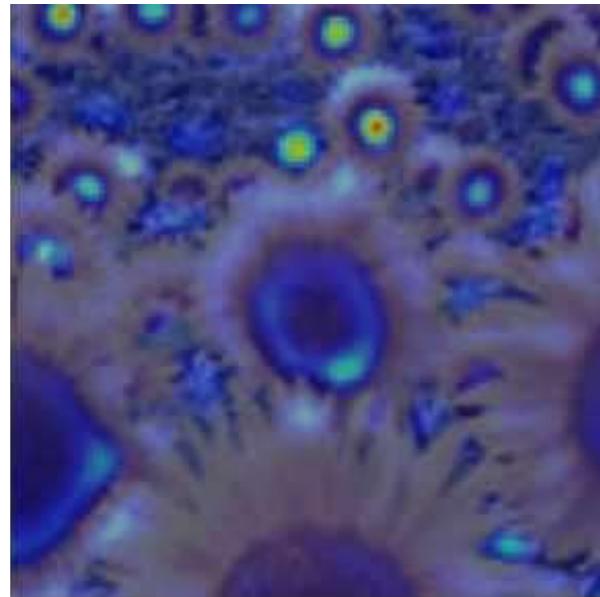
3/4 size



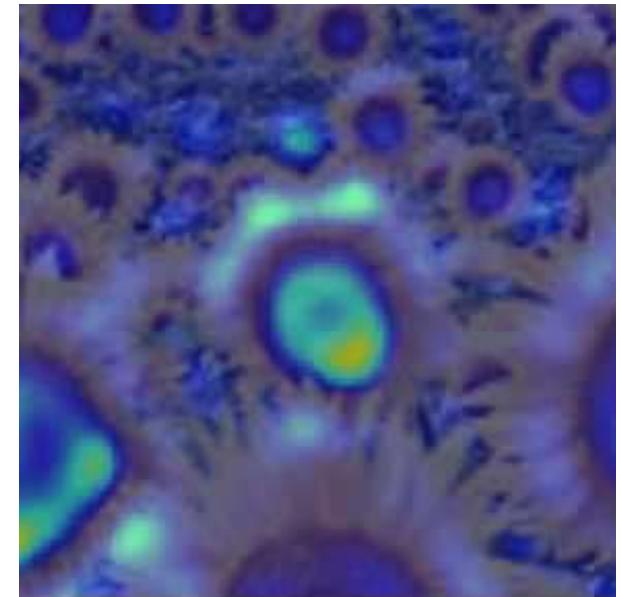
2.1



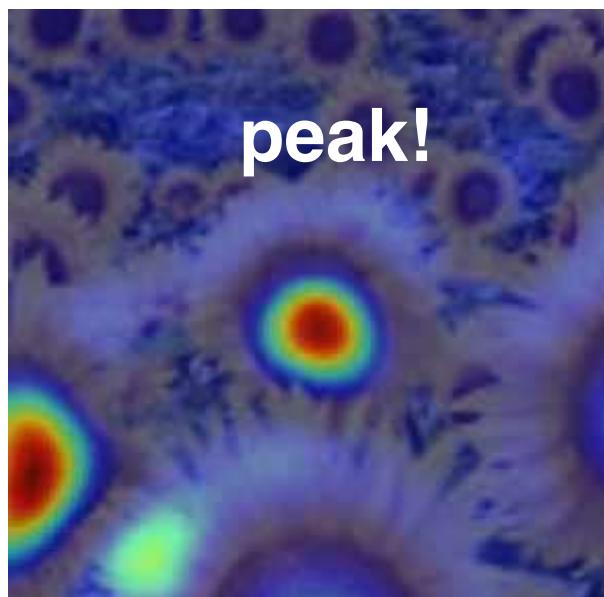
4.2



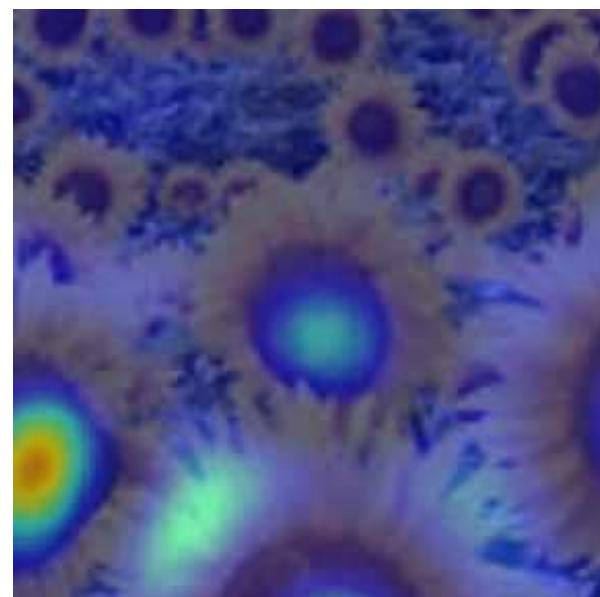
6.0



9.8



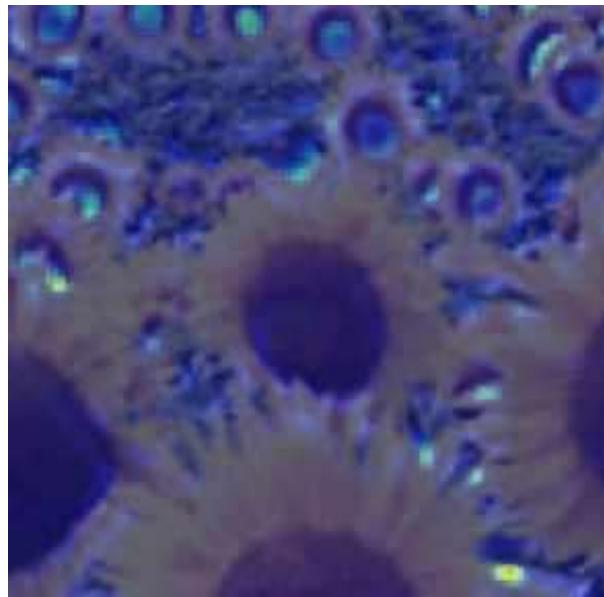
15.5



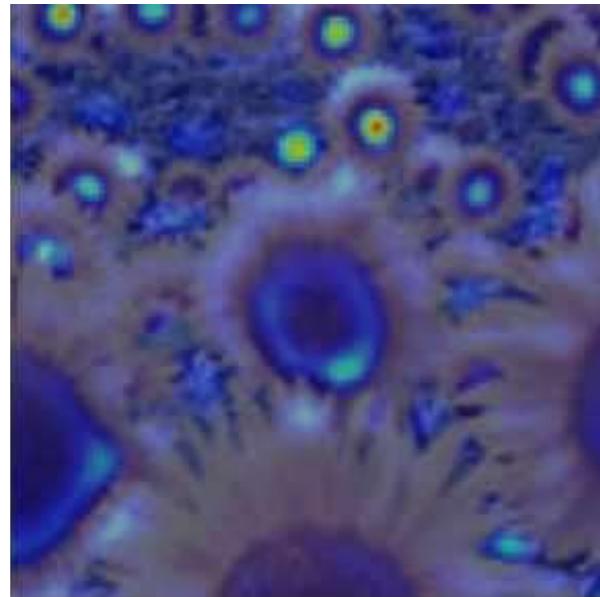
17.0



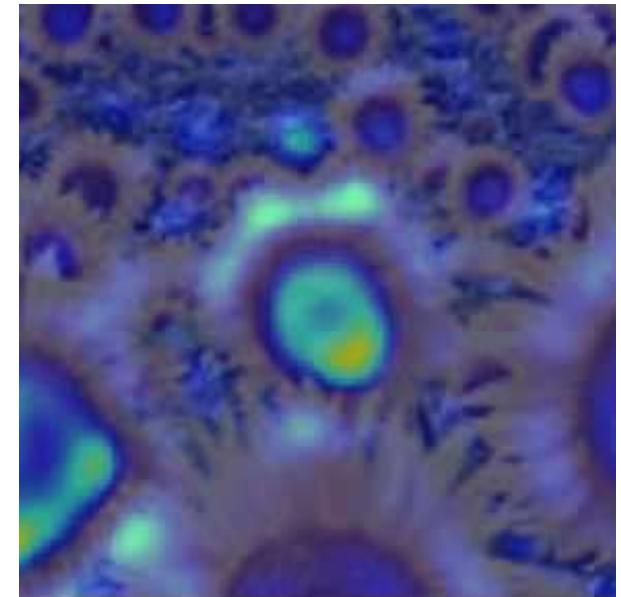
2.1



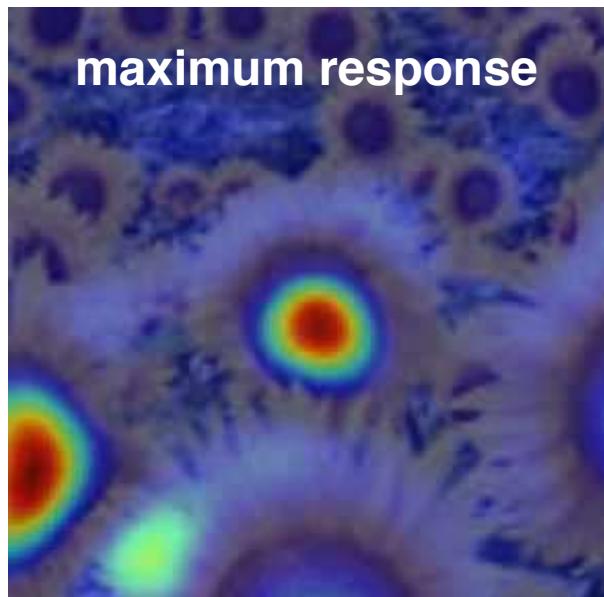
4.2



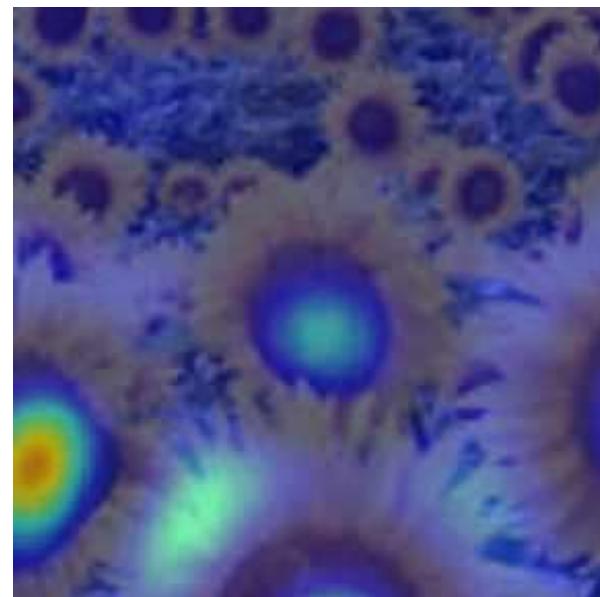
6.0



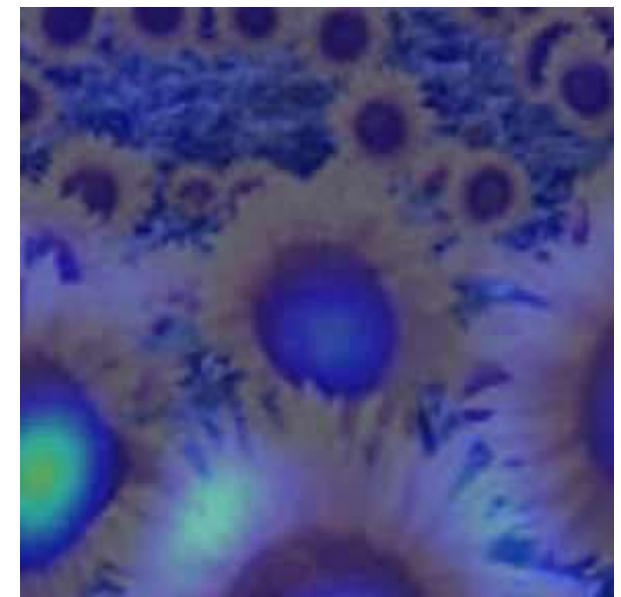
9.8



15.5

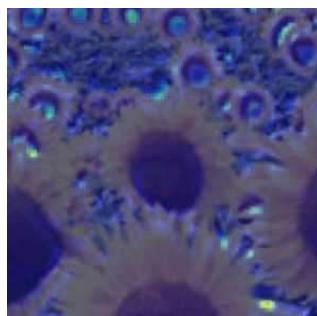


17.0

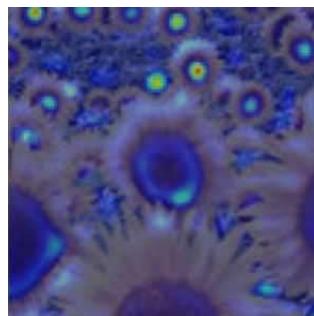


# optimal scale

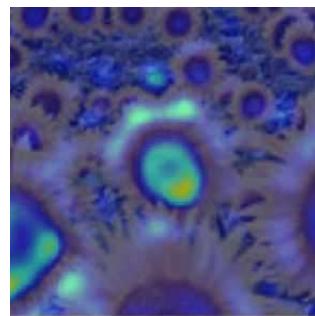
2.1



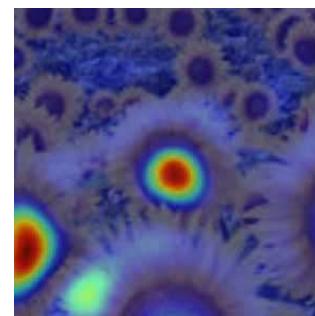
4.2



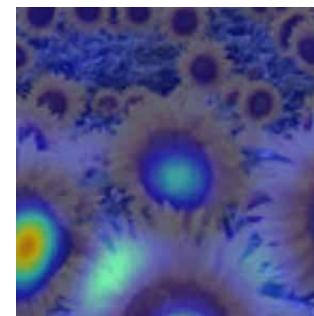
6.0



9.8



15.5

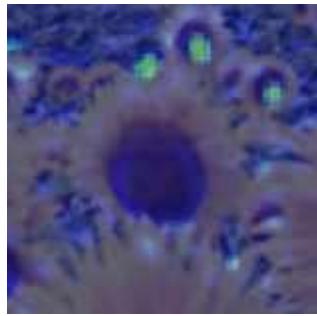


17.0

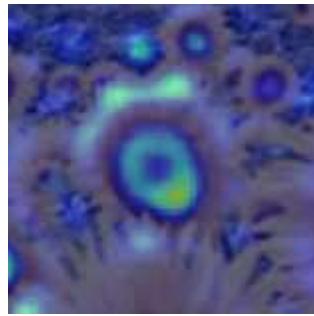


Full size image

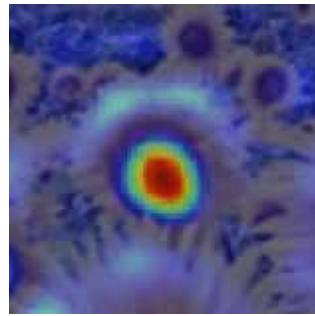
2.1



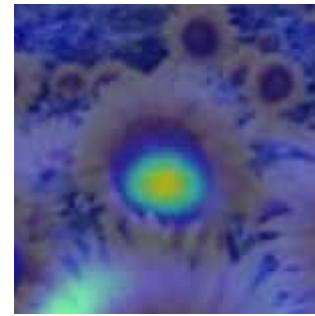
4.2



6.0



9.8



15.5



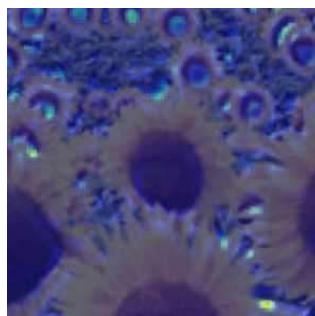
17.0



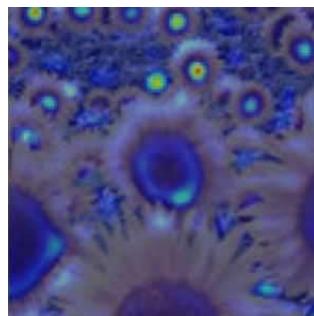
3/4 size image

# optimal scale

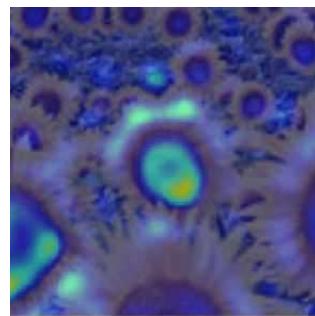
2.1



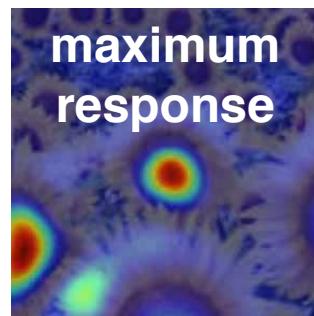
4.2



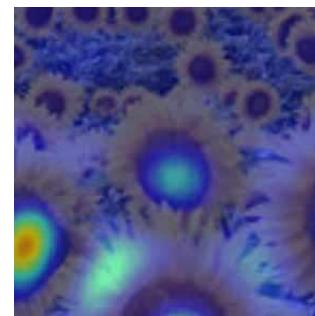
6.0



9.8



15.5

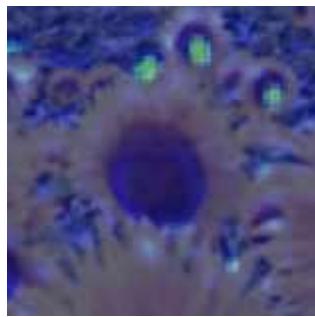


17.0

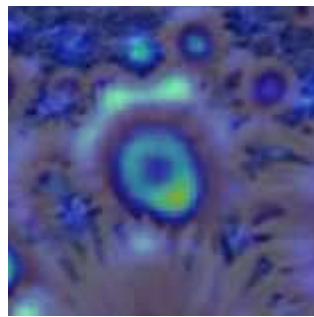


Full size image

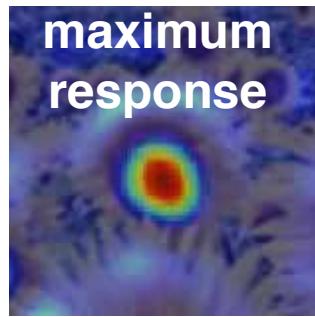
2.1



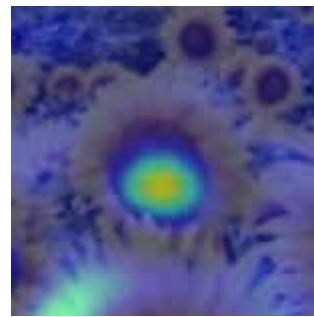
4.2



6.0



9.8



15.5

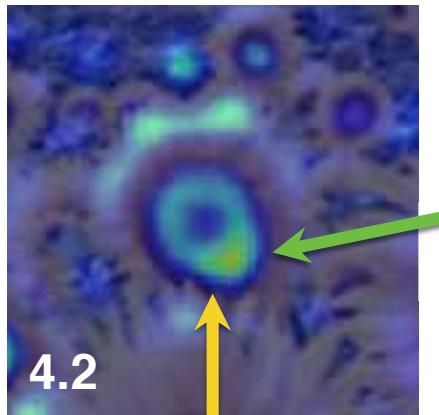


17.0

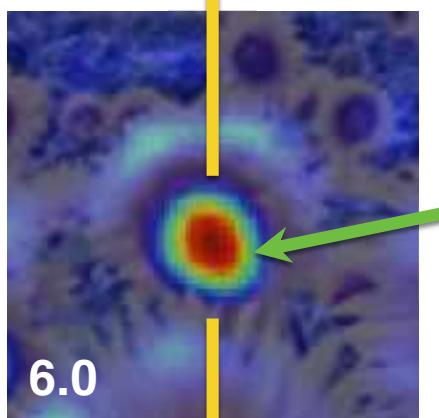


3/4 size image

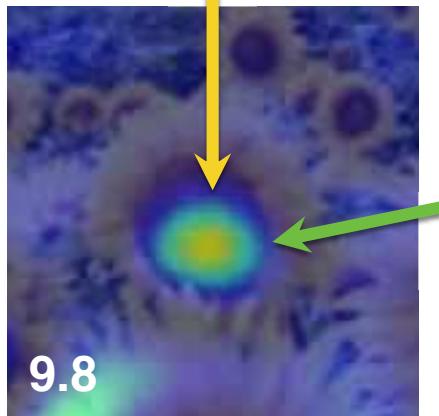
cross-scale maximum



local maximum



local maximum



local maximum

How would you  
implement scale  
selection?

# 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  $(x, y, s)$



