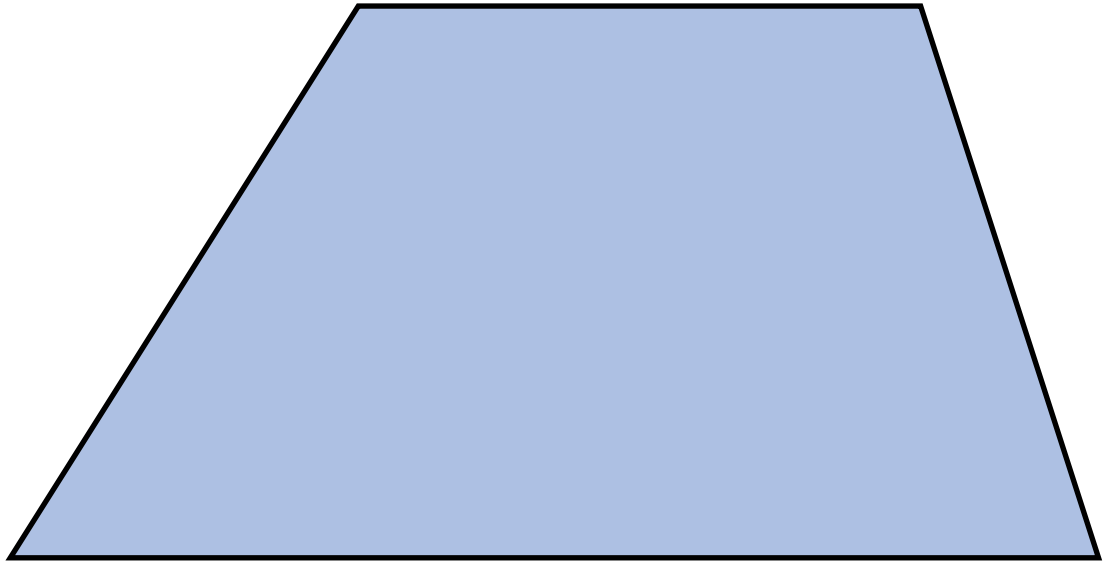
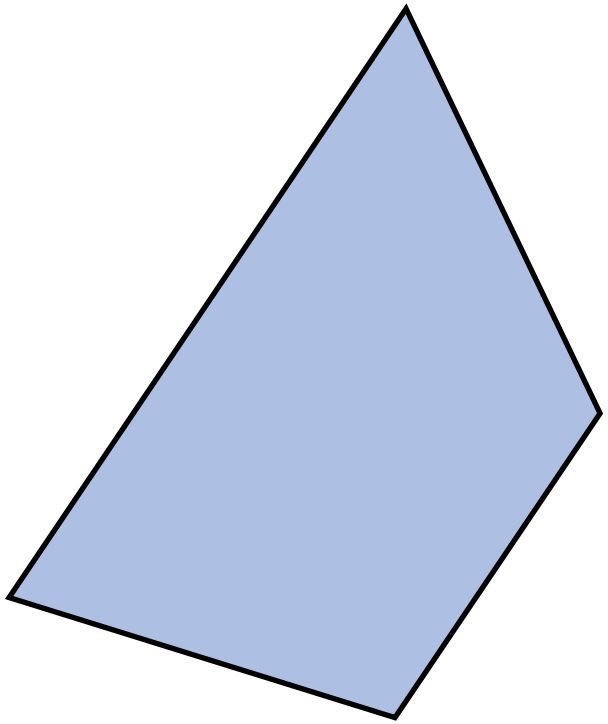
Corner Detection





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

What type of feature would you select?

a corner

Why detect corners?

Image alignment (homography, fundamental matrix)

3D reconstruction

Motion tracking

Object recognition

Indexing and database retrieval

Robot navigation

And so on….

Location Recognition

Robot Localization

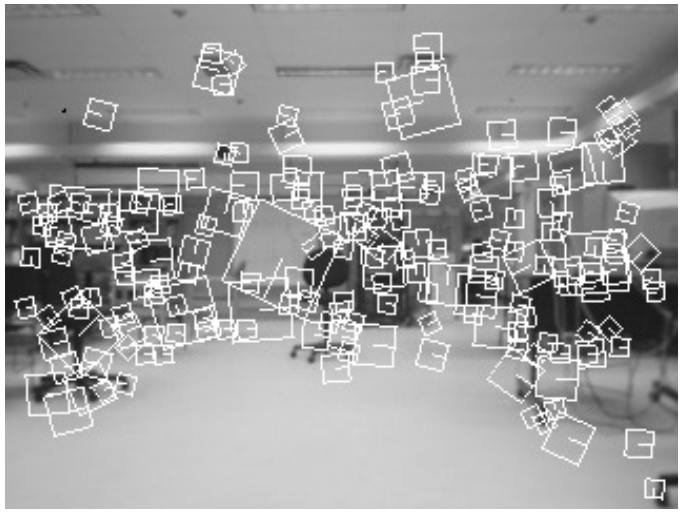
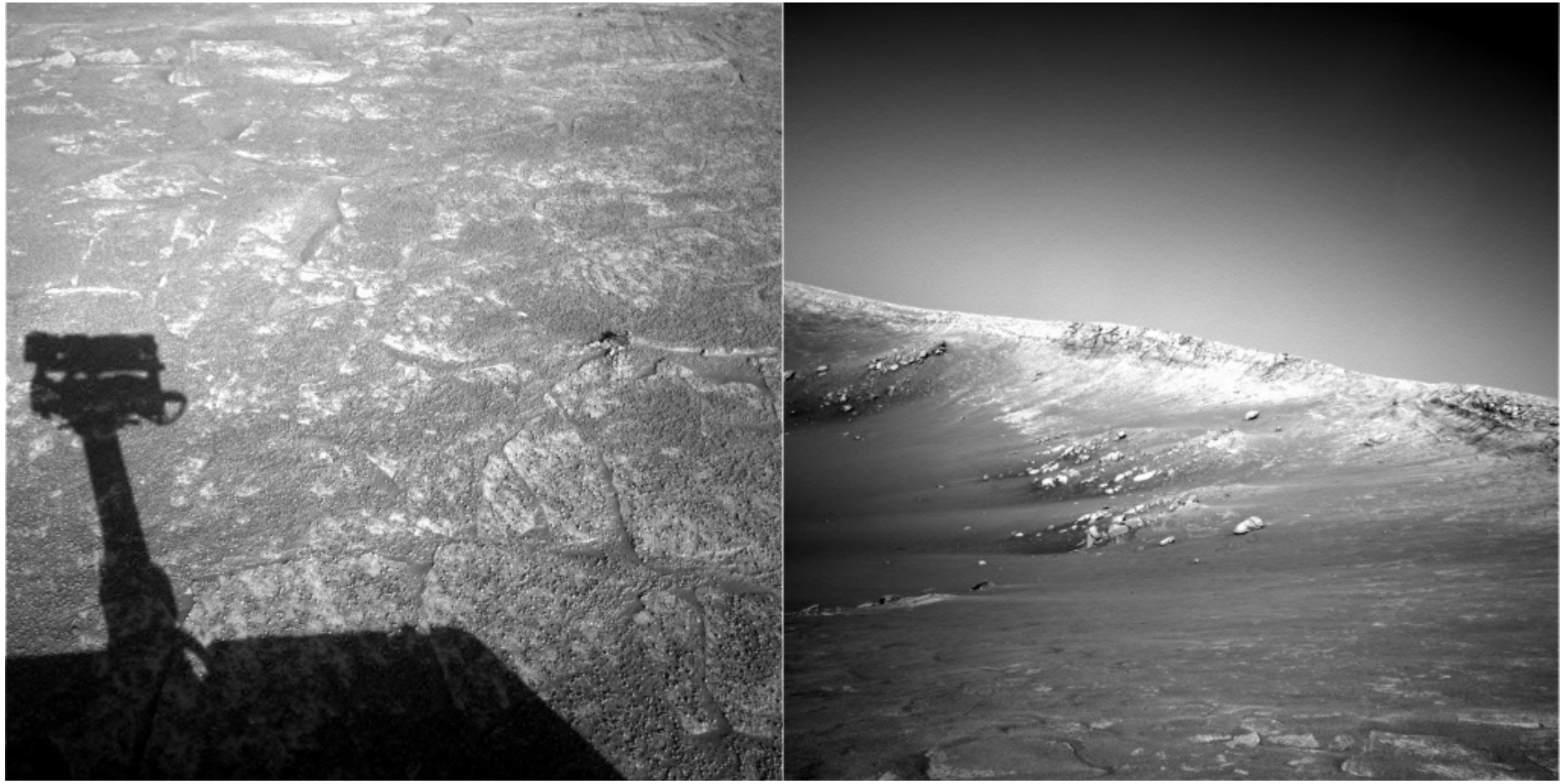
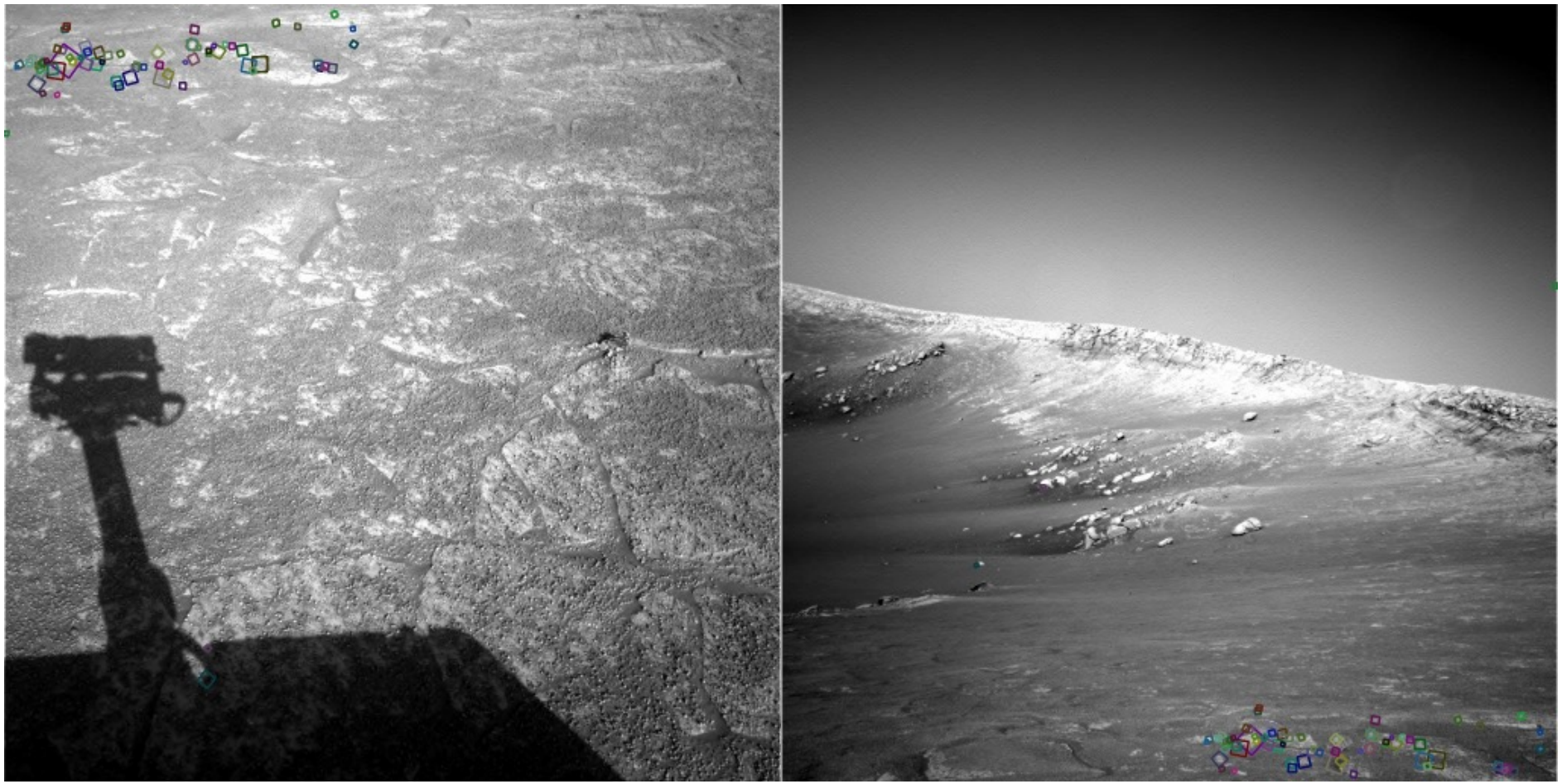


Image matching



NASA Mars Rover images

Where are the corresponding points?

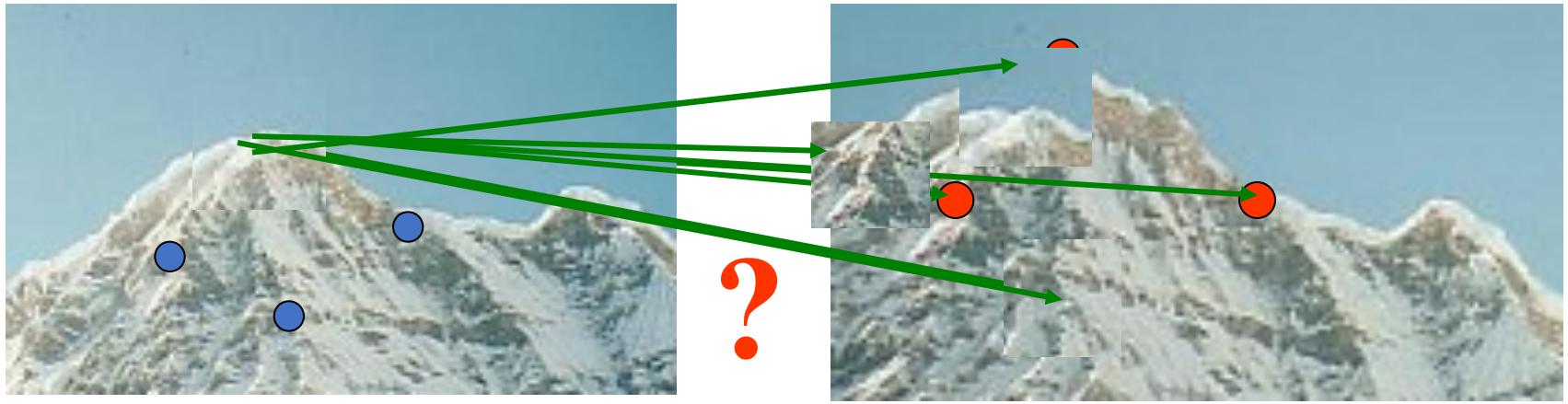


Example: Panorama Stitching

We have two images –
how do we overlay them?

Goal: Distinctiveness

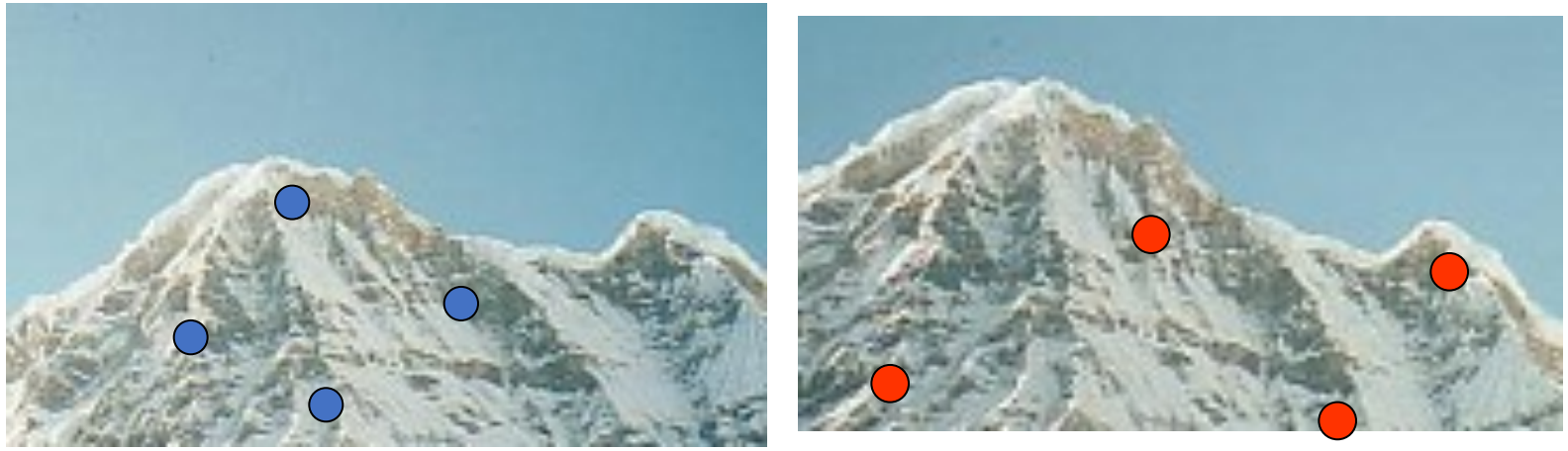
We want to be able to reliably determine which point goes with which.



May be difficult in structured environments
with repeated elements

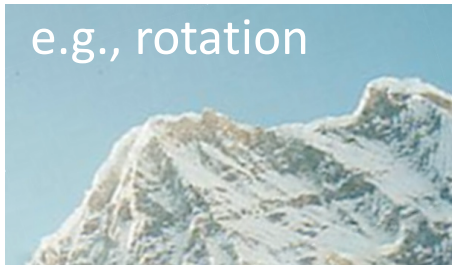
Goal: Repeatability

We want to detect (at least some of)
the same points in both images.



With these points, there's no chance to find true matches!

Under geometric and photometric variations.



Goal: Compactness and Efficiency

We want the representation to be as small and as fast as possible

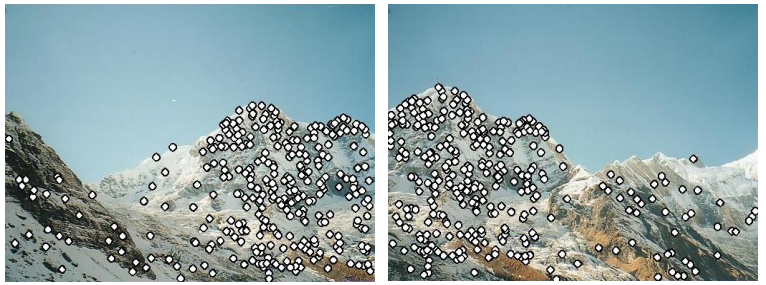
- Much smaller than a whole image

Sometimes, we'd like to run the detection procedure *independently* per image

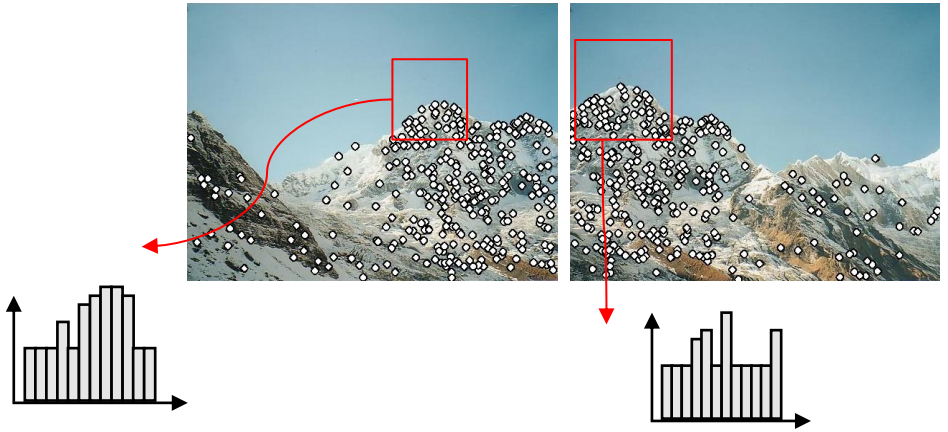
- Match just the compact descriptors for speed.
- *Difficult!* We don't get to see 'the other image' at match time, e.g., object detection.

Local Feature Matching: Main Components

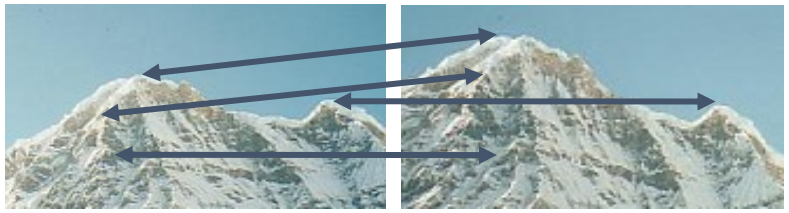
1) Detection:
Find a set of distinctive features (e.g., key points)



2) Description:
Extract feature descriptor around each interest point as vector.

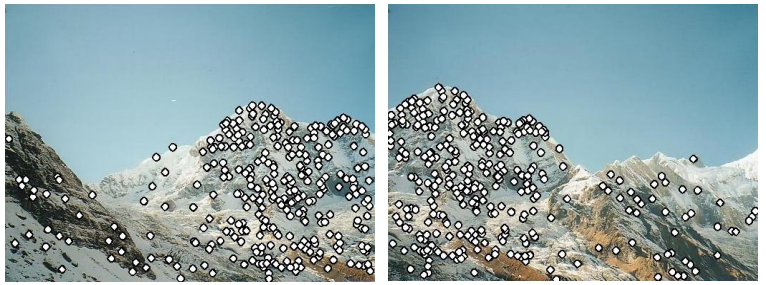


3) Matching:
Compute distance between feature vectors to find correspondence.

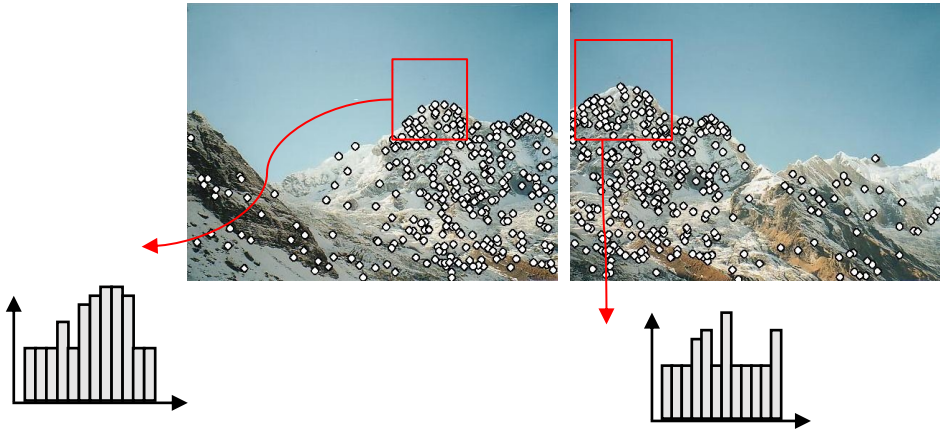


Local Feature Matching: Main Components

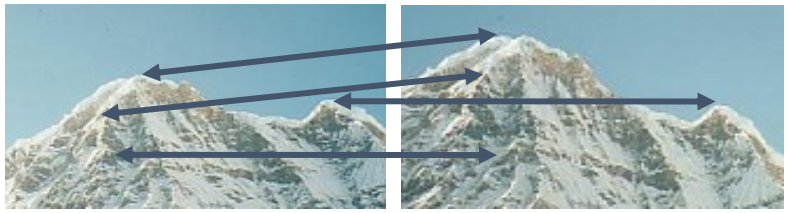
1) **Detection:**
   Find a set of distinctive features (e.g., key points)



2) **Description:**
   Extract feature descriptor around each interest point as vector.



3) **Matching:**
   Compute distance between feature vectors to find correspondence.



Corner Detection: Basic Idea

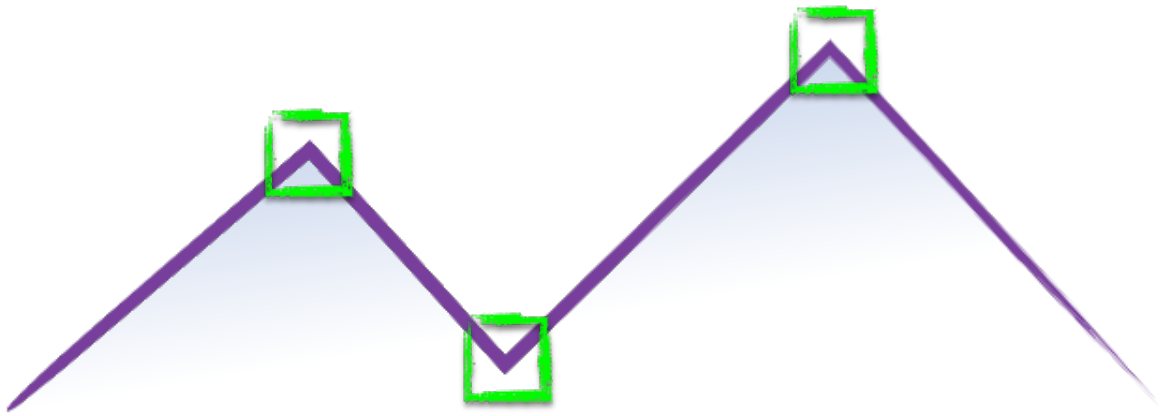
We do not know which other image locations the feature will end up being matched against.

But we can compute how stable a location is in appearance with respect to small variations in its position.

Strategy: Compare image patch against local neighbors.

How do you find a corner?

[Moravec 1980]

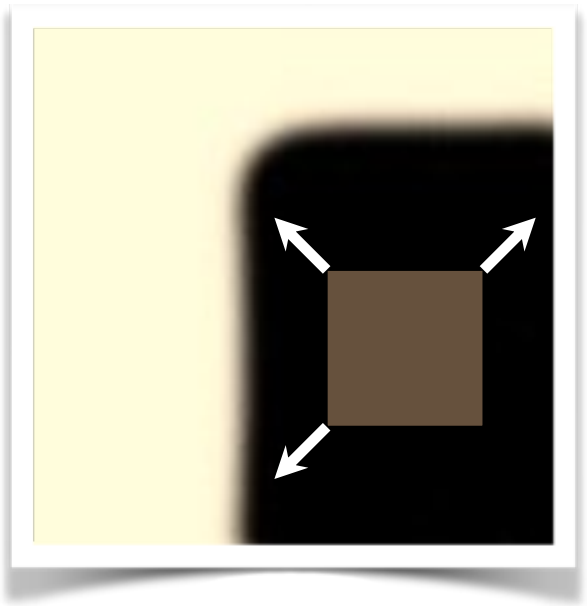


Easily recognized by looking through a small window

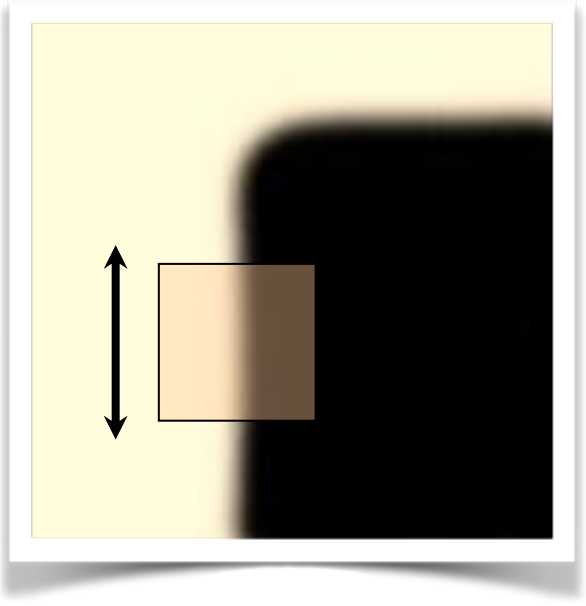
Shifting the window should give large change in intensity

Easily recognized by looking through a small window

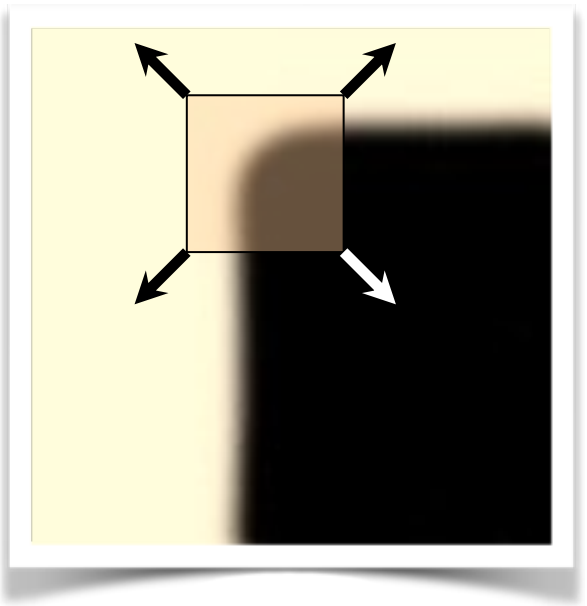
Shifting the window should give large change in intensity



"flat" region:
no change in all directions



"edge":
no change along the edge direction



"corner":
significant change in all directions

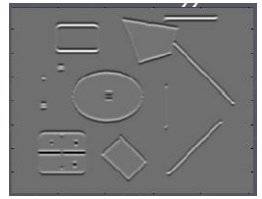
[Moravec 1980]

Finding corners

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

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

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



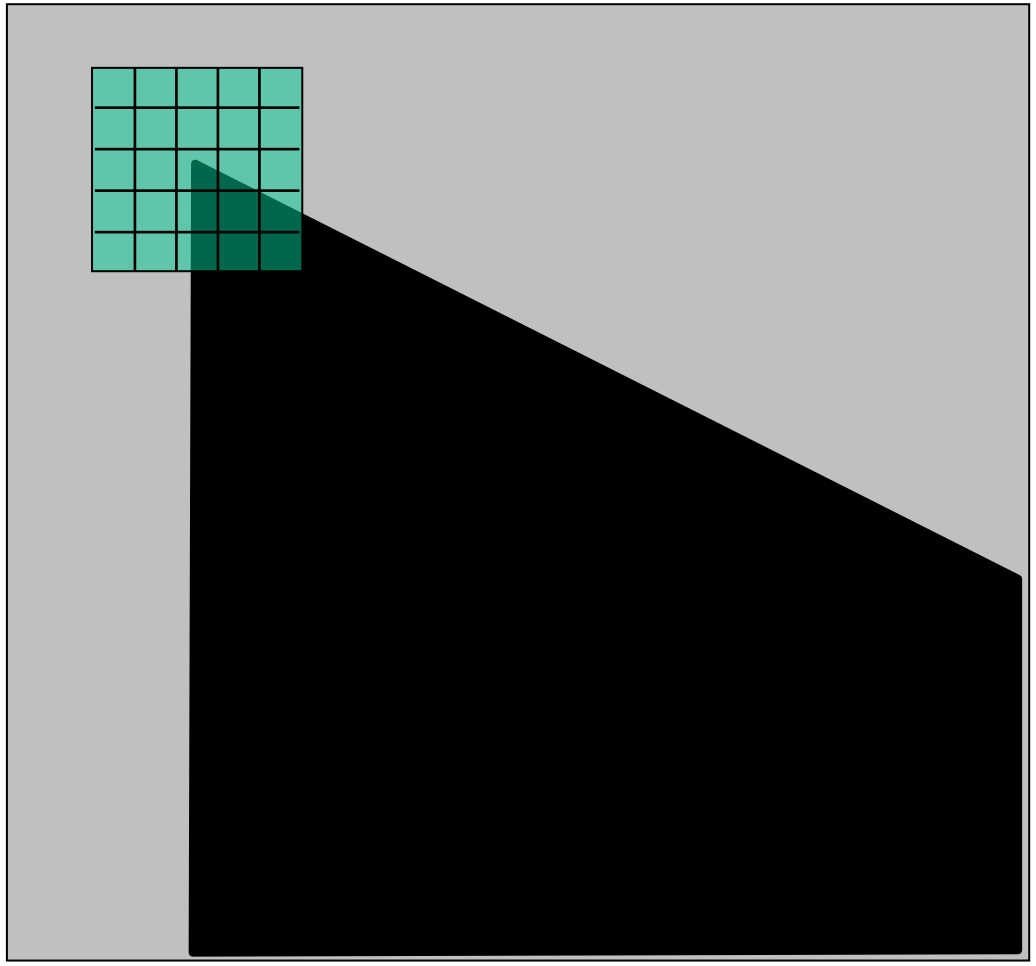
$$\begin{bmatrix} \sum_{p \in P} I_x I_x & \sum_{p \in P} I_x I_y \\ \sum_{p \in P} I_y I_x & \sum_{p \in P} I_y I_y \end{bmatrix}$$

1. Compute image gradients over a small region

(not just a single pixel)

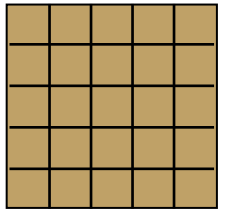
1. Compute image gradients over a small region

(not just a single pixel)



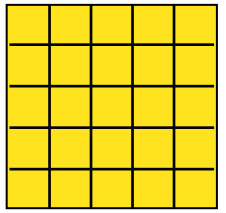
array of x gradients

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

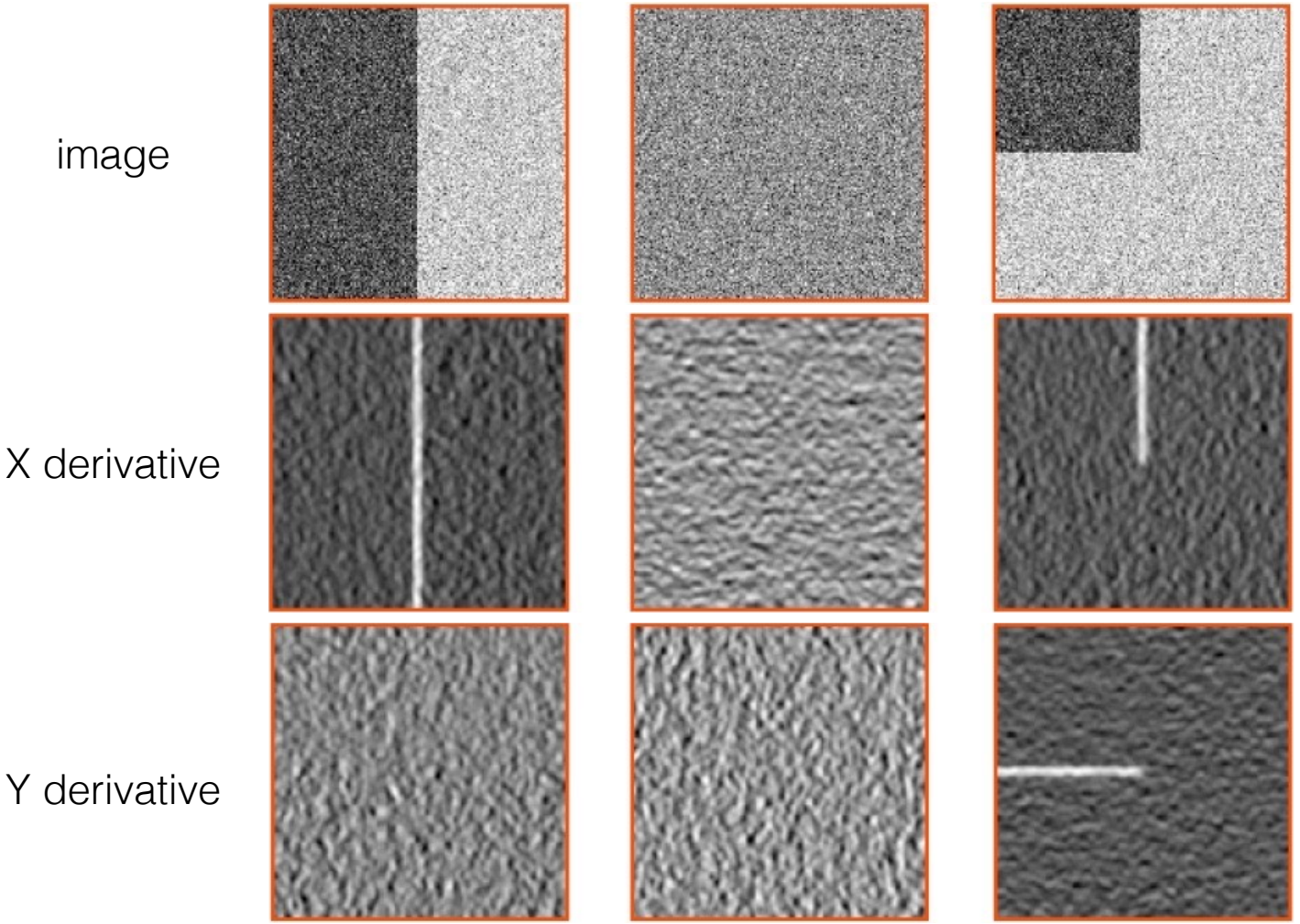


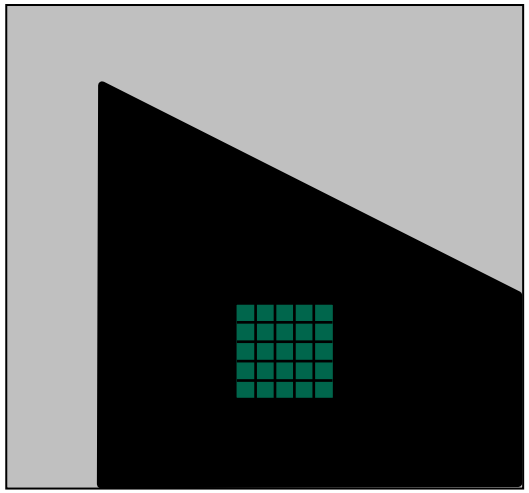
array of y gradients

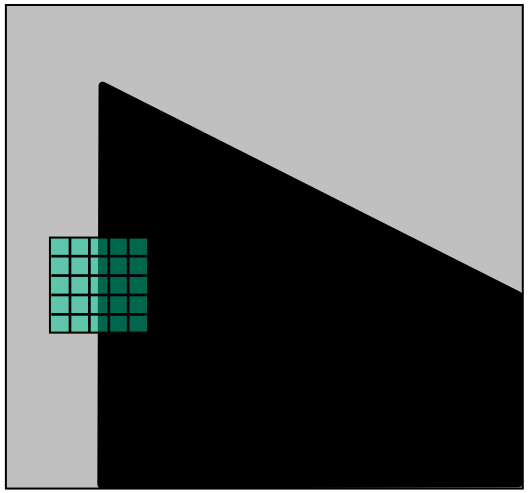
$$I_y = \frac{\partial I}{\partial y}$$

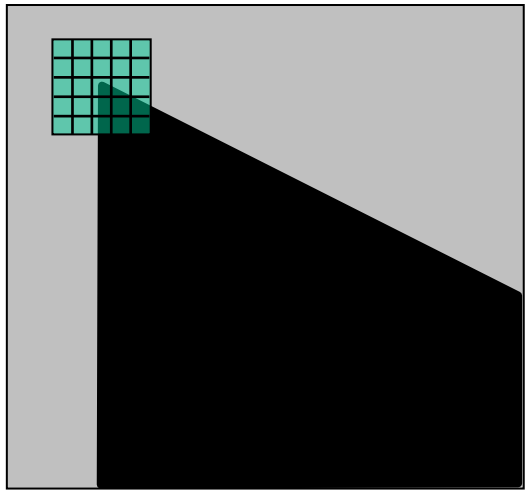


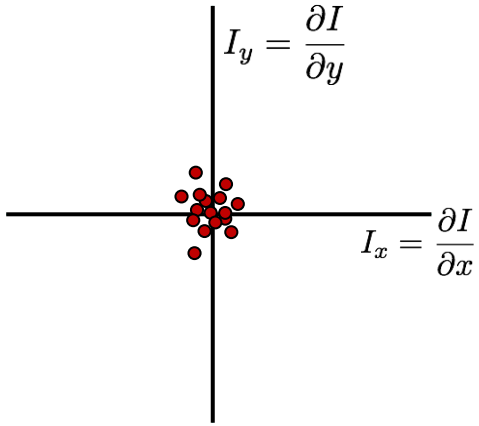
visualization of gradients

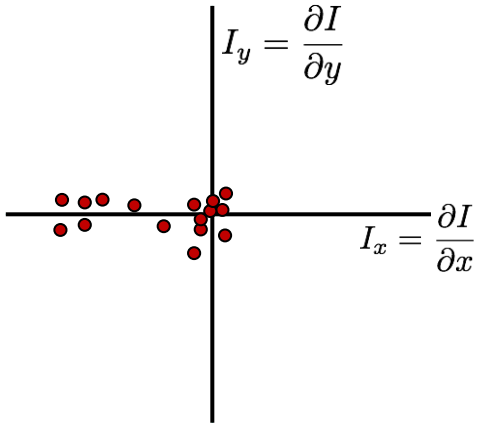


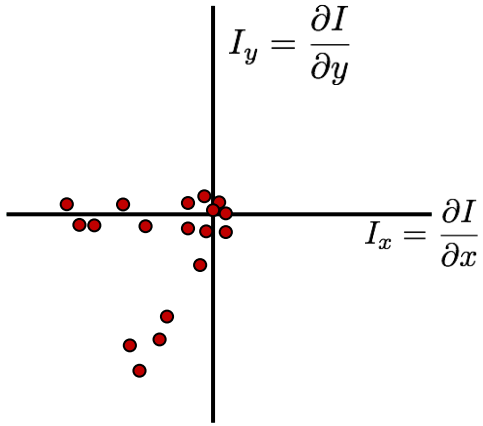




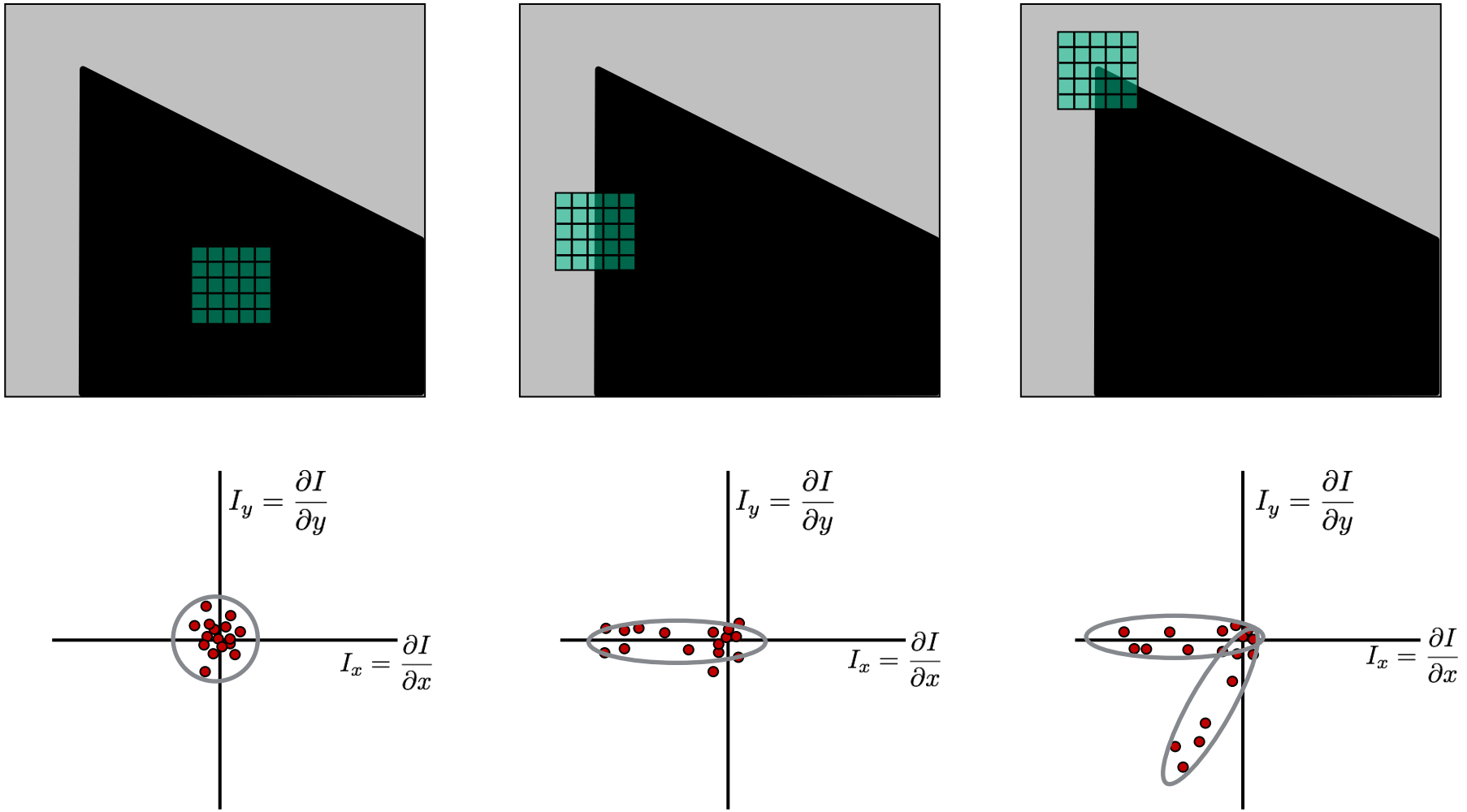




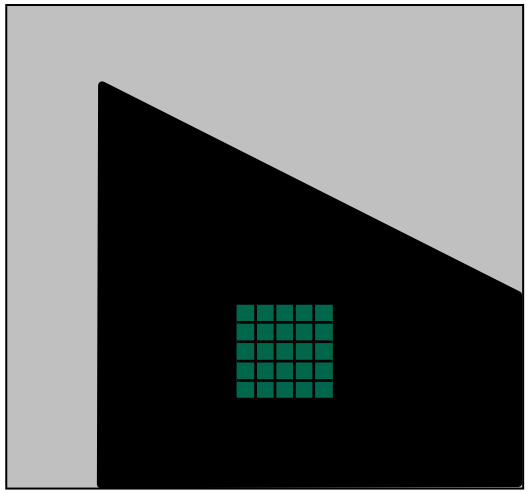


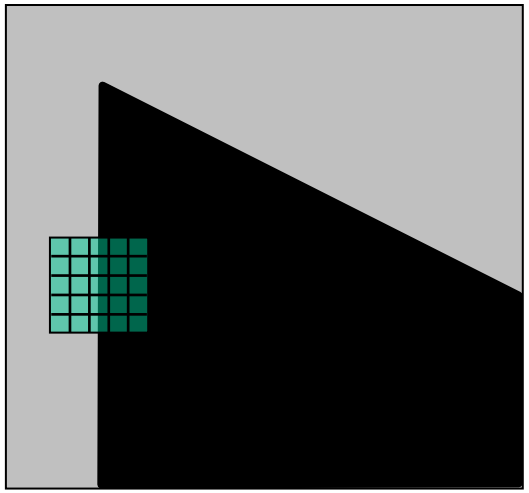


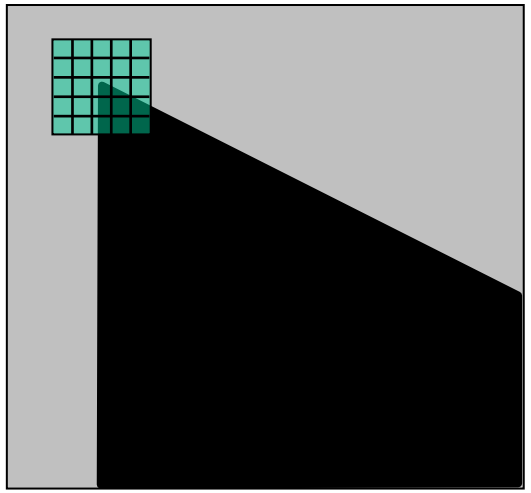
What does the distribution tell you about the region?

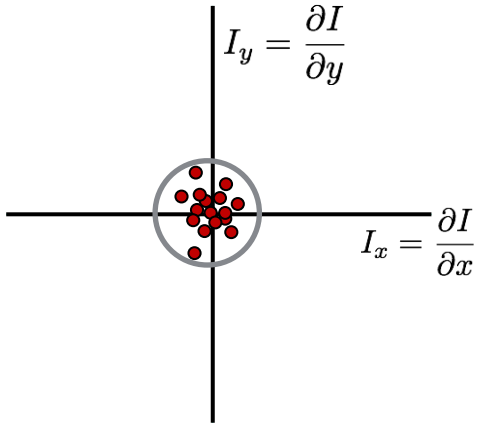


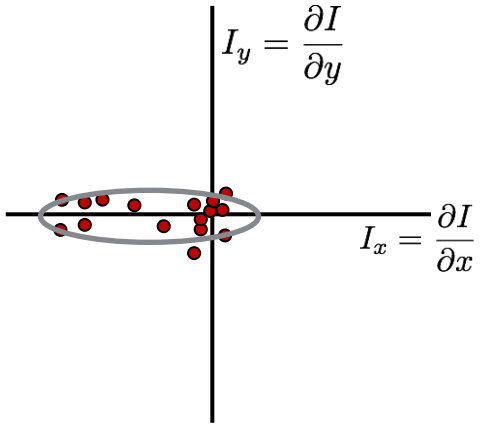
distribution reveals edge orientation and magnitude

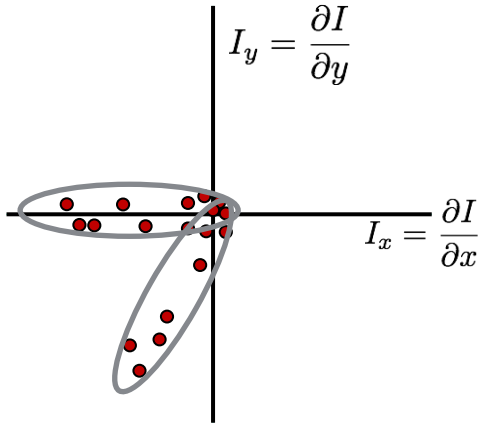










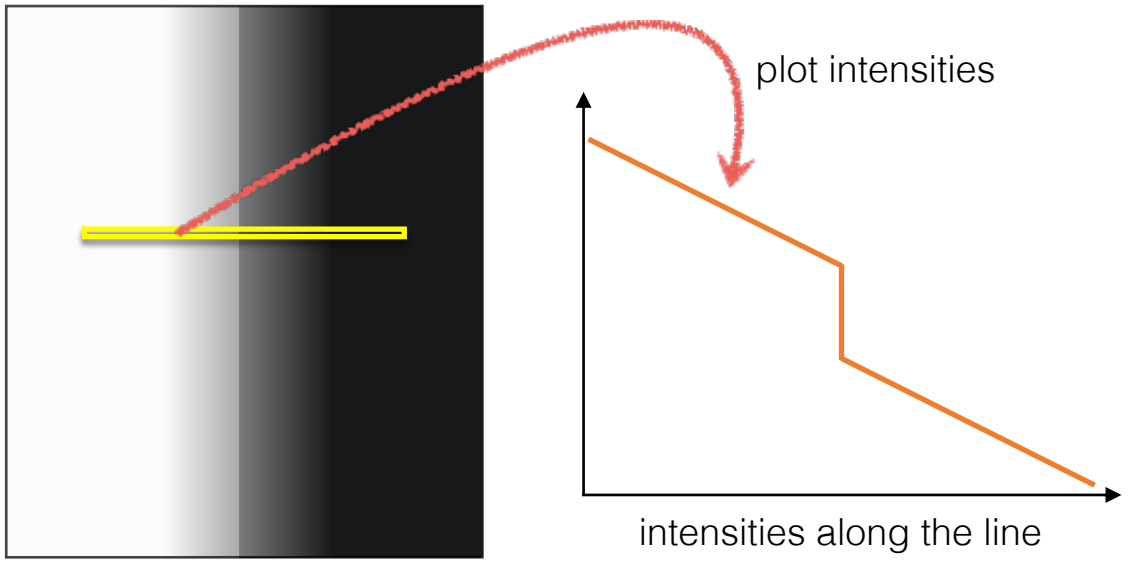


How do you quantify orientation and magnitude?

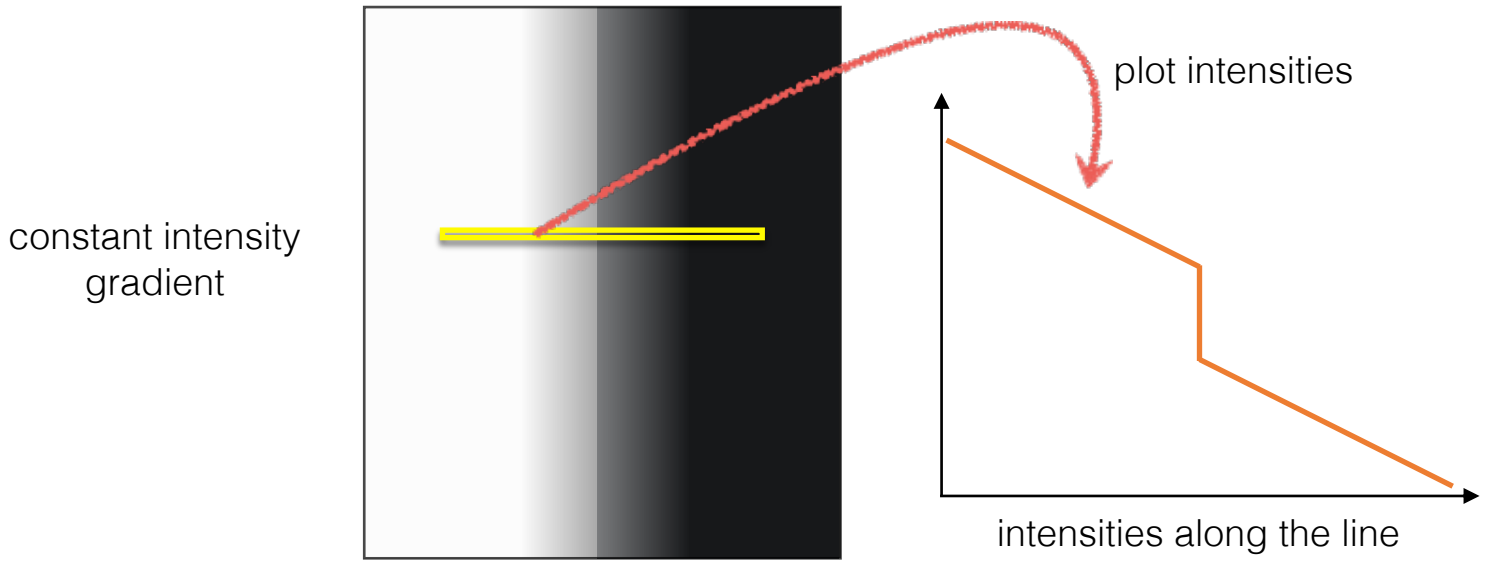
2. Subtract the mean from each image gradient

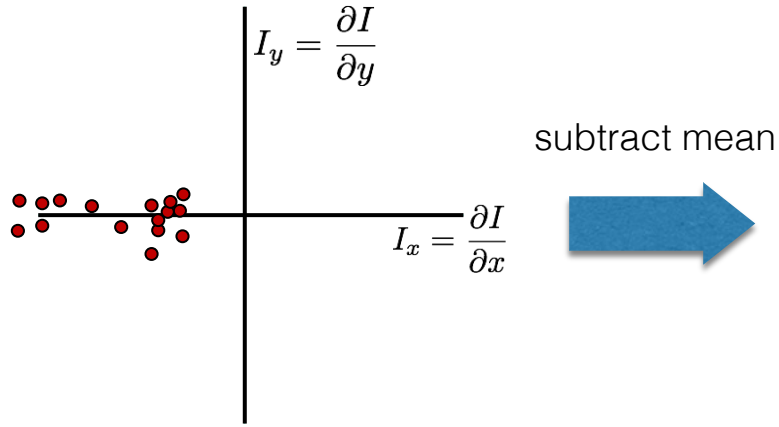
2. Subtract the mean from each image gradient

constant intensity gradient



2. Subtract the mean from each image gradient

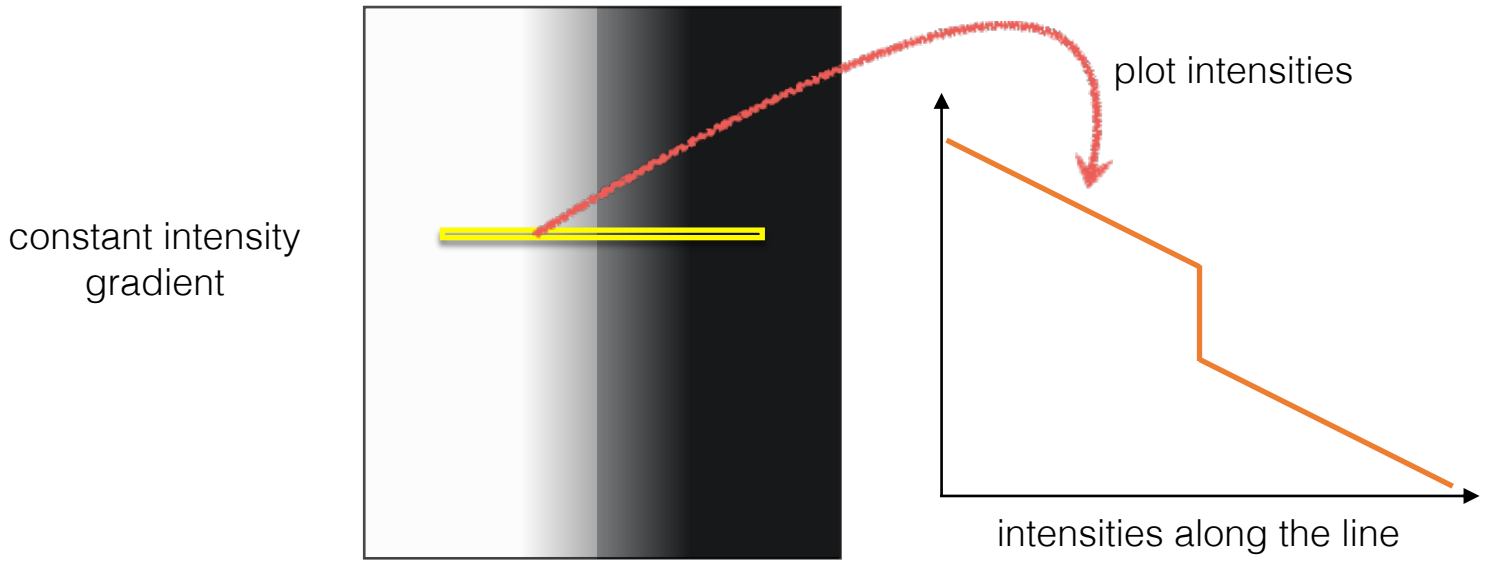


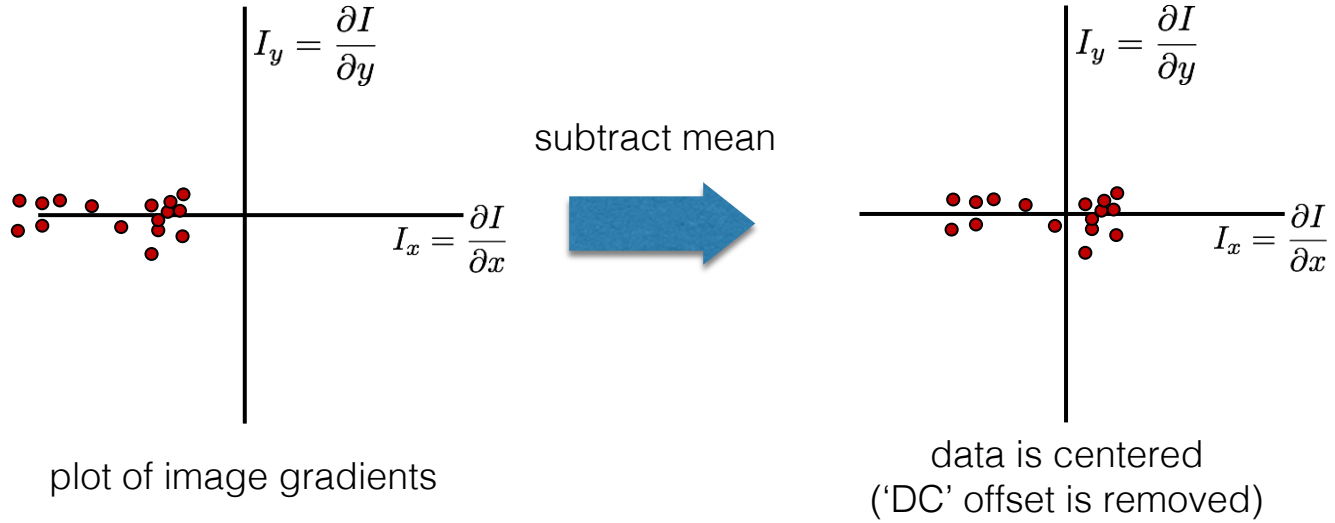


subtract mean

plot of image gradients

2. Subtract the mean from each image gradient





plot of image gradients

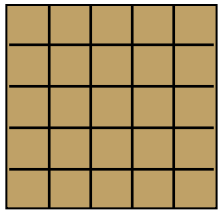
data is centered
('DC' offset is removed)

3. Compute the covariance matrix

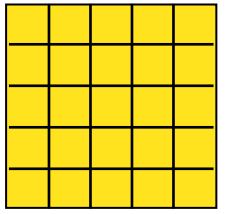
3. Compute the covariance matrix

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

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

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

array of x gradients

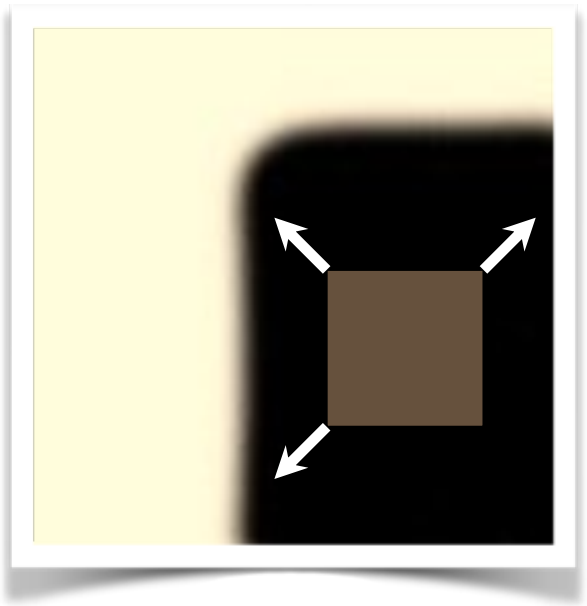
$I_y = \frac{\partial I}{\partial y}$ 

array of y gradients

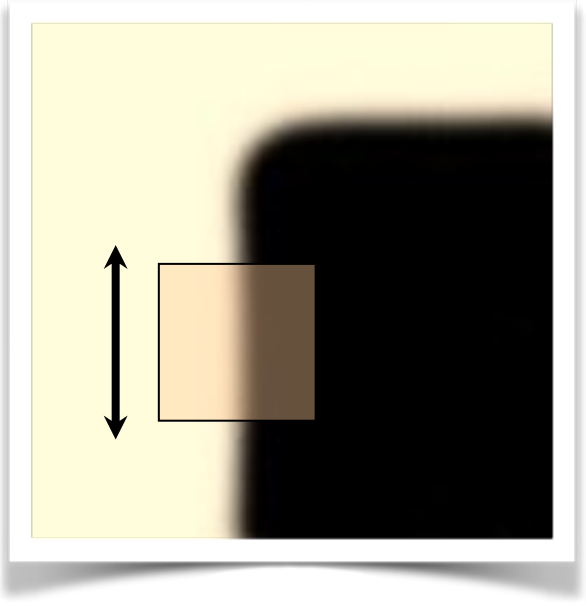
Where does this covariance matrix come from?

Easily recognized by looking through a small window

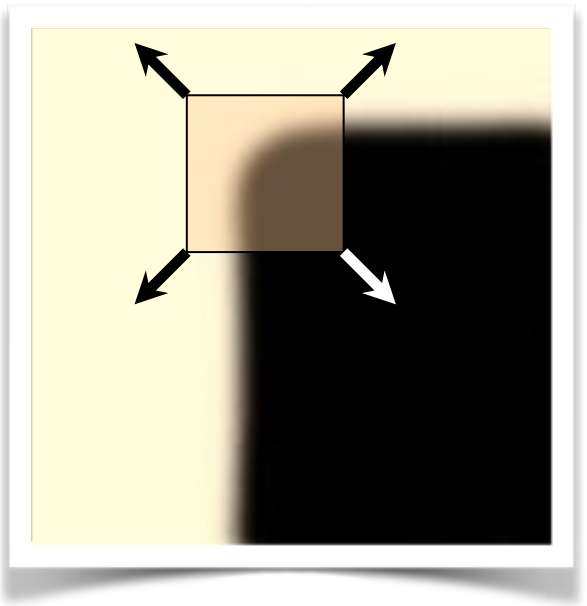
Shifting the window should give large change in intensity



"flat" region:
no change in all directions



"edge":
no change along the edge direction



"corner":
significant change in all directions

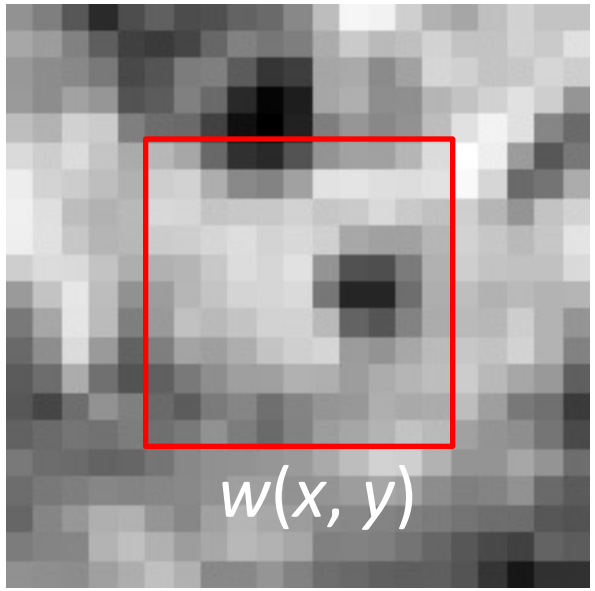
[Moravec 1980]

Corner Detection by Auto-correlation

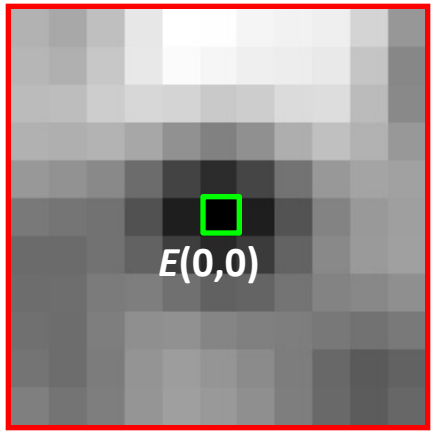
Change in appearance of window $w(x,y)$ for shift $[u,v]$:

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

$I(x, y)$



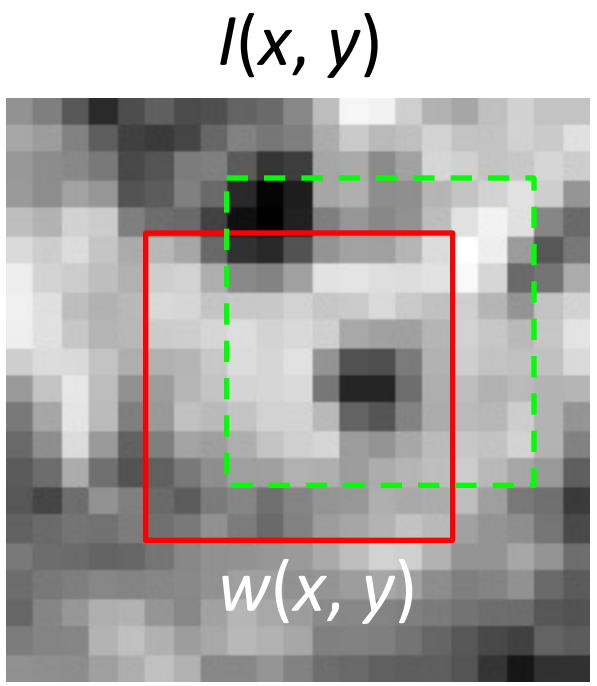
$E(u, v)$

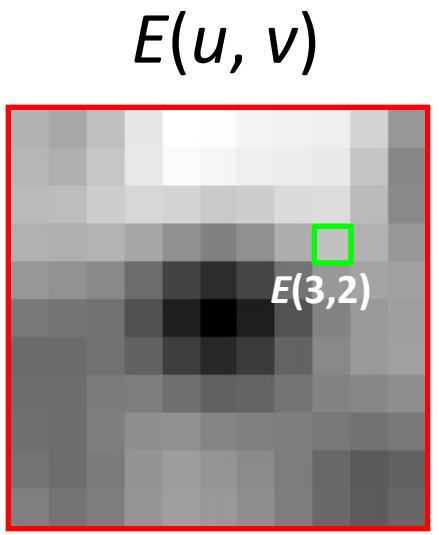


Corner Detection by Auto-correlation

Change in appearance of window $w(x,y)$ for shift $[u,v]$:

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





Visualizing Error function

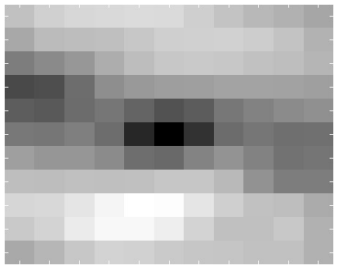
$$E(u,v) = \sum_{x,y} w(x,y)\left[I(x+u,y+v) - I(x,y)\right]^2$$

Correspond the three red crosses to (b,c,d).

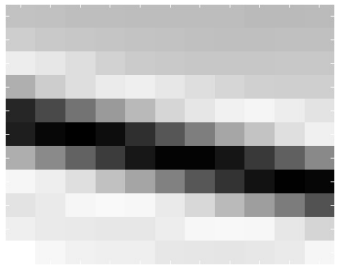
b) is from the flower bed (good unique minimum);
c) is from the roof edge (one-dimensional aperture problem); and
d) is from the cloud (no good peak).

$E(u,v)$

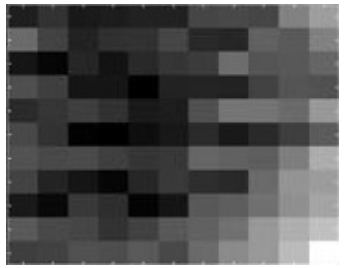
b)



c)

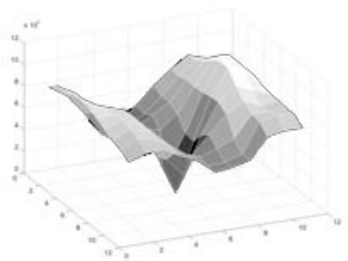


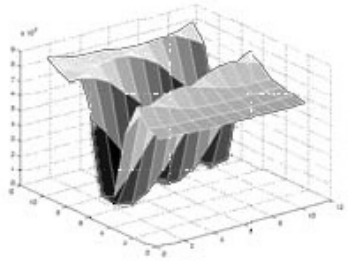
d)

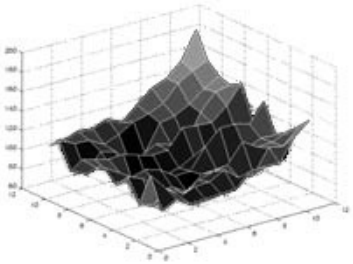


$E(u,v)$

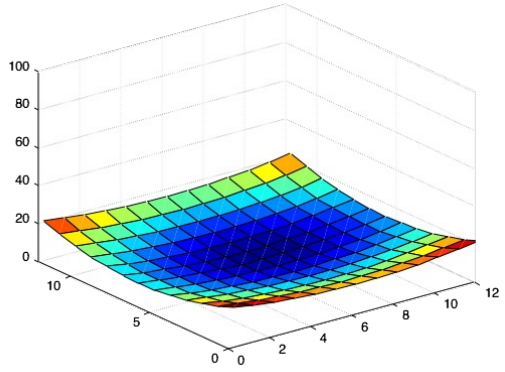
As a surface



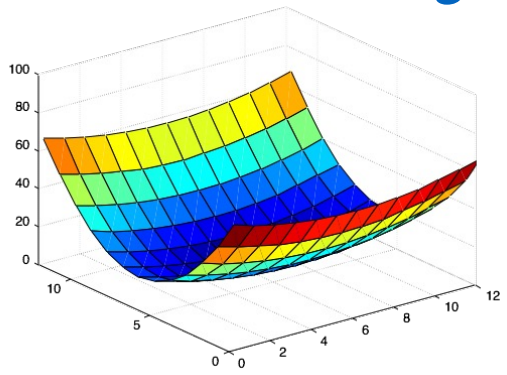




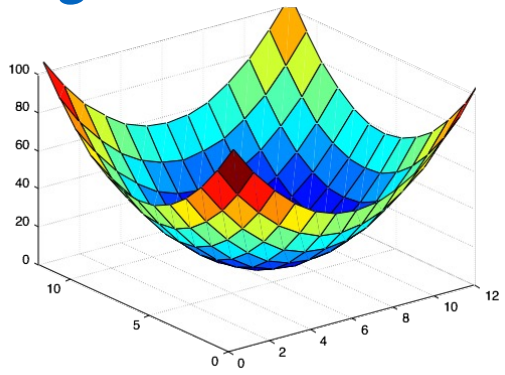
Which error surface indicates a good image feature?



flat



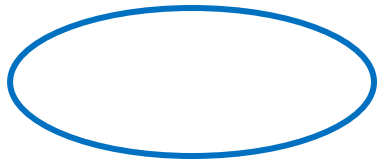
edge
'line'



corner
'dot'

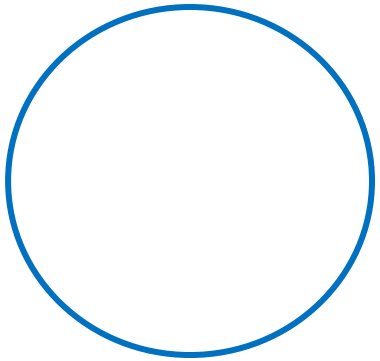
What kind of image patch do these surfaces represent?

Assume, we plot for f(x,y) = some fixed value p.



edge
'line'

corner
'dot'



flat

Corner Detection by Auto-correlation

Without loss of generality, assume a grayscale 2−dimensional image, I is used.
Consider taking an image patch $(x, y) \in W$ (window) and shift it by $(\Delta x, \Delta y)$.

The sum of squared differences (SSD) between these two patches, f is :

$$f(\Delta x, \Delta y) = \sum_{(x_k, y_k) \in W} \left(I(x_k, y_k) - I(x_k + \Delta x, y_k + \Delta y)\right)^2$$

$I(x + \Delta x, y + \Delta y)$ can be approximated by a Taylor expansion.
Let I_x and I_y be the partial derivatives of I, such that

$$I(x + \Delta x, y + \Delta y) \approx I(x, y) + I_x(x, y)\Delta x + I_y(x, y)\Delta y$$

$$\Rightarrow f(\Delta x, \Delta y) \approx \sum_{(x,y) \in W} \left(I_x(x, y)\Delta x + I_y(x, y)\Delta y\right)^2$$

Alternatively, $f(\Delta x, \Delta y) \approx \begin{pmatrix} \Delta x & \Delta y \end{pmatrix} M \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}$

where M is the structure tensor, given by :

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

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

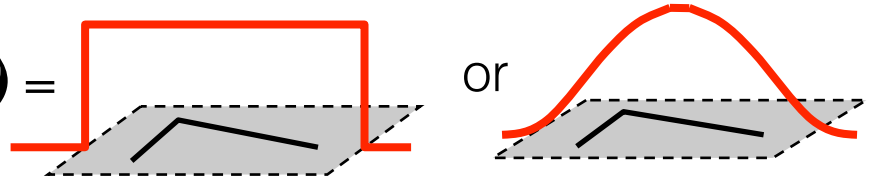
Window function

Shifted intensity

Intensity

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

Window function $w(x,y)$ =



1 in window, 0 outside

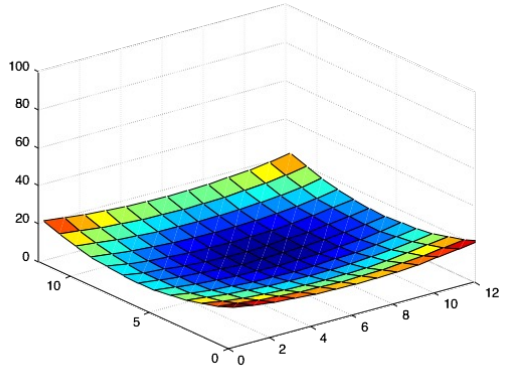
or

Gaussian

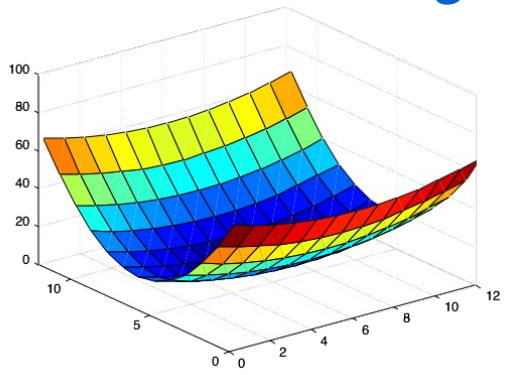
$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

'second moment' matrix
'structure tensor'

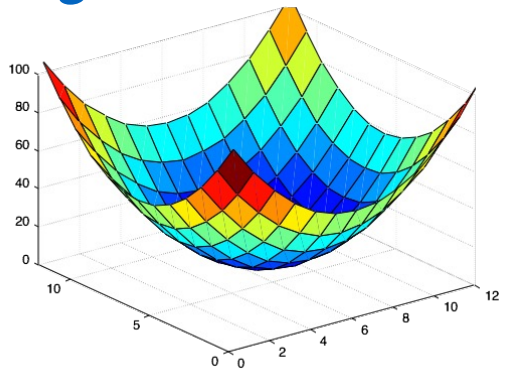
Which error surface indicates a good image feature?



flat



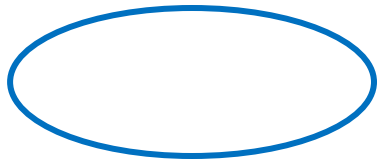
edge
'line'



corner
'dot'

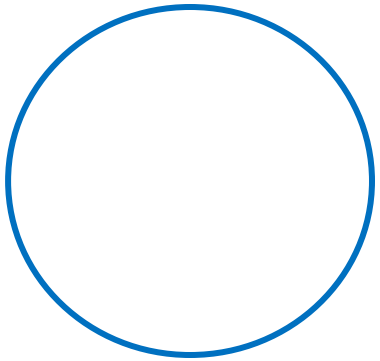
What kind of image patch do these surfaces represent?

Assume, we plot for f(x,y) = some fixed value p.



edge
'line'

corner
'dot'



flat

4. Compute eigenvalues and eigenvectors

4. Compute eigenvalues and eigenvectors

eigenvalue

$$Me = \lambda e$$

eigenvector

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

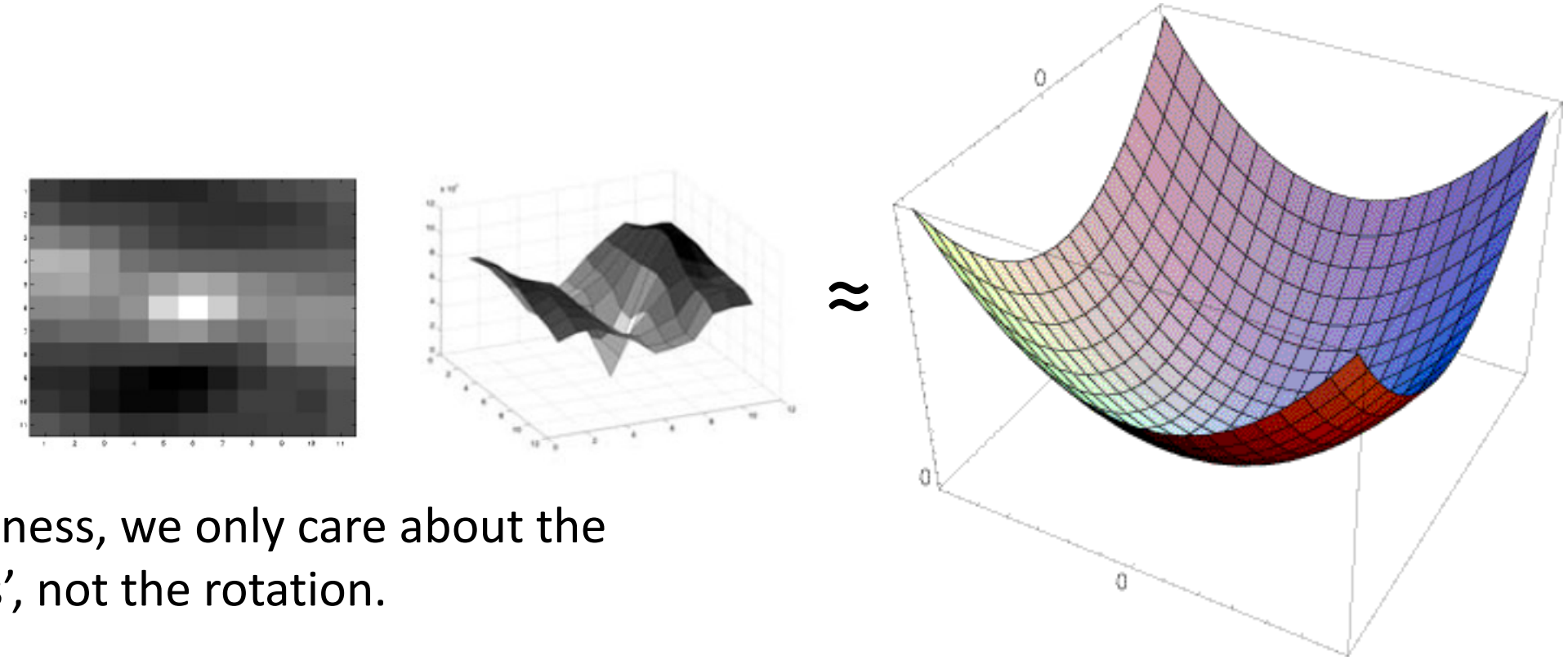
1. Compute the determinant of $M - \lambda I$
   (returns a polynomial)

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

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

Strategy

Approximate $E(u,v)$ locally by a quadratic surface, and look for that instead.



For cornerness, we only care about the ‘steepness’, not the rotation.

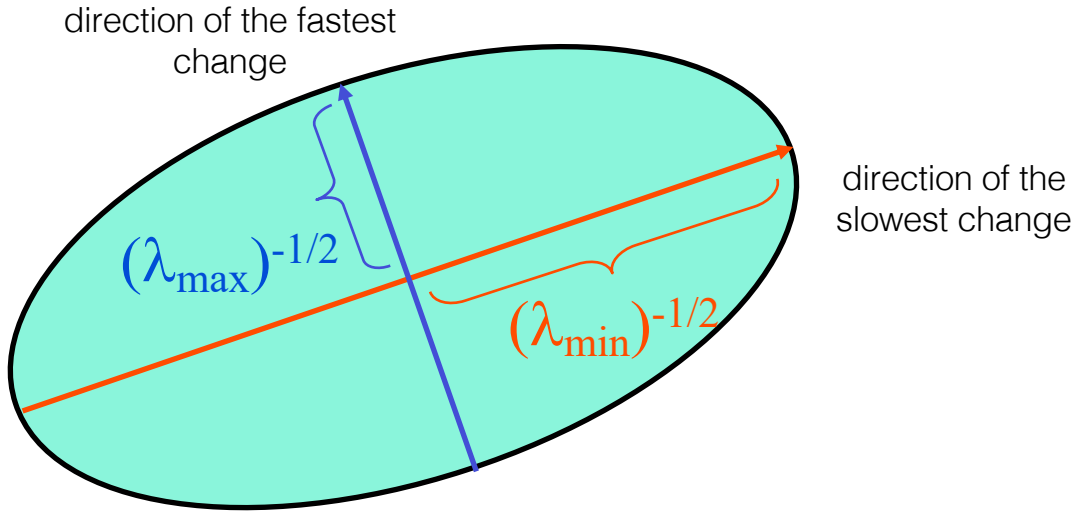
Visualization as an ellipse

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

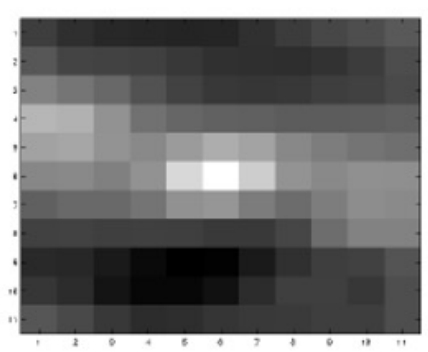
We can visualize *M* as an ellipse with axis lengths determined by the eigenvalues and orientation determined by *R*

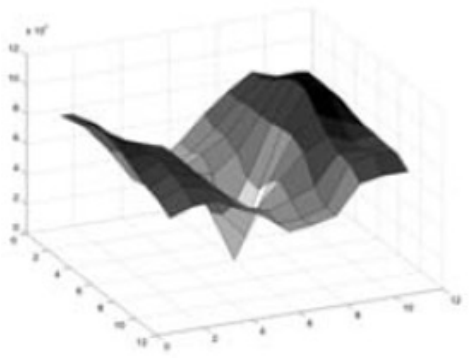
Ellipse equation:

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

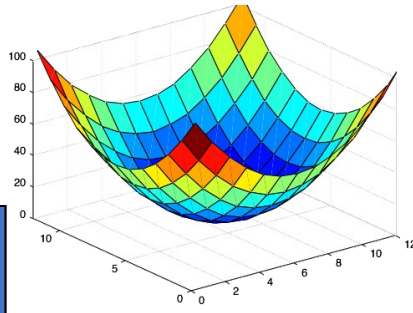


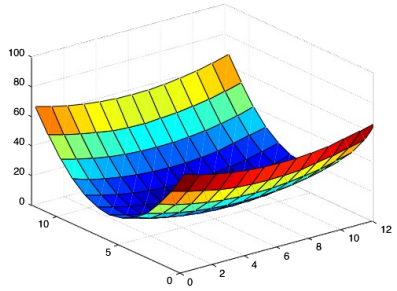
Classification of image points using eigenvalues of *M*

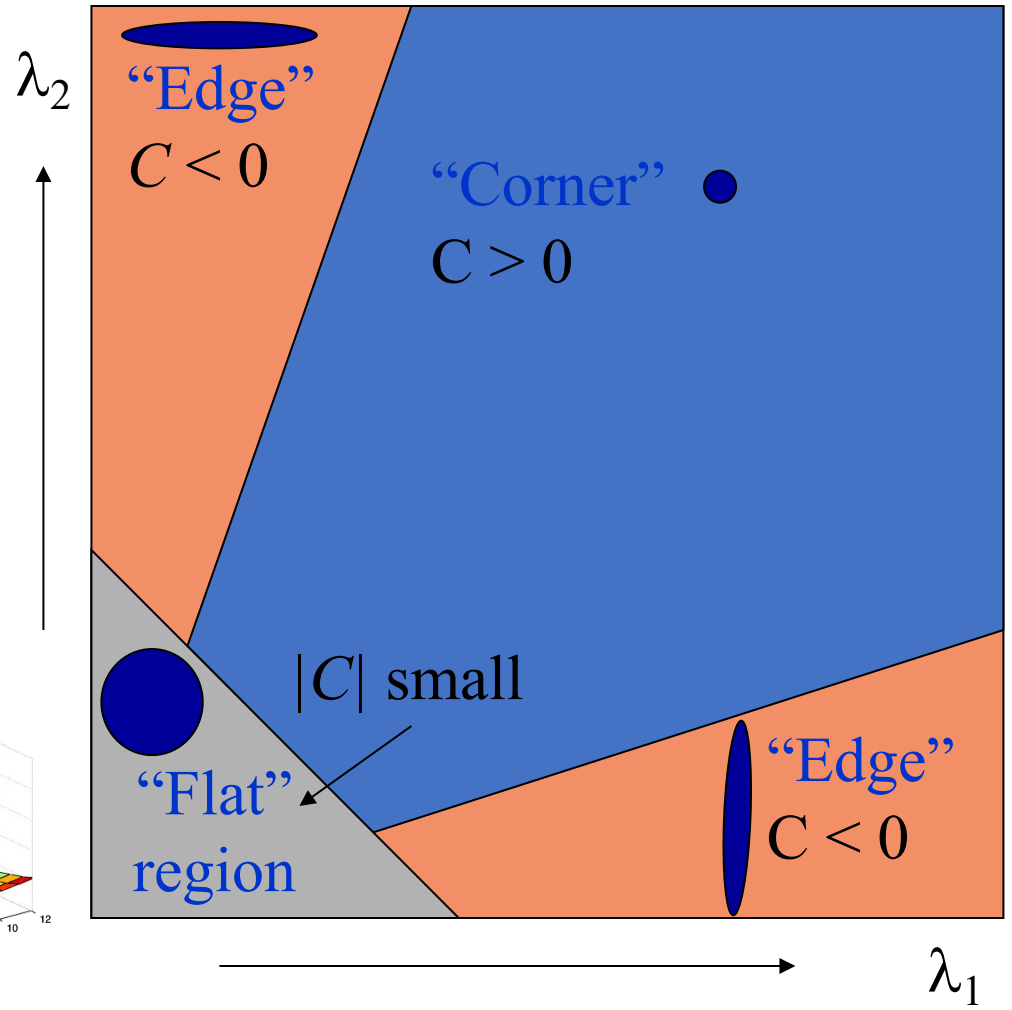


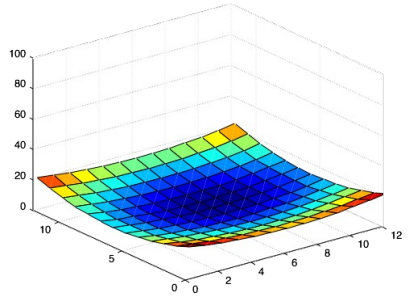


Classification of image points using eigenvalues of *M*









Cornerness score:

$C = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$

α: some constant (~0.04 to 0.06)

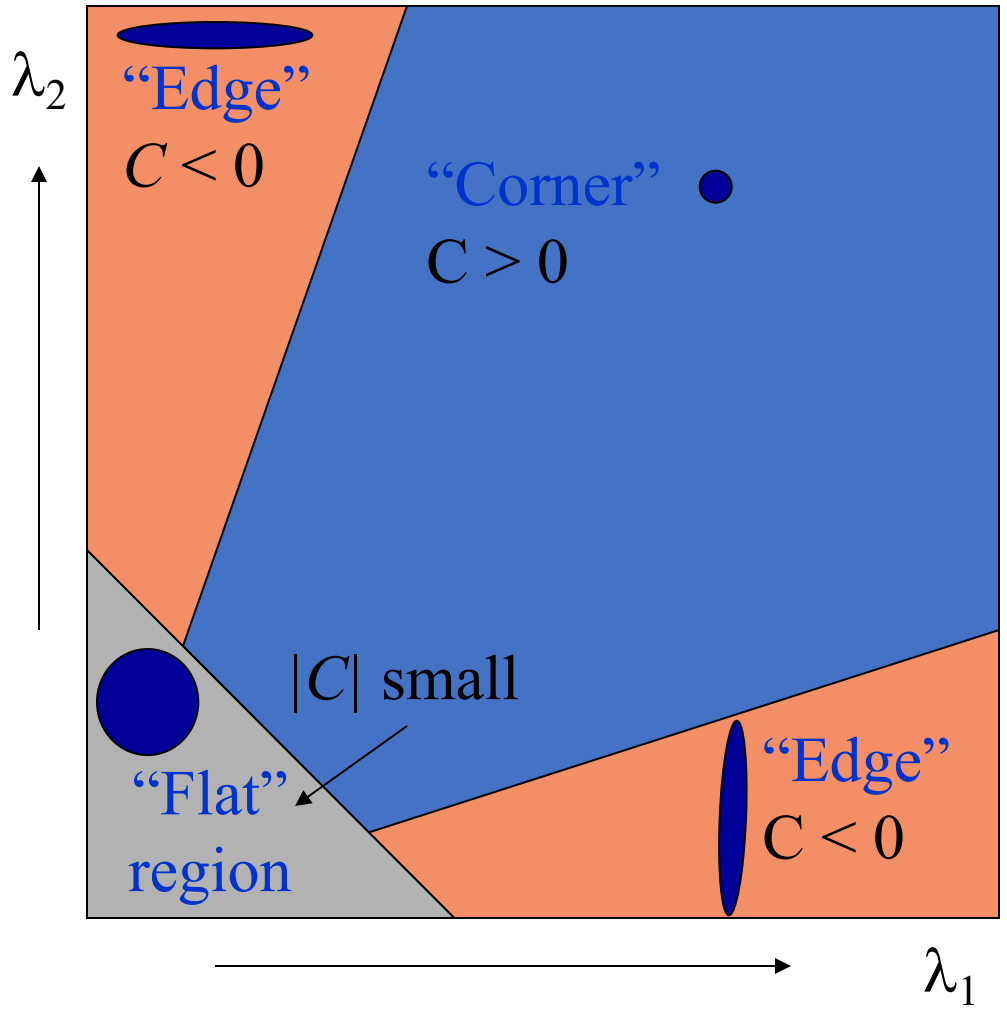
5. Use threshold on eigenvalues to detect corners

Classification of image points using eigenvalues of *M*

Cornerness score:

$C = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$

α: some constant (~0.04 to 0.06)



Linear Algebra: Recap

Determinant of a diagonal matrix is the *product* of all eigenvalues.

A is diagonal.

Trace of a square matrix is the *sum* of its diagonal entries; and is the sum of its eigenvalues.

Classification of image points using eigenvalues of *M*

Cornerness score:

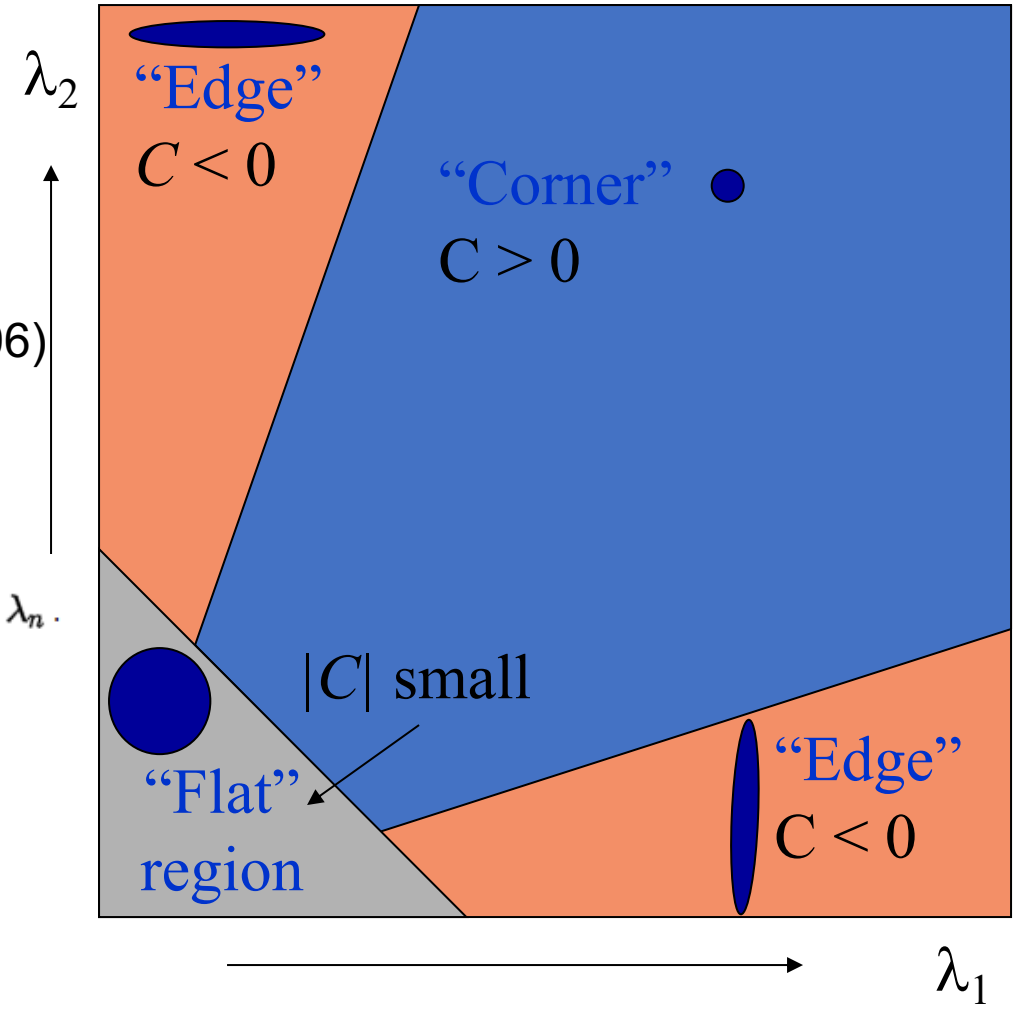
$C = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$

α: some constant (~0.04 to 0.06)

Remember your linear algebra:

Determinant: $\det(A) = \prod_{i=1}^{n} \lambda_i = \lambda_1 \lambda_2 \cdots \lambda_n$.
(diagonal matrices)

Trace: $\mathrm{tr}(A) = \sum_i \lambda_i$.



$\det M = \lambda_1\lambda_2$

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

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

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

$$C = \det(A) - \alpha \mathrm{trace}(A)^2$$

Linear Algebra: Recap

But I don't know A directly without eigen-decomposition.

- M exhibits *similarity invariance* – key properties of M are invariant under similarity transforms.
- Determinant and trace of M are equal to those of A.

If M is square and diagonalizable into $M = R^{-1}AR$, then:

$$\det(M) = \det(R^{-1})\det(A)\det(R)$$

$$\det(M) = \det(A) \quad \text{as} \quad \det(R^{-1})\det(R) = 1$$

For trace, given matrices B, C, then $\text{trace}(BC) = \text{trace}(CB)$

$$\text{trace}(R^{-1}(AR)) = \text{trace}\big((AR)R^{-1}\big) = \text{trace}(A)$$

$$\text{so} \quad \text{trace}(M) = \text{trace}(A)$$

Classification of image points using eigenvalues of *M*

Cornerness score:

$C = \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$

α: some constant (~0.04 to 0.06)

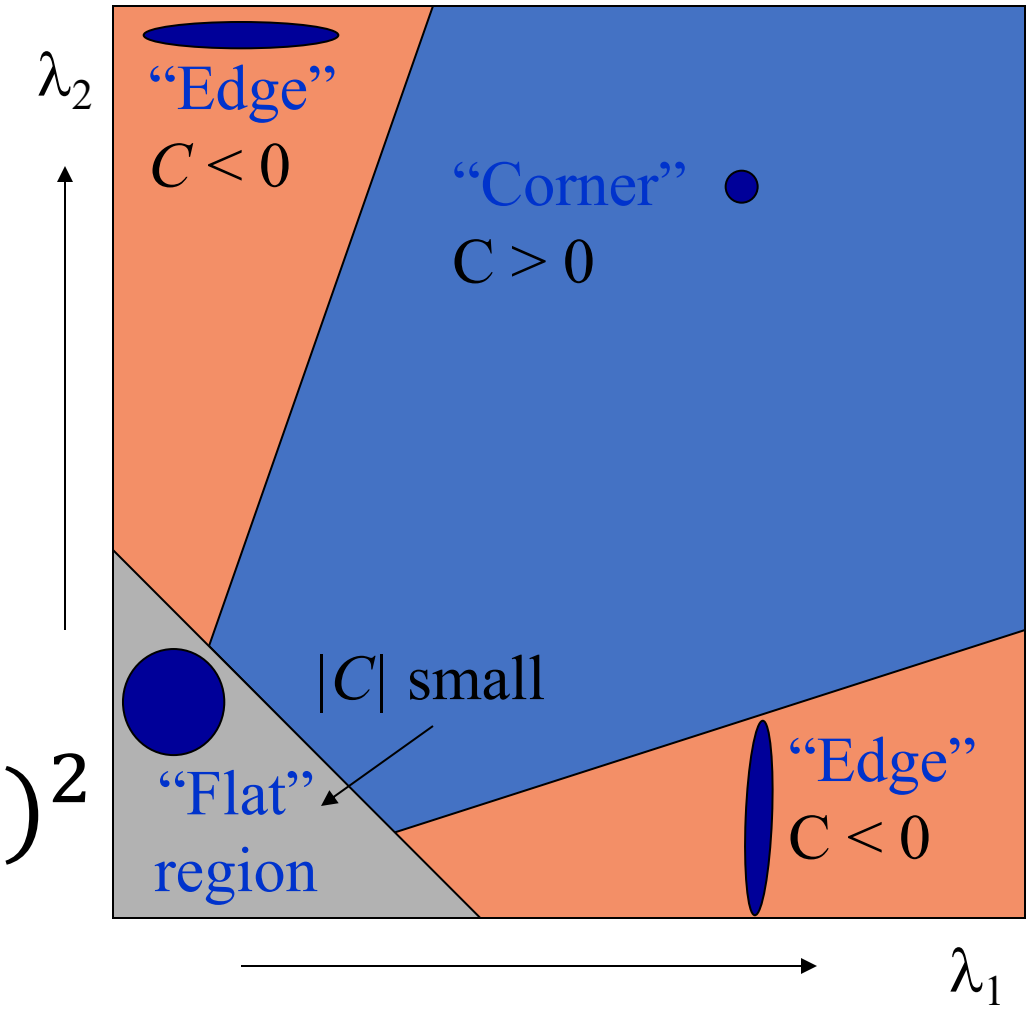
Remember your linear algebra:

Determinant: $\det(A) = \prod_{i=1}^{n} \lambda_i = \lambda_1 \lambda_2 \cdots \lambda_n$.
(diagonal matrices)

Trace: $\mathrm{tr}(A) = \sum_i \lambda_i$.

$$C = \det(M) - \alpha \mathrm{trace}(M)^2$$

Avoids explicit eigenvalue computation!



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 Corner Detector

1)Compute *M* matrix for each pixel to recover *cornerness* score C.

Note: We can find *M* purely from the per-pixel image derivatives!

2)Threshold to find pixels which give large corner response (C > threshold).

3)Find the local maxima pixels, i.e., non-maxima suppression.

C.Harris and M.Stephens. "A Combined Corner and Edge Detector." *Proceedings of the 4th Alvey Vision Conference*: pages 147—151, 1988.

Harris Corner Detector

- Compute image discrete first derivatives.
- Compute products of derivates to use in M
- Create M via window function: Gaussian filter *g()* with σ

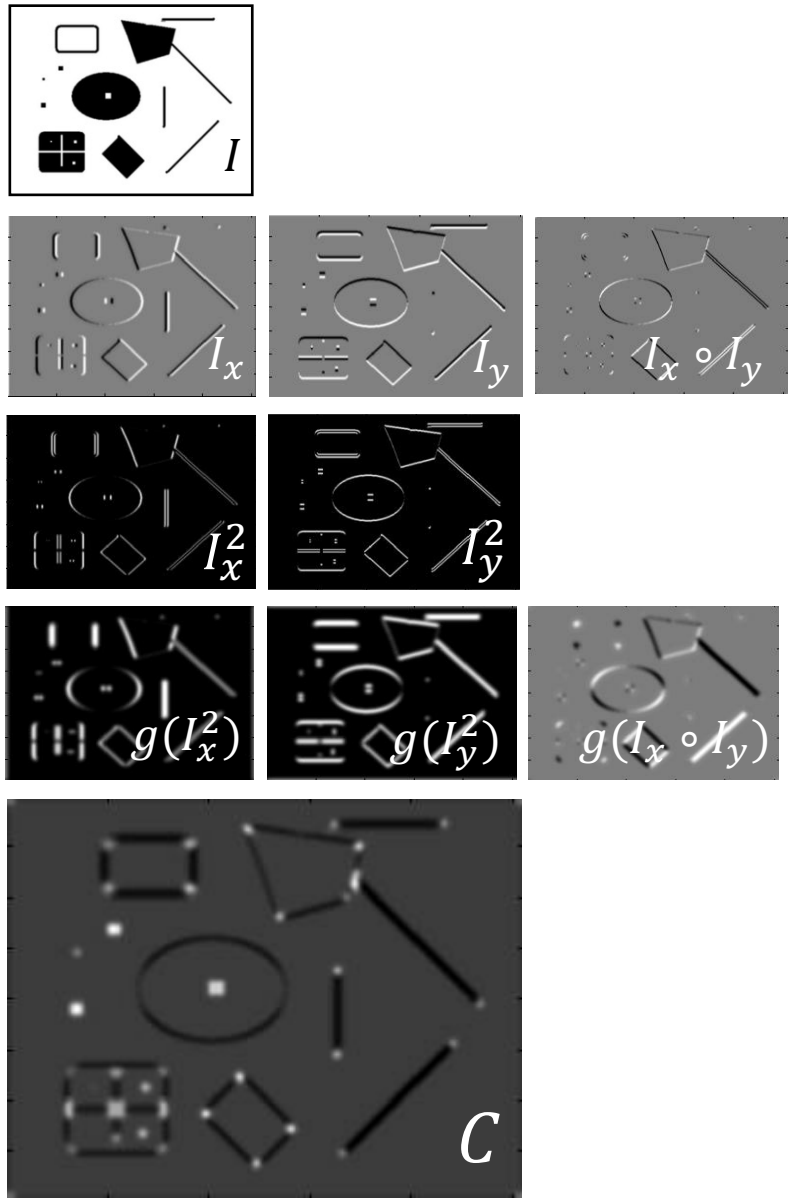
$$g(I_x^2), g(I_y^2), g(I_x \circ I_y)$$

- Compute cornerness

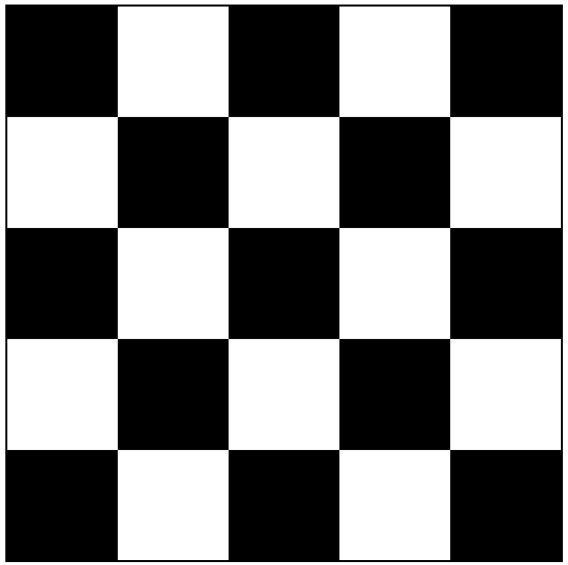
$$C = \det(M) - \alpha \operatorname{trace}(M)^2 = g(I_x^2) \circ g(I_y^2) - g(I_x \circ I_y)^2 - \alpha\left[g(I_x^2) + g(I_y^2)\right]^2$$

Reminder: $a \circ b$ is Hadamard product (element-wise multiplication)

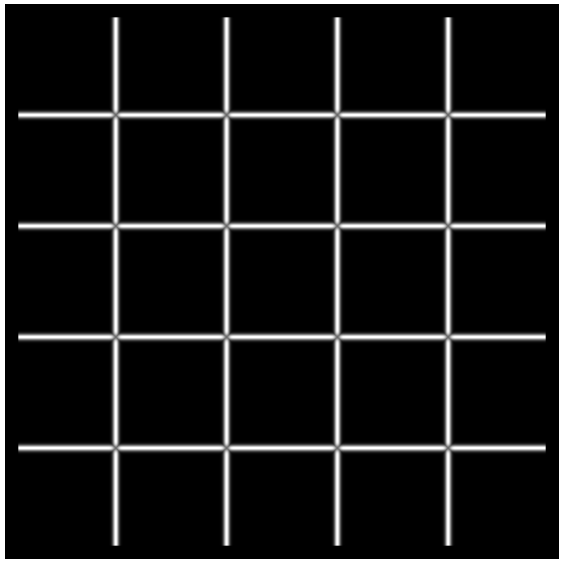
5. Threshold on C to pick high cornerness
6. Non-maximal suppression to pick peaks.



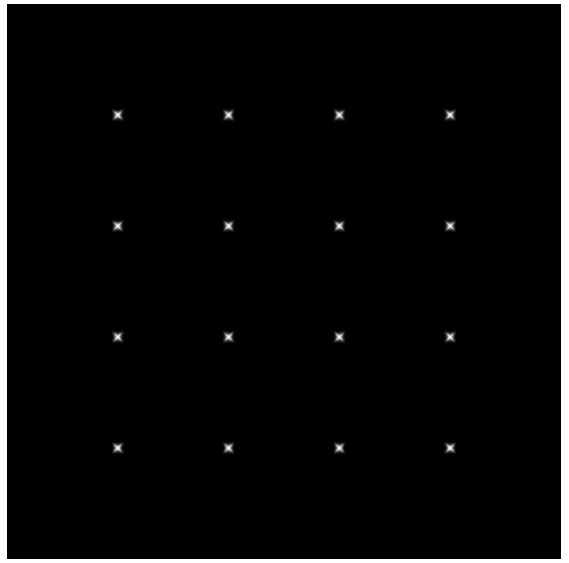
Harris (1988).



I



$\lambda_{\max}$



$\lambda_{\min}$

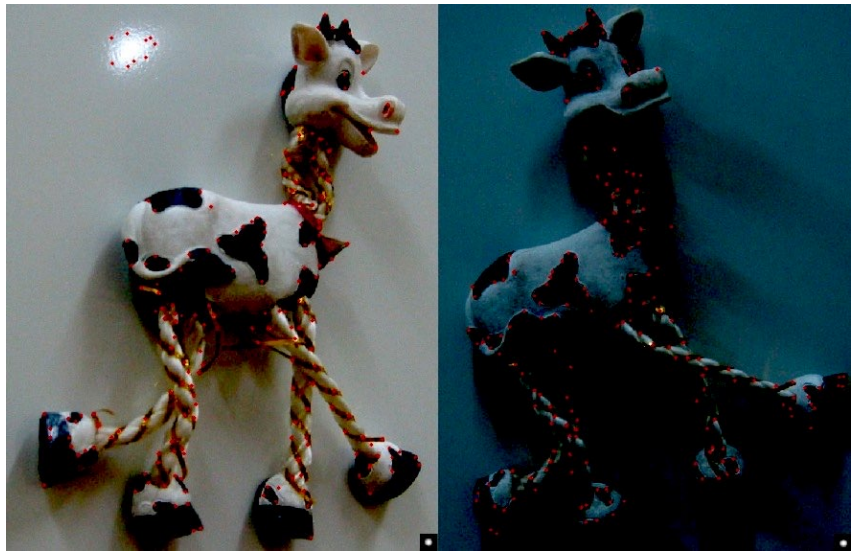
Yet another option:

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

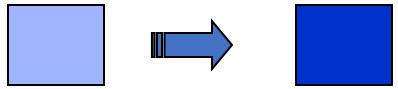
Invariance and Covariance

Are locations *invariant* to photometric transformations and *covariant* to geometric transformations?

- **Invariance:** image is transformed and corner locations do not change
- **Covariance / Equivariance:** if we have two transformed versions of the same image, features should be detected in corresponding locations



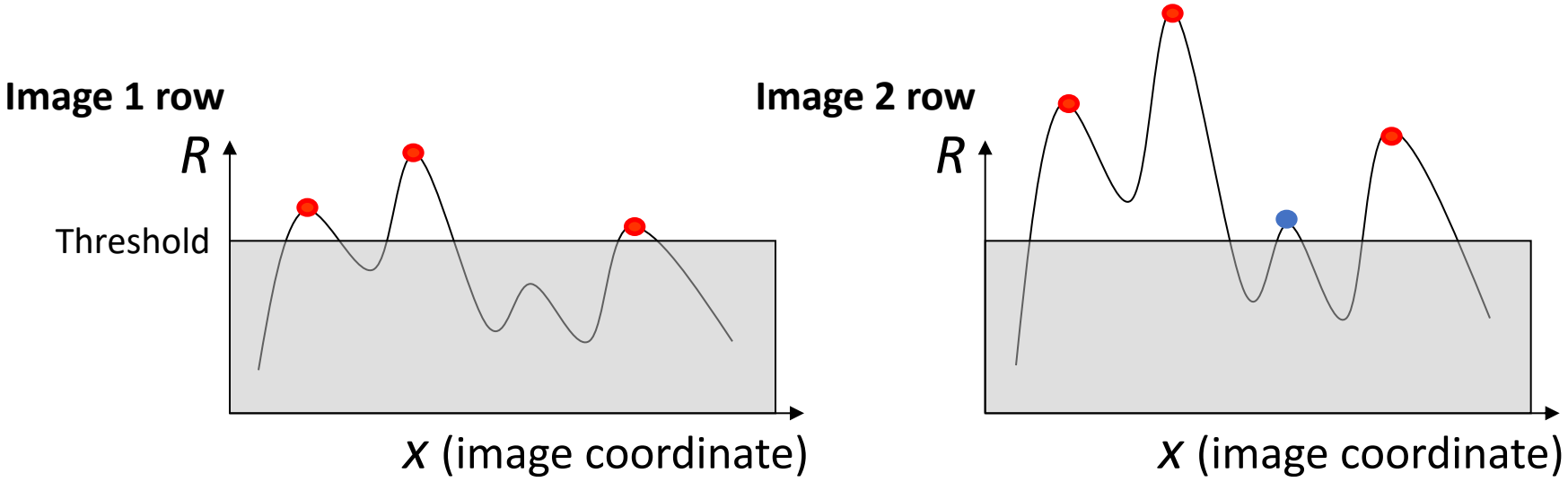
Affine Intensity Change



$I \rightarrow a\,I + b$

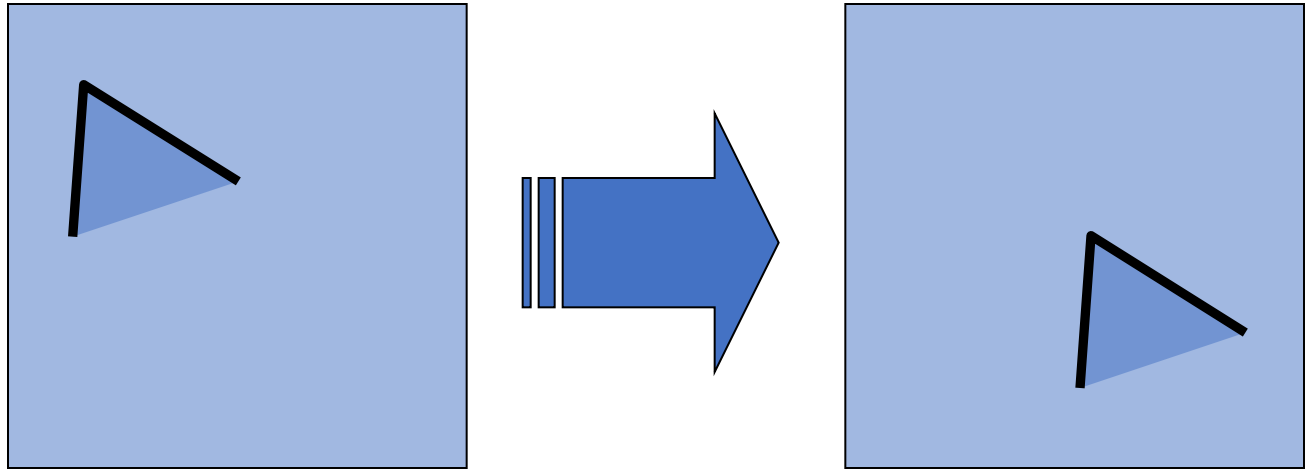
Only derivatives are used => invariance to intensity shift $I \rightarrow I + b$

Intensity scaling: $I \rightarrow a\,I$ => *still affected*



Partially invariant to affine intensity change

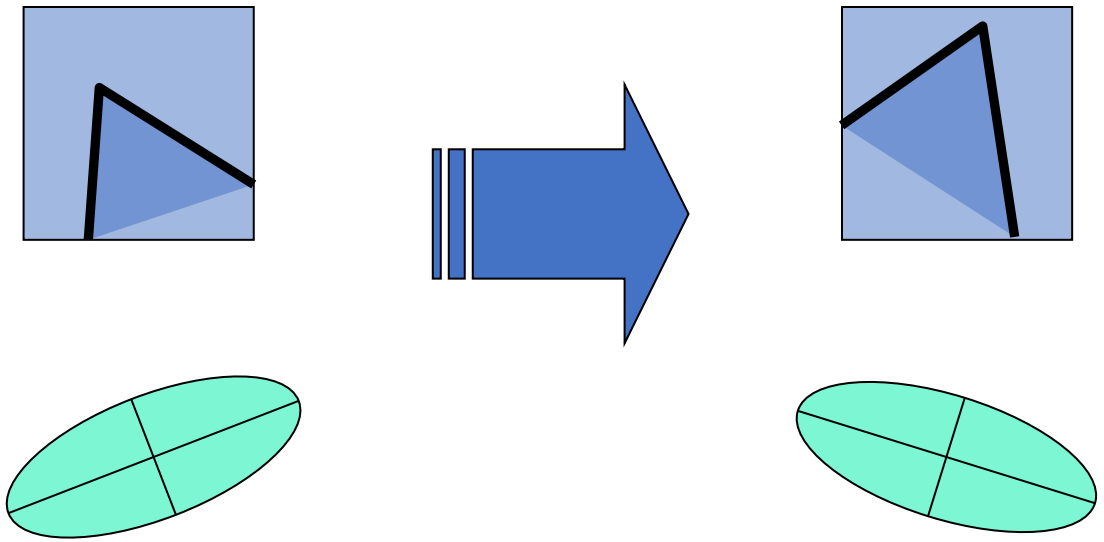
Image Translation



- Derivatives and window function are shift-invariant.

Corner location is covariant w.r.t. translation

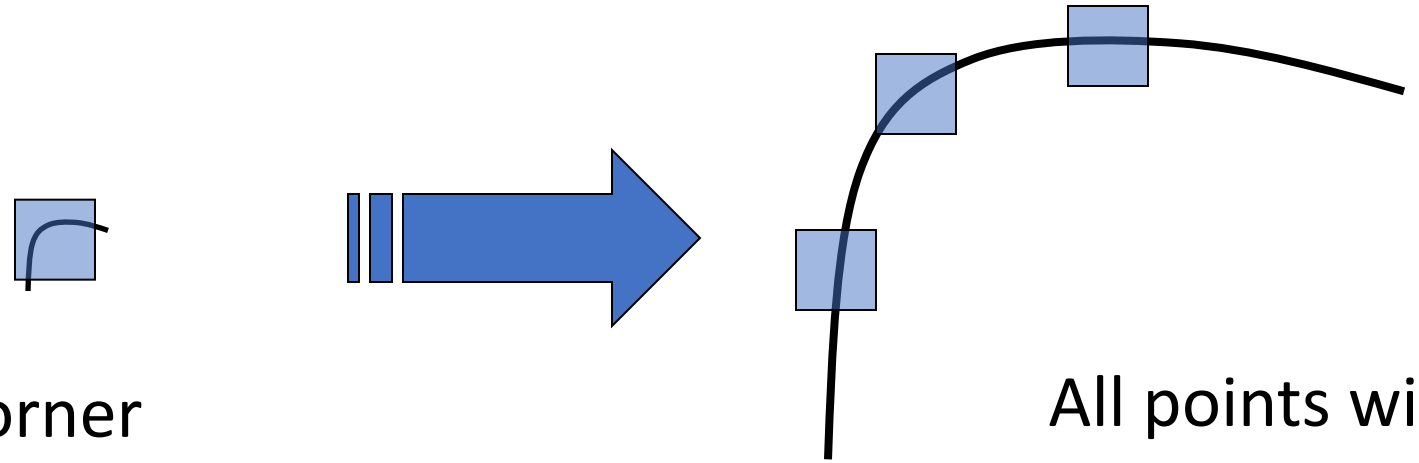
Image Rotation



Second moment ellipse rotates but its shape (i.e., eigenvalues) remains the same.

Corner location is covariant w.r.t. rotation

Scaling



Corner

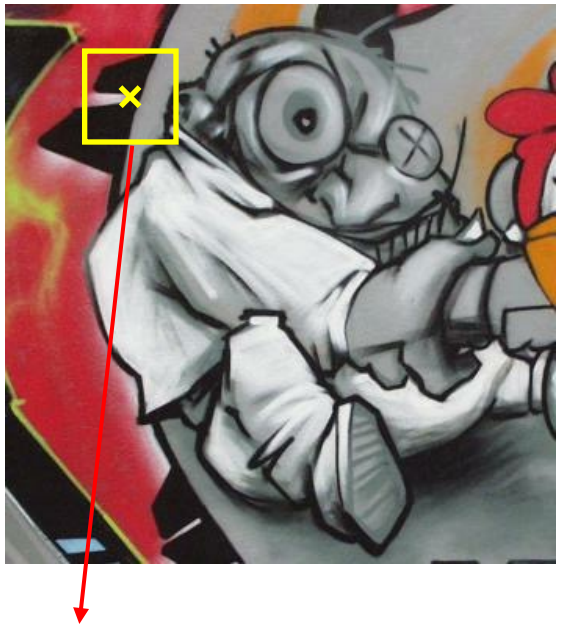
All points will be incorrectly classified as edges

Corner location is not covariant to scaling!

How can we make a feature detector scale-invariant?

Multi-scale detection

Automatic Scale Selection



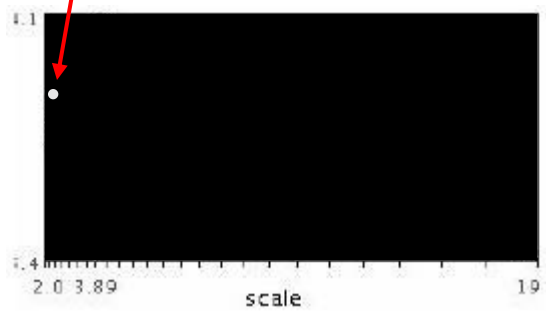
How to find patch sizes at which f response is equal?

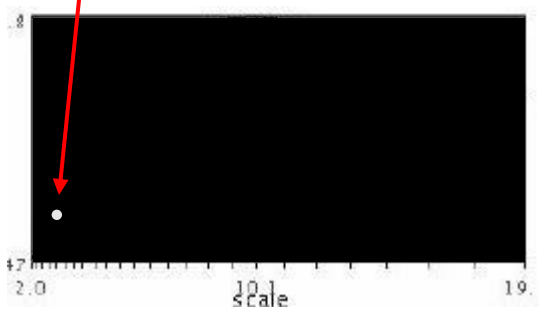
What is a good f ?

Automatic Scale Selection

- Function responses for increasing scale (scale signature)

Response of some function *f*

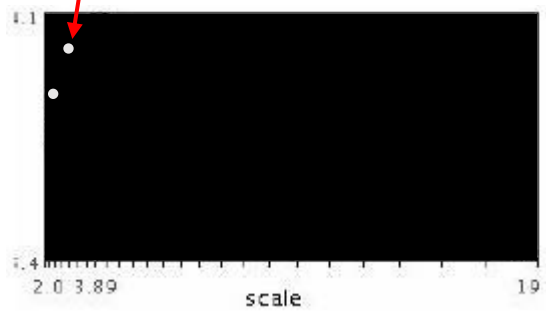


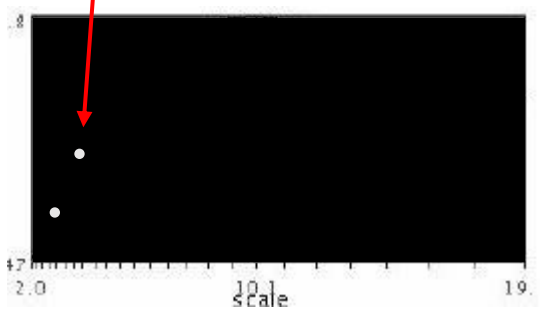


Automatic Scale Selection

- Function responses for increasing scale (scale signature)

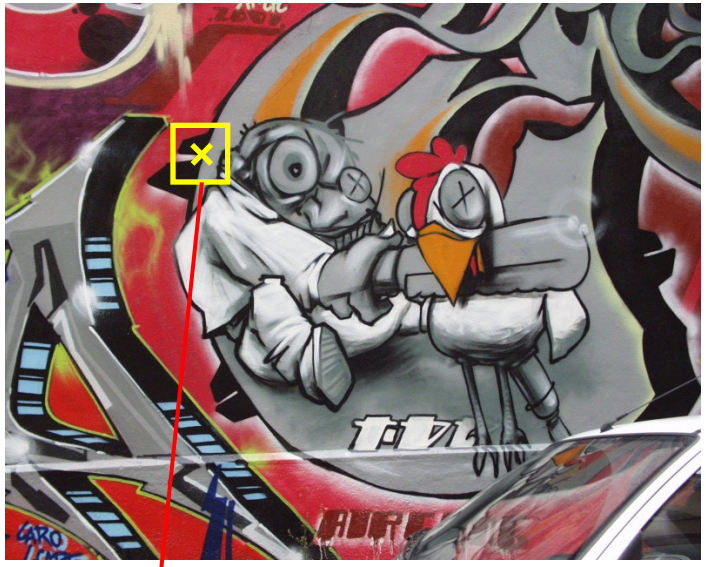
Response of some function *f*



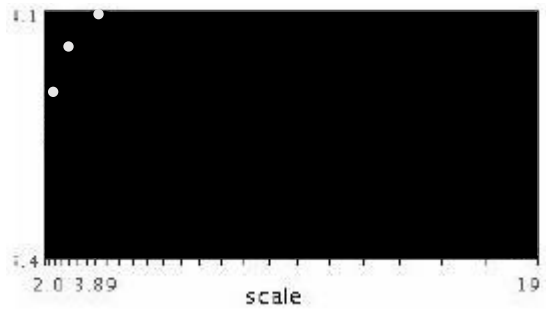


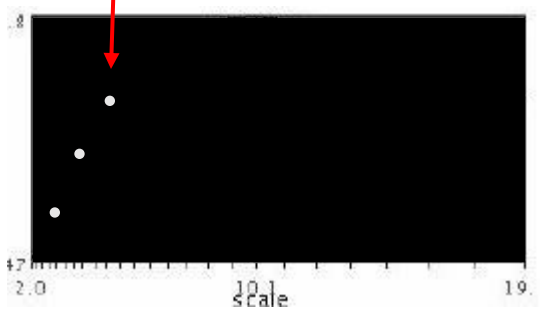
Automatic Scale Selection

- Function responses for increasing scale (scale signature)



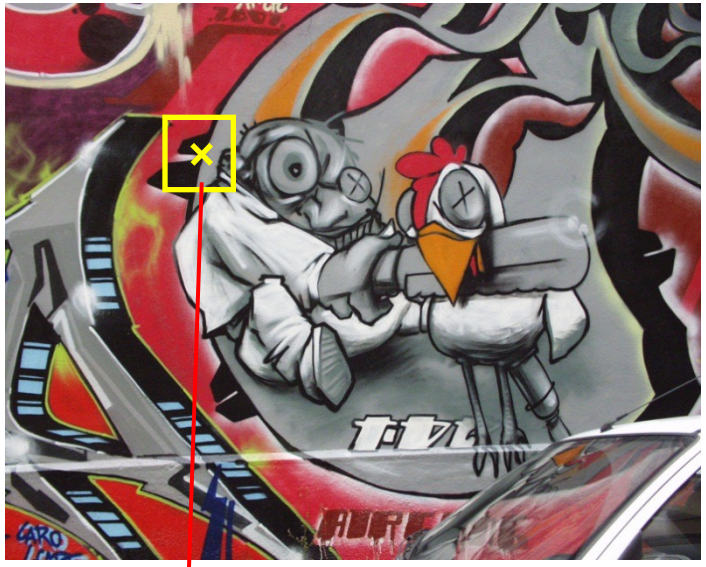
Response of some function *f*



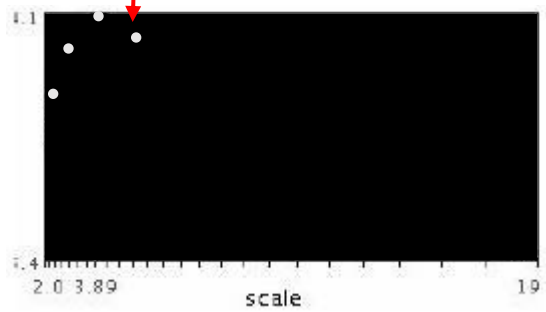


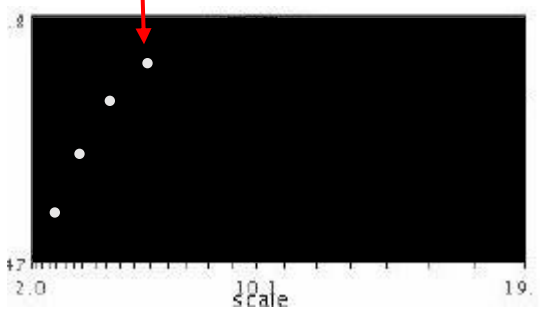
Automatic Scale Selection

- Function responses for increasing scale (scale signature)



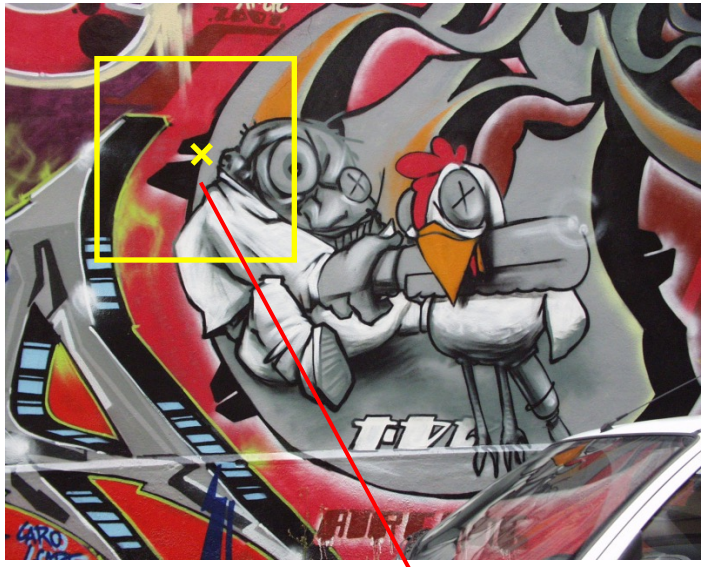
Response of some function *f*



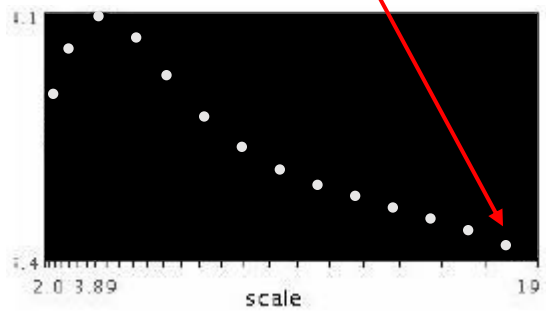


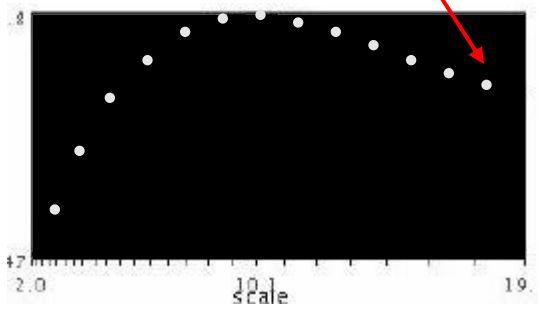
Automatic Scale Selection

- Function responses for increasing scale (scale signature)



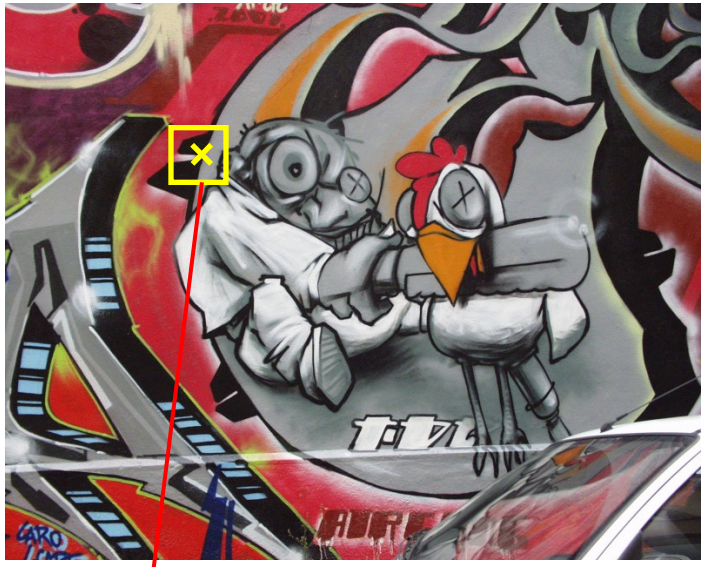
Response of some function *f*



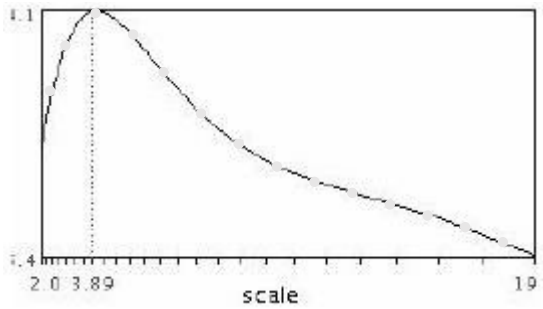


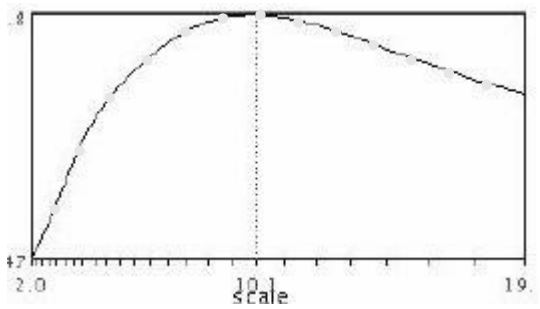
Automatic Scale Selection

- Function responses for increasing scale (scale signature)



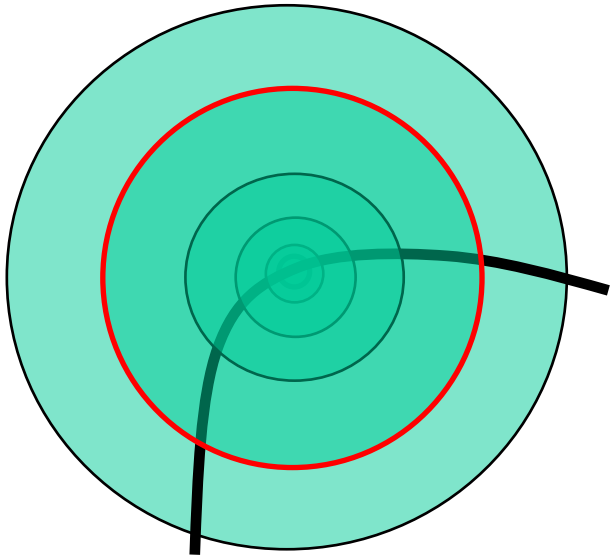
Response of some function *f*

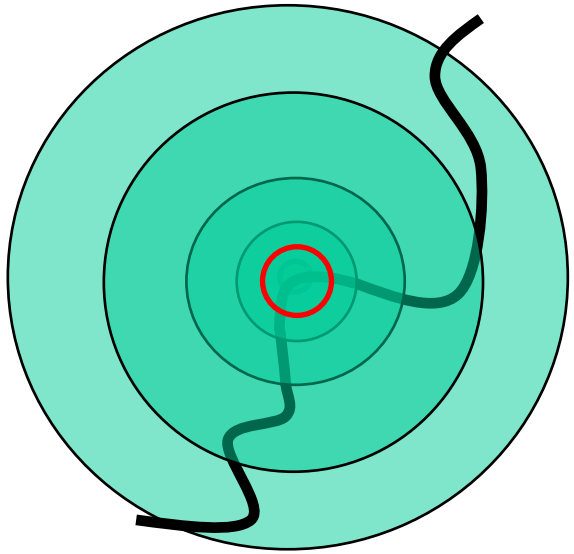


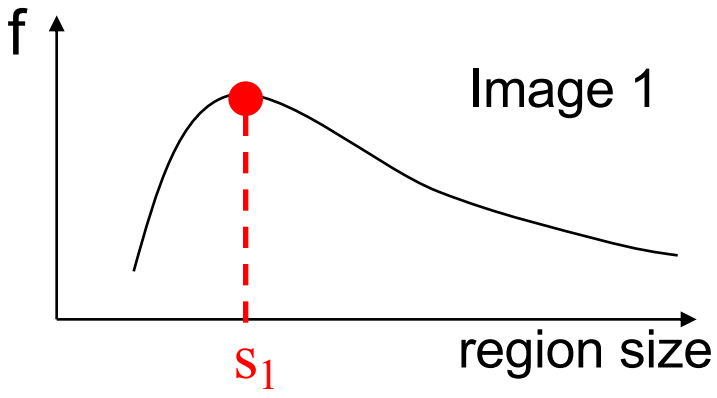


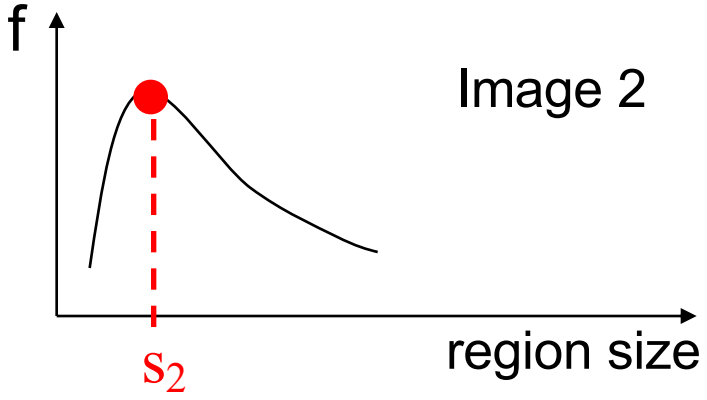
Intuitively…

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



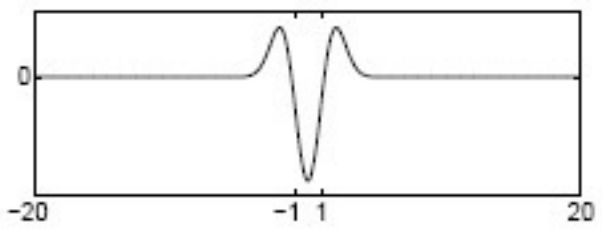


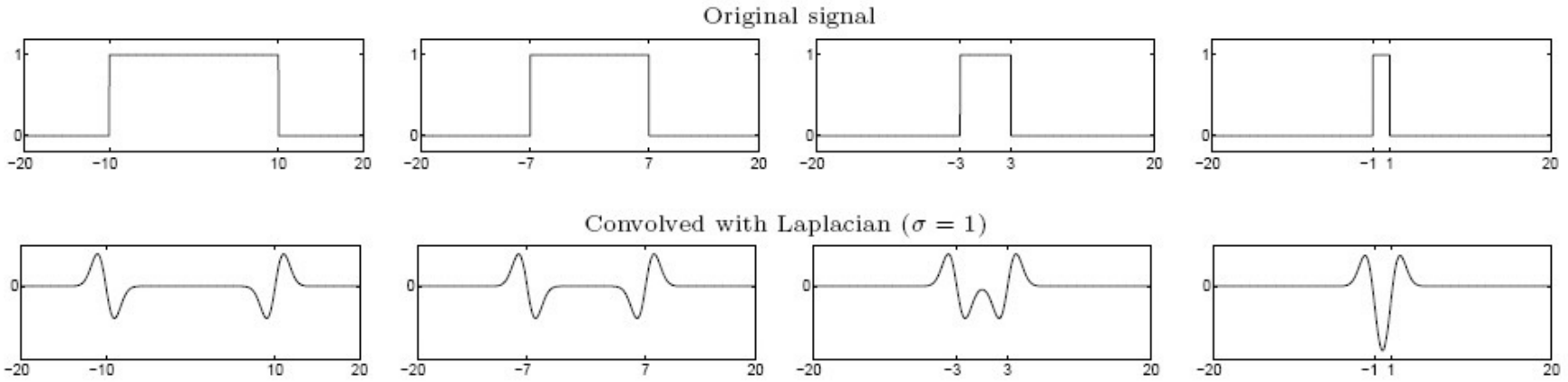




Formally…

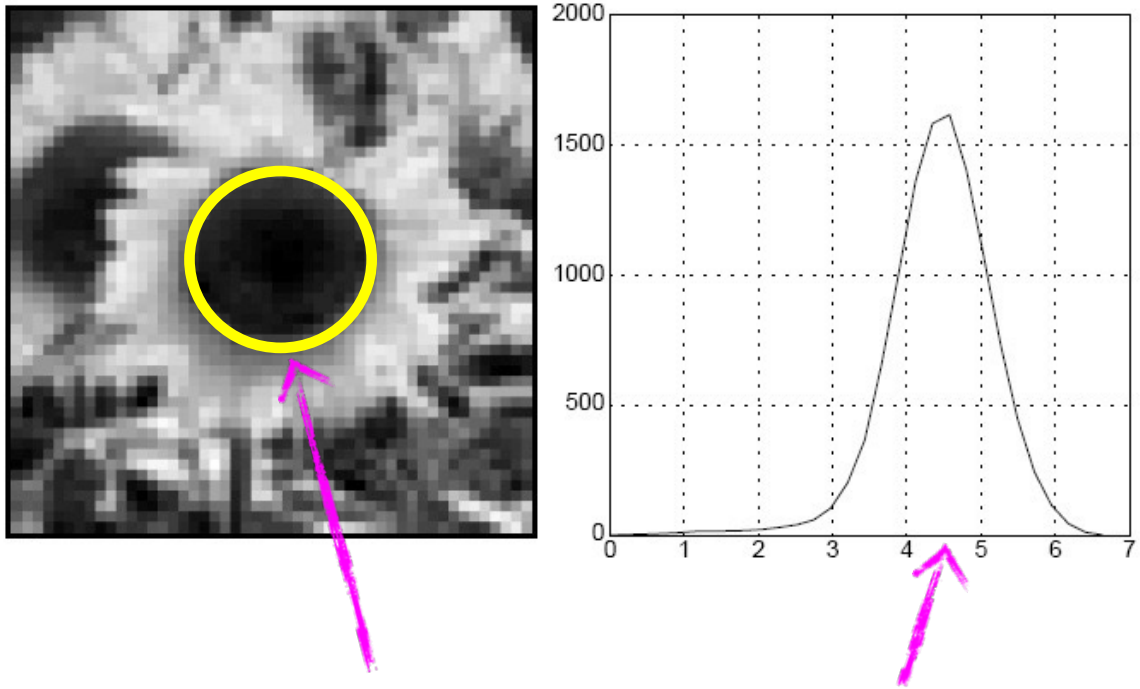
Laplacian filter





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

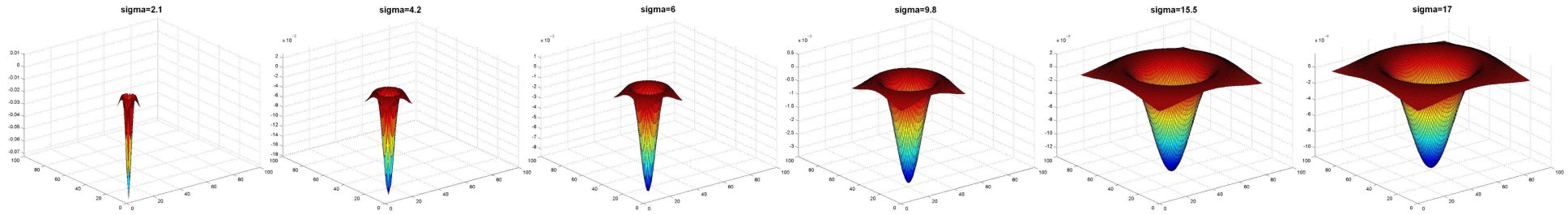
characteristic scale - the scale that produces peak filter response



characteristic scale

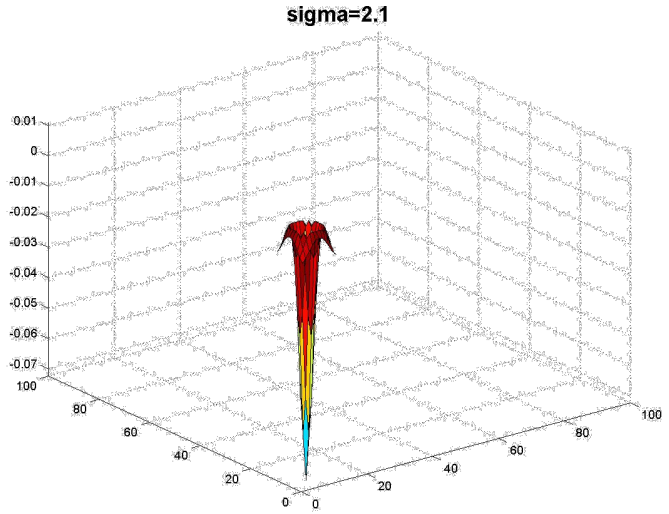
we need to search over characteristic scales

What happens if you apply different Laplacian filters?

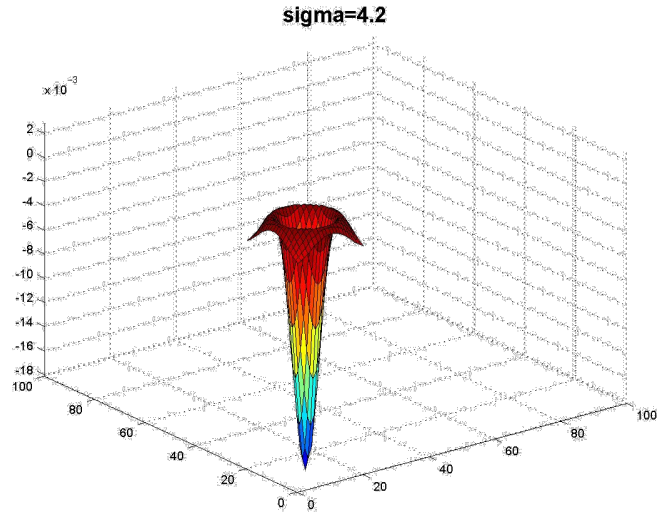


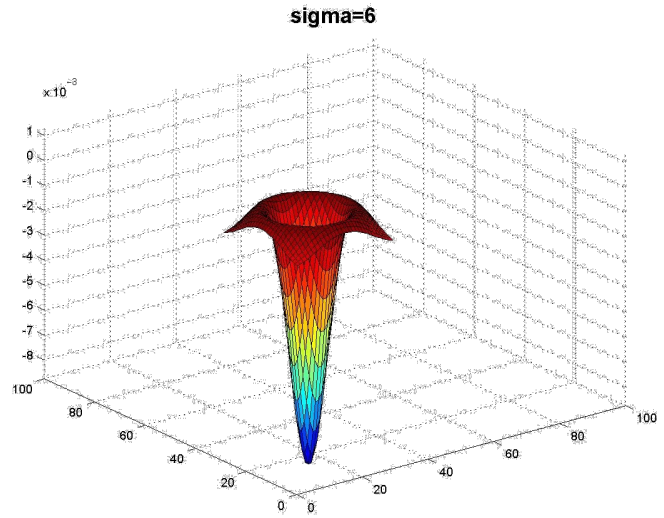
Full size

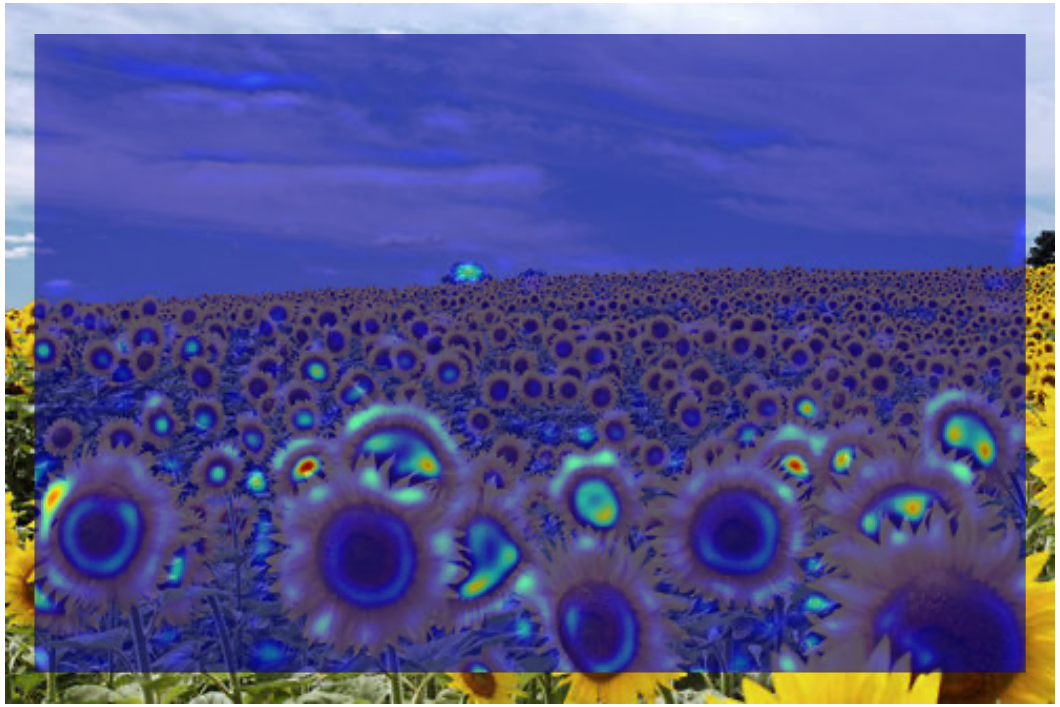
3/4 size

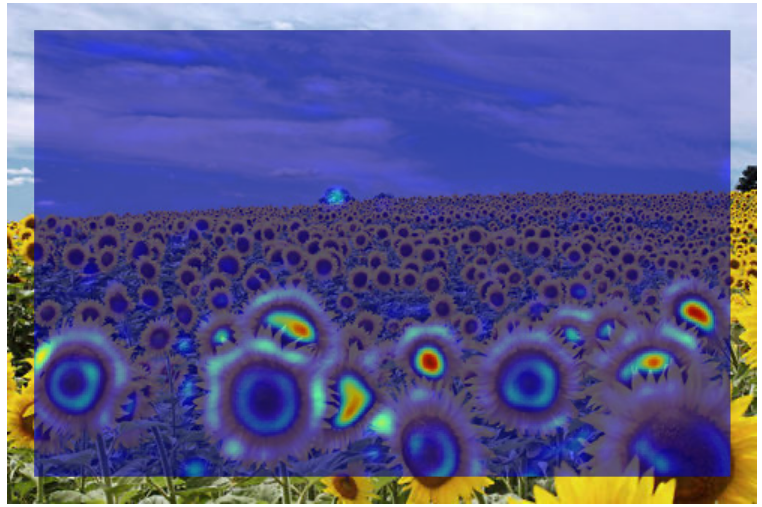


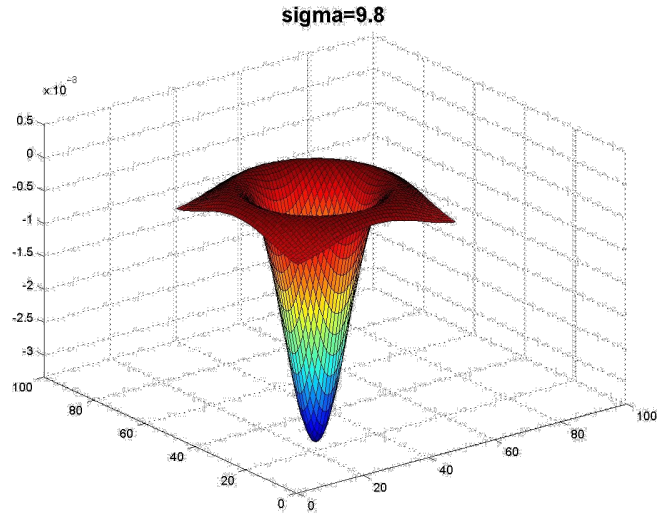
jet color scale
blue: low, red: high

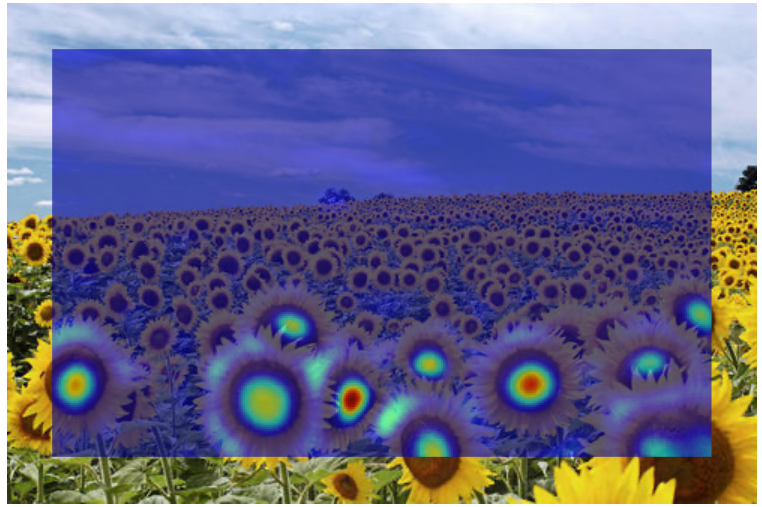

sigma=4.2
x 10^-3
2
0
-2
-4
-6
-8
-10
-12
-14
-16
-18
100
80
60
40
20
0
0
20
40
60
80
100

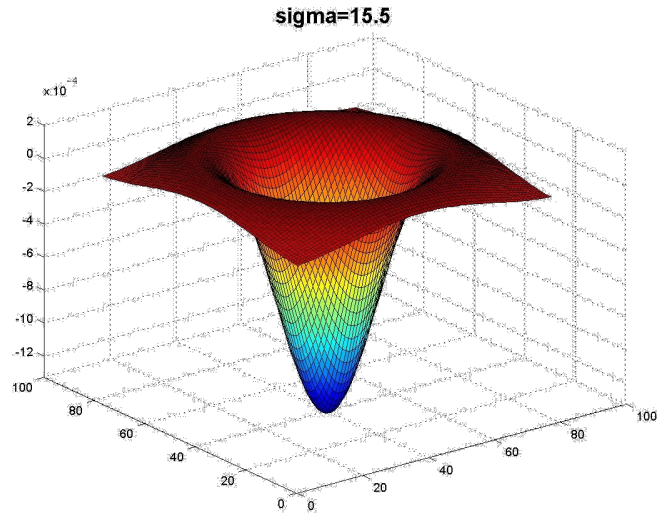

sigma=6
x 10^-3
1
0
-1
-2
-3
-4
-5
-6
-7
-8
100
80
60
40
20
0
0
20
40
60
80
100

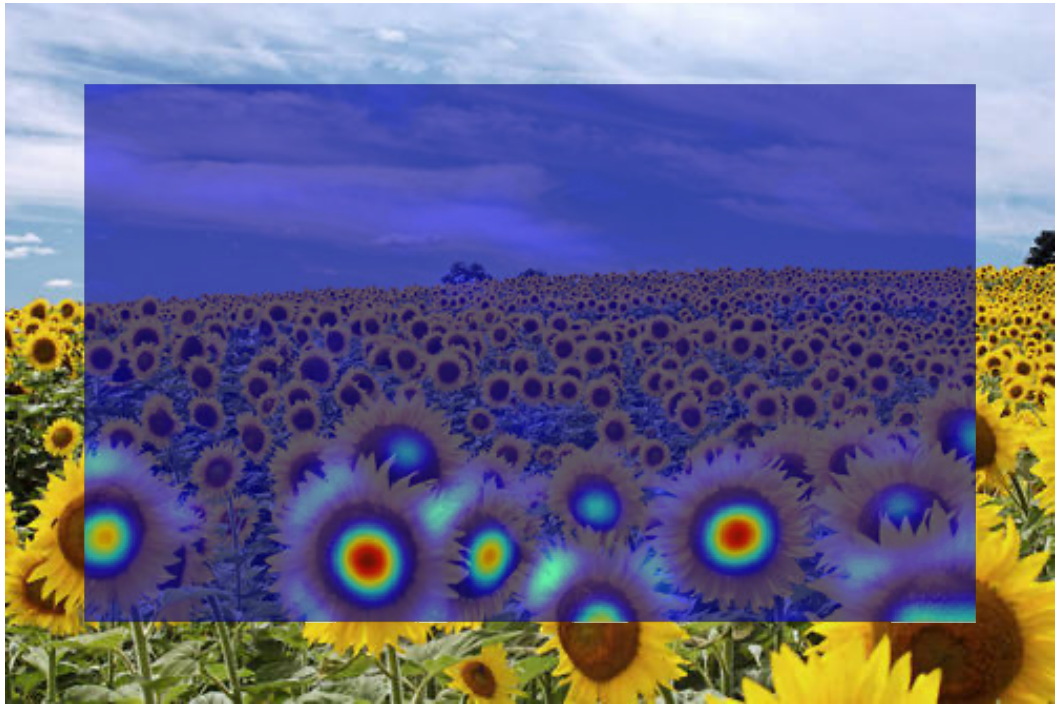


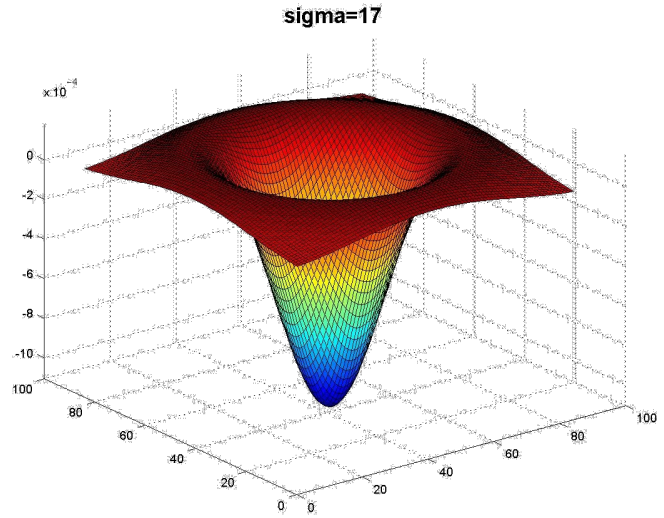


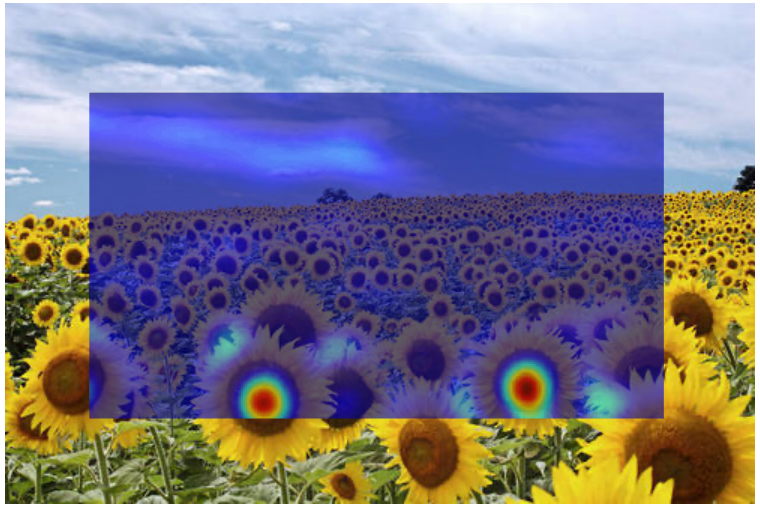

sigma=9.8
x 10^-3
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
100
80
60
40
20
0
0
20
40
60
80
100




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



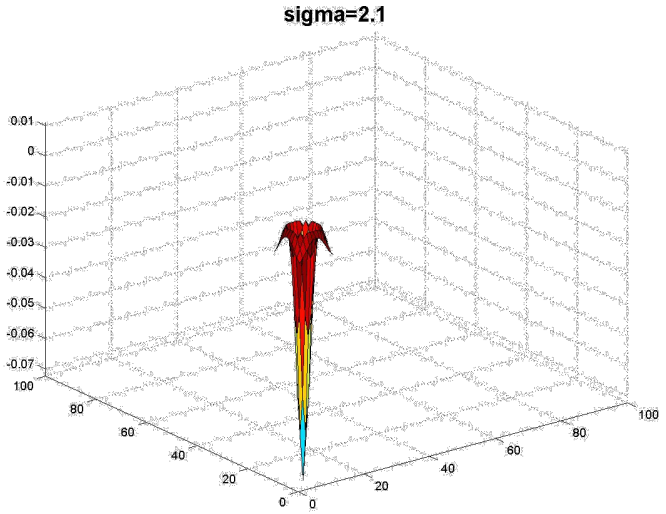

sigma=17
x 10^-4
0
-2
-4
-6
-8
-10
100
80
60
40
20
0
0
20
40
60
80
100

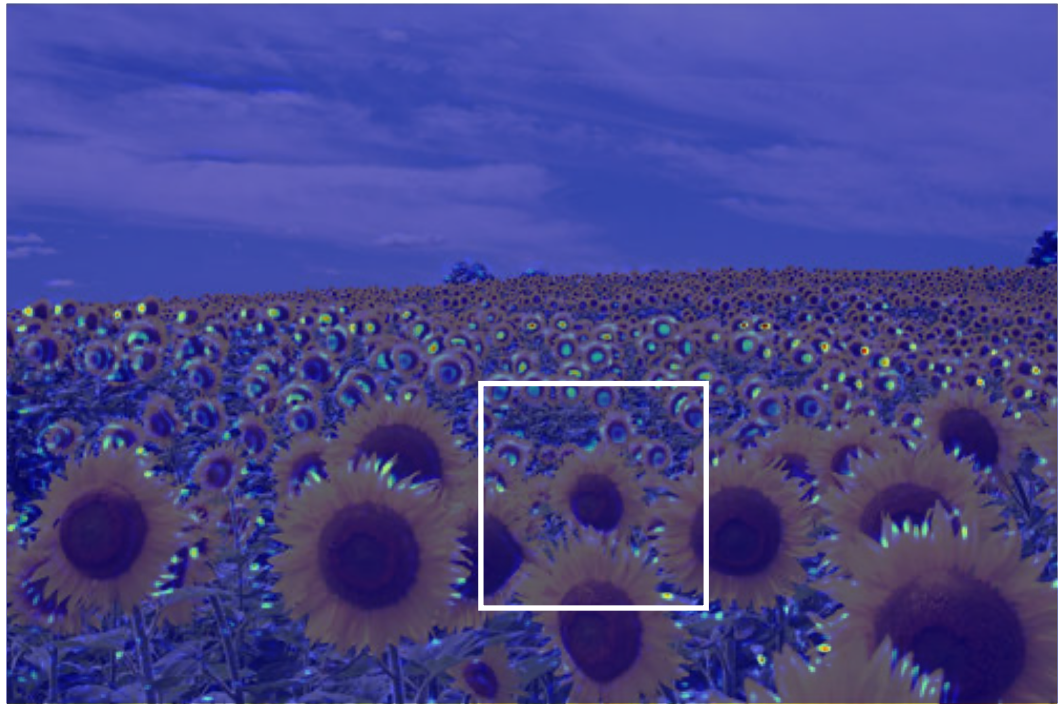


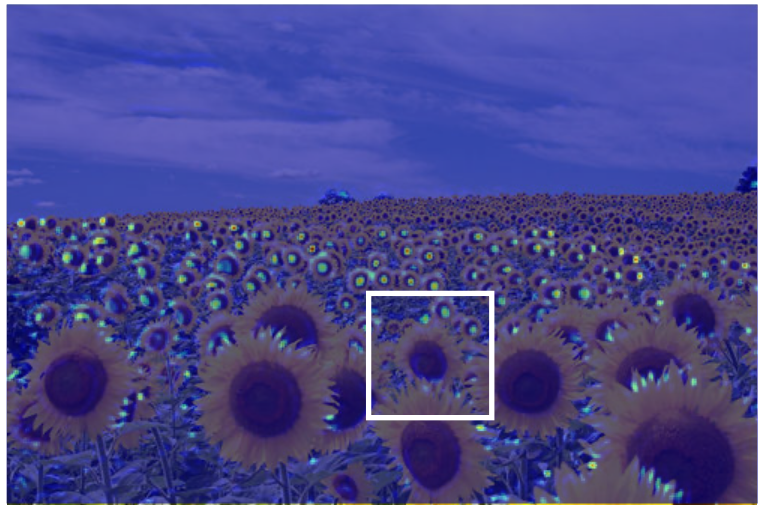
What happened when you applied different Laplacian filters?

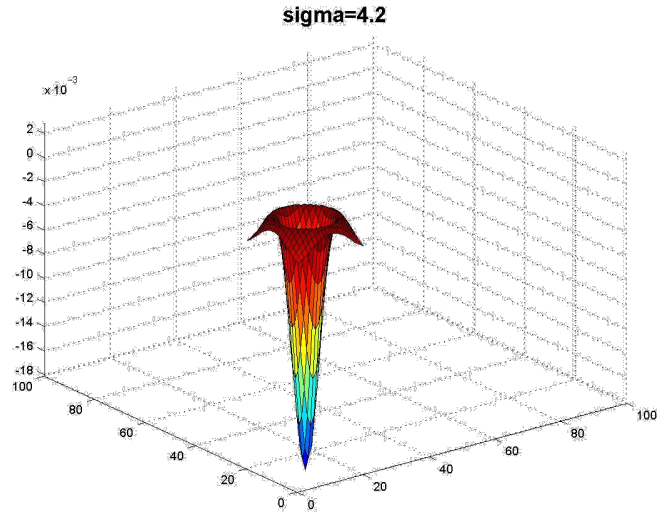
Full size

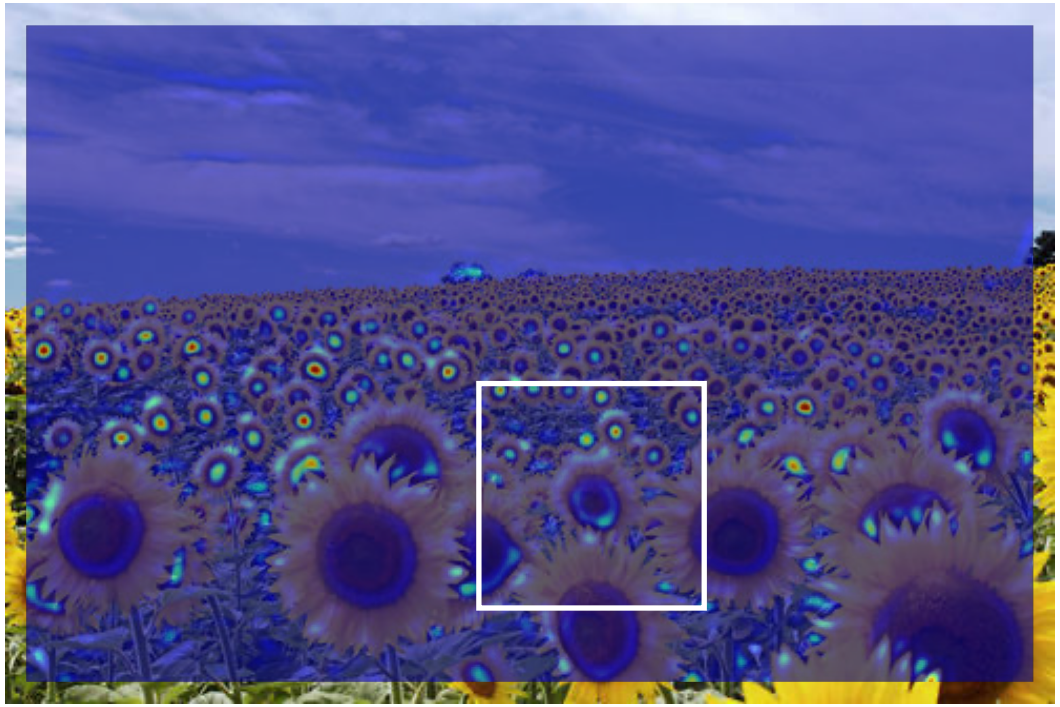
3/4 size

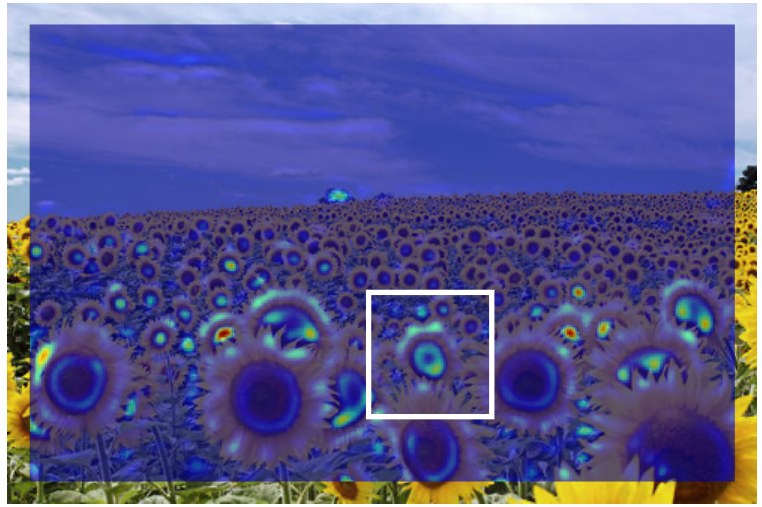

sigma=2.1
0.01
0
-0.01
-0.02
-0.03
-0.04
-0.05
-0.06
-0.07
100
80
60
40
20
0
0
20
40
60
80
100

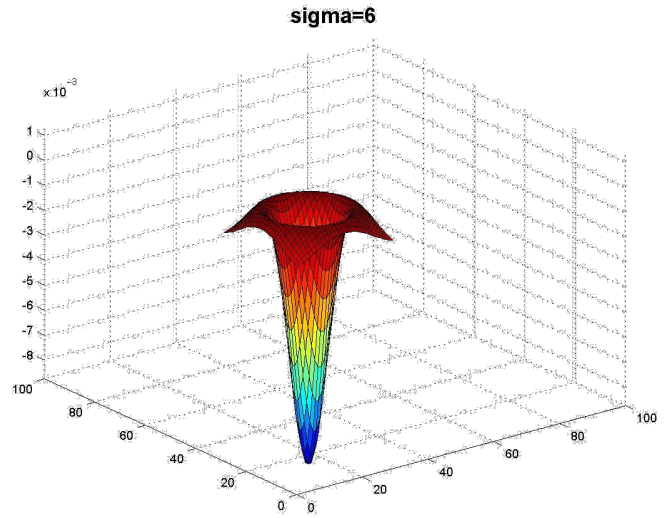


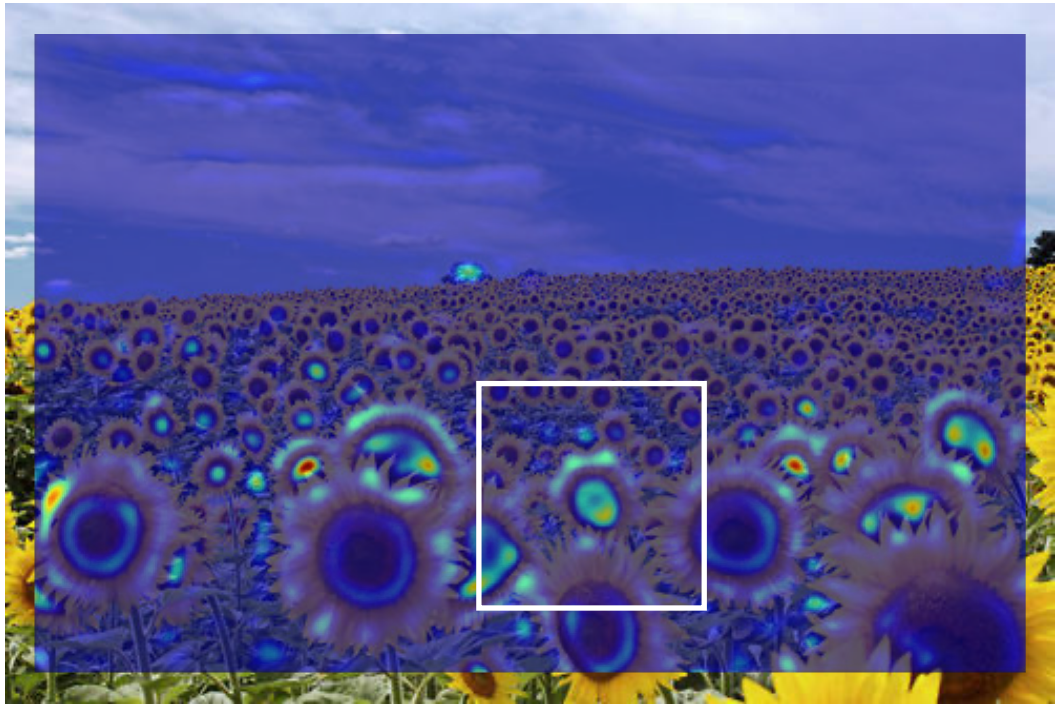



sigma=4.2
x 10^-3
2
0
-2
-4
-6
-8
-10
-12
-14
-16
-18
100
80
60
40
20
0
0
20
40
60
80
100

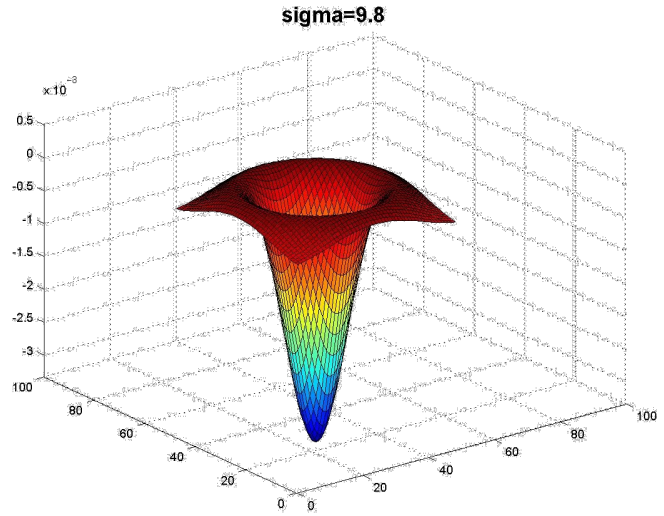




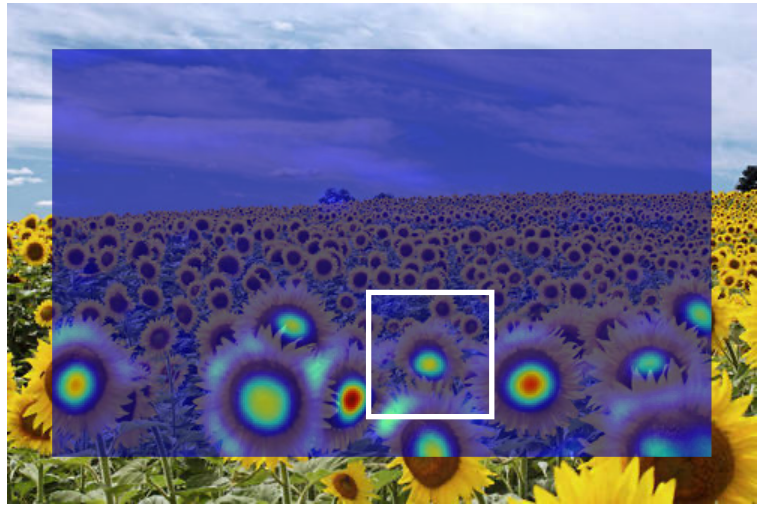

sigma=6
x 10^-3
1
0
-1
-2
-3
-4
-5
-6
-7
-8
100
80
60
40
20
0
0
20
40
60
80
100

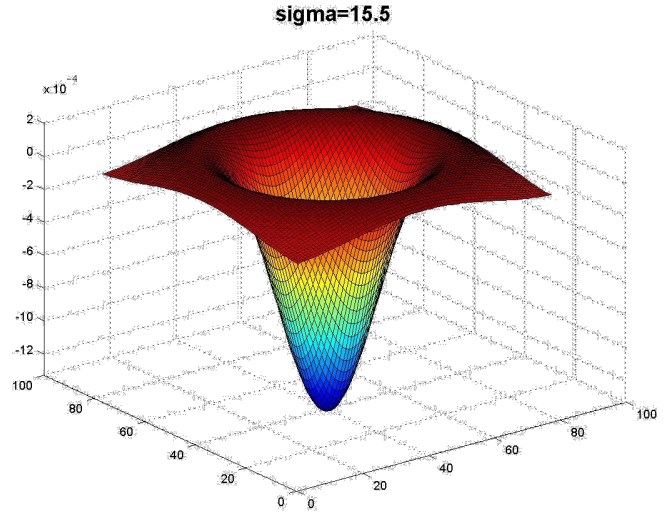


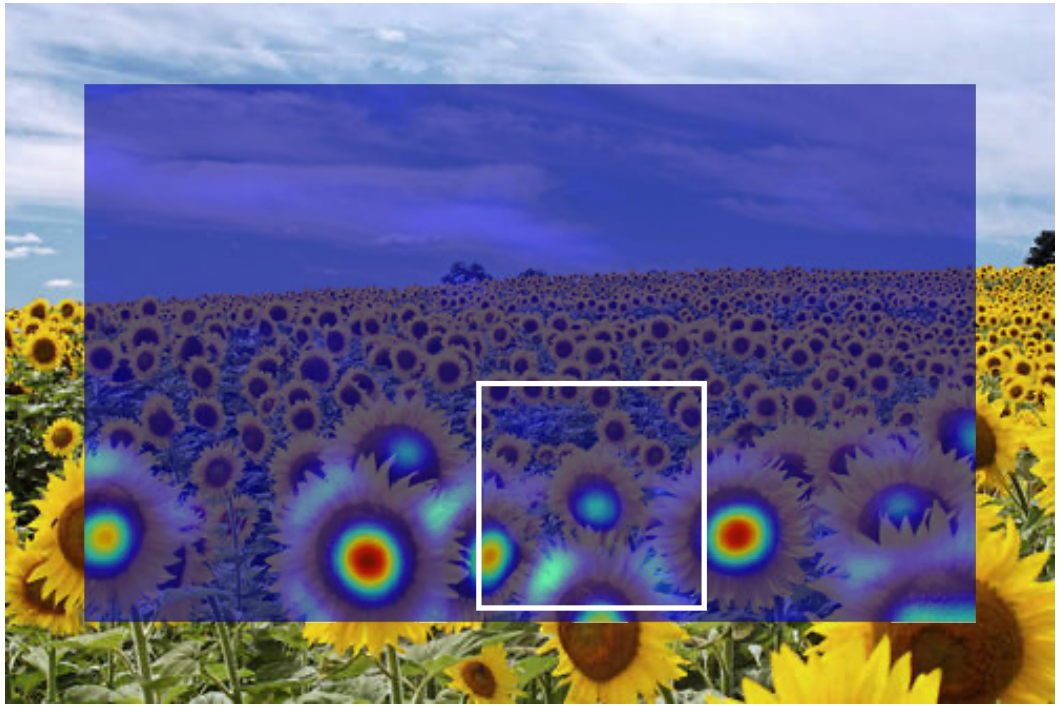

peak!

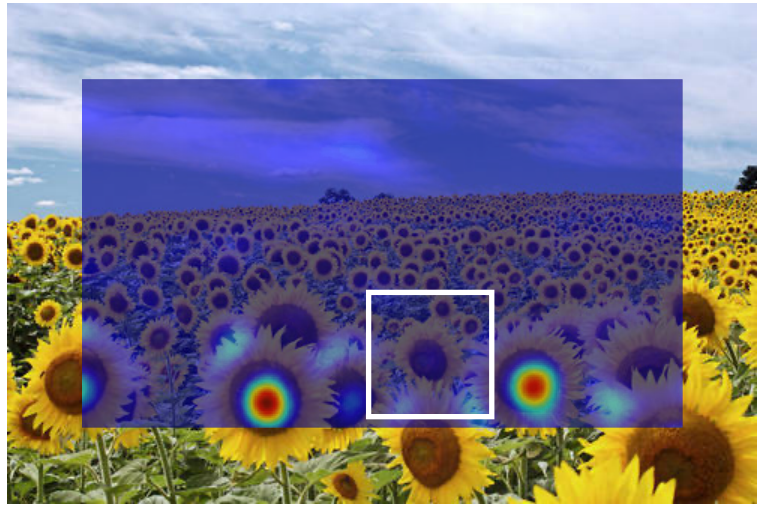

sigma=9.8
x 10^-3
0.5
0
-0.5
-1
-1.5
-2
-2.5
-3
100
80
60
40
20
0
0
20
40
60
80
100

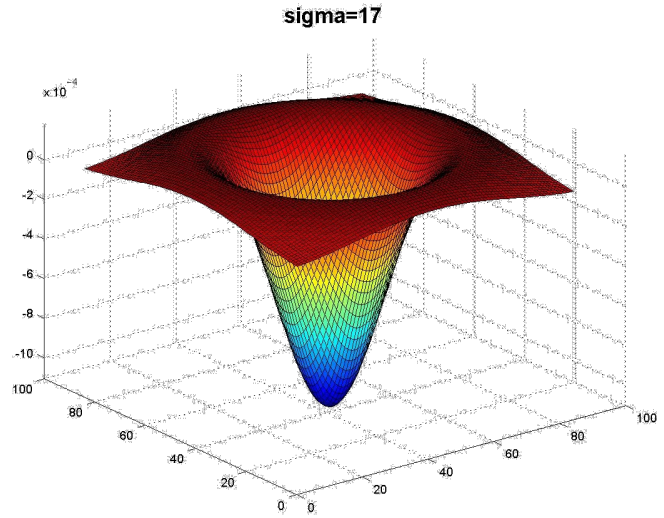

peak!

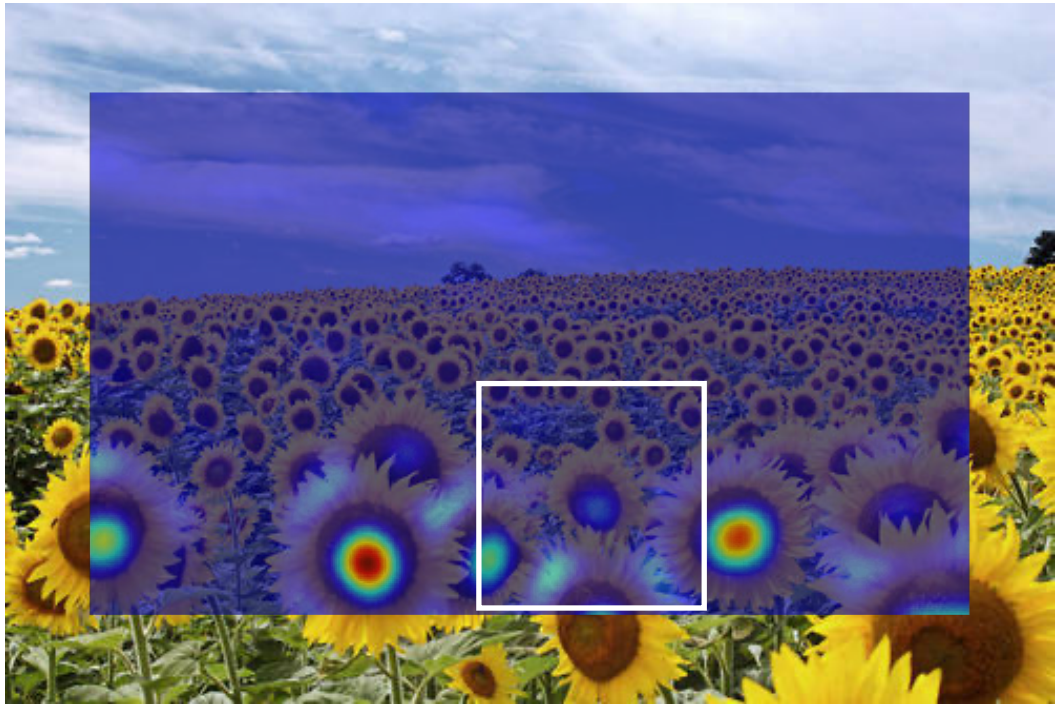


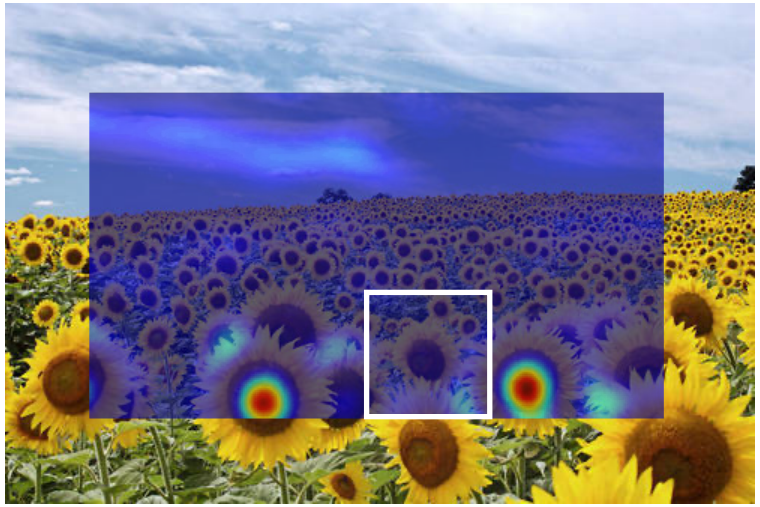

sigma=15.5
x 10^-4
2
0
-2
-4
-6
-8
-10
-12
100
80
60
40
20
0
0
20
40
60
80
100






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

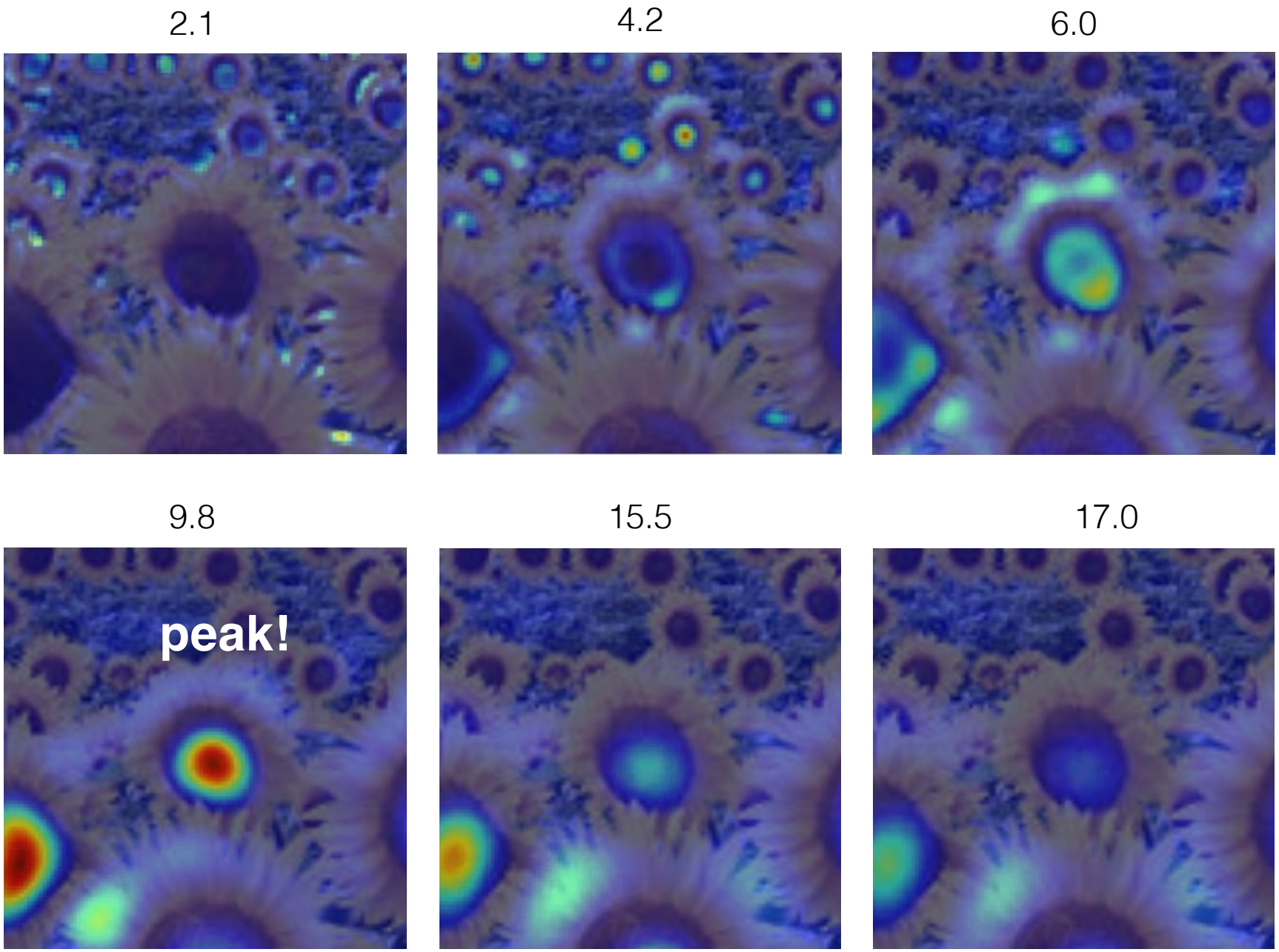


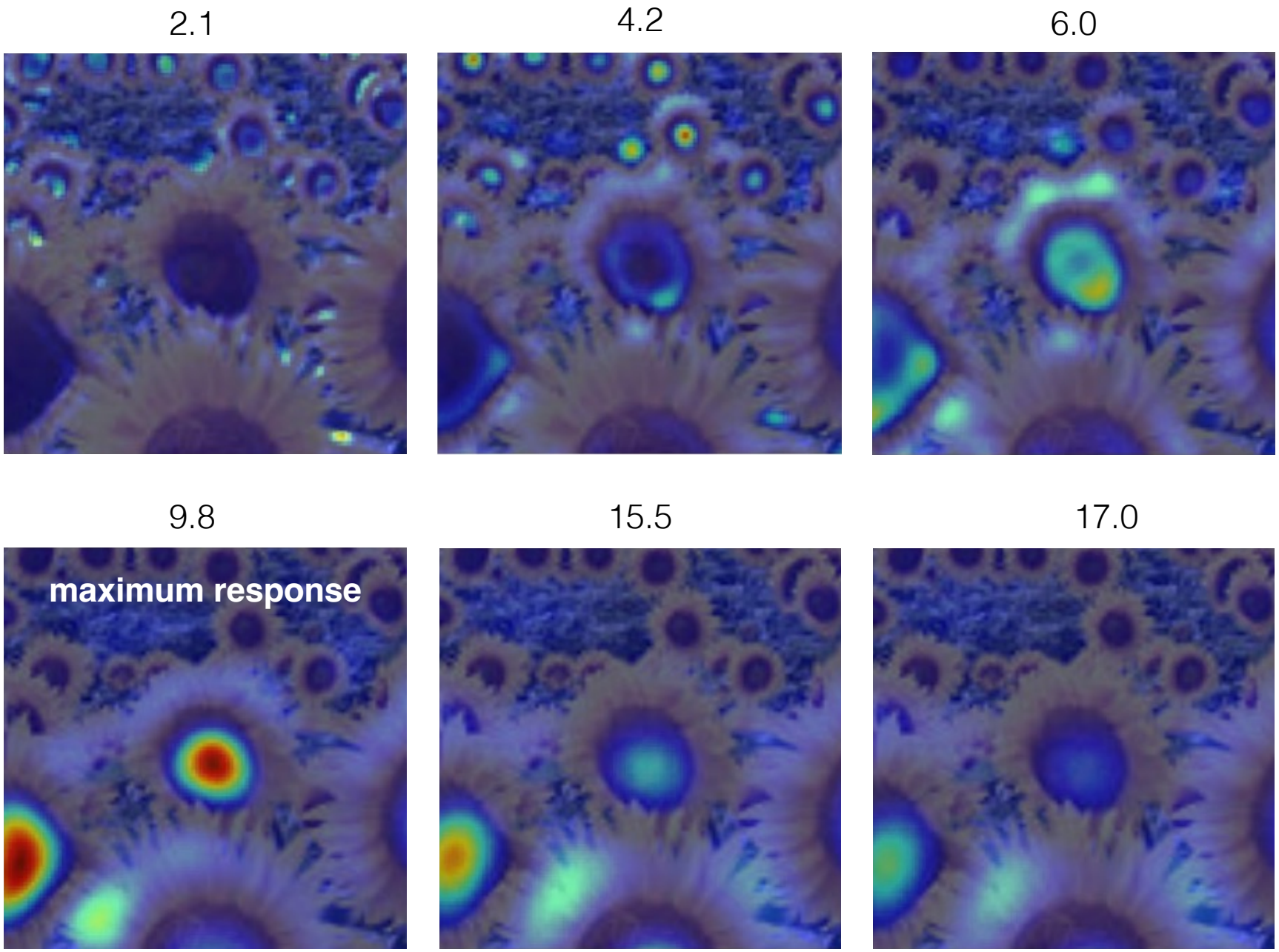


What happened when you applied different Laplacian filters?

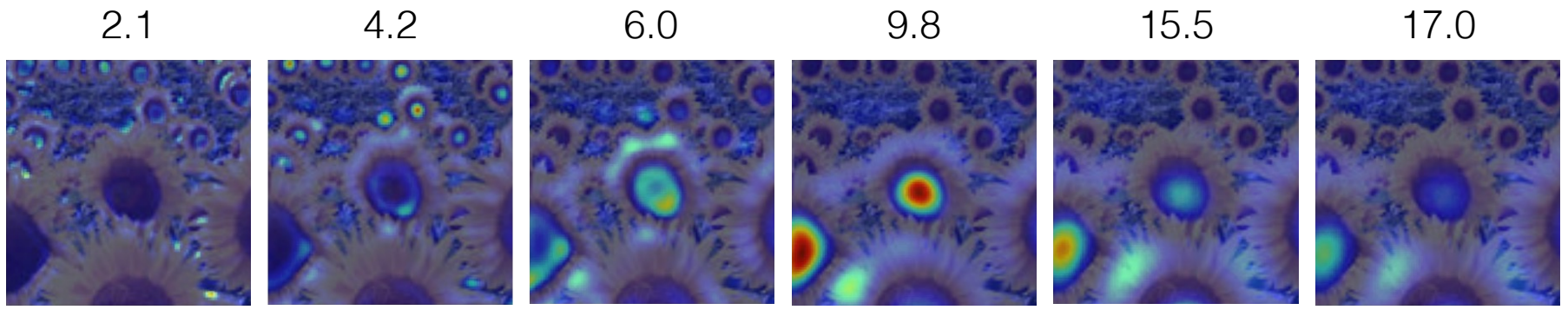
Full size

3/4 size

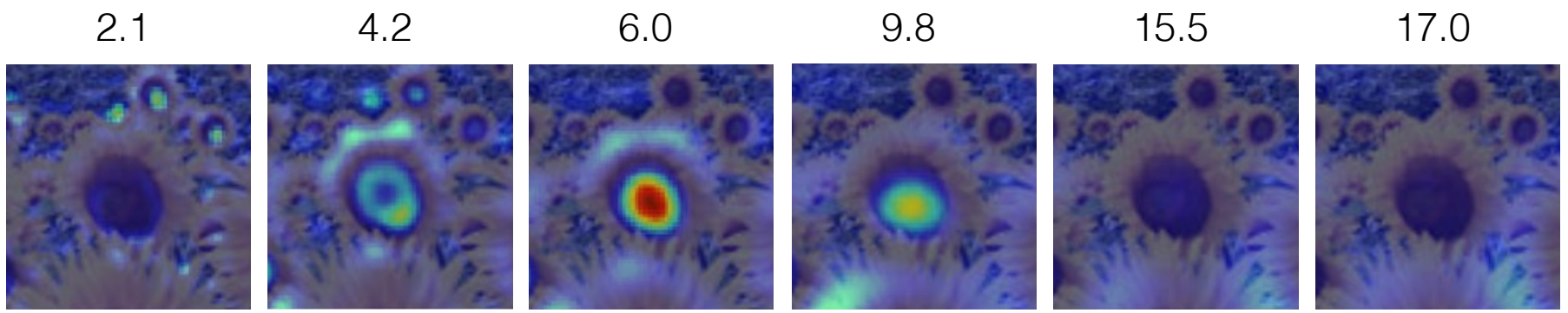

2.1
4.2
6.0
9.8
peak!
15.5
17.0


2.1
4.2
6.0
9.8
maximum response
15.5
17.0

optimal scale

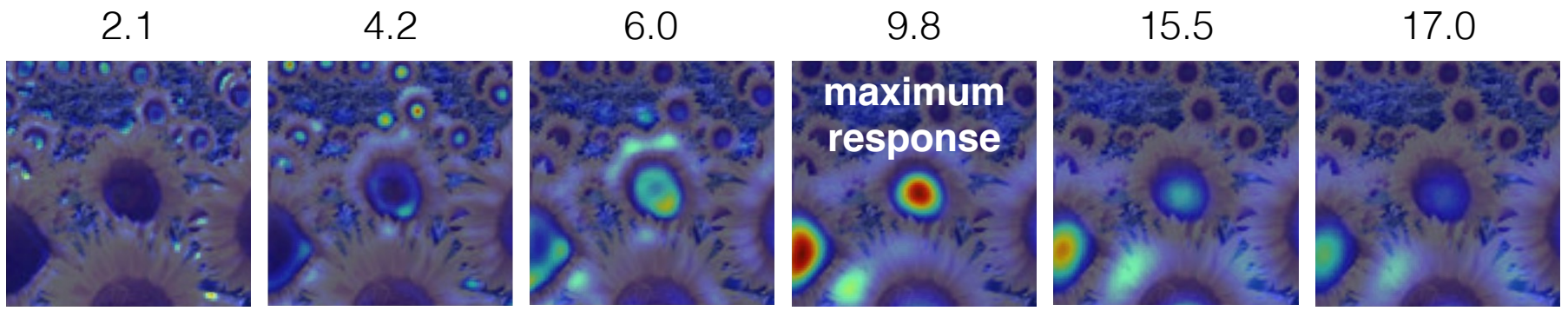


Full size image

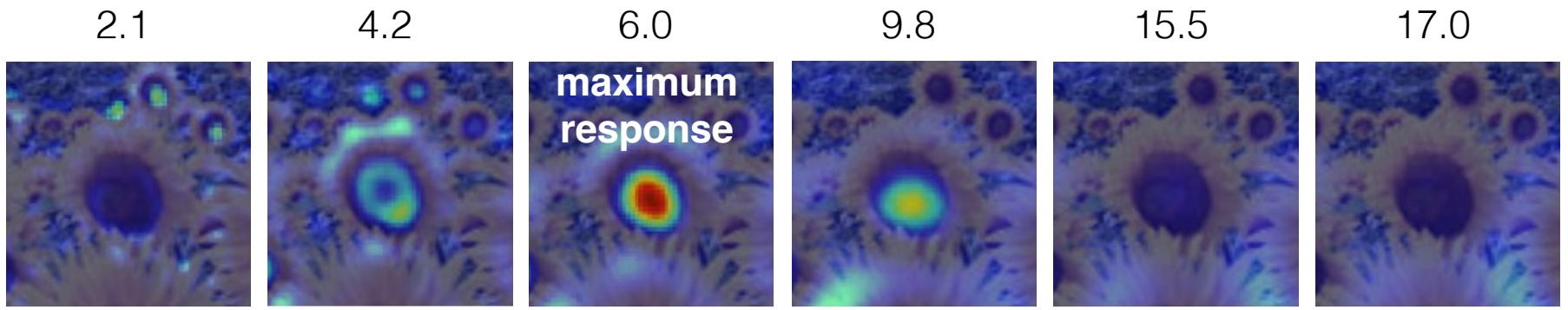


3/4 size image

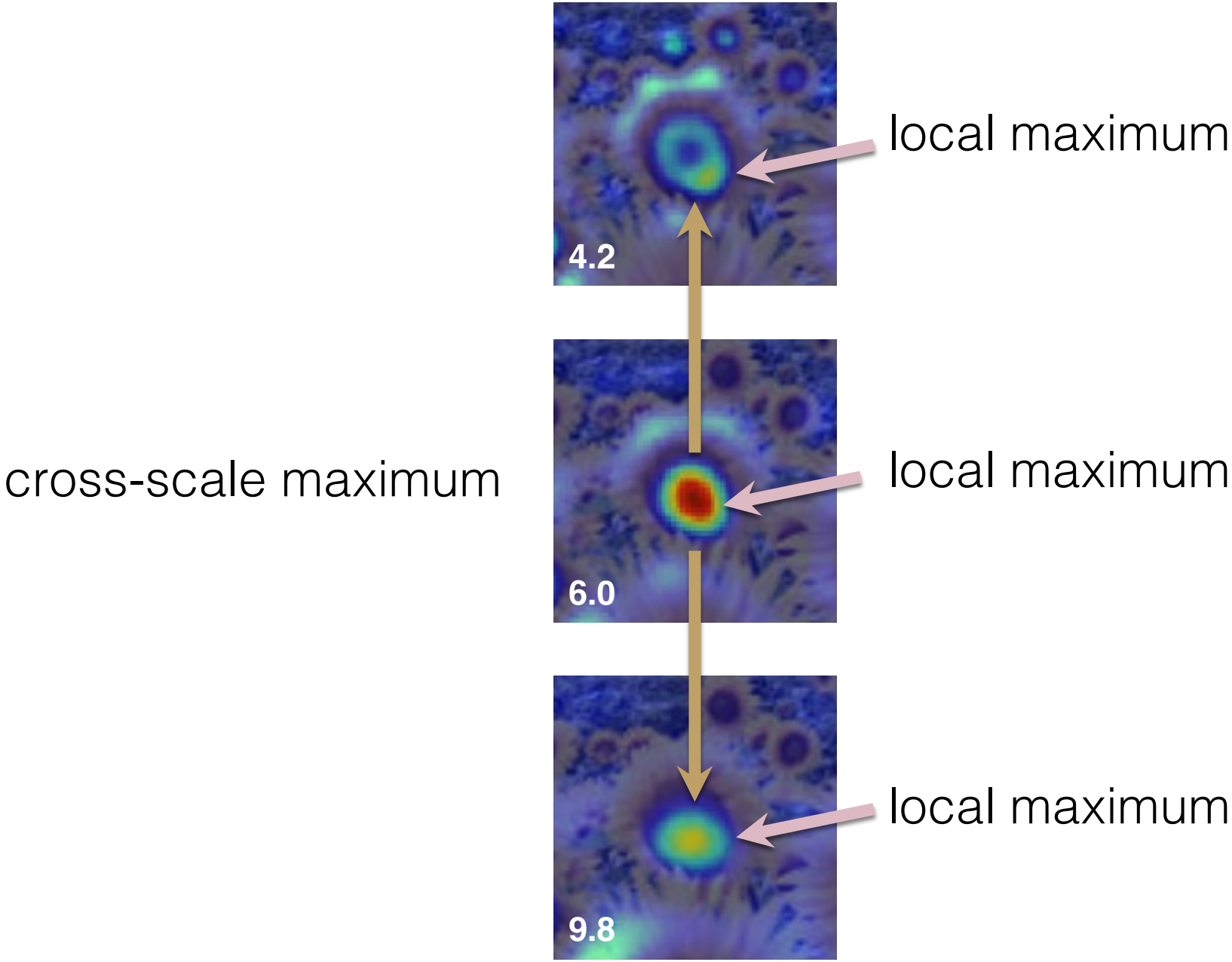
optimal scale



Full size image



3/4 size image


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

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)