MLRF Lecture 02

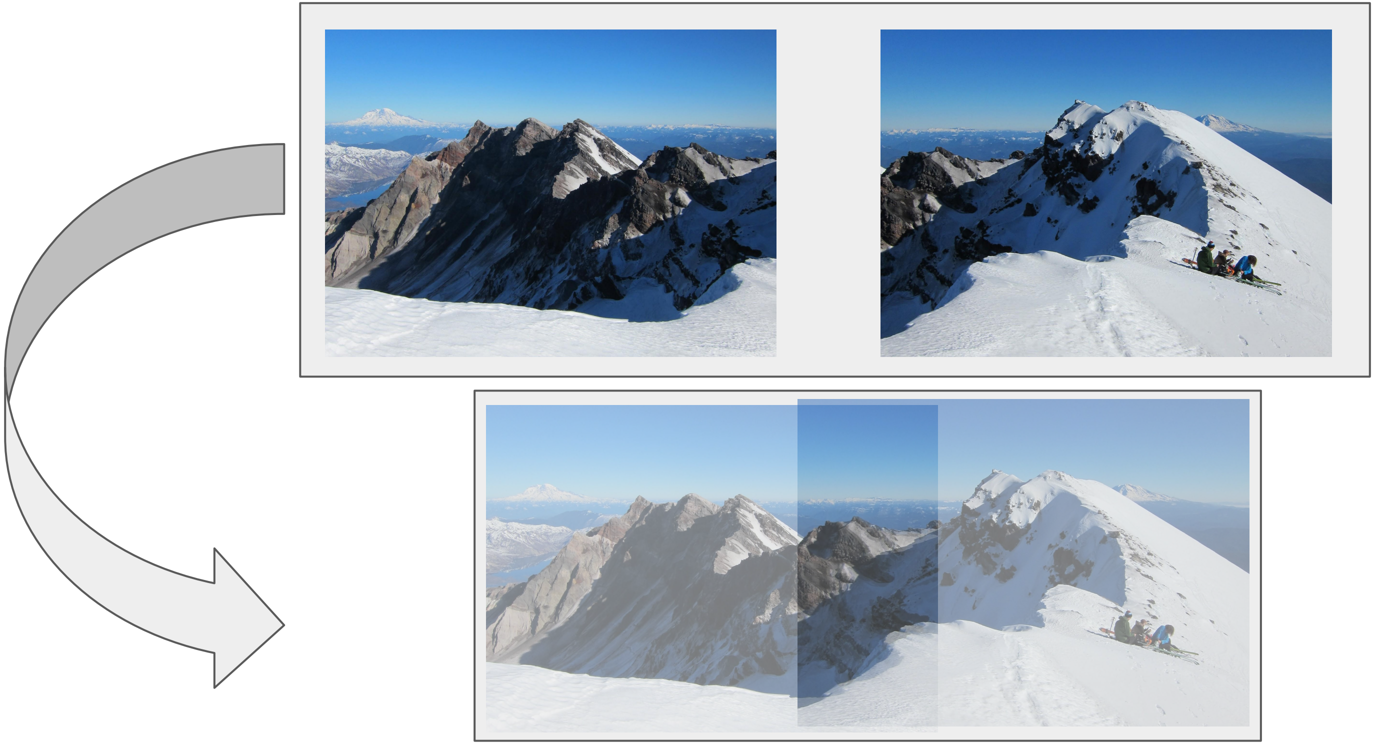
J. Chazalon, LRDE/EPITA, 2021

Local feature detectors

Lecture 02 part 04

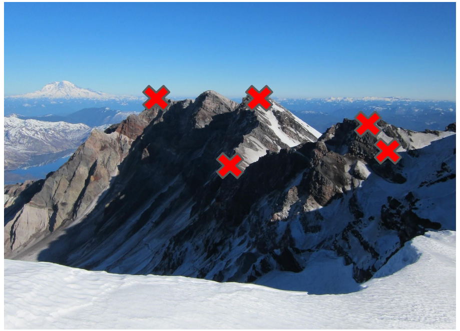
Introduction

How are panorama pictures created from multiple pictures?



Introduction

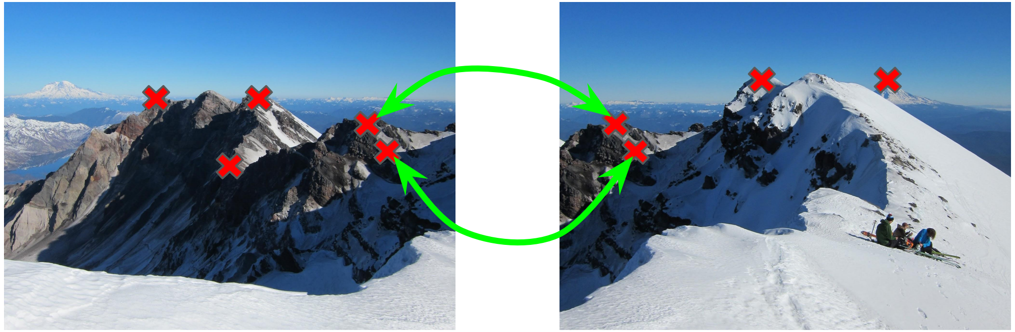
How are panorama pictures created from multiple pictures?



1. Detect small parts invariant under viewpoint change: “Keypoints”

Introduction

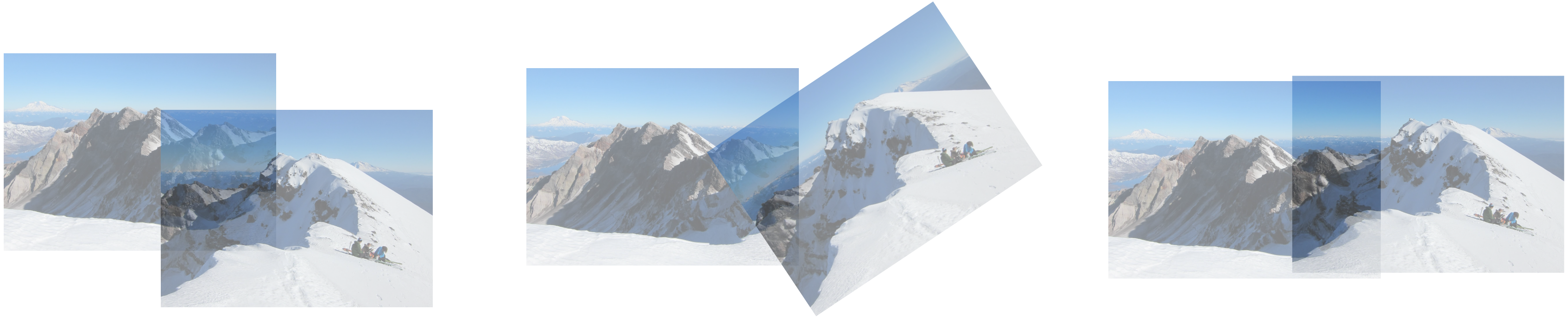
How are panorama pictures created from multiple pictures?



1. Detect small parts invariant under viewpoint change: **<u>keypoints</u>**
2. Find pairs of matching keypoints using a **<u>description</u>** of their neighborhood

Introduction

How are panorama pictures created from multiple pictures?



1. Detect small parts invariant under viewpoint change: **keypoints**
2. Find pairs of matching keypoints using a **description** of their neighborhood
3. Compute the **most likely transformation** to blend images together

The need for local feature detectors

While **dense computation** of local feature descriptors is possible (grid of points), this is **rarely used in practice** (lots of computations, lots of useless features).

Detection = Find **anchors** to describe a **feature of interest**.

- Edge / line
- Area around a corner / a stable point
- Blob (area of variable size)

A good feature of interest is **stable over the perturbations** our signal will face:

- Translation, rotation, zoom, perspective
- Illumination changes
- Noise, compression
- …

Some classical detectors

Edge (gradient detectors)

- Sobel
- Canny

Corner

- Harris & Stephens *and variants*
- FAST
- Laplacian of Gaussian, Difference of Gaussian, Determinant of Hessian

Blob

- MSER

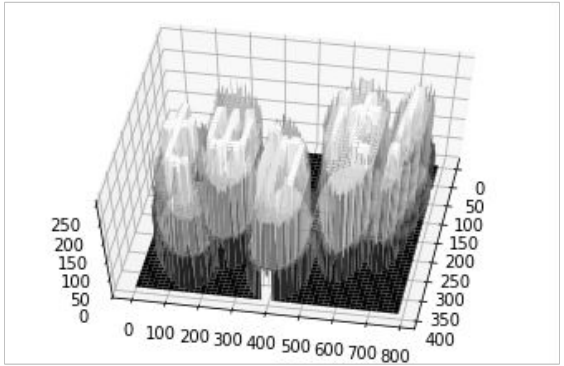
Edge detectors

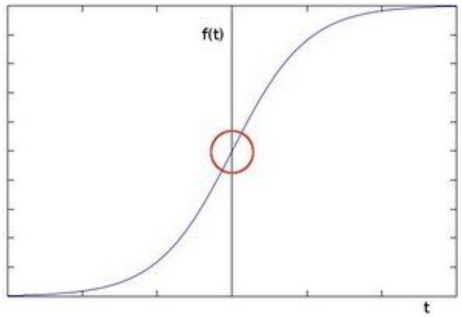
What’s an edge?

Image is a function

Edges are rapid changes in this function

The derivative of a function exhibits the edges





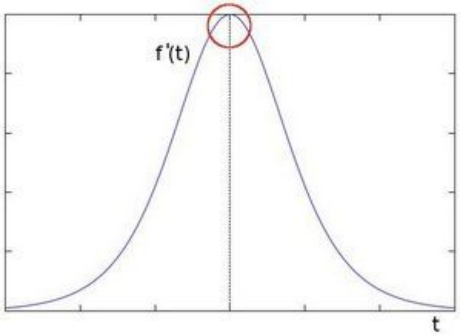


Image derivatives

Recall:

$$f'(a) = \lim_{h \to 0} \frac{f(a+h) - f(a-h)}{2h}$$

We don’t have an “actual” function, must estimate

Possibility: set h = 1

Apply filter $\boxed{-1 \;|\; 0 \;|\; +1}$ to the image
(x gradient)

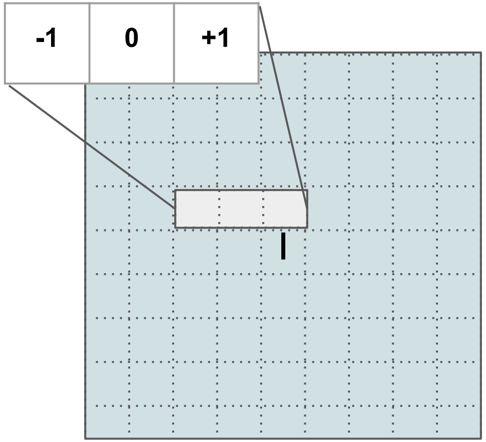
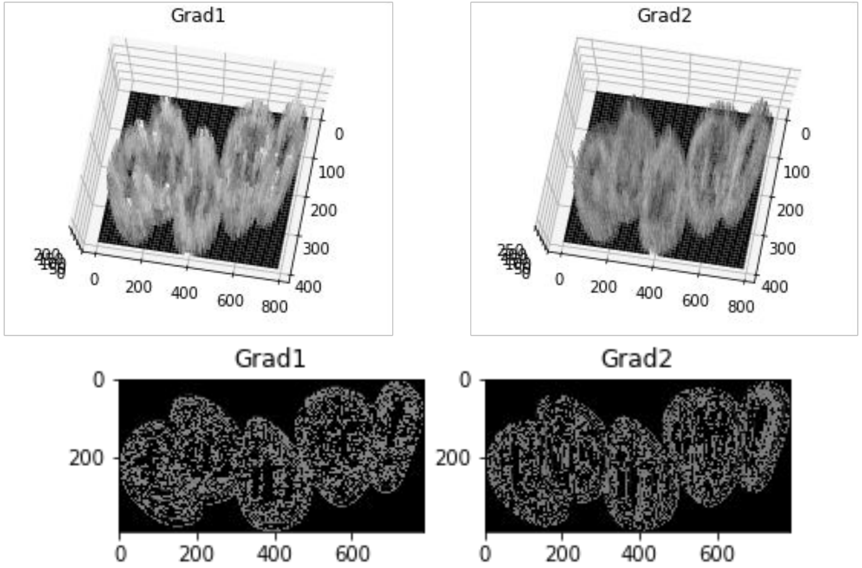


Image derivatives

We get terribly spiky results,
we need to interpolate / smooth.
⇒ Gaussian filter

We get a Sobel filter

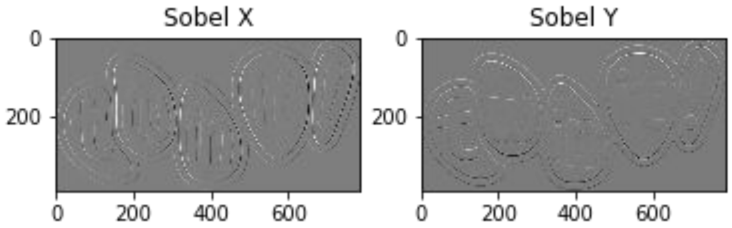


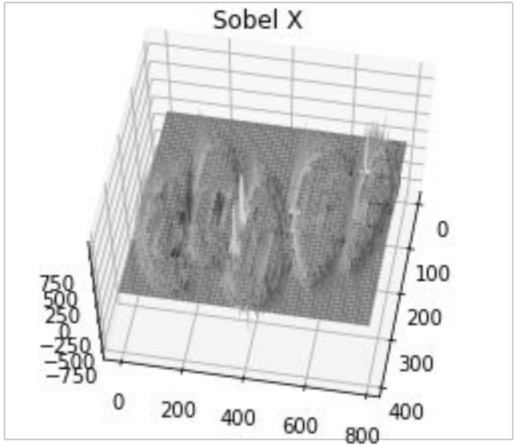
$$\frac{1}{2} \times \left(\begin{bmatrix} -1 & 0 & +1 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \right) = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

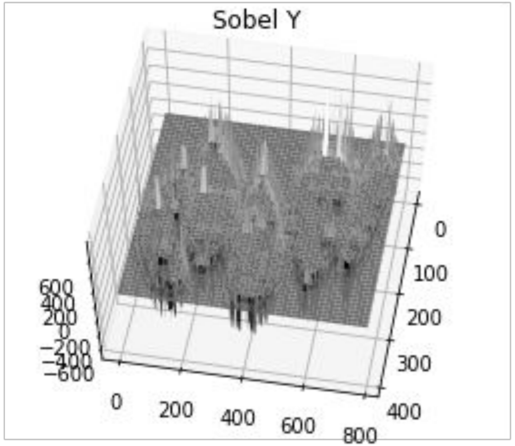
Horizontal Sobel

Vertical Sobel

Sobel filter

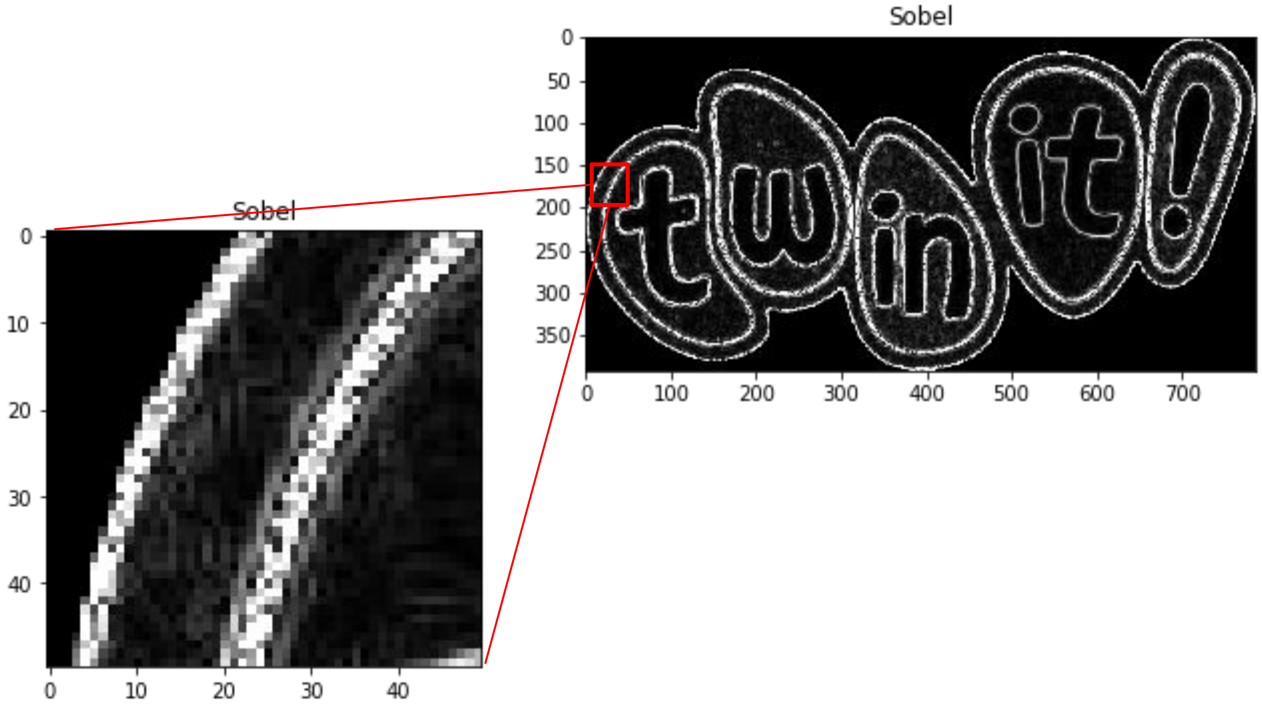






Gradient magnitude with Sobel

sqrt(Sobel_x² + Sobel_y²)



Canny edge detection

Extract real lines!

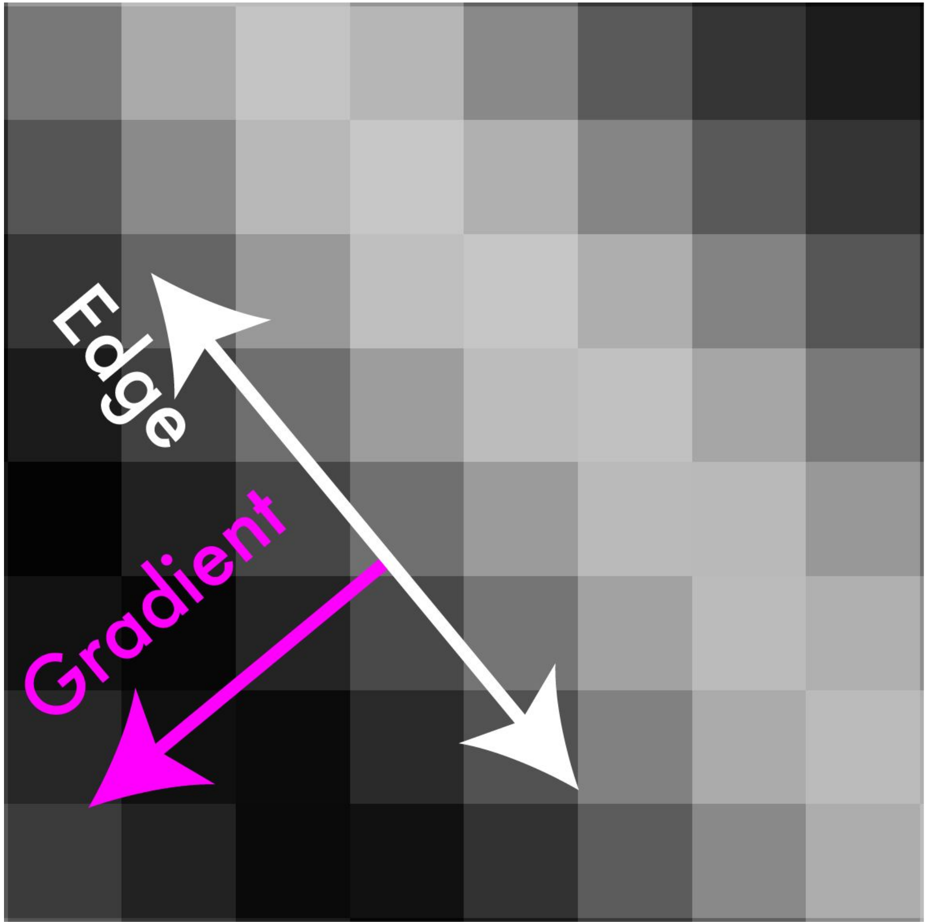
Algorithm:

Sobel operator

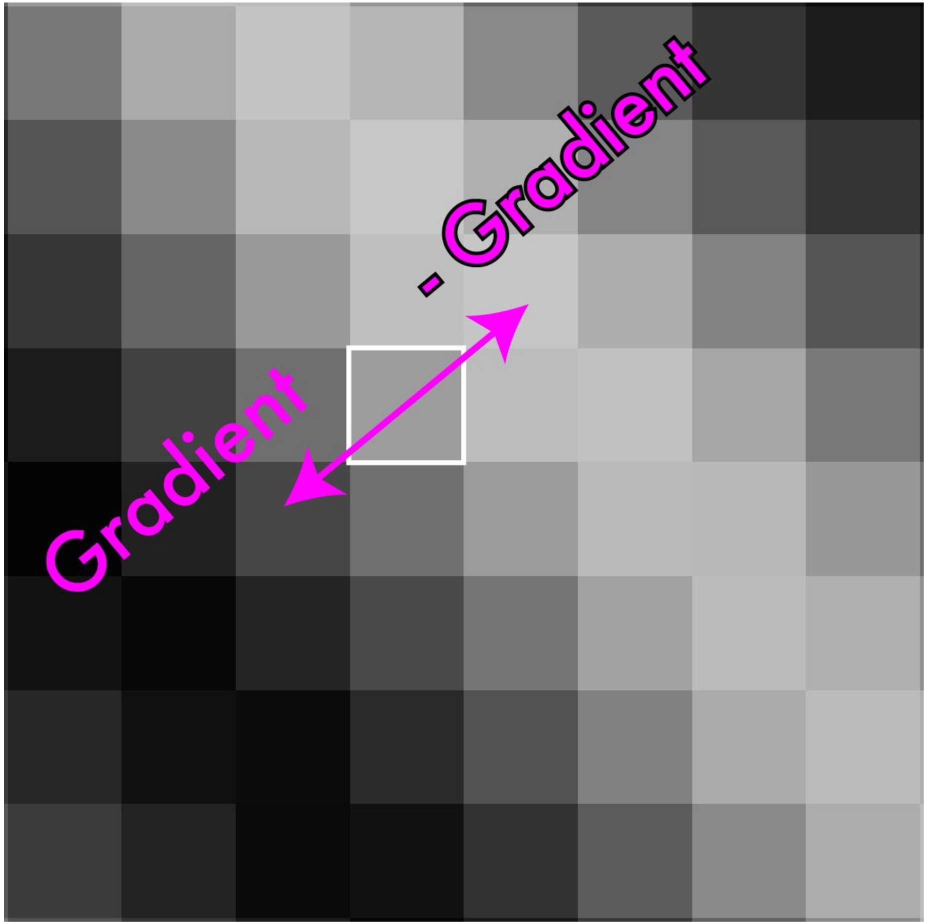
- Smooth image (only want “real” edges, not noise)
- Calculate gradient direction and magnitude
- Non-maximum suppression perpendicular to edge
- Threshold into strong, weak, no edge
- Keep only weak pixels connected to strong ones

John Canny. A computational approach to edge detection. Pattern Analysis and Machine Intelligence, IEEE Transactions on, (6):679–698, 1986.

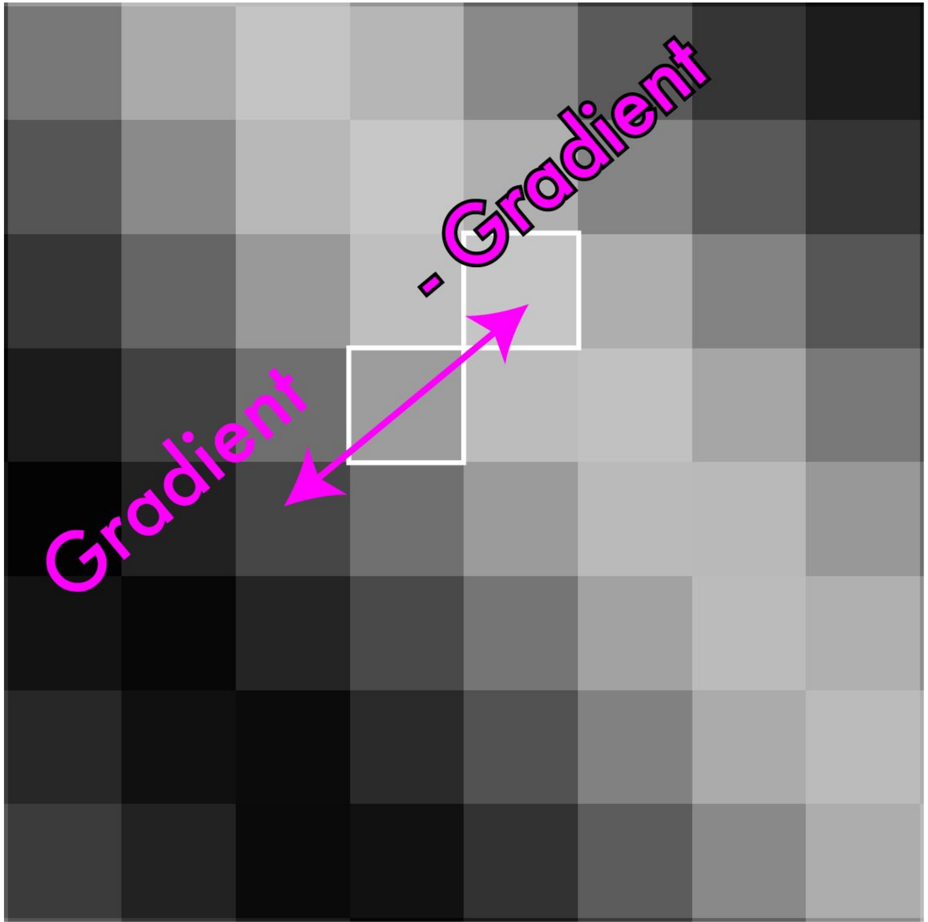
Canny: Non-maximum suppression



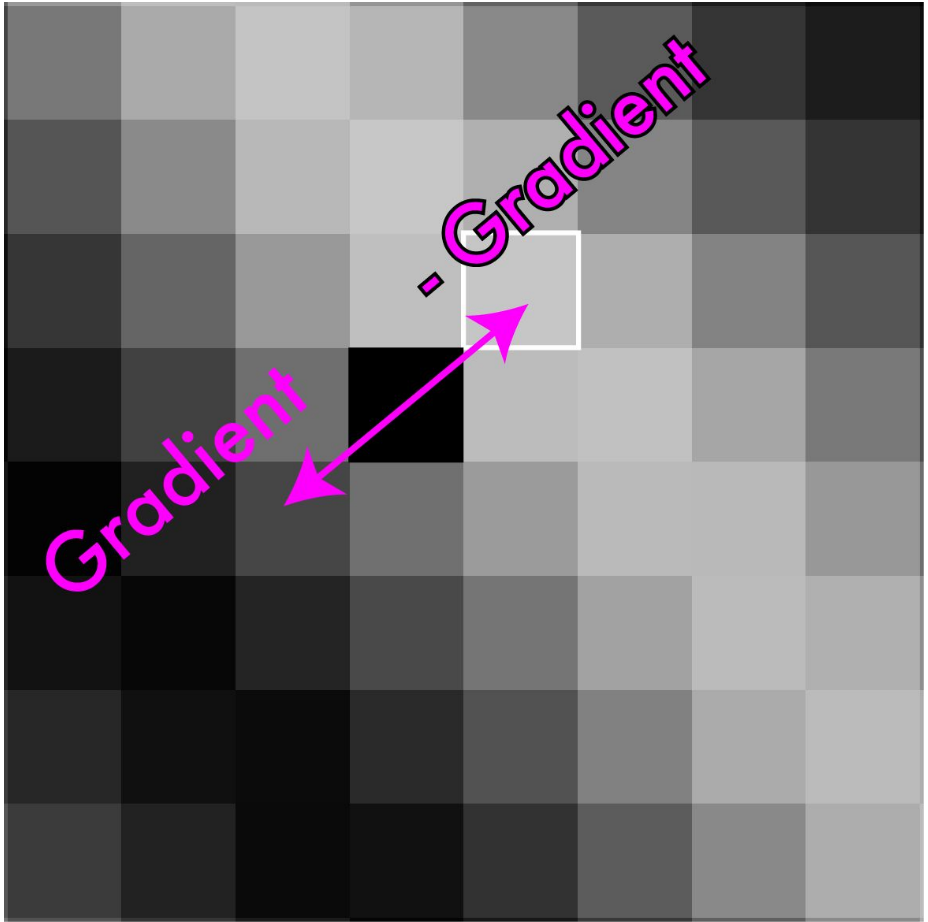
Canny: Non-maximum suppression



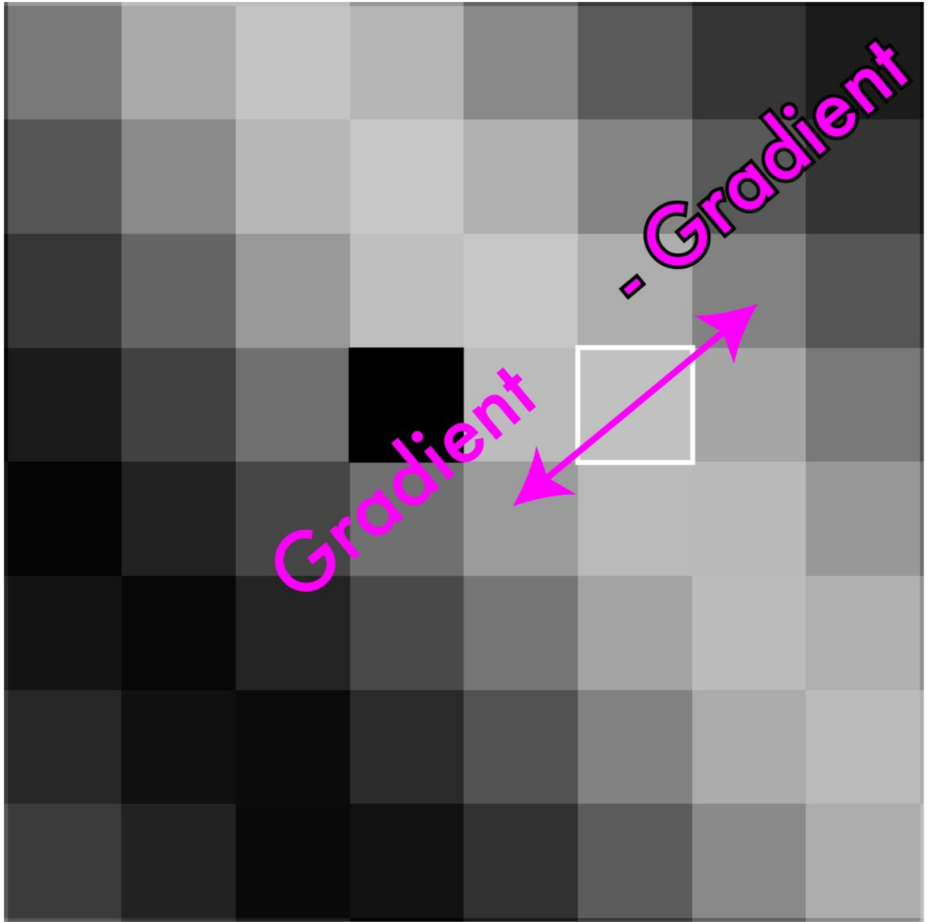
Canny: Non-maximum suppression



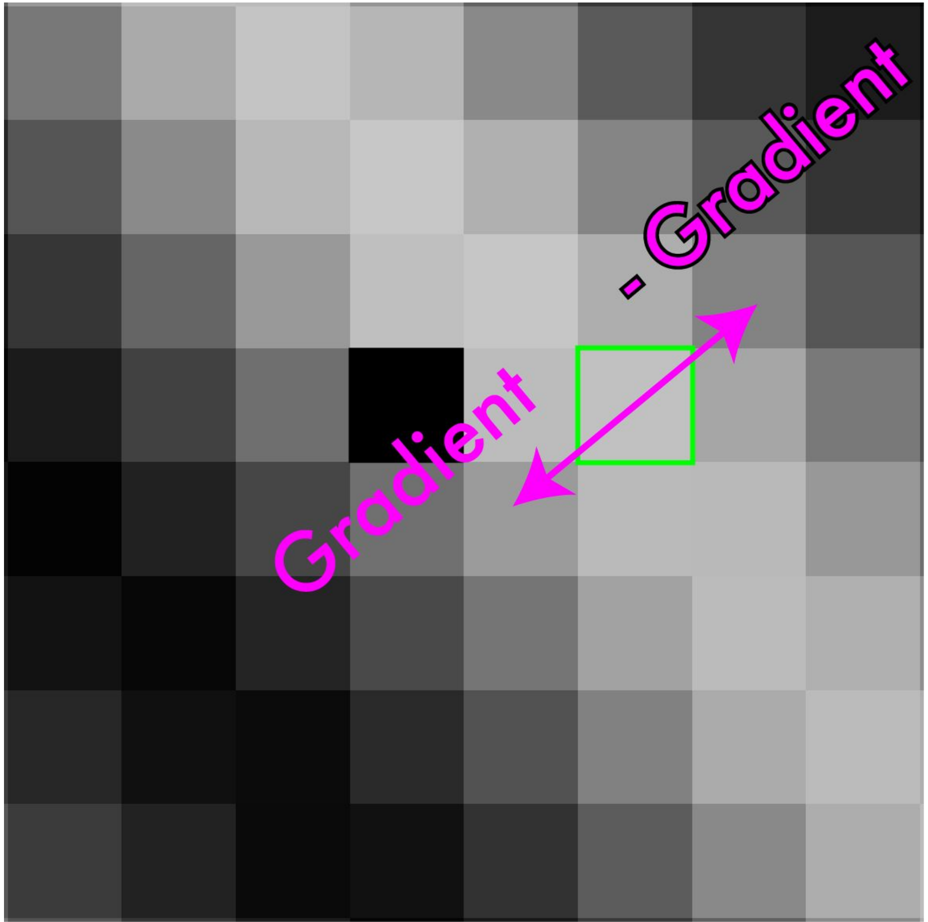
Canny: Non-maximum suppression



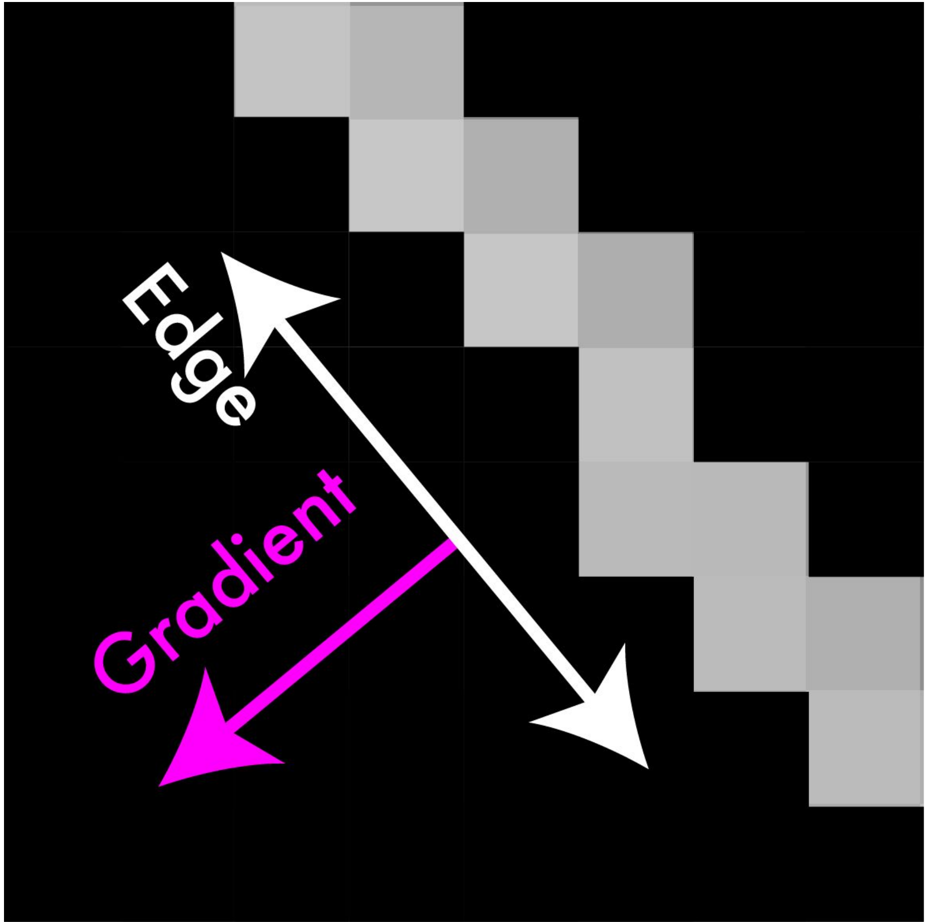
Canny: Non-maximum suppression



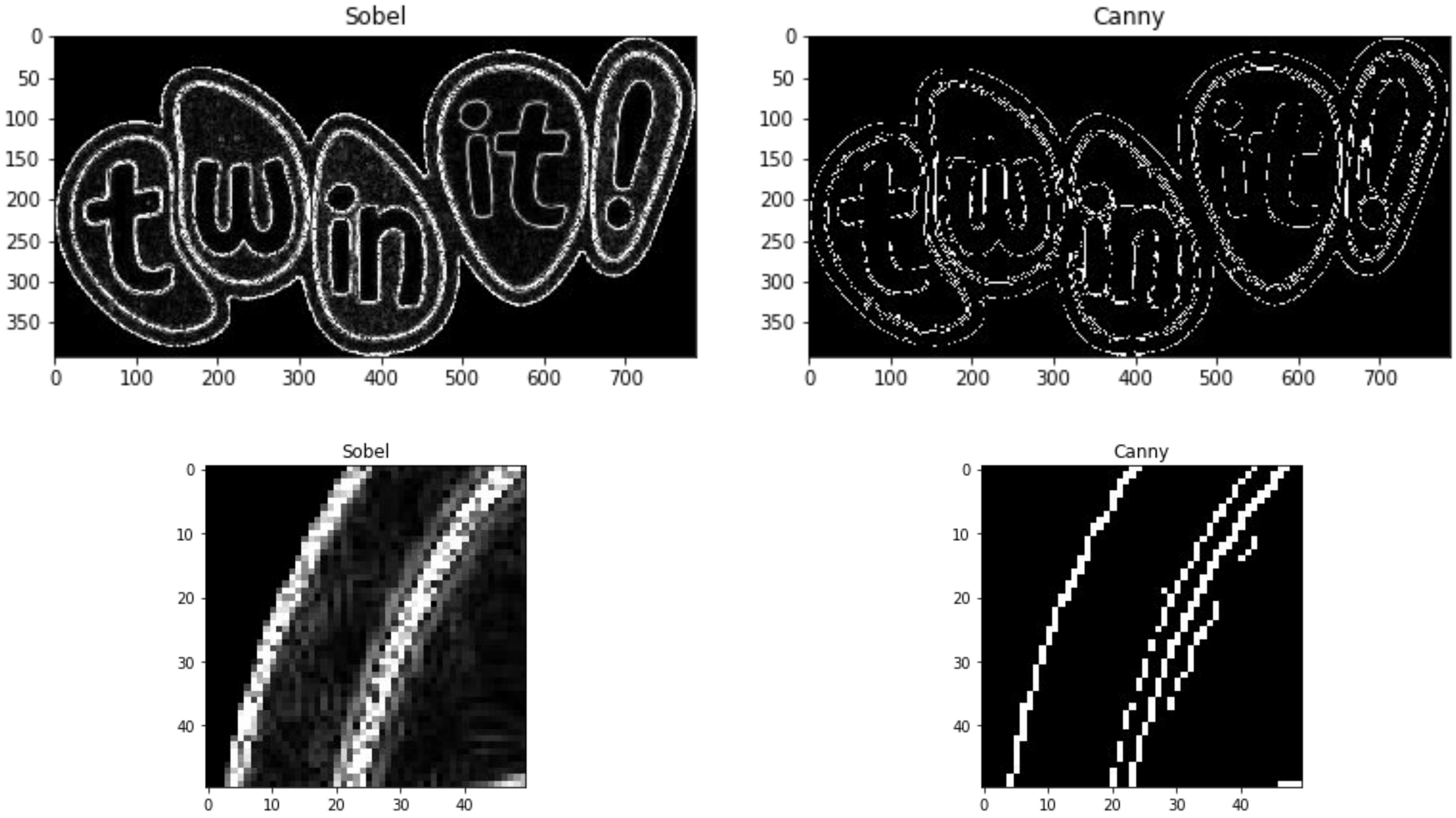
Canny: Non-maximum suppression



Canny: Non-maximum suppression



Canny: Non-maximum suppression



Canny: finalization

Threshold edges

- Still some noise
- Only want strong edges
- 2 thresholds, 3 cases
  - R > T: strong edge
  - R < T but R > t: weak edge
  - R < t: no edge
- Why two thresholds?

Connect weak edges to strong edges

- Strong edges are edges!
- Weak edges are edges
  iff they connect to strong
- Look in some neighborhood
  (usually 8 closest)

Corner detectors
Introduction, Harris detector

Good features

Reminder:

Good features are unique!

- Can find the “same” feature easily
- Not mistaken for “different” features

Good features are robust under perturbation

- Can detect them under translation, rotation…
- Intensity shift…
- Noise…

How close are two patches?

- Sum squared difference
- Images I, J
- $\Sigma_{x,y} (I(x,y) - J(x,y))^2$

How can we find unique patches?

Say we are stitching a panorama

Want patches in image to match to other image

Need to only match one spot

How can we find unique patches?

Sky? Bad!

- Very little variation
- Could match any other sky

How can we find unique patches?

Sky? Bad!

- Very little variation
- Could match any other sky

Edge? OK...

- Variation in one direction
- Could match other patches along same edge

How can we find unique patches?

Sky? Bad!

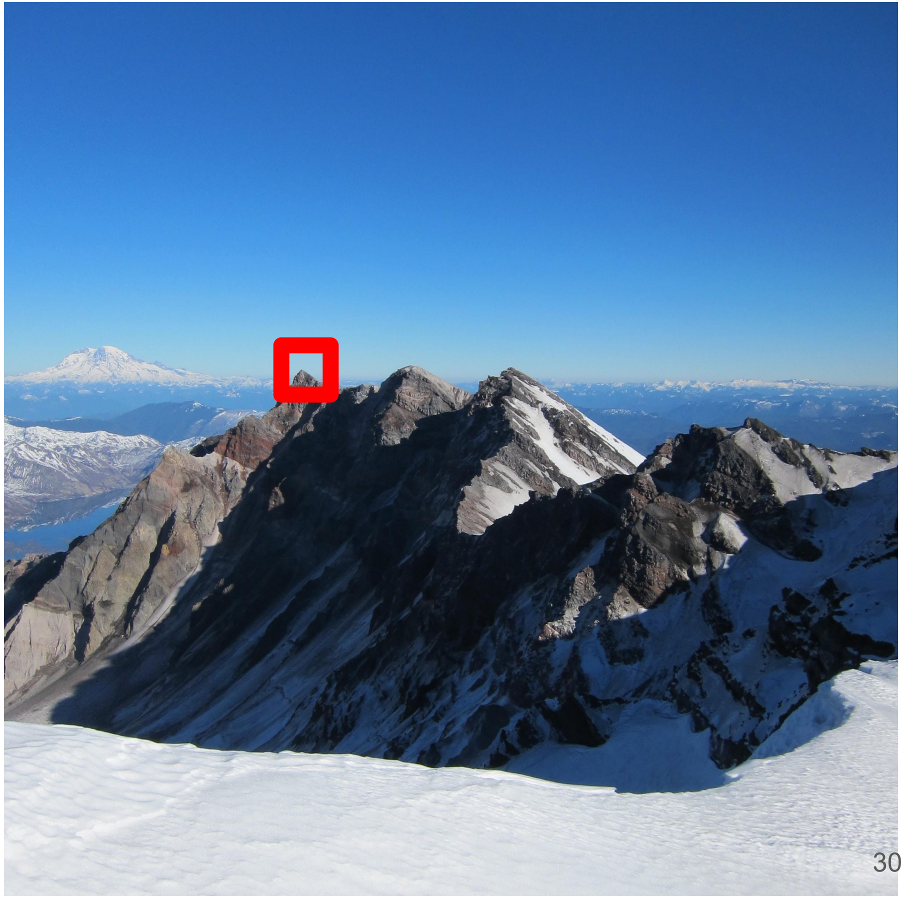
- Very little variation
- Could match any other sky

Edge? OK...

- Variation in one direction
- Could match other patches along same edge

Corners? good!

- Only one alignment matches



How can we find unique patches?

Want a patch that is unique in the image

Can calculate distance between patch
and every other patch, lot of computation

*

How can we find unique patches?

Want a patch that is unique in the image

Can calculate distance between patch and every other patch, lot of computation

Instead, we could think about auto-correlation:

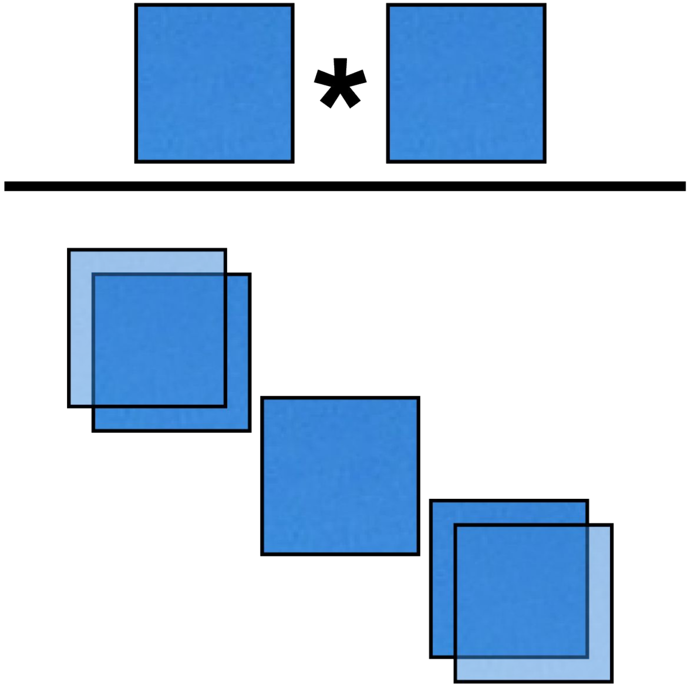
How well does image match shifted version of itself?

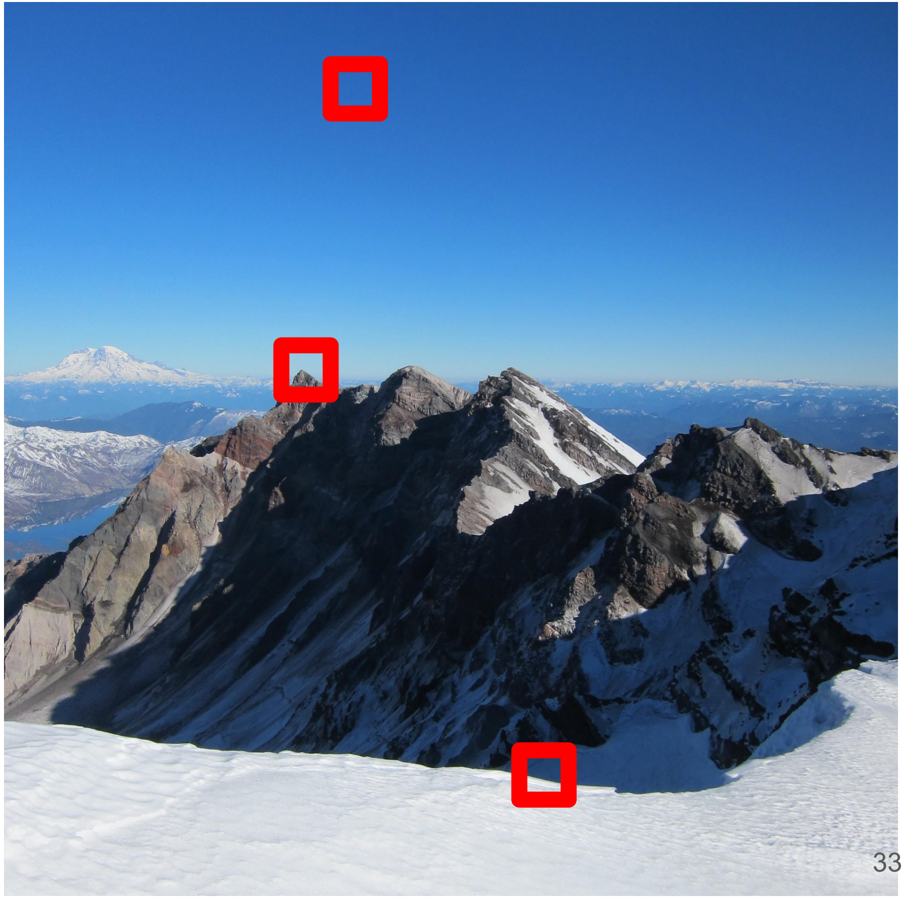
$$\Sigma_{\mathbf{d}}\Sigma_{x,y}\,(I(x,y) - I(x+\mathbf{d}_x,y+\mathbf{d}_y))^2$$

Measure of self-difference (how am I not myself?)

Self-difference

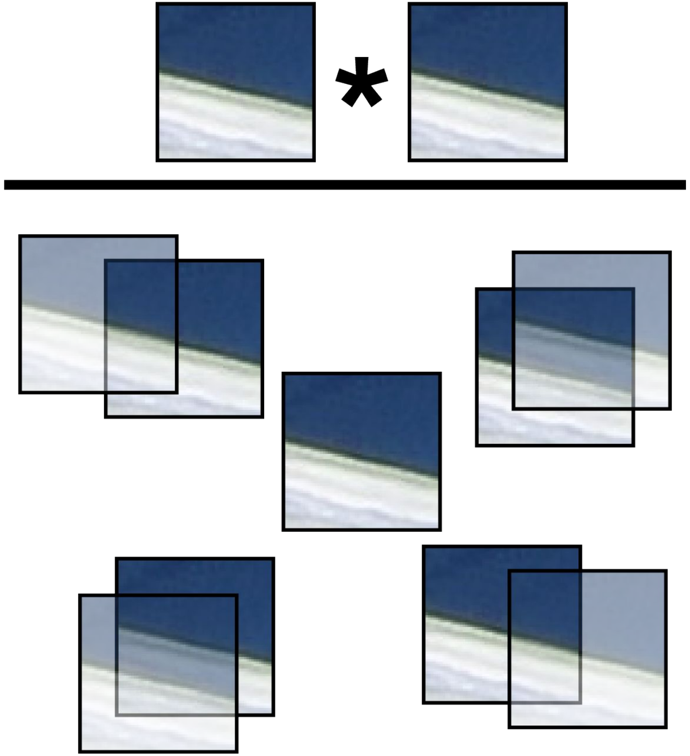
Sky: low everywhere

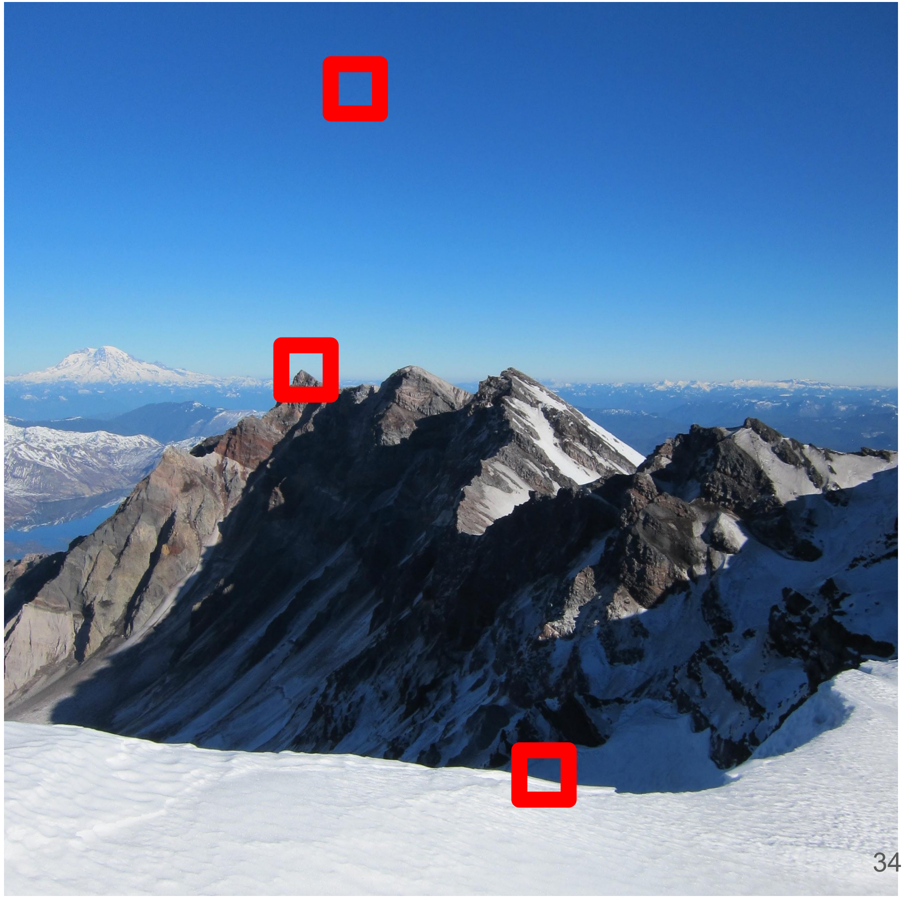




Self-difference

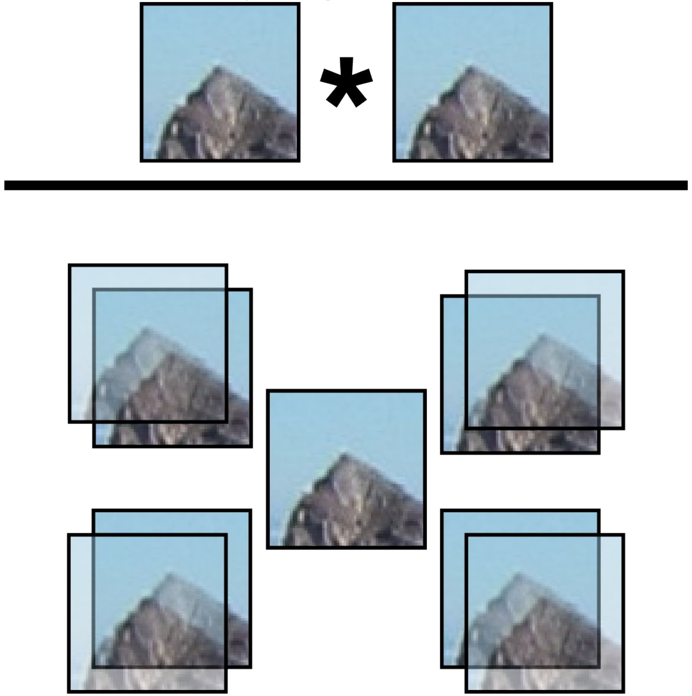
Edge: low along edge

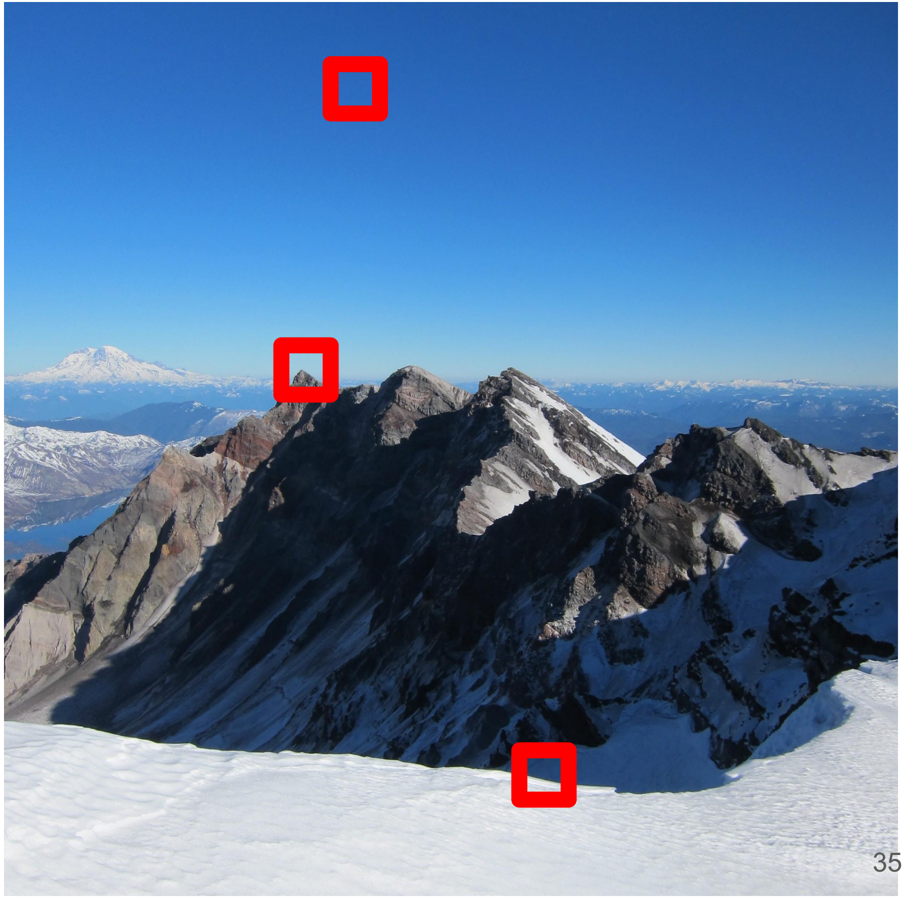




Self-difference

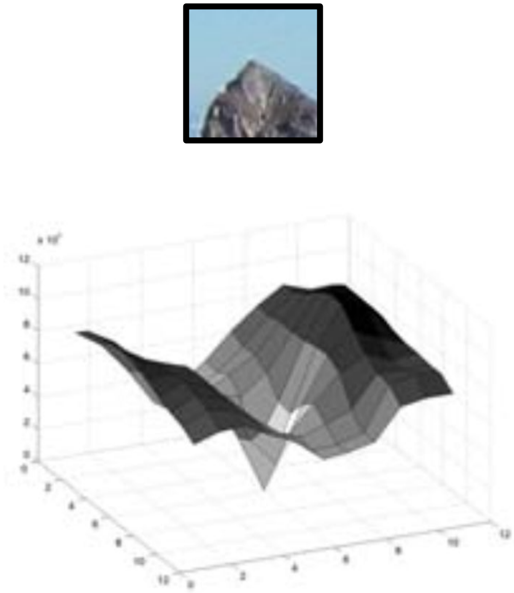
Corner: mostly high



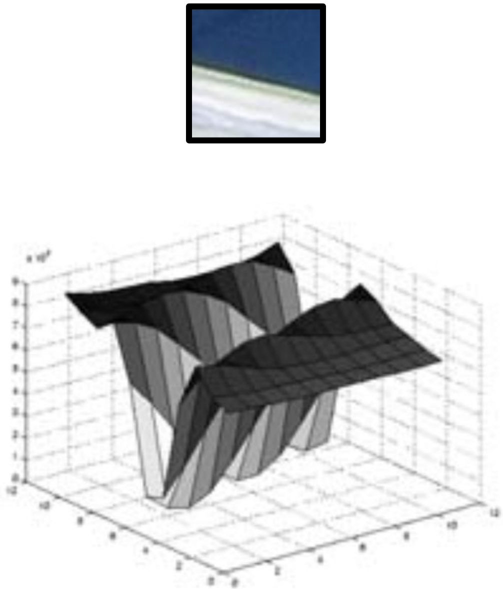


Self-difference

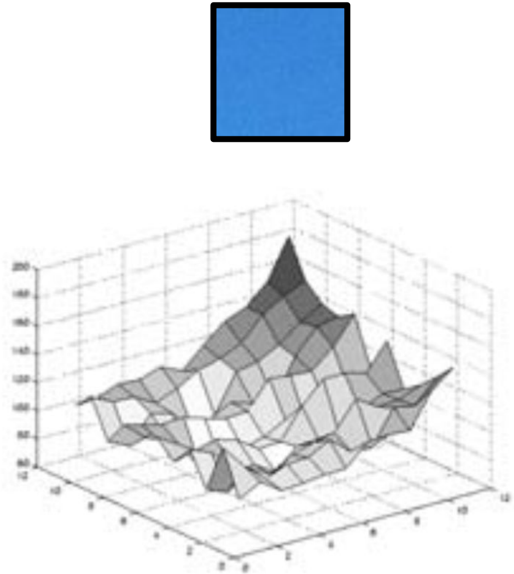
Corner: mostly high



Edge: low along edge



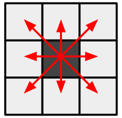
Sky: low everywhere



Self-difference

Naive computation:

$$\Sigma_{\mathbf{d}}\Sigma_{x,y}\ (I(x,y) - I(x+\mathbf{d}_x,y+\mathbf{d}_y))^2$$

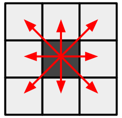


$(I(x,y) -$
$I(x+\mathbf{d}_x,y+\mathbf{d}_y))^2$

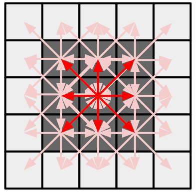
Harris corner detector

In practice we pool the previous indicator function over a small region (u,v) and we use a window $w(u,v)$ to weight the contribution of each displacement to the global sum.

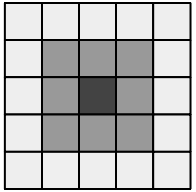
$$S(x,y) = \sum_u \sum_v w(u,v) \left(I(x+u+d_x, y+v+d_y) - I(x+u, y+v) \right)^2$$



$(I(x,y) - I(x+\mathbf{d}_x, y+\mathbf{d}_y))^2$



$$\sum_u \sum_v$$

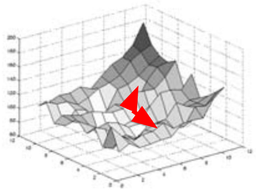


$$w(u,v)$$

Harris corner detector

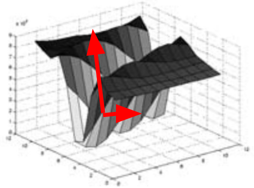
$$\Sigma_{\mathbf{d}}\Sigma_{x,y}\ (I(x,y) - I(x+\mathbf{d}_x,y+\mathbf{d}_y))^2$$

Lots of summing => Need an approximation

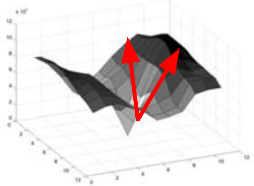


Look at nearby gradients Ix and Iy

- If gradients are **mostly zero**, not a lot going on
  ⇒ Low self-difference



- If gradients are **mostly in one direction**, edge
  ⇒ Still low self-difference
- If gradients are **in twoish directions**, corner!
  ⇒ High self-difference, good patch!



Harris corner detector

Trick to precompute the derivatives

$$I(x + d_x, y + d_y)$$

can be approximated by a Taylor expansion

$$I(x + d_x, y + d_y) \approx I(x, y) + d_x \frac{\partial I(x, y)}{\partial x} + d_y \frac{\partial I(x, y)}{\partial y} + \cdots$$

Harris corner detector

This allows us to "simplify" the original equation,

$$S(x, y) \approx \sum_u \sum_v w(u, v)\left(d_x \frac{\partial I(x+u, y+v)}{\partial x} + d_y \frac{\partial I(x+u, y+v)}{\partial y} \right)^2$$

and more important making it **faster to compute**,
thanks to simpler derivatives which can be **computed for the whole image**.

Harris corner detector

If we develop the equation and write is as usual matrix form, we get:

$$S(x,y) \approx \begin{pmatrix} d_x & d_y \end{pmatrix} A(x,y) \begin{pmatrix} d_x \\ d_y \end{pmatrix}$$

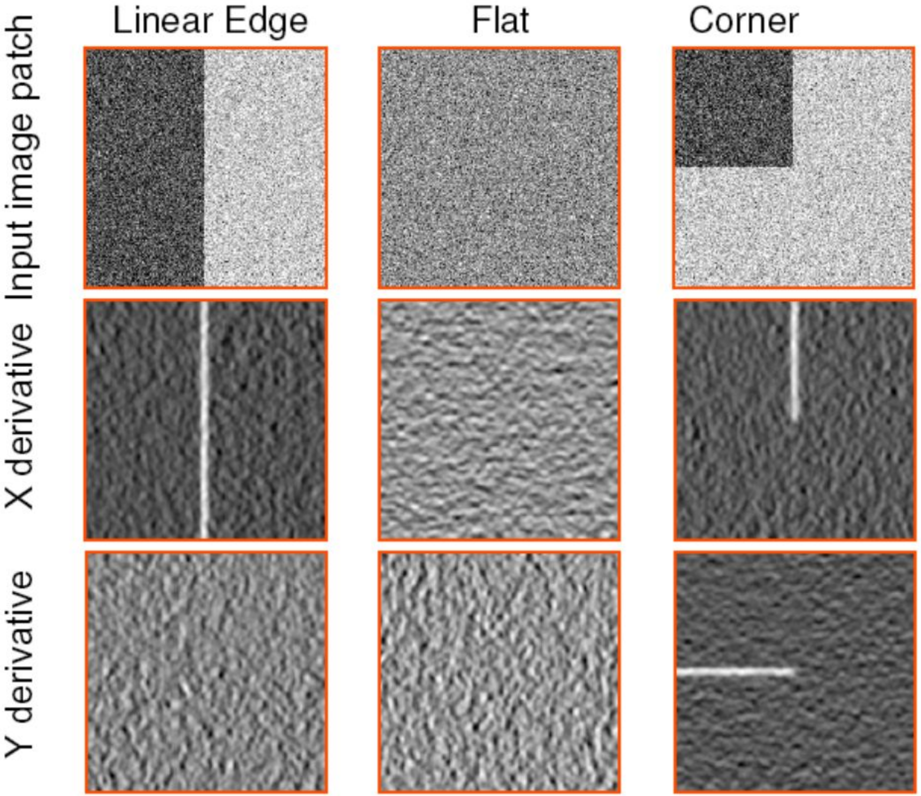
where $A(x,y)$ is the structure tensor:

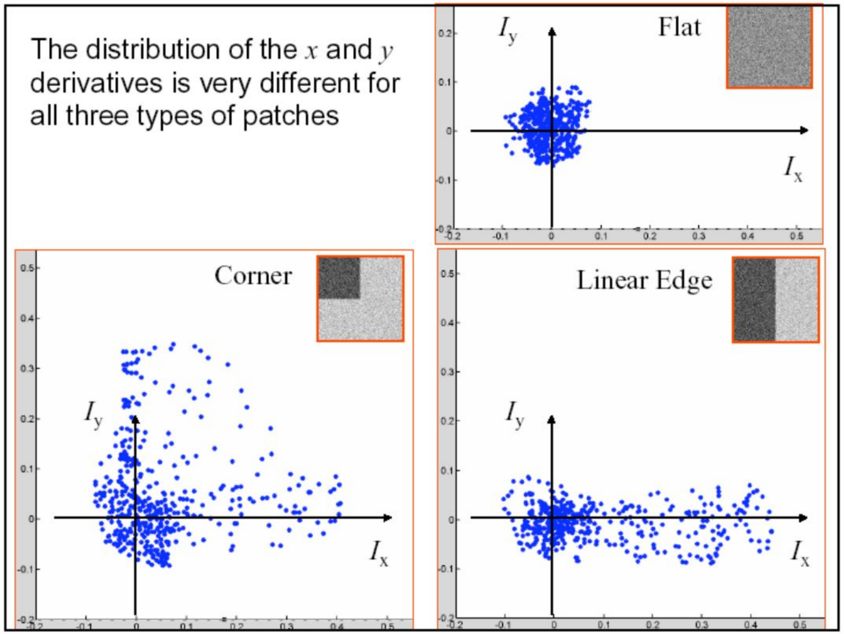
$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} \frac{\partial I^2(x+u,y+v)}{\partial x} & \frac{\partial I(x+u,y+v)}{\partial x} \frac{\partial I(x+u,y+v)}{\partial y} \\ \frac{\partial I(x+u,y+v)}{\partial x} \frac{\partial I(x+u,y+v)}{\partial y} & \frac{\partial I^2(x+u,y+v)}{\partial y} \end{bmatrix}$$

$$= \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$$

This trick is useful because Ix and Iy can be precomputed very simply.

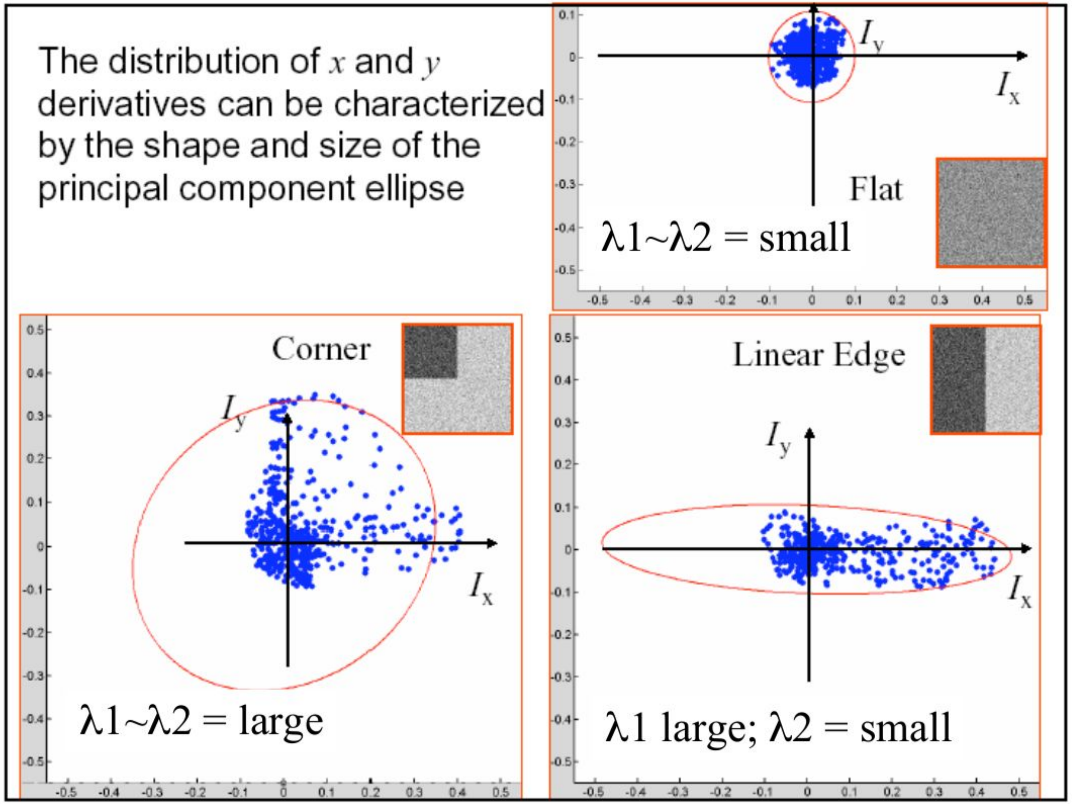
Harris corner detector





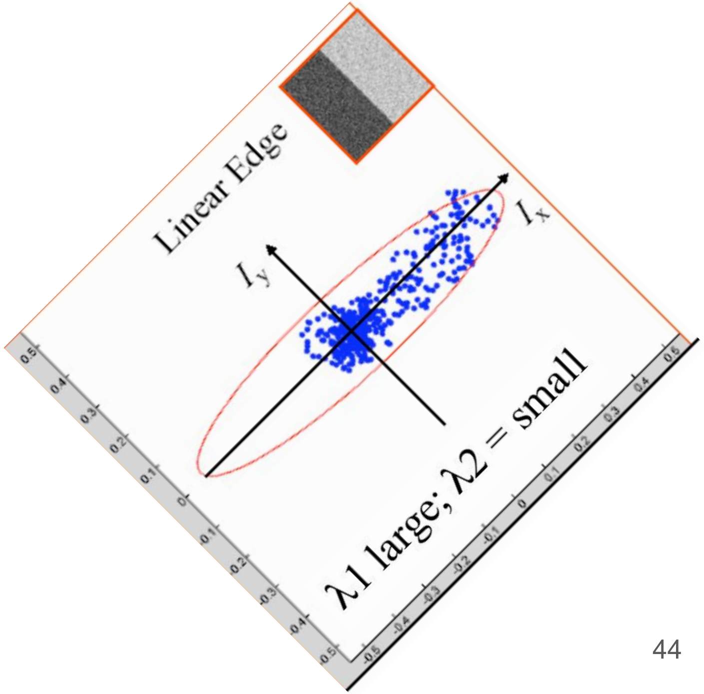
Illustrations: Robert Collins

Harris corner detector



<u>The need for eigenvalues:</u>
If the edge is rotated,
so are the values of I_x and I_y.

Eigenvalues give us the ellipsis axis len.



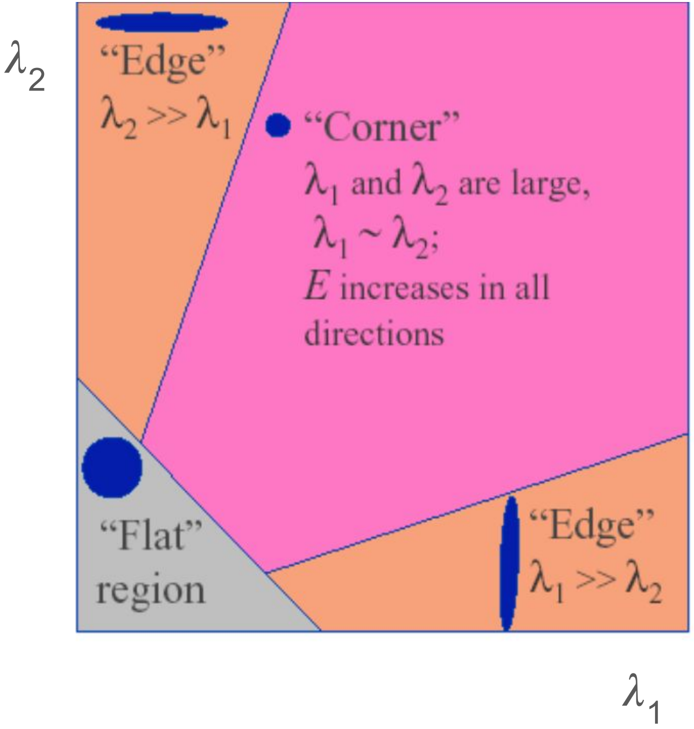
Illustrations: Robert Collins

Harris corner detector

A corner is characterized by a large variation of S in all directions of the vector $(x\ y)$.

Analyse the eigenvalues of A to check whether we have two large variations.

- If $\lambda_1 \approx 0$ and $\lambda_2 \approx 0$ then this pixel (x,y) has no features of interest.
- If $\lambda_1 \approx 0$ and λ_2 has some large positive value, then an edge is found.
- If λ_1 and λ_2 have large positive values, then a corner is found.



Illustrations: Robert Collins

Harris corner detector

To avoid the computation of the eigenvalues, which used to be expensive, Harris and Stephens instead suggest the following function Mc , where κ is a tunable sensitivity parameter:

$$M_c = \lambda_1 \lambda_2 - \kappa (\lambda_1 + \lambda_2)^2 = \underbrace{\det(A) - \kappa \operatorname{trace}^2(A)}_{\text{approximation}}$$

We will use Noble's trick to remove κ:

$$M_c' = 2 \frac{\det(A)}{\operatorname{trace}(A) + \epsilon}$$

ϵ being a small positive constant.

Harris corner detector

A being a 2x2 matrix, we have the following relations:

- det(A)=$A_{1,1}A_{2,2}-A_{2,1}A_{1,2}$
- trace(A)=$A_{1,1}+A_{2,2}$

Using previous definitions, we obtain:

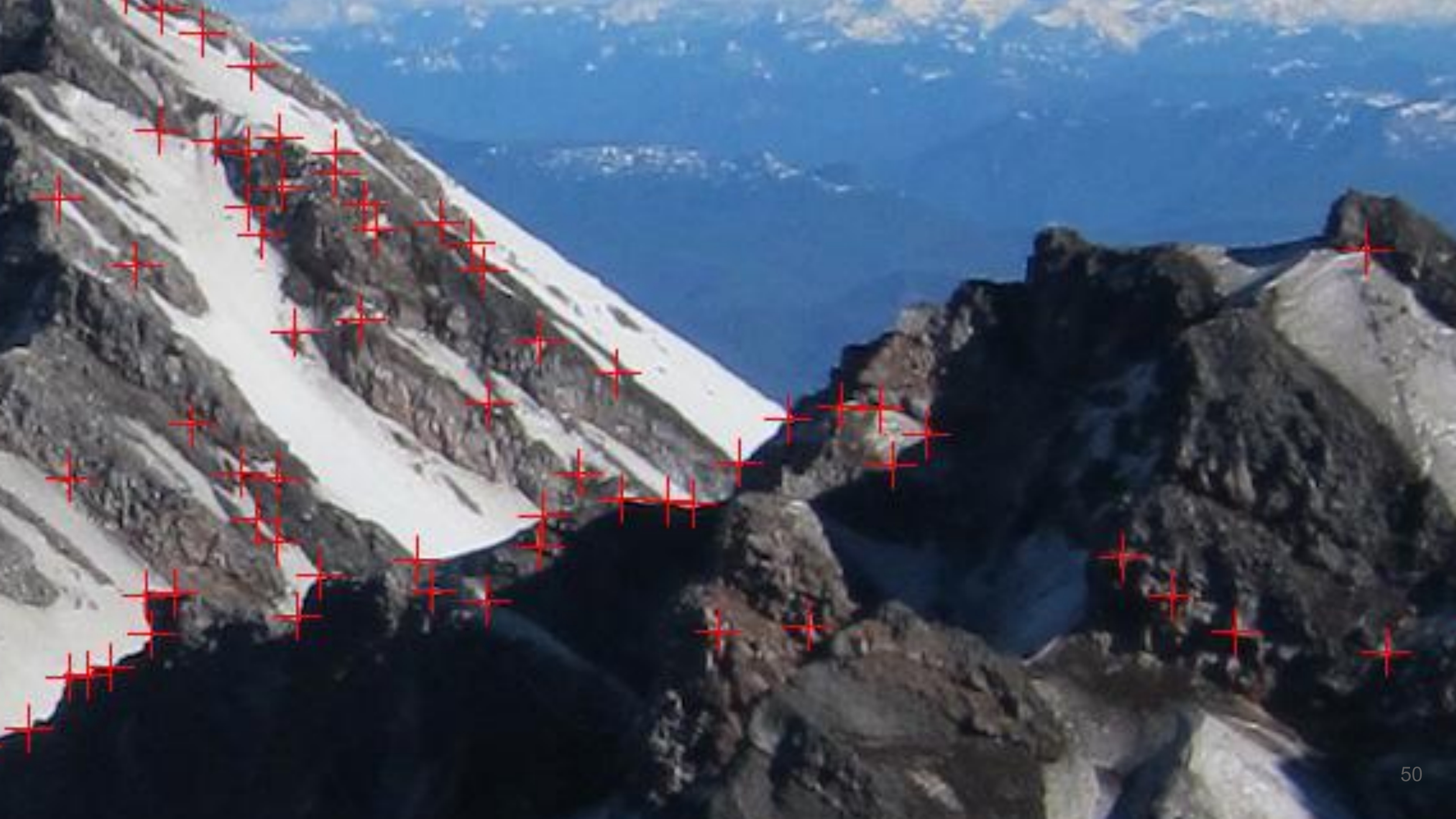
- det(A)=$\langle I^2x\rangle\langle I^2y\rangle-\langle IxIy\rangle^2$
- trace(A)=$\langle I^2x\rangle+\langle I^2y\rangle$

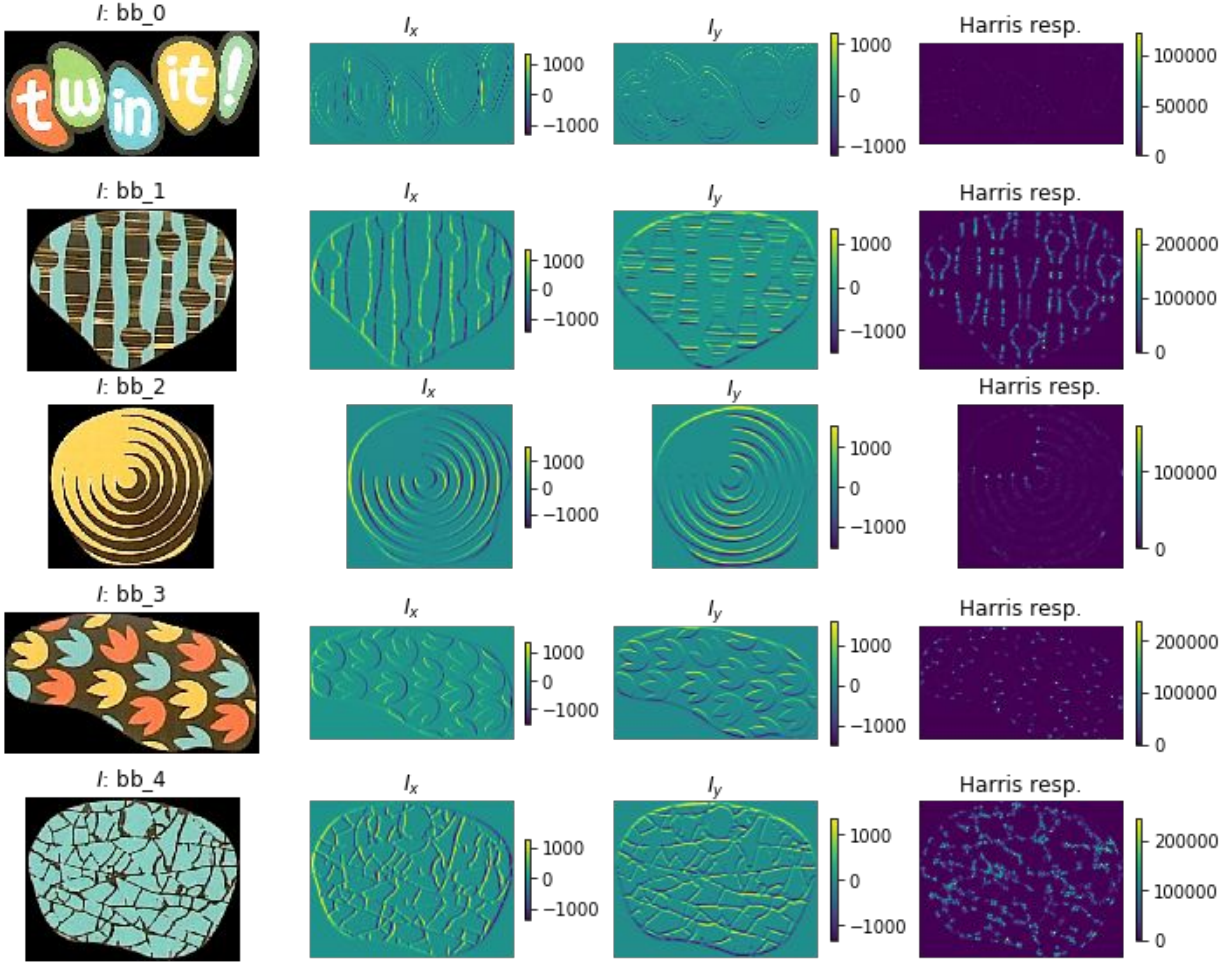
Harris corner detector

In summary, given an image, we can compute the Harris corner response image by simply computing:

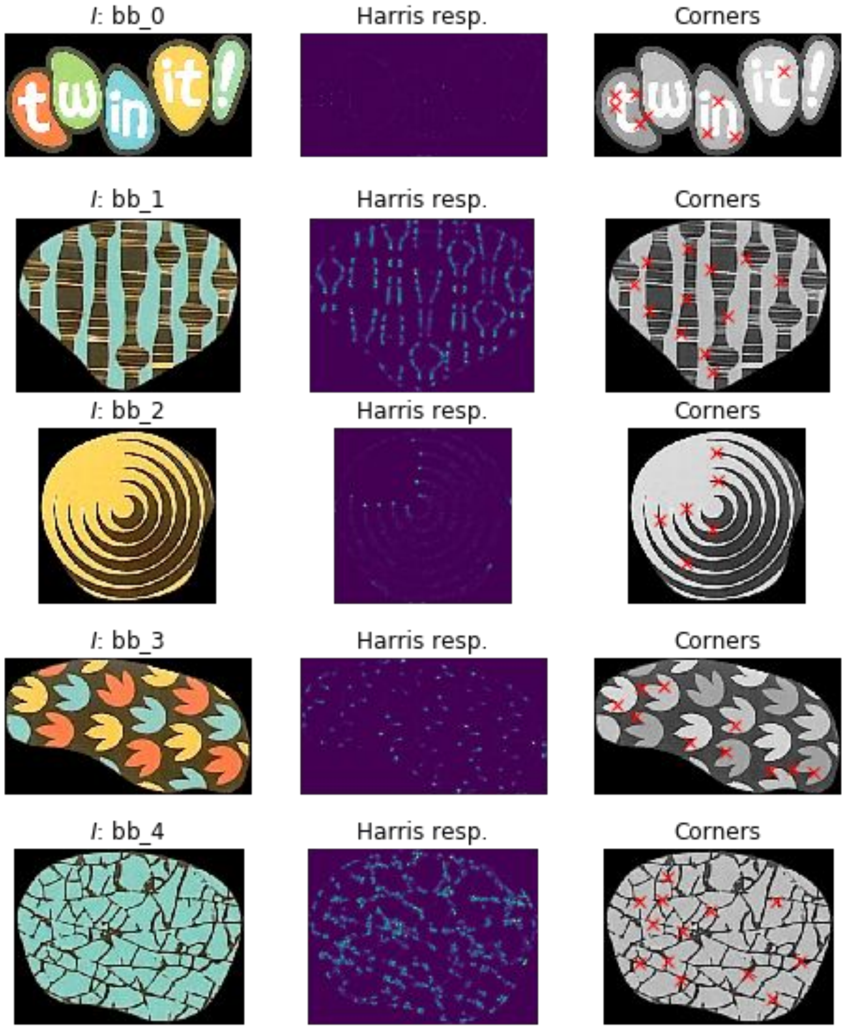
- Ix : I 's smoothed (interpolated) partial derivative with respect to x ;
- Iy : I 's smoothed (interpolated) partial derivative with respect to y ;
- $\langle I^2x \rangle$: the windowed sum of I^2x ;
- $\langle I^2y \rangle$: the windowed sum of I^2y ;
- $\langle IxIy \rangle$: the windowed sum of $IxIy$;
- det(A) ;
- trace(A) ;
- M''_c=det(A) / (trace(A)+ϵ).

Then, we just perform **non-maximal suppression** to keep local maximas.





I: bb_0
I_x
I_y
Harris resp.
I: bb_1
I: bb_2
I: bb_3
I: bb_4
1000
0
−1000
100000
50000
200000


I: bb_0
Harris resp.
Corners
I: bb_1
Harris resp.
Corners
I: bb_2
Harris resp.
Corners
I: bb_3
Harris resp.
Corners
I: bb_4
Harris resp.
Corners

Harris & Stephens Conclusion

Good features to track *aka* Shi-Tomasi *aka* Kanade-Tomasi

Remember the Harris-Stephens trick to avoid computing the eigenvalues?

$$M_c = \lambda_1 \lambda_2 - \kappa (\lambda_1 + \lambda_2)^2 = \underbrace{\det(A) - \kappa \operatorname{trace}^2(A)}_{\text{approximation}}$$

Well, nowadays, linear algebra is cheap, so **compute the real eigenvalues**.

Then filter using $min(\lambda_1, \lambda_2) > \lambda$, λ being a predefined threshold.

You get the Shi-Tomasi variant.

Jianbo Shi and Carlo Tomasi. Good features to track. In Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on, pages 593–600. IEEE, 1994.

Build your own edge/corner detector

Hessian matrix with block-wise summing

You just need eigenvalues λ_1 and λ_2 of the structure tensor

$$A = \sum_u \sum_v w(u,v) \begin{bmatrix} \frac{\partial I^2(x+u,y+v)}{\partial x} & \frac{\partial I(x+u,y+v)}{\partial x}\frac{\partial I(x+u,y+v)}{\partial y} \\ \frac{\partial I(x+u,y+v)}{\partial x}\frac{\partial I(x+u,y+v)}{\partial y} & \frac{\partial I^2(x+u,y+v)}{\partial y} \end{bmatrix}$$

$$= \begin{bmatrix} \langle I_x^2 \rangle & \langle I_x I_y \rangle \\ \langle I_x I_y \rangle & \langle I_y^2 \rangle \end{bmatrix}$$

```
dst = cv2.cornerEigenValsAndVecs(src, neighborhood_size, sobel_aperture)
dst = cv2.cornerMinEigenVal(src, neighborhood_size, sobel_aperture)
```

Harris summary

Pros

Translation invariant
⇒ Large gradients in both directions
= stable point

Cons

Not so fast
⇒ Avoid to compute all those derivatives

Not scale invariant
⇒ Detect corners at different *scales*

Not rotation invariant
⇒ Normalization orientation

Corner detectors, binary tests
FAST

Features from accelerated segment test (FAST)

Keypoint detector used by ORB (described in next lecture)

Segment test:
compare pixel ***P*** intensity I_p
with surrounding pixels
(circle of 16 pixels)

If ***n*** contiguous pixels are either
- all darker than $I_p - t$
- all brighter than $I_p + t$

then ***P*** is a detected as a corner

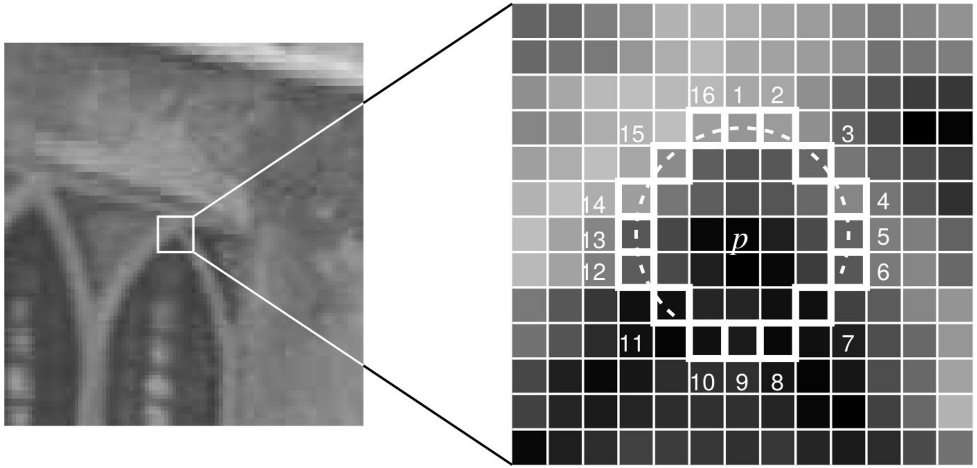


Figure 1. 12 point segment test corner detection in an image patch. The highlighted squares are the pixels used in the corner detection. The pixel at p is the centre of a candidate corner. The arc is indicated by the dashed line passes through 12 contiguous pixels which are brighter than p by more than the threshold.

E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in European conference on computer vision, 2006, pp. 430–443.

Tricks

1. **Cascading:** If n = 12 (¾ of the circle), then many non-corners can be discarded by testing pixels at the 4 compass directions. The full test is only applied to the candidates which passed the first test.
2. **Machine learning:** Learn on a dataset which pixels should be tested first to discard a non-corner as quickly as possible.
   Learn a decision tree, then compile the decisions as nested if-then rules.
3. How to perform **non-maximal suppression**?
   Need to assign a score ***V*** to each corner.
   ⇒ The sum of the absolute difference between the pixels in the contiguous arc and the centre pixel

$$V = \max\left(\sum_{x \in S_{\text{bright}}} |I_{p \to x} - I_p| - t \;,\; \sum_{x \in S_{\text{dark}}} |I_p - I_{p \to x}| - t\right)$$

FAST summary

Pros

Very fast

Authors tests:

- 20 times faster than Harris
- 40 times faster than DoG *(next slide)*

Very robust to transformations (perspective in particular)

Cons

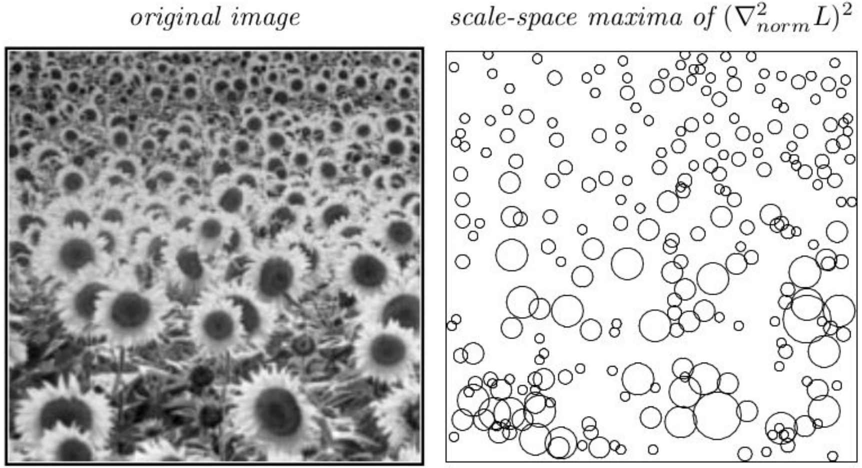
Very sensitive to blur

Corner detectors at different scales
LoG, DoG, DoH

Laplacian of Gaussian (LoG)

The theoretical, slow way.

If you need to remember only 1 thing:
it is a **band-pass filter** – it **detects objects of a certain size**.



T. Lindeberg, “Feature Detection with Automatic Scale Selection,” Int. J. of Computer Vision, vol. 30, no. 2, p. 53, 1998.

Laplacian (plain, not Gaussian here) = second derivative

Second derivative of an image? Like Sobel… with 1 more derivation…

Taylor, again:

$$f(x+h) = f(x) + hf'(x) + \frac{1}{2}h^2 f''(x) + \frac{1}{3!}h^3 f'''(x) + O(h^4)$$

add

$$+ \left[f(x-h) = f(x) - hf'(x) + \frac{1}{2}h^2 f''(x) - \frac{1}{3!}h^3 f'''(x) + O(h^4) \right]$$

$$f(x+h) + f(x-h) = 2f(x) + h^2 f''(x) + O(h^4)$$

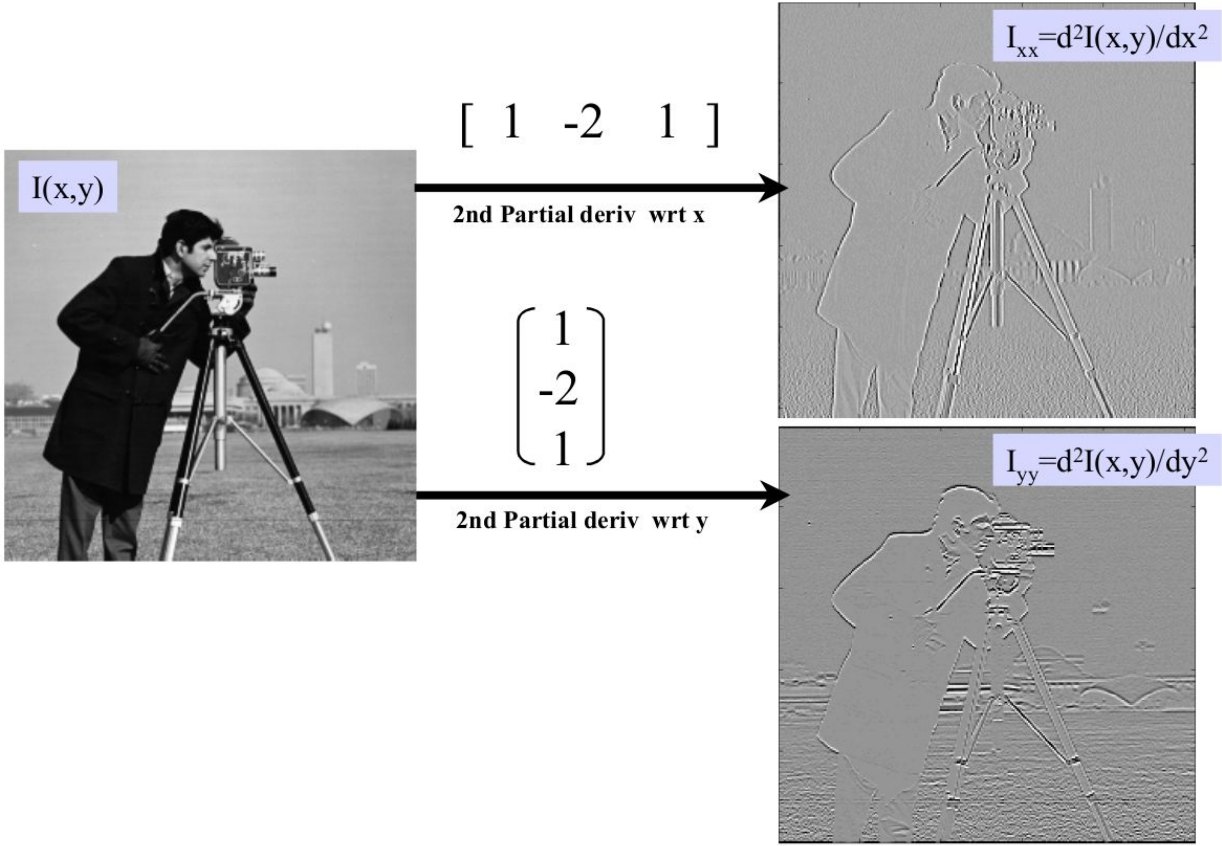
$$\frac{f(x-h) - 2f(x) + f(x+h)}{h^2} = f''(x) + O(h^2)$$

New filter: I_{xx} =

1	-2	1

$* I$

Second partial derivatives of an image



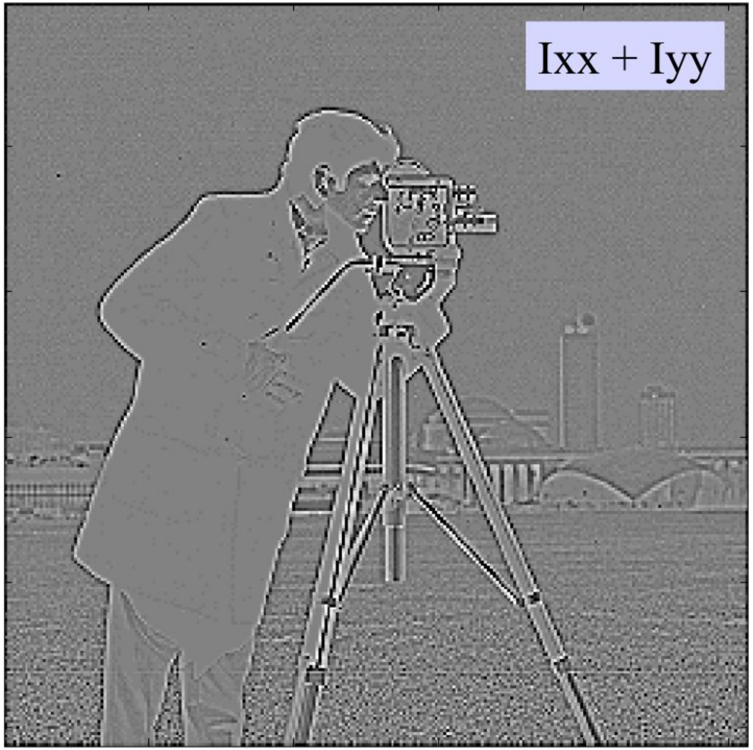
Laplacian filter ∇^2 I(x,y)

Edge detector, like Sobel but with 2nd derivatives

$$I_{xx}+I_{yy}=\left(\begin{bmatrix}1 & -2 & 1\end{bmatrix}+\begin{bmatrix}1\\-2\\1\end{bmatrix}\right)*I$$

$$=\begin{bmatrix}0 & 1 & 0\\1 & -4 & 1\\0 & 1 & 0\end{bmatrix}*I$$

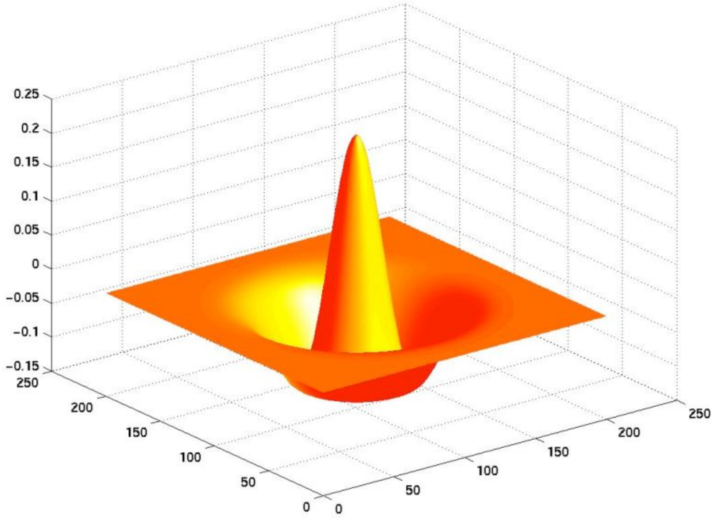
$$\begin{bmatrix}0 & 1 & 0\\1 & -4 & 1\\0 & 1 & 0\end{bmatrix}*I$$



Laplacian of Gaussian

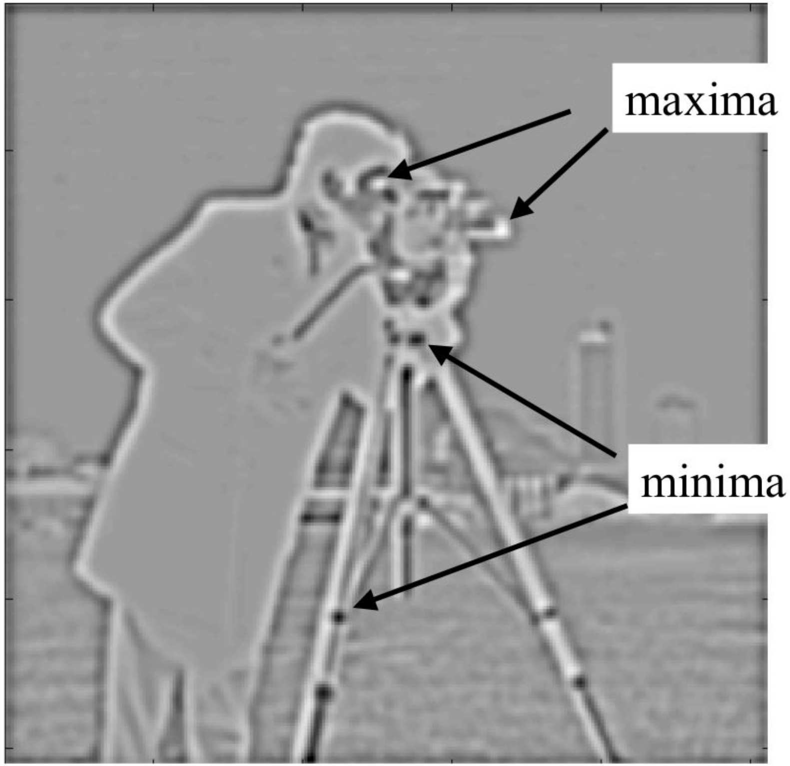
Second derivative of a Gaussian: **“Mexican hat”**

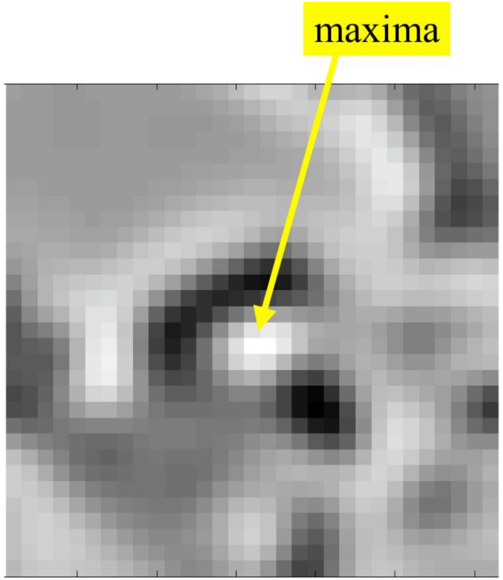
$$g''(x) = \left(\frac{x^2}{\sigma^4} - \frac{1}{\sigma^2}\right) e^{-\frac{x^2}{2\sigma^2}}$$

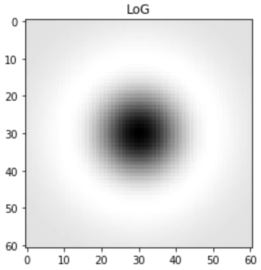


2D formula = exercise.

LoG = detector of circular shapes





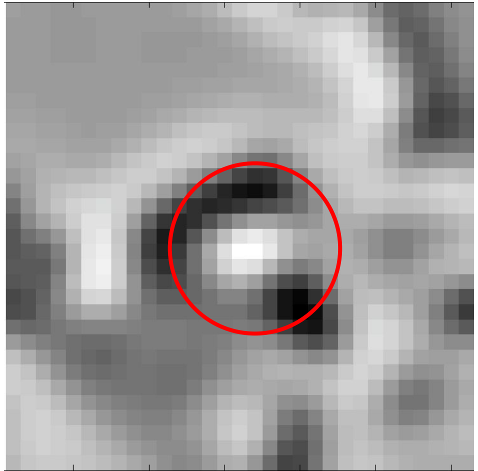


LoG = detector of circular shapes

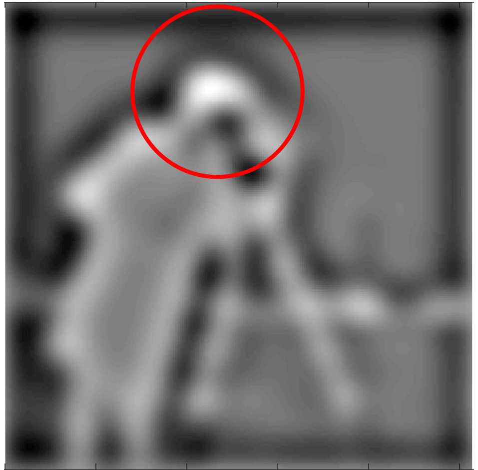
LoG filter extrema locates “blobs”

- maxima = dark blobs on light background
- minima = light blobs on dark background

Scale of blob (size ; radius in pixels) is determined by the **sigma** parameter of the LoG filter.



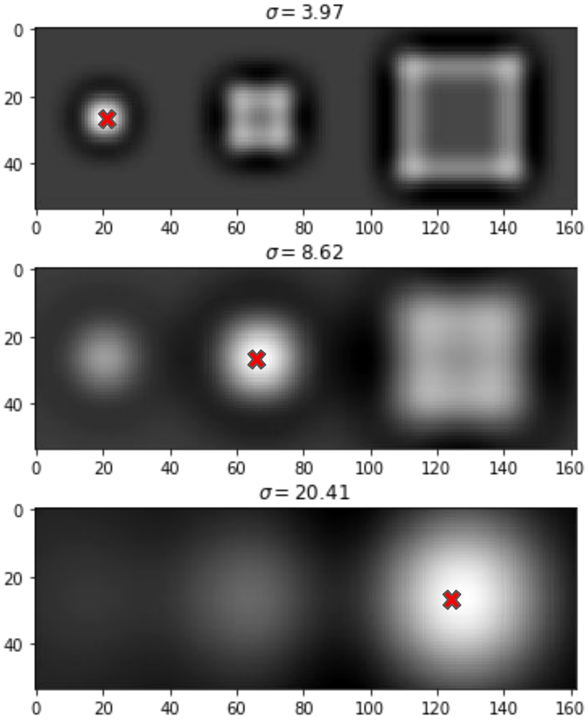
LoG σ=2

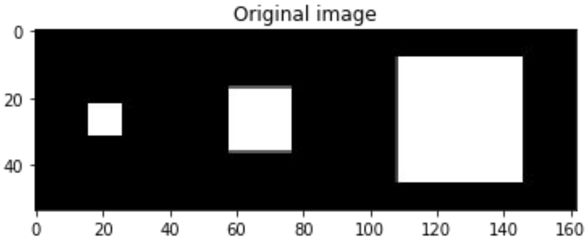


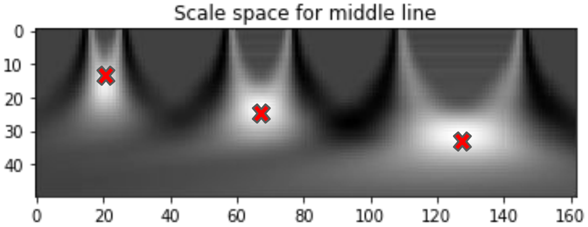
LoG σ=10

Detecting corners / blobs

Build a scale space representation: *stack of images (3D) with increasing sigma*







Then find local extremas in the scale space volume.

Difference of Gaussian (DoG)

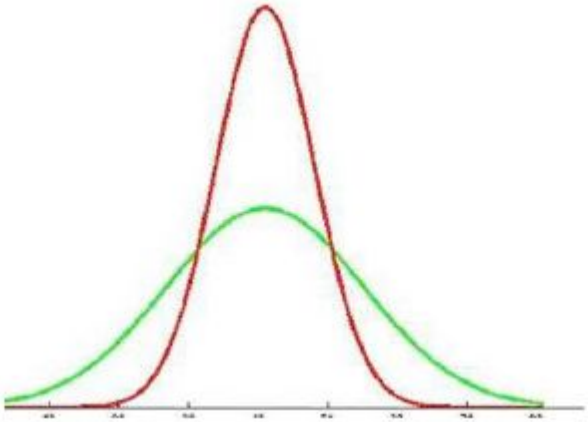
Fast approximation of LoG. Used by SIFT (next lecture).

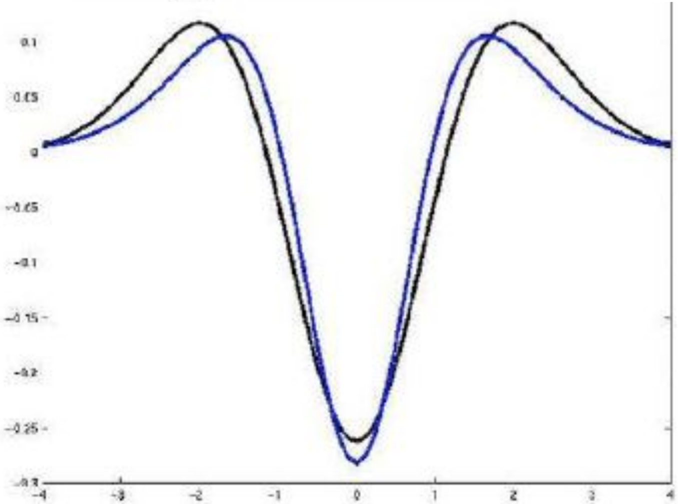
LoG can be approximate by a Difference of two Gaussians (DoG) at different scales.

$$\nabla^2 G_\sigma \approx G_{\sigma_1} - G_{\sigma_2}$$

Best approximation when:

$\sigma_1 = \frac{\sigma}{\sqrt{2}},\ \sigma_2 = \sqrt{2}\sigma$



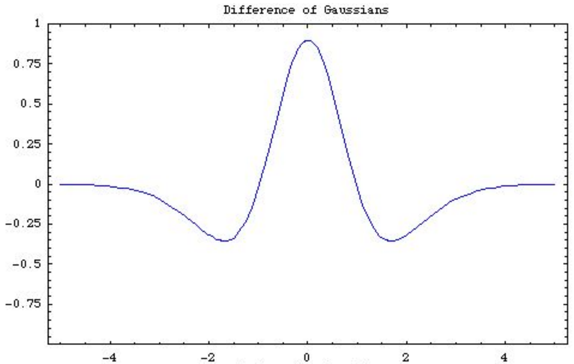


DoG filter

It is a band-pass filter.

$$\Gamma_{\sigma,K\sigma}(x,y) = I * \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)} - I * \frac{1}{2\pi K^2\sigma^2} e^{-(x^2+y^2)/(2K^2\sigma^2)}$$

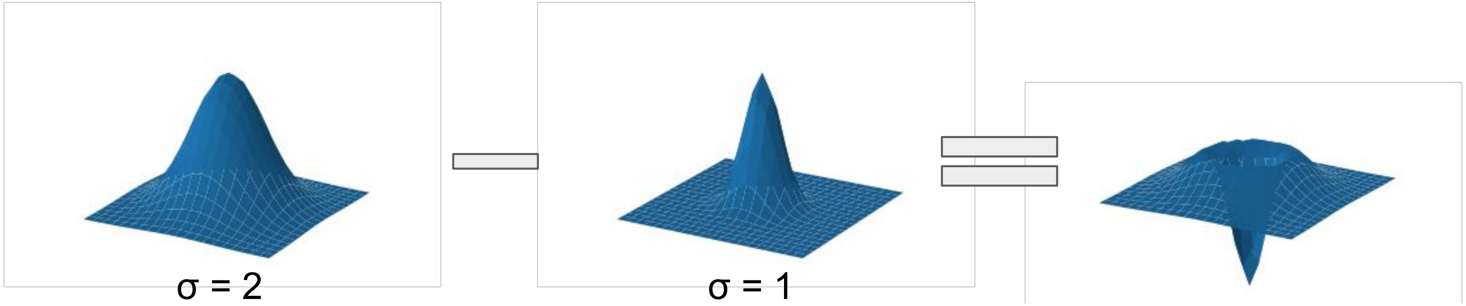
$$\Gamma_{\sigma,K\sigma}(x,y) = I * \left(\frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/(2\sigma^2)} - \frac{1}{2\pi K^2\sigma^2} e^{-(x^2+y^2)/(2K^2\sigma^2)}\right)$$



DoG filter

Intuition

- Gaussian (g) is a low pass filter
- Strongly reduce components with frequency $f < \sigma$
- $(g*I)$ low frequency components
- $I - (g*I)$ high frequency components
- $g(\sigma_1)*I - g(\sigma_2)*I \Leftarrow$ Components in between these frequencies
- $g(\sigma_1)*I - g(\sigma_2)*I = [g(\sigma_1) - g(\sigma_2)]*I$



DoG computation in practice

Take a image.

Blur it.

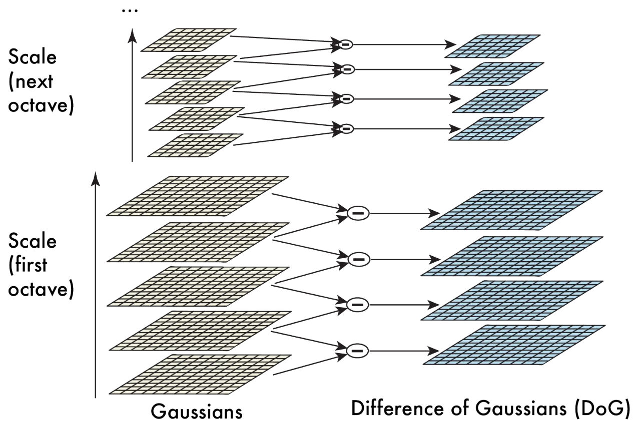
Take the difference.

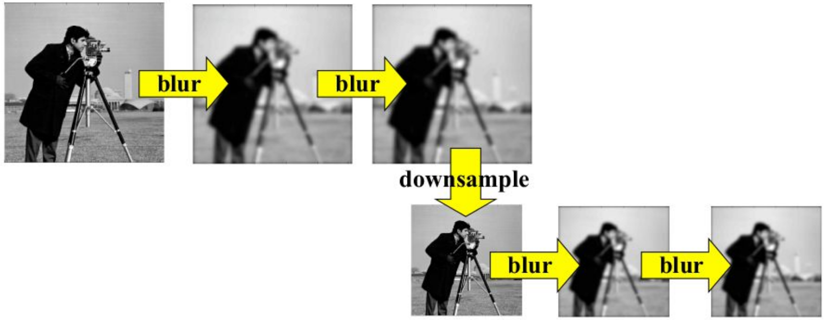
DoG scale generation trick

DoG computation: use “octaves”

- “Octave” because frequency doubles/halves between octaves
- If sigma = sqrt(2), then 3 levels per octave
- Downsample images for next octave (like double sized kernel)
- Compute the DoG between images

Illustration: D. Lowe





Crowley et.al., “Fast Computation of Characteristic Scale using a Half-Octave Pyramid.” Proc International Workshop on Cognitive Vision (CogVis), Zurich, Switzerland, 2002.

DoG: Corner selection

Throw out weak responses and edges

Estimate gradients

- Similar to Harris, look at nearby responses
- Not whole image, only a few points! Faster!
- Throw out weak responses

Find cornery things

- Same deal, structure matrix, use det and trace information (SIFT variant)

→

D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004., see p. 12

$$\frac{\mathrm{Tr}(\mathbf{H})^2}{\mathrm{Det}(\mathbf{H})} < \frac{(r+1)^2}{r} \qquad \mathbf{H} = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Determinant of Hessian (DoH)

Faster approximation. Used by SURF.
Better resistance to perspective

Computes the scale-normalized determinant of the Hessian (strength of the curvature at a given point)

$$\boxed{\det H_{norm} L = \sigma^2 (L_{xx} L_{yy} - L_{xy}^2)}$$

⇒ Precompute ***Lxx***, ***Lyy***, ***Lxy***
⇒ Blur them with the right sigma while computing **det *H L***: 3 additions
⇒ normalize: different scales – same value range

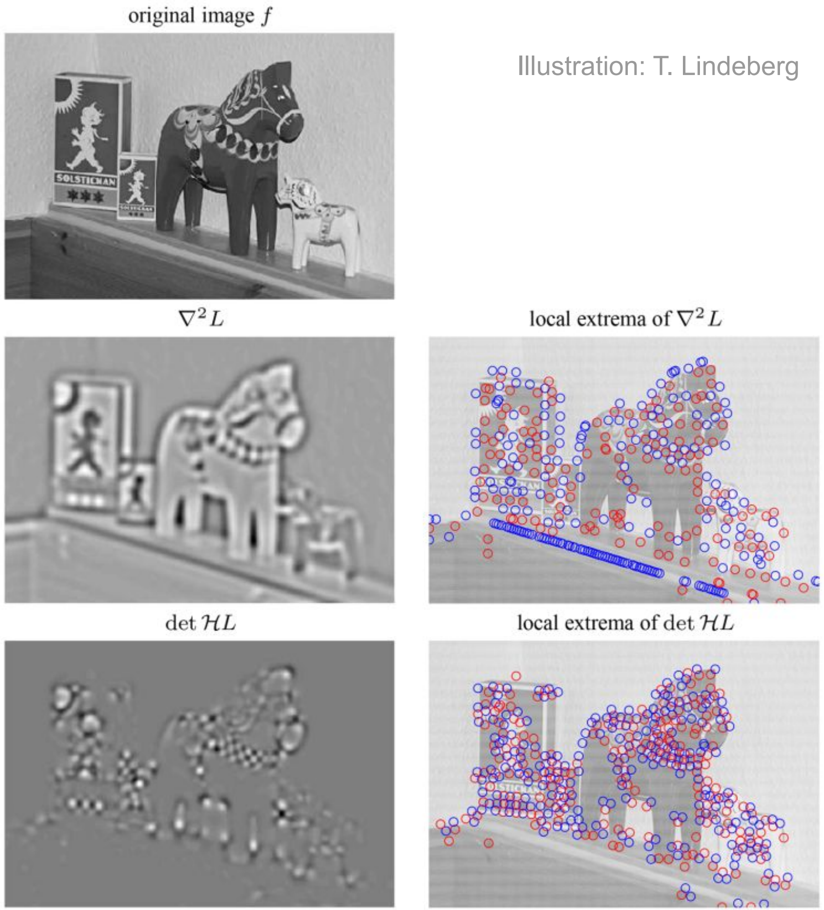
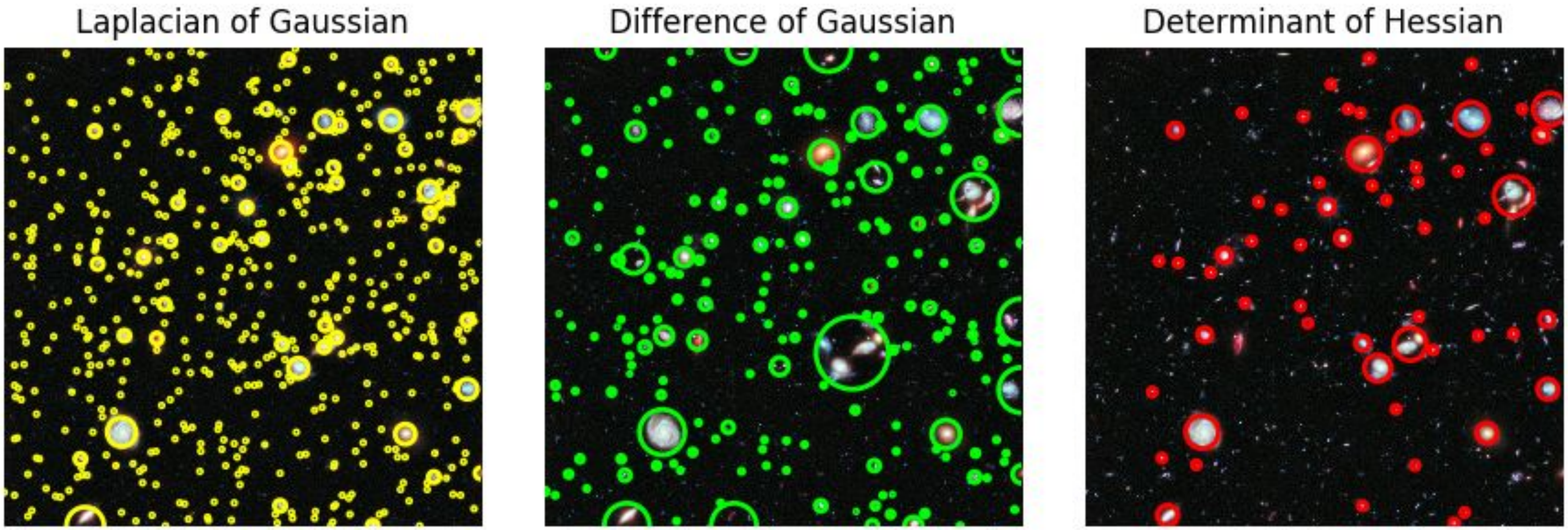


Illustration: T. Lindeberg

LoG vs DoG vs DoH



https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_blob.html

LoG, DoG, DoH summary

Pros

Very robust to transformations

- Perspective
- Blur

Adjustable size (scale)

Cons

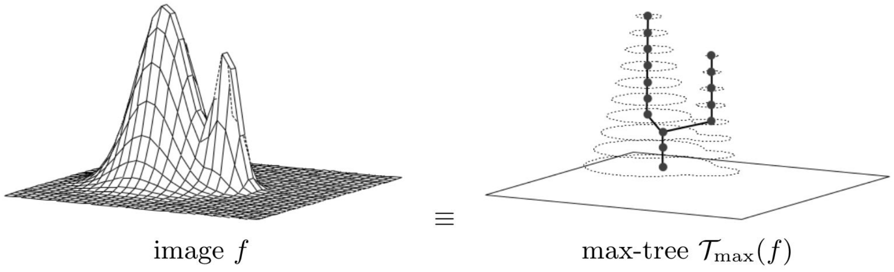
Slow

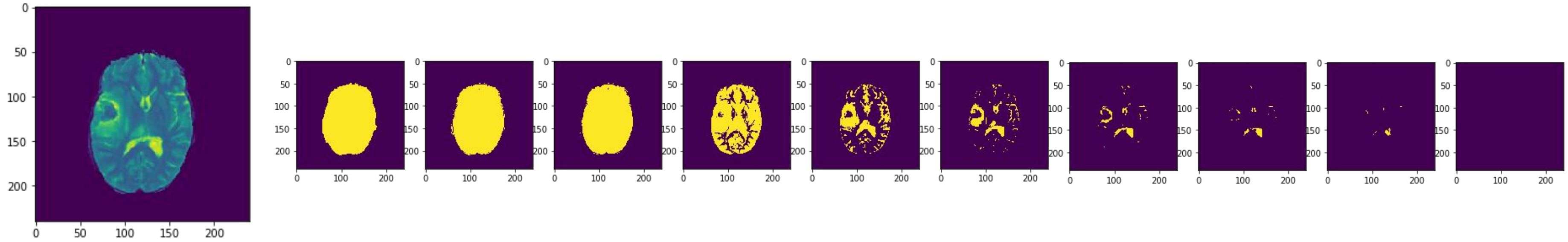
Blob detectors
MSER

Maximally Stable Extremal Regions (MSER)

Detects regions which are stable over thresholds.

1. **Create min- & max-tree of the image**
   tree of included components
   when thresholding the image
   at each possible level

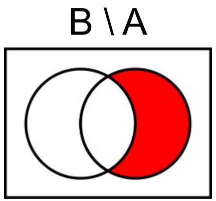




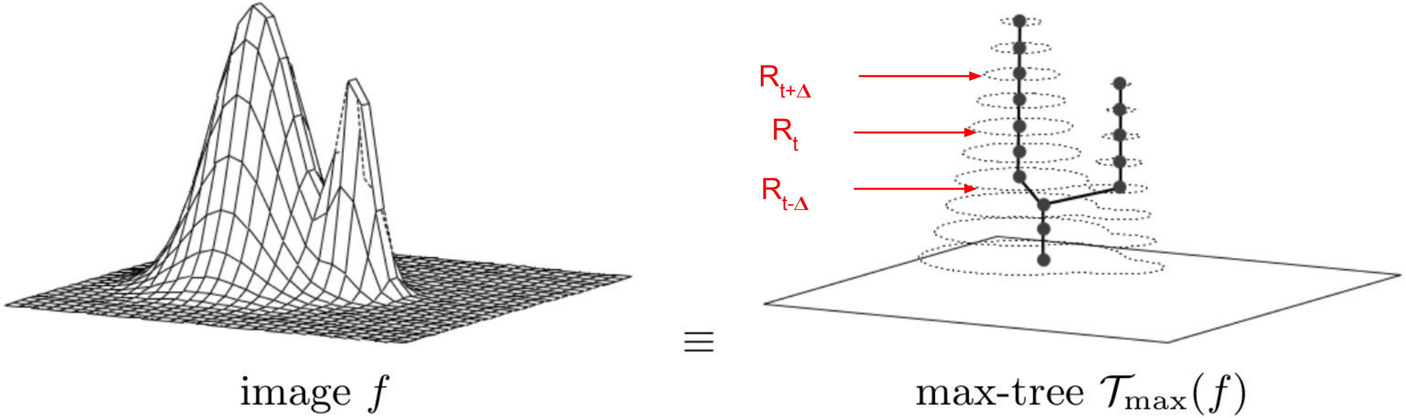
J. Matas, O. Chum, M. Urban, and T. Pajdla, “Robust wide-baseline stereo from maximally stable extremal regions,” Image and vision computing, vol. 22, no. 10, pp. 761–767, 2004.

Maximally Stable Extremal Regions (MSER)

2. **Select most stable regions** between t-Δ and t+Δ
   R_{t^*} is maximally stable iif $\mathbf{q(t) = | R_{t-\Delta} \setminus R_{t+\Delta} | \; / \; | R_t |}$
   as local minimum at t*



$| R |$ = card(R); Δ = parameter; $R_{t-\Delta} \setminus R_{t+\Delta}$ = set difference



MSER summary

Pros

Very robust to transformations

- Affine transformations
- Intensity changes

Quit fast

Cons

Does not support blur

Local feature detectors
Conclusion

Local feature detectors: Conclusion

Harris Stephens: Can be very stable when combined with DoG

Shi-Tomasi: Assumes affine transformation (avoid it with perspective)

DoG: slow but very robust (perspective, blur, illumination)

DoH: faster than DoG, misses small elements, better with perspective.

FAST: very fast, robust to perspective change (like DoG), but blur quickly kills it

MSER: fast, very stable, good choice when no blur

Classification

Feature detector	Edge	Corner	Blob
Canny	X		
Sobel	X		
Harris & Stephens / Plessey / Shi–Tomasi	X	X	
Shi & Tomasi		X	
FAST		X	
Laplacian of Gaussian		X	X
Difference of Gaussians		X	X
Determinant of Hessian		X	X
MSER			X

https://en.wikipedia.org/wiki/Feature_detection_(computer_vision)