Descriptors

EECS 442 – David Fouhey

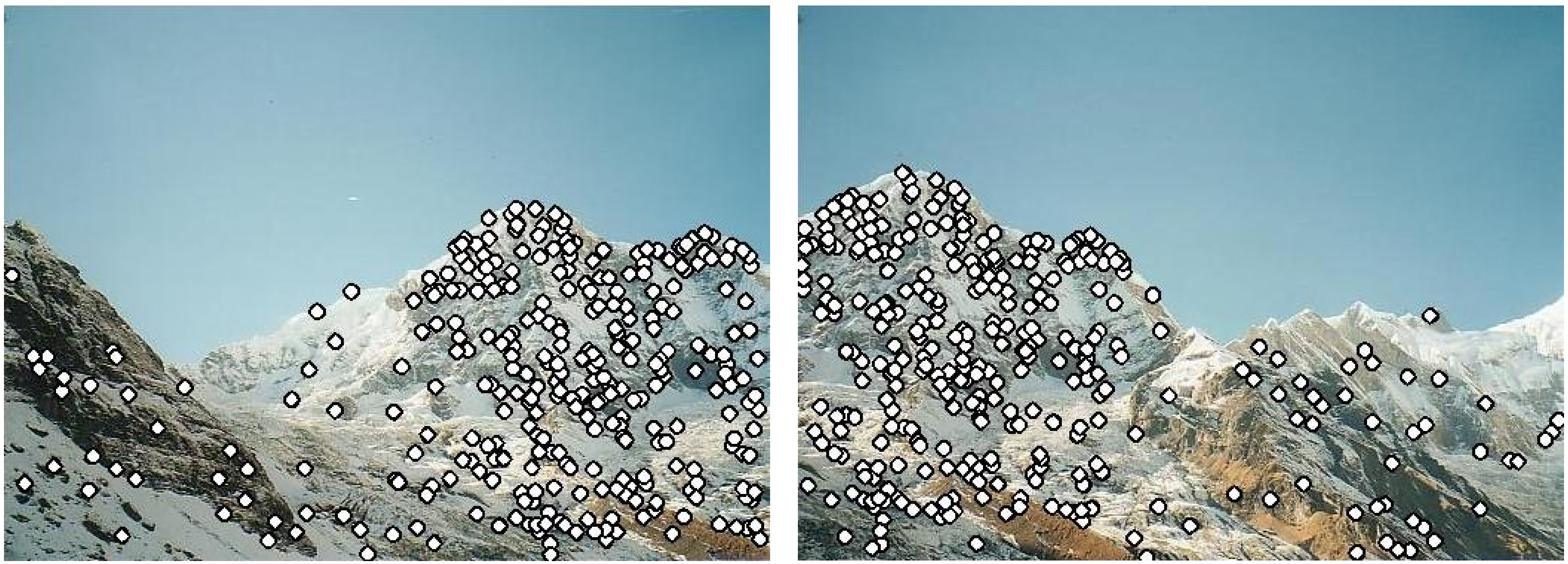
Winter 2023, University of Michigan

https://web.eecs.umich.edu/~fouhey/teaching/EECS442_W23/

Administrivia

- Extra OH, zoom on Tuesday. Post HW topics you think are challenging; will try to cover top-voted questions.

Recap: Motivation

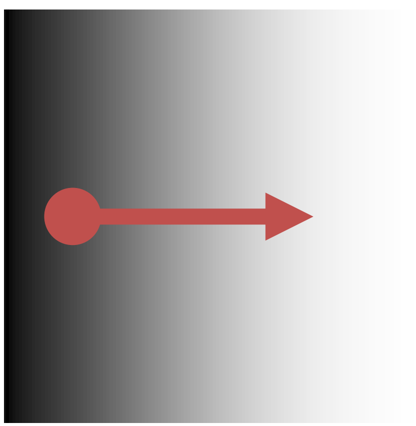


1: find corners+features

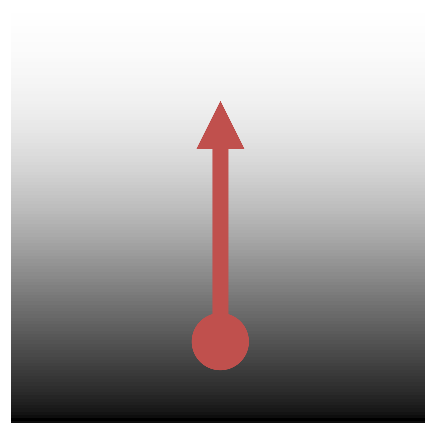
Image credit: M. Brown

Last Time – Gradients

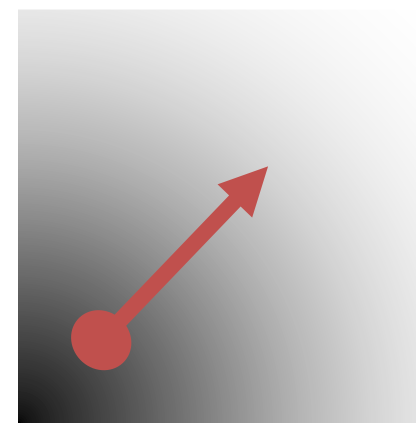
Image gradients – treat image like function of x,y – gives edges, corners, etc.



$$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$$



$$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$$

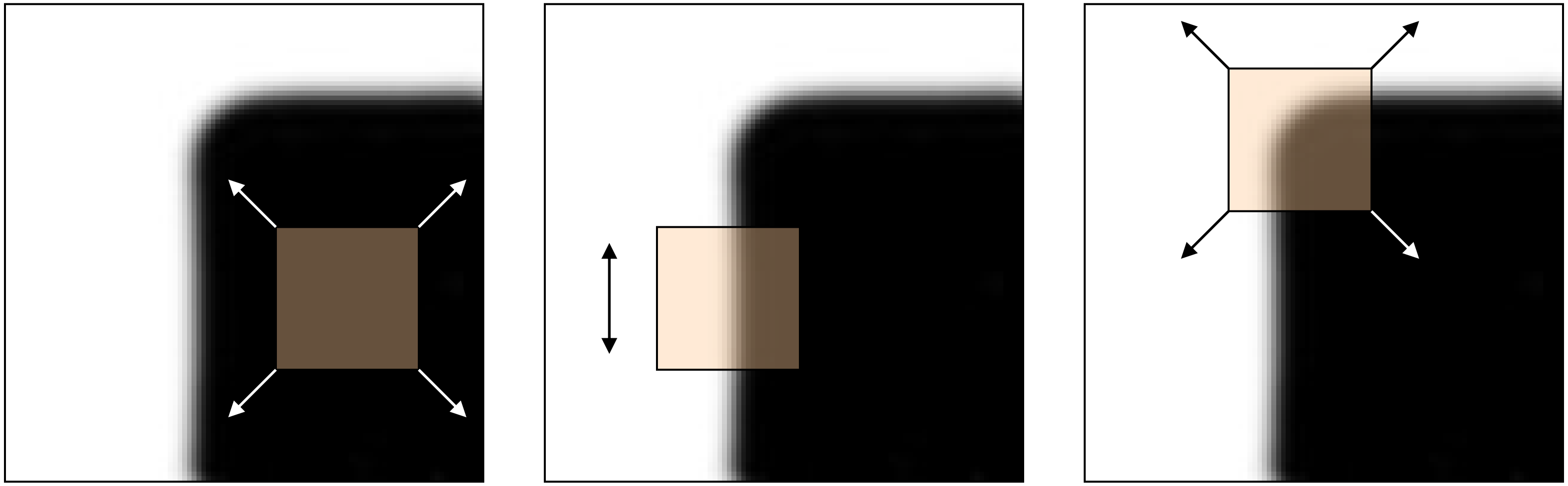


$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

Figure credit: S. Seitz

Last Time – Corner Detection

Can localize the location, or any shift → big intensity change.



"flat" region: no change in all directions

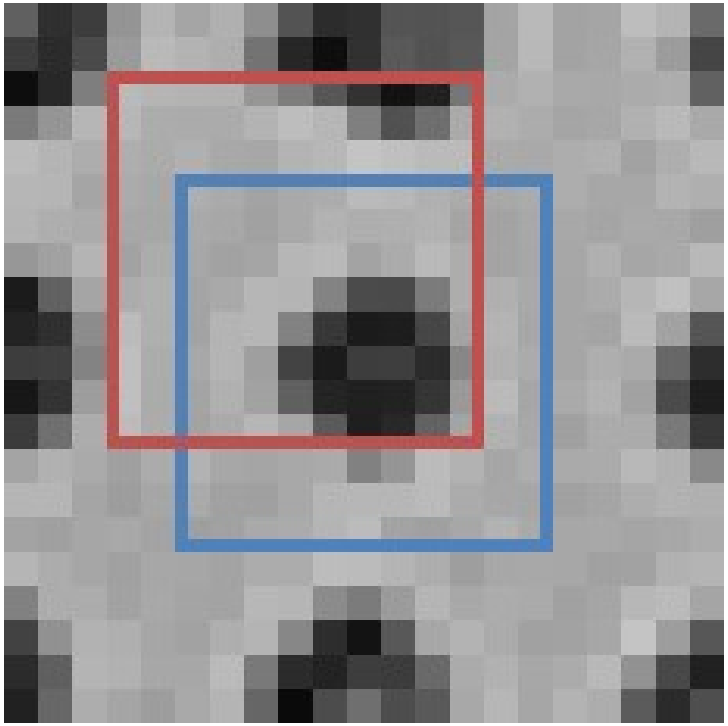
"edge": no change along the edge direction

"corner": significant change in all directions

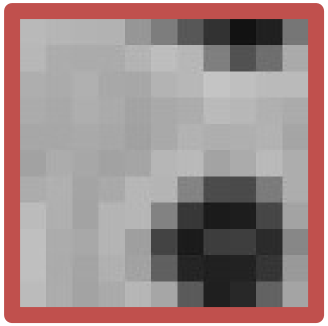
Diagram credit: S. Lazebnik

Last Time – Corner Detection

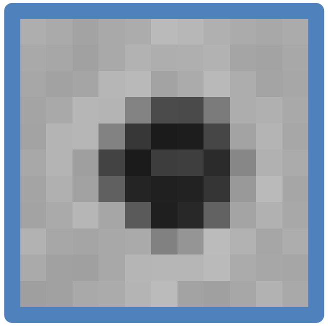
Zoom-In at x,y



Window with and w/o Offset



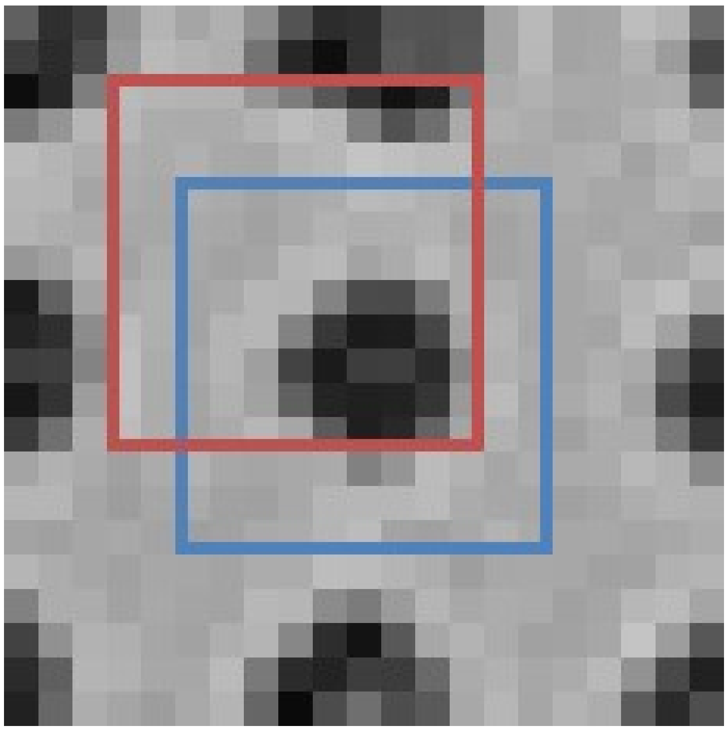
“Window”
At x+u, y+v
Here: u=-2,v=-3



“Window”
At x, y

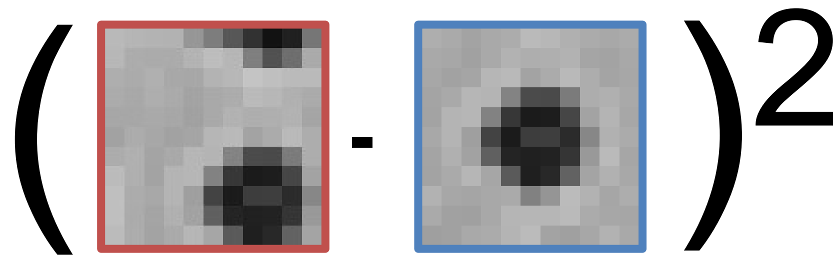
Last Time – Corner Detection

Zoom-In at x,y



Error (Sum Sqs) for u,v offset

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



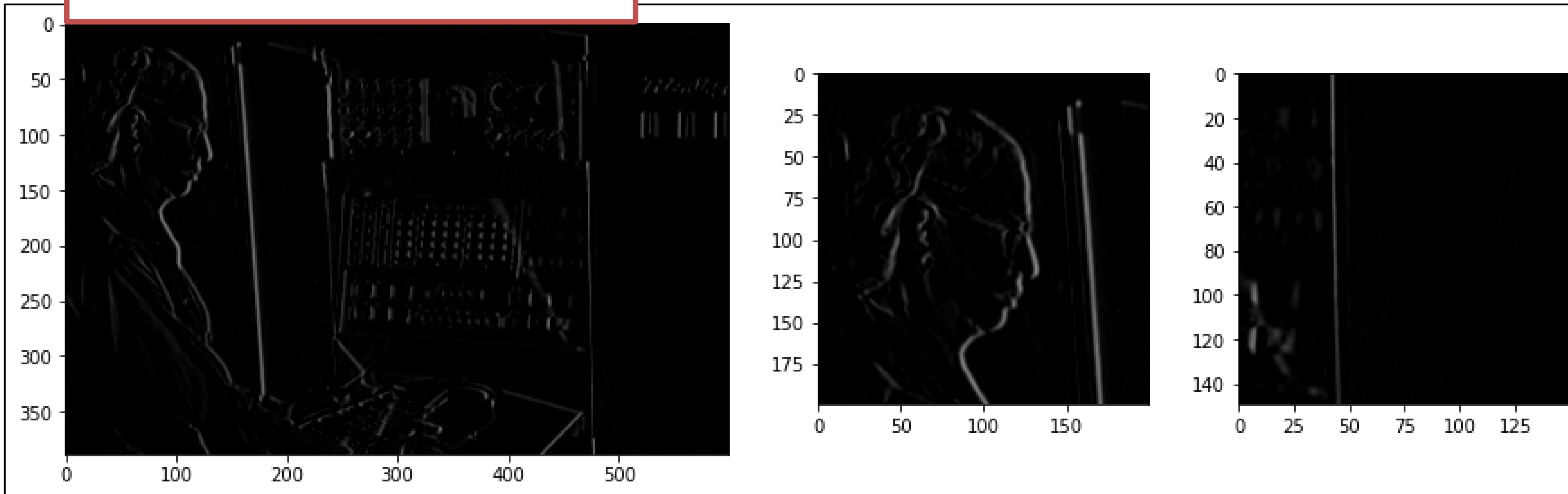
Formalizing Corner Detection

By linearizing image, we can approximate E(u,v) with quadratic function of u and v

$$E(u,v) \approx \sum_{(x,y)\in W} \left(I_x^2 u^2 + 2I_x I_y uv + I_y^2 v^2\right)$$
$$= [u,v]\boldsymbol{M}[u,v]^T$$

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

Ix = x derivative

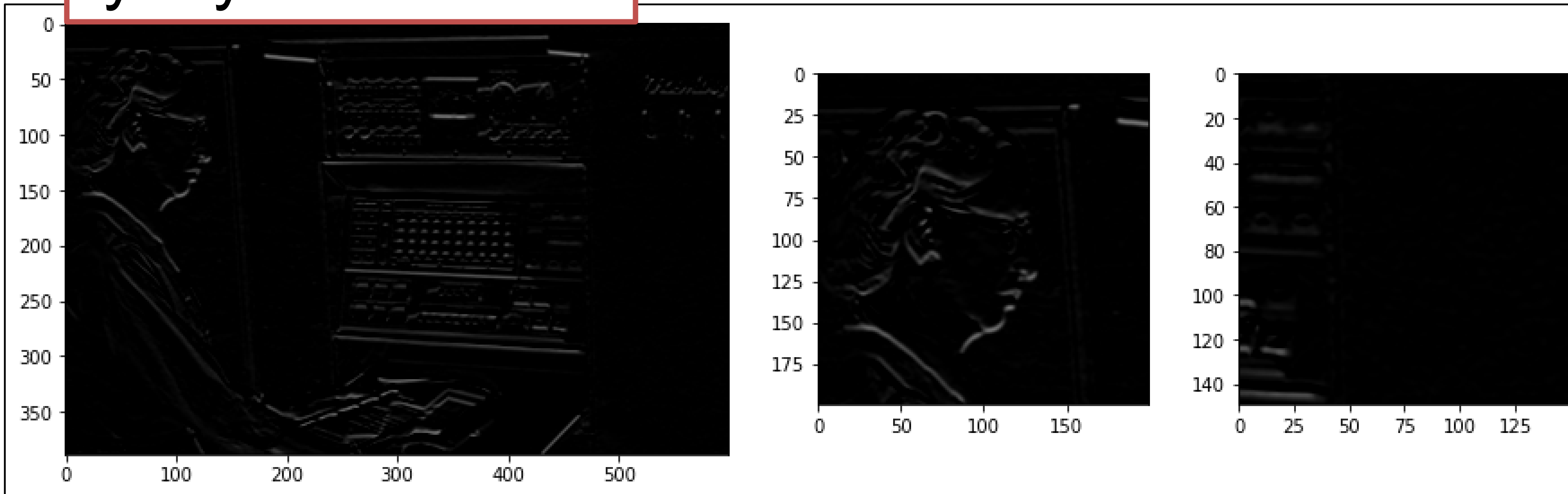


$$(x,y)\in W$$

$$= [u,v]\boldsymbol{M}[u,v]^T$$

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

Iy = y derivative



$$(x,y)\in W$$

$$= [u,v]\boldsymbol{M}[u,v]^T$$

$$\boldsymbol{M} = \begin{bmatrix} \sum\limits_{x,y\in W} I_x^2 & \sum\limits_{x,y\in W} I_x I_y \\ \sum\limits_{x,y\in W} I_x I_y & \sum\limits_{x,y\in W} I_y^2 \end{bmatrix}$$

Sum goes over all the pixels in window W:

$$\sum_{x,y\in W} I_x^2 = \sum_{x,y\in W} \left(I_x[y,x]\right)^2$$

i.e., sum of squares of x gradients in window

$$= [u,v]\boldsymbol{M}[u,v]^T$$

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

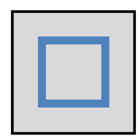
Intuitively what is M?

Pretend gradients are *either* vertical or horizontal at a pixel (so Ix Iy = 0)

Obviously Wrong!

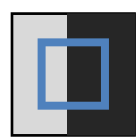
$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y \in W} I_x^2 & \sum_{x,y \in W} I_x I_y \\ \sum_{x,y \in W} I_x I_y & \sum_{x,y \in W} I_y^2 \end{bmatrix} \approx \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$$

a,b both small: flat



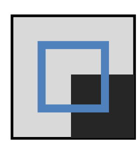
$$\begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}$$

One big, other small: edge



$$\begin{bmatrix} 50 & 0 \\ 0 & 0.1 \end{bmatrix} \text{ or } \begin{bmatrix} 0.1 & 0 \\ 0 & 50 \end{bmatrix}$$

a,b both big: corner



$$\begin{bmatrix} 50 & 0 \\ 0 & 50 \end{bmatrix}$$

Intuitively what is M?

Pretend gradients are *either* vertical or horizontal at a pixel (so Ix Iy = 0)

$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y \in W} I_x^2 & \sum_{x,y \in W} I_x I_y \\ \sum_{x,y \in W} I_x I_y & \sum_{x,y \in W} I_y^2 \end{bmatrix} \approx? \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}$$

a,b both small: flat

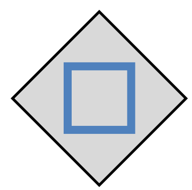
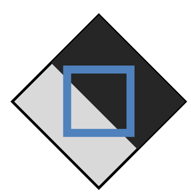
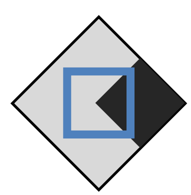


Image might be rotated by rotation θ!

One big, other small: edge



a,b both big: corner

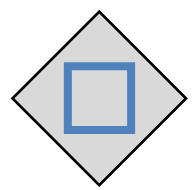


Intuitively what is M?

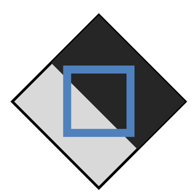
Pretend gradients are *either* vertical or horizontal at a pixel (so Ix Iy = 0)

$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y\in W} I_x^2 & \sum_{x,y\in W} I_x I_y \\ \sum_{x,y\in W} I_x I_y & \sum_{x,y\in W} I_y^2 \end{bmatrix} = \boldsymbol{V}^{-1} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \boldsymbol{V}$$

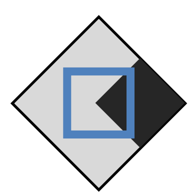
a,b both small: flat



One big, other small: edge



a,b both big: corner



If image rotated by rotation θ / matrix **V**

M will look like

$$\boldsymbol{V}^{-1} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \boldsymbol{V}$$

So What Now?

Can calculate M at pixel, by summing nearby gradients, but need access to a and b.

$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y \in W} I_x^2 & \sum_{x,y \in W} I_x I_y \\ \sum_{x,y \in W} I_x I_y & \sum_{x,y \in W} I_y^2 \end{bmatrix} = \boldsymbol{V}^{-\mathbf{1}} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \boldsymbol{V}$$

Given **M**, can decompose it into eigenvectors **V** and eigenvalues λ_1, λ_2 with $\mathbf{M} = \boldsymbol{V}^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \boldsymbol{V}$.

Really slow. Why?

So What Now?

Can calculate M at pixel, by summing nearby gradients, but need access to a and b.

$$\boldsymbol{M} = \begin{bmatrix} \sum_{x,y\in W} I_x^2 & \sum_{x,y\in W} I_x I_y \\ \sum_{x,y\in W} I_x I_y & \sum_{x,y\in W} I_y^2 \end{bmatrix} = \boldsymbol{V}^{-1} \begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix} \boldsymbol{V}$$

Instead: compute quantity R from **M**

$$R = \det(\boldsymbol{M}) - \alpha \operatorname{trace}(\boldsymbol{M})^2 = \lambda_1 \lambda_2 - \alpha\, (\lambda_1 + \lambda_2)^2$$

Easy fast formula for 2x2

Fast – sum the diagonal

Empirical value, usually 0.04-0.06

The tl;dr

TL;DR: Taylor expansion for error E(u,v). All terms in equation are sums of image gradients and in **M**

<u>Should know</u>

$$\boldsymbol{M} = \overbrace{\begin{bmatrix} \sum_{x,y \in W} I_x^2 & \sum_{x,y \in W} I_x I_y \\ \sum_{x,y \in W} I_x I_y & \sum_{x,y \in W} I_y^2 \end{bmatrix}}^{\text{Can compute at each pixel}} =$$

Ix = Ix at point (x,y), Iy = Iy at point (x,y)

<u>Optional</u>

Directions

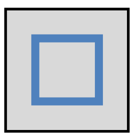
Amounts

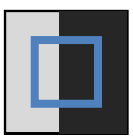
$$\boldsymbol{V}^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \boldsymbol{V}$$

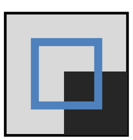
Putting It Together

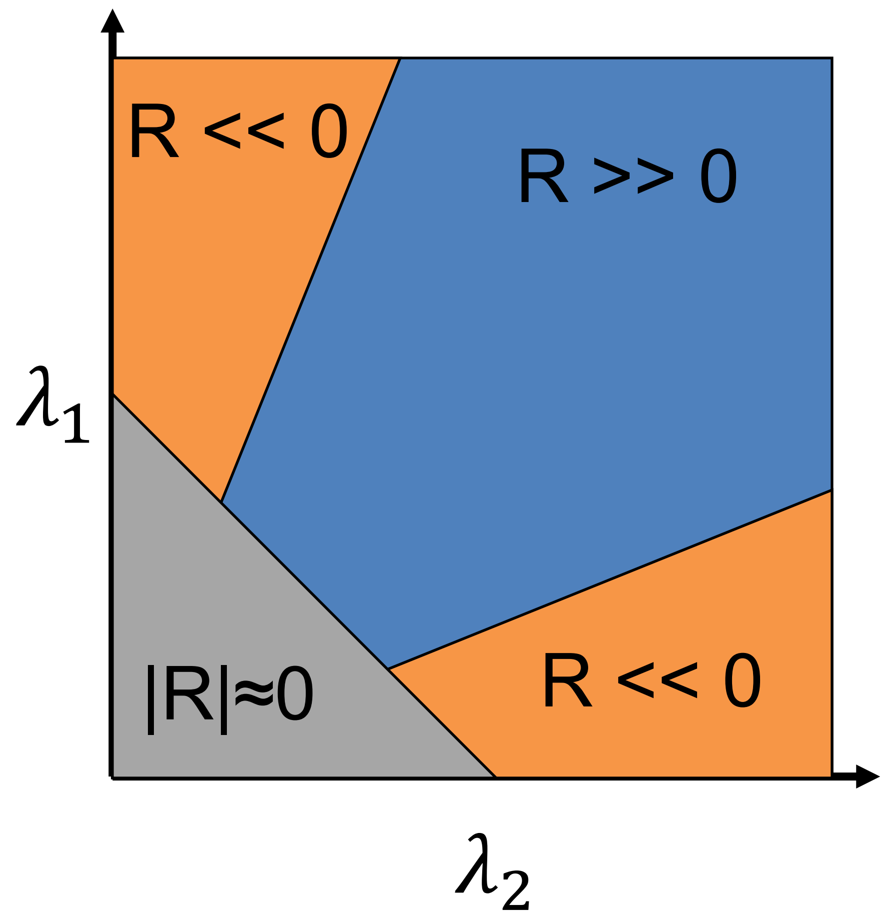
$$R = \det(\boldsymbol{M}) - \alpha\,\mathrm{trace}(\boldsymbol{M})^2 = \lambda_1\lambda_2 - \alpha\,(\lambda_1 + \lambda_2)^2$$

det, trace are *fast*

flat  $\lambda_1, \lambda_2 \approx 0$

edge  $\lambda_1 \gg \lambda_2 \gg 0$
$\lambda_2 \gg \lambda_1 \gg 0$

corner  $\lambda_1 \approx \lambda_2 \gg 0$



Remake of standard diagram from S. Lazebnik from original Harris paper.

In Practice

1. Compute partial derivatives Ix, Iy per pixel
2. Compute **M** at each pixel, using Gaussian weighting w

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

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

Slide credit: S. Lazebnik

In Practice

1. Compute partial derivatives Ix, Iy per pixel
2. Compute **M** at each pixel, using Gaussian weighting w
3. Compute response function R

$$R = \det(\boldsymbol{M}) - \alpha\, trace(\boldsymbol{M})^2$$
$$= \lambda_1\lambda_2 - \alpha(\lambda_1 + \lambda_2)^2$$

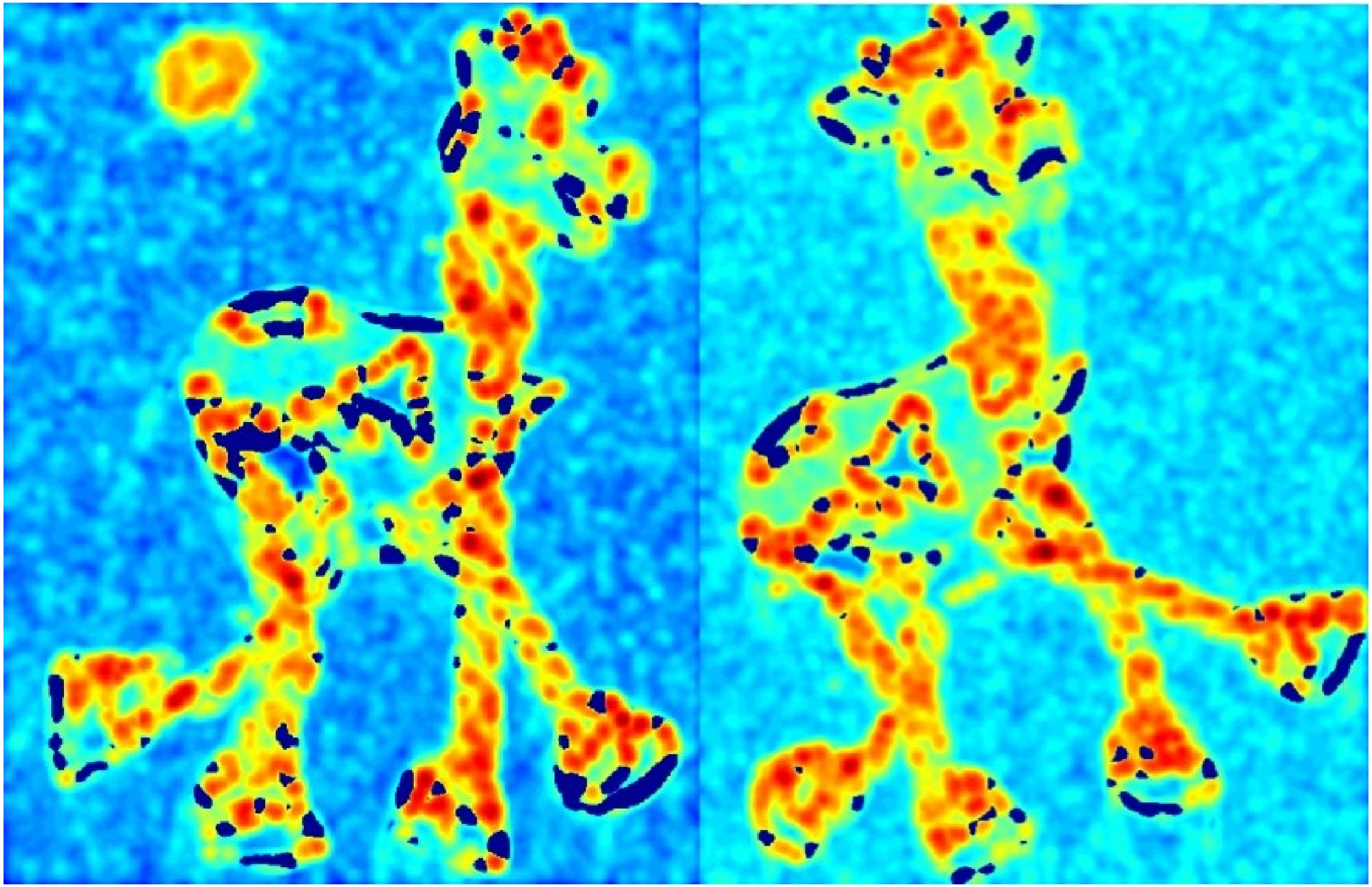
C.Harris and M.Stephens. "A Combined Corner and Edge Detector." *Proceedings of the 4th Alvey Vision Conference*: pages 147—151, 1988.

Slide credit: S. Lazebnik

Computing R

Slide credit: S. Lazebnik

Computing R



Slide credit: S. Lazebnik

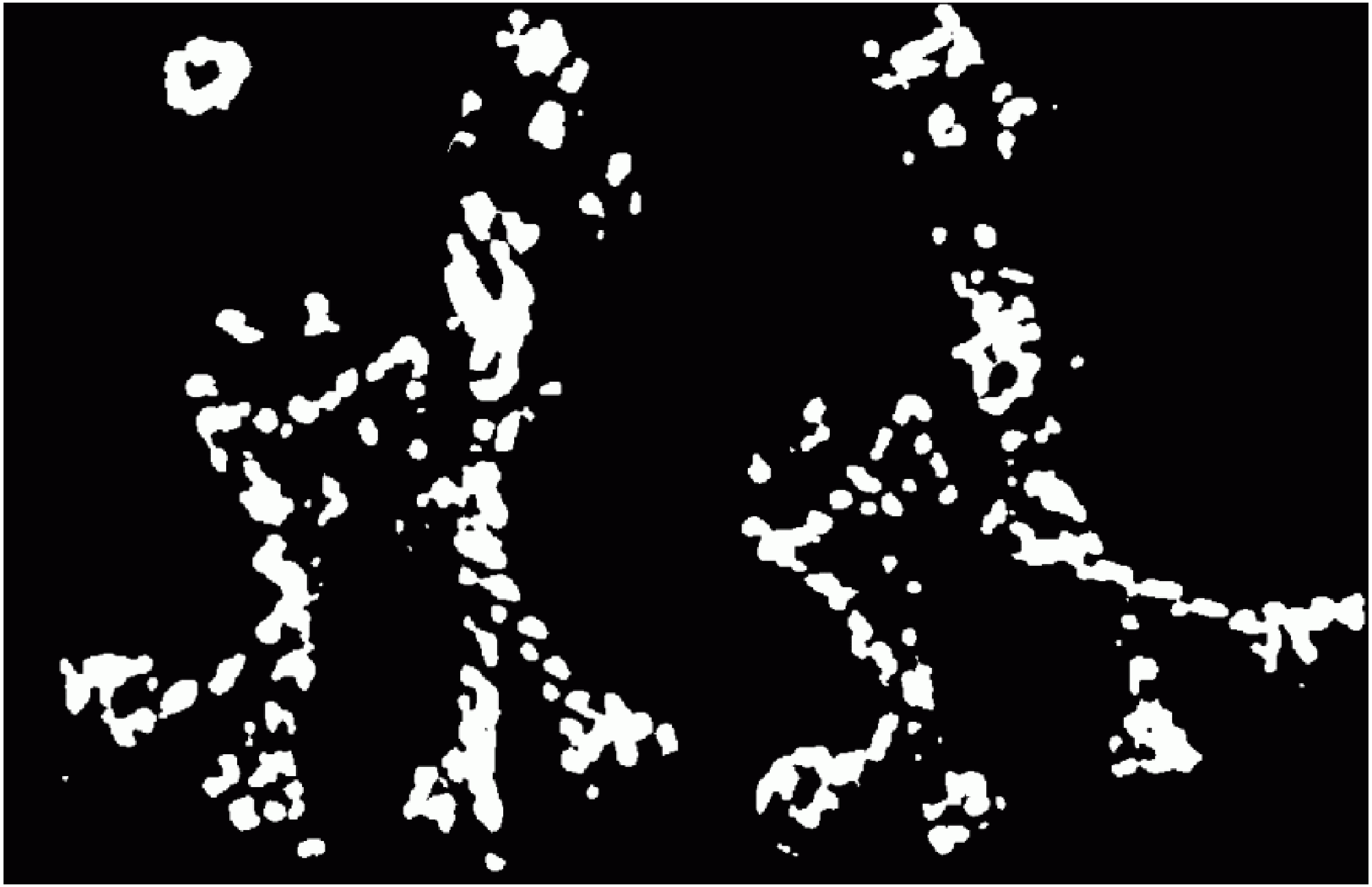
In Practice

1. Compute partial derivatives Ix, Iy per pixel
2. Compute **M** at each pixel, using Gaussian weighting w
3. Compute response function R
4. Threshold R

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

Slide credit: S. Lazebnik

Thresholded R



Slide credit: S. Lazebnik

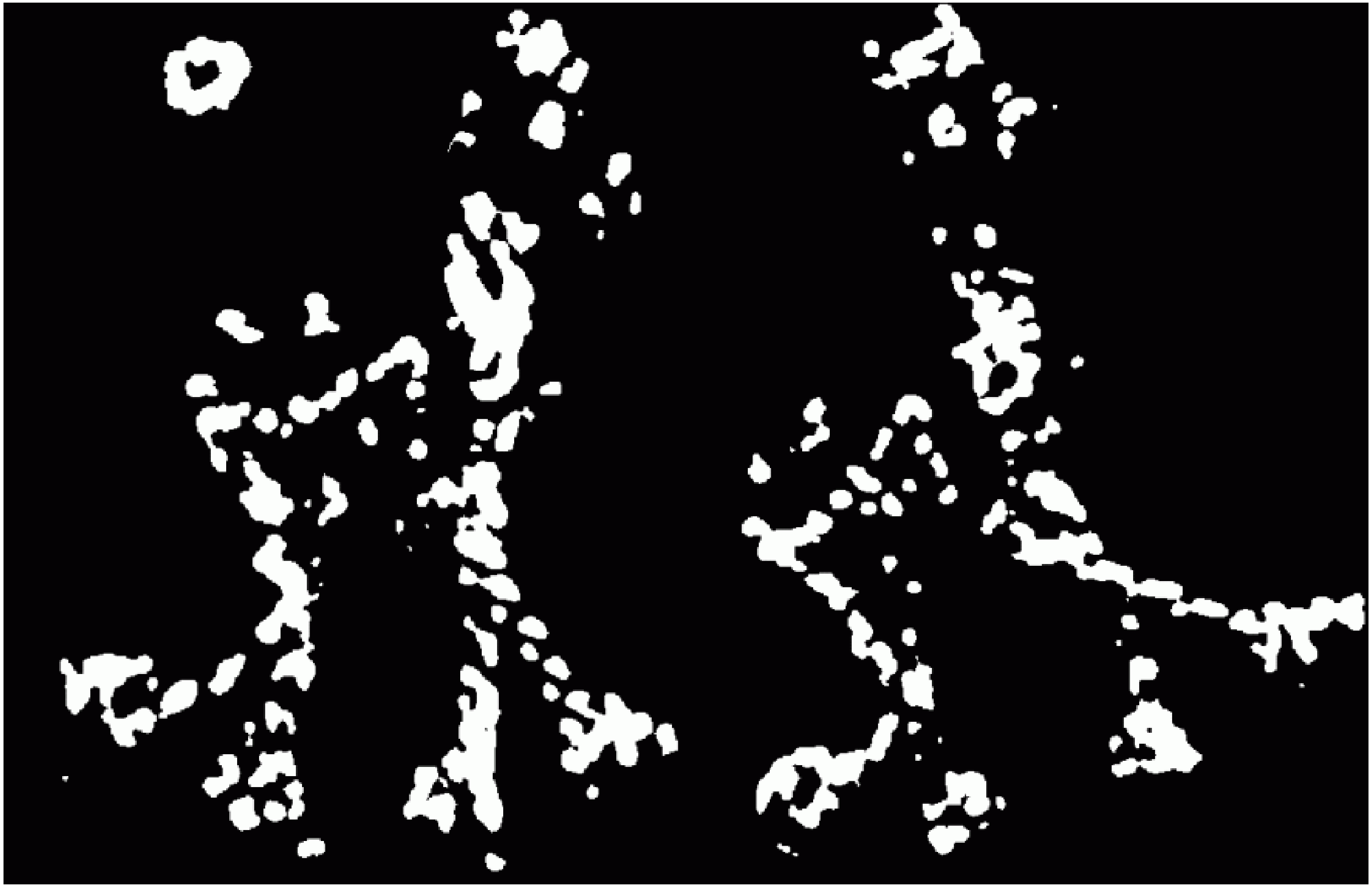
In Practice

1. Compute partial derivatives Ix, Iy per pixel
2. Compute **M** at each pixel, using Gaussian weighting w
3. Compute response function R
4. Threshold R
5. Take only local maxima (called 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.

Slide credit: S. Lazebnik

Thresholded



Slide credit: S. Lazebnik

Final Results

Slide credit: S. Lazebnik

Desirable Properties

If our detectors are repeatable, they should be:

- **Invariant** to some things: image is transformed and corners remain the same
- **Covariant/equivariant** with some things: image is transformed and corners transform with it.

Slide credit: S. Lazebnik

Recall Motivating Problem

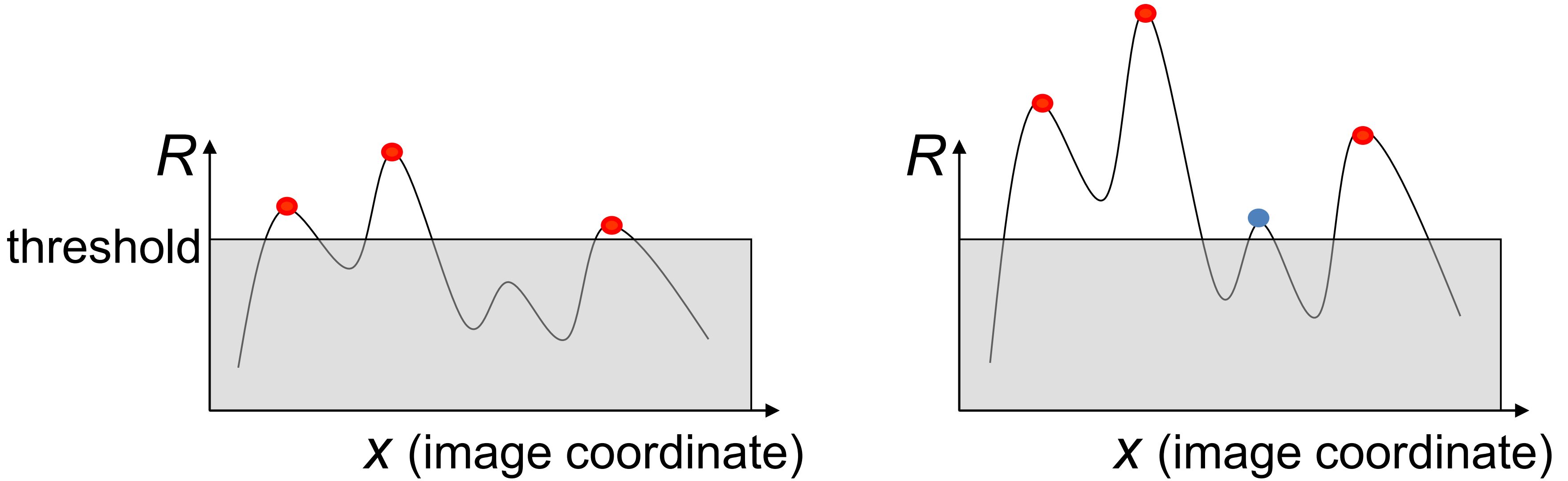
Images may be different in lighting and geometry

Affine Intensity Change

$$I_{new} = aI_{old} + b$$

M only depends on derivatives, so *b* is irrelevant

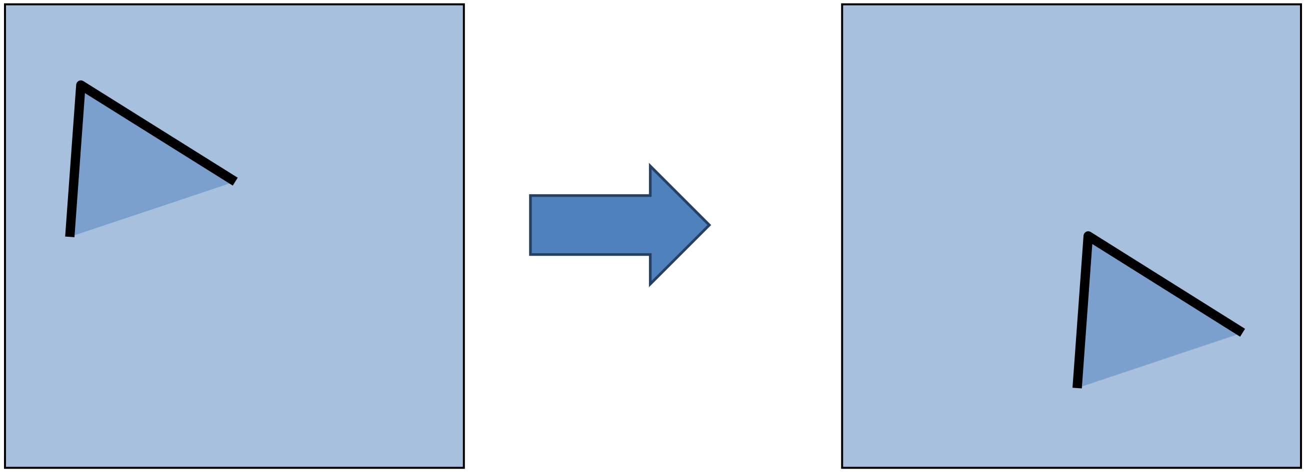
But *a* scales derivatives and there's a threshold



Partially invariant to affine intensity changes

Slide credit: S. Lazebnik

Image Translation

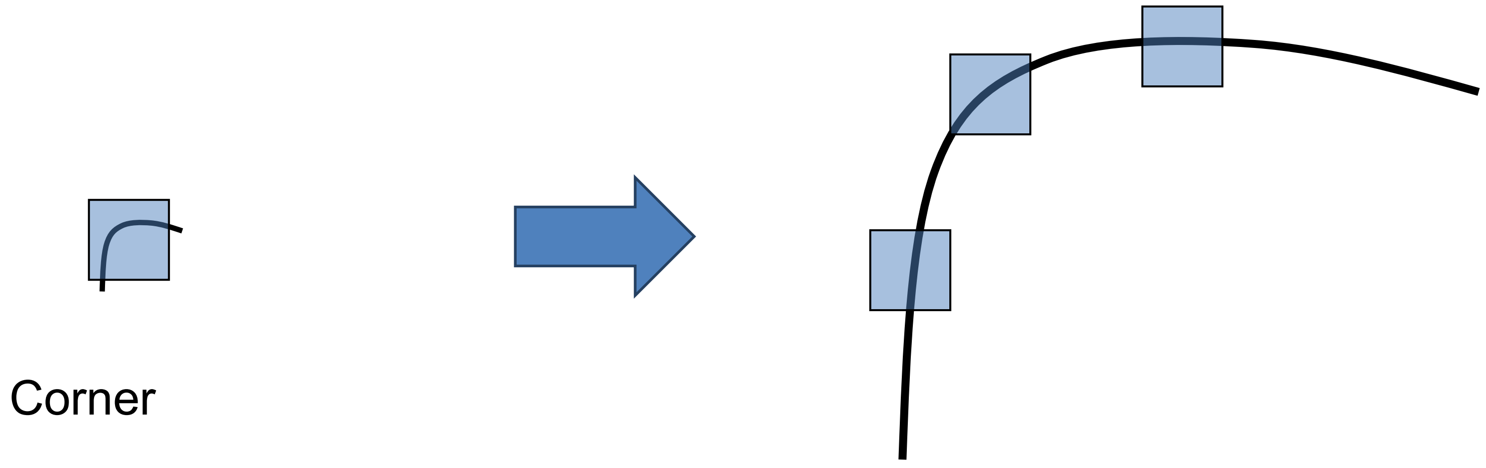


All done with convolution. Convolution is translation equivariant.

Equivariant with translation

Slide credit: S. Lazebnik

Image Scaling



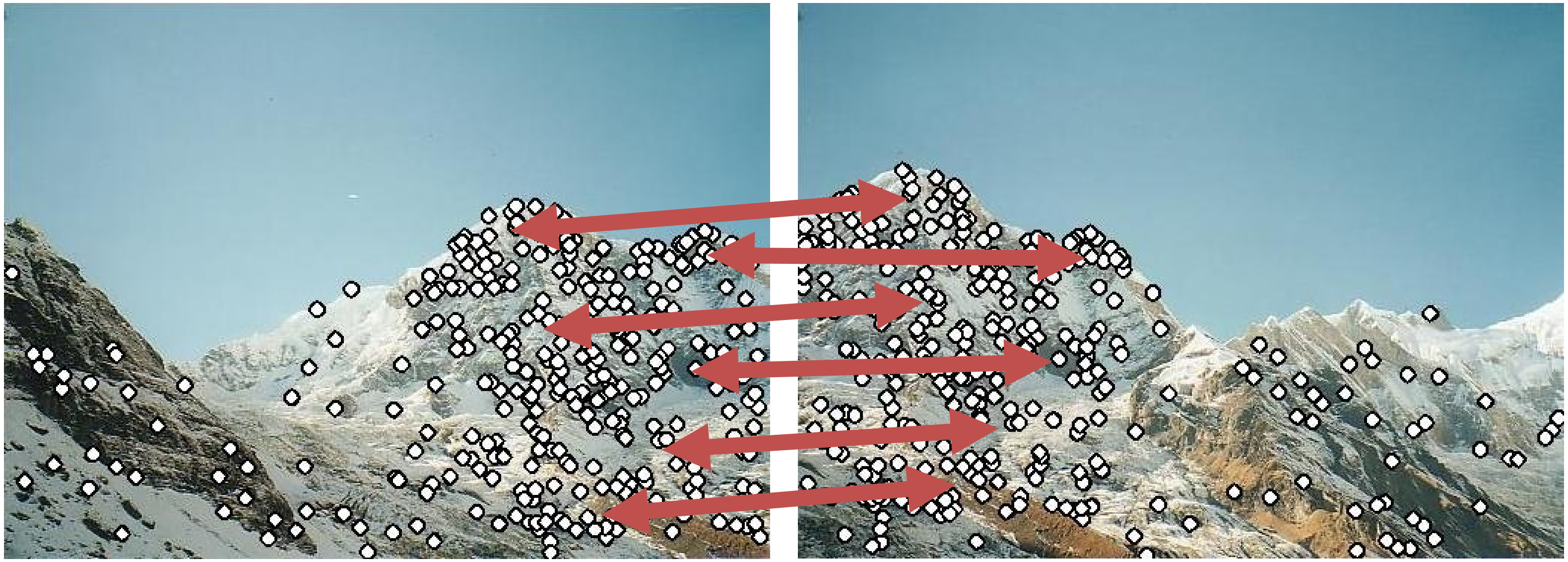
One pixel can become many pixels and vice-versa.

Not equivariant with scaling

How do we fix this?

Slide credit: S. Lazebnik

Recap: Motivation



1: find corners+features

2: match based on local image data

How?

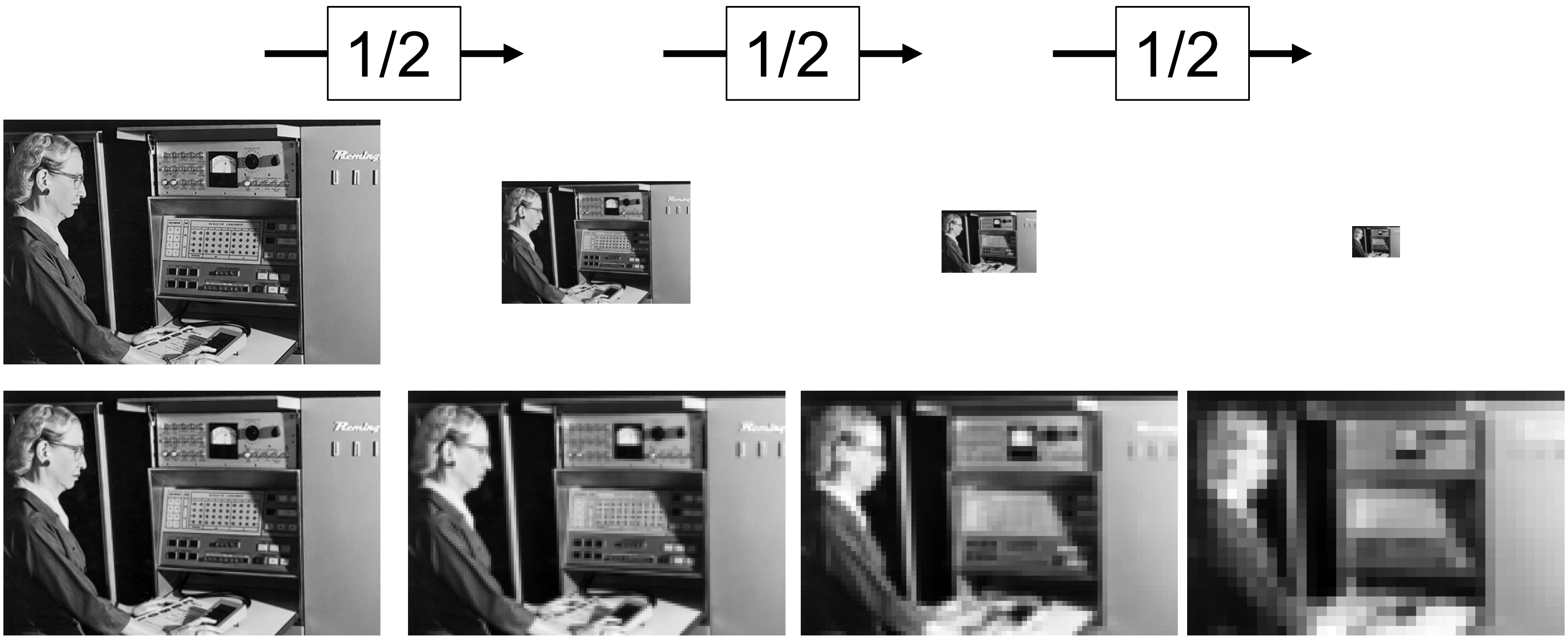
Image credit: M. Brown

Today

- Fixing scaling by making detectors in both location **and scale**
- Enabling matching between features by **describing regions**

Key Idea: Scale Space

Left to right: each image is half-sized
Upsampled with big pixels below

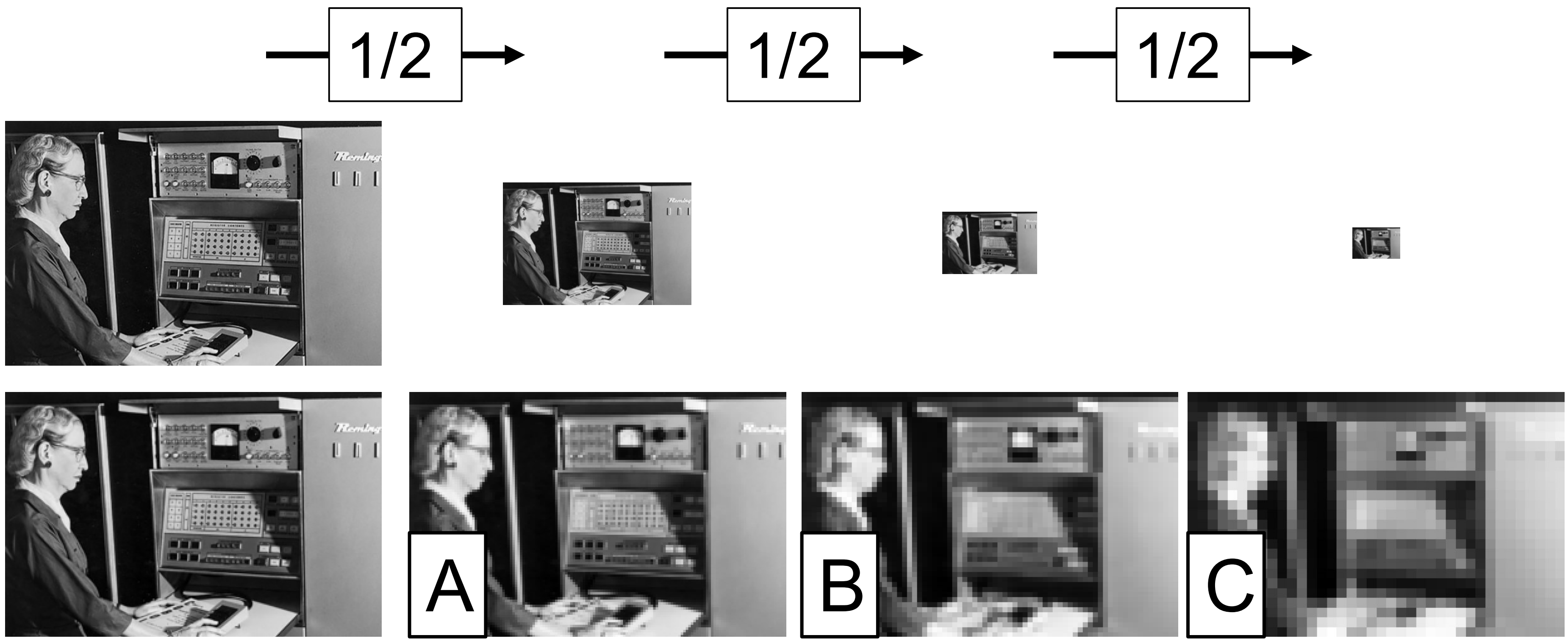


Note: I'm also slightly blurring to prevent aliasing (https://en.wikipedia.org/wiki/Aliasing)

Key Idea: Scale Space

Left to right: each image is half-sized

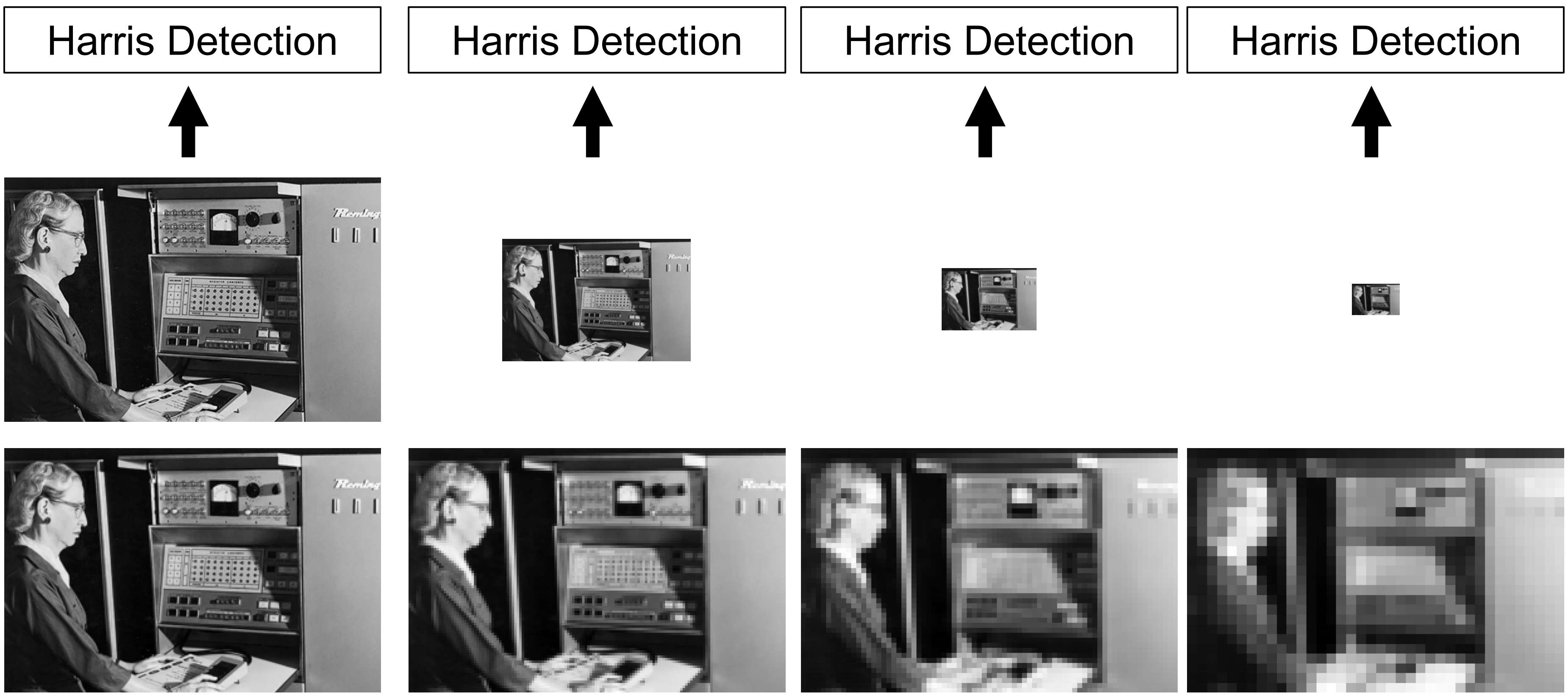
If I apply a KxK filter, how much of the original image does it see in each image?



Note: I'm also slightly blurring to prevent aliasing (https://en.wikipedia.org/wiki/Aliasing)

Solution to Scales

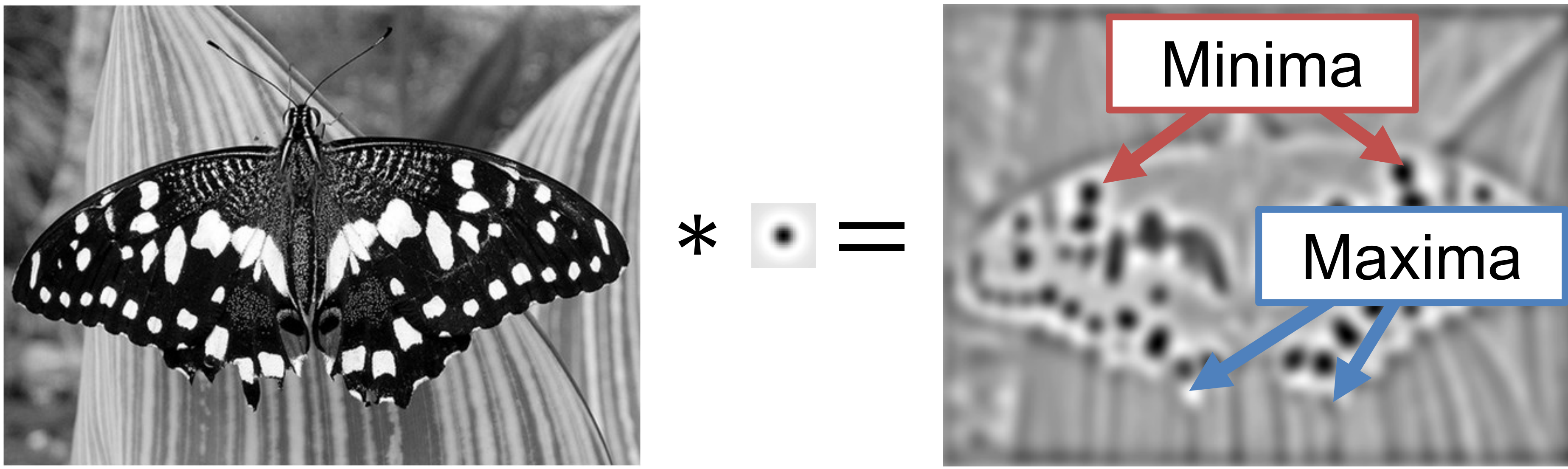
Try them all!



See: Multi-Image Matching using Multi-Scale Oriented Patches, Brown et al. CVPR 2005

Blob Detection

Another detector (has some nice properties)

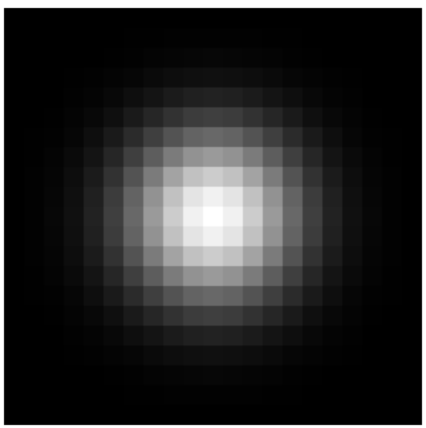


Find maxima ***and minima*** of blob filter response in scale ***and space***

Slide credit: N. Snavely

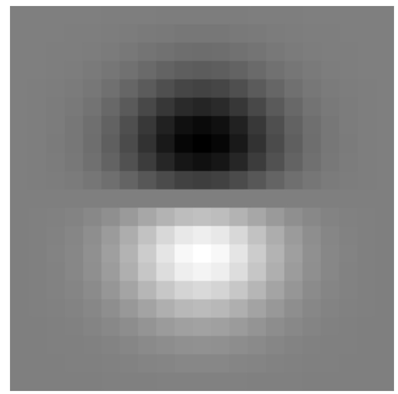
Gaussian Derivatives

Gaussian

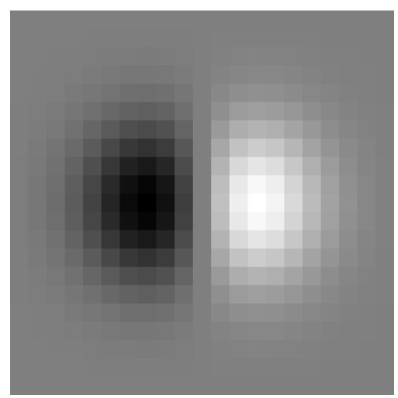


1st Deriv

$$\frac{\partial}{\partial y} g$$

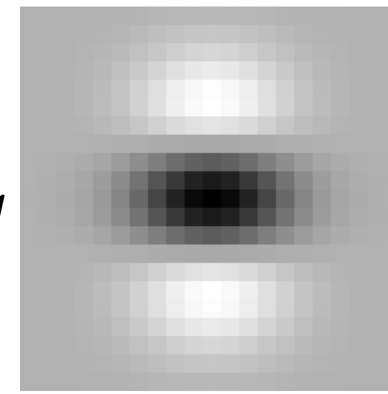


$$\frac{\partial}{\partial x} g$$

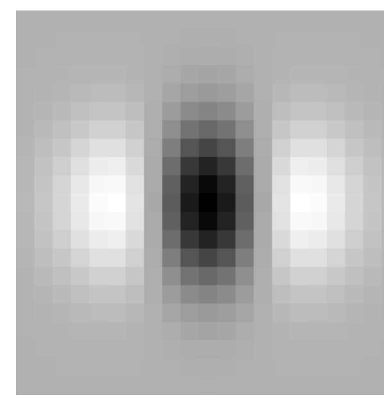


2nd Deriv

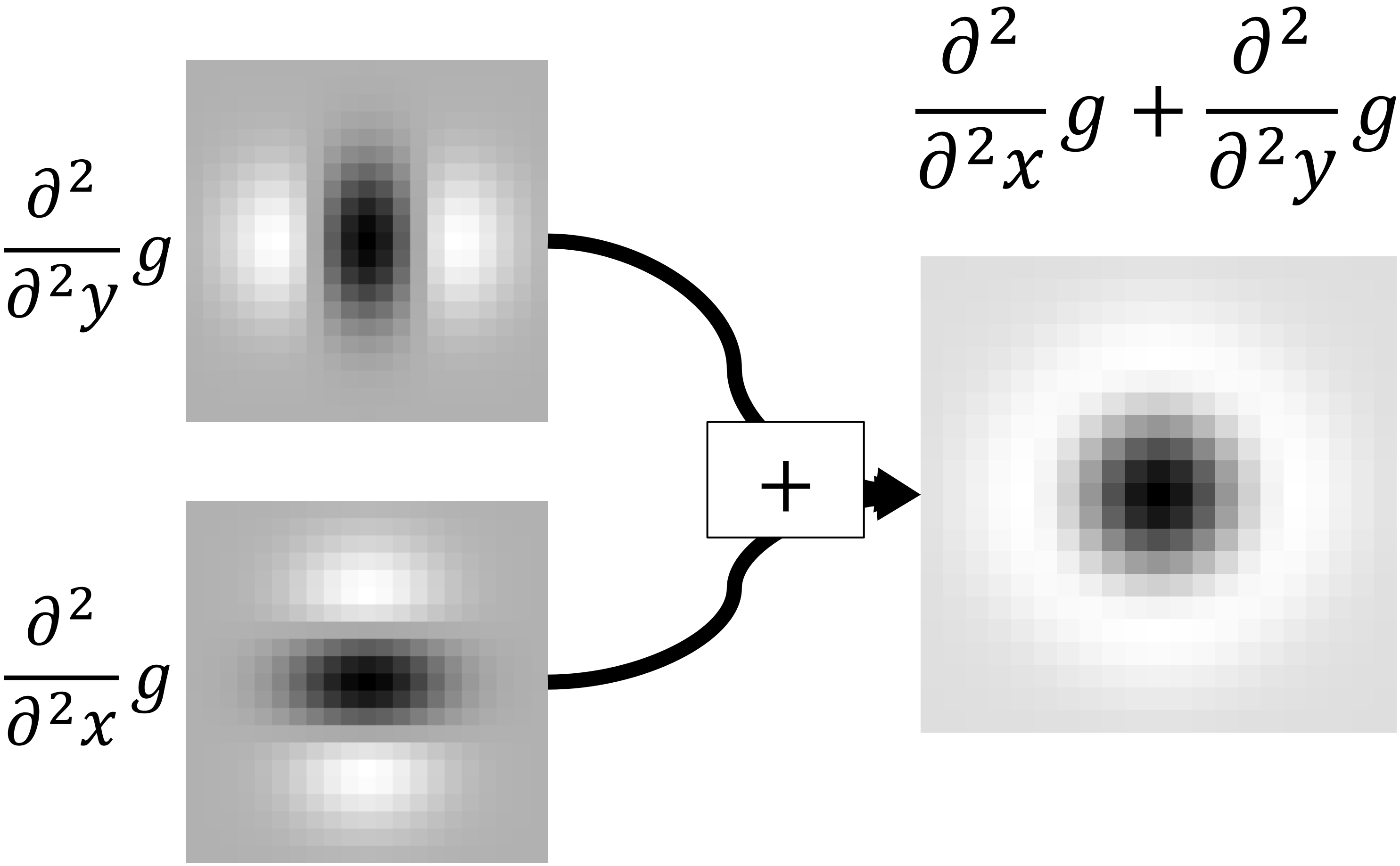
$$\frac{\partial^2}{\partial^2 y} g$$



$$\frac{\partial^2}{\partial^2 x} g$$



Laplacian of Gaussian (LoG)



Slight detail: for technical reasons, you need to scale the Laplacian of Gaussian if you want to compare across sigmas.

$$\nabla^2_{norm} = \sigma^2 \left(\frac{\partial^2}{\partial x^2} g + \frac{\partial^2}{\partial^2 y} g \right)$$

Edge Detection with LoG

f

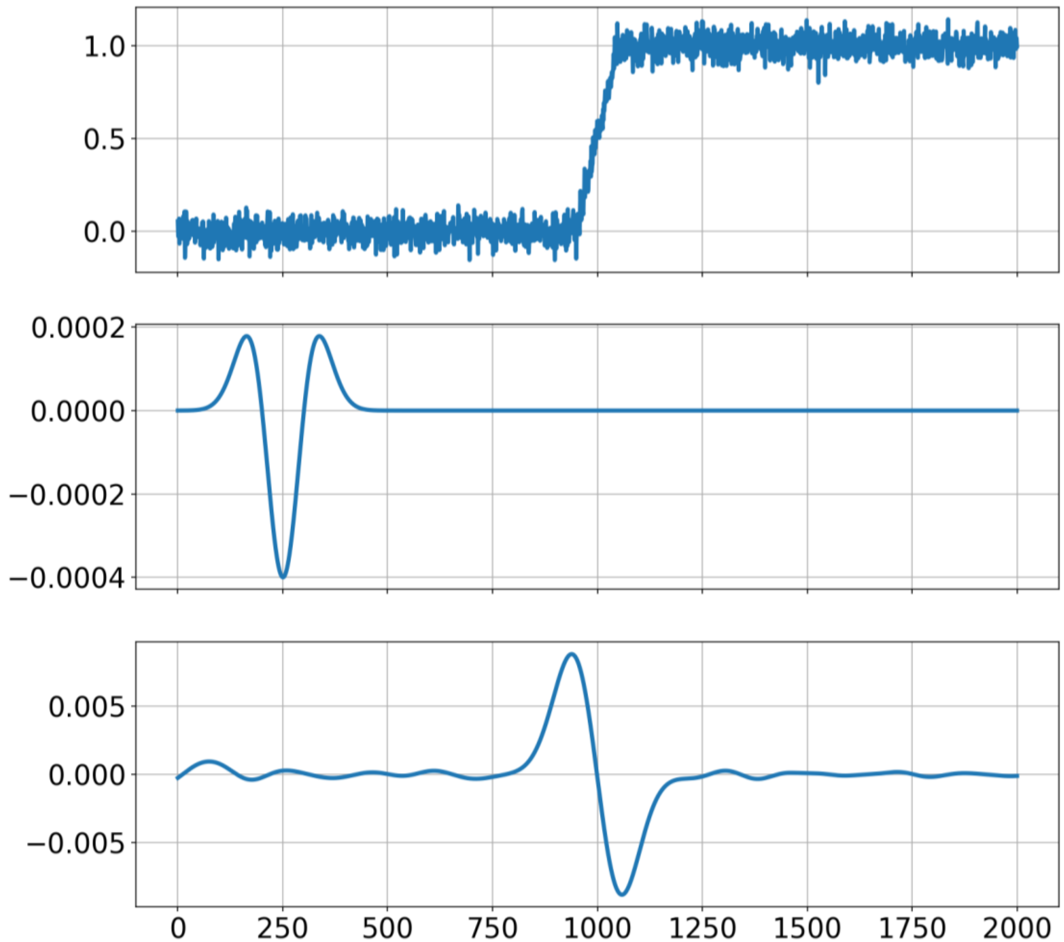
Edge

$$\frac{\partial^2}{\partial^2 x} g$$

Laplacian Of Gaussian (LoG)

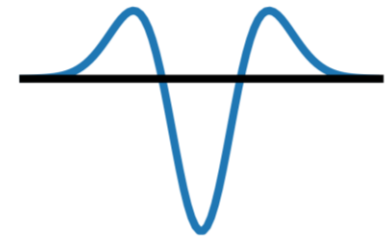
$$f * \frac{\partial^2}{\partial^2 x} g$$

Edge = Zero-crossing

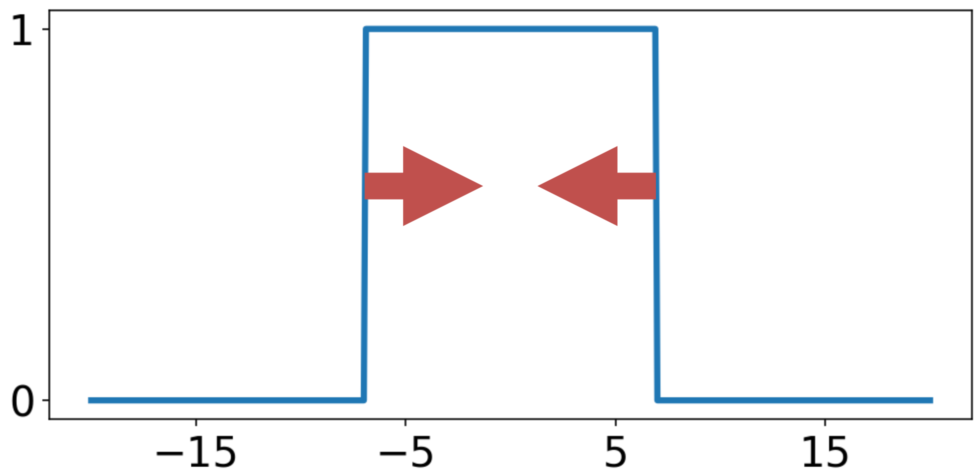


Modern remake of classic S. Seitz slide

Edge Detection with LoG

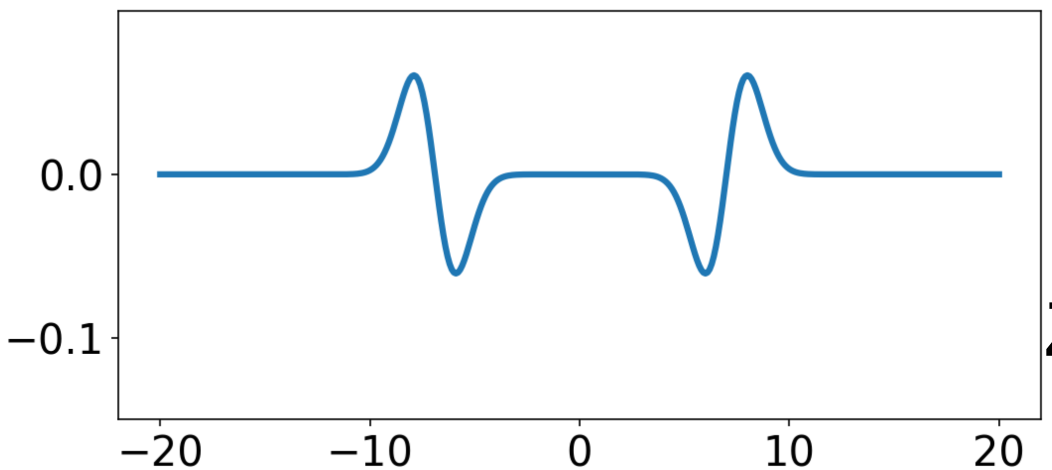


f



Edges

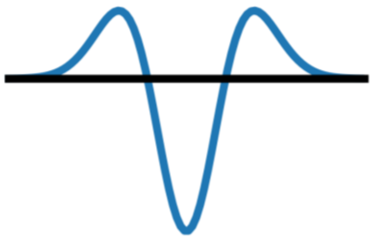
$$f * \frac{\partial^2}{\partial^2 x} g$$



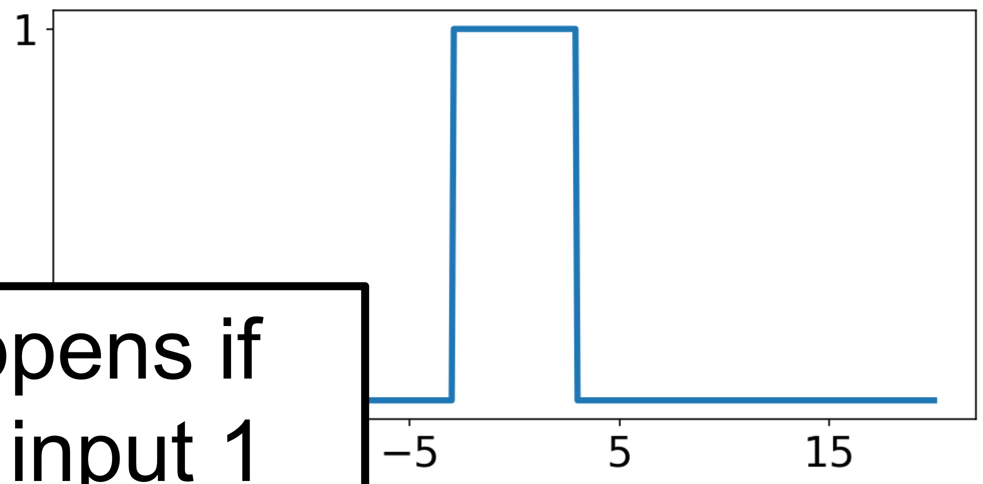
Edges * LoG = Zero-crossings

Modern remake of classic S. Seitz slide

Edge Detection with LoG



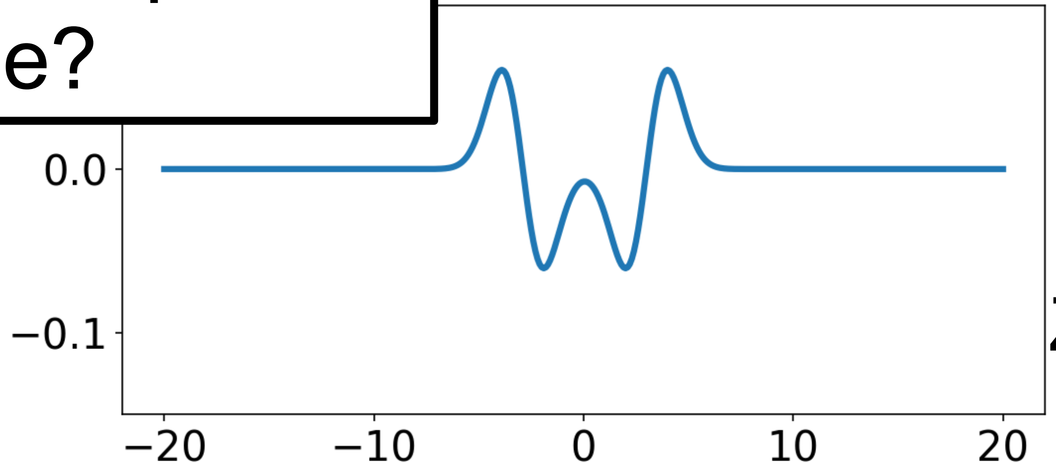
f



Edges

What happens if we make input 1 unit wide?

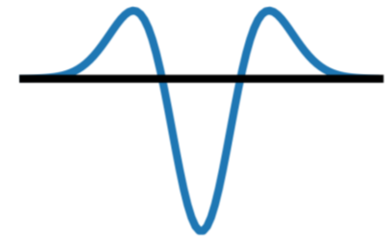
$$f * \frac{\partial^2}{\partial^2 x} g$$



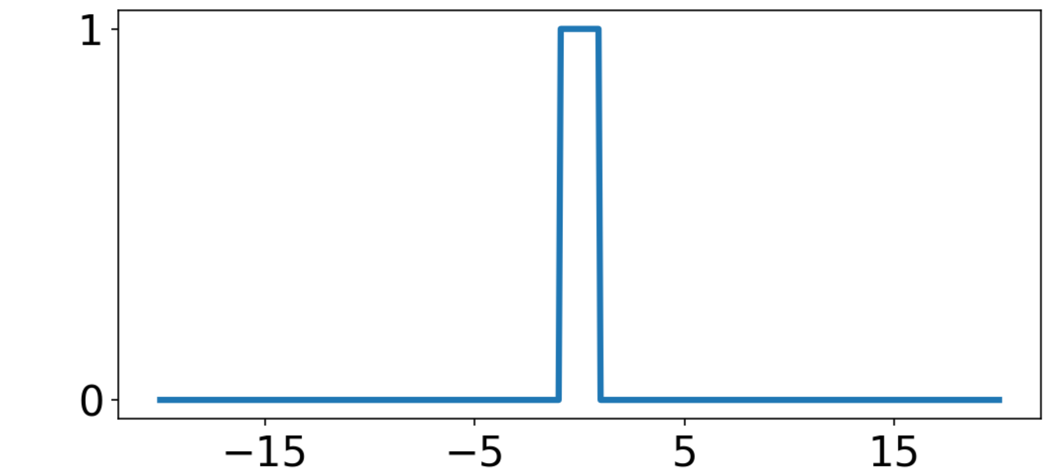
Edges * LoG = Zero-crossings

Modern remake of classic S. Seitz slide

Edge Detection with LoG

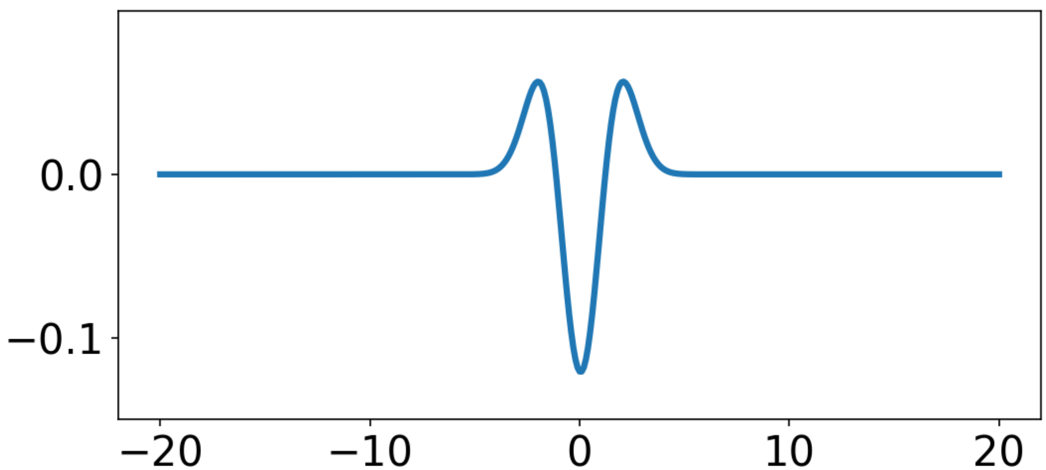


f



Edge

$$f * \frac{\partial^2}{\partial^2 x} g$$



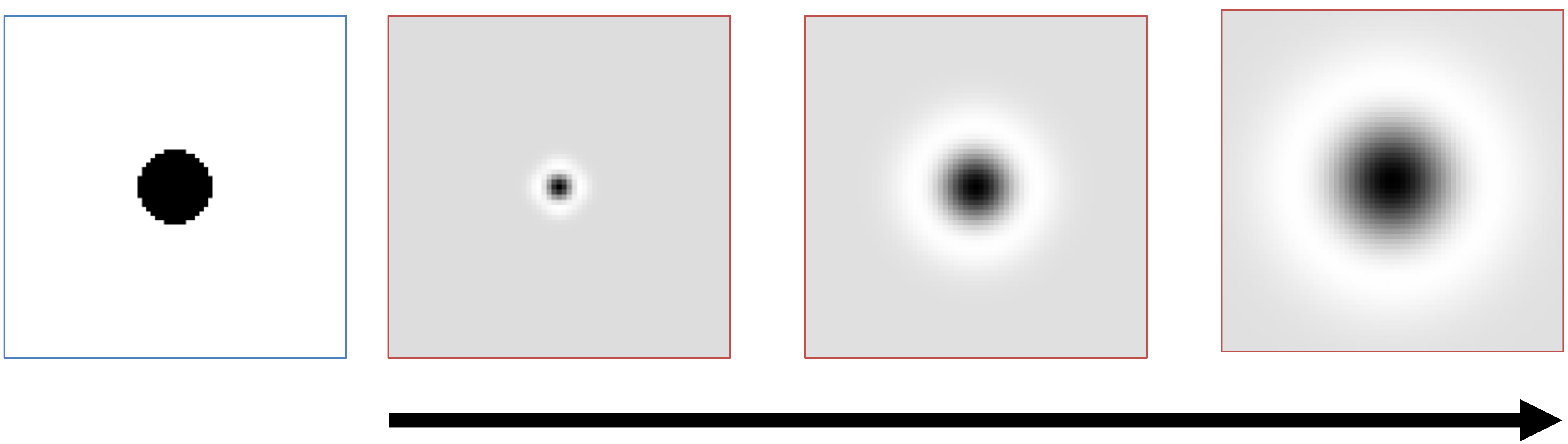
Edge * LoG = Zero-crossing

Modern remake of classic S. Seitz slide

Scale Selection

Given binary circle and Laplacian filter of scale σ, we can compute the response as a function of the scale.

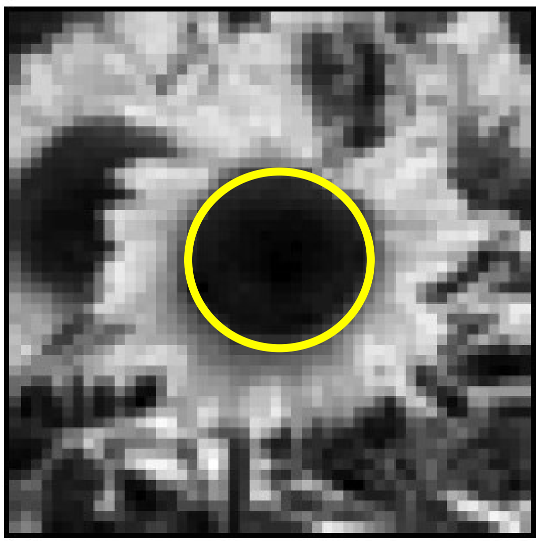
Image	$\sigma = 2$	$\sigma = 6$	$\sigma = 10$
Radius: 8	R: 0.02	R: 2.9	R: 1.8



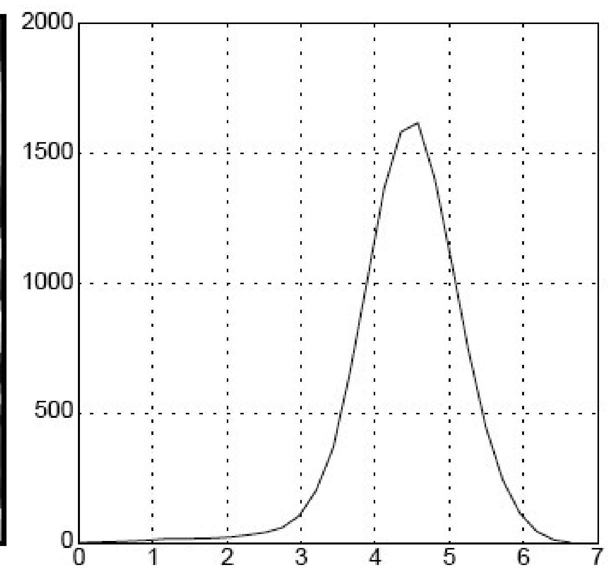
Characteristic Scale

Characteristic scale of a blob is the scale that produces the maximum response

Image



Abs. Response



Slide credit: S. Lazebnik. For more, see: T. Lindeberg (1998). "Feature detection with automatic scale selection." *International Journal of Computer Vision* **30** (2): pp 77--116.

Scale-space blob detector

1. Convolve image with scale-normalized Laplacian at several scales

Slide credit: S. Lazebnik

Scale-space blob detector: Example

Slide credit: S. Lazebnik

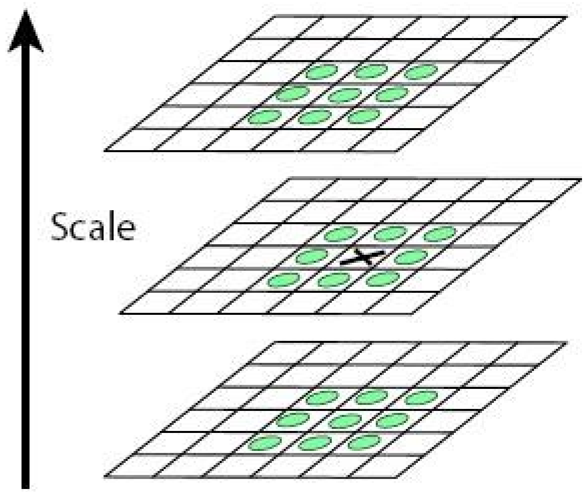
Scale-space blob detector: Example

sigma = 11.9912

Slide credit: S. Lazebnik

Scale-space blob detector

1. Convolve image with scale-normalized Laplacian at several scales
2. Find maxima of squared Laplacian response in scale-space



Slide credit: S. Lazebnik

Finding Maxima

Point i,j is maxima (minima if you flip sign) in image I if it's bigger than all neighbors

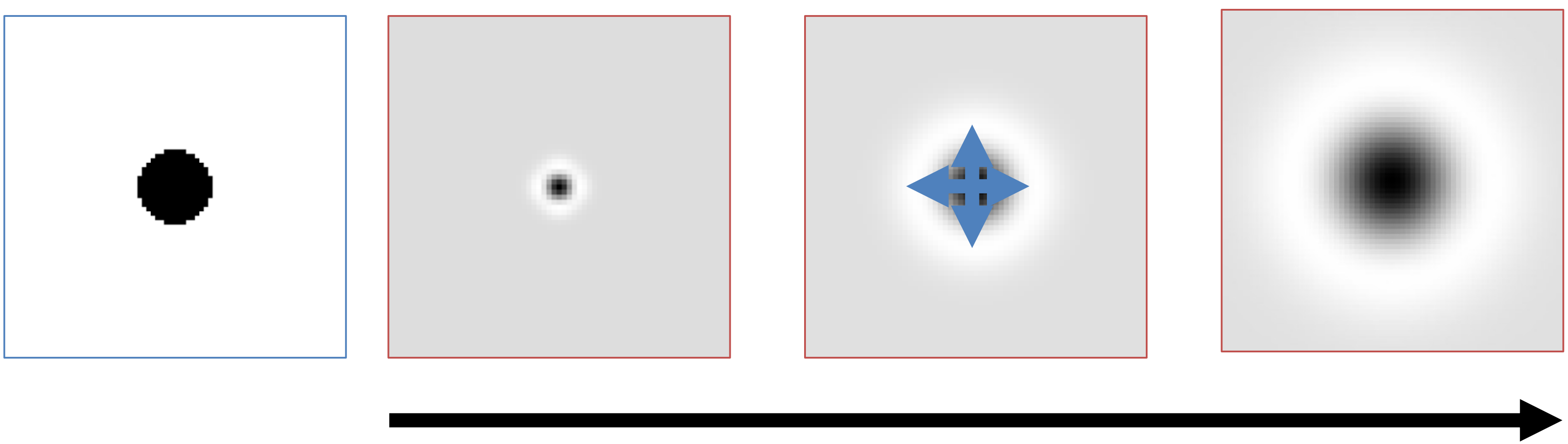
```
for y=range(i-1,i+1+1):
        for x in range(j-1,j+1+1):
                if y == i and x== j: continue
                #below has to be true
                I[y,x] < I[i,j]
```

Scale Space

Blue lines are image-space neighbors (should be just one pixel over but that's impossible to draw)

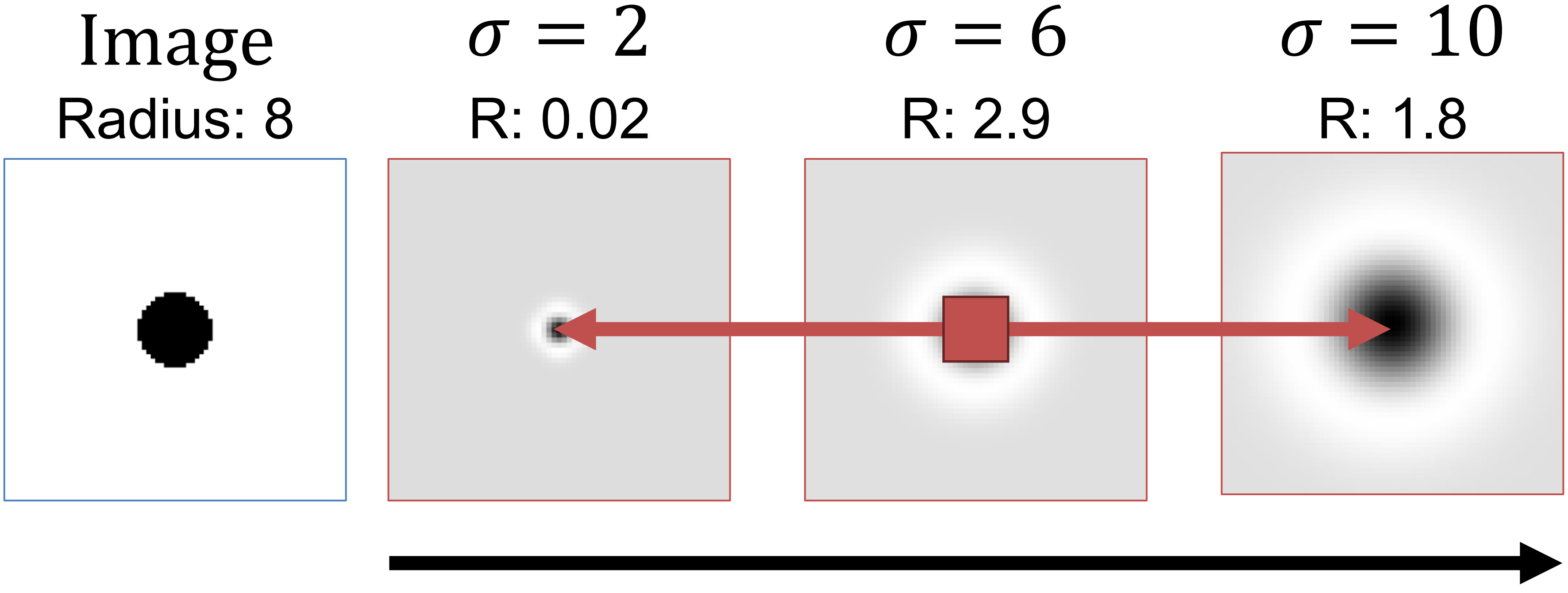
Image | $\sigma = 2$ | $\sigma = 6$ | $\sigma = 10$

Radius: 8 | R: 0.02 | R: 2.9 | R: 1.8



Scale Space

Red lines are the scale-space neighbors

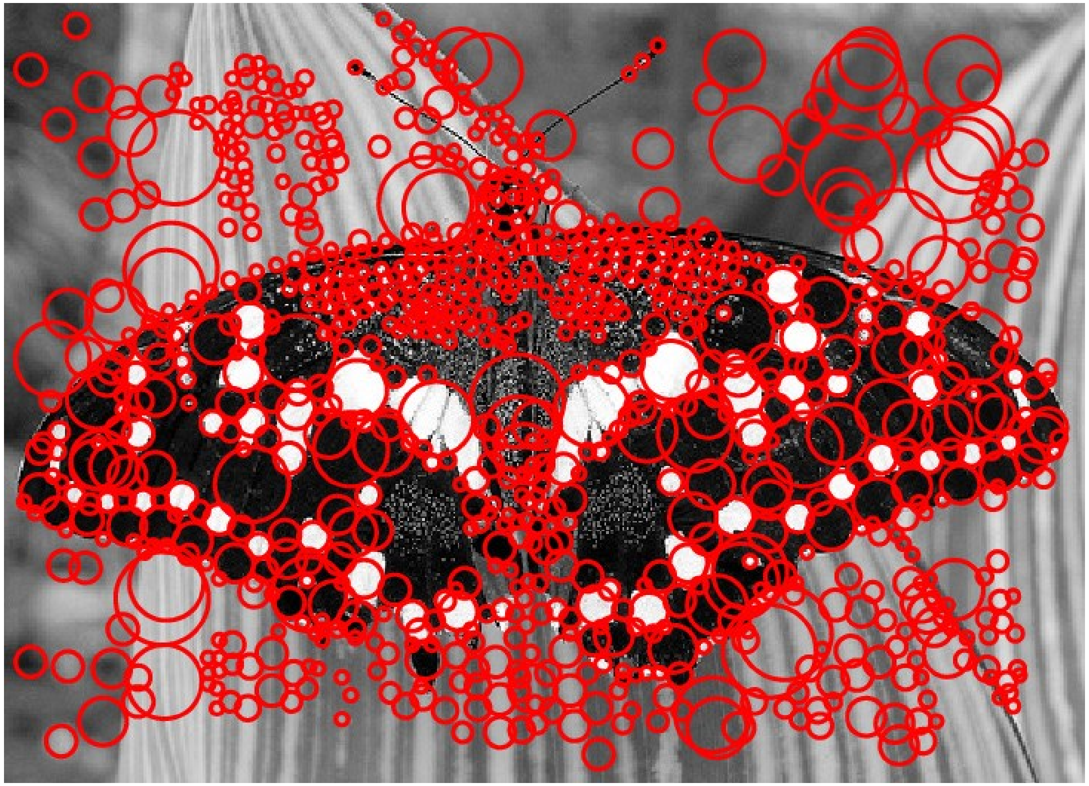


Finding Maxima

Suppose I[:,:,k] is image at scale k. Point i,j,k is maxima (minima if you flip sign) in image I if:

```
for y=range(i-1,i+1+1):
        for x in range(j-1,j+1+1):
                for c in range(k-1,k+1+1):
                        if y == i and x== j and c==k:
                                continue
                #below has to be true
                I[y,x,c] < I[i,j,k]
```

Scale-space blob detector: Example



Slide credit: S. Lazebnik

Efficient implementation

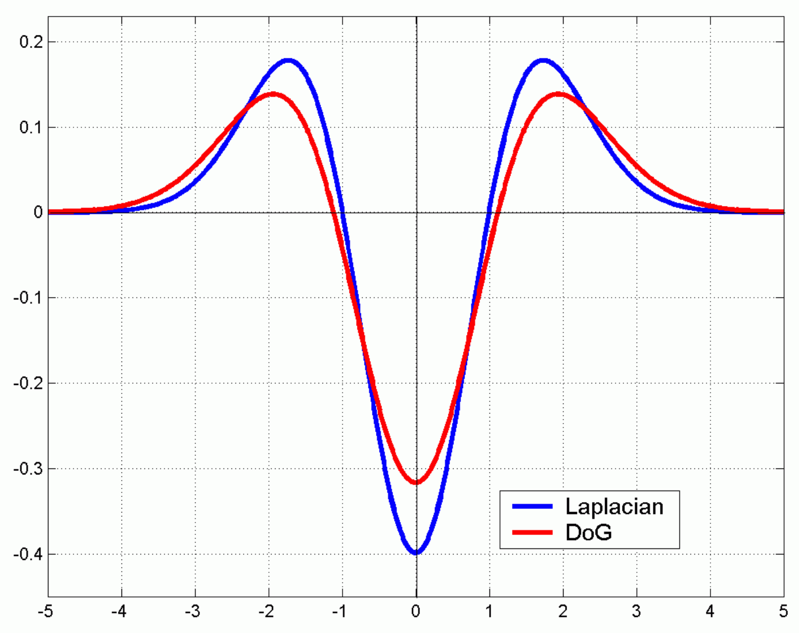
- Approximating the Laplacian with a difference of Gaussians:

$$L = \sigma^2 \left(G_{xx}(x, y, \sigma) + G_{yy}(x, y, \sigma) \right)$$

(Laplacian)

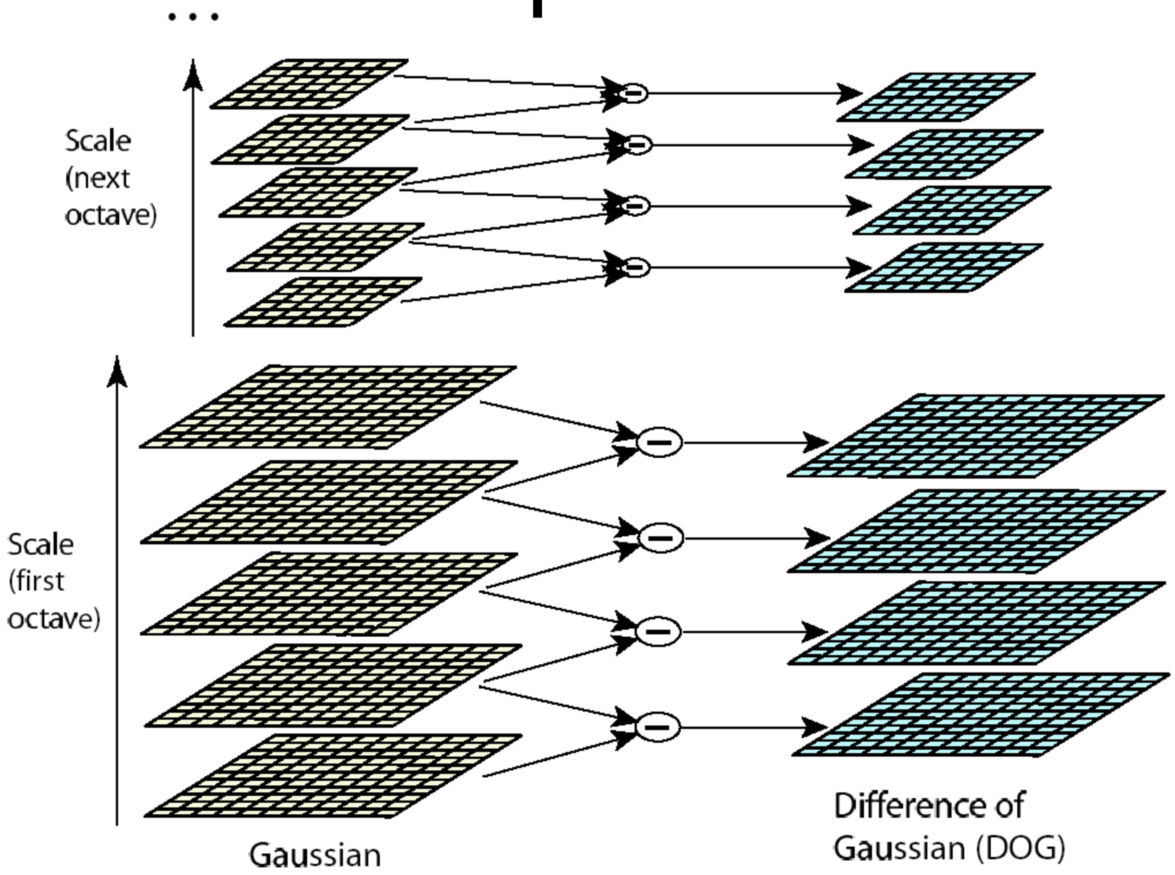
$$DoG = G(x, y, k\sigma) - G(x, y, \sigma)$$

(Difference of Gaussians)



Slide credit: S. Lazebnik

Efficient implementation



David G. Lowe. **["Distinctive image features from scale-invariant keypoints."]** *IJCV* 60 (2), pp. 91-110, 2004.

Slide credit: S. Lazebnik

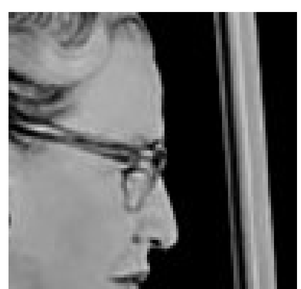
Problem 1 Solved

- How do we deal with scales: try them all
- **Why is this efficient?**

Vast majority of effort is in the first and second scales

$$1 + \frac{1}{4} + \frac{1}{16} + \frac{1}{64} + \frac{1}{4^i} \ldots = \frac{4}{3}$$

Problem 2 – Describing Features

	Image – 40	1/2 size, rot. 45° Lightened+40
Full Image		
100x100 crop at Glasses		

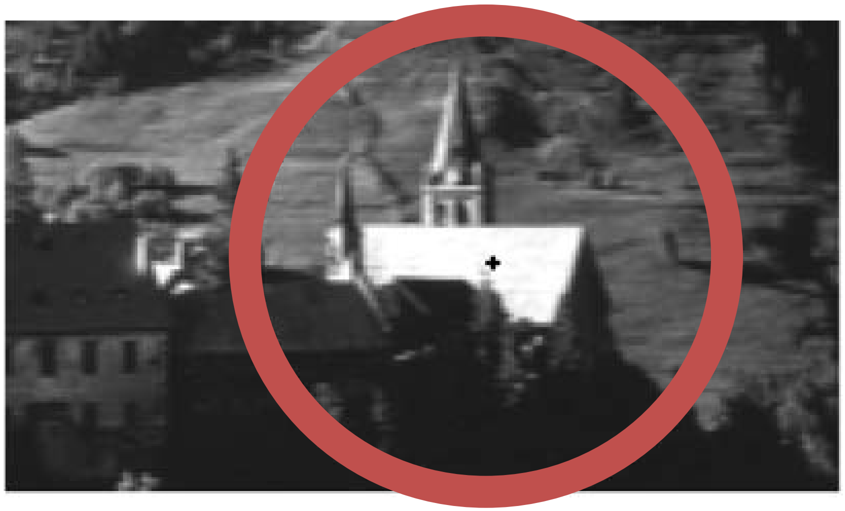
Problem 2 – Describing Features

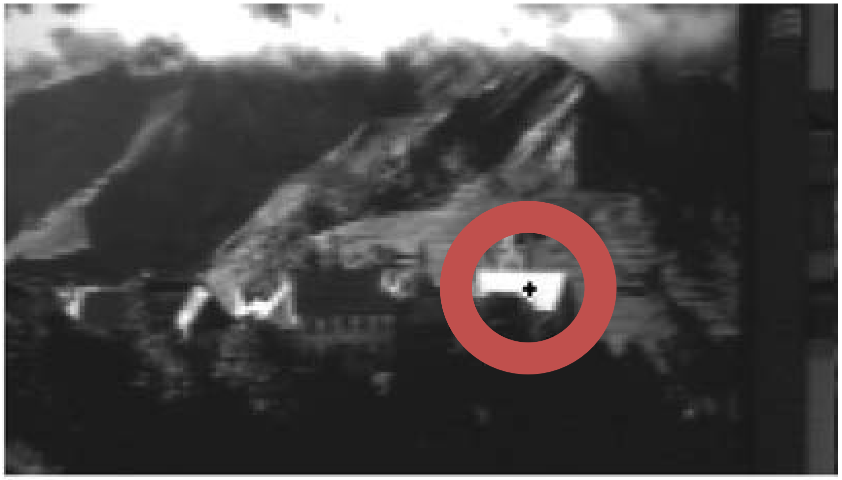
Once we've found a corner/blobs, we can't just use the image nearby. What about:

1. Scale?
2. Rotation?
3. Additive light?

Handling Scale

Given characteristic scale (maximum Laplacian response), we can just rescale image

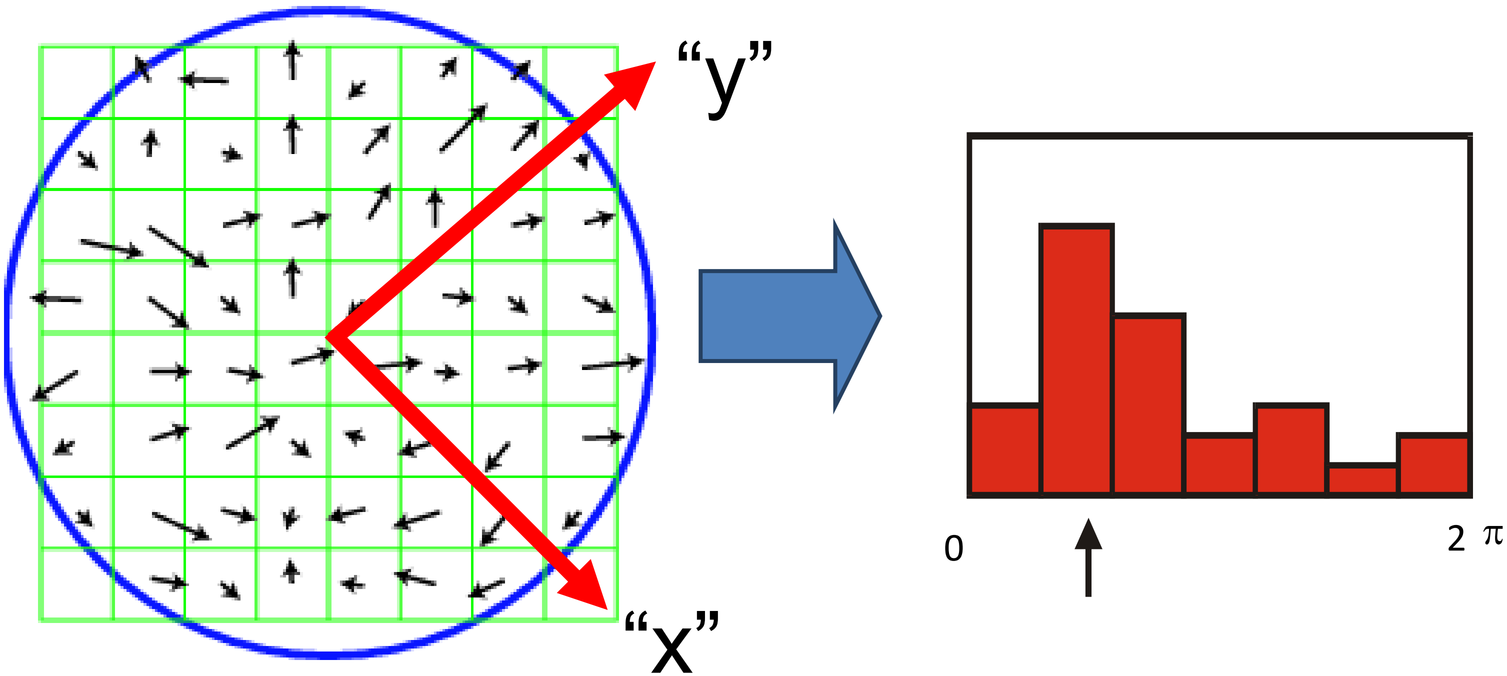




Slide credit: S. Lazebnik

Handling Rotation

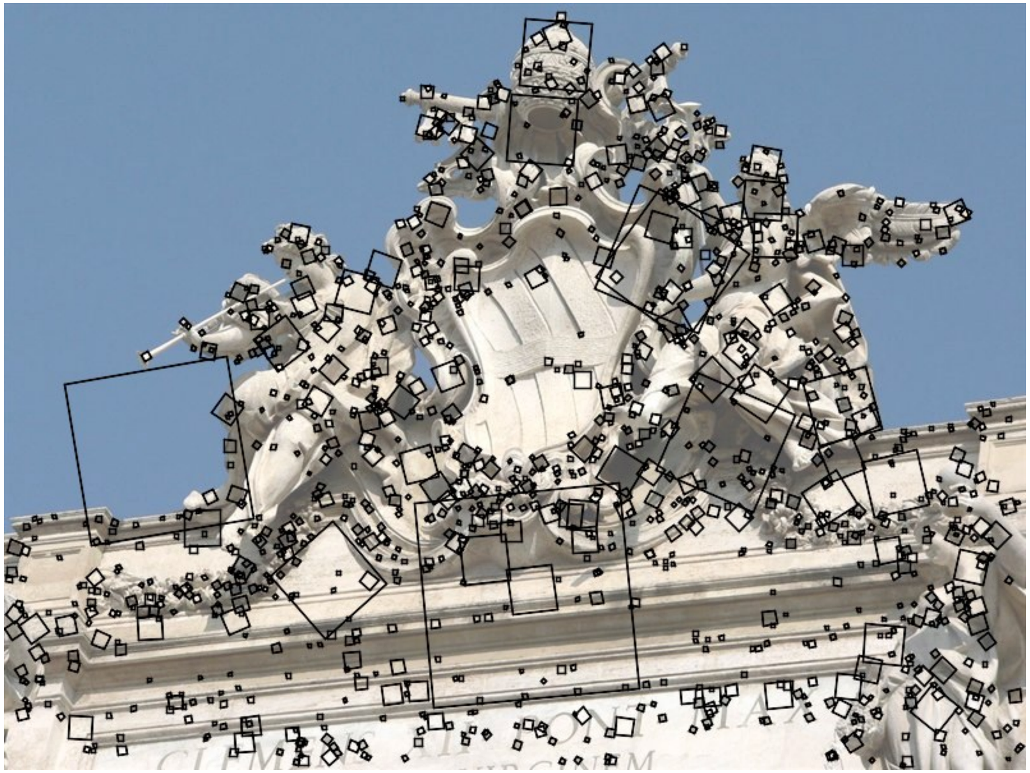
Given window, can compute “dominant orientation” and then rotate image



Slide credit: S. Lazebnik

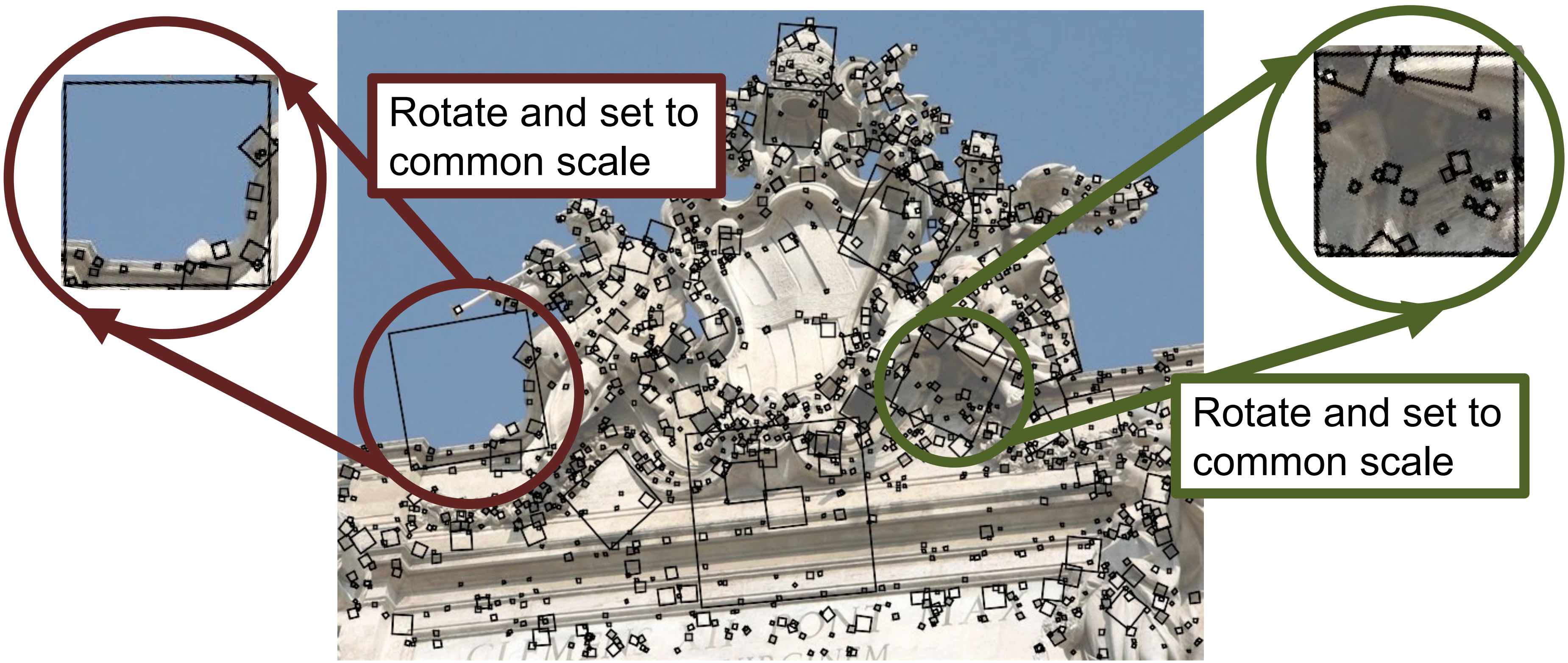
Scale and Rotation

SIFT features at characteristic scales and dominant orientations



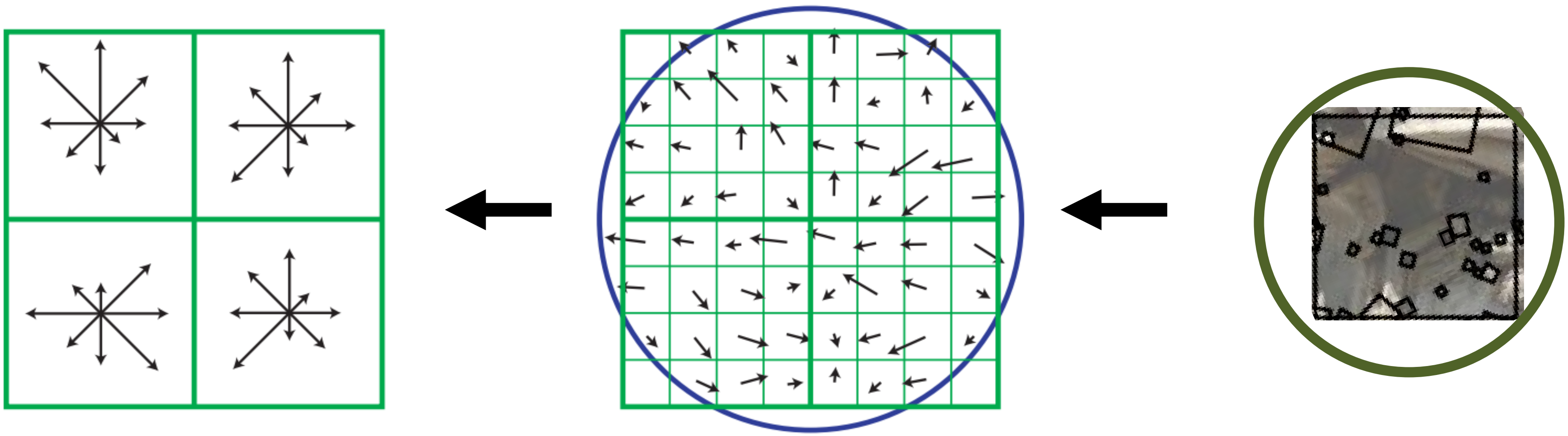
Picture credit: S. Lazebnik. Paper: David G. Lowe. **"Distinctive image features from scale-invariant keypoints."** *IJCV* 60 (2), pp. 91-110, 2004.

Scale and Rotation



Picture credit: S. Lazebnik. Paper: David G. Lowe. **"Distinctive image features from scale-invariant keypoints."** *IJCV* 60 (2), pp. 91-110, 2004.

SIFT Descriptors



1. Compute gradients
2. Build histogram (2x2 here, 4x4 in practice)

Gradients ignore global illumination changes

Figure from David G. Lowe. **"Distinctive image features from scale-invariant keypoints."** *IJCV* 60 (2), pp. 91-110, 2004.

SIFT Descriptors

- In principle: build a histogram of the gradients
- In reality: quite complicated
  - Gaussian weighting: smooth response
  - Normalization: reduces illumination effects
  - Clamping
  - Tons of more stuff

Properties of SIFT

- Can handle: up to ~60 degree out-of-plane rotation, changes of illumination
- Fast, efficient, code available (but was patented)

Slide credit: N. Snavely

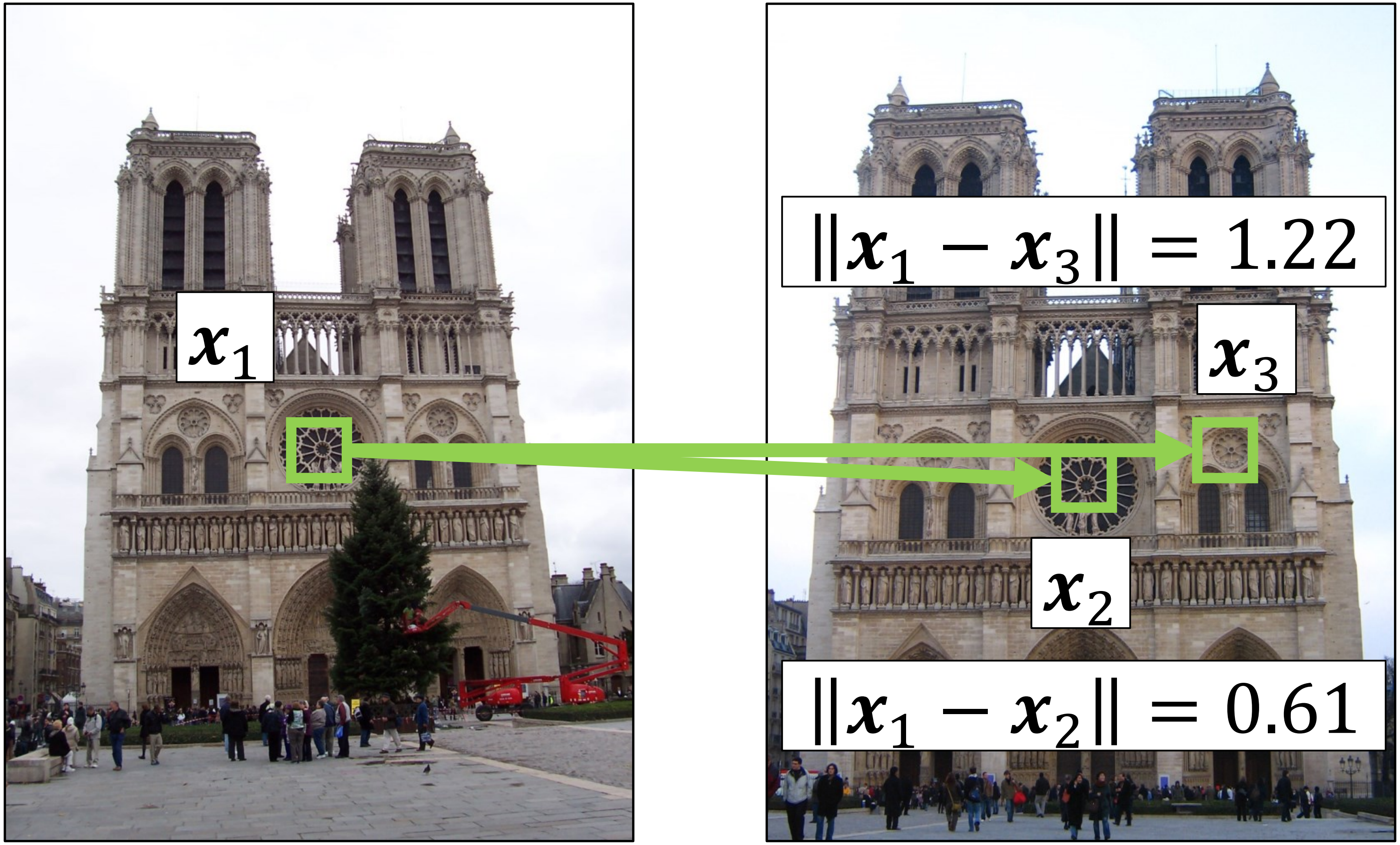
Feature Descriptors

Think of feature as some non-linear filter that maps pixels to 128D feature

128D vector **x**

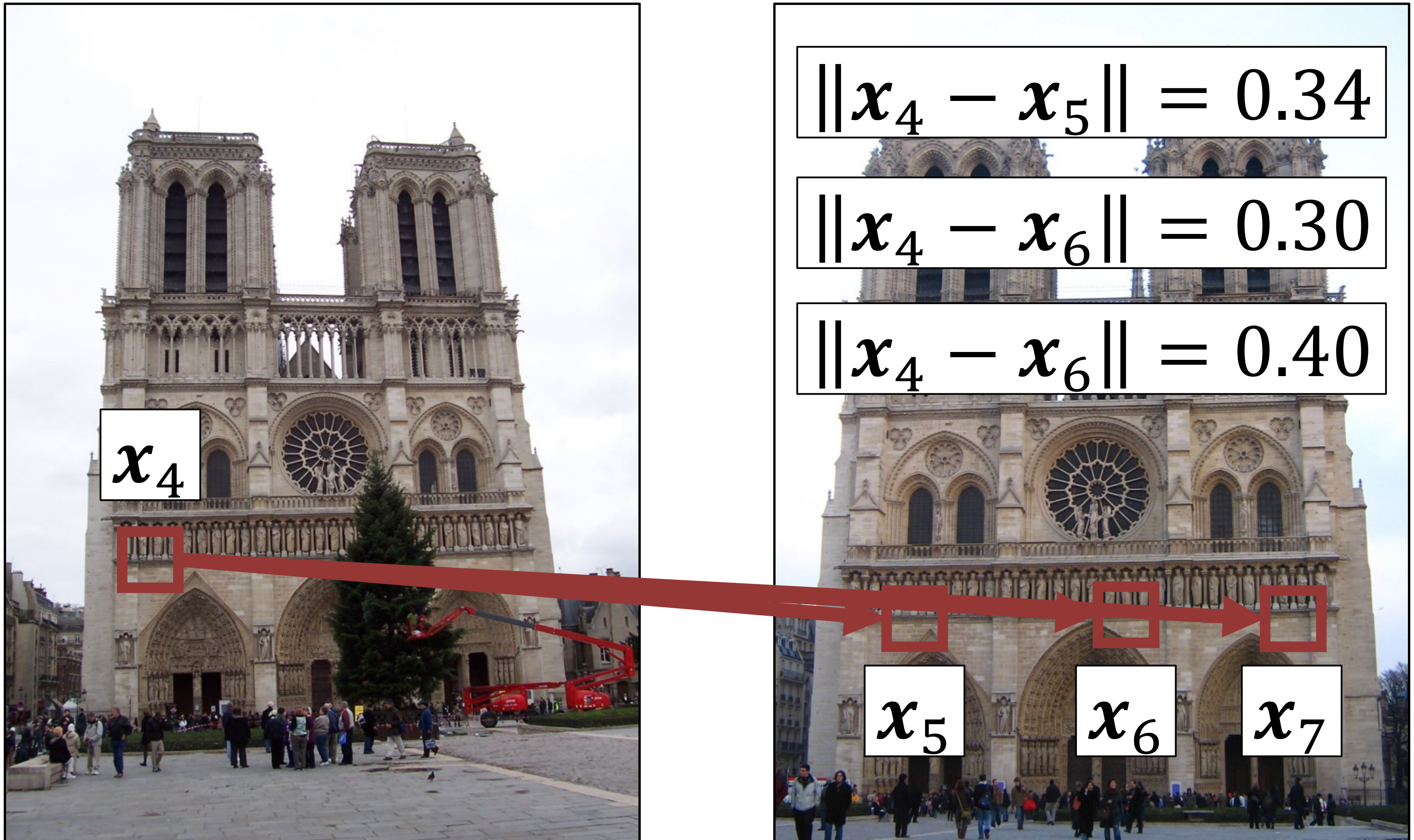
Photo credit: N. Snavely

Instance Matching



Example credit: J. Hays

Instance Matching



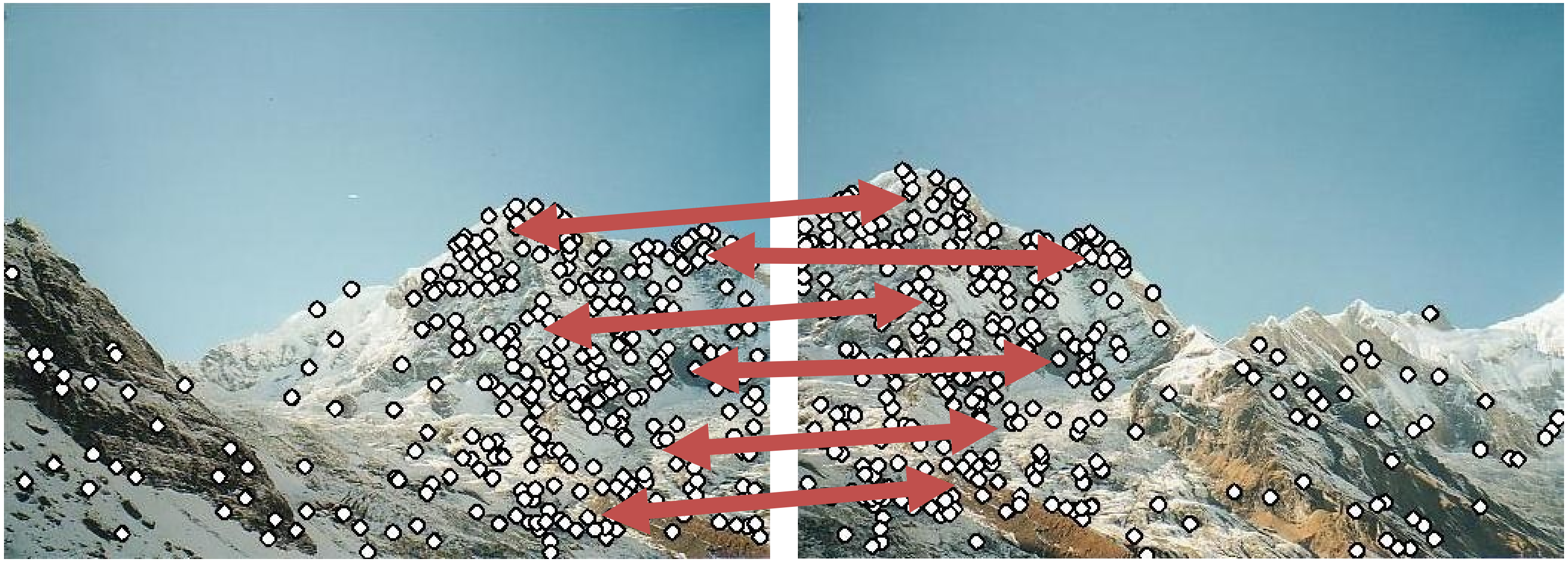
Example credit: J. Hays

2^{nd} Nearest Neighbor Trick

- Given a feature x_q, nearest neighbor to x is a good match, but distances can't be thresholded.
- Instead, find nearest neighbor (x_{1NN}) and second nearest neighbor (x_{2NN}). This ratio is a good test for matches:

$$r = \frac{\left\| \boldsymbol{x}_q - \boldsymbol{x}_{1NN} \right\|}{\left\| \boldsymbol{x}_q - \boldsymbol{x}_{2NN} \right\|}$$

So Far; What's Next?



1: find corners+features

2: match based on local image data

3: next time: compute offsets from matches

Extra Reading for the Curious

Aside: This Trick is Common

Given a 50x16 person detector, how do I detect:
(a) 250x80 (b) 150x48 (c) 100x32 (d) 25x8 people?

Sample people from image

Aside: This Trick is Common

Detecting all the people
The red box is a fixed size

Sample people from image

Aside: This Trick is Common

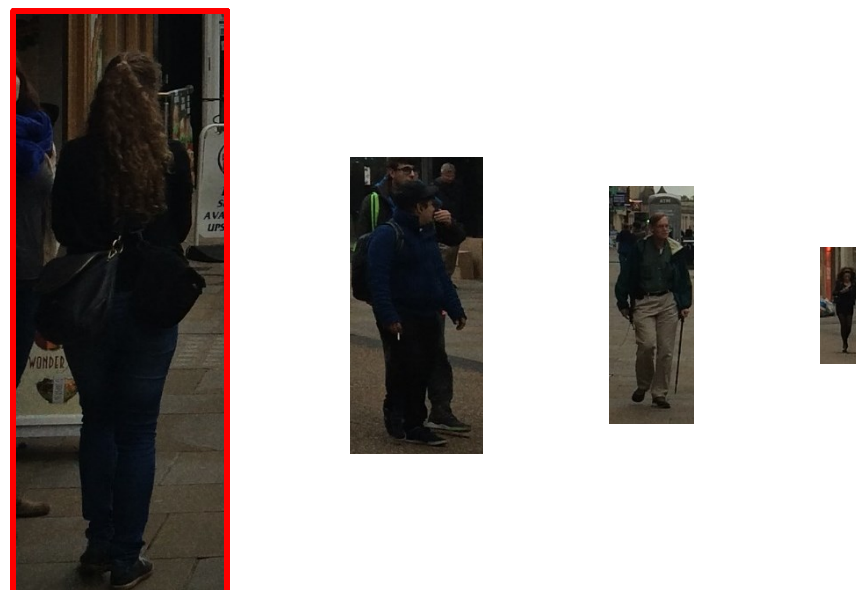
Detecting all the people
The red box is a fixed size

Sample people from image

Aside: This Trick is Common

Detecting all the people
The red box is a fixed size

Sample people from image



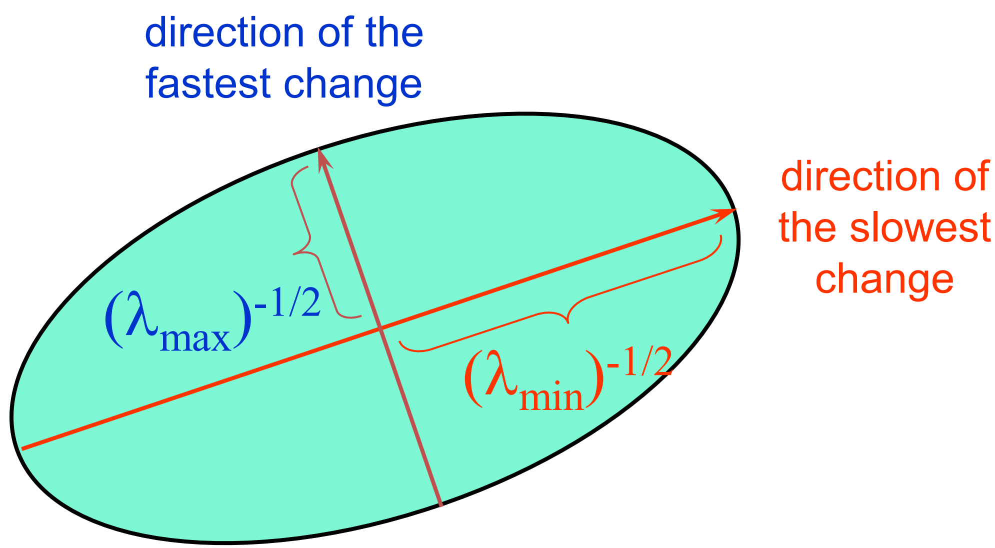
Affine adaptation

Consider the second moment matrix of the window containing the blob:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} = R^{-1} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

Recall:

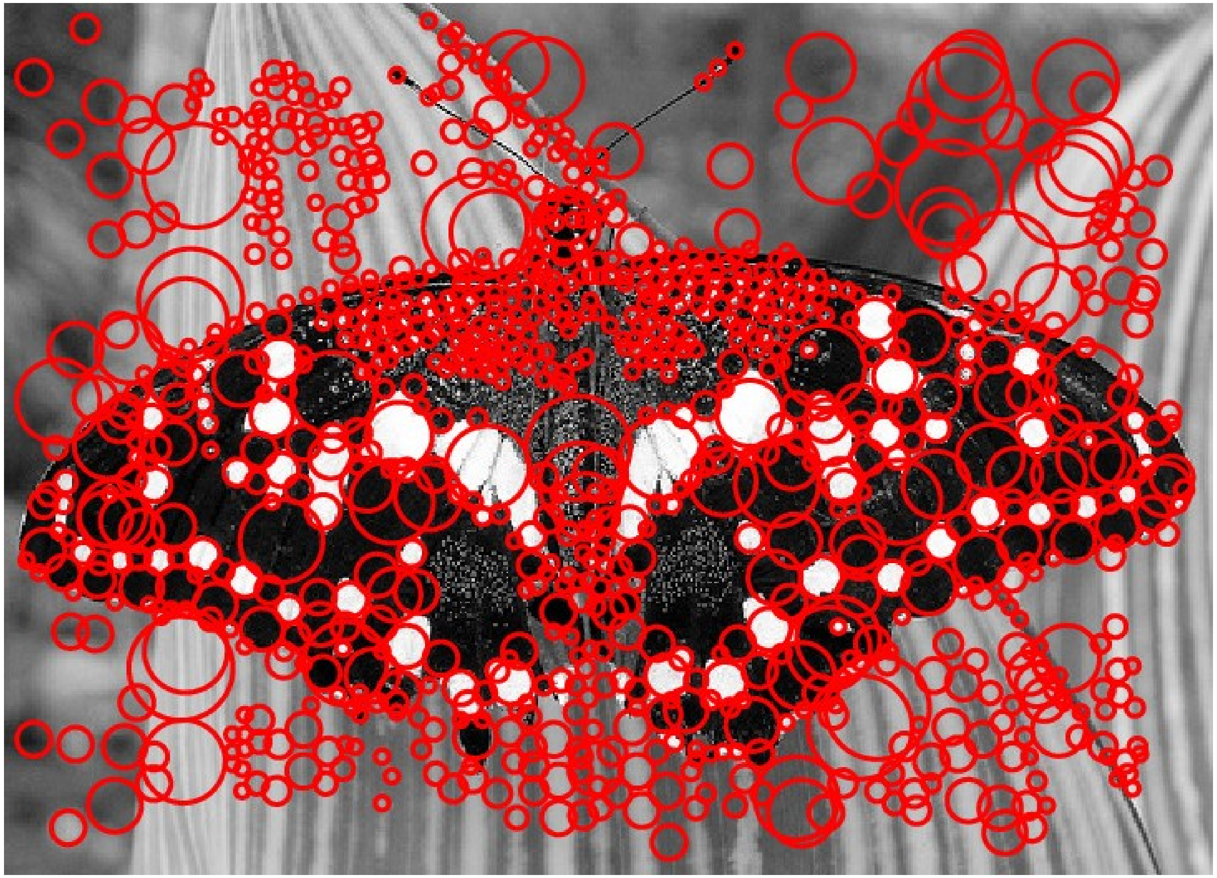
$$[u \;\; v] \; M \begin{bmatrix} u \\ v \end{bmatrix} = \text{const}$$



This ellipse visualizes the "characteristic shape" of the window

Slide: S. Lazebnik

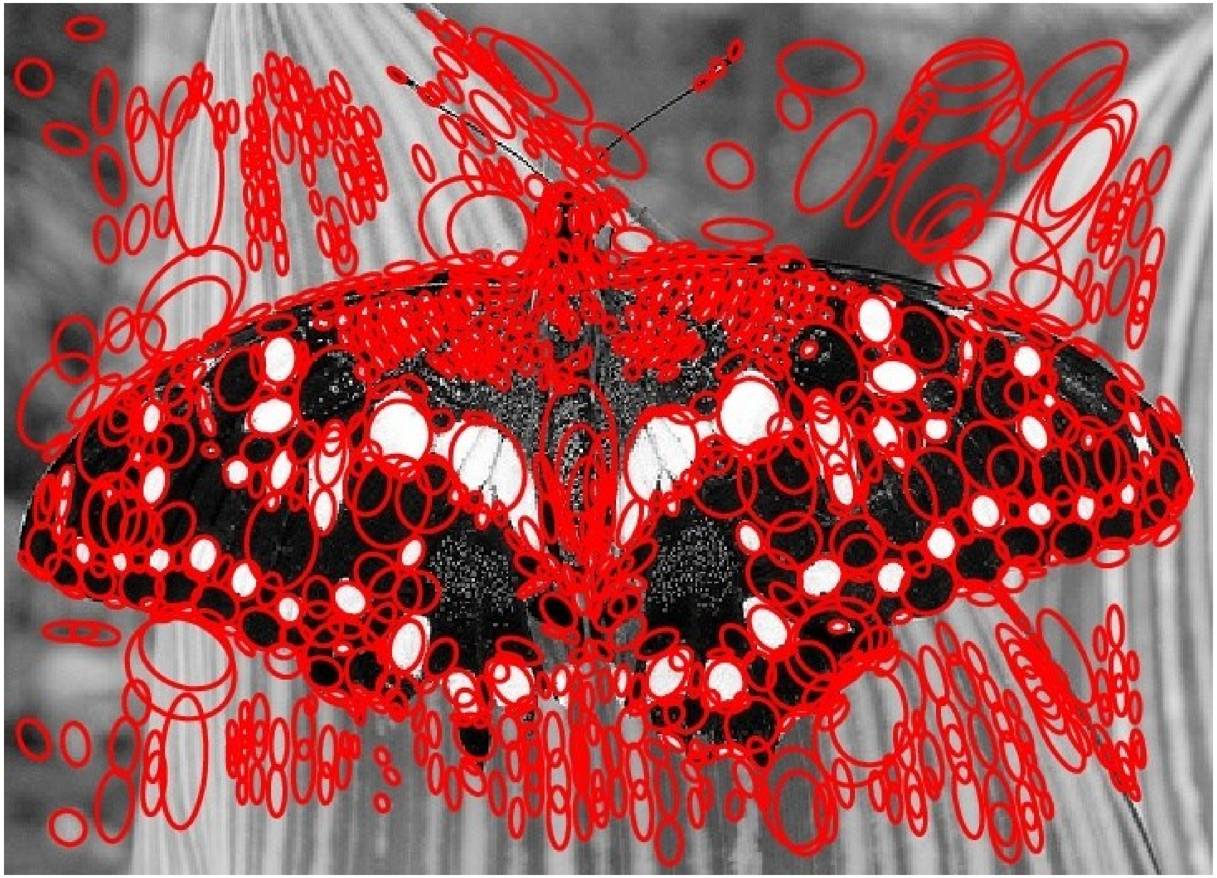
Affine adaptation example



Scale-invariant regions (blobs)

Slide: S. Lazebnik

Affine adaptation example



Affine-adapted blobs

Slide: S. Lazebnik

2^{nd} Nearest Neighbor Trick

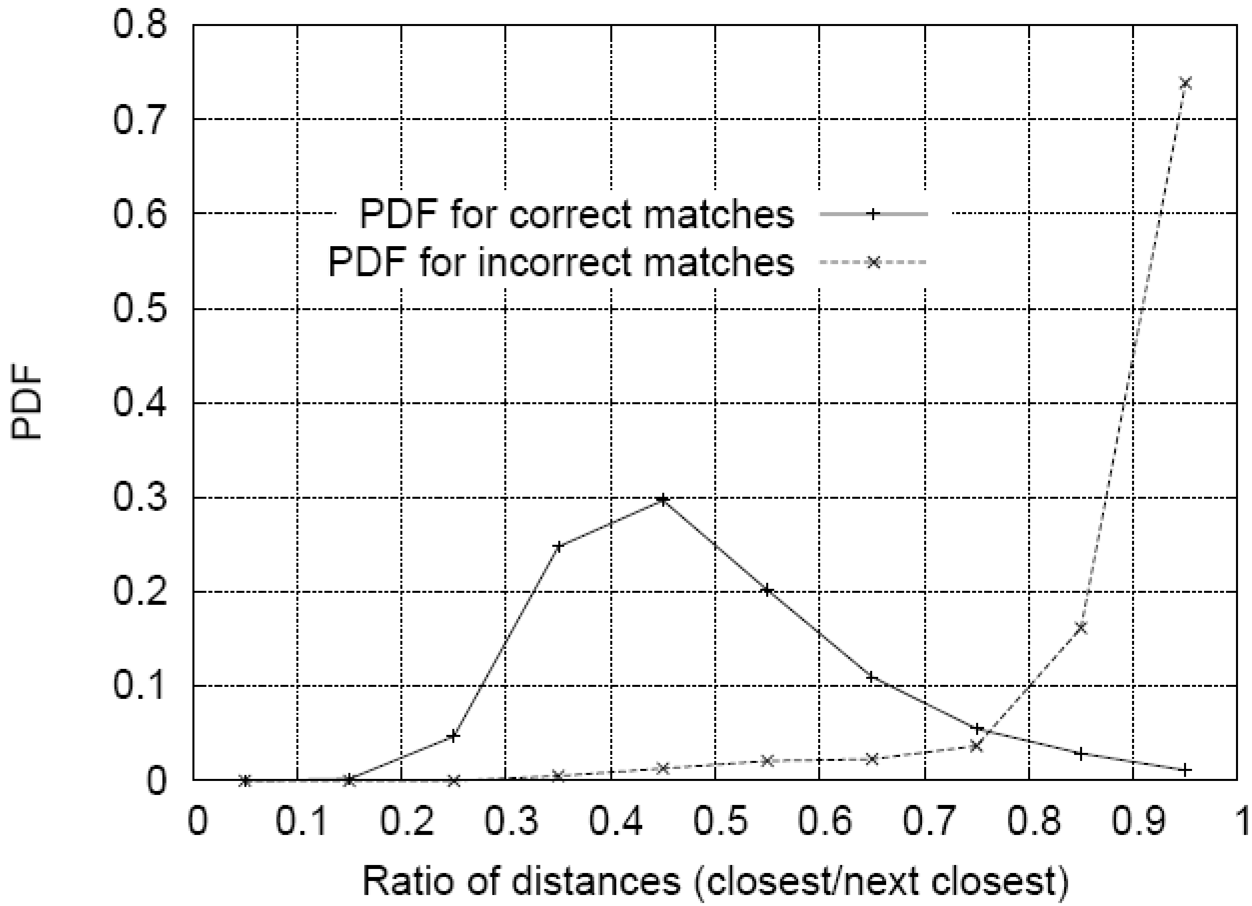


Figure from David G. Lowe. **["Distinctive image features from scale-invariant keypoints."]()** *IJCV* 60 (2), pp. 91-110, 2004.