Edges, Lines

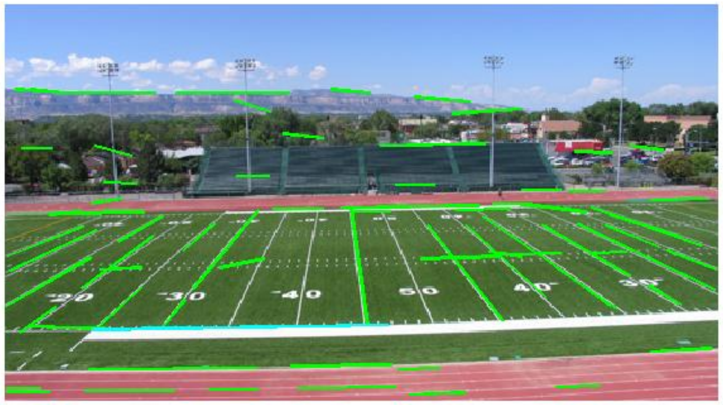
CS 600.361/600.461

Instructor: Greg Hager

Adapted from slides by N. Padoy, T. Darrell, K. Grauman, Fei-Fei Li and others

Outline

- Edge detection
- Line detection



Edges

Edge detection

- **Goal**: map image from 2d array of pixels to a set of curves or line segments or contours.
- **Why?**

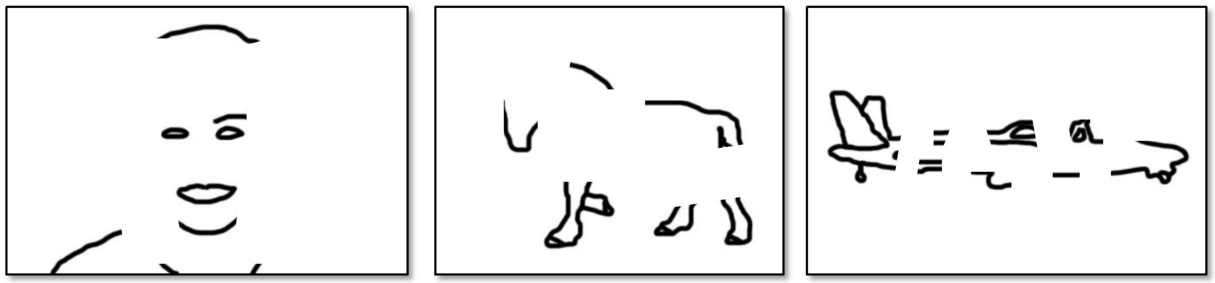


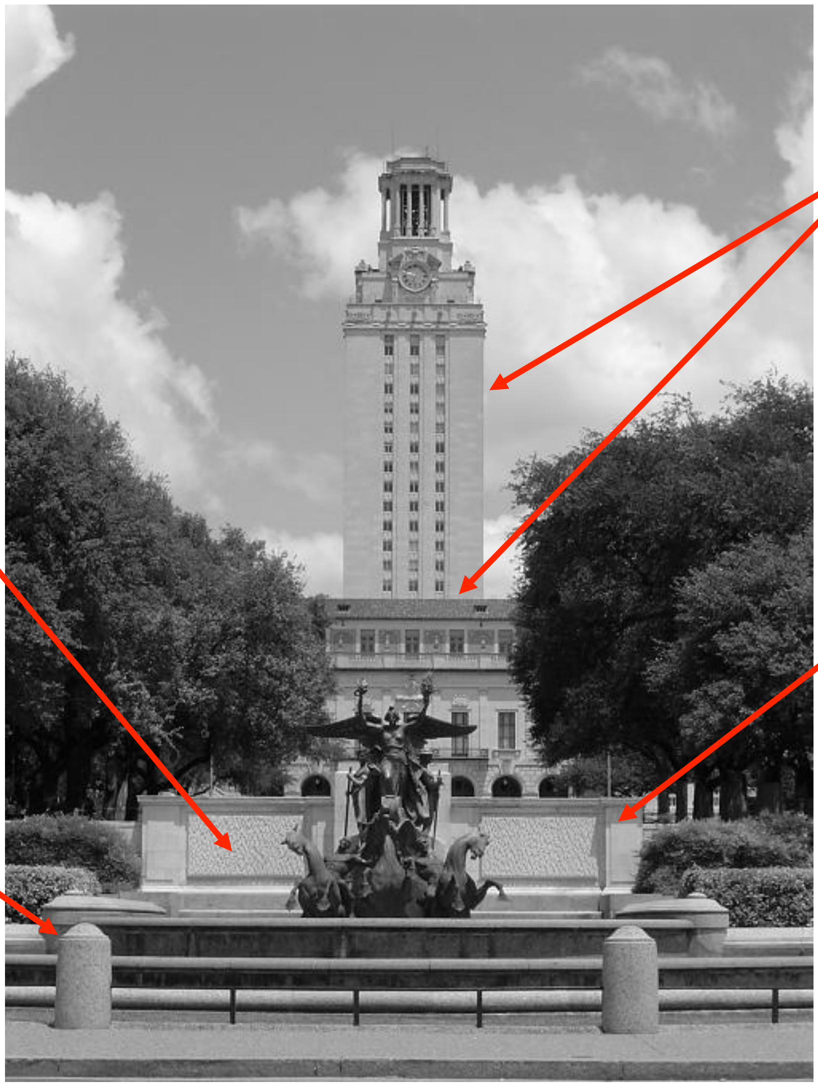
Figure from J. Shotton et al., PAMI 2007

- **Main idea**: look for strong gradients, post-process

What can cause an edge ?

Reflectance change: appearance information, texture

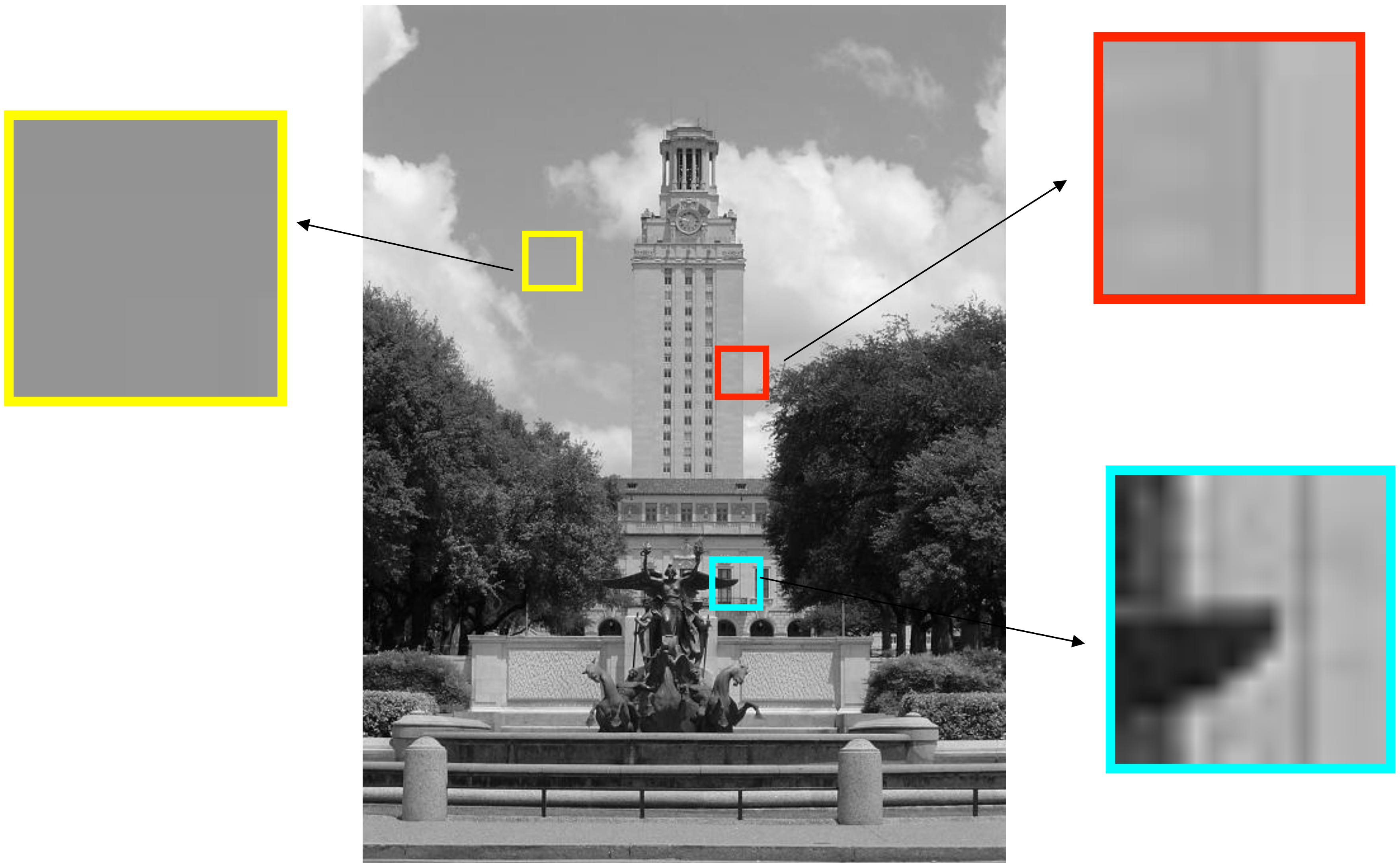
Change in surface orientation: shape



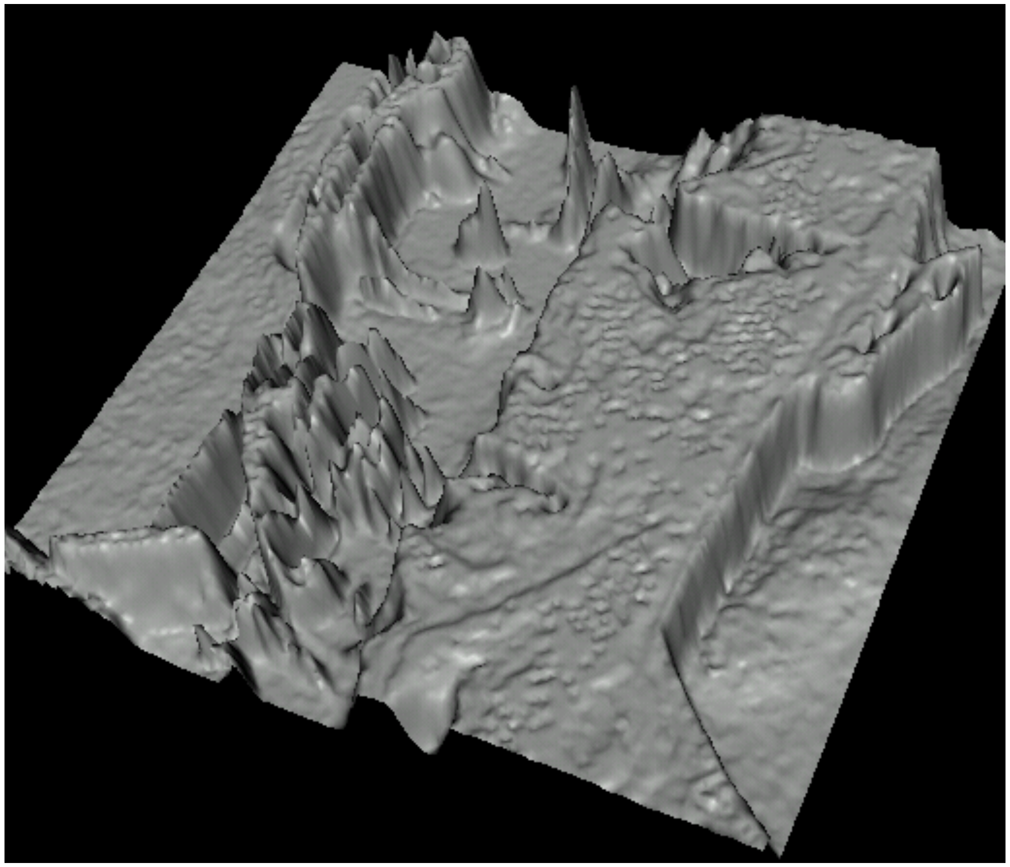
Depth discontinuity: object boundary

Cast shadows

Contrast and invariance



Recall : Images as functions

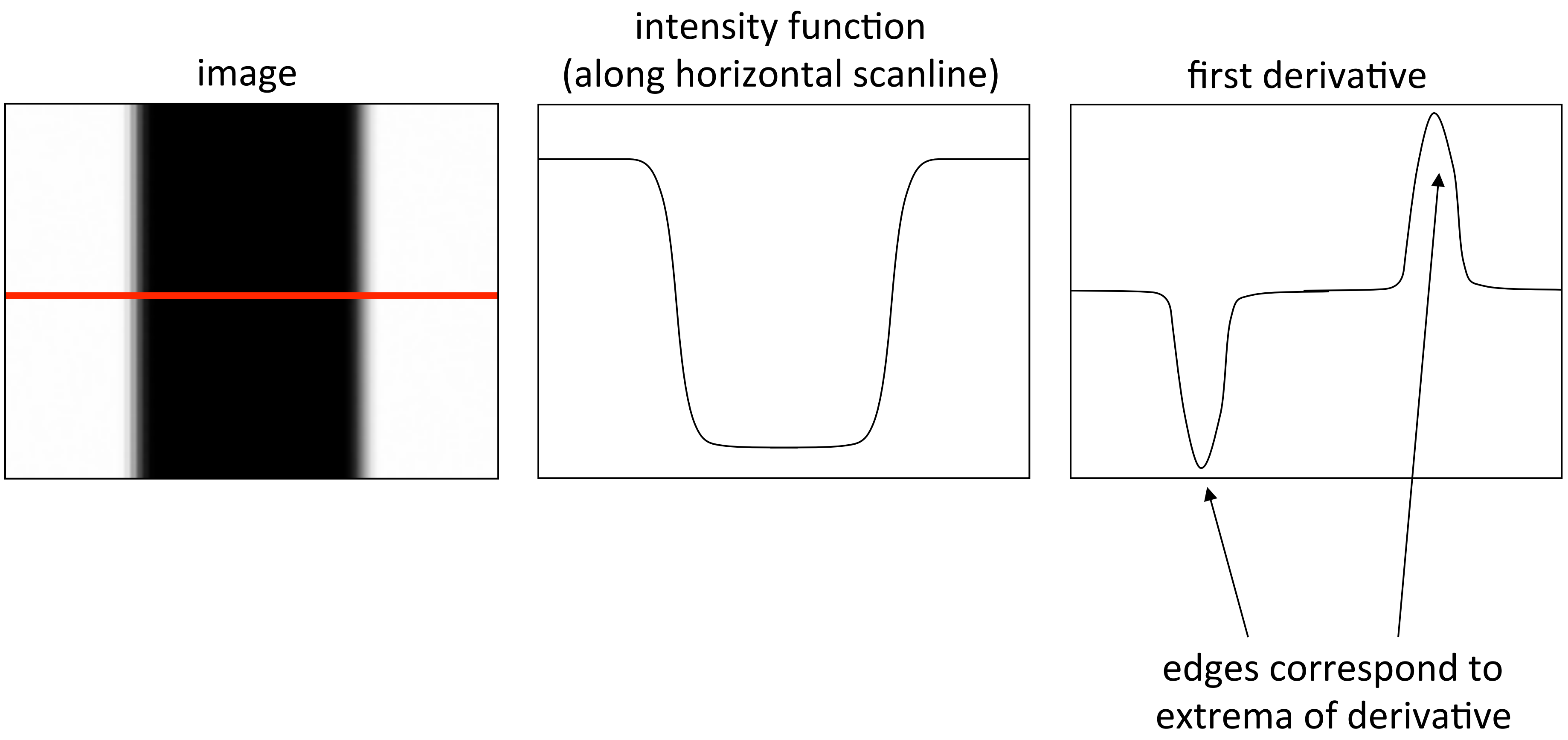


- Edges look like steep cliffs

Source: S. Seitz

Derivatives and edges

An edge is a place of rapid change in the image intensity function.

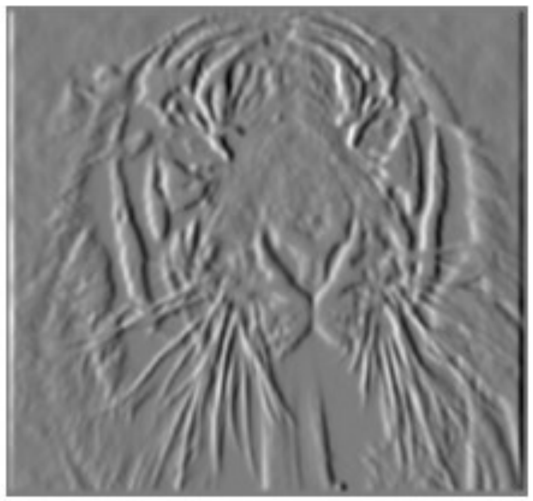


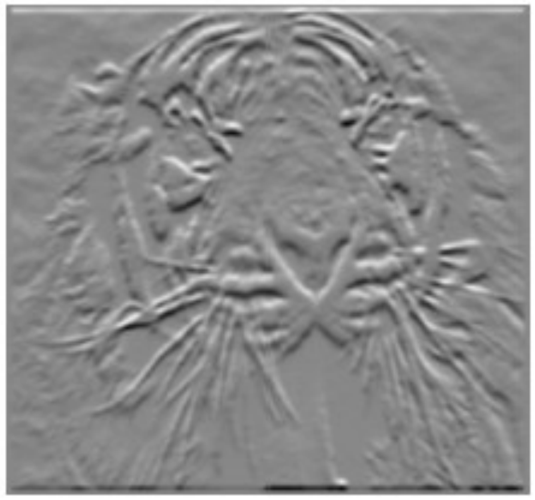
Source: L. Lazebnik

Partial derivatives of an image

$$\frac{\partial f(x,y)}{\partial x}$$

$$\frac{\partial f(x,y)}{\partial y}$$





-1	1

-1
1

? or

1
-1

Which one shows changes with respect to x? (showing flipped filters)

Assorted finite difference filters

Prewitt: $M_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$; $M_y = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$

Sobel: $M_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$; $M_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$

Roberts: $M_x = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline -1 & 0 \\ \hline \end{array}$; $M_y = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array}$

```
>> My = fspecial('sobel');
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```

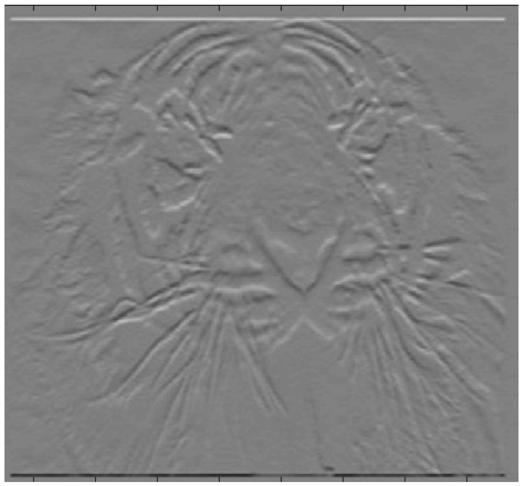
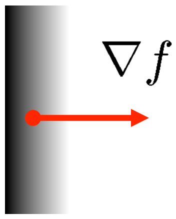


Image gradient

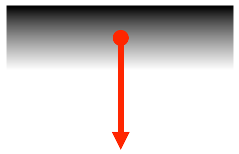
The gradient of an image:

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

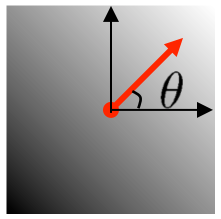
The gradient points in the direction of most rapid change in intensity



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



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



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

The gradient direction (orientation of edge normal) is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

The *edge strength* is given by the gradient magnitude

$$||\nabla f|| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

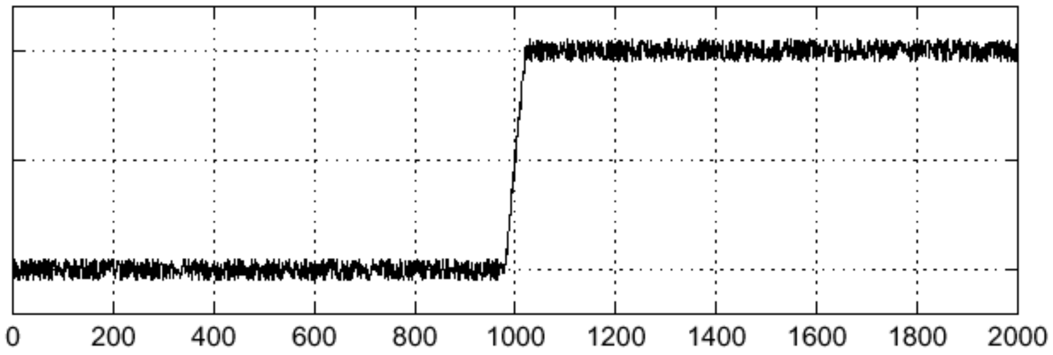
Slide credit S. Seitz

Effects of noise

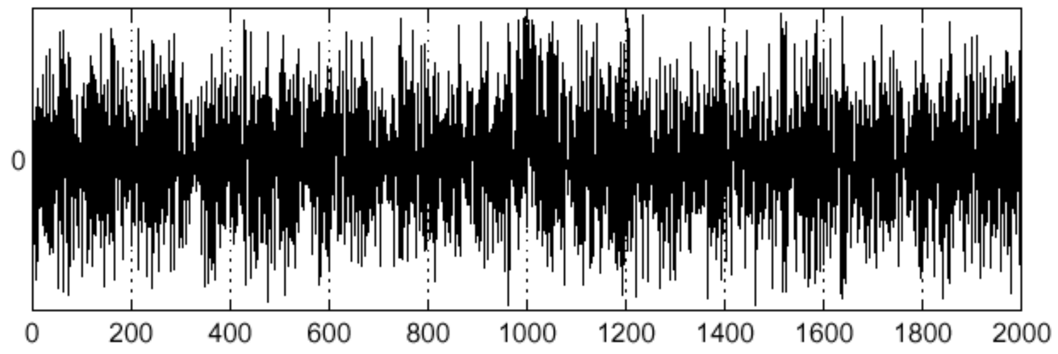
Consider a single row or column of the image

– Plotting intensity as a function of position gives a signal

$f(x)$



$\frac{d}{dx}f(x)$



Where is the edge?

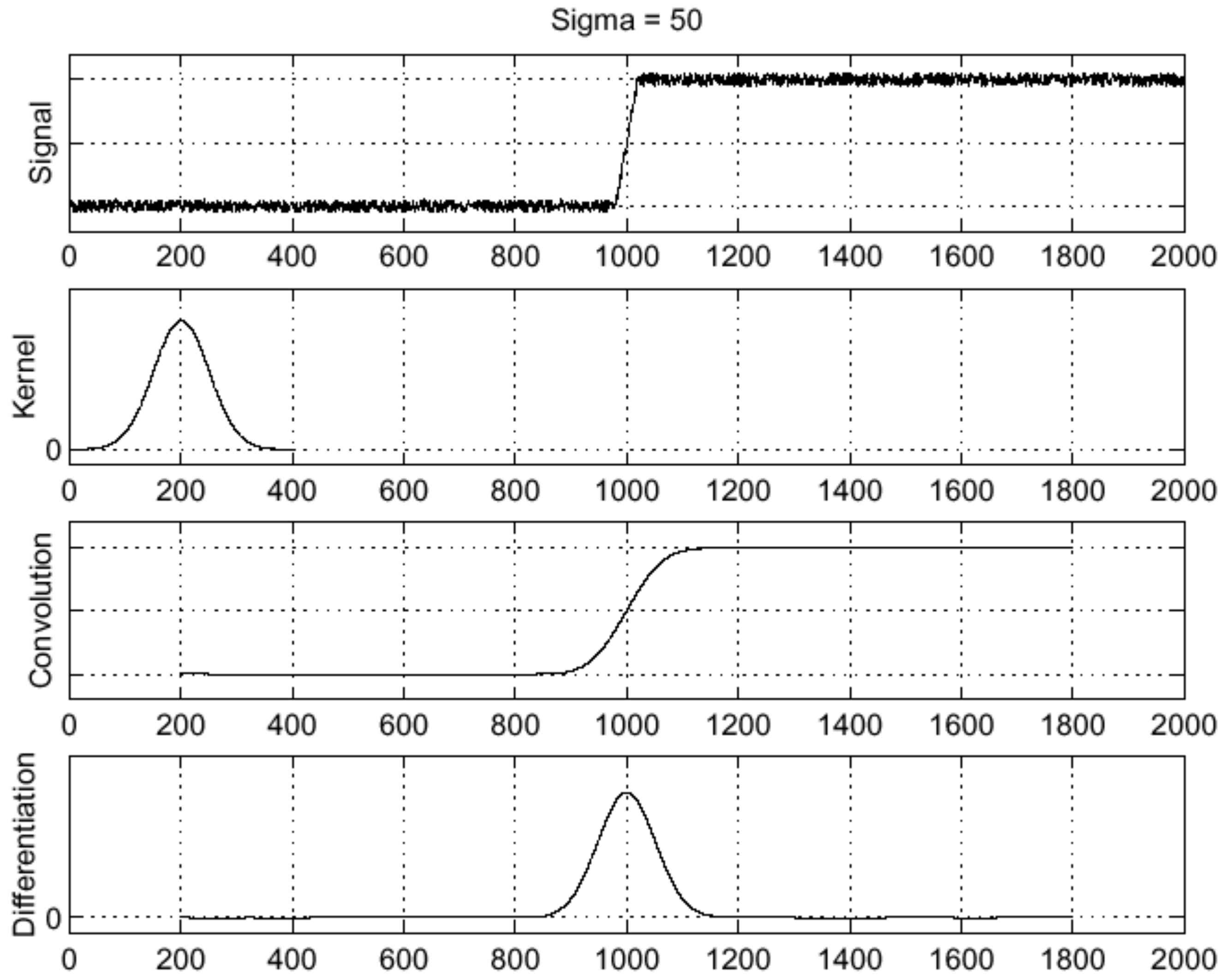
Solution: smooth first

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



Where is the edge? Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Properties of convolution

$$(I * K)(x,y) = \sum_u \sum_v I(x \underset{\nwarrow\ \text{flip}}{-} u, y - v) \times K(u,v)$$

- Linear & shift invariant
- Commutative: f * g = g * f
- Associative: (f * g) * h = f * (g * h)
- Identity:

  unit impulse e = [..., 0, 0, 1, 0, 0, ...]. f * e = f
- Differentiation:

$$\frac{\partial}{\partial x}(f * g) = \frac{\partial f}{\partial x} * g$$

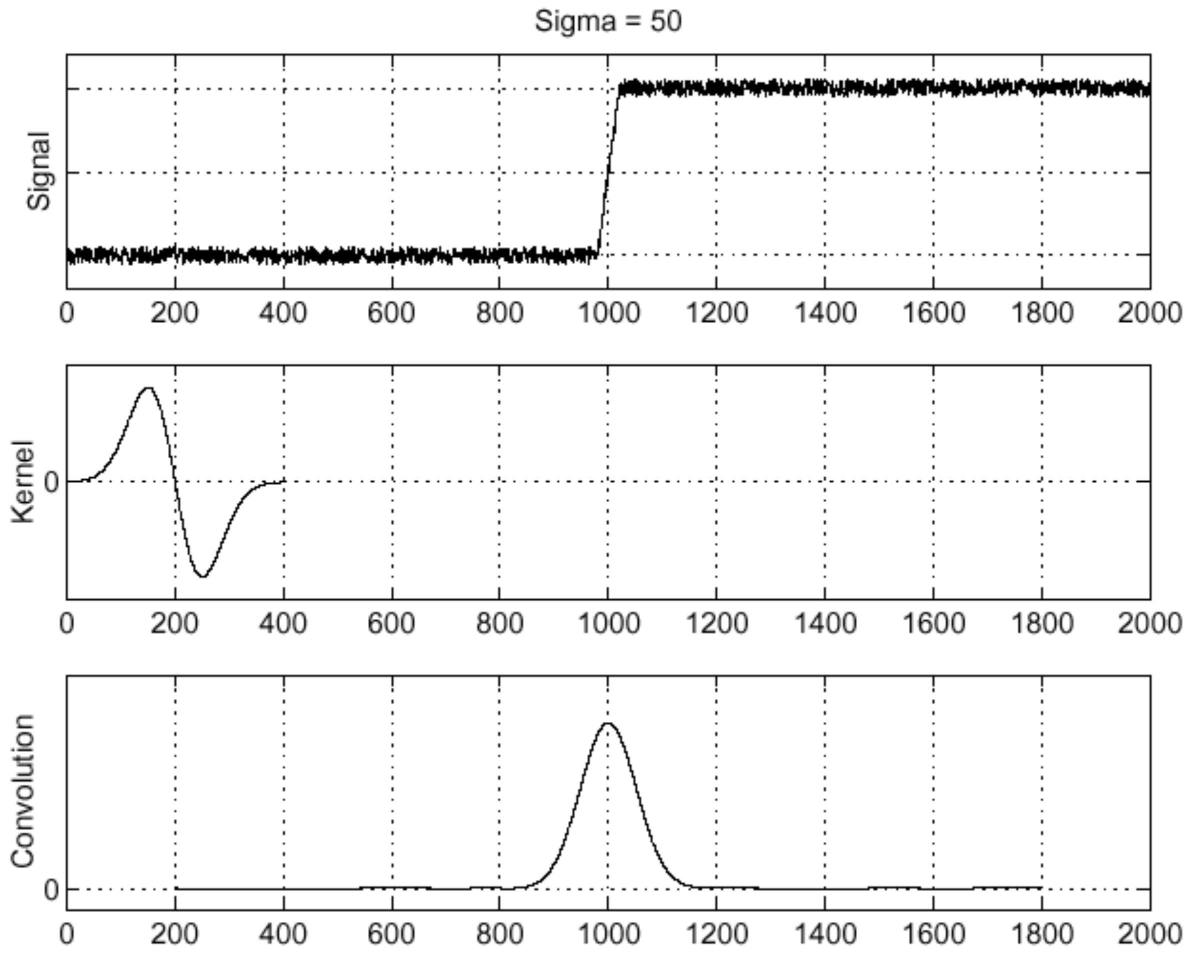
Matlab
check conv2, imfilter

Derivative of a Gaussian

f

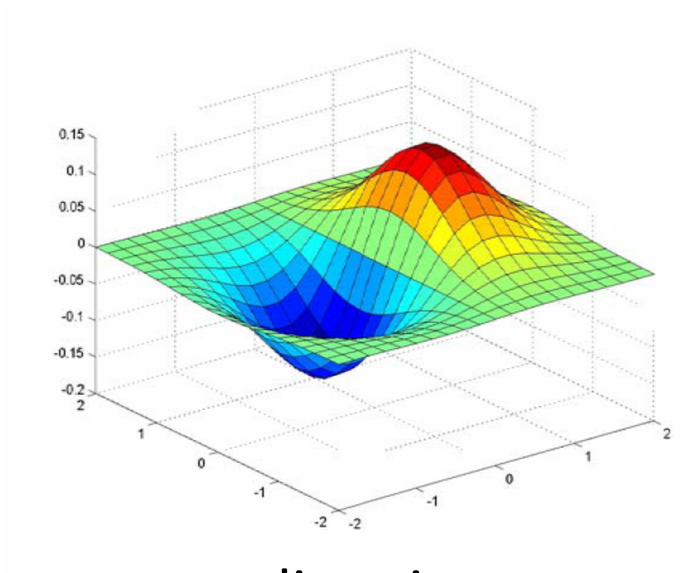
$\frac{\partial}{\partial x}h$

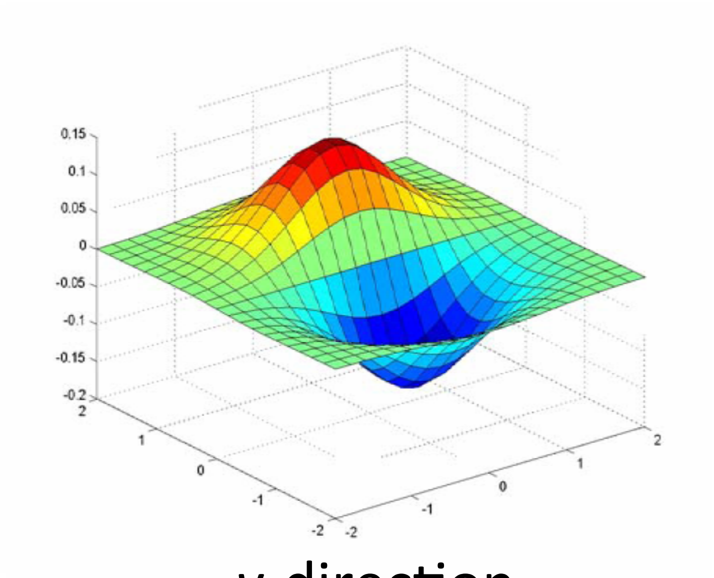
$(\frac{\partial}{\partial x}h) \star f$



Look for peaks in $(\frac{\partial}{\partial x}h) \star f$

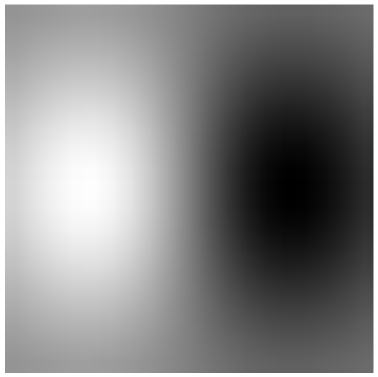
Derivative of Gaussian filters

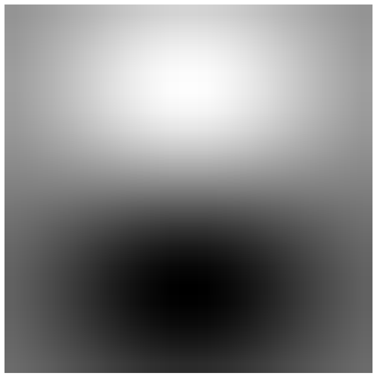




x-direction

y-direction





Source: L. Lazebnik

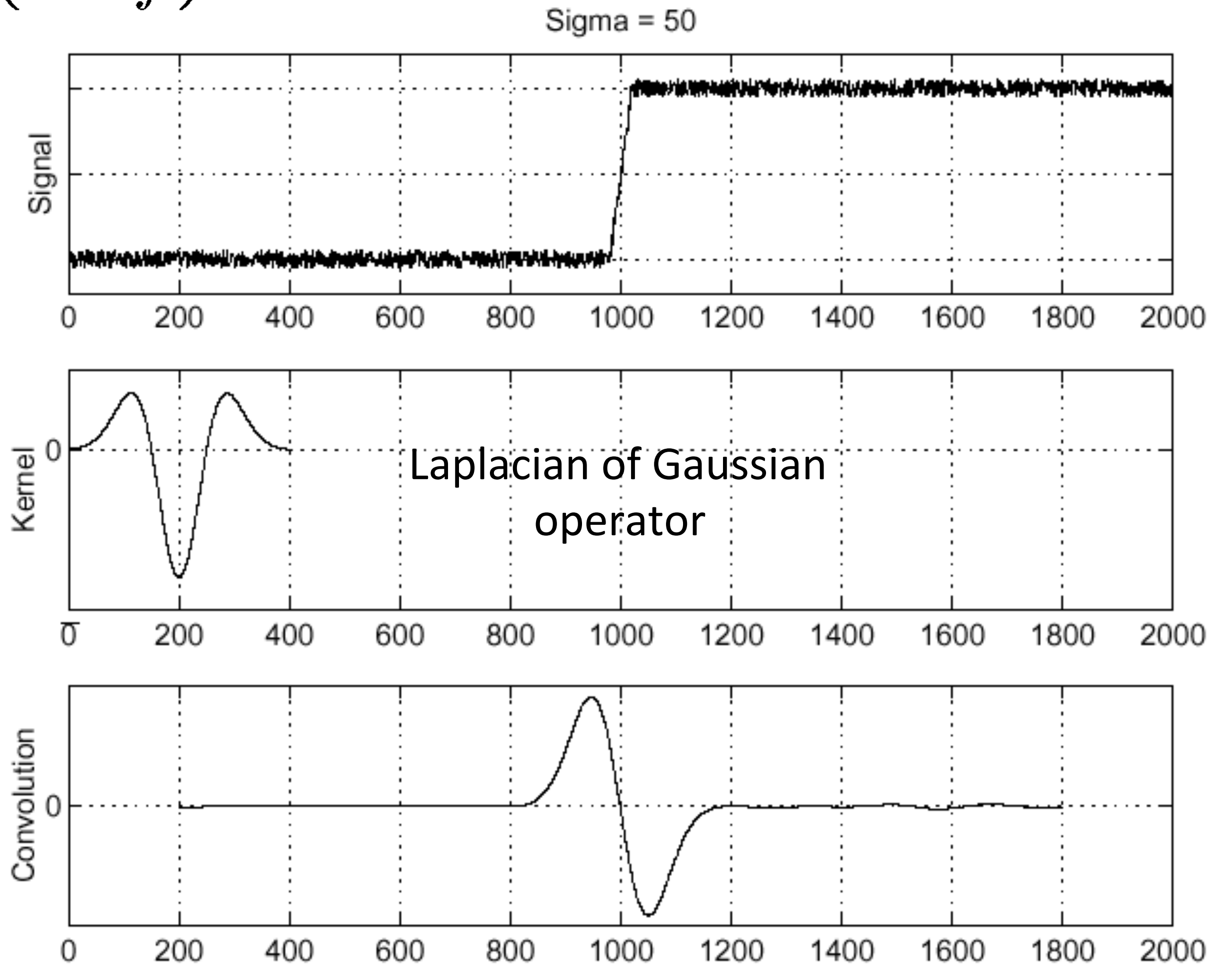
Laplacian of Gaussian

Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

f

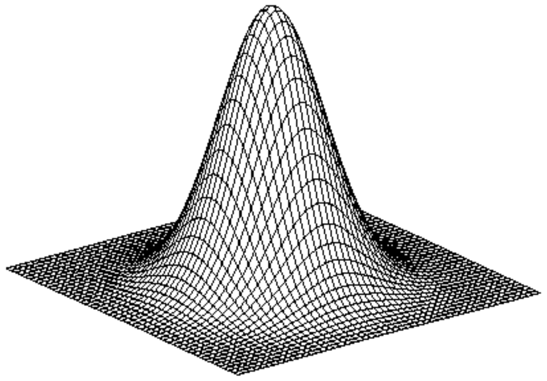
$\frac{\partial^2}{\partial x^2}h$

$(\frac{\partial^2}{\partial x^2}h) \star f$



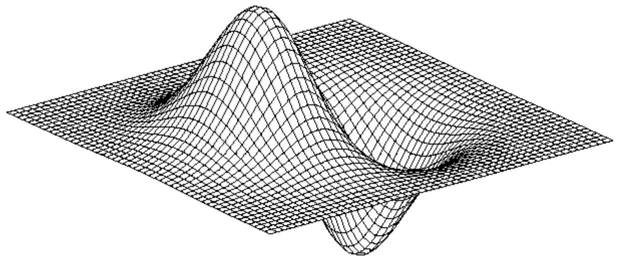
Where is the edge? Zero-crossings of bottom graph

2D edge detection filters



Gaussian

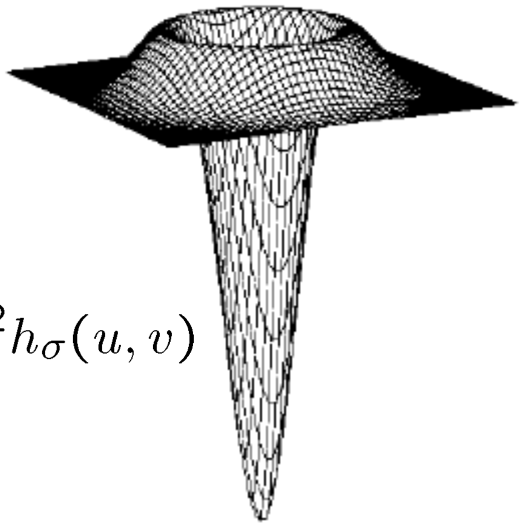
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

$$\frac{\partial}{\partial x} h_\sigma(u,v)$$

Laplacian of Gaussian



$$\nabla^2 h_\sigma(u,v)$$

- ∇^2 is the Laplacian operator:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Notes on Kernels

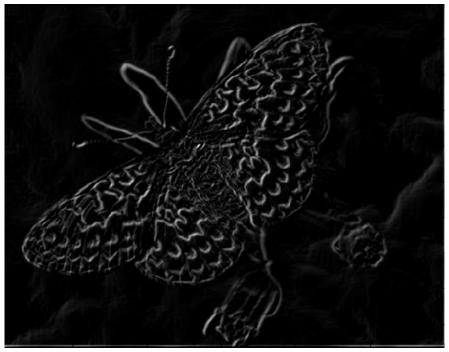
- Smoothing
  - Values positive
  - Sum to 1 → constant regions same as input
  - Amount of smoothing proportional to mask size
  - Remove "high-frequency" components; "low-pass" filter
- Derivatives
  - Opposite signs used to get high response in regions of high contrast
  - Sum to 0 → no response in constant regions
  - High absolute value at points of high contrast
- Filters act as templates
  - Highest response for regions that "look the most like the filter"
  - Correlation can be seen as a dot product if image and mask are represented as two vectors

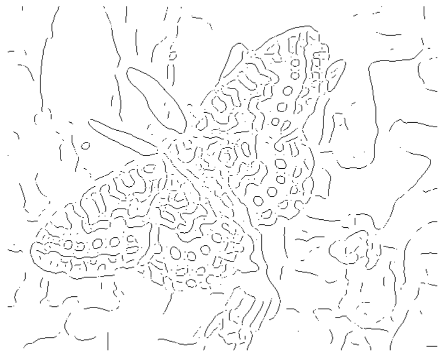
From gradients to edges

Primary edge detection steps:

1. Smoothing: suppress noise
2. Edge enhancement: filter for contrast
3. Edge localization

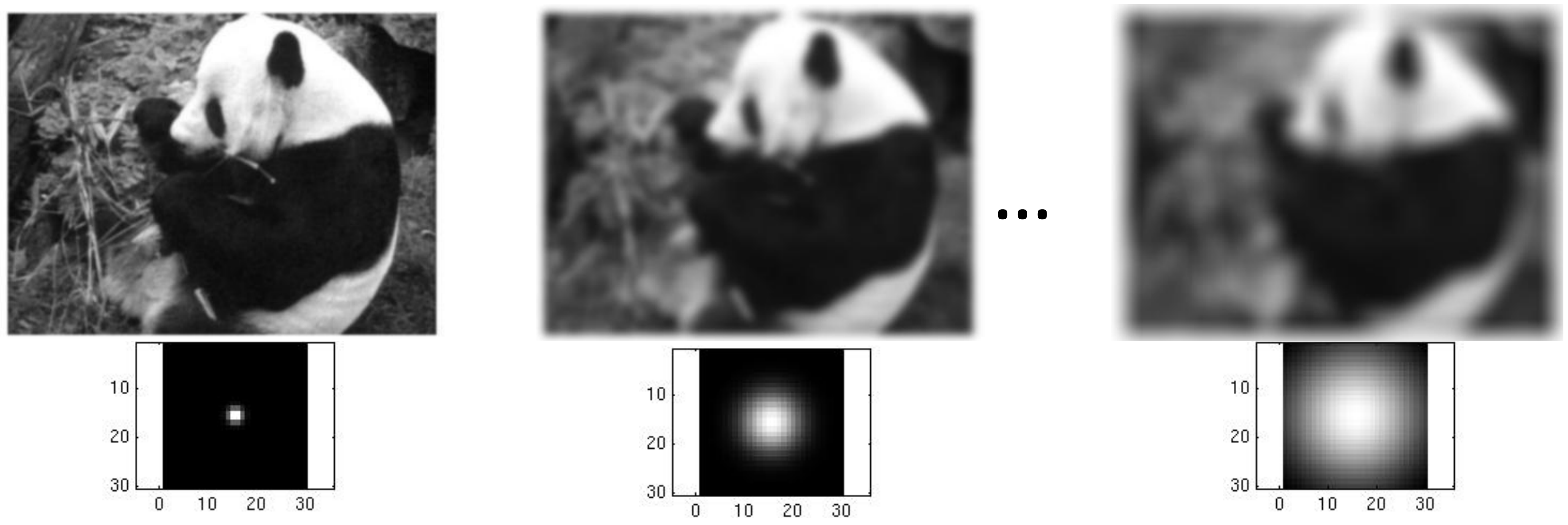
   Determine which local maxima from filter output are actually edges vs. noise → **thresholding, thinning**



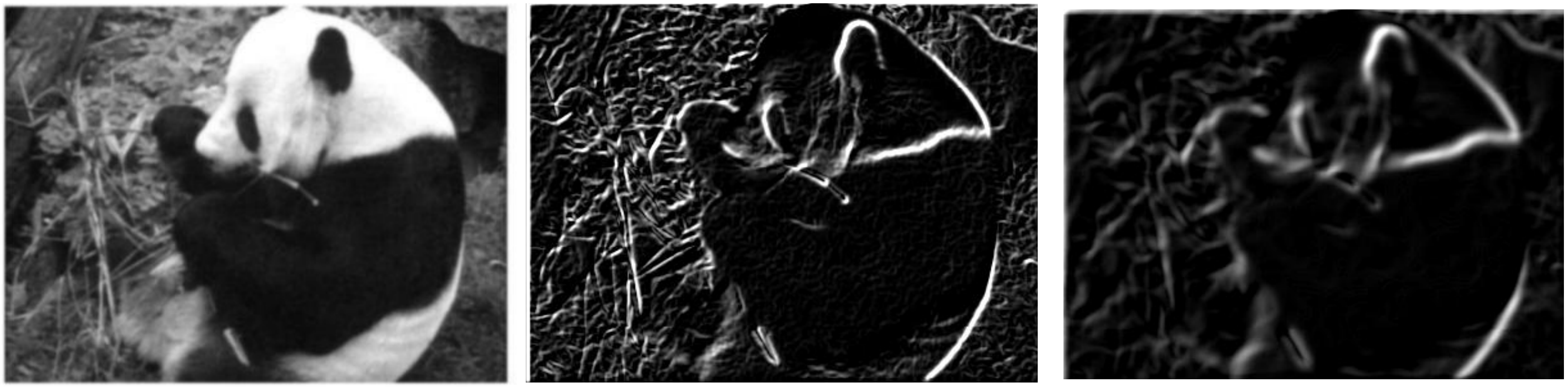


Smoothing with a Gaussian

Recall: parameter σ is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.



Effect of σ on derivatives



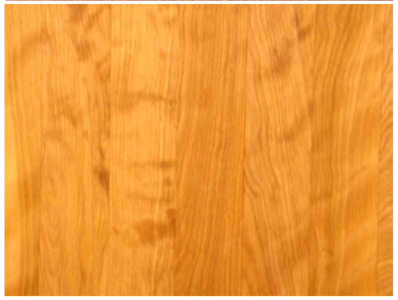
σ = 1 pixel σ = 3 pixels

The apparent structures differ depending on Gaussian's scale parameter.

Larger values: larger scale edges detected
Smaller values: finer features detected

So, what scale to choose?

It depends on what we're looking for:



Too fine of a scale…can't see the forest for the trees.
Too coarse of a scale…can't tell the maple grain from the cherry.

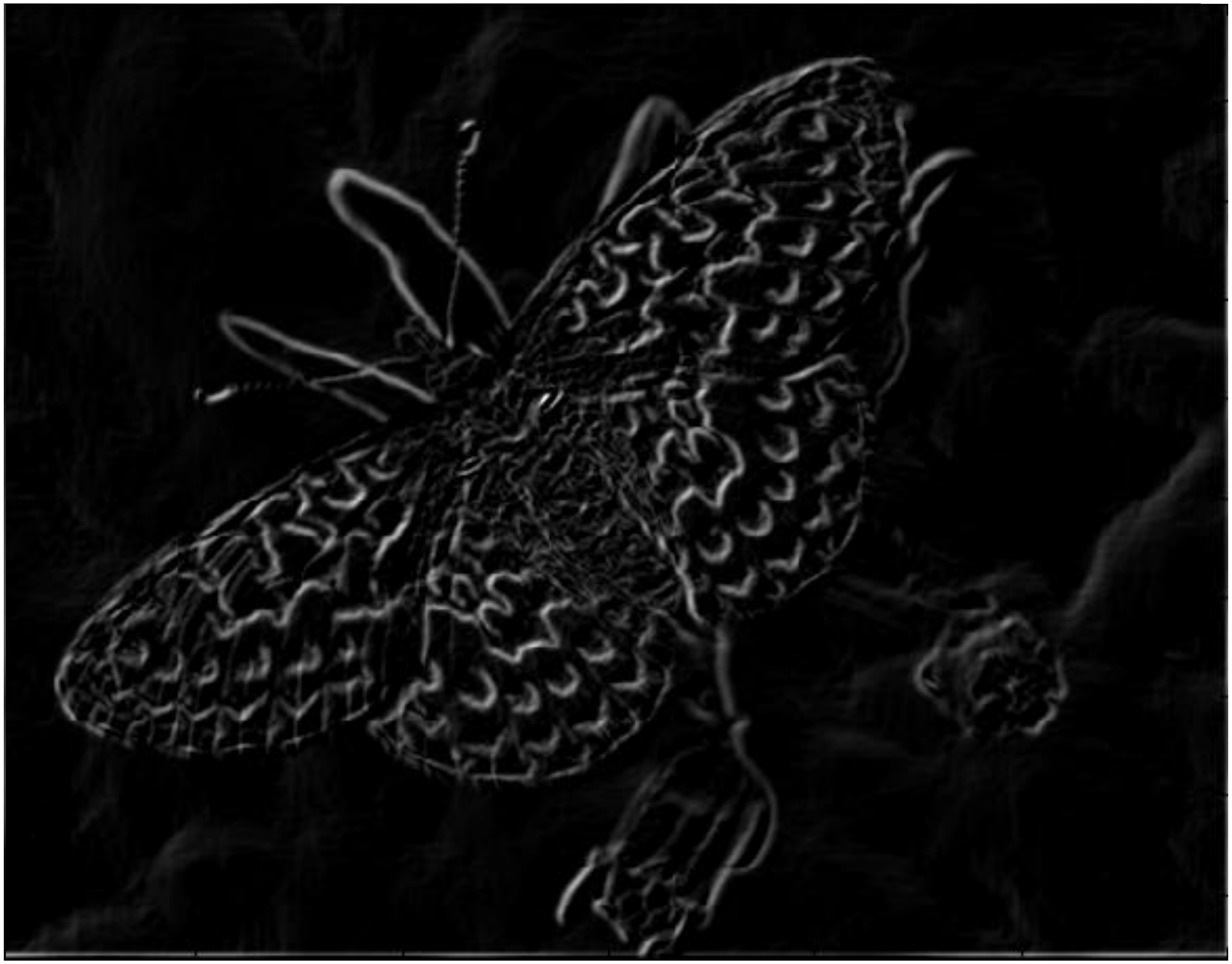
Reminder: thresholding

- Choose a threshold value a
- Set any pixels less than a to zero
- Set any pixels greater than or equal to a to one

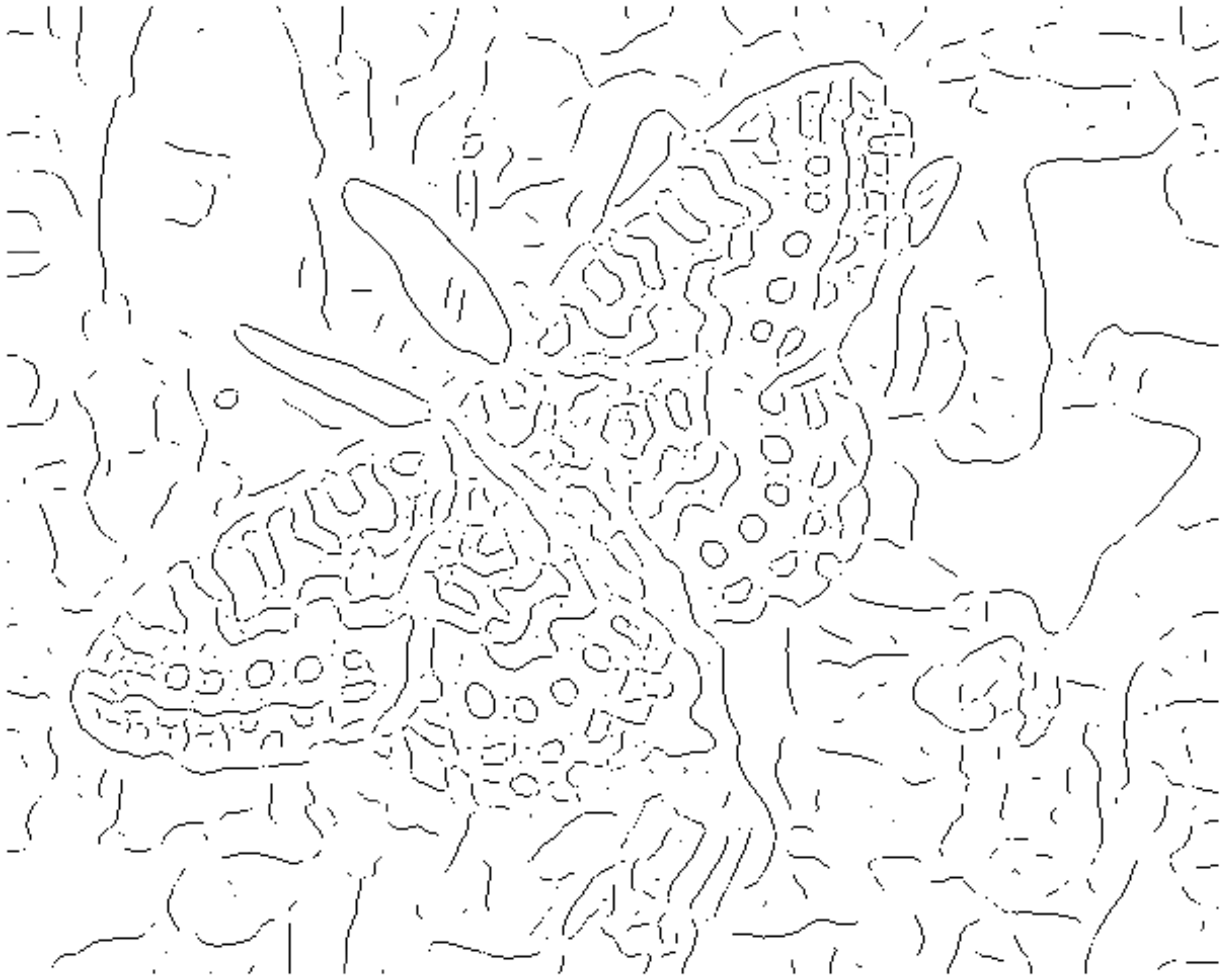
$$I^{'}(x,y) = \begin{cases} 1 & \text{if} \quad I(x,y) \geq a \\ 0 & \text{else} \end{cases}$$

Original image

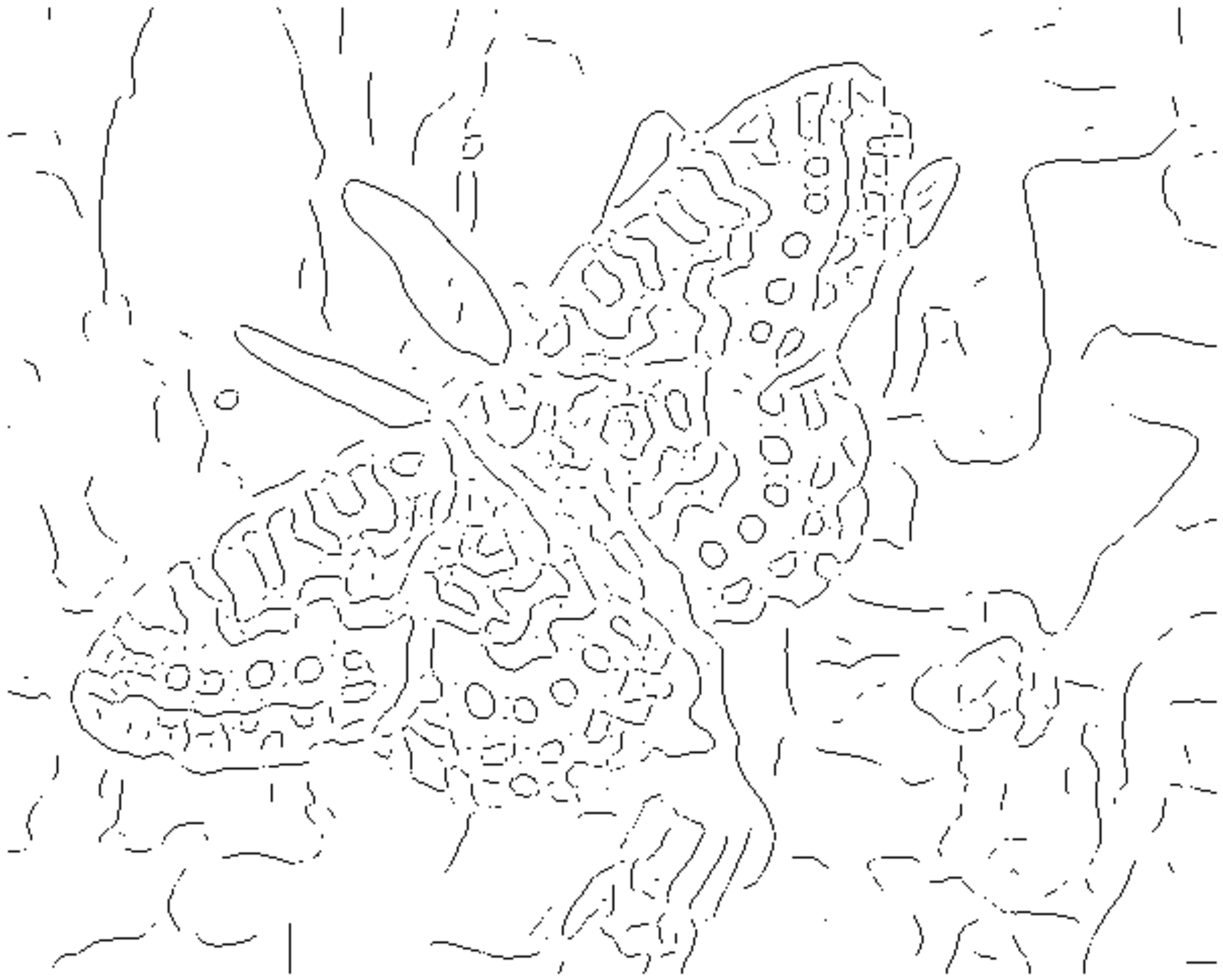
Gradient magnitude image



Thresholding gradient with a lower threshold

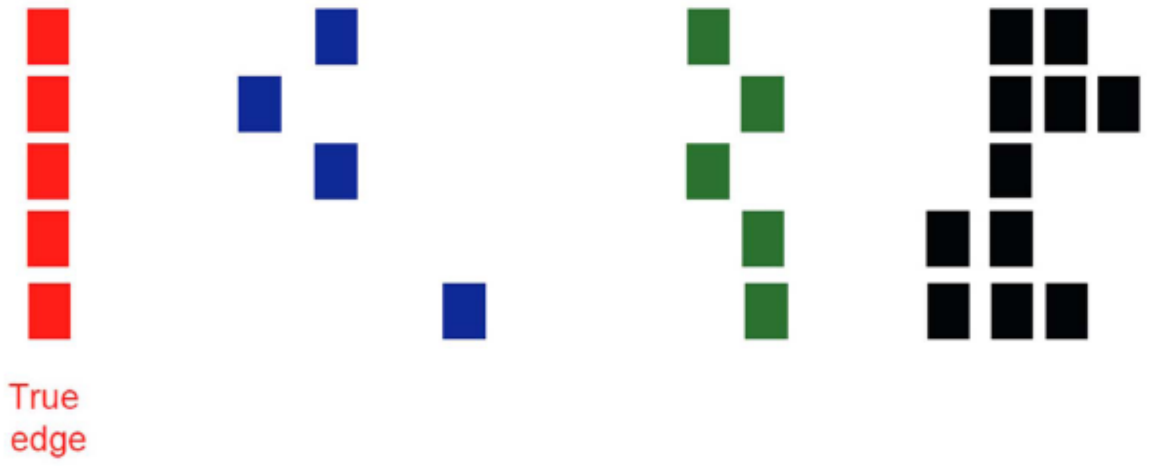


Thresholding gradient with a higher threshold



Designing an edge detector

- Criteria for an "optimal" detector
  - Good detection
  - Good localization
  - Single response



Canny edge detector

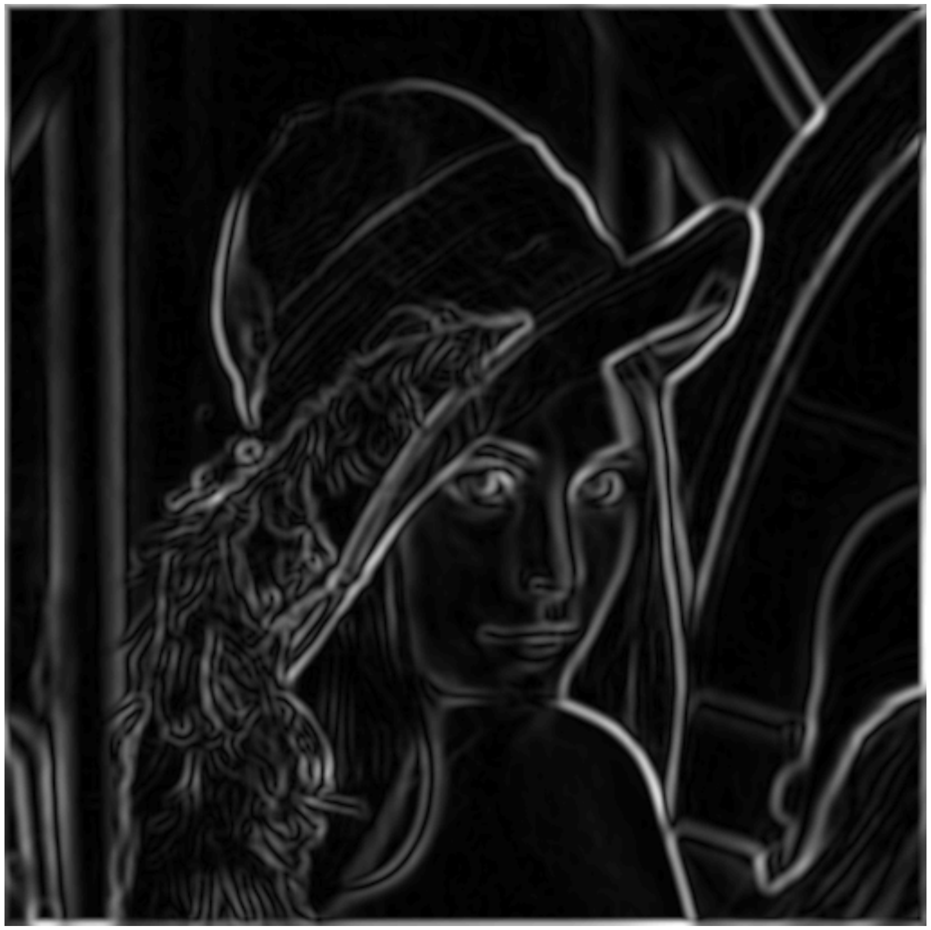
- Filter image with derivative of Gaussian (DoG)
- Find magnitude and orientation of gradient
- **Non-maximum suppression**:
  - Thin multi-pixel wide "ridges" down to single pixel width
- Linking and thresholding (**hysteresis**):
  - Define two thresholds: low and high
  - Use the high threshold to start edge curves and the low threshold to continue them

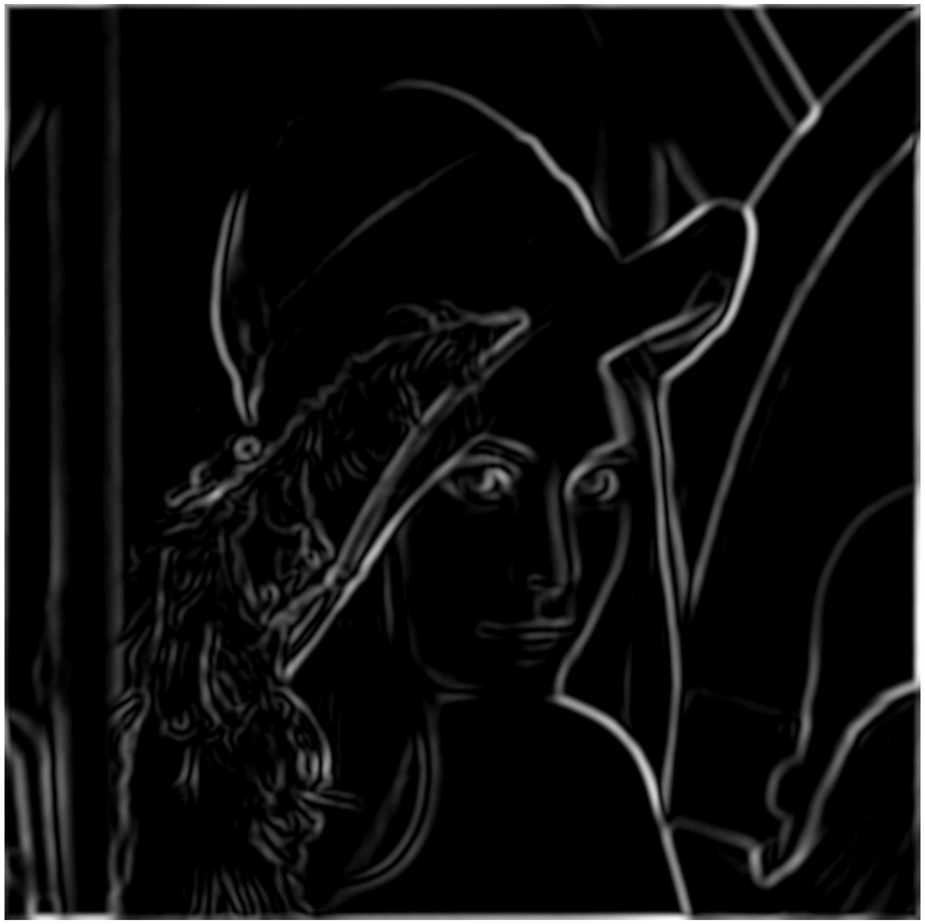
- MATLAB: **`edge(image, 'canny');`**
- **`>>help edge`**

Source: D. Lowe, L. Fei-Fei

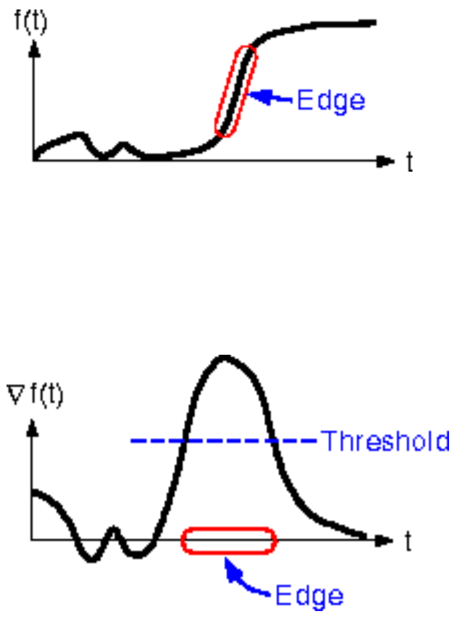
The Canny edge detector

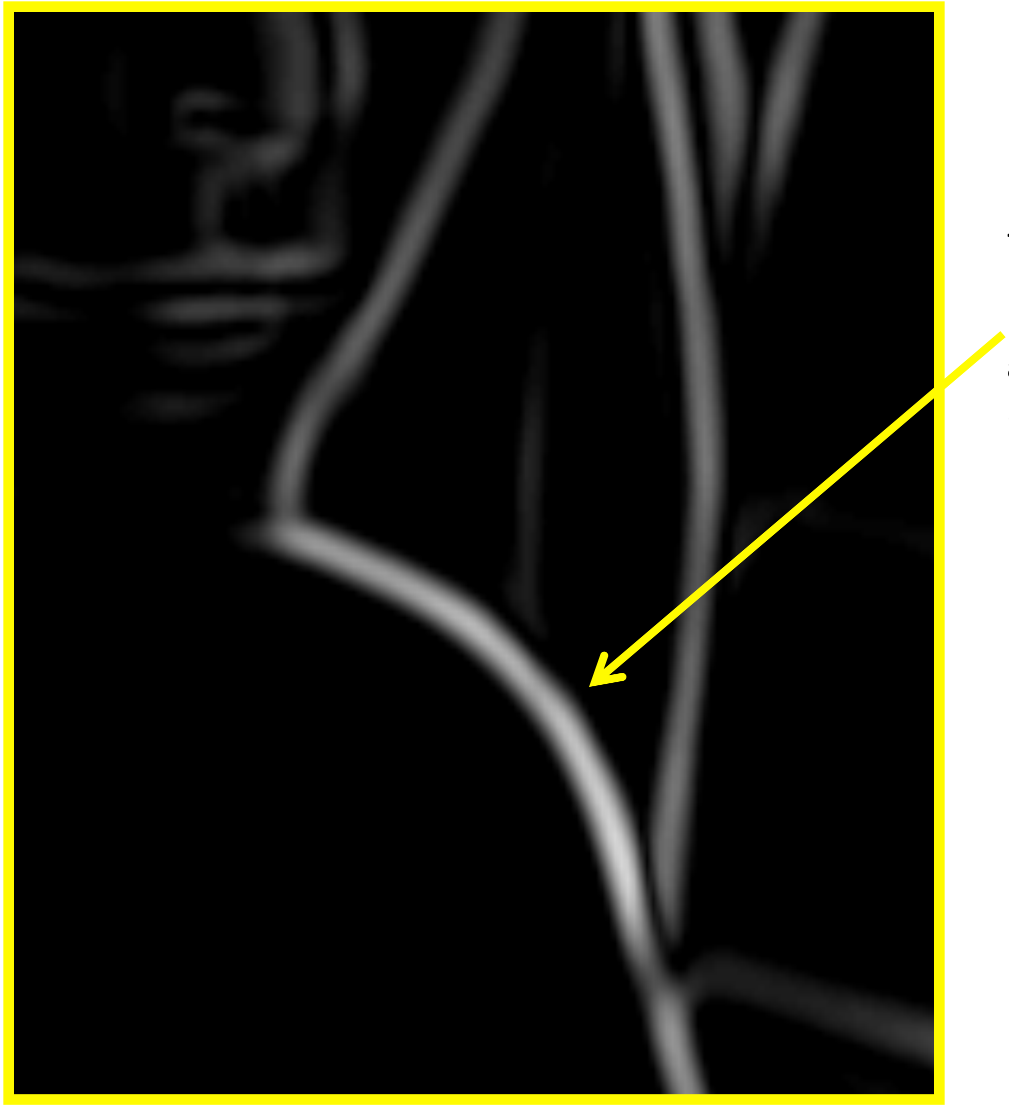
original image (Lena)


norm of the gradient



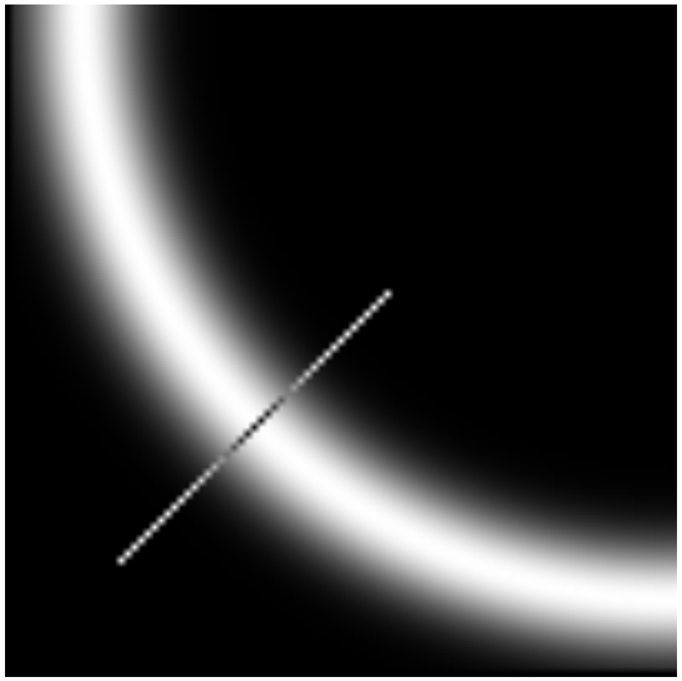
thresholding

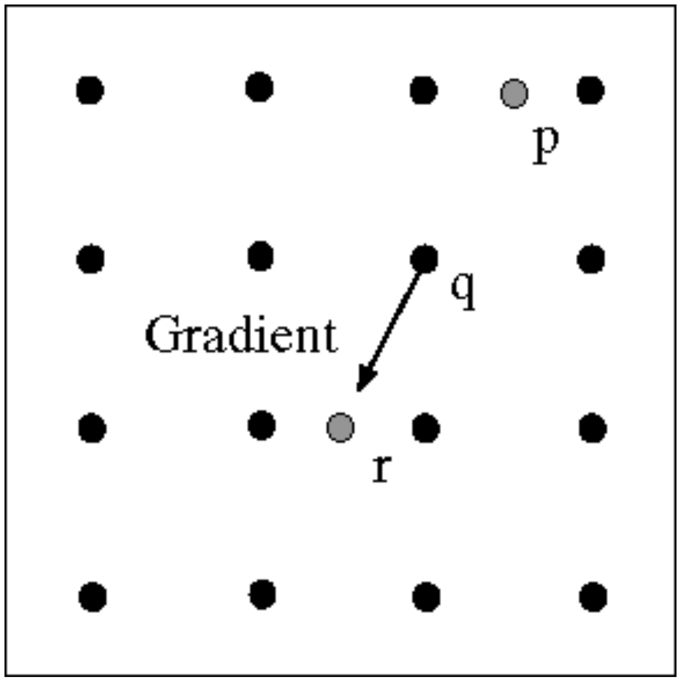




How to turn these thick regions of the gradient into curves?

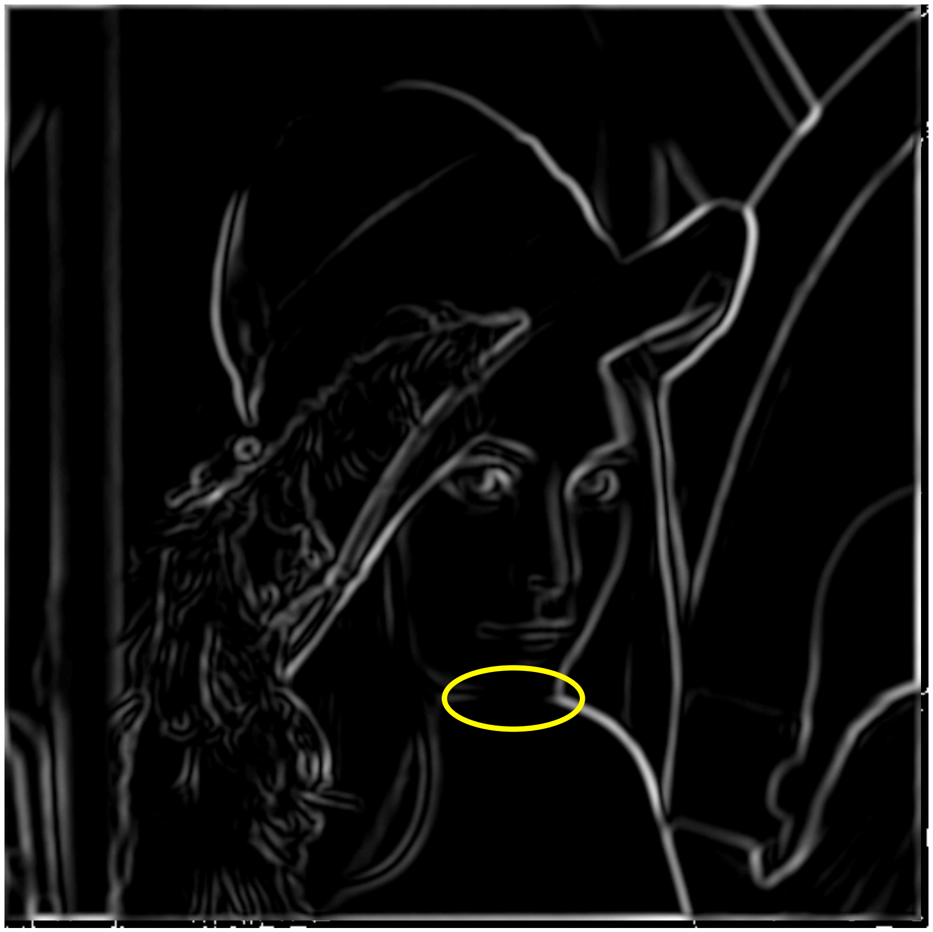
Non-maximum suppression





Check if pixel is local maximum along gradient direction, select single max across width of the edge

– requires checking interpolated pixels p and r

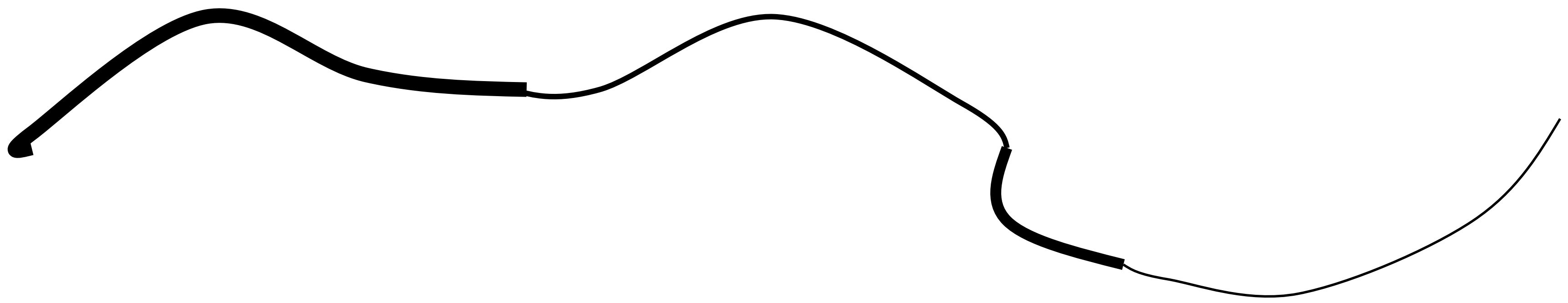


Problem: pixels along this edge didn't survive the thresholding

thinning
(non-maximum suppression)

Hysteresis thresholding

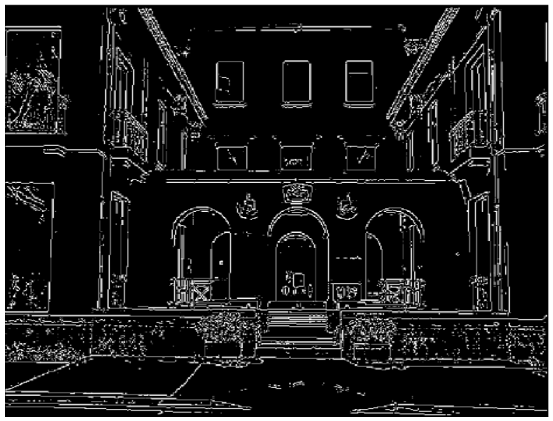
- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.

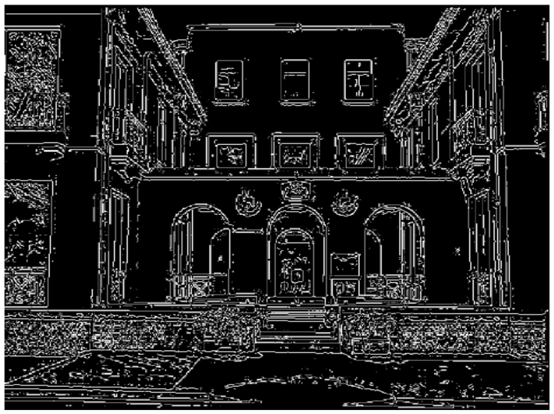


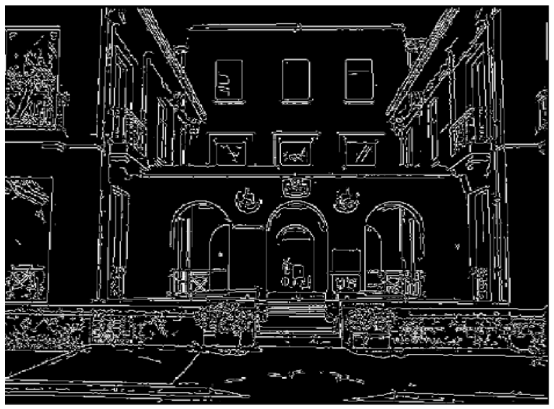
Source: S. Seitz

Hysteresis thresholding

original image

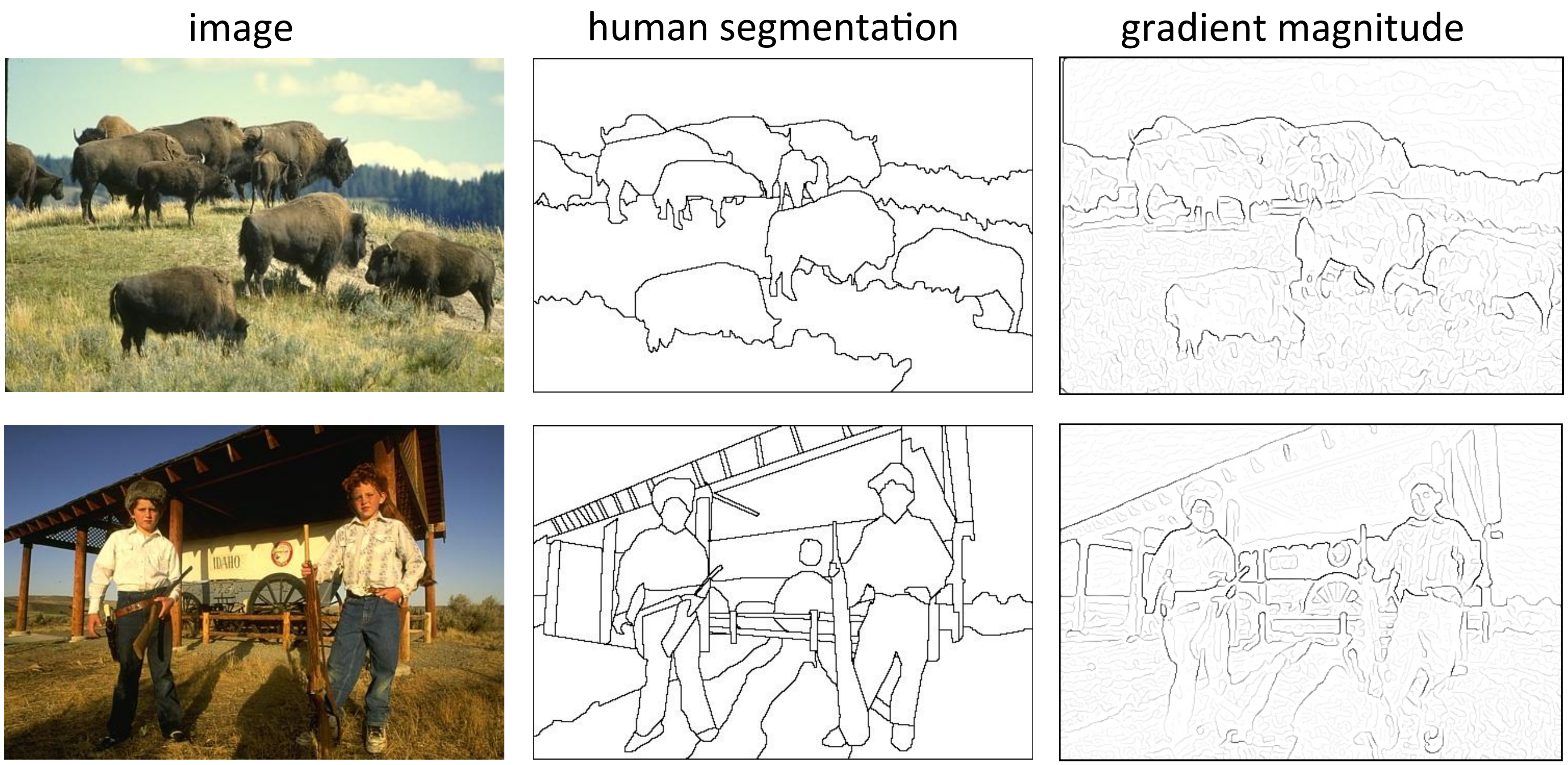

high threshold
(strong edges)


low threshold
(weak edges)


hysteresis threshold

Source: L. Fei-Fei

Edge detection is just the beginning…



Berkeley segmentation database:

http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/

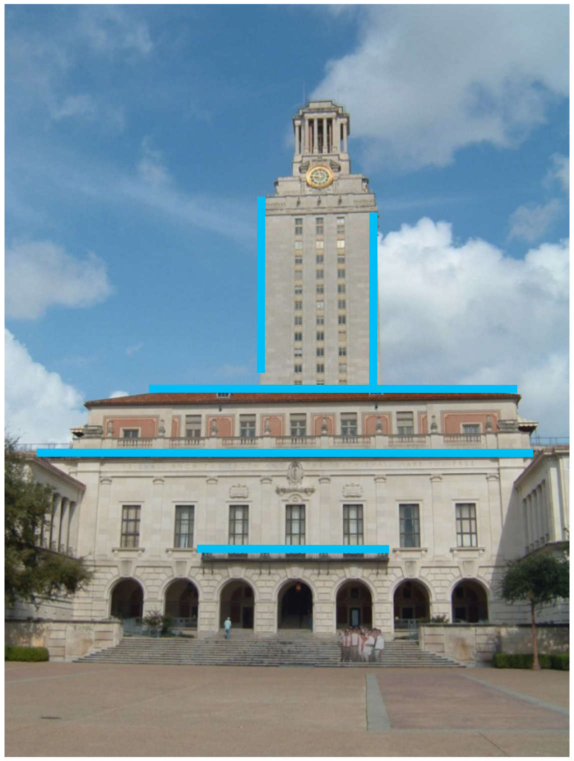
More on segmentation later in term…

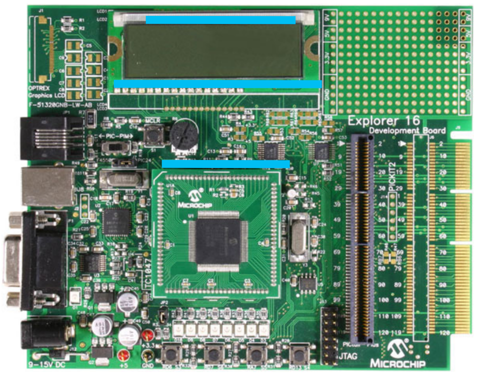
Source: L. Lazebnik

Lines - Hough

Line fitting

- Why fit lines? Many objects characterized by presence of straight lines

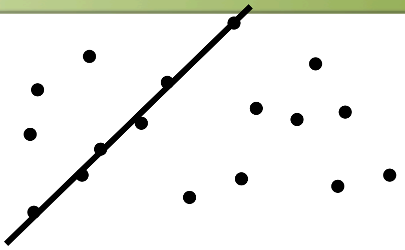


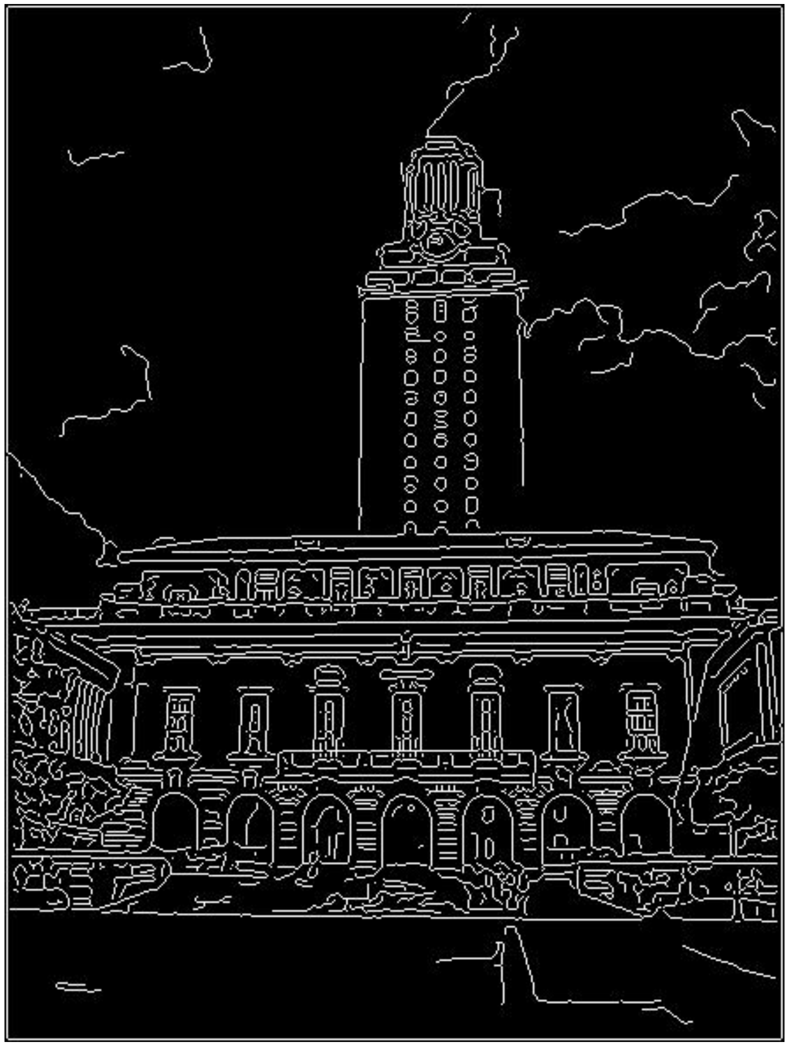


- Wait, why aren't we done just by running edge detection?

Kristen Grauman

Difficulty of line fitting





- **Extra** edge points (clutter), multiple models:
  - which points go with which line, if any?
- Only some parts of each line detected, and some parts are **missing:**
  - how to find a line that bridges missing evidence?
- **Noise** in measured edge points, orientations:
  - how to detect true underlying parameters?

Kristen Grauman

Voting

- It’s not feasible to check all combinations of features by fitting a model to each possible subset.
- **Voting** is a general technique where we let the features *vote for all models that are compatible with it*.
  - Cycle through features, cast votes for model parameters.
  - Look for model parameters that receive a lot of votes.
- Noise & clutter features will cast votes too, *but* typically their votes should be inconsistent with the majority of “good” features.

Kristen Grauman

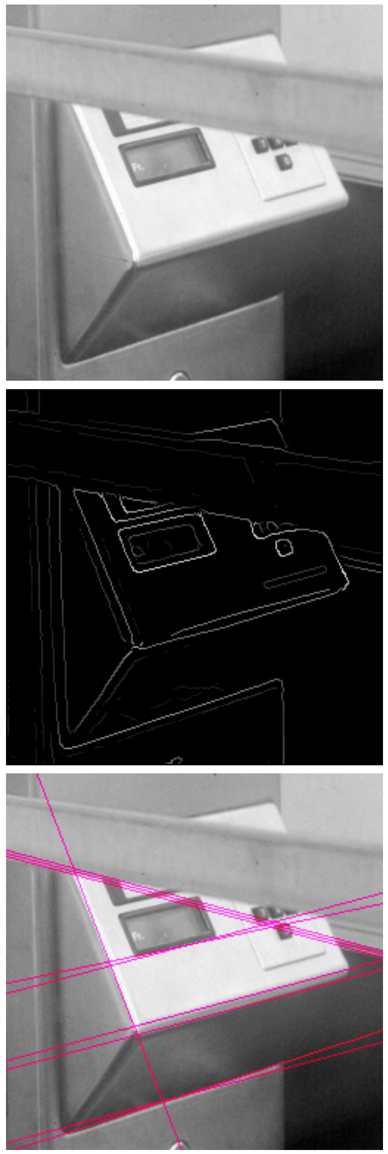
Fitting lines: Hough transform

- Given points that belong to a line, what is the line?
- How many lines are there?
- Which points belong to which lines?

- **Hough Transform** is a voting technique that can be used to answer all of these questions.

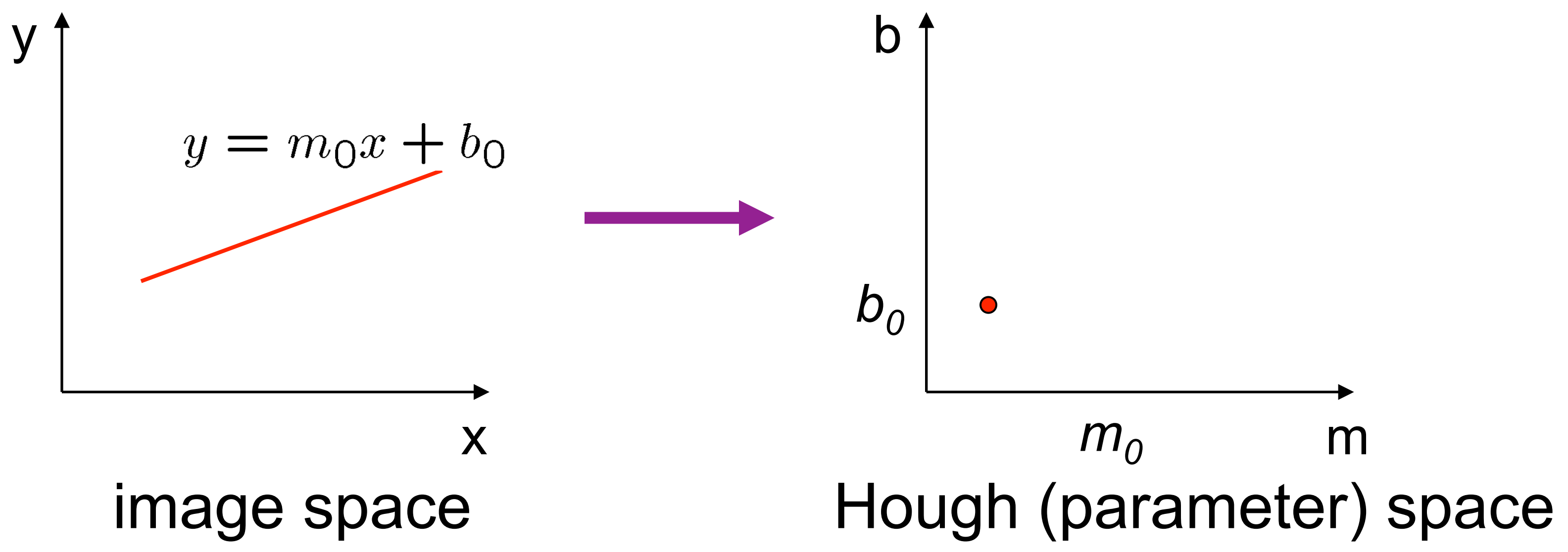
  Main idea:

  1. Record vote for each possible line on which each edge point lies.
  2. Look for lines that get many votes.



Kristen Grauman

Finding lines in an image: Hough space

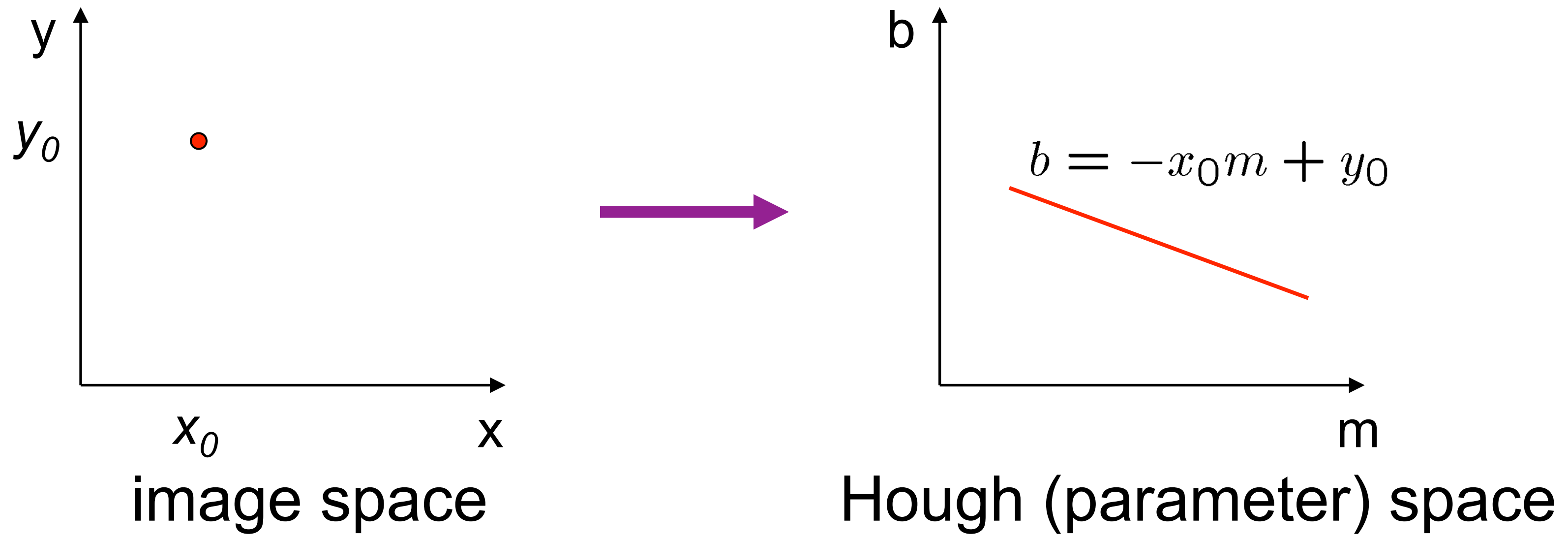


Connection between image (x,y) and Hough (m,b) spaces

- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
    - given a set of points (x,y), find all (m,b) such that y = mx + b

Slide credit: Steve Seitz

Finding lines in an image: Hough space



Connection between image (x,y) and Hough (m,b) spaces

- A line in the image corresponds to a point in Hough space
- To go from image space to Hough space:
  - given a set of points (x,y), find all (m,b) such that y = mx + b
- What does a point (x_0, y_0) in the image space map to?
  - Answer: the solutions of $b = -x_0 m + y_0$
  - this is a line in Hough space

Slide credit: Steve Seitz

Finding lines in an image: Hough space

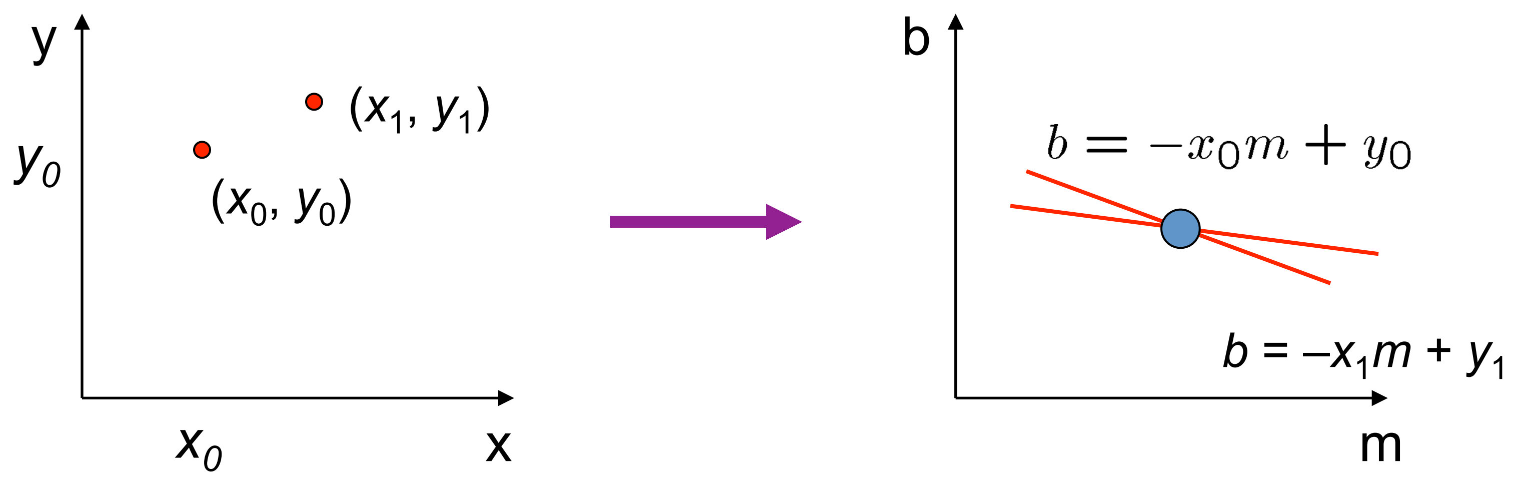


image space Hough (parameter) space

What are the line parameters for the line that contains both (x_0, y_0) and (x_1, y_1)?

- It is the intersection of the lines $b = -x_0 m + y_0$ and $b = -x_1 m + y_1$

Hough algorithm

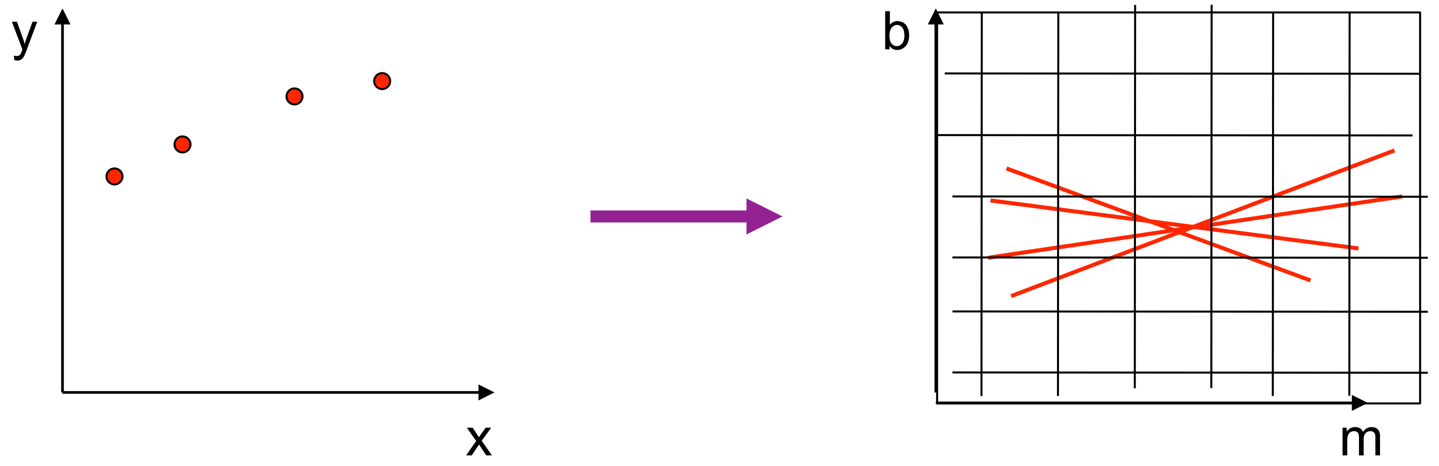


image space

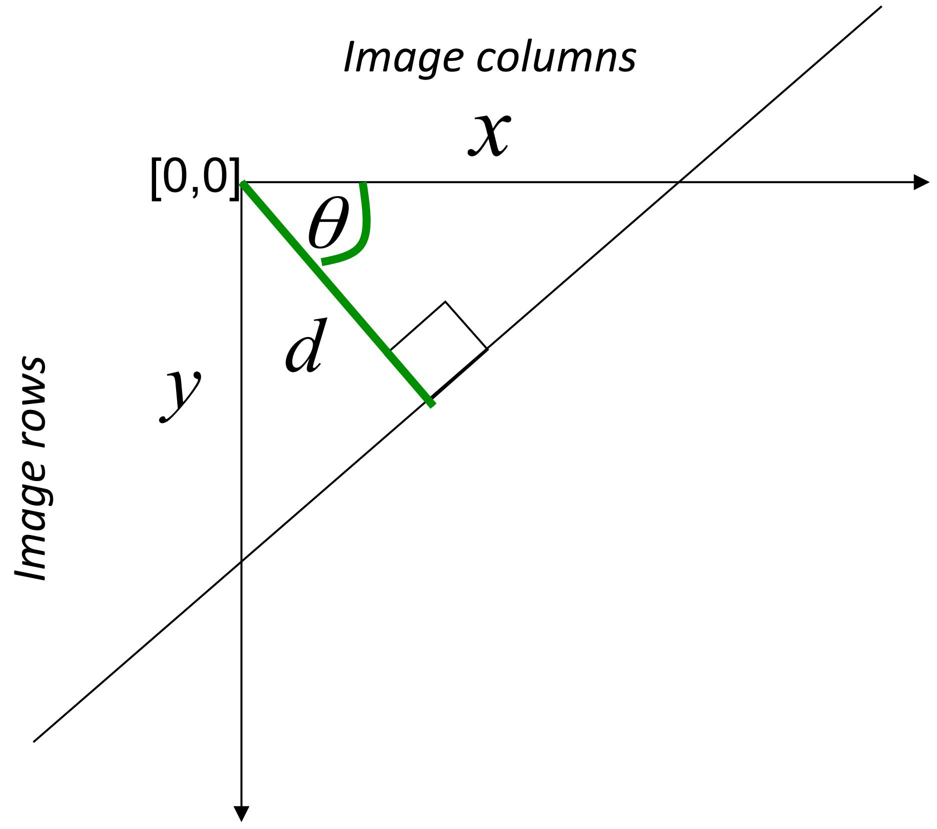
Hough (parameter) space

How can we use this to find the most likely parameters (m,b) for the most prominent line in the image space?

- Let each edge point in image space *vote* for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.

Polar representation for lines

Issues with usual (*m*,*b*) parameter space: can take on infinite values, undefined for vertical lines.



d : perpendicular distance from line to origin

θ : angle the perpendicular makes with the x-axis

$$x\cos\theta - y\sin\theta = d$$

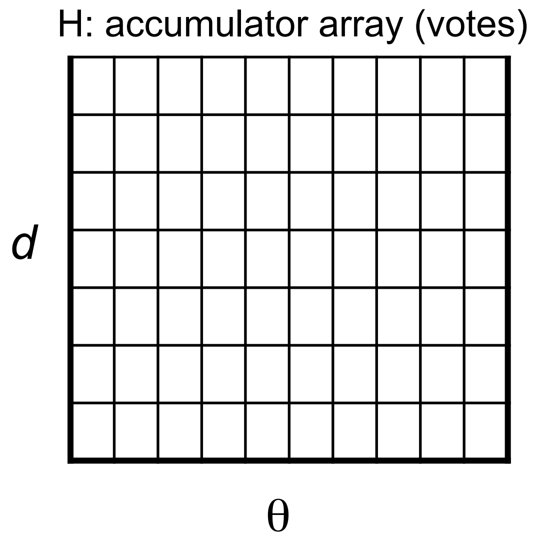
Point in image space → sinusoid segment in Hough space

Kristen Grauman

Hough transform algorithm

Using the polar parameterization:

$$x\cos\theta - y\sin\theta = d$$



Basic Hough transform algorithm

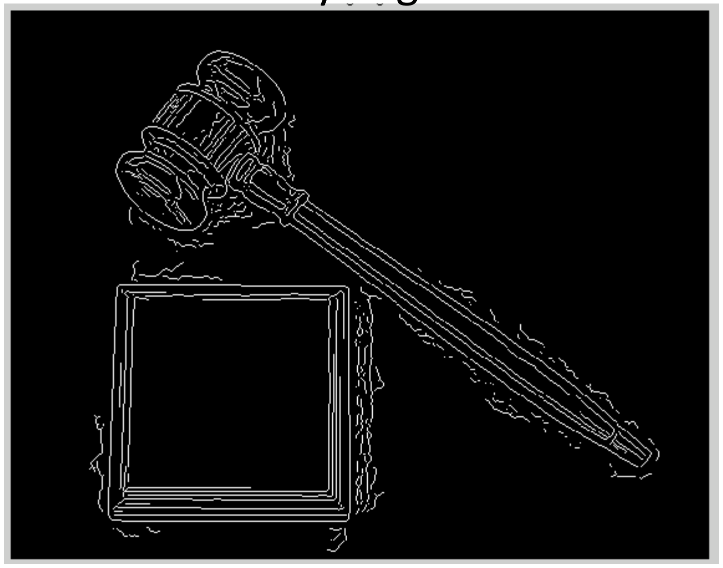
```
1. Initialize H[d, θ]=0
2. for each edge point I[x,y] in the image
       for θ = [$\theta_{min}$ to $\theta_{max}$ ]  // some quantization
           $d = x\cos\theta - y\sin\theta$
           H[d, θ] += 1
3. Find the value(s) of (d, θ) where H[d, θ] is maximum
4. The detected line in the image is given by $d = x\cos\theta - y\sin\theta$
```

Time complexity (in terms of number of votes per pt)?

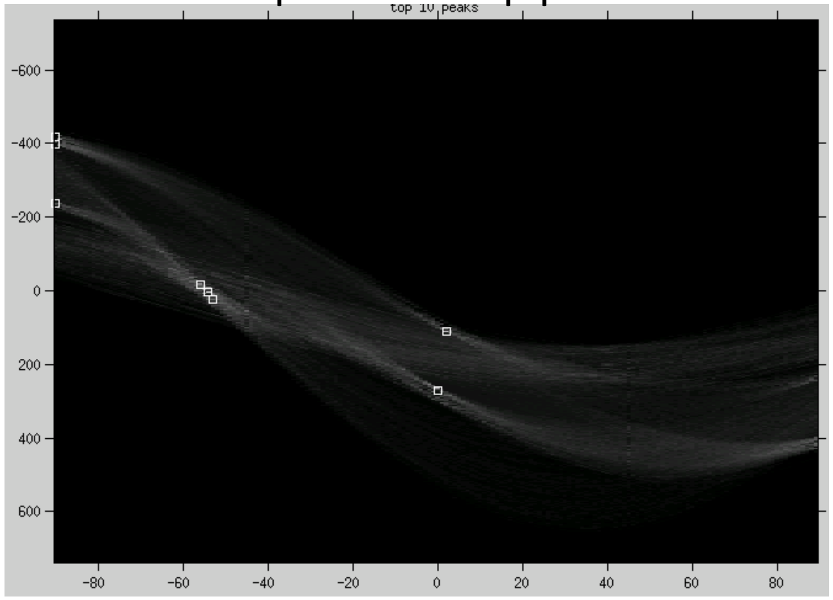
Source: Steve Seitz

Original image

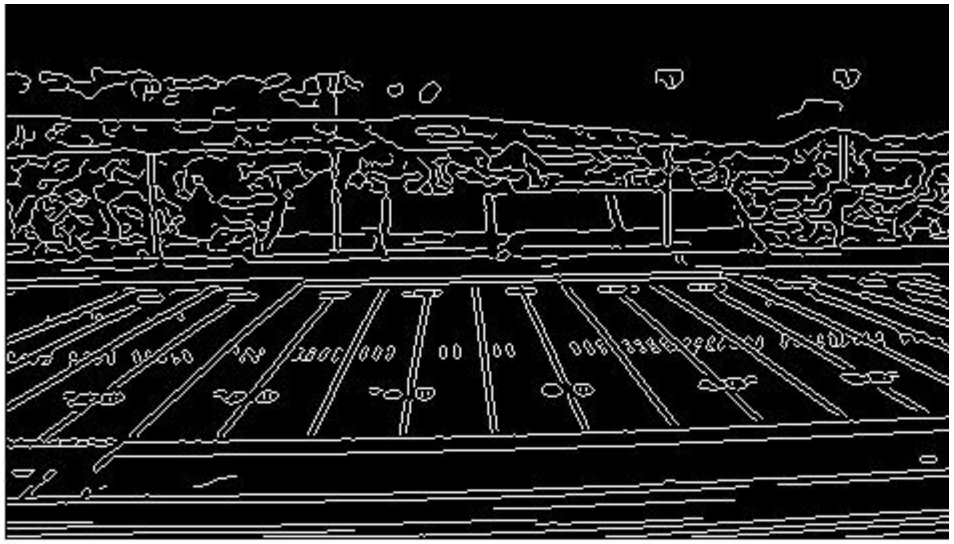
Canny edges

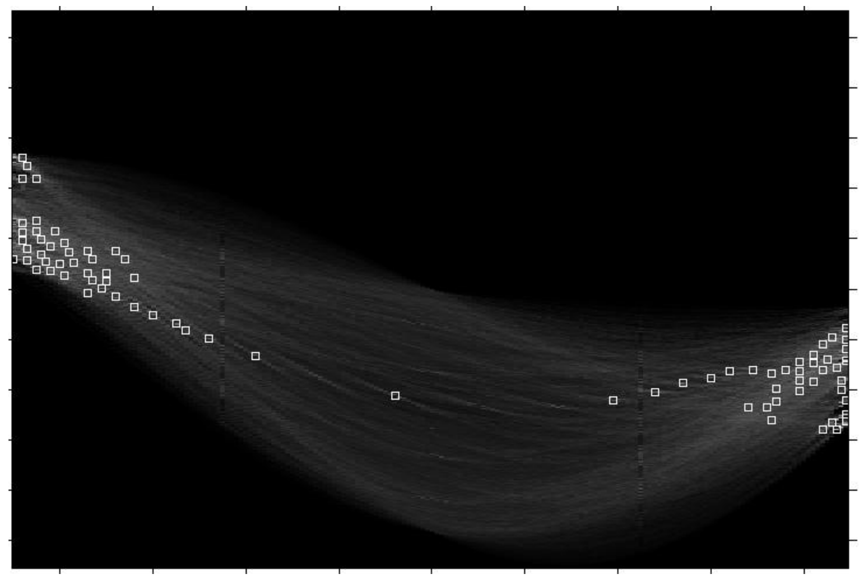


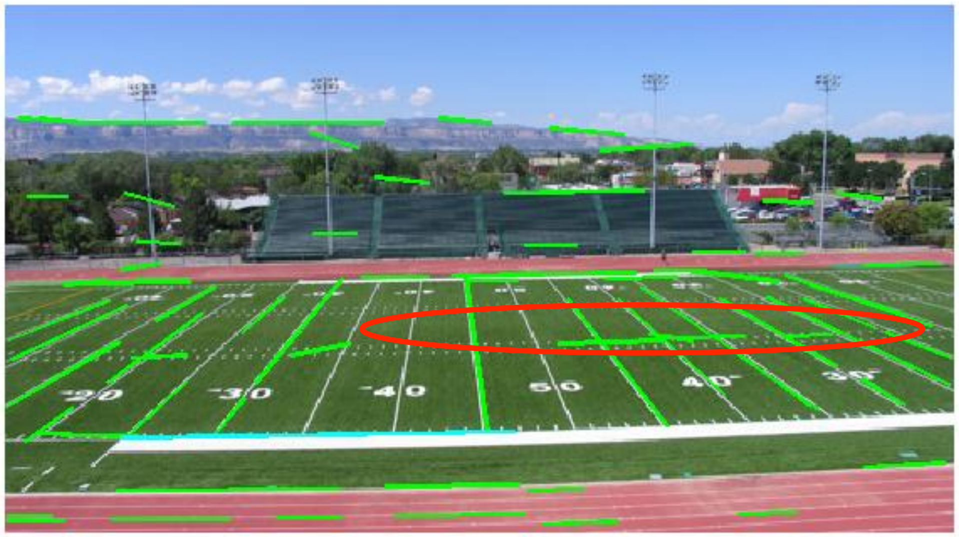
Vote space and top peaks



Kristen Grauman



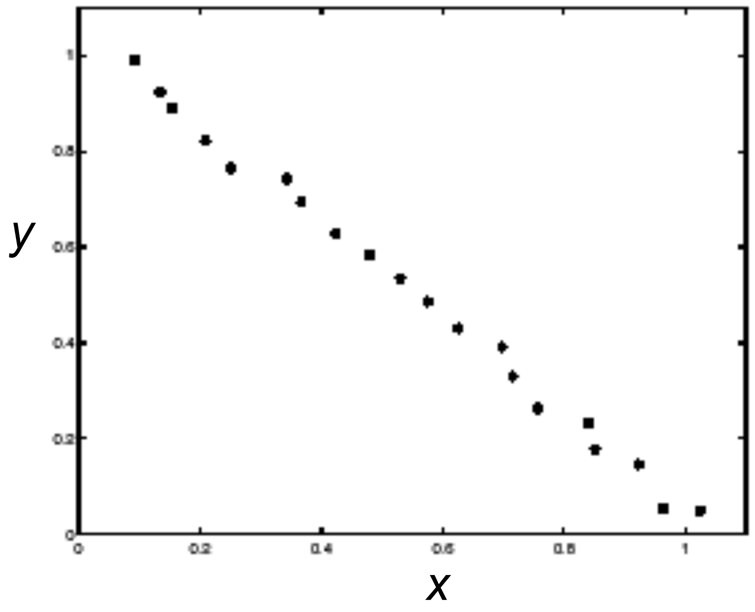




Showing longest segments found

Kristen Grauman

Impact of noise on Hough



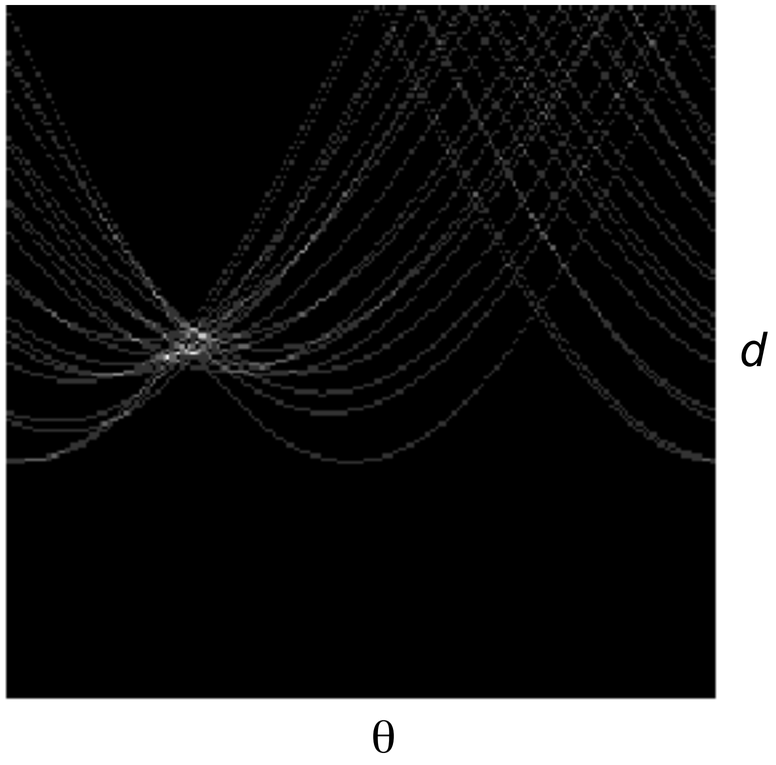
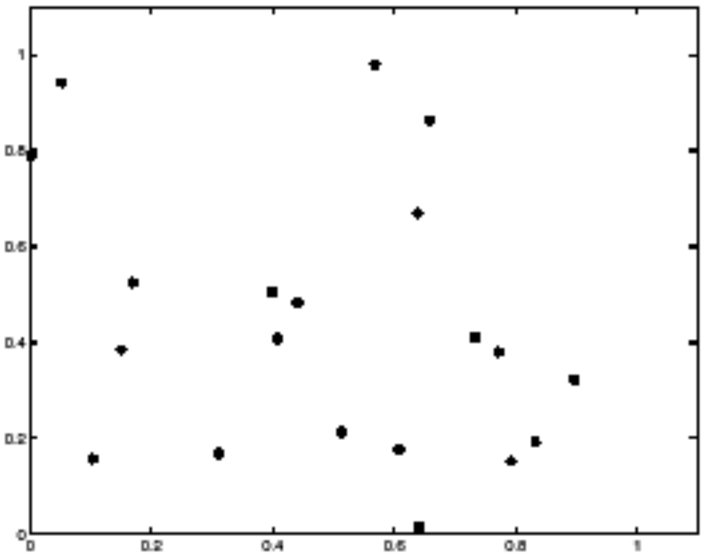


Image space edge coordinates

Votes

What difficulty does this present for an implementation?

Impact of noise on Hough



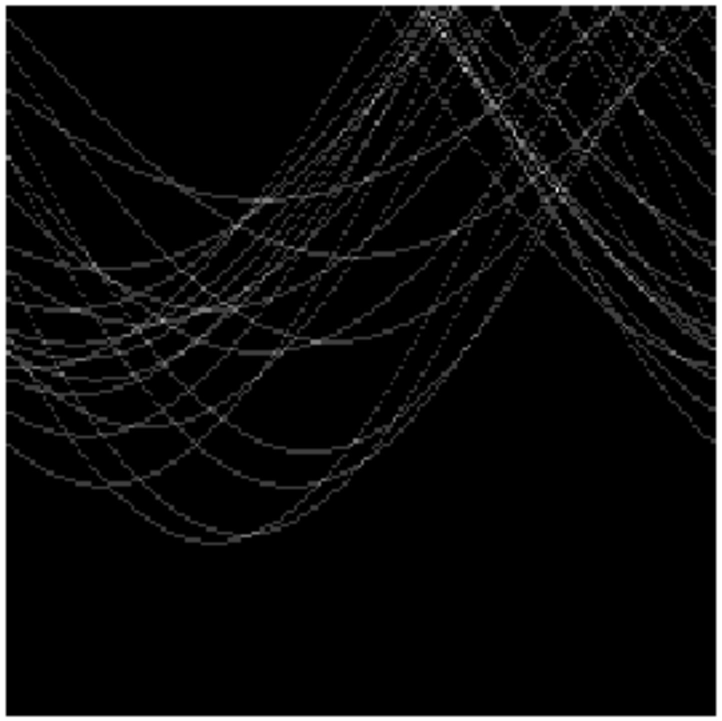


Image space
edge coordinates

Votes

Here, everything appears to be "noise", or random edge points, but we still see peaks in the vote space.

Extensions

Extension 1: Use the image gradient

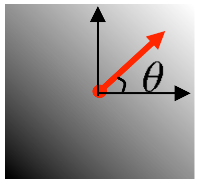
1. same
2. for each edge point I[x,y] in the image

   θ = gradient at (x,y)

   $d = x\cos\theta - y\sin\theta$

   H[d, θ] += 1
3. same
4. same

(Reduces degrees of freedom)



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

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

Extensions

Extension 1: Use the image gradient

```
1. same
2. for each edge point I[x,y] in the image
       compute unique (d, $\theta$) based on image gradient at (x,y)
       H[d, $\theta$] += 1
3. same
4. same
```

(Reduces degrees of freedom)

Extension 2

– give more votes for stronger edges (use magnitude of gradient)

Extension 3

– change the sampling of (d, θ) to give more/less resolution

Extension 4

– The same procedure can be used with circles, squares, or any other shape…

Source: Steve Seitz

Hough transform for circles

- Circle: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- For a fixed radius r, unknown gradient direction

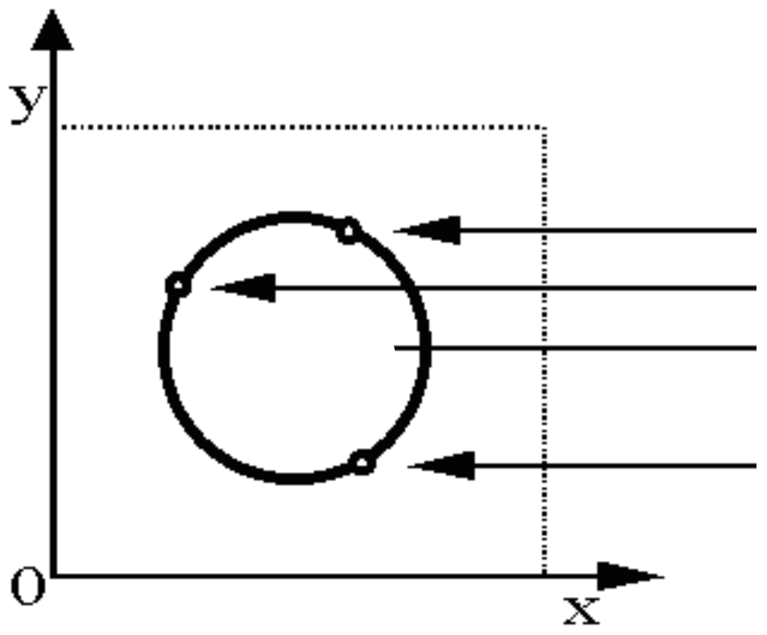
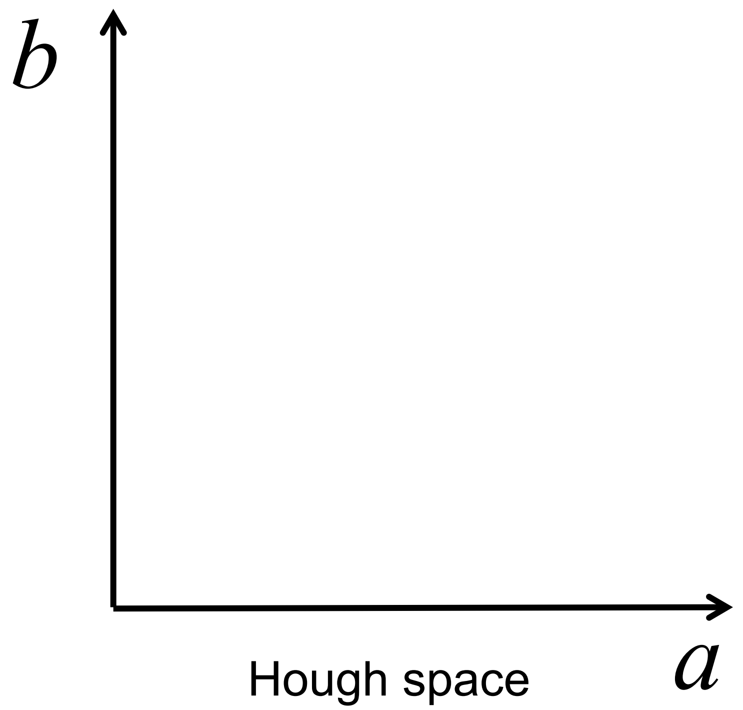


Image space



Hough space

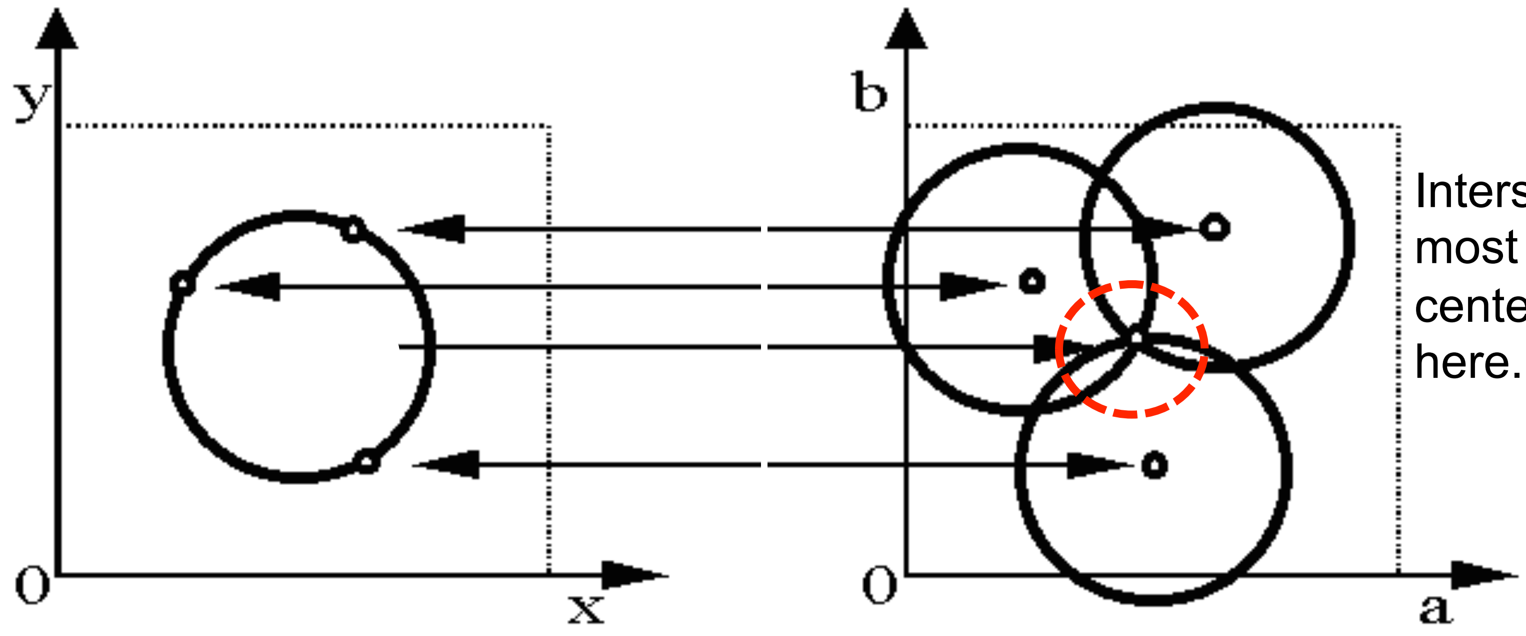
Kristen Grauman

Hough transform for circles

- Circle: center (a,b) and radius r

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

- For a fixed radius r, unknown gradient direction



Intersection: most votes for center occur here.

Image space

Hough space

Kristen Grauman

Hough Extensions for Other Constraints

- In principle, any constraint of the form f(x, d) = 0 can be checked
  - Grid the parameter (x) space
  - For each observed d and grid element x_i if $|f(x_i, d) < t|$ add a vote
  - Continue with the rest of the algorithm as before

http://www.youtube.com/watch?v=NXWjmPuwVnI

Hough: summary

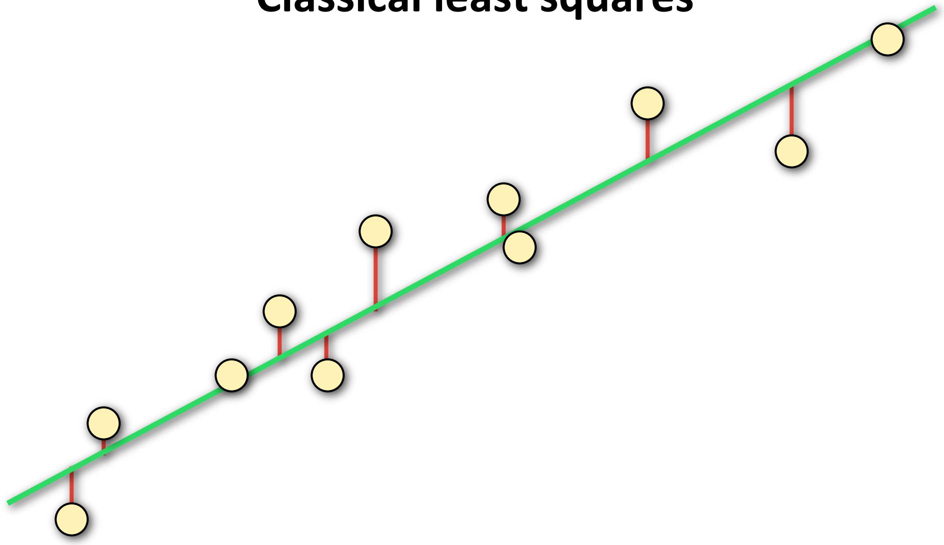
- Pros:
  - All points are processed independently, so can cope with occlusion
  - Some robustness to noise: noise points unlikely to contribute consistently to any single bin
  - Can detect multiple instances of a model in a single pass
- Cons:
  - Complexity of search time increases exponentially with the number of model parameters
  - Non-target shapes can produce spurious peaks in parameter space
  - Quantization: hard to pick a good grid size

Lines - RANSAC

Reminder: least squares line fitting

- we know how to fit a line to 2 or more points

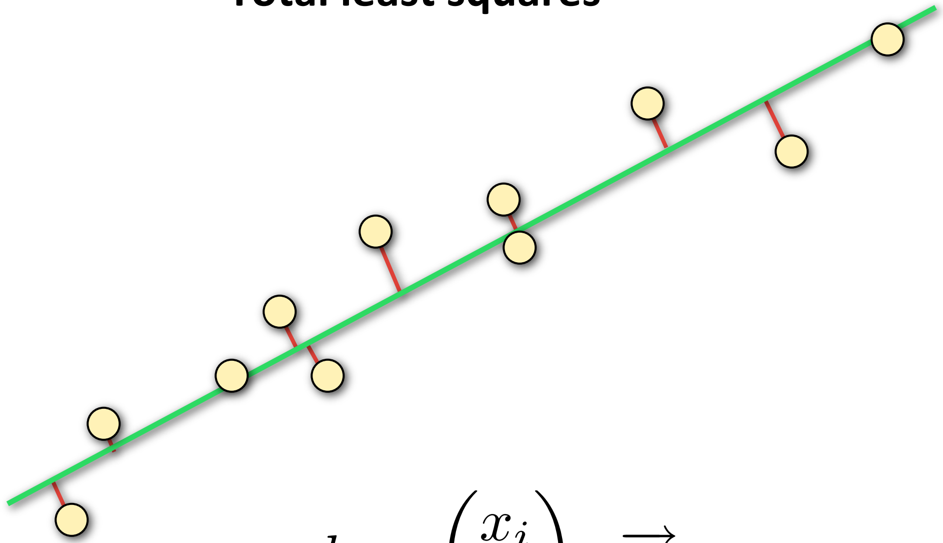
Classical least squares



$$d_i = y_i - (ax_i + b)$$

$$e = \sum_i d_i^2 = \sum_i (y_i - (ax_i + b))^2$$

Total least squares

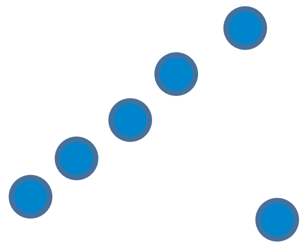


$$d_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \cdot \vec{n} - a$$

$$e = \sum_i d_i^2 = \sum_i \left[\begin{pmatrix} x_i \\ y_i \end{pmatrix} \cdot \vec{n} - a \right]^2$$

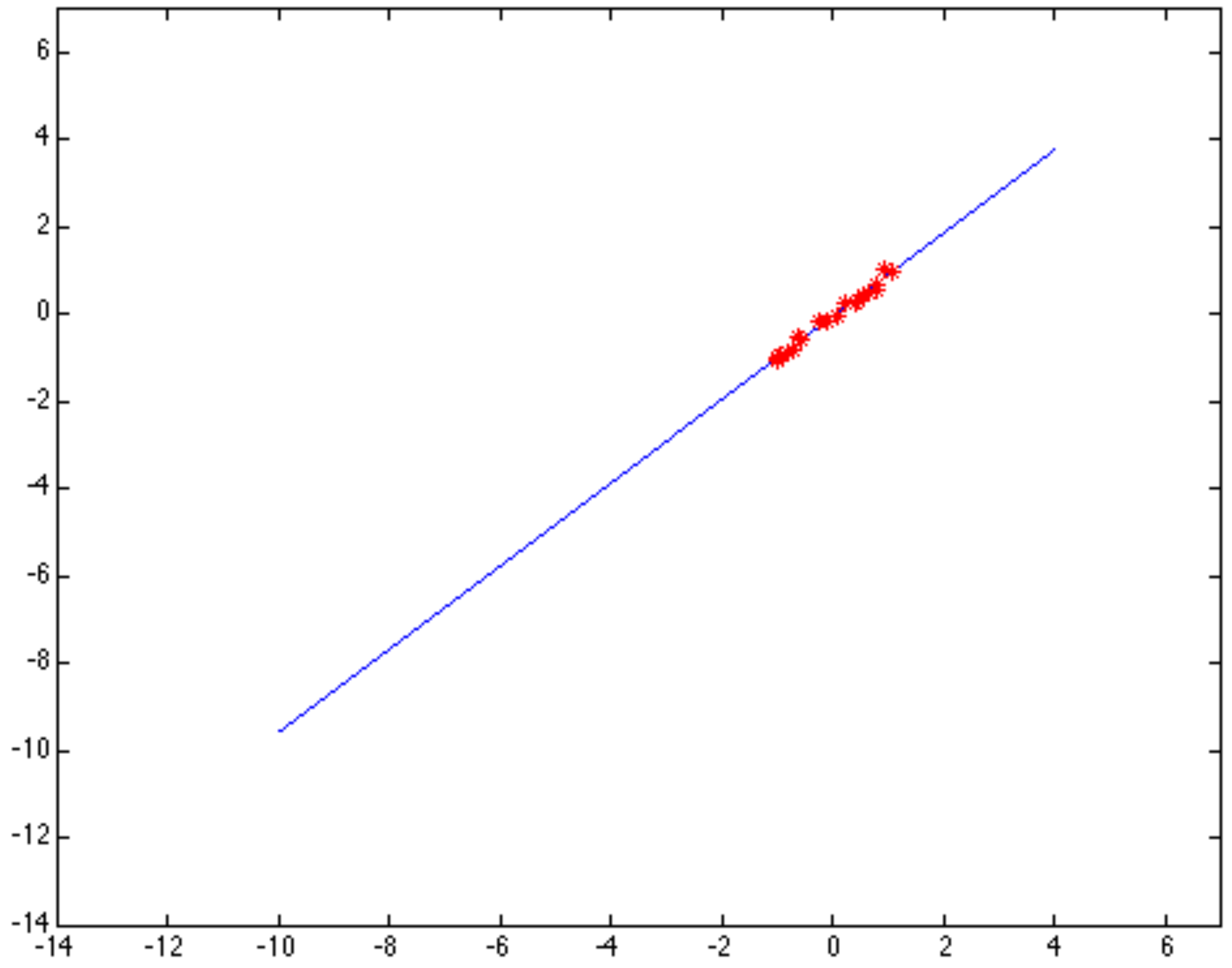
Outliers

- **Outliers** can hurt the quality of our parameter estimates, e.g. an edge point that is noise, or doesn't belong to the line we are fitting.

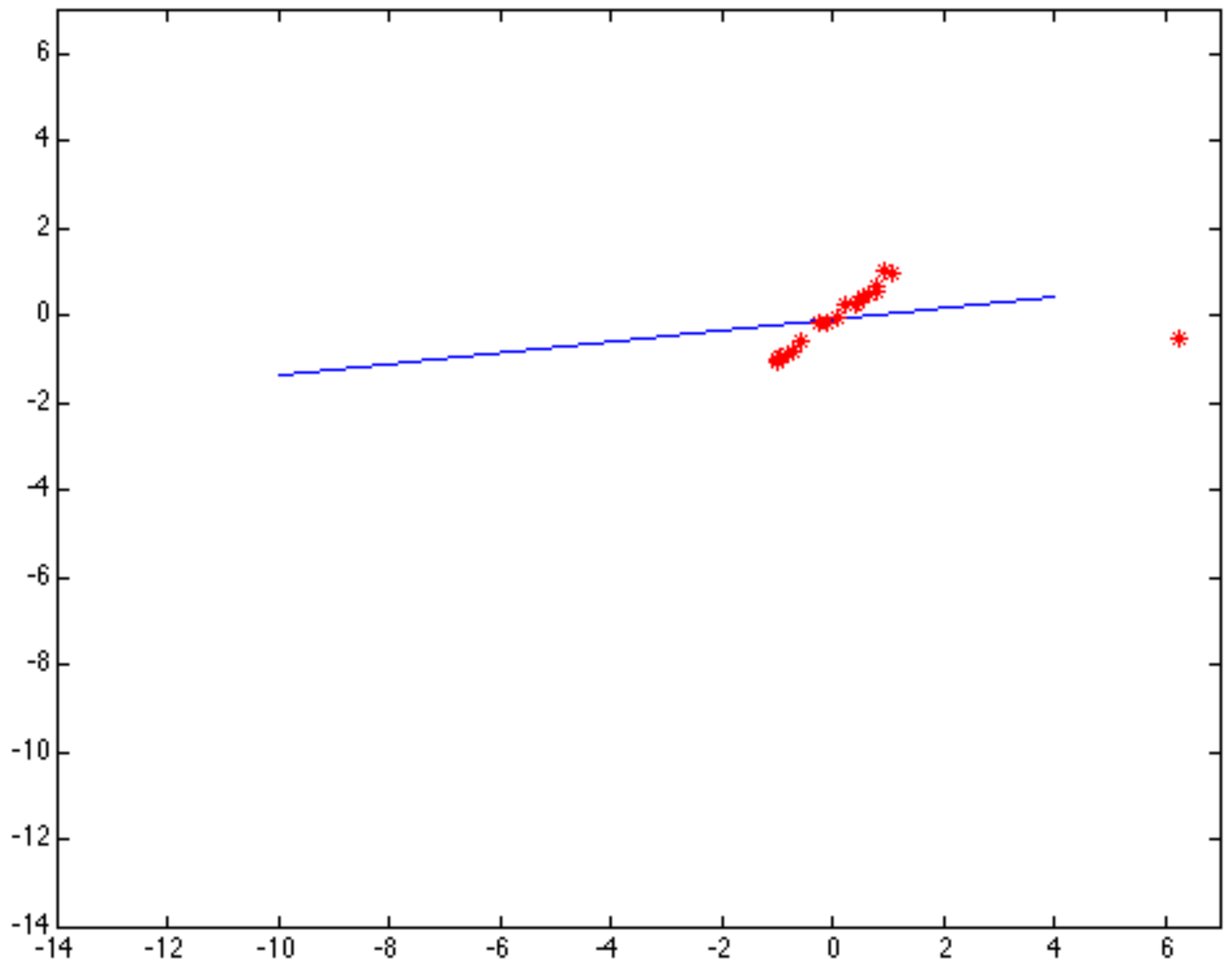


Kristen Grauman

Outliers affect least squares fit



Outliers affect least squares fit



RANSAC

- RANdom Sample Consensus
- **Approach**: we want to avoid the impact of outliers, so let's look for "inliers", and use those only.
- **Intuition**: if an outlier is chosen to compute the current fit, then the resulting line won't have much support from rest of the points.

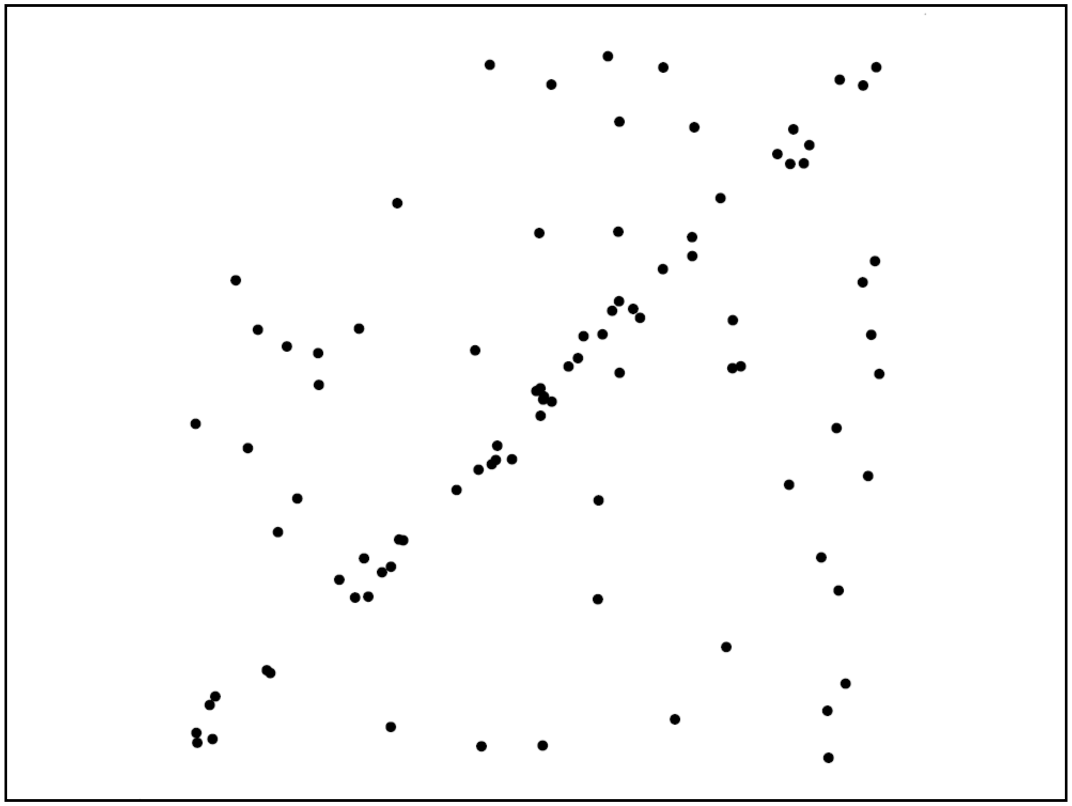
RANSAC: General form

- RANSAC loop:

1. Randomly select a *seed group* of points on which to base transformation estimate (e.g., a group of matches)
2. Compute transformation from seed group
3. Find *inliers* to this transformation
4. If the number of inliers is sufficiently large, re-compute estimate of transformation on all of the inliers

- Keep the transformation with the largest number of inliers

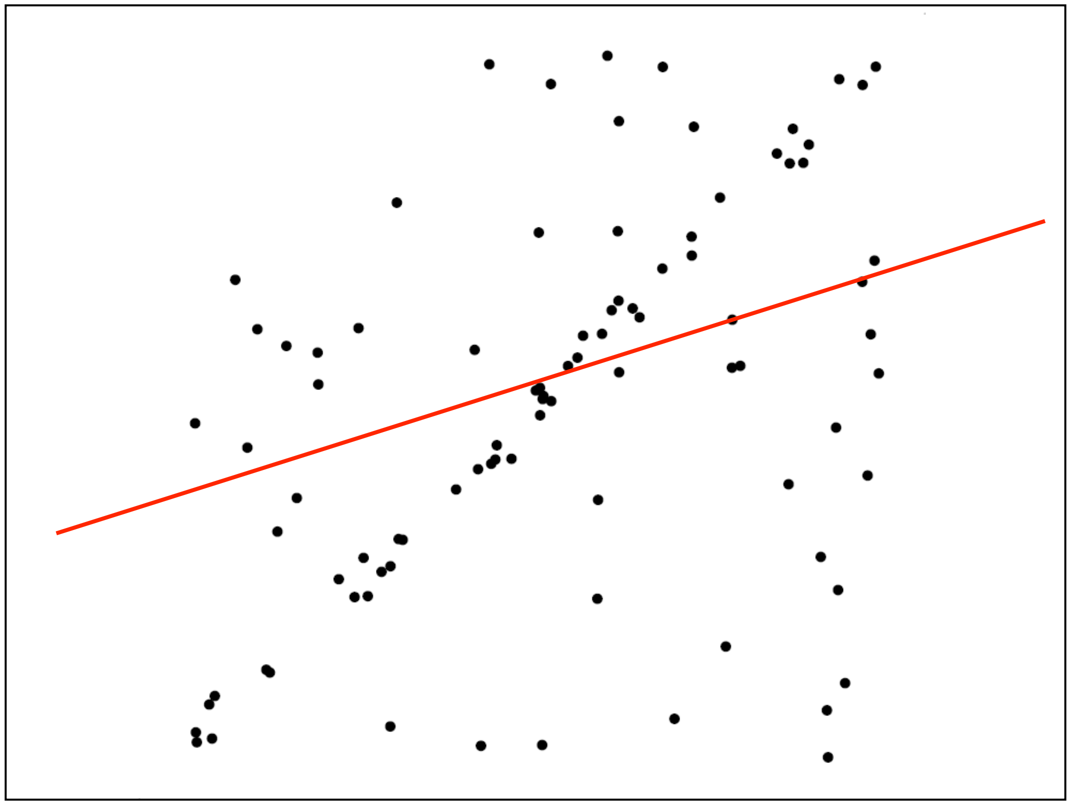
RANSAC for line fitting example



Source: R. Raguram

Lana Lazebnik

RANSAC for line fitting example

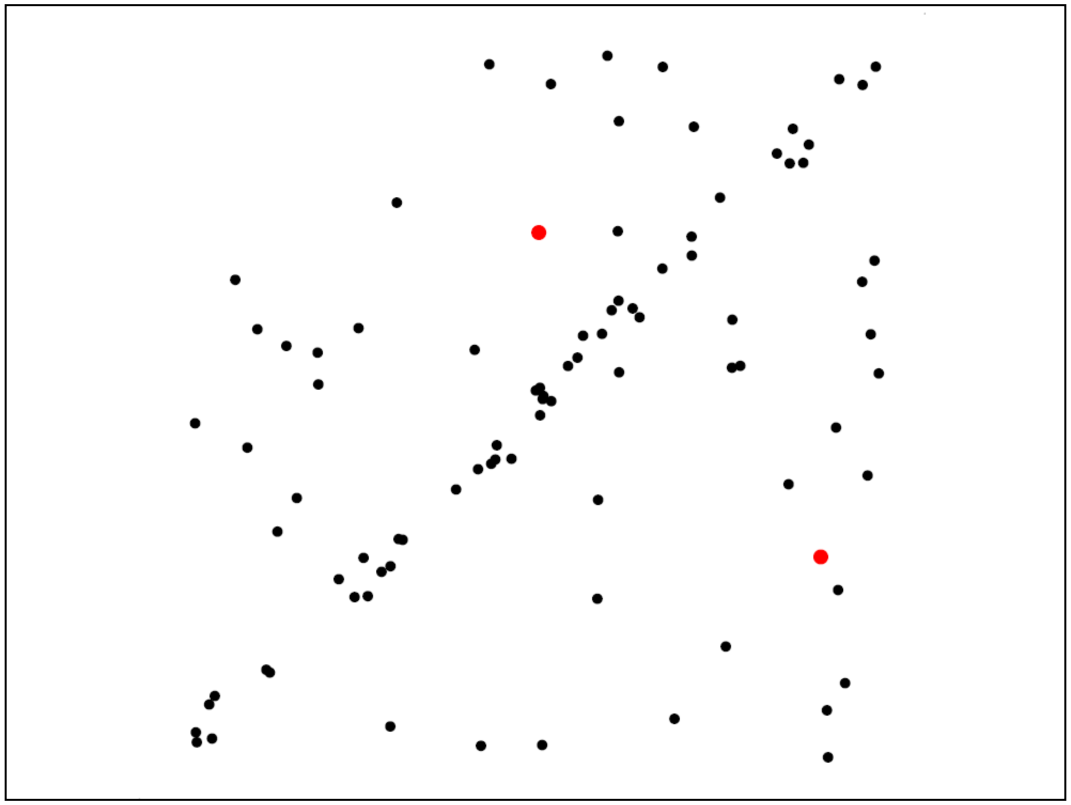


Least-squares fit

Source: R. Raguram

Lana Lazebnik

RANSAC for line fitting example

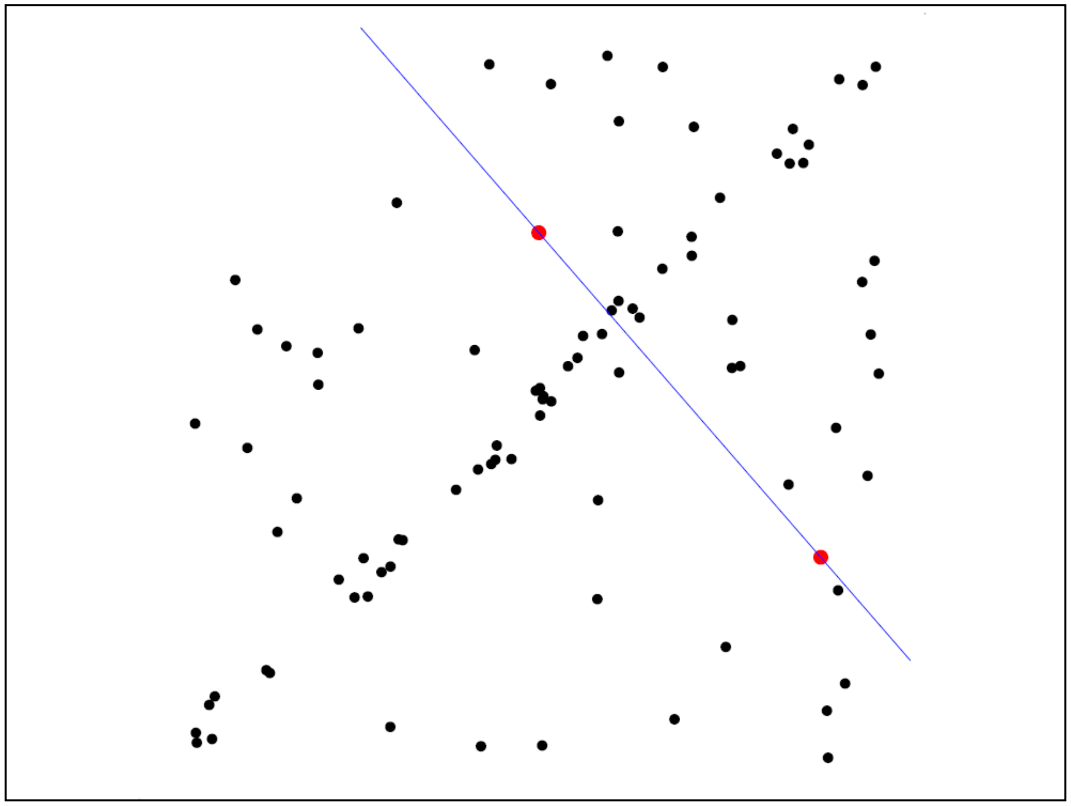


1. Randomly select minimal subset of points

Source: R. Raguram

Lana Lazebnik

RANSAC for line fitting example

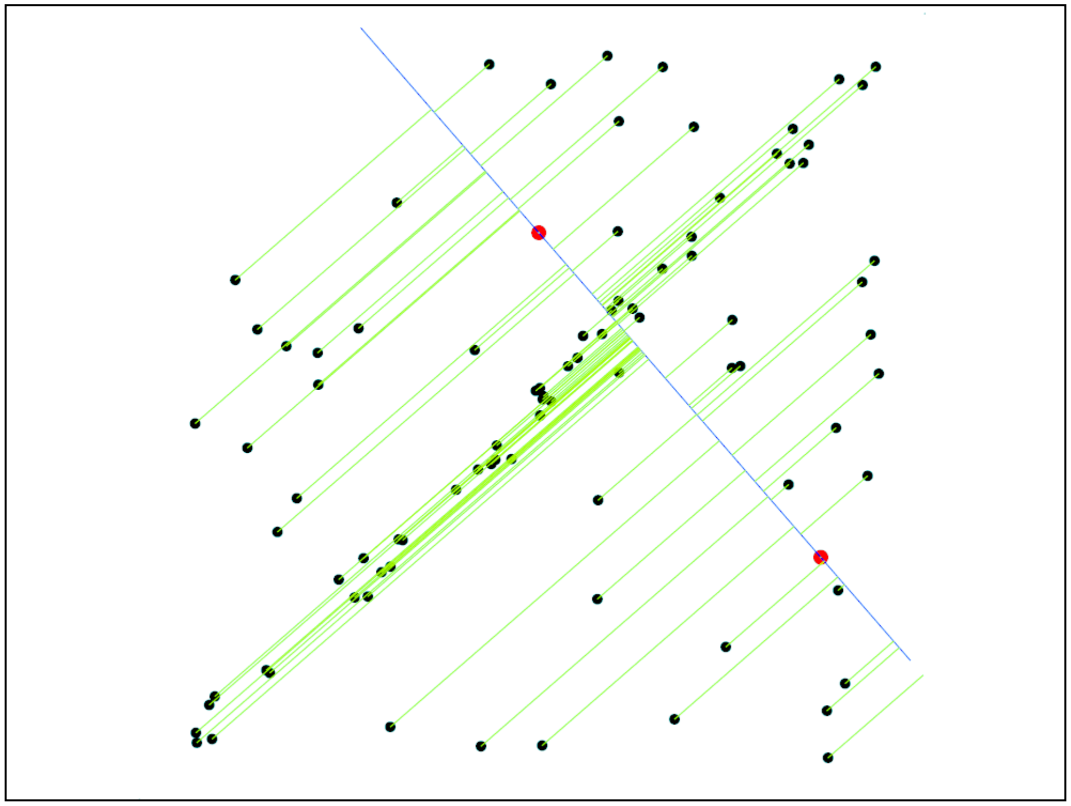


1. Randomly select minimal subset of points
2. Hypothesize a model

Source: R. Raguram

Lana Lazebnik

RANSAC for line fitting example

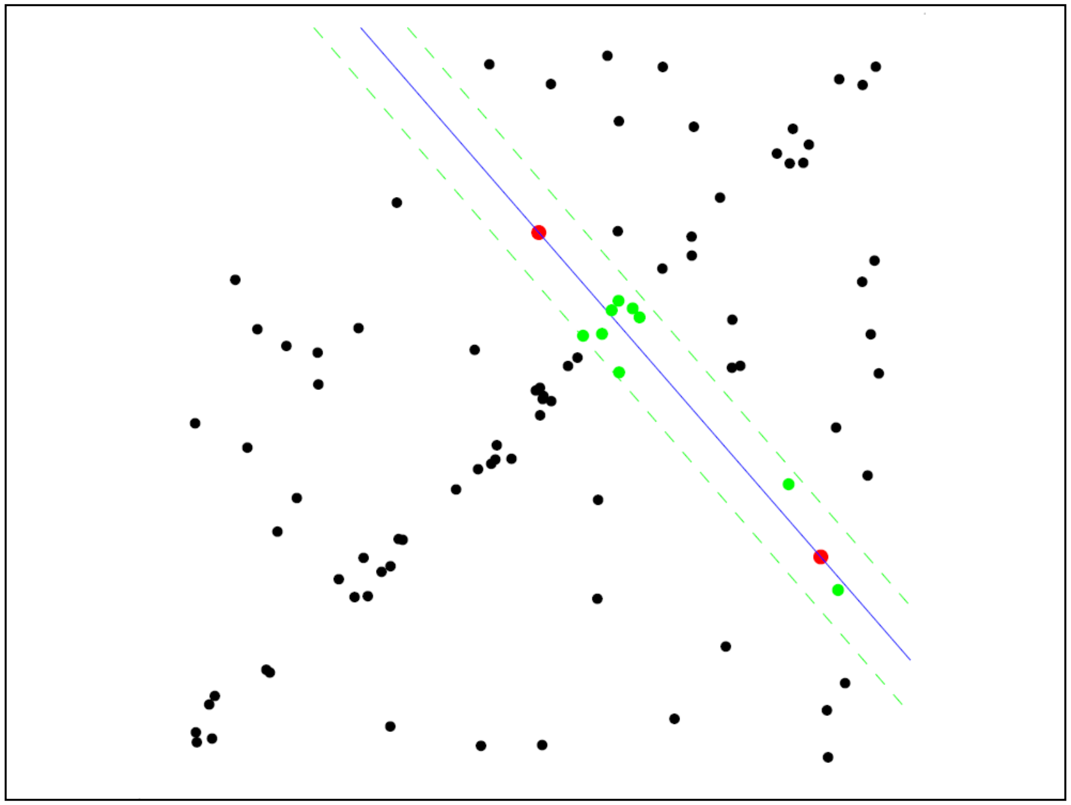


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function

Source: R. Raguram

Lana Lazebnik

RANSAC for line fitting example

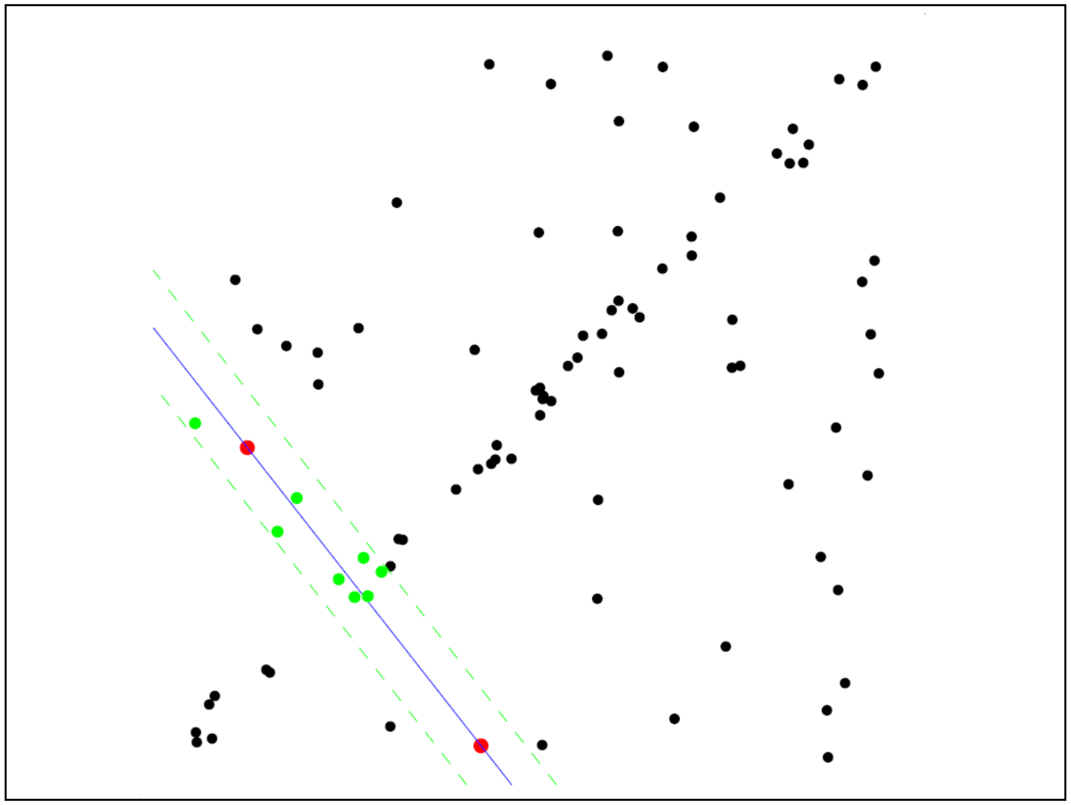


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model

Source: R. Raguram

Lana Lazebnik

RANSAC for line fitting example

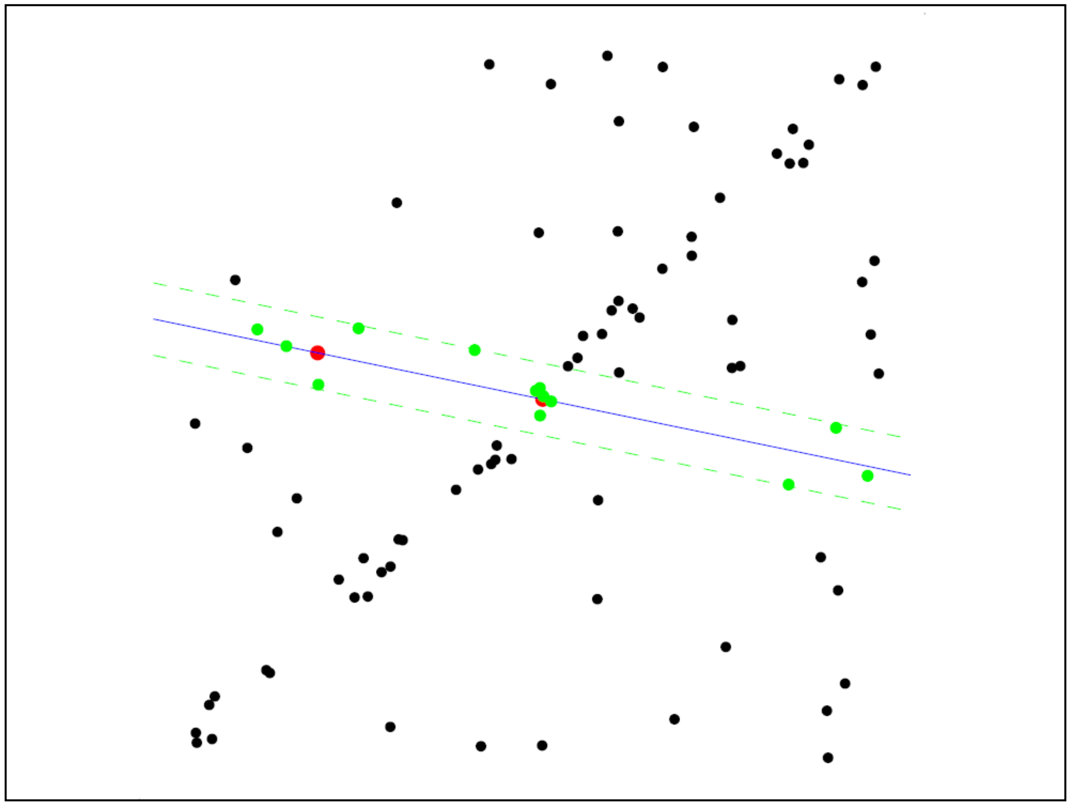


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

Lana Lazebnik

RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

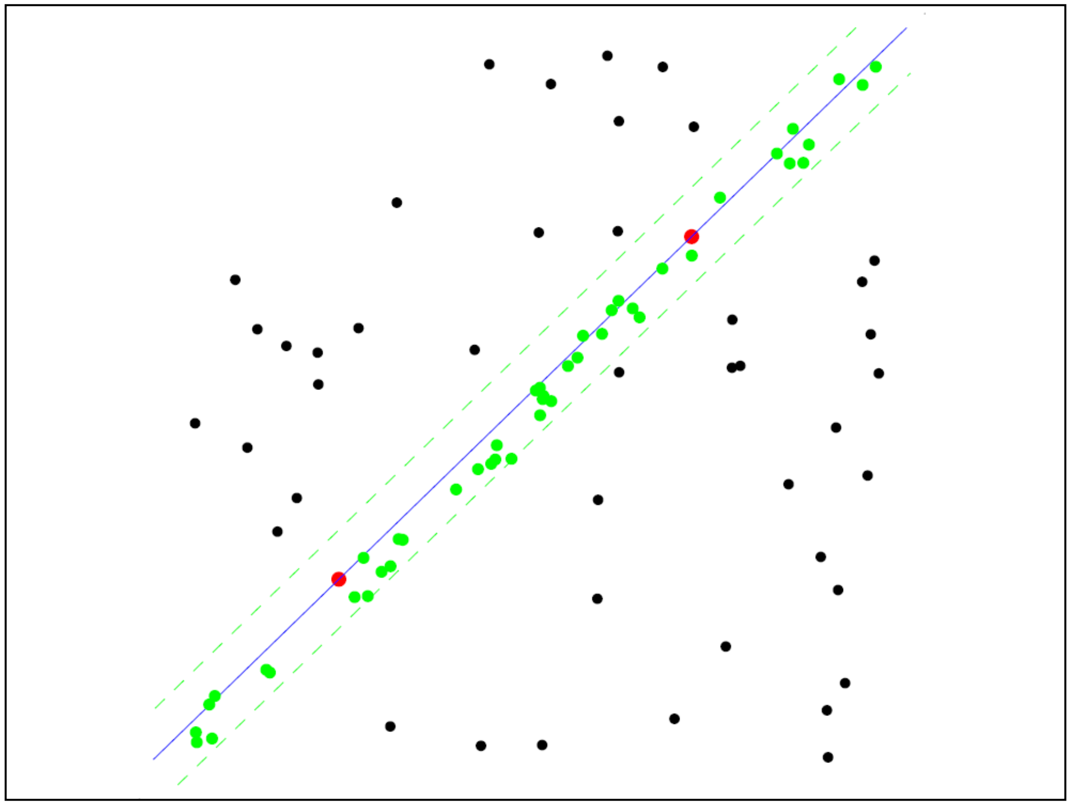
81

Source: R. Raguram

Lana Lazebnik

RANSAC for line fitting example

Uncontaminated sample

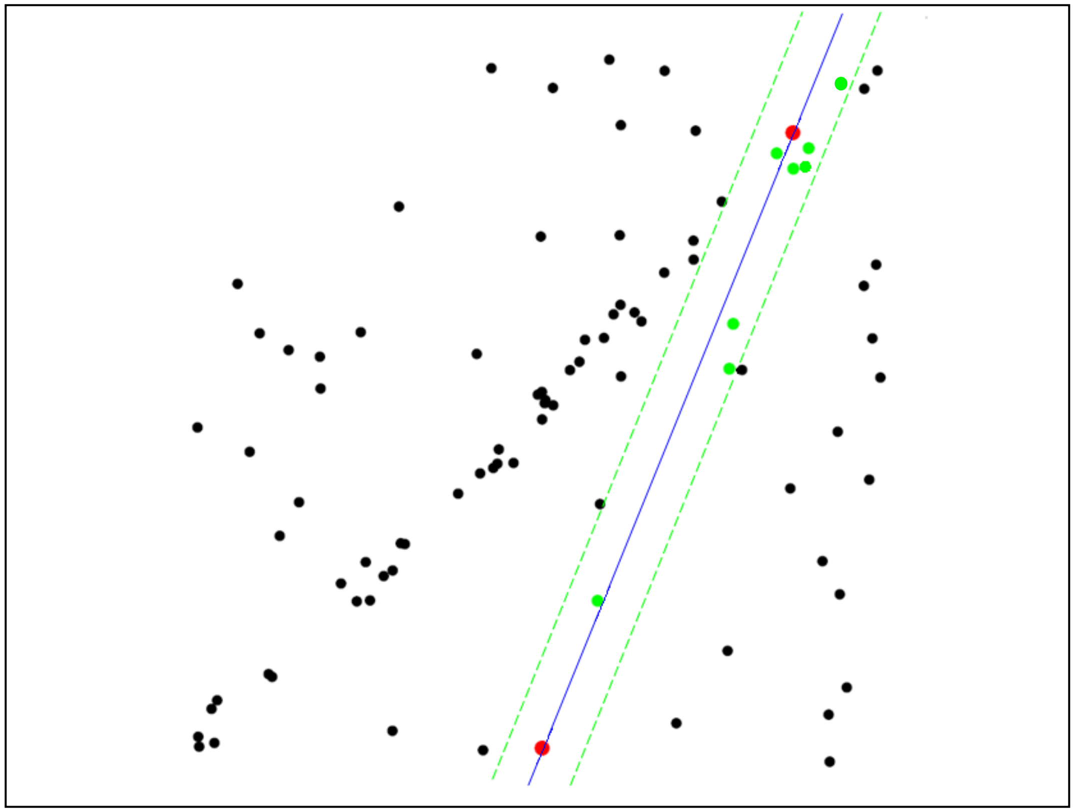


1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

Lana Lazebnik

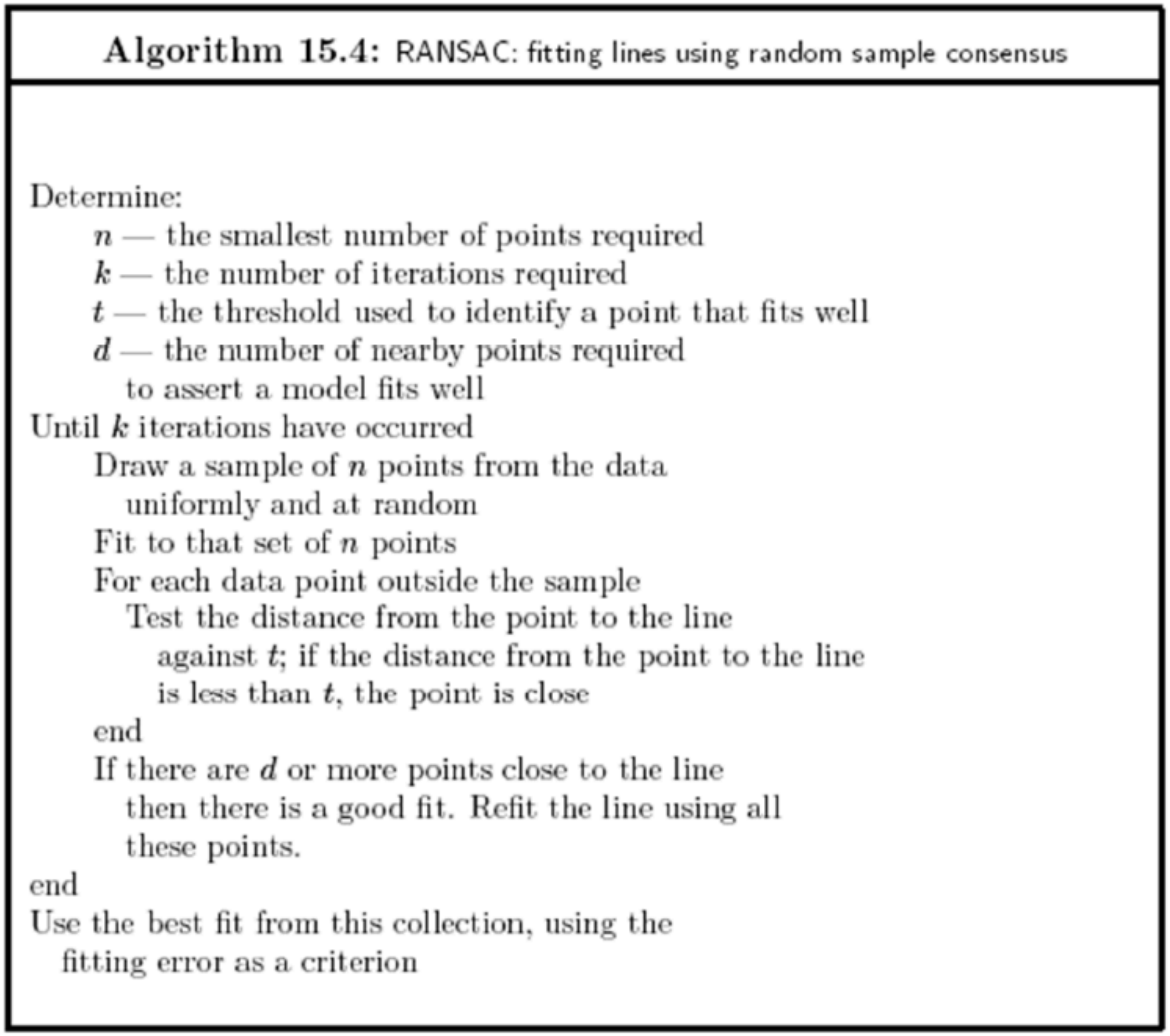
RANSAC for line fitting example



1. Randomly select minimal subset of points
2. Hypothesize a model
3. Compute error function
4. Select points consistent with model
5. Repeat *hypothesize-and-verify* loop

Source: R. Raguram

Lana Lazebnik



Algorithm 15.4: RANSAC: fitting lines using random sample consensus

```
Determine:
    $n$ — the smallest number of points required
    $k$ — the number of iterations required
    $t$ — the threshold used to identify a point that fits well
    $d$ — the number of nearby points required
      to assert a model fits well
Until $k$ iterations have occurred
    Draw a sample of $n$ points from the data
      uniformly and at random
    Fit to that set of $n$ points
    For each data point outside the sample
      Test the distance from the point to the line
        against $t$; if the distance from the point to the line
        is less than $t$, the point is close
    end
    If there are $d$ or more points close to the line
      then there is a good fit. Refit the line using all
      these points.
end
Use the best fit from this collection, using the
  fitting error as a criterion
```

RANSAC: how many samples ?

- How many samples are needed?
  - Suppose w is fraction of inliers (points from line).
  - n points needed to define hypothesis (2 for lines)
  - k samples chosen.
- Prob. that a single sample of n points is correct: w^n
- Prob. that all k samples fail is: $(1-w^n)^k$

$\Rightarrow$ Choose k high enough to keep this below desired failure rate.

Slide credit: David Lowe

RANSAC: computed k for p=0.99

Sample size n	Proportion of outliers						
	5%	10%	20%	25%	30%	40%	50%
2	2	3	5	6	7	11	17
3	3	4	7	9	11	19	35
4	3	5	9	13	17	34	72
5	4	6	12	17	26	57	146
6	4	7	16	24	37	97	293
7	4	8	20	33	54	163	588
8	5	9	26	44	78	272	1177

Slide credit: David Lowe

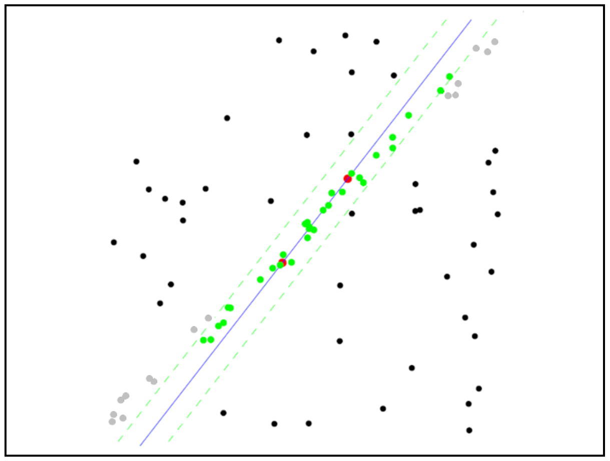
After RANSAC

- RANSAC divides data into inliers and outliers and yields estimate computed from minimal set of inliers
- Improve this initial estimate with estimation over all inliers (e.g. with standard least-squares minimization)
- But this may change inliers, so alternate fitting with re-classification as inlier/outlier

Slide credit: David Lowe

RANSAC pros and cons

- Pros
  - Simple and general
  - Applicable to many different problems
  - Often works well in practice
- Cons
  - Lots of parameters to tune
  - Doesn't work well for low inlier ratios (too many iterations, or can fail completely)



Lana Lazebnik

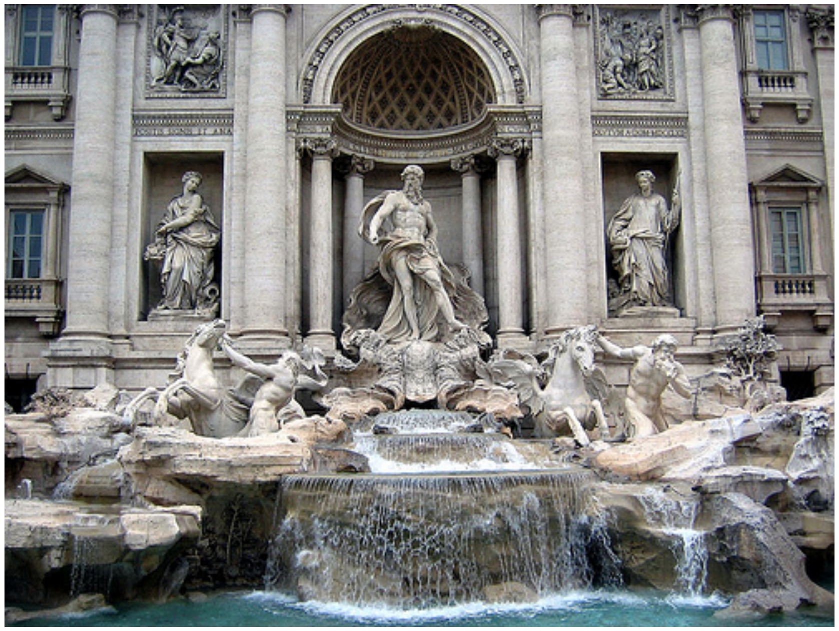
Conclusion

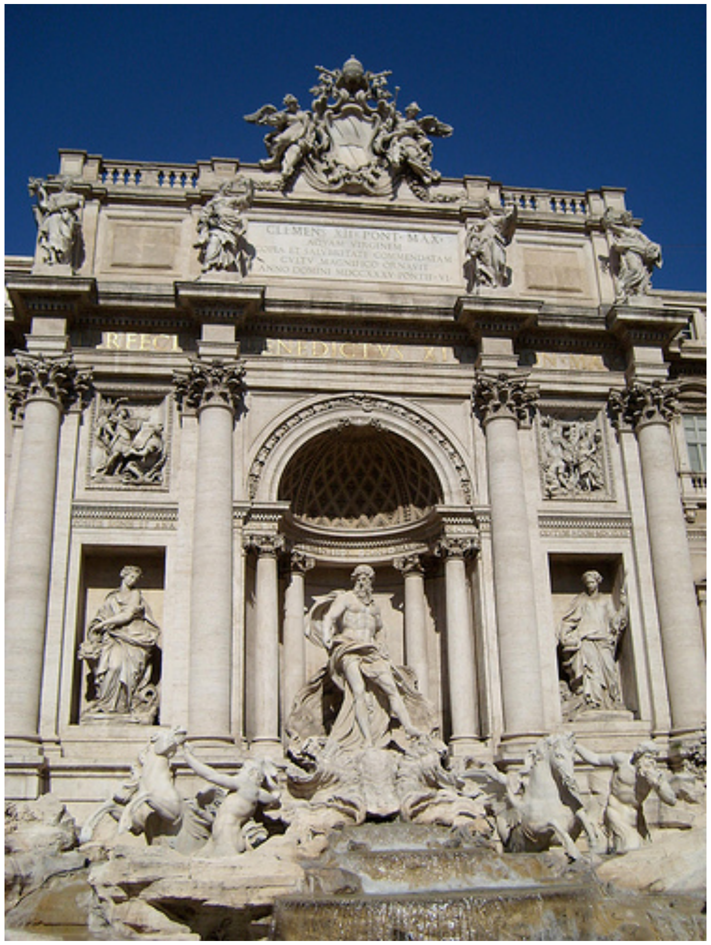
- So Far
  - Edges
    - Gradient operators
    - Canny edge detector
  - Lines
    - Hough transform
    - RANSAC

- What is next ?
  - Corners
  - SIFT features

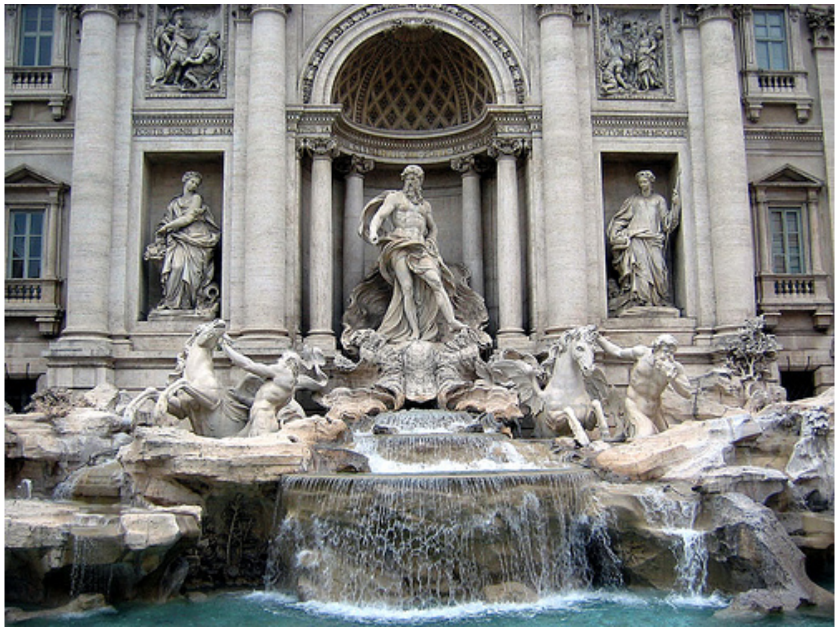
Corners

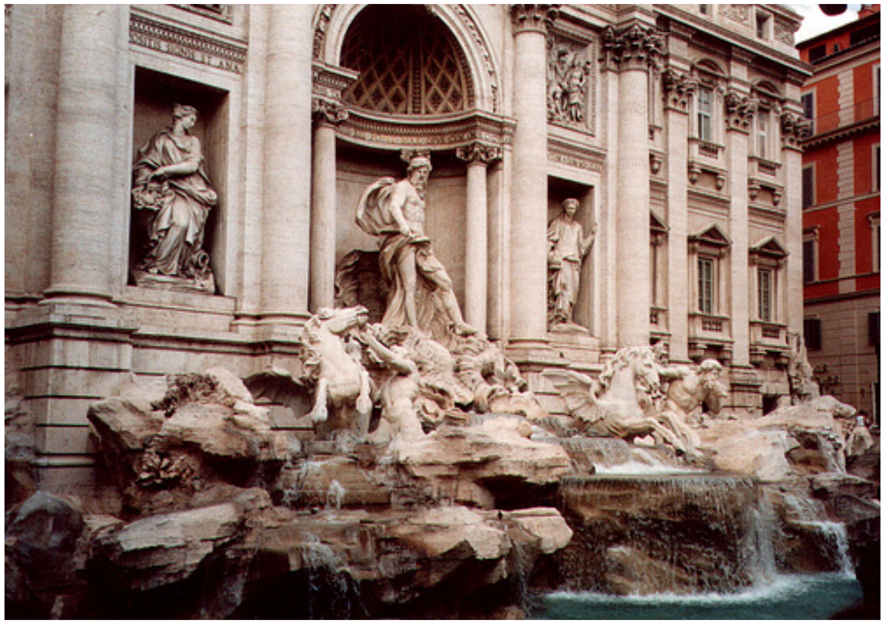
Image matching


by Diva Sian

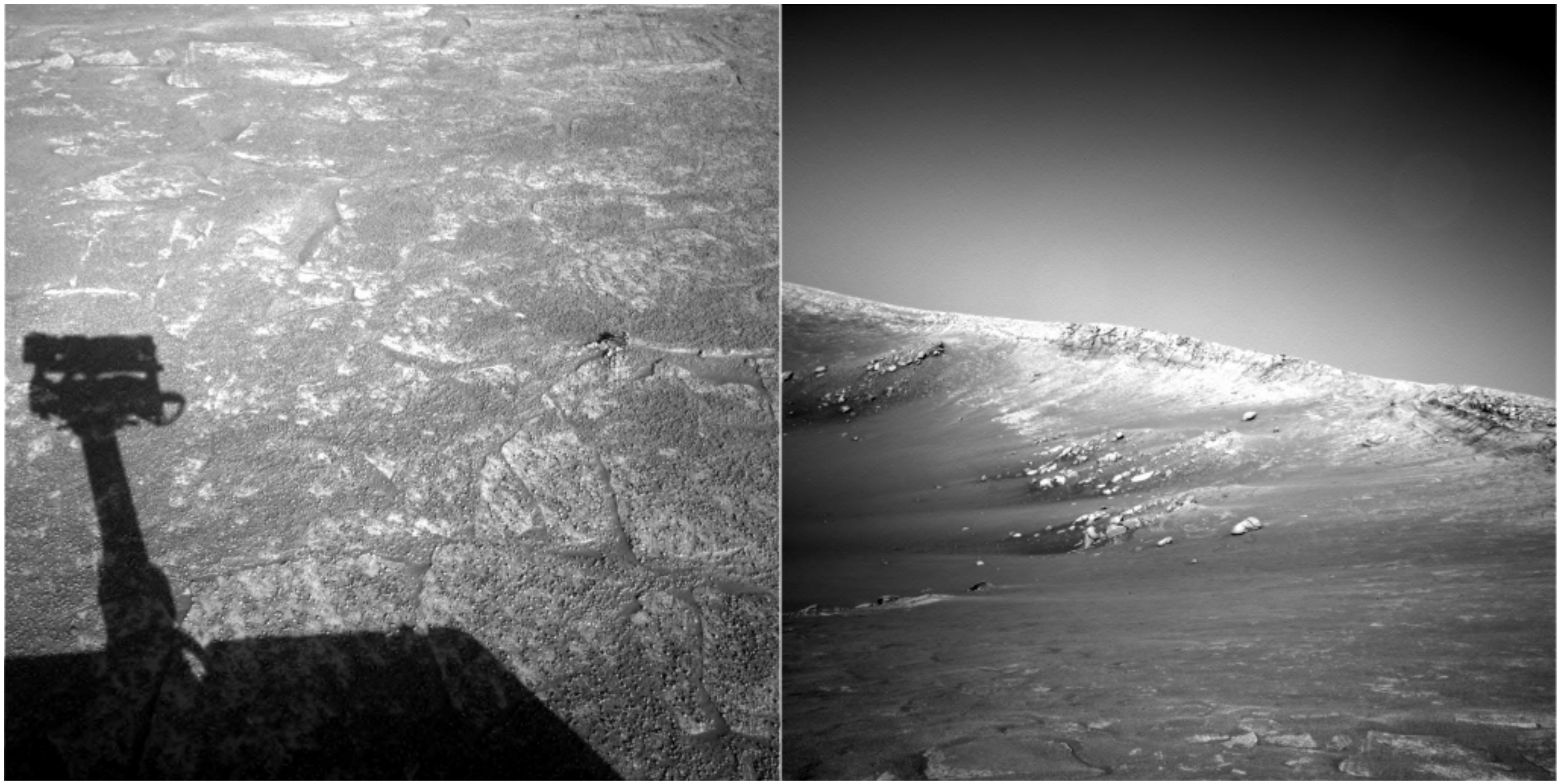

by swashford

Harder case

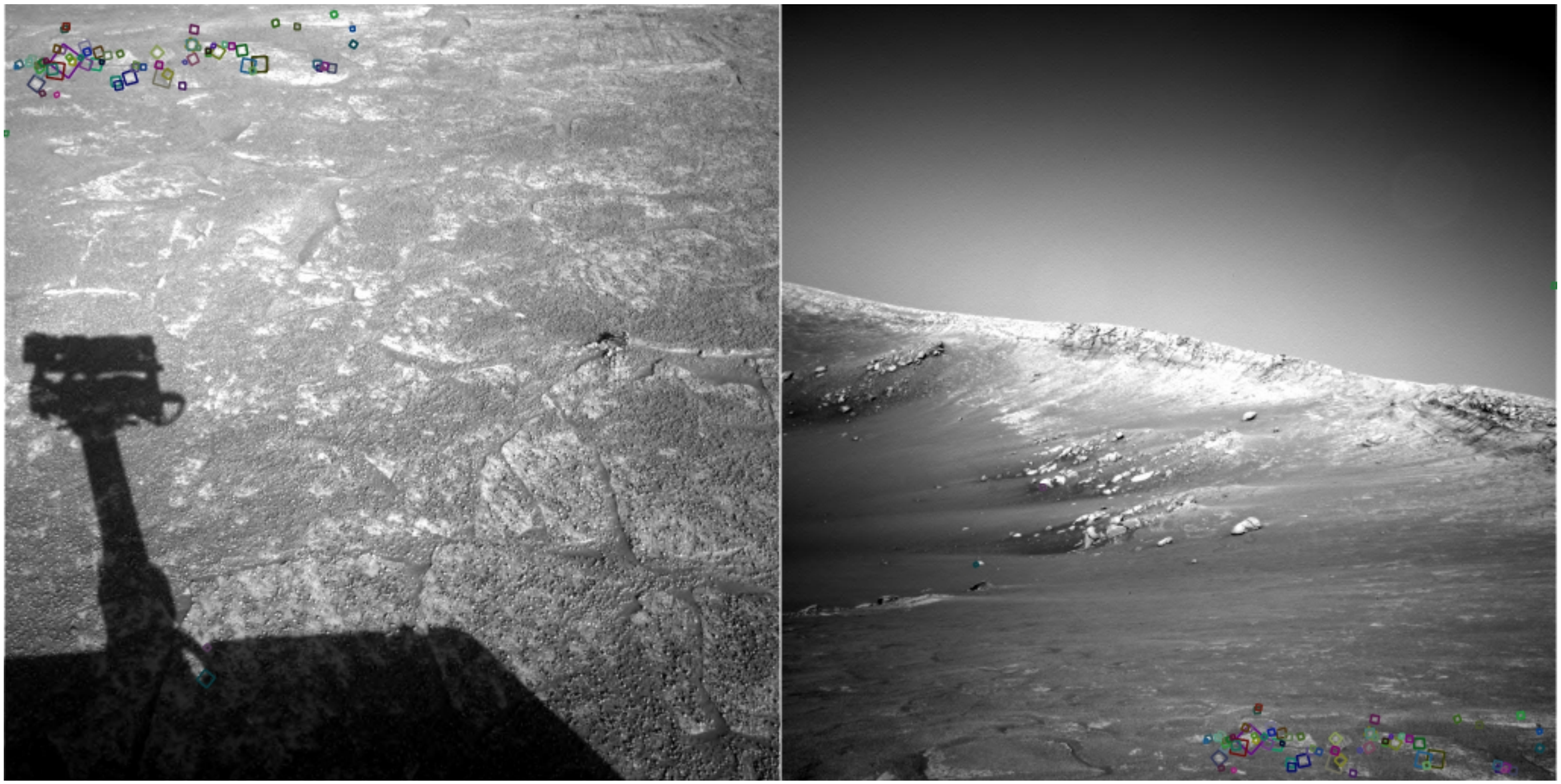

by Diva Sian


by scgbt

Harder still?

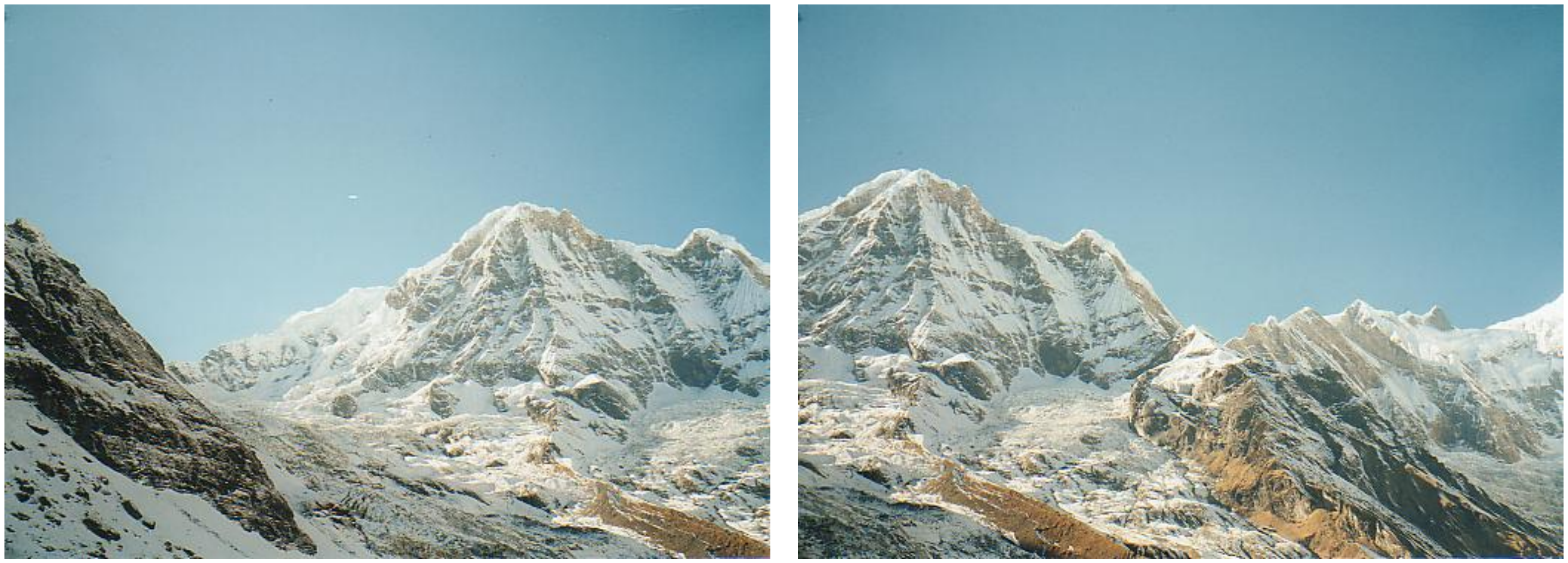

NASA Mars Rover images

Answer below (look for tiny colored squares…)



NASA Mars Rover images
with SIFT feature matches
Figure by Noah Snavely

Local features and alignment

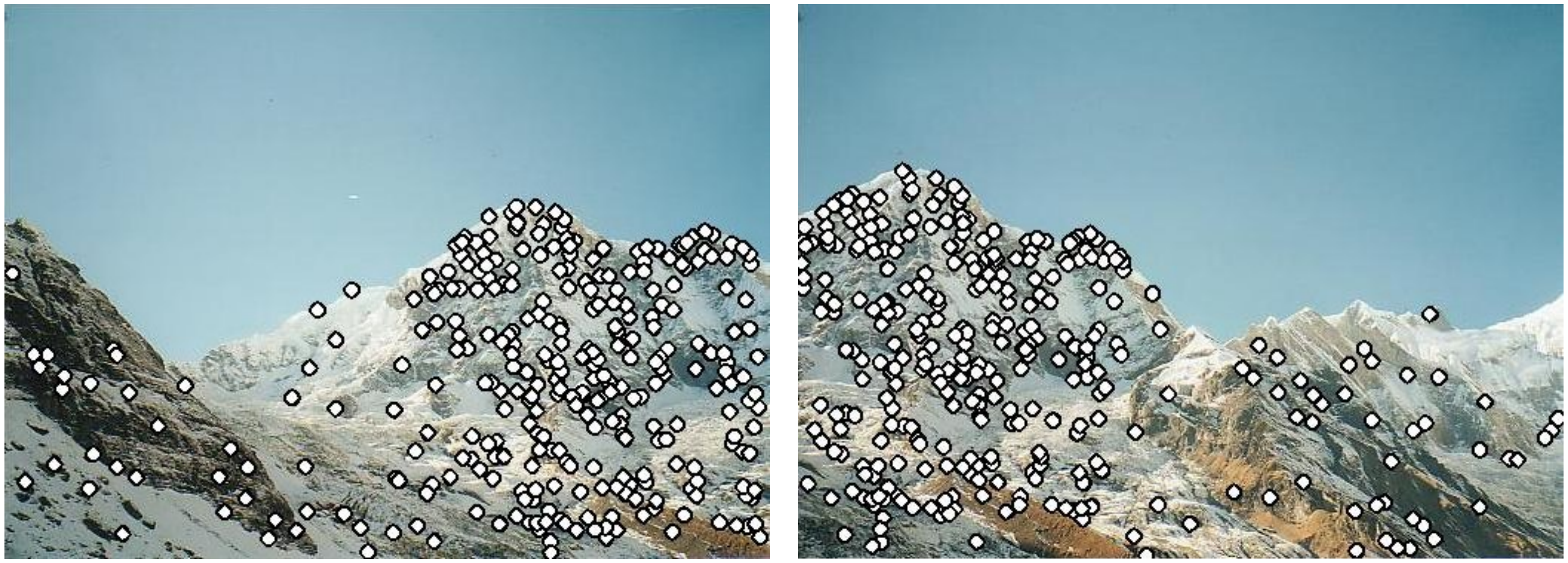


- We need to match (align) images
- How would you do it by eye?

[Darya Frolova and Denis Simakov]

Local features and alignment

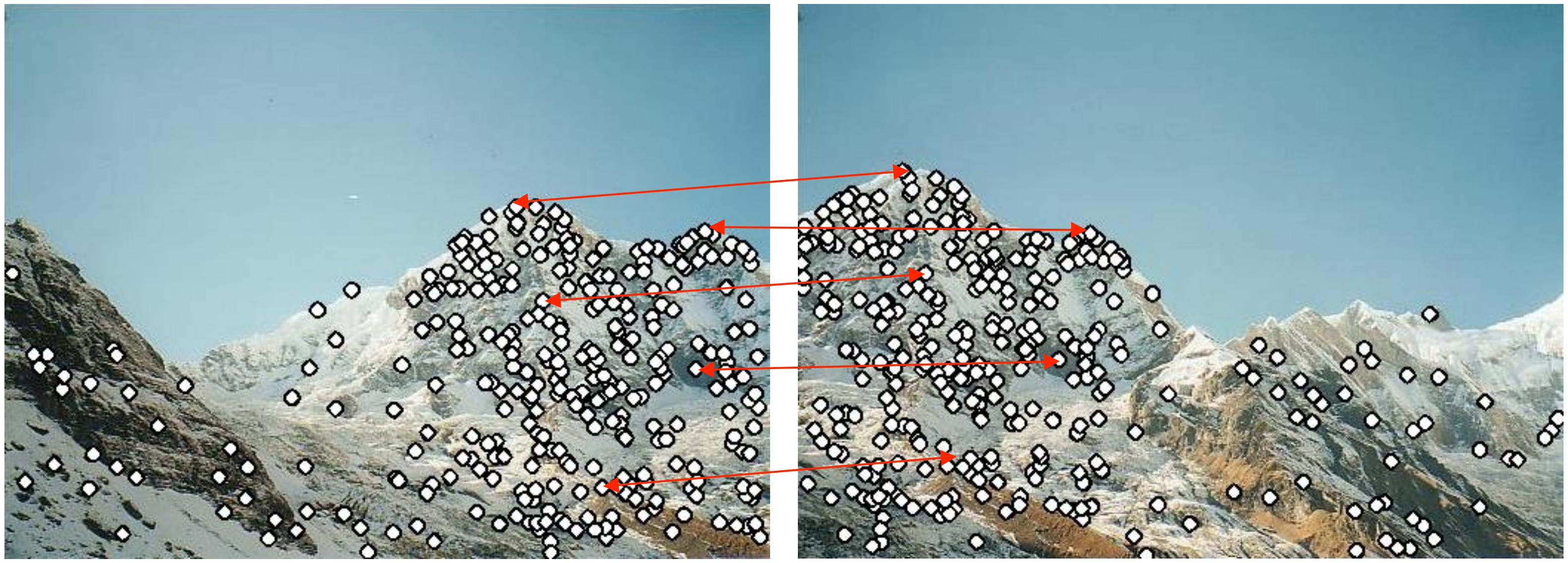
- Detect feature points in both images



[Darya Frolova and Denis Simakov]

Local features and alignment

- Detect feature points in both images
- Find corresponding pairs



[Darya Frolova and Denis Simakov]

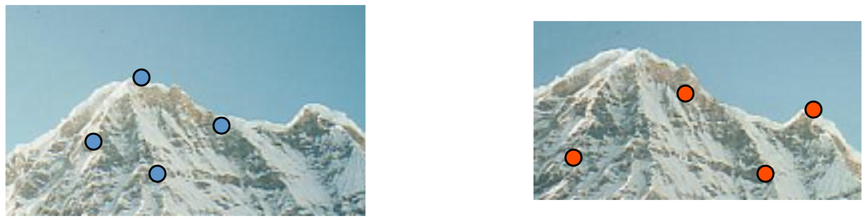
Local features and alignment

- Detect feature points in both images
- Find corresponding pairs
- Use these pairs to align images

[Darya Frolova and Denis Simakov]

Local features and alignment

- Problem 1:
  - Detect the *same* point *independently* in both images



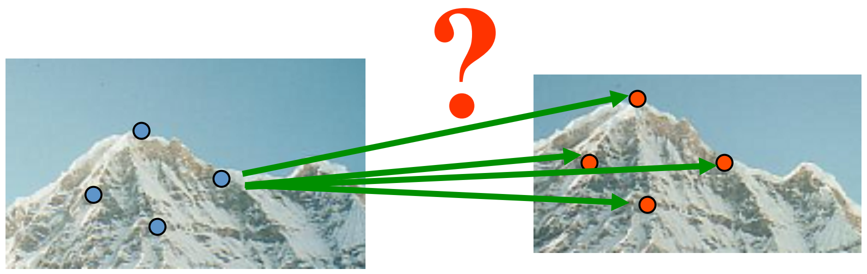
no chance to match!

We need a repeatable detector

[Darya Frolova and Denis Simakov]

Local features and alignment

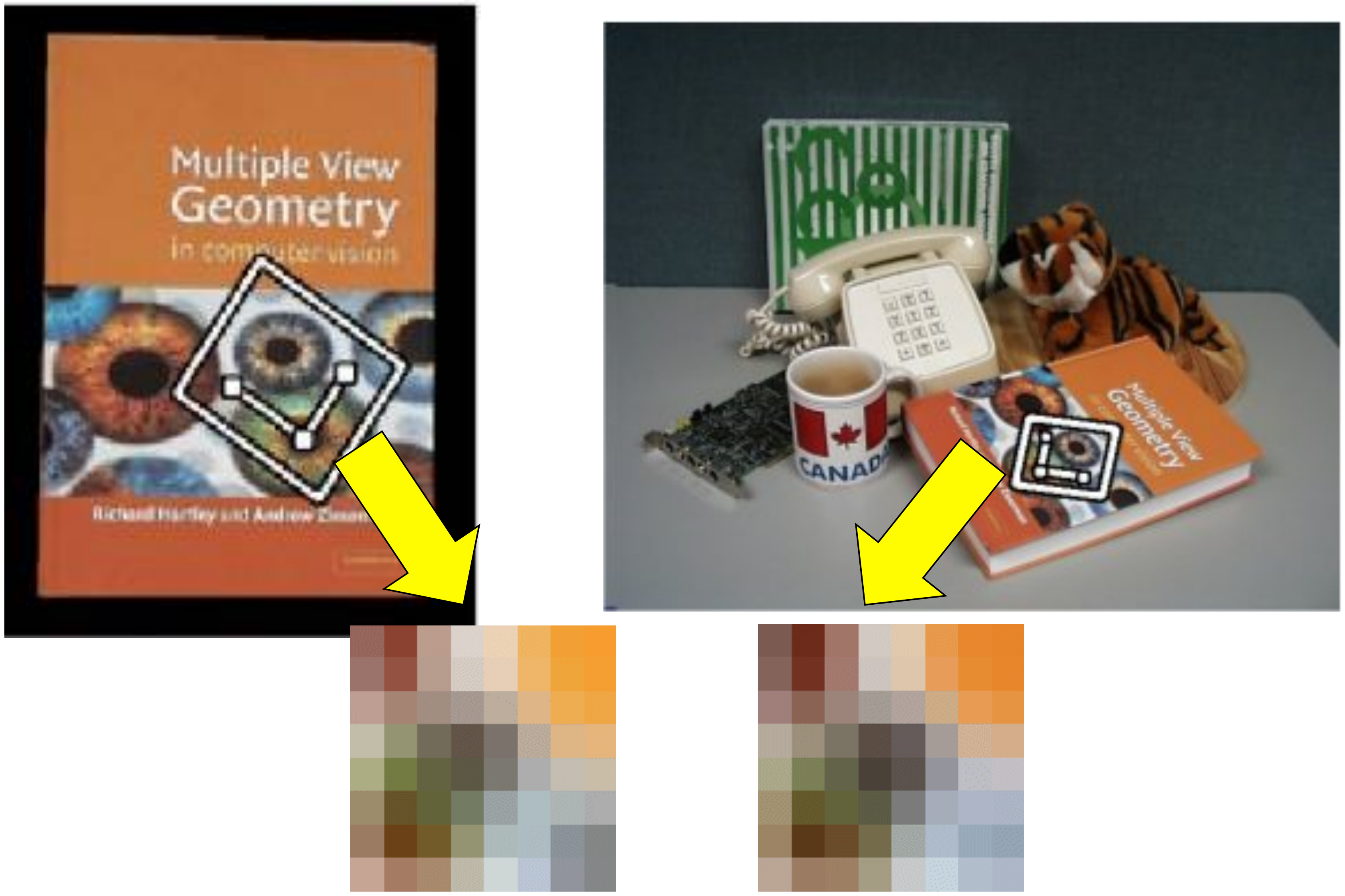
- Problem 2:
  - For each point correctly recognize the corresponding one



We need a reliable and distinctive descriptor

[Darya Frolova and Denis Simakov]

Geometric transformations


Multiple View Geometry in computer vision
Richard Hartley and Andrew Zisserman
CANADA
Multiple View Geometry

Photometric transformations

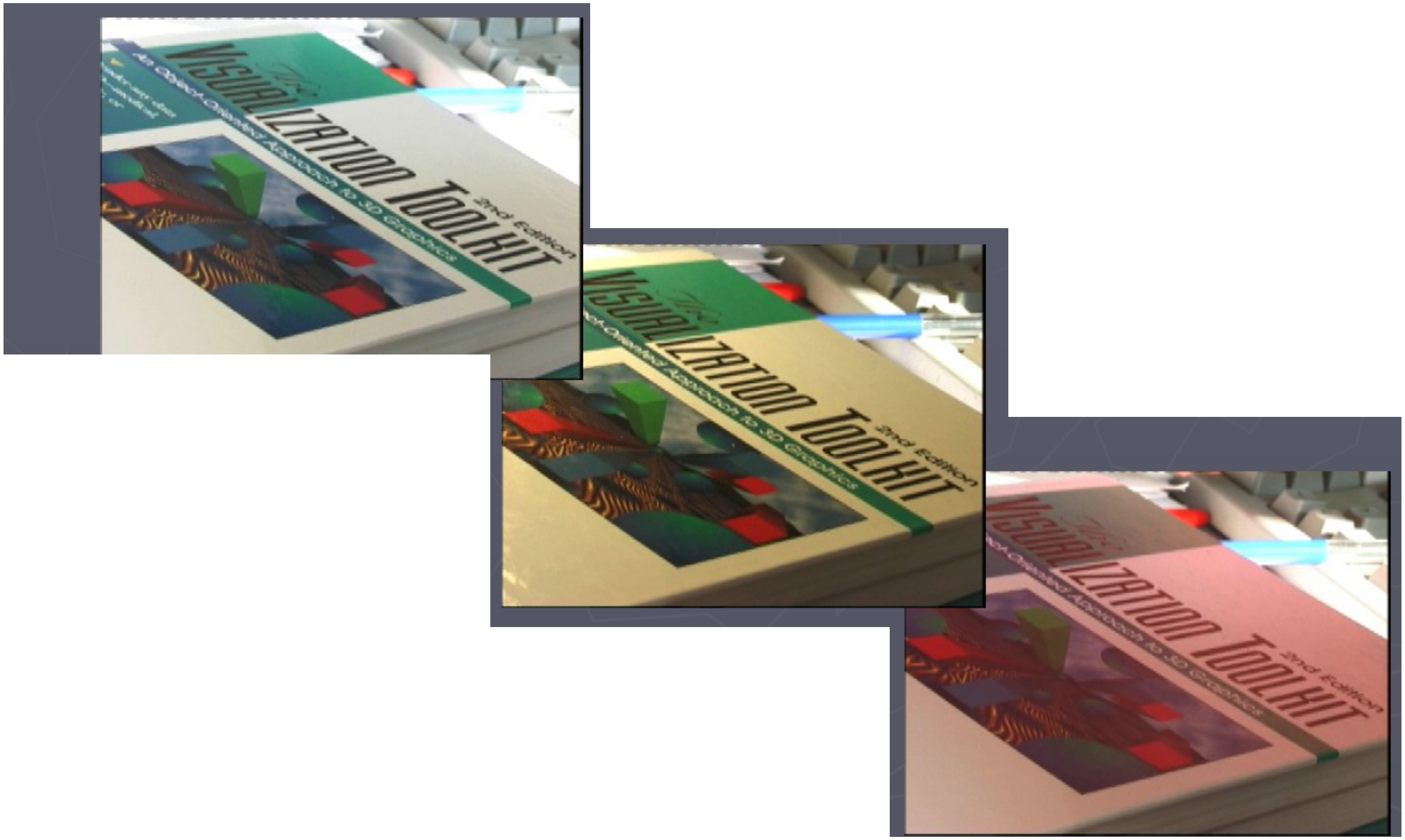

The Visualization Toolkit
2nd Edition
An Object-Oriented Approach to 3D Graphics

Figure from T. Tuytelaars ECCV 2006 tutorial

And other nuisances…

- Noise
- Blur
- Compression artifacts
- …

Invariant local features

Subset of local feature types designed to be invariant to common geometric and photometric transformations.

Basic steps:

1) Detect distinctive interest points

2) Extract invariant descriptors

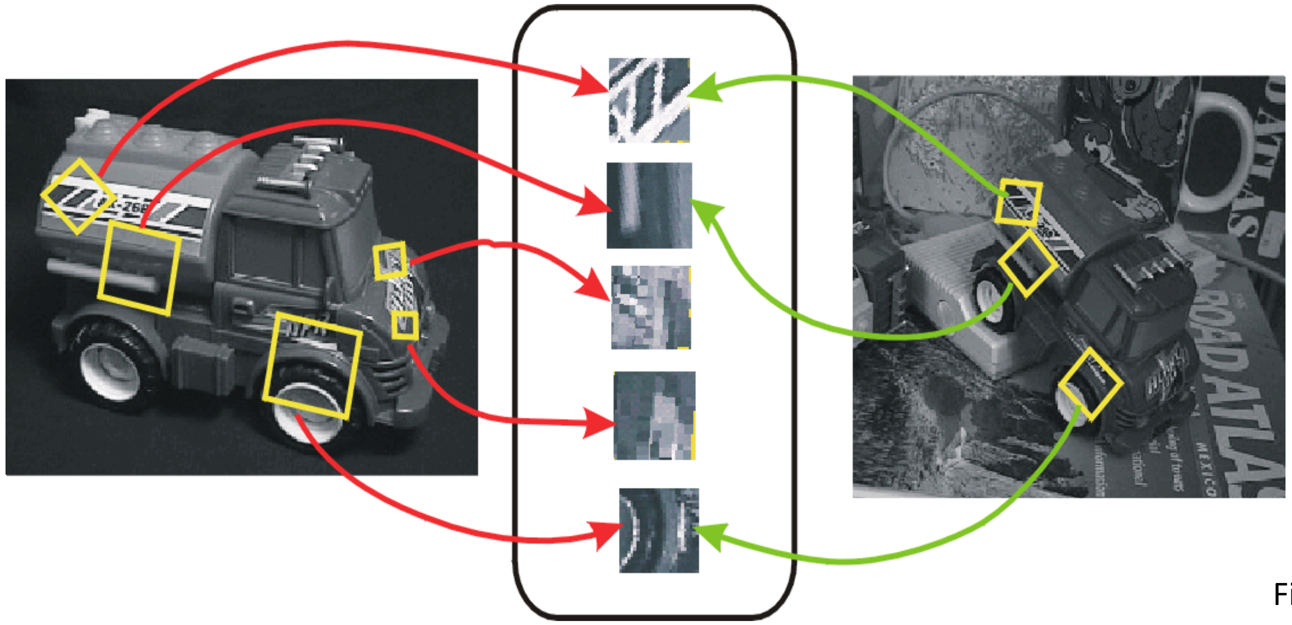


Figure: David Lowe

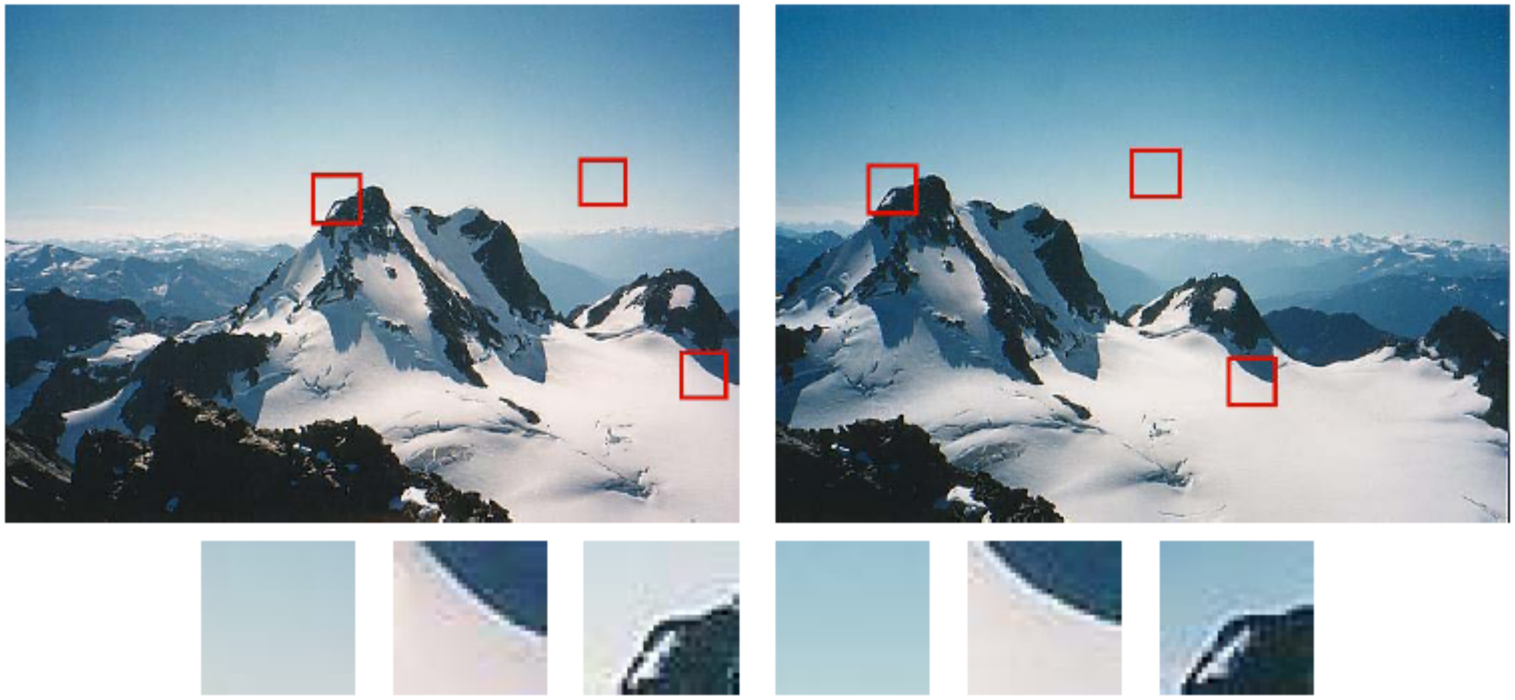
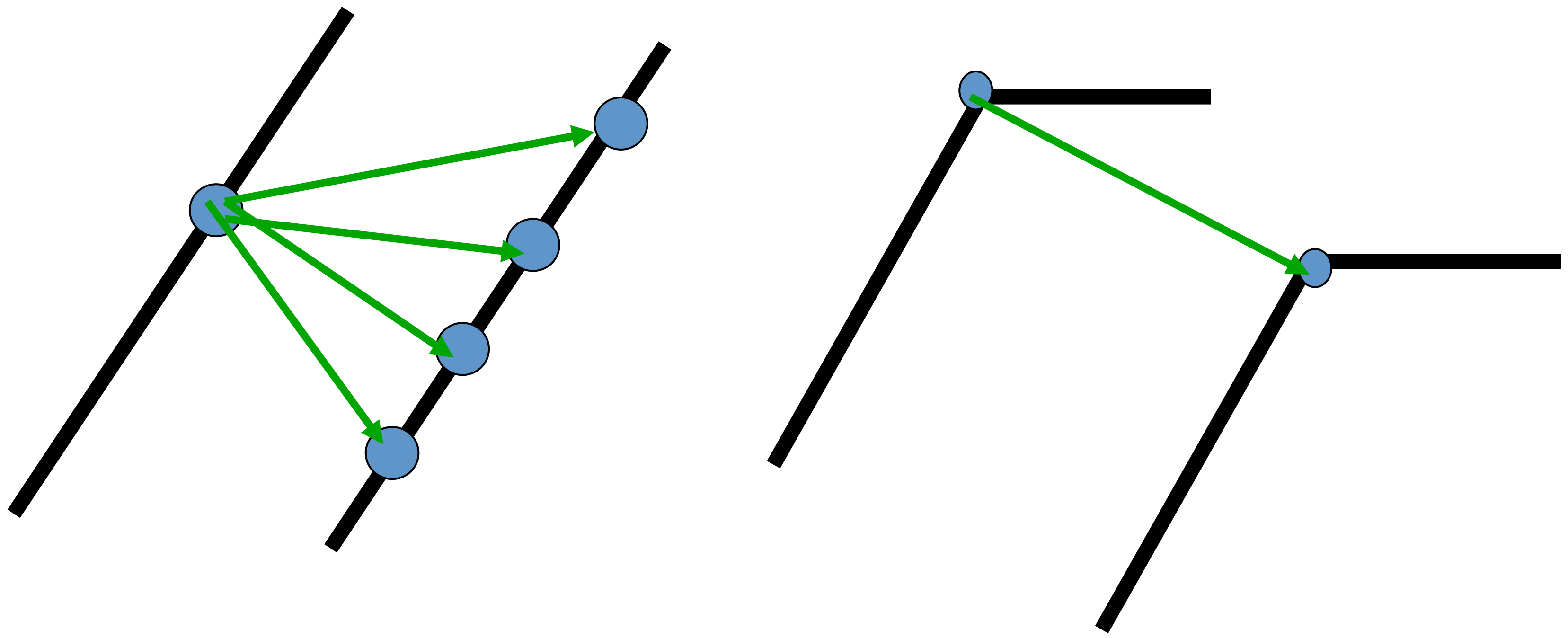


Figure 4.3: *Image pairs with extracted patches below. Notice how some patches can be localized or matched with higher accuracy than others.*

Corners contain more info than lines.

- A point on a line is hard to match; corners are “easy”

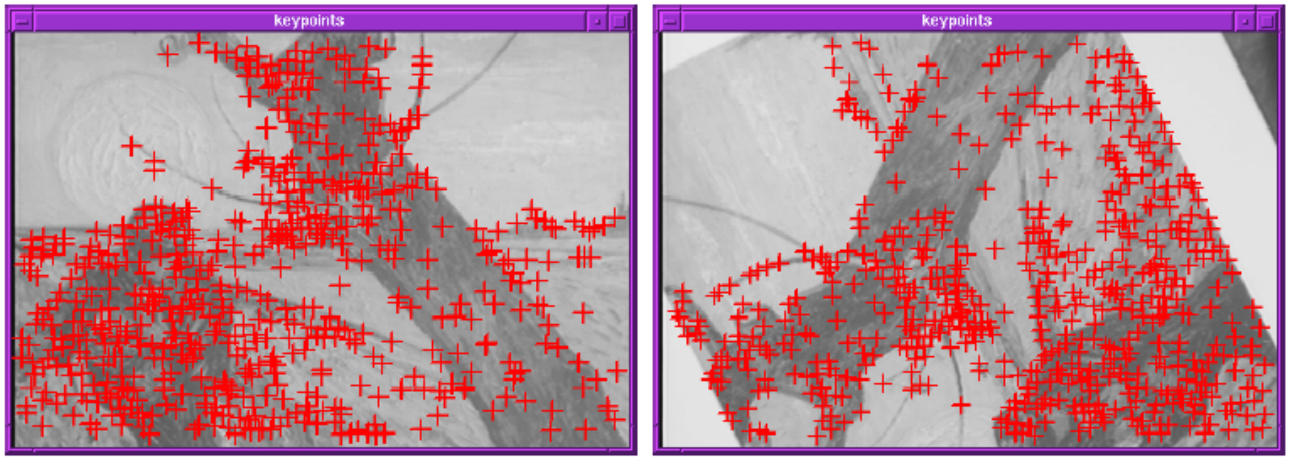


Main questions

- Where will the interest points come from?
  - What are salient features that we'll *detect* in multiple views?
- How to *describe* a local region?
- How to establish *correspondences*, i.e., compute matches?

Today: Harris corner detector with SSD matching
Tomorrow: scale invariant detectors and descriptors, SIFT

Finding Corners



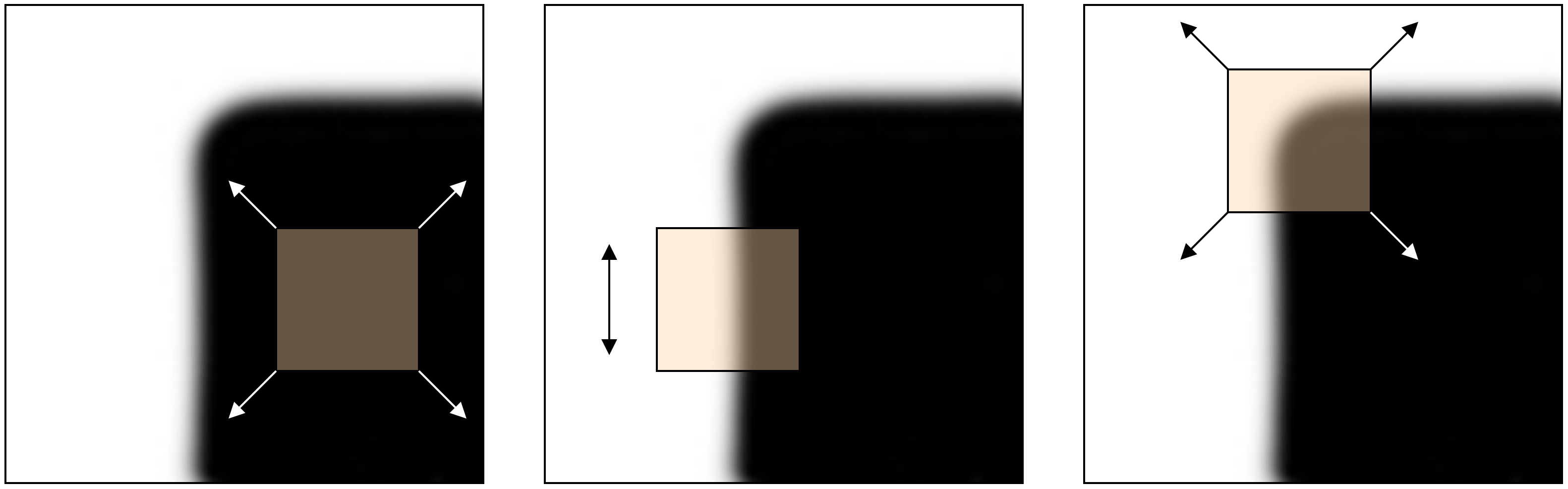
- Key property: in the region around a corner, image gradient has two or more dominant directions
- Corners are repeatable and **distinctive**

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

Source: Lana Lazebnik

Corners as distinctive interest points

- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give *a large change* in intensity



"flat" region: no change in all directions

"edge": no change along the edge direction

"corner": significant change in all directions

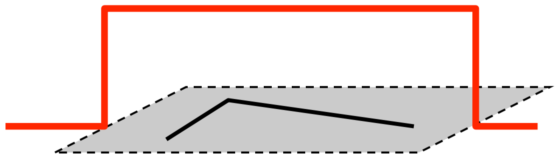
Source: A. Efros

Harris Detector formulation

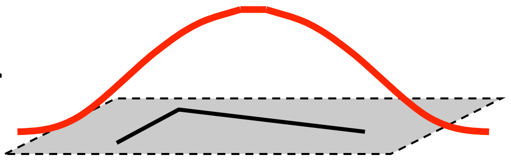
Change of intensity for the shift [*u,v*]:

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{Window function}} \Big[\underbrace{I(x+u, y+v)}_{\text{Shifted intensity}} - \underbrace{I(x,y)}_{\text{Intensity}} \Big]^2$$

Window function $w(x,y) =$

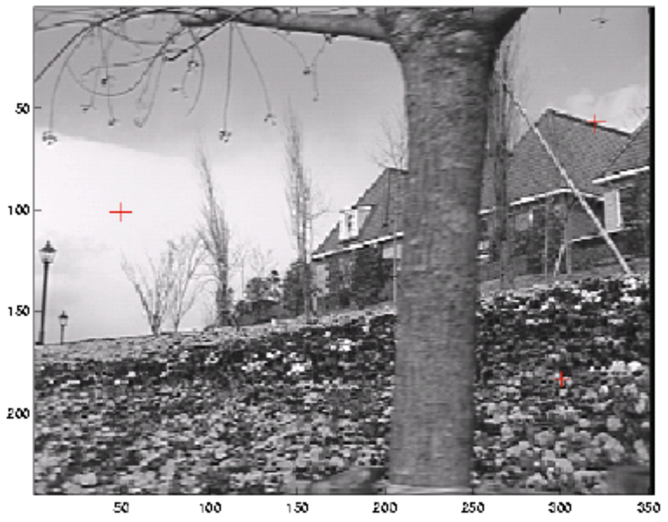


or

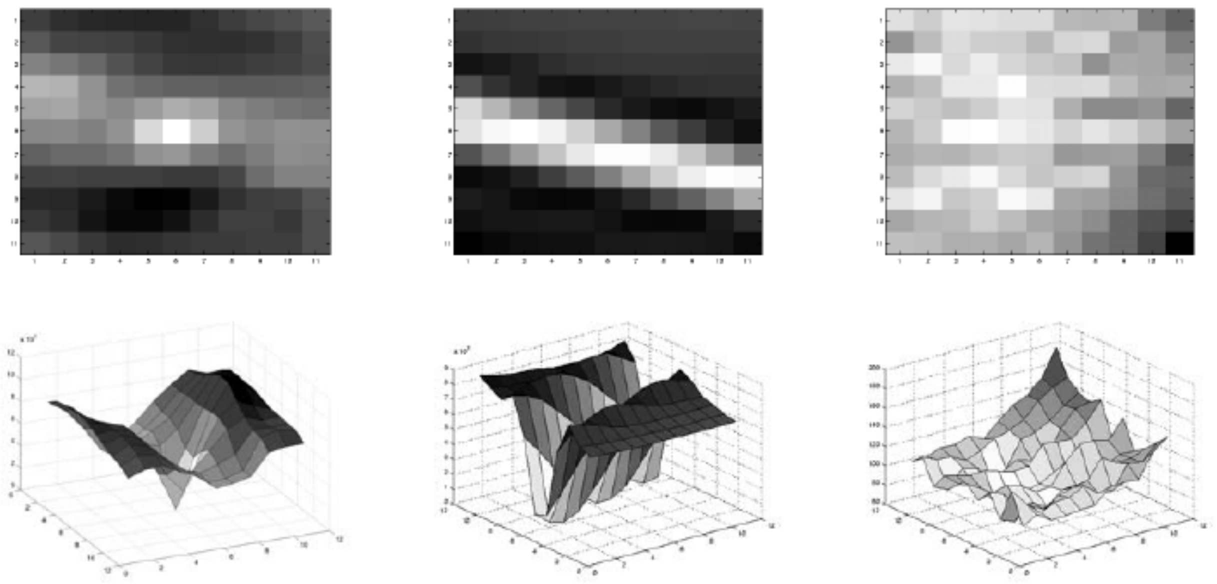


1 in window, 0 outside Gaussian

Source: R. Szeliski



(a)



Harris Detector formulation

This measure of change can be approximated by:

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

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

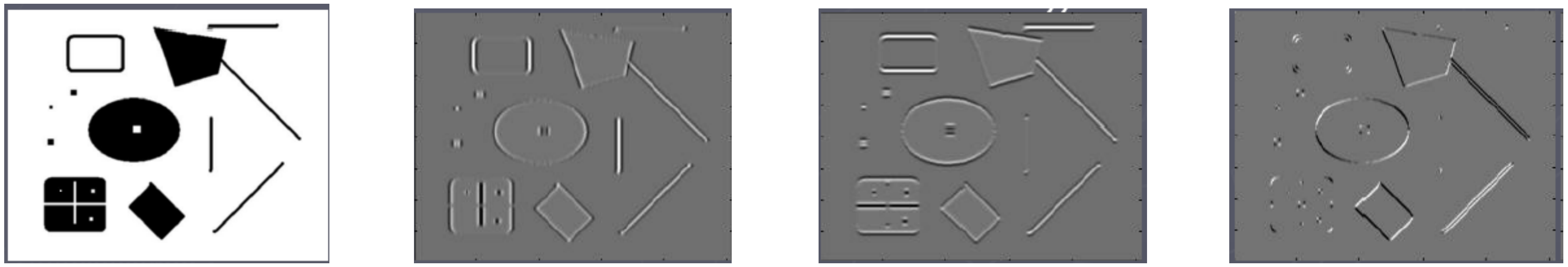
$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

Gradient with respect to x, times gradient with respect to y

Sum over image region – area we are checking for corner

$$M = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \; I_y]$$

Harris Detector formulation



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

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

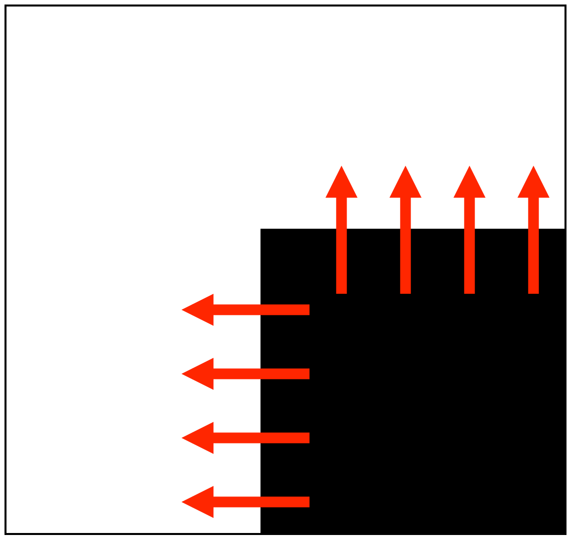
Gradient with respect to x, times gradient with respect to y

Sum over image region – area we are checking for corner

$$M = \begin{bmatrix} \sum I_x I_x & \sum I_x I_y \\ \sum I_x I_y & \sum I_y I_y \end{bmatrix} = \sum \begin{bmatrix} I_x \\ I_y \end{bmatrix} [I_x \; I_y]$$

What does this matrix reveal?

First, consider an axis-aligned corner:



What does this matrix reveal?

First, consider an axis-aligned corner:

$$M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

This means dominant gradient directions align with x or y axis

If either λ is close to 0, then this is **not** a corner, so look for locations where both are large.

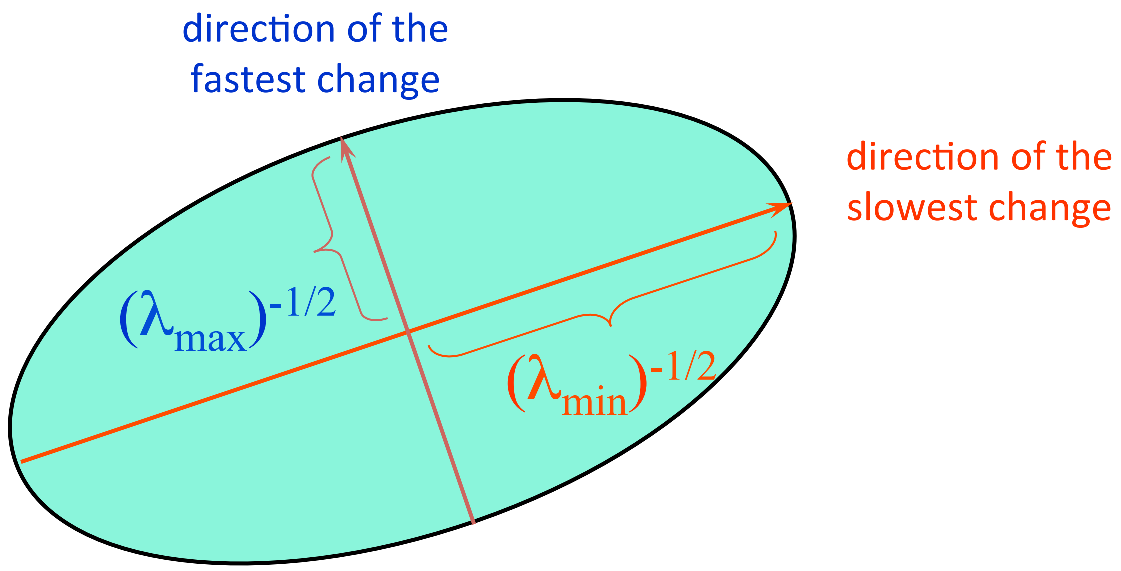
What if we have a corner that is not aligned with the image axes?

Slide credit: David Jacobs

General Case

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

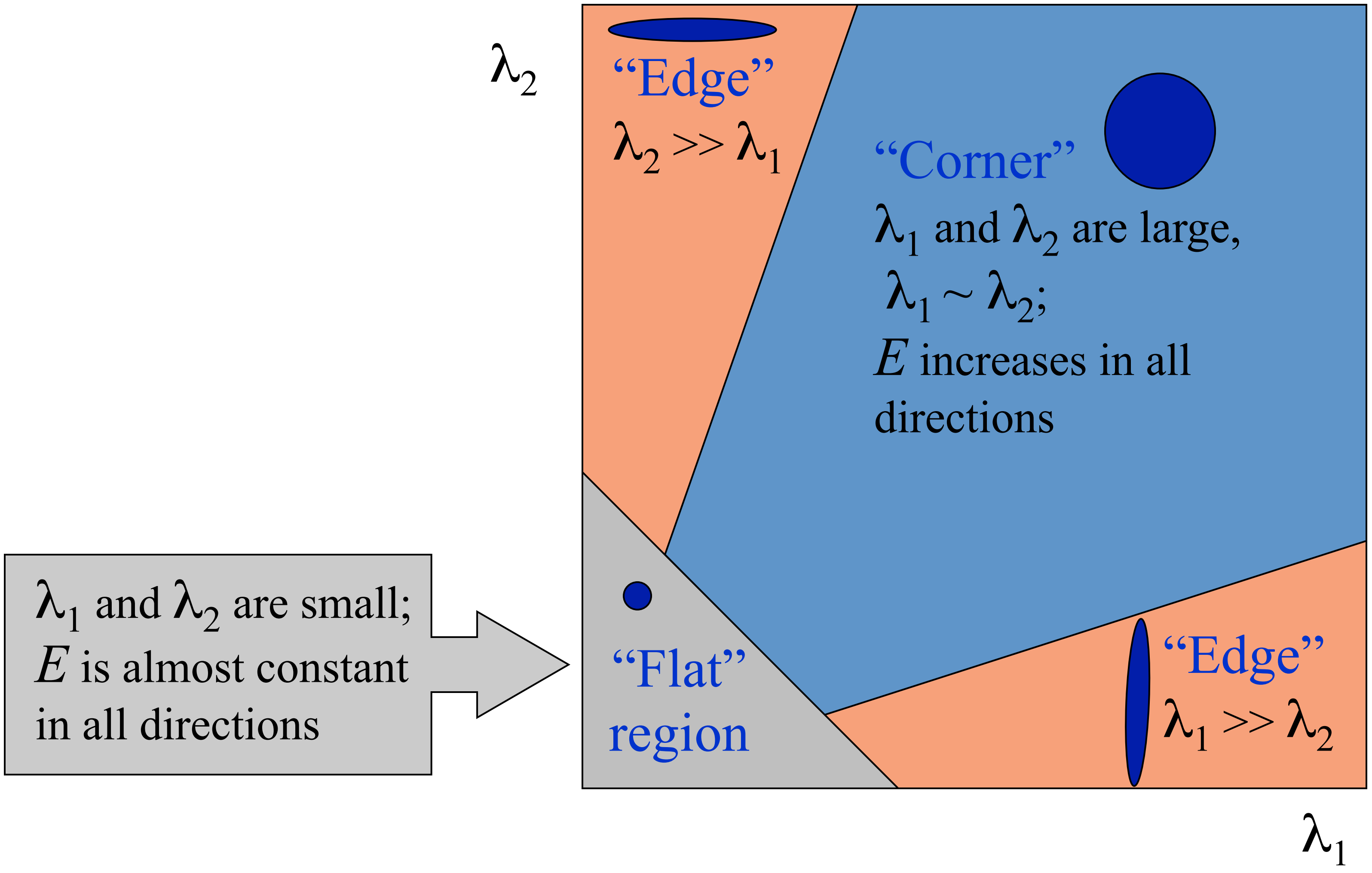
We can visualize *M* as an ellipse with axis lengths determined by the eigenvalues and orientation determined by *R*


direction of the fastest change
direction of the slowest change
$(\lambda_{max})^{-1/2}$
$(\lambda_{min})^{-1/2}$

Slide adapted form Darya Frolova, Denis Simakov.

Interpreting the eigenvalues

Classification of image points using eigenvalues of *M*:

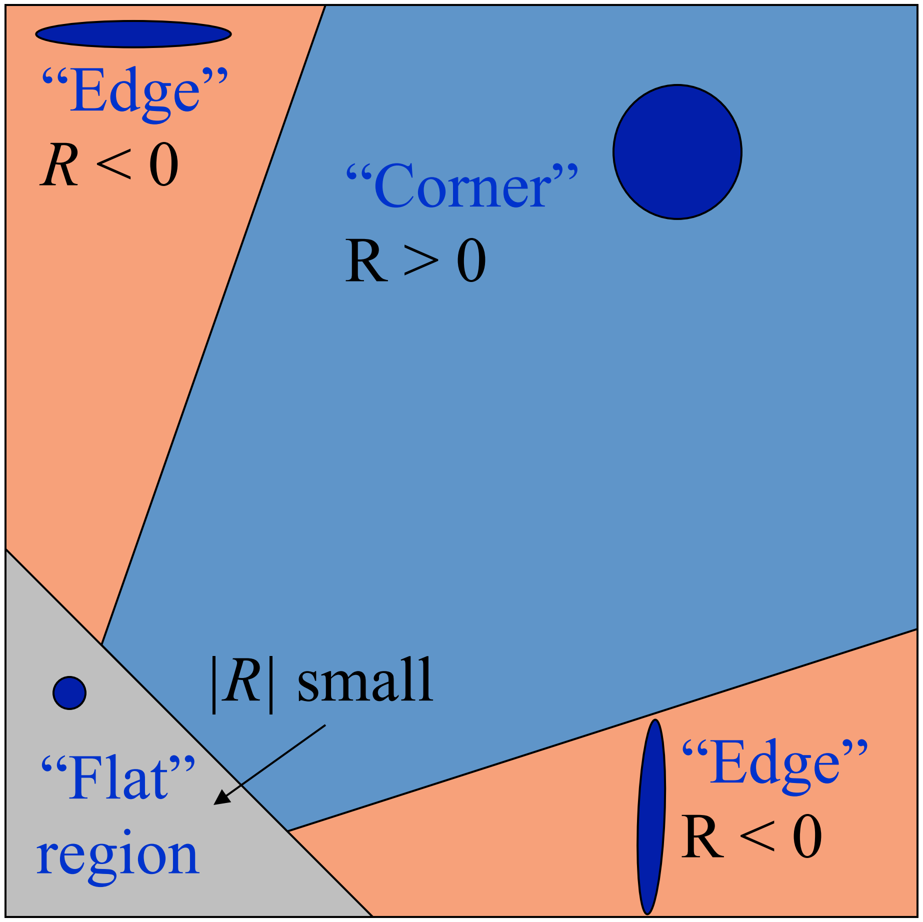


λ_2
"Edge"
$\lambda_2 >> \lambda_1$
"Corner"
λ_1 and λ_2 are large,
$\lambda_1 \sim \lambda_2$;
E increases in all directions
λ_1 and λ_2 are small;
E is almost constant in all directions
"Flat" region
"Edge"
$\lambda_1 >> \lambda_2$
λ_1

Corner response function

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

α: constant (0.04 to 0.06)



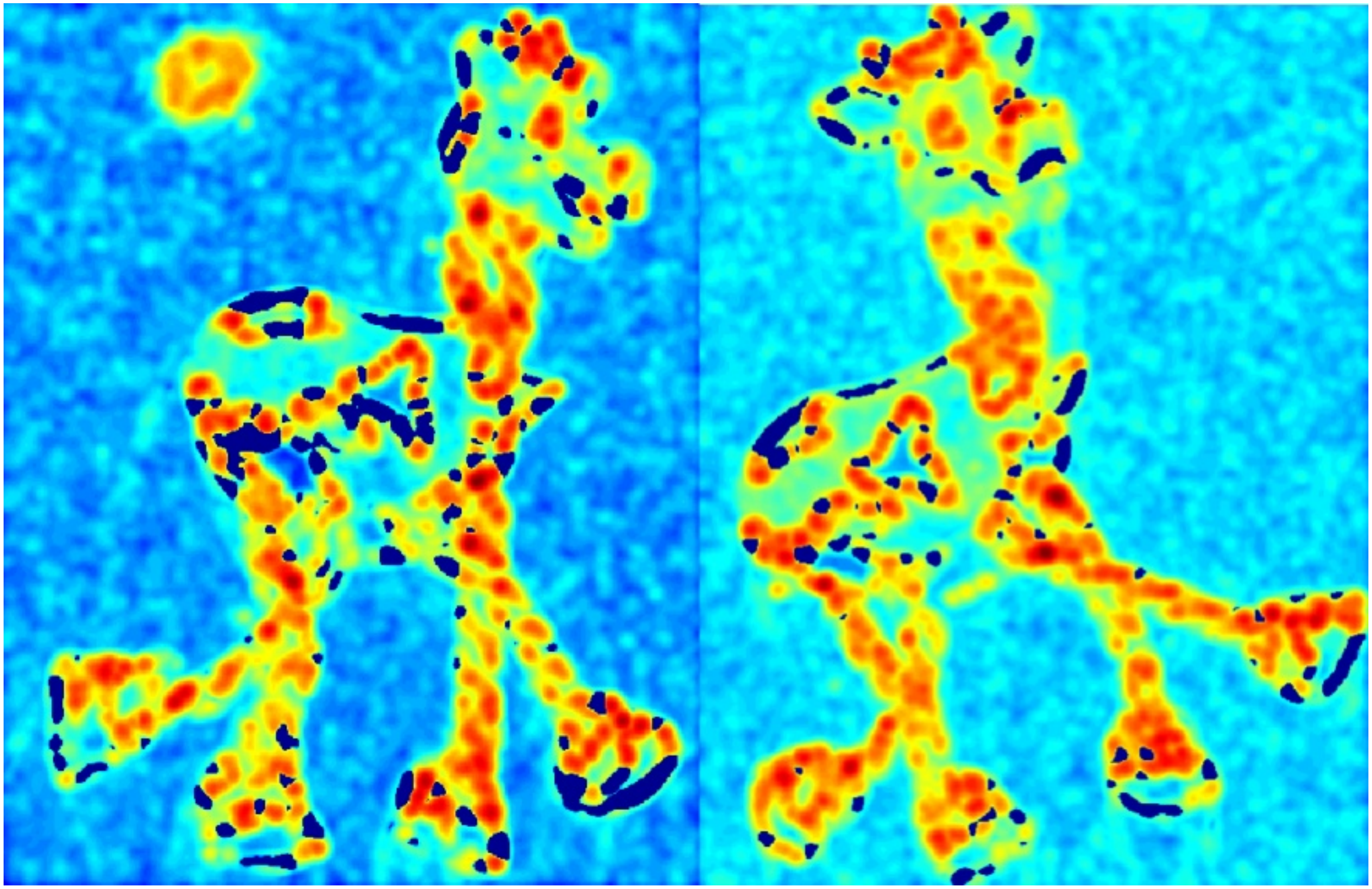
Harris Corner Detector

- Algorithm steps:
  - Compute M matrix within all image windows to get their R scores
  - Find points with large corner response
    (R > threshold)
  - Take the points of local maxima of R

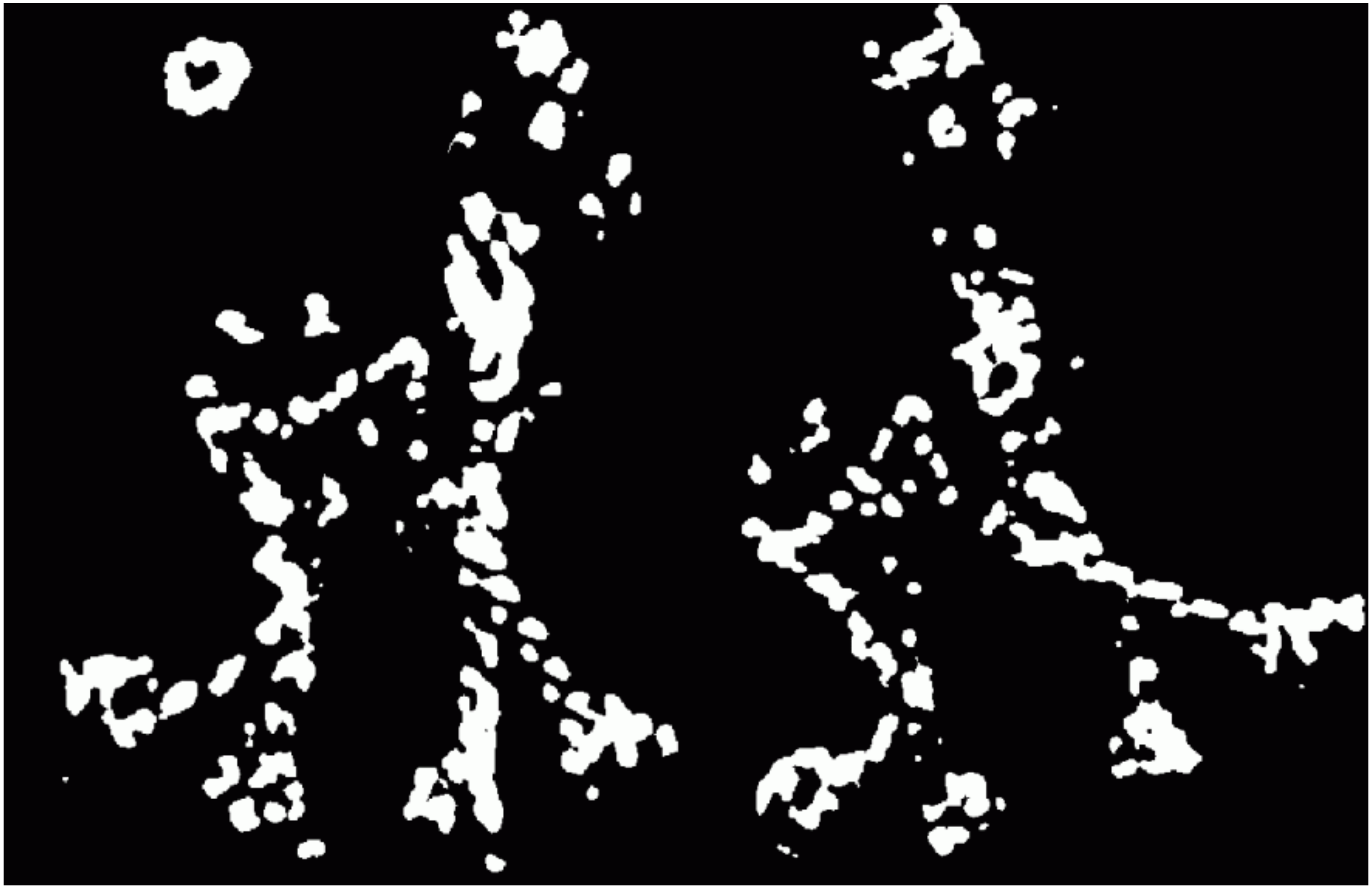
Harris Detector: Workflow

Slide adapted form Darya Frolova, Denis Simakov, Weizmann Institute.

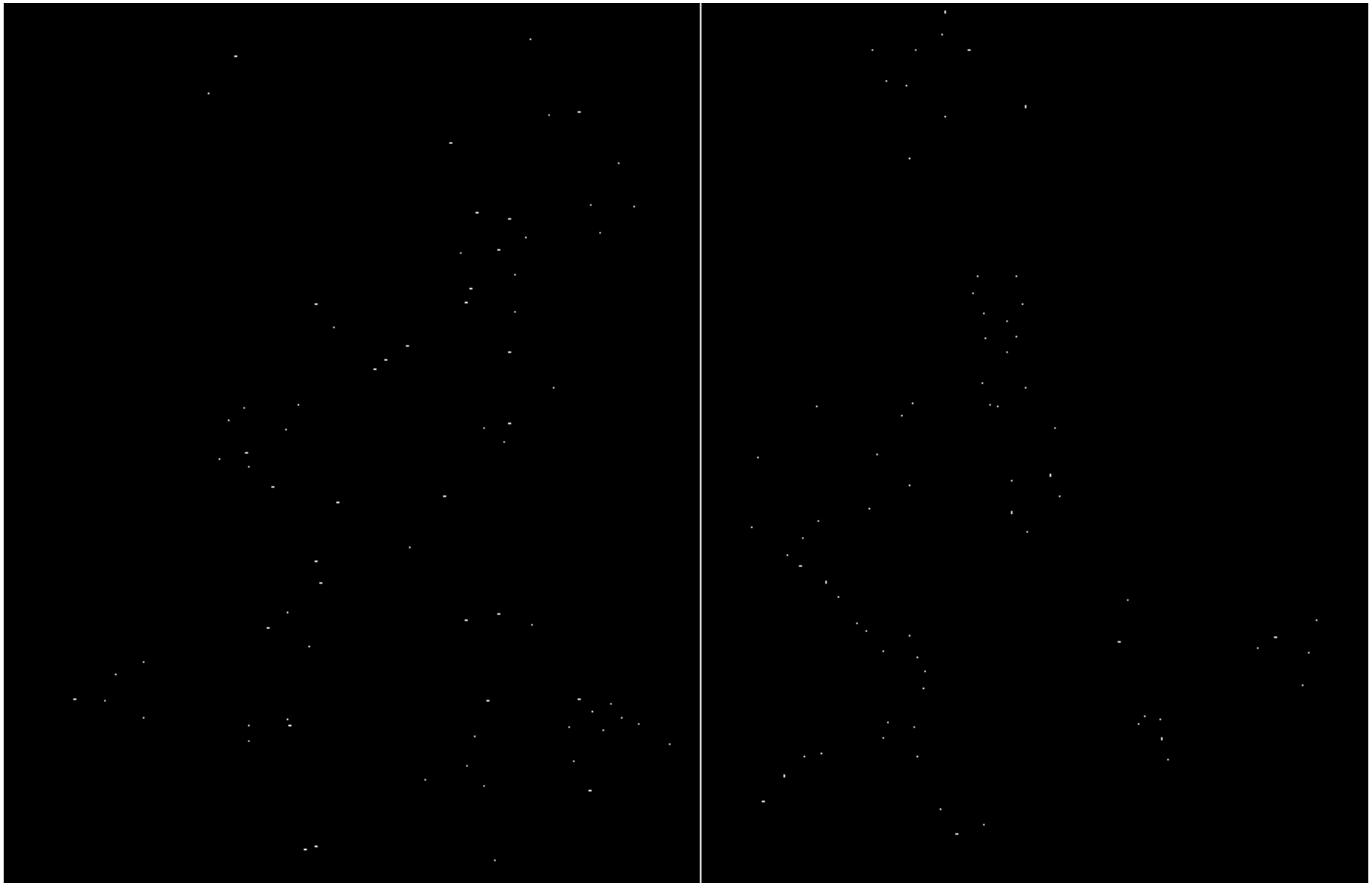
Compute corner response R

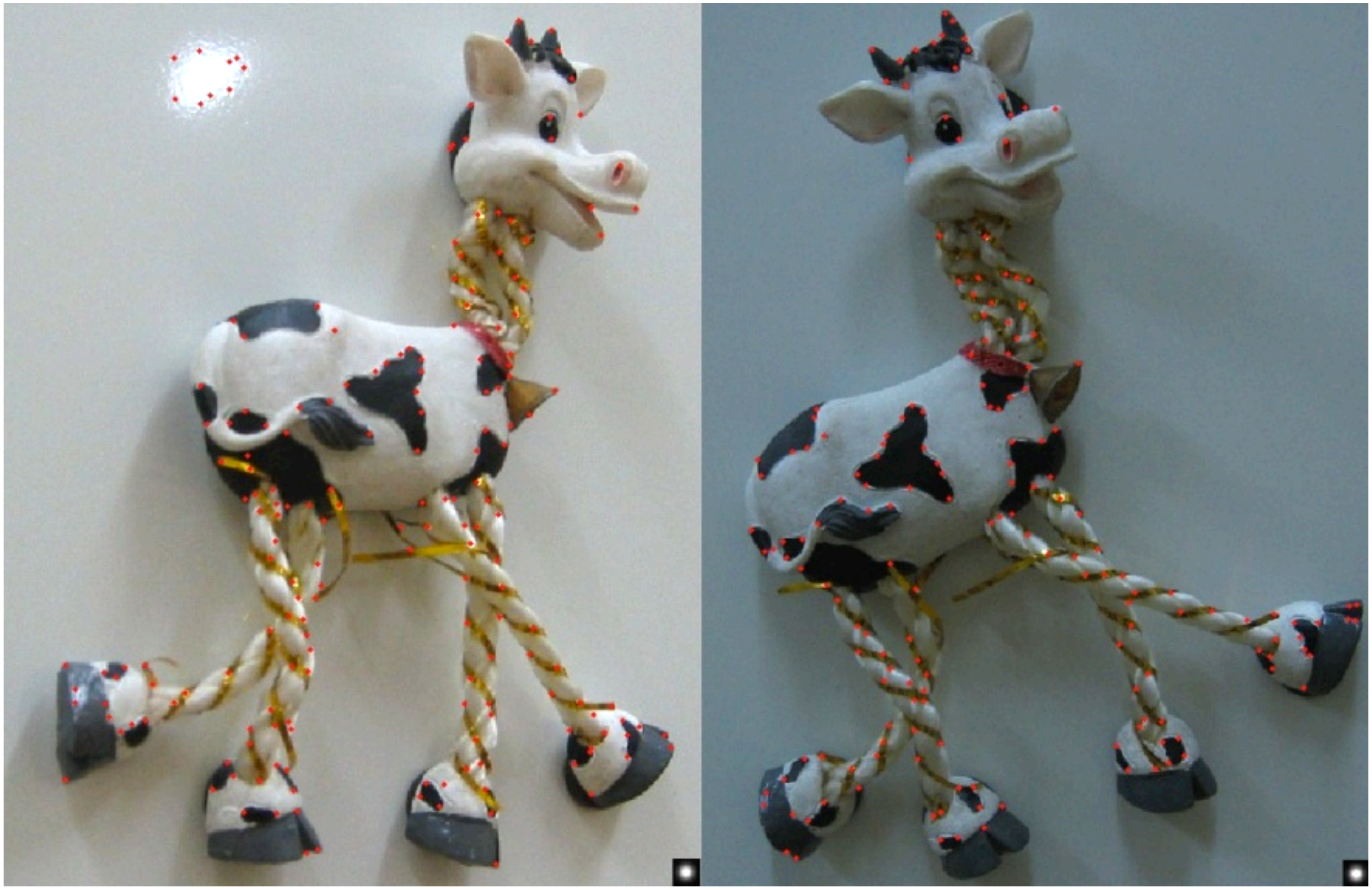


Find points with large corner response: R>threshold



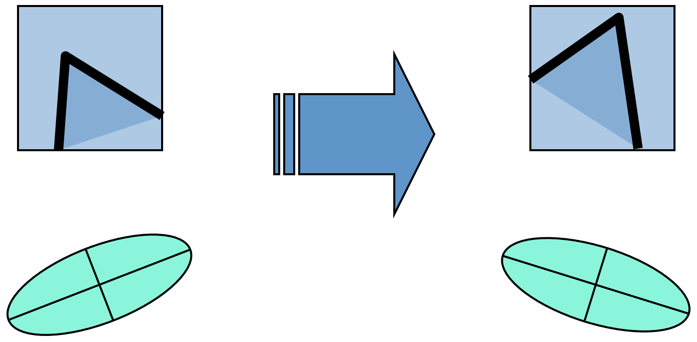
Take only the points of local maxima of R





Harris Detector: Properties

- Rotation invariance

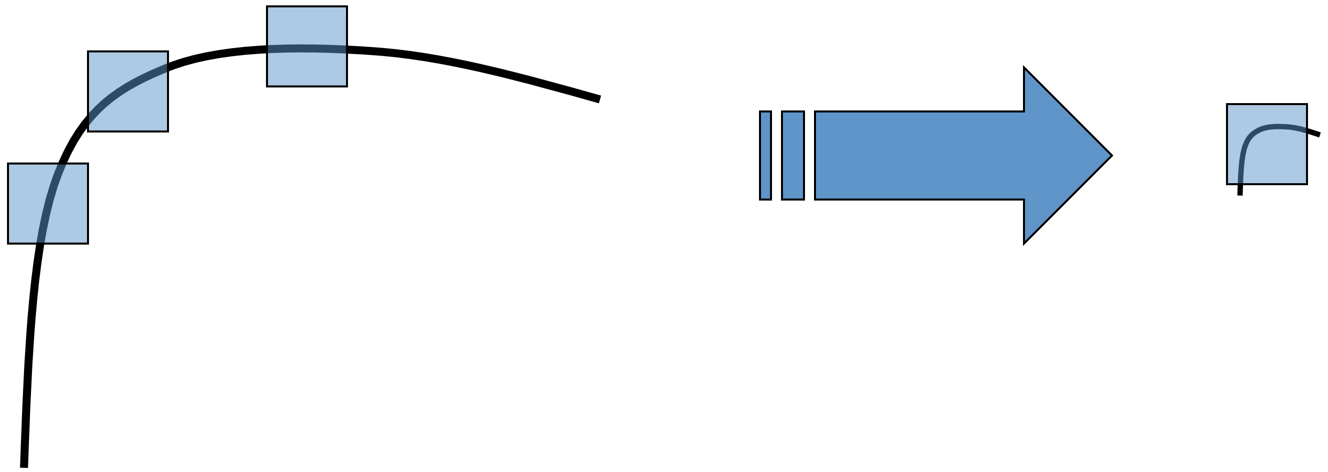


Ellipse rotates but its shape (i.e. eigenvalues) remains the same

Corner response R is invariant to image rotation

Harris Detector: Properties

- Not invariant to image scale

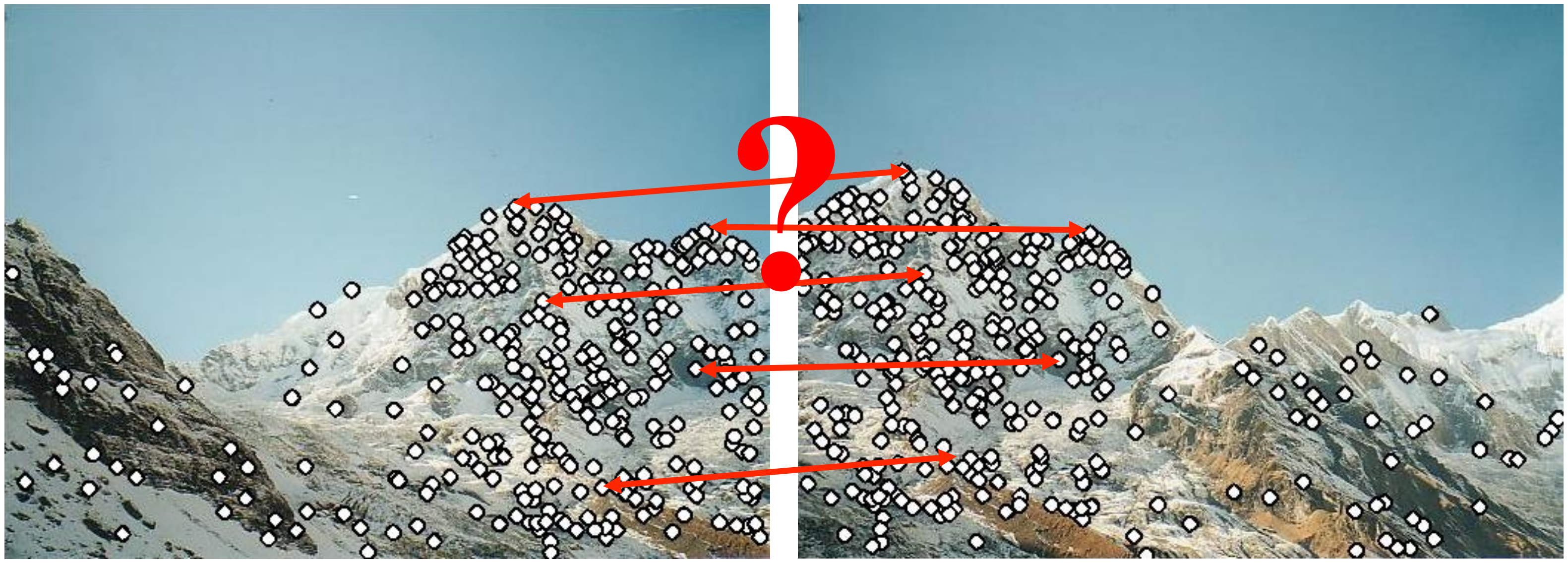


All points will be classified as edges

Corner !

Feature descriptors

We know how to detect **and describe** good points
Next question: **How to match them?**



Feature matching

Given a feature in I_1, how to find the best match in I_2?

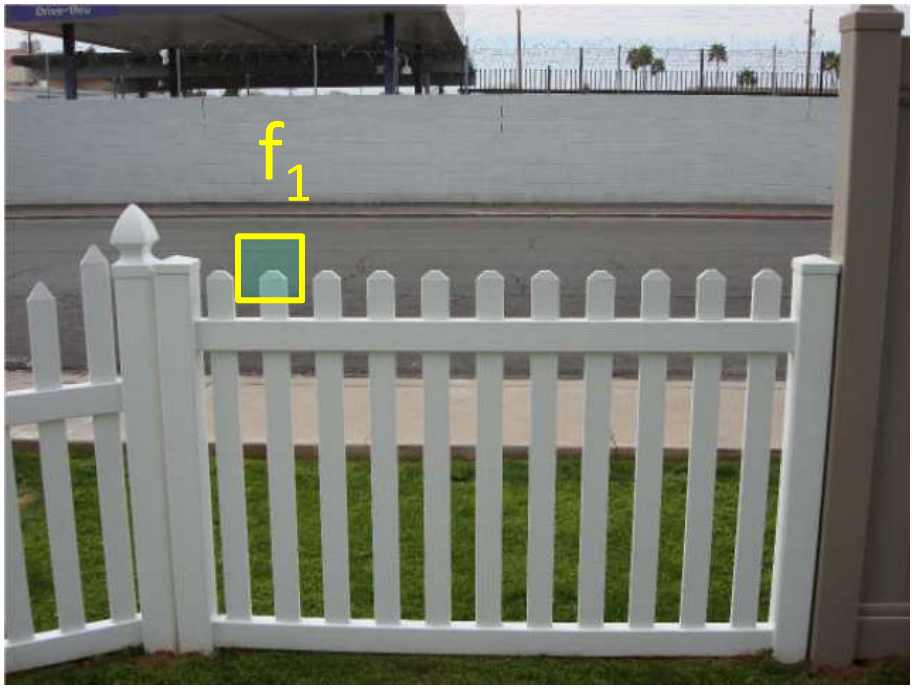
1. Define distance function that compares two descriptors
2. Test all the features in I_2, find the one with min distance

1. Many possible distances:
   1. SSD
   2. NCC
   3. SAD
   4. Earthmovers

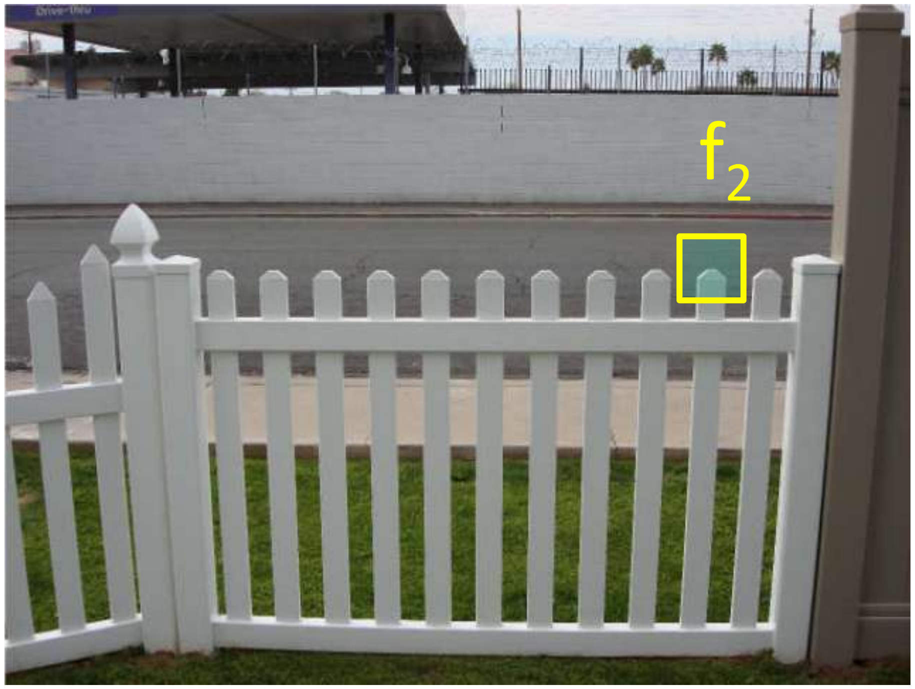
Feature distance

How to define the difference between two features f_1, f_2?

- Simple approach is SSD(f_1, f_2)
  - sum of square differences between entries of the two descriptors
  - can give good scores to very ambiguous (bad) matches



I_1



I_2

Matching High Texture Regions

- Basic SSD

$$SSD(d) = \sum_{q \in R_1} (I_1(q) - I_2(q+d))^2/n$$

- Zero mean SSD

$$\bar{I}_1 = \sum_{q \in \mathbb{R}_1} I_1(q)/n \qquad \tilde{I}_1(q) = I_1(q) - \bar{I}_1$$

- Normalized SSD $\bar{\sigma}_1^2 = \sum_{q \in \mathbb{R}_1} \tilde{I}_1^2/n$

$$SSD(d) = \sum_{q \in R_1} \left(\frac{\tilde{I}_1(q)}{\bar{\sigma}_1} - \frac{\tilde{I}_2(q+d)}{\bar{\sigma}_2(d)}\right)^2/n$$

- Normalized SSD = Normalized Cross-Correlation = Dot Product

$$ZNCC(d) = \sum_{q \in R_1} \frac{\tilde{I}_1(q)\tilde{I}_2(q+d)}{\bar{\sigma}_1 \bar{\sigma}_2(d)}$$

Matching High Texture Regions

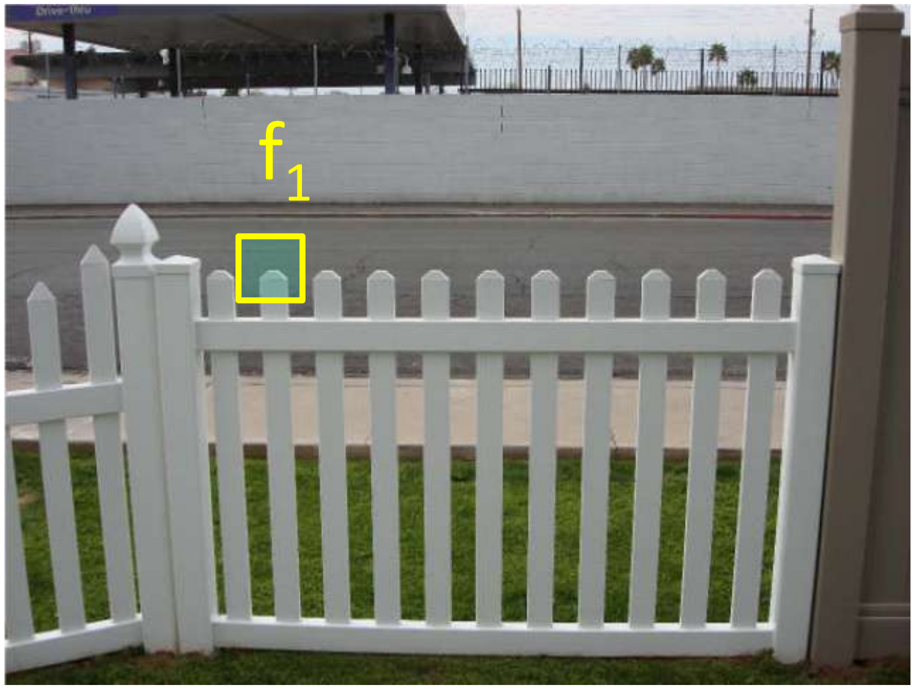
- SAD $$SAD(d) = \sum_{q \in R_1} |I_1(q) - I_2(q+d)|/n$$
- Zero mean SAD
- Other methods
  - Ordinal methods
  - Shuffle distance
  - Mutual Information
  - Other robust matching functions (Huber functions)
- Other pre-processing:
  - Note all of the above can also be done on filtered images
  - Note all of the above can be extended to multi-dimensional feature spaces

Feature distance

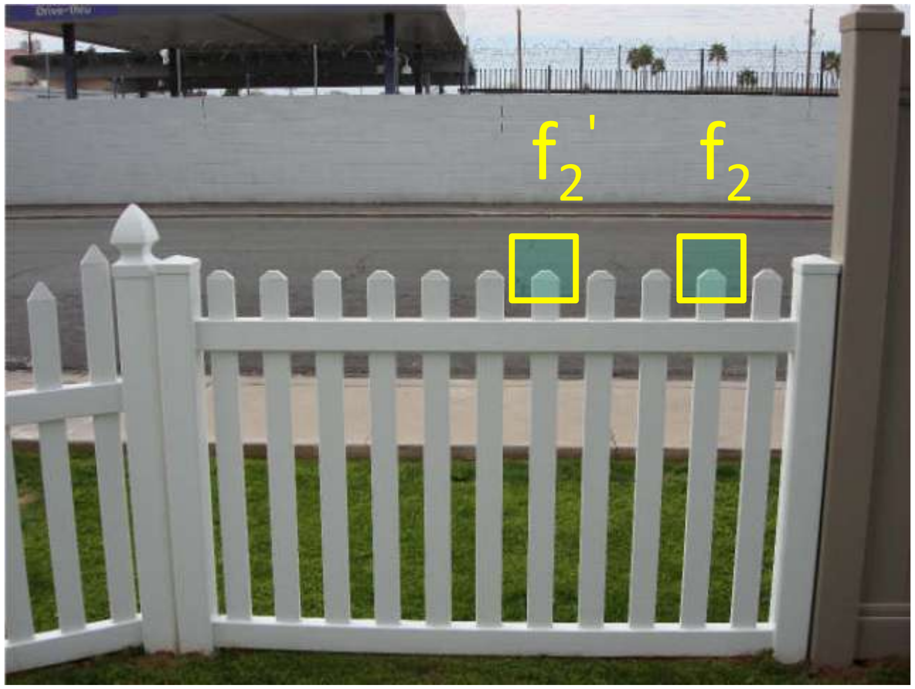
How to define the difference between two features f_1, f_2?

– Better approach: ratio distance = SSD(f_1, f_2) / SSD(f_1, f_2')

- f_2 is best SSD match to f_1 in I_2
- f_2' is 2nd best SSD match to f_1 in I_2
- gives small values for ambiguous matches



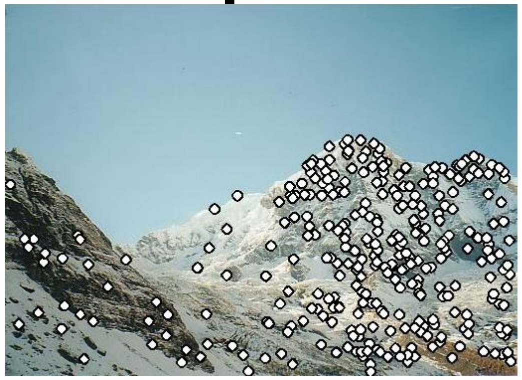
I_1



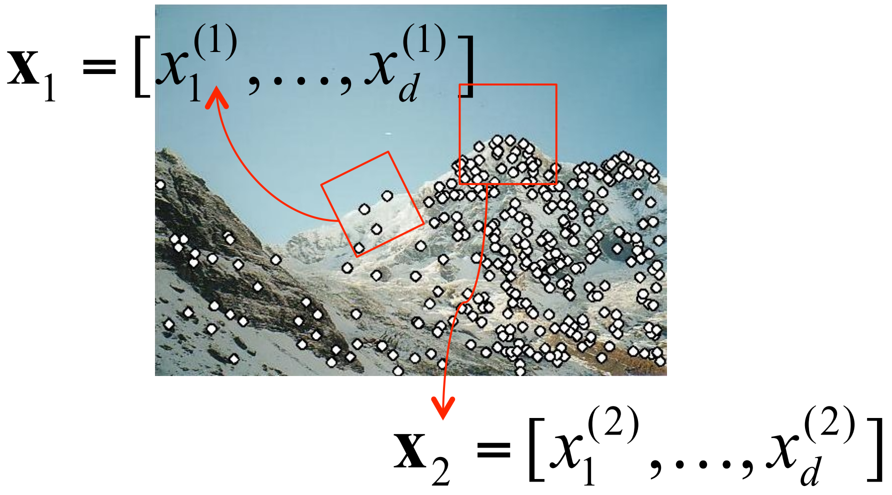
I_2

Local features: main components

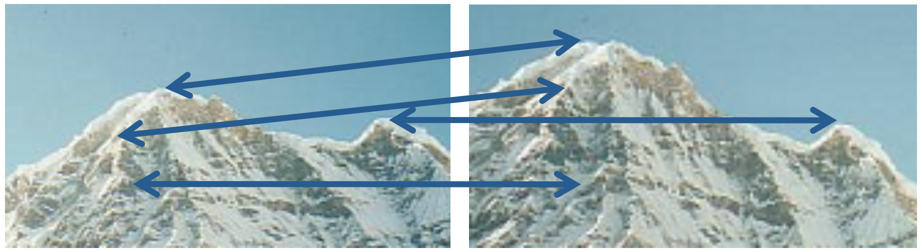
1) **Detection:** Identify the interest points



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

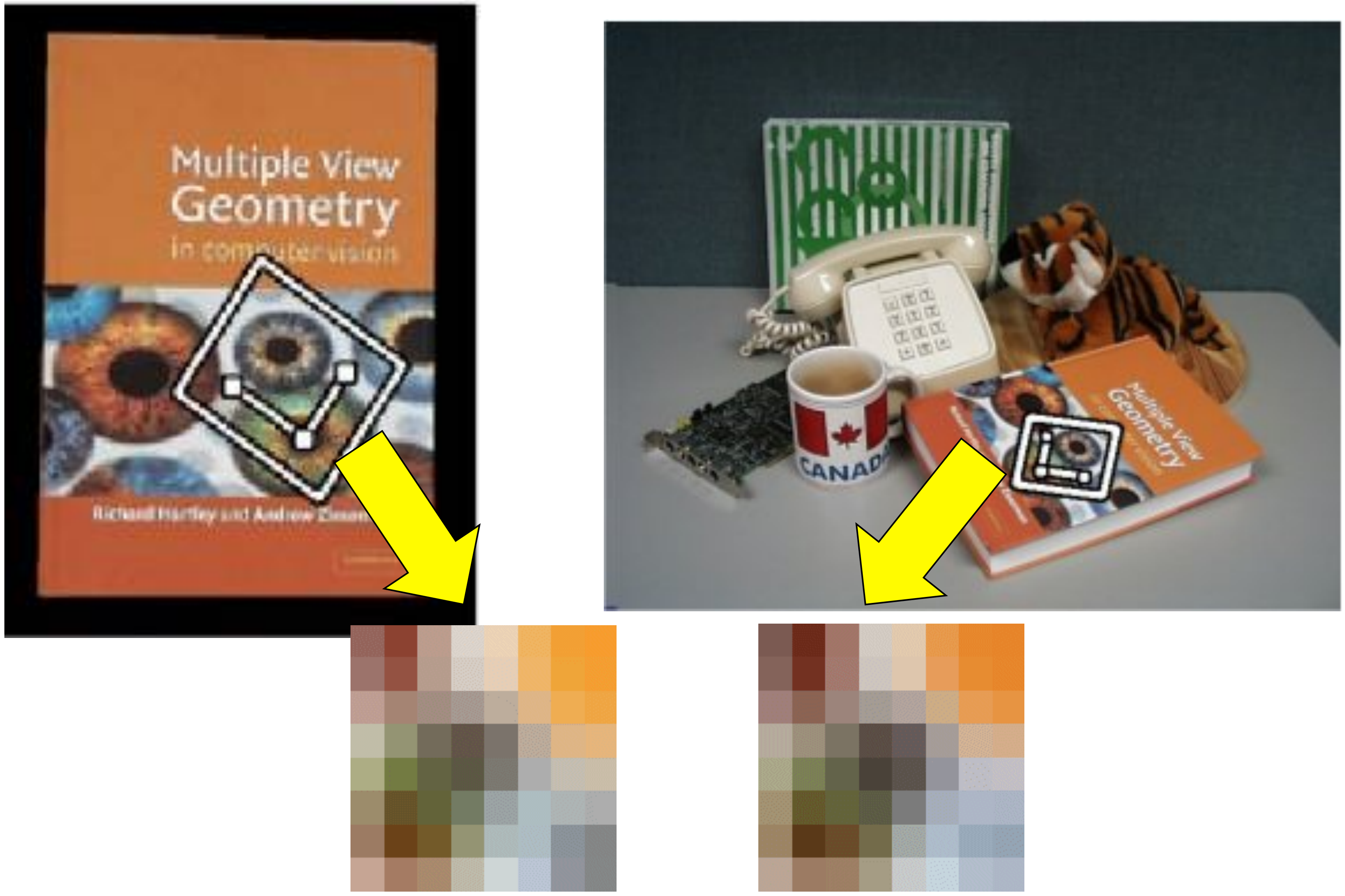

$\mathbf{x}_1 = [x_1^{(1)}, \ldots, x_d^{(1)}]$
$\mathbf{x}_2 = [x_1^{(2)}, \ldots, x_d^{(2)}]$

3) **Matching:** Determine correspondence between descriptors in two views



Kristen Grauman

Issue: Geometric transformations


Multiple View Geometry in computer vision
Richard Hartley and Andrew Zisserman
CANADA
Multiple View Geometry

Invariance

We'd like to find the same features regardless of the transformation

- This is called transformational ***invariance***
- Most feature methods are designed to be invariant to
  - Translation, 2D rotation, scale
- They can usually also handle
  - Limited 3D rotations (SIFT works up to about 60 degrees)
  - Limited affine transformations (2D rotation, scale, shear)
  - Limited illumination/contrast changes

How to achieve invariance

Need both of the following:

1. Make sure your detector is invariant
   - Harris is invariant to translation and rotation
   - Scale is trickier
     - common approach is to detect features at many scales using a Gaussian pyramid (e.g., MOPS)
     - More sophisticated methods find "the best scale" to represent each feature (e.g., SIFT)
2. Design an invariant feature *descriptor*
   - A descriptor captures the information in a region around the detected feature point
   - The simplest descriptor: a square window of pixels
   - Let's look at some better approaches…

Rotation invariance

Find dominant orientation of the image patch

- This is given by $\mathbf{x}_+$, the eigenvector of $\mathbf{H}$ corresponding to λ_+ (the *larger* eigenvalue)
- Rotate the patch according to this angle

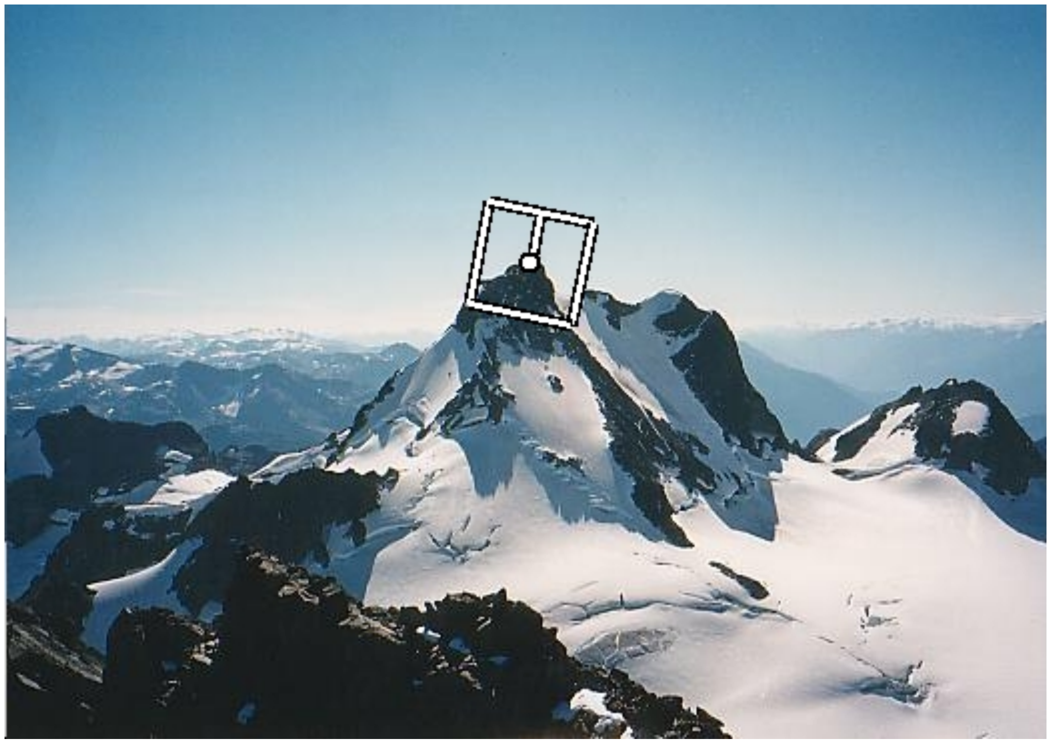
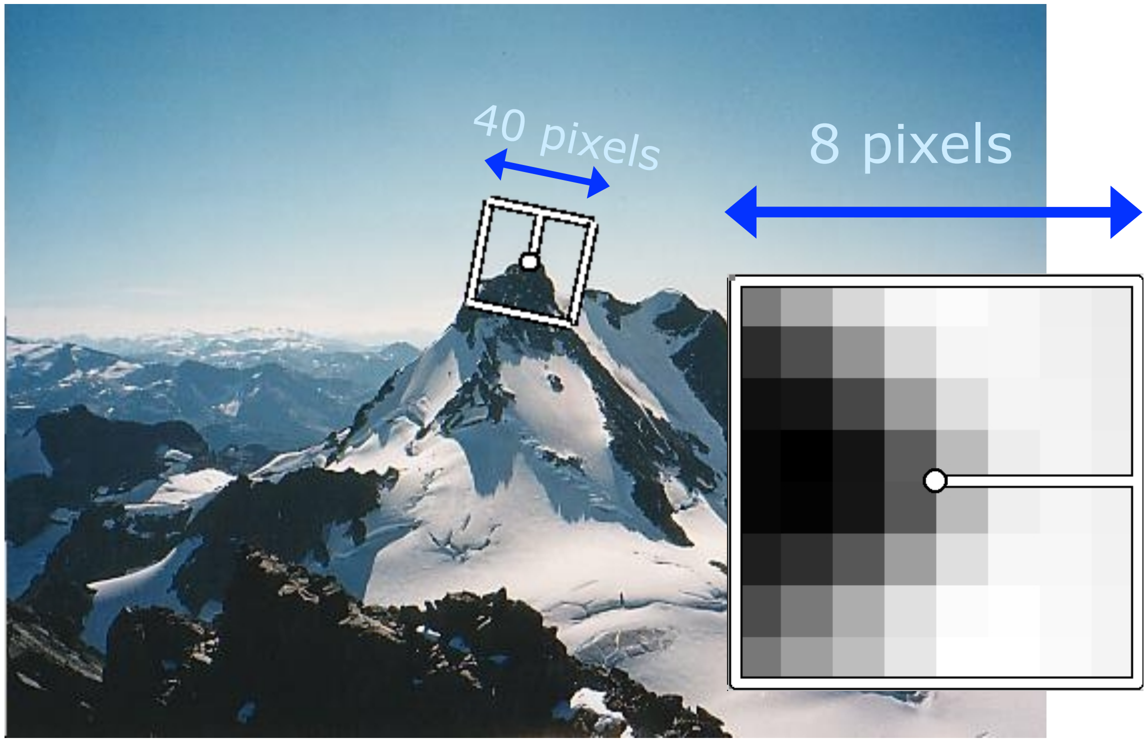


Figure by Matthew Brown

Multiscale Oriented PatcheS descriptor

Take a 40x40 square window around detected feature

- Scale to 1/5 size (using prefiltering)
- Rotate to horizontal
- Sample 8x8 square window centered at feature
- Intensity normalize the window by subtracting the mean, dividing by the standard deviation in the window (both window $\mathbf{I}$ and $a\mathbf{I}+b$ will match)



Adapted from slide by Matthew Brown

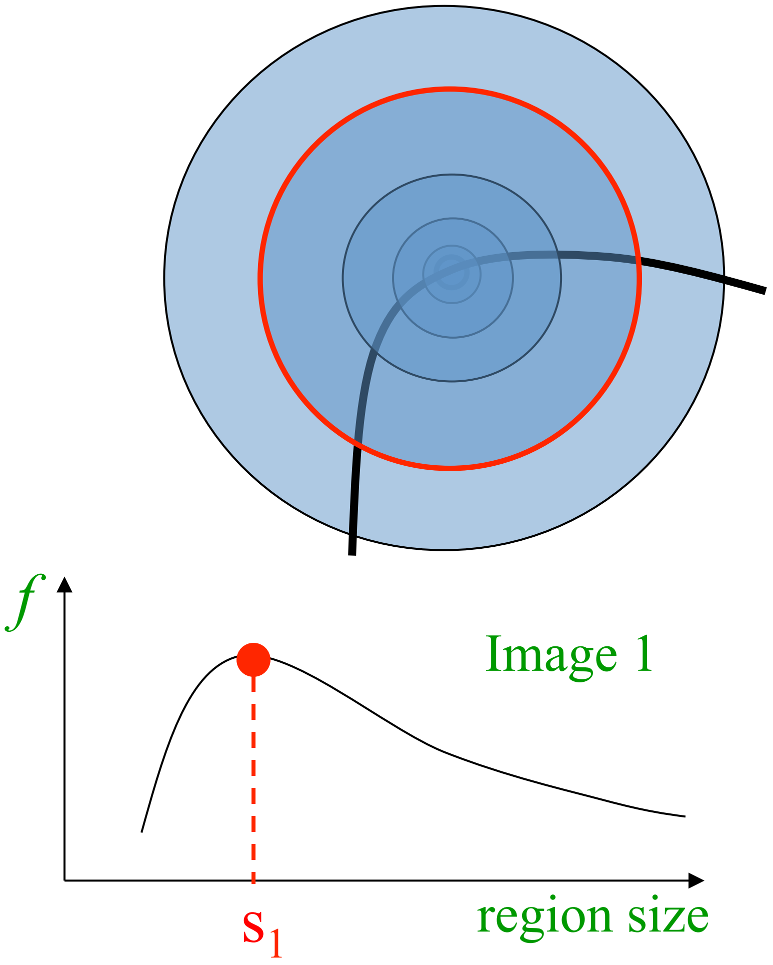
Scale invariant interest points

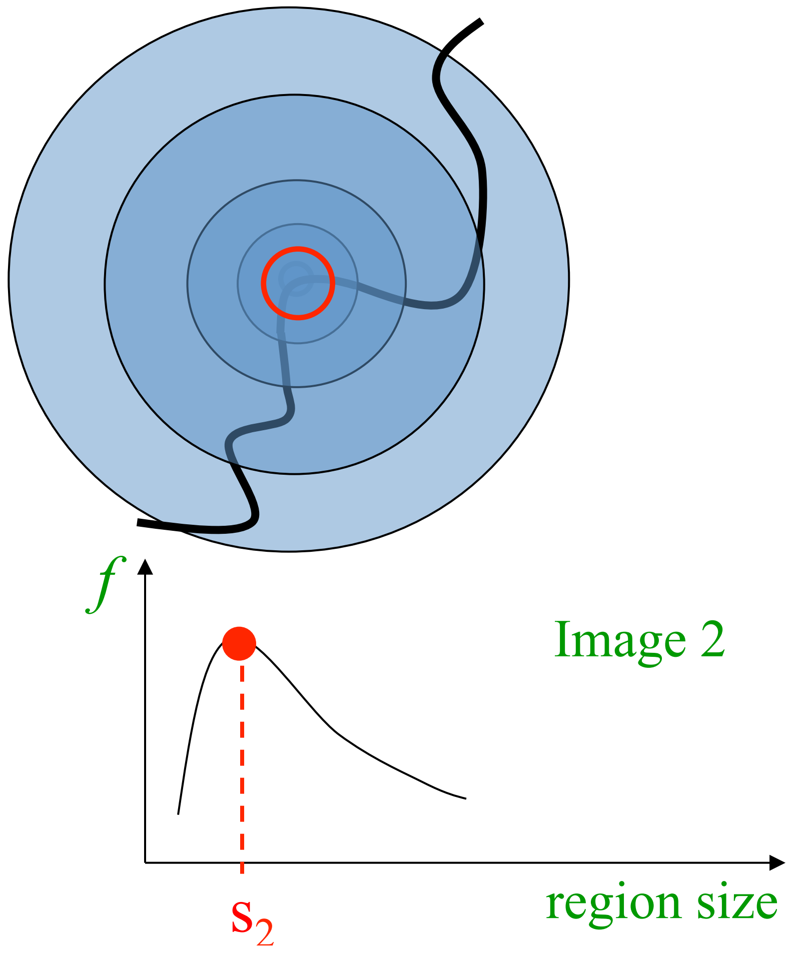
How can we independently select interest points in each image, such that the detections are repeatable across different scales?

Automatic scale selection

Intuition:

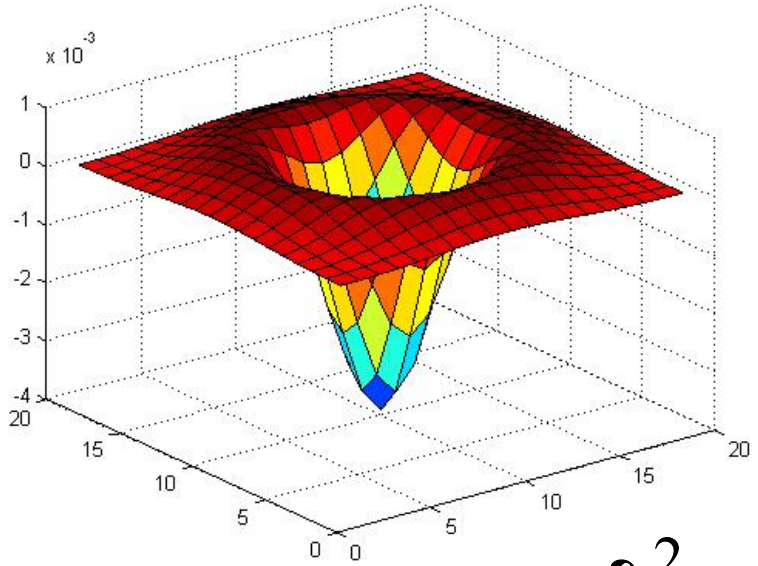
- Find scale that gives local maxima of some function *f* in both position and scale.

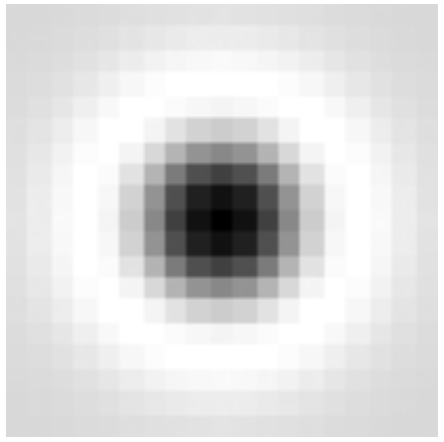




Blob detection in 2D

- Laplacian of Gaussian: Circularly symmetric operator for blob detection in 2D



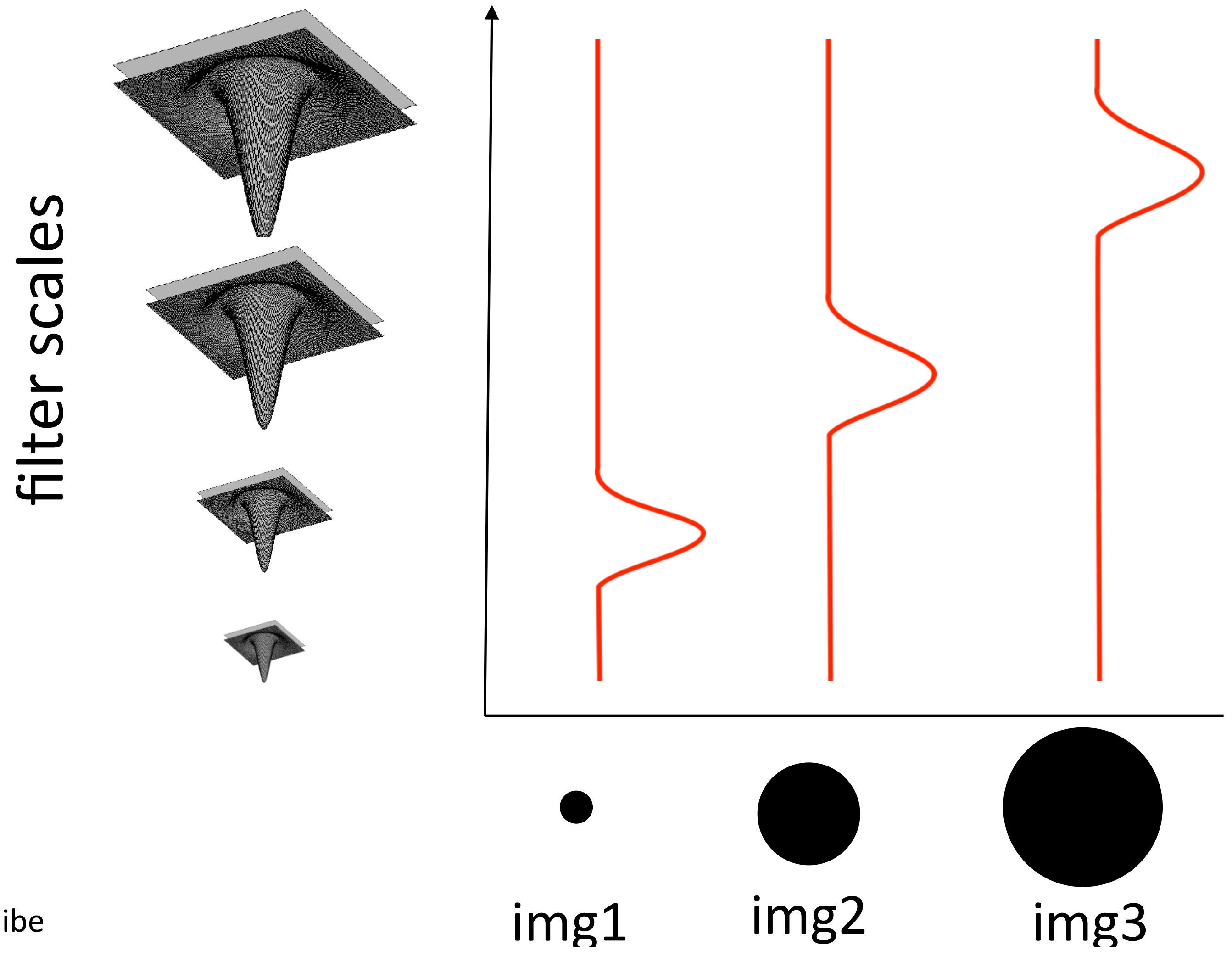


$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$

Blob detection in 2D: scale selection

- Laplacian-of-Gaussian = “blob” detector

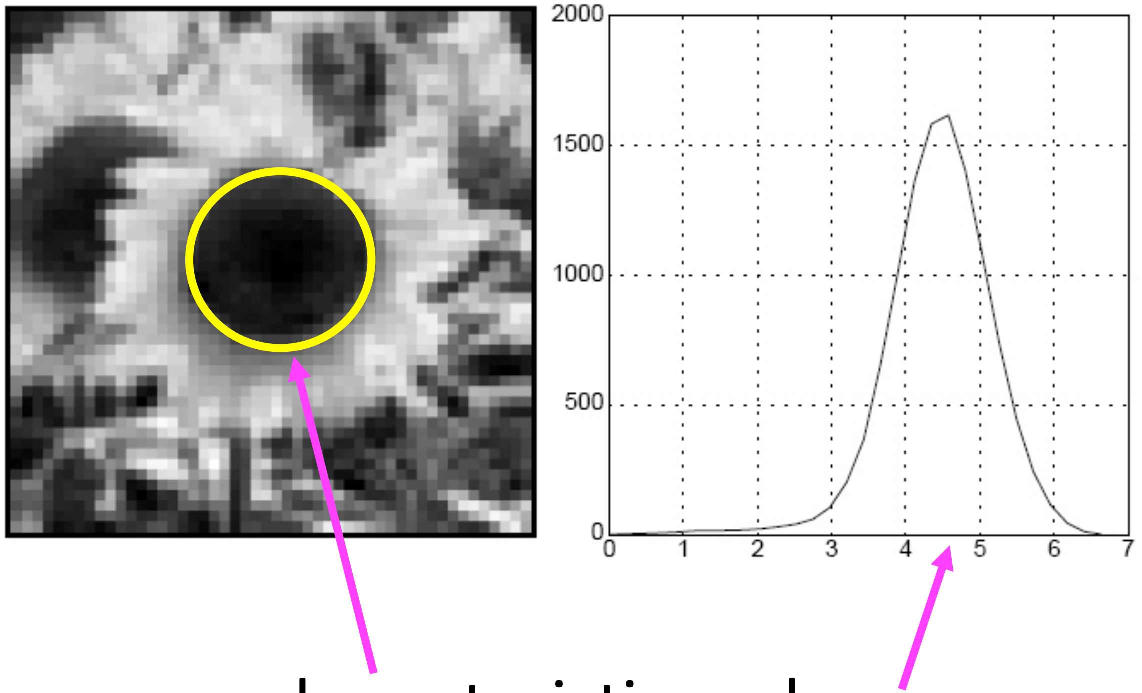
$$\nabla^2 g = \frac{\partial^2 g}{\partial x^2} + \frac{\partial^2 g}{\partial y^2}$$



Bastian Leibe

Blob detection in 2D

- We define the *characteristic scale* as the scale that produces peak of Laplacian response



characteristic scale

Slide credit: Lana Lazebnik

Image

This image is too big to fit on the screen. How can we generate a half-sized version?

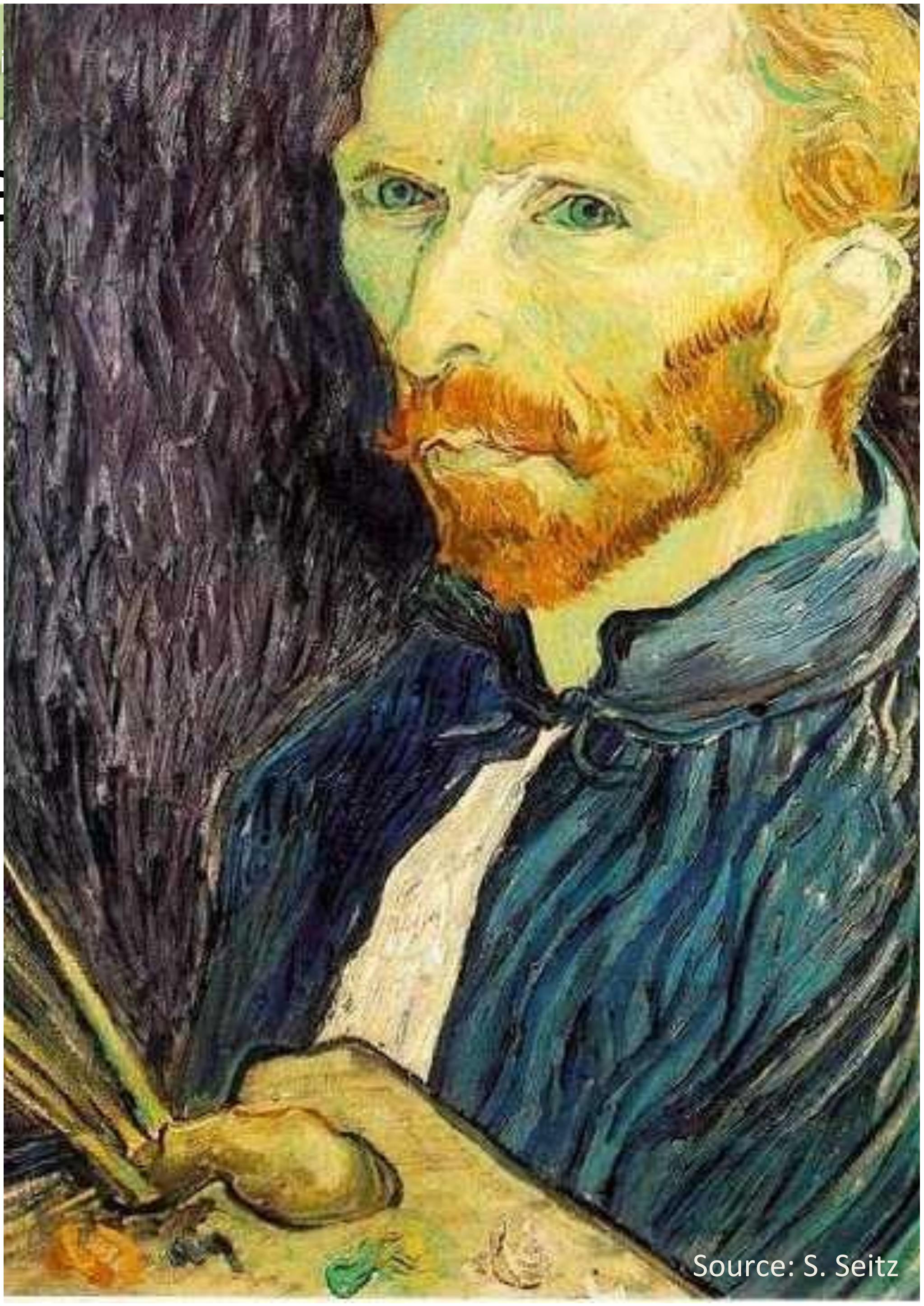
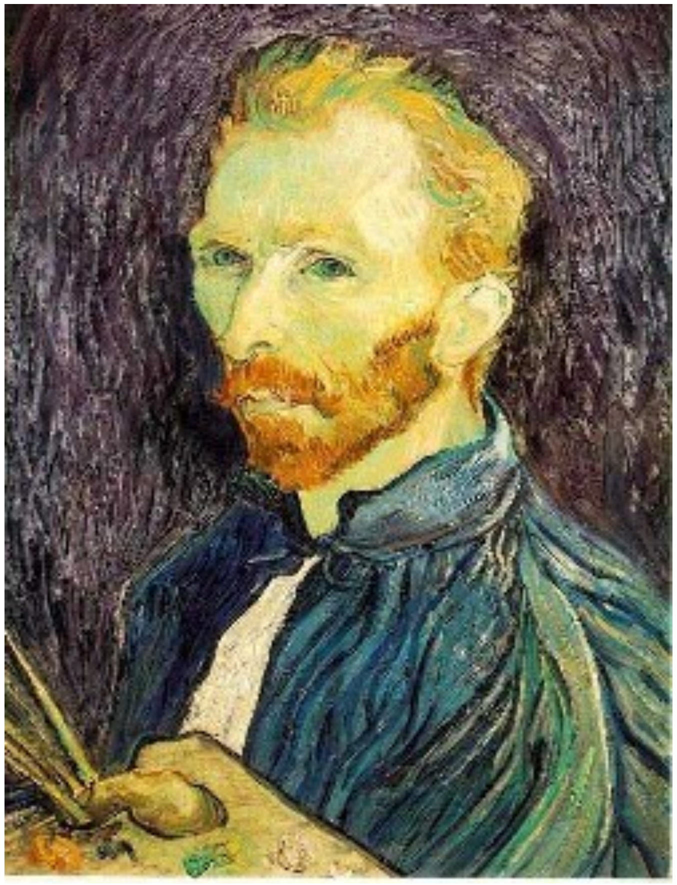
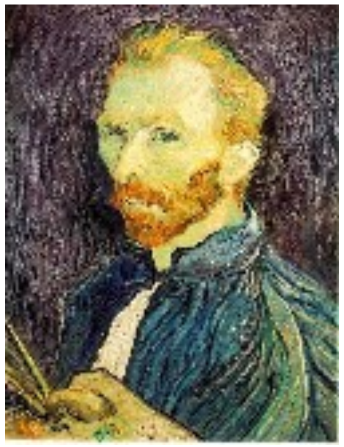
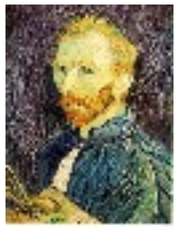


Image sub-sampling





1/4

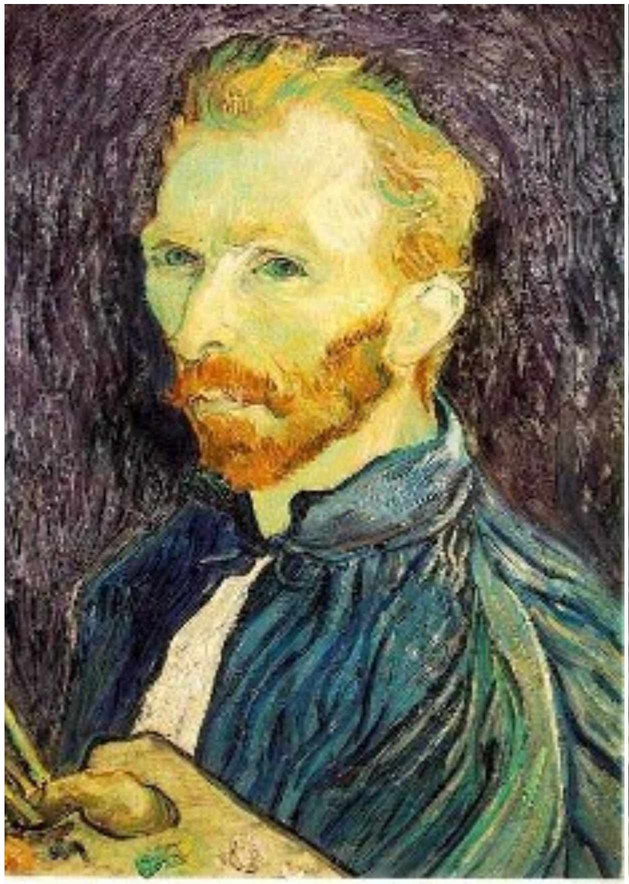


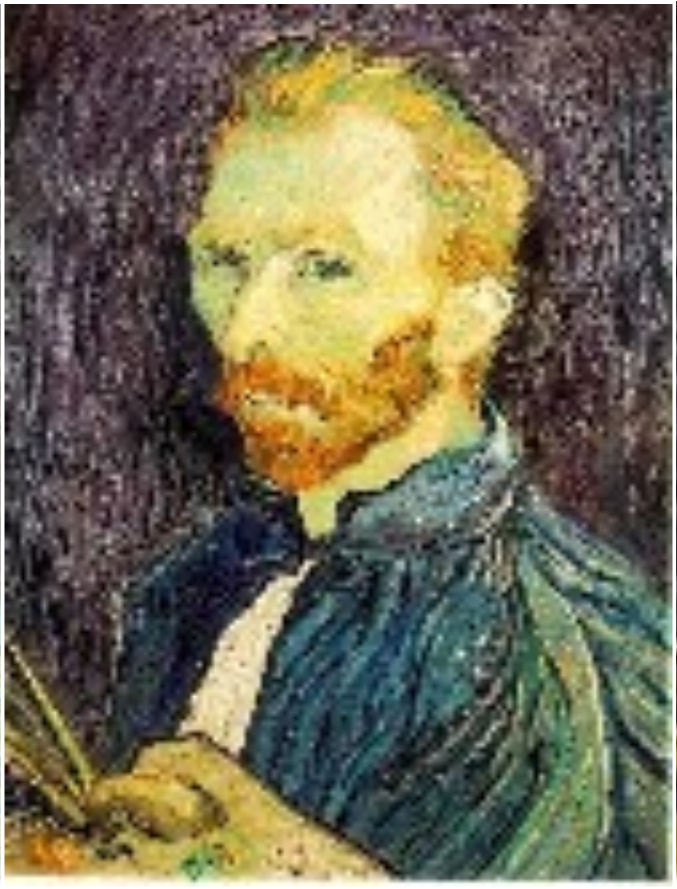
1/8

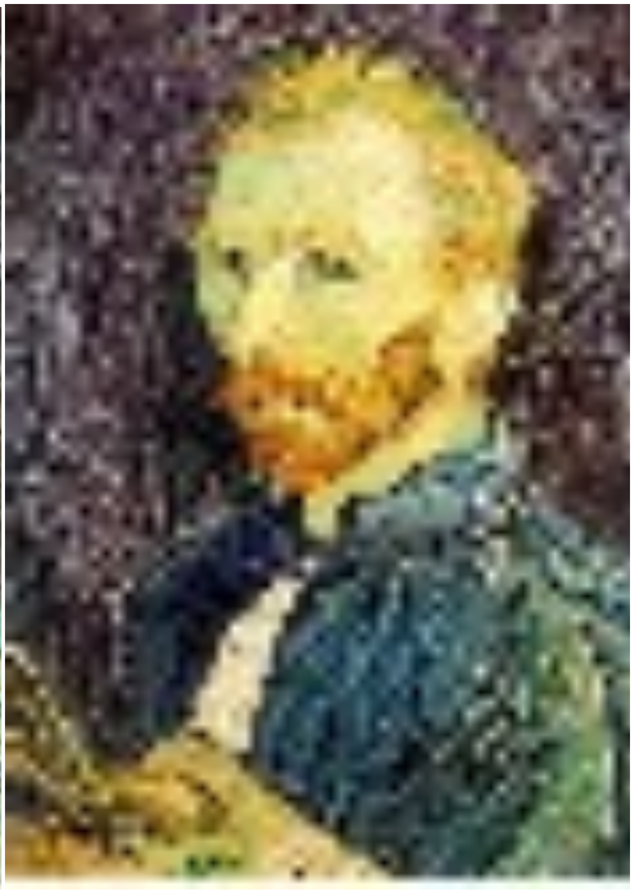
Throw away every other row and column to create a 1/2 size image
- called *image sub-sampling*

Source: S. Seitz

Image sub-sampling


1/2

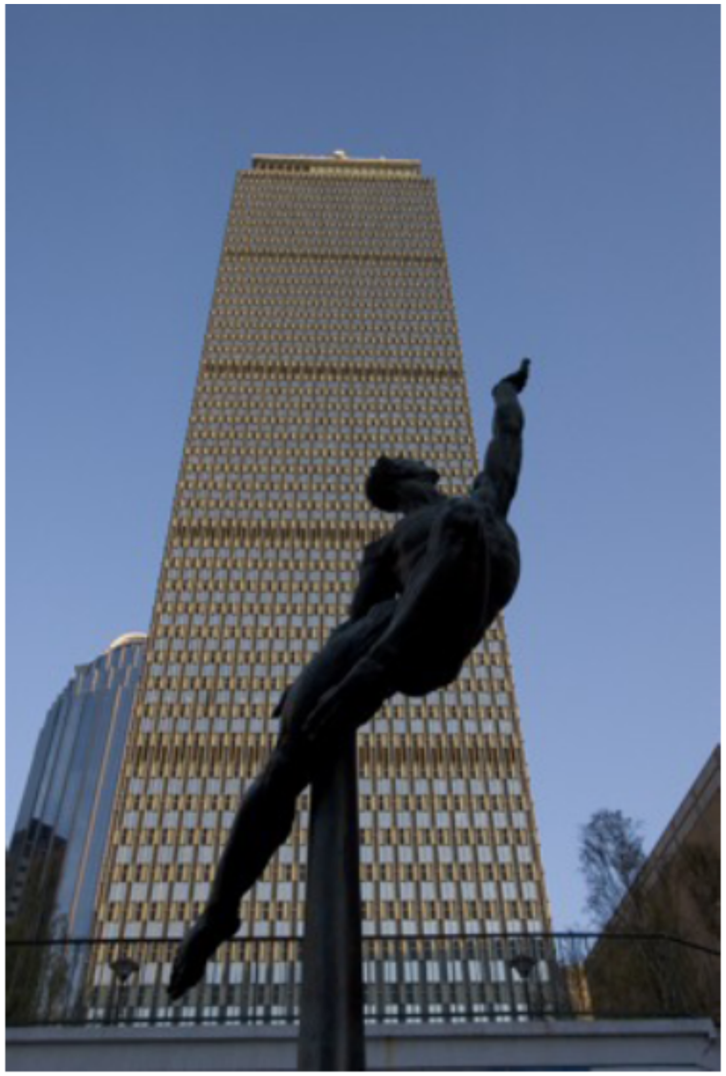

1/4 (2x zoom)

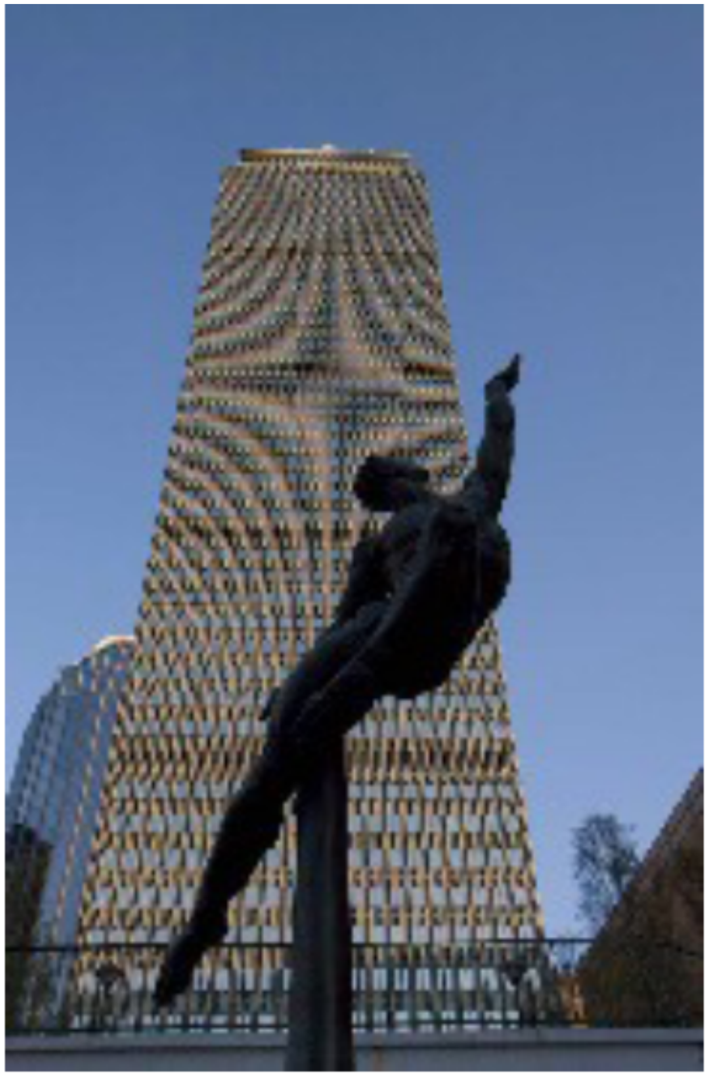

1/8 (4x zoom)

Why does this look so crufty?

Source: S. Seitz

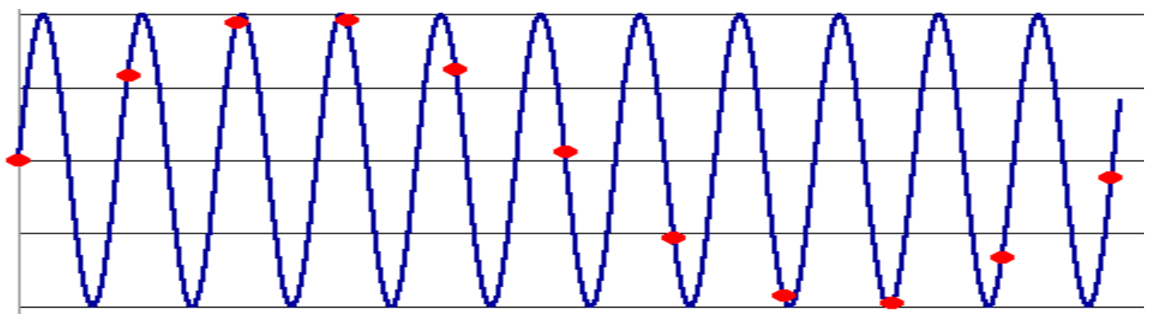
Image sub-sampling





Source: F. Durand

Aliasing



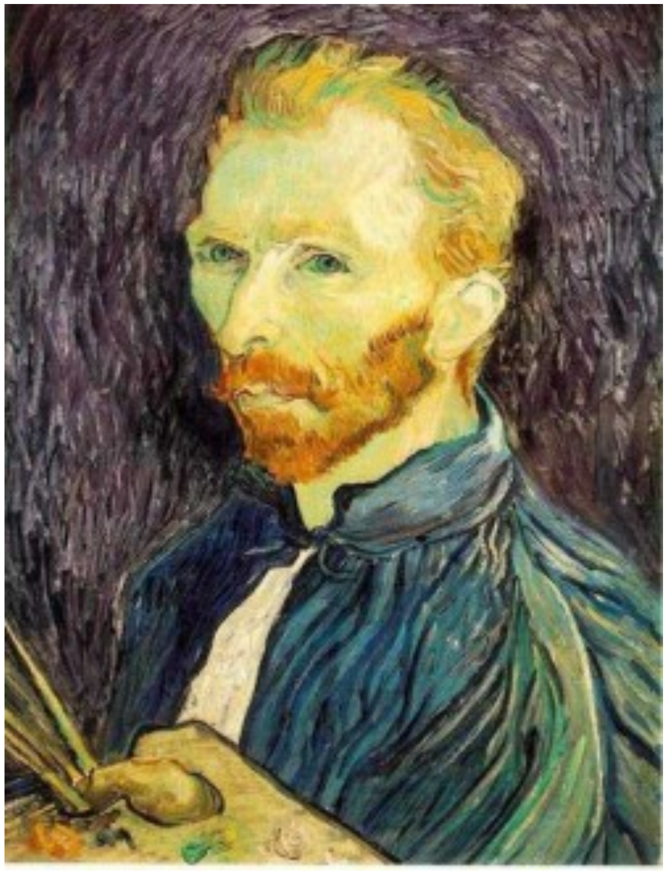
- Occurs when your sampling rate is not high enough to capture the amount of detail in your image
- Can give you the wrong signal/image—an *alias*

- To do sampling right, need to understand the structure of your signal/image
- Enter Monsieur Fourier…

- To avoid aliasing:
  - sampling rate ≥ 2 * max frequency in the image
    - said another way: ≥ two samples per cycle
  - This minimum sampling rate is called the **Nyquist rate**

Source: L. Zhang

Gaussian pre-filtering



Gaussian 1/2

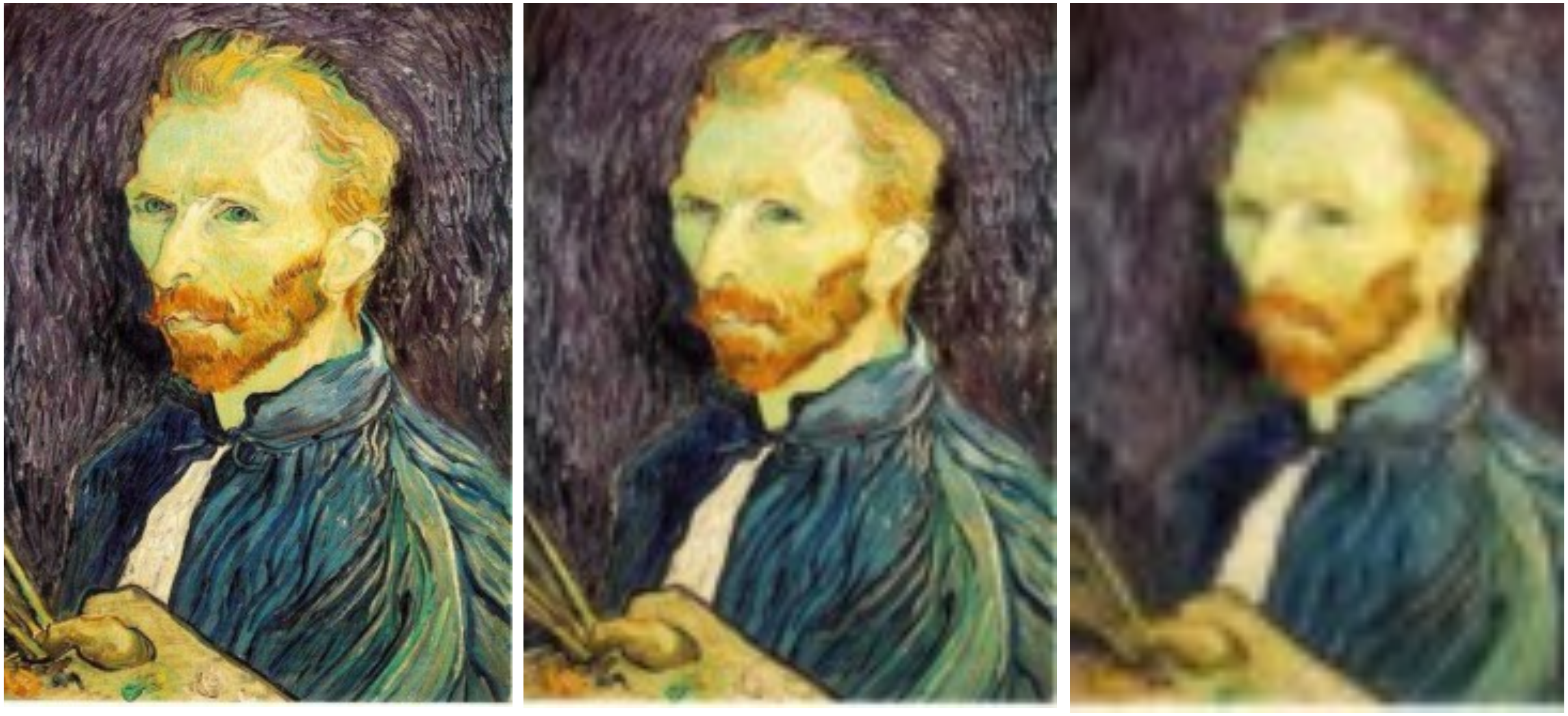
G 1/4

G 1/8

- Solution: filter the image, *then* subsample

Source: S. Seitz

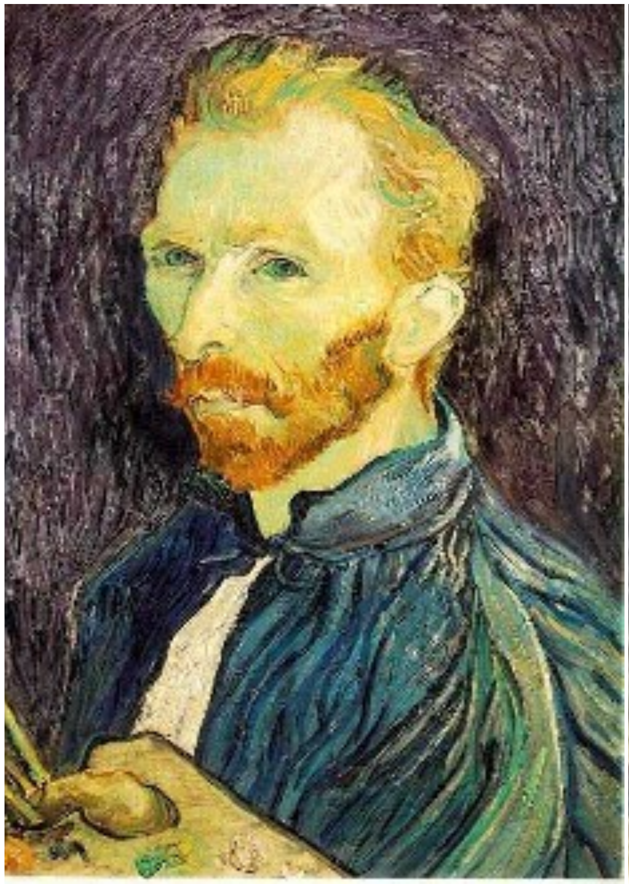
Subsampling with Gaussian pre-filtering

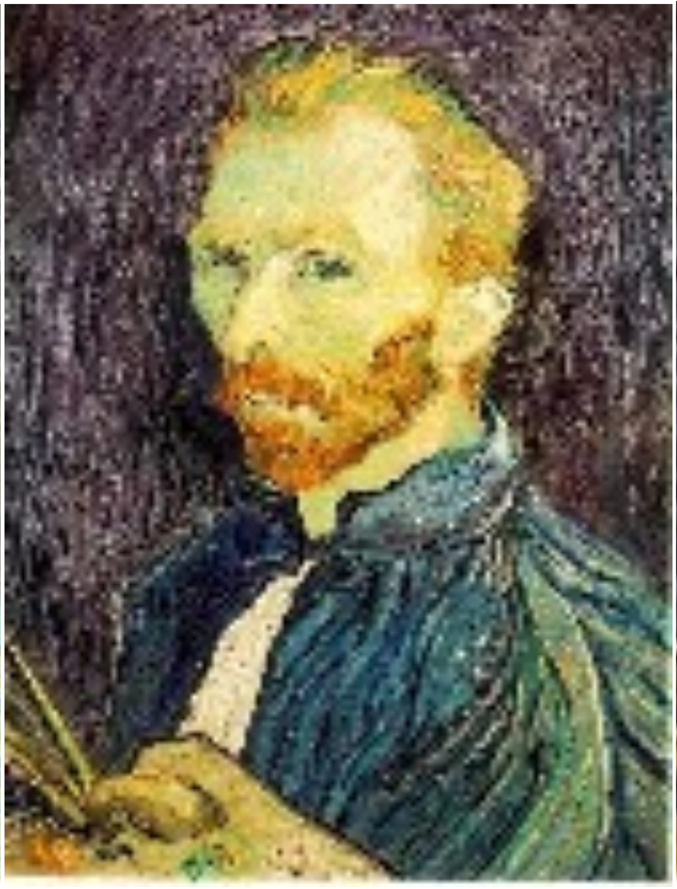

Gaussian 1/2 G 1/4 G 1/8

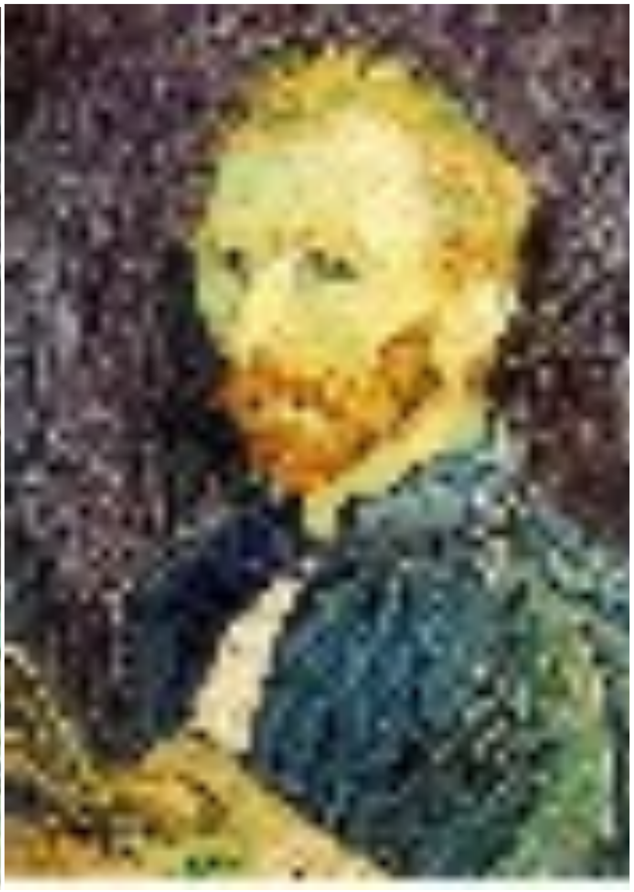
- Solution: filter the image, *then* subsample

Source: S. Seitz

Compare with...


1/2

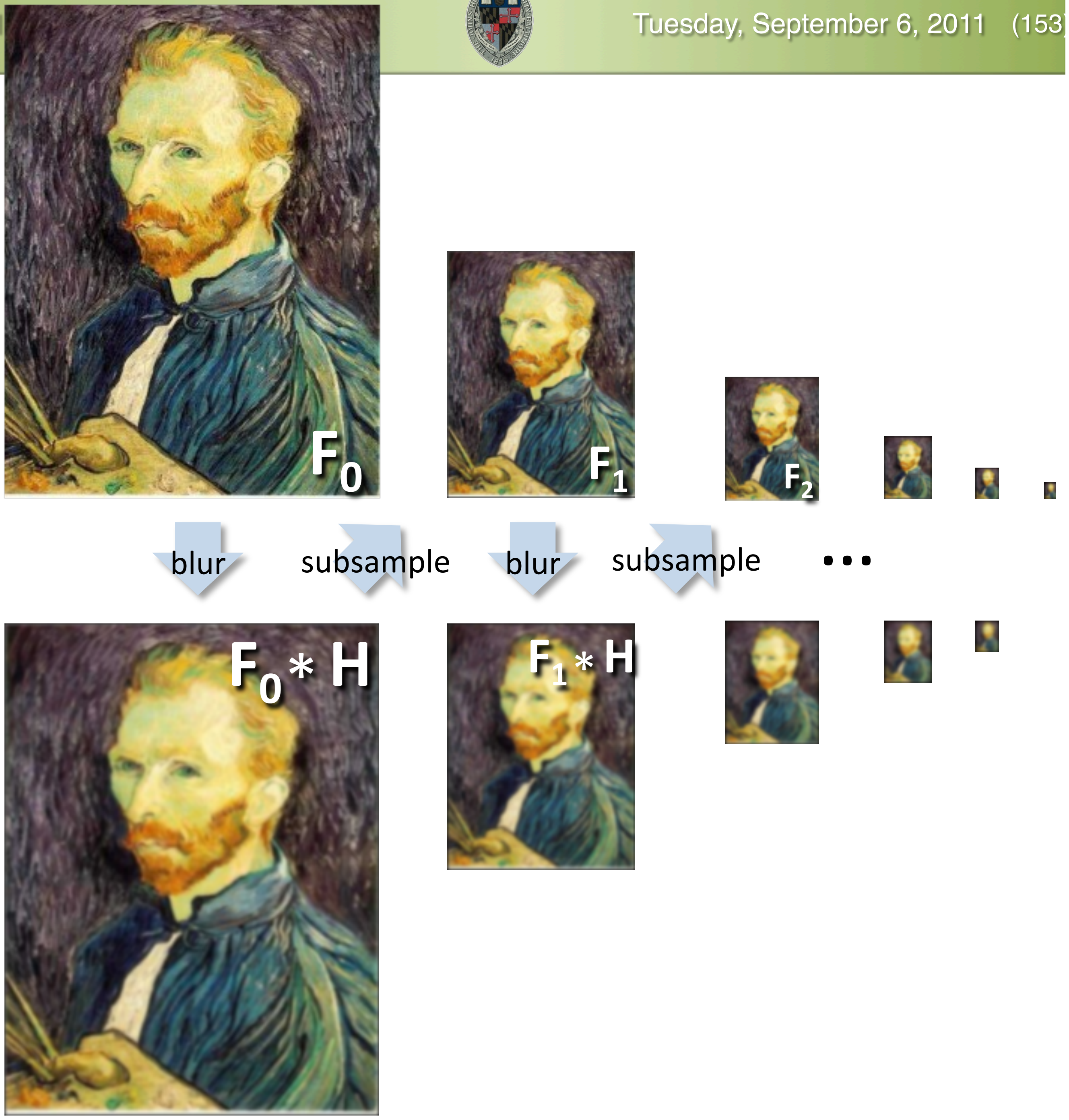

1/4 (2x zoom)


1/8 (4x zoom)

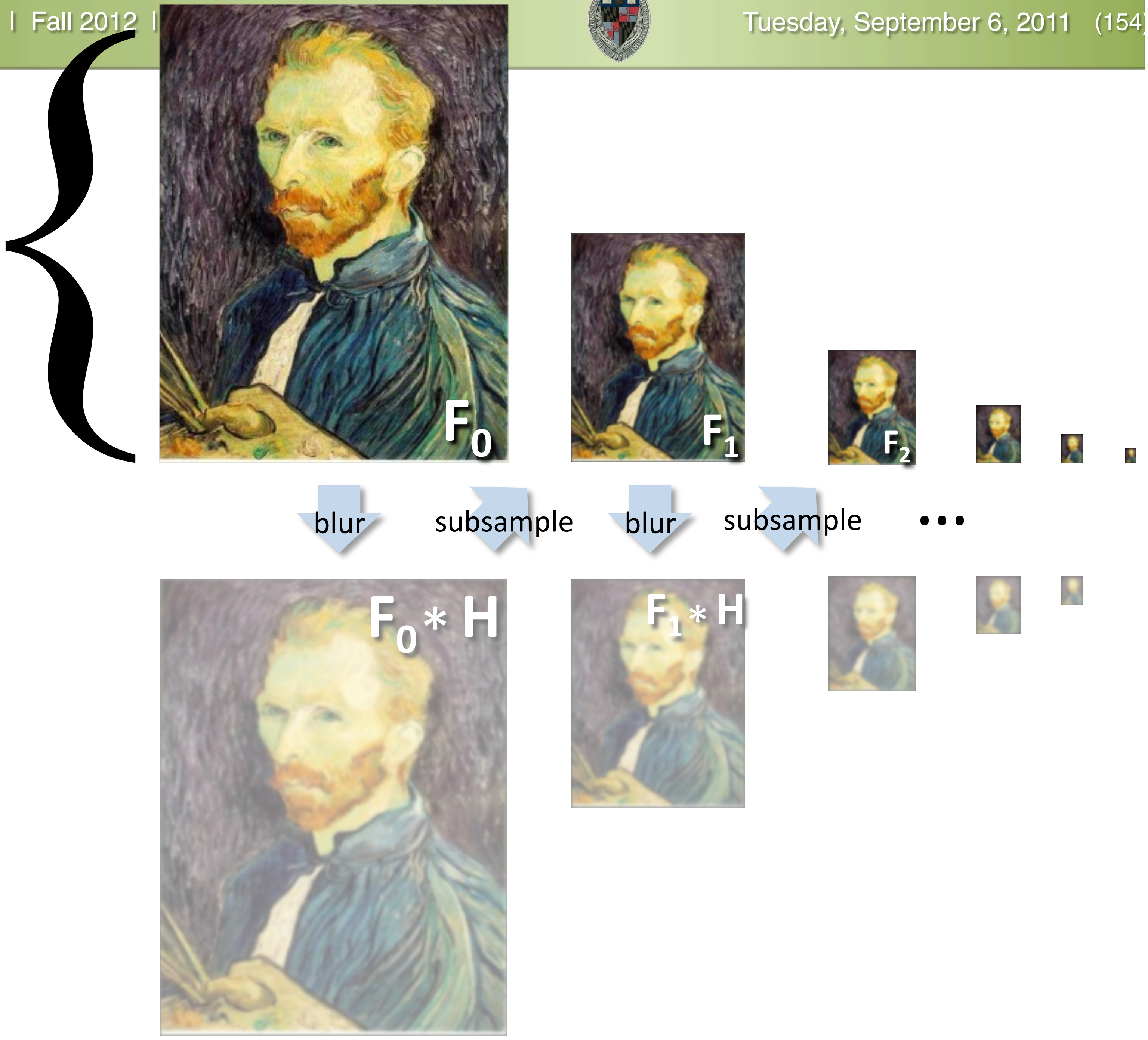
Source: S. Seitz

Gaussian pre-filtering

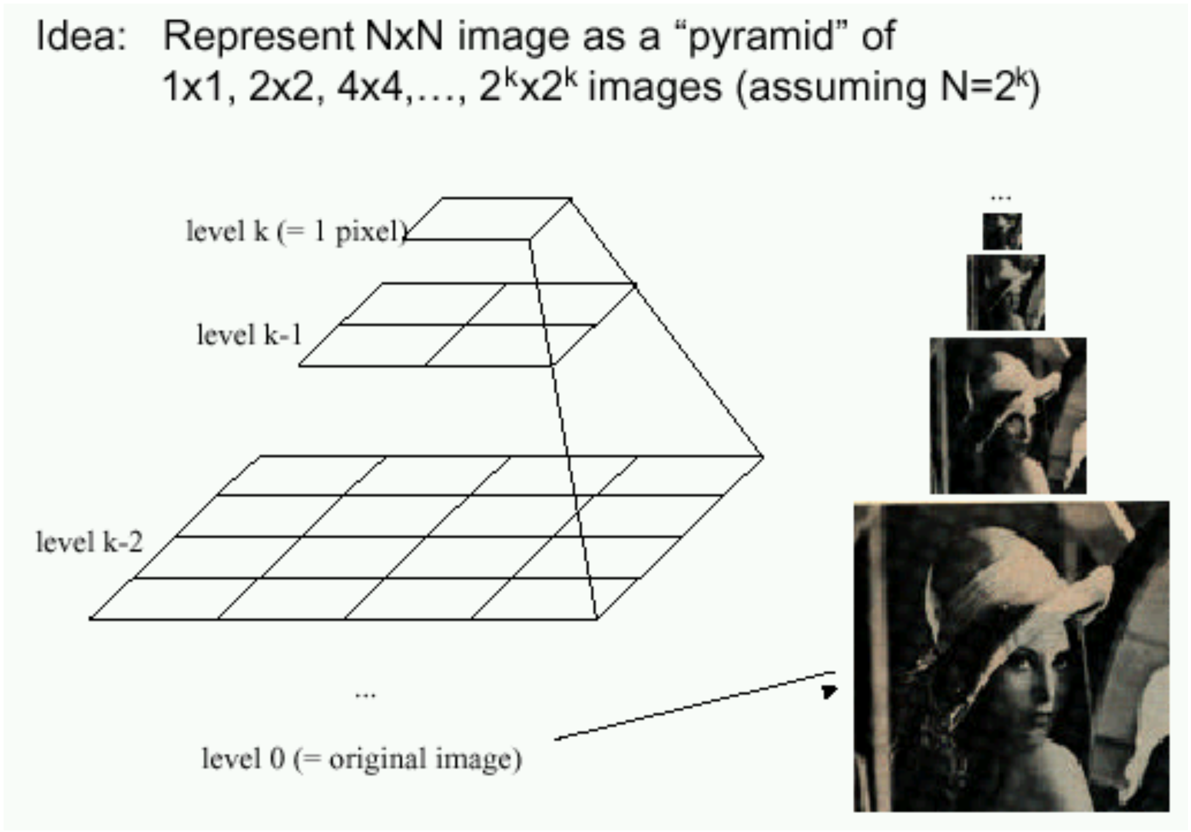
- Solution: filter the image, *then* subsample



Gaussian pyramid



Gaussian pyramids [Burt and Adelson, 1983]

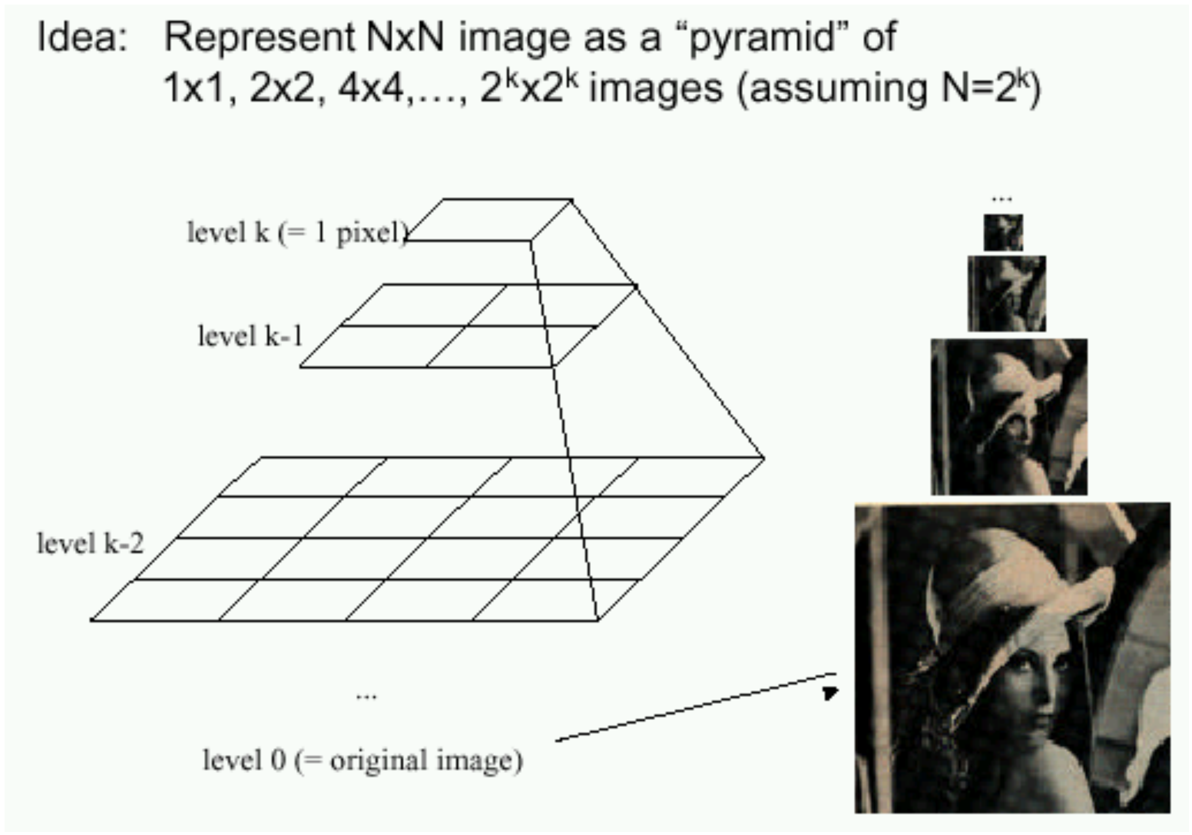

Idea: Represent NxN image as a "pyramid" of 1x1, 2x2, 4x4,..., 2^kx2^k images (assuming N=2^k)
level k (= 1 pixel)
level k-1
level k-2
...
level 0 (= original image)
...

- In computer graphics, a *mip map* [Williams, 1983]
- A precursor to *wavelet transform*

Gaussian Pyramids have all sorts of applications in computer vision

Source: S. Seitz

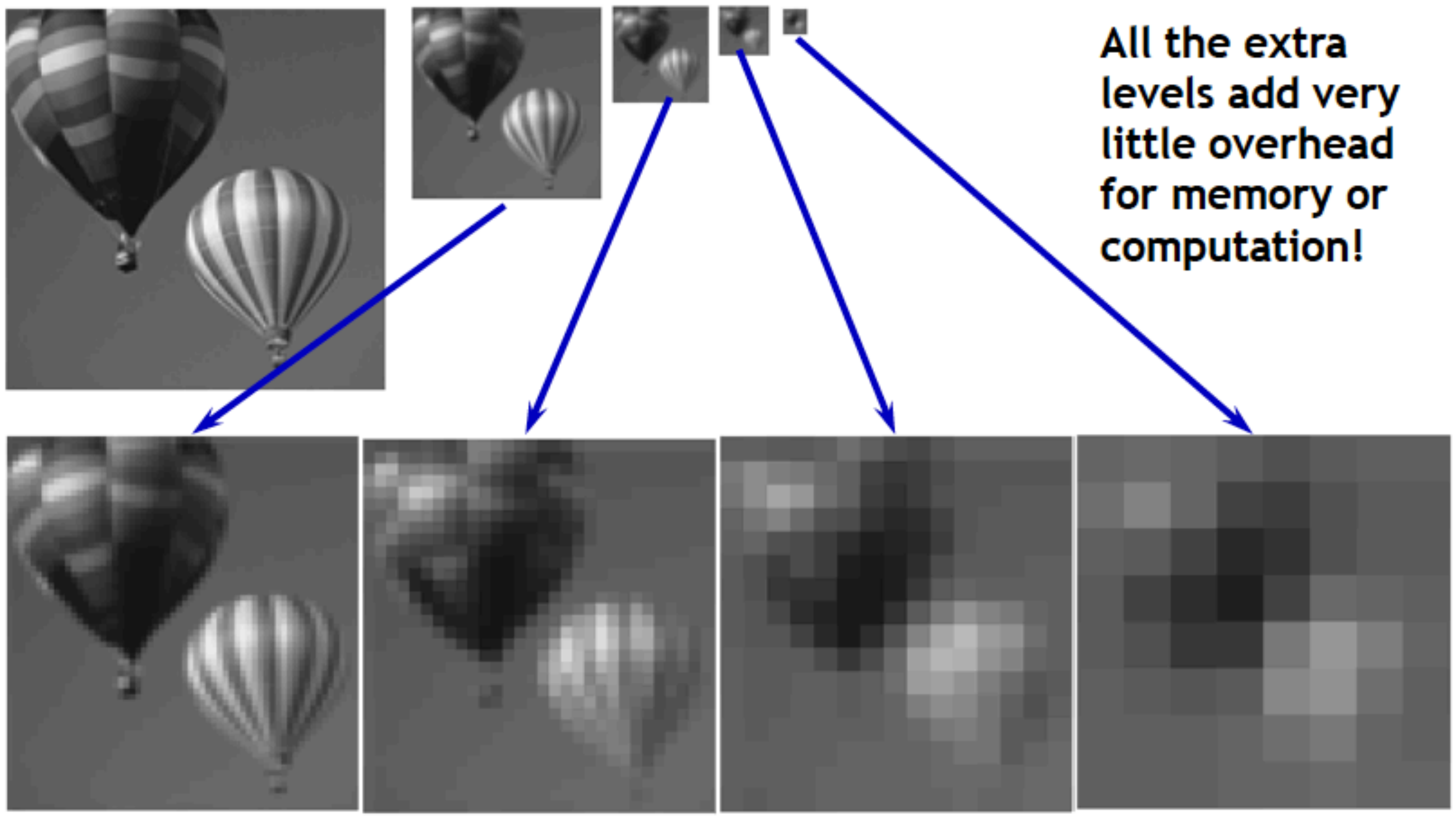
Gaussian pyramids [Burt and Adelson, 1983]


Idea: Represent NxN image as a “pyramid” of 1x1, 2x2, 4x4,…, 2^kx2^k images (assuming N=2^k)

- How much space does a Gaussian pyramid take compared to the original image?

Source: S. Seitz

Gaussian pyramid example



All the extra levels add very little overhead for memory or computation!

Source: Irani & Basri

One Useful Feature of GPs

- We can approximate the Laplacian by taking the differences of Gaussians at different scales (sigmas)

- Note the Laplacian pyramid already has that

- So, taking differences of adjacent levels (sampled to the same size) gives us the Laplacian of Gaussian "for free"

Scale Invariant Feature Transform (SIFT)

Distinctive image features from scale-invariant keypoints. David G. Lowe, International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

Stanford CS223B Computer Vision, Winter 2005

Lecture 4
Advanced Features

Sebastian Thrun, Stanford

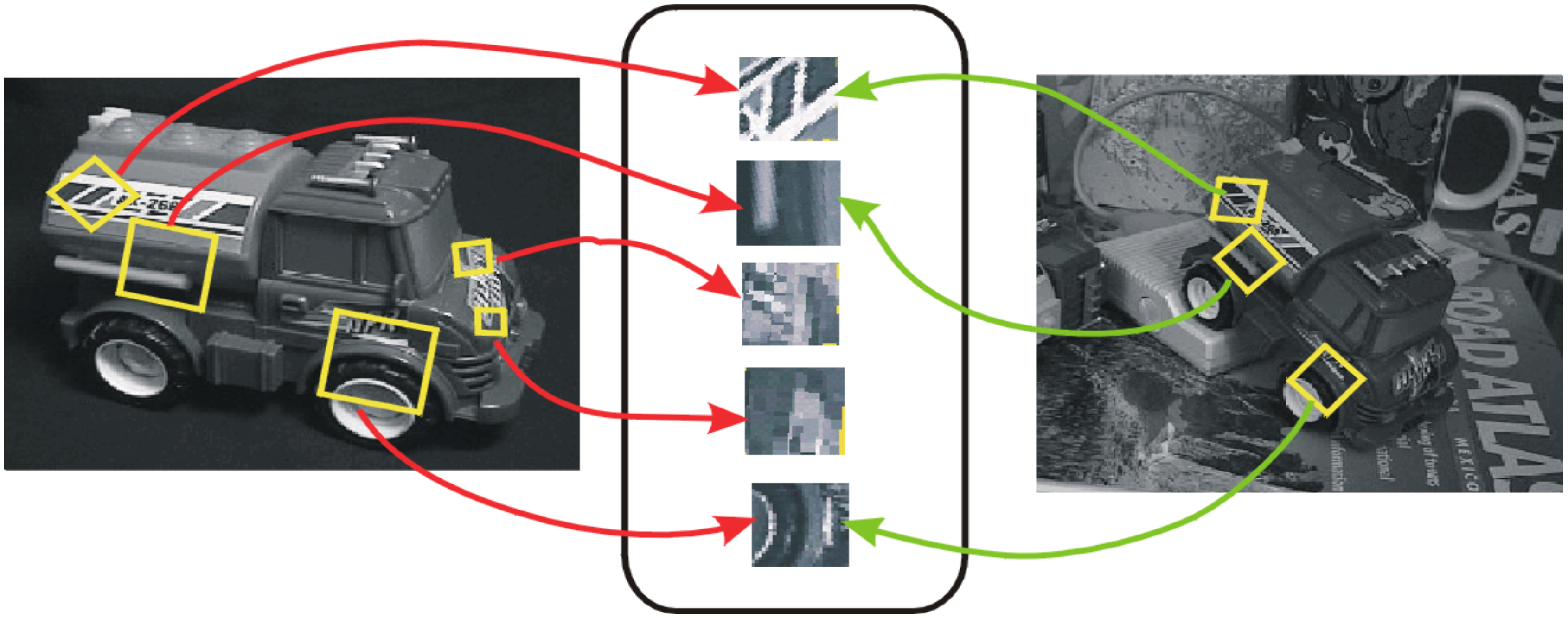
Rick Szeliski, Microsoft

Hendrik Dahlkamp, Stanford

[with slides by D Lowe, M. Polleyfeys]

Invariant Local Features

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



SIFT Features

SIFT On-A-Slide

1. **Enforce invariance to scale:** Compute Gaussian difference max, for may different scales; non-maximum suppression, find local maxima: keypoint candidates
2. **Localizable corner:** For each maximum fit quadratic function. Compute center with sub-pixel accuracy by setting first derivative to zero.
3. **Eliminate edges:** Compute ratio of eigenvalues, drop keypoints for which this ratio is larger than a threshold.
4. **Enforce invariance to orientation:** Compute orientation, to achieve scale invariance, by finding the strongest second derivative direction in the smoothed image (possibly multiple orientations). Rotate patch so that orientation points up.
5. **Compute feature signature:** Compute a "gradient histogram" of the local image region in a 4x4 pixel region. Do this for 4x4 regions of that size. Orient so that largest gradient points up (possibly multiple solutions). Result: feature vector with 128 values (15 fields, 8 gradients).
6. **Enforce invariance to illumination change and camera saturation:** Normalize to unit length to increase invariance to illumination. Then threshold all gradients, to become invariant to camera saturation.

SIFT On-A-Slide

1. **Enforce invariance to scale:** **Compute Gaussian difference max, for may different scales; non-maximum suppression, find local maxima: keypoint candidates**
2. **Localizable corner:** **For each maximum fit quadratic function. Compute center with sub-pixel accuracy by setting first derivative to zero.**
3. **Eliminate edges:** **Compute ratio of eigenvalues, drop keypoints for which this ratio is larger than a threshold.**
4. **Enforce invariance to orientation:** **Compute orientation, to achieve scale invariance, by finding the strongest second derivative direction in the smoothed image (possibly multiple orientations). Rotate patch so that orientation points up.**
5. **Compute feature signature:** **Compute a "gradient histogram" of the local image region in a 4x4 pixel region. Do this for 4x4 regions of that size. Orient so that largest gradient points up (possibly multiple solutions). Result: feature vector with 128 values (15 fields, 8 gradients).**
6. **Enforce invariance to illumination change and camera saturation:** **Normalize to unit length to increase invariance to illumination. Then threshold all gradients, to become invariant to camera saturation.**

Find Invariant Corners

1. **Enforce invariance to scale: Compute Gaussian difference max, for may different scales; non-maximum suppression, find local maxima: keypoint candidates**

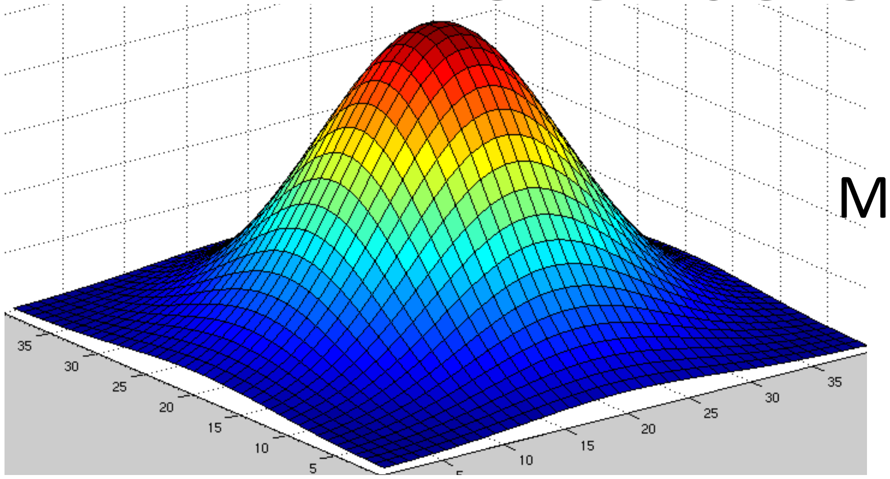
Finding “Keypoints” (Corners)

Idea: Find Corners, but scale invariance

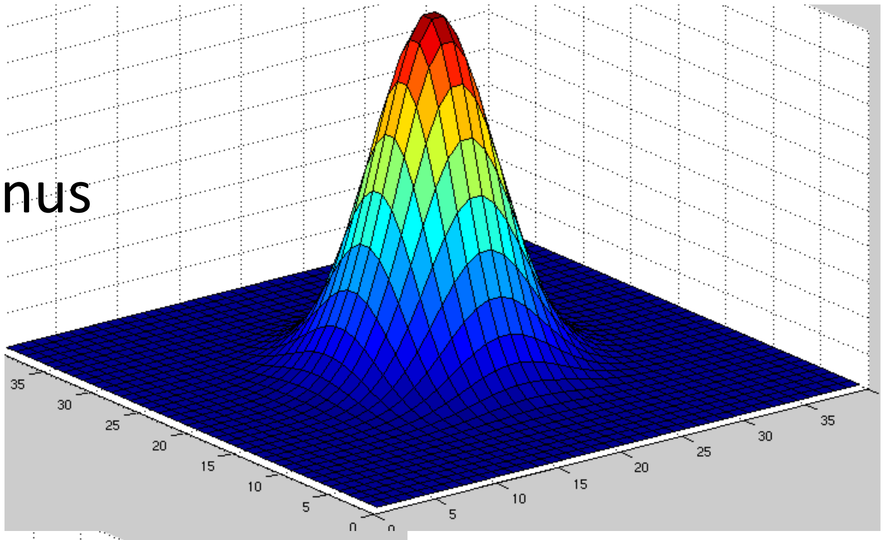
Approach:

- Run linear filter (diff of Gaussians)
- At different resolutions of image pyramid

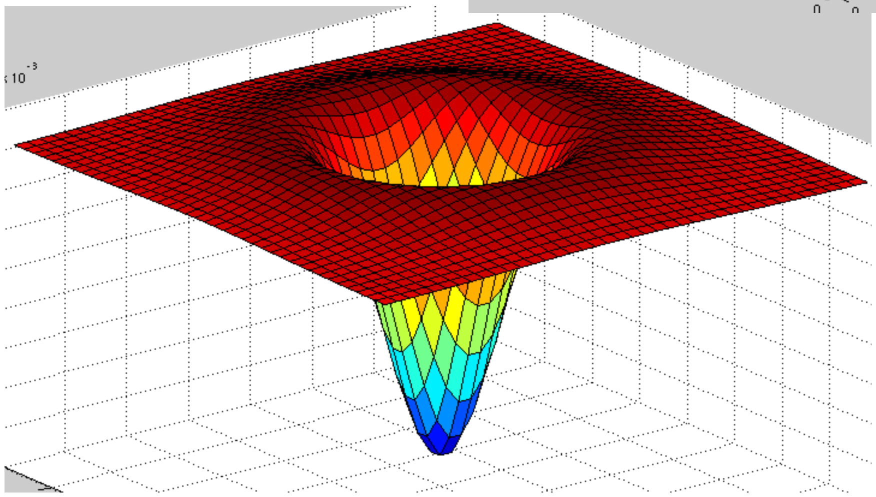
Difference of Gaussians



Minus

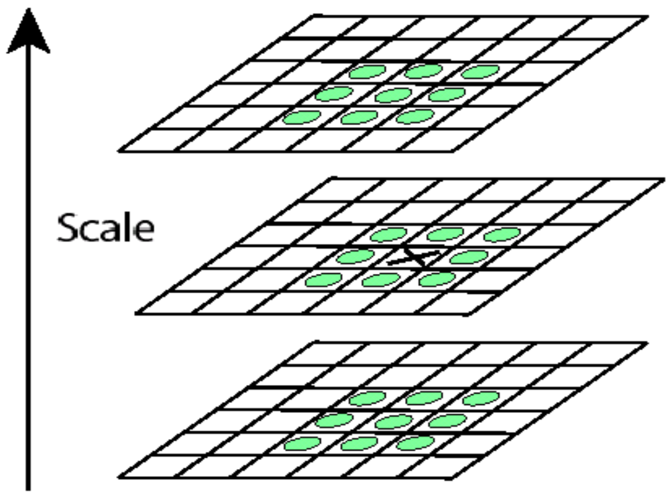


Equals

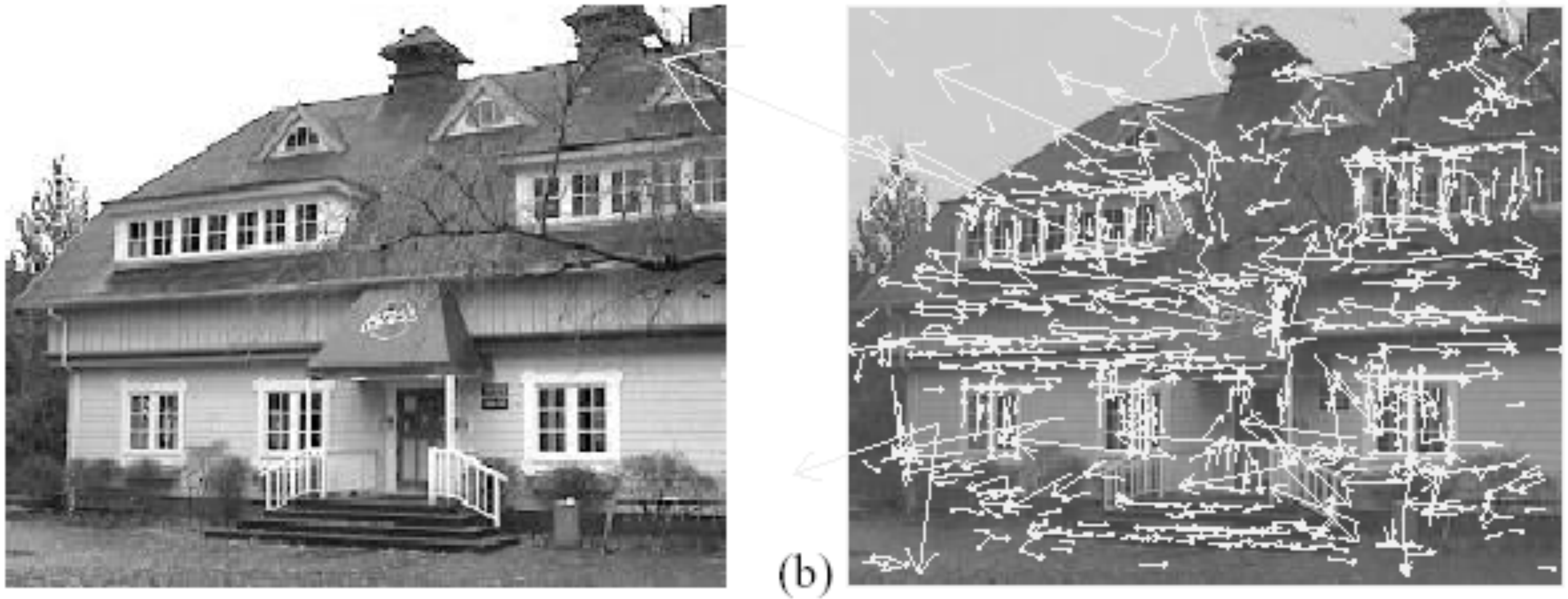


Key point localization

- Detect maxima and minima of difference-of-Gaussian in scale space



Example of keypoint detection

(a)  (b)

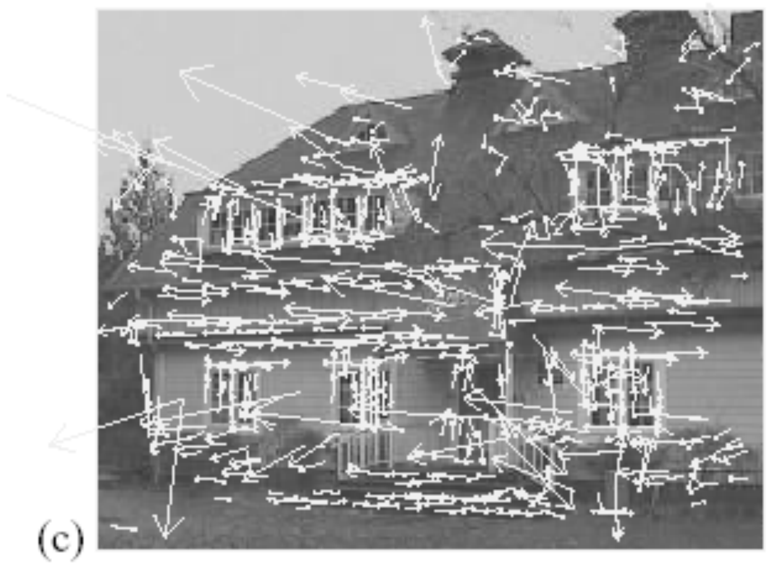
(a) 233x189 image
(b) 832 DOG extrema

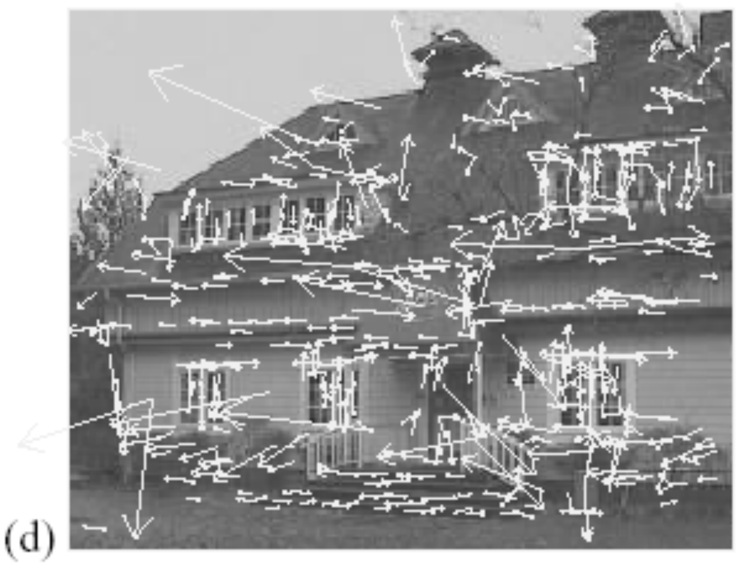
SIFT On-A-Slide

1. **Enforce invariance to scale:** Compute Gaussian difference max, for may different scales; non-maximum suppression, find local maxima: keypoint candidates
2. **Localizable corner:** For each maximum fit quadratic function. Compute center with sub-pixel accuracy by setting first derivative to zero.
3. **Eliminate edges:** Compute ratio of eigenvalues, drop keypoints for which this ratio is larger than a threshold.
4. **Enforce invariance to orientation:** Compute orientation, to achieve scale invariance, by finding the strongest second derivative direction in the smoothed image (possibly multiple orientations). Rotate patch so that orientation points up.
5. **Compute feature signature:** Compute a "gradient histogram" of the local image region in a 4x4 pixel region. Do this for 4x4 regions of that size. Orient so that largest gradient points up (possibly multiple solutions). Result: feature vector with 128 values (15 fields, 8 gradients).
6. **Enforce invariance to illumination change and camera saturation:** Normalize to unit length to increase invariance to illumination. Then threshold all gradients, to become invariant to camera saturation.

Example of keypoint detection

Threshold on value at DOG peak and on ratio of principle curvatures (Harris approach)


(c)


(d)

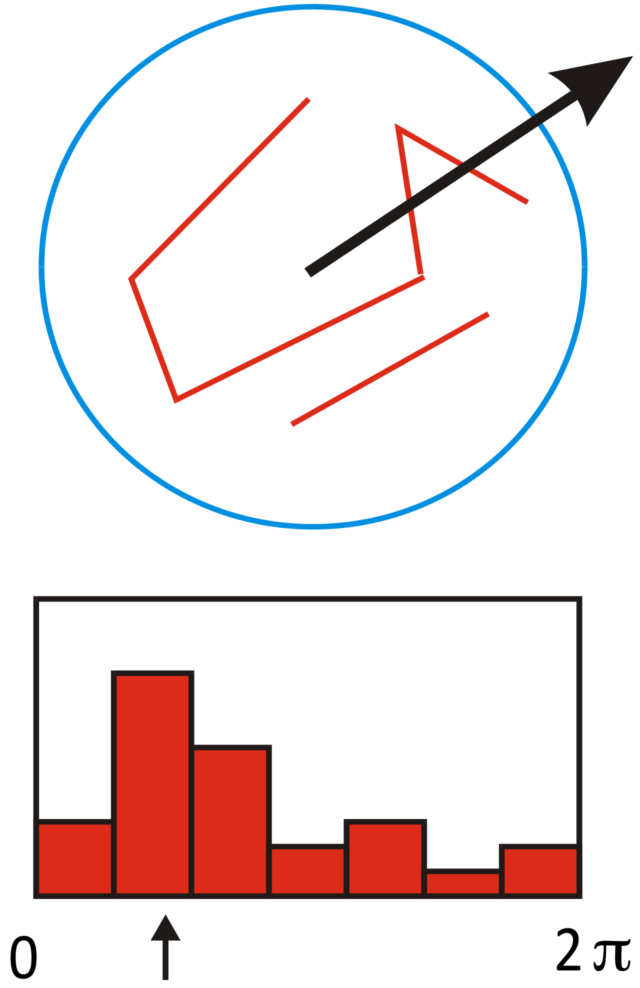
(c) 729 left after peak value threshold
(d) 536 left after testing ratio of principle curvatures

SIFT On-A-Slide

1. **Enforce invariance to scale:** Compute Gaussian difference max, for may different scales; non-maximum suppression, find local maxima: keypoint candidates
2. **Localizable corner:** For each maximum fit quadratic function. Compute center with sub-pixel accuracy by setting first derivative to zero.
3. **Eliminate edges:** Compute ratio of eigenvalues, drop keypoints for which this ratio is larger than a threshold.
4. **Enforce invariance to orientation:** Compute orientation, to achieve scale invariance, by finding the strongest second derivative direction in the smoothed image (possibly multiple orientations). Rotate patch so that orientation points up.
5. **Compute feature signature:** Compute a "gradient histogram" of the local image region in a 4x4 pixel region. Do this for 4x4 regions of that size. Orient so that largest gradient points up (possibly multiple solutions). Result: feature vector with 128 values (15 fields, 8 gradients).
6. **Enforce invariance to illumination change and camera saturation:** Normalize to unit length to increase invariance to illumination. Then threshold all gradients, to become invariant to camera saturation.

Select canonical orientation

- Create histogram of local gradient directions computed at selected scale
- Assign canonical orientation at peak of smoothed histogram
- Each key specifies stable 2D coordinates (x, y, scale, orientation)

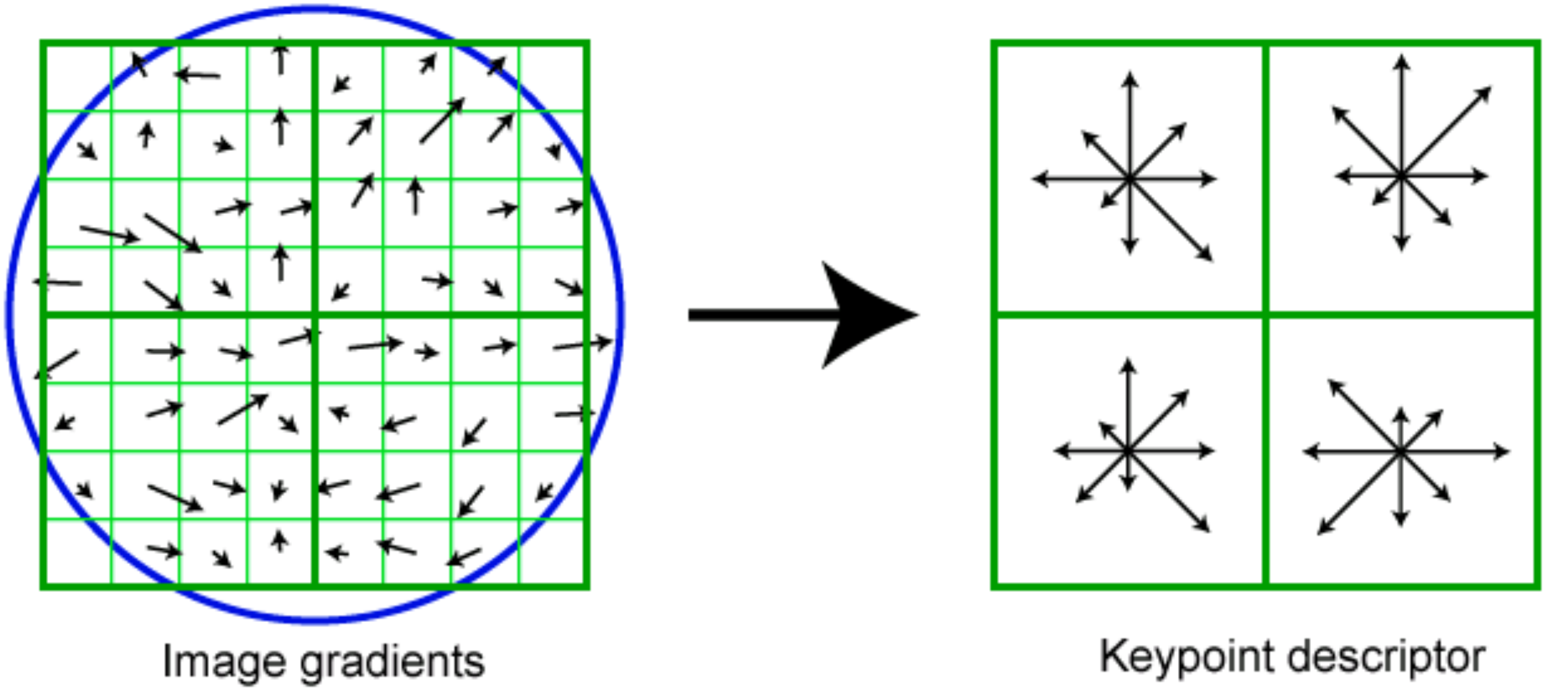


SIFT On-A-Slide

1. **Enforce invariance to scale: Compute Gaussian difference max, for may different scales; non-maximum suppression, find local maxima: keypoint candidates**
2. **Localizable corner: For each maximum fit quadratic function. Compute center with sub-pixel accuracy by setting first derivative to zero.**
3. **Eliminate edges: Compute ratio of eigenvalues, drop keypoints for which this ratio is larger than a threshold.**
4. **Enforce invariance to orientation: Compute orientation, to achieve scale invariance, by finding the strongest second derivative direction in the smoothed image (possibly multiple orientations). Rotate patch so that orientation points up.**
5. **Compute feature signature: Compute a "gradient histogram" of the local image region in a 4x4 pixel region. Do this for 4x4 regions of that size. Orient so that largest gradient points up (possibly multiple solutions). Result: feature vector with 128 values (15 fields, 8 gradients).**
6. **Enforce invariance to illumination change and camera saturation: Normalize to unit length to increase invariance to illumination. Then threshold all gradients, to become invariant to camera saturation.**

SIFT vector formation

- Thresholded image gradients are sampled over 16x16 array of locations in scale space
- Create array of orientation histograms
- 8 orientations x 4x4 histogram array = 128 dimensions



Match Scaling

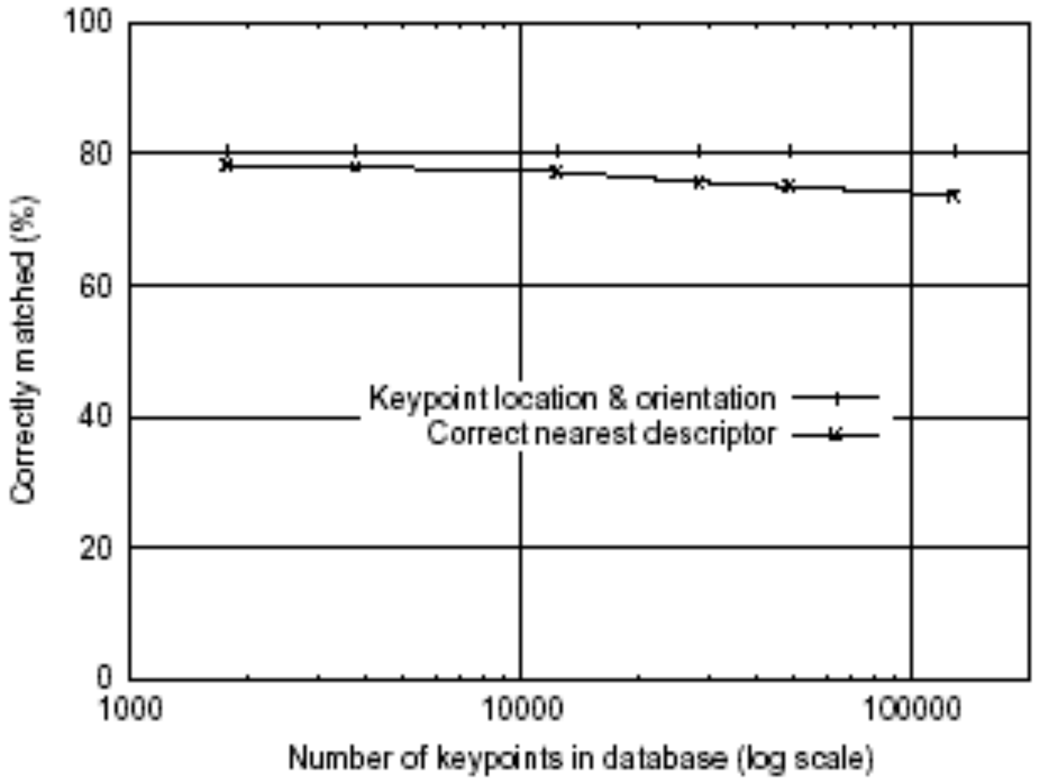
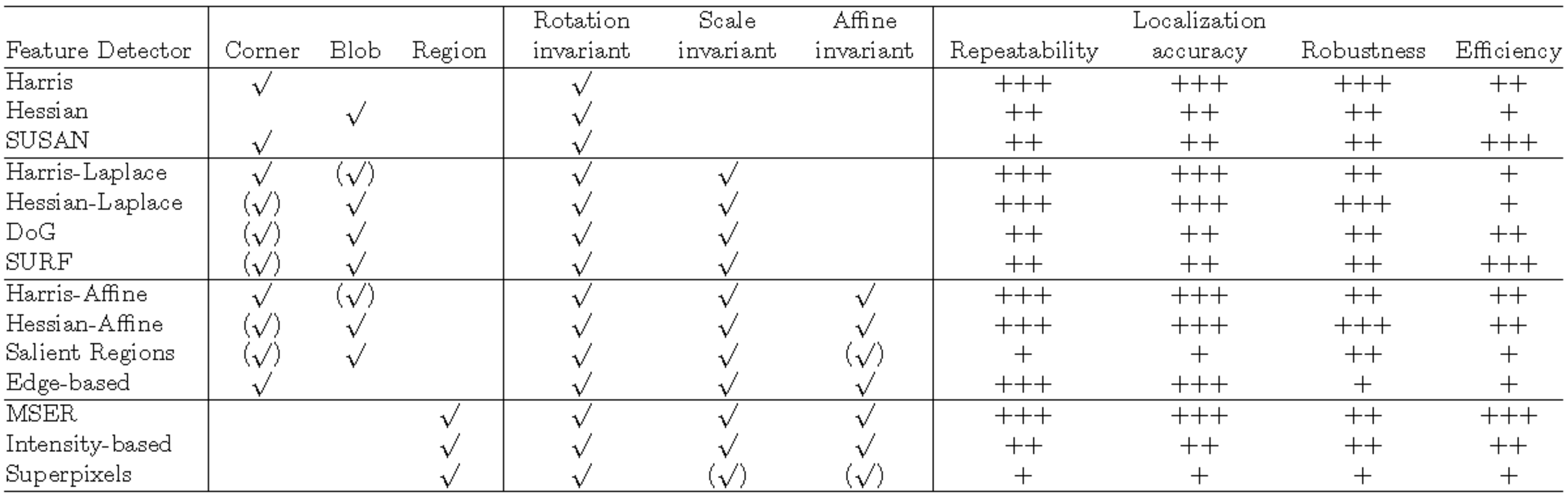


Figure 10: The dashed line shows the percent of keypoints correctly matched to a database as a function of database size (using a logarithmic scale). The solid line shows the percent of keypoints assigned the correct location and orientation.

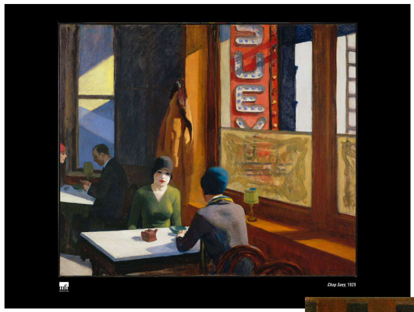
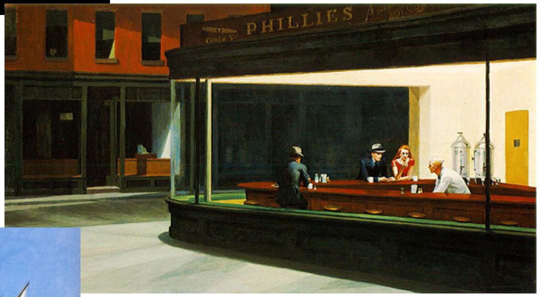
Overview of feature detectors



Feature Detector	Corner	Blob	Region	Rotation invariant	Scale invariant	Affine invariant	Repeatability	Localization accuracy	Robustness	Efficiency
Harris	√			√			+++	+++	+++	++
Hessian		√		√			++	++	++	+
SUSAN	√			√			++	++	++	+++
Harris-Laplace	√	(√)		√	√		+++	+++	++	+
Hessian-Laplace	(√)	√		√	√		+++	+++	+++	+
DoG	(√)	√		√	√		++	++	++	++
SURF	(√)	√		√	√		++	++	++	+++
Harris-Affine	√	(√)		√	√	√	+++	+++	++	++
Hessian-Affine	(√)	√		√	√	√	+++	+++	+++	++
Salient Regions	(√)	√		√	√	(√)	+	+	++	+
Edge-based	√			√	√	√	+++	+++	+	+
MSER			√	√	√	√	+++	+++	++	+++
Intensity-based			√	√	√	√	++	++	++	++
Superpixels			√	√	(√)	(√)	+	+	+	+

Tuytelaars T, Mikolajczyk K. Local invariant feature detectors: a survey. Foundations and Trends® in Computer Graphics and Vision 2008:3:177-280.

An Example: “Bag of Words” Scene Recognition

- Edward Hopper Images
  - Different sizes, IDs, cropping
  - Record features for each image
  - When presented with an image, choose stored record with most matches

An Example: Object Detection w/SIFT

- Registration: Given similarity metric S, geometric transformation T with parameters μ and image I_1 and I_2 find μ^* s.t.

$$S(I_1,T(I_2,\mu^*)) = max_\mu S(I_1,T(I_2,\mu))$$

- Object detection: solve for both identity and registration of objects in an image

- Hough Transform (D. Lowe)
  - Grid registration space and image identities
  - Vote directly from feature matches

- RANSAC
  - Use pairs of matches and compute location and orientation

D. Lowe Feature Matching

- Uses a Hough transform
  - parameters are position, orientation and scale for each training view
  - features are matched to closest Euclidean distance neighbor in database; each database feature indexed to object and view as well as location, orientation and scale
  - features are linked to adjacent model views; these links are also followed and accumulated
  - implemented using a hash table

Verification and Training

- Views are matched under affine transformations:
  - u’ = s R u + d → leads to a linear system Ax = b
  - (geometric) match error e = sqrt(2 || A x* - b ||/(r-4)) where r is the # of matched features
  - in learning stage, use e to decide if a view should be clustered or create a new cluster; threshold T = 0.05 * max(r,c) where r,c is size of training image

- Training simply requires many images of objects, not necessarily organized in any way; three cases:
  - training image doesn’t match an existing object model; new object model is formed with this image
  - training image matches an existing model view, but e > T;
    - new model view created and linked to three closes model views; overlapping features are linked.
  - training image matches an existing model view and e < T;
    - aggregate any new features into the existing model view

Final Probability Model

- There can still be many false positives and negatives
- Compute P(m | f) where f are the k matched features and m is a model view
- probability of false match for a single feature is
    - p = d l r s
    - d = fraction of database features in this model view
    - l = 0.2^2 = 0.04 (location ranges of 20% of model size)
    - r = 30/360 = 0.085
    - s = 0.5
    - P(f | : m) = binomial using p, n (# of features) and k (# of matches)
- P (m | f) = P(f|m) P(m) / (P(f|m) P(m) + P(f | : m) P(: m))
- Assume P(: m)=1 and P(f | m) = 1
- Thus P(m | f) = P(m) / (P(m) + P(f | : m))
- Assume P(m) is roughly constant and = 0.01
- Accept a model if P(m|f) > 0.95
- Requires 3-10 features depending on object and level of clutter

PDF of Matching

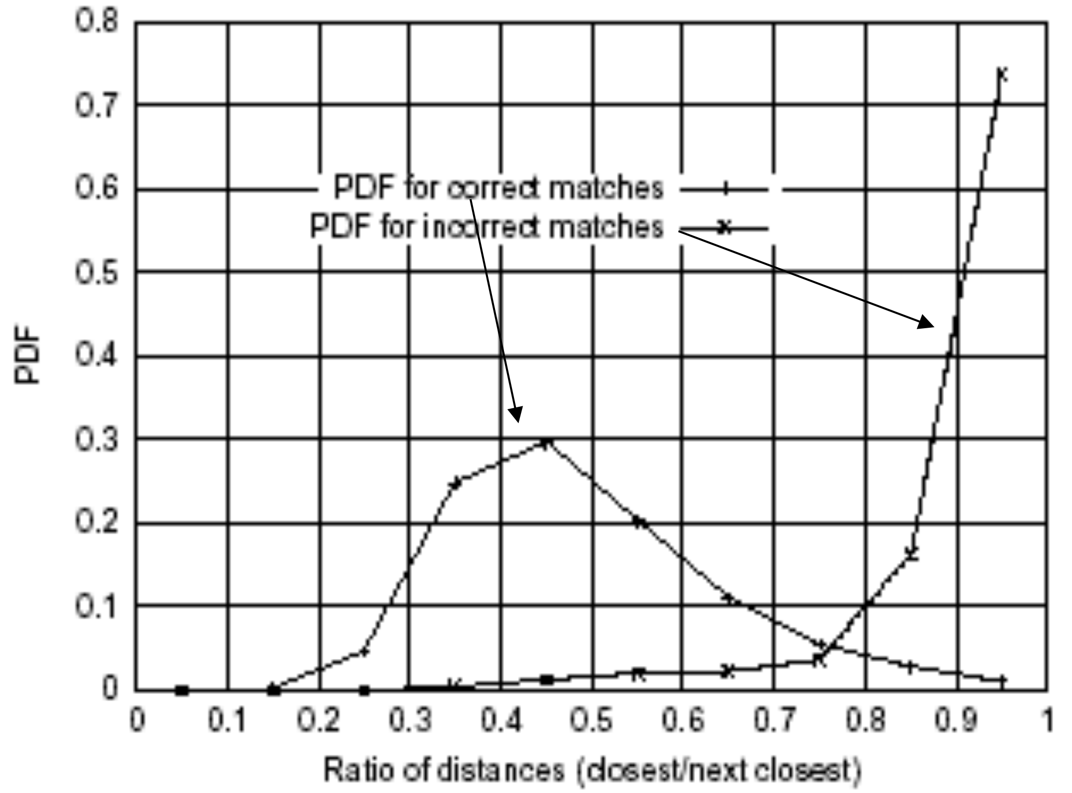


Figure 11: The probability that a match is correct can be determined by taking the ratio of distance from the closest neighbor to the distance of the second closest. Using a database of 40,000 key-points, the solid line shows the PDF of this ratio for correct matches, while the dotted line is for matches that were incorrect.

Results

•

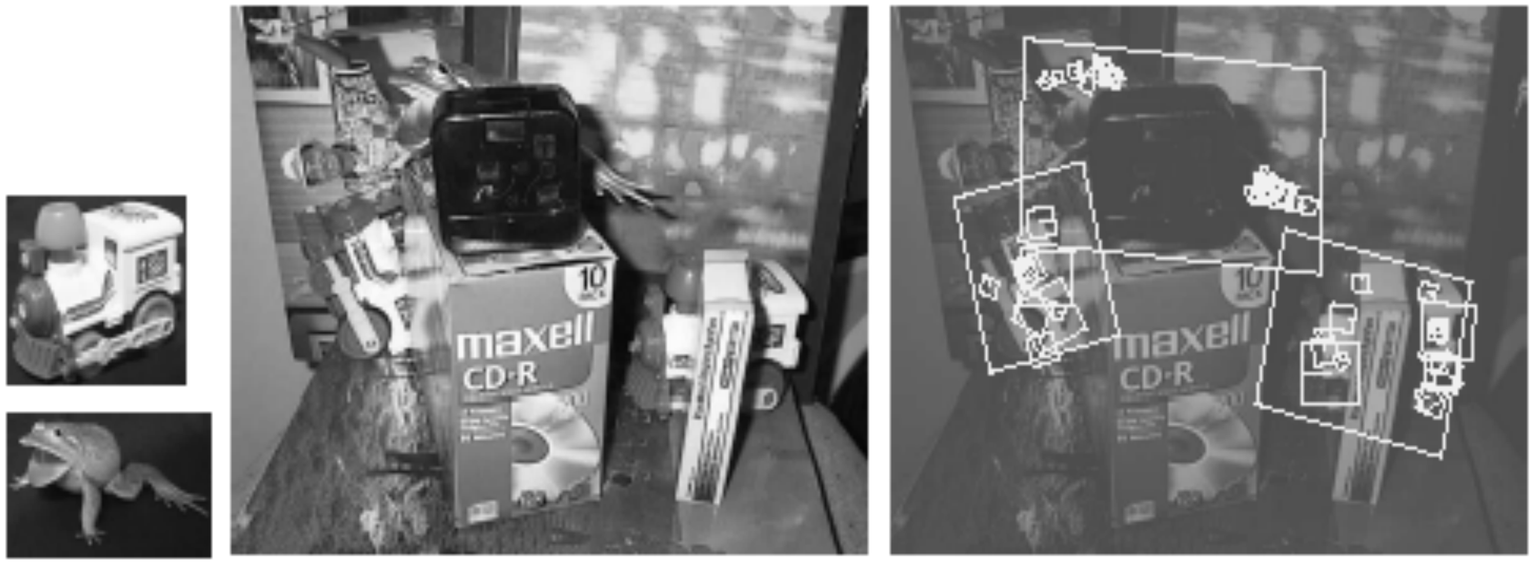


Figure 12: The training images for two objects are shown on the left. These can be recognized in a cluttered image with extensive occlusion, shown in the middle. The results of recognition are shown on the right overlaid on a reduced contrast version of the image. A parallelogram is drawn around each recognized object showing the boundaries of the original training image under the affi ne transformation solved for during recognition. Smaller squares indicate the keypoints that were used for recognition.

Results

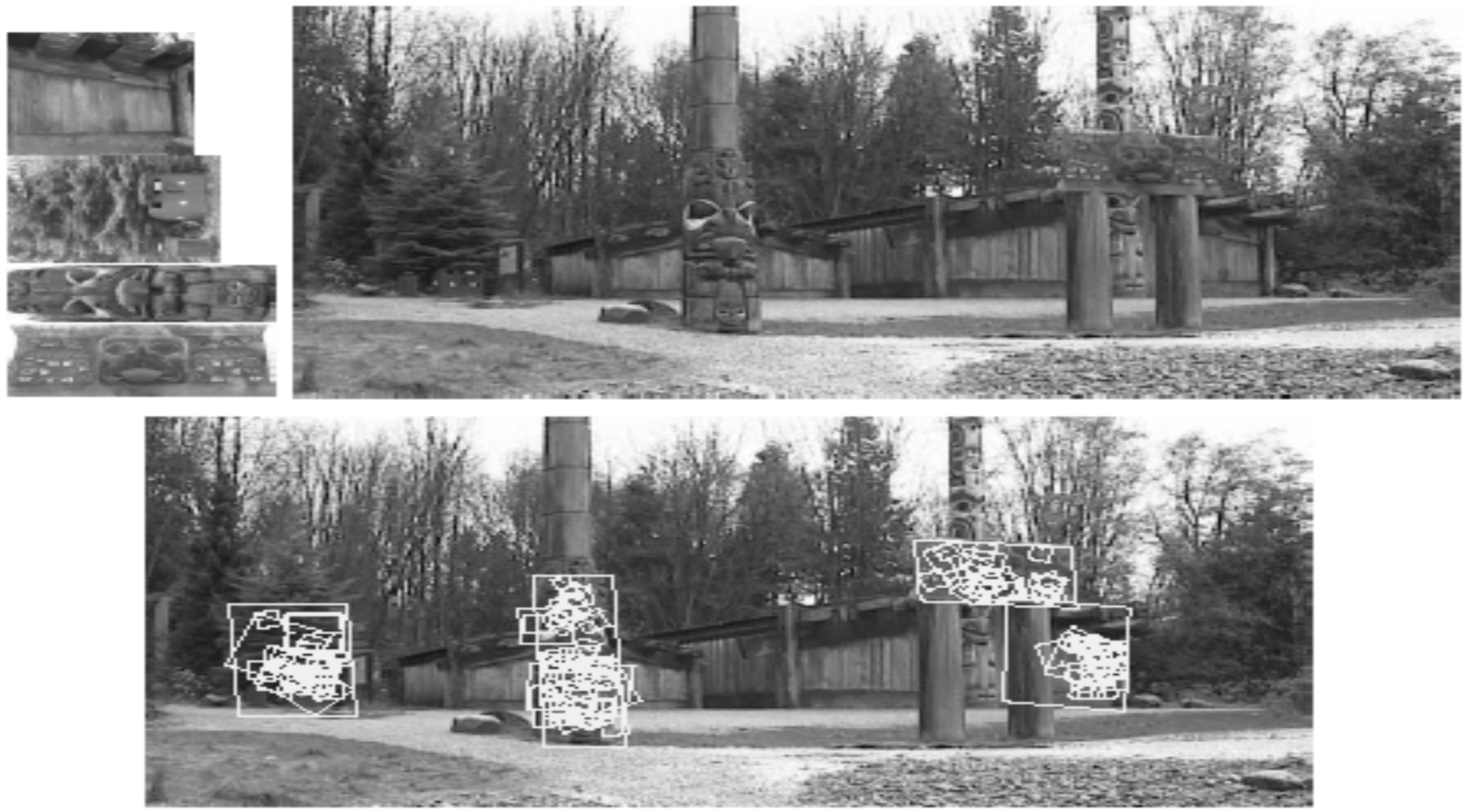


Figure 13: This example shows location recognition within a complex scene. The training images for locations are shown at the upper left and the 640x315 pixel test image taken from a different viewpoint is on the upper right. The recognized regions are shown on the lower image, with keypoints shown as squares and an outer parallelogram showing the boundaries of the training images under the affi ne transform used for recognition.

Image Feature Detection Summary

- Canny detector is probably the most widely used algorithm for performing edge detection
- Feature matching metrics
- Corner detectors can be formulated also in terms of image gradient structures
- SIFT combines detection and matching in a robust fashion
- Hough transform and RANSAC are methods of incorporating geometric constraints into the matching process