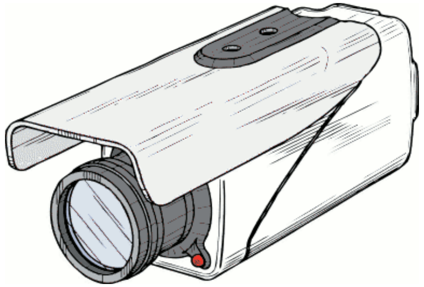
Univerza *v Ljubljani*

Machine perception
Derivatives and edge detection

Matej Kristan

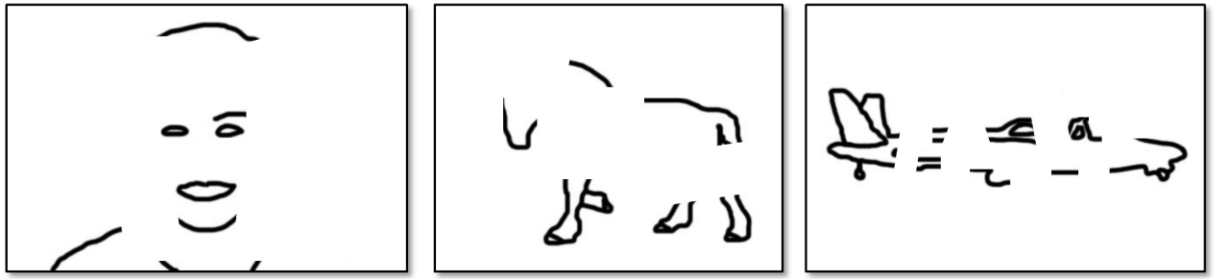


Laboratorij za Umetne Vizualne Spoznavne Sisteme,
Fakulteta za računalništvo in informatiko,
Univerza v Ljubljani

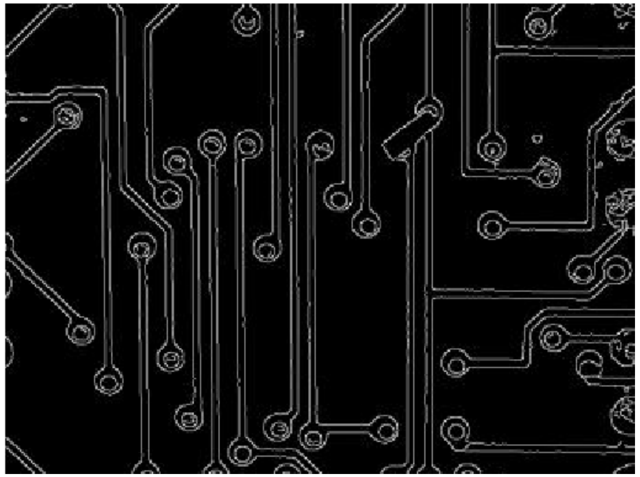
Edge detection

- Goal: map image from 2D grayscale intensity pixel array into a set of binary curves and lines.
- Why?

abstraction



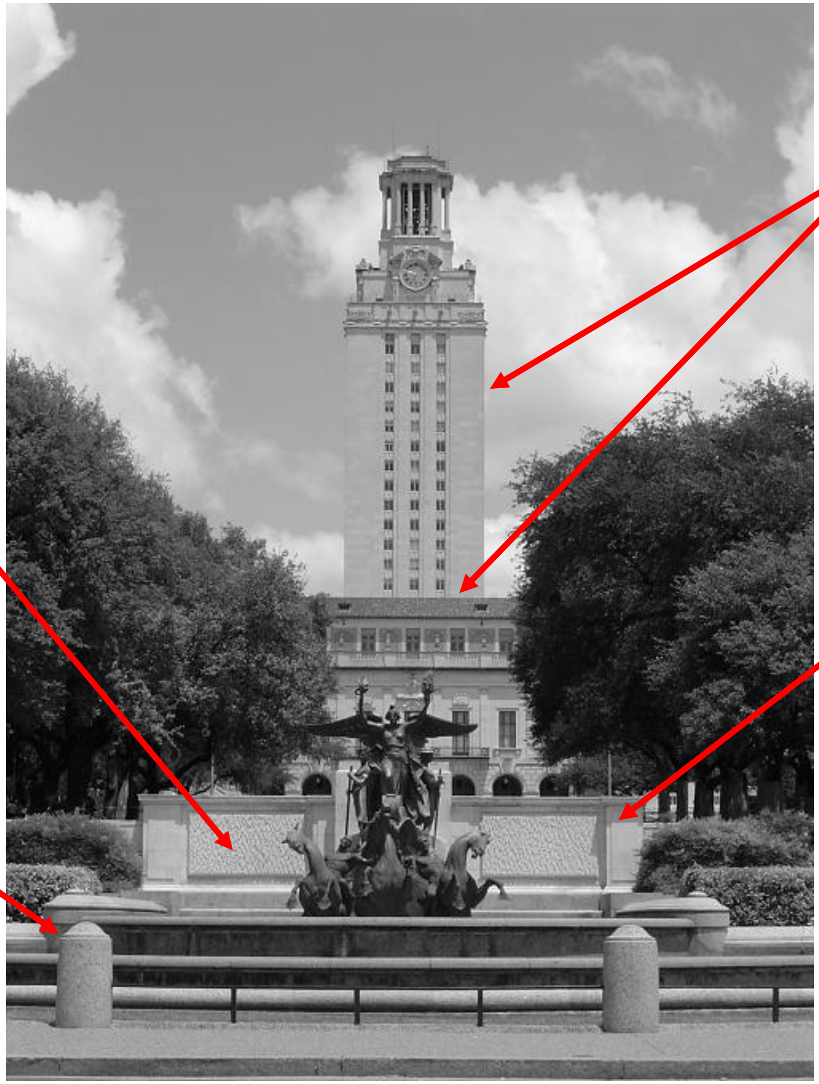
Robust, compact representation



Measurement

What constitutes an edge?

Anything that *appears* as an edge…



Discontinuity of depth: object borders

Local texture:

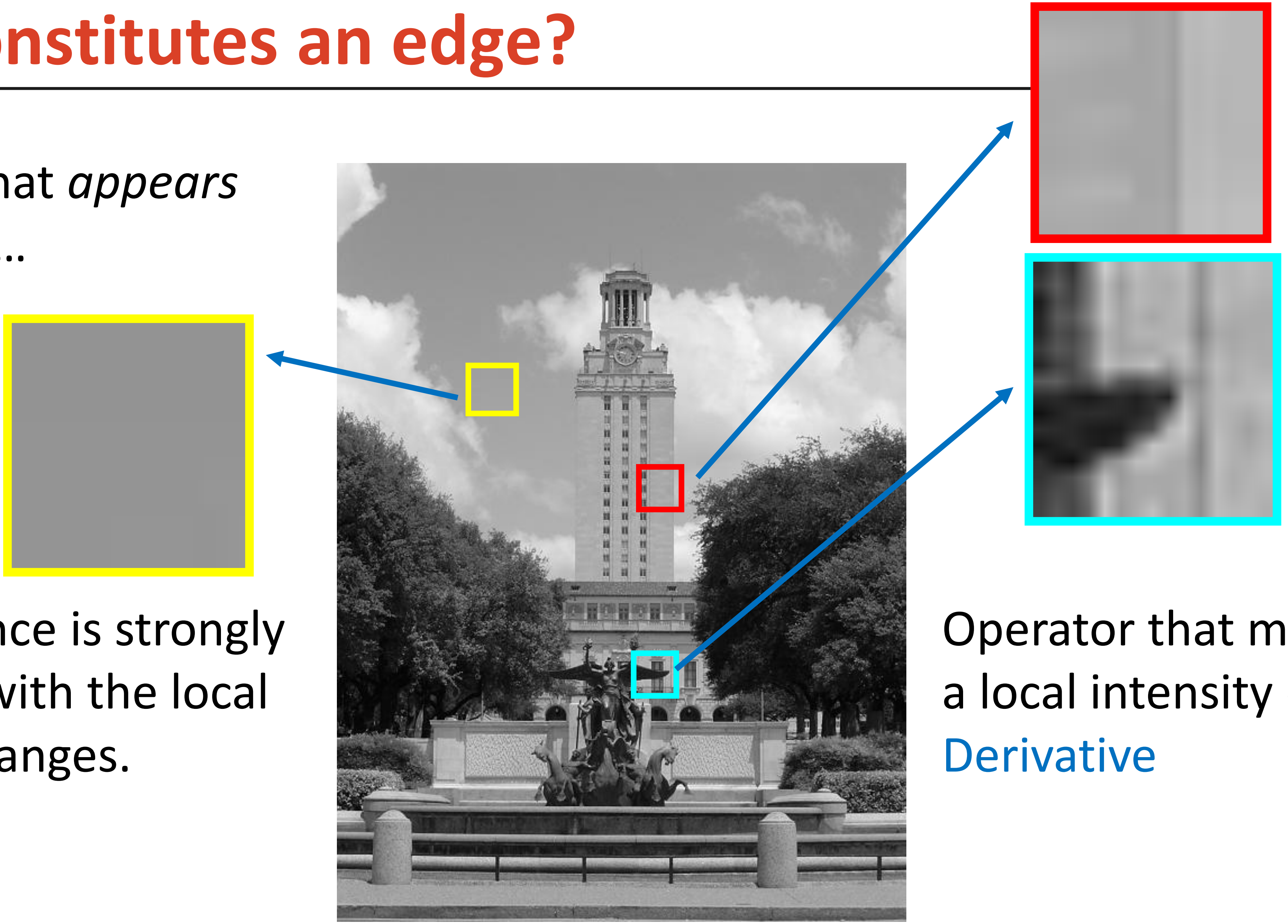
Shadows

Changes in 3D normal orientation caused by shape changes

Slide credit: Kristen Grauman

What constitutes an edge?

Anything that *appears* as an edge…



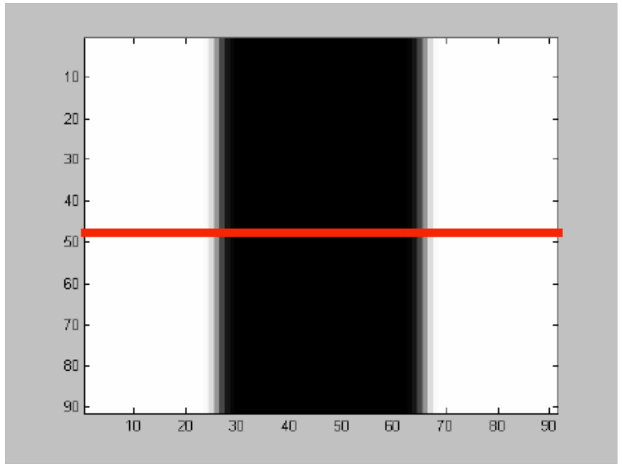
Edge presence is strongly correlated with the local intensity changes.

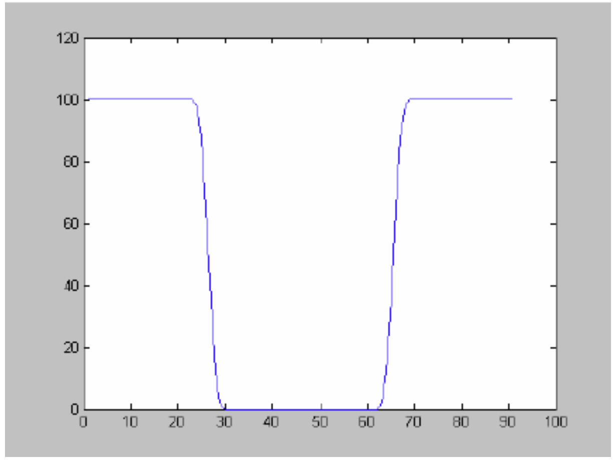
Operator that measures a local intensity change: Derivative

Machine perception

IMAGE DERIVATIVES

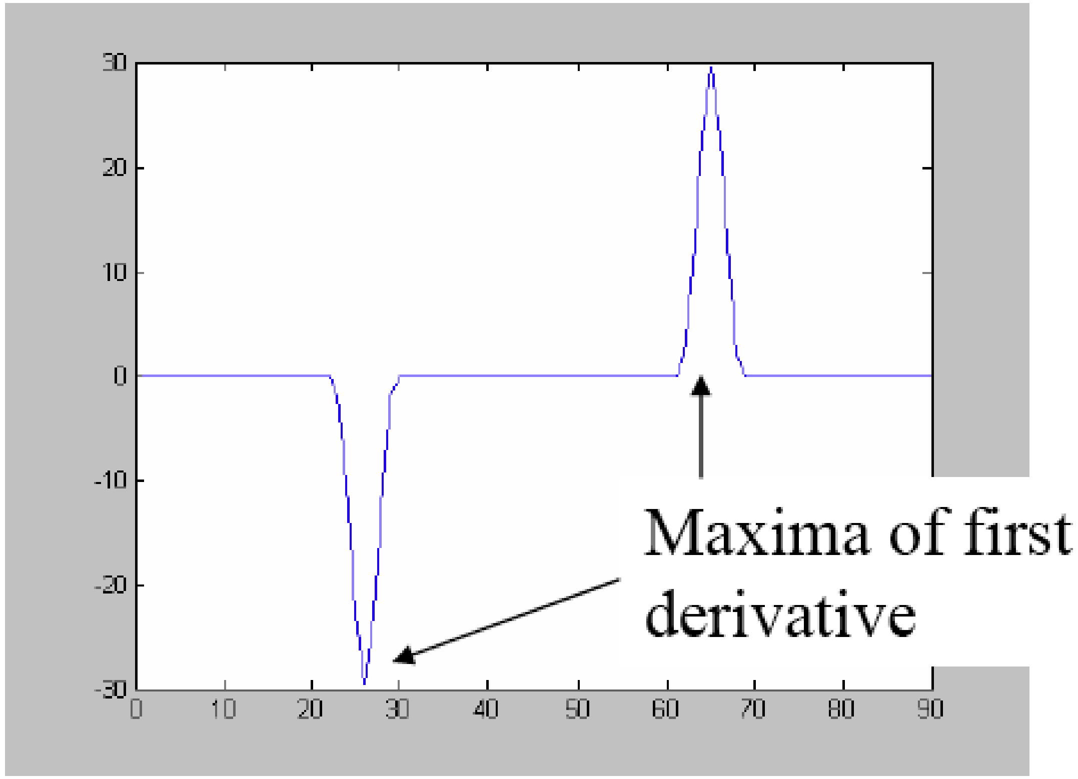
1D derivative: Intuition





First order derivative

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



Maxima of first derivative

Derivatives and convolution

- A partial derivative of a continuous 2D function $f(x,y)$:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$$

- For a discrete case, approximate by using finite differences:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1, y) - f(x,y)}{1}$$

- Question: If implemented by convolution, what would the convolution kernel for derivative look like? (Next slide)

Slide credit: Kristen Grauman

Partial derivatives: Implementation

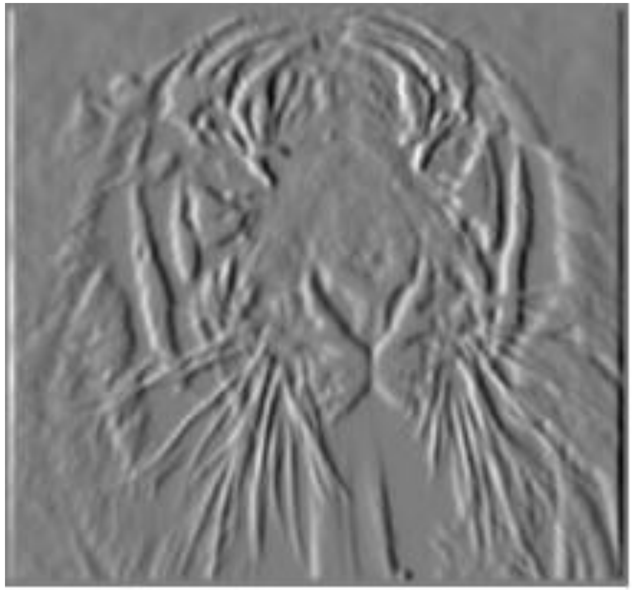
Horizontal derivative

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{\underline{f(x+1,y)} - \underline{f(x,y)}}{1}$$

*

-1	1

=



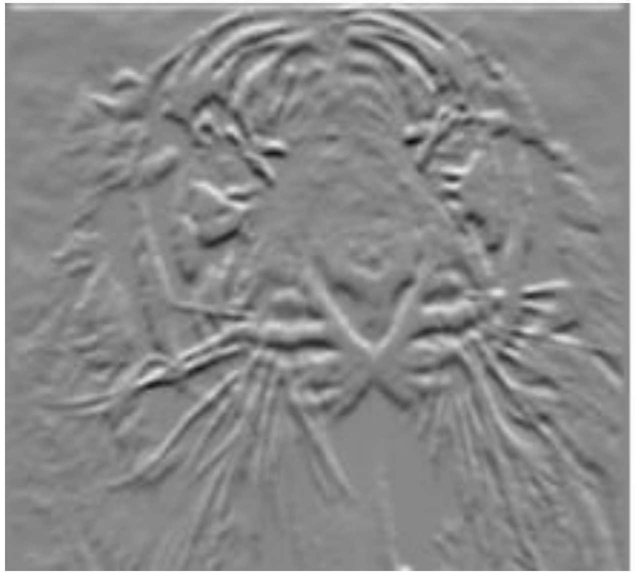
Vertical derivative

$$\frac{\partial f(x,y)}{\partial y} \approx \frac{f(x,y+1) - f(x,y)}{1}$$

*

-1
1

=

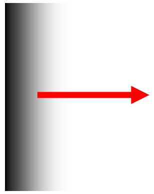


Partial derivatives: Image gradient

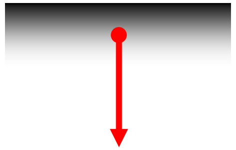
- Image gradient: $$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

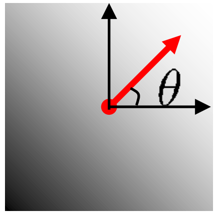
- Gradient points in direction of greatest intensity change:

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



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





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

- Gradient direction (orientation of edge normal):

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

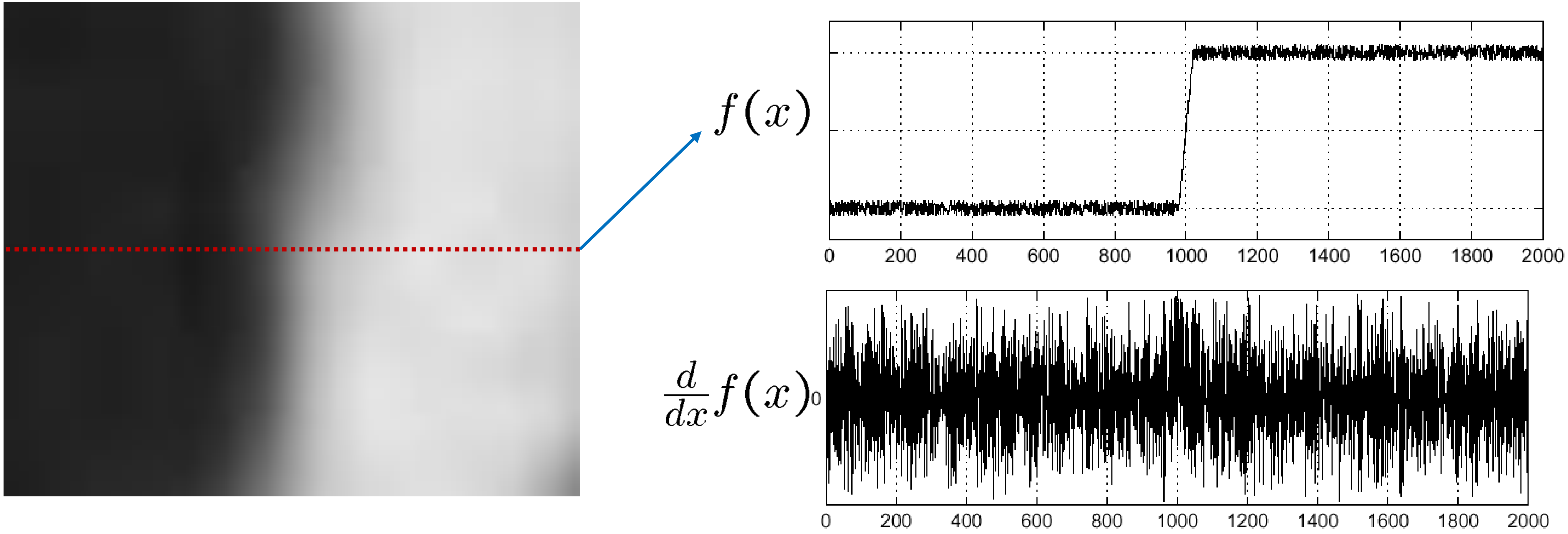
- Gradient strength is defined by its magnitude:

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

Gradient magnitude

Discrete world is noisy...

- Take a single line in the image:
    - Plot intensities w.r.t. pixels:



So where did the edge go!? Noise gets amplified by derivation...

Slide credit: Steve Seitz

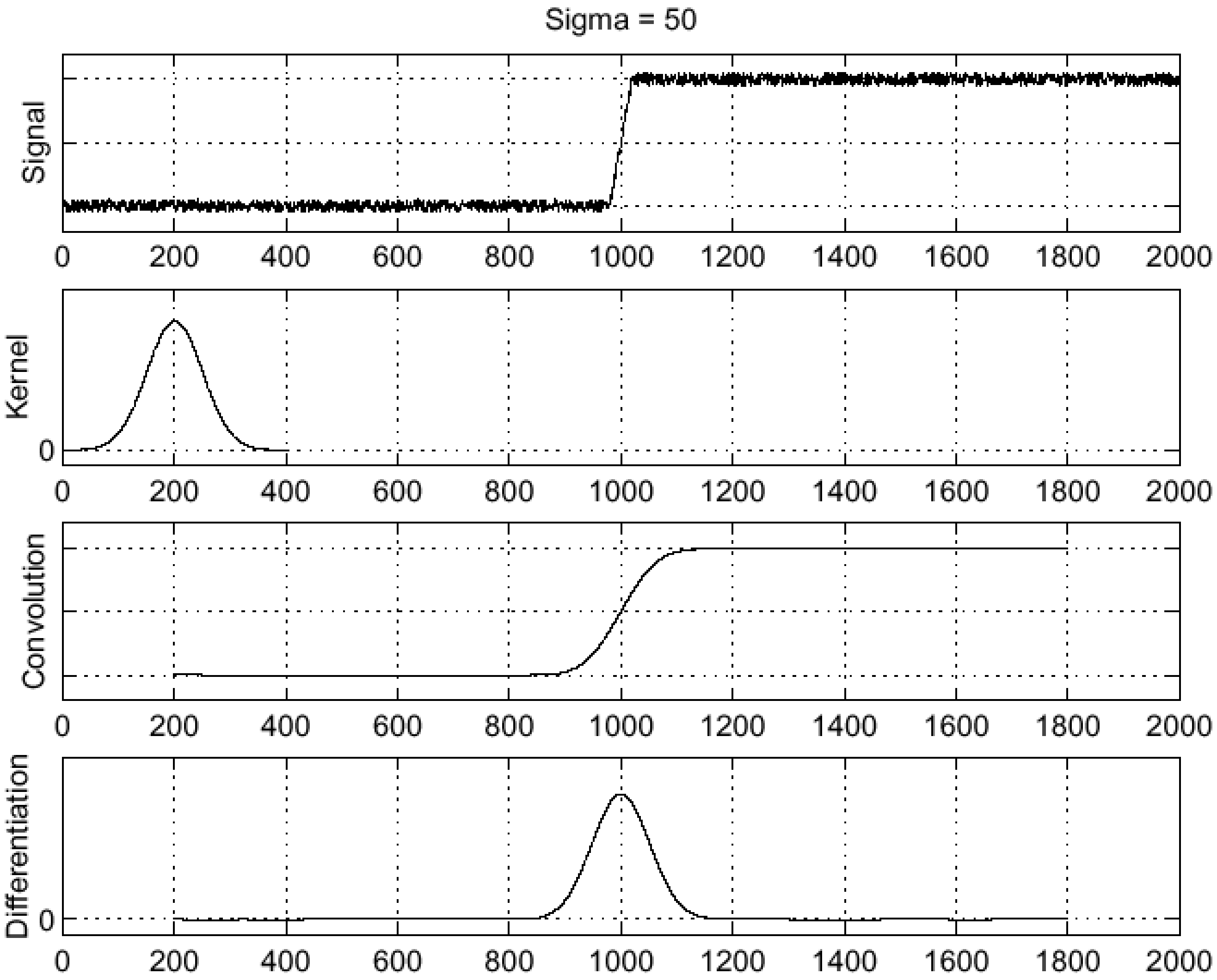
Solution: Smooth the image first

f

h

$h \star f$

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



Where’s the edge?

Find maxima in map $\frac{\partial}{\partial x}(h \star f)$

Slide credit: Steve Seitz

Remember convolution properties

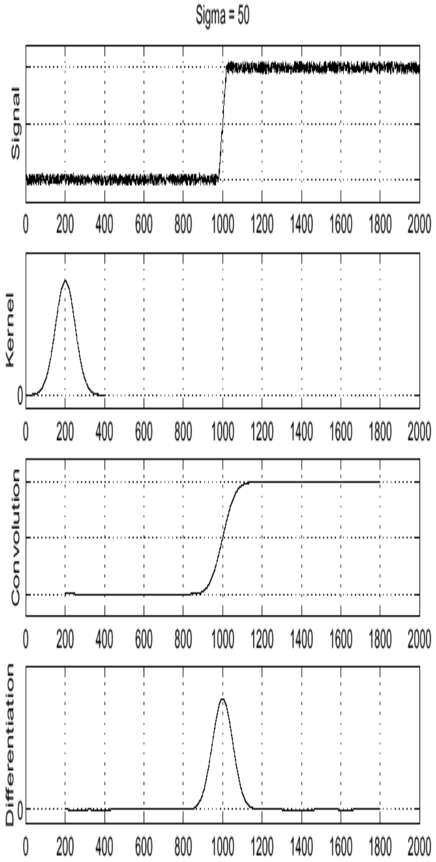
- Derivatives: $\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x} h) \star f$

f

h

$h \star f$

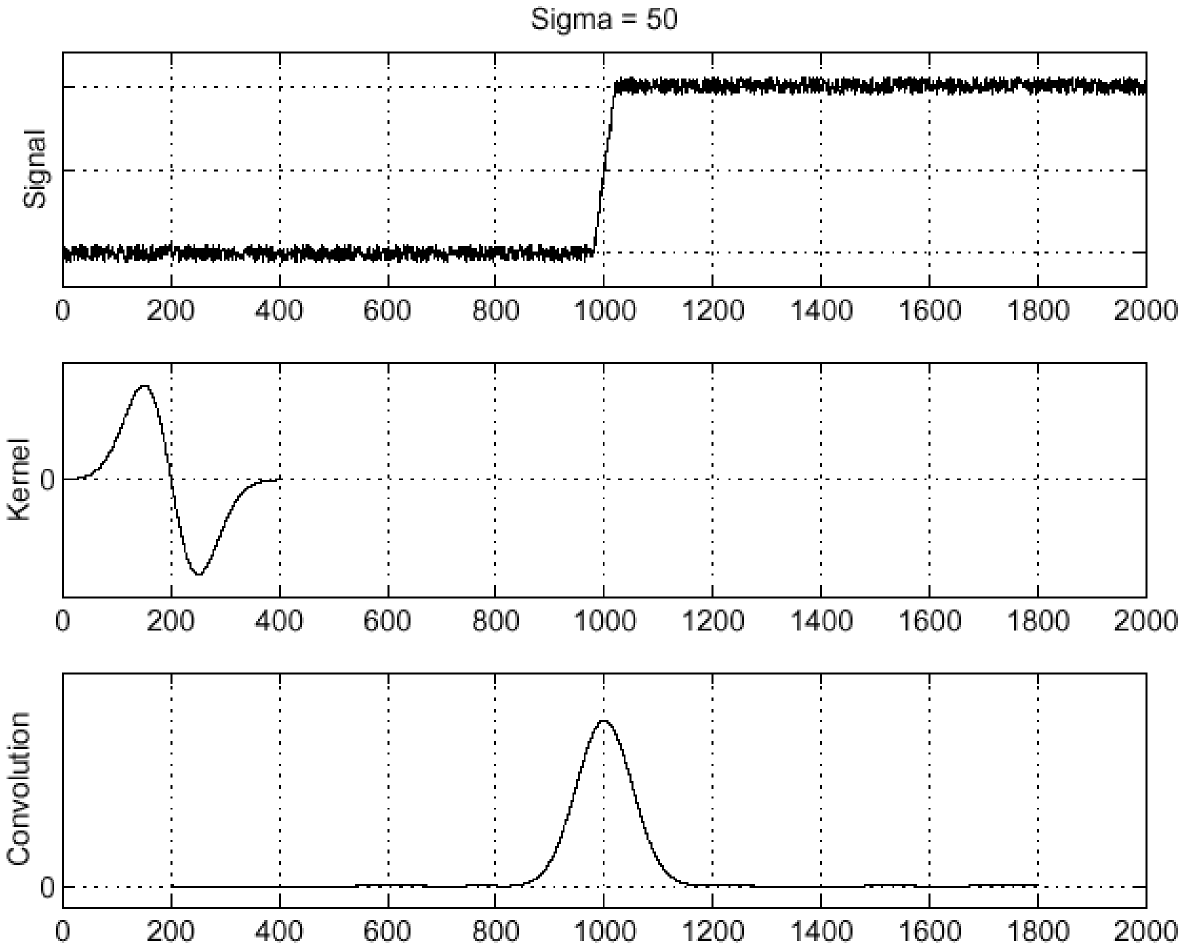
$\frac{\partial}{\partial x}(h \star f)$



f

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

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



2D partial derivatives – naiive approach

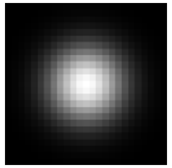
1. Smooth the image by a 2D Gaussian filter
2. Take derivative w.r.t. x

1 Blurring

Image

*

Gaussian kernel



=

Blurred image

2 Differentiating w.r.t. x

Blurred image

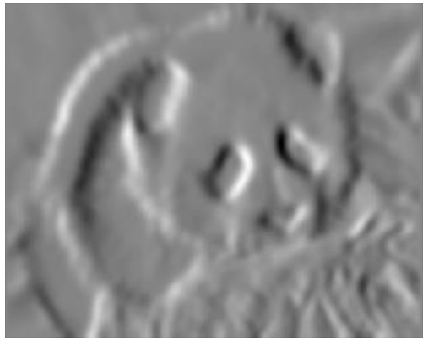
*

Derivative kernel

1	-1

=

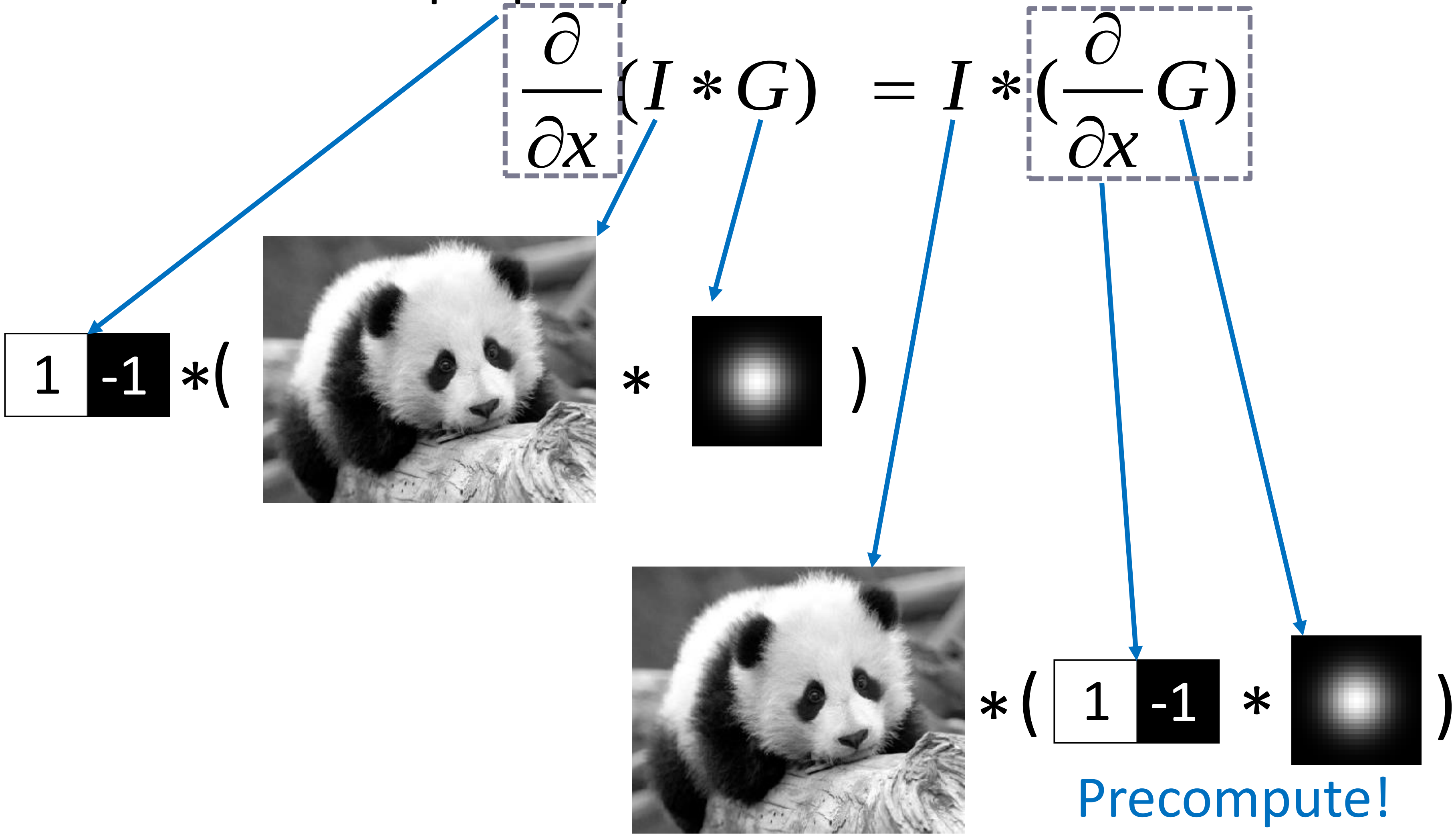
Derivative image



2D partial derivatives – smarter approach

- Recall the convolution property:

$$\frac{\partial}{\partial x}(I * G) = I * \left(\frac{\partial}{\partial x} G\right)$$

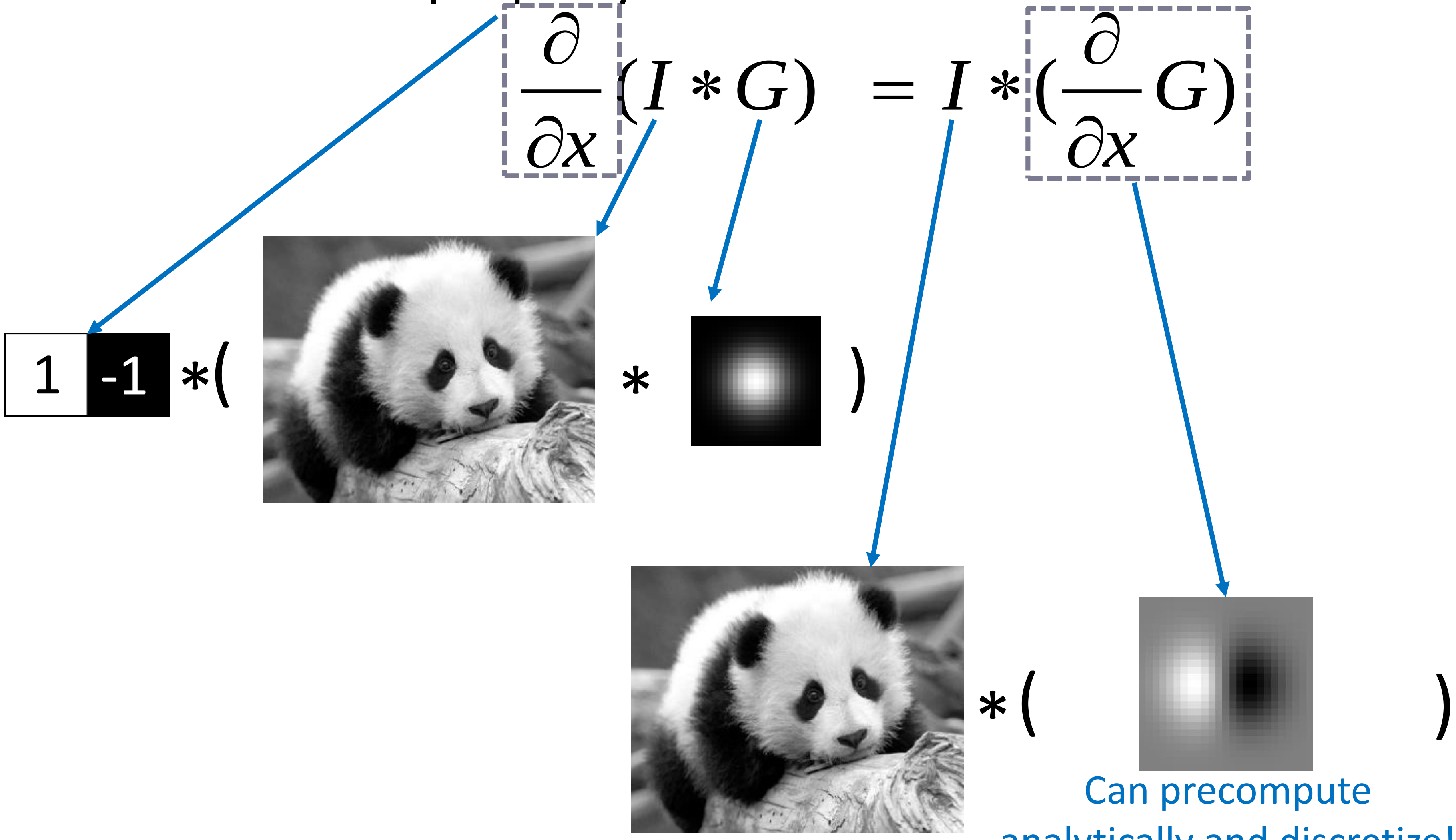


Precompute!

Smarter way

- Recall the convolution property:

$$\frac{\partial}{\partial x}(I * G) = I * \left(\frac{\partial}{\partial x} G\right)$$



Can precompute analytically and discretize!

2D partial derivatives – smarter approach

- Recall the convolution property:

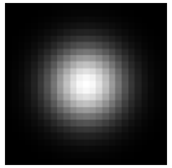
$$\frac{\partial}{\partial x}(I * G) = I * (\frac{\partial}{\partial x} G)$$

Naiive:

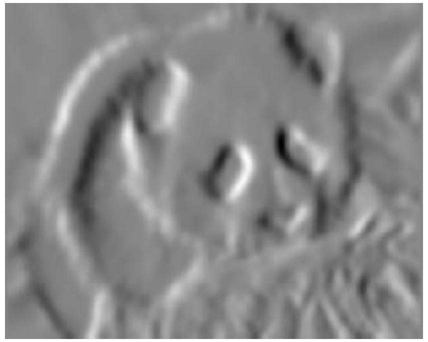
1	-1

*(

*

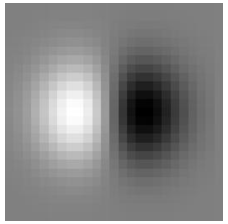


) =

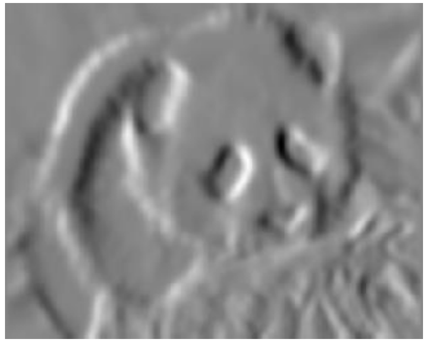


Smarter:

*

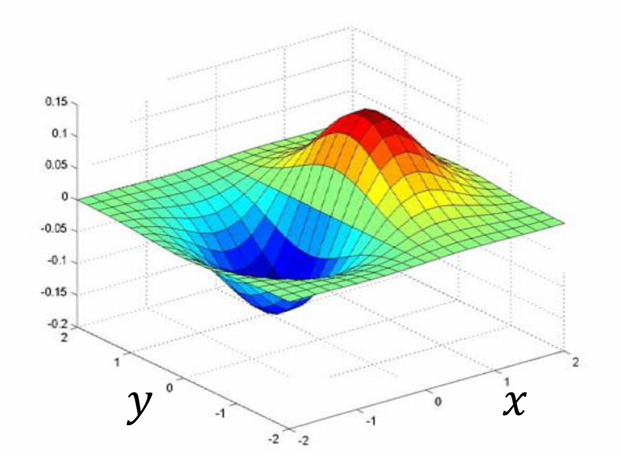


=

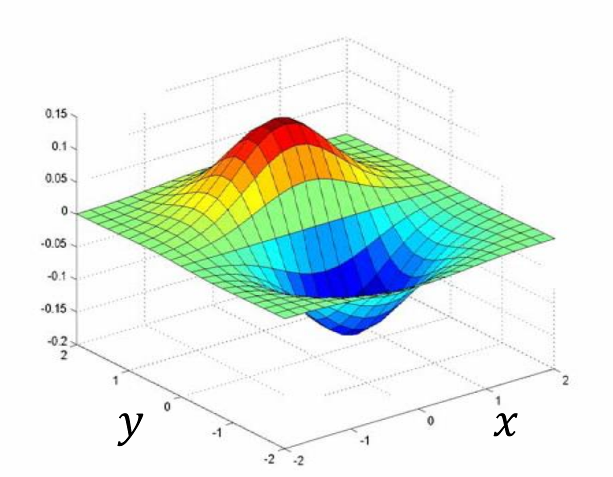


Gaussian partial derivatives

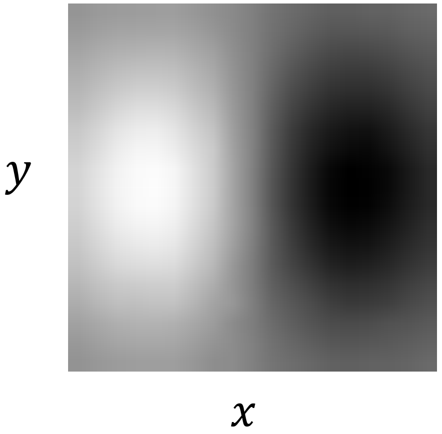
- Convolution kernels for taking partial derivatives w.r.t. x and y:

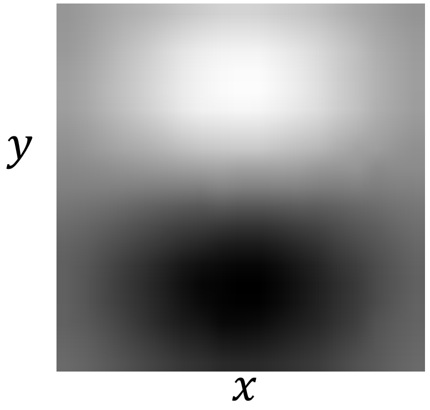


w.r.t. x



w.r.t. y





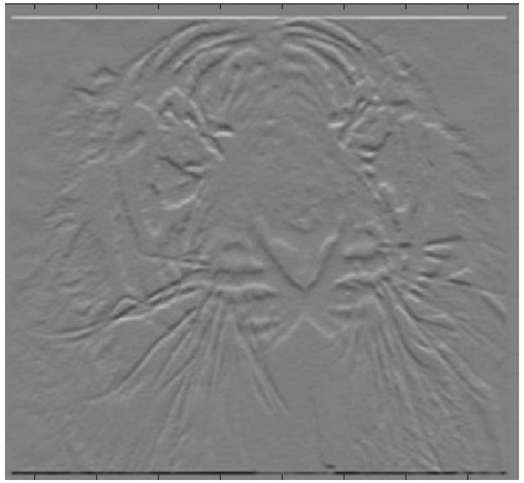
Some other popular kernels

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

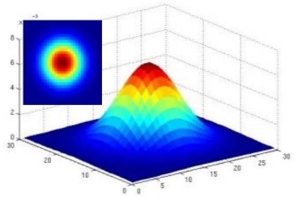
```
>> My = fspecial(‘sobel’);
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```

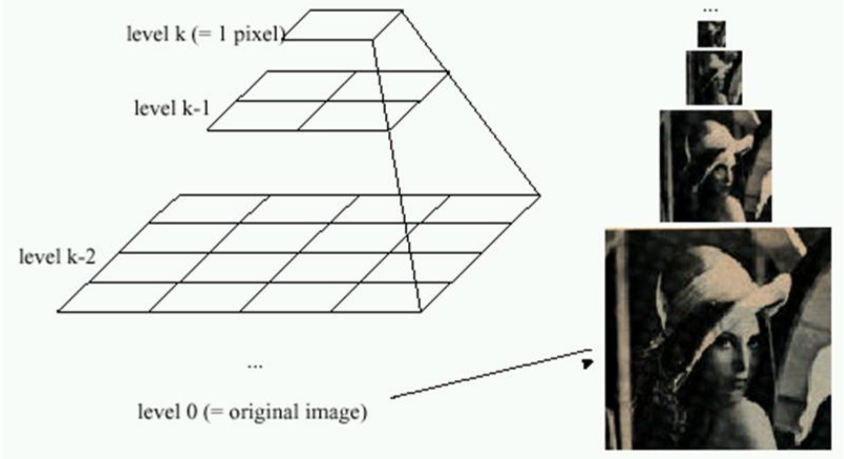


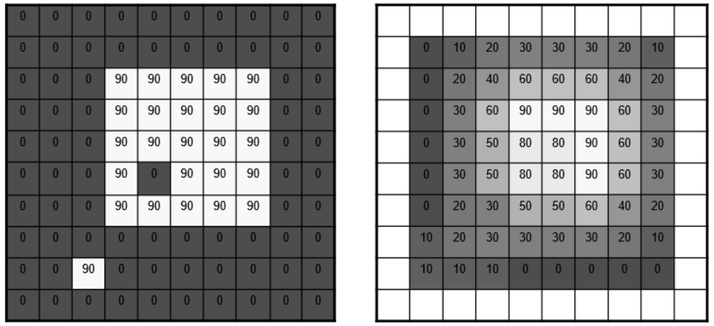
Slide credit: Kristen Grauman

Previously at MP…

- Linear filters: convolution, correlation
- Nonlinear filters: Median filter





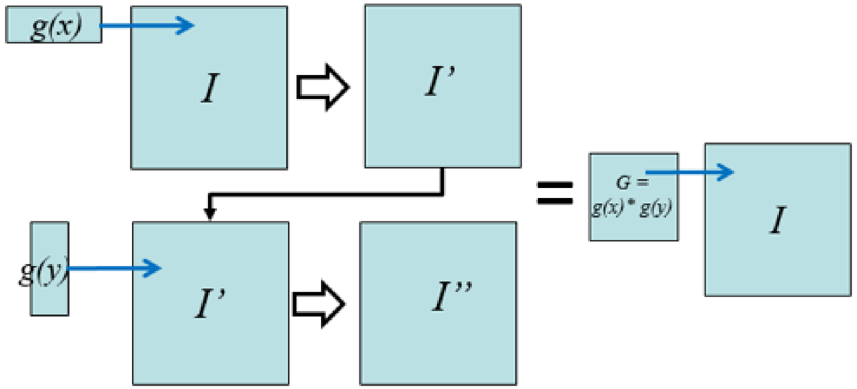


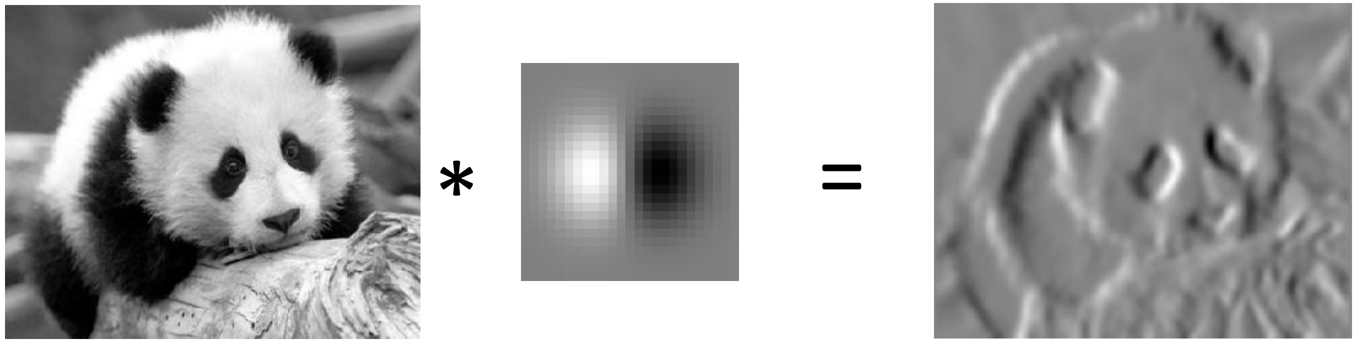
$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

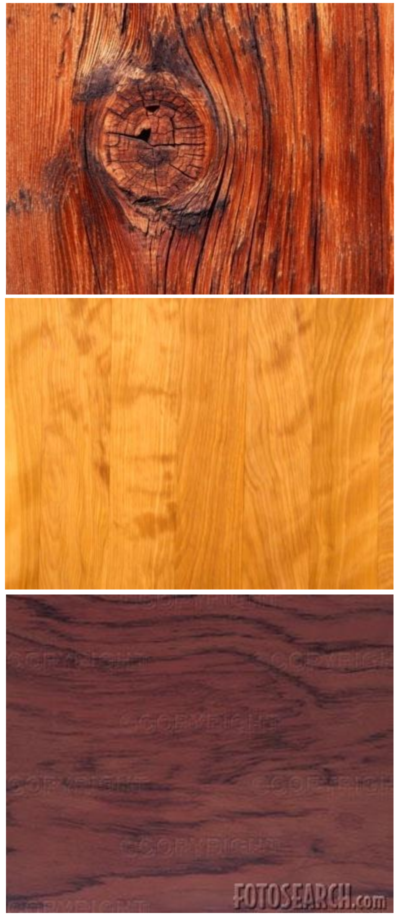




Edges exist at different scales

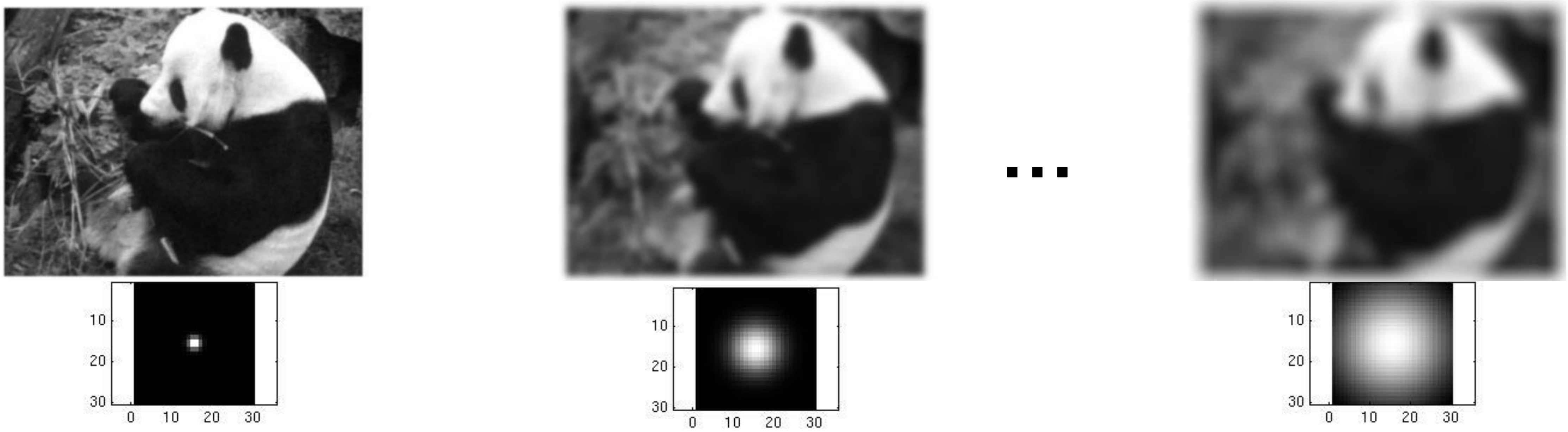
Depends on what we’re looking for...

Thin edges or thick edges (leaves, branches, trunks,…)



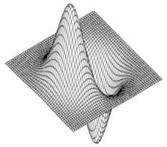
Tuning the filter to the right scale

Parameter σ is the “scale”/“width” of a Gaussian kernel that determines the extent of smoothing, i.e., determines which *edges will be removed*.

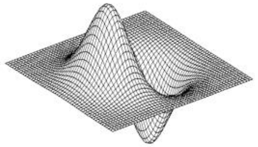


Tuning the filter to the right scale

How does σ affect the derivative?



σ = 1 pixel



σ = 3 pixels

The enhanced/detected structures depend on the Gaussian kernel size.

Large kernels: detect edges on a larger scale.
Small kernels: detect edges on a smaller scale.

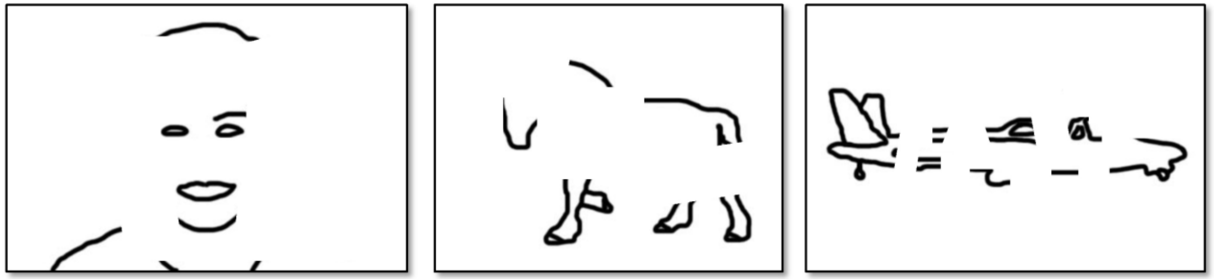
Machine perception

FROM DERIVATIVES TO EDGE DETECTION

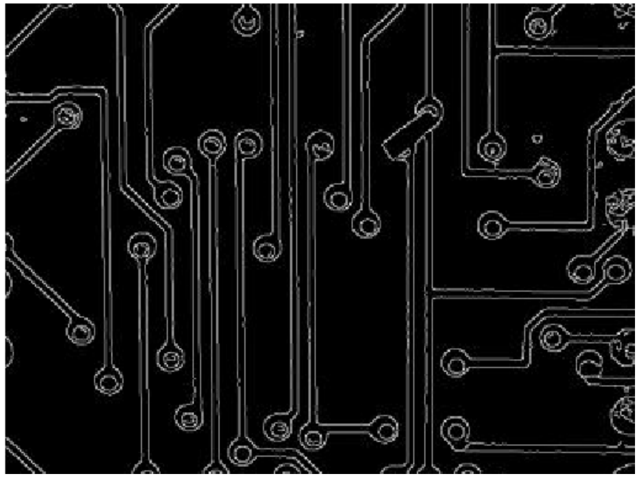
Recall: The task of edge detection

- Goal: map image from 2D grayscale intensity pixel array into a set of binary curves and lines.

abstraction



Robust, compact representation



Measurement

Derivative enhances the edges, but these are not *binary curves*.

The task of edge detection

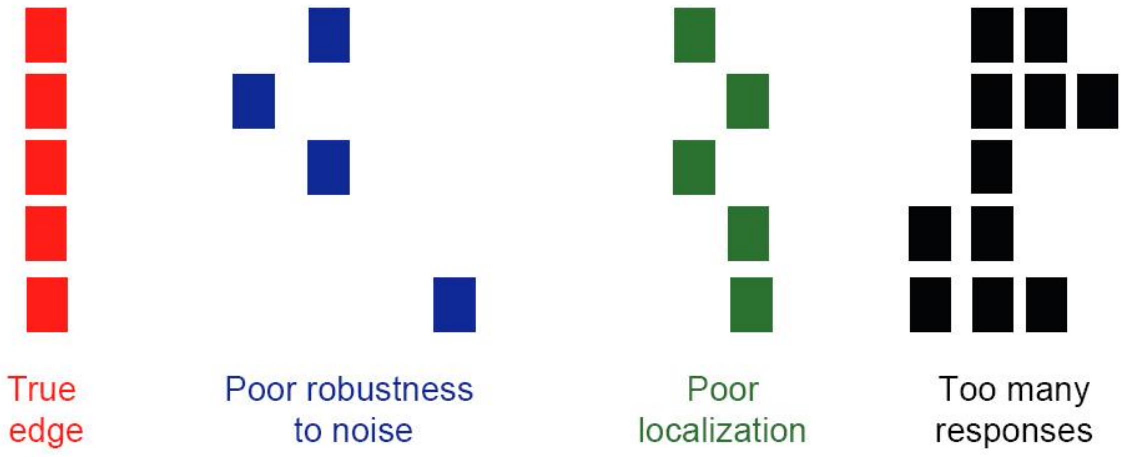
$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

- Basic approach:

  find strong gradients + post process

Designing an edge detector...

- Criteria of "optimal" edge detector:
  1. **Good detection:** optimal detector minimizes probability of false positives (edges caused by noise), and false negatives (missing true edges)
  2. **Good localization:** detected edges should be close to the location of the true edges.
  3. **Specificity:** detector should return only a single point per true edge; minimize number of local maxima around true edge.



Source: Li Fei-Fei

The Canny edge detector [Canny, IEEETPAMI 1986]

- Most popular edge detector in computer vision.
- Theoretical model of *the edge*:
  A step function + Gaussian noise.
- Canny showed that first derivative of a Gaussian well approximates an operator that optimizes a tradeoff between signal-to-noise ratio and localization on the specified theoretical edge model.

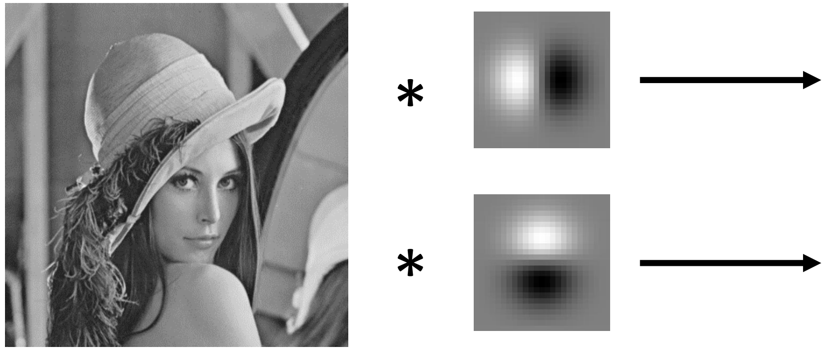
Python:

```
>> cv2.Canny(image, Th_lo, Th_hi,…)
```

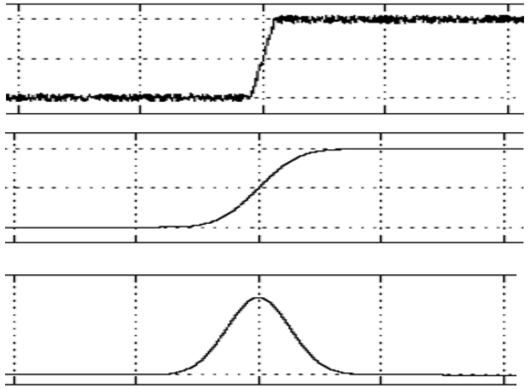
J. Canny, *A Computational Approach To Edge Detection*, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Canny edge detector

1. Filter image by a derivative of a Gaussian (smooth and enhance)



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

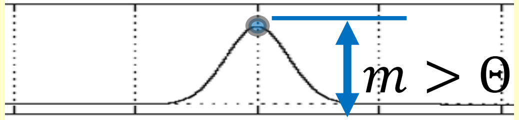


2. Calculate the gradient magnitude and orientation

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

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

3. Thin potential edges to a single pixel thickness



4. Select sequences of connected pixels that are likely an edge

Canny: enhancing the potential edge pixels

Original image (Lena)

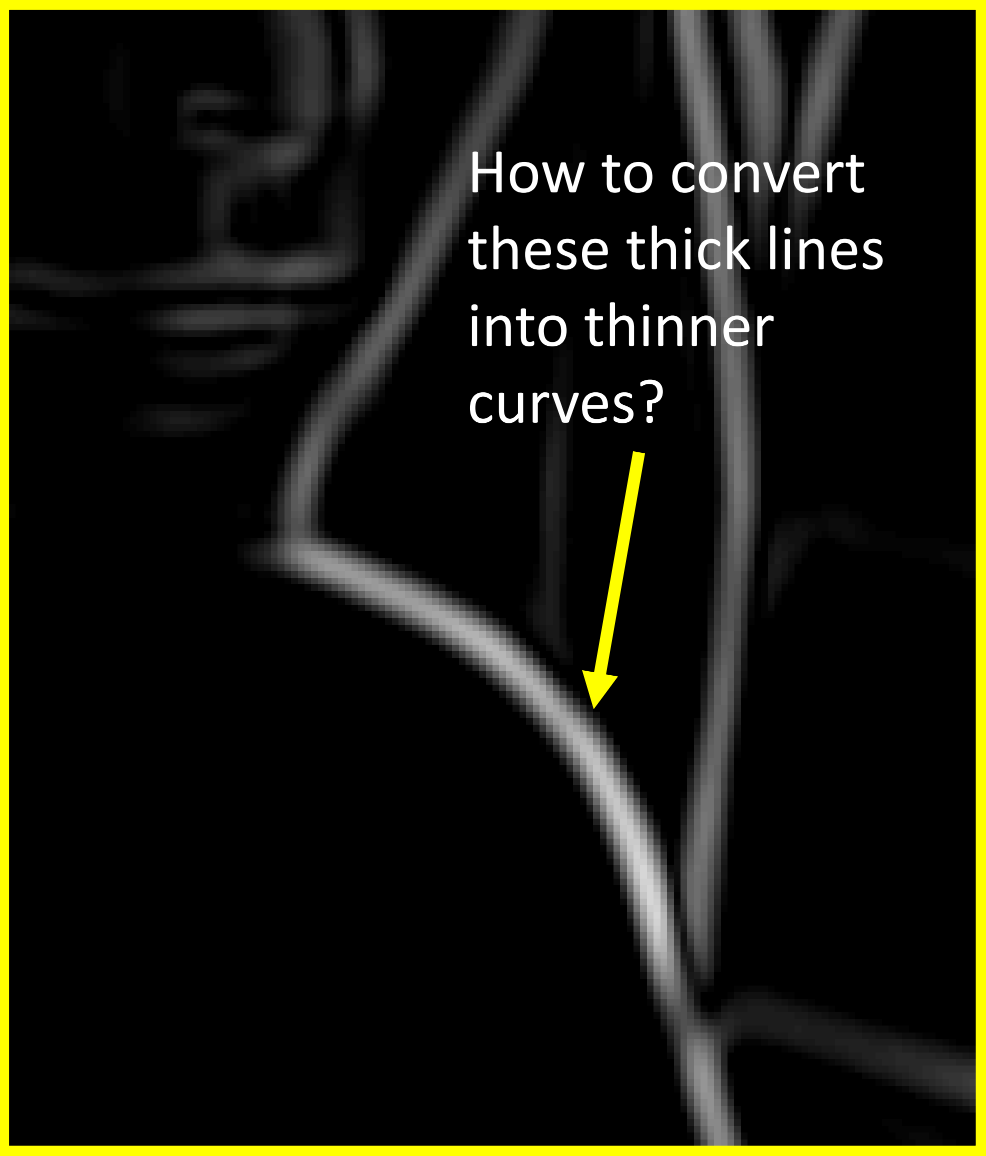
Gradient magnitude

Canny: enhancing the potential edge pixels

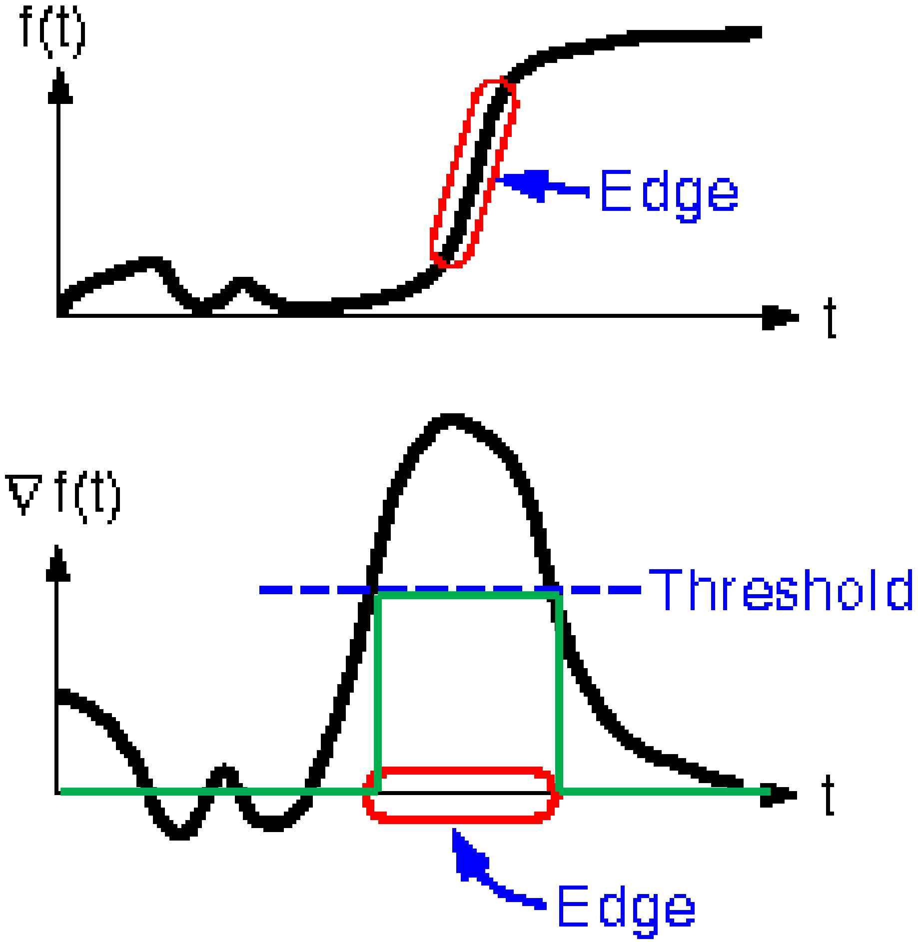
Original image (Lena)

„Thresholding“: Set magnitudes lower than a prescribed threshold to 0.

Canny: thinning the edges

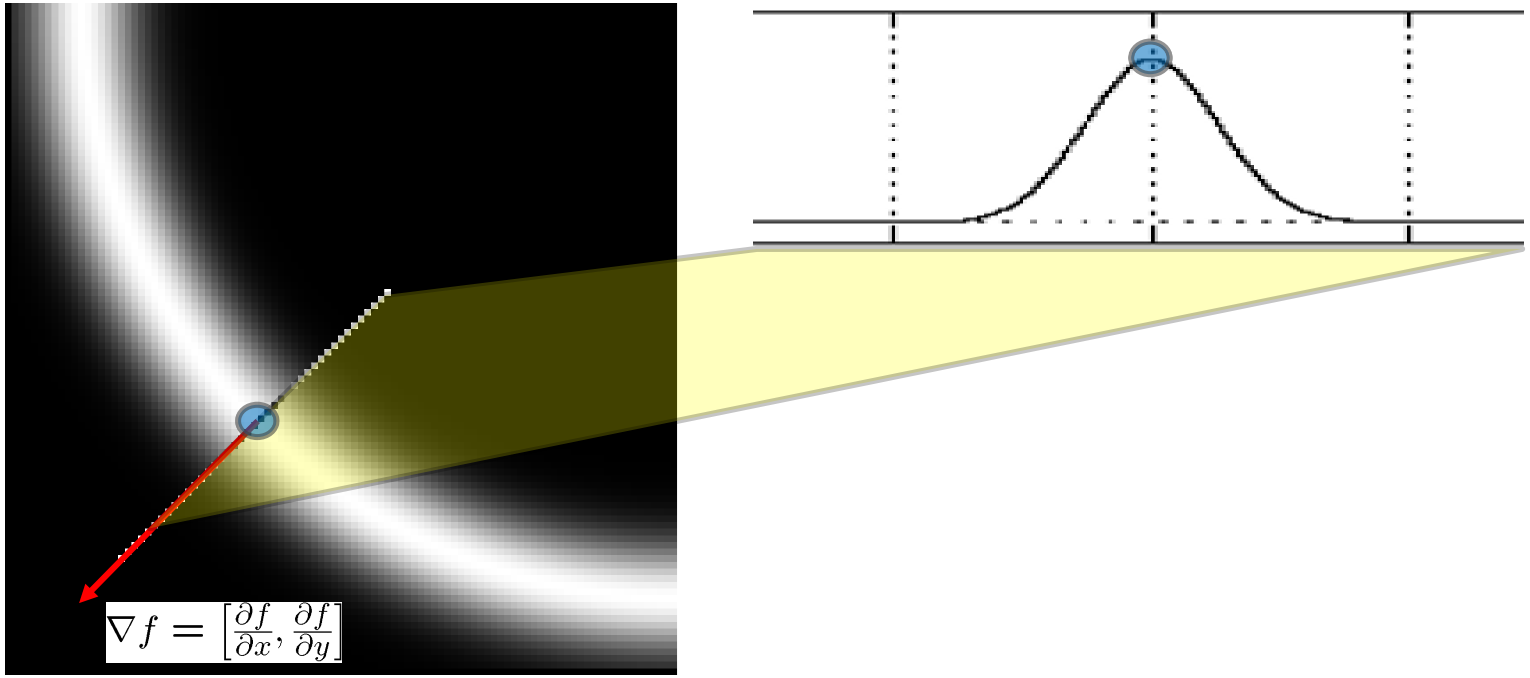


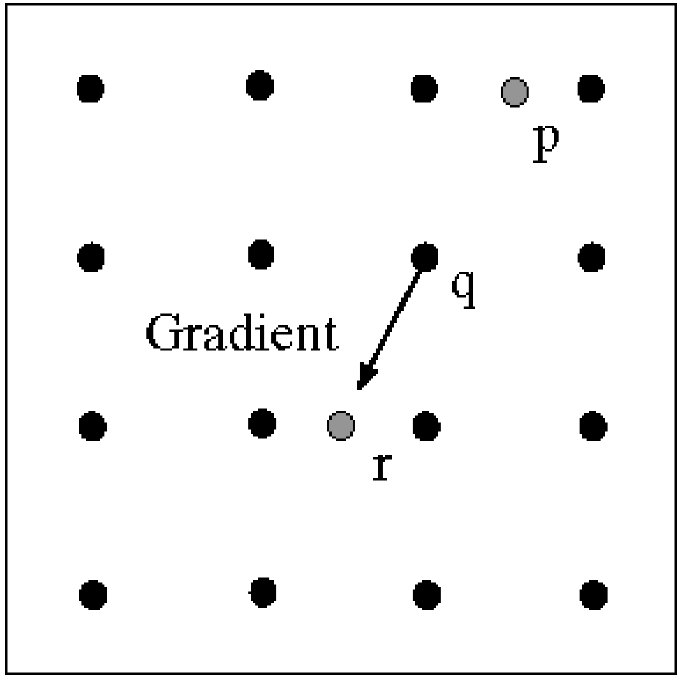
How to convert these thick lines into thinner curves?



Not by thresholding…

Thinning by non-maxima suppression





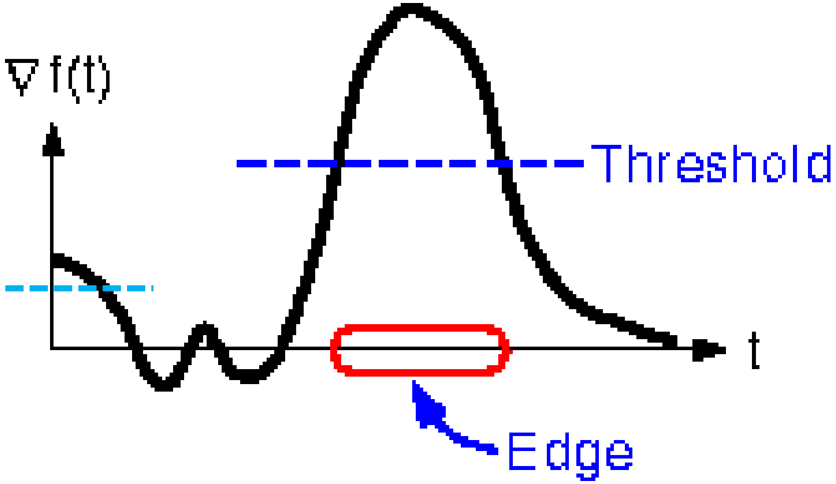
- For each pixel check if it is a local maximum along its gradient direction.
    - Advanced: Actually, for **q**, we should check interpolated pixel values at **p** and **r**.
- Only local maxima should remain.

Canny: thinning the edges

Thinning

(non-maximum suppression)

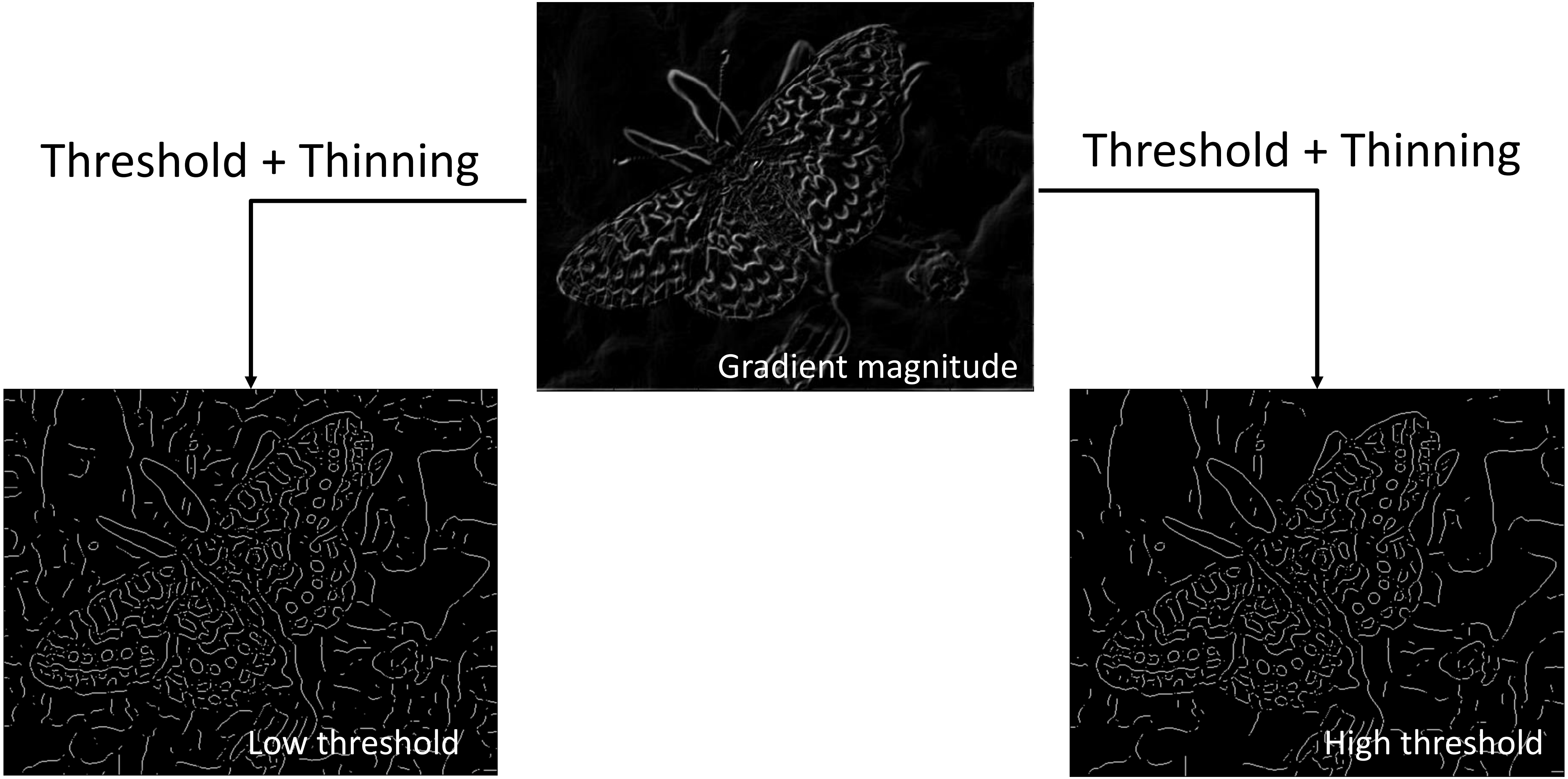
Canny: thinning the edges



Thinning
(non-maximum suppression)

Problem: pixels along this edge did not „survive“ thresholding.

How to select a threshold?

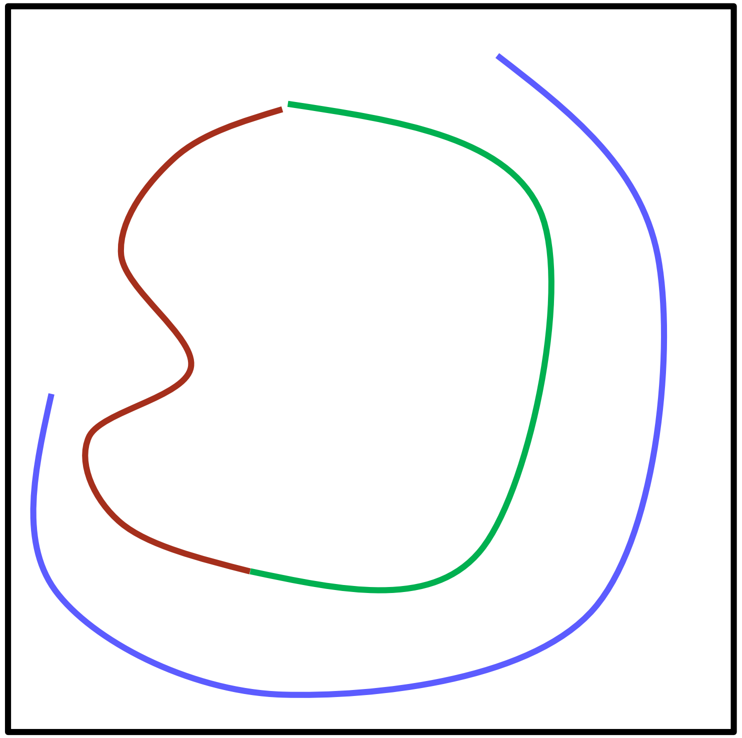


Threshold + Thinning

Threshold + Thinning

Canny edge detector: Hysteresis thresholding

- Trace each contour separately (e.g., using 4-connectedness).
- Apply two thresholds k_{high} and k_{low}
  - Start tracing a line only at pixels that exceed a high threshold k_{high}.
  - Continue tracing if the pixels exceed a lower threshold k_{low}.
- Typical threshold ratio: $k_{high} / k_{low} = 2$



Hysteresis thresholding

Original

High threshold
(strong edges)

Low threshold
(weak edges)

Hysteresis thresholding

Source: L. Fei-Fei

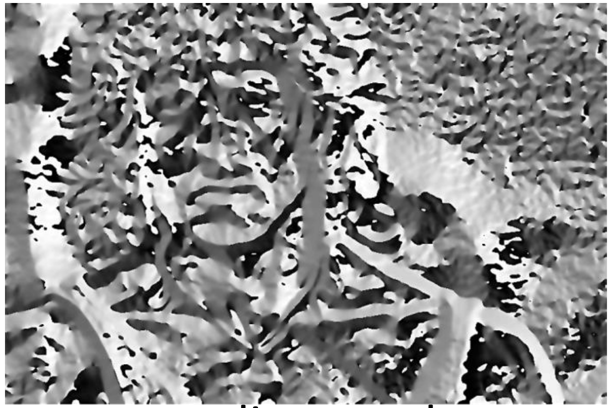
The Canny edge detector in a nutshell

1. Convolve the image by a derivative of a Gaussian.
2. Calculate the gradient magnitude and orientation
3. Non-maxima suppression (NMS)
    - Set low gradient magnitudes to zero to reduce the number of candidates in NMS
    - Thin edges to one-pixel width.
4. Trace the edges by hysteresis thresholding
    - Apply a high threshold on the magnitude to initialize a contours and continue tracing the contour until the magnitude falls below a low threshold.

Canny edge detector in “*action*”

Input image

gradient magnitude



gradient angle

thinned

Thresholded by hysteresis

Canny edge detector in “*action*”


Input image


gradient magnitude

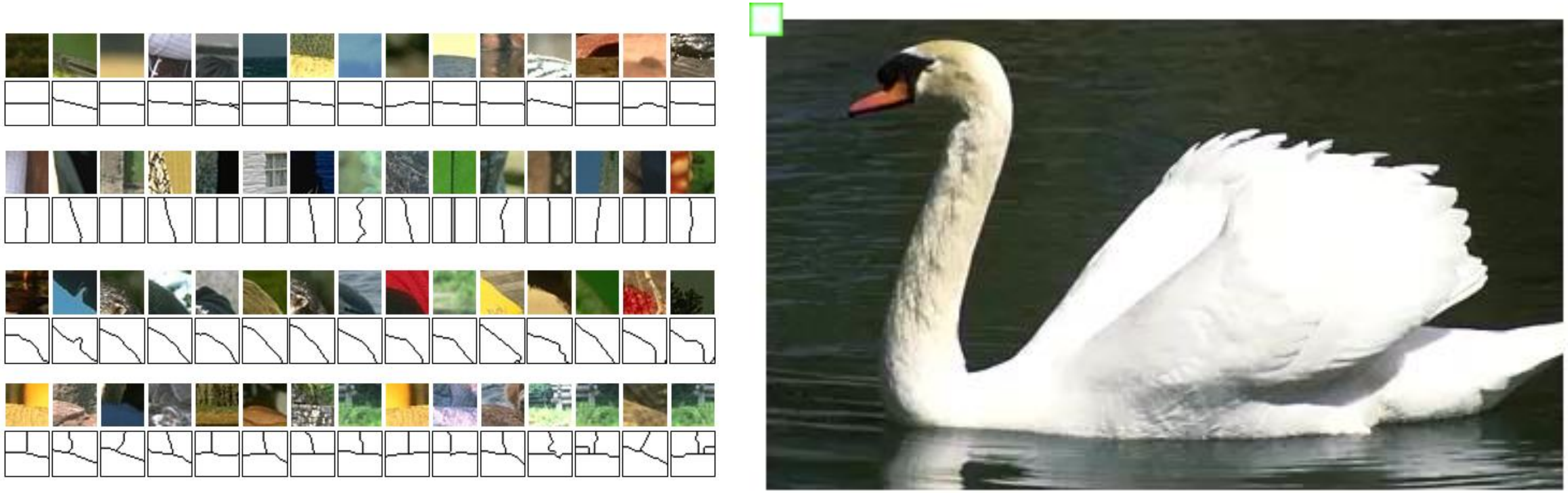

gradient angle


thinned


Thresholded by hysteresis

Beyond Canny edge detector

- Since Canny’s publication, lots of new approaches for edge detection by machine learning.
- Essentially, look at patches and learn what an edge is by inferring the structure from intensities.



Sketch Tokens, CVPR 2013. Joseph Lim, C. Zitnick, and P. Dollár

Beyond Canny edge detector

- Recently a CNN used as feature extractor and classifier

Kung and Fowlkes, Recurrent Pixel Embedding for Instance Grouping, CVPR2018

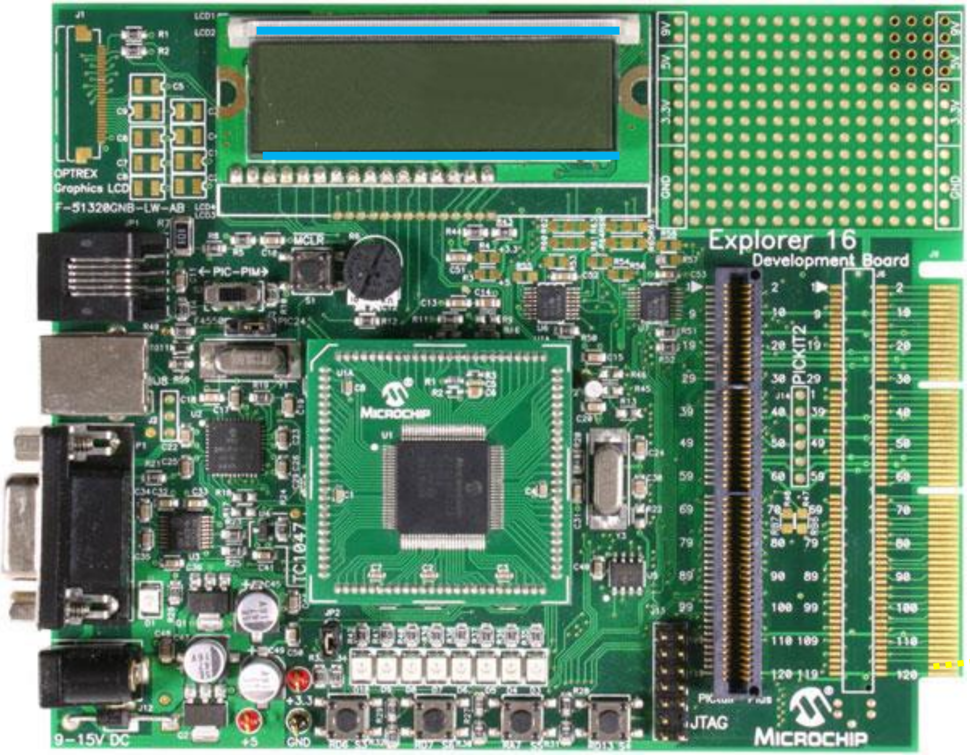
Machine perception

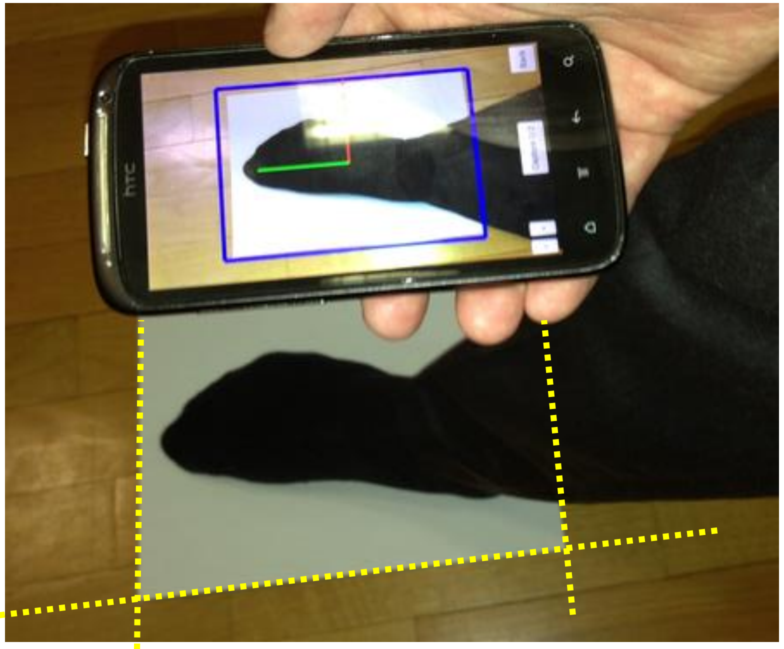
EDGE DETECTION BY PARAMETRIC MODELS

Example: line fitting

- Why should we fit lines?

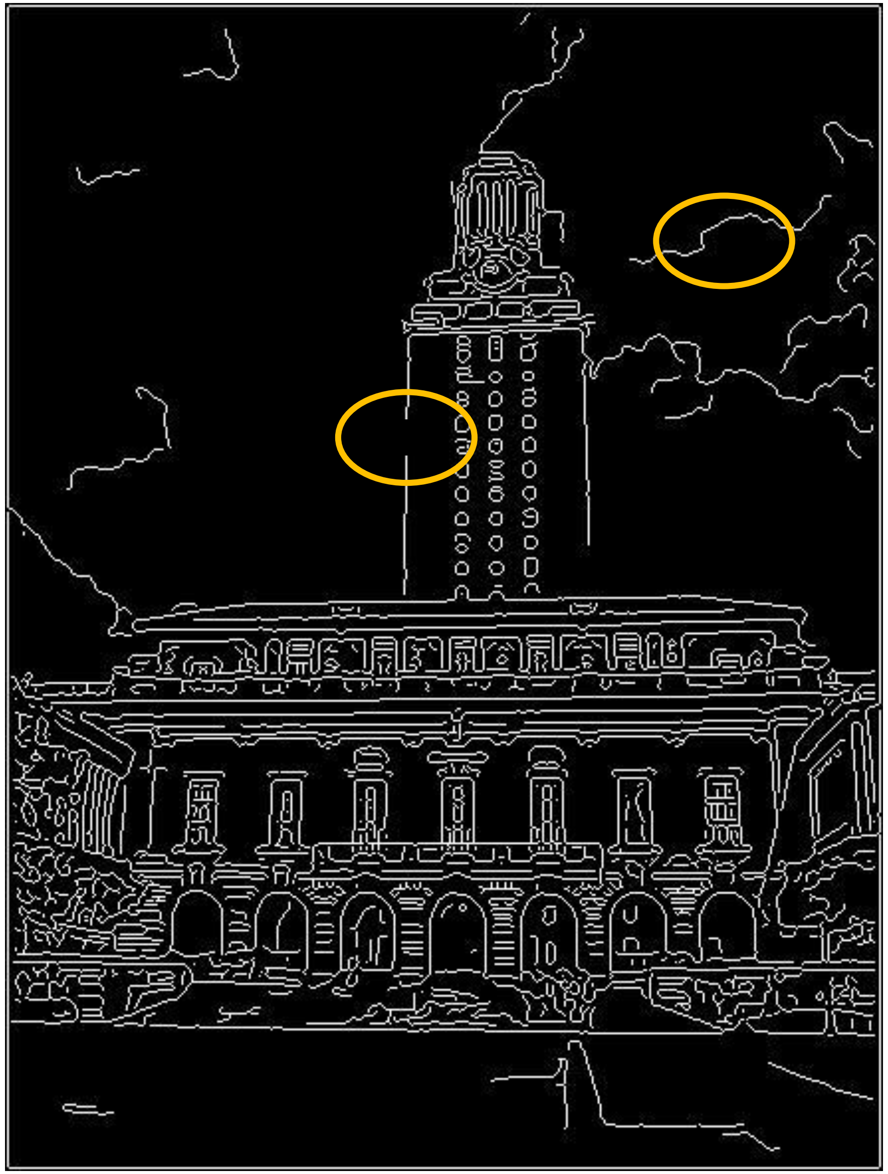
  Many scenes are composed of straight lines



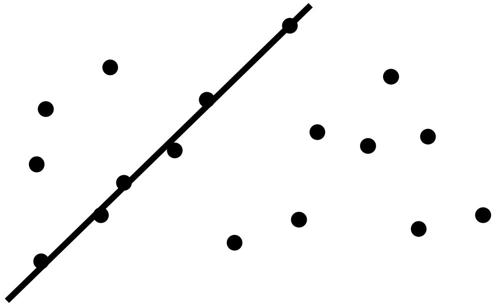


Slide credit: Kristen Grauman

Problems with line fitting



- Noisy edges, multiple models:
  - Which points correspond to which line, if at all?
- Some parts of lines are not detected:
  - How to find a line that connects the missing points?
- Noisy orientation:
  - How do we determine the unknown parameters of true lines?

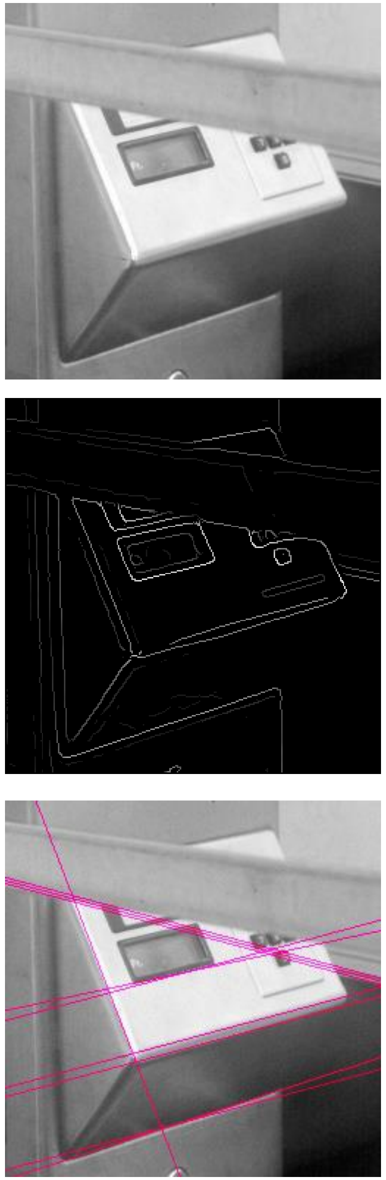


$y = 1.6x + 21.5$

Line fitting by voting for parameters

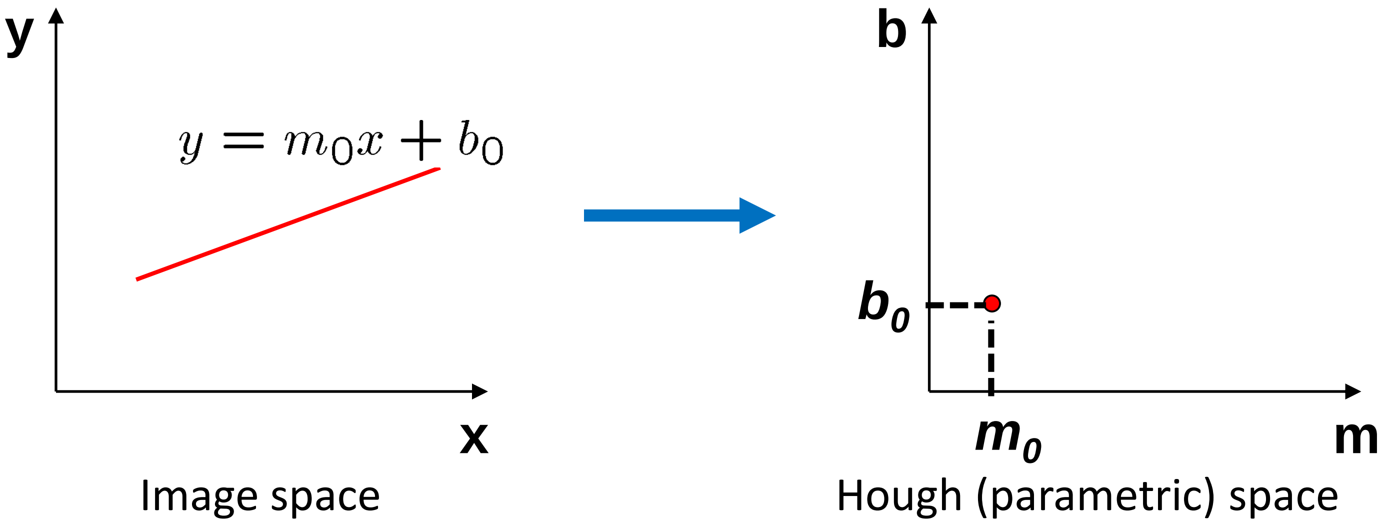
- Given a set of points, find the lines.
- How many lines?
- Which points correspond to which lines?

- *Hough Transform* is a voting technique that answers these questions.
- Main idea:
  1. For each edge point compute parameters of all possible lines passing through that point
  2. For each set of parameters cast a vote
  3. Select the lines (parameter combinations) that receive enough votes.



Slide credit: Kristen Grauman

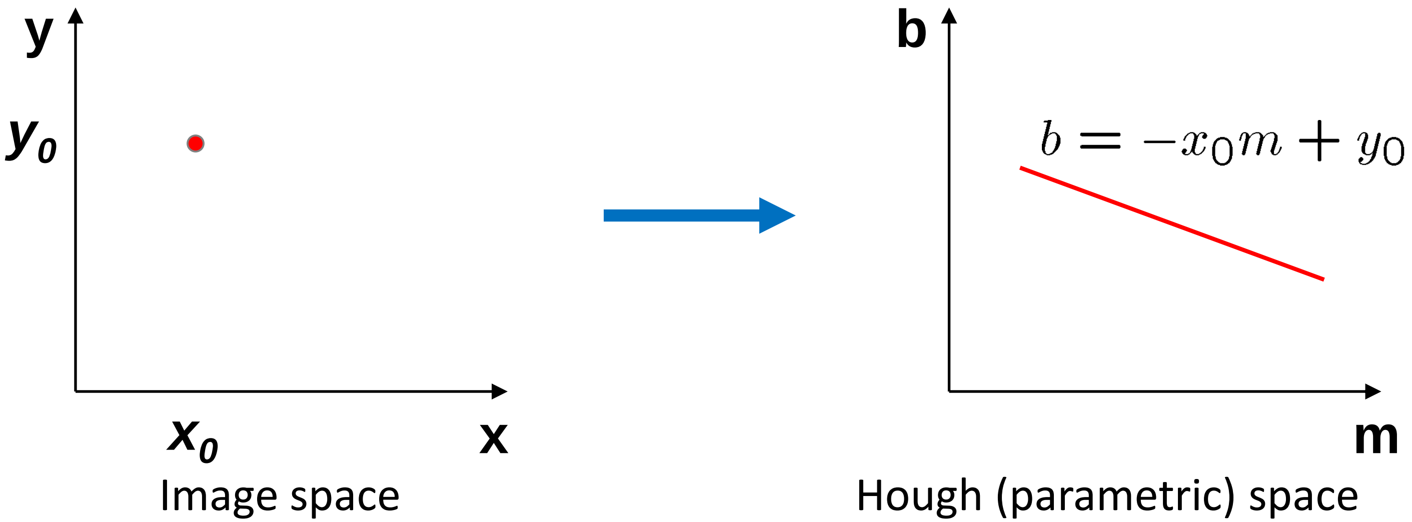
Hough space: straight lines



- Connection between spatial (x,y) and Hough space (m,b):
  - A line in image corresponds to a point in the Hough space.

Slide credit: Steve Seitz

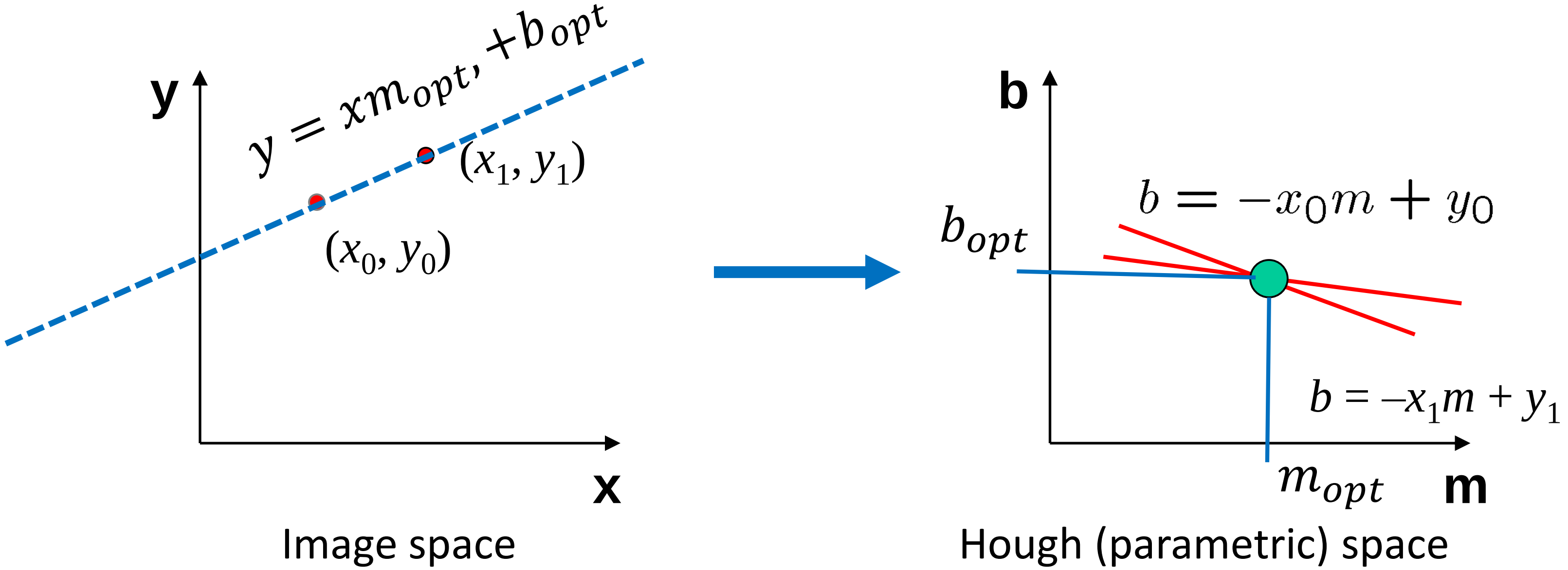
Hough space: straight lines



- Connection between spatial (x,y) and Hough space (m,b):
  - A line in image corresponds to a point in the Hough space.
  - Mapping from image to Hough space:
    - For a point (x,y), find all (m,b) for which this holds : $y = mx + b$

Slide credit: Steve Seitz

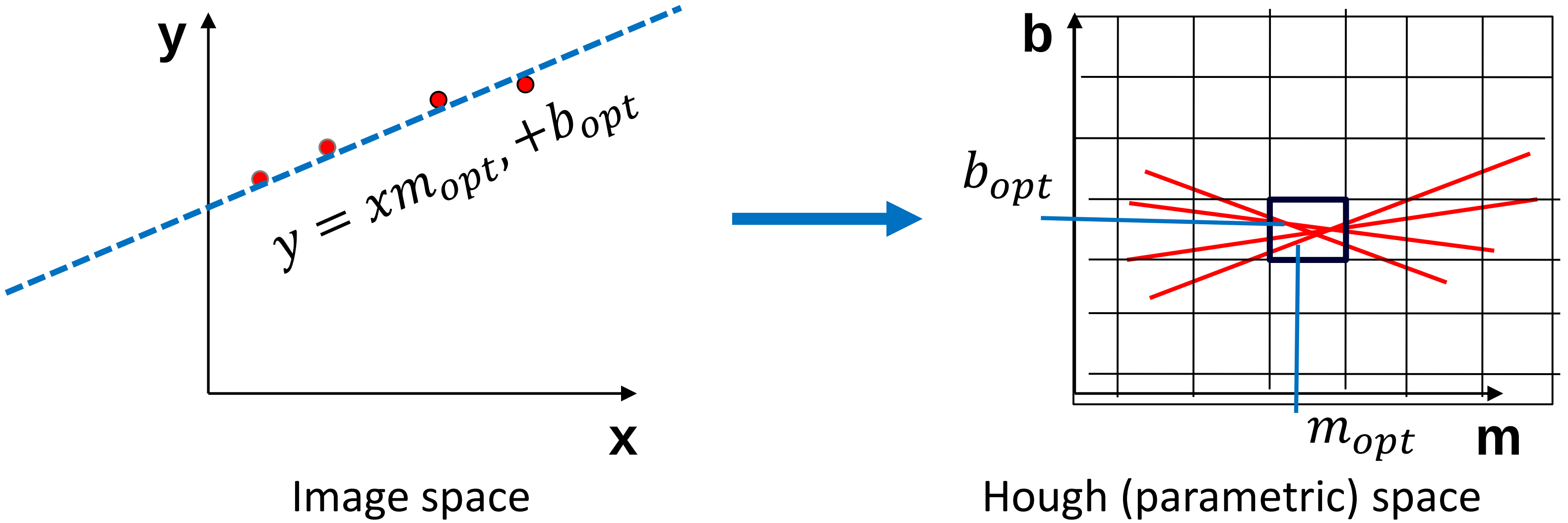
Hough space: straight lines



- Connection between spatial (x,y) and Hough space (m,b):
  - A line in image corresponds to a point in the Hough space.
  - Mapping from image to Hough space:
    - For a point (x,y), find all (m,b) for which this holds : $y = mx + b$

Slide credit: Steve Seitz

Hough space: straight lines



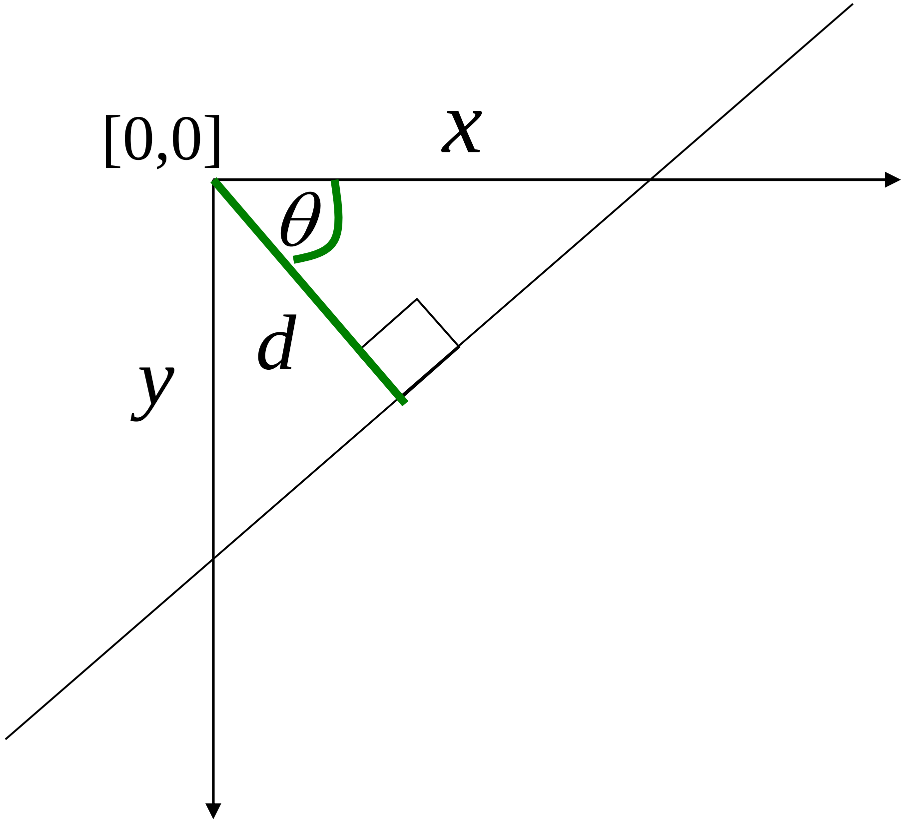
Discretize the parameter space…

- Connection between spatial (x,y) and Hough space (m,b):
  - A line in image corresponds to a point in the Hough space.
  - Mapping from image to Hough space:
    - For a point (x,y), find all (m,b) for which this holds : $y = mx + b$

Slide credit: Steve Seitz

Encode the line in polar coordinates

- Issue with Cartesian (m,b): infinite values for vertical lines!



d : perpendicular distance from the origin

θ: angle of perpendicular line with x axis

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

- Point in image $\Rightarrow$ sinusoid in Hough space

Algorithm: Straight lines

Using polar representation:

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

Basic Hough transform:

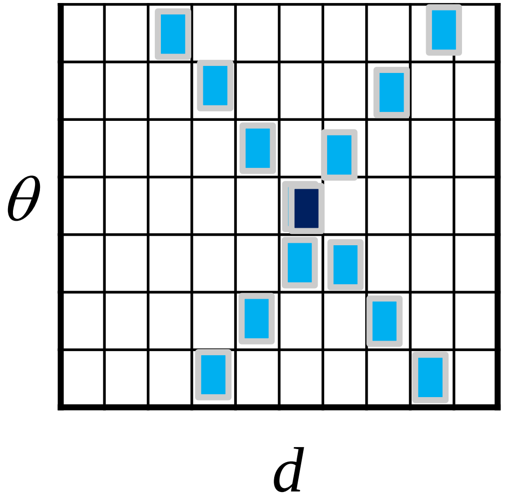
1. Initialize $H[d,\theta] = 0$.
2. For each edge point (x,y) in image

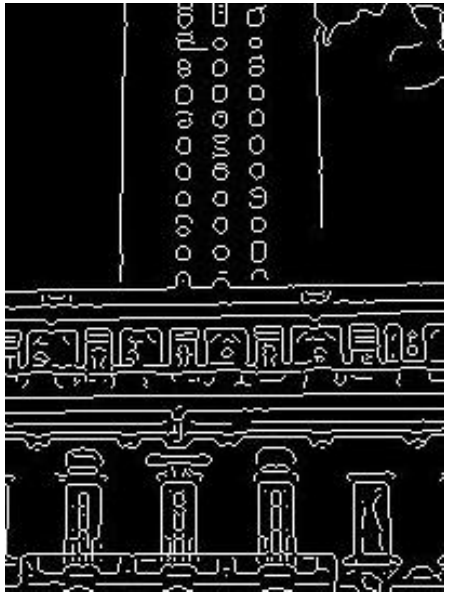
   For $\theta = 0$ to 180 // over quantized values!!

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

   H[d, θ] += 1
3. Find local maxima $\{d^i_{opt}, \theta^i_{opt}\}_{i=1:N}$ in accumulator array $H[d,\theta]$.
4. Detected line is defined by: $d^i_{opt} = x cos\theta^i_{opt} - sin\theta^i_{opt}$

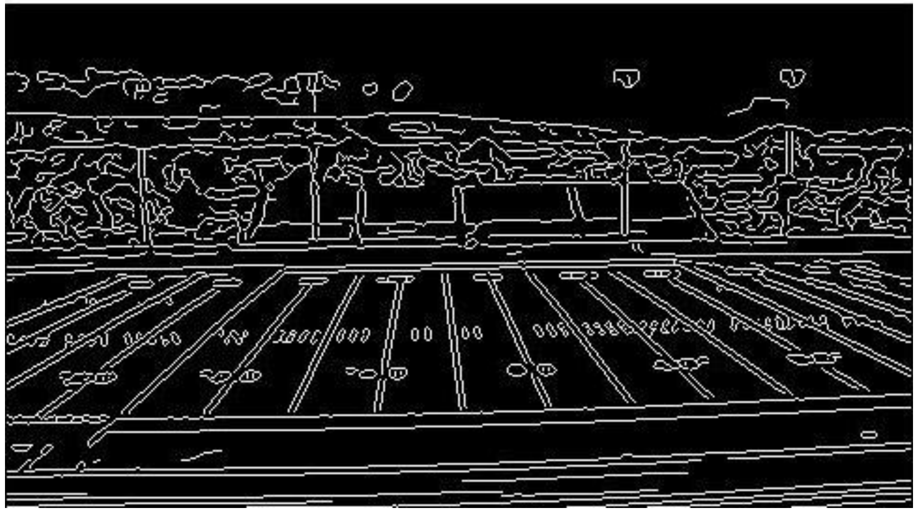
H: accumulator array (votes)

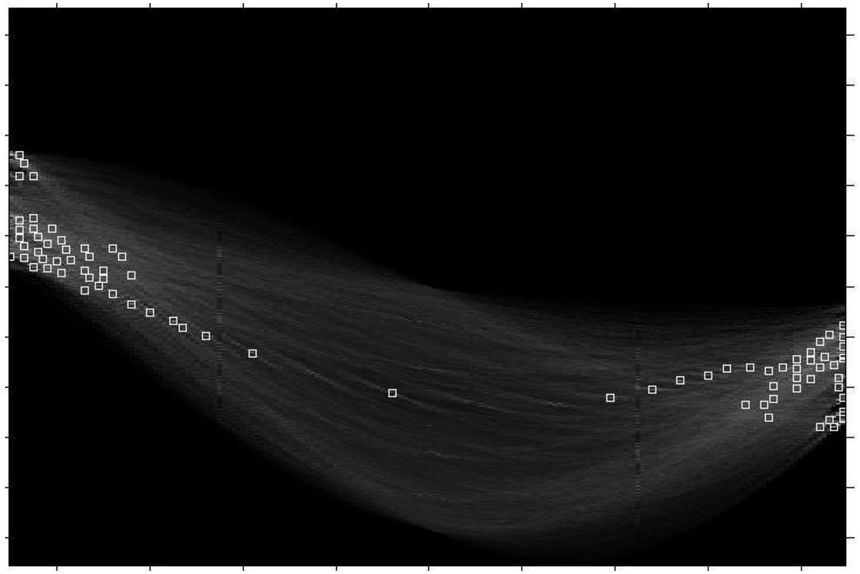


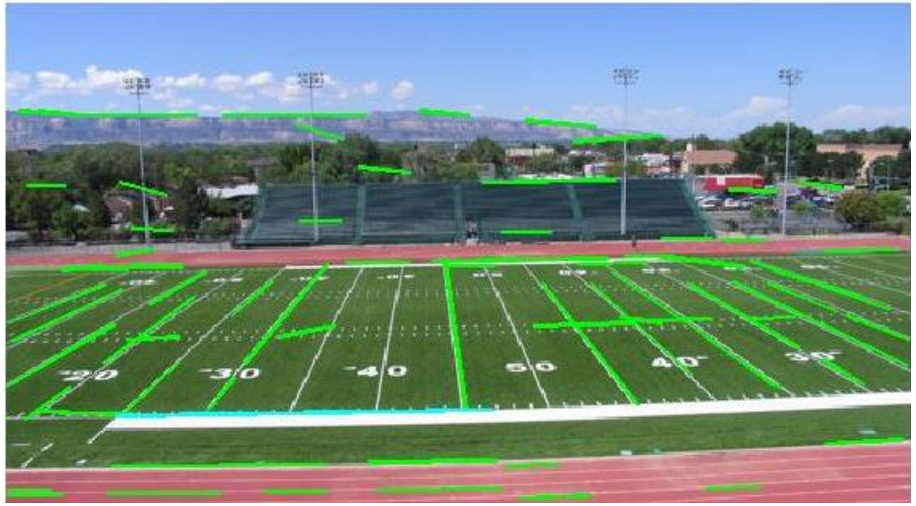


Hough line demo

Hough transform in action



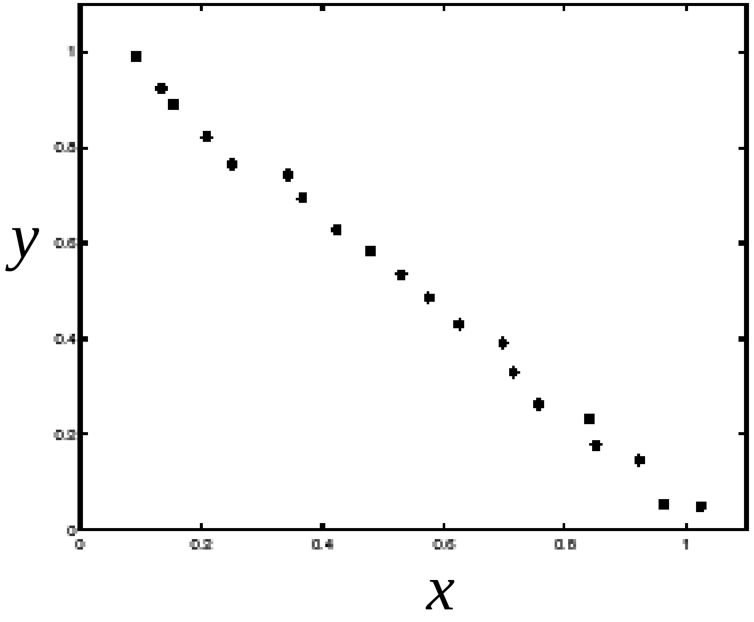




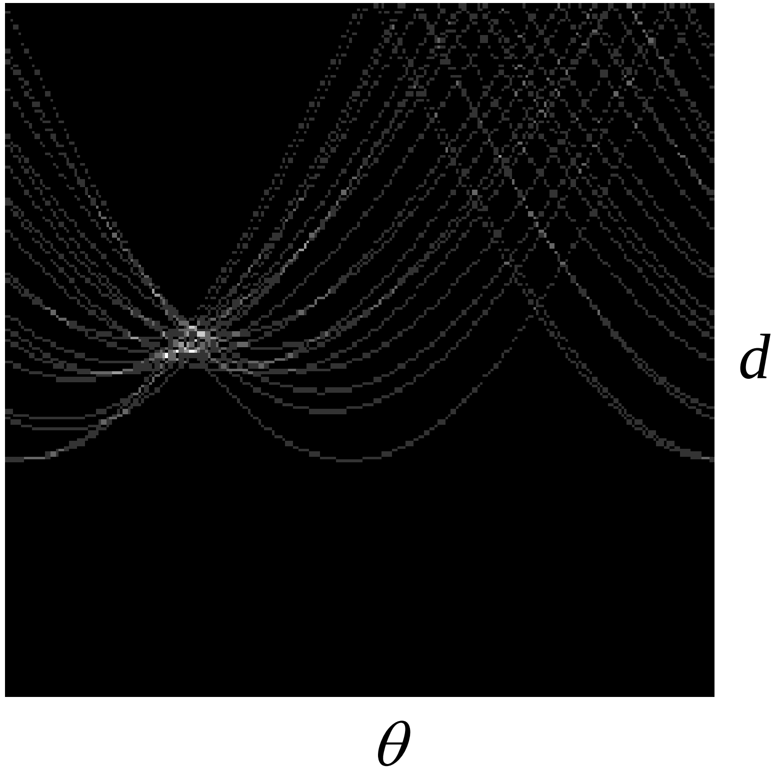
Only the longest segments along each detected line are shown here.

Slide credit: Kristen Grauman

Hough transform: Noise – binning



Coordinates of
edge points in image

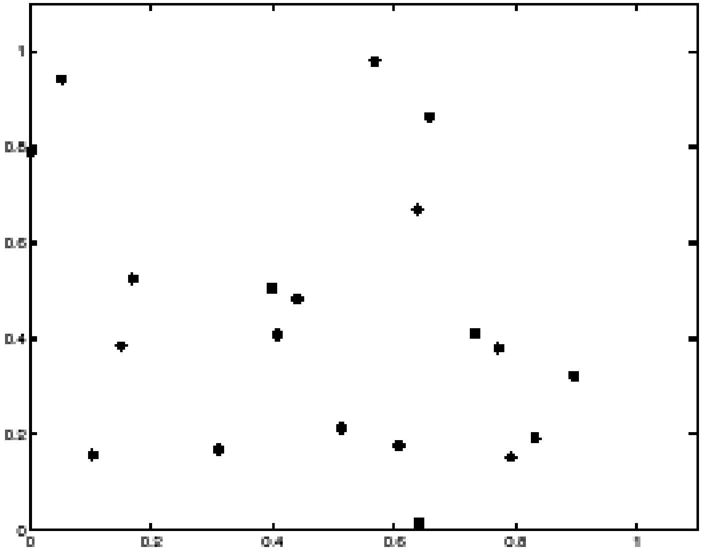


Votes

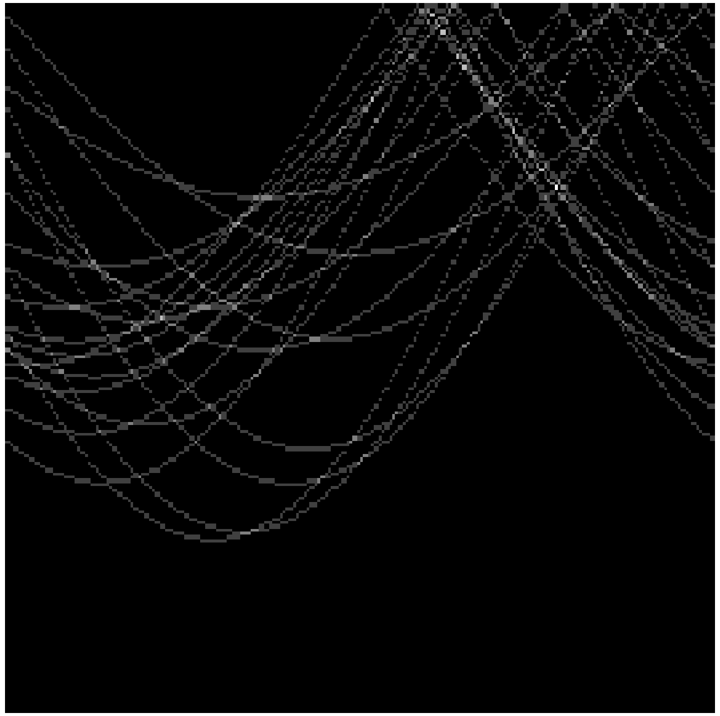
Are there any significant problems with the noise?

Slide credit: David Lowe

Hough transform: Noise – amplitude of votes



Coordinates of
edge points in image



Votes

Random points still form some local maxima in the accumulator array!

Slide credit: David Lowe

Hough transform: Extensions

Extension 1: Use the gradient *direction!*

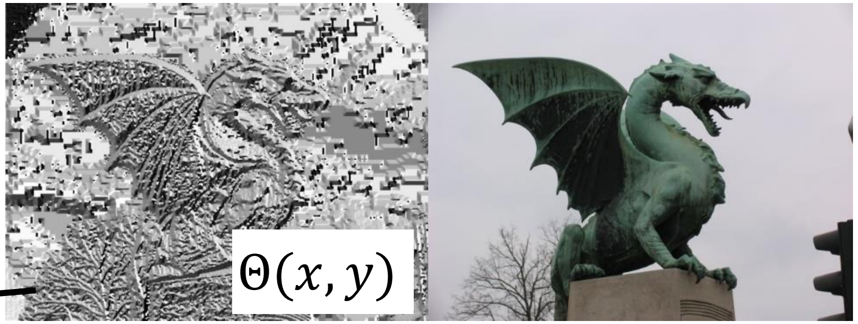
1. *same as standard HT*
2. For each edge point $[x,y]$

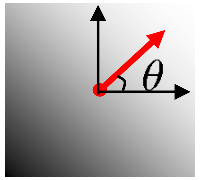
   θ = gradient direction at (x,y)

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

   $$\mathrm{H}[d,\theta] \mathrel{+}= 1$$
3. *same as standard HT*
4. *same as standard HT*

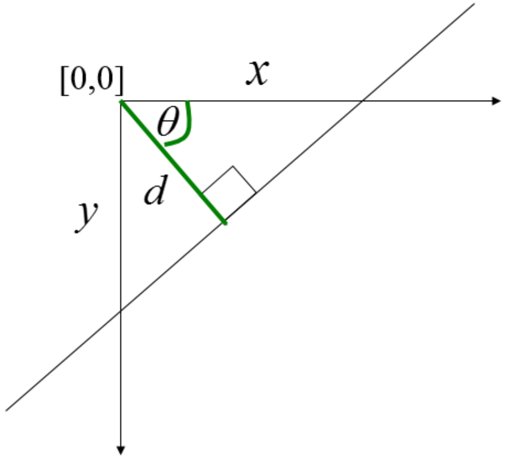
Reduces the number of degrees of freedom (dof)!



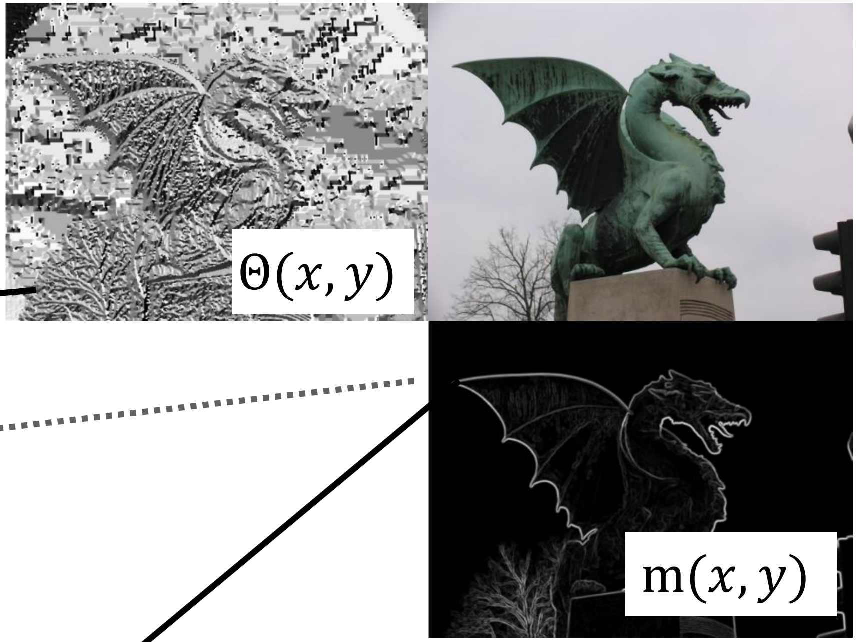


$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

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



Hough transform: Extensions



Extension 1: Use the gradient *direction!*

1. *same as standard HT*
2. For each edge point $[x,y]$

   θ = gradient direction at (x,y)

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

   $$H[d,\theta] \mathrel{+}= 1$$

3. *same as standard HT*
4. *same as standard HT*

Extension 2:

- Assign higher weight in votes to points with large edge magnitude. Instead $H[d,\theta] \mathrel{+}= 1$, use $H[d,\theta] \mathrel{+}= \mathrm{m}(x,y)$.

- These extensions can be applied in general:

  line, circles, squares, general shapes...

Slide credit: Kristen Grauman

Hough transform for circles

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

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

- Example of center detection at known radius r

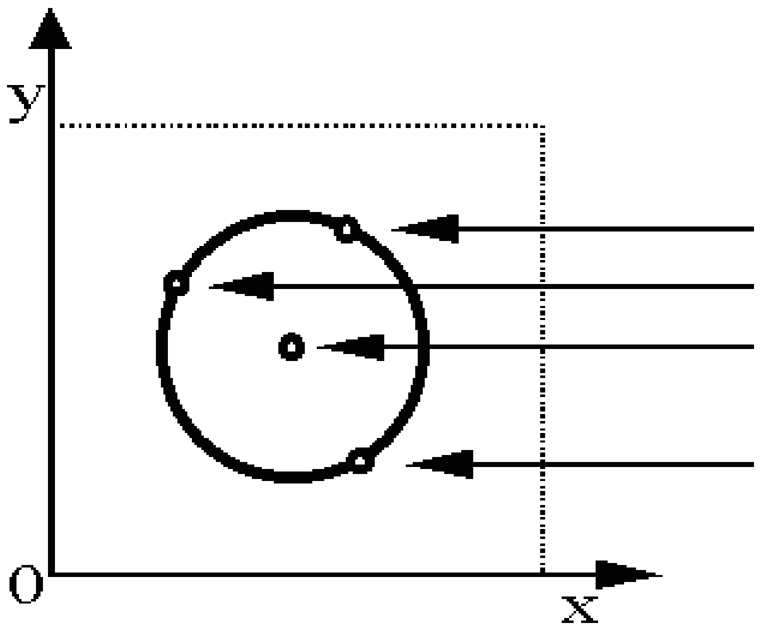
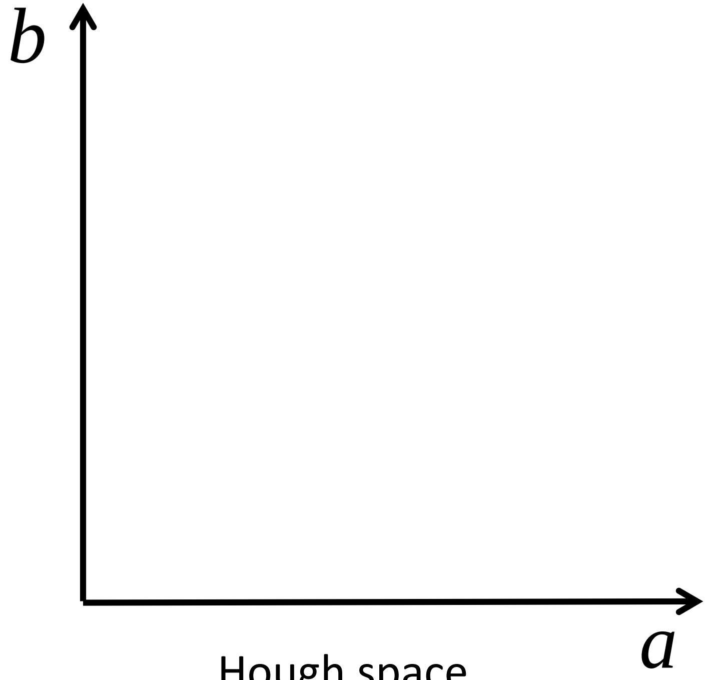


Image space



Hough space

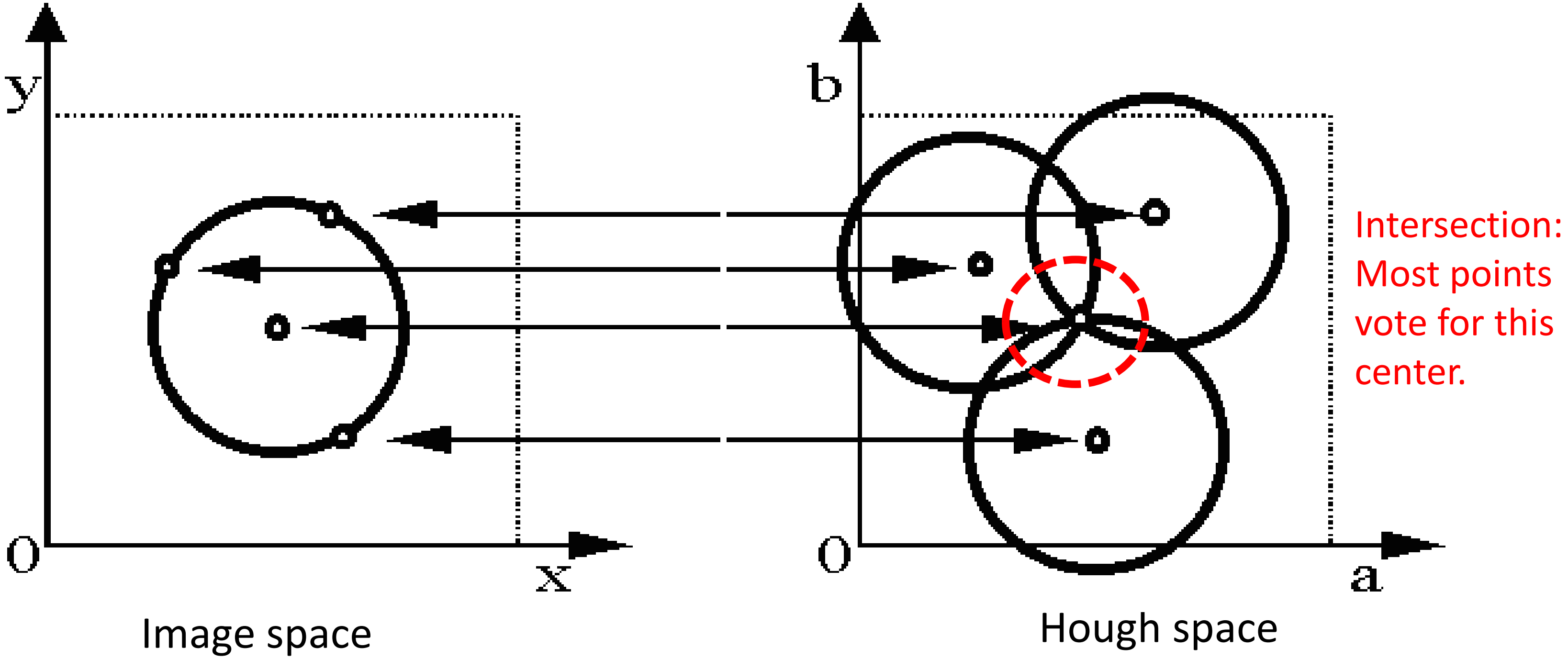
Slide credit: Kristen Grauman

Hough transform for circles

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

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

- Example of center detection at known radius r



Intersection: Most points vote for this center.

Slide credit: Kristen Grauman

Hough transform for circles

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

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

- Unknown radius r – *How many dimensions in Hough Space?*

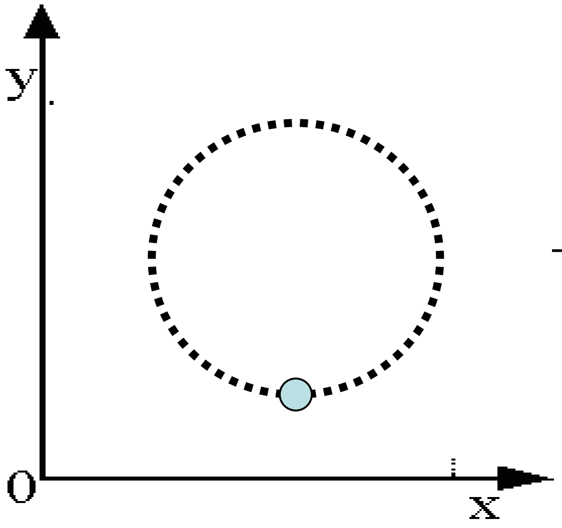
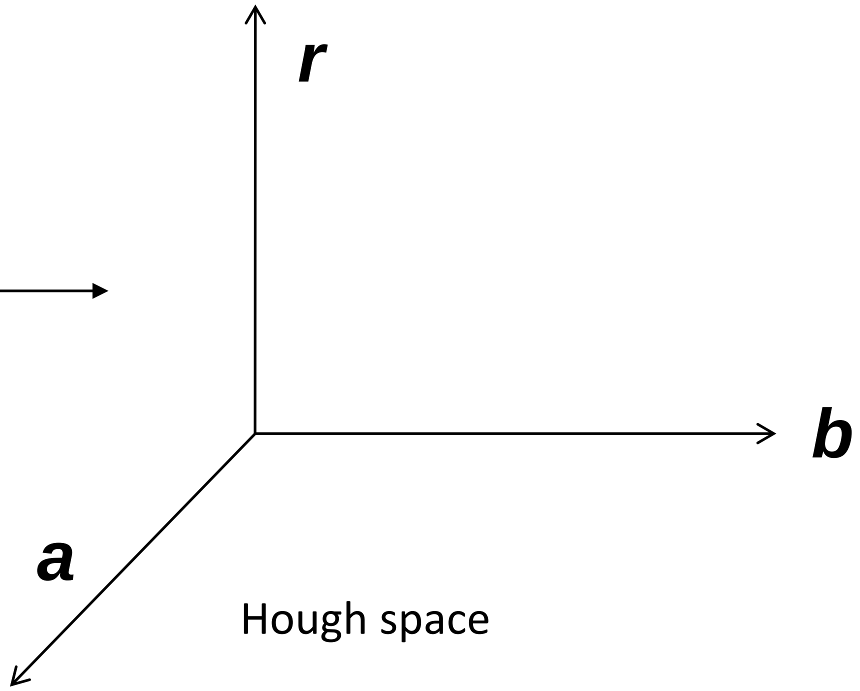


Image space



Hough space

Slide credit: Kristen Grauman

Hough transform for circles

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

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

- Unknown radius r

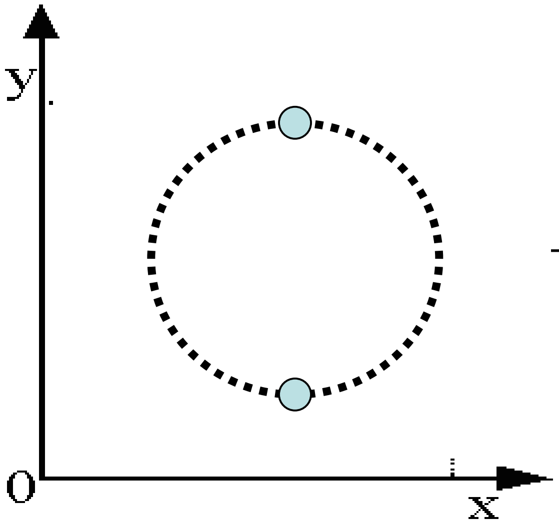
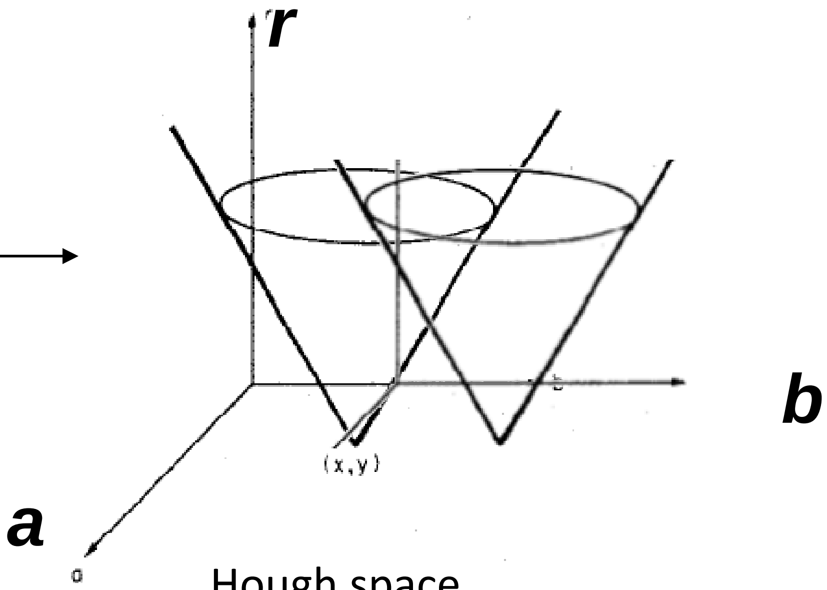


Image space



Hough space

Slide credit: Kristen Grauman

Hough transform for circles

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

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

- Unknown radius r

But assume we know the gradient direction!

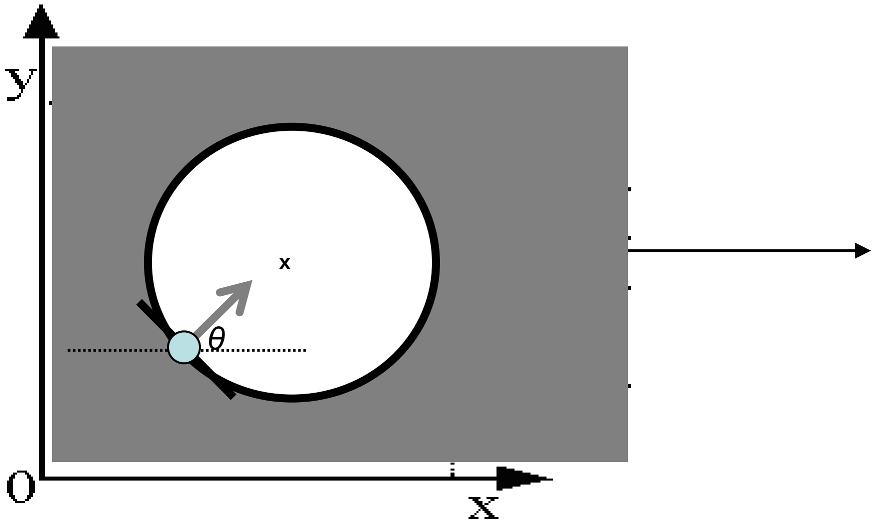
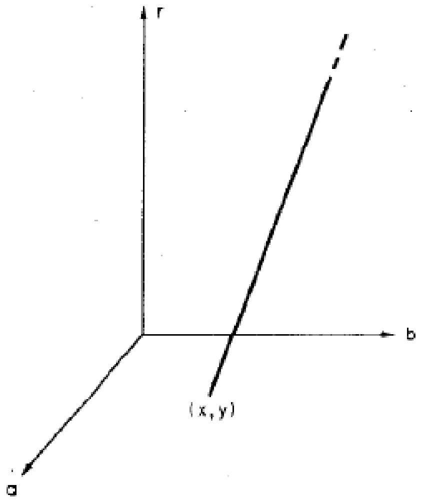


Image space



Hough space

Slide credit: Kristen Grauman

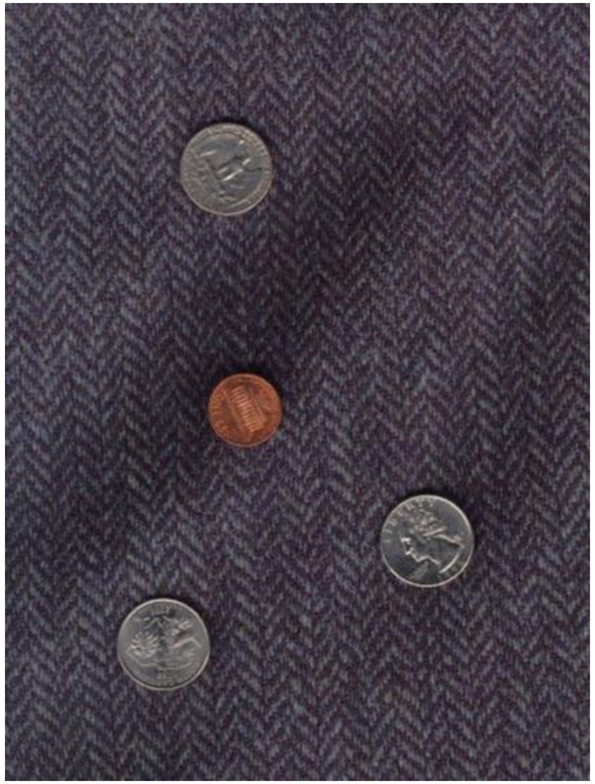
Hough transform for circles

```
For each edge pixel $(x,y)$ :
    For each radius value $r$:
        For each gradient direction $\theta$:
            // or use the estimated direction only
        $a = x - r\cos(\theta)$
        $b = y + r\sin(\theta)$
        $H[a,b,r] += 1$ (or the magnitude)
    end for
  end for
end for
```

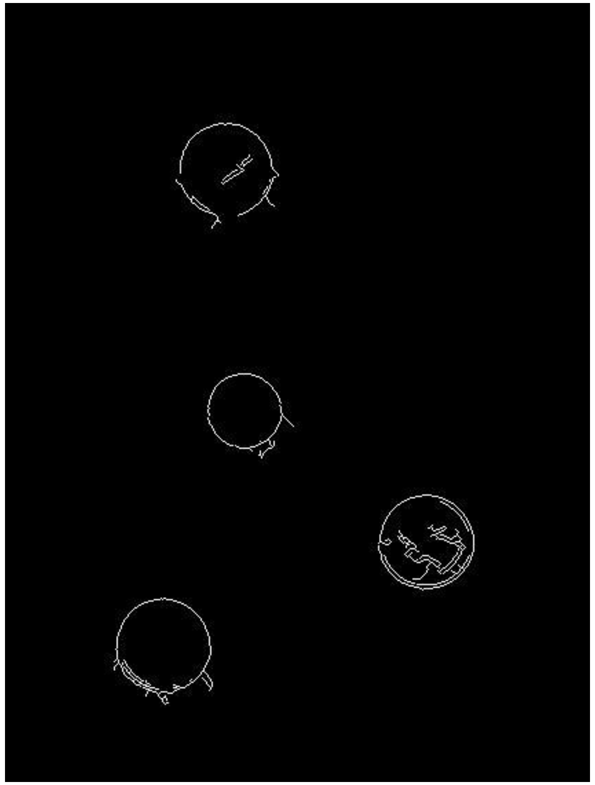
Slide credit: Kristen Grauman

Hough circle detection in action!

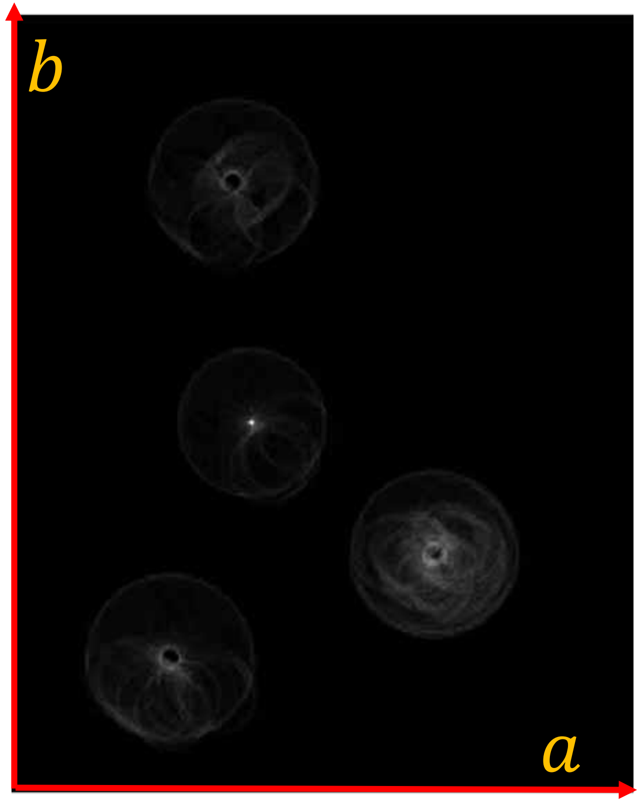
Original



Edges (e.g., Canny)



Votes for penny



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

Given a known radius:

$r = 50\,pixels,$

what are the center coordinates, i.e.,

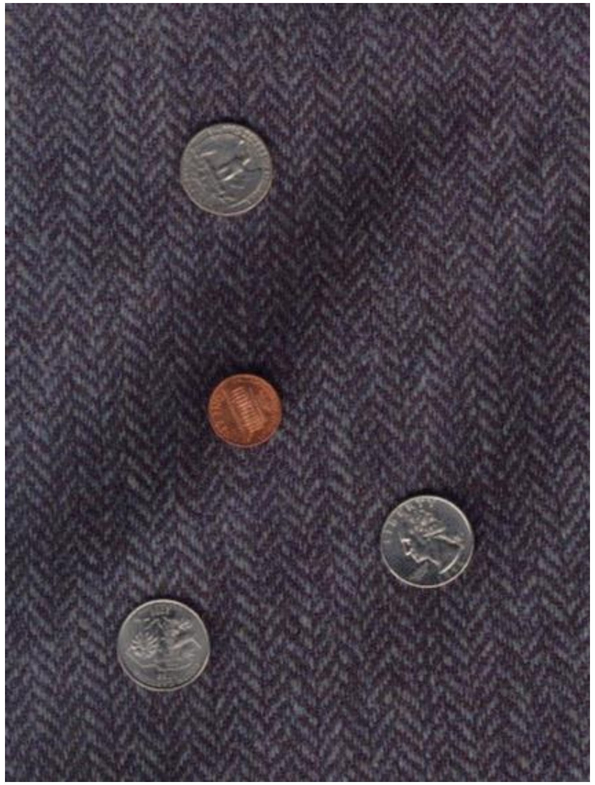
$a = ?\ pixels,$
$b = ?\,pixels\ .$

Comment: here we use a separate HT for each coin size.

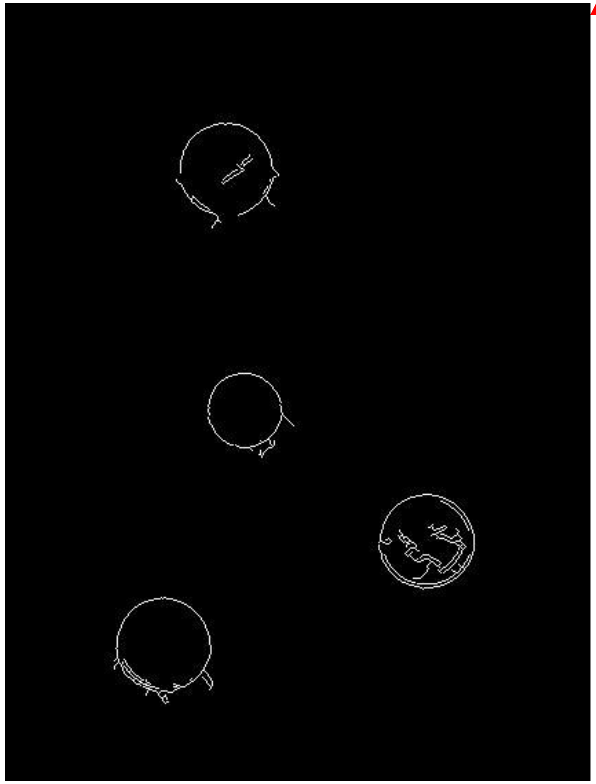
Images from: Vivek Kwatra

Hough circle detection in action!

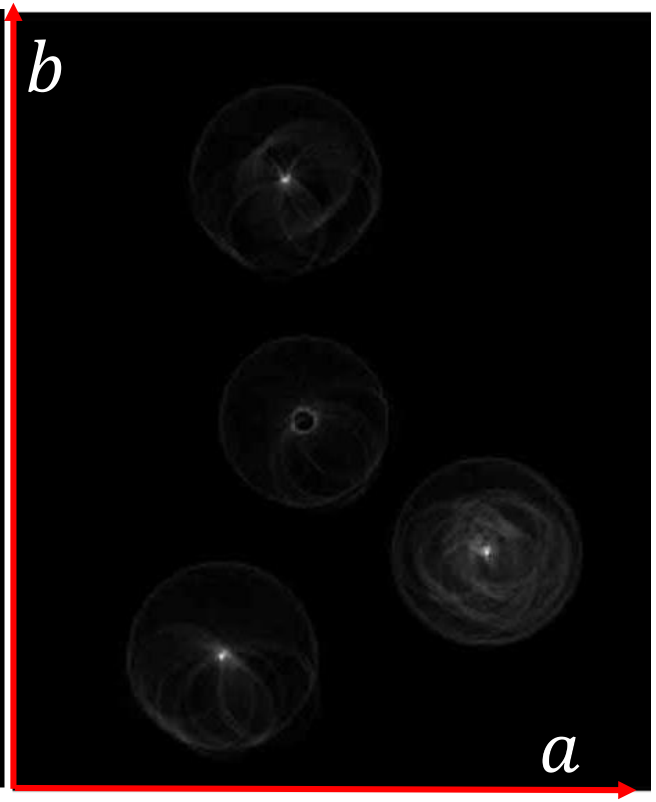
Original



Edges (e.g., Canny)



Votes for 25cent



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

Given a known radius:

$r = 80\ pixels$,

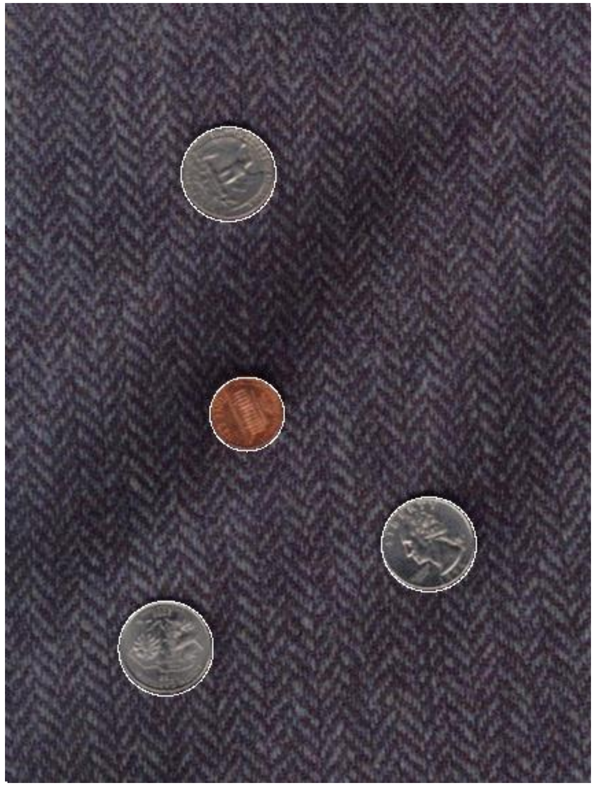
what are the center coordinates, i.e.,

$a = ?\ pixels$,
$b = ?\ pixels$.

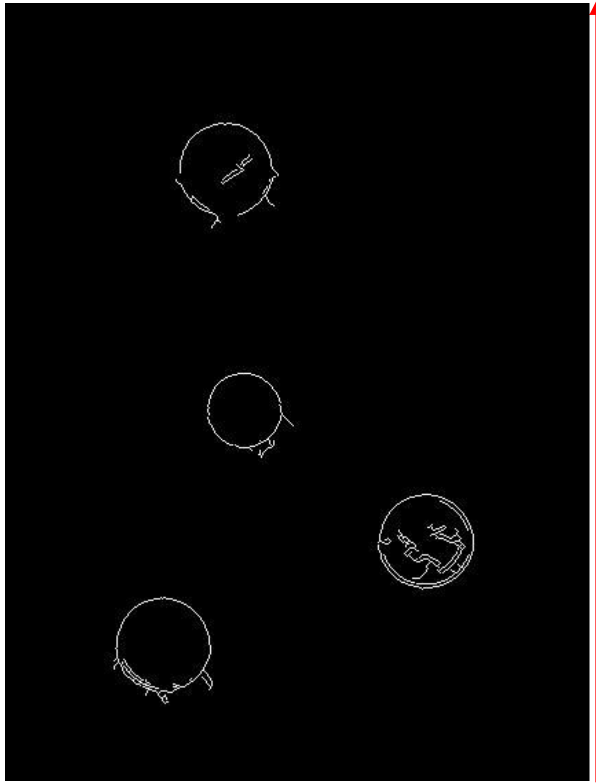
Comment: here we use a separate HT for each coin size.

Images from: Vivek Kwatra

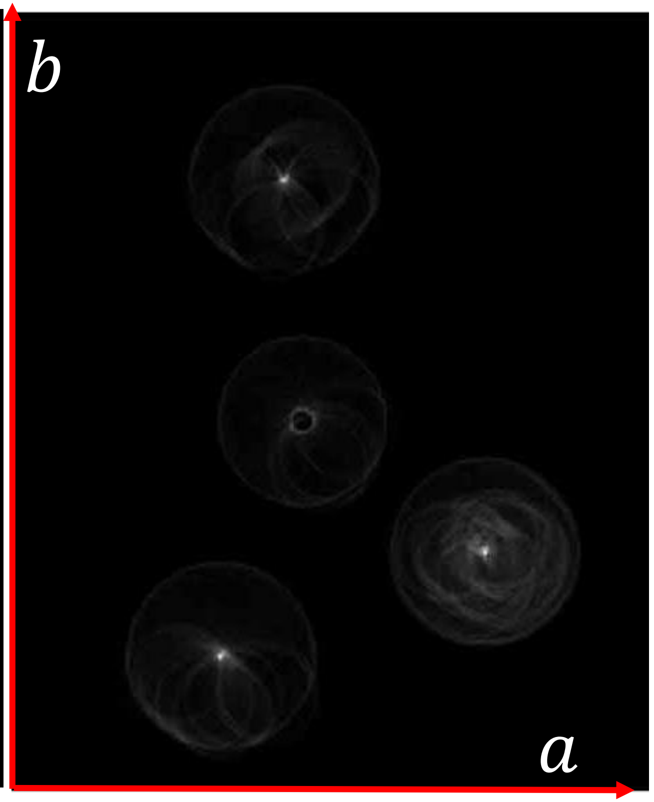
Hough circle detection in action!



Combined detections



Edges (e.g., Canny)



Votes for 25cent

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

Given a known radius:

$r = 50\ or\ 80\ pixels,$

what are the center coordinates, i.e.,

$a = ?\ pixels,$
$b = ?\ pixels\ .$

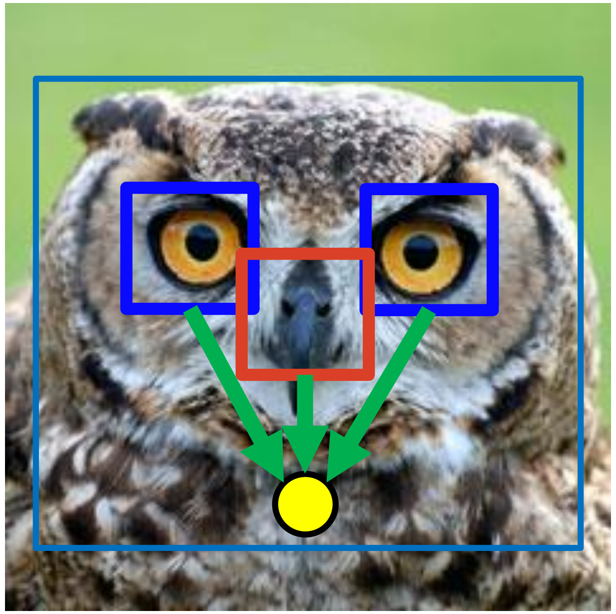
Comment: here we use a separate HT for each coin size.

Images from: Vivek Kwatra

Generalized Hough transform (GHT)

Building a model to detect objects by GHT – intuition:

- Assume we know how to detect parts (recognize+localize), i.e., eyes and beak of an howl. Task: *create an howl head detector.*
- Encode parts by displacements to the neck center.



The owl head model:

Given a part, where is the neck center?

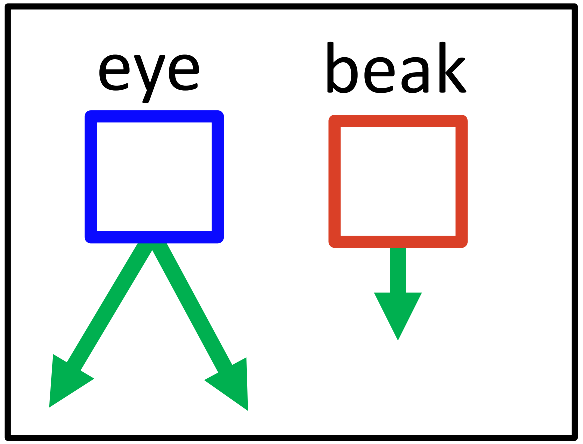
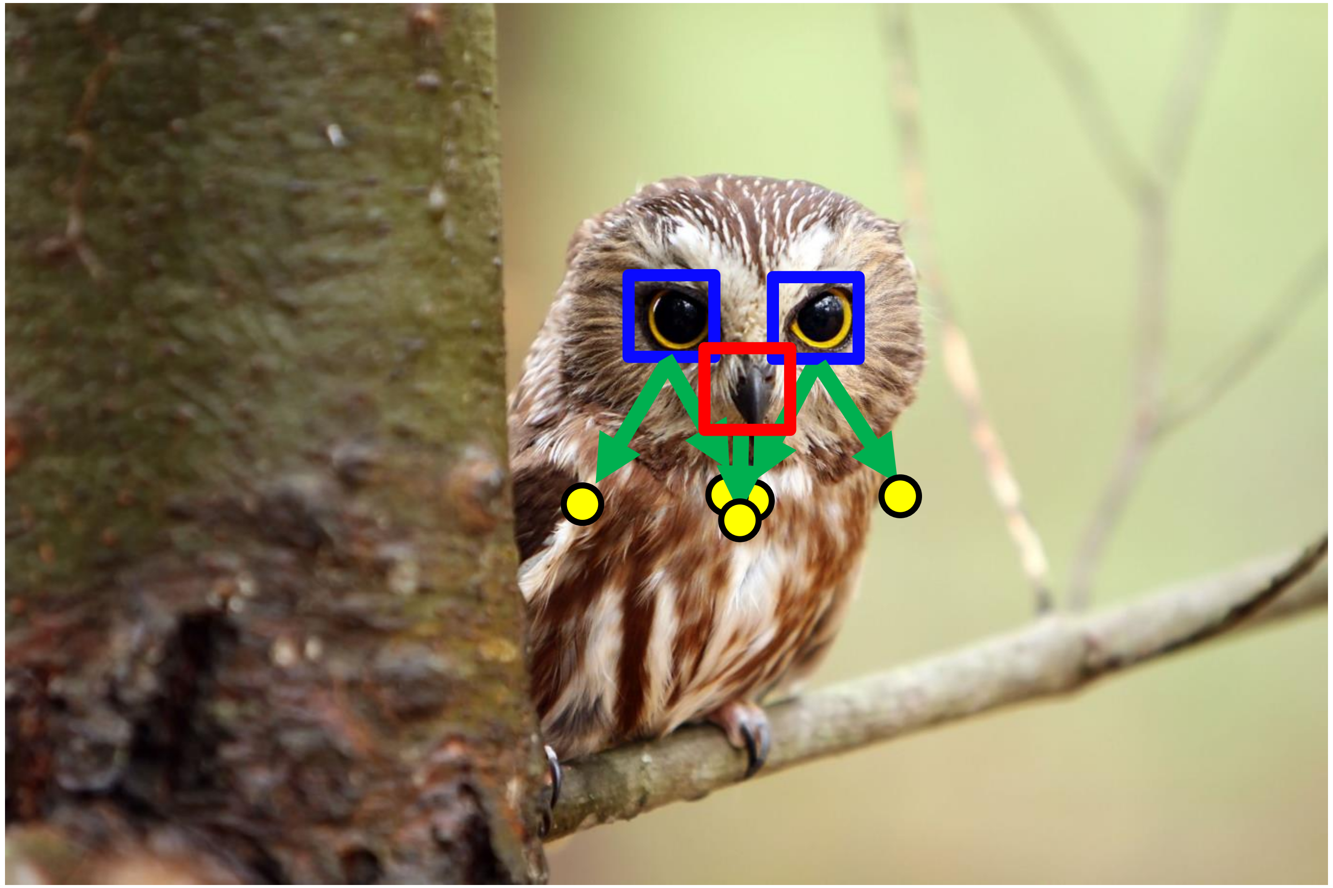


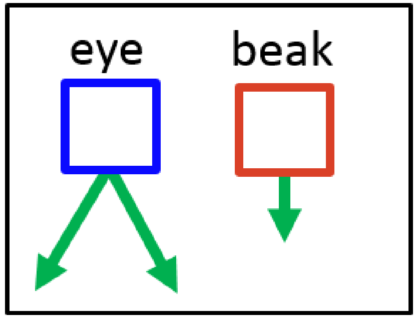
Image from: http://www.exposureguide.com/

Generalized Hough transform (GHT)

- Detection – intuition



The owl head model:



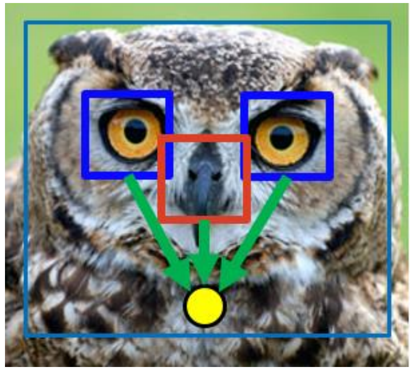
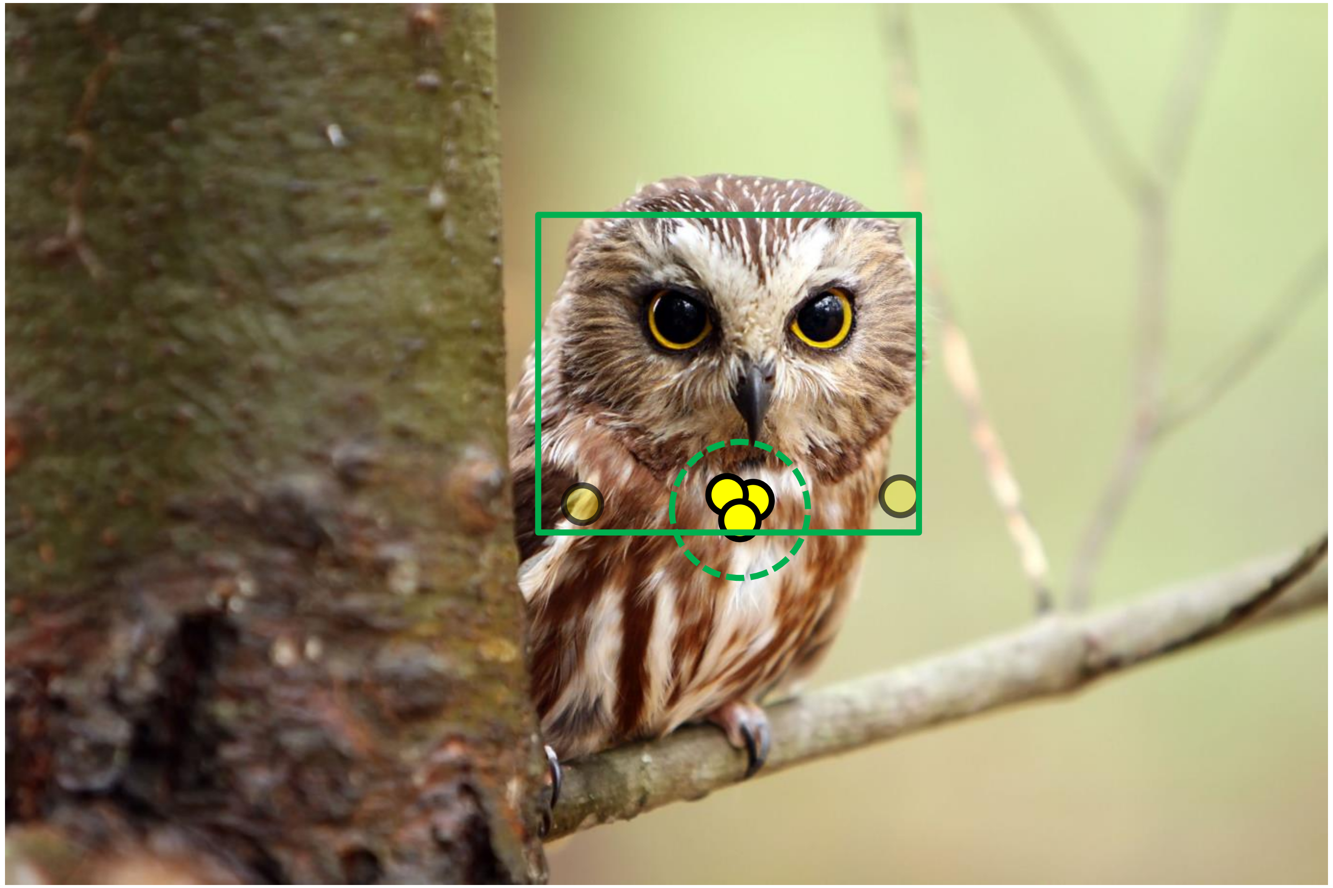


Image from: http://pugetsoundbirds.org/projects/owl-project/

Generalized Hough transform (GHT)

- Detection – intuition



The owl head model:

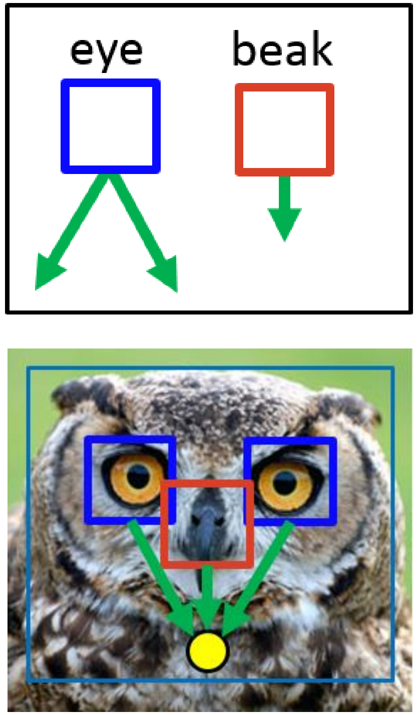
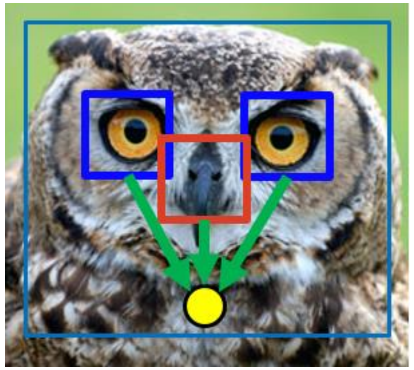
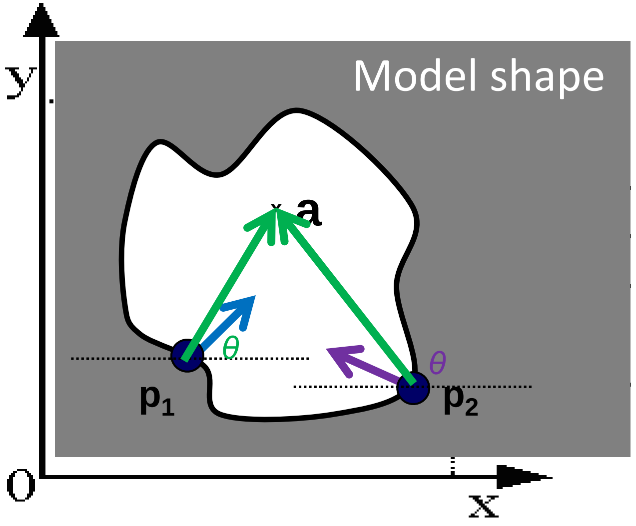


Image from: http://pugetsoundbirds.org/projects/owl-project/

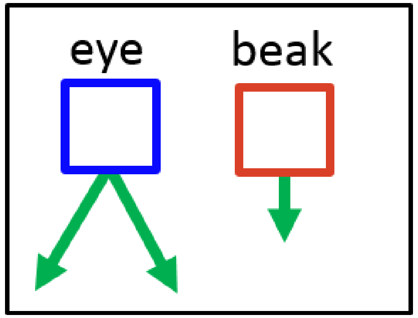
GHT for shape-based models

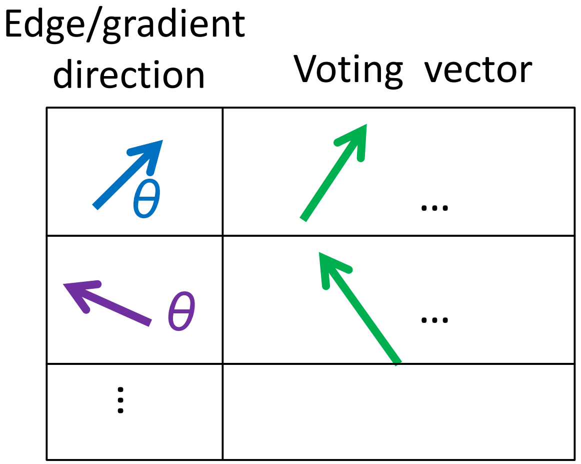
- Define the shape model by edge points and a reference point.





The owl head model:





Model learning:

For each edge point calculate the displacement vector to the reference point: $\mathbf{r} = \mathbf{a} - \mathbf{p}_i$.

Collect displacements in table, indexed by gradient direction θ.

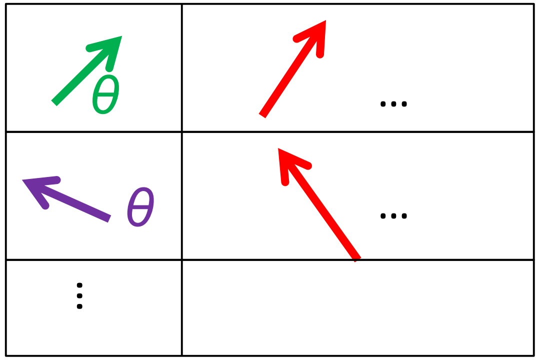
[Dana H. Ballard, Generalizing the Hough Transform to Detect Arbitrary Shapes, 1980] Slide credit: Kristen Grauman

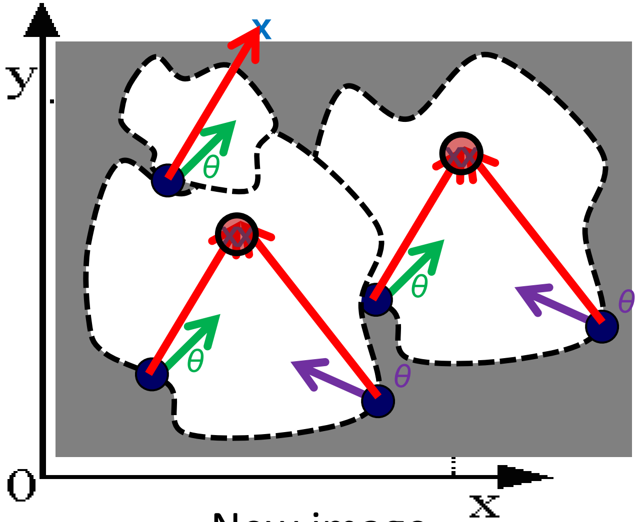
GHT for shape-based models

Detection:

For each edge point:

- Use its gradient orientation θ to index into the table.
- Use the displacement vectors **r** to cast a vote for the center.





New image

Assumption: the only transformation is the translation (orientation+scaling are fixed)

Slide credit: Kristen Grauman

Hough transform line detection: Practical advices

- First minimize irrelevant responses
  (use only edges with significant magnitude of gradient)
- Appropriately discretize the parametric space
  - Too coarse: votes from different lines fall into the same accumulator
  - Too fine: losing lines – due to noise, collinear points cast votes into nearby (BUT DIFFERENT) accumulators.

- Vote for neighboring cells as well
  - Correct: cast a vote by a Gaussian or a bilinear interpolation
  - Approximate: convolve the voting array by a Gaussian
- Use the gradient direction to reduce the number of free parameters

Hough transform: +/-

Pros

- Each point is processed independently:
  - robustness to partial occlusion,
  - highly parallelizable.
- Robustness to noise: noise will unlikely contribute consistently to a single cell
- Can detect multiple instances of a single model in one pass.

Cons

- Time complexity increases exponentially with the number of free parameters.
- Spurious shapes may generate false local maxima in the parametric space.
- Quantization: Not particularly easy to choose a proper accumulator cell size – Application dependent!

References

- David A. Forsyth, Jean Ponce, Computer Vision: A Modern Approach (2nd Edition), (prva izdaja dostopna na spletu)
- R. Szeliski,Computer Vision: Algorithms and Applications, 2010
- R. Hartley, A. Zisserman, Multiple View Geometry in Computer Vision, 2nd Edition, Cambridge University Press, 2004
- Kristen Grauman, „Computer Vision“, lectures