

CS 1674: Intro to Computer Vision

Grouping: Edges, Lines, Circles, Segments

Prof. Adriana Kovashka
University of Pittsburgh
September 22, 2020

Plan for this lecture

- Edges
 - Extract gradients and threshold
- Lines and circles
 - Find which edge points are collinear or belong to another shape e.g. circle
 - Automatically detect and ignore outliers
- Segments
 - Find which pixels form a consistent region
 - Clustering (e.g. K-means)

Edge detection

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

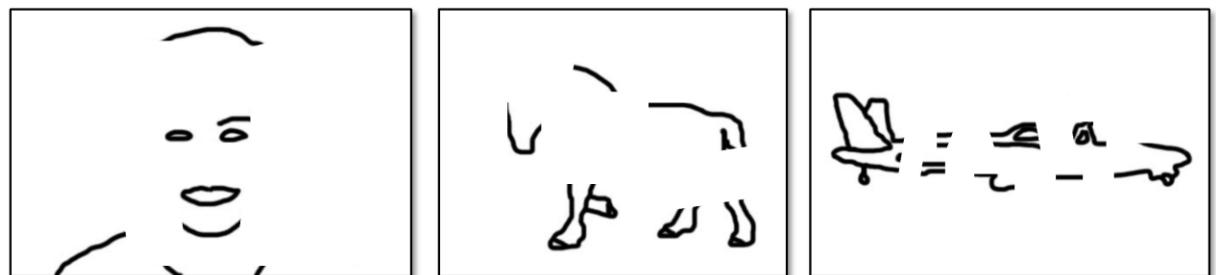


Figure from J. Shotton et al., PAMI 2007

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

Designing an edge detector

- Criteria for a good edge detector
 - **Good categorization (edge vs not edge)**
 - find all real edges, ignoring noise or other artifacts
 - **Good localization**
 - detect edges as close as possible to the true edges
 - return one point only for each true edge point
(true edges = the edges humans drew on an image)
- Cues of edge detection
 - Bottom-up: Differences in color, intensity, or texture across the boundary
 - Top-down: Continuity and closure, high-level knowledge

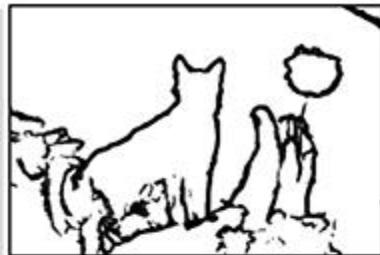
Examples of edge detection results



(a) original image



(b) ground truth



(c) HED: output



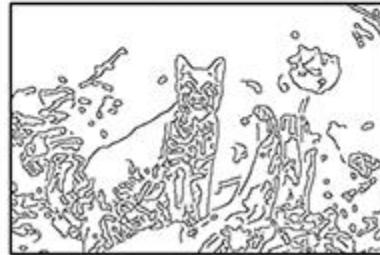
(d) HED: side output 2



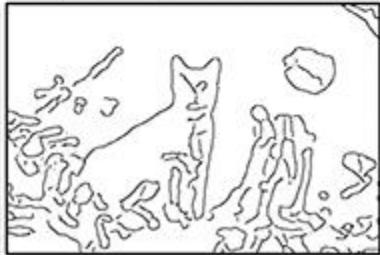
(e) HED: side output 3



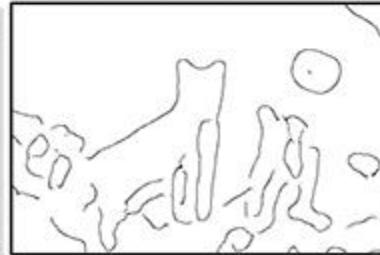
(f) HED: side output 4



(g) Canny: $\sigma = 2$



(h) Canny: $\sigma = 4$

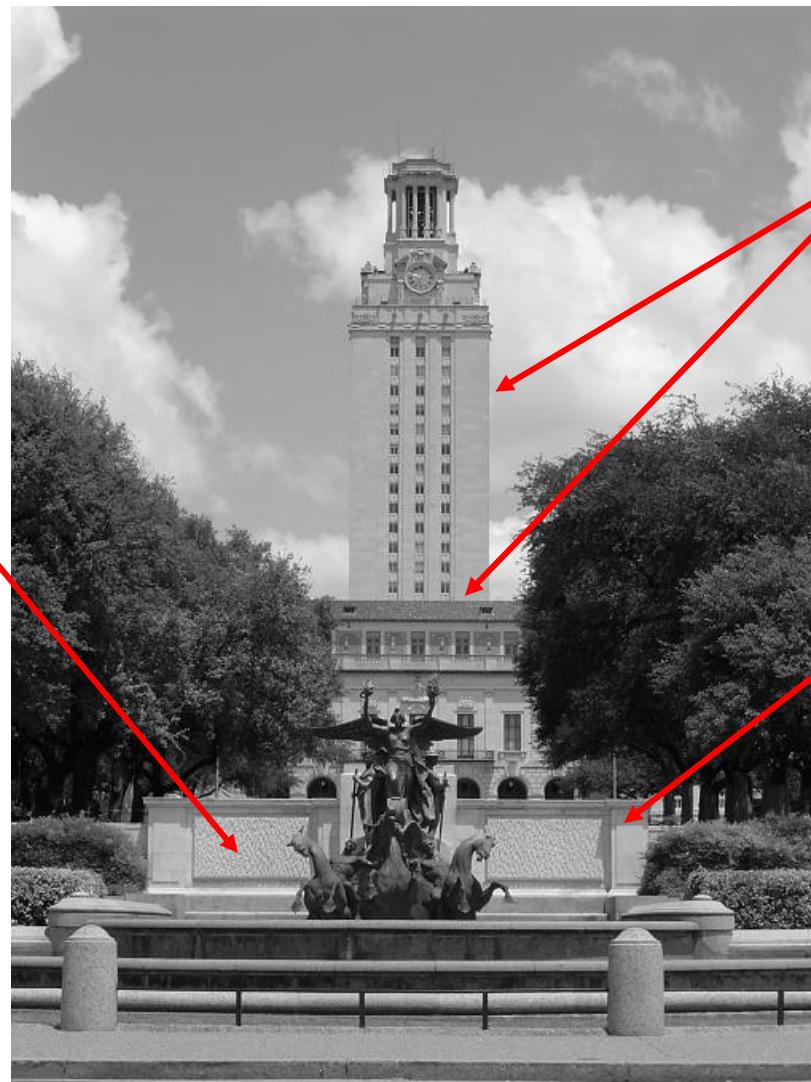


(i) Canny: $\sigma = 8$



What causes an edge?

Reflectance change:
appearance
information, texture

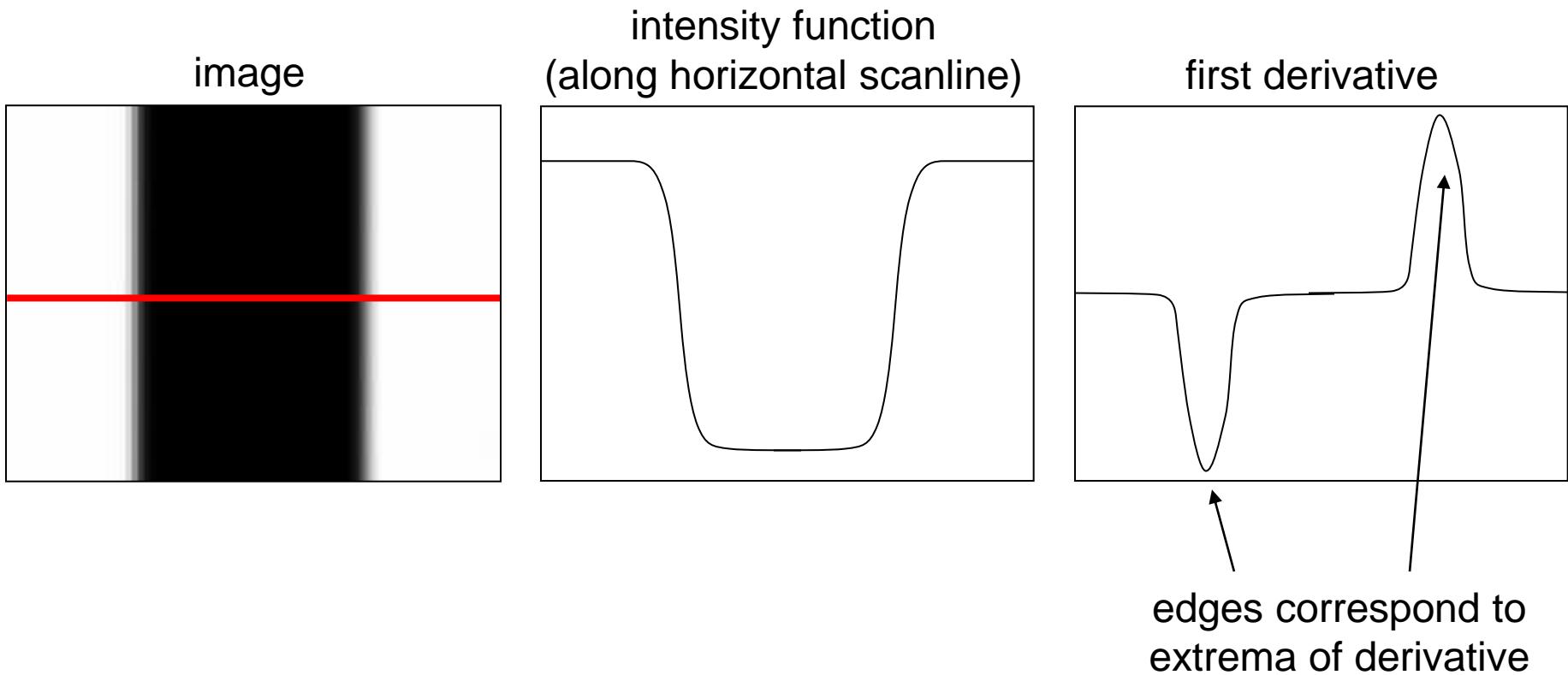


Depth discontinuity:
object boundary

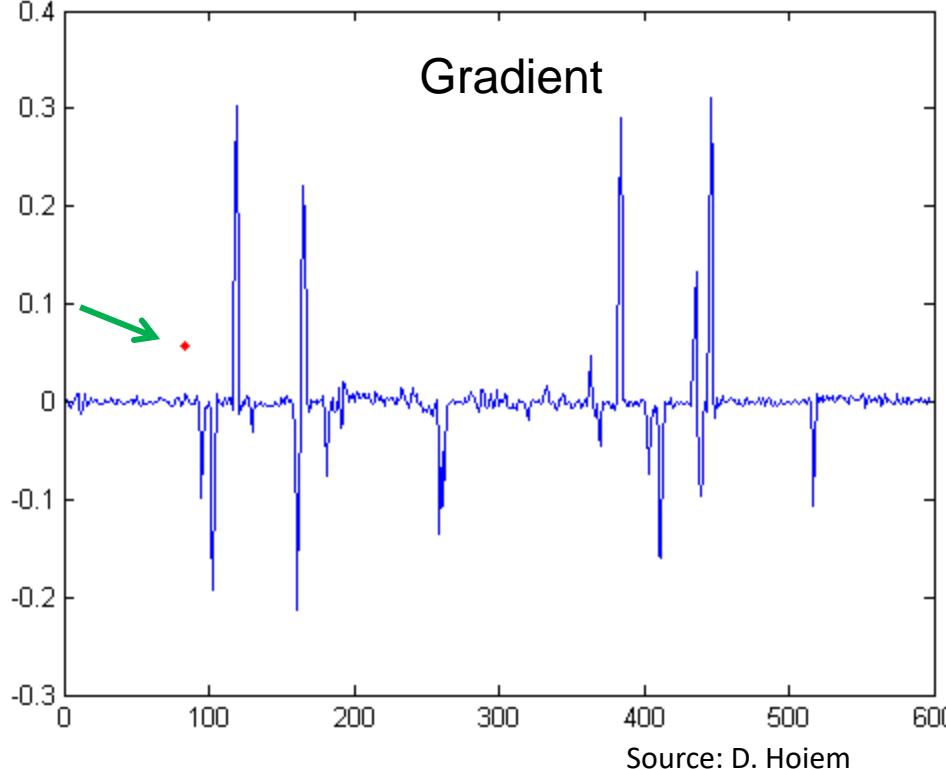
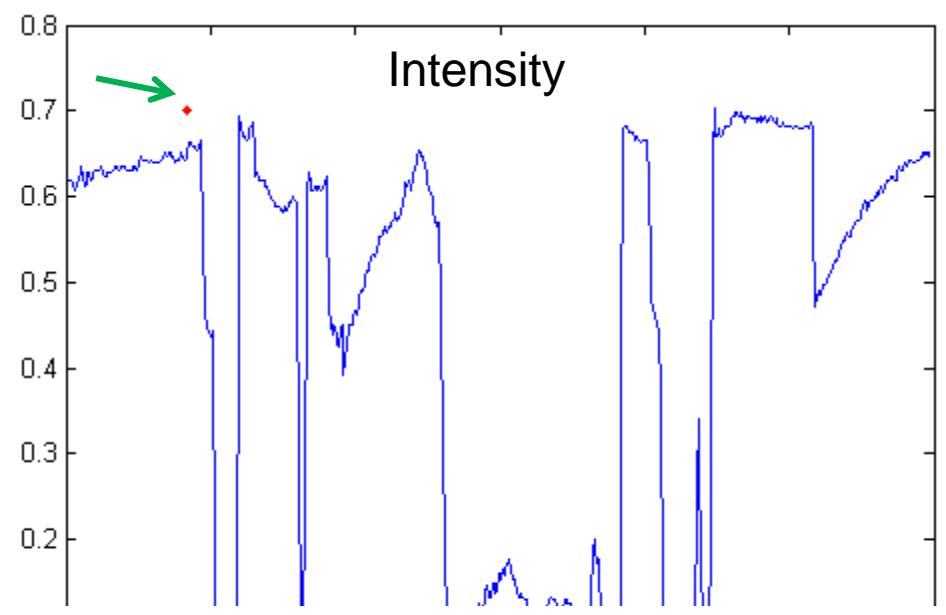
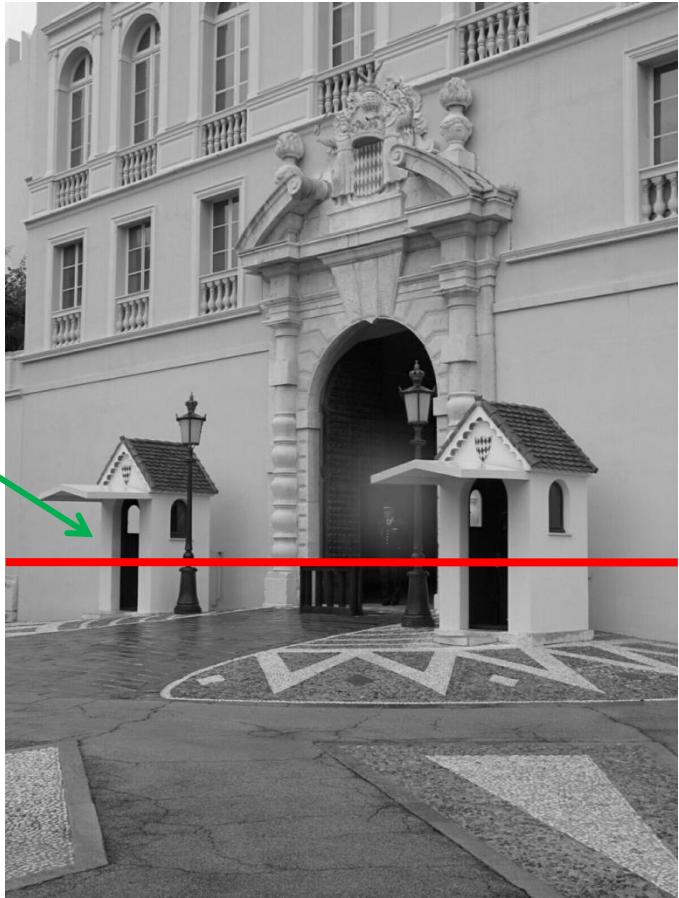
Cast shadows

Characterizing edges

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

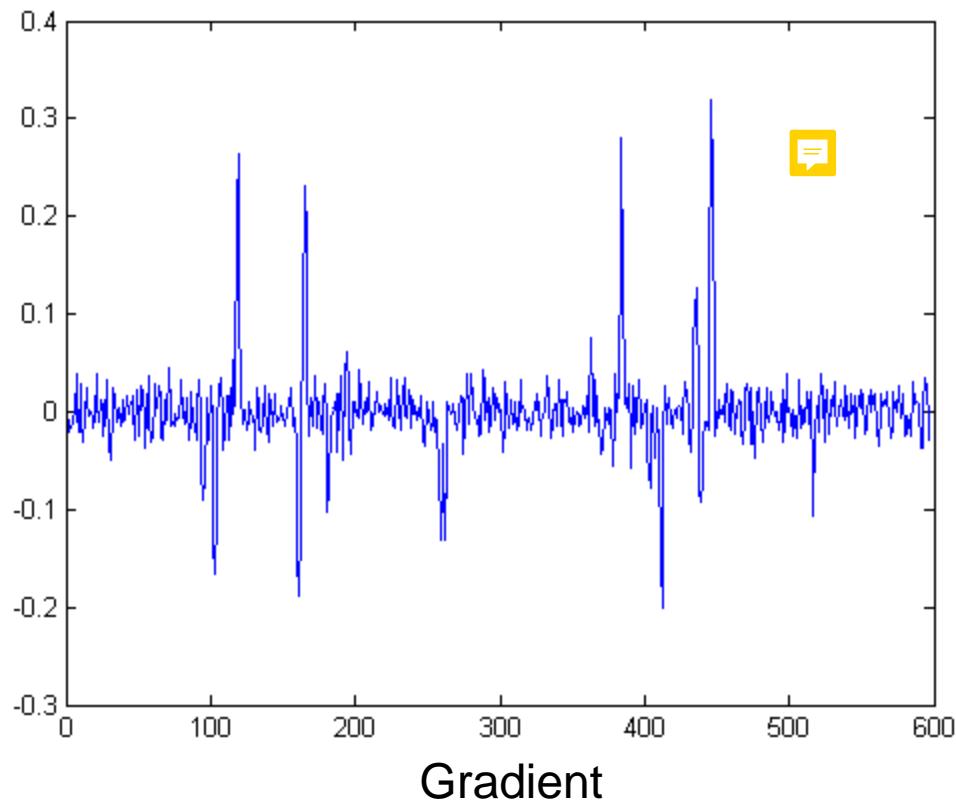
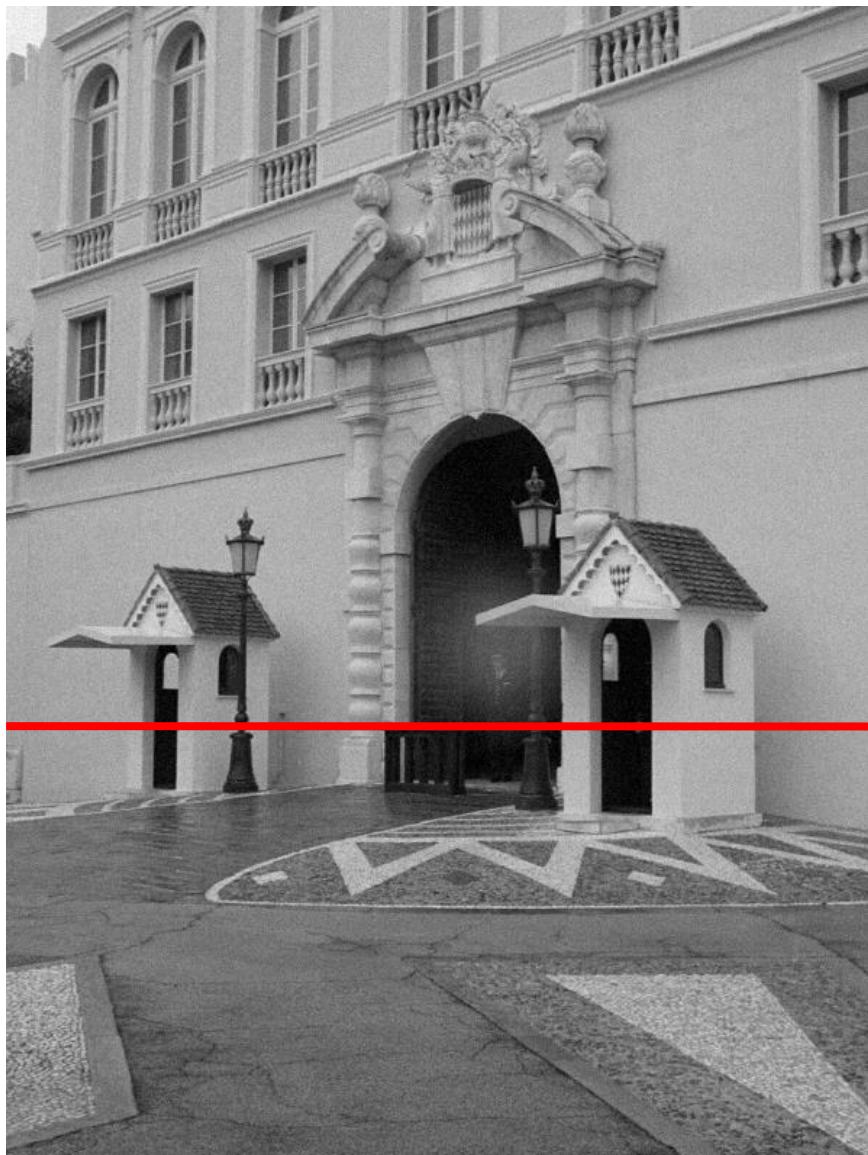


Intensity profile



Source: D. Hoiem

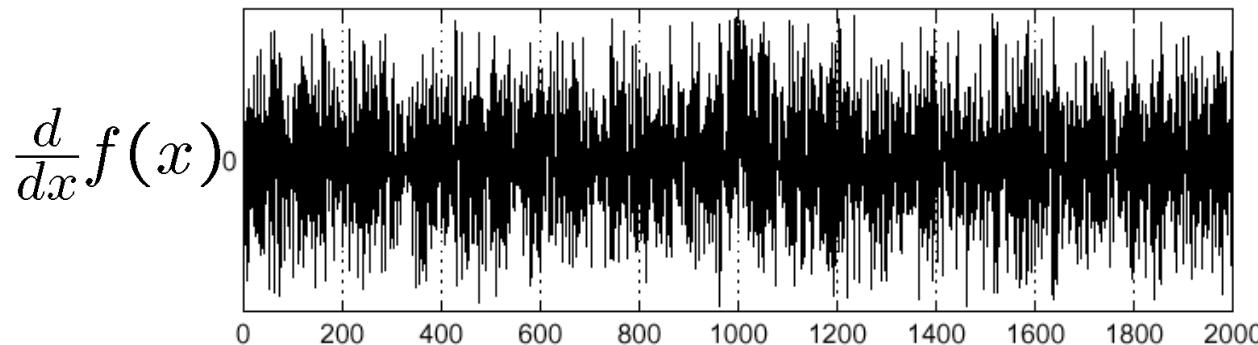
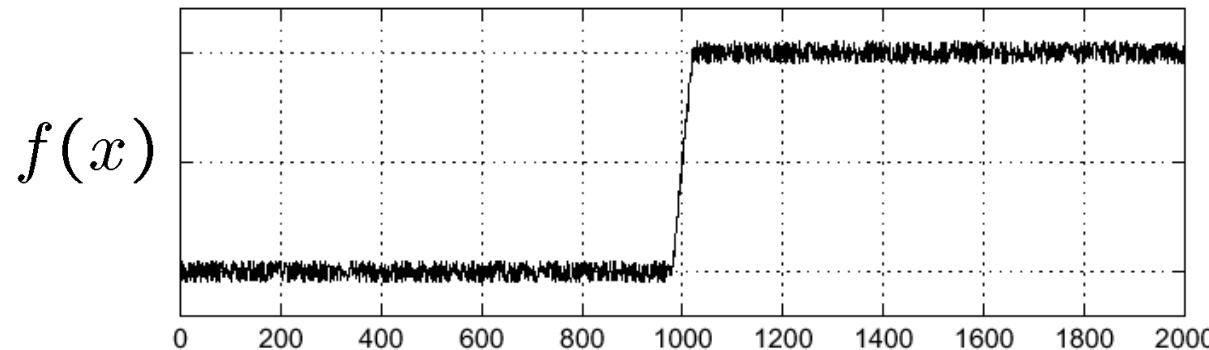
With a little Gaussian noise



Source: D. Hoiem

Effects of noise

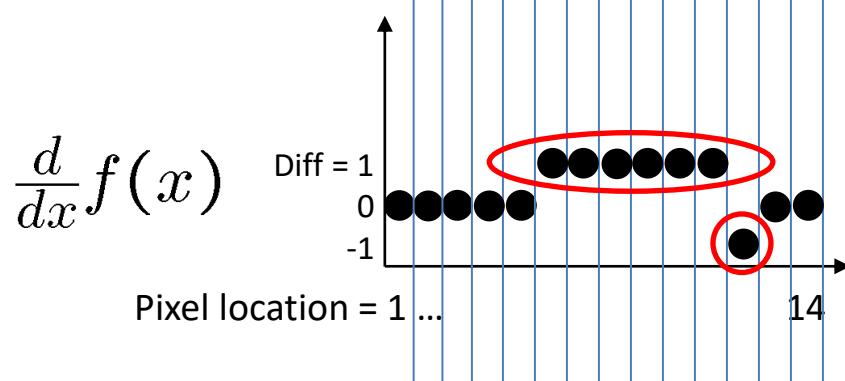
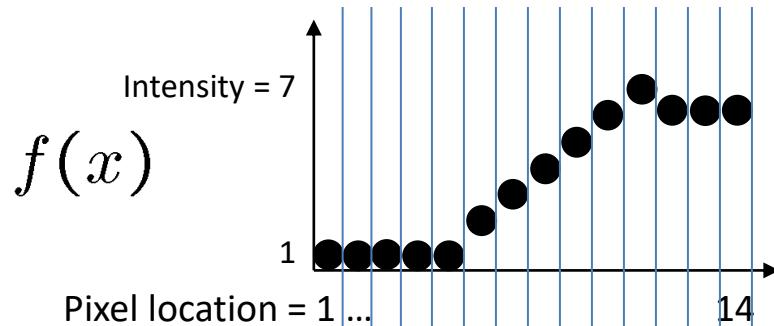
- Consider a single row or column of the image
 - Plotting intensity as a function of position gives a signal



Where is the edge?

Without noise

- Consider a single row or column of the image
 - Plotting intensity as a function of position gives a signal



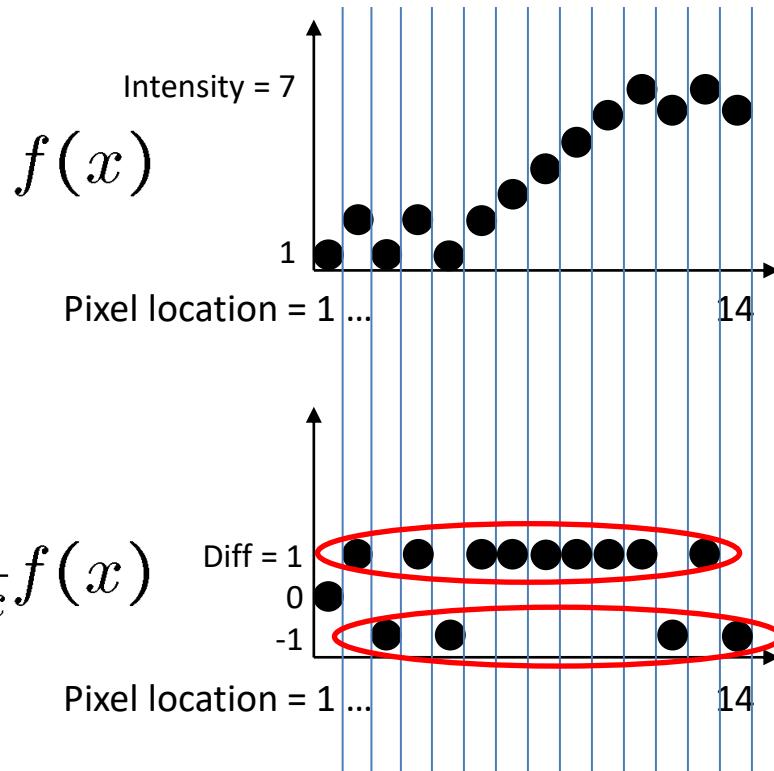
$$\begin{aligned}\frac{d}{dx}f(x) &= \frac{\Delta f(a)}{\Delta a} = \frac{f(a+h) - f(a)}{(a+h) - (a)} = \frac{f(a+h) - f(a)}{h} \\ &= f(x+1) - f(x)\end{aligned}$$

Where is the edge?



With noise

- Consider a single row or column of the image
 - Plotting intensity as a function of position gives a signal



$$\begin{aligned} \frac{d}{dx} f(x) &= \frac{\Delta f(a)}{\Delta a} = \frac{f(a+h) - f(a)}{(a+h) - (a)} = \frac{f(a+h) - f(a)}{h} \\ &= f(x+1) - f(x) \end{aligned}$$

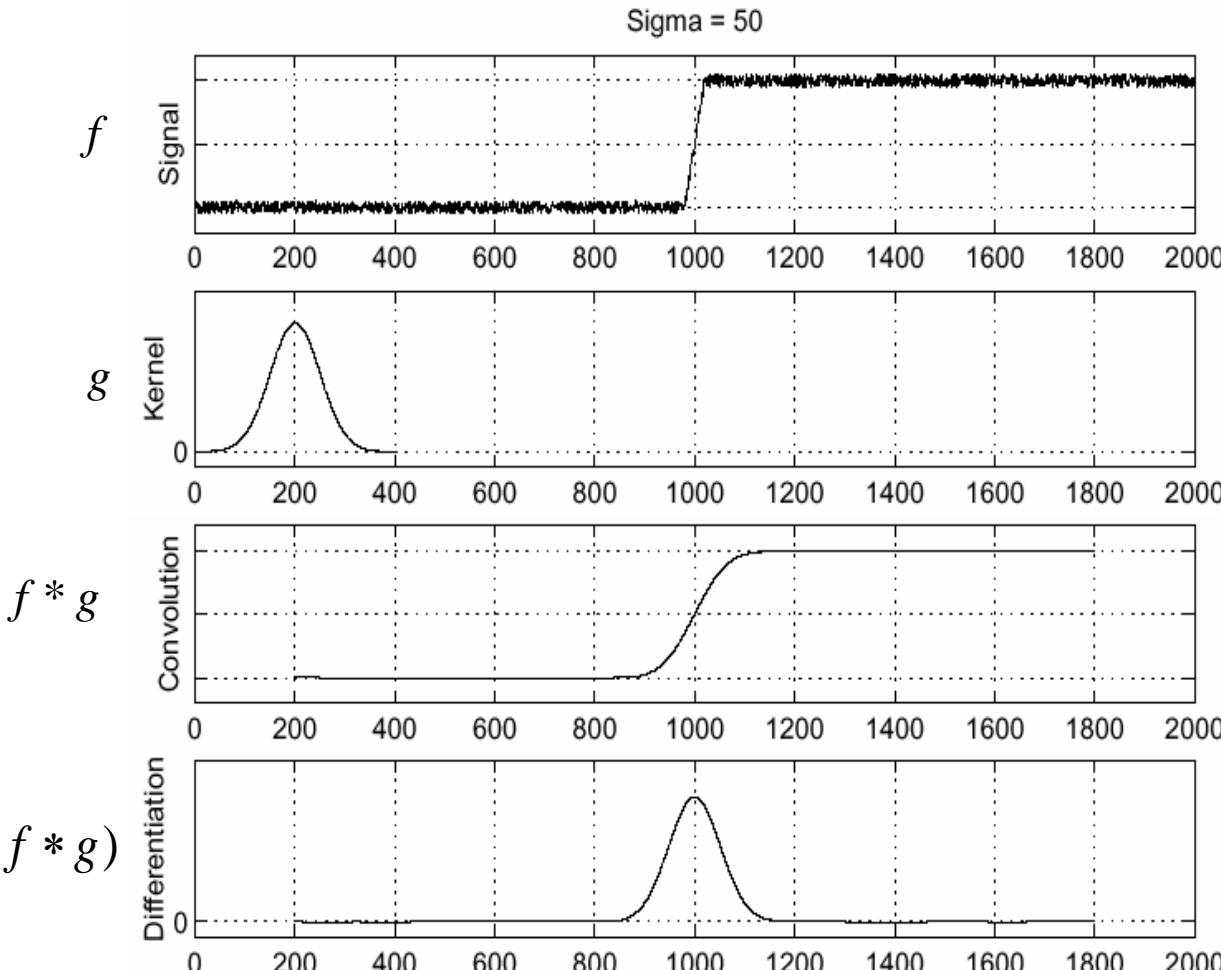
Where is the edge?



Effects of noise

- Difference filters respond strongly to noise
 - Image noise results in pixels that look very different from their neighbors
 - Generally, the larger the noise the stronger the response
- What can we do about it?

Solution: smooth first



- To find edges, look for peaks in $\frac{d}{dx}(f * g)$

Derivative theorem of convolution

- Differentiation is convolution, and convolution is associative:

$$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$$



- This saves us one operation:

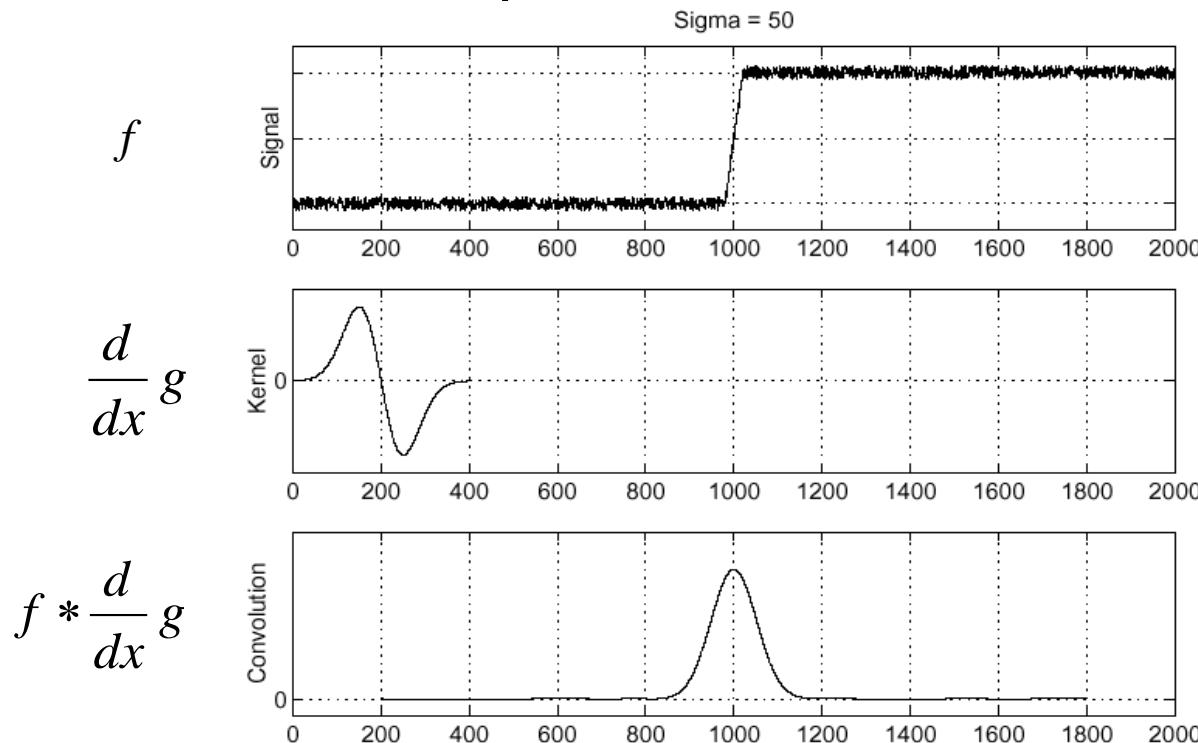


Image
with edge

Derivative
of Gaussian

Edge = max
of derivative

Canny edge detector

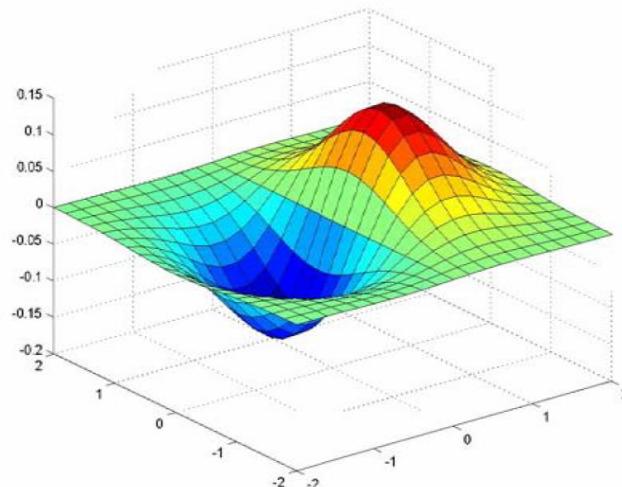
- Filter image with derivative of Gaussian
- Find magnitude and orientation of gradient
- Threshold: Determine which local maxima from filter output are actually edges
- **Non-maximum suppression:**
 - Thin wide “ridges” down to single pixel width 
- **Linking and thresholding (hysteresis):**
 - Define two thresholds: low and high 
 - Use the high threshold to start edge curves and the low threshold to continue them

Example

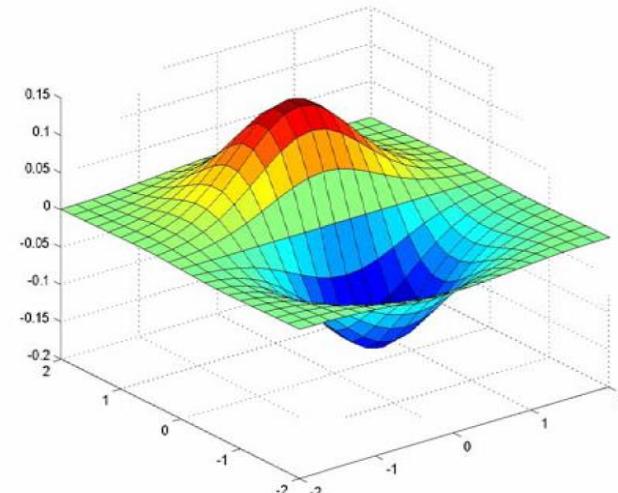


input image (“Lena”)

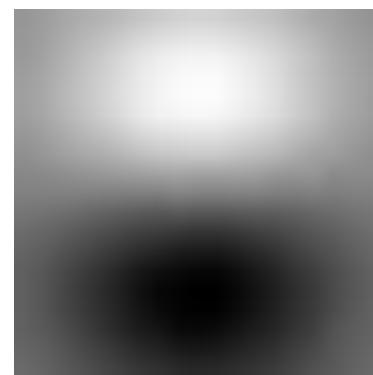
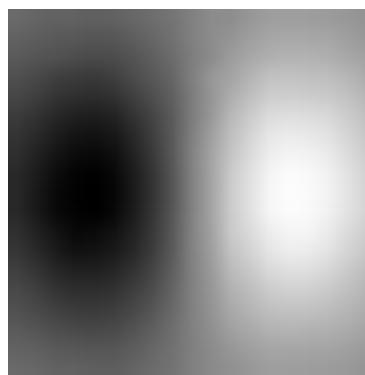
Derivative of Gaussian filter



x-direction



y-direction



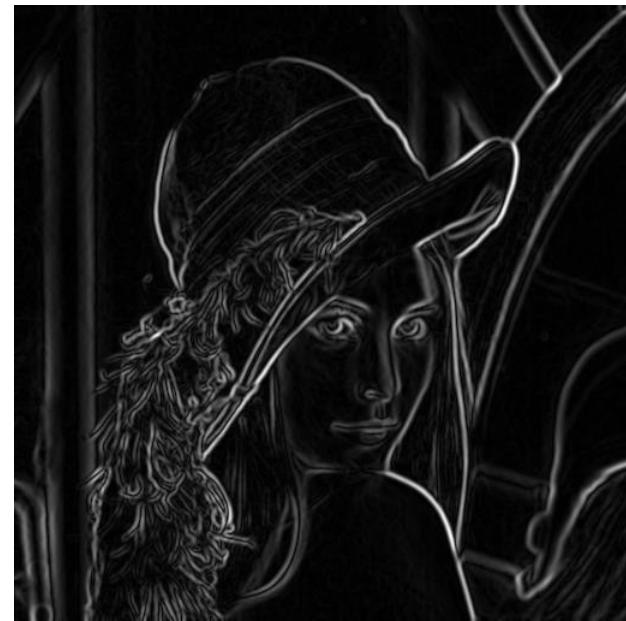
Compute Gradients



X-Derivative of Gaussian



Y-Derivative of Gaussian



Gradient Magnitude



Thresholding

- Choose a threshold value t
- Set any pixels less than t to 0 (off)
- Set any pixels greater than or equal to t to 1 (on)

The Canny edge detector



norm of the gradient (magnitude)

The Canny edge detector



thresholding

Another example: Gradient magnitudes



Thresholding gradient with a lower threshold



Source: K. Grauman

Thresholding gradient with a higher threshold



Source: K. Grauman

Effect of σ of Gaussian kernel



original



Canny with $\sigma = 1$



Canny with $\sigma = 2$

The choice of σ depends on desired behavior

- large σ detects large scale edges 
- small σ detects fine edges

Plan for this lecture

- Edges
 - Extract gradients and threshold
- Lines and circles
 - Find which edge points are collinear or belong to another shape e.g. circle
 - Automatically detect and ignore outliers
- Segments
 - Find which pixels form a consistent region
 - Clustering (e.g. K-means)

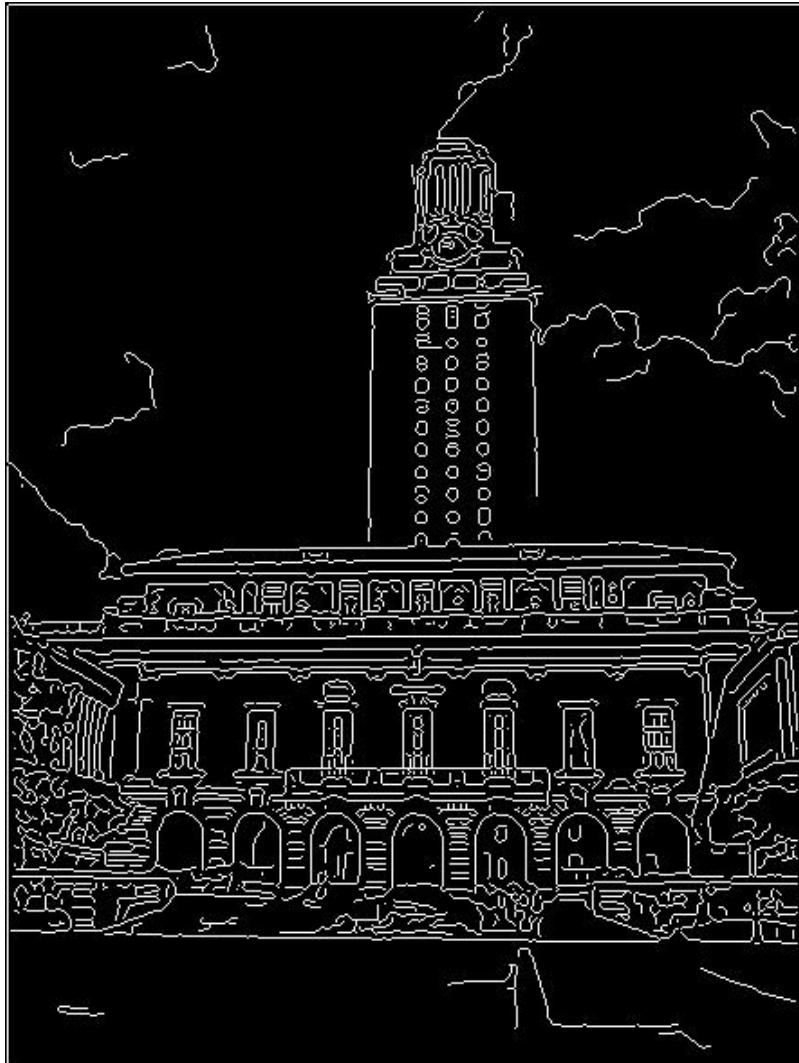
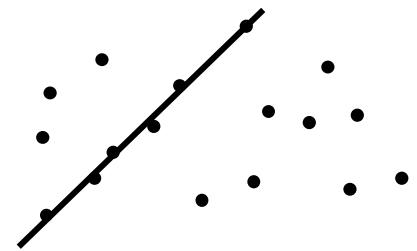
Line detection (fitting)

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



- Why aren't we done just by running edge detection?

Difficulty of line fitting



- **Noise** in measured edge points, orientations:
 - e.g. edges not collinear where they should be
 - how to detect true underlying parameters?
- **Extra** edge points (clutter):
 - 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?



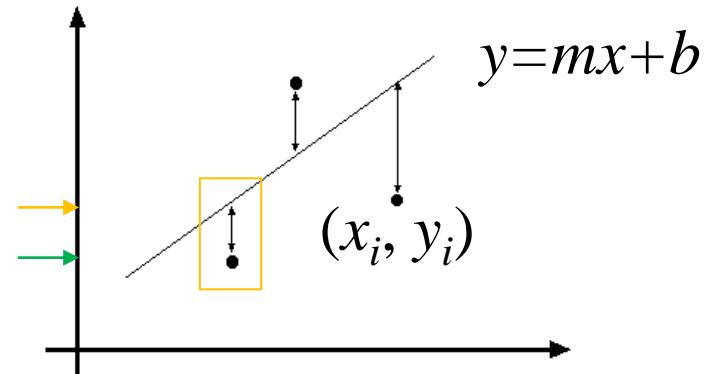
Least squares line fitting

- Data: $(x_1, y_1), \dots, (x_n, y_n)$
- Line equation: $y_i = mx_i + b$
- Find (m, b) to minimize 

$$E = \sum_{i=1}^n (mx_i + b - y_i)^2$$

 where line you found tells
you point is along y axis

where point really is
along y axis

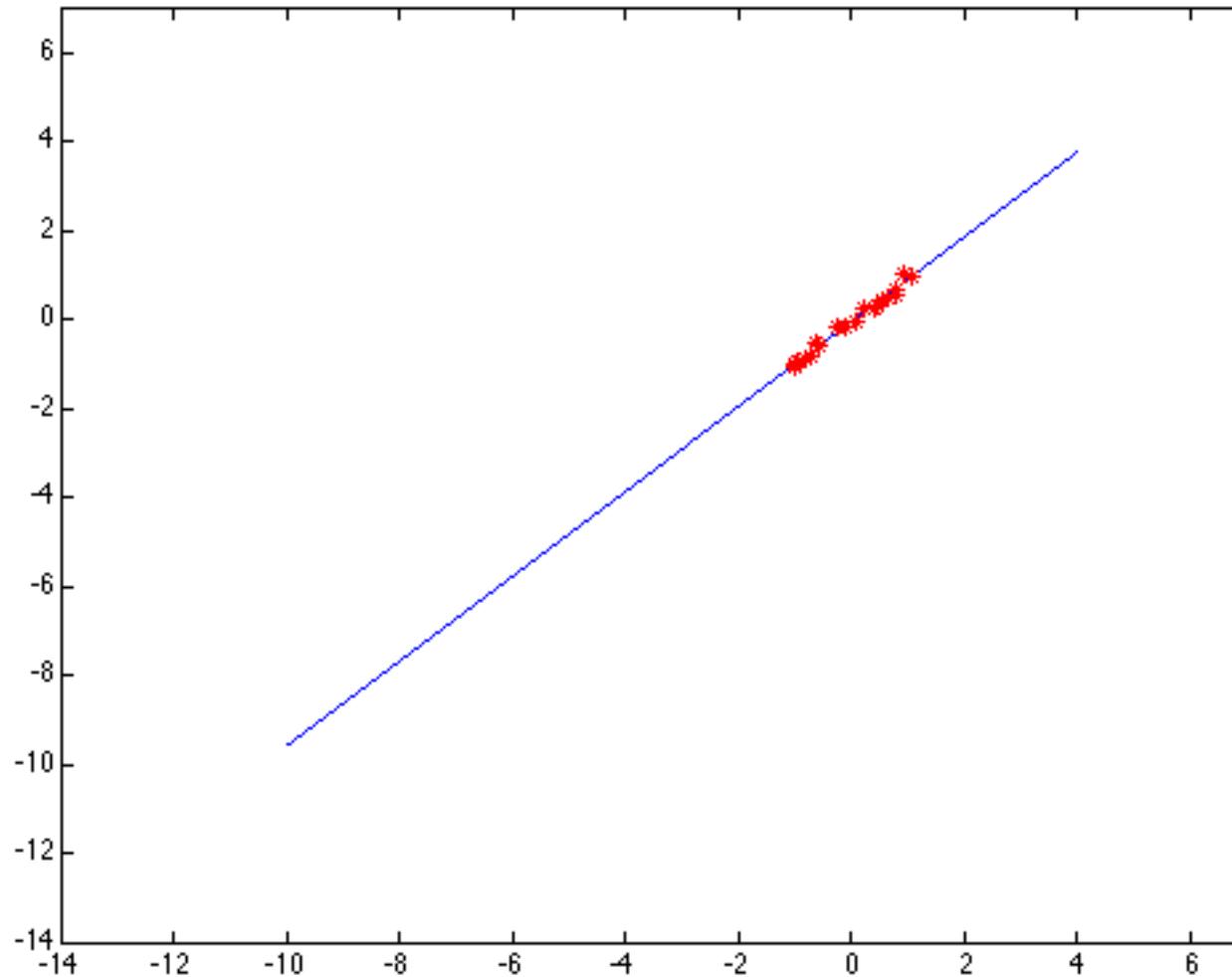


You want to find a single line that
“explains” all of the points in your data,
but data may be noisy!

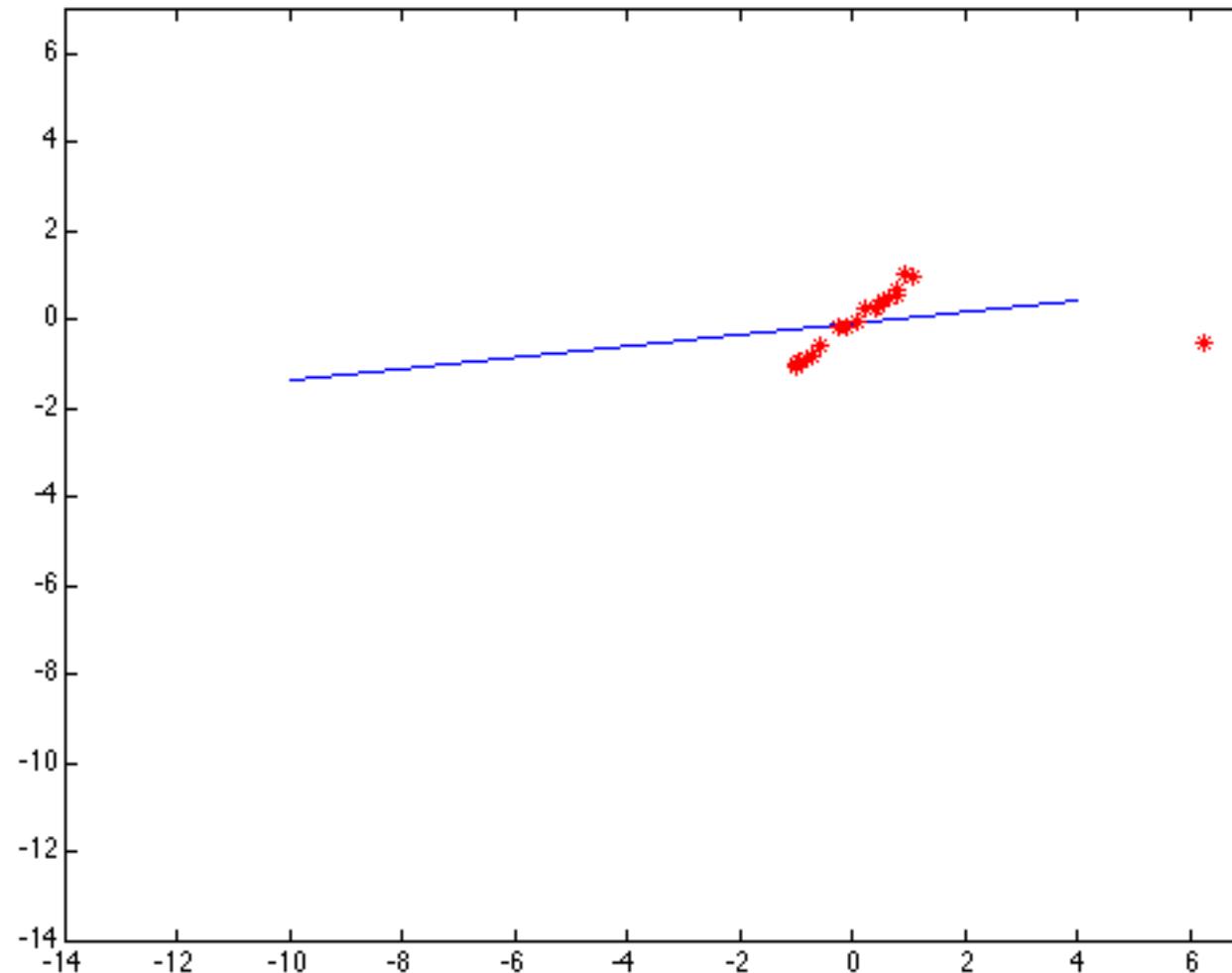
$$E = \sum_{i=1}^n \left(\begin{bmatrix} x_i & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - y_i \right)^2 = \left\| \begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ \vdots & \vdots \\ x_n & 1 \end{bmatrix} \begin{bmatrix} m \\ b \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \right\|^2 = \|\mathbf{Ap} - \mathbf{y}\|^2$$

Matlab: $\mathbf{p} = \mathbf{A} \setminus \mathbf{y}$; or $\mathbf{p} = \text{pinv}(\mathbf{A}) * \mathbf{y}$;

Outliers affect least squares fit



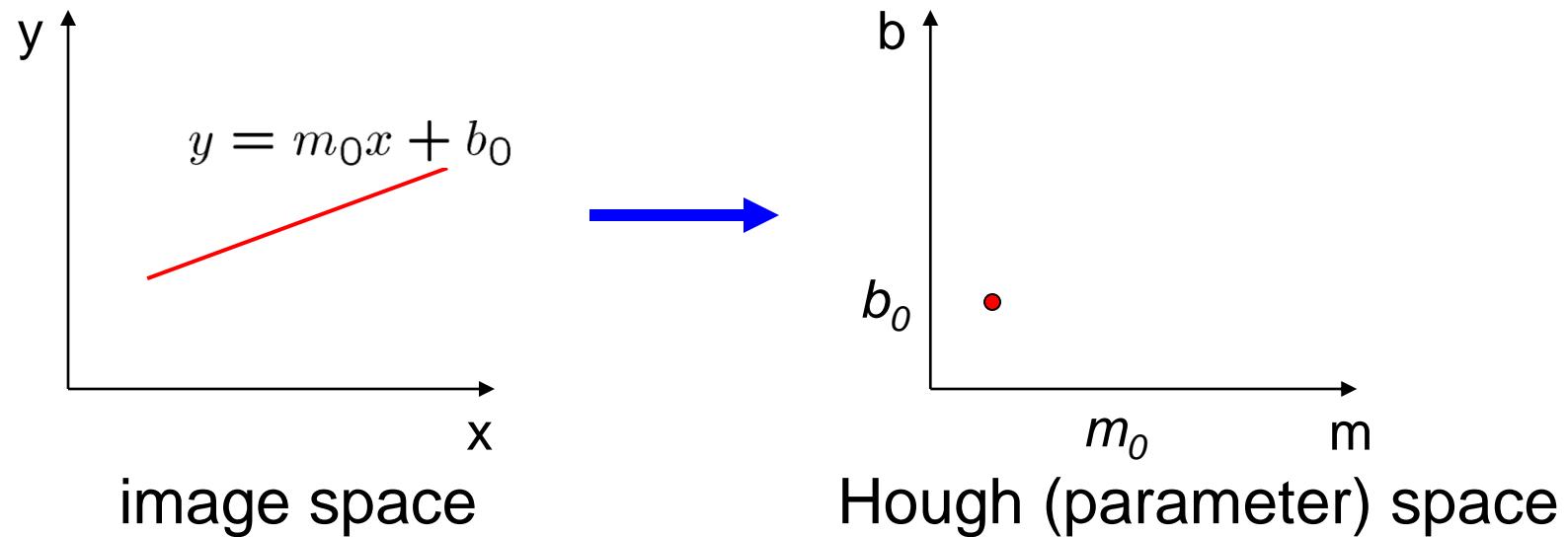
Outliers affect least squares fit



Dealing with outliers: Voting

- **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?
 - They will cast votes too, *but* typically their votes should be inconsistent with the majority of “good” features.
- Common techniques
 - Hough transform
 - RANSAC

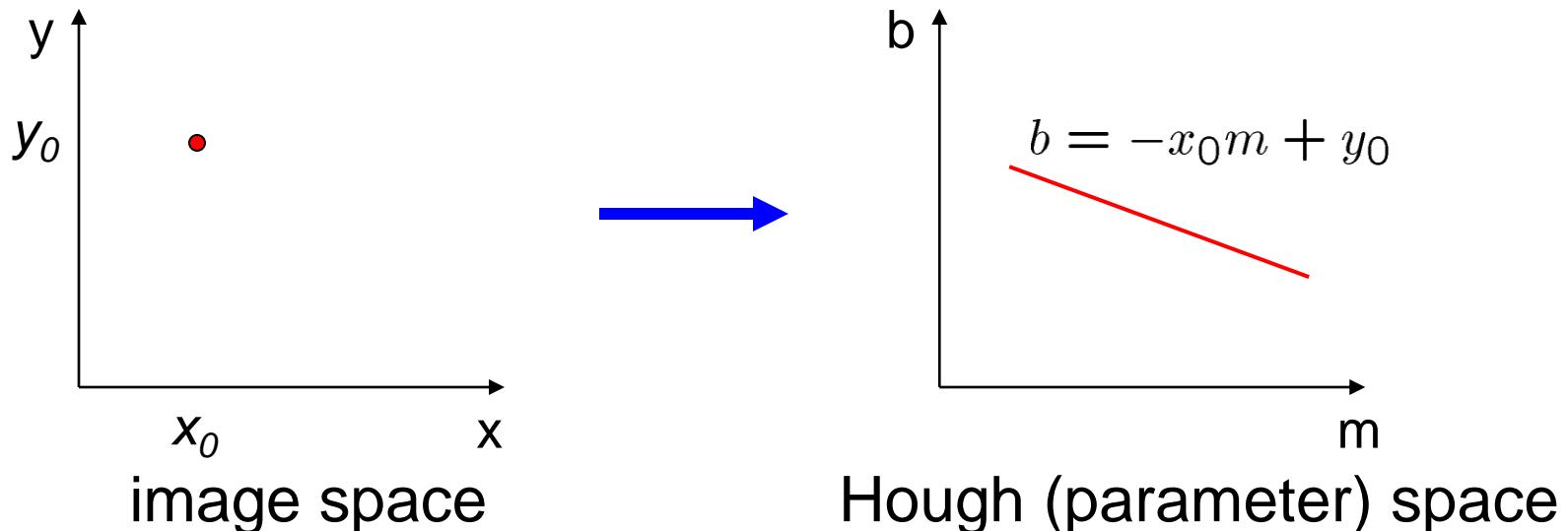
Finding lines in an image: Hough space



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

$y = m_0x + b_0$ • A line in the image corresponds to a point in Hough space

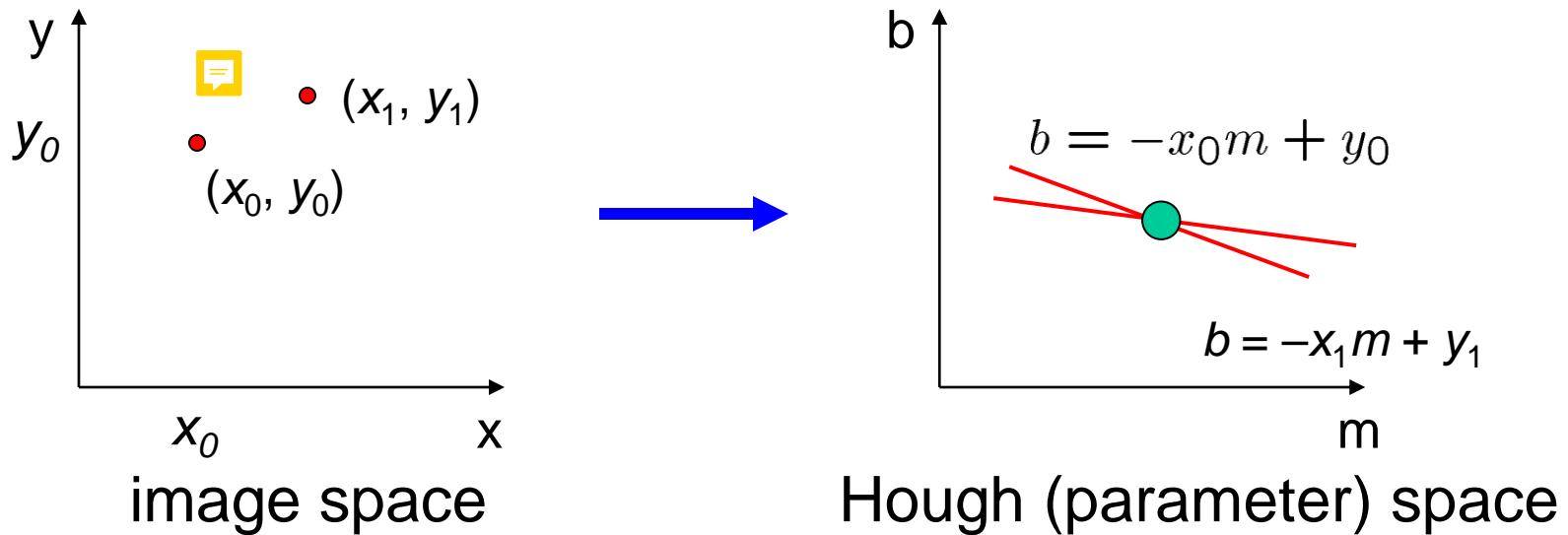
Finding lines in an image: Hough space



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

- $$y = m_0 x + b_0$$
- A line in the image corresponds to a point in Hough space
 - 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
 - Given a pair of points (x,y) , find all (m,b) such that $y = mx + b$

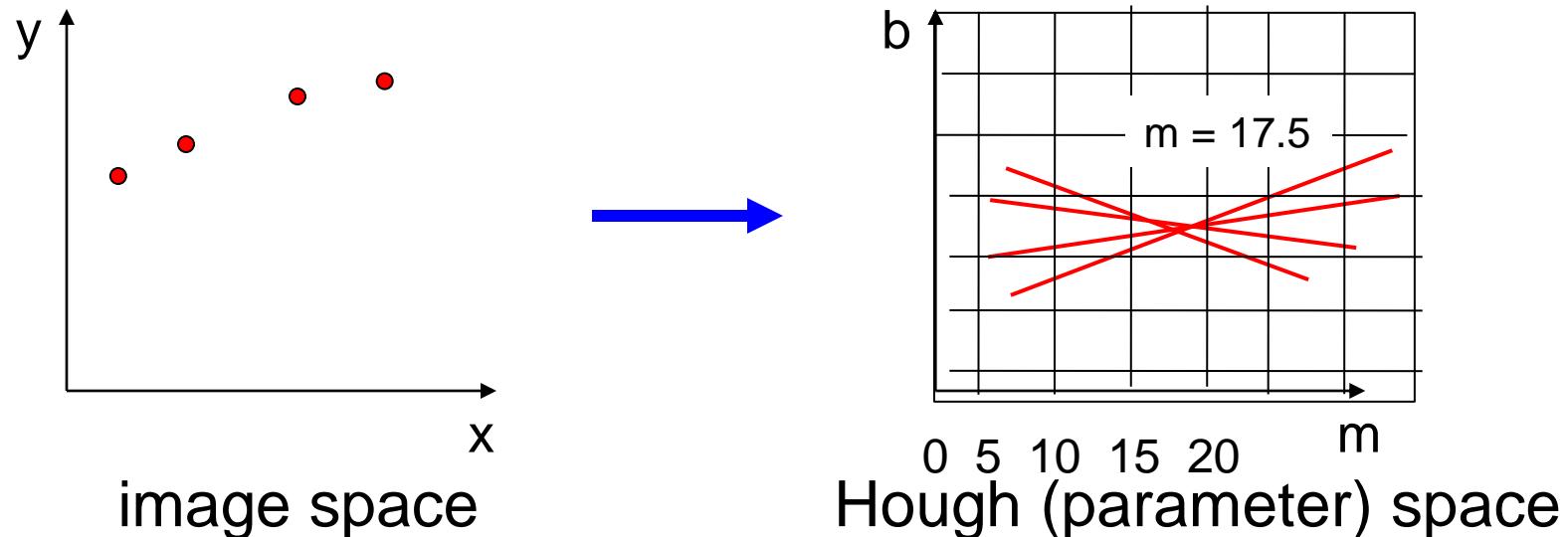
Finding lines in an image: Hough space



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 $b = -x_0m + y_0$ and $b = -x_1m + y_1$

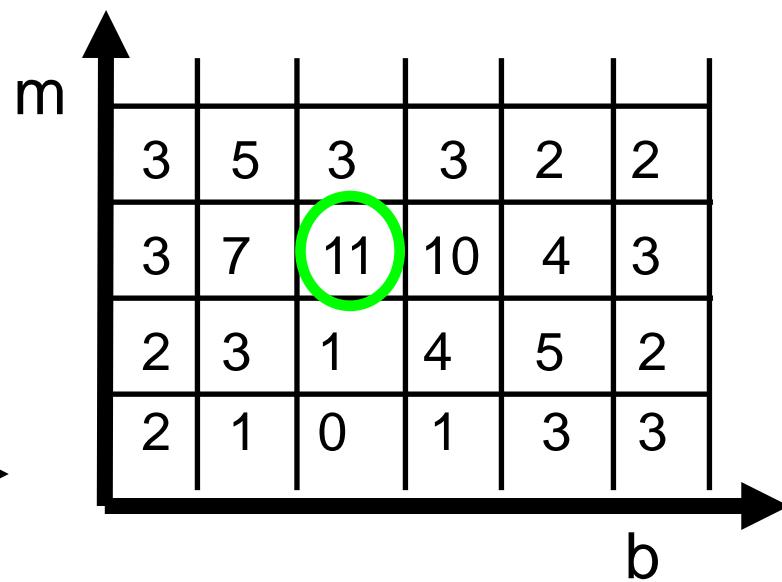
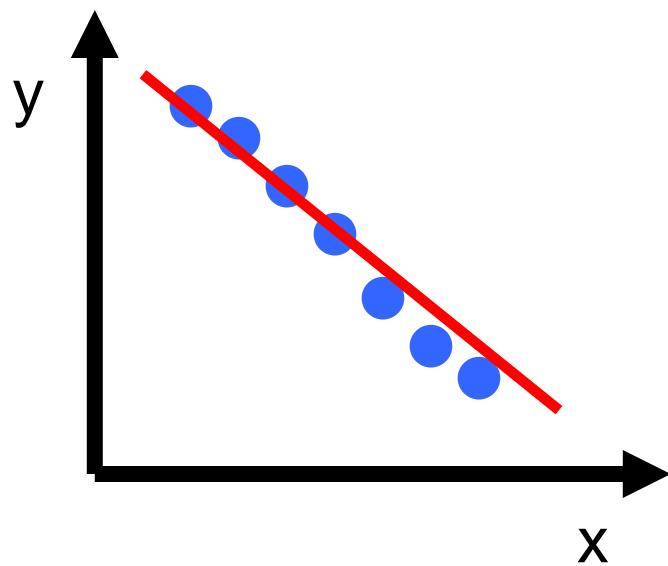
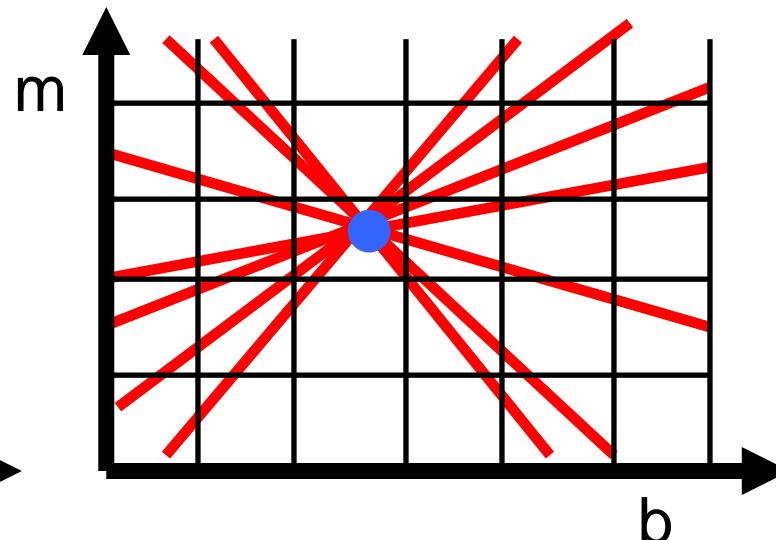
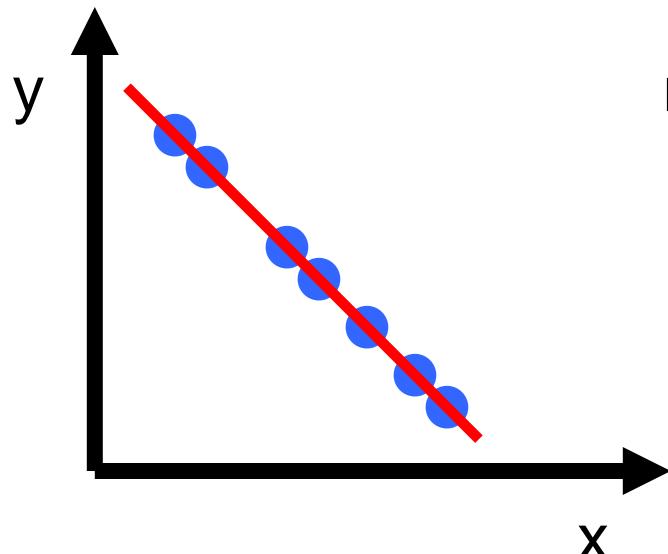
Finding lines in an image: Hough space



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

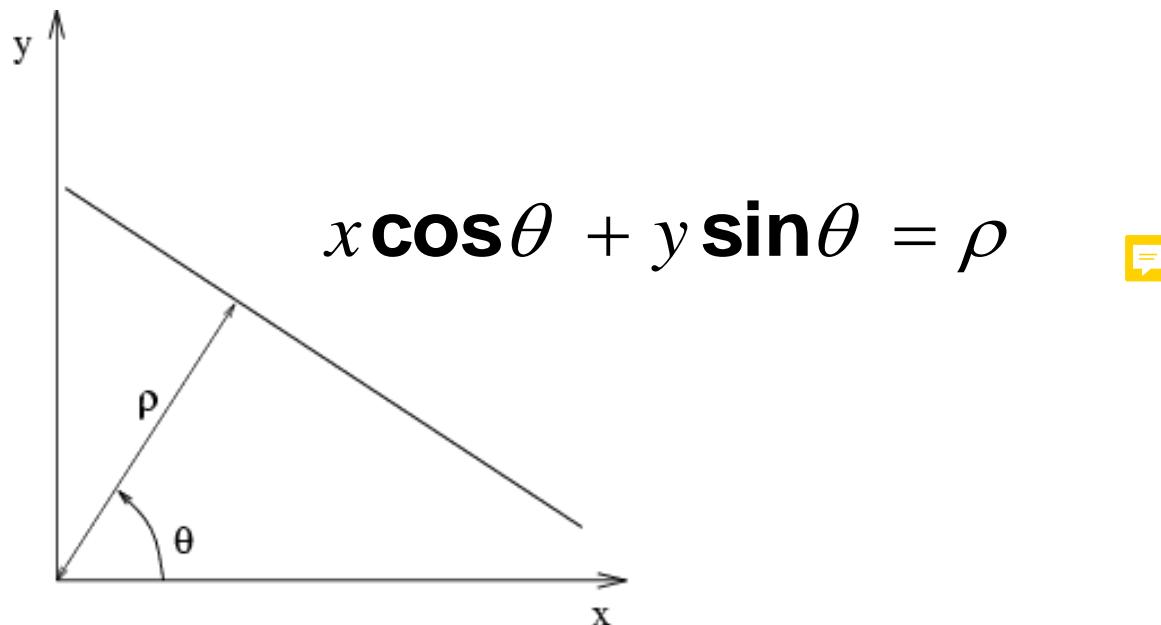
- Let each edge point in image space *vote* for a set of possible parameters in Hough space
- Accumulate votes in discrete set of bins; parameters with the most votes indicate line in image space.

Finding lines in an image: Hough space



Parameter space representation

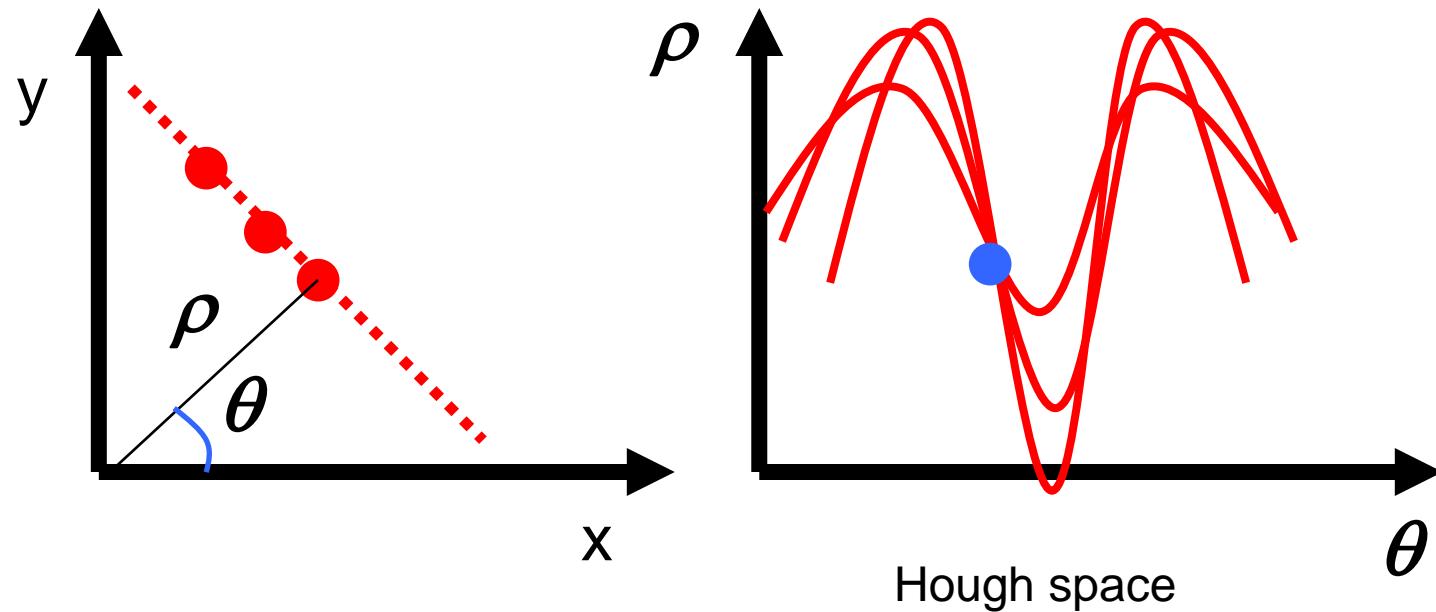
- Problems with the (m, b) space:
 - Unbounded parameter domains
 - Vertical lines require infinite m
- Alternative: *polar representation*



Each point (x,y) will add a sinusoid in the (θ,ρ) parameter space

Parameter space representation

- Problems with the (m,b) space:
 - Unbounded parameter domains
 - Vertical lines require infinite m
- Alternative: *polar representation*



Each point (x,y) will add a sinusoid in the (θ,ρ) parameter space

Algorithm outline: Hough transform

- Initialize accumulator H to all zeros
- For each edge point (x, y) in the image

For $\theta = 0$ to 180

$$\rho = x \cos \theta + y \sin \theta$$

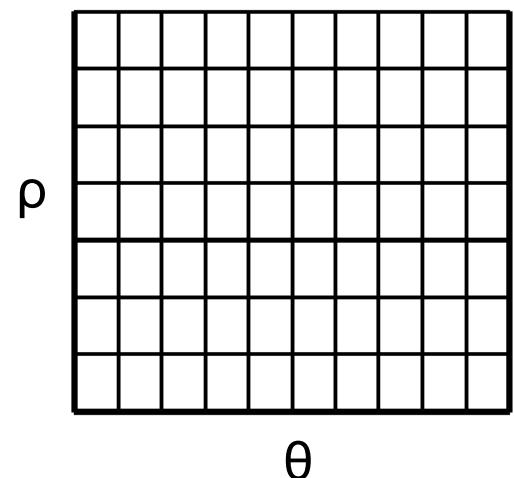
$$H(\theta, \rho) = H(\theta, \rho) + 1$$

end

end

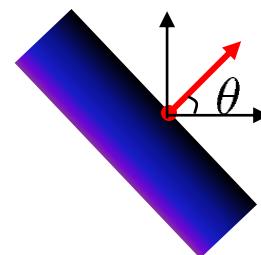
- Find the value(s) of (θ^*, ρ^*) where $H(\theta, \rho)$ is a local maximum
 - The detected line in the image is given by
- $$\rho^* = x \cos \theta^* + y \sin \theta^*$$

H : accumulator array (votes)



Incorporating image gradients

- Recall: when we detect an edge point, we also know its gradient direction
- But this means that the line is uniquely determined!
- Modified Hough transform:



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

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

For each edge point (x, y) in the image

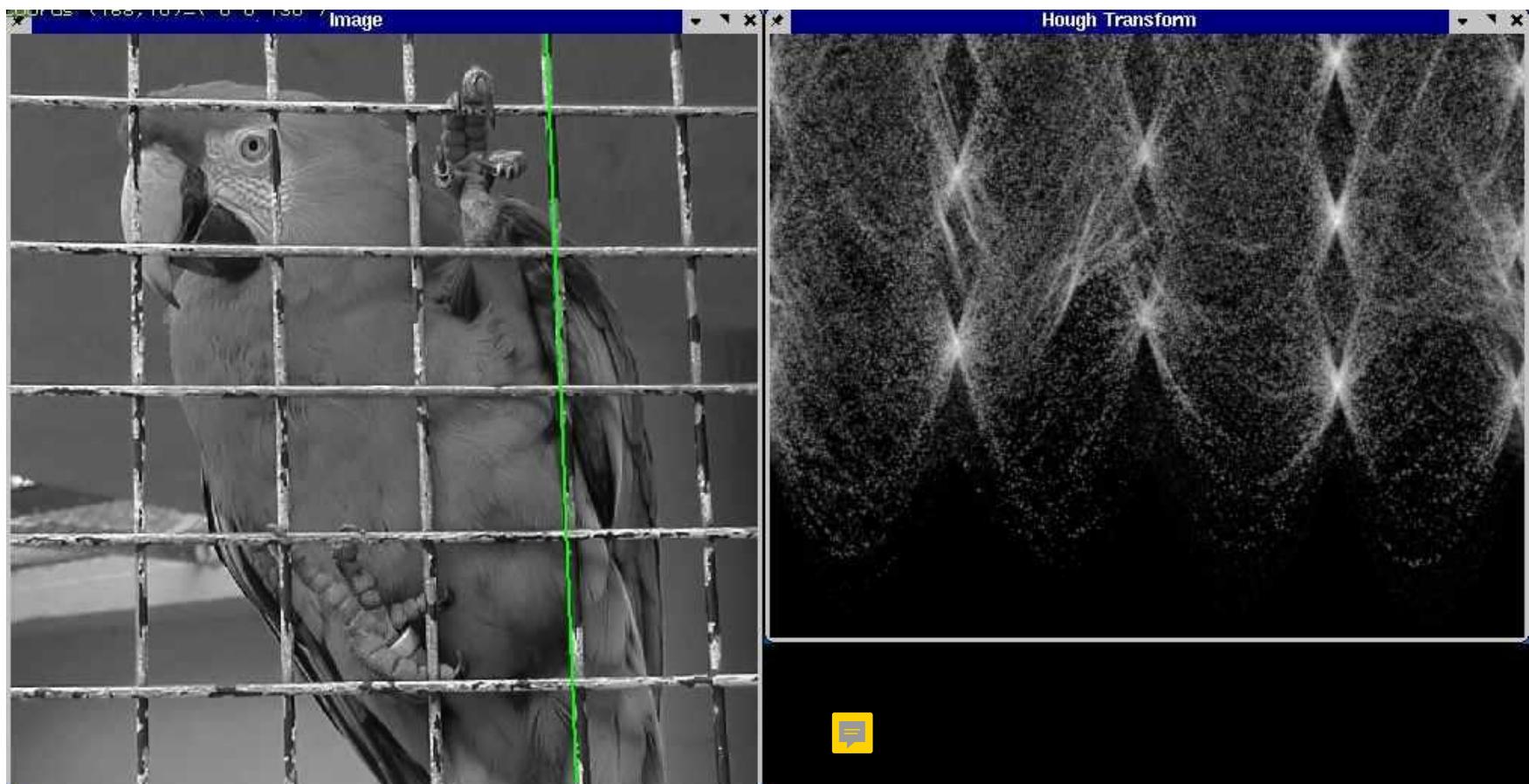
θ = gradient orientation at (x, y)

$\rho = x \cos \theta + y \sin \theta$

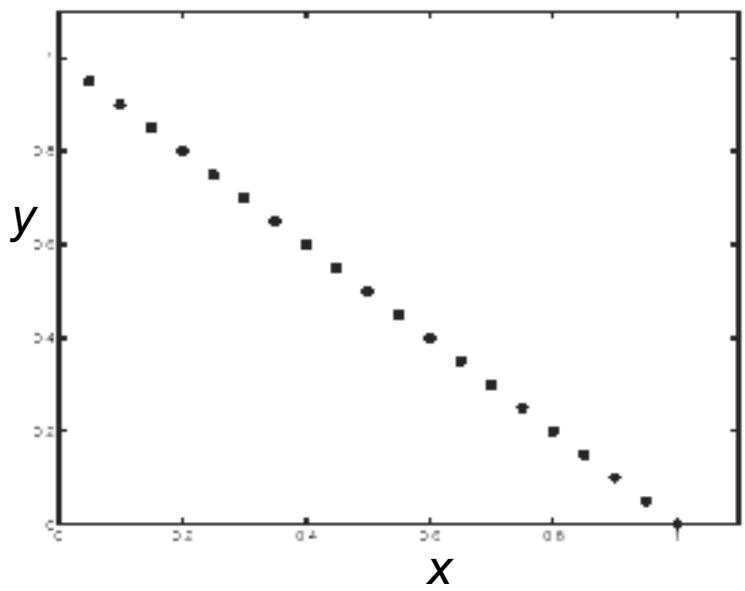
$H(\theta, \rho) = H(\theta, \rho) + 1$

end

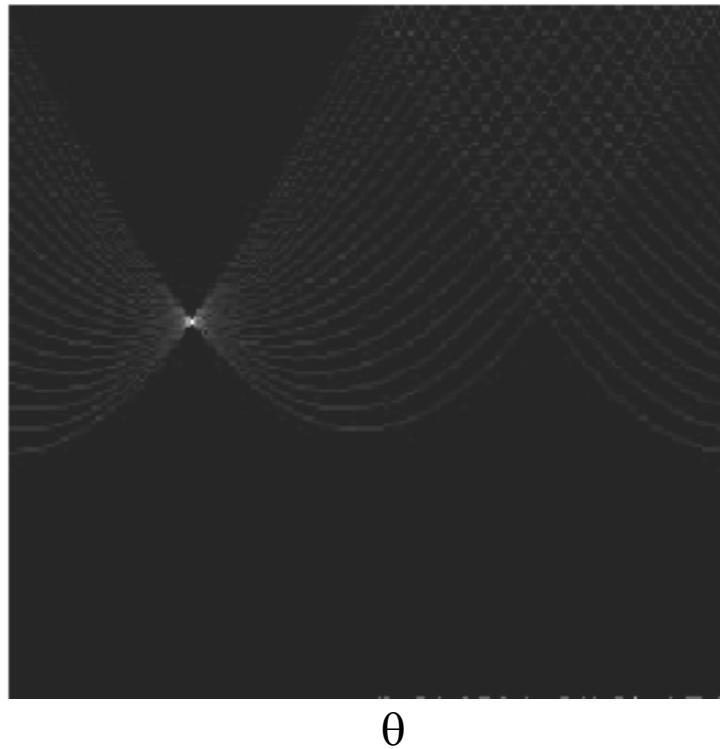
Hough transform example



Impact of noise on Hough

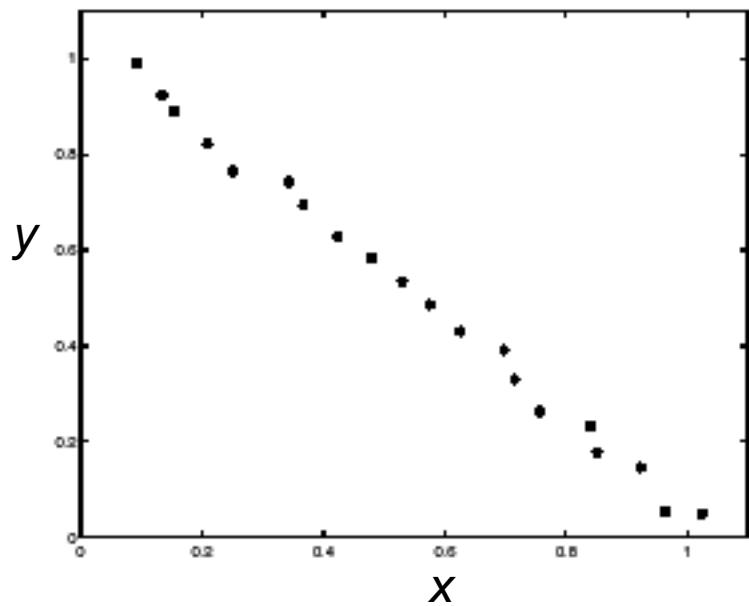


**Image space
edge coordinates**

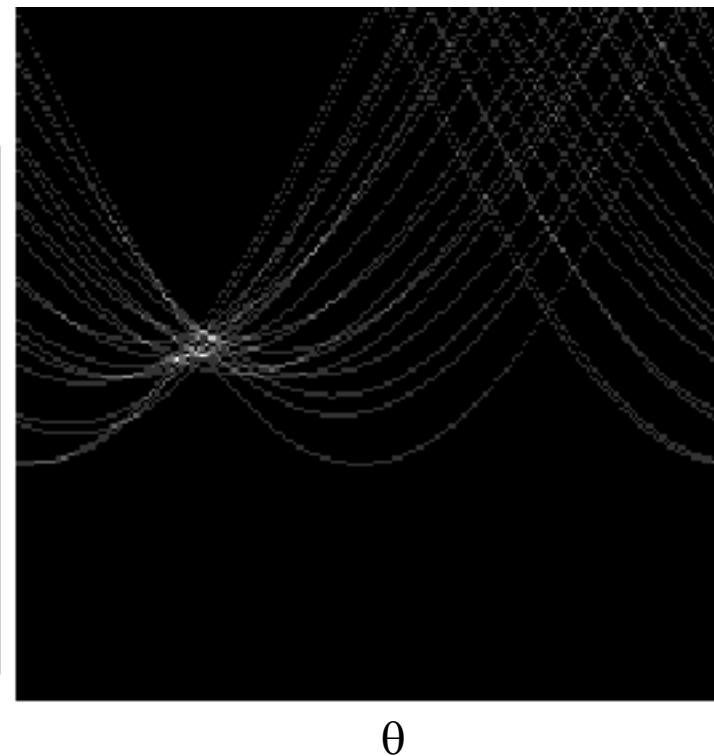


Votes

Impact of noise on Hough



**Image space
edge coordinates**



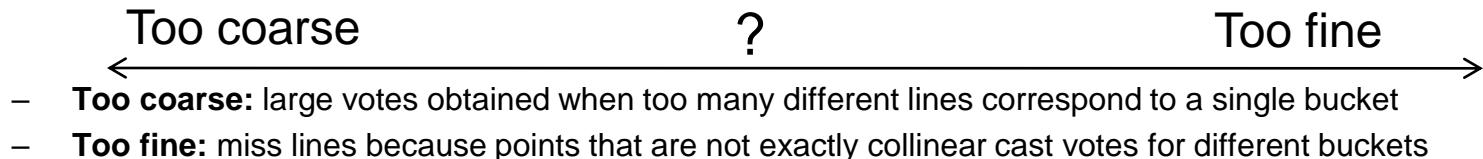
Votes

What difficulty does this present for an implementation?



Voting: practical tips

- Minimize irrelevant tokens first (reduce noise)
- Choose a good grid / discretization



- Vote for neighbors (smoothing in accumulator array)
- Use direction of edge to reduce parameters by 1
- To read back which points voted for “winning” peaks, keep tags on the votes

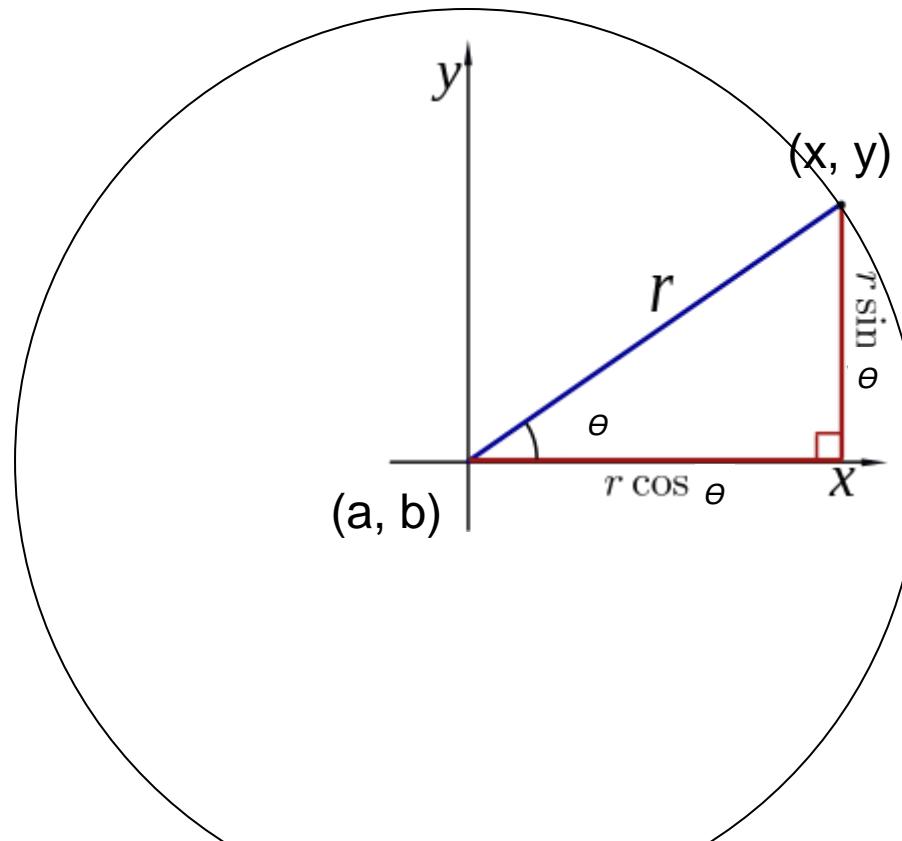


Hough transform for circles

- A circle with radius r and center (a, b) can be described as:

$$x = a + r \cos(\theta)$$

$$y = b + r \sin(\theta)$$

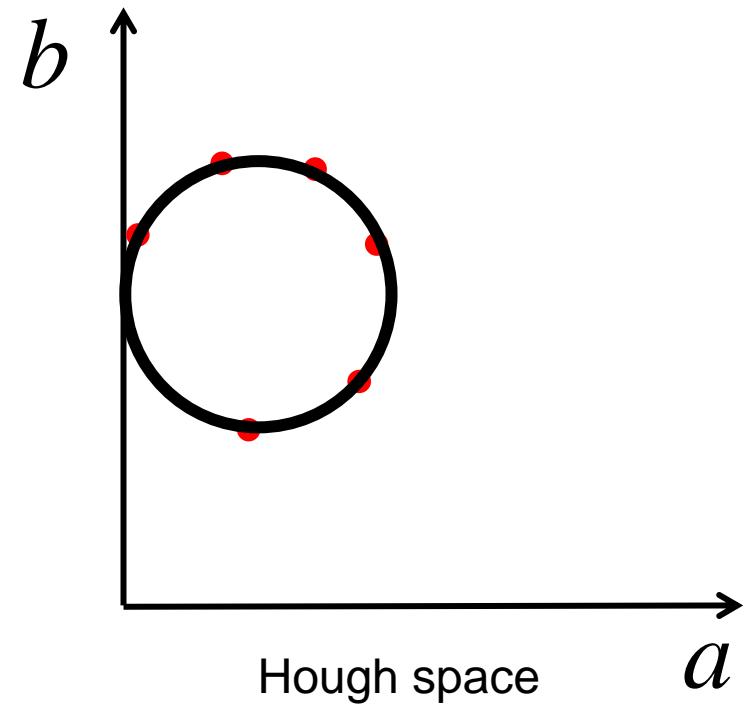
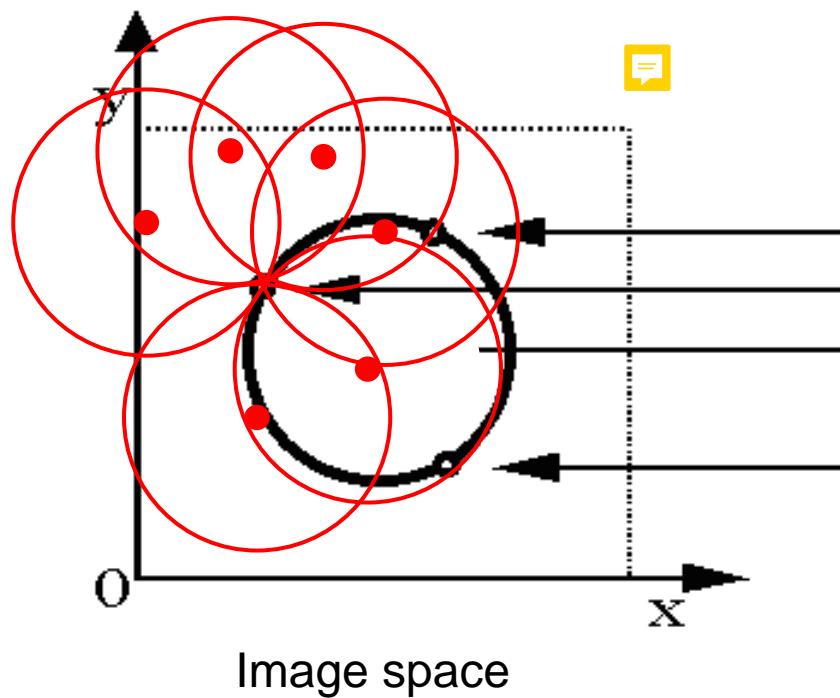


Hough transform for circles

- 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

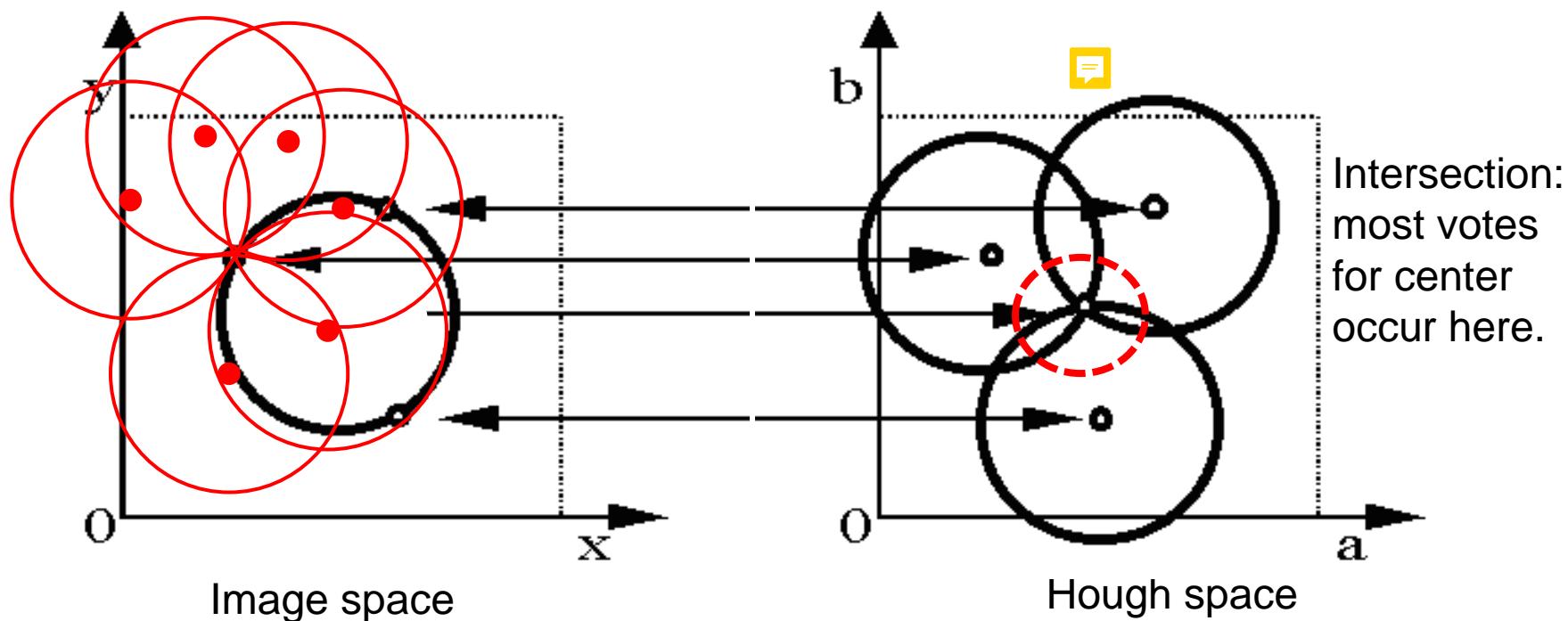


Hough transform for circles

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

$$(x_i - a)^2 + (y_i - b)^2 = r^2 \quad \text{💡}$$

- For a fixed radius r , unknown gradient direction



Hough transform for circles

For every edge pixel (x,y) :

$$x = a + r \cos(\theta)$$
$$y = b + r \sin(\theta)$$

For each possible radius value r :

For each possible gradient direction θ : 

// or use estimated gradient at (x,y) 

$a = x - r \cos(\theta)$ // column

 $b = y - r \sin(\theta)$ // row

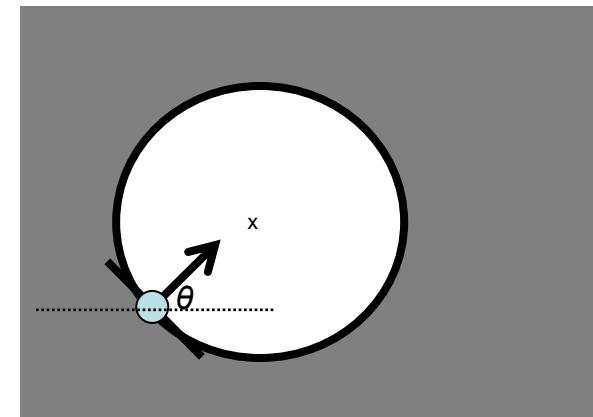
$H[a,b,r] += 1$

end 

end

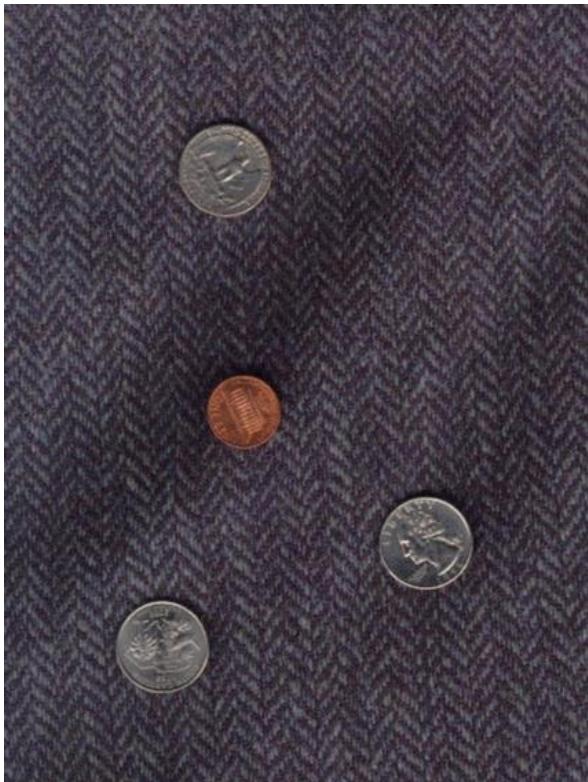
end

Your homework!

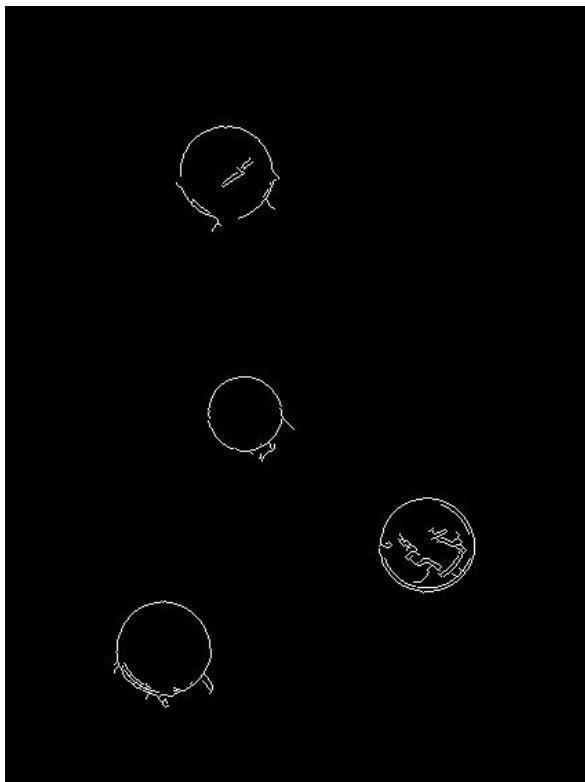


Example: detecting circles with Hough

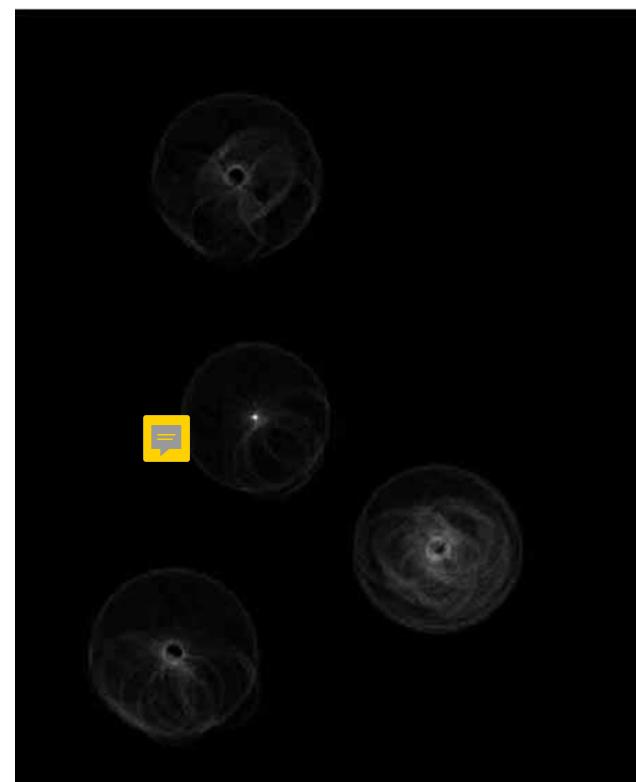
Original



Edges



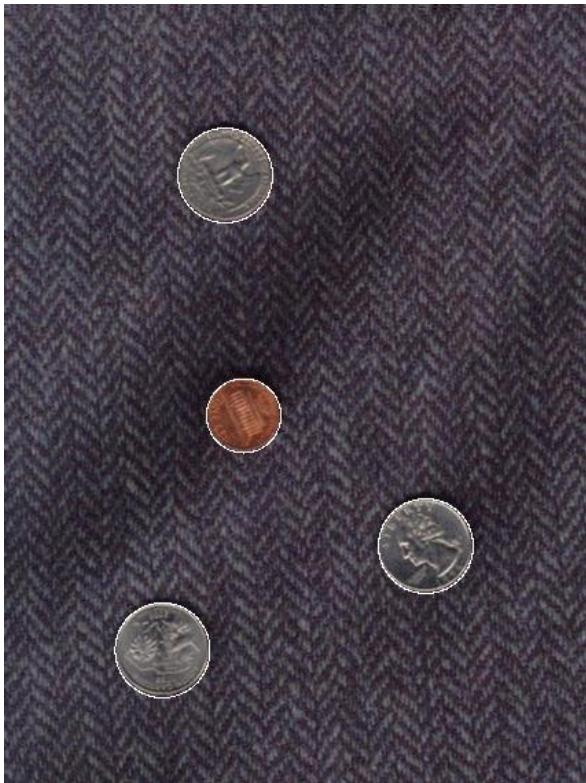
Votes: Penny



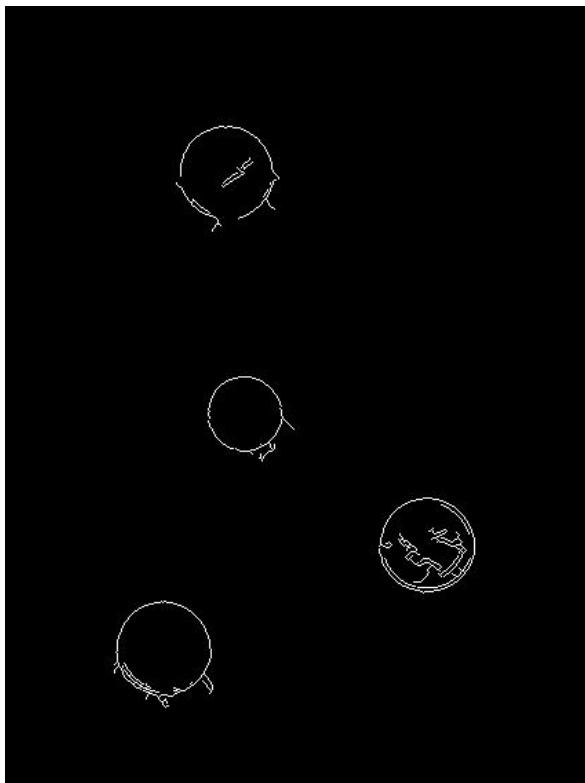
Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

Example: detecting circles with Hough

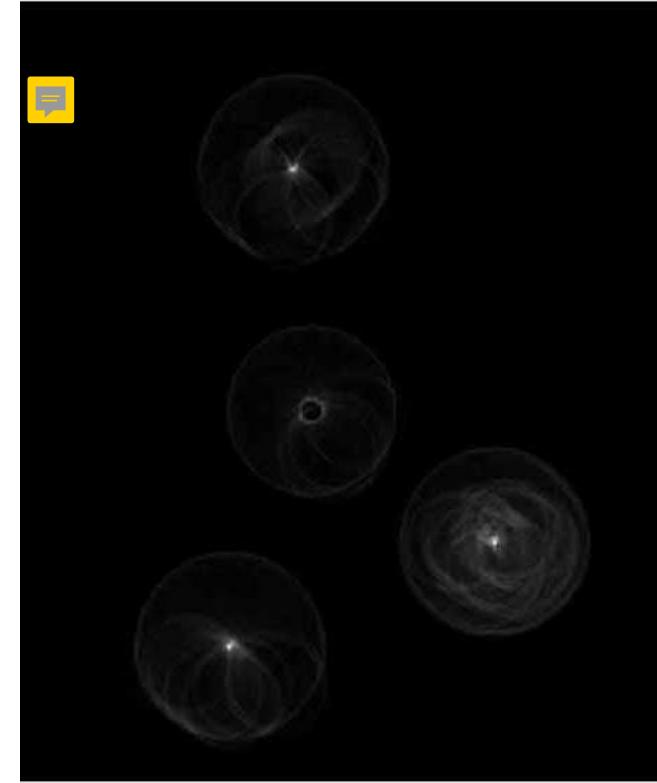
Combined original detections



Edges



Votes: Quarter



Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

Hough transform: pros and cons

Pros

- All points are processed independently, so can cope with occlusion, gaps
- 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 for maxima increases exponentially with the number of model parameters
 - If 3 parameters and 10 choices for each, search is $O(10^3)$
- Quantization: can be tricky to pick a good grid size

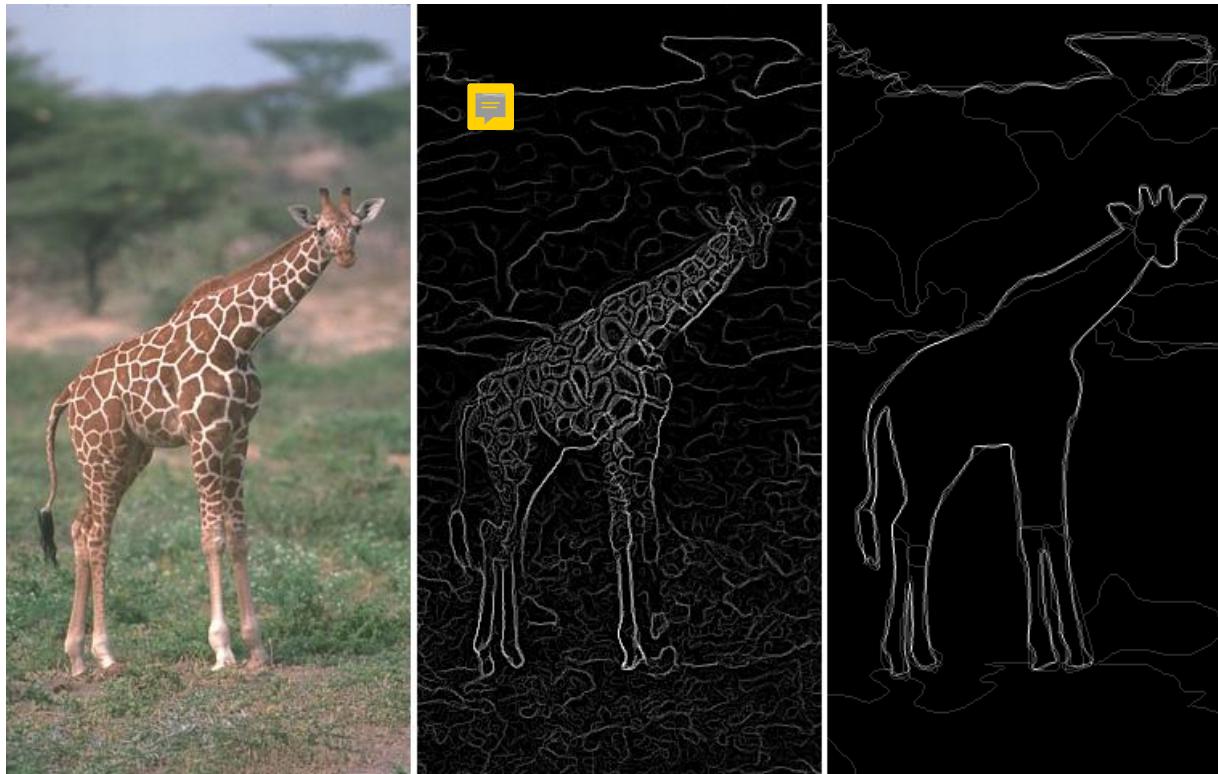
(Optional) Check hidden slides for:

- Generalized Hough transform algorithm
 - RANSAC (another voting algorithm)

Plan for today

- Edges
 - Extract gradients and threshold
- Lines and circles
 - Find which edge points are collinear or belong to another shape e.g. circle
 - Automatically detect and ignore outliers
- Segments
 - Find which pixels form a consistent region
 - Clustering (e.g. K-means)

Edges vs Segments



- Edges: More low-level; don't need to be closed
- Segments: Ideally one segment for each semantic group/object; should include closed contours



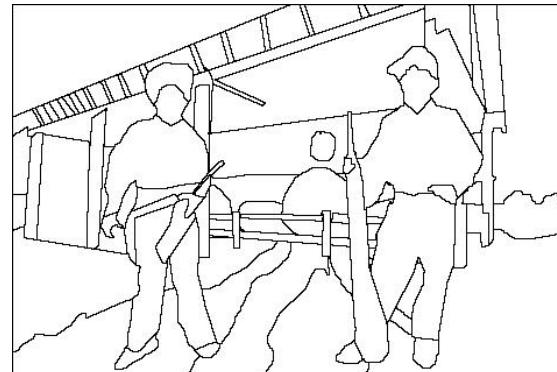
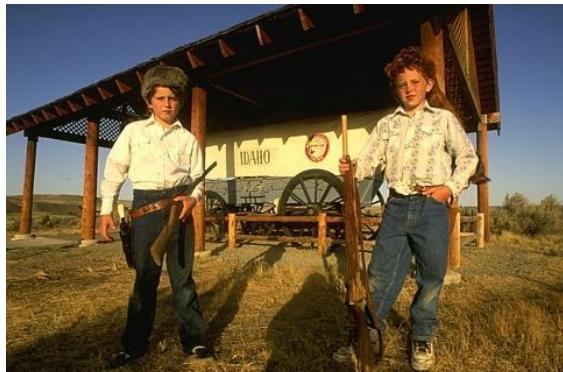
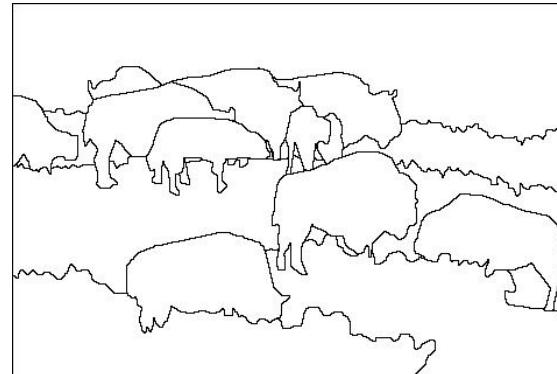
The goals of segmentation

- Separate image into coherent “objects”

image



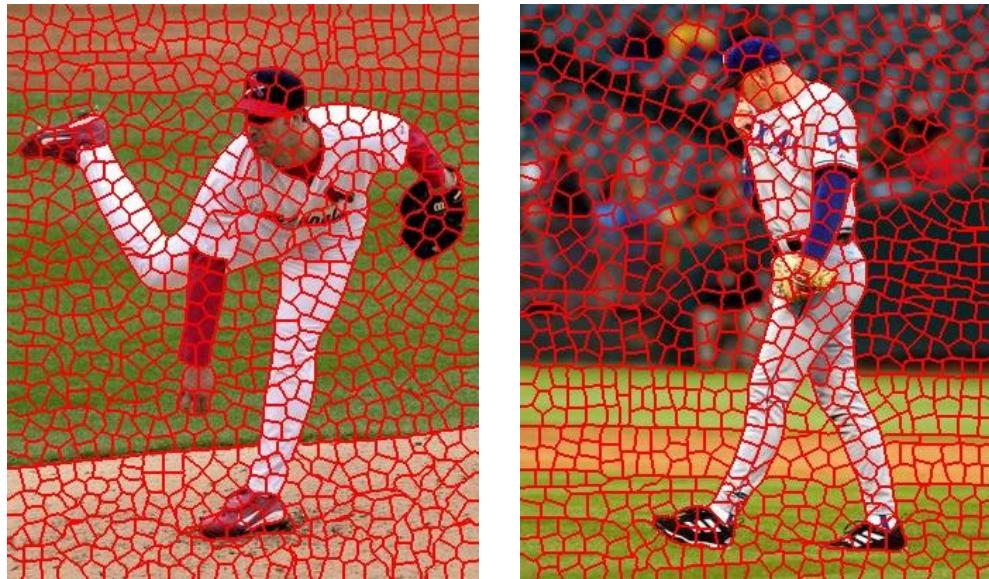
human segmentation



The goals of segmentation

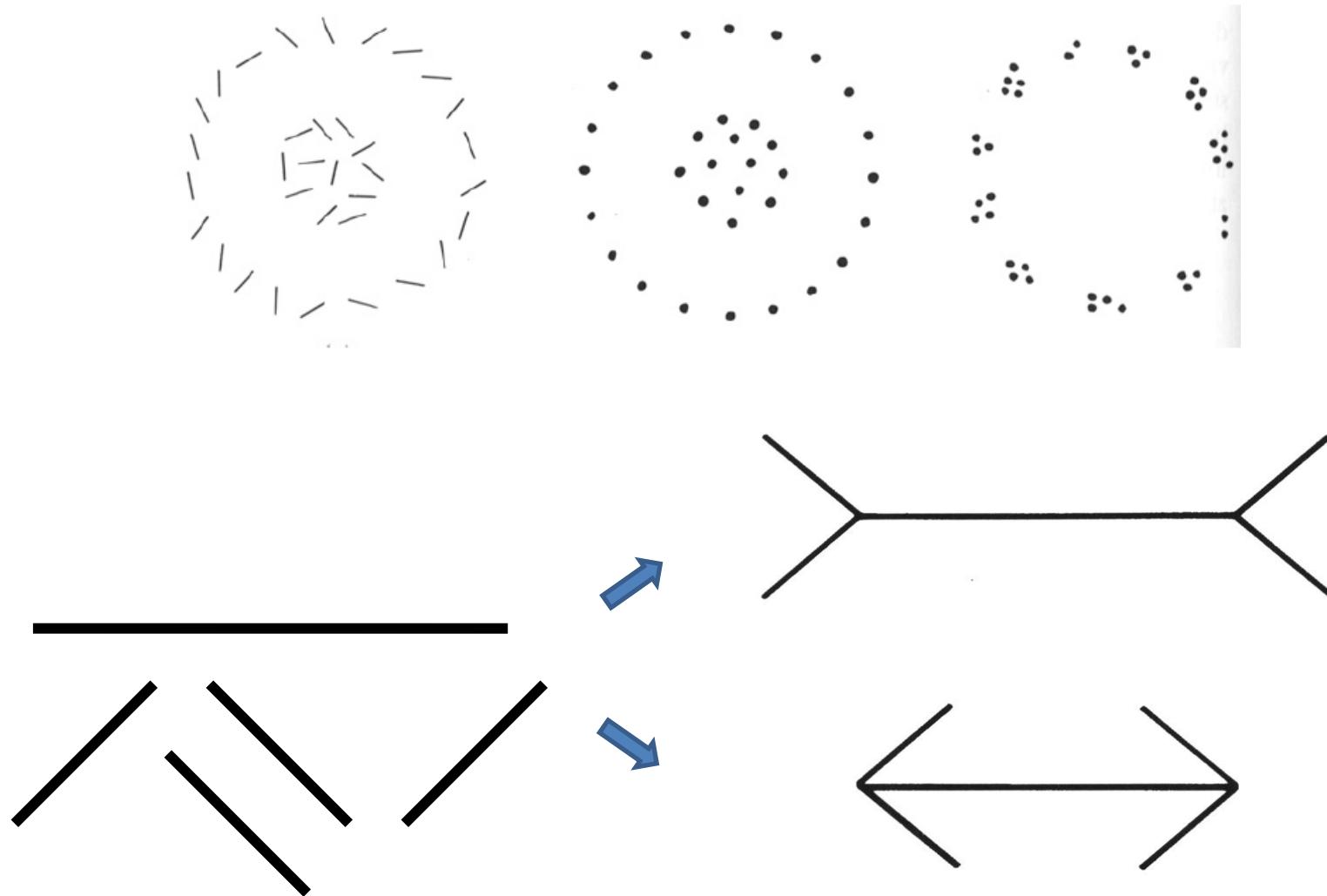
- Separate image into coherent “objects”
- Group together similar-looking pixels for efficiency of further processing

“superpixels”



X. Ren and J. Malik. [Learning a classification model for segmentation](#). ICCV 2003.

We perceive the interpretation



The Muller-Lyer illusion

Similarity



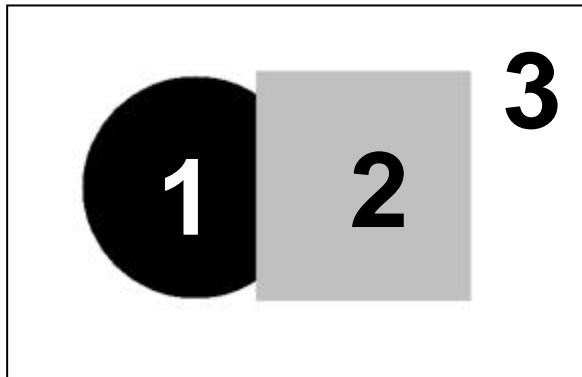
Proximity



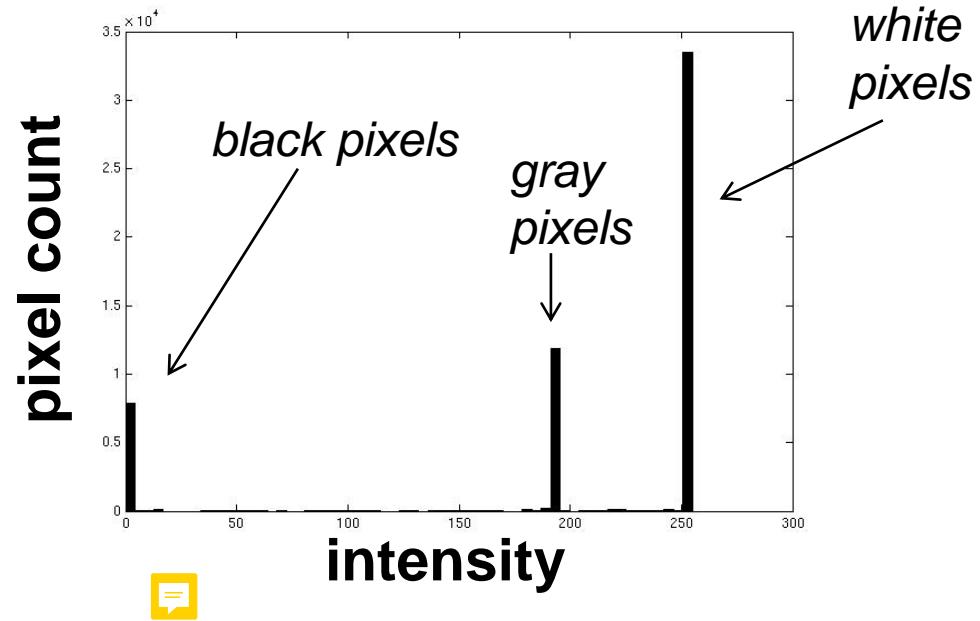
Common fate



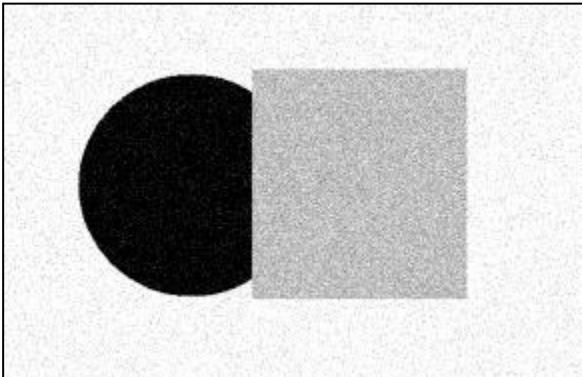
Image segmentation: toy example



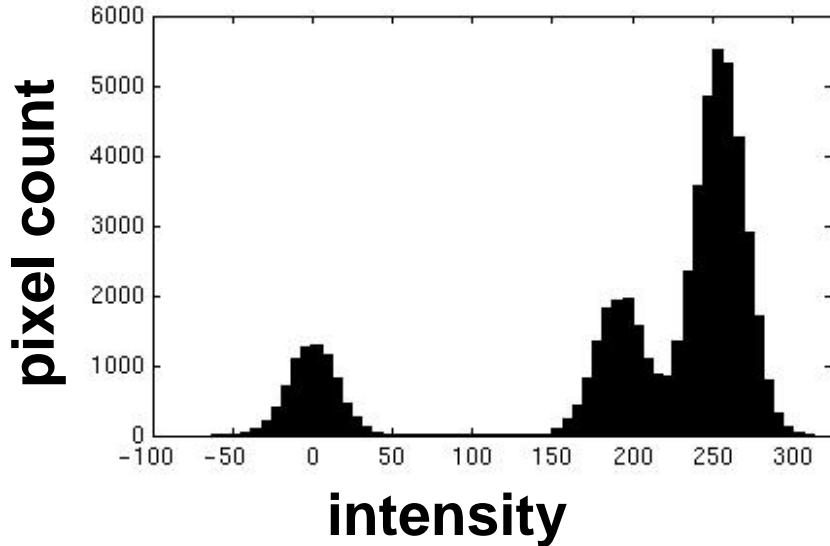
input image



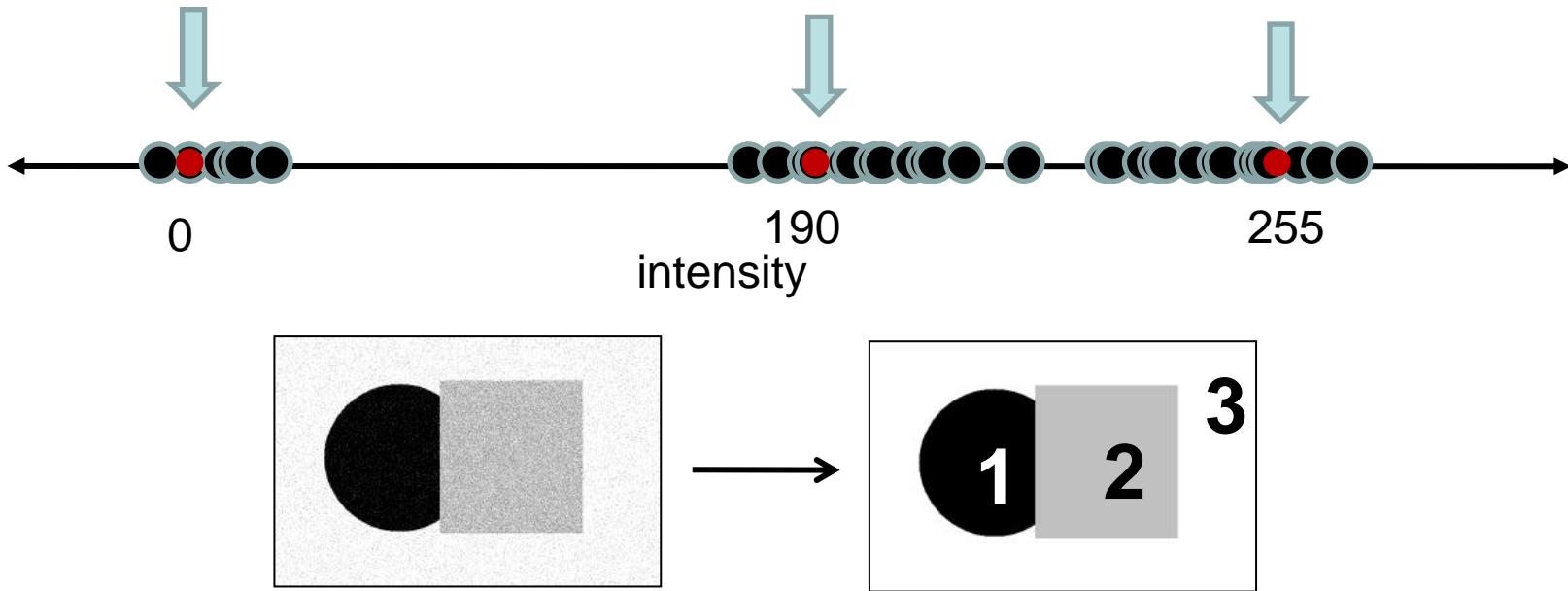
- These intensities define the three groups.
- We could label every pixel in the image according to which of these primary intensities it is.
 - i.e., *segment* the image based on the intensity feature.
- What if the image isn't quite so simple?



input image



- Now how to determine the three main intensities that define our groups?
- We need to ***cluster***.

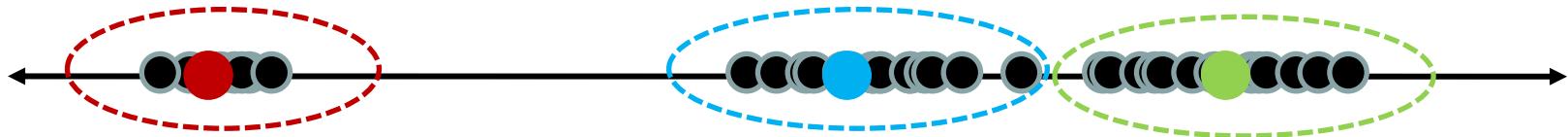


- Goal: choose three “centers” as the **representative** intensities, and label every pixel according to which of these centers it is nearest to.
- Best cluster centers are those that minimize *sum of squared differences* (SSD) between all points and their nearest cluster center c_i :

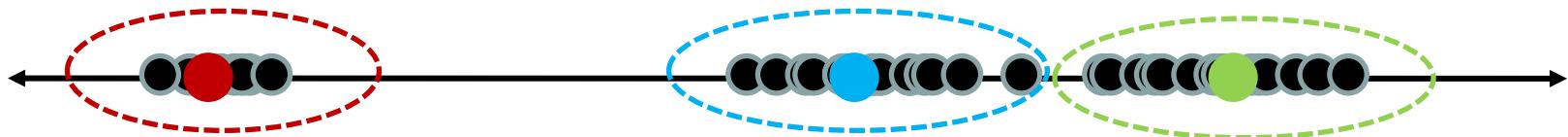
$$\sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} \|p - c_i\|^2 \quad \square$$

Clustering

- With this objective, it is a “chicken and egg” problem:
 - If we knew the **cluster centers**, we could allocate points to groups by assigning each to its closest center.

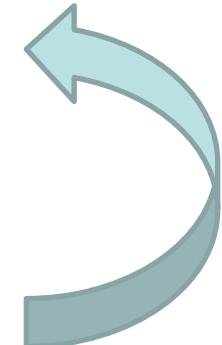


- If we knew the **group memberships**, we could get the centers by computing the mean per group.



K-means clustering

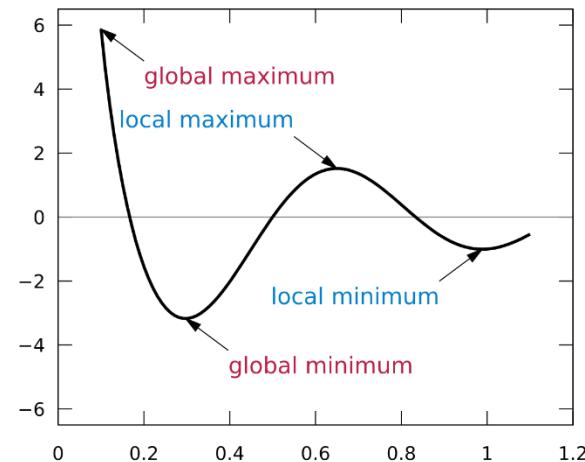
- Basic idea: randomly initialize the k cluster centers, and iterate between the two steps we just saw.
 1. *Randomly* initialize the **cluster centers**, c_1, \dots, c_K
 2. *Given cluster centers*, determine **points** in each cluster
 - For each point p , find the closest c_i . Put p into **cluster i**
 3. *Given points in each cluster*, solve for c_i
 - Set c_i to be the mean of **points** in **cluster i**
 4. If c_i have changed, repeat Step 2



Properties

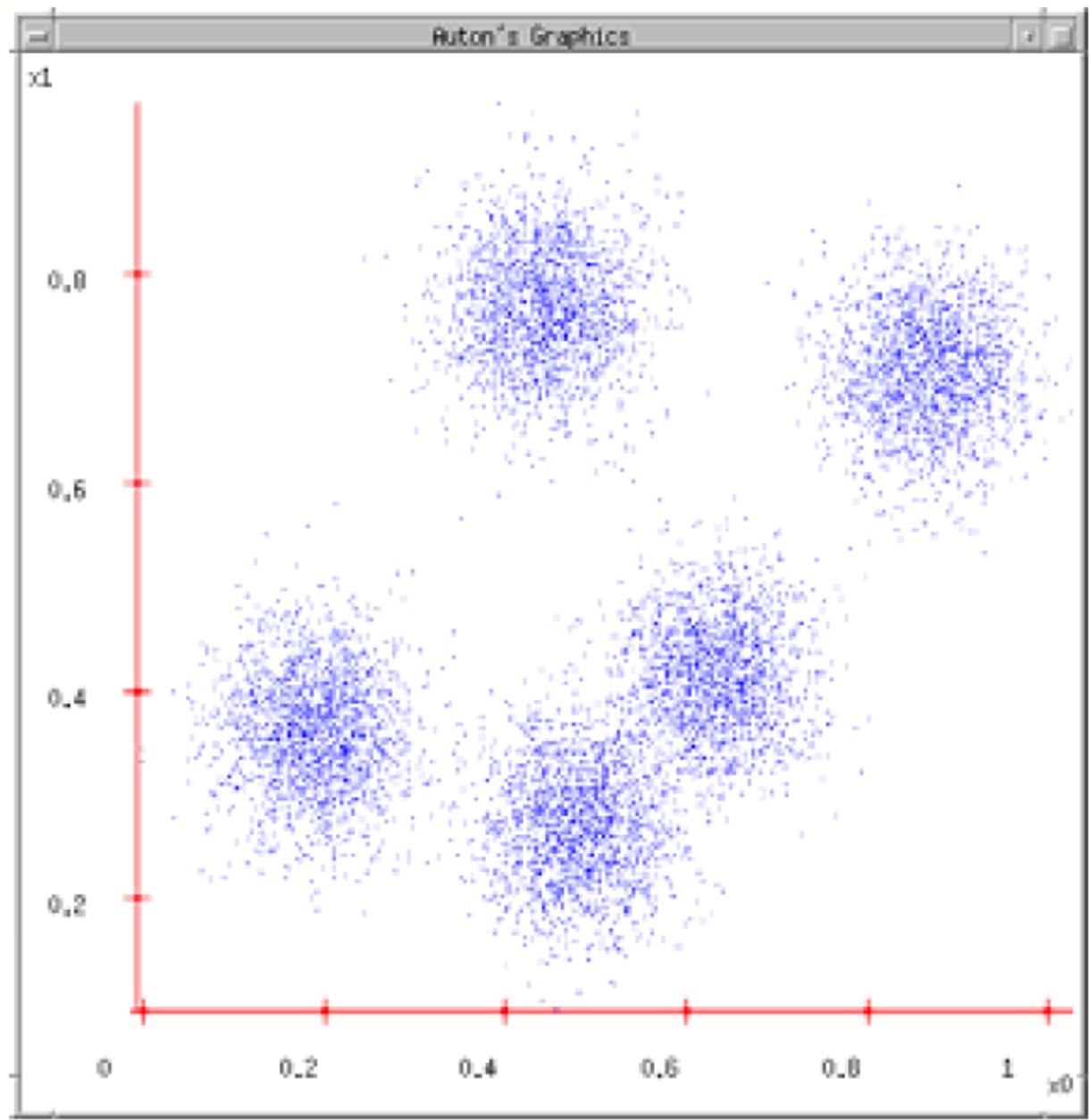
- Will always converge to *some* solution
- Can be a “local minimum” of objective:

$$\sum_{\text{clusters } i} \sum_{\text{points } p \text{ in cluster } i} \|p - c_i\|^2$$



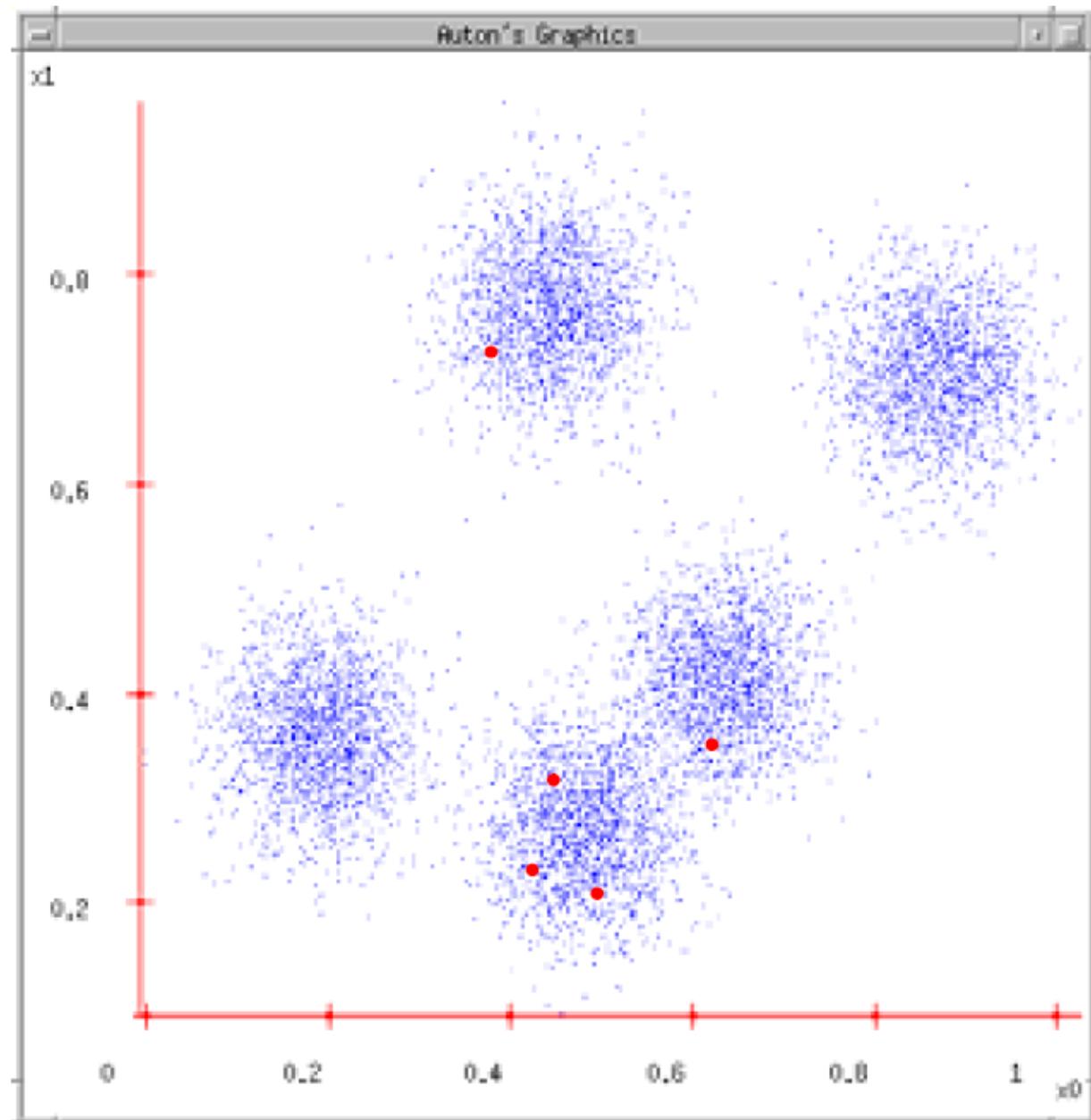
K-means

1. Ask user how many clusters they'd like.
(e.g. k=5)



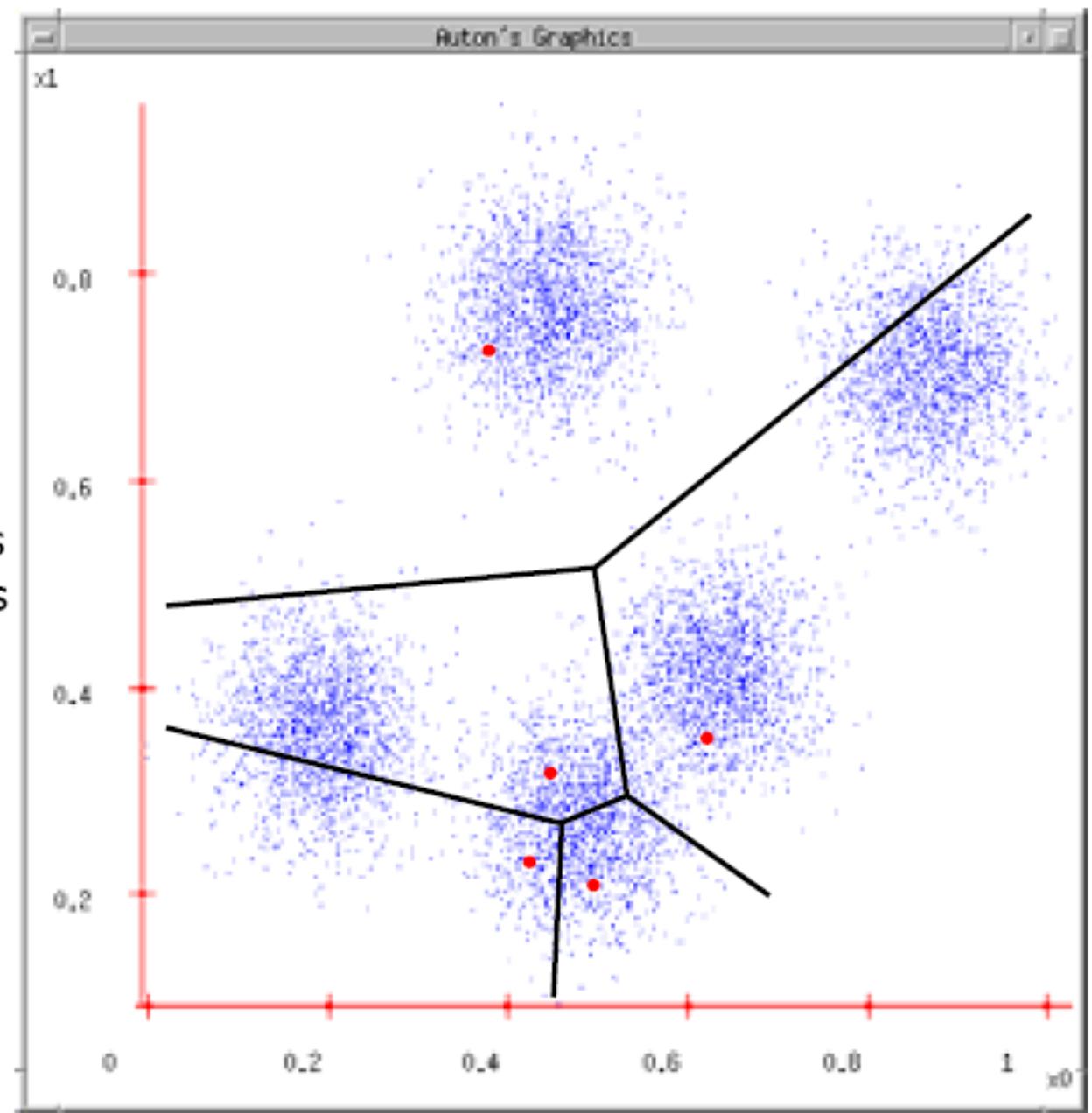
K-means

1. Ask user how many clusters they'd like.
(e.g. k=5)
2. Randomly guess k cluster Center locations



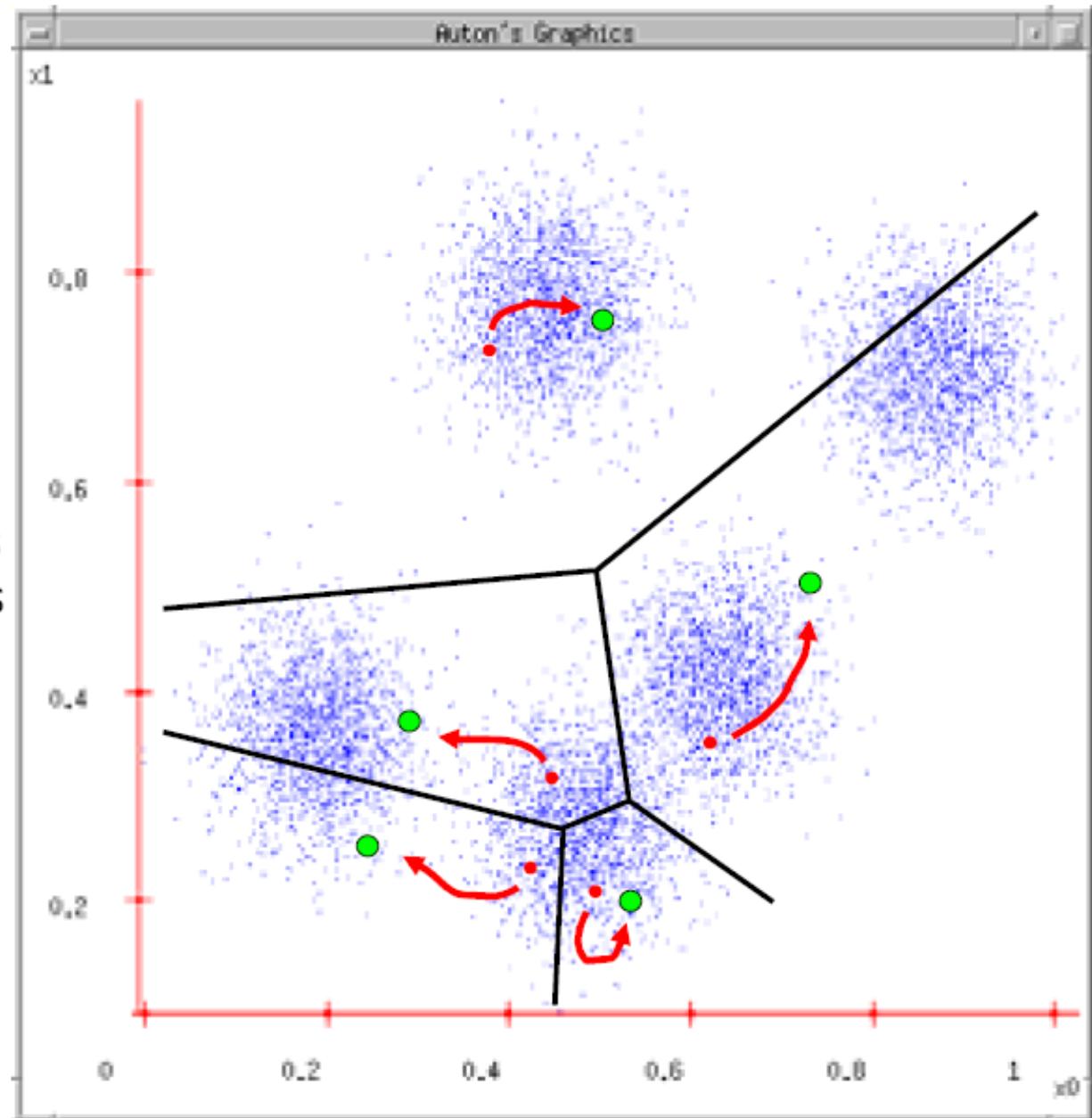
K-means

1. Ask user how many clusters they'd like.
(e.g. k=5)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)



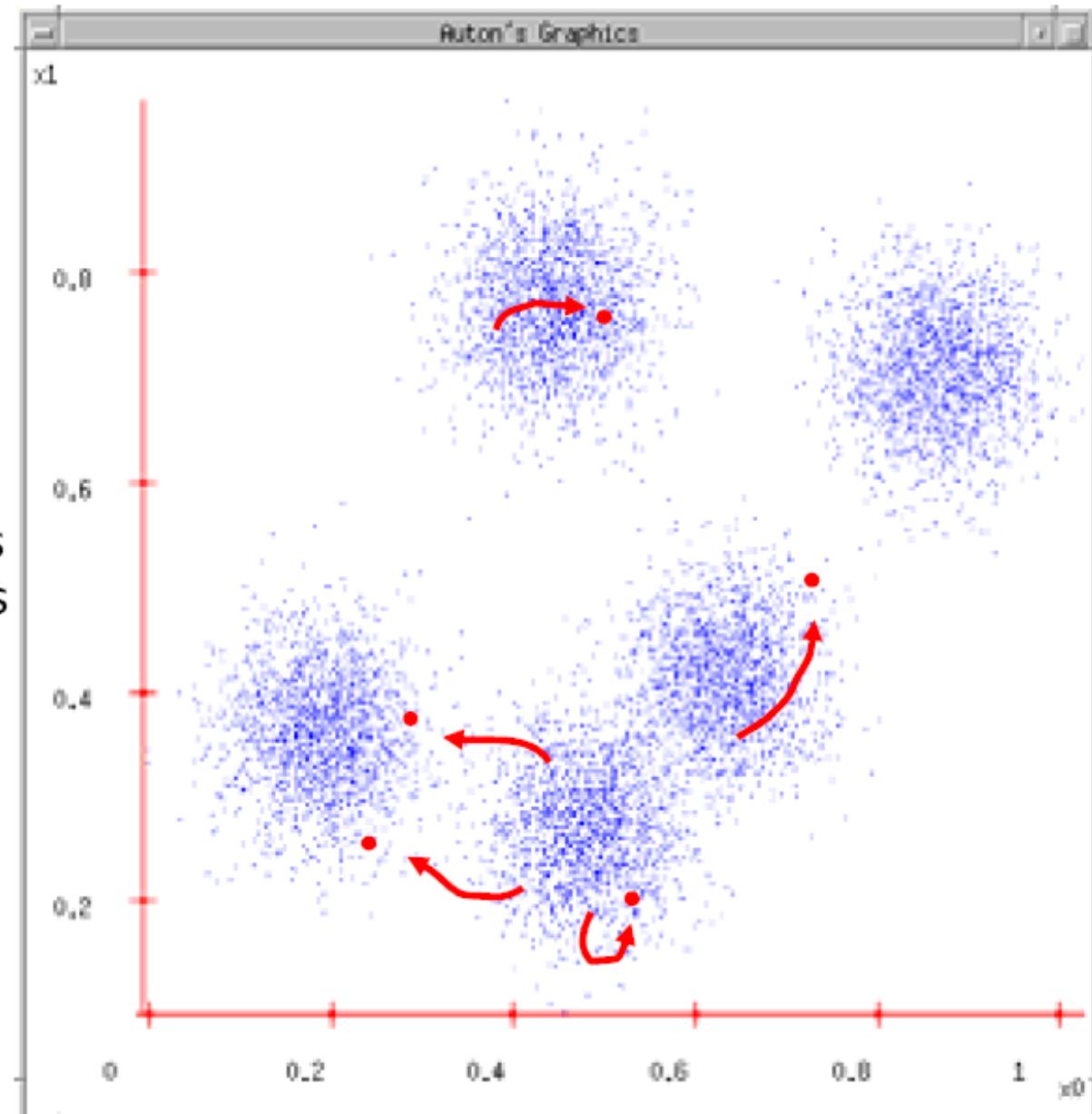
K-means

1. Ask user how many clusters they'd like.
(e.g. k=5)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns

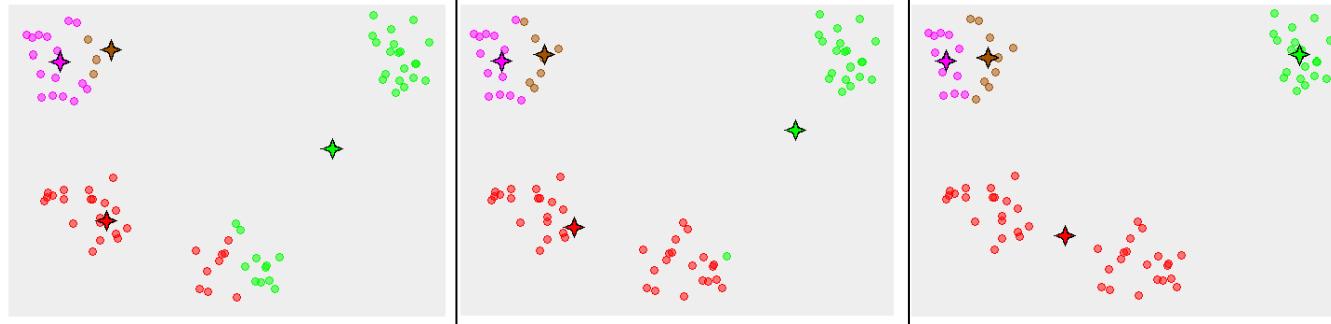
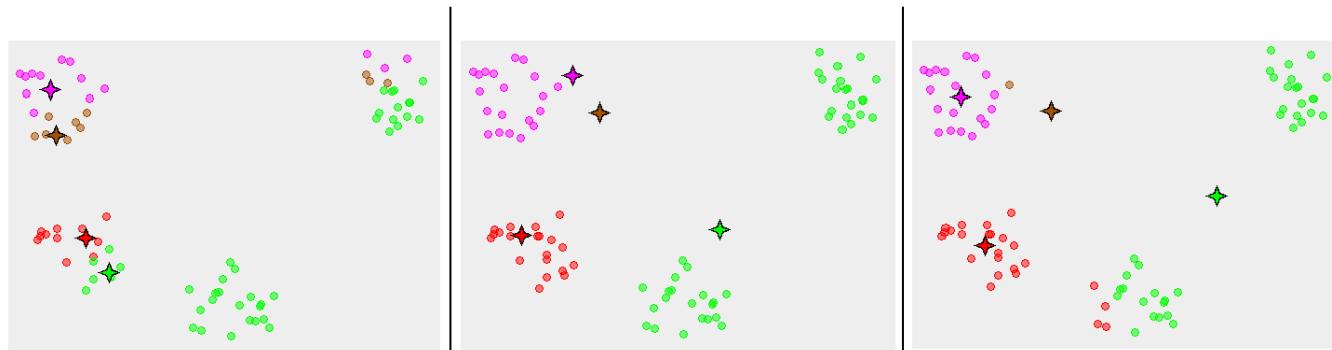


K-means

1. Ask user how many clusters they'd like.
(e.g. k=5)
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns...
5. ...and jumps there
6. ...Repeat until terminated!



K-means converges to a local minimum



How can I try to fix this problem?



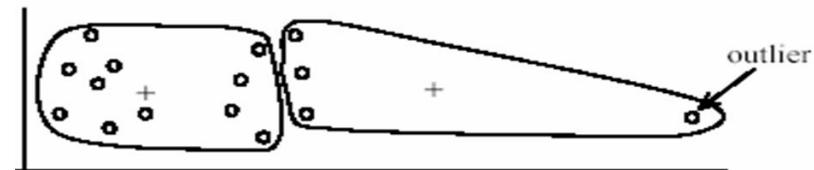
K-means: pros and cons

Pros

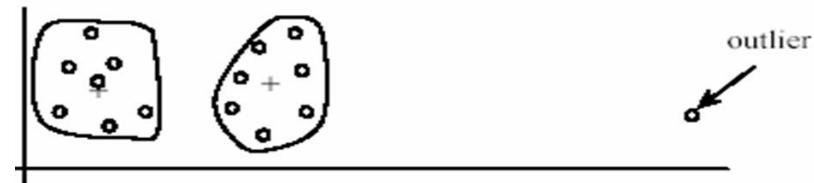
- Simple, fast to compute
- Converges to local minimum of within-cluster squared error

Cons/issues

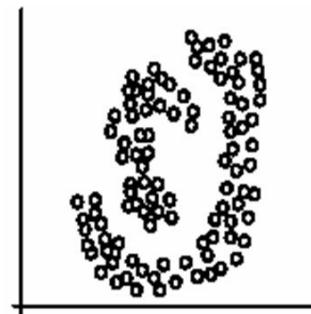
- Setting k ?
 - One way: silhouette coefficient
- Sensitive to initial centers
 - Use heuristics or output of another method
- Sensitive to outliers
- Detects spherical clusters



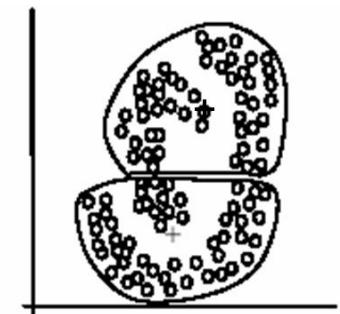
(A): Undesirable clusters



(B): Ideal clusters



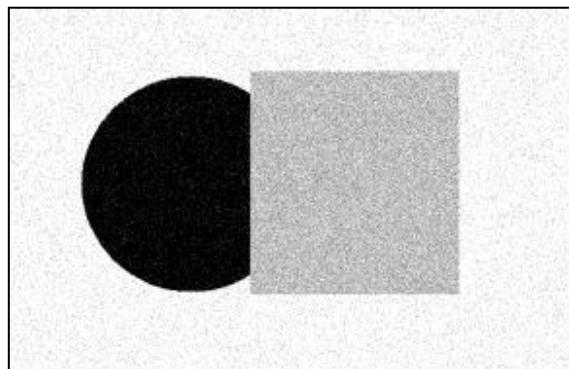
(A): Two natural clusters



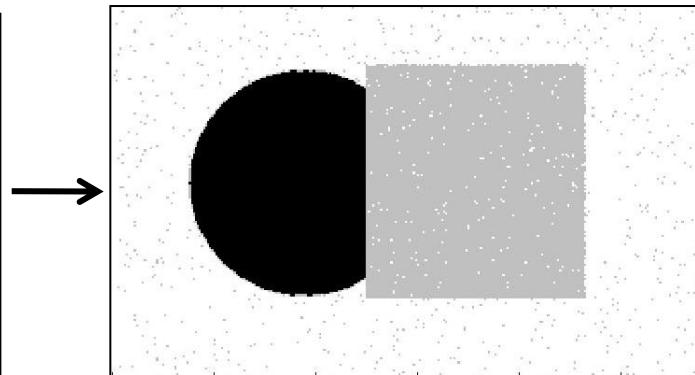
(B): k -means clusters

Aside: Smoothing cluster assignments

- Assigning a cluster label per pixel may yield outliers:

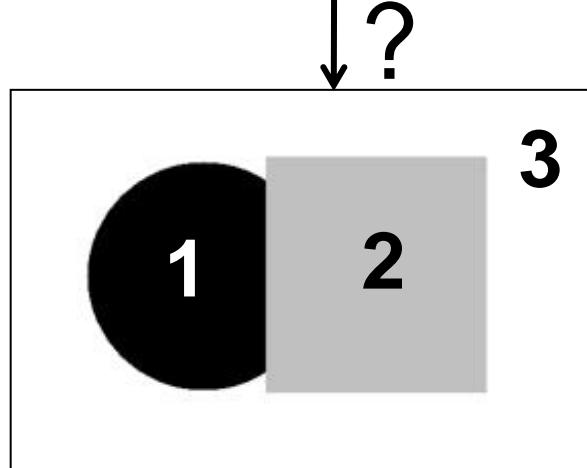


original



labeled by cluster center's
intensity

- How to ensure they are spatially smooth?



Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

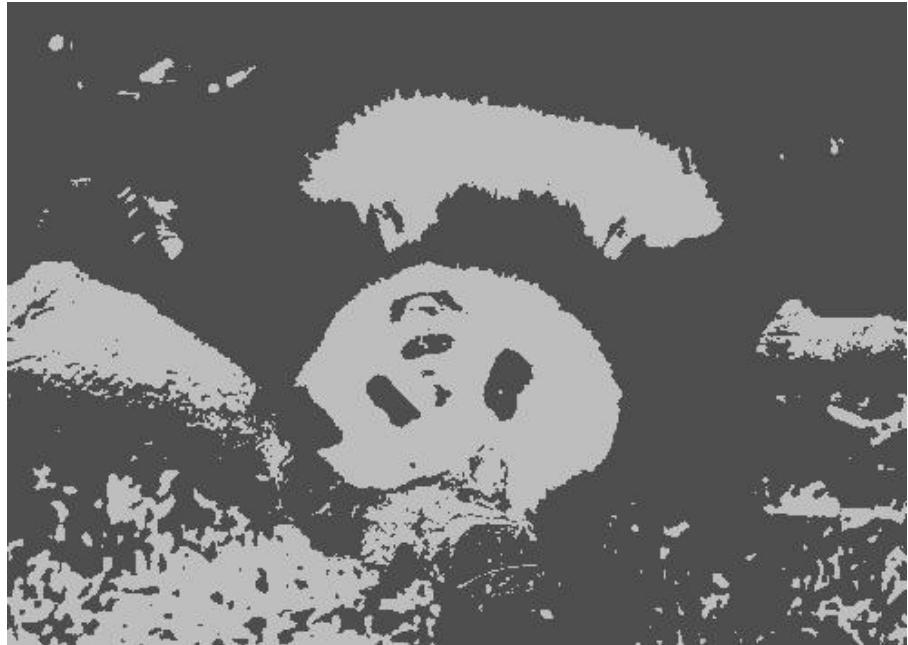
Grouping pixels based
on **intensity** similarity



Feature space: intensity value (1-d)



K=2



K=3



Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based
on **intensity** similarity



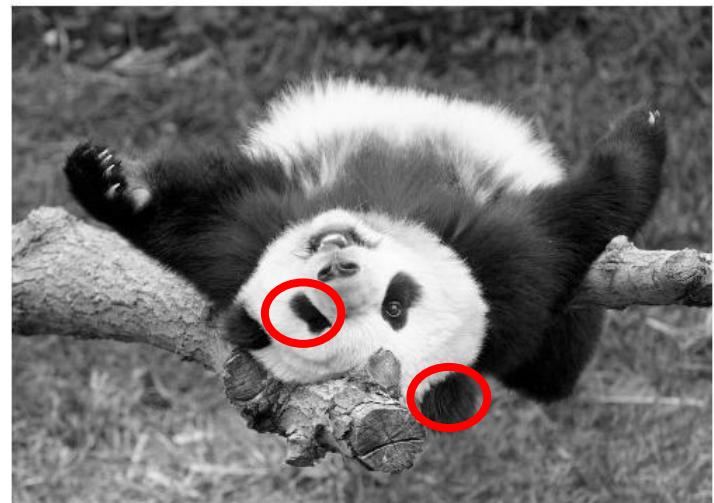
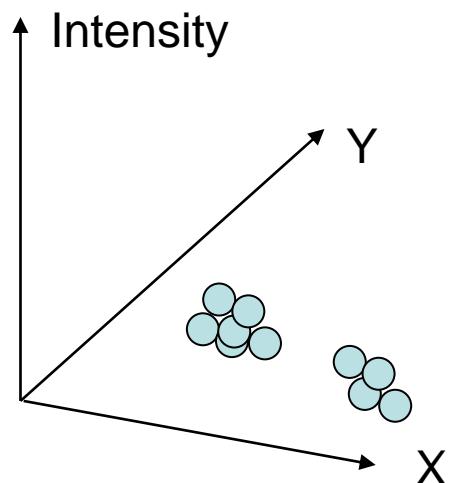
Clusters based on intensity
similarity don't have to be spatially
coherent.



Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based
on **intensity+position** similarity

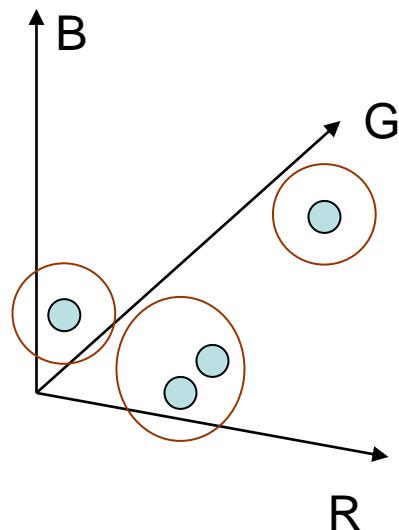


Both regions are black, but if we also include **position (x,y)**, then we could group the two into distinct segments; way to encode both similarity & proximity.

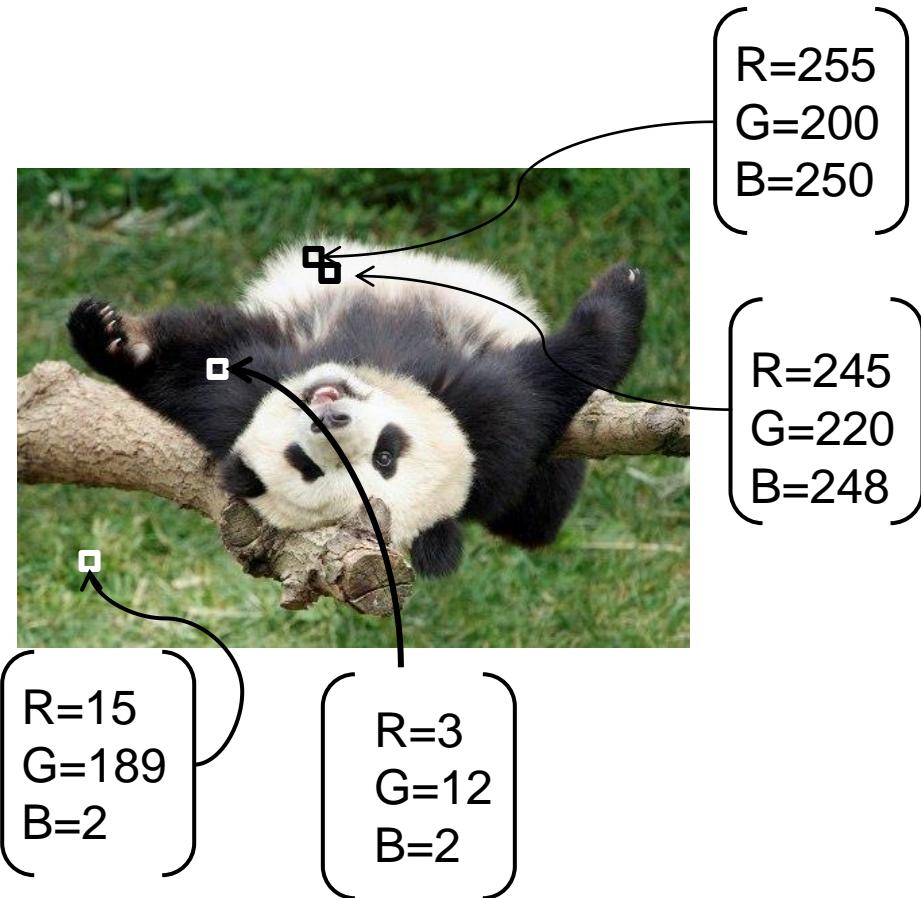
Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **color** similarity



Feature space: color value (3-d)



Source: K. Grauman

Segmentation as clustering

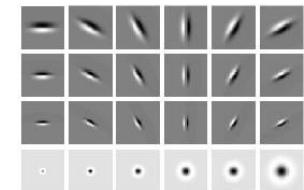
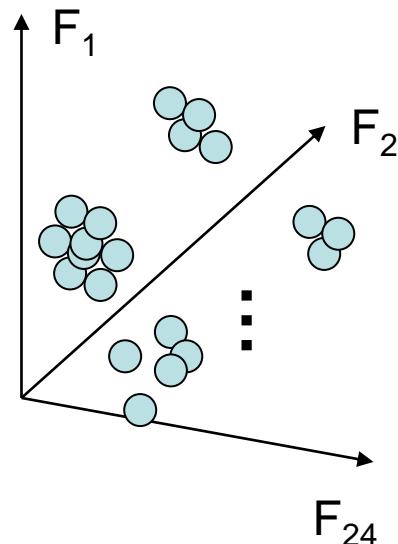
- Color, brightness, position alone are not enough to distinguish all regions...



Segmentation as clustering

Depending on what we choose as the *feature space*, we can group pixels in different ways.

Grouping pixels based on **texture** similarity

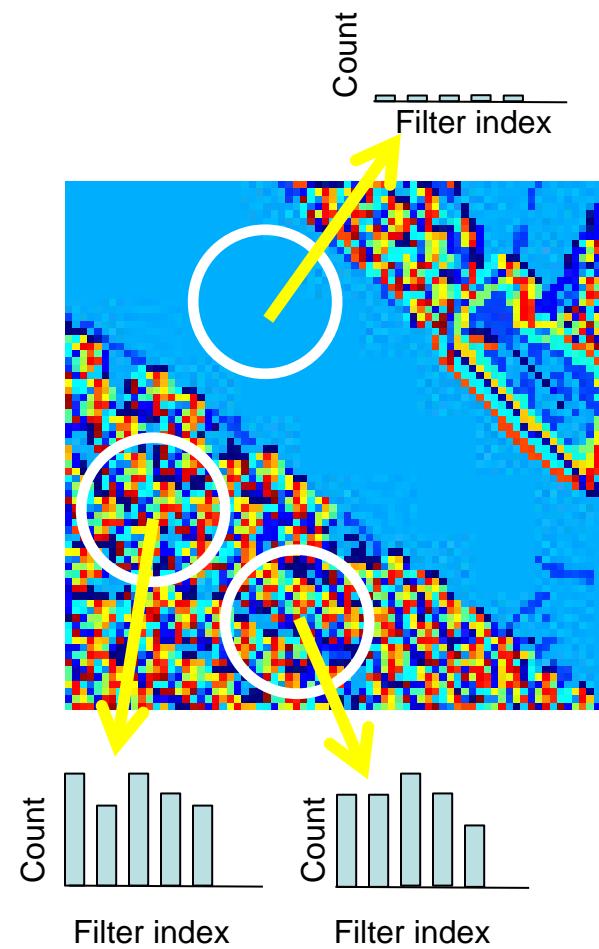


Filter bank
of 24 filters

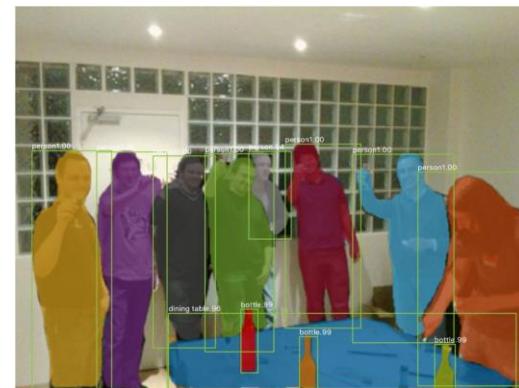
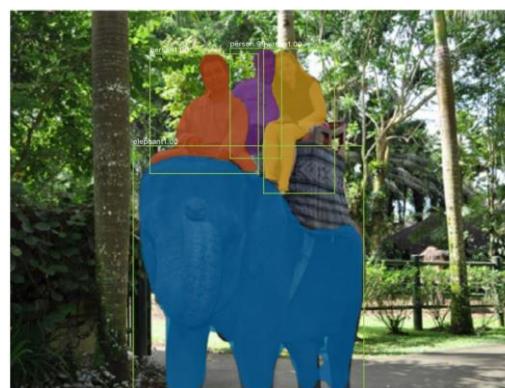
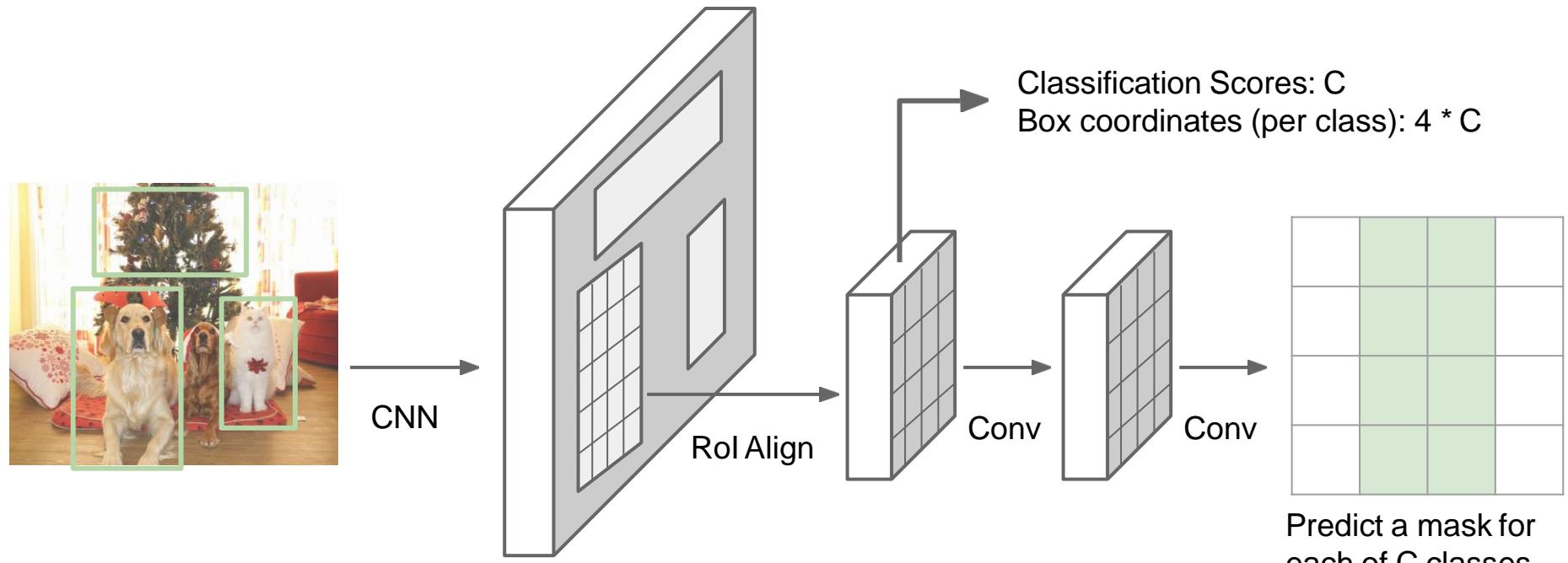
Feature space: filter bank responses (e.g., 24-d)

Segmentation w/ texture features

- Describe texture in a window as histogram over filter bank responses (simplified version, better: use “textons”)



State-of-the-art (instance) segmentation: Mask R-CNN



Summary: classic approaches

- Edges: threshold gradient magnitude 
- Lines: edge points vote for parameters of line, circle, etc. (works for general objects)
- Segments: use clustering (e.g. K-means) to group pixels by intensity, texture, etc.