*CS 2770: Computer Vision*

# Grouping & Transformations

Prof. Adriana Kovashka
University of Pittsburgh
January 29, 2018

# Plan for this lecture

- Group pixels into:
  - Edges: Extract gradients and threshold
  - Lines: Find which edge points are collinear or belong to another shape
  - Segments: Find which pixels form a consistent region, e.g. via clustering
- Transform pixels:
  - Find relationships between multiple views of the same world point
  - *Both parts rely on finding geometric relationships between pixels*

# Edge detection

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

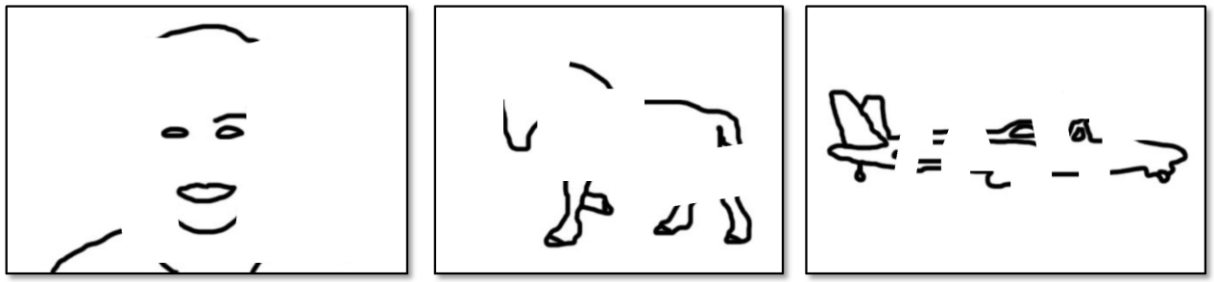


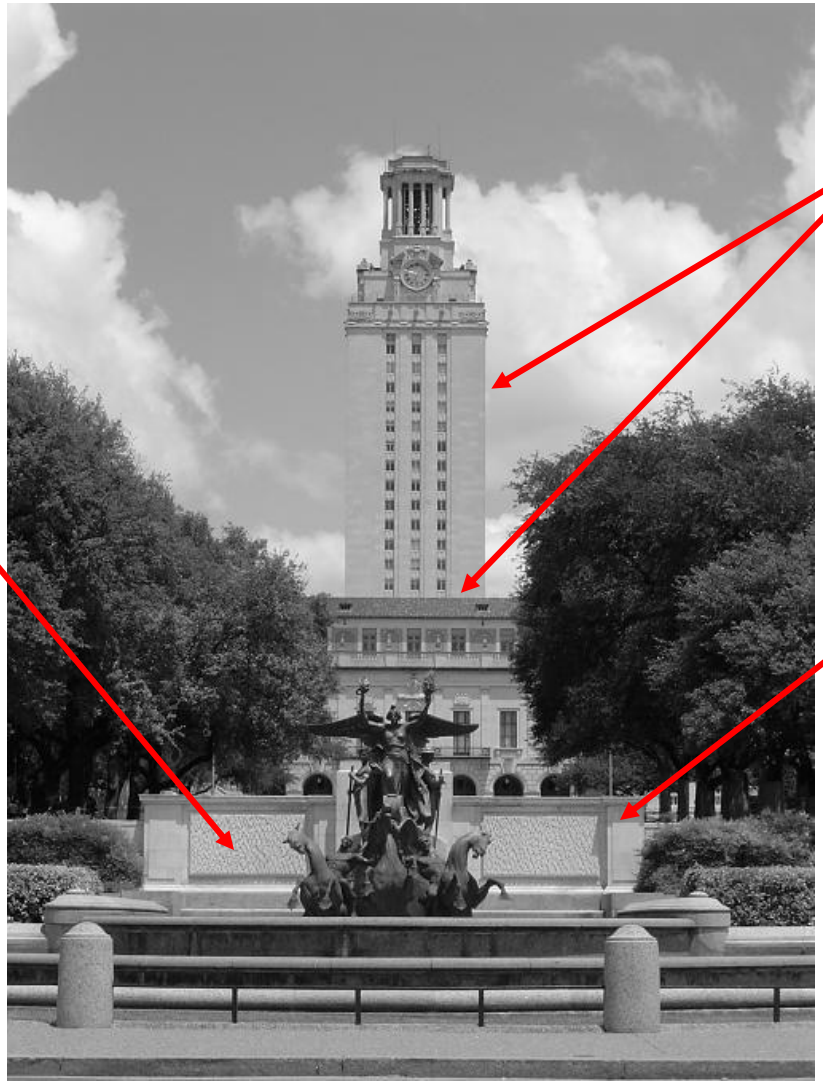
**Figure from J. Shotton et al., PAMI 2007**

- **Main idea**: look for differences in intensity, i.e. find strong gradients, then post-process

Adapted from K. Grauman

# What causes an edge?

Reflectance change: appearance information, texture



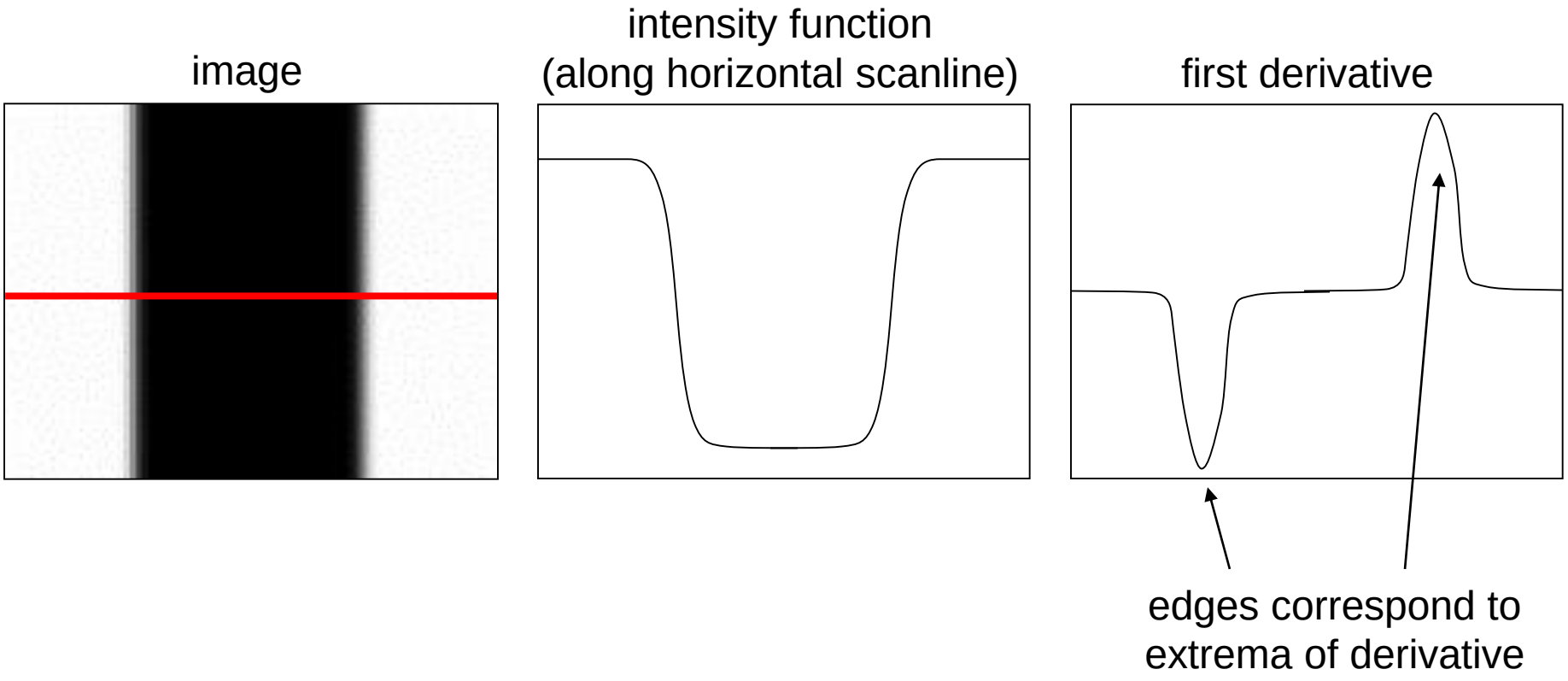
Depth discontinuity: object boundary

Cast shadows

Adapted from K. Grauman

# Characterizing edges

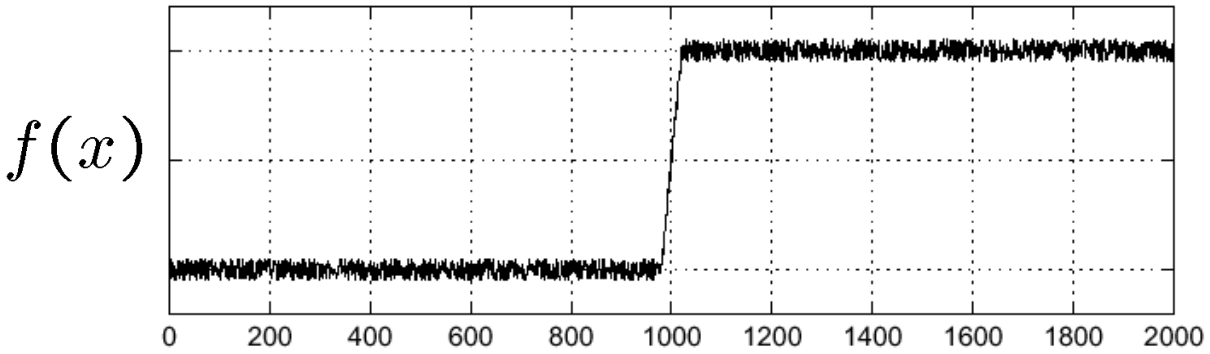
- An edge is a place of rapid change in the image intensity function

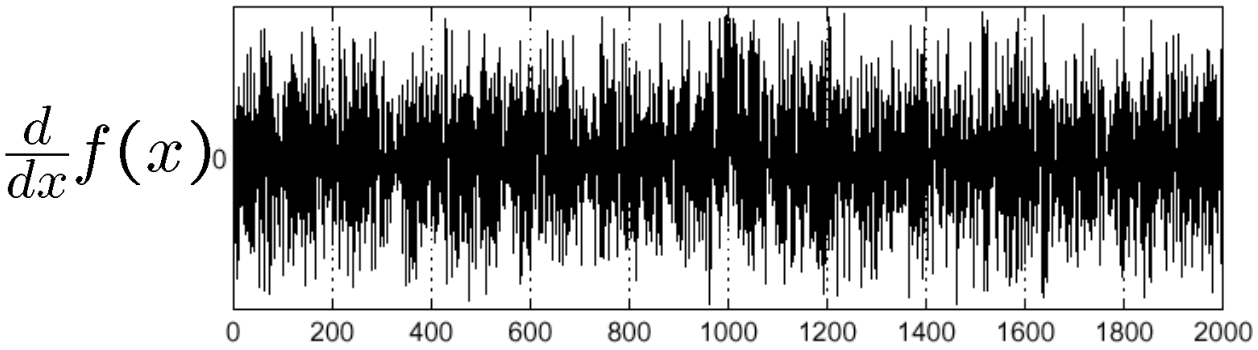



Source: L. Lazebnik

# Now with a little noise…

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

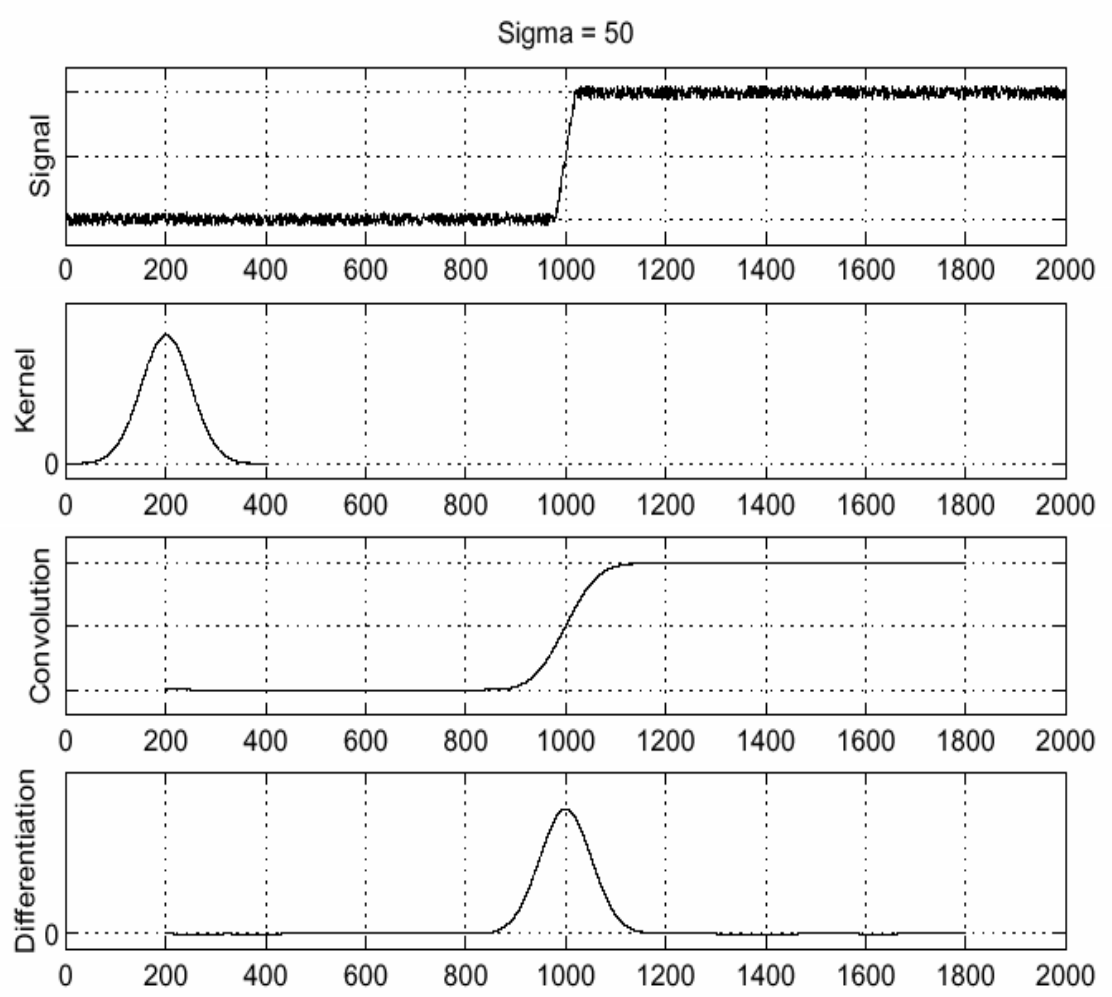



Where is the edge?

Source: S. Seitz

# Solution: smooth first




$f$

$g$

$f * g$

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

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

Source: S. Seitz

# 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:

$f$

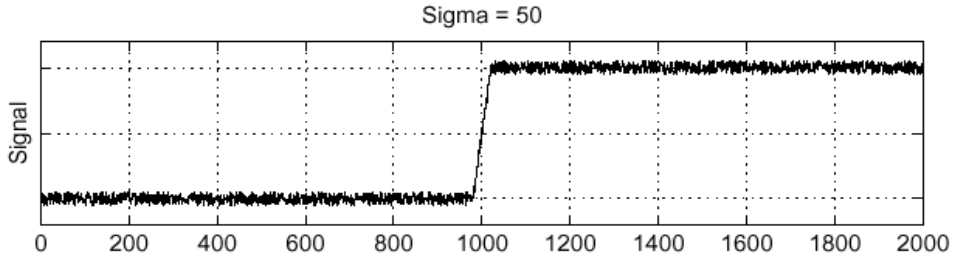
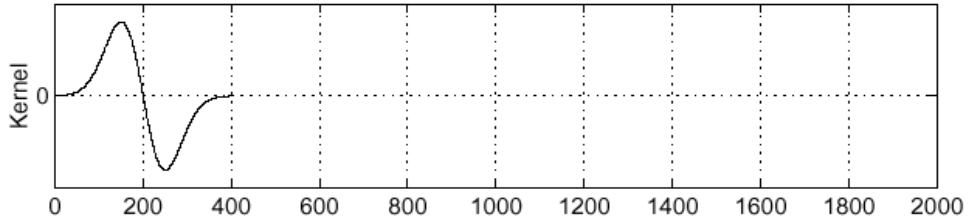


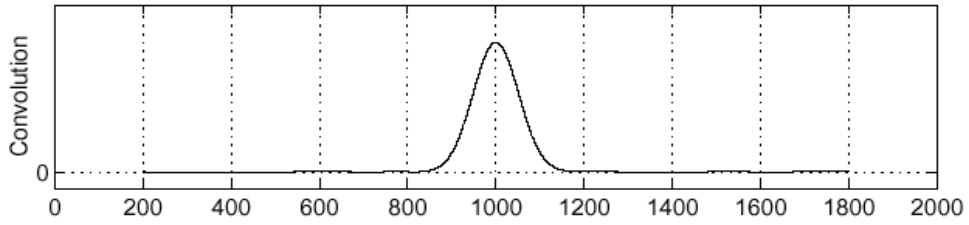

Image with edge

$\frac{d}{dx} g$




Derivative of Gaussian

$f * \frac{d}{dx} g$




Edge = max of derivative

Source: S. Seitz

# Canny edge detector

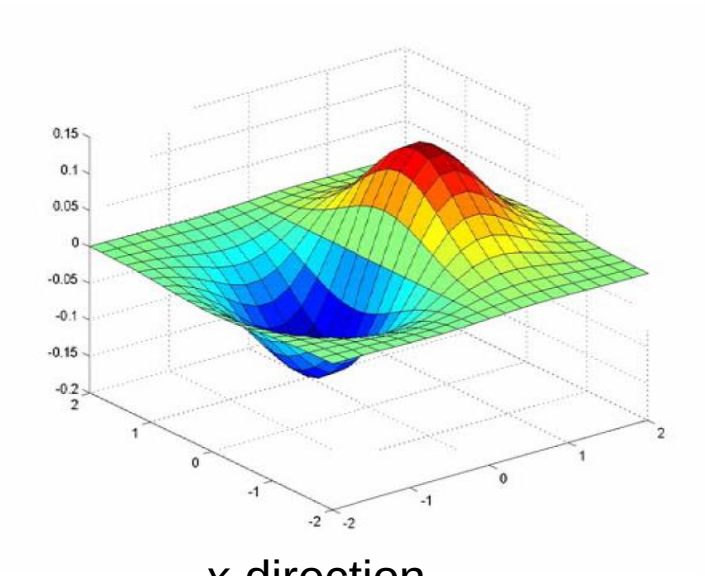
- Filter image with derivative of Gaussian
- Find magnitude and orientation of gradient
- Threshold: Determine which local maxima from filter output are actually edges
- **Non-maximum suppression**:
  - Thin wide “ridges” down to single pixel width
- **Linking and thresholding** (**hysteresis**):
  -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

Adapted from K. Grauman, D. Lowe, L. Fei-Fei

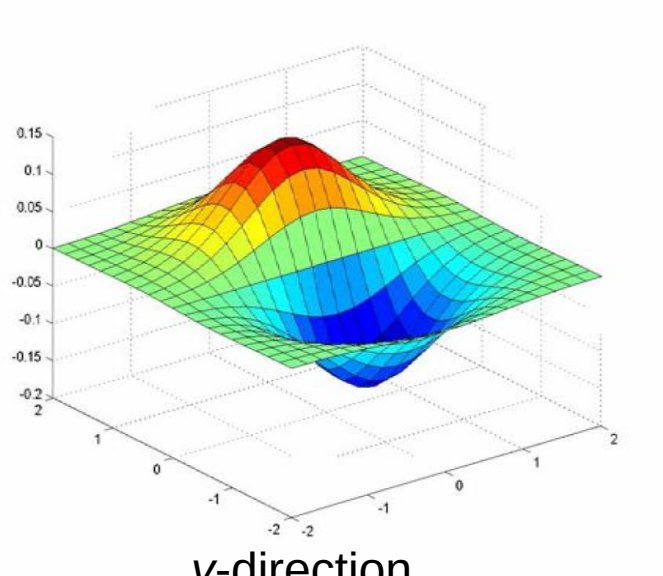
# Example

input image (“Lena”)

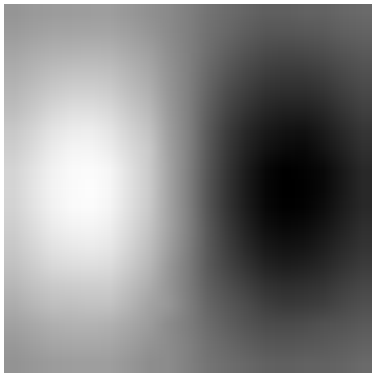
# Derivative of Gaussian filter

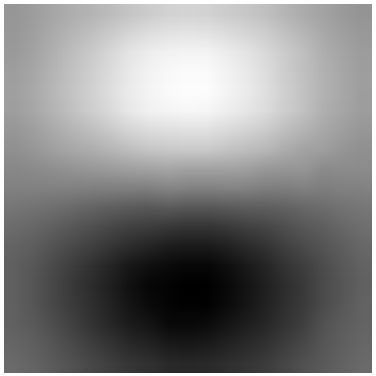



*x*-direction



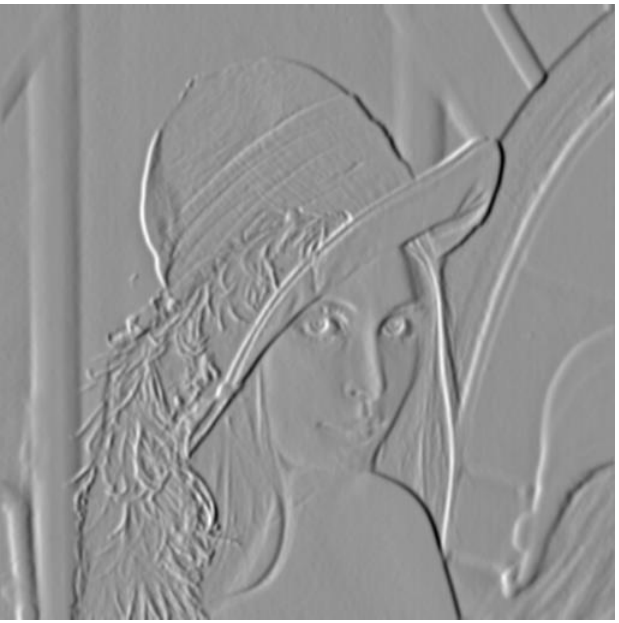

*y*-direction





Source: L. Lazebnik

# Compute Gradients



X-Derivative of Gaussian

Y-Derivative of Gaussian

Gradient Magnitude

Source: D. Hoiem

# 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)

Source: K. Grauman

# The Canny edge detector

norm of the gradient (magnitude)

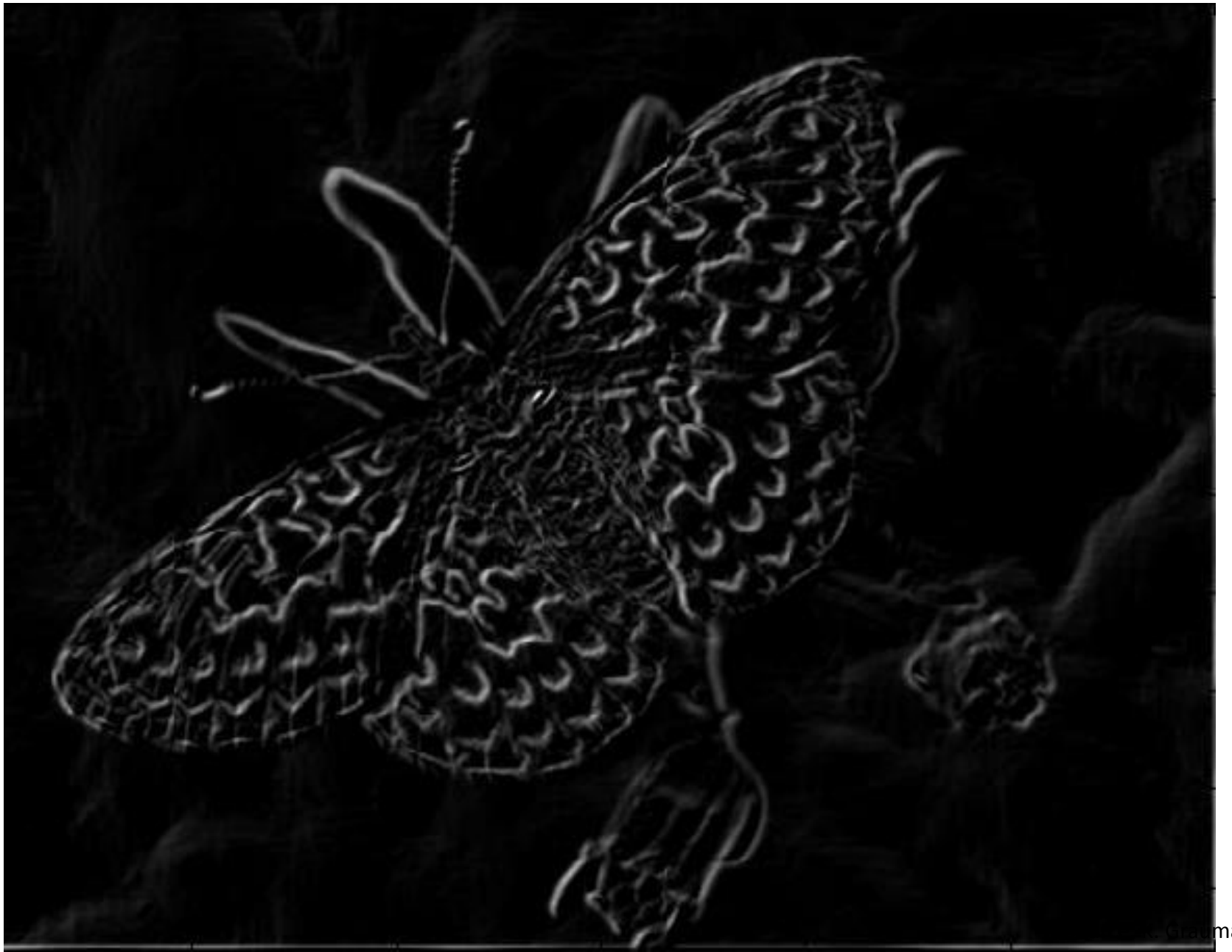
Source: K. Grauman

# The Canny edge detector

thresholding

Source: K. Grauman

# Another example: Gradient magnitudes



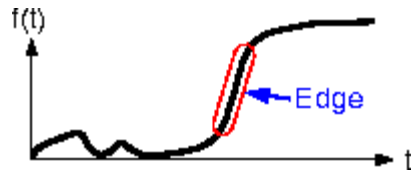
# Thresholding gradient with a lower threshold

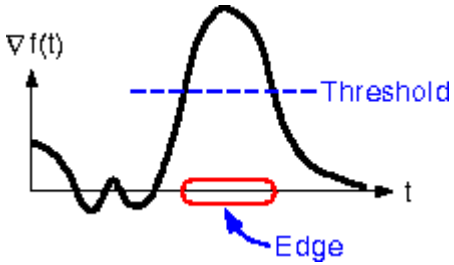
Source: K. Grauman

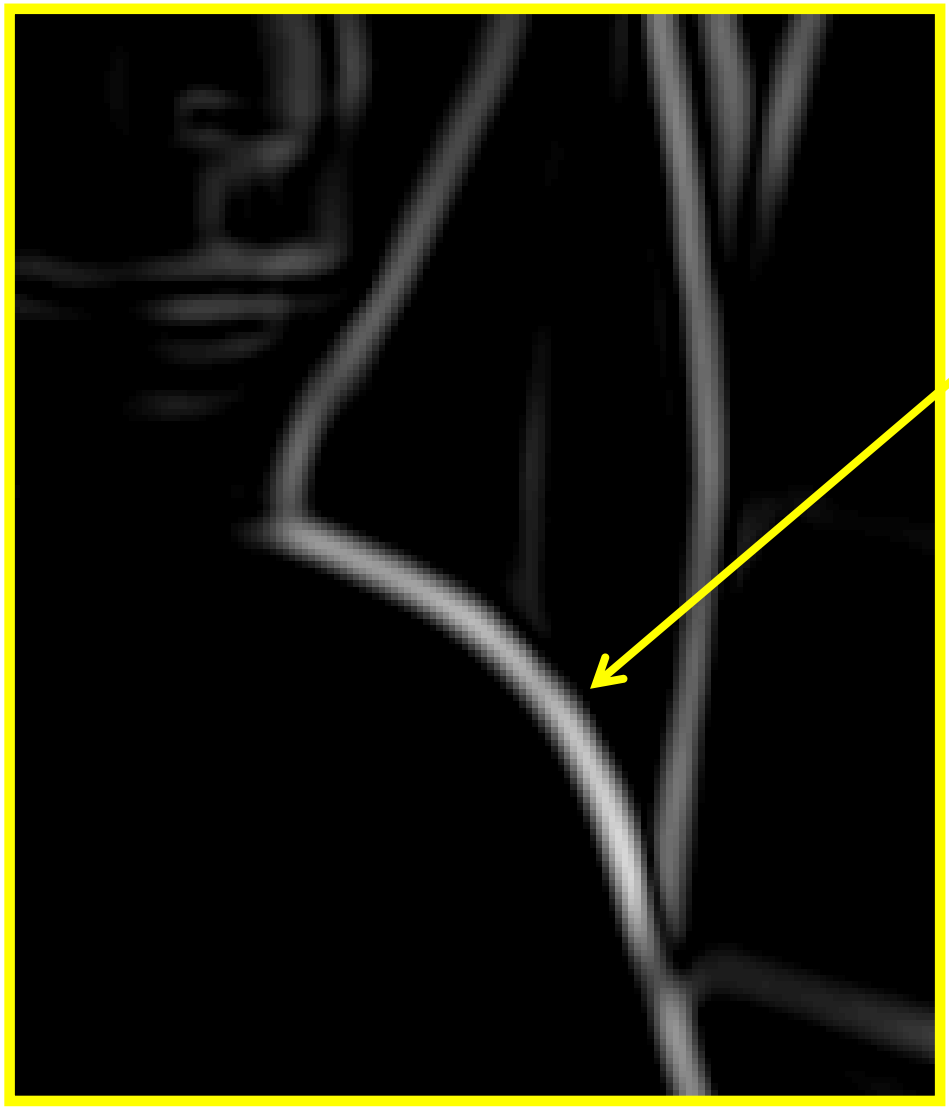
# Thresholding gradient with a higher threshold

Source: K. Grauman

# The Canny edge detector



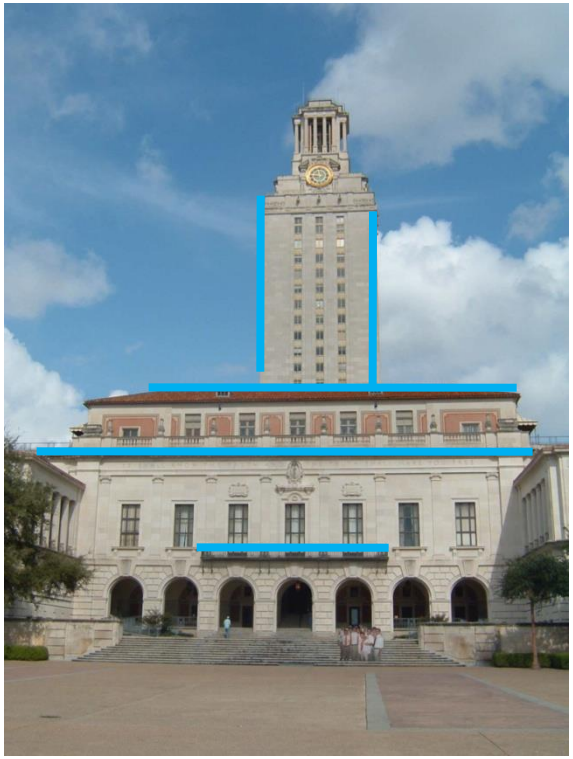
How to turn these thick regions of the gradient into curves?

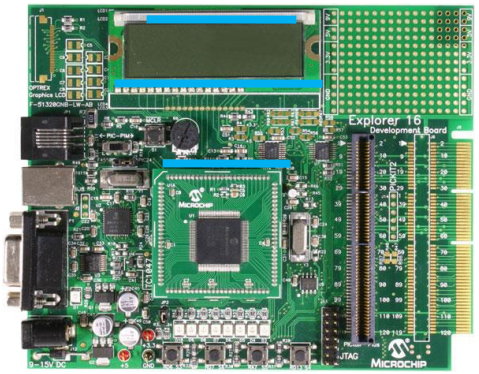
**See hidden slides.**

Source: K. Grauman

# Related: Line detection (fitting)

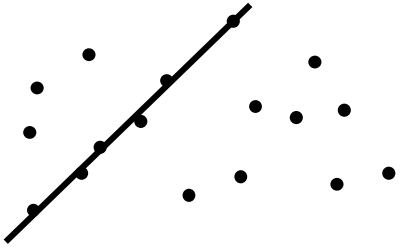
- Why fit lines?
  Many objects characterized by presence of straight lines

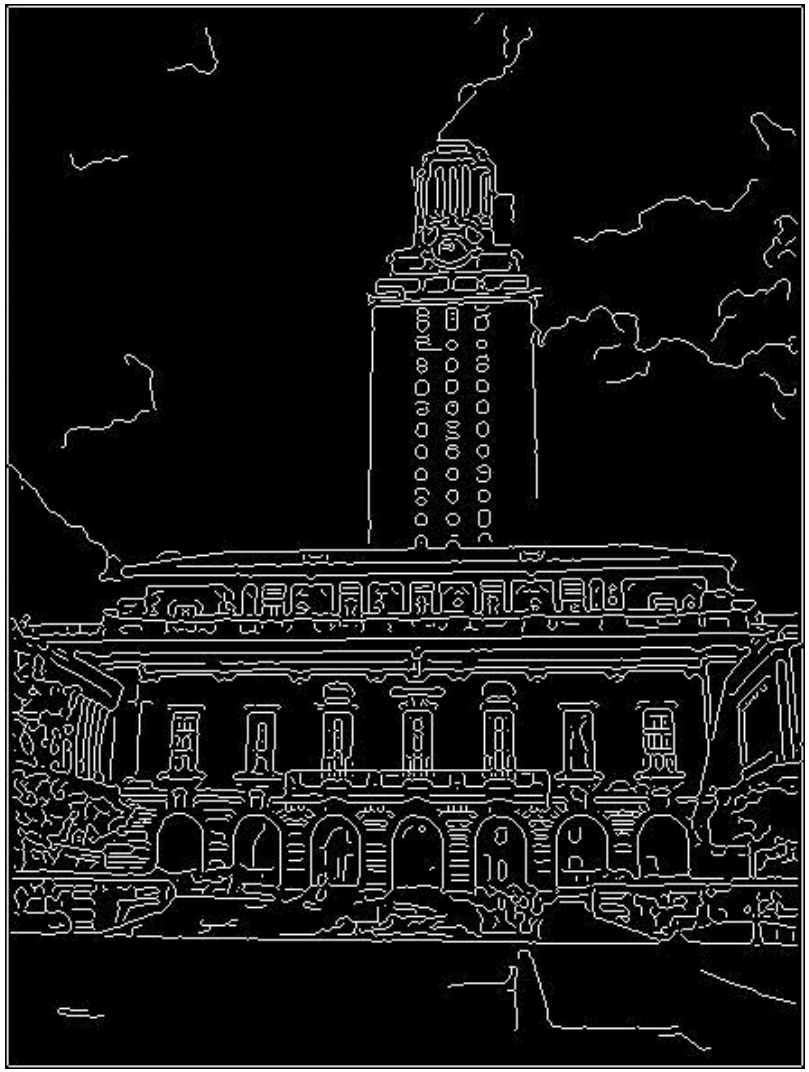





- Why aren’t we done just by running edge detection?

# Difficulty of line fitting



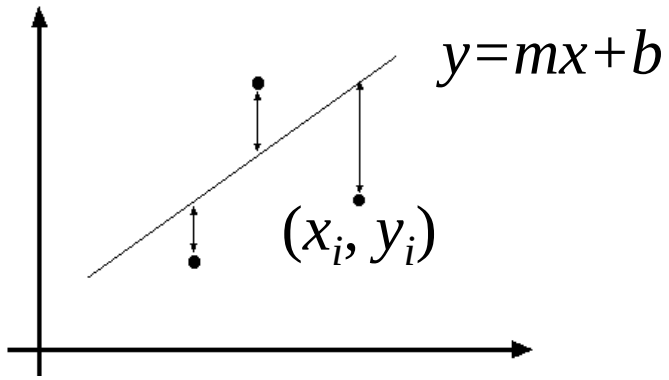


- **Noise** in measured edge points, orientations:
  - e.g. edges not collinear where they should be
  - how to detect true underlying parameters?
- **Extra** edge points (clutter):
  - which points go with which line, if any?
- Only some parts of each line detected, and some parts are **missing:**
  - how to find a line that bridges missing evidence?

Adapted from Kristen Grauman

# Least squares line fitting

- Data: $(x_1, y_1), \ldots, (x_n, y_n)$
- Line equation: $y_i = m x_i + b$




- Find $(m, b)$ to minimize

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

where line you found tells you point is along y axis

where point really is along y axis

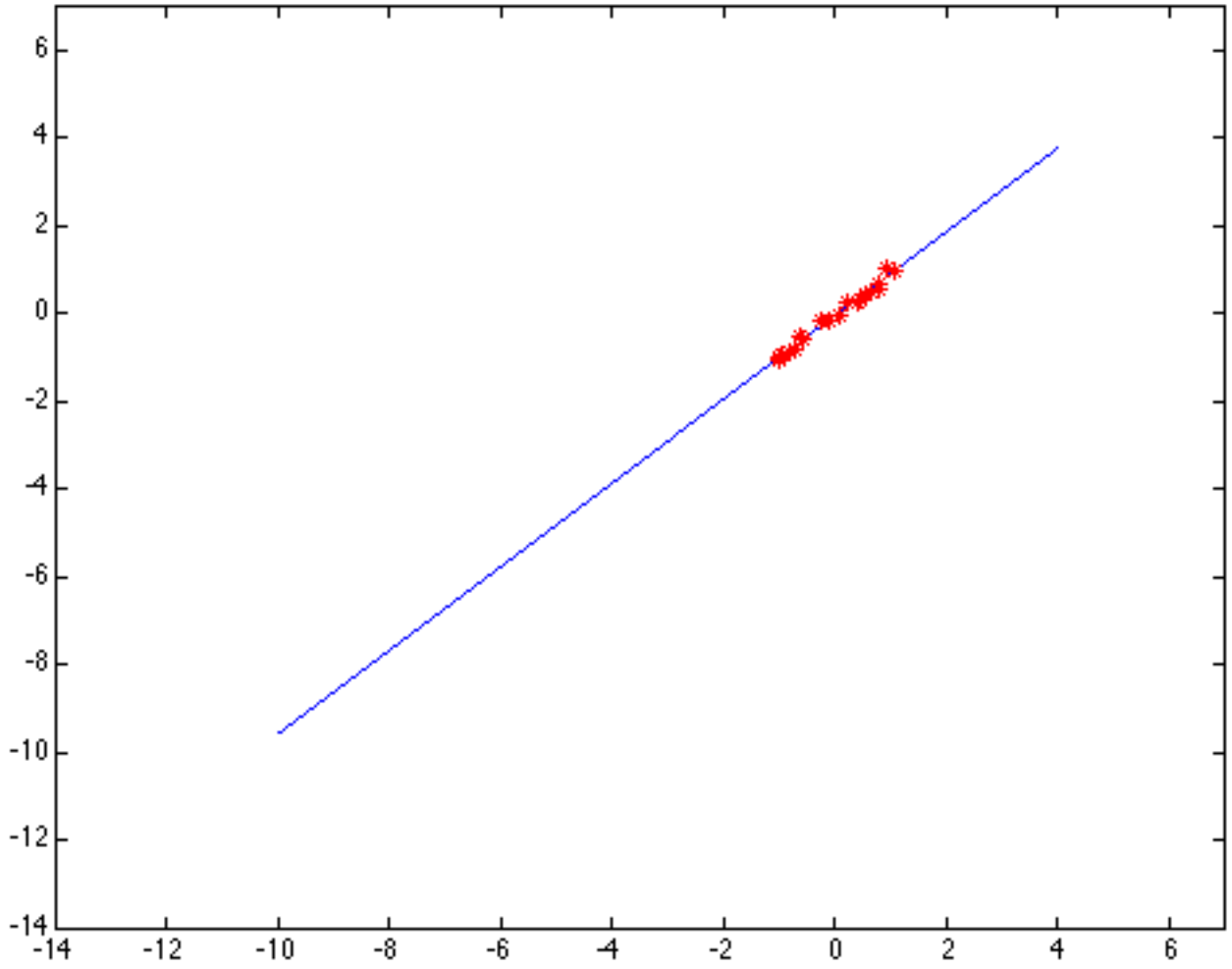
You want to find a single line that "explains" all of the points in your data, but data may be noisy!

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

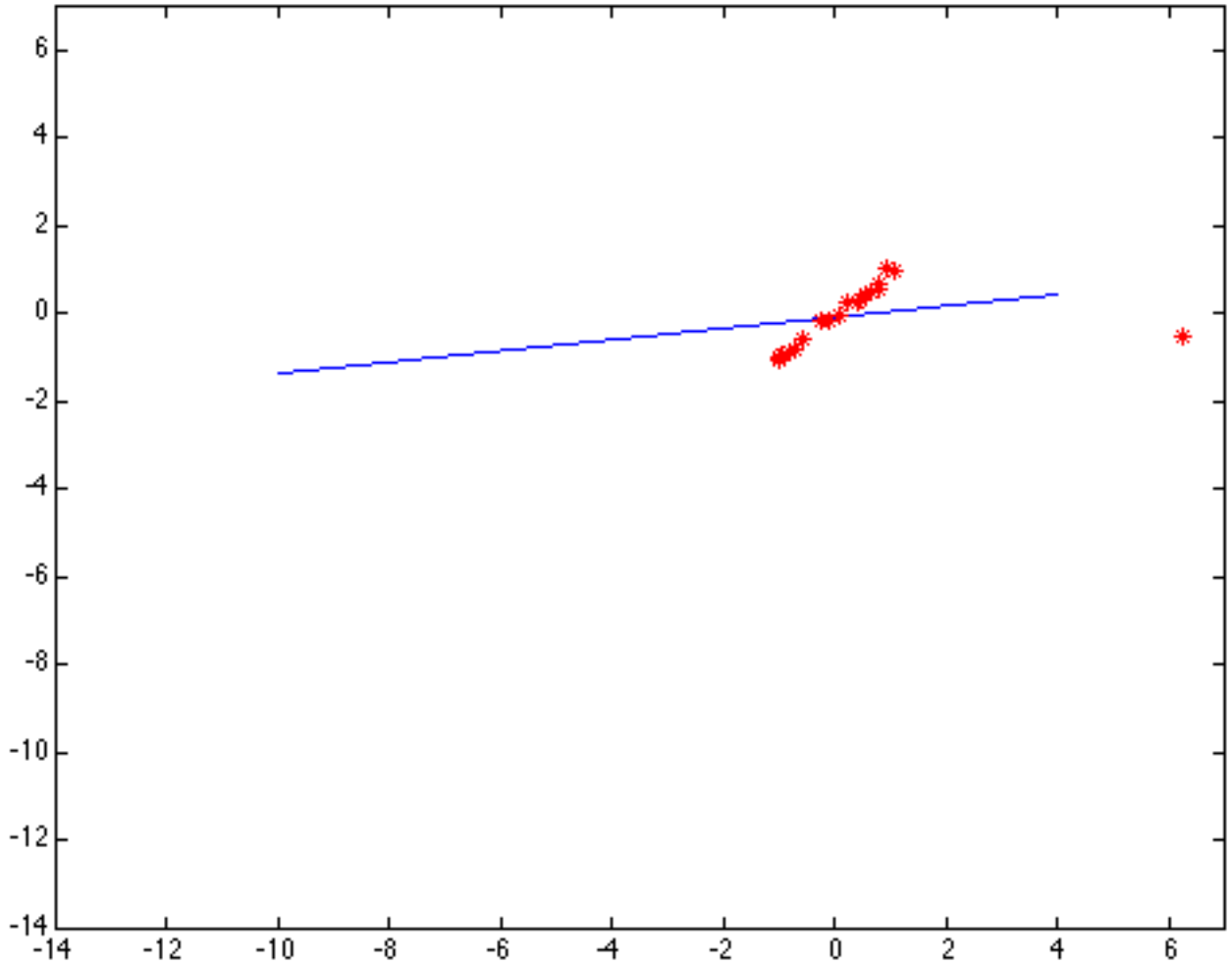
Matlab: `p = A \ y;`

Adapted from Svetlana Lazebnik

# Outliers affect least squares fit



# Outliers affect least squares fit

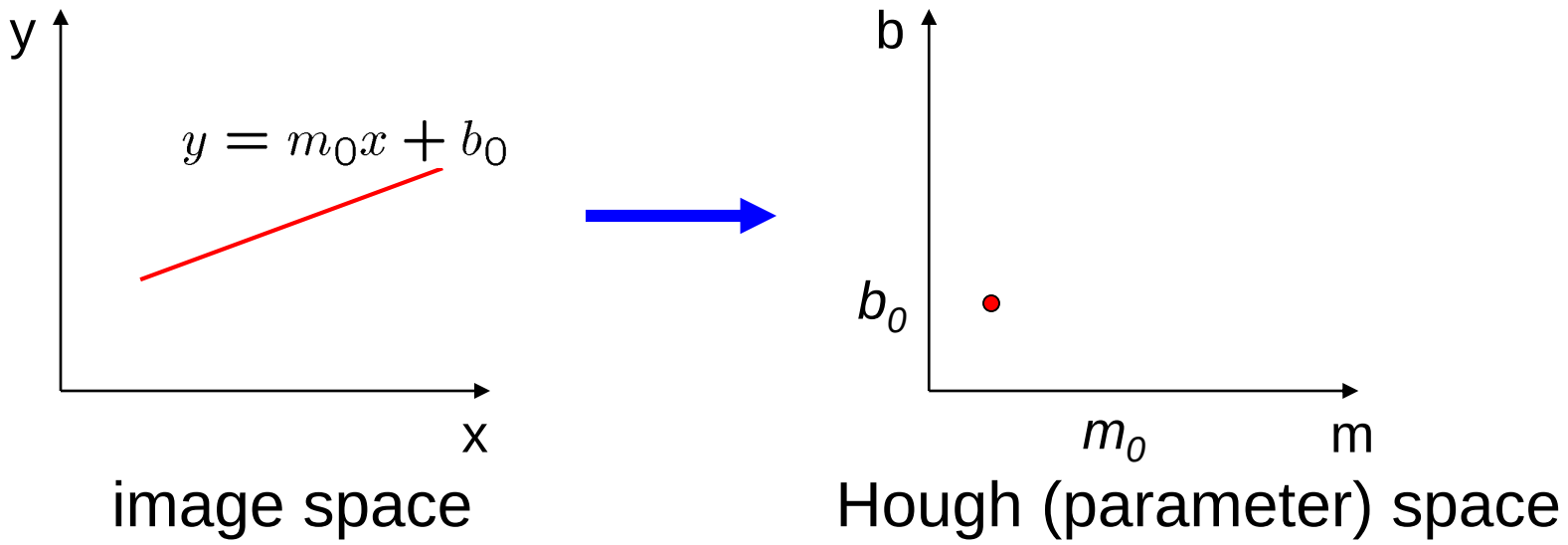


# Dealing with outliers: Voting

- **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.
- Two common techniques
  - Hough transform
  - RANSAC (hidden slides)

Adapted from Kristen Grauman

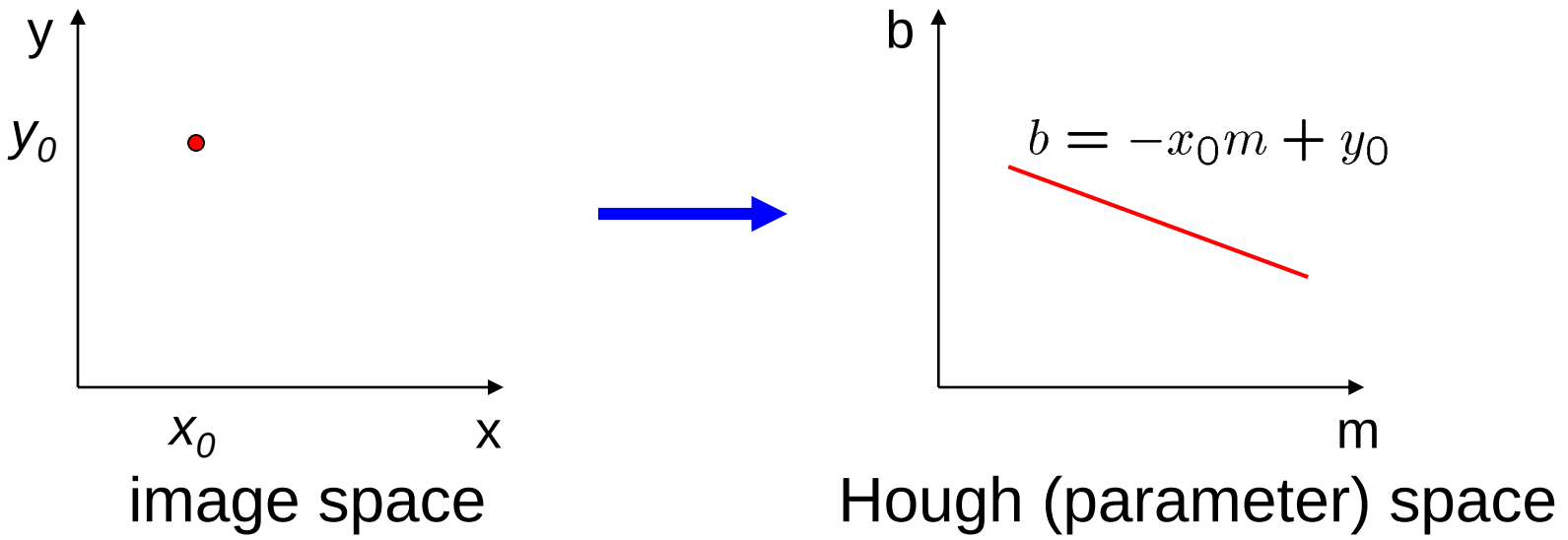
# Finding lines in an image: Hough space




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

Adapted from Steve Seitz

# Finding lines in an image: Hough space

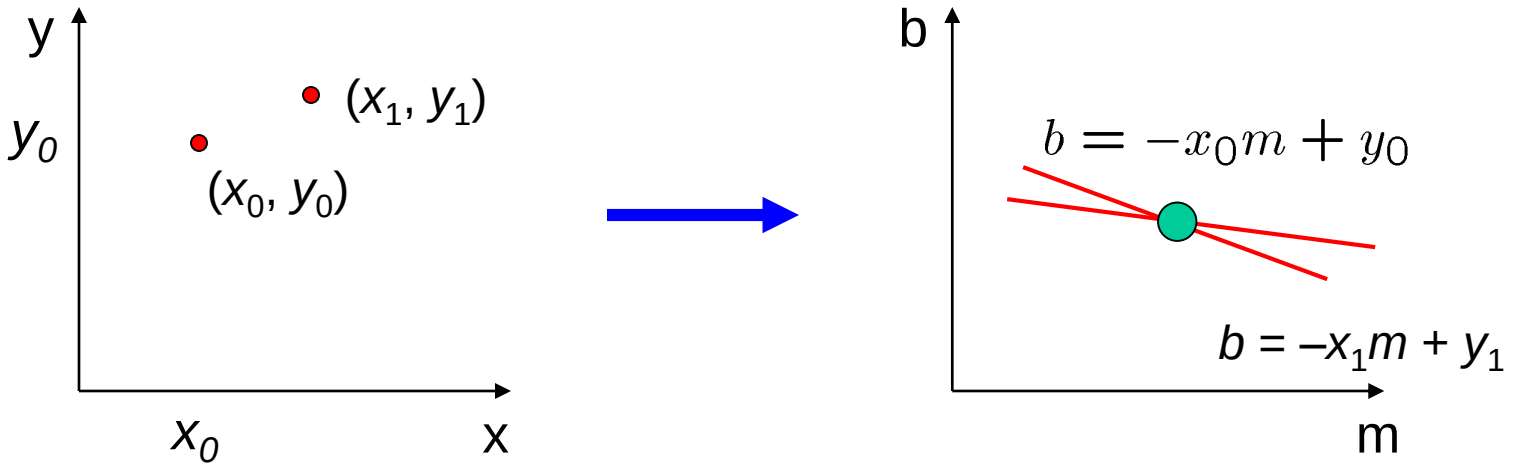



image space

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

# Finding lines in an image: Hough space

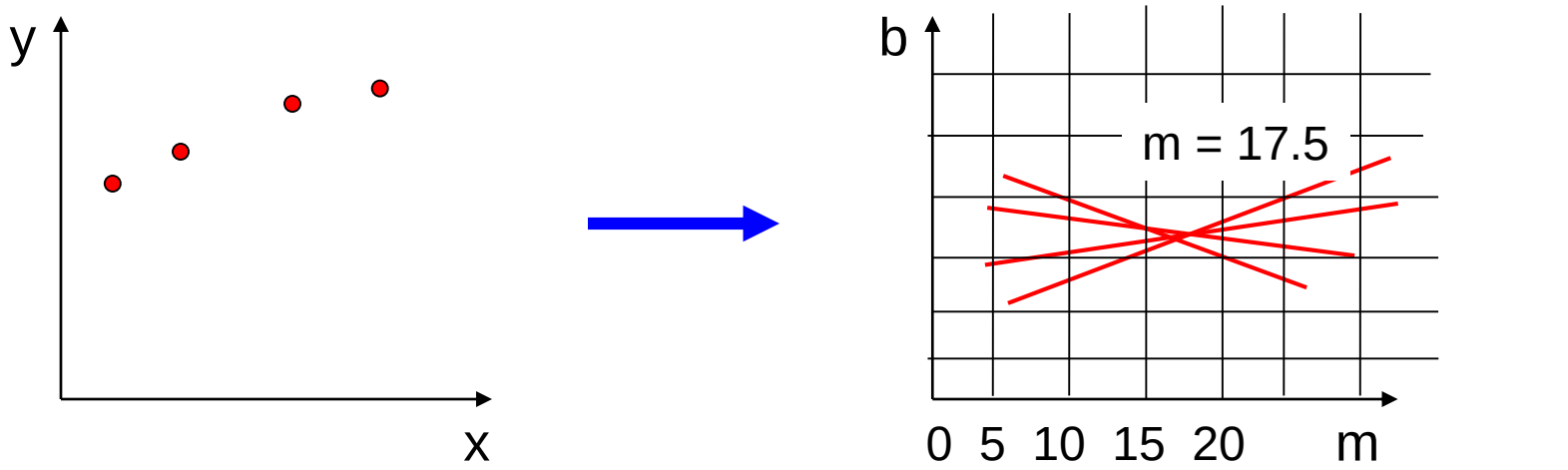


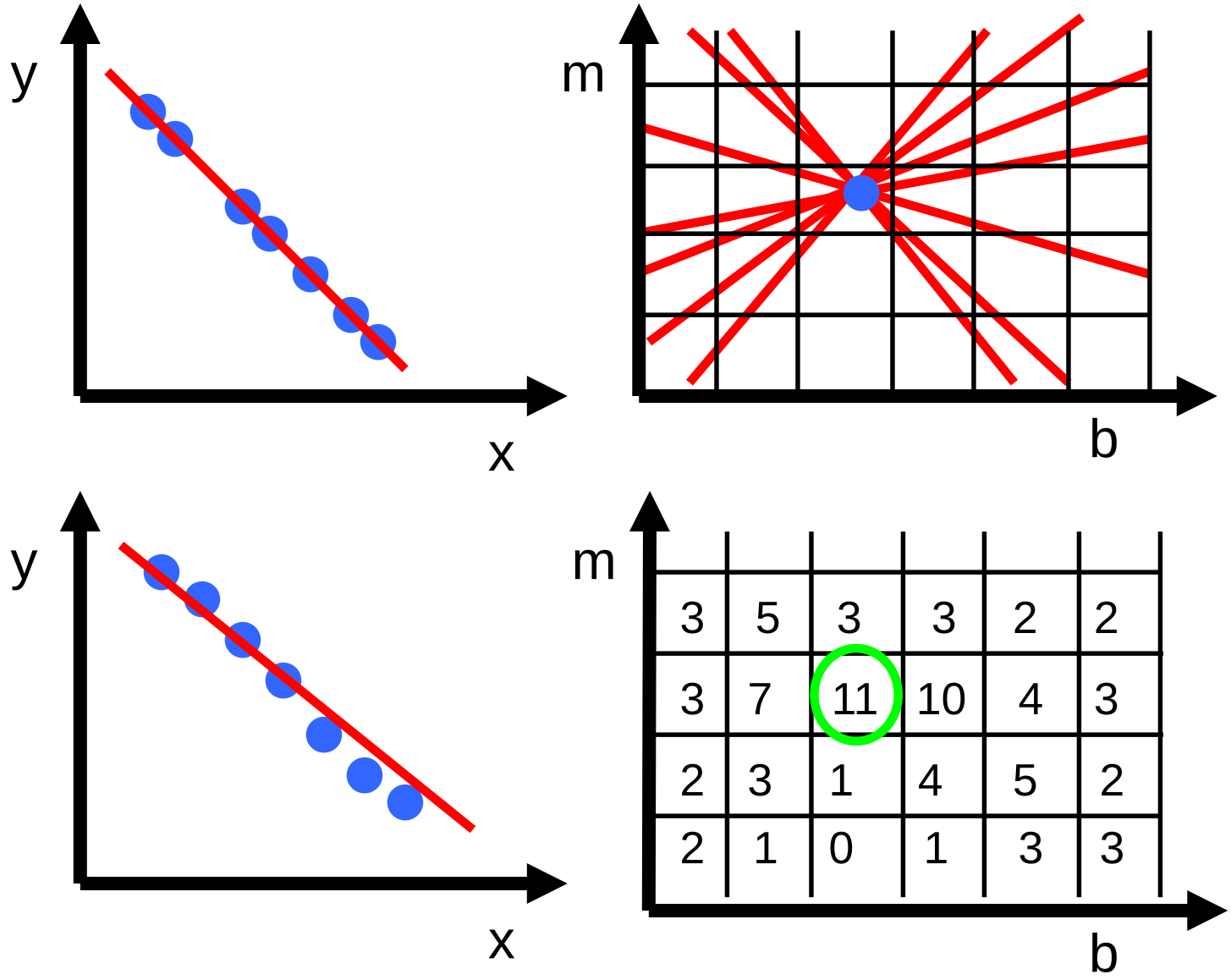

image space

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.

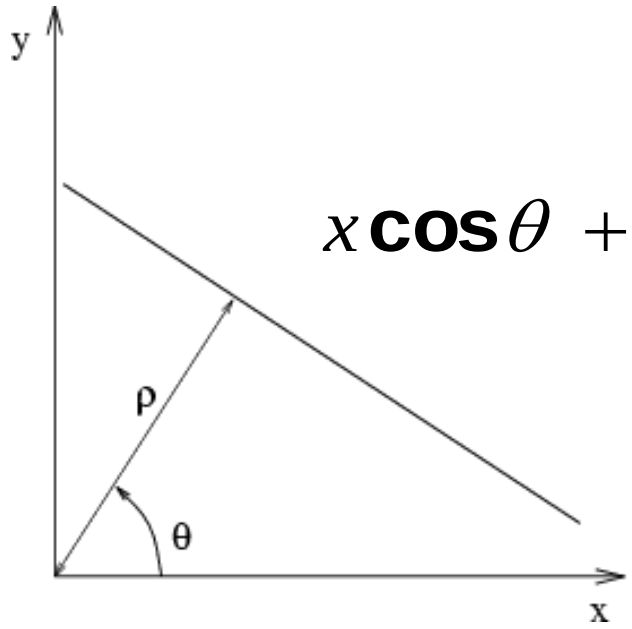
# Finding lines in an image: Hough space




Adapted from Silvio Savarese

# Parameter space representation

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

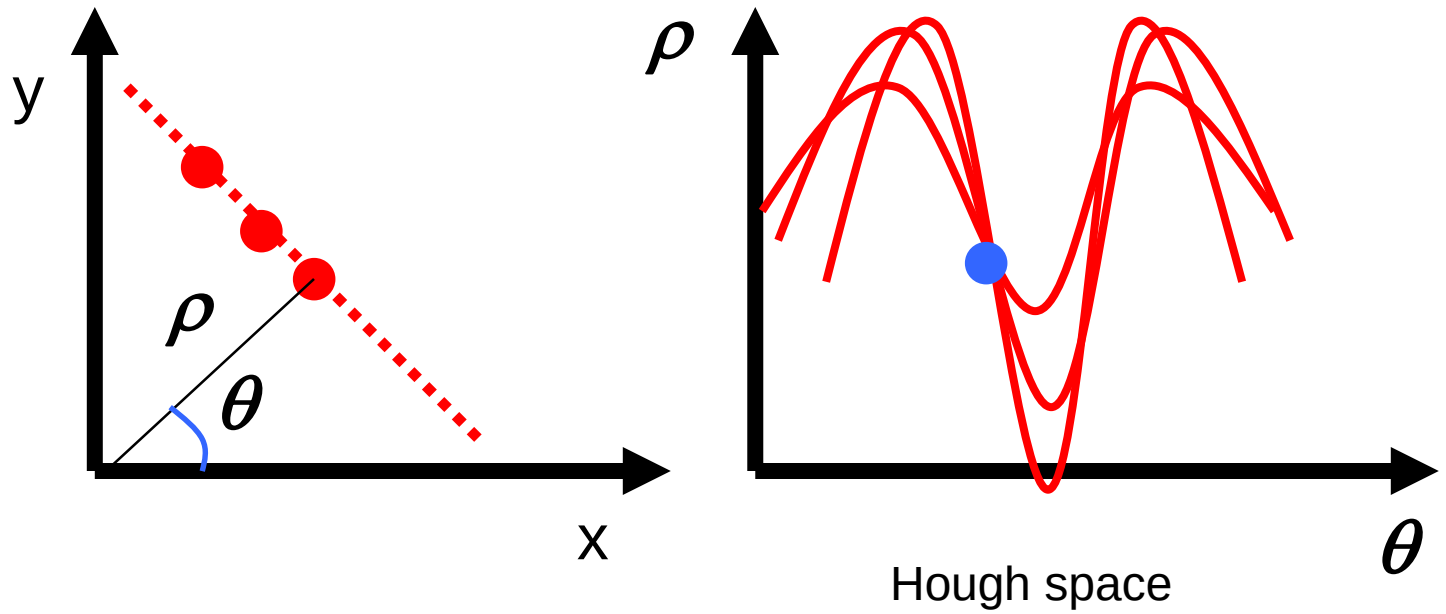



$$x\cos\theta + y\sin\theta = \rho$$

Each point (x,y) will add a sinusoid in the ($\theta$,$\rho$) parameter space

# Parameter space representation

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

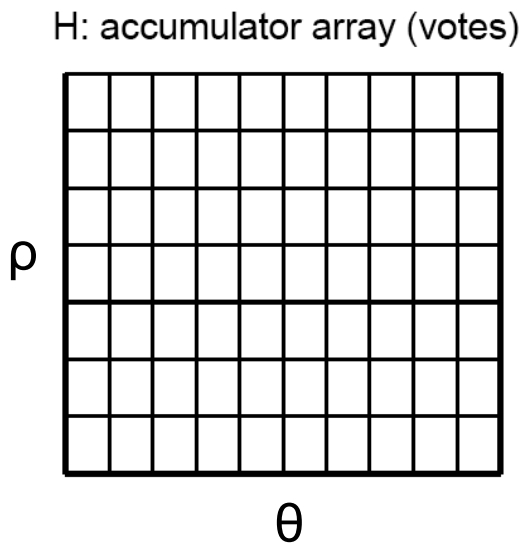



Each point (x,y) will add a sinusoid in the ($\theta$,$\rho$) parameter space

# Algorithm outline: Hough transform

- Initialize accumulator H to all zeros

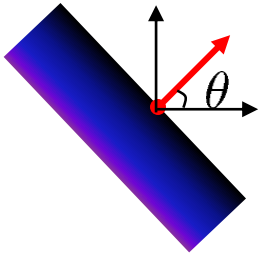
- For each edge point (x,y) in the image
  - For θ = 0 to 180
    - ρ = x cos θ + y sin θ
    - H(θ, ρ) = H(θ, ρ) + 1
  - end
- end




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

# 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!




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

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

- Modified Hough transform:

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

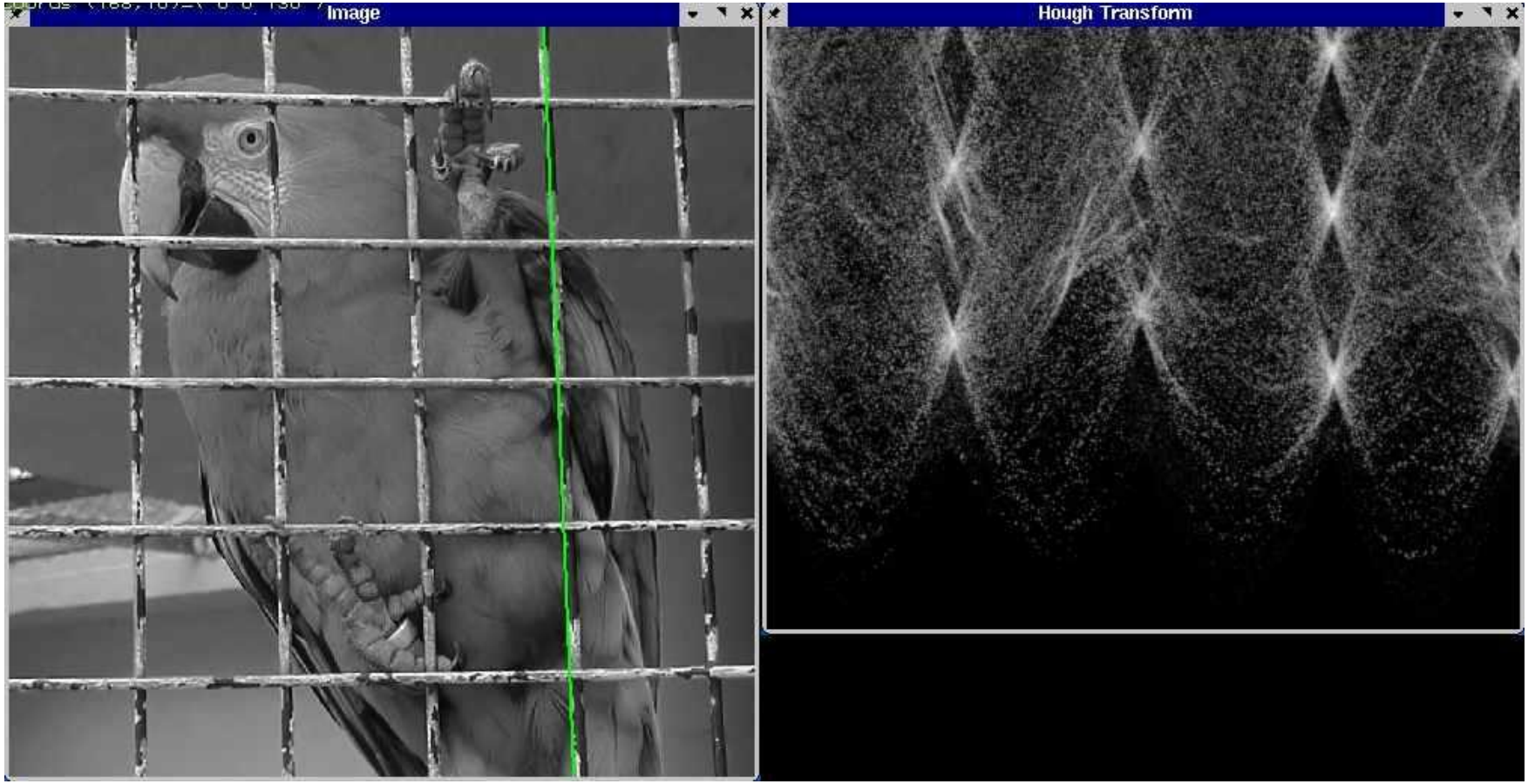
θ = gradient orientation at (x,y)

ρ = x cos θ + y sin θ

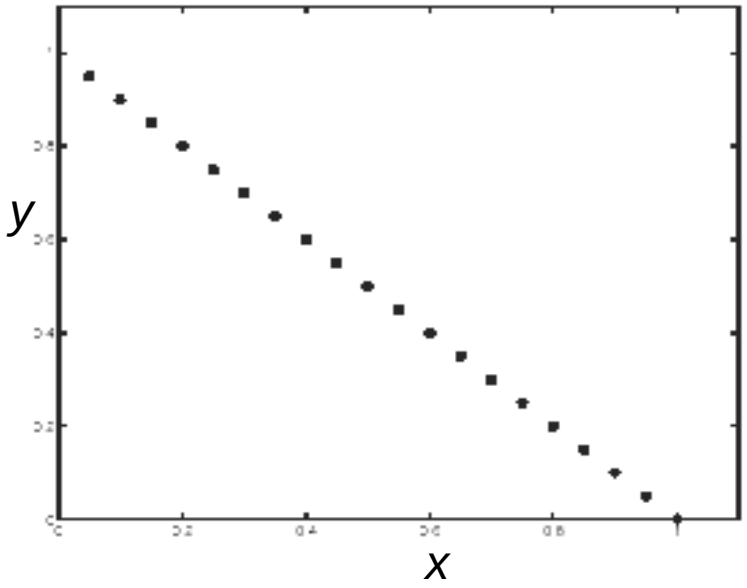
H(θ, ρ) = H(θ, ρ) + 1

end

# Hough transform example



# Impact of noise on Hough



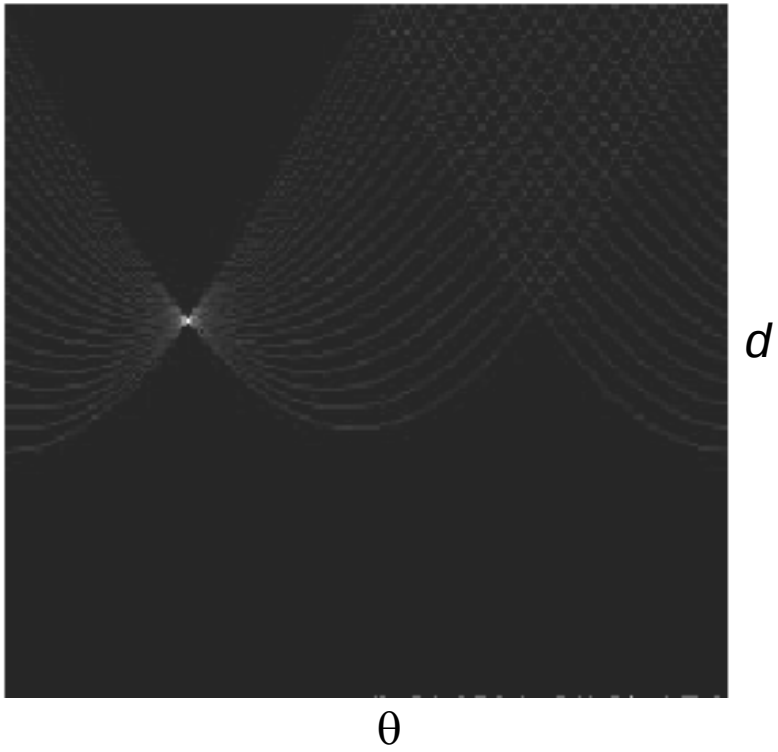




**Image space**
**edge coordinates**

**Votes**

# Impact of noise on Hough

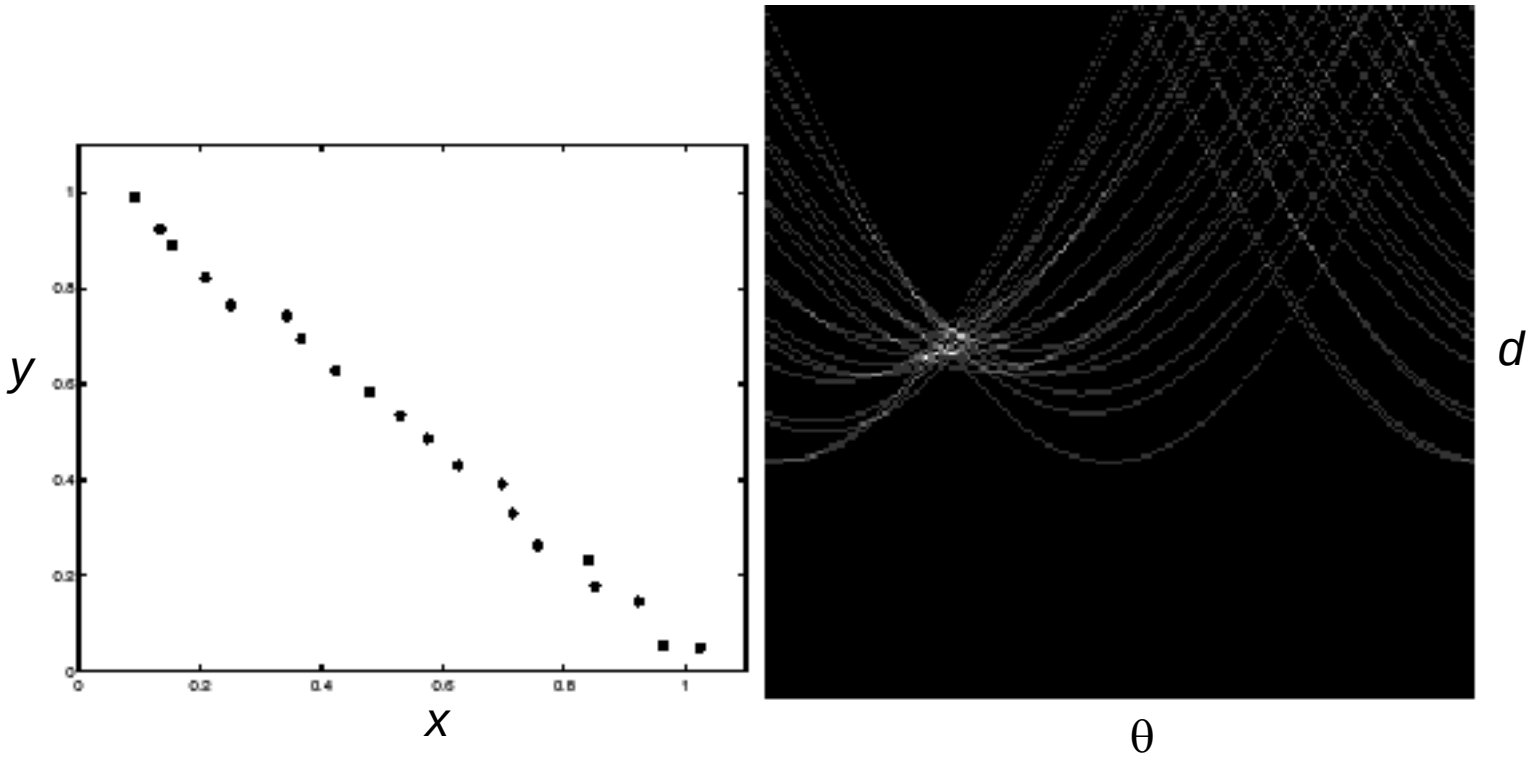


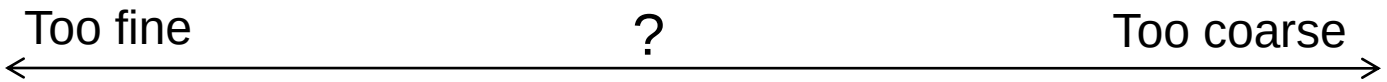

**Image space edge coordinates**

**Votes**

What difficulty does this present for an implementation?

# Voting: practical tips

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




  - **Too coarse:** large votes obtained when too many different lines correspond to a single bucket
  - **Too fine:** miss lines because points that are not exactly collinear cast votes for different buckets

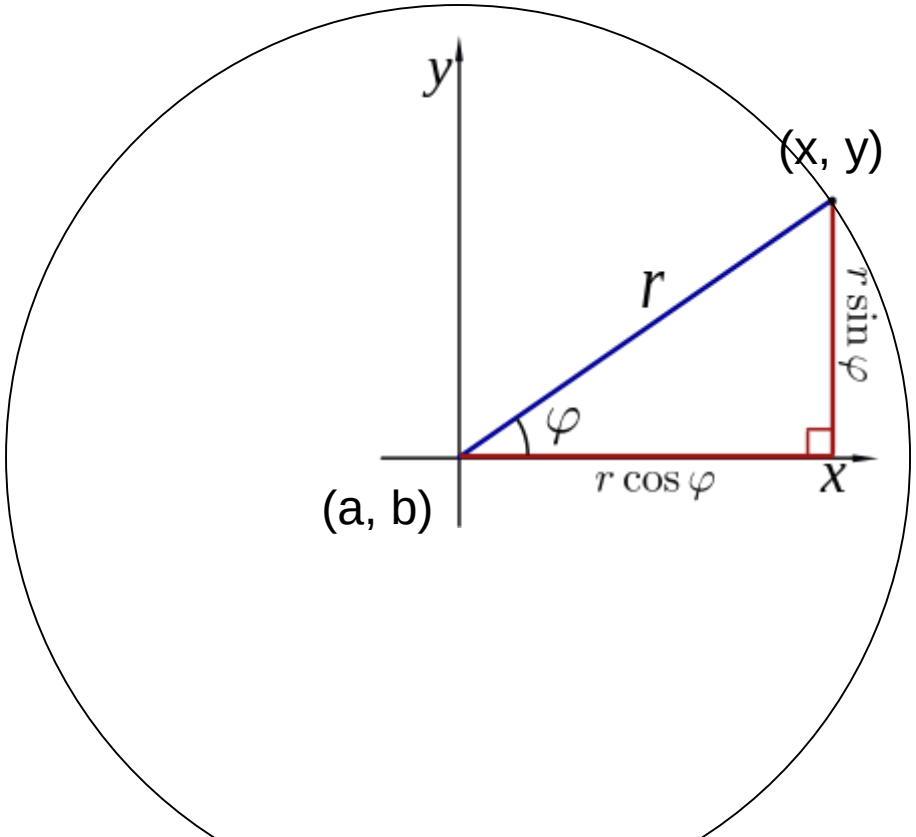
- Vote for neighbors (smoothing in accumulator array)
- Use direction of edge to reduce parameters by 1
- To read back which points voted for “winning” peaks, keep tags on the votes

# 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

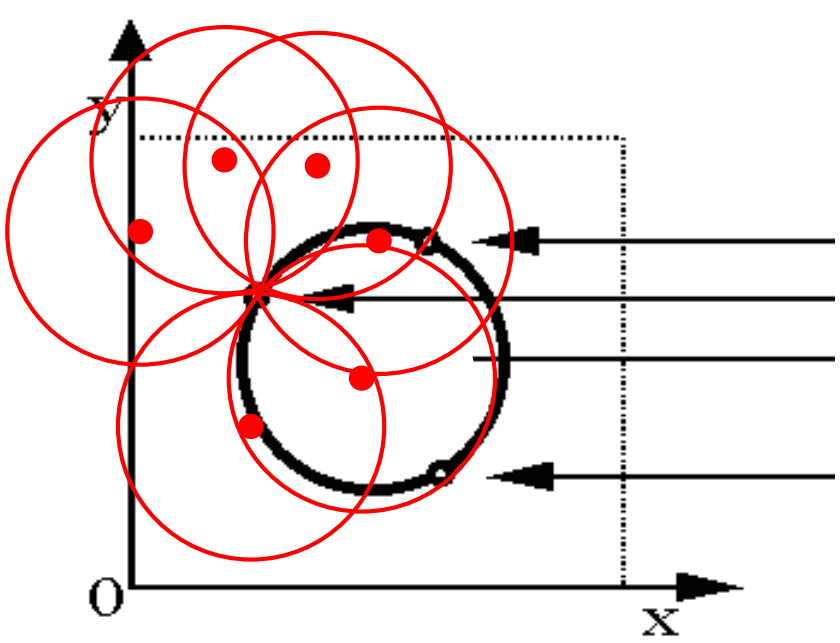
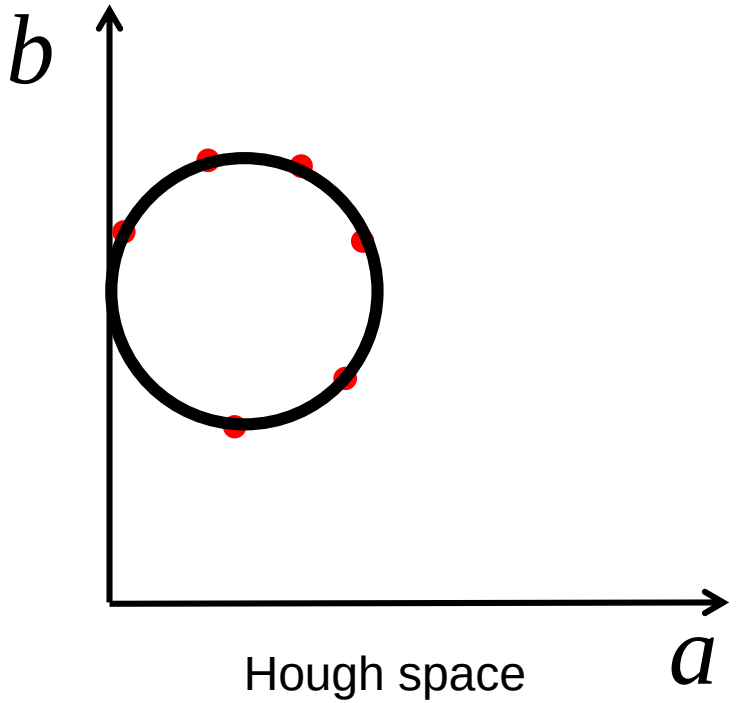



Image space



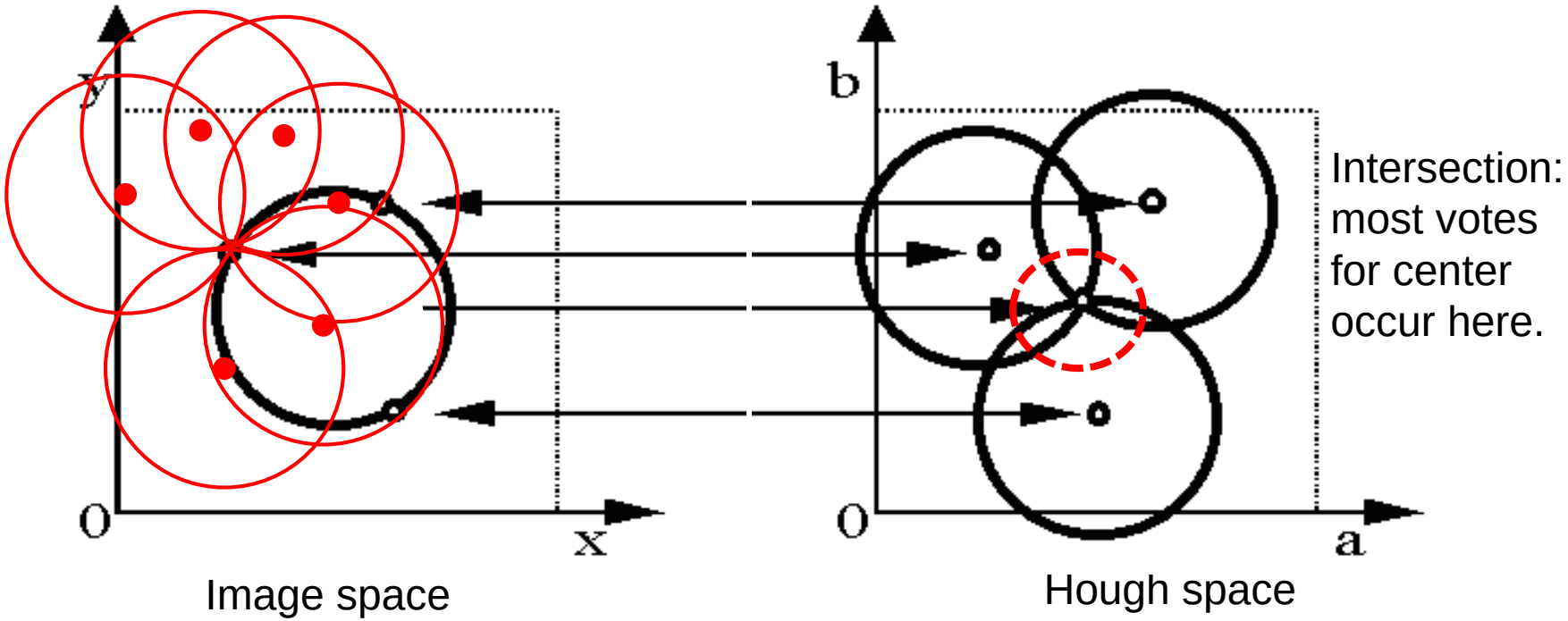

Hough space

# 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

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

For every edge pixel ($x$,$y$) :

  For each possible radius value $r$:

    For each possible gradient direction $\theta$:

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

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

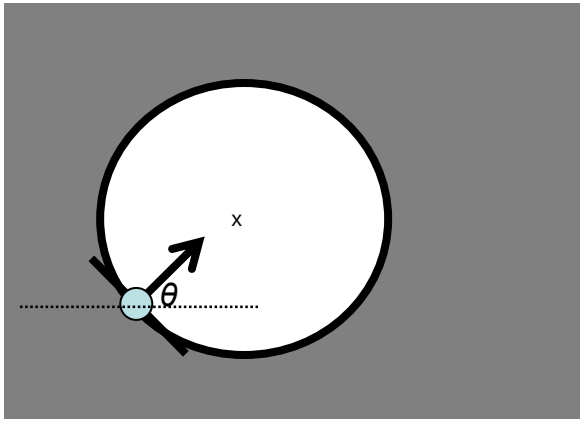
      $b = y - r \sin(\theta)$ // row

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

    end

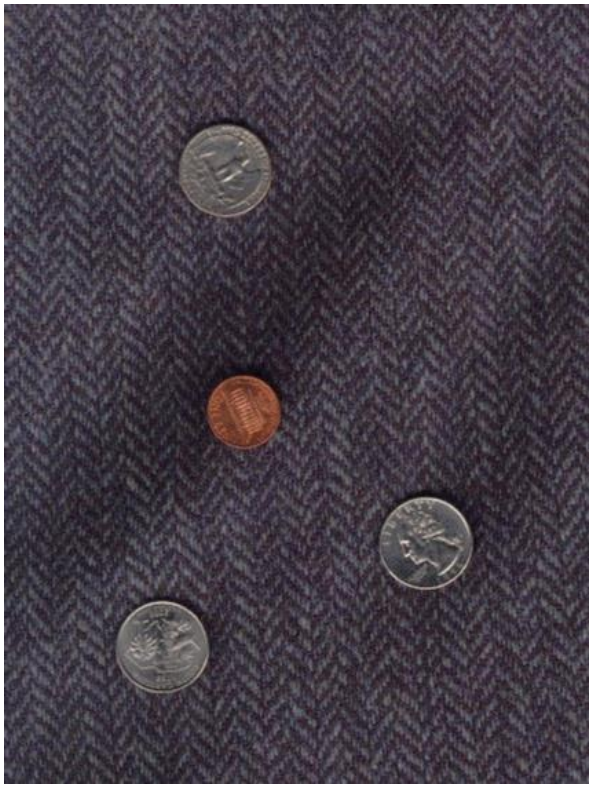
  end

end

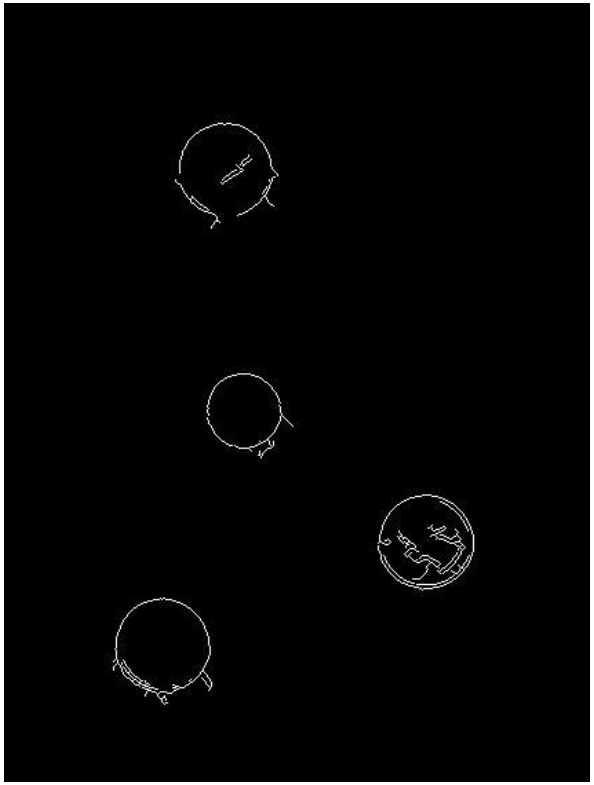


# Example: detecting circles with Hough

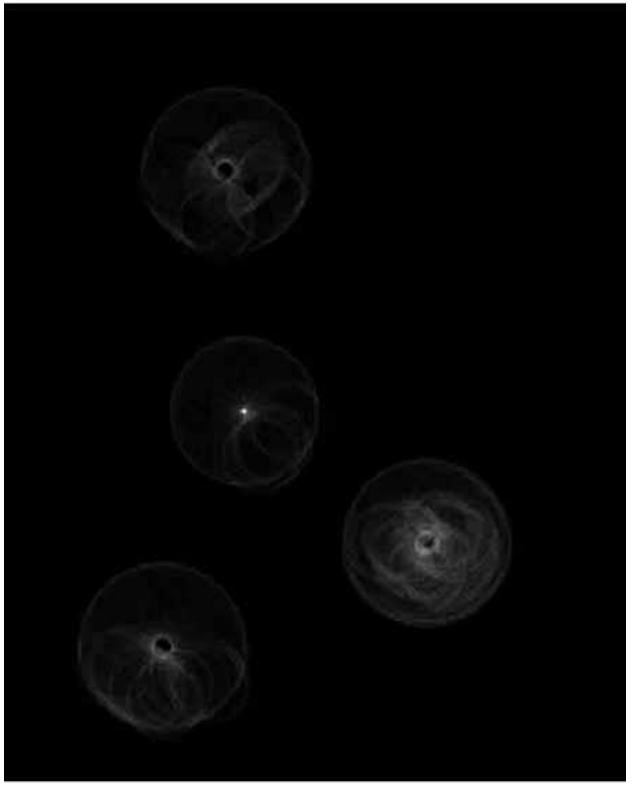
Original



Edges



Votes: Penny

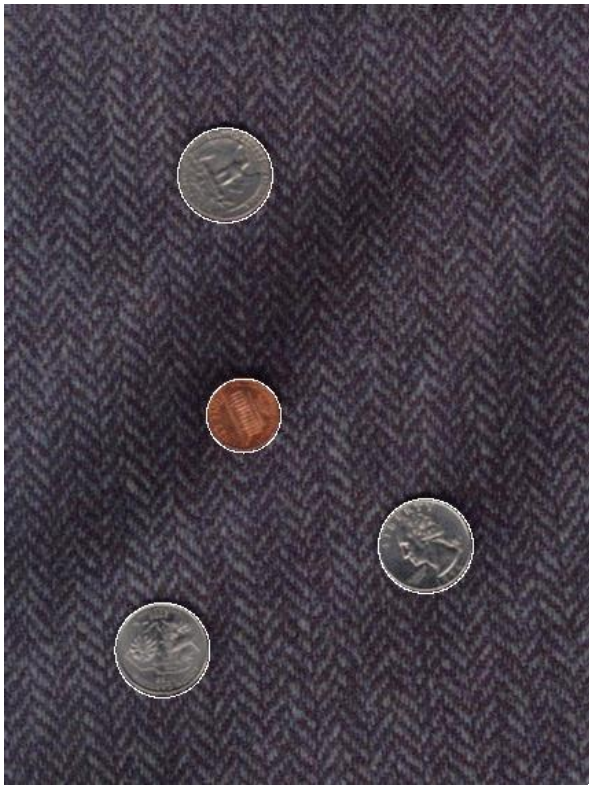


Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

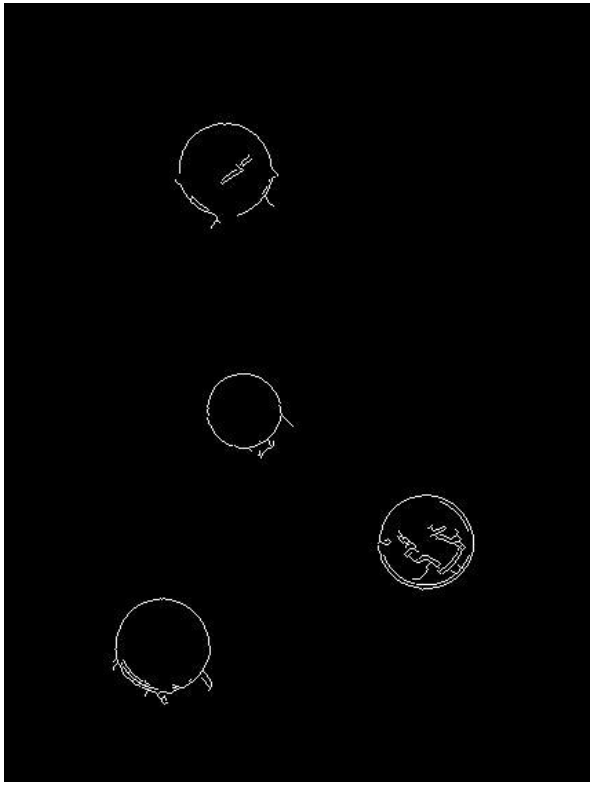
Kristen Grauman, images from Vivek Kwatra

# Example: detecting circles with Hough

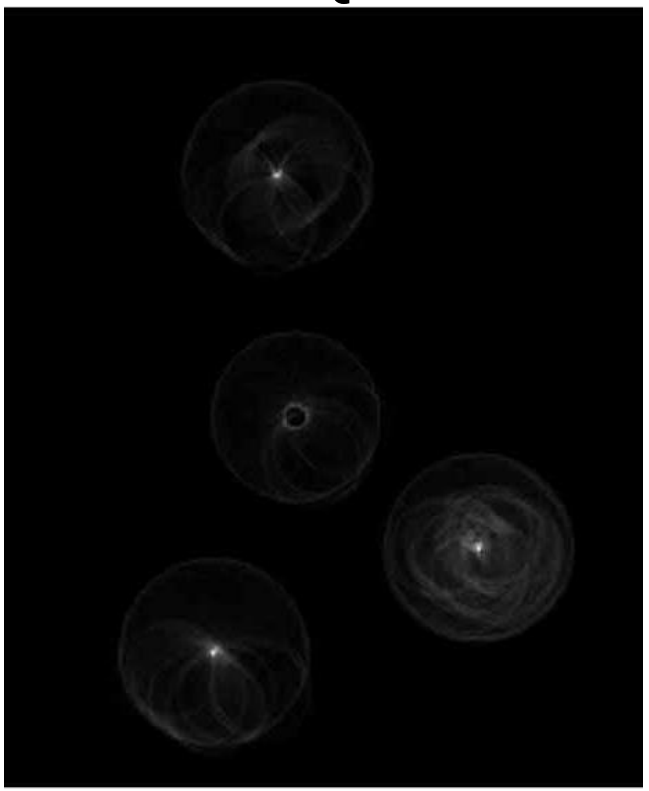
Original



Edges



Votes: Quarter



Combined detections

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

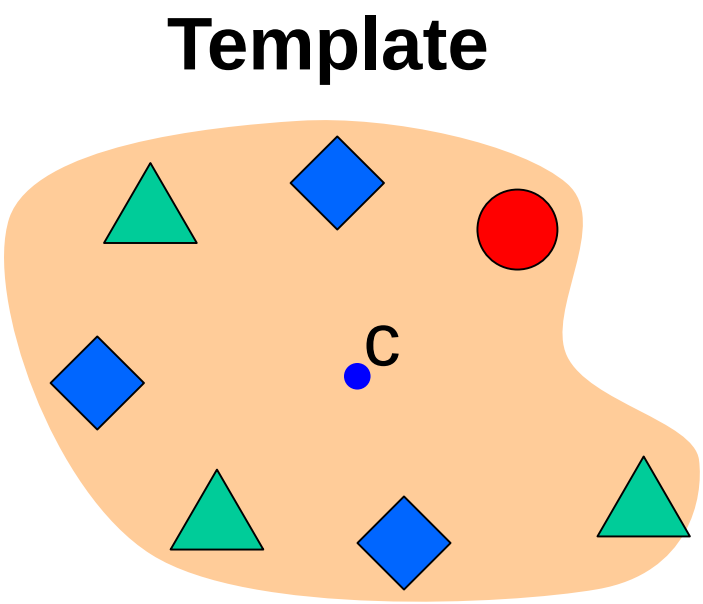
## Cons

- 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

Adapted from Kristen Grauman

# Generalized Hough transform

- We want to find a template defined by its reference point (center) and several distinct types of landmark points in stable spatial configuration



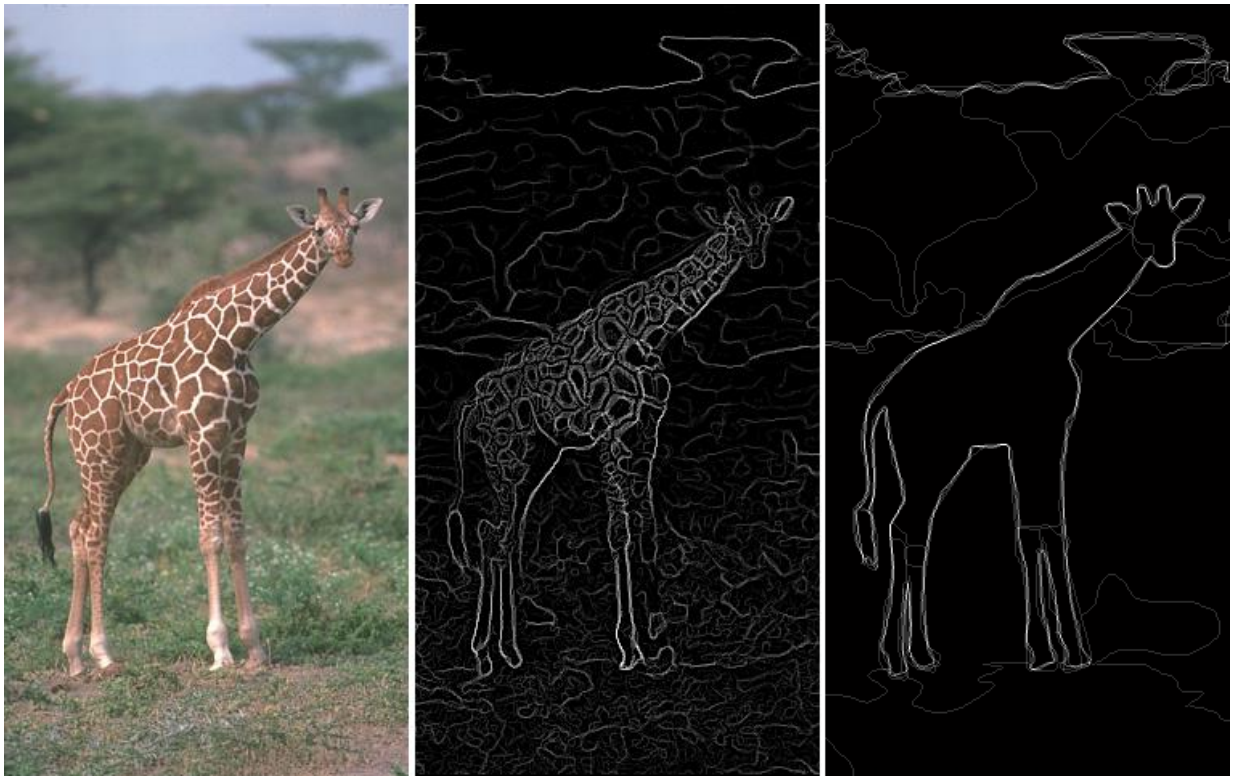

Triangle, circle, diamond: some *type* of visual token, e.g. feature or edge point

Adapted from Svetlana Lazebnik

# Plan for this lecture

- Group pixels into:
  - Edges: Extract gradients and threshold
  - Lines: Find which edge points are collinear or belong to another shape
  - Segments: Find which pixels form a consistent region, e.g. via clustering
- Transform pixels:
  - Find relationships between multiple views of the same world point
  - *Both parts rely on finding geometric relationships between pixels*

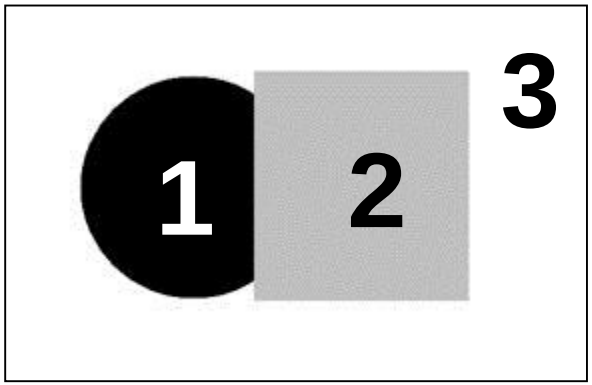
# Edges vs Segments



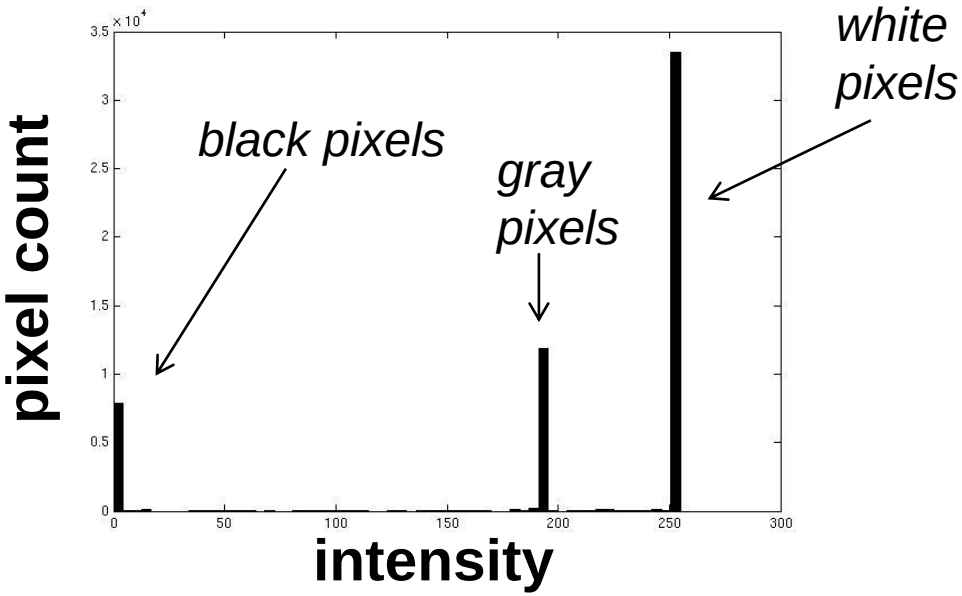
- Edges: More low-level; don’t need to be closed
- Segments: Ideally one segment for each semantic group/object; should include closed contours

Figure adapted from J. Hays

# Image segmentation: toy example

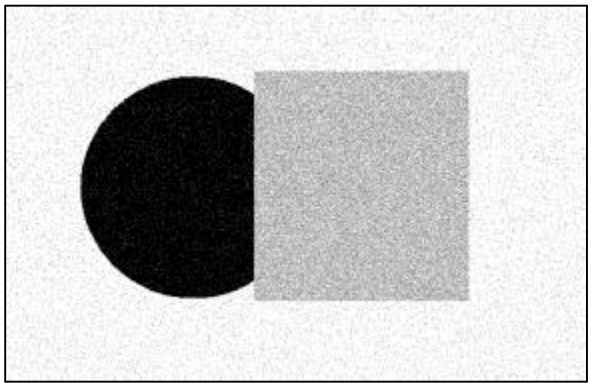



**input image**

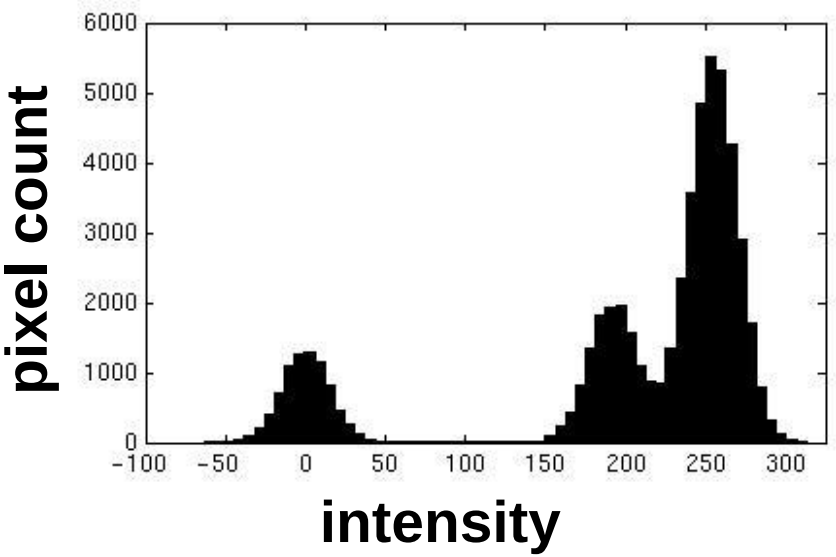



- These intensities define the three groups.
- We could label every pixel in the image according to which of these primary intensities it is.
  - i.e., *segment* the image based on the intensity feature.
- What if the image isn't quite so simple?

Source: K. Grauman

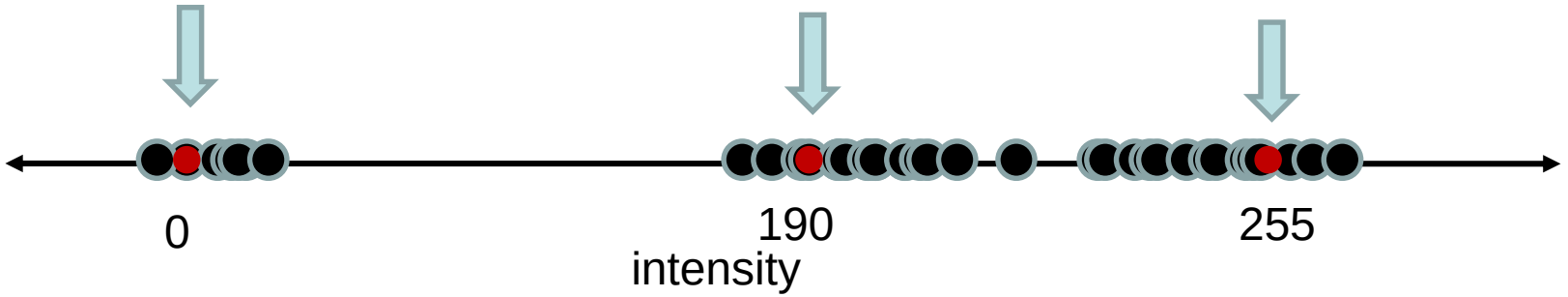


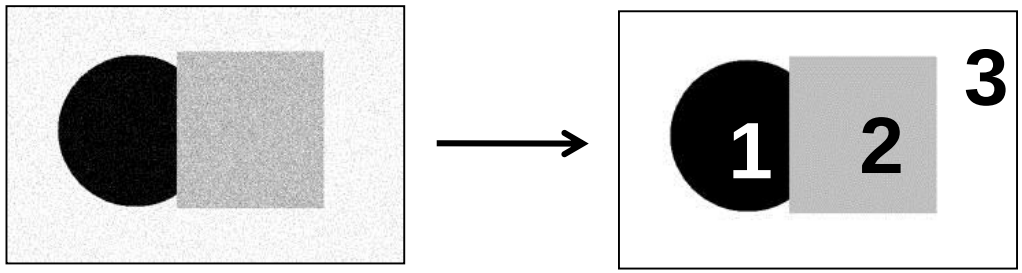
**input image**




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

Source: K. Grauman



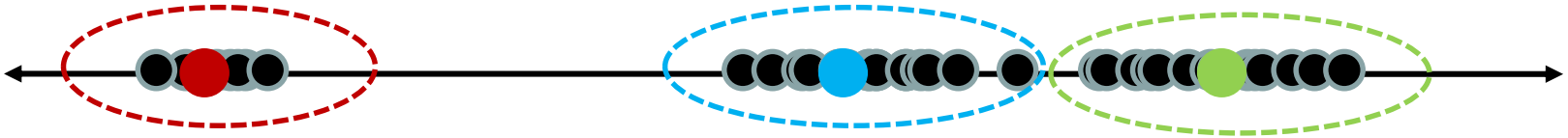

- Goal: choose three “centers” as the representative intensities, and label every pixel according to which of these centers it is nearest to.
- Best cluster centers are those that minimize *sum of squared differences* (SSD) between all points and their nearest cluster center $c_i$:

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

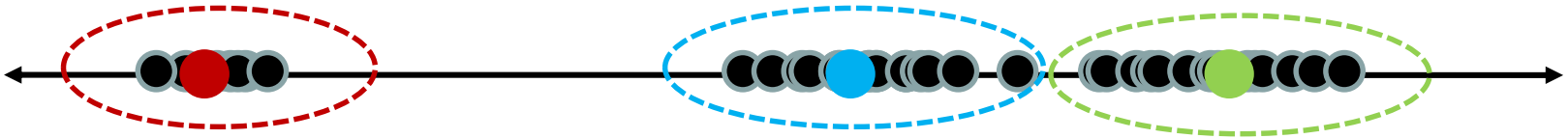
Source: K. Grauman

# 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.

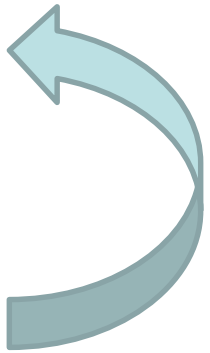


Source: K. Grauman

# K-means clustering

- Basic idea: randomly initialize the *k* cluster centers, and iterate between the two steps we just saw.

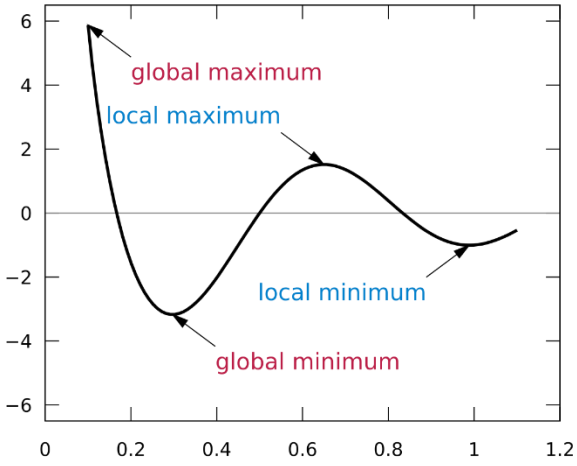
  1. *Randomly* initialize the cluster centers, $c_1$, ..., $c_K$
  2. *Given cluster centers*, determine points in each cluster
     - For each point p, find the closest $c_i$. Put p into cluster i
  3. *Given points in each cluster*, solve for $c_i$
     - Set $c_i$ to be the mean of points in cluster i
  4. If $c_i$ have changed, repeat Step 2



Properties

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

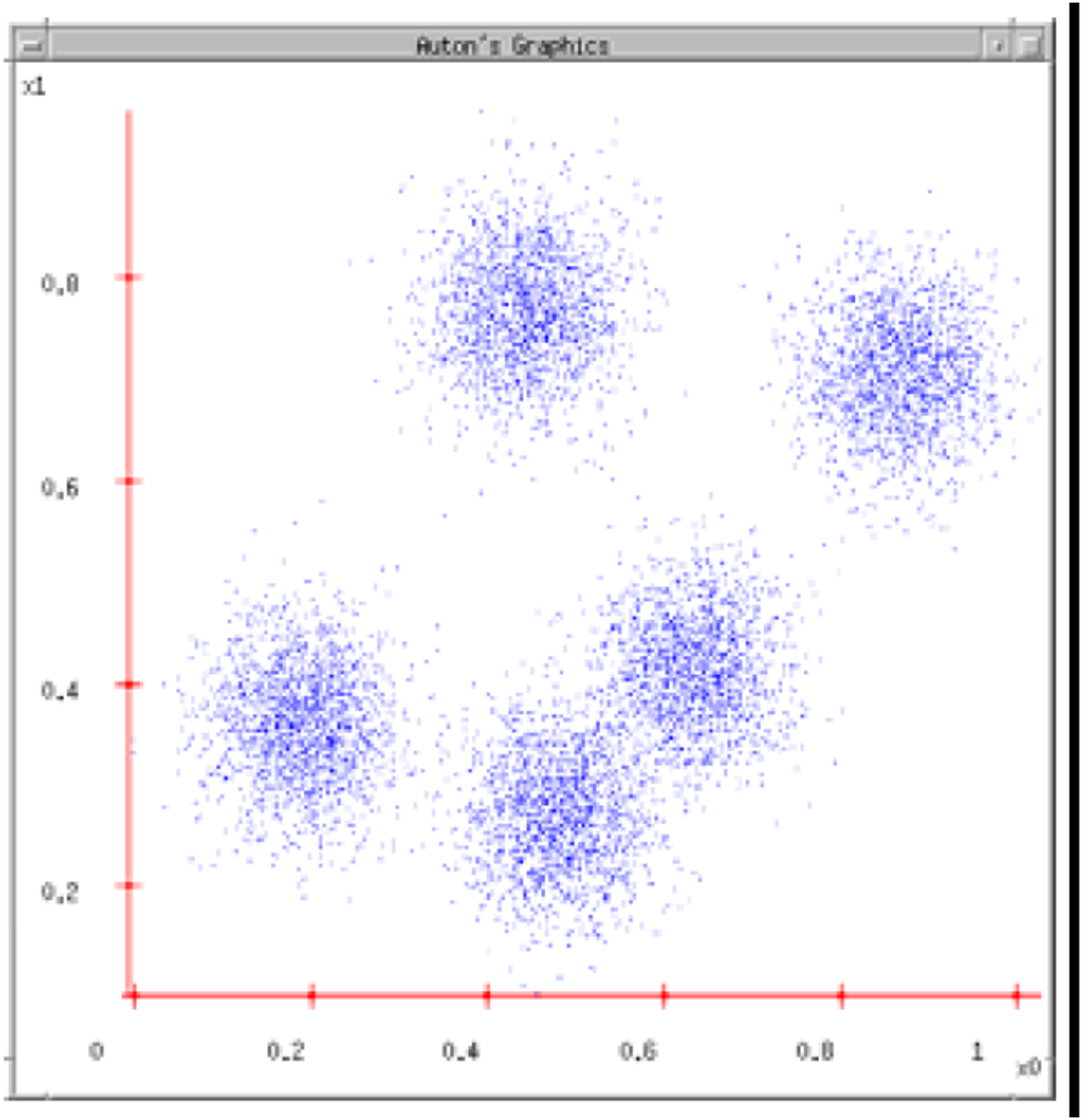
$$\sum_{\text{clusters } i} \quad \sum_{\text{points p in cluster } i} \|p - c_i\|^2$$




Slide: Steve Seitz, image: Wikipedia

# K-means

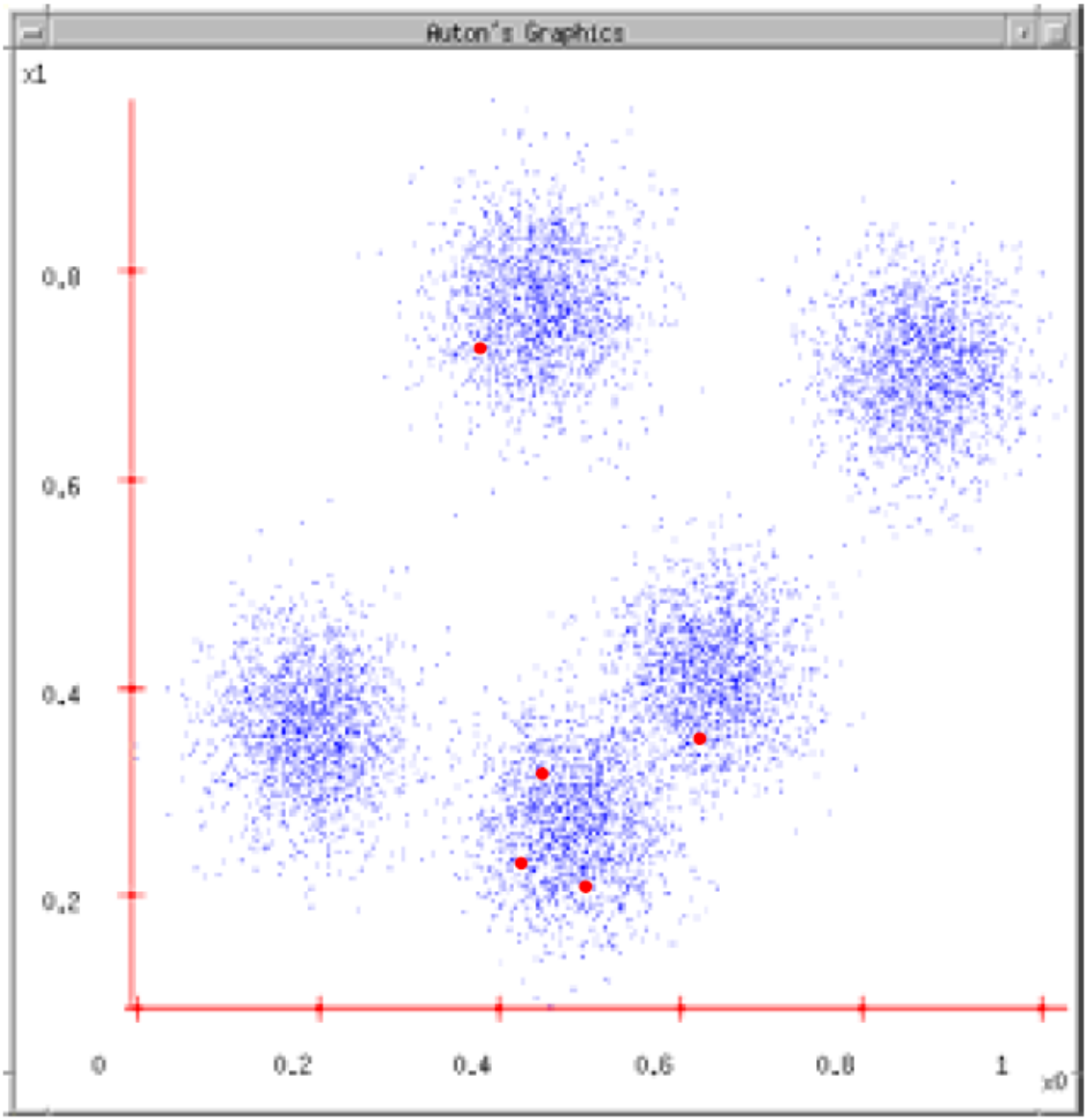
1. Ask user how many clusters they'd like. *(e.g. k=5)*




Source: A. Moore

# K-means

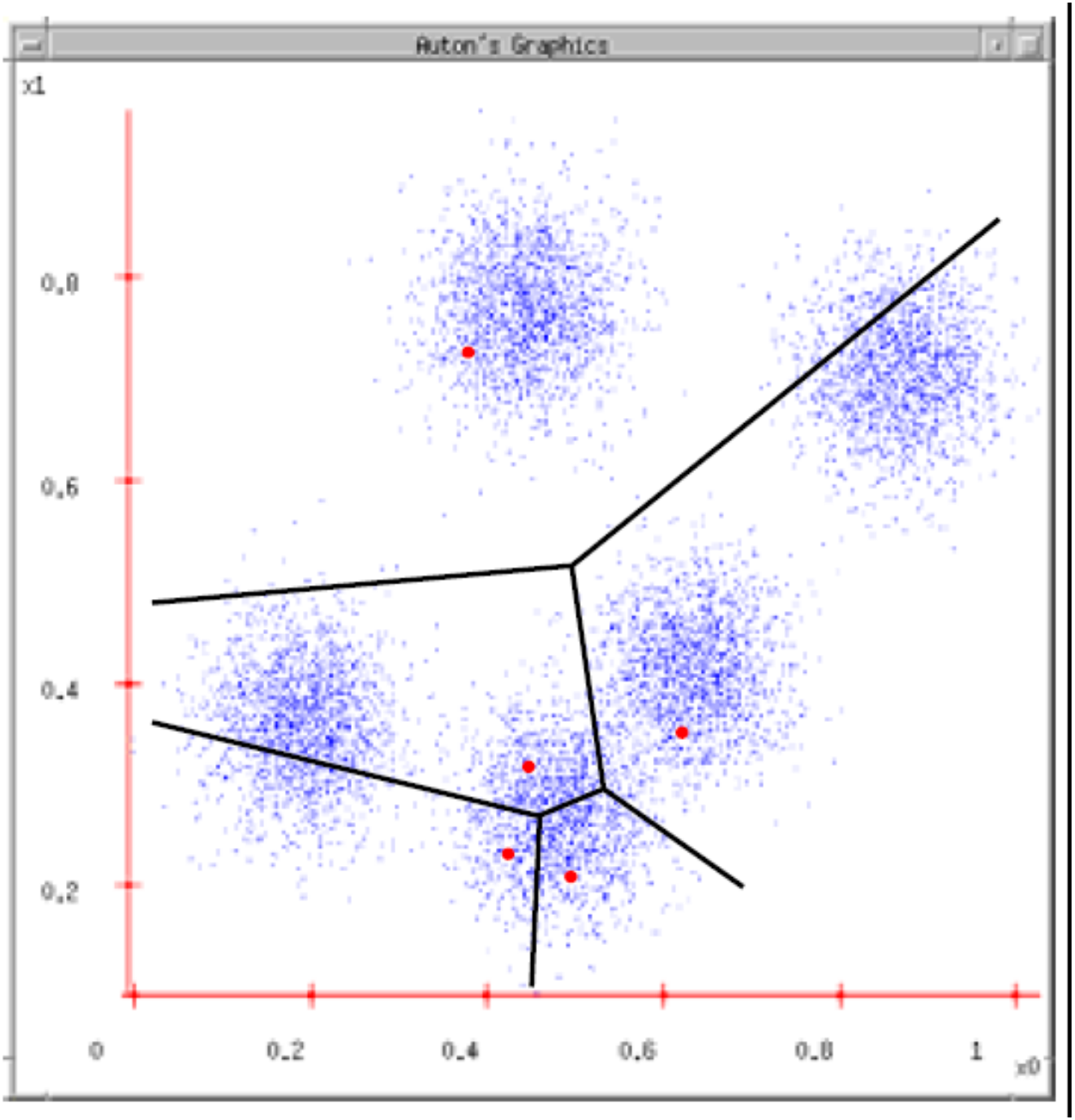
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations




Source: A. Moore

# K-means

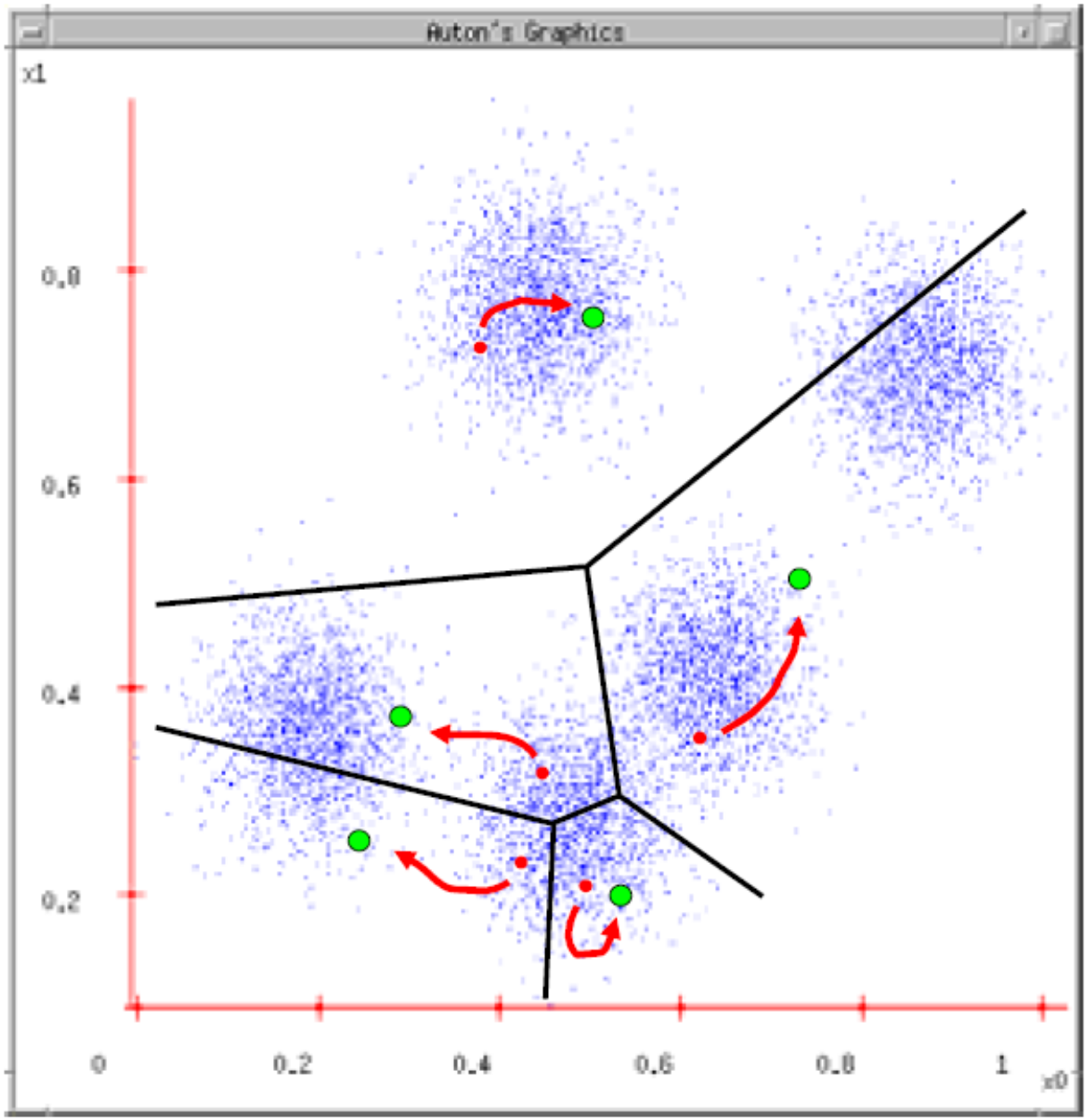
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints)




Source: A. Moore

# K-means

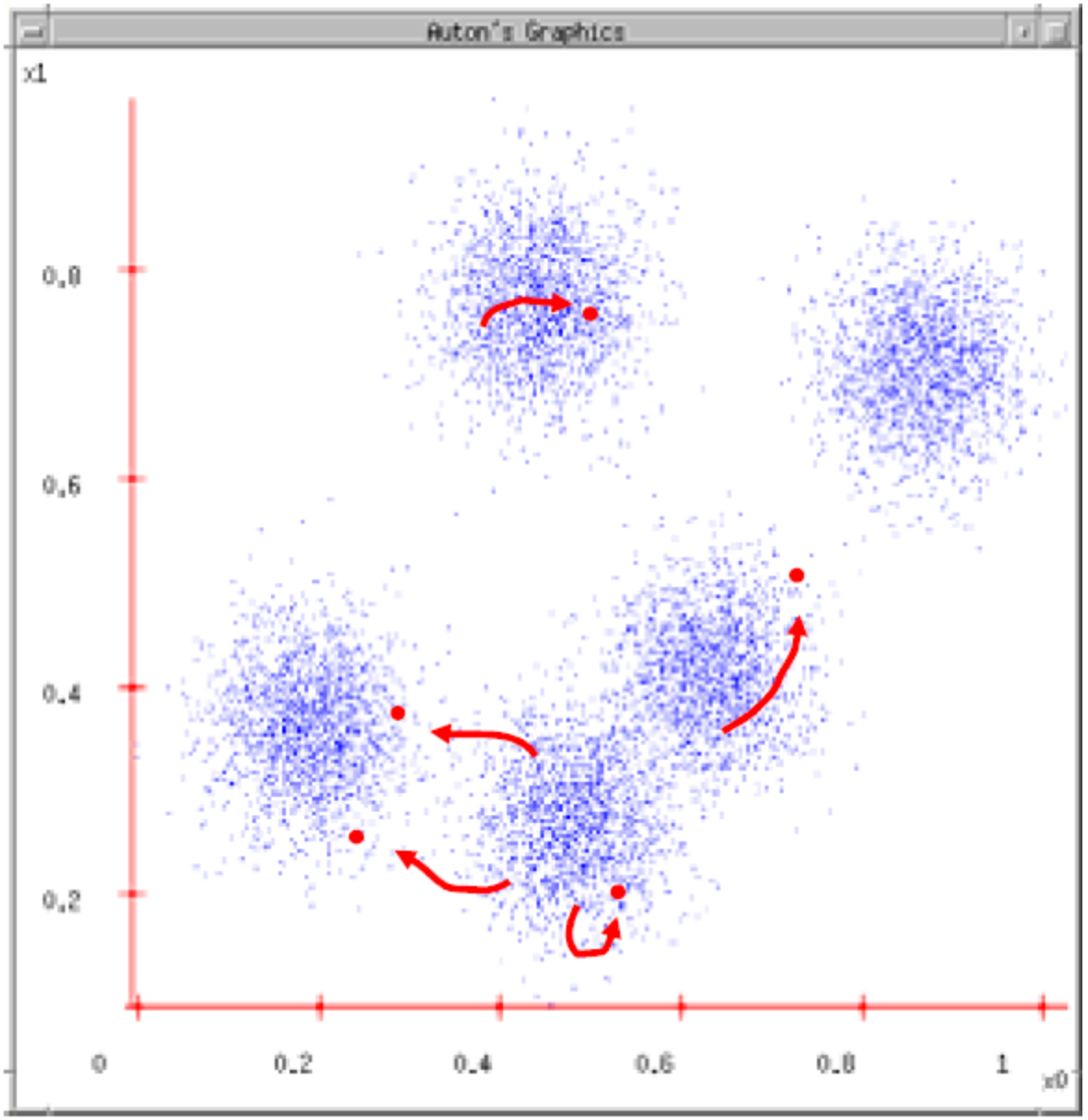
1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns




Source: A. Moore

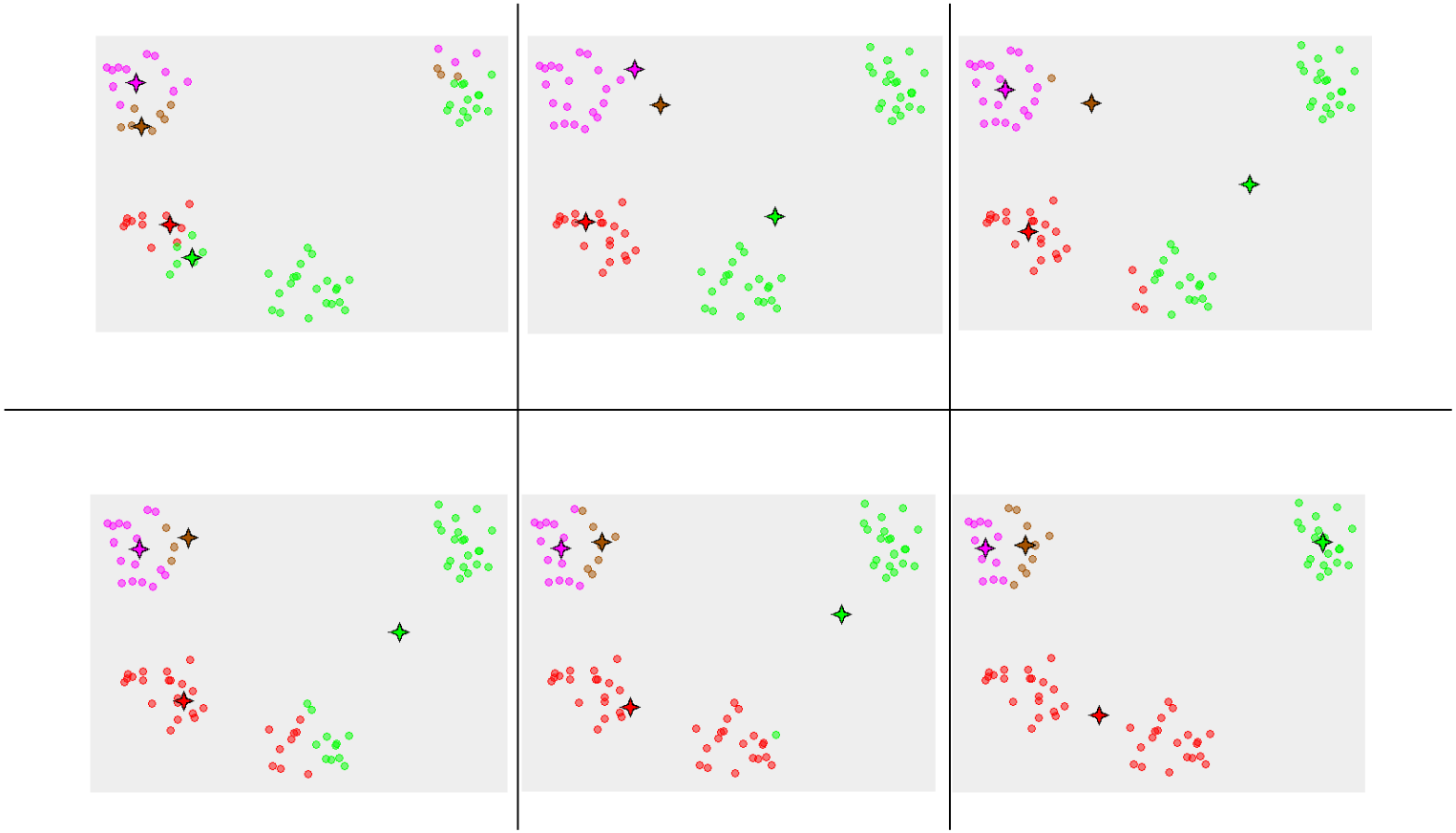
# K-means

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!




Source: A. Moore

# K-means converges to a local minimum



How can I try to fix this problem?

Adapted from James Hays

# K-means: pros and cons

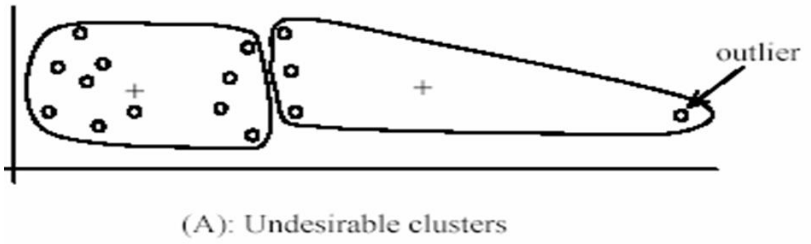
## Pros

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

## Cons/issues

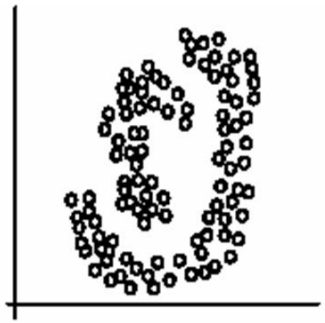
- Setting k?
  - One way: silhouette coefficient
- Sensitive to initial centers
  - Use heuristics or output of another method
  - Try different initializations
- Sensitive to outliers
- Detects spherical clusters

ALTERNATIVES?

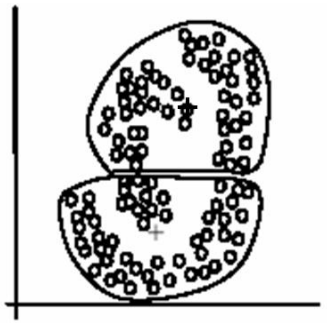



(A): Undesirable clusters


(B): Ideal clusters



(A): Two natural clusters



(B): *k*-means clusters

Adapted from K. Grauman

# Segmentation as clustering

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

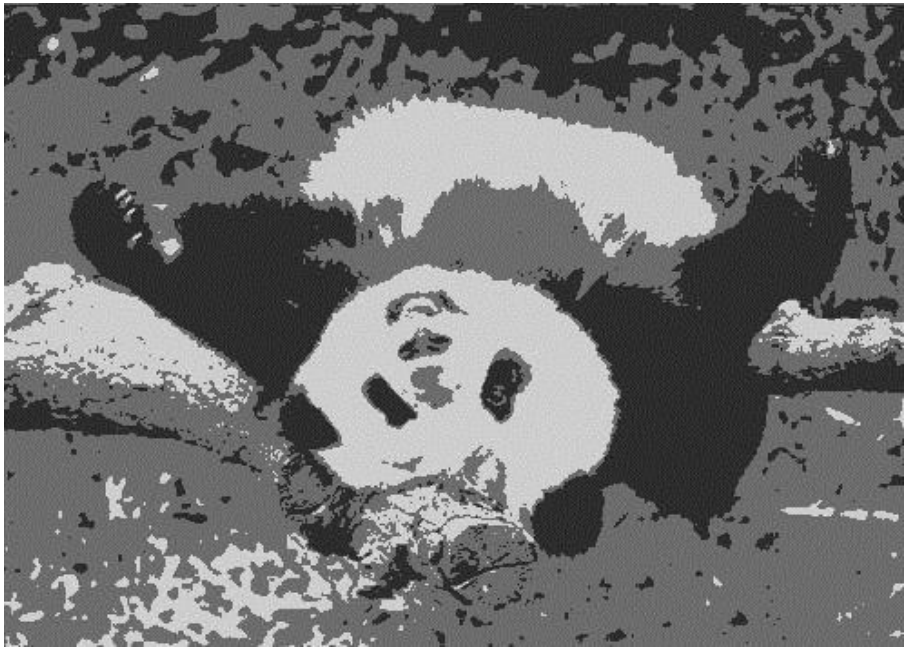
Grouping pixels based on **intensity** similarity

Feature space: intensity value (1-d)

Source: K. Grauman

K=2

K=3



Adapted from K. Grauman

# 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

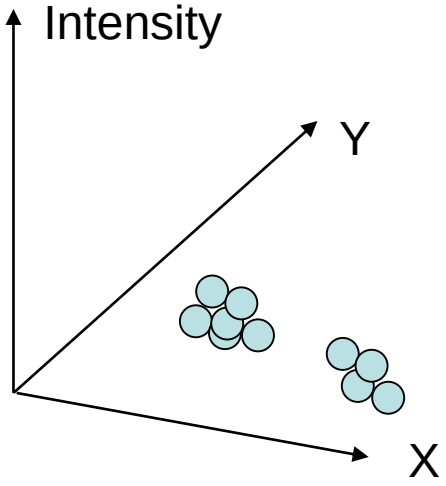
Clusters based on intensity similarity don’t have to be spatially coherent.

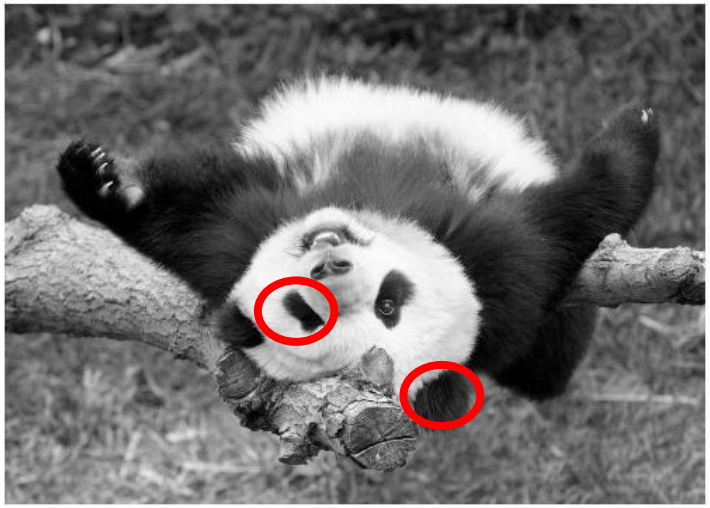
Source: K. Grauman

# 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



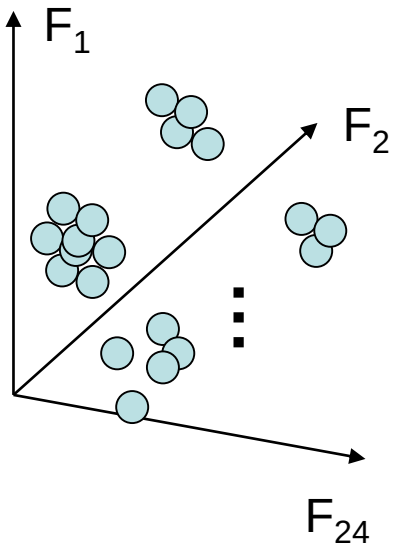
Both regions are black, but if we also include **position (x,y),** then we could group the two into distinct segments; way to encode both similarity & proximity.

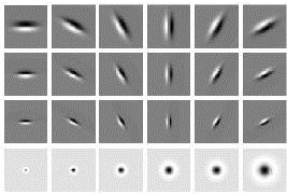
Source: K. Grauman

# 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)

Source: K. Grauman

# Summary

- 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.

# Plan for this lecture

- Group pixels into:
  - Edges: Extract gradients and threshold
  - Lines: Find which edge points are collinear or belong to another shape
  - Segments: Find which pixels form a consistent region, e.g. via clustering
- Transform pixels:
  - Find relationships between multiple views of the same world point
  - *Both parts rely on finding geometric relationships between pixels*

# Why multiple views?

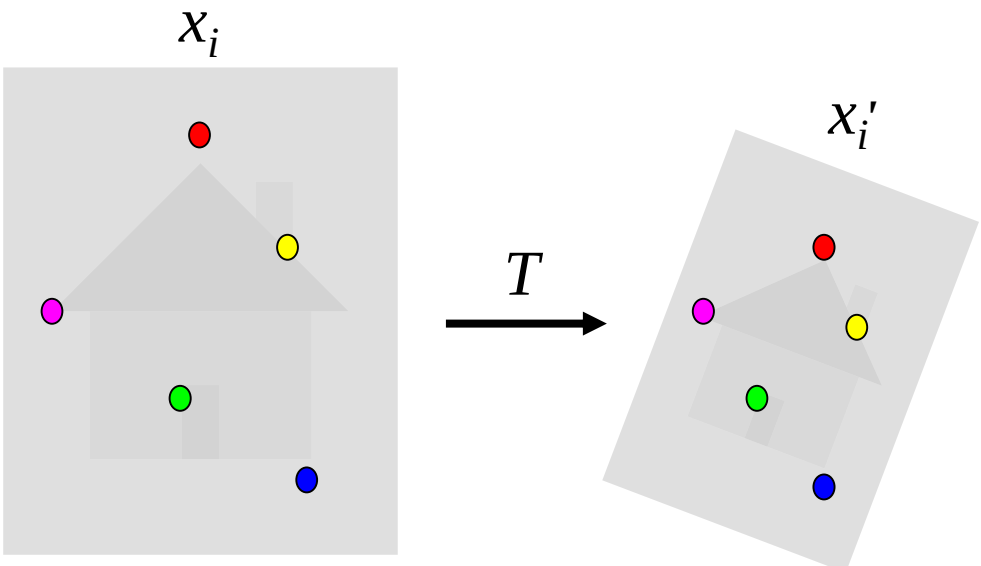
- Structure and depth are inherently ambiguous from single views.

- Multiple views help us to perceive 3d shape and depth.

Kristen Grauman, images from Svetlana Lazebnik

# Alignment problem

- We previously discussed how to match features across images, of the same or different objects
- Now let's focus on the case of "two images of the same object"(e.g. $x_i$ and $x_i$')
- What transformation relates $x_i$ and $x_i$'?
- In *alignment*, we will **fit the parameters of some transformation** according to a set of matching feature pairs ("correspondences").




Adapted from Kristen Grauman and Derek Hoiem

# Motivation: Image mosaics

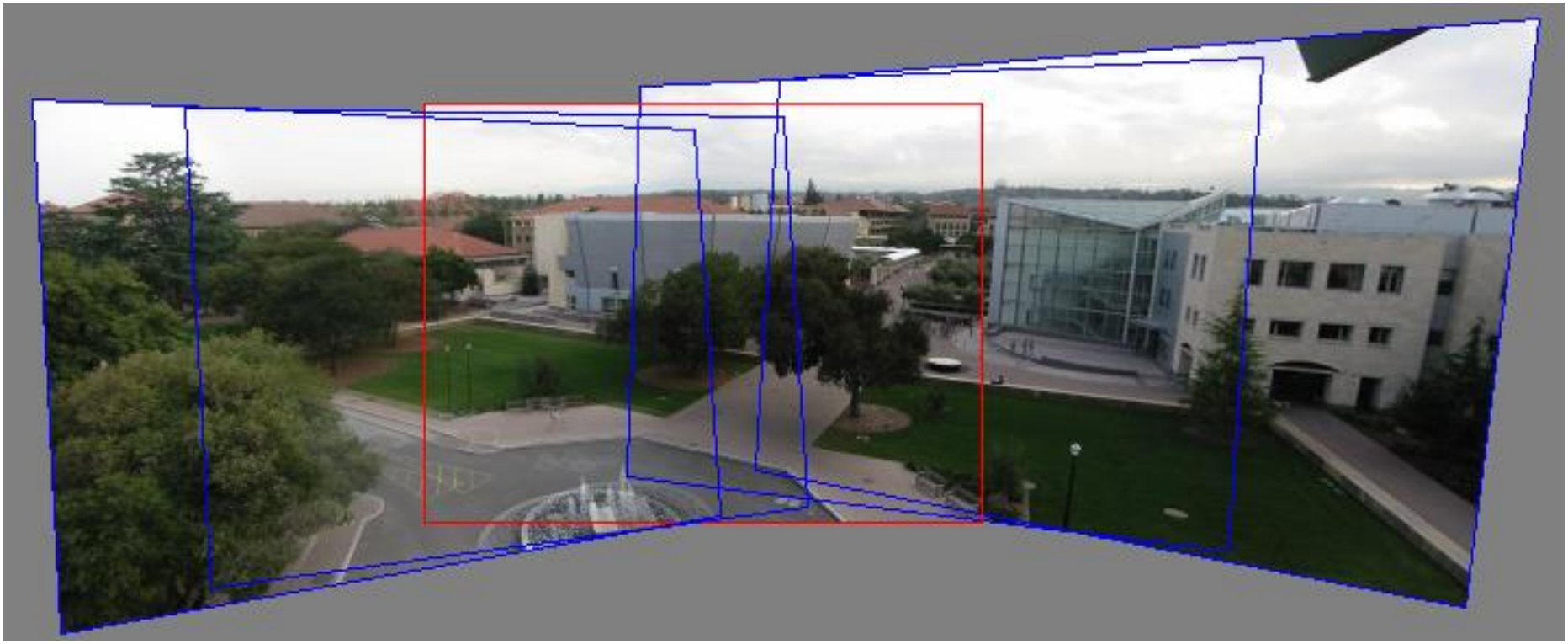
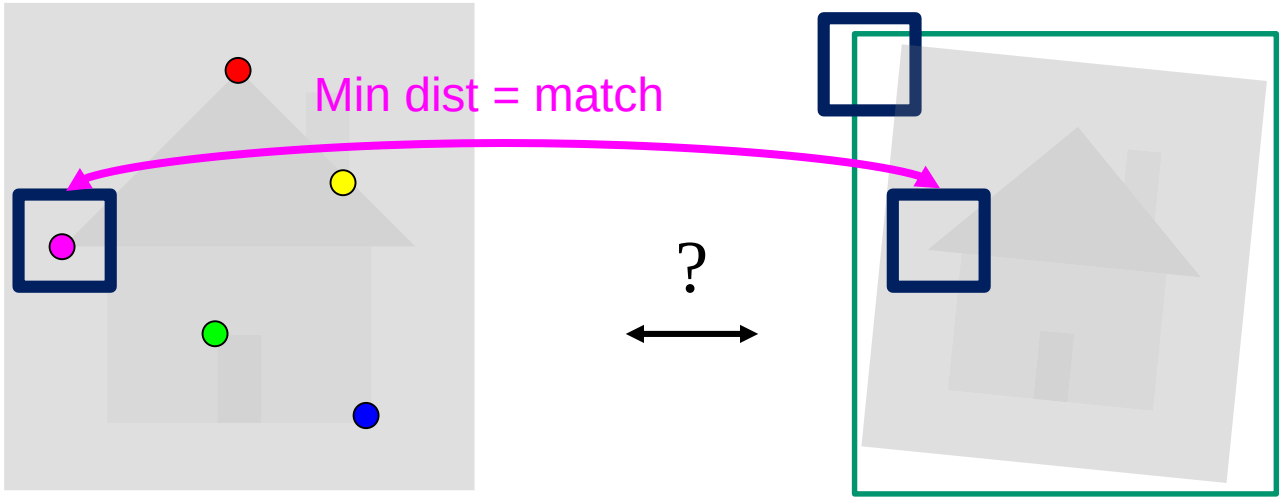




Image from http://graphics.cs.cmu.edu/courses/15-463/2010_fall/

# First, what are the correspondences?




- Compare content in **local** patches, find best matches.
  - Scan $x_i$' with template formed from a point in $x_i$, and compute e.g. Euclidean distance between SIFT features of the patches

Adapted from Kristen Grauman

# Second, what are the transformations?

Examples of transformations:

translate

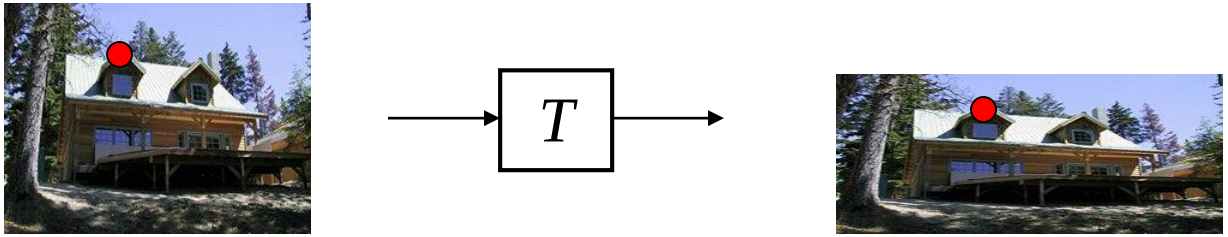
rotate

change aspect ratio

squish/shear

change perspective

# Parametric (global) warping




$\mathbf{p} = (x,y)$ $\mathbf{p'} = (x',y')$

Transformation T is a coordinate-changing machine:

$$p' = T(p)$$

What does it mean that *T* is **global**?

- It is the same for any point p
- It can be described by just a few numbers (parameters)

Let's represent *T* as a matrix:

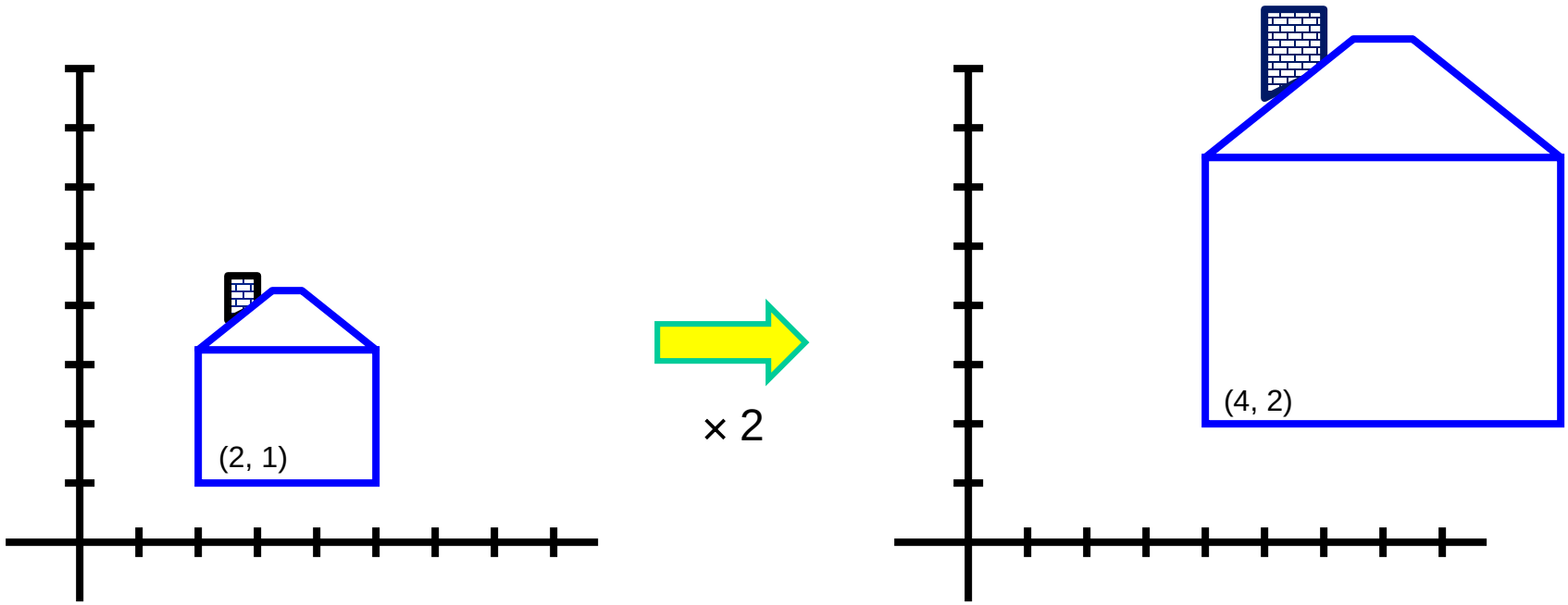
$$p' = \mathbf{M}p$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \mathbf{M} \begin{bmatrix} x \\ y \end{bmatrix}$$

# Scaling

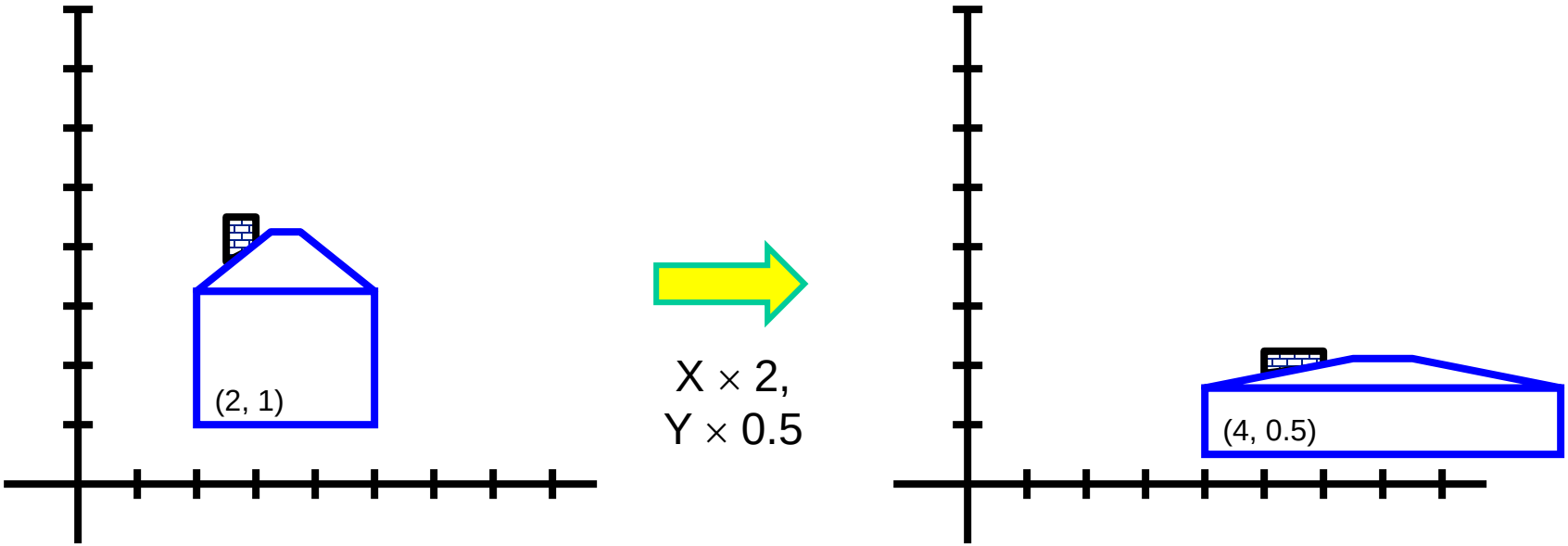
*Scaling* a coordinate means multiplying each of its components by a scalar

*Uniform scaling* means this scalar is the same for all components:



# Scaling

*Non-uniform scaling*: different scalars per component



# Scaling

Scaling operation:

$$x' = ax$$

$$y' = by$$

Or, in matrix form:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \underbrace{\begin{bmatrix} a & 0 \\ 0 & b \end{bmatrix}}_{\text{scaling matrix } S} \begin{bmatrix} x \\ y \end{bmatrix}$$

# 2D Linear transformations

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Only linear 2D transformations can be represented with a 2x2 matrix.

Linear transformations are combinations of …

- Scale,
- Rotation,
- Shear, and
- Mirror

# What transforms can we write w/ 2x2 matrix?

2D Scaling?

$$x' = s_x * x$$

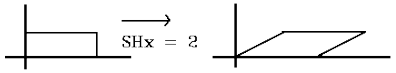
$$y' = s_y * y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Rotate around (0,0)? (see hidden slide)

$$x' = \cos\Theta * x - \sin\Theta * y$$
$$y' = \sin\Theta * x + \cos\Theta * y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\Theta & -\sin\Theta \\ \sin\Theta & \cos\Theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Shear? 


$$x' = x + sh_x * y$$

$$y' = sh_y * x + y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & sh_x \\ sh_y & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Modified from Alyosha Efros

Fig. from https://www.siggraph.org/education/materials/HyperGraph/modeling/mod_tran/2dshear.htm

# What transforms can we write w/ 2x2 matrix?

2D Mirror about Y axis?

$$x' = -x$$
$$y' = y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Mirror over (0,0)?

$$x' = -x$$
$$y' = -y$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

2D Translation?

$$\boldsymbol{x}' = \boldsymbol{x} + \boldsymbol{t}_x$$
$$\boldsymbol{y}' = \boldsymbol{y} + \boldsymbol{t}_y$$

**CAN'T DO!**

# Homogeneous coordinates

To convert to homogeneous coordinates:

$$(x, y) \Rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

homogeneous image coordinates

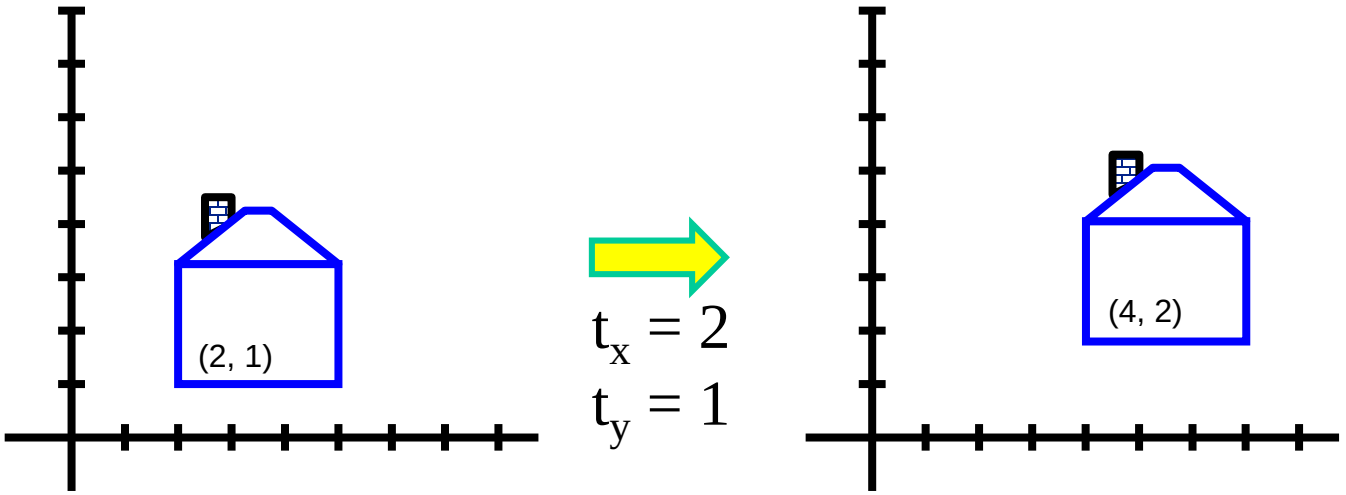
Converting *from* homogeneous coordinates

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} \Rightarrow (x/w, y/w)$$

# Translation

## Homogeneous Coordinates

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x + t_x \\ y + t_y \\ 1 \end{bmatrix}$$




Adapted from Alyosha Efros

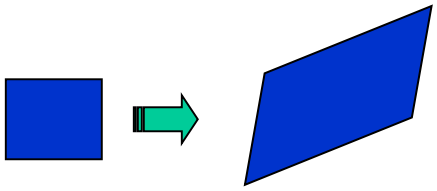
# 2D affine transformations

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

Affine transformations are combinations of …

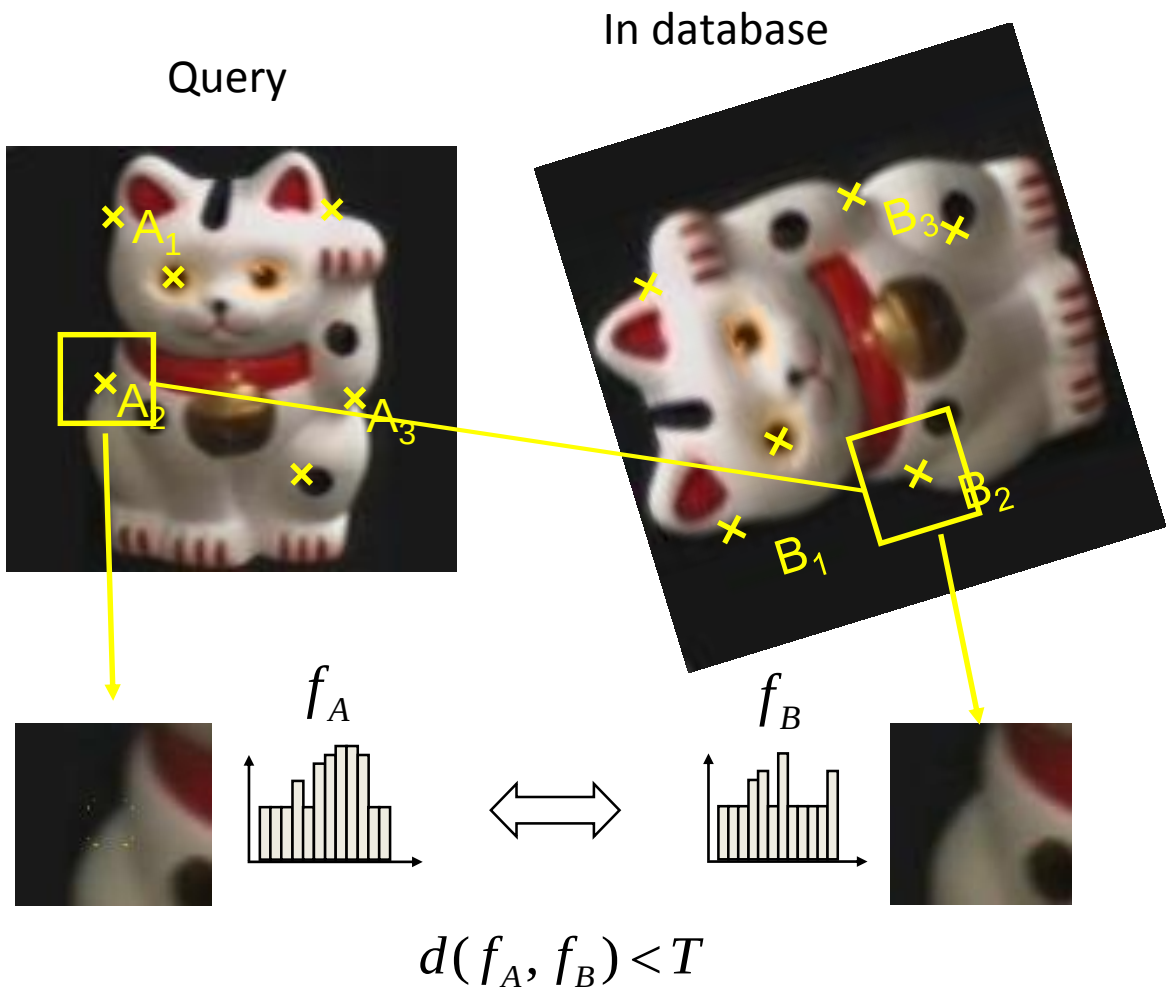
- Linear transformations, and
- Translations

Maps lines to lines, parallel lines remain parallel



Adapted from Alyosha Efros

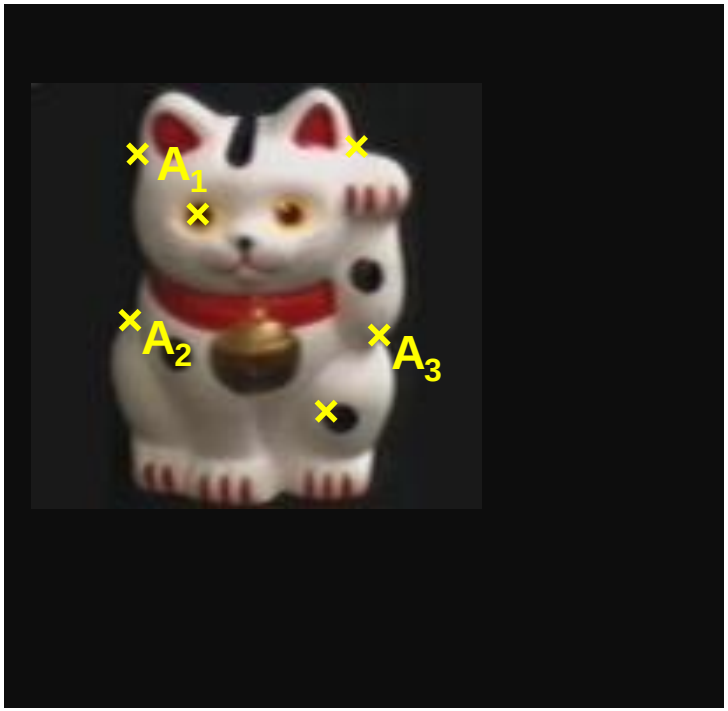
# Detour: Keypoint matching for search




1. Find a set of distinctive key-points
2. Define a region around each keypoint (window)
3. Compute a local descriptor from the region
4. Match descriptors

Adapted from K. Grauman, B. Leibe

# Detour: solving for translation with outliers

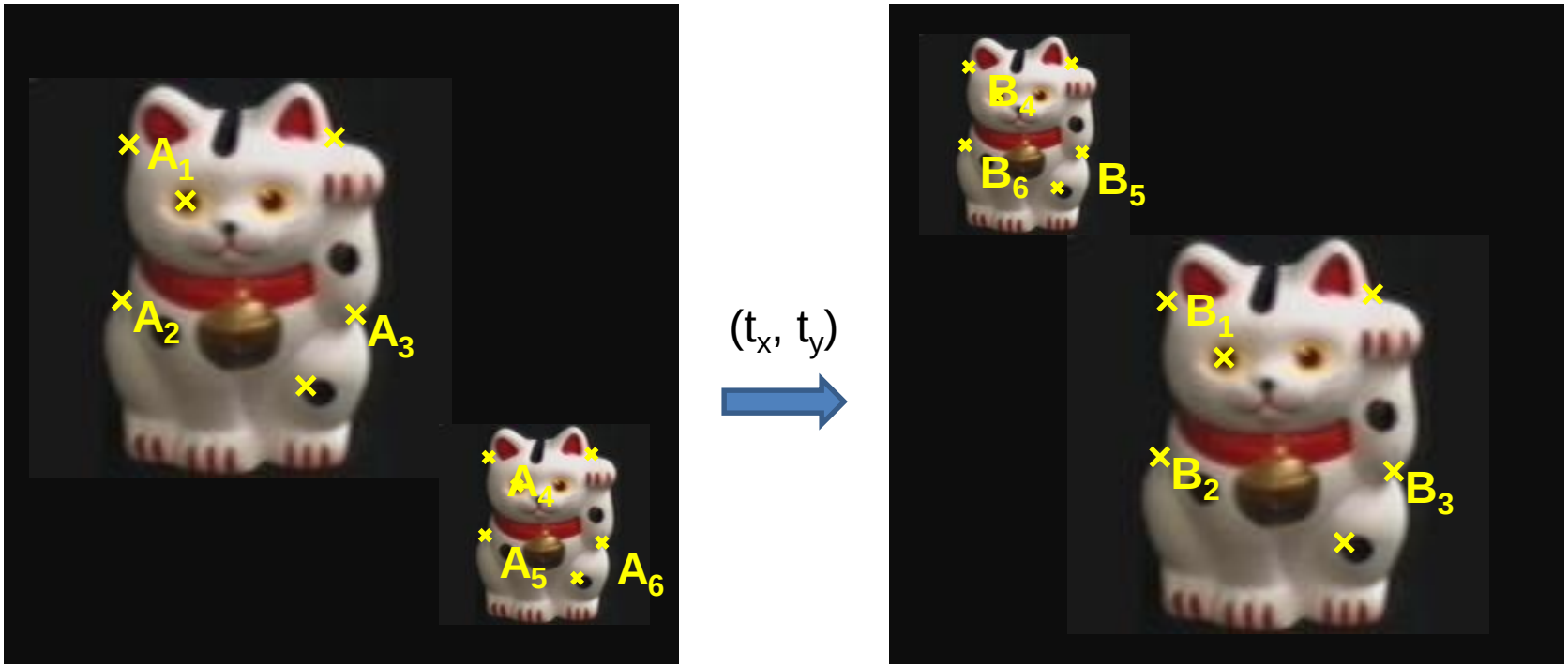




Given matched points in {A} and {B}, estimate the translation of the object

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

# Detour: solving for translation with outliers




**Problem: outliers, multiple objects**

## Hough transform solution

1. Initialize a grid of parameter values
2. Each matched pair casts a vote for consistent values
3. Find the parameters with the most votes

$$\begin{bmatrix} x_i^B \\ y_i^B \end{bmatrix} = \begin{bmatrix} x_i^A \\ y_i^A \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Adapted from Derek Hoiem

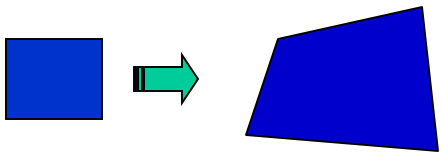
# Projective transformations

$$\begin{bmatrix} x' \\ y' \\ w' \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

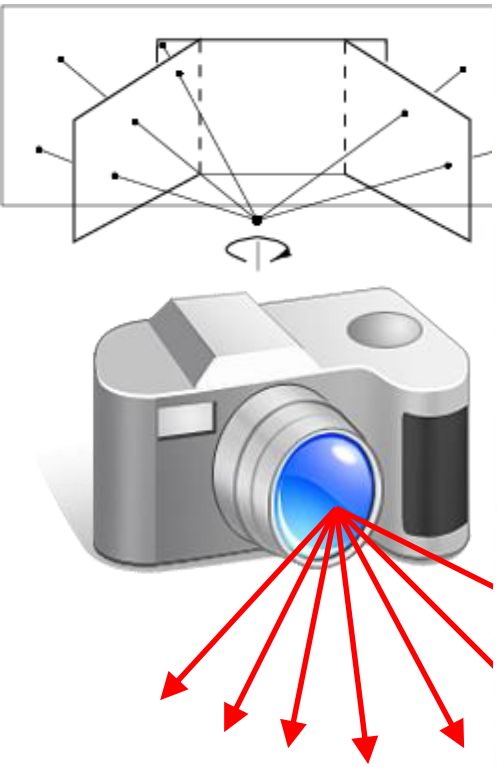
Projective transformations:

- Affine transformations, and
- Projective warps

Parallel lines do not necessarily remain parallel



# Image mosaics: Goals



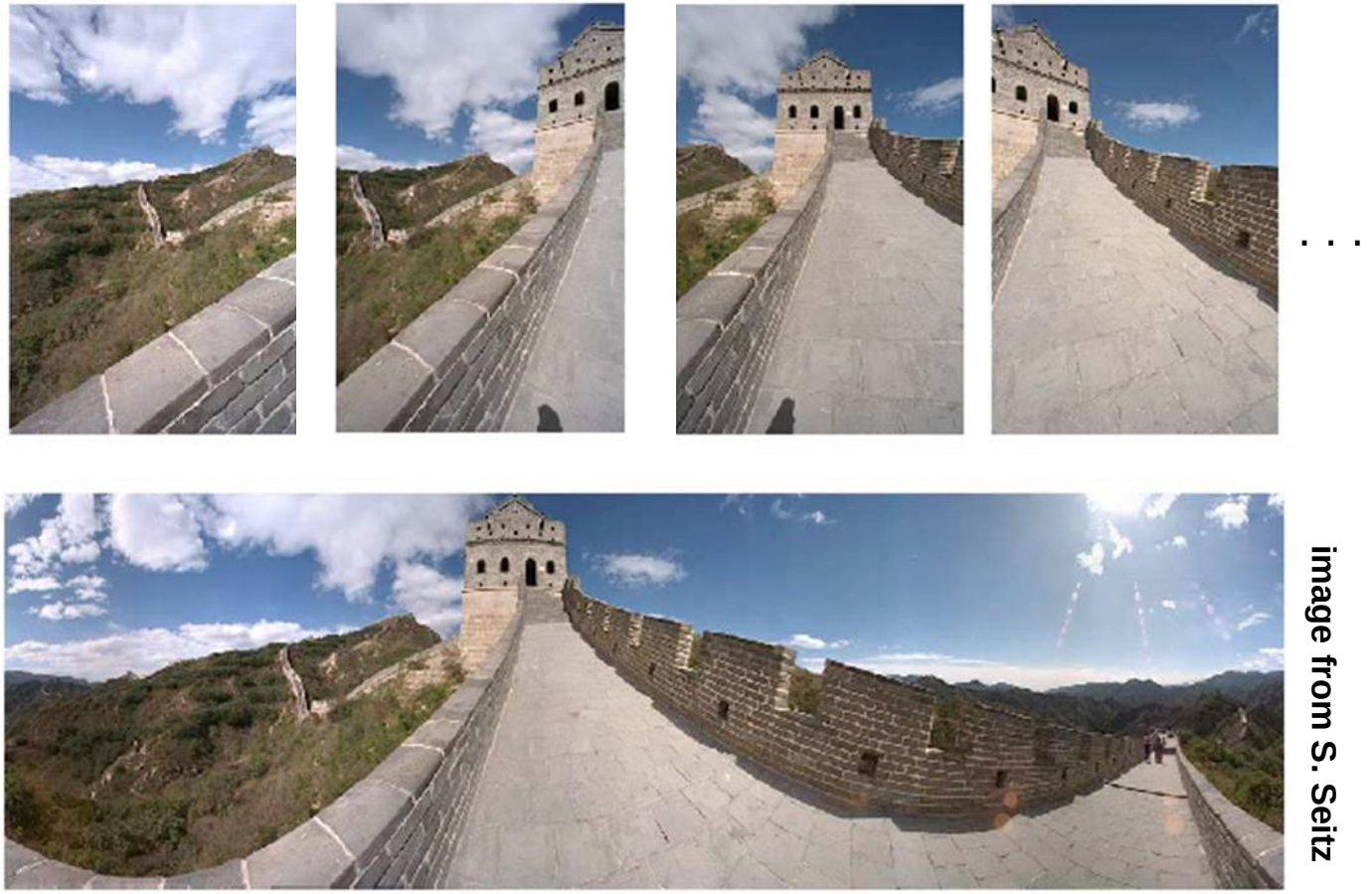
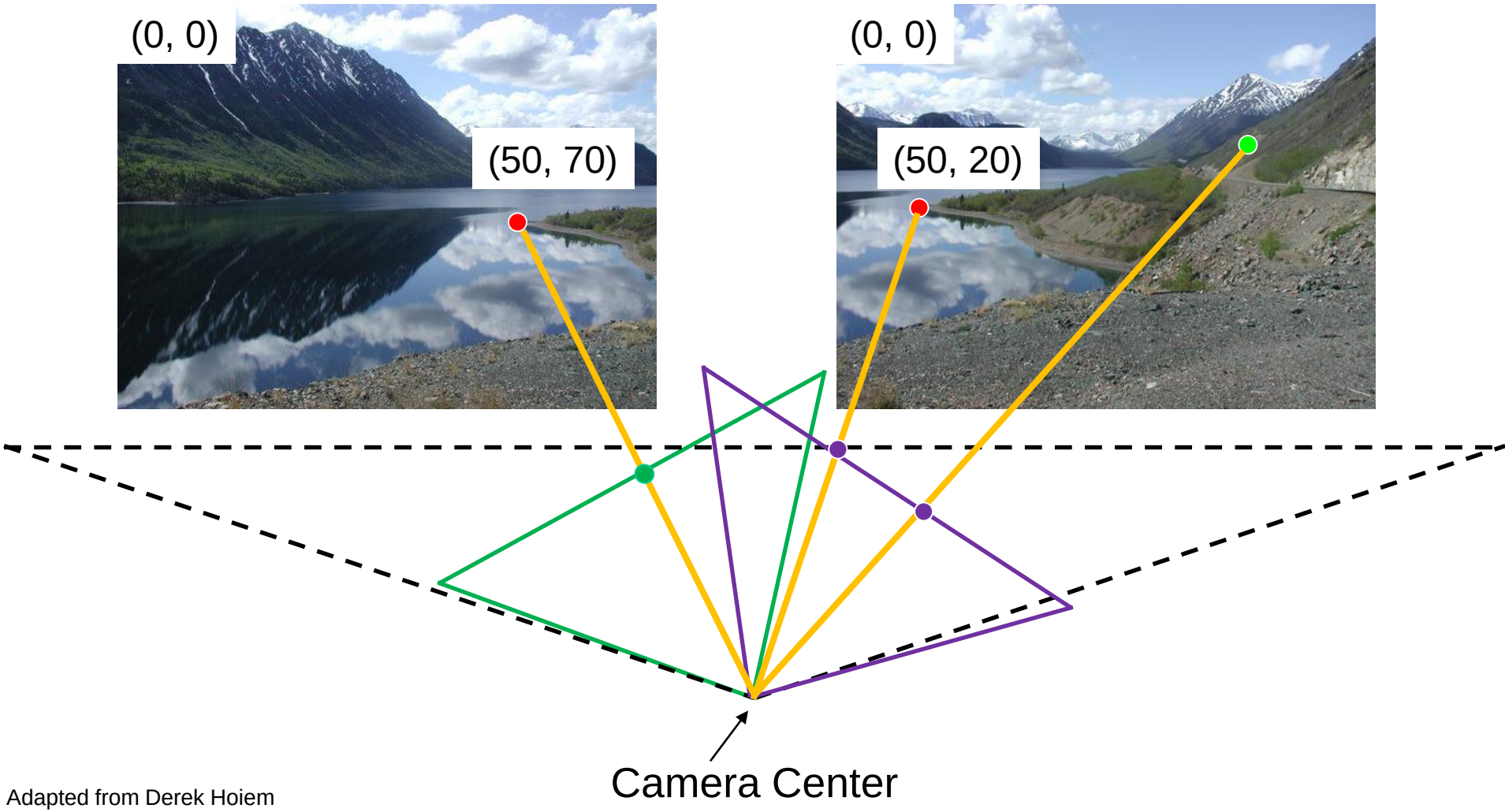


image from S. Seitz

Obtain a wider angle view by combining multiple images.

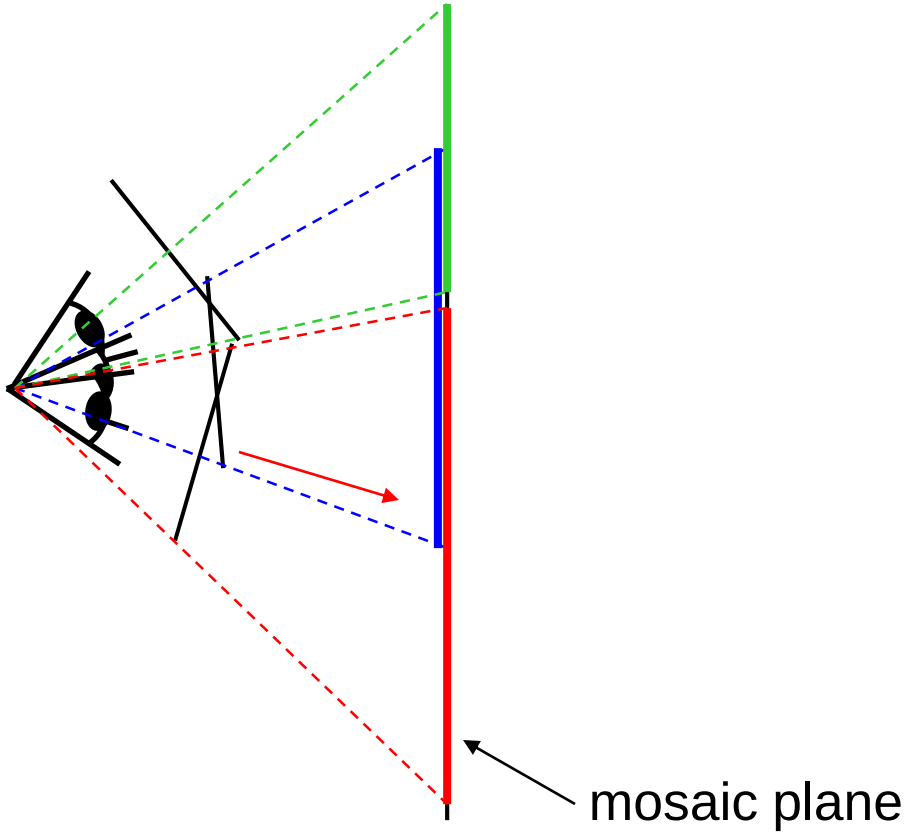
# Image mosaics: Camera setup

Two images with camera rotation but no translation




Adapted from Derek Hoiem

# Image mosaics: Many 2D views, one 3D object




The mosaic has a natural interpretation in 3D

- The images are reprojected onto a common plane
- The mosaic is formed on this plane
- Mosaic is a *synthetic wide-angle camera*

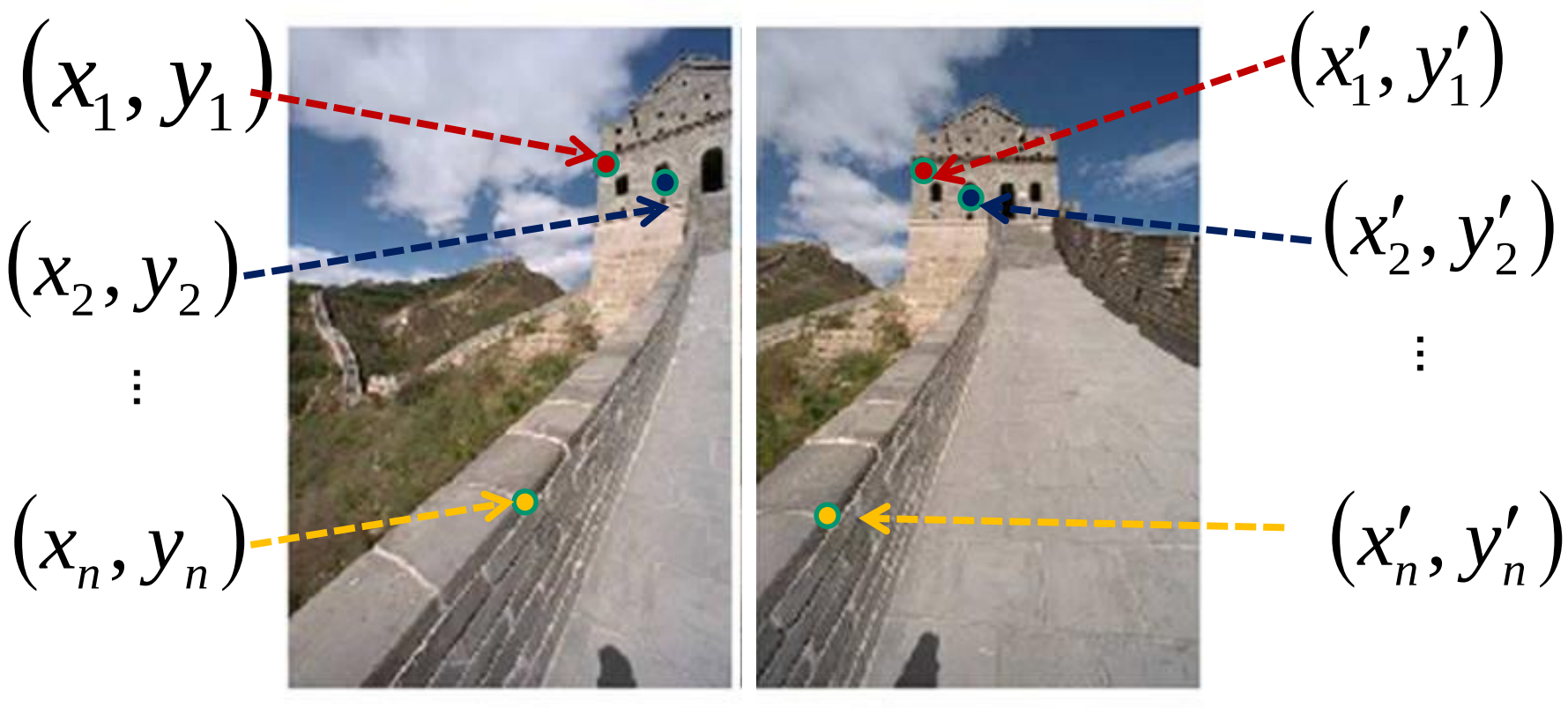
# How to stitch together panorama (mosaic)?

## Basic Procedure

- Take a sequence of images from the same position
  - Rotate the camera about its optical center
- **Compute the homography (transformation) between first and second image**
- **Transform the second image to overlap with the first**
- Blend the two together to create a mosaic
- (If there are more images, repeat)

Adapted from Steve Seitz

# Computing the homography




To **compute** the homography given pairs of corresponding points in the images, we need to set up an equation where the parameters of **H** are the unknowns…

# Computing the homography

- Assume we have four matched points: How do we compute homography **H**?

**p'=Hp**

$$\mathbf{p}' = \begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} \quad \mathbf{H} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \quad \mathbf{p} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \quad \mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix}$$

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & h_9 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Can set scale factor $h_9 = 1$.
So, there are 8 unknowns.
Need at least 8 eqs, but the more the better…

**A**

$$\begin{bmatrix} -x & -y & -1 & 0 & 0 & 0 & xx' & yx' & x' \\ 0 & 0 & 0 & -x & -y & -1 & xy' & yy' & y' \end{bmatrix} \mathbf{h} = \mathbf{0}$$

**DEMO**

Adapted from Derek Hoiem, Kristen Grauman

Derivation: http://www.cse.psu.edu/~rtc12/CSE486/lecture16.pdf

# How to stitch together panorama (mosaic)?

## Basic Procedure

- Take a sequence of images from the same position
  - Rotate the camera about its optical center
- **Compute the homography (transformation) between first and second image**
- **Transform the second image to overlap with the first**
- Blend the two together to create a mosaic
- (If there are more images, repeat)

Adapted from Steve Seitz

# Transforming the second image


To **apply** a given homography **H**

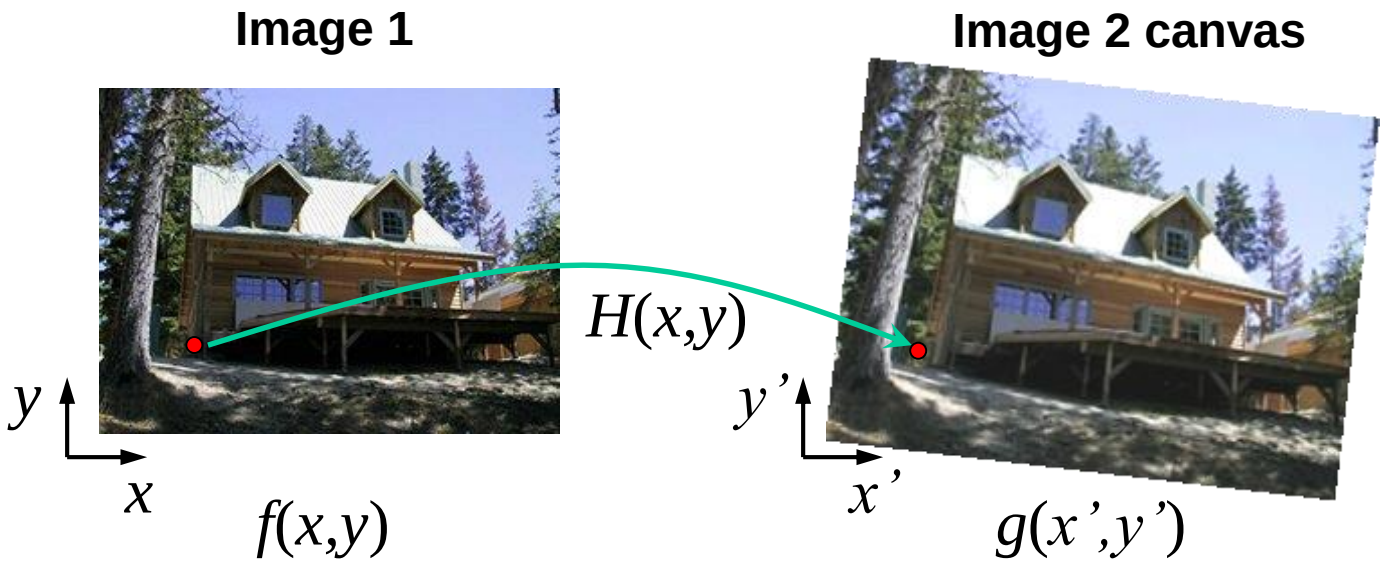
- Compute **p'** = **Hp** (regular matrix multiply)
- Convert **p'** from homogeneous to image coordinates

$$\begin{bmatrix} wx' \\ wy' \\ w \end{bmatrix} = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

**p'** **H** **p**

Modified from Kristen Grauman

# Transforming the second image



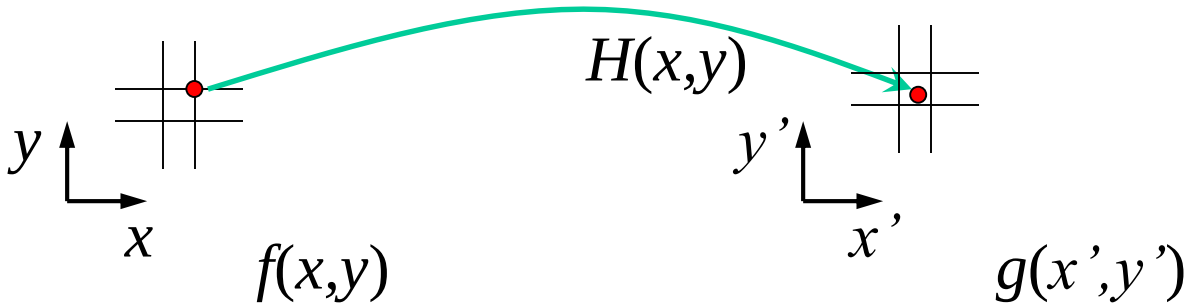

**Forward warping:**

Send each pixel $f(x,y)$ to its corresponding location

$(x',y') = H(x,y)$ in the right image

# Transforming the second image

ALTERNATIVES?




**Forward warping:**

Send each pixel $f(x,y)$ to its corresponding location

$(x',y') = H(x,y)$ in the right image

Q: what if pixel lands "between" two pixels?

A: distribute color among neighboring pixels (x',y')

# Next: Stereo vision

- Homography: Same camera center, but camera rotates
- Stereo vision: Camera center is not the same (we have multiple cameras)

- Epipolar geometry
  - Relates cameras from two positions/cameras

- Stereo depth estimation
  - Recover depth from disparities between two images

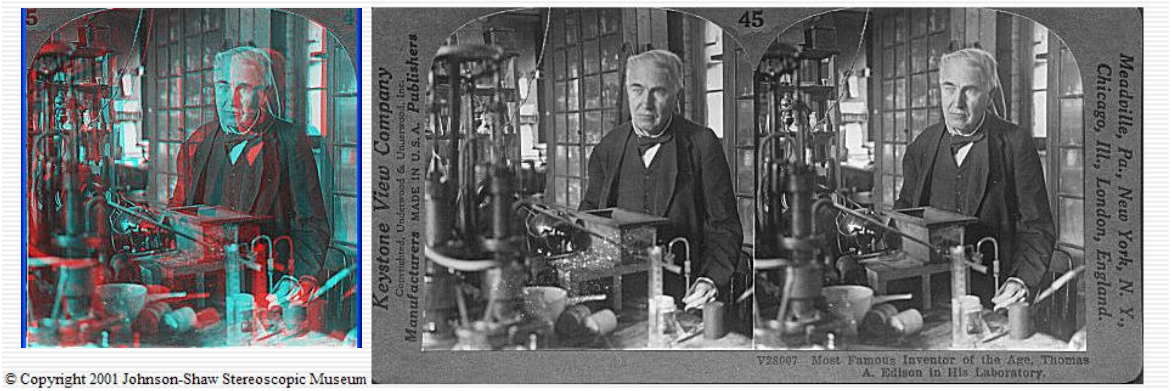
Adapted from Derek Hoiem

# Stereo photography and stereo viewers

Take two pictures of the same subject from two slightly different viewpoints and display so that each eye sees only one of the images.

Invented by Sir Charles Wheatstone, 1838

Image from fisher-price.com



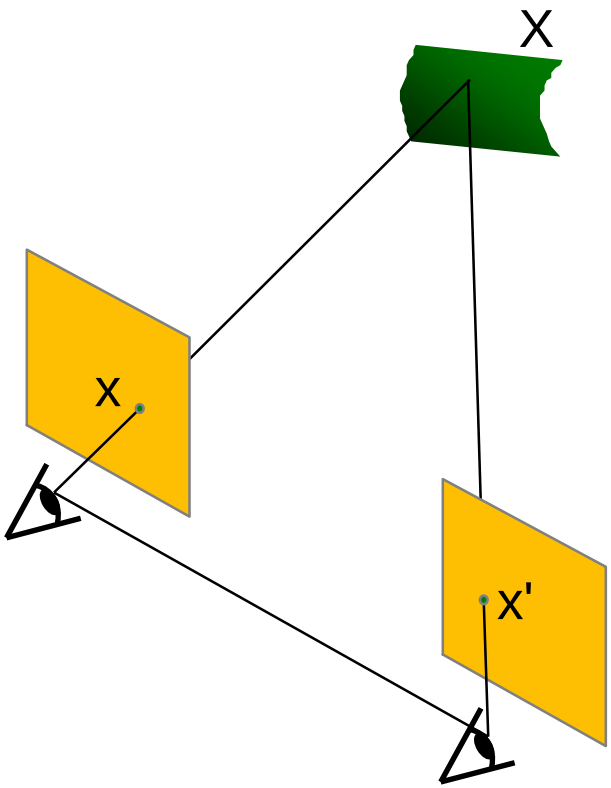
# Depth from stereo for computers

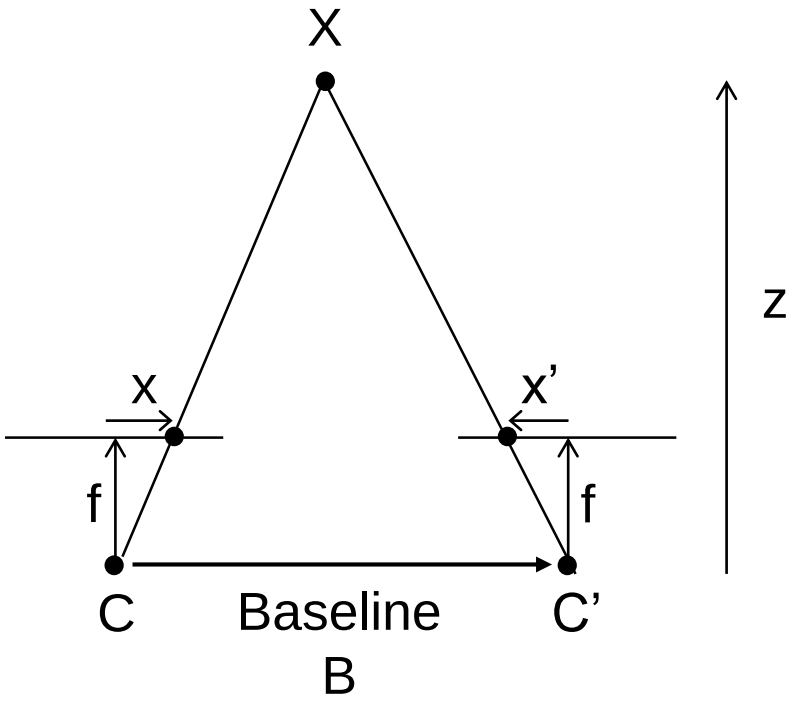
Two cameras, simultaneous views

Single moving camera and static scene

# Depth from stereo

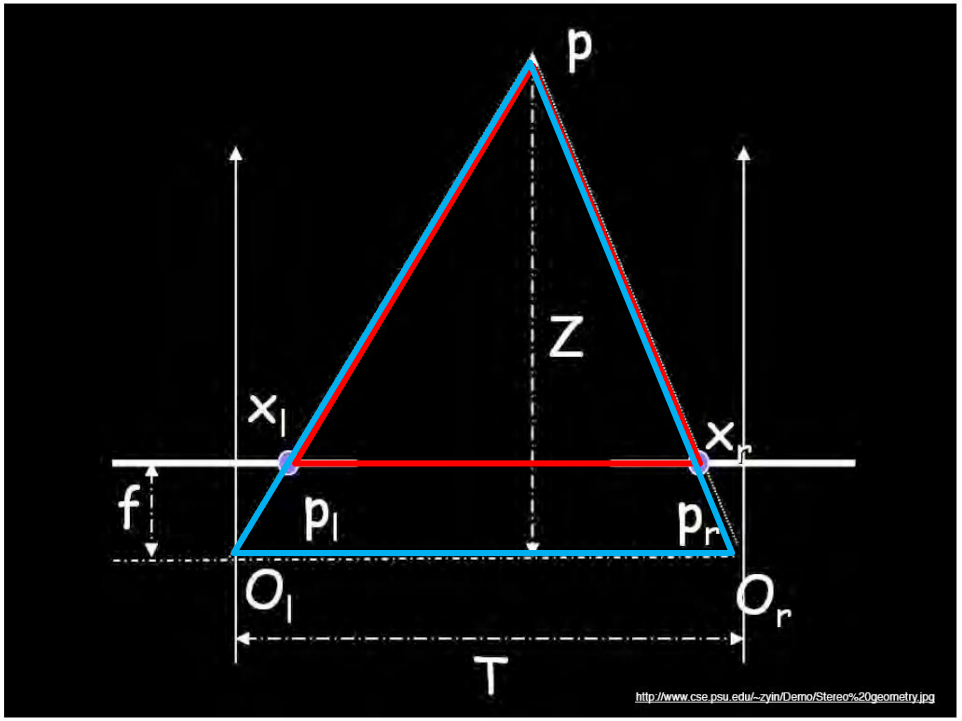
- Goal: recover depth by finding image coordinate x’ that corresponds to x, then measuring discrepancy between x and x’



# Geometry for a simple stereo system

- Assume parallel optical axes, known camera parameters (i.e., calibrated cameras). **What is expression for Z?**




Similar triangles ($p_l$, P, $p_r$) and ($O_l$, P, $O_r$):

$$\frac{T + x_l - x_r}{Z - f} = \frac{T}{Z}$$

**depth**

$$Z = f\frac{T}{x_r - x_l}$$

**disparity**

Depth is inversely proportional to disparity.

Adapted from Kristen Grauman

# Depth from disparity

- We have two images from different cameras.
- First, find **corresponding points** in two images
  - How to do this efficiently?
- Second, **estimate relative depth** from correspondences

image I(x,y)

Disparity map D(x,y)

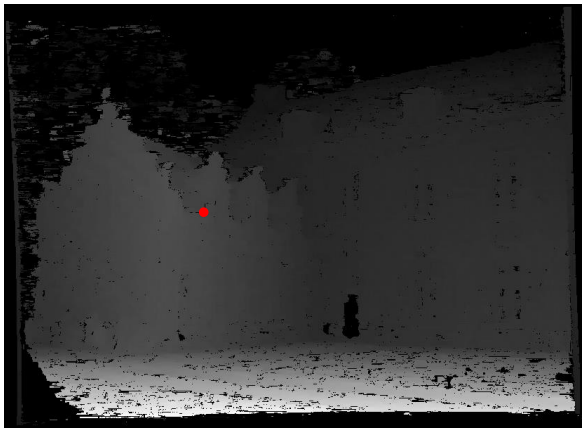
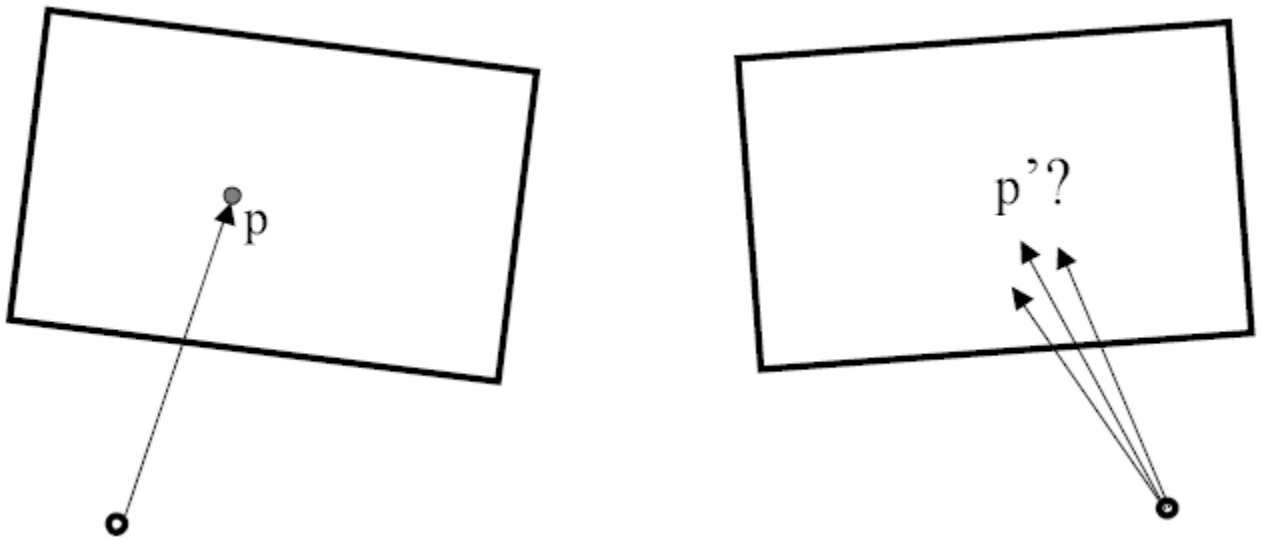


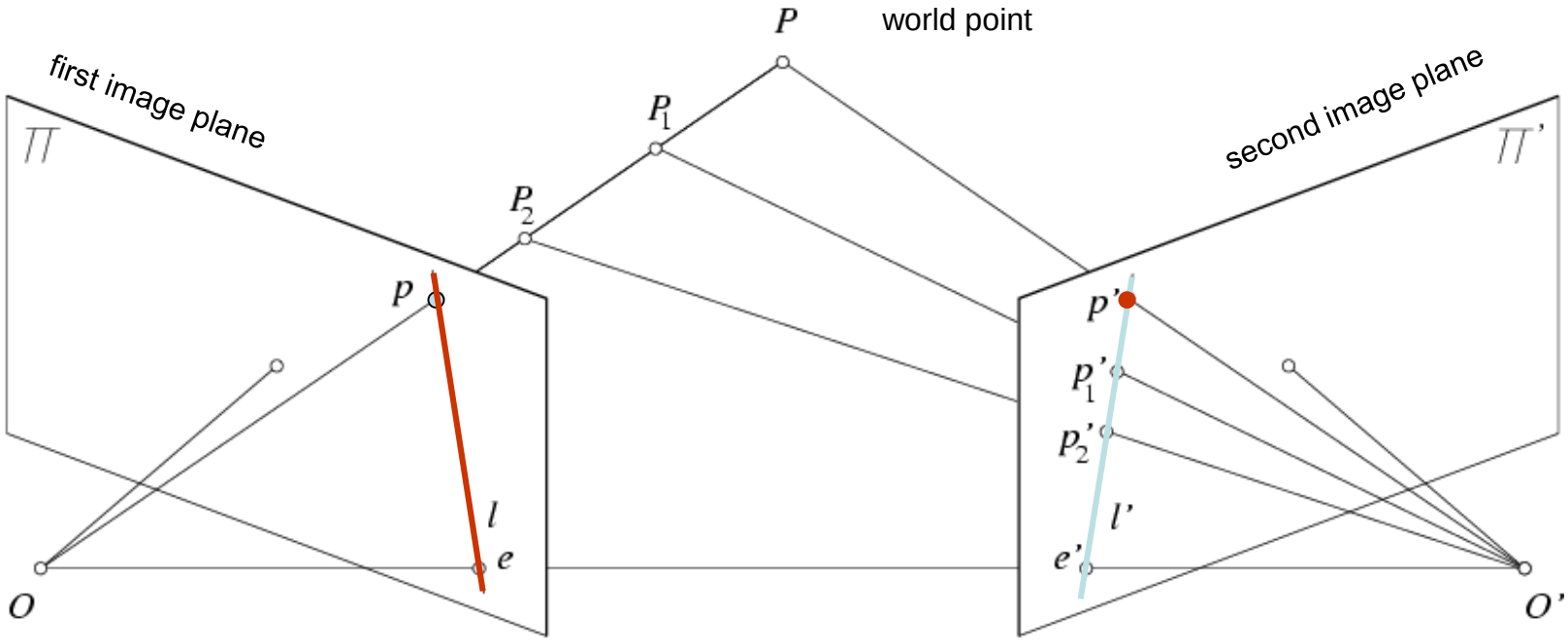
image I´(x´,y´)

# Stereo correspondence constraints




- Given p in left image, where can corresponding point p' be?

# Epipolar constraint

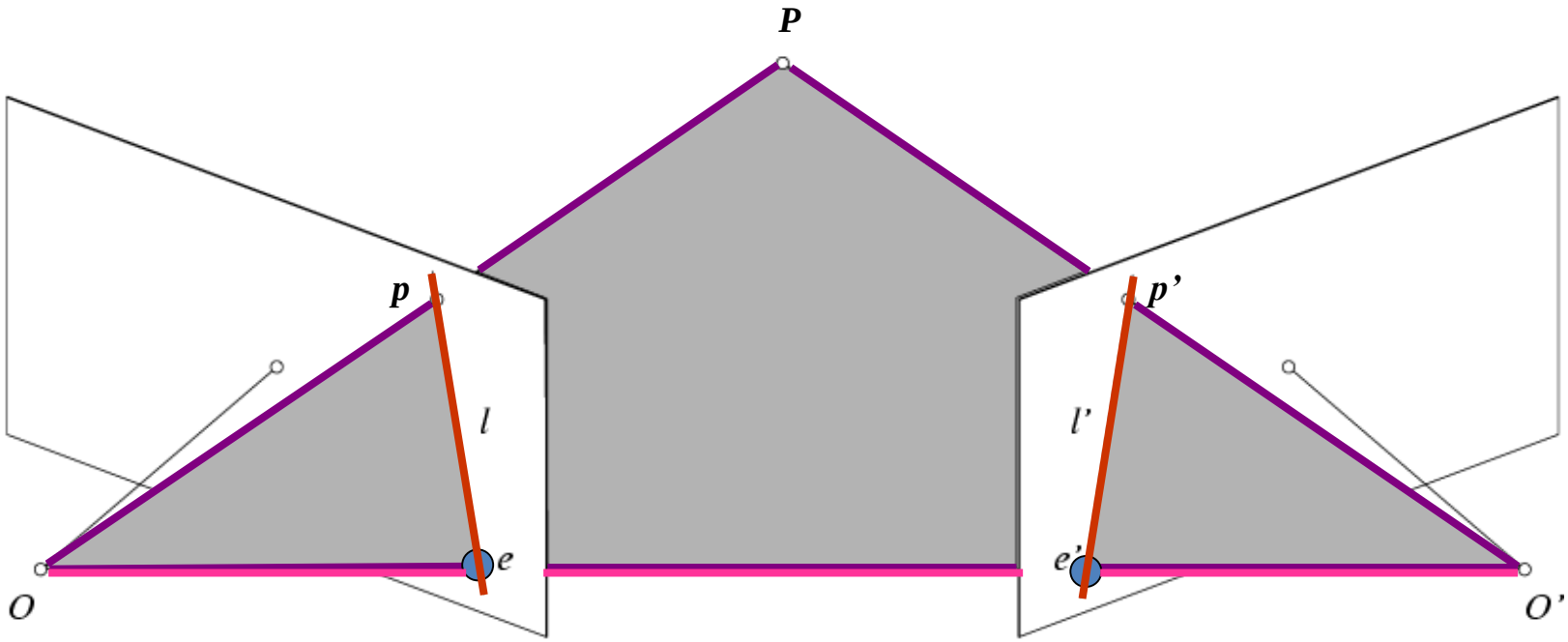



Geometry of two views constrains where the corresponding pixel for some image point in the first view must occur in the second view.

- It must be on the line where (1) the plane connecting the world point and optical centers, and (2) the image plane, intersect.
- Potential matches for *p* have to lie on the corresponding line *l'*.
- Potential matches for *p'* have to lie on the corresponding line *l*.

Adapted from Kristen Grauman, Derek Hoiem

# Epipolar geometry: notation




- **Baseline** – line connecting the two camera centers
- **Epipoles**

= intersections of baseline with image planes

= projections of the other camera center

- **Epipolar Plane** – plane containing baseline
- **Epipolar Lines** - intersections of epipolar plane with image planes (always come in corresponding pairs)

Adapted from Derek Hoiem

# Epipolar constraint

The epipolar constraint is useful because it reduces the correspondence problem to a 1D search along an epipolar line.

**See hidden slides for details.**

Kristen Grauman, image from Andrew Zisserman

Rigs related by:
Rotation: 3x3 matrix **R**
Translation: 3x1 vector **T**.

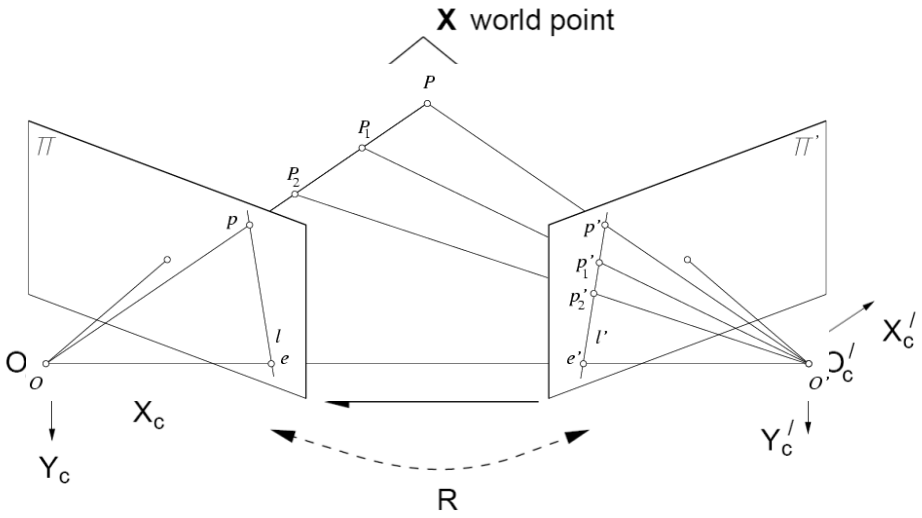
# Essential matrix

$$\mathbf{X}' \cdot \left(\mathbf{T} \times \mathbf{R}\mathbf{X}\right) = 0$$

$$\mathbf{X}' \cdot \left([\mathrm{T}_x]\mathbf{R}\mathbf{X}\right) = 0$$

Let $\mathbf{E} = [\mathrm{T}_x]\mathbf{R}$

$$\mathbf{X}' \cdot \mathbf{E}\mathbf{X} = \mathbf{X}'^{T}\mathbf{E}\mathbf{X} = 0$$



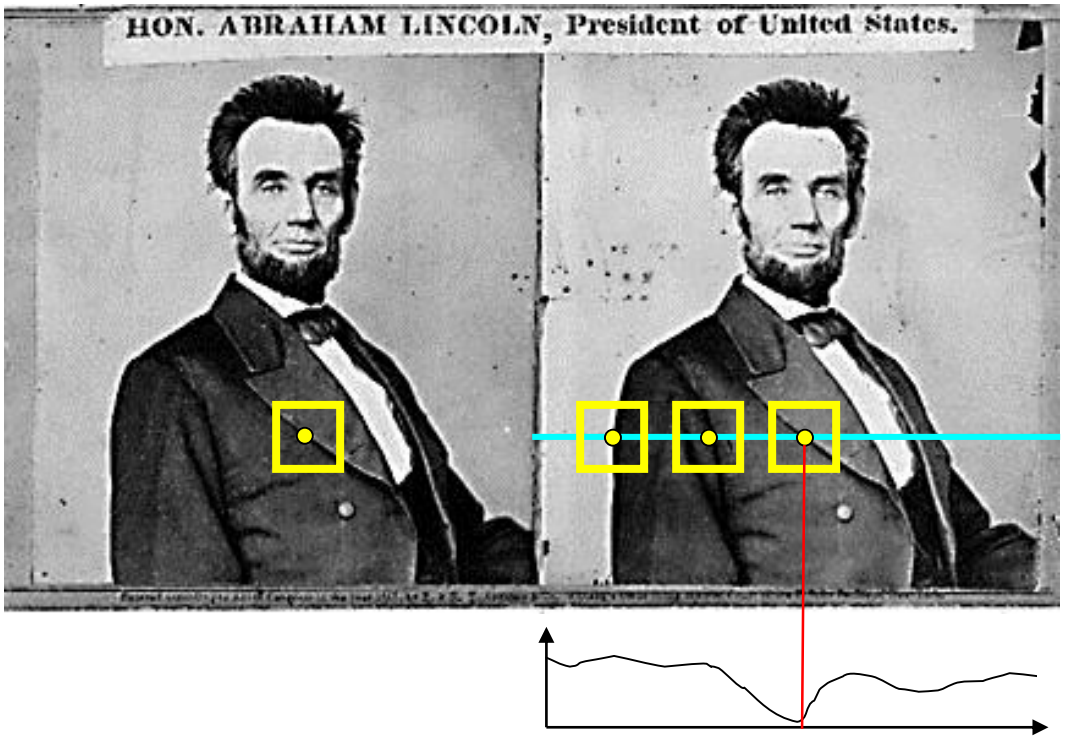

**E** is called the **essential matrix**, and it relates corresponding image points between both cameras, given the **rotation** and **translation**.

Before we said: If we observe a point in one image, its position in other image is constrained to lie on line defined by above. It turns out that:

- $E^T x$ is the epipolar line l' through x' in the second image, corresponding to x.
- Ex' is the epipolar line l through x in the first image, corresponding to x'.

Adapted from Kristen Grauman, Derek Hoiem

# Basic stereo matching algorithm

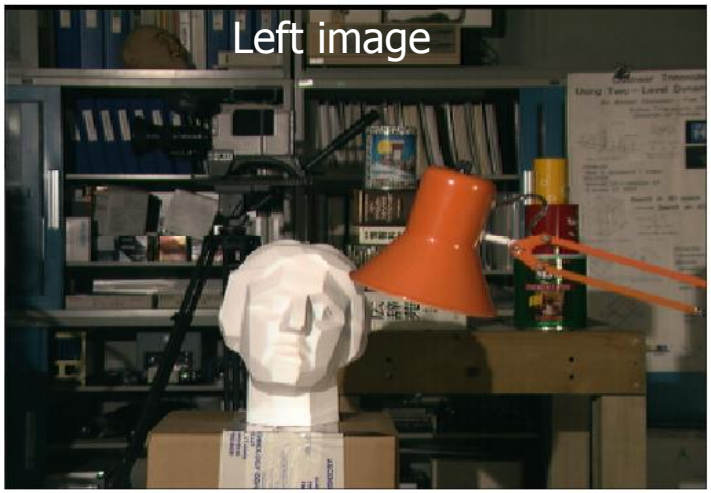


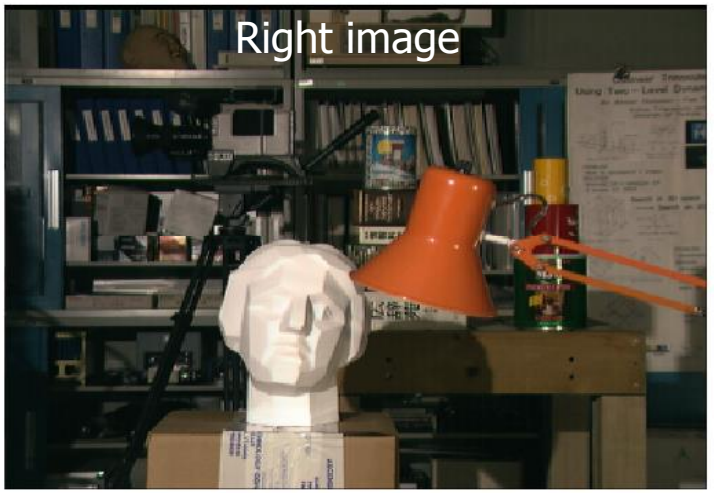

- For each pixel in the first image
  - Find corresponding epipolar scanline in the right image
  - Search along epipolar line and pick the best match x’ (e.g. smallest Euclidean distance between SIFT in patch)
  - Compute disparity x-x’ and set depth(x) = f*T/(x-x’)

Adapted from Derek Hoiem

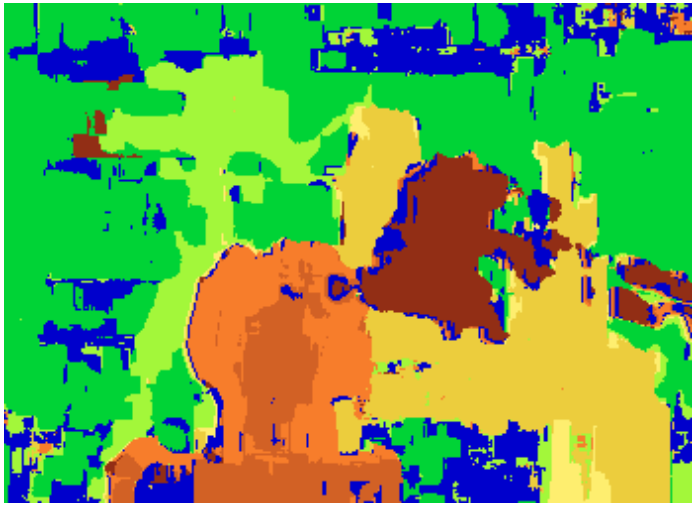
# Results with window search

Data




Predicted depth



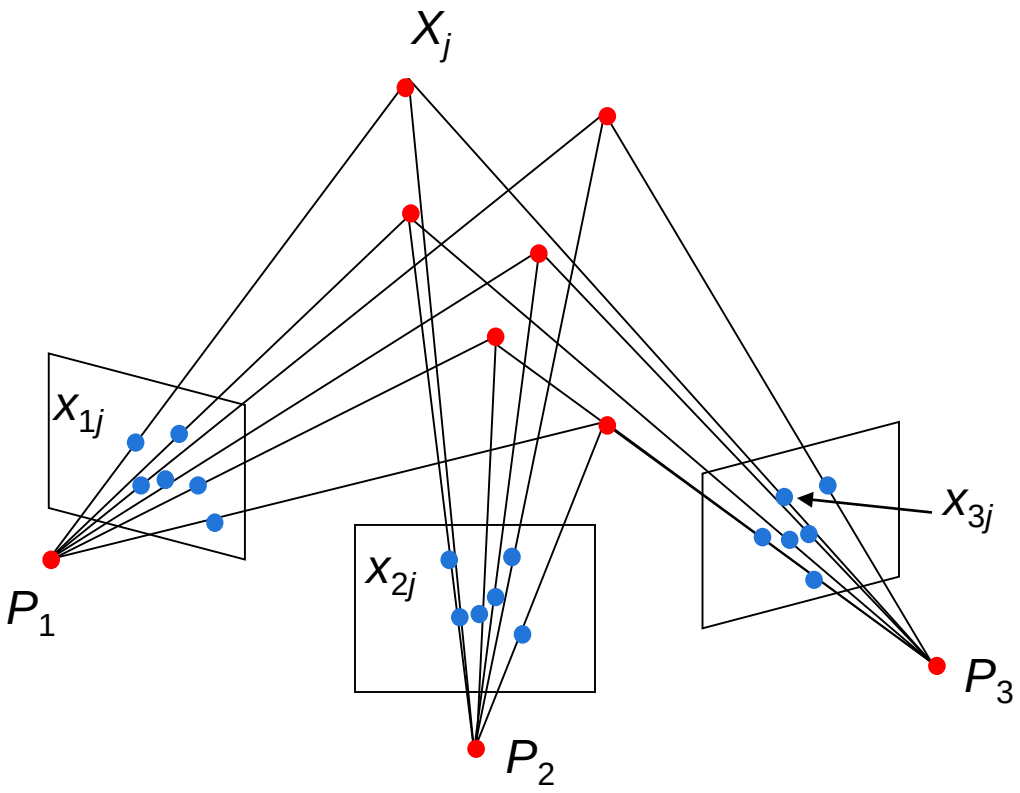
Ground truth

# Projective structure from motion

- Given: *m* images of *n* fixed 3D points

$$\mathbf{x}_{ij} = \mathbf{P}_i \mathbf{X}_j, \qquad i = 1, \ldots, m, \quad j = 1, \ldots, n$$

- Problem: estimate *m* projection matrices $\mathbf{P}_i$ and *n* 3D points $\mathbf{X}_j$ from the *mn* corresponding 2D points $\mathbf{x}_{ij}$



# Photo tourism

Noah Snavely, Steven M. Seitz, Richard Szeliski, "Photo tourism: Exploring photo collections in 3D," SIGGRAPH 2006


http://phototour.cs.washington.edu/

# 3D from multiple images

Sameer Agarwala, Noah Snavely, Ian Simon, Steven M. Seitz, Richard Szeliski, "Building Rome in a Day," ICCV 2009

# Summary of multiple views

- Write **2d transformations** as matrix-vector multiplication
- **Fitting transformations**: Solve for unknown parameters given corresponding points from two views – linear, affine, projective (homography)
- **Mosaics**: Uses homography and image warping to merge views taken from same center of projection
- **Stereo depth estimation:** Find corresponding points along epipolar scanline, then measure disparity (as inverse to depth)
- **Epipolar geometry:** Matching point in second image is on a line passing through its epipole; makes search for correspondences quicker

Adapted from Kristen Grauman and Derek Hoiem