Dr. George Karraz, Ph. D.

Computer Vision

Lecture IV: **Edge & Structure Extraction**

Dr. George Karraz, Ph.D.

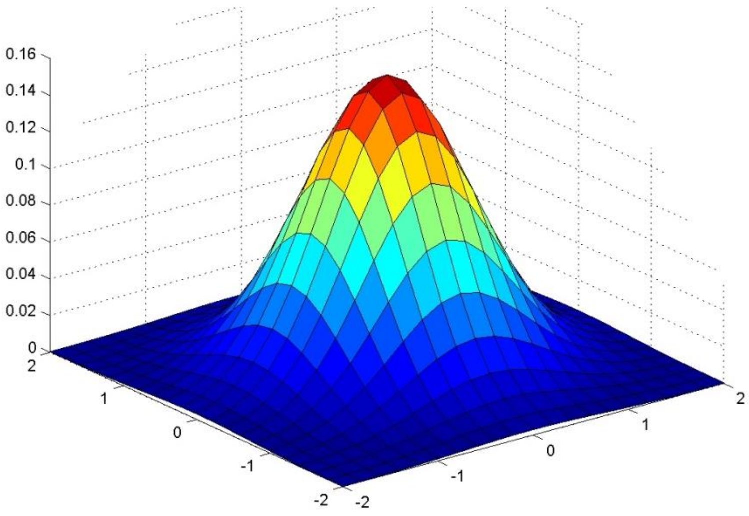
Course Outline

- Image Processing Basics
  - Image Formation
  - Binary Image Processing
  - Linear Filters
  - Edge & Structure Extraction
  - Color
- Segmentation
- Local Features & Matching
- Object Recognition and Categorization
- 3D Reconstruction
- Motion and Tracking

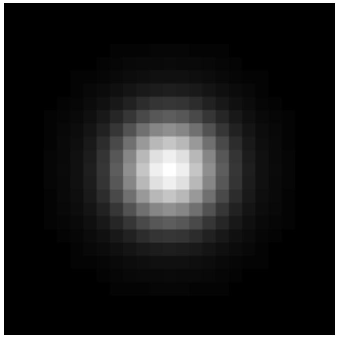
Recap: Gaussian Smoothing

- Gaussian kernel

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$



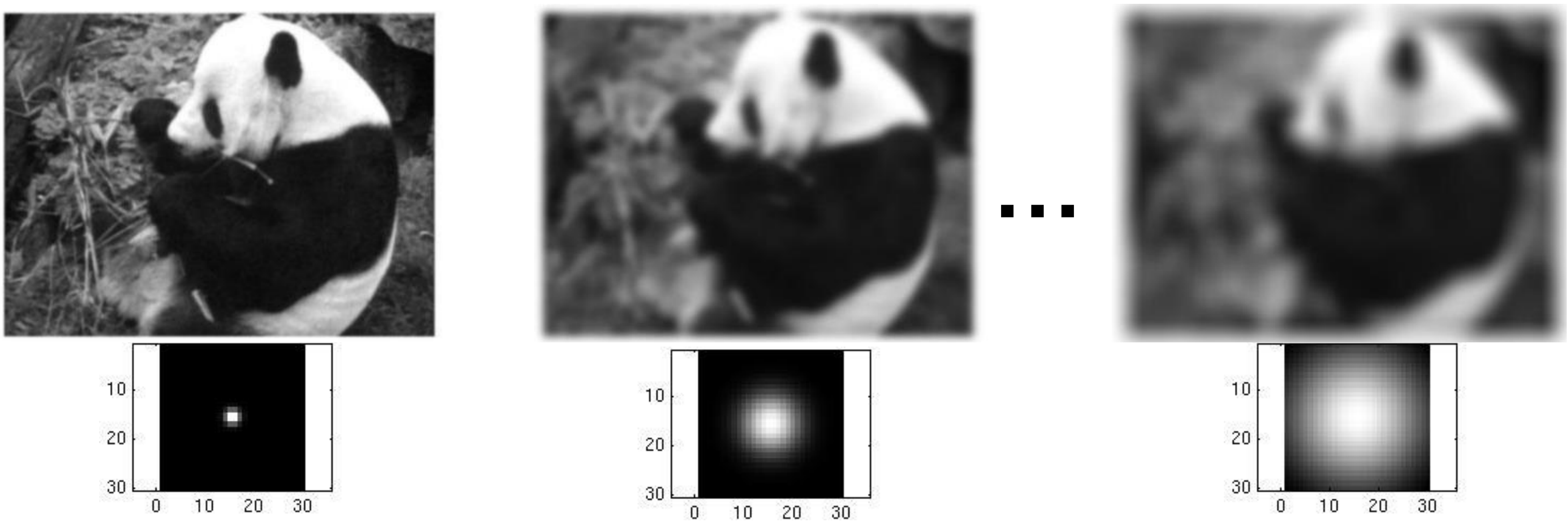
- Rotationally symmetric
- Weights nearby pixels more than distant ones
  - This makes sense as ‘probabilistic’ inference about the signal



- A Gaussian gives a good model of a fuzzy blob

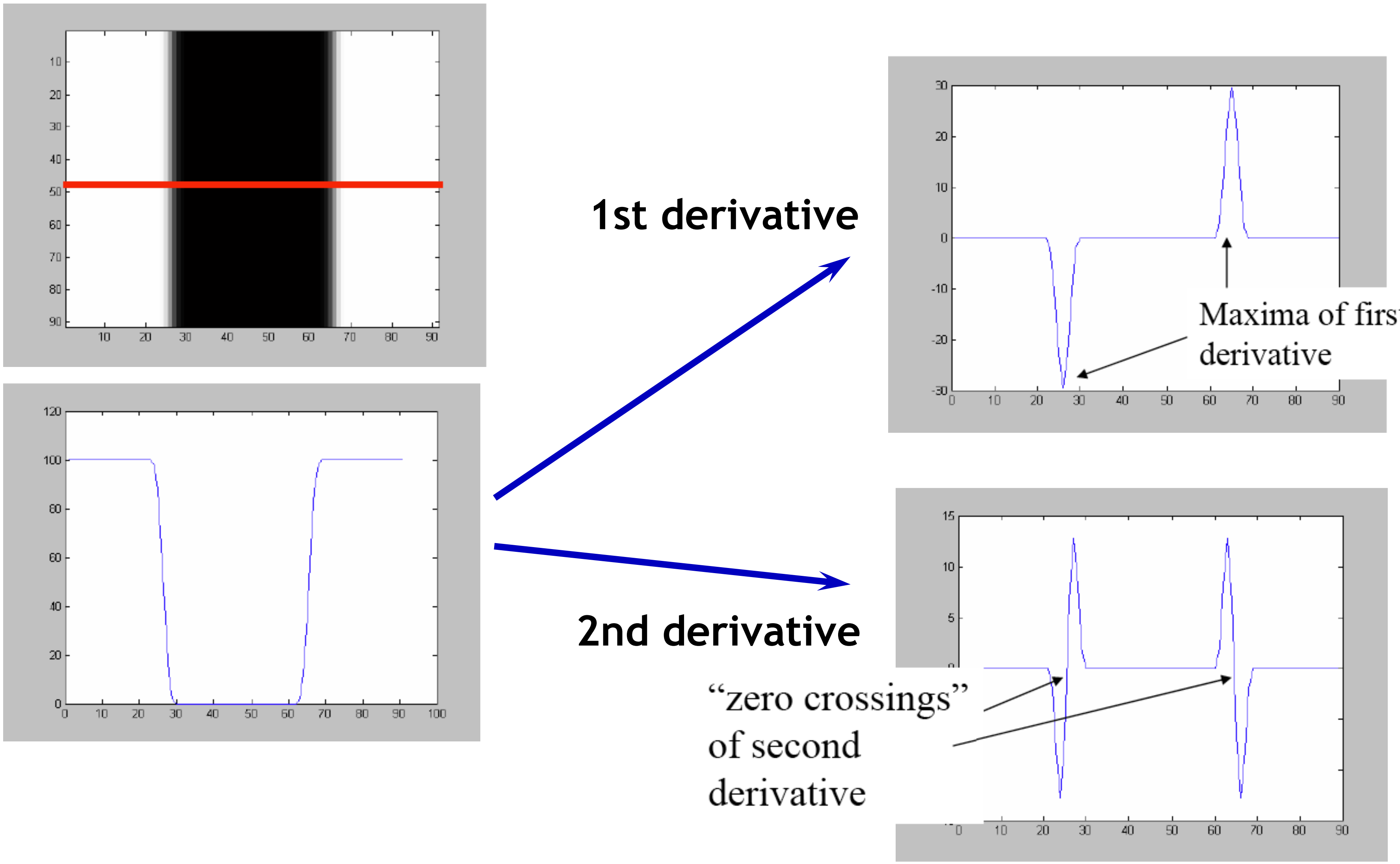
Smoothing with a Gaussian

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

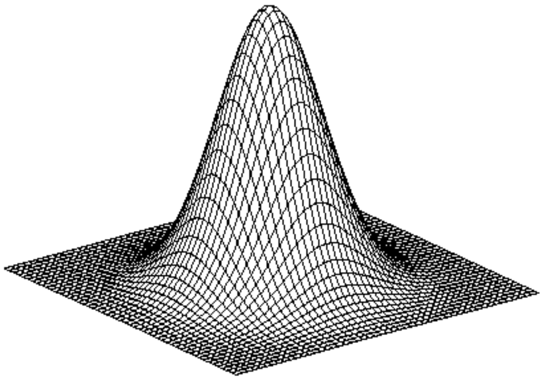



```
for sigma=1:3:10
   h = fspecial('gaussian‘, fsize, sigma);
   out = imfilter(im, h);
   imshow(out);
   pause;
end
```

Recap: Derivatives and Edges…

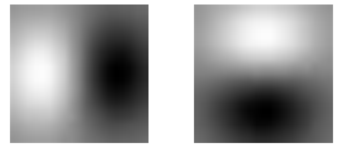


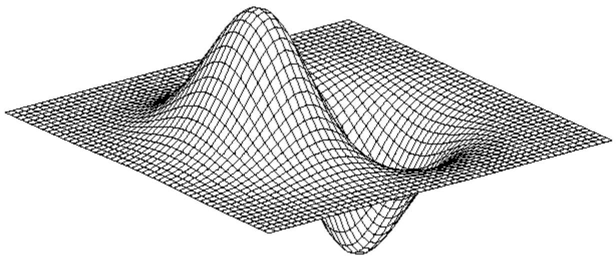
Recap: 2D Edge Detection Filters



Gaussian

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

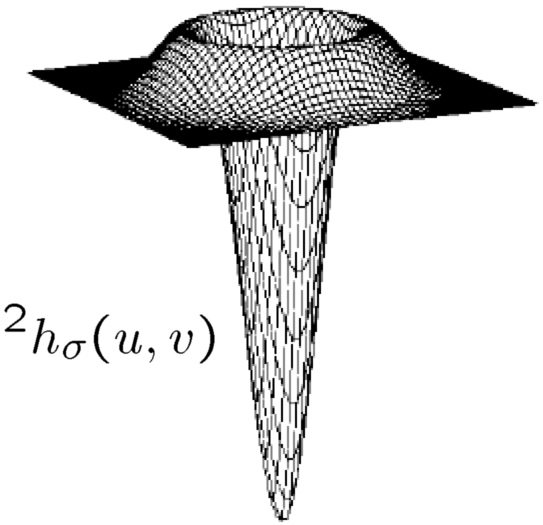




Derivative of Gaussian

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

Laplacian of Gaussian



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

- ∇^2 is the Laplacian operator:

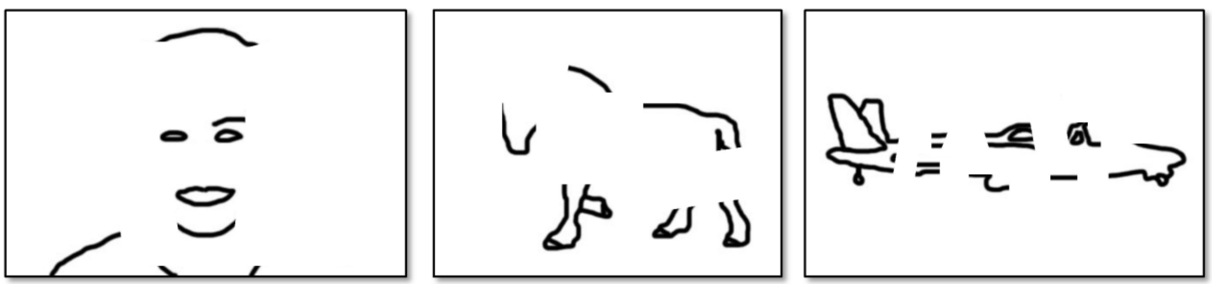
$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

Topics of This Lecture

- **Edge detection**
  - Recap: Gradients, scale influence
  - Canny edge detector
- Fitting as template matching
  - Distance transform
  - Chamfer matching
  - Application: traffic sign detection
- Fitting as parametric search
  - Line detection
  - Hough transform
  - Extension to circles
  - Generalized Hough transform

Edge Detection

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

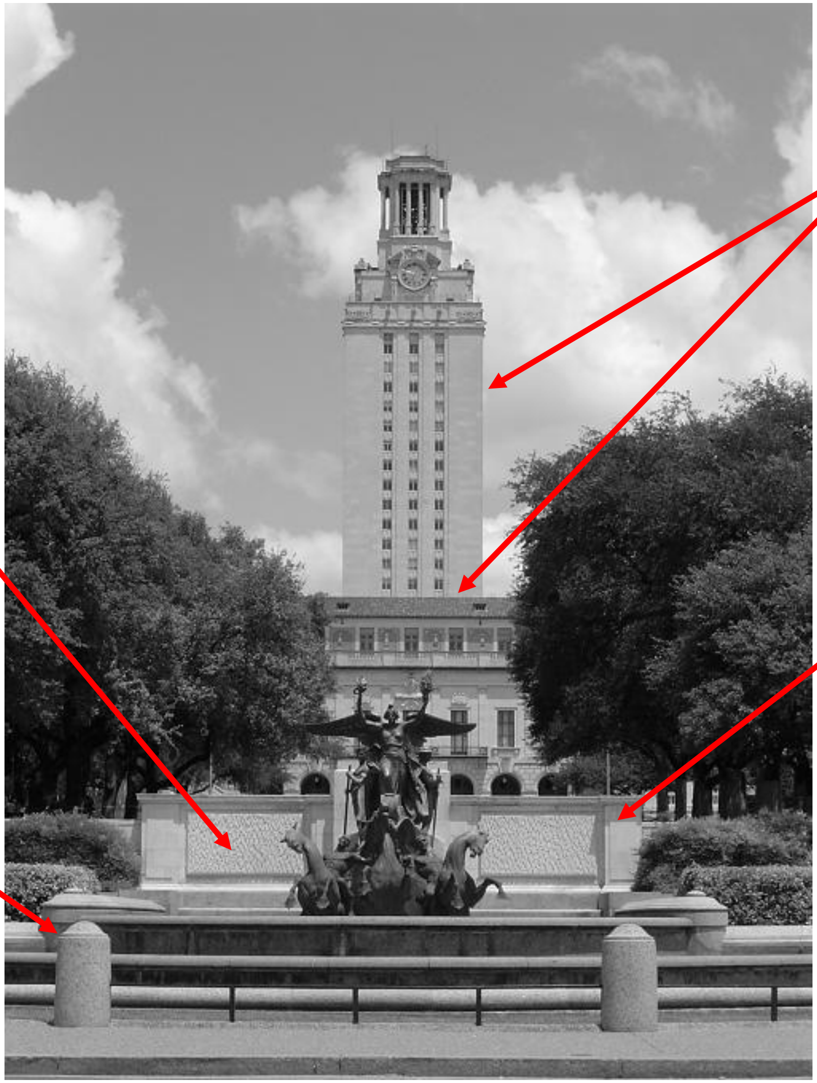


- Main idea: look for strong gradients, post-process

What Can Cause an Edge?

Reflectance change: appearance information, texture

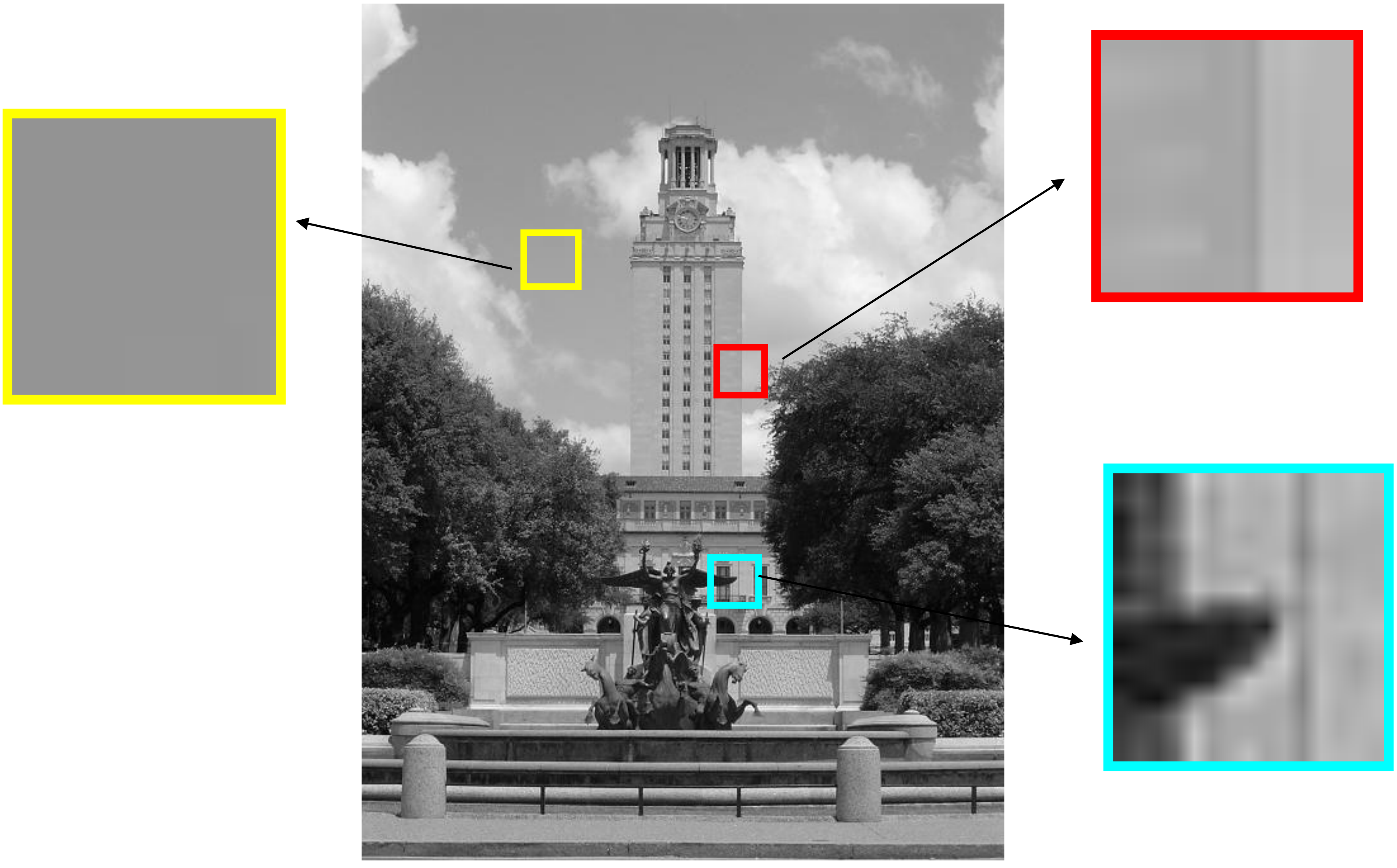
Change in surface orientation: shape



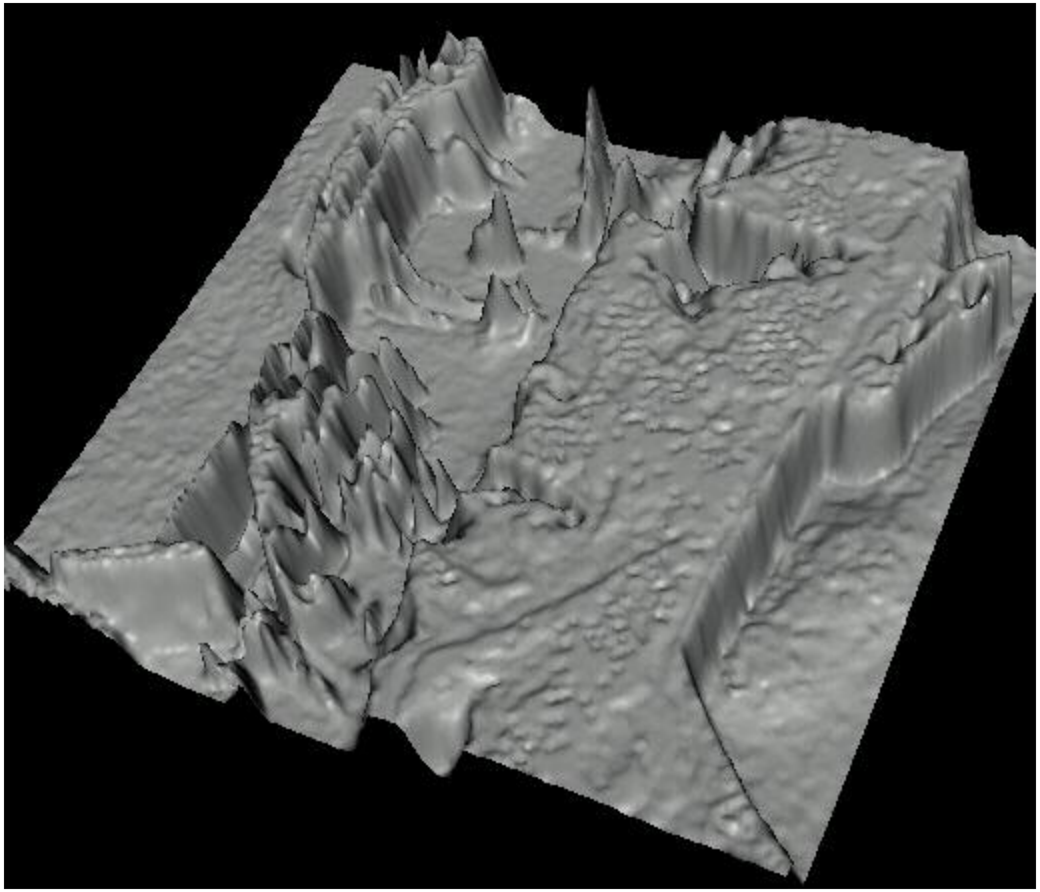
Depth discontinuity: object boundary

Cast shadows

Contrast and Invariance



Recall: Images as Functions



Edges look like steep cliffs

Gradients → Edges

Primary edge detection steps

1. Smoothing: suppress noise
2. Edge enhancement: filter for contrast
3. Edge localization
   - Determine which local maxima from filter output are actually edges vs. noise
   - Thresholding, thinning

Effect of σ on Derivatives

σ = 1 pixel

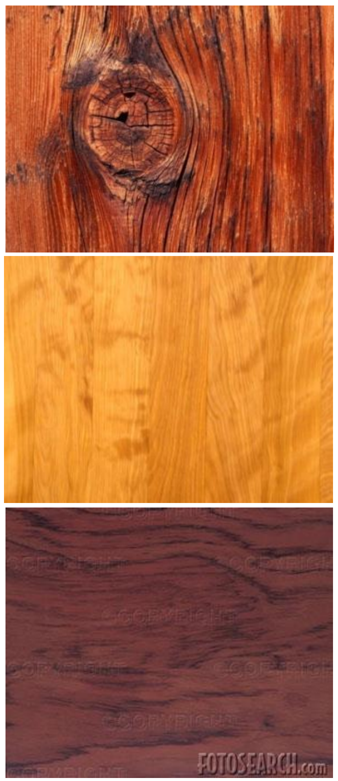
σ = 3 pixels

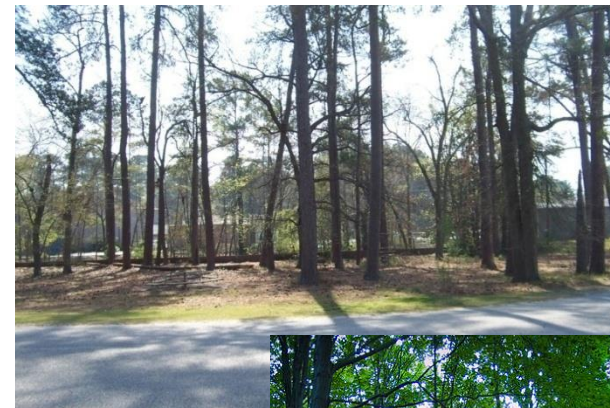
- **The apparent structures differ depending on Gaussian's scale parameter.**

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

So, What Scale to Choose?

- It depends on what we're looking for…



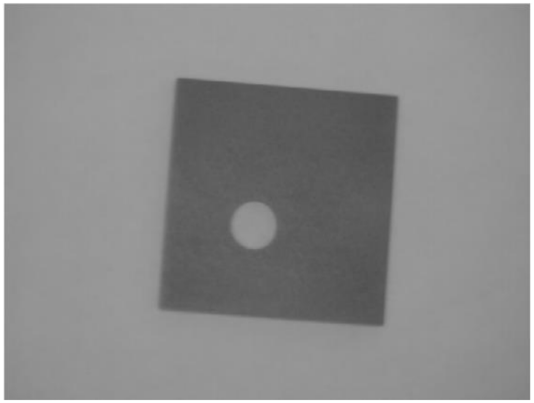



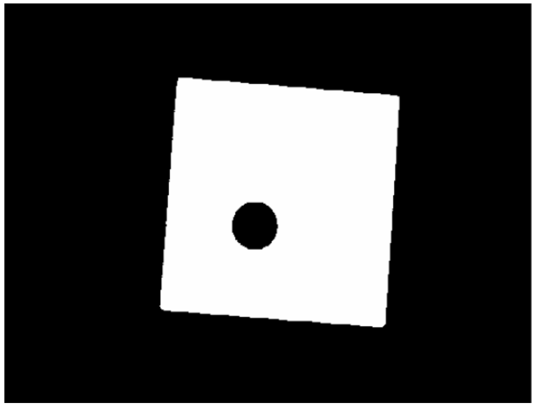

- Too fine a scale… can't see the forest for the trees.
- Too coarse a scale… can't tell the maple from the cherry.

Recall: Thresholding

- Choose a threshold t
- Set any pixels less than t to zero (off).
- Set any pixels greater than or equal t to one (on).

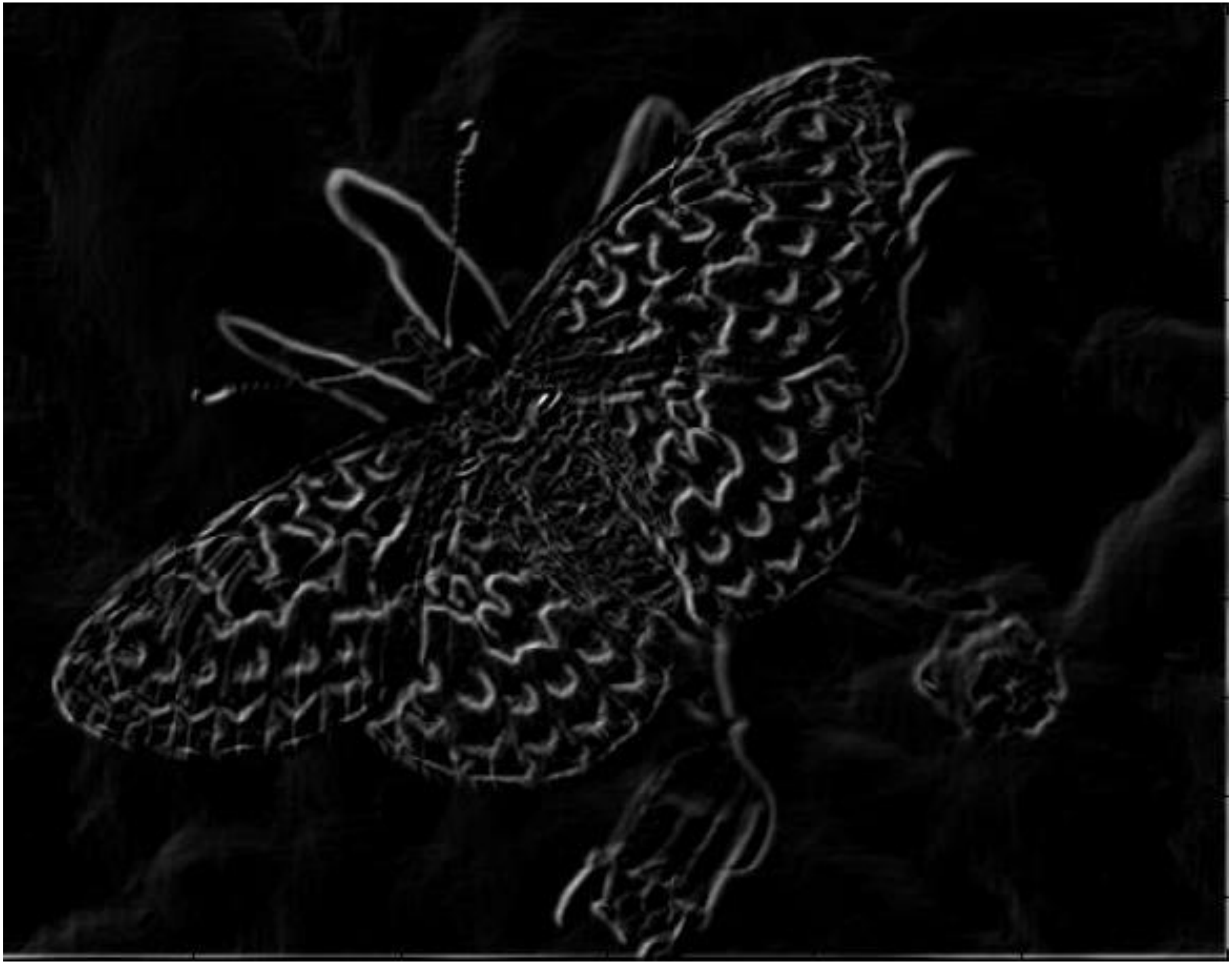
$$F_T[i,j] = \begin{cases} 1, & \text{if } F[i,j] \geq t \\ 0, & \text{otherwise} \end{cases}$$





Original Image

Gradient Magnitude Image

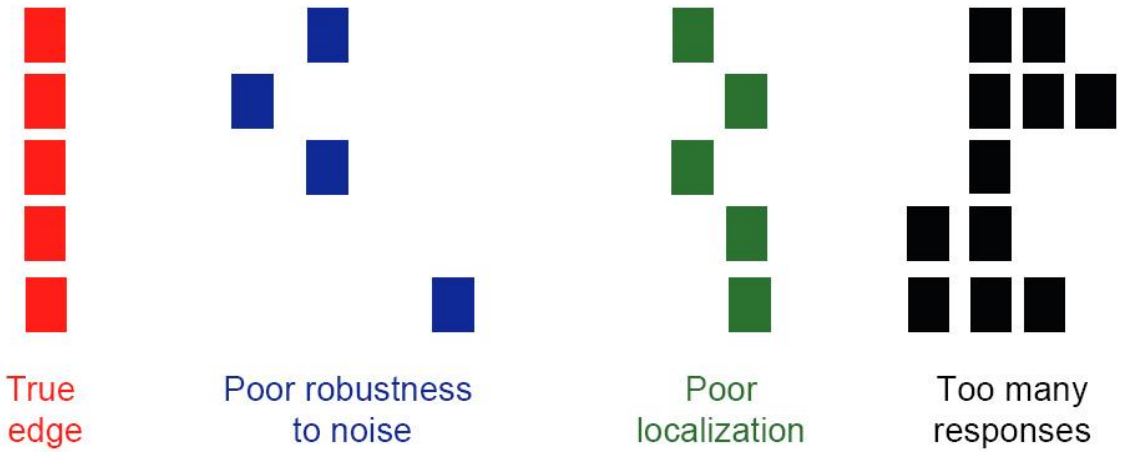


Thresholding with a lower threshold

Thresholding with a Higher Threshold

Designing an Edge Detector

- Criteria for an "optimal" edge detector:
  - Good detection: the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges)
  - Good localization: the edges detected must be as close as possible to the true edges
  - Single response: the detector must return one point only for each true edge point; that is, minimize the number of local maxima around the true edge



Canny Edge Detector

- This is probably the most widely used edge detector in computer vision
- Theoretical model: step-edges corrupted by additive Gaussian noise
- Canny has shown that the first derivative of the Gaussian closely approximates the operator that optimizes the product of *signal-to-noise ratio* and localization

Canny Edge Detector

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

- MATLAB:

```
>> edge(image, 'canny');
>> help edge
```

The Canny Edge Detector

original image (Lena)

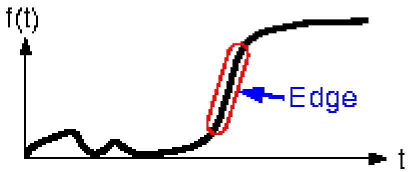
The Canny Edge Detector

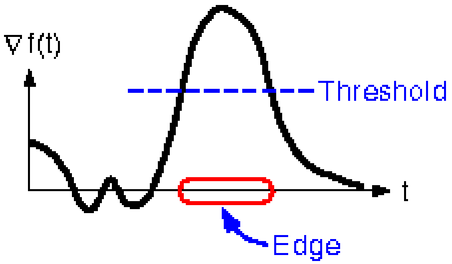
Norm of the gradient

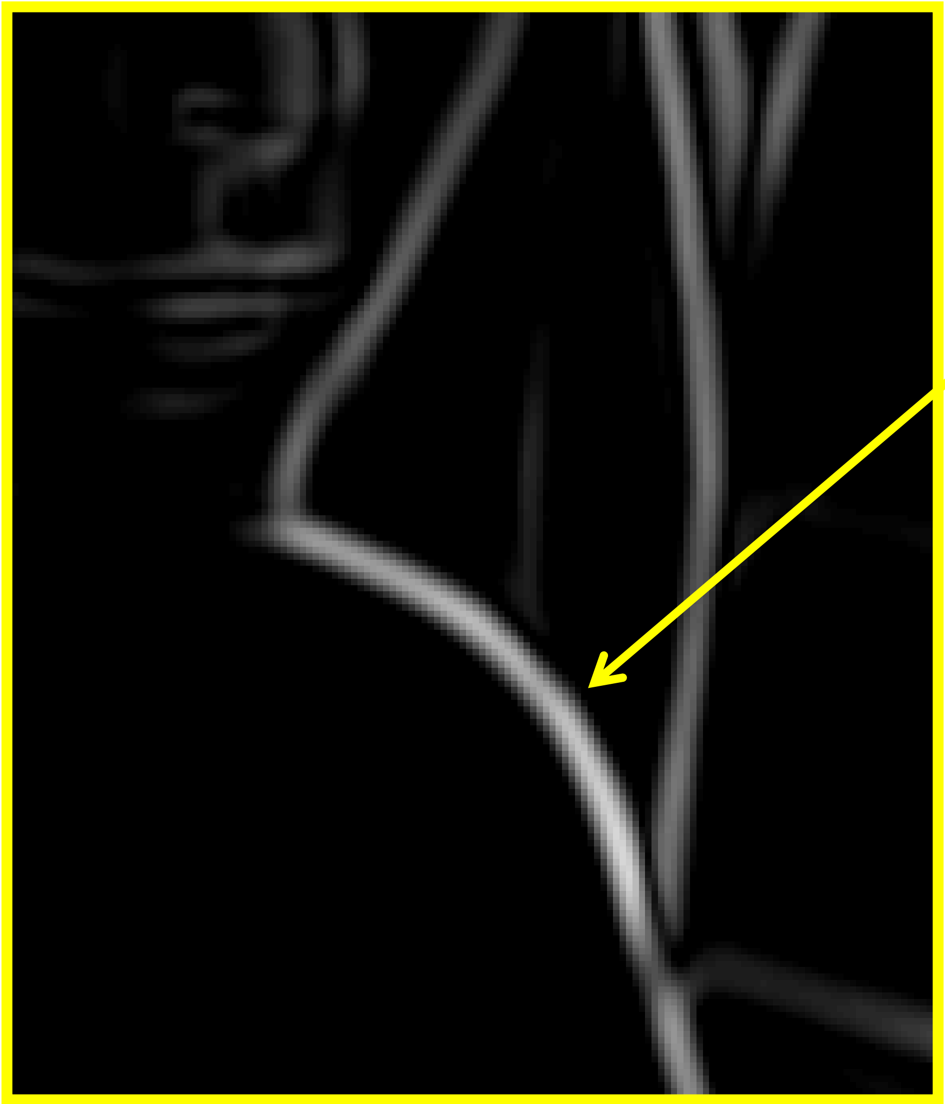
The Canny Edge Detector

Thresholding

The Canny Edge Detector

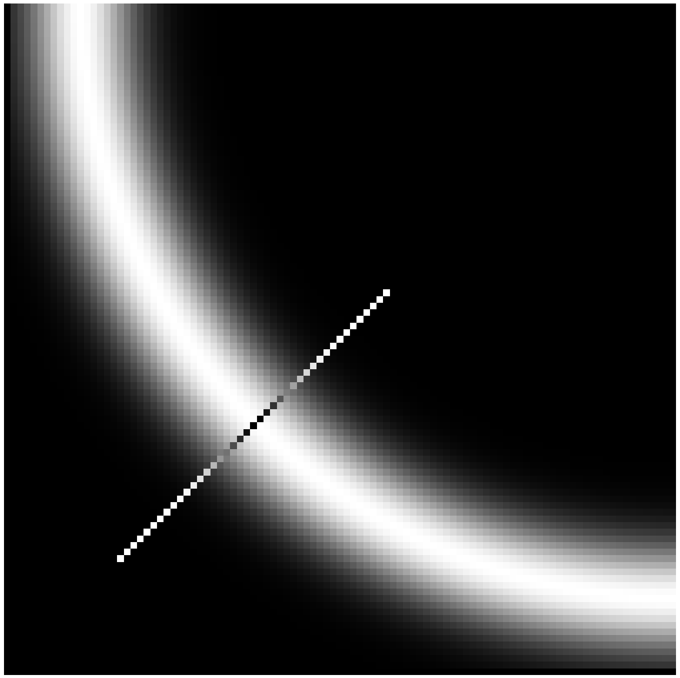


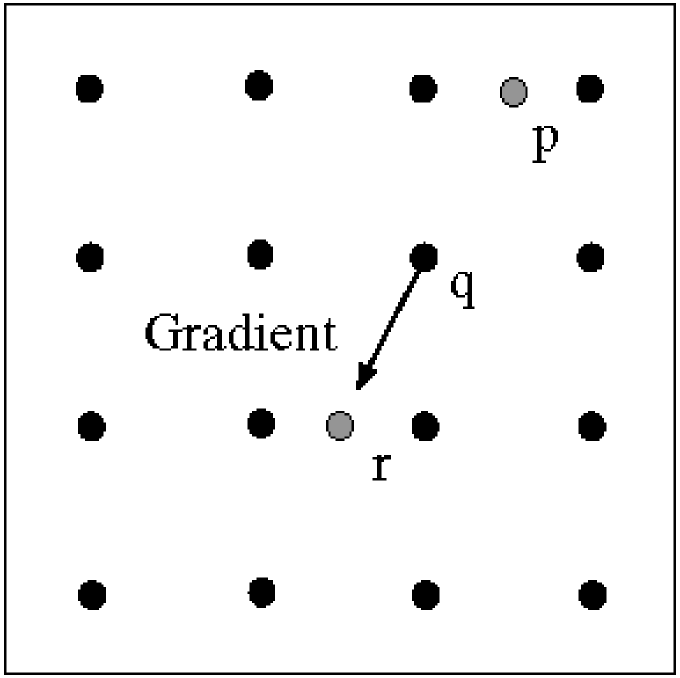




How to turn these thick regions of the gradient into curves?

Non-Maximum Suppression





- Check if pixel is local maximum along gradient direction, select single max across width of the edge
  - requires checking interpolated pixels p and r

The Canny Edge Detector

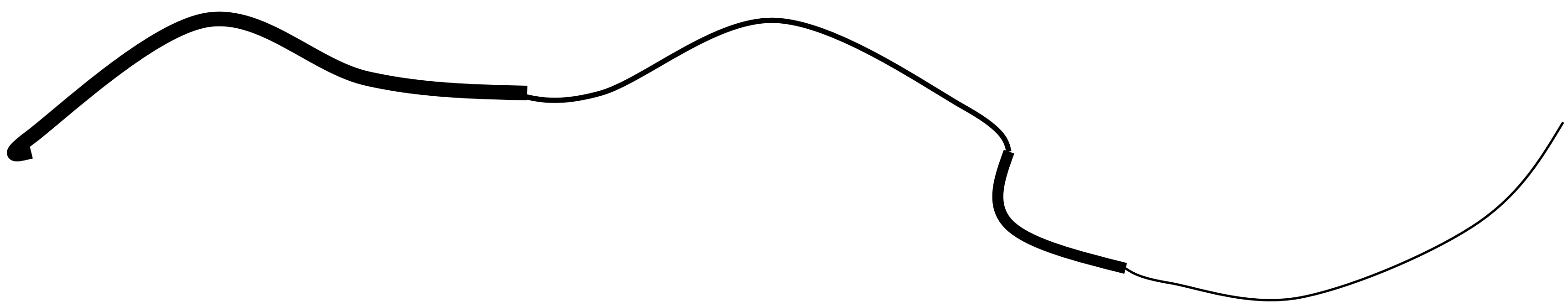
Problem: pixels along this edge didn't survive the thresholding

Thinning (non-maximum suppression)

Hysteresis Thresholding

- Hysteresis: A lag or momentum factor
- Idea: Maintain two thresholds k_{high} and k_{low}
  - Use k_{high} to find strong edges to start edge chain
  - Use k_{low} to find weak edges which continue edge chain
- Typical ratio of thresholds is roughly

$$k_{high} / k_{low} = 2$$



Hysteresis Thresholding

Original image

High threshold
(strong edges)

Low threshold
(weak edges)

courtesy of G. Loy

Hysteresis threshold

Object Boundaries vs. Edges

Background

Texture

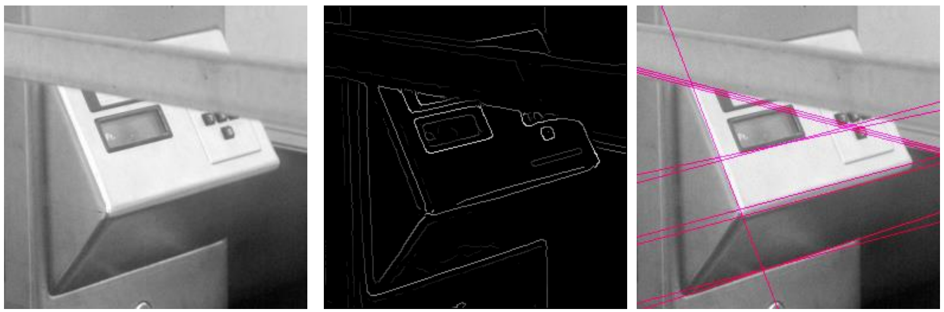
Shadows

Edge Detection is Just the Beginning…

Image | **Human segmentation** | **Gradient magnitude**

Fitting

- **Want to associate a model with observed features**



For example, the model could be a line, a circle, or an arbitrary shape.

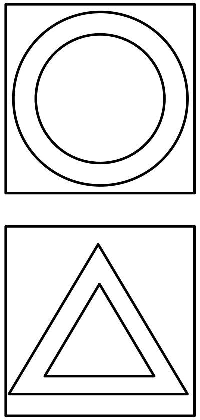
Topics of This Lecture

- Edge detection
  - Recap: Gradients, scale influence
  - Canny edge detector
- Fitting as template matching
  - Distance transform
  - Chamfer matching
  - Application: traffic sign detection
- Fitting as parametric search
  - Line detection
  - Hough transform
  - Extension to circles
  - Generalized Hough transform

Fitting as Template Matching

- We’ve already seen that correlation filtering can be used for template matching in an image.

- Let’s try this idea with “edge templates”.
  - Example: traffic sign detection in (grayvalue) video.



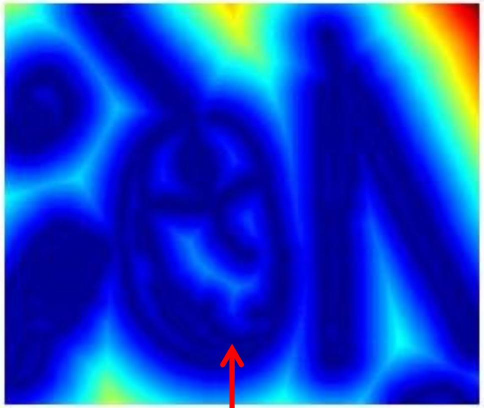
Templates

How Can This Be Made Efficient?

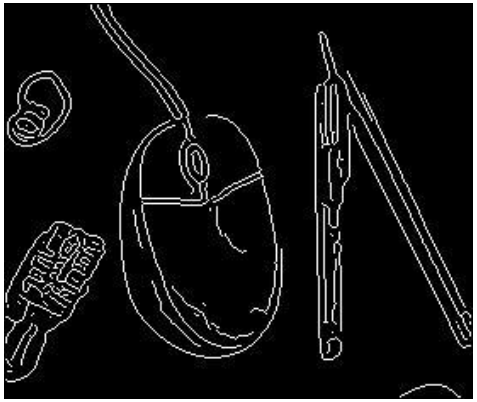
- Fast edge-based template matching
  - Distance transform of the edge image

Original

Gradient



Distance transform



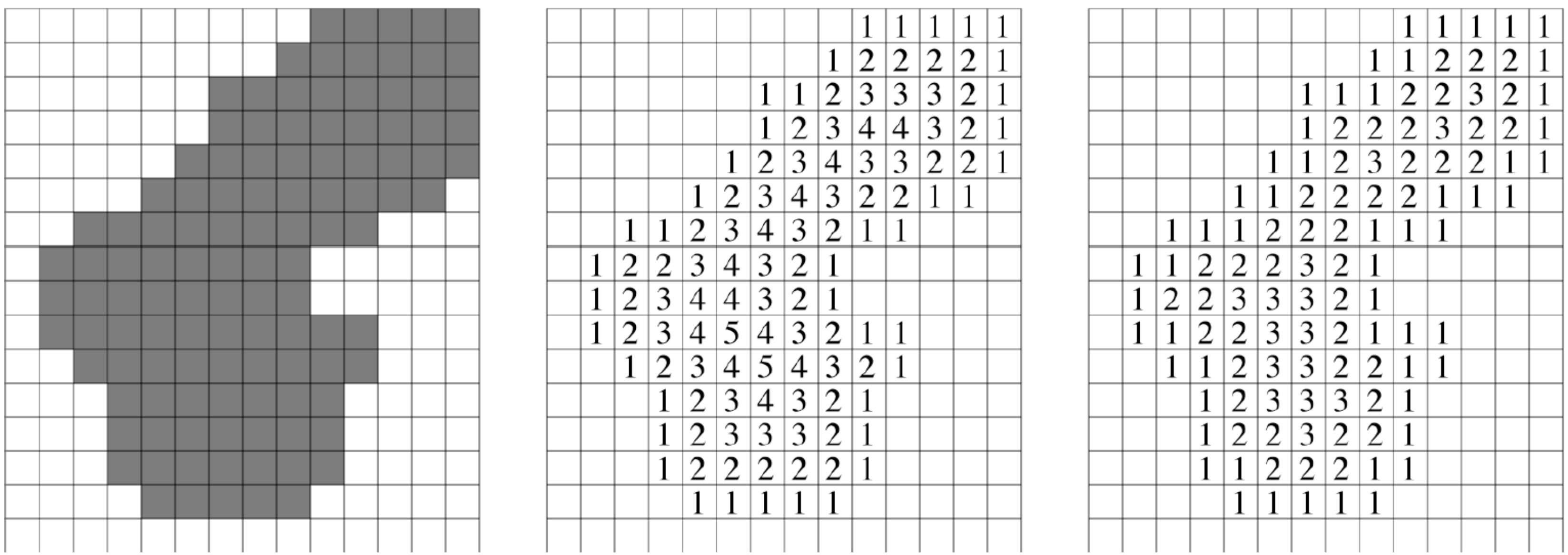
Edges

Value at (x,y) tells how far that position is from the nearest edge point (or other binary mage structure)

```
>> help bwdist
```

Distance Transform

- **Image reflecting distance to nearest point in point set (e.g., edge pixels, or foreground pixels).**



4-connected adjacency

8-connected adjacency

Distance Transform Algorithm (1D)

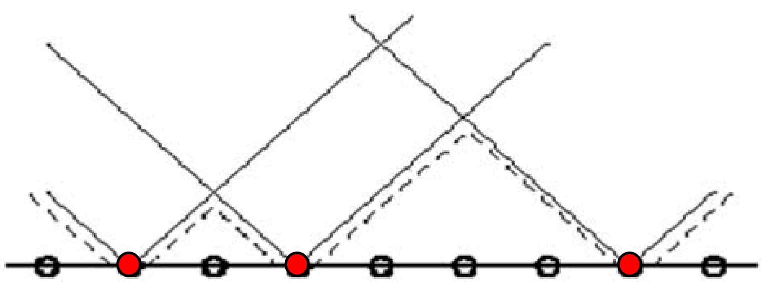
- Two-pass O(n) algorithm for 1D L_1 norm

1. <u>Initialize:</u> For all j
   - $D[j] \leftarrow 1_P[j]$ // 0 if j is in **P**, infinity otherwise
2. <u>Forward:</u> For j from 1 up to n-1
   - $D[j] \leftarrow \min(\ D[j],\ D[j-1]+1\ )$

1	0

3. <u>Backward:</u> For j from n-2 down to 0
   - $D[j] \leftarrow \min(\ D[j],\ D[j+1]+1\ )$

0	1



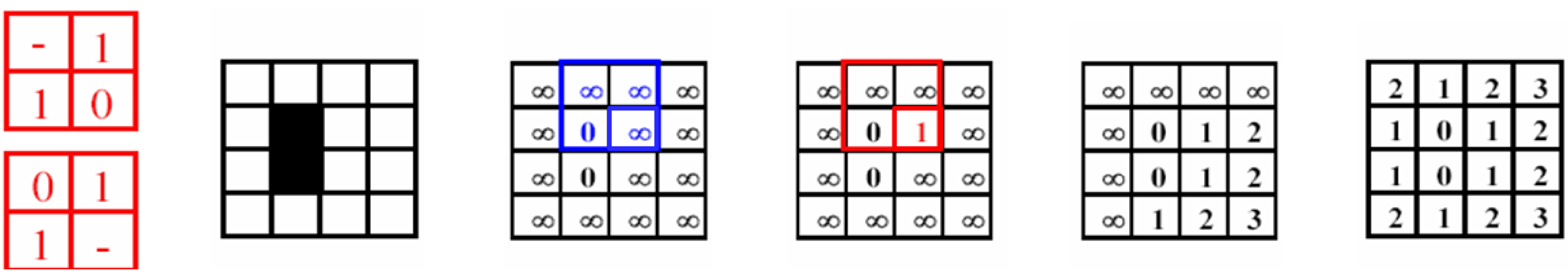
∞	0	∞	0	∞	∞	∞	0	∞

∞	0	1	0	1	2	3	0	1

1	0	1	0	1	2	1	0	1

Distance Transform Algorithm (2D)

- 2D case analogous to 1D
  - Initialization
  - Forward and backward pass
    - Fwd pass finds closest above and to the left
    - Bwd pass finds closest below and to the right

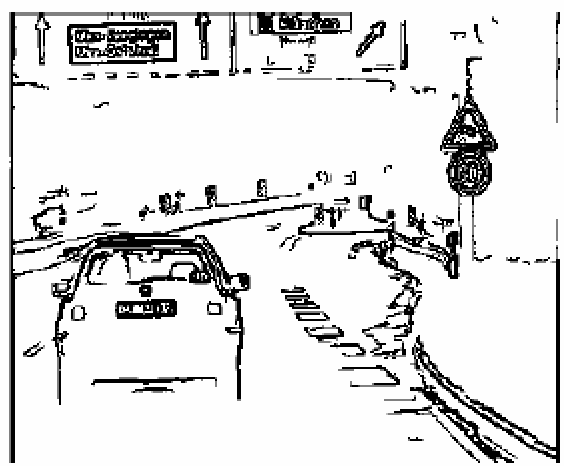


Chamfer Matching

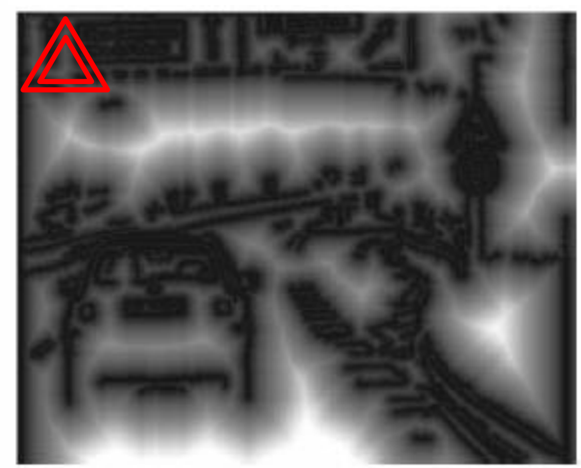
- **Chamfer Distance**
  - Average distance to nearest feature

$$D_{chamfer}(T,I) \equiv \frac{1}{|T|}\sum_{t\in T} d_I(t)$$

  - This can be computed efficiently by correlating the edge template with the distance-transformed image



Edge image



Distance transform image

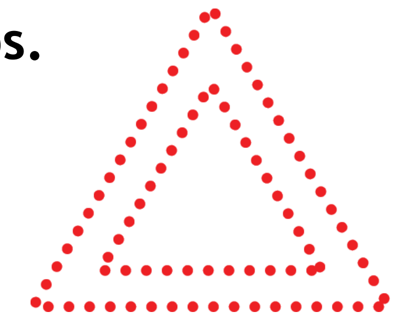
Chamfer Matching

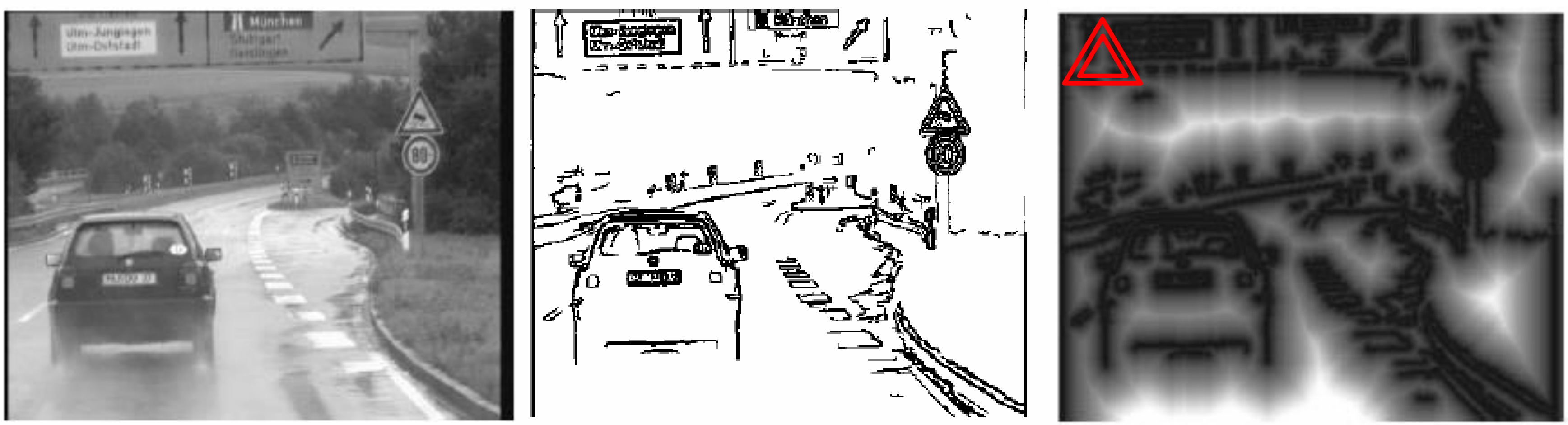
- **Efficient implementation**
  - Instead of correlation, sample fixed number of points on template contour.

  $\Rightarrow$ Chamfer score boils down to series of DT lookups.

$$D_{chamfer}(T, I) \equiv \frac{1}{|T|} \sum_{t \in T} d_I(t)$$

  $\Rightarrow$ Computational effort independent of scale.

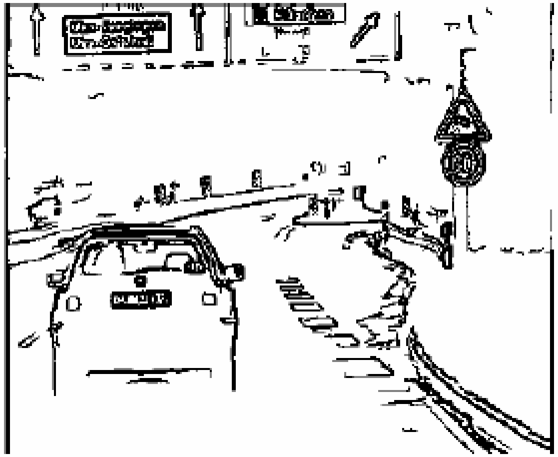




Edge image

Distance transform image

Chamfer Matching Results

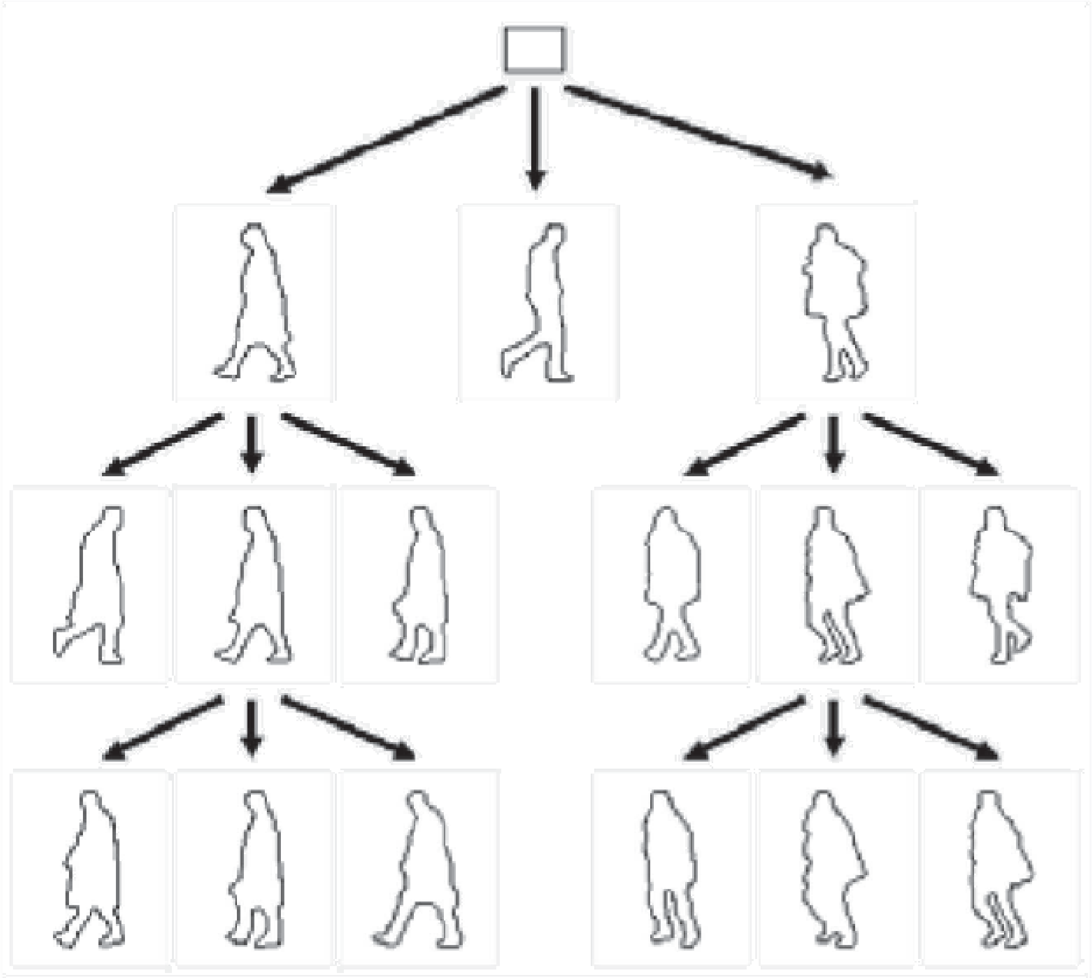


Edge image

Distance transform image

Chamfer Matching for Pedestrian Detection

- **Organize templates in tree structure for fast matching**

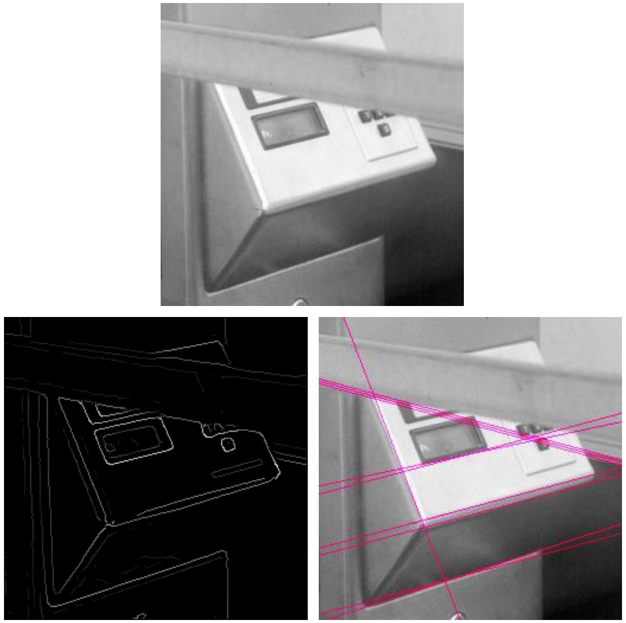


Summary Chamfer Matching

- **Pros**
  - Fast and simple method for matching edge-based templates.
  - Works well for matching upright shapes with little intra-class variation.
  - Good method for finding candidate matches in a longer recognition pipeline.
- **Cons**
  - Chamfer score averages over entire contour, not very discriminative in practice.
    $\Rightarrow$ Further verification needed.
  - Low matching cost in cluttered regions with many edges.
    $\Rightarrow$ Many false positive detections.
  - In order to detect rotated & rescaled shapes, need to match with rotated & rescaled templates $\Rightarrow$ can get very expensive.

Topics of This Lecture

- Edge detection
  - Recap: Gradients, scale influence
  - Canny edge detector
- Fitting as template matching
  - Distance transform
  - Chamfer matching
  - Application: traffic sign detection
- **Fitting as parametric search**
  - **Line detection**
  - **Hough transform**
  - **Extension to circles**
  - **Generalized Hough transform**

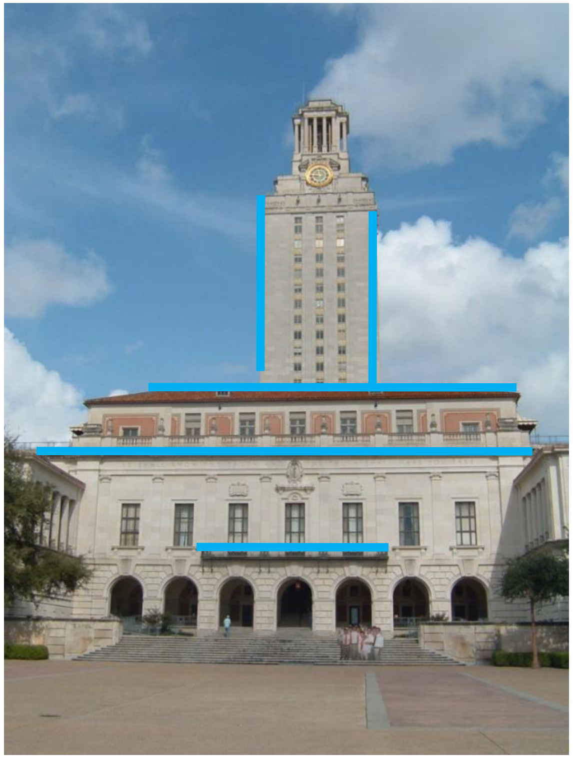


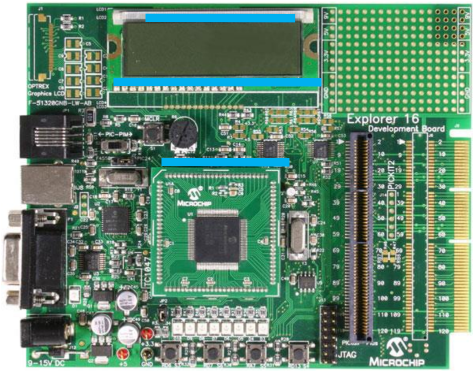
Fitting as Search in Parametric Space

- Choose a parametric model to represent a set of features
- Membership criterion is not local
  - Can't tell whether a point belongs to a given model just by looking at that point.
- Three main questions:
  - What model represents this set of features best?
  - Which of several model instances gets which feature?
  - How many model instances are there?
- Computational complexity is important
  - It is infeasible to examine every possible set of parameters and every possible combination of features

Example: Line Fitting

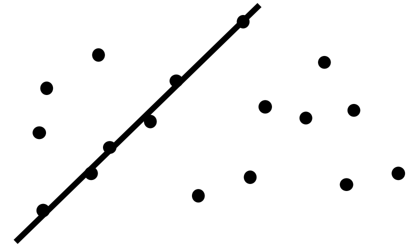
- Why fit lines?
  Many objects characterized by presence of straight lines

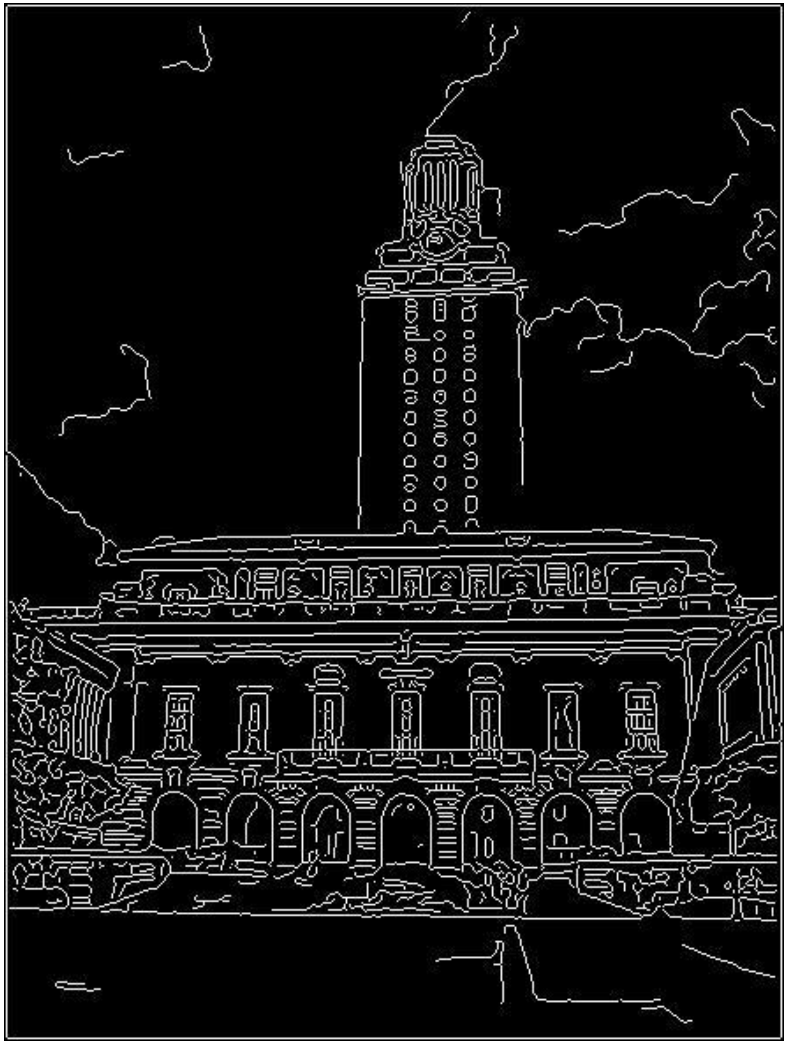




- Wait, why aren’t we done just by running edge detection?

Difficulty of Line Fitting





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

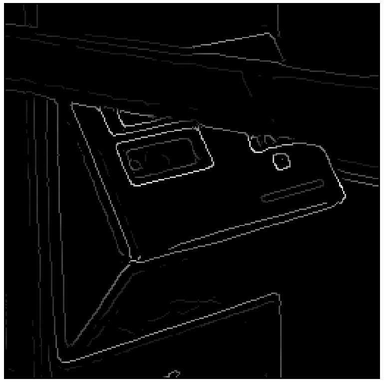
Voting

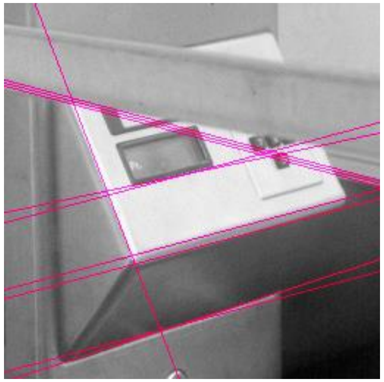
- It's not feasible to check all combinations of features by fitting a model to each possible subset.
- Voting is a general technique where we let the features vote for all models that are compatible with it.
  - Cycle through features, cast votes for model parameters.
  - Look for model parameters that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

Fitting Lines

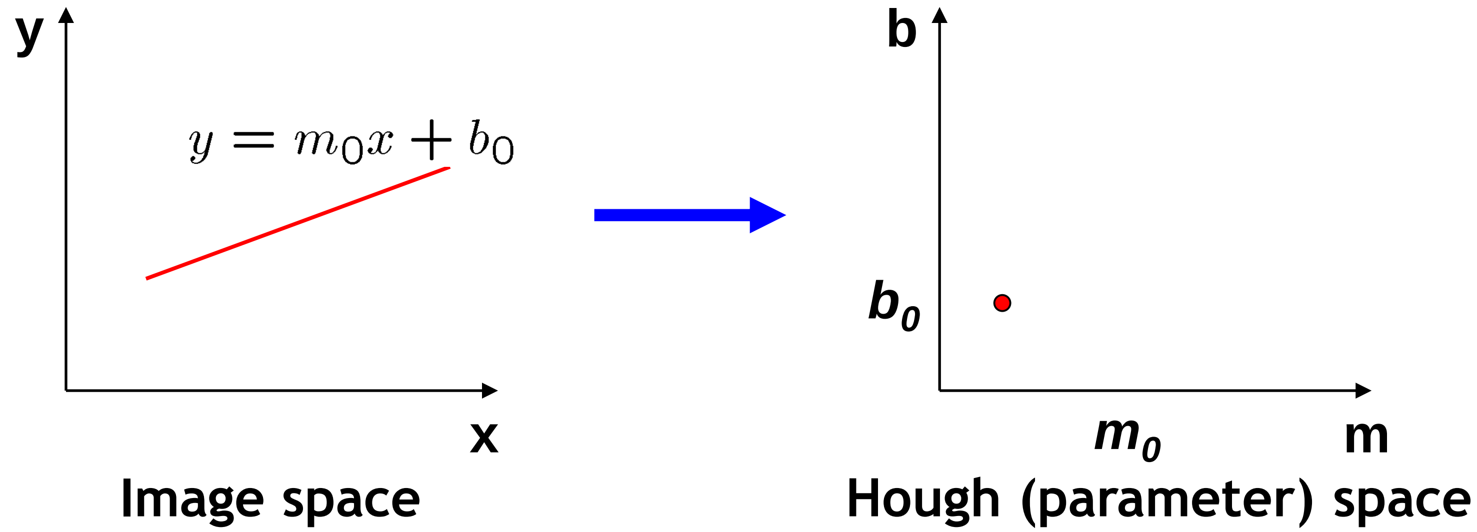
- Given points that belong to a line, what is the line?
- How many lines are there?
- Which points belong to which lines?

- *Hough Transform* is a voting technique that can be used to answer all of these
- Main idea:
  1. Record all possible lines on which each edge point lies.
  2. Look for lines that get many votes.





Finding Lines in an Image: Hough Space



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

Finding Lines in an Image: Hough Space

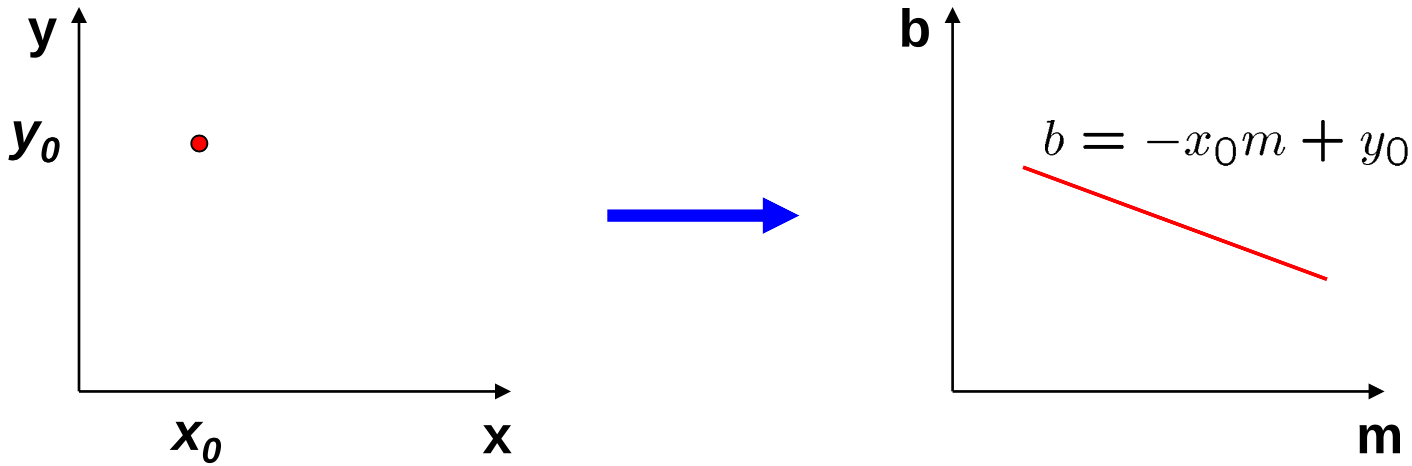
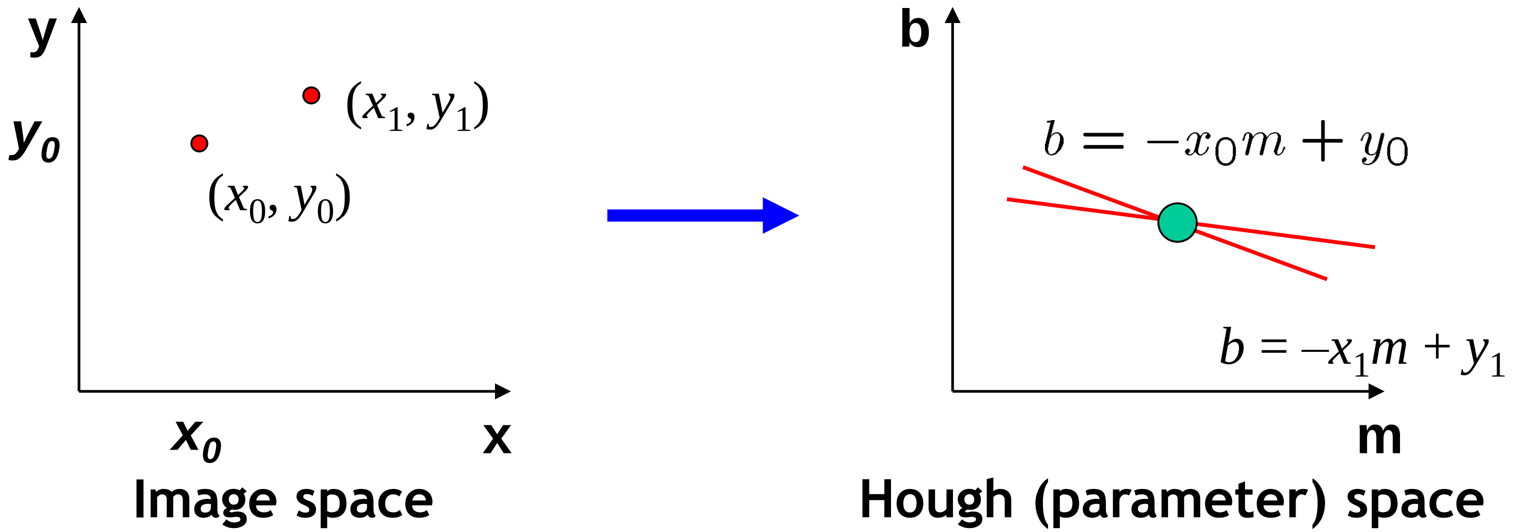


Image space

Hough (parameter) space

- Connection between image (x,y) and Hough (m,b) spaces
  - A line in the image corresponds to a point in Hough space.
  - To go from image space to Hough space:
    - Given a set of points (x,y), find all (m,b) such that $y = mx + b$
  - What does a point (x_0, y_0) in the image space map to?
    - Answer: the solutions of $b = -x_0m + y_0$
    - This is a line in Hough space

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 $b = -x_0 m + y_0$ and $b = -x_1 m + y_1$**

Finding Lines in an Image: Hough Space

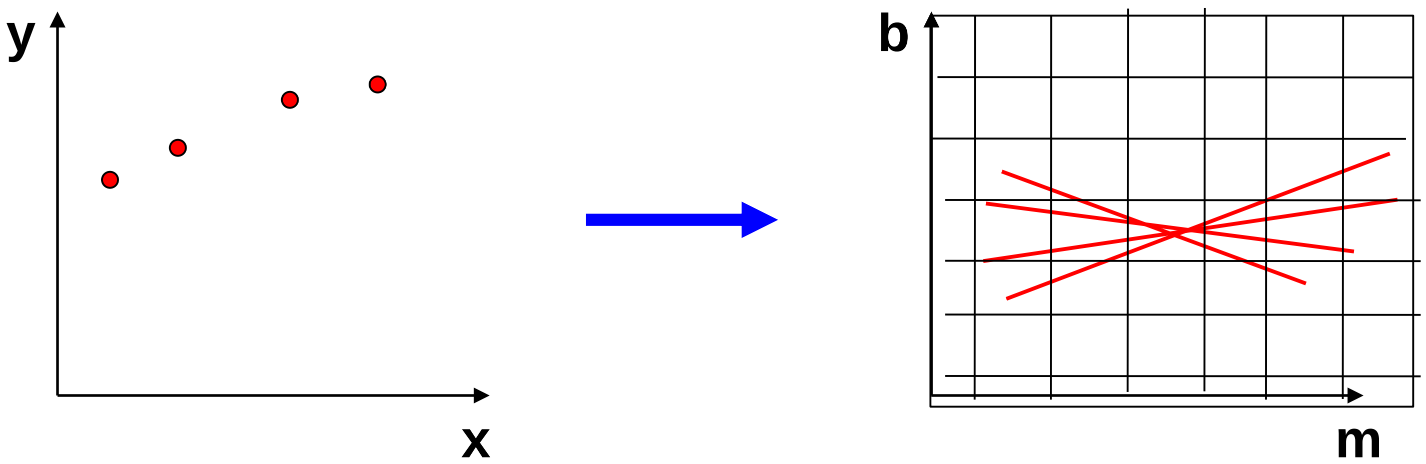


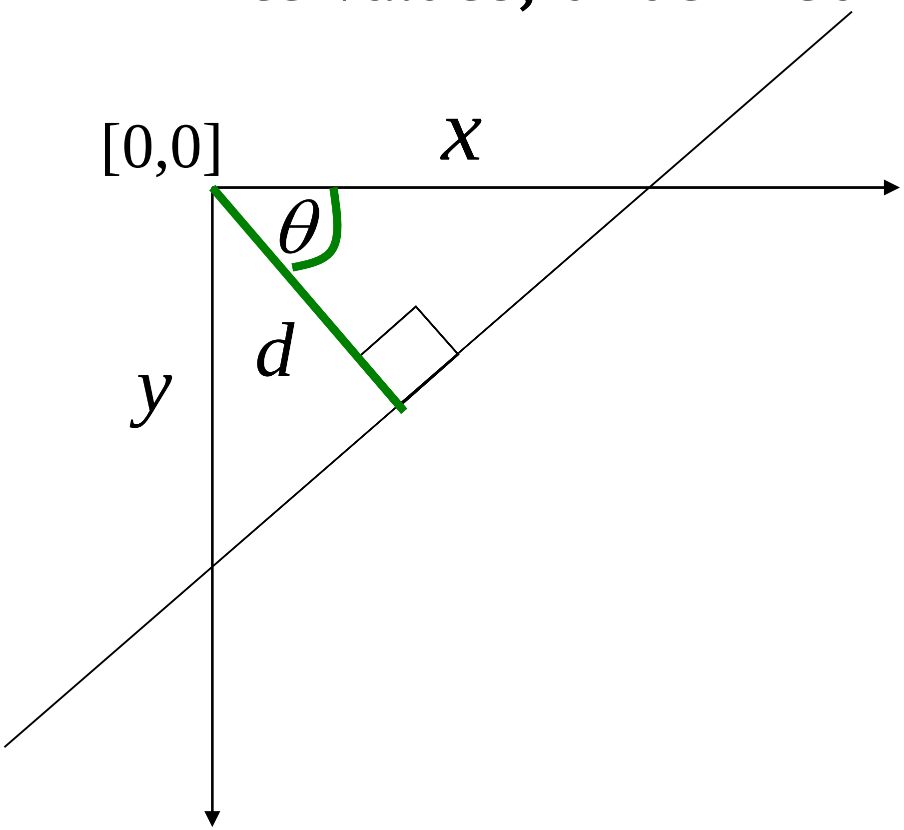
Image space

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.

Polar Representation for Lines

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



d : perpendicular distance from line to origin

θ : angle the perpendicular makes with the x-axis

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

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

Hough Transform Algorithm

Using the polar parameterization:

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

Basic Hough transform algorithm

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

   for $\theta = 0$ **to** 180 // some quantization

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

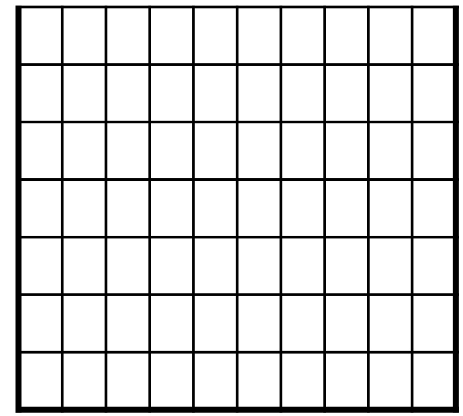
   H[d, θ] += 1

3. **Find the value(s) of** (d,θ) **where** $H[d,\theta]$ **is maximum.**
4. **The detected line in the image is given by** $d = x\cos\theta - y\sin\theta$

[**Hough line demo**]

H: accumulator array (votes)

θ



d

- **Time complexity (in terms of number of votes)?**

Example: HT for Straight Lines

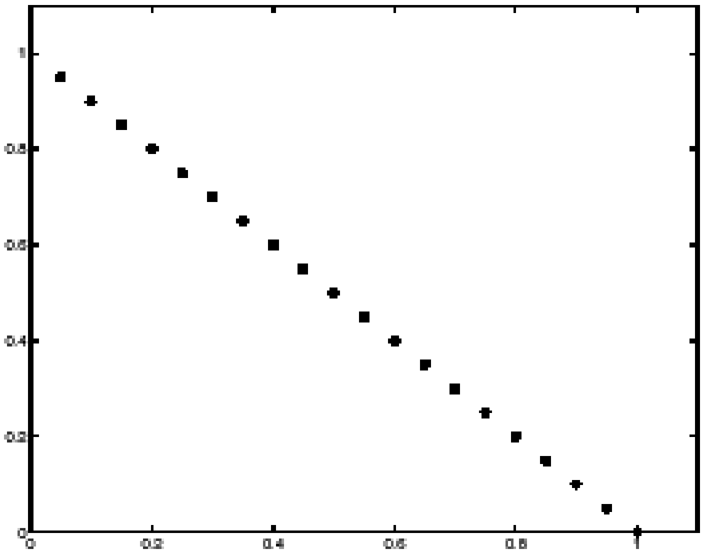
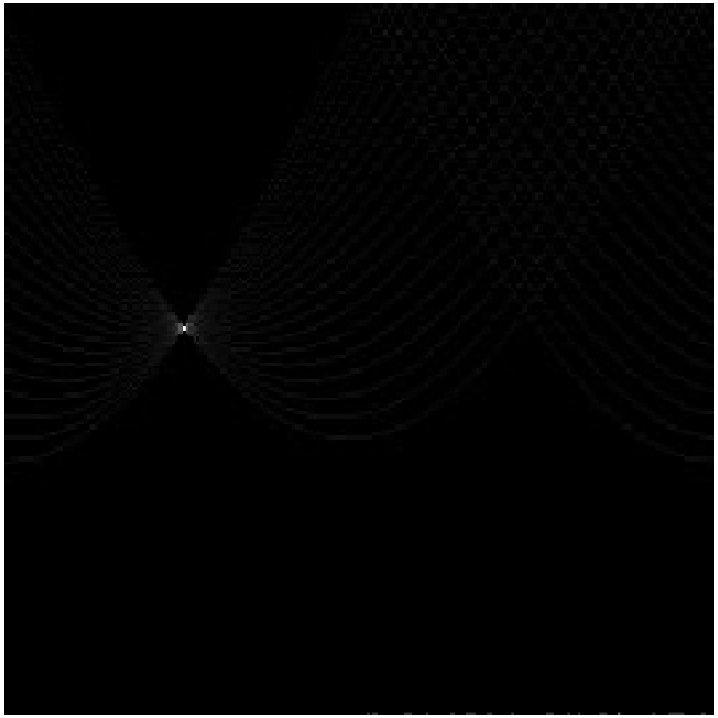


Image space
edge coordinates



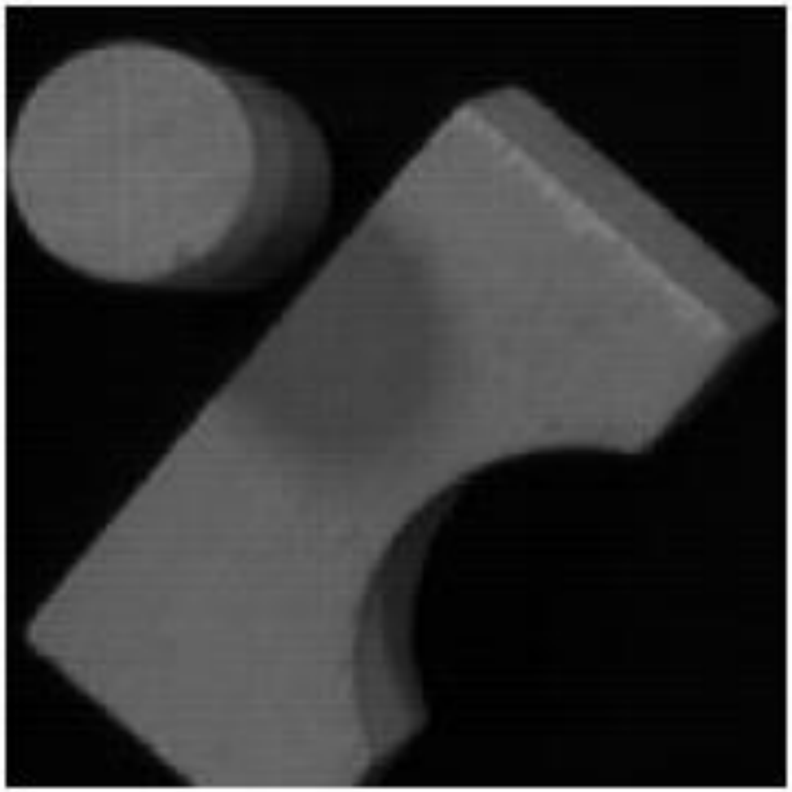
Votes

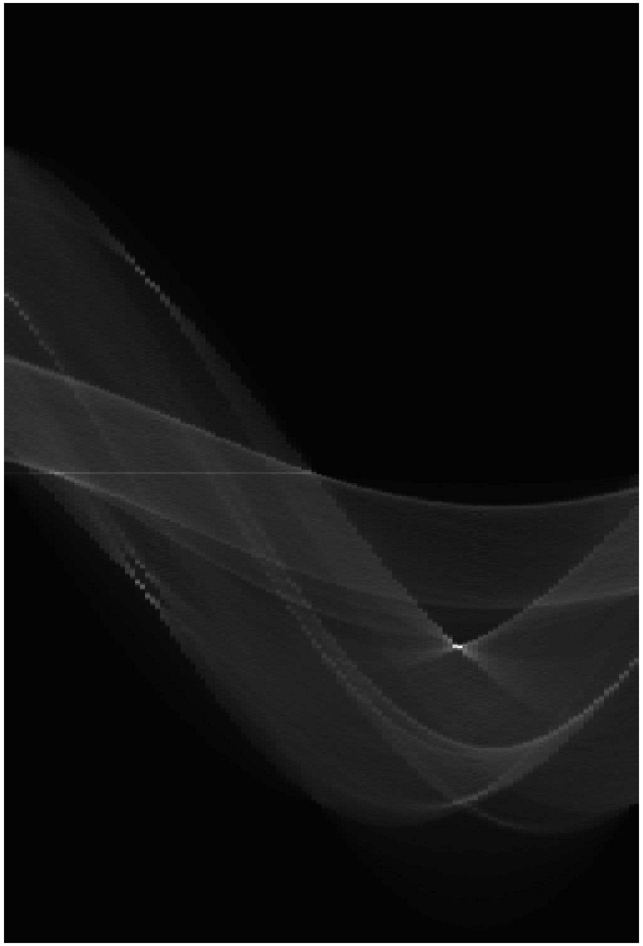
Bright value = high vote count
Black = no votes

Example: HT for Straight Lines

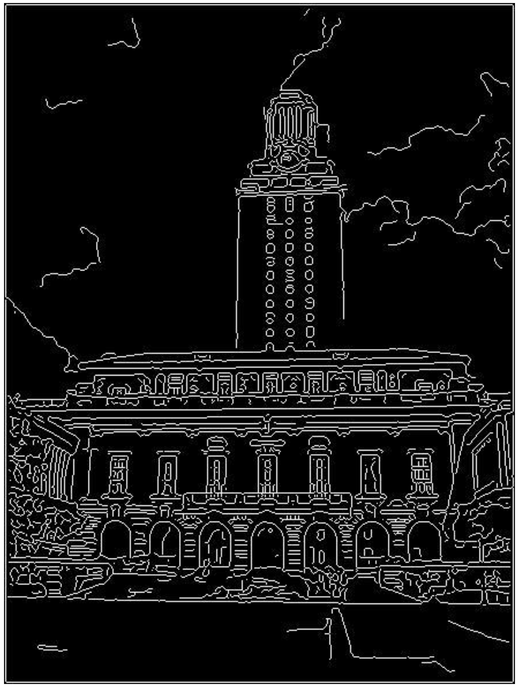
Square:

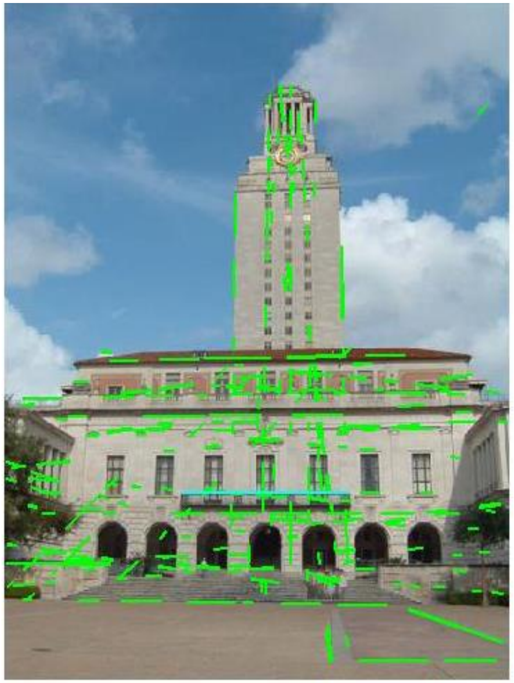
Example: HT for Straight Lines

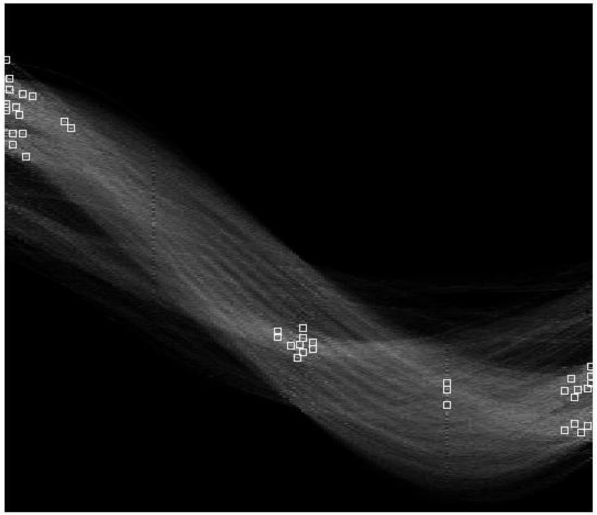


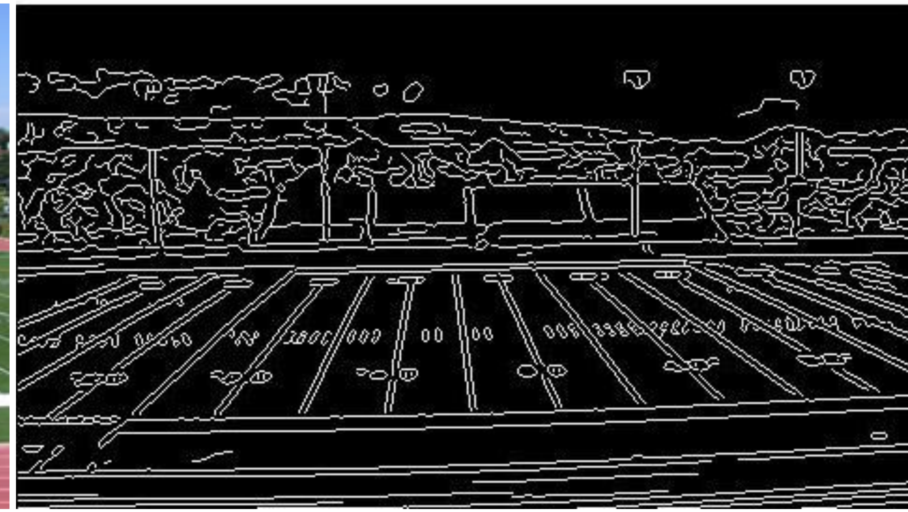


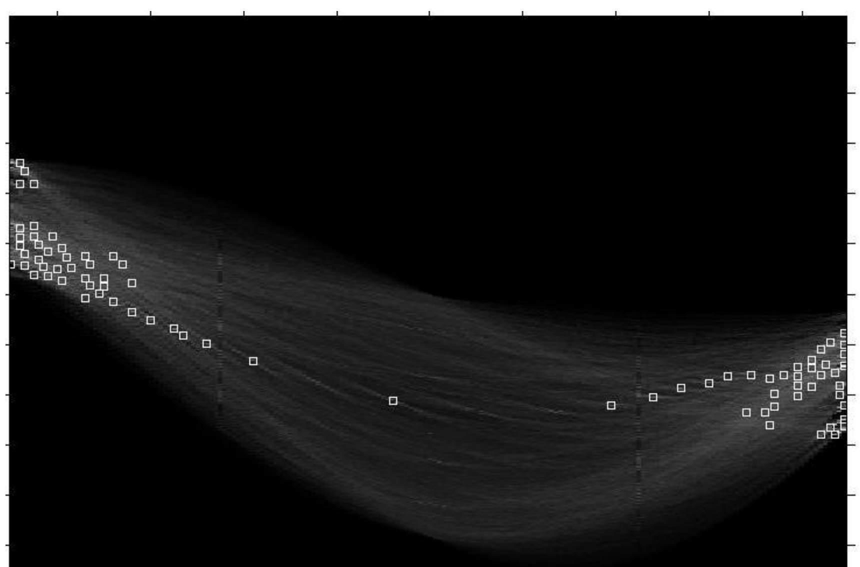
Real-World Examples

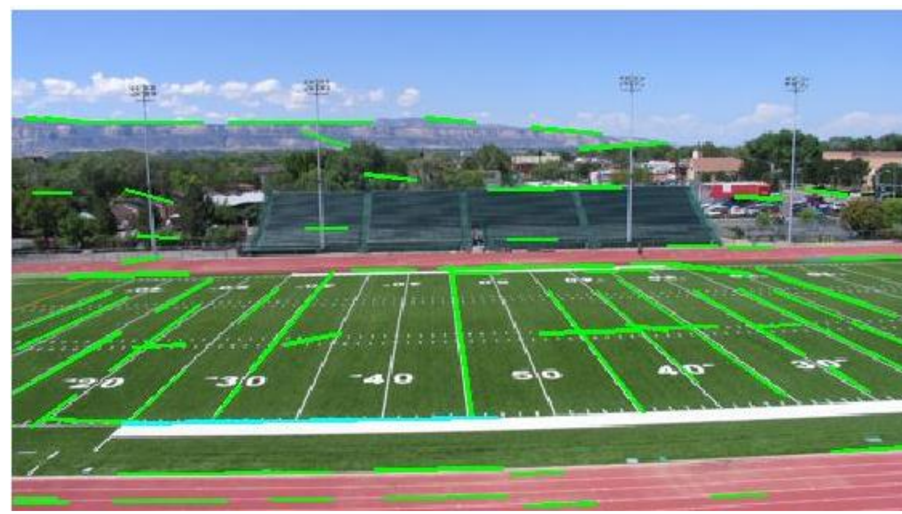












Showing longest segments found

Impact of Noise on Hough Transform

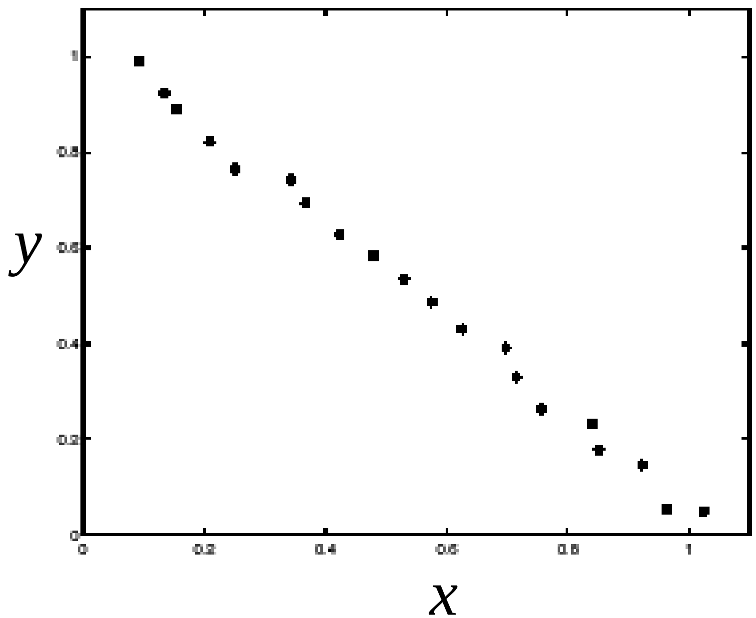
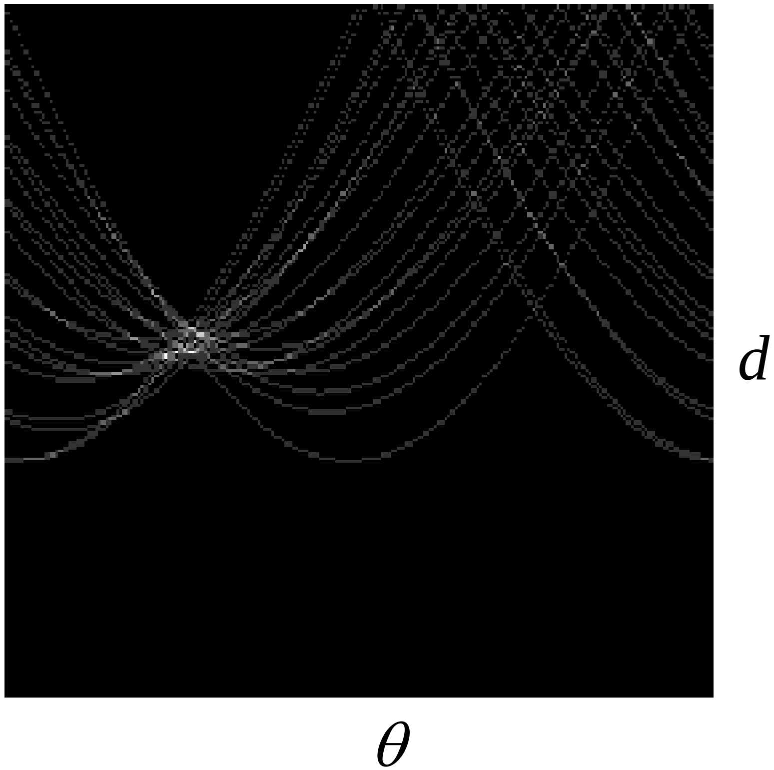


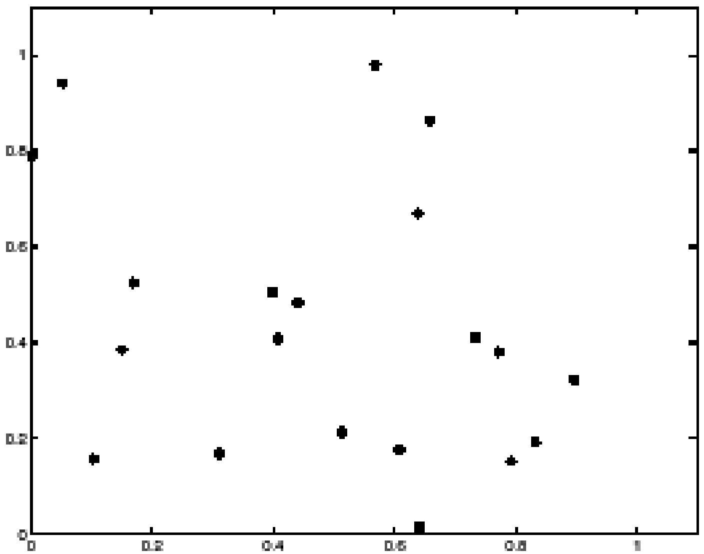
Image space edge coordinates



Votes

What difficulty does this present for an implementation?

Impact of Noise on Hough Transform



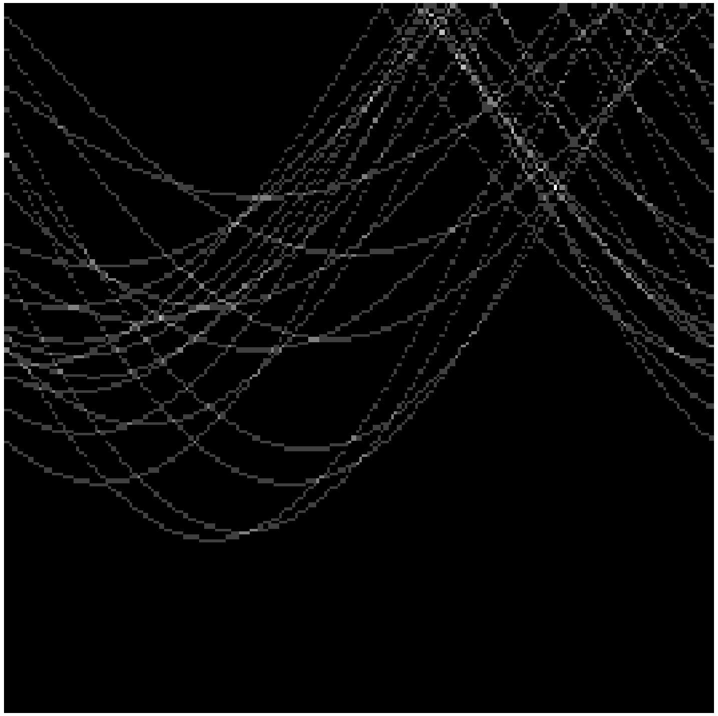
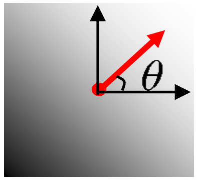


Image space
edge coordinates

Votes

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

Extensions



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

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

Extension 1: Use the image gradient

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

   θ = gradient at (x,y)

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

   $H[d,\theta]$ += 1
3. same
4. same

(Reduces degrees of freedom)

Extensions

Extension 1: Use the image gradient

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

   compute unique (d,θ) based on image gradient at (x,y)

   $H[d,\theta]$ += 1
3. same
4. same

(Reduces degrees of freedom)

Extension 2

- Give more votes for stronger edges (use magnitude of gradient)

Extension 3

- Change the sampling of (d,θ) to give more/less resolution

Extension 4

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

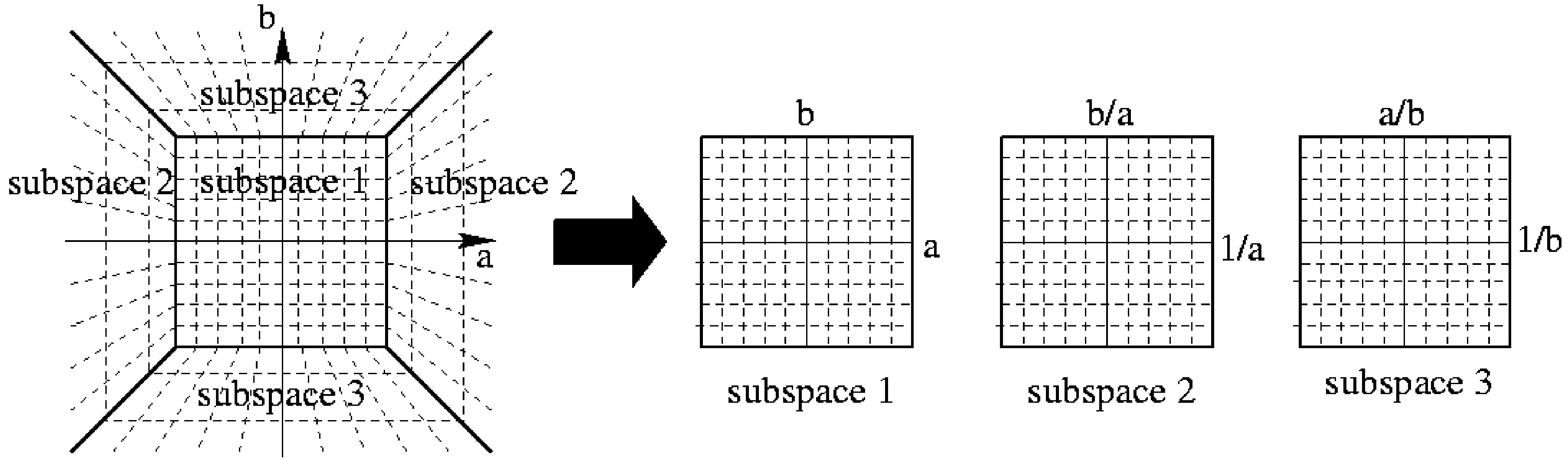
Extension: Cascaded Hough Transform

- Let's go back to the original (m,b) parametrization
- A line in the image maps to a pencil of lines in the Hough space
- What do we get with parallel lines or a pencil of lines?
  - Collinear peaks in the Hough space!
- So we can apply a Hough transform to the output of the first Hough transform to find vanishing points

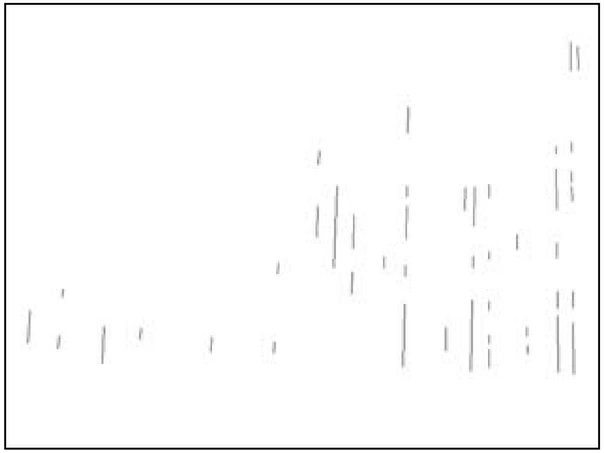
Finding Vanishing Points

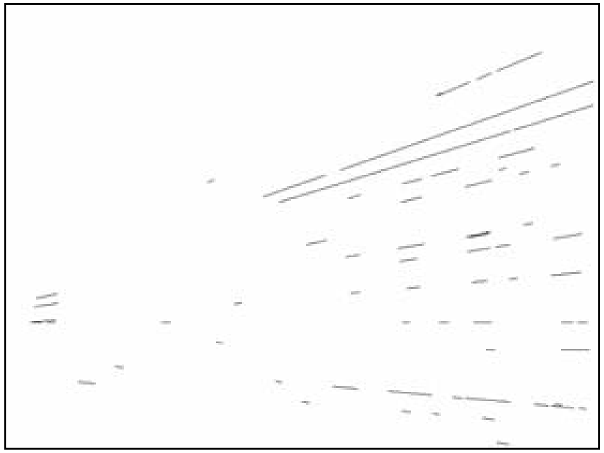
Cascaded Hough Transform

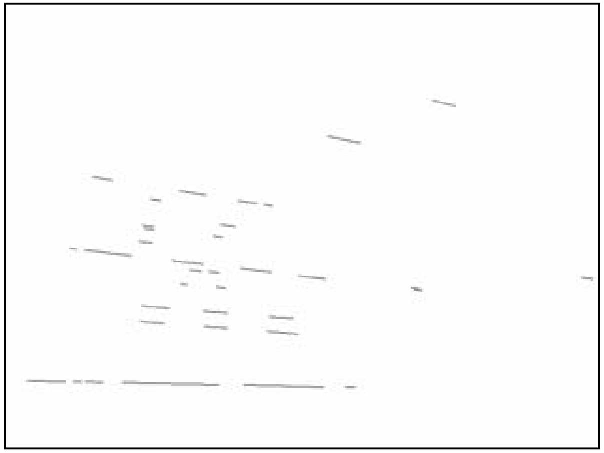
- **Issue: Dealing with the unbounded parameter space**



Cascaded Hough Transform







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**

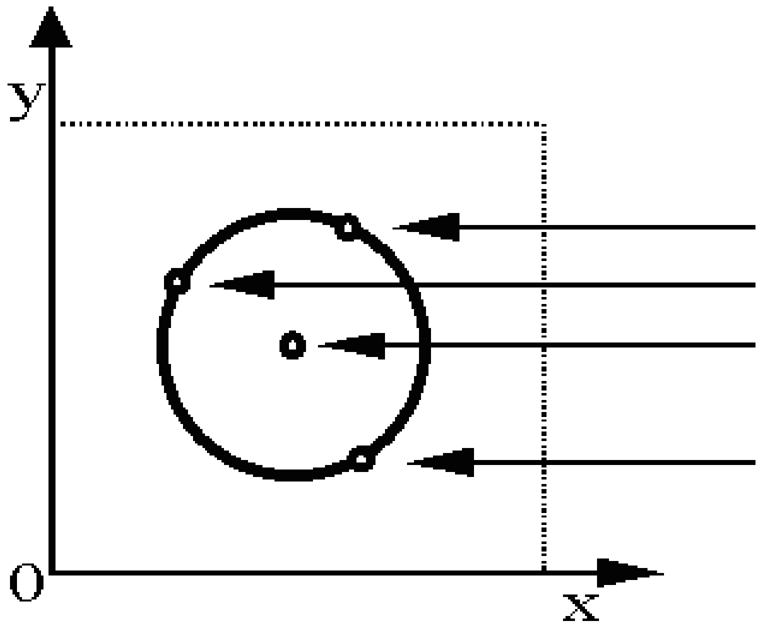
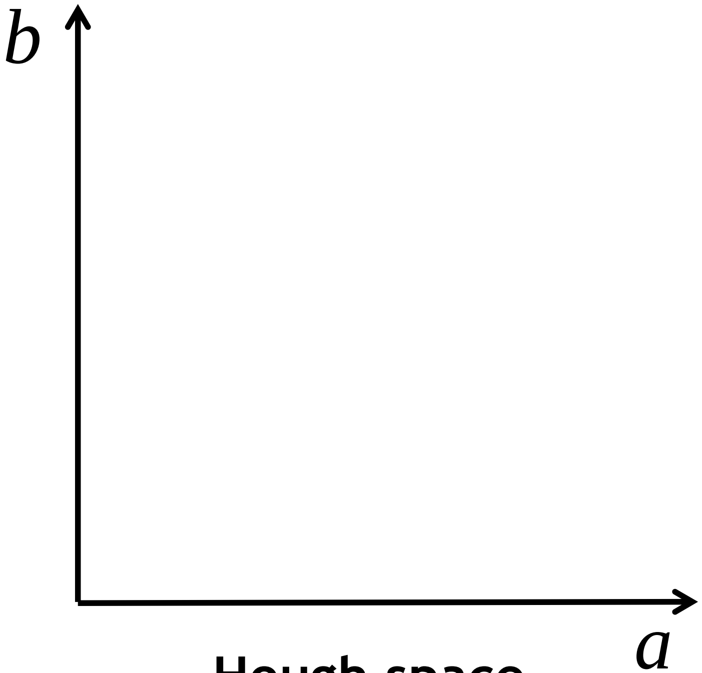


Image space



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**

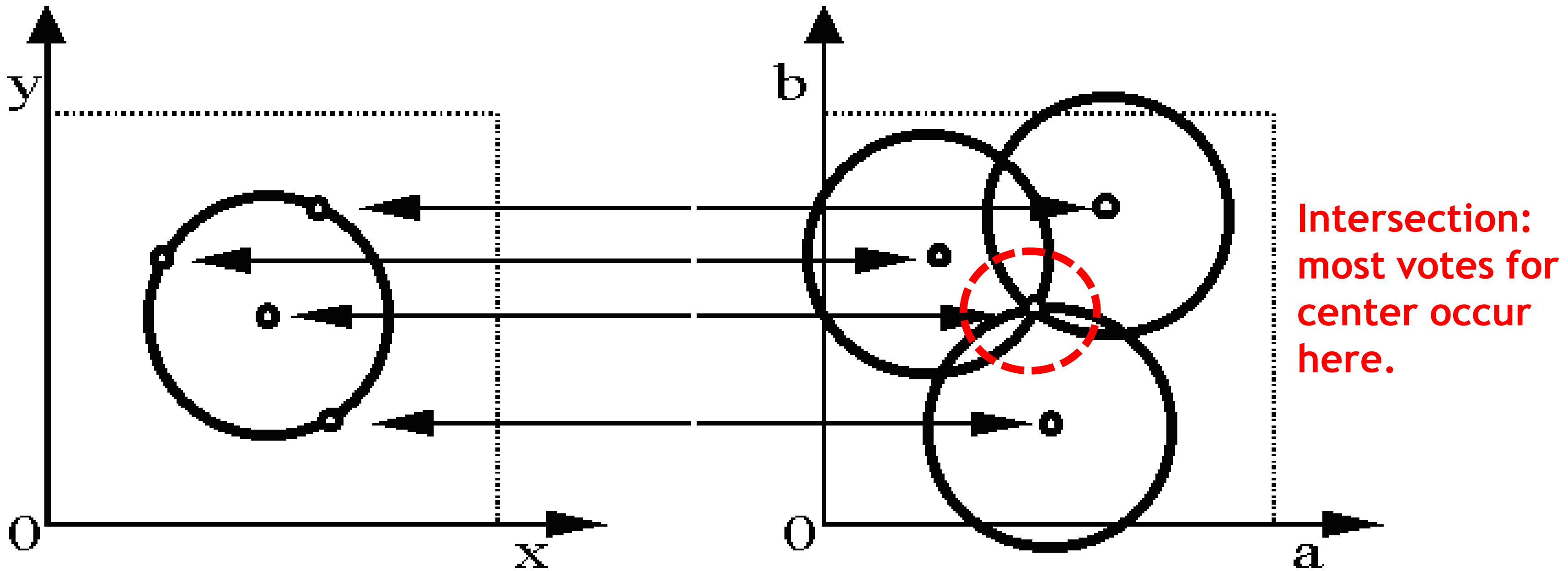


Image space **Hough space**

Hough Transform for Circles

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

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

- **For an unknown radius r, unknown gradient direction**

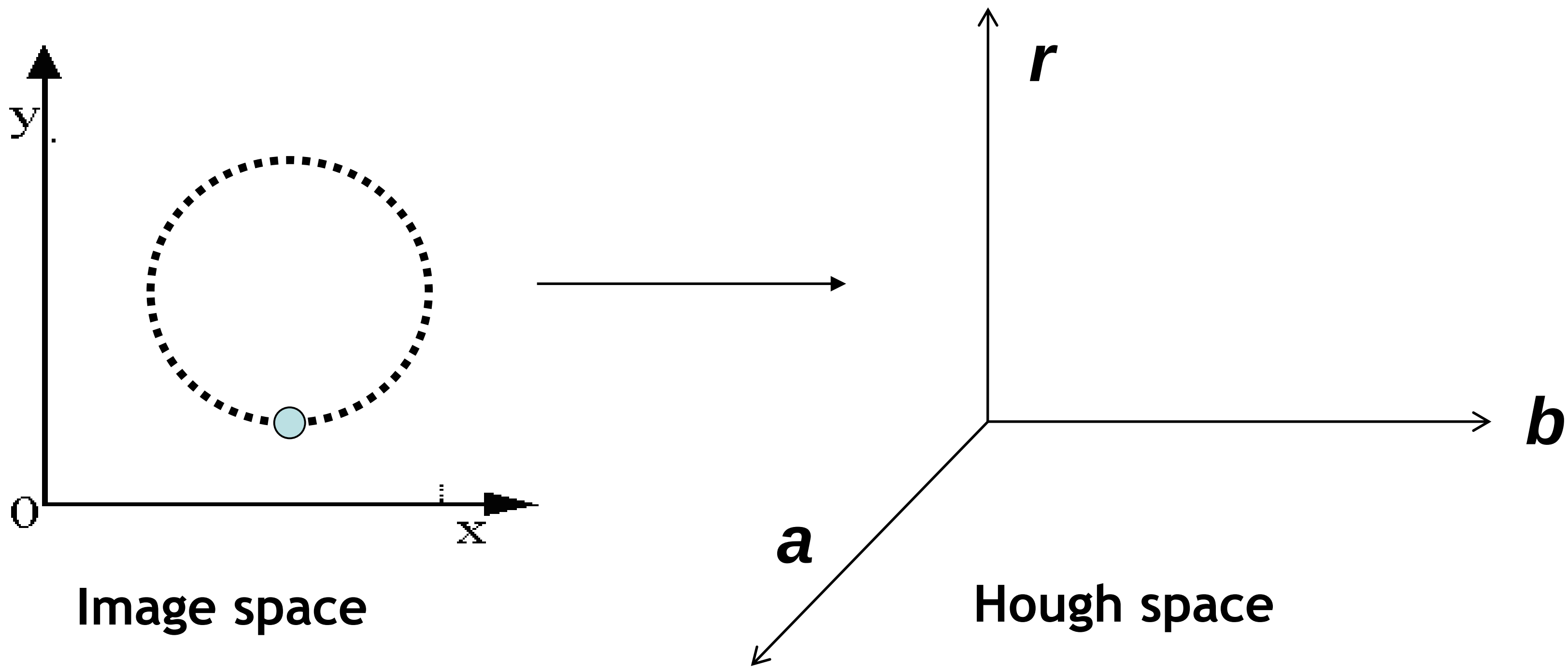


Image space **Hough space**

Hough Transform for Circles

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

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

- **For an unknown radius r, unknown gradient direction**

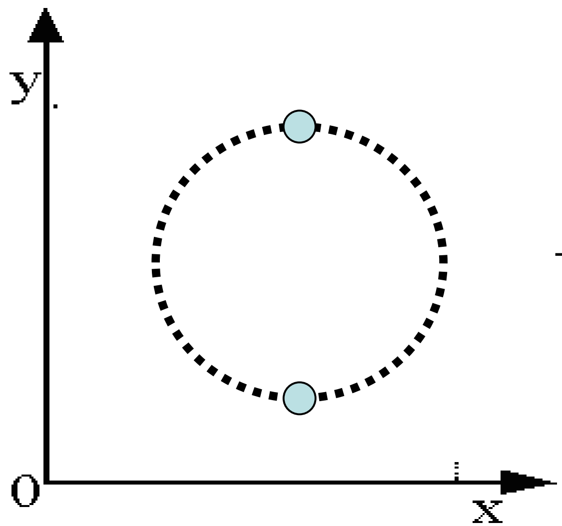
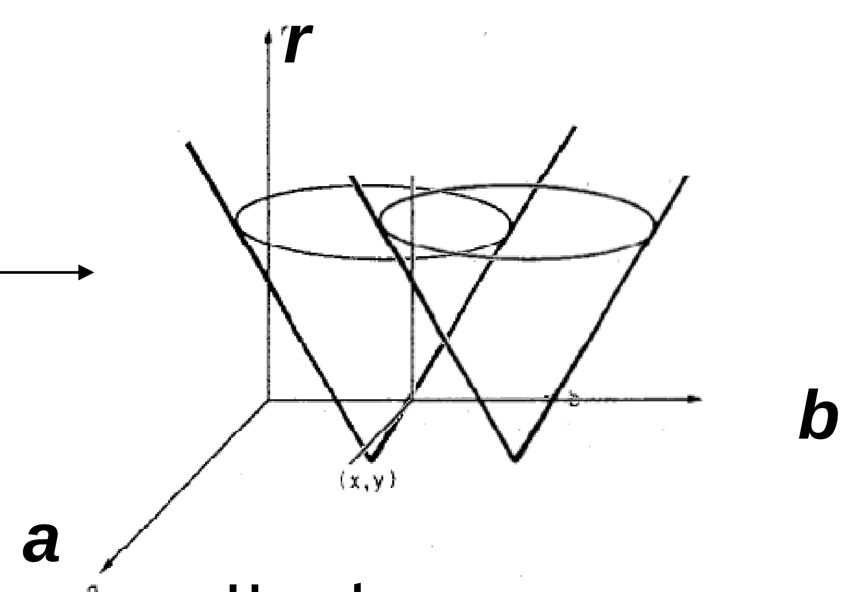


Image space



Hough space

Hough Transform for Circles

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

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

- **For an unknown radius r, *known* gradient direction**

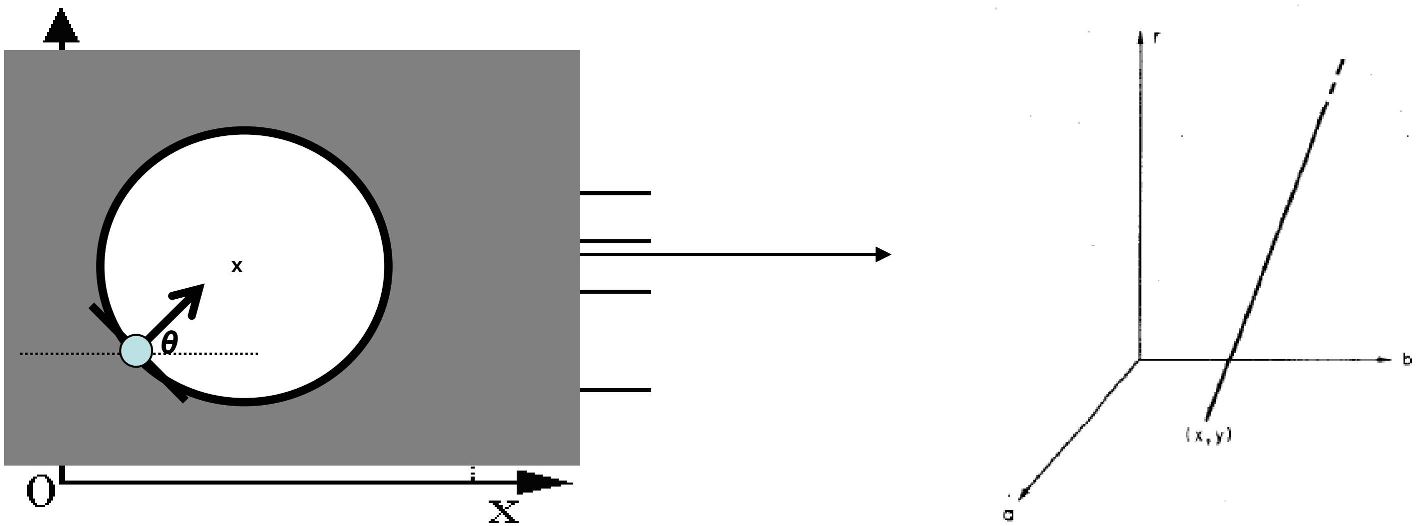


Image space **Hough space**

Hough Transform for Circles

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

For each possible radius value r**:**

For each possible gradient direction θ**:**

// or use estimated gradient

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

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

$$H[a,b,r] \mathrel{+}= 1$$

end

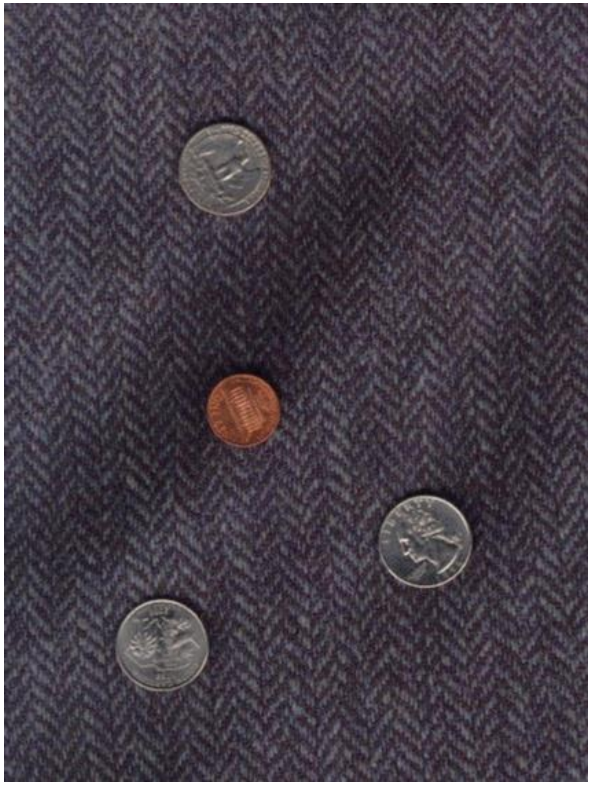
end

Example: Detecting Circles with Hough

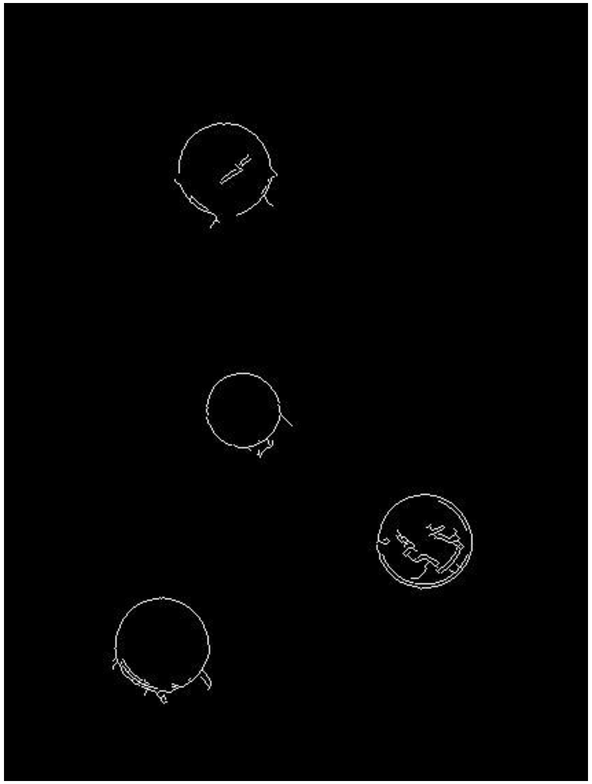
Crosshair indicates results of Hough transform, bounding box found via motion differencing.

Example: Detecting Circles with Hough

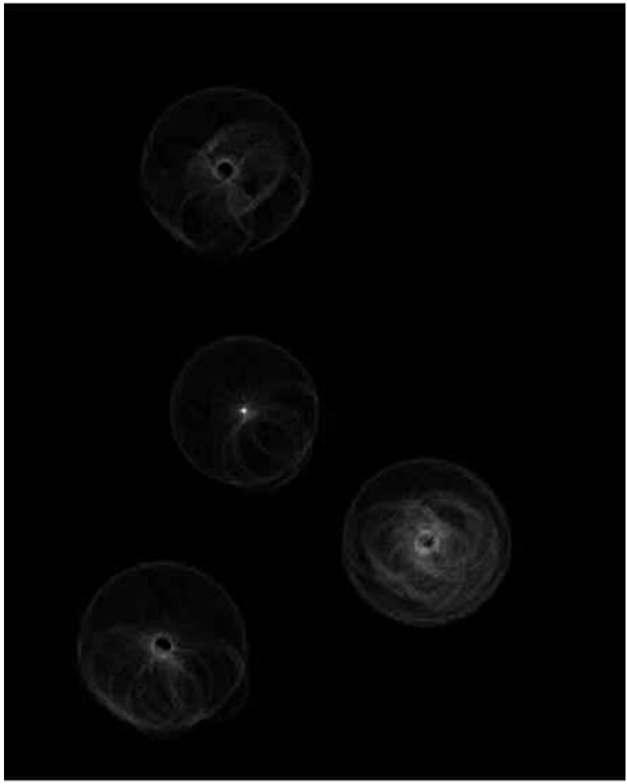
Original



Edges



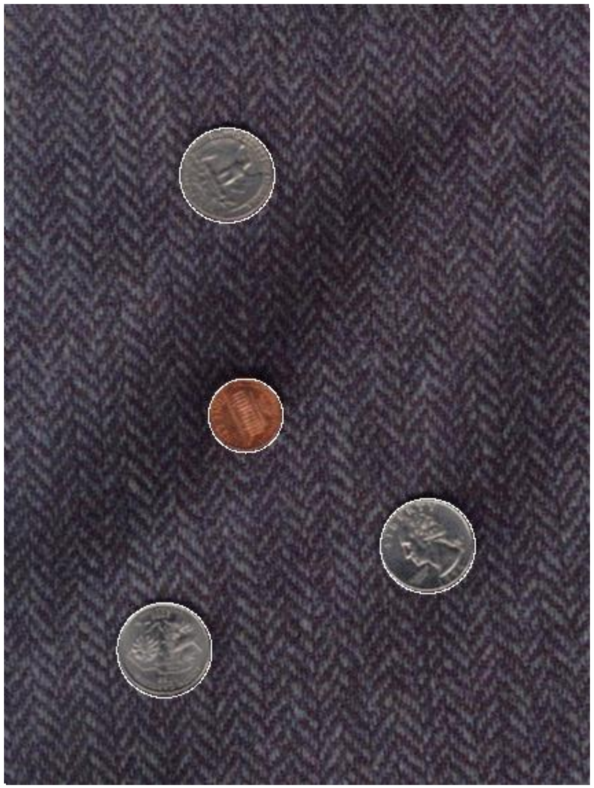
Votes: Penny



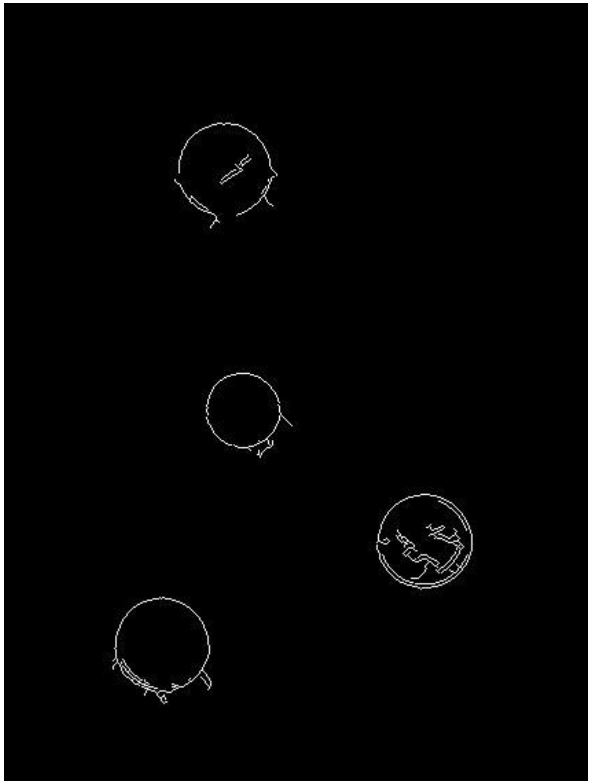
Note: a different Hough transform (with separate accumulators) was used for each circle radius (quarters vs. penny).

Example: Detecting Circles with Hough

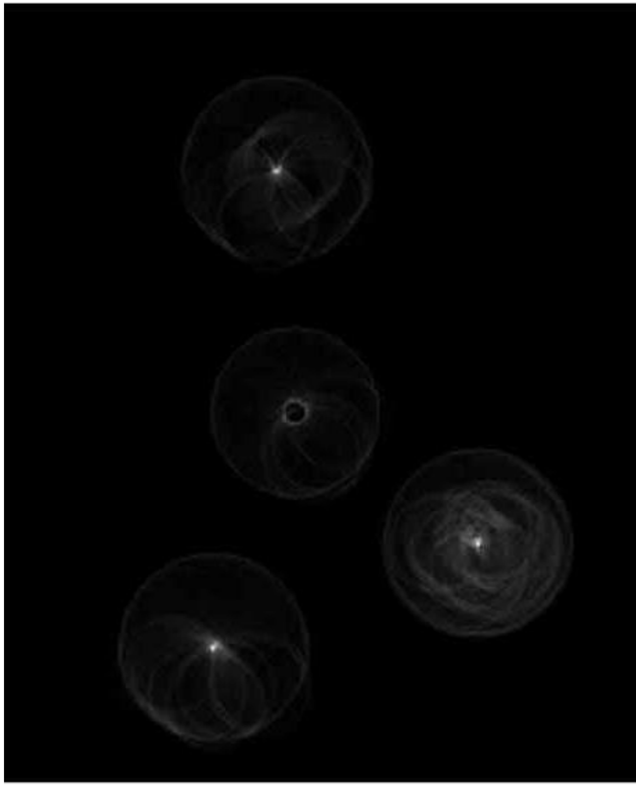
Original



Edges



Votes: Quarter



Combined detections

Voting: Practical Tips

- Minimize irrelevant tokens first (take edge points with significant gradient magnitude)
- 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 some points that are not exactly collinear cast votes for different buckets
- Vote for neighbors, also (smoothing in accumulator array)
- Utilize direction of edge to reduce free parameters by 1
- To read back which points voted for “winning” peaks, keep tags on the votes.

Hough Transform: Pros and Cons

Pros

- All points are processed independently, so can cope with occlusion
- Some robustness to noise: noise points unlikely to contribute consistently to any single bin
- Can detect multiple instances of a model in a single pass

Cons

- Complexity of search time increases exponentially with the number of model parameters
- Non-target shapes can produce spurious peaks in parameter space
- Quantization: hard to pick a good grid size

Generalized Hough Transform

- What if want to detect arbitrary shapes defined by boundary points and a reference point?

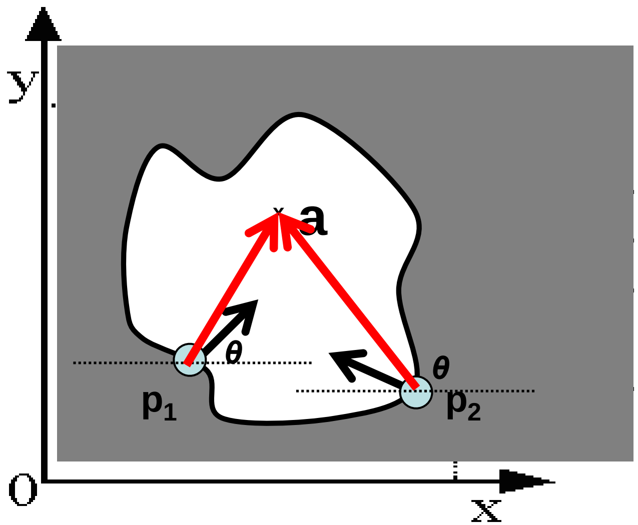


Image space

At each boundary point, compute displacement vector: $r = a - p_i$.

For a given model shape: store these vectors in a table indexed by gradient orientation θ.

[Dana H. Ballard, Generalizing the Hough Transform to Detect Arbitrary Shapes, 1980]

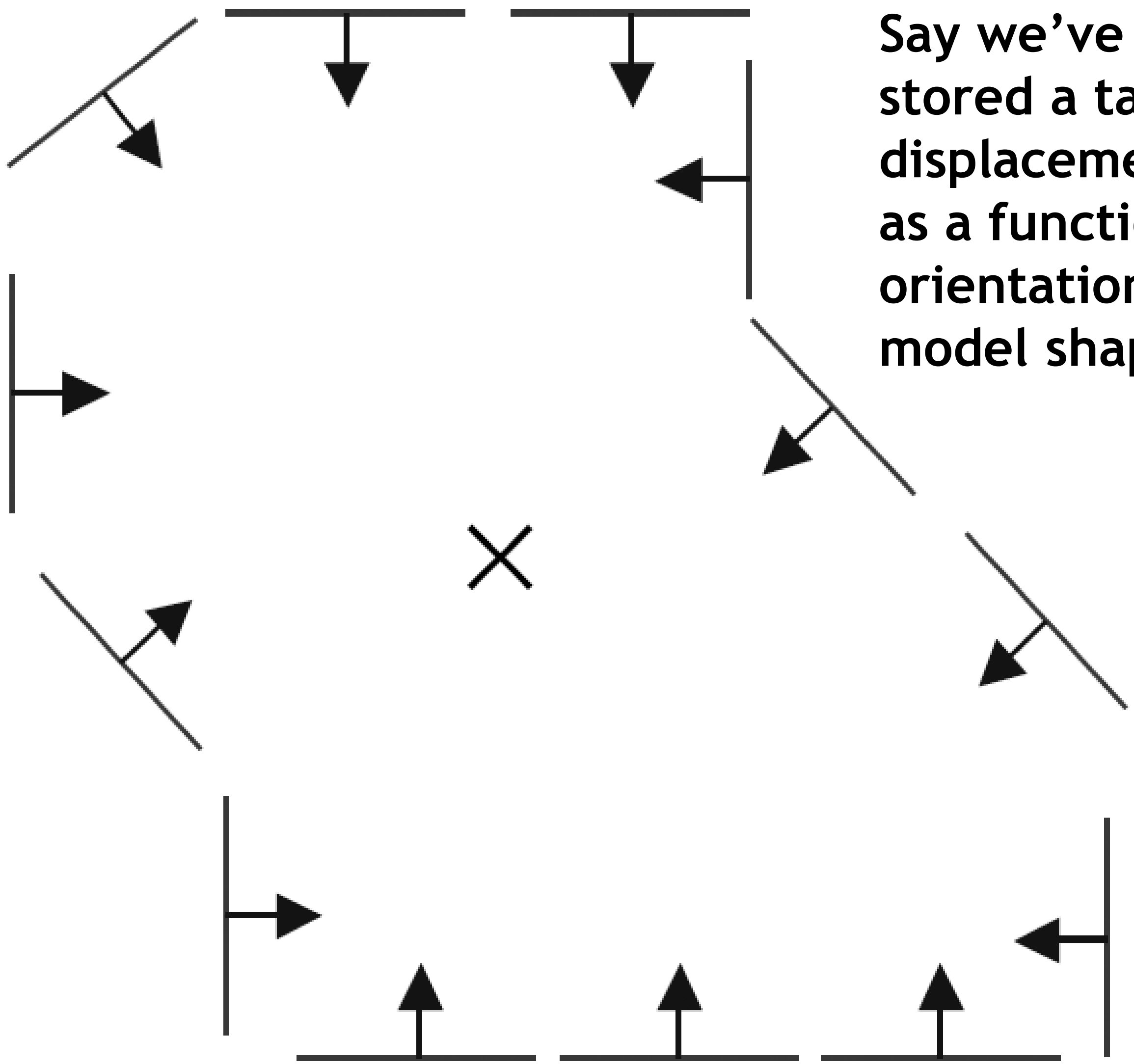
Generalized Hough Transform

To *detect* the model shape in a new image:

- **For each edge point**
  - **Index into table with its gradient orientation θ**
  - **Use retrieved r vectors to vote for position of reference point**
- **Peak in this Hough space is reference point with most supporting edges**

Assuming translation is the only transformation here, i.e., orientation and scale are fixed.

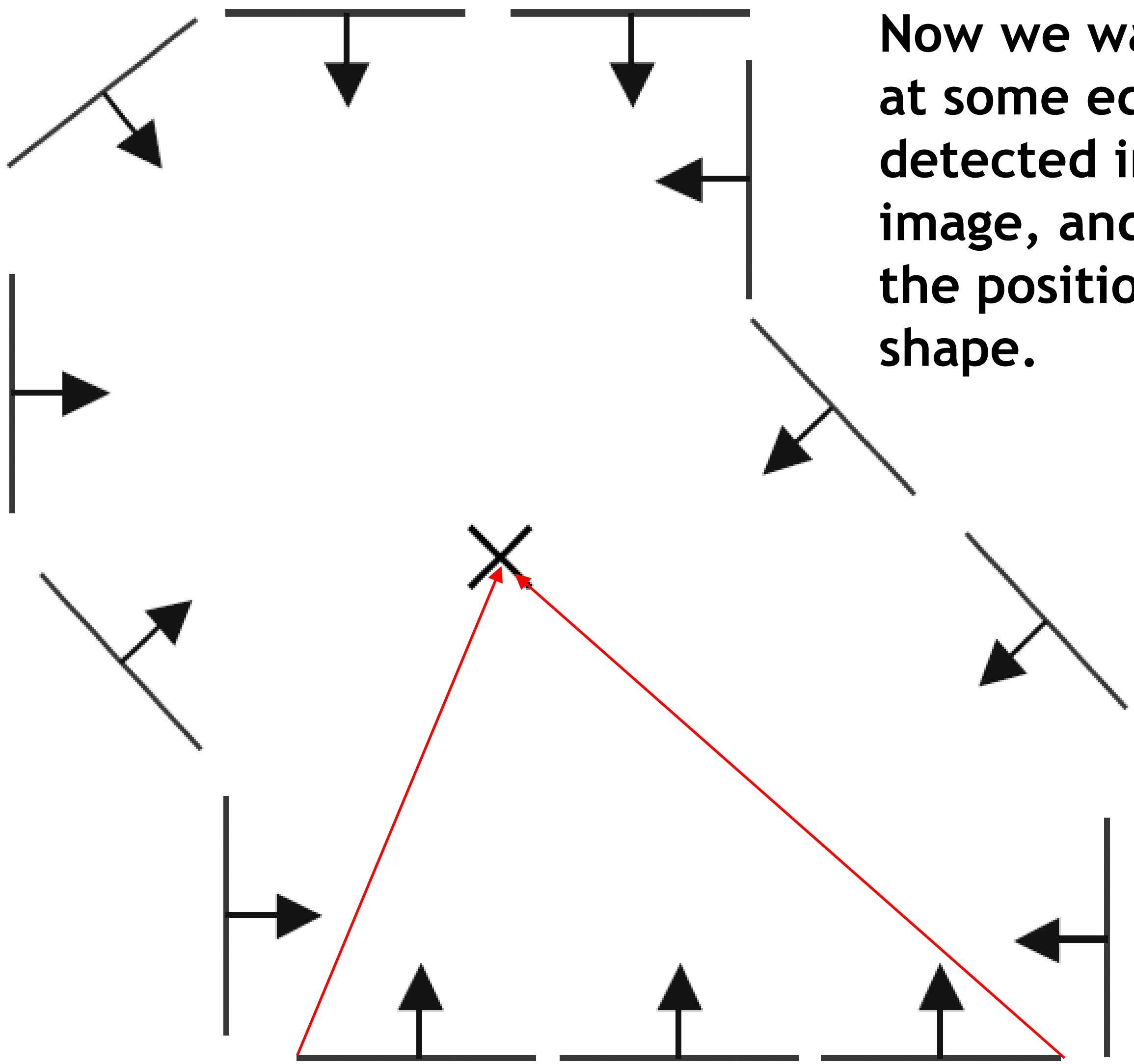
Example: Generalized Hough Transform



Model shape

Say we've already stored a table of displacement vectors as a function of edge orientation for this model shape.

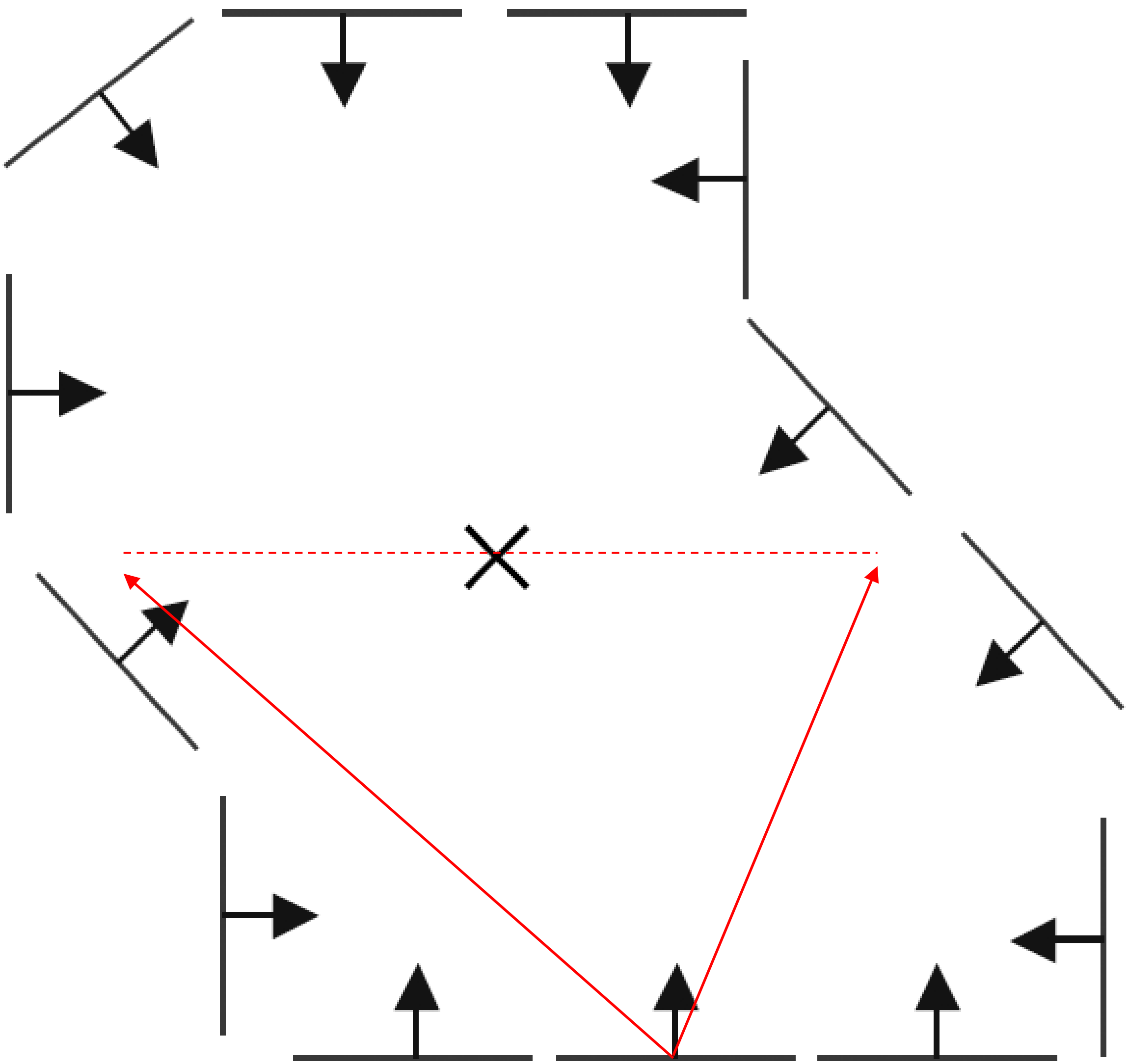
Example: Generalized Hough Transform



Now we want to look at some edge points detected in a *new* image, and vote on the position of that shape.

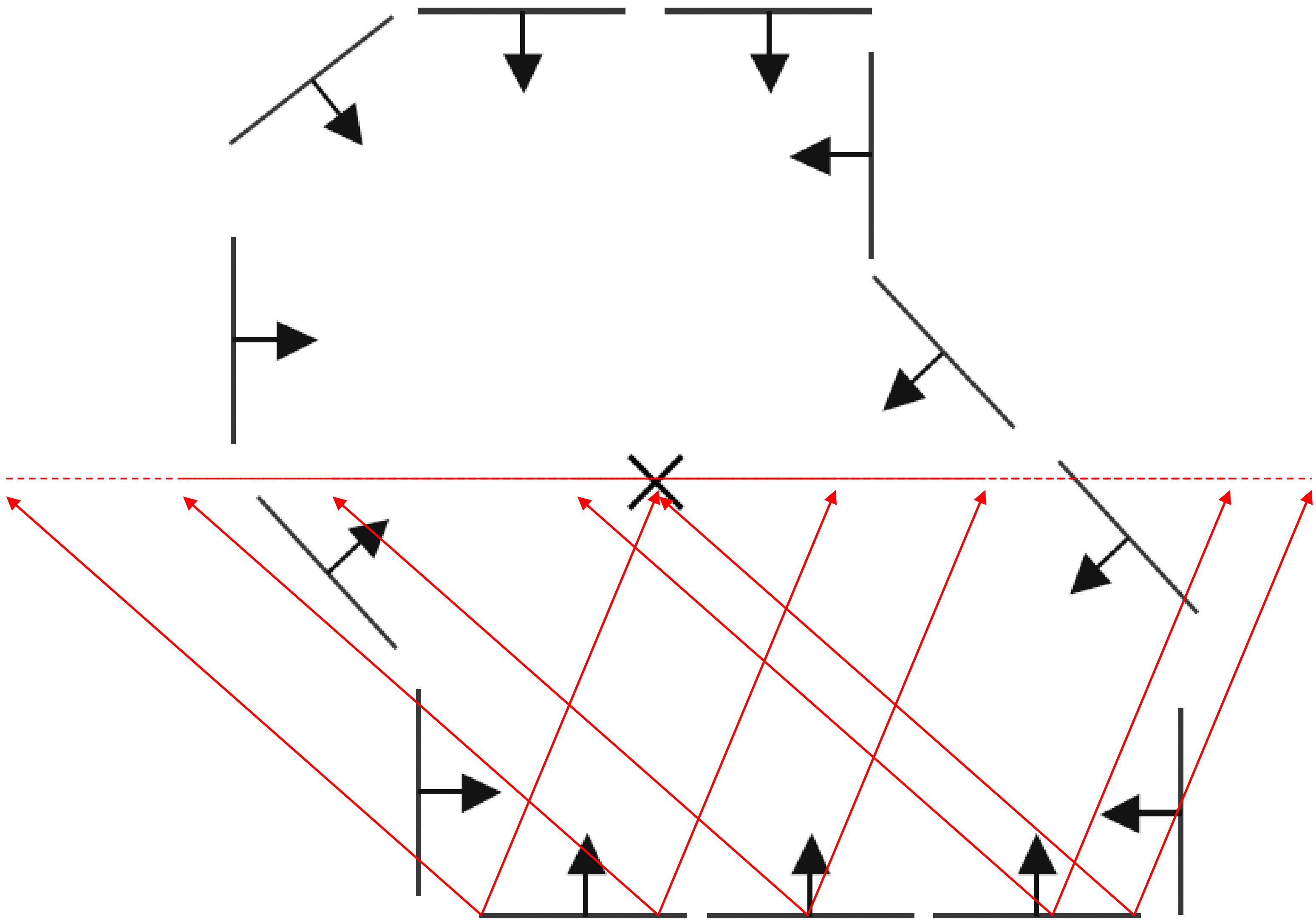
Displacement vectors for model points

Example: Generalized Hough Transform



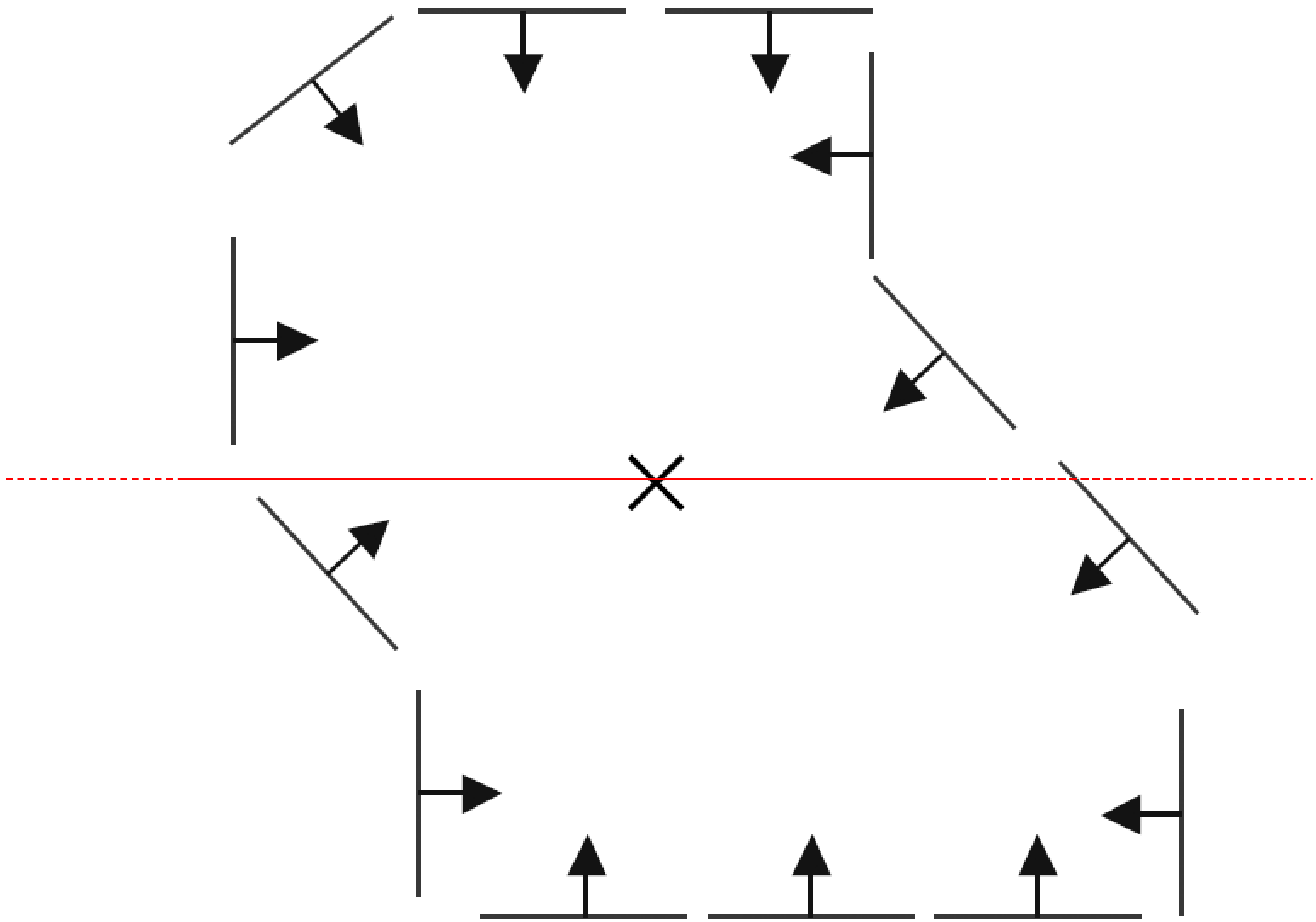
Range of voting locations for test point

Example: Generalized Hough Transform



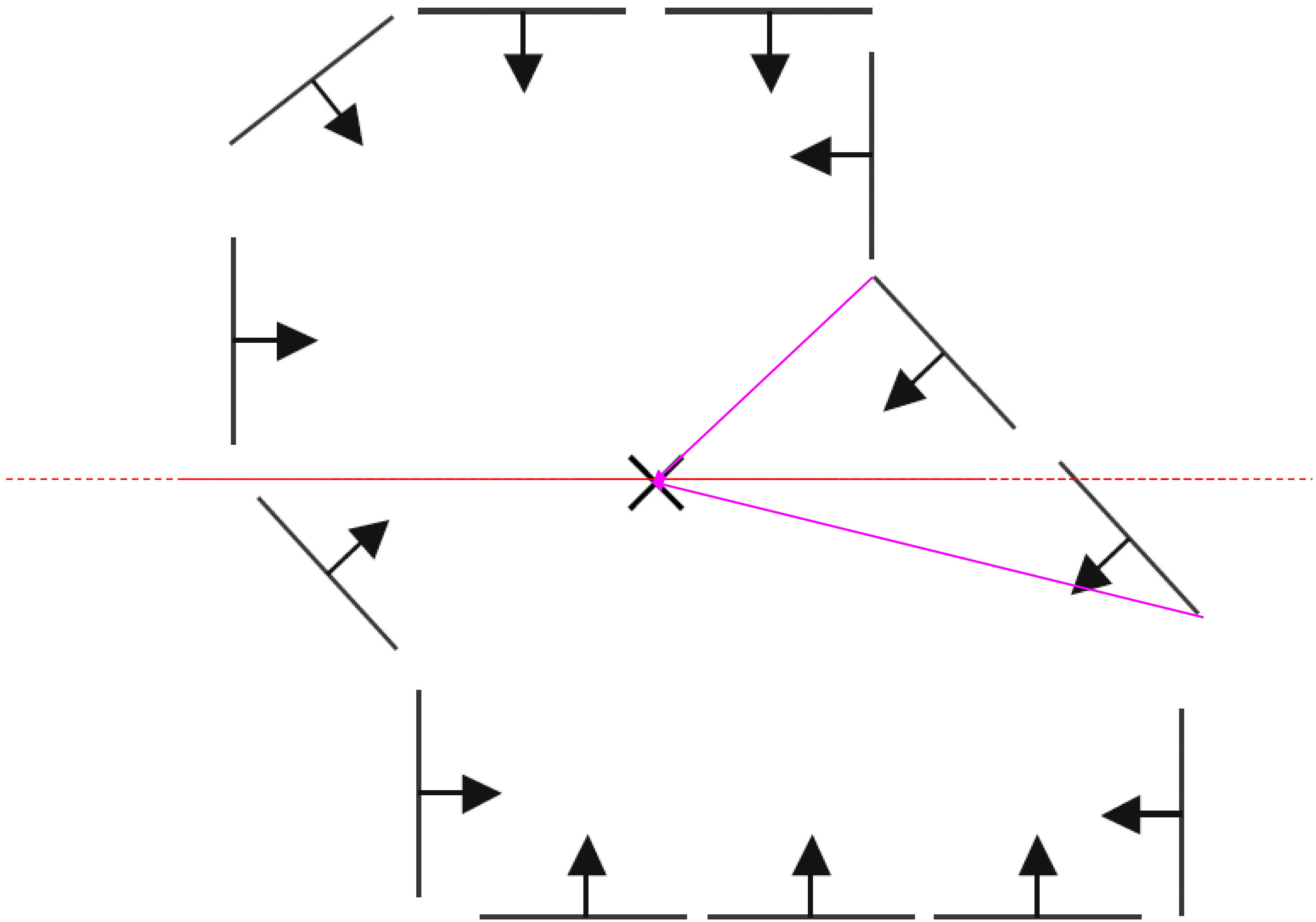
Range of voting locations for test point

Example: Generalized Hough Transform



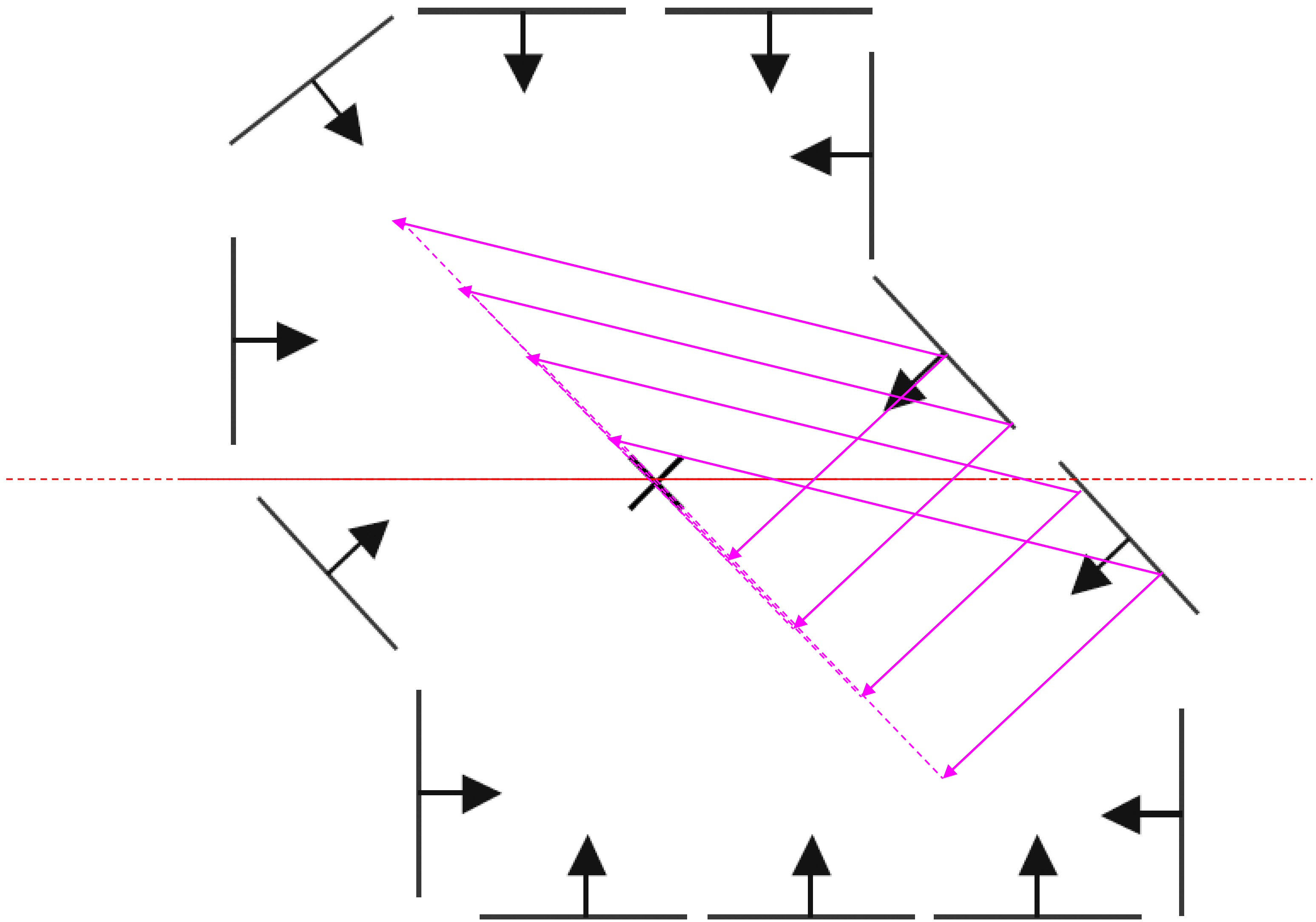
Votes for points with $\theta = \uparrow$

Example: Generalized Hough Transform



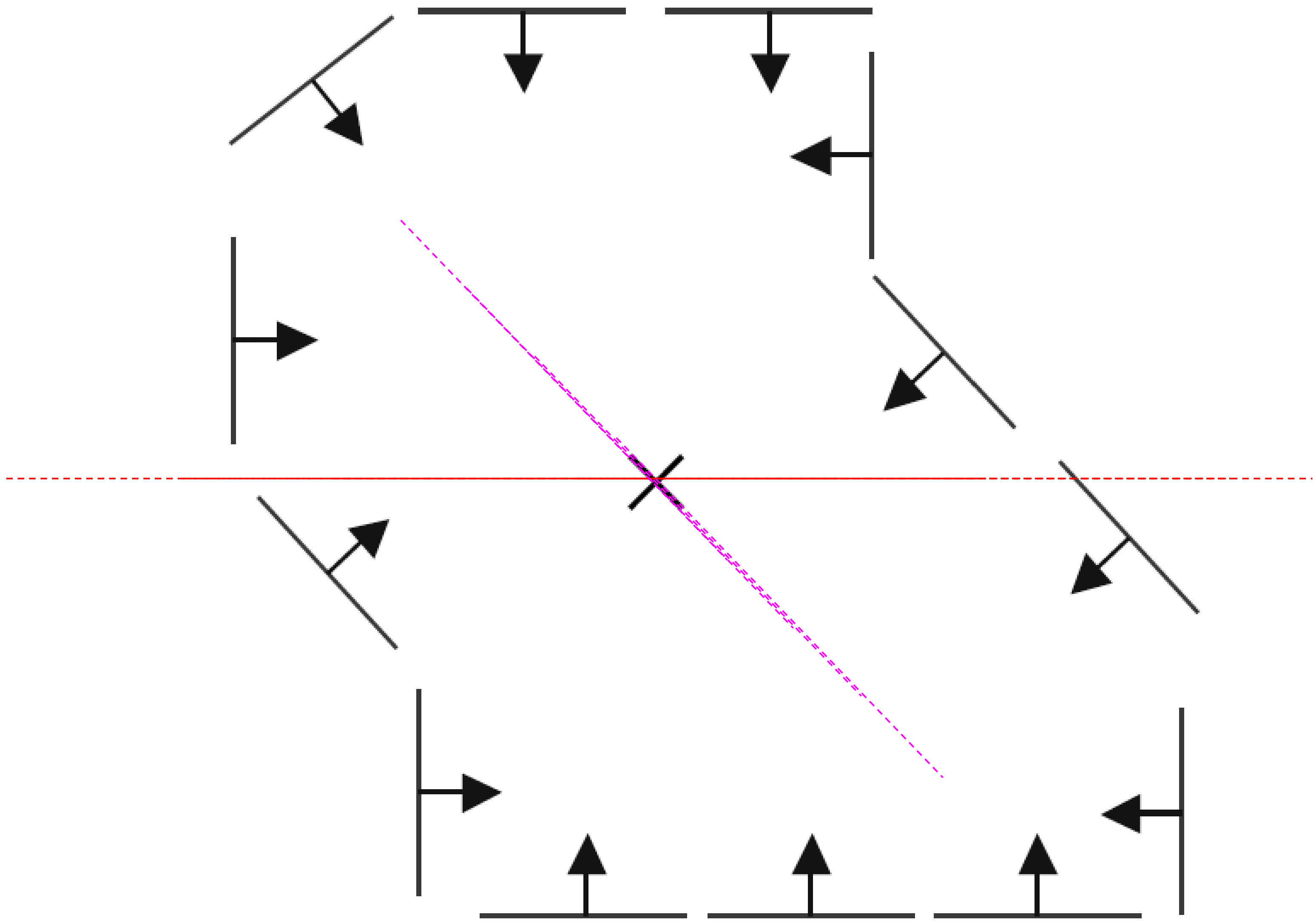
Displacement vectors for model points

Example: Generalized Hough Transform



Range of voting locations for test point

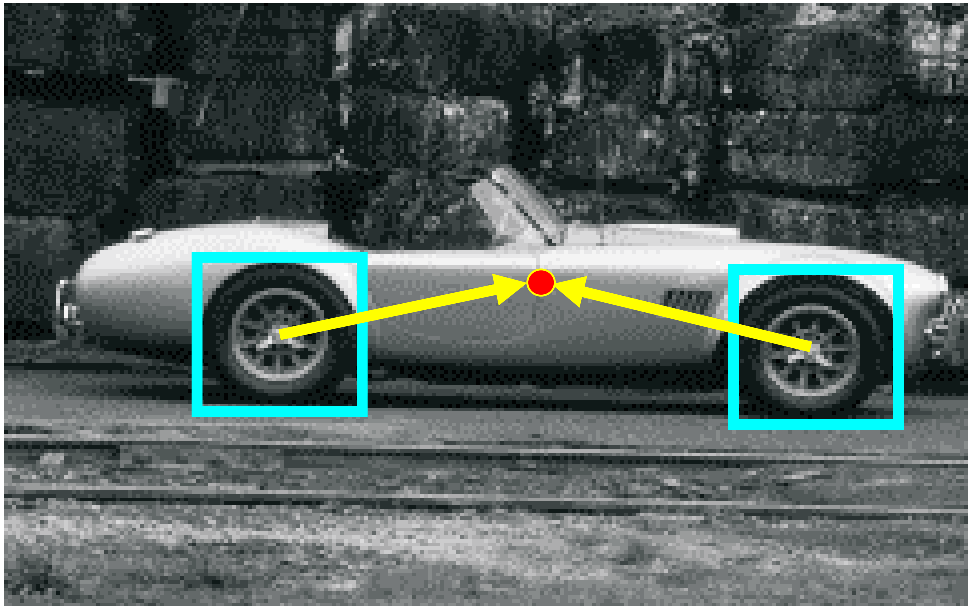
Example: Generalized Hough Transform



Votes for points with θ = ↙

Application in Recognition

- Instead of indexing displacements by gradient orientation, index by “visual codeword”.



Training image

Visual codeword with displacement vectors

Application in Recognition

- Instead of indexing displacements by gradient orientation, index by "visual codeword".

Test image

- We'll hear more about this method in lecture 14…