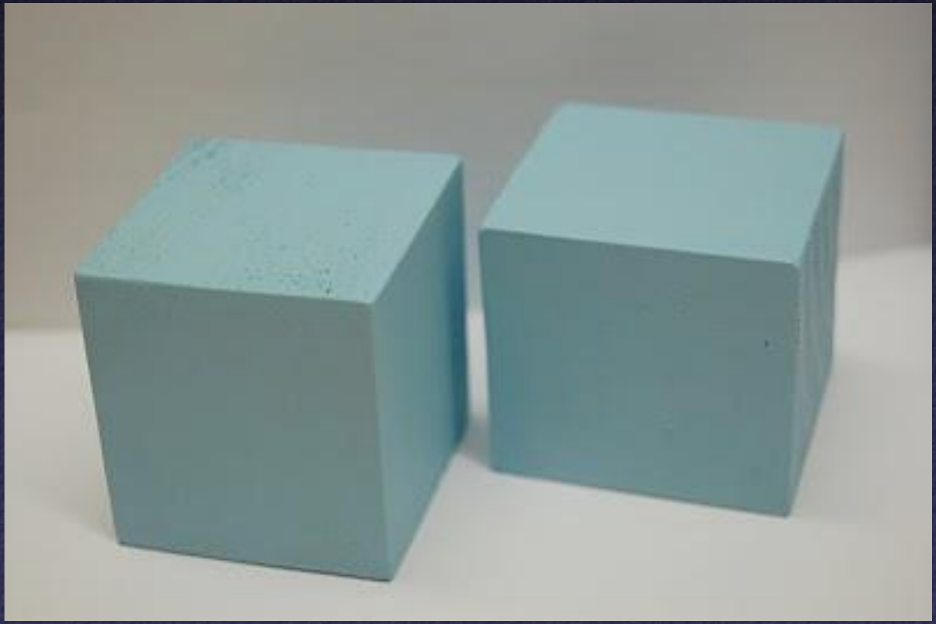
Feature Detection

Goal

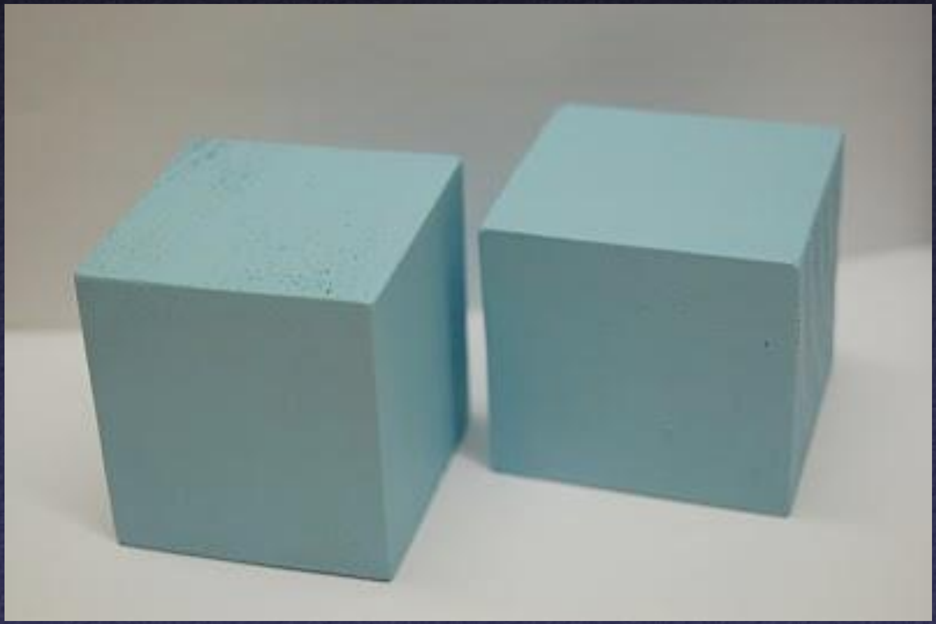
Extract "structural features" from an image

- Non-accidental properties



Goal

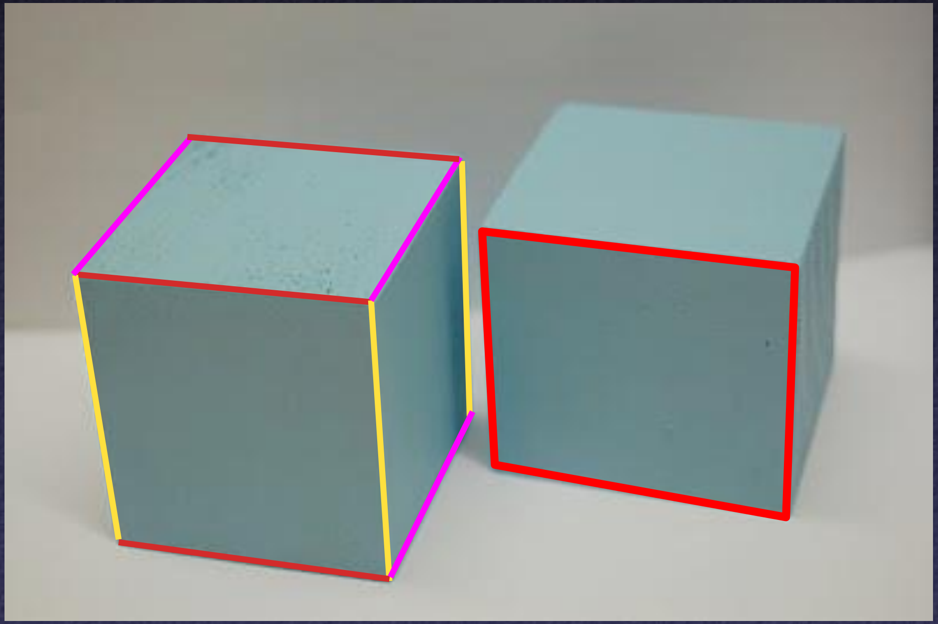
What types of “structure” are in this image?



Goal

What types of "structure" are in this image?

- Straight lines
- Parallel lines
- Symmetric pairs of lines
- Trapezoids
- Monochromatic regions
- etc.



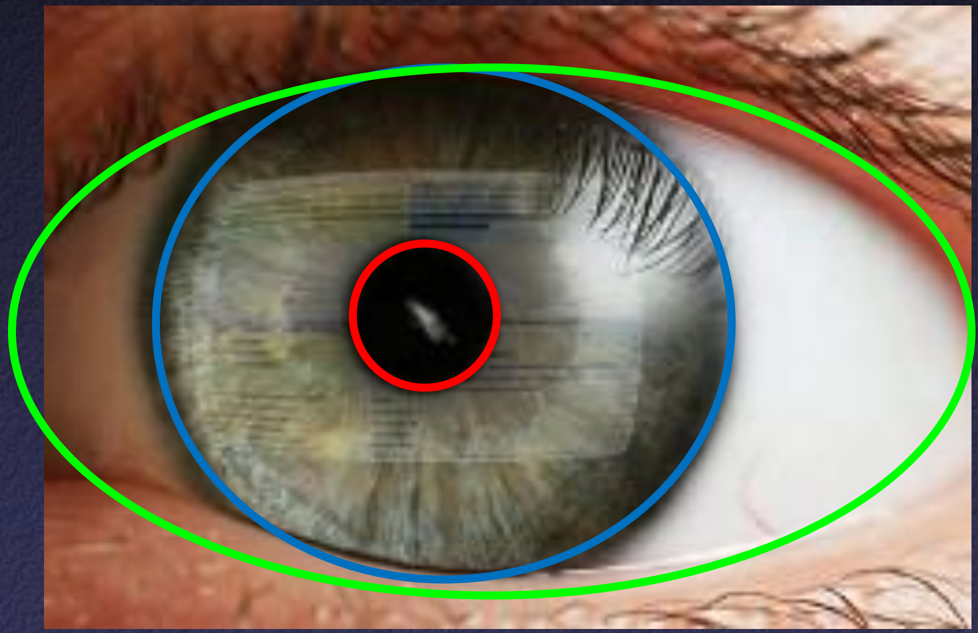
Goal

What types of "structure" are in this image?

Goal

What types of "structure" are in this image?

- Circles
- Ellipses
- Symmetries in color and texture
- etc.



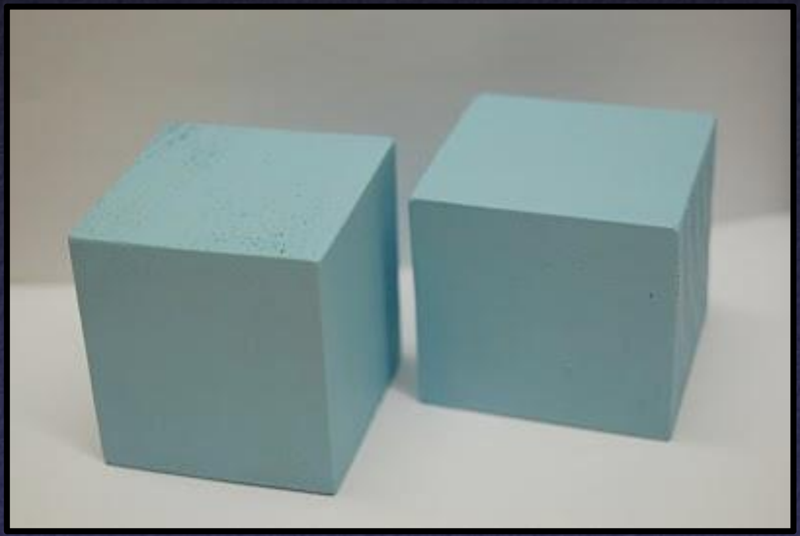
This Lecture

Algorithms for “structure detection”

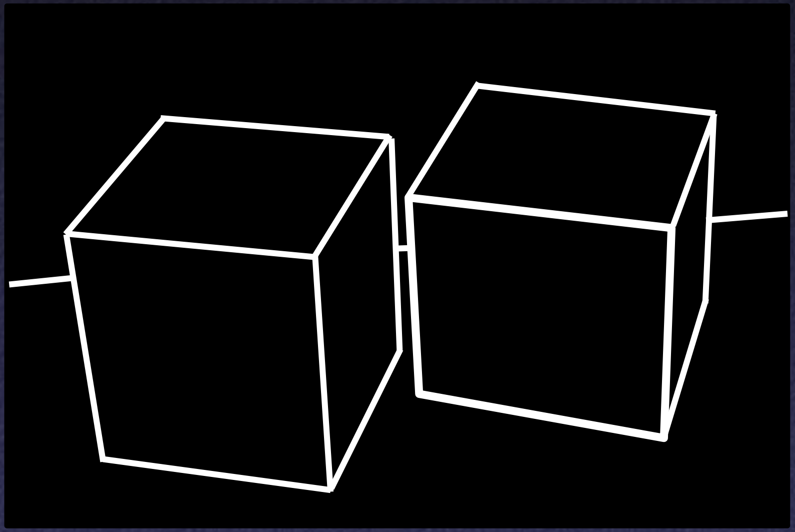
- Line detection
- Circle detection
- Symmetry detection

Line Detection

Let's first consider how to detect lines



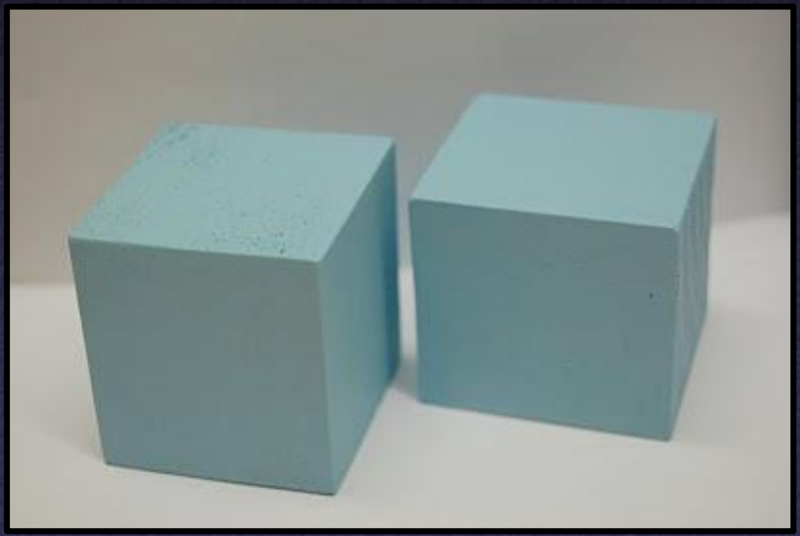
Input



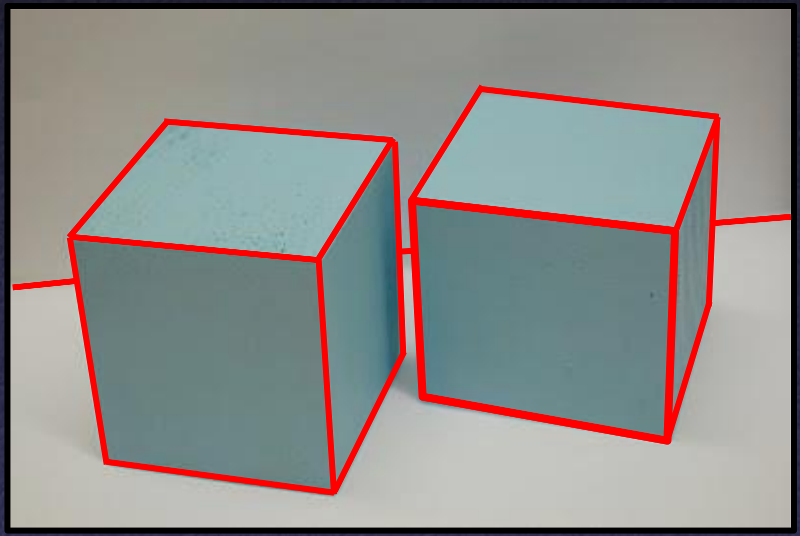
Output

Line Detection

Let's first consider how to detect lines



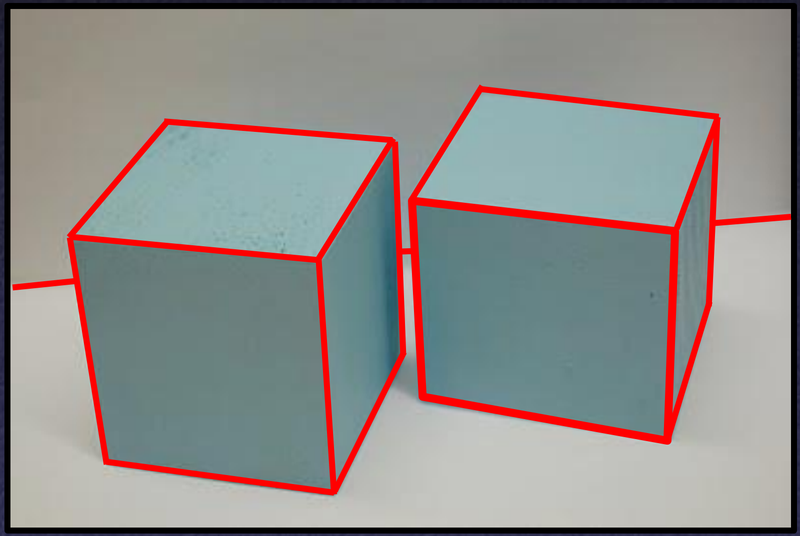
Input



Overlay

Line Detection

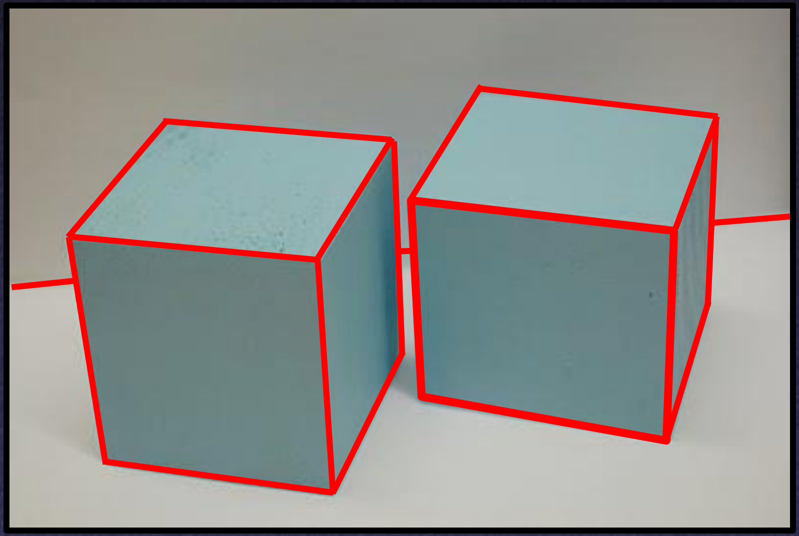
Desirable properties of a line detection algorithm?



Line Detection

Desirable properties of a line detection algorithm:

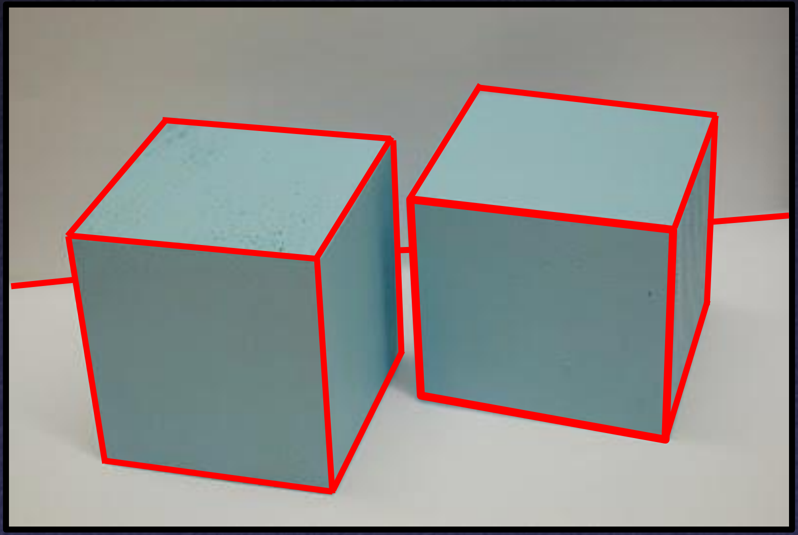
- Straight, long lines only
- Few missed or extra lines
- Provides confidence of prediction for each pixel
- Robust to differences in occlusion, noise, scale, rotation, translation, slight non-straightness, brightness, etc.
- Efficient computation



Line Detection

Not the same as edge detection:

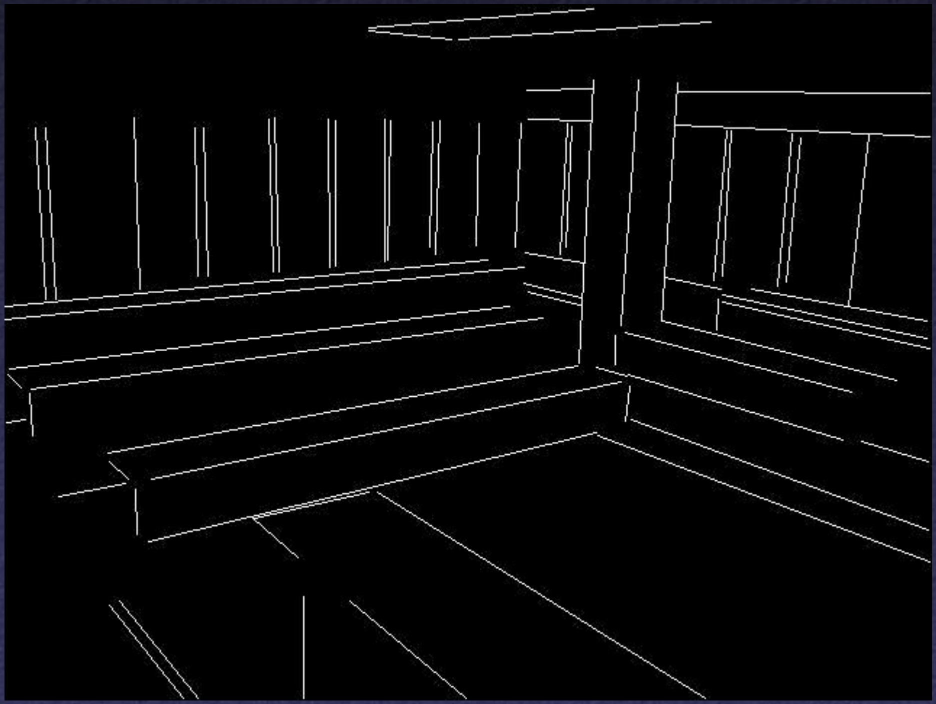
- Edges are small-scale, local properties
- Lines are large-scale, structural properties



Line Detection

Applications:

- Removing radial distortion
- Camera pose estimation
- Segmentation
- Scene classification
- Object detection
- etc.



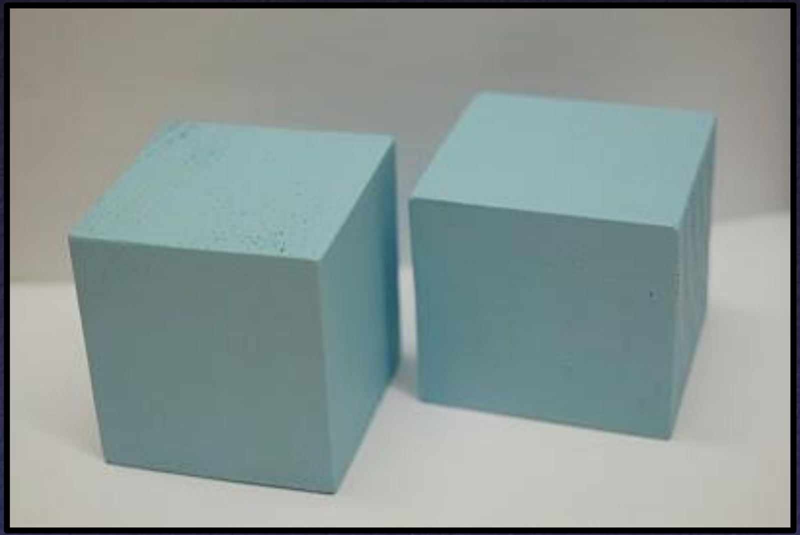
Line Detection

Applications:

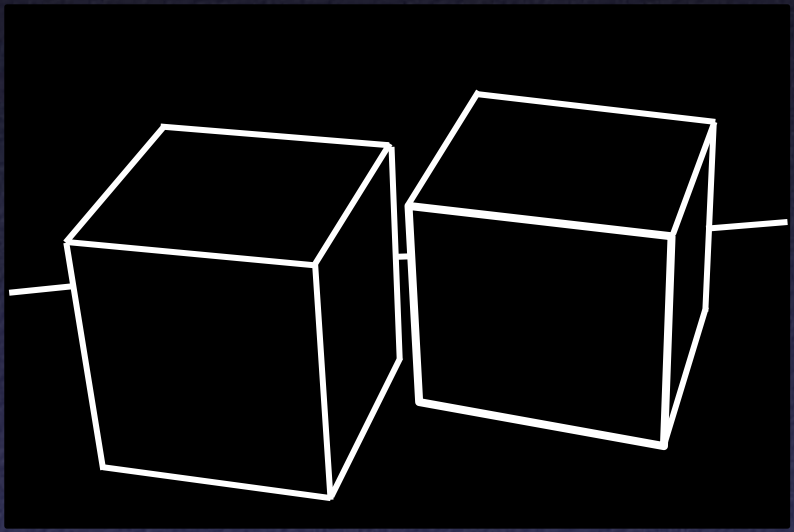
- Removing radial distortion
- Camera pose estimation
- Segmentation
- Scene classification
- Object detection
- etc.

Line Detection

Please propose a line detection algorithm



Input

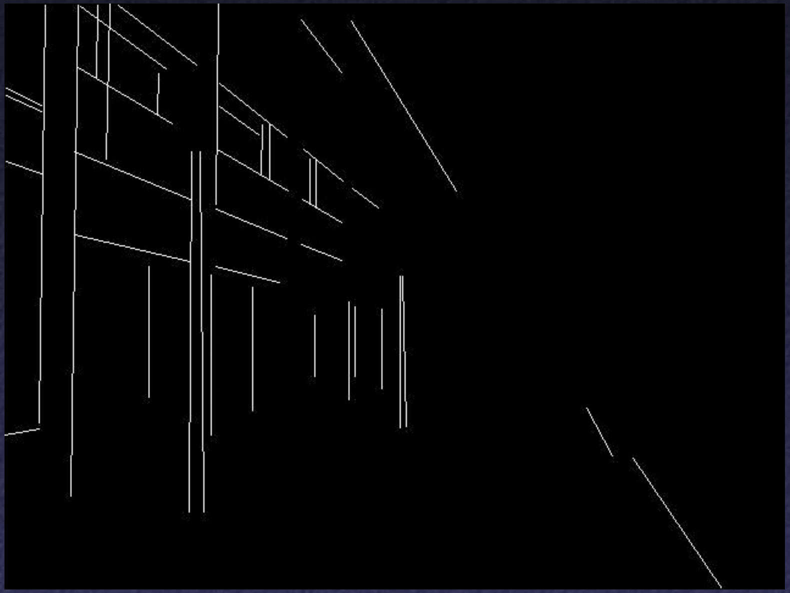


Output

Line Detection

OK, but what about this harder example?

Input

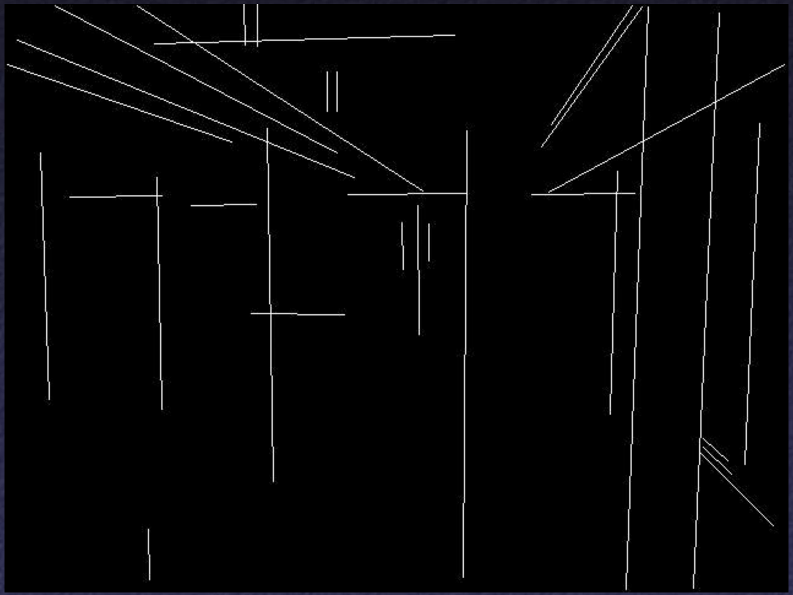


Output

Line Detection

This one?

Input



Output

Line Detection

Two common algorithms:

- RANSAC
- Hough transform

Line Detection

Two common algorithms:

- RANSAC
- Hough transform

RANSAC in General

RANdom SAmple Consensus

Take many random samples of data

- Compute fit for each sample
- See how many points agree
- Remember the best

RANSAC for Line Detection

At beginning:

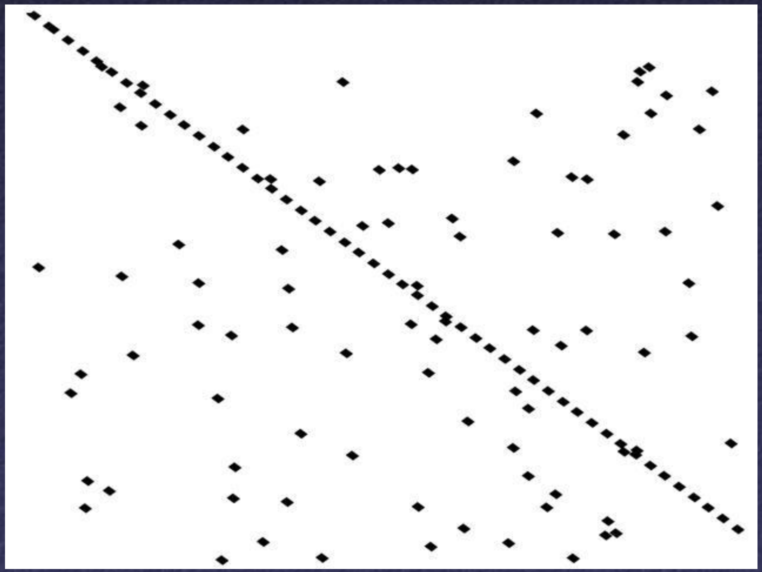
- Compute gradient direction N and magnitude G

Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels "support" L

At end:

- Report the line L* with the most "support"



Input

RANSAC for Line Detection

At beginning:

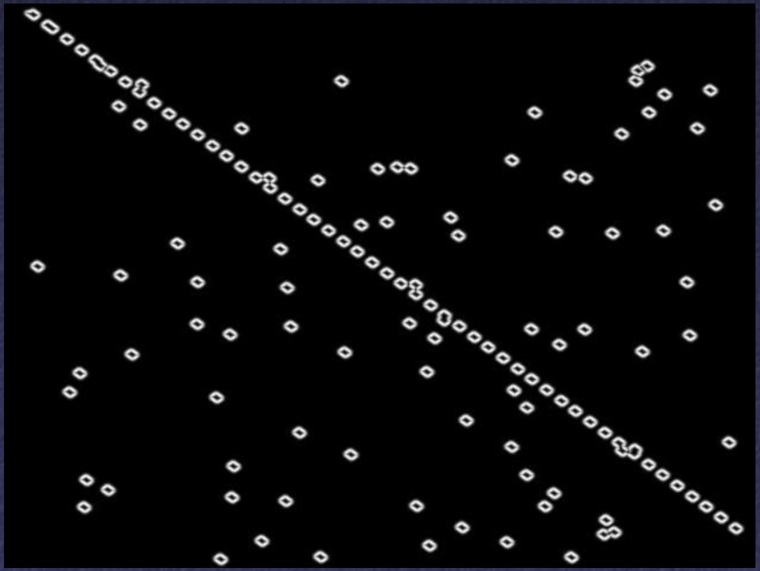
- Compute gradient direction N and magnitude G

Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels "support" L

At end:

- Report the line L* with the most "support"



Gradient Magnitude (G)

RANSAC for Line Detection

At beginning:

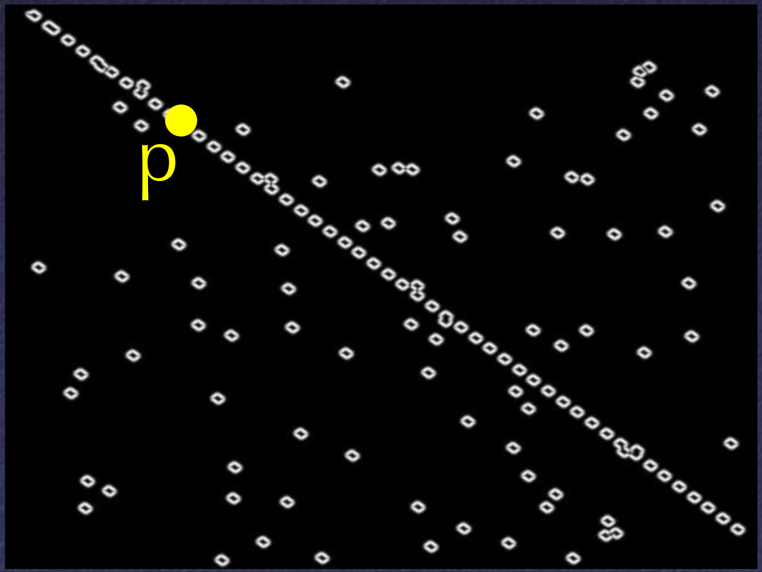
- Compute gradient direction N and magnitude G

Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels “support” L

At end:

- Report the line L* with the most “support”



Point p and Normal N(p)

RANSAC for Line Detection

At beginning:

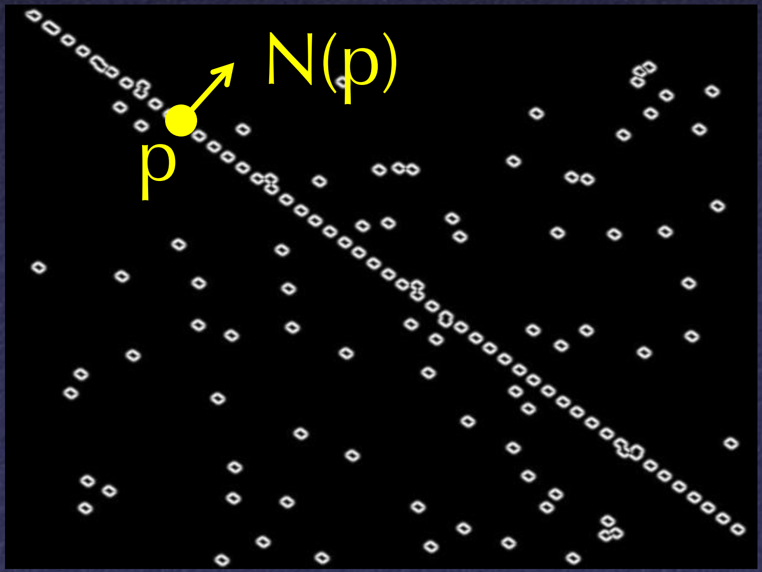
- Compute gradient direction N and magnitude G

Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels "support" L

At end:

- Report the line L* with the most "support"



Line L through p

RANSAC for Line Detection

At beginning:

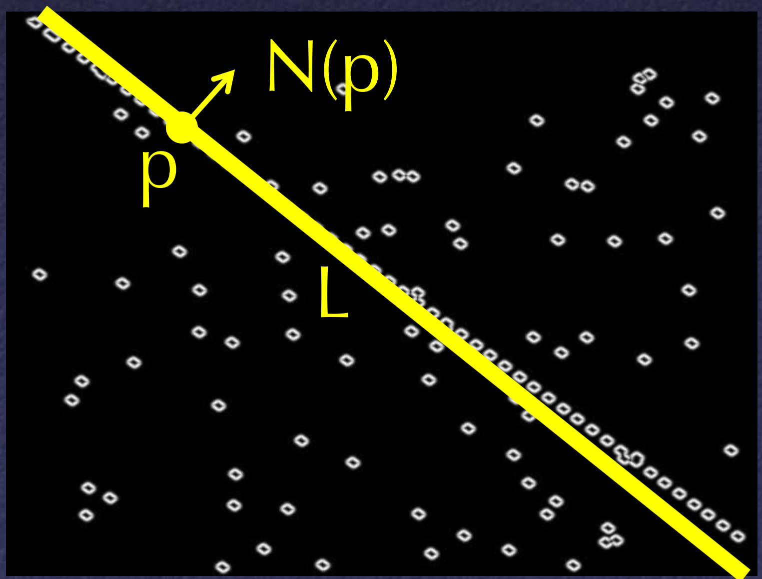
- Compute gradient direction N and magnitude G

Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels "support" L

At end:

- Report the line L* with the most "support"



Line L through p

RANSAC for Line Detection

At beginning:

- Compute gradient direction N and magnitude G

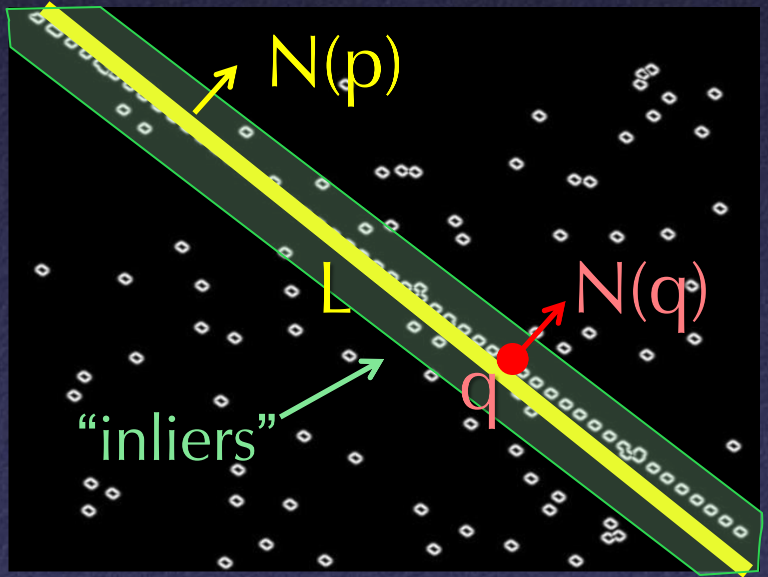
Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels "support" L

$$Support(L) = \sum_{q \in L} G(q)\ |N(p) \cdot N(q)|$$

At end:

- Report the line L* with the most "support"



Compute support

RANSAC for Line Detection

At beginning:

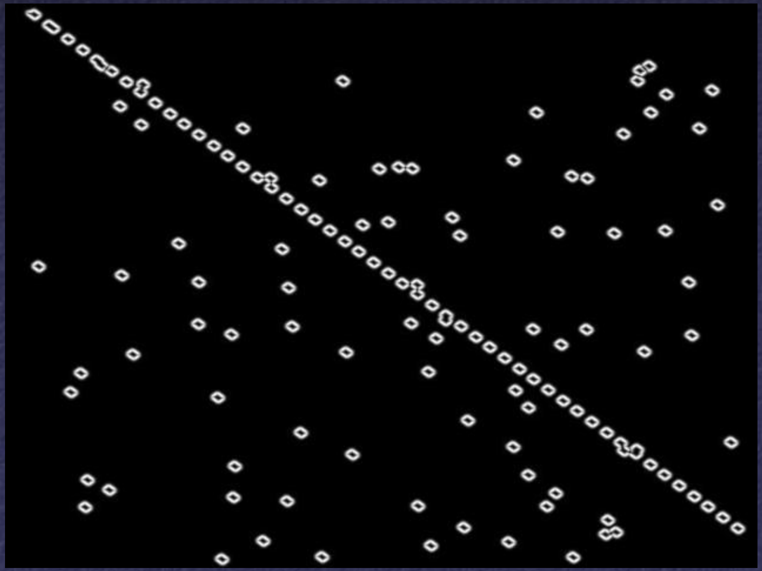
- Compute gradient direction N and magnitude G

Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels "support" L

At end:

- Report the line L* with the most "support"



RANSAC for Line Detection

At beginning:

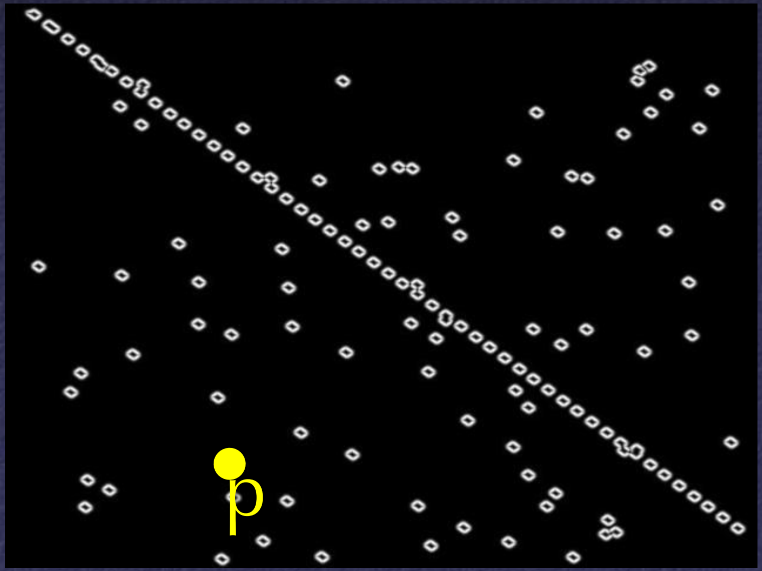
- Compute gradient direction N and magnitude G

Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels "support" L

At end:

- Report the line L* with the most "support"



Point p and Normal N(p)

RANSAC for Line Detection

At beginning:

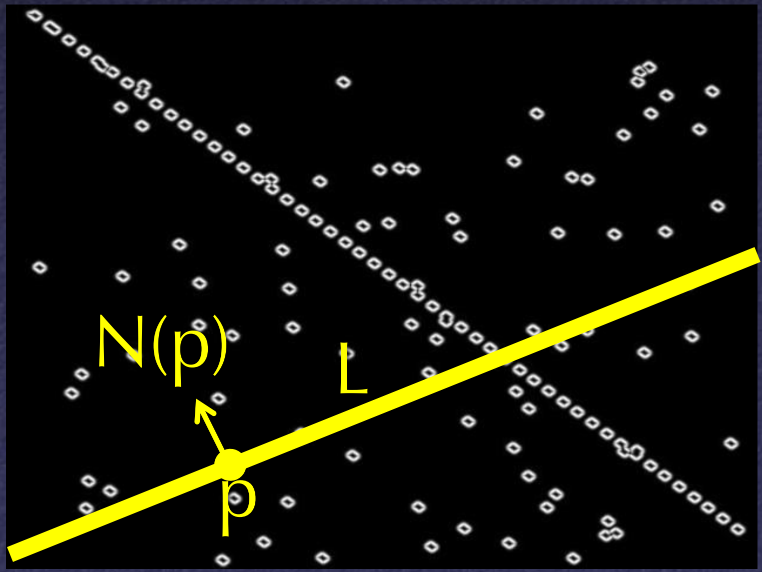
- Compute gradient direction N and magnitude G

Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels "support" L

At end:

- Report the line L* with the most "support"



Line L through p

RANSAC for Line Detection

At beginning:

- Compute gradient direction N and magnitude G

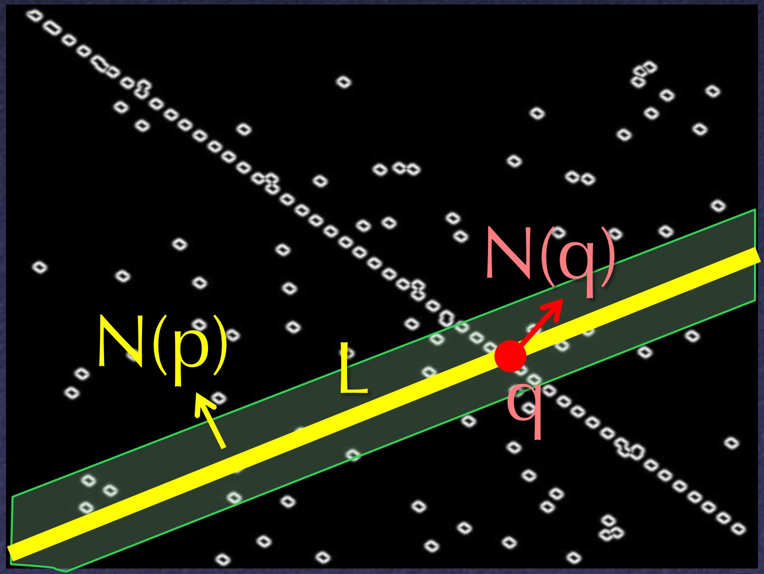
Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels "support" L

$$Support(L) = \sum_{q \in L} G(q)\ |N(p) \cdot N(q)|$$

At end:

- Report the line L* with the most "support"



Compute support

RANSAC for Line Detection

At beginning:

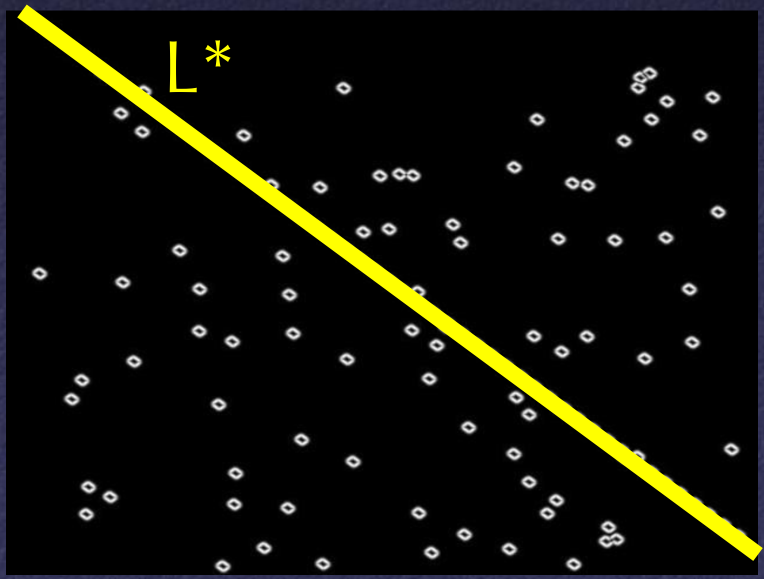
- Compute gradient direction N and magnitude G

Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels “support” L

At end:

- Report the line L* with the most “support”



Line L* with most support

RANSAC for Line Detection

Many possible variants:

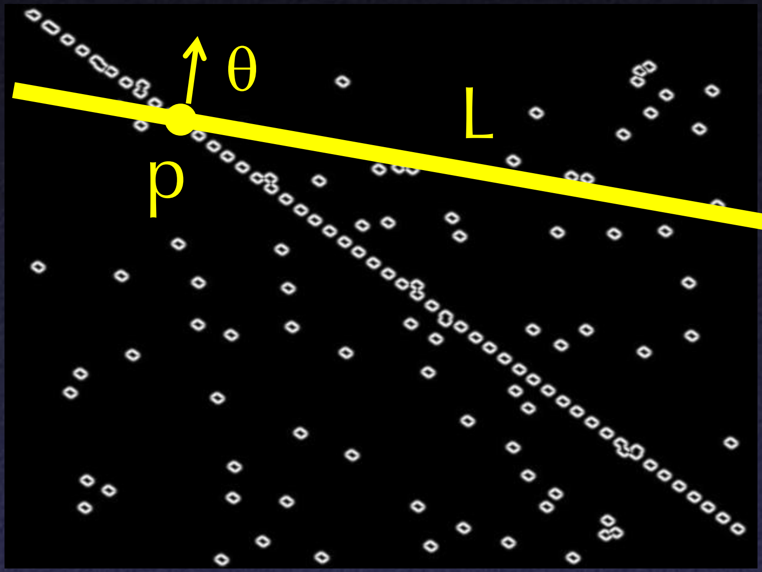
- How to choose L?
  - Point (and local gradient)
  - Point and angle
  - Two points
  - Three points
  - etc.

- How compute "support" for L?
  - $\sum G(q) \, |N(p) \bullet N(q)|$
  - Optimize L to fit "inliers"
  - etc.

RANSAC for Line Detection

Many possible variants:

- How to choose L?
  - Point (and local gradient)
  - Point and angle
  - Two points
  - Three points
  - etc.
- How compute “support” for L?
  - $\sum G(q)\ |N(p) \bullet N(q)|$
  - Optimize L to fit “inliers”
  - etc.

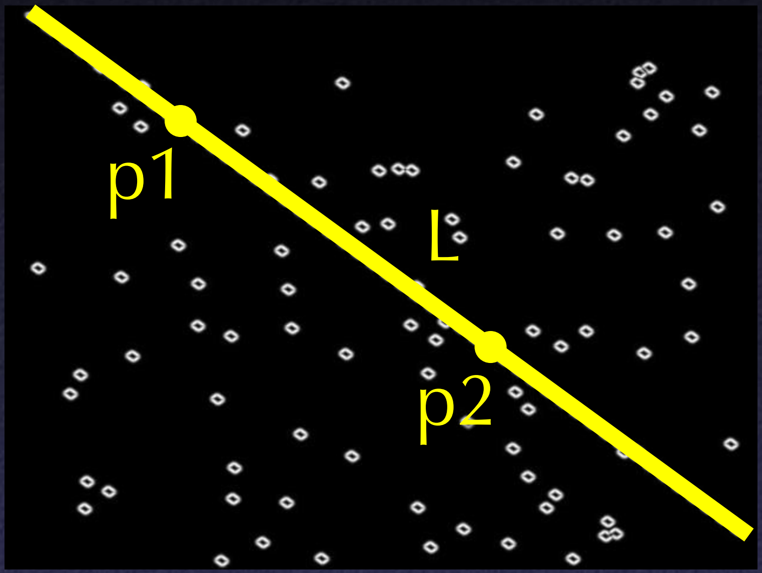


Line through point with angle

RANSAC for Line Detection

Many possible variants:

- How to choose L?
  - Point (and local gradient)
  - Point and angle
  - Two points
  - Three points
  - etc.



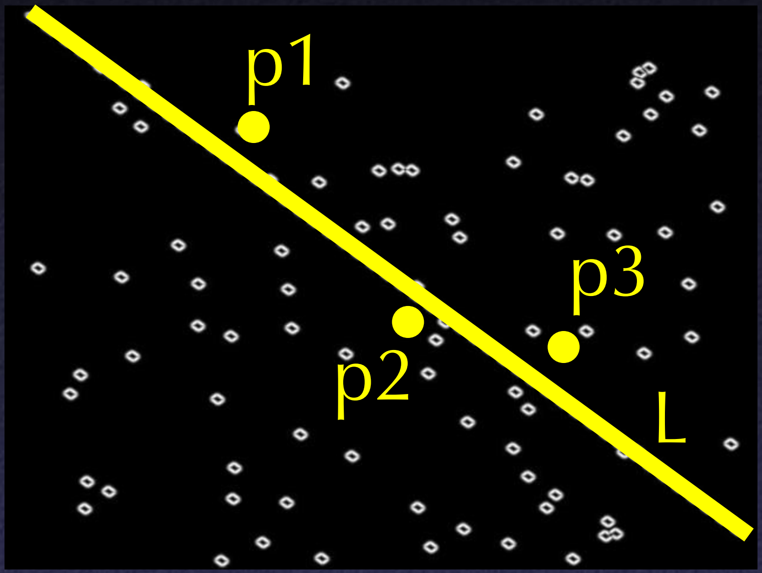
Line through two points

- How compute "support" for L?
  - ∑ G(q) |N(p)•N(q)|
  - Optimize L to fit "inliers"
  - etc.

RANSAC for Line Detection

Many possible variants:

- How to choose L?
  - Point (and local gradient)
  - Point and angle
  - Two points
  - Three points
  - etc.



Line through three points

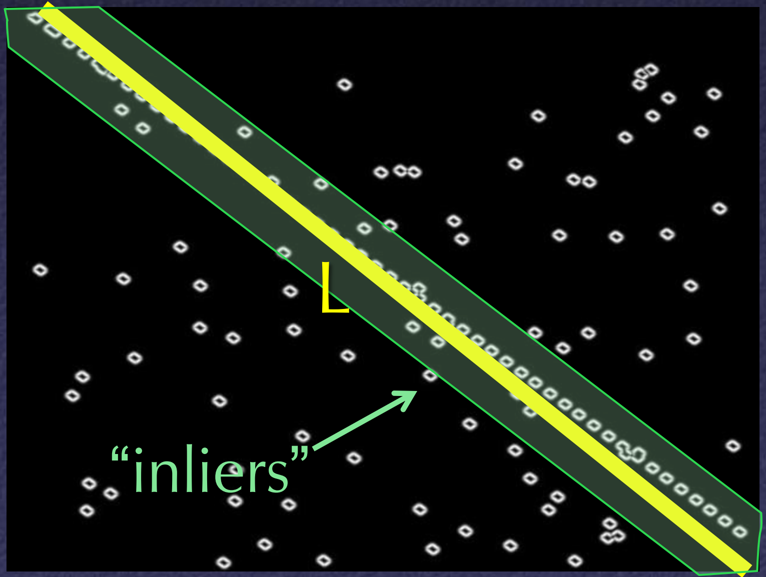
- How compute "support" for L?
  - $\sum G(q)\ |N(p) \bullet N(q)|$
  - Optimize L to fit "inliers"
  - etc.

RANSAC for Line Detection

Many possible variants:

- How to choose L?
  - Point (and local gradient)
  - Point and angle
  - Two points
  - Three points
  - etc.
- How compute "support" for L?
  - $\sum G(q)\ |N(p) \bullet N(q)|$
  - Optimize L to fit "inliers"
  - etc.

$$Support(L) = \sum_{q \in L} G(q)\ |N(p) \cdot N(q)|$$

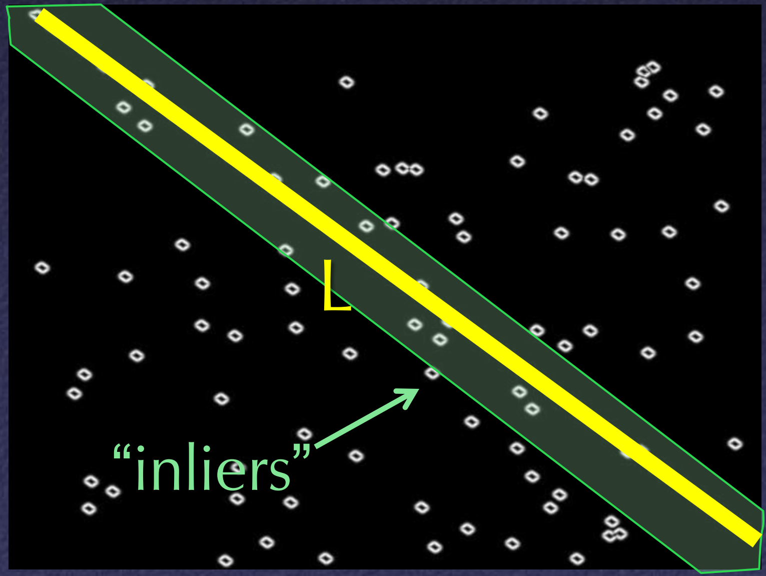


RANSAC for Line Detection

Many possible variants:

- How to choose L?
  - Point (and local gradient)
  - Point and angle
  - Two points
  - Three points
  - etc.
- How compute “support” for L?
  - $\sum G(q)\ |N(p) \cdot N(q)|$
  - Optimize L to fit “inliers”
  - etc.

$$Support(L) = \sum_{q \in L} G(q)\ |N(p) \cdot N(q)|$$



RANSAC for Line Detection

At beginning:

- Compute gradient direction N and magnitude G

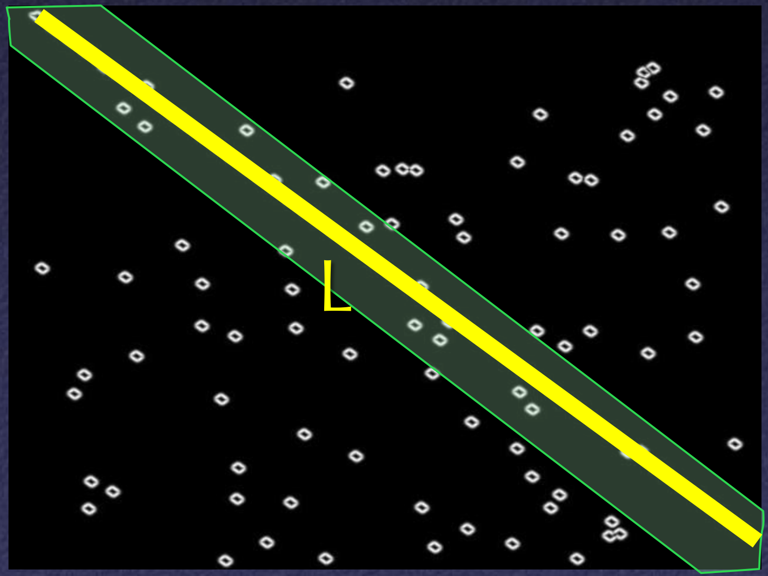
Iterate:

- Randomly choose a pixel p
- Choose a line L through p
- Compute how well other pixels "support" L

At end:

- Report the line L* with the most "support"

How many iterations?
What is running time?

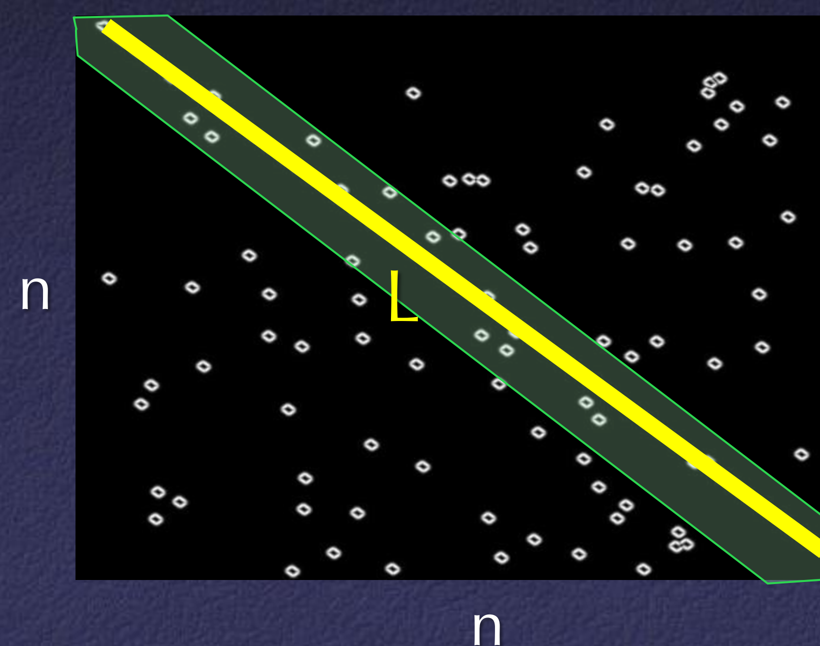


RANSAC for Line Detection

S = # samples (iterations)
T = time to evaluate each sample
n = width/height of image
d = degrees of freedom in
line parameterization

Running time = $O(ST) = O(n^d)$

- $S = O(n^{d-1})$
- $T = O(n)$



RANSAC in General

RANdom SAmple Consensus

Take many random subsets of data

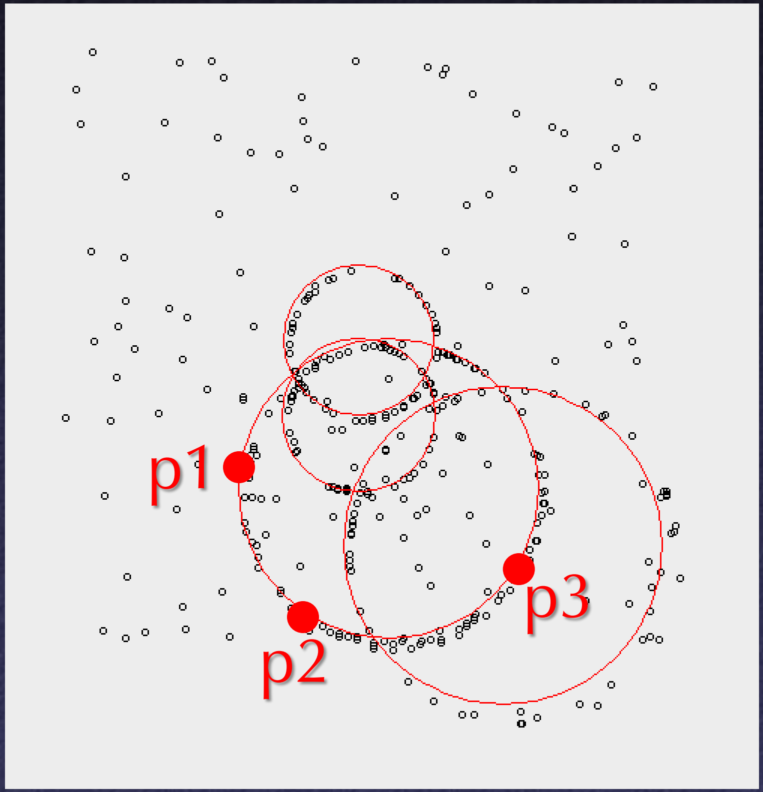
- Compute fit for each sample
- See how many points agree
- Remember the best

What else could this algorithm detect?

RANSAC for Circle Detection

Detecting circles:

- Randomly choose three pixels p1, p2, and p3
- Compute a circle C through p1, p2, and p3
- Compute how well other pixels “support” C

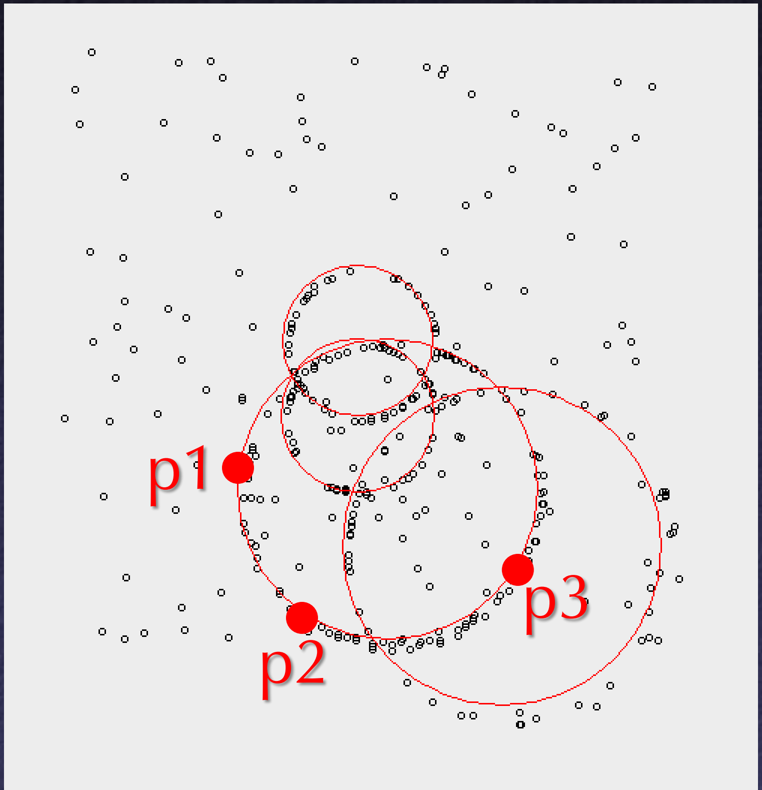


RANSAC for Circle Detection

Detecting circles:

- Randomly choose three pixels p1, p2, and p3
- Compute a circle C through p1, p2, and p3
- Compute how well other pixels "support" C

What is the running time?

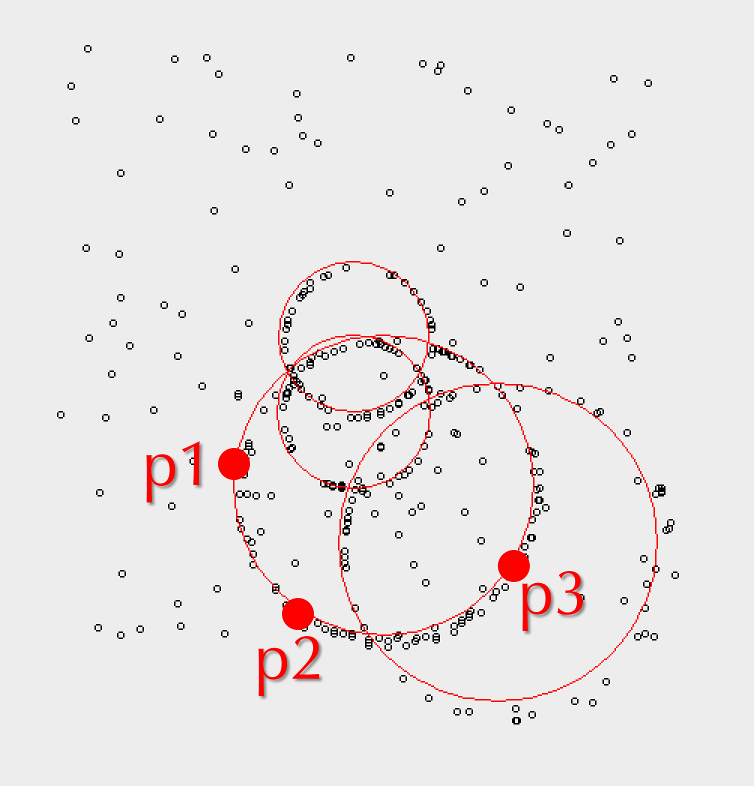


RANSAC for Circle Detection

Detecting circles:

- Randomly choose three pixels p1, p2, and p3
- Compute a circle C through p1, p2, and p3
- Compute how well other pixels "support" C

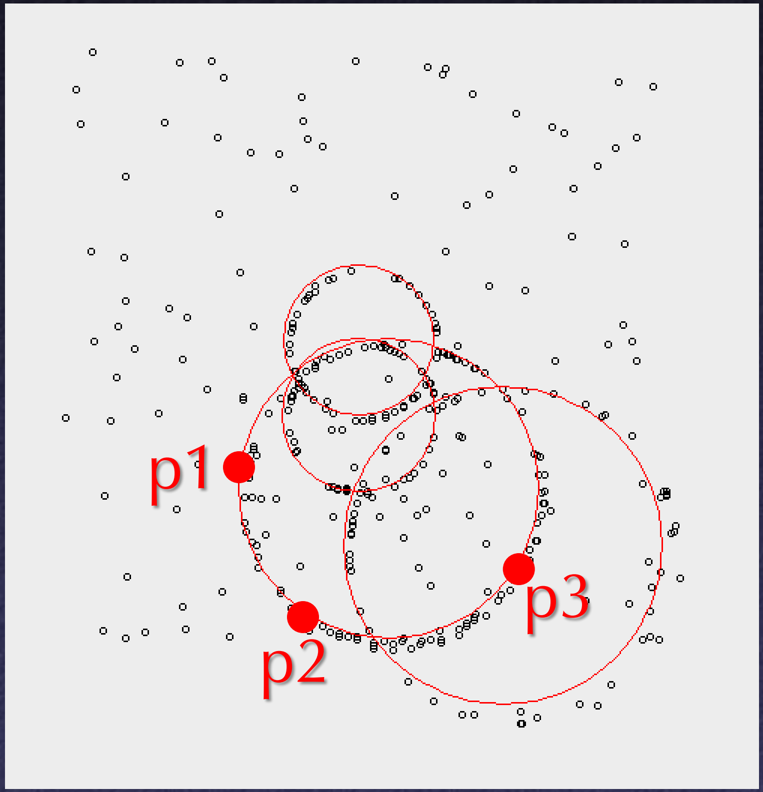
How can we improve the running time?



RANSAC for Circle Detection

Possible parameterizations for circles:

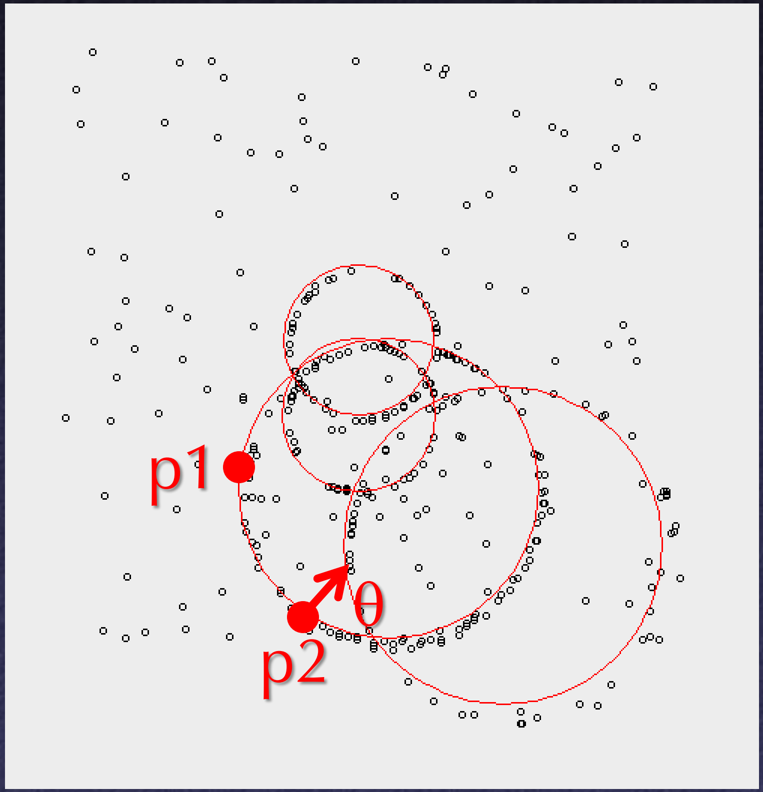
- 6 dof: three points
- 5 dof: two points, one angle
- 4 dof: one point, one angle,
  one radius



RANSAC for Circle Detection

Possible parameterizations for circles:

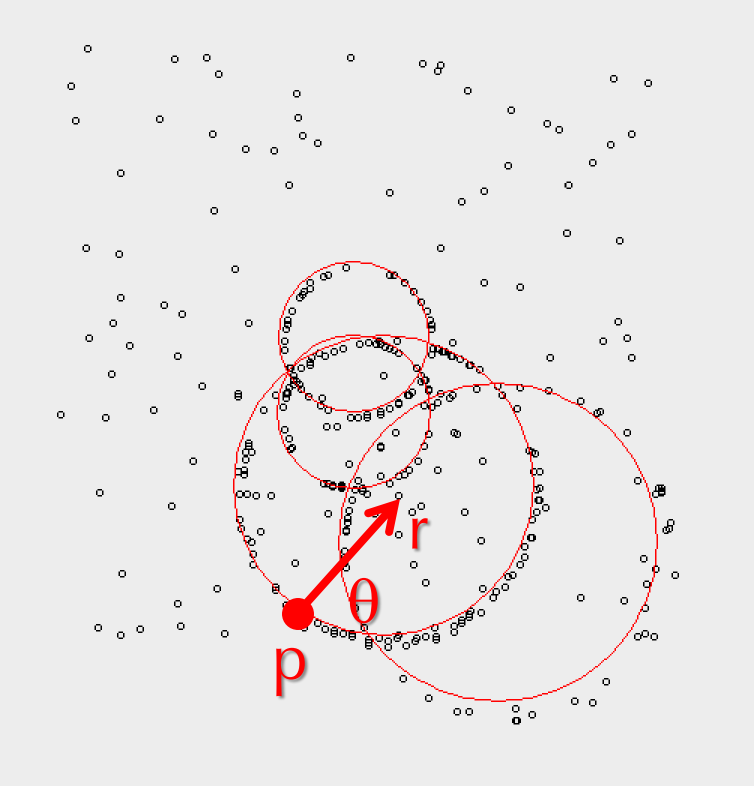
- 6 dof: three points
- 5 dof: two points, one angle
- 4 dof: one point, one angle, one radius



RANSAC for Circle Detection

Possible parameterizations for circles:

- 6 dof: three points
- 5 dof: two points, one angle
- 4 dof: one point, one angle, one radius
- 3 dof: one point, one radius



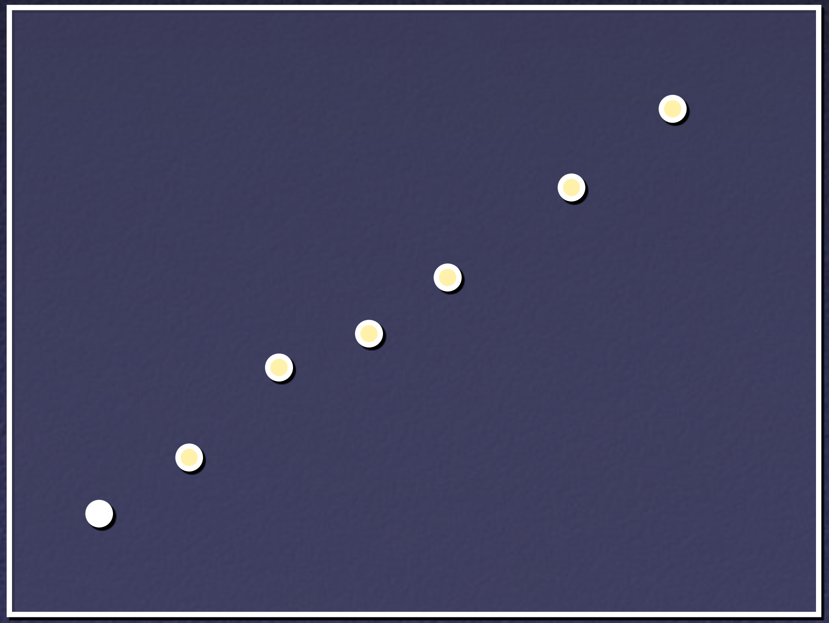
Line Detection

Two common algorithms:

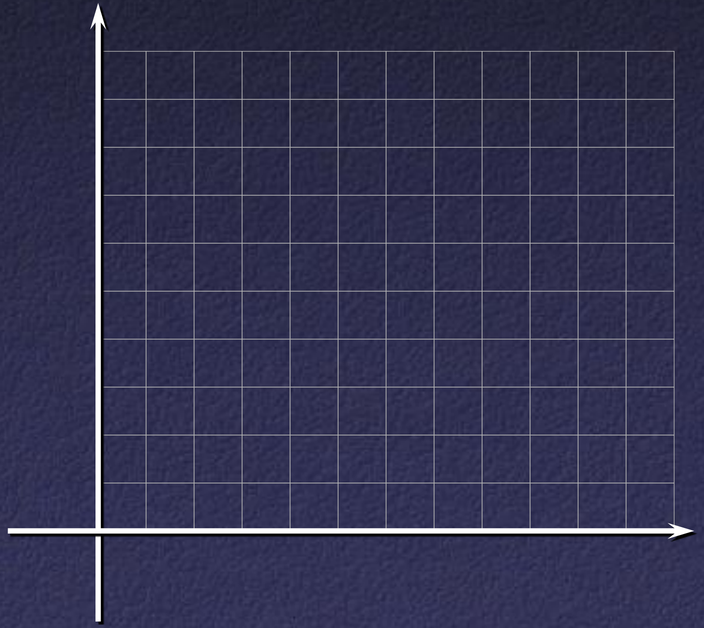
- RANSAC
- Hough transform ←

Hough Transform

Like RANSAC, except visit pixels p one-by-one and accumulate “support” (vote) for all primitives containing p in hash table bins



Image



Hough Space

Hough Transform

At beginning:

- Initialize all Hough space bins to zero

For each pixel sample p:

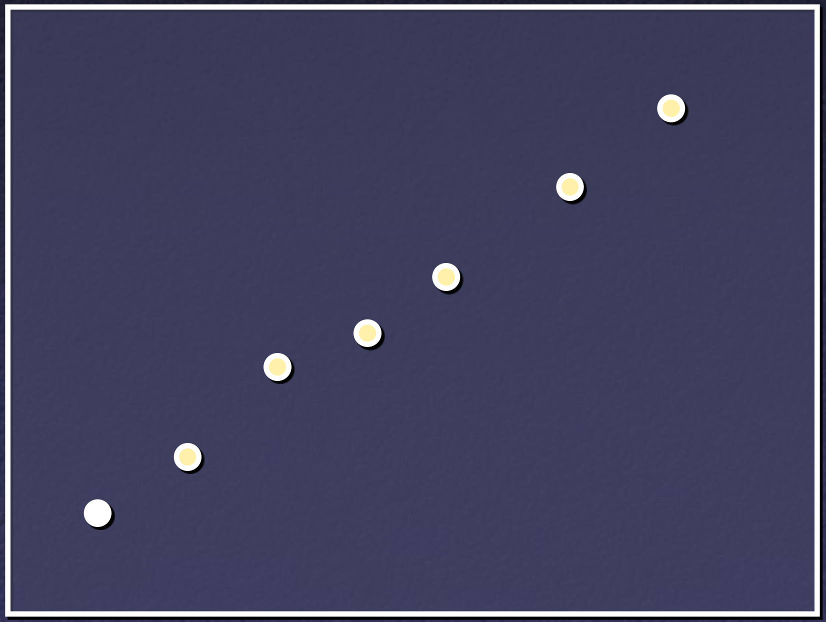
- Add support to all Hough space bins representing primitives containing p

At end:

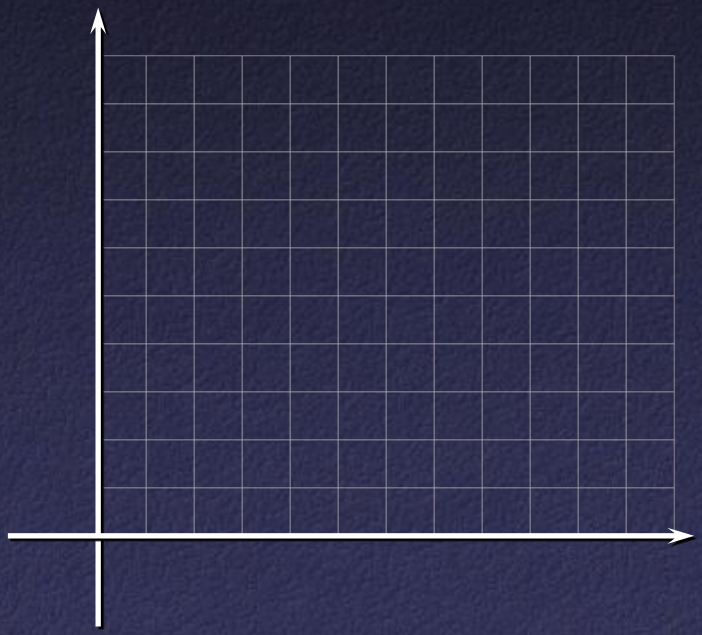
- Find the Hough space bin(s) with the most support

Hough Transform for Line Detection

Example:



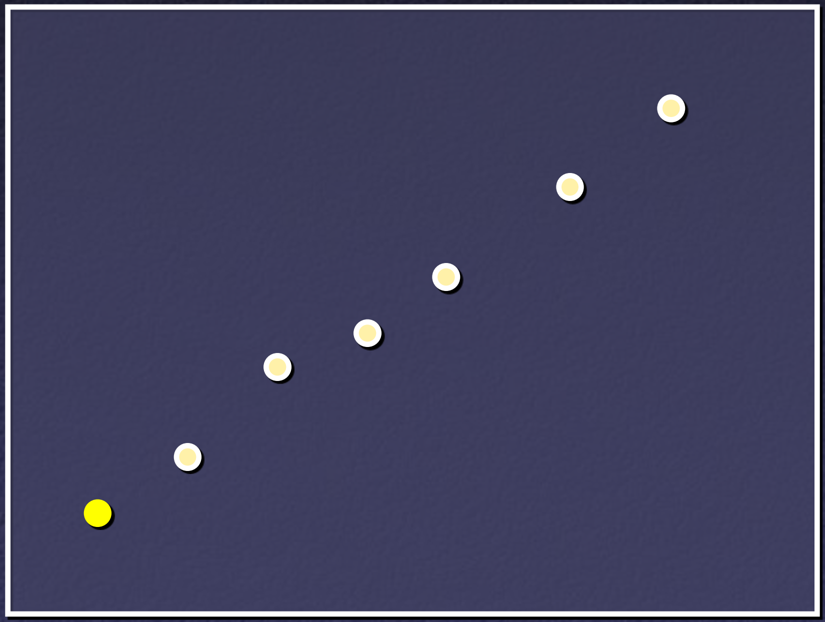
Image



Hough Space

Hough Transform for Line Detection

Example:

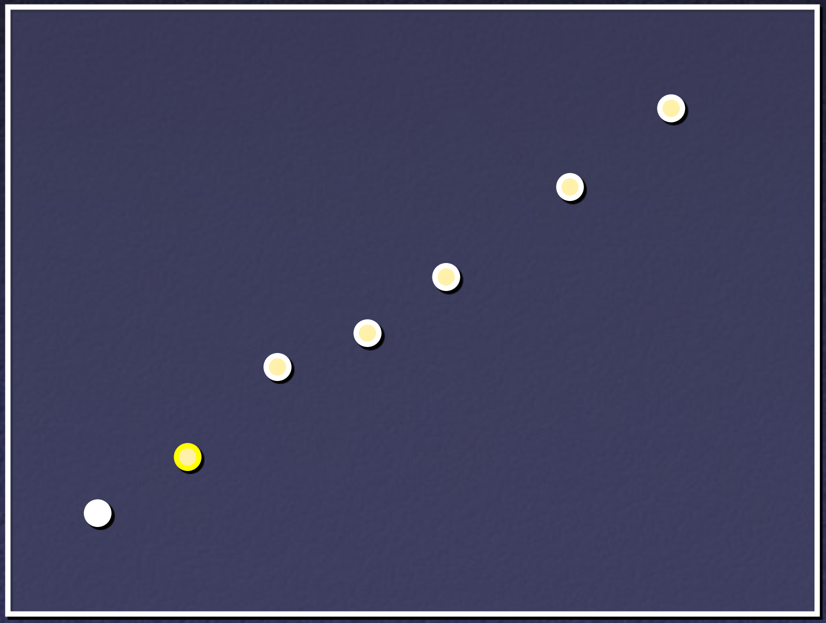


Image

Hough Space

Hough Transform for Line Detection

Example:

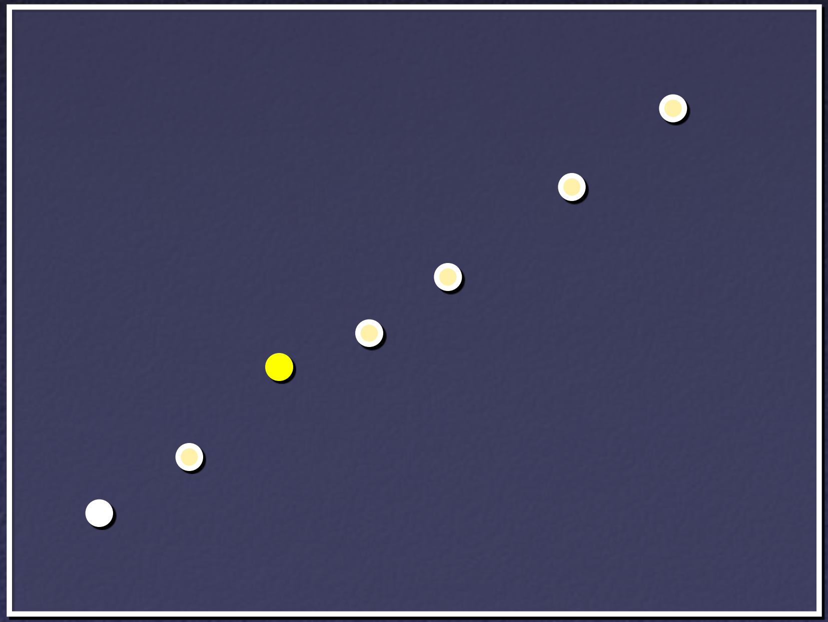


Image

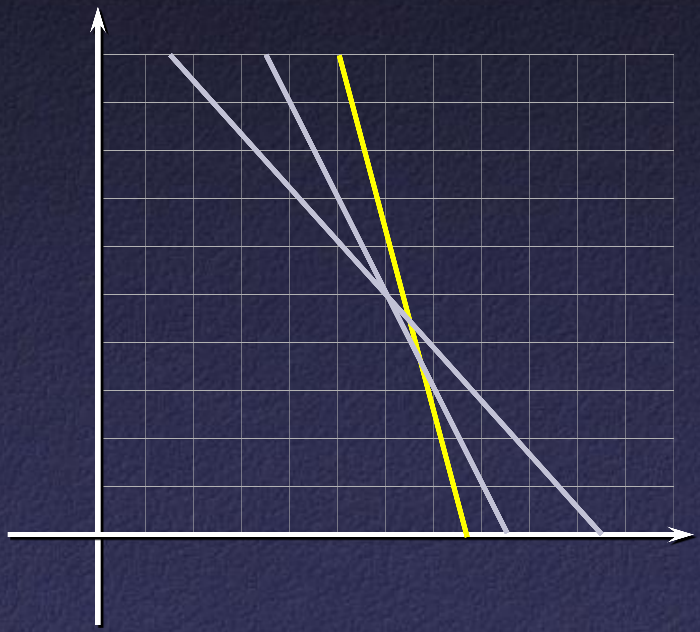
Hough Space

Hough Transform for Line Detection

Example:



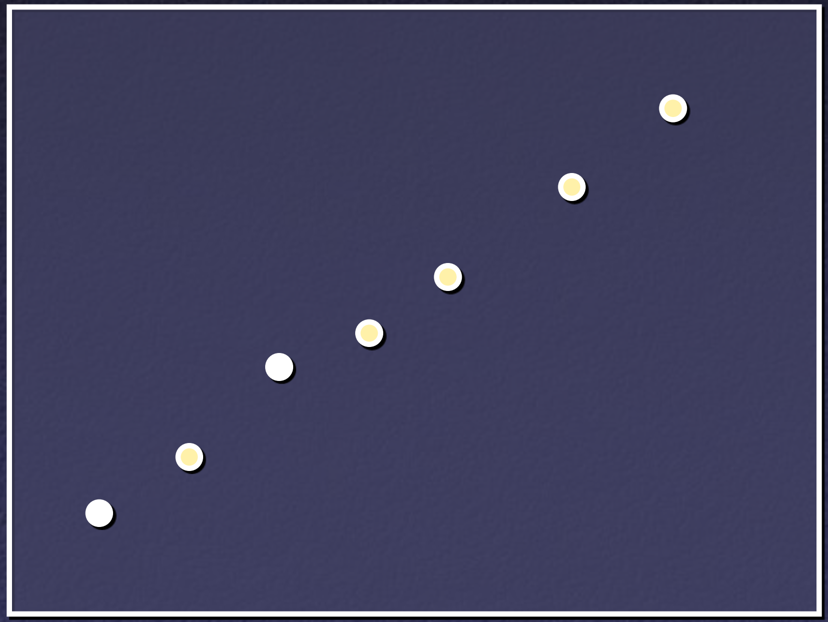
Image



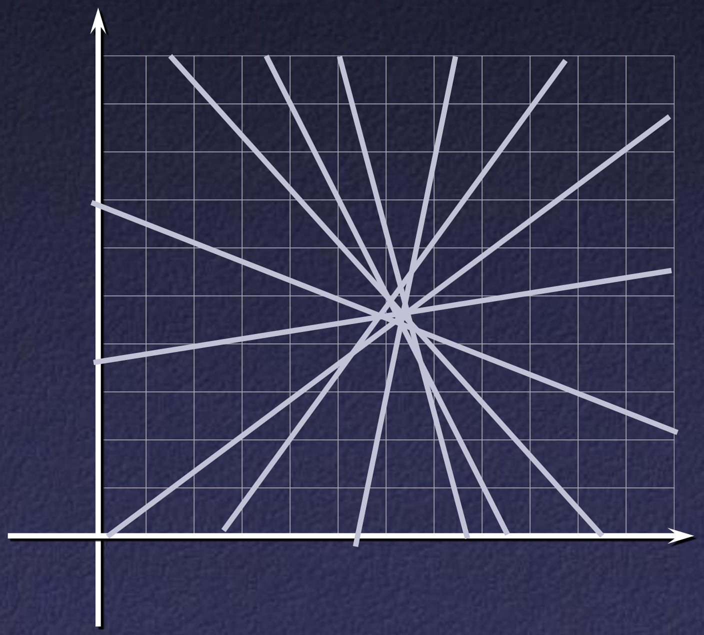
Hough Space

Hough Transform for Line Detection

Example:



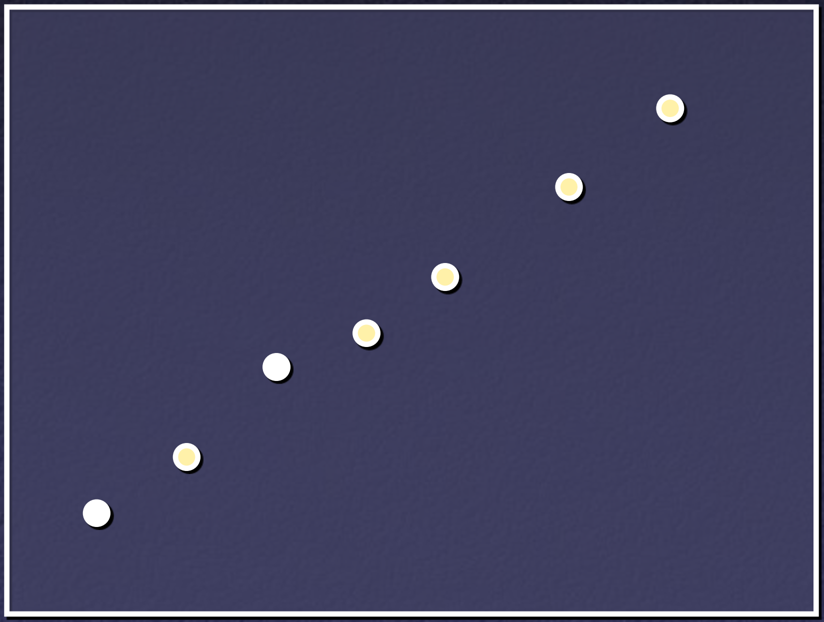
Image



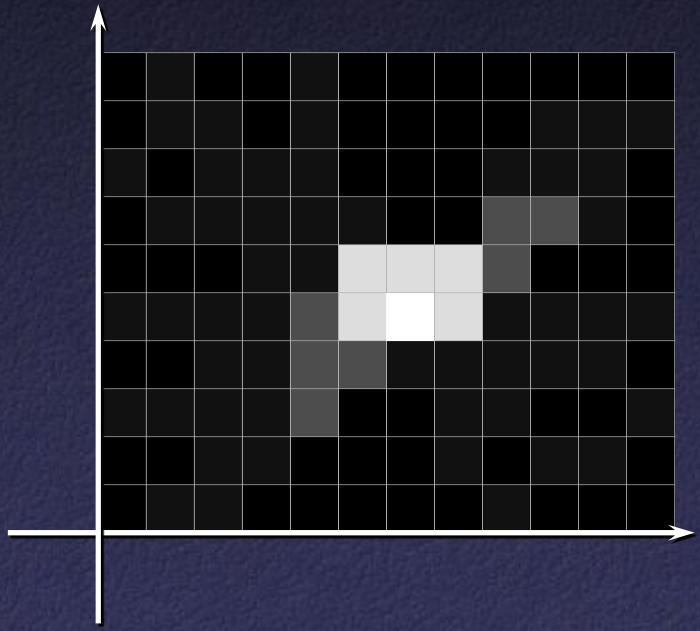
Hough Space

Hough Transform for Line Detection

Example:



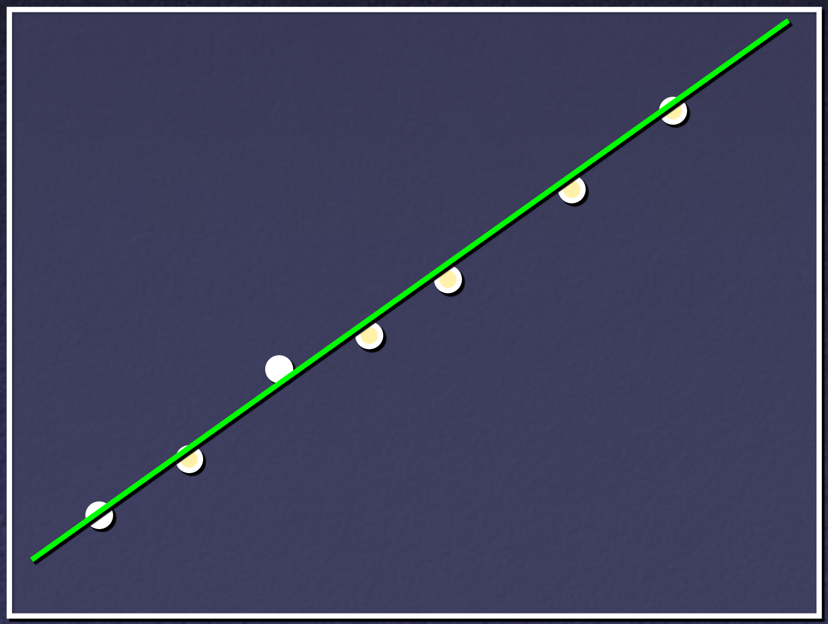
Image



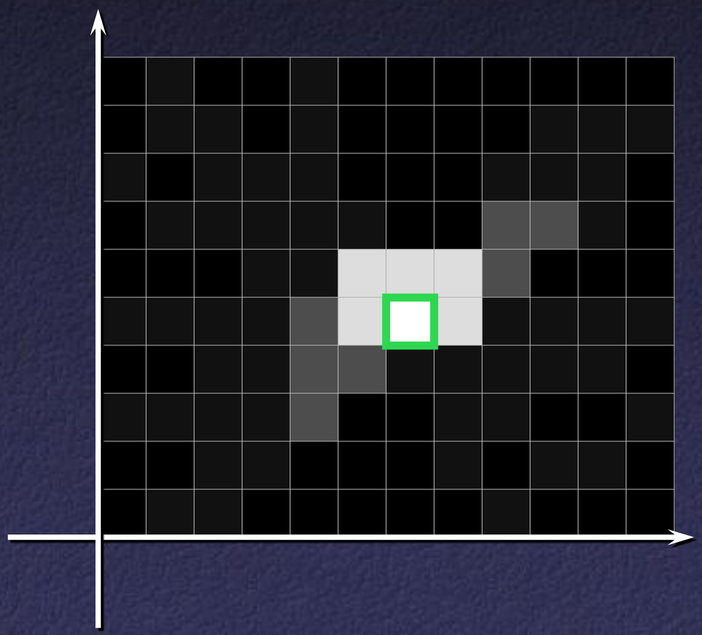
Hough Space

Hough Transform for Line Detection

Example:



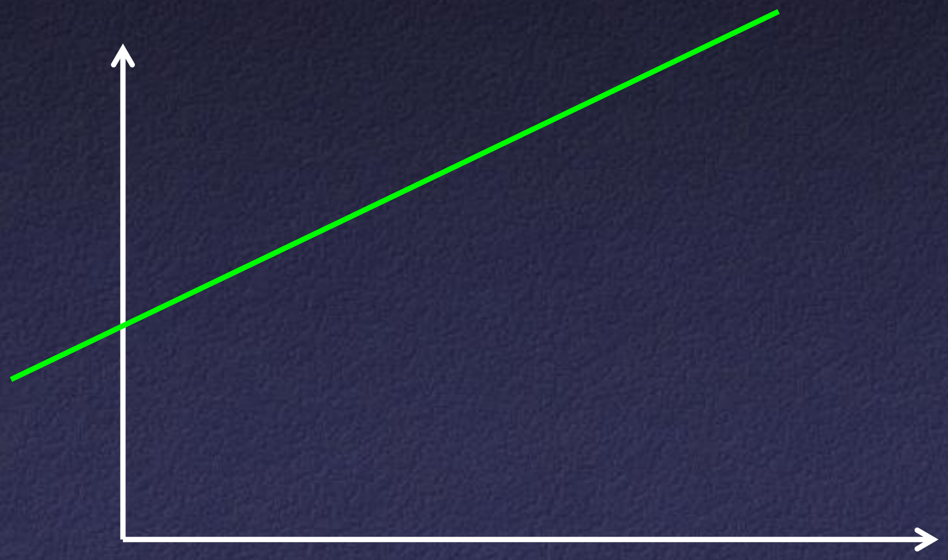
Image



Hough Space

Hough Transform for Line Detection

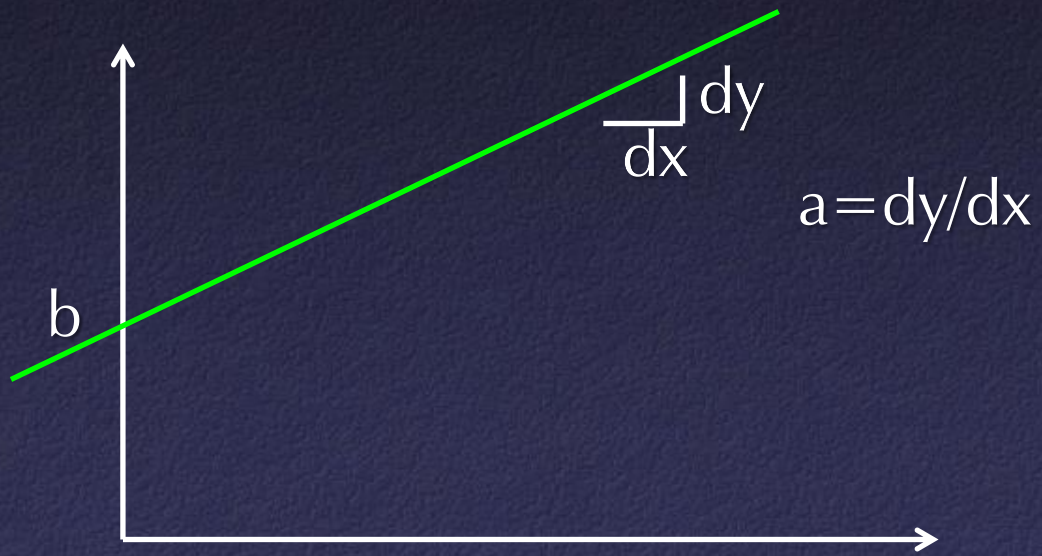
Key question: how to parameterize Hough space?



Hough Transform for Line Detection

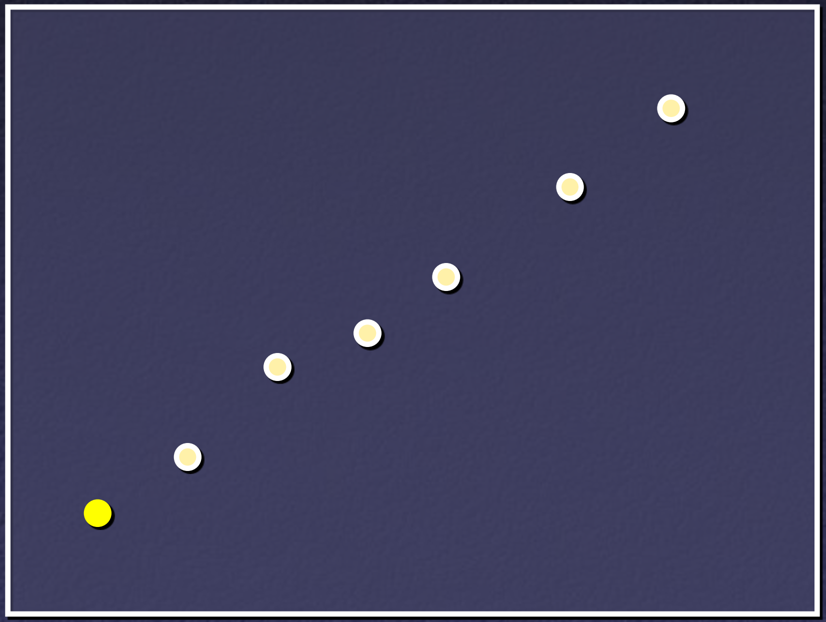
A 2 dof parameterization for lines: $y = ax + b$

Parameters: a and b

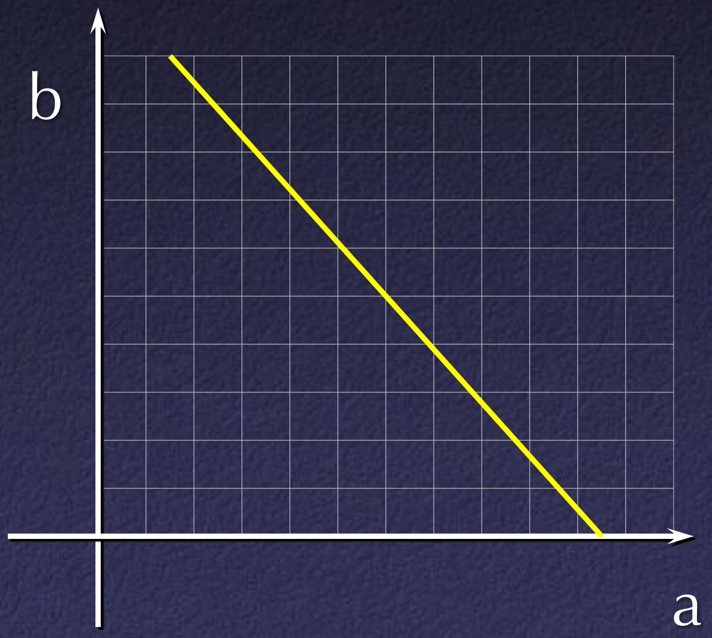


Hough Transform for Line Detection

Every point in image lies on "line" of bins in Hough space with this parameterization



Image

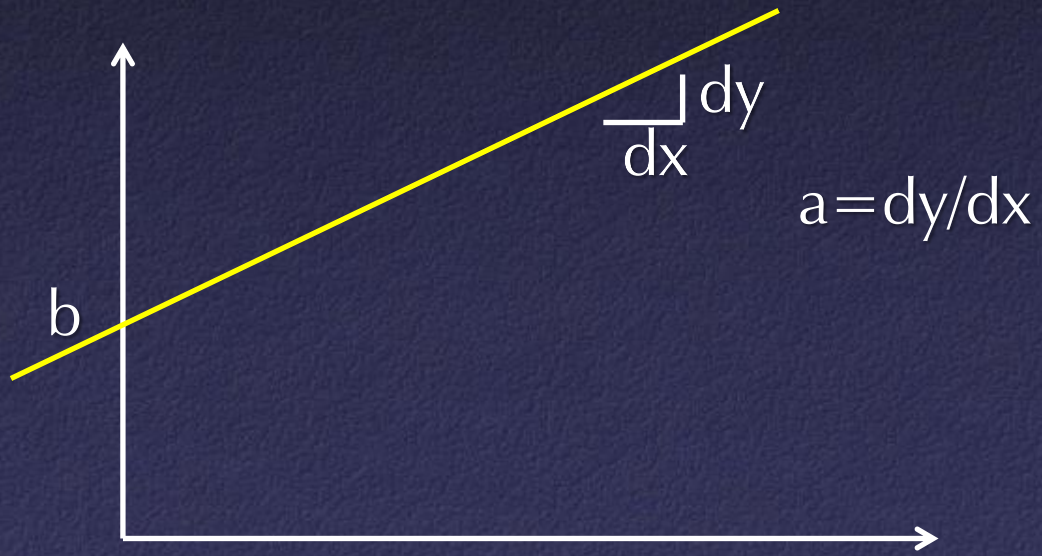


Hough Space

Hough Transform for Line Detection

Problems with slope / intercept parameterization

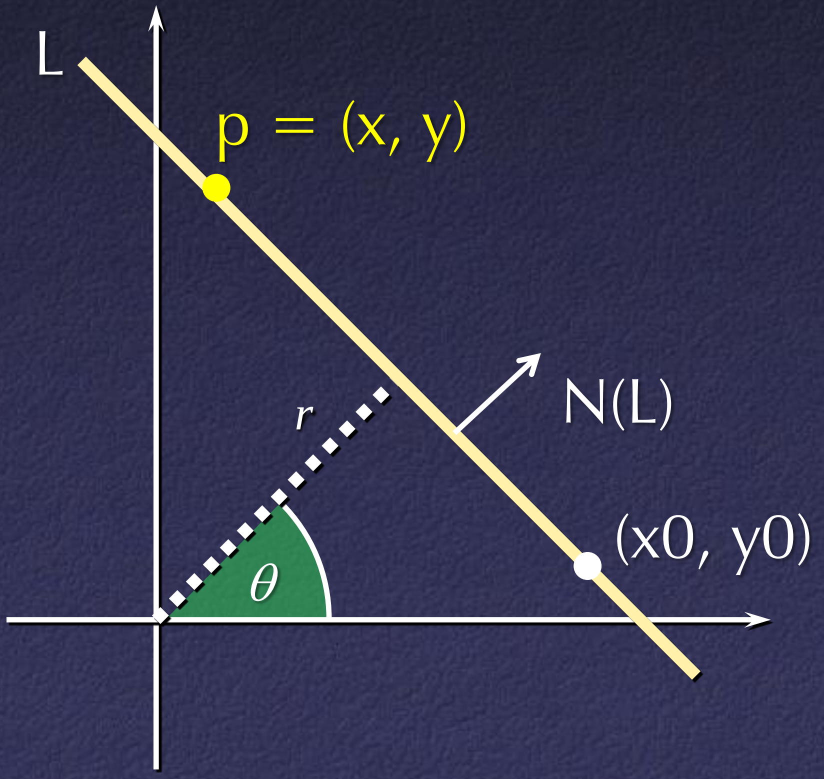
- Non-uniform sampling of directions
- Can't represent vertical lines



Hough Transform for Line Detection

Alternative: angle / distance parameterization

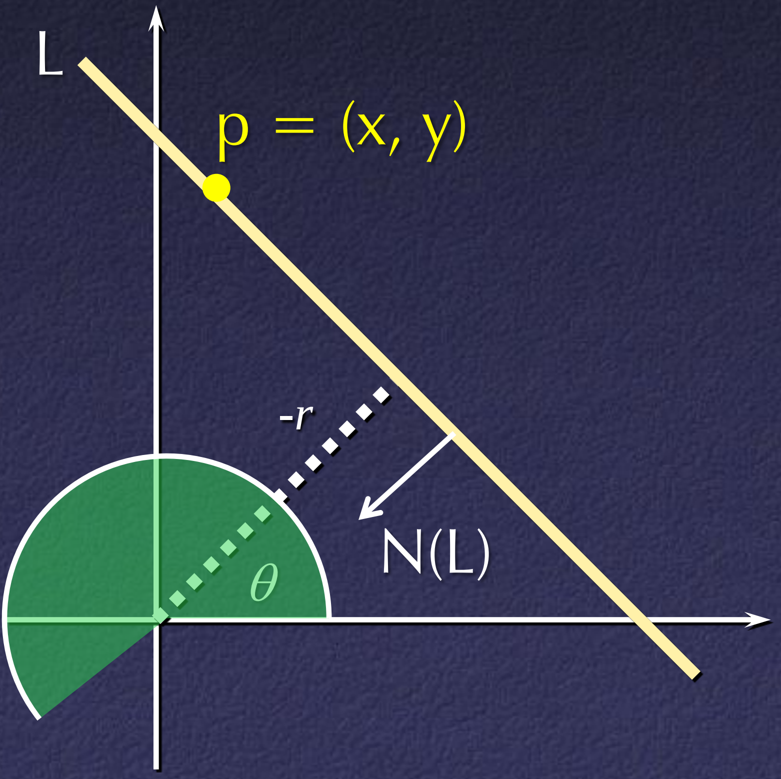
- Line represented as (r, θ) where
  - $x \cos \theta + y \sin \theta = r$
  - $r = -\cos \theta \cdot x0 - \sin \theta \cdot y0$
  - $N(L) = (\cos \theta, \sin \theta)$
  - $dist(L, p) = x \cos \theta + y \sin \theta - r$



Hough Transform for Line Detection

Alternative: angle / distance parameterization

- Line represented as (r, θ) where
  - $x \cos\theta + y \sin\theta = r$
  - $r = -\cos\theta \cdot x0 - \sin\theta \cdot y0$
  - $N(L) = (\cos\theta, \sin\theta)$
  - $dist(L, p) = x \cos\theta + y \sin\theta - r$
  - $L(r, \theta) \approx L(-r, \theta + \pi)$



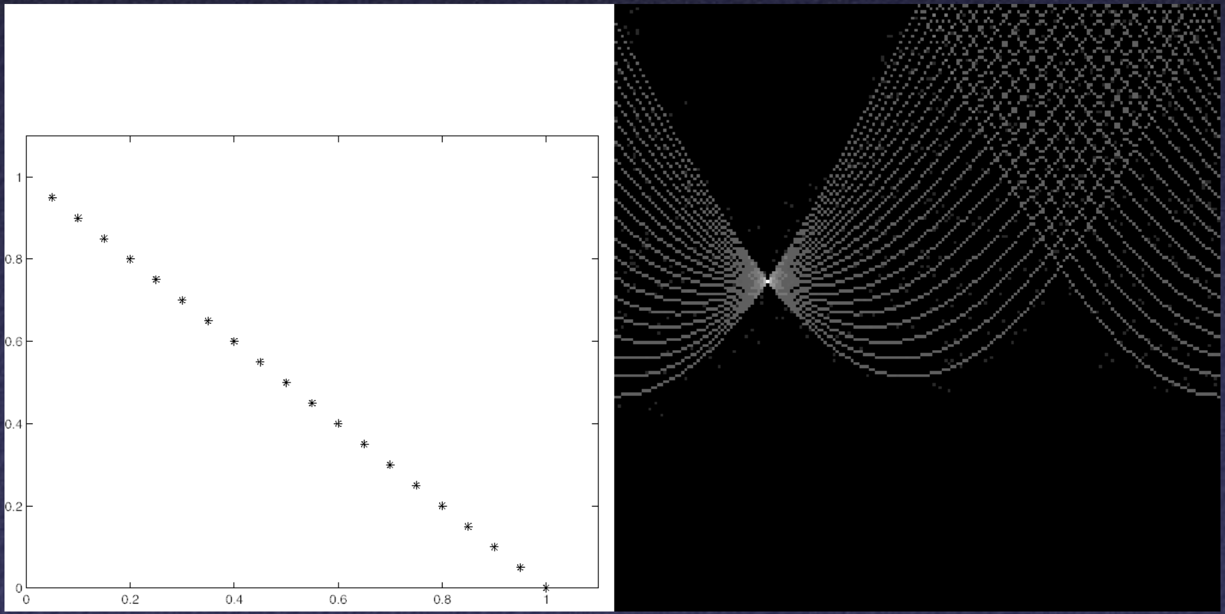
Hough Transform for Line Detection

Alternative: angle / distance parameterization

- Line represented as (r,θ)

  + *Uniform sampling of angles*

  -- *Lines through point lie on sinusoid in* (r,θ)



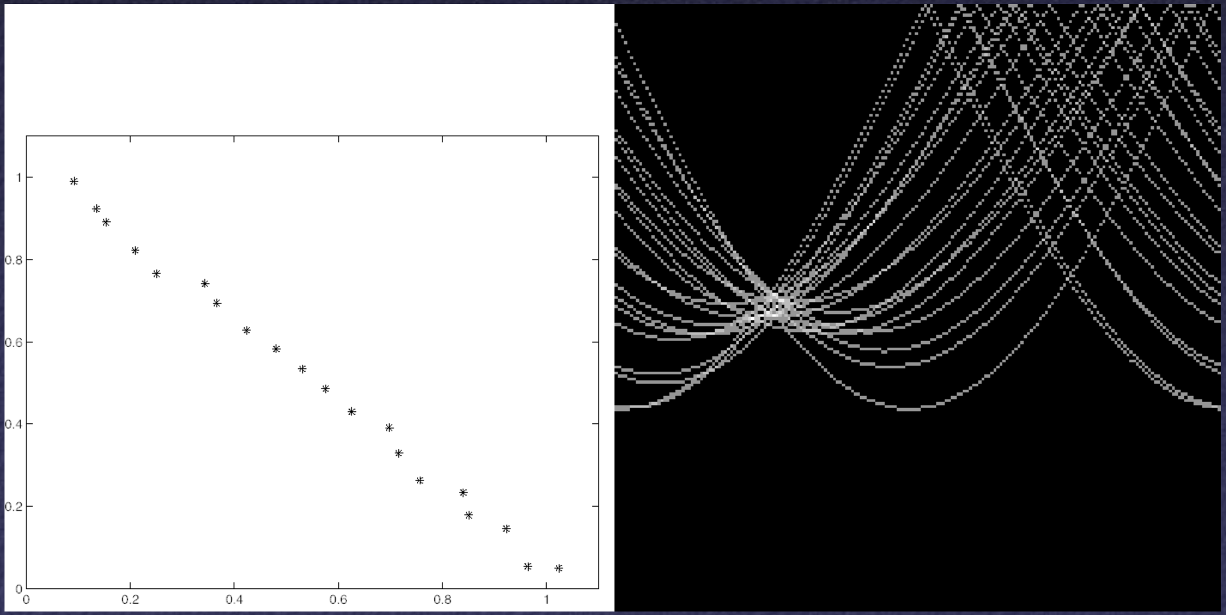
Hough Transform for Line Detection

Alternative: angle / distance parameterization

- Line represented as (r, θ)

  + *Uniform sampling of angles*

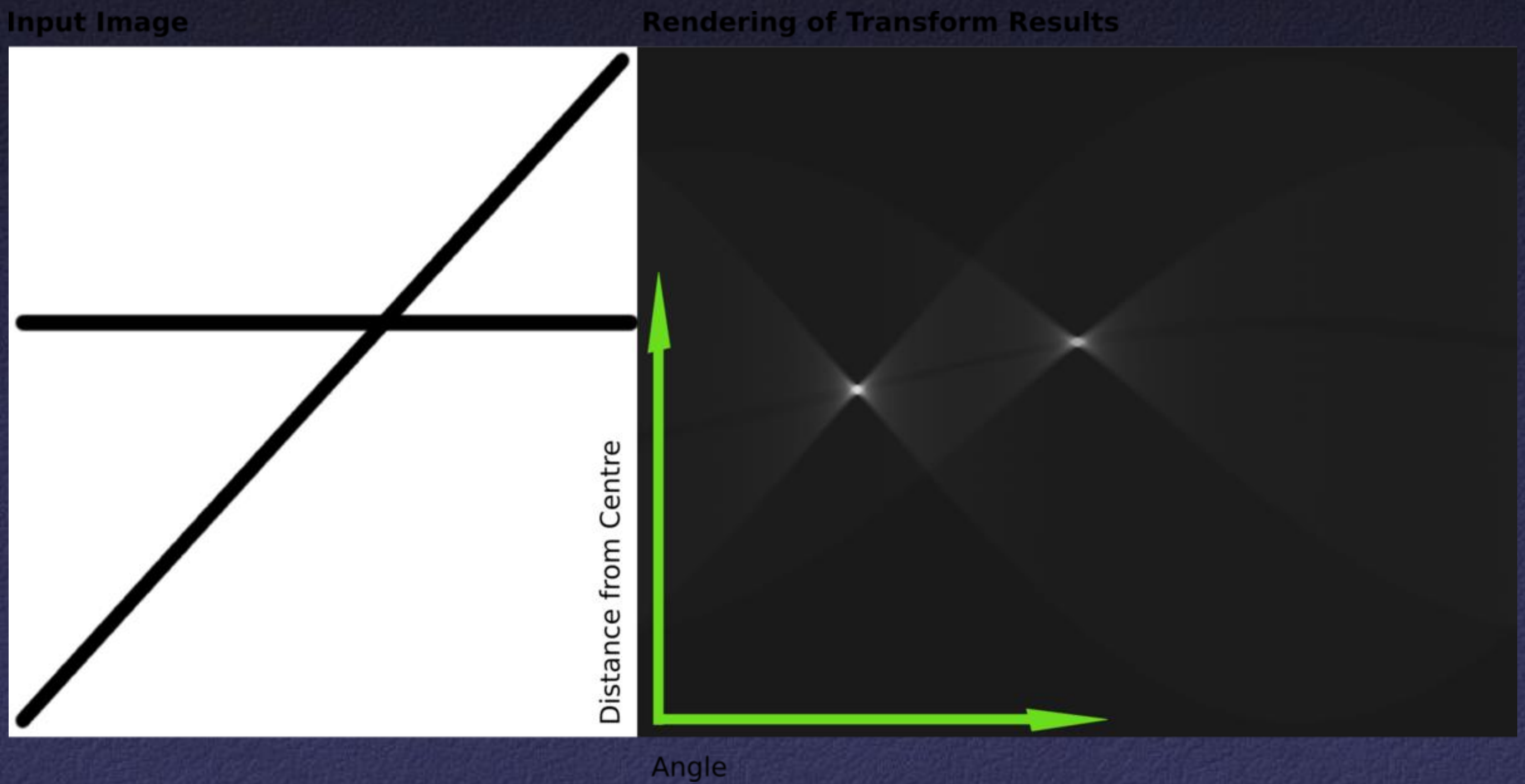
  -- *Lines through point*
  lie on sinusoid in (r, θ)



Hough Transform for Line Detection

Most people use angle / distance parameterization

- Line represented as (r, θ)



Hough Transform for Line Detection

Issue: How to select bucket size?

Hough Transform for Line Detection

Issue: How to select bucket size?

- Too small: poor performance on noisy data
- Too large: poor accuracy, possibility of false positives

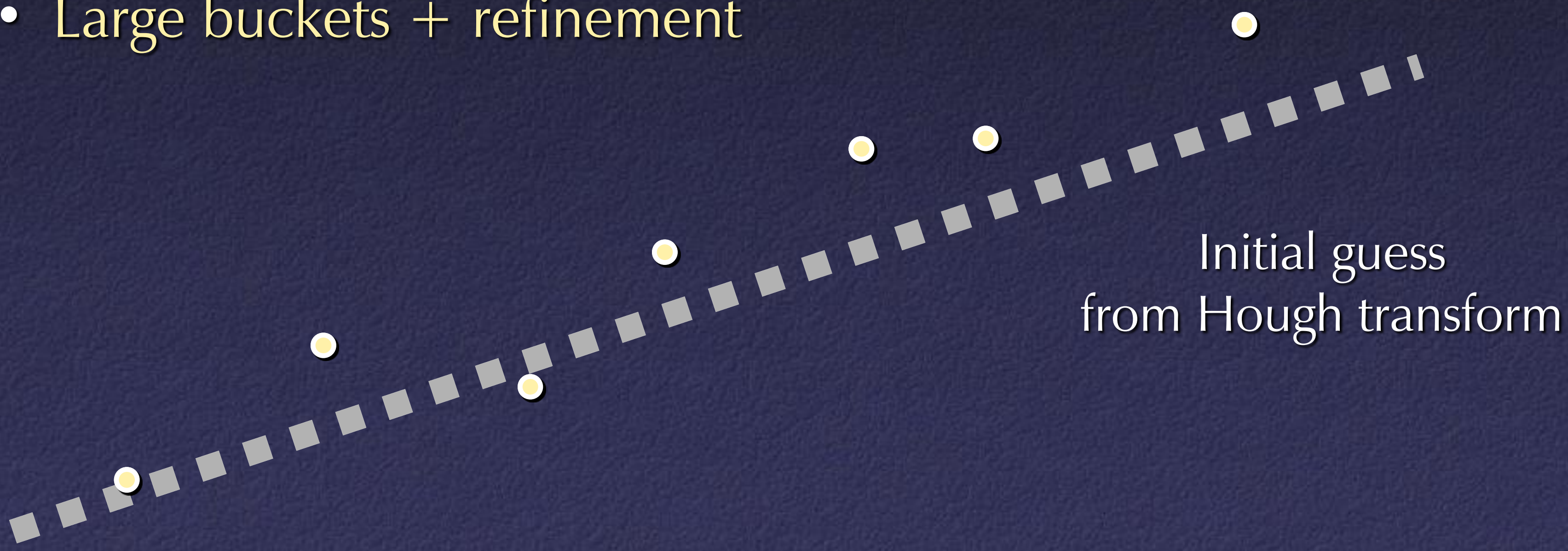
Hough Transform for Line Detection

Issue: How to select bucket size?

- Too small: poor performance on noisy data
- Too large: poor accuracy, possibility of false positives

One solution:

- Large buckets + refinement



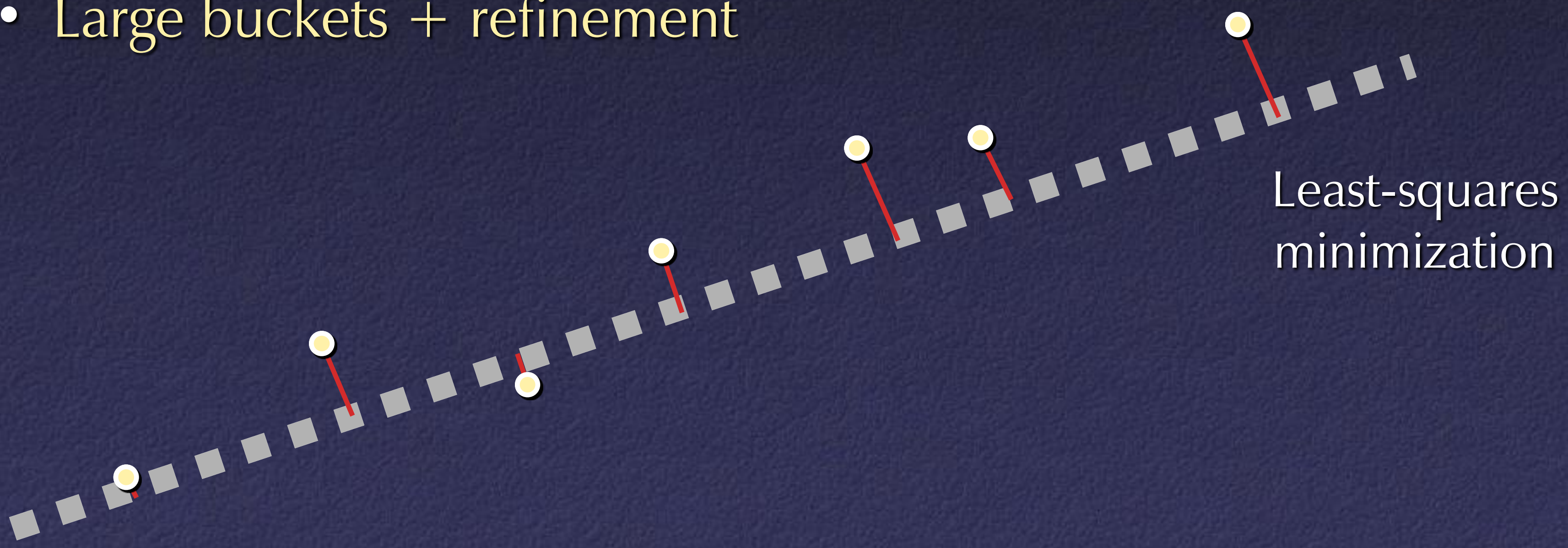
Hough Transform for Line Detection

Issue: How to select bucket size?

- Too small: poor performance on noisy data
- Too large: poor accuracy, possibility of false positives

One solution:

- Large buckets + refinement



Hough Transform for Line Detection

Issue: How to select bucket size?

- Too small: poor performance on noisy data
- Too large: poor accuracy, possibility of false positives

One solution:

- Large buckets + refinement

Hough Transform in General

What else can be detected with a Hough transform?

- Circles
- Ellipses
- Boxes
- Symmetries
- etc.

Anything that can be parameterized
(in a small number of dimensions)

Hough Transform for Circle Detections

2D circles have 3 degrees of freedom

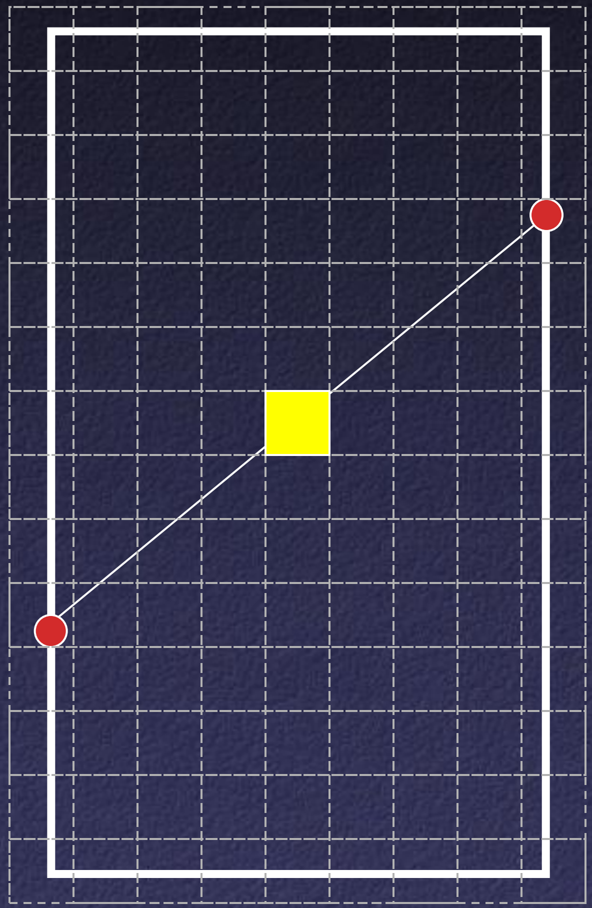
- Possible parameterization = 2D position and radius

So, each pixel gives rise to 2D sheet of values in 3D Hough space

Hough Transform for Symmetry Detection

Symmetry transform:

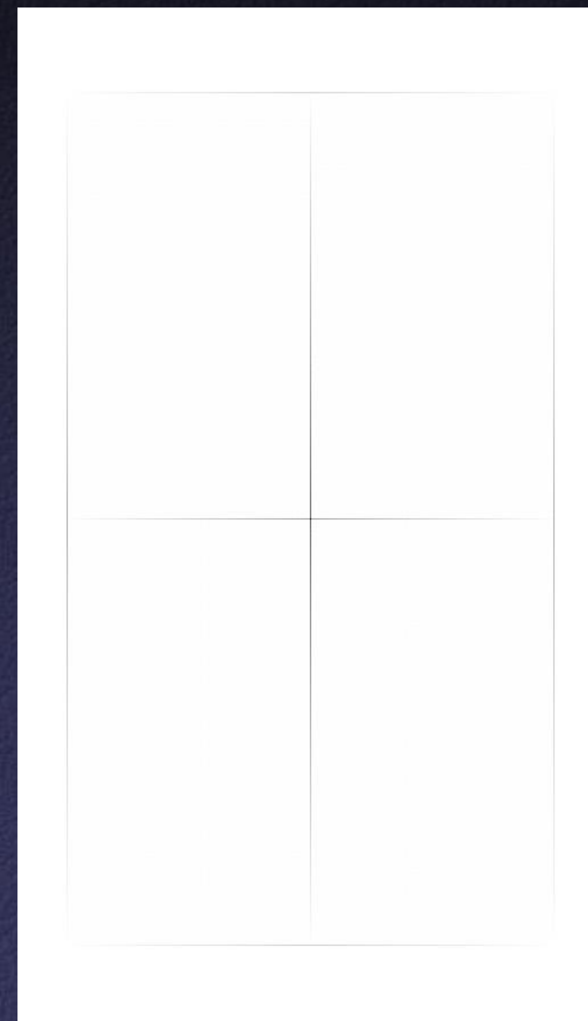
- Vote for midpoints between pixels



Hough Transform for Symmetry Detection

Symmetry transform:

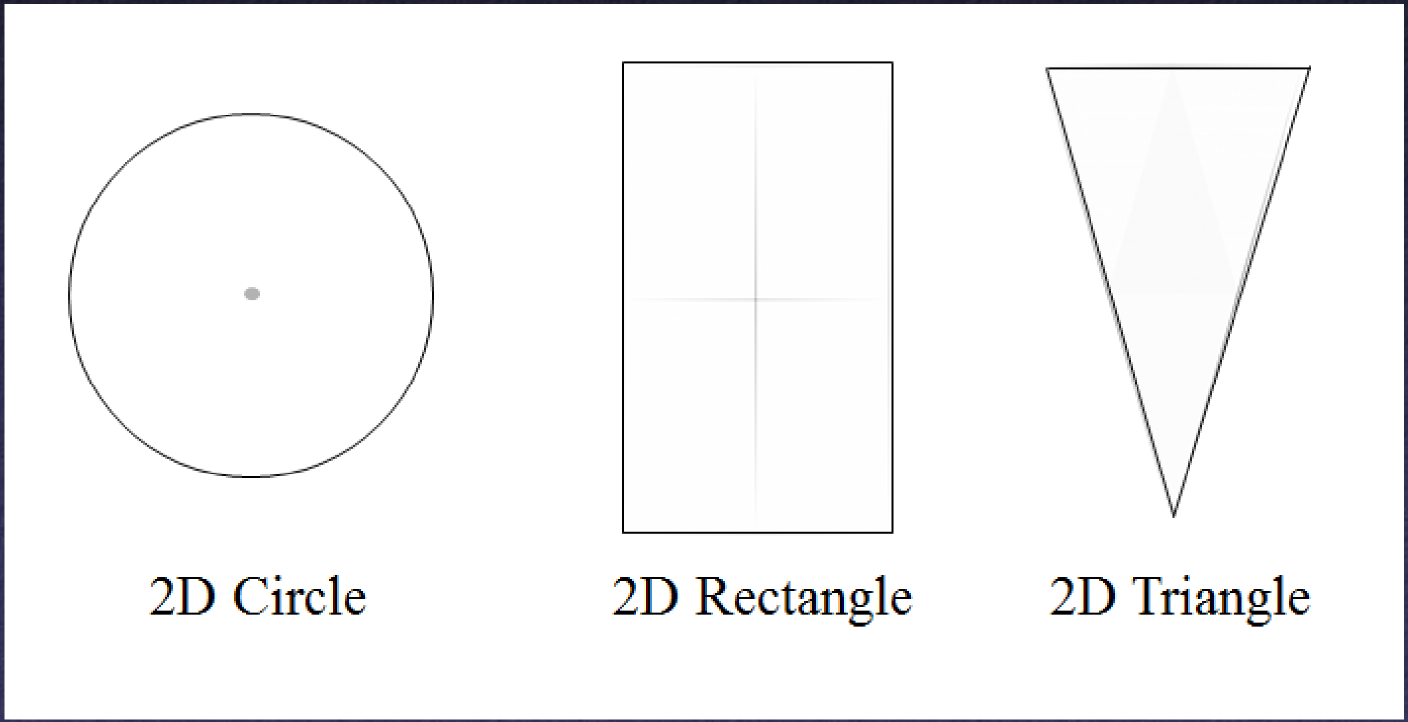
- Vote for midpoints between pixels



Hough Transform for Symmetry Detection

Symmetry transform:

- Vote for midpoints between pixels

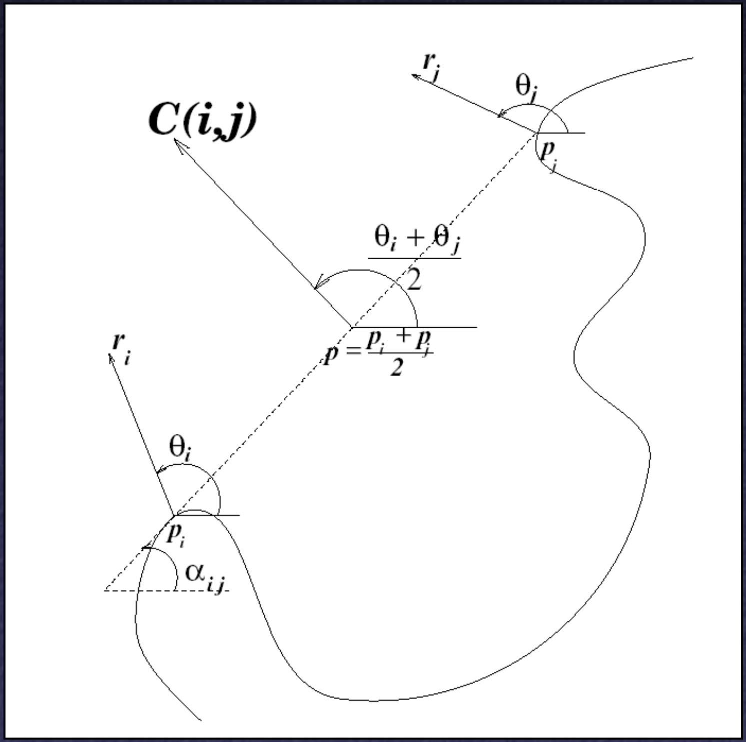


Hough Transform for Symmetry Detection

Symmetry transform:

- Vote for midpoints between pixels
- Weight votes by functions of distances, gradients, directions, etc.

$$C(i,j) = D_\sigma(i,j)P(i,j)r_i r_j$$

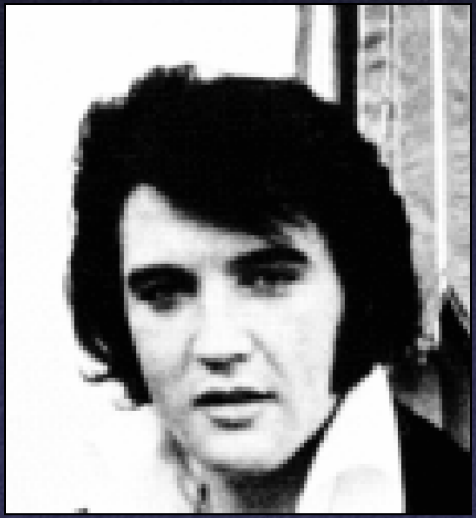


Reisfeld, et. al.

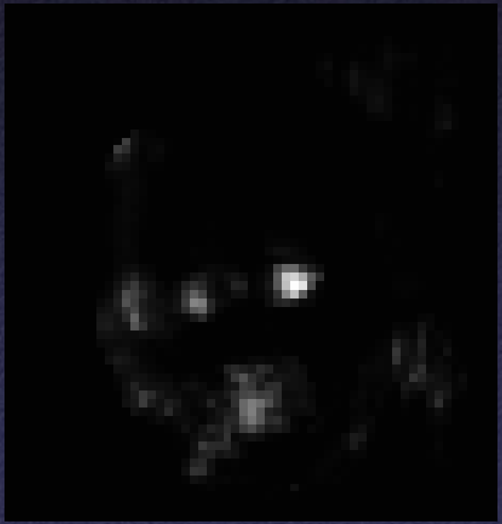
Hough Transform for Symmetry Detection

Symmetry transform:

- Used for eye detection!!!



Input



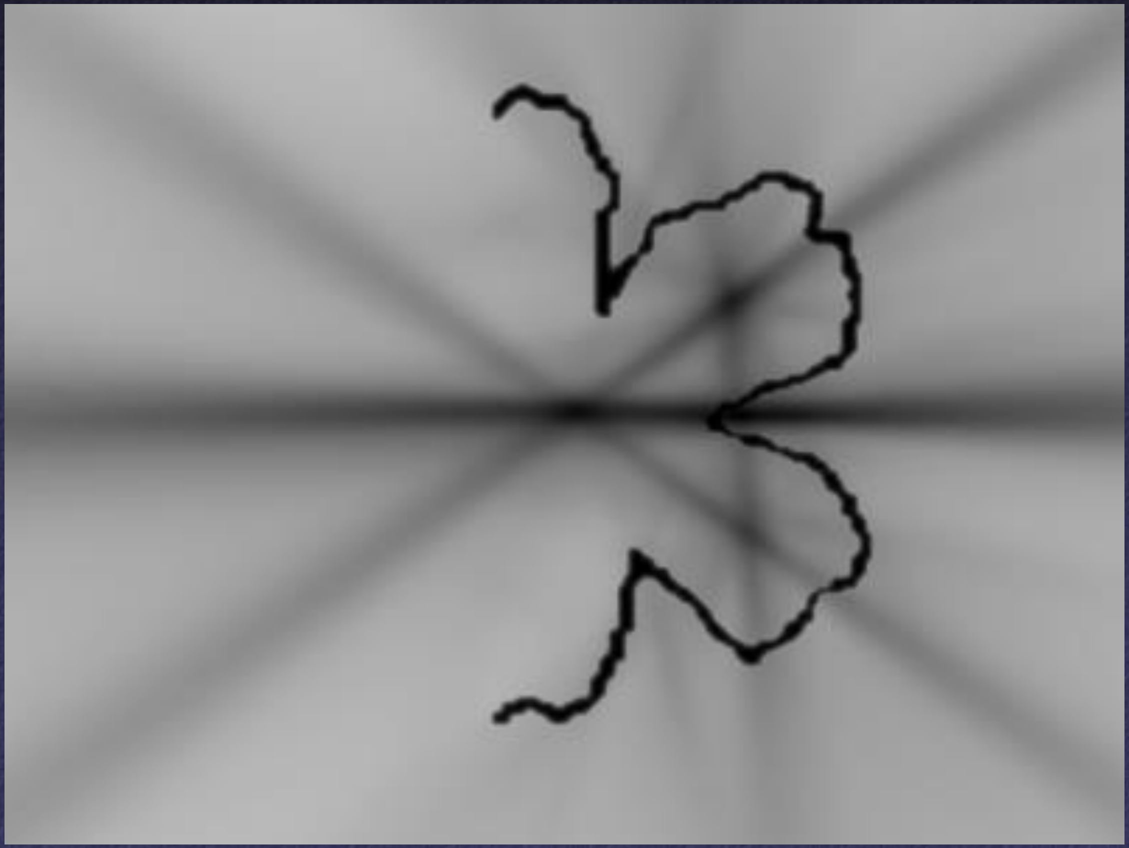
Hough Votes

Feature detections

Hough Transform for Symmetry Detection

Reflective symmetry transform:

- Vote for bisector lines



Hough Transform and RANSAC

Very general computational techniques:

- Useful for detecting anything
  that can be parameterized
  in a low-dimensional space

Hough Transform vs. RANSAC?

How are algorithms similar / different?

- This question is part of the thought exercise for assignment #1

Assignment #1

http://www.cs.princeton.edu/courses/archive/fall13/cos429/assignment1/assignment1.html

COS 429: Computer Vision, Fall 2013
Assignment 1: Line Detection

Part 1: Thought Exercise

A. In one paragraph, please discuss the relative advantages of using the Hough transform versus RANSAC algorithms when detecting lines in images. Under what conditions is one preferable to the other and vice-versa? How would this change for ellipses, triangles, or other more complex primitives?

B. The finite size of an image implies that, on average, the length in pixels of the visible portions of lines close to the image center *C* is greater than that of lines distant from *C*. Please provide a mathematical formula for how the Hough transform is biased by this effect and explain in a sentence or two how you could counter this bias when computing the Hough transform.

Part 2: Programming Exercise

Your goal for this part of the assignment is to write a MATLAB program that predicts the locations of long, straight lines in an input image. This is just like the warmup exercise, except that now you are detecting lines rather than eyes.

As in the previous assignment, "runme()" should write gray-level images in the output directory where the brightness of each pixel is proportional to the predicted probability of finding a straight line at that pixel in the corresponding image in the input directory. For example, running runme() for the input shown on the left below might produce the output shown in the middle. The rightmost image shows an overlay of the two (where the output has been added to the red channel of the input). More examples can be found here.

Assignment #1

Detect edges locally

- Canny algorithm

Detect lines globally

- Hough algorithm

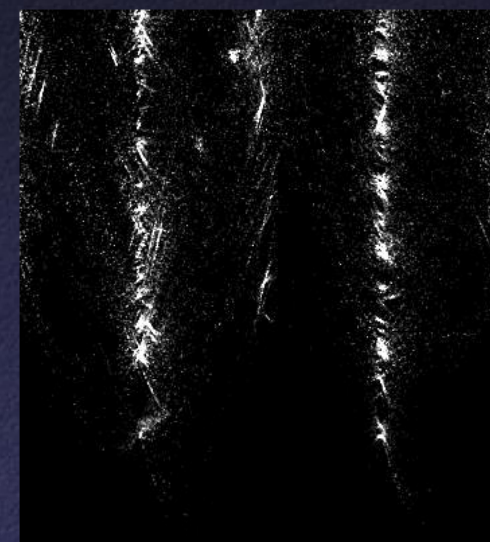
Combine

```
for each pixel p in the input image
     output_image(p) = max  ( E(p)^α · H(L)^β · (N(p)·N(L))^γ )
                        L
end
```

Assignment #1 Example

Input

Output

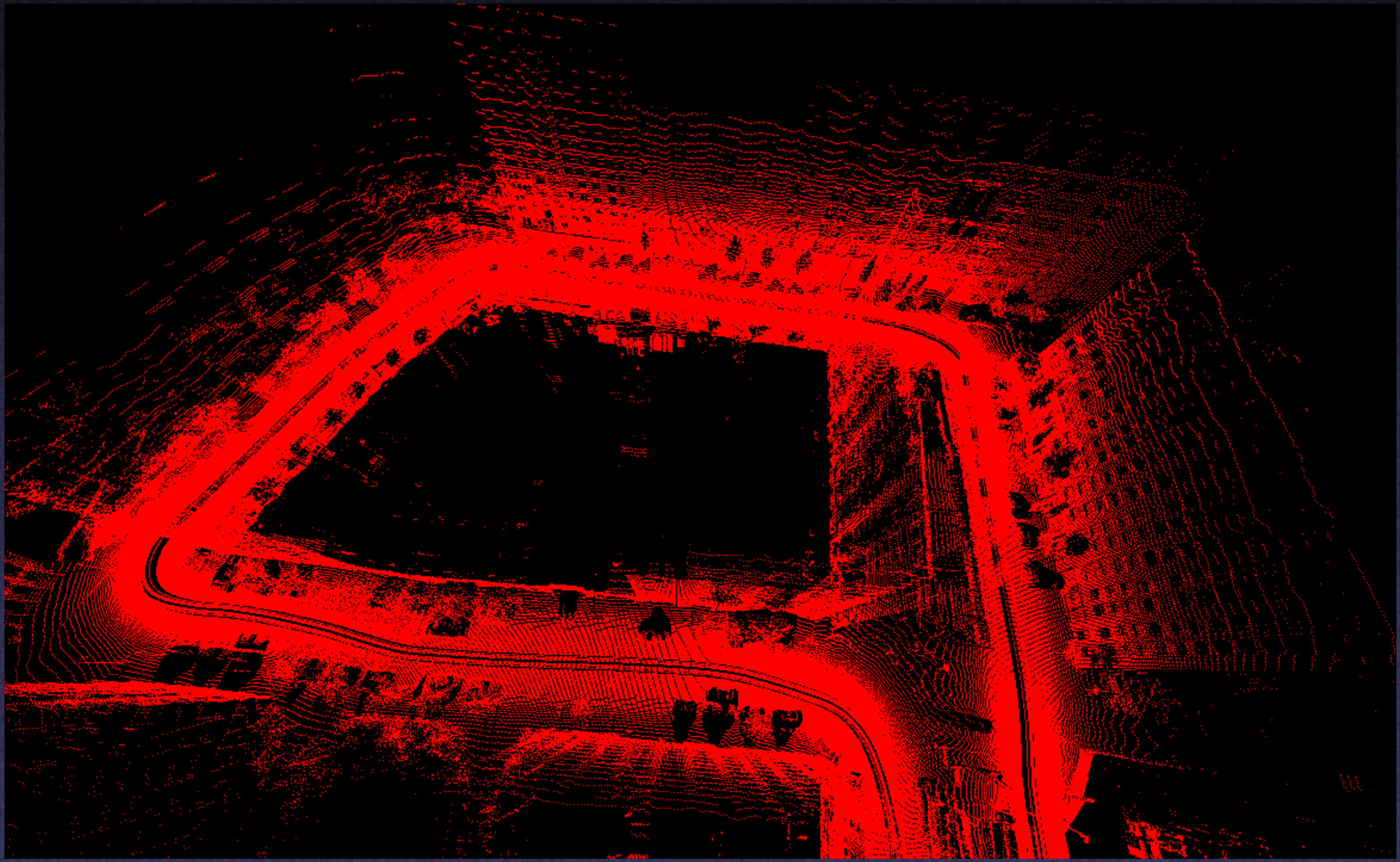


Hough Transform

Canny edges

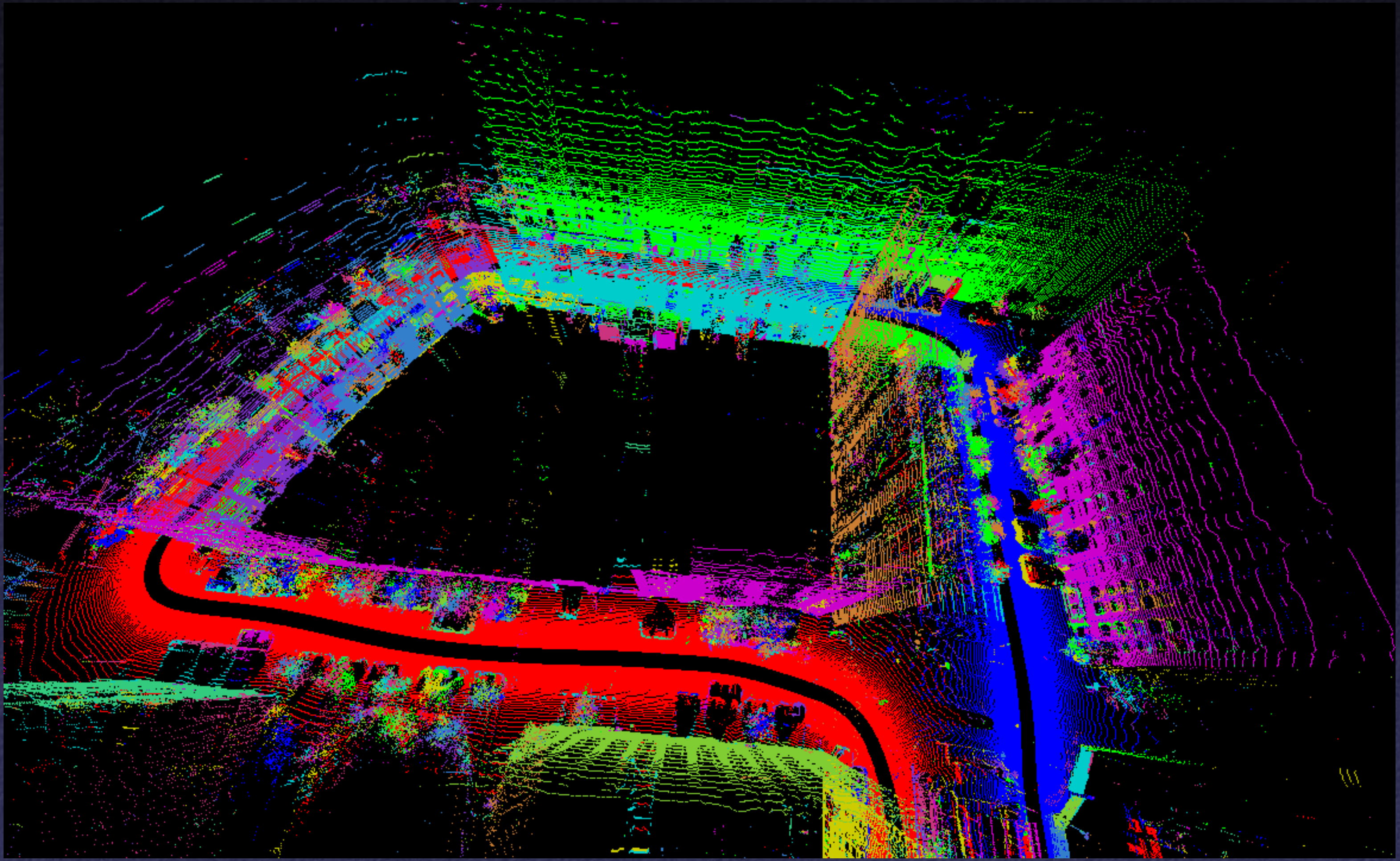
Strong Hough lines

Application: Plane Detection in Lidar Data



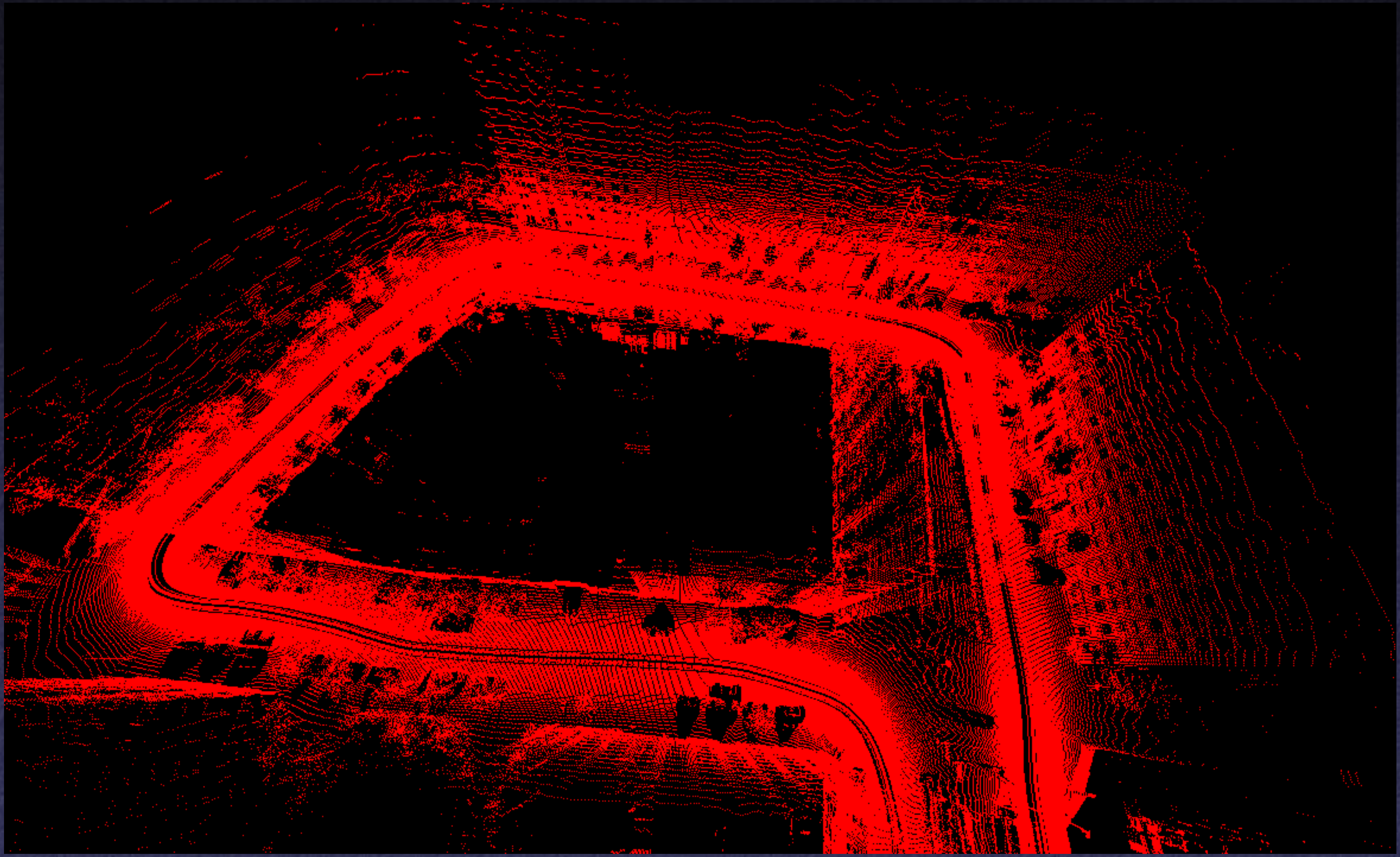
Lidar Scan of City Block

Application: Plane Detection in Lidar Data



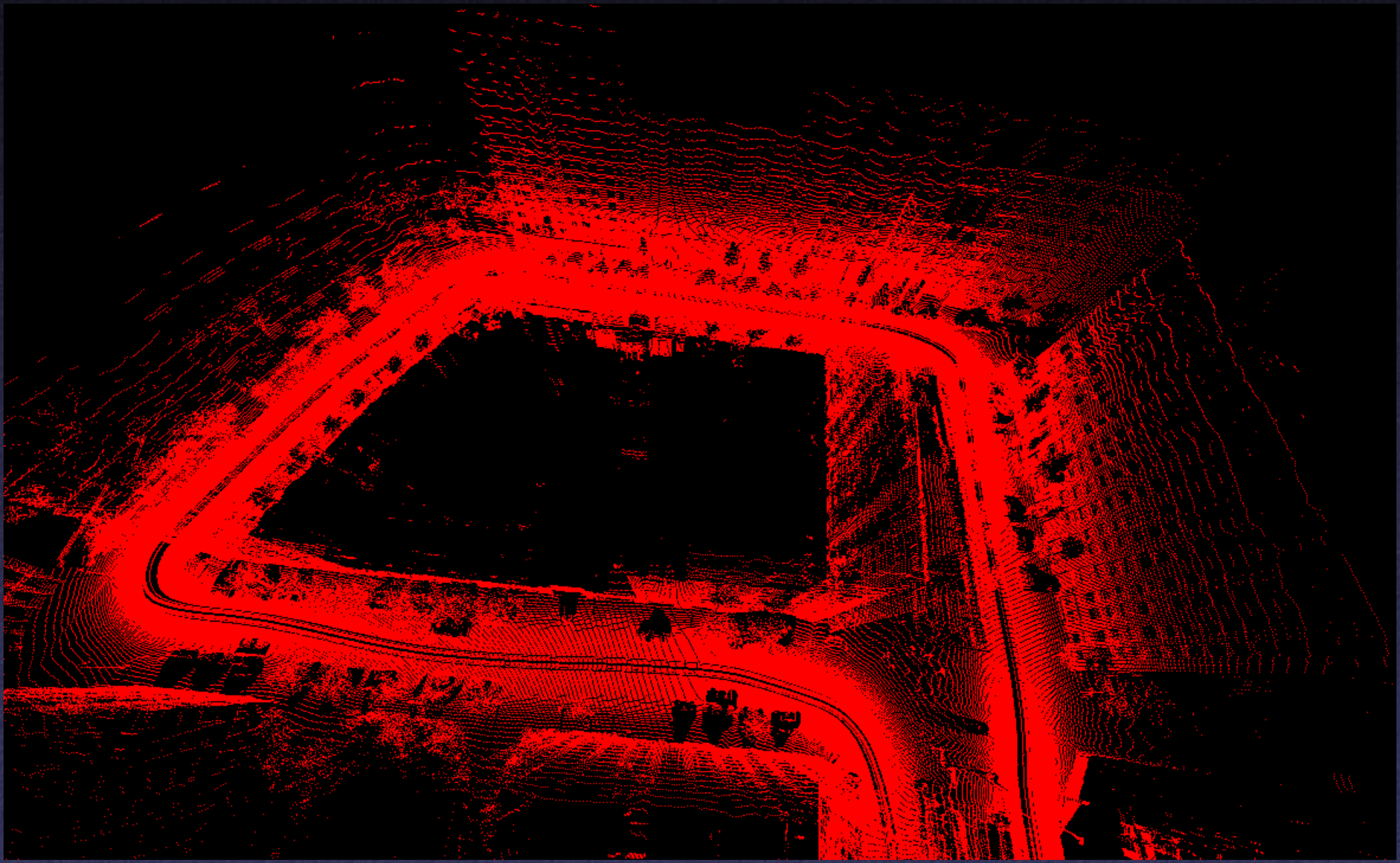
Lidar Scan of City Block after Plane Detection

Application: Plane Detection in Lidar Data



Before Enforcing Planarity

Application: Plane Detection in Lidar Data



After Enforcing Planarity

Summary

Problem:

- Structure detection

Focus: line detection

- RANSAC
- Hough transform

Extensions

- Circles and other primitives
- Symmetries

Applications

- Segmentation
- Alignment