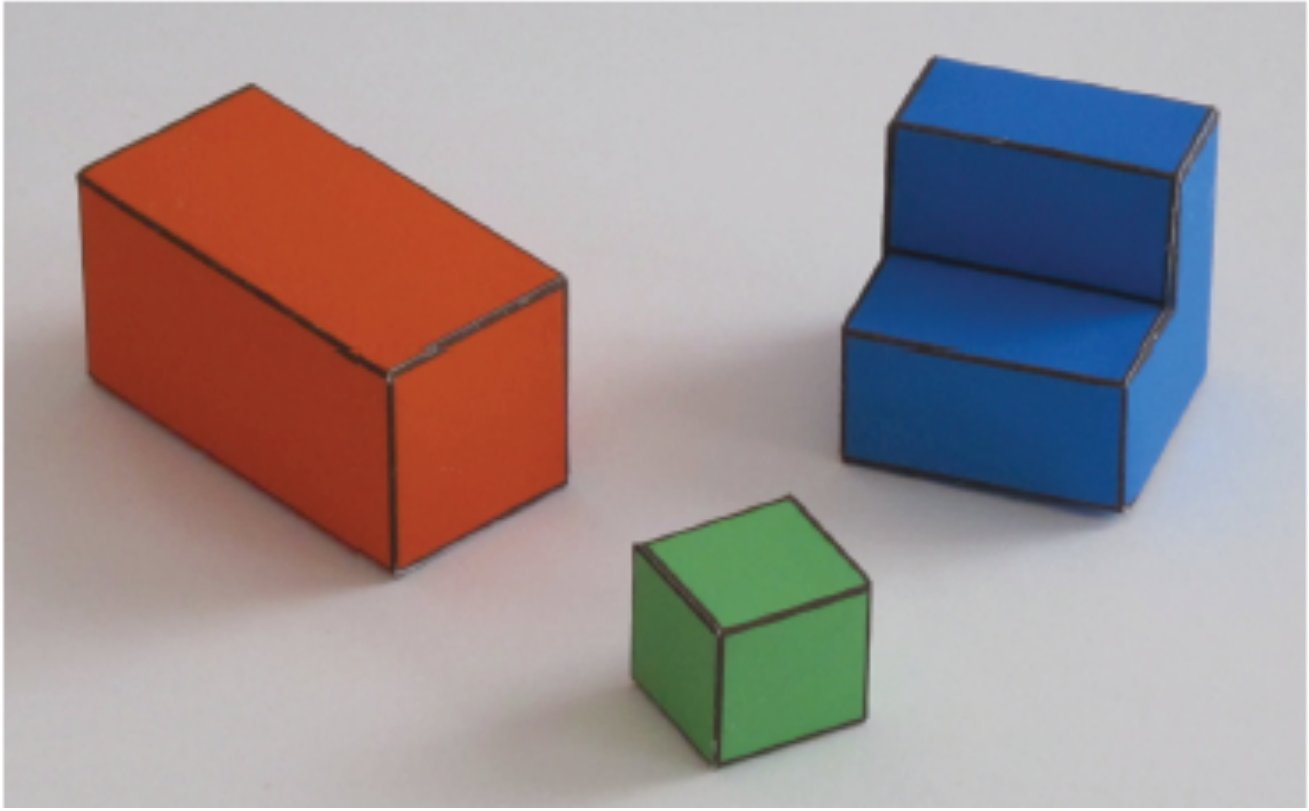
Let's make the world simpler



Last Last Time: Image filtering

$$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$f[.,.]$

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

$h[.,.]$

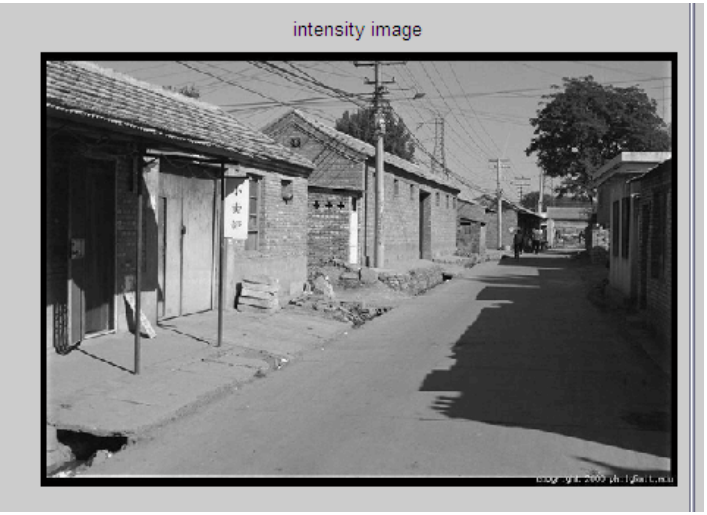
	0	10	20						

$$h[m,n]=\sum_{k,l} f[k,l]\, g[m+k,n+l]$$

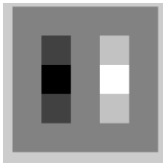
Credit: S. Seitz

Filtering in spatial domain

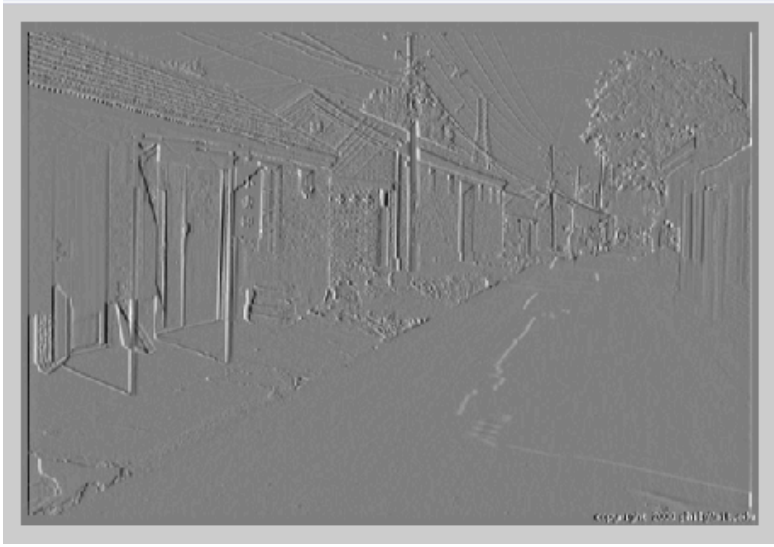
1	0	-1
2	0	-2
1	0	-1



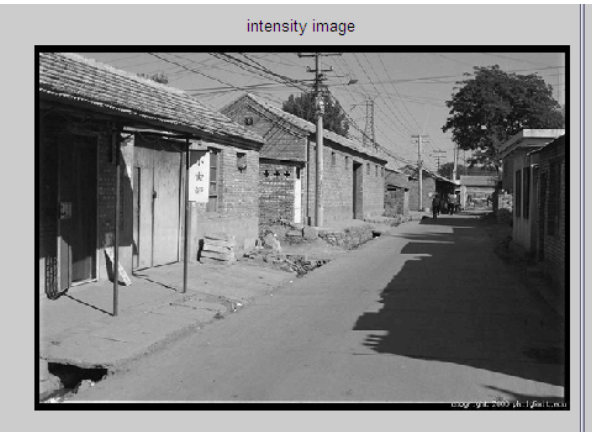
*



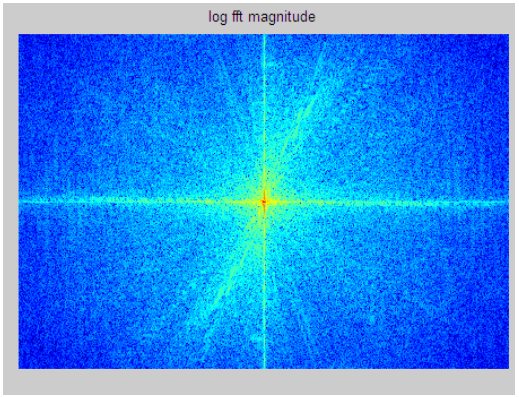
=

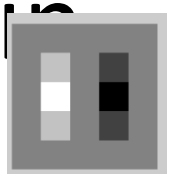


Filtering in frequency domain

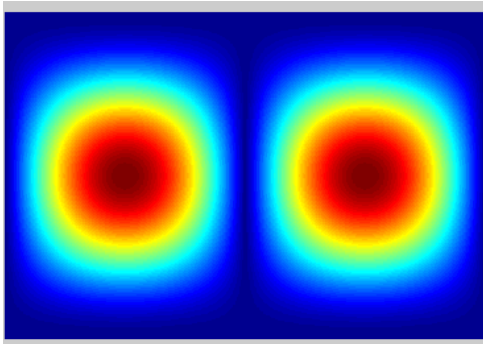


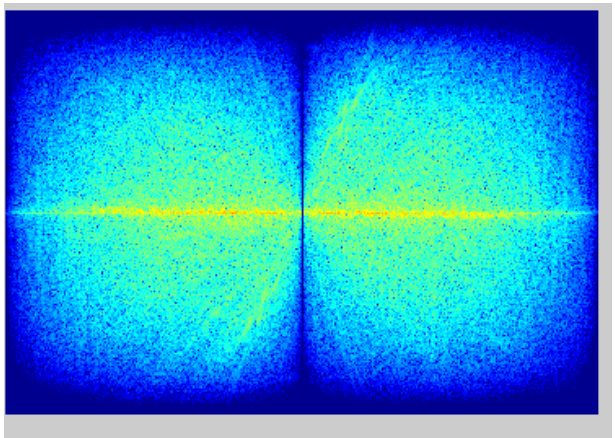
FFT



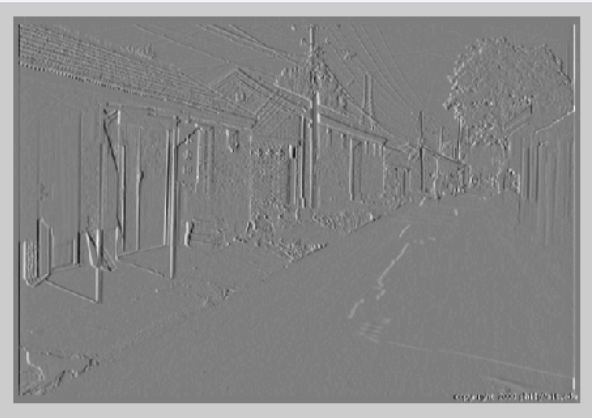


FFT





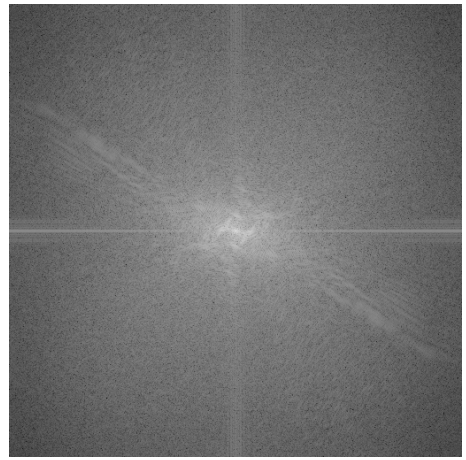
Inverse FFT

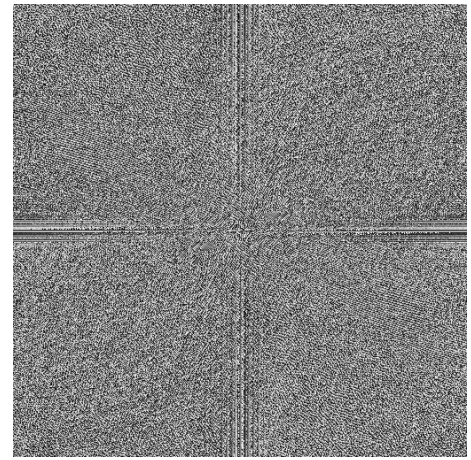


Phase and Magnitude

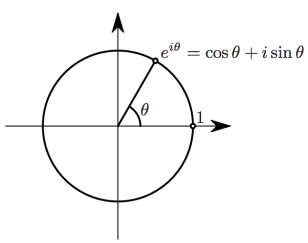
all natural images have about the same magnitude transform

hence, phase seems to matter, but magnitude largely doesn't

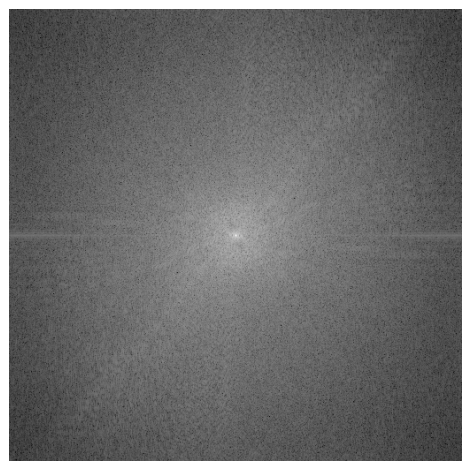


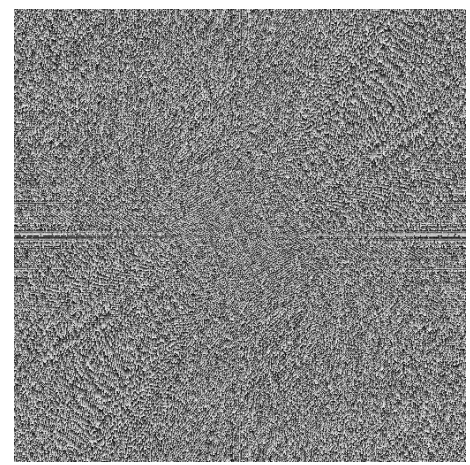


$$A\sin(\omega x + \phi)$$



Euler's definition of $e^{i\theta}$





```
    F(u,v) = SUM{ f(x,y)*exp(-j*2*pi*(u*x+v*y)/N) }
and
    f(x,y) = SUM{ F(u,v)*exp(+j*2*pi*(u*x+v*y)/N) }

where u = 0,1,2,...,N-1 and v = 0,1,2,...,N-1
      x = 0,1,2,...,N-1 and y = 0,1,2,...,N-1
      j = SQRT( -1 )
      and SUM means double summation  over proper
      x,y or u,v ranges
```

Magnitude

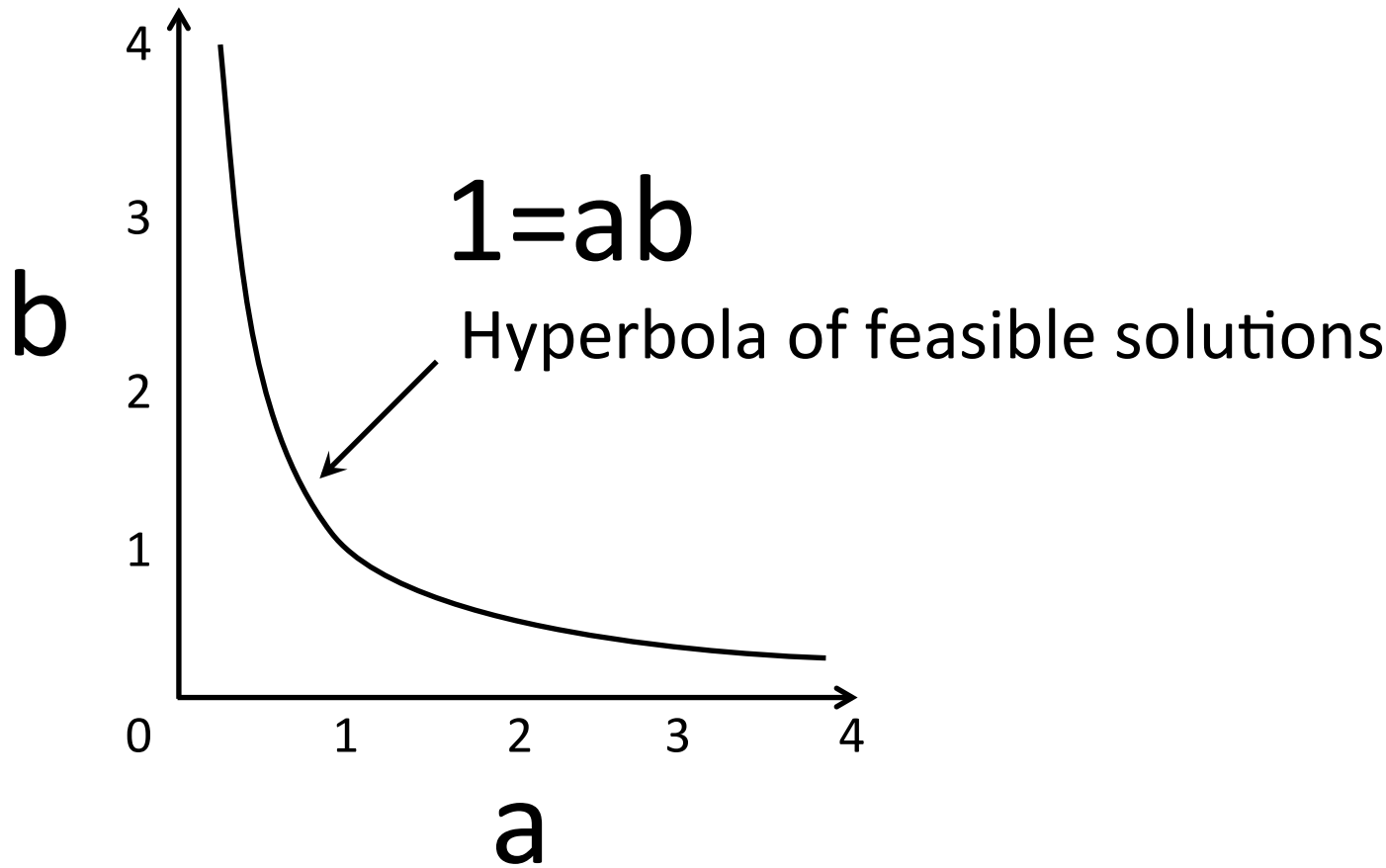
$$A = \pm\sqrt{R(u,v)^2 + I(u,v)^2}$$

`log(abs(fftshift(im_fft)))`

Phase

$$\phi = \tan^{-1}\frac{I(u,v)}{R(u,v)}$$

Last Time: Inverse Ill Pose Problem

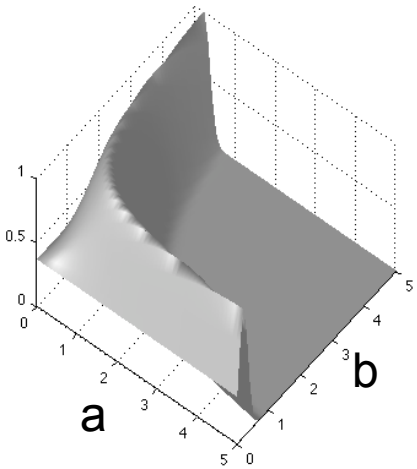


Last Time: Inverse Ill Pose Problem

Use P(a, b | y = 1) = k P(y=1|a, b) P(a, b)

Likelihood function

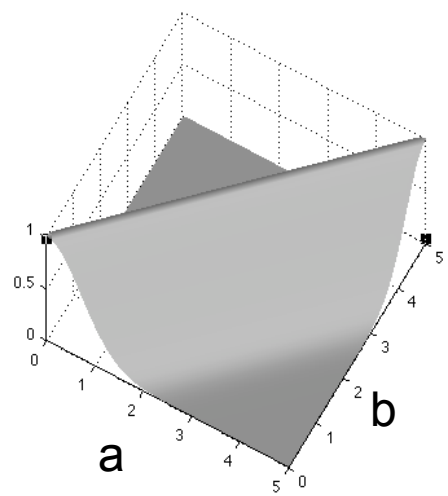
$$P(y=1 \mid a,b) = ke^{-\frac{(1-ab)^2}{2\sigma^2}}$$



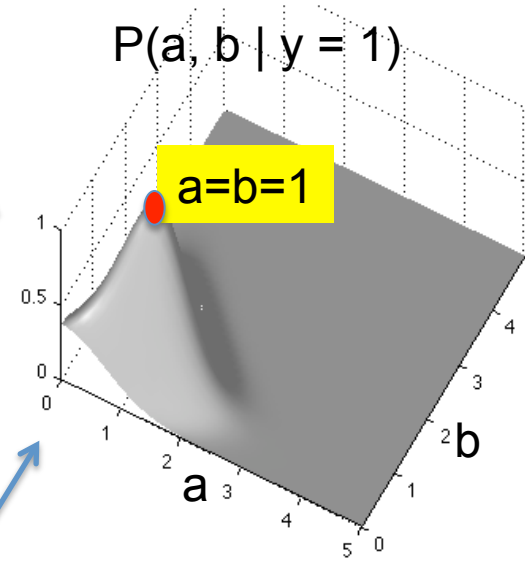
Prior probability

$$P(a,b) = ke^{-\frac{(a-b)^2}{2\sigma^2}} \quad \text{if } a>0,\ b>0$$

$$= 0 \quad \text{otherwise}$$



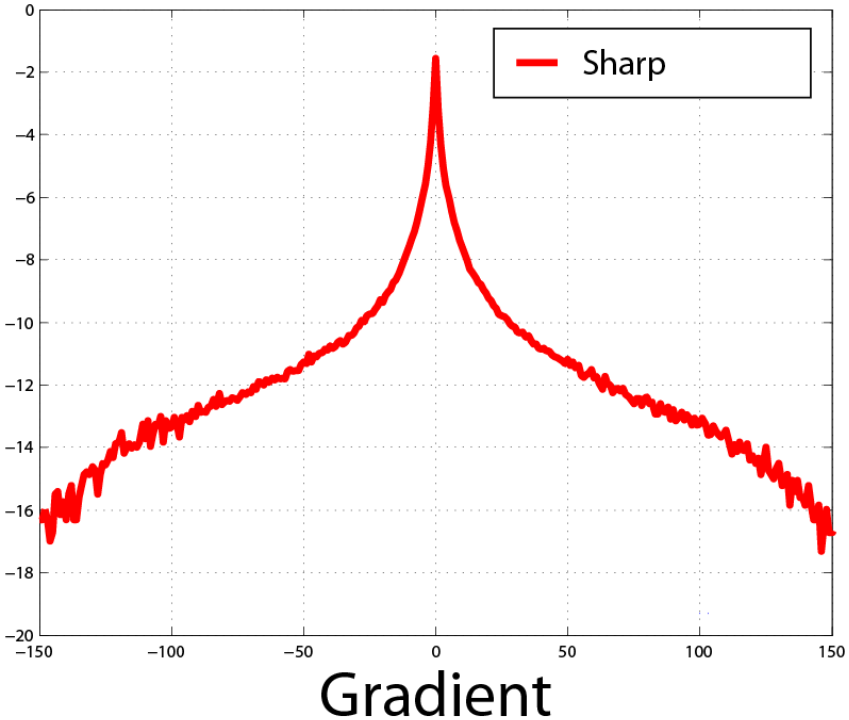
P(a, b | y = 1)



Natural image statistics

Characteristic distribution with heavy tails

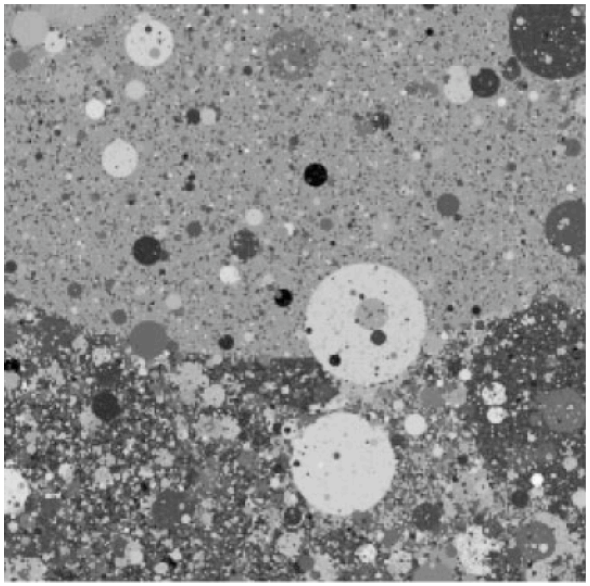
Histogram of image gradients

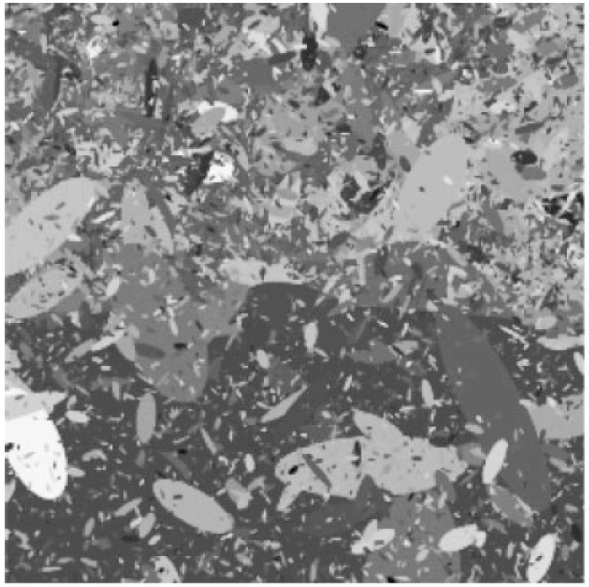


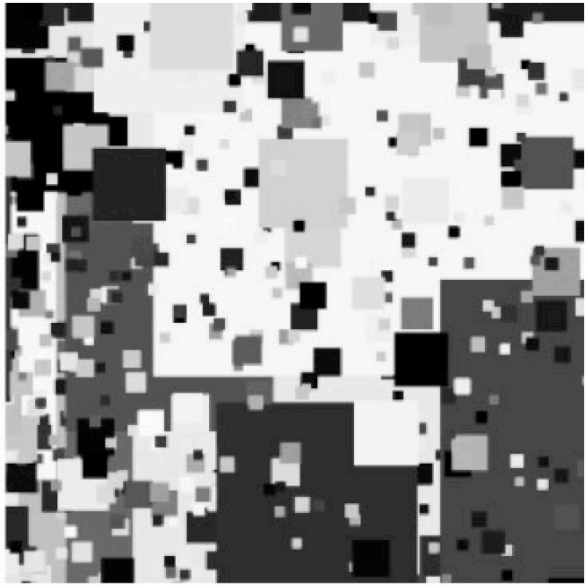
FIELD, D. 1994. What is the goal of sensory coding? Neural Computation 6, 559–601.

Dead leaves models

Introduced in the 60's by Matheron (67) and popularized by Ruderman (97)







From *Lee, Mumford and Huang 2001*

Statistics of Images

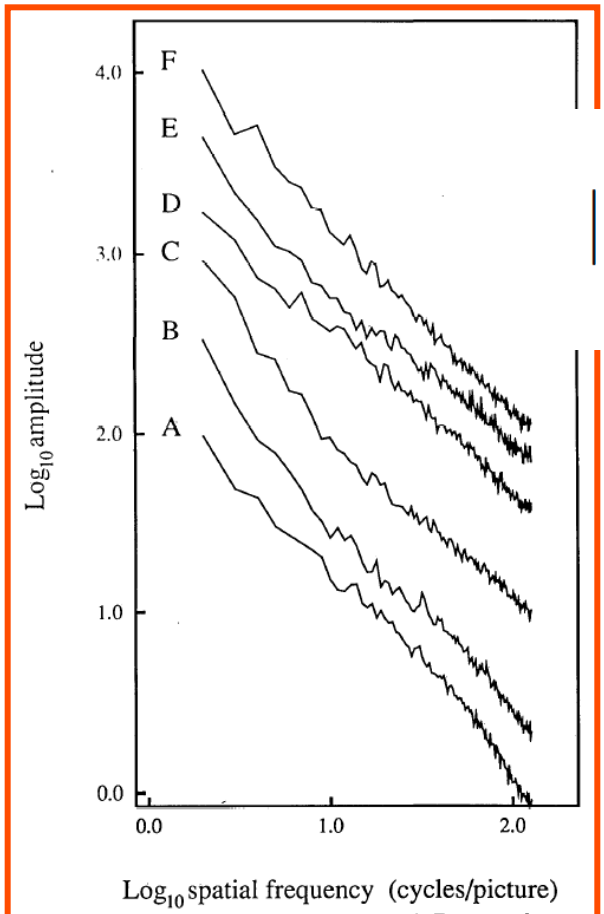
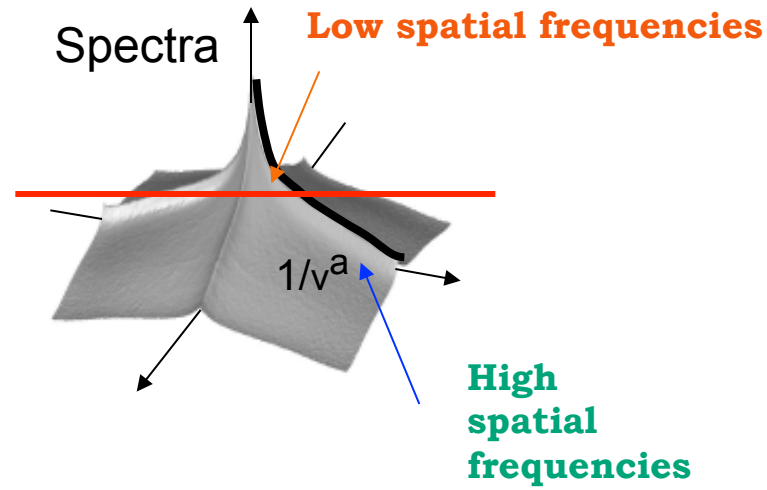
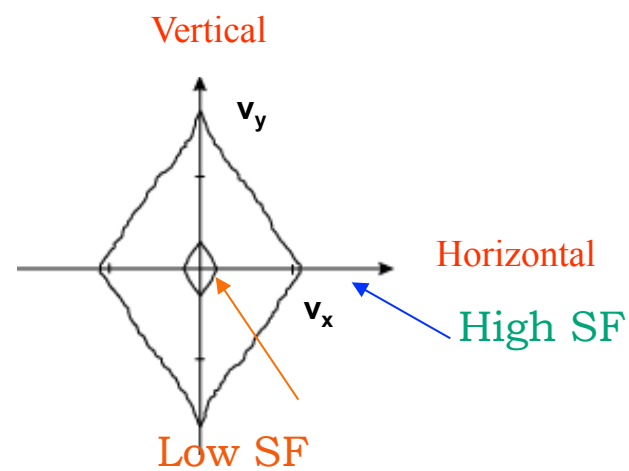


Fig. 8. Amplitude spectra for the six images A–F, averaged across all orientations. The spectra have been shifted up for clarity. On these log–log coordinates the spectra fall off by a factor of roughly $1/f$ (a slope of -1). Therefore the power spectra fall off as $1/f^2$.

Power spectra fall off as

$$|\hat{\mathbf{I}}(v)| \simeq \frac{1}{|v|^{\alpha}}$$





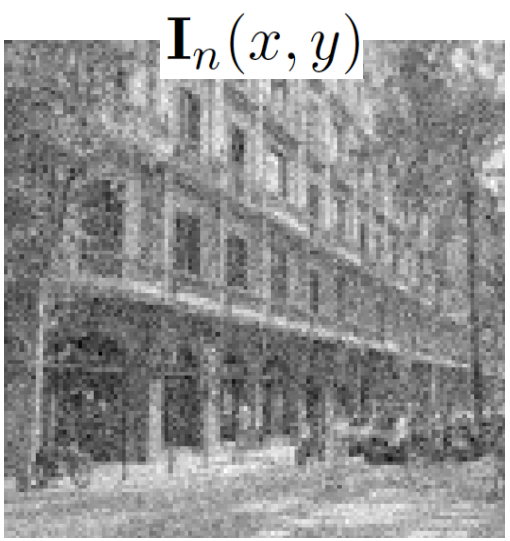
D. J. Field, "Relations between the statistics of natural images and the response properties of cortical cells," J. Opt. Soc. Am. A **4**, 2379- (1987)

Natural Image Statistics

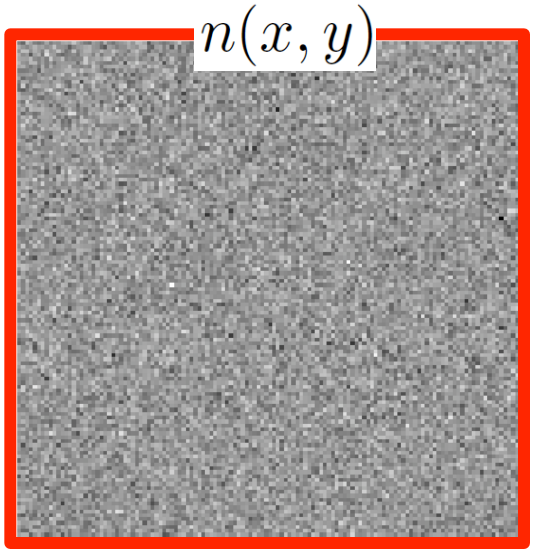
- Motivation: Separating into components
- Deal with ill pose: Bayesian Approach!
- Application 1: Denoising
- Application 2: Intrinsic Image
- Application 3: Reflection Separation
- Application 4: Deblurring
- Application 5: Digital Forensics

Denoising

Decomposition of a noisy image



=



+

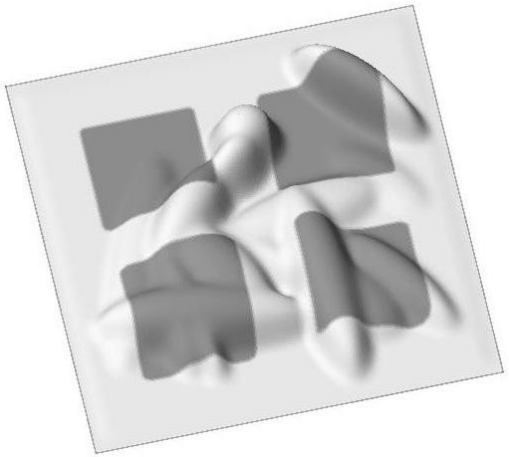
White Gaussian noise: $N(0, \sigma_n^2)$ Natural image

Find I(x,y) that maximizes the posterior (maximum a posteriory, MAP):

$$\max_{\mathbf{I}} p(\mathbf{I}|\mathbf{I}_n) = \max_{\mathbf{I}} \underbrace{\boxed{p(\mathbf{I}_n|\mathbf{I})}}_{\text{likelihood}} \times \underbrace{\boxed{p(\mathbf{I})}}_{\text{prior}}$$

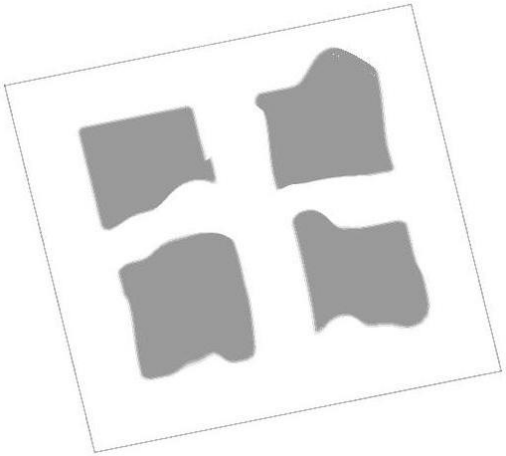
$$= \max_{\mathbf{I}} \boxed{\exp(-|\mathbf{I}_n - \mathbf{I}|^2/\sigma_n^2)} \times \boxed{\exp\left(-\frac{1}{2}\mathbf{I}^T\mathbf{C}^{-1}\mathbf{I}\right)}$$

Goal: Intrinsic images



Image

Shading Image



Reflectance Image

Slide: Marshal Tappen

Retinex

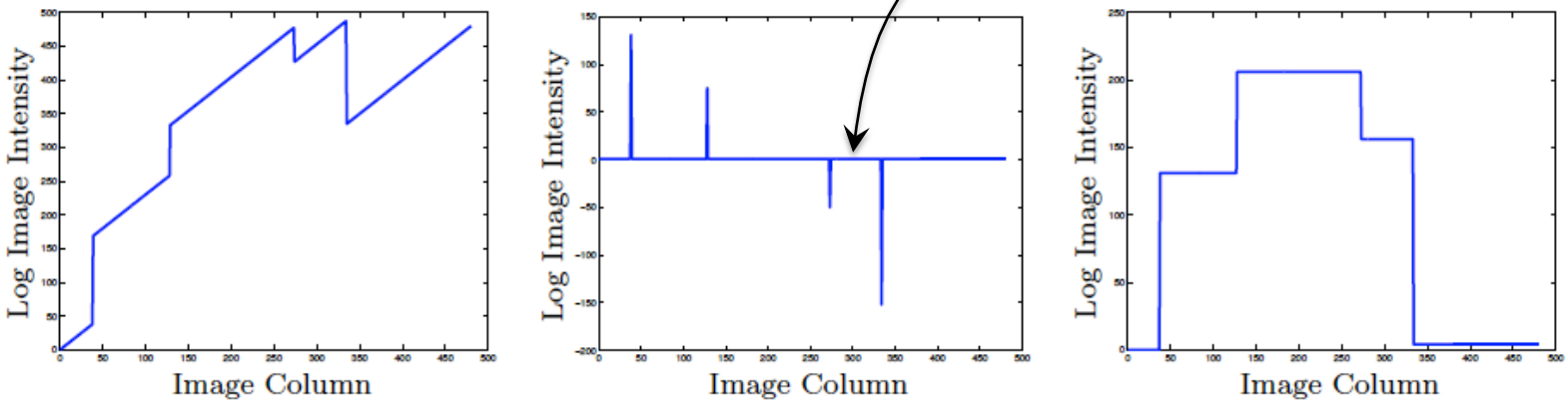
Again, we are trying to solve an ill-posed problem:

24 = ? x ?

From M. Tappen, PhD

Retinex

too small? let it be zero!



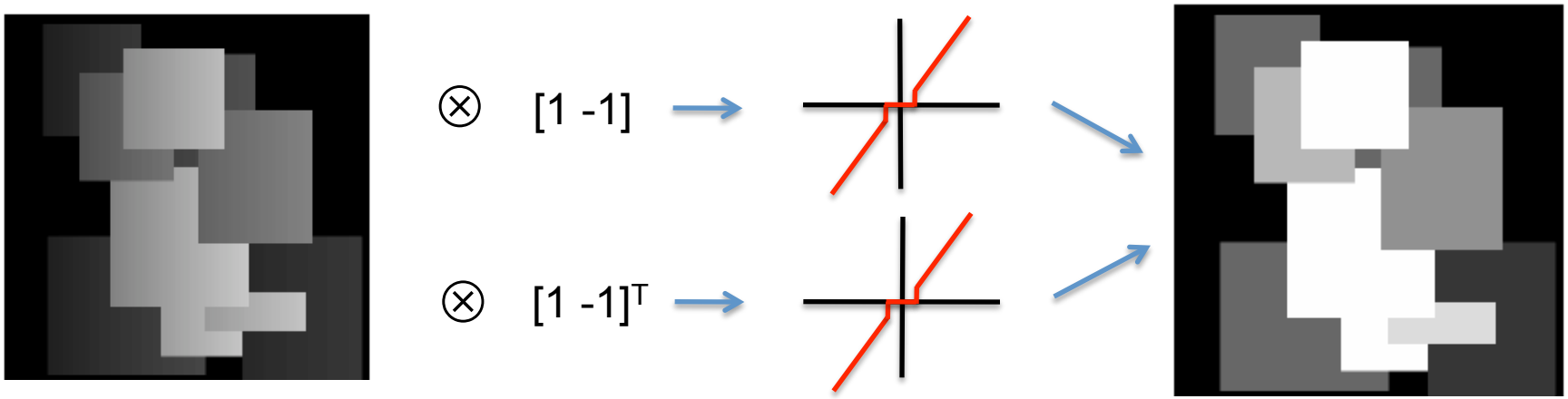
(a) One column from the observed image.

(b) The derivative of the plot from (a).

(c) The estimate of the log shading

From M. Tappen, PhD

log



From M. Tappen, PhD

Separating reflections from a single image using local features

Anat Levin Assaf Zomet Yair Weiss

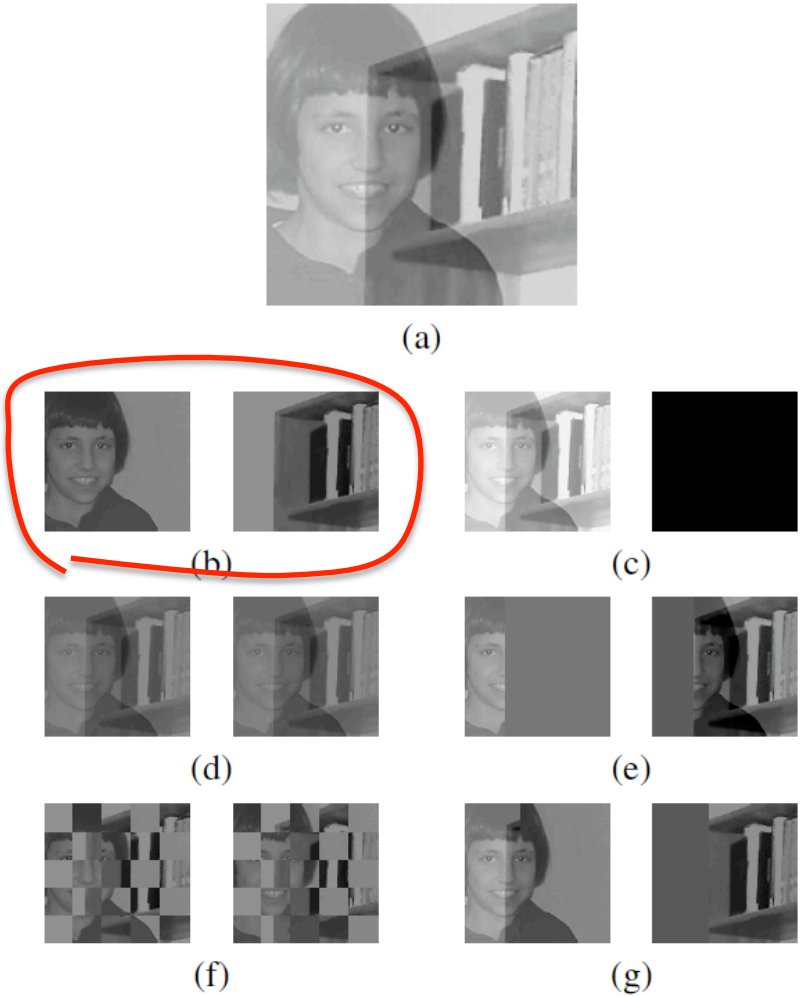


Figure 1: (a) Original input image (constructed by summing the two images in b). (b) the correct decomposition. (c)-(g) alternative possible decompositions. Why should the decomposition in (b) be favored?

very simple cost function: it favors decompositions which have a small number of edges and corners. Surprisingly, this simple cost function gives the "right" decompositions for challenging real images.[1]

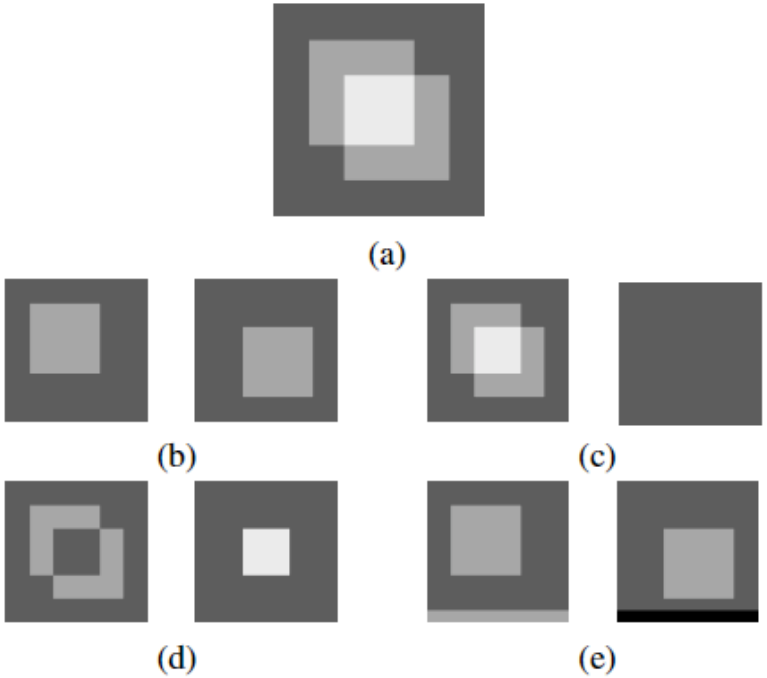


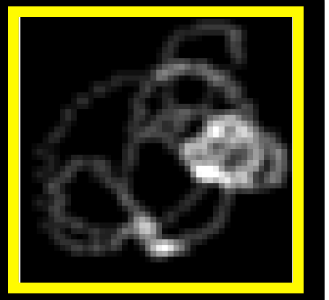
Figure 2: An input image and some decompositions

Three sources of information

1. Reconstruction constraint:

Estimated sharp image

$\otimes$

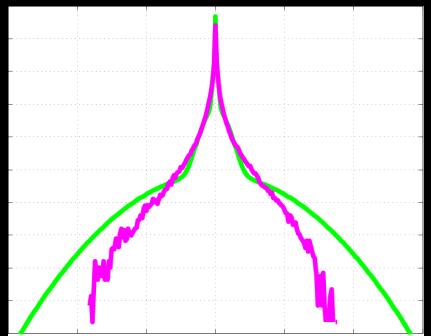


Estimated blur kernel

$=$

Input blurry image

2. Image prior:



Distribution of gradients

3. Blur prior:

Positive & Sparse

Image circulated on internet

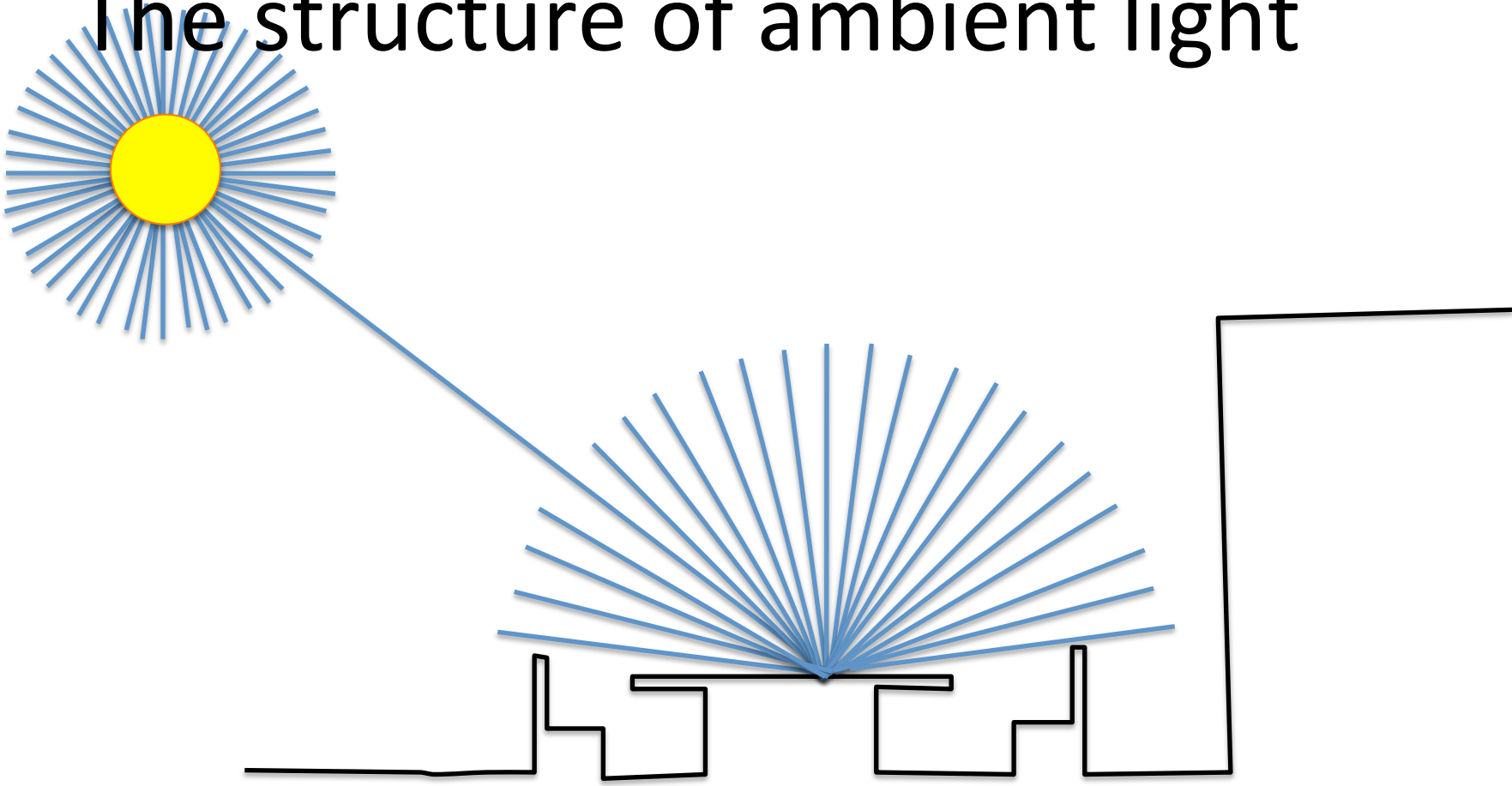
Fonda Speaks To Vietnam Veterans At Anti-War Rally

Actress And Anti-War Activist Jane Fonda Speaks to a crowd of Vietnam Veterans as Activist and former Vietnam Vet John Kerry (LEFT) listens and prepares to speak next concerning the war in Vietnam (AP Photo)

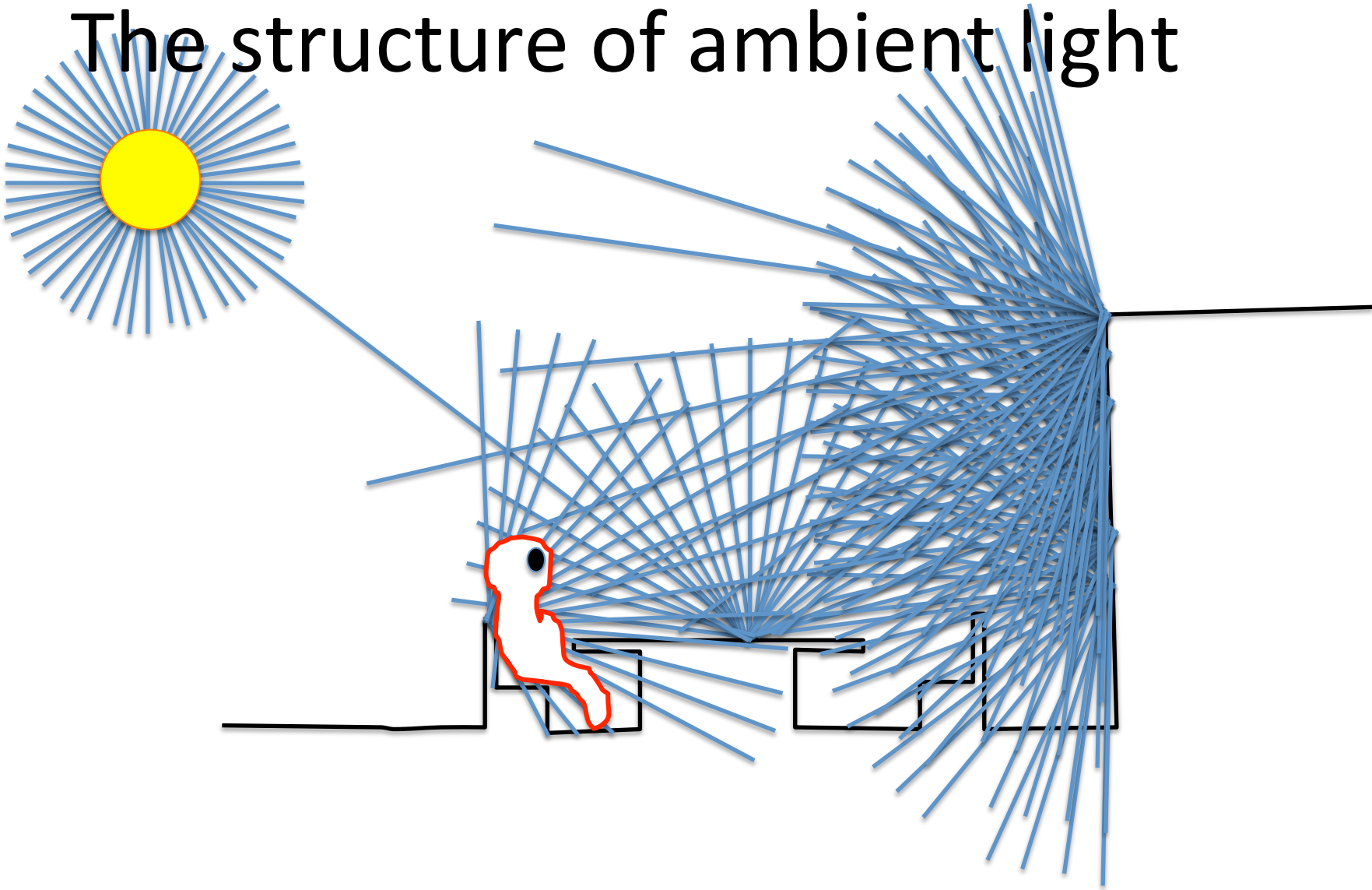
http://www.cs.dartmouth.edu/farid/publications/deception09.pdf
http://www.cs.dartmouth.edu/farid/publications/significance06.pdf

Vision is hard!

The structure of ambient light

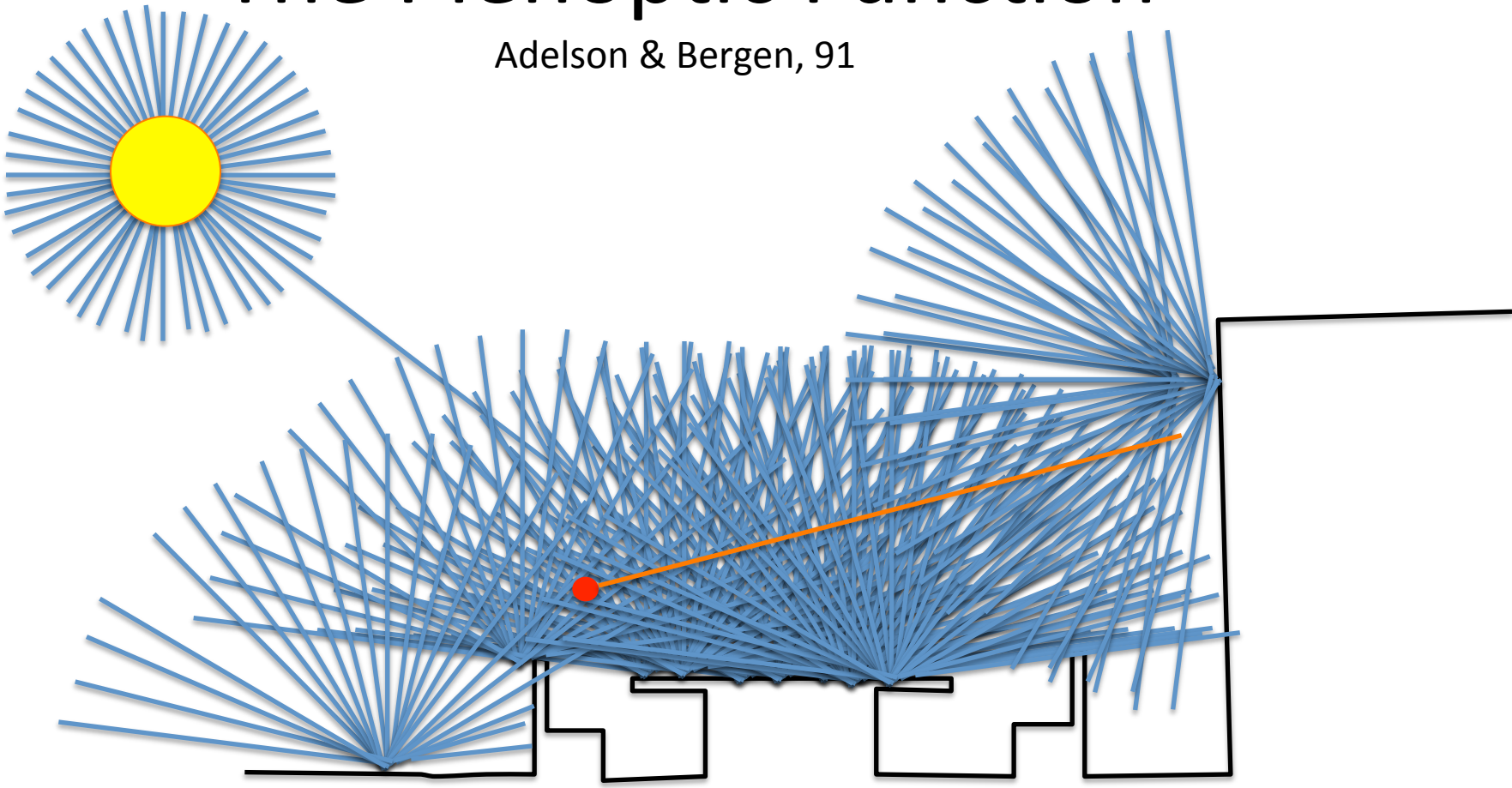


The structure of ambient light



The Plenoptic Function

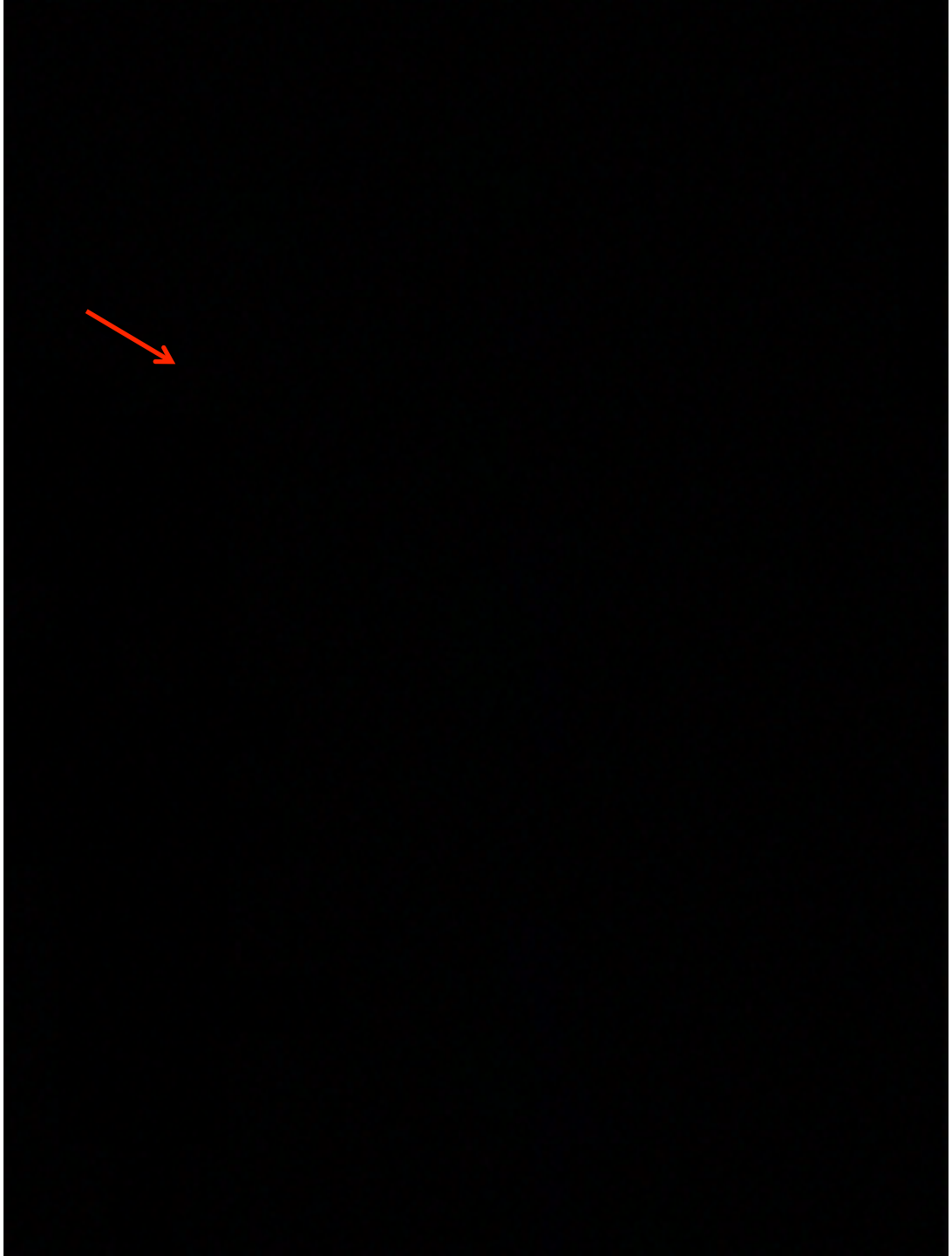
Adelson & Bergen, 91



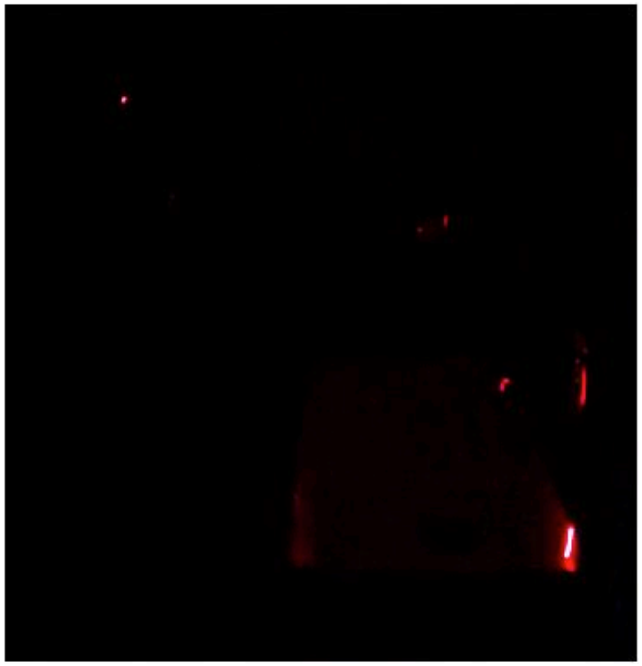
The intensity P can be parameterized as:

$$P\ (\theta, \phi, \lambda, t, X, Y, Z)$$

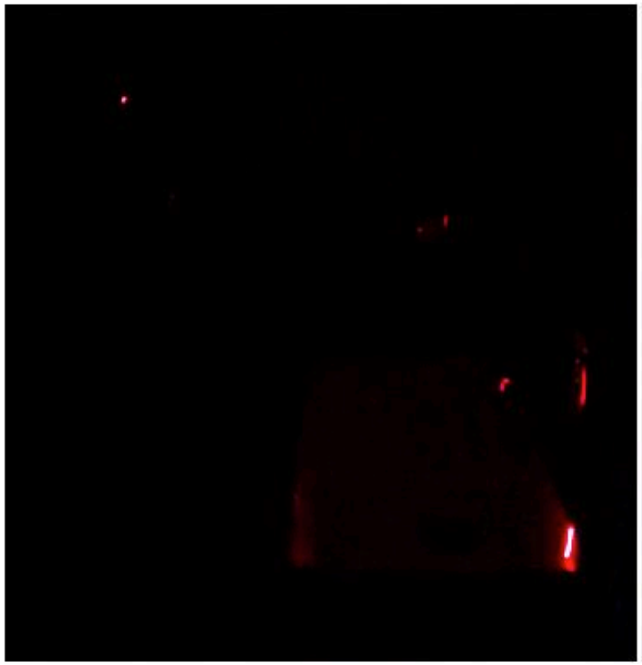
"The complete set of all convergence points constitutes the permanent possibilities of vision." Gibson



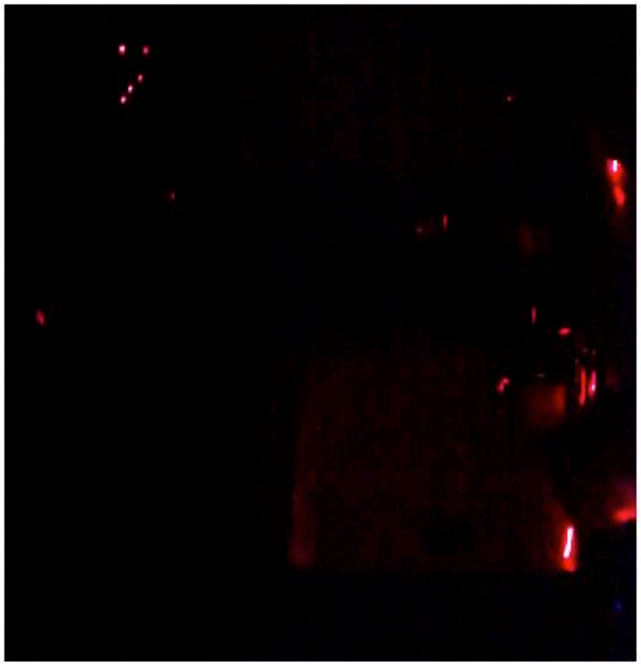
1



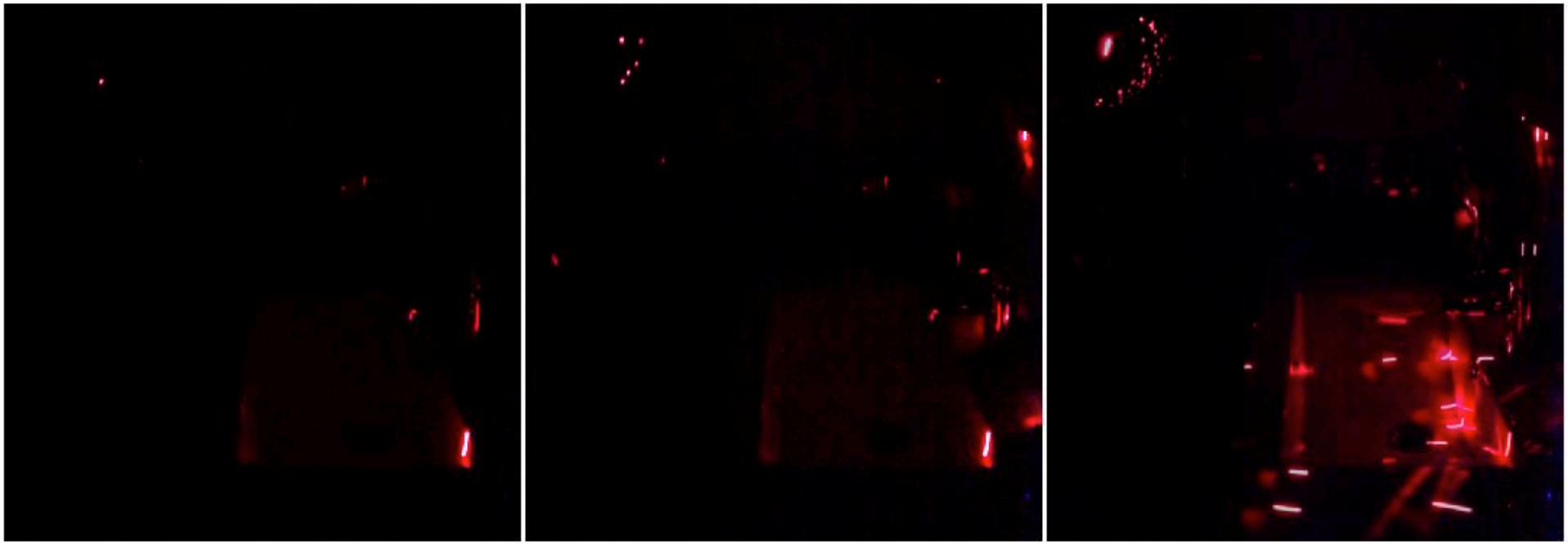
1



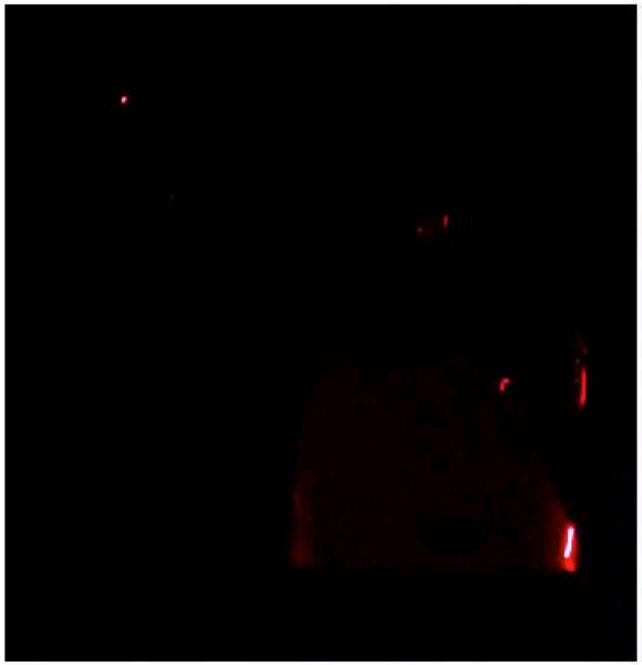
5



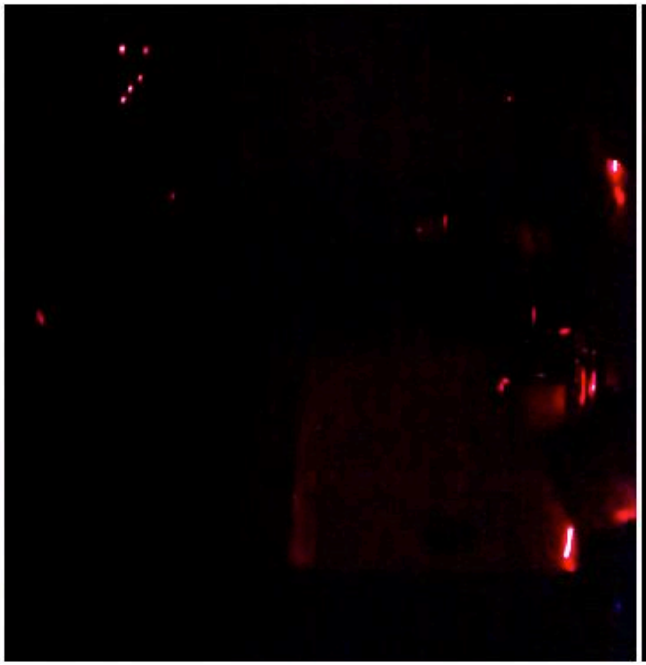
1 5 50



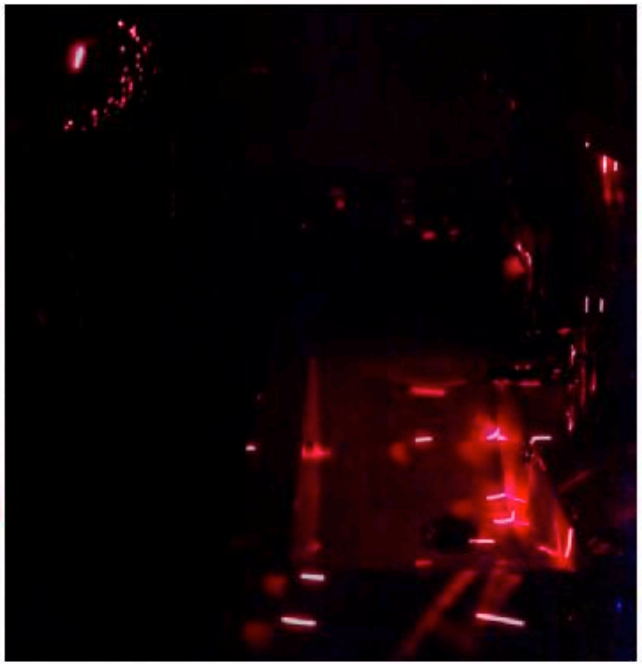
1

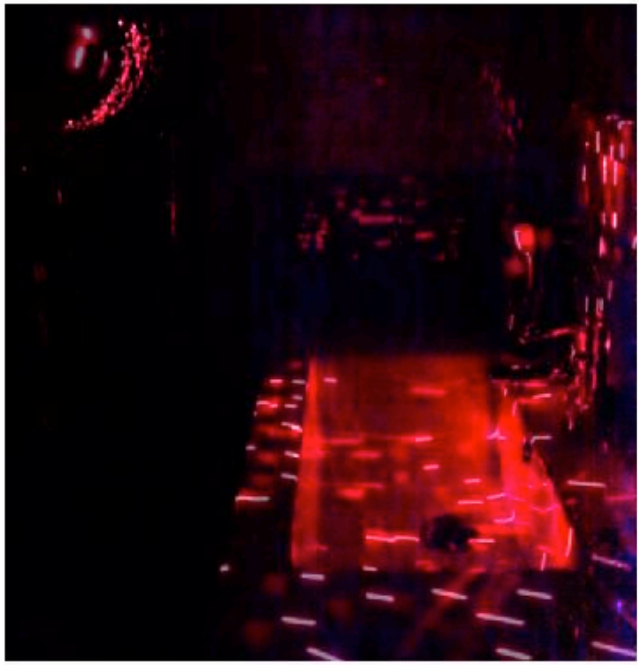


5

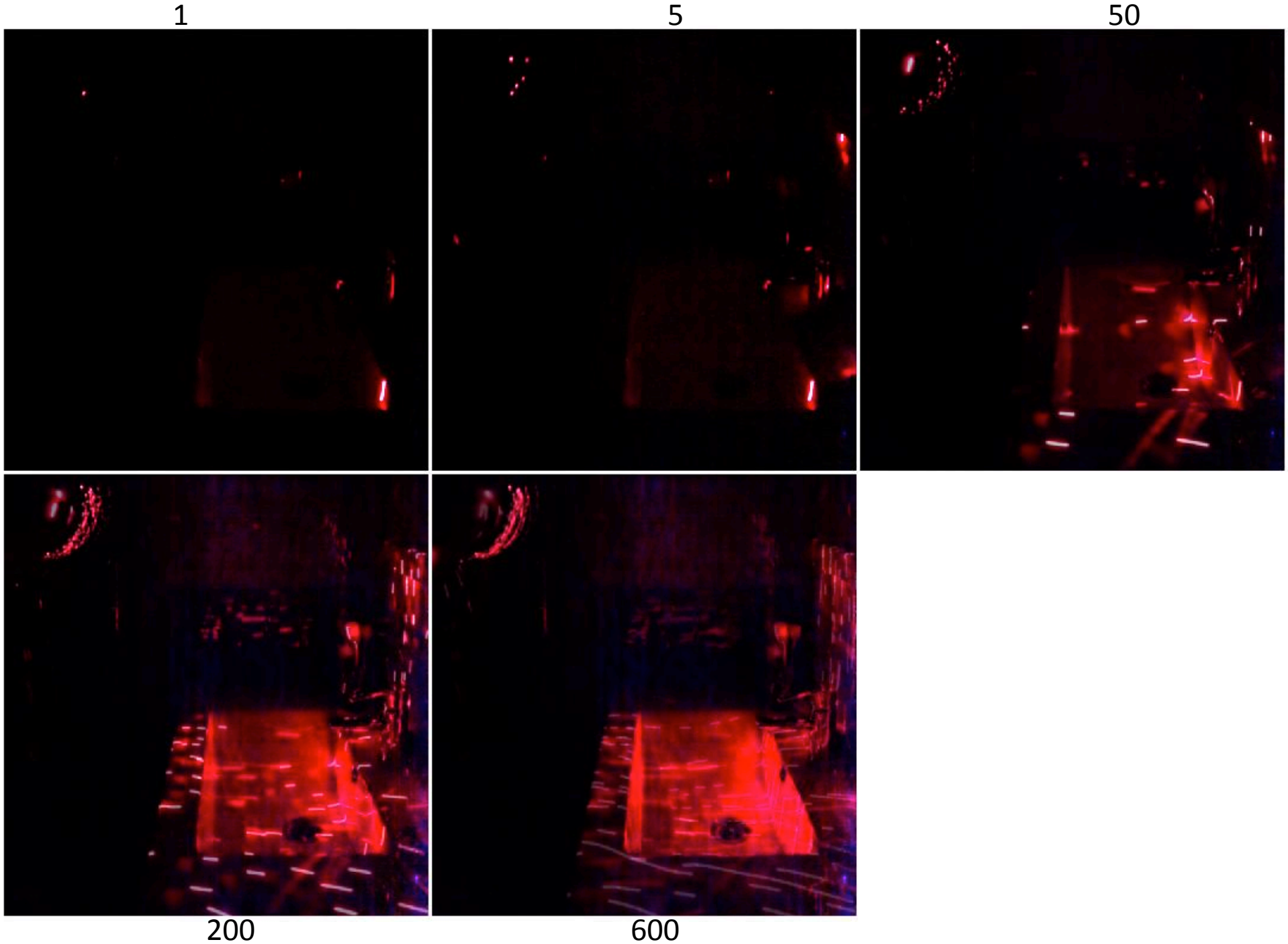


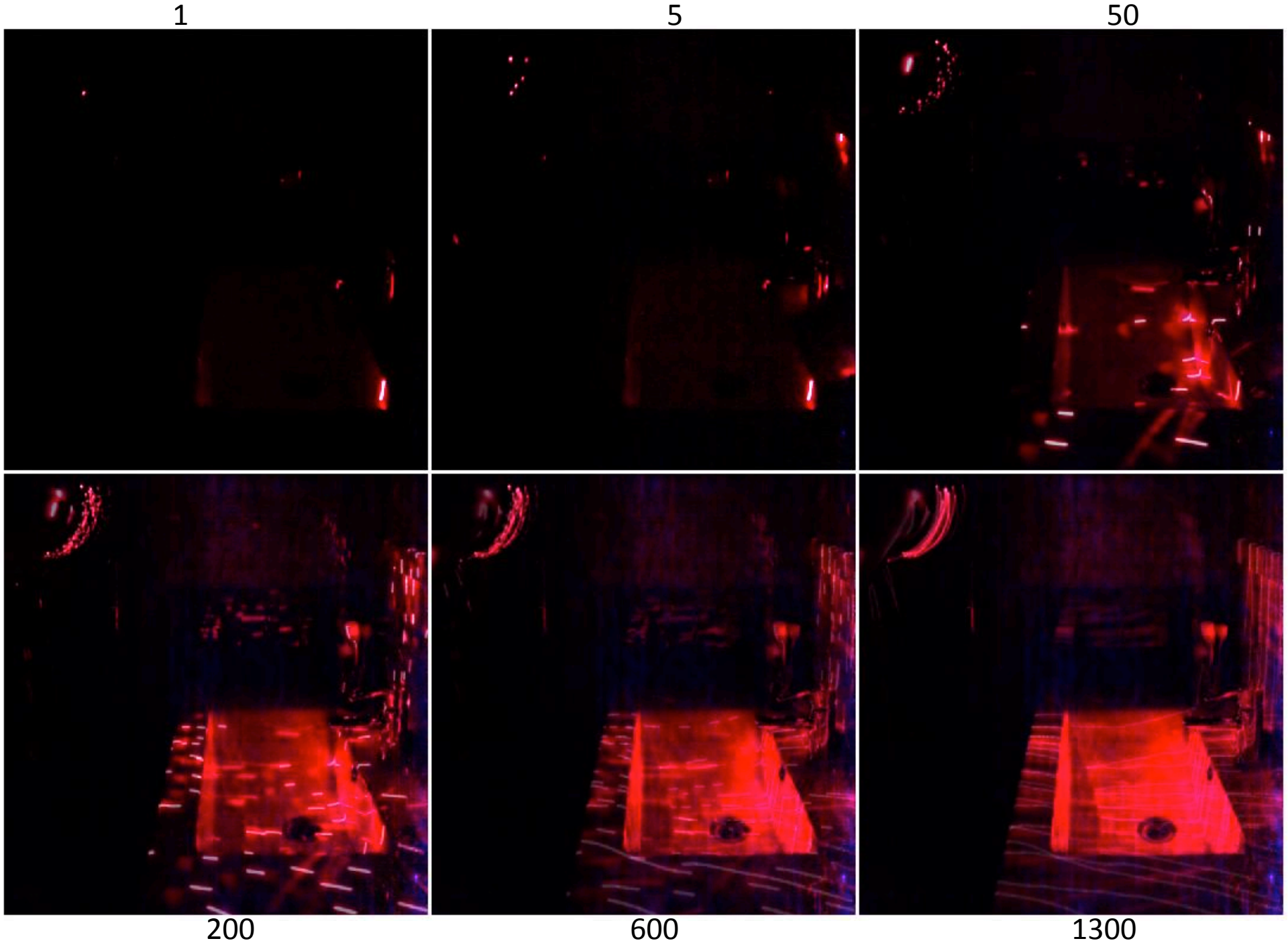
50





200


1
5
50
200
600


1
5
50
200
600
1300

Vision is hard
because the world is complicated!

- Better Vision Algorithm
- Or, just to make the world simpler!

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
PROJECT MAC

Artificial Intelligence Group July 7, 1966
Vision Memo. No. 100.

THE SUMMER VISION PROJECT

Seymour Papert

The summer vision project is an attempt to use our summer workers effectively in the construction of a significant part of a visual system. The particular task was chosen partly because it can be segmented into sub-problems which will allow individuals to work independently and yet participate in the construction of a system complex enough to be a real landmark in the development of "pattern recognition".

The “three weeks” vision project

The goal of PS2 is

to solve computer vision in 3 weeks

A Simple Visual System

- A simple world
- A simple image formation model
- A simple goal

Today

- A simple world
- A catalog of edges
- Generic view assumption
- Constrained solver
- Edge detection: Canny
- Line detection
  - RANSAC
  - Hough transformation

A Simple World

A Simple World

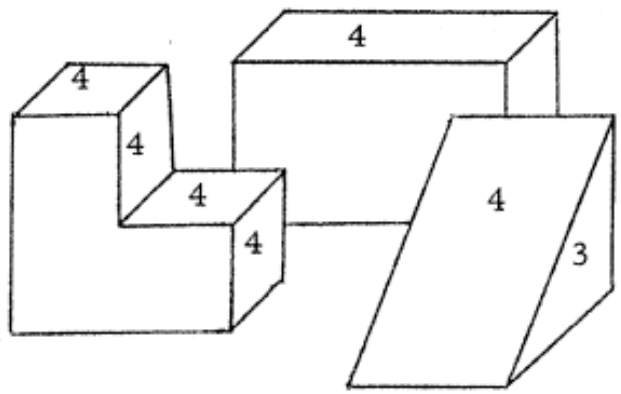
MACHINE PERCEPTION OF THREE-DIMENSIONAL SOLIDS

by

LAWRENCE GILMAN ROBERTS

Submitted to the Department of Electrical Engineering on May 10, 1963, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

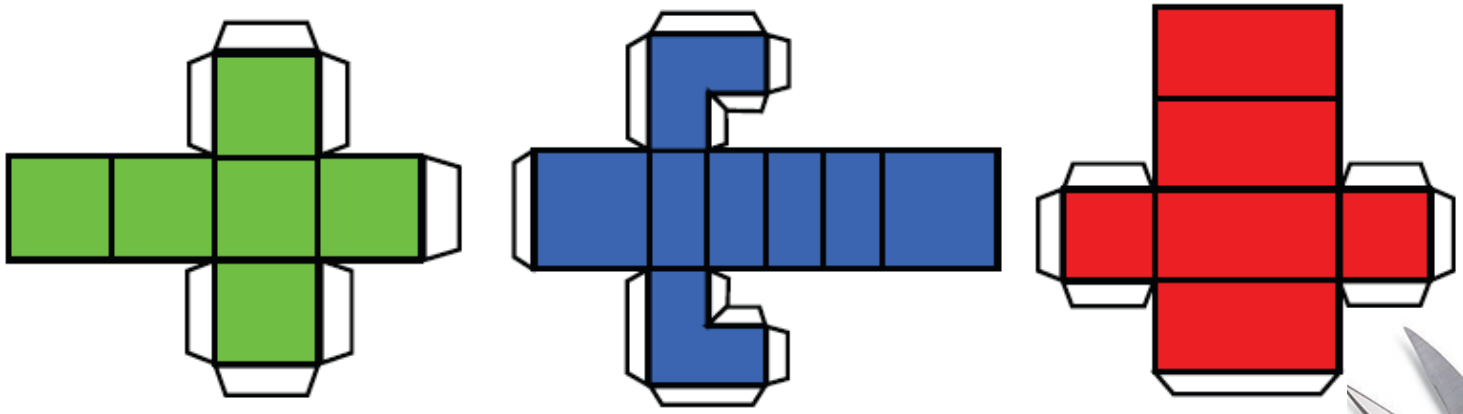
The problem of machine recognition of pictorial data has long been a challenging goal, but has seldom been attempted with anything more complex than alphabetic characters. Many people have felt that research on character recognition would be a first step, leading the way to a more general pattern recognition system. However, the multitudinous attempts at character recognition, including my own, have not led very far. The reason, I feel, is that the study of abstract, two-dimensional forms leads us away from, not toward, the techniques necessary for the recognition of three-dimensional objects. The per-

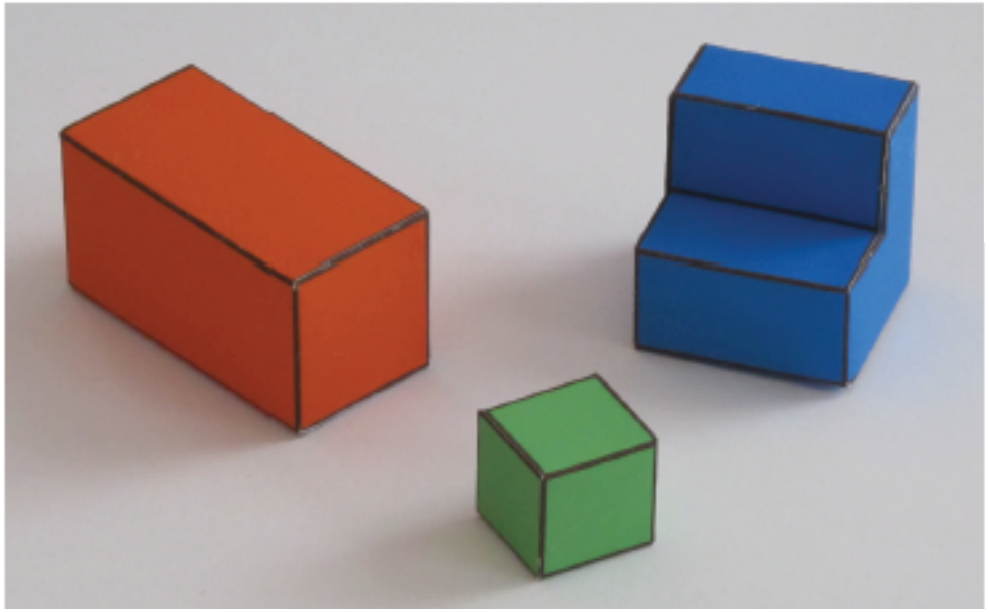


Complete Convex Polygons. The polygon selection procedure would select the numbered polygons as complete and convex. The number indicates the probable number of sides. A polygon is incomplete if one of its points is a collinear joint of another polygon.

http://www.packet.cc/files/mach-per-3D-solids.html

A Simple World

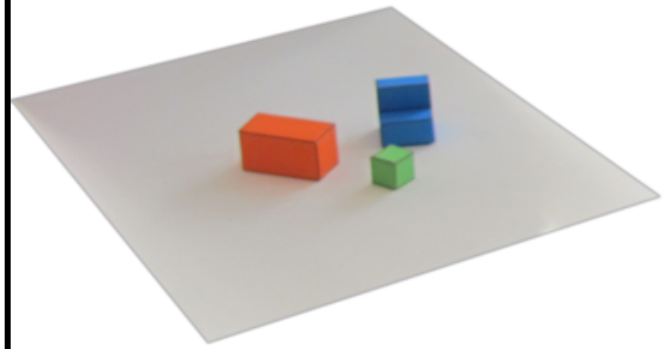




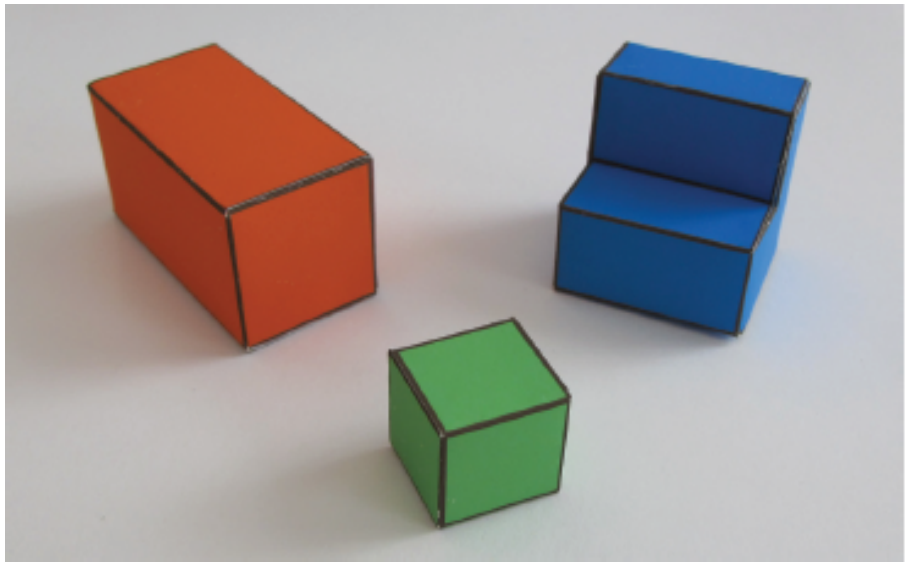
A simple image formation model

Simple world rules:

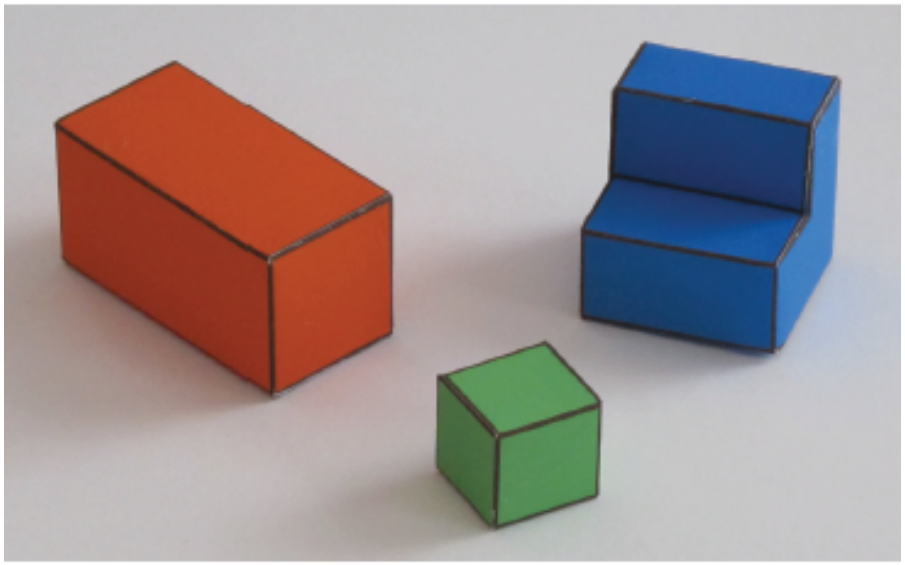
- Surfaces can be horizontal or vertical.
- Objects will be resting on a white horizontal ground plane



A simple image formation model

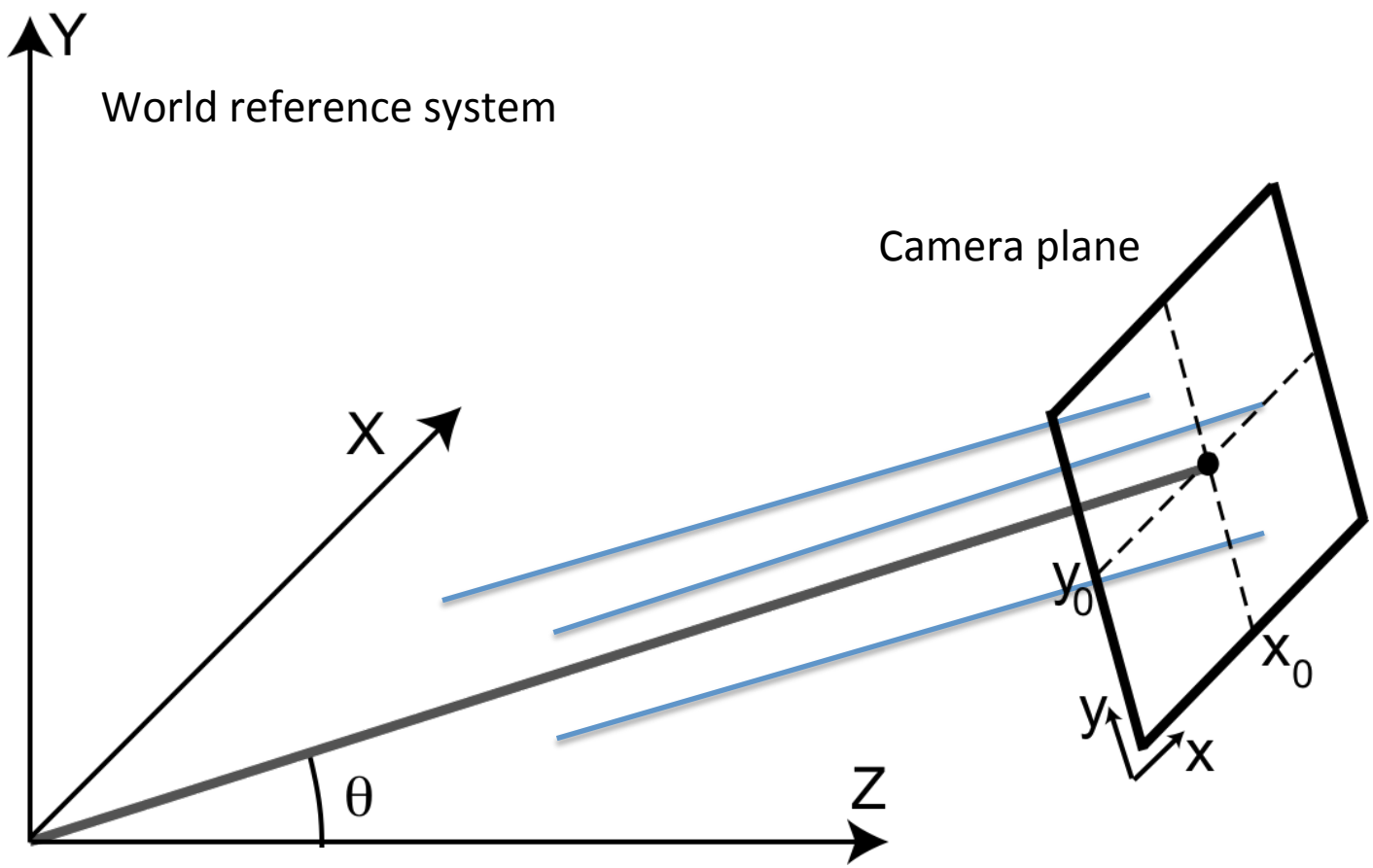


Perspective projection



Parallel (orthographic) projection

A simple image formation model



(right-handed reference system)

A simple image formation model

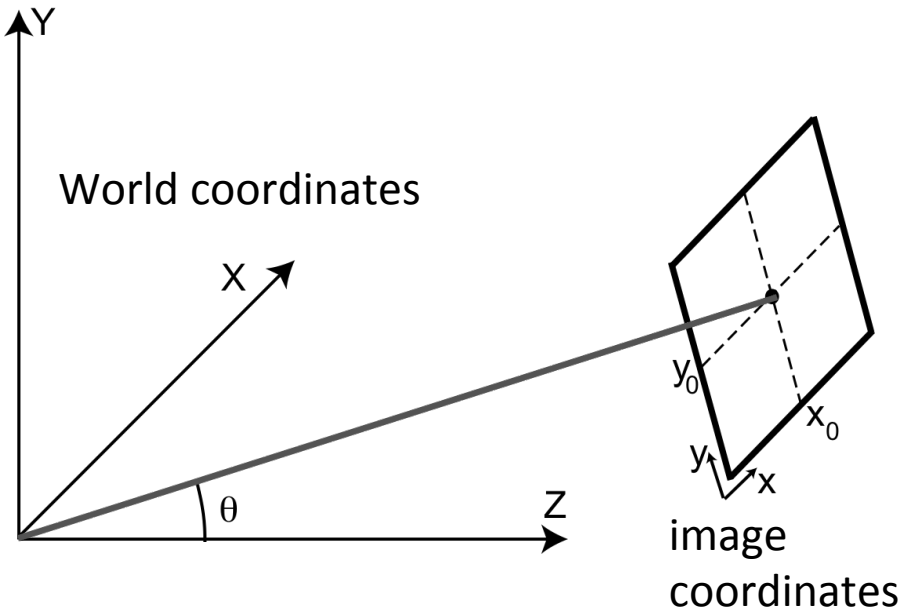
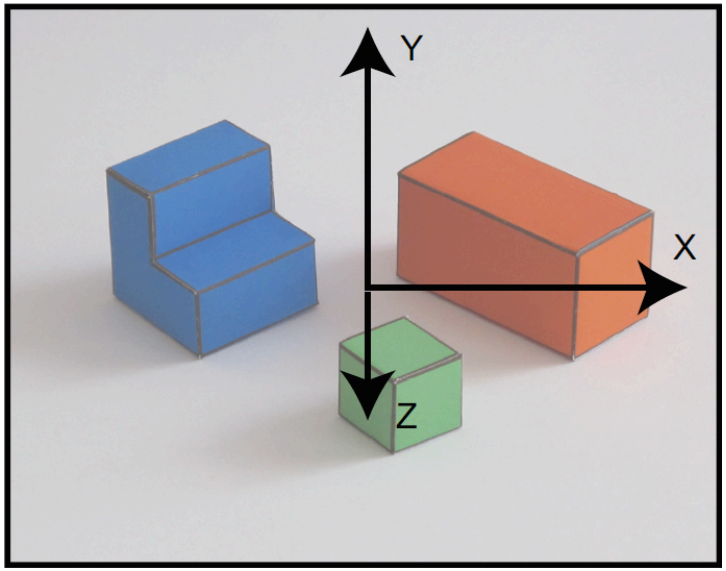


Image and projection of the world coordinate axes into the image plane



World coordinates

$$x = X + x_0$$

$$y = \cos(\theta)\, Y - \sin(\theta)\, Z + y_0$$

image coordinates

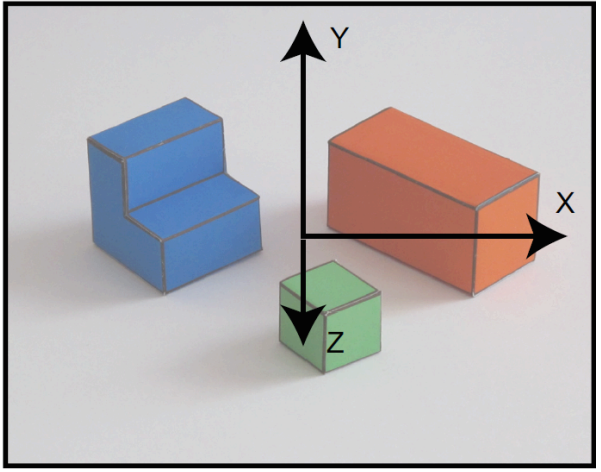
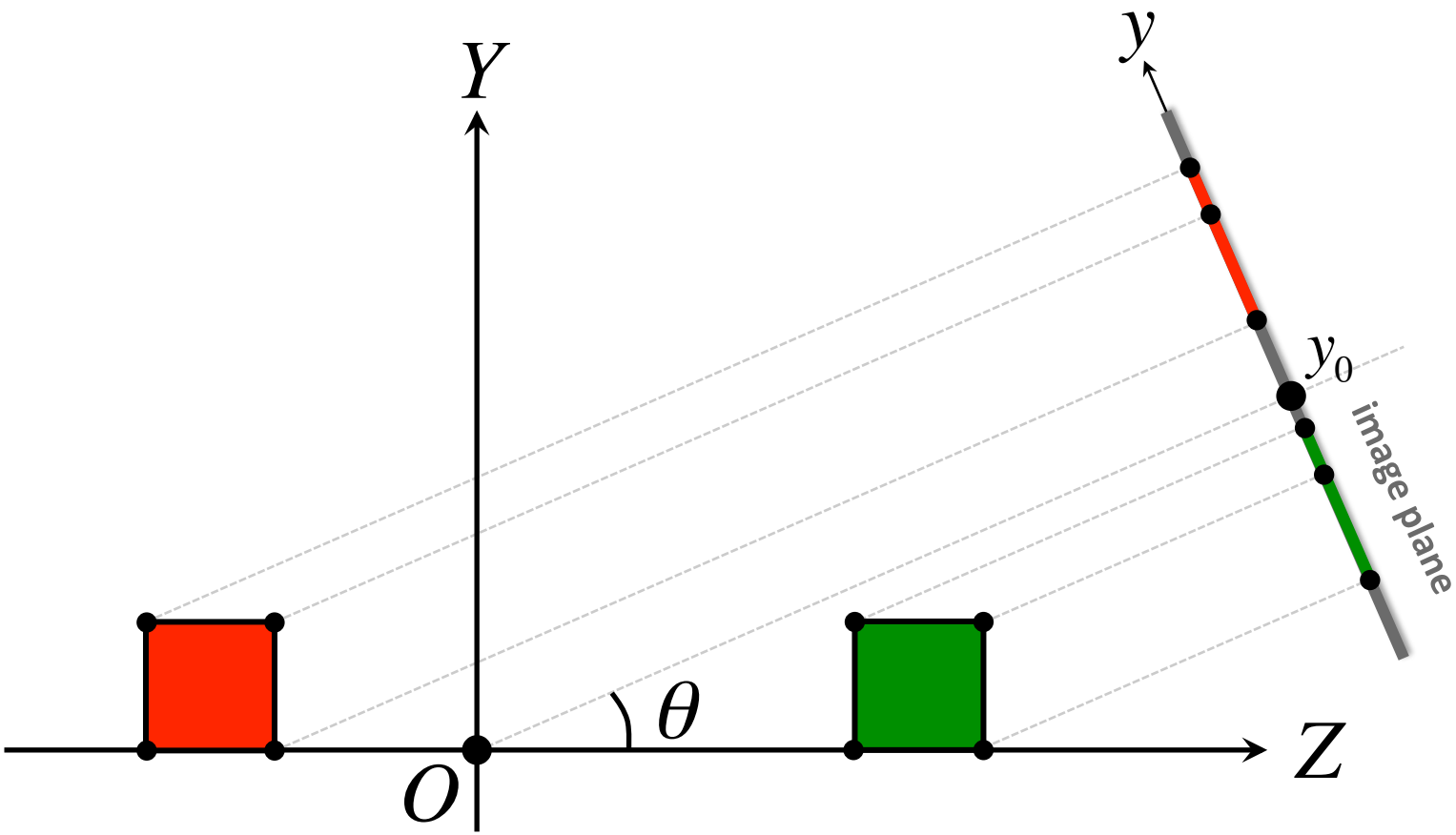


image coordinates

World coordinates

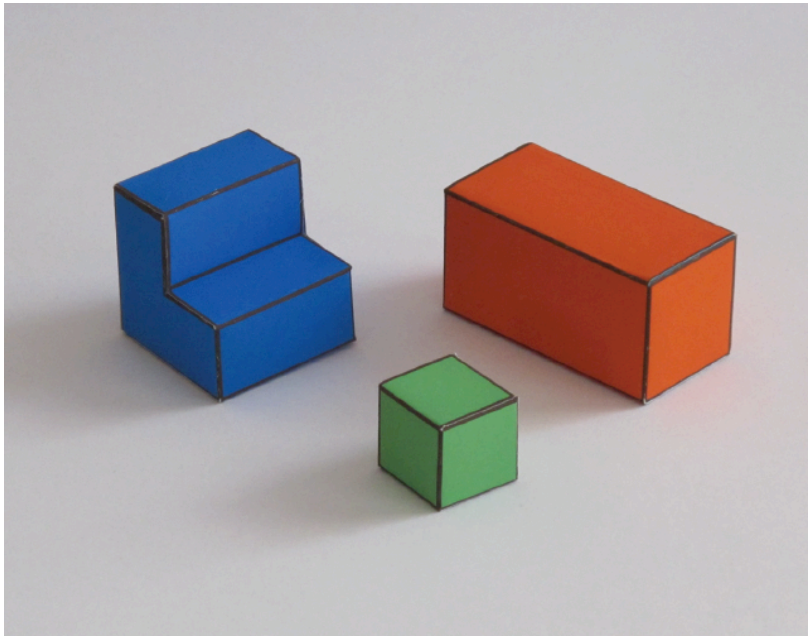
$$x = X + x_0$$

$$y = Y\cos\theta - Z\sin\theta + y_0$$



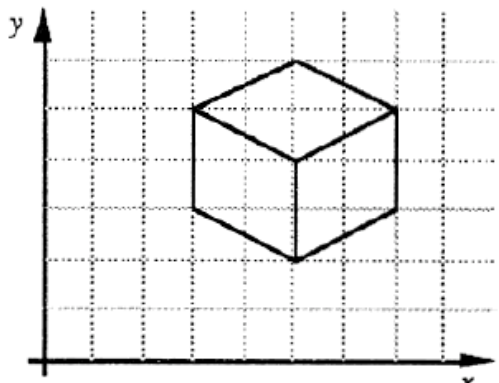
A simple goal

To recover the 3D structure of the world



We want to recover X(x,y), Y(x,y), Z(x,y) using as input I(x,y)

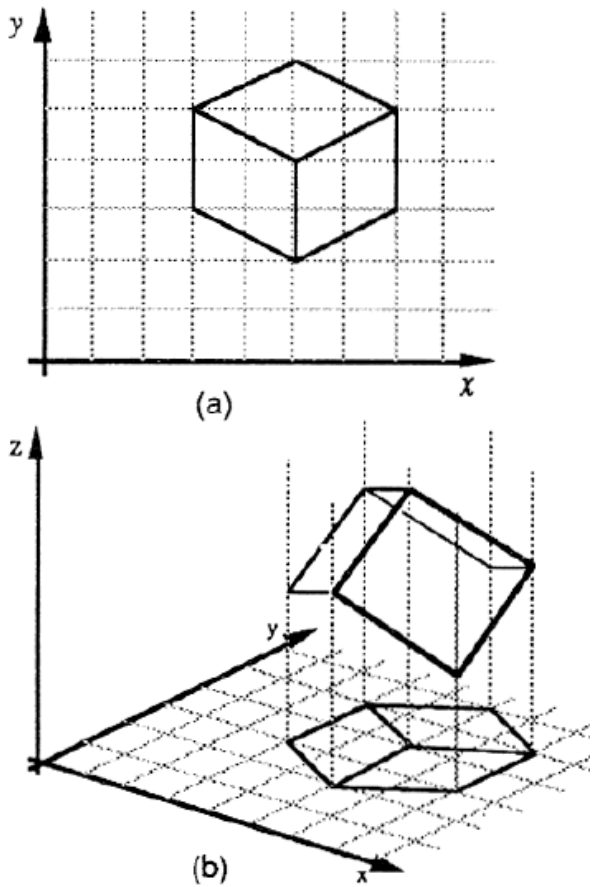
Why is this hard?



(a)

Sinha & Adelson 93

Why is this hard?



Sinha & Adelson 93

Why is this hard?

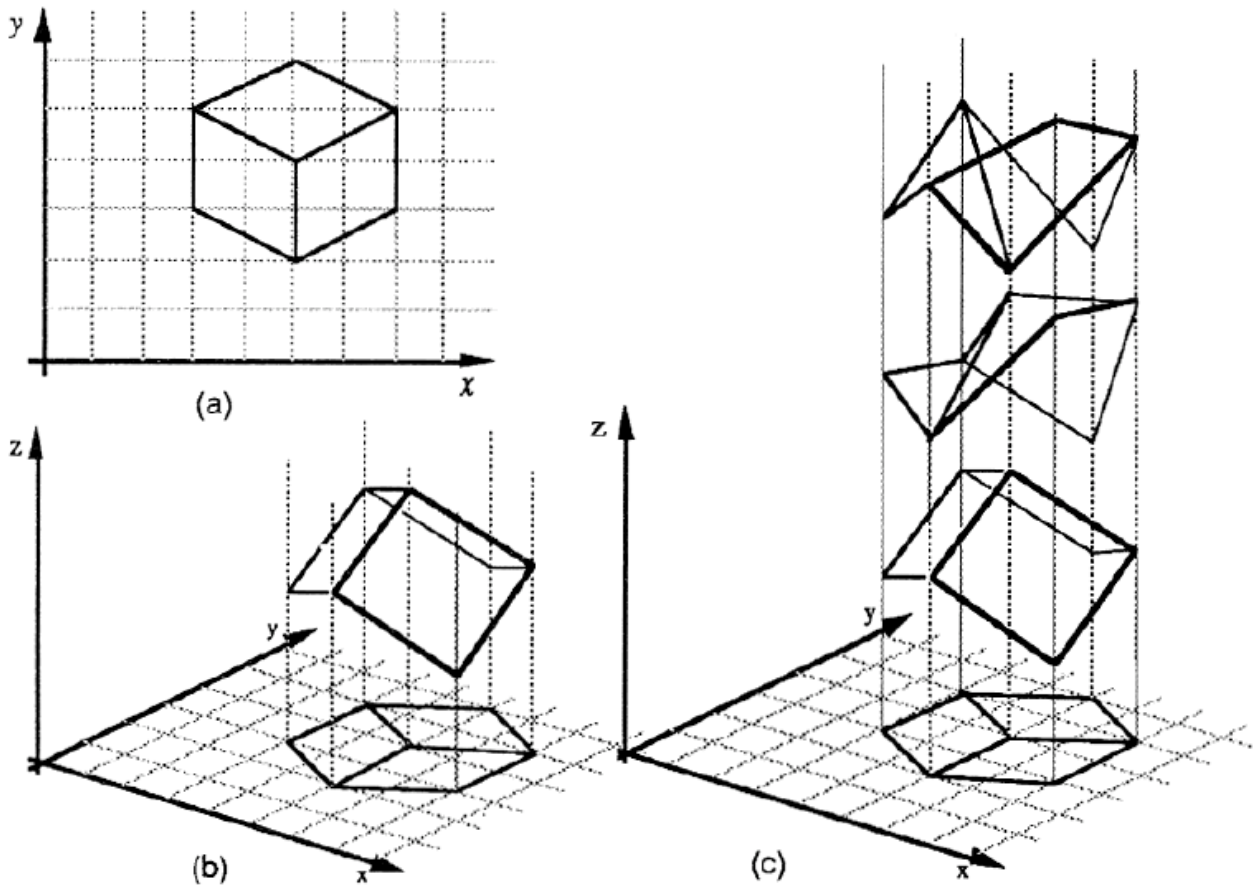
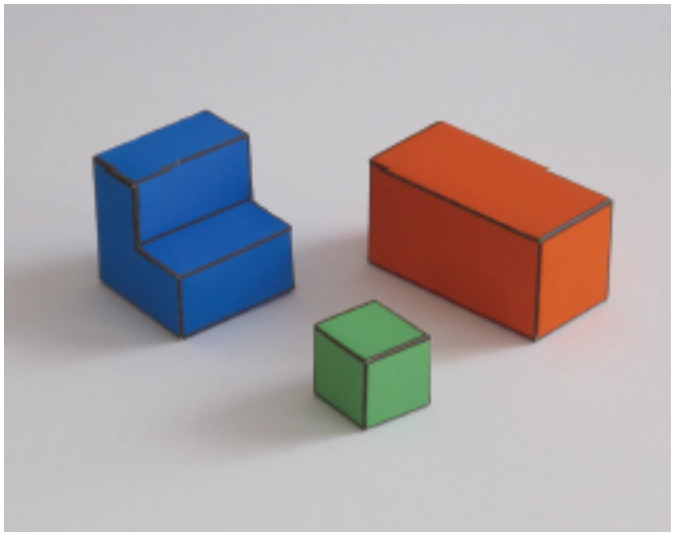
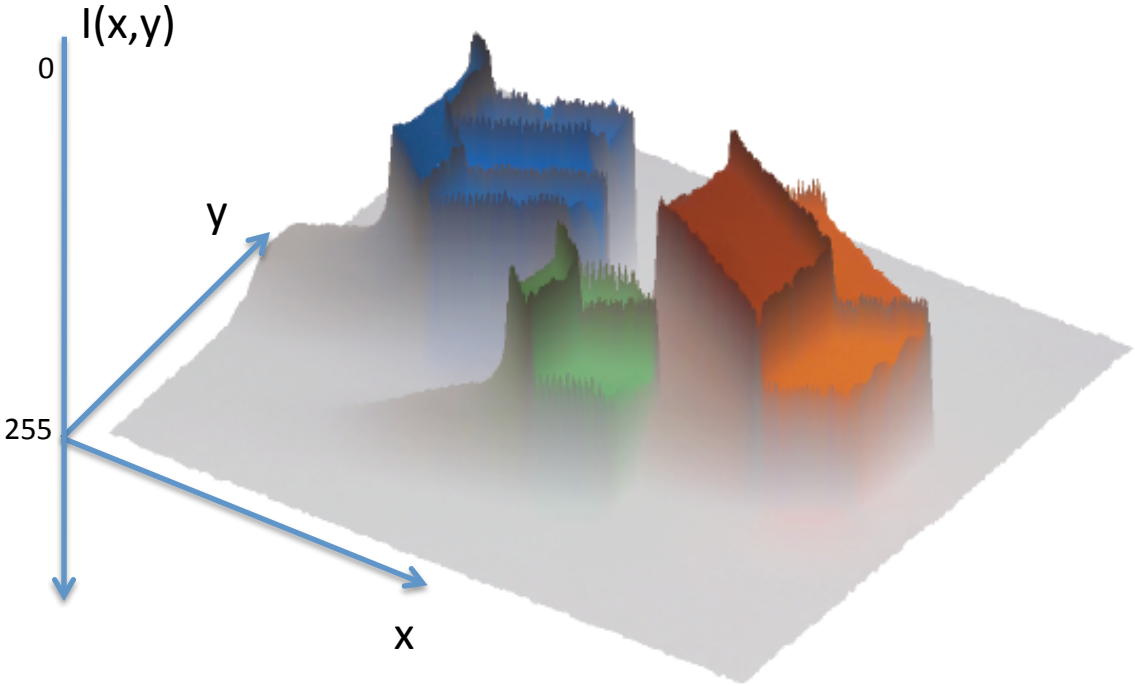


Figure 1. (a) A line drawing provides information only about the x, y coordinates of points lying along the object contours. (b) The human visual system is usually able to reconstruct an object in three dimensions given only a single 2D projection (c) Any planar line-drawing is geometrically consistent with infinitely many 3D structures.

Sinha & Adelson 93

A simple visual system
The input image



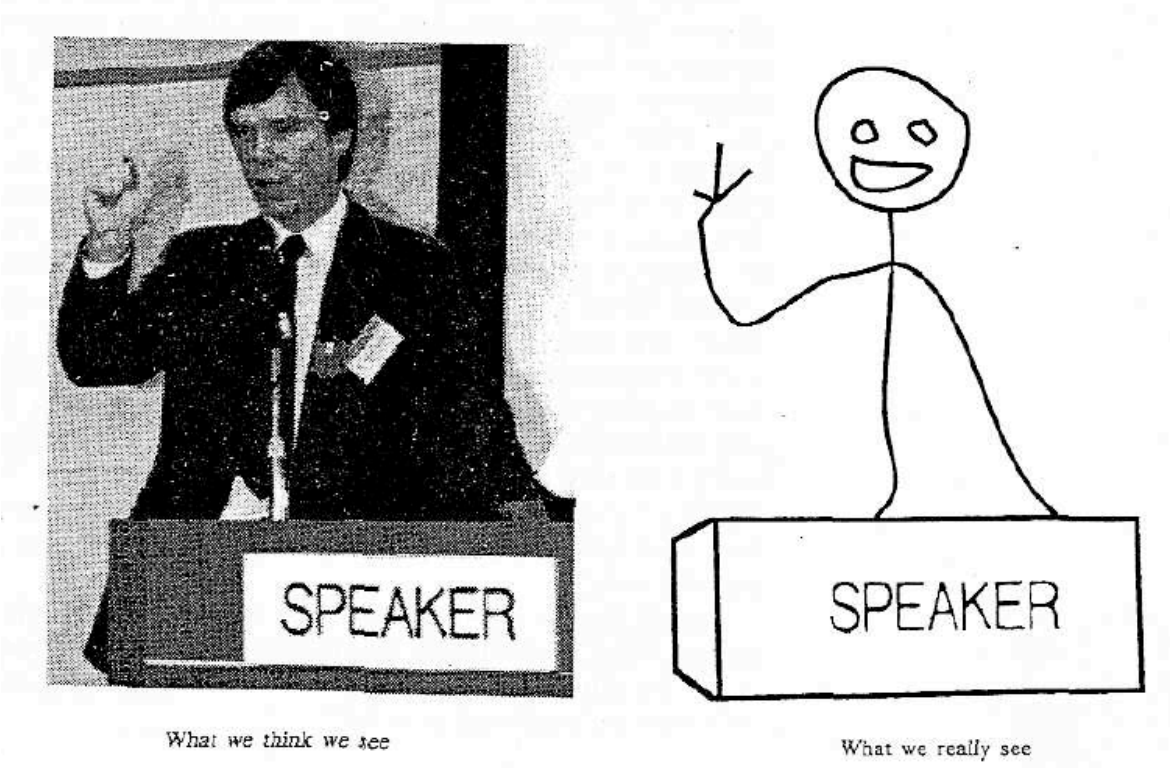


• Proposition 1. The primary task of early vision is to deliver a small set of useful measurements about each observable location in the plenoptic function.

• Proposition 2. The elemental operations of early vision involve the measurement of local change along various directions within the plenoptic function.

Adelson, Bergen. 91

• Goal: to transform the image into other representations (rather than pixel values) that makes scene information more explicit



Cavanagh, Perception 95

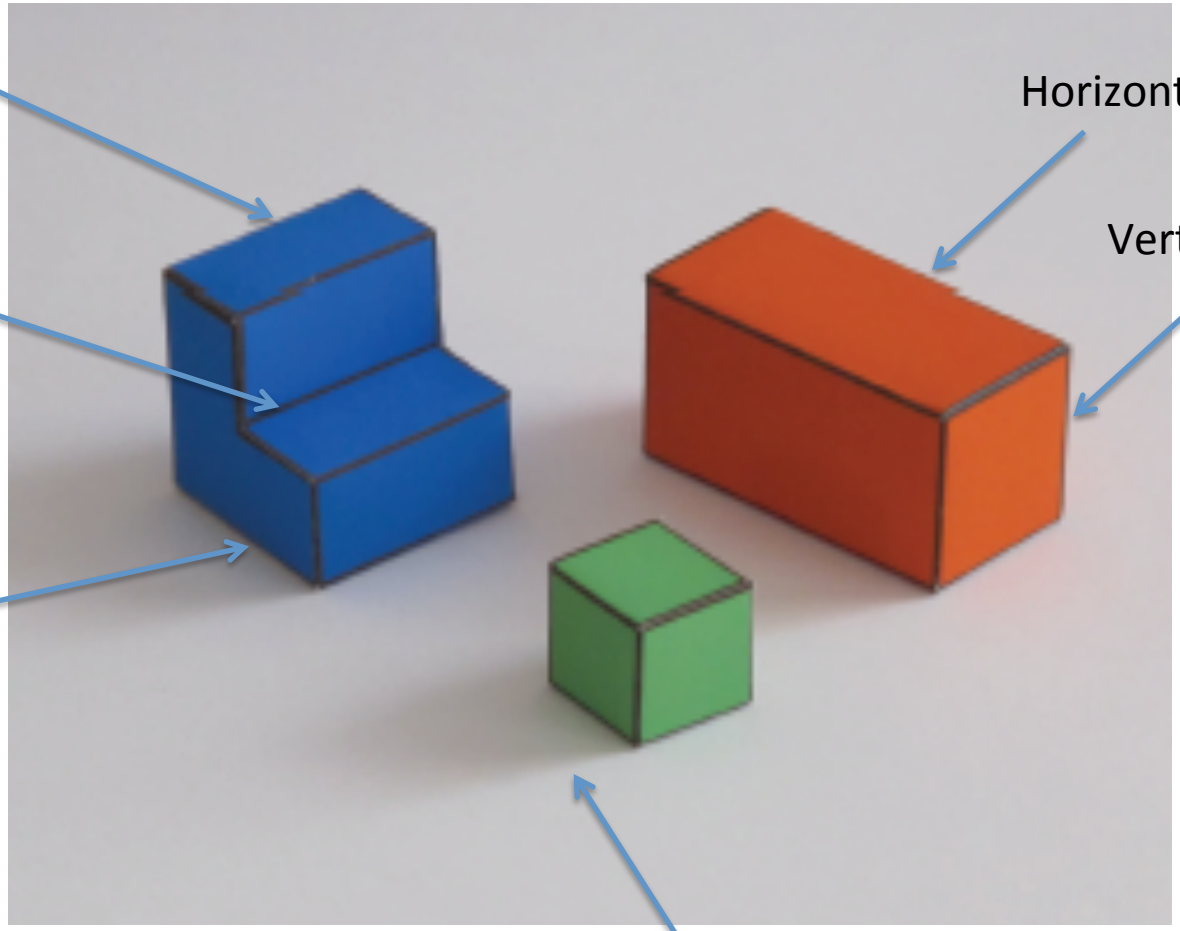
Today

- A simple world
- A catalog of edges
- Generic view assumption
- Constrained solver
- Edge detection: Canny
- Line detection
  - RANSAC
  - Hough transformation

Object boundaries

1. **Changes in surface orientation**:
2. **Occlusion boundaries**: in this simple world, we will position the objects in such a way that objects do not occlude each other.
3. **Contact edges**: this is an edge between two objects but with no depth discontinuity.
4. **Shadow edges**: this can be harder than it seems. In this simple world, shadows are soft creating slow transitions between dark and light.

Edges


Occlusion
Change of
Surface orientation
Contact edge
Horizontal 3D edge
Vertical 3D edge
Shadow boundary

Finding edges in the image

Image gradient:

$$\nabla \mathbf{I} = \left(\frac{\partial \mathbf{I}}{\partial x}, \frac{\partial \mathbf{I}}{\partial y} \right)$$

Approximation image derivative:

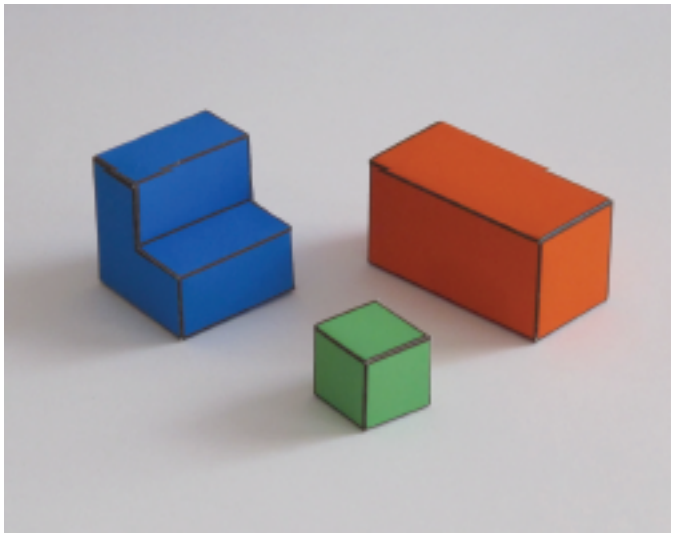
$$\frac{\partial \mathbf{I}}{\partial x} \simeq \mathbf{I}(x, y) - \mathbf{I}(x-1, y)$$

Edge strength $E(x, y) = |\nabla \mathbf{I}(x, y)|$

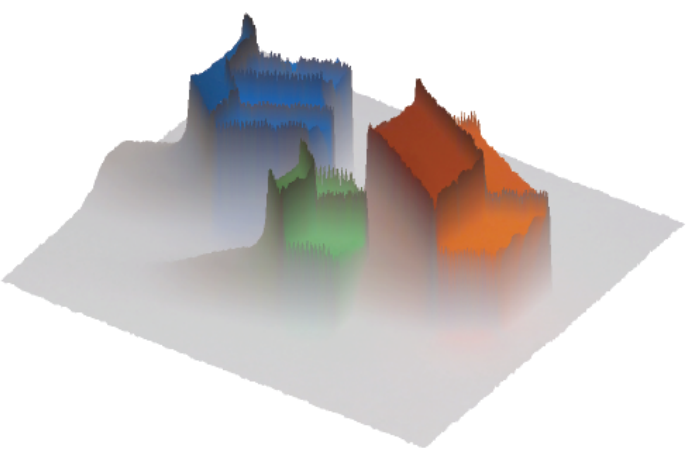
Edge orientation: $\theta(x, y) = \angle \nabla \mathbf{I} = \arctan \frac{\partial \mathbf{I}/\partial y}{\partial \mathbf{I}/\partial x}$

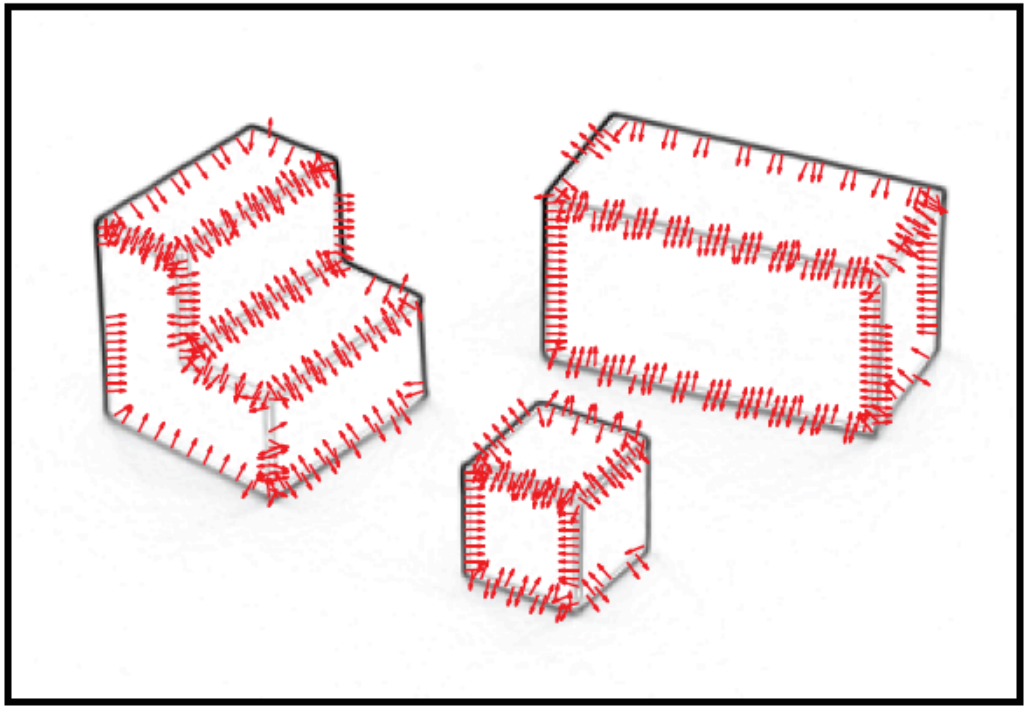
Edge normal: $\mathbf{n} = \frac{\nabla \mathbf{I}}{|\nabla \mathbf{I}|}$

Finding edges in the image



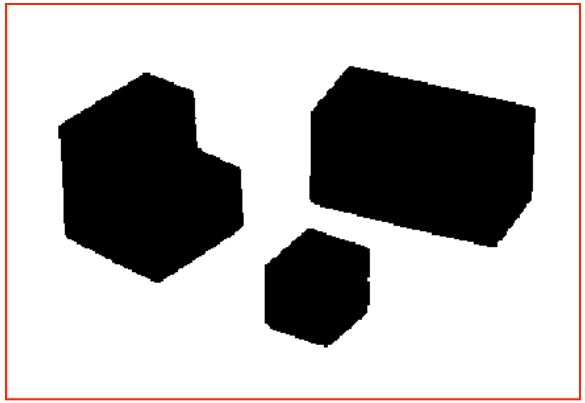
$$\nabla \mathbf{I} = \left(\frac{\partial \mathbf{I}}{\partial x}, \frac{\partial \mathbf{I}}{\partial y} \right) \qquad \mathbf{n} = \frac{\nabla \mathbf{I}}{|\nabla \mathbf{I}|}$$



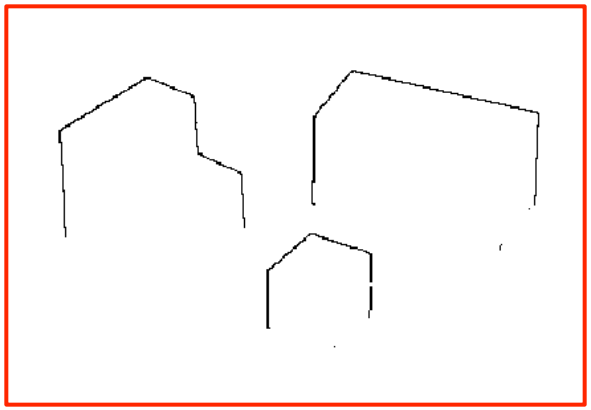


Edge classification

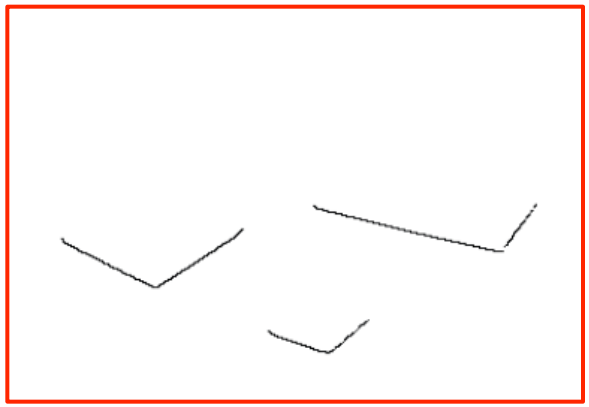
- Figure/ground segmentation



- Occlusion edges
  - Occlusion edges are owned by the foreground



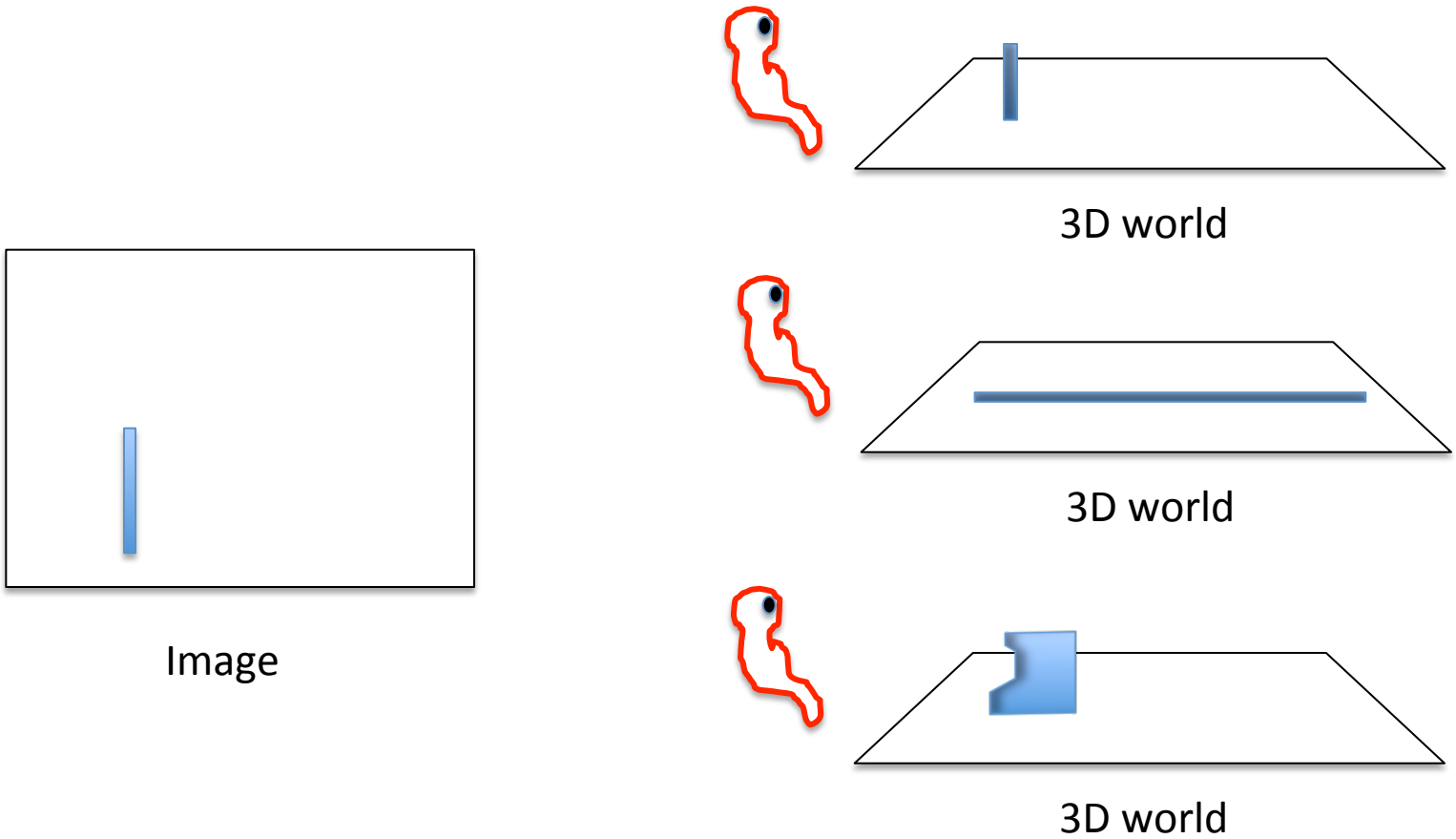
- Contact edges



Today

- A simple world
- A catalog of edges
- Generic view assumption
- Constrained solver
- Edge detection: Canny
- Line detection
  - RANSAC
  - Hough transformation

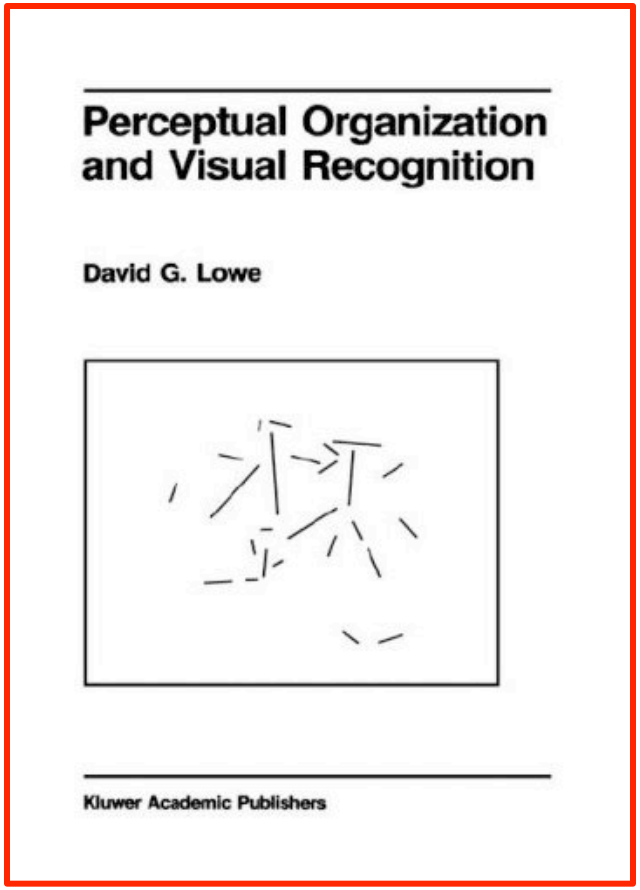
Generic view assumption



Generic view assumption: the observer should not assume that he has a special position in the world… The most generic interpretation is to see a vertical line as a vertical line in 3D.

Freeman, 93

Non-accidental properties



D. Lowe, 1985

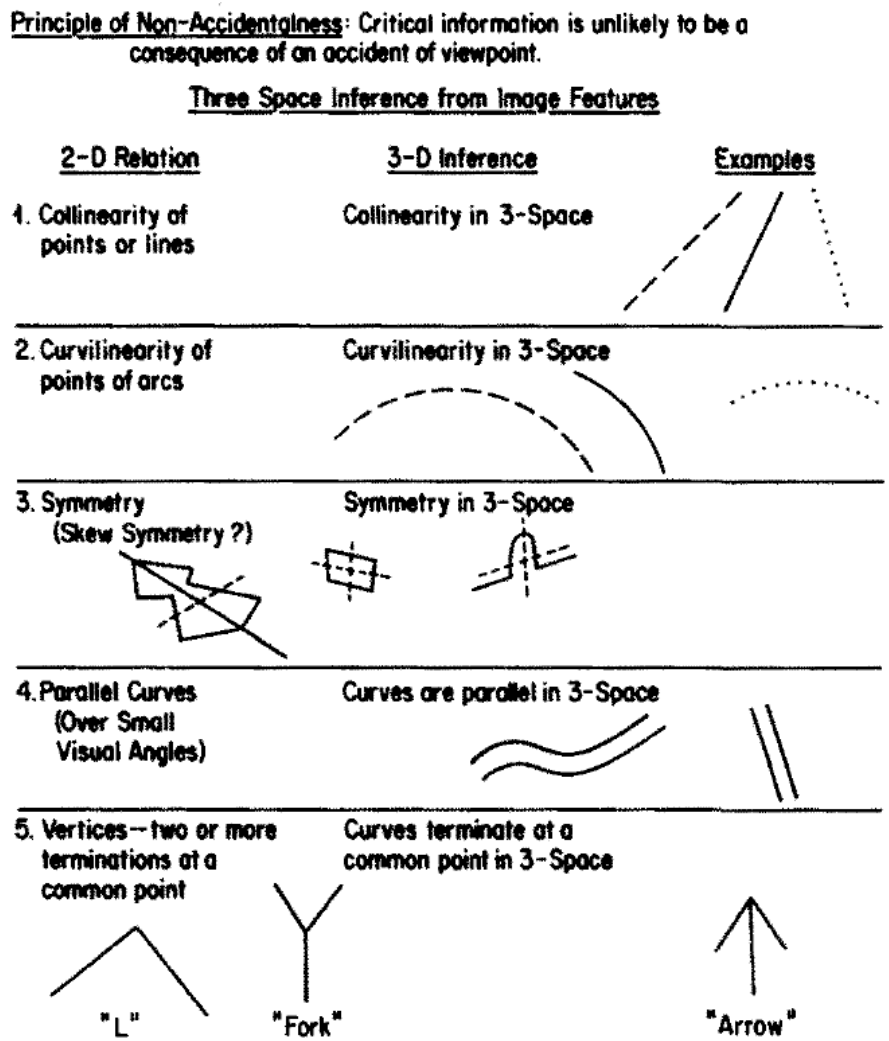
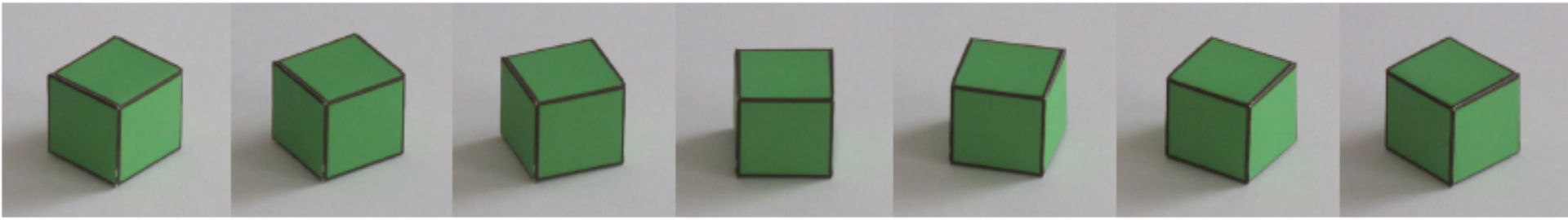


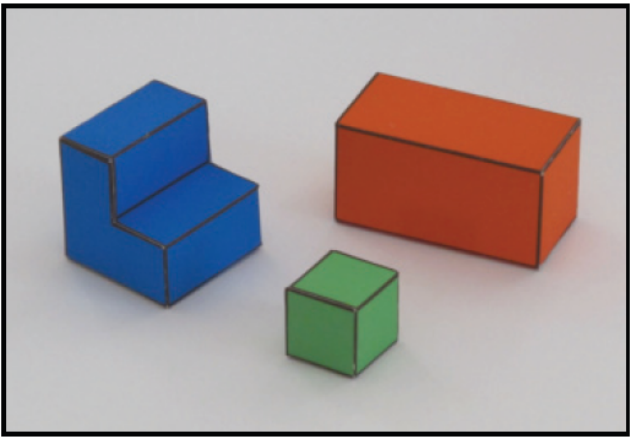
Figure 4. Five nonaccidental relations. (From Figure 5.2, *Perceptual organization and visual recognition* [p. 77] by David Lowe. Unpublished doctorial dissertation, Stanford University. Adapted by permission.)

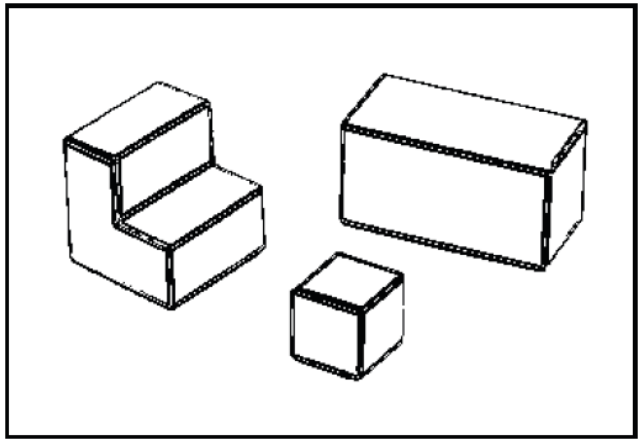
Biederman_RBC_1987

Non-accidental properties in the simple world

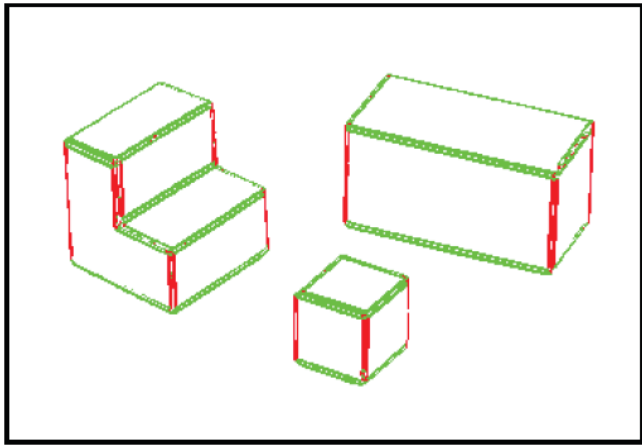


generic | generic | generic | accidental | generic | generic | generic





Using E(x,y)



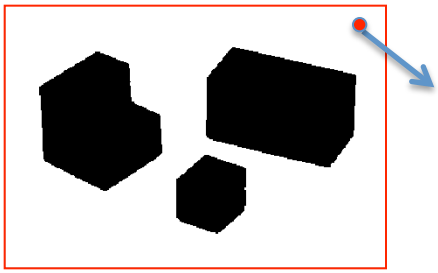
Using θ(x,y)

Today

- A simple world
- A catalog of edges
- Generic view assumption
- Constrained solver
- Edge detection: Canny
- Line detection
  - RANSAC
  - Hough transformation

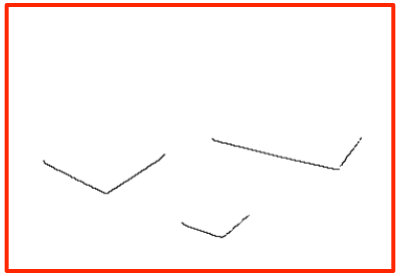
From edges to surface constraints

- Ground



Y(x,y) = 0 if (x,y) belongs to a ground pixel

- Contact edge



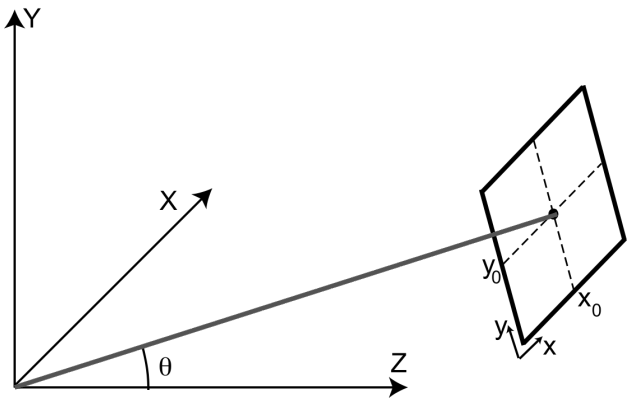
Y(x,y) = 0 if (x,y) belongs to foreground and is a contact edge

Next, things get a bit more complicated.

From edges to surface constraints

How can we relate the information in the pixels with 3D surfaces in the world?

- Vertical edges



World coordinates

$$x = X + x_0$$

$$y = \cos(\theta)\, Y - \sin(\theta)\, Z + y_0$$

image coordinates

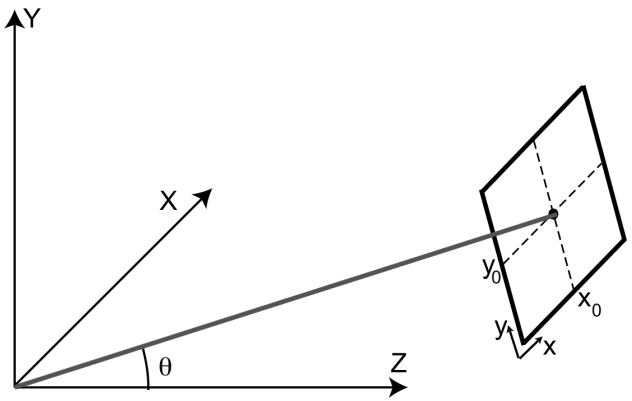
Given the image, what can we say about X, Y and Z in the pixels that belong to a vertical edge?

Z = constant along the edge

$$\partial Y / \partial y = 1/\cos(\theta)$$

From edges to surface constraints

- Horizontal edges



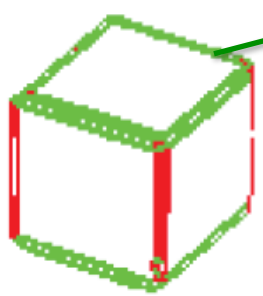
World coordinates

$$x = X + x_0$$

$$y = \cos(\theta)\, Y - \sin(\theta)\, Z + y_0$$

image coordinates

Given the image, what can we say about X, Y and Z in the pixels that belong to an horizontal 3D edge?



Y = constant along the edge

$$\partial Y / \partial \mathbf{t} = 0$$

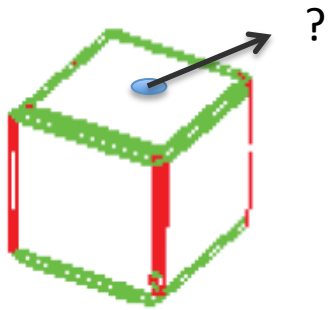
Where **t** is the vector parallel to the edge

$$\mathbf{t} = (-n_y, n_x)$$

$$\partial Y / \partial \mathbf{t} = \nabla Y \cdot \mathbf{t} = -n_y \partial Y / \partial x + n_x \partial Y / \partial y$$

From edges to surface constraints

- What happens where there are no edges?



Assumption of planar faces:
i.e. change of change is zero

$$\begin{aligned} \partial^2 Y/\partial x^2 &= 0 \\ \partial^2 Y/\partial y^2 &= 0 \\ \partial^2 Y/\partial y \partial x &= 0 \end{aligned}$$

Information has to be propagated from the edges

A simple inference scheme

All the constraints are linear

$$Y = 0$$ if (x,y) belongs to a ground pixel

$$\partial Y/\partial y = 1/\cos(\theta)$$ if (x,y) belongs to a vertical edge

$$\partial Y/\partial \mathbf{t} = 0$$ if (x,y) belongs to an horizontal edge

$$\begin{aligned} \partial^2 Y/\partial x^2 &= 0 \\ \partial^2 Y/\partial y^2 &= 0 \\ \partial^2 Y/\partial y \partial x &= 0 \end{aligned}$$ if (x,y) is not an edge

Discrete approximation

We can transform every differential constrain into a discrete linear constraint on Y(x,y)

Y(x,y)

111	115	113	111	112	111	112	111
135	138	137	139	145	146	149	147
163	168	188	196	206	202	206	207
180	184	206	219	202	200	195	193
189	193	214	216	104	79	83	77
191	201	217	220	103	59	60	68
195	205	216	222	113	68	69	83
199	203	223	228	108	68	71	77

$$\frac{dY}{dx} \approx Y(x,y) - Y(x-1,y)$$

-1	1

A slightly better approximation

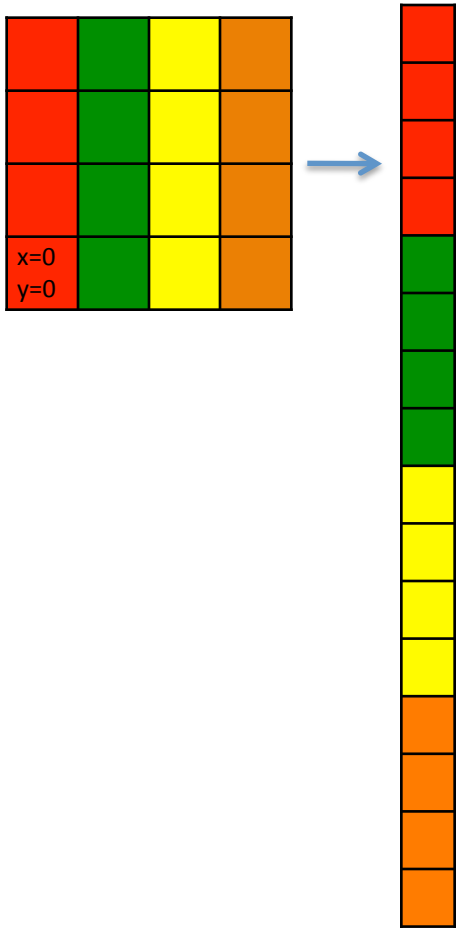
(it is symmetric, and it averages horizontal derivatives over 3 vertical locations)

-1	0	1
-2	0	2
-1	0	1

Discrete approximation

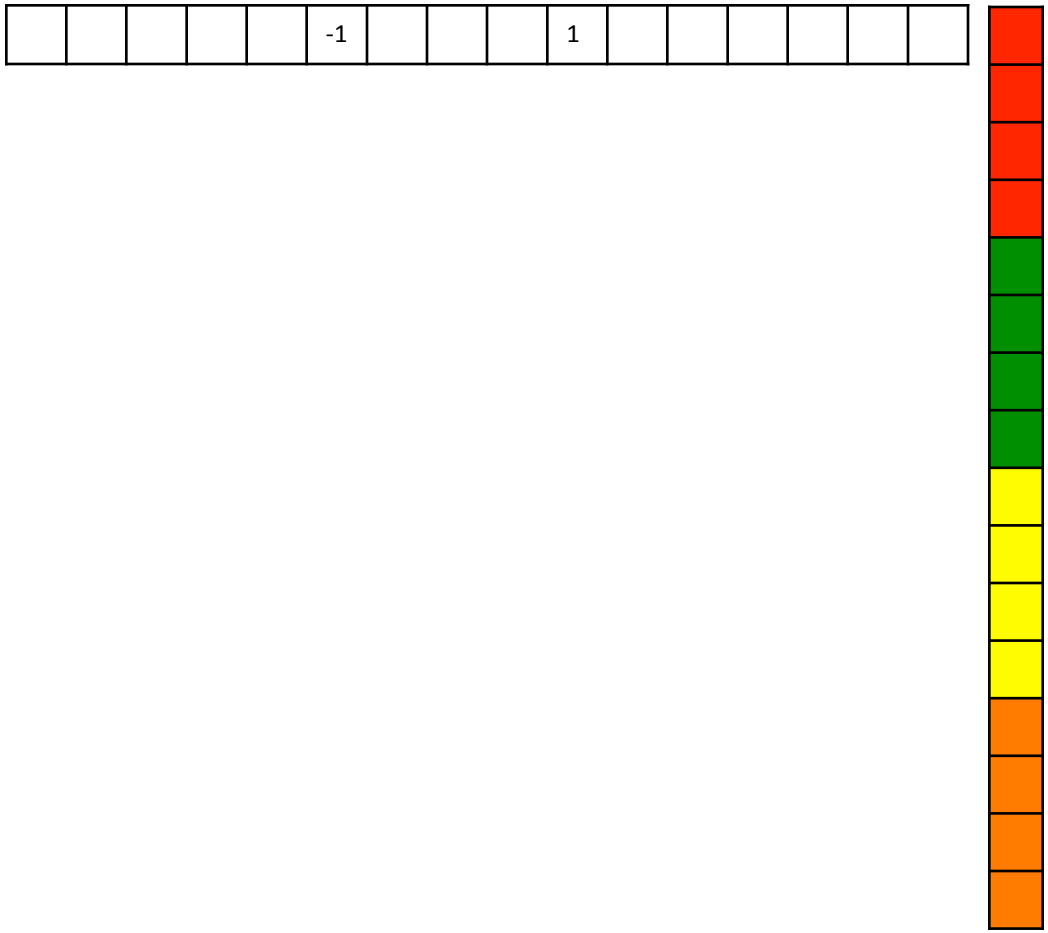
Transform the "image" Y(x,y) into a column vector:

Y(x,y)

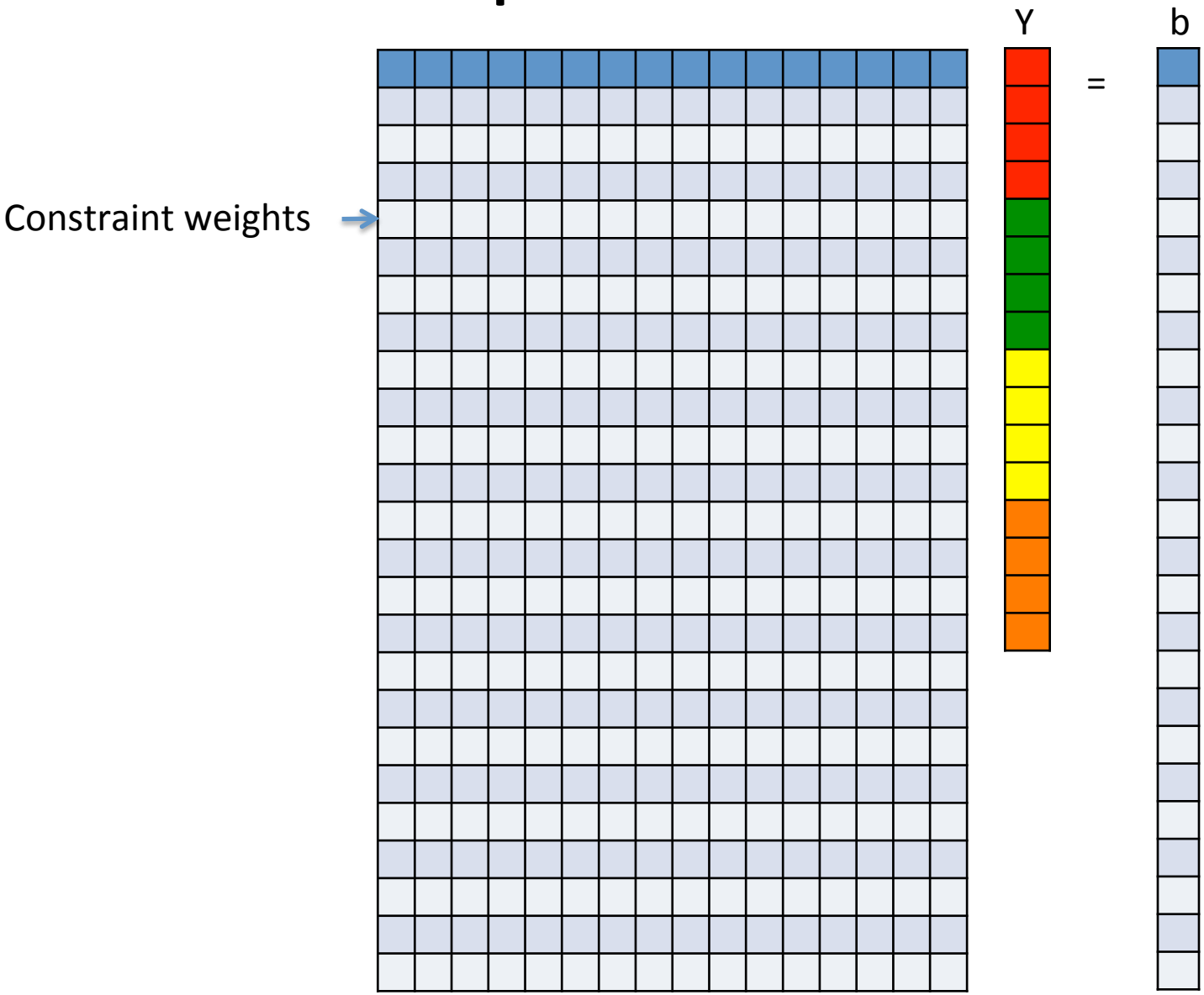


x=2, y=2

$$\frac{dY}{dx} \approx Y(x,y) - Y(x-1,y) = Y(2,2) - Y(1,2) =$$



A simple inference scheme

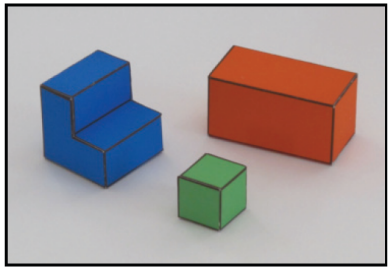


$A\,Y = b$

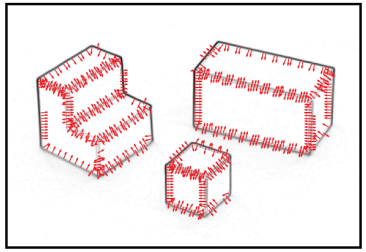
$Y = (A^T A)^{-1} A^T b$

```
Matlab
Y = A\b;
```

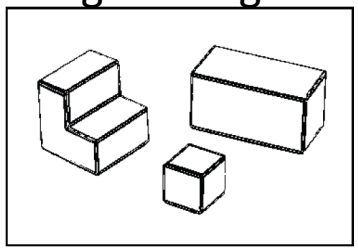
Results



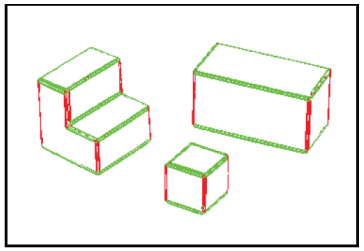
Edge normals



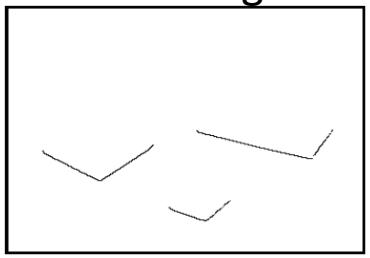
Edge strength



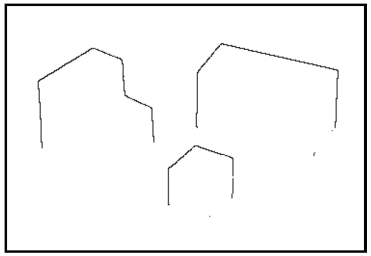
3D orientation

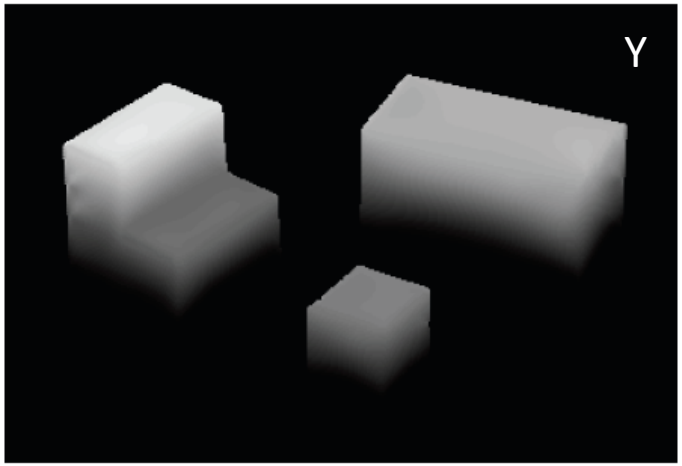


Contact edges

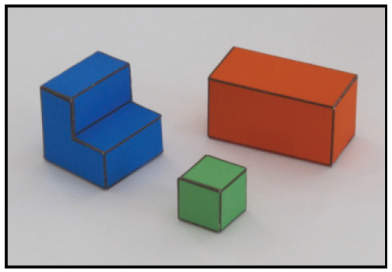


Depth discontinuities



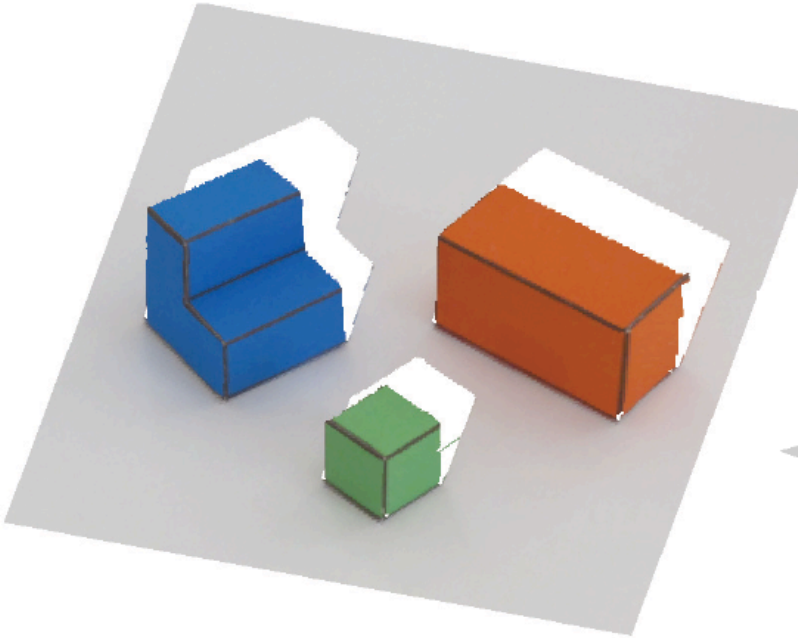


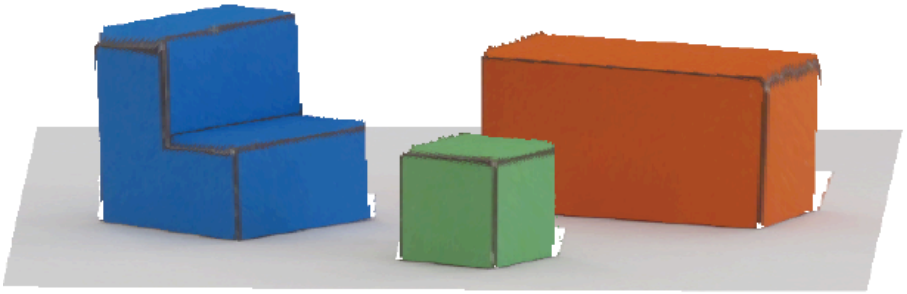
Input

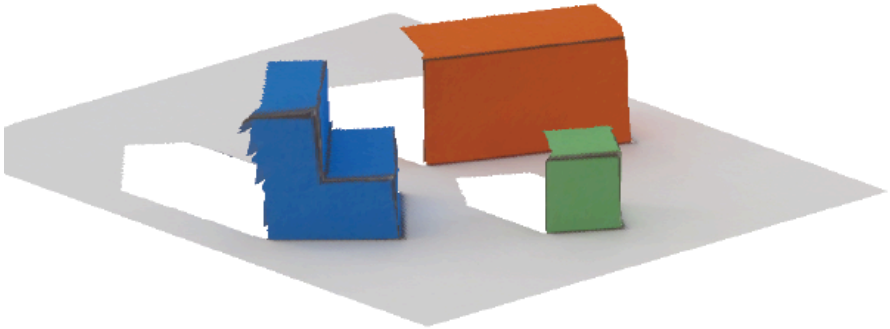


Changing view point

New view points:

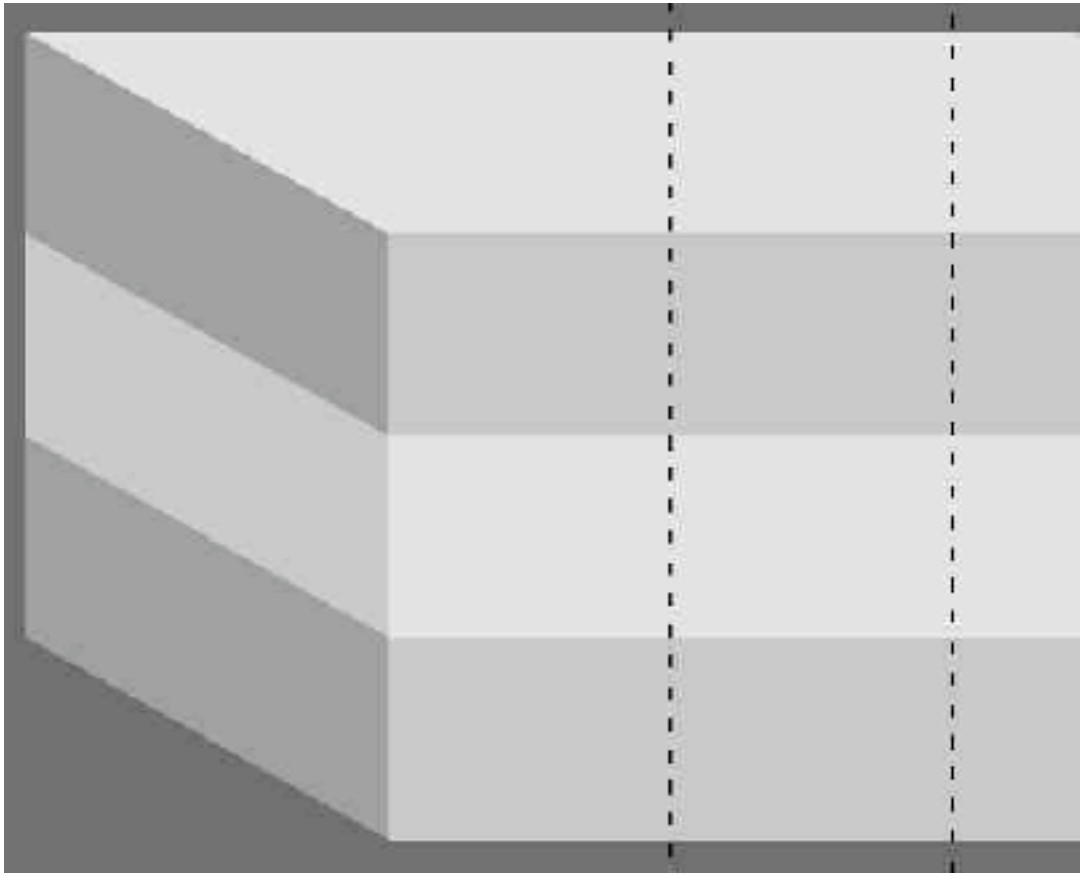




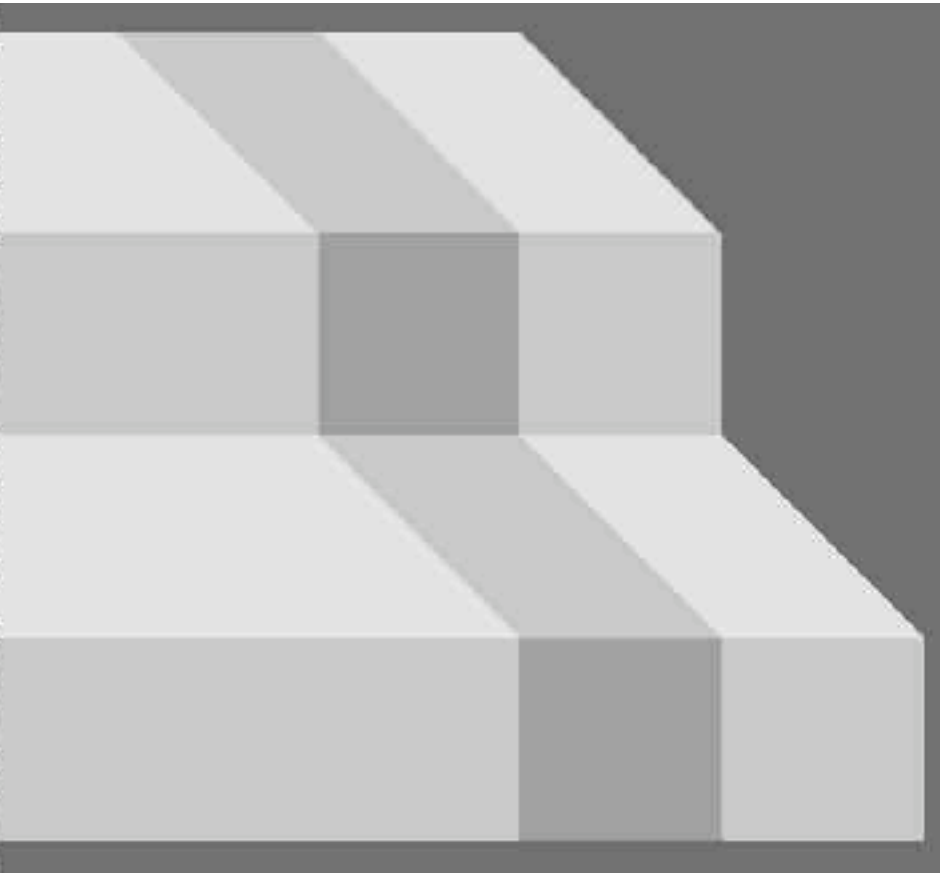


Violations of simple world assumptions

Shading is due to painted stripes



Violations of simple world assumptions



Shading is due to illumination

Violations of simple world assumptions

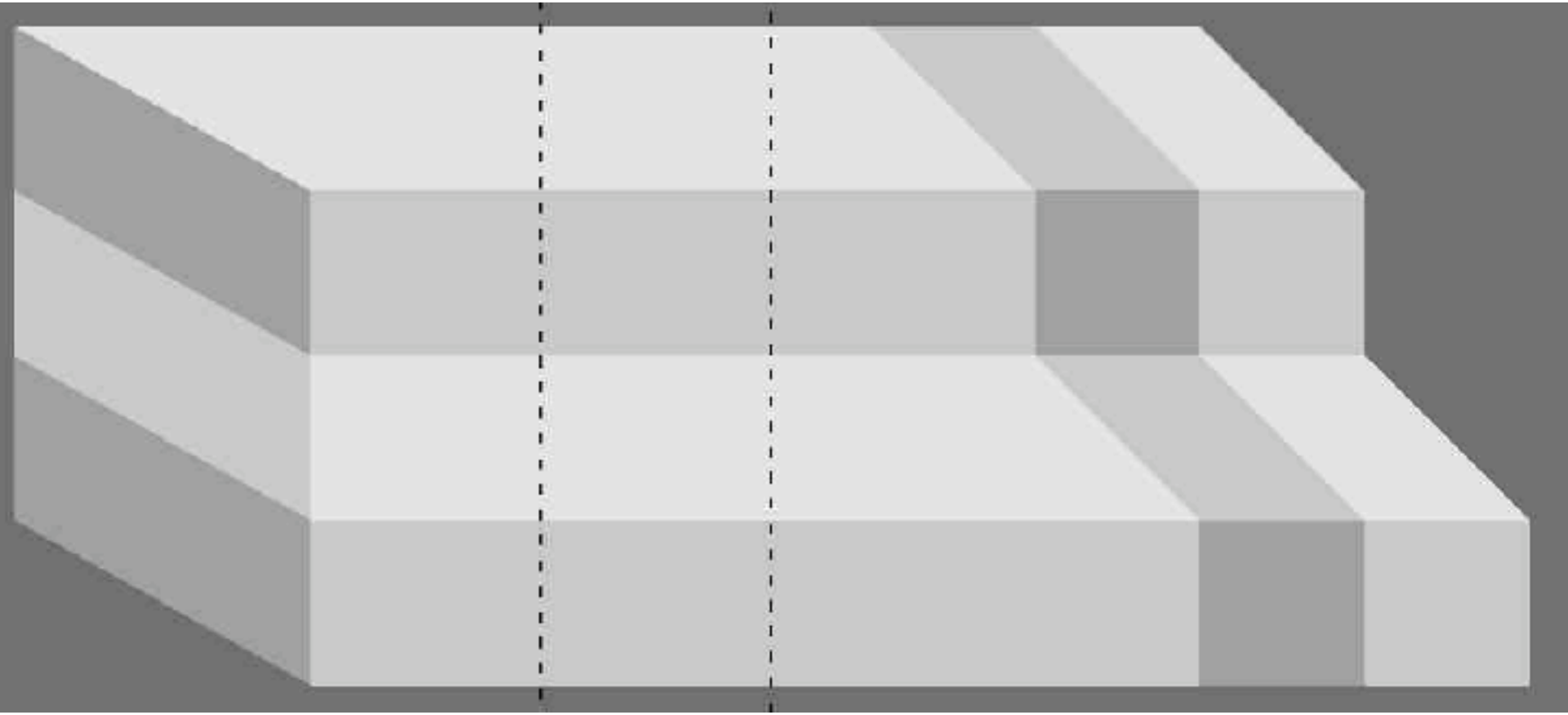


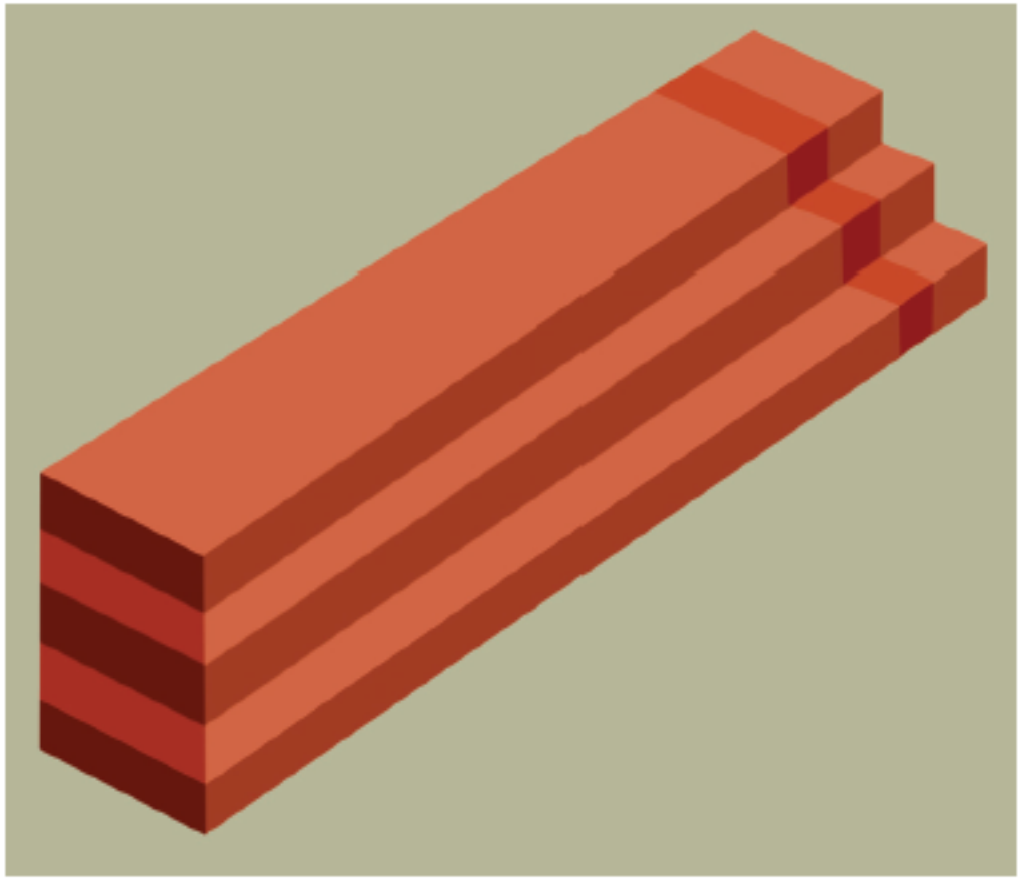
FIGURE 24.9 The impossible steps. On the left, the horizontal stripes appear to be due to paint; on the right, they appear to be due to shading.

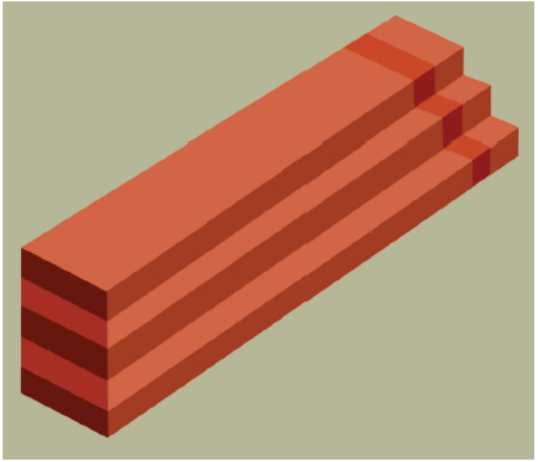
Adelson, E.H. Lightness Perception and Lightness Illusions. In *The New Cognitive Neurosciences*, 2nd ed., M. Gazzaniga, ed. Cambridge, MA: MIT Press, pp. 339-351, (2000).

24 Lightness Perception and Lightness Illusions

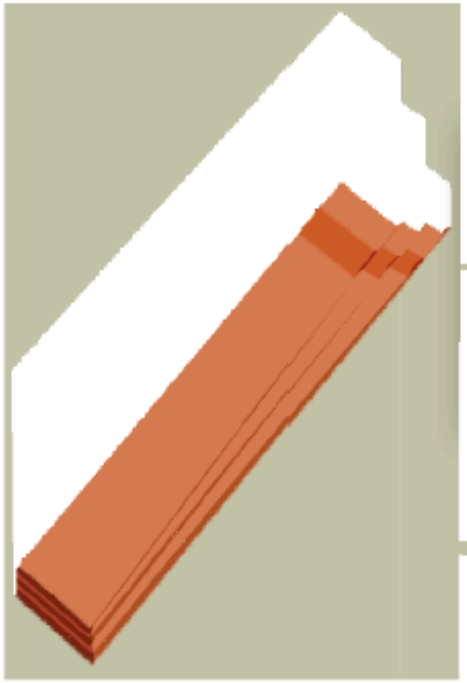
EDWARD H. ADELSON

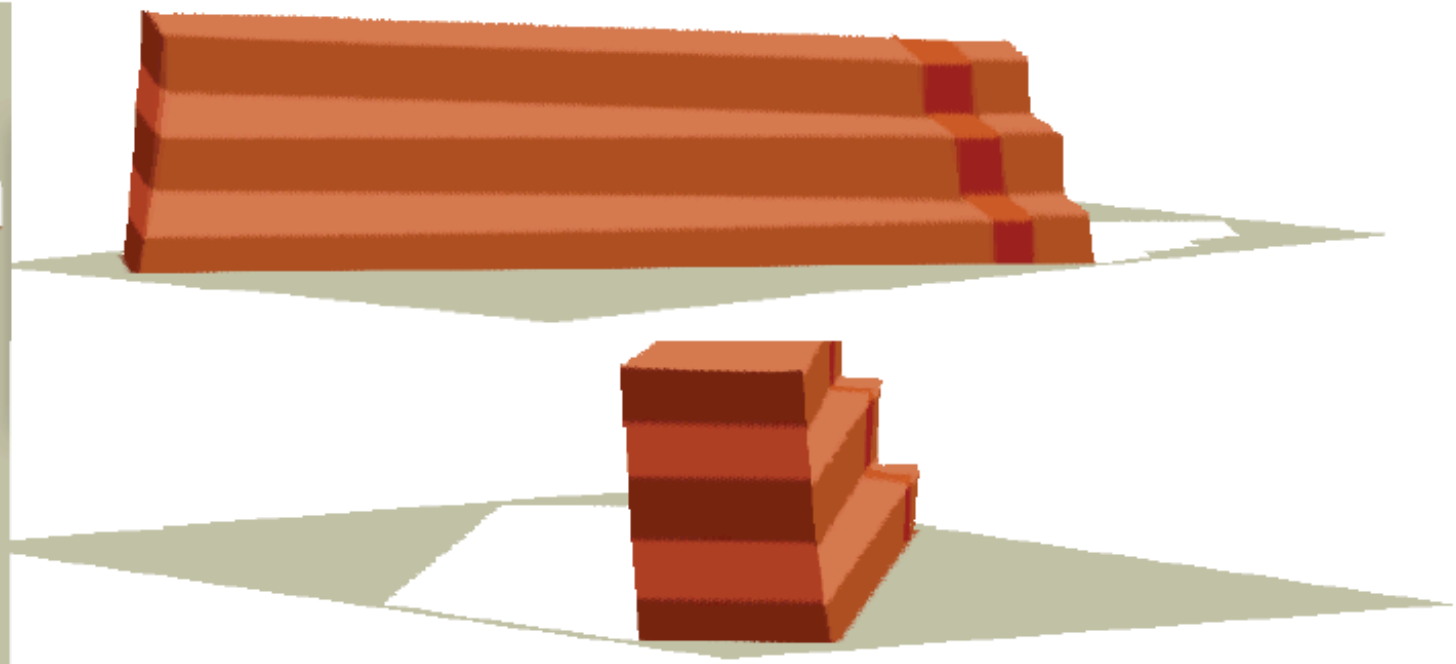
Impossible steps





Impossible steps





What is missing?

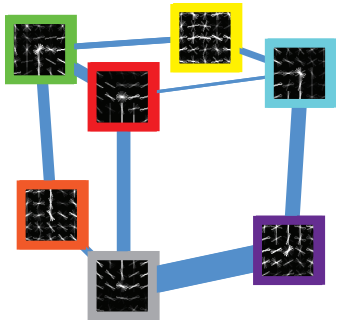
- Recognition
- The system can not do simple tasks like counting the number of cubes.

Different Approach

- A different approach to the one discussed here is model based scene interpretation where we could have a set of predefined models of the objects that can be present in the scene and the system should try to decide if they are present or not in the image, and recover their parameters (pose, color, etc.)
- Recognition allows indexing properties that are not directly available in the image. For instance, we can infer the shape of the invisible surfaces. Recognizing each geometric figure also implies extracting the parameters of each figure (pose, size, ...).

3D Cuboid Detector

Input image



detect

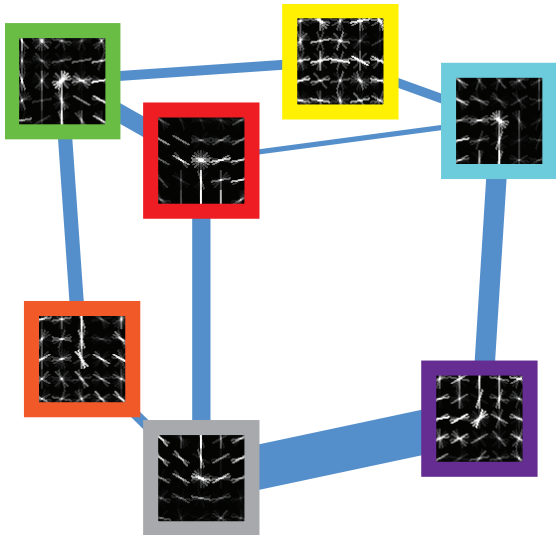
Output detections

Synthesized New Views

http://vision.princeton.edu/projects/2012/SUNprimitive/

Scoring Function

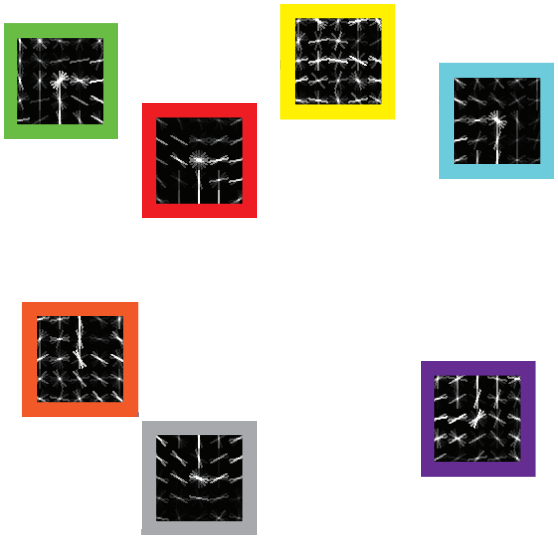
$$S(I,p) = \sum_{i \in \mathcal{V}} w_i^H \cdot \mathrm{HOG}(I, p_i) + \sum_{ij \in \mathcal{E}} w_{ij}^D \cdot \mathrm{Displacement}^{\mathrm{2D}}(p_i, p_j)$$
$$+ \sum_{ij \in \mathcal{E}} w_{ij}^E \cdot \mathrm{Edge}(I, p_i, p_j) + w^S \cdot \mathrm{Shape}^{\mathrm{3D}}(p)$$



Scoring Function

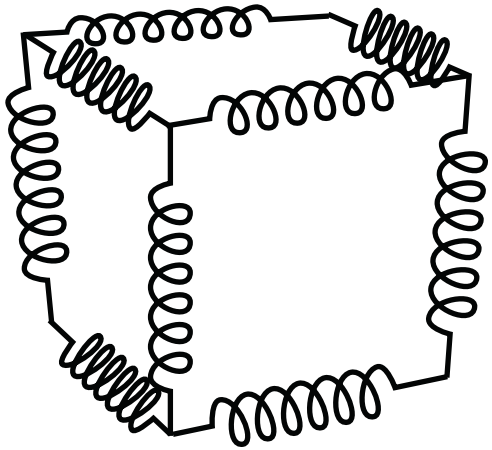
$$S(I,p) = \boxed{\sum_{i \in \mathcal{V}} w_i^H \cdot \mathrm{HOG}(I, p_i)} + \sum_{ij \in \mathcal{E}} w_{ij}^D \cdot \mathrm{Displacement}^{\mathrm{2D}}(p_i, p_j)$$

$$+ \sum_{ij \in \mathcal{E}} w_{ij}^E \cdot \mathrm{Edge}(I, p_i, p_j) + w^S \cdot \mathrm{Shape}^{\mathrm{3D}}(p)$$



Scoring Function

$$S(I,p) = \sum_{i \in \mathcal{V}} w_i^H \cdot \mathrm{HOG}(I, p_i) + \boxed{\sum_{ij \in \mathcal{E}} w_{ij}^D \cdot \mathrm{Displacement}^{\mathrm{2D}}(p_i, p_j)}$$
$$+ \sum_{ij \in \mathcal{E}} w_{ij}^E \cdot \mathrm{Edge}(I, p_i, p_j) + w^S \cdot \mathrm{Shape}^{\mathrm{3D}}(p)$$



Scoring Function

$$S(I,p) = \sum_{i \in \mathcal{V}} w_i^H \cdot \mathrm{HOG}(I, p_i) + \sum_{ij \in \mathcal{E}} w_{ij}^D \cdot \mathrm{Displacement}^{2\mathrm{D}}(p_i, p_j)$$

$$+ \boxed{\sum_{ij \in \mathcal{E}} w_{ij}^E \cdot \mathrm{Edge}(I, p_i, p_j)} + w^S \cdot \mathrm{Shape}^{3\mathrm{D}}(p)$$

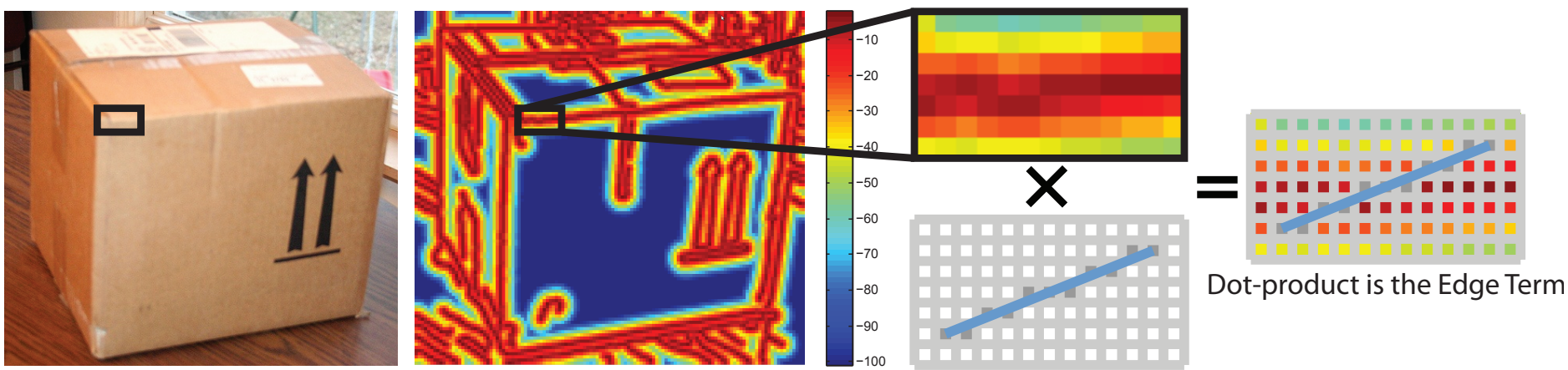


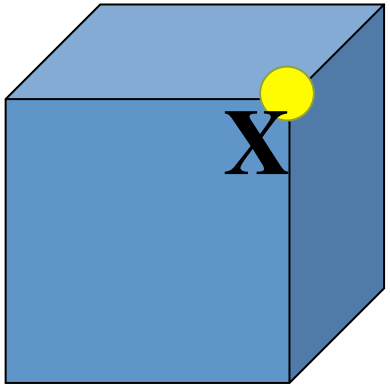
Image | Distance Transformed Edge Map | Pixels Covered by Line Segment

Scoring Function

$$S(I,p) = \sum_{i \in \mathcal{V}} w_i^H \cdot \mathrm{HOG}(I, p_i) + \sum_{ij \in \mathcal{E}} w_{ij}^D \cdot \mathrm{Displacement}^{2\mathrm{D}}(p_i, p_j)$$
$$+ \sum_{ij \in \mathcal{E}} w_{ij}^E \cdot \mathrm{Edge}(I, p_i, p_j) + \boxed{w^S \cdot \mathrm{Shape}^{3\mathrm{D}}(p)}$$

Image (2D)

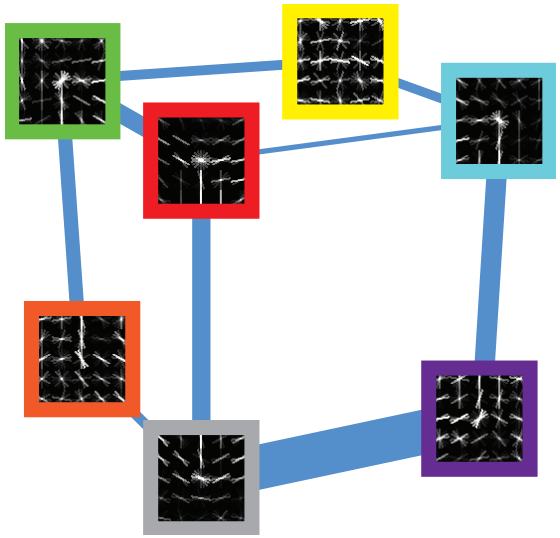
$$\|\mathbf{x} - \mathbf{PLX}\|$$



Unit Cuboid (3D)

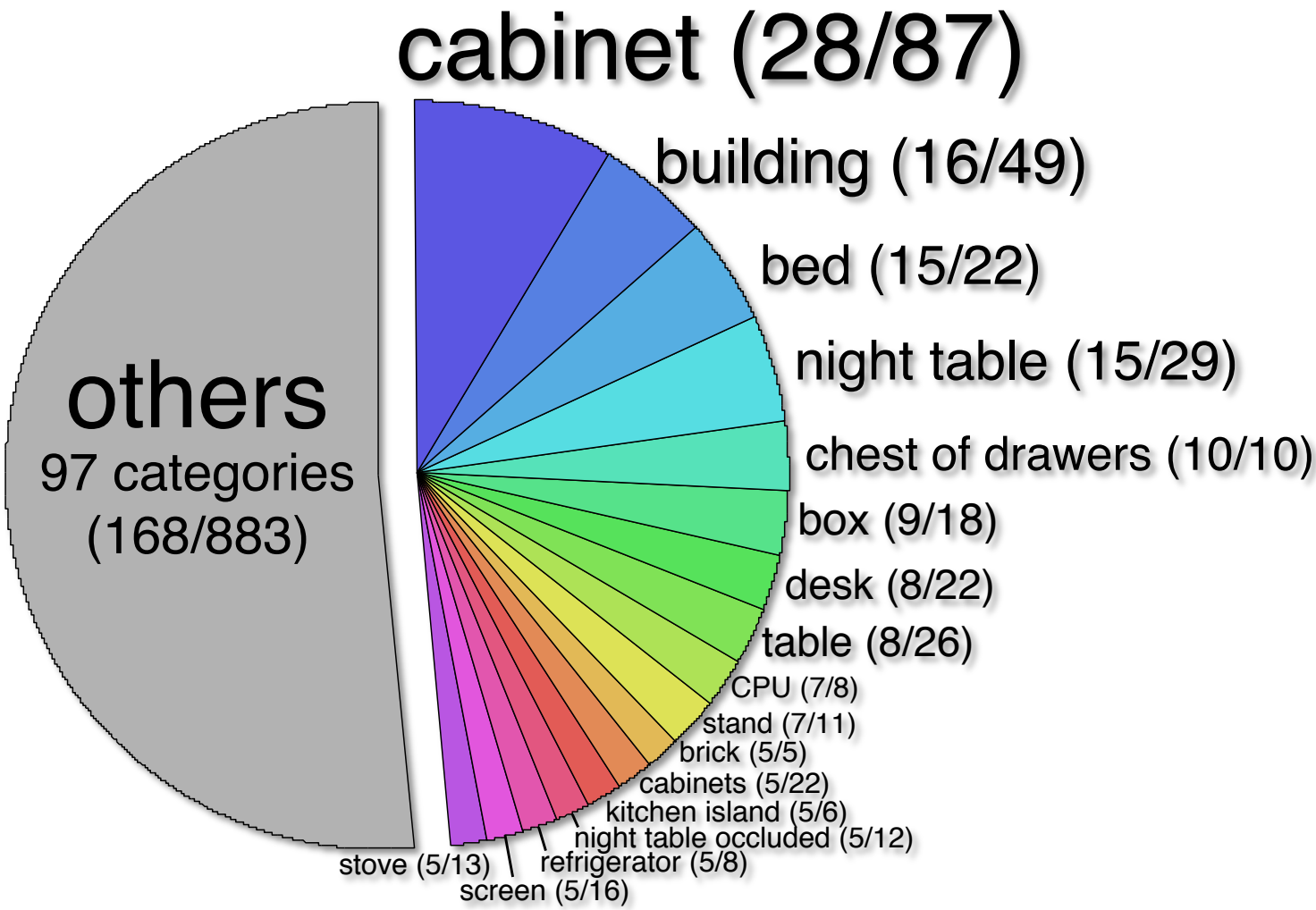
Scoring Function

$$S(I,p) = \sum_{i \in \mathcal{V}} \boxed{w_i^H} \cdot \mathrm{HOG}(I, p_i) + \sum_{ij \in \mathcal{E}} \boxed{w_{ij}^D} \cdot \mathrm{Displacement}^{\mathrm{2D}}(p_i, p_j)$$
$$+ \sum_{ij \in \mathcal{E}} \boxed{w_{ij}^E} \cdot \mathrm{Edge}(I, p_i, p_j) + \boxed{w^S} \cdot \mathrm{Shape}^{\mathrm{3D}}(p)$$

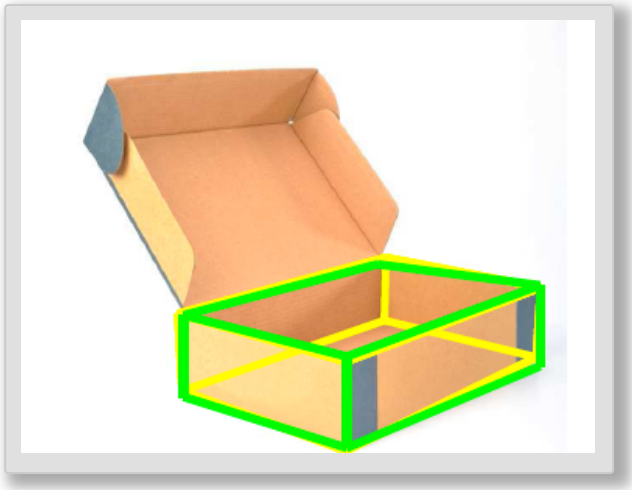


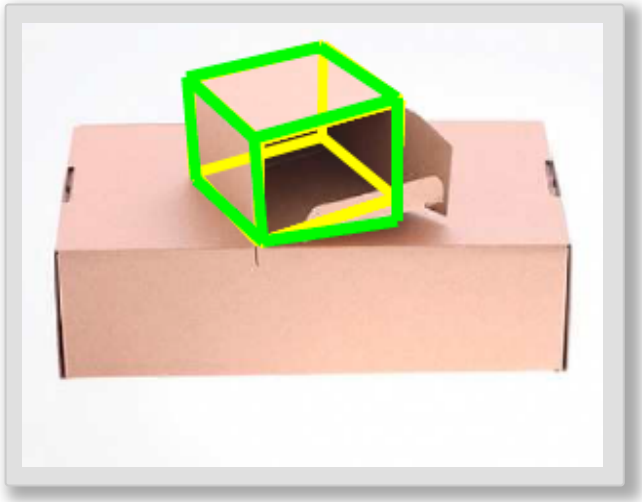
SUN Primitive Database

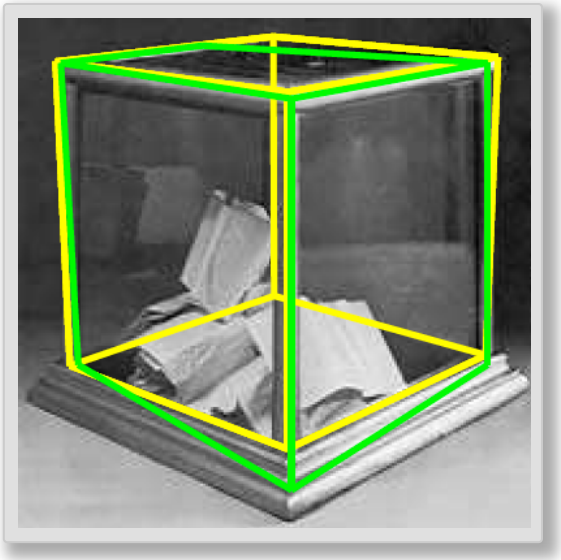
What objects are cuboids?

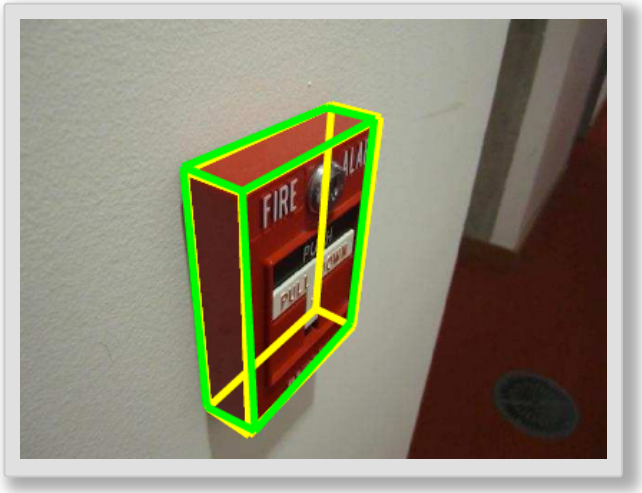


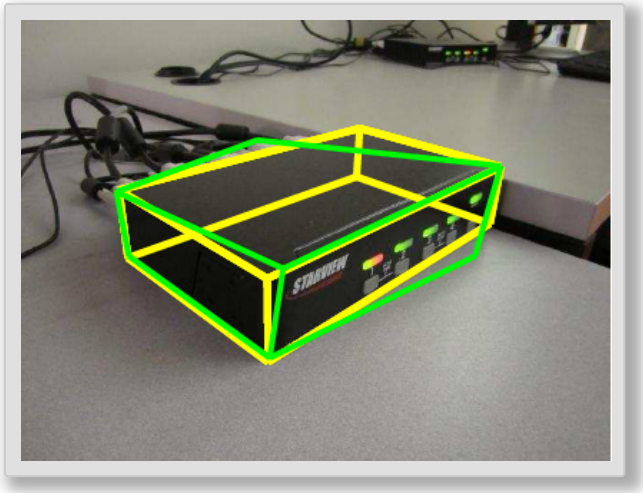
Easy Cases



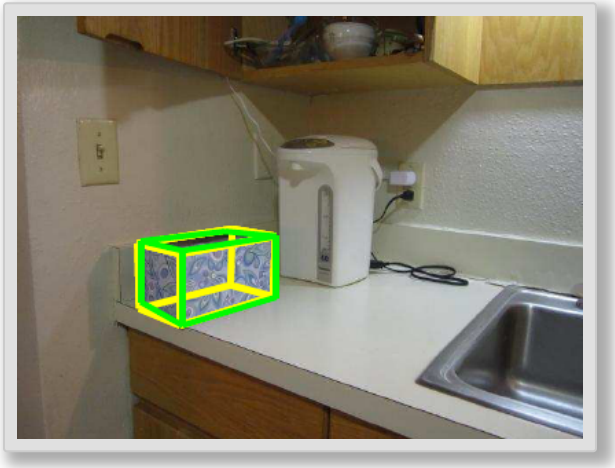






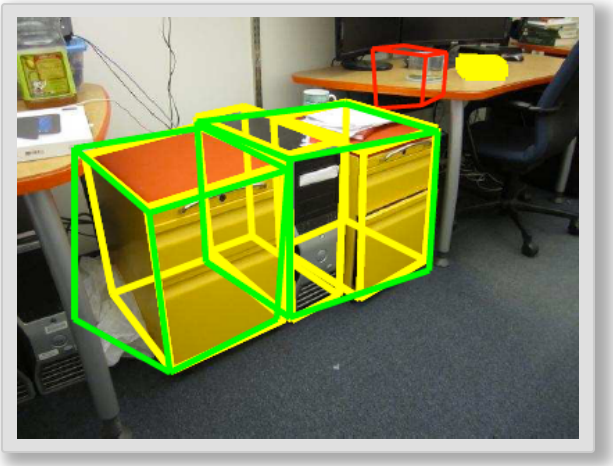
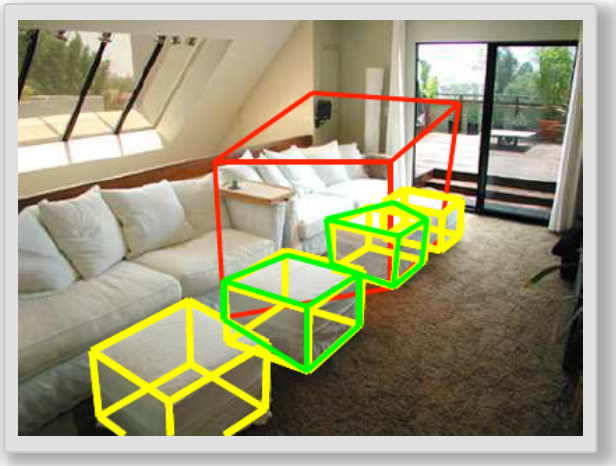
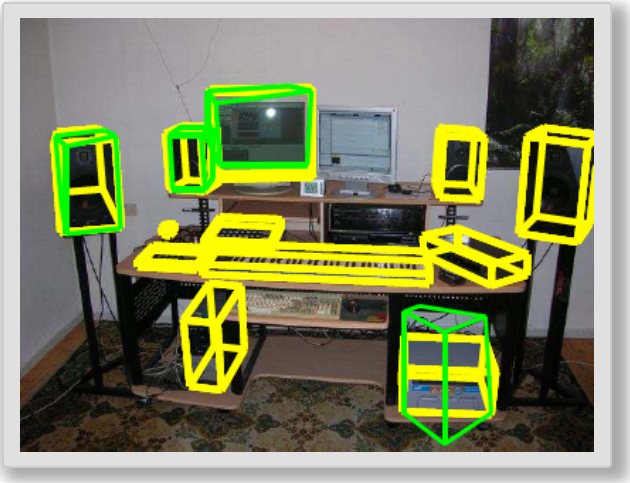
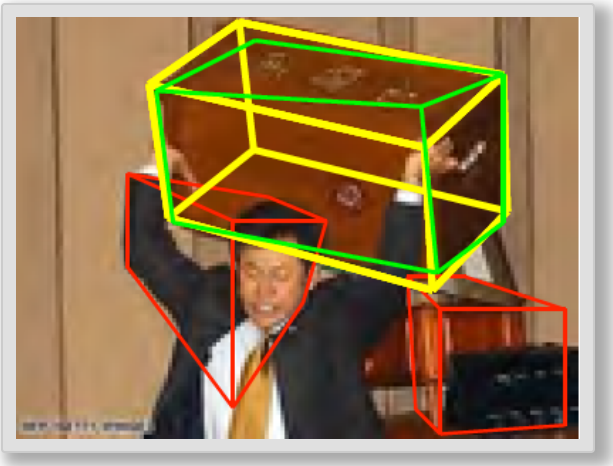
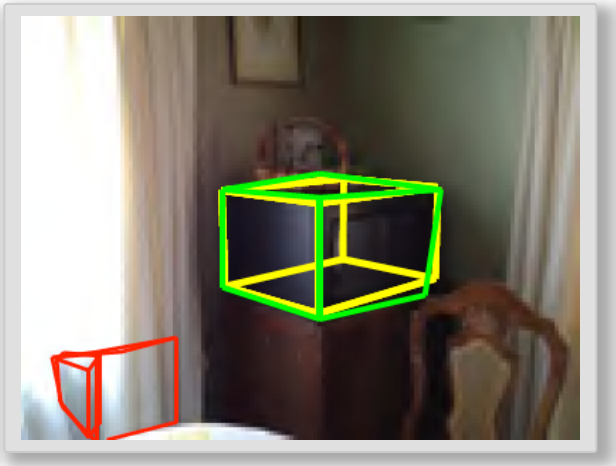
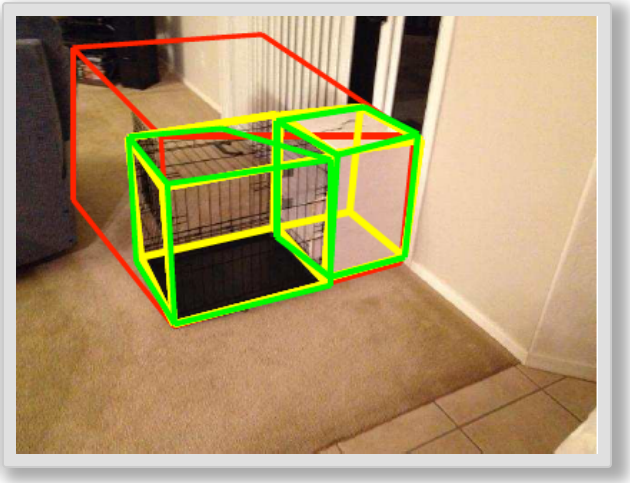


Detection Result

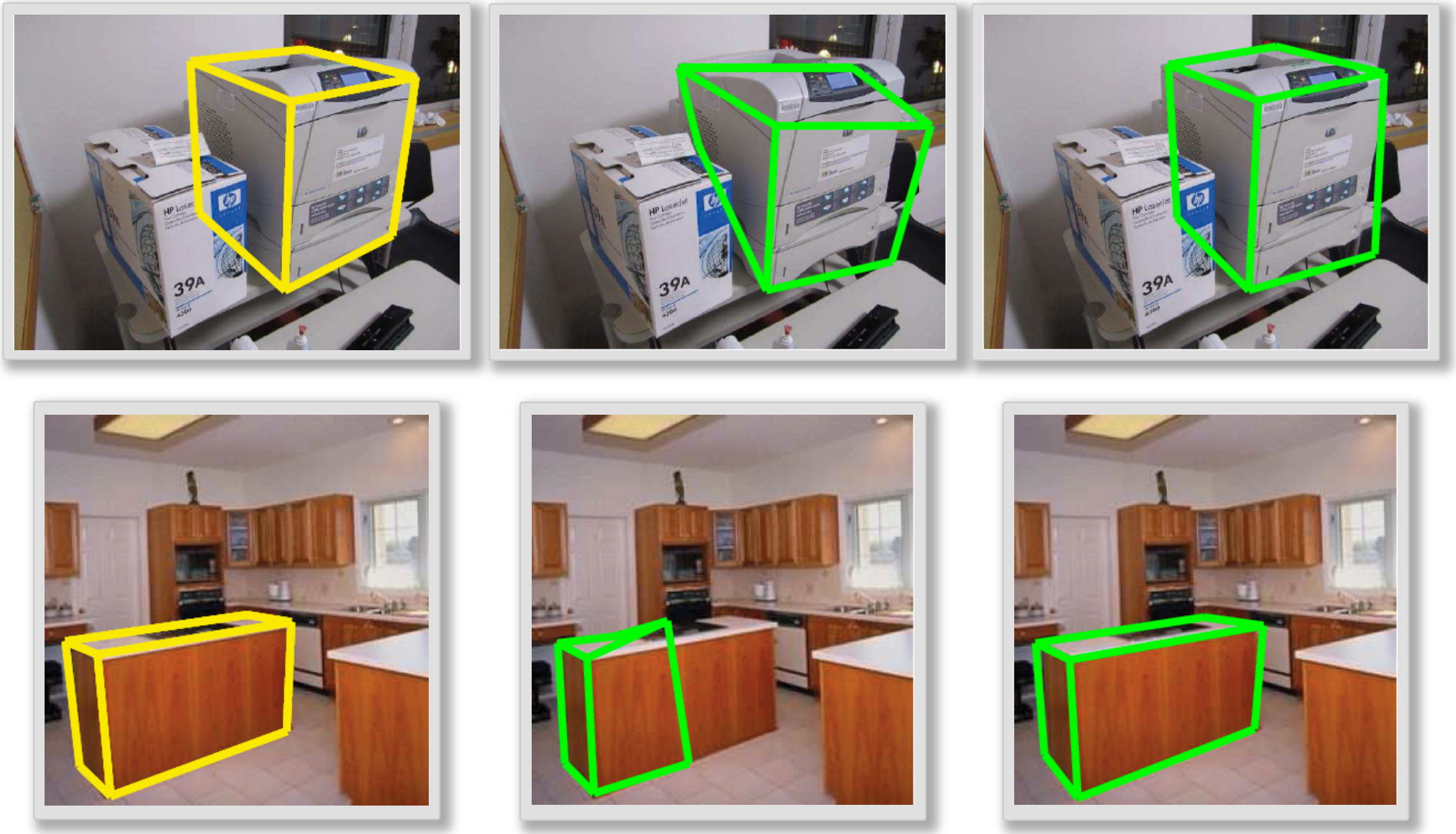




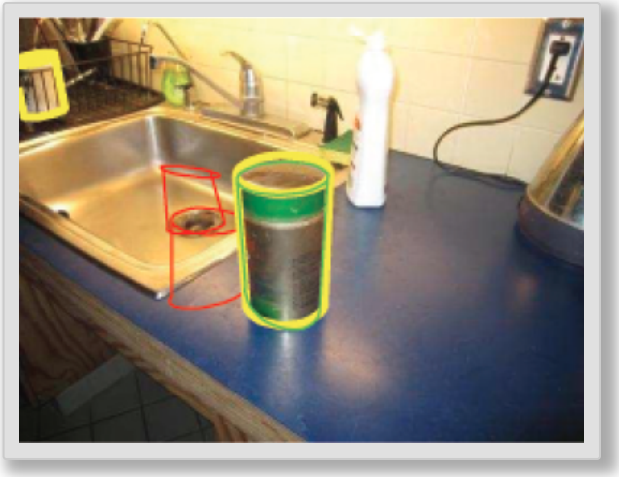

Multiple Instances

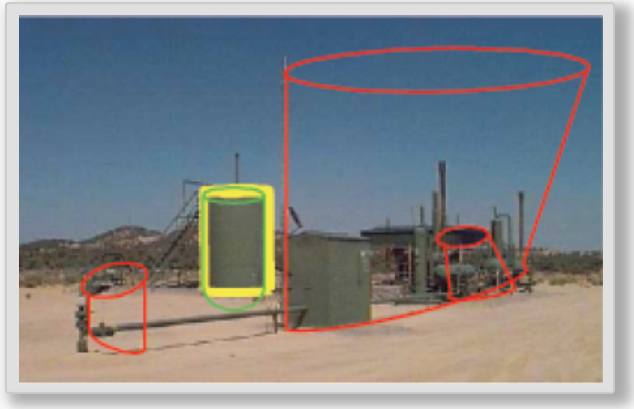







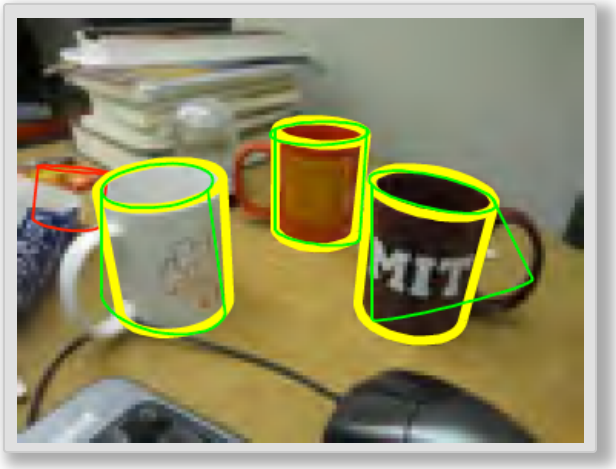
2D vs 3D

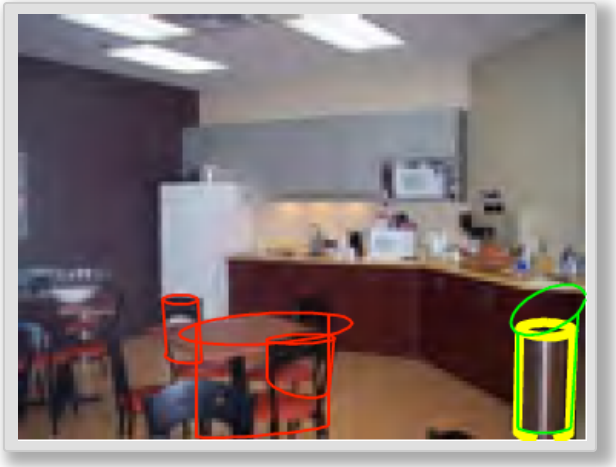


Cylinder Detection

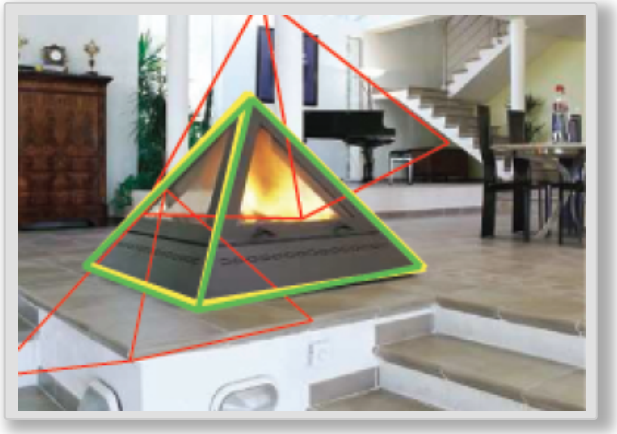


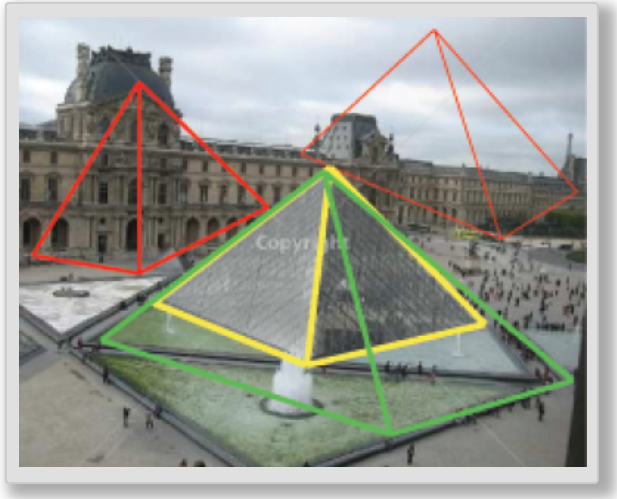
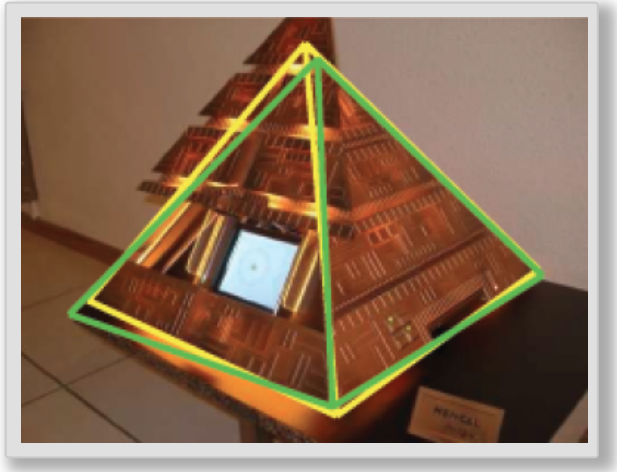






Pyramid Detection

Today

- A simple world
- A catalog of edges
- Generic view assumption
- Constrained solver
- Edge detection: Canny
- Line detection
  - RANSAC
  - Hough transformation

Image Analysis

- Let's look at gradients

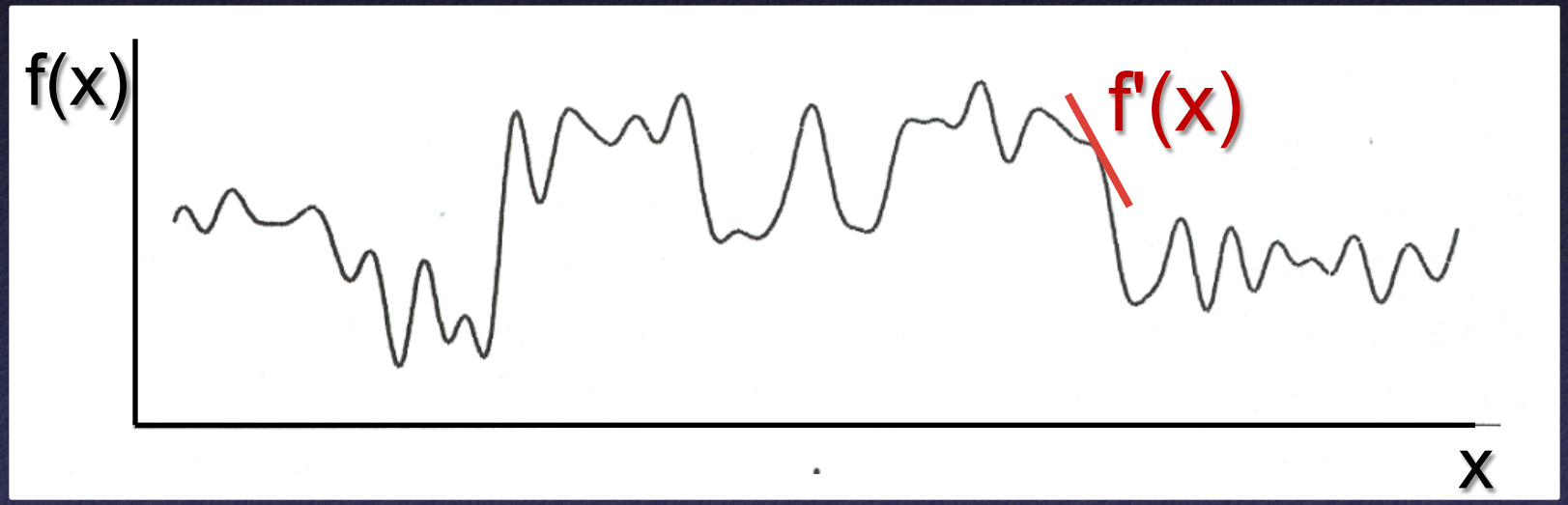


Image Gradients

For 2D function f(x,y), the partial derivative is:

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

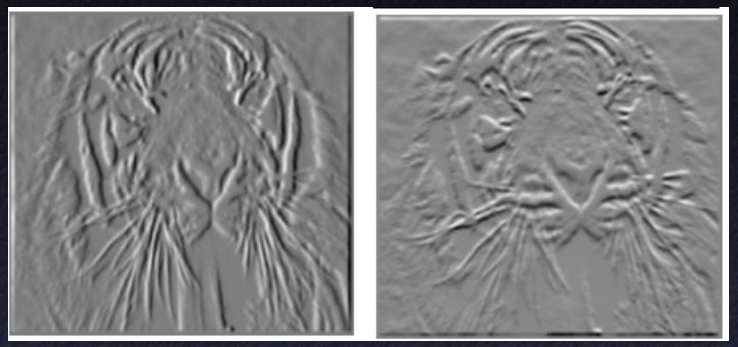
For discrete data, we can approximate using finite differences:

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

Image Gradients

The gradient of an image:

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



The magnitude of the gradient:

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

The direction of the gradient:

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

Computing Image Gradients

This is a convolution with two simple filters:

$$\frac{\partial f(x,y)}{\partial x}$$

-1	1

$$\frac{\partial f(x,y)}{\partial y}$$

-1
1

or

1
-1

Computing Image Gradients

Other common gradient filters:

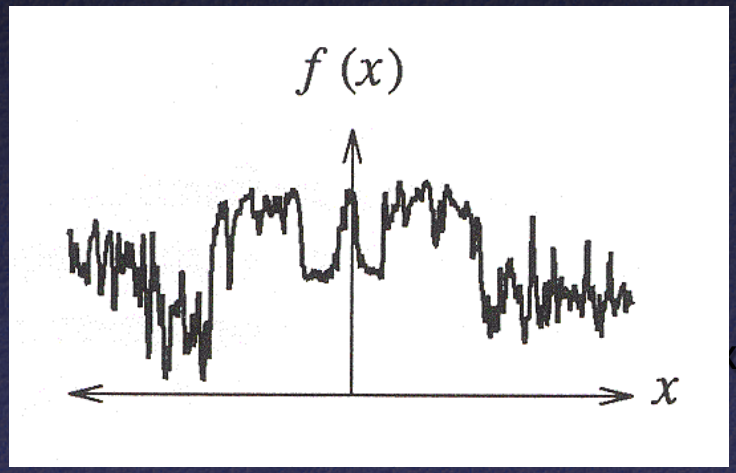
Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

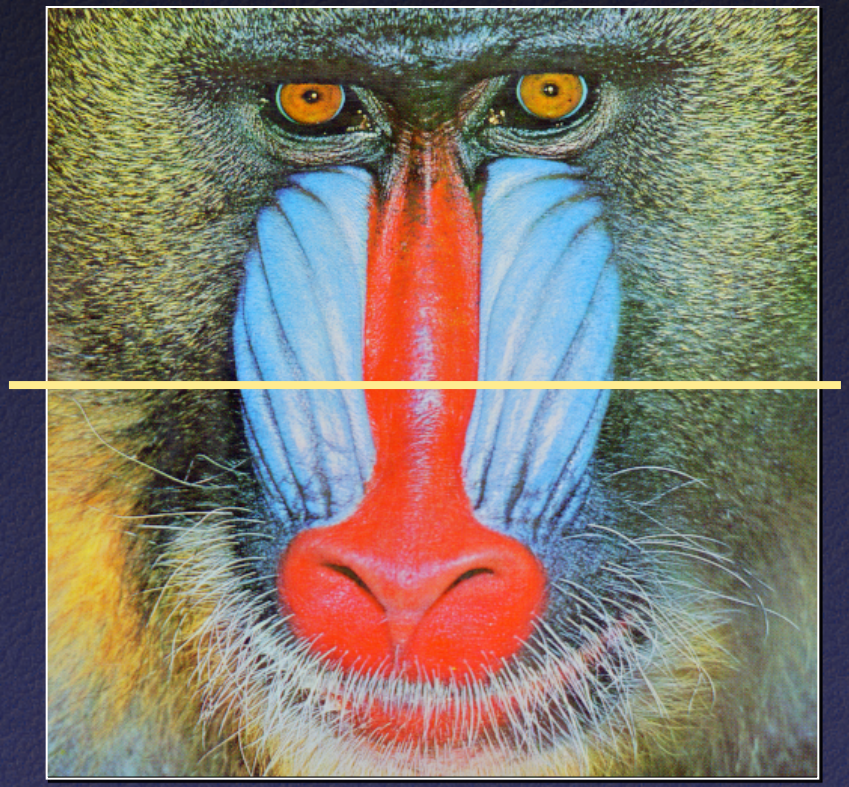
Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

Computing Image Gradients

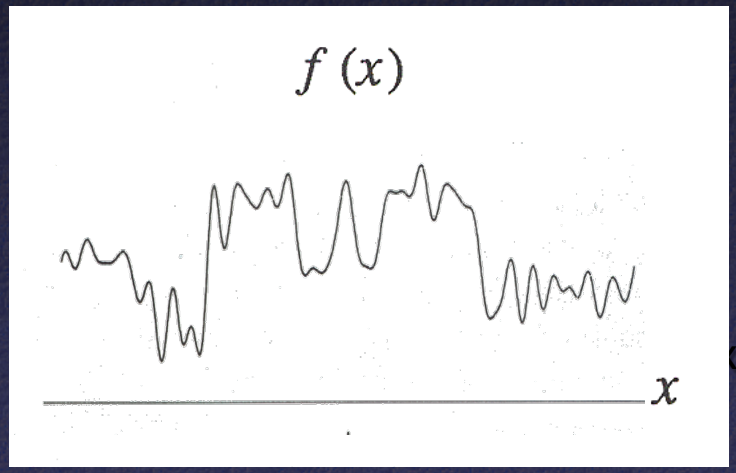
We usually limit high frequencies when computing gradient





Computing Image Gradients

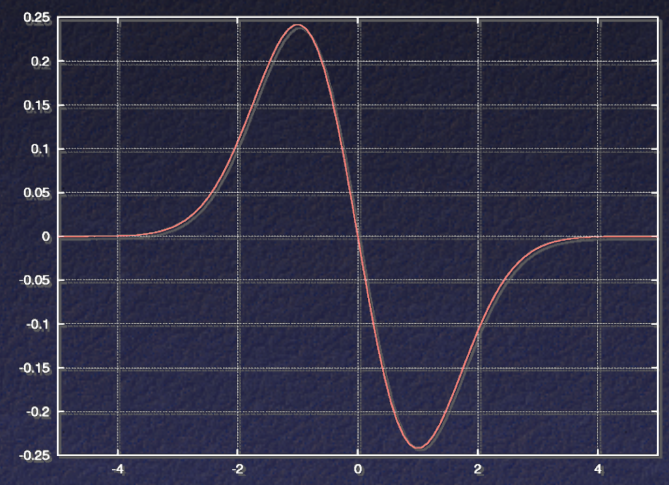
We usually limit high frequencies when computing gradient



Computing Image Gradients

Useful fact #1: differentiation "commutes" with convolution

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



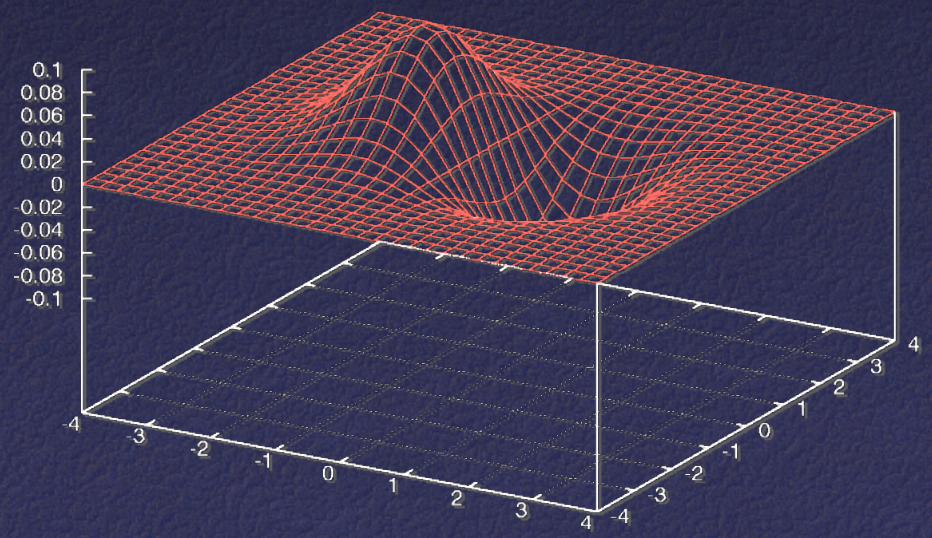
Useful fact #2: Gaussian is separable:

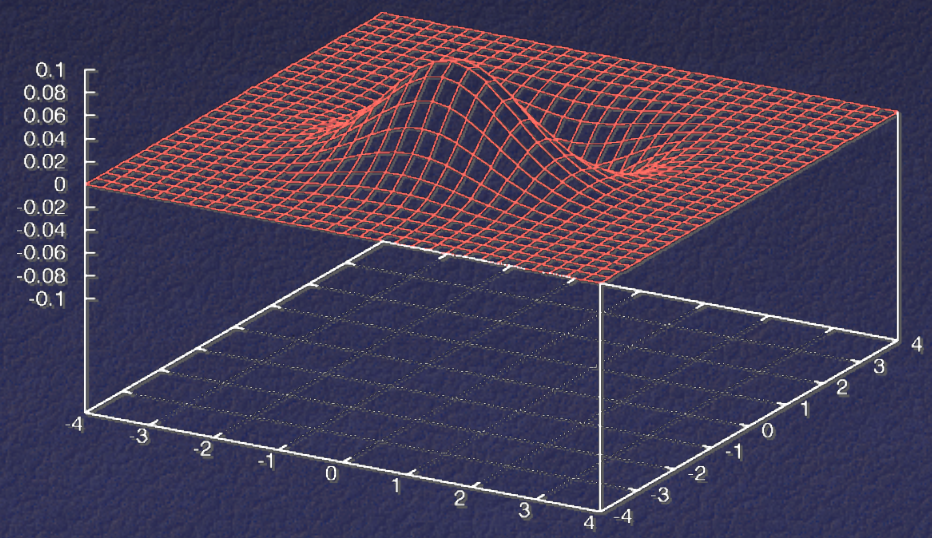
$$G_2(x, y) = G_1(x)G_1(y)$$

Computing Image Gradients

Thus, combine smoothing with gradient computation:

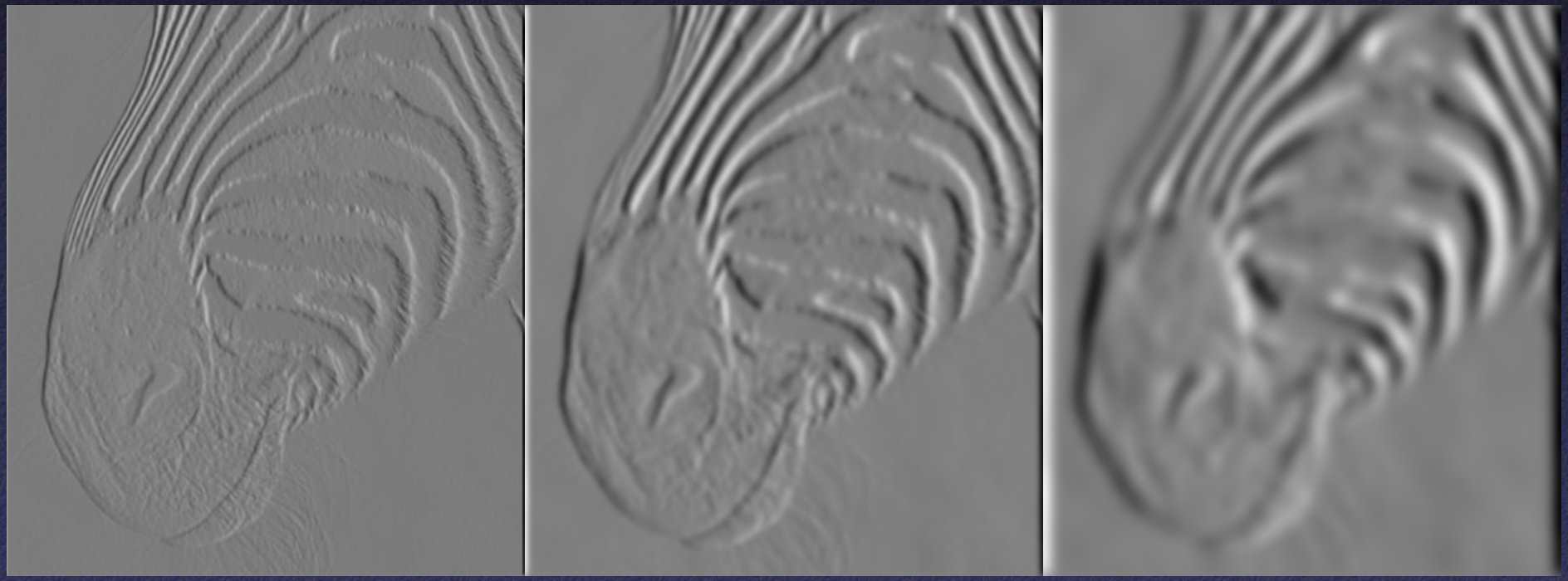
$$\nabla\left(f(x,y) * G_2(x,y)\right) = \begin{bmatrix} f(x,y) * \left(G_1'(x)G_1(y)\right) \\ f(x,y) * \left(G_1(x)G_1'(y)\right) \end{bmatrix} = \begin{bmatrix} f(x,y) * G_1'(x) * G_1(y) \\ f(x,y) * G_1(x) * G_1'(y) \end{bmatrix}$$





Computing Image Gradients

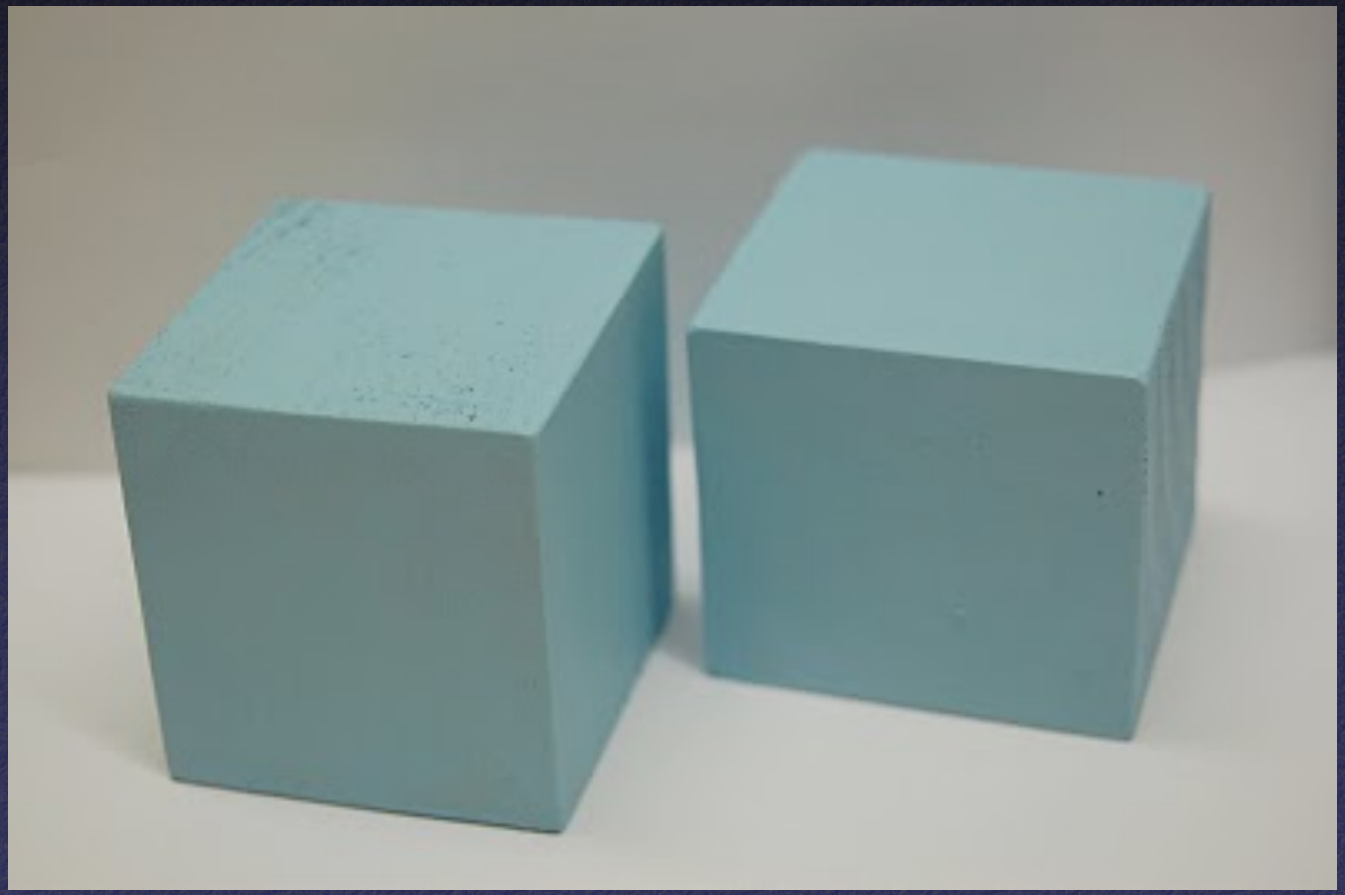
Can use different sigma to find gradients at different “scales”



1 pixel 3 pixels 7 pixels

Gradient Analysis

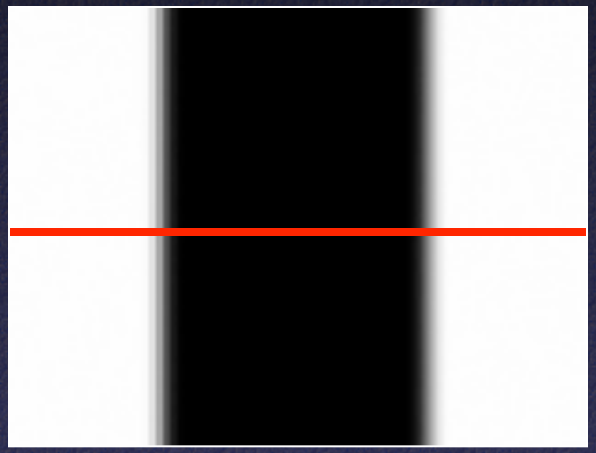
How are image gradients useful?



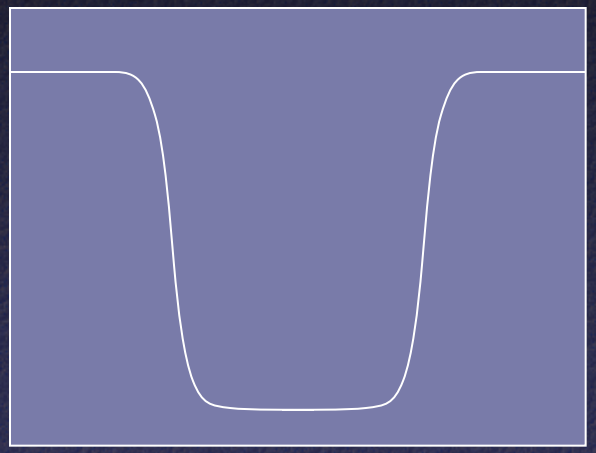
Edges

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

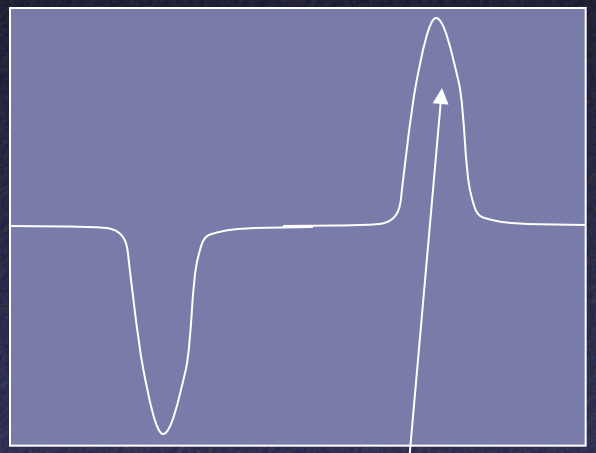
image



intensity function
(along horizontal scanline)

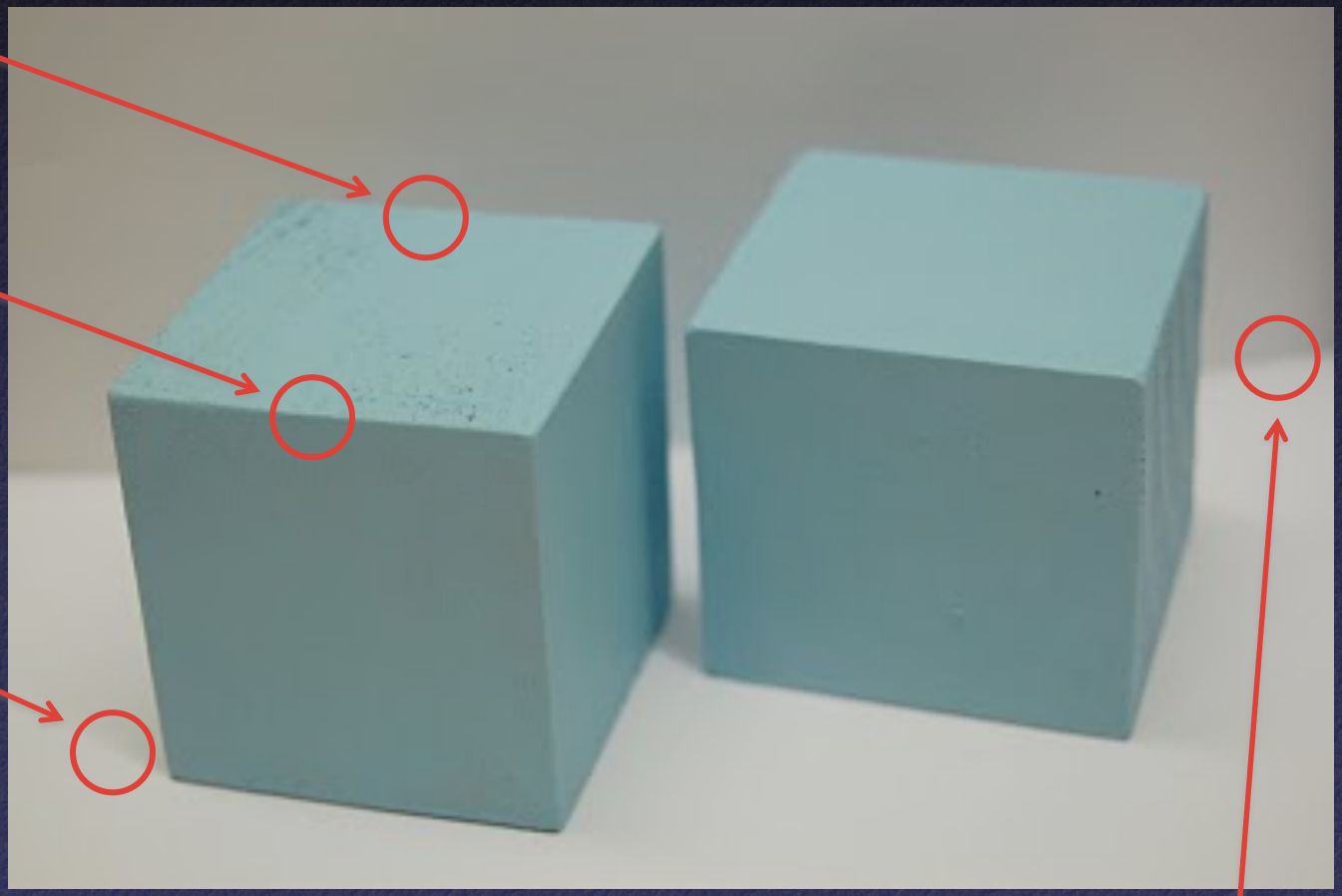


first derivative



edges correspond to
extrema of derivative

Edges


Silhouette Boundary
Convex Crease
Shadow Boundary
Concave Crease

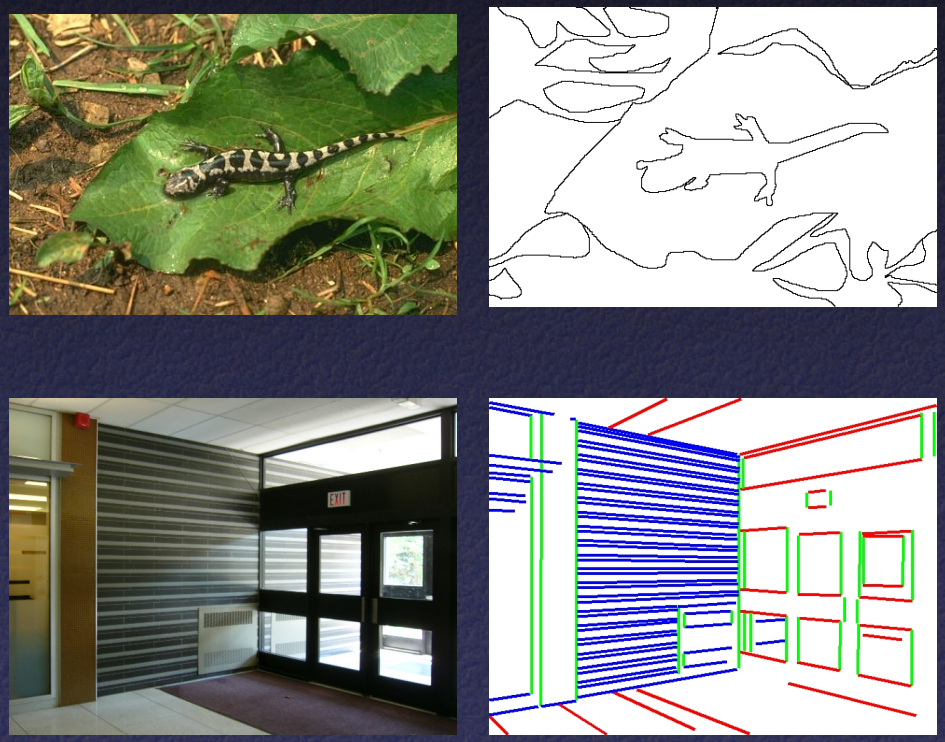
Edges

Edges

Edge Detection

Useful for many applications in vision

- Segmentation
- Camera pose estimation
- 3D reconstruction
- Object classification
- Object recognition
- etc.



Canny Edge Detector

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression
4. Hysteresis thresholding

Canny Edge Detector

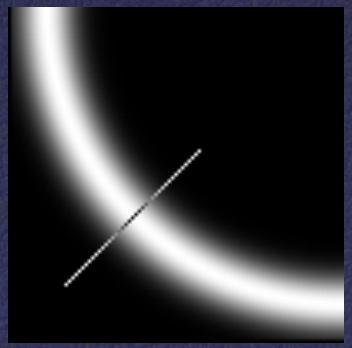
Original Image

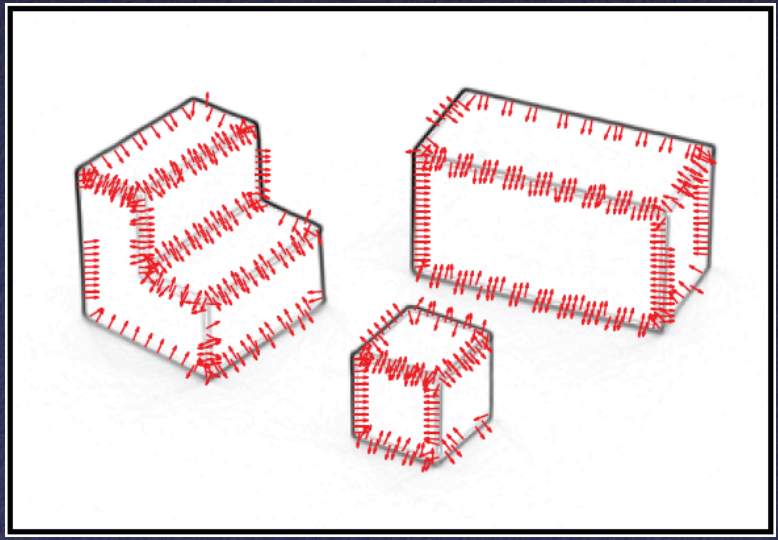
Smoothed Gradient Magnitude

Canny Edge Detector

Nonmaximum suppression

- Eliminate all but local maxima in *gradient magnitude* (sqrt of sum of squares of *x* and *y* components)
- At each pixel p look along *direction* of gradient: if either neighbor is bigger, set p to zero
- In practice, quantize direction to horizontal, vertical, and two diagonals
- Result: “thinned edge image”





Canny Edge Detector

Smoothed gradient magnitude

Non-maximum suppression

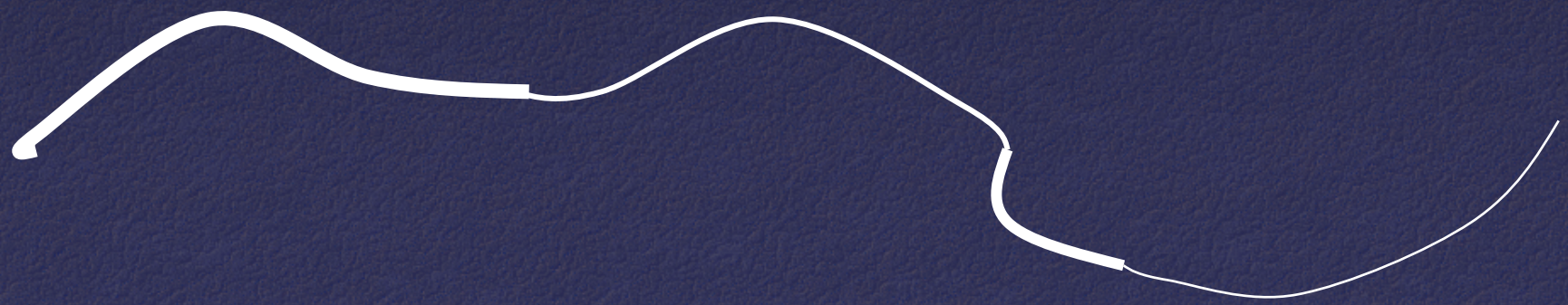
Canny Edge Detector

Final stage: thresholding

Simplest: use a single threshold

Better: use two thresholds

- Find chains of touching edge pixels, all $\geq \tau_{low}$
- Each chain must contain at least one pixel $\geq \tau_{high}$
- Helps eliminate dropouts in chains, without being too susceptible to noise
- "Thresholding with hysteresis"



Canny Edge Detector

Non-maximum suppression

Canny edges

Canny Edge Detector

Original Image

Canny edges

Summary of Canny Edge Detector

1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
   - Thin wide “ridges” down to single pixel width
4. Hysteresis thresholding:
   - Define two thresholds: low and high
   - Use the high threshold to start edge curves and the low threshold to continue them

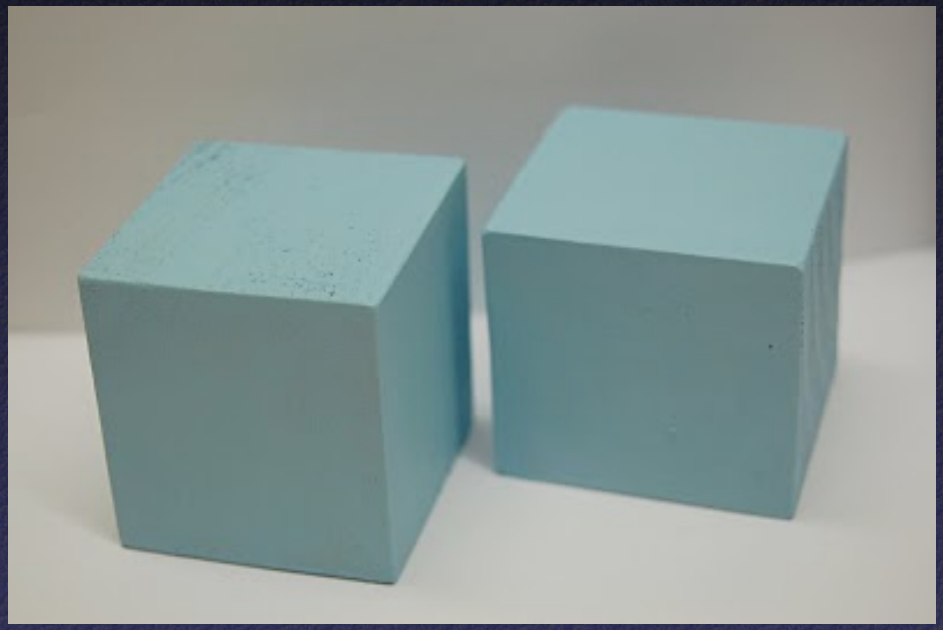
Today

- A simple world
- A catalog of edges
- Generic view assumption
- Constrained solver
- Edge detection: Canny
- Line detection
  - RANSAC
  - Hough transformation

Goal

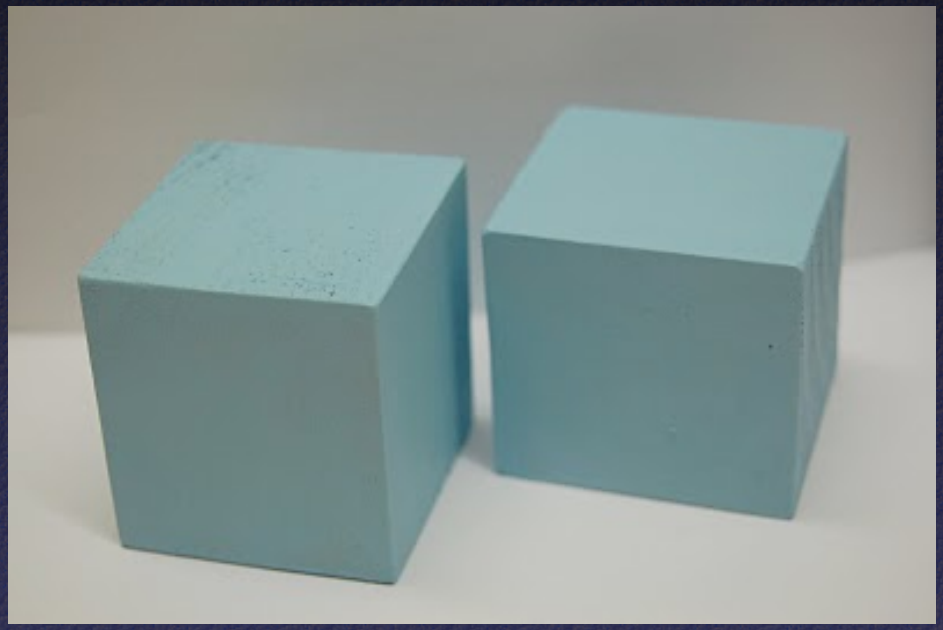
Extract “structural features” from an image

- Non-accidental properties



Goal

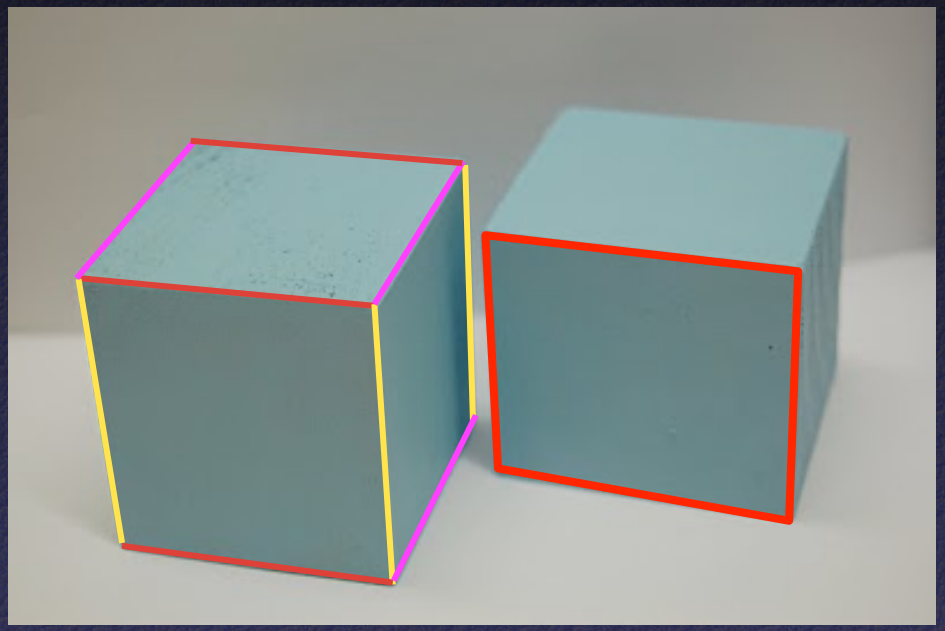
What types of “structure” are in this image?



Goal

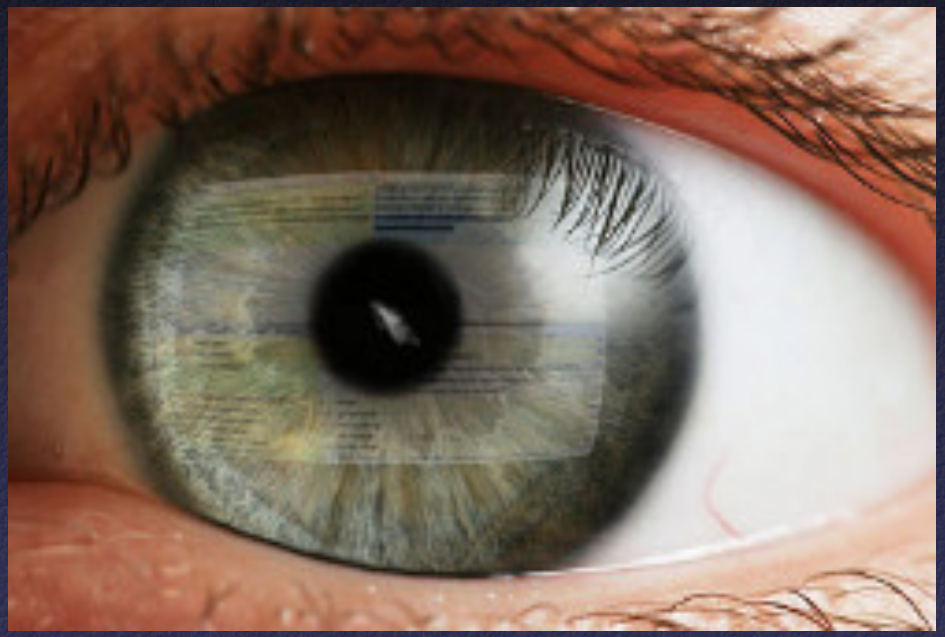
What types of “structure” are in this image?

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



Goal

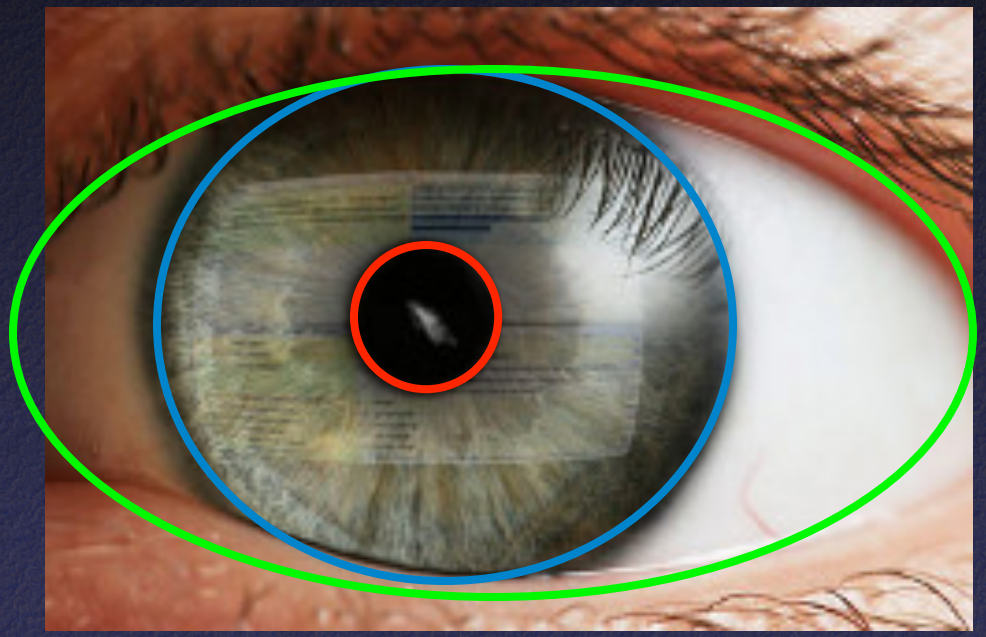
What types of “structure” are in this image?



Goal

What types of “structure” are in this image?

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



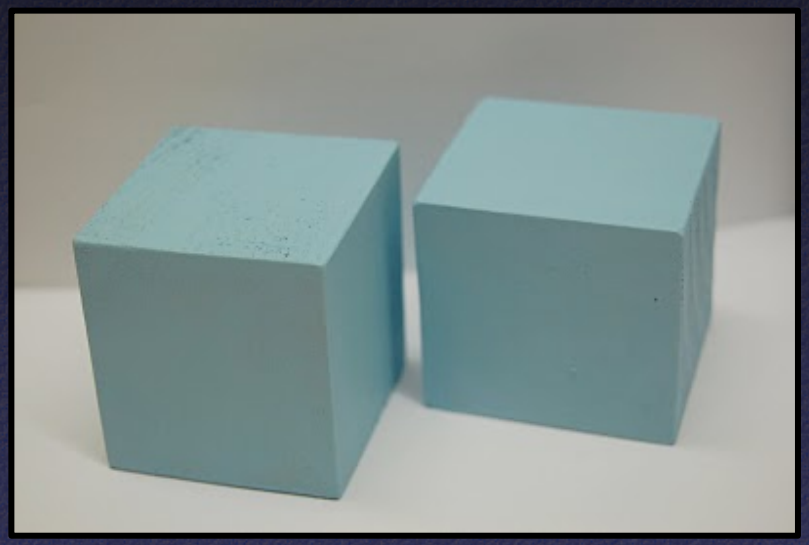
Structure Detection

Algorithms for “structure detection”

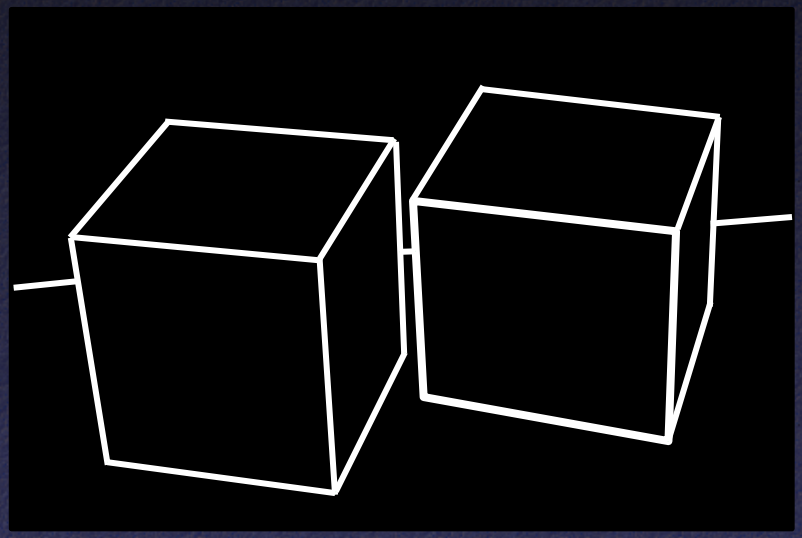
- Line detection
- Circle detection
- Symmetry detection

Line Detection

Let's first consider how to detect lines



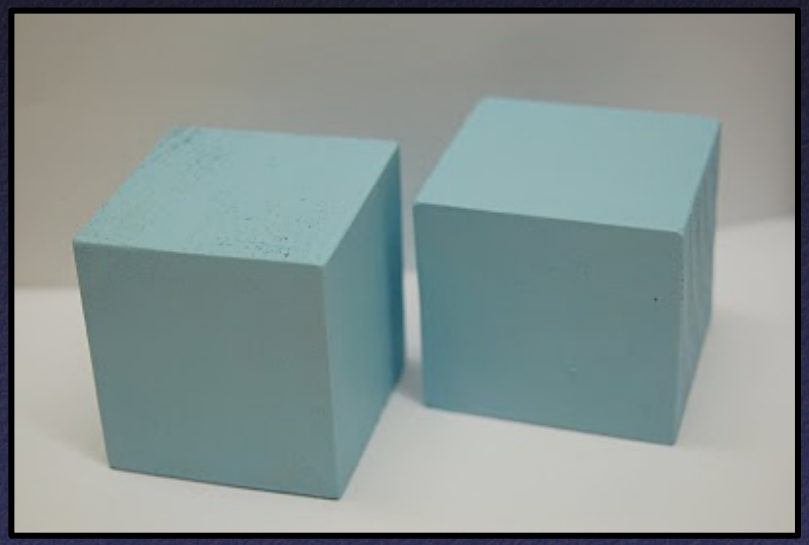
Input



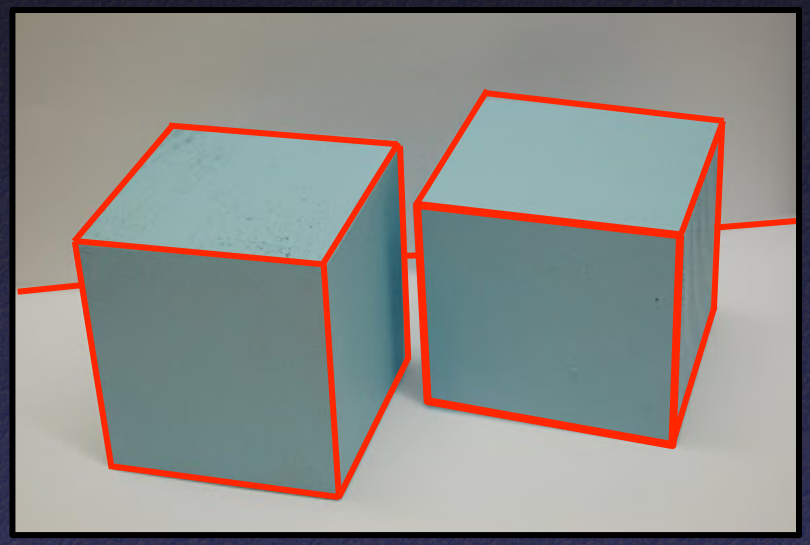
Output

Line Detection

Let's first consider how to detect lines



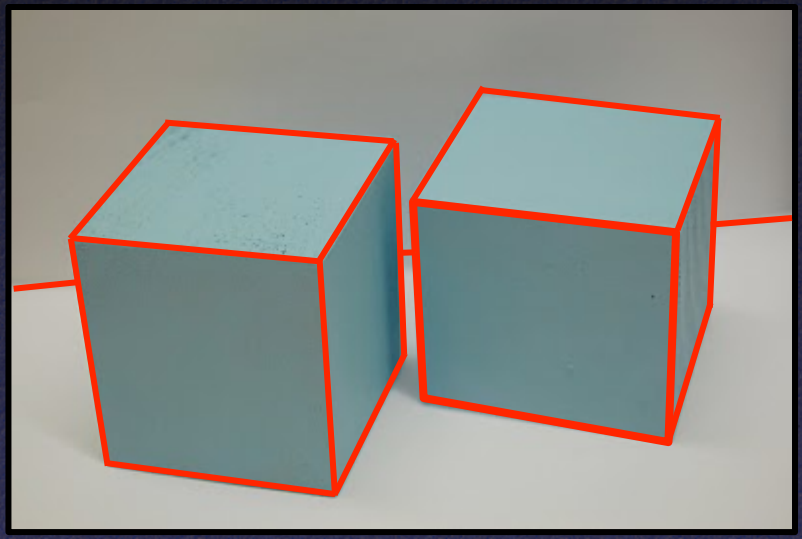
Input



Overlay

Line Detection

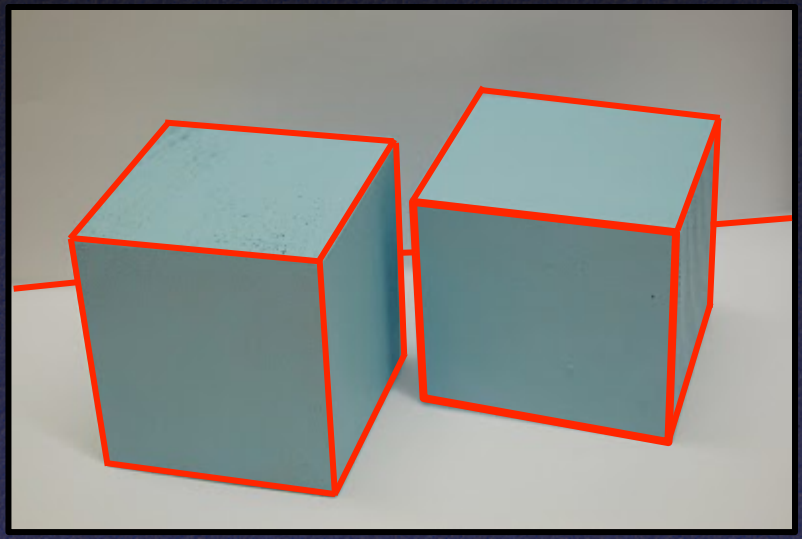
Desirable properties of a line detection algorithm?



Line Detection

Desirable properties of a line detection algorithm:

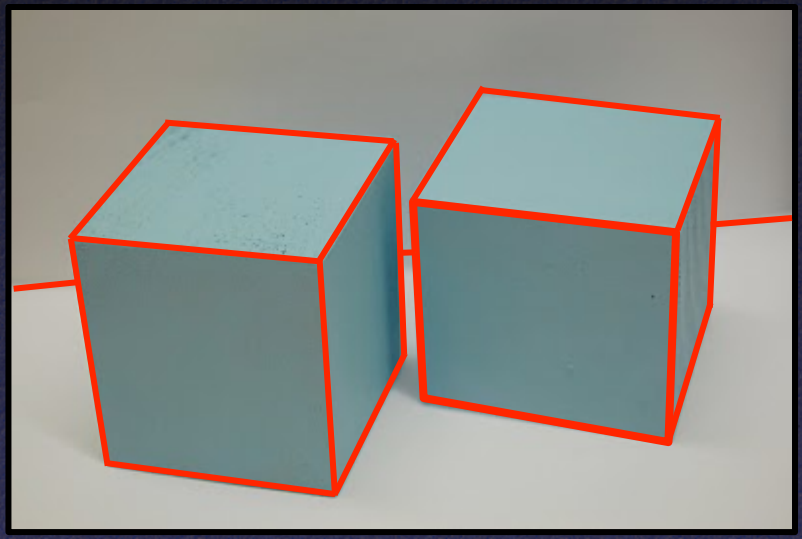
- Straight, long lines only
- Few missed or extra lines
- Provides confidence of prediction for each pixel
- Robust to differences in occlusion, noise, scale, rotation, translation, slight non-straightness, brightness, etc.
- Efficient computation



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.

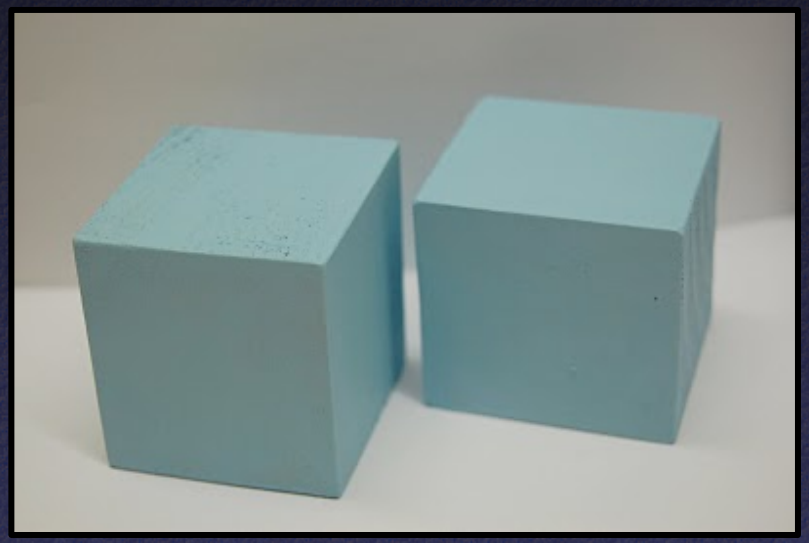
Line Detection

Applications:

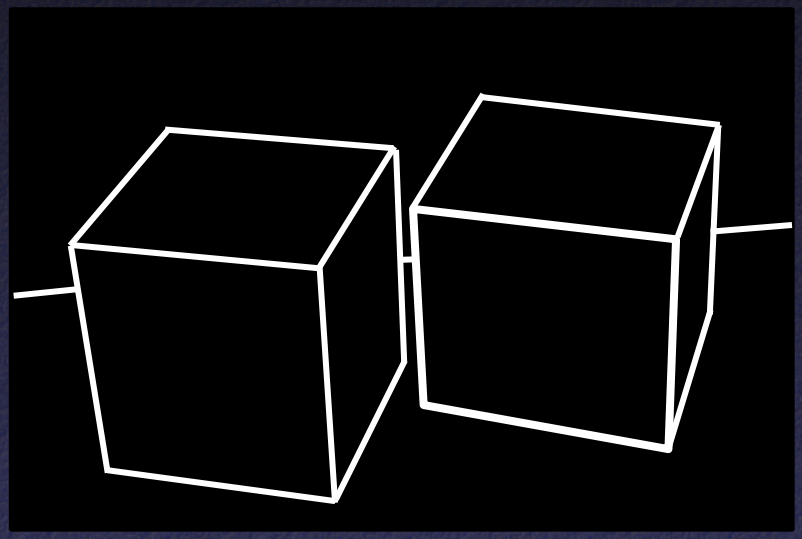
- Removing radial distortion
- Camera pose estimation
- Segmentation
- Scene classification
- Object detection
- etc.

Line Detection

Please propose a line detection algorithm



Input

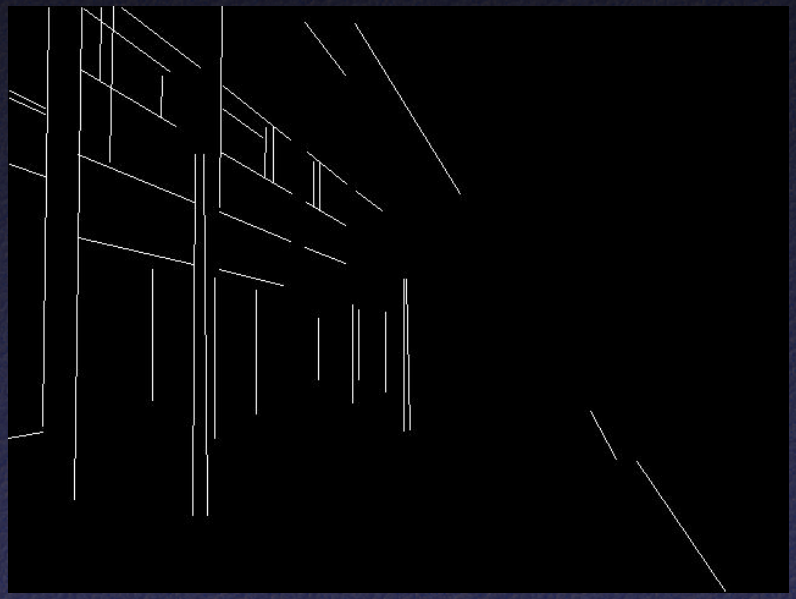


Output

Line Detection

OK, but what about this harder example?

Input

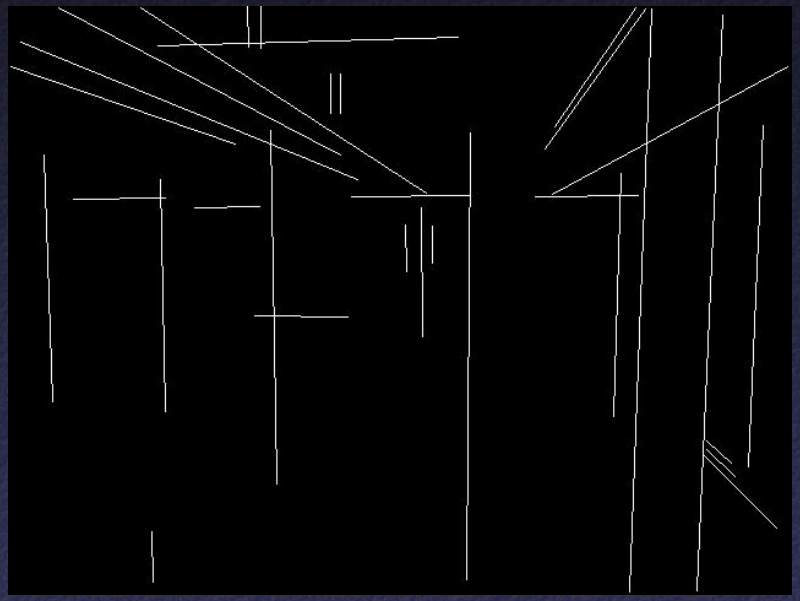


Output

Line Detection

This one?

Input



Output

Today

- A simple world
- A catalog of edges
- Generic view assumption
- Constrained solver
- Edge detection: Canny
- Line detection
  - RANSAC
  - Hough transformation

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:

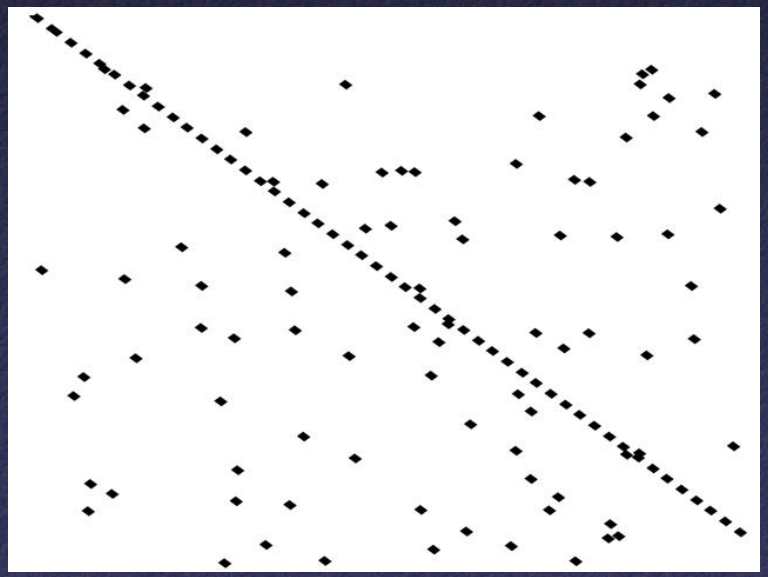
- Compute gradient direction N and magnitude G

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:

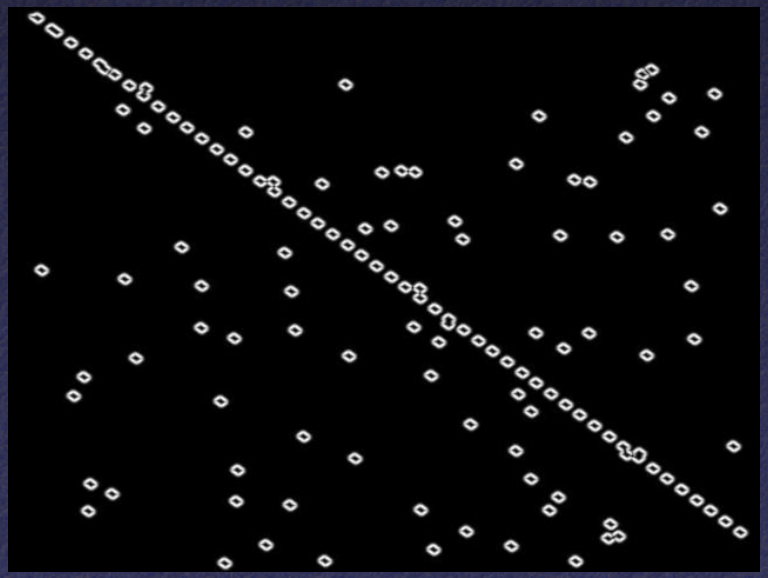
- Compute gradient direction N and magnitude G

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:

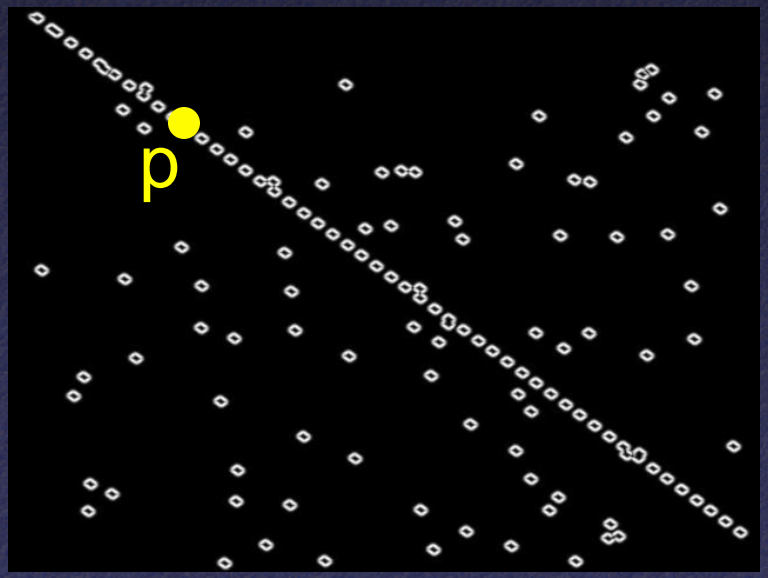
- Compute gradient direction N and magnitude G

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:

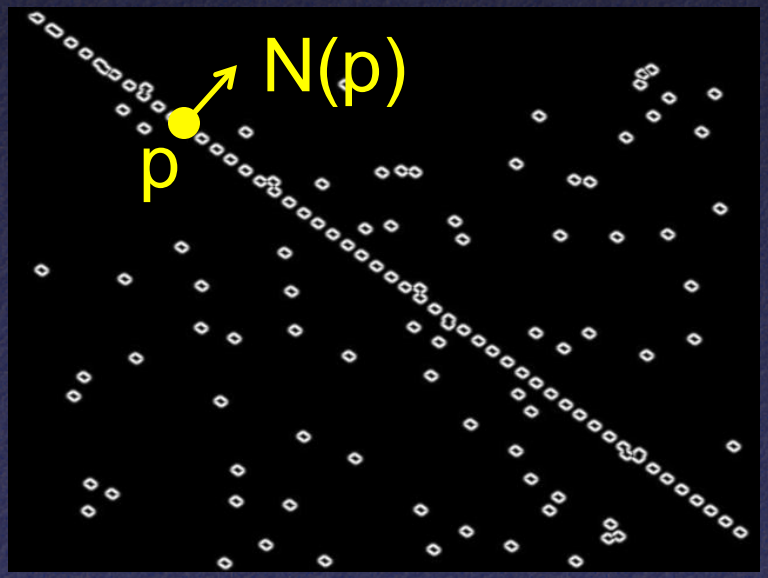
- Compute gradient direction N and magnitude G

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:

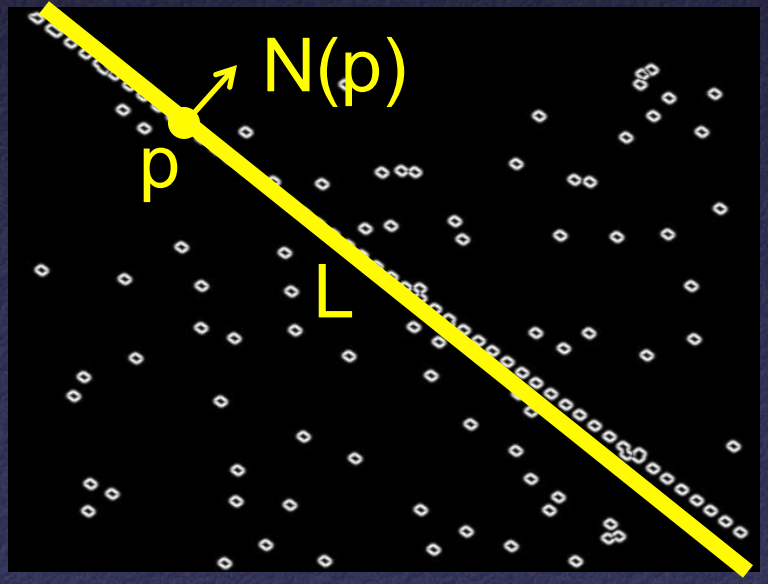
- Compute gradient direction N and magnitude G

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

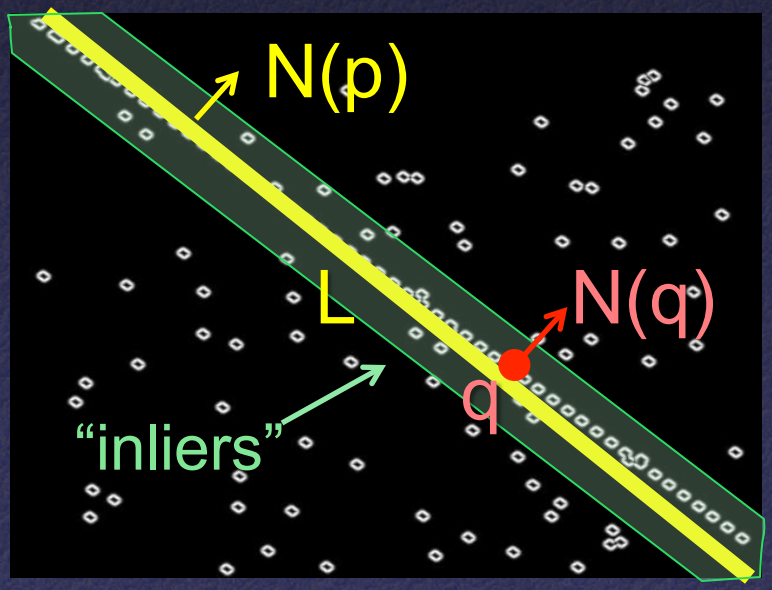
Iterate:

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

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

At end:

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



Compute support

RANSAC for Line Detection

At beginning:

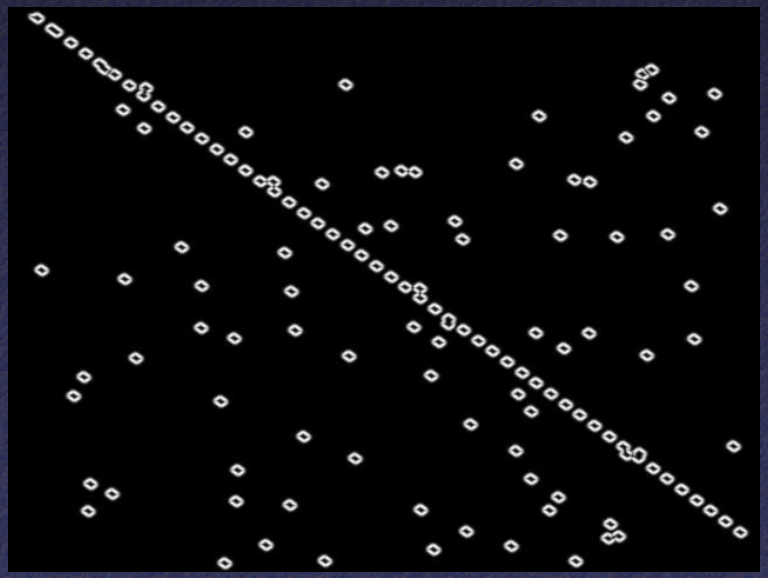
- Compute gradient direction N and magnitude G

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:

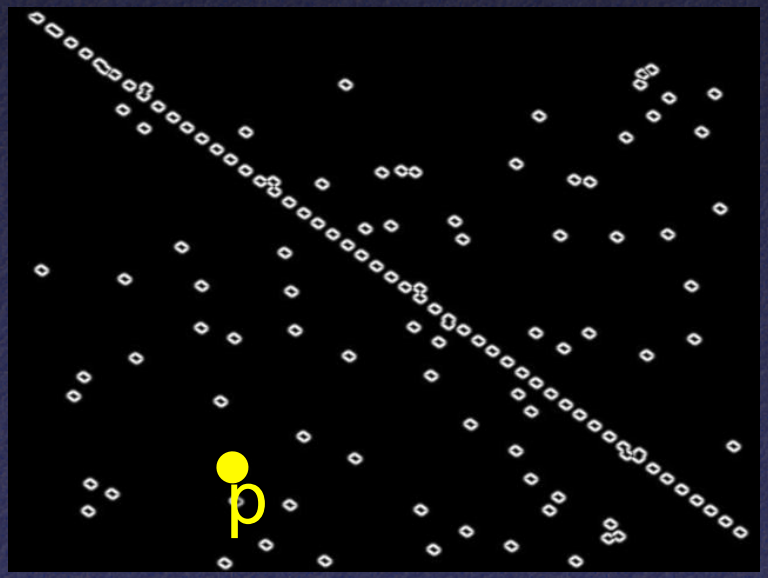
- Compute gradient direction N and magnitude G

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:

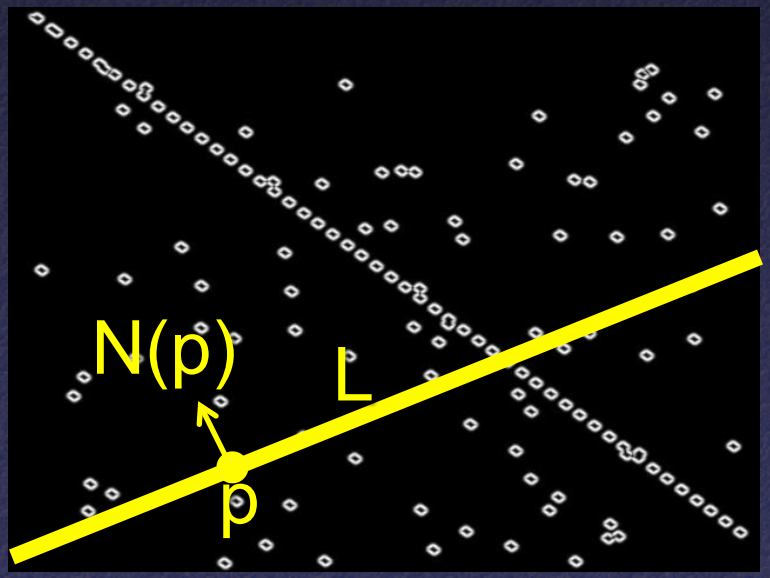
- Compute gradient direction N and magnitude G

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

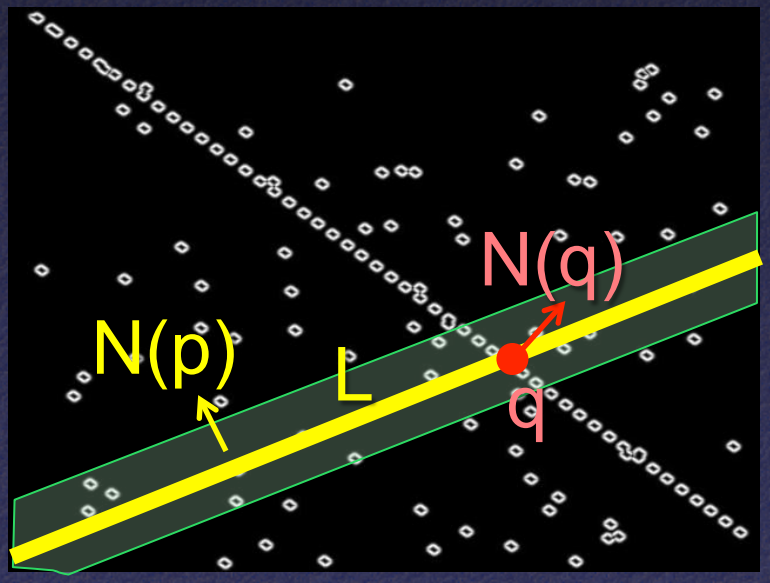
Iterate:

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

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

At end:

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



Compute support

RANSAC for Line Detection

At beginning:

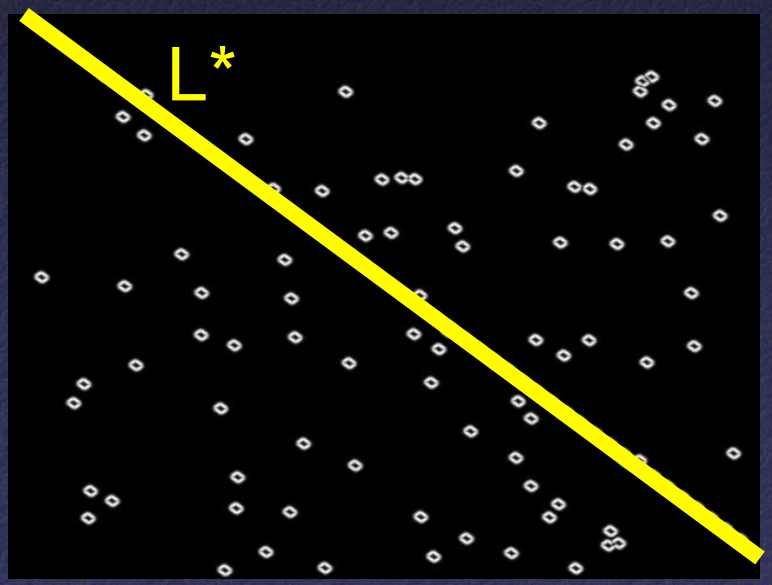
- Compute gradient direction N and magnitude G

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

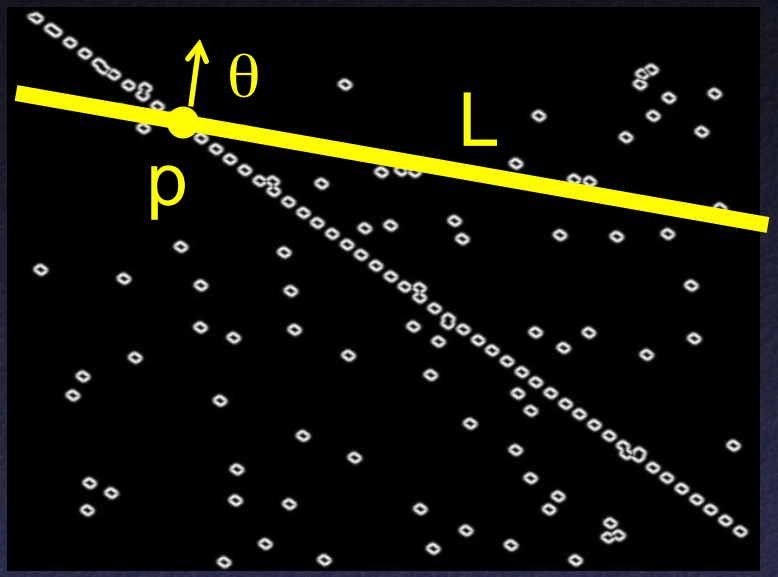
Many possible variants:

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



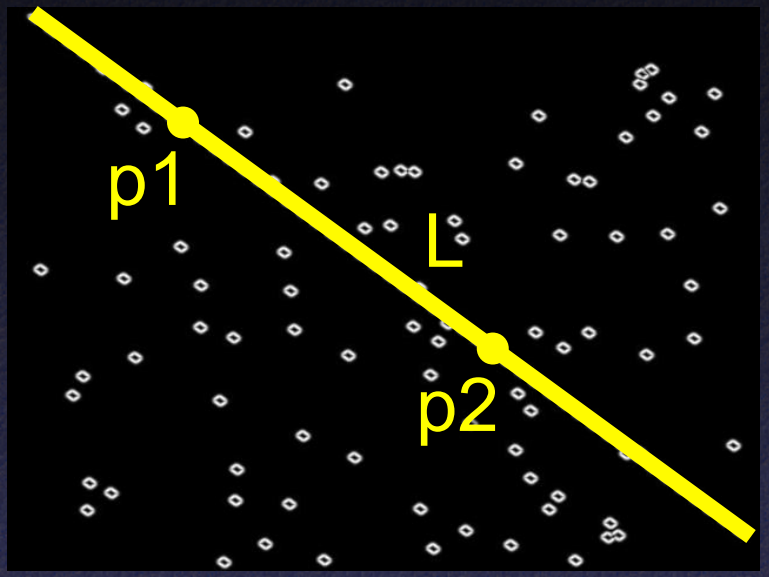
Line through point with angle

- How compute "support" for L?
  - $\sum G(q)\,|N(p) \cdot N(q)|$
  - 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.



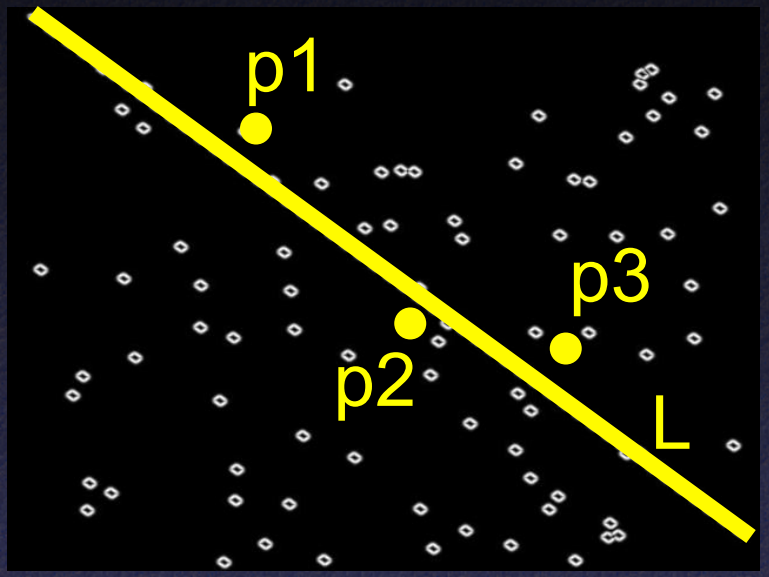
Line through two points

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

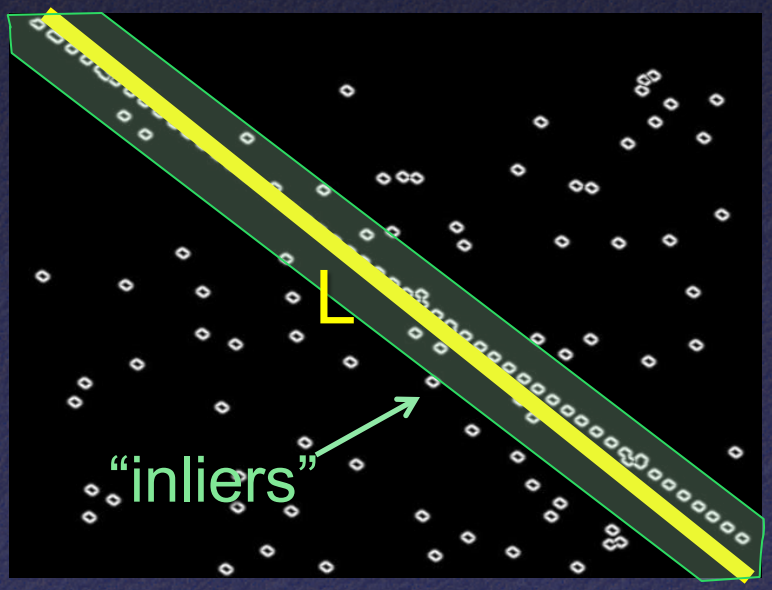
RANSAC for Line Detection

Many possible variants:

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

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

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



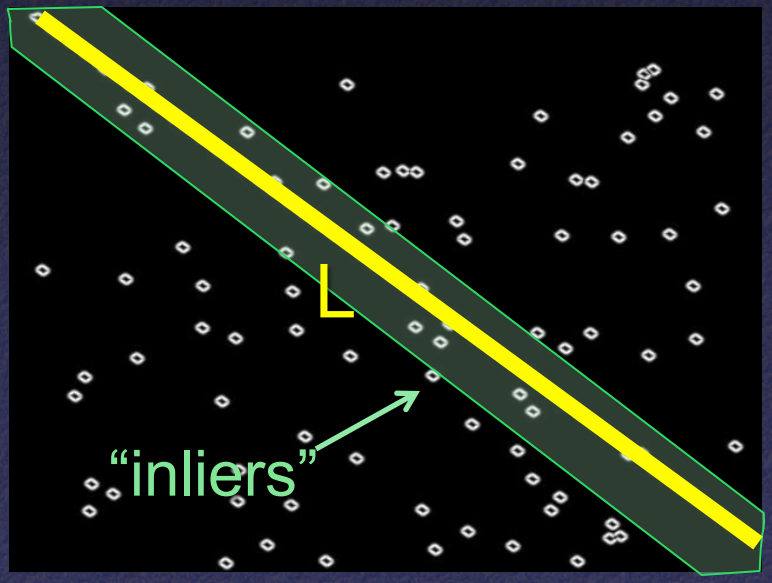
RANSAC for Line Detection

Many possible variants:

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

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

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



RANSAC for Line Detection

At beginning:

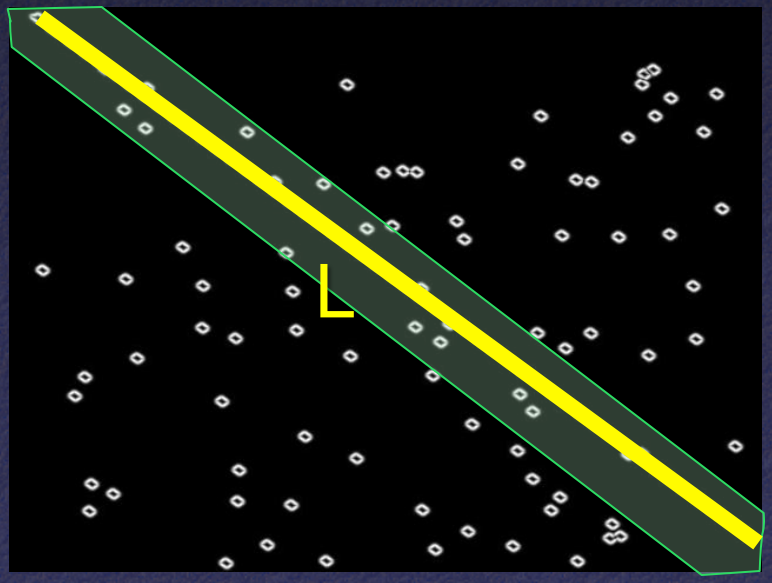
- Compute gradient direction N and magnitude G

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

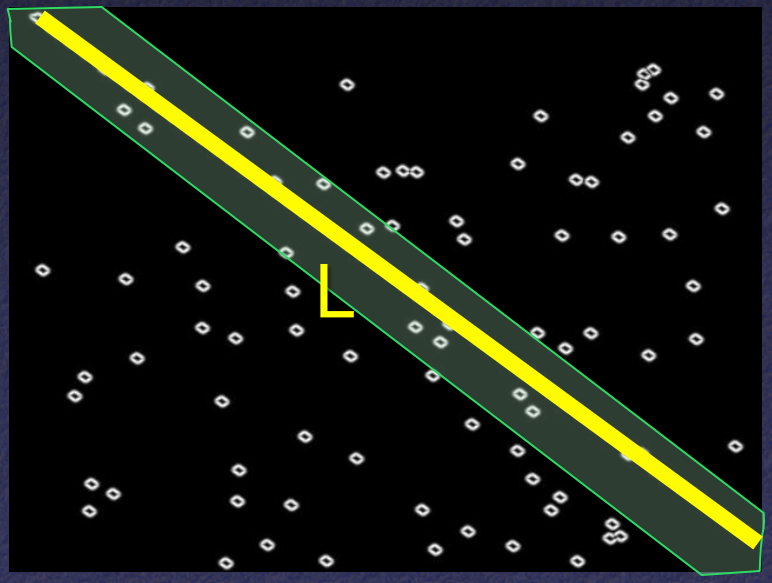
Iterate:

- 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"

How many iterations?
What is running time?

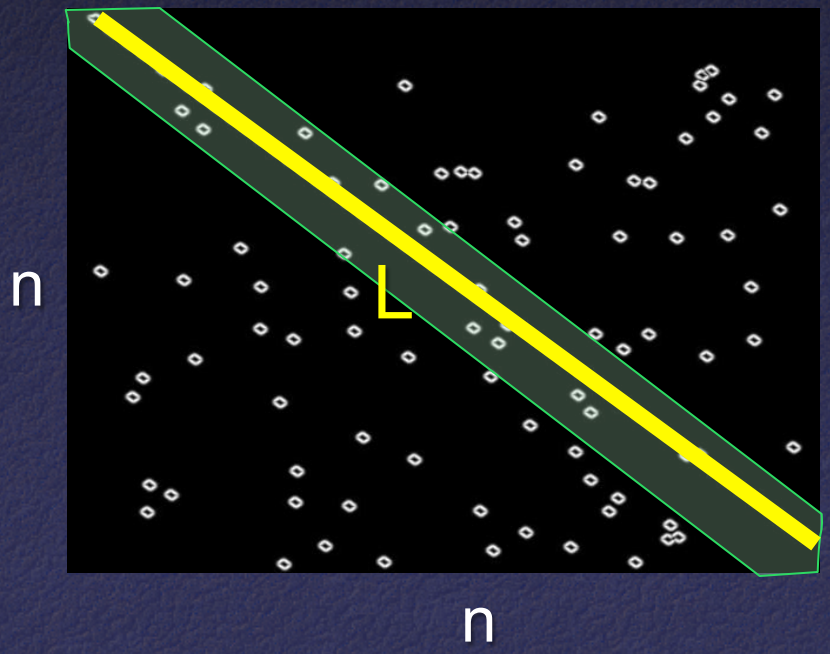


RANSAC for Line Detection

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

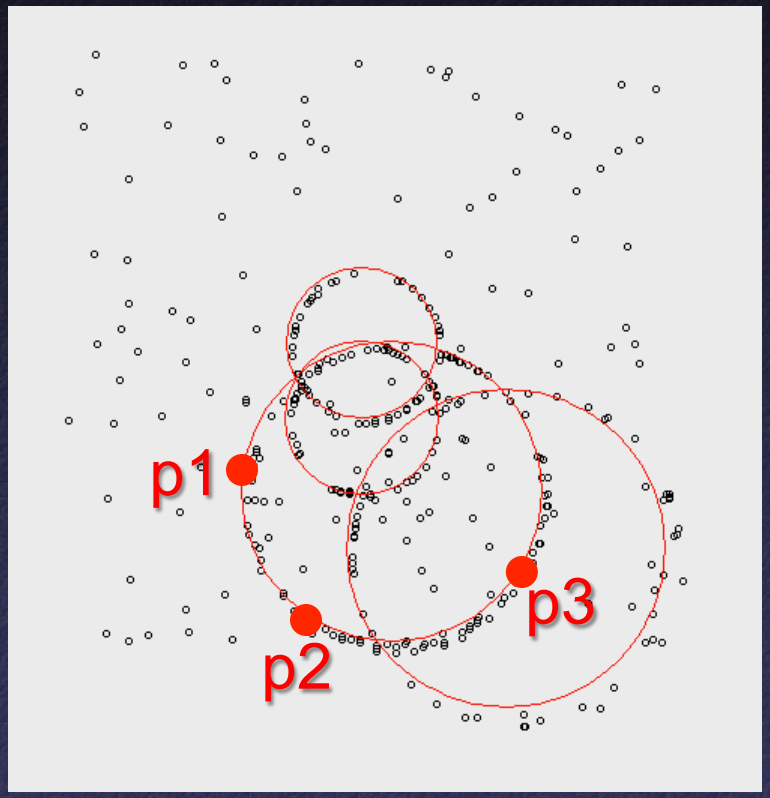
- Compute fit for each sample
- See how many points agree
- Remember the best

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

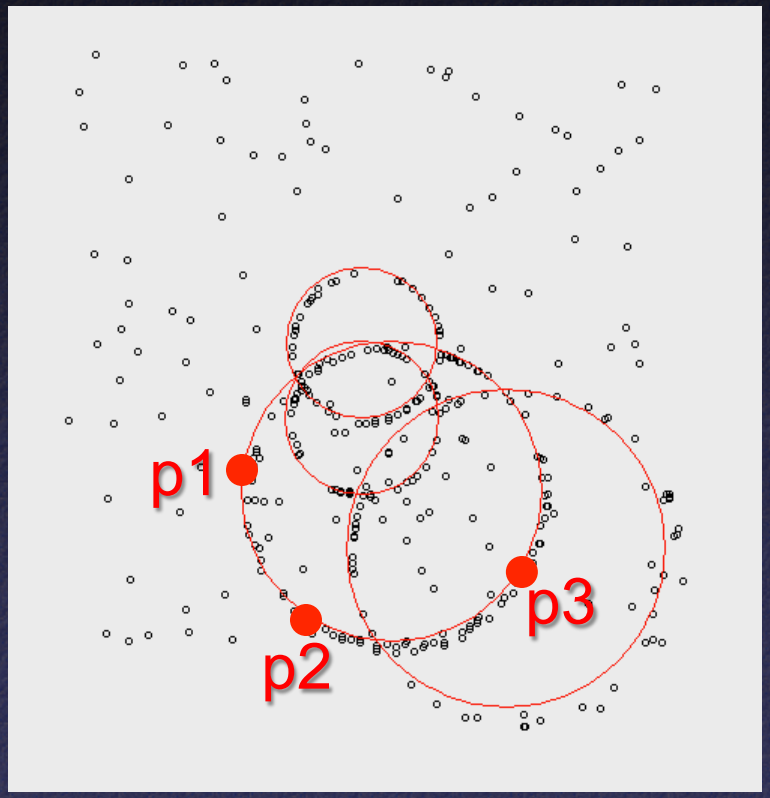


RANSAC for Circle Detection

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?

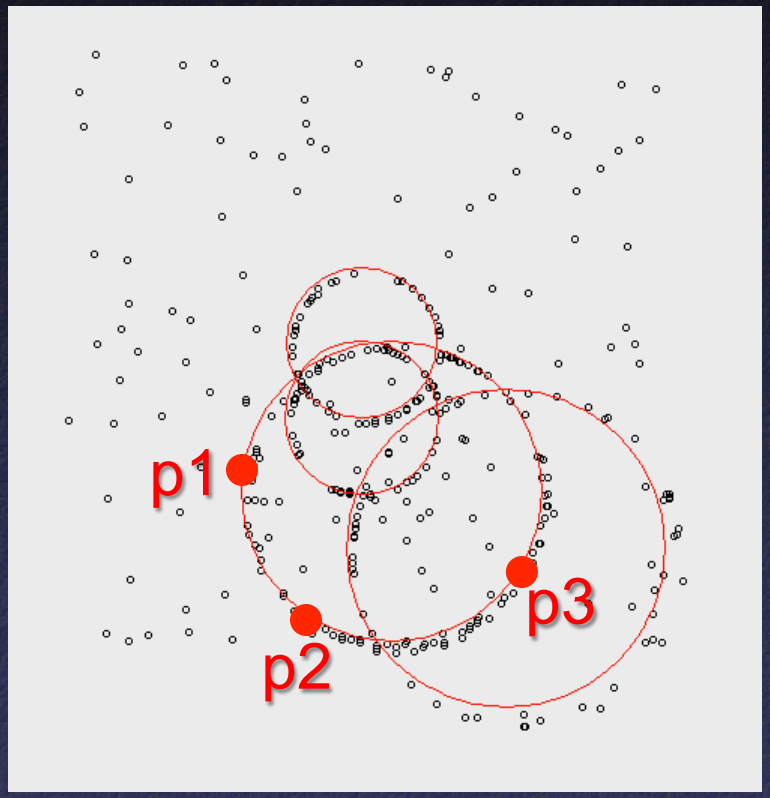


RANSAC for Circle Detection

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

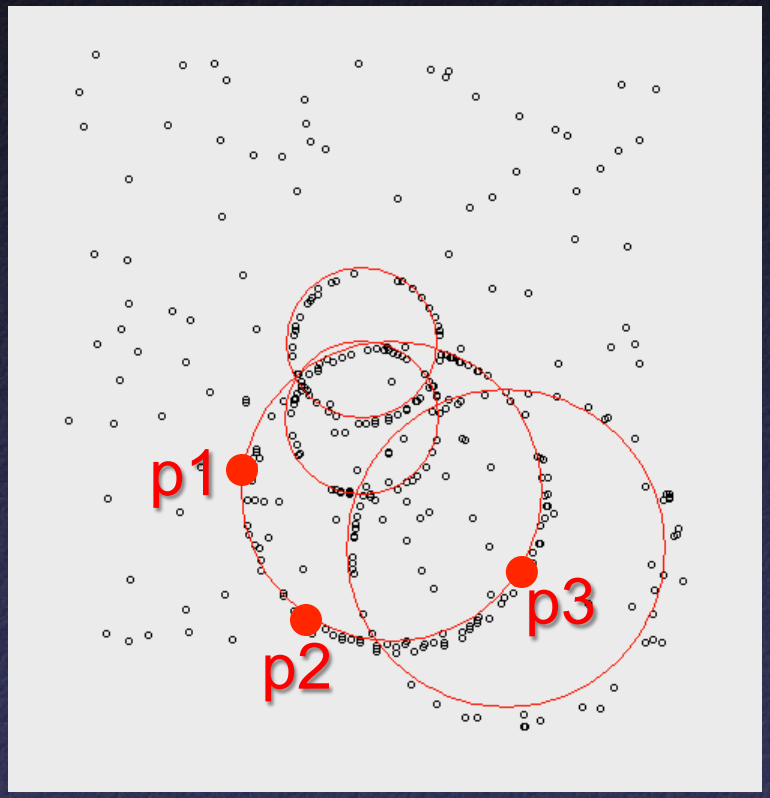
How can we improve the running time?



RANSAC for Circle Detection

Possible parameterizations for circles:

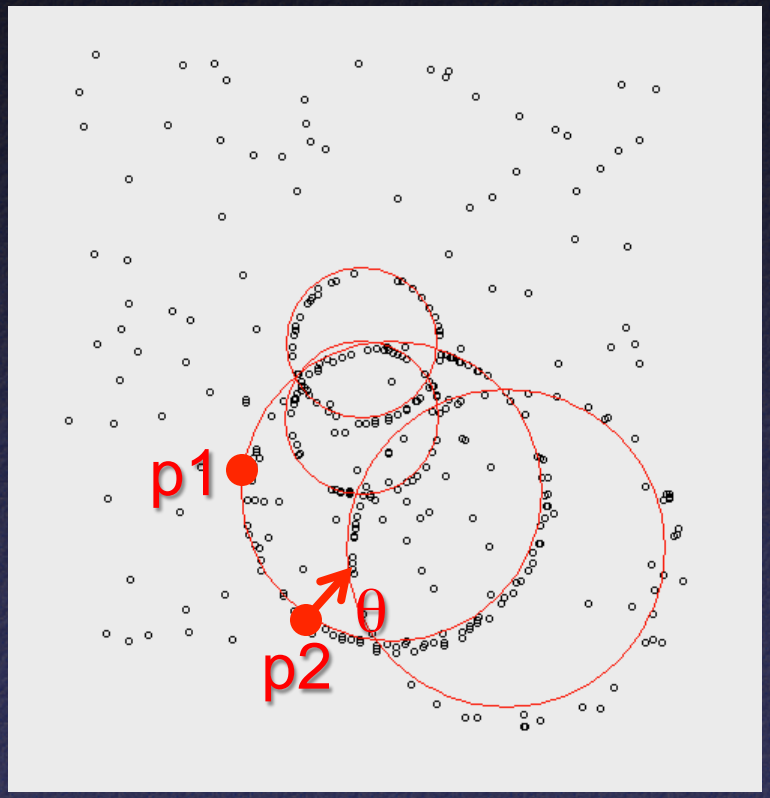
- 6 dof: three points
- 5 dof: two points, one angle
- 4 dof: one point, one angle, one radius



RANSAC for Circle Detection

Possible parameterizations for circles:

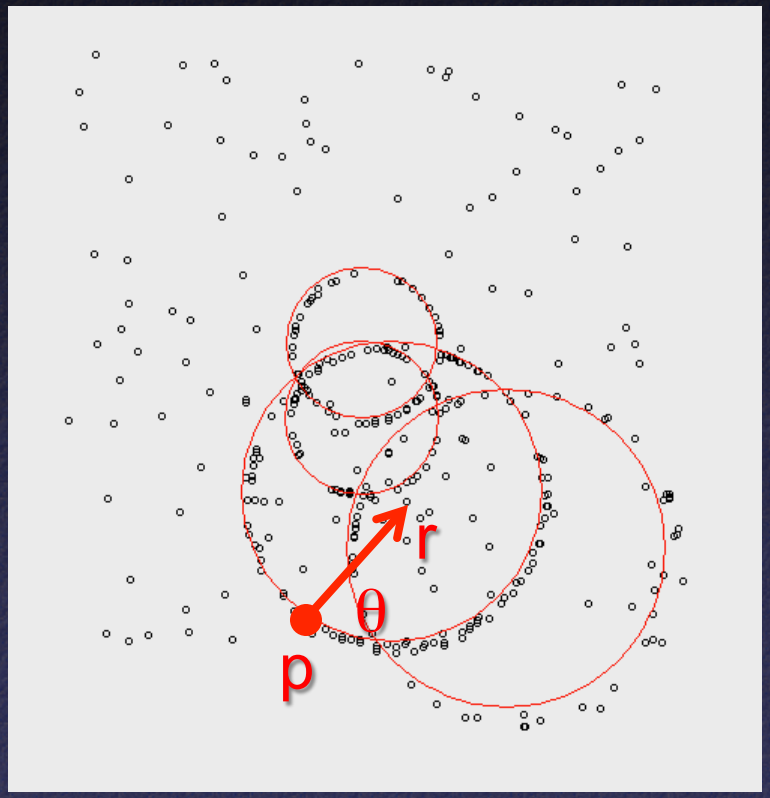
- 6 dof: three points
- 5 dof: two points, one angle
- 4 dof: one point, one angle, one radius



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

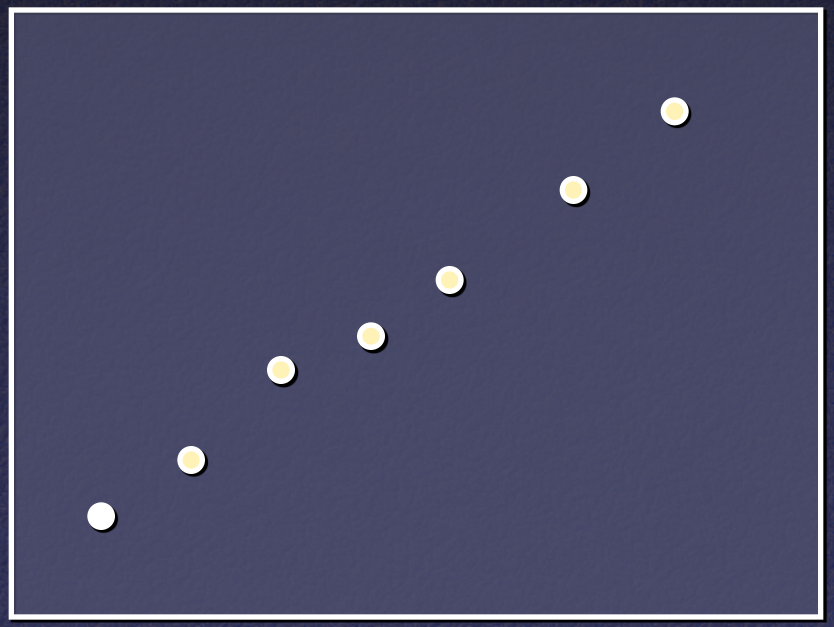


Today

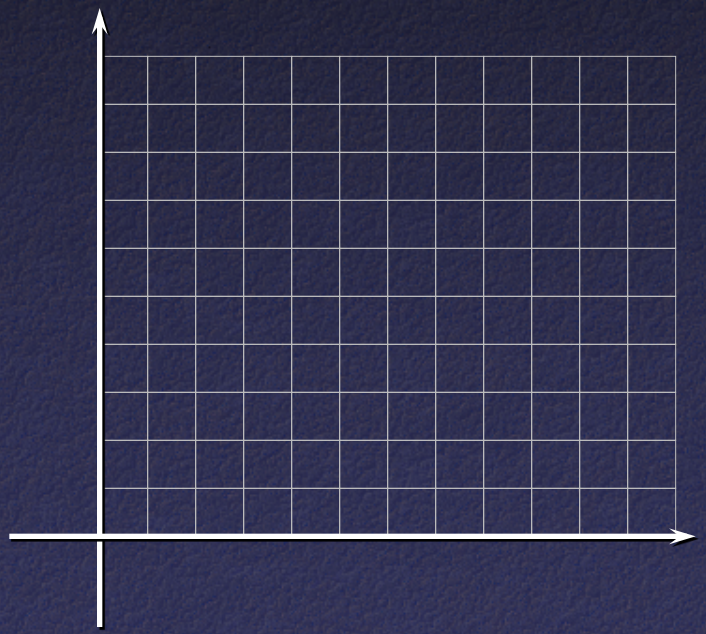
- A simple world
- A catalog of edges
- Generic view assumption
- Constrained solver
- Edge detection: Canny
- Line detection
  - RANSAC
  - Hough transformation

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:

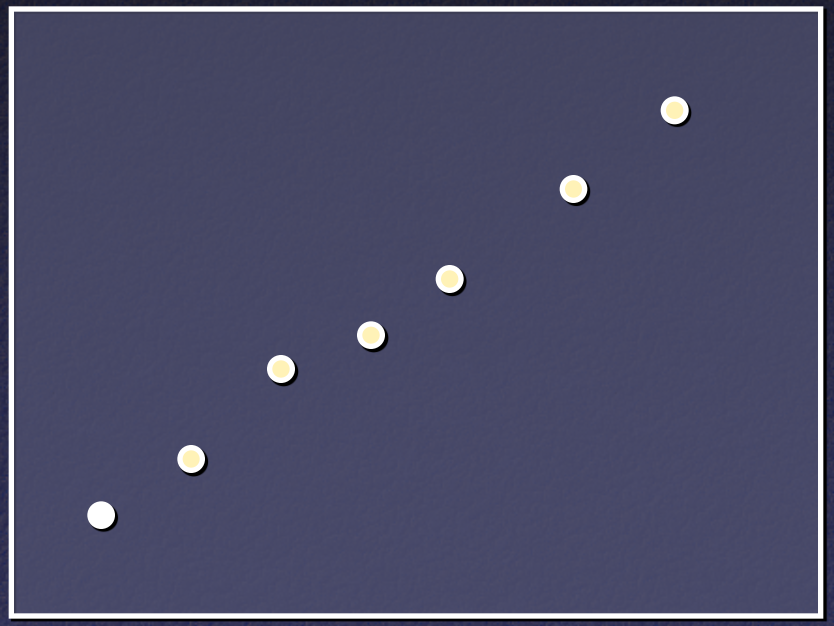
- Add support to all Hough space bins representing primitives containing p

At end:

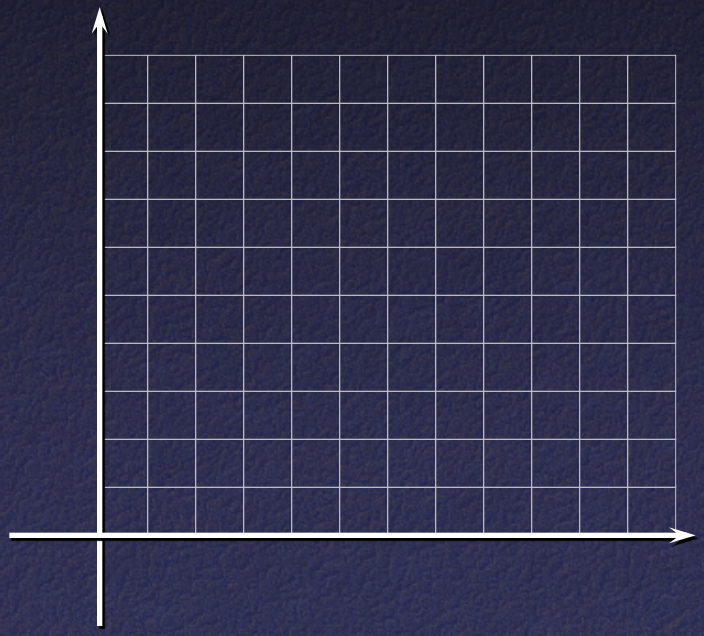
- Find the Hough space bin(s) with the most support

Hough Transform for Line Detection

Example:



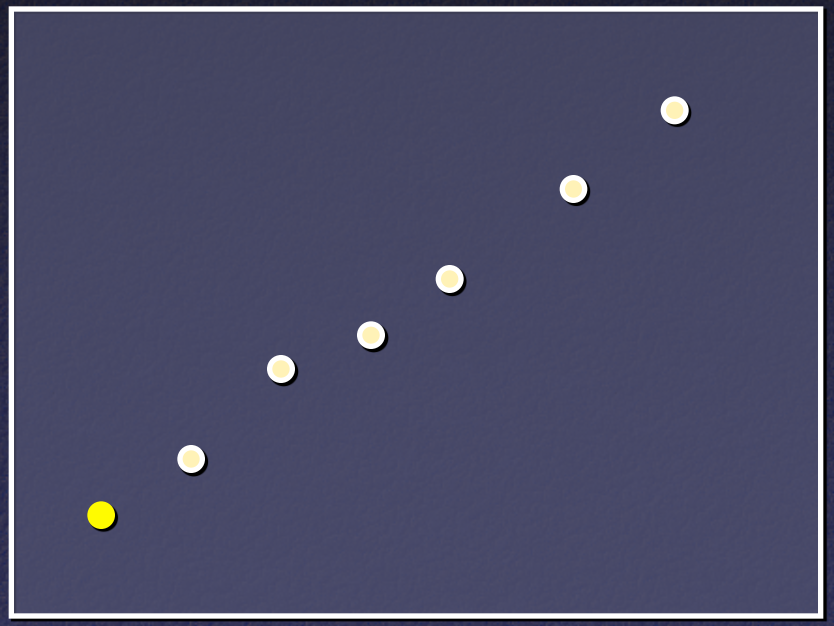
Image



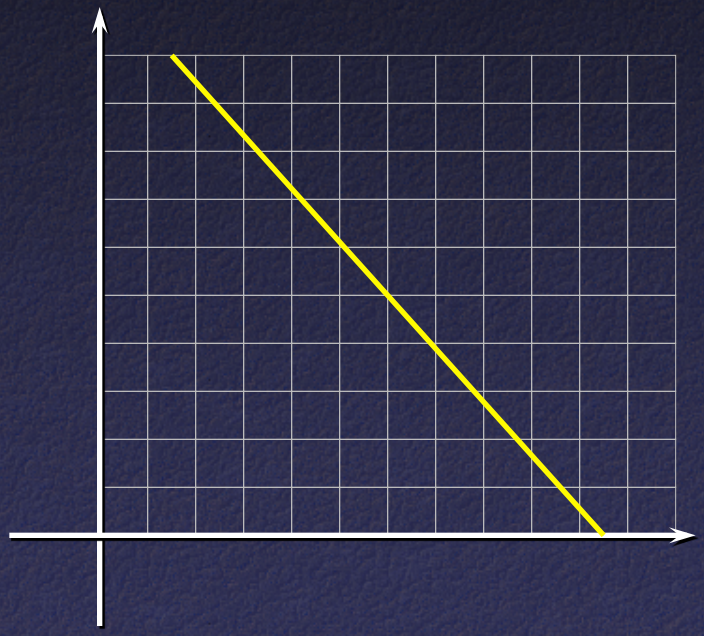
Hough Space

Hough Transform for Line Detection

Example:



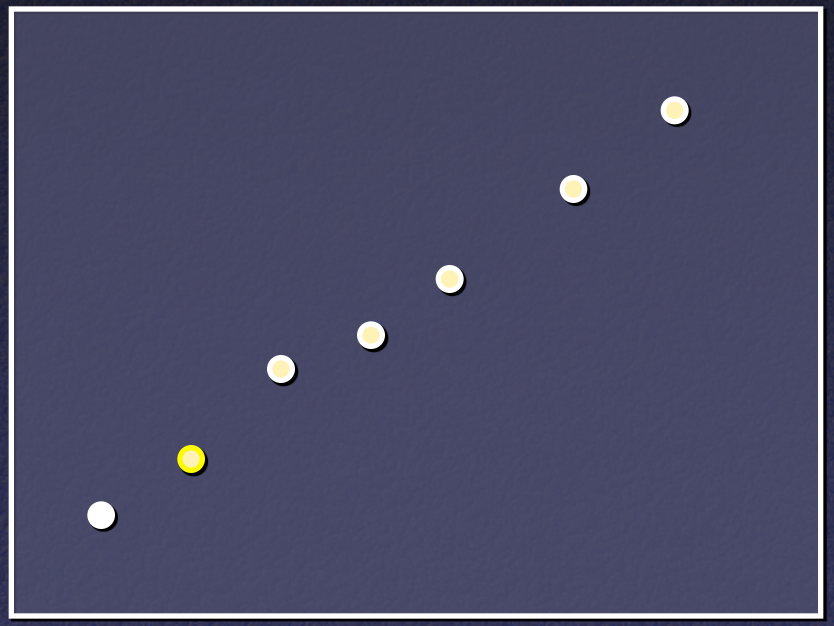
Image



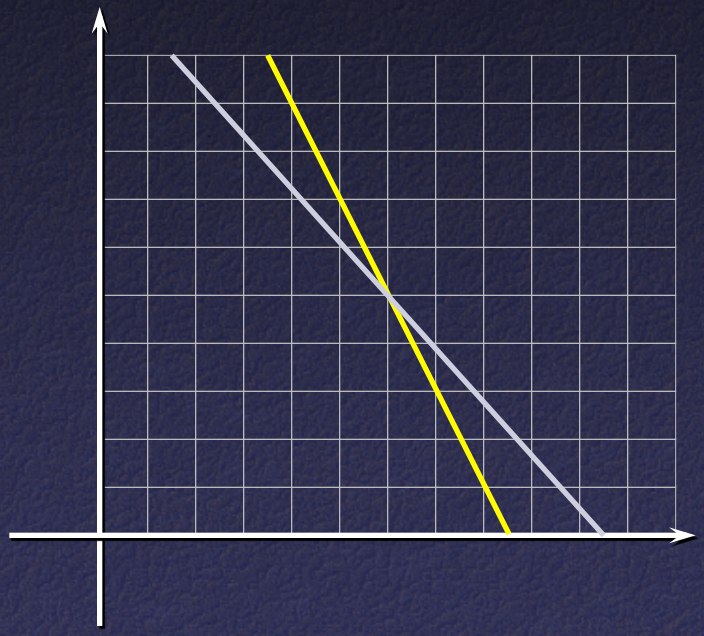
Hough Space

Hough Transform for Line Detection

Example:



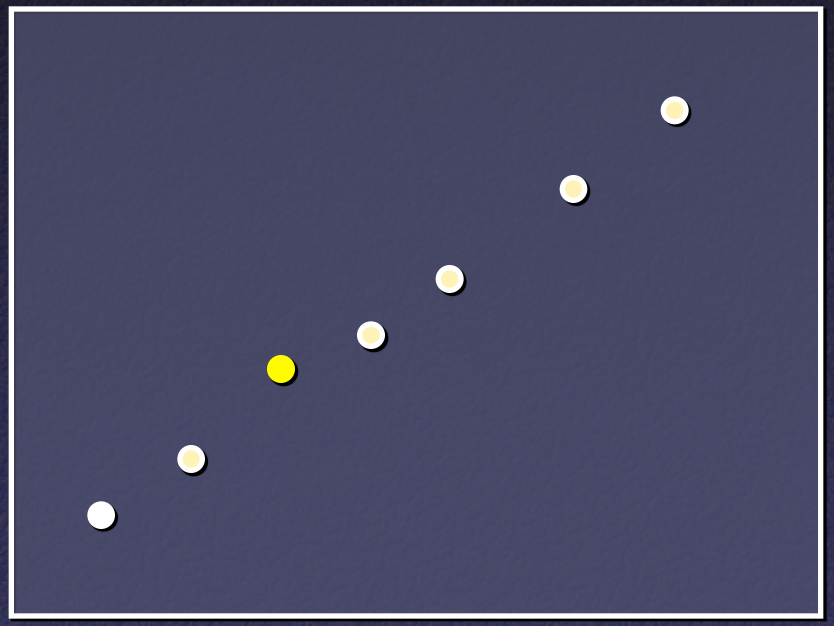
Image



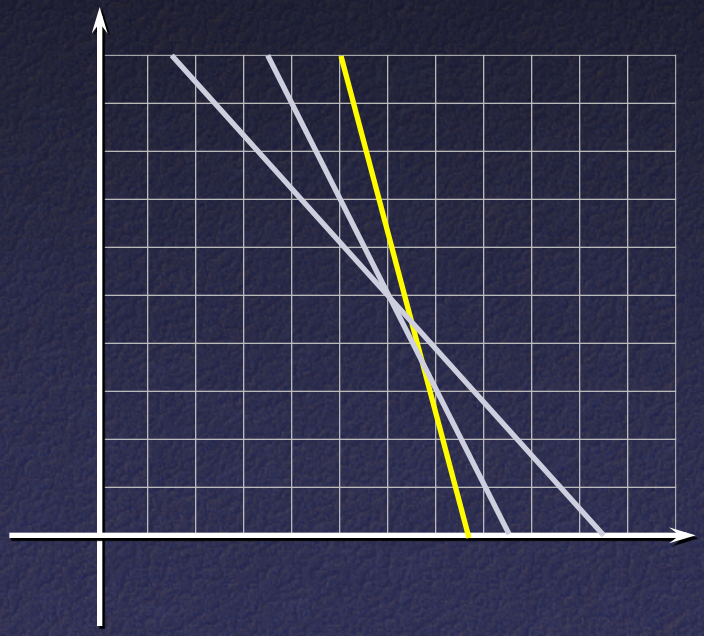
Hough Space

Hough Transform for Line Detection

Example:



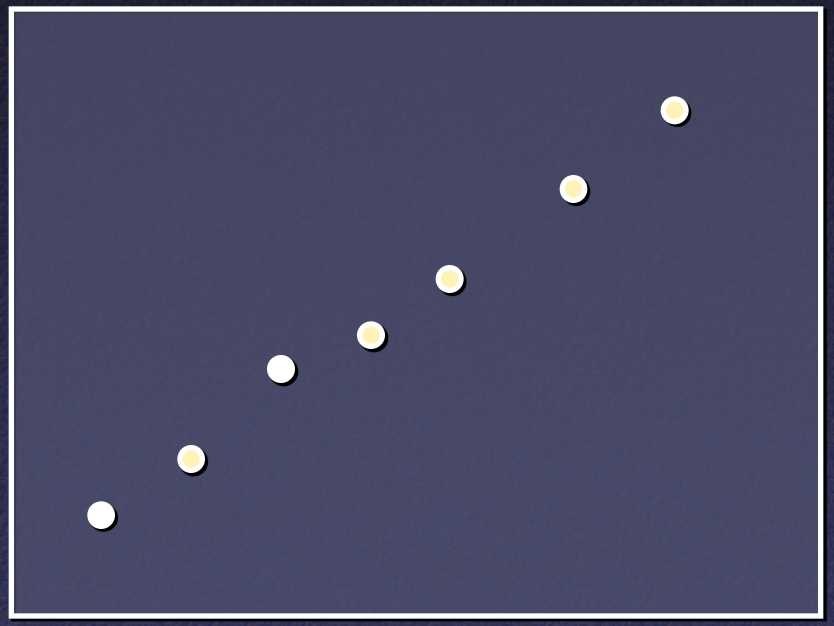
Image



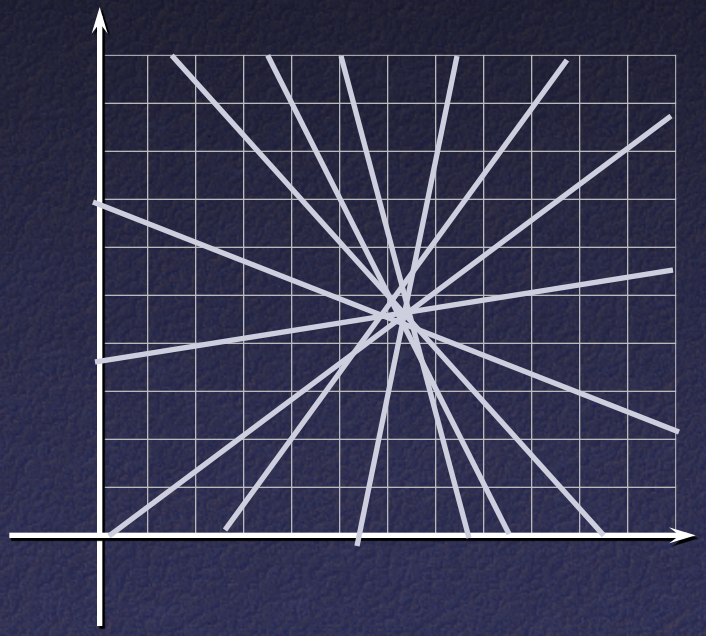
Hough Space

Hough Transform for Line Detection

Example:



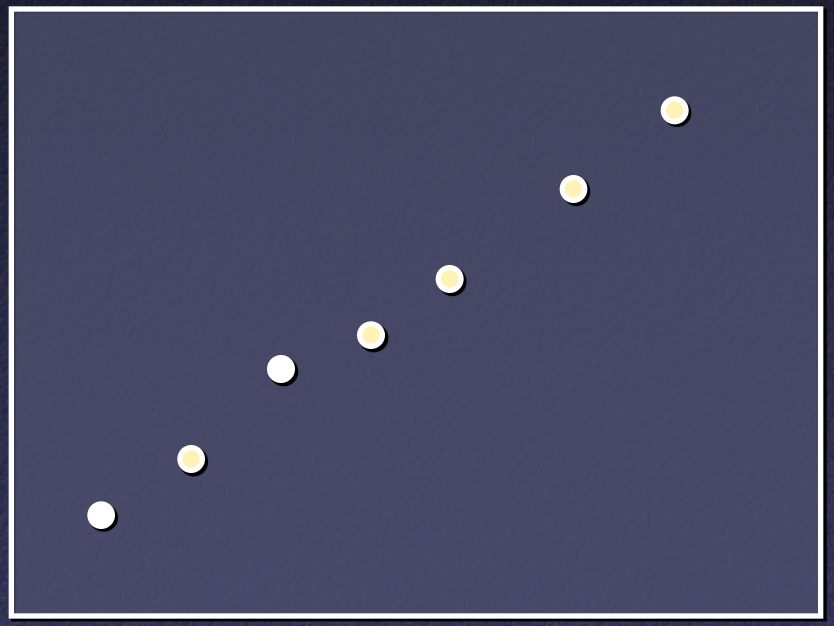
Image



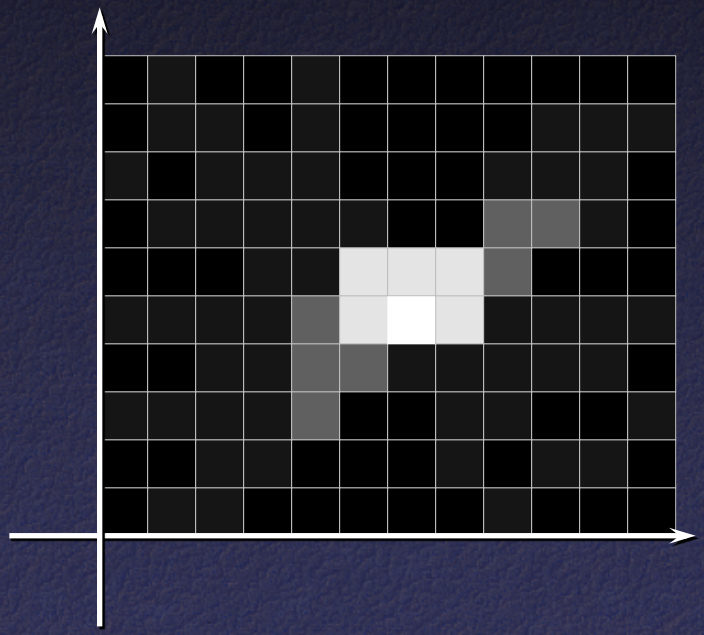
Hough Space

Hough Transform for Line Detection

Example:



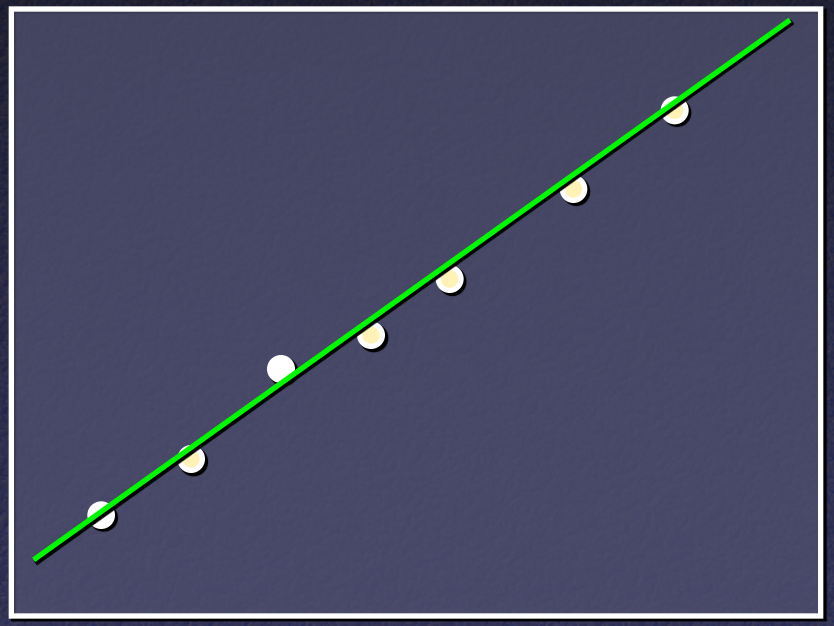
Image



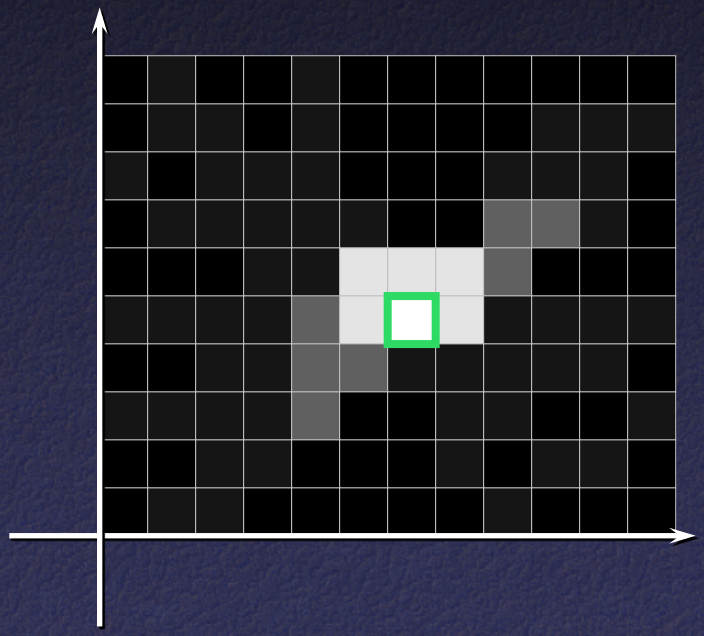
Hough Space

Hough Transform for Line Detection

Example:



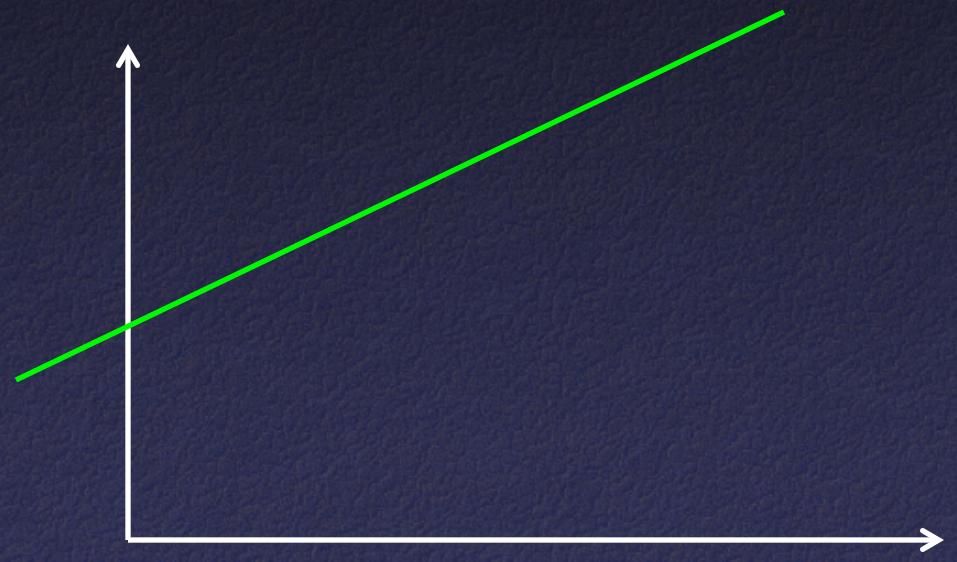
Image



Hough Space

Hough Transform for Line Detection

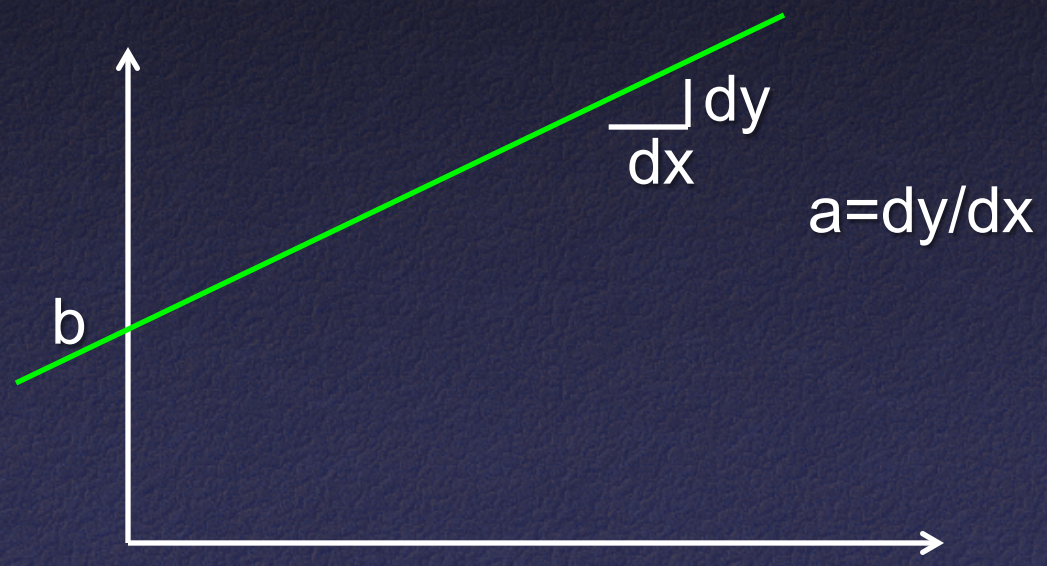
Key question: how to parameterize Hough space?



Hough Transform for Line Detection

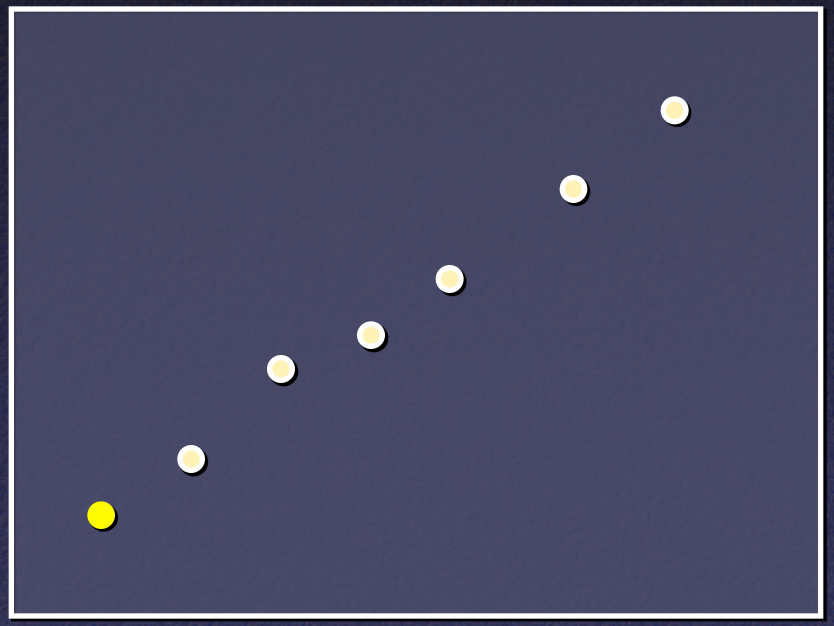
A 2 dof parameterization for lines: $y = ax+b$

Parameters: a and b

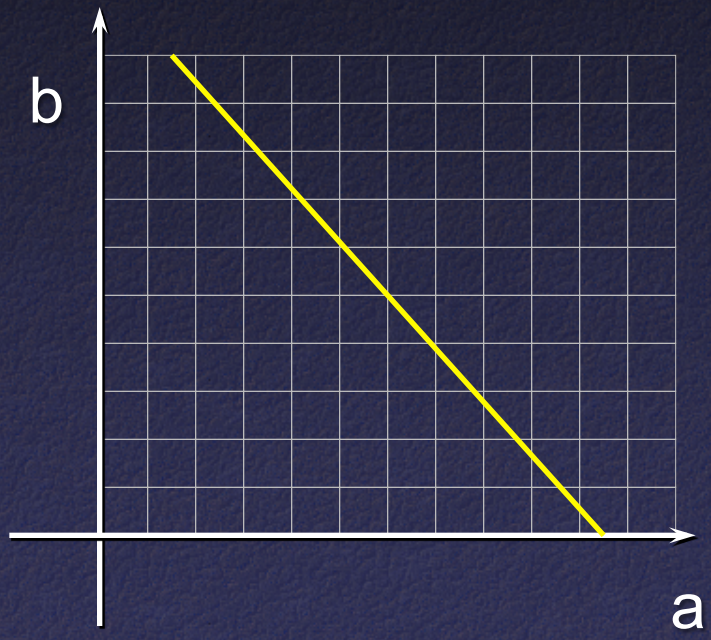


Hough Transform for Line Detection

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



Image

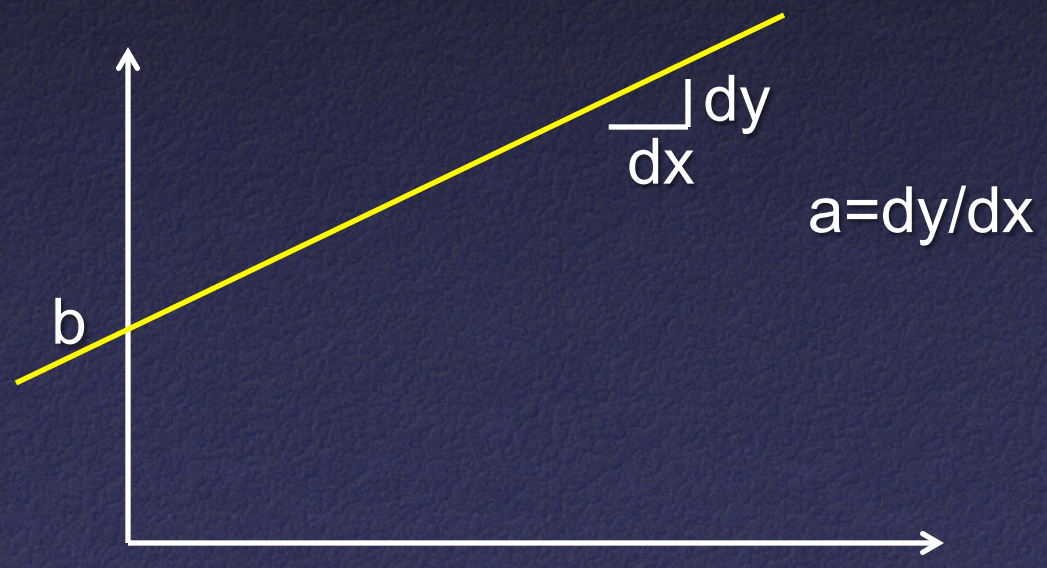


Hough Space

Hough Transform for Line Detection

Problems with slope / intercept parameterization

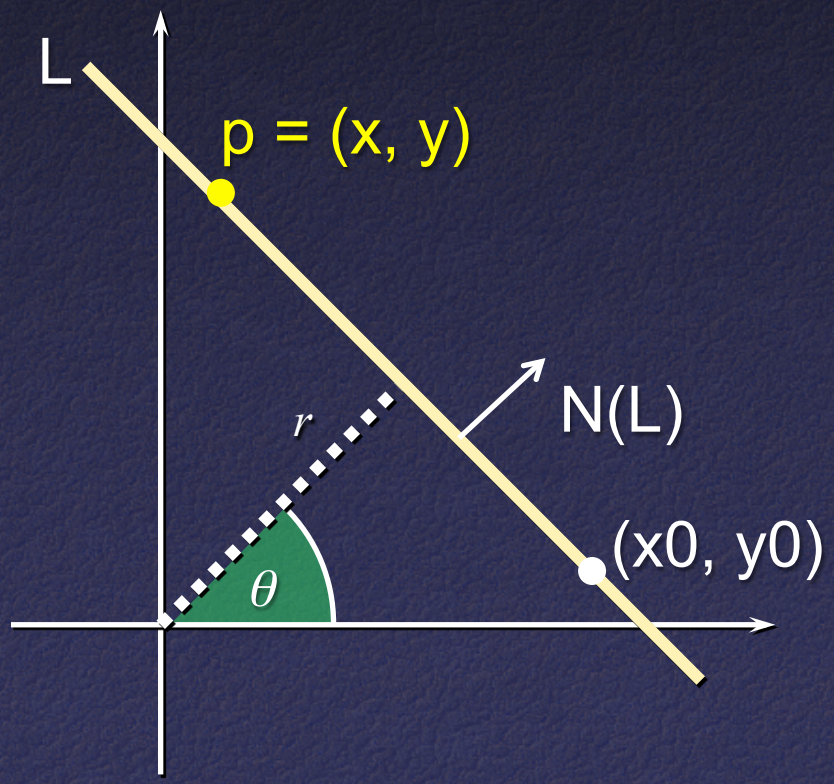
- Non-uniform sampling of directions
- Can’t represent vertical lines



Hough Transform for Line Detection

Alternative: angle / distance parameterization

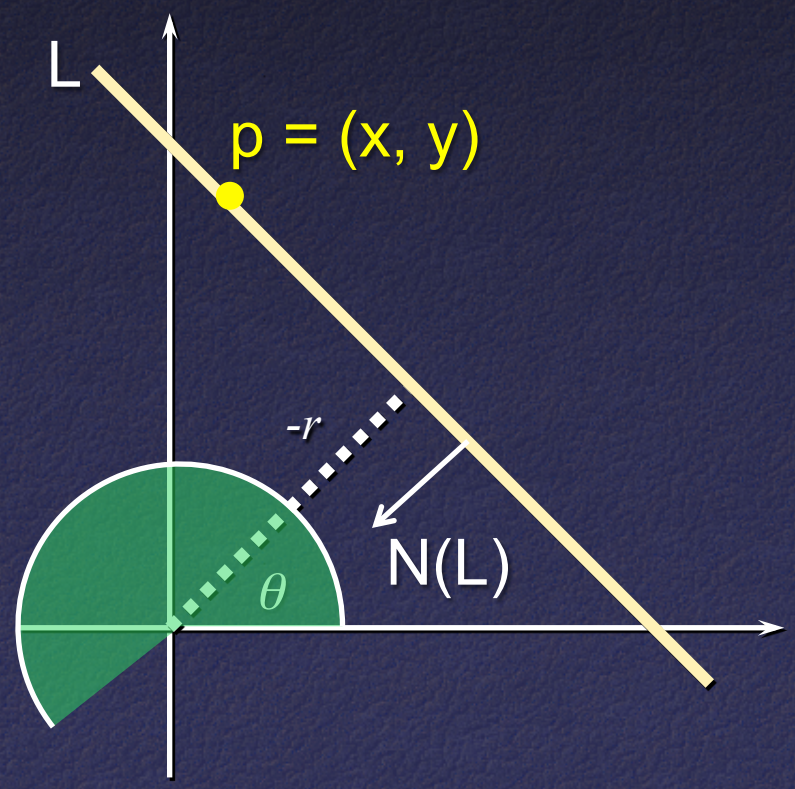
- Line represented as (r,θ) where
  - $x \cos\theta + y \sin\theta = r$
  - $r = -\cos\theta \bullet x - \sin\theta \bullet y$
  - $N(L) = (\cos\theta, \sin\theta)$
  - $dist(L, p) = x \cos\theta + y \sin\theta - r$



Hough Transform for Line Detection

Alternative: angle / distance parameterization

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



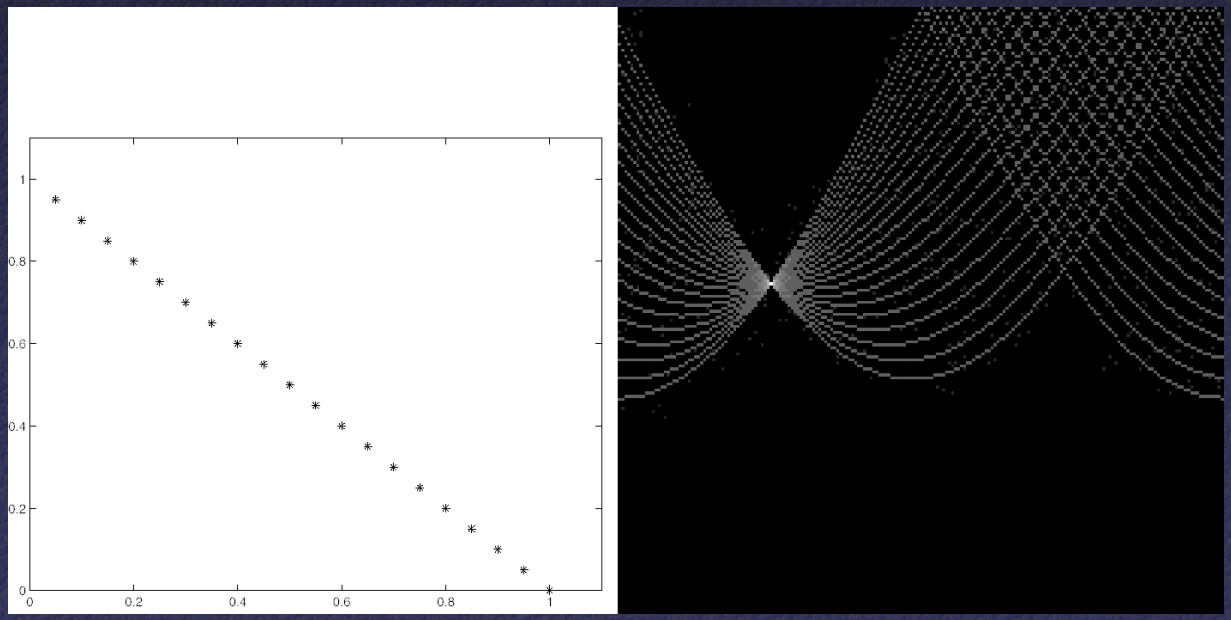
Hough Transform for Line Detection

Alternative: angle / distance parameterization

- Line represented as (r,θ)

  + *Uniform sampling of angles*

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



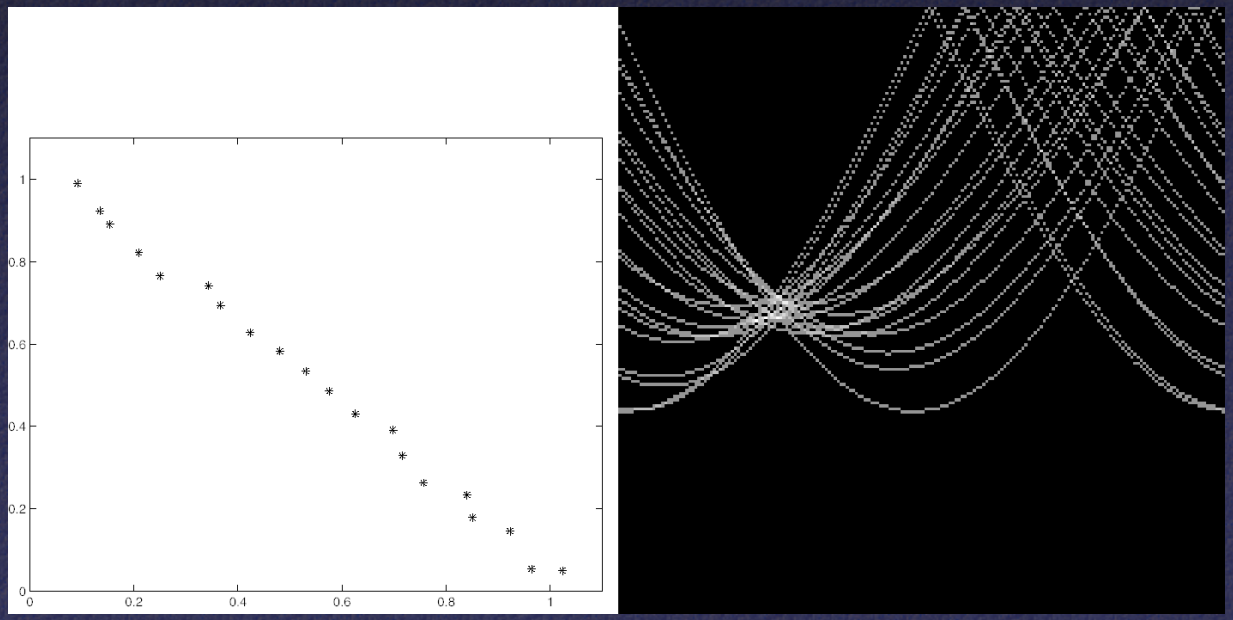
Hough Transform for Line Detection

Alternative: angle / distance parameterization

- Line represented as (r,θ)

  + *Uniform sampling of angles*

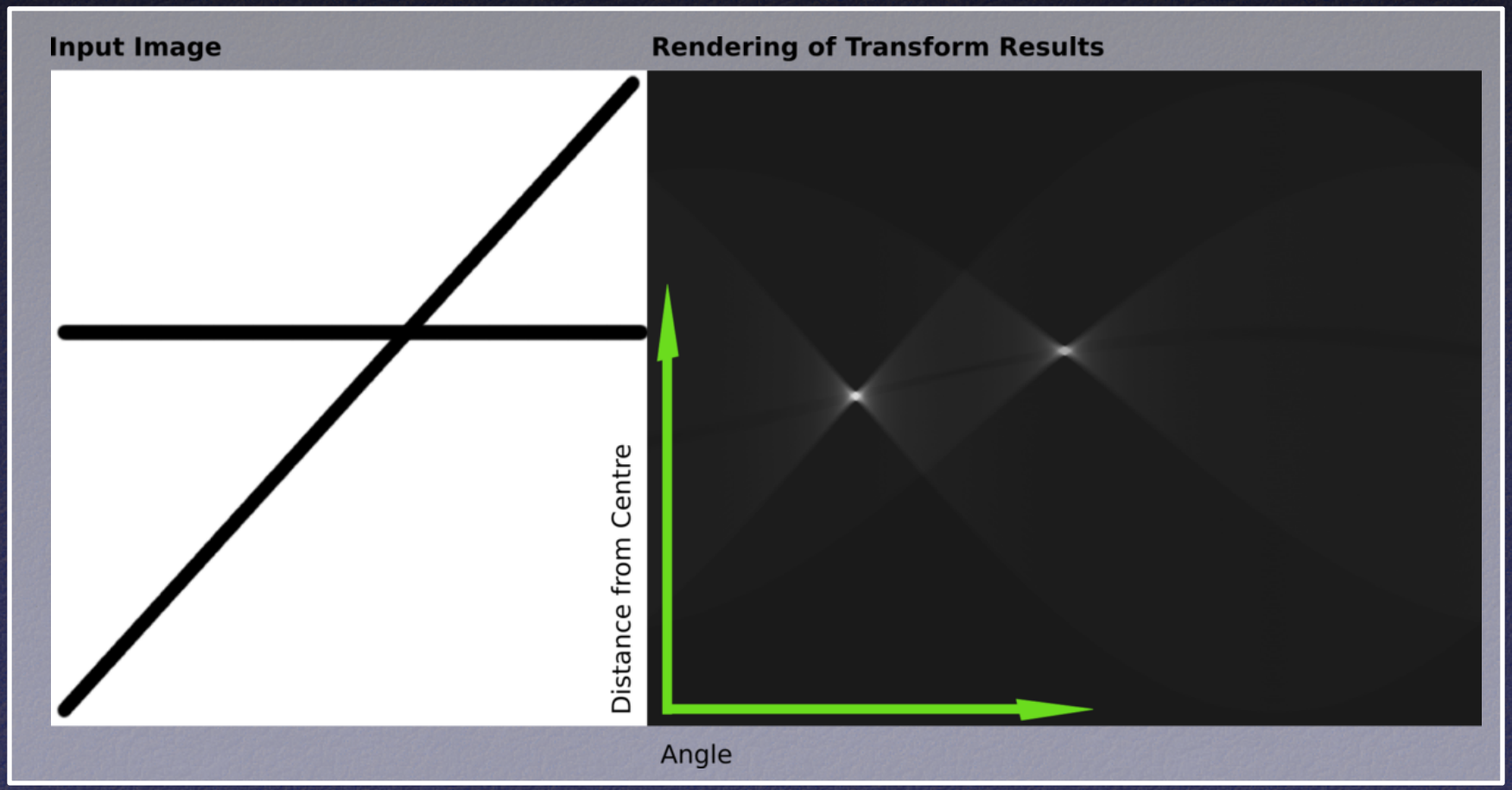
  -- *Lines through point*
  lie on sinusoid in (r,θ)



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

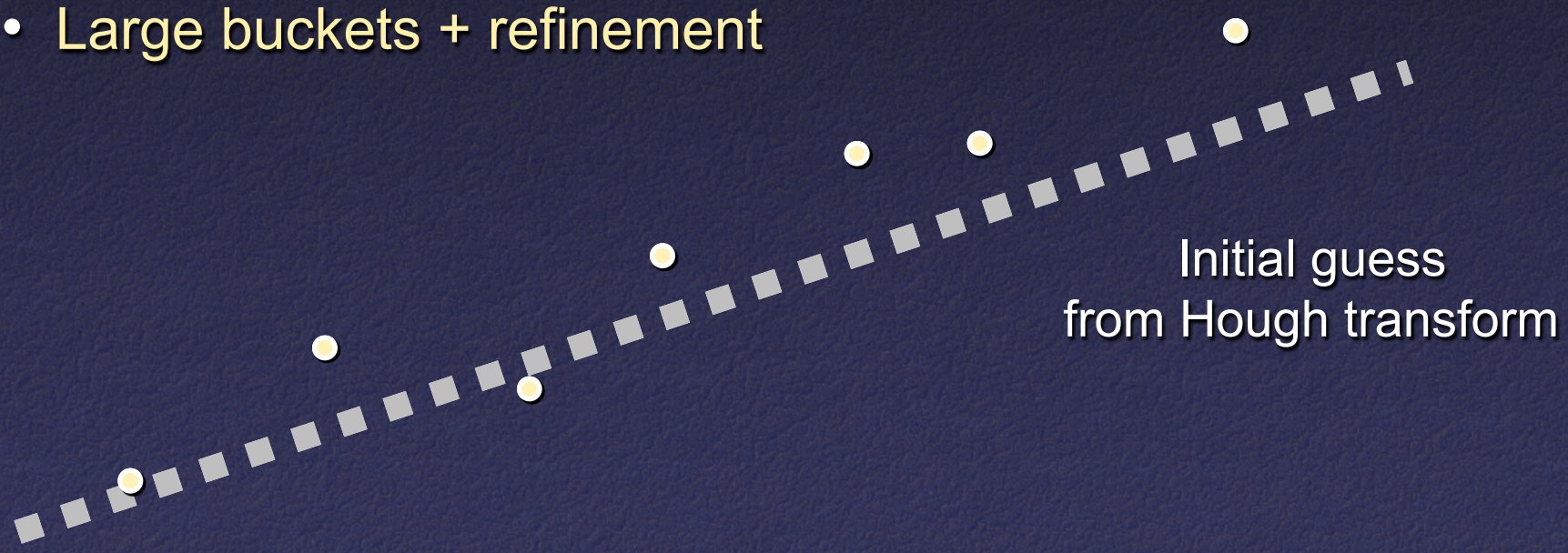
Hough Transform for Line Detection

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

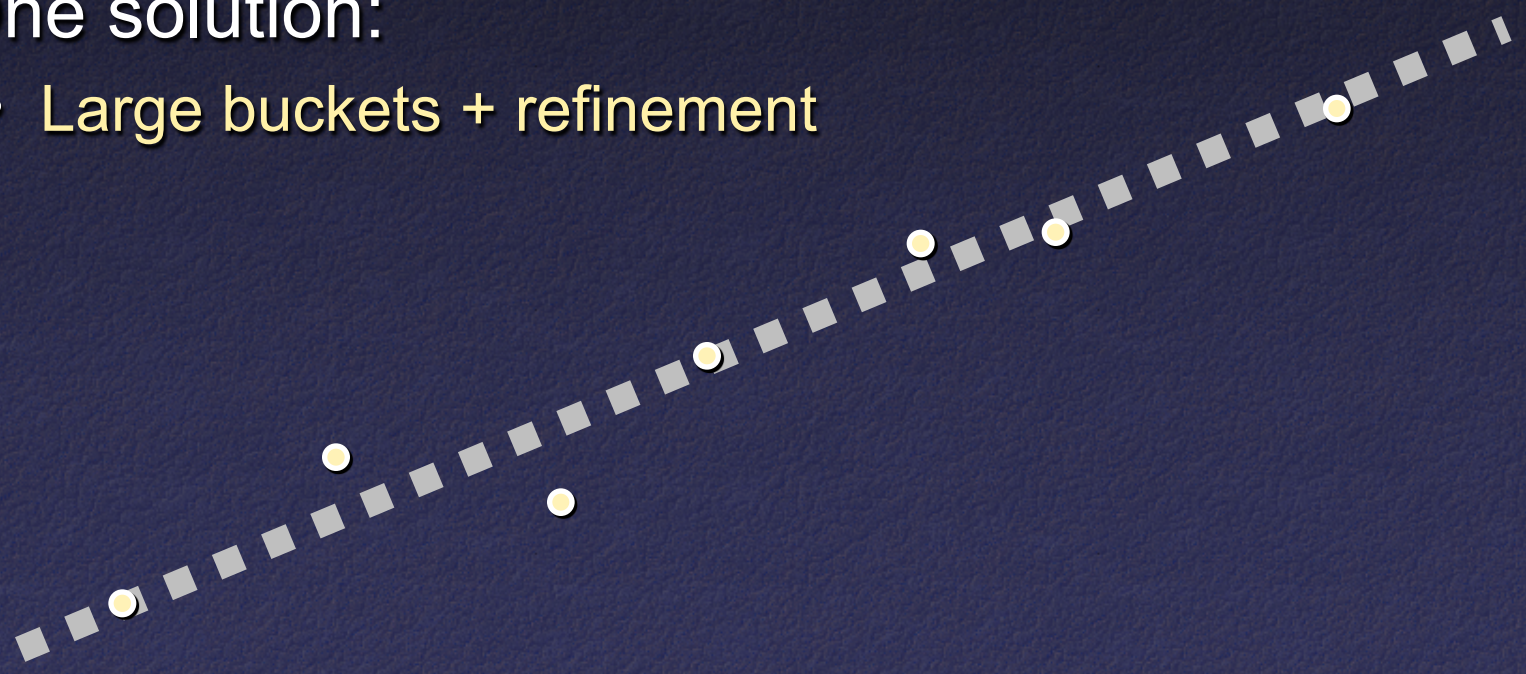
Hough Transform for Line Detection

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

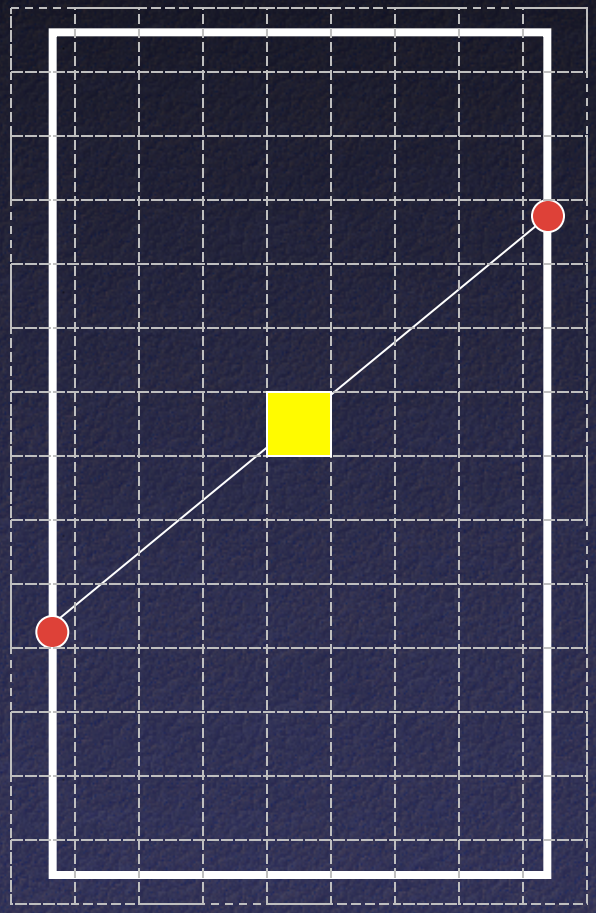
- Possible parameterization = 2D position and radius

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

Hough Transform for Symmetry Detection

Symmetry transform:

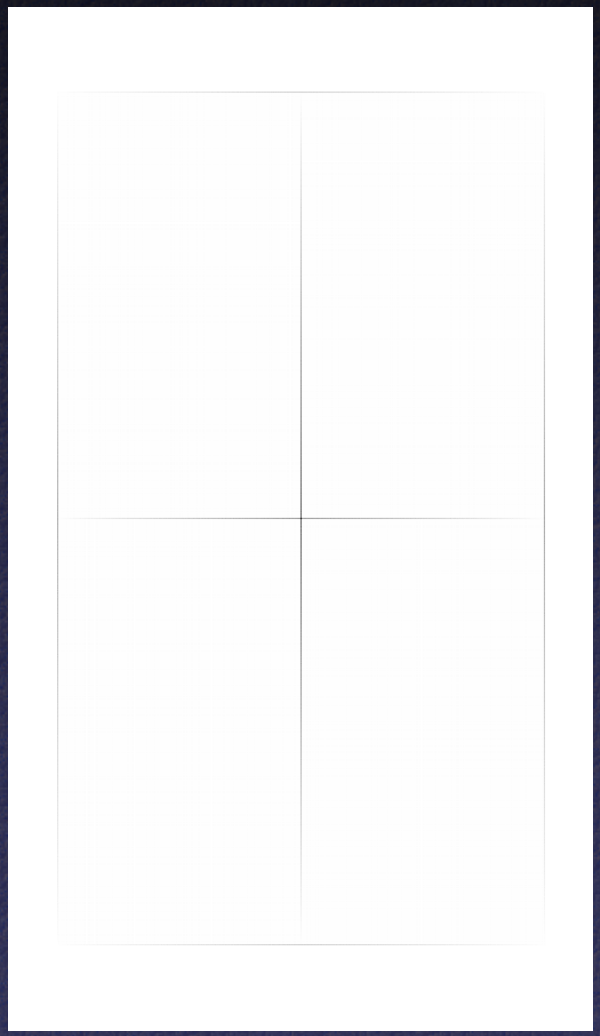
- Vote for midpoints between pixels



Hough Transform for Symmetry Detection

Symmetry transform:

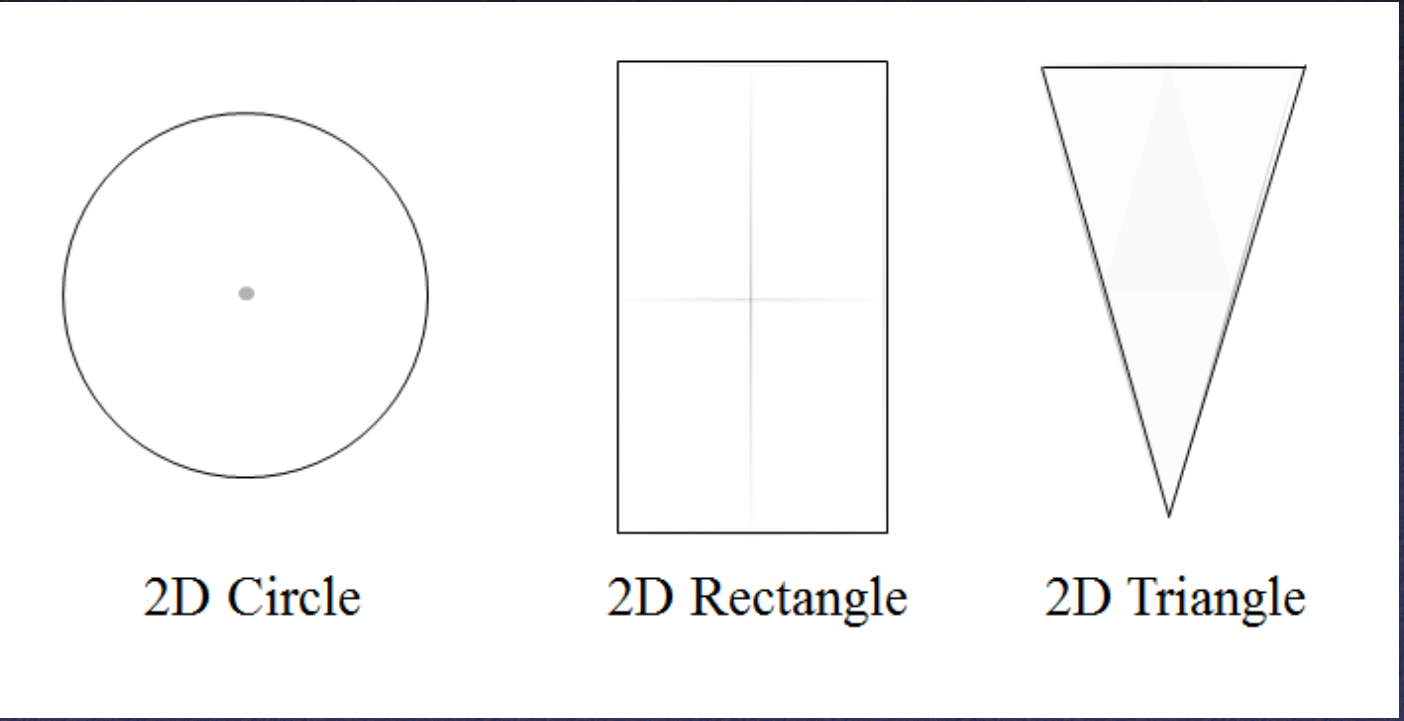
- Vote for midpoints between pixels



Hough Transform for Symmetry Detection

Symmetry transform:

- Vote for midpoints between pixels

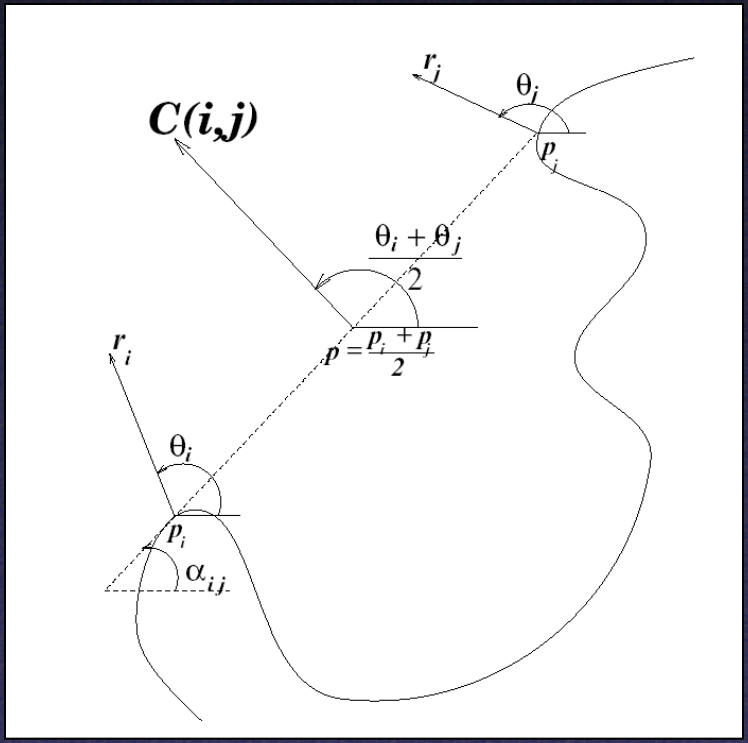


Hough Transform for Symmetry Detection

Symmetry transform:

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

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

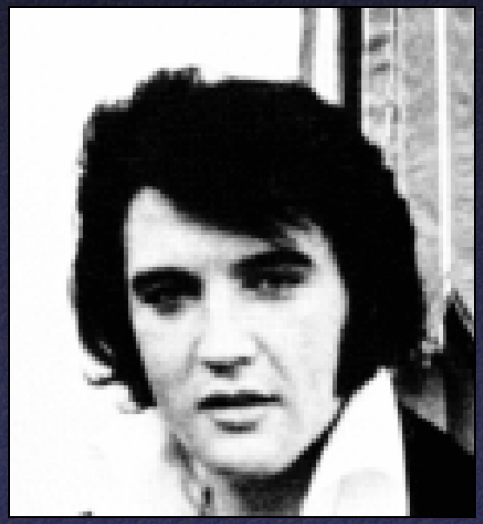


Reisfeld, et. al.

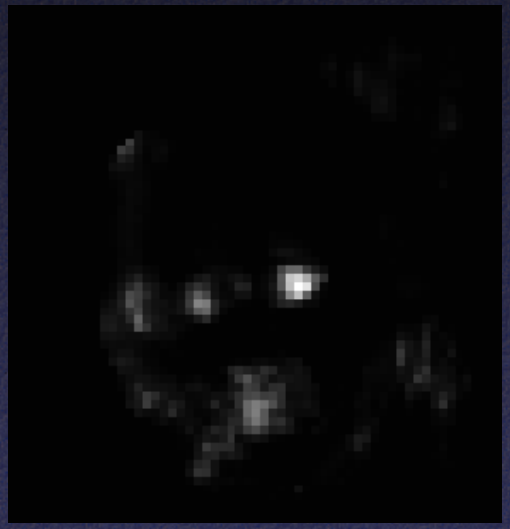
Hough Transform for Symmetry Detection

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?

Combine Edges and Lines

Detect edges locally

- Canny algorithm

Detect lines globally

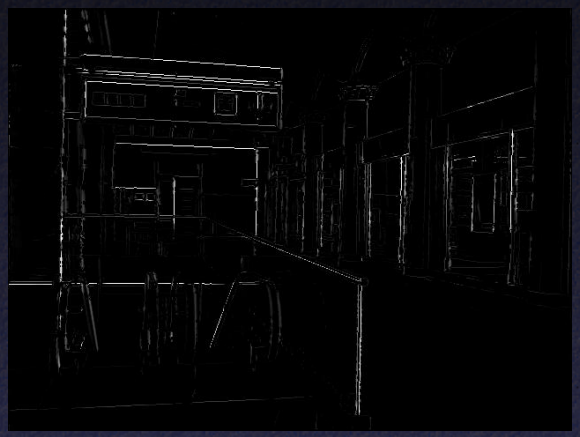
- Hough algorithm

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
```

Combine Edges and Lines

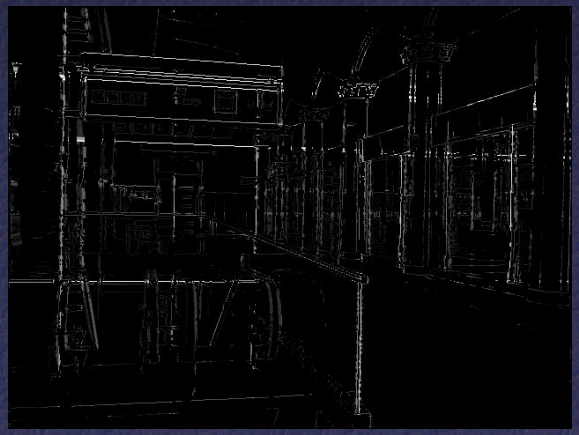
Input



Output

Hough Transform

Canny edges



Strong Hough lines

Summary

Problem:

- Structure detection

Focus: line detection

- RANSAC
- Hough transform

Extensions

- Circles and other primitives
- Symmetries

Applications

- Segmentation
- Alignment

Today

- A simple world
- A catalog of edges
- Generic view assumption
- Constrained solver
- Edge detection: Canny
- Line detection
  - RANSAC
  - Hough transformation