Image and Vision Computing

Features

Instructor: Chen Yisong

HCI & Multimedia Lab, Peking University

实践出真知

纸上得来终觉浅
绝知此事要躬行

——陆游《冬夜读书示子聿》

古人学问无遗力，少壮工夫老始成。纸上得来终觉浅，绝知此事要躬行。

Today's Goals

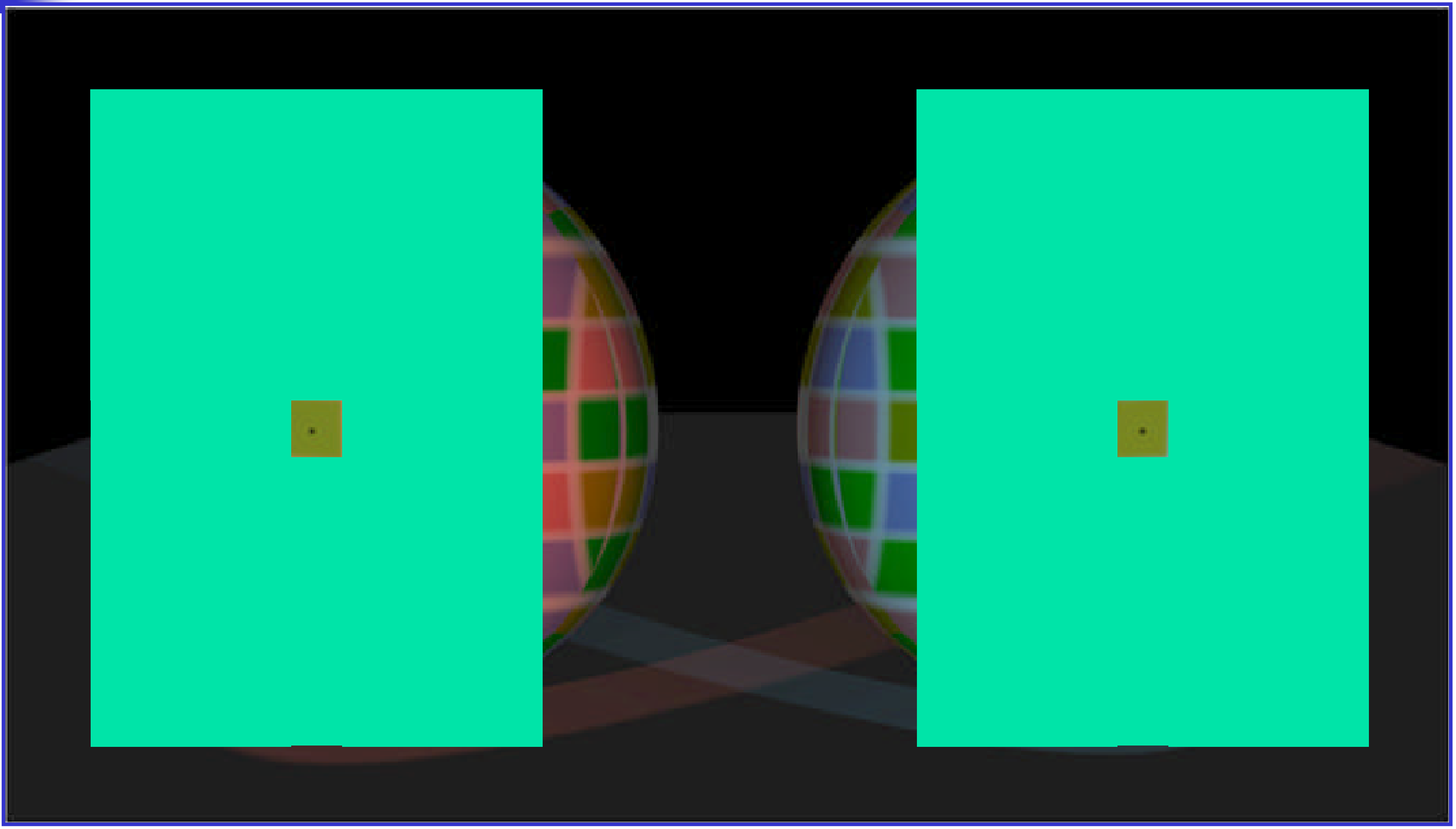
- Features Overview
- Canny Edge Detector
- Harris Corner Detector
- Templates and Image Pyramid
- SIFT Features

Today’s Questions

- What is a feature?
- What is an image filter?
- How can we find edges?
- How can we find corners?

- (How can we find cars in images?)

What is a Feature?



- Local, meaningful, detectable parts of the image

Features in Computer Vision

- What is a feature?
  - Location of sudden change
- Why use features?
  - Information content high
  - Invariant to change of view point, illumination
  - Reduces computational burden

Vanishing Points (无穷远点/灭点)

Vanishing Line (地平线)

Local versus global

Vanishing Line

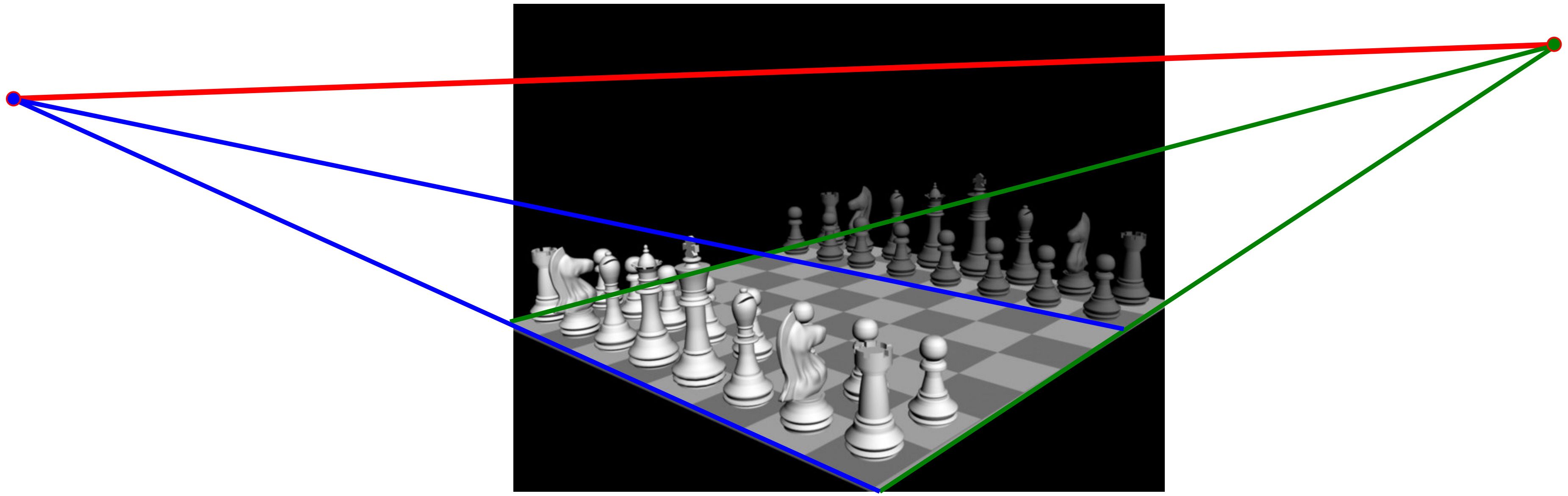
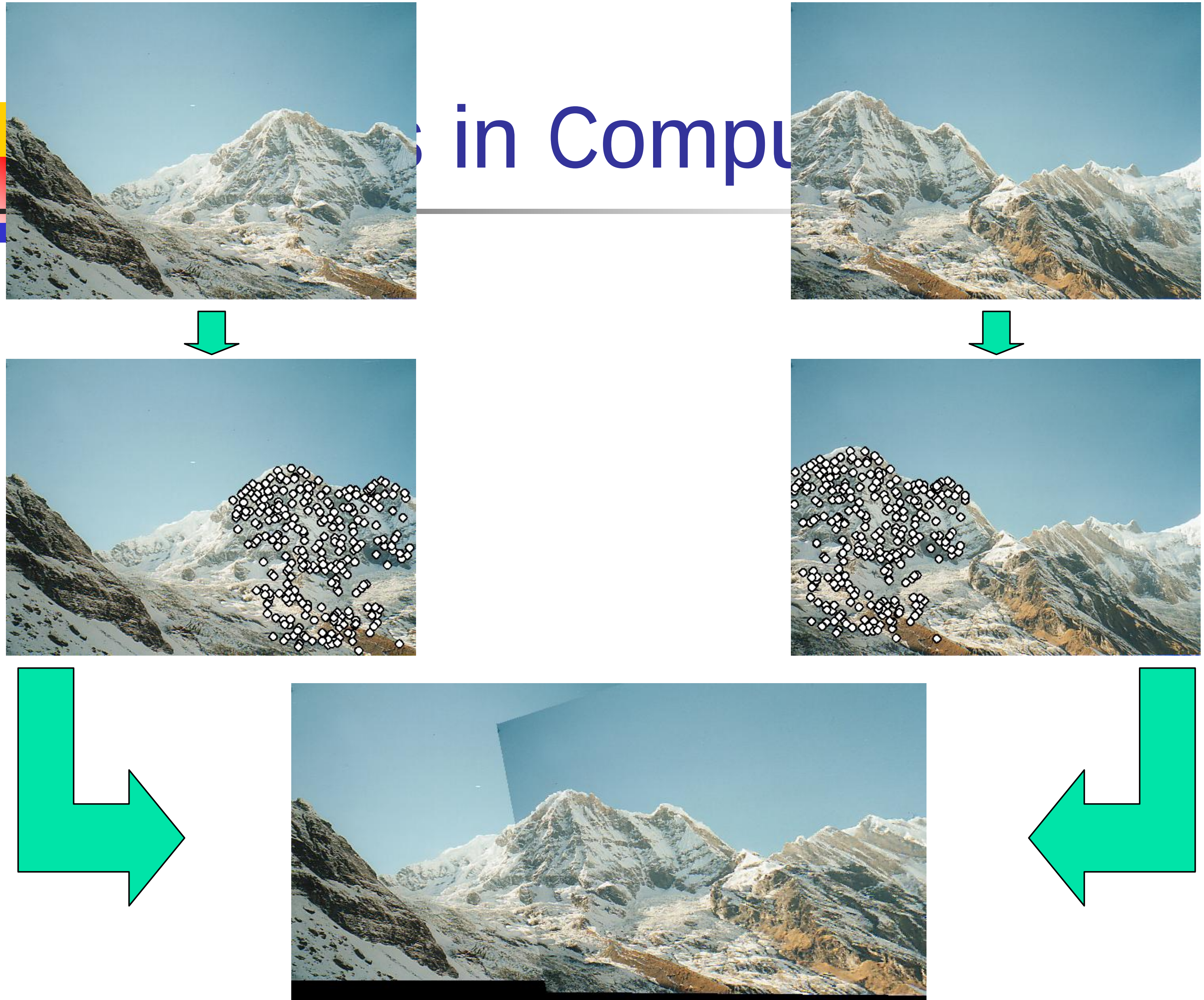
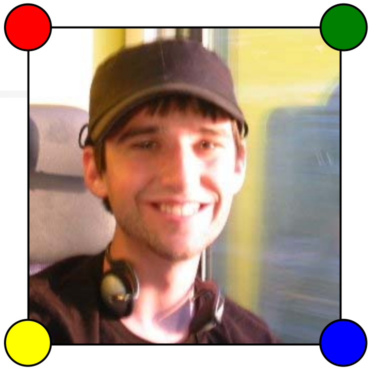


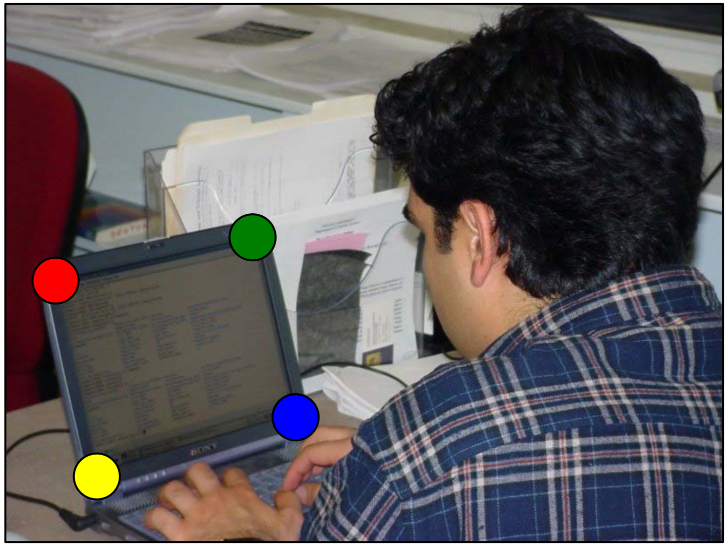
Image 1 **Image 2**



Features in computer vision

- Compositing



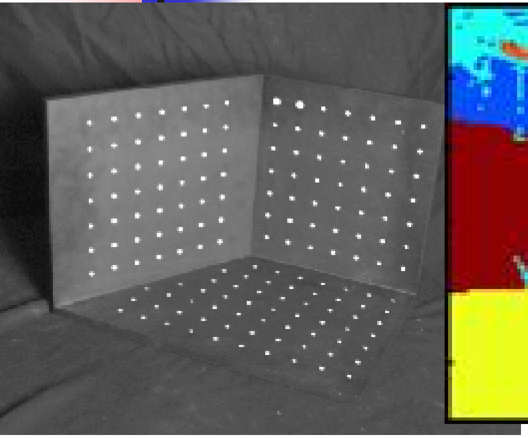


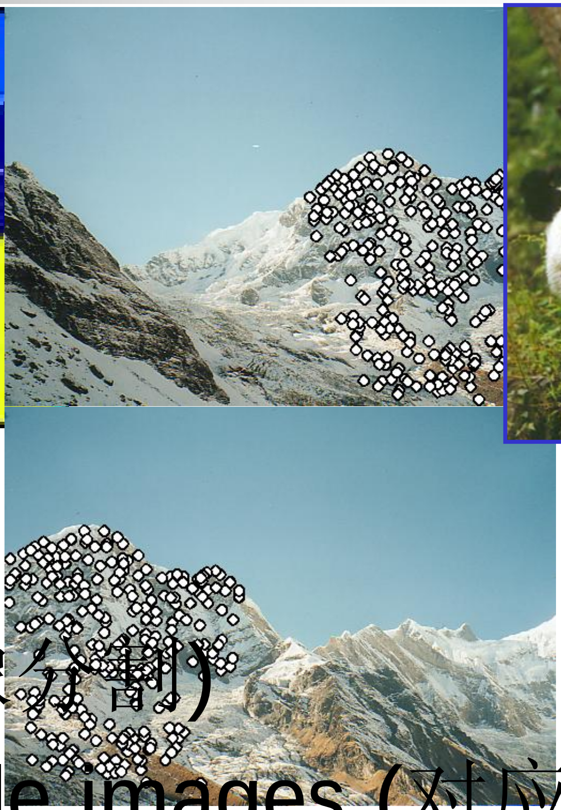
This is your test image set

Features in computer vision

- Mosaic

Where Features Are Used





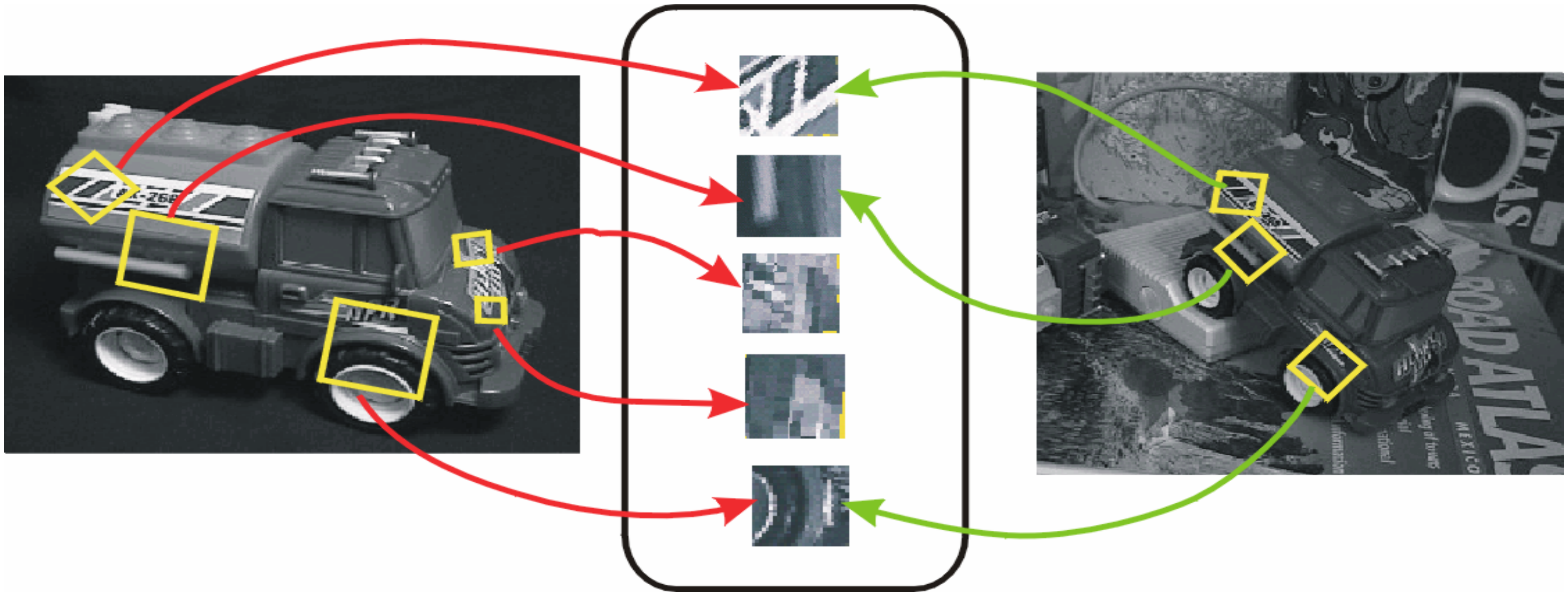
- Calibration(相机标定)
- Image Segmentation(图像分割)
- Correspondence in multiple images (对应匹配)
- Object detection, recognition(检测识别)

What Makes For *Good* Features?

- Invariance
  - View point (scale, orientation, translation)
  - Lighting condition
  - Object deformations
  - Partial occlusion
- Other Characteristics
  - Uniqueness
  - Sufficiently many
  - Tuned to the task

Invariant Local Features

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



SIFT = Scale Invariant Feature Transform

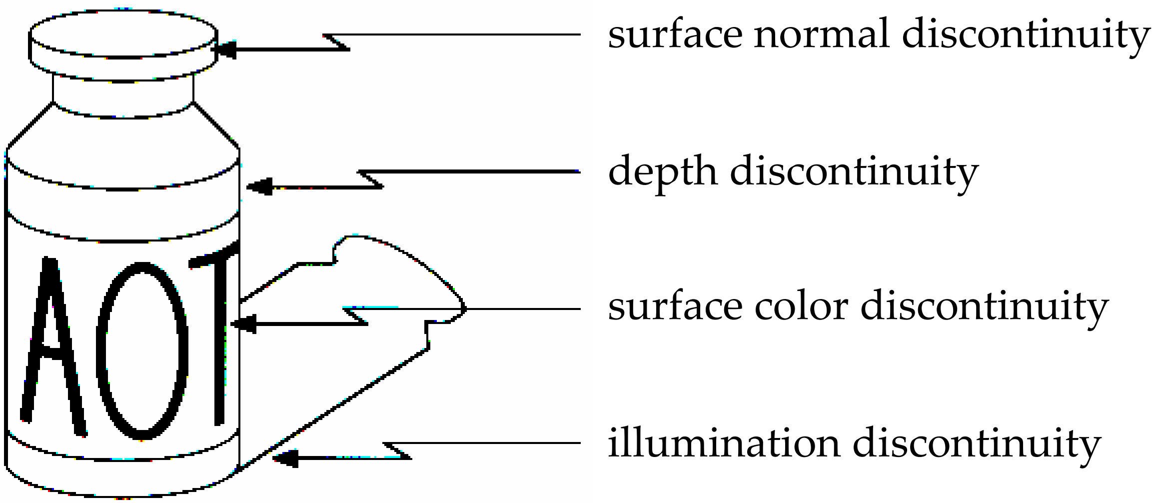
Advantages of local features

- **Locality:** features are local, so robust to occlusion and clutter (no prior segmentation)
- **Distinctiveness:** individual features can be matched to a large database of objects
- **Quantity:** many features can be generated for even small objects
- **Efficiency:** close to real-time performance
- **Extensibility:** can easily be extended to wide range of differing feature types, with each adding robustness

More motivation…

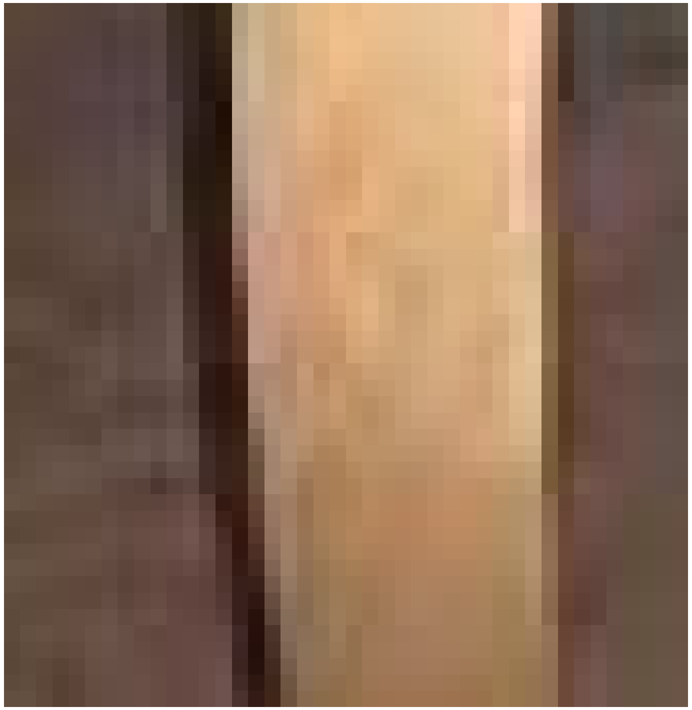
- Feature points are used also for:
  - Image alignment (图像配准/对齐)
  - 3D reconstruction(三维重构)
  - Motion tracking(运动跟踪)
  - Object recognition(目标识别)
  - Indexing and database retrieval(信息检索)
  - Robot vision(机器人视觉)
  - Others……

Origin of Edges

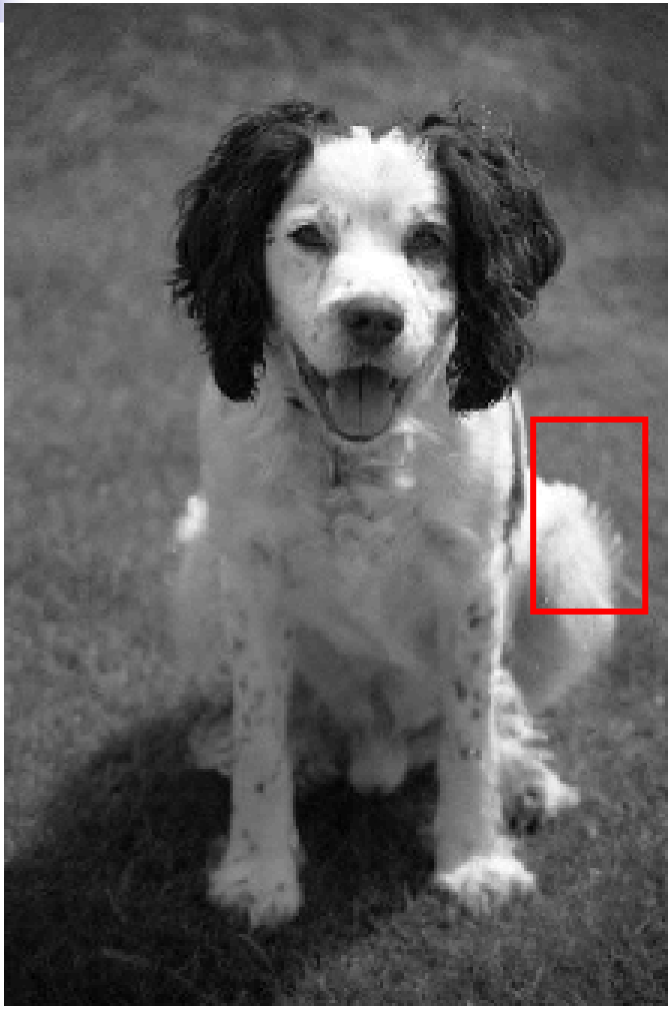


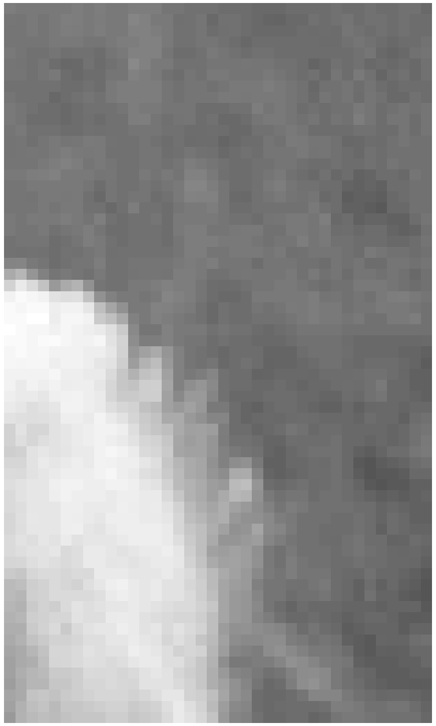
- Edges are caused by a variety of factors

We also get:Boundaries of surfaces

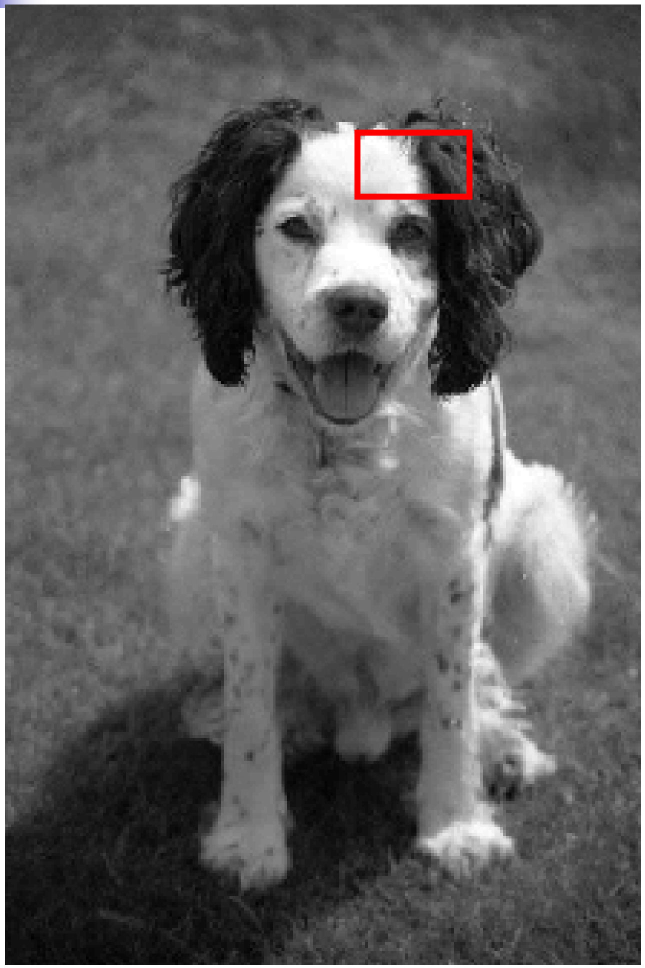


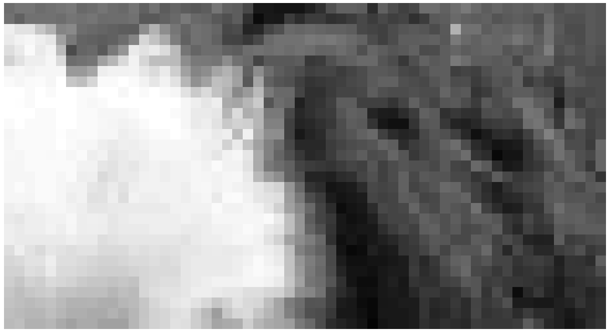
Boundaries of depths



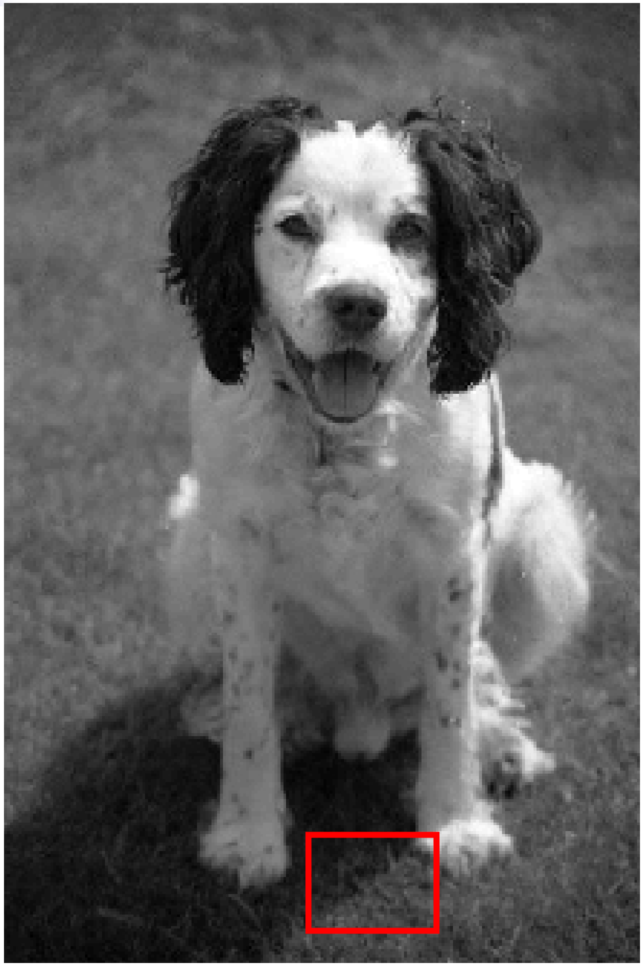


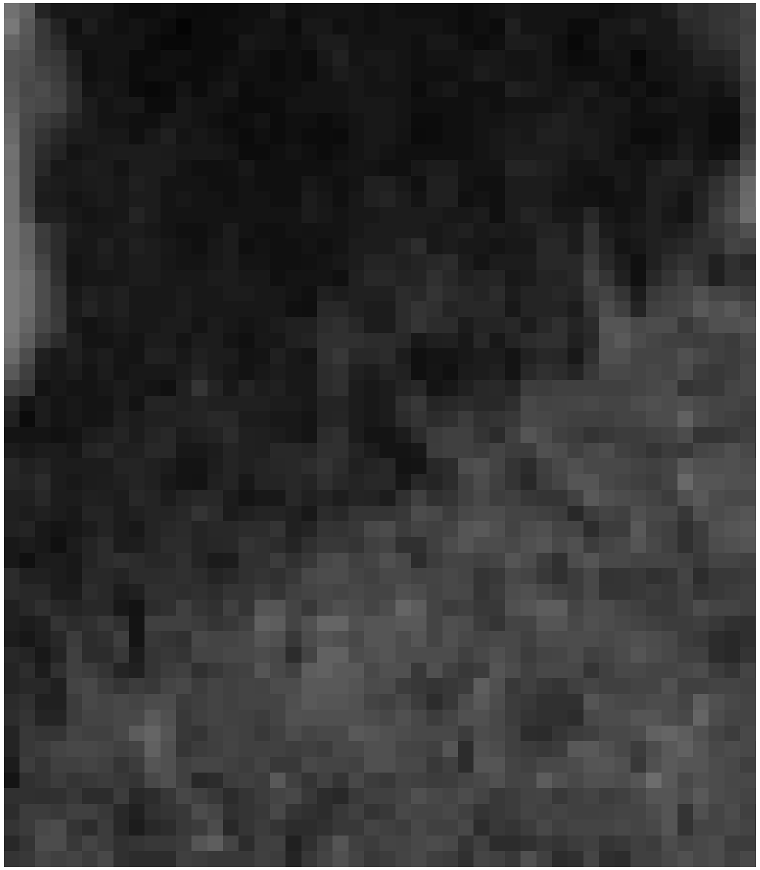
Boundaries of materials properties



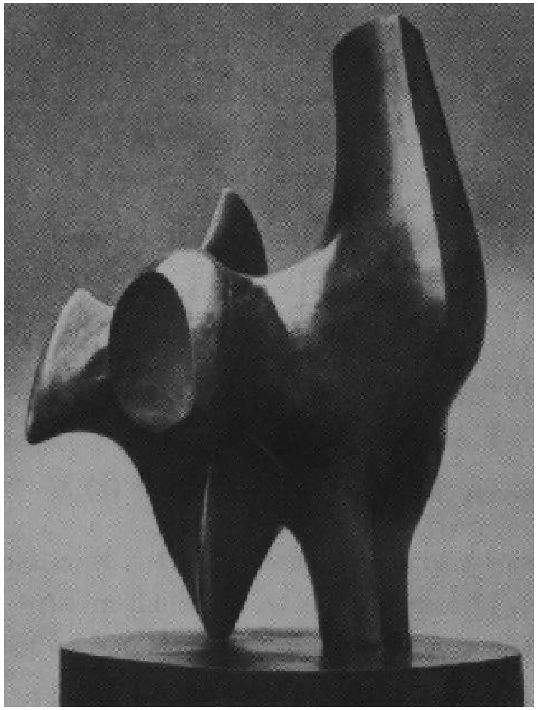


Boundaries of lighting



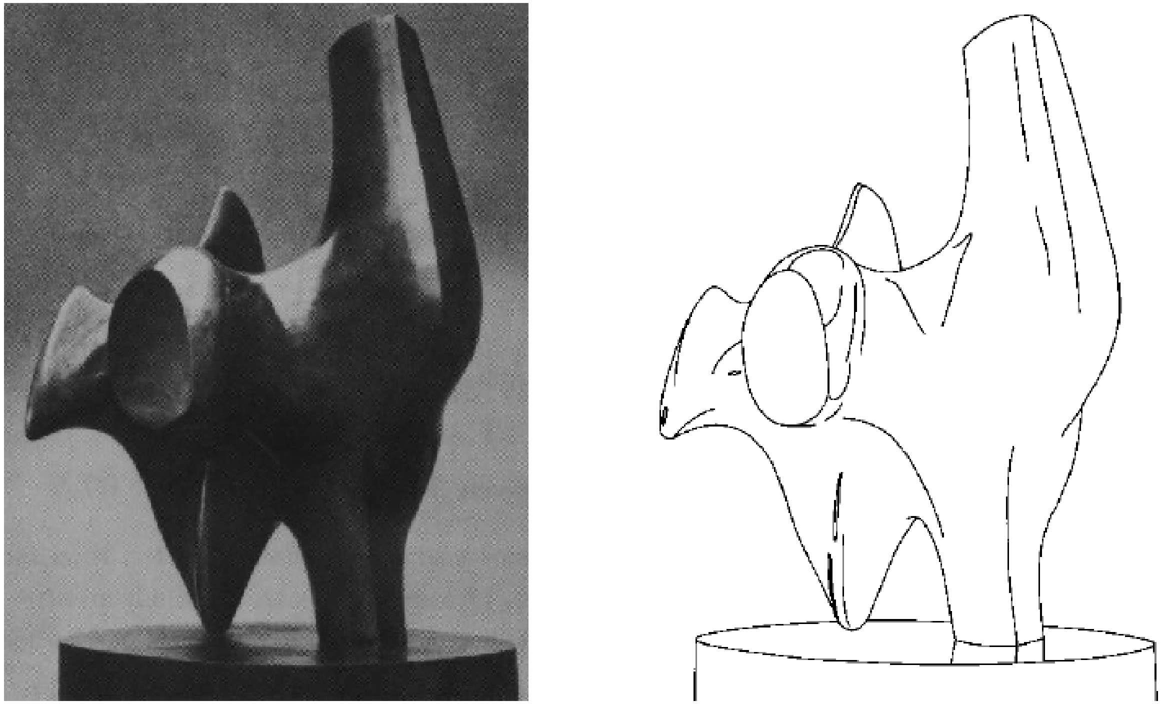


Edge Detection



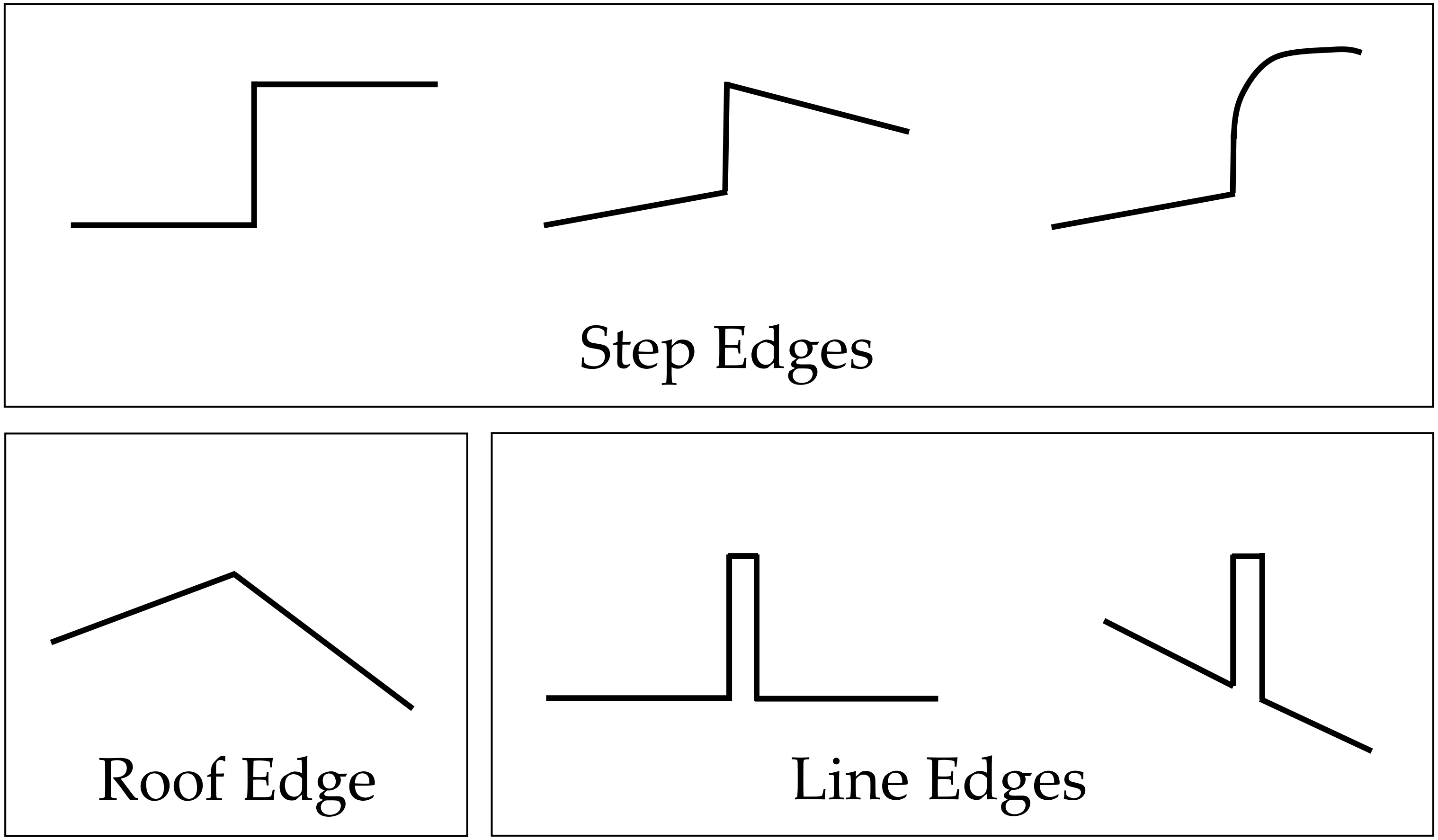
- Convert a 2D image into a set of curves
  - Extracts *salient features* of the scene
  - More compact than pixels

Edge Detection

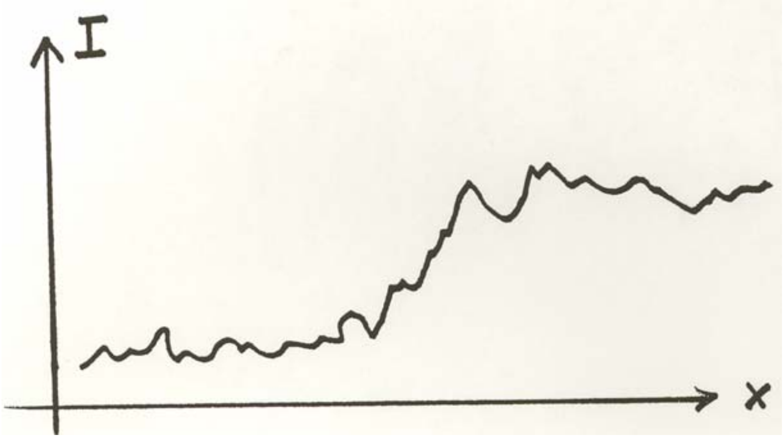


- How can you tell that a pixel is on an edge?

Edge Types



Real Edges



Noisy and Discrete!

We want an **Edge Operator** that produces:

- Edge Magnitude
- Edge Orientation
- High Detection Rate and Good Localization

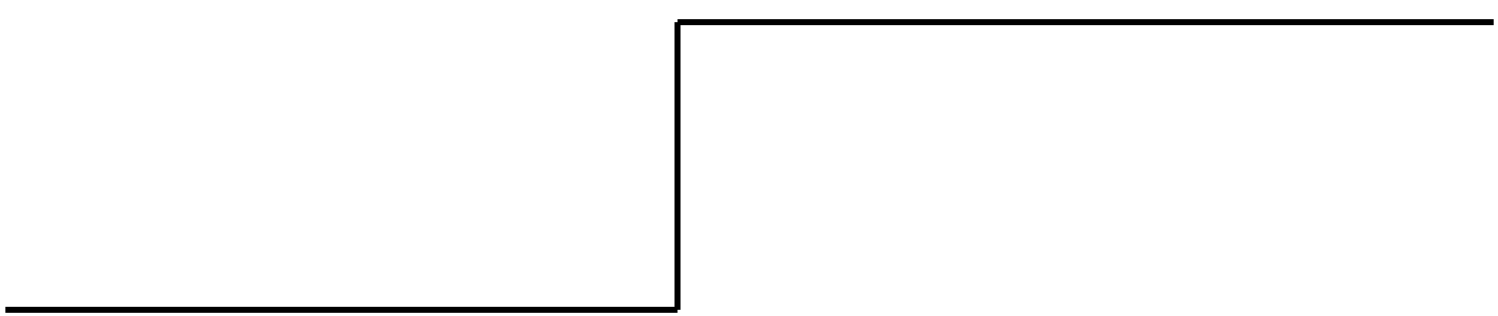
Edge Detection Continued

Boundary Detection – Edges

- Boundaries of objects
  - Usually different materials/orientations, intensity changes.

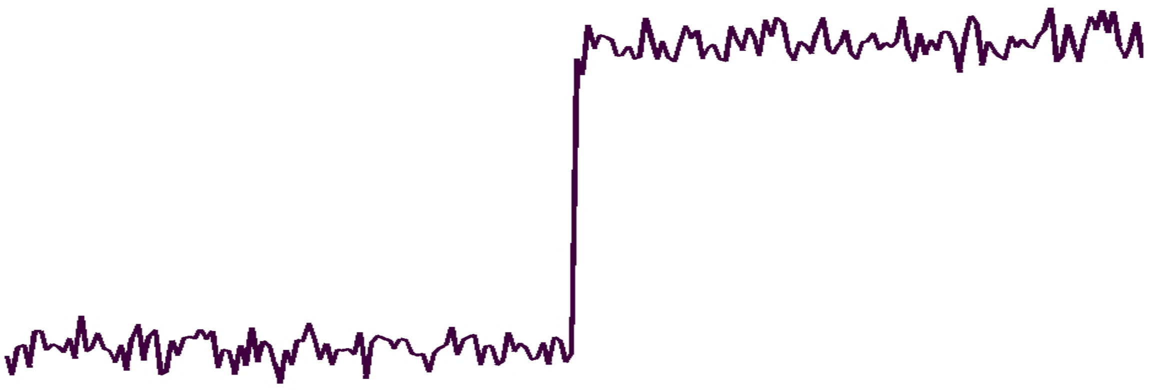
Edge is Where Change Occurs

- Change is measured by derivative in 1D
- Biggest change, derivative has maximum magnitude
- Or 2nd derivative is zero.



Noisy Step Edge

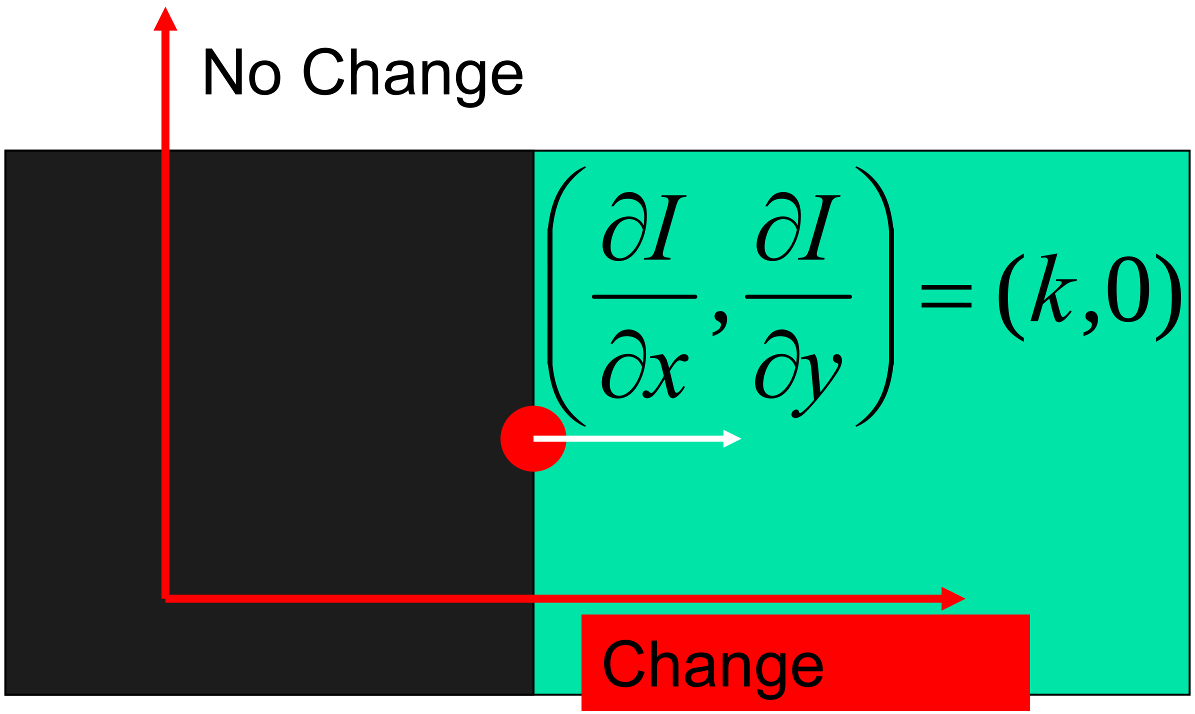
- Gradient is high everywhere.
- Must smooth before taking gradient.



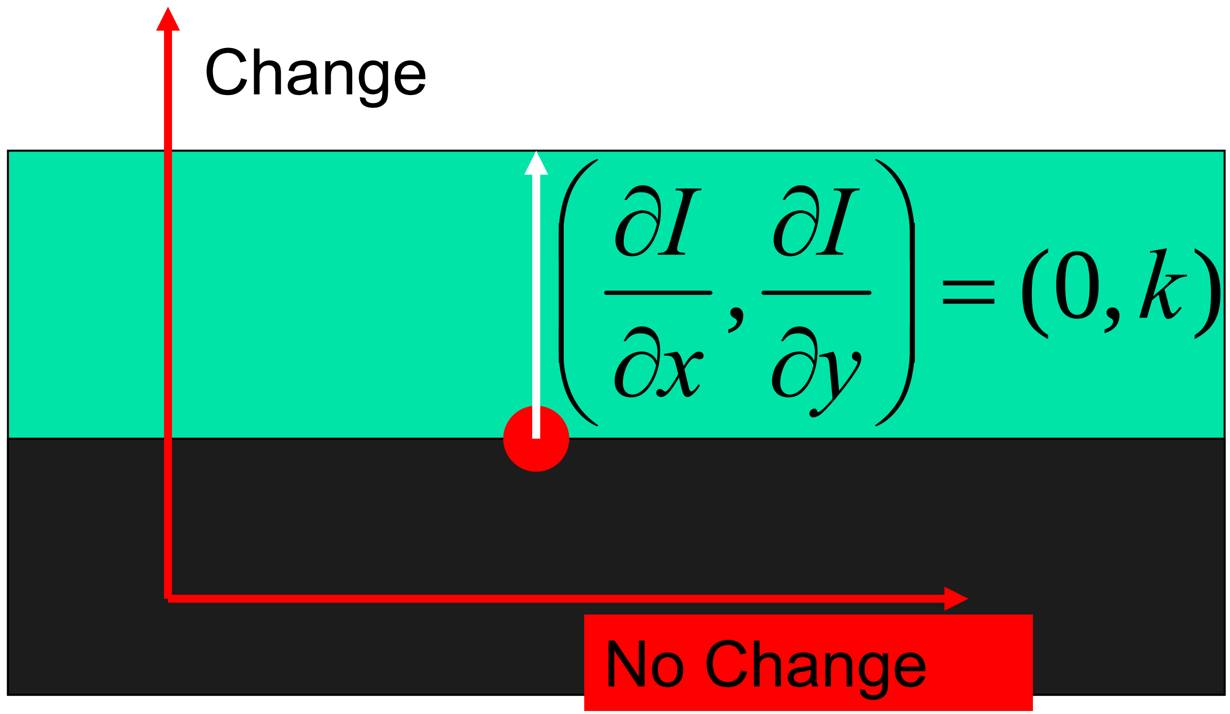
So, 1D Edge Detection has steps:

1. Filter out noise: convolve with Gaussian
2. Take a derivative: convolve with [-1 0 1]
3. Find the peak. Two issues:
   - Should be a local maximum.
   - Should be sufficiently high.

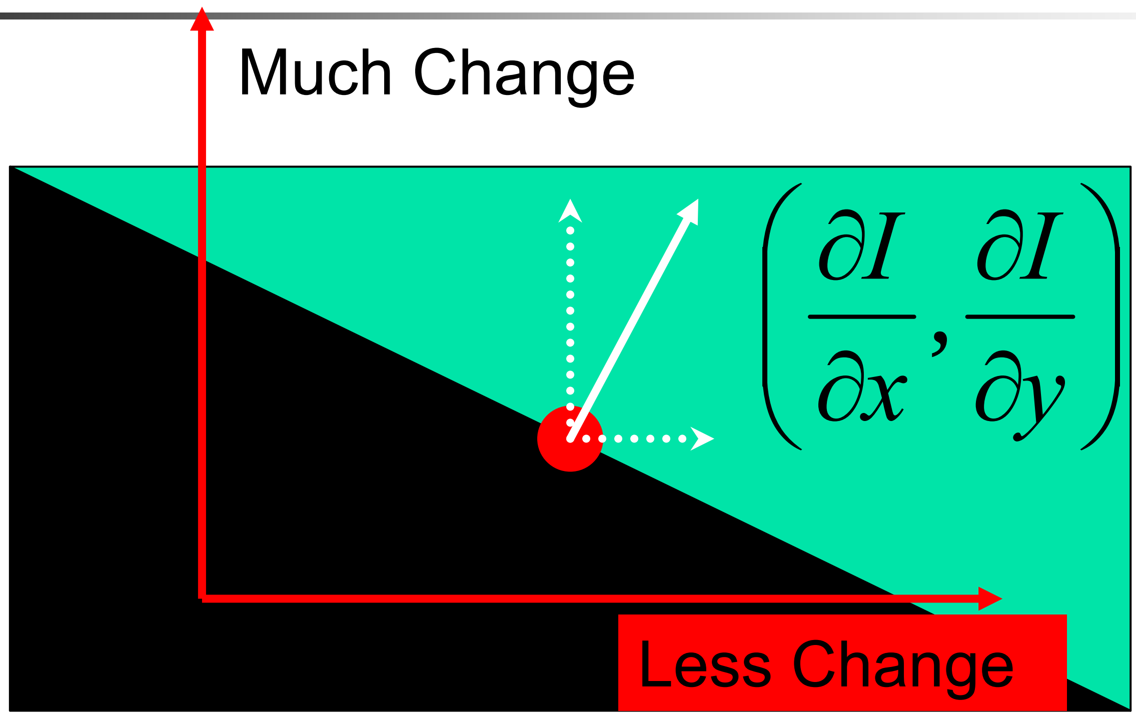
What is the gradient?



What is the gradient?



What is the gradient?



$$\left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right) = (k_x, k_y)$$

Gradient direction is perpendicular to edge.

Gradient Magnitude measures edge strength.

Detecting Discontinuities

- Image derivatives

$$\frac{\partial f}{\partial x} = \lim_{\varepsilon \to 0} \left(\frac{f(x+\varepsilon) - f(x)}{\varepsilon} \right) \Rightarrow \frac{\partial f}{\partial x} \approx \frac{f(x_{n+1}) - f(x)}{\Delta x}$$

- Convolve image with derivative filters

Backward difference	[-1 1]
Forward difference	[1 -1]
Central difference	[-1 0 1]

Derivative in Two-Dimensions

- Definition

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

- Approximation

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x_{n+1}, y_m) - f(x_n, y_m)}{\Delta x} \qquad \frac{\partial f(x,y)}{\partial y} \approx \frac{f(x_n, y_{m+1}) - f(x_n, y_m)}{\Delta x}$$

- Convolution kernels

$$f_x = \begin{bmatrix} 1 & -1 \end{bmatrix} \qquad f_y = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

Discrete Edge Operators

How can we differentiate a ***digital*** image?

Finite difference approximations:

Gradients:

$$\frac{\partial I}{\partial x} \approx \frac{1}{2\varepsilon}\left(\left(I_{i+1,j+1} - I_{i,j+1}\right) + \left(I_{i+1,j} - I_{i,j}\right)\right)$$

$$\frac{\partial I}{\partial y} \approx \frac{1}{2\varepsilon}\left(\left(I_{i+1,j+1} - I_{i+1,j}\right) + \left(I_{i,j+1} - I_{i,j}\right)\right)$$

$I_{i,j+1}$	$I_{i+1,j+1}$
$I_{i,j}$	$I_{i+1,j}$

ε

Convolution (cross-correlation) masks :

$$\frac{\partial I}{\partial x} \approx \frac{1}{2\varepsilon}$$

-1	1
-1	1

$$\frac{\partial I}{\partial y} \approx \frac{1}{2\varepsilon}$$

1	1
-1	-1

Discrete Edge Operators

2nd order partial derivatives:

$$\frac{\partial^2 I}{\partial x^2} \approx \frac{1}{\varepsilon^2}\left(I_{i-1,j} - 2I_{i,j} + I_{i+1,j}\right)$$

$$\frac{\partial^2 I}{\partial y^2} \approx \frac{1}{\varepsilon^2}\left(I_{i,j-1} - 2I_{i,j} + I_{i,j+1}\right)$$

$I_{i-1,j+1}$	$I_{i,j+1}$	$I_{i+1,j+1}$
$I_{i-1,j}$	$I_{i,j}$	$I_{i+1,j}$
$I_{i-1,j-1}$	$I_{i,j-1}$	$I_{i+1,j-1}$

Laplacian :

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

Convolution (cross-correlation) masks :

$$\nabla^2 I \approx \frac{1}{\varepsilon^2}\begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \quad \text{or} \quad \frac{1}{6\varepsilon^2}\begin{array}{|c|c|c|}\hline 1 & 4 & 1 \\ \hline 4 & -20 & 4 \\ \hline 1 & 4 & 1 \\ \hline \end{array}$$

The Sobel Operator

- Better approximations of the gradients exist
  - The *Sobel* operators below are very commonly used

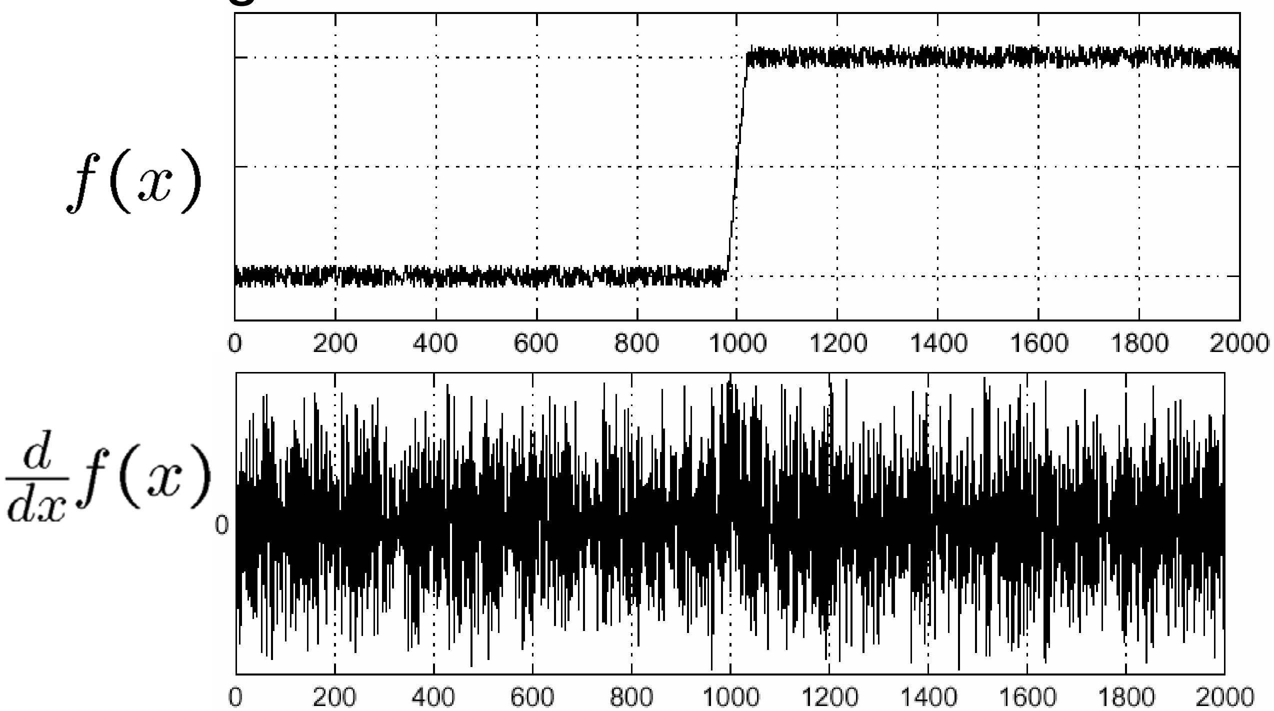
$$\frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad \frac{1}{8}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

$$s_x \qquad\qquad\qquad s_y$$

  - The standard defn. of the Sobel operator omits the 1/8 term
    - doesn't make a difference for edge detection
    - the 1/8 term **is** needed to get the right gradient value

Effects of noise

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



- Where is the edge?

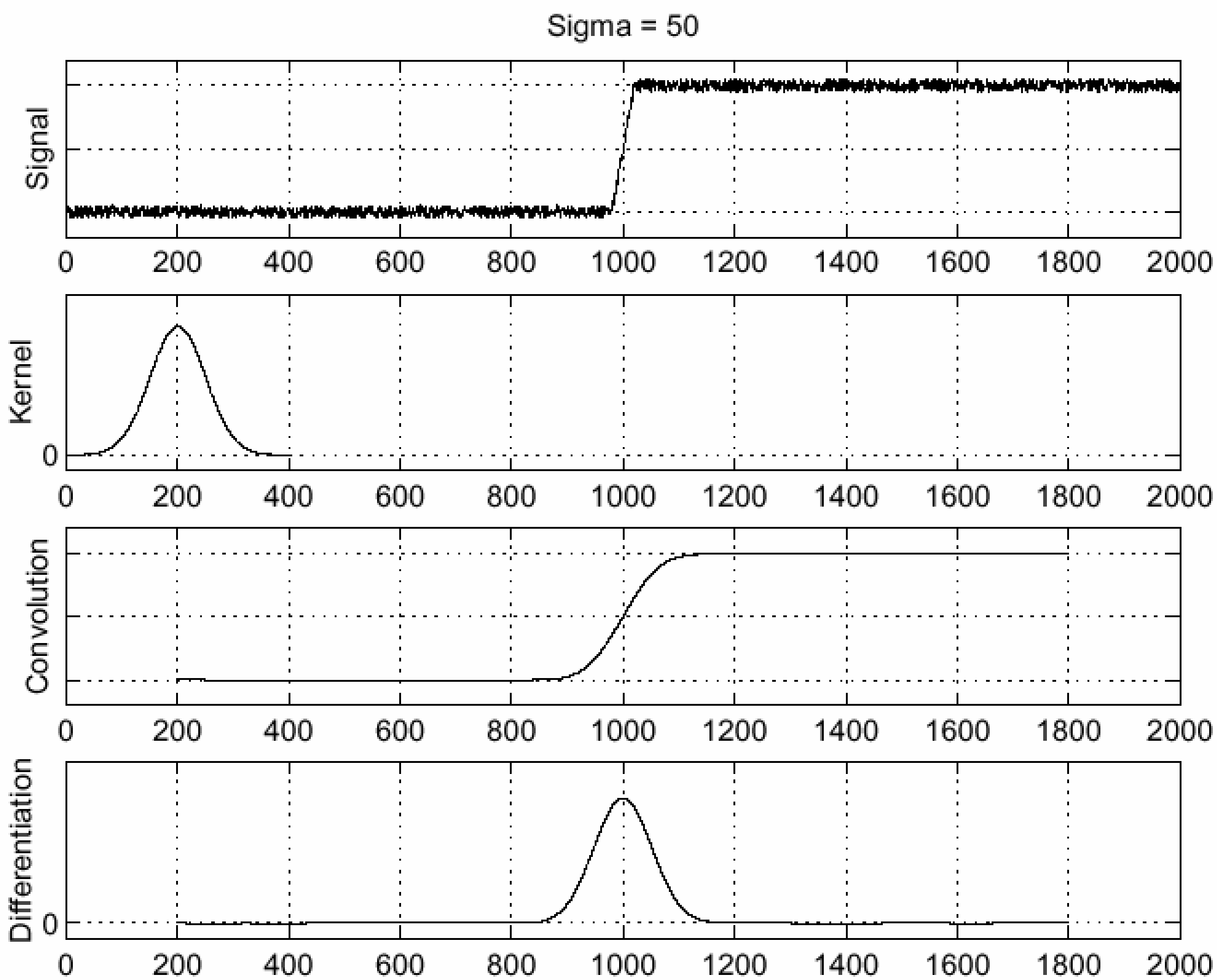
Solution: smooth first

f

h

$h \star f$

$\frac{\partial}{\partial x}(h \star f)$



Where is the edge? Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Derivative theorem of convolution

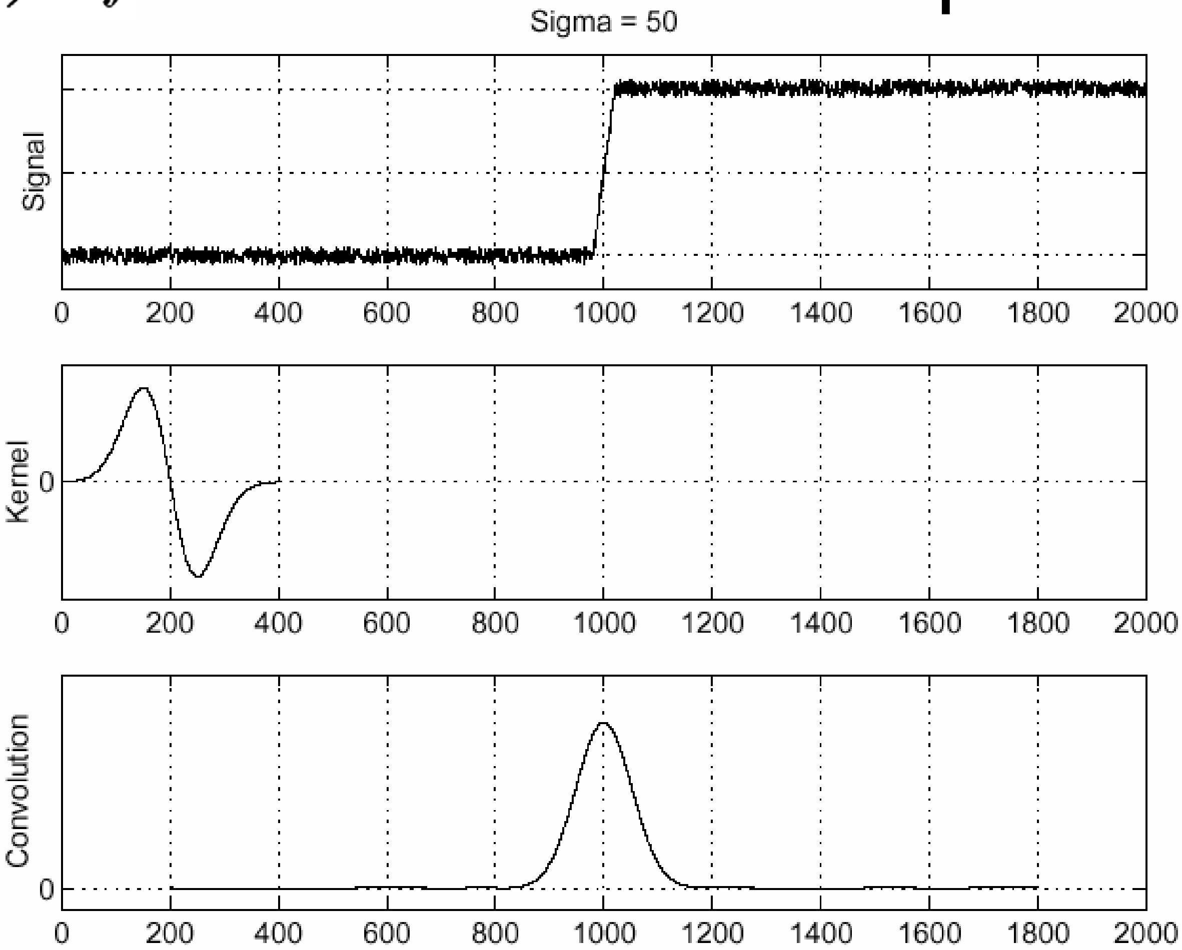
$$\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$$

This saves us one operation:

f

$\frac{\partial}{\partial x}h$

$(\frac{\partial}{\partial x}h) \star f$



Laplacian of Gaussian

$$\frac{\partial^2}{\partial x^2}(h * f) = \boxed{\left(\frac{\partial^2}{\partial x^2} h\right)} * f$$

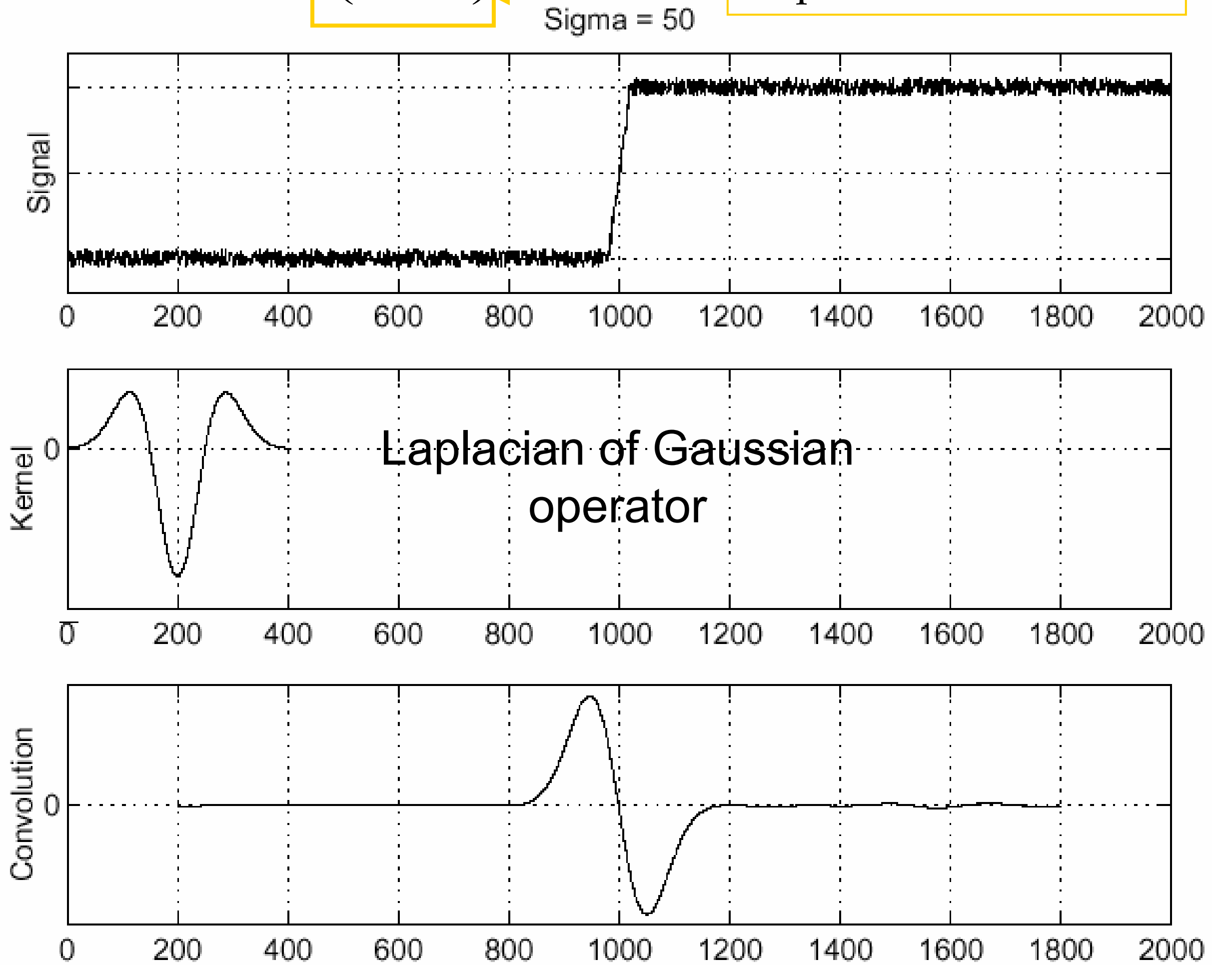
Laplacian of Gaussian

Consider $\frac{\partial^2}{\partial x^2}(h \star f)$

f

$\frac{\partial^2}{\partial x^2} h$

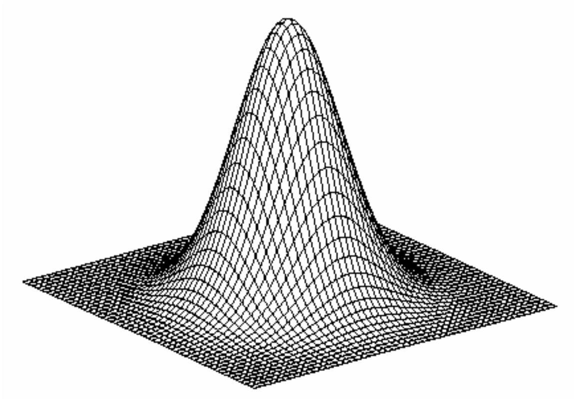
$(\frac{\partial^2}{\partial x^2} h) \star f$



Where is the edge?

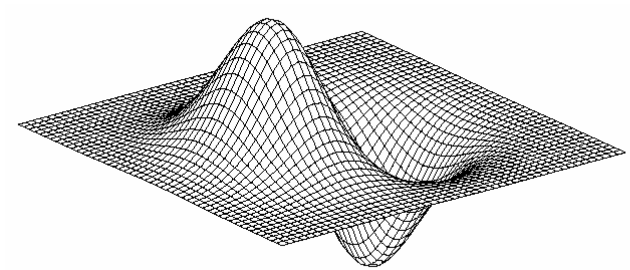
Zero-crossings of bottom graph

2D edge detection filters



Gaussian

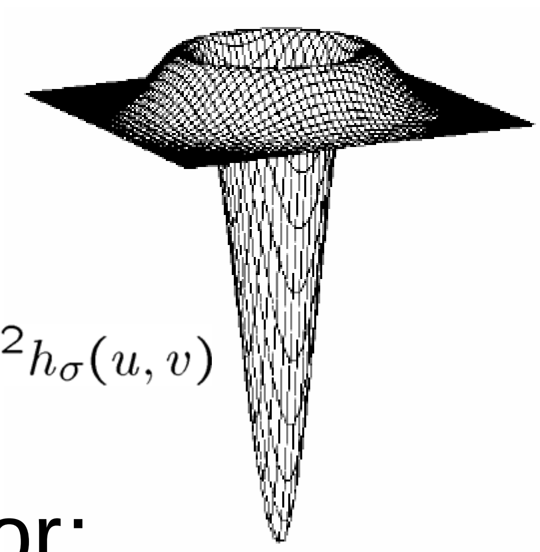
$$h_\sigma(u,v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{2\sigma^2}}$$



derivative of Gaussian

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

Laplacian of Gaussian



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

- ∇^2 is the **Laplacian** operator:

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

Edge Detection

- Prewitt and Sobel edge detectors
  - Compute derivatives
    - In x and y directions
  - Find gradient magnitude
  - Threshold gradient magnitude
- Difference between Prewitt and Sobel is the derivative filters

Prewitt Edge Detector

Prewitt's edges in x direction

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} \longrightarrow I_x$$

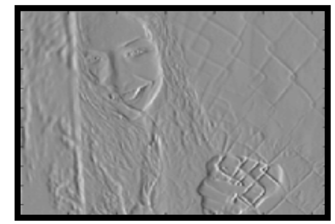
Prewitt's edges in y direction

$$\begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \longrightarrow I_y$$

Sobel Edge Detector

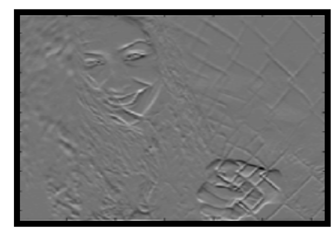
Sobel's edges in x direction

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \longrightarrow I_x$$



Sobel's edges in y direction

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \longrightarrow I_y$$



Prewitt Edge Detector

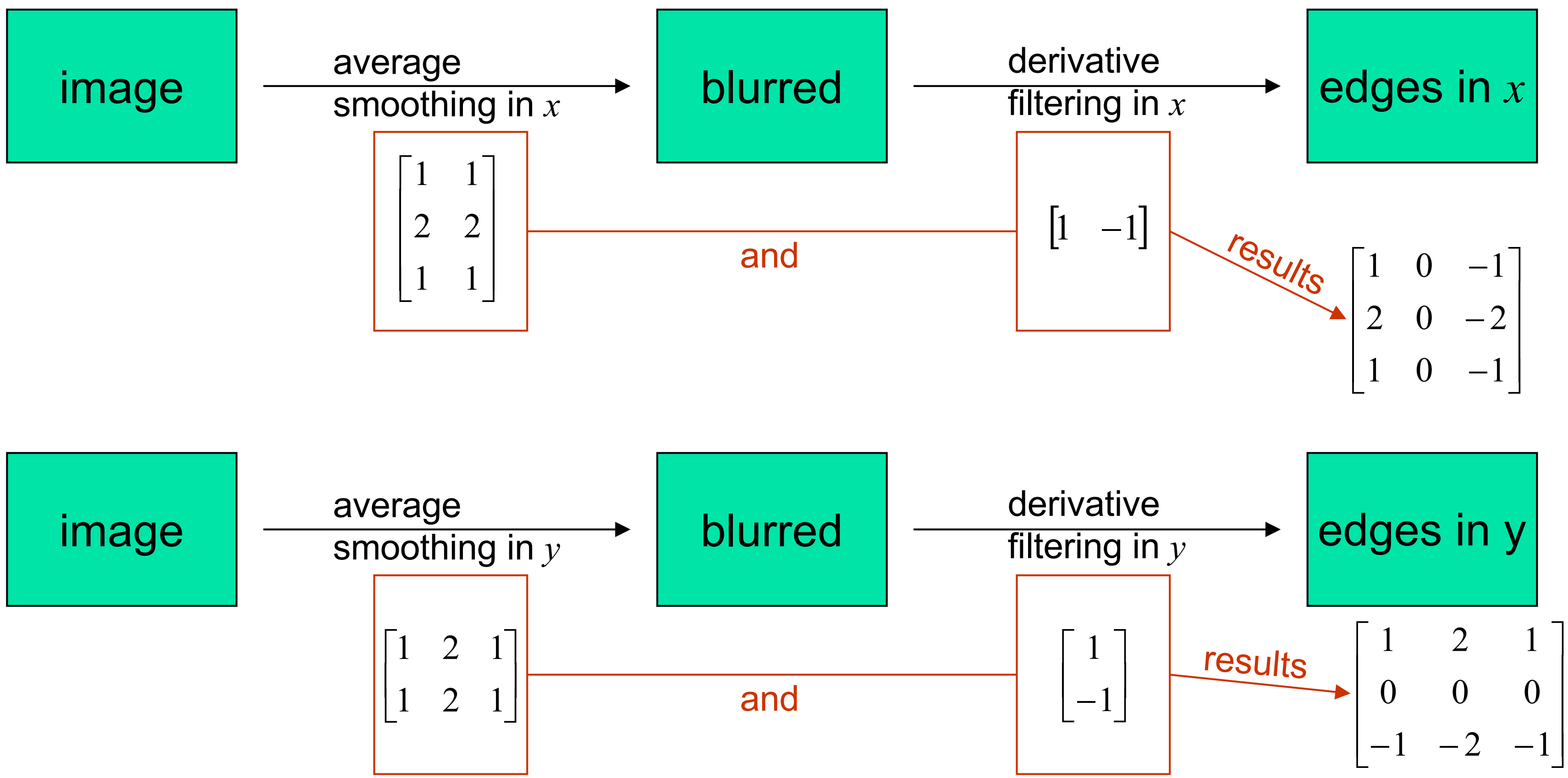
image $\xrightarrow[\text{smoothing in } x]{\text{average}}$ blurred $\xrightarrow[\text{filtering in } x]{\text{derivative}}$ edges in x

$$\begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} 1 & -1 \end{bmatrix} \xrightarrow{\text{results}} \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

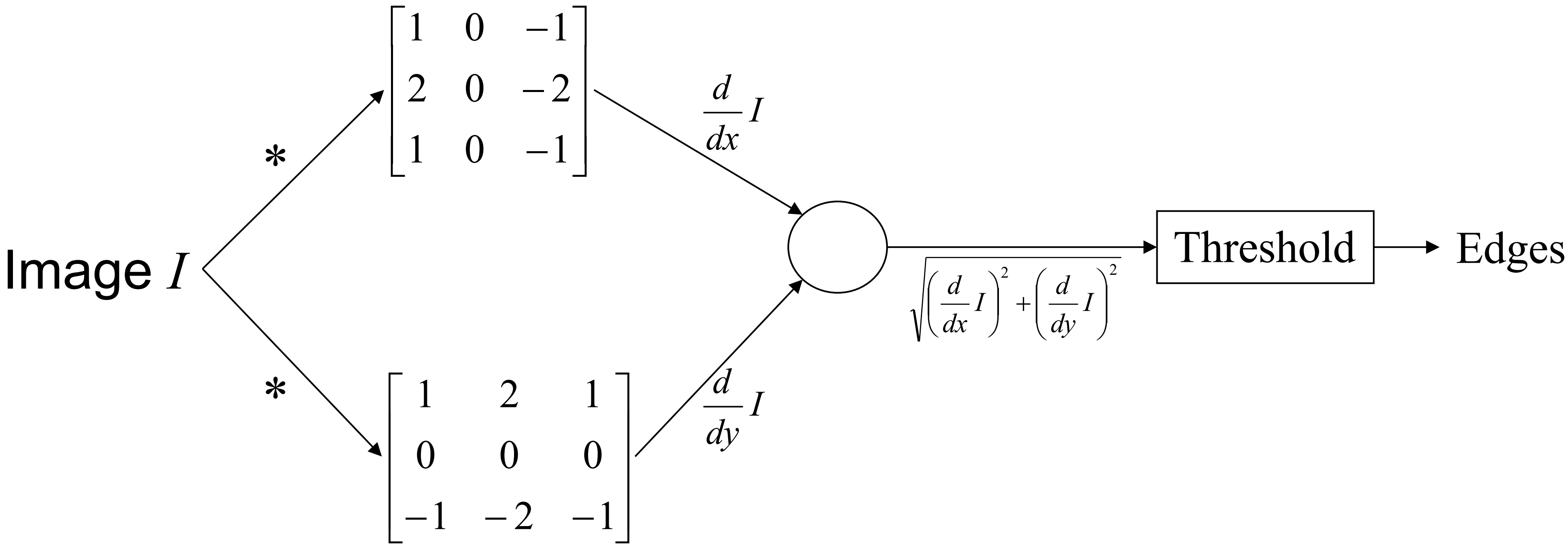
image $\xrightarrow[\text{smoothing in } y]{\text{average}}$ blurred $\xrightarrow[\text{filtering in } y]{\text{derivative}}$ edges in y

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \text{ and } \begin{bmatrix} 1 \\ -1 \end{bmatrix} \xrightarrow{\text{results}} \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

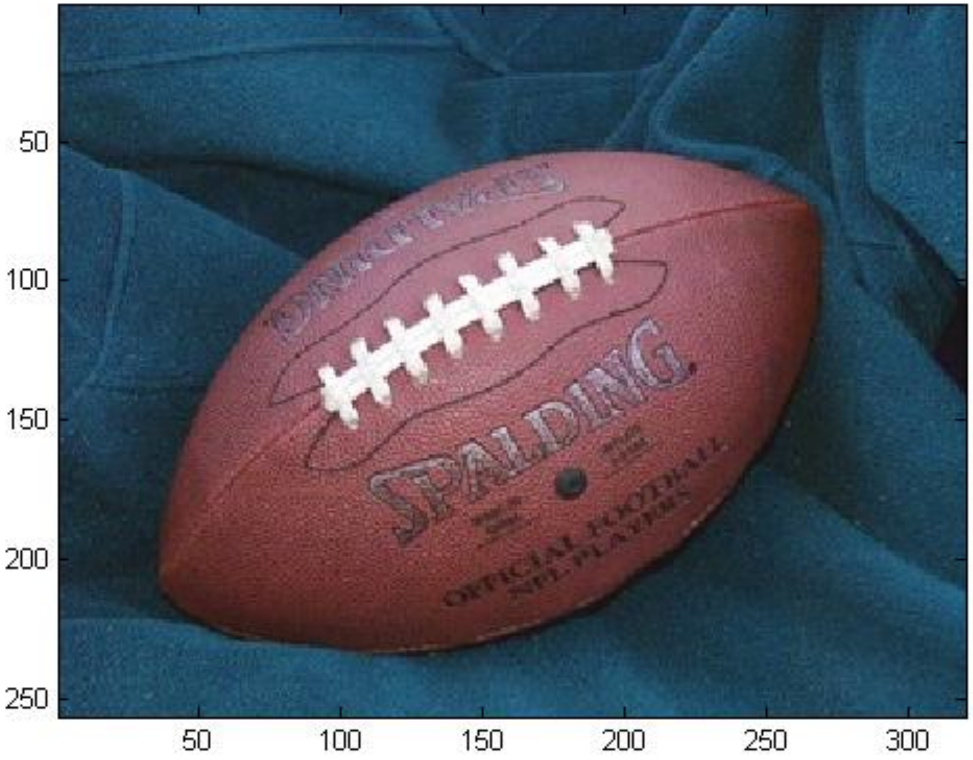
Sobel Edge Detector

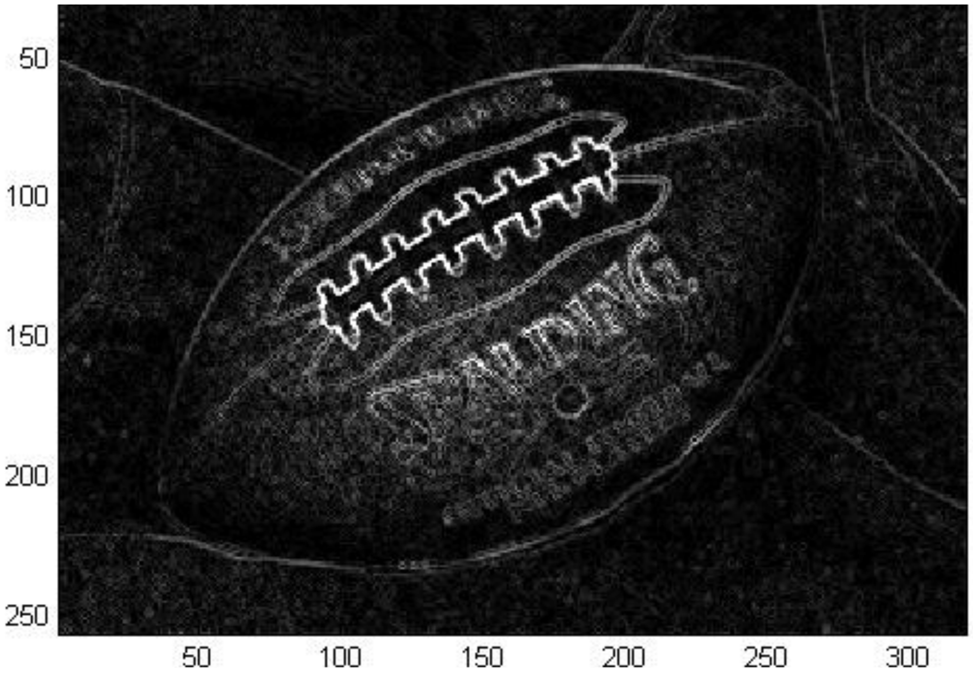


Sobel Edge Detector



Edge Finding: Matlab Demo

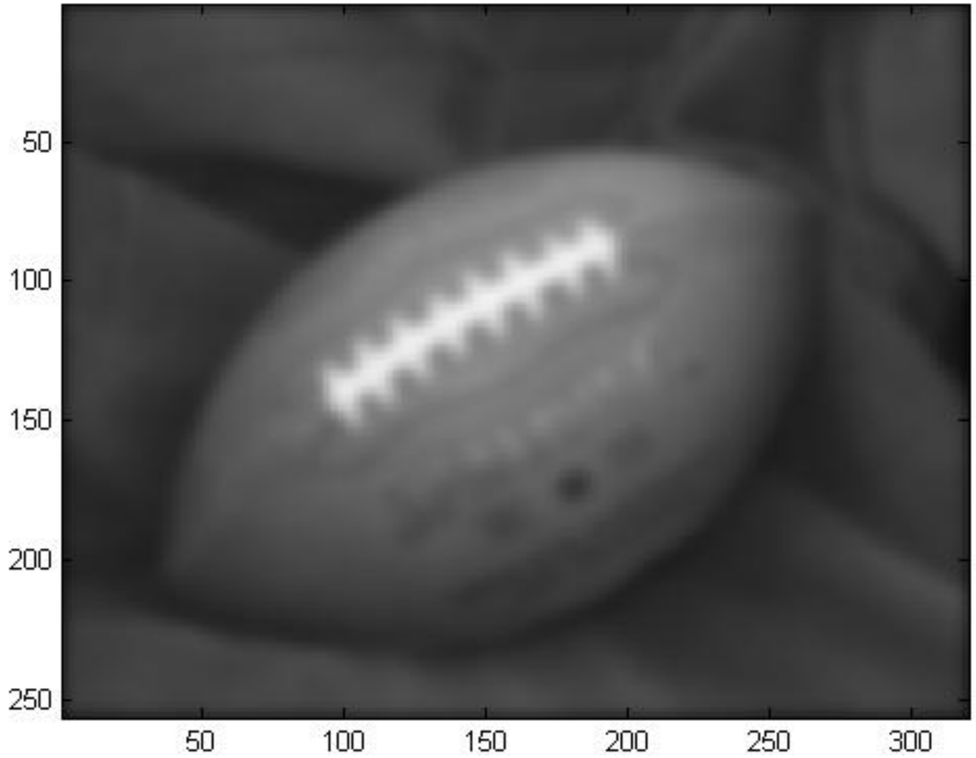





```
im = imread('football.jpg');
image(im);
figure(2);
bw = double(rgb2gray(im));

[dx,dy] = gradient(bw);
gradmag = sqrt(dx.^2 + dy.^2);
image(gradmag);
```

Image Smoothing With Gaussian

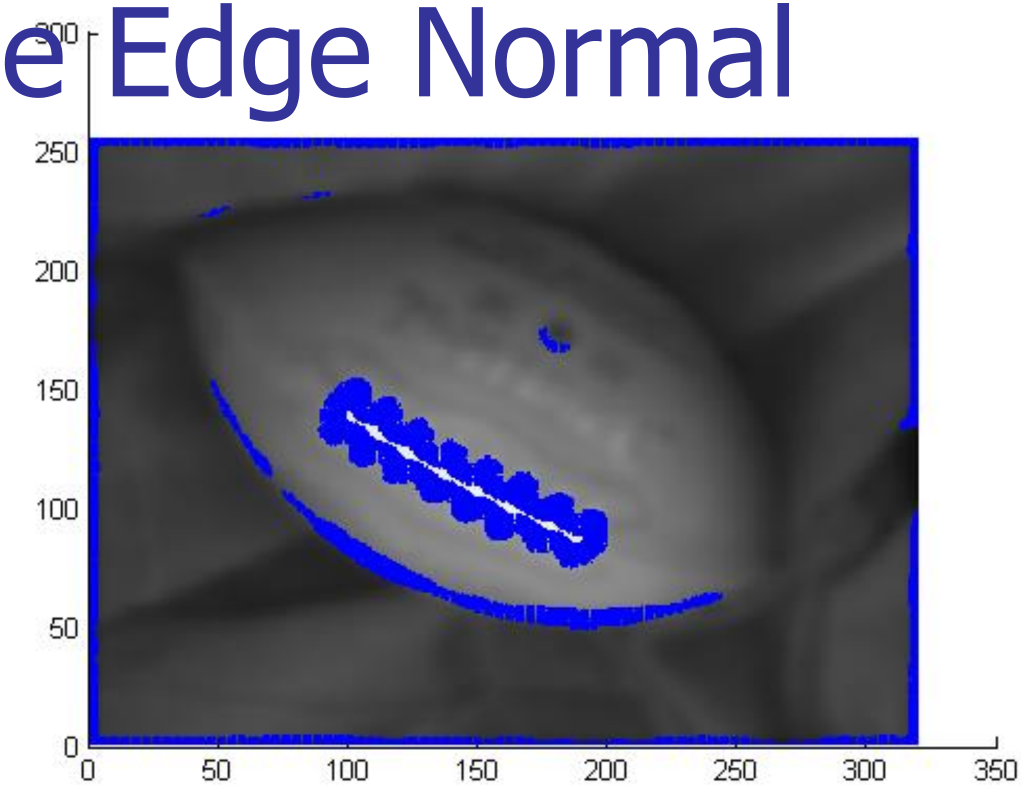



```
figure(3);
sigma = 3;
width = 3 * sigma;
support = -width : width;
gauss2D = exp( - (support / sigma).^2 / 2);
gauss2D = gauss2D / sum(gauss2D);
smooth = conv2(bw, gauss2D, 'same');
image(smooth);
colormap(gray(255));
gauss3D = gauss2D' * gauss2D;
tic ; smooth = conv2(bw,gauss3D, 'same'); toc
```

Edge Detection With Smoothed Images

```
figure(4);
[dx,dy] = gradient(smooth);
gradmag = sqrt(dx.^2 + dy.^2);
gmax = max(max(gradmag));
imshow(gradmag);
colormap(gray(gmax));
```

Displaying the Edge Normal




```
figure(5);
hold on;
image(smooth);
colormap(gray(255));
[m,n] = size(gradmag);

edges = (gradmag > 0.3 * gmax);
inds = find(edges);
[posx,posy] = meshgrid(1:n,1:m); posx2=posx(inds); posy2=posy(inds);
gm2= gradmag(inds);
sintheta = dx(inds) ./ gm2;
costheta = - dy(inds) ./ gm2;
quiver(posx2,posy2, gm2 .* sintheta / 10, -gm2 .* costheta / 10,0);
hold off;
```

Sobel Operator

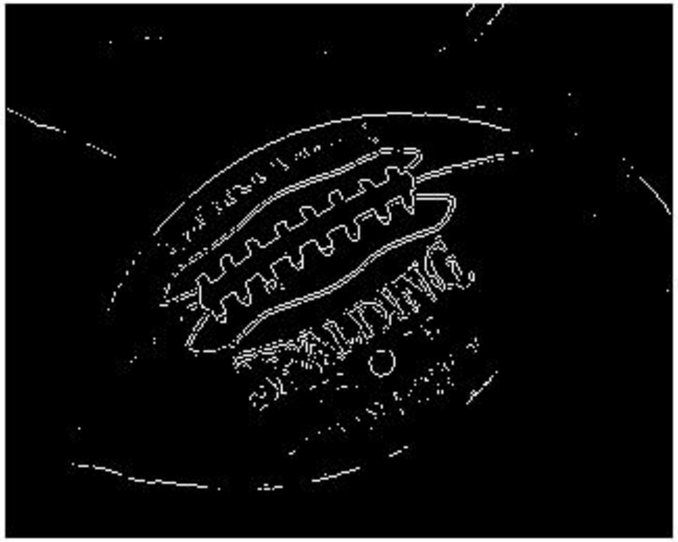
$$S_1 = \begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \qquad S_2 = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

$$\text{Edge Magnitude} = \sqrt{S_1^2 + S_1^2}$$

$$\text{Edge Direction} = \tan^{-1}\left(\frac{S_1}{S_2}\right)$$

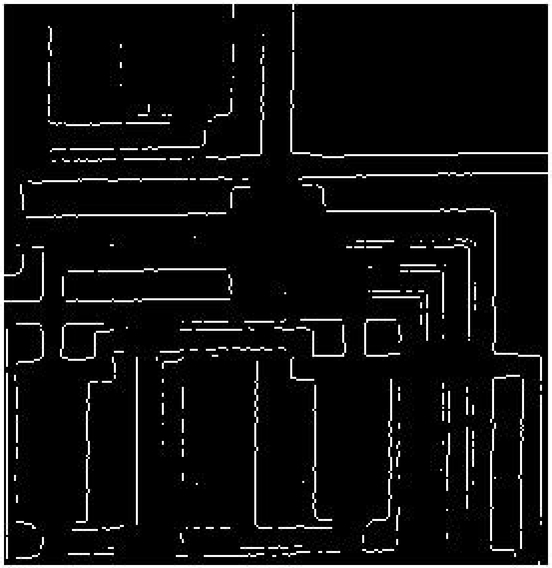
Sobel Edge Detector

```
figure(6)
edge(bw, 'sobel')
```

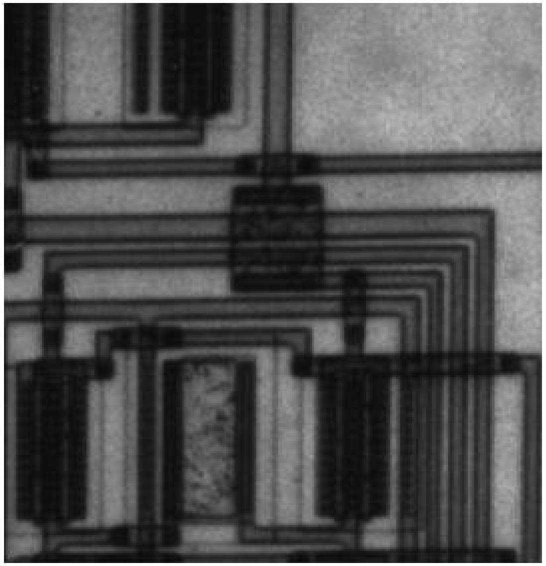


Edge detection - Matlab demo

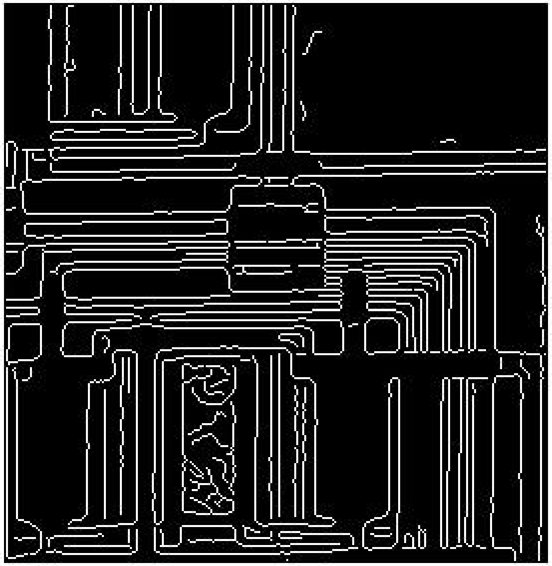
```
I = imread('circuit.tif');
imshow(I);
BW1 = edge(I,'prewitt');
BW2 = edge(I,'canny');
Figure;
imshow(BW1);
Figure;
imshow(BW2);
```



Prewitt filter



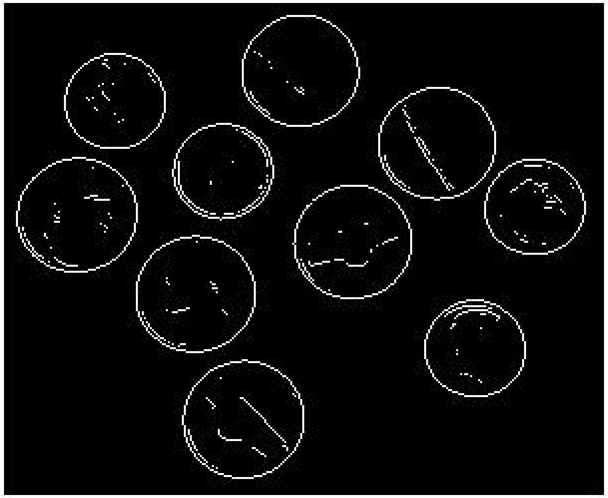
Original image



Canny filter

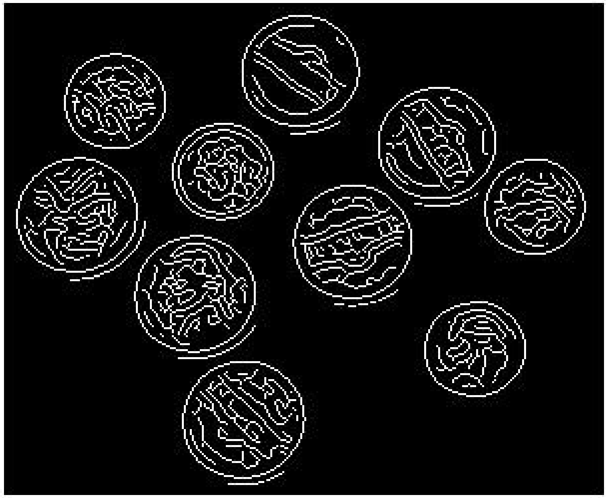
Edge detection - Matlab demo

```
I = imread('coins.png');
imshow(I);
BW1 = edge(I,'sobel');
BW2 = edge(I,'canny');
Figure;
imshow(BW1);
Figure;
imshow(BW2);
```



Sobel filter

Original image



Canny filter

Features in Matlab

edge(im,'prewitt') - (almost) linear

edge(im,'sobel') - (almost) linear

edge(im,'canny') - not local, no closed form

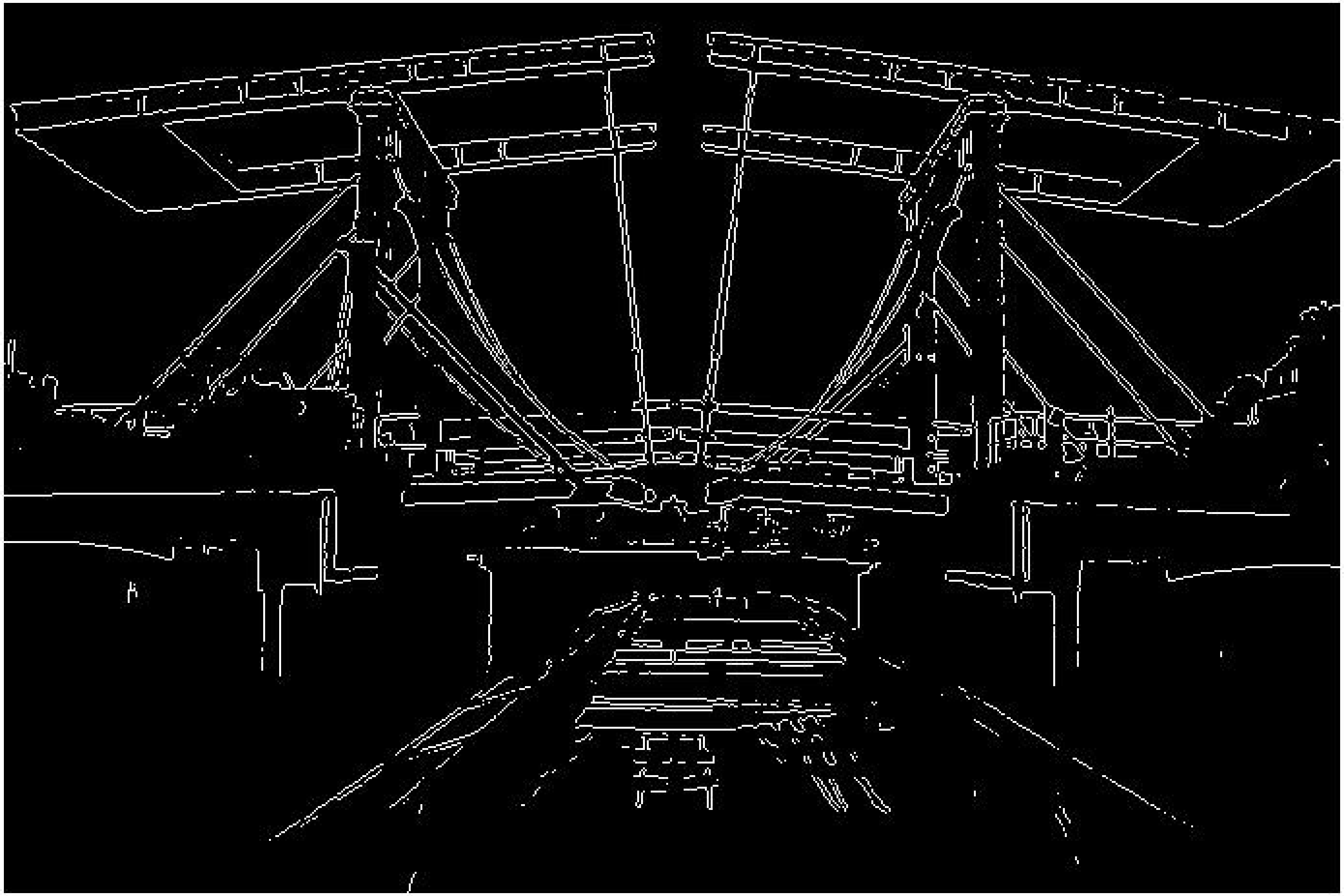
Sobel Operator

$$S_1 = \begin{array}{|c|c|c|}\hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} \qquad S_2 = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$

$$\text{Edge Magnitude} = \sqrt{S_1^2 + S_1^2}$$

$$\text{Edge Direction} = \tan^{-1}\left(\frac{S_1}{S_2}\right)$$

Sobel filter



edge(im,’sobel’)

Today’s Goals

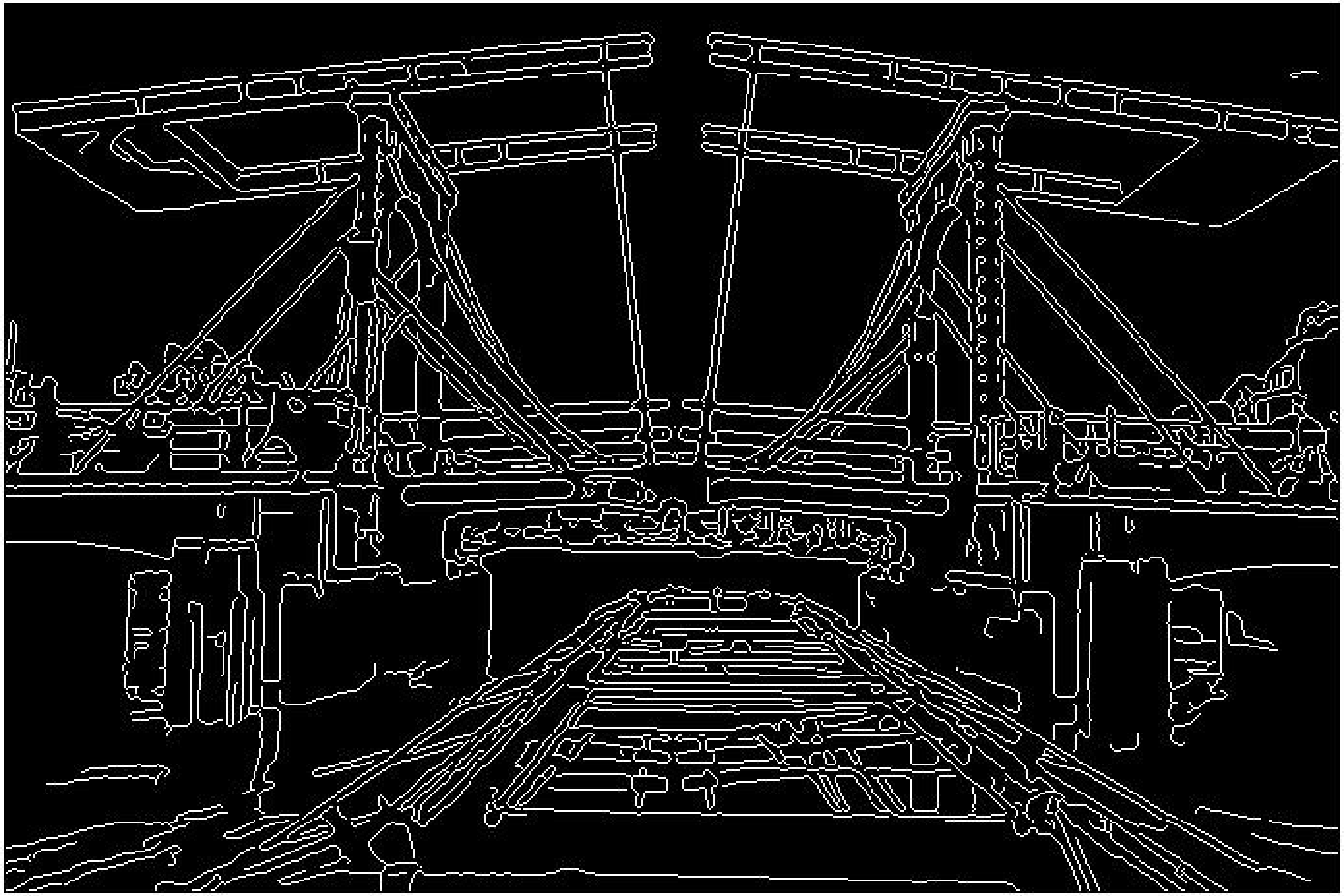
- Features Overview
- Canny Edge Detector
- Harris Corner Detector
- Templates and Image Pyramid
- SIFT Features

Canny Edge Detector

- J. Canny, "A computational approach to edge detection, " IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, pp. 679--698, 1986

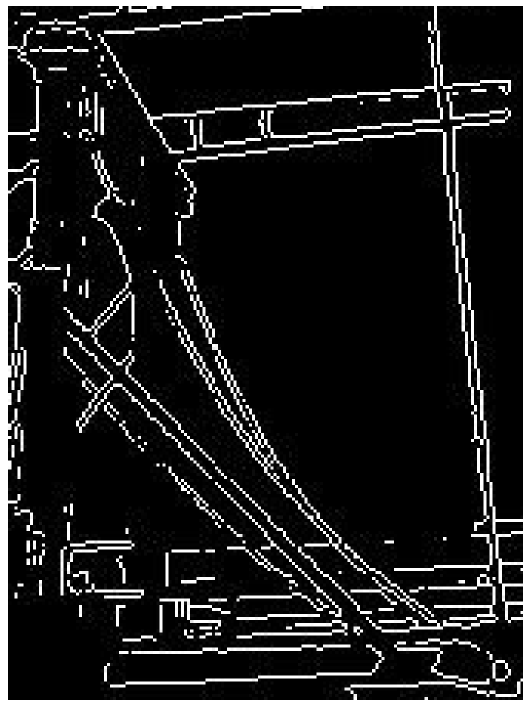
- Source code:
  - ftp://figment.csee.usf.edu/pub/Edge_Comparison/source_code/canny.src

Canny Edge Detector

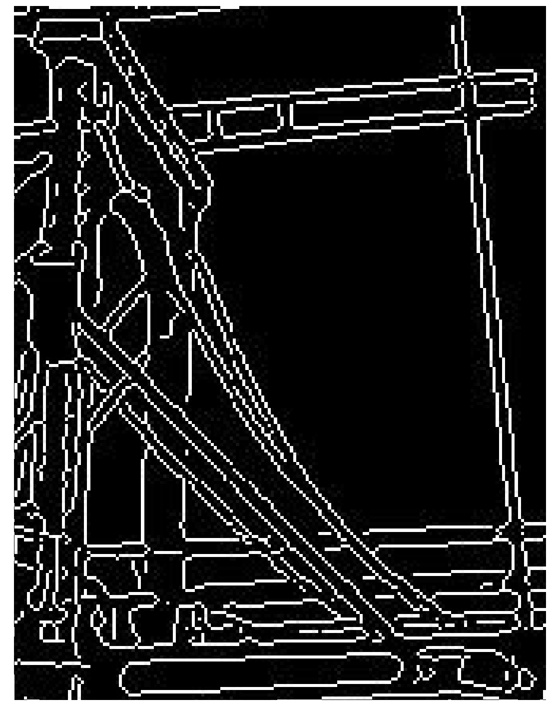


edge(im,'canny')

Comparison



Sobel



Canny

Optimal Edge Detection: Canny

- Assume:
  - Linear filtering
  - Additive iid Gaussian noise
- Edge detector should have:
  - Good Detection. Filter responds to edge, not noise.
  - Good Localization: detected edge near true edge.
  - Single Response: one per edge.

Optimal Edge Detection: Canny (continued)

- Optimal Detector is approximately Derivative of Gaussian.
- Detection/Localization trade-off
  - More smoothing improves detection
  - And hurts localization.
- This is what you might guess from (detect change) + (remove noise)

Canny Edge Detector

- Criterion 1: Good Detection: The optimal detector must minimize the probability of false positives as well as false negatives.

- Criterion 2: Good Localization: The edges detected must be as close as possible to the true edges.

- Single Response Constraint: The detector must return one point only for each edge point.

Canny Edge Detector Steps

1. Smooth image with Gaussian filter
2. Compute derivative of filtered image
3. Find magnitude and orientation of gradient
4. Apply “Non-maximum Suppression”
5. Apply “Hysteresis Threshold”

Canny Edge Detector First Two Steps

1. Filter out noise
   - Use a 2D Gaussian Filter. $J = I \otimes G$
2. Take a derivative
   - Compute the magnitude of the gradient:

$$\nabla J = (J_x, J_y) = \left(\frac{\partial J}{\partial x}, \frac{\partial J}{\partial y} \right) \text{ is the Gradient}$$

$$\|\nabla J\| = \sqrt{J_x^2 + J_y^2}$$

Canny Edge Detector
First Two Steps

- Smoothing

$$S = I * g(x,y) = g(x,y) * I \qquad g(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- Derivative

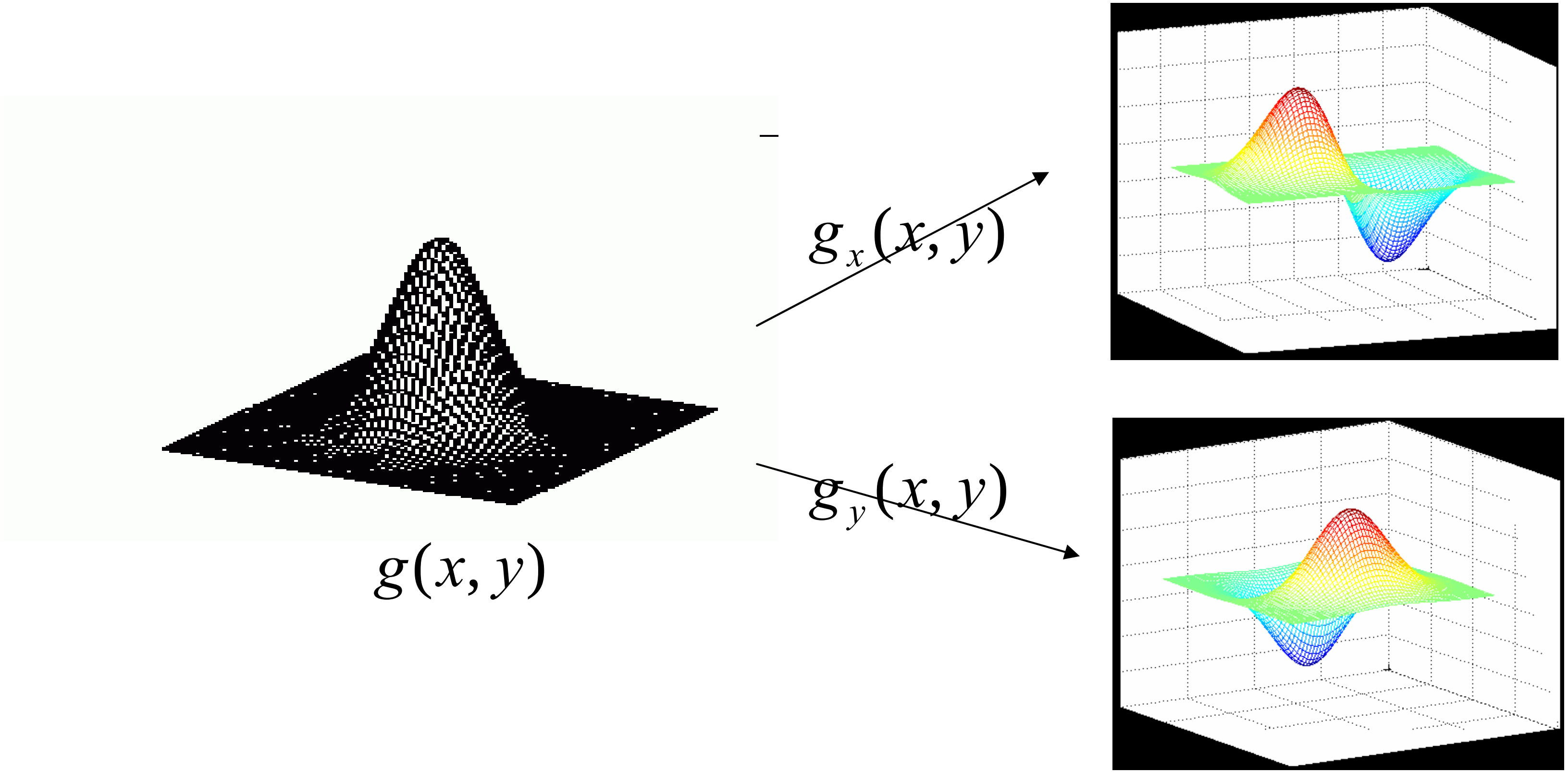
$$\nabla S = \nabla(g * I) = (\nabla g) * I$$

$$\nabla S = \begin{bmatrix} g_x \\ g_y \end{bmatrix} * I = \begin{bmatrix} g_x * I \\ g_y * I \end{bmatrix}$$

Homework

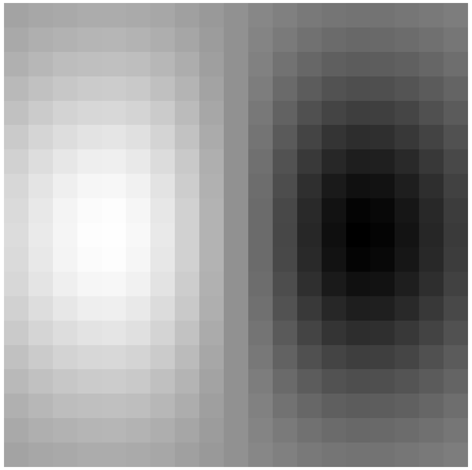
$$\nabla g = \begin{bmatrix} \frac{\partial g}{\partial x} \\ \frac{\partial g}{\partial y} \end{bmatrix} = \begin{bmatrix} g_x \\ g_y \end{bmatrix}$$

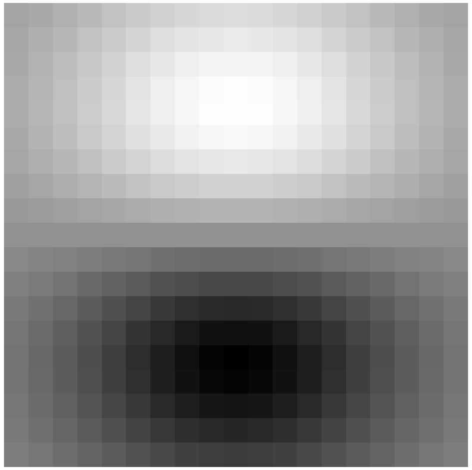
Canny Edge Detector
Derivative of Gaussian



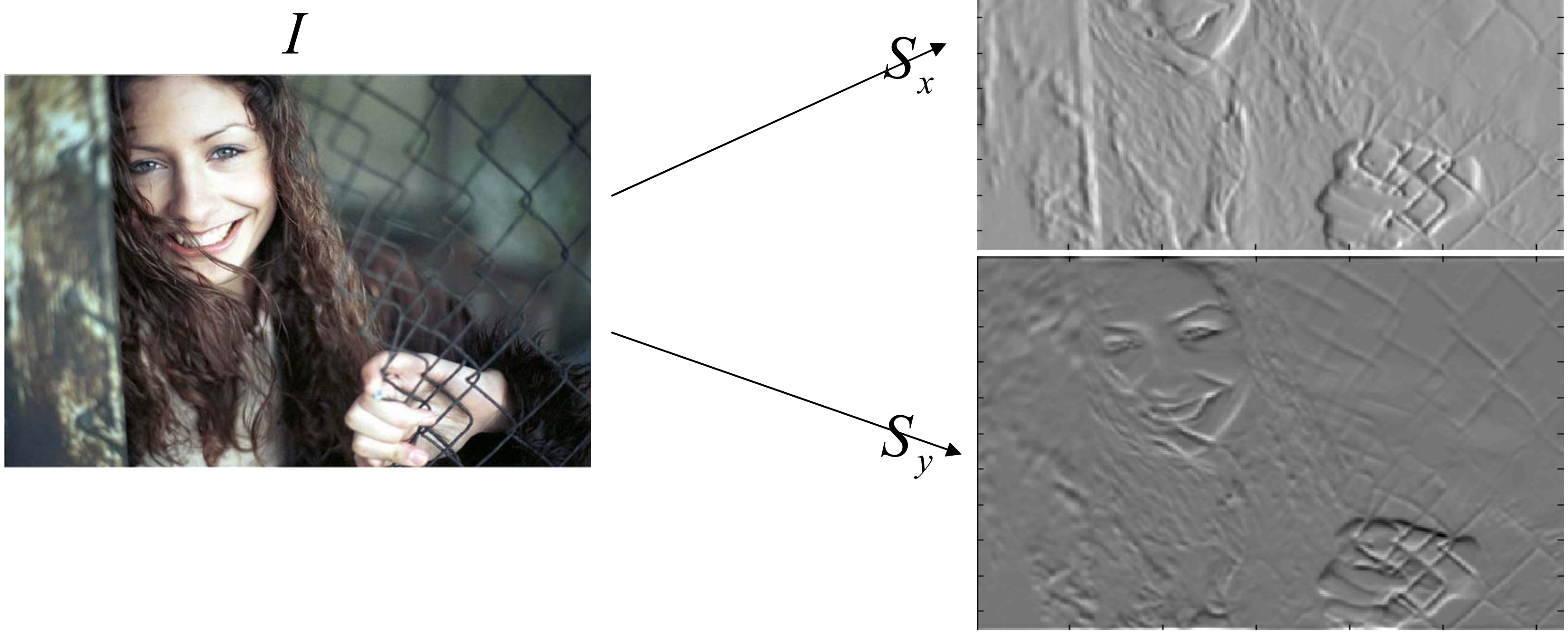
Smoothing and Differentiation

- Need two derivatives, in x and y direction.
- We can use a derivative of Gaussian filter
  - because differentiation is convolution, and convolution is associative





Canny Edge Detector
First Two Steps



Canny Edge Detector
Third Step

- Gradient magnitude and gradient direction

(S_x, S_y) Gradient Vector

magnitude $= \sqrt{(S_x^2 + S_y^2)}$

direction $= \theta = \tan^{-1} \dfrac{S_y}{S_x}$

image

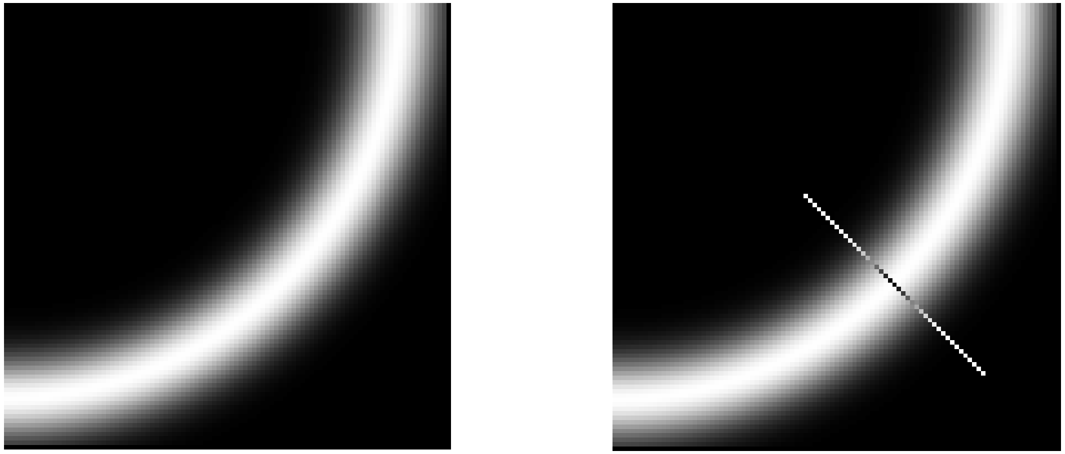
gradient magnitude

Finding the Peak

1) The gradient magnitude is large along thick trail; how do we identify the significant points?
2) How do we link the relevant points up into curves?

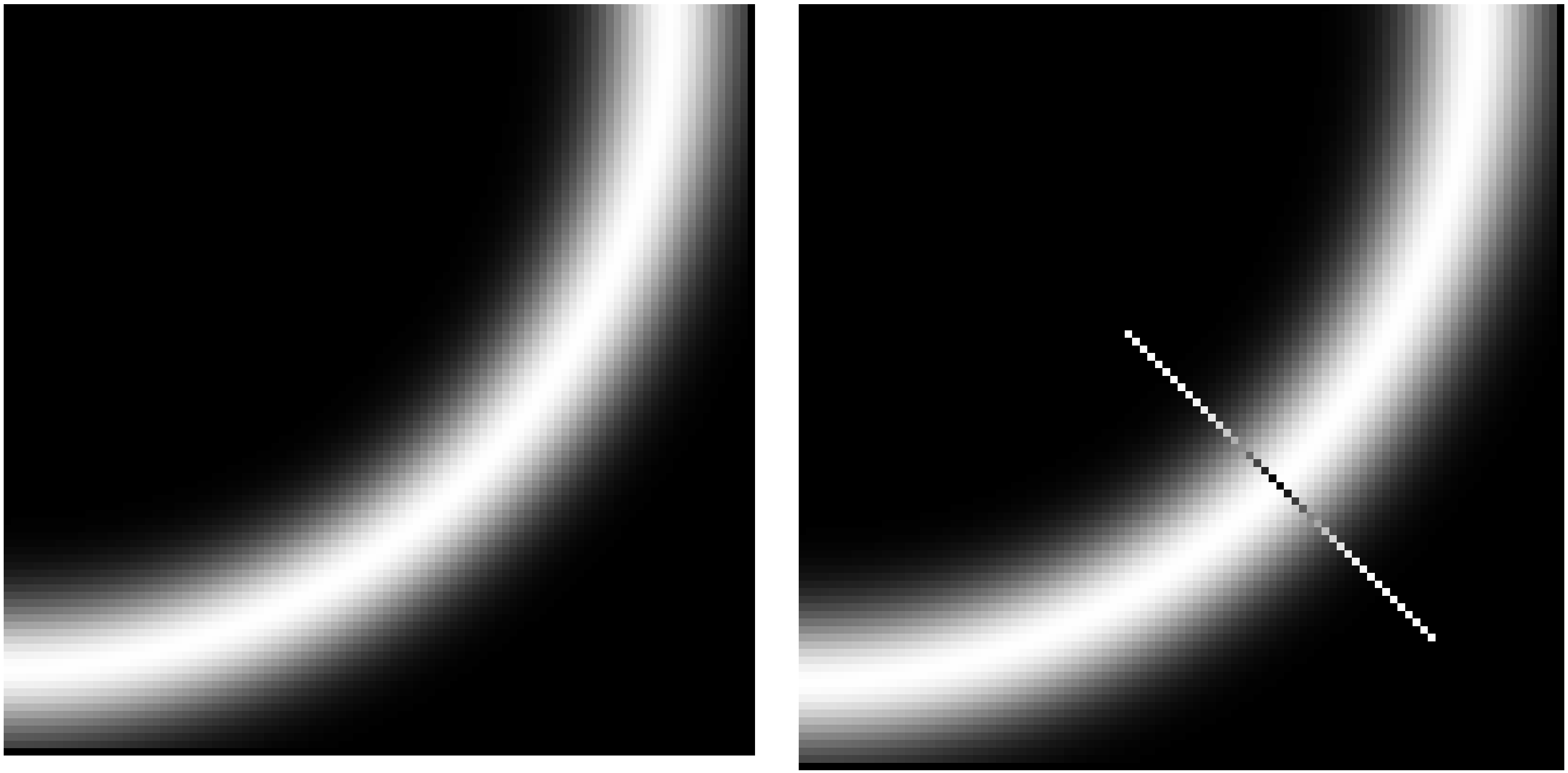
Canny Edge Detector
Fourth Step

- Non maximum suppression



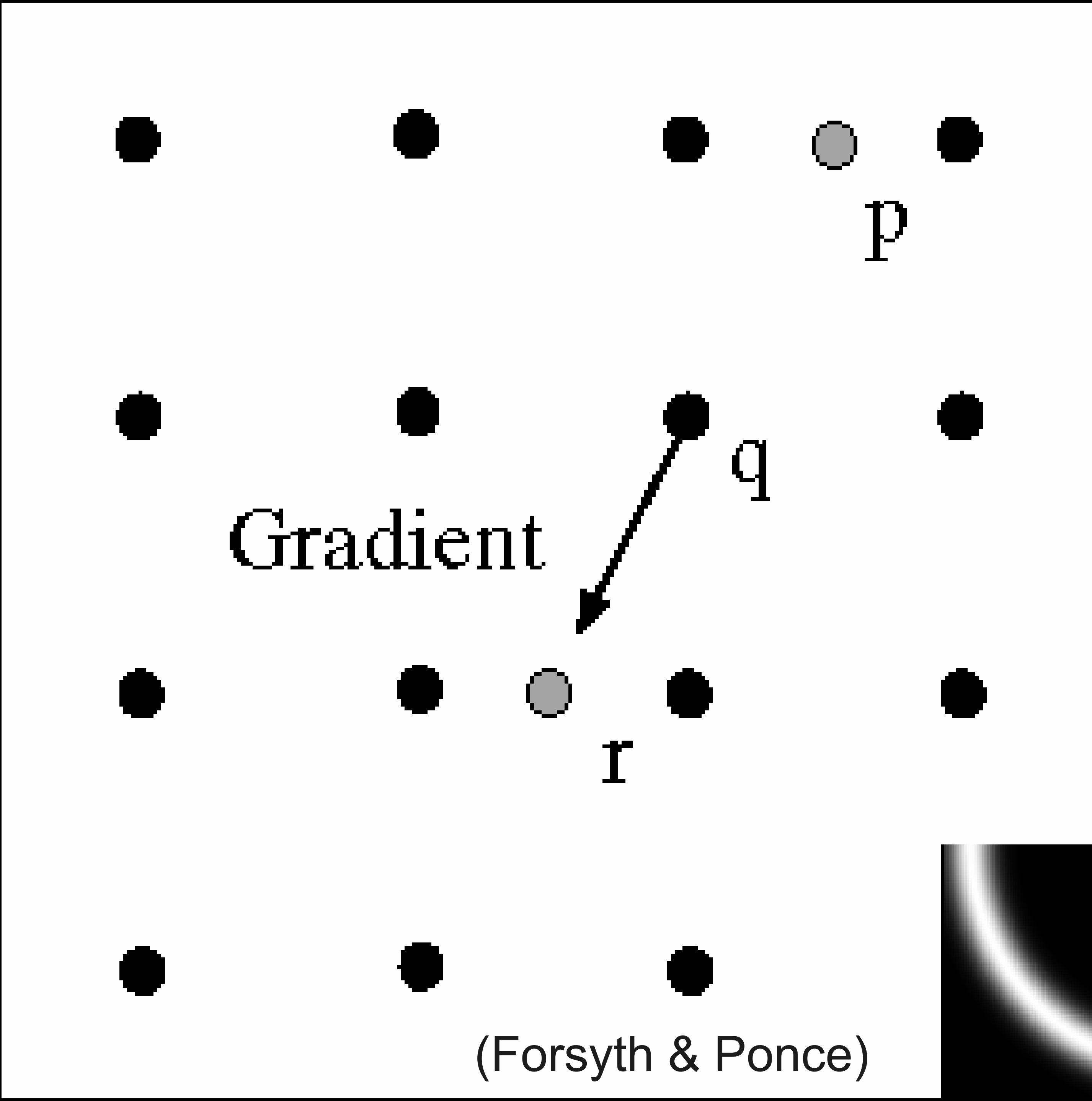
We wish to mark points along the curve where the **magnitude is biggest**. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?

Non-Maximum Supression



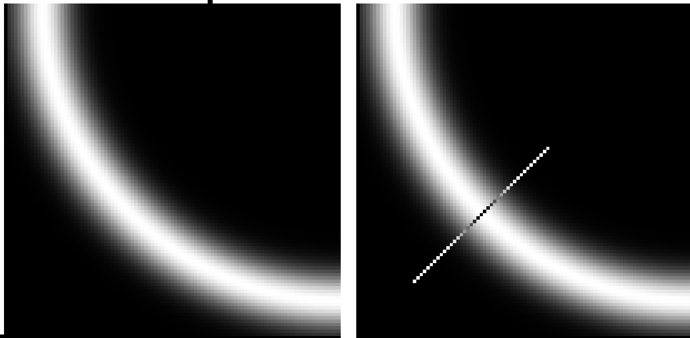
Non-maximum suppression:

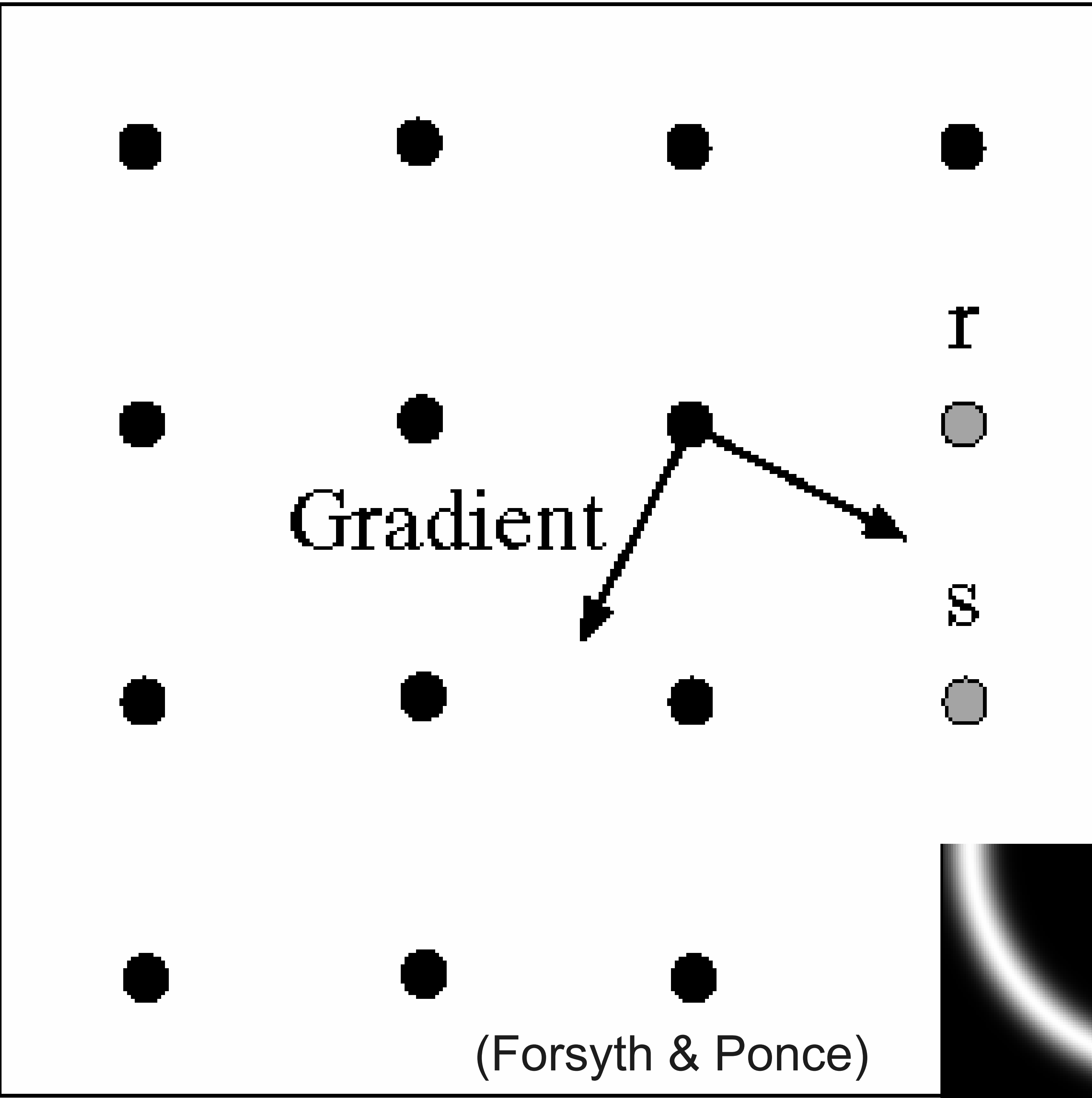
Select the single maximum point across the width of an edge.



Non-maximum suppression

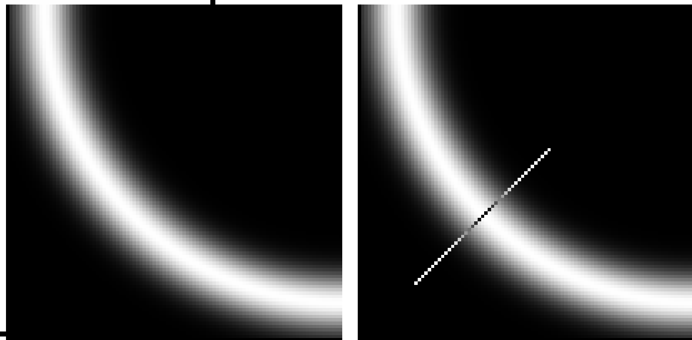
At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.





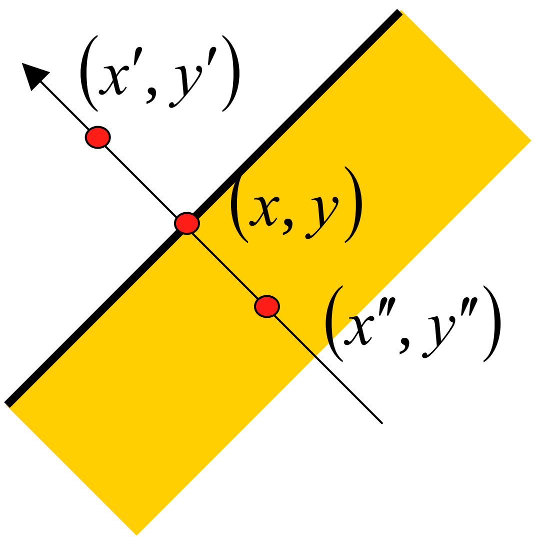
Predicting the next edge point

Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either r or s).



Canny Edge Detector
Non-Maximum Suppression

- Suppress the pixels in $|\nabla S|$ which are not local maximum



$$M(x,y)=\begin{cases} |\nabla S|(x,y) & \text{if } |\Delta S|(x,y) > |\Delta S|(x',y') \\ & \;\&\; |\Delta S|(x,y) > |\Delta S|(x'',y'') \\ 0 & \text{otherwise} \end{cases}$$

x' and **x**'' are the neighbors of **x** along normal direction to an edge

Canny Edge Detector
Quantization of Normal Directions

$$\tan\theta = \frac{S_y}{S_x}$$

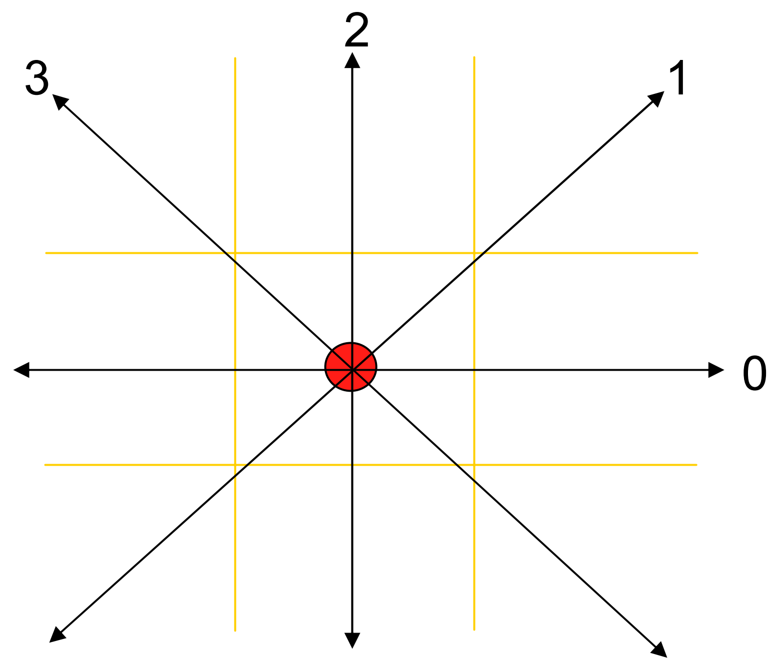
Quantizations:

0: $-0.4142 < \tan\theta \le 0.4142$

1: $0.4142 < \tan\theta < 2.4142$

2: $|\tan\theta| \ge 2.4142$

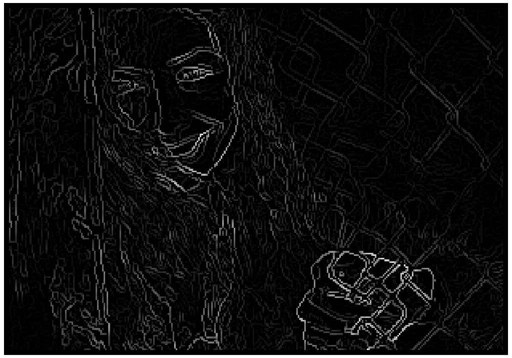
3: $-2.4142 < \tan\theta \le -0.4142$



Canny Edge Detector
Non-Maximum Suppression

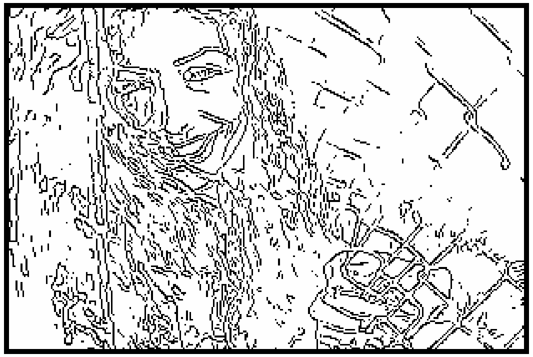
$$|\Delta S| = \sqrt{S_x^2 + S_y^2}$$

M



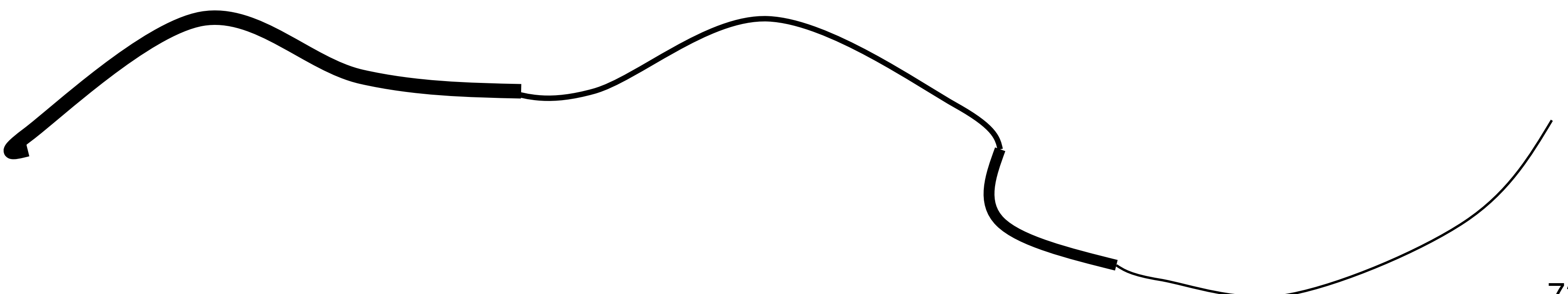
For visualization

$M \geq Threshold = 25$



Hysteresis(滞变)

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.



Edge Hysteresis

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

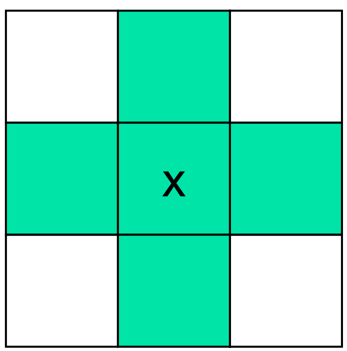
$$k_{high} / k_{low} = 2$$

Canny Edge Detector Hysteresis Thresholding

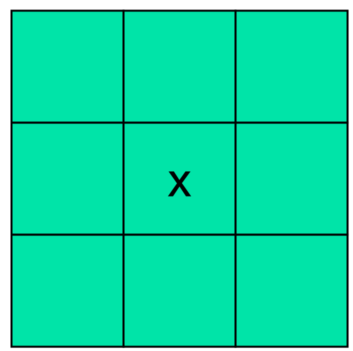
- If the gradient at a pixel is
  - above "**High**", declare it an '**edge pixel**'
  - below "**Low**", declare it a "**non-edge-pixel**"
  - **between** "low" and "high"
    - Consider its neighbors iteratively then declare it an "edge pixel" if it is **connected** to an 'edge pixel' **directly** or via pixels **between** "low" and "high".

Canny Edge Detector
Hysteresis Thresholding

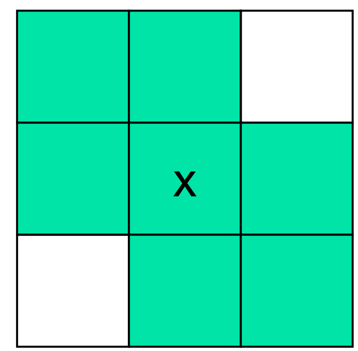
- Connectedness



4 connected

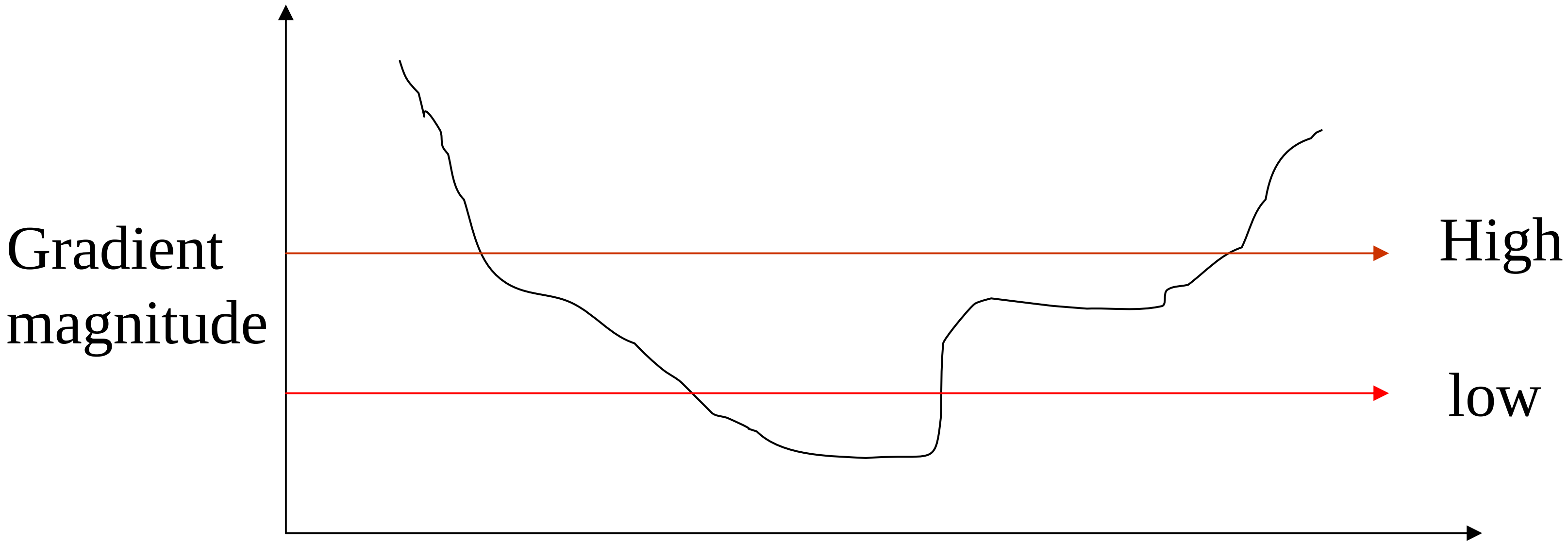


8 connected



6 connected

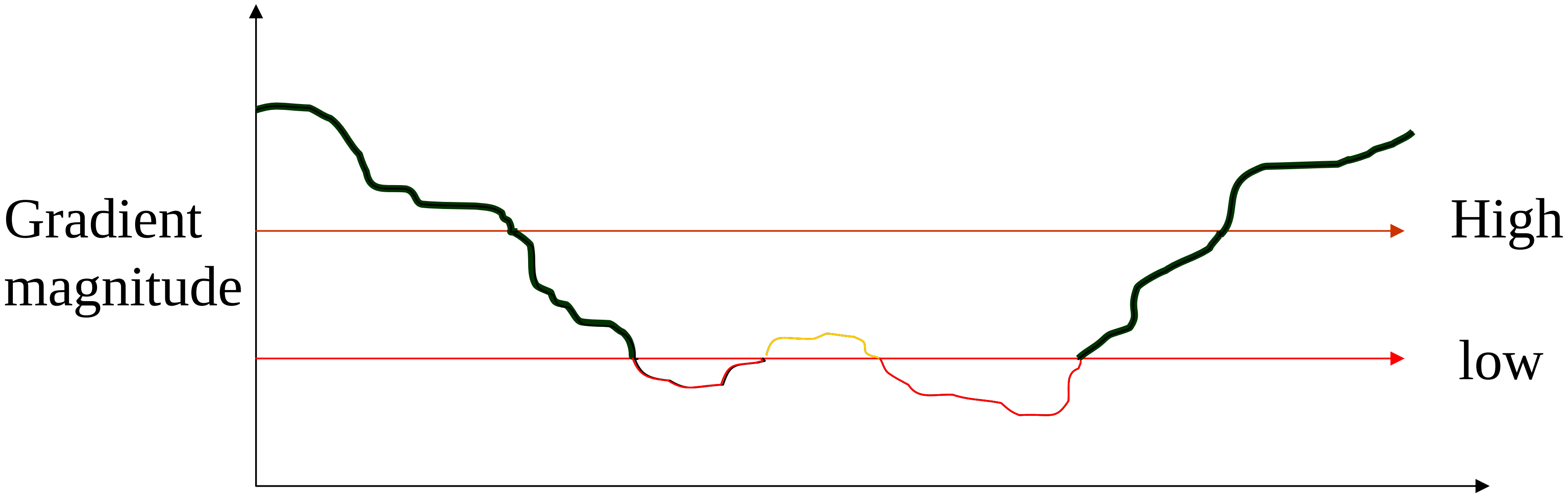
Canny Edge Detector
Hysteresis Thresholding



Canny Edge Detector
Hysteresis Thresholding

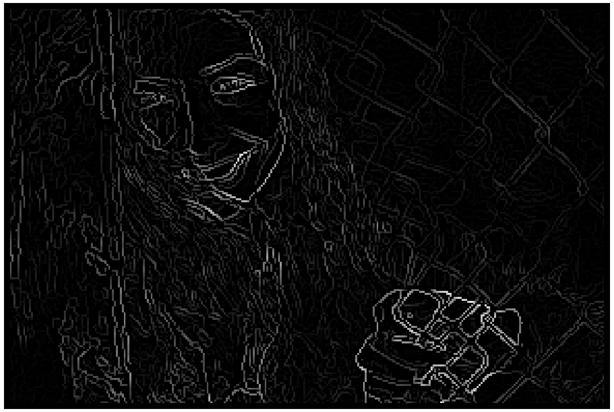
- Scan the image from left to right, top-bottom.
  - The gradient magnitude at a pixel is above a high threshold declare that as an edge point
  - Then recursively consider the *neighbors* of this pixel.
    - If the gradient magnitude is above the low threshold declare that as an edge pixel.

Canny Edge Detector
Hysteresis Thresholding



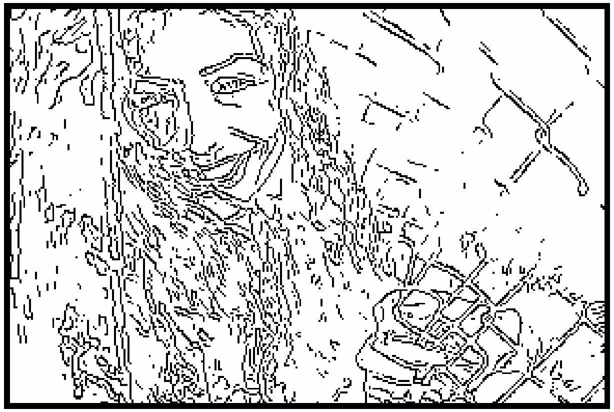
Canny Edge Detector
Hysteresis Thresholding

M



regular

$M \geq 25$



Hysteresis

$High = 35$

$Low = 15$

Summary: Canny Edge Detector

Steps:

1. Apply derivative of Gaussian
2. Non-maximum suppression
   - Thin multi-pixel wide “ridges” down to single pixel width
3. Linking and thresholding
   - Low, high edge-strength thresholds
   - Accept all edges over low threshold that are connected to edge over high threshold

Summary: Canny Edge Operator

- Smooth image *I* with 2D Gaussian: $G * I$
- Find local edge normal directions for each pixel $\overline{\mathbf{n}} = \dfrac{\nabla(G * I)}{\left|\nabla(G * I)\right|}$
- Compute edge magnitudes $\left|\nabla(G * I)\right|$
- Find the location of the edges by finding zero-crossings along the edge normal directions (**non-maximum suppression**) $\dfrac{\partial^2 (G * I)}{\partial \overline{\mathbf{n}}^2} = 0$
- Threshold edges in the image with **hysteresis** to eliminate spurious responses

Read Canny's original paper for further details

Why is Canny so Dominant

- Still widely used after 20 years.

1. Theory is nice (but end result same).
2. Details good (magnitude of gradient).
3. Hysteresis an important heuristic.
4. Code was distributed.
5. Perhaps this is about all you can do with linear filtering.

Demo of Edge Detection

Canny Edge Detection (Example)

Original image

gap is gone

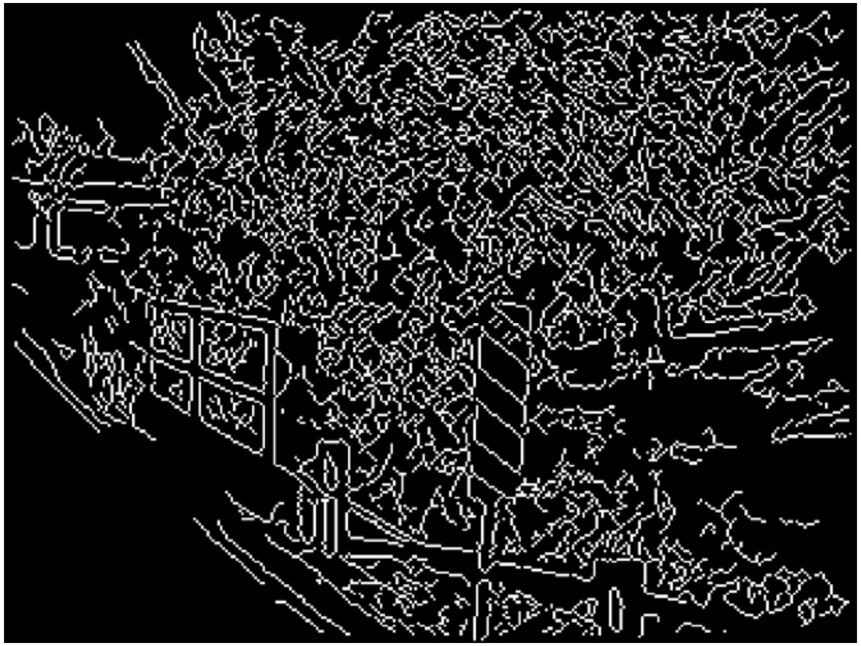
Strong + connected weak edges

Strong edges only

Weak edges

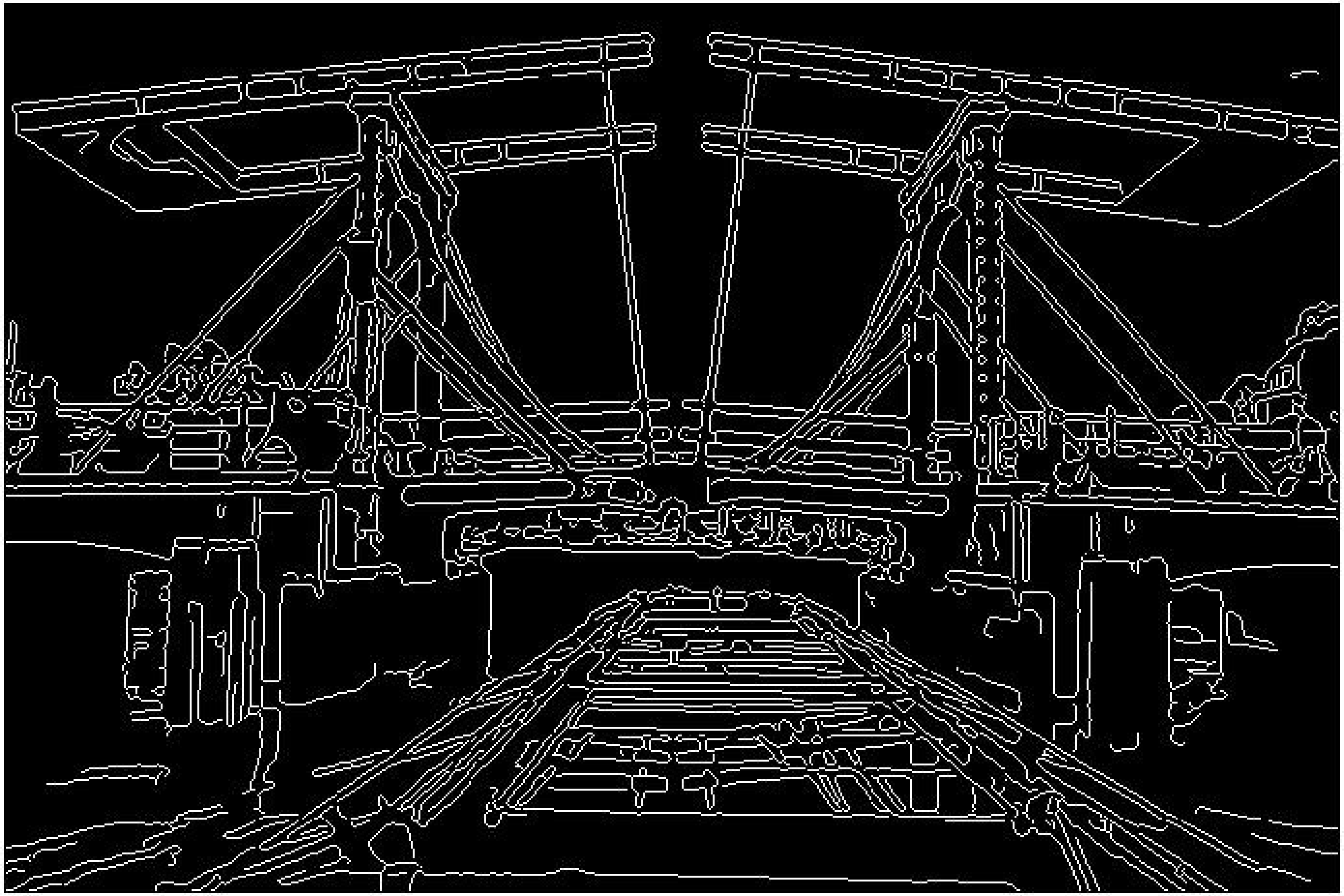
courtesy of G. Loy

Canny Edge Detection (Example)



Using Matlab with default thresholds

Bridge Example



edge(im,'canny')

The Canny Edge Detector

original image (Lena)

The Canny Edge Detector

magnitude of the gradient

The Canny Edge Detector

After non-maximum suppression and thresholding with hysterisis

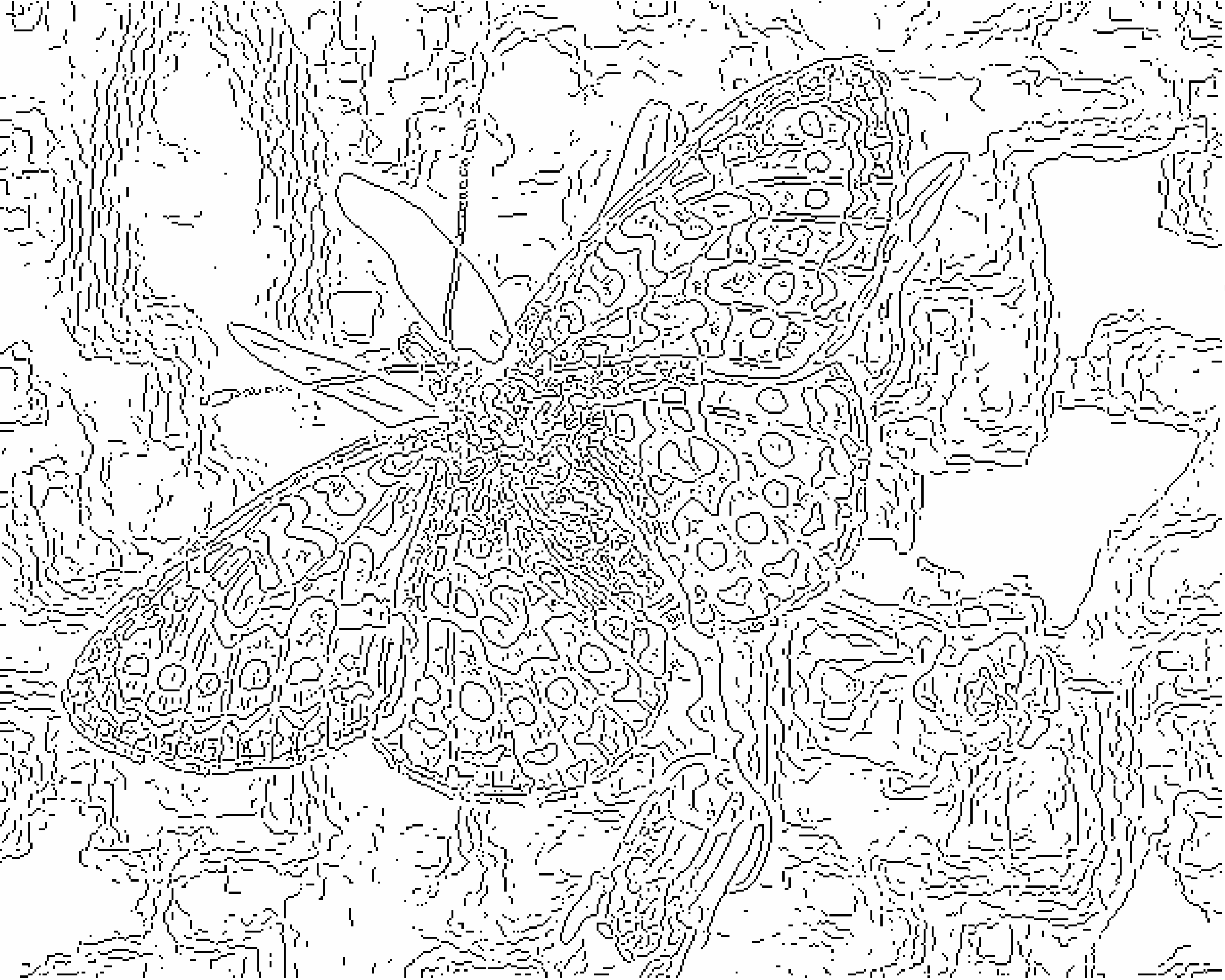
Canny Edge Operator

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

- The choice of σ depends on desired behavior
  - large σ detects large scale edges
  - small σ detects fine features

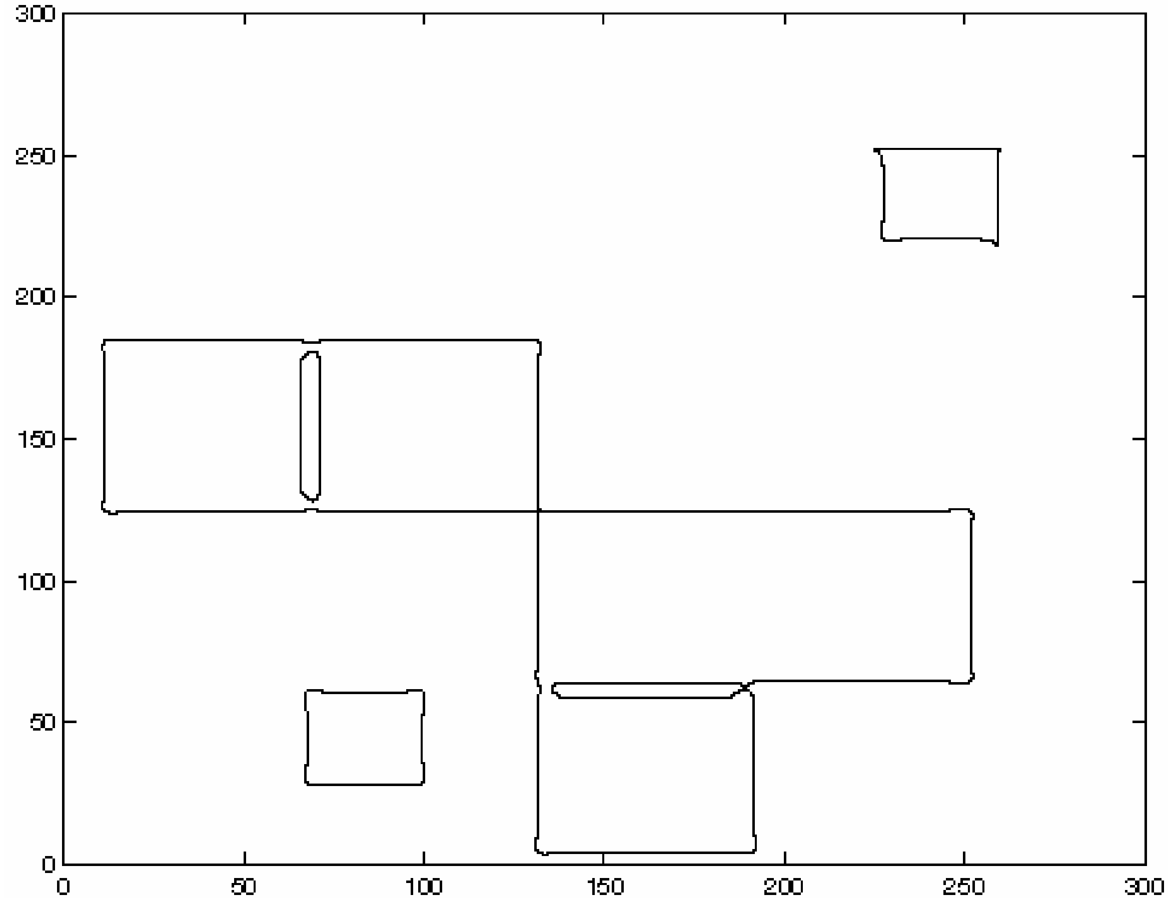


fine scale
high
threshold

coarse
scale,
high
threshold

coarse
scale
low
threshold

Corner Effects

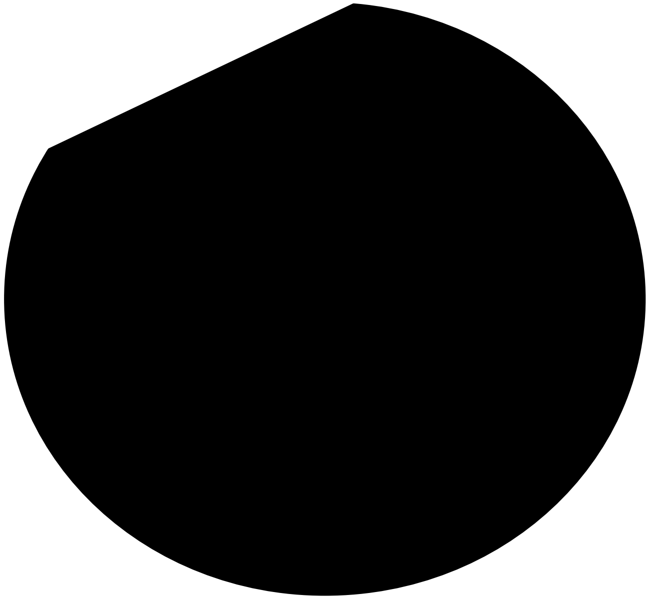


Today's Goals (Break)

- Features Overview
- Canny Edge Detector
- Harris Corner Detector
- Templates and Image Pyramid
- SIFT Features

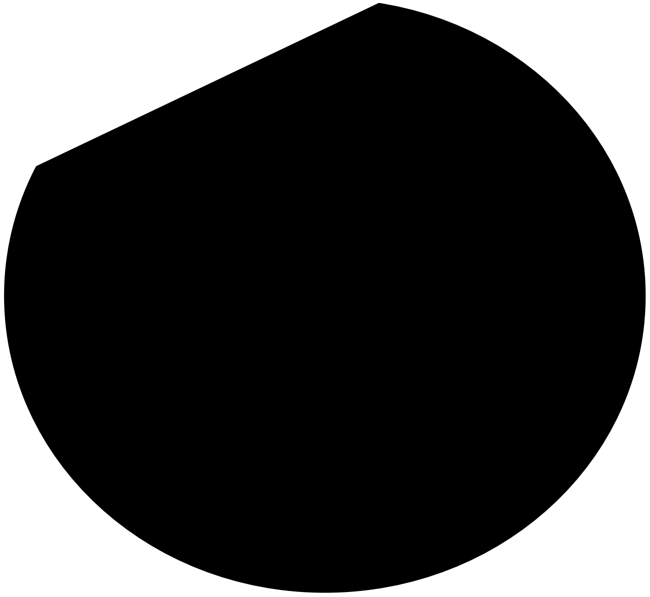
Corners(Start)

- Why are they important?



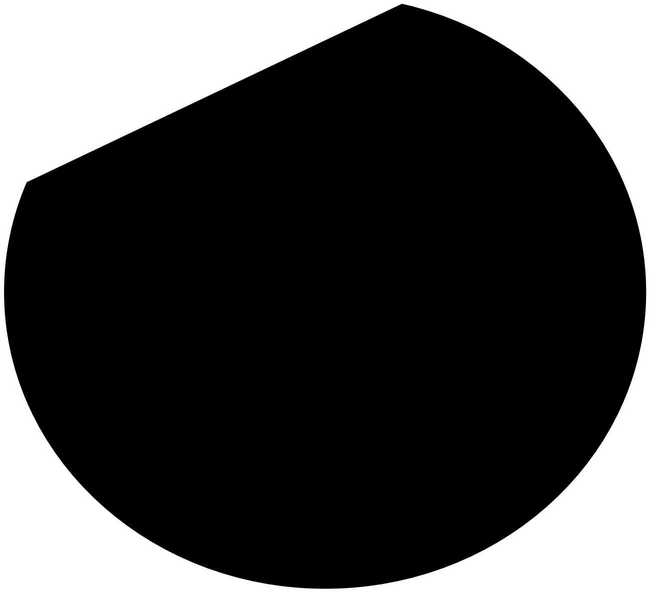
Corners

- Why are they important?



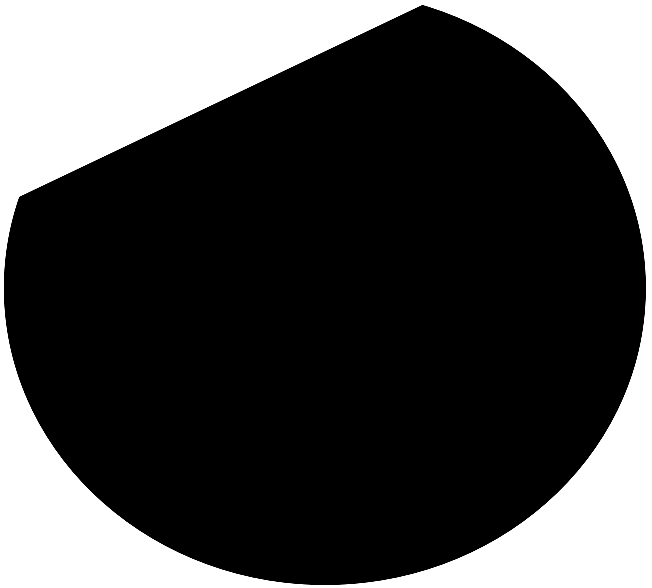
Corners

- Why are they important?



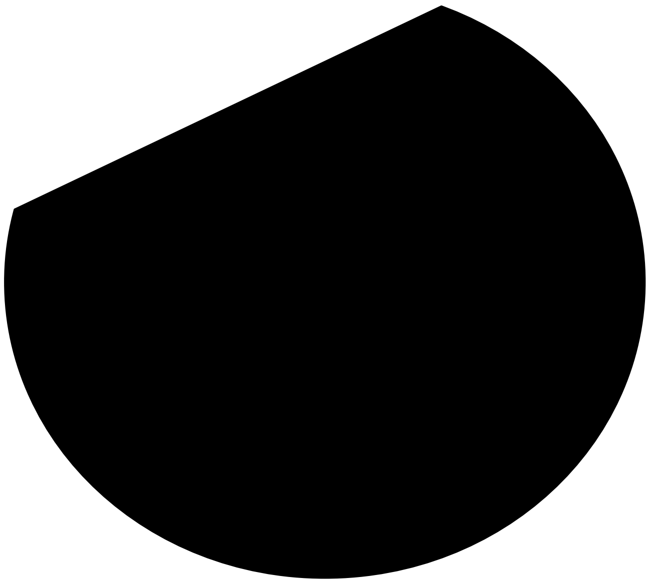
Corners

- Why are they important?



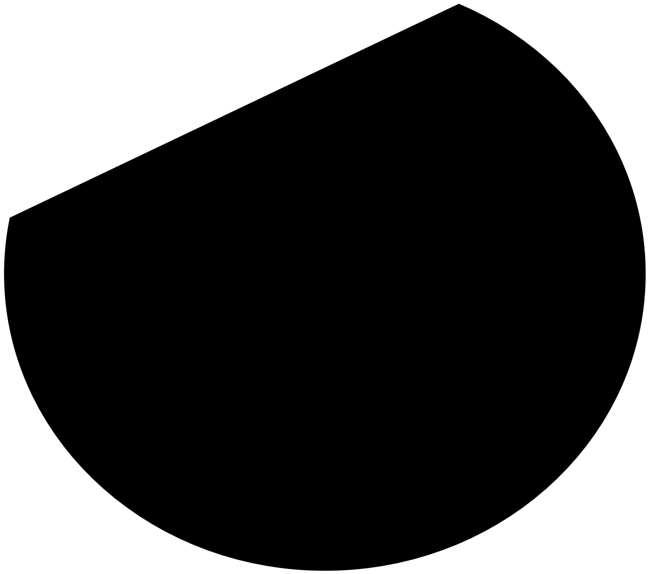
Corners

- Why are they important?



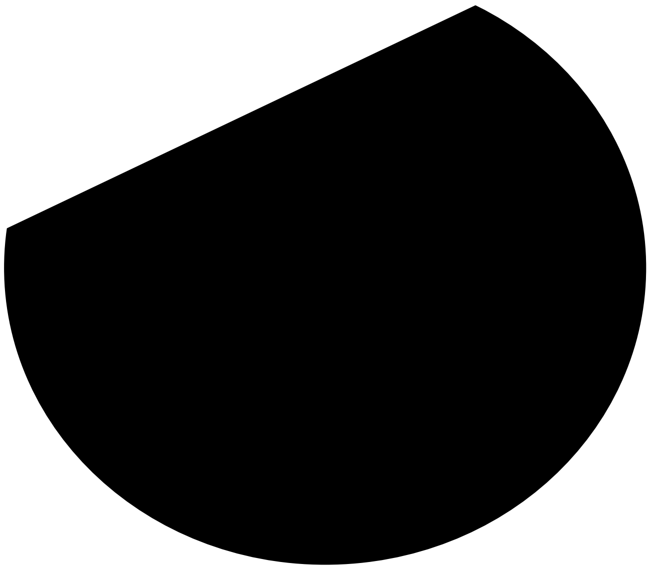
Corners

- Why are they important?



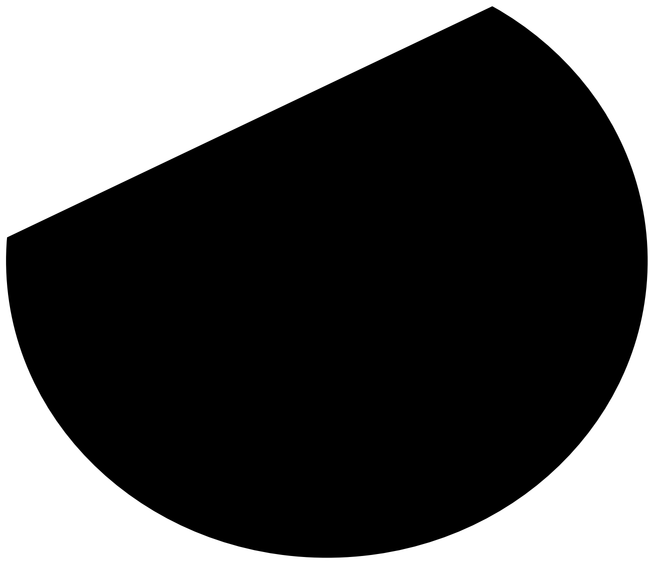
Corners

- Why are they important?



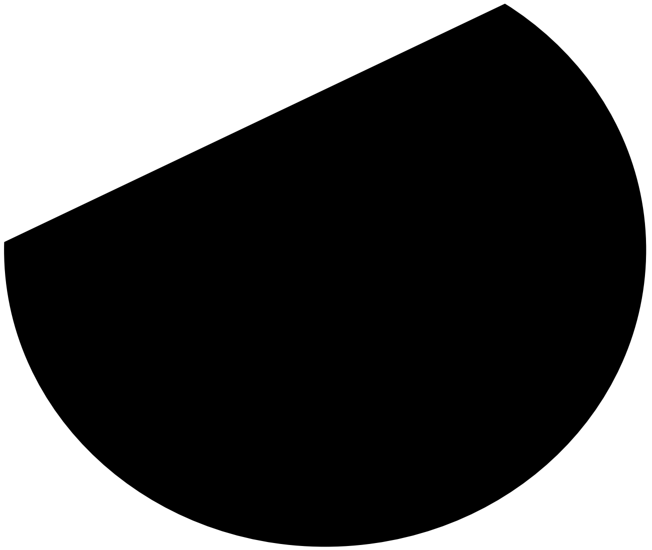
Corners

- Why are they important?



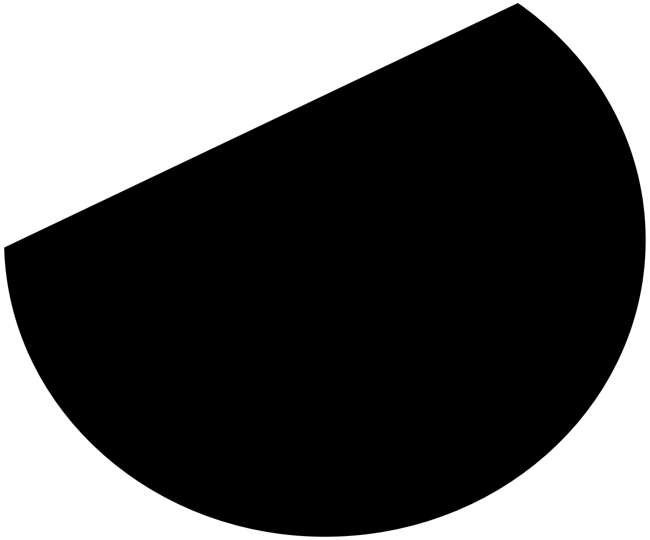
Corners

- Why are they important?



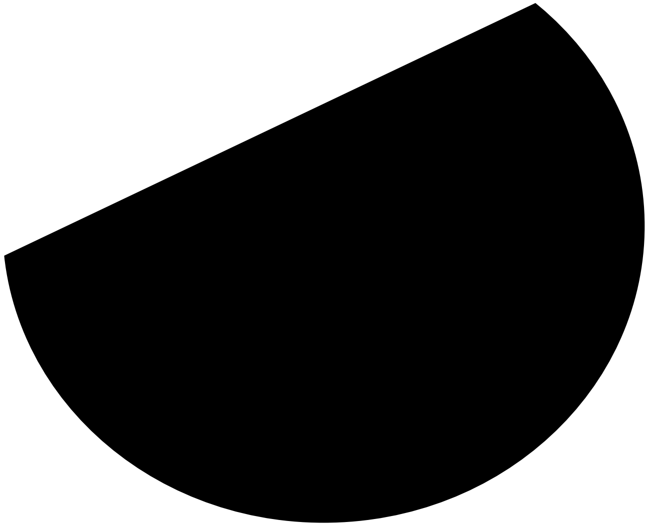
Corners

- Why are they important?



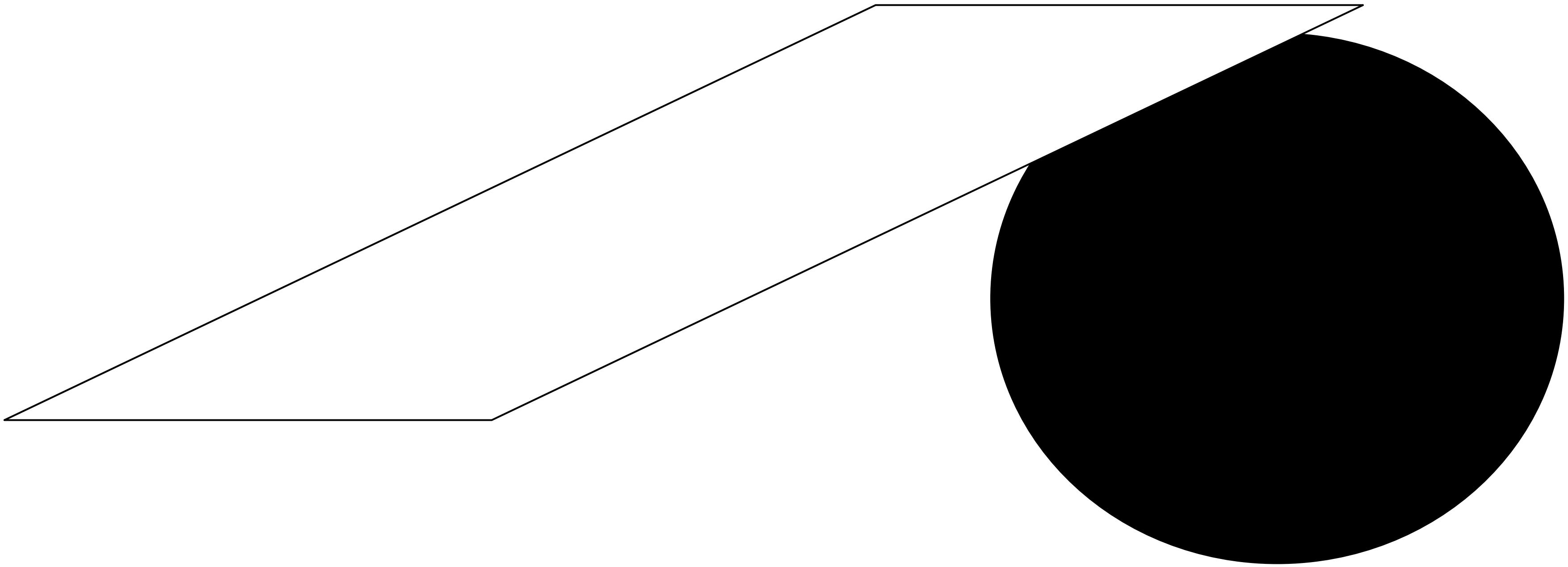
Corners(End)

- Why are they important?



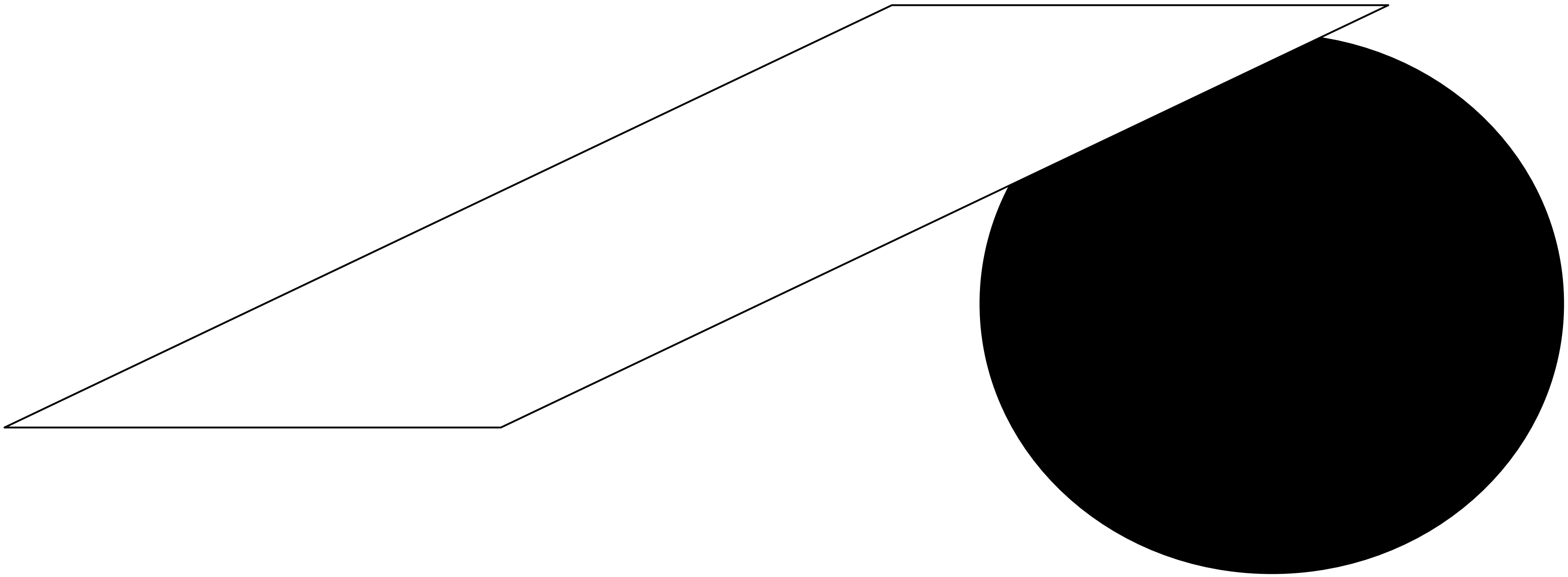
Corners

- Why are they important?



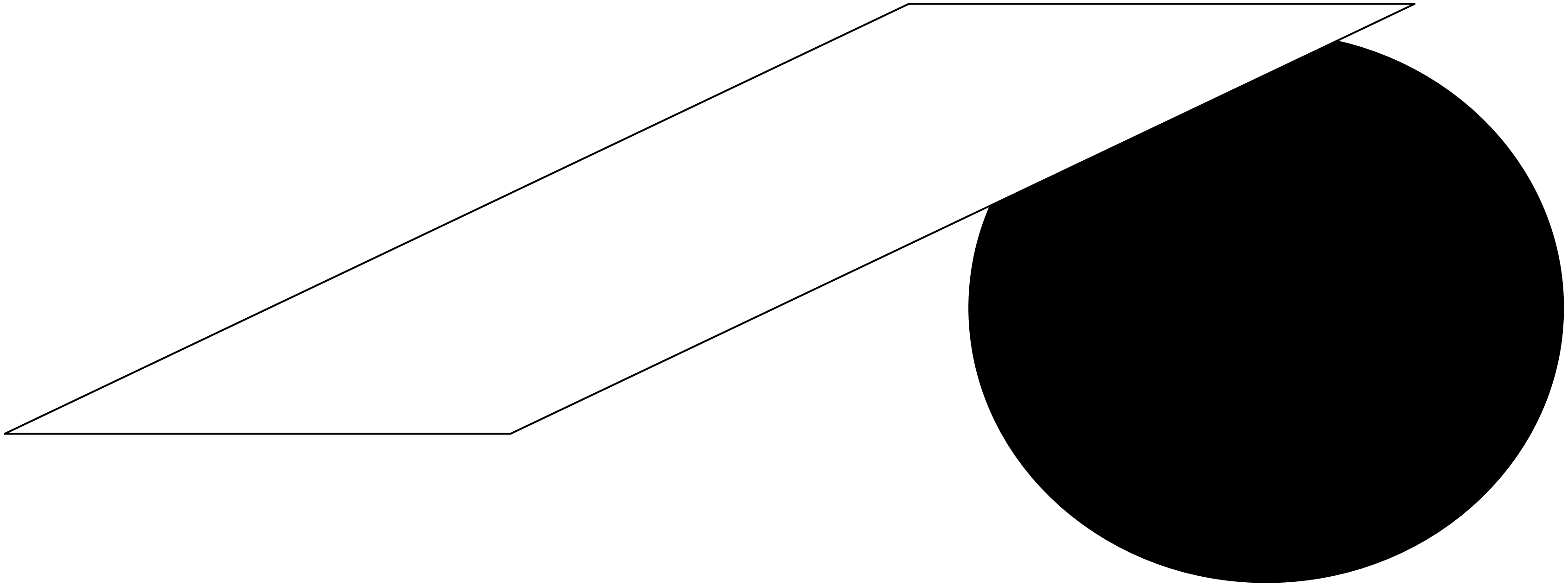
Corners

- Why are they important?



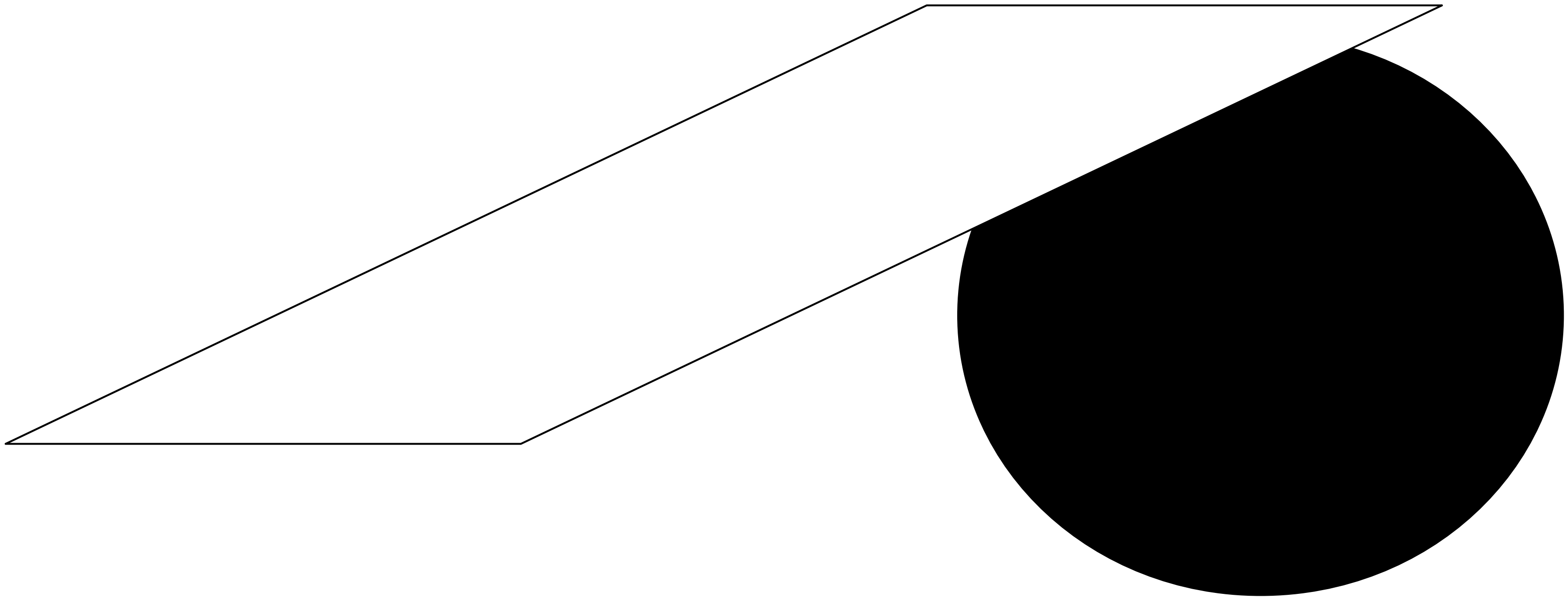
Corners

- Why are they important?



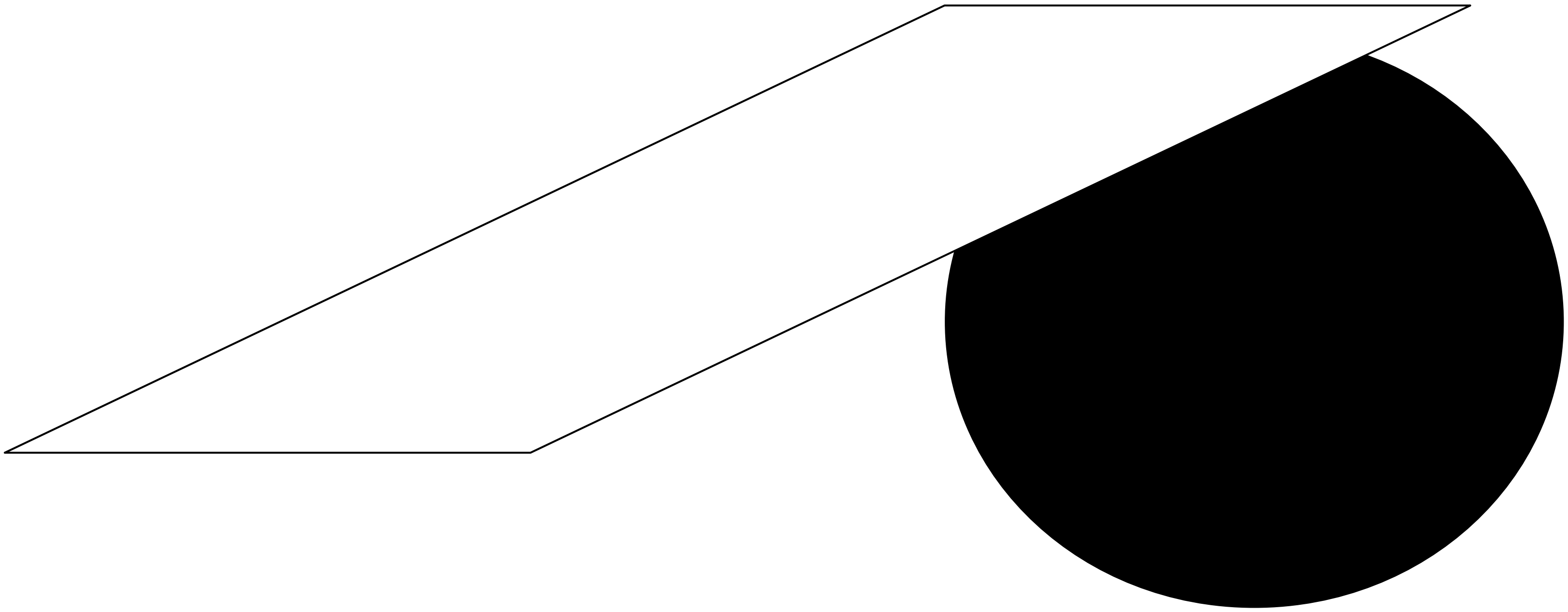
Corners

- Why are they important?



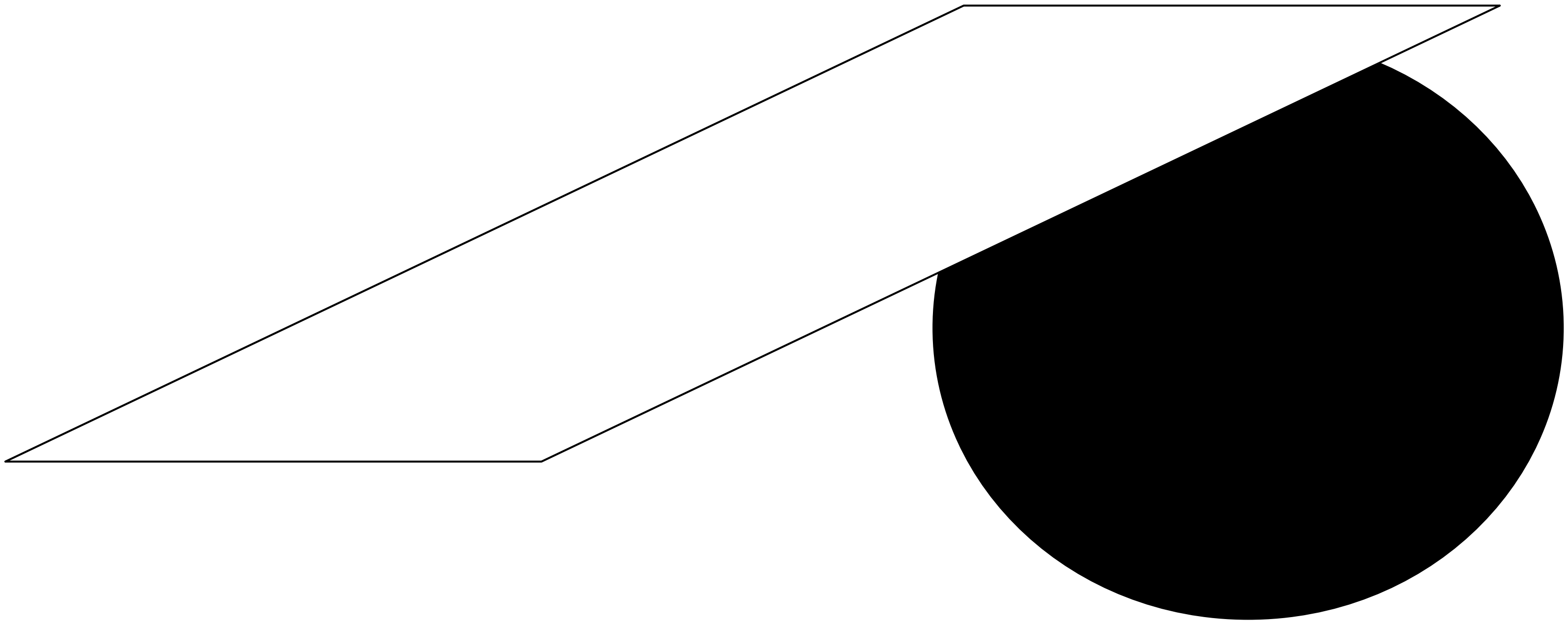
Corners

- Why are they important?



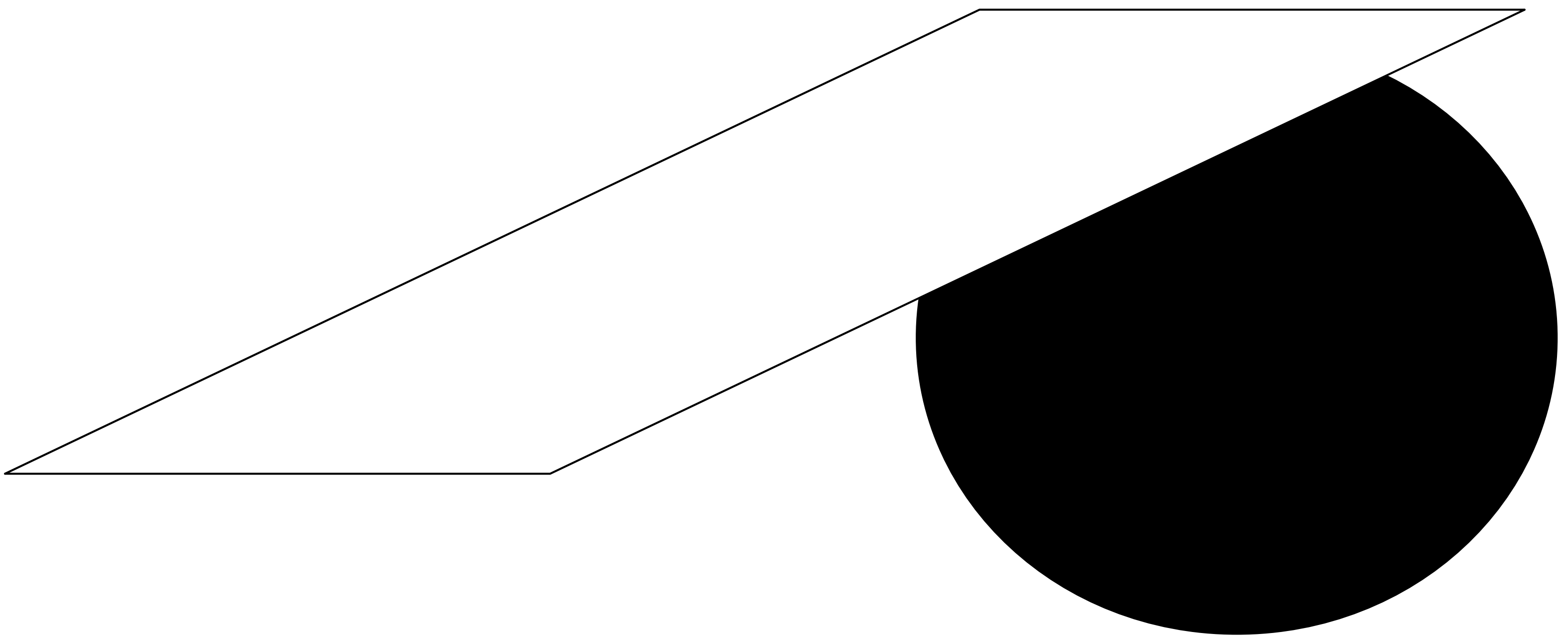
Corners

- Why are they important?



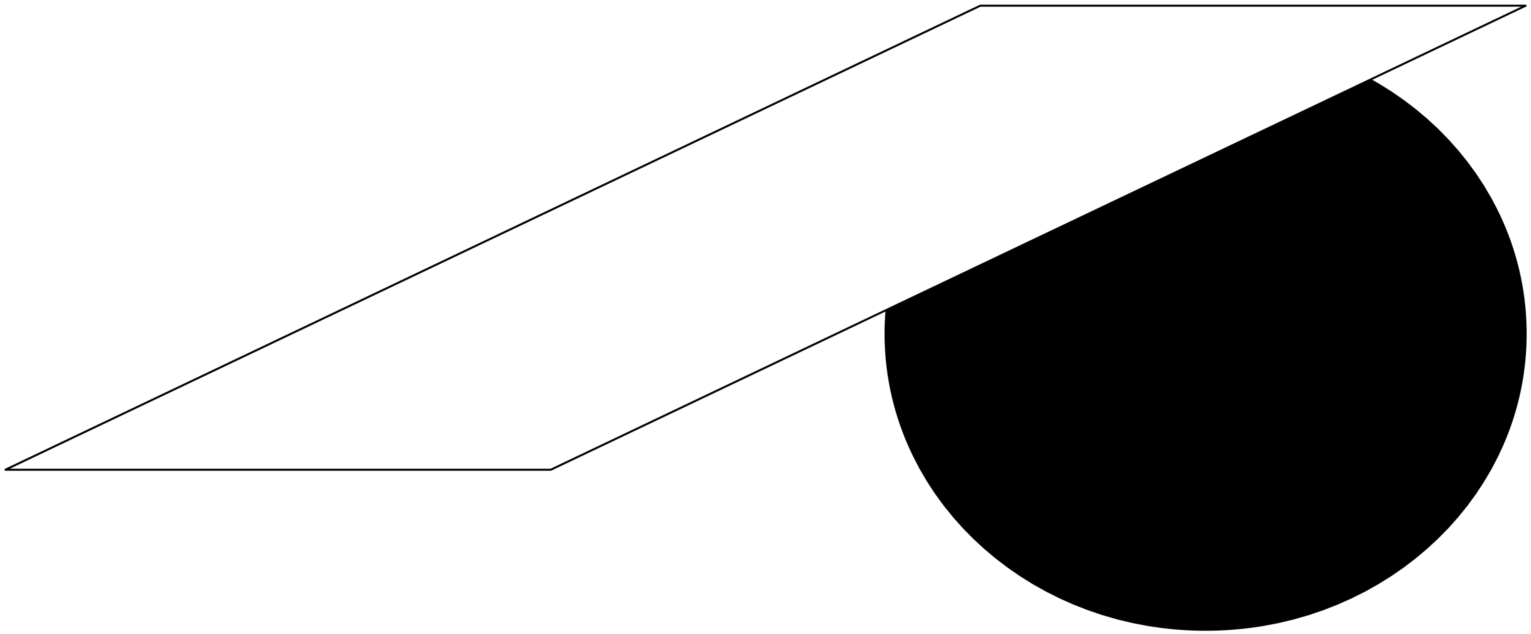
Corners

- Why are they important?



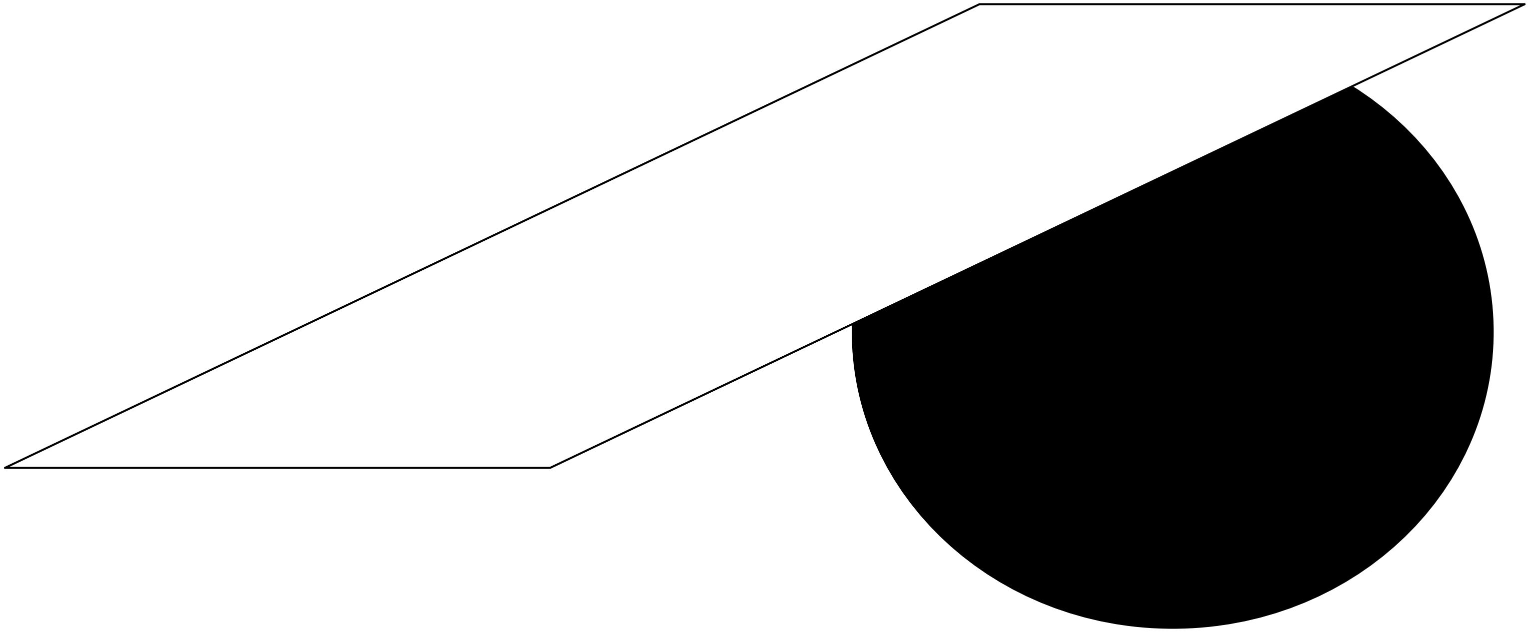
Corners

- Why are they important?



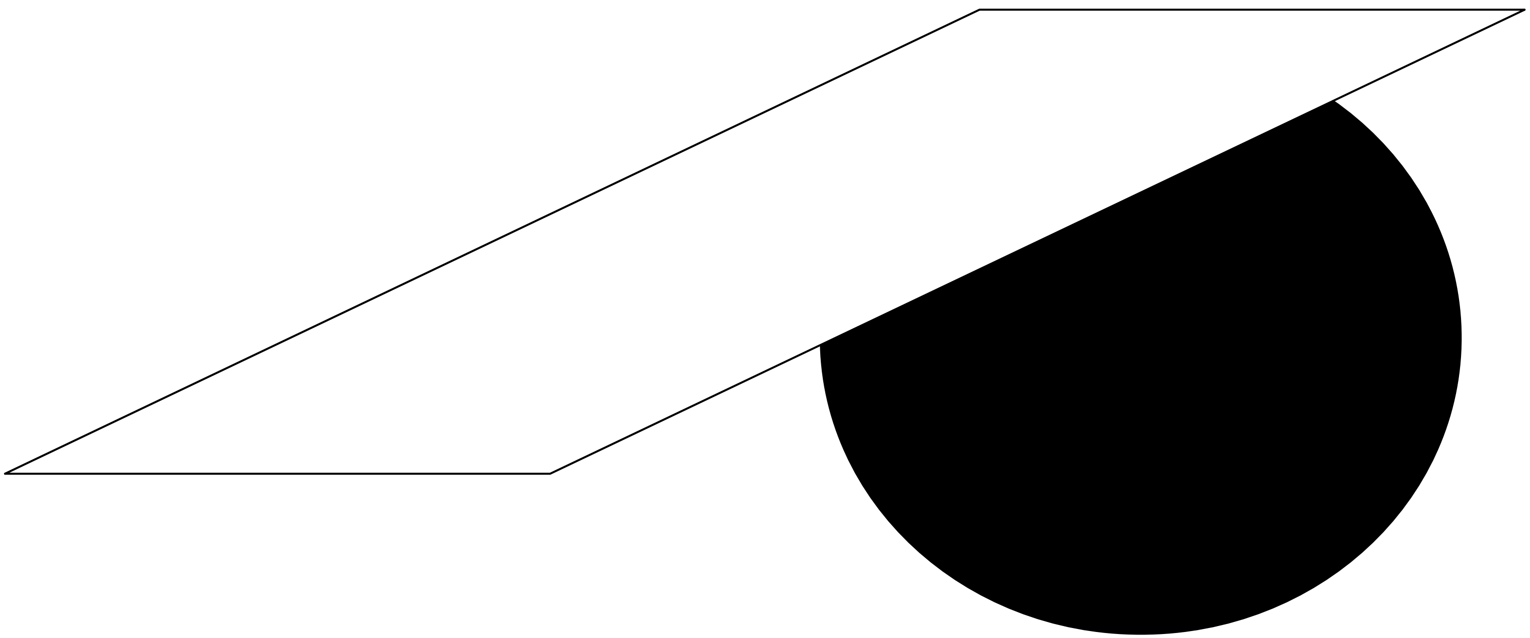
Corners

- Why are they important?



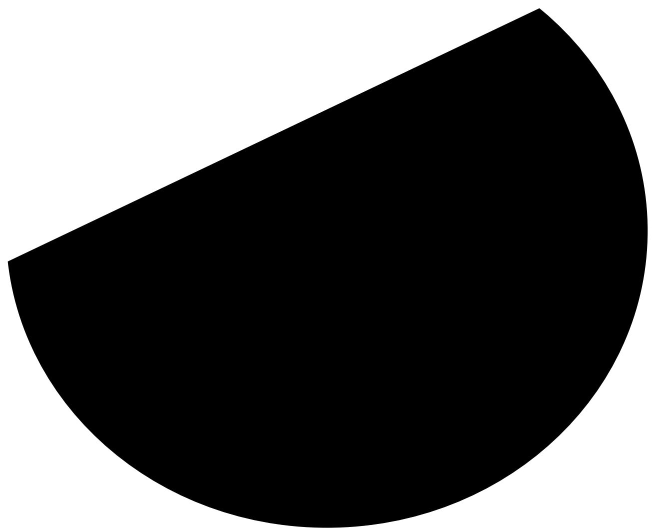
Corners(End)

- Why are they important?



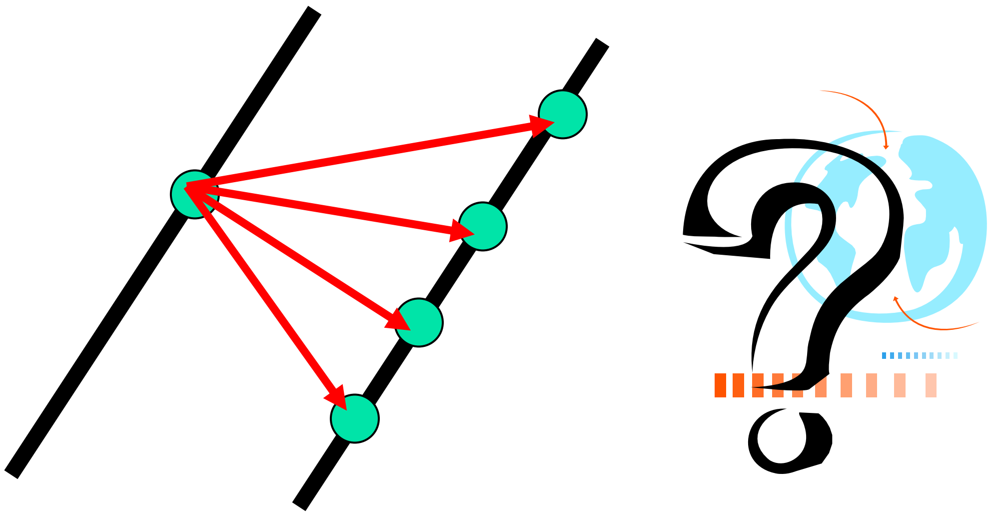
Corners

- Why are they important?



Corners contain more edges than lines.

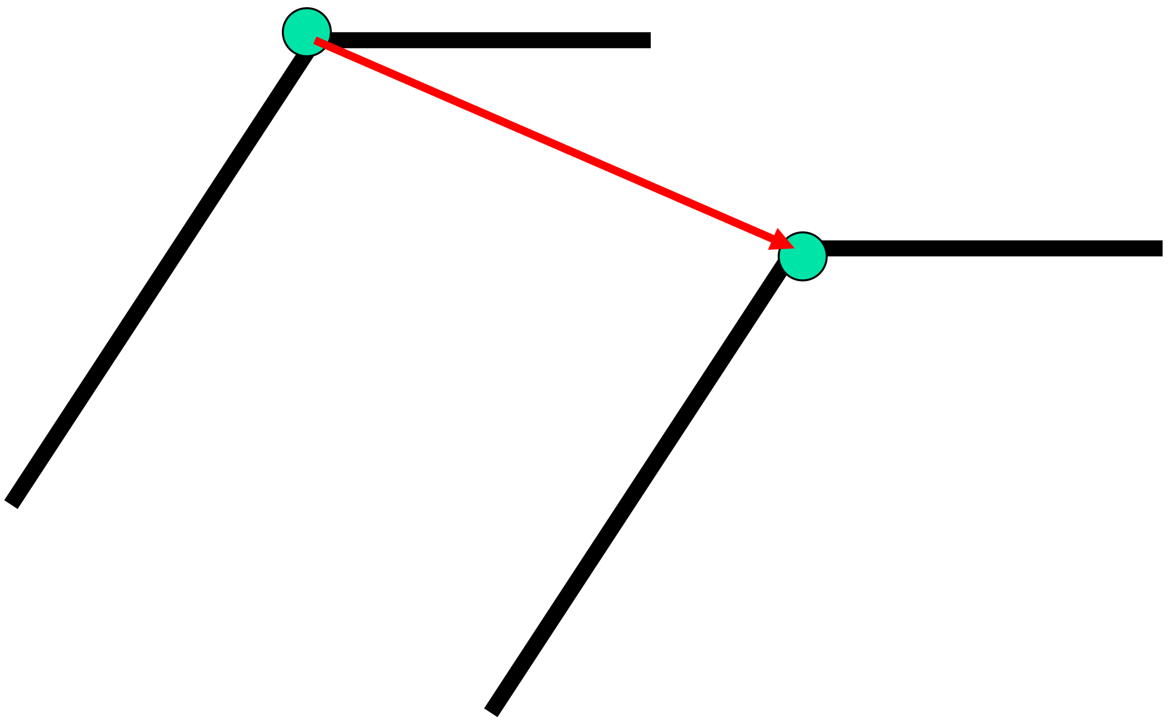
- A point on a line is hard to match.



Which one is the correct correspondence?

Corners contain more edges than lines.

- A corner is easier



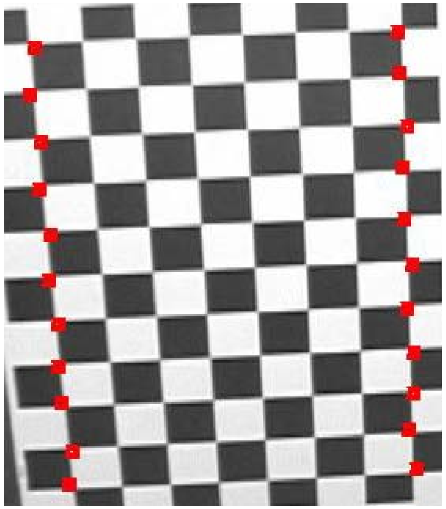
Finding Corners

Edge detectors perform poorly at corners.

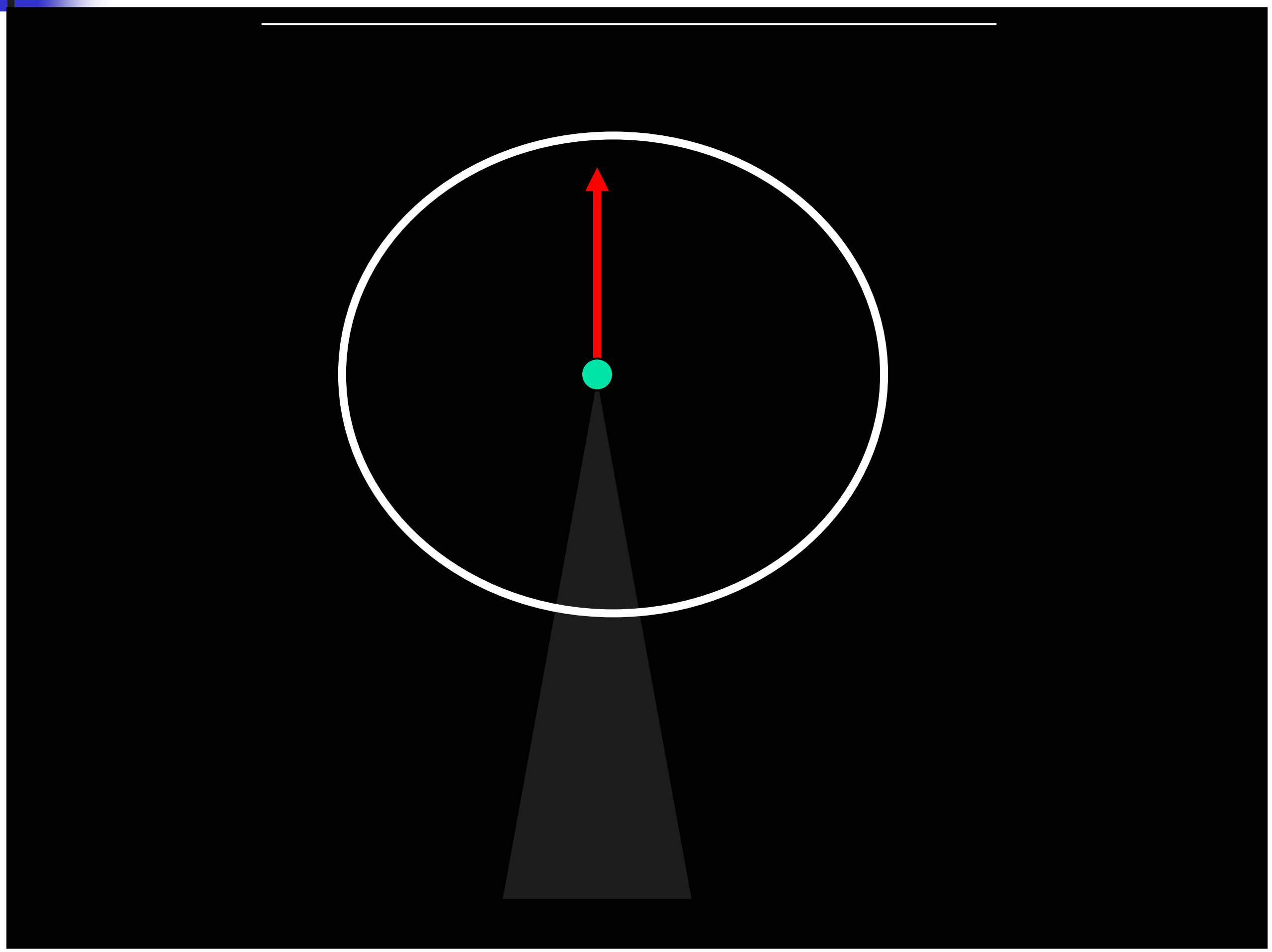
Corners provide repeatable points for matching, so are worth detecting.

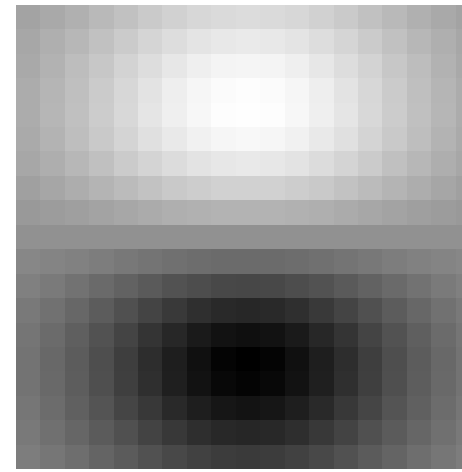
Idea:

- Right at a corner, gradient is ill defined.
- Near a corner, gradient has two or more different values.



Edge Detectors Tend to Fail at Corners





$$I_x = ?$$

$$I_y = ?$$

Formula for Finding Corners

Look at the second-moment matrix:

Sum over a small region, the hypothetical corner

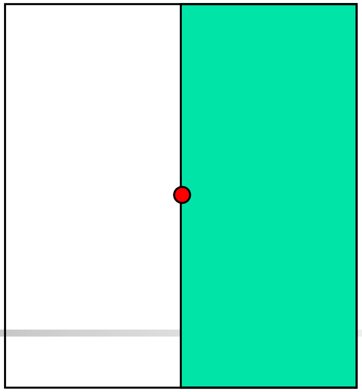
Gradient with respect to x, times gradient with respect to y

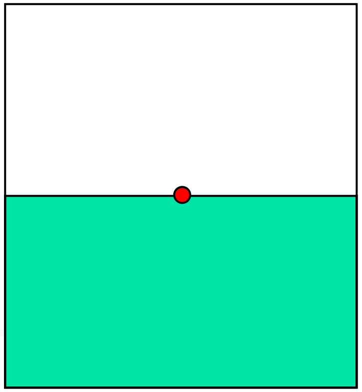
$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

Matrix is symmetric

WHY THIS?

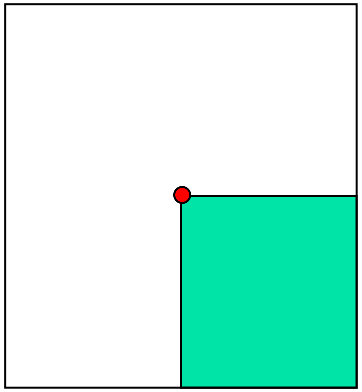
Simple Case





First, consider case where:

$$C = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} = \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}$$

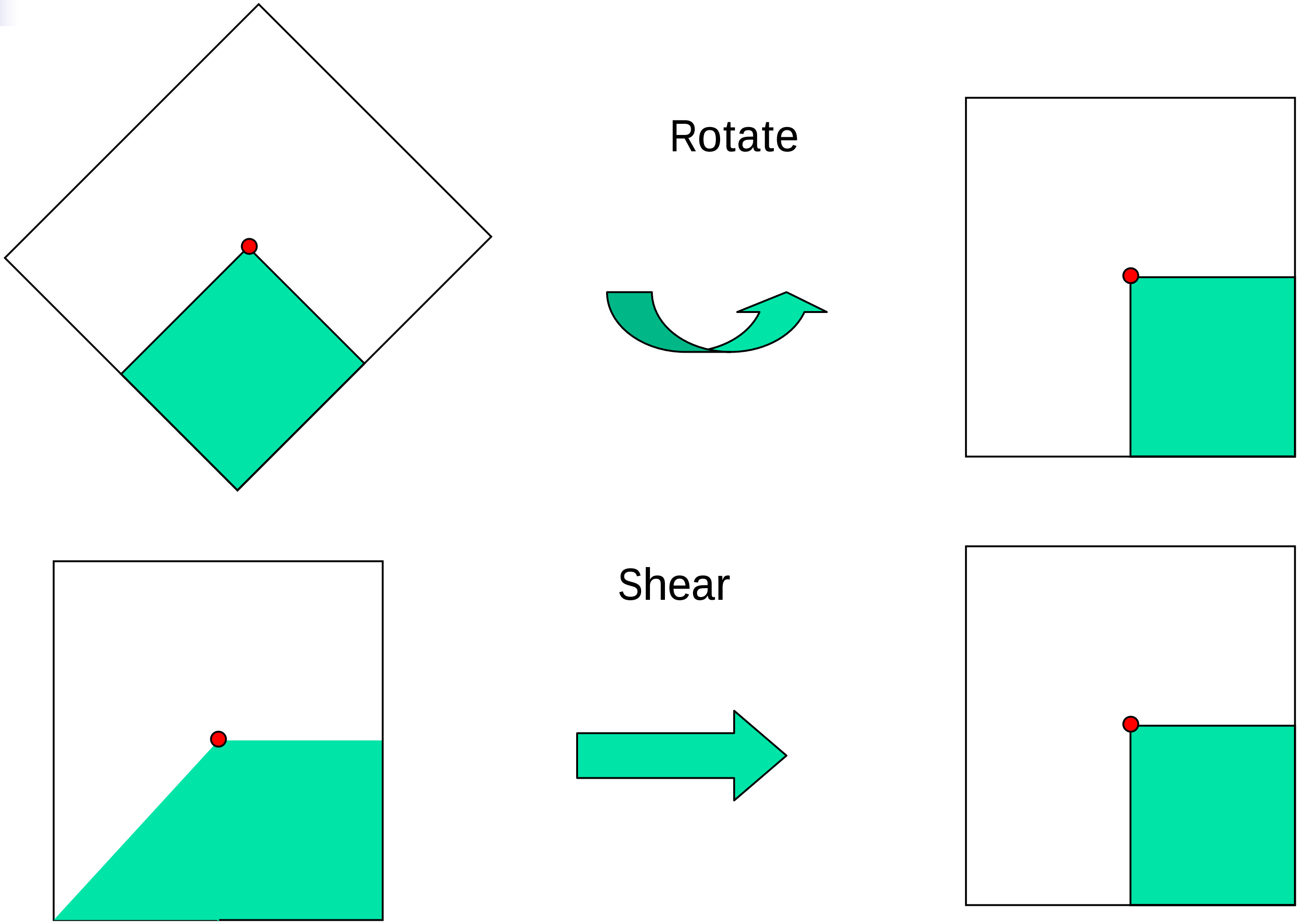


This means dominant gradient directions align with x or y axis

If either λ is close to 0, then this is **not** a corner, so look for locations where both are large.

Slide credit: David Jacobs

General Case



General Case

It can be shown that since C is rotationally symmetric:

$$C = R^{-1}\begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} R$$

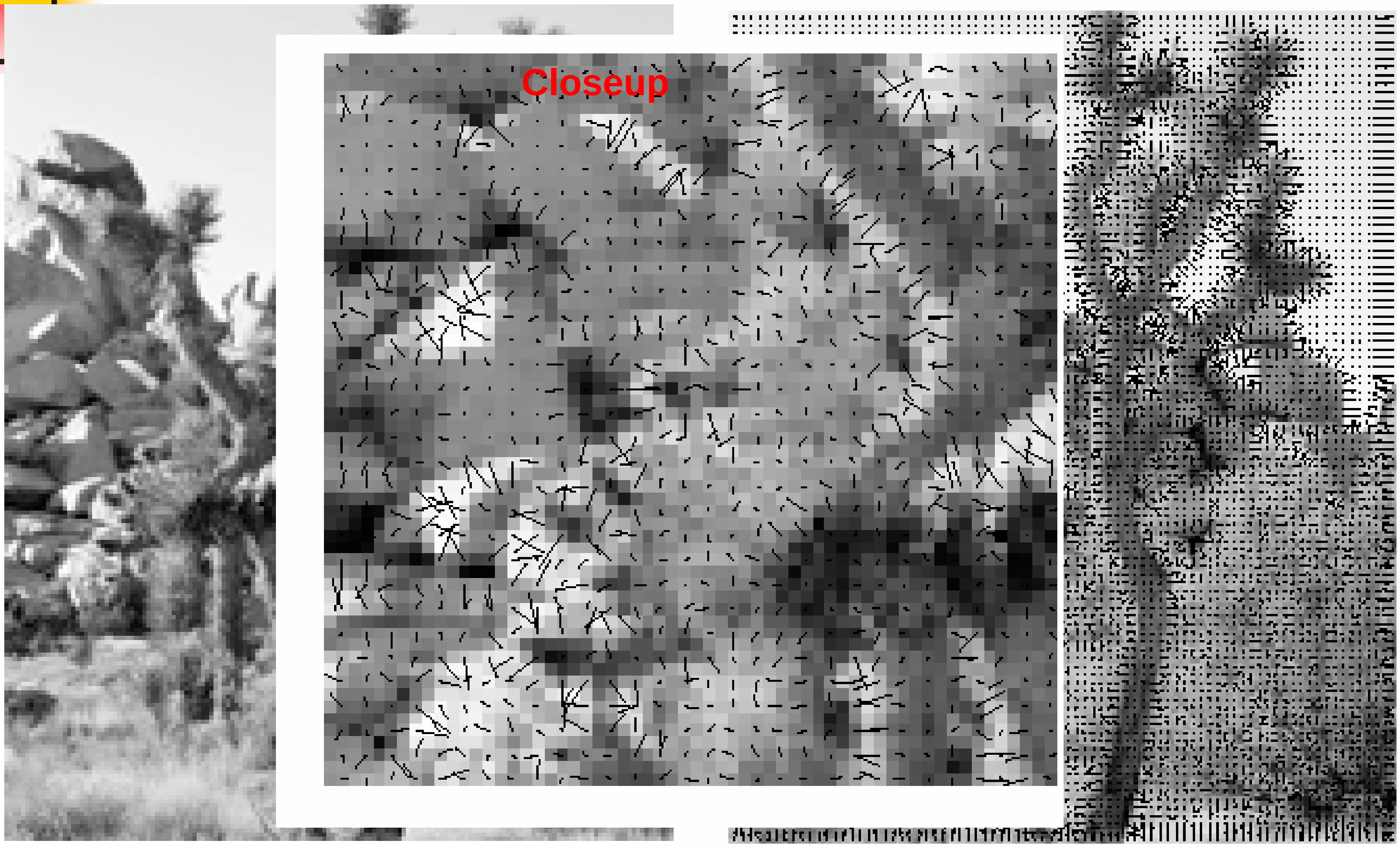
So every case is like a rotated version of the standard one on last slide.

Slide credit: David Jacobs

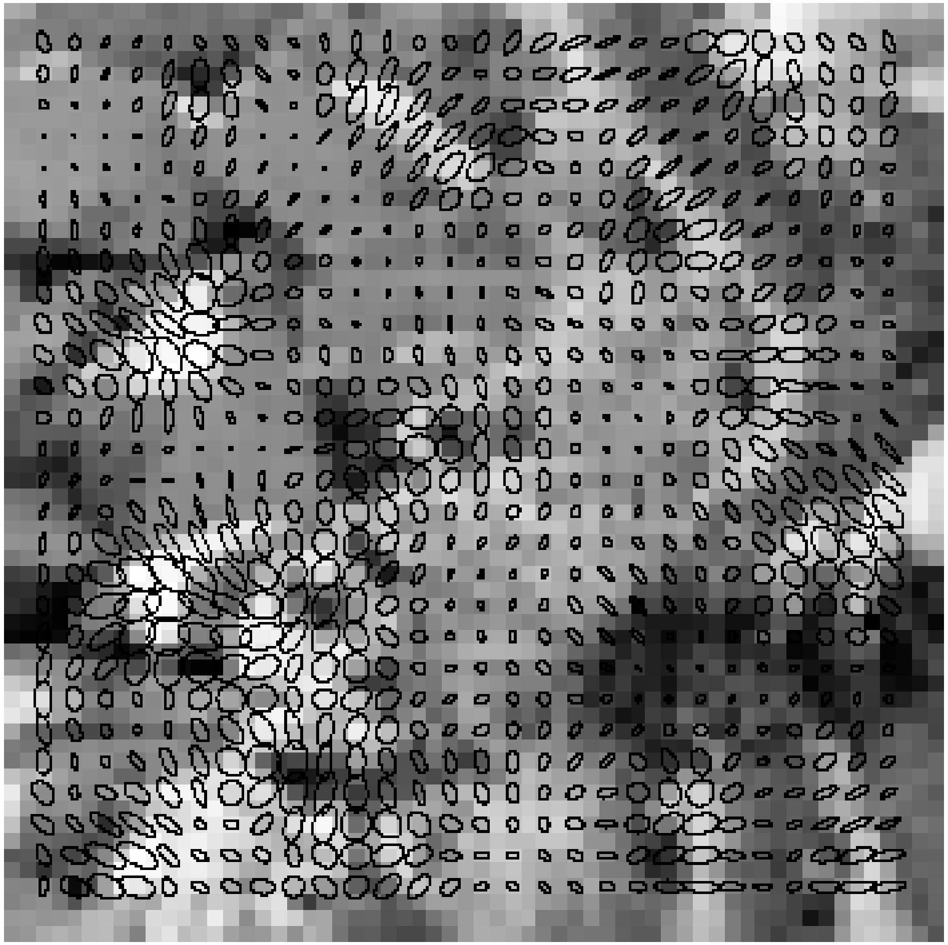
So, to detect corners

- Filter image.
- Compute magnitude of the gradient everywhere.
- Construct C in a window around the target pixel.
- Use Linear Algebra to find $\lambda 1$ and $\lambda 2$.
- If they are both big, we have a corner.

Gradient Orientation

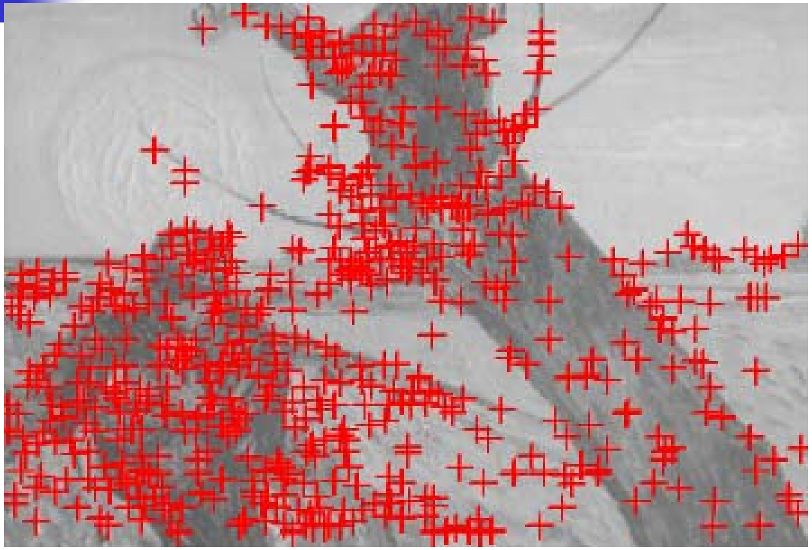


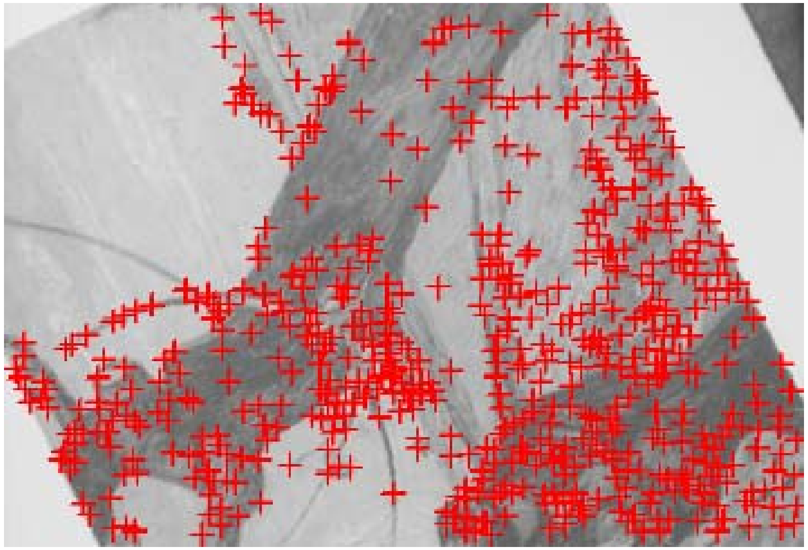
Corner Detection



Corners are detected where the product of the ellipse axis lengths reaches a local maximum.

Harris Corners



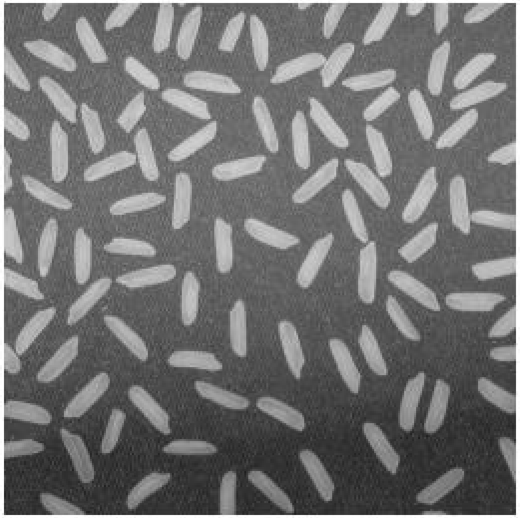


- Originally developed as features for motion tracking
- Greatly reduces amount of computation compared to tracking every pixel
- Translation and rotation invariant (but not scale invariant)

Harris Corner: Matlab code

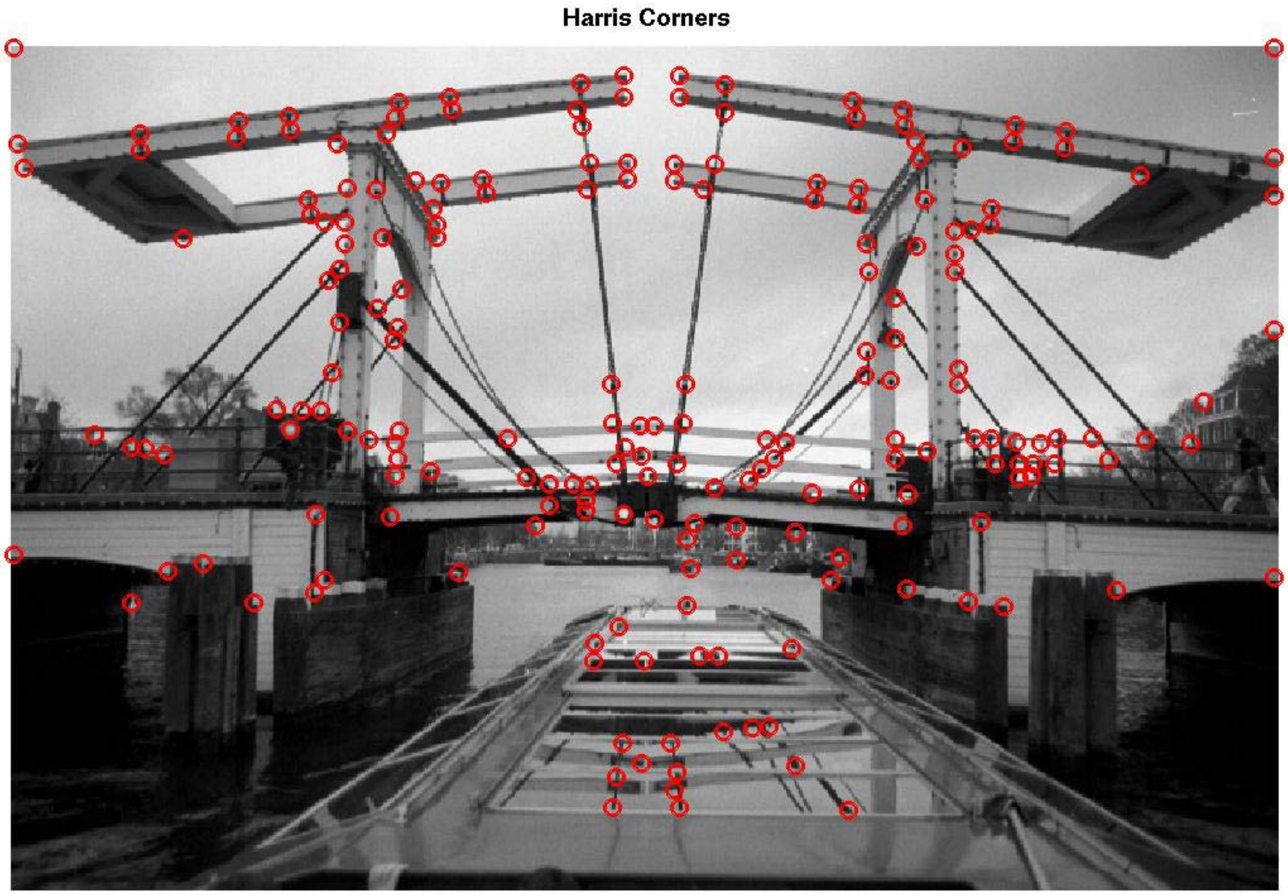
```
% Harris Corner detector - by Kashif Shahzad
sigma=2; thresh=0.1; sze=11; disp=0;eps=0.0;
dy = [-1 0 1; -1 0 1; -1 0 1];    % Derivative masks
dx = dy'; %dx is the transpose matrix of dy
% Ix and Iy are the horizontal and vertical edges of image
I = imread('rice.png');
imshow(I);
title('\bf Original image');%use bold font for the title
bw=double(I);%convert uint8 to double
Ix = conv2(bw, dx, 'same'); % Calculating the gradient
Iy = conv2(bw, dy, 'same'); %return a matrix the sane
g = fspecial('gaussian',max(1,fix(6*sigma)), sigma); %
Ix2 = conv2(Ix.^2, g, 'same'); %Smoothed squared image
Iy2 = conv2(Iy.^2, g, 'same');
Ixy = conv2(Ix.*Iy, g, 'same');
cornerness = (Ix2.*Iy2 - Ixy.^2)./(Ix2 + Iy2 + eps); %
mx = ordfilt2(cornerness,sze^2,ones(sze));             %
cornerness = (cornerness==mx)&(cornerness>thresh);   %
[rws,cols] = find(cornerness);                         %
figure;imshow(bw,[0 255]);
hold on;
p=[cols rws];
plot(p(:,1),p(:,2),'or'); % display corners as red cir
title('\bf Harris Corners');
```

Original image

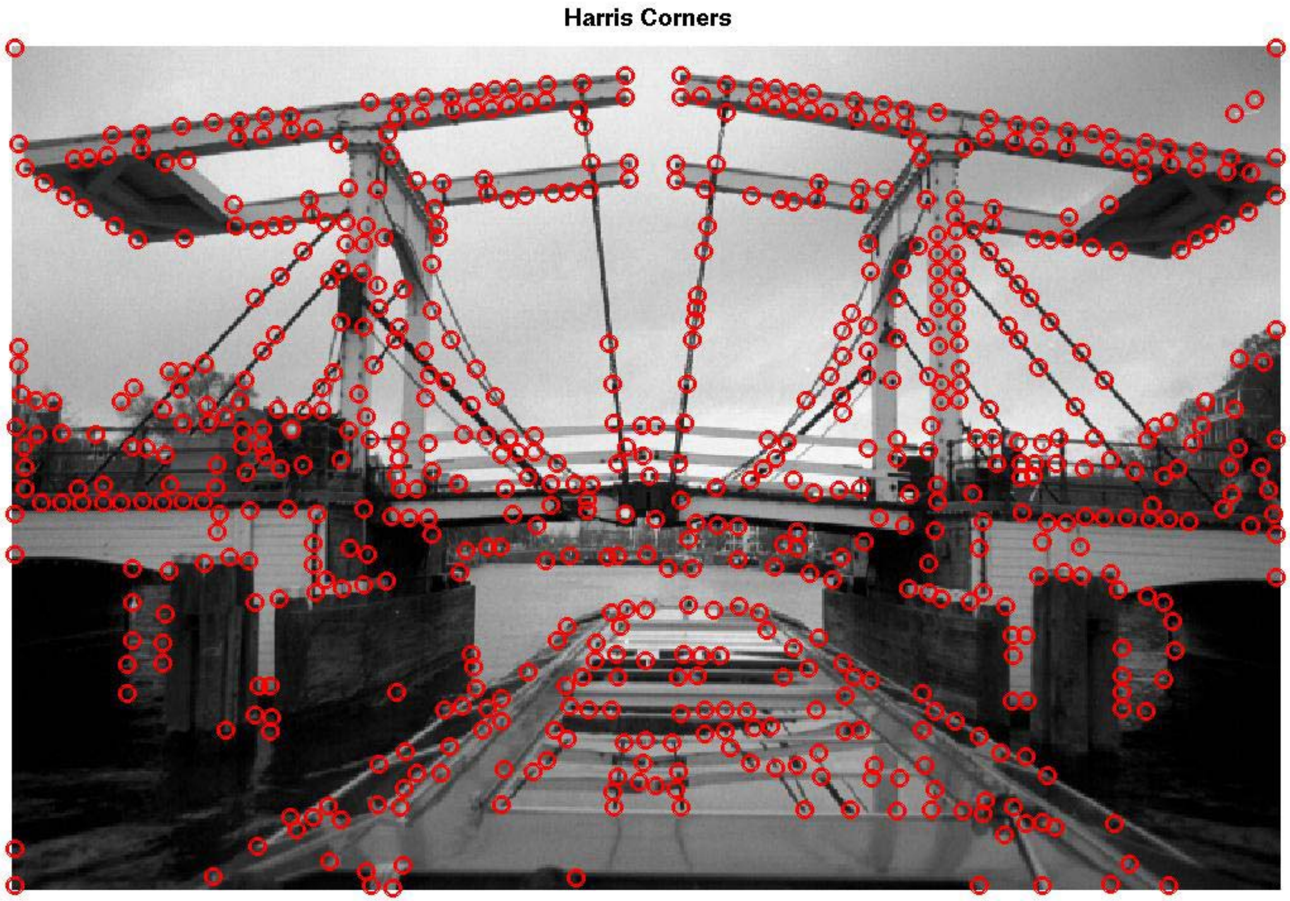


Harris Corners

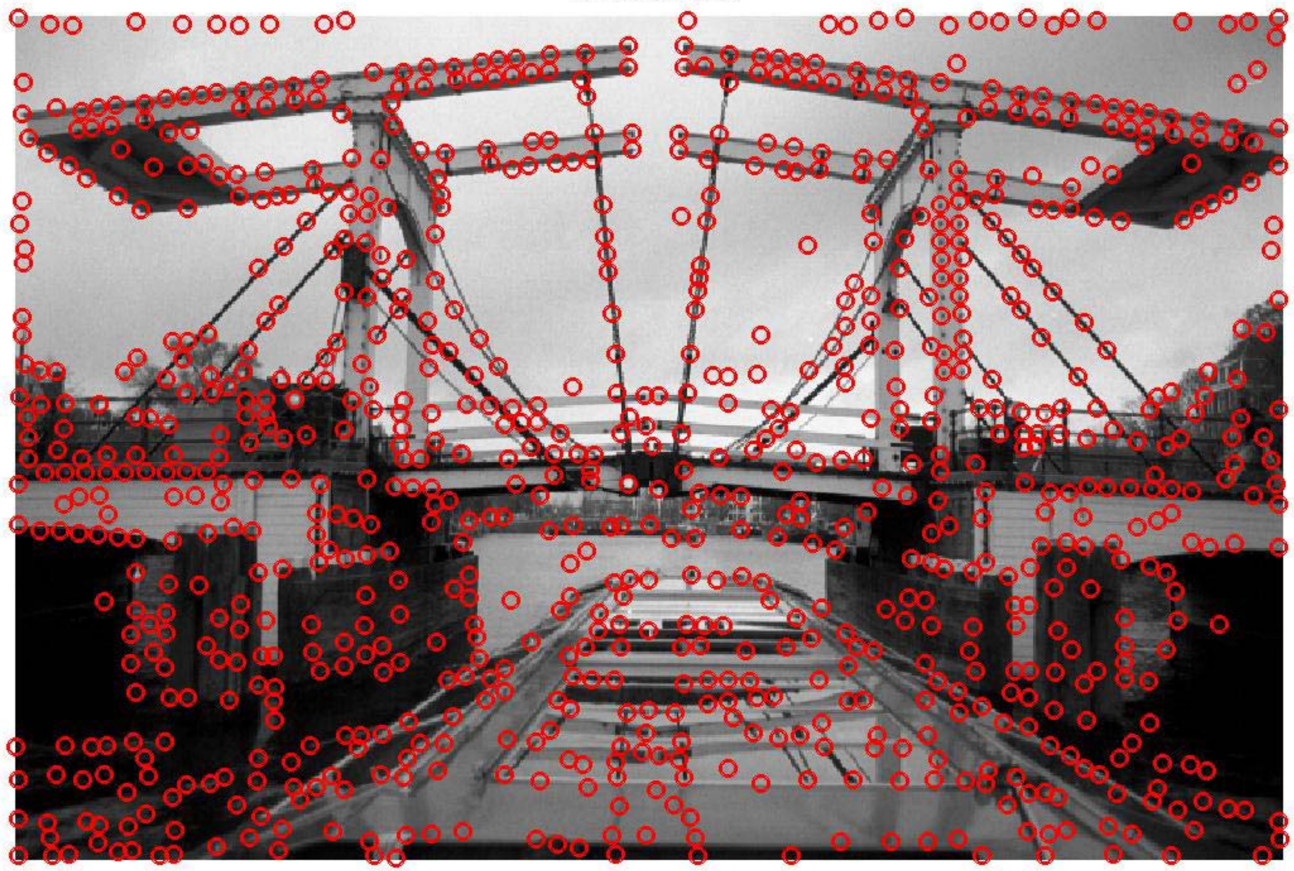
Example (σ=0.1)



Example (σ=0.01)



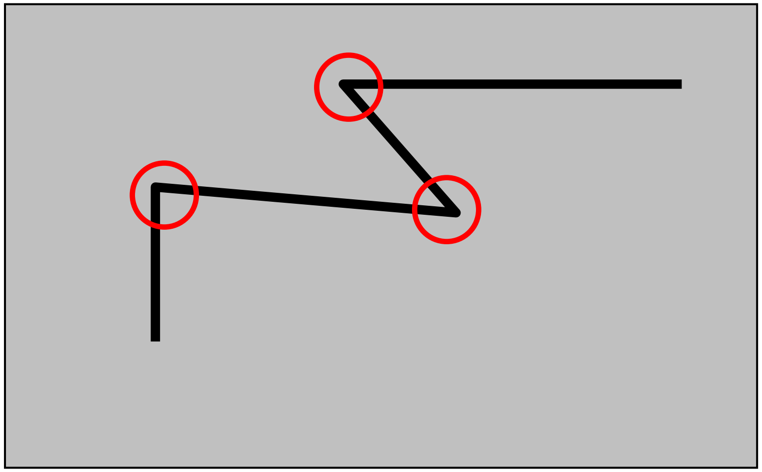
Example (σ=0.001)



Reading: Matching with Invariant Features (www.cs.washington.edu, computer vision course)

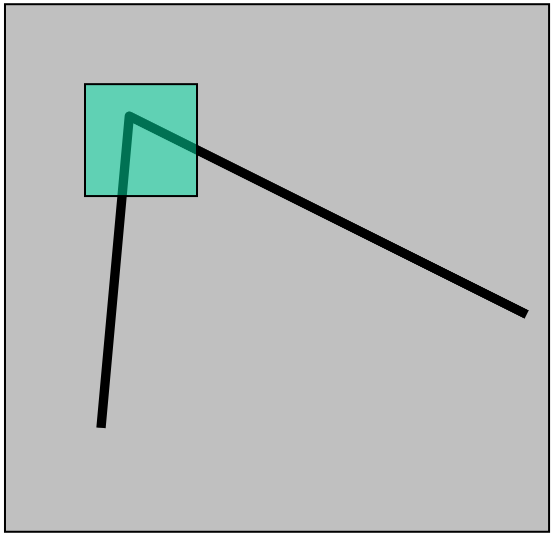
Harris corner detector

- C.Harris, M.Stephens. “A Combined Corner and Edge Detector”. 1988

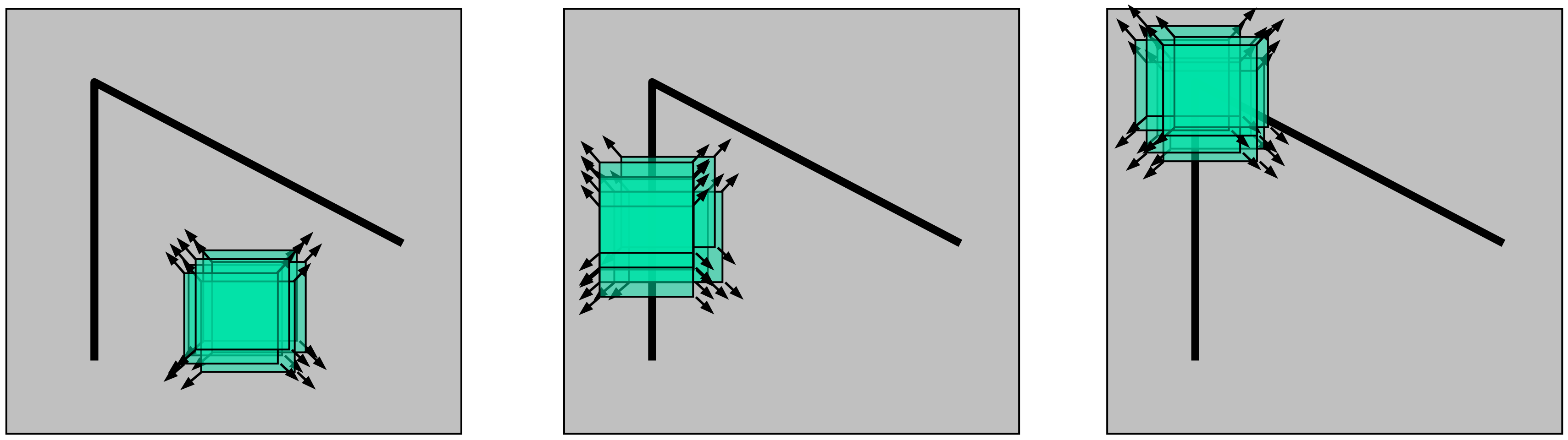


The Basic Idea

- We should easily recognize the point by looking through a small window
- Shifting a window in *any direction* should give *a large change* in intensity



Harris Detector: Basic Idea



"flat" region:
no change in
all directions

"edge":
no change along
the edge direction

"corner":
significant change
in all directions

Harris Detector: Mathematics

Change of intensity for the shift [u,v]:

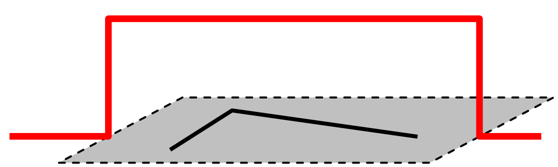
$$E(u,v)=\sum_{x,y} w(x,y)\left[I(x+u,y+v)-I(x,y)\right]^2$$

Window function

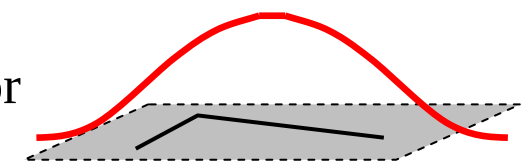
Shifted intensity

Intensity

Window function $w(x,y)$ =



or



1 in window, 0 outside

Gaussian

Harris Detector: Mathematics

For small shifts $[u,v]$ we have a *bilinear* approximation:

$$E(u,v) \cong [u,v] \; M \; \begin{bmatrix} u \\ v \end{bmatrix}$$

where M is a 2×2 matrix computed from image derivatives:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

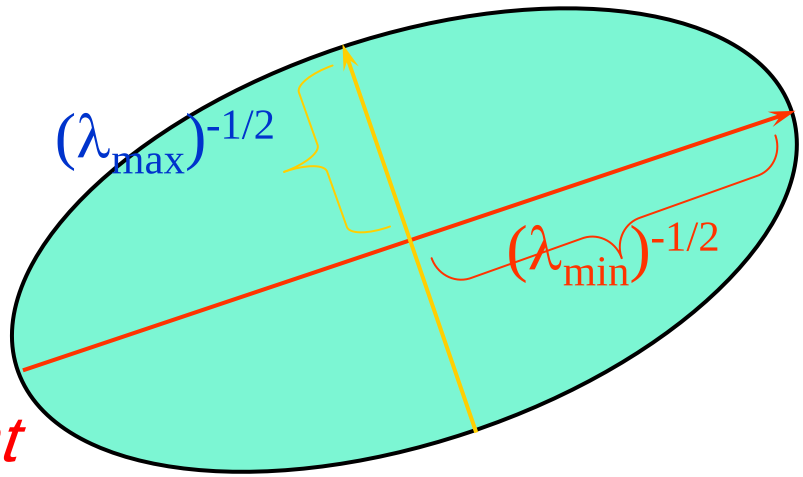
Harris Detector: Mathematics

Intensity change in shifting window: eigenvalue analysis

$$E(u,v) \cong [u,v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

λ_1, λ_2 – eigenvalues of M

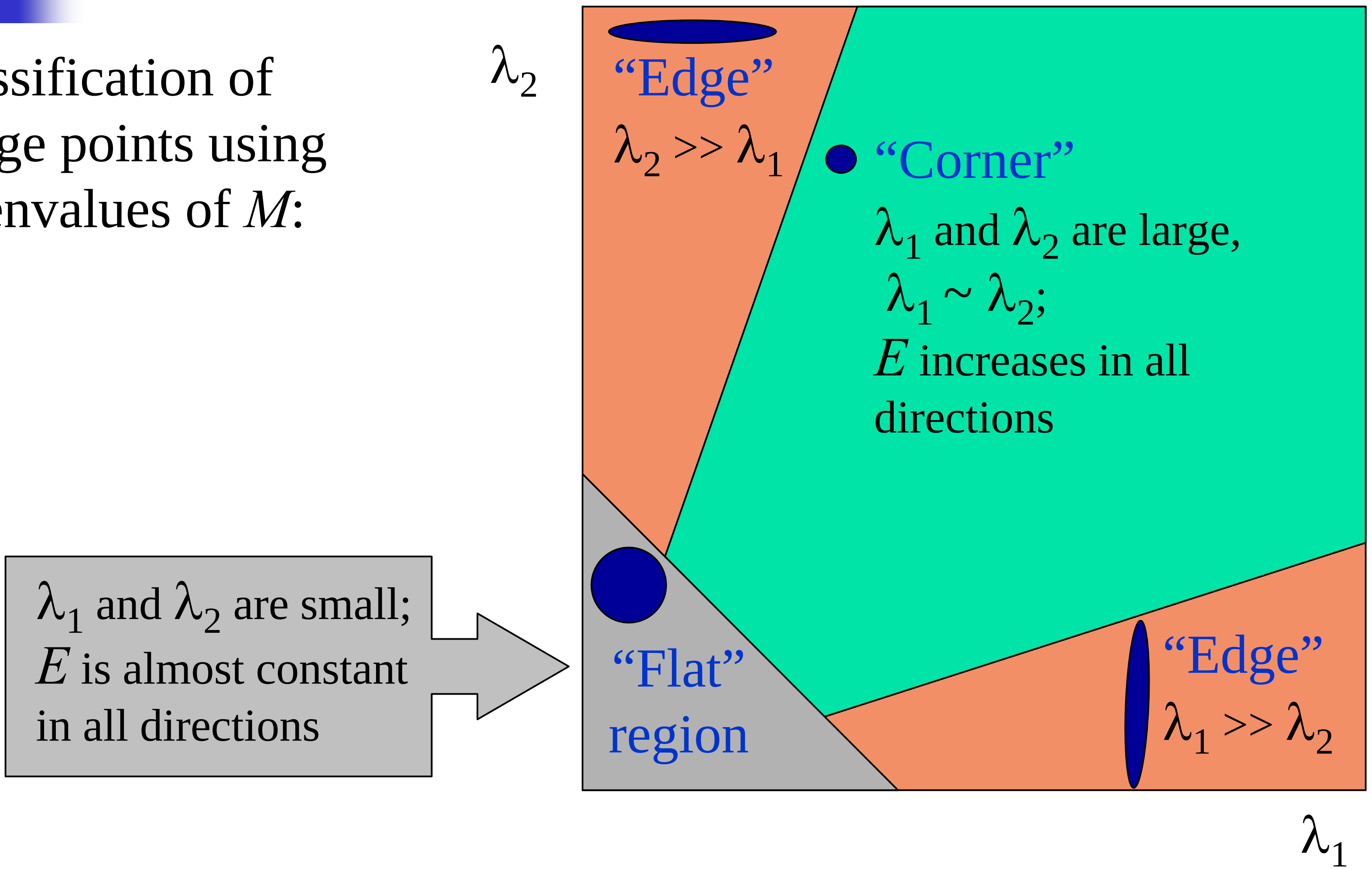
Ellipse $E(u,v)$ = const



Why? Optional assignment

Harris Detector: Mathematics

Classification of image points using eigenvalues of M:



Harris Detector: Mathematics

Measure of corner response:

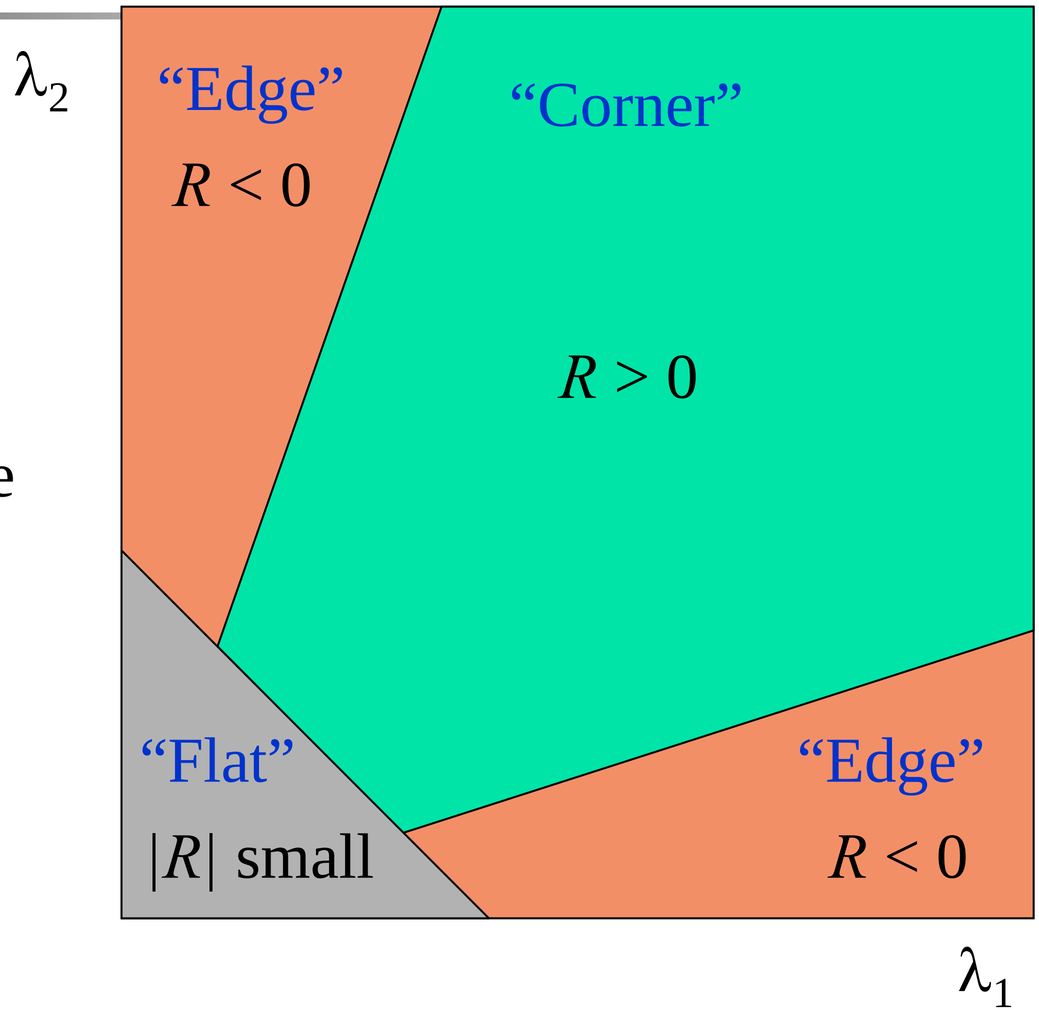
$$R = \det M - k\left(\operatorname{trace} M\right)^2$$

$$\det M = \lambda_1 \lambda_2$$
$$\operatorname{trace} M = \lambda_1 + \lambda_2$$

(k – empirical constant, k = 0.04-0.06)

Harris Detector: Mathematics

- R depends only on eigenvalues of M
- R is large for a corner
- R is negative with large magnitude for an edge
- $|R|$ is small for a flat region



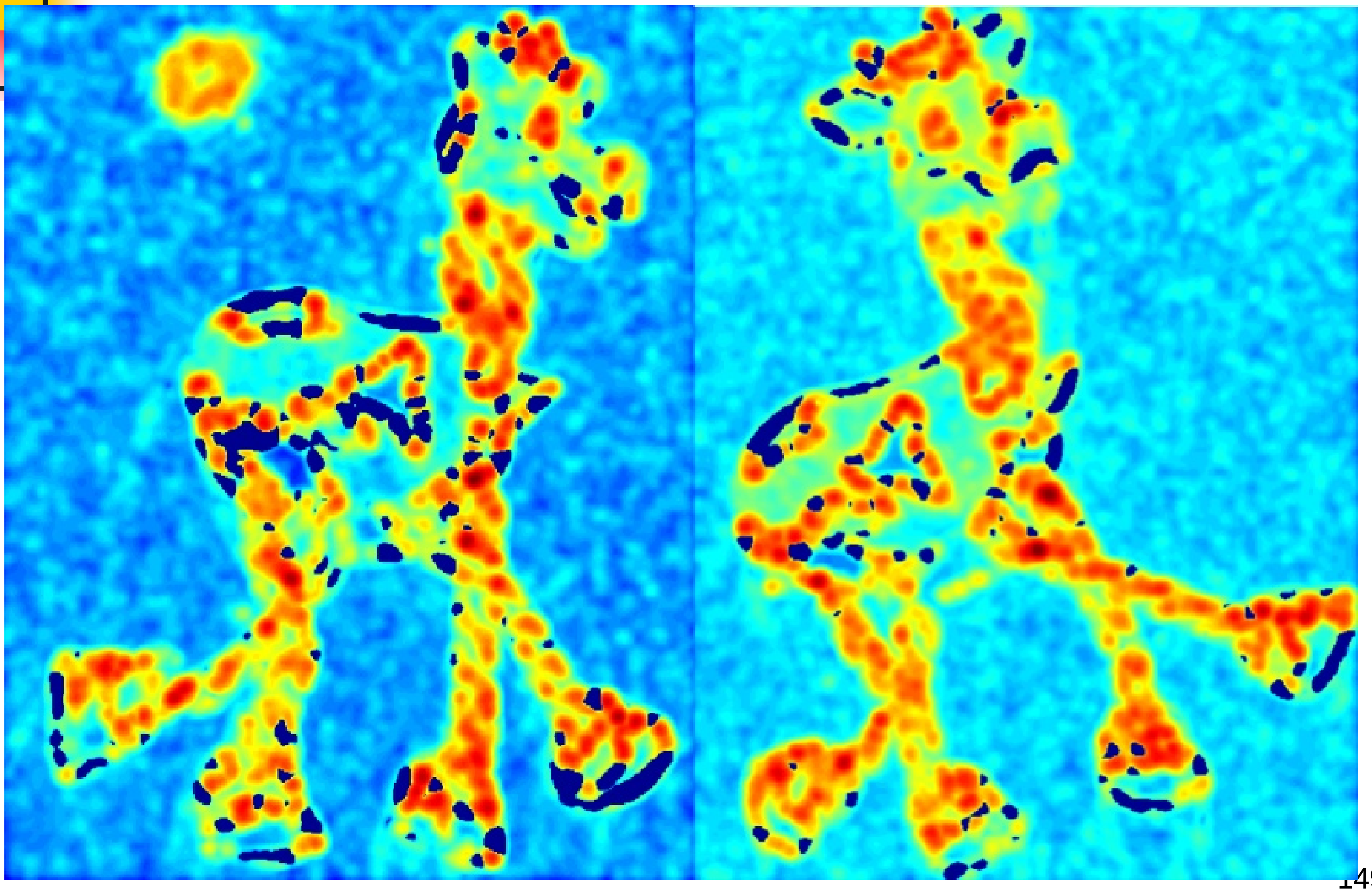
Harris Detector

- The Algorithm:
  - Find points with large corner response function R ($R >$ threshold)
  - Take the points of local maxima of R

Harris Detector: Workflow

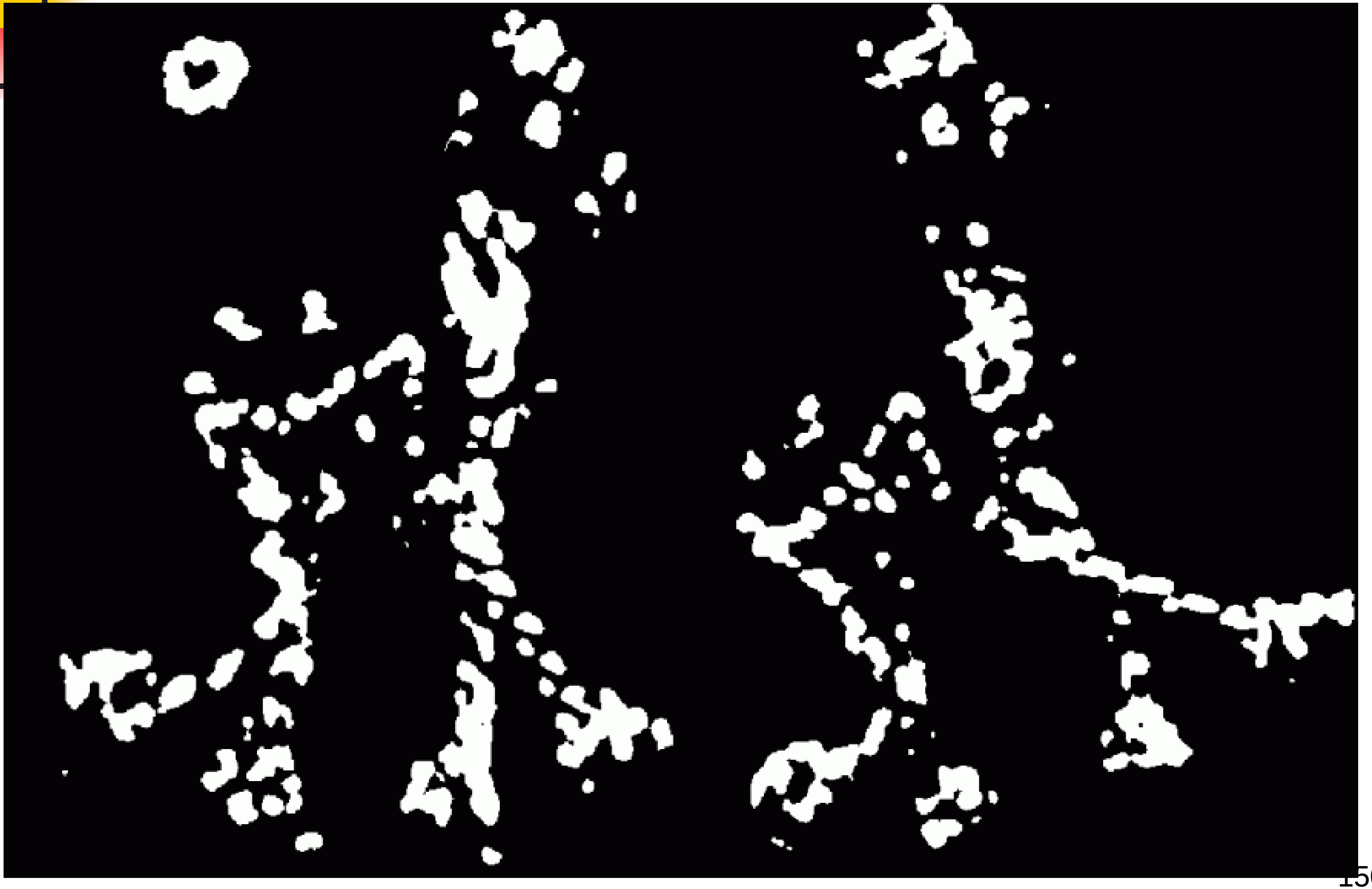
Harris Detector: Workflow

- Compute corner response R



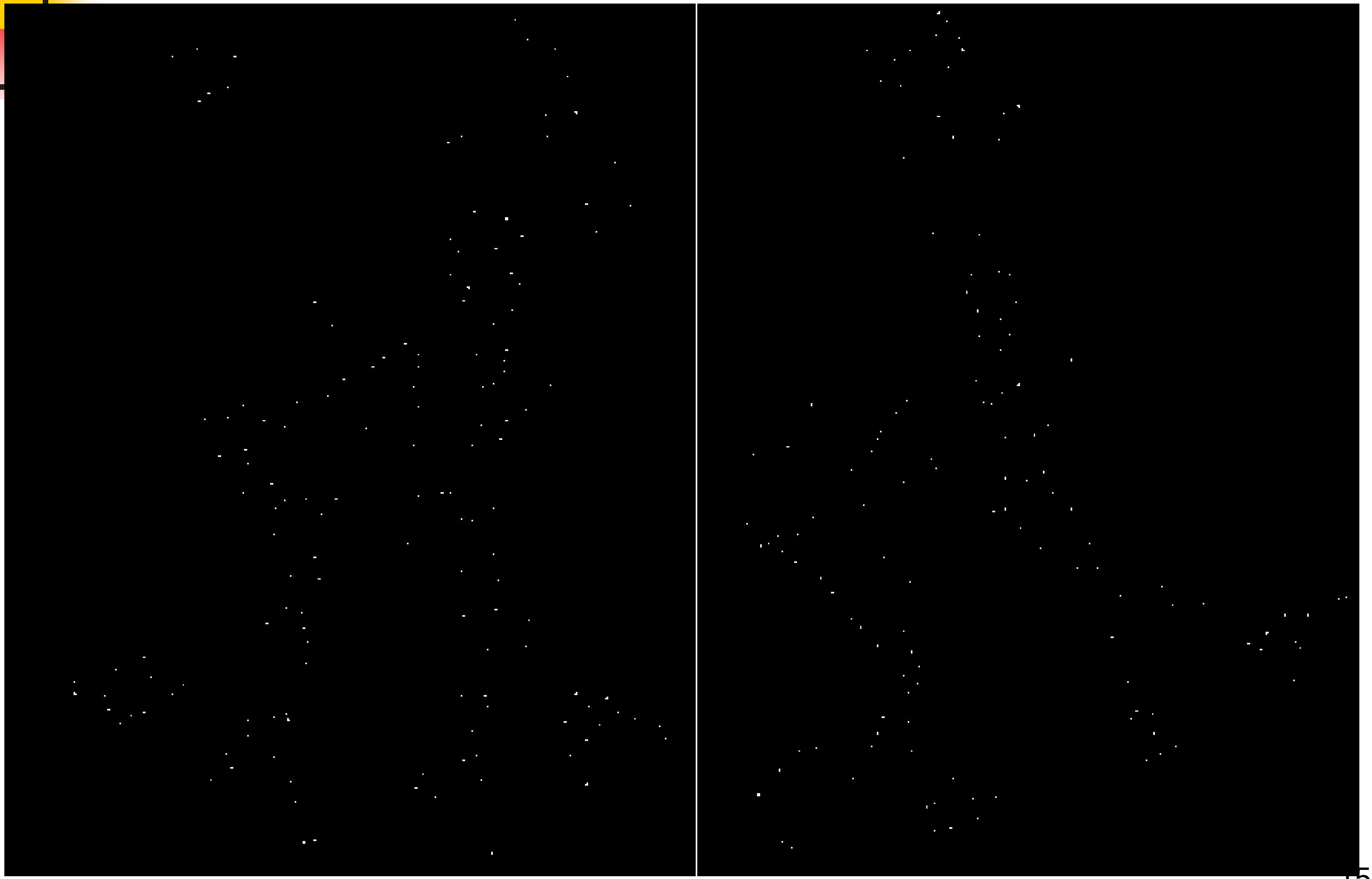
Harris Detector: Workflow

- Find points with large corner response: R>threshold



Harris Detector: Workflow

- Take only the points of local maxima of R



Harris Detector: Workflow

Harris Detector: Summary

- Average intensity change in direction $[u, v]$ can be expressed as a bilinear form:

$$E(u,v) \cong [u,v] \; M \begin{bmatrix} u \\ v \end{bmatrix}$$

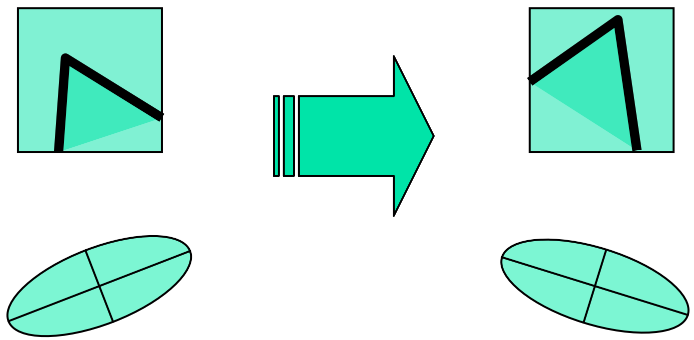
- Describe a point in terms of eigenvalues of *M*: *measure of corner response*

$$R = \lambda_1 \lambda_2 - k\left(\lambda_1 + \lambda_2\right)^2$$

- A good (corner) point should have a *large intensity change* in *all directions*, i.e. *R* should be large positive

Harris Detector: Some Properties

- Rotation invariance



Ellipse rotates but its shape (i.e. eigenvalues) remains the same

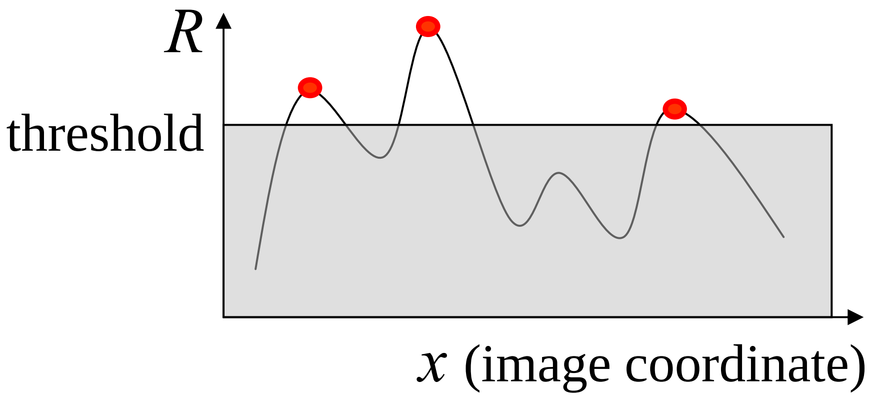
Corner response R is invariant to image rotation

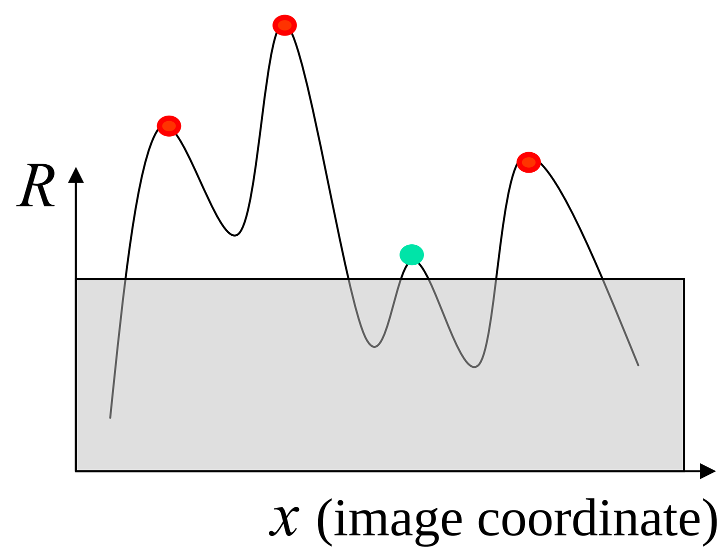
Harris Detector: Some Properties

- Partial invariance to *affine intensity* change

✓ Only derivatives are used =>
invariance to intensity shift: $I \to I + b$

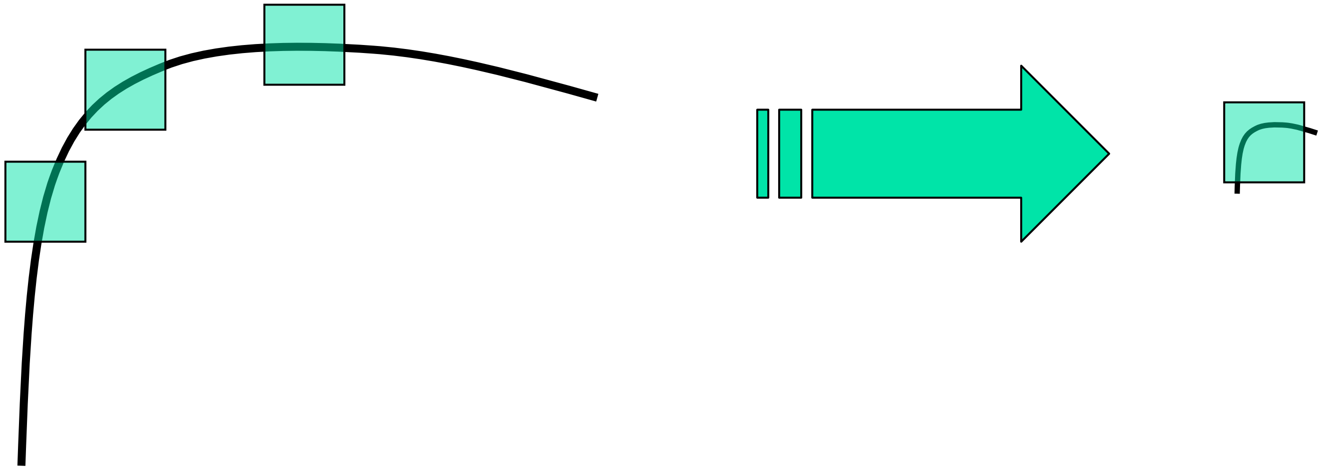
✓ Intensity scale: $I \to a\,I$





Harris Detector: Some Properties

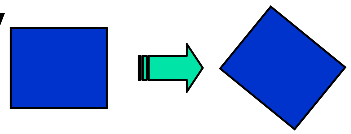
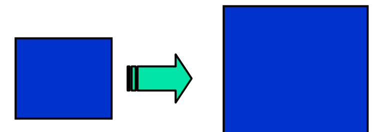
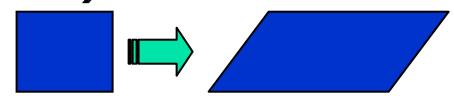
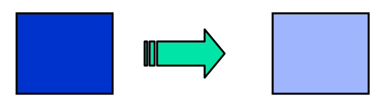
- But: non-invariant to *image scale*!



All points will be classified as edges

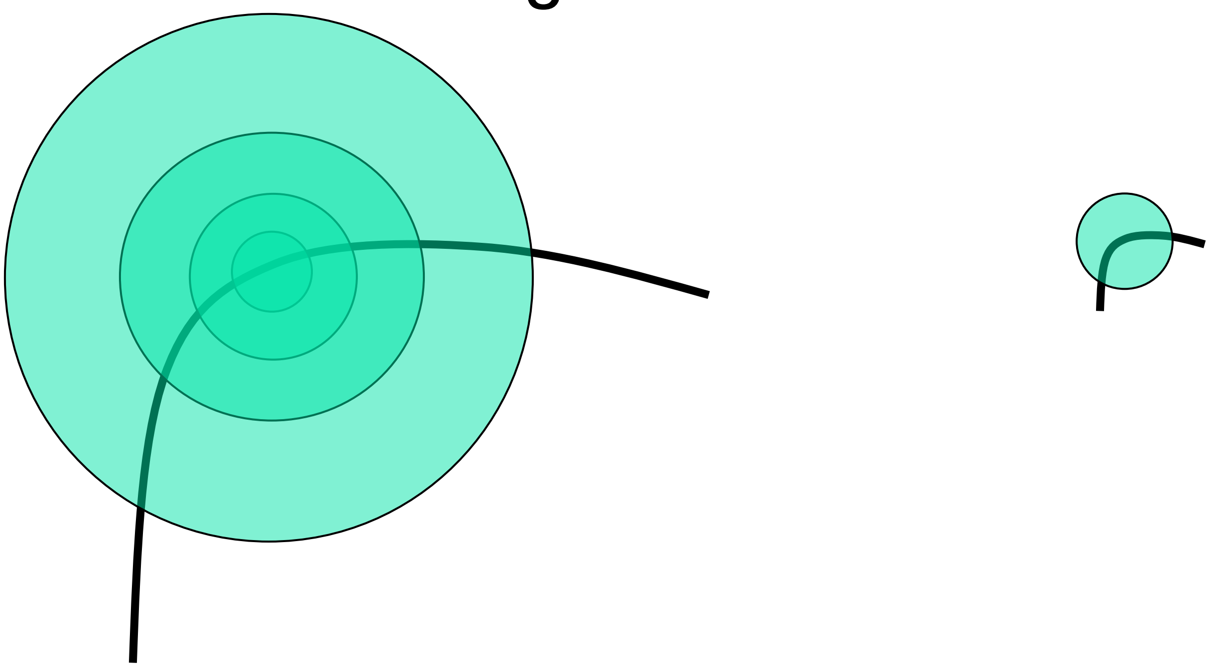
Corner !

Models of Image Change

- Geometry 
  - Rotation
  
  - Similarity (rotation + uniform scale)
  
  - Affine (scale dependent on direction) valid for: orthographic camera, locally planar object
- Photometry
  
  - Affine intensity change ($I \to a\, I + b$)

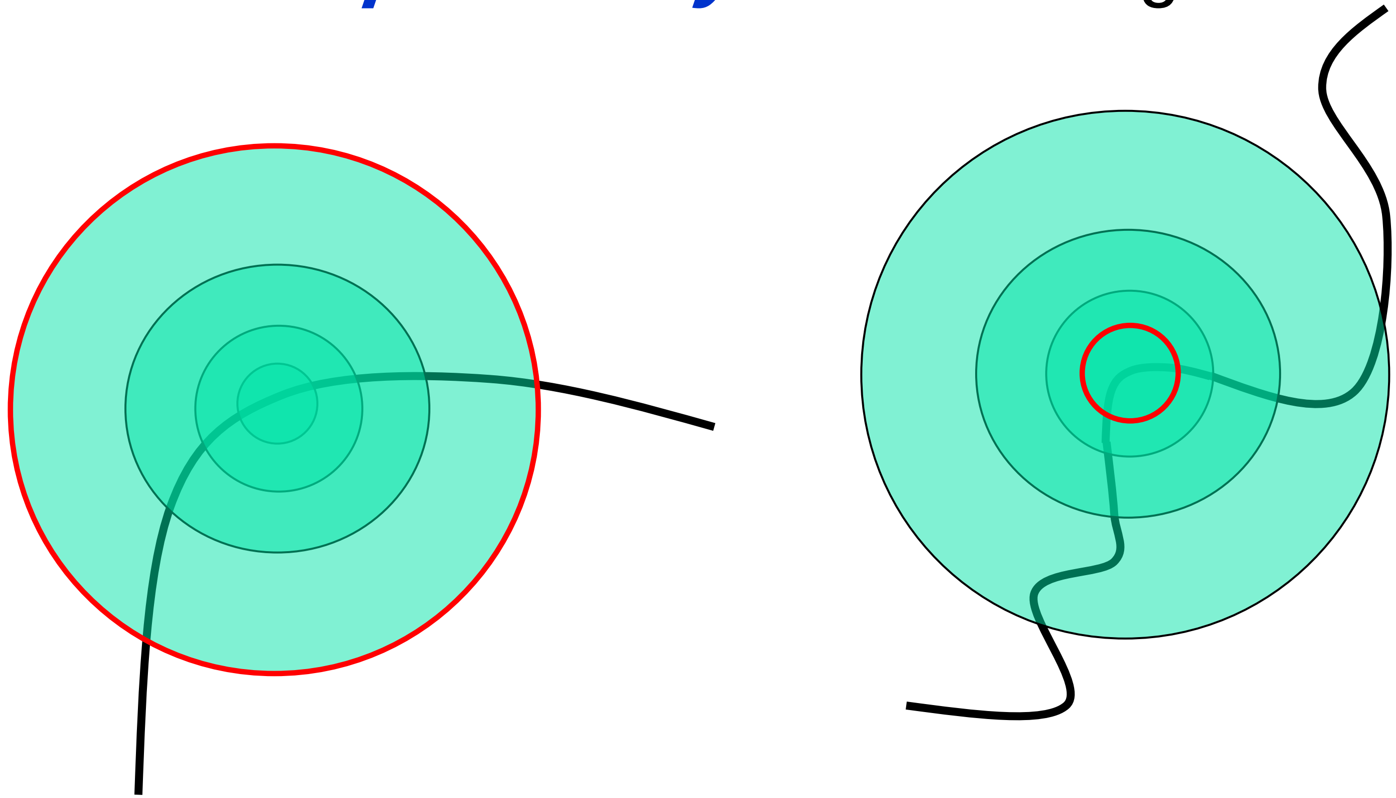
Scale Invariant Detection

- Consider regions (e.g. circles) of different sizes around a point
- Regions of corresponding sizes will look the same in both images



Scale Invariant Detection

- The problem: how do we choose corresponding circles ***independently*** in each image?

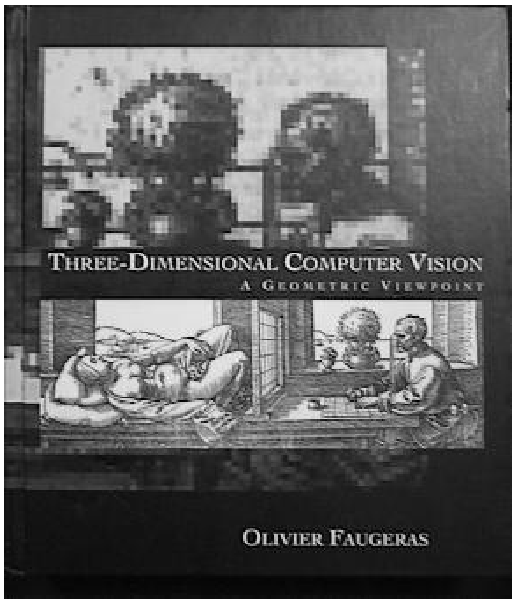


Today's Goals

- Features Overview
- Canny Edge Detector
- Harris Corner Detector
- Templates and Image Pyramid
- SIFT Features

Problem: Features for Recognition

Want to find



… in here

Correlation(相关)

template

How do we locate the template in the image?

Minimize

$$E(i,j)=\sum_m\sum_n\left[f(m,n)-t(m-i,n-j)\right]^2$$
$$=\sum_m\sum_n\left[f^2(m,n)+t^2(m-i,n-j)-2f(m,n)t(m-i,n-j)\right]$$

Maximize

$$R_{tf}(i,j)=\sum_m\sum_n t(m-i,n-j)f(m,n)$$

Cross-correlation

Correlation (相关)

- Cauchy inequality (柯西不等式)

$$a^2 + b^2 + c^2 \geq ab + bc + ca$$

R{(a,b,c), (a,b,c)}> R{(a,b,c), (b,c,a)}

$$4a^2 + 4b^2 + 4c^2 \geq a^2 + b^2 + c^2$$

R{(a,b,c), (4a,4b,4c)}> R{(a,b,c), (a,b,c)} ?

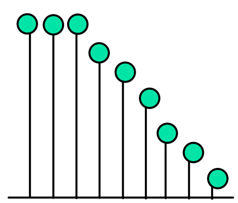
Cross-correlation (互相关)

$$R_{tf}(i,j) = \sum_m \sum_n t(m-i, n-j) f(m,n) \qquad R_{tf} = t \otimes f$$

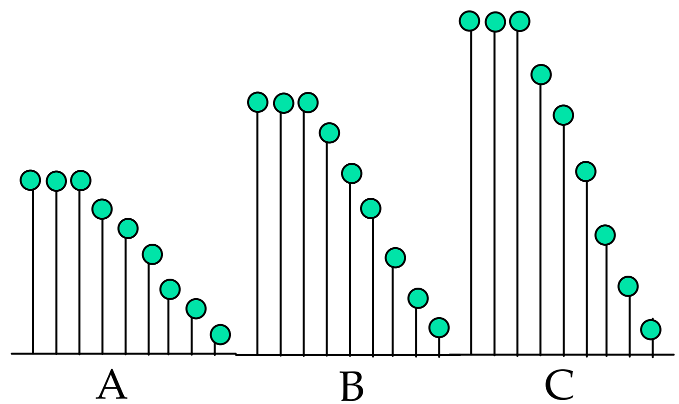
Note: $t \otimes f \neq f \otimes t$

$$R_{ff} = f \otimes f$$ Auto-correlation

Problem:



t



f

$R_{tf}(C) > R_{tf}(B) > R_{tf}(A)$ We need $R_{tf}(A)$ to be the maximum!

Correlation

- Cauchy inequality (柯西不等式)

Corr(A,B)=dot(A,B)/sqrt(|A||B|)

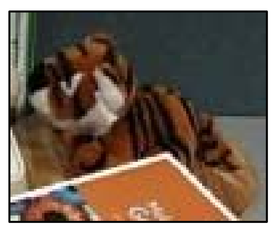
Corr{(a,b,c), (4a,4b,4c)}= Corr{(a,b,c), (a,b,c)}=1.0

Normalized Correlation

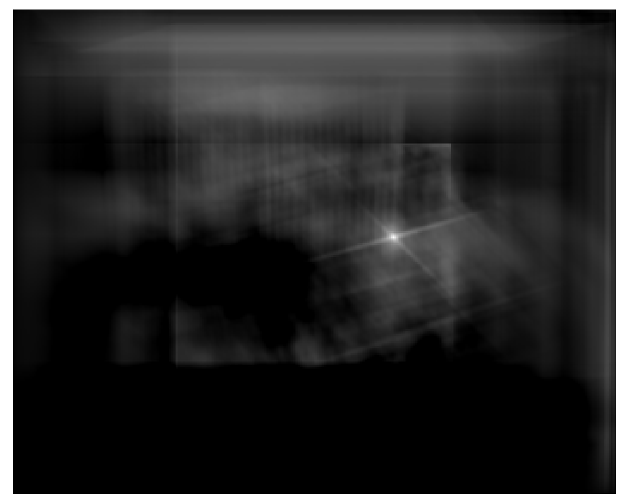
Account for energy differences

$$N_{tf}(i,j) = \frac{\sum_m \sum_n t(m-i, n-j) f(m,n)}{\left[\sum_m \sum_n t^2(m-i, n-i)\right]^{1/2} \left[\sum_m \sum_n f^2(m,n)\right]^{1/2}}$$

$\otimes$



$=$



Normalized Correlation

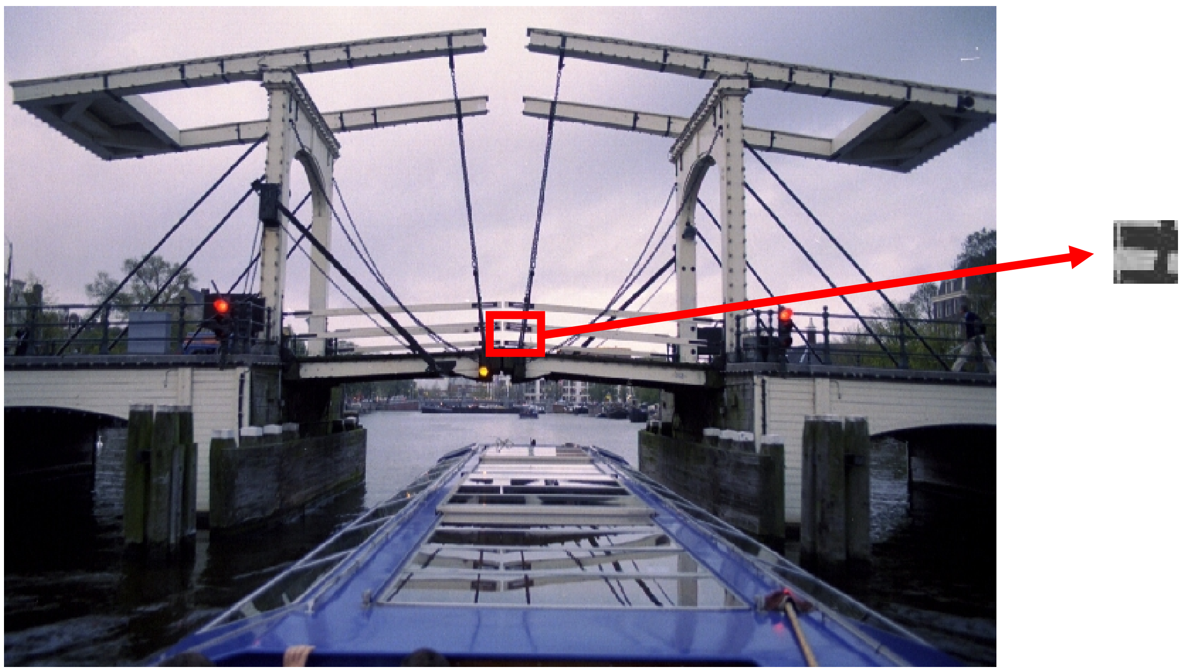
```
onion = imread('onion.png');
peppers = imread('peppers.png');
imshow(onion);
figure, imshow(peppers);
rect_onion = [111 33 65 58];
rect_peppers = [163 47 143 151];
sub_onion = imcrop(onion,rect_onion);
sub_peppers = imcrop(peppers,rect_peppers);
c = normxcorr2(sub_onion(:,:,1),sub_peppers(:,:,1));
[max_c, imax] = max(abs(c(:)));
[ypeak, xpeak] = ind2sub(size(c),imax(1));
corr_offset = [(xpeak-size(sub_onion,2))； (ypeak-size(sub_onion,1))];
rect_offset = [(rect_peppers(1)-rect_onion(1))； (rect_peppers(2)-rect_onion(2))];
offset = corr_offset + rect_offset;
xoffset = offset(1);
yoffset = offset(2);
xbegin = round(xoffset+1);
xend   = round(xoffset+ size(onion,2));
ybegin = round(yoffset+1);
yend   = round(yoffset+size(onion,1));
extracted_onion = peppers(ybegin:yend,xbegin:xend,:);
recovered_onion = uint8(zeros(size(peppers)));
recovered_onion(ybegin:yend,xbegin:xend,:) = onion;
[m,n,p] = size(peppers);
mask = ones(m,n);
i = find(recovered_onion(:,:,1)==0);
mask(i) = .2;
figure, imshow(peppers(:,:,1))；
hold on；
h = imshow(recovered_onion);
set(h,'AlphaData',mask);
```

Templates

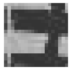
- Find an object in an image!

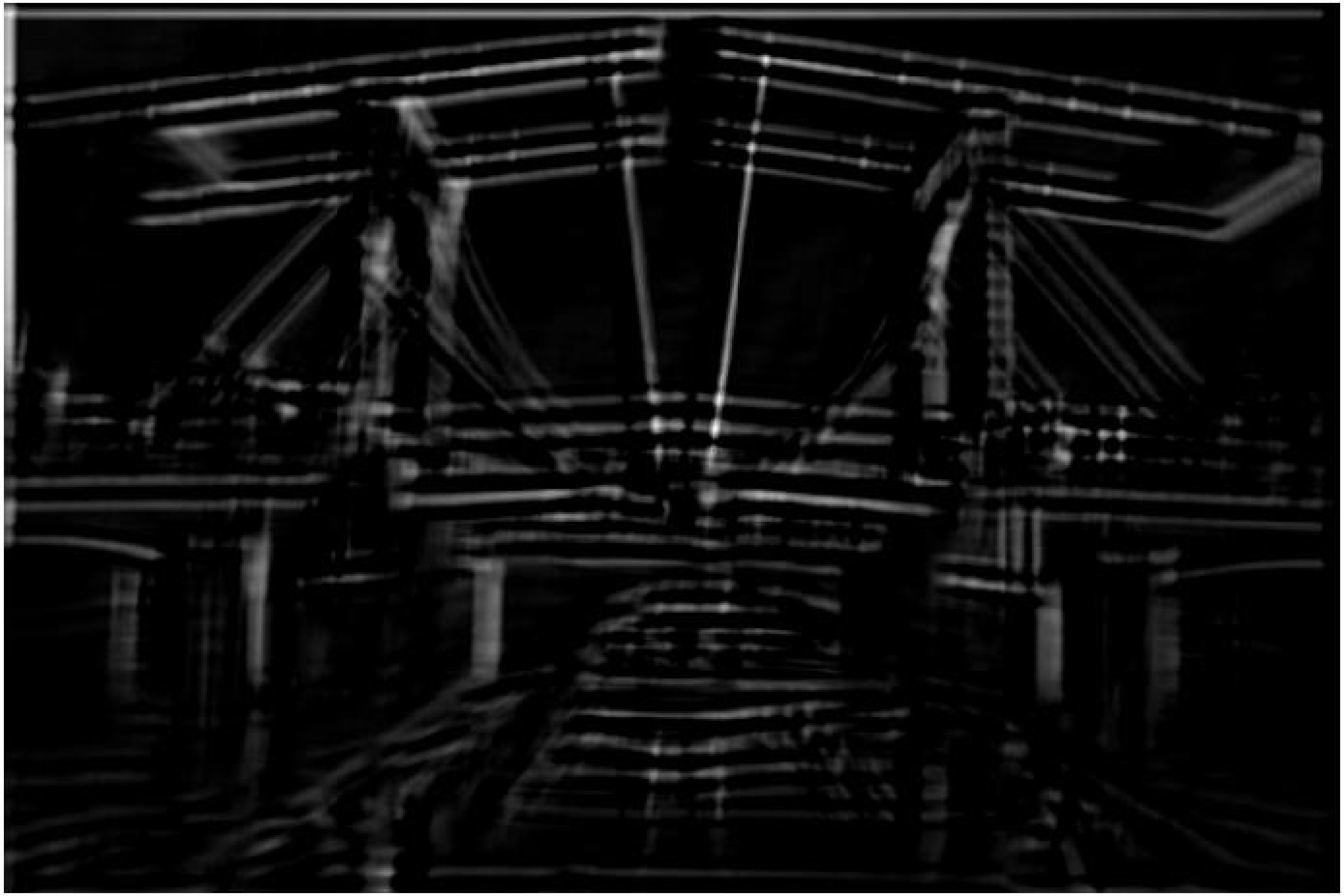
- Want Invariance!
  - Scaling
  - Rotation
  - Illumination
  - Deformation

Template Convolution



Template Convolution

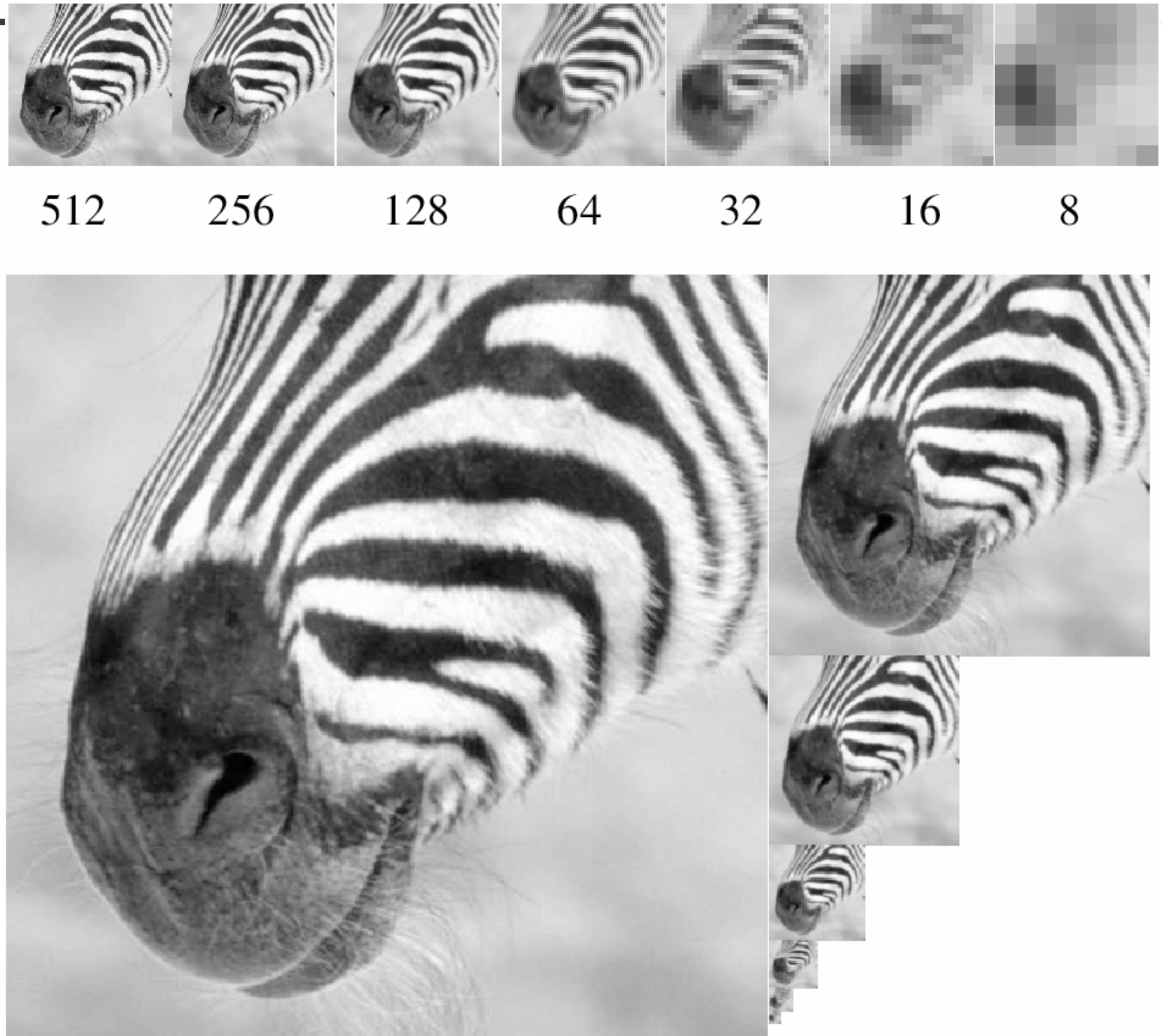




Convolution with Templates

- Invariances:
  - Scaling — No
  - Rotation — No
  - Illumination — No
  - Deformation — Maybe
- Provides
  - Good localization — No

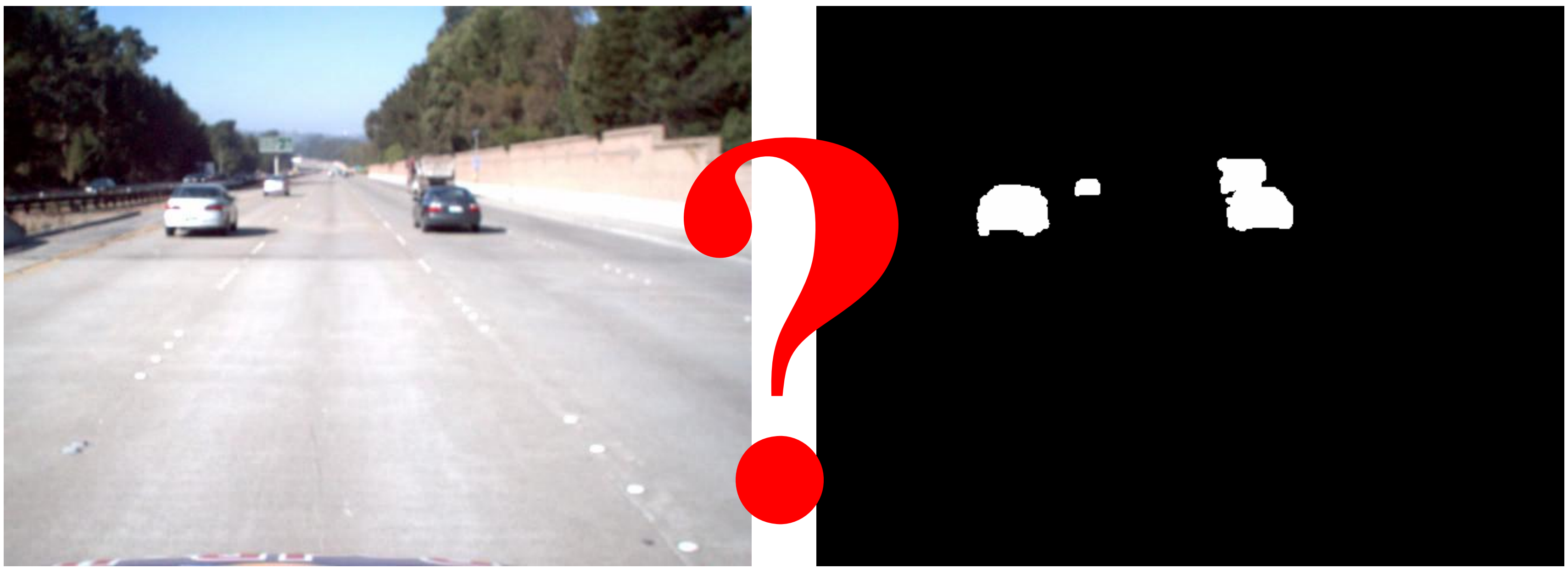
Scale Invariance: Image Pyramid



Templates with Image Pyramid

- Invariances:
  - Scaling — Yes
  - Rotation — No
  - Illumination — No
  - Deformation — Maybe
- Provides
  - Good localization — No

Templates



Optional Assignment—Feature detector

- Point feature detector
- Line feature detector
- Conic feature detector
- Invariance under different cases
- Feature matching/Correspondence
- ……

Today's Goals

- Canny Edge Detector
- Harris Corner Detector
- Hough Transform
- Templates and Image Pyramid
- SIFT Features

This is the end of features....

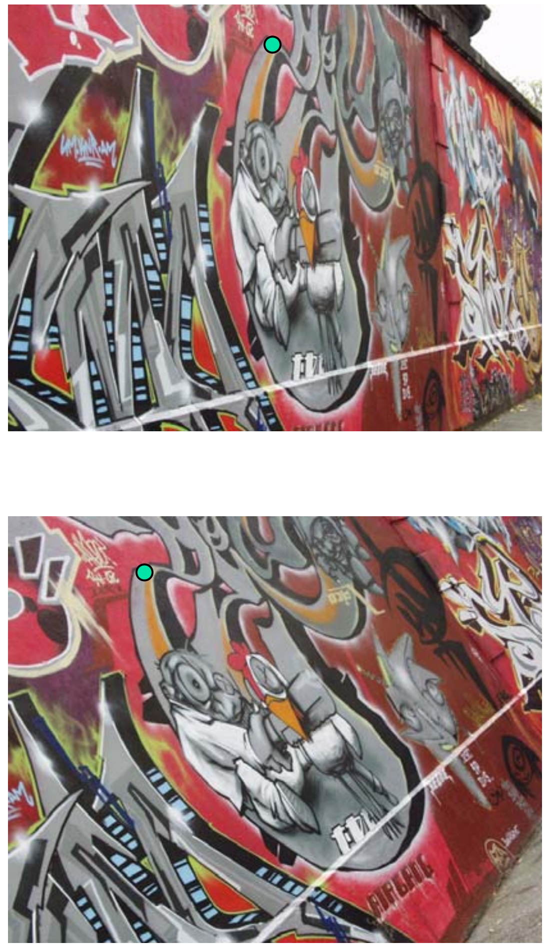
Appendix

SIFT

- Invariances:
  - Scaling — Yes
  - Rotation — Yes
  - Illumination — Yes
  - Deformation — Maybe
- Provides
  - Good localization — Yes

Invariance to …

Scaling and rotation



Viewpoint

Illumination

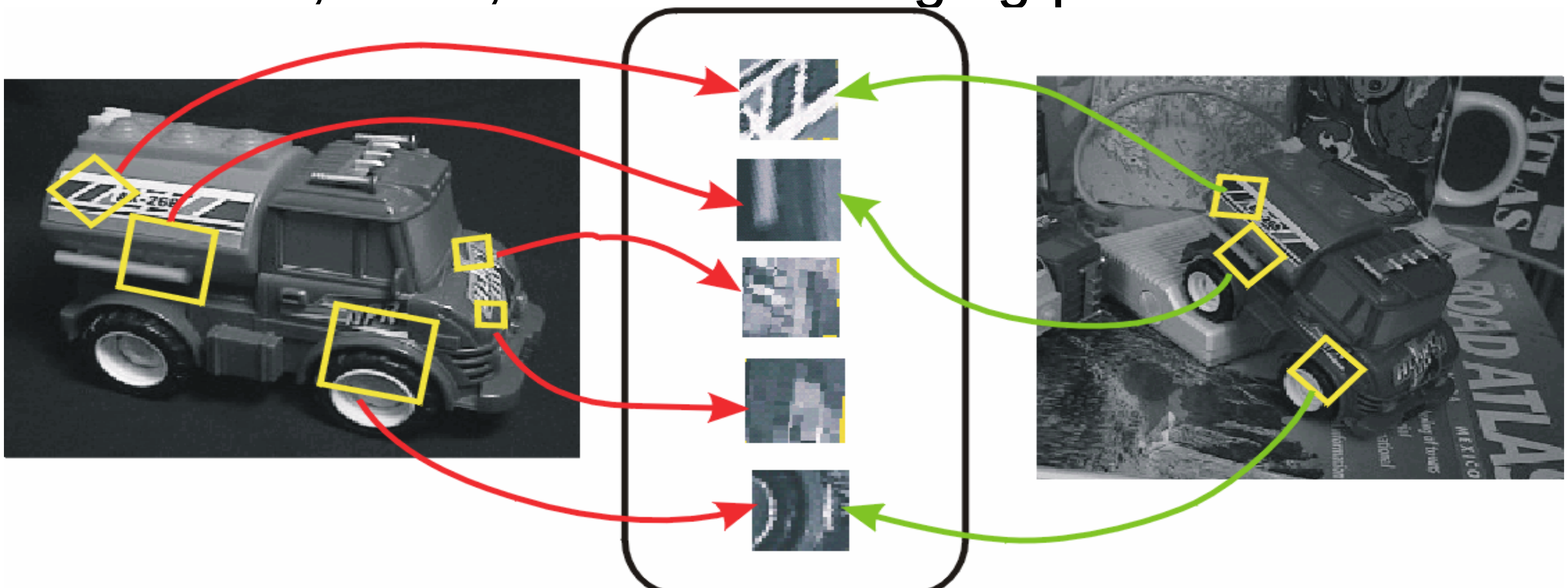
SIFT Reference

Distinctive image features from scale-invariant keypoints. David G. Lowe, International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

SIFT = Scale Invariant Feature Transform

Invariant Local Features

- Image content is transformed into local feature coordinates that are invariant to translation, rotation, scale, and other imaging parameters



SIFT Features

Advantages of invariant local features

- **Locality:** features are local, so robust to occlusion and clutter (no prior segmentation)
- **Distinctiveness:** individual features can be matched to a large database of objects
- **Quantity:** many features can be generated for even small objects
- **Efficiency:** close to real-time performance
- **Extensibility:** can easily be extended to wide range of differing feature types, with each adding robustness

SIFT On-A-Slide

1. **Enforce invariance to scale:** Compute Gaussian difference max, for may different scales; non-maximum suppression, find local maxima: keypoint candidates
2. **Localizable corner:** For each maximum fit quadratic function. Compute center with sub-pixel accuracy by setting first derivative to zero.
3. **Eliminate edges:** Compute ratio of eigenvalues, drop keypoints for which this ratio is larger than a threshold.
4. **Enforce invariance to orientation:** Compute orientation, to achieve scale invariance, by finding the strongest second derivative direction in the smoothed image (possibly multiple orientations). Rotate patch so that orientation points up.
5. **Compute feature signature:** Compute a "gradient histogram" of the local image region in a 4x4 pixel region. Do this for 4x4 regions of that size. Orient so that largest gradient points up (possibly multiple solutions). Result: feature vector with 128 values (15 fields, 8 gradients).
6. **Enforce invariance to illumination change and camera saturation:** Normalize to unit length to increase invariance to illumination. Then threshold all gradients, to become invariant to camera saturation.

SIFT On-A-Slide

1. **Enforce invariance to scale:** Compute Gaussian difference max, for may different scales; non-maximum suppression, find local maxima: keypoint candidates
2. **Localizable corner:** For each maximum fit quadratic function. Compute center with sub-pixel accuracy by setting first derivative to zero.
3. **Eliminate edges:** Compute ratio of eigenvalues, drop keypoints for which this ratio is larger than a threshold.
4. **Enforce invariance to orientation:** Compute orientation, to achieve scale invariance, by finding the strongest second derivative direction in the smoothed image (possibly multiple orientations). Rotate patch so that orientation points up.
5. **Compute feature signature:** Compute a "gradient histogram" of the local image region in a 4x4 pixel region. Do this for 4x4 regions of that size. Orient so that largest gradient points up (possibly multiple solutions). Result: feature vector with 128 values (15 fields, 8 gradients).
6. **Enforce invariance to illumination change and camera saturation:** Normalize to unit length to increase invariance to illumination. Then threshold all gradients, to become invariant to camera saturation.

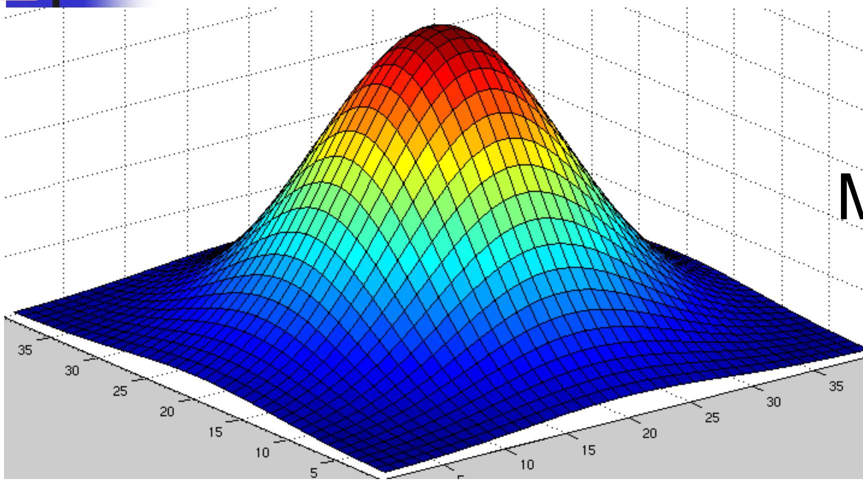
Finding “Keypoints” (Corners)

Idea: Find Corners, but scale invariance

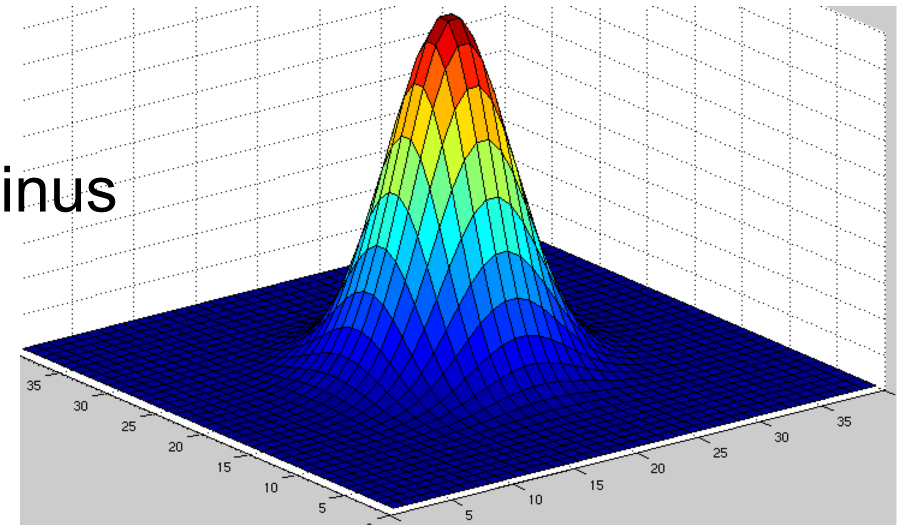
Approach:

- Run linear filter (diff of Gaussians)
- At different resolutions of image pyramid

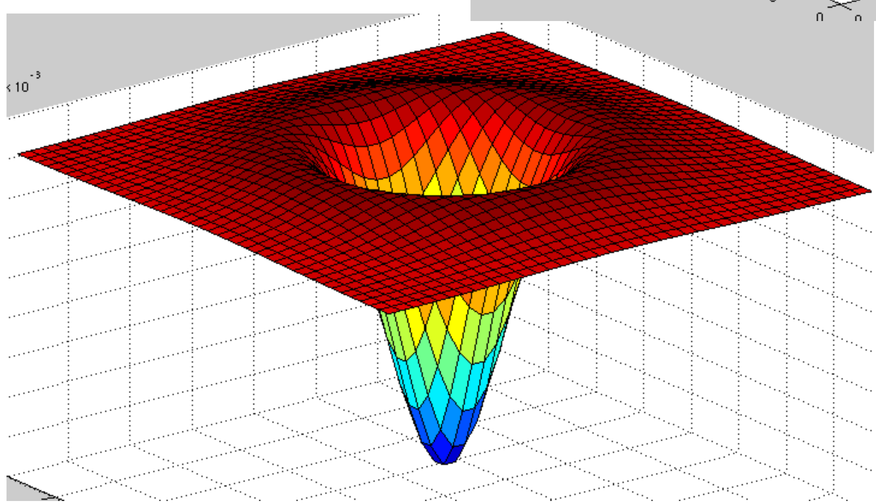
Difference of Gaussians



Minus



Equals



Difference of Gaussians

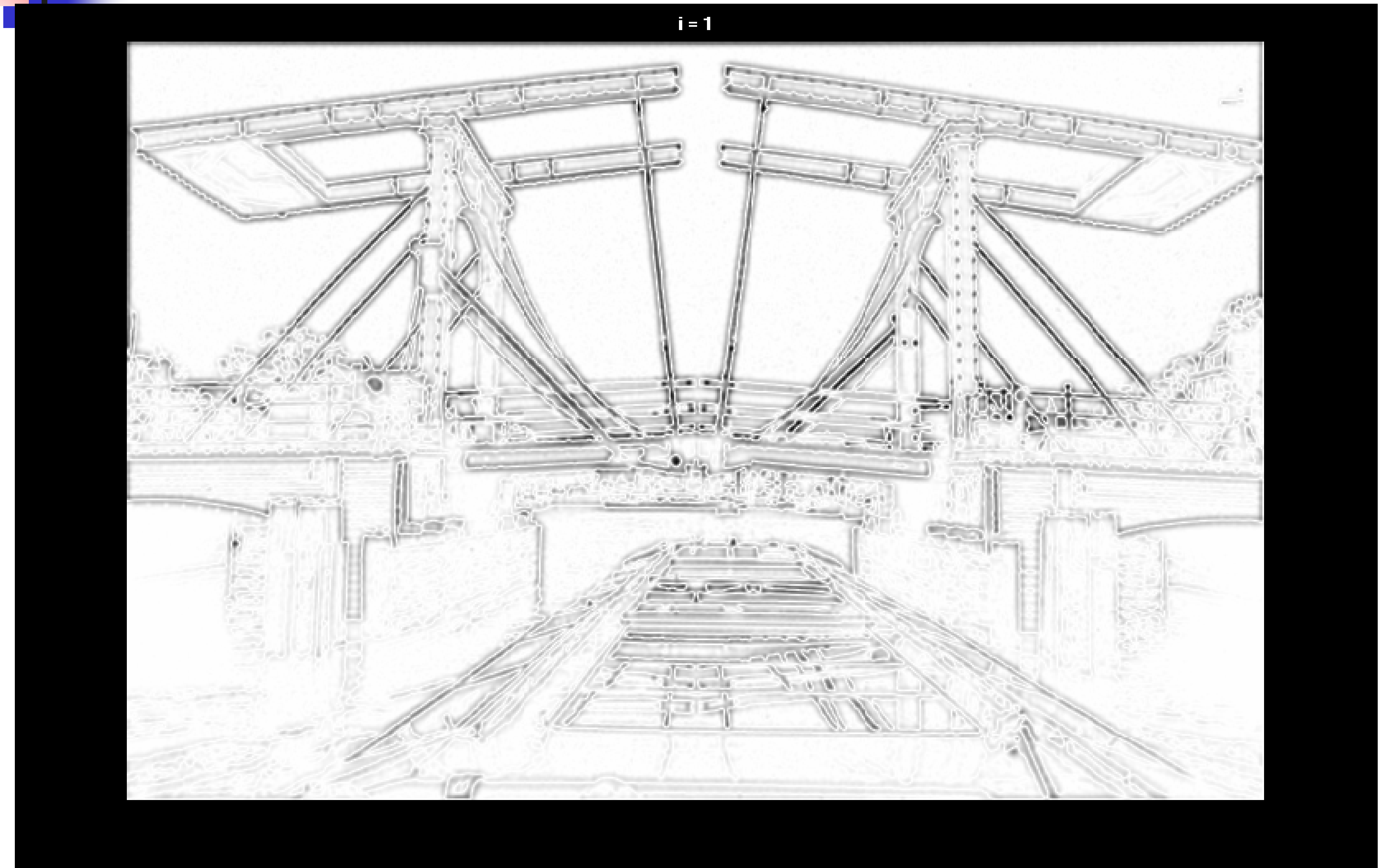
```
surf(fspecial('gaussian',40,4))
surf(fspecial('gaussian',40,8))
surf(fspecial('gaussian',40,8) - fspecial('gaussian',40,4))
```

Find Corners with DiffOfGauss

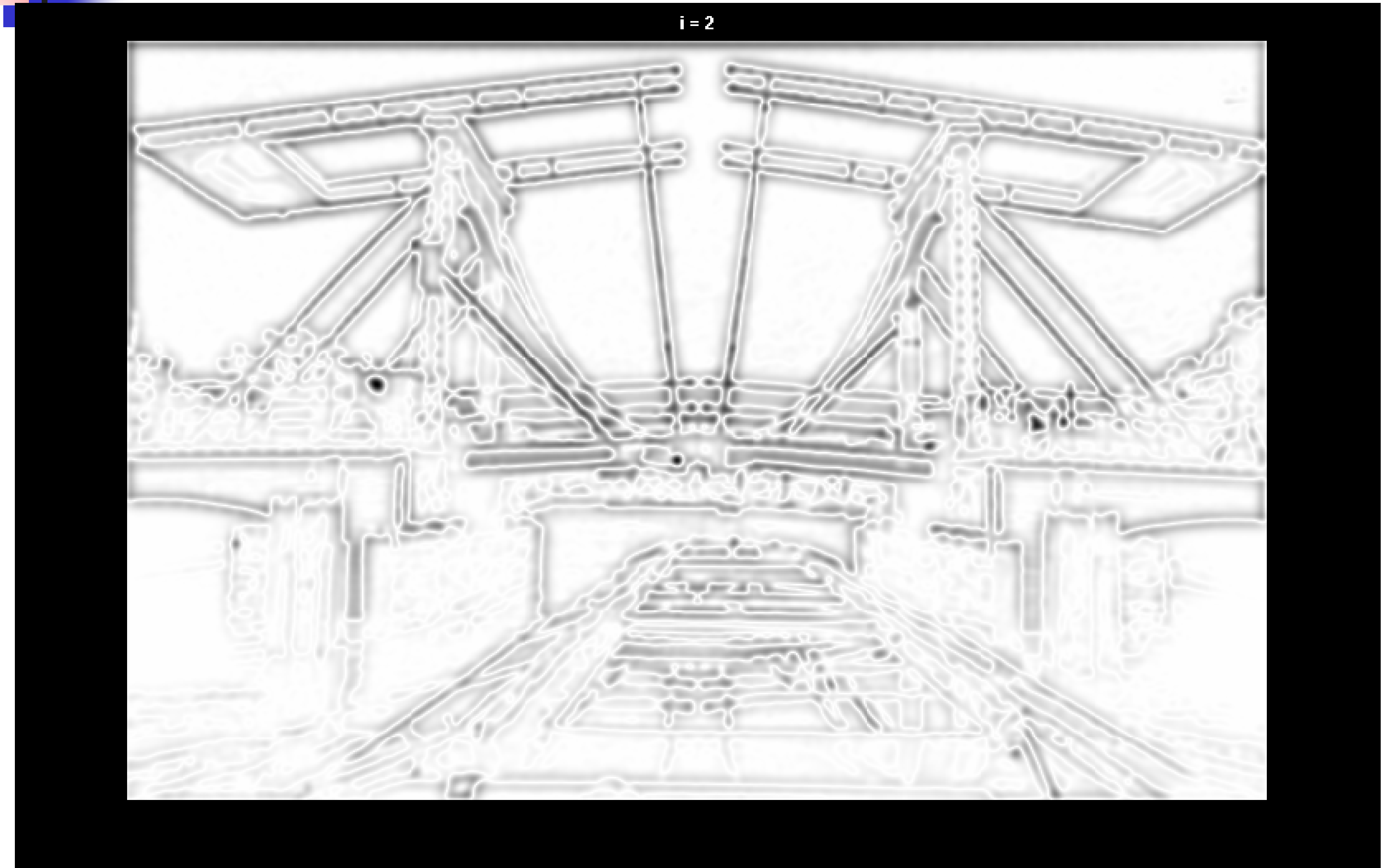
```
im =imread('bridge.jpg');
bw = double(im(:,:,1)) / 256;

for i = 1 : 10
  gaussD = fspecial('gaussian',40,2*i) -
    fspecial('gaussian',40,i);
  res = abs(conv2(bw, gaussD,  'same'));
  res = res / max(max(res));
  imshow(res) ; title(['\bf i = ' num2str(i)]); drawnow
end
```

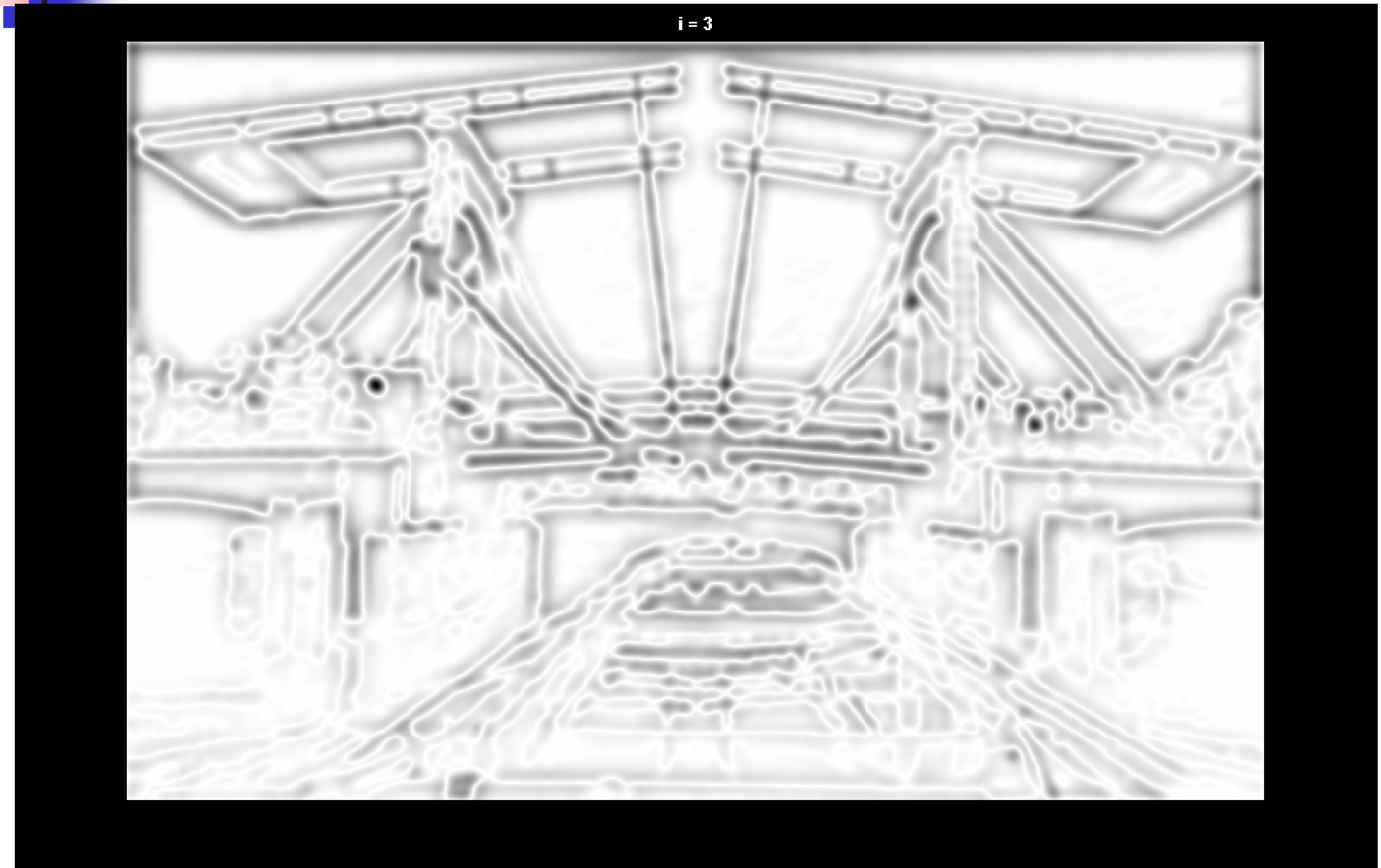
Gaussian Kernel Size i=1



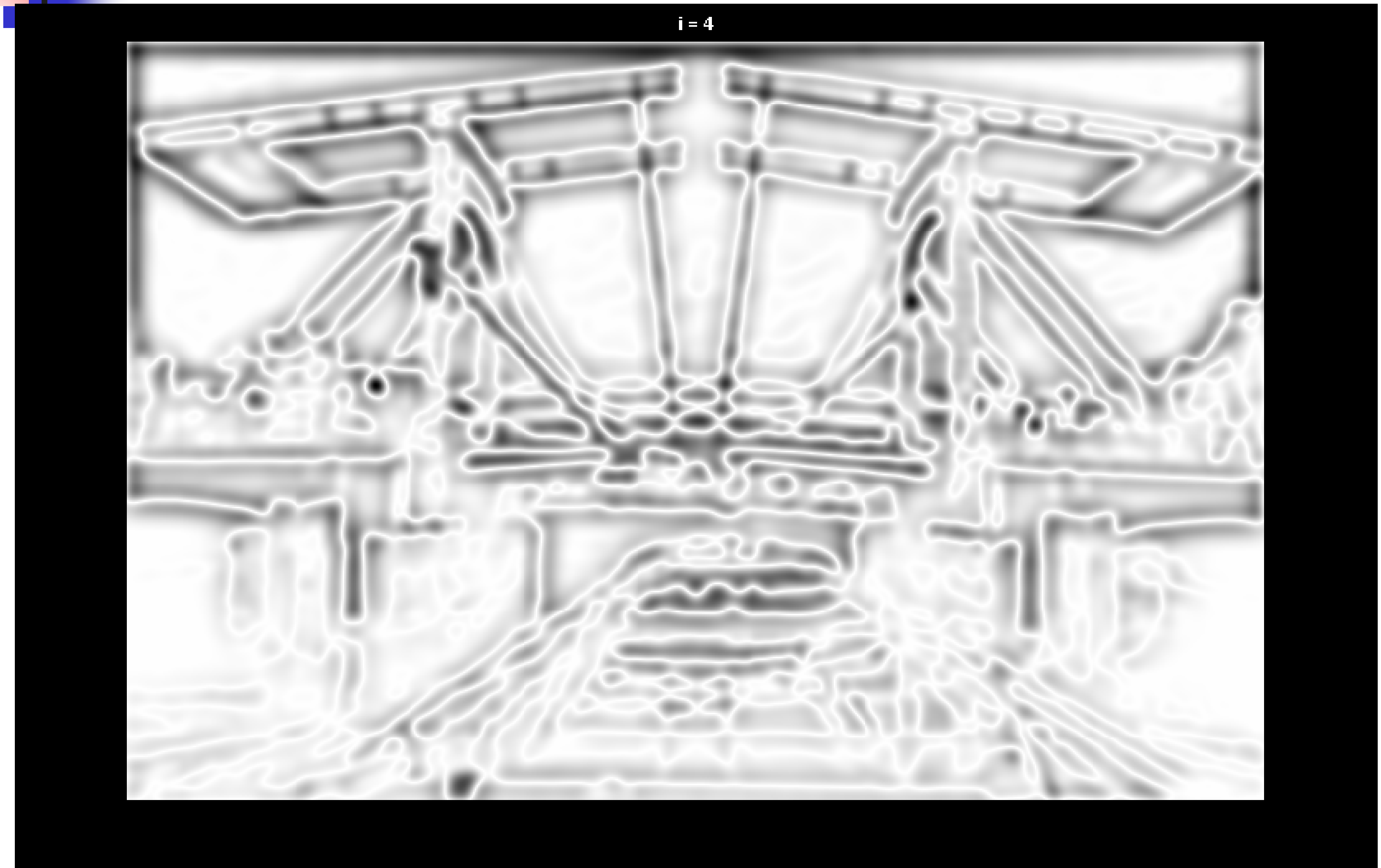
Gaussian Kernel Size i=2



Gaussian Kernel Size i=3



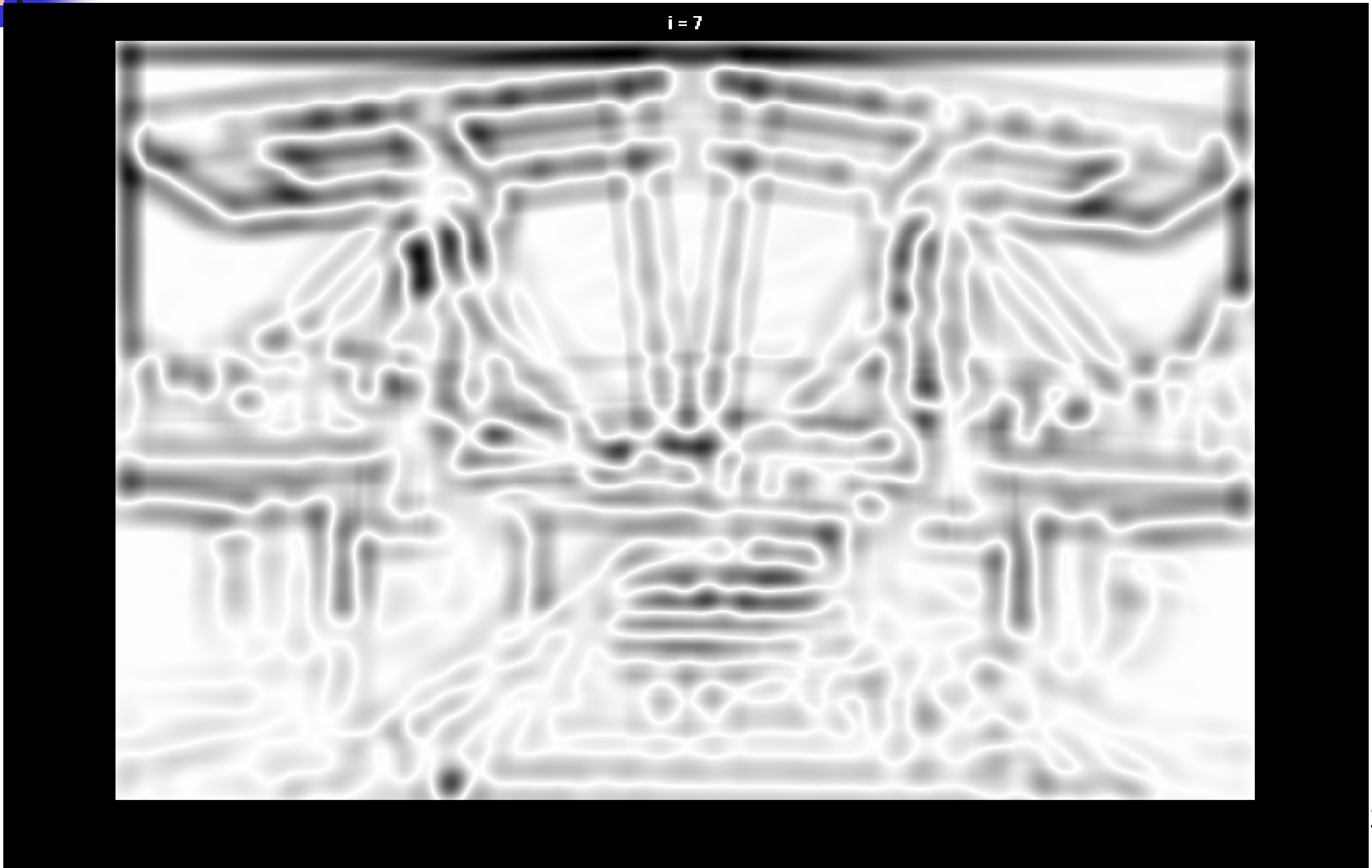
Gaussian Kernel Size i=4



Gaussian Kernel Size i=5

Gaussian Kernel Size i=6

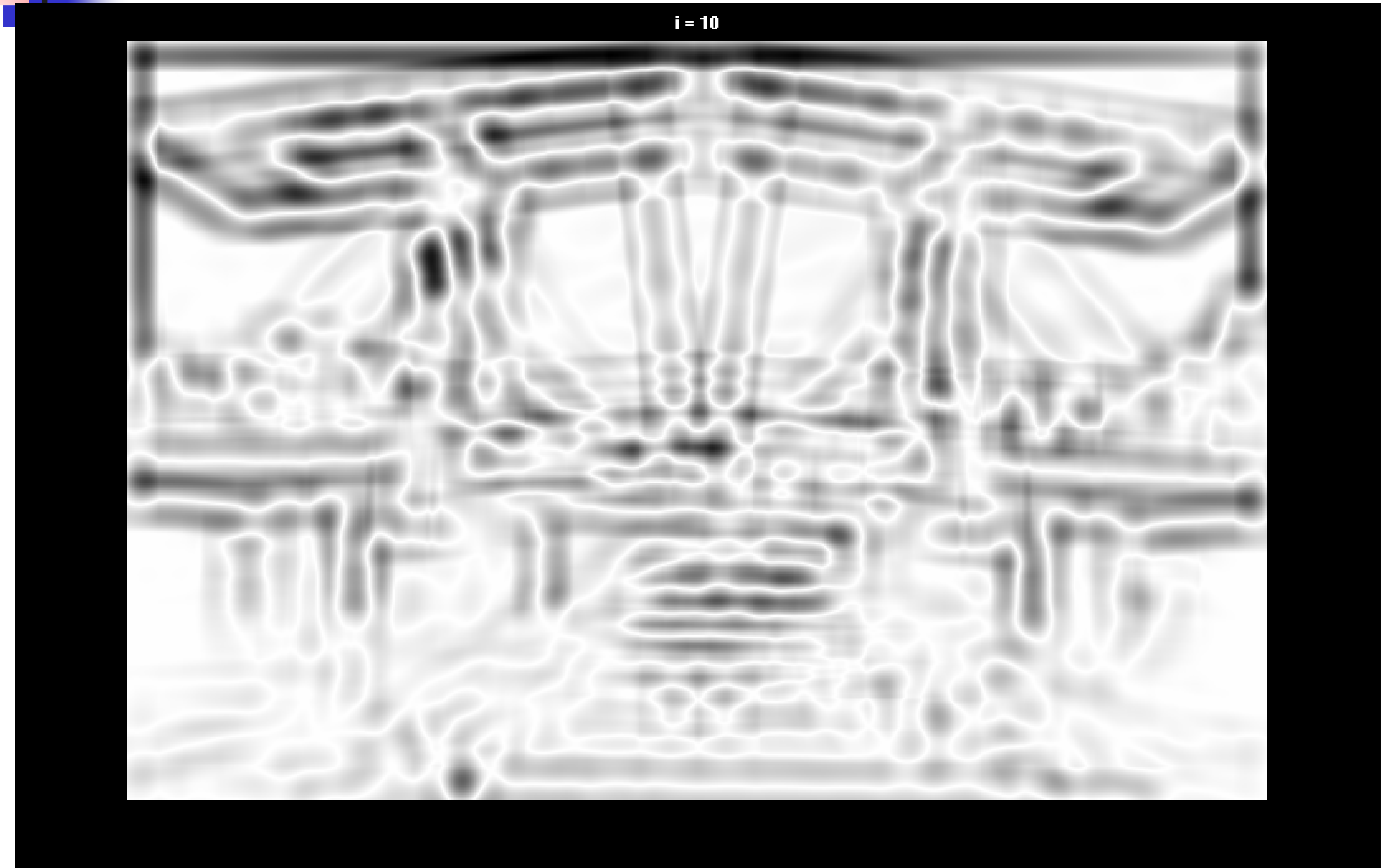
Gaussian Kernel Size i=7



Gaussian Kernel Size i=8

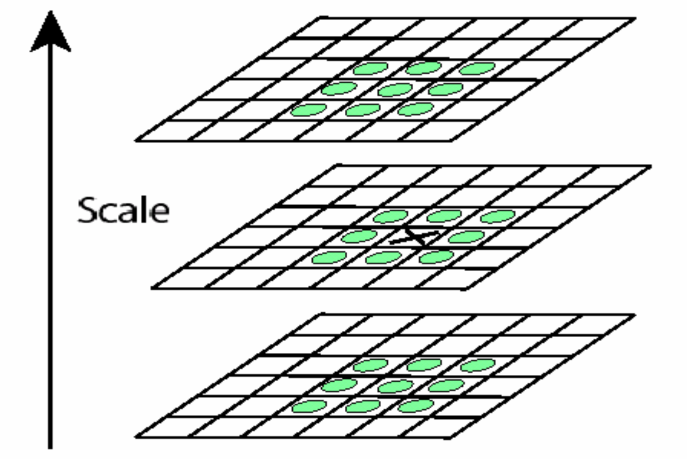
Gaussian Kernel Size i=9

Gaussian Kernel Size i=10

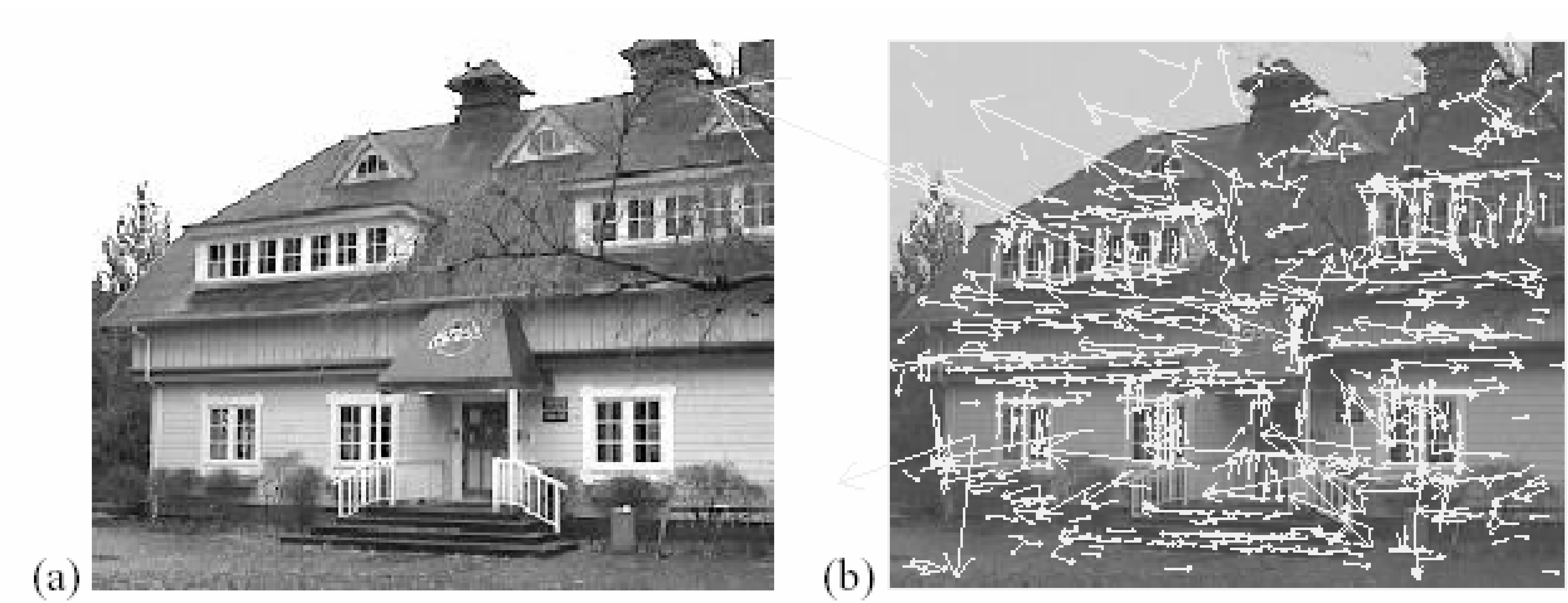


Key point localization

- Detect maxima and minima of difference-of-Gaussian in scale space



Example of keypoint detection



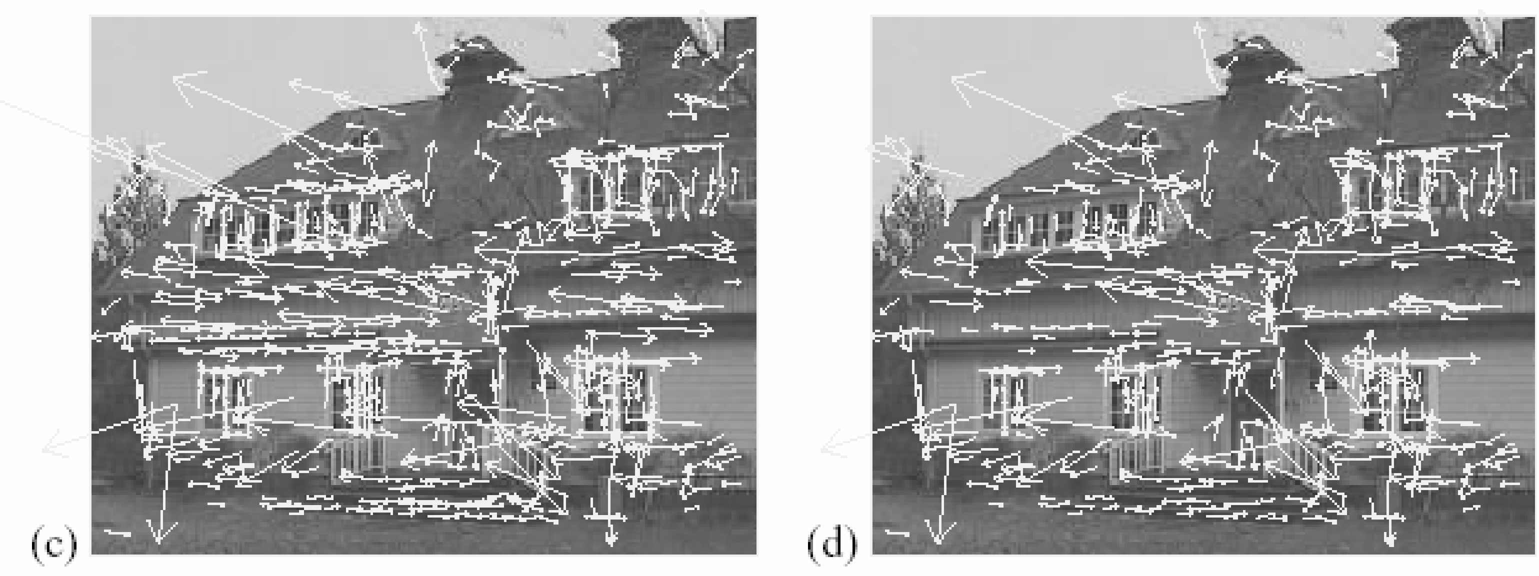
(a) 233x189 image
(b) 832 DOG extrema
(c) 729 above threshold

SIFT On-A-Slide

1. **Enforce invariance to scale:** Compute Gaussian difference max, for may different scales; non-maximum suppression, find local maxima: keypoint candidates
2. **Localizable corner:** For each maximum fit quadratic function. Compute center with sub-pixel accuracy by setting first derivative to zero.
3. **Eliminate edges:** Compute ratio of eigenvalues, drop keypoints for which this ratio is larger than a threshold.
4. **Enforce invariance to orientation:** Compute orientation, to achieve scale invariance, by finding the strongest second derivative direction in the smoothed image (possibly multiple orientations). Rotate patch so that orientation points up.
5. **Compute feature signature:** Compute a "gradient histogram" of the local image region in a 4x4 pixel region. Do this for 4x4 regions of that size. Orient so that largest gradient points up (possibly multiple solutions). Result: feature vector with 128 values (15 fields, 8 gradients).
6. **Enforce invariance to illumination change and camera saturation:** Normalize to unit length to increase invariance to illumination. Then threshold all gradients, to become invariant to camera saturation.

Example of keypoint detection

Threshold on value at DOG peak and on ratio of principle curvatures (Harris approach)



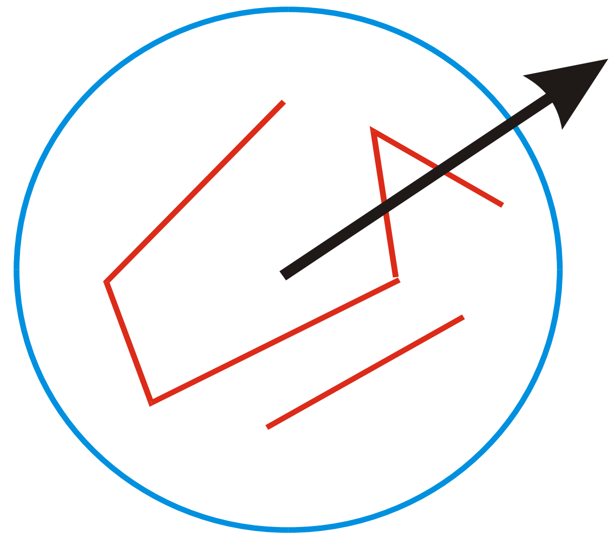
(c) 729 left after peak value threshold (from 832)
(d) 536 left after testing ratio of principle curvatures

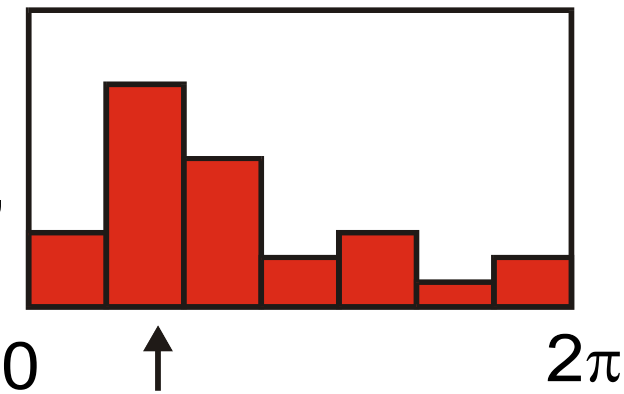
SIFT On-A-Slide

1. **Enforce invariance to scale:** Compute Gaussian difference max, for may different scales; non-maximum suppression, find local maxima: keypoint candidates
2. **Localizable corner:** For each maximum fit quadratic function. Compute center with sub-pixel accuracy by setting first derivative to zero.
3. **Eliminate edges:** Compute ratio of eigenvalues, drop keypoints for which this ratio is larger than a threshold.
4. **Enforce invariance to orientation:** Compute orientation, to achieve scale invariance, by finding the strongest second derivative direction in the smoothed image (possibly multiple orientations). Rotate patch so that orientation points up.
5. **Compute feature signature:** Compute a "gradient histogram" of the local image region in a 4x4 pixel region. Do this for 4x4 regions of that size. Orient so that largest gradient points up (possibly multiple solutions). Result: feature vector with 128 values (15 fields, 8 gradients).
6. **Enforce invariance to illumination change and camera saturation:** Normalize to unit length to increase invariance to illumination. Then threshold all gradients, to become invariant to camera saturation.

Select canonical orientation

- Create histogram of local gradient directions computed at selected scale
- Assign canonical orientation at peak of smoothed histogram
- Each key specifies stable 2D coordinates (x, y, scale, orientation)



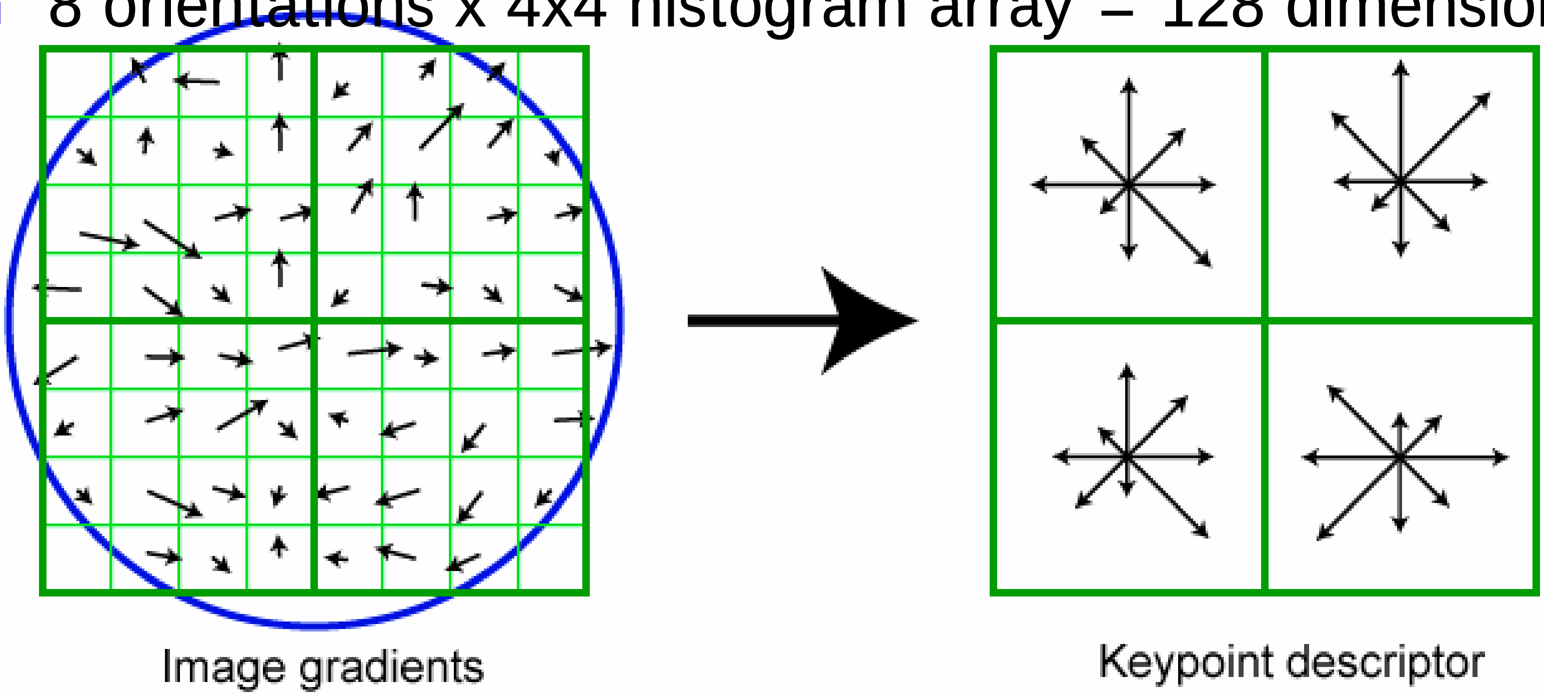


SIFT On-A-Slide

1. **Enforce invariance to scale:** Compute Gaussian difference max, for may different scales; non-maximum suppression, find local maxima: keypoint candidates
2. **Localizable corner:** For each maximum fit quadratic function. Compute center with sub-pixel accuracy by setting first derivative to zero.
3. **Eliminate edges:** Compute ratio of eigenvalues, drop keypoints for which this ratio is larger than a threshold.
4. **Enforce invariance to orientation:** Compute orientation, to achieve scale invariance, by finding the strongest second derivative direction in the smoothed image (possibly multiple orientations). Rotate patch so that orientation points up.
5. **Compute feature signature:** Compute a "gradient histogram" of the local image region in a 4x4 pixel region. Do this for 4x4 regions of that size. Orient so that largest gradient points up (possibly multiple solutions). Result: feature vector with 128 values (15 fields, 8 gradients).
6. **Enforce invariance to illumination change and camera saturation:** Normalize to unit length to increase invariance to illumination. Then threshold all gradients, to become invariant to camera saturation.

SIFT vector formation

- Thresholded image gradients are sampled over 16x16 array of locations in scale space
- Create array of orientation histograms
- 8 orientations x 4x4 histogram array = 128 dimensions

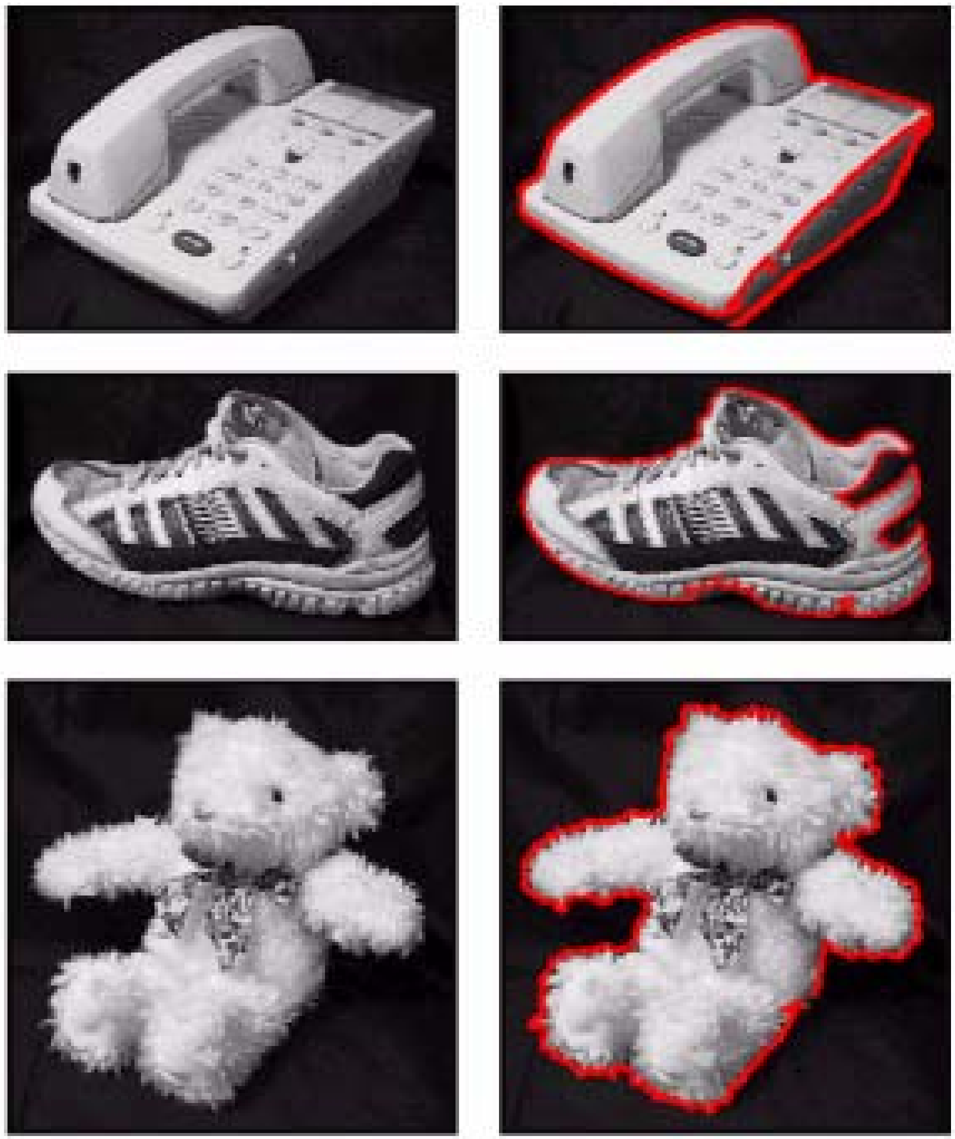


Nearest-neighbor matching to feature database

- Hypotheses are generated by **approximate nearest neighbor** matching of each feature to vectors in the database
  - SIFT use best-bin-first (Beis & Lowe, 97) modification to k-d tree algorithm
  - Use heap data structure to identify bins in order by their distance from query point

- **Result:** Can give speedup by factor of 1000 while finding nearest neighbor (of interest) 95% of the time

3D Object Recognition

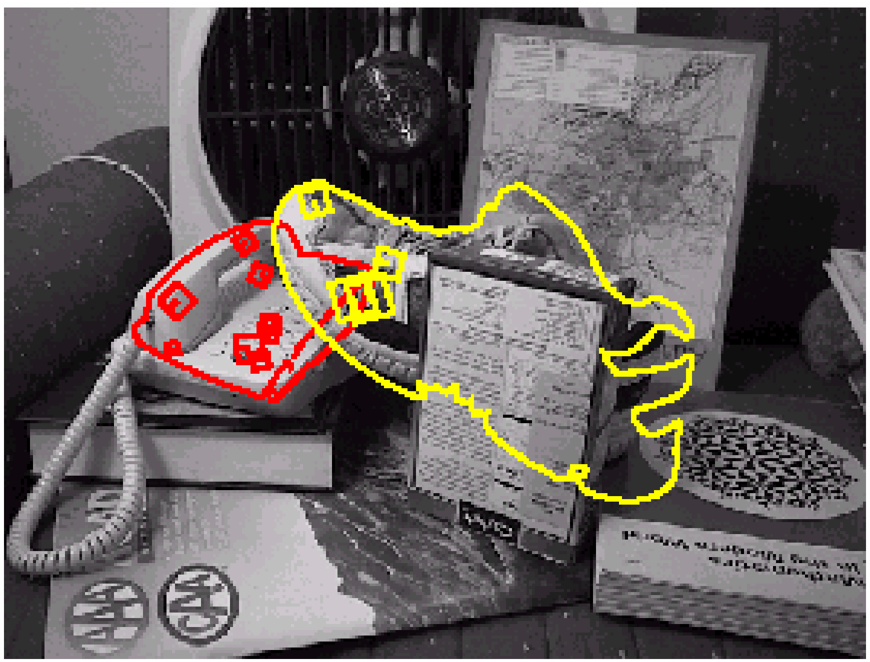
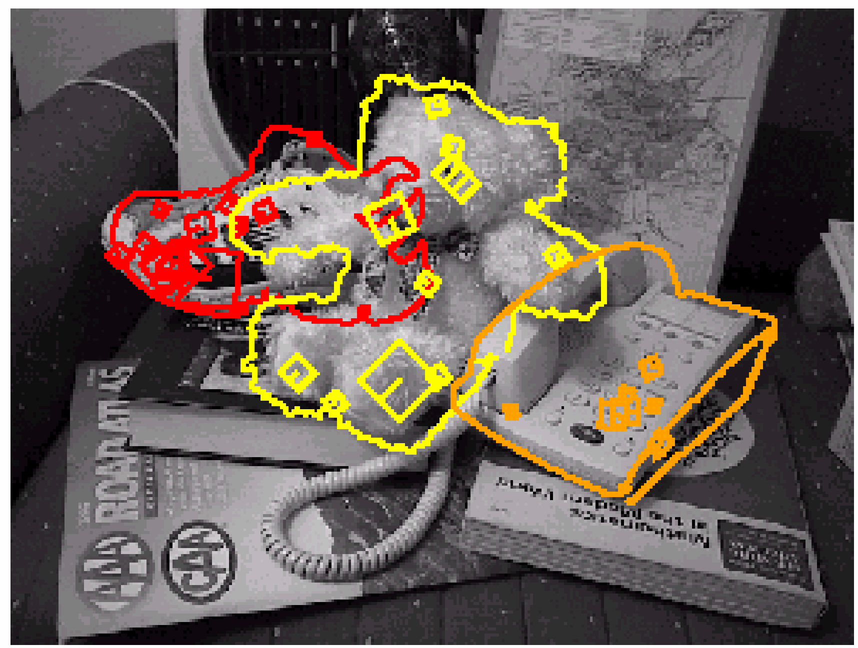


- Extract outlines with background subtraction

3D Object Recognition

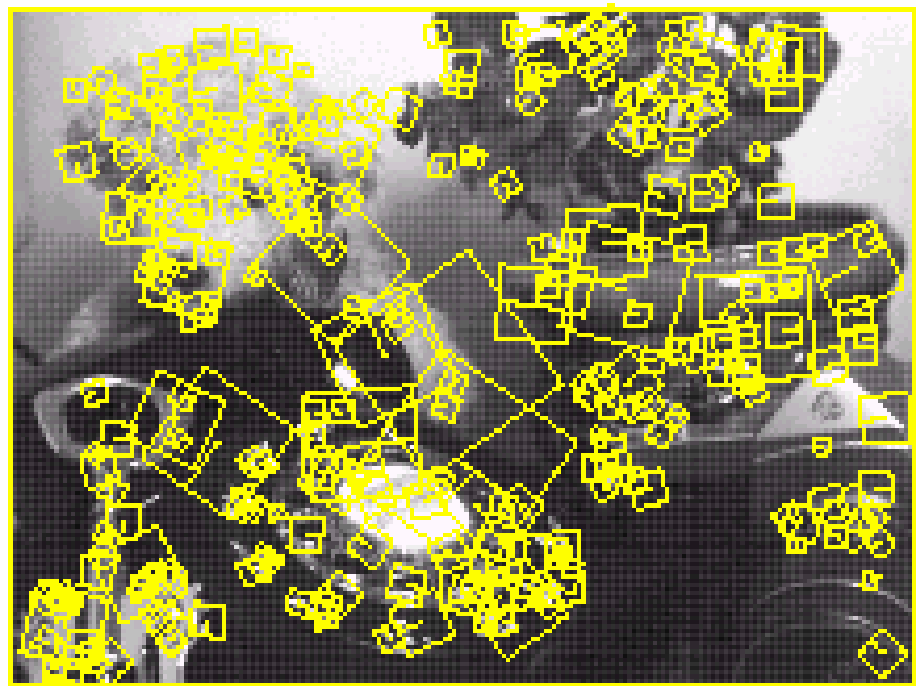
- Only 3 keys are needed for recognition, so extra keys provide robustness
- Affine model is no longer as accurate

Recognition under occlusion

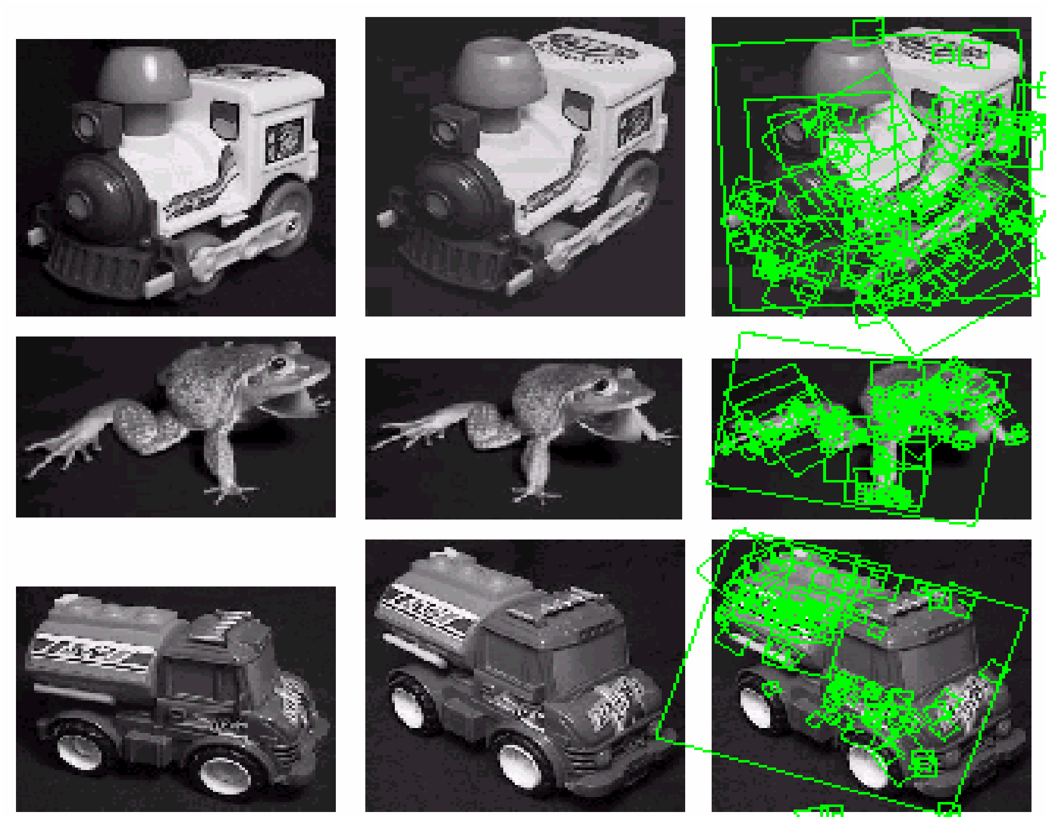
Test of illumination invariance

- Same image under differing illumination

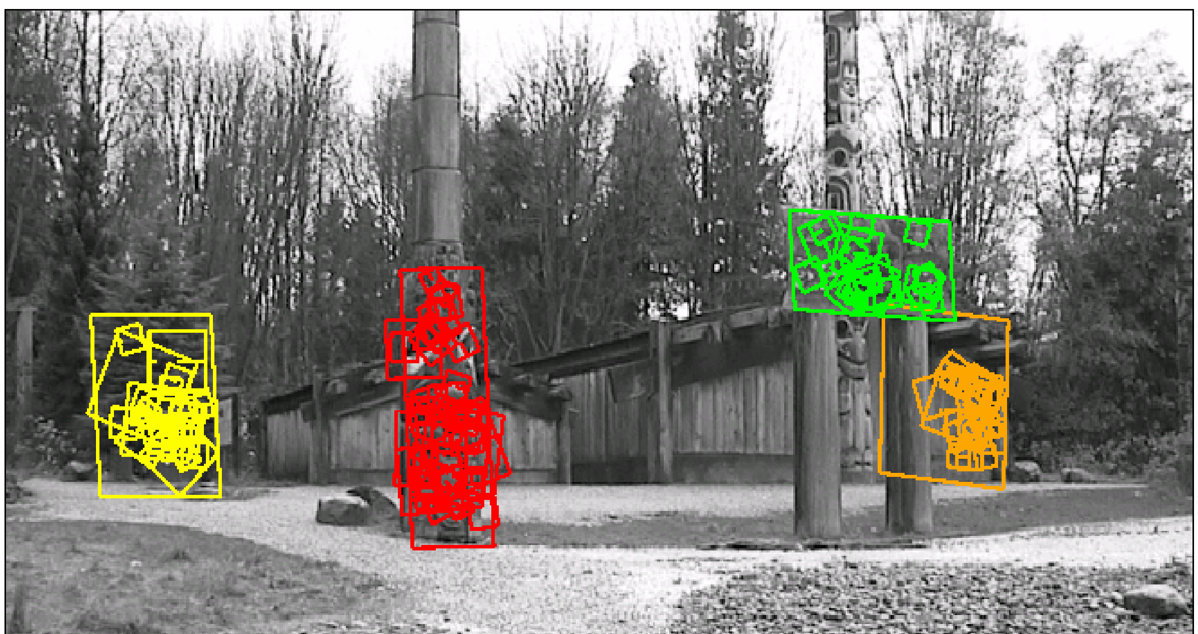


273 keys verified in final match

Examples of view interpolation



Location recognition



SIFT

- Invariances:
  - Scaling Yes
  - Rotation Yes
  - Illumination Yes
  - Deformation Maybe
- Provides
  - Good localization Yes