CS4442/9542b: Artificial Intelligence II

Prof. Olga Veksler

Lecture 13: Computer Vision

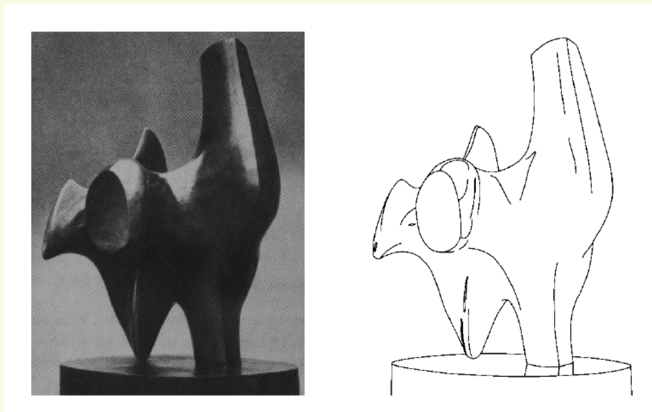
Edge Detection

Slides are from Steve Seitz (UW), David Jacobs (UMD), D. Lowe (UBC), Hong Man

Outline

- Edge Detection
  - Edge types
  - Image Gradient
  - Canny Edge Detector

Edge detection

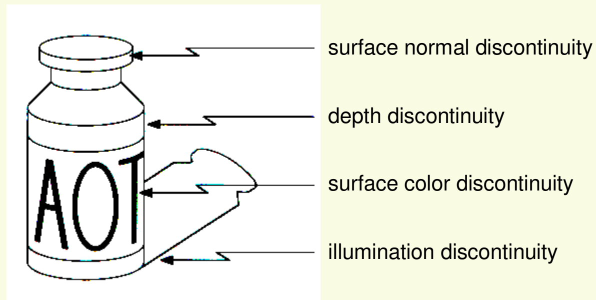


- Convert a 2D image into a set of “prominent” curves
  - What is a “prominent” curve or an edge? No exact definition. Intuitively, it’s a place where abrupt changes occur
- Why?
  - Extracts salient features of the scene, useful for may applications
  - More compact representation than pixels

Edge detection

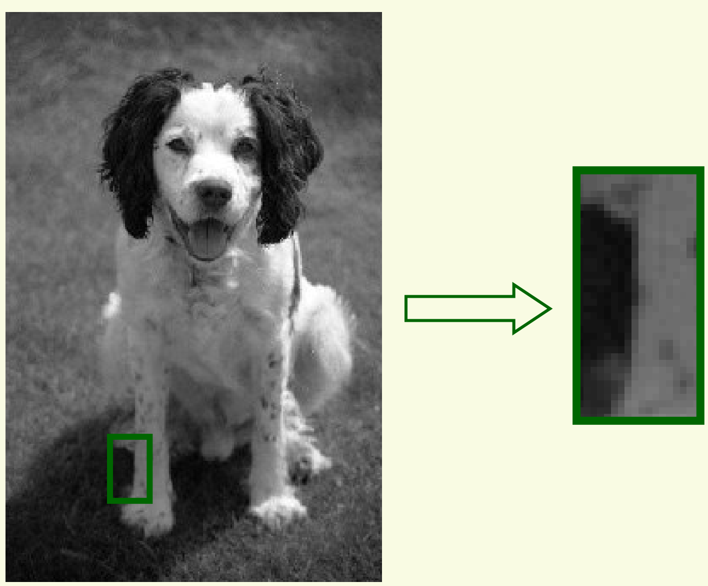
- Artists also do it
  - They do it much better, they have high level knowledge which edges are more perceptually important

Origin of Edges

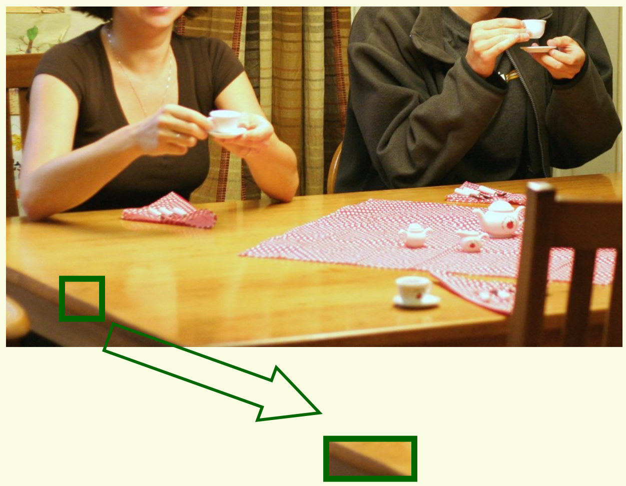


- Edges are caused by a variety of factors

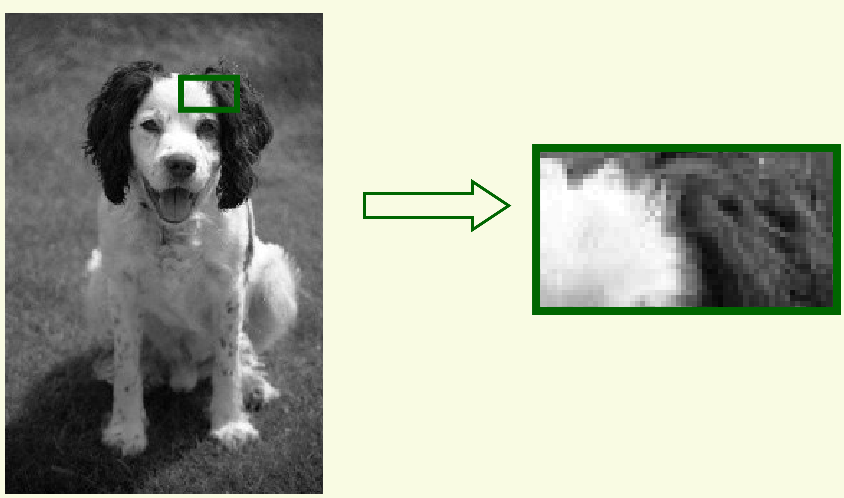
Depth Discontinuity



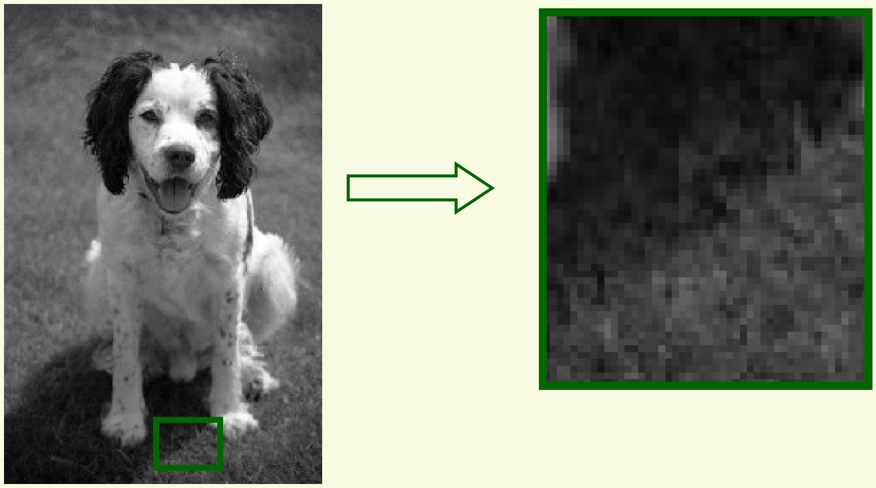
Surface Normal Discontinuity



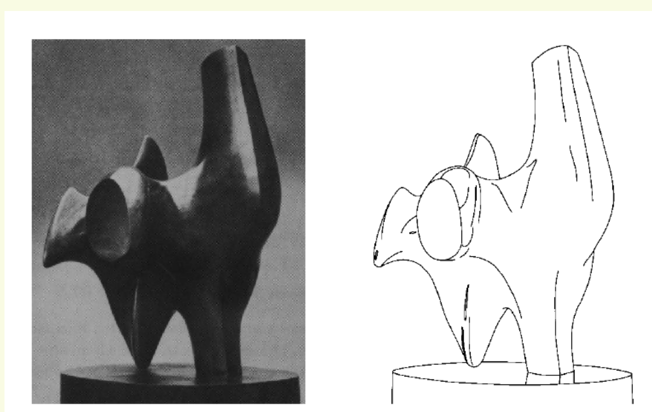
Surface Color Discontinuity



Illumination Discontinuity



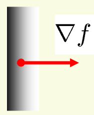
Edge detection



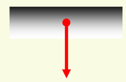
- How can you tell that a pixel is on an edge?

Image gradient

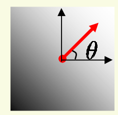
- The gradient of an image: $\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$



$\nabla f = \left[\frac{\partial f}{\partial x}, 0\right]$



$\nabla f = \left[0, \frac{\partial f}{\partial y}\right]$



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

- The gradient points in the direction of most rapid increase in intensity
- The gradient direction is given by:

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

  - gradient direction is perpendicular to edge

- The *edge strength* is given by the gradient magnitude

The discrete gradient

- How can we differentiate a *digital* image f(x,y)?
  - take discrete derivative (finite difference)

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

- How would you implement this as a convolution?

	-1	1

H

- Similarly, $\dfrac{\partial f(x,y)}{\partial y} = f(x,y+1) - f(x,y)$

	-1	
	1	

H

The discrete gradient

- The discrete gradient simply detects changes between neighboring pixels

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

change in vertical direction

change in horizontal direction

- Basic edge detection algorithm:

image $f(x,y)$ → Gradient operator → $g(x,y)$ → Thresholding → Edge map $E(x,y)$

$$E(x,y) = \begin{cases} 1 & |g(x,y)| > threshold \\ 0 & otherwise \end{cases}$$

The Sobel operator

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

$$s_x = \frac{1}{8}\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad s_y = \frac{1}{8}\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

  - The standard definition 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, however

Effects of Noise

too many pixels are detected as edges due to noise

Effects of Noise

- How do we deal with noise?
- We already know, filter the noise out
  - Using Gaussian kernel, for example
- First convolve image with a Gaussian filter
- Then convolve image with an edge detection filter (Sobel, for example)

Derivative theorem of convolution

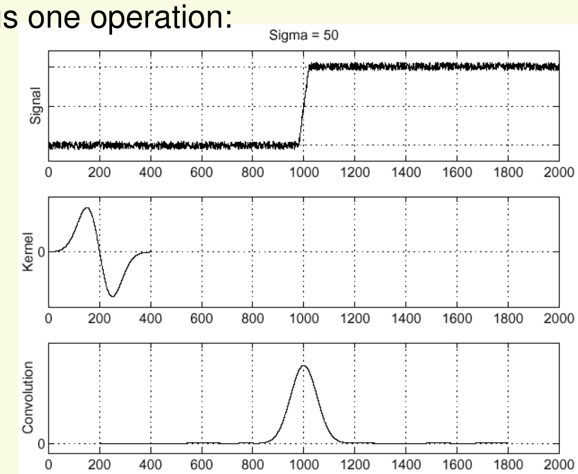
$$\frac{\partial}{\partial x}(H * f) = \left(\frac{\partial}{\partial x} H\right) * f$$

- This saves us one operation:

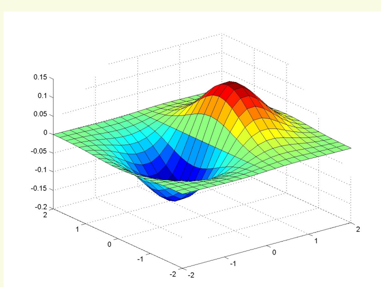
f

$\frac{\partial}{\partial x} H$

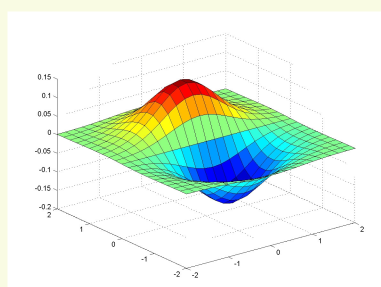
$\left(\frac{\partial}{\partial x} H\right) * f$



Derivative of Gaussian



$$\frac{\partial}{\partial x} G_\sigma$$

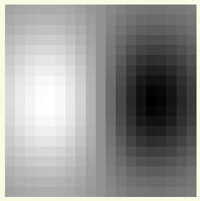


$$\frac{\partial}{\partial y} G_\sigma$$

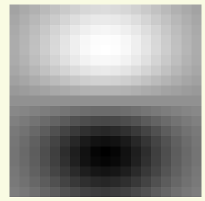
Gradient magnitude is computed from these

Slide credit: Christopher Rasmussen

Derivative of Gaussian



$$\frac{\partial}{\partial x} G_\sigma$$



$$\frac{\partial}{\partial y} G_\sigma$$

Bright corresponds to positive values, dark to negative values

Derivative of Gaussian: Example

- Ignoring normalizing constant: $G_\sigma(x,y) = e^{-\frac{(x^2+y^2)}{2\sigma^2}}$
- Differentiate with respect to x and y

$$\frac{\partial}{\partial x} G_\sigma(x,y) = -\frac{x}{\sigma^2} \cdot e^{-\frac{(x^2+y^2)}{2\sigma^2}} \qquad \frac{\partial}{\partial y} G_\sigma(x,y) = -\frac{y}{\sigma^2} \cdot e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

- Plug some values to get gradient detection masks $\mathbf{H_x}$ and $\mathbf{H_y}$
  - for example, let σ = 5, and let (x,y) be in [-2x2][-2x2] window

(-2,2)	(-1,2)	(0,2)	(1,2)	(2,2)
(-2,1)	(-1,1)	(0,1)	(1,1)	(2,1)
(-2,0)	(-1,0)	(0,0)	(1,0)	(2,0)
(-2,-1)	(-1,-1)	(0,-1)	(1,-1)	(2,-1)
(-2,-2)	(-1,-2)	(0,-2)	(1,-2)	(2,-2)

$\mathbf{H_x}$

0.04	0.08	0	-0.08	-0.04
0.16	0.37	0	-0.37	-0.16
0.27	0.61	0	-0.61	-0.27
0.16	0.37	0	-0.37	-0.16
0.04	0.08	0	-0.08	-0.04

$\mathbf{H_y}$

-0.04	-0.04	-0.04	-0.04	-0.04
-0.08	-0.08	-0.08	-0.08	-0.08
0	0	0	0	0
0.08	0.08	0.08	0.08	0.08
0.04	0.04	0.04	0.04	0.04

Derivative of Gaussian: Example

H_x

0.04	0.08	0	-0.08	-0.04
0.16	0.37	0	-0.37	-0.16
0.27	0.61	0	-0.61	-0.27
0.16	0.37	0	-0.37	-0.16
0.04	0.08	0	-0.08	-0.04

H_y

-0.04	-0.04	-0.04	-0.04	-0.04
-0.08	-0.08	-0.08	-0.08	-0.08
0	0	0	0	0
0.08	0.08	0.08	0.08	0.08
0.04	0.04	0.04	0.04	0.04

121	121	122	123	122	123
121	121	122	123	122	123
122	123	124	123	124	123
120	122	122	123	122	123
121	121	124	123	124	123
125	120	124	123	124	123

apply H_x to the red image pixel: -0.78
aply H_y to the red image pixel: 0.46

121	121	122	123	20	20
121	121	122	123	22	22
122	123	124	123	24	21
120	122	122	123	22	22
121	121	124	123	24	23
125	120	124	123	24	24

apply H_x to the red image pixel: **217**
aply H_y to the red image pixel: 0.69

Derivative of Gaussian: Example

H_x

0.04	0.08	0	-0.08	-0.04
0.16	0.37	0	-0.37	-0.16
0.27	0.61	0	-0.61	-0.27
0.16	0.37	0	-0.37	-0.16
0.04	0.08	0	-0.08	-0.04

H_y

-0.04	-0.04	-0.04	-0.04	-0.04
-0.08	-0.08	-0.08	-0.08	-0.08
0	0	0	0	0
0.08	0.08	0.08	0.08	0.08
0.04	0.04	0.04	0.04	0.04

121	121	122	123	20	20
121	121	122	123	22	22
122	123	124	123	24	21
120	122	122	123	22	22
121	121	124	123	24	23
125	120	124	123	24	24

apply H_x to the red image pixel: **217**
aply H_y to the red image pixel: 0.69

121	121	122	120	121	125
121	121	123	122	121	120
122	122	124	122	124	124
123	123	123	123	123	123
20	22	24	22	24	24
20	22	21	22	23	24

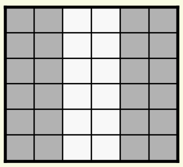
apply H_x to the red image pixel: -0.69
aply H_y to the red image pixel: -217

Notice a mask looks like a pattern it is trying to detect!

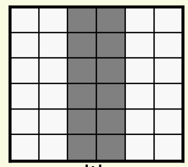
What does this Mask Detects?

H

2	2	-4	-4	2	2
2	2	-4	-4	2	2
2	2	-4	-4	2	2
2	2	-4	-4	2	2
2	2	-4	-4	2	2



strong negative response

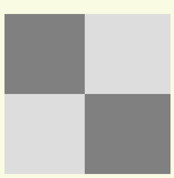


strong positive response

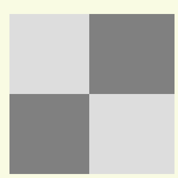
What Does this Mask Detects?

H

2	2	-2	-2
2	2	-2	-2
-2	-2	2	2
-2	-2	2	2



strong negative response
in the middle



strong positive response
in the middle

The Canny edge detector

- original image (Lena)

The Canny edge detector

- norm of the gradient

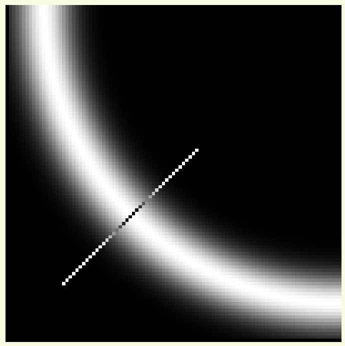
The Canny edge detector

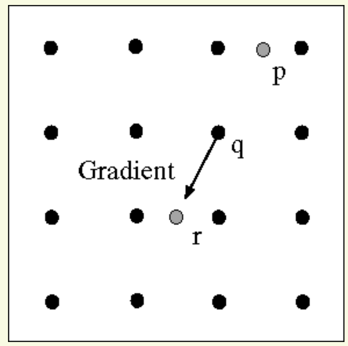
- thresholding

The Canny edge detector

- thinning
- (non-maximum suppression)

Non-maximum suppression

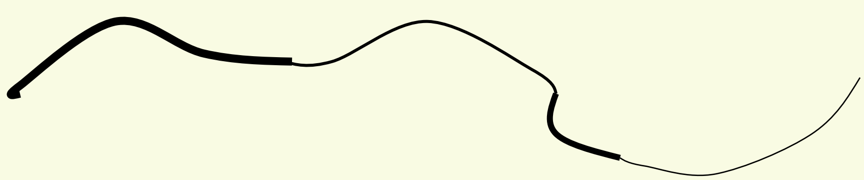




- Check if pixel is local maximum along gradient direction
  - requires checking interpolated pixels p and r

Hysteresis

- Strong Edges reinforce adjacent weak edges
- 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.



Effect of σ (Gaussian kernel spread/size)

original | Canny with $\sigma = 1$ | Canny with $\sigma = 2$

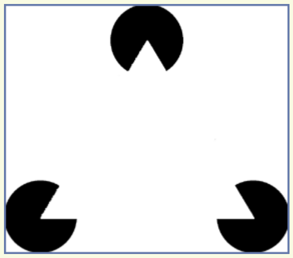
- The choice of σ depends on desired behavior
  - large σ detects large scale edges
  - small σ detects fine features

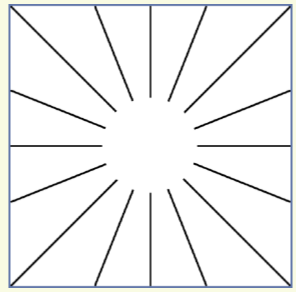
Why is Canny so Dominant

- Still widely used after 20 years.

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

Illusory Contours





- Triangle and circle floating in front of background
- Not possible to detect the "illusory" contours using local edge detection