Medical Image Processing

5 Filtering and Edge Detection

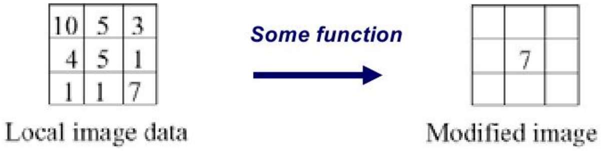
Prof. Dr. Marcin Grzegorzek

Research Group for Pattern Recognition
Institute for Vision and Graphics
University of Siegen, Germany

Table of Contents

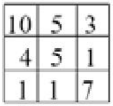
Image Filtering

- **Modify the pixels in an image based on some function of a local neighborhood of the pixels.**



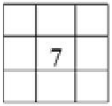
Linear Functions

- Simplest: linear filtering.
  - Replace each pixel by a linear combination of its neighbors.
- The prescription for the linear combination is called the “convolution kernel”.



10	5	3
4	5	1
1	1	7

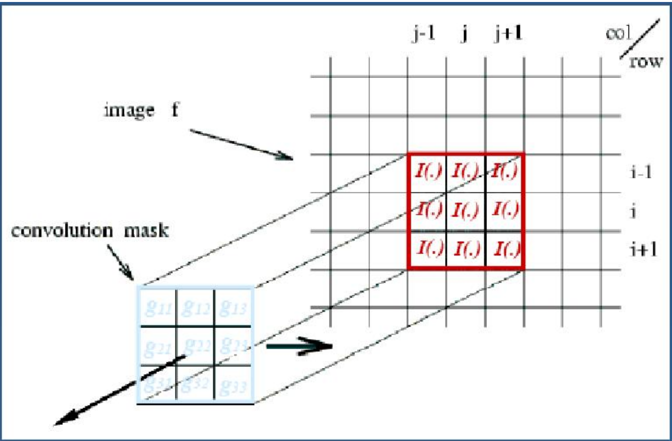
Local image data



	7	

Modified image data

Mask



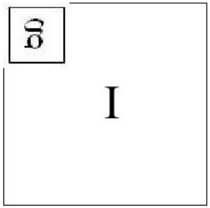
j-1
j
j+1
col
row
image f
i-1
i
i+1
convolution mask
I(.)
g_{11} g_{12} g_{13}
g_{21} g_{22} g_{23}
g_{31} g_{32} g_{33}

$$f(i,j) = g_{11}\, I(i-1,j-1) + g_{12}\, I(i-1,j) + g_{13}\, I(i-1,j+1) + g_{21}\, I(i,j-1) + g_{22}\, I(i,j) + g_{23}\, I(i,j+1) + g_{31}\, I(i+1,j-1) + g_{32}\, I(i+1,j) + g_{33}\, I(i+1,j+1)$$

Convolution

- Let **I** be the image and **g** be the kernel. The output of convolving **I** with **g** is denoted $\mathbf{I} * g$

$$\mathbf{f}[m,n] = \mathbf{I} * \mathbf{g} = \sum_{k,l} \mathbf{I}[m-k, n-l]\mathbf{g}[k,l]$$



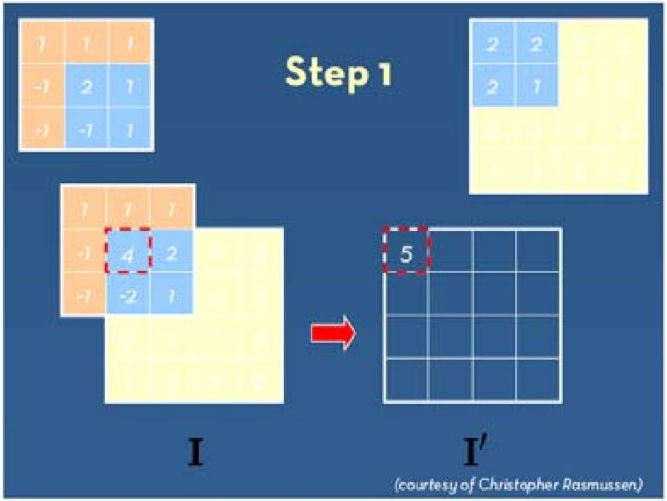
Key Properties

- **Linearity:**
  - filter($I_1 + I_2$) = filter(I_1) + filter(I_2)
- **Shift invariance:**
  - same behavior regardless of pixel location
  - filter(shift(I)) = shift(filter(I))
- Theoretical result:
  - ***Any linear shift-invariant operator can be represented as a convolution***

Key Properties

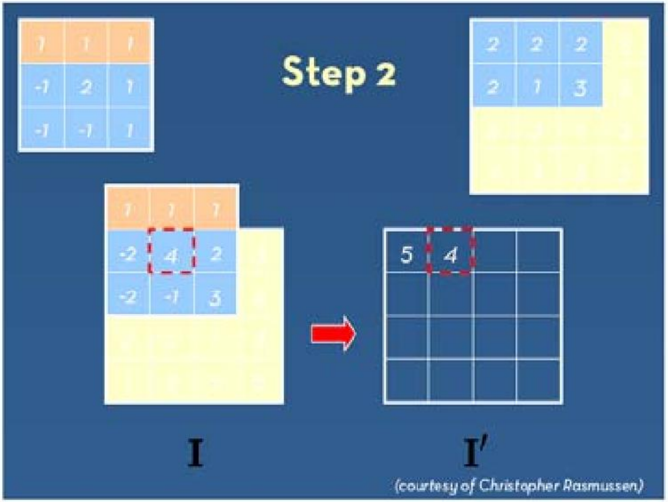
- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and signal
- Associative: $a * (b * c) = (a * b) * c$
  - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - This is equivalent to applying one filter: $a * (b_1 * b_2 * b_3)$
- Distributes over addition: $a * (b + c) = (a * b) + (a * c)$
- Scalars factor out: $ka * b = a * kb = k\,(a * b)$
- Identity: unit impulse $e = [\ldots, 0, 0, 1, 0, 0, \ldots]$,
  $a * e = a$

Example



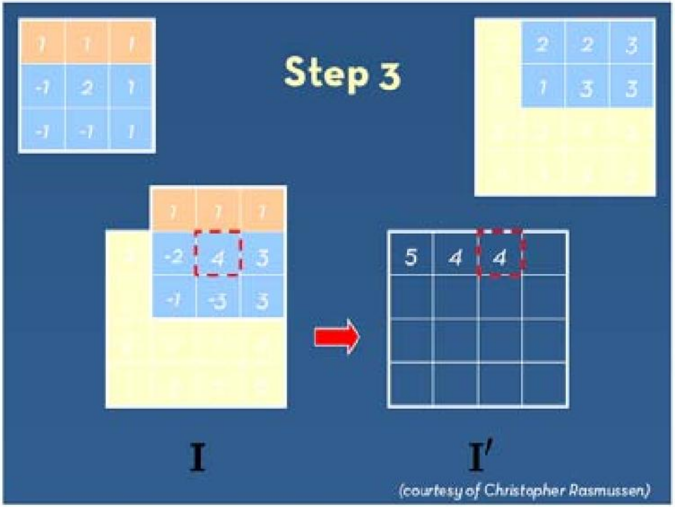
Step 1
1 1 1
-1 2 1
-1 -1 1
2 2
2 1
1 1 1
-1 4 2
-1 -2 1
5
I
I′
(courtesy of Christopher Rasmussen)

Example



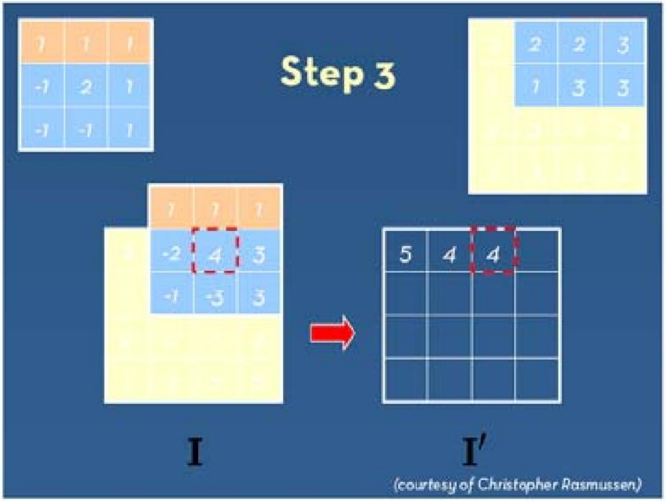
Step 2
1 1 1
-1 2 1
-1 -1 1
2 2 2
2 1 3
1 1 1
-2 4 2
-2 -1 3
5 4
I
I′
(courtesy of Christopher Rasmussen)

Example

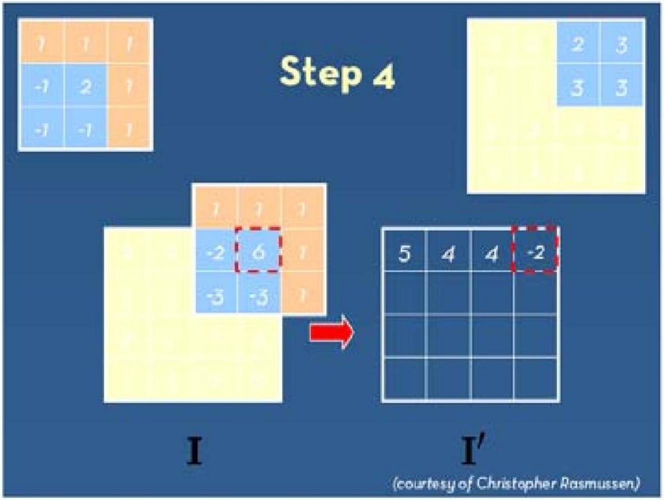


Step 3
1 1 1
-1 2 1
-1 -1 1
2 2 3
1 3 3
1 1 1
-2 4 3
-1 -3 3
5 4 4
I
I′
(courtesy of Christopher Rasmussen)

Example

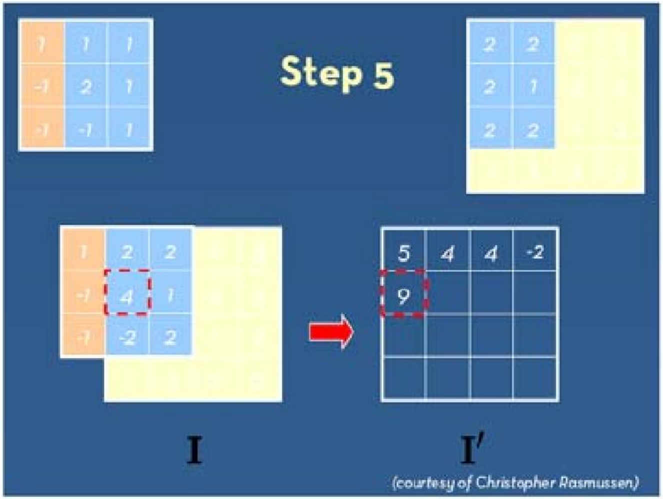


Example

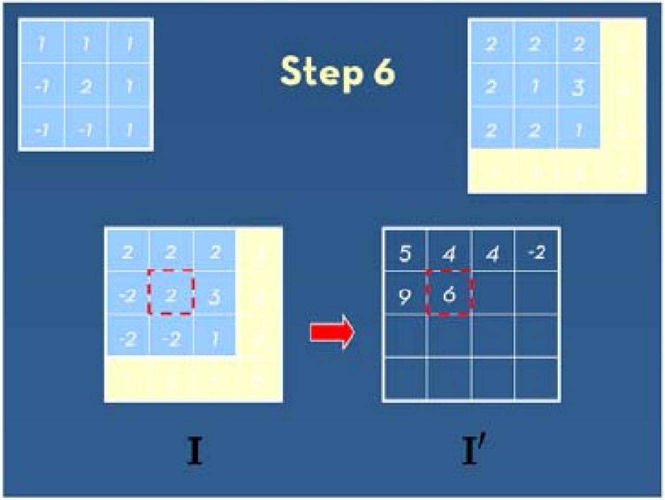


Step 4
1 1 1
-1 2 1
-1 -1 1
2 3
3 3
-2 6 1
-3 -3 1
5 4 4 -2
I
I′
(courtesy of Christopher Rasmussen)

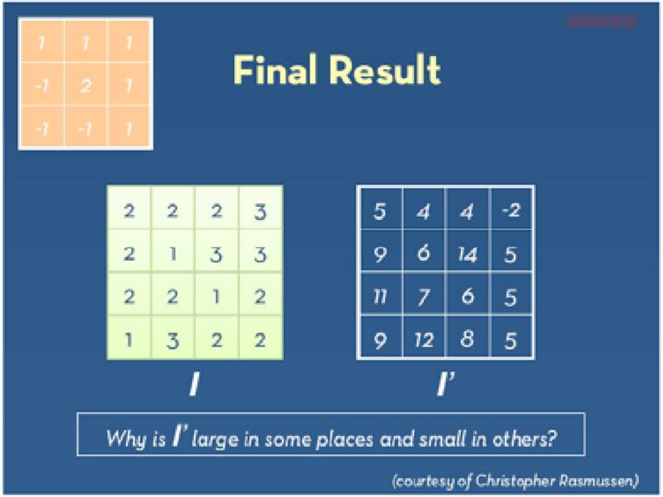
Example


Step 5
1
1
1
-1
2
1
-1
-1
1
2
2
2
1
2
2
1
2
2
-1
4
1
-1
-2
2
5
4
4
-2
9
I
I′
(courtesy of Christopher Rasmussen)

Example



Example



Final Result

1	1	1
-1	2	1
-1	-1	1

I

2	2	2	3
2	1	3	3
2	2	1	2
1	3	2	2

I'

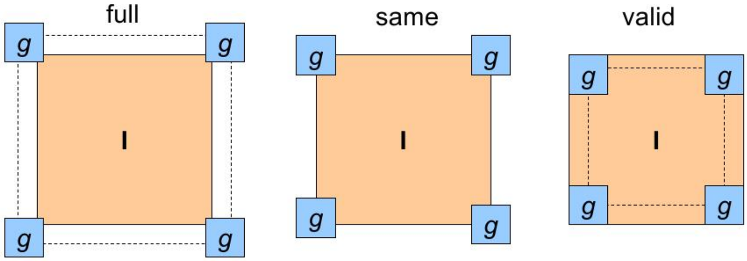
5	4	4	-2
9	6	14	5
11	7	6	5
9	12	8	5

*Why is **I'** large in some places and small in others?*

(courtesy of Christopher Rasmussen)

Some Details

- What is the size of the output?
- MATLAB: filter2(g, **I**, *shape*)
  - *shape* = 'full': output size is sum of sizes of **I** and g
  - *shape* = 'same': output size is same as **I**
  - *shape* = 'valid':output size is of difference sizes for **I** & g



Implementation

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge

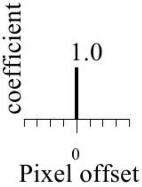
Implementation

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods (MATLAB):

clip filter (black):	imfilter(f, g, 0)
wrap around:	imfilter(f, g, ‘circular’)
copy edge:	imfilter(f, g, ‘replicate’)
reflect across edge:	imfilter(f, g, ‘symmetric’)

Linear Filtering

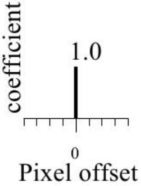
original



?

Linear Filtering

original

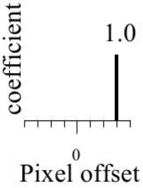


0	0	0
0	1	0
0	0	0

Filtered
(no change)

Linear Filtering

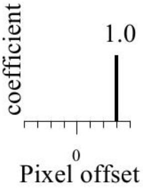
original

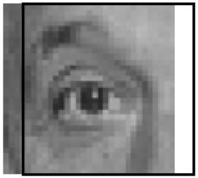


?

Shift

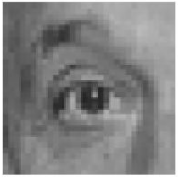
original

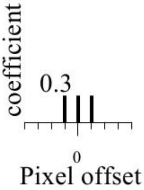



shifted

0	0	0
0	0	1
0	0	0

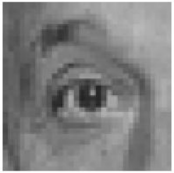
Linear Filtering


original

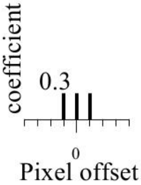


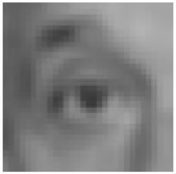
?

Blurring



original



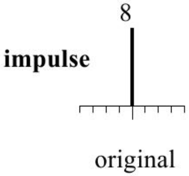


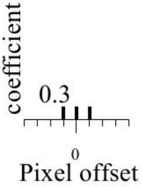
Blurred (filter applied in **both dimensions**).

Box filter:

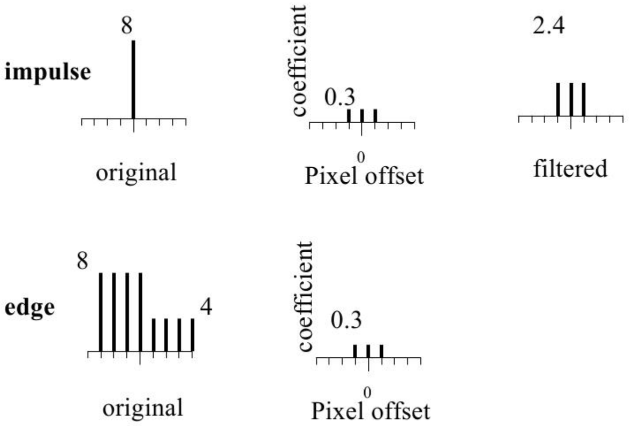
$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$

Blur Examples



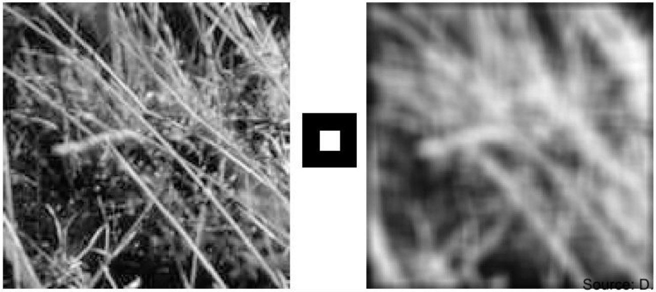


Blur Examples



Smoothing with Box Filter

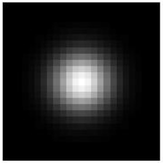
- Smoothing with an average actually doesn't compare at all well with a defocused lens
- Most obvious difference is that a single point of light viewed in a defocused lens looks like a fuzzy blob; but the averaging process would give a little square



Source: D. Forsyth

Smoothing with Box Filter

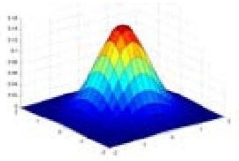
- Smoothing with an average actually doesn't compare at all well with a defocused lens
- Most obvious difference is that a single point of light viewed in a defocused lens looks like a fuzzy blob; but the averaging process would give a little square
- Better idea: to eliminate edge effects, weight contribution of neighborhood pixels according to their closeness to the center, like so:

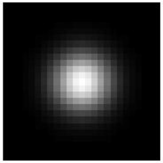


"fuzzy blob"

Gaussian Kernel

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





0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

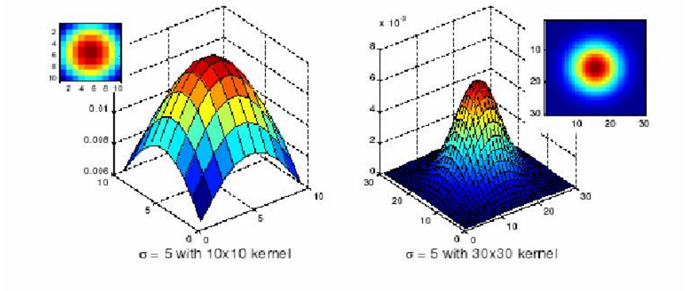
5 x 5, σ = 1

- Constant factor at front makes volume sum to 1 (can be ignored, as we should re-normalize weights to sum to 1 in any case)

Source: C. Rasmussen

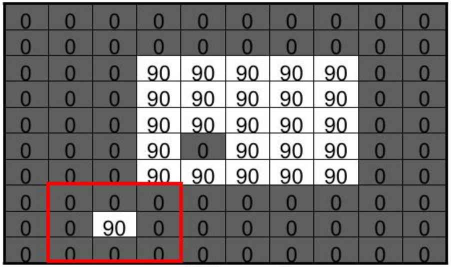
Gaussian Kernel

- Gaussian filters have infinite support, but discrete filters use finite kernels



Gaussian Filtering

- A Gaussian kernel gives less weight to pixels further from the center of the window



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

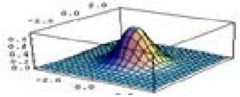
$F[x, y]$

$$\frac{1}{16}\boxed{\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline\end{array}}$$

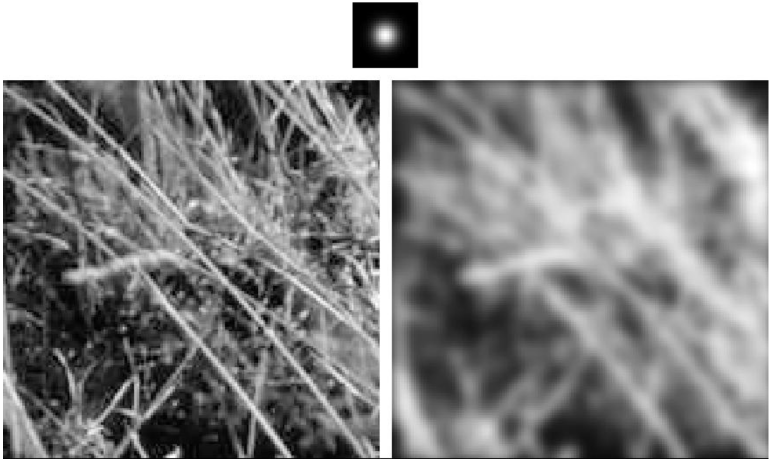
$H[u, v]$

- This kernel is an approximation of a Gaussian function:

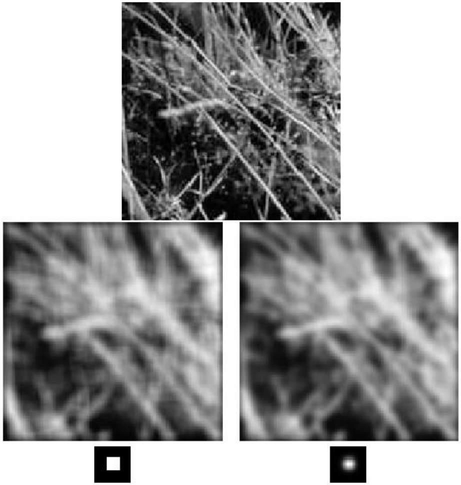
$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



Smoothing with Gaussian



Mean vs. Gaussian Filtering



Separability of the Gaussian Filter

$$\begin{aligned} G_\sigma(x,y) &= \frac{1}{2\pi\sigma^2}\exp^{-\frac{x^2+y^2}{2\sigma^2}} \\ &= \left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{x^2}{2\sigma^2}}\right)\left(\frac{1}{\sqrt{2\pi}\sigma}\exp^{-\frac{y^2}{2\sigma^2}}\right) \end{aligned}$$

The 2D Gaussian can be expressed as the product of two functions, one a function of x and the other a function of y

In this case, the two functions are the (identical) 1D Gaussian

Separability Example

2D convolution (center location only)

$$\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array}$$

The filter factors into a product of 1D filters:

$$\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array} = \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} \text{ x } \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array}$$

Perform convolution along rows:

$$\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array} * \begin{array}{|c|c|c|}\hline 2 & 3 & 3 \\ \hline 3 & 5 & 5 \\ \hline 4 & 4 & 6 \\ \hline \end{array} = \begin{array}{|c|c|c|}\hline & 11 & \\ \hline & 18 & \\ \hline & 18 & \\ \hline \end{array}$$

Followed by convolution along the remaining column:

Gaussian Filters

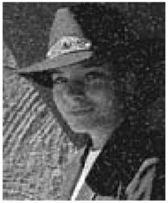
- Remove “high-frequency” components from the image (low-pass filter)
- Convolution with self is another Gaussian
  - So can smooth with small-width kernel, repeat, and get same result as larger-width kernel would have
  - Convolving two times with Gaussian kernel of width σ is same as convolving once with kernel of width sqrt(2) σ
- *Separable* kernel
  - Factors into product of two 1D Gaussians
  - Useful: can convolve all rows, then all columns
  - How does this change the computational complexity?
    - Linear vs. quadratic in mask size

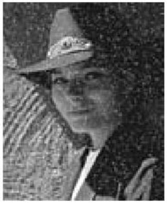
Review: Linear Filtering

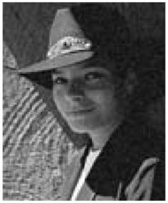
- What are the defining mathematical properties of a convolution?
- What is the difference between blurring with a box filter and blurring with a Gaussian?
- What happens when we convolve a Gaussian with another Gaussian?
- What is separability?
- How does separability affect computational complexity?

Noise

Original


Salt and pepper noise

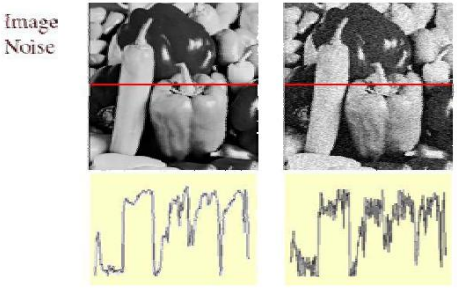

Impulse noise


Gaussian noise

- **Salt and pepper noise**: contains random occurrences of black and white pixels
- **Impulse noise:** contains random occurrences of white pixels
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Gaussian Noise

- Mathematical model: sum of many independent factors
- Good for small standard deviations
- Assumption: independent, zero-mean noise

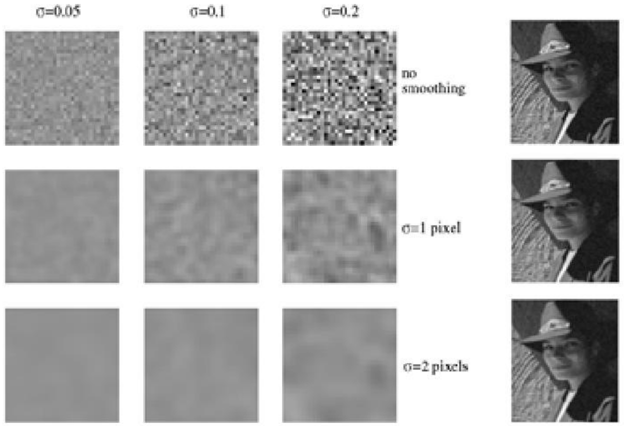


$f(x,y) = \overbrace{\tilde{f}(x,y)}^{\text{Ideal Image}} + \overbrace{\eta(x,y)}^{\text{Noise process}}$

Gaussian i.i.d. ("white") noise:
$\eta(x,y) \sim N(\mu,\sigma)$

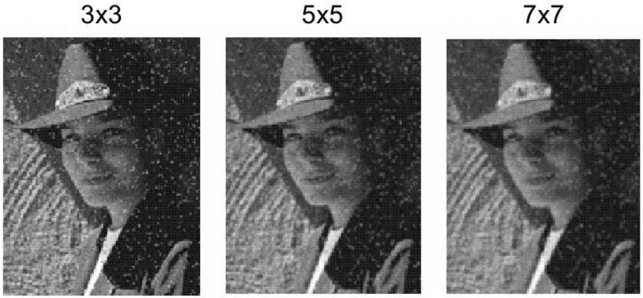
Source: K. Grauman

Reducing Gaussian Noise



Smoothing with larger standard deviations suppresses noise, but also blurs the image

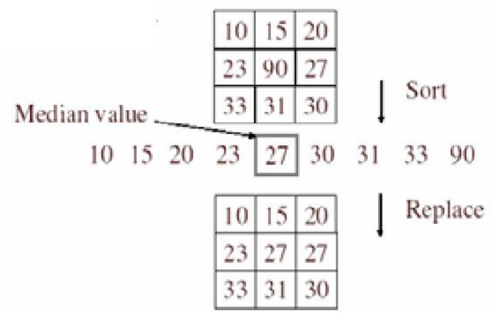
Reducing Salt-and-Pepper Noise



- What's wrong with the results?

Median Filtering

- A **median filter** operates over a window by selecting the median intensity in the window



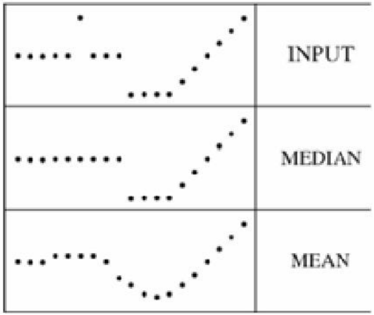
- Is median filtering linear?
  - No. ←→Not a convolution

Source: K. Grauman

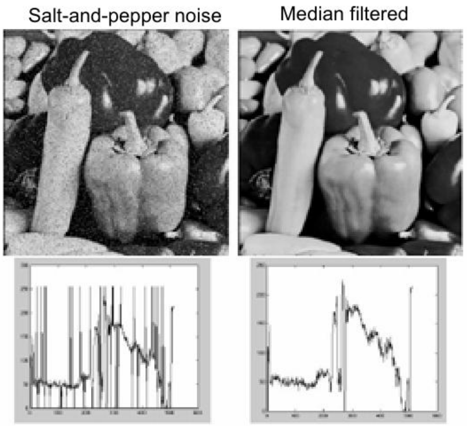
Median Filtering

- What advantage does median filtering have over Gaussian filtering?
  - Robustness to outliers

filters have width 5 :

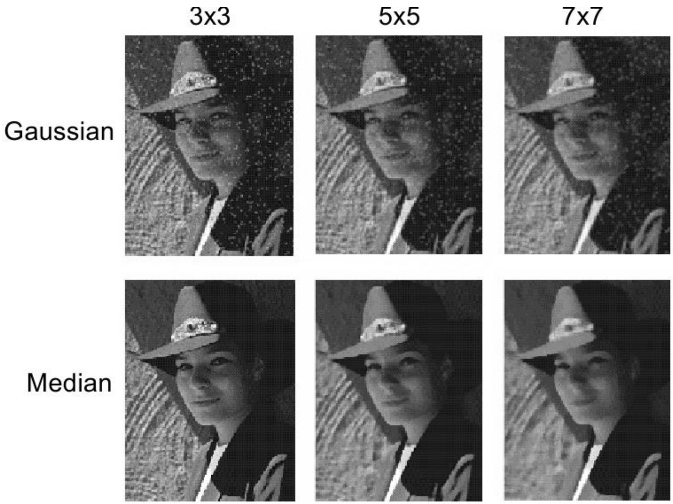


Median Filtering



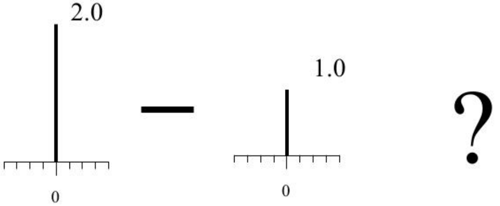
- MATLAB: medfilt2(image, [h w])

Median vs. Gaussian Filtering



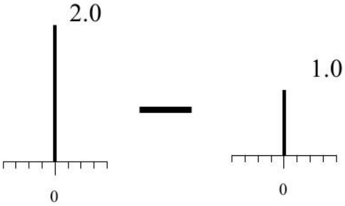
Linear Filtering

original



Linear Filtering

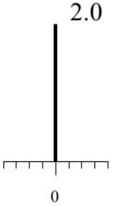
original



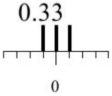
Filtered
(no change)

Linear Filtering

original

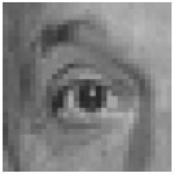


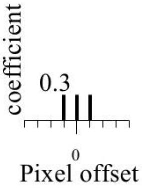
−

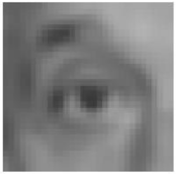


?

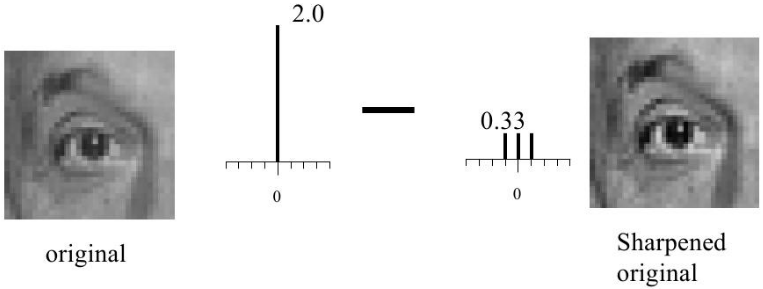
Blurring


original



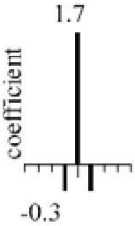

Blurred (filter applied in both dimensions).

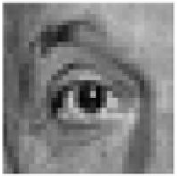
Sharpening



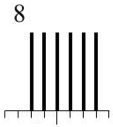
Sharpening

original

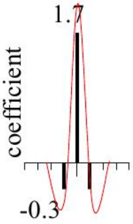


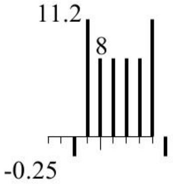

Sharpened original

Sharpening



original





Sharpened (differences are accentuated; constant areas are left untouched).

Sharpening

Original

$$\begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

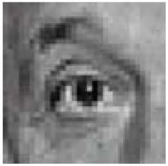
(Note that filter sums to 1)

?

Sharpening

Original

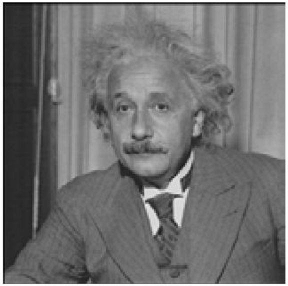
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

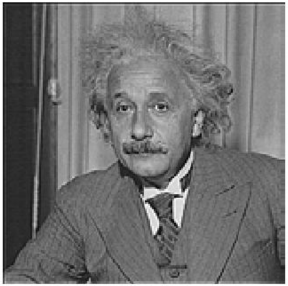


Sharpening filter

- Accentuates differences with local average

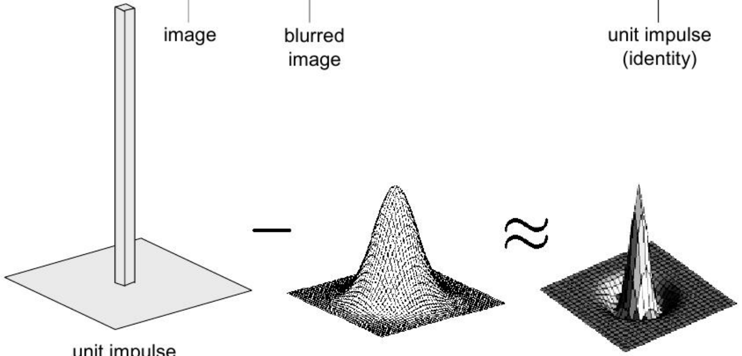
Sharpening


before


after

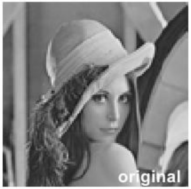
Unsharp Mask Filter

$$\underset{\substack{\uparrow \\ \text{image}}}{I} + \alpha\big(I - \underset{\substack{\uparrow \\ \text{blurred} \\ \text{image}}}{I * g}\big) = (1+\alpha)I - \alpha I * g = I * \big((1+\alpha)\underset{\substack{\uparrow \\ \text{unit impulse} \\ \text{(identity)}}}{e} - g\big)$$

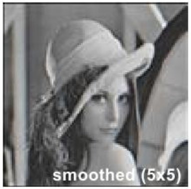


Sharpening Revised

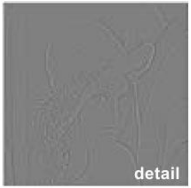
- What does blurring take away?



−

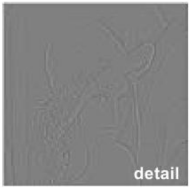


=

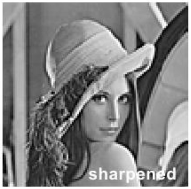


- Let's add it back:

+ α

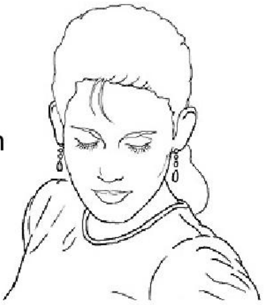


=



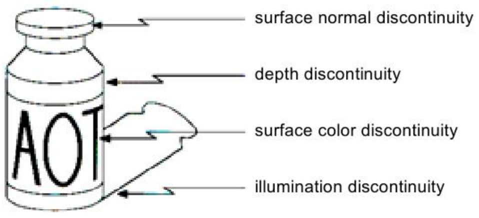
Edge Detection

- **Goal:** Identify sudden changes (discontinuities) in an image
  - Intuitively, most semantic and shape information from the image can be encoded in the edges
  - More compact than pixels
- **Ideal:** artist's line drawing (but artist is also using object-level knowledge)



Source: D. Lowe

Origin of Edges


surface normal discontinuity
depth discontinuity
surface color discontinuity
illumination discontinuity
AOT

- Edges are caused by a variety of factors

Characterising Edges

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

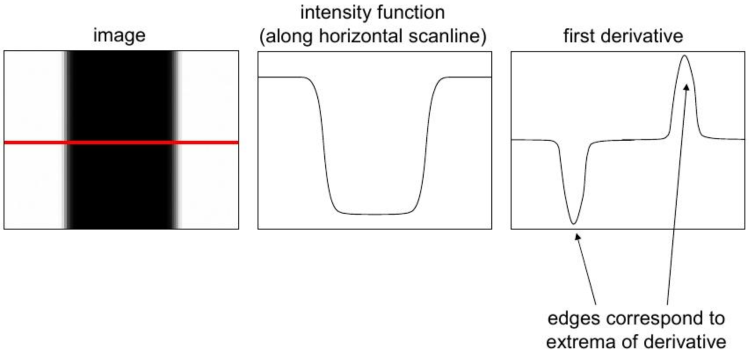
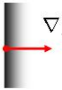


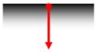
Image Gradient

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

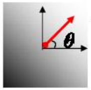
- The gradient points in the direction of most rapid change in intensity



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



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



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

- The gradient direction is given by:

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

  - how does this relate to the direction of the edge? *perpendicular*

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

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

Differentiation and Convolution

- Recall, for 2D function, f(x,y):

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

- This is linear and shift invariant, so must be the result of a convolution.

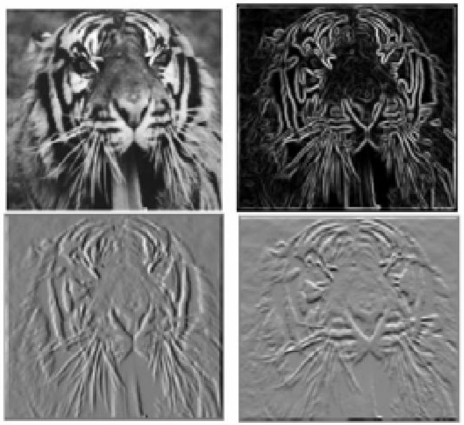
- We could approximate this as

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

- (which is obviously a convolution)

-1	1

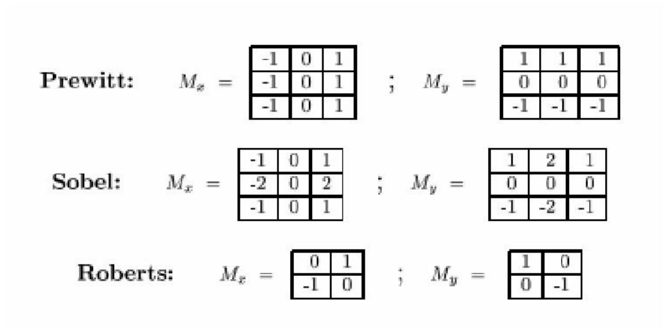
Finite Differences: Example



- Which one is the gradient in the x-direction (resp. y-direction)?

Finite Difference Filters

- Other approximations of derivative filters exist:



Prewitt: $M_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$; $M_y = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$

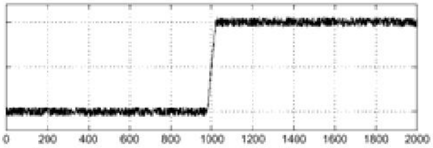
Sobel: $M_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}$; $M_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$

Roberts: $M_x = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline -1 & 0 \\ \hline \end{array}$; $M_y = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array}$

Effects of Noise

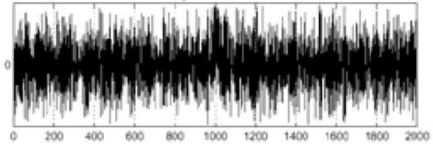
- Consider a single row or column of the image
  - Plotting intensity as a function of position gives a signal

$f(x)$



How to compute a derivative?

$\frac{d}{dx}f(x)$



- Where is the edge?

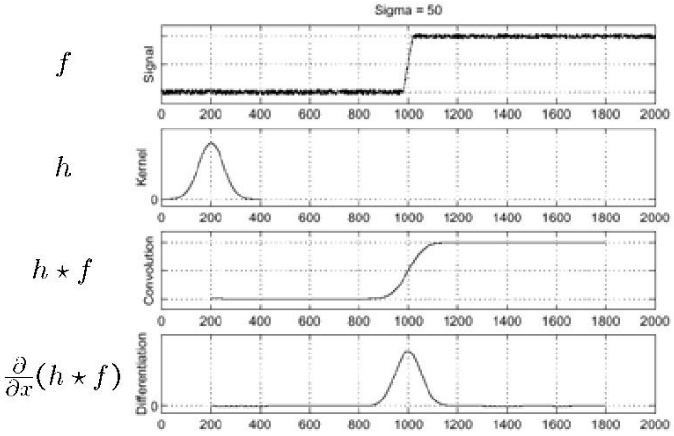
Effects of Noise

- Finite difference filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbors
  - Generally, the larger the noise the stronger the response
- What is to be done?

Effects of Noise

- Finite difference filters respond strongly to noise
  - Image noise results in pixels that look very different from their neighbors
  - Generally, the larger the noise the stronger the response
- What is to be done?
  - Smoothing the image should help, by forcing pixels different to their neighbors (=noise pixels?) to look more like neighbors

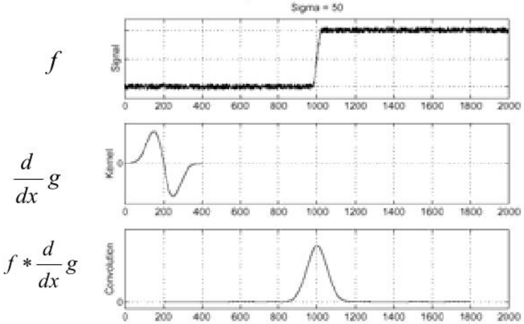
Solution: Smooth First



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

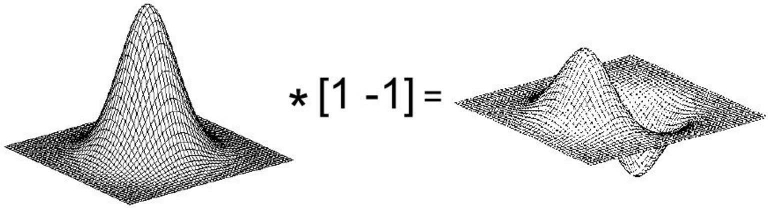
Derivative Theorem of Convolution

- Differentiation is convolution, and convolution is associative: $$\frac{d}{dx}(f * g) = f * \frac{d}{dx} g$$
- This saves us one operation:

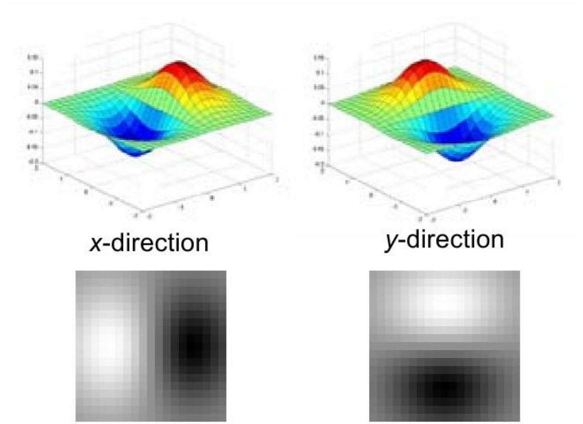


Source: S. Seitz

Derivative of Gaussian Filter



Derivative of Gaussian Filter

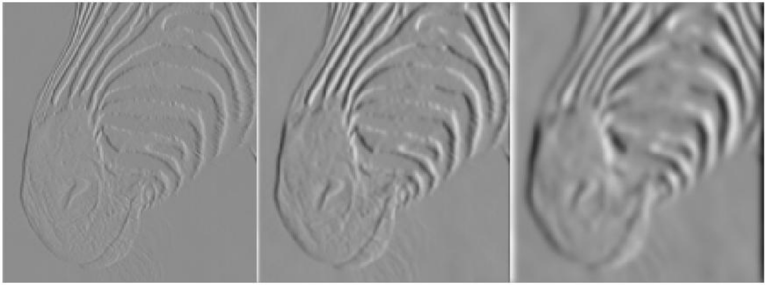


- Which one finds horizontal/vertical edges?

Summary: Filter Mask Properties

- Filters act as templates
  - Highest response for regions that "look the most like the filter"
  - Dot product as correlation
- Smoothing masks
  - Values positive
  - Sum to 1 $\rightarrow$ constant regions are unchanged
  - Amount of smoothing proportional to mask size
- Derivative masks
  - Opposite signs used to get high response in regions of high contrast
  - Sum to 0 $\rightarrow$ no response in constant regions
  - High absolute value at points of high contrast

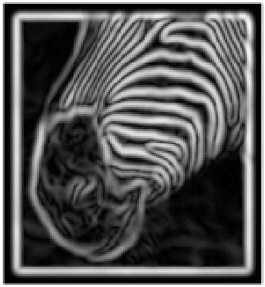
Smoothing vs. Localisation



1 pixel 3 pixels 7 pixels

- Smoothed derivative removes noise, but blurs edge. Also finds edges at different “scales”.

Implementation Issues



- The gradient magnitude is large along a thick "trail" or "ridge," so how do we identify the actual edge points?
- How do we link the edge points to form curves?

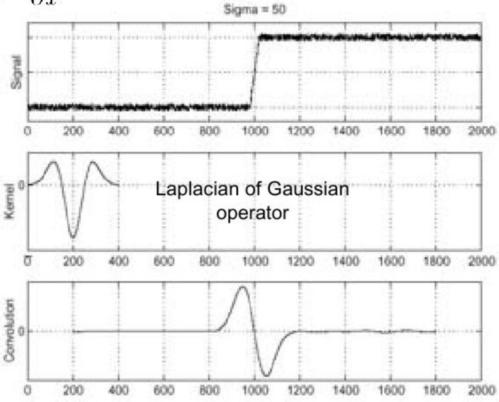
Laplacian of Gaussian

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

f

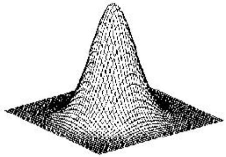
$\frac{\partial^2}{\partial x^2} h$

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

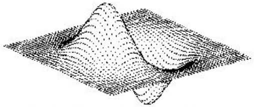


- Where is the edge?
- Zero-crossings of bottom graph

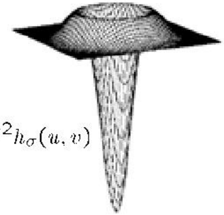
2D Edge Detection Filters


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


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

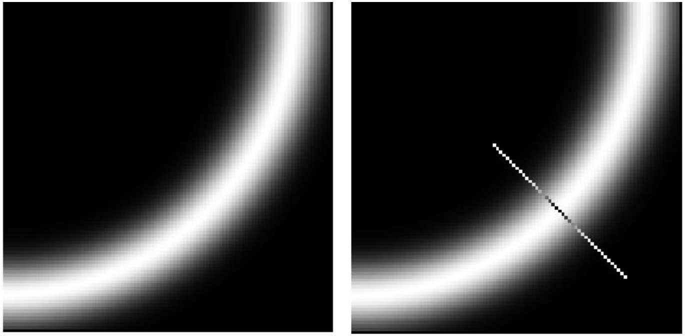
∇^2 is the **Laplacian** operator:

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

MATLAB demo

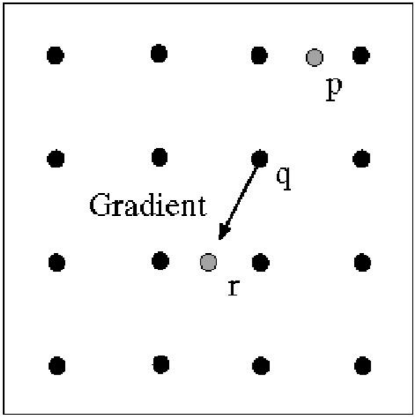
```
g = fspecial('gaussian',15,2);
imagesc(g)
surfl(g)
gclown = conv2(clown,g,'same');
imagesc(conv2(clown,[-1 1],'same'));
imagesc(conv2(gclown,[-1 1],'same'));
dx = conv2(g,[-1 1],'same');
imagesc(conv2(clown,dx,'same'));
lg = fspecial('log',15,2);
lclown = conv2(clown,lg,'same');
imagesc(lclown)
imagesc(clown + .2*lclown)
```

Edge Finding

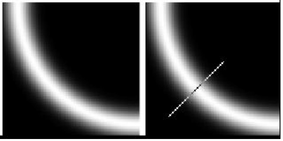


We wish to mark points along the curve where the magnitude is biggest. We can do this by looking for a maximum along a slice normal to the curve (non-maximum suppression). These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?

Non-Maximum Suppression

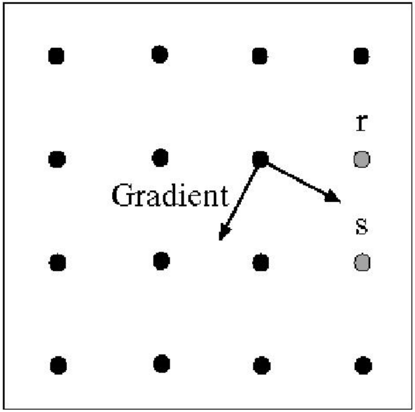


At q, we have a maximum if the value is larger than those at both p and at r. Interpolate to get these values.

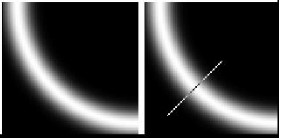


Source: D. Forsyth

Predicting the Next Edge Point



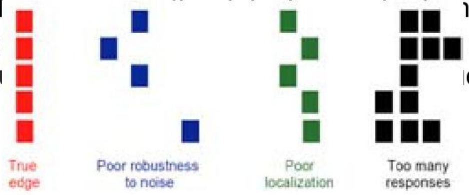
Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either r or s).



Source: D. Forsyth

Designing an Edge Detector

- Criteria for an "optimal" edge detector:
  - **Good detection:** the optimal detector must minimize the probability of false positives (detecting spurious edges caused by noise), as well as that of false negatives (missing real edges)
  - **Good localization:** the edges detected must be as close as possible to the true edges
  - **Singl** ı one point minimize the nı ıe edge



Canny Edge Detector

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

J. Canny, ***A Computational Approach To Edge Detection***, IEEE Trans. Pattern Analysis and Machine Intelligence, 8:679-714, 1986.

Canny Edge Detector

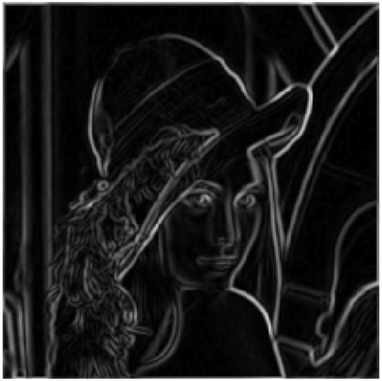
1. Filter image with derivative of Gaussian
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
   - Thin multi-pixel wide “ridges” down to single pixel width
4. Linking and thresholding (hysteresis):
   - Define two thresholds: low and high
   - Use the high threshold to start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

- MATLAB: edge(image, ‘canny’)

Canny Edge Detector

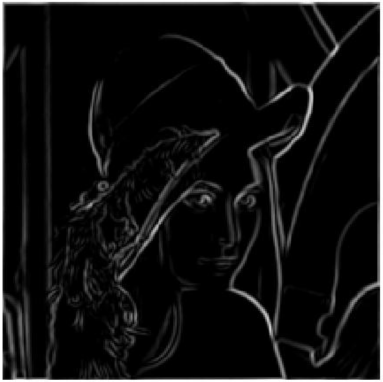
- original image (Lena)

Canny Edge Detector



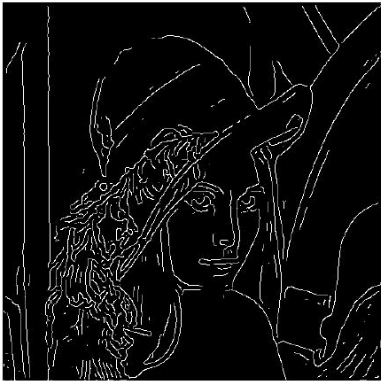
- norm of the gradient

Canny Edge Detector



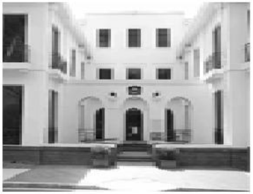
- thresholding

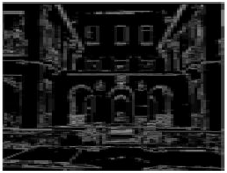
Canny Edge Detector

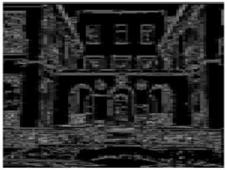


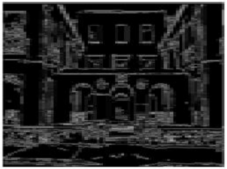
- thinning
- (non-maximum suppression)

Hysteresis Thresholding

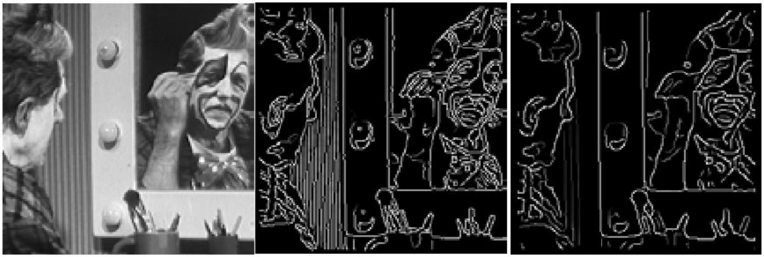

original image


high threshold
(strong edges)


low threshold
(weak edges)


hysteresis threshold

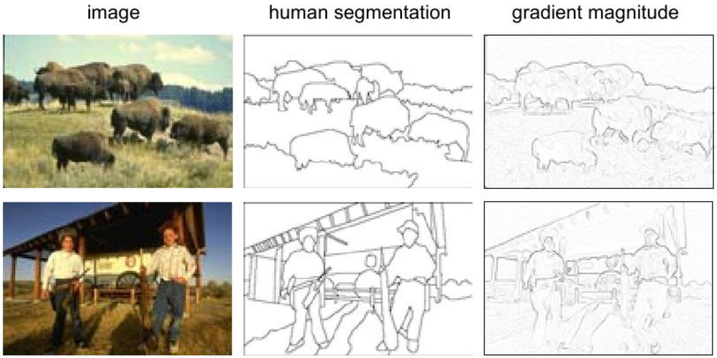
Effect of γ



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

Source: S. Seitz

Edge Detection



- Berkeley segmentation database: http://www.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/segbench/