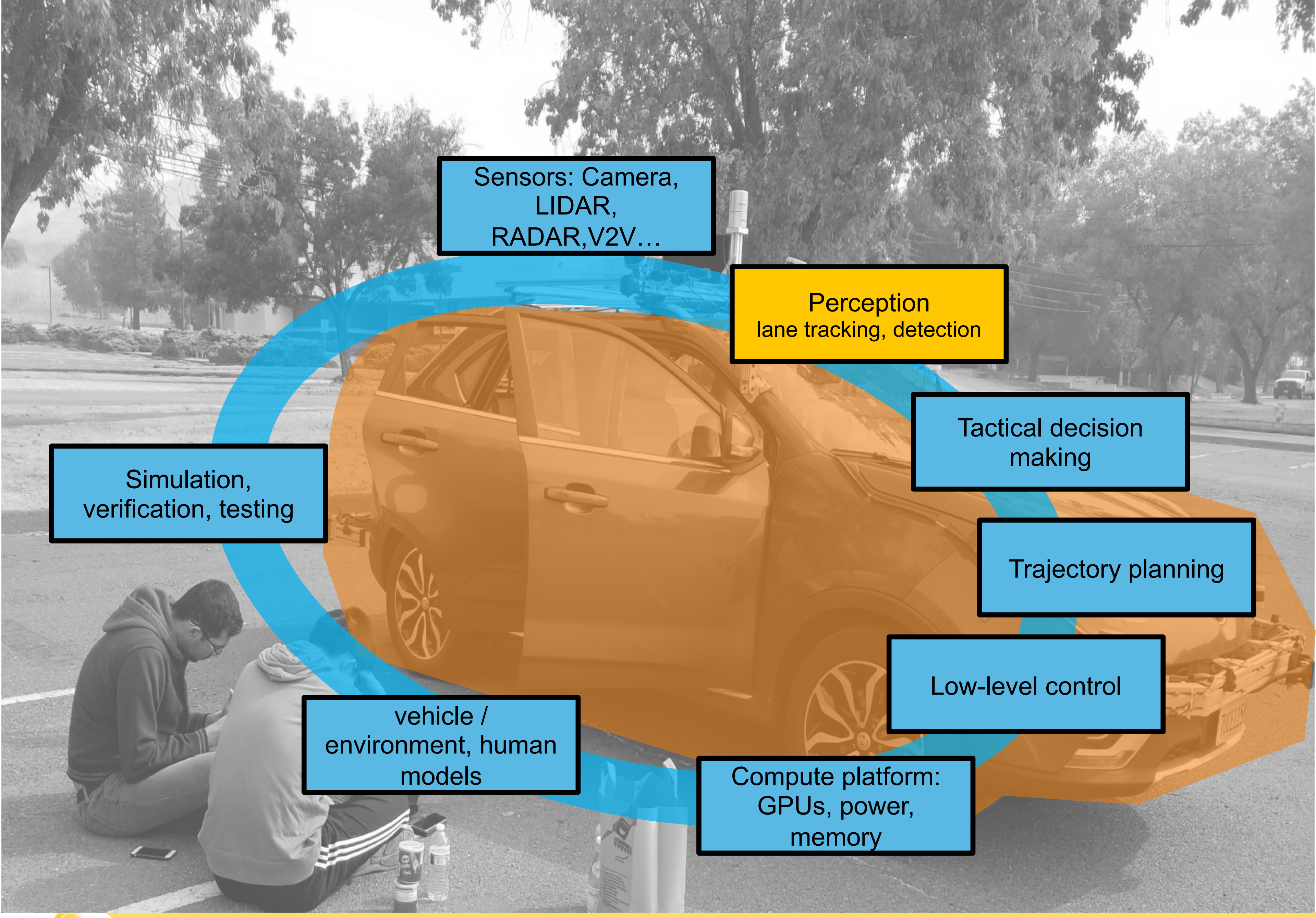
Principles of Safe Autonomy
Lecture 3: Vision 1

Sayan Mitra

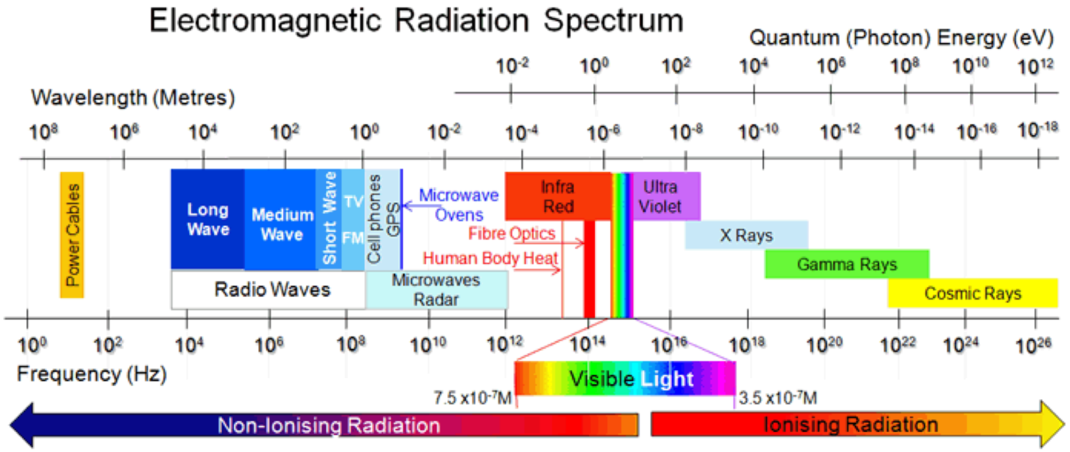
Jan 28, 2019

slides from Svetlana Lazebnik


Sensors: Camera, LIDAR, RADAR,V2V…
Perception
lane tracking, detection
Tactical decision making
Simulation, verification, testing
Trajectory planning
Low-level control
vehicle / environment, human models
Compute platform: GPUs, power, memory

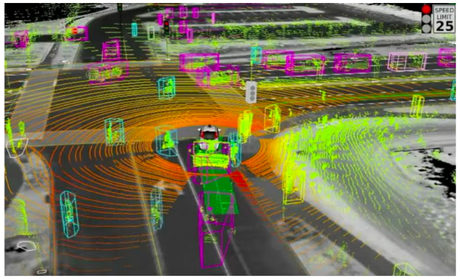
Perception: EM to objects

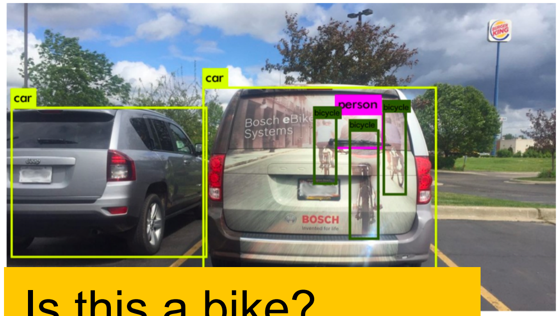
Problem: Process electromagnetic radiation from the environment to construct a *model* of the world, so that the constructed model is close to the real world



Challenging for computers: millions of years of evolution

Ill-defined problem: impossibility of defining meaning “car”, “bicycle”, etc.

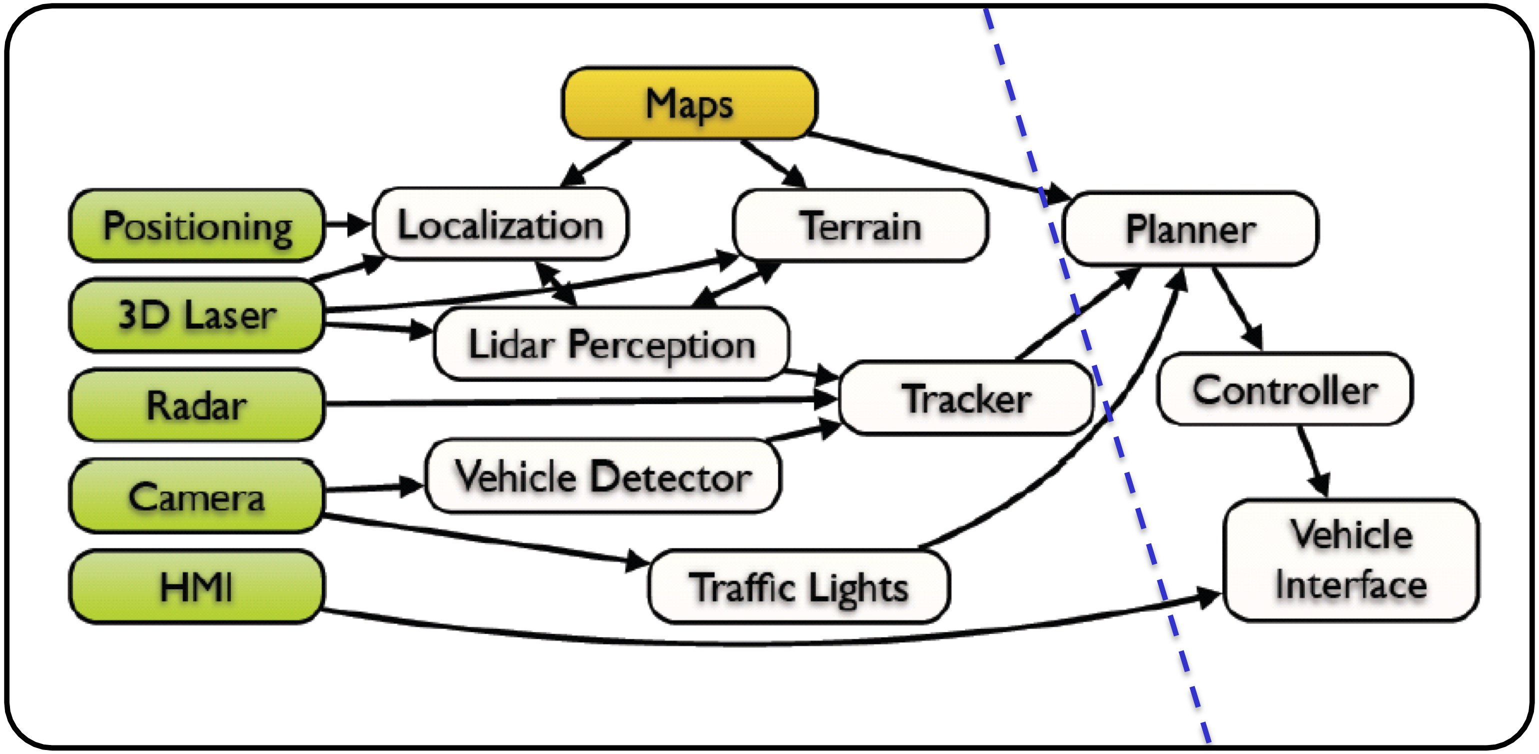




Is this a bike?

Is this a car?

Perception has many pieces

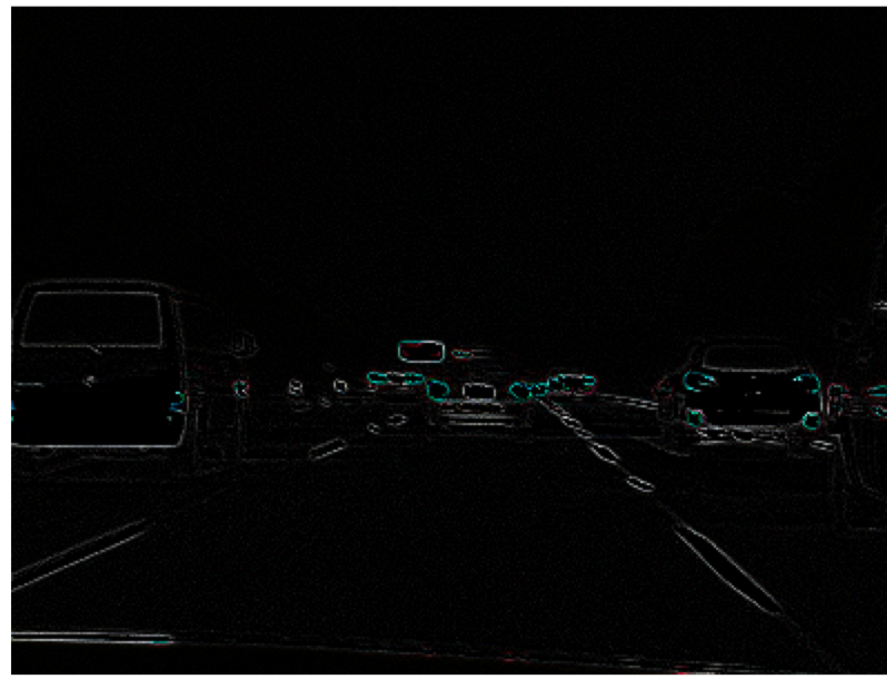


This architecture from a slide from M. James of Toyota Research Institute, North America

Outline

- Linear filtering (Today)
- Edge detection (Today)
- Classical recognition (Wed)
- Neural networks (Next week)

Linear filtering



Motivation: Image denoising

- How can we reduce noise in a photograph?

Moving average

- Let's replace each pixel with a *weighted* average of its neighborhood
- The weights are called the *filter kernel*
- What are the weights for the average of a 3x3 neighborhood?

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$

"box filter"

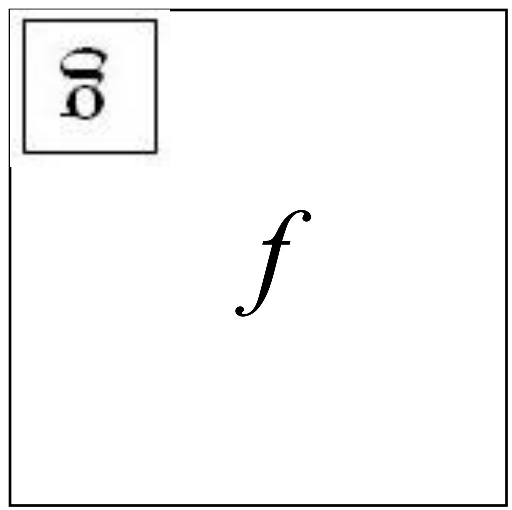
Source: D. Lowe

Defining convolution

- Let *f* be the image and *g* be the kernel. The output of convolving *f* with *g* is denoted *f* * *g*.

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

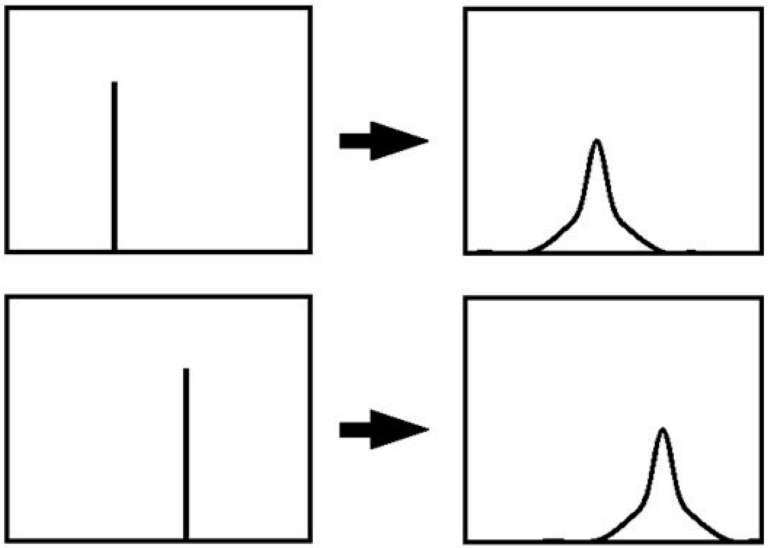
Convention: kernel is "flipped"



Source: F. Durand

Key properties

- **Shift invariance:** same behavior regardless of pixel location: filter(shift(f)) = shift(filter(f))



- **Linearity:** filter(f_1 + f_2) = filter(f_1) + filter(f_2)

- <u>**Theoretical result:**</u> any linear shift-invariant operator can be represented as a convolution

Properties in more detail

- 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 $e = [\ldots, 0, 0, 1, 0, 0, \ldots]$, $a * e = a$

openCV: filter2D

Output image same size as input

Multi-channel: each channel is processed independently

Extrapolation of border

Examples

Dealing with edges

What about missing pixel values?

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

Source: S. Marschner

cv.filter2D(src, dst, cv.CV_8U, M, anchor, 0,
cv.BORDER_DEFAULT);

src input image

dst output image of the same size, same # of channels

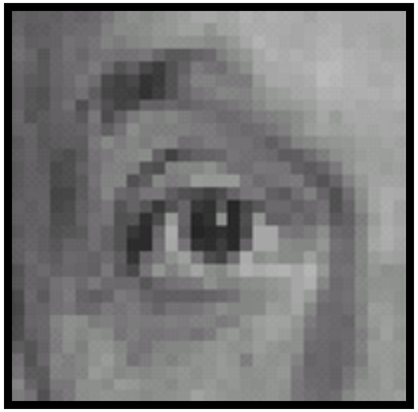
kernel convolution kernel, a single-channel floating point matrix

anchor relative position of a filtered point within the kernel; default value cv.Point(-1, -1) = kernel center.

borderType pixel extrapolation method(see cv.BorderTypes)

BORDER_DEFAULT, BORDER_WRAP

Practice with linear filters



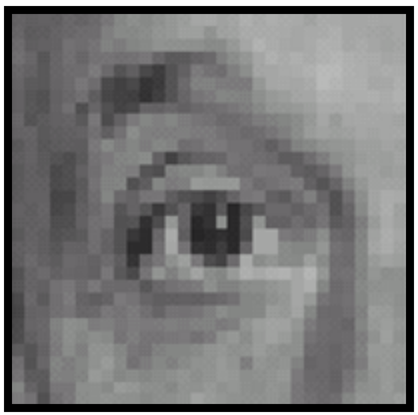
Original

0	0	0
0	1	0
0	0	0

?

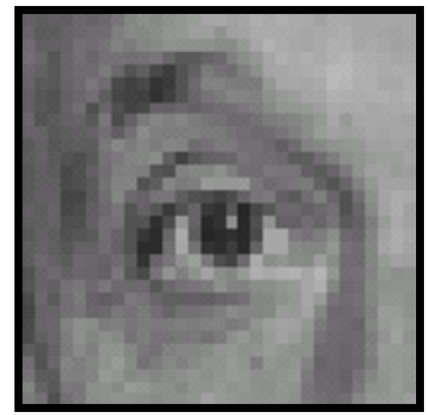
Source: D. Lowe

Practice with linear filters



Original

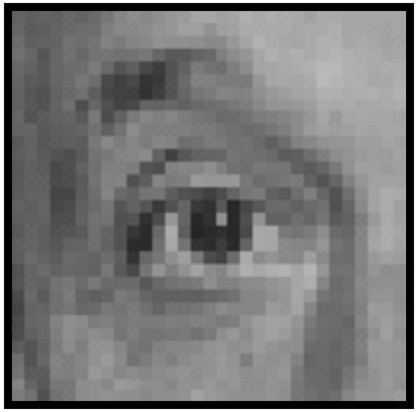
0	0	0
0	1	0
0	0	0



Filtered
(no change)

Source: D. Lowe

Practice with linear filters

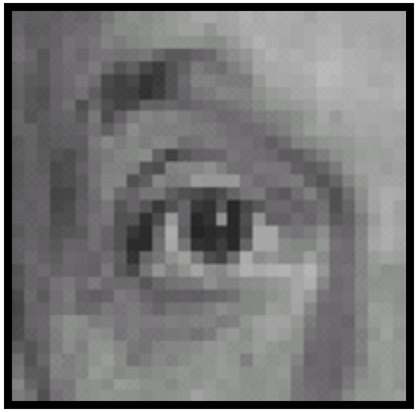


Original

0	0	0
0	0	1
0	0	0

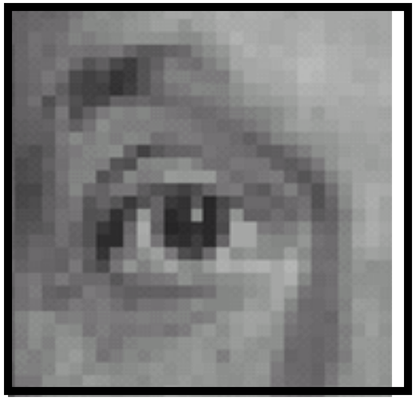
?

Practice with linear filters



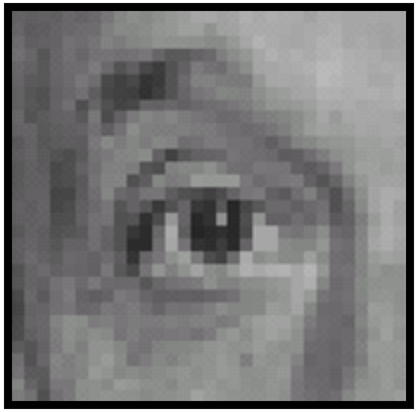
Original

0	0	0
0	0	1
0	0	0



Shifted *left*
By 1 pixel

Practice with linear filters

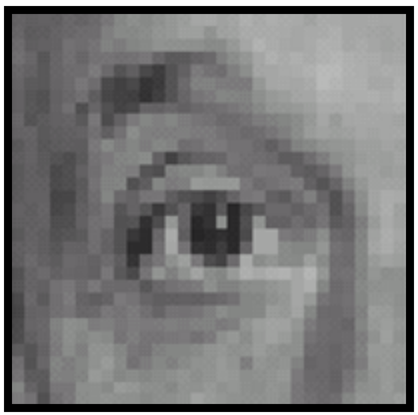


Original

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

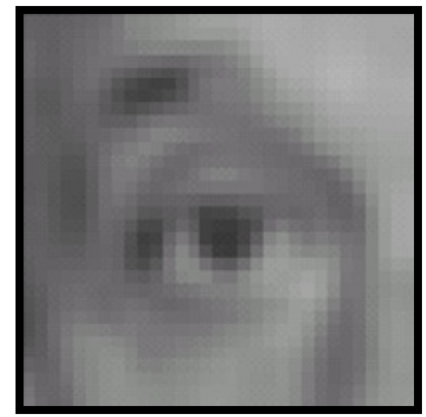
?

Practice with linear filters



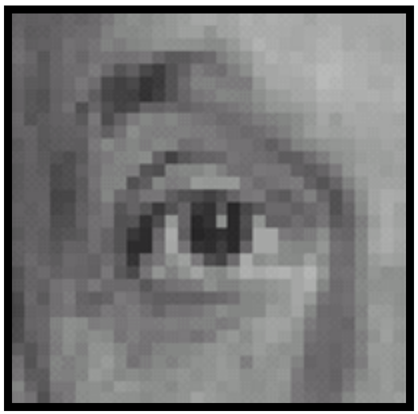
Original

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



Blur (with a box filter)

Practice with linear filters



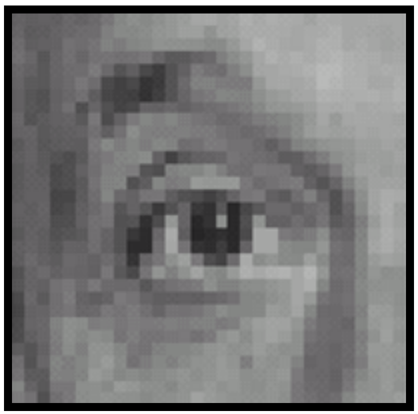
Original

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

?

(Note that filter sums to 1)

Practice with linear filters



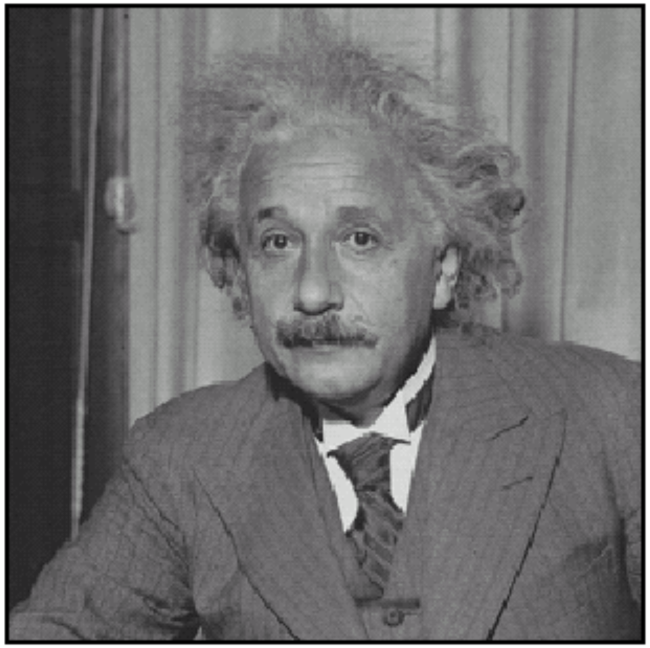
Original

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

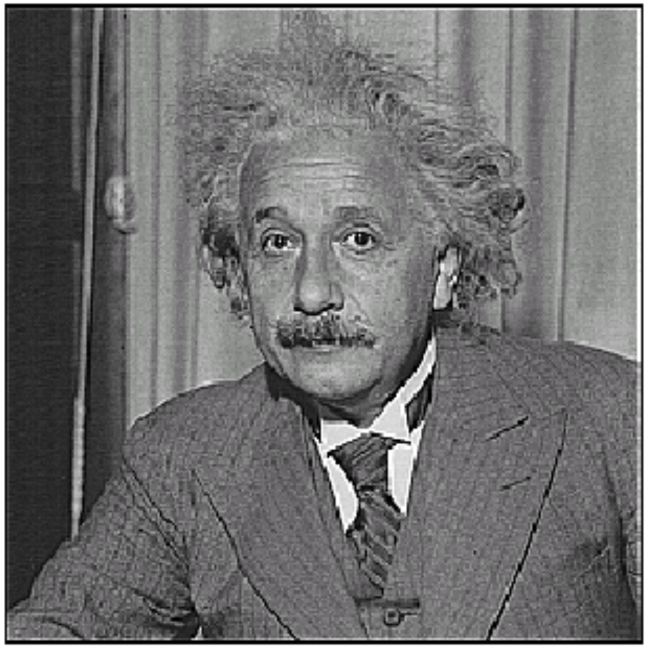
Sharpening filter

- Accentuates differences with local average

Sharpening



before



after

Sharpening

What does blurring take away?

−

=

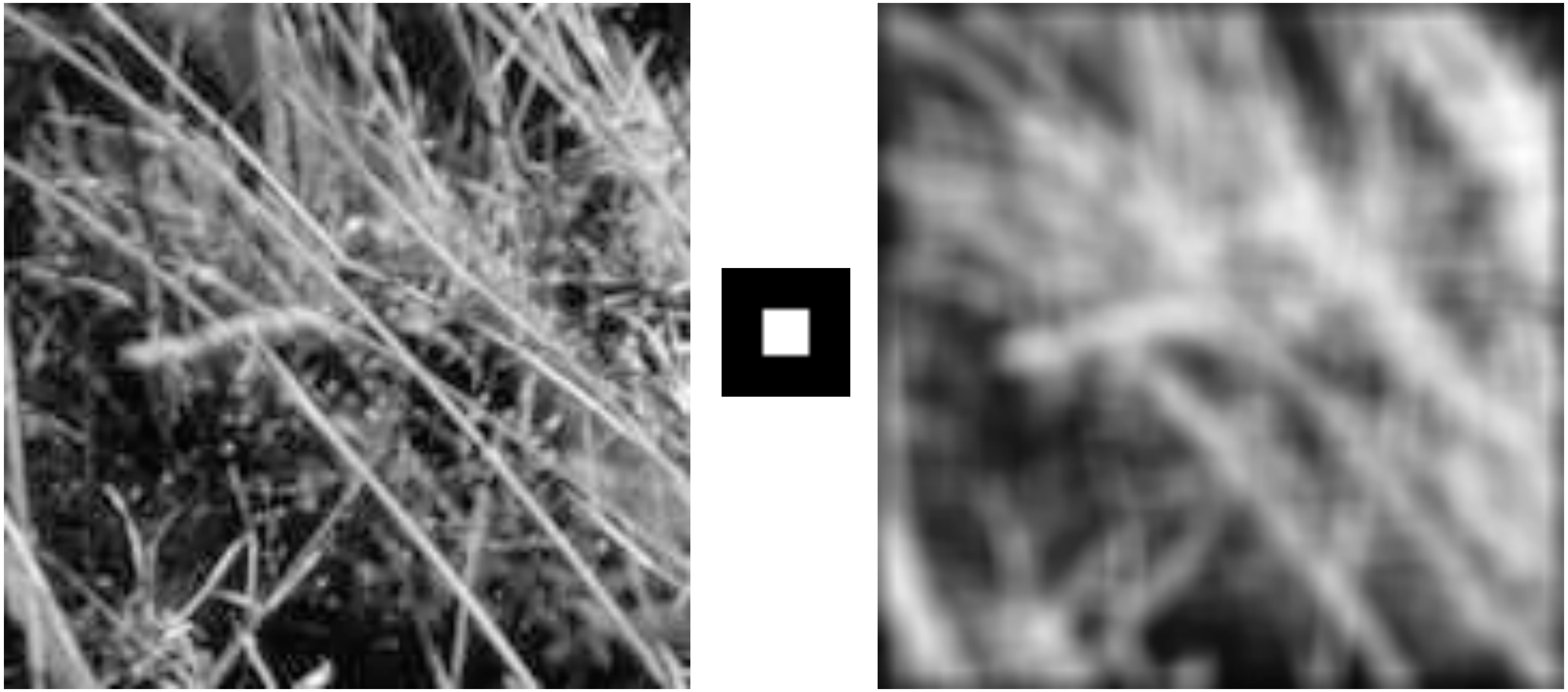
Let's add it back:

+

=

Smoothing with box filter revisited

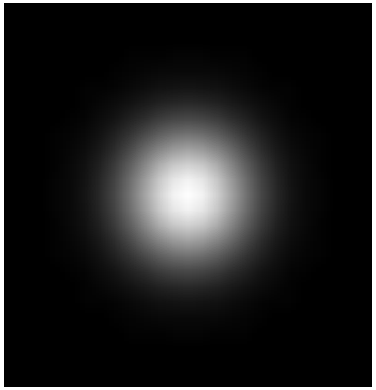
- What’s wrong with this picture?
- What’s the solution?



Source: D. Forsyth

Smoothing with box filter revisited

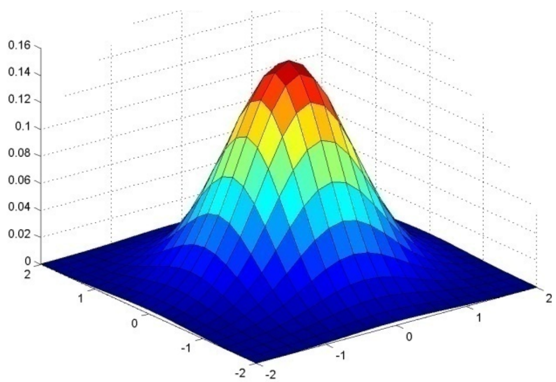
- What's wrong with this picture?
- What's the solution?
  - To eliminate edge effects, weight contribution of neighborhood pixels according to their closeness to the center

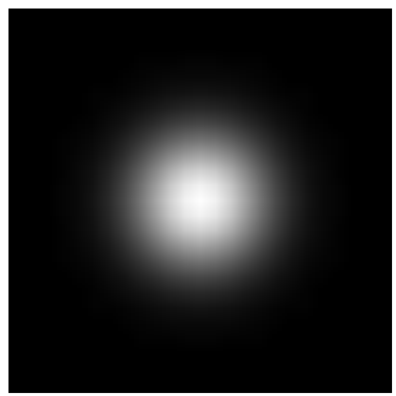


"fuzzy blob"

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

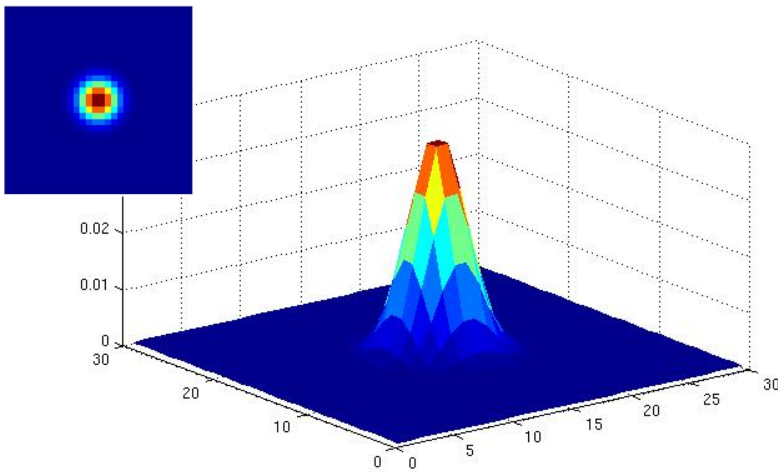
5 x 5, $\sigma = 1$

Constant factor at front makes volume sum to 1 (can be ignored when computing the filter values, as we should renormalize weights to sum to 1 in any case)

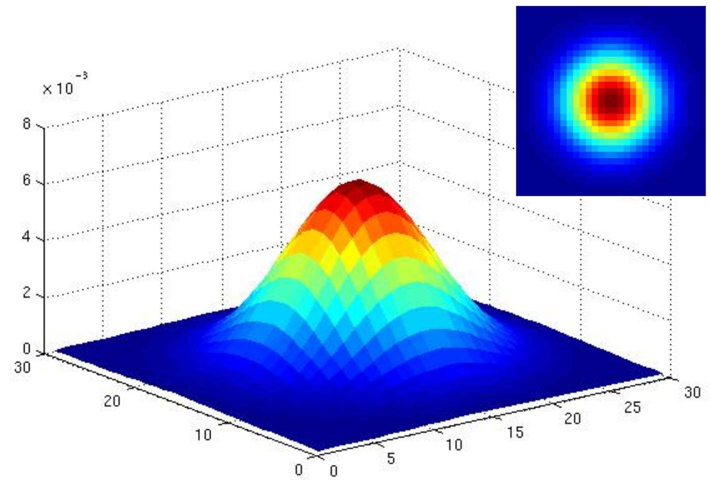
Source: C. Rasmussen

Gaussian Kernel

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



σ = 2 with 30 x 30 kernel



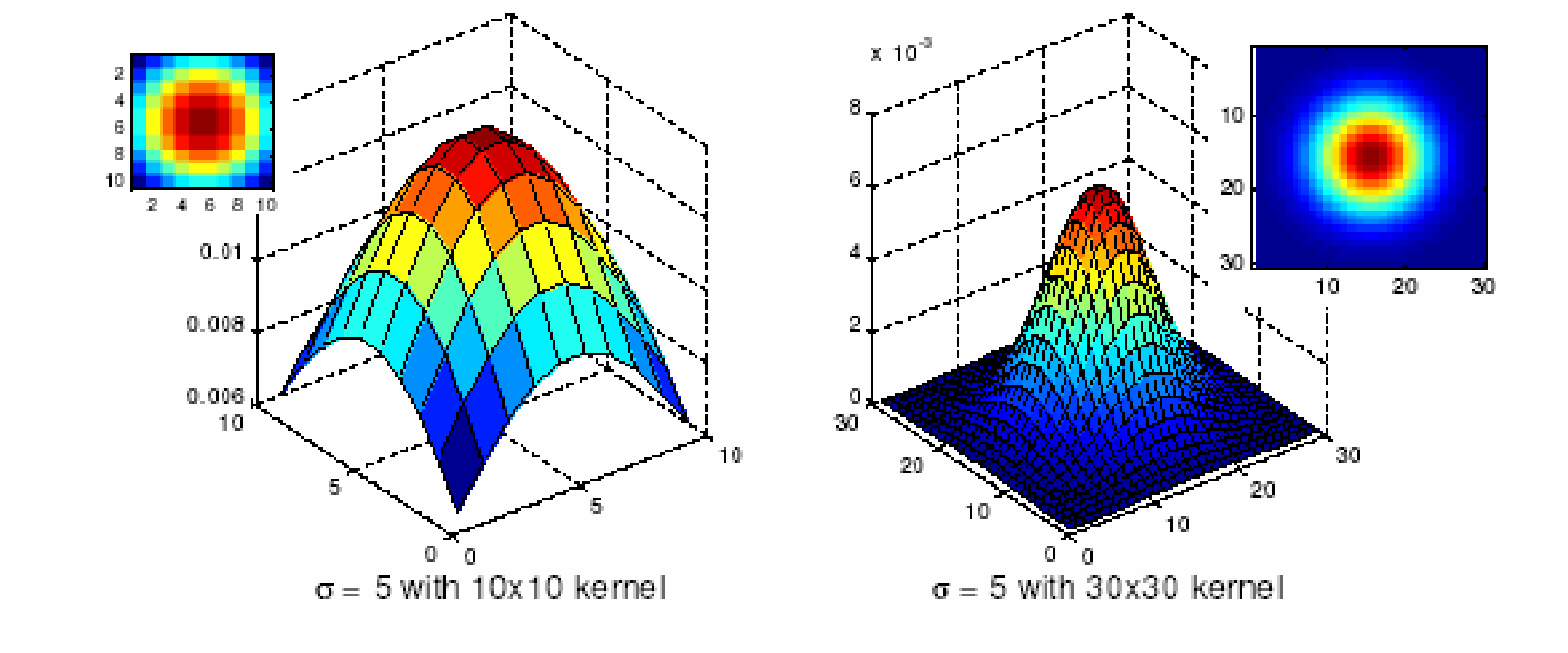
σ = 5 with 30 x 30 kernel

Standard deviation σ: determines extent of smoothing

Source: K. Grauman

Choosing kernel width

The Gaussian function has infinite support, but discrete filters use finite kernels

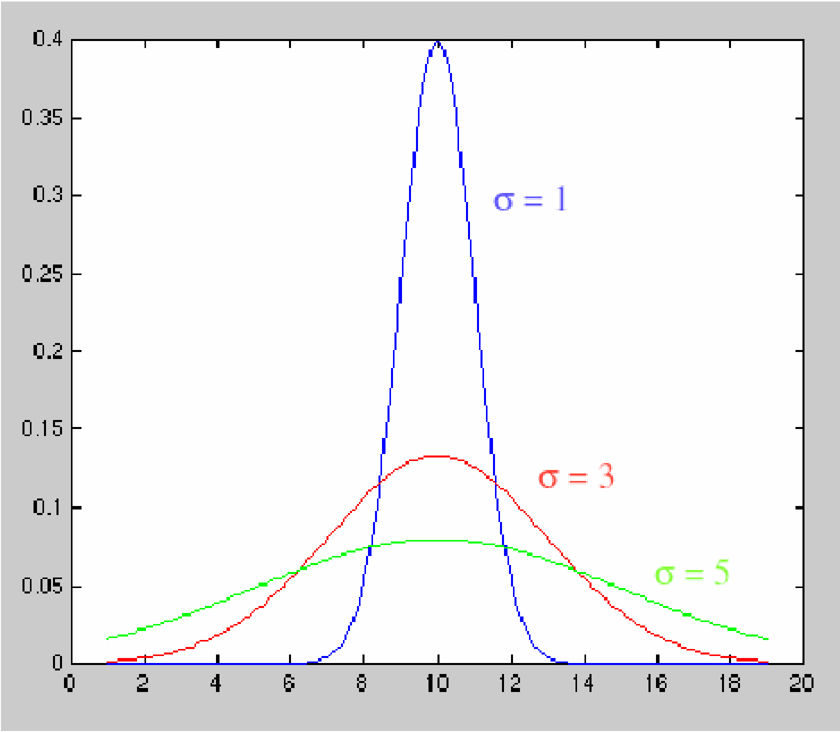


Source: K. Grauman

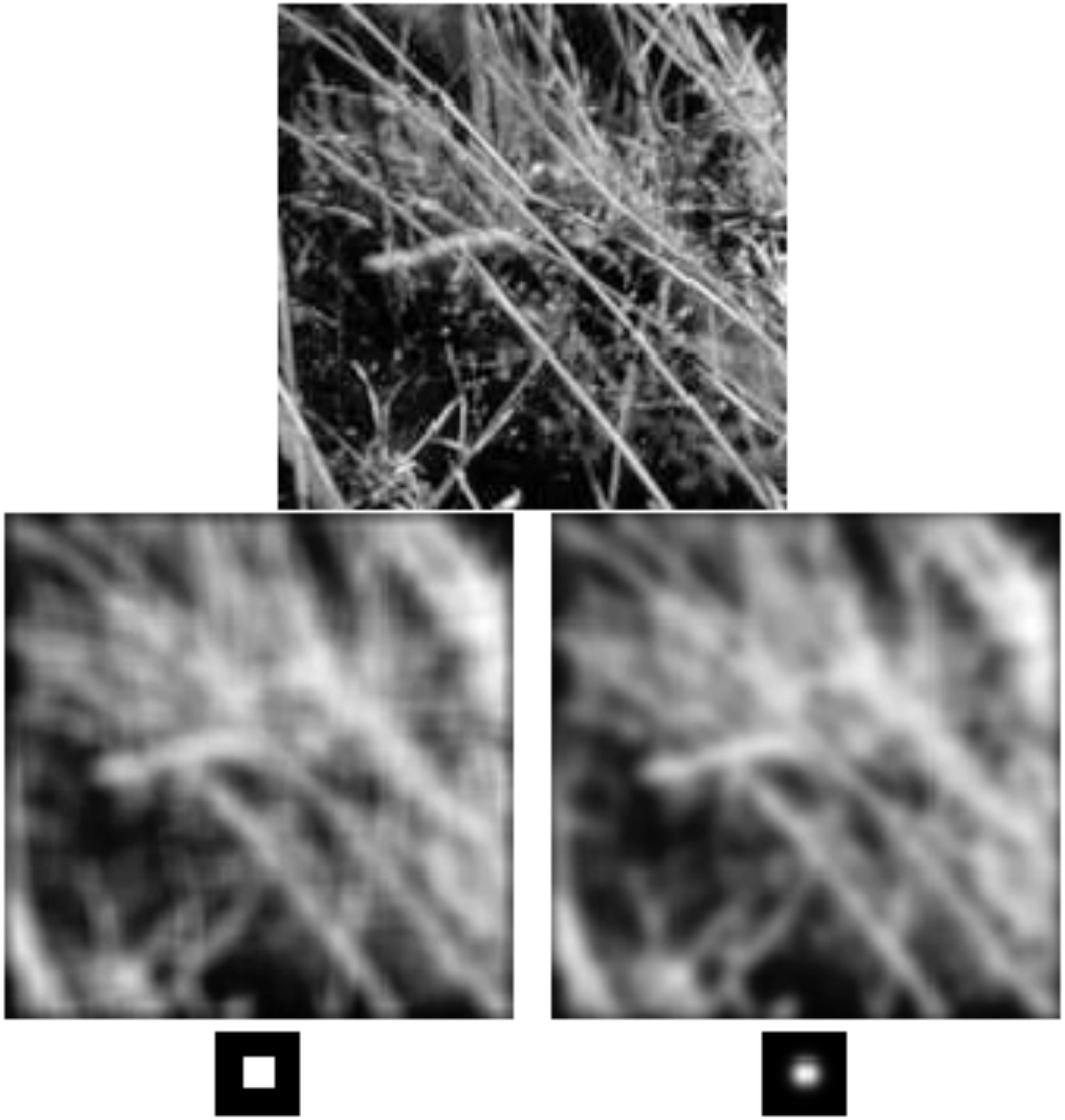
Choosing kernel width

Rule of thumb: set filter half-width to about 3σ

Effect of σ



Gaussian vs. box filtering



Gaussian filters

- Remove high-frequency components from the image (*low-pass filter*)
- Convolution with self is another Gaussian
  - So can smooth with small-σ kernel, repeat, and get same result as larger-σ kernel would have
  - Convolving two times with Gaussian kernel with std. dev. σ is same as convolving once with kernel with std. dev. $\sigma\sqrt{2}$
- *Separable* kernel
  - Factors into product of two 1D Gaussians
  - Discrete example:

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$$

Source: K. Grauman

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

Why is separability useful?

- Separability means that a 2D convolution can be reduced to two 1D convolutions (one along rows and one along columns)
- What is the complexity of filtering an $n \times n$ image with an $m \times m$ kernel?
  - $O(n^2 m^2)$
- What if the kernel is separable?
  - $O(n^2 m)$

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

Reducing salt-and-pepper noise

3x3

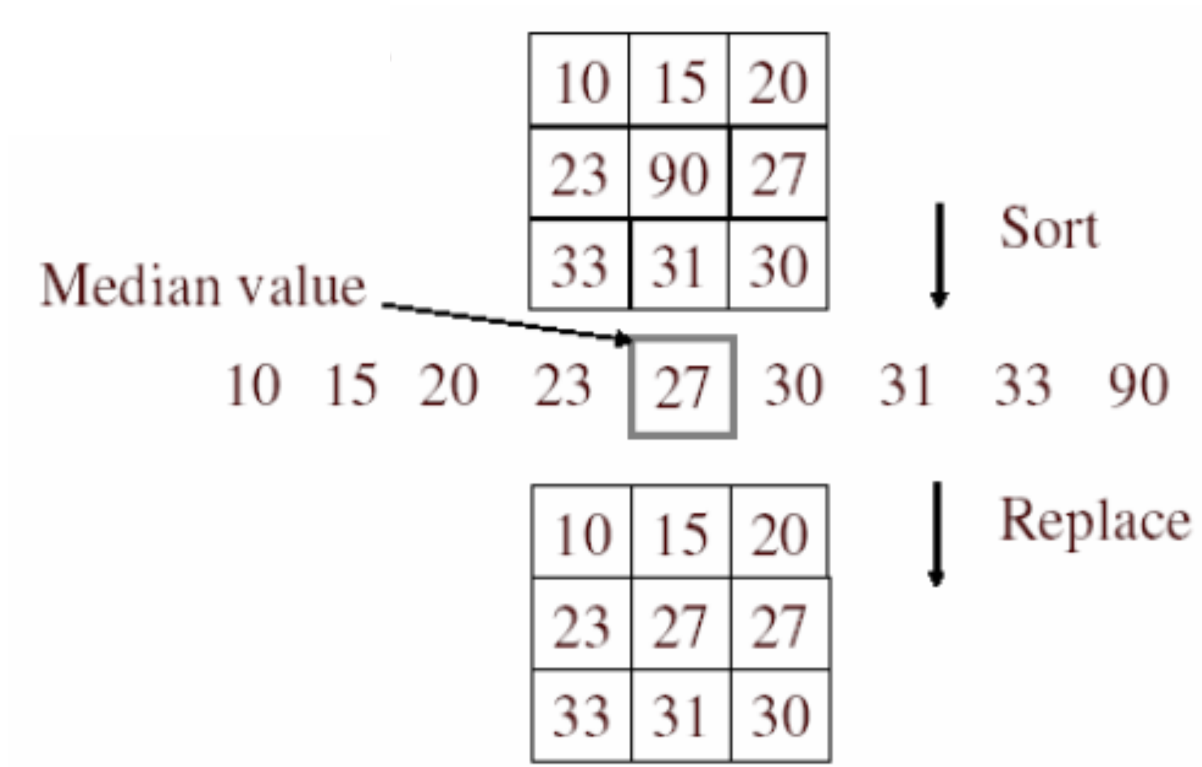
5x5

7x7

What’s wrong with the results?

Alternative idea: Median filtering

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



- Is median filtering linear?

Median filter

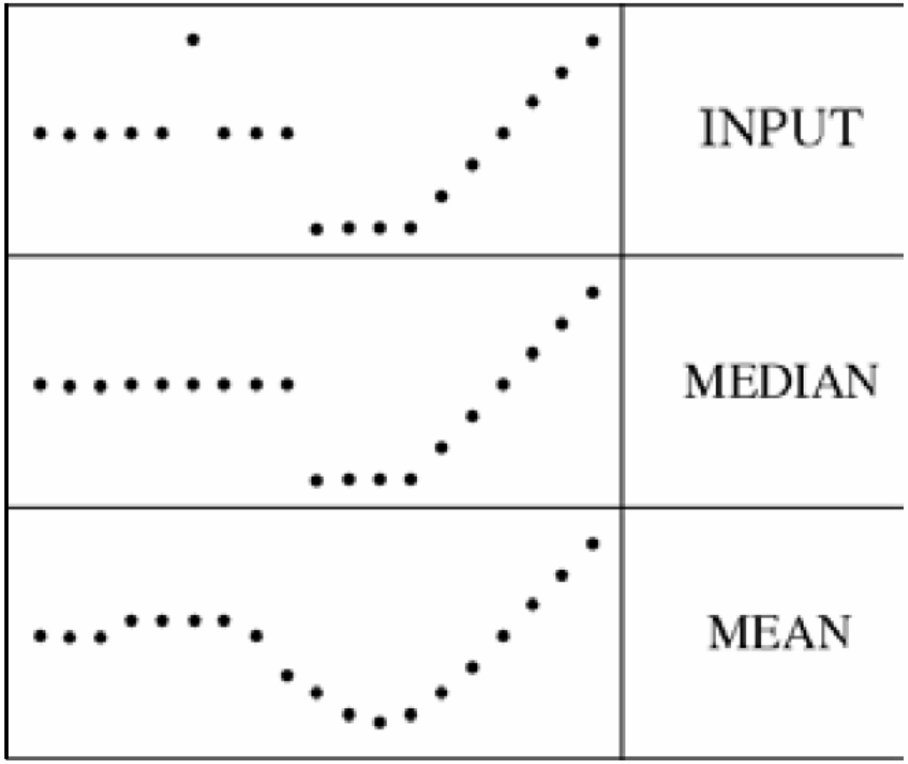
- Is median filtering linear?
- Let's try filtering

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 2 \\ 2 & 2 & 2 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Median filter

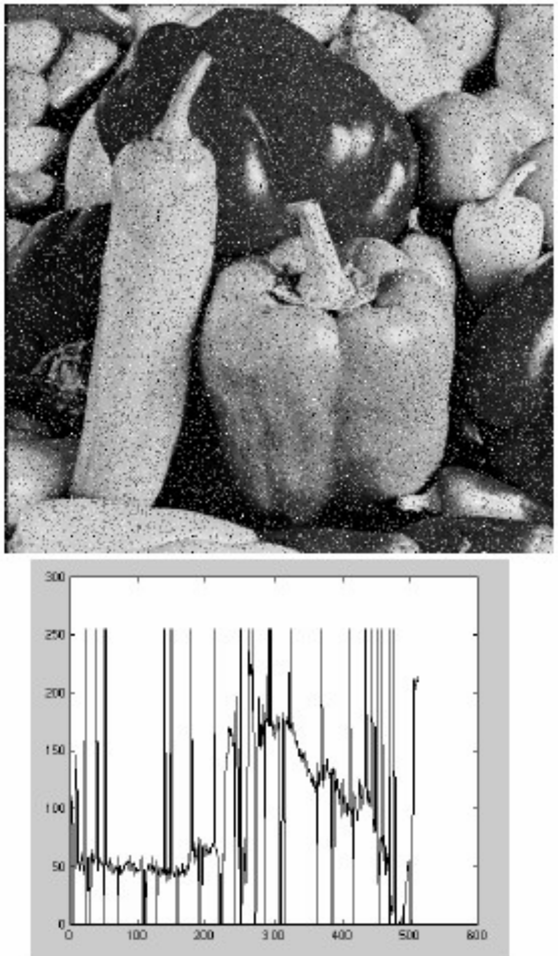
- What advantage does median filtering have over Gaussian filtering?
    - Robustness to outliers

filters have width 5 :

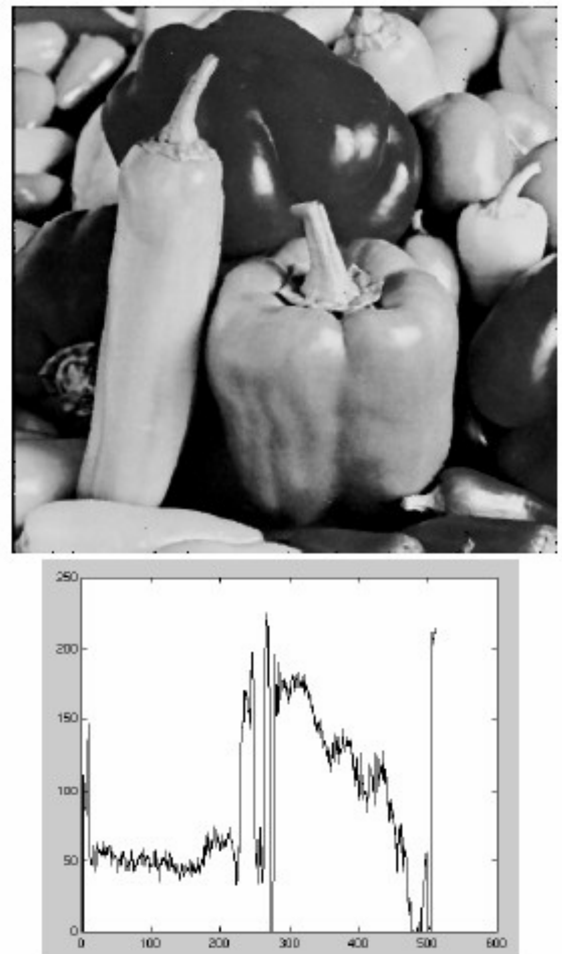


Median filter

Salt-and-pepper noise



Median filtered



open cv: cv2.medianBlur (input, output,ksize)

Source: M. Hebert

Gaussian vs. median filtering

3x3 5x5 7x7

Gaussian

Median

Review: Image filtering

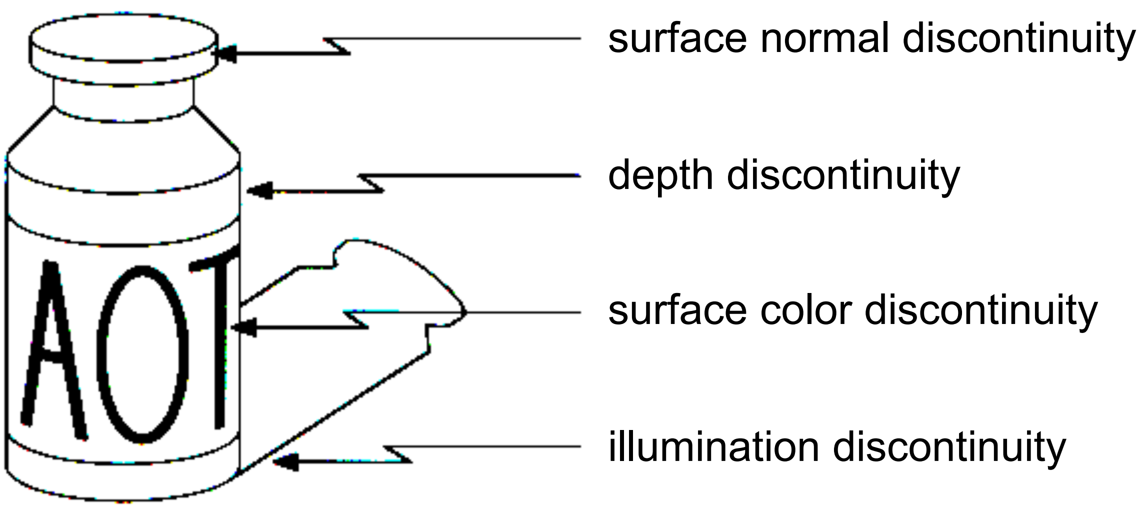
- Convolution
- Box vs. Gaussian filter
- Separability
- Median filter

Edge detection

Winter in Kraków photographed by Marcin Ryczek

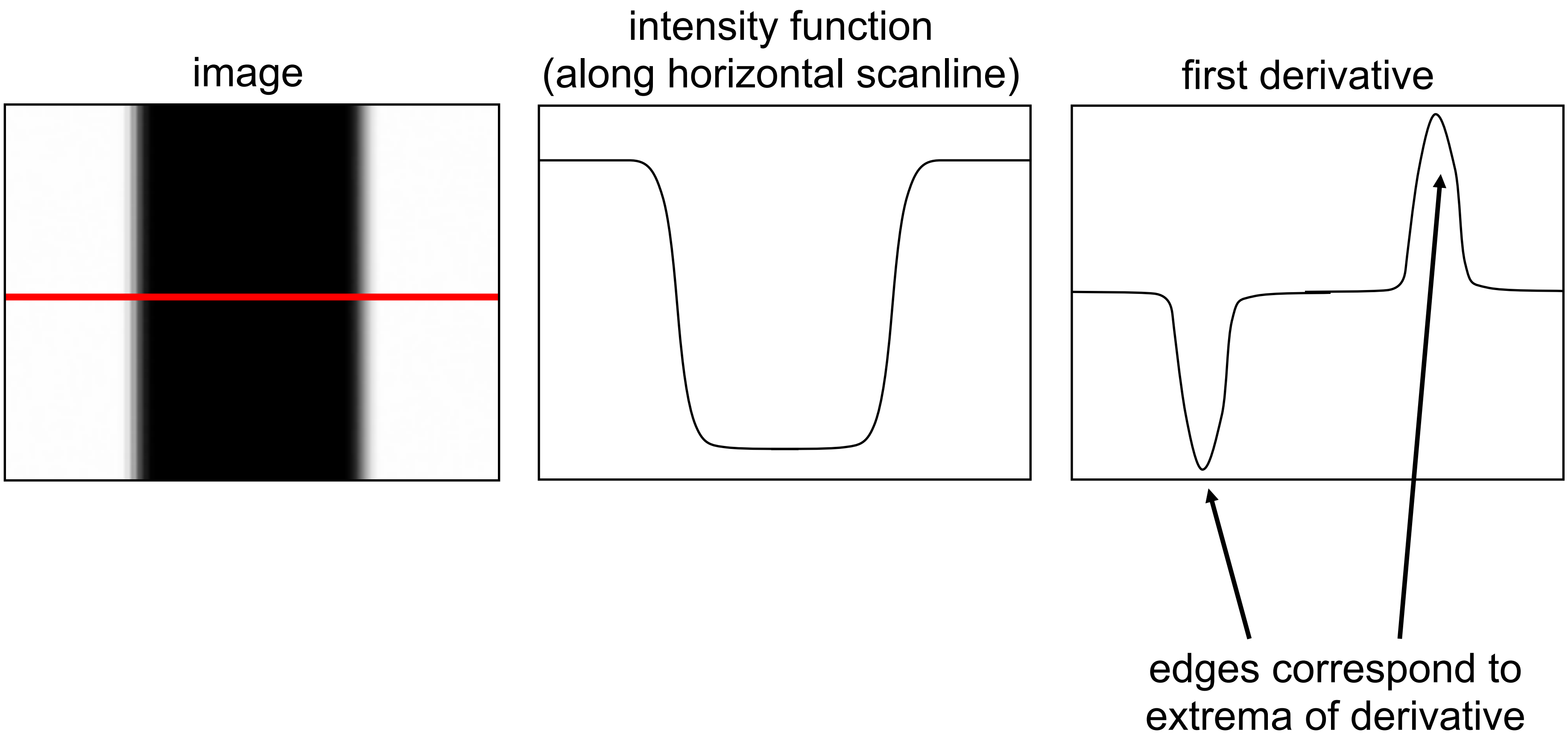
Edge detection

- **Goal:** Identify sudden changes (discontinuities) in an image
- Intuitively, edges carry most of the semantic and shape information from the image
  - E.g., Lanes, traffic signs, cars



Edge detection

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



Derivatives with convolution

For 2D function f(x,y), the partial derivative w.r.t *x* is:

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

For discrete data, we can approximate using finite differences:

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

To implement the above as convolution, what would be the associated filter?

Source: K. Grauman

Partial derivatives of an image

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

$$\begin{array}{|c|c|}\hline -1 & 1 \\ \hline \end{array}$$

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

$$\begin{array}{|c|}\hline -1 \\ \hline 1 \\ \hline \end{array} \qquad \begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline \end{array}$$

Which shows changes with respect to x?

Finite difference filters

Other approximations of derivative filters exist:

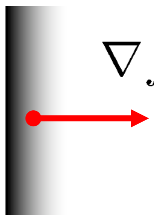
Prewitt: $M_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline\end{array}$; $M_y = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline\end{array}$

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

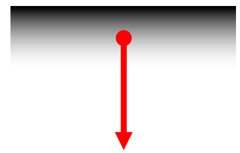
Roberts: $M_x = \begin{array}{|c|c|}\hline 0 & 1 \\ \hline -1 & 0 \\ \hline\end{array}$; $M_y = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & -1 \\ \hline\end{array}$

Image gradient

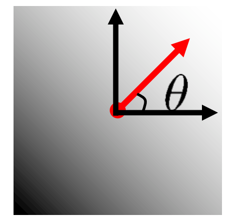
The gradient of an image: $$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$



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



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



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

The gradient points in the direction of most rapid increase in intensity

- How does this direction relate to the direction of the edge?

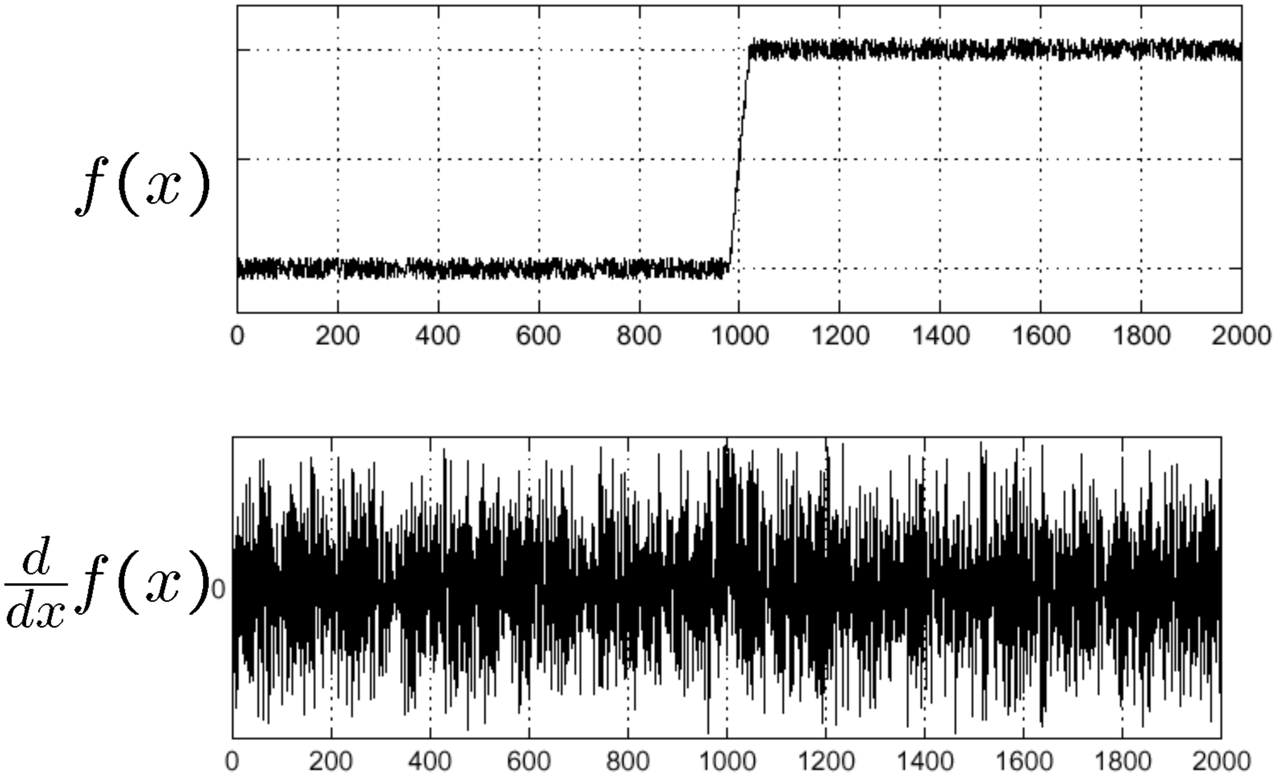
The gradient direction is given by $$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

The edge strength is given by the gradient magnitude (norm)

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

Effects of noise

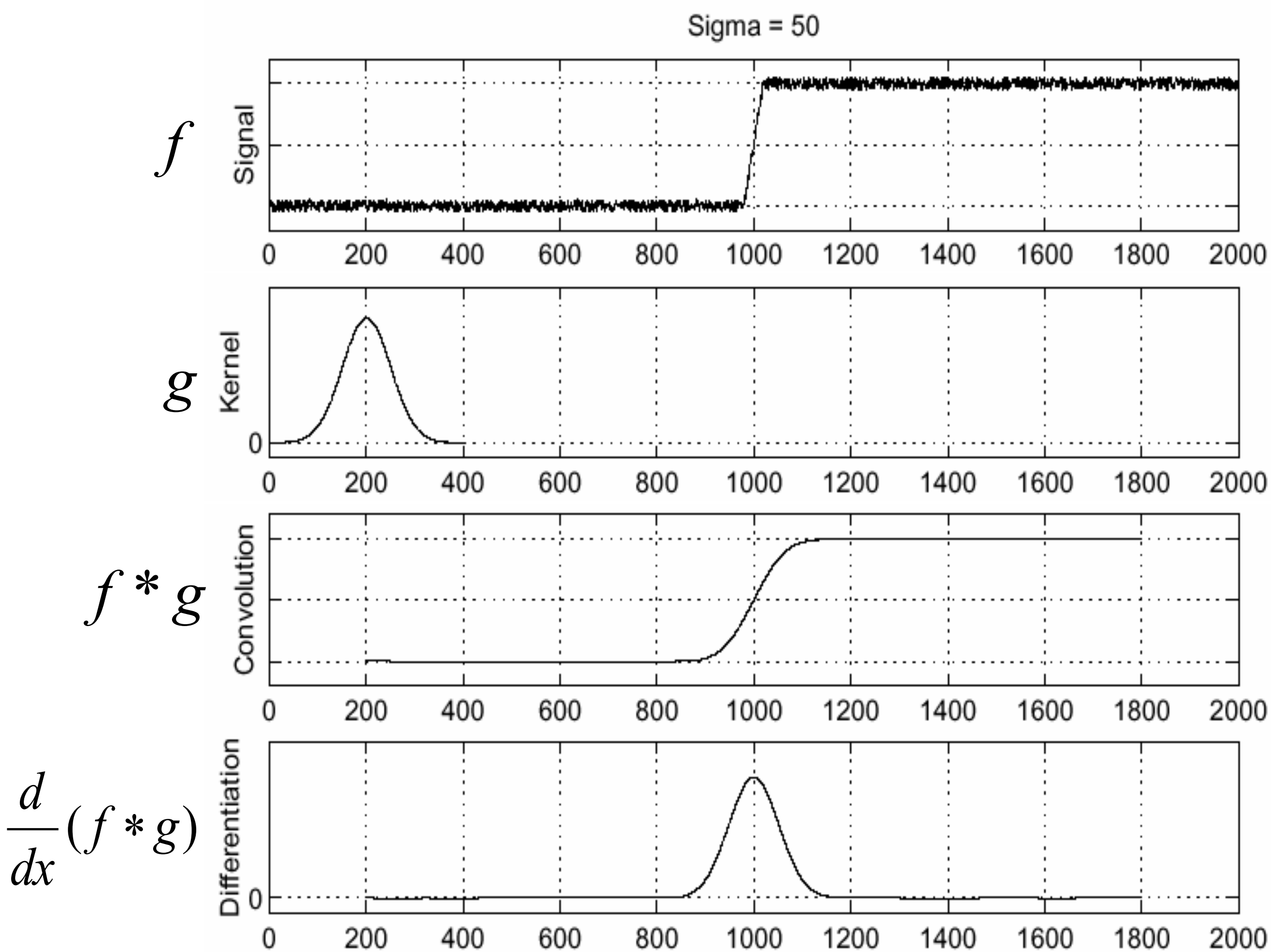
Consider a single row or column of the image



Where is the edge?

Source: S. Seitz

Solution: smooth first



- To find edges, look for peaks in $\frac{d}{dx}(f * g)$

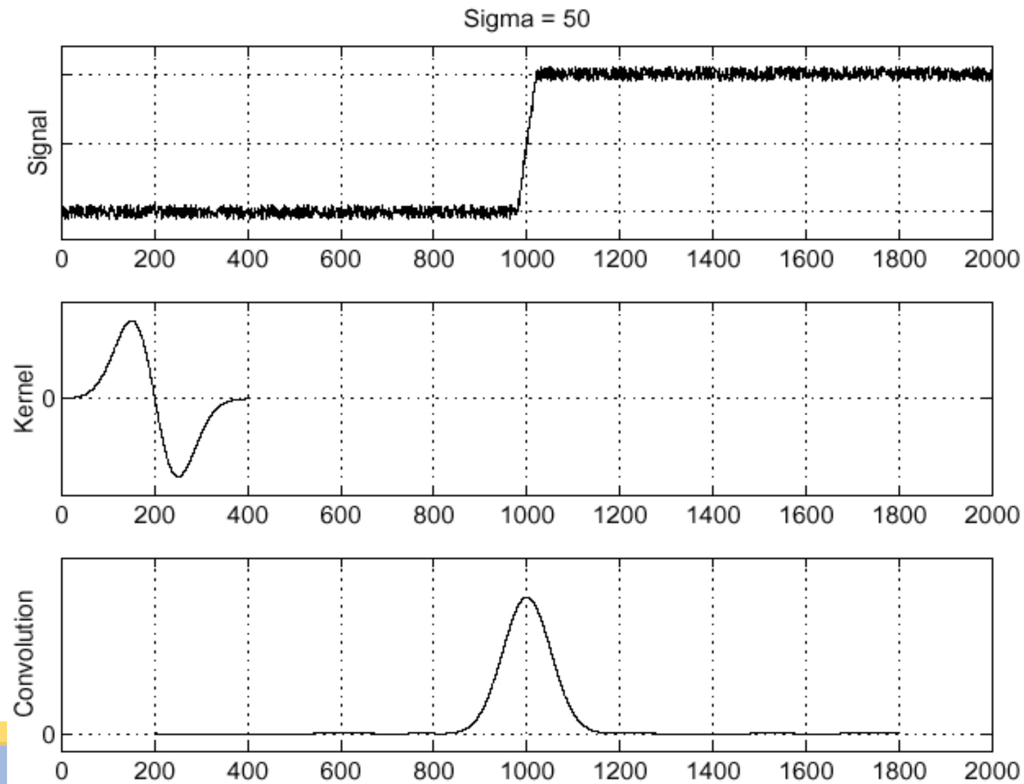
Derivative theorem of convolution

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

f

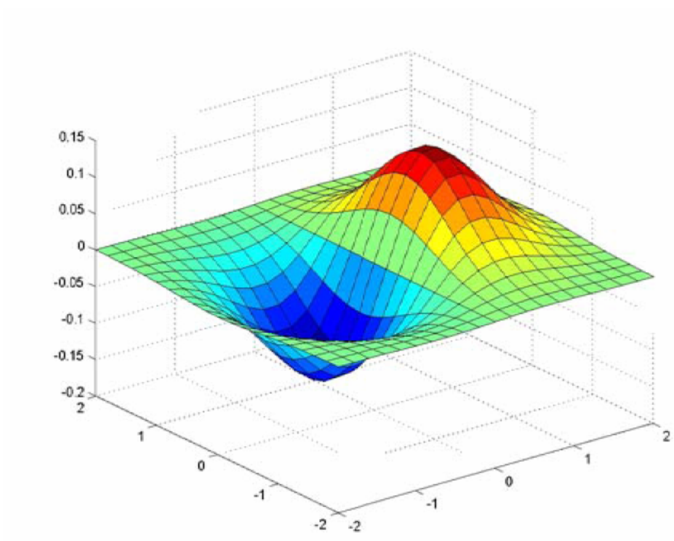
$\frac{d}{dx} g$

$f * \frac{d}{dx} g$

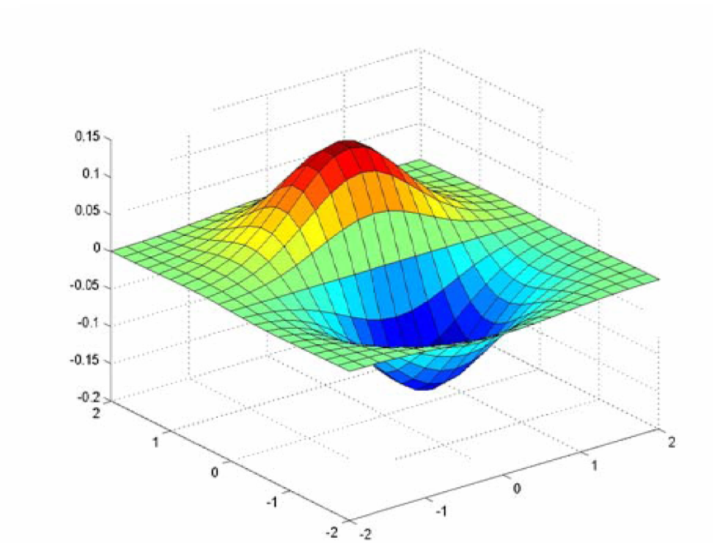


Source: S. Seitz

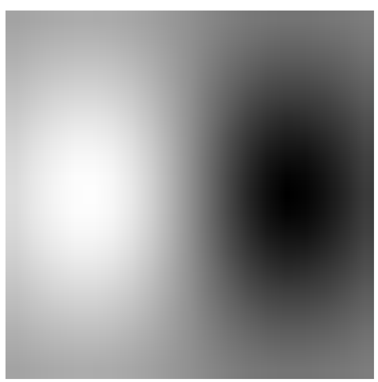
Derivative of Gaussian filters

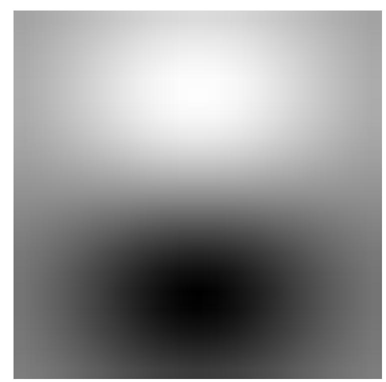


x-direction



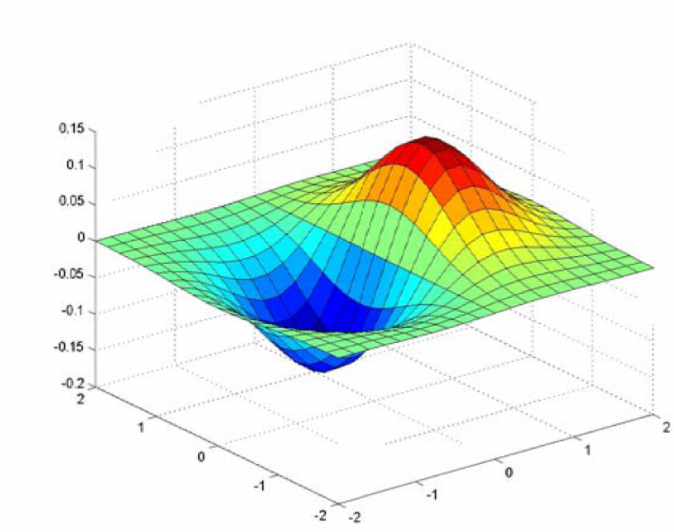
y-direction



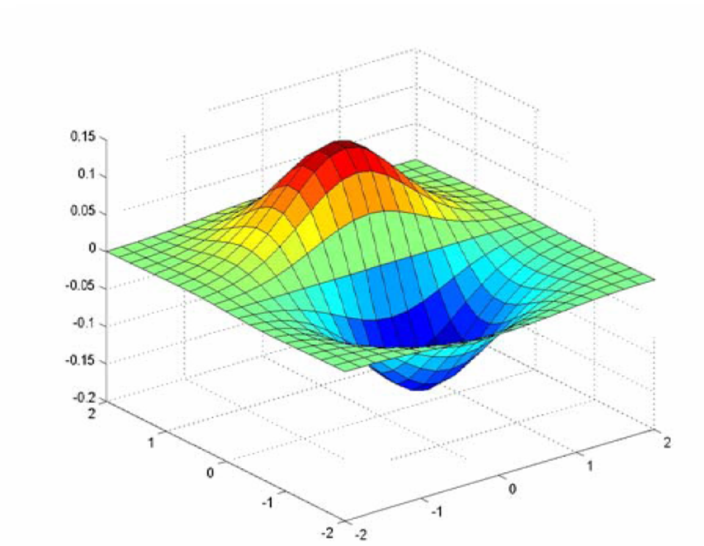


Which one finds horizontal/vertical edges?

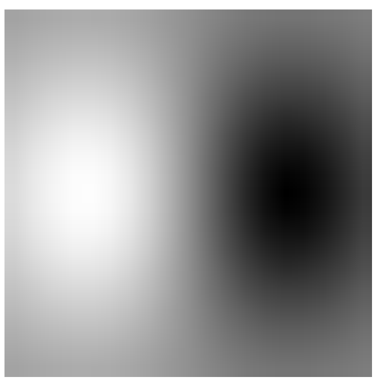
Derivative of Gaussian filters

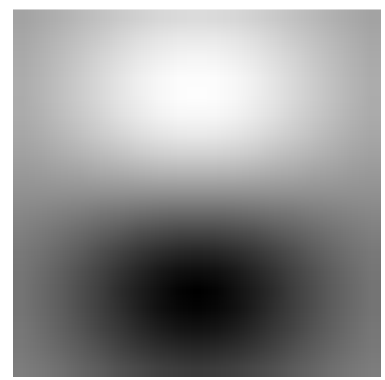


x-direction



y-direction





Are these filters separable?

Recall: Separability of the Gaussian filter

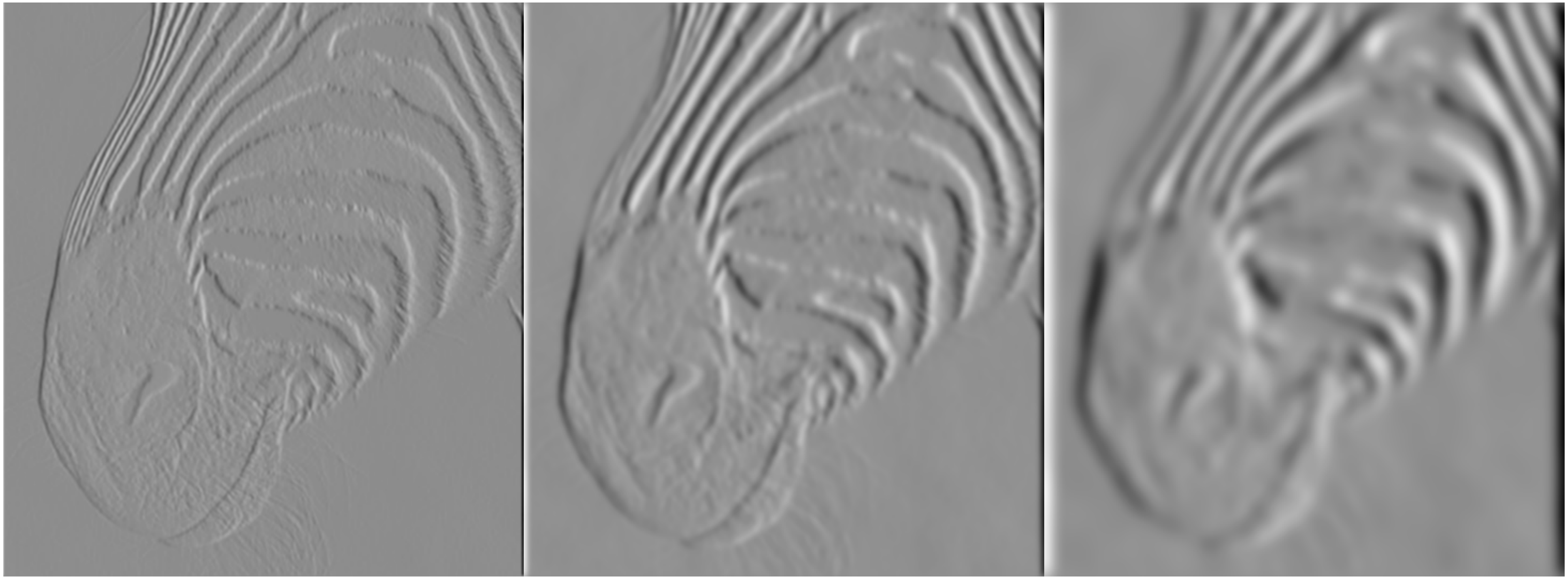
$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}}$$

$$= \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{x^2}{2\sigma^2}} \right) \left(\frac{1}{\sqrt{2\pi}\sigma} \exp^{-\frac{y^2}{2\sigma^2}} \right)$$

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

Scale of Gaussian derivative filter



1 pixel 3 pixels 7 pixels

Smoothed derivative removes noise, but blurs edge

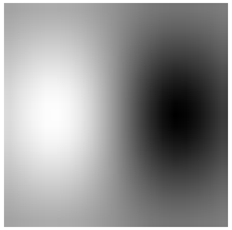
Also finds edges at different "scales"

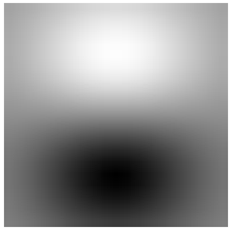
Review: Smoothing vs. derivative filters

Smoothing filters

- Gaussian: remove "high-frequency" components; "low-pass" filter
- Can the values of a smoothing filter be negative?
- What should the values sum to?
  - **One:** constant regions are not affected by the filter

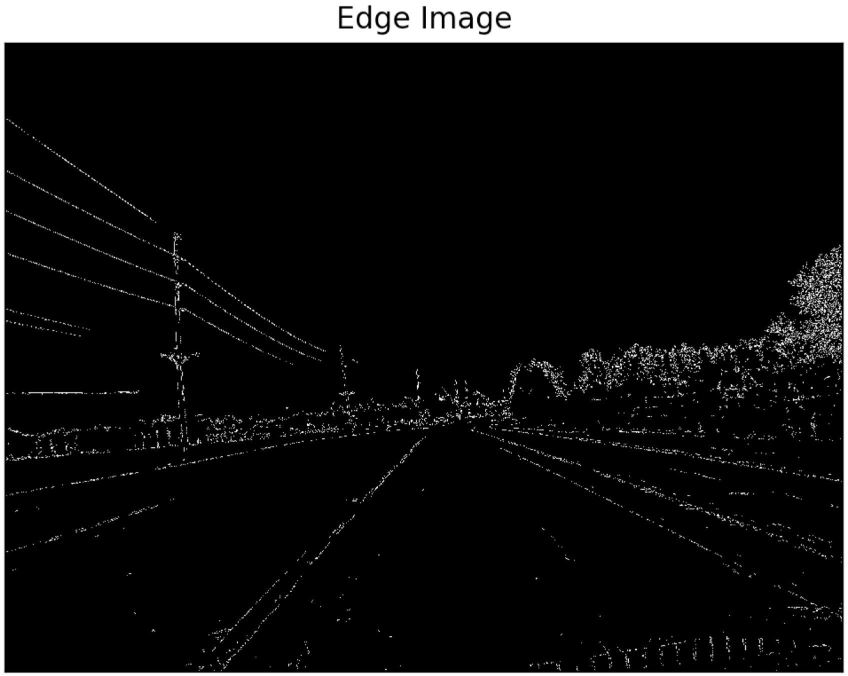
Derivative filters





- Derivatives of Gaussian
- Can the values of a derivative filter be negative?
- What should the values sum to?
  - **Zero:** no response in constant regions

Building an edge detector



original image

final output

norm of the gradient $\|\nabla f\| = \sqrt{(\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2}$

Building an edge detector

How to turn these thick regions of the gradient into curves?

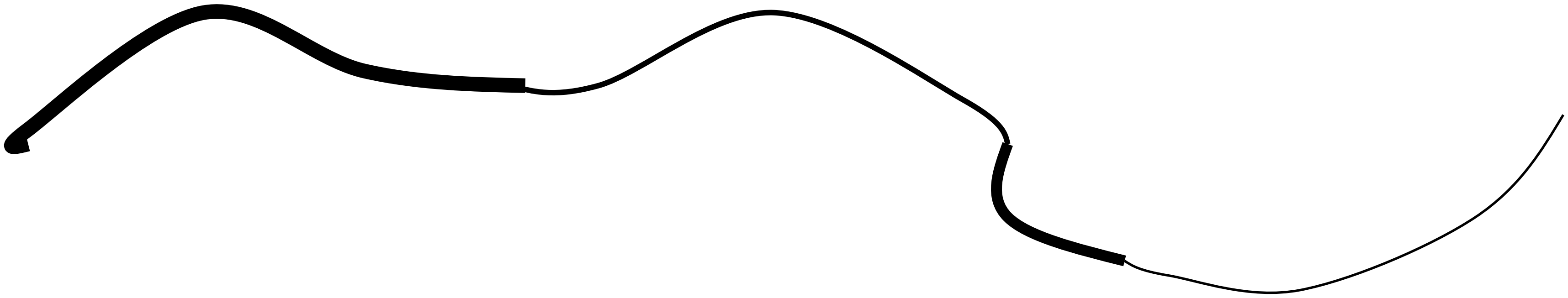
Thresholded norm of the gradient

Non-maximum suppression

Another problem: pixels along this edge didn't survive thresholding

Hysteresis thresholding

Use a high threshold to start edge curves, and a low threshold to continue them.



Source: Steve Seitz

Hysteresis thresholding

original image

high threshold (strong edges)

low threshold (weak edges)

hysteresis threshold

Source: L. Fei-Fei

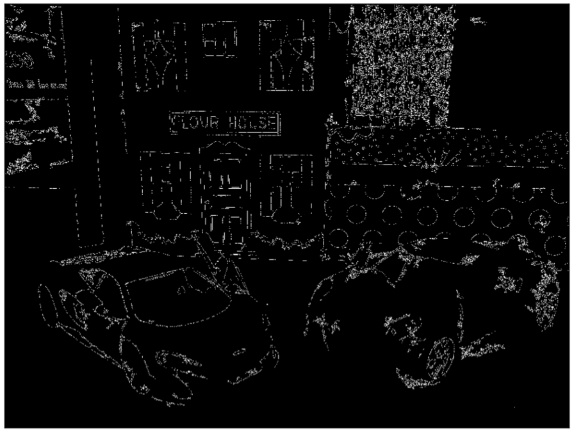
Recap: Canny edge detector

1. Compute x and y gradient images
2. Find magnitude and orientation of gradient
3. **Non-maximum suppression**:
   - Thin 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

opencv: `canny(image,th1,th2)`

Original Image

Edge Image



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