Vision and Perception

Lecture 1: Images and image filtering

SAPIENZA
UNIVERSITÀ DI ROMA

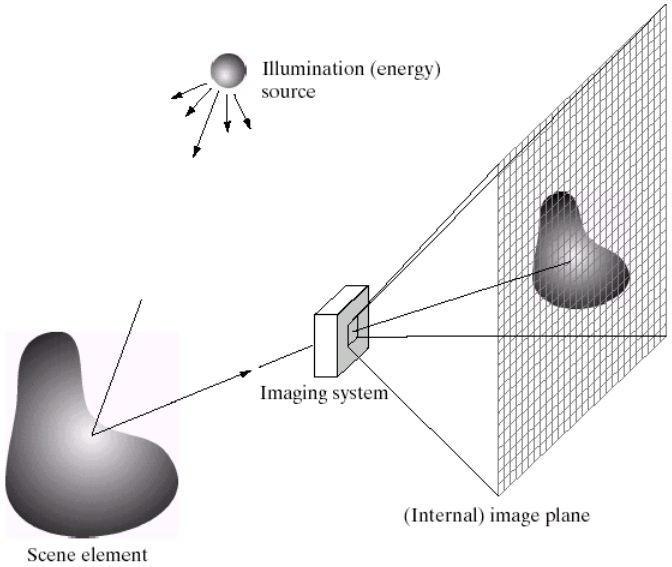
Reading

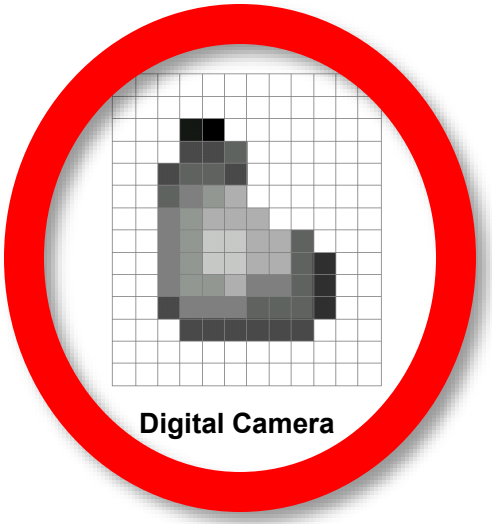
- Szeliski textbook, Chapter 3.1-3.2

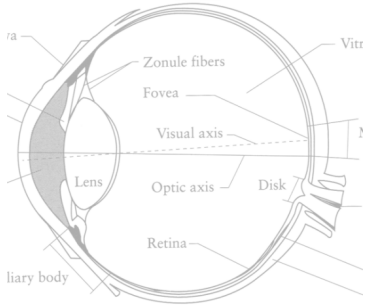
Overview of today's lecture

- Types of image transformations
- Point image processing
- Linear shift-invariant image filtering
- Convolution
- Image gradients

What is an image?







The Eye

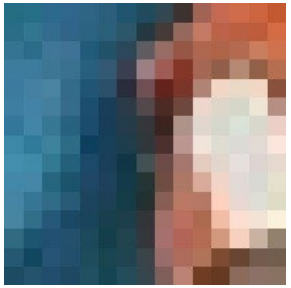
Source: A. Efros

What is an image?

What is an image?

A (color) image is a 3D tensor of numbers.

What is an image?

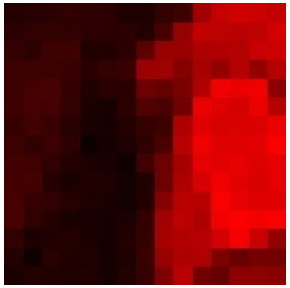


color image patch

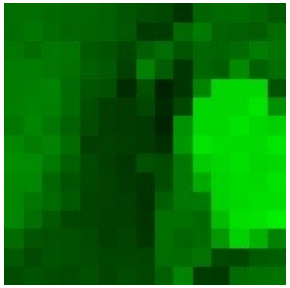
How many bits are the intensity values?

8 bit to represent 256 intensity values

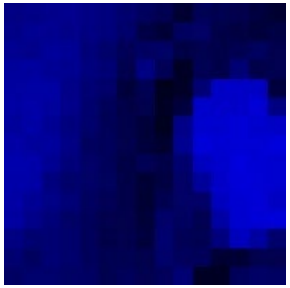
red



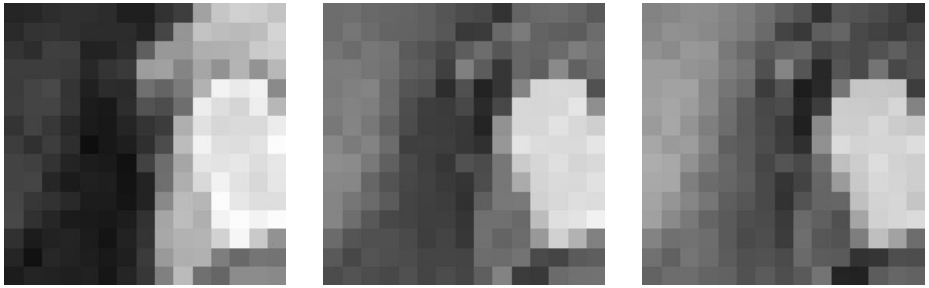
green



blue



colorized for visualization



actual intensity values per channel

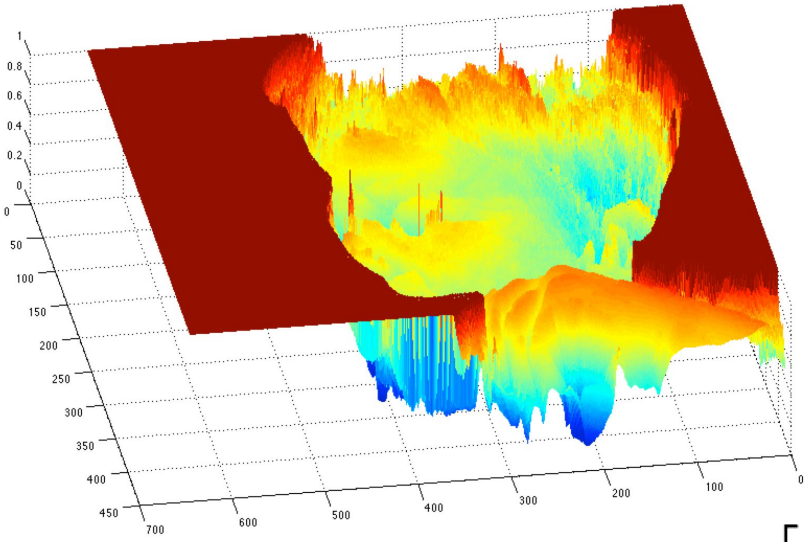
Each channel is a 2D array of numbers.

What is an image?

grayscale image

What is the range of the image function f?

$f(\boldsymbol{x})$

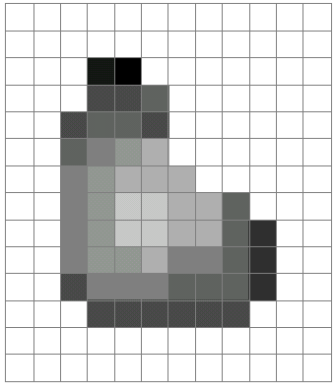


domain $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

A (grayscale) image is a 2D function.

What is an image?

- A grid (matrix) of intensity values



=

255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	20	0	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255
255	255	127	145	175	175	175	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255
255	255	127	145	200	200	175	175	95	47	255	255
255	255	127	145	145	175	127	127	95	47	255	255
255	255	74	127	127	127	95	95	95	47	255	255
255	255	255	74	74	74	74	74	74	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255

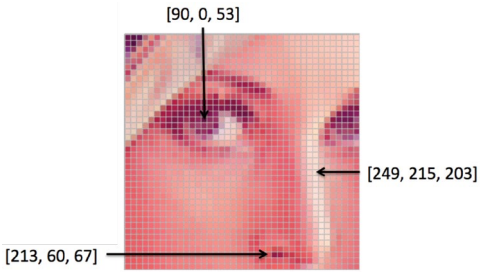
(common to use one byte per value: 0 = black, 255 = white)

Images as functions

- We can think of an **image** as a function, *f*, from $\mathbb{R}^2$ to $\mathbb{R}$:
  - $f(x, y)$ gives the **intensity** at position (x, y)
  - A **digital** image is a discrete (**sampled**, **quantized**) version of this function
  - Realistically, we expect the image only to be defined over a rectangle, with a finite range:
    - $f: [a,b]\text{x}[c,d] \rightarrow [0, 255]$

- A color image is just three functions pasted together.
  - We can write this as a "vector-valued" function:

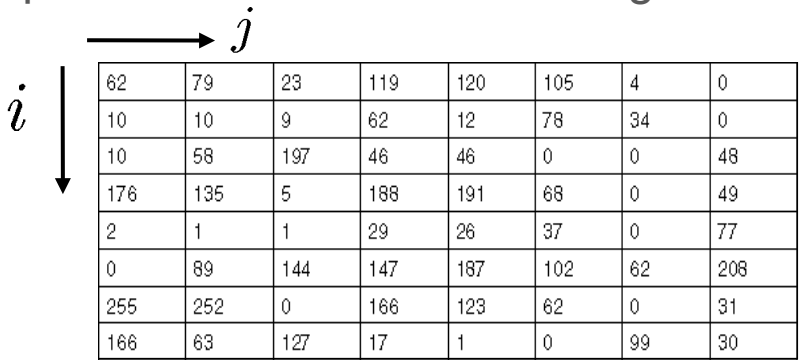
$$f(x,y) = \begin{bmatrix} r(x,y) \\ g(x,y) \\ b(x,y) \end{bmatrix}$$



Digital image

- We usually work with **digital** (**discrete**) images:
  - **Sample** the 2D space on a regular grid
  - **Quantize** each sample (round to nearest integer)
- If our samples are Δ apart, we can write this as:

  $f[i, j] = \text{Quantize}\{ f(i\,\Delta, j\,\Delta) \}$
- The image is represented as a matrix of integer values



62	79	23	119	120	105	4	0
10	10	9	62	12	78	34	0
10	58	197	46	46	0	0	48
176	135	5	188	191	68	0	49
2	1	1	29	26	37	0	77
0	89	144	147	187	102	62	208
255	252	0	166	123	62	0	31
166	63	127	17	1	0	99	30

Image processing

- What types of image transformations can we do?
  - image filtering: changes the *range* (i.e. the pixel values) of an image, so the colors of the image are altered without changing the pixel positions,
  - image warping: changes the *domain* (i.e. the pixel positions) of an image, where points are mapped to other points without changing the colors.

Image Filtering

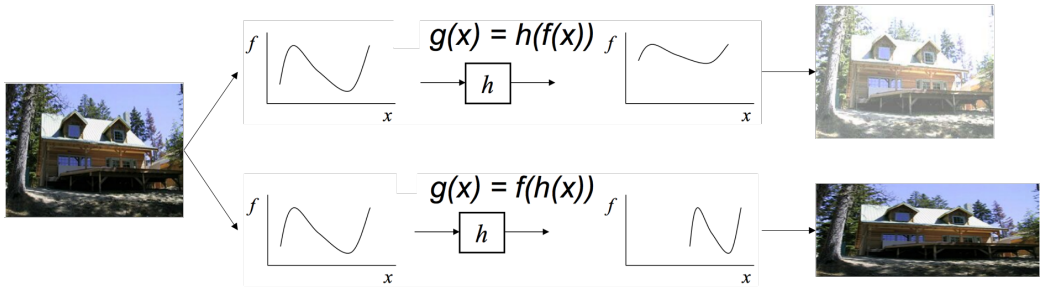


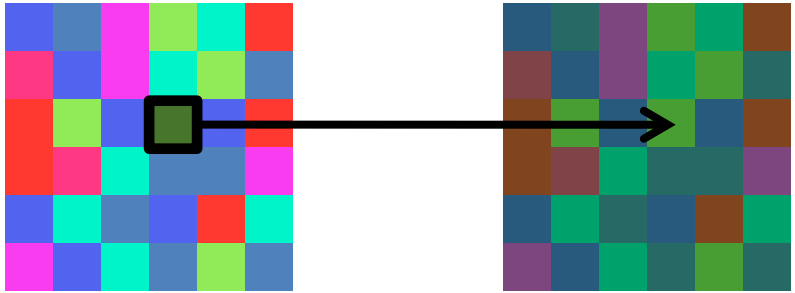
Image Warping

Canonical Image Processing problems

- Image Restoration
  - denoising
  - deblurring
- Image Compression
  - JPEG, JPEG2000, MPEG….
- Computing Field Properties
  - optical flow
- Locating Structural Features
  - corners
  - edges

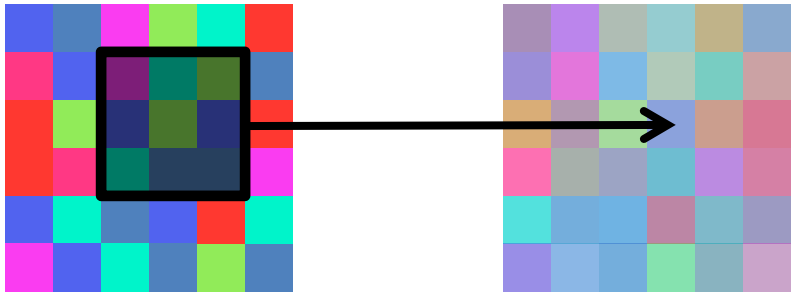
What types of image filtering can we do?

Point Operation



point processing

Neighborhood Operation



“filtering”

Point processing

Examples of point processing

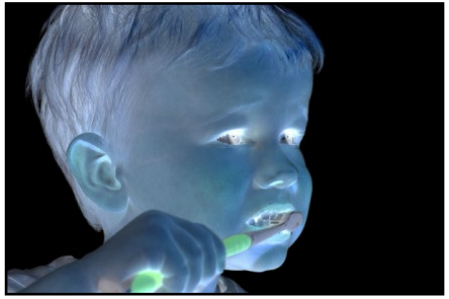
original

darken

lower contrast

non-linear lower contrast

invert



lighten

raise contrast

non-linear raise contrast

How would you implement these?

Examples of point processing

original

x

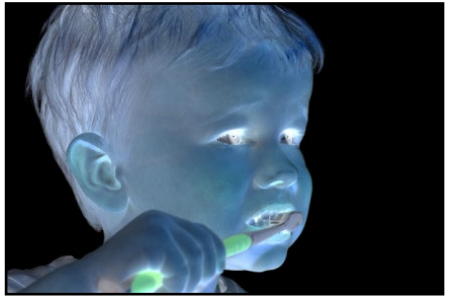
darken

$x - 128$

lower contrast

non-linear lower contrast

invert



lighten

raise contrast

non-linear raise contrast

How would you implement these?

Examples of point processing

original

x

darken

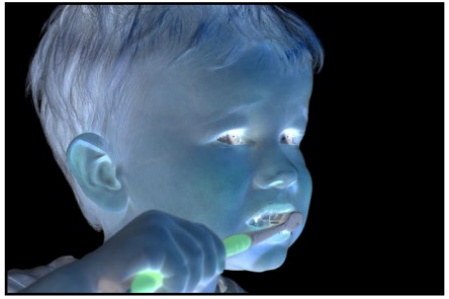
$x - 128$

lower contrast

$\frac{x}{2}$

non-linear lower contrast

invert



lighten

raise contrast

non-linear raise contrast

How would you implement these?

Examples of point processing

original

x

darken

$x - 128$

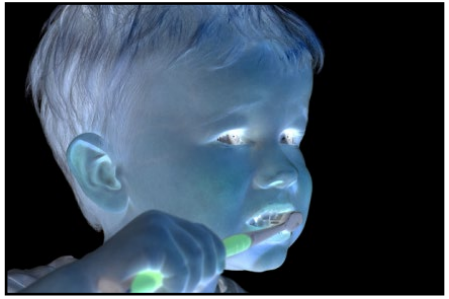
lower contrast

$\frac{x}{2}$

non-linear lower contrast

$\left(\frac{x}{255}\right)^{1/3} \times 255$

invert



lighten

raise contrast

non-linear raise contrast

How would you implement these?

Examples of point processing

original

x

darken

$x - 128$

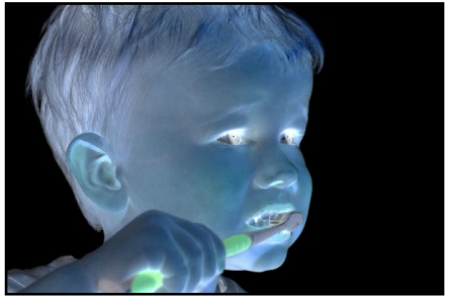
lower contrast

$\frac{x}{2}$

non-linear lower contrast

$\left(\frac{x}{255}\right)^{1/3} \times 255$

invert



$255 - x$

lighten

raise contrast

non-linear raise contrast

How would you implement these?

Examples of point processing

original

x

darken

$x - 128$

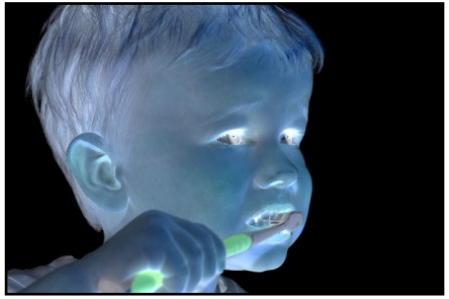
lower contrast

$\frac{x}{2}$

non-linear lower contrast

$\left(\frac{x}{255}\right)^{1/3} \times 255$

invert



$255 - x$

lighten

$x + 128$

raise contrast

non-linear raise contrast

How would you implement these?

Examples of point processing

original

x

darken

$x - 128$

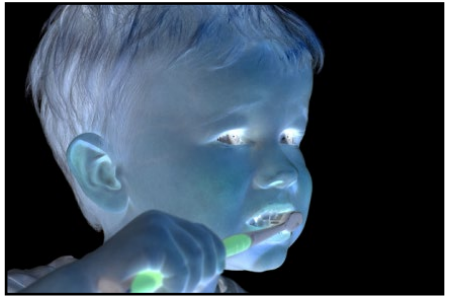
lower contrast

$\frac{x}{2}$

non-linear lower contrast

$\left(\frac{x}{255}\right)^{1/3} \times 255$

invert



$255 - x$

lighten

$x + 128$

raise contrast

$x \times 2$

non-linear raise contrast

How would you implement these?

Examples of point processing

original

x

darken

$x - 128$

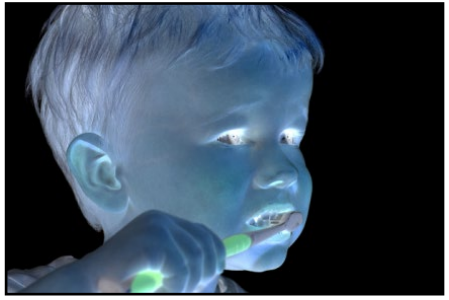
lower contrast

$\frac{x}{2}$

non-linear lower contrast

$\left(\frac{x}{255}\right)^{1/3} \times 255$

invert



$255 - x$

lighten

$x + 128$

raise contrast

$x \times 2$

non-linear raise contrast

$\left(\frac{x}{255}\right)^{2} \times 255$

Many other types of point processing

camera output

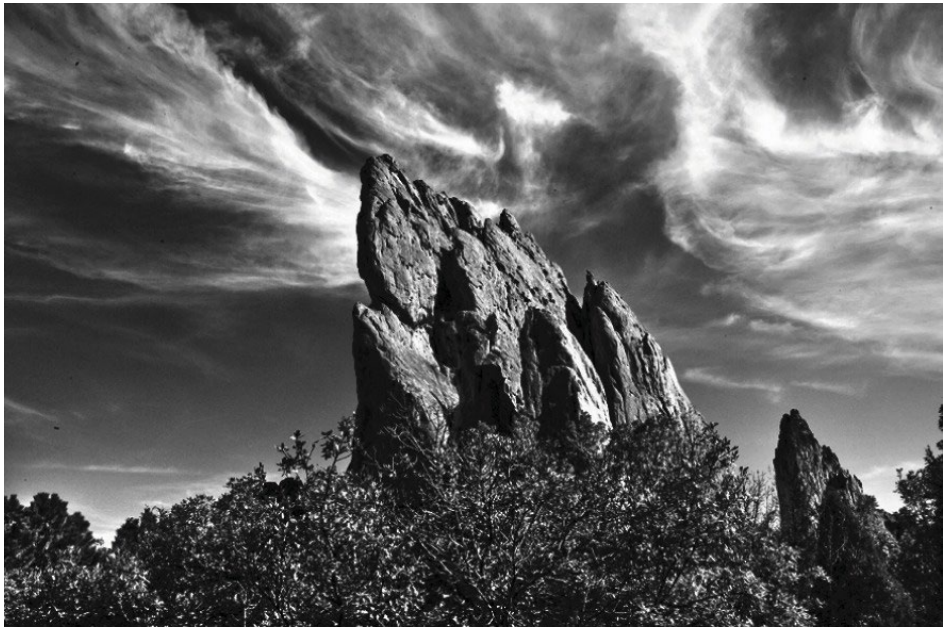
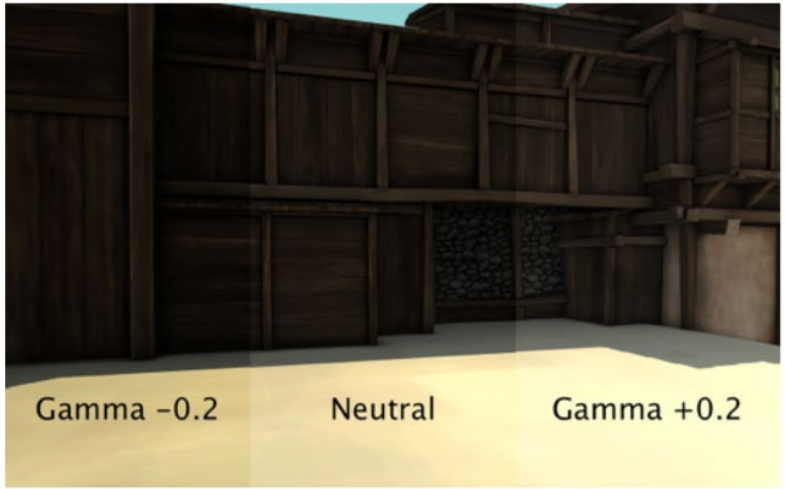


image after stylistic tonemapping

Vout =Vin/(Vin+1)
Vout= AVin^gamma ----> gamma compression A<0, 0<gamma<1

[Bae et al., SIGGRAPH 2006]

Many other types of point processing



Filtering

Form a new image whose pixels are a combination of the original pixels

- Why?
  - To get useful information from images
    - E.g., extract edges or contours (to understand shape)
  - To enhance the image
    - E.g., to remove noise
    - E.g., to sharpen and get better details (CSI mode ☺)

De-noising

Salt and pepper noise

Super-resolution

In-painting

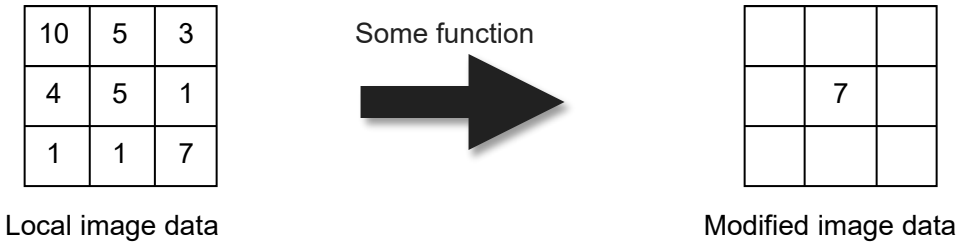
Linear shift-invariant image filtering

Linear shift-invariant image filtering

- Replace each pixel by a *linear* combination of its neighbors (and itself).
- The combination is determined by the filter's *kernel*.
- The same kernel is *shifted* to all pixel locations so that all pixels use the same linear combination of their neighbors.

Image filtering

- Modify the pixels in an image based on some function of a local neighborhood of each pixel



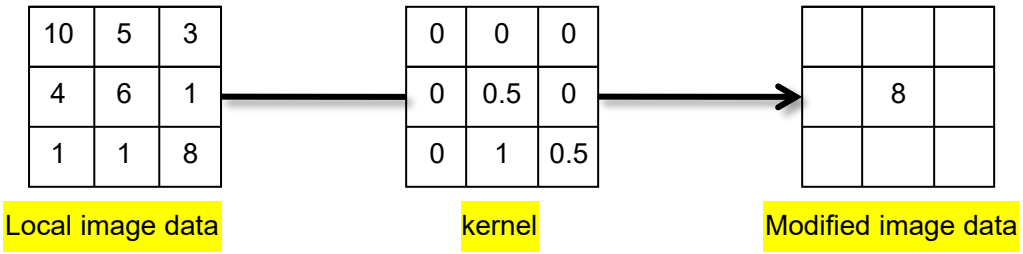
Local image data

Modified image data

Source: L. Zhang

Linear filtering

- The main operations are correlation and convolution
  - Replace each pixel by a linear combination (a weighted sum) of its neighbors
- The prescription for the linear combination is called the “kernel” (or “mask”, “filter”)



3+1+4=8

Source: L. Zhang

Example: the box filter

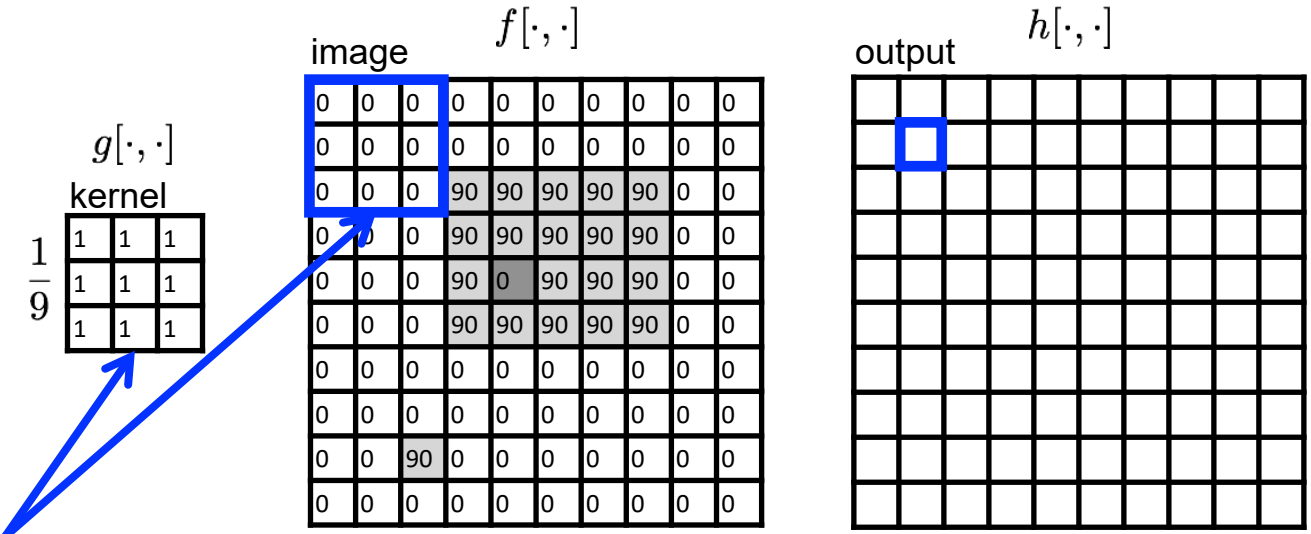
- also known as the 2D rect filter or square mean filter

kernel $g[\cdot,\cdot] = \frac{1}{9}$

1	1	1
1	1	1
1	1	1

- replaces pixel with local average
- has smoothing (blurring) effect

Let's run the box filter

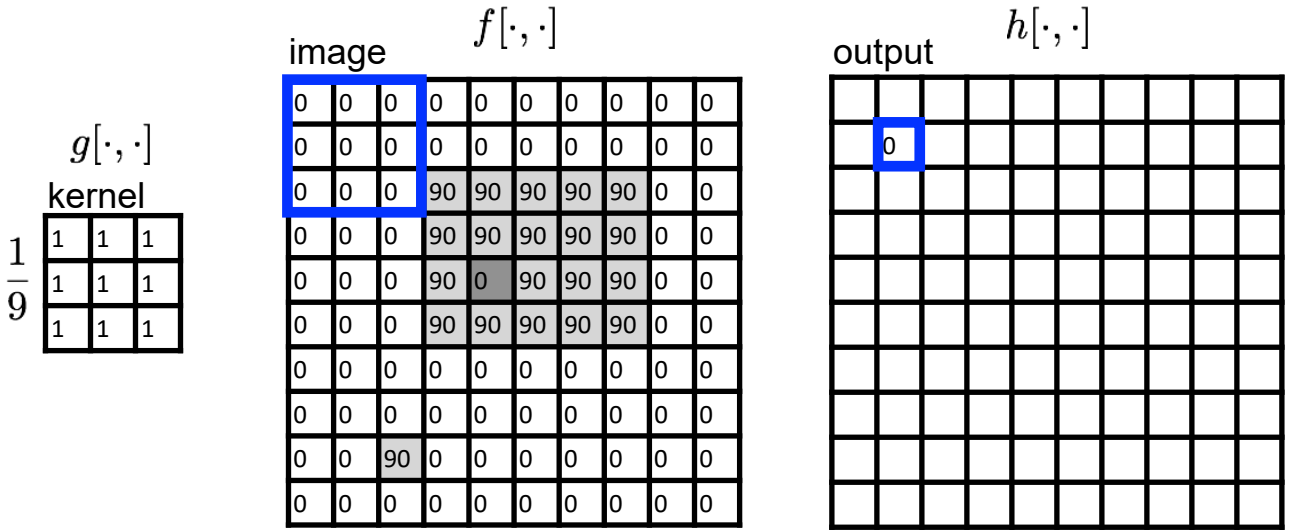


note that we assume that the kernel coordinates are centered

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

output filter image (signal)

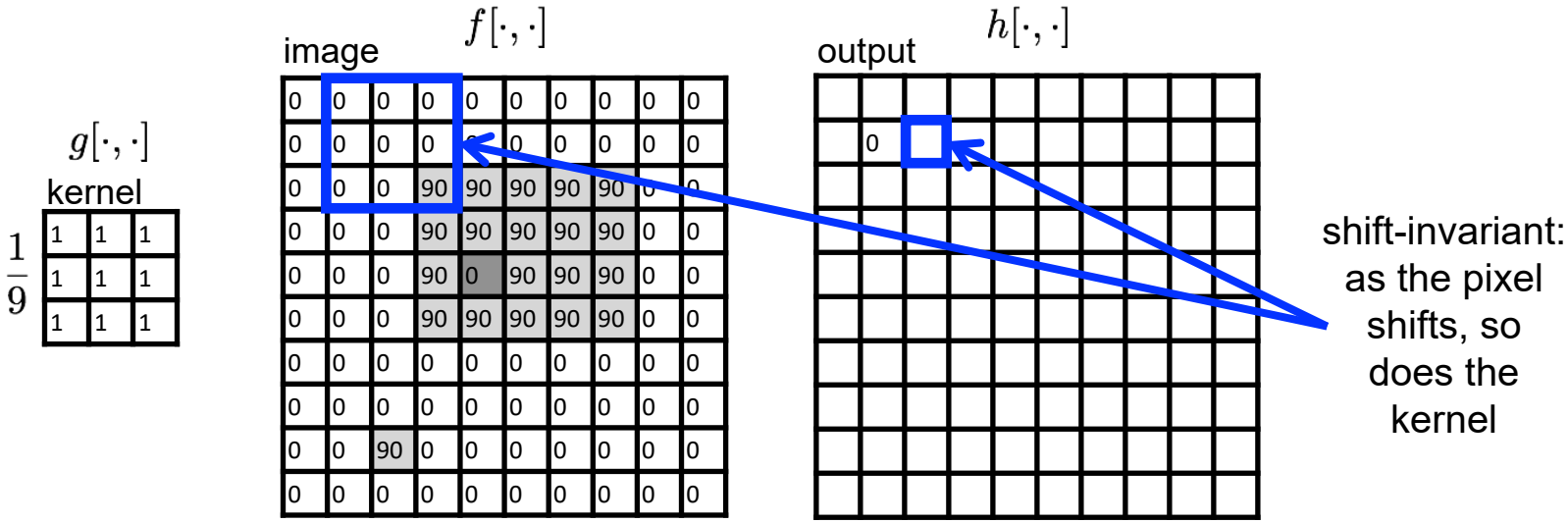
Let's run the box filter



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

output filter image (signal)

Let's run the box filter



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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

	0	10							

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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

	0	10							

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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

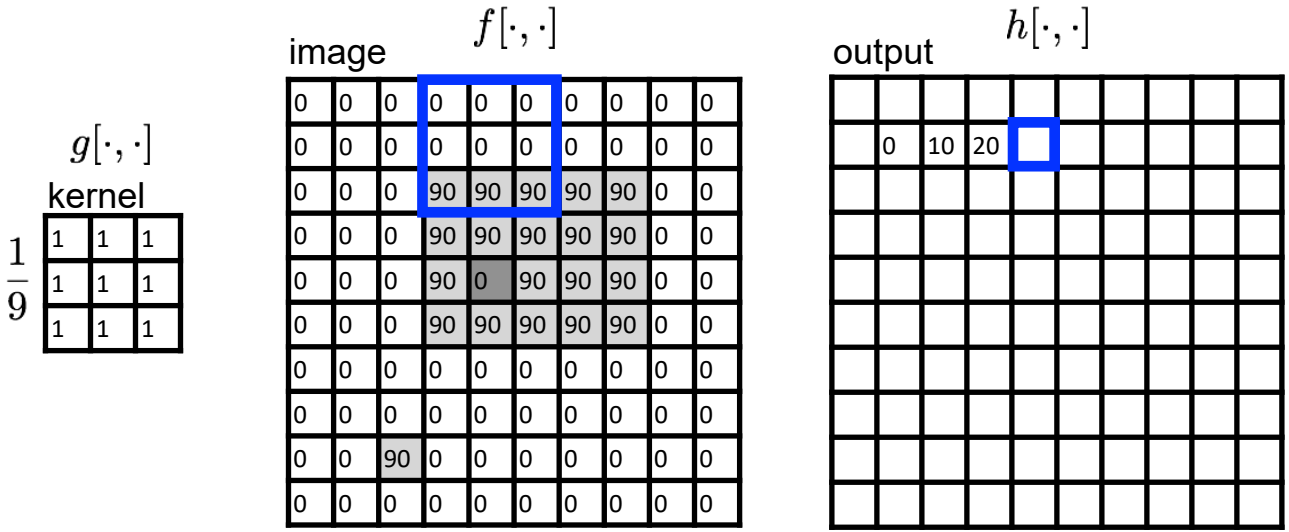
output

	0	10	20						

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

output filter image (signal)

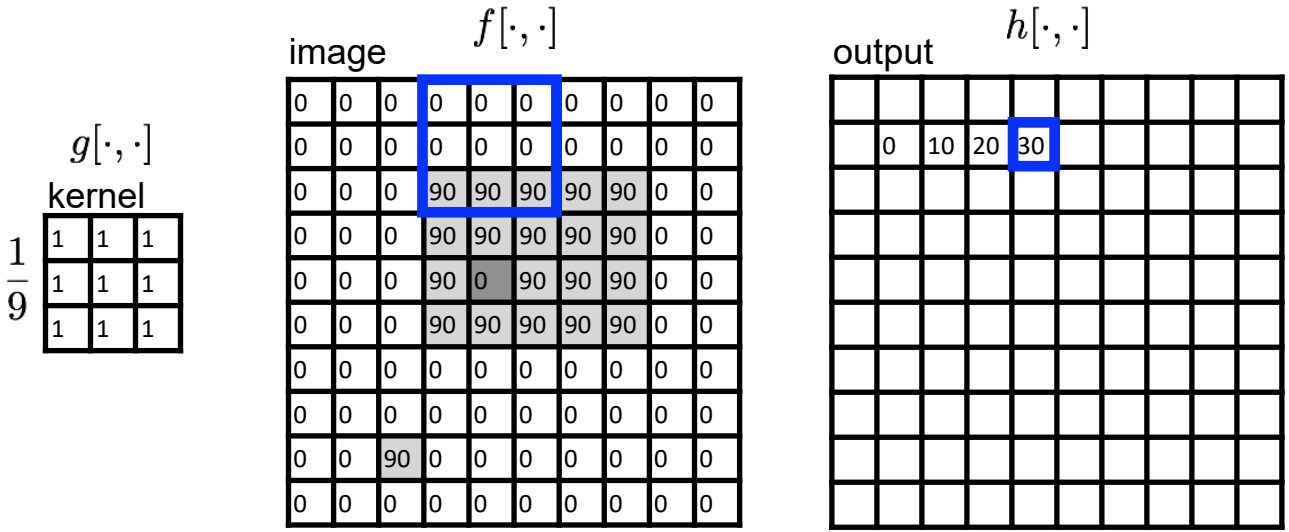
Let's run the box filter



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

output filter image (signal)

Let's run the box filter



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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

	0	10	20	30	30				

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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

	0	10	20	30	30	30			

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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

	0	10	20	30	30	30	20		

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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

0	10	20	30	30	30	20	10	

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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

	0	10	20	30	30	30	20	10	
	0								

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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	0	10	20	30	30	30	20	10	
	10	10	10	10	0	0	0	0	
	10								

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

output filter image (signal)

Let's run the box filter

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	0	10	20	30	30	30	20	10	
	10	10	10	10	0	0	0	0	
	10	10	10	10	0	0	0	0	

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

output filter image (signal)

… and the result is

$g[\cdot,\cdot]$

kernel

$\frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[\cdot,\cdot]$

image

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

$h[\cdot,\cdot]$

output

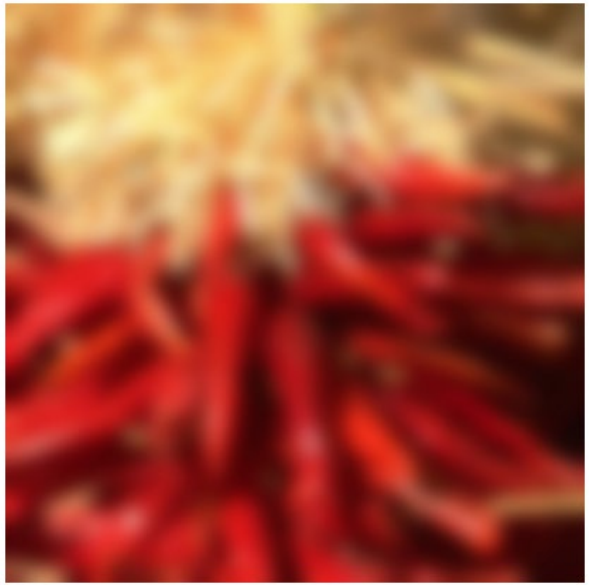
	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	0	10	20	30	30	30	20	10	
	10	10	10	10	0	0	0	0	
	10	10	10	10	0	0	0	0	

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

output filter image (signal)

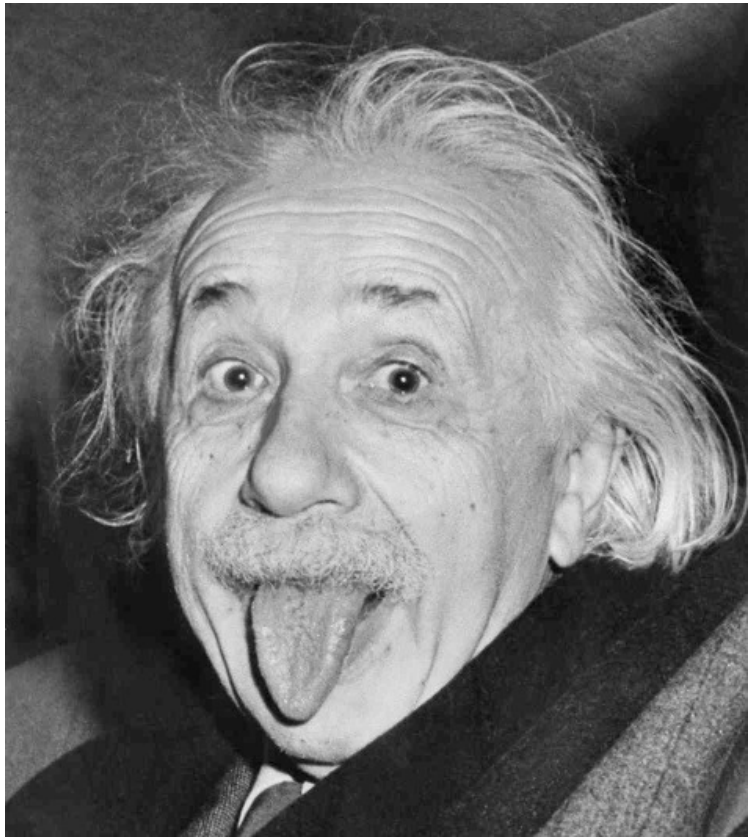
Practical matters

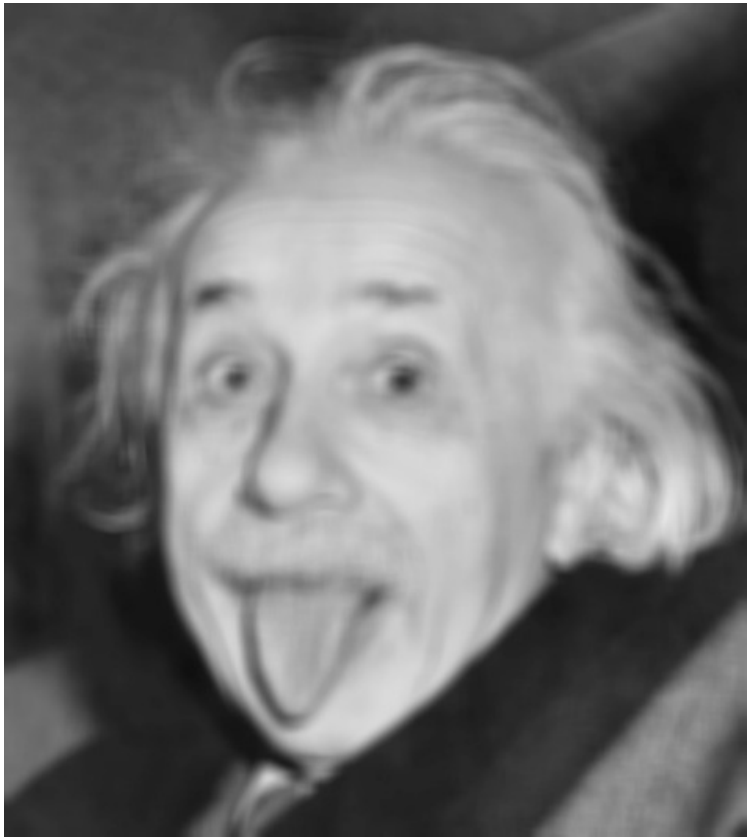
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - Methods (padding):
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge



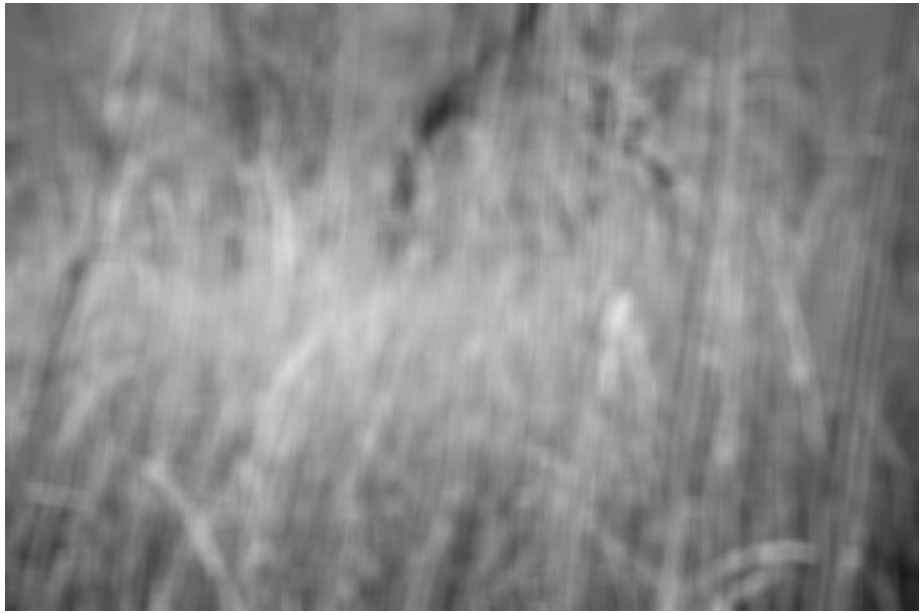
Source: S. Marschner

Some more realistic examples





Some more realistic examples



Some more realistic examples

Convolution

Convolution for 1D continuous signals

Definition of filtering as convolution:

$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)dy$$

filtered signal ↗ $(f * g)(x)$ $= \int_{-\infty}^{\infty} f(y)g(x-y)dy$ ↖ filter ↖ input signal

Convolution for 1D continuous signals

Definition of filtering as convolution:

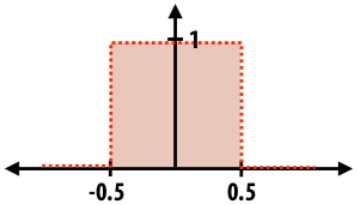
$$(f * g)(x) = \int_{-\infty}^{\infty} f(y)g(x-y)dy$$

filtered signal ← $(f * g)(x)$; $f(y)$ ← filter ; $g(x-y)$ ← input signal

Consider the box filter example:

1D continuous box filter

$$f(x) = \begin{cases} 1 & |x| \leq 0.5 \\ 0 & otherwise \end{cases}$$



filtering output is a blurred version of g

$$(f * g)(x) = \int_{-0.5}^{0.5} g(x-y)dy$$

Convolution for 2D discrete signals

Definition of filtering as convolution:

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j) I(x - i, y - j)$$

filtered image ← $(f * g)(x, y)$; $f(i, j)$ ← filter ; $I(x - i, y - j)$ ← input image

Convolution for 2D discrete signals

Definition of filtering as convolution:

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j) I(x - i, y - j)$$

filtered image ← $(f * g)(x, y)$; filter ← $f(i, j)$; input image ← $I(x - i, y - j)$

If the filter $f(i, j)$ is non-zero only within $-1 \leq i, j \leq 1$, then

$$(f * g)(x, y) = \sum_{i,j=-1}^{1} f(i, j) I(x - i, y - j)$$

The kernel we saw earlier is the 3x3 matrix representation of $f(i, j)$

Convolution vs correlation

Definition of discrete 2D convolution:

notice the flip

$$(f * g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j)I(x - i, y - j)$$

the patch is “flipped” (horizontally and vertically)

Definition of discrete 2D correlation:

notice the lack of a flip

$$(f \otimes g)(x, y) = \sum_{i,j=-\infty}^{\infty} f(i, j)I(x + i, y + j)$$

- Most of the time won’t matter, because our kernels will be symmetric.
- Will be important when we discuss frequency-domain filtering.
- While a convolution is a filtering operation, correlation measures the similarity of two signals, comparing them as they are shifted by one another. When two signals match, the correlation result is maximized.

Convolution is nice!

- Notation: $b = c \star a$
- Convolution is a multiplication-like operation
  - commutative $a \star b = b \star a$
  - associative $a \star (b \star c) = (a \star b) \star c$
  - distributes over addition $a \star (b + c) = a \star b + a \star c$
  - scalars factor out $\alpha a \star b = a \star \alpha b = \alpha(a \star b)$
  - Identity unit impulse *e* = […, 0, 0, 1, 0, 0, …] $a \star e = a$
- Usefulness of associativity
  - 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)

2D signal and convolution

- 2D convolution (discrete): $f[m,n] = I \otimes g = \sum_{k,l} I[m-k, n-l]g[k,l]$
  - discrete Image: *I[m,n]*
  - filter 'kernel': *g[k,l]*
  - 'filtered' image: *f[m,n]*

$f[m,n]$ | $I[k,l]$ | $g[k,l]$

	18	

$=$

8	5	2
7	5	3
9	4	1

$\otimes$

−1	0	1
−1	0	1
−1	0	1

- special case:
  - convolution (discrete) of a 2D-image with a 1D-filter

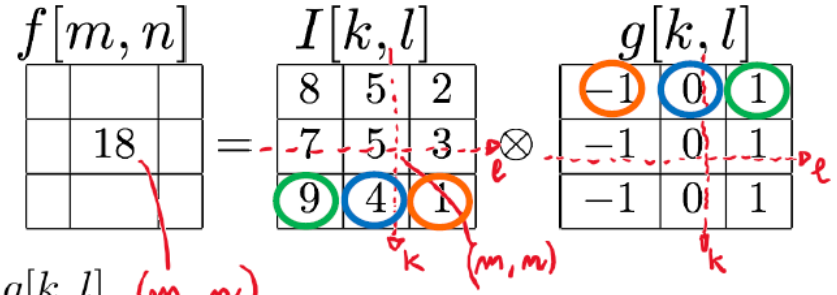
$$f[m,n] = I \otimes g = \sum_{k} I[m-k, n]g[k]$$

$g[k]$

-1
0
1

Example: 2D signal and convolution

- 2D convolution (discrete): $f[m,n] = I \otimes g = \sum_{k,l} I[m-k, n-l]g[k,l]$
  - discrete Image: *I[m,n]*
  - filter 'kernel': *g[k,l]*
  - 'filtered' image: *f[m,n]*



$$= \sum_{\substack{-1<k<+1 \\ -1<l<+1}} I[m-k, n-l]g[k,l] \quad (m, n)$$

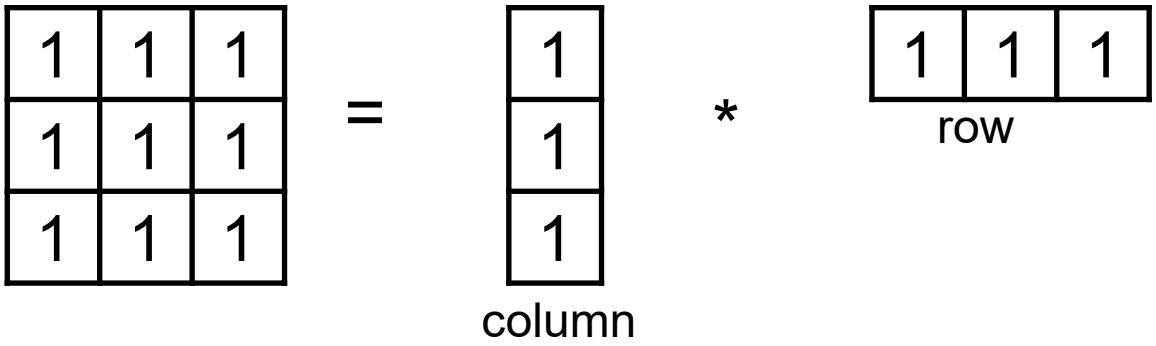
- mirror the filter (k and l)
- swipe it across the image
- multiply and sum

$$= I[m+1, n+1]g[-1,-1] \qquad (k=-1, l=-1)$$
$$+ I[m+1, n]g[-1,0] \qquad (k=-1, l=0)$$
$$+ I[m+1, n-1]g[-1,+1] \qquad (k=-1, l=+1)$$
$$+ \ldots$$

Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

example:
box filter



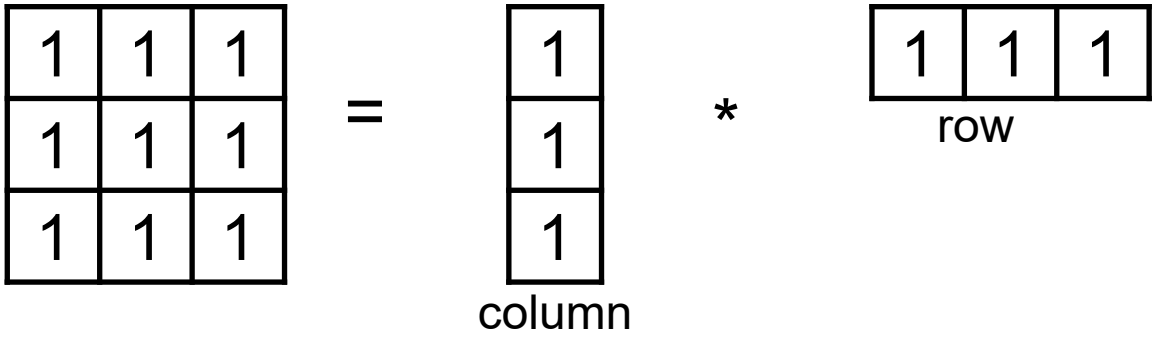
2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

We want that the kernal has the property to be saparable: in such a way the kernel that is applied /convolved to the source image is barely a 1-D filter
A 2D function is separable if it is formed by the product of two 1D functions

Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

example:
box filter



2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

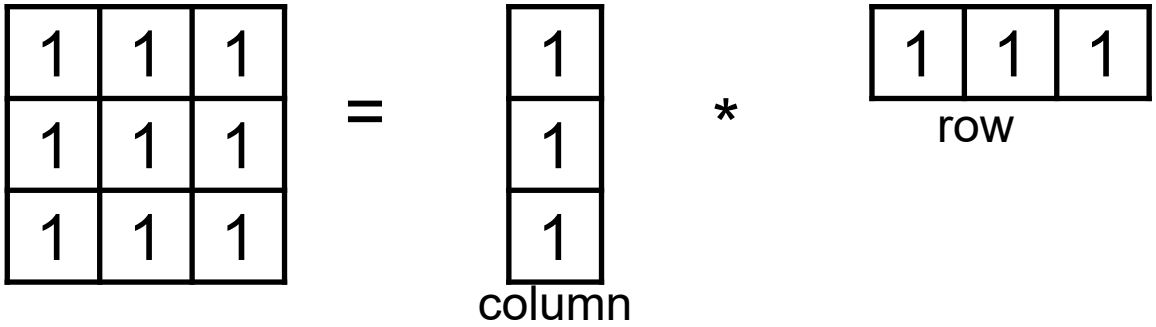
If the image has M x N pixels and the filter kernel has size L x L:

- What is the cost of convolution with a non-separable filter?

Separable filters

A 2D filter is separable if it can be written as the product of a “column” and a “row”.

example:
box filter



2D convolution with a separable filter is equivalent to two 1D convolutions (with the “column” and “row” filters).

If the image has M x N pixels and the filter kernel has size L x L:

- What is the cost of convolution with a non-separable filter? → L^2 x MxN
- What is the cost of convolution with a separable filter? → 2 x (L x Mx N)

A few more filters

original

3x3 box filter

do you see any problems in this image?

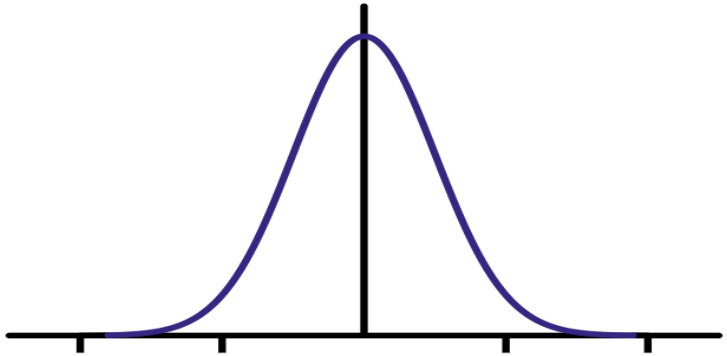
The box filter equally weights all samples within a square region of the image. Although computationally efficient, it's just about the worst filter possible.

Discontinuous result even though the original function was smooth

The Gaussian filter

- named (like many other things) after Carl Friedrich Gauss (1755-1855)
- kernel values sampled from the 2D Gaussian function:

$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$



- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance

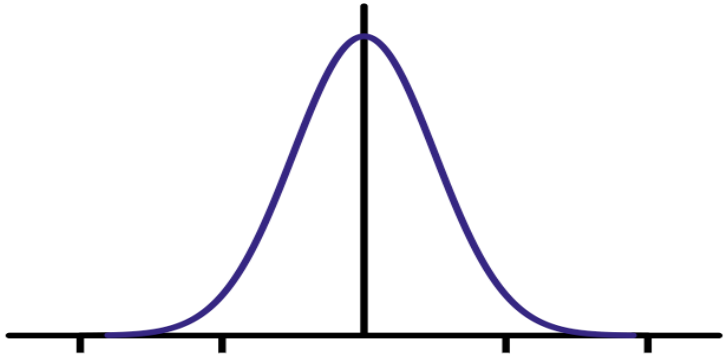
Any heuristics for selecting where to truncate?

The Gaussian filter

- named (like many other things) after Carl Friedrich Gauss

- kernel values sampled from the 2D Gaussian function:

$$f(i,j) = \frac{1}{2\pi\sigma^2} e^{-\frac{i^2+j^2}{2\sigma^2}}$$



Is this a separable filter? Yes!

- weight falls off with distance from center pixel
- theoretically infinite, in practice truncated to some maximum distance

Any heuristics for selecting where to truncate?

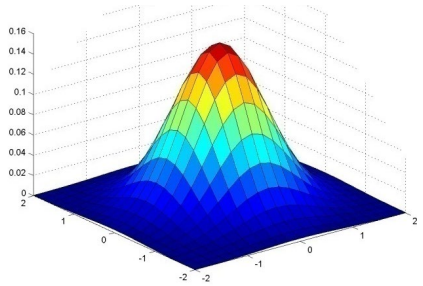
kernel $\frac{1}{16}$

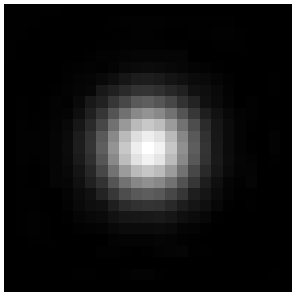
1	2	1
2	4	2
1	2	1

- usually at 2σ or 3σ (sigma standard deviation of a filter can be interpreted as a measure of its size, width of the gaussian function)

The Gaussian filter

- Weight contributions of neighboring pixels by nearness





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

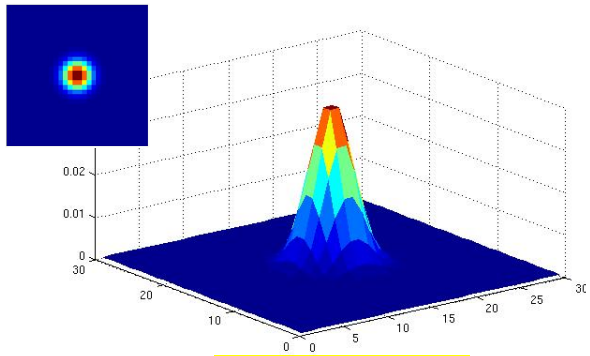
5 x 5, $\sigma = 1$

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

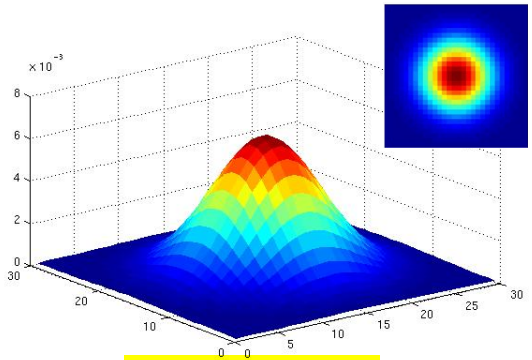
Slide credit: Christopher Rasmussen

The Gaussian filter: example

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



σ = 2 with 30 x 30 kernel

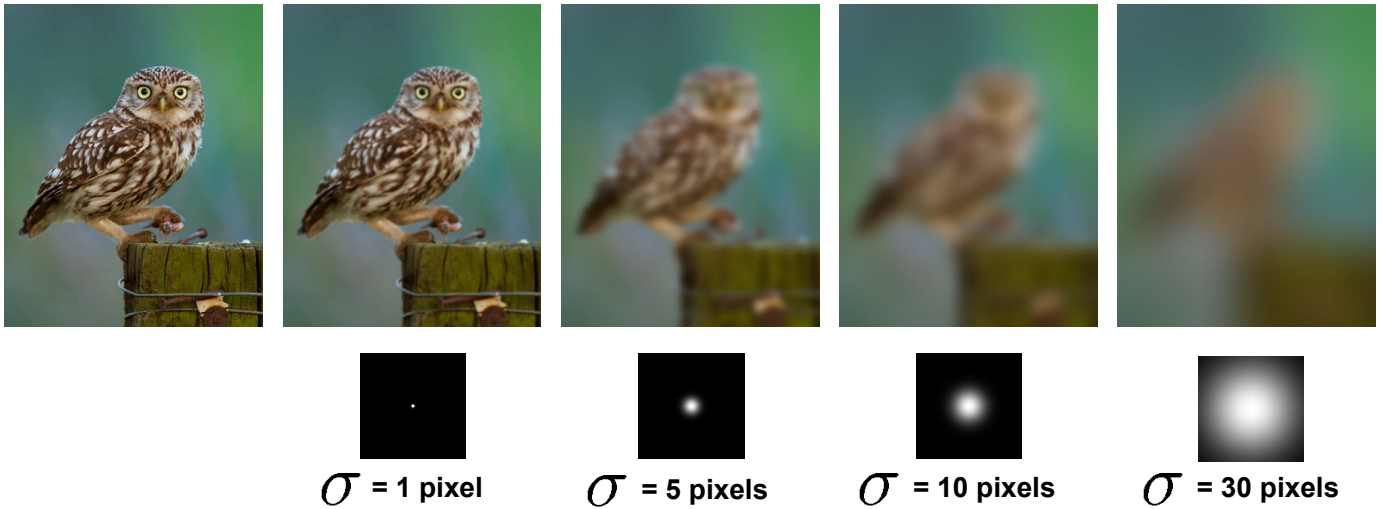


σ = 5 with 30 x 30 kernel

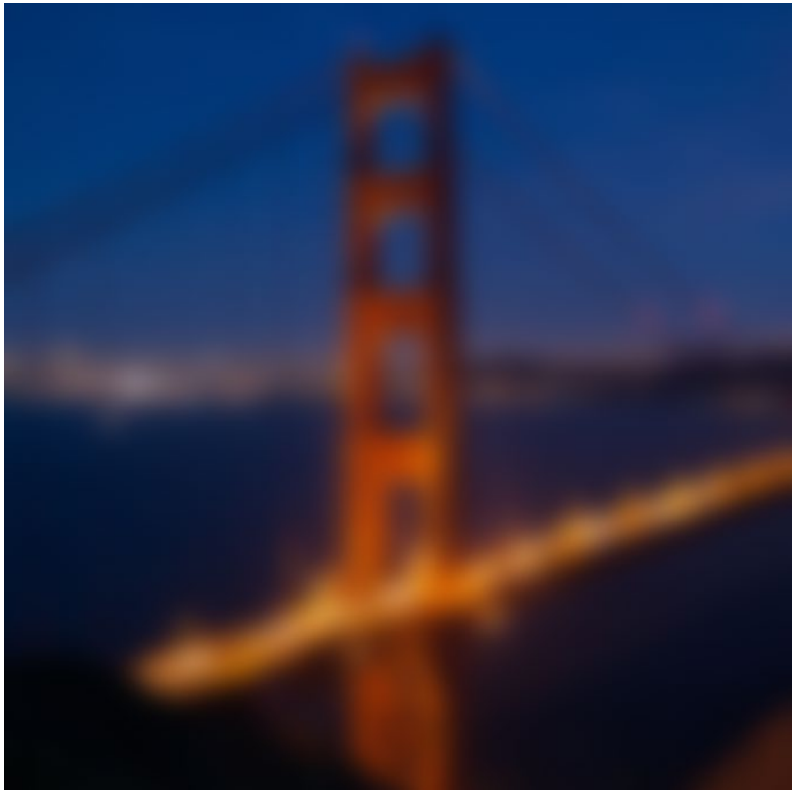
- Standard deviation σ: determines extent of smoothing

Source: K. Grauman

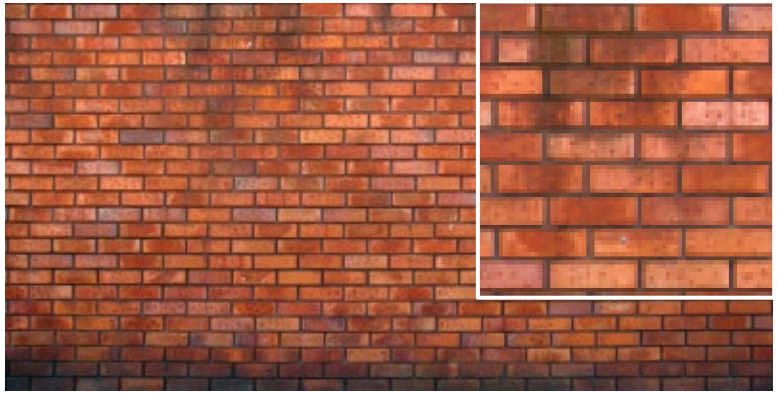
Gaussian filters



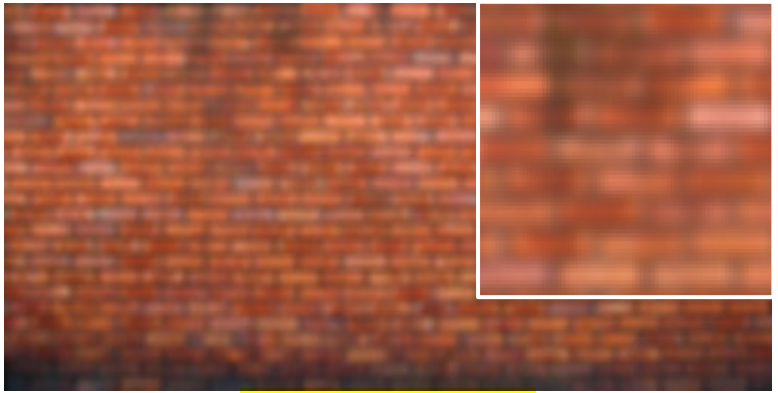
Gaussian filtering example



Gaussian vs box filtering



original



7x7 Gaussian

7x7 box

Which blur do you like better?

Other filters

input

filter

0	0	0
0	1	0
0	0	0

output

?

Other filters

input

filter

0	0	0
0	1	0
0	0	0

output

unchanged

Other filters

input

filter

0	0	0
0	1	0
0	0	0

output

unchanged

input

filter

0	0	0
0	0	1
0	0	0

output

?

Other filters

input

filter

0	0	0
0	1	0
0	0	0

output

unchanged

input

filter

0	0	0
0	0	1
0	0	0

output

shift to left by one

Other filters

input

filter

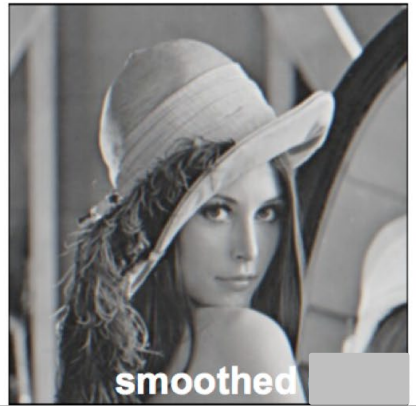
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

output

?

Step 1 -smoothing

–



=

•0	•0	•0
•0	•1	•0
•0	•0	•0

–

$\frac{1}{9}$

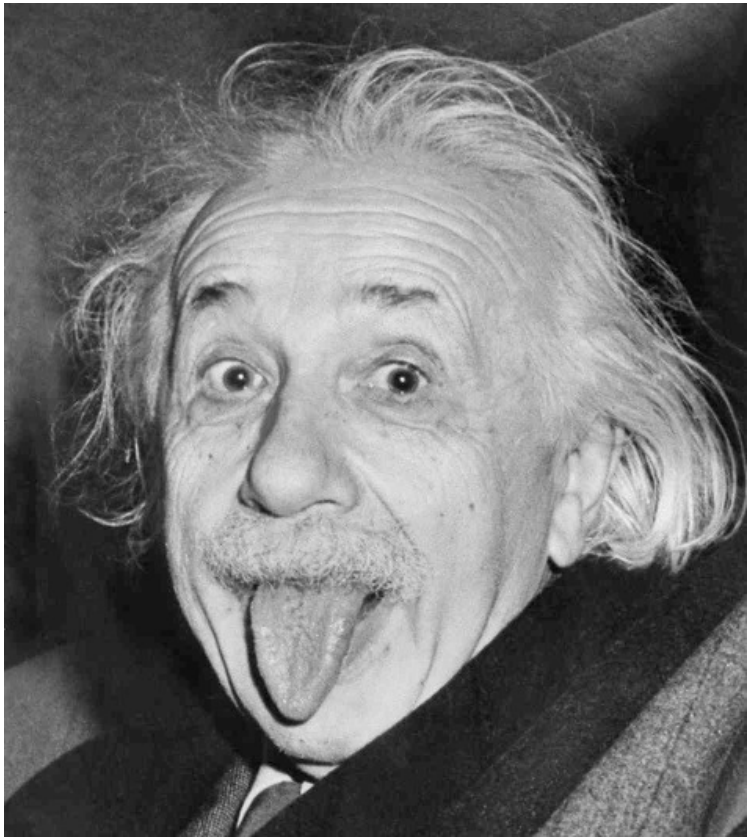
•1	•1	•1
•1	•1	•1
•1	•1	•1

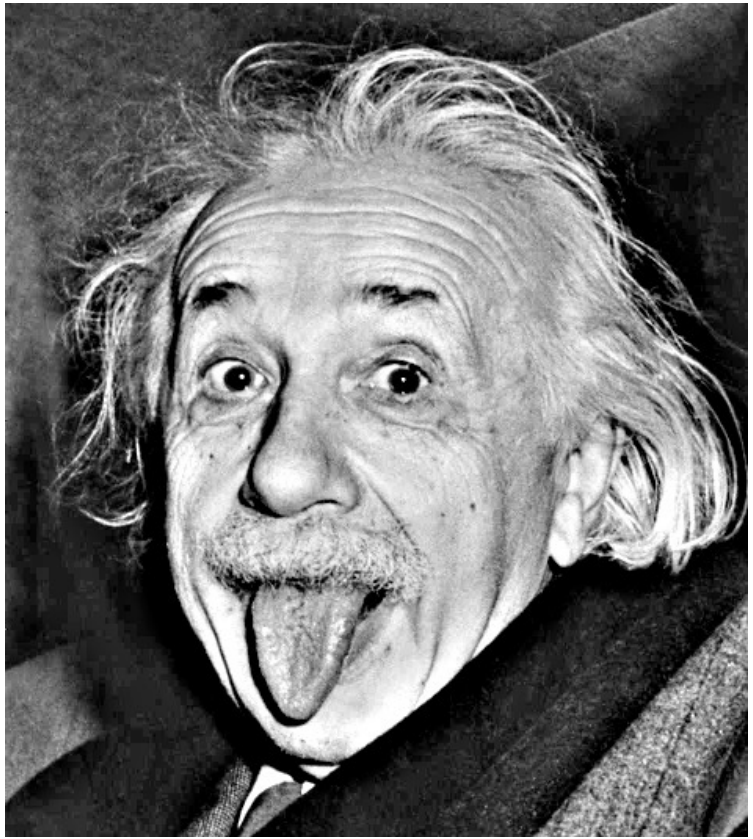
Blur (with a mean filter)

Step 2 - sharpening

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

Sharpening examples





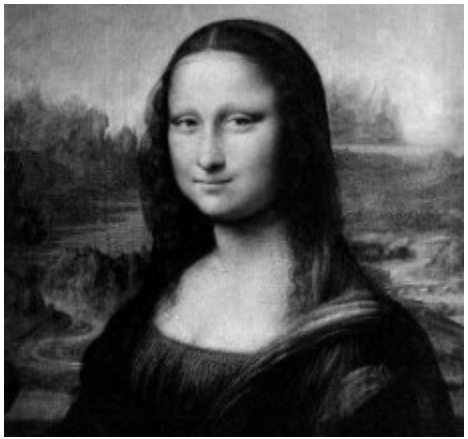
- do nothing for flat areas
- stress intensity peaks

Sharpening examples

Sharpening examples

Sharpening examples

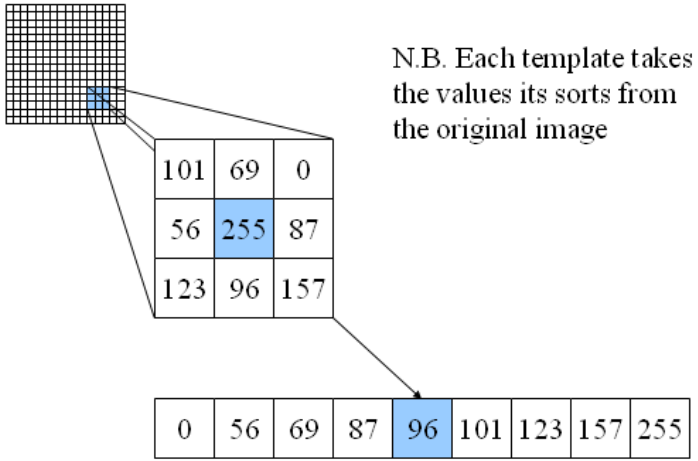
Filters: Thresholding



$$g(m,n) = \begin{cases} 255, & f(m,n) > A \\ 0 & otherwise \end{cases}$$

Non-Linear filters

- Is thresholding a linear filter?
- No is a non-linear filter together with image equalization or median filters
- Median filters:

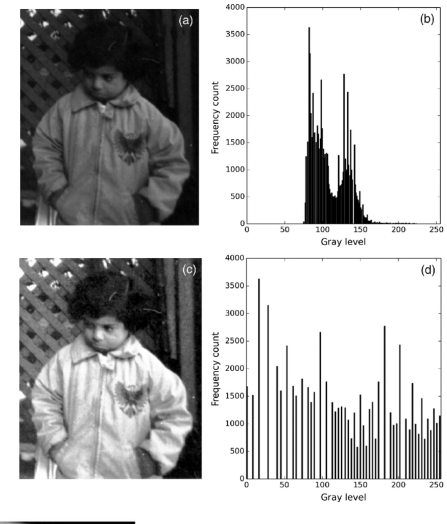


Salt and pepper noise

Image histogram

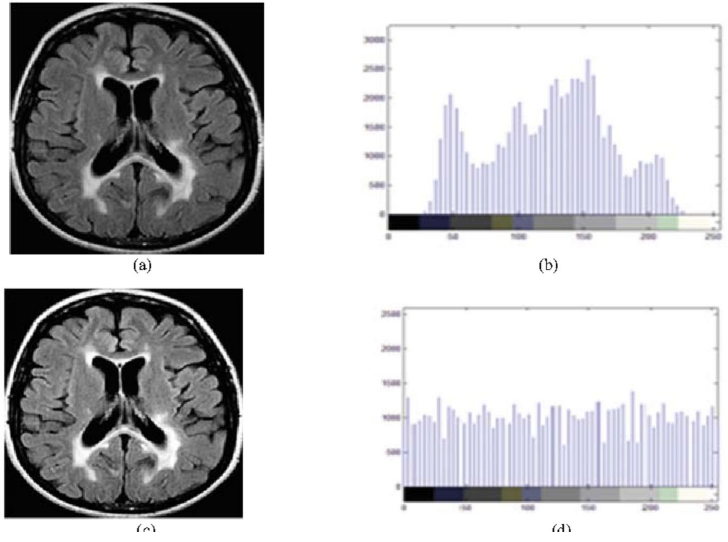
Intensity histogram:

The image is scanned in a single pass and a count of the number of pixels found at each intensity value is kept.
This is then used to construct a suitable histogram.



Histogram equalization:

The idea is that the pixels should be distributed evenly over the whole intensity range, i.e. the aim is to transform the image so that the output image has a flat histogram.



More on filters

- More sophisticated filtering techniques can often yield superior results for these and other tasks:
  - Polynomial (e.g., bicubic) filters
  - Steerable filters
  - Bilateral Filters
  - …
- (see text, web for more details on these)

Image gradients

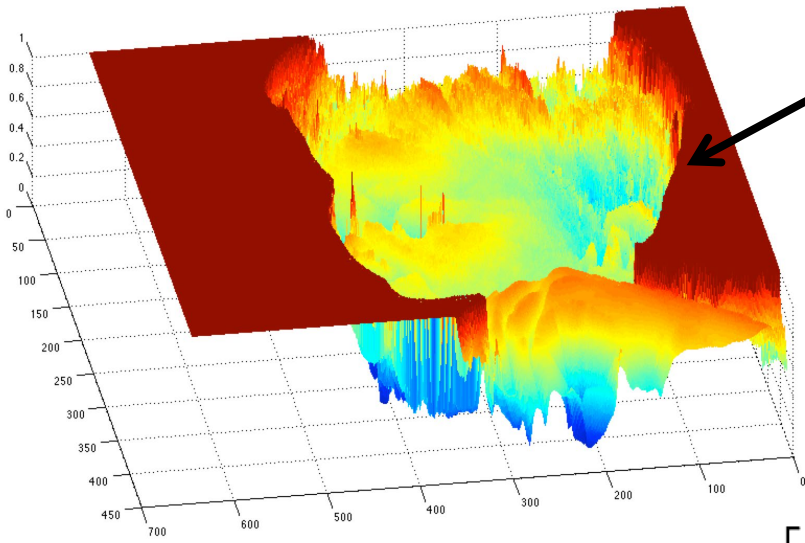
Reading

- Szeliski textbook, Chapter 3.1-3.2
- Szeliski textbook, Chapter 7.2

What are image edges?

grayscale image

$f(\boldsymbol{x})$

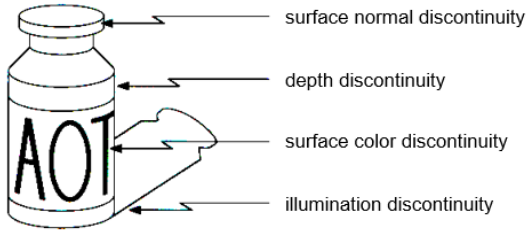


Very sharp discontinuities in intensity.

domain $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$

Edge detection

- In computer vision, edges are sudden discontinuities in an image, which can arise from surface normal, surface color, depth, illumination, or other discontinuities.

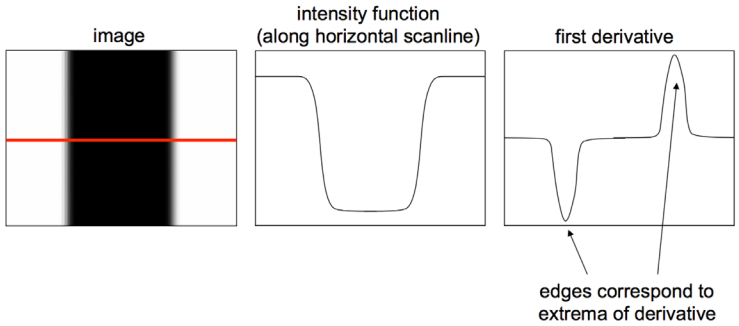


- Edges are important for two main reasons.

1) Most semantic and shape information can be deduced from them, so we can perform object recognition and analyze perspectives and geometry of an image.

2) They are a more compact representation than pixels.

- We can pinpoint where edges occur from an image's intensity profile along a row or column of the image.
- Wherever there is a rapid change in the intensity function indicates an edge, as seen where the function's first derivative has a local extrema.



Detecting edges

How would you go about detecting edges in an image (i.e., discontinuities in a function)?

Detecting edges

How would you go about detecting edges in an image (i.e., discontinuities in a function)?

✓ You take derivatives: derivatives are large at discontinuities.

How do you differentiate a discrete image (or any other discrete signal)?

Detecting edges

How would you go about detecting edges in an image (i.e., discontinuities in a function)?

✓ You take derivatives: derivatives are large at discontinuities.

How do you differentiate a discrete image (or any other discrete signal)?

✓ You use finite differences.

Finite differences

Reminder: definition of a derivative using forward difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Finite differences

Reminder: definition of a derivative using forward difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Alternative: use central difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

For discrete signals: remove limit and set h = 2

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

What convolution kernel does this correspond to?

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

Finite differences

reminder: definition of a derivative using forward difference

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Alternative: use central difference

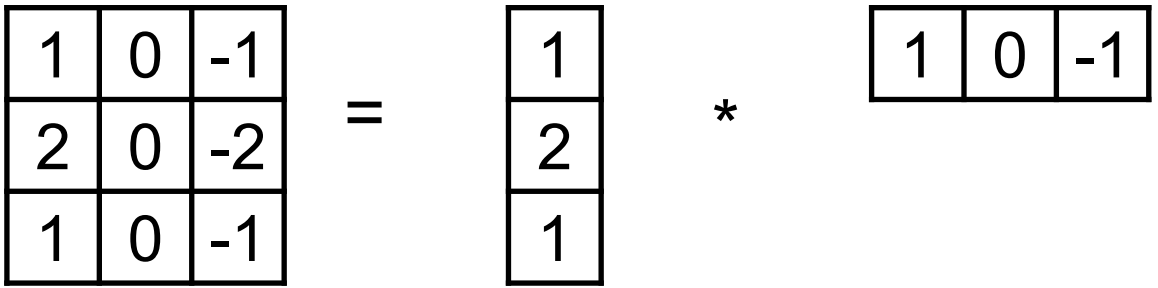
$$f'(x) = \lim_{h \to 0} \frac{f(x+0.5h) - f(x-0.5h)}{h}$$

For discrete signals: remove limit and set h = 2

$$f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

-1	0	1	?
1	0	-1	?

The Sobel filter

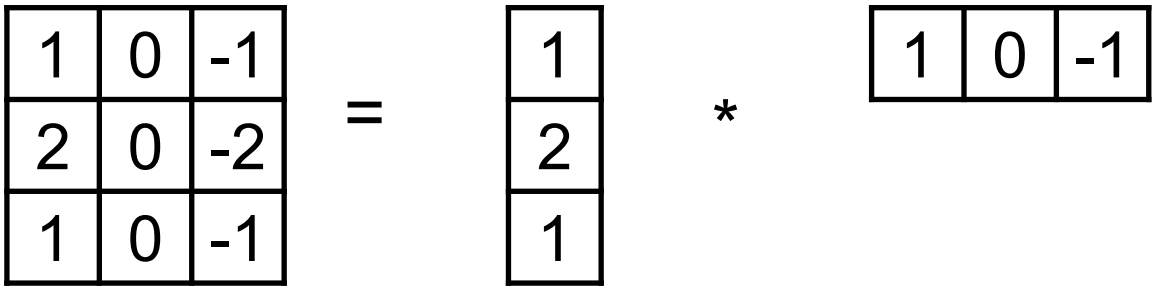


Sobel filter

What filter is this?

1D derivative filter

The Sobel filter

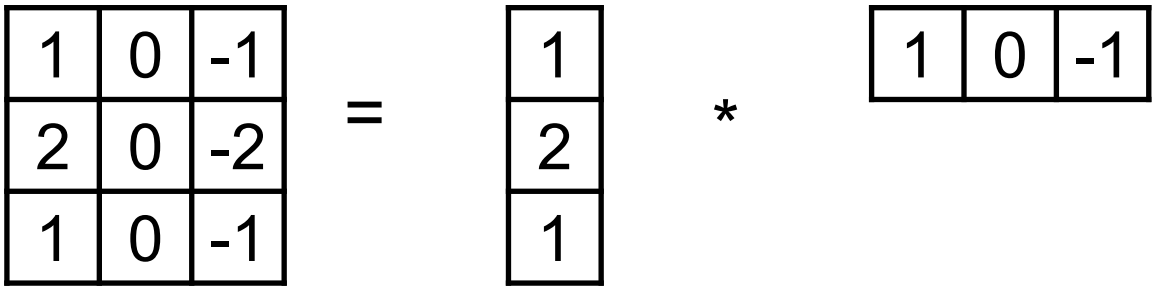


Sobel filter | Blurring | 1D derivative filter

Does this filter return large responses on vertical or horizontal lines?

The Sobel filter

Horizontal Sober filter:



What does the vertical Sobel filter look like?

The Sobel filter

Horizontal Sober filter:

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

=

1
2
1

*

1	0	-1

Vertical Sobel filter:

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

=

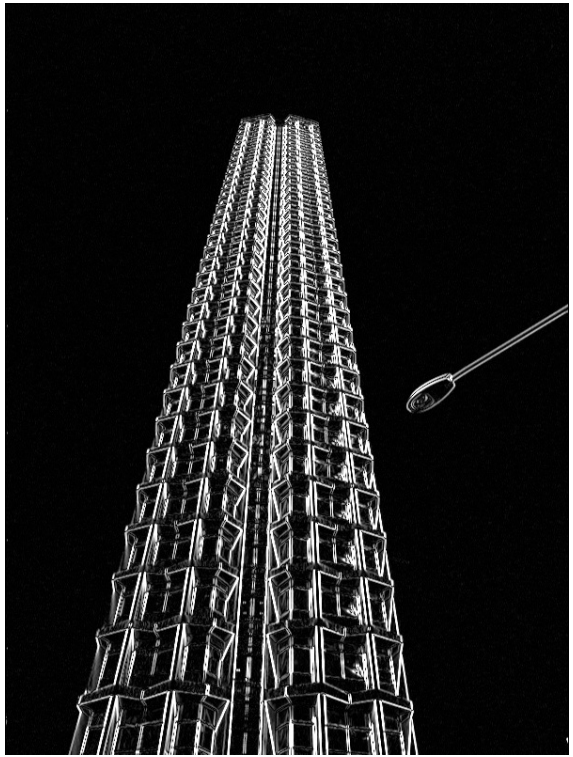
1
0
-1

*

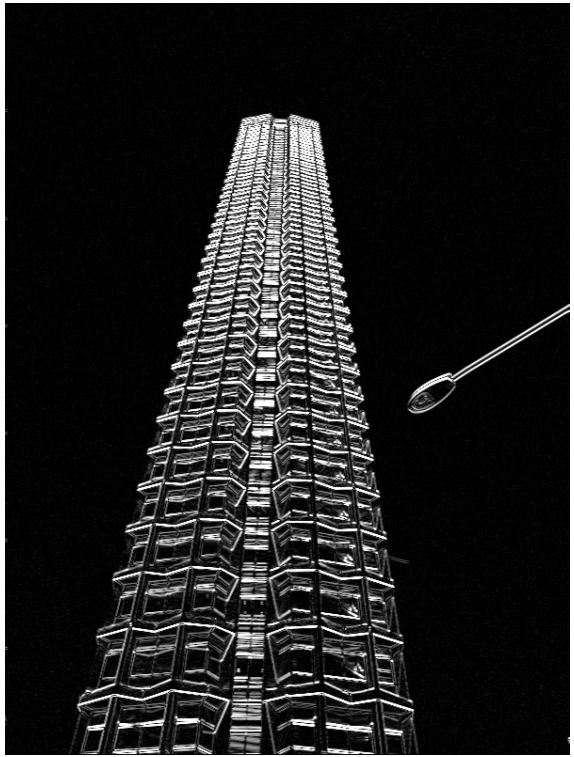
1	2	1

Sobel filter example

original



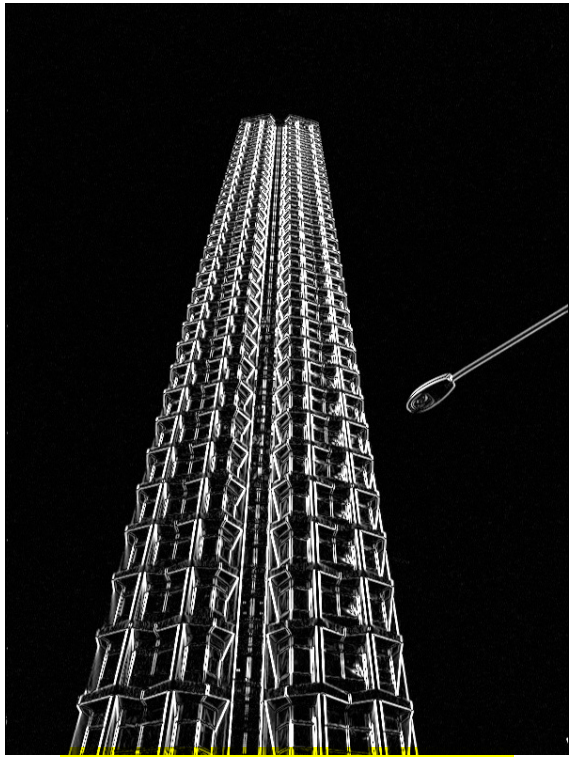
which Sobel filter?



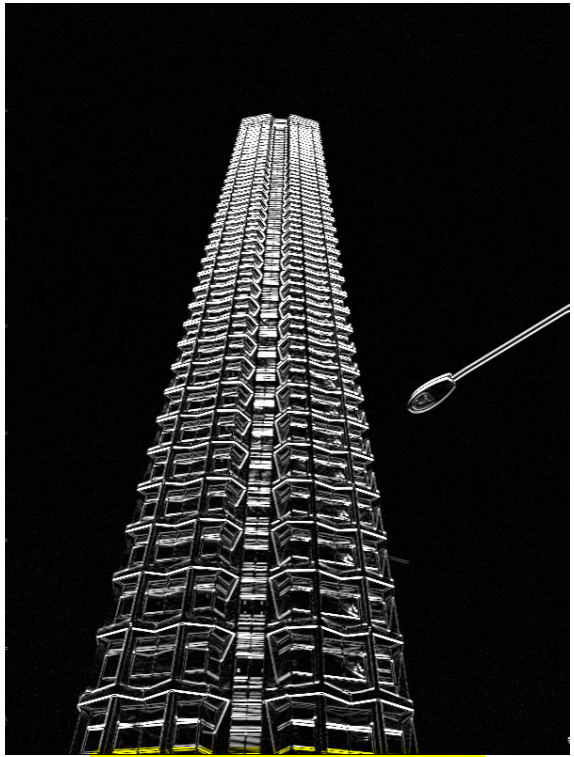
which Sobel filter?

Sobel filter example

original

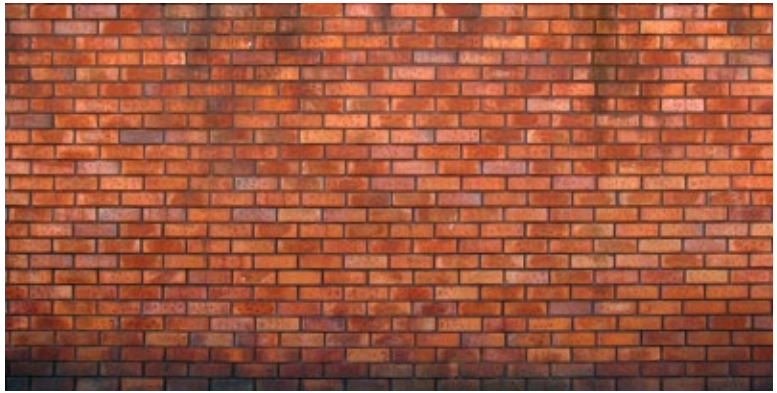


horizontal Sobel filter



vertical Sobel filter

Sobel filter example



original

horizontal Sobel filter

vertical Sobel filter

Several derivative filters

Sobel

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

Scharr

3	0	-3	3	10	3
10	0	-10	0	0	0
3	0	-3	-3	-10	-3

Prewitt

1	0	-1	1	1	1
1	0	-1	0	0	0
1	0	-1	-1	-1	-1

Roberts

0	1
-1	0

1	0
0	-1

Computing image gradients

1. Select your favorite derivative filters.

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

Computing image gradients

1. Select your favorite derivative filters.

$$\boldsymbol{S}_x = \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \qquad \boldsymbol{S}_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

2. Convolve with the image to compute derivatives.

$$\frac{\partial \boldsymbol{f}}{\partial x} = \boldsymbol{S}_x \otimes \boldsymbol{f} \qquad \frac{\partial \boldsymbol{f}}{\partial y} = \boldsymbol{S}_y \otimes \boldsymbol{f}$$

Computing image gradients

1. Select your favorite derivative filters.

$$\boldsymbol{S}_x = \begin{array}{|c|c|c|}\hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array} \qquad \boldsymbol{S}_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

2. Convolve with the image to compute derivatives.

$$\frac{\partial \boldsymbol{f}}{\partial x} = \boldsymbol{S}_x \otimes \boldsymbol{f} \qquad \frac{\partial \boldsymbol{f}}{\partial y} = \boldsymbol{S}_y \otimes \boldsymbol{f}$$

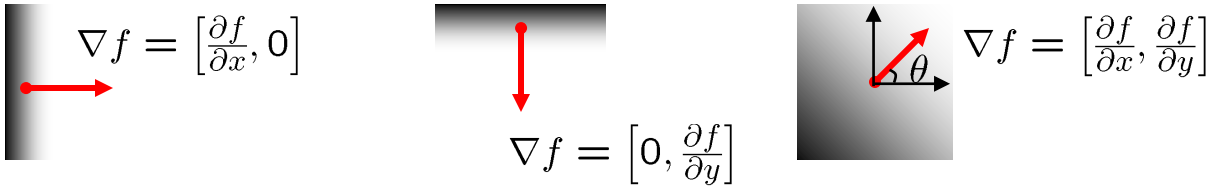
3. Form the image gradient and compute its direction and amplitude.

$$\nabla \boldsymbol{f} = \left[\frac{\partial \boldsymbol{f}}{\partial x}, \frac{\partial \boldsymbol{f}}{\partial y}\right] \qquad \theta = \tan^{-1}\left(\frac{\partial f}{\partial y} \Big/ \frac{\partial f}{\partial x}\right) \qquad \|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

gradient direction amplitude

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 increase in intensity

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

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

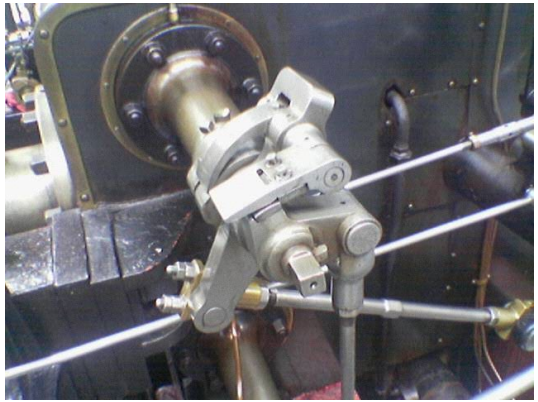
The *edge strength* is given by the gradient magnitude

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

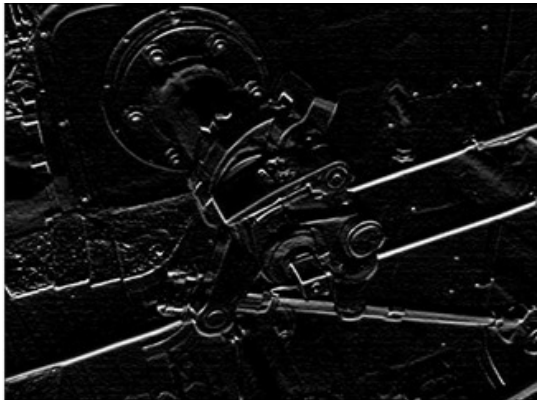
Source: Steve Seitz

Image gradient example

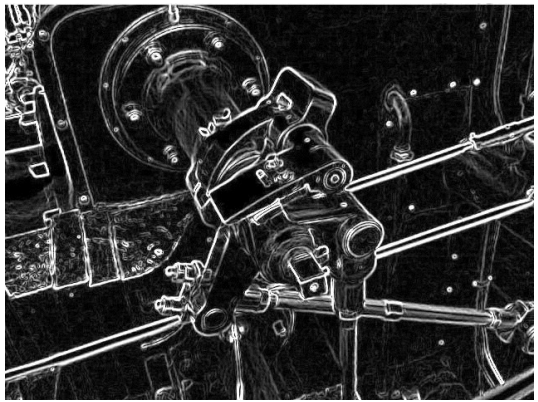
original



vertical derivative



gradient amplitude



horizontal derivative

How does the gradient direction relate to these edges?

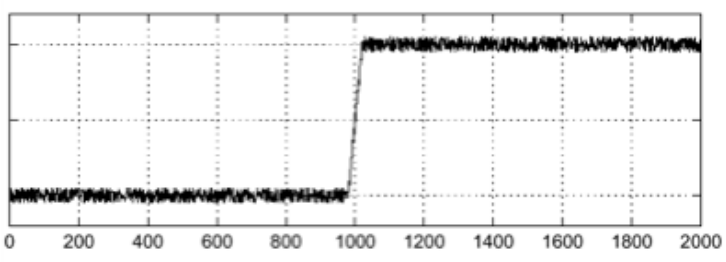
Image gradient (recap)

- An image gradient, which is a generalization of the concept of derivative to more than one dimension, points in the direction where intensity increases the most.
- After gradient images have been computed, pixels with large gradient values become possible edge pixels.
- The pixels with the largest gradient values in the direction of the gradient become edge pixels, and edges may be traced in the direction perpendicular to the gradient direction.

- However, plotting the pixel intensities of the gradient often results in noise, making it almost impossible to identify where an edge is by only taking the first derivative of the function.

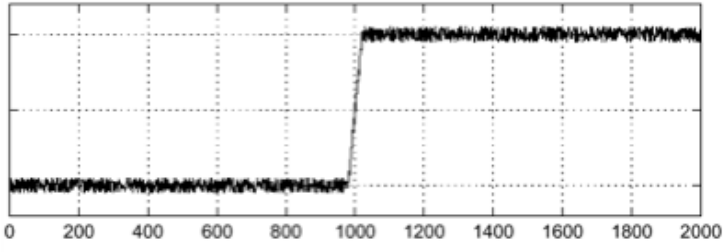
How do you find the edge of this signal?

intensity plot



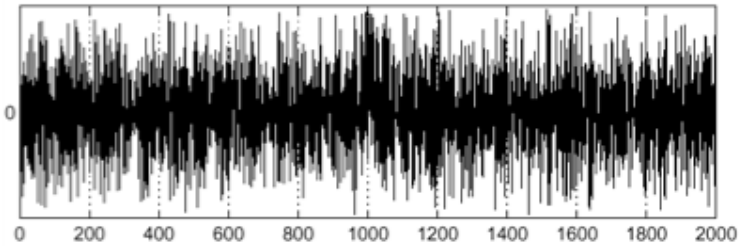
How do you find the edge of this signal?

intensity plot



Using a derivative filter:

derivative plot



What's the problem here? Where is the edge?

Differentiation is very sensitive to noise

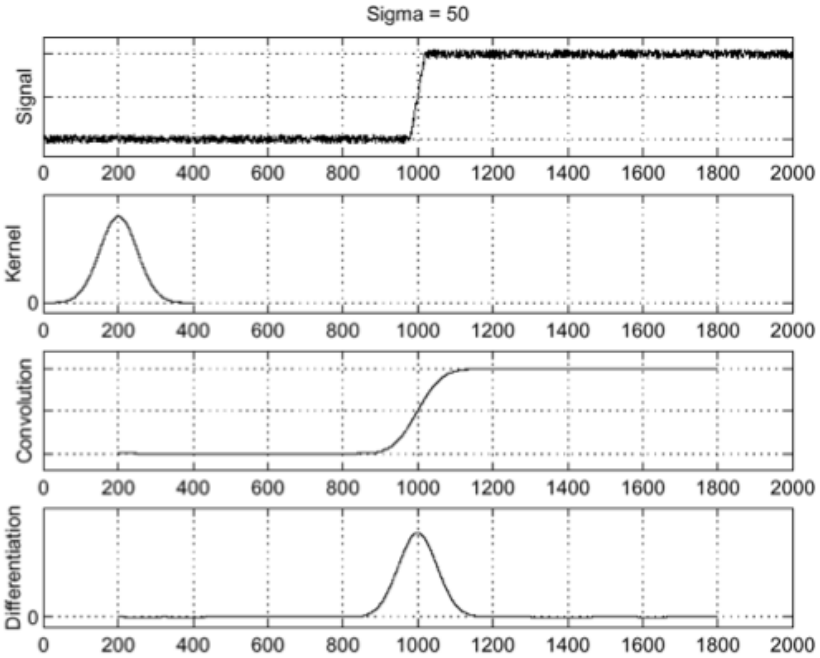
When using derivative filters, it is critical to blur first!

input

Gaussian

blurred

derivative of blurred



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

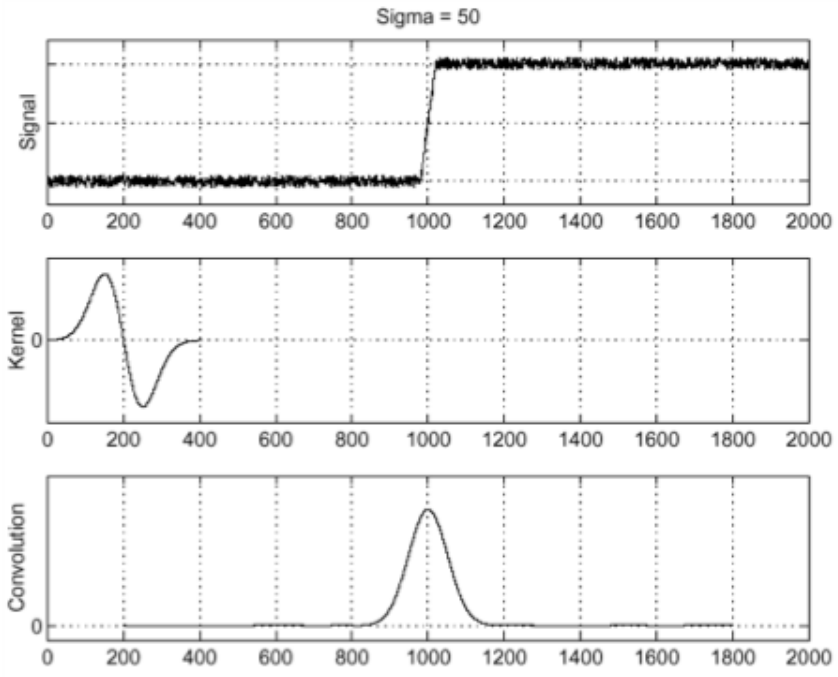
Derivative of Gaussian filter

Derivative theorem of convolution: $\frac{\partial}{\partial x}(h \star f) = (\frac{\partial}{\partial x}h) \star f$

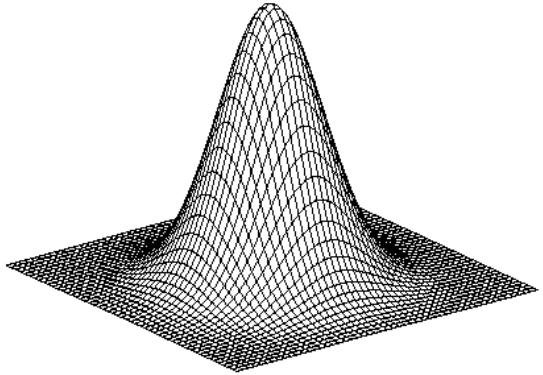
input

derivative of Gaussian

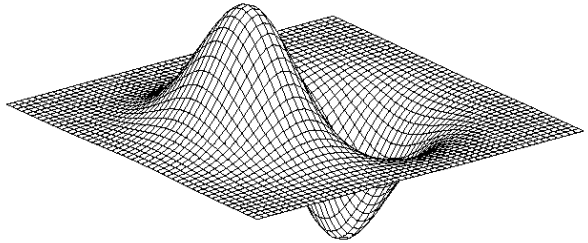
output (same as before)



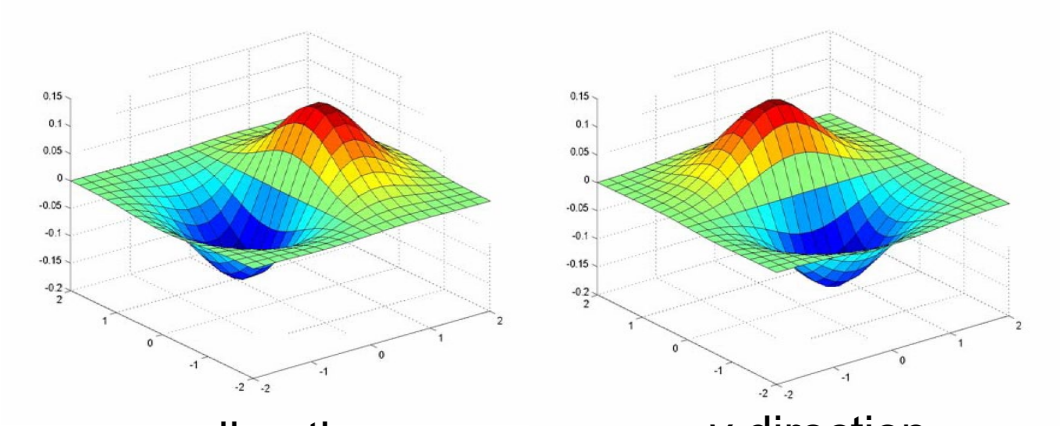
Derivative of Gaussian filter



* [1 -1] =



Derivative of Gaussian filter



x-direction

y-direction

- Which one finds horizontal/vertical edges?

Laplace filter

Basically a second derivative filter.

- We can use finite differences to derive it, as with first derivative filter.

first-order finite difference

$$f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h}$$

→ 1D derivative filter

1	0	-1

second-order finite difference

$$f''(x) = \lim_{h \to 0} \frac{f(x + h) - 2f(x) + f(x - h)}{h^2}$$

→ Laplace filter

?

Laplace filter

Basically a second derivative filter.

- We can use finite differences to derive it, as with first derivative filter.

first-order finite difference

$$f'(x) = \lim_{h \to 0} \frac{f(x + 0.5h) - f(x - 0.5h)}{h}$$

→

1D derivative filter

1	0	-1

second-order finite difference

$$f''(x) = \lim_{h \to 0} \frac{f(x + h) - 2f(x) + f(x - h)}{h^2}$$

→

Laplace filter

1	-2	1

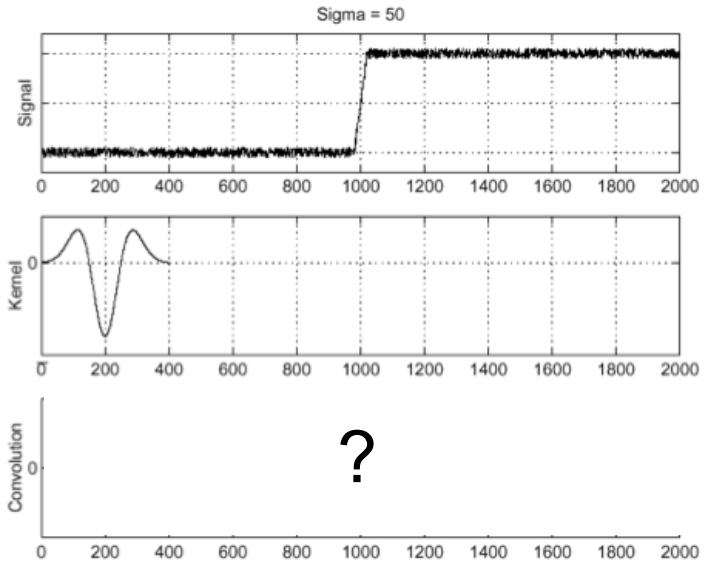
Laplacian of Gaussian (LoG) filter

As with derivative, we can combine Laplace filtering with Gaussian filtering

input

Laplacian of Gaussian

output



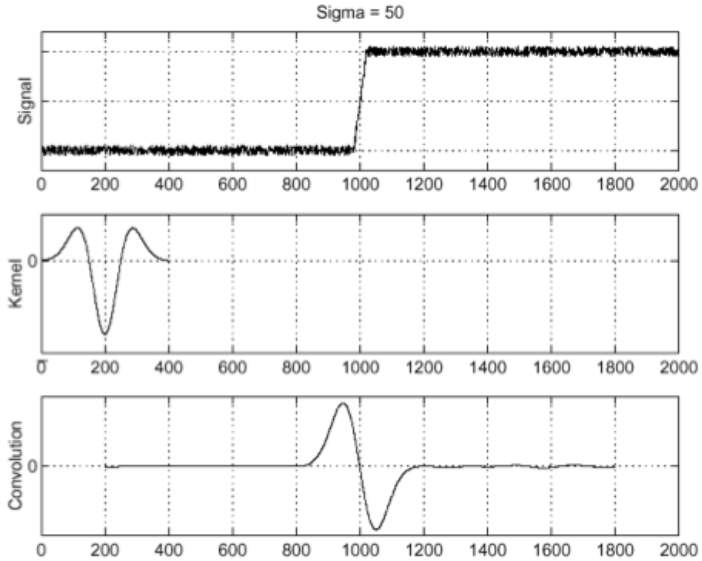
Laplacian of Gaussian (LoG) filter

As with derivative, we can combine Laplace filtering with Gaussian filtering

input

Laplacian of Gaussian

output

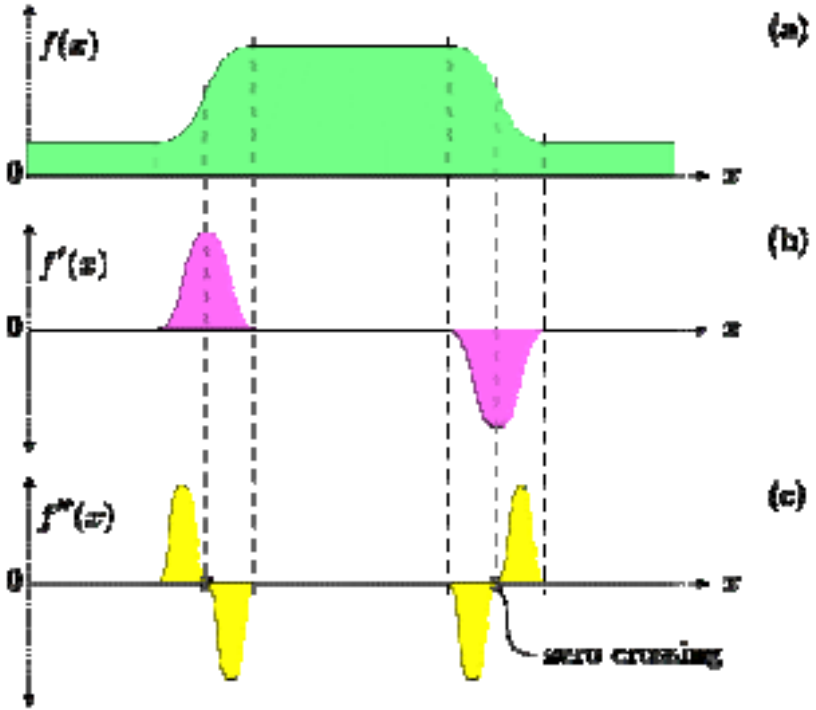


“zero crossings” at edges

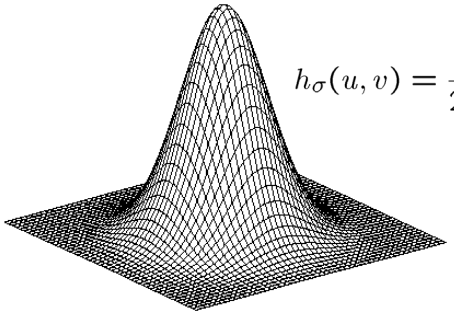
Zero-crossing

- First derivative at edge is a maximum
- Second derivative at edge is zero

- It is more easy to understand when a function is zero than find a maximum



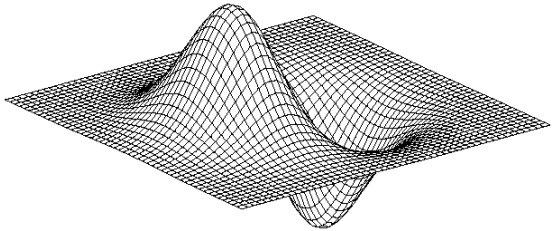
2D Gaussian filters



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

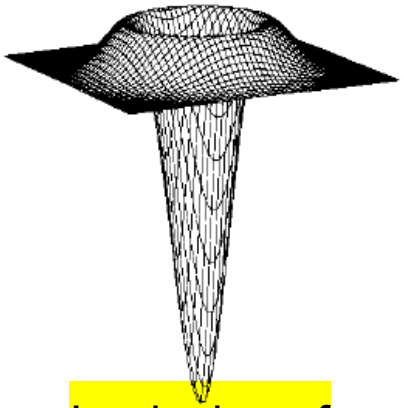
Gaussian

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



Derivative of Gaussian

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



Laplacian of Gaussian

Laplace and LoG filtering examples

Laplacian of Gaussian filtering

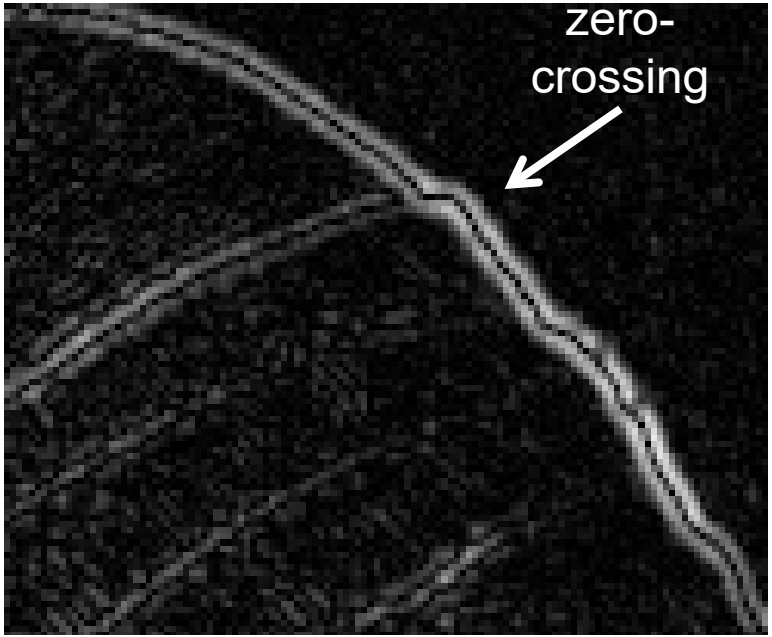
Laplace filtering

Laplacian of Gaussian vs Derivative of Gaussian

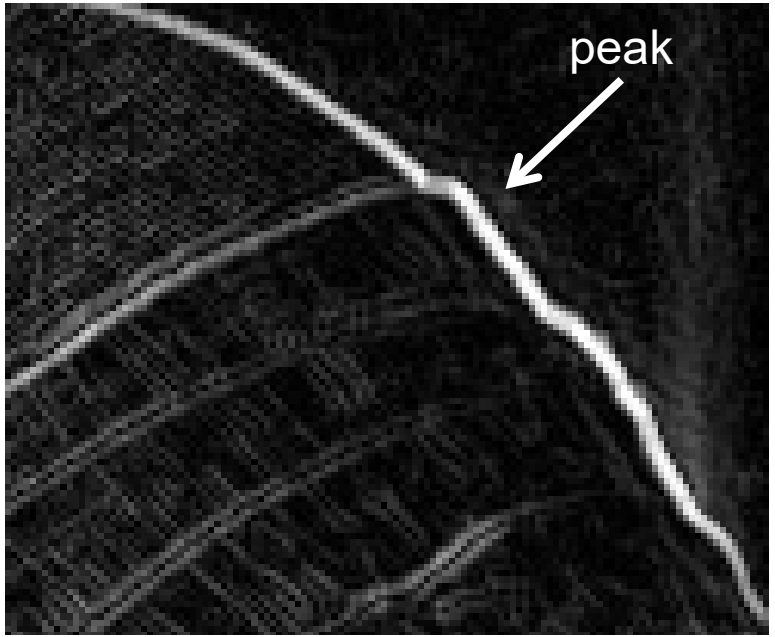
Laplacian of Gaussian filtering

Derivative of Gaussian filtering

Laplacian of Gaussian vs Derivative of Gaussian



Laplacian of Gaussian filtering



Derivative of Gaussian filtering

Zero crossings are more accurate at localizing edges

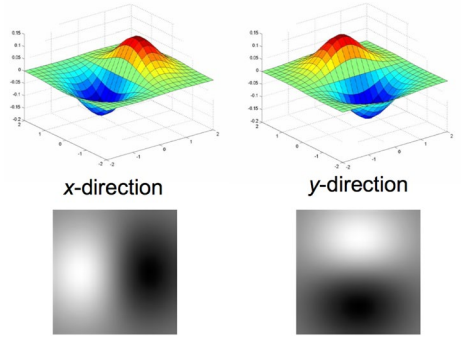
Edge detector

- Building off of this procedure, we can design an edge detector.
- The optimal edge detector must be accurate, minimizing the number of false positives and false negatives; have precise localization, pinpointing edges at the positions where they actually occur; and have single response, ensuring that only one edge is found where there only is one edge

The Canny edge detector 1/3

- The Canny edge detector is the most commonly used edge detector in the field. It detects edges by:

1. Applying the x and y derivatives of a Gaussian filter to the image to eliminate noise, improve localization, and have single response.

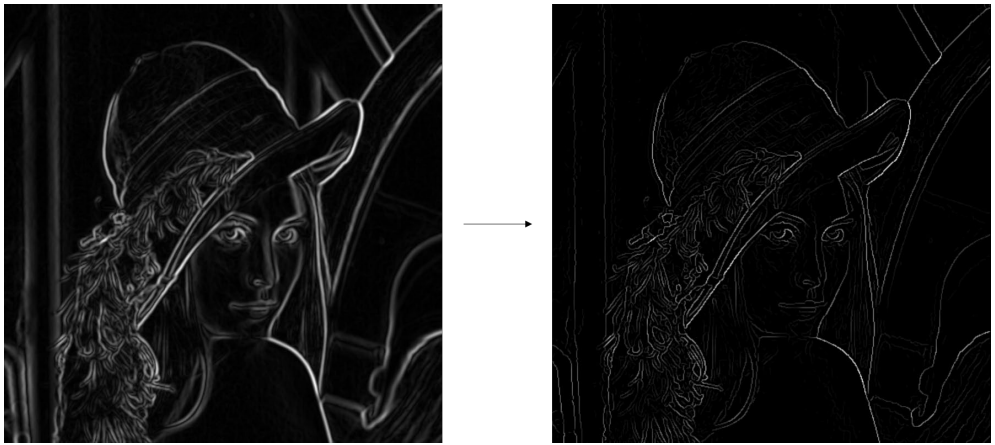


2. Finding the magnitude and orientation of the gradient at each pixel.

https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

The Canny edge detector 2/3

3. Performing non-maximum suppression, which thins the edges down to a single pixel in width: the extracted edge from the gradient after step 2 would be quite blurry but it should be only one accurate response.



The Canny edge detector 3/3

4. Thresholding and linking, also known as hysteresis, to create connected edges.
    1. determine the weak and strong edge pixels by defining a low and a high threshold, respectively,
    2. link the edge curves with the high threshold first to start with the strong edge pixels, continuing the curves with the low threshold.

Example

original image

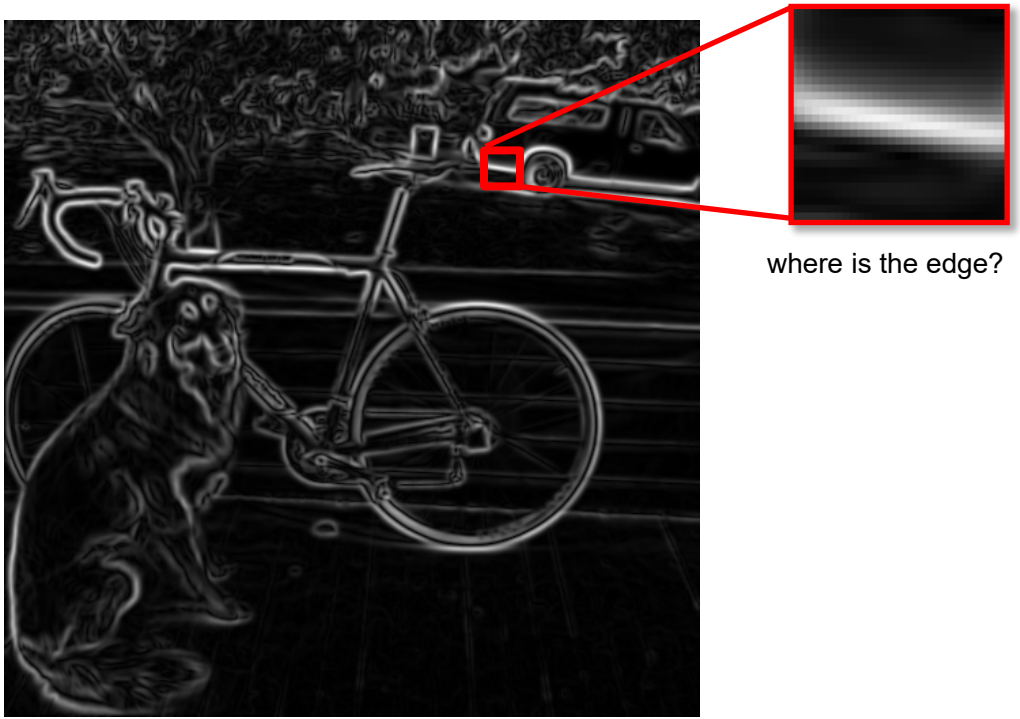
Demo: http://bigwww.epfl.ch/demo/ip/demos/edgeDetector/

Image credit: Joseph Redmon

Phase 1: Finding edges

smoothed gradient magnitude

Phase 1: Finding edges

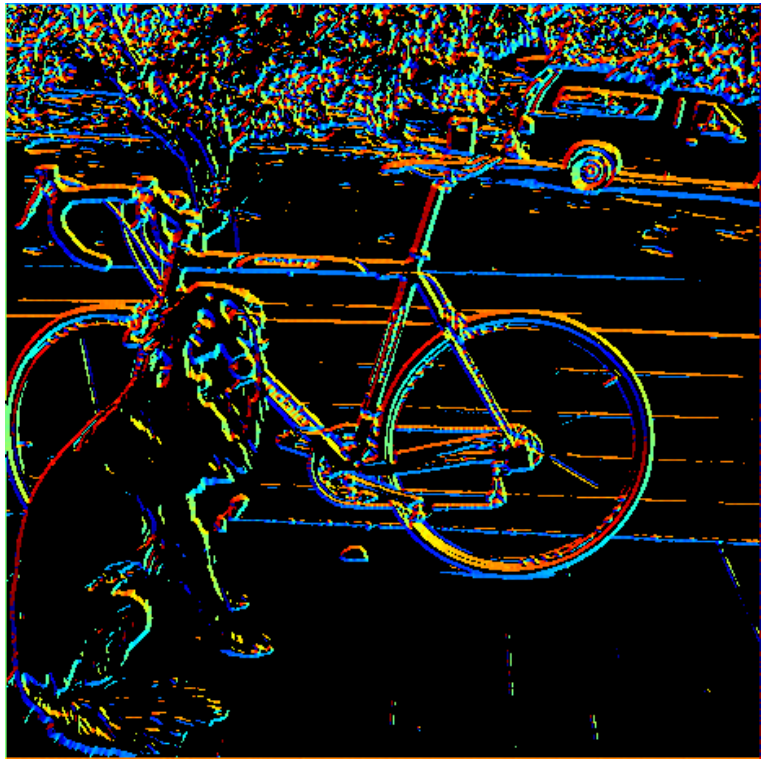


where is the edge?

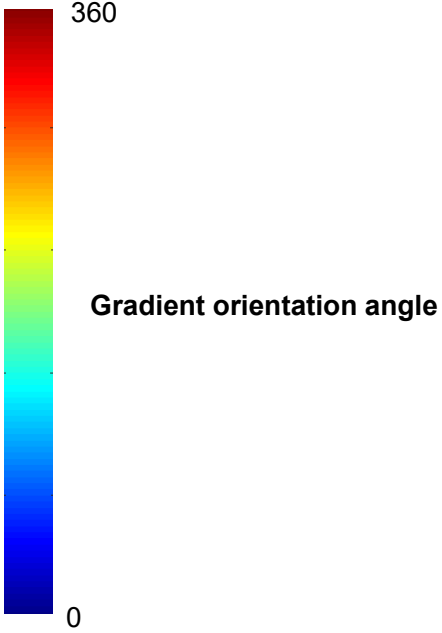
thresholding

Phase 2: get orientation at each pixel

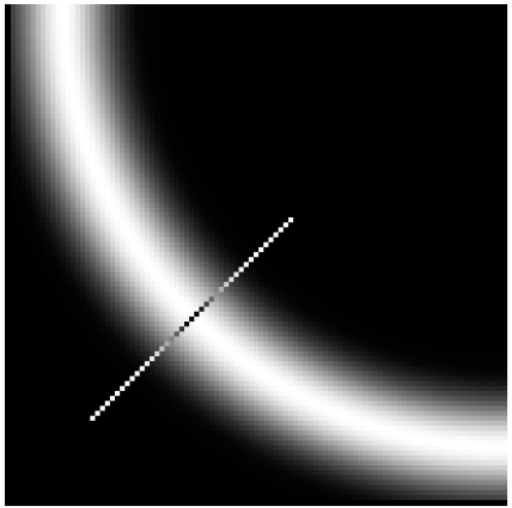
- Get orientation

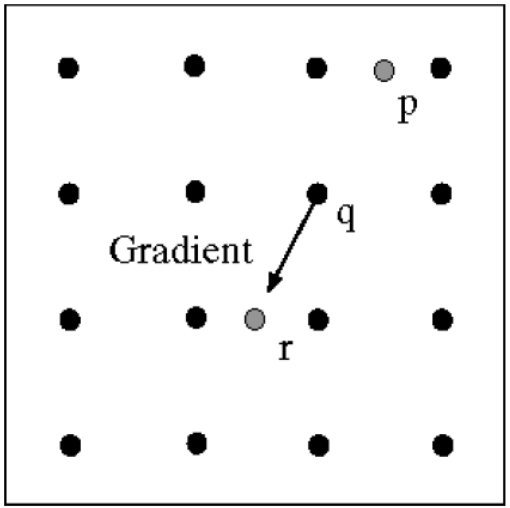


theta = atan2(gy, gx)



Phase 3: Non-maximum suppression



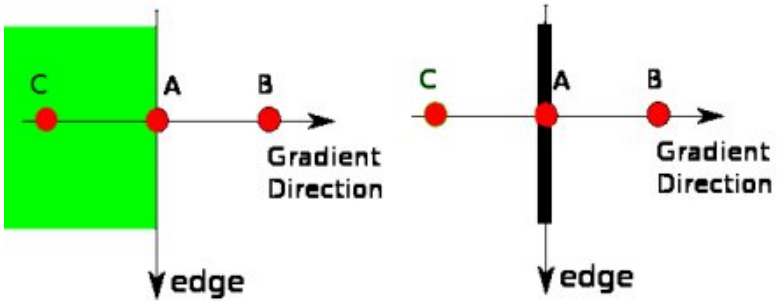


- Check if pixel is local maximum along gradient direction
  - choose the largest gradient magnitude along the gradient direction
  - requires checking interpolated pixels p and r

https://towardsdatascience.com/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123

Non-maximum suppression (example)

- A full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient.



- Point A is on the edge (in vertical direction).
- Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero).
- In short, the result you get is a binary image with "thin edges".

Before Non-max Suppression

After Non-max Suppression

Phase 4: thresholding edges

- Still some noise
- Only want strong edges

- 2 thresholds, 3 cases
  - R > T: strong edge
  - R < T but R > t: weak edge
  - R < t: no edge

Phase 4: Double threshold

The double threshold step aims at identifying 3 kinds of pixels: strong, weak, and non-relevant

- **Strong pixels** are pixels that have an intensity so high that we are sure they contribute to the final edge.
- **Weak pixels** are pixels that have an intensity value that is not enough to be considered as strong ones, but yet not small enough to be considered as non-relevant for the edge detection.
- Other pixels are considered as **non-relevant** for the edge.

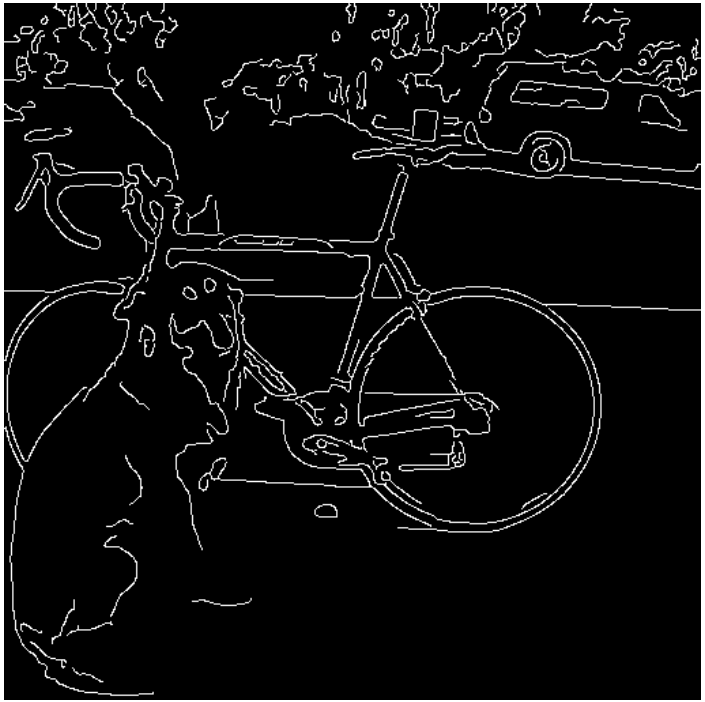
Now you can see what the double thresholds holds for:

- High threshold is used to identify the strong pixels (intensity higher than the high threshold)
- Low threshold is used to identify the non-relevant pixels (intensity lower than the low threshold)
- All pixels having intensity between both thresholds are flagged as weak and the **Hysteresis mechanism** (next step) will help us identify the ones that could be considered as strong and the ones that are considered as non-relevant.
- The result of this step is an image with only 2 pixel intensity values (strong and weak)

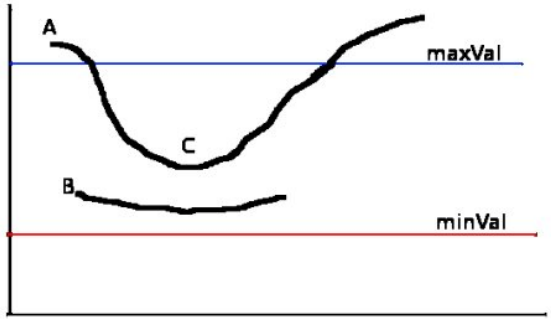
Phase 3: connecting edges

- Strong edges are edges!
- Weak edges are edges
  iff they connect to strong

- Look in some neighborhood
  (usually 8 closest)



Hysteresis thresholding

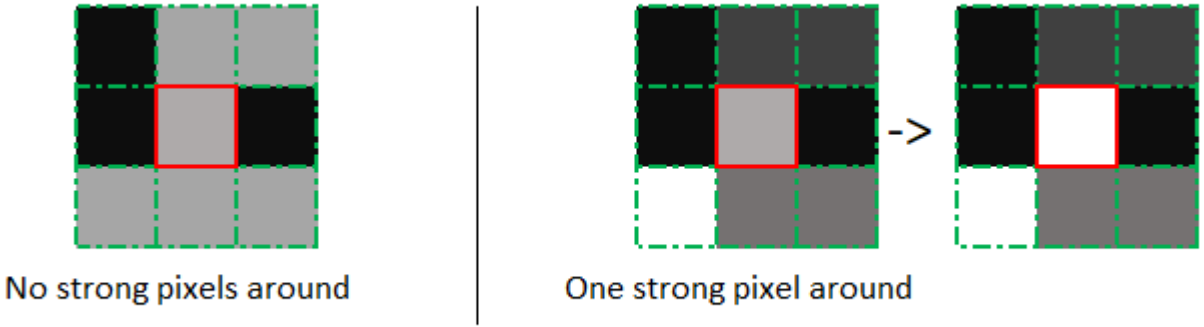


- The edge A is above the maxVal, so considered as "sure-edge".
- Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve.
- Edge B, although it is above minVal and is in same region as that of edge C, it is not connected to any "sure-edge", so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result.

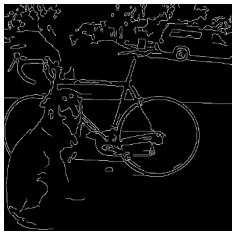
This stage also removes small pixels noises on the assumption that edges are long lines.

Edge Tracking by Hysteresis

Based on the threshold results, the hysteresis consists of transforming weak pixels into strong ones, if and only if at least one of the pixels around the one being processed is a strong one, as described below:



Canny edge detector



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

Source: D. Lowe, L. Fei-Fei, J. Redmon

Canny edge detector

- Our first computer vision pipeline!
- Still a widely used edge detector in computer vision

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

- Depends on several parameters:

  high threshold
  low threshold

  σ : width of the Gaussian blur

Canny edge detector

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 edges

Source: S. Seitz

Recap

- Linear filters and non linear filters
- Convolution is a linear filters
    - Box and Gaussian filters
    - Smoothing (removing noise) and sharpening
- Image gradients
- Canny edge detector

Acknowledgements: some slides and material from Bernt Schiele, Mario Fritz, Michael Black, Bill Freeman, Fei-Fei, Justin Johnson, Serena Yeung, R. Szelisky, Fabio Galasso, Ioannis Gkioulekas