Short Master Machine Learning 2020

Prof. Costantino Grana

Images and color

Sensor Array

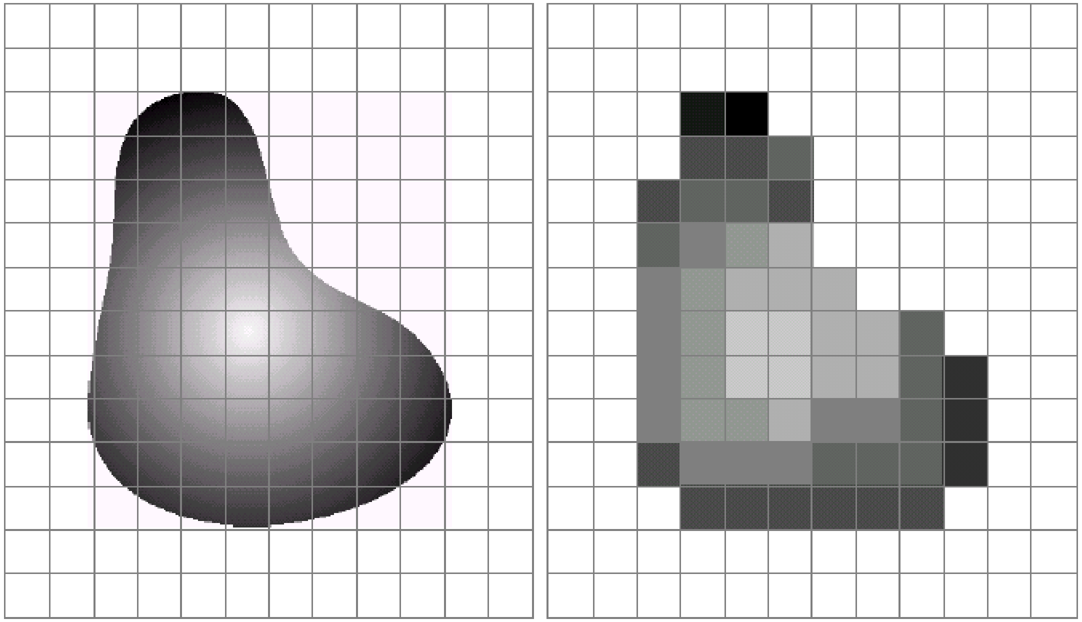
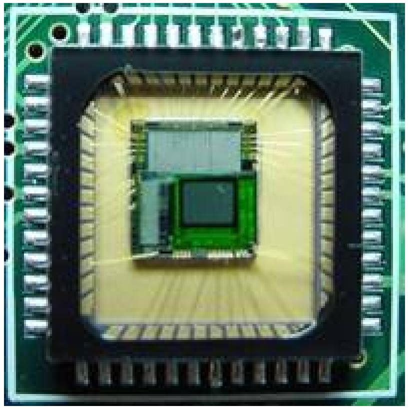


FIGURE 2.17 (a) Continuos image projected onto a sensor array. (b) Result of image sampling and quantization.



CMOS sensor

Sampling and Quantization

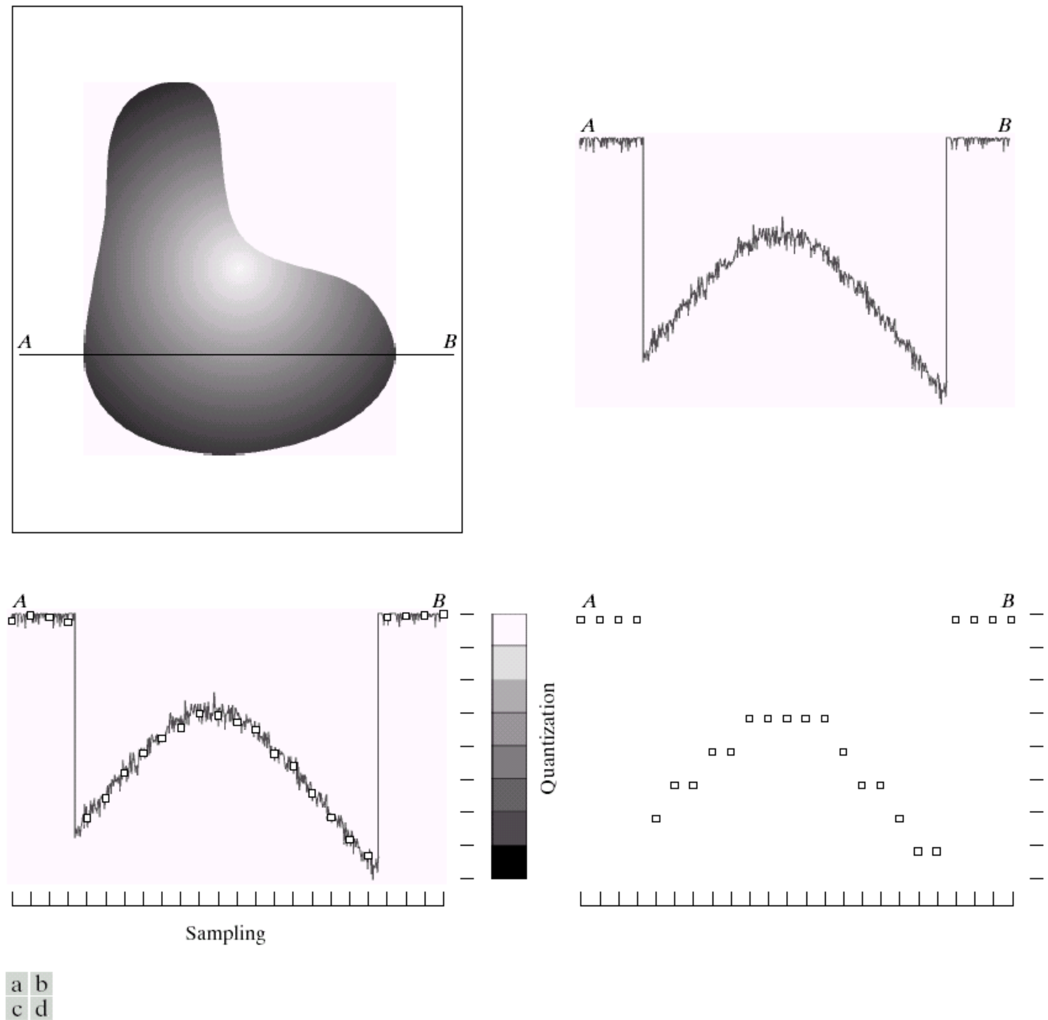


FIGURE 2.16 Generating a digital image. (a) Continuous image. (b) A scan line from A to B in the continuous image, used to illustrate the concepts of sampling and quantization. (c) Sampling and quantization. (d) Digital scan line.

Images

- A (digital) image is defined by *integrating* and *sampling* continuous (analog) data in a spatial domain.
- It consists of a rectangular array of *pixels* (x, y, u), each combining a location $(x, y) \in \mathbb{Z}^2$ and a value u, the *sample* at location (x, y).
- An image *I* with N_{cols} and N_{rows} is defined on a rectangular set

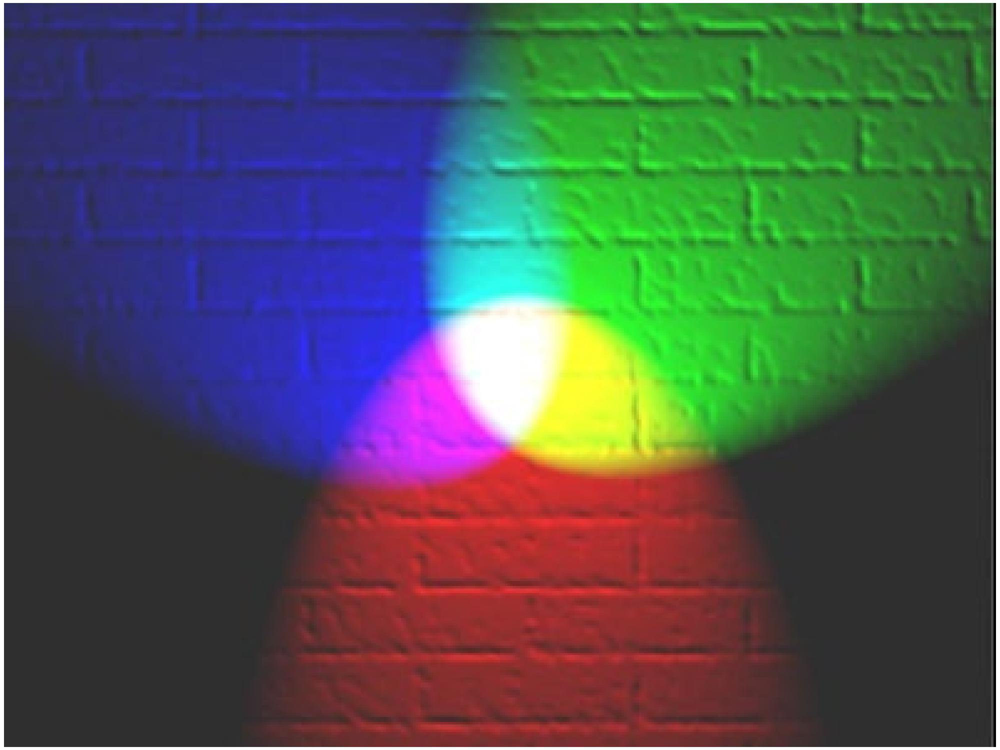
$$\Omega = \{(x, y): 1 \leq x \leq N_{cols} \wedge 1 \leq y \leq N_{rows}\} \subset \mathbb{Z}^2$$

containing the *pixel locations*.

- It is common practice to have *x* increase from left to right and *y* increase from *top to bottom*, which is contrary to the classical Cartesian practice.
- The sample *u* can be a scalar value, which usually represents light intensity, or a vector value, that is the intensity of different light spectra.
- The value can be binary (0 or 1) in case of black and white images, or an integer value from 0 to 2^n, when images have *n* bits per pixel (bpp). A typical graylevel image has 256 levels (8 bpp), but current digital cameras can deliver values with 12, 14 or even 16 bpp.

Color spaces

- How can we represent color?

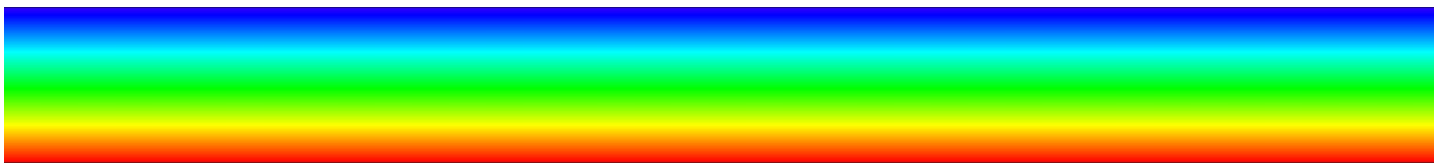


http://en.wikipedia.org/wiki/File:RGB_illumination.jpg

RGB color space

- Color representation is usually done with 3 color channels representing Red, Green and Blue wavelengths.

- Typically we use *truecolor* images, which have three 8 bit values for every pixel, used to drive the monitor display.



- Without other information, it’s logical to assume that the values have been *gamma corrected*, meaning that a power low transformation has been applied to every value.

Color spaces for transmission

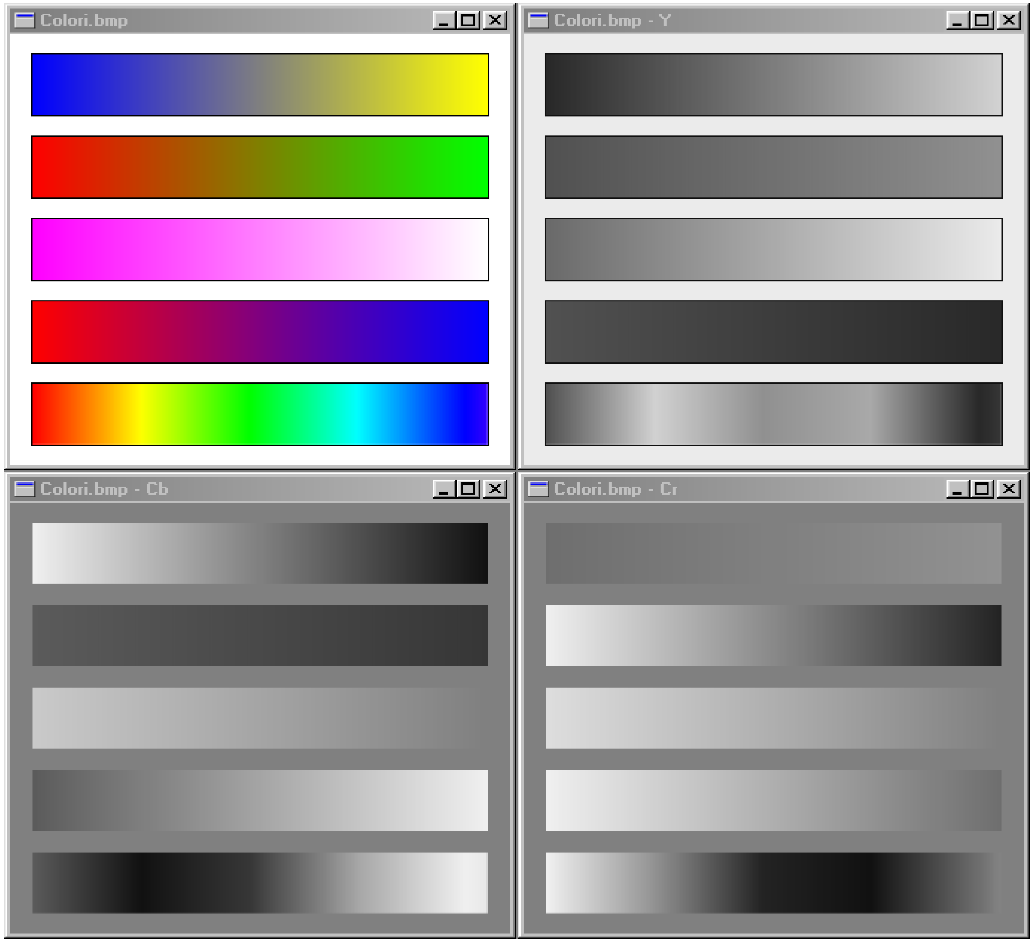
- When created, the color video signal had to be seen also by black and white TVs, so the color components were separated from the luminance one.
- Many standard were proposed, between which we just remember the following:
  - **YIQ** (NTSC)
  - **YUV** (PAL)
  - **YCC** (Kodak PhotoCD)
  - **YC_BC_R** (Digital Video, JPEG, MPEG)
- YIQ and YUV were designed for analogic signals, while in the digital domain the most important one is currently YC_BC_R.
- The conversion formulas from gamma corrected RGB values in the range [0..255] are:

$$\begin{bmatrix} Y \\ C_B \\ C_R \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} + \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix}$$

- These are the ones used in the JPEG File Interchange Format, the one we use every day for storing digital pictures.

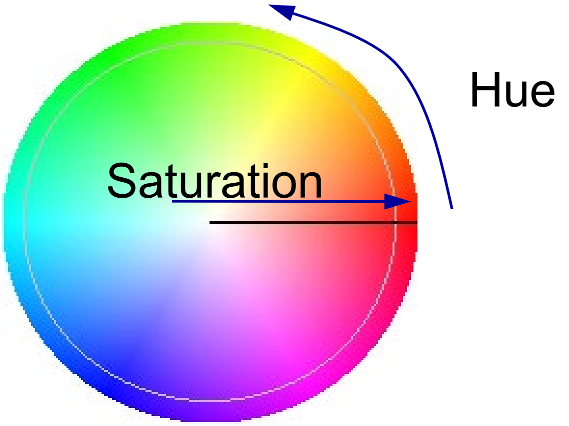
Example of YC_BC_R

- In this example it's possible to see the effect of the YC_BC_R decomposition.



Graphics oriented color spaces

- In addition to the RGB standard, other color spaces used to introduce some kind of numerical specification of color.
- This kind of transformation is useful when dealing with an interface with the human operator.
- These color spaces have no claim of accuracy and are obtained as a transformation from an undefined RGB color space, contrary to other professional colorimetry standards.
- Their representation is based on the concept of luminance and chrominance (A.H. Munsell)
- All characterized by two basic concepts:
  - H = Hue
  - S = Saturation



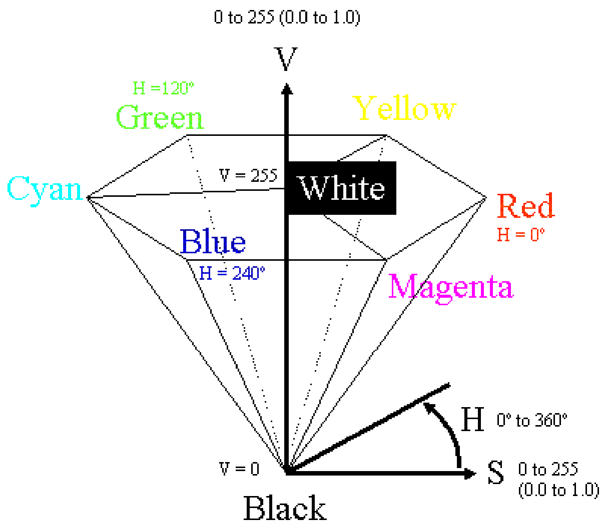
HSV color space

- **HSV** (hue, saturation, value): H is an angle between 0 and 360 degrees, S and V are values in the range [0,1]. This is a transformation of the RGB color space with $0 \le R,G,B \le 1$.
- For every pixel we define

$$Max = \max(R,G,B)$$
$$Min = \min(R,G,B)$$

- The coordinates are given by the following equations:

$$V = Max$$
$$S = \frac{(Max - Min)}{Max}$$
$$H = \frac{\pi}{3}\begin{cases} \frac{G-B}{(Max-Min)} & Max = R \\ 2 + \frac{B-R}{(Max-Min)} & Max = G \\ 4 + \frac{R-G}{(Max-Min)} & Max = B \end{cases}$$



HLS color space

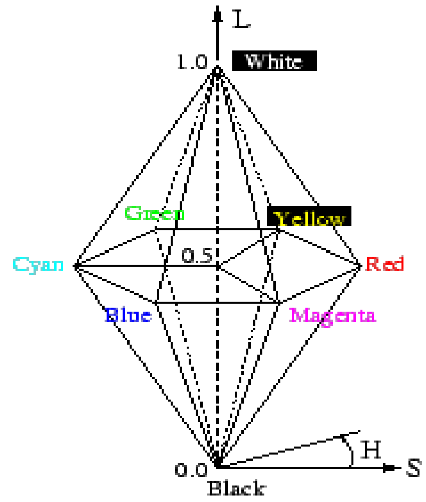
- It’s just a variation of the HSV color system where *“value”* is substituted with *“lightness”:*

$$L = \frac{Max + Min}{2}$$

- We need to change the definition of S accordingly:

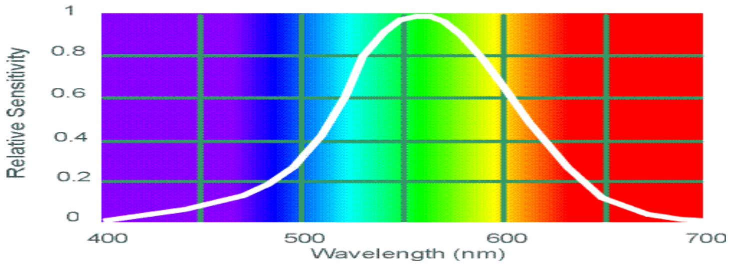
$$S = \begin{cases} \dfrac{Max - Min}{Max + Min} & L \le 0.5 \\ \dfrac{Max - Min}{2 - (Max + Min)} & L > 0.5 \end{cases}$$

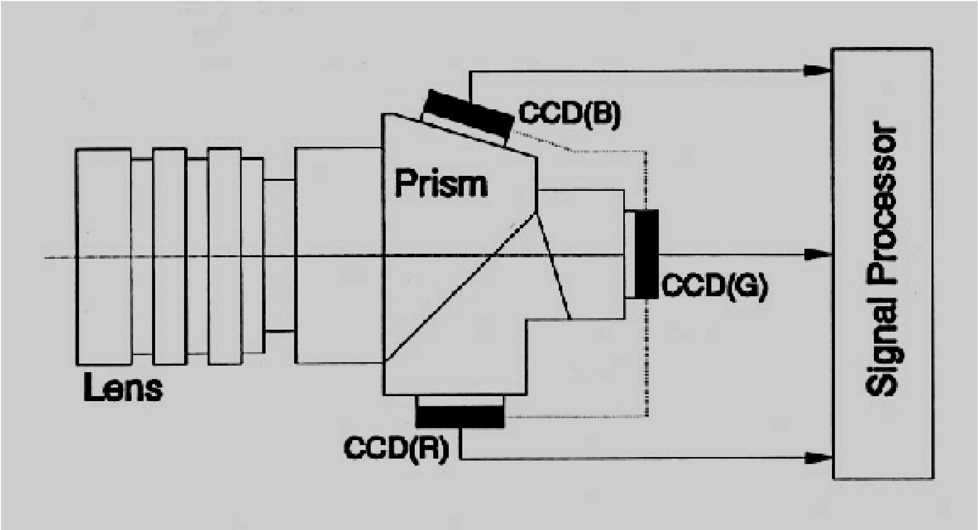
  - The pyramidal structure of HSV is doubled as shown here.
  - This is the color system used in the standard color selection dialog box on Windows systems.

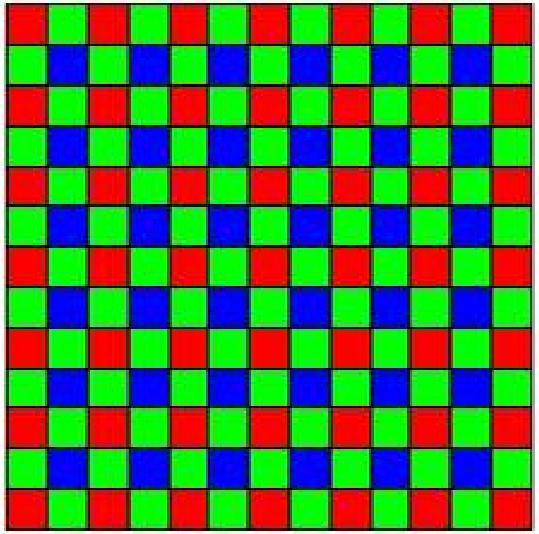


Color Sensing in Camera (RGB)

- 3-chip vs. 1-chip: quality vs. cost
- Why more green?



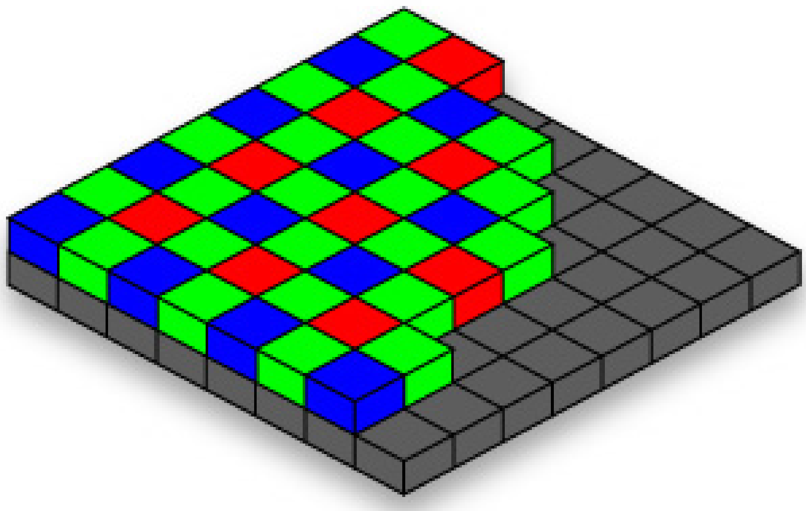


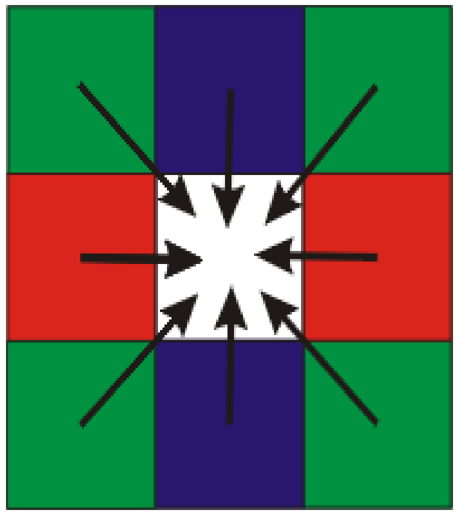


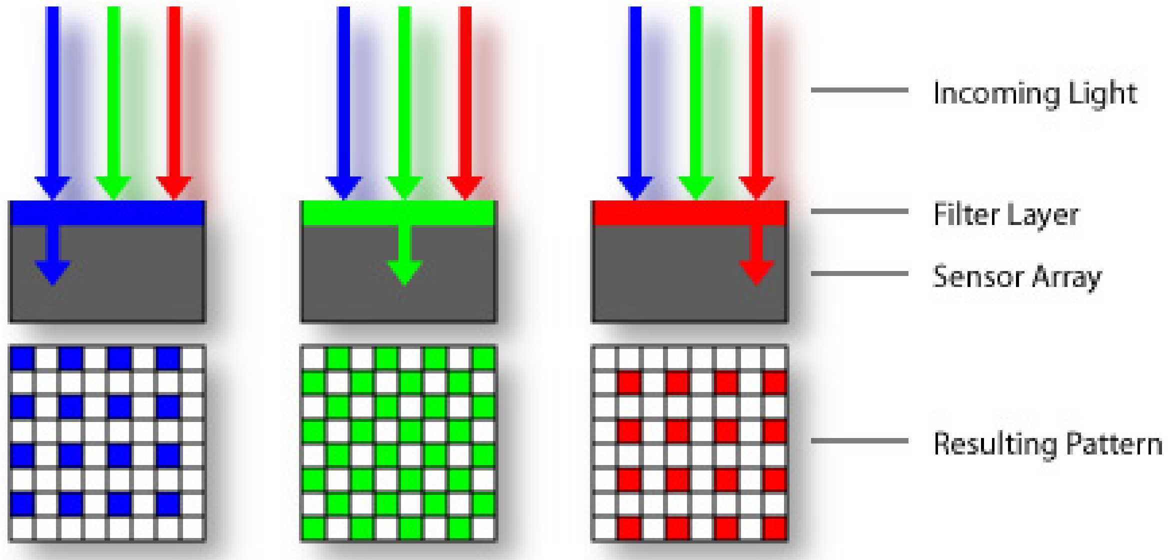
Bayer filter

Why 3 colors?

Practical Color Sensing: Bayer Grid

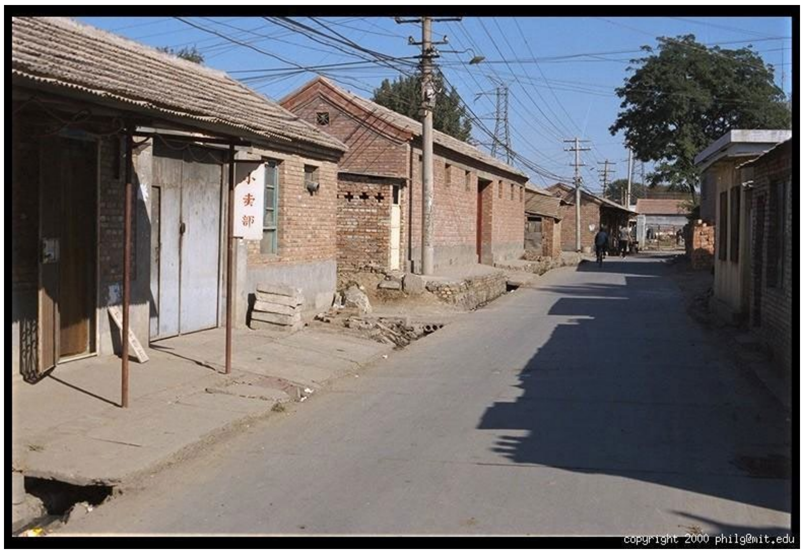




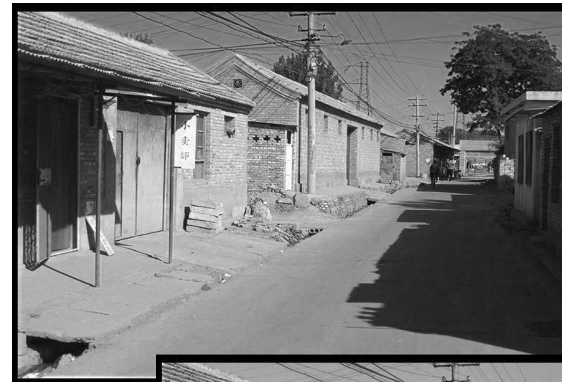


- Estimate RGB at ‘G’ cells from neighboring values

Color Image



R



G

B

If you had to choose, would you rather go without luminance or chrominance?

If you had to choose, would you rather go without luminance or chrominance?

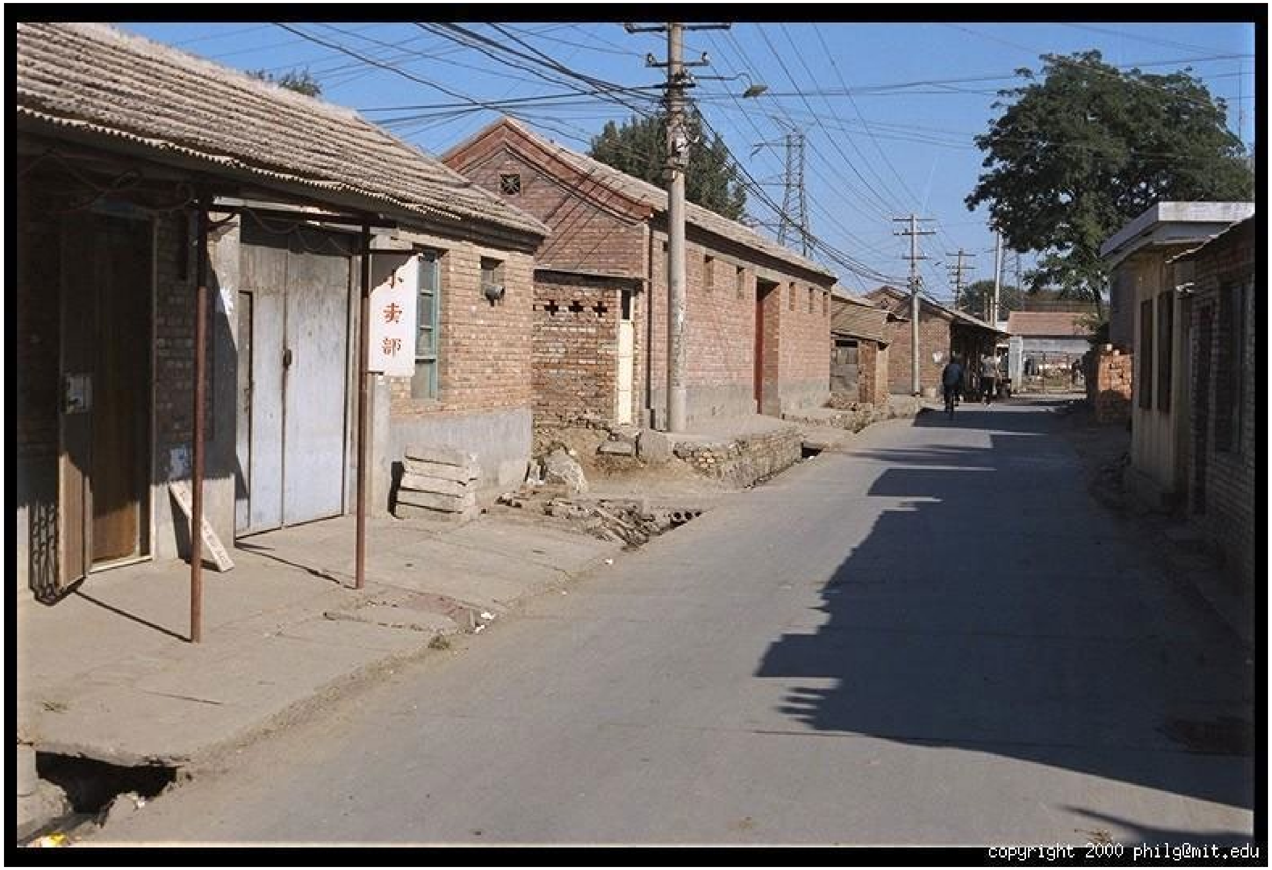
Most information in intensity

Only color shown – constant intensity

Most information in intensity

Only intensity shown – constant color

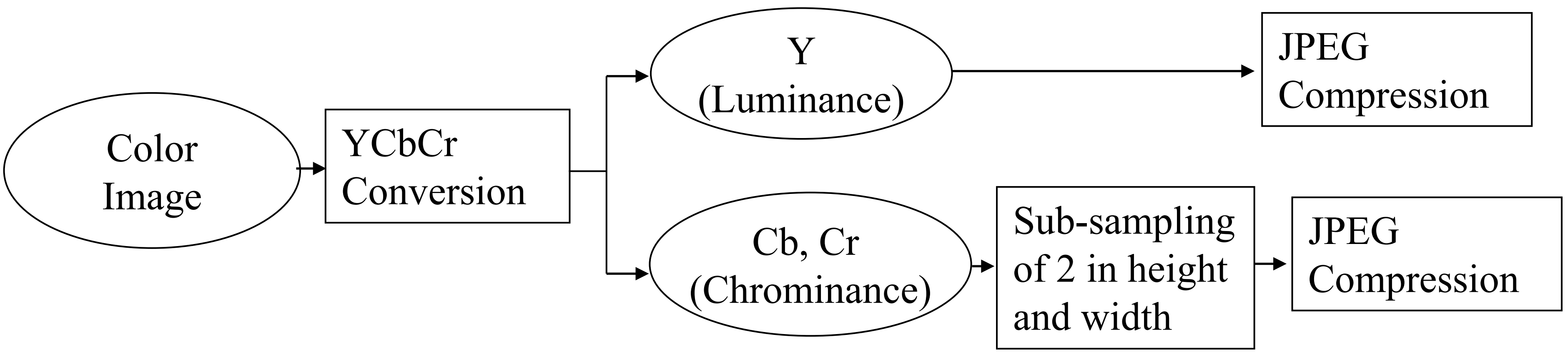
Most information in intensity



Original image

Color Image Management

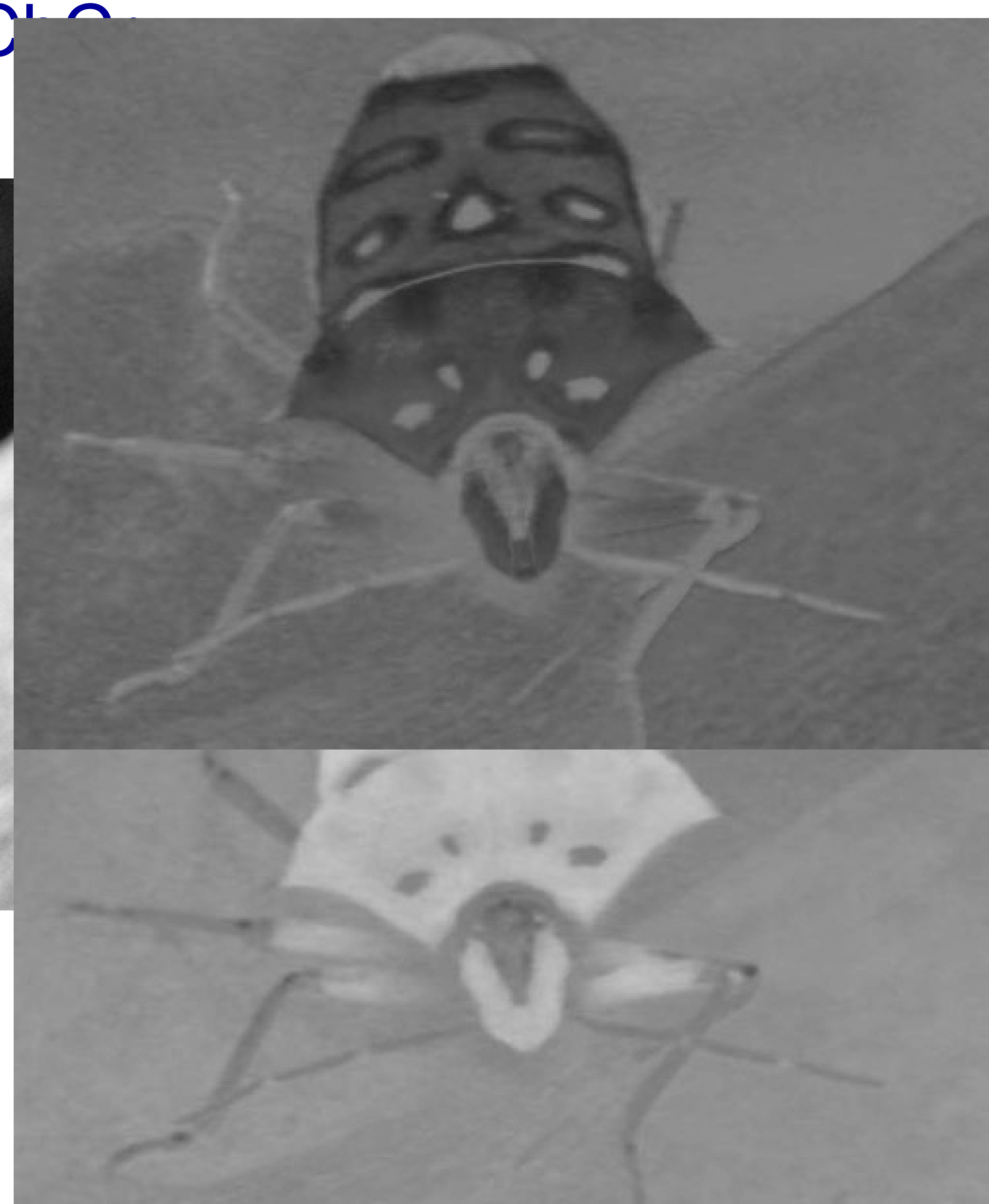
- However, the JFIF format has been defined (JPEG File Interchange Format) which provides the following transformation:



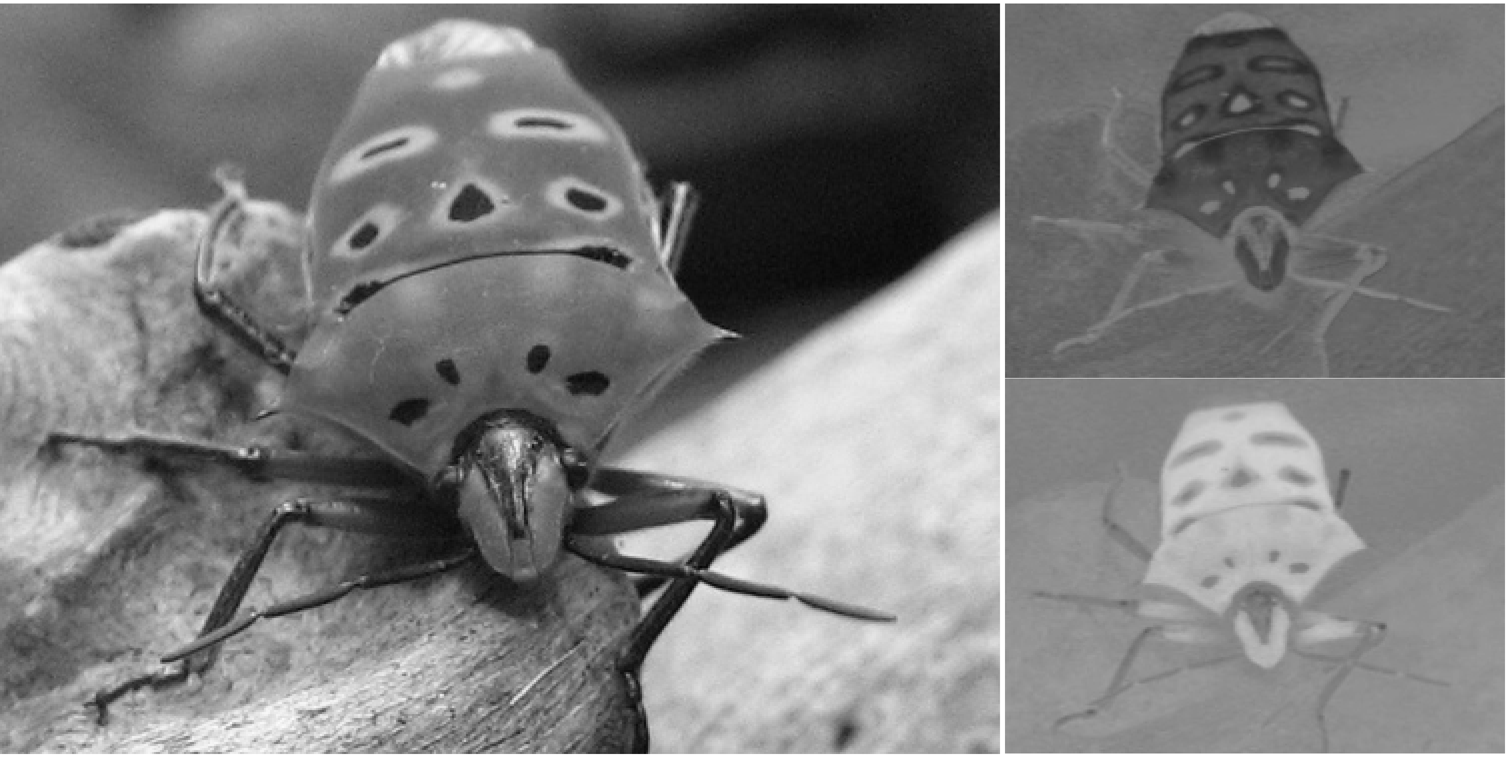
- The color components are sub-sampled by reducing the size to one quarter of the original.

Example of Subsampling in YCbCr

Converting to YC



Normale (1/2) Cb and Cr Subsampling



RGB Reconstruction

Original

Exceeding (1/8) Cb and Cr Subsampling

RGB Reconstruction

Original

Absurd (1/32) Cb and Cr Subsampling

RGB Reconstruction

Original

Short Master Machine Learning 2020

Prof. Costantino Grana

Image processing point operators

Image processing operators

1. **Point operators:** the value of each pixel of the resulting image depends only on the original pixel in the same image's spatial position (e.g. thresholding)

$$I'(x,y) = h\big(I(x,y)\big) \text{ or in vectorial form } I'(\boldsymbol{x}) = h\big(I(\boldsymbol{x})\big)$$

2. **Local (Neighborhood) operators**: the value of each pixel depends on the original pixel in the same image's position and on those in a local Neighborhood (e.g. filters)

$$I'(x,y) = h\left(I(x,y), \aleph\big(I(x,y)\big)\right)$$

3. **Global operators:** the value of each pixel depends on all the pixels of the original image (e.g. Fourier transform)

$$I'(x,y) = h\left(\iint I(x,y)\right)$$

Linear point operator

- Punctual (point) operators can be applied on one or more images:
  - $g(\boldsymbol{x}) = h(f(\boldsymbol{x}))$ or $g(\boldsymbol{x}) = h\left(f_0(\boldsymbol{x}), f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x})\right)$
- $\boldsymbol{x}$ is defined on the domain of the function/image; for the image, it is the pixel location, represented by the point coordinates in the 2D plane $\boldsymbol{x} = (i, j)$.
- *h() is the operator which transforms an image to another image after an image processing operation*
- If the h() **transformation** is **linear** it can be written as

$$g(\boldsymbol{x}) = h\left(f(\boldsymbol{x})\right) = s \cdot f(\boldsymbol{x}) + k$$

- s is the **scale factor**, often called also *gain or contrast*
- k is the **offset constant** often called also bias or brightness
- Linear operators obey the superposition principle, i.e. $h(f_0 + f_1) = h(f_0) + h(f_1)$

Saturated arithmetic

- Be careful in discretized computer world: in integer arithmetic we approximate to the nearest integer.

- It's also common to use saturated arithmetic:
  - $I'(x) = 0 \text{ if } s \cdot I(x) + k < 0$
  - $I'(x) = maxrange \text{ if } s \cdot I(x) + k > maxrange$

- maxrange is often 255.

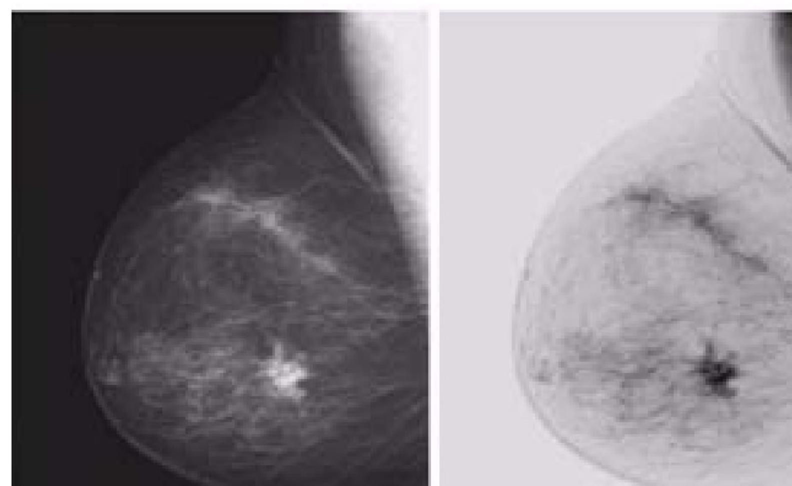
Luminance variation

- If the image has low contrast it's possible to improve the visibility of details, by changing the *scale* factor (e.g. a 10% more):

$$s = 1.1, k = 0$$

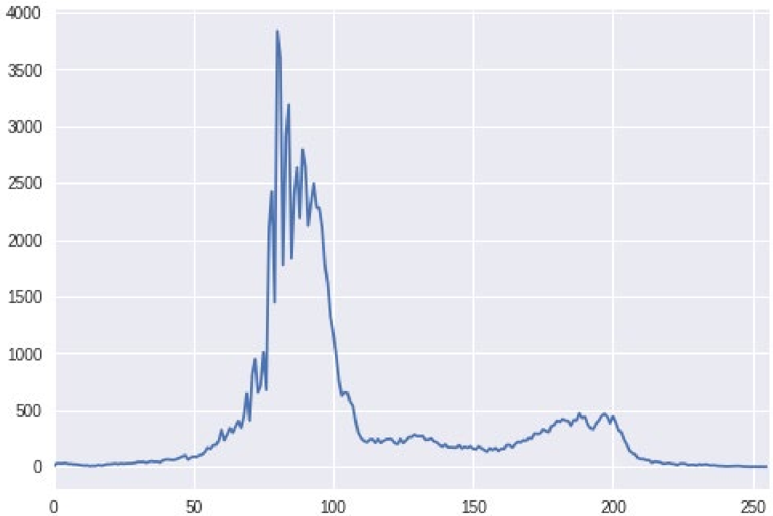
- Sometimes white on black is poorly intelligible, so changing it to black on white is better. This is called the negative operator:

$$s = -1, k = maxvalue$$



Histogram

- The gray level **histogram** is a vector with as many elements (**bins**) as the number of gray levels;
- The value of each bin is the accumulation of the number of pixels which, in that image, assume the correspondent gray level;
- The histogram gives important information for image processing, especially for contrast enhancement and segmentation.



Histogram

- **Histogram** can be viewed as a discrete approximation of a **probability distribution**

$$H_I(i) = h_i = \#\{\boldsymbol{x} : I(\boldsymbol{x}) = i\}$$

where # means “the number of elements in the following set”, and $0 \leq i \leq L$, with L the number of possible levels in the image.

- In order to treat the bin values as the probability of occurrence of a gray level in the image, the normalized version of histogram must be used (the sum of all bins shall be equal to 1):

$$\sum_{j=0}^{L-1} p_I(j) = 1$$

Normalized histogram

- Consider a discrete grayscale image I(***x***) and let n_i be the number of occurrences of gray level *i*.
- The probability of an occurrence of a pixel of level *i* in the image is

$$\mathrm{p_I}(i) = \frac{\#\{x: I(x)=i\}}{n} \qquad 0 \leq i \leq L$$

- *L* being the total number of gray levels in the image, *n* being the total number of pixels in the image, and *p*(*i*) being in fact the image's histogram for pixel value i, normalized to [0,1].
- → *the normalized histogram is the probability distribution*

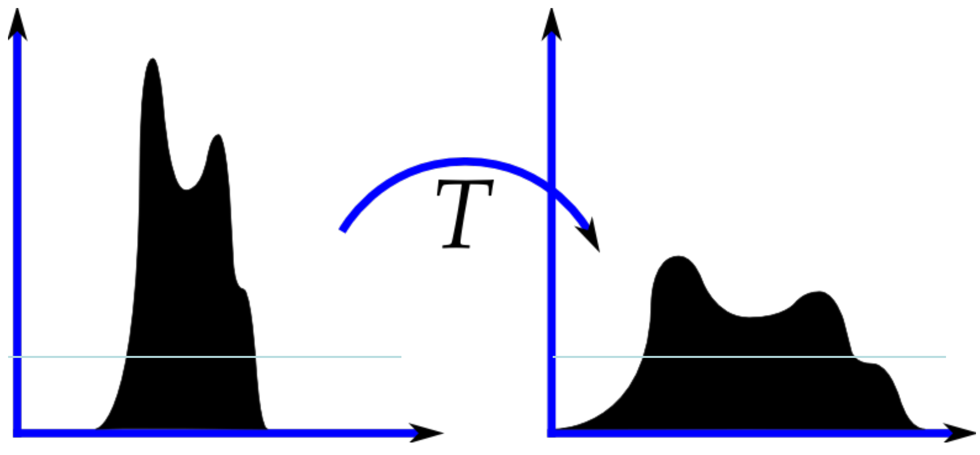
HISTOGRAM

- We can compute:
  - **Mean**
  - **Standard Deviation**
- Histogram is fundamental to

1) Measuring the distribution of a feature in the image (gray level, color, motion, gradient…)

2) Verifying the mono-multimodality of an image for **segmentation**

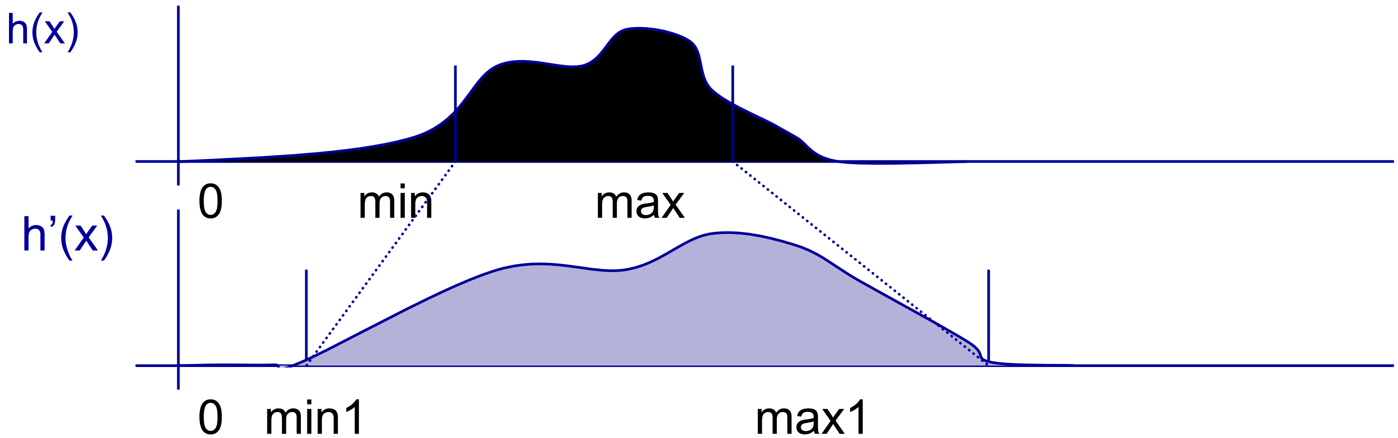
3) Implementing tools for imaging such as **histogram equalization** etc



Similar mean
different sd

Contrast-stretching

Contrast stretching expansion for gray level with a dynamic range given the histogram

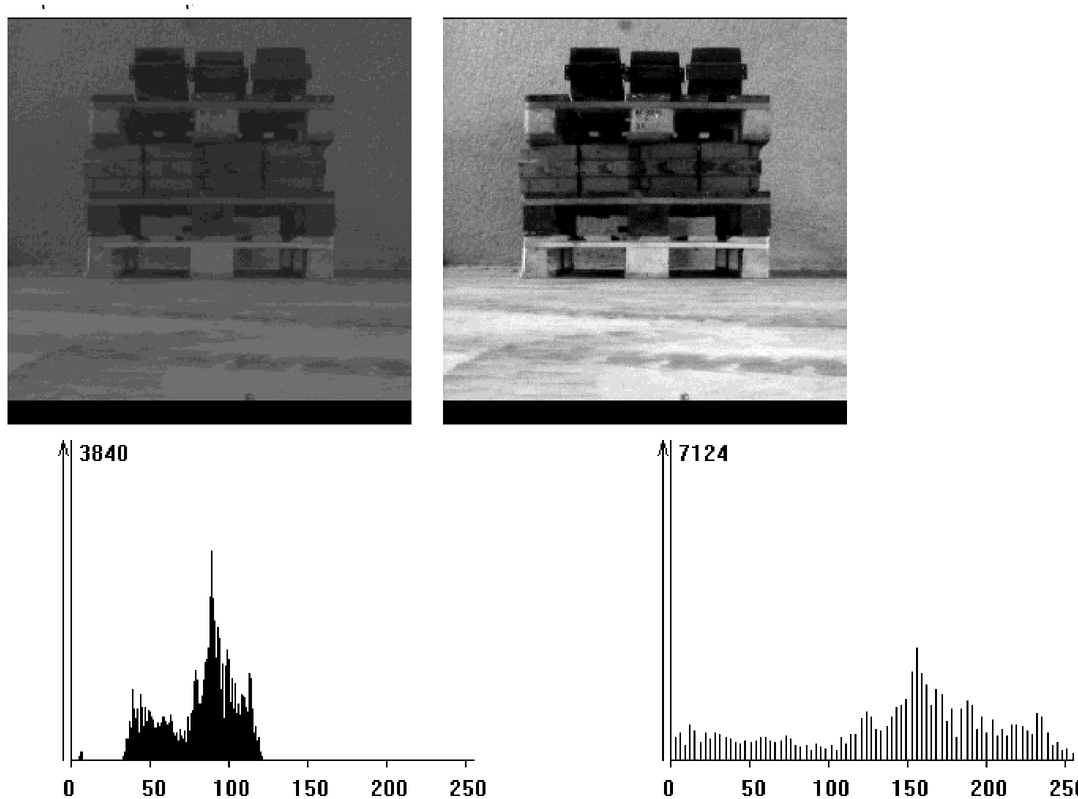


for each pixel **p** it computes $O(\mathbf{p})$:

```
ScaleFactor = (max1 - min1)/(max - min);
if (I(p) <= min)
        O(p) = min1;
else if (I(p) >= max)
        O(p) = max1;
else
        O(p) = (I(p) - min) * ScaleFactor + min1;
```

again $O(\mathbf{p}) = f(I(\mathbf{p})) = s\, I(\mathbf{p}) + k$

Examples



Histogram Equalization

- To improve the appearance of the image for visual enhancement, the histogram can provide useful clues for automatic modifications.
- One solution is equalization, i.e. obtaining a histogram such that all values are used equally, or $H(i) = K$.
- We would like to create a transformation from image x to image y, $y = T(x)$ **to produce a new image *y* with a flat histogram,** such that its cumulative density function *cdf* is a straight across the value range, i.e.

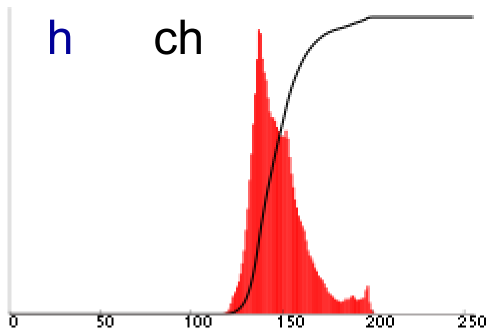
$$cdf_x(i) \stackrel{\text{def}}{=} \sum_{j=0}^{i} p_x(j) \qquad cdf_y(i) = iK$$

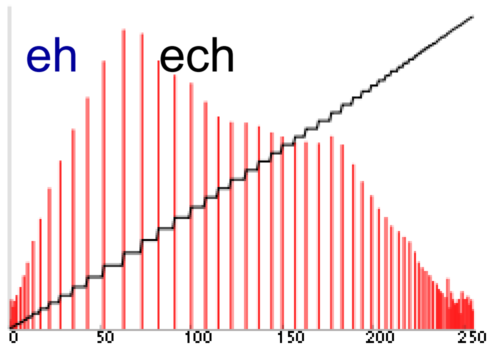
- Such transform is given by:

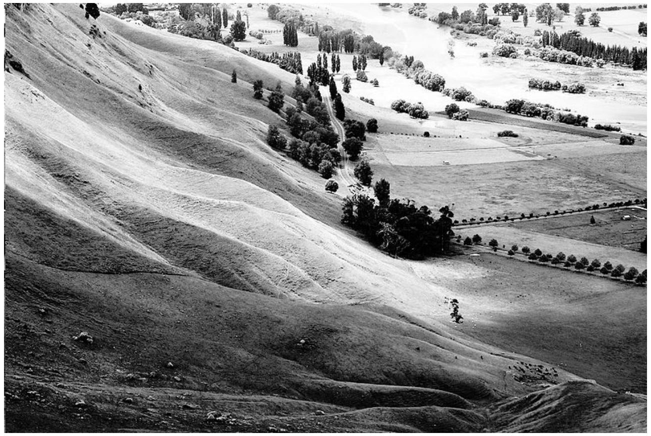
$$y(p) = T\big(x(p)\big) = cdf_x\big(x(p)\big)$$

- This maps the levels into the range $[m, 1]$, with m the probability of the minimum value of the image, so we need to contrast stretch to the desired range.

Histogram Equalization







Thresholding

- **Thresholding**: it consists in the selection of a value T of brightness (intensity) capable of dividing the image into 2 regions of pixels with intensity greater or less than T .
- It is an operator which transforms a grey level image into a binary image (thresholding-based binarization)
- Given an image I(**x**), with **x**=(i,j), it is transformed to O(**x**)
- $O(\boldsymbol{x}) = Thresh(I(\boldsymbol{x}), T)$

```
if (I(i,j) >= T)
      O(i,j) = 1;
else
      O(i,j) = 0;
```

Global threshold if $T = K$ or $T = f(I)$

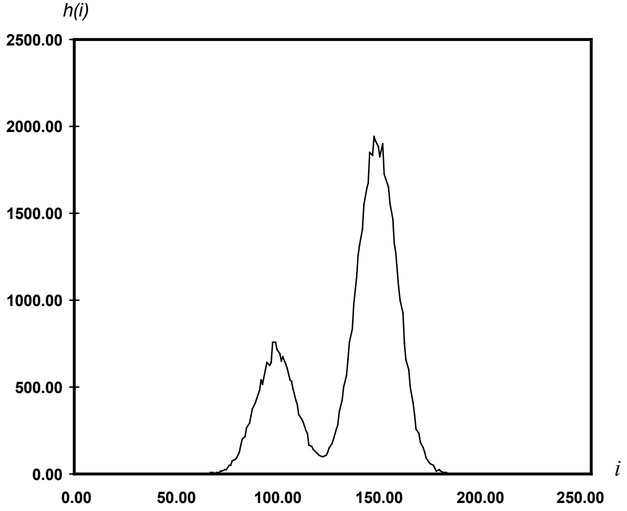
Adaptive threshold if $T = f(I, \mathbf{x})$, i.e. if it depends on a window $W(\mathbf{x})$ around the current position.

Automatic thresholding

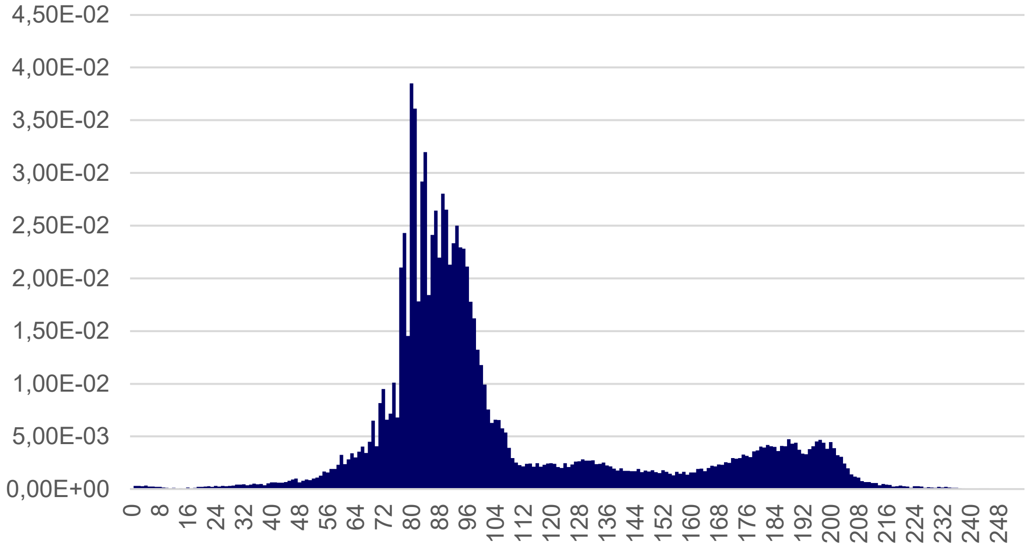
- How to define T automatically?:

- Without knowing the objects of interest let's give the computer the chance of seeing, measuring the statistic proprieties of the histogram
- Hp: the points of the target object have a grey level different from the background gray level (**bimodal histogram**)

  ➔ T must divide the two modes



Example


4,50E-02
4,00E-02
3,50E-02
3,00E-02
2,50E-02
2,00E-02
1,50E-02
1,00E-02
5,00E-03
0,00E+00
0
8
16
24
32
40
48
56
64
72
80
88
96
104
112
120
128
136
144
152
160
168
176
184
192
200
208
216
224
232
240
248

Otsu thresholding

- Histogram is regarded as a probability distribution:

$$p_i = \frac{n_i}{N}$$

- If we threshold at level k, we are making two classes C_0 and C_1, where C_0 has all pixels with levels from 1 to k, and C_1 from $k+1$ to L.
- We can now compute the zeroth- and first-order cumulative moments of the histogram

$$\omega(k) \stackrel{\text{def}}{=} \sum_{i=1}^{k} p_i$$

$$\mu(k) \stackrel{\text{def}}{=} \sum_{i=1}^{k} i p_i$$

- and from these, for every class, the probability of occurrence and the mean value.

Otsu thresholding

- Probability of occurrence

$$\omega_0 \stackrel{\text{def}}{=} Pr(C_0) = \sum_{i=1}^{k} p_i = \omega(k)$$

$$\omega_1 \stackrel{\text{def}}{=} Pr(C_1) = \sum_{i=k+1}^{L} p_i = 1 - \omega(k)$$

- Mean value

$$\mu_0 \stackrel{\text{def}}{=} \sum_{i=1}^{k} iPr(i|C_0) = \sum_{i=1}^{k} \frac{ip_i}{\omega_0} = \frac{1}{\omega_0}\sum_{i=1}^{k} ip_i = \frac{\mu(k)}{\omega(k)}$$

$$\mu_1 \stackrel{\text{def}}{=} \sum_{i=k+1}^{L} iPr(i|C_1) = \sum_{i=k+1}^{L} \frac{ip_i}{\omega_1} = \frac{\mu(L) - \mu(k)}{1 - \omega(k)}$$

- Let's call $\mu(L) = \mu_T$, and stress that

$$\omega_0\mu_0 + \omega_1\mu_1 = \mu_T$$
$$\omega_0 + \omega_1 = 1$$

Otsu thresholding

- The class variances are given by

$$\sigma_0^2 \stackrel{\text{def}}{=} \sum_{i=1}^{k} (i-\mu_0)^2 Pr(i|C_0) = \sum_{i=1}^{k} (i-\mu_0)^2 \frac{p_i}{\omega_0}$$

$$\sigma_1^2 \stackrel{\text{def}}{=} \sum_{i=k+1}^{L} (i-\mu_1)^2 Pr(i|C_1) = \sum_{i=k+1}^{L} (i-\mu_1)^2 \frac{p_i}{\omega_1}$$

- With this we characterized the distributions, and now we need to measure how different/separated they are. We can either check how compact each part is (low *within* class variance, σ_W^2), or how separated they are (high *between* class variance, σ_B^2). Formally

$$\sigma_W^2 \stackrel{\text{def}}{=} \omega_0\sigma_0^2 + \omega_1\sigma_1^2$$

$$\sigma_B^2 \stackrel{\text{def}}{=} \omega_0(\mu_0-\mu_T)^2 + \omega_1(\mu_1-\mu_T)^2$$

- We want both compactness and separation, so we will maximize

$$\lambda = \frac{\sigma_B^2}{\sigma_W^2}$$

Otsu thresholding

- Maximizing λ is not so nice, because we need, for every k to first compute the means and then compute the variances.
- But two observations greatly simplify the task. The first is that if we define

$$\sigma_T^2 \stackrel{\text{def}}{=} \sum_{i=1}^{L} (i - \mu_T)^2 p_i$$

- the following relation holds:

$$\sigma_W^2 + \sigma_B^2 = \sigma_T^2$$

- So we can rewrite λ as follows:

$$\lambda = \frac{\sigma_B^2}{\sigma_W^2} = \frac{\sigma_B^2}{\sigma_T^2 - \sigma_B^2} = \frac{1}{\frac{\sigma_T^2}{\sigma_B^2} - 1}$$

- Maximizing λ is equivalent to minimize $\frac{\sigma_T^2}{\sigma_B^2}$, but the numerator is constant, **so the same result is obtained just by maximizing $\boldsymbol{\sigma_B^2}$.**

Otsu thresholding

- Now we can concentrate on σ_B^2, in order to get the simplest possible version.

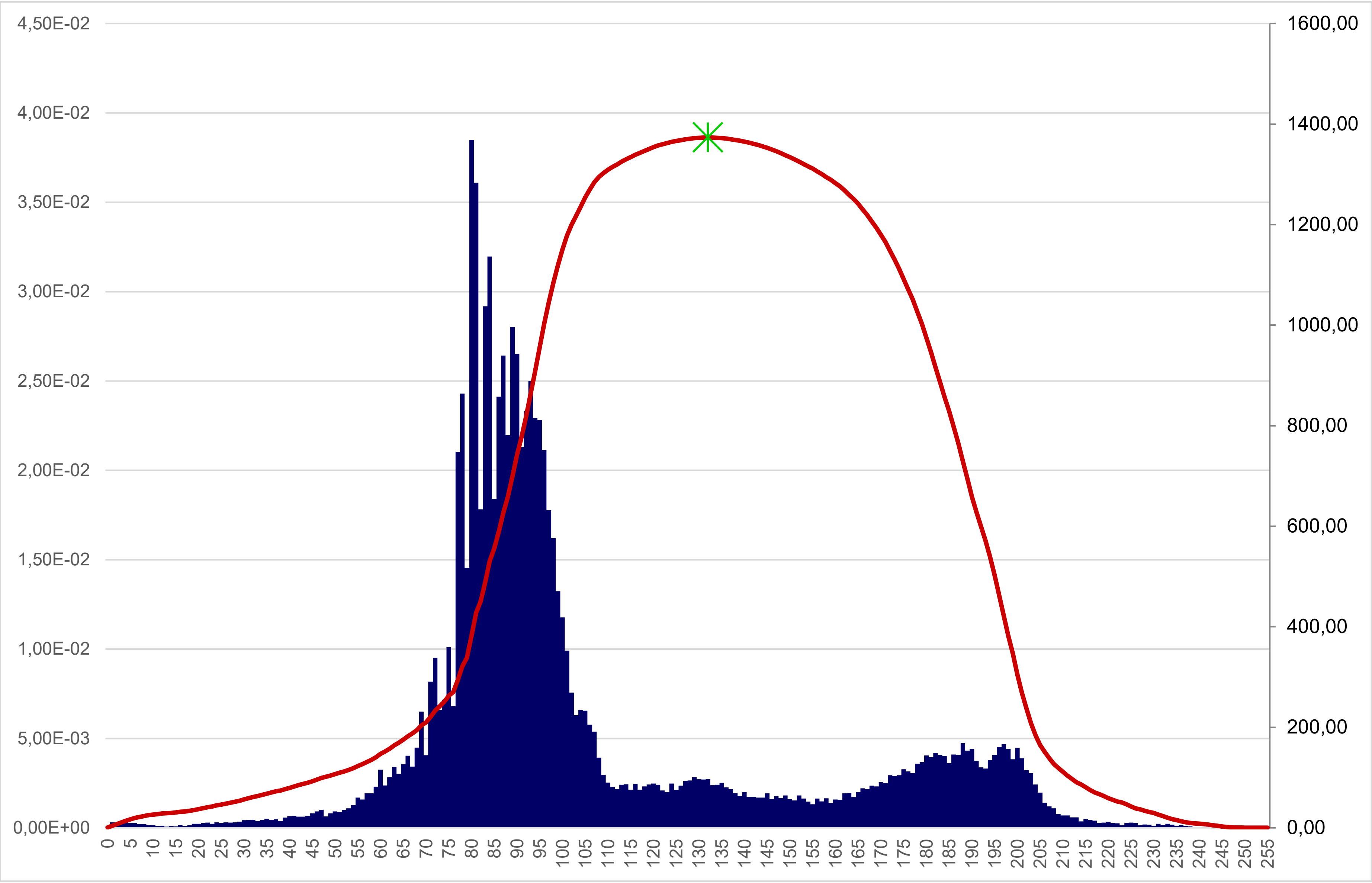
$$\sigma_B^2 = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_T)^2 =$$
$$= \omega_0(\mu_0 - \omega_0\mu_0 - \omega_1\mu_1)^2 + \omega_1(\mu_1 - \omega_0\mu_0 - \omega_1\mu_1)^2 =$$
$$= \omega_0[(1 - \omega_0)\mu_0 - \omega_1\mu_1]^2 + \omega_1[(1 - \omega_1)\mu_1 - \omega_0\mu_0]^2 =$$
$$= \omega_0[\omega_1\mu_0 - \omega_1\mu_1]^2 + \omega_1[\omega_0\mu_1 - \omega_0\mu_0]^2 =$$
$$= \omega_0\omega_1^2[\mu_0 - \mu_1]^2 + \omega_1\omega_0^2[\mu_1 - \mu_0]^2 =$$
$$= \omega_0\omega_1\ (\mu_0 - \mu_1)^2(\omega_1 + \omega_0\ ) =$$
$$= \omega_0\omega_1\ (\mu_0 - \mu_1)^2$$

- So the final quantity to be maximized is given by:

$$\sigma_B^2 = \frac{[\mu_T\omega(k) - \mu(k)]^2}{\omega(k)[1 - \omega(k)]}$$

- The maximization is done by just trying all possible values of k.

Example


4,50E-02
4,00E-02
3,50E-02
3,00E-02
2,50E-02
2,00E-02
1,50E-02
1,00E-02
5,00E-03
0,00E+00
1600,00
1400,00
1200,00
1000,00
800,00
600,00
400,00
200,00
0,00
0 5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100 105 110 115 120 125 130 135 140 145 150 155 160 165 170 175 180 185 190 195 200 205 210 215 220 225 230 235 240 245 250 255

Example

Adaptive thresholding

- Instead of computing T on the whole image, compute $T(i,j)$ for every point (i,j) only in a window $W_{i,j}$ of side $2 \times r + 1$.
- Many simple algorithms:

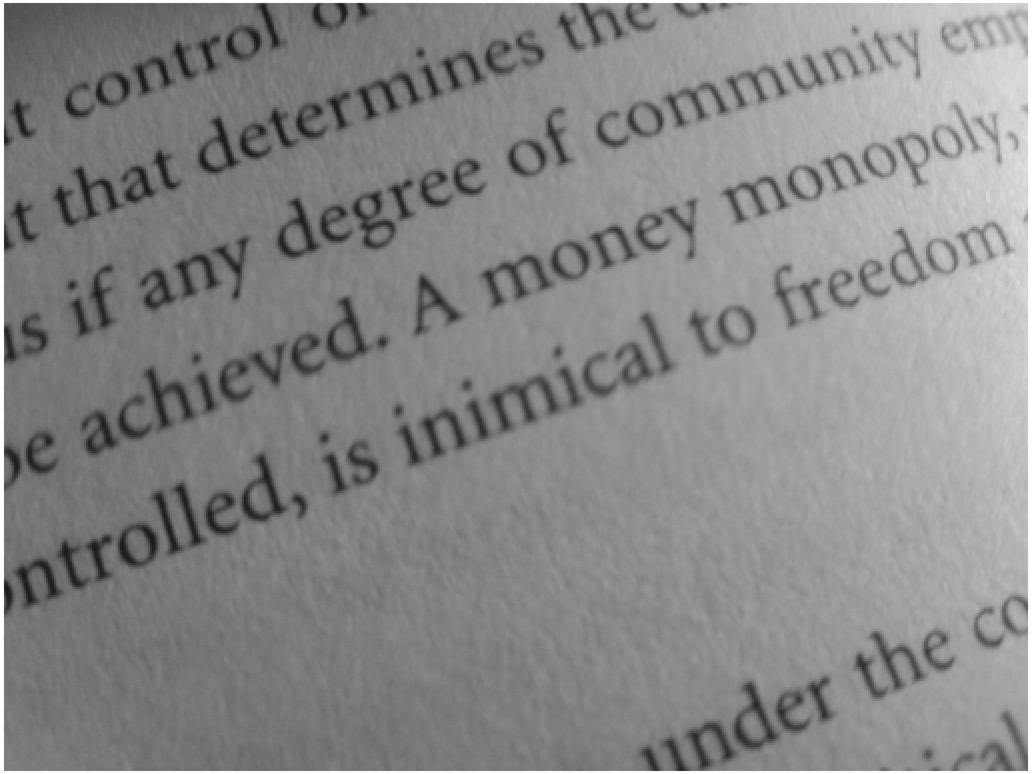
$$T(i,j) = \mathrm{mean}\left(W_{i,j}\right)$$
$$T(i,j) = \mathrm{median}\left(W_{i,j}\right)$$
$$T(i,j) = \left(\max\left(W_{i,j}\right) - \min\left(W_{i,j}\right)\right)/2$$

- Nice results can be obtained by using a variation of the previous ones, lowering them with a constant computed with some global method:

$$T(i,j) = \mathrm{mean}\left(W_{i,j}\right) - C$$

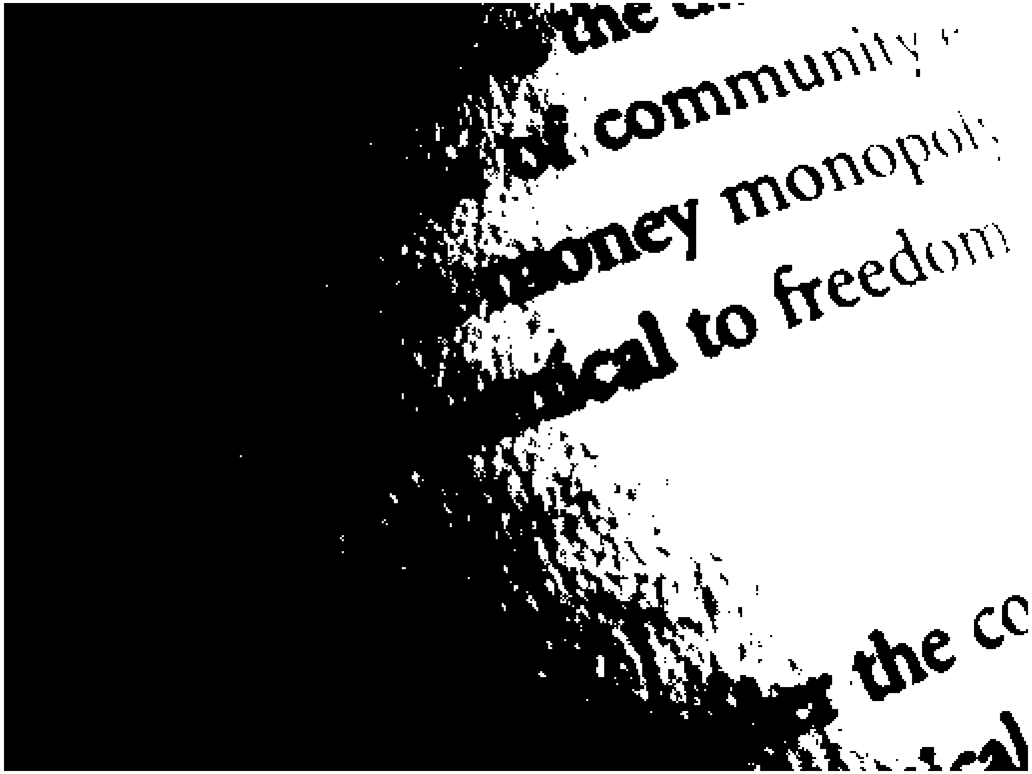
- The constant may also be a user tuned threshold.

Example



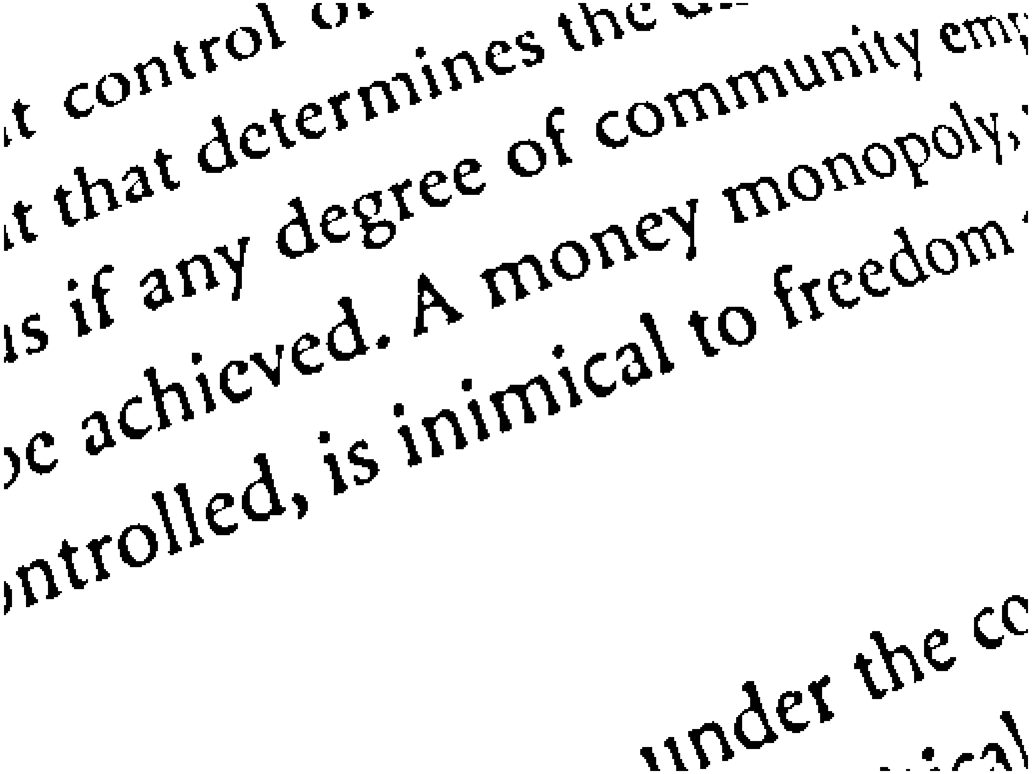
Original image

Example



Otsu thresholding

Example



Adaptive thresholding (mean)

$r = 10, C = 15$

Short Master Machine Learning 2020

Prof. Costantino Grana

Neighborhood operators

Linear filtering

- *Given an initial image F, **linear filtering** consists in a process which gives in output a new image array G , where each location is a weighted sum of the original pixel values from the locations surrounding the corresponding location in the image, <u>using the same set of weights each time</u>.*
- The result is
- **shift-invariant** — meaning that the value depends on the pattern in an image neighborhood, rather than the position of the neighborhood — and
- **linear** — meaning that the output for the sum of two images is the same as the sum of the outputs obtained for the images separately.
- The pattern of weights used for a linear filter is usually referred to as the **kernel** of the filter.
- The process of applying the filter is usually referred to as correlation or convolution (slight difference, but in practice equivalent).

- Depending on the kernel the linear filtering can have effects ad a low pass, high-pass filter etc.

Correlation and convolution

- Local operators or neighborhood operators can be used as a **linear filter.**
- Given an image transformation H: F→G
- A linear filter on the image F produces the output G as the weighted sum of the input pixels, weighted with a **kernel** or mask H that are the **filter coefficients**

$$g(i,j) = \sum_{k,l} f(i+k, j+l)h(k,l)$$

- It is called **cross-correlation**.
- Often we uses its variant with the – instead of + that is called **convolution**, borrowed by signal processing

$$g(i,j) = \sum_{k,l} f(i-k, j-l)h(k,l) = \sum_{k,l} f(k,l)h(i-k, j-l)$$

- They are normally identical in CV since kernel coefficients are usually symmetric.

example

45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

=

69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

Linear filters properties

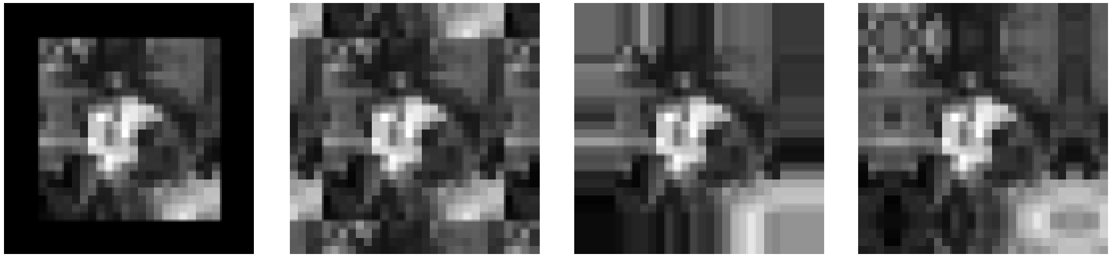
- **linearity**: $h^\circ(f_o + f_1) = h^\circ f_o + h^\circ f_1$
- **scalarity**: h° $(kf) = kh^\circ f$
- **shift invariance**: the answer to a shifted stimulus is the shift of the answer to the stimulus
  - $g(i,j) = f(i+k, j+l) \leftrightarrow (h^\circ g)(i,j) = (h^\circ f)(i+k, j+l)$
- The output value depends on the pixel value and maybe its neighborhood but **not on its position in the image**

- **Not only for punctual operators but also for neighborhood or local operators**
- If shift invariant linear systems → We can apply convolution.

Padding

- What happens when the filter kernel goes outside the image borders?

- Many possibilities of padding, or extension mode in an are where the correct information is not available:
  - **Zero padding**: insert 0 pixels
  - **Constant padding** insert a specific color in the border
  - **Clamp to edge**, repeat the edge value
  - **Wrap** : loop around in a toroidal configuration
  - **Mirror**: reflect the edge



zero wrap clamp mirror

Noise reduction: smoothing

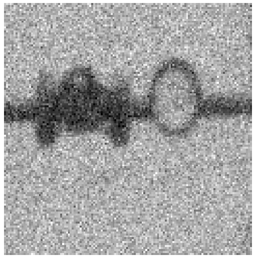
- **Mean filter (moving average filter)** (**smoothing or blurring**)

The simple low pass blurring is given by averaging the pixel with the neighbor ones. It corresponds to **convolving the image with a kernel of 1 values and then scaling .** The size of the filter depends on the size fo the noise frequency w.r.t. the signal spatial frequency

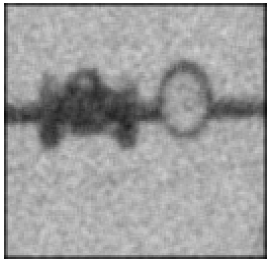
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

3 x 3 kernel

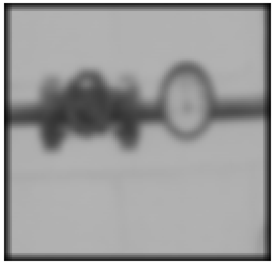
cons: it limits also the information with the same spatial frequency (**blurring**) , does not work on salt-pepper noise



Gaussian noise



Filter 3 x 3



Filter 5 x 5

Moving Average In 2D

$$F[x, y]$$

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

$$G[x, y]$$

	0								

Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0	10							

Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0	10	20						

Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

$G[x, y]$

	0	10	20	30					

Source: S. Seitz

Moving Average In 2D

$F[x, y]$

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

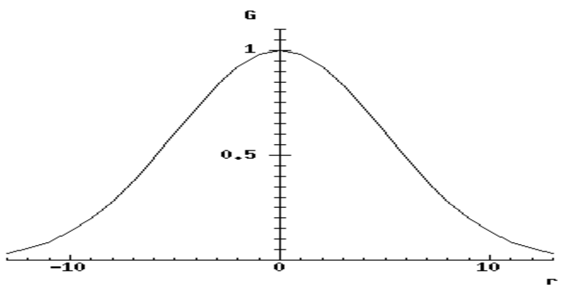
$G[x, y]$

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

Source: S. Seitz

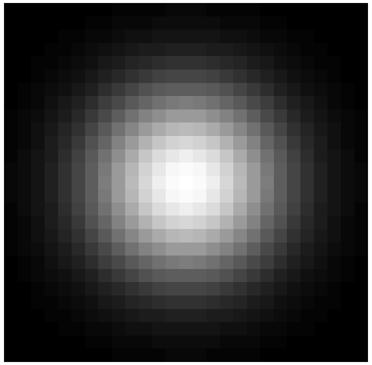
Gaussian Filter

- **The best filter to smooth Gaussian noise**
- Is an isotropic mask given by a Gaussian function with zero average value and a given standard deviation, convolved with the image



$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-x^2/2\sigma^2}$$

- In 2D $G(x, y) = G(x)G(y) = \frac{1}{\sigma^2 2\pi} e^{-(x^2+y^2)/2\sigma^2}$
- Gaussian 13 x 13



Gaussian Filter

The filter must be discretized, choosing **k, that is the size of the filter, and the standard deviation:**

Mask k x k , **k about 5 σ** (cover 98.7%)

e.g. σ=1 k=5

h =

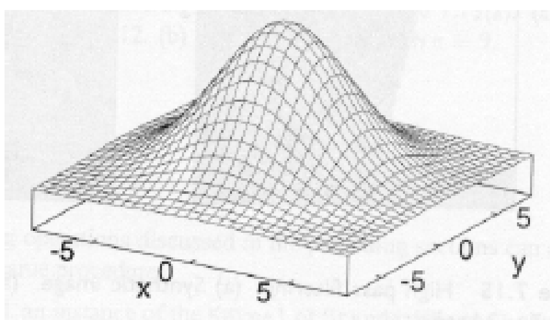
0.0029	0.0131	0.0215	0.0131	0.0029
0.0131	0.0585	0.0965	0.0585	0.0131
0.0215	0.0965	0.1592	0.0965	0.0215
0.0131	0.0585	0.0965	0.0585	0.0131
0.0029	0.0131	0.0215	0.0131	0.0029

- Best values
- σ=1 7 X 7
- σ=2 13 X 13
- σ=3 19 X 19
- *It is computational severe but now is always adopted*

Smoothing: Gaussian filter

- The weights are samples of the Gaussian function

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



7×7 Gaussian mask

1	1	2	2	2	1	1
1	2	2	4	2	2	1
2	2	4	8	4	2	2
2	4	8	16	8	4	2
2	2	4	8	4	2	2
1	2	2	4	2	2	1
1	1	2	2	2	1	1

$\sigma = 1.4$

mask size: $height = width = 5\sigma$ (subtends 98.76% of the area)

The simplest

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

$F[x, y]$

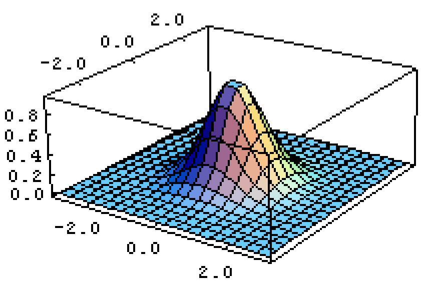
$\frac{1}{16}$

1	2	1
2	4	2
1	2	1

$H[u, v]$

This kernel is an approximation of a Gaussian function:

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



Smoothing: Gaussian

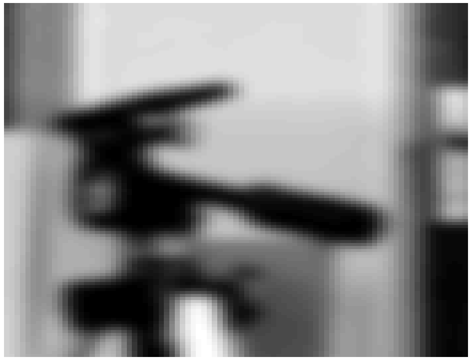
- As σ increases, more samples must be obtained to represent the Gaussian function accurately.
- Therefore, σ controls the amount of smoothing

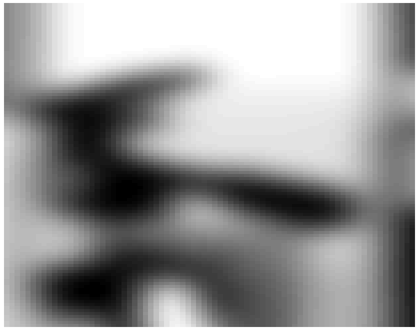
$\sigma = 3$

15 × 15 Gaussian mask

2	2	3	4	5	5	6	6	6	5	5	4	3	2	2
2	3	4	5	7	7	8	8	8	7	7	5	4	3	2
3	4	6	7	9	10	10	11	10	10	9	7	6	4	3
4	5	7	9	10	12	13	13	13	12	10	9	7	5	4
5	7	9	11	13	14	15	16	15	14	13	11	9	7	5
5	7	10	12	14	16	17	18	17	16	14	12	10	7	5
6	8	10	13	15	17	19	19	19	17	15	13	10	8	6
6	8	11	13	16	18	19	20	19	18	16	13	11	8	6
6	8	10	13	15	17	19	19	19	17	15	13	10	8	6
5	7	10	12	14	16	17	18	17	16	14	12	10	7	5
5	7	9	11	13	14	15	16	15	14	13	11	9	7	5
4	5	7	9	10	12	13	13	13	12	10	9	7	5	4
3	4	6	7	9	10	10	11	10	10	9	7	6	4	3
2	3	4	5	7	7	8	8	8	7	7	5	4	3	2
2	2	3	4	5	5	6	6	6	5	5	4	3	2	2

Examples

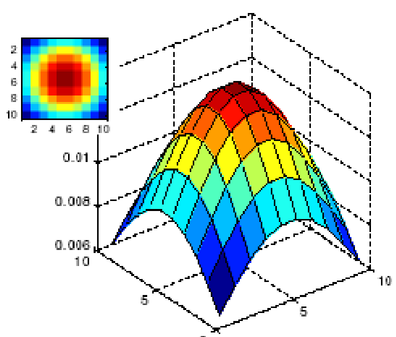


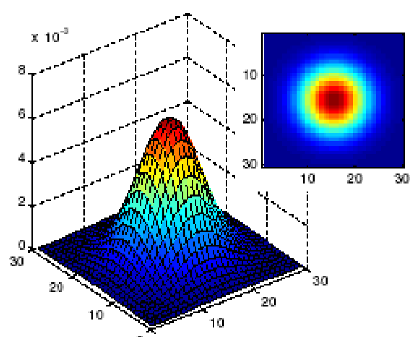


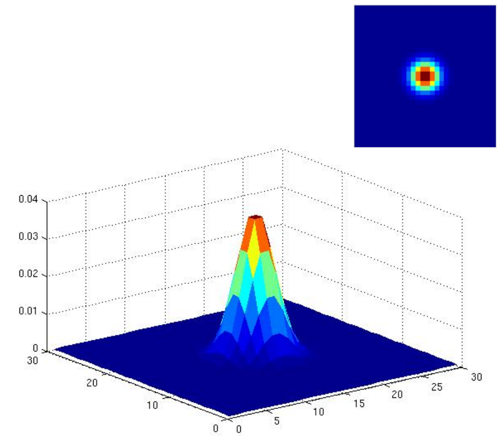
comments

- We must select
- **Size** of kernel or mask: Gaussian function has infinite support, but discrete filters use finite kernels
- Eg 10x10 vs 30x30 with sigma=5

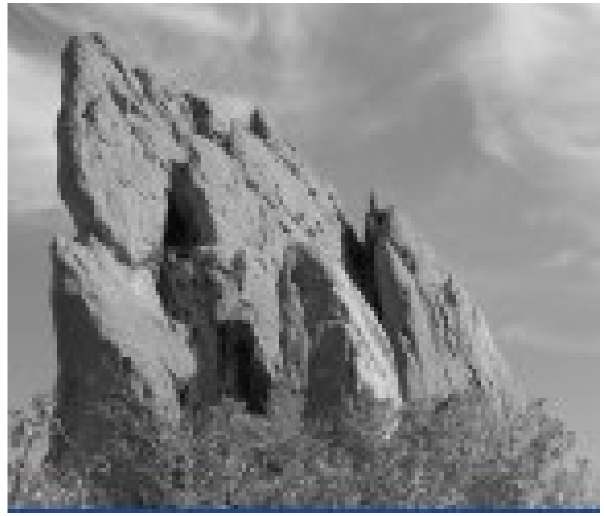
- **Variance** of Gaussian: determines extent of smoothing
- 30x30 with sigma=2

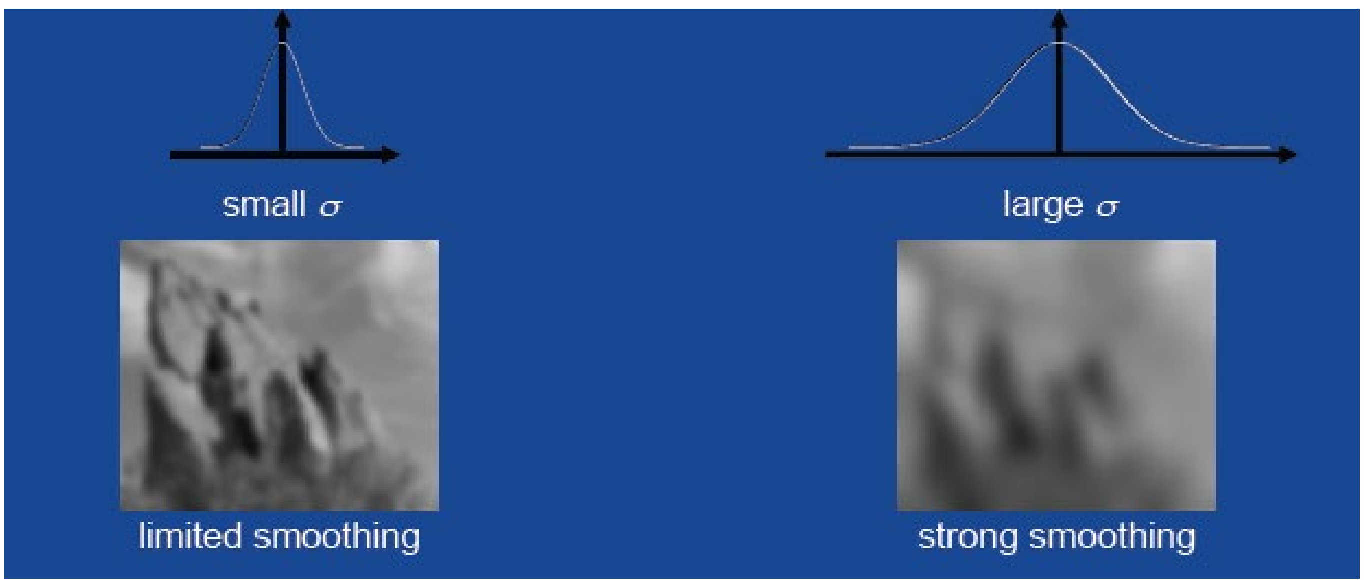




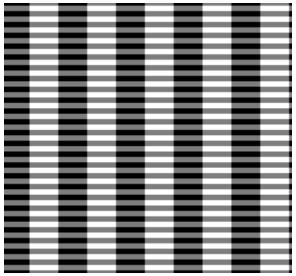


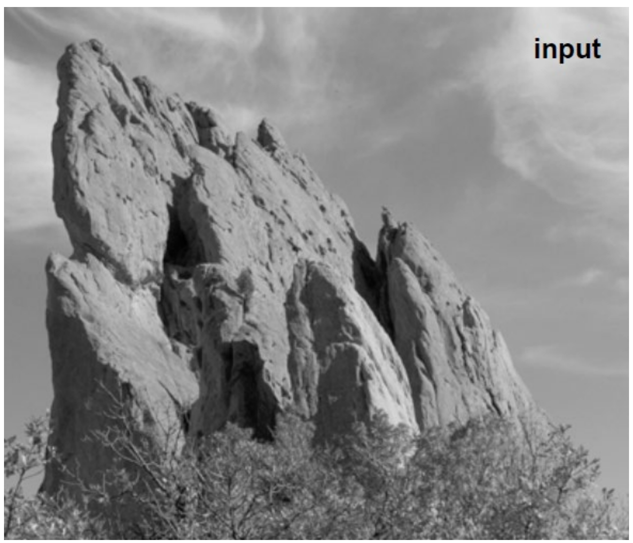
example

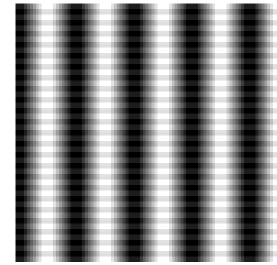




Averaging vs Gaussian Smoothing

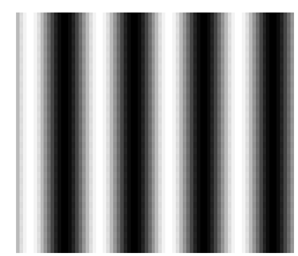






Averaging

Gaussian



Gaussian Smoothing

The Gaussian smoothing is provided by our

Vision system

In our eyes, by lens, depending

on the distance of observation

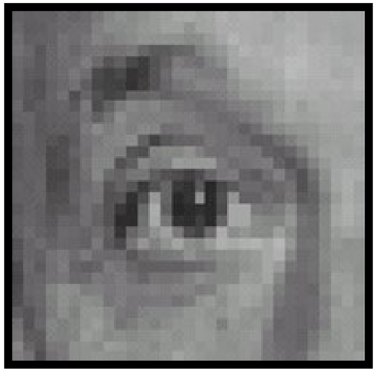
by Charles Allen Gillbert

All is Vanity

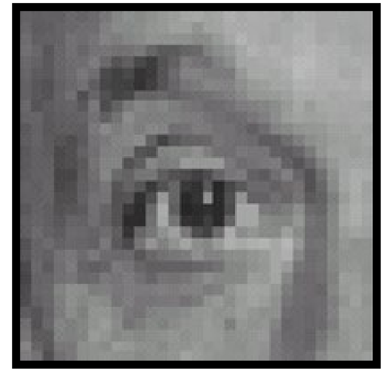
http://www.michaelbach.de/ot/cog_blureffects/index.html

Gaussian Smoothing

Predict the filtered outputs

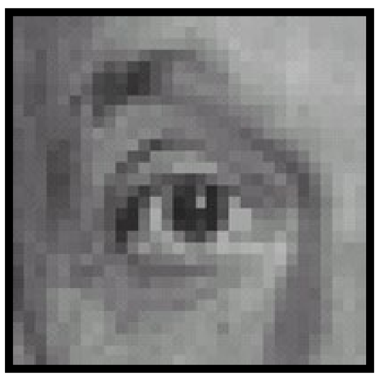


$$* \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} \; ?$$



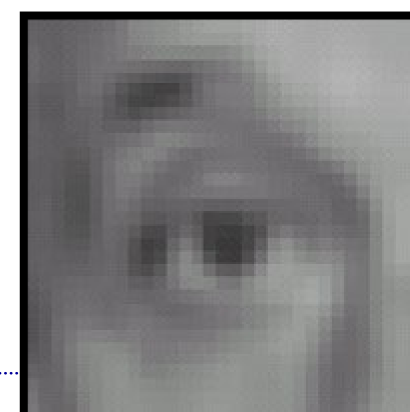
$$* \begin{bmatrix}0&0&0\\0&0&1\\0&0&0\end{bmatrix} = ?$$

filter

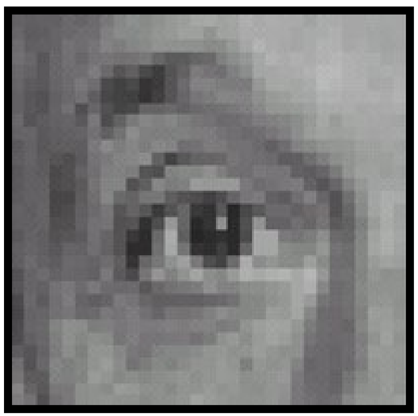


$$* \begin{bmatrix}0&0&0\\0&2&0\\0&0&0\end{bmatrix} - \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix} = ?$$

?

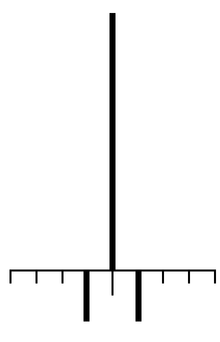


Practice with linear filters



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

Source: D. Lowe

High pass filter

Using this mask of convolution

-1	-1	-1
-1	9	-1
-1	-1	-1

A **high pass filter** is achieved for sharpening

- Useful for emphasizing transitions in image intensity (e.g., edges).
- Note that the response of high-pass filtering might be negative.
- Values must be re-mapped to [0, 255]

sharpen

A>=1
w = 9A-1

-1	-1	-1
-1	w	-1
-1	-1	-1

A=2
w = 17

-1	-1	-1
-1	17	-1
-1	-1	-1

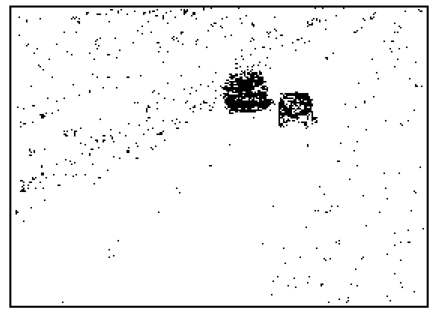
Median filter

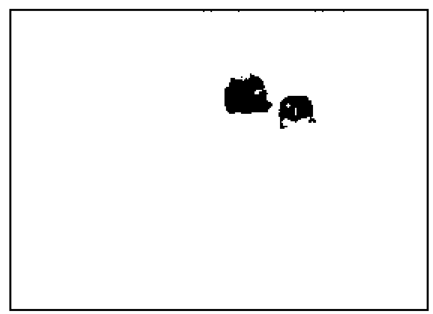
- **Median filter** NON LINEAR filter useful for impulsive noise
- The output pixel is the median value in the neighborhood

120 123 123 130 128
121 123 128 130 128
120 125 **146** 132 126
129 120 123 122 130
120 123 123 120 129

120 122 123 123 **125** 128 130 132 146

120 123 123 130 128
121 123 128 130 128
120 125 **125** 132 126
129 120 123 122 130
120 123 123 120 129





Median filter

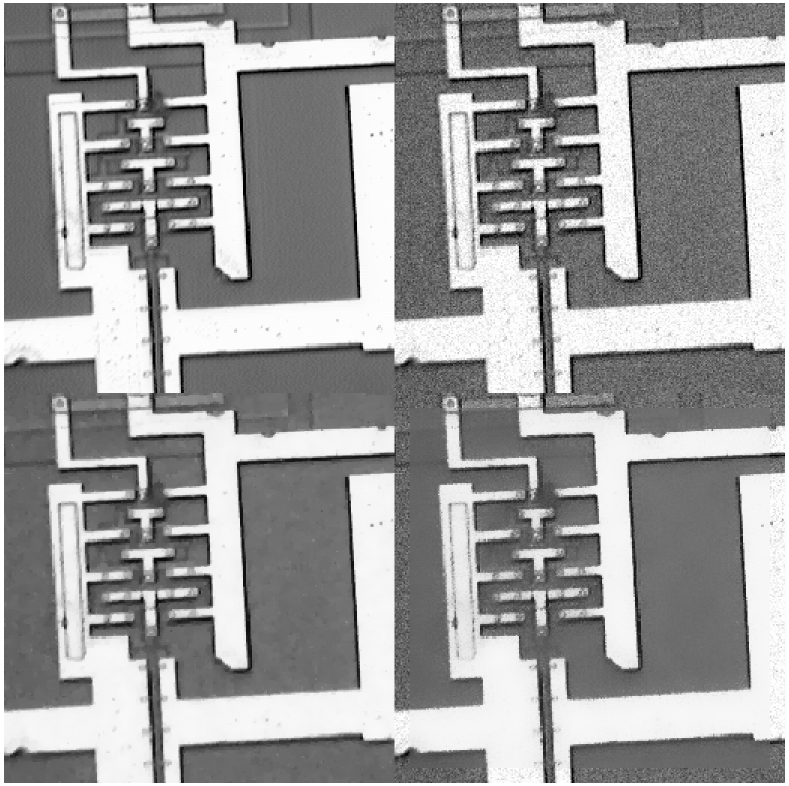
- Discard the outliers: the **median filter** for salt and pepper noise

```
1) Consider a 2D neighborhood
2) Order it
3) Choose the central value
4) Substitute pixel with the median one
```

- This implementation will be slow. Specialized algorithms exist to speed up the process.

Bilateral filters

- Bilateral filters combine a weighted filter kernel with outlier rejection
- **-Every sample is replaced by a weighted average of its neighbors**.
- These weights reflect two forces
  - How close are the neighbor and the center sample, so that larger weight to closer samples,
  - How similar are the neighbor and the center sample – larger weight to similar samples.
- All the weights should be normalized to preserve the local mean.



bilateral

Bilateral filter

- Improved weighted filter
- In a neighborhood of f(i,j) the result g(i,j) is a normalized weighted sum

$$g(i,j) = \frac{\sum_{k,l} f(k,l)w(i,j,k,l)}{\sum_{k,l} w(i,j,k,l)}$$

- Weights are given by

$$w(i,j,k,l) = \exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2} - \frac{\|f(i,j)-f(k,l)\|^2}{2\sigma_r^2}\right)$$

$$d(i,j,k,l) = \exp\left(-\frac{(i-k)^2+(j-l)^2}{2\sigma_d^2}\right)$$ Domain kernel

$$r(i,j,k,l) = \exp\left(-\frac{\|f(i,j)-f(k,l)\|^2}{2\sigma_r^2}\right)$$ Range kernel

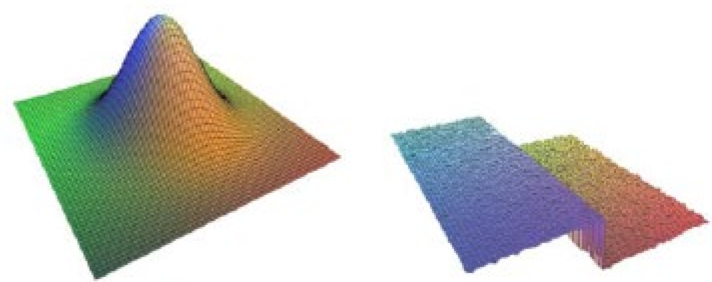
	2	1	0	1	2
2	0.1	0.3	0.4	0.3	0.1
1	0.3	0.6	0.8	0.6	0.3
0	0.4	0.8	1.0	0.8	0.4
1	0.3	0.6	0.8	0.6	0.3
2	0.1	0.3	0.4	0.3	0.1

(c) domain filter

0.0	0.0	0.0	0.0	0.2
0.0	0.0	0.0	0.4	0.8
0.0	0.0	1.0	0.8	0.4
0.0	0.2	0.8	0.8	1.0
0.2	0.4	1.0	0.8	0.4

(d) range filter

- In color the range kernel is a **vector distance**



(b) (c)

Bilateral filter

- Remove texture

→
[Bilateral Filtering] for Gray and Color Images
www.cs.duke.edu/~tomasi/.../tomasiIccv98.pdf

Short Master Machine Learning 2020

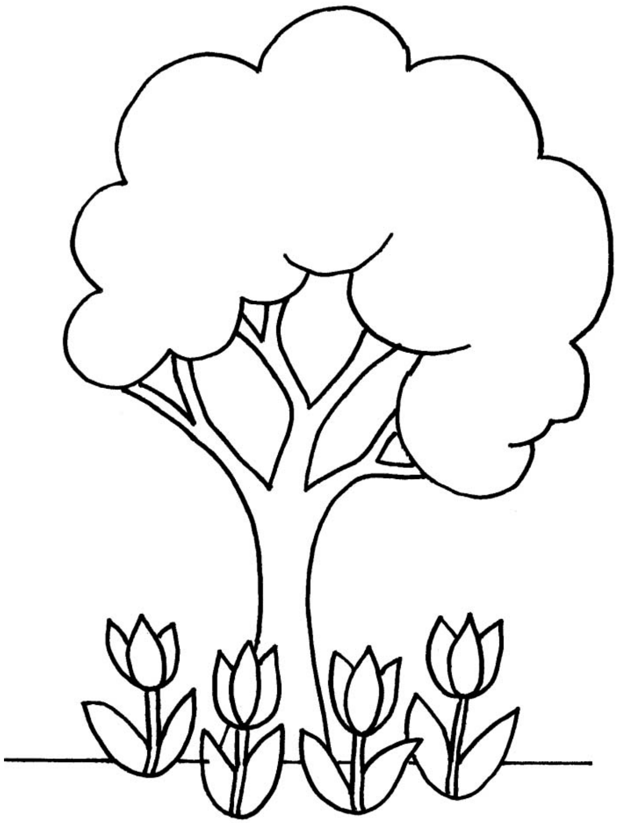
Prof. Costantino Grana

Edge detectors

Contours

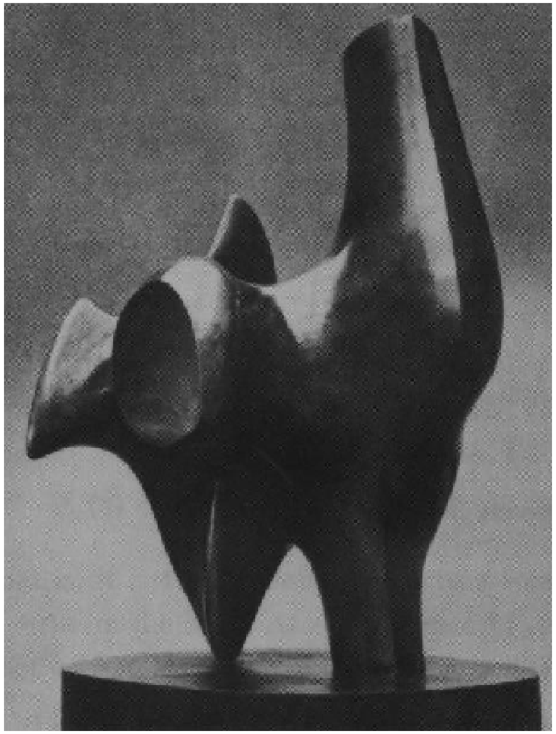
- We perceive shape by strong luminance variation
- We recognize objects by contours

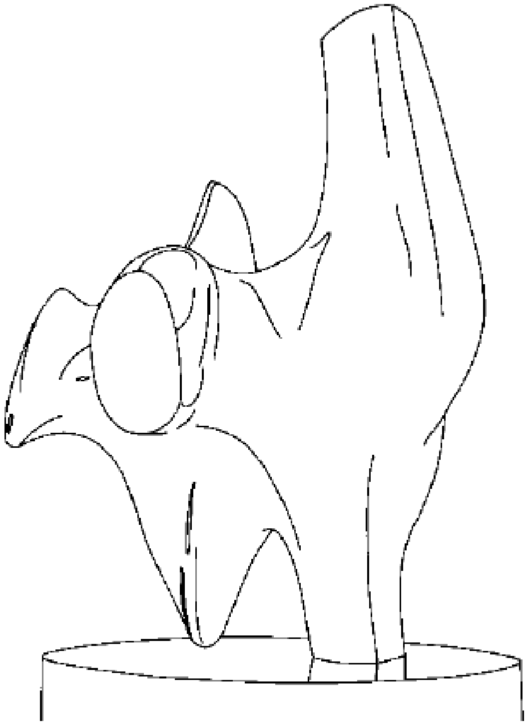
- How can we compute contours?



Borders

- An important vision task: border detection to
- Convert an image in a shape representation
- Convert a 2D image into a set of curves
- Extract salient features of the scene more compact than pixels





Boundary detection

- Task easy for human (but often subjective) but a challenging problem for machine
- Edges occur at boundary between homogeneous region, but often segmentation is difficult so that edge detection is provided **using local variation**

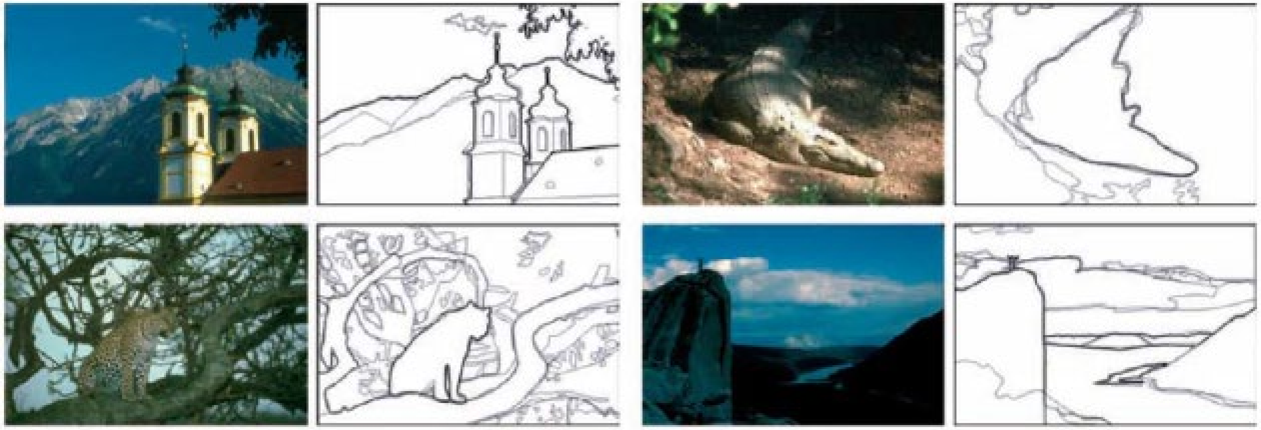
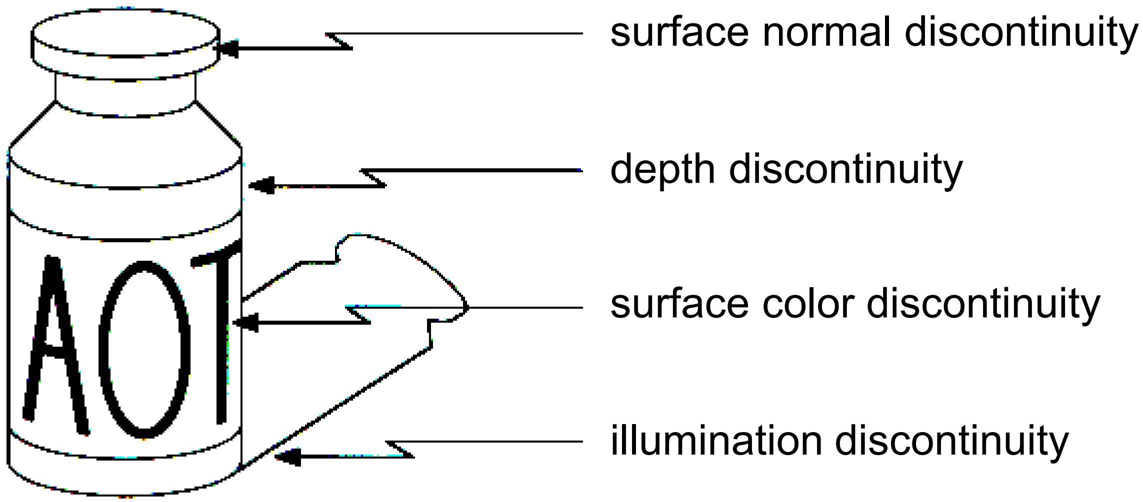


Figure 4.31 Human boundary detection (Martin, Fowlkes, and Malik 2004) © 2004 IEEE. The darkness of the edges corresponds to how many human subjects marked an object boundary at that location.

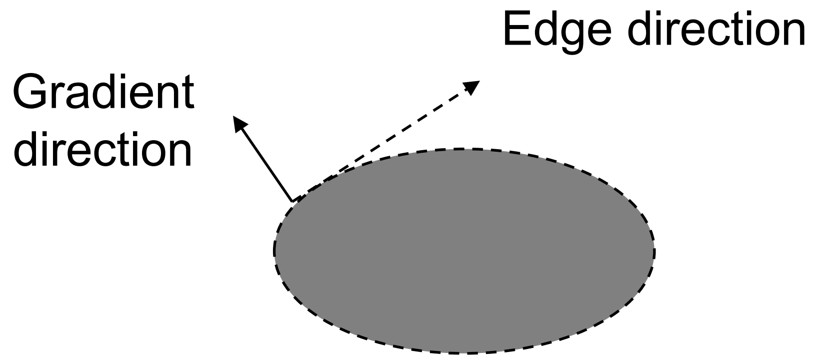
Edge and borders

- Many physical causes for edges:



EDGE

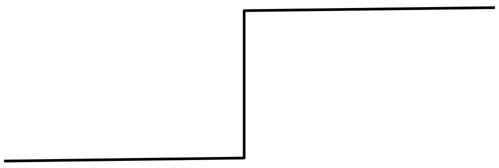
- EDGE: is a local proprety of a pixel and its neighborhood to have a «rapid intensity variation» (or it is the location where the rapid intensity variation occurs)
- Edge it is a VECTOR with a magnitude and a direction
- It depends on the luminance variation: We can compute this luminance variation as a gradient . The edge has the direction perpendicular to the gradient direction



- The BORDER is a propretey af a Region while Edge is a local propriety
- We can compute borders by selecting the high edges

Edge detection

- Edge: point or set of points where there is a «high» gradient.



- Border detection:
1. Use of an edge detection operator (edge detector)
2. Selection of strong edges with some given criteria
3. Linking the edge

- Problem: noise creates many false edge points

Edge detection

- Many algorithms

1. Methods based on first derivative computation
   - gradient masks (Roberts, Sobel..)

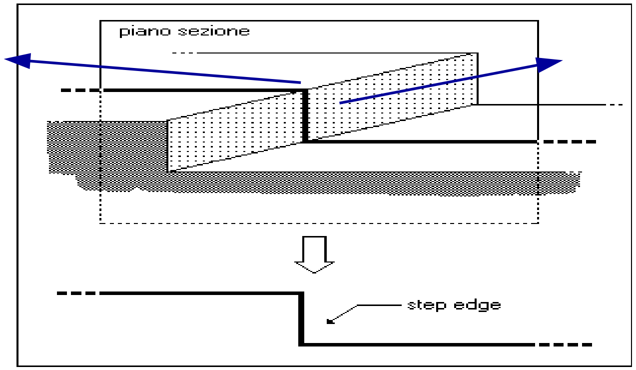
2. Regularization techniques using filtering and optimal masks
   - Canny
   - Sarkar-Bowyler
   - Marr-Hildreth

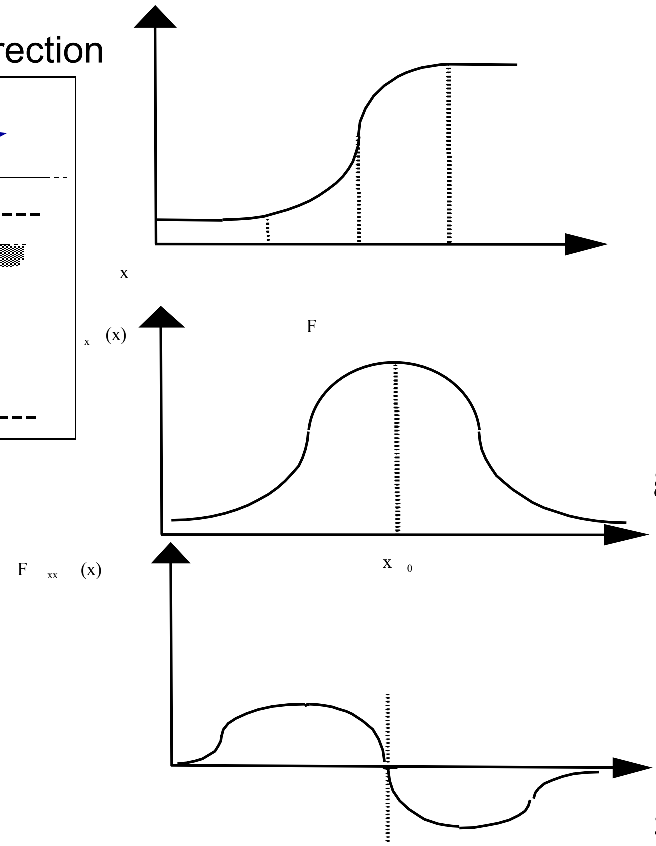
3. Border following local techniques based on neighborhood operation of labeled edges

Edge: ideal case

Gradient direction

Edge direction





Luminosity

luminosity gradient

second derivative Laplacian

Change is measured by derivative in 1D

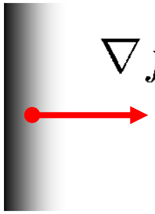
Biggest change, derivative has maximum magnitude

Or 2nd derivative is zero.

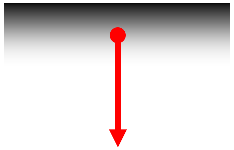
Image gradient

- The gradient of a 2D continuous function f(x,y) :

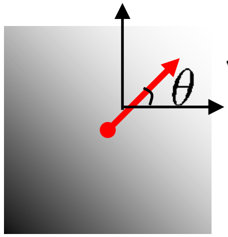
$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$



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



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



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

The **gradient direction** is given by:

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

the direction of the edge is perpendicular to the gradient direction

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

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

Discrete detectors

- In one dimension: Three forms are commonly considered: forward, backward, and central differences.
- A **forward difference** is an expression of the form $\Delta_h[f](x) = f(x+h) - f(x).$
- A **backward difference** uses the function values at *x* and *x* − *h*, instead of the values at *x* + *h* and *x*: $\nabla_h[f](x) = f(x) - f(x-h).$
- The **central difference** is given by

$$\delta_h[f](x) = f(x + \tfrac{1}{2}h) - f(x - \tfrac{1}{2}h).$$

- We can compute central difference given the function f(x,y) and the discretization f(r,c)

$$\frac{\partial f(r,c)}{\partial r} = f(r+1,c) - f(r-1,c)$$

$$\frac{\partial f(r,c)}{\partial c} = f(r,c+1) - f(r,c-1)$$

- Corresponding to the convolution masks:

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

The Sobel Operators

- Better approximations of the gradients exist, but the most commonly used is called **Sobel operator**; it uses a first central derivative , smoothed in the opposite direction
  - The *Sobel* operators below are commonly used

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

s_x

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

s_y

  - Still better : **Frei and Chen** operator

$$\begin{matrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{matrix} \qquad \begin{matrix} -1 & 0 & 1 \\ -\sqrt{2} & 0 & \sqrt{2} \\ -1 & 0 & 1 \end{matrix}$$

(e)

Comparing Edge Operators

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

Good Localization
Noise Sensitive
Poor Detection

Roberts (2 x 2):

0	1
-1	0

1	0
0	-1

Prewitt (3 x 3):

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

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

Sobel (3 x 3)

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

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

Sobel (5 x 5):

-1	-2	0	2	1
-2	-3	0	3	2
-3	-5	0	5	3
-2	-3	0	3	2
-1	-2	0	2	1

1	2	3	2	1
2	3	5	3	2
0	0	0	0	0
-2	-3	-5	-3	-2
-1	-2	-3	-2	-1

Poor Localization
Less Noise Sensitive
Good Detection

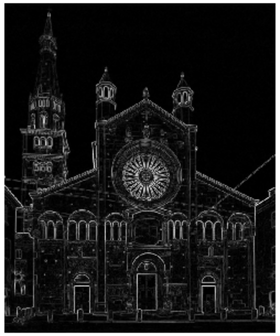
EDGE detectiOn with SOBEL

For each point :

SOBELx
Sx

SOBELy
Sy

$$G = \sqrt{[Sx]^2 + [Sy]^2}$$



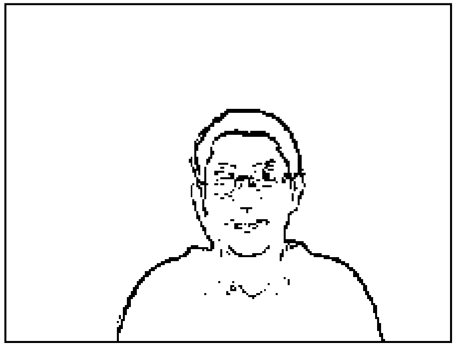
G>Th ?

Edge=0

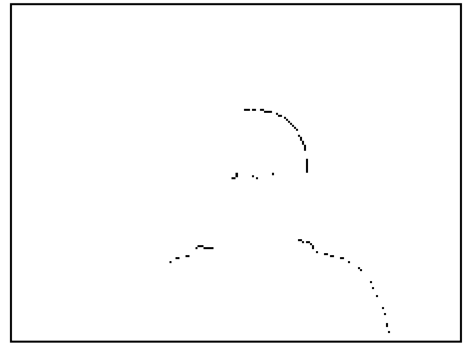
Edge=1

Examples

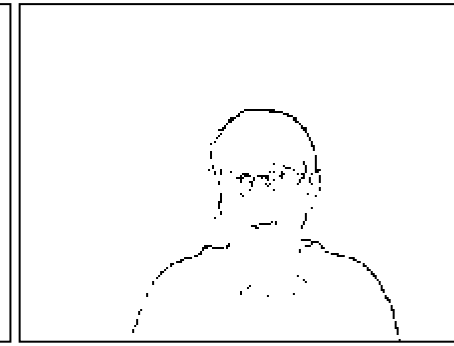
Original



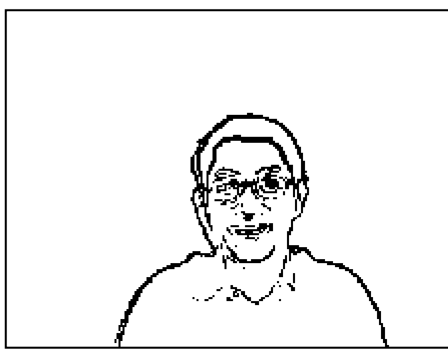
Prewitt,
Th = 100



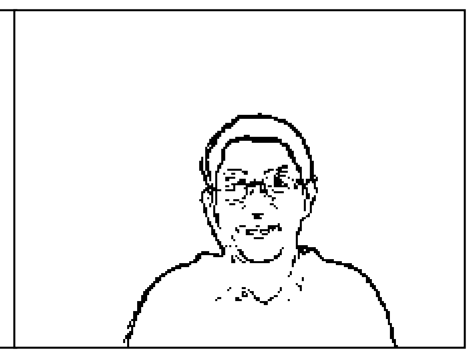
Roberts,
Th = 100



Roberts,
Th = 50



Sobel,
Th = 100

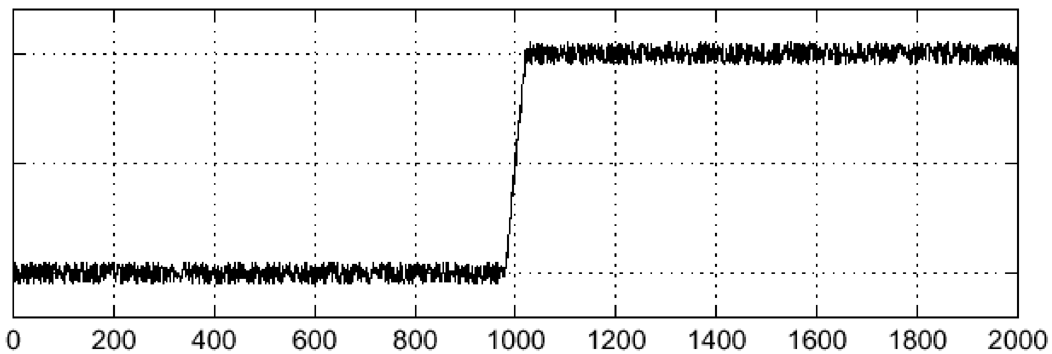


Frei and Chen
Th = 100

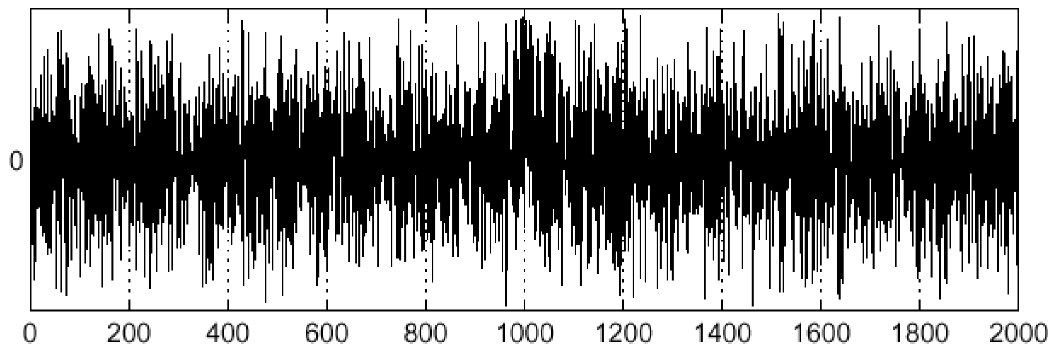
Effects of noise

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

$$f(x)$$



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



Where is the edge? Using the first derivative is very sensitive to noise

smoothing

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

- What is to be done?
- • **Smoothing the image should help**, by forcing pixels different to their neighbors (=noise pixels?) to look more like neighbors

- Smoothing and then derivate
- Derivate of a smoothing filter
- →Derivate of a gaussian **DOG**

Edges: effect of orientation

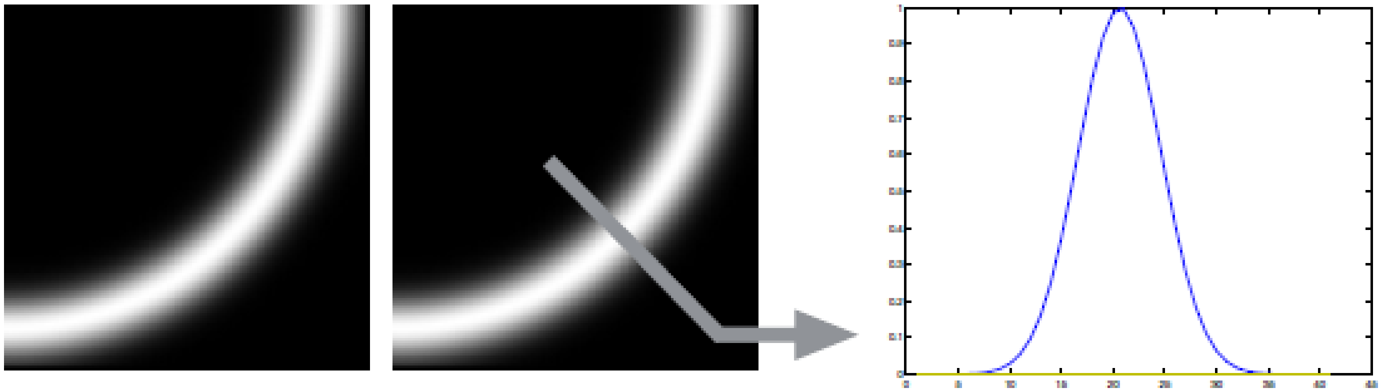


Figure 8.9. The gradient magnitude tends to be large along thick trails in an image. Typically, we would like to condense these trails into curves of representative edge points. A natural way to do this is to cut the trail perpendicular to its direction and look for a peak. We will use the gradient direction as an estimate of the direction in which to cut. The top left figure shows a trail of large gradient magnitude; the figure on the top right shows an appropriate cutting direction; and below, we show the peak in this direction.

From forsyth

CRITERIA (Canny)

A good operator should have three crteria

1) **GOOD DETECTION**

search for low error probability in recognizing true edges and recognizing false edges. If non-edge is considered noise, both probabilities are monotonic descent function with the SNR, this criterium corresponds to try *to maximize the signal-to-noise ratio* (good detection→high SNR, high recall)

2) **GOOD LOCALIZATION**

The selected edge points by the operator must be more closed as possible to the true edge , to have *perfect localization* (good localization→precise position)

3) **ONE RESPONSE TO SINGLE EDGE**

If there is a single edge the operator should return a single edge too and *low false positives* (one response →high precision) (partially in the first criterium since if we have two answers, one is a false one!)

The edge selection is a ill-posed problem!!

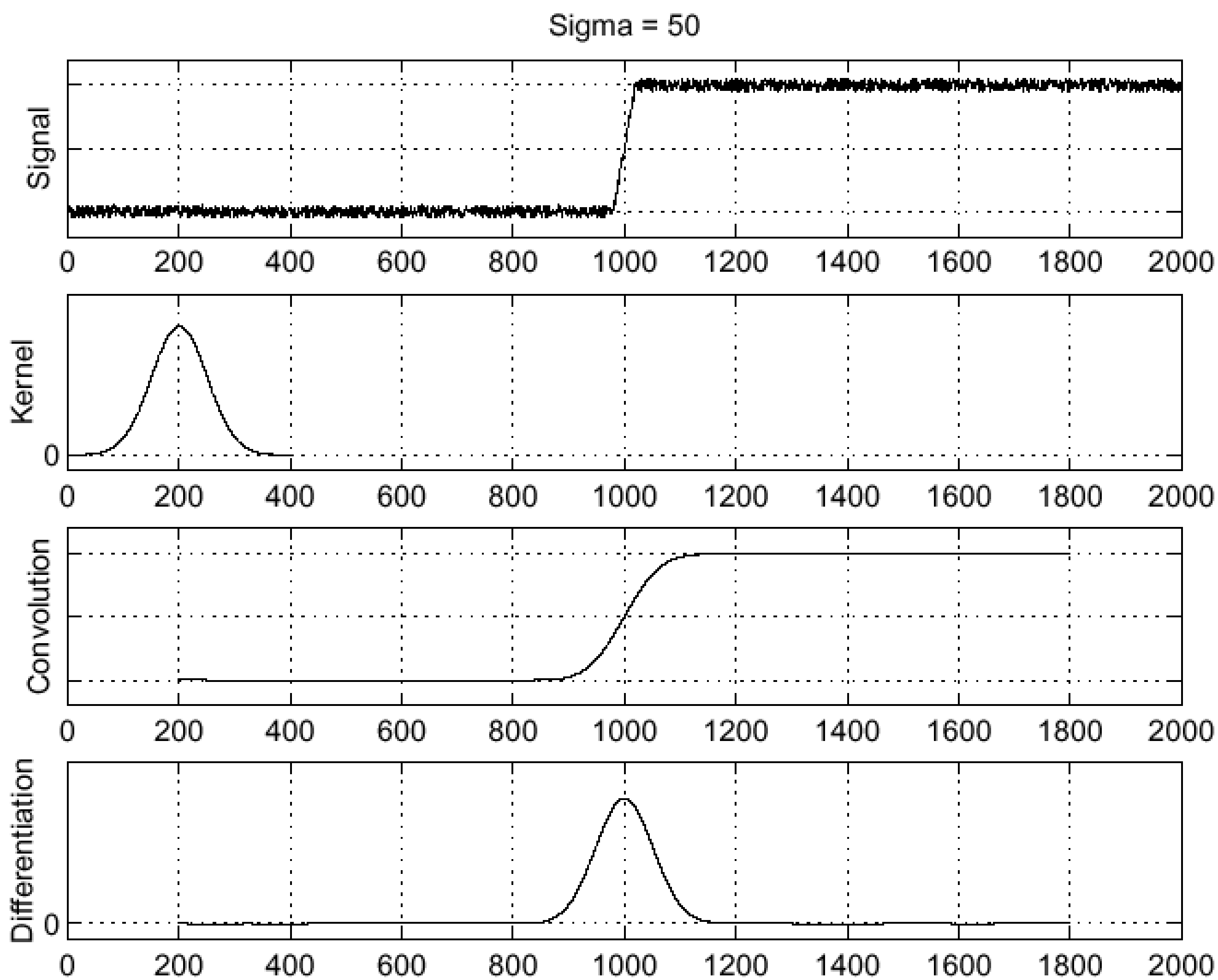
Solution: smooth first

f

h

$h \star f$

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



Where is the edge?

Look for peaks in $\frac{\partial}{\partial x}(h \star f)$

Canny proposal

- Define the best continuous filter (in 1D) and then discretize it and extend to 2D

A. It needs **a smoothing filter** to keep high the SNR

B. It needs **to find the true direction** of gradients to extract only one edge

C. It needs to **suppress the false edges**

- Smoothing filter? Gaussian!
- And then use the **derivative of gaussian**!

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

Canny Edge Operator

- Algorithm:

A. Smooth image with 2D Gaussian:

B. Find local edge normal directions for each pixel

C. Compute edge magnitudes

D. Locate edges by finding zero-crossings along the edge normal directions (non-maximum suppression): search for points which cross zero with second derivative

E. Hysteresis-thresholding

DOG

A. Regularization with a smooth function

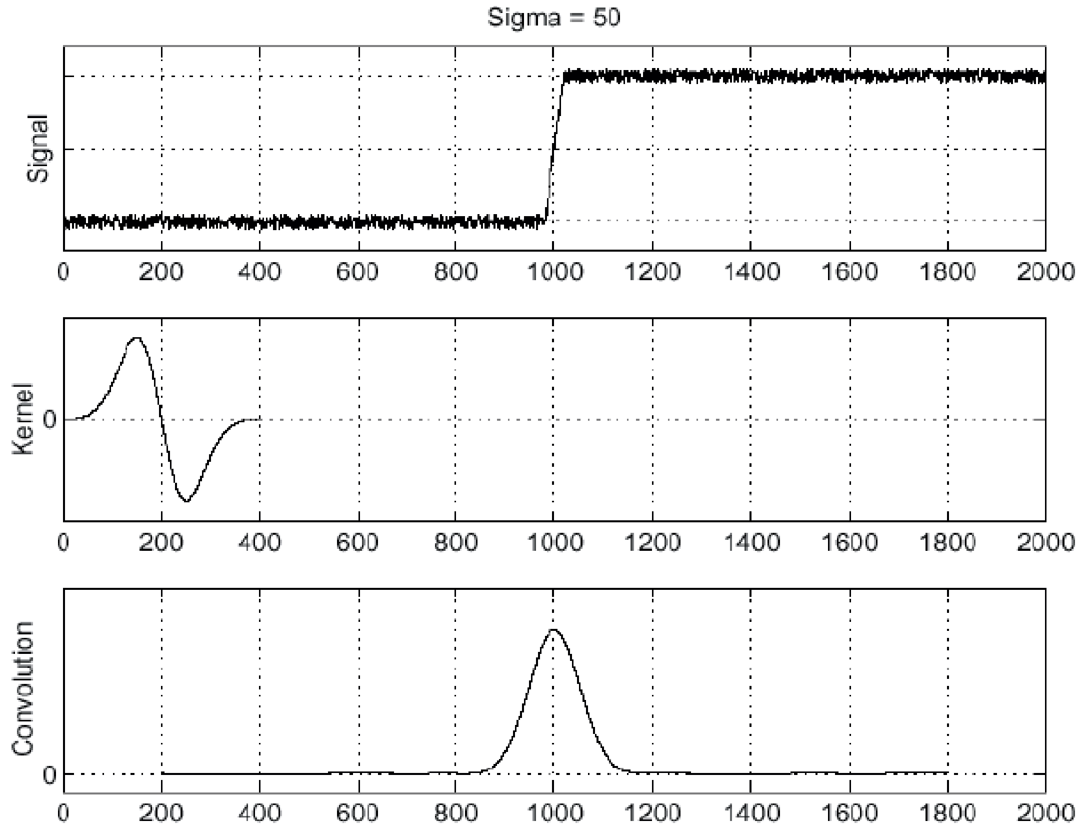
- 1.2.3. points: find the direction and the magnitude of the gradient of the gaussian convolution of the image
- Smoothing and image and then differentiating is the same as convolving the image with the derivative of the smoothed kernel.
- $\frac{\partial (G_\sigma * * I)}{\partial x} = (\frac{\partial G_\sigma}{\partial x}) * * I$ that we can do a single convolution.

DOG

$$f$$

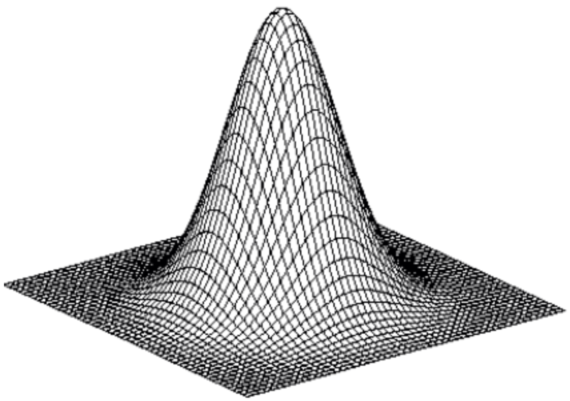
$$\frac{d}{dx}g$$

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

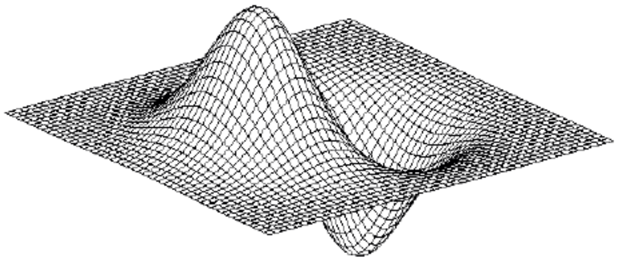


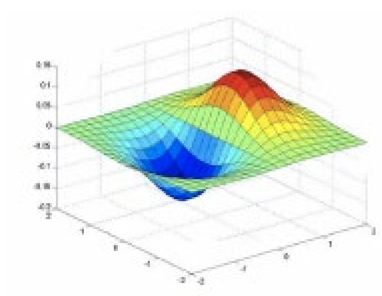
$$\frac{d}{dx}(f * g) = f * \frac{d}{dx}g$$

Derivative of gaussian

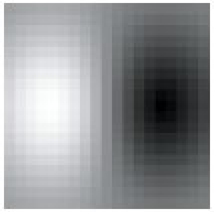


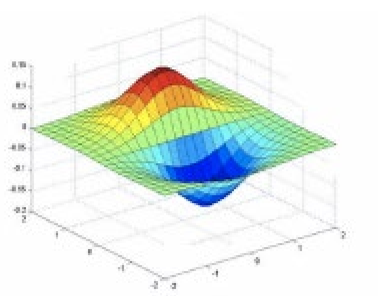
* [1 -1] =



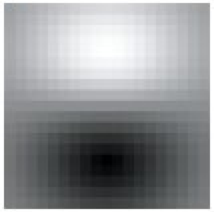


x-direction

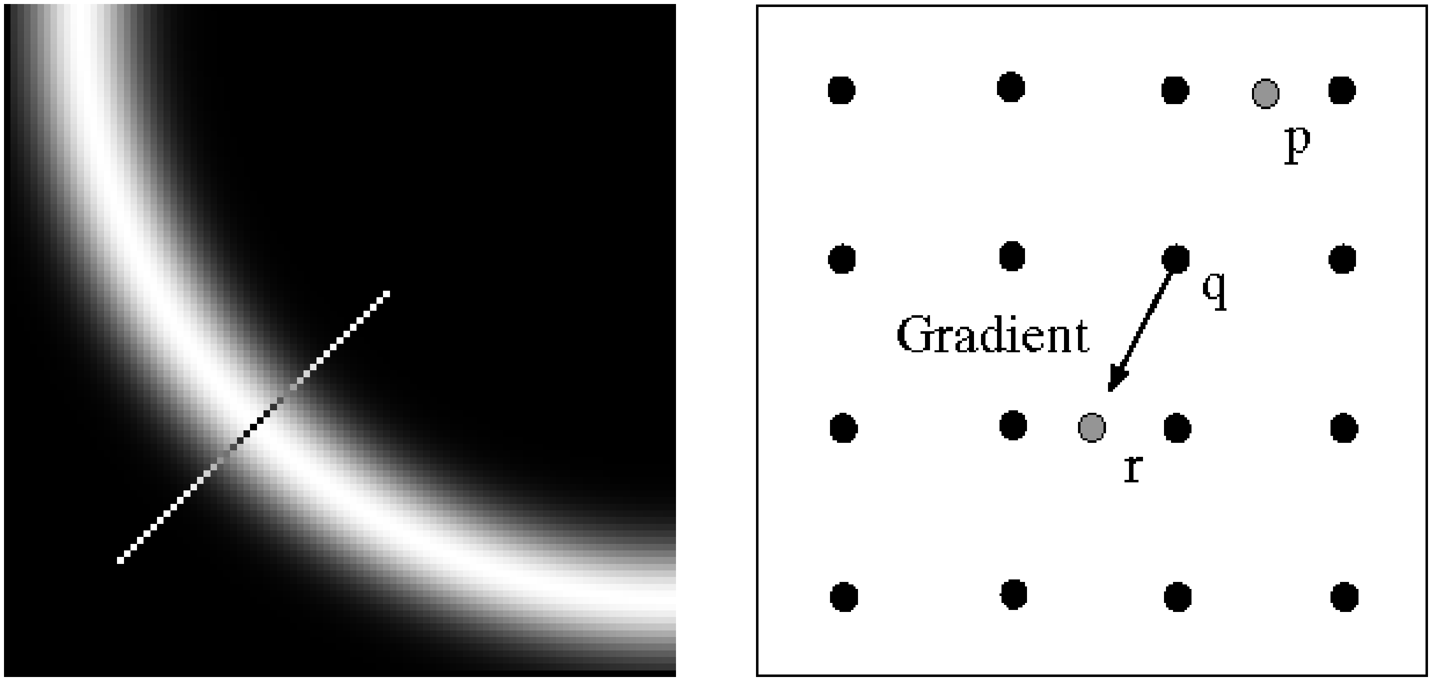




y-direction



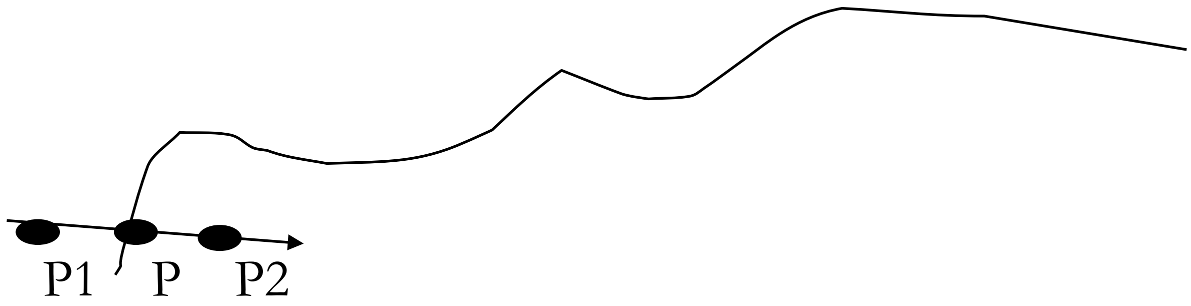
Non-maxima Suppression



- B. To find the true edge we verify if pixel is local maximum along gradient direction
- Thus we do not select with a high gradient of with a local maximum of DOG but only the true maxima which are the one in the direction of edges
- We need subpixel interpolation

Non-maxima Suppression

NON MAXIMA SUPPRESSION:



if $\begin{cases} Grad(P) \geq Grad(P_1) \\ Grad(P) \geq Grad(P_2) \end{cases} \Rightarrow$ P is a valid edge

The hp. is that all the points in the neighborhood of P have the same gradient direction

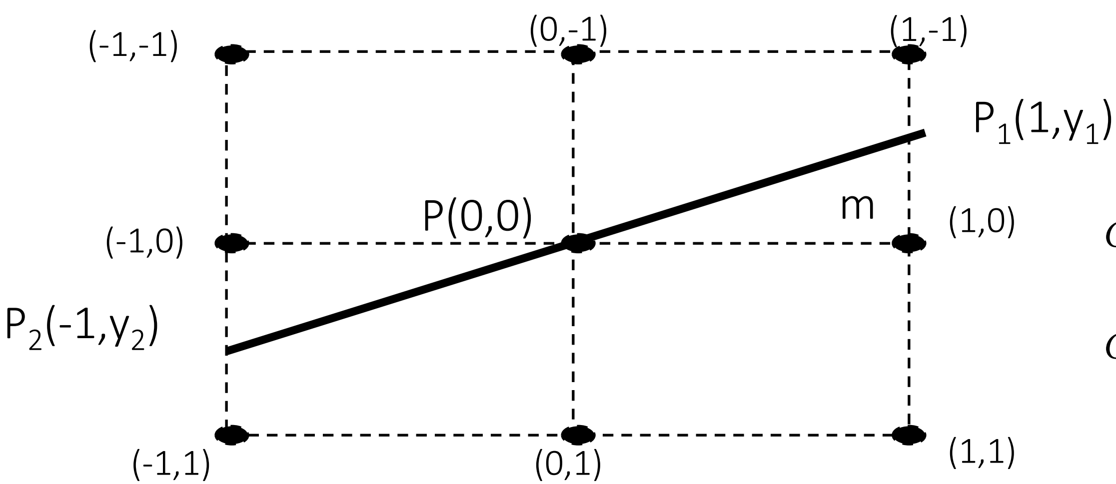
To find the direction we need of a **suitable interpolation** of neighbor gradient

Non-maxima Suppression

NON MAXIMA SUPPRESSION

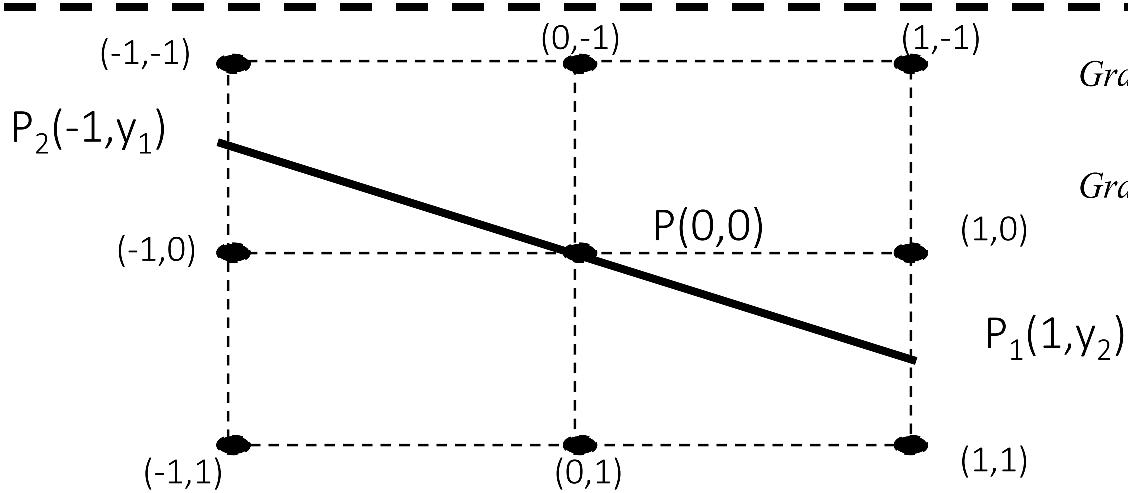
if $Grad_x(i,j) > Grad_y(i,j)$

$$m = \frac{Grad_y(i,j)}{Grad_x(i,j)} \qquad y_1 = -m \qquad y_2 = m$$



$$Grad(P_1) = Grad(1,-1)\cdot m + Grad(1,0)\cdot(1-m)$$

$$Grad(P_2) = Grad(-1,1)\cdot m + Grad(-1,0)\cdot(1-m)$$



$$Grad(P_1) = Grad(1,1)\cdot m + Grad(1,0)\cdot(1-m)$$

$$Grad(P_2) = Grad(-1,-1)\cdot m + Grad(-1,0)\cdot(1-m)$$

Non-maxima Suppression

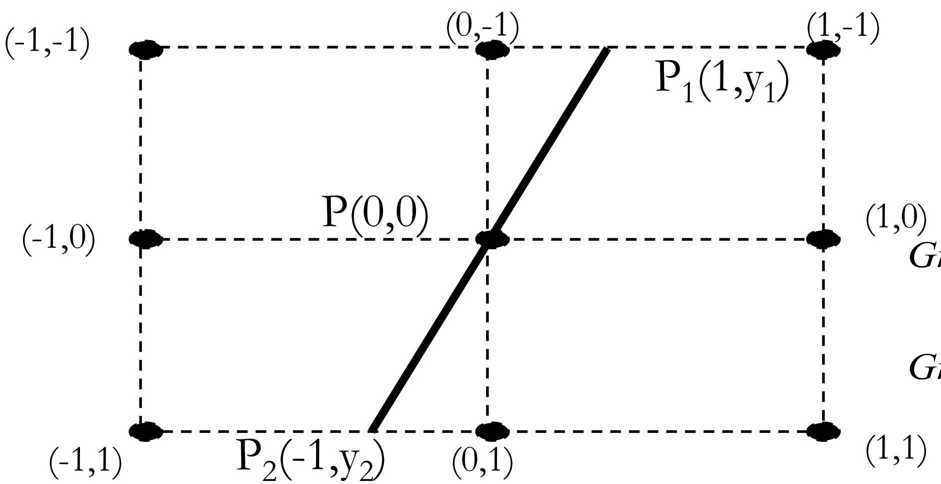
NON MAXIMA SUPPRESSION

if $Grad_y(i,j) > Grad_x(i,j)$

$$m = \frac{Grad_x(i,j)}{Grad_y(i,j)}$$

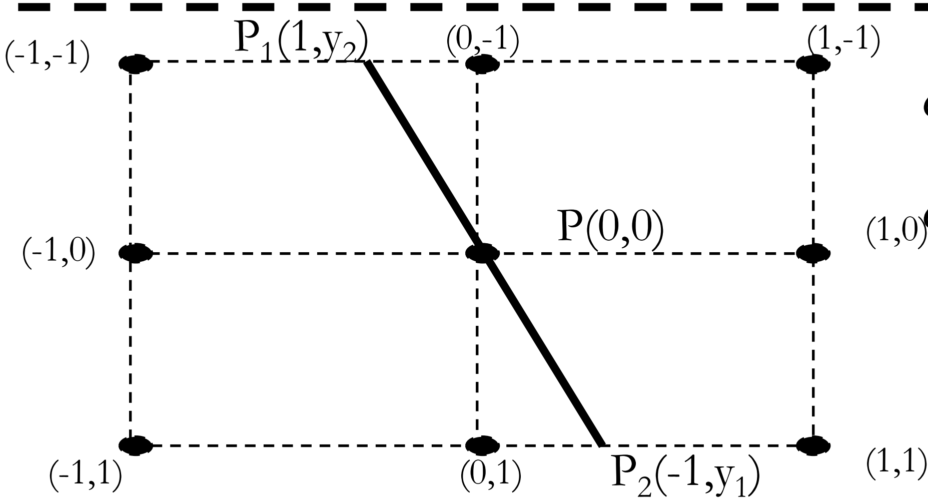
$y_1 = m$

$y_2 = -m$



$$Grad(P_1) = Grad(1,-1)\cdot m + Grad(0,-1)\cdot(1-m)$$

$$Grad(P_2) = Grad(-1,1)\cdot m + Grad(0,1)\cdot(1-m)$$



$$Grad(P_1) = Grad(-1,-1)\cdot m + Grad(0,-1)\cdot(1-m)$$

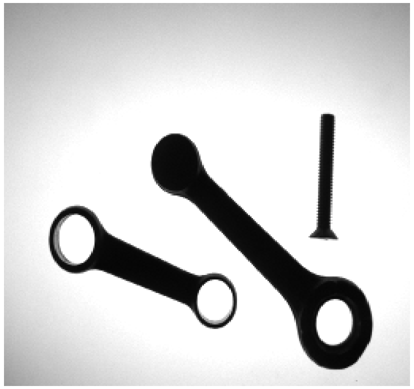
$$Grad(P_2) = Grad(1,1)\cdot m + Grad(0,1)\cdot(1-m)$$

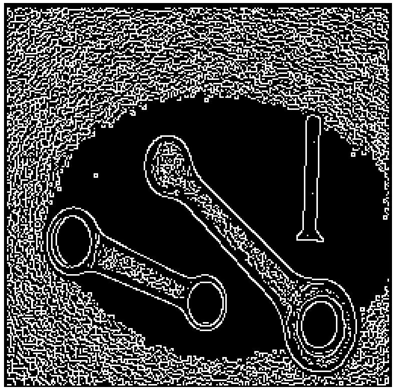
Thresholding

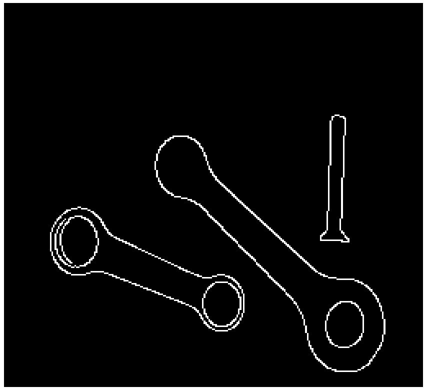
- After non maxima suppression the selected edges have the propriety to **create closed curves**; nevertheless only the strong edges should be selected Canny proposed the use of an Hysteresis -based threshloding

Canny

canny +hysteresis

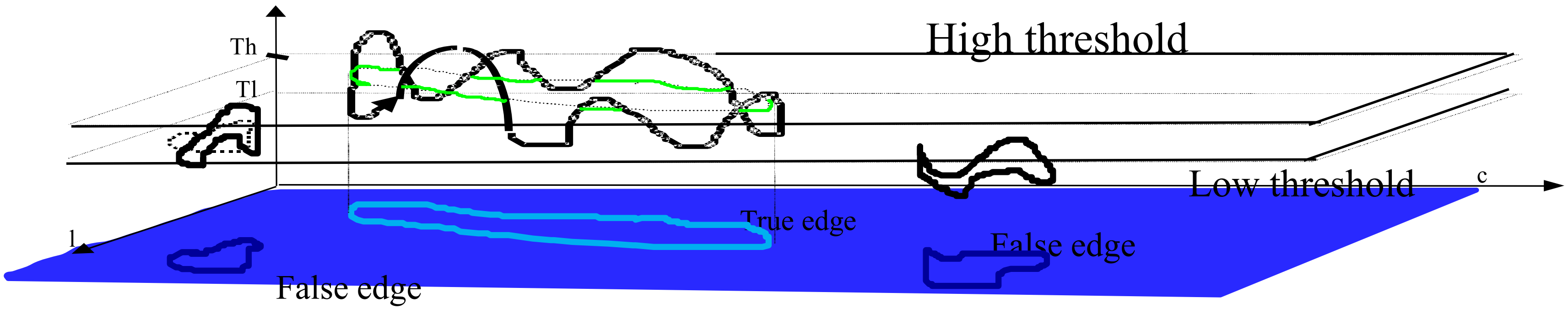






Threshold with hysteresis

- **Hysteresis threshold:**
- 1.Select a very strong **Thh**, (strong edges)
- 2. Edges are the weak edges (highest than a low threshold **Thl**) but connected with strong edges



- Normally:

$$T_H \cong 2 \div 3 T_L$$

$$E^{T_H T_L}(i,j) = \begin{cases} 1 & \text{if } G(i,j) > T_L \ \wedge \ \exists\,(k,l) \in N(i,j) \mid G(k,l) > T_H \\ \text{otherwise} & \end{cases}$$

- Use Thh to start and low threshold to continue the curve

Canny edge detector

original image (Lena)

Canny edge detector

magnitude of the gradient

Canny edge detector

Non maxima suppression

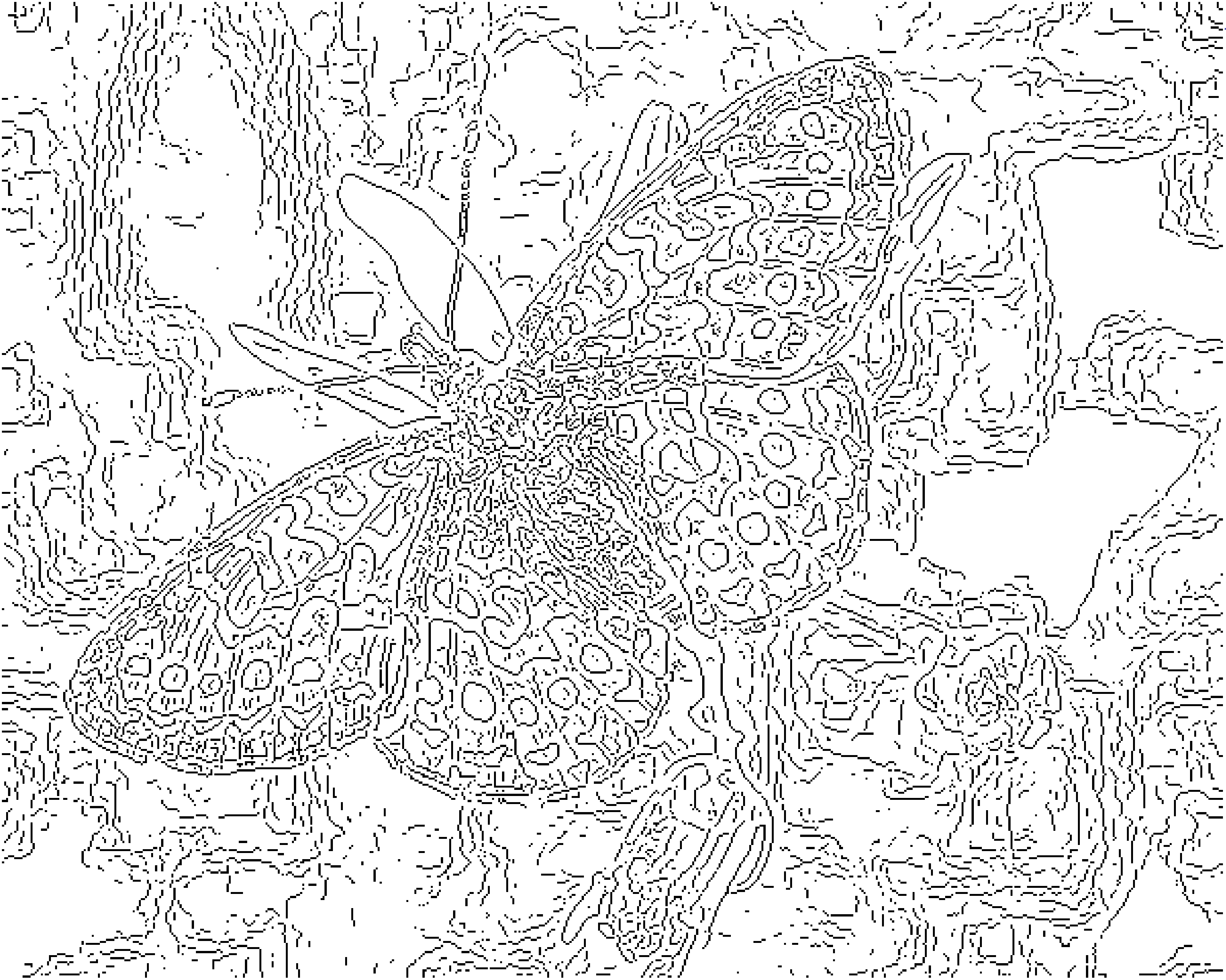
Canny Edge Operator

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

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



fine scale
high
threshold

coarse
scale,
high
threshold

coarse
scale
low
threshold