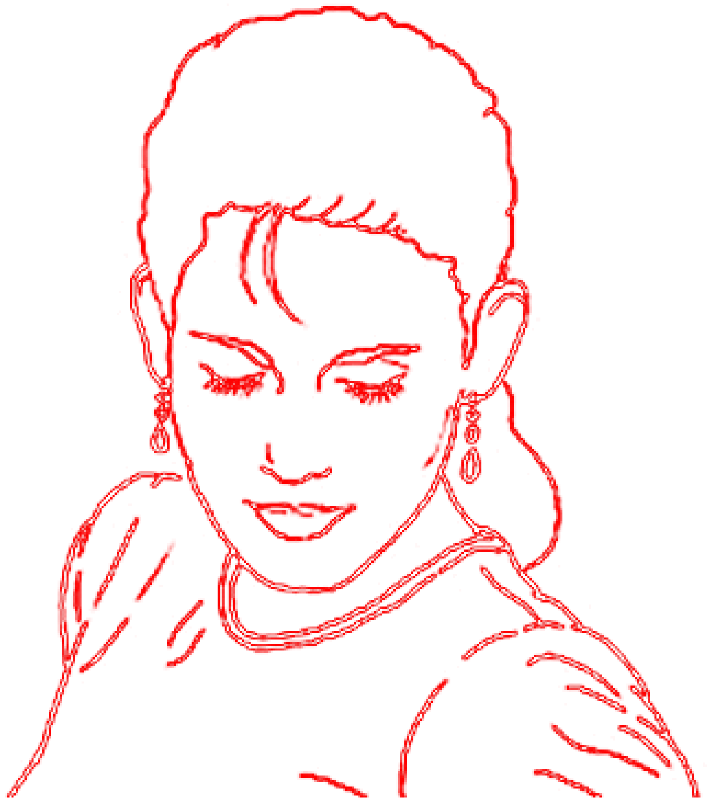
Shape extraction: contour

Edge detection

Edge detection

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



Source: D. Lowe

Segmentation

- Image segmentation consists into the decomposition of the image in segments (i.e. components)
- This process is based on a given criteria of homogeneity (chromatic, morphologic, motion, depth, etc.)
- From the operational viewpoint, three approach have been proposed:
  - Clustering image data and growing regions
  - Border following
  - Search of borders

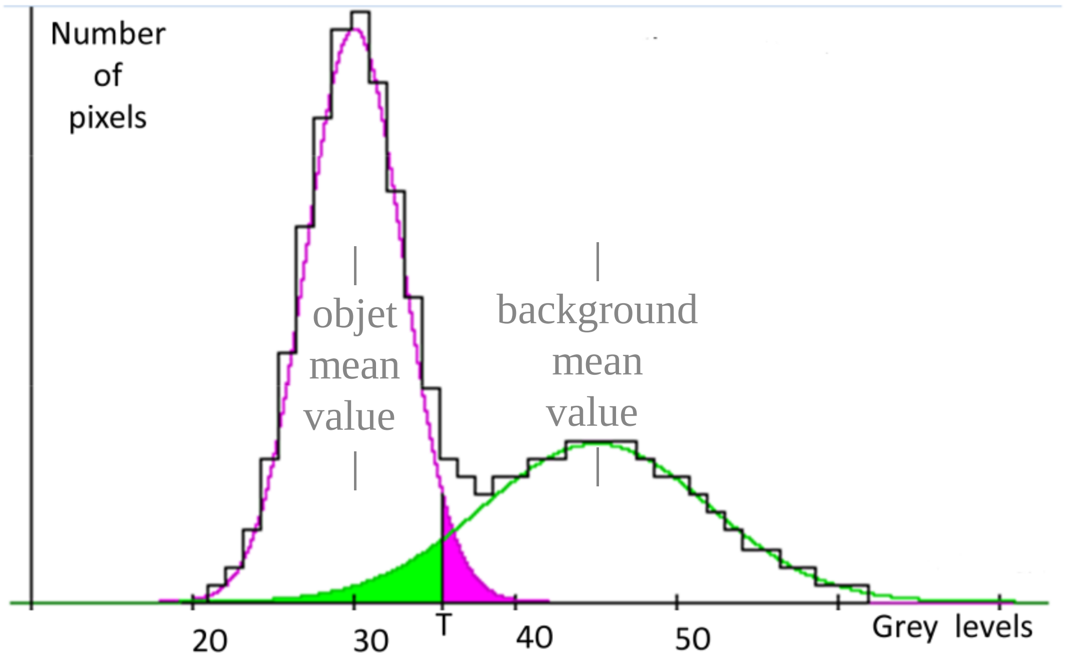
Binary Images

- The segmentation process leads to detect an individual object – foreground - in contrast to the background so it is a binarization process
- Some applications are by nature binary: black and white printing, writing, mechanical parts, bio-imagery like cells or chromosomes, etc. ....
- Often the originals contains various grey levels due to:
  - Non-uniform scene illuminations
  - Shadowing
  - Electric noise of the camera
  - …

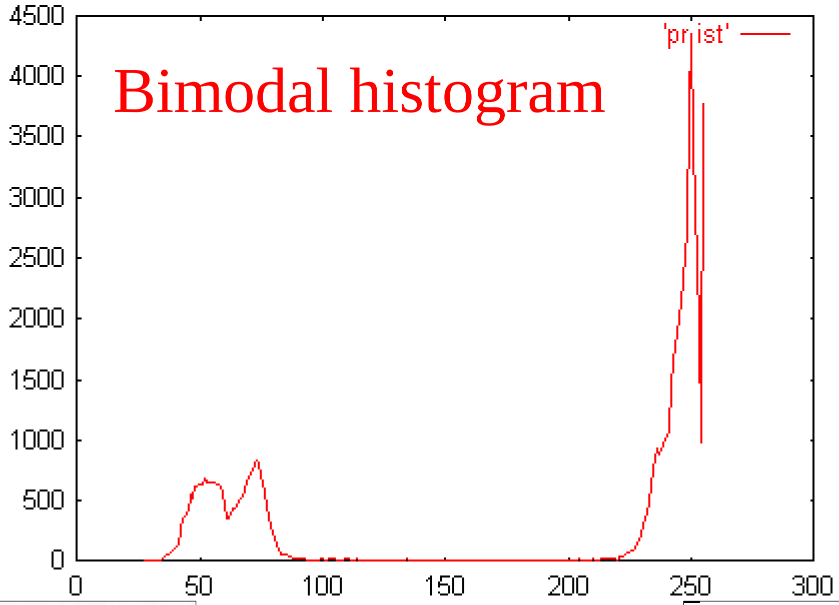
Bimodal Distribution

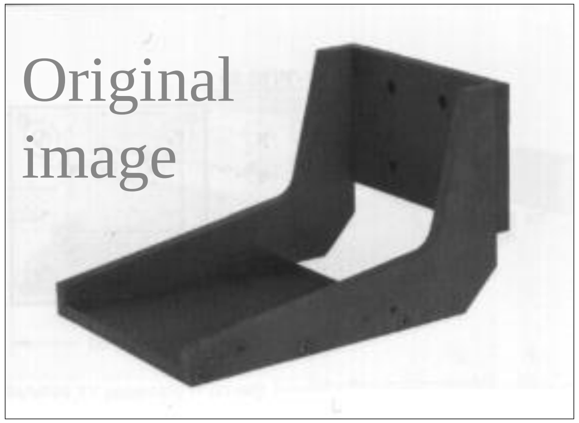
- The easiest solution is a threshold applied to the grey levels:
  - O(i, j) = 255 if I(i, j) < Th
  - O(i, j) = 0 otherwise
- It is required the evaluation of the optimal threshold Th.
- Operating on the histogram, there are two possibilities:
  - Finding the minimum
  - Applying statistic criteria

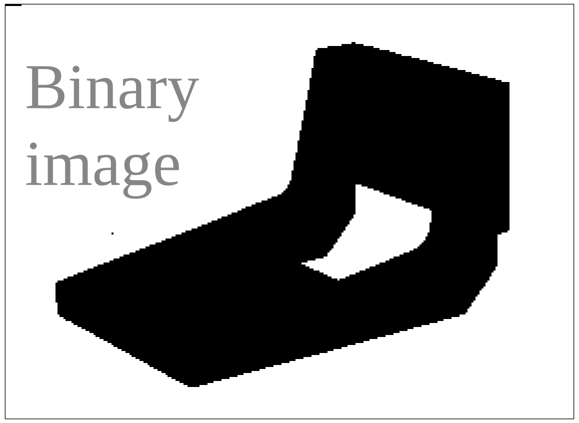
Bimodal histogram



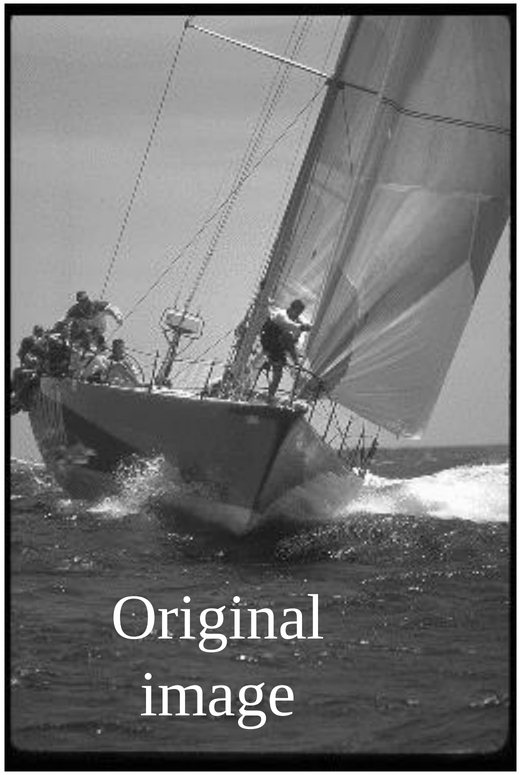
Example: mechanical part

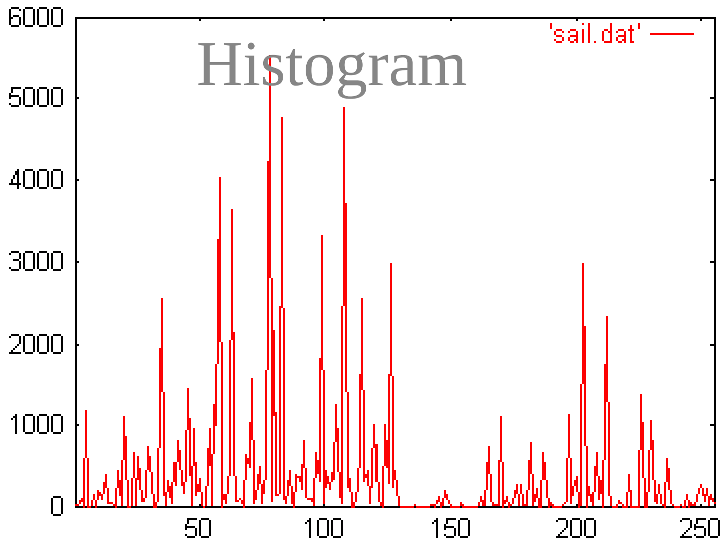




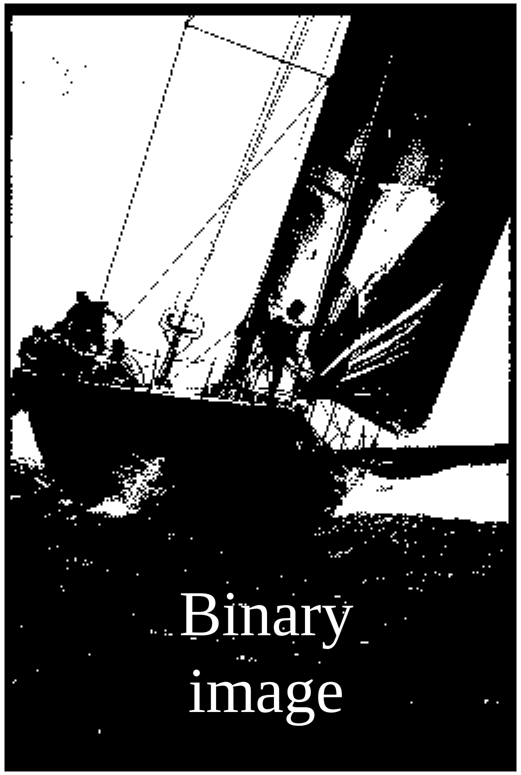


Example: sailing

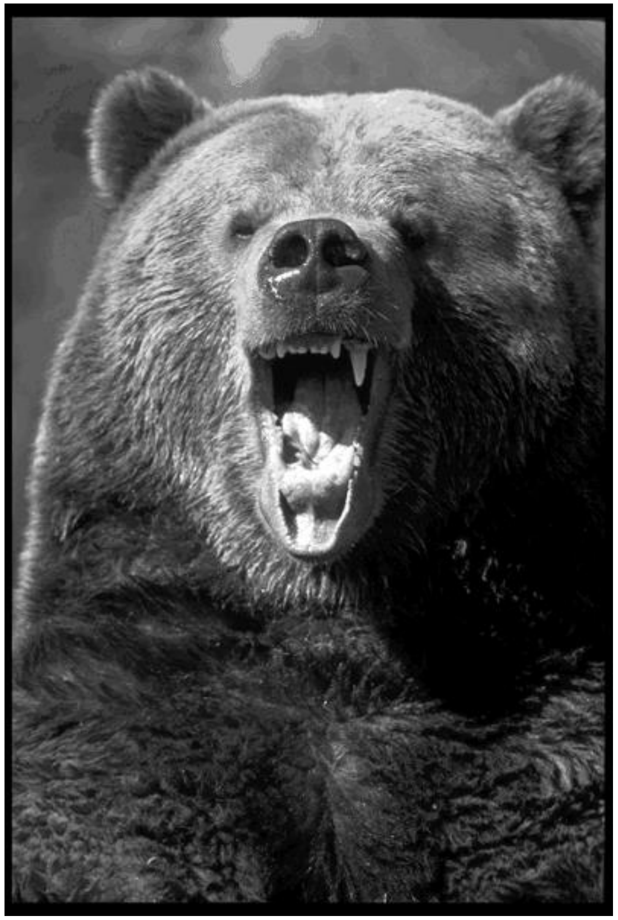


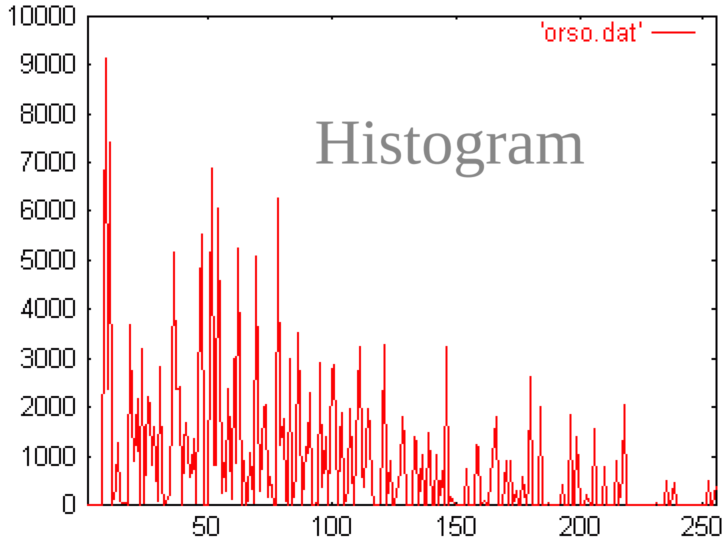


Threshold = 140

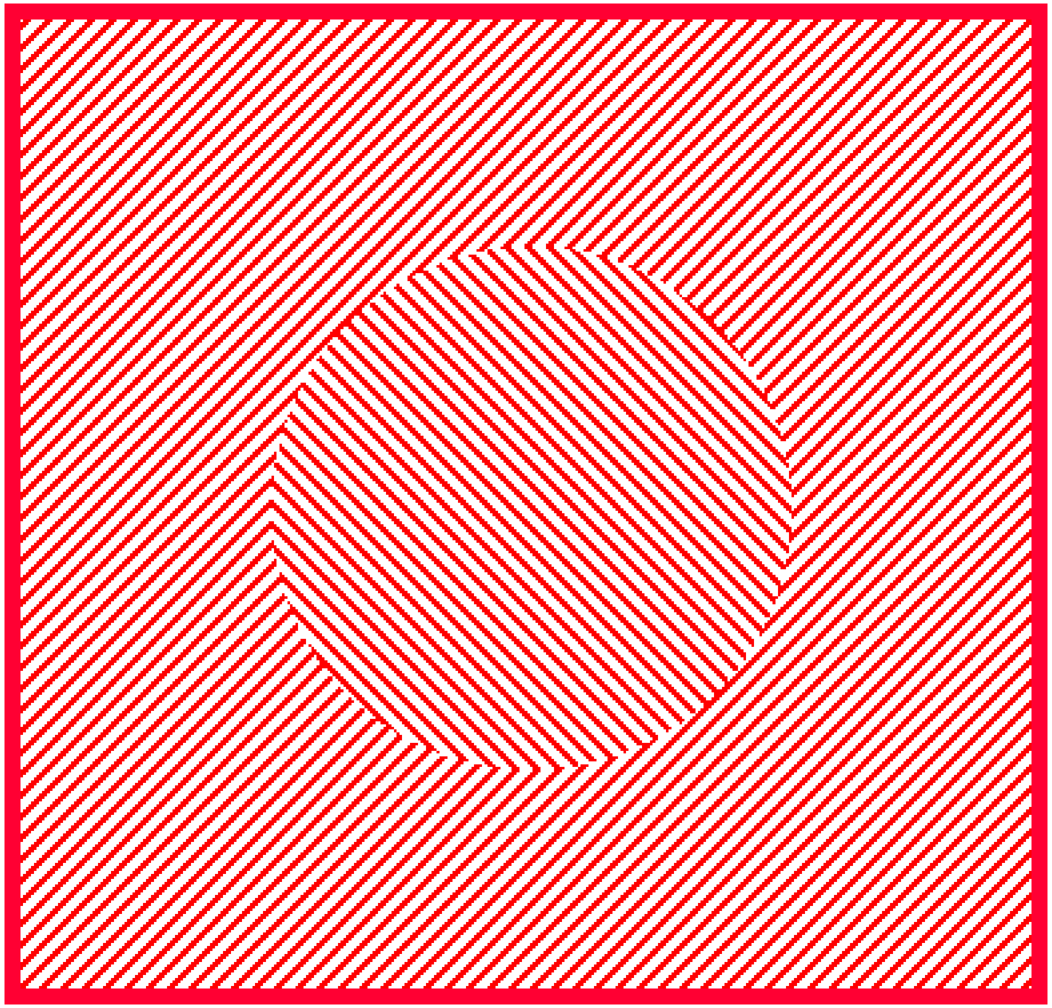


Example: bear


Original image

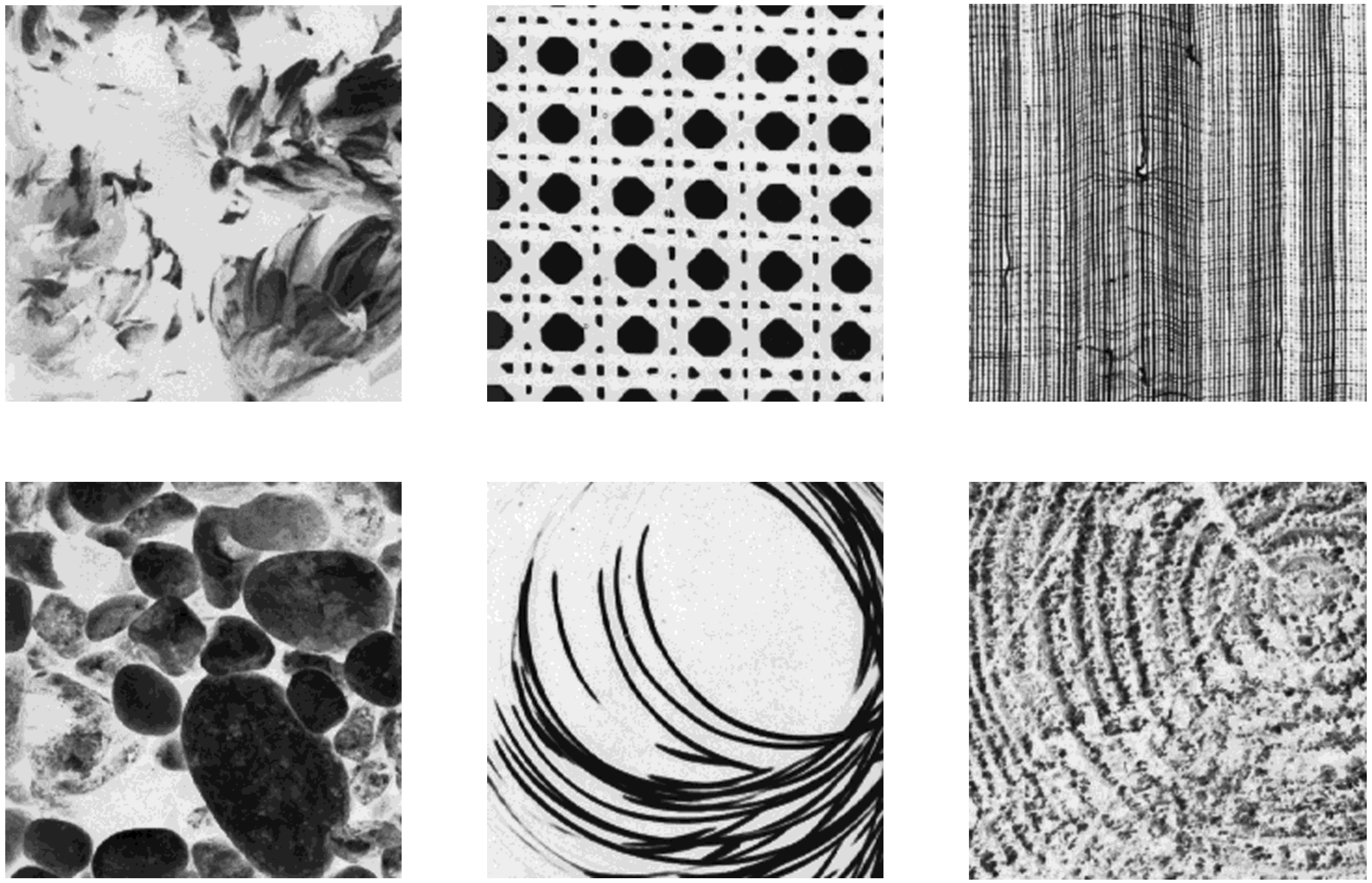


Example: circle



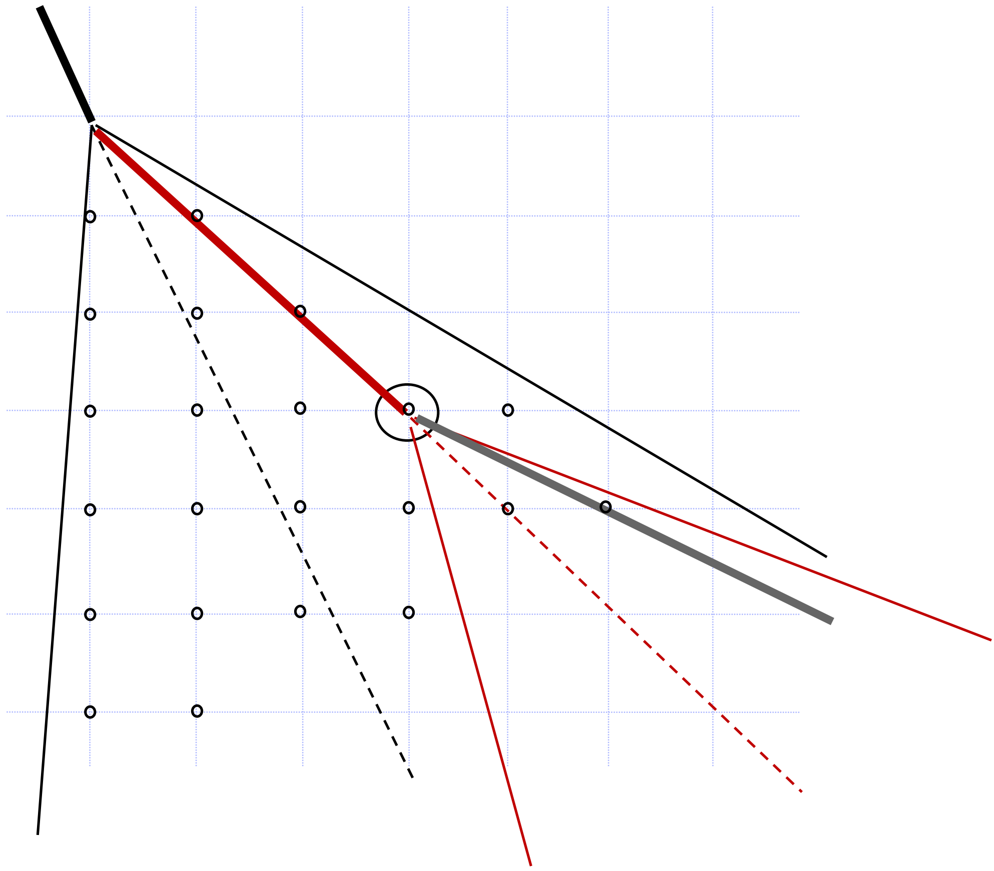
Co-occurence: higheer order ststistics

Texture: Brodatz album

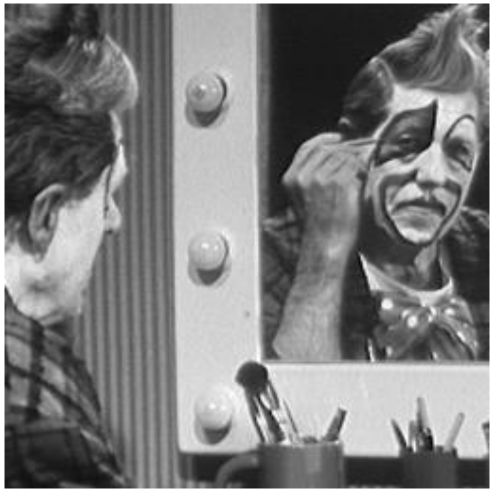


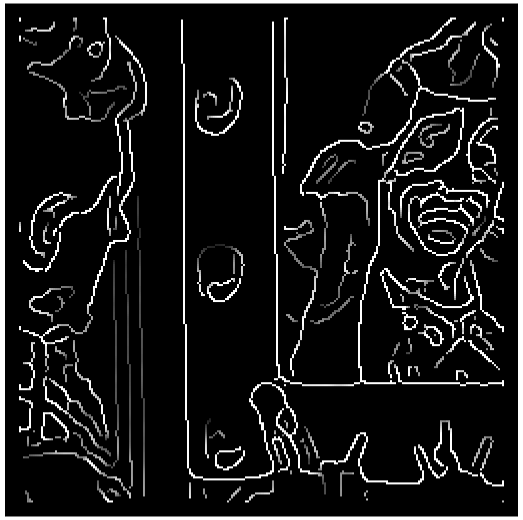
Border following

- An example of a recursive walk over the image, following the contour to be exhibited. In the example the horizon of an edge point is the triangle of depth 5 and basis 6, in the direction of the last found edge segment.



Search of borders



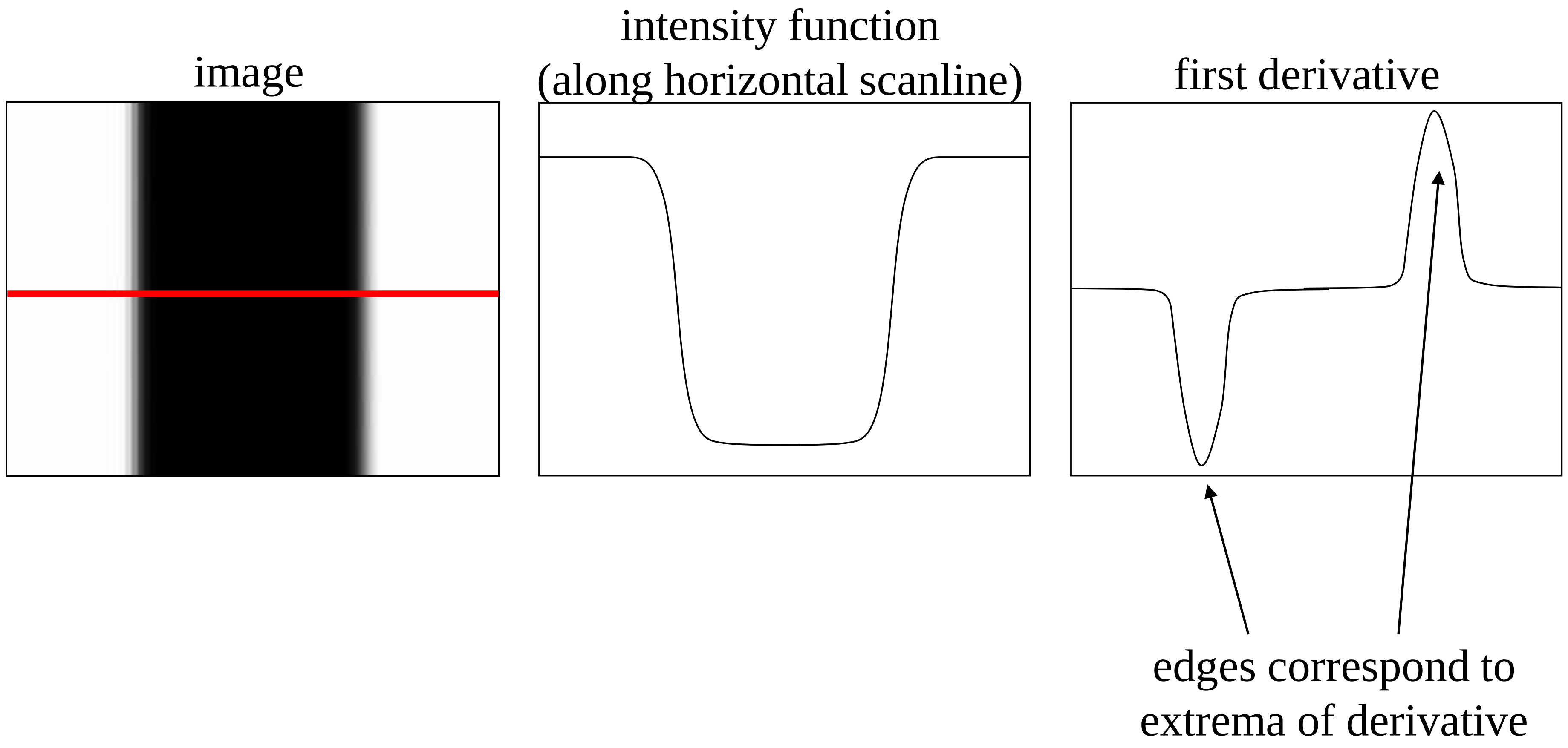


Analytic derivative model

- The border search can be based on the discontinuity of an image feature like the grey level, a texture or a motion parameter, the depth in the scene, etc.
- For operators stemming from first order partial derivatives a maximum response is looked for, either local maximum or over a threshold whether given or adapted
- Note that the second derivative is used too, and among second order operators the Laplacian is peculiarly popular as being scalar then isotropic. There, of course, the zero crossing – inflection points - are looked for.

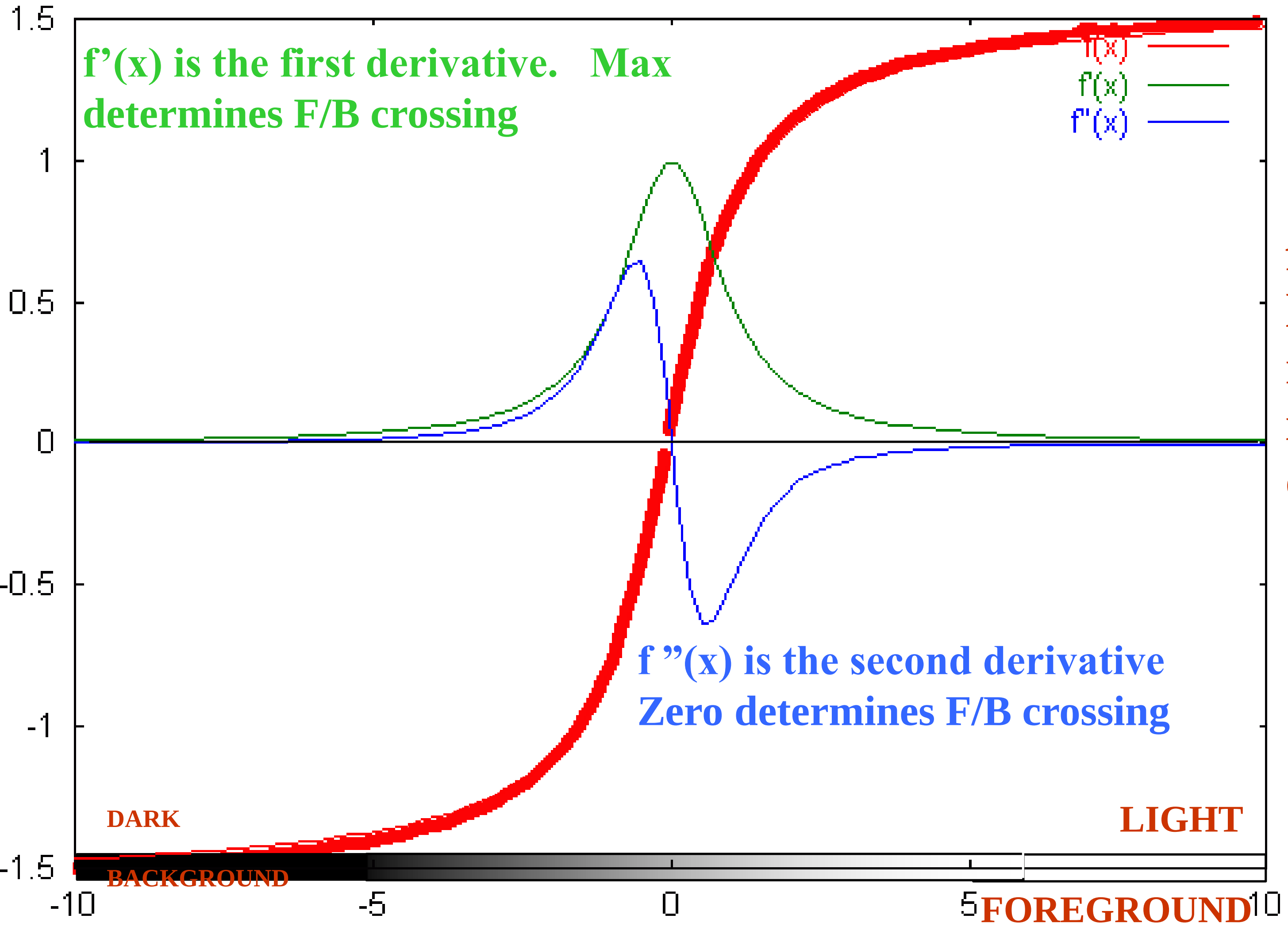
Derivatives and edges

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



Source: L. Lazebnik

Analytic derivative model



f(x) is the grey level, here representing the image in one dimension

Analytic derivative model

- The first derivative is given by:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

- The second derivative is given by:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

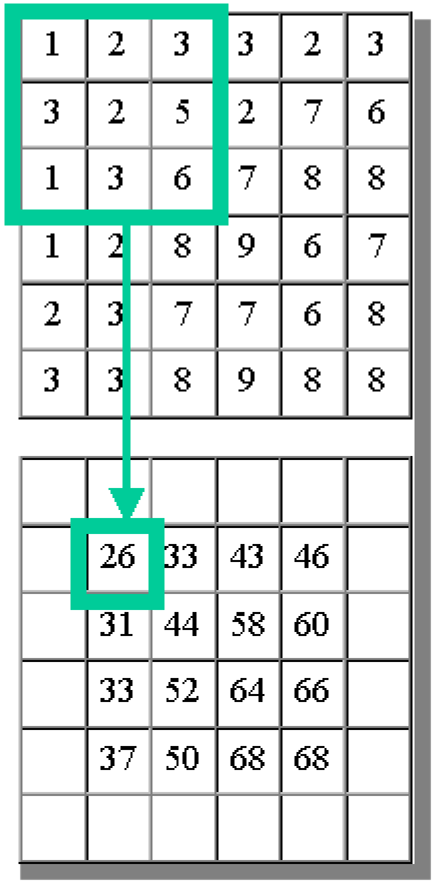
- In 2D the derivate is substituted by the vector gradient

Convolution

- The convolution is a linear operator, that is applied when the image I(x, y) is continue. To the digital image I(i, j) a filter is applied represented by the mask:

$$O(x_0,y_0) = \iint f(x_0-x,\ y_0-y)\ I(x,y)\ dx\ dy$$

$$O(x,y) = \sum\sum f(x-i,\ y-j)\ I(i,j)$$



1	2	3	3	2	3
3	2	5	2	7	6
1	3	6	7	8	8
1	2	8	9	6	7
2	3	7	7	6	8
3	3	8	9	8	8

	26	33	43	46	
	31	44	58	60	
	33	52	64	66	
	37	50	68	68	

Example: box filter

$$g[\cdot\,,\cdot\,]$$

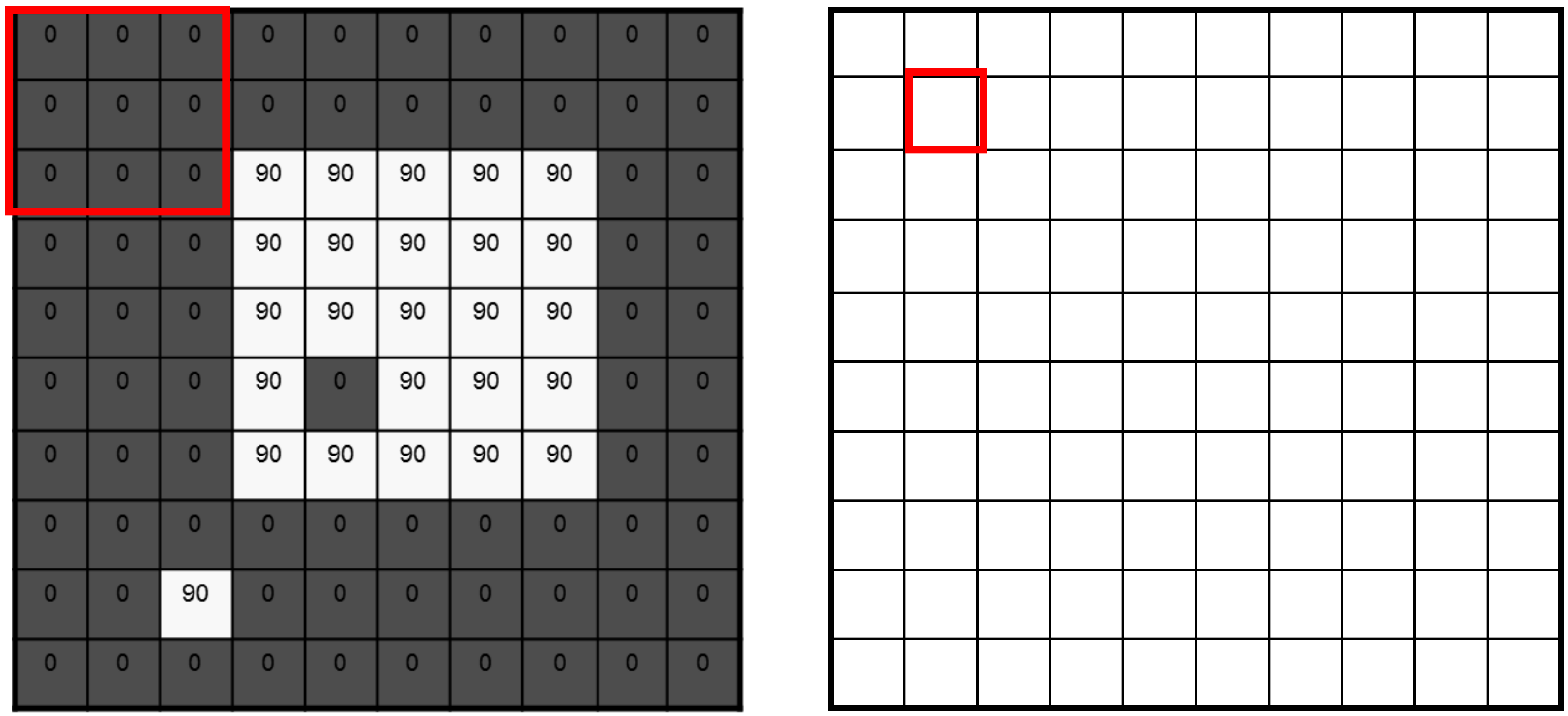
$$\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$$

Slide credit: David Lowe (UBC)

Image filtering

$$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$f[.,.]$ $\quad$ $h[.,.]$



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

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

Credit: S. Seitz

Image filtering

$g[\cdot,\cdot]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$

$f[.,.]$

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

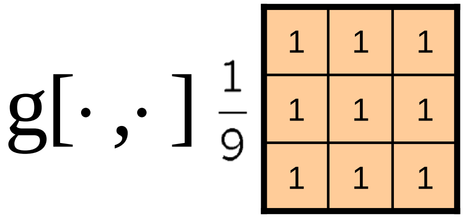
$h[.,.]$

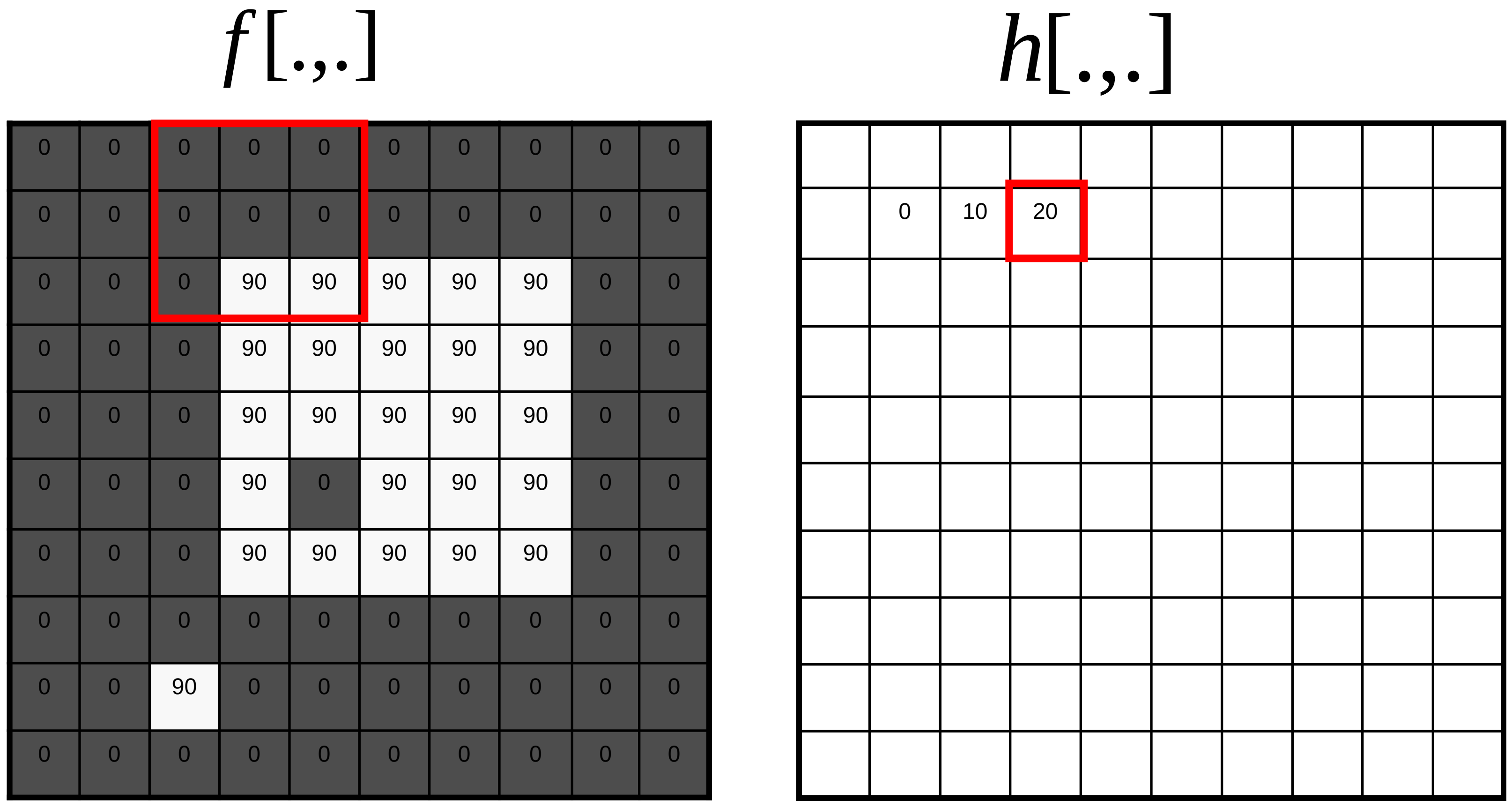
	0	10							

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

Credit: S. Seitz

Image filtering





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

	0	10	20						

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

Credit: S. Seitz

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30					

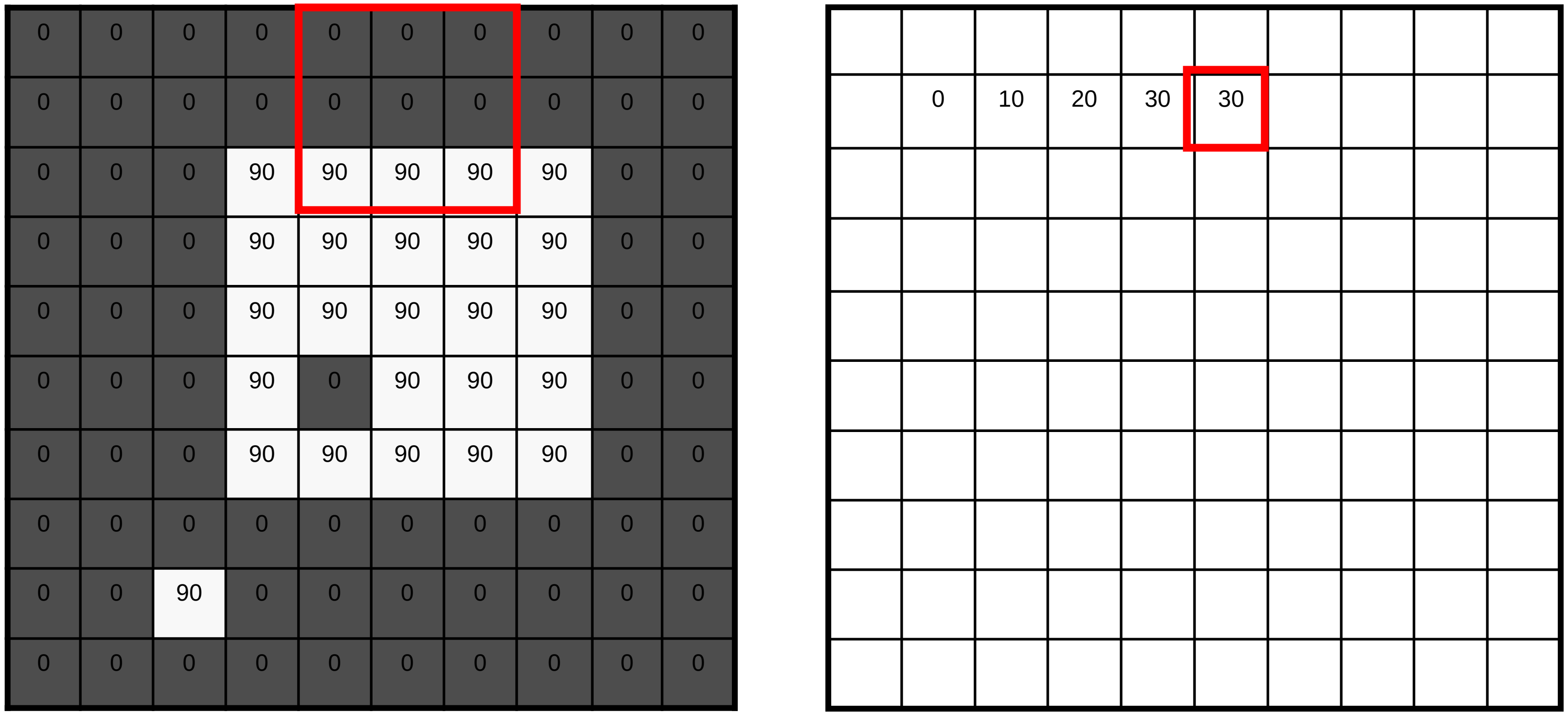
$$h[m,n] = \sum_{k,l} g[k,l]\ f[m+k,n+l]$$

Credit: S. Seitz

Image filtering

$$g[\cdot\,,\cdot\,]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$f[.,.]$ $h[.,.]$



$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				

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

Credit: S. Seitz

Image filtering

$$g[\cdot\,,\cdot\,]\ \frac{1}{9}\begin{bmatrix}1&1&1\\1&1&1\\1&1&1\end{bmatrix}$$

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				
				?					

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

Credit: S. Seitz

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				
						?			
				50					

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

Credit: S. Seitz

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	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	

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

Credit: S. Seitz

Box Filter

What does it do?

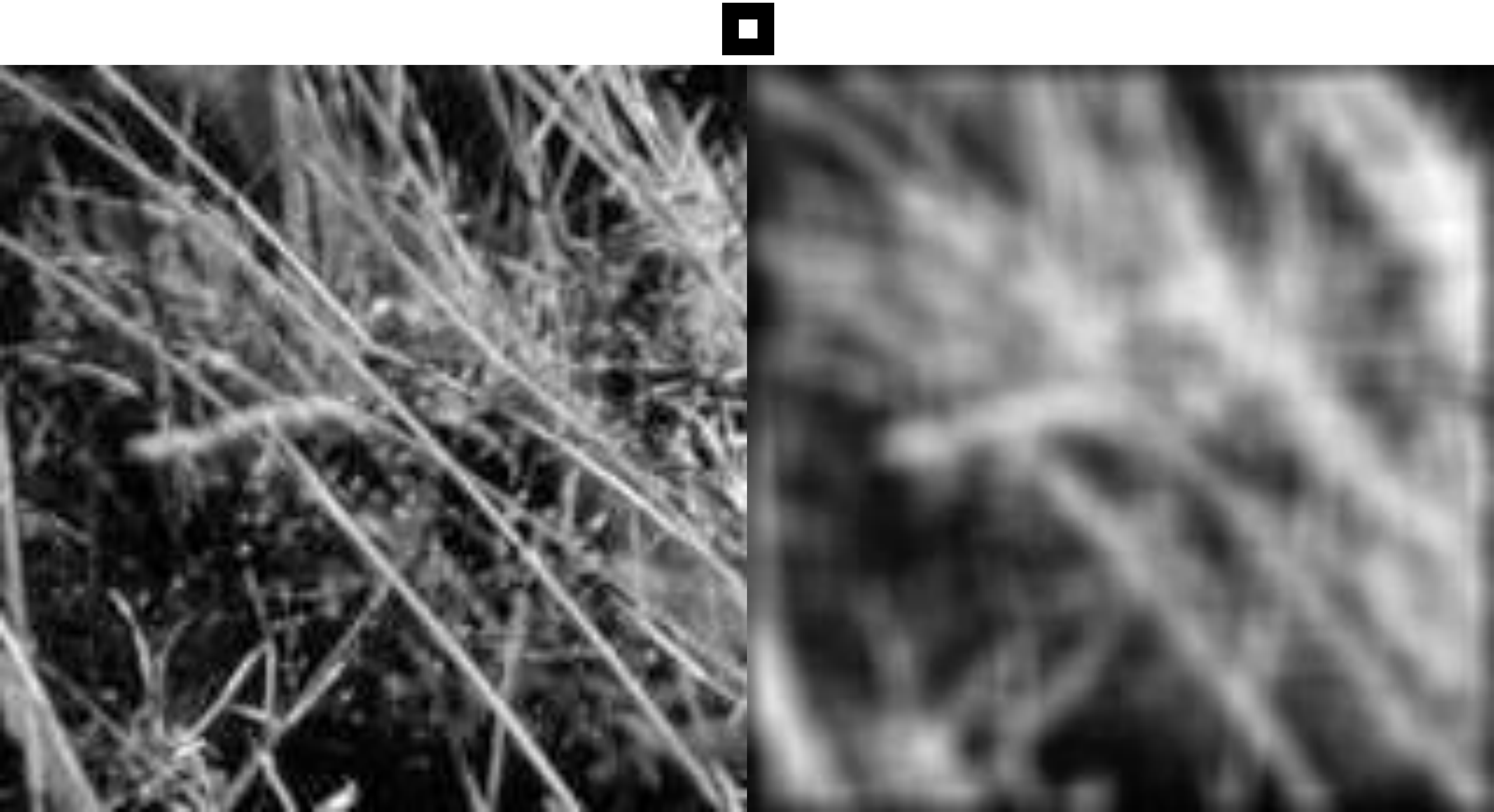
- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

$$g[\cdot\,,\cdot\,]$$

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

Slide credit: David Lowe (UBC)

Smoothing with box filter



Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

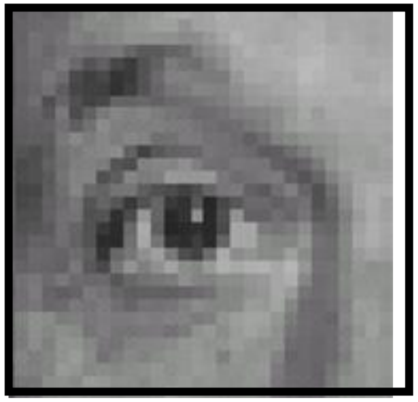
?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0


Shifted left
By 1 pixel

Source: D. Lowe

Practice with linear filters

Original

$$\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array} - \frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \qquad ?$$

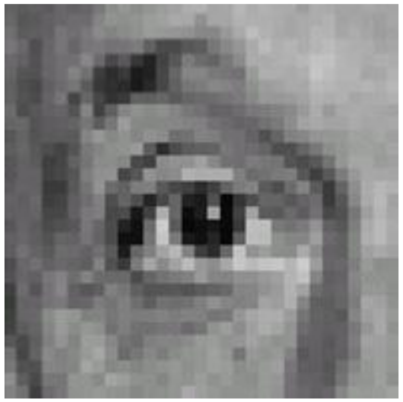
(Note that filter sums to 1)

Source: D. Lowe

Practice with linear filters

Original

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

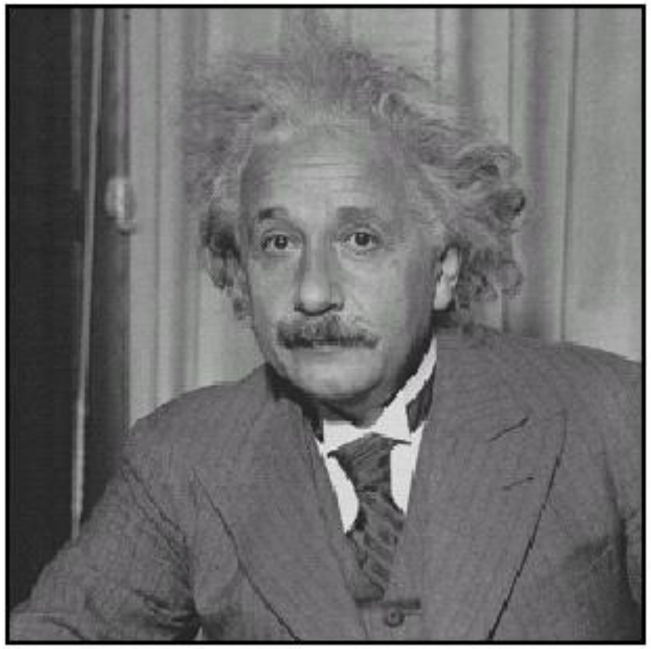


Sharpening filter

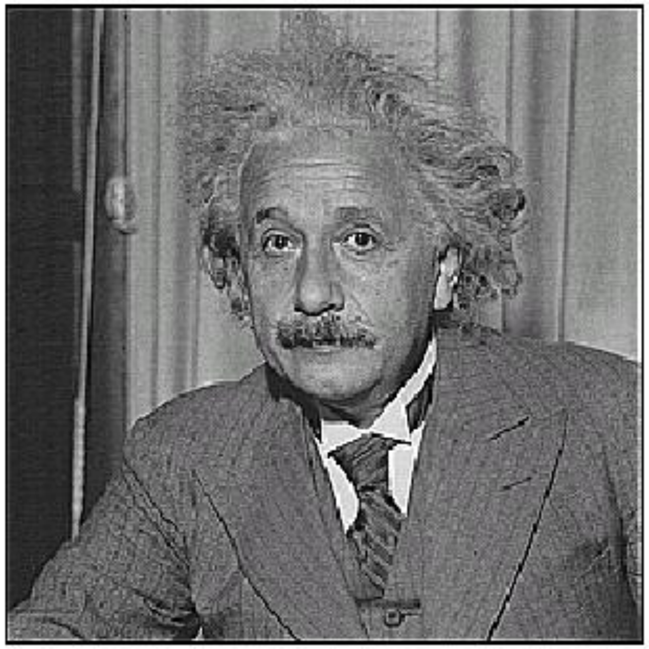
- Accentuates differences with local average

Source: D. Lowe

Sharpening



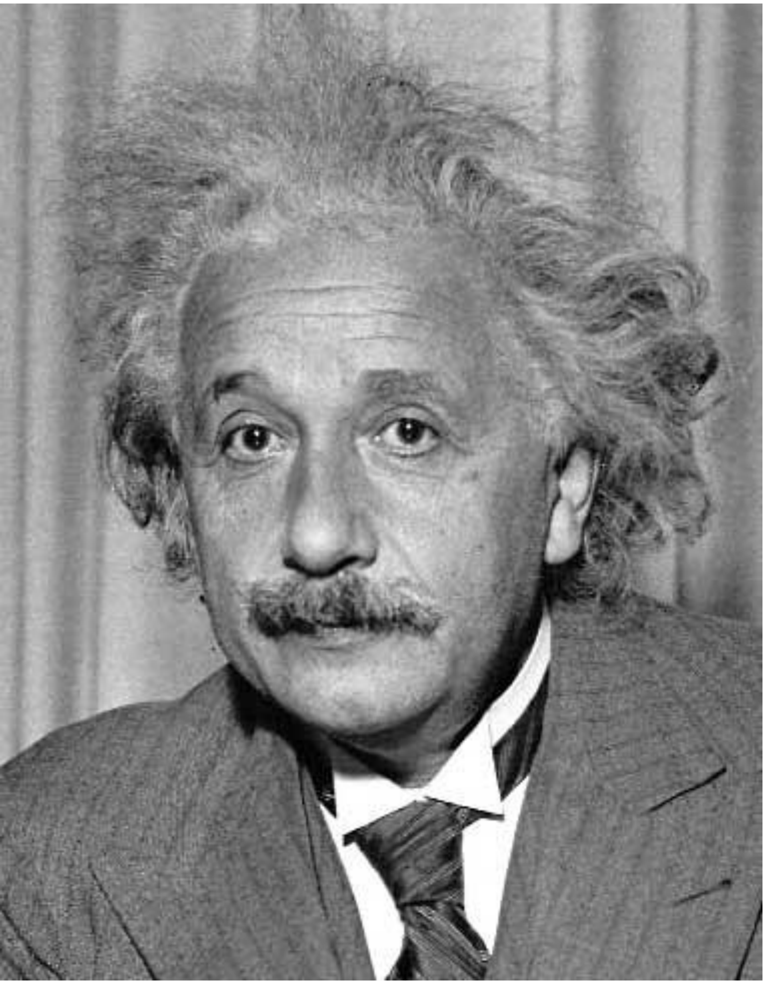
before



after

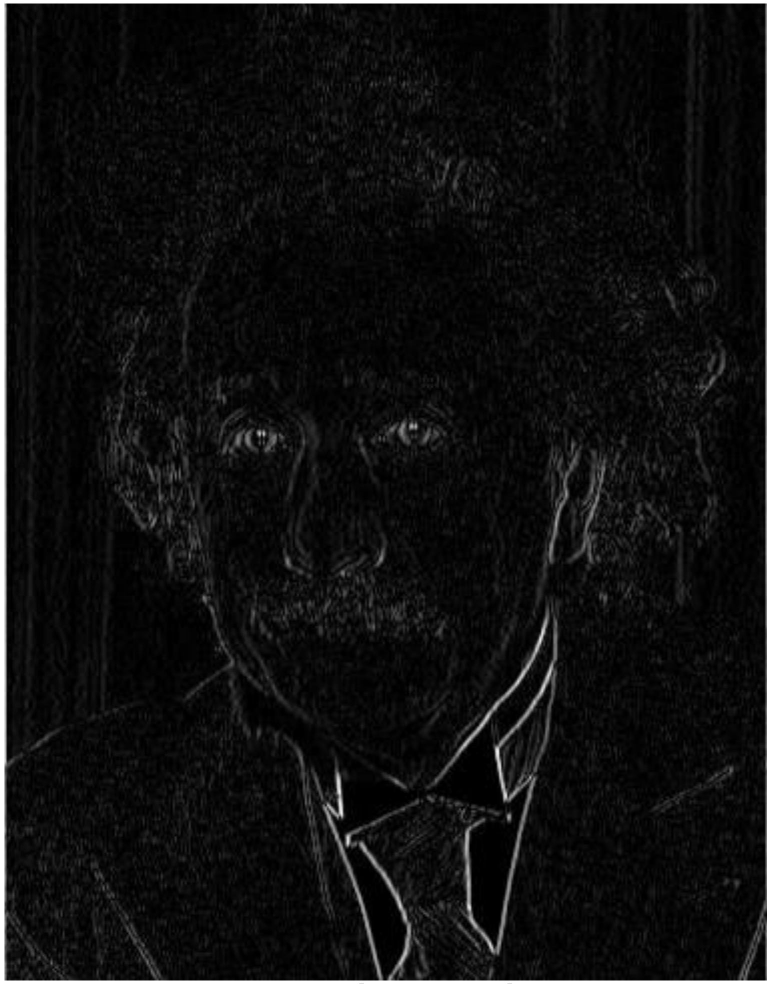
Source: D. Lowe

Other filters



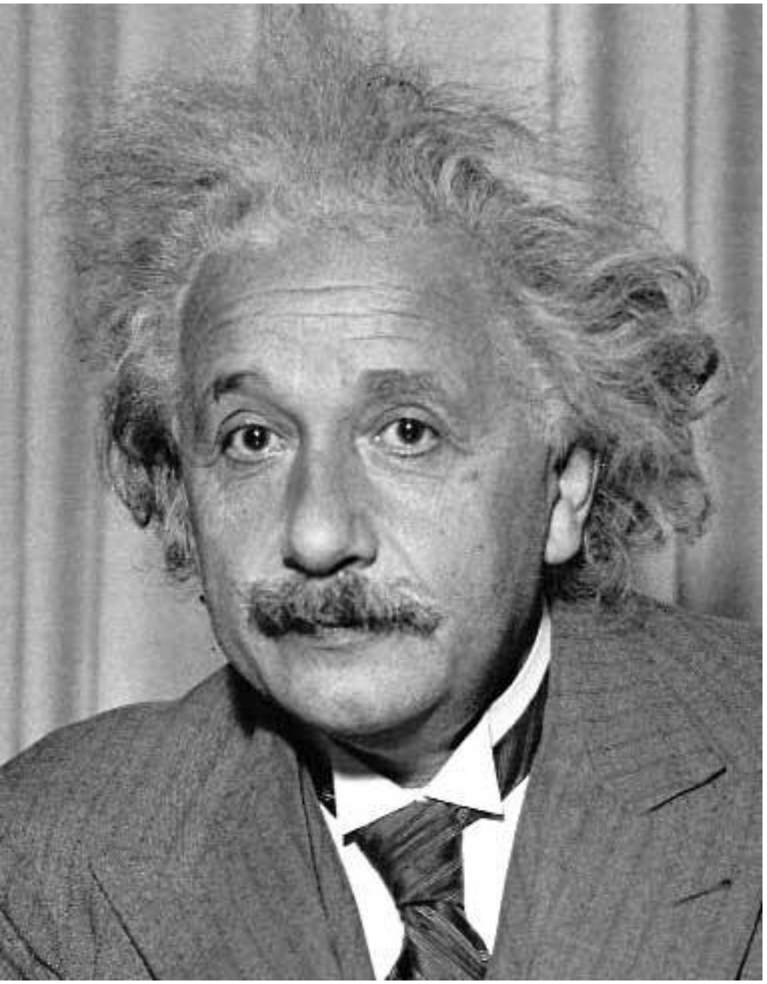
1	0	-1
2	0	-2
1	0	-1

Sobel



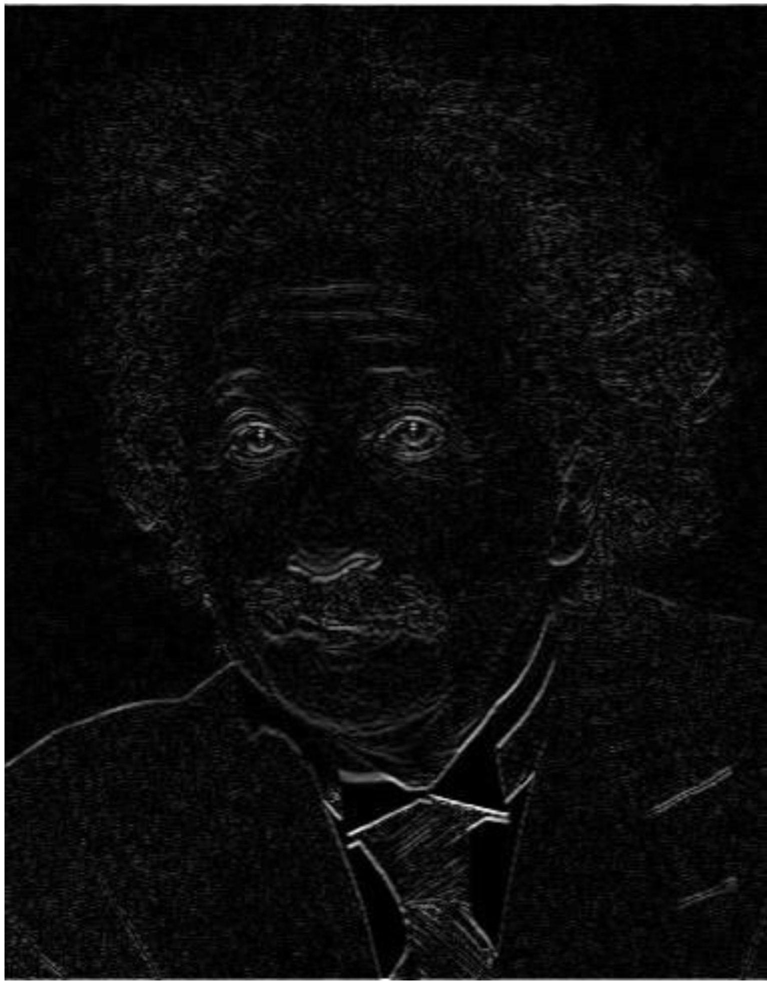
Vertical Edge
(absolute value)

Other filters



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

Sobel



Horizontal Edge
(absolute value)

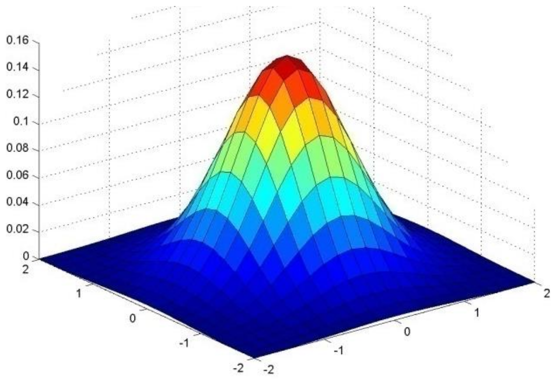
More properties

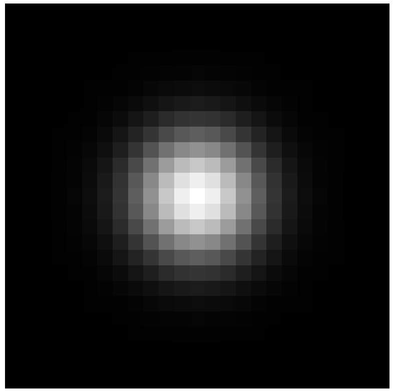
- Commutative: $a * b = b * a$
  - Conceptually no difference between filter and signal
  - But particular filtering implementations might break this equality
- Associative: $a * (b * c) = (a * b) * c$
  - Often apply several filters one after another: $(((a * b_1) * b_2) * b_3)$
  - This is equivalent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

Source: S. Lazebnik

Important filter: Gaussian

- Weight contributions of neighboring pixels by nearness



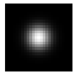


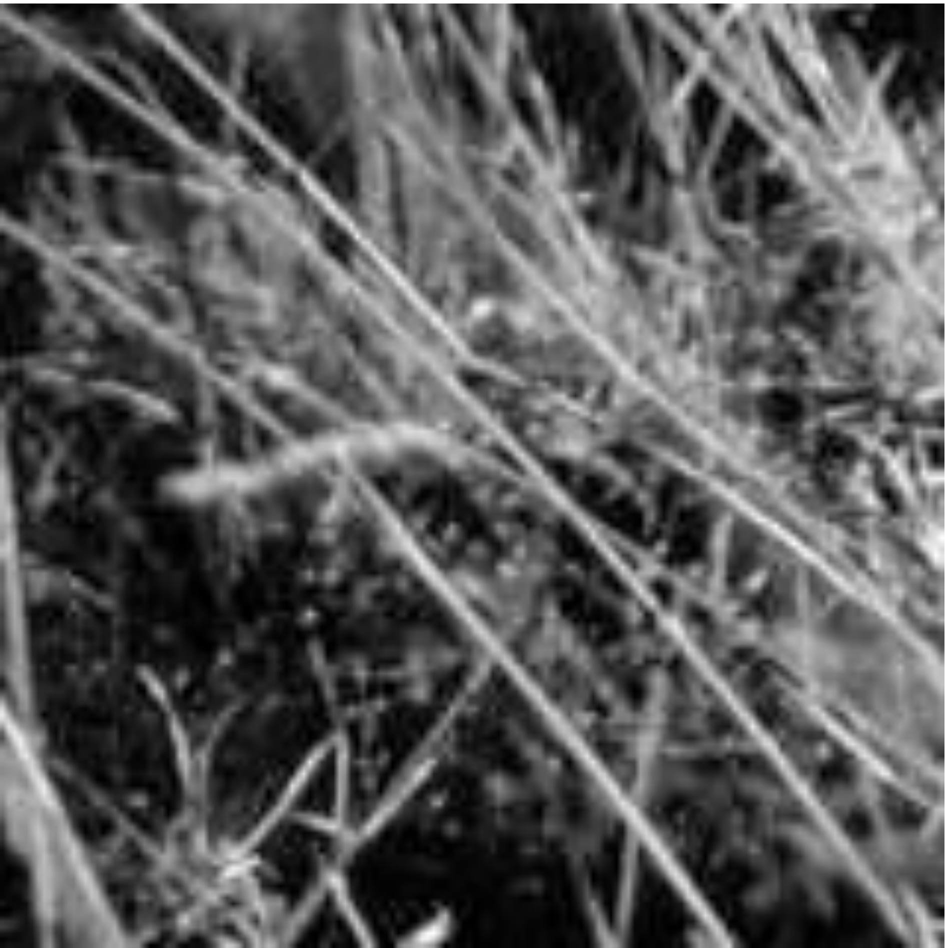
0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

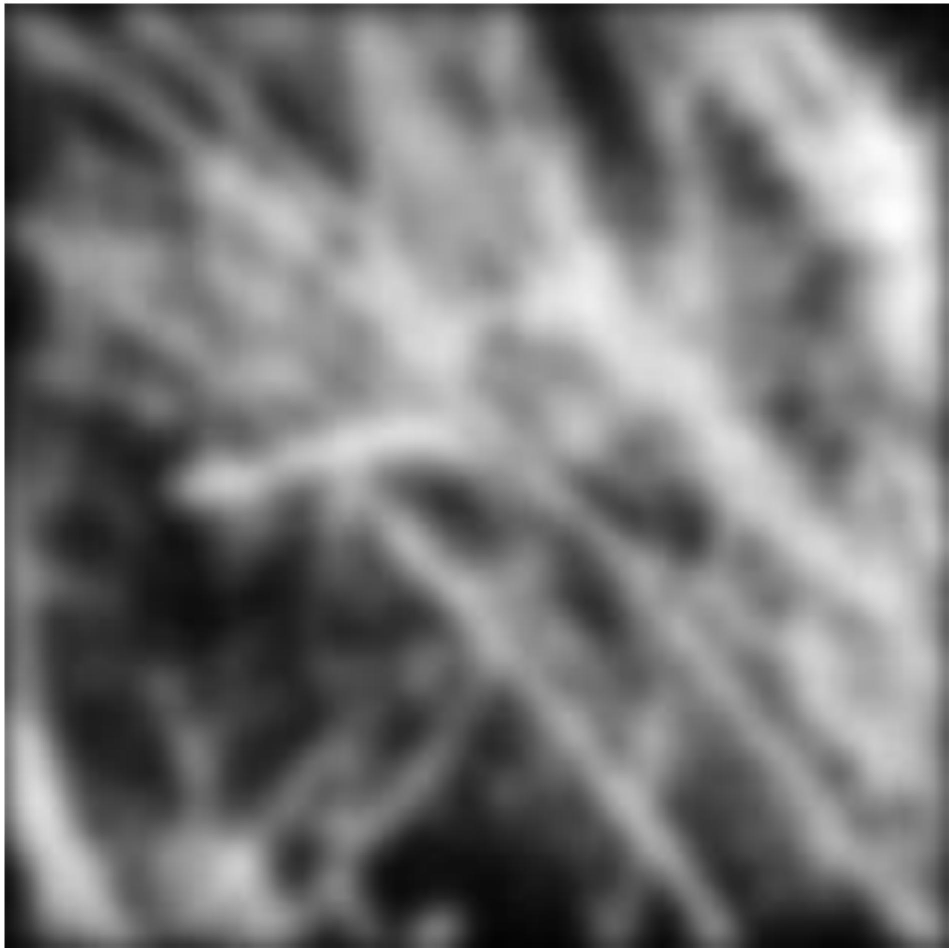
5 x 5, σ = 1

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

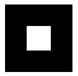
Smoothing with Gaussian filter

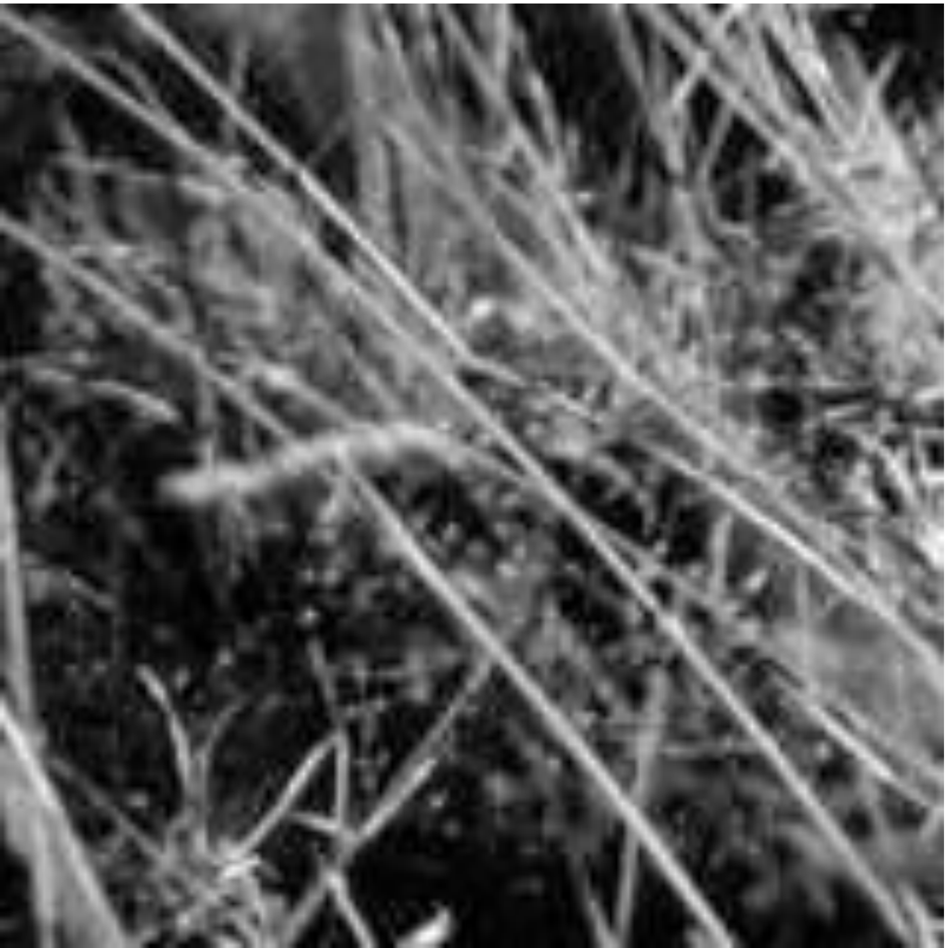


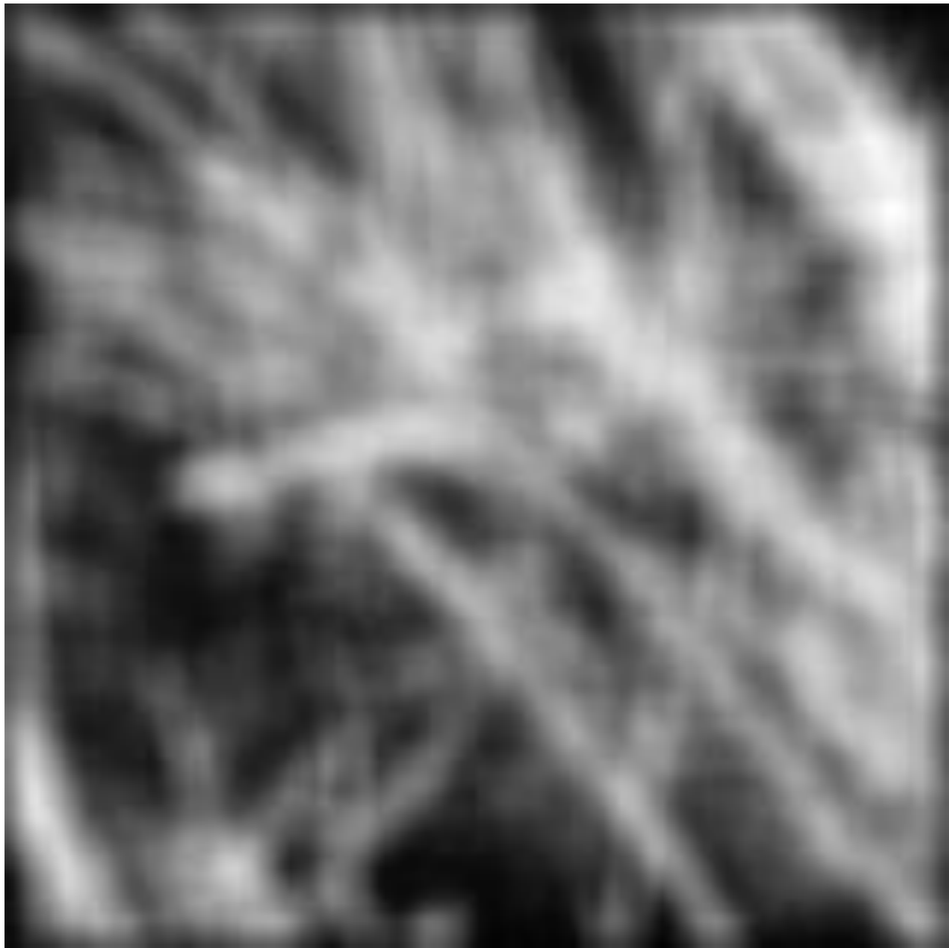




Smoothing with box filter







Gaussian filters

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

Source: K. Grauman

Separability of the Gaussian filter

- The 2d Gaussian can be expressed by a product of two functions, one function of x and the other function of y

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

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

Source: D. Lowe

Boundary issues

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

Source: S. Marschner

Filtering basics

Say the averaging window size is 2k+1 x 2k+1:

$$G[i,j] = \underbrace{\frac{1}{(2k+1)^2}}_{\text{Attribute uniform weight to each pixel}} \underbrace{\sum_{u=-k}^{k} \sum_{v=-k}^{k} F[i+u, j+v]}_{\text{Loop over all pixels in neighborhood around image pixel } F[i,j]}$$

Now generalize to allow different weights depending on neighboring pixel's relative position:

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} \underbrace{H[u,v]}_{\text{Non-uniform weights}} F[i+u, j+v]$$

Correlation filtering

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v] F[i+u, j+v]$$

This is called cross-correlation, denoted $G = H \otimes F$

Filtering an image: replace each pixel with a linear combination of its neighbors.

The filter "kernel" or "mask" $H[u,v]$ is the prescription for the weights in the linear combination.

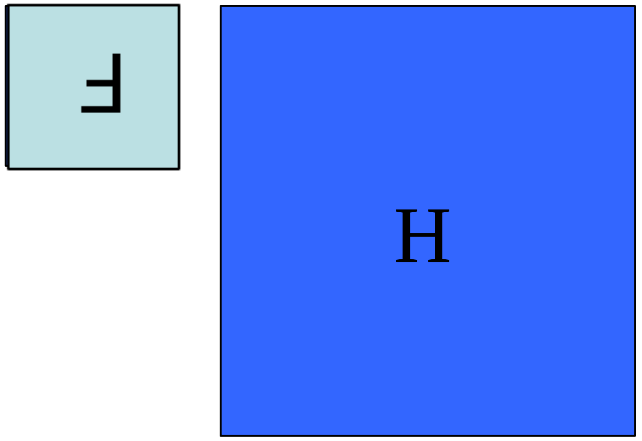
Convolution

- Convolution:
  - Flip the filter in both dimensions (bottom to top, right to left)
  - Then apply cross-correlation

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

$$G = \underset{\uparrow}{H \star F}$$

Notation for convolution operator



Convolution vs. correlation

Convolution

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

$$G = H \star F$$

Cross-correlation

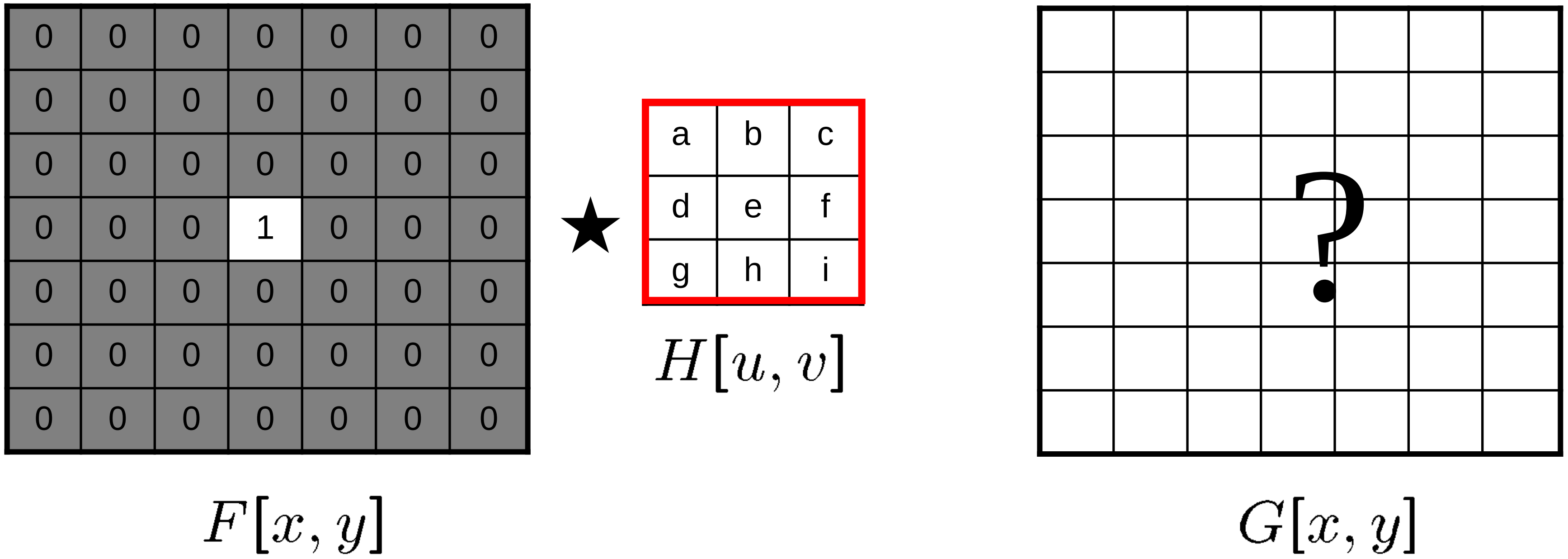
$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i+u,j+v]$$

$$G = H \otimes F$$

For a Gaussian or box filter, how will the outputs differ?

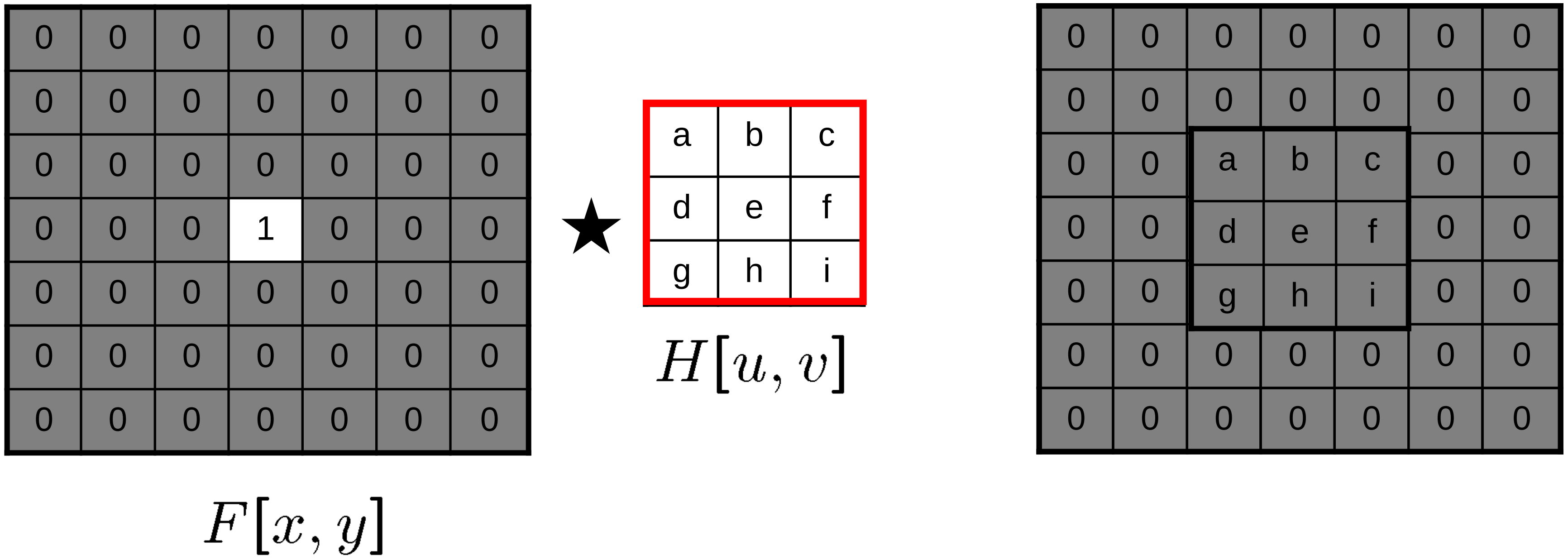
Filtering an impulse signal

What is the result of filtering the impulse signal (image) *F* with the arbitrary kernel *H*?



Filtering an impulse signal

What is the result of filtering the impulse signal (image) *F* with the arbitrary kernel *H*?



0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

$F[x,y]$

$\star$

a	b	c
d	e	f
g	h	i

$H[u,v]$

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	a	b	c	0	0
0	0	d	e	f	0	0
0	0	g	h	i	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Separability example

2D convolution
(center location only)

$$\begin{bmatrix}1&2&1\\2&4&2\\1&2&1\end{bmatrix} * \begin{bmatrix}2&3&3\\3&5&5\\4&4&6\end{bmatrix} \quad \begin{array}{r} =2+6+3=11\\ =6+20+10=36\\ =4+8+6=18\\ \hline 65 \end{array}$$

The filter factors
into a product of 1D
filters:

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

Perform convolution
along rows:

$$\begin{bmatrix}1&2&1\end{bmatrix} * \begin{bmatrix}2&3&3\\3&5&5\\4&4&6\end{bmatrix} = \begin{bmatrix}&11&\\&18&\\&18&\end{bmatrix}$$

Followed by convolution
along the remaining column:

$$\begin{bmatrix}1\\2\\1\end{bmatrix} * \begin{bmatrix}&11&\\&18&\\&18&\end{bmatrix} = \begin{bmatrix}&&\\&65&\\&&\end{bmatrix}$$

Source: K. Grauman

Convolution (decomposition)

- In general the convolution is a computer demanding operator, e.g. the 5x5 template:

```
1   4   6   4   1
4  16  24  16   4
6  24  36  24   6
4  16  24  16   4
1   4   6   4   1
```

  is implemented by 25 multiplications for each pixel; note that often complex template may be decomposed in simple 1D operators (e. g. the isotropic, monotonic decreasing template)

- The previous convolution can be decomposed in the following two 1D operators:

```
1  4  6  4  1                    1
                                 4
                                 6
                                 4
                                 1
```

  in this implementation only10 (5+5) multiplications per pixel are required

- Note that applying several filters one after another (((a * b1) * b2) * b3) is equivalent to applying one filter a * b4 where b4=(b1 * b2 * b3). If this three templates are 3x3 arrays b4 is a 7x7 template.

- Each 3x3 kernel has 9 independent values for a total of 27 values meanwhile a general 7x7 templates has 49 independent values: Not al templates are decomposable in a short sequence of smaller ones! Fortunately in important practical cases (e.g. circular symmetric and monotonic decreasing) they are.

Gradient approximations

- The gradient is a 2D vector
- The digital differential operators are implemented by template in which the sum of the kernel parameters is null: in a uniform area the result must be zero (no variation)
- The basic and historical convolution kernels have an extension limited to 2x2 and 3x3, for each of the two components

Roberts Operator

- It is the simplest solution
  - Two templates are applied M_1 and M_2, obtaining the two orthogonal gradient components:
  - $G_1=M_1*I, \quad G_2=M_2*I$
  - It is very sensible to noise
- The gradient module and phase are:

$$G_m = \sqrt{G_1^2 + G_2^2}$$

$$G_\phi = \mathrm{arctg}(G_2/G_1)+\pi/4$$

$$G_1 \quad \begin{array}{|c|c|}\hline 0 & 1 \\ \hline -1 & 0 \\ \hline \end{array}$$

$$G_2 \quad \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array}$$

Isotropic operator

- Two templates are applied M_1 and M_2, obtaining the two orthogonal gradient components:

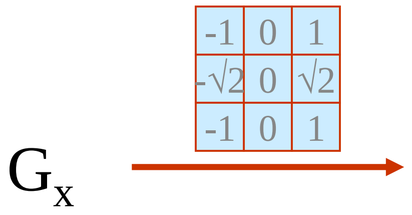
  $G_x=M_x*I,\ G_y=M_y*I$

- The gradient module and phase are:

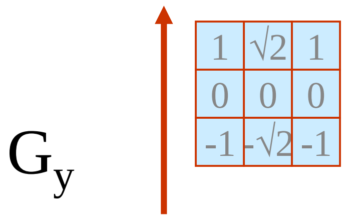
  $G_m=\sqrt{G_1^2+G_2^2}$

  $G_\phi = \mathrm{arctg}(G_y/G_x)$

- In C, Java, …:
  - phi = atan2(gy, gx)



$$\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -\sqrt{2} & 0 & \sqrt{2} \\ \hline -1 & 0 & 1 \\ \hline \end{array}$$



$$\begin{array}{|c|c|c|}\hline 1 & \sqrt{2} & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -\sqrt{2} & -1 \\ \hline \end{array}$$

Prewitt and Sobel operators

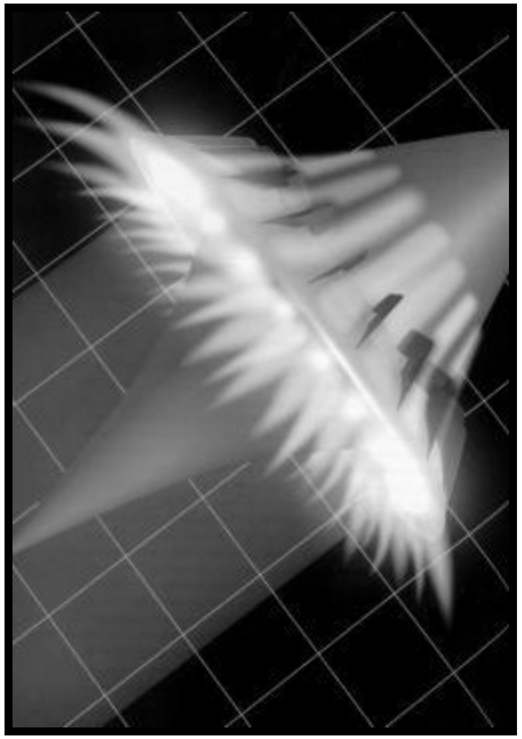
- To simplify the computation often the isotropic filter is implemented by these two simplified solutions:
  - Prewitt

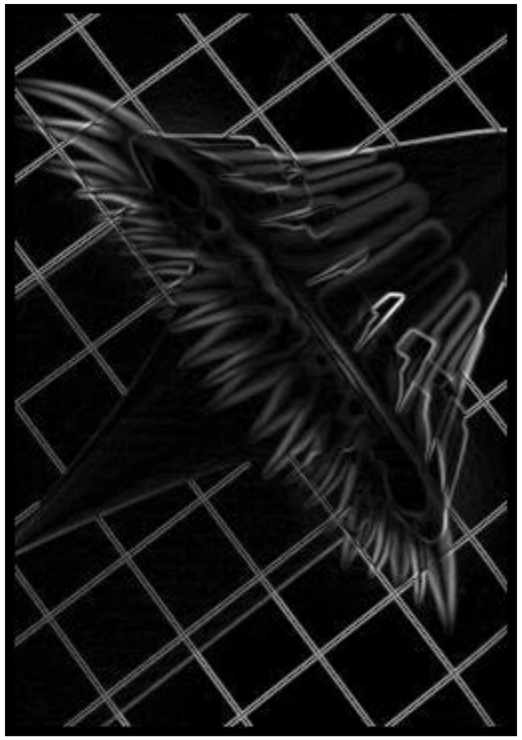
$$G_x \quad \underset{\longrightarrow}{\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array}} \qquad\qquad G_y \quad \uparrow \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array}$$

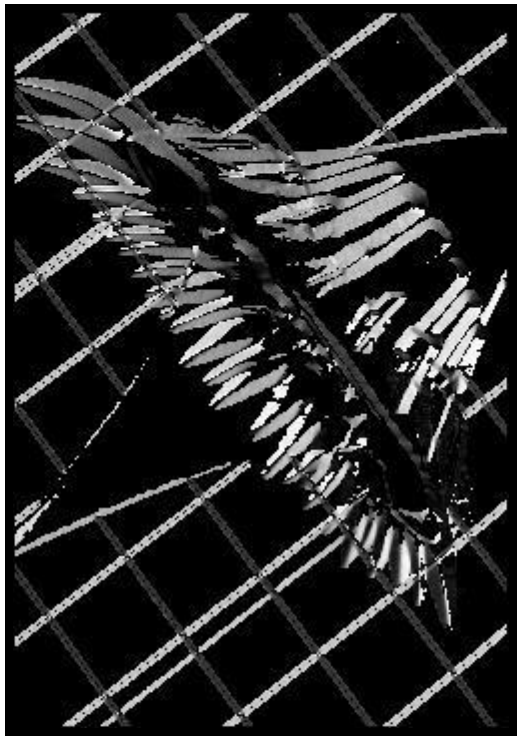
  - Sobel

$$G_x \quad \underset{\longrightarrow}{\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}} \qquad\qquad G_y \quad \uparrow \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array}$$

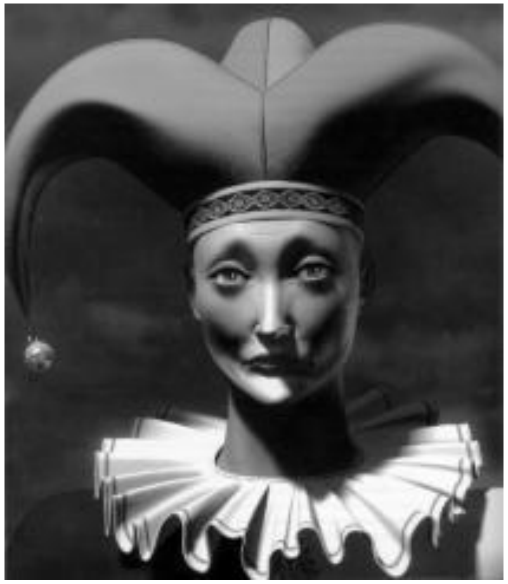
Example Sobel


Original image

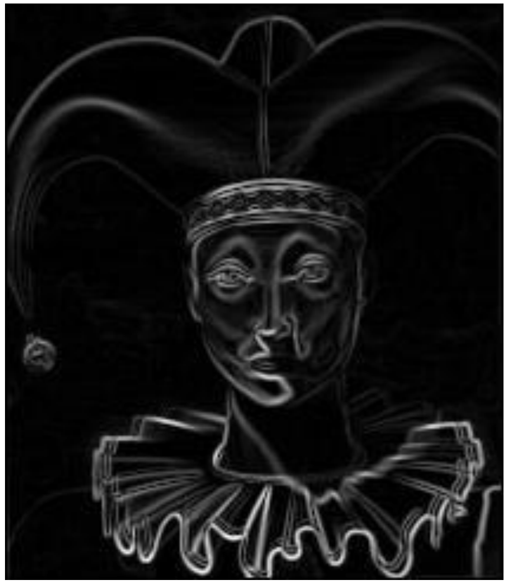

Module


Phase

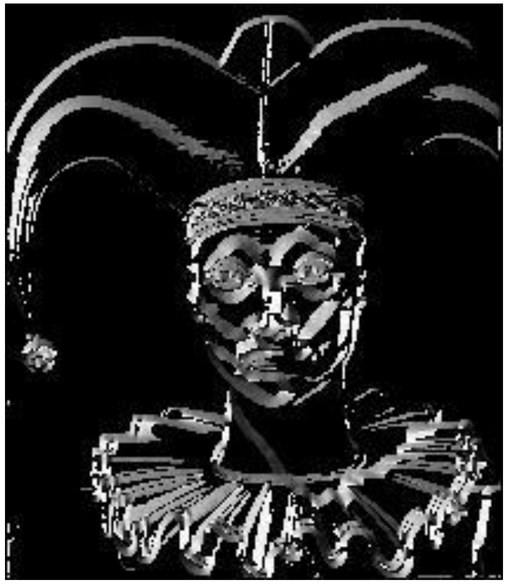
Example Sobel



Original image



Module



Phase

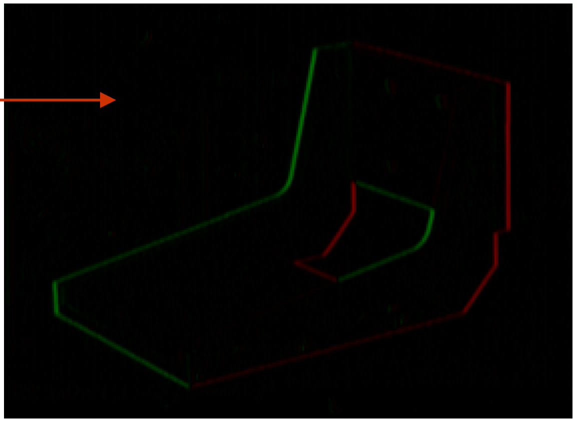
Sobel operator

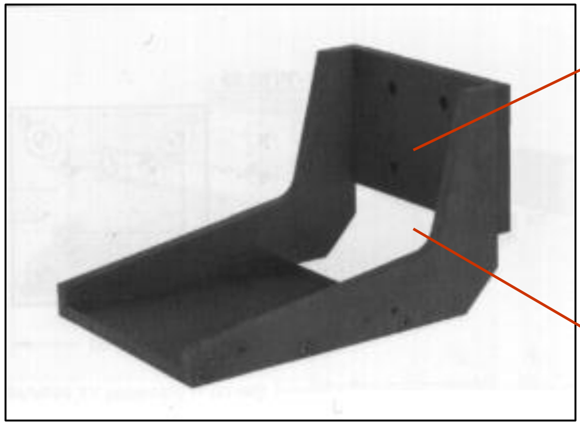
horizontal gradient

vertical contour

G_1

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

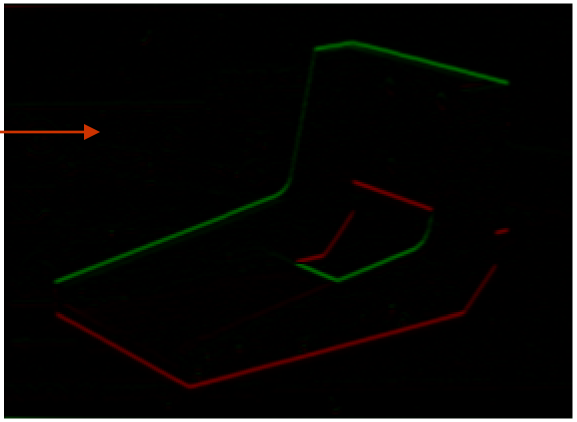




Original image

G_2

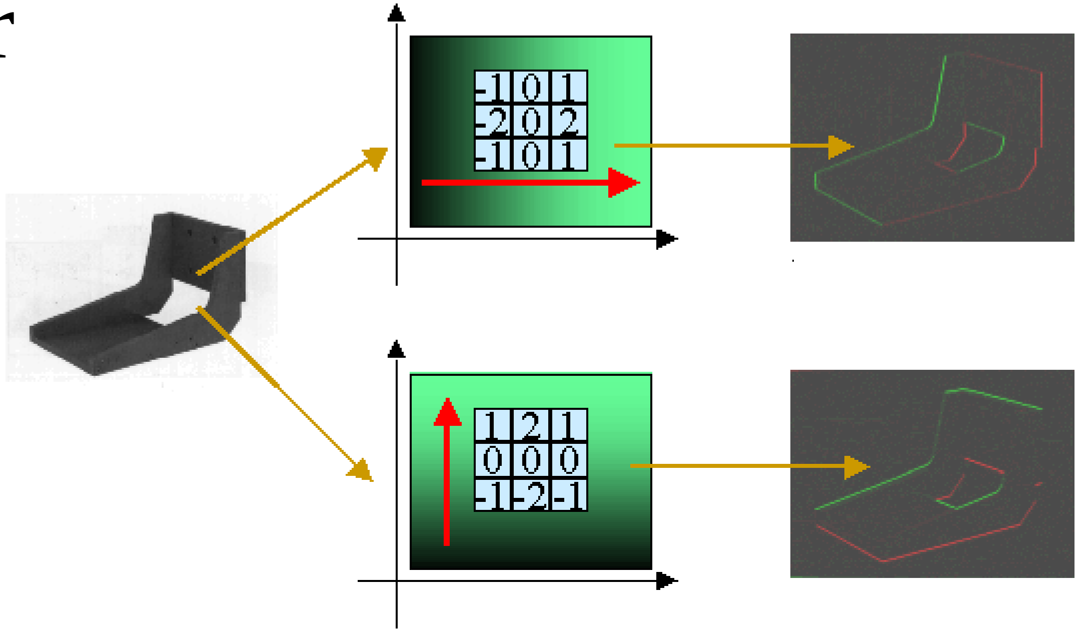
1	2	1
0	0	0
-1	-2	-1



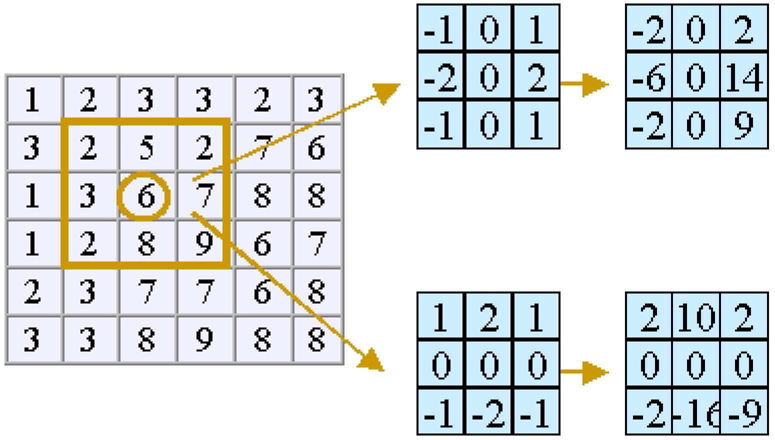
vertical gradient

horizontal contour

Sobel operator



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



1	2	3	3	2	3
3	2	5	2	7	6
1	3	6	7	8	8
1	2	8	9	6	7
2	3	7	7	6	8
3	3	8	9	8	8

$$\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix} \rightarrow \begin{bmatrix}-2 & 0 & 2\\ -6 & 0 & 14\\ -2 & 0 & 9\end{bmatrix}$$

$G_x=15$

$G\approx 28$

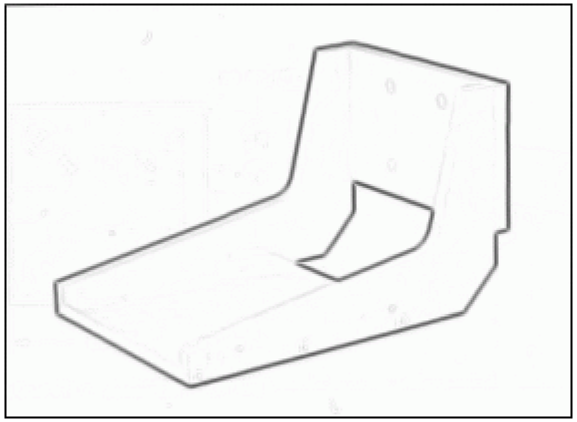
$\phi=\mathrm{arctg}(-13/15)$

$$\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix} \rightarrow \begin{bmatrix}2 & 10 & 2\\ 0 & 0 & 0\\ -2 & -16 & -9\end{bmatrix}$$

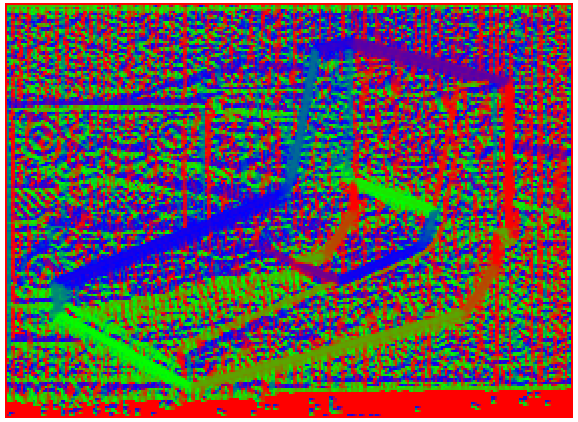
$G_y=-13$

Sobel operator

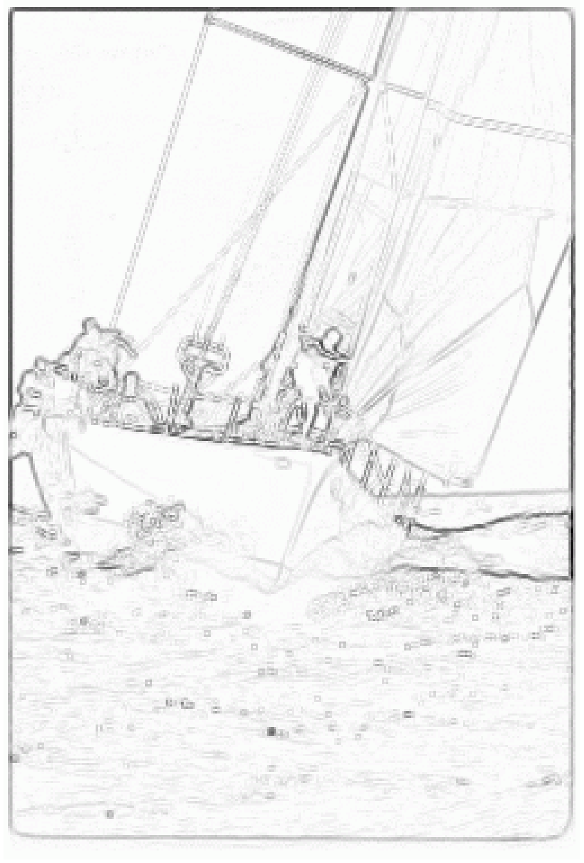
Module

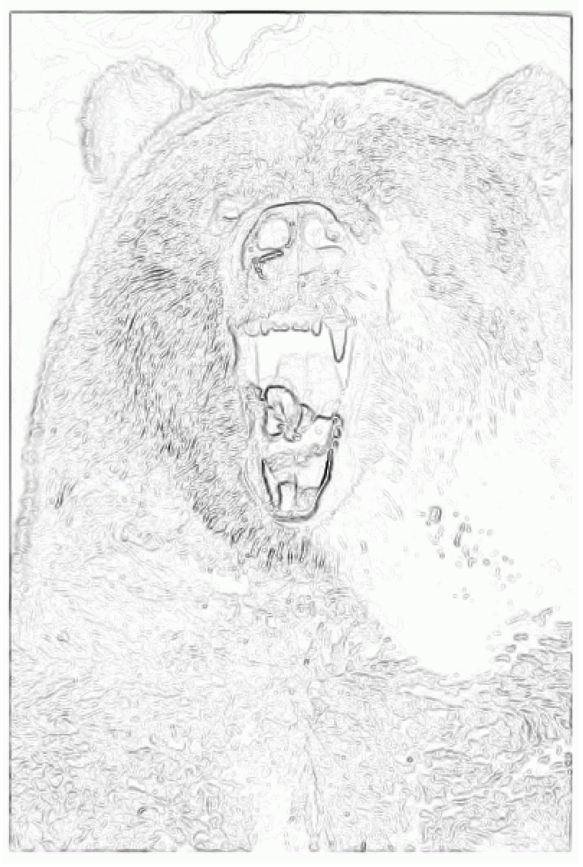


Phase

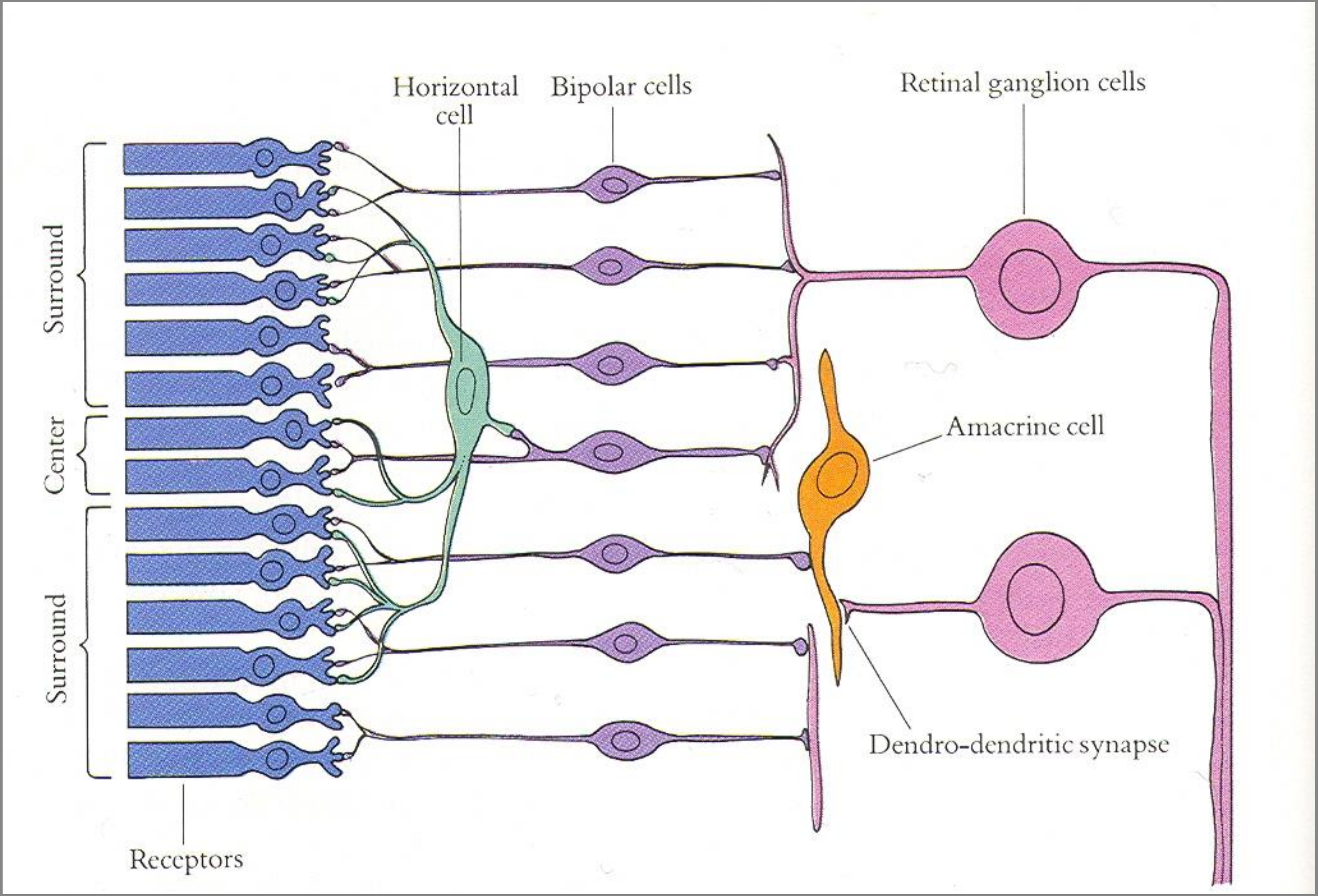


Example (module)





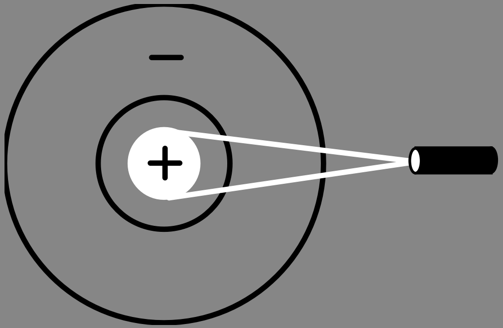
Lateral inhibition

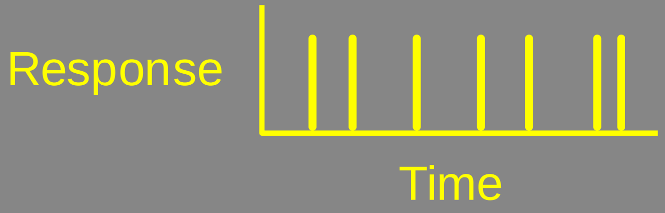


Retinal Receptive Fields

Receptive field structure in ganglion cells:

On-center Off-surround





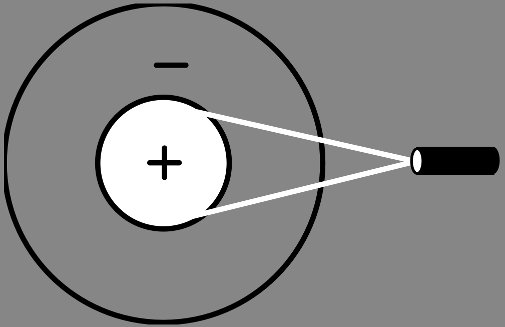
Stimulus condition

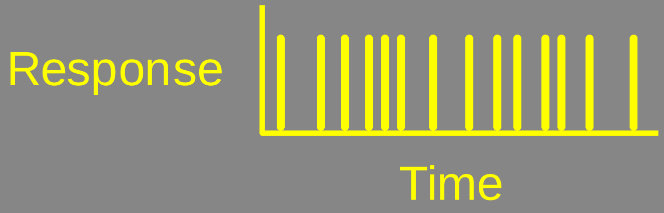
Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:

On-center Off-surround





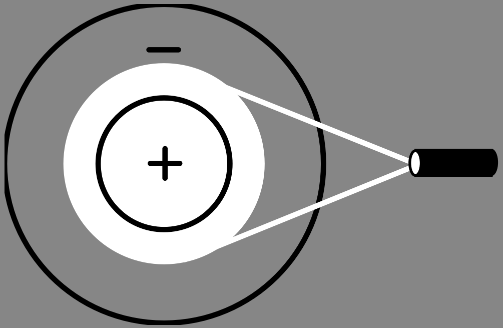
Stimulus condition

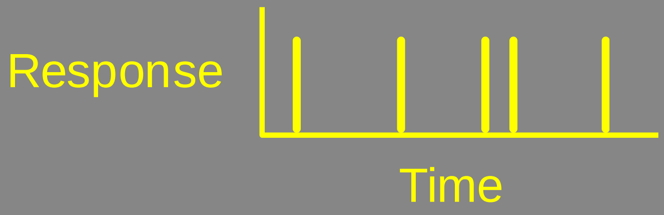
Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:

On-center Off-surround





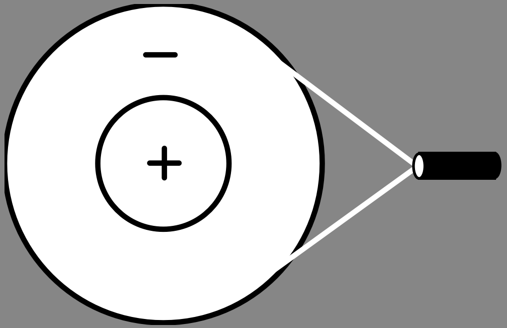
Stimulus condition

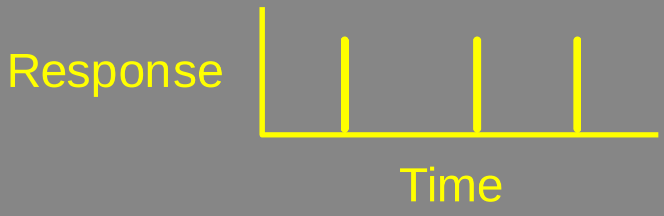
Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:

On-center Off-surround





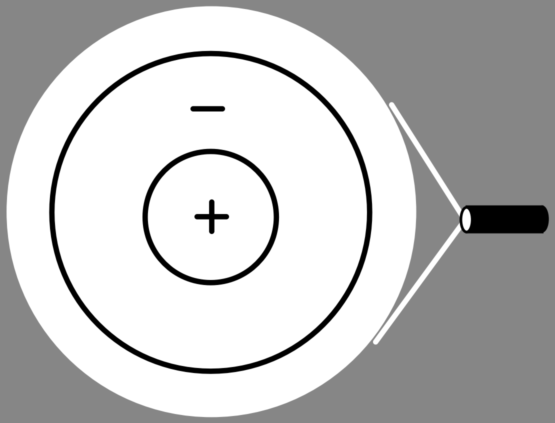
Stimulus condition

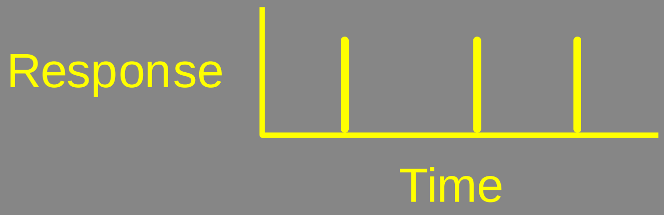
Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:

On-center Off-surround





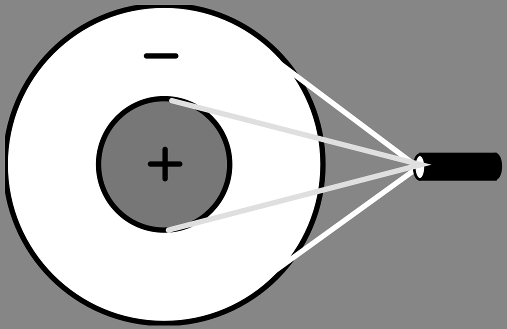
Stimulus condition

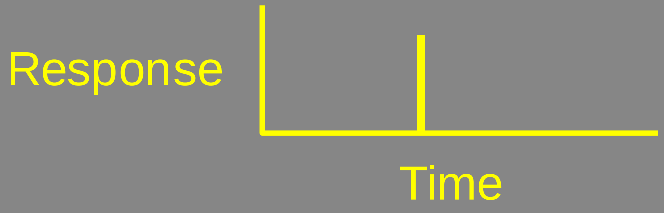
Electrical response

Retinal Receptive Fields

Receptive field structure in ganglion cells:

On-center Off-surround



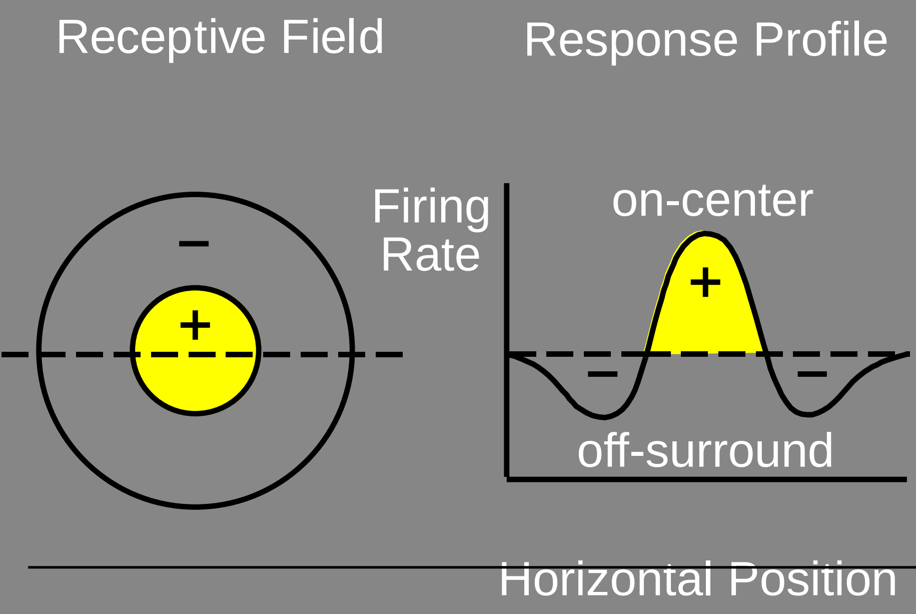


Stimulus condition

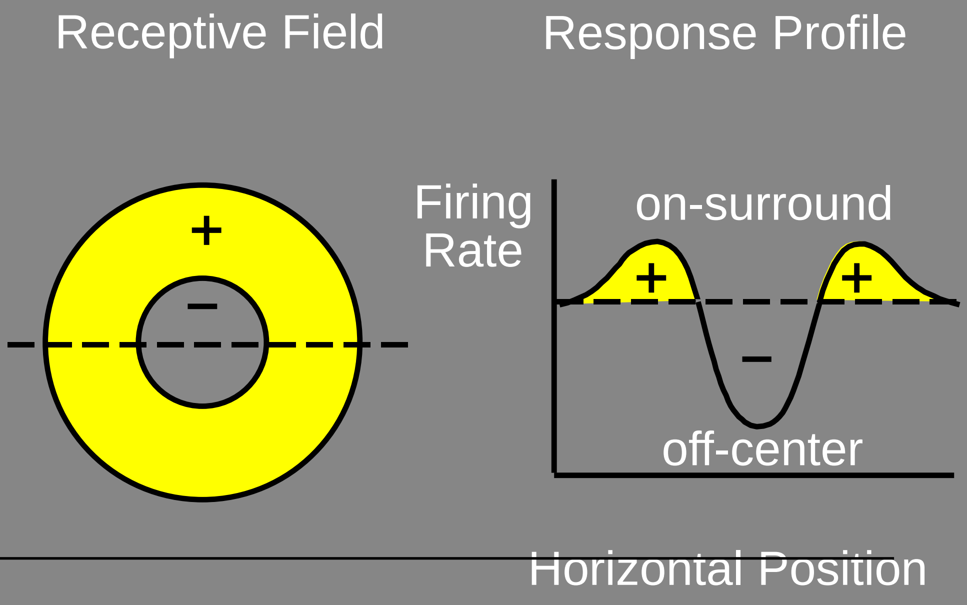
Electrical response

Retinal Receptive Fields

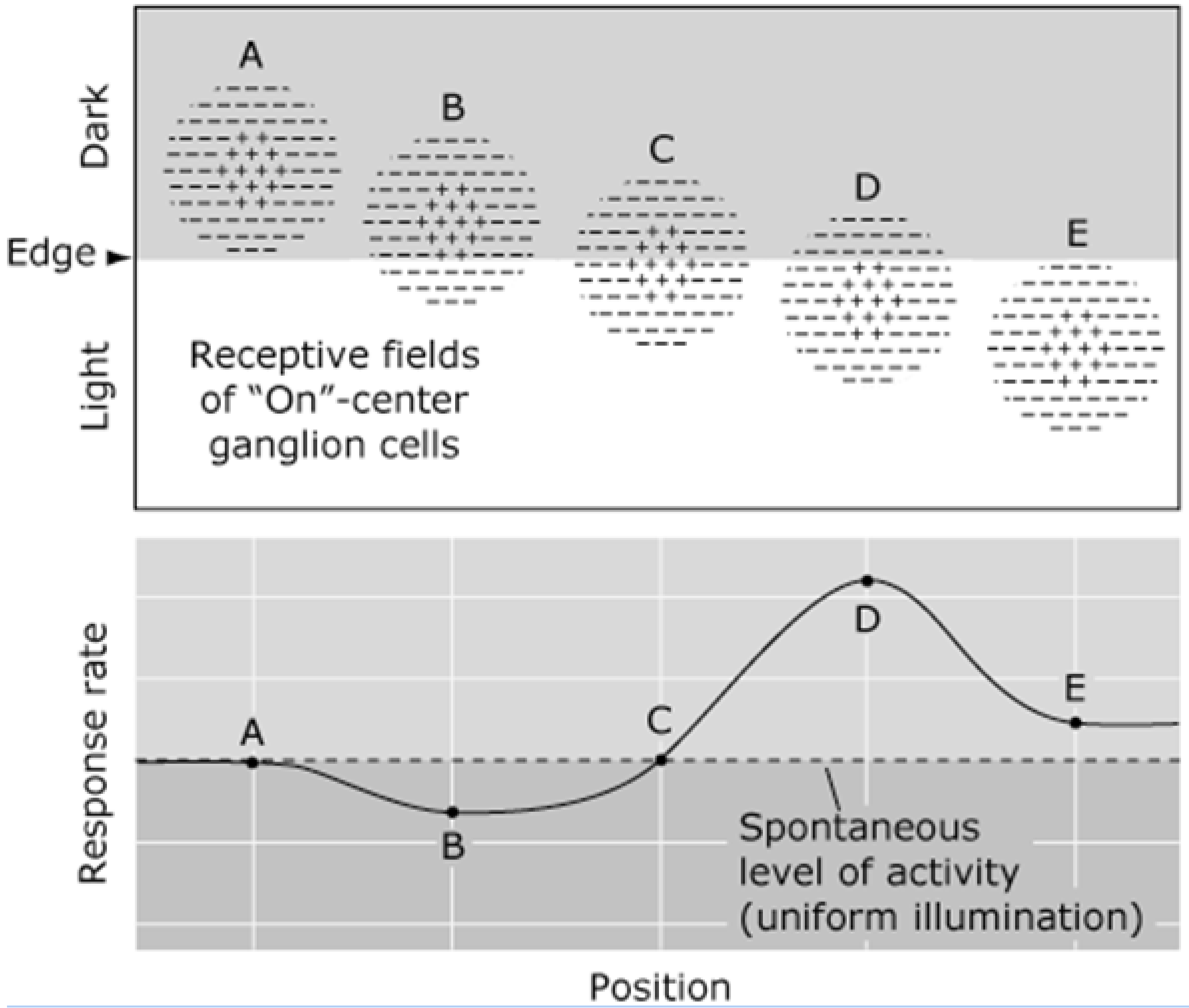
RF of On-center Off-surround cells



RF of Off-center On-surround cells

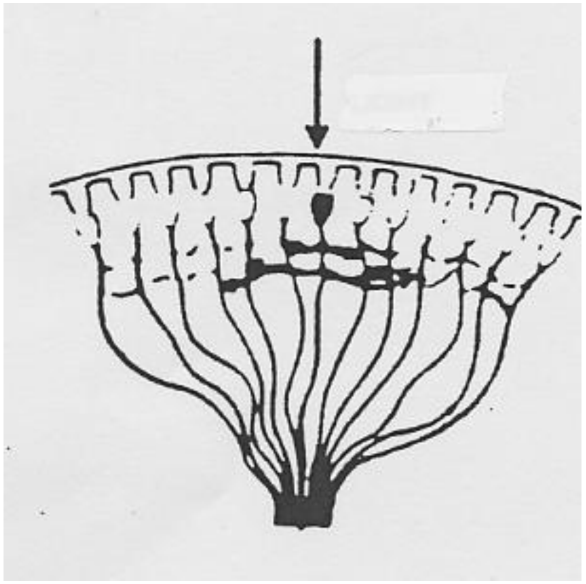


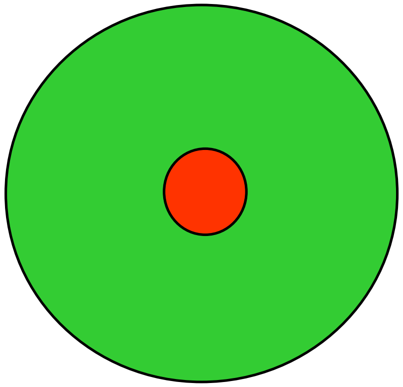
Lateral inhibition

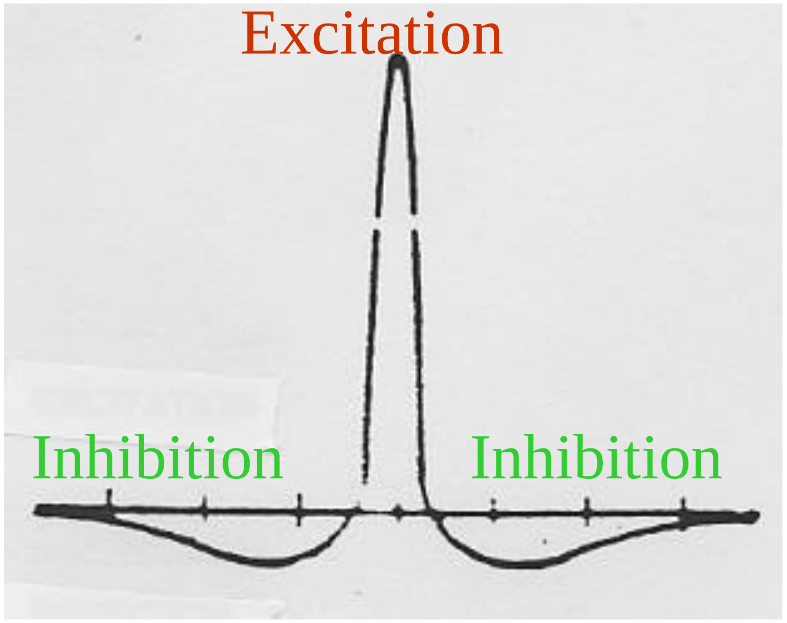


Dark
Edge
Light
A
B
C
D
E
Receptive fields of "On"-center ganglion cells
Response rate
Position
Spontaneous level of activity (uniform illumination)

Lateral inhibition







Lateral inhibition: theDoG filter

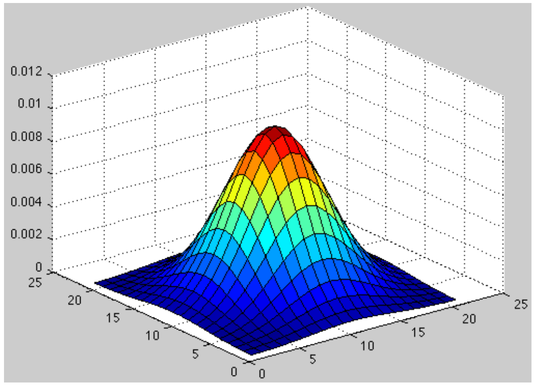
- The retina receptor apply a lateral inhibition mechanism.
- The implementation of this mechanism can be done by a filter obtained by the difference of two Gaussian of equal area, having different σ (and amplitude):

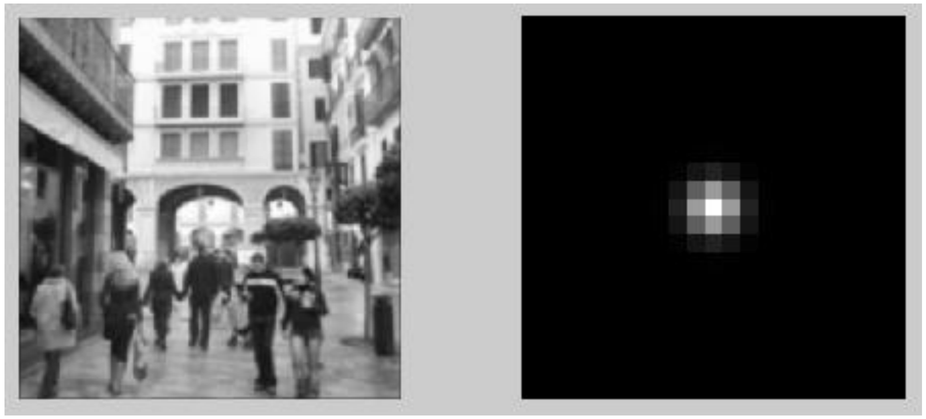
$$\frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

- The 'zero-crossing' correspond to the border points. An advantage of this technique is that the produced contour are closed.

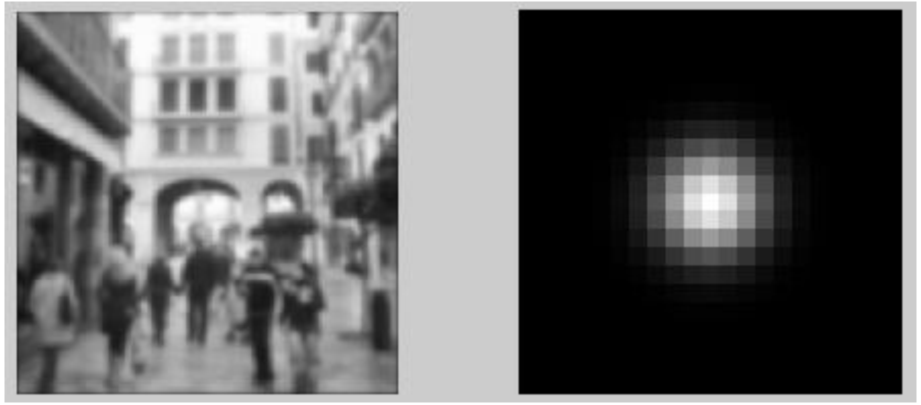
Gaussian filter

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

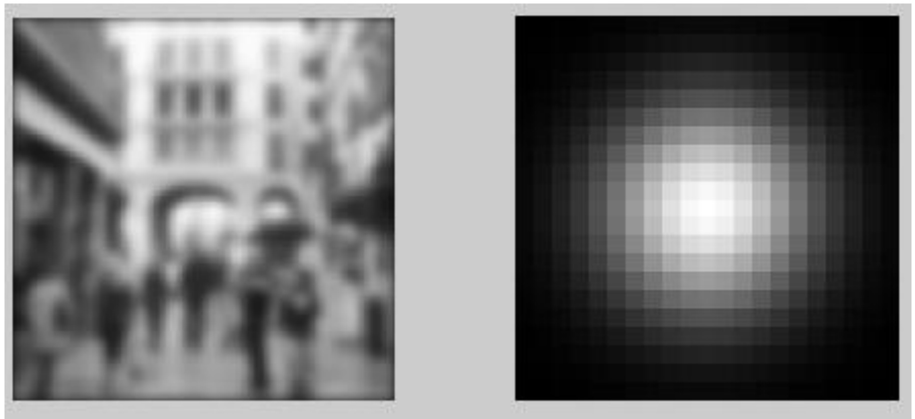




σ=1



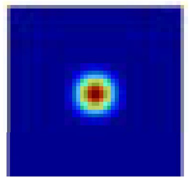
σ=2

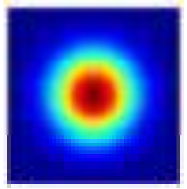


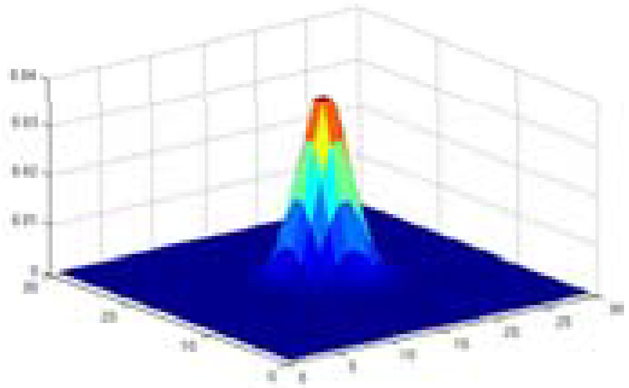
σ=4

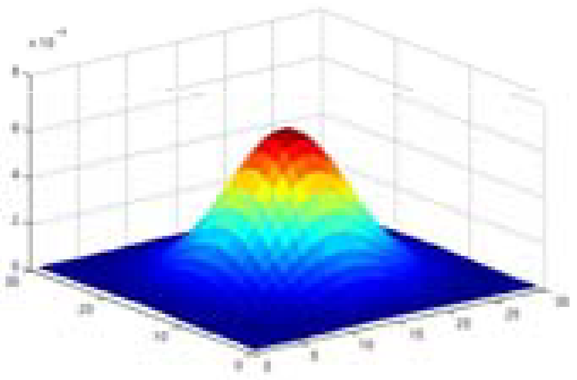
Gaussian filters

- What parameters matter here?
- **Variance** of Gaussian: determines extent of smoothing









σ = 2 with 30 x 30 kernel

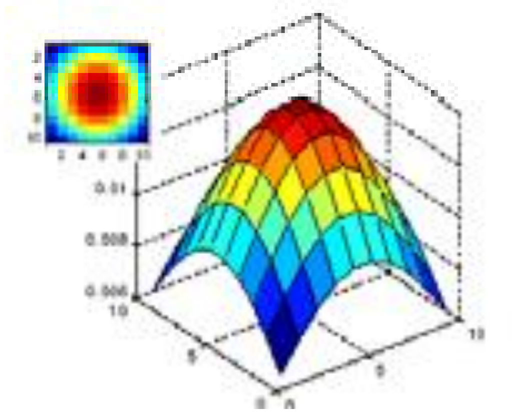
σ = 5 with 30 x 30 kernel

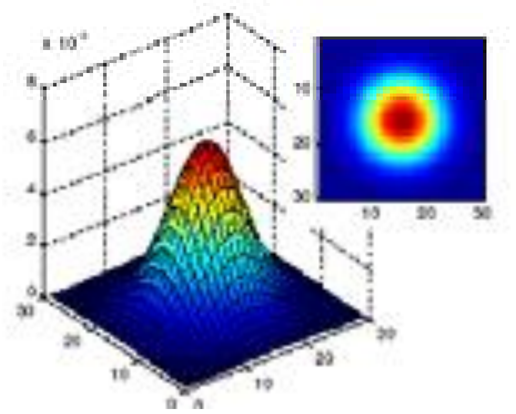
Gaussian filters

- What parameters matter here?

  Size of kernel or mask

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





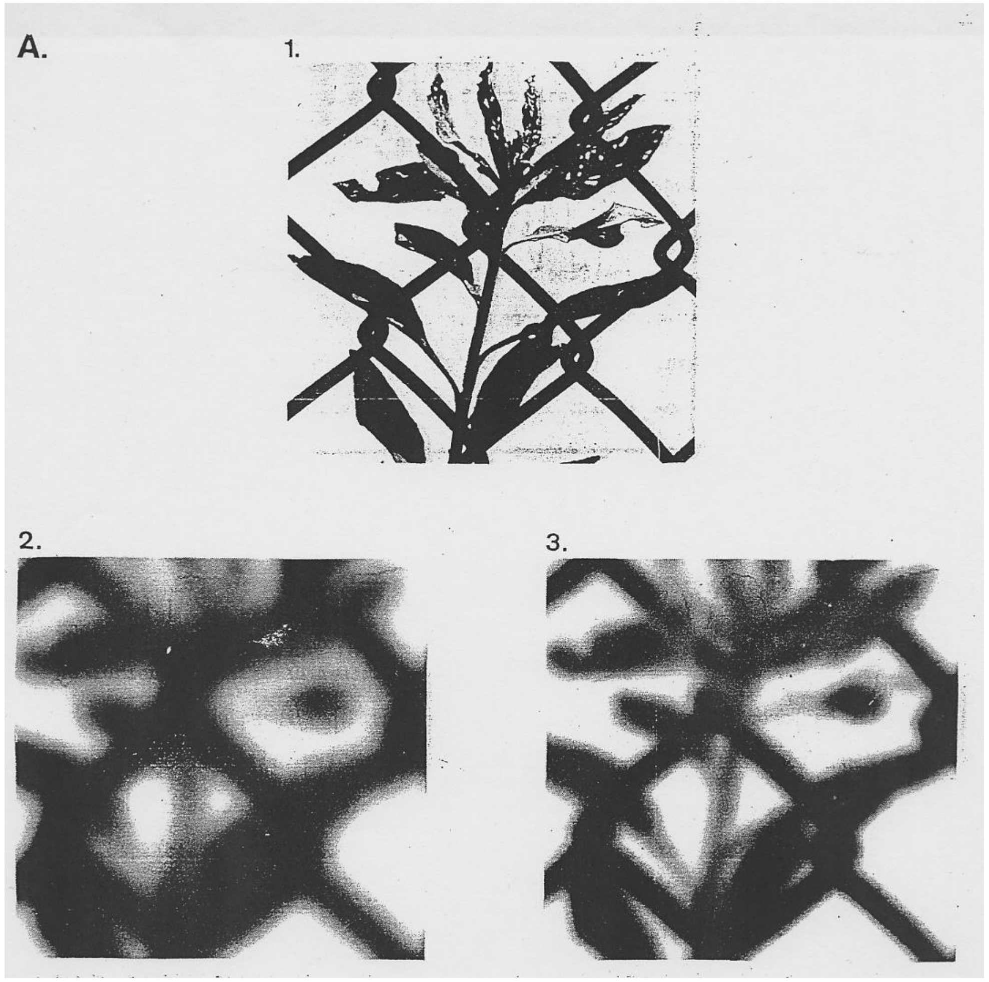
σ = 5 with 10 x 10 kernel σ = 5 with 30 x 30 kernel

Gaussian Filter

1 Original image

2 Filtered image σ=8

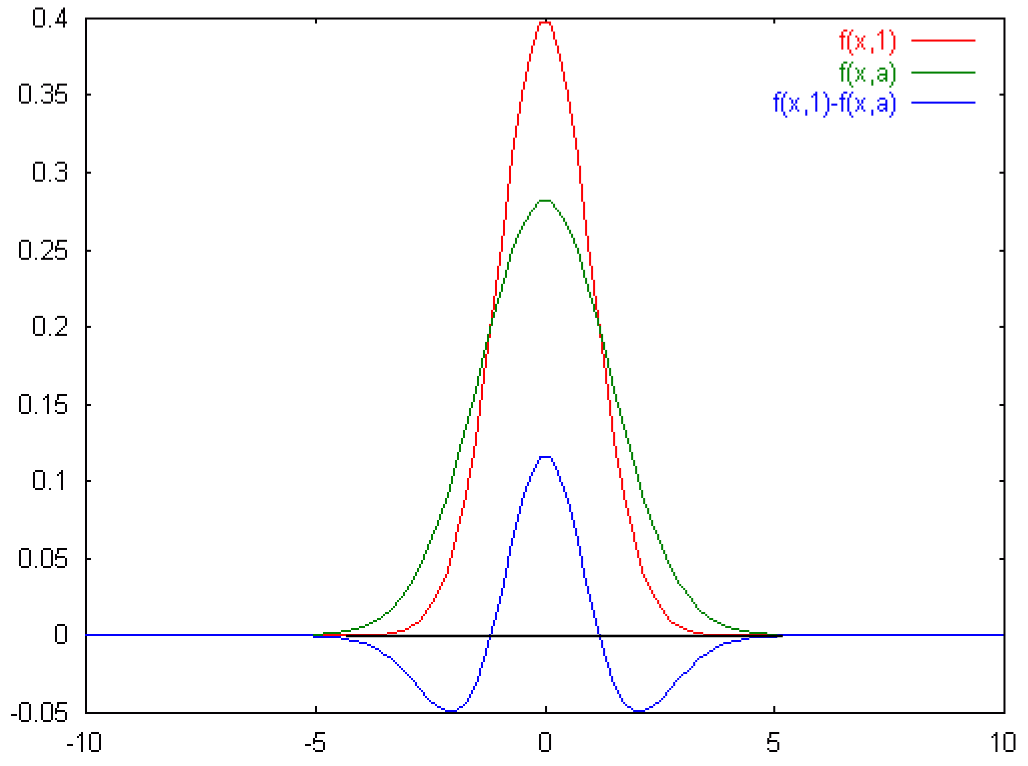
3 Filtered image σ=4



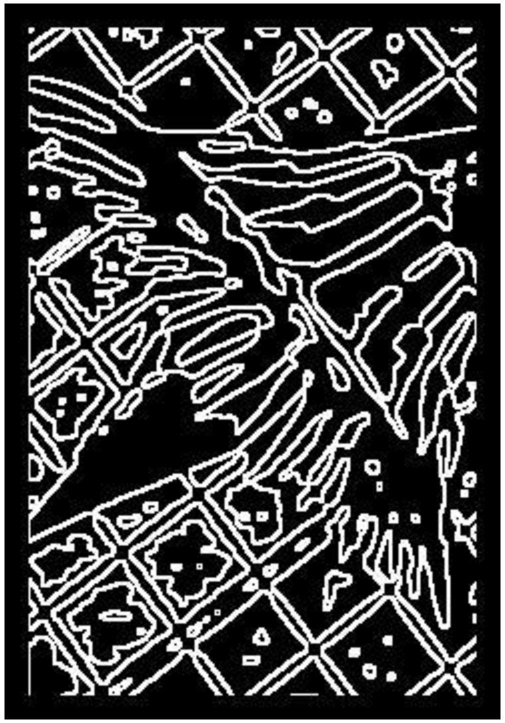
The DoG operator

- This operator is called usually Difference of Gaussians (DoG)
- The best results are obtained maintaining the external Gaussian as large as possible but avoiding to include more than one border
- The internal Gaussian is optimized if it covers just the transition area
- Complex scene are better analyzed if a set of different DoG filters with various σ are applied.

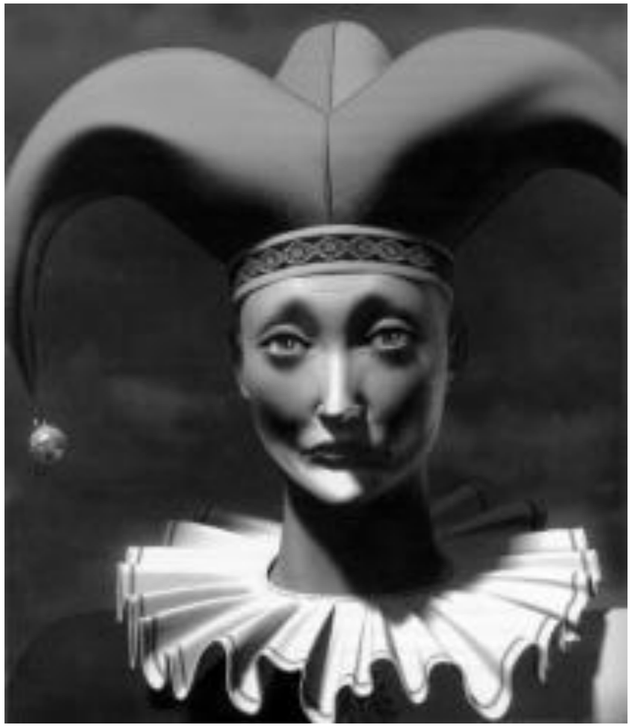
The DoG operator

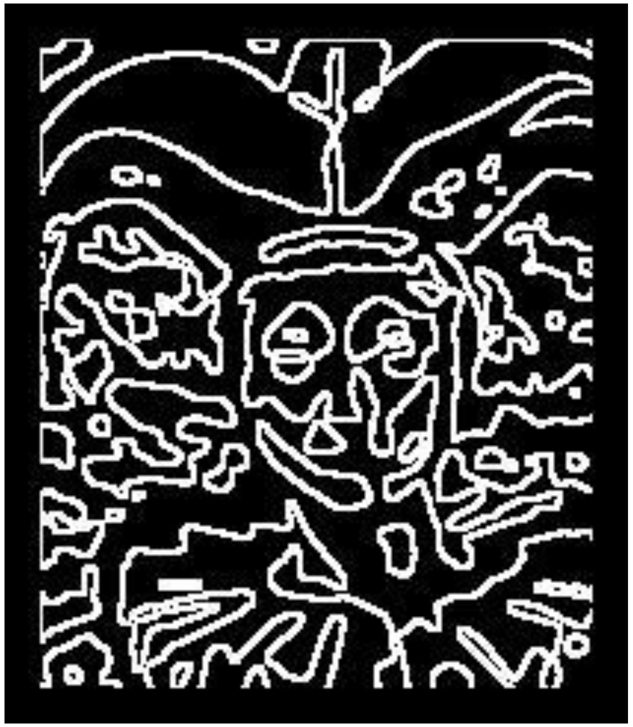


DoG Example



DoG Example





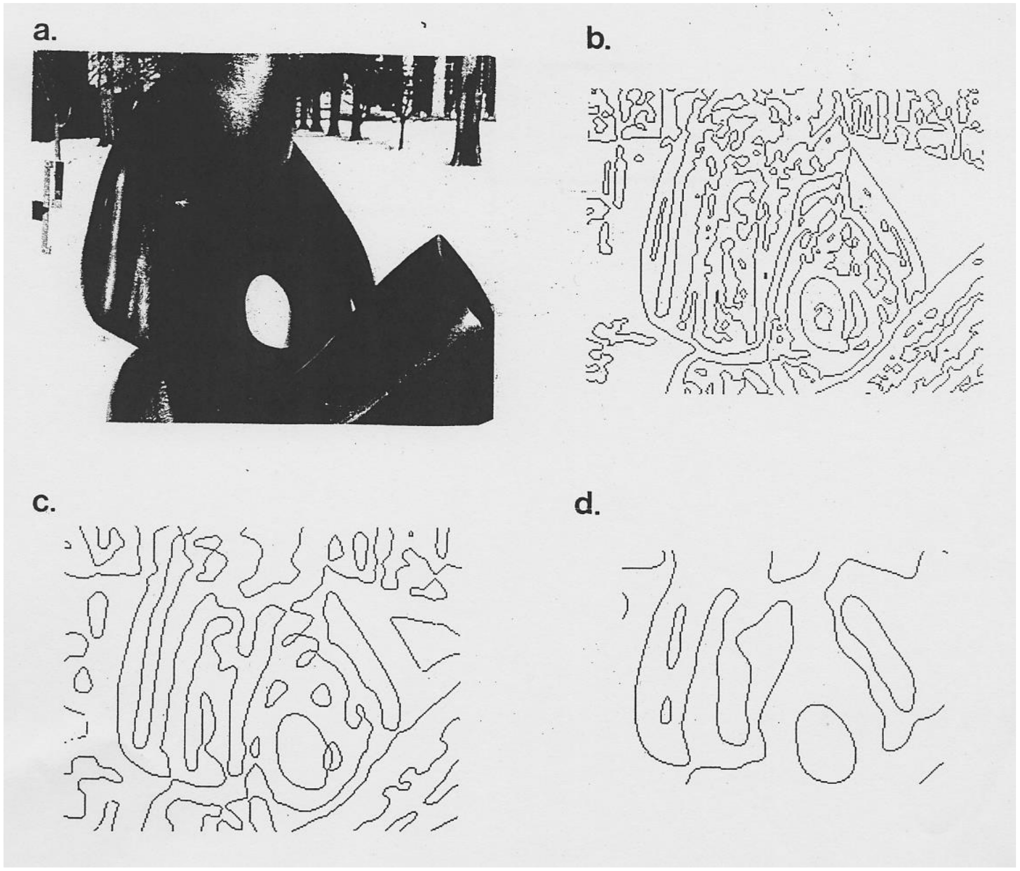
DoG: σ dependence

Original

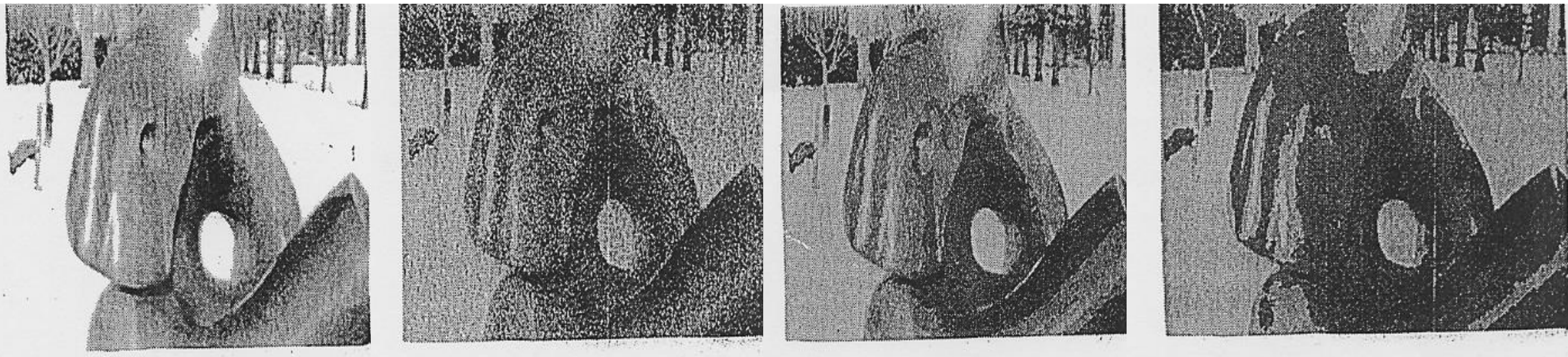
σ = 6

σ = 12

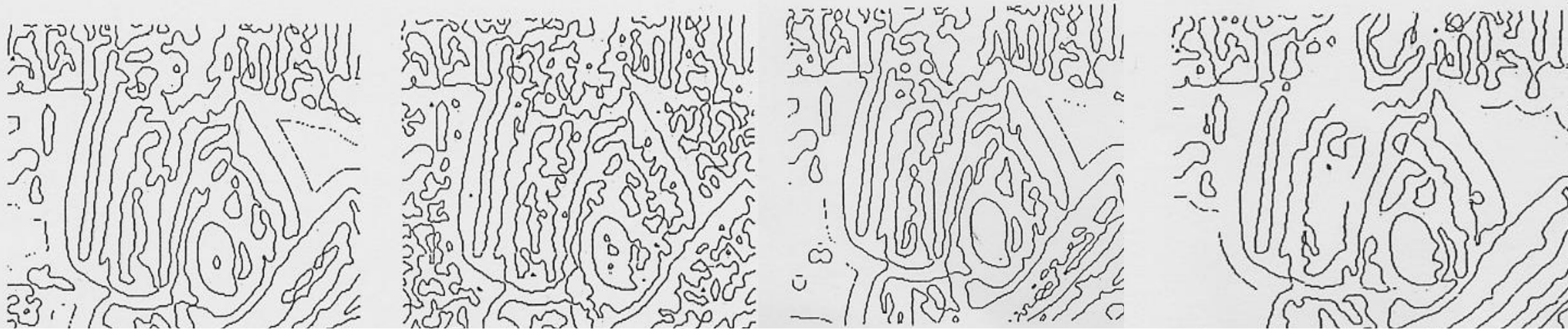
σ = 24



DoG: discretization of grey level and noise

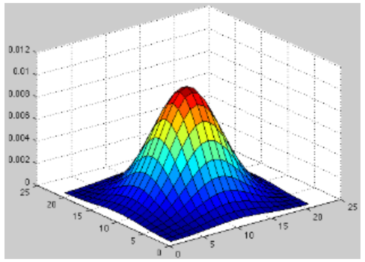


Original　　with noise　　16 grey levels　　8 grey levels



Laplacian

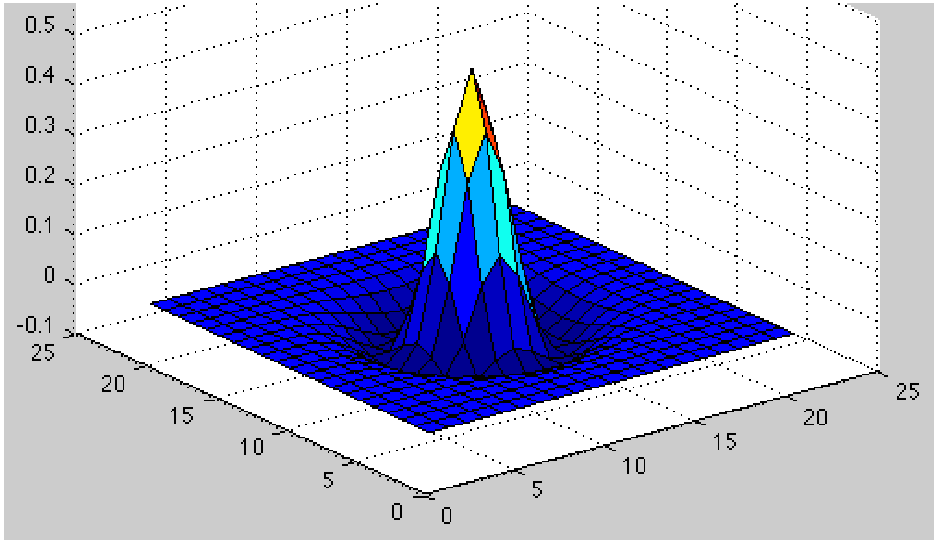
$$h(x,y)=\frac{1}{2\pi\sigma^2}e^{-\frac{x^2+y^2}{2\sigma^2}}$$



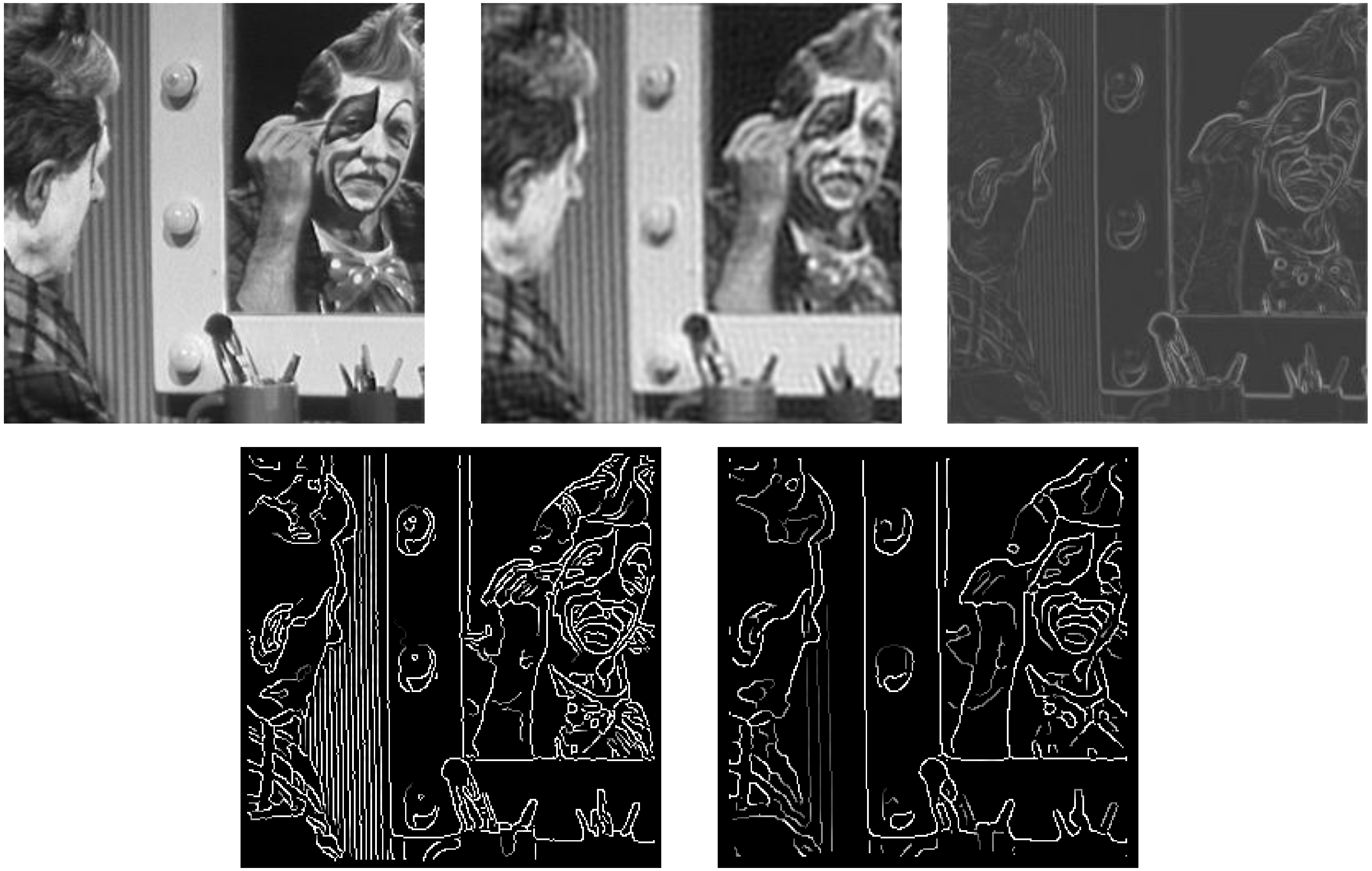
$$\nabla^2 g\otimes h=\left(\frac{\partial^2 g(x,y)}{\partial x^2}+\frac{\partial^2 g(x,y)}{\partial y^2}\right)\otimes h(x,y)$$

$$\nabla^2 g\otimes h=g\otimes\nabla^2 h$$

$$\nabla^2 h(x,y)=\left(\frac{x^2+y^2}{\sigma^4}-\frac{2}{\sigma^2}\right)\otimes h(x,y)$$



John Canny, Rachid Deriche, etc operators



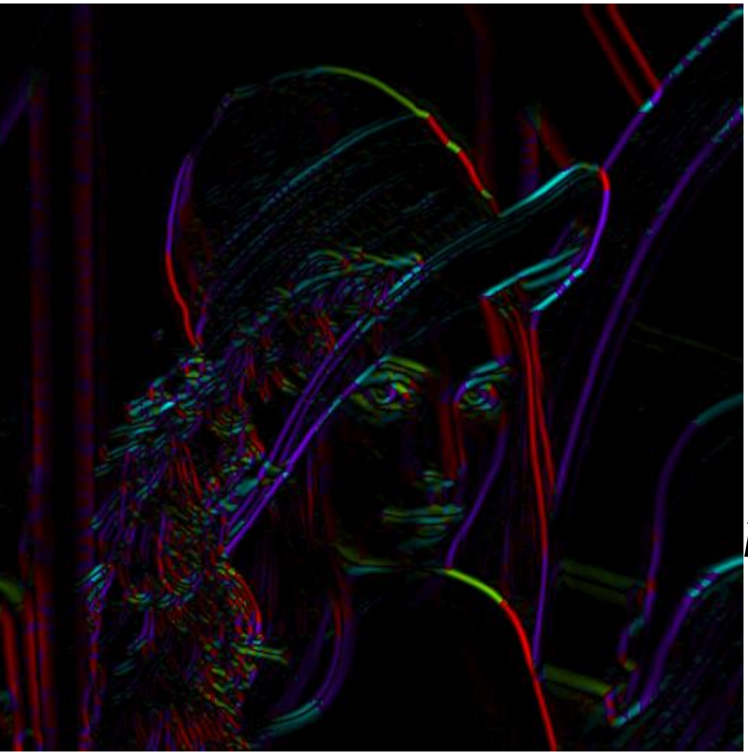
Canny edge detector (CED)

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

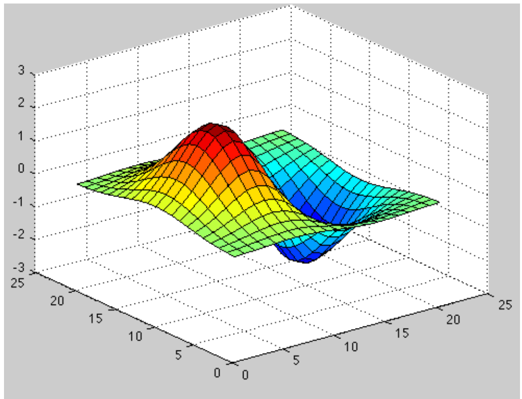
- MATLAB: **`edge(image, ‘canny’);`**
- **`>>help edge`**

Source: D. Lowe, L. Fei-Fei

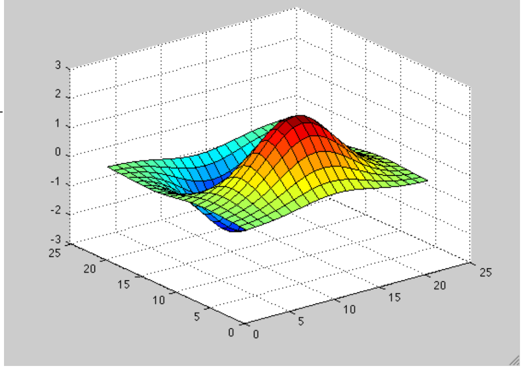
CED: b) magnitude and orientation of gradient



$$h_x(x,y) = \frac{\partial h(x,y)}{\partial x} = \frac{-x}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

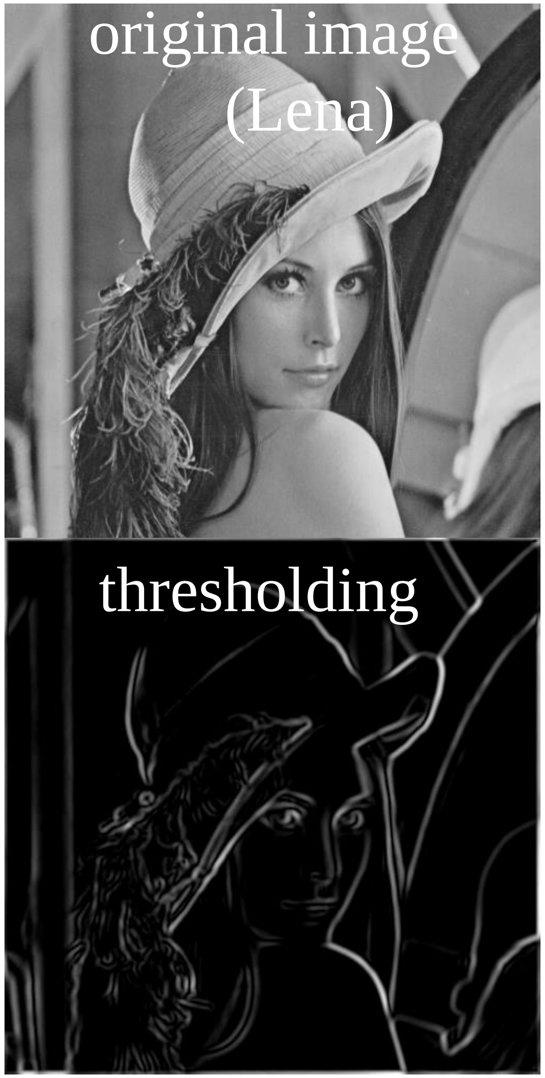


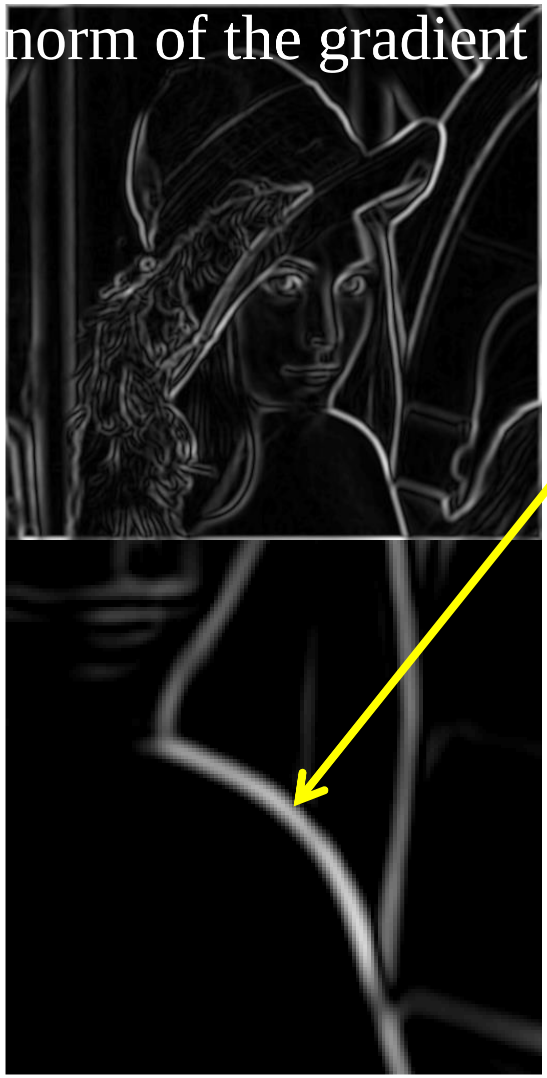
$$h_y(x,y) = \frac{\partial h(x,y)}{\partial y} = \frac{-y}{2\pi\sigma^4} e^{-\frac{x^2+y^2}{2\sigma^2}}$$



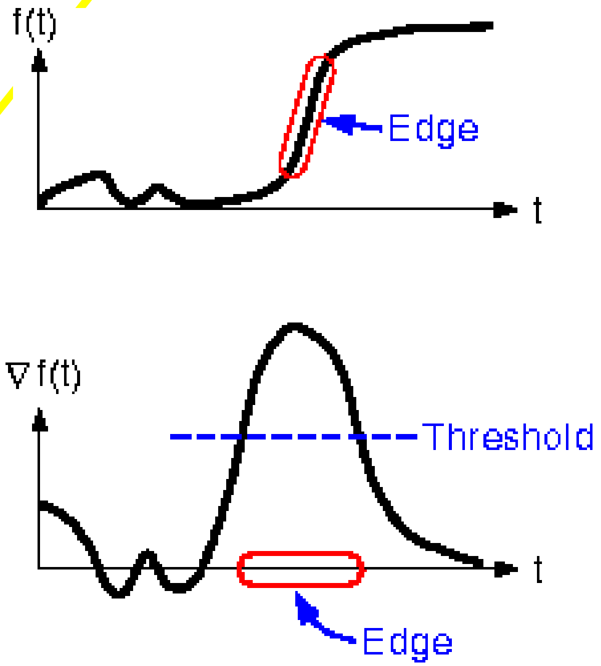
Magnitude: *Edge strength* $\sqrt{h_x(x,y)^2 + h_y(x,y)^2}$ Angle: *Edge normal* $\arctan\left(\frac{h_y(x,y)}{h_x(x,y)}\right)$

CED: c) Non-maximum suppression

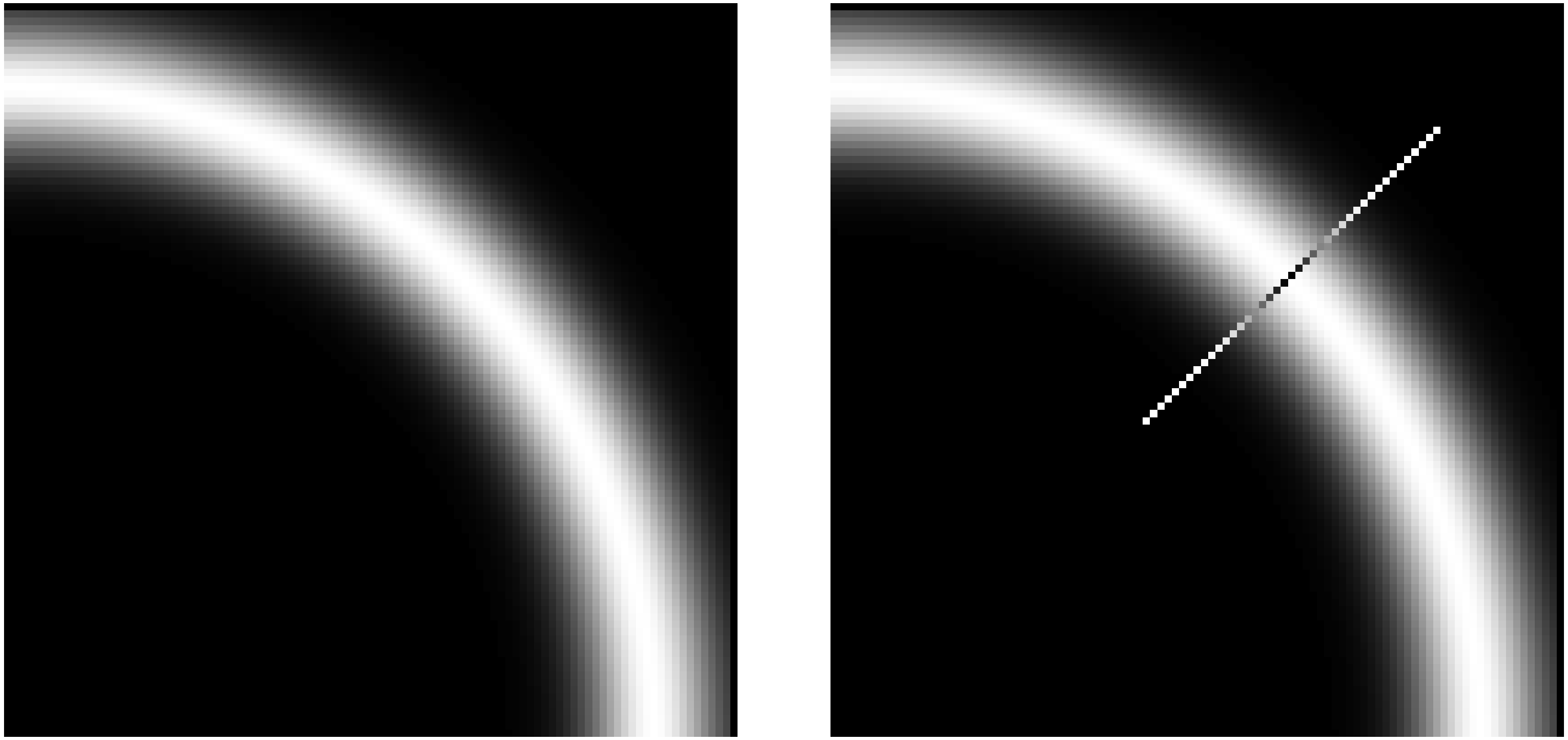




How to turn these thick regions of the gradient into curves?



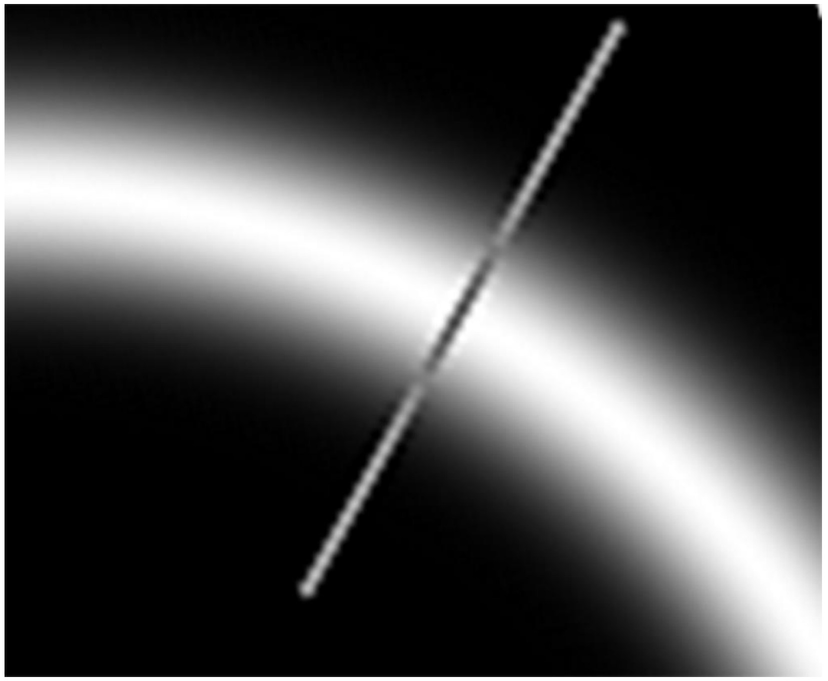
CED: c) Non-maximum suppression

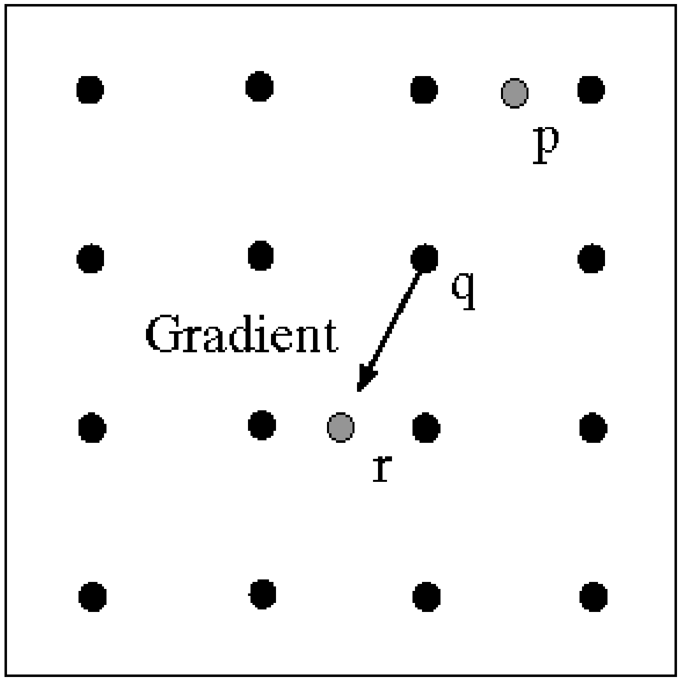


We wish to mark points along the curve where the magnitude is biggest. We can do this by looking for a maximum along a slice normal to the curve - non-maximum suppressin. These points should form a curve. There are then two algorithmic issues: at which point is the maximum, and where is the next one?

Forsyth, 2002

CED: Non-maximum suppression

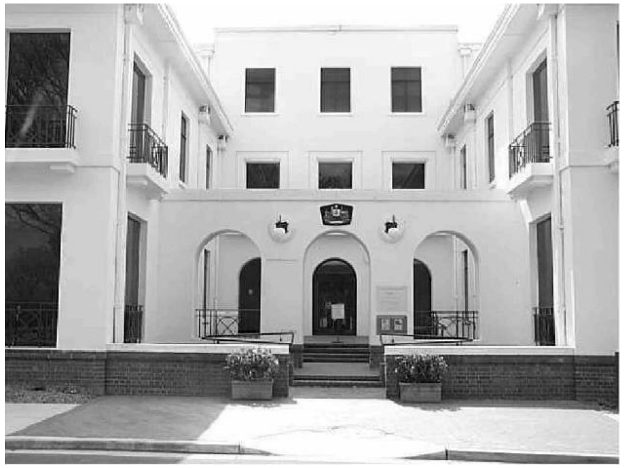


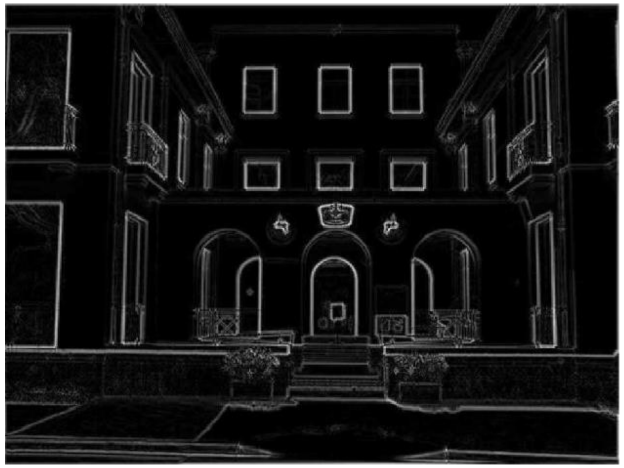


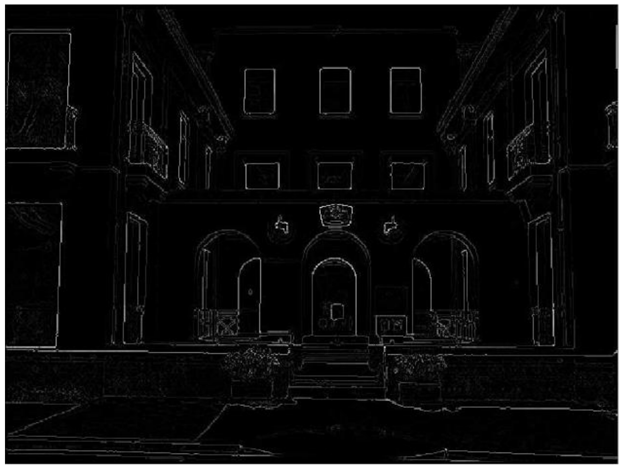
Check if pixel is local maximum along gradient direction, select single max across width of the edge

- requires checking interpolated pixels p and r

Examples: Non-Maximum Suppression


Original image

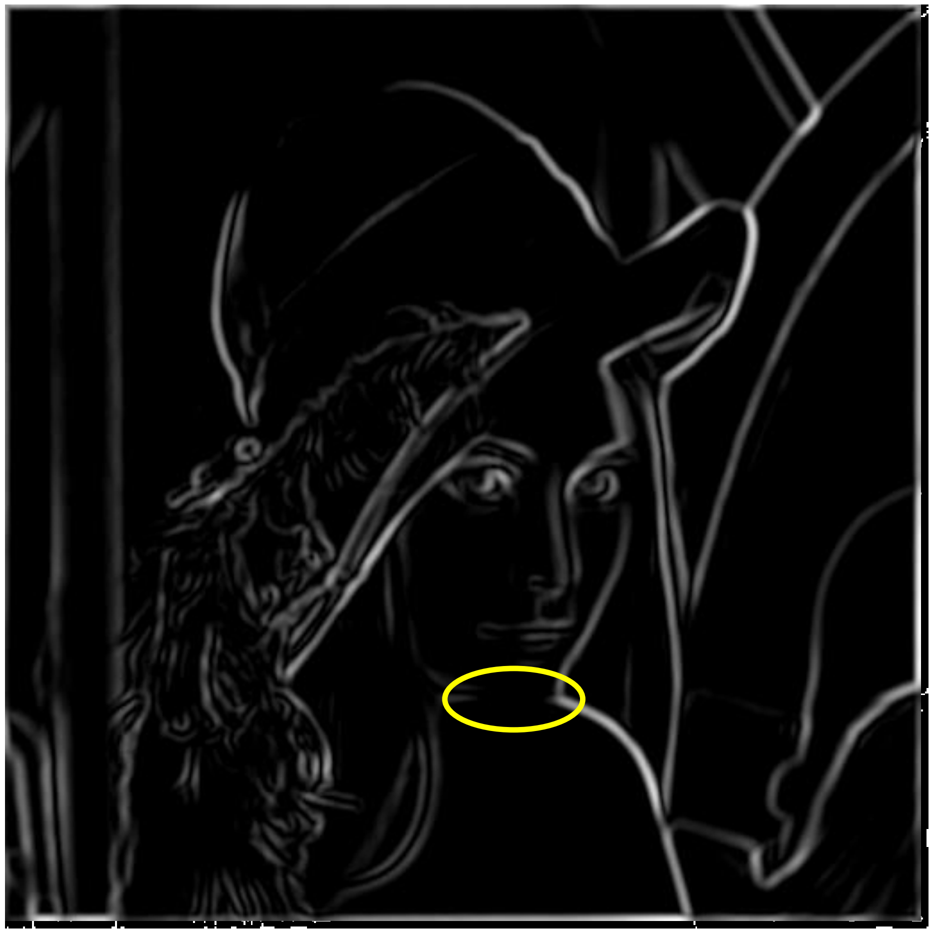

Gradient magnitude


courtesy of G. Loy

Non-maxima suppressed

Slide credit: Christopher Rasmussen

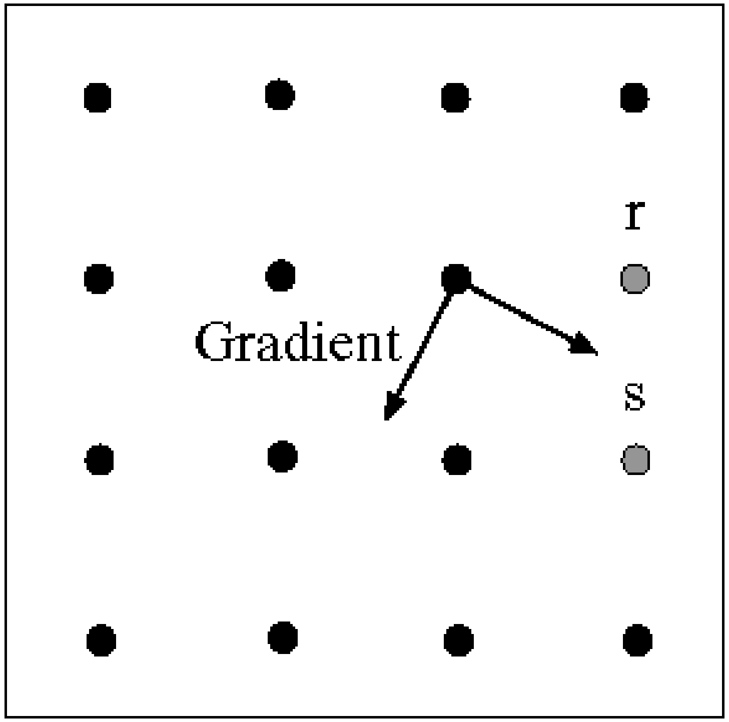
CED: d) Linking and thresholding (**hysteresis**)



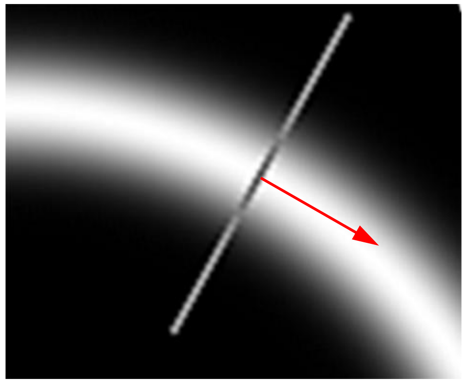
Problem: pixels along this edge didn't survive the thresholding

Thinning (non-maximum suppression)

CED: Predicting the next edge point



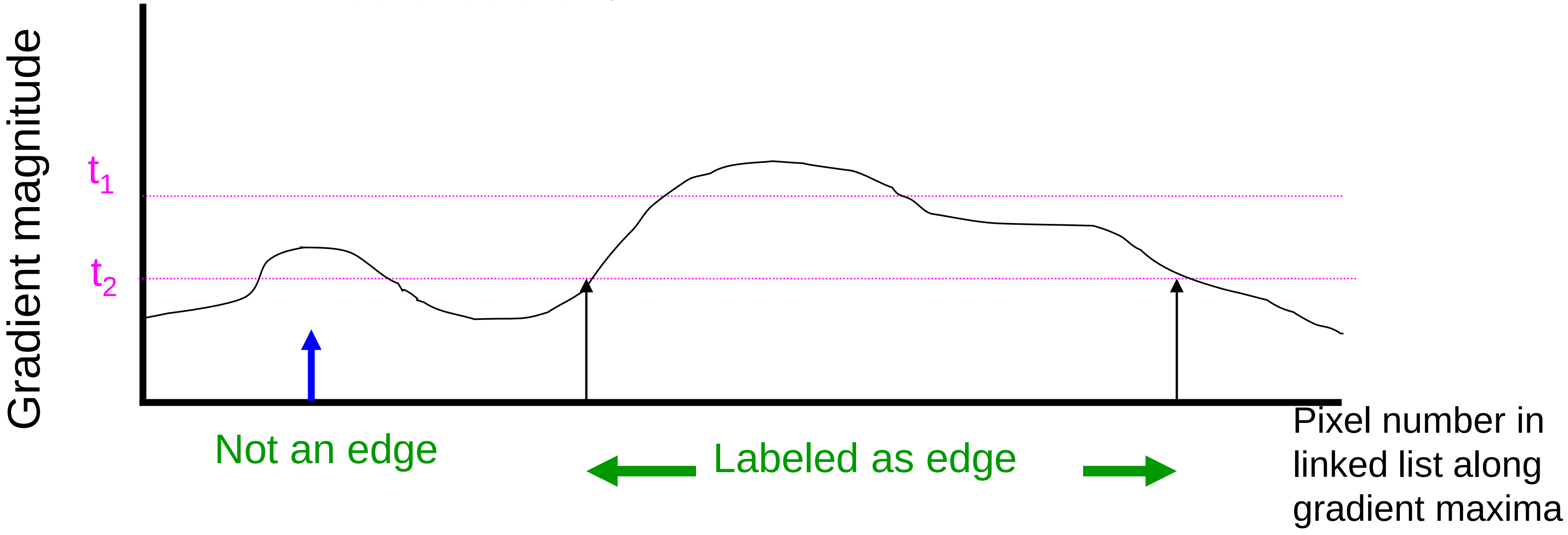
Assume the marked point is an edge point. Then we construct the tangent to the edge curve (which is normal to the gradient at that point) and use this to predict the next points (here either r or s).



Forsyth, 2002

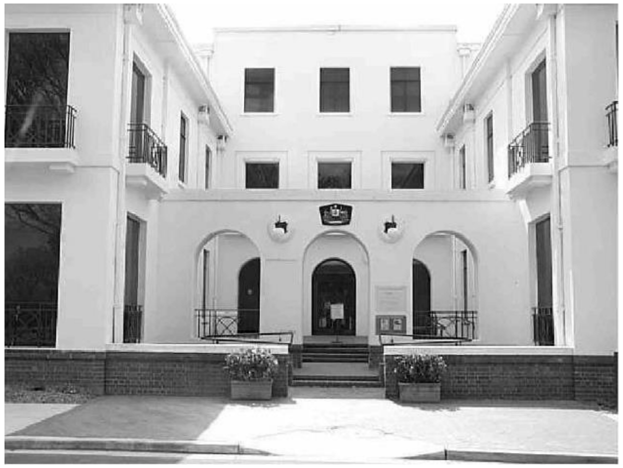
CED: Closing edge gaps

- Check that maximum value of gradient value is sufficiently large
  - drop-outs? use **hysteresis**
    - use a high threshold to start edge curves and a low threshold to continue them.

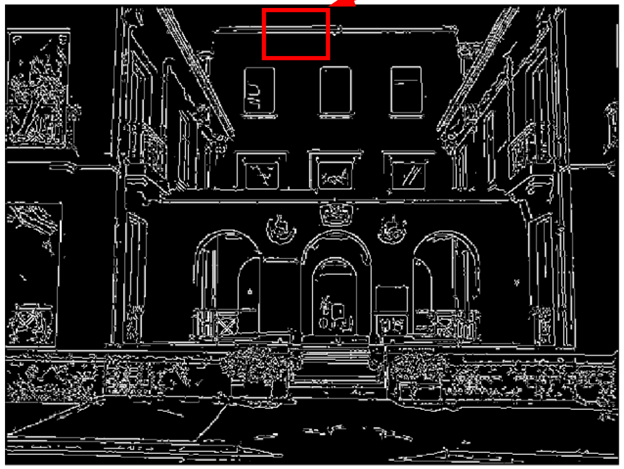


Example: Canny Edge Detection

Original image

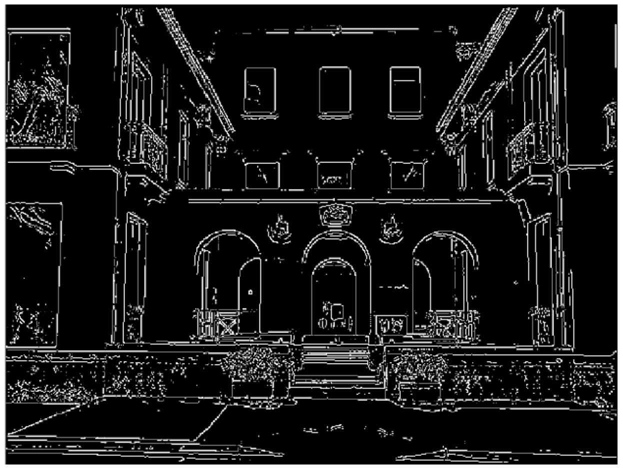


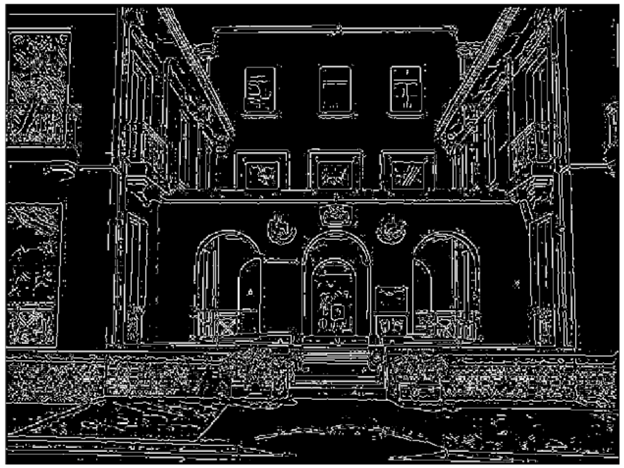
gap is gone



Strong + connected weak edges

Strong edges only





Weak edges

courtesy of G. Loy

Canny edge detector

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

- MATLAB: edge(image, ‘canny’)

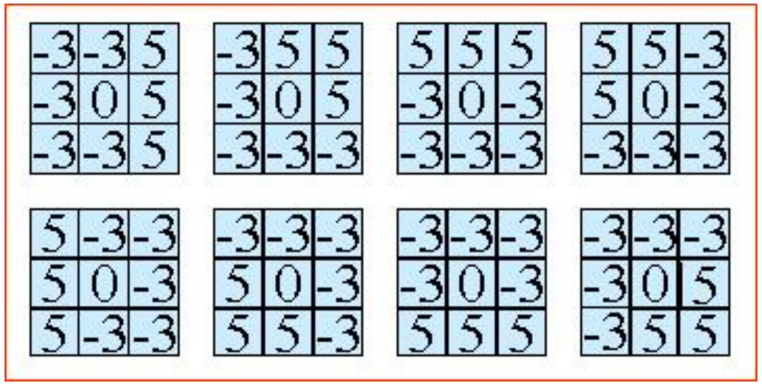
Source: D. Lowe, L. Fei-Fei

Template Matching

- An alternative method for edge detection computes the closest (over all four/eight directions) approximations of *g(i,j)* in every 3x3 neighborhood, to keep the one with maximum convolution value, provided it is large enough

Template Matching

- Kirsh’s operator



-3	-3	5
-3	0	5
-3	-3	5

-3	5	5
-3	0	5
-3	-3	-3

5	5	5
-3	0	-3
-3	-3	-3

5	5	-3
5	0	-3
-3	-3	-3

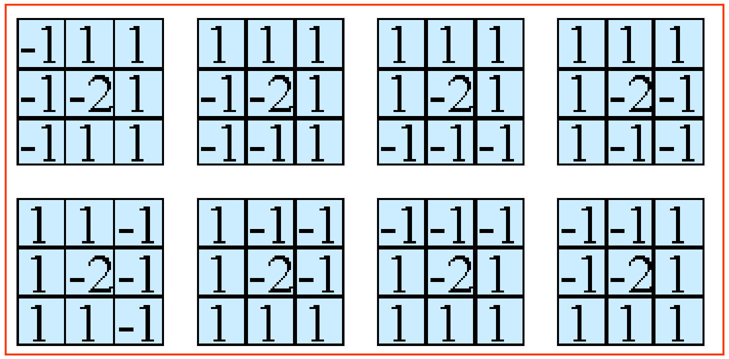
5	-3	-3
5	0	-3
5	-3	-3

-3	-3	-3
5	0	-3
5	5	-3

-3	-3	-3
-3	0	-3
5	5	5

-3	-3	-3
-3	0	5
-3	5	5

- Compass operator



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

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

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

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

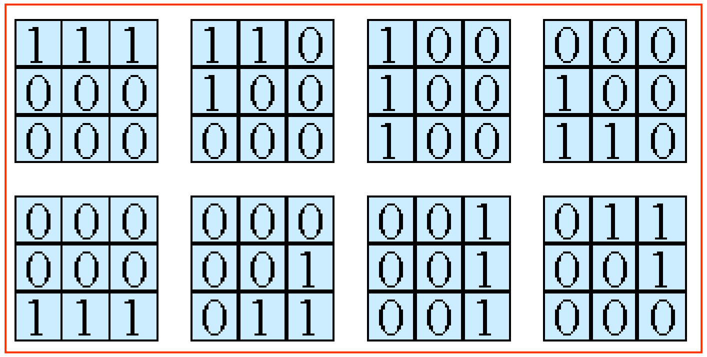
1	1	-1
1	-2	-1
1	1	-1

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

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

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

3/9 operator



1	1	1
0	0	0
0	0	0

1	1	0
1	0	0
0	0	0

1	0	0
1	0	0
1	0	0

0	0	0
1	0	0
1	1	0

0	0	0
0	0	0
1	1	1

0	0	0
0	0	1
0	1	1

0	0	1
0	0	1
0	0	1

0	1	1
0	0	1
0	0	0

$$P_i = \sum_{j=0}^{8} I_{i,j}$$

3	2	1
4	8	0
5	6	7

$P_{i,j} = I_{i,j} + I_{i,j-1} + I_{i,j+1}$, with $(j=1,8)_{\text{mod } 8}$

Contour extraction

$$P = 1.5\left[\frac{P_{i,k}}{P_i} - 0.333\right]$$

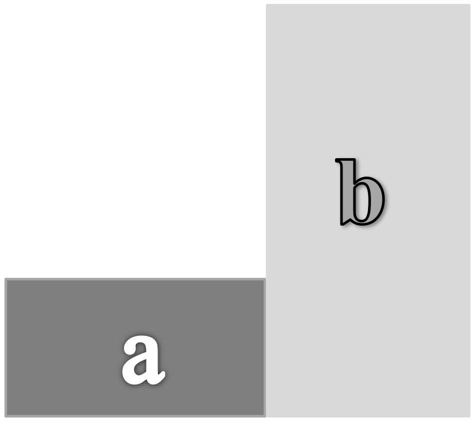
- $P_{i,k}$ is the maximum among the 8 parameters $P_{i,j}$
- The coefficients 3/2 et 1/3 are introduced to normalize the result so that monochromatic area has P=0
- The final threshold can be applied depending on the minimum average contrast τ admitted in the neighborhood

Practical aspects of the 3/9 filter

- The filter implements a relative gray level intensity analisys. Also the human eye apply a similir approach.
- It must be payed attention when looking contours in the dark!
- Note that if P_i is low this edge estimation suffers very much for the effect of the noise (if the intensity in the area is 0 then $P=0/0$).
- Selecting the threshold for P_i note that it is 9 time di average intensity of the area (if the average area intensity is 10 over 255, that is very low, then $P_i = 90$, and edges are looked for in the very dark)

Contrast and threshold

- Let us call **contrast** the ratio $\tau = \frac{a}{b}$, the threshold Th is given by:

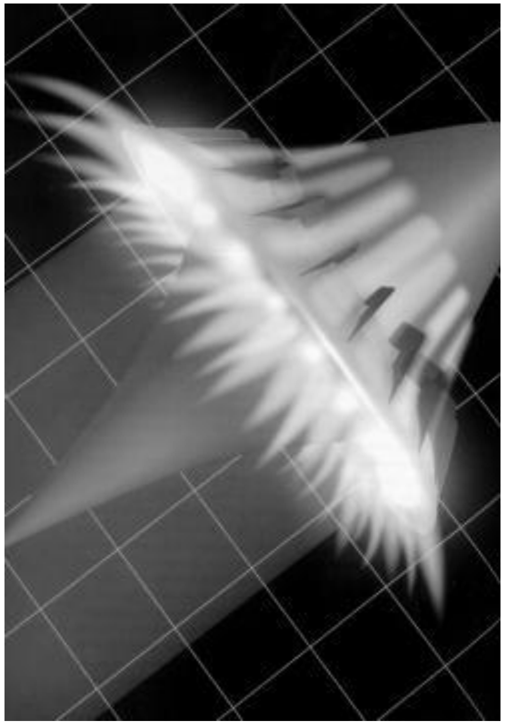


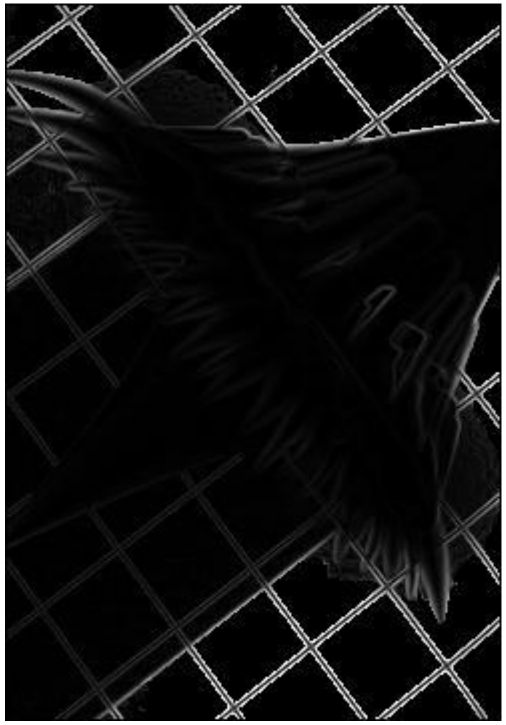
$$P_i = \frac{3}{2}\left[\frac{3b}{6a+3b} - \frac{1}{3}\right]$$

$$P_i = \frac{3}{2}\left[\frac{1}{2\tau+1} - \frac{1}{3}\right]$$

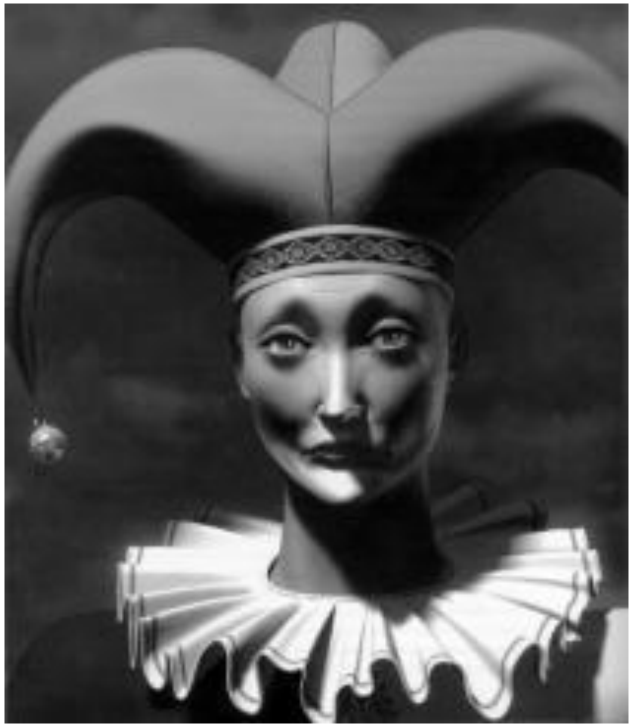
$$Th = \frac{1-\tau}{2\tau+1}$$

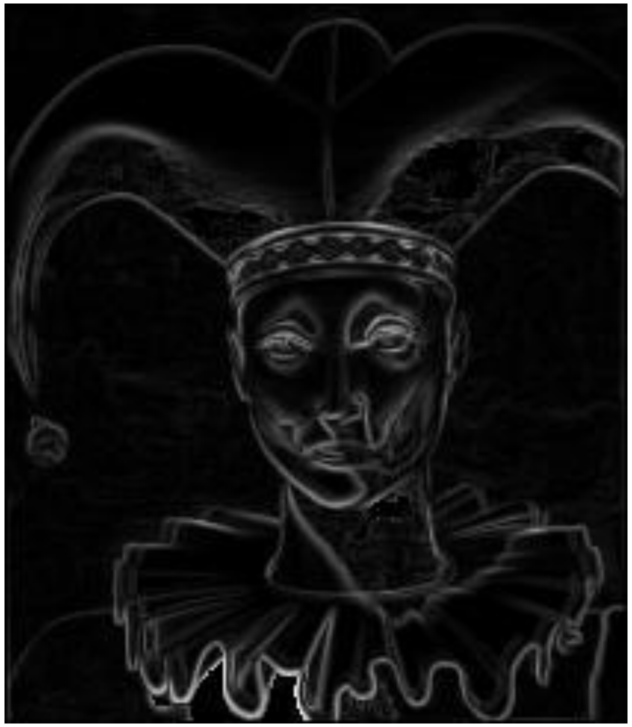
Example: Op. 3 / 9





Example: Op. 3 / 9

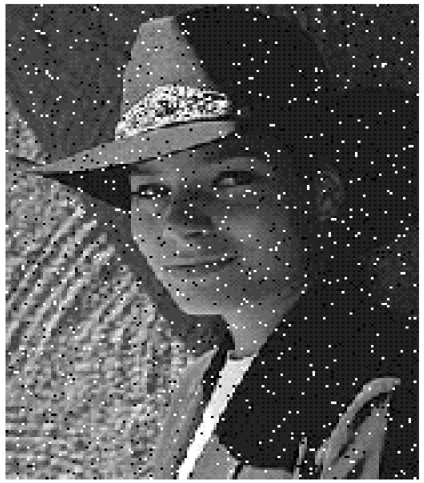


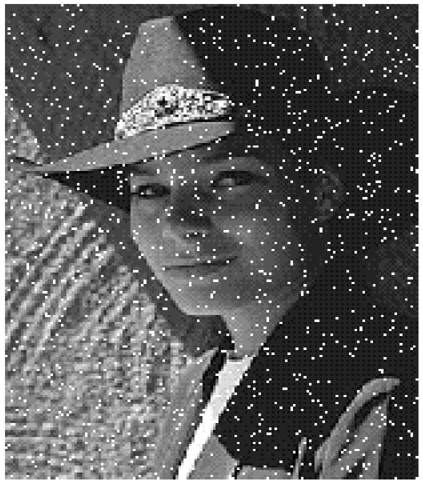


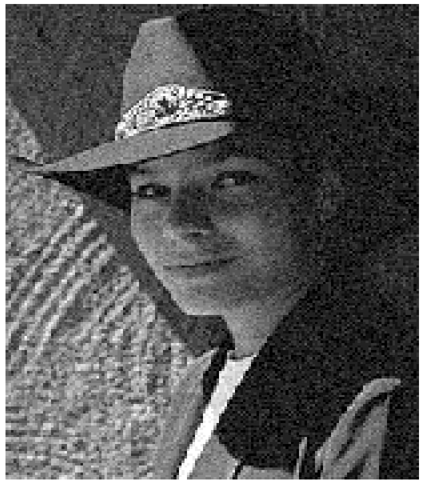
Common types of noise

- **Salt and pepper noise**: random occurrences of black and white pixels
- **Impulse noise:** random occurrences of white pixels
- **Uniform noise**: constant probability density in a given range ±k
- **Gaussian noise**: variations in intensity drawn from a Gaussian normal distribution

Original


Salt and pepper noise

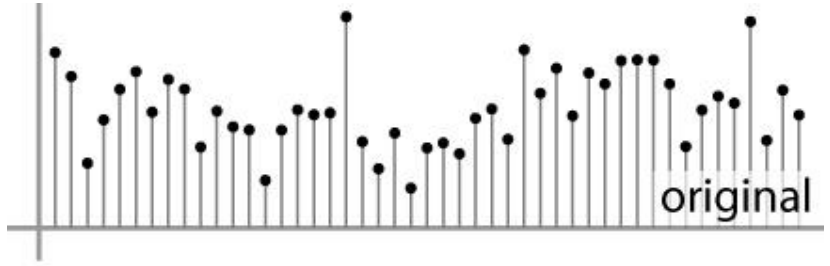

Impulse noise

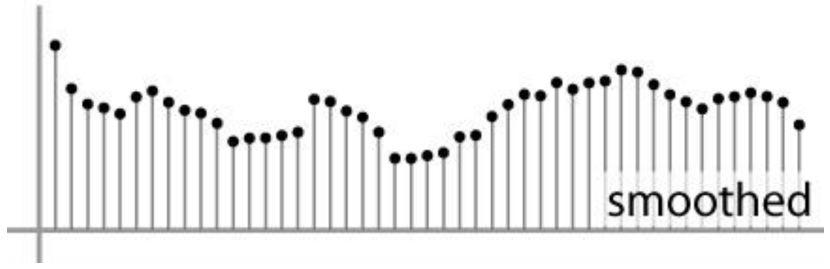

Gaussian noise

Source: S. Seitz

First attempt at a solution

- Let's replace each pixel with an average of all the values in its neighborhood
- Moving average in 1D:

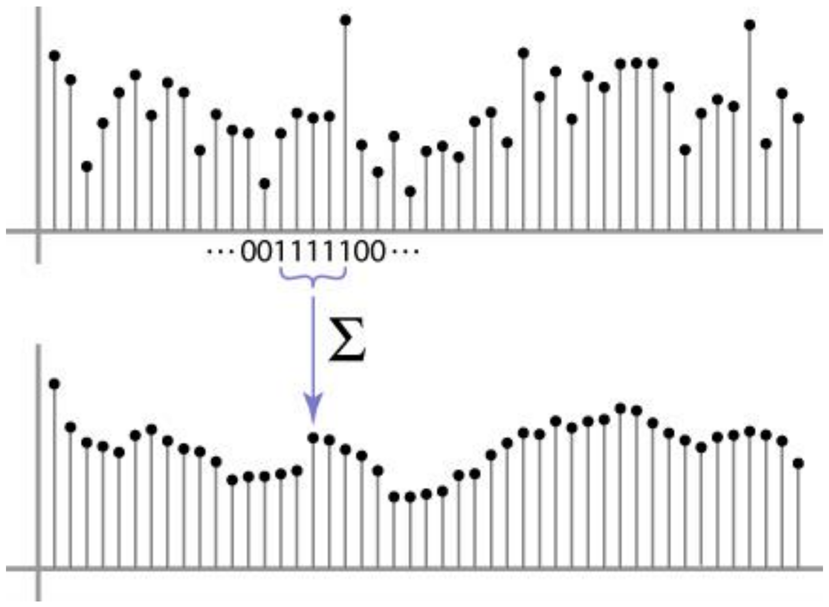




Source: S. Marschner

Weighted Moving Average

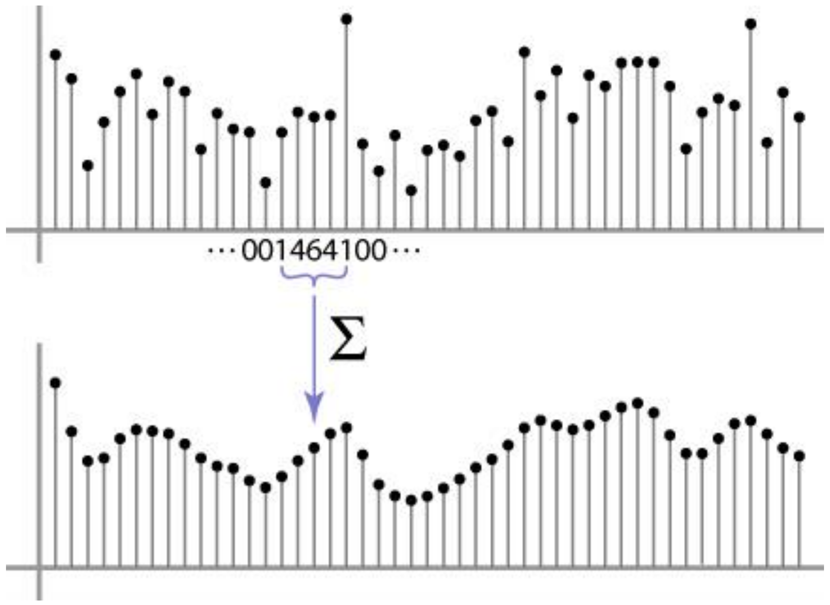
- Can add weights to our moving average
- *Weights* [1, 1, 1, 1, 1] / 5



Source: S. Marschner

Weighted Moving Average

- Non-uniform weights [1, 4, 6, 4, 1] / 16



Source: S. Marschner

Degraded image: uniform noise

- The standard model of this noise is additive, independent at each pixel and independent of the signal intensity with continuous uniform distribution in a given interval**.** The noise caused by quantizing the pixels to discrete levels has an approximately uniform distribution.

This noise can be simulated adding in each pixel $n=2k(\textit{rnd} - 0{,}5)$ being k the noise max intensity and *rnd* a random number with $0 \le \text{rnd} \le 1$

Degraded image: ‘salt and pepper’

- This is an impulsive or spike noise for which the image has dark pixels and bright pixels randomly distributed.

This noise can be simulated for each pixel in this way:

if $rnd \geq Th_1$ I(i,j) = 255

if $rnd \leq Th_2$ I(i,j) = 0

else $n=2[(K- Th_2)/(Th_1- Th_2)](rnd\text{-}0,5)$ and if I(i,j)+n>255: I(i,j)=255, if I(i,j)+n<0:I(i,j)=0

being K the uniform component noise intensity, $0 \leq rnd \leq 1$, and Th_1 and Th_2 two suitable thresholds (1-Th_1 and Th_2 are the percentage of extra white and black pixels respectively)

Average value filter

- Each pixel takes the average value over the neighbors (3x3 in the example)

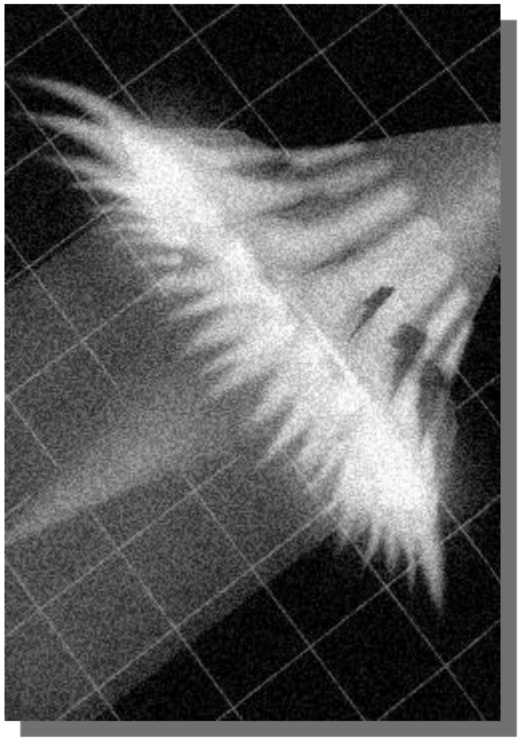
- Example - given the neighborhood:

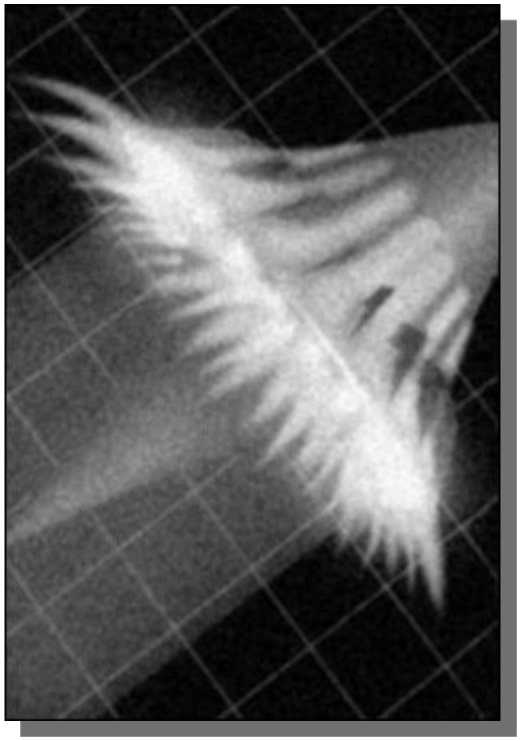
3	6	8
3	4	2
5	8	3

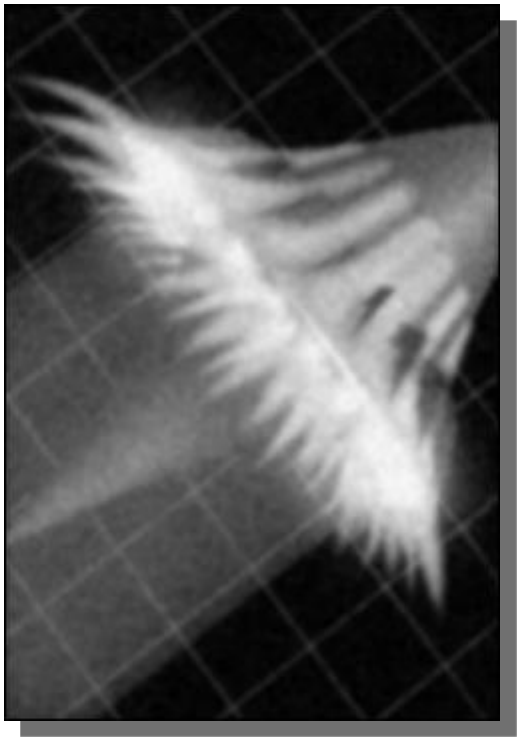
the central pixel will take the new value:

(3+6+8+3+4+2+5+8+3)/9 = 4.67

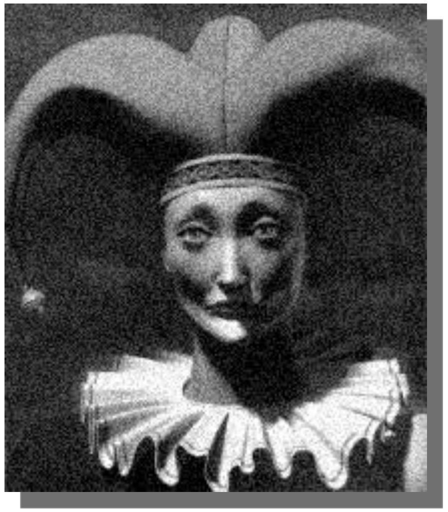
Average value filter: uniform noise

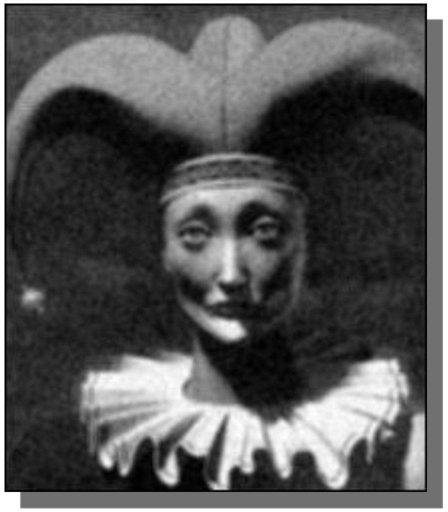

Noisy image

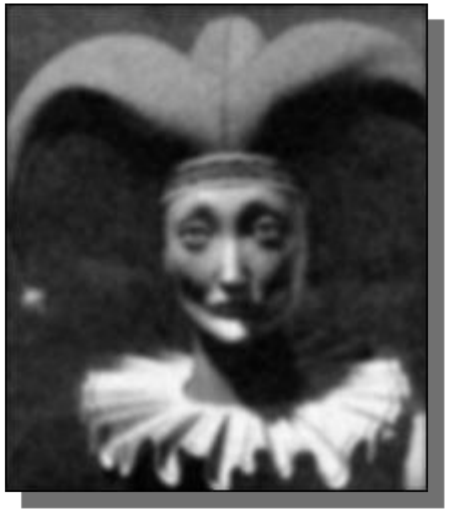

Filtered image


Second iteration

Average value filter: uniform noise





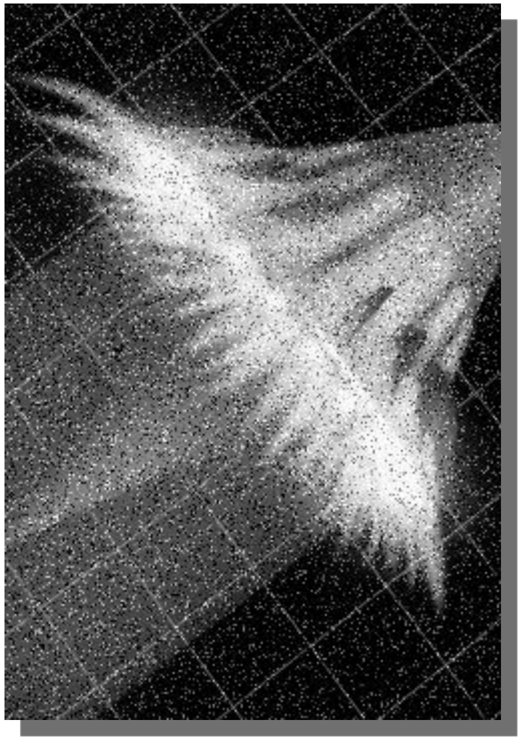


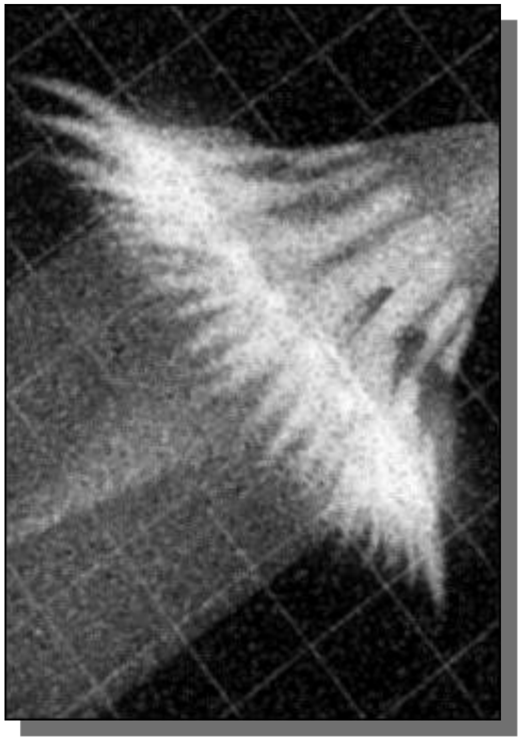
Noisy image

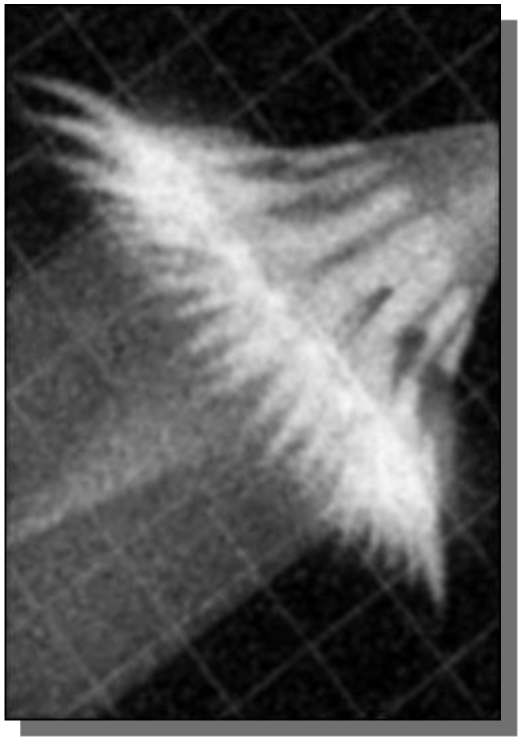
Filtered image

Second iteration

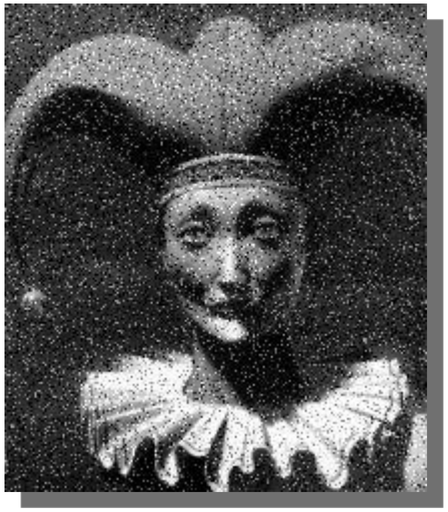
Average value filter: salt and pepper


Noisy image

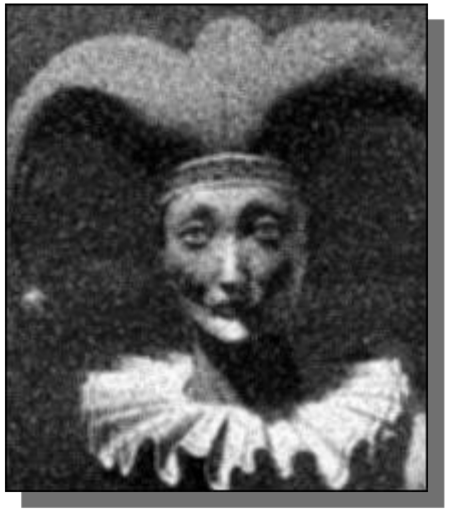

Filtered image


Second iteration

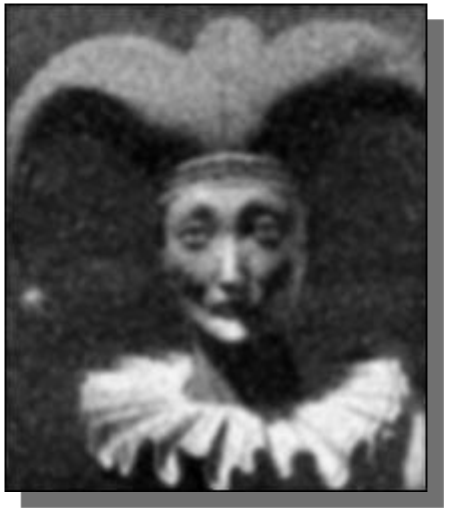
Average value filter: salt and pepper



Noisy image



Filtered image



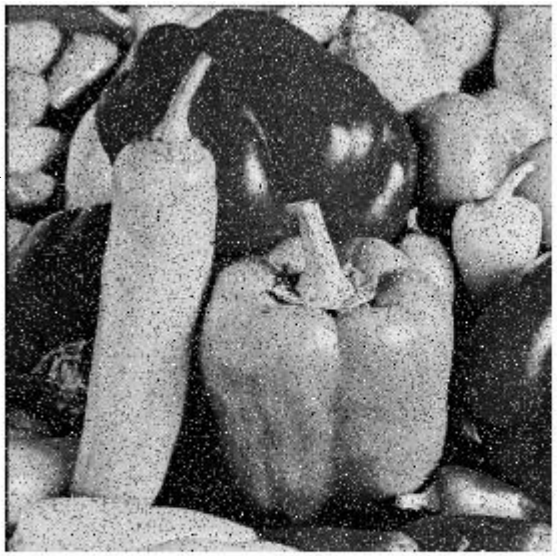
Second iteration

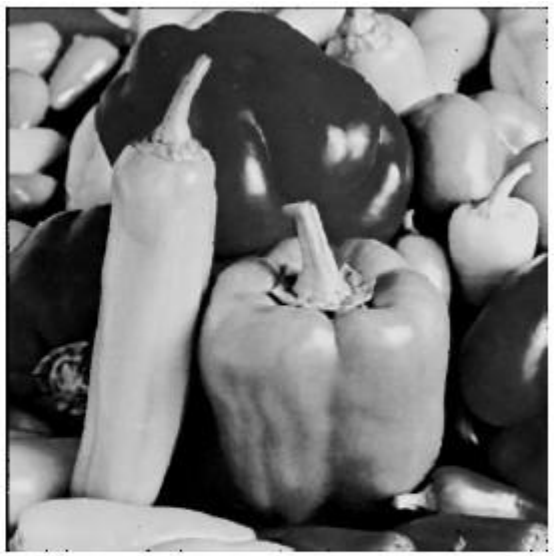
Median filters

- A **Median Filter** operates over a window by selecting the median intensity in the window.
- What advantage does a median filter have over a mean filter?
- Is a median filter a kind of convolution?

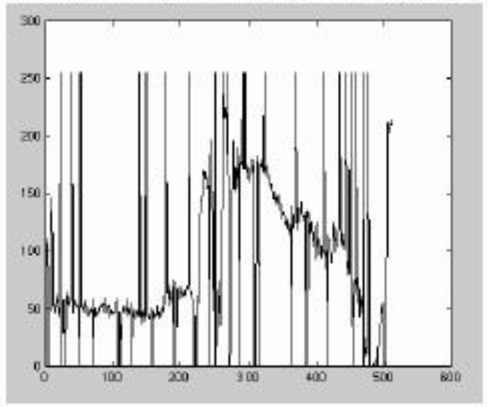
Median filter

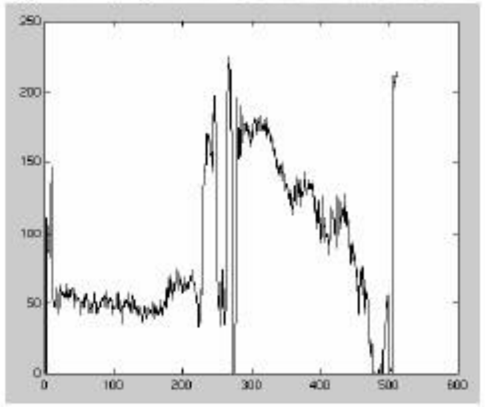
Salt and pepper noise





Median filtered





Plots of a row of the image

Source: M. Hebert

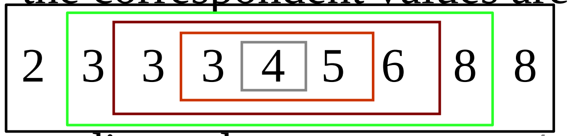
Median and rank filters

- The median filter assigns to pixel the median value of neighborhood
- It is a particular case of the *rank filters* family, in which to the pixel is assigned the average value over a predefined range of the neighbors histogram.
- The average excluding the extremes is suited for impulse or spike noise such as the salt and pepper case.

- Example - given the neighborhood:

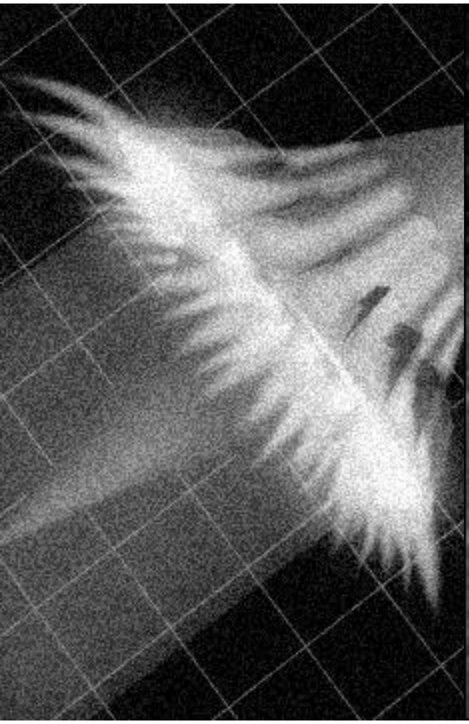
3	6	8
3	4	2
5	8	3

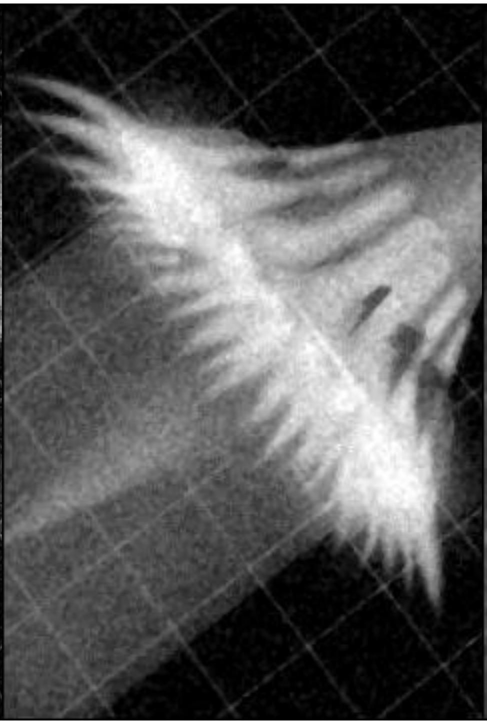
the correspondent values are:

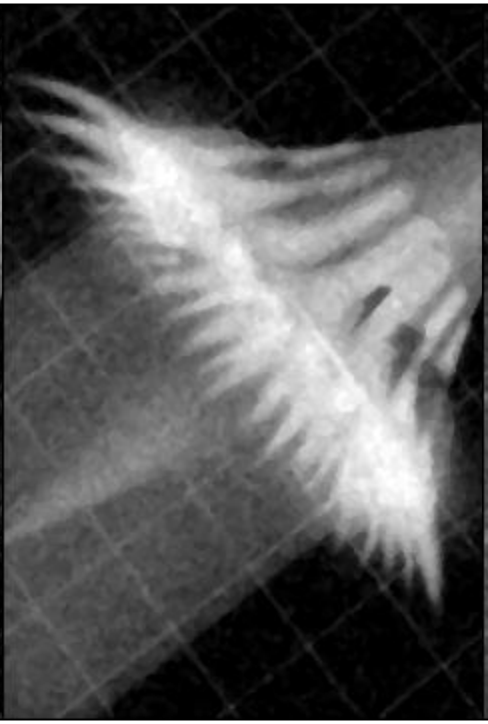


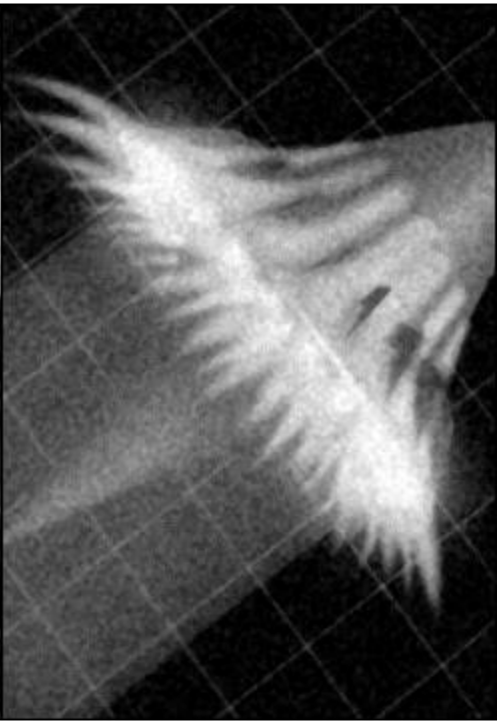
median value:	4;
over three values:	4;
over five values:	4,2;
over seven values:	4,57
over nine values:	4,66

Median filter: uniform noise

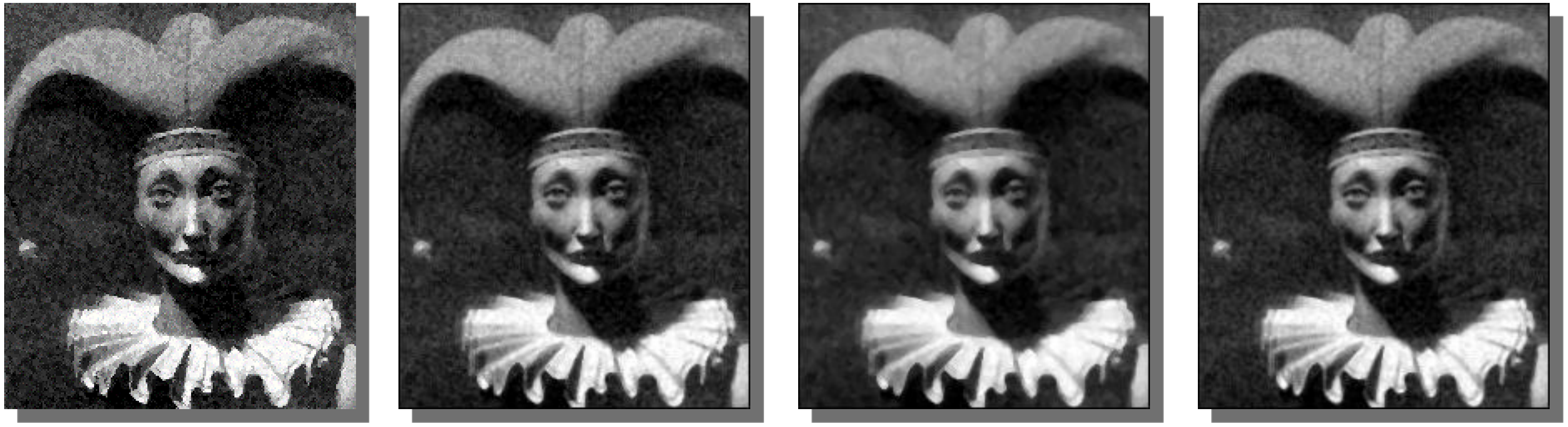

Noisy image


Filtered image


Second iteration

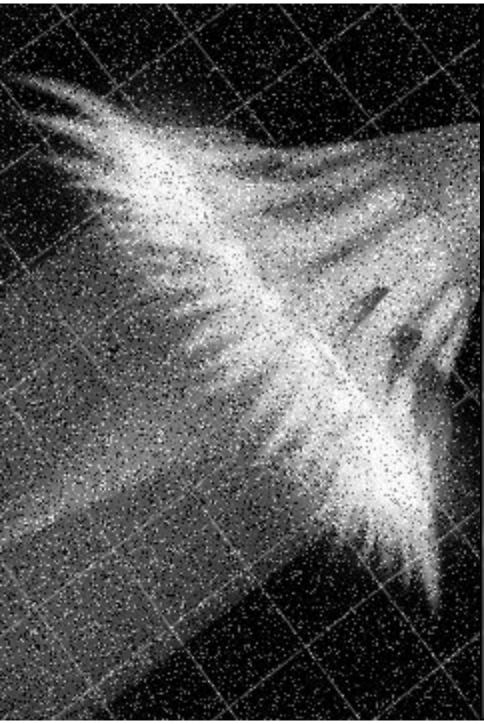

Rank 3

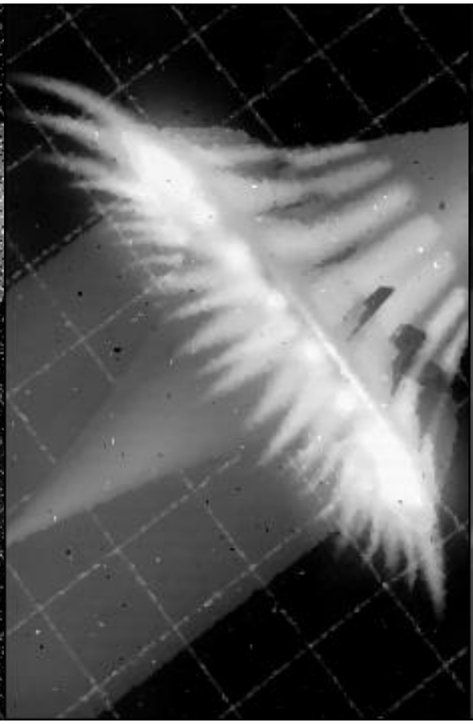
Median filter: uniform noise

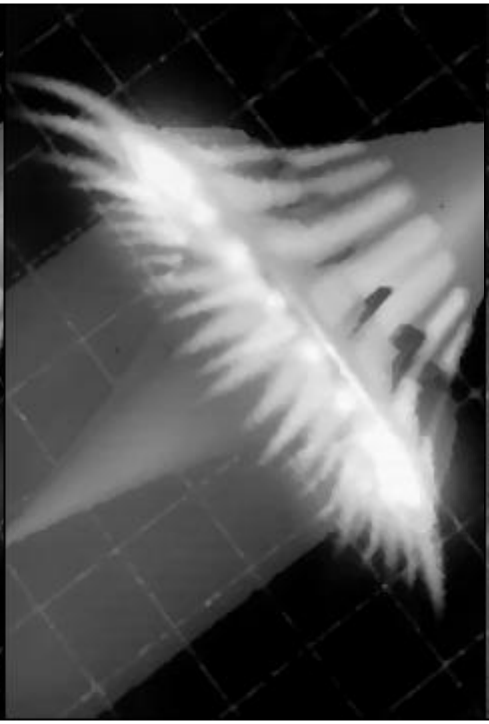


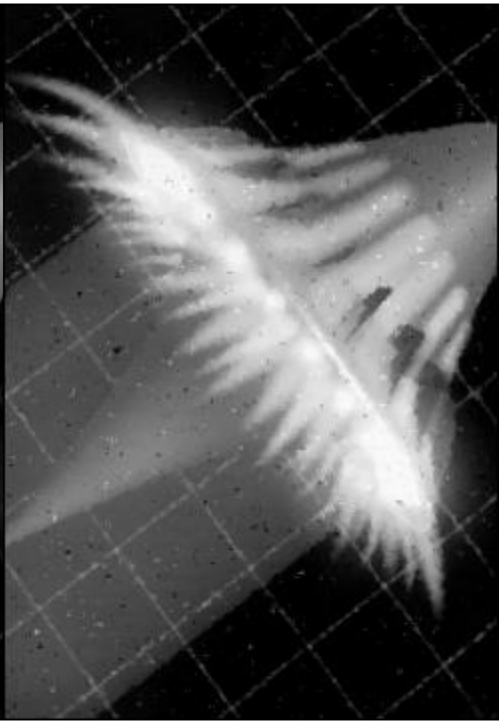
Noisy image Filtered image Second iteration Rank 3

Median filter: salt and pepper

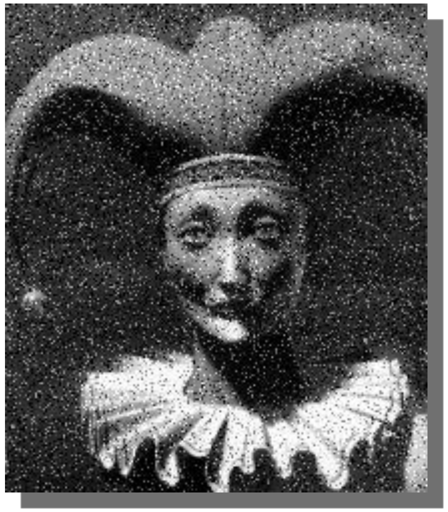

Noisy image

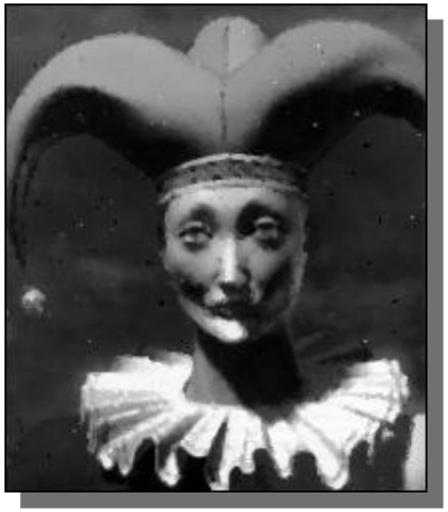

Filtered image

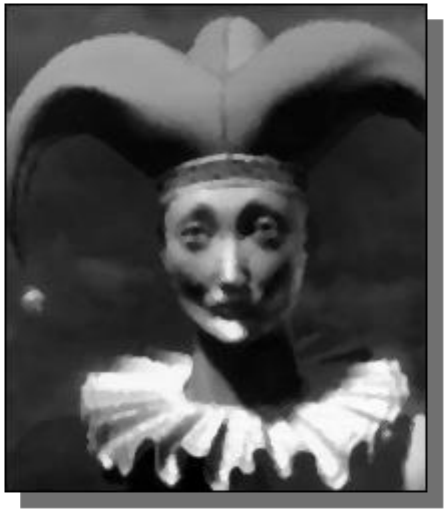

Second iteration

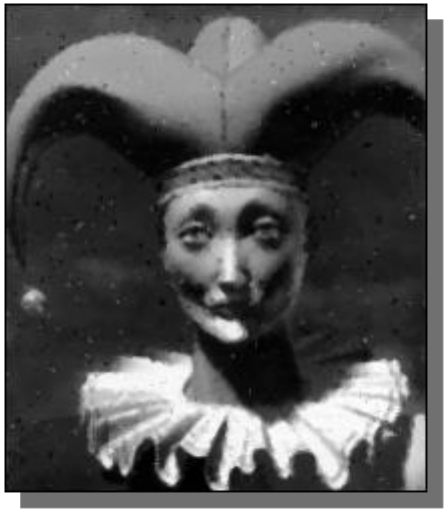

Rank 3

Median filter: salt and pepper

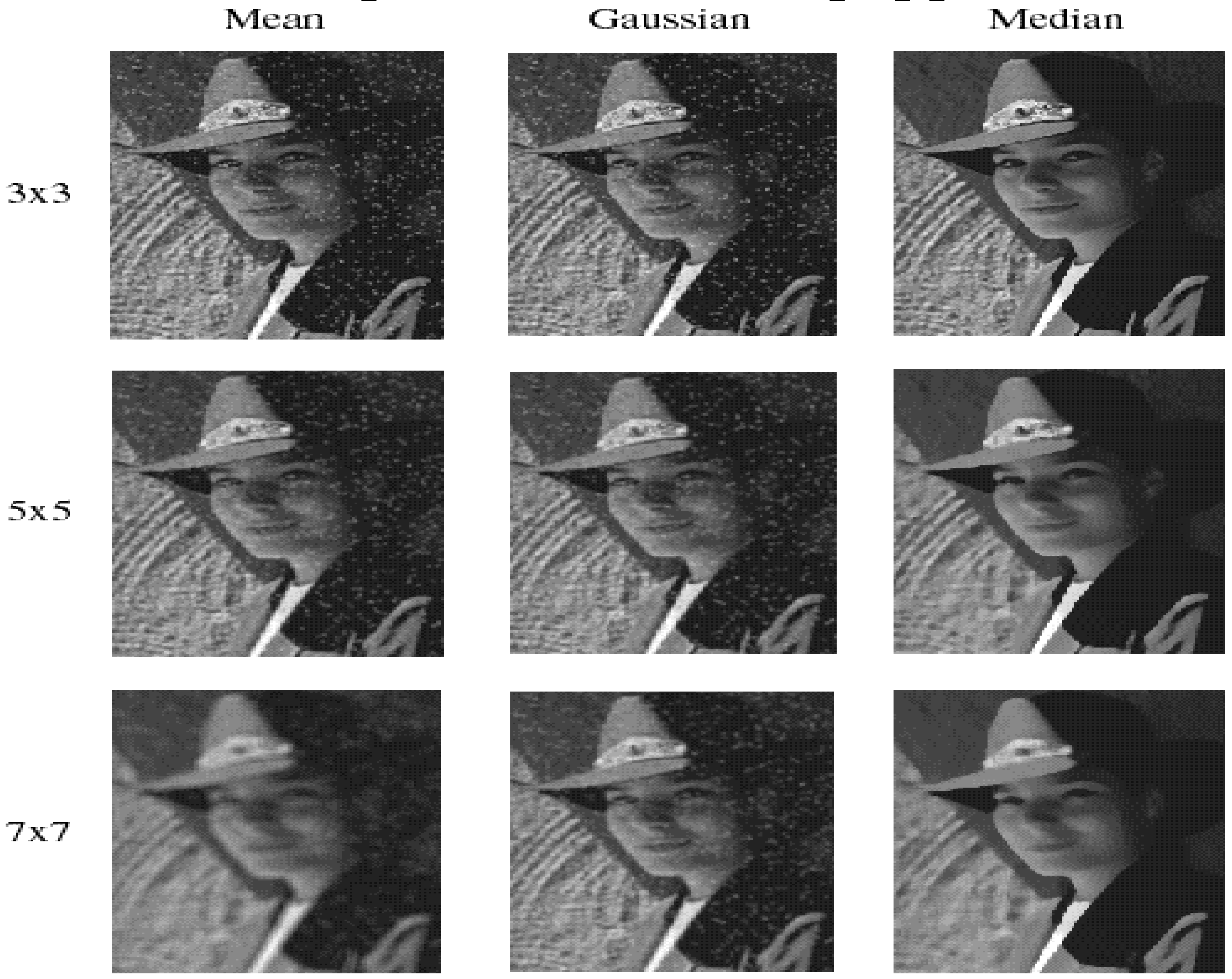

Noisy image


Filtered image


Second iteration


Rank 3

Comparison: salt and pepper noise



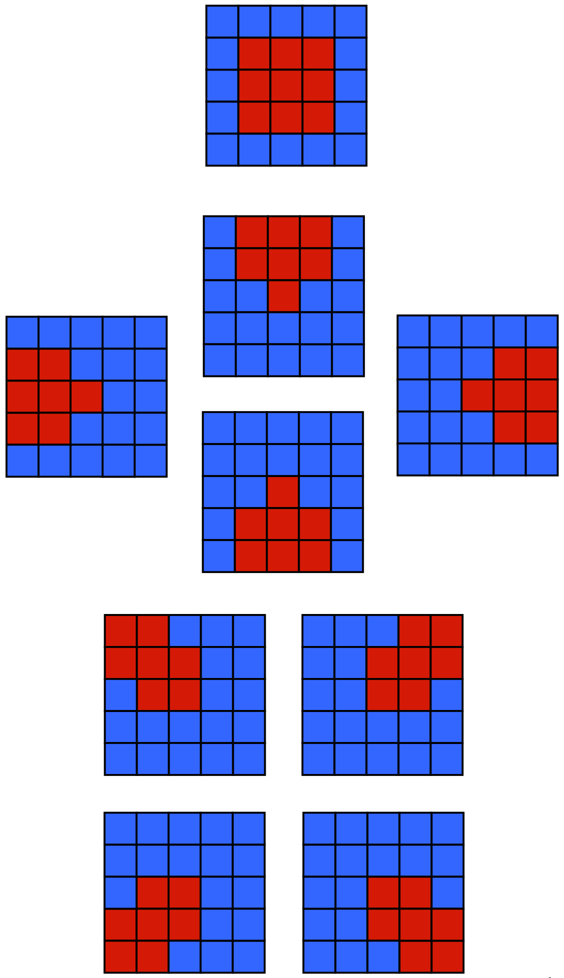
Slide by Steve Seitz

The Nagao-Matsuyama Filter

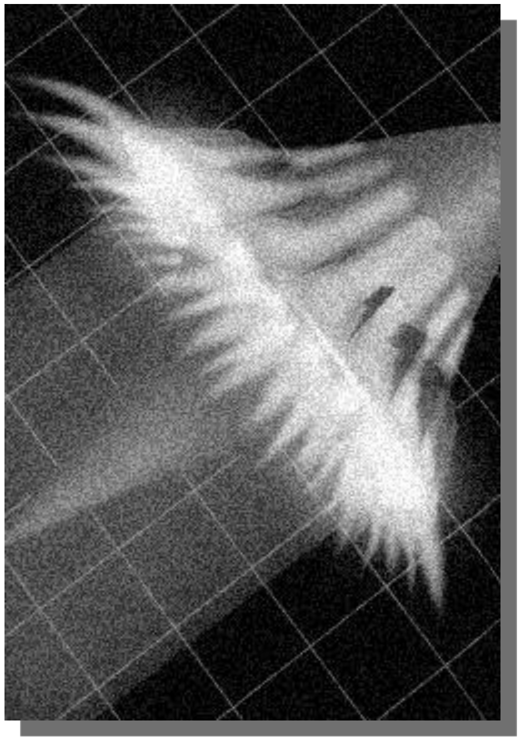
This filter selects for the centre pixel the average for the orientation with the least variation. Hence, the steps are as follows:

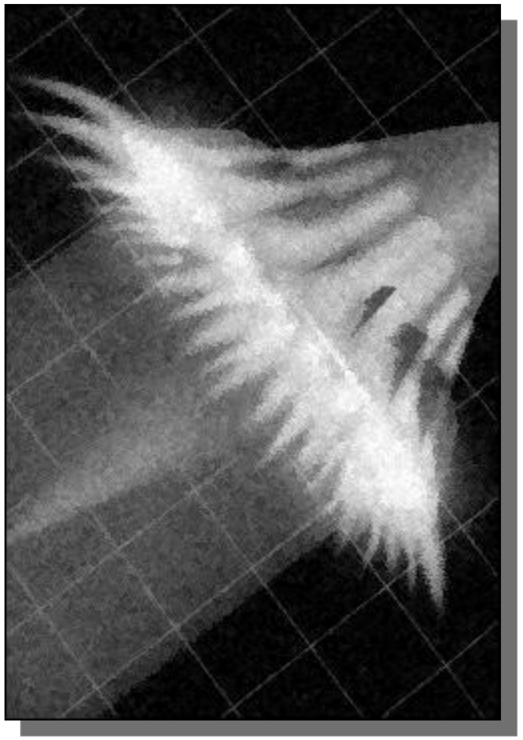
1. Calculate the variance for each of the nine sub-groups shown to the right (including the centre pixel).
2. Determine the sub-group with the lowest variance.
3. Assign the mean of this sub-group to the centre pixel.

Nagao-Matsuyama improves the borders, and is effective at reducing the edges smoothing. Clearly there is a cost in terms of computation due to the calculation of nine variances for each pixel.

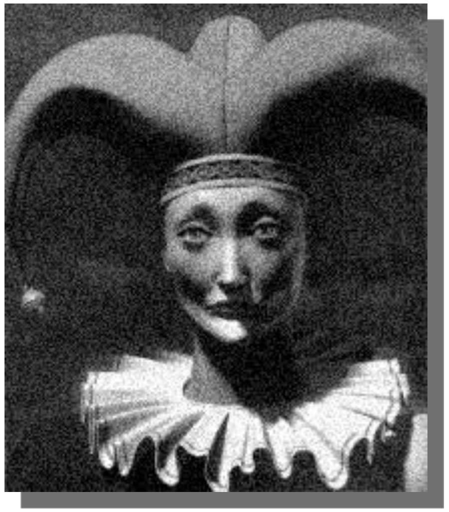


Nagao filter: uniform noise

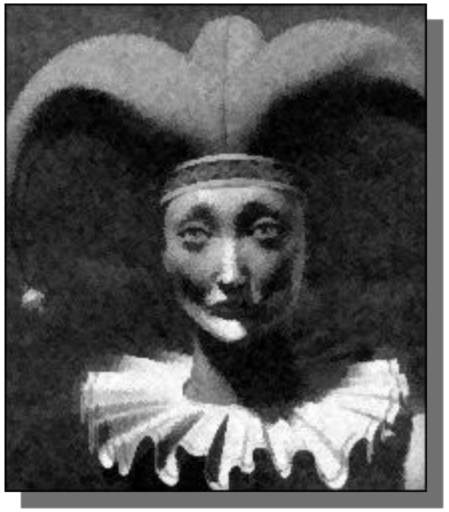

Noisy image


Filtered image

Nagao filter: uniform noise

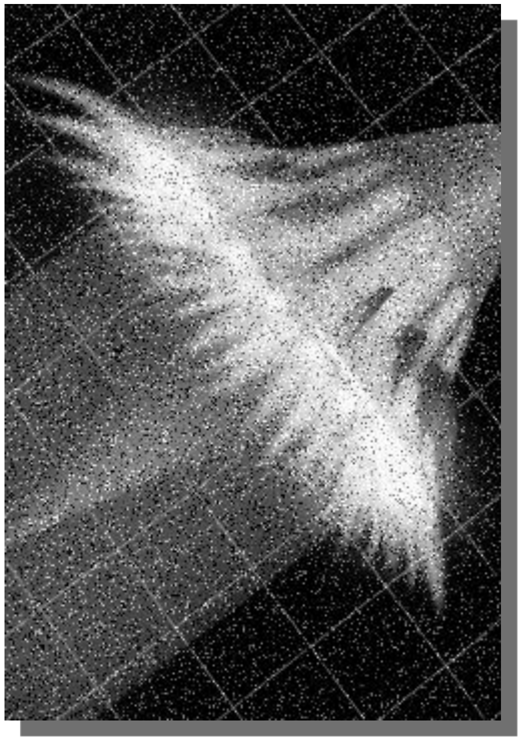


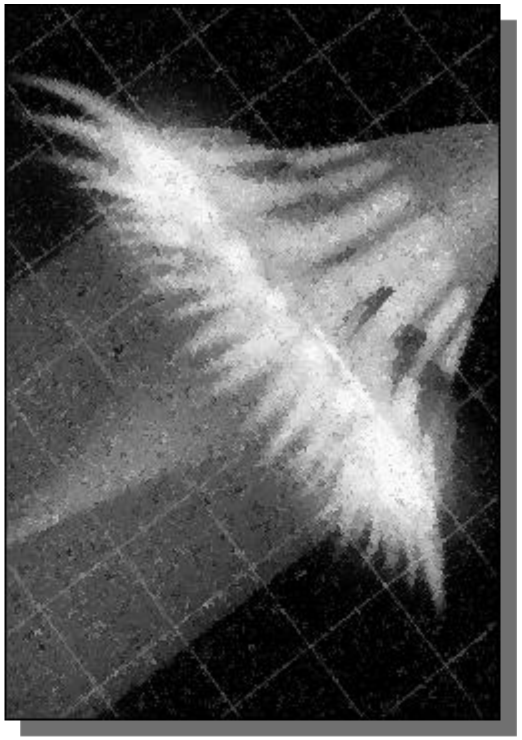
Noisy image



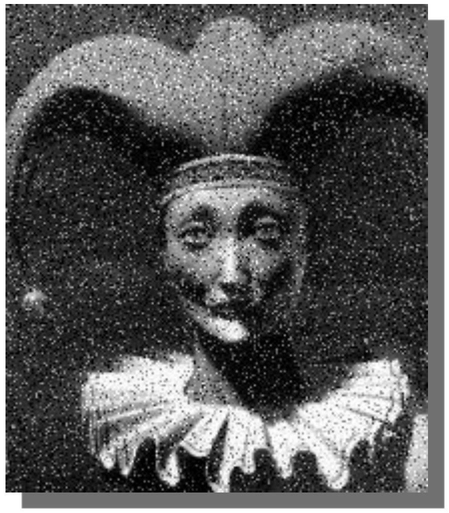
Filtered image

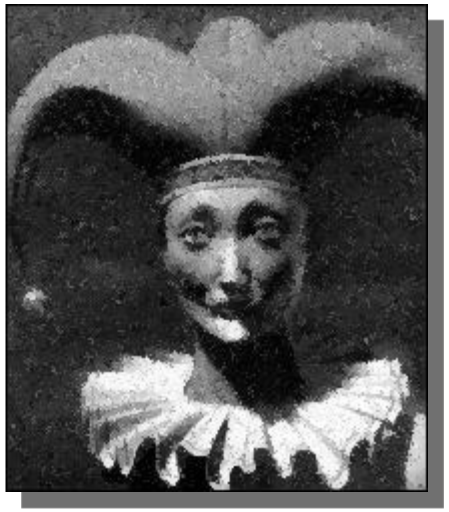
Nagao filter: salt and pepper


Noisy image


Filtered image

Nagao filter: salt and pepper


Noisy image


Filtered image