Computer Vision I - Algorithms and Applications: *Basics of Image Processing*

Carsten Rother

28/10/2013

Link to lectures

- Slides of Lectures and Exercises will be online:

  http://www.inf.tu-Dresden/index.php?node_id=2091&ln=en

  (on our webpage > teaching > Computer Vision)

Roadmap: Basics Digital Image Processing

- Images
- Point operators (ch. 3.1)
- Filtering: (ch. 3.2, ch 3.3, ch. 3.4) – main focus
  - Linear filtering
  - Non-linear filtering
- Fourier Transformation (ch. 3.4)
- Multi-scale image representation (ch. 3.5)
- Edges (ch. 4.2)
  - Edge detection and linking
- Lines (ch. 4.3)
  - Line detection and vanishing point detection

Roadmap: Basics Digital Image Processing

- Images
- Point operators (ch. 3.1)
- Filtering: (ch. 3.2, ch 3.3, ch. 3.4) – main focus
  - Linear filtering
  - Non-linear filtering
- Fourier Transformation (ch. 3.4)
- Multi-scale image representation (ch. 3.5)
- Edges (ch. 4.2)
  - Edge detection and linking
- Lines (ch. 4.3)
  - Line detection and vanishing point detection

What is an Image

- We can think of the image as a function:

$$I(x, y), \qquad I: \ \mathbb{R} \times \mathbb{R} \to \mathbb{R}$$

  - For every 2D point (pixel) it tells us the amount of light it receives
  - The size and range of the sensor is limited:

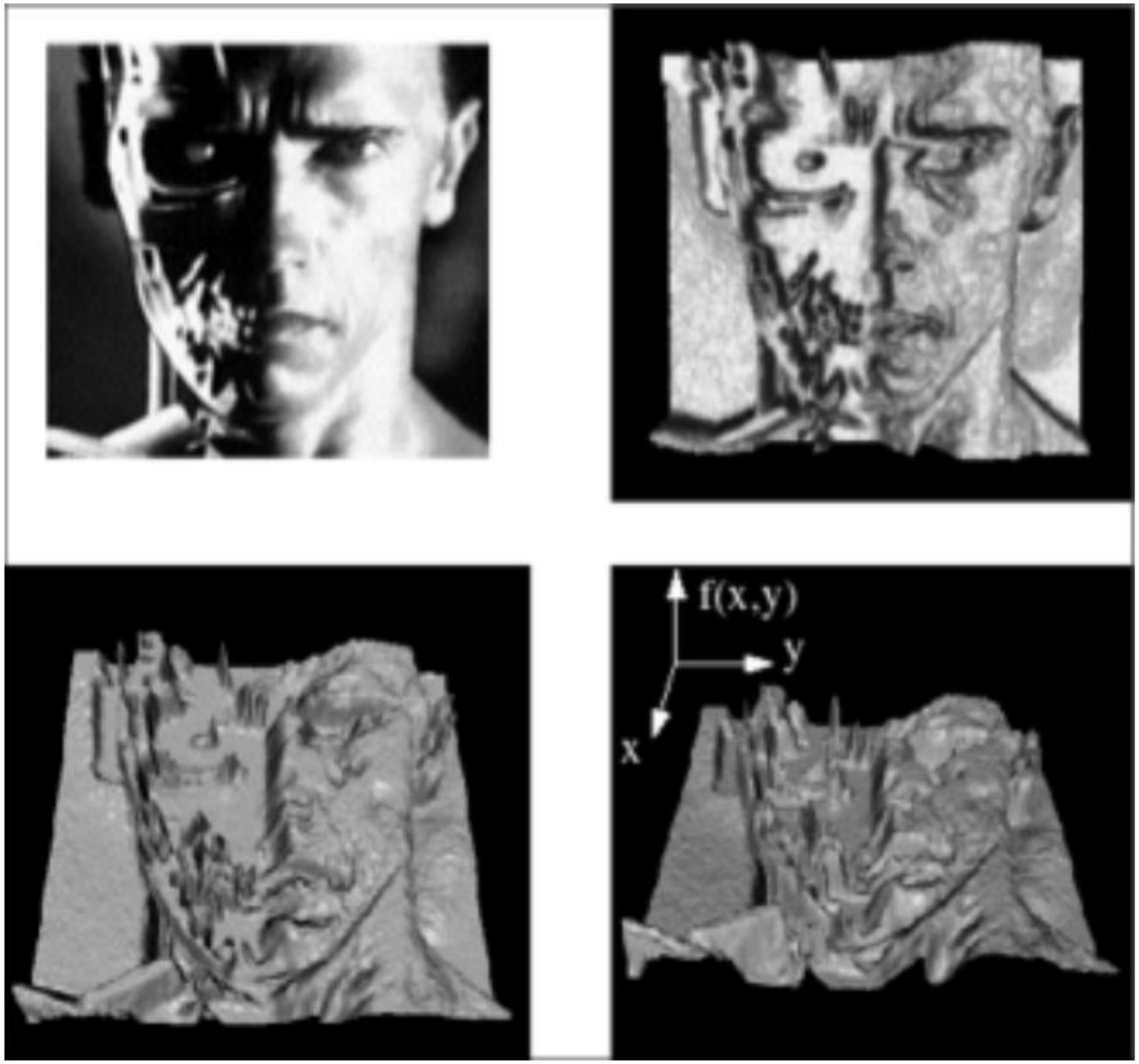
$$I(x, y), \qquad I: [a, b] \times [c, d] \to [0, m]$$

  - Colour image is then a vector-valued function:

$$I(x, y) = \begin{pmatrix} I_R(x, y) \\ I_G(x, y) \\ I_B(x, y) \end{pmatrix}, \qquad I: [a, b] \times [c, d] \to [0, m]^3$$

  - Comment, in most lectures we deal with grey-valued images and extension to colour is "obvious"

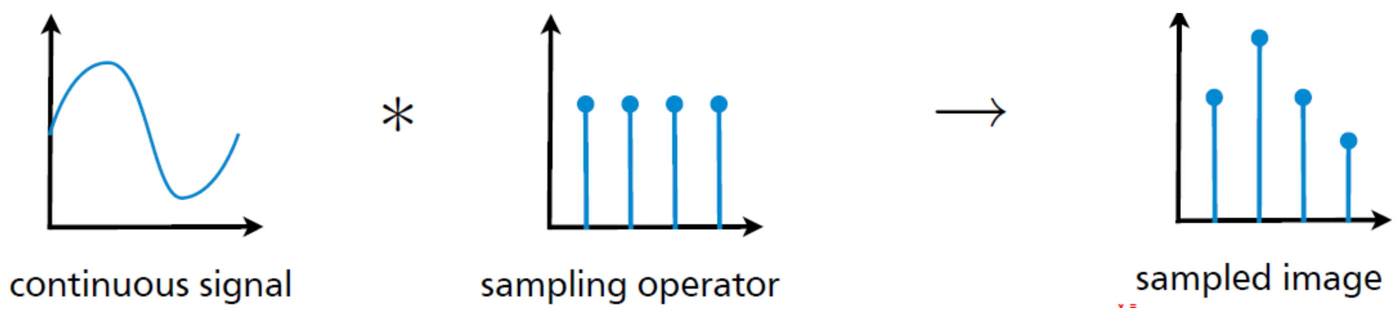
Images as functions



[from Steve Seitz]

Digital Images

- We usually do not work with spatially continuous functions, since our cameras do not sense in this way.
- Instead we use (spatially) discrete images
- Sample the 2D domain on a regular grid (1D version)

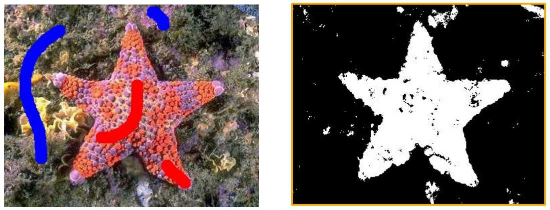


- Intensity/color values usually also discrete. Quantize the values per channel (e.g. 8 bit per channel)

y = \ x =	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72
41	210	209	204	202	197	247	143	71	64	80	84	54	54	57	58
42	206	196	203	197	195	210	207	56	63	58	53	53	61	62	51
43	201	207	192	201	198	213	156	69	65	57	55	52	53	60	50
44	216	206	211	193	202	207	208	57	69	60	55	77	49	62	61
45	221	206	211	194	196	197	220	56	63	60	55	46	97	58	106
46	209	214	224	199	194	193	204	173	64	60	59	51	62	56	48
47	204	212	213	208	191	190	191	214	60	62	66	76	51	49	55
48	214	215	215	207	208	180	172	188	69	72	55	49	56	52	56
49	209	205	214	205	204	196	187	196	86	62	66	87	57	60	48
50	208	209	205	203	202	186	174	185	149	71	63	55	55	45	56
51	207	210	211	199	217	194	183	177	209	90	62	64	52	93	52
52	208	205	209	209	197	194	183	187	187	239	58	68	61	51	56
53	204	206	203	209	195	203	188	185	183	221	75	61	58	60	60
54	200	203	199	236	188	197	183	190	183	196	122	63	58	64	66
55	205	210	202	203	199	197	196	181	173	186	105	62	57	64	63

Comment on Continuous Domain / Range

- There is a branch of computer vision research ("variational methods"), which operates on continuous domain for input images and output results
- Continuous domain methods are typically used for physics-based vision: segmentation, optical flow, etc. (we may consider this briefly in later lectures)



- Continues domain methods then use different optimization techniques, but still discretize in the end.
- In this lecture and other lectures we mainly operate in discrete domain and discrete or continuous range for output results

Roadmap: Basics Digital Image Processing

- Images
- Point operators (ch. 3.1)
- Filtering: (ch. 3.2, ch 3.3, ch. 3.4) – main focus
  - Linear filtering
  - Non-linear filtering
- Fourier Transformation (ch. 3.4)
- Multi-scale image representation (ch. 3.5)
- Edges (ch. 4.2)
  - Edge detection and linking
- Lines (ch. 4.3)
  - Line detection and vanishing point detection

Point operators

- Point operators work on every pixel independently:

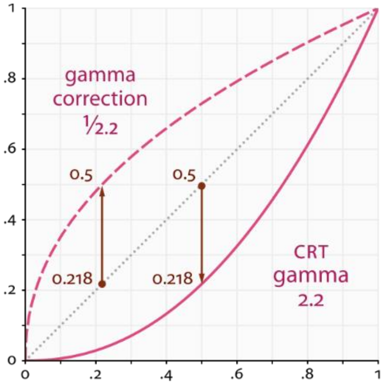
$$J(x,y) = h\big(I(x,y)\big)$$

- Examples for h:
    - Control contrast and brightness; $h(z) = az + b$

original

Contrast enhanced

Example for Point operators: Gamma correction



Intensity range: [0,1]

Inside cameras:
$h(z) = z^{1/\gamma}$ where
often $\gamma = 2.2$ (called gamma correction)

In (old) CRT monitors
An intensity z was perceived as:
$h(z) = z^{\gamma}$ ($\gamma = 2.2$ typically)

Today: even with "linear mapping" monitors, it is good to keep the gamma corrected image. Since human vision is more sensitive in dark areas.

Important: for many tasks in vision, e.g. estimation of a normal, it is good to run $h(z) = z^{\gamma}$ to get to a linear function

Example for Point Operators: Alpha Matting

Foreground F

Background B

Matte α
(amount of transparency)

Composite C

$$C(x,y) = \alpha(x,y)F(x,y) + \big(1 - \alpha(x,y)\big)B(x,y)$$

Roadmap: Basics Digital Image Processing

- Images
- Point operators (ch. 3.1)
- Filtering: (ch. 3.2, ch 3.3, ch. 3.4) – main focus
  - Linear filtering
  - Non-linear filtering
- Fourier Transformation (ch. 3.4)
- Multi-scale image representation (ch. 3.5)
- Edges (ch. 4.2)
  - Edge detection and linking
- Lines (ch 4.3)
  - Line detection and vanishing point detection

Linear Filters / Operators

- Properties:
  - Homogeneity: $T[aX] = aT[X]$
  - Additivity: $T[X + Y] = T[X] + T[Y]$
  - Superposition: $T[aX + bY] = aT[X] + bT[Y]$

- Example:
  - Convolution
  - Matrix-Vector operations

Convolution

- Replace each pixel by a linear combination of its neighbours and itself.
- 2D convolution (discrete)

$$g = f * h$$

Centred at 0,0

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

image
$f(x, y)$

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

filter (kernel)
$h(x, y)$

=

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

filtered image
$g(x, y)$

smaller output?

$$g(x,y) = \sum_{k,l} f(x-k, y-l)h(k,l) = \sum_{k,l} f(k,l)h(x-k, y-l)$$

Convolution

- Linear $h * (f_0 + f_1) = h * f_0 + h * f_1$
- Associative $(f * g) * h = f * (g * h)$
- Commutative $f * h = h * f$
- Shift-Invariant $g(x, y) = f(x + k, y + l) \leftrightarrow (h * g)(x, y) = (h * f)(x + k, y + l)$

  (behaves everywhere the same)

$$\begin{array}{|c|c|c|c|c|}\hline 72 & 88 & 62 & 52 & 37 \\ \hline\end{array} * \begin{array}{|c|c|c|}\hline 1/4 & 1/2 & 1/4 \\ \hline\end{array} \Leftrightarrow$$

$$\frac{1}{4}\begin{bmatrix} 2 & 1 & . & . & . \\ 1 & 2 & 1 & . & . \\ . & 1 & 2 & 1 & . \\ . & . & 1 & 2 & 1 \\ . & . & . & 1 & 2 \end{bmatrix} \begin{bmatrix} 72 \\ 88 \\ 62 \\ 52 \\ 37 \end{bmatrix}$$

- Can be written in Matrix form: g = H f
- Correlation (not mirrored filter):

$$g(x, y) = \sum_{k,l} f(x + k, y + l) h(k, l)$$

Examples

δ

y

- Impulse function: $f = f * \delta$

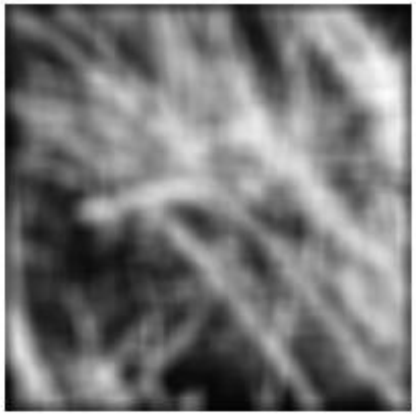
x

- Box Filter:

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

Original Image

Box-filtered image



Application: Noise removal

- Noise is what we are not interested in:
  sensor noise (Gaussian, shot noise), quantisation artefacts, light fluctuation, etc.

- Typical assumption is that the noise is not correlated between pixels

- Basic Idea:

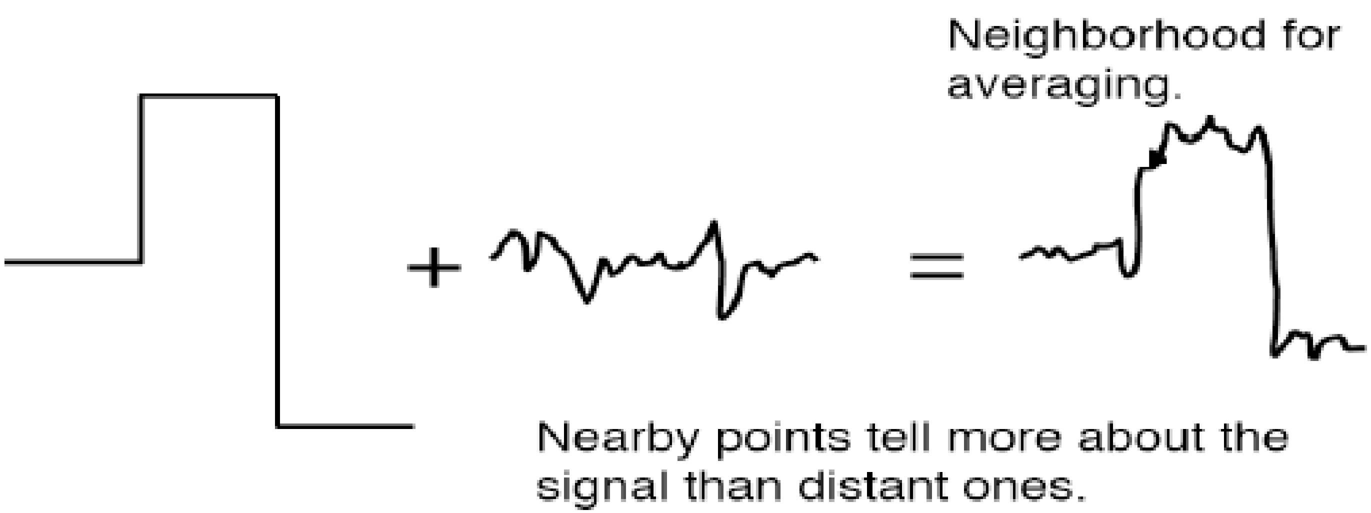
  neighbouring pixel contain information about intensity

$$\begin{array}{|c|c|c|}\hline 2 & 3 & 3 \\ \hline 3 & 20 & 2 \\ \hline 3 & 2 & 3 \\ \hline \end{array} \rightarrow \begin{array}{|c|c|c|}\hline 2 & 3 & 3 \\ \hline 3 & 3 & 2 \\ \hline 3 & 2 & 3 \\ \hline \end{array}$$

Noise removal

Signal + Noise = Image

$$S + N = I$$



The box filter does noise removal

- Box filter takes the mean in a neighbourhood

Image

Noise

Pixel-independent Gaussian noise added

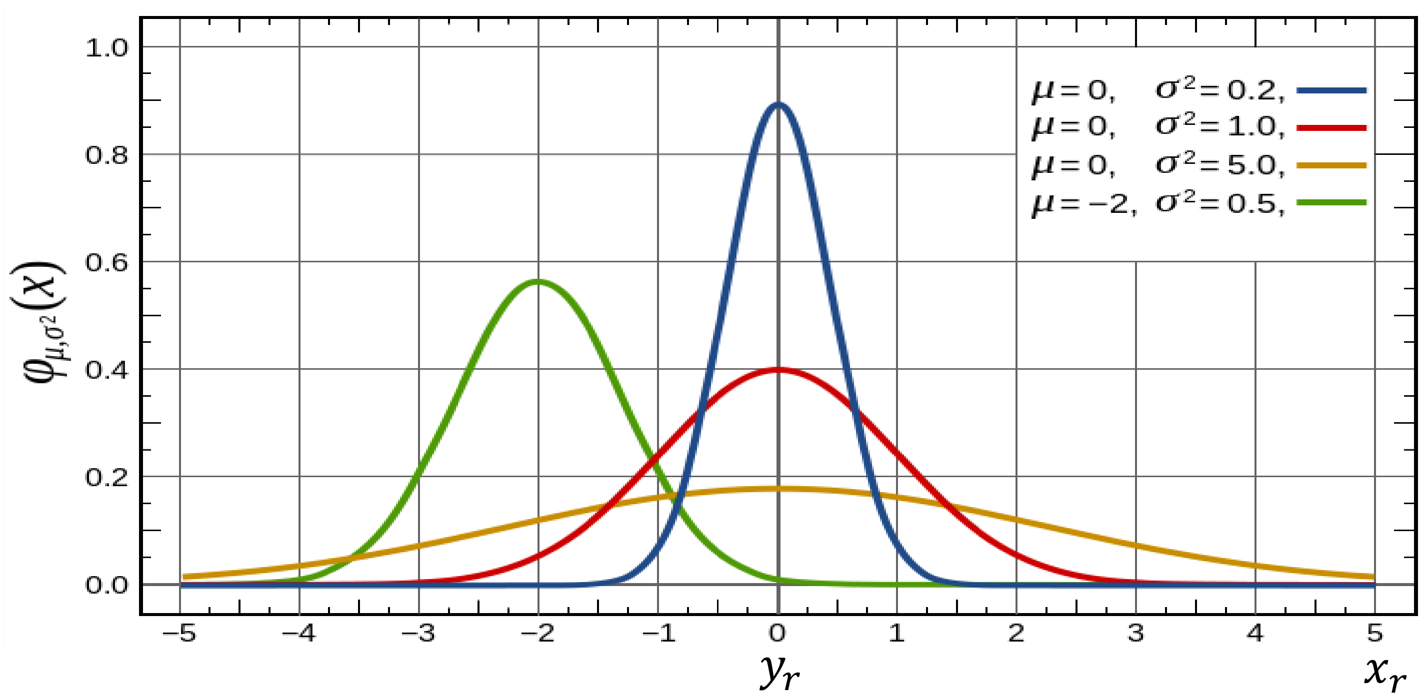
$$\frac{1}{9} \cdot \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}$$

Filtered Image

Derivation of the Box Filter

- y_r is true gray value (color)
- x_r observed gray value (color)
- Noise model: Gaussian noise:

$$p(x_r|y_r) = N\ (x_r; y_r, \sigma) \sim \exp[-\frac{||x_r - y_r||^2}{2\sigma^2}]$$



Derivation of Box Filter

Further assumption: independent noise

$$p(x|y) \sim \prod_r \exp[-\frac{||x_r - y_r||^2}{2\sigma^2}]$$

Find the most likely solution the true signal y
Maximum-Likelihood principle (probability maximization):

$$y^* = argmax_y \underbrace{p(y|x)}_{posterior} = argmax_y \frac{\overbrace{p(y)}^{prior}\overbrace{p(x|y)}^{likelihood}}{p(x)}$$

$p(x)$ is a constant (drop it out), assume (for now) uniform prior $p(y)$.
So we get:

$$p(y|x) = p(x|y) \sim \prod_r \exp[-\frac{||x_r - y_r||^2}{2\sigma^2}]$$

➡ the solution is trivial: $y_r = x_r$ for all r ☹

➡ additional assumptions about the **signal** $\boldsymbol{y}$ are necessary !!!

Derivation of Box Filter

Assumption: not uniform prior $p(y)$ but …
in a small vicinity $W(r) \subset D$ the "true" signal y_r is nearly constant

Only one y_r in a window $W(r)$

Maximum-a-posteriori:

$$p(y|x) \sim \prod_{r} \prod_{r' \in W(r)} \exp[-\frac{||x_{r'} - y_{r'}||^2}{2\sigma^2}]$$

For one pixel r :

$$y_r^* = argmax_{y_r} \prod_{r' \in W(r)} \exp[-\frac{||x_{r'} - y_r||^2}{2\sigma^2}]$$

take neg. logarithm:

$$y_r^* = argmin_{y_r} \sum_{r' \in W(r)} ||x_{r'} - y_r||^2$$

Derivation of Box Filter

How to do the minimization:

$$y_r^* = argmin_{y_r} \sum_{r' \in W(r)} ||x_{r'} - y_r||^2$$

Take derivative and set to 0:

$$F(y_r) = \sum_{r' \in W(r)} ||x_{r'} - y_r||^2$$

$$\frac{\partial F}{\partial y_r} = \sum_{r'} (x_{r'} - y_r) = \sum_{r'} x_{r'} - |W| \cdot y_r = 0$$

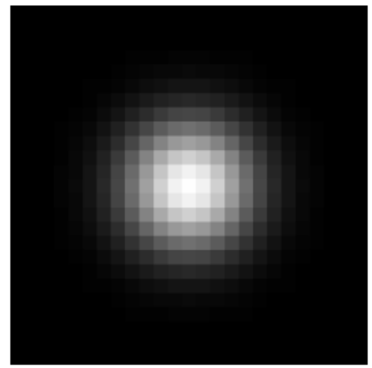
$\Rightarrow$ $y_r^* = \frac{1}{|W|} \sum_{r'} x_{r'}$ (the average)

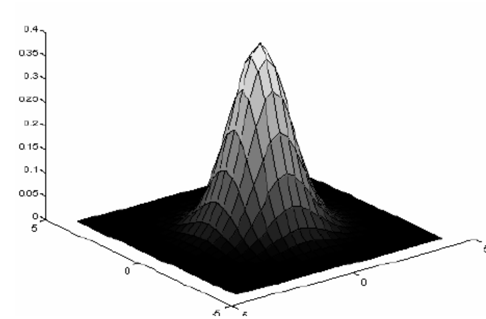
> Box filter optimal under pixel-independent Gaussian Noise and constant signal in window

Gaussian (Smoothing) Filters

- Nearby pixels are weighted more than distant pixels
- Isotropic Gaussian (rotational symmetric)

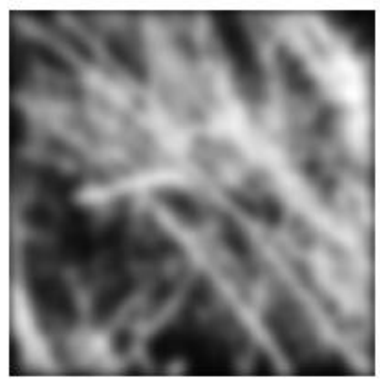
$$G_\sigma(x,y) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$$



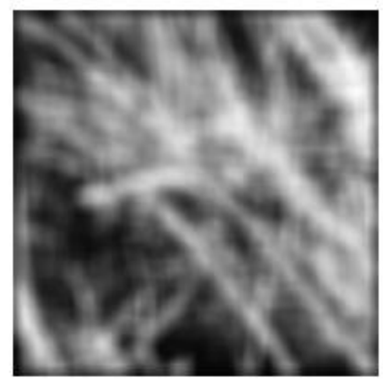


Original Image

Gaussian-filtered image

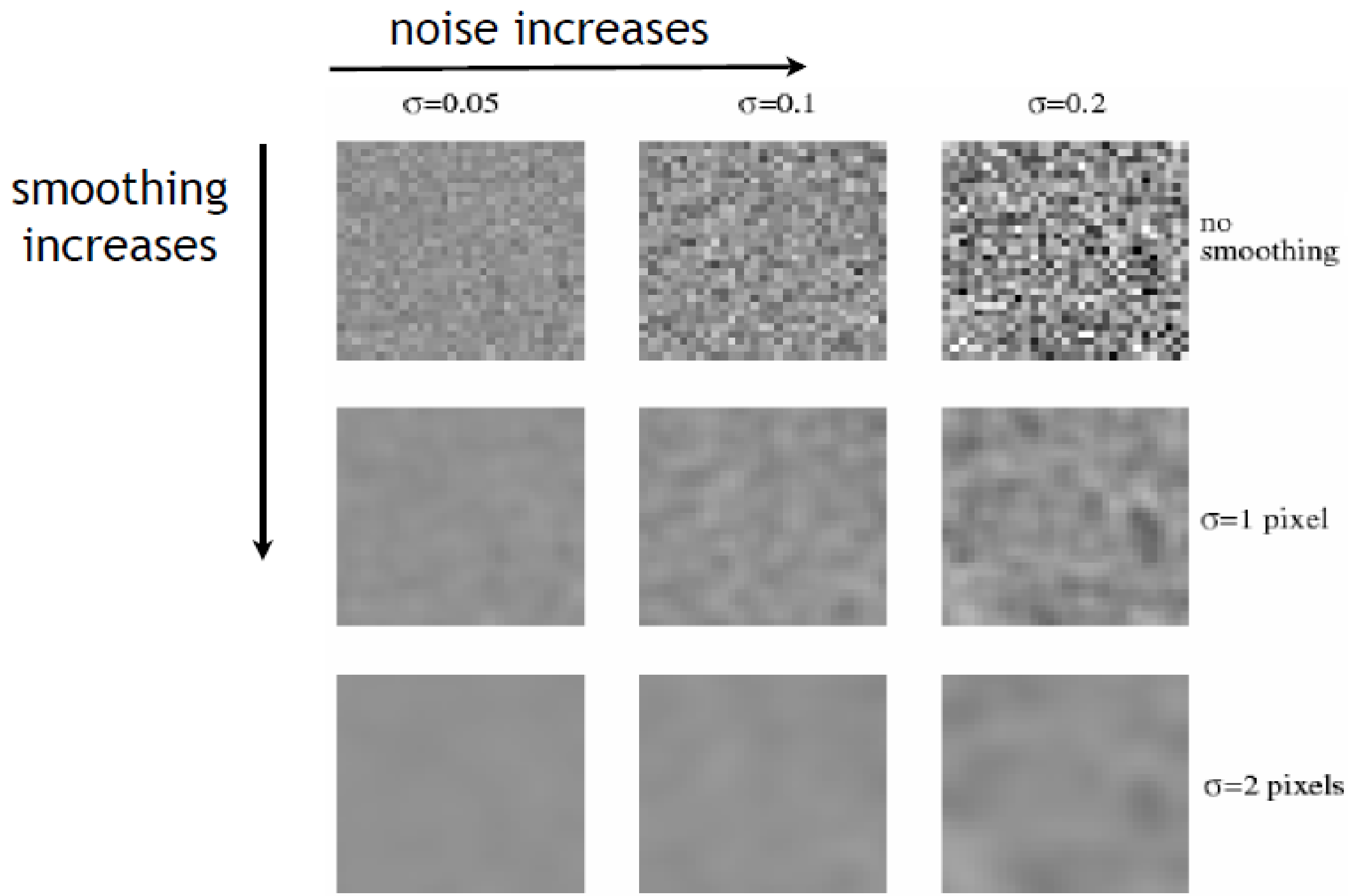


Box-filtered image



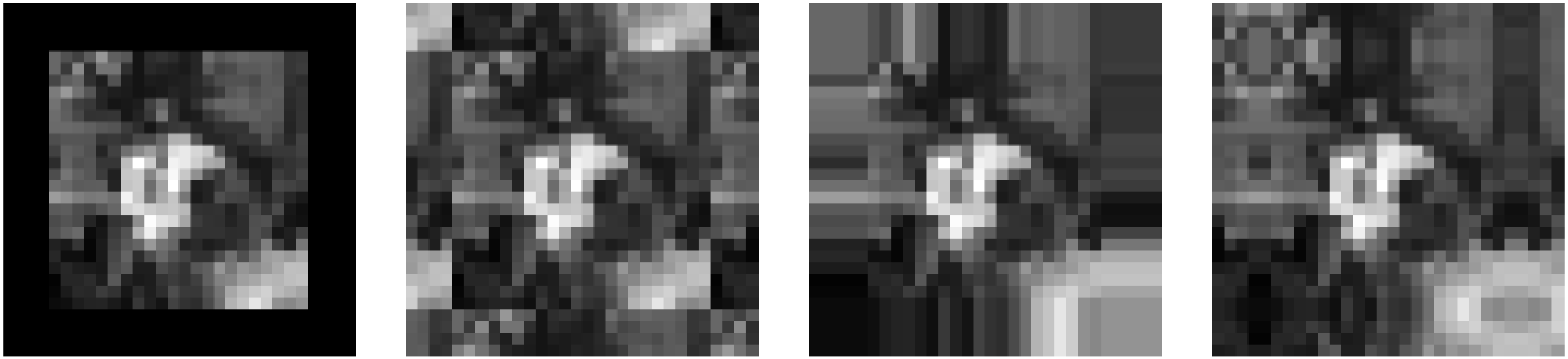
Gaussian Filter

Input: constant grey-value image

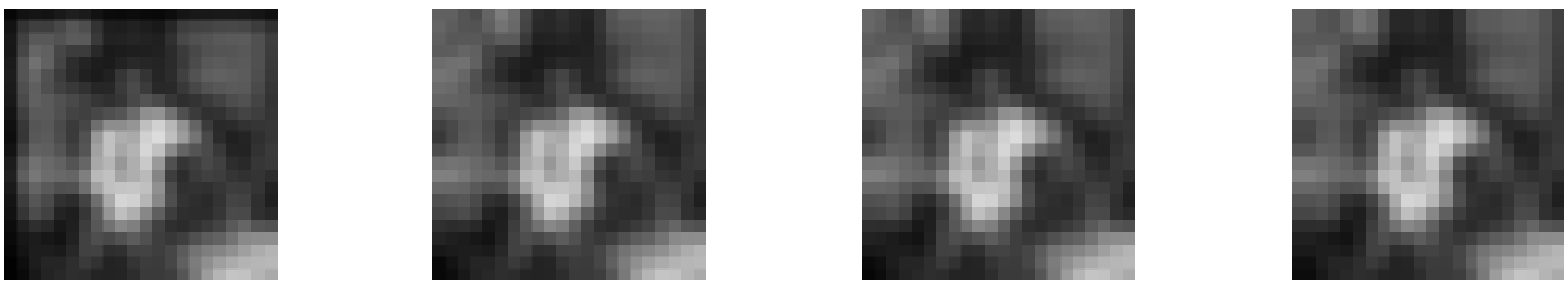


More noise needs larger sigma

Handling the Boundary (Padding)



zero | wrap | clamp | mirror

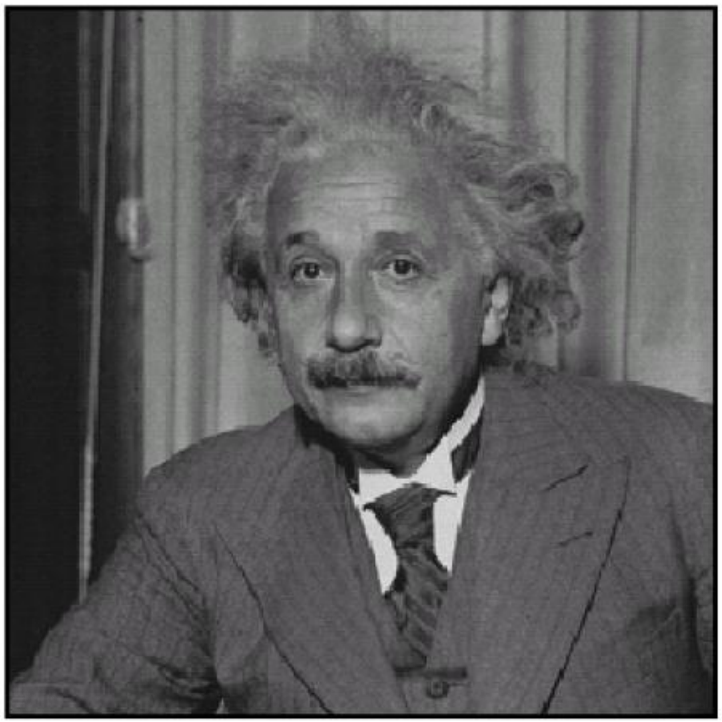


blurred zero | normalized zero | blurred clamp | blurred mirror

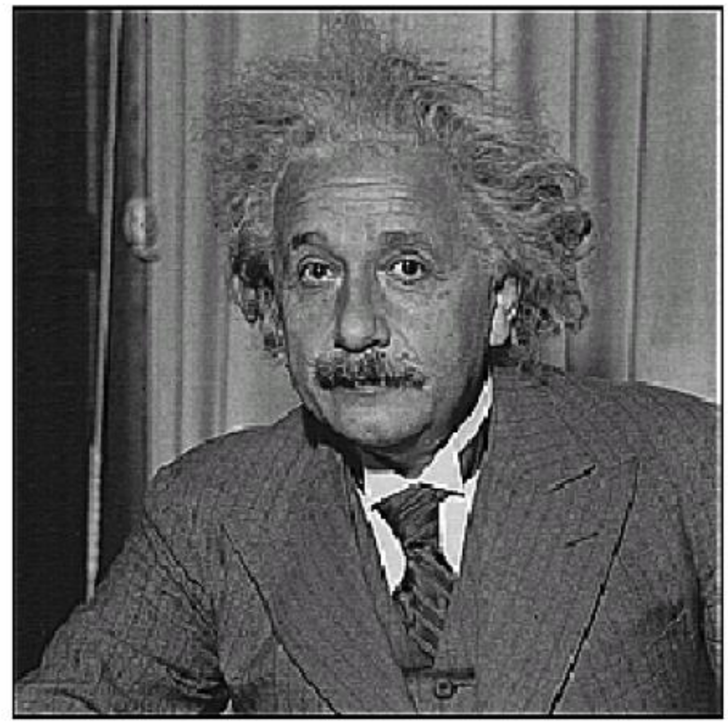
Gaussian for Sharpening

Sharpen an image by amplifying what is smoothing removes:

$$g = f + \gamma\,(f - h_{blur} * f)$$



original



sharpened

How to compute convolution efficiently?

- Separable filters (next)
- Fourier transformation (see later)
- Integral Image trick (see exercise)

<u>Important for later (integral Image trick):</u>
The Box filter (mean filter) can be computed in $O(N)$.
Naive implemettaioin would be $O(Nw)$
where w is the number of elements in box filter

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

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

image

*

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

filter (kernel)

=

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

filtered image

Separable filters

For some filters we have: $f * h = f * (h_x * h_y)$

Where h_x, h_y are 1D filters.

Example Box filter:

$$\underbrace{\frac{1}{9} \cdot \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline \end{array}}_{h_x * h_y} = \underbrace{\frac{1}{3} \cdot \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline \end{array}}_{h_x} * \underbrace{\frac{1}{3} \cdot \begin{array}{|c|}\hline 1 \\ \hline 1 \\ \hline 1 \\ \hline \end{array}}_{h_y}$$

Now we can do two 1D convolutions:

$$f * h = f * \left(h_x * h_y\right) = (f * h_x) * h_y$$

Naïve implementation for 3x3 filter: 9N operations versus 3N+3N operations

Can any filter be made separable?

Note:
$$\underbrace{\frac{1}{9}\cdot\begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array}}_{h_x * h_y} = \frac{1}{3}\cdot\underbrace{\begin{array}{|c|c|c|}\hline 1&1&1\\\hline\end{array}}_{h_x} * \frac{1}{3}\cdot\underbrace{\begin{array}{|c|}\hline 1\\\hline 1\\\hline 1\\\hline\end{array}}_{h_y} = \frac{1}{3}\cdot\underbrace{\begin{array}{|c|}\hline 1\\\hline 1\\\hline 1\\\hline\end{array}}_{h_x}\cdot\frac{1}{3}\cdot\underbrace{\begin{array}{|c|c|c|}\hline 1&1&1\\\hline\end{array}}_{h_y}$$

<u>Apply SVD to the kernel matrix:</u>

$$\boldsymbol{A} = \left[\begin{array}{c|c|c} u_0 & \cdots & u_{p-1} \end{array}\right]\begin{bmatrix}\sigma_0 & & \\ & \ddots & \\ & & \sigma_{p-1}\end{bmatrix}\left[\begin{array}{c} v_0^T \\ \hline \cdots \\ \hline v_{p-1}^T \end{array}\right]$$

$$= \sum_{j=0}^{t}\sigma_j u_j v_j^T,$$

If all σ_i are 0 (apart from σ_0) then it is separable.

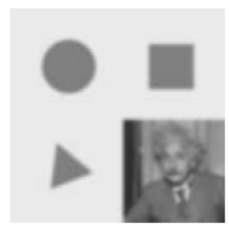
Example of separable filters

$$\frac{1}{K^2}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline 1 & 1 & \cdots & 1 \\ \hline \vdots & \vdots & 1 & \vdots \\ \hline 1 & 1 & \cdots & 1 \\ \hline \end{array}
\qquad
\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}
\qquad
\frac{1}{256}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 6 & 24 & 36 & 24 & 6 \\ \hline 4 & 16 & 24 & 16 & 4 \\ \hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}
\qquad
\frac{1}{8}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array}
\qquad
\frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline -2 & 4 & -2 \\ \hline 1 & -2 & 1 \\ \hline \end{array}$$

$$\frac{1}{K}\begin{array}{|c|c|c|c|}\hline 1 & 1 & \cdots & 1 \\ \hline \end{array}
\qquad
\frac{1}{4}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline \end{array}
\qquad
\frac{1}{16}\begin{array}{|c|c|c|c|c|}\hline 1 & 4 & 6 & 4 & 1 \\ \hline \end{array}
\qquad
\frac{1}{2}\begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline \end{array}
\qquad
\frac{1}{2}\begin{array}{|c|c|c|}\hline 1 & -2 & 1 \\ \hline \end{array}$$

(a) box, $K = 5$

(b) bilinear



(c) “Gaussian”

(d) Sobel

(e) corner

Roadmap: Basics Digital Image Processing

- Images
- Point operators (ch. 3.1)
- Filtering: (ch. 3.2, ch 3.3, ch. 3.4) – main focus
  - Linear filtering
  - Non-linear filtering
- Fourier Transformation (ch, 3.4)
- Multi-scale image representation (ch. 3.5)
- Edges (ch. 4.2)
  - Edge detection and linking
- Lines (ch. 4.3)
  - Line detection and vanishing point detection

Non-linear filters

- There are many different non-linear filters.
  We look at a selection:
  - Median filter
  - Bilateral filter (Guided Filter)
  - Morphological operations

Shot noise (Salt and Pepper Noise) - motivation

Original + shot noise

Gaussian filtered

Median filtered

Another example

Original

Noised

Mean

Median

Median Filter

Replace each pixel with the median in a neighbourhood:

5	6	5
4	20	5
4	6	5

5	6	5
4	5	5
4	6	5

Median filter: order the values and take the **middle** one

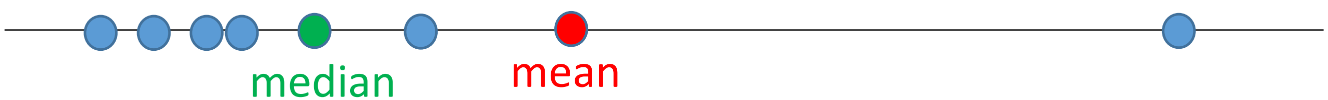
- No strong smoothing effect since values are not averaged
- Very good to remove outliers (shot noise)

Used a lot for post processing of outputs (e.g. optical flow)

Median Filter: Derivation

Reminder: for Gaussian noise we did solve the following ML problem

$$y_r^* = argmax_{y_r} \underbrace{\prod_{r' \in W(r)} \exp[-\frac{||x_{r'} - y_r||^2}{2\sigma^2}]}_{p(y|x)} = argmin_{y_r} \sum_{r' \in W(r)} ||x_{r'} - y_r||^2 = 1/|W| \sum_{r' \in W(r)} x_r$$



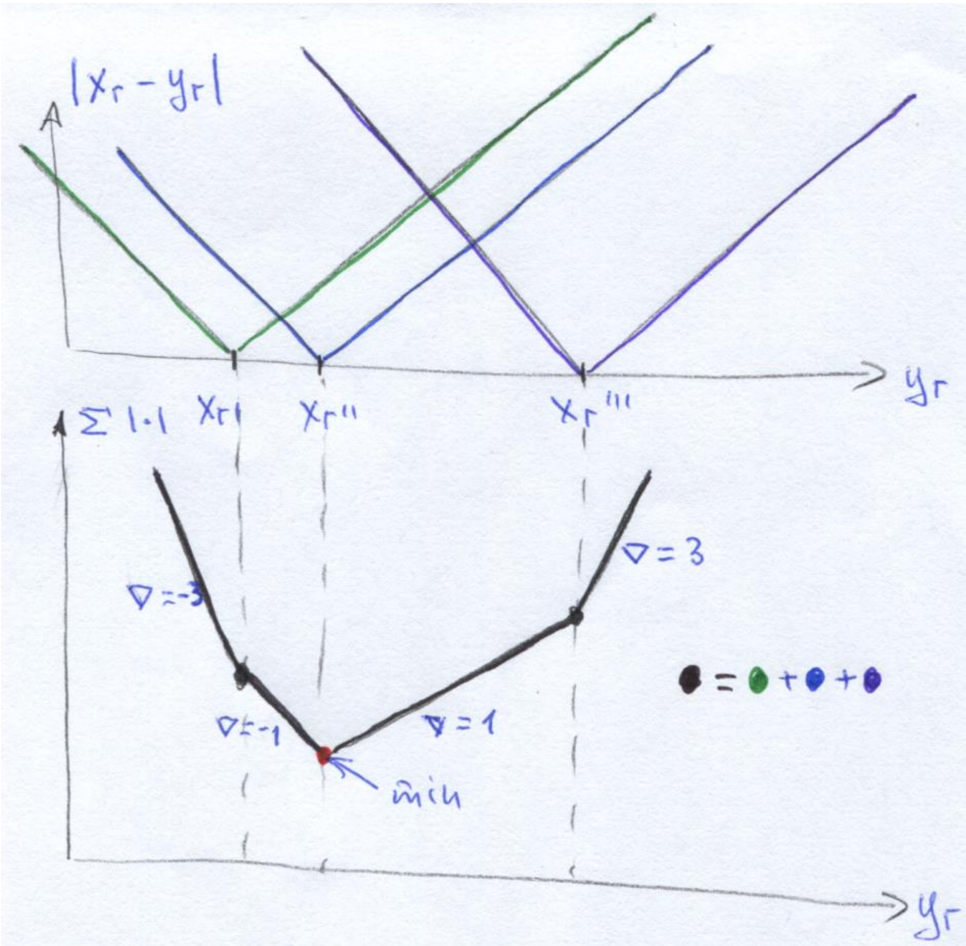
Does not look like a Gaussian distribution

For Median we solve the following problem:

$$y_r^* = argmax_{y_r} \prod_{r' \in W(r)} \exp[-\frac{|x_{r'} - y_r|}{2\sigma^2}] = argmin_{y_r} \sum_{r' \in W(r)} |x_{r'} - y_r| = Median\ (W(r))$$

Due to absolute norm it is more robust

Median Filter Derivation



Optimal solution is
the mean of all values

minimize the following:

$$F(y_r) = \sum_{r' \in W(r)} |x_{r'} - y_r|$$

Problem: not differentiable ☹,
good news: it is convex ☺

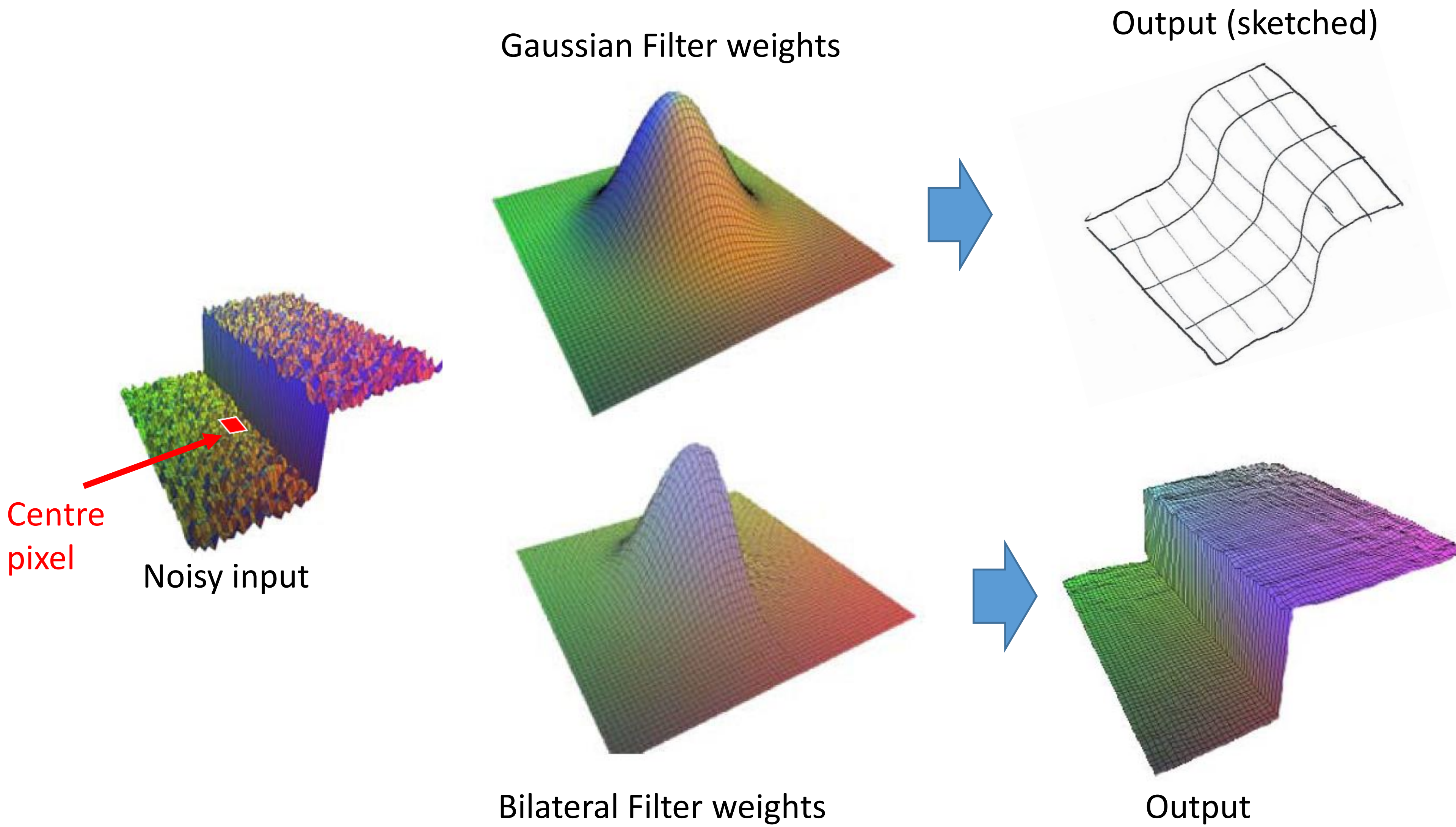
Motivation – Bilateral Filter

Original + Gaussian noise

Gaussian filtered

Bilateral filtered

Bilateral Filter – in pictures



Bilateral Filter – in equations

Filters looks at: a) distance of surrounding pixels (as Gaussian)
b) Intensity of surrounding pixels

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

Linear combination

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

Similar to Gaussian filter *Consider intensity*

Problem: computation is slow $O(Nw)$; approximations can be done in $O(N)$
Comment: Guided filter (see later) is similar and can be computed exactly in $O(N)$

See a tutorial on: http://people.csail.mit.edu/sparis/bf_course/

Application: Bilteral Filter

Cartoonization

Original HDR

Bilateral Filter

HDR compression
(Tone mapping)

Joint Bilteral Filter

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

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

Similar to Gaussian *Consider intensity*

f is the input image – which is processed

$\tilde{f}$ is a guidance image – where we look for pixel similarity

Application: combine Flash and No-Flash

input image f

guidance image $\tilde{f}$

Joint Bilateral Filter

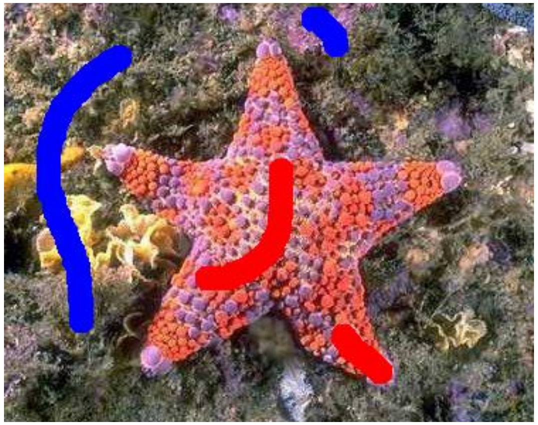
We don't care about absolute colors

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

[Petschnigg et al. Siggraph '04]

Application: Cost Volume Filtering

Reminder from first Lecture: Interactive Segmentation



Goal

$\boldsymbol{z} = (R, G, B)^n$ $\qquad$ $\mathbf{x} = \{0,1\}^n$

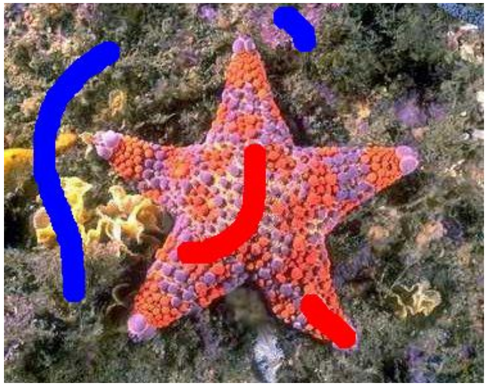
Given **z**; derive binary **x**:

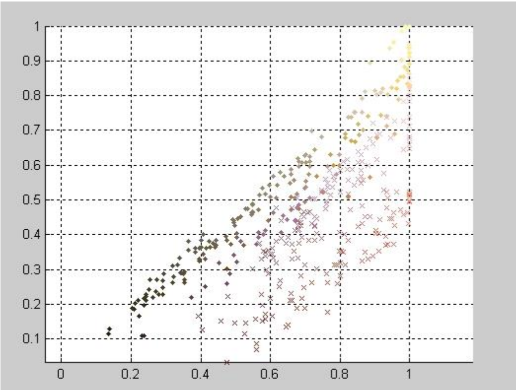
Model: Energy function $\boldsymbol{E}(\boldsymbol{x}) = \sum_i \theta_i(x_i) + \sum_{i,j} \theta_{ij}(x_i, x_j)$

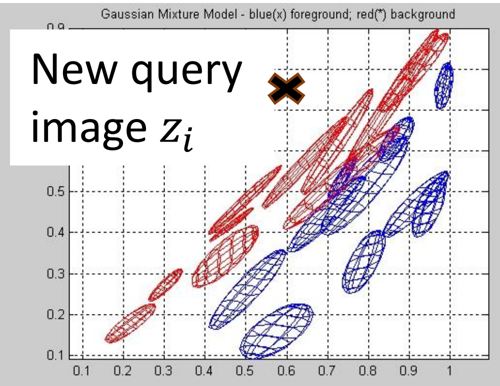
Unary terms $\qquad$ Pairwise terms

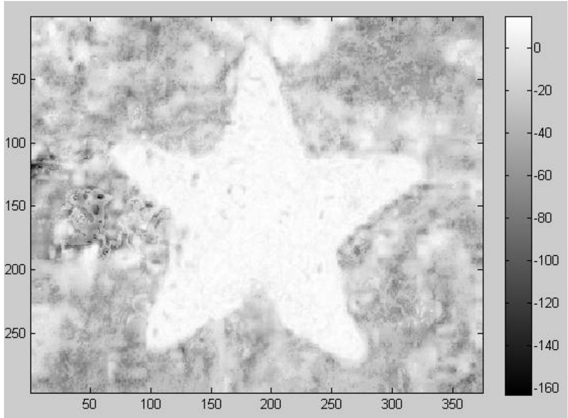
Algorithm to minimization: $\boldsymbol{x}^* = argmin_x\, E(\boldsymbol{x})$

Reminder: Unary term



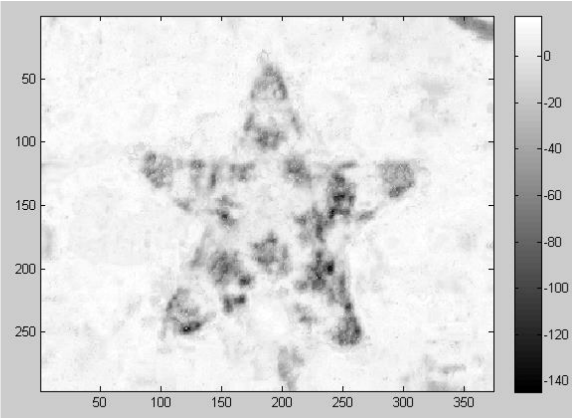






$\theta_i(x_i = 0)$

Dark means likely background

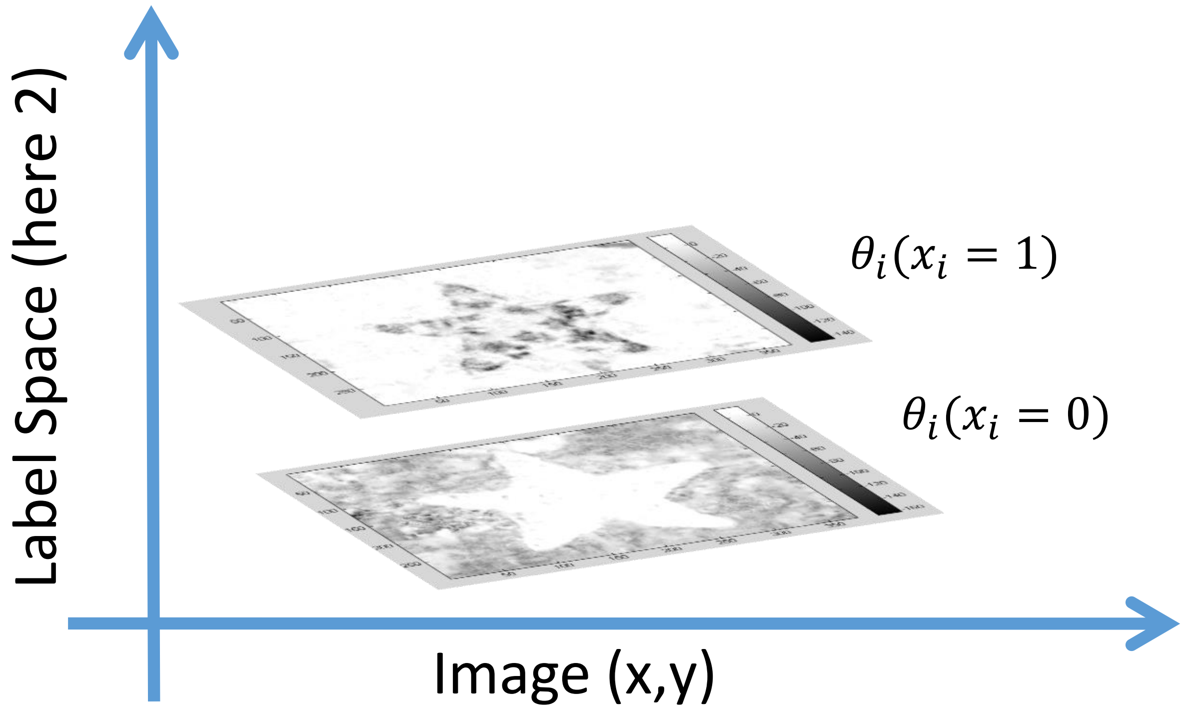


$\theta_i(x_i = 1)$

Dark means likely foreground

Optimum with unary terms only

Cost Volumne for Binary Segmenation

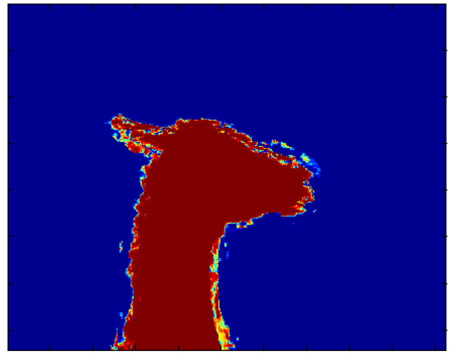


For 2 Labels, we can also look at the ratio Image:

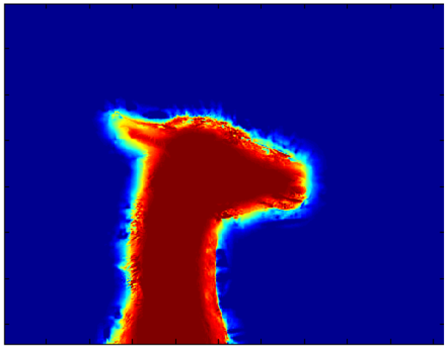
$$I_i = \theta_i(x_i = 1) \ / \ \theta_i(x_i = 0)$$

Application: Cost Volumne Filtering

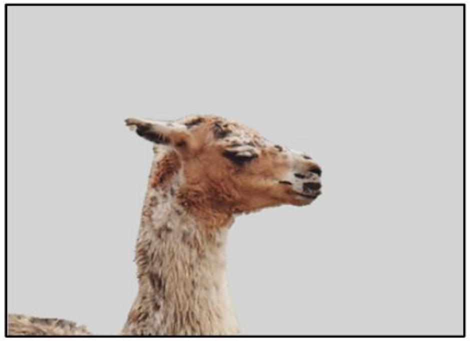
Guidance Input Image $\tilde{f}$ (user brush strokes)



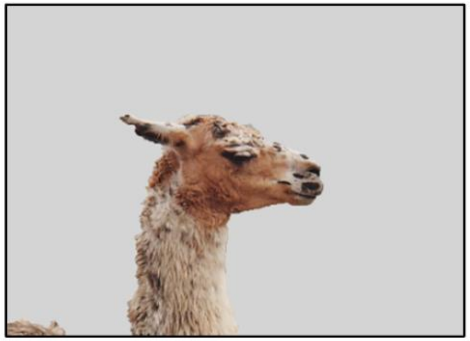
Ratio Cost-volume is the Input Image f



Filtered cost volume



Winner takes all Result



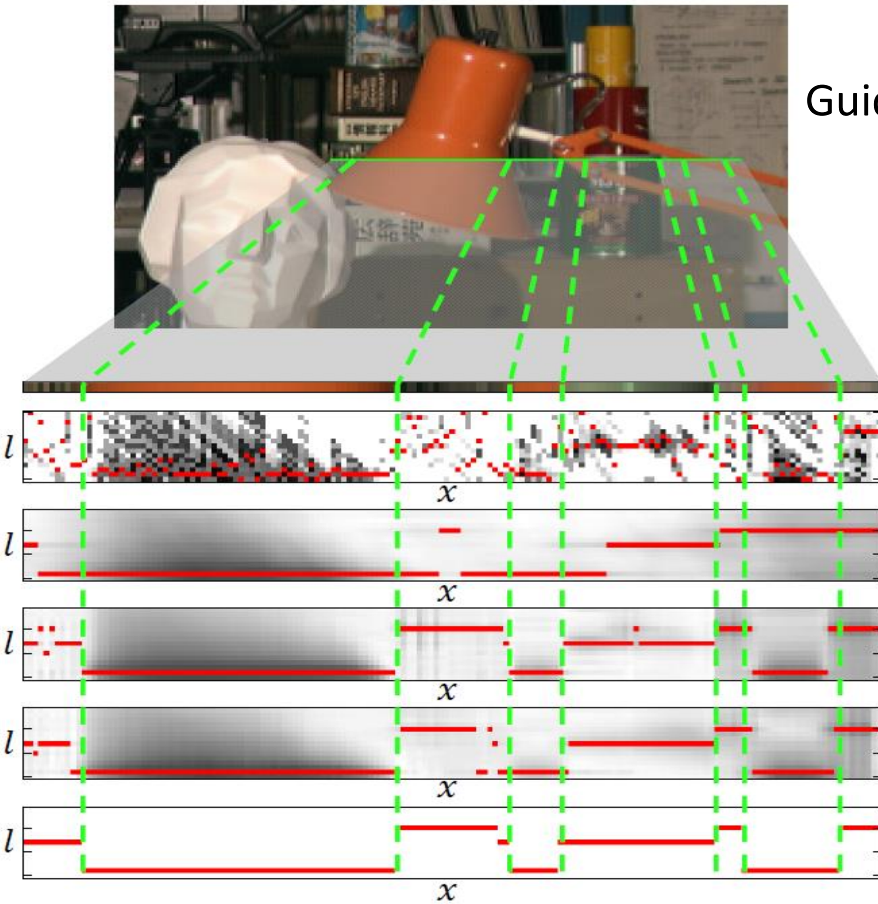
Energy minimization

An alternative to energy minimization

[C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, Fast Cost-Volume Filtering for Visual Correspondence and Beyond, CVPR 11]

Application: Cost volume filtering for dense Stereo

Stereo Image pair



Guidance Image $\tilde{f}$

20-label cost volume f

Box filter

Bilateral filter

Guided filter

True solution

Stereo result
(winner takes all)

[C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, Fast Cost-Volume Filtering for Visual Correspondence and Beyond, CVPR 11]

Application: Cost volume filtering for dense Stereo

Method	Rank	Avg. Error (%)	Avg. Runtime (ms)
Ours	9	5.55	65
GeoSup [12]	12	5.80	16000
Plane-fit BP	13	5.78	650
Ours using AdaptWeight [31]	15	5.86	15000
AdaptWeight [31]	32	6.67	8550
Real-time GPU	66	9.82	122.5
Reliability DP	69	10.7	187.8
DCB Grid [19]	76	10.9	95.4*

Very competative in terms of results for a fast methods
(Middleburry Ranking)

[C. Rhemann, A. Hosni, M. Bleyer, C. Rother, and M. Gelautz, Fast Cost-Volume Filtering for Visual Correspondence and Beyond, CVPR 11]

Recent Trend: Guided Filter

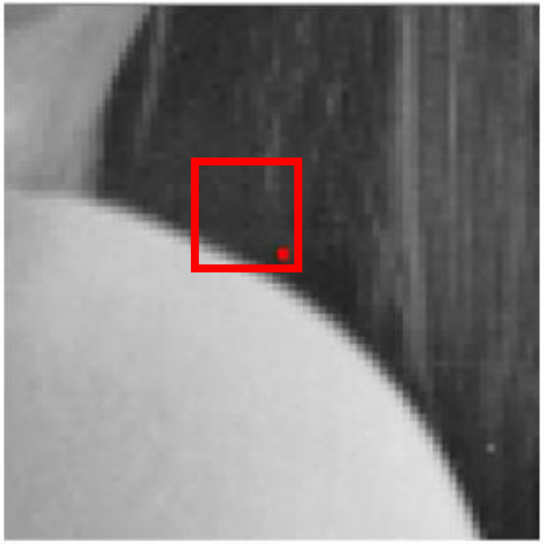
$g_i = \sum_j W_{ij}(\tilde{f}) f_i$

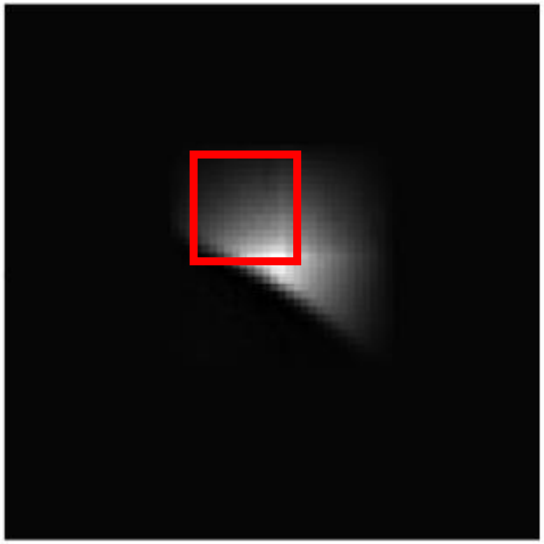
Diferent pixel coordinates i, j
linear combination of image f

$$W_{ij}(\tilde{f}) = \frac{1}{|\omega|^2} \sum_{k:(i,j)\in\omega_k} \left(1 + \frac{(\tilde{f}_i - \mu_k)(\tilde{f}_j - \mu_k)}{\sigma_k^2 + \epsilon}\right)$$

„Different to biltarel filter since a sum over small windows“

Size of window ω_k is fixed, e.g. 7x7.
Sum over all windows ω_k which contain pixels: i and j





ω_k
7x7 pixels

[He, Sun ECCV ‘10]

Recent Trend: Guided Filter

$g_i = \sum_j W_{ij}(\tilde{f}) f$

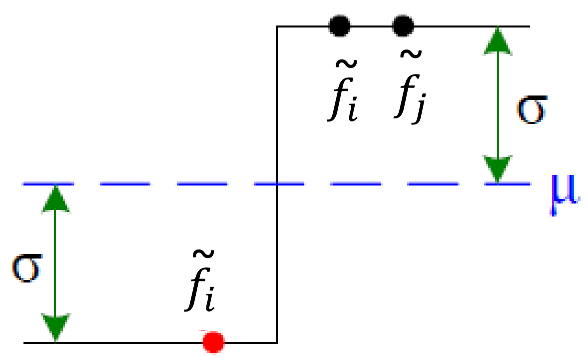
Diferent pixel cooridinates i, j
linear combination of image f

„Different to biltarel filter since a sum over small windows“

$$W_{ij}(\tilde{f}) = \frac{1}{|\omega|^2} \sum_{k:(i,j)\in\omega_k} \left(1 + \frac{(\tilde{f}_i - \mu_k)(\tilde{f}_j - \mu_k)}{\sigma_k^2 + \epsilon}\right)$$

mean in window ω_k
variance in window ω_k

Size of window ω_k is fixed, e.g. 7x7.
Sum over all windows ω_k which contain pixels: i and j

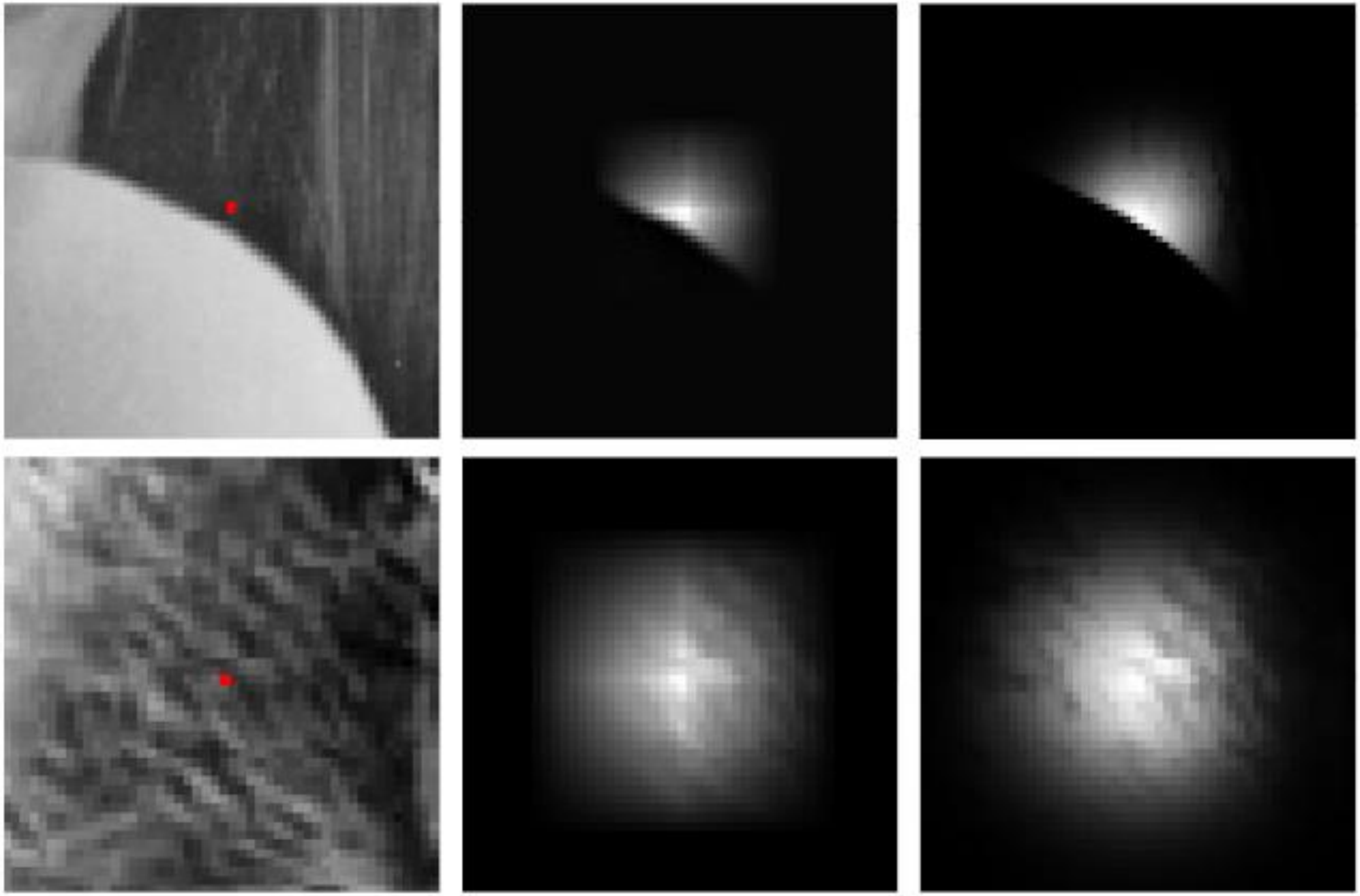


A window ω_k which is centred exactly on the edge

Case 1: I_i , I_j on the same side: $(\tilde{f}_i - \mu_k)$ $(\tilde{f}_j - \mu_k)$ have the same sign. Then W_{ij} large

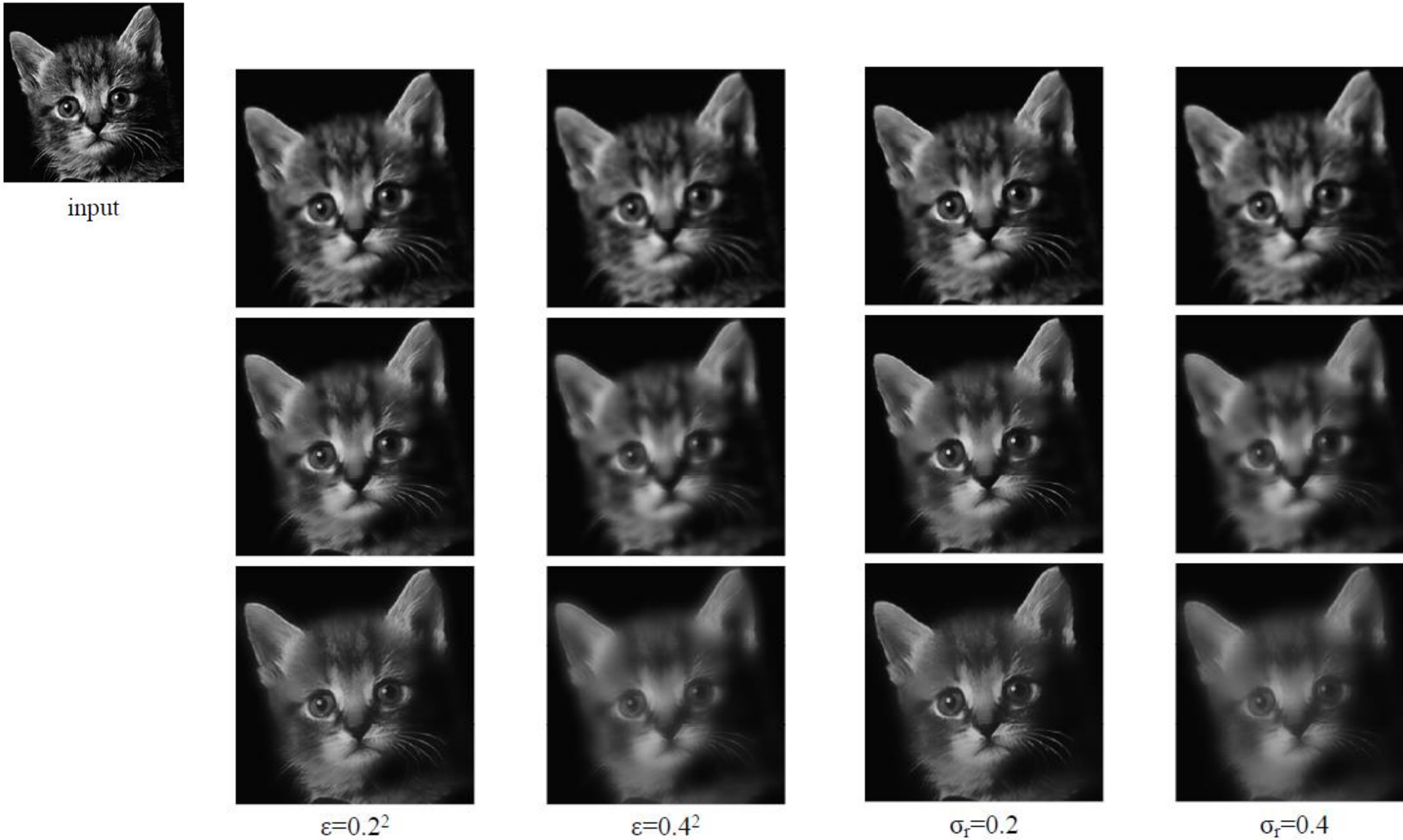
Case 2: I_i , I_j on the same side: $(\tilde{f}_i - \mu_k)$ $(\tilde{f}_j - \mu_k)$ have different sign. Then W_{ij} small

Bilteral Filter and Guided Filter behave very similiarly



Guidance I | Guided Filter's Kernel | Bilateral Filter's Kernel

Bilteral Filter and Guided Filter behave very similiarly



Guided Filter

Bilteral Filter

Guided Filter: Can be computed in O(N)

$$g_i = \sum_j W_{ij}(\tilde{f}) f \qquad W_{ij}(\tilde{f}) = \frac{1}{|\omega|^2} \sum_{k:(i,j)\in\omega_k} \left(1 + \frac{(\tilde{f}_i - \mu_k)(\tilde{f}_j - \mu_k)}{\sigma_k^2 + \epsilon}\right)$$

Can also be written as: $g_i = \bar{a}_i \tilde{f}_i + \bar{b}_i$ (see paper for detail)

$$\mu_k = \frac{1}{|\omega|} \sum_{j\in\omega_k} \tilde{f}_j \text{ mean guidance image } O(N)$$

$$\sigma_k^2 = \frac{1}{|\omega|} \sum_{j\in\omega_k} (\tilde{f}_j - \mu_k)^2 \text{ variance } \tilde{f} \; 3O(N)$$

$$\sigma_k^2 = \frac{1}{|\omega|} \sum_{j\in\omega_k} \tilde{f}_j^2 - \frac{2\mu_k}{|\omega|} \sum_{j\in\omega_k} \tilde{f}_j + \mu_k$$

$$\bar{p}_k = \frac{1}{|\omega|} \sum_{j\in\omega_k} f_j \text{ mean image } O(N)$$

$$a_k = \frac{\left(\frac{1}{|\omega|} \sum_{j\in\omega_k} f_j \tilde{f}_j\right) - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon} \text{ computation } 2O(N)$$

$$b_k = \bar{p}_k - a_k \mu_k \text{ linear combination } O(N)$$

$$\bar{b}_i = \frac{1}{|\omega|} \sum_{k\in\omega} b_k \text{ mean computation } O(N)$$

$$\bar{a}_i = \frac{1}{|\omega|} \sum_{k\in\omega} a_k \text{ mean computation } O(N)$$

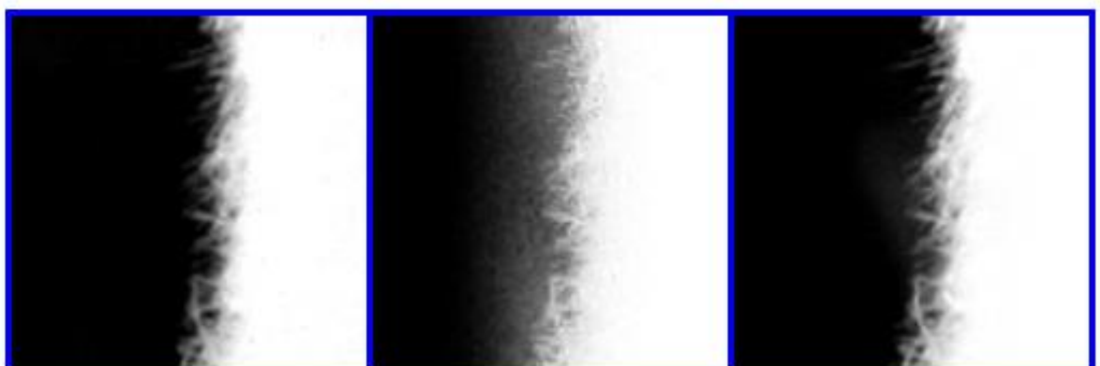
$$g_i = \bar{a}_i \tilde{f}_i + \bar{b}_i \text{ linear combination } O(N)$$

Integral Image trick

[He, Sun ECCV '10]

Applications: Matting

Guideanace Image $\tilde{f}$ | Input Image f | Output Image g

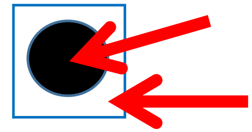


Guided Filter | Photoshop | Closed-form

[He, Sun ECCV '10]

Morphological operations

- Perform convolution with a "structural element": binary mask (e.g. circle or square)



- Then perform thresholding to recover a binary image

$$\theta(f,t) = \begin{cases} 1 & \text{if } f \geq t, \\ 0 & \text{else,} \end{cases}$$

original

dilation $\theta(f * s, 1)$

erosion $\theta(f * s, S)$

# of pixels in s

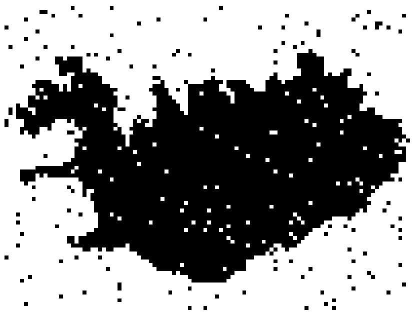
Opening and Closing Operations

- Opening operation: $dilate(erode(f, s), s)$
- Closing opertiaon: $erode(dialte(f, s), s)$

Input image opening closing

erode and dilate are not **commutative**

Application: Denoise Binary Segmentation



Input
Segmentation

Opening → than closing

Closing → than opening

Note: nothing is **commutative**

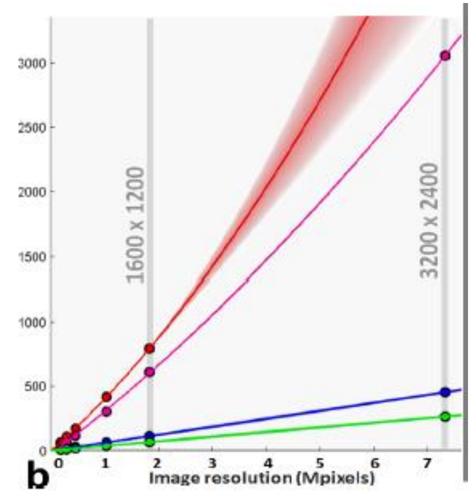
Application: Binary Segmentation

Extend morphological operations to deal with cost volume and make it edge preserving (same idea as in joint bilateral filter)

Ratio Cost-volume is the Input Image f

Result: Edge preserving Opening and closing

Energy minimization



Energy minimization

ours

Again: An alternative to energy minization

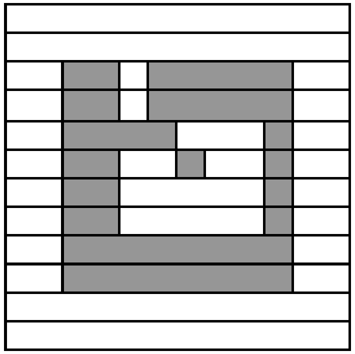
[Criminisi, Sharp, Blake, GeoS: Geodesic Image Segmentation, ECCV 08]

Related nonlinear operations on binary images

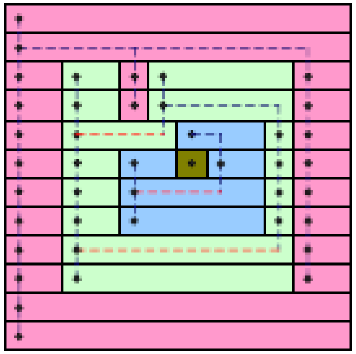
Binary Image

Distance transform

Skeleton



Binary Input Image



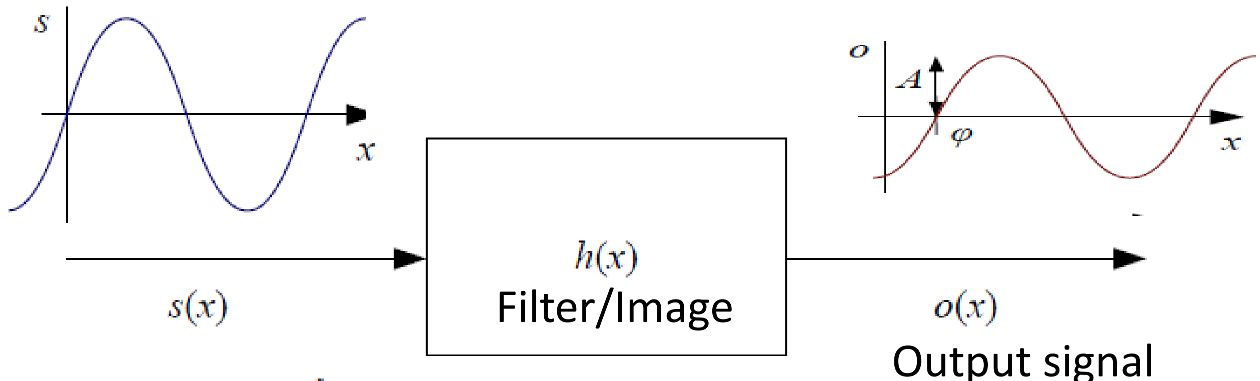
Connected components

Roadmap: Basics Digital Image Processing

- Images
- Point operators (Ch. 3.1)
- Filtering: (ch. 3.2, ch 3.3, ch. 3.4) – main focus
  - Linear filtering
  - Non-linear filtering
- Fourier Transformation (ch. 3.4)
- Multi-scale image representation (ch. 3.5)
- Edges (Ch. 4.2)
  - Edge detection and linking
- Lines (Ch 4.3)
  - Line detection and vanishing point detection

Fourier Transformation ... to analyse Filters

How does a sinusoid influences a given filter/Image $h(x)$?



$$s(x) = e^{j\omega x} = \cos \omega x + j \sin \omega x.$$

Complex valued, continuous sinusoid for different frequency ω

$$o(x) = h(x) * s(x) = \underbrace{A}_{\textit{Amplitude}}\, e^{j(wx+\phi)} = A\,[\cos(\omega x + \underbrace{\phi}_{\textit{phase}}) + j\, sin(\omega x + \phi)\,]$$

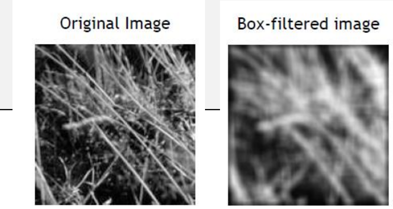
The output is also a sinusoid

Simply try all possible ω and record A, ϕ .

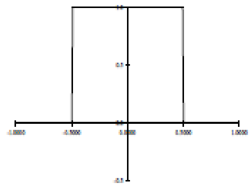
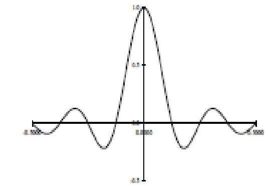
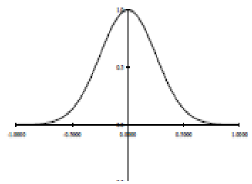
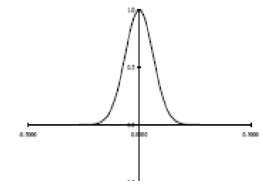
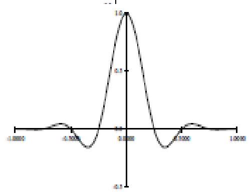
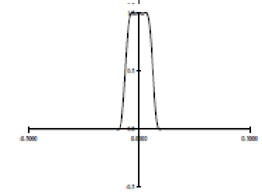
The Fourier transformation of $h(x)$ is then:

$H(\omega) = \mathcal{F}\,\{h(x)\} = A\, e^{j\phi} = A\,(\cos(\phi) + j\,\sin(\phi))$

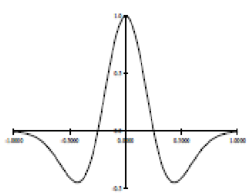
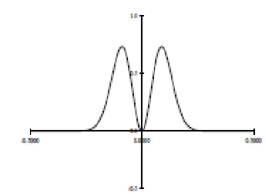
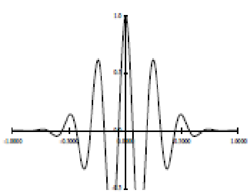
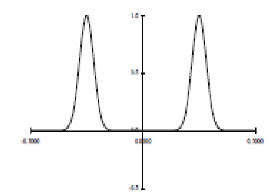
Fourier Transform



Low-pass filter:

box filter		$\text{box}(x/a)$	$\overset{\mathcal{F}}{\Leftrightarrow}$	$a\text{sinc}(a\omega)$	
Gaussian		$G(x;\sigma)$	$\overset{\mathcal{F}}{\Leftrightarrow}$	$\frac{\sqrt{2\pi}}{\sigma}G(\omega;\sigma^{-1})$	
windowed sinc		$\text{rcos}(x/(aW))$ $\text{sinc}(x/a)$	$\overset{\mathcal{F}}{\Leftrightarrow}$	(see Figure 3.29)	

Band-pass filter:

Laplacian of Gaussian		$(\frac{x^2}{\sigma^4}-\frac{1}{\sigma^2})G(x;\sigma)$	$\overset{\mathcal{F}}{\Leftrightarrow}$	$-\frac{\sqrt{2\pi}}{\sigma}\omega^2 G(\omega;\sigma^{-1})$	
Gabor		$\cos(\omega_0 x)G(x;\sigma)$	$\overset{\mathcal{F}}{\Leftrightarrow}$	$\frac{\sqrt{2\pi}}{\sigma}G(\omega\pm\omega_0;\sigma^{-1})$	

Fourier Pair: Computation

$$h(x) \overset{\mathcal{F}}{\leftrightarrow} H(\omega)$$

continuous

$$H(\omega) = \int_{-\infty}^{\infty} h(x)\, e^{-j\omega x}\, dx \qquad h(x) = \int_{-\infty}^{\infty} H(\omega)\, e^{j\omega x}\, d\omega$$

discrete

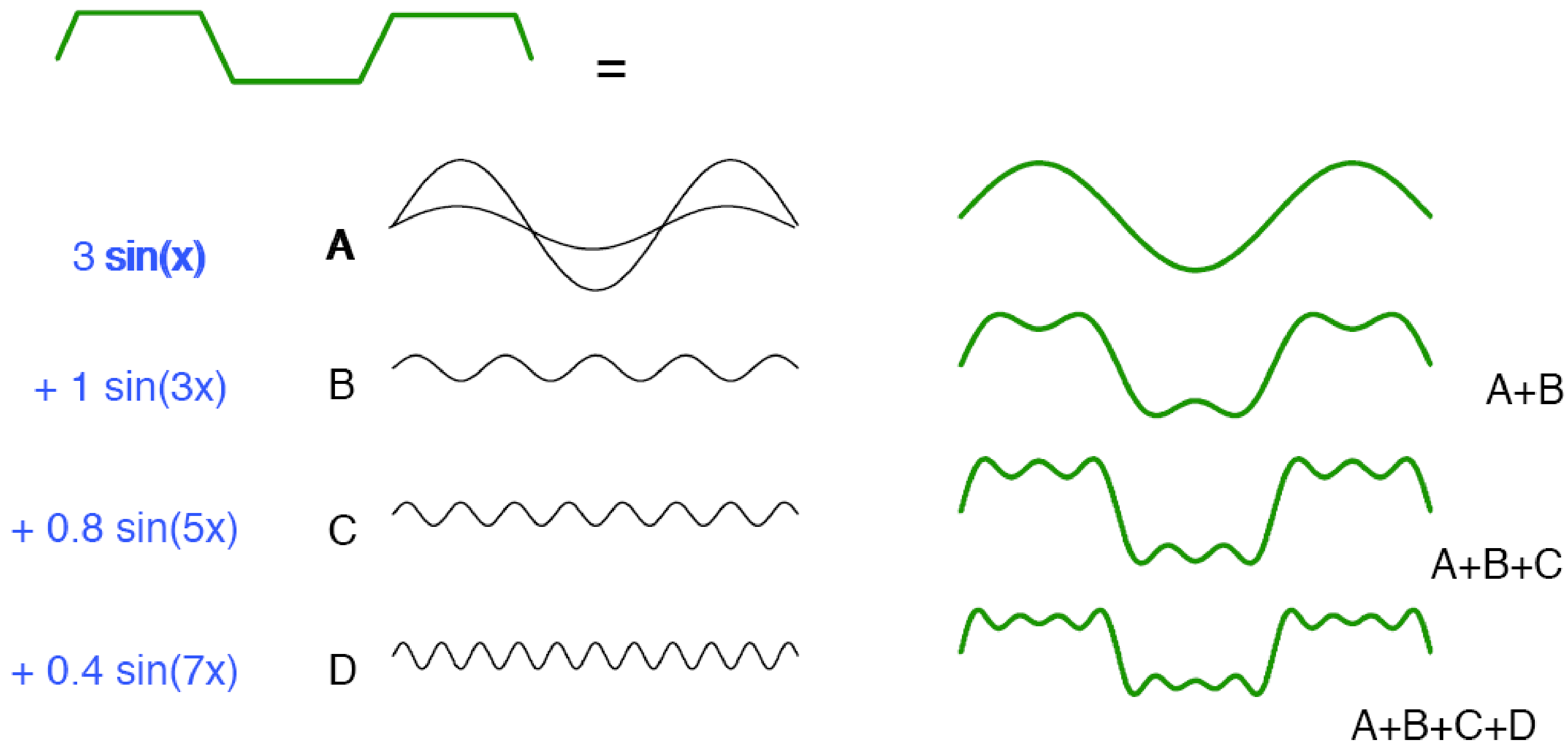
$$H(k) = \frac{1}{N} \sum_{x=0}^{N-1} h(x)\, e^{-j\frac{2\pi k x}{N}} \qquad h(x) = \frac{1}{N} \sum_{k=0}^{N-1} H(k) e^{j\frac{2\pi k x}{N}}$$

Discrete Fourier transformation
N is the range of signal (image region)

Inverse Discrete Fourier transformation

Discrete Inverse Fourier Transform: Visualization

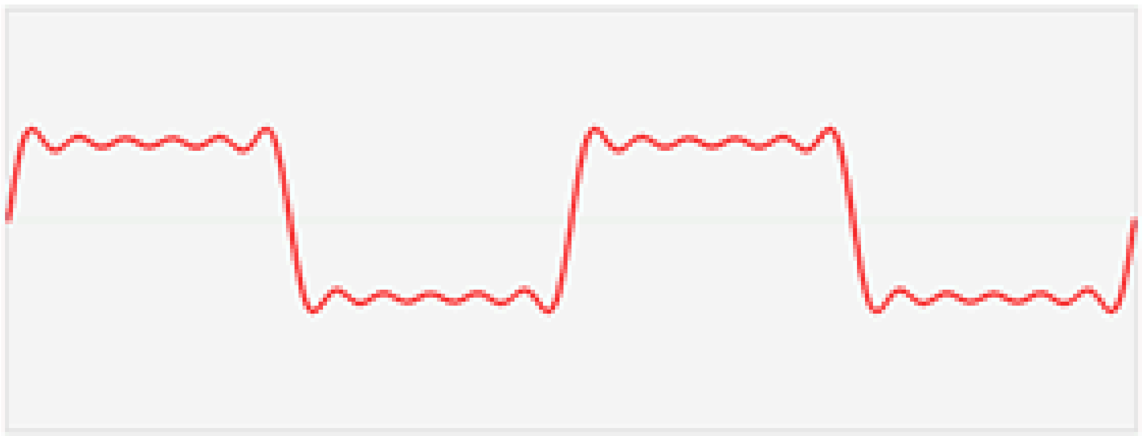
$$\mathrm{h(x)} = \frac{1}{N}\sum_{k=0}^{N-1} H(k)e^{j\frac{2\pi kx}{N}}$$



For this signal a reconstruction with sinus function only is sufficient

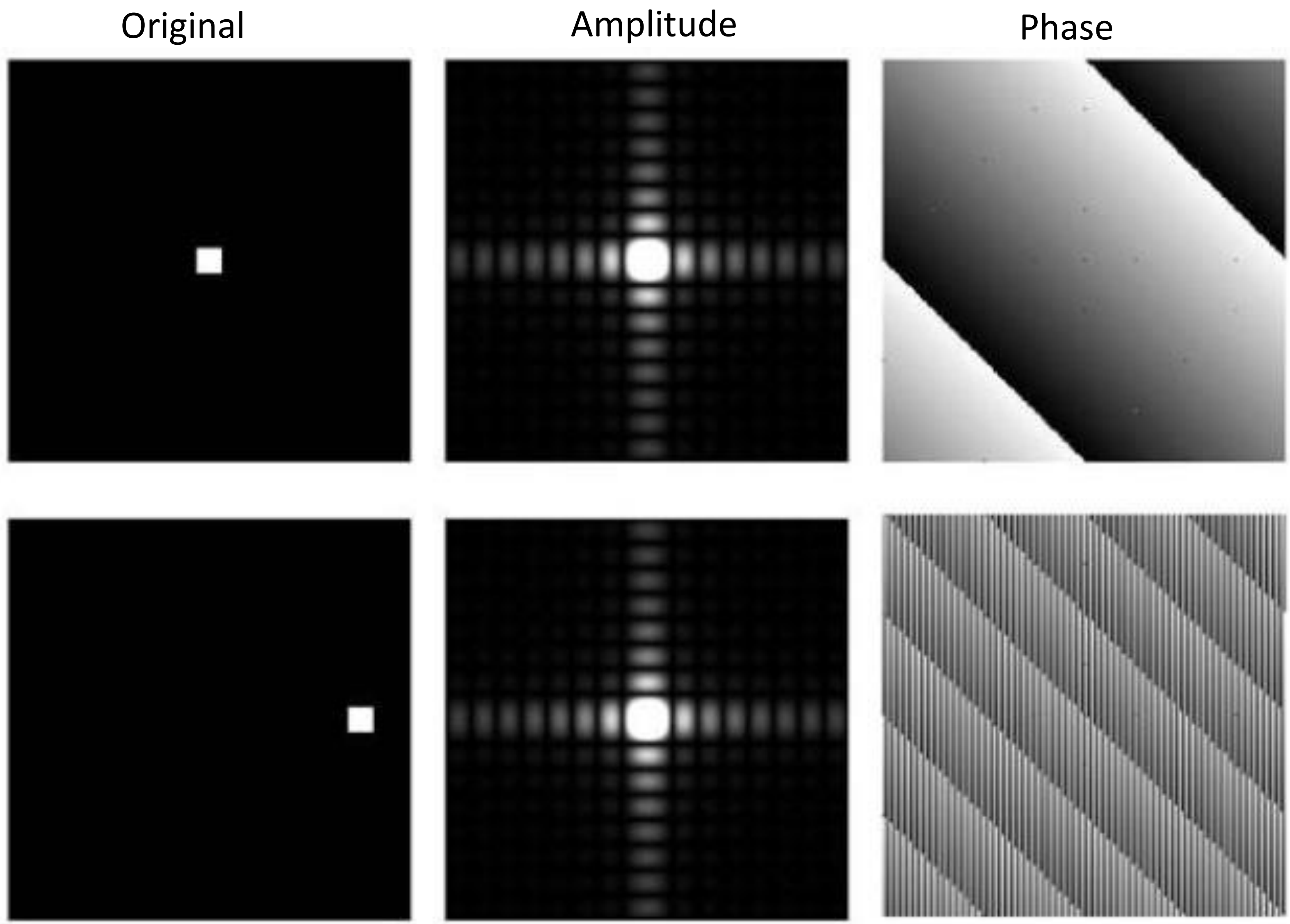
Discrete Inverse Fourier Transform: Visualization

$$\mathrm{h(x)} = \frac{1}{N}\sum_{k=0}^{N-1} H(k)e^{j\frac{2\pi kx}{N}}$$

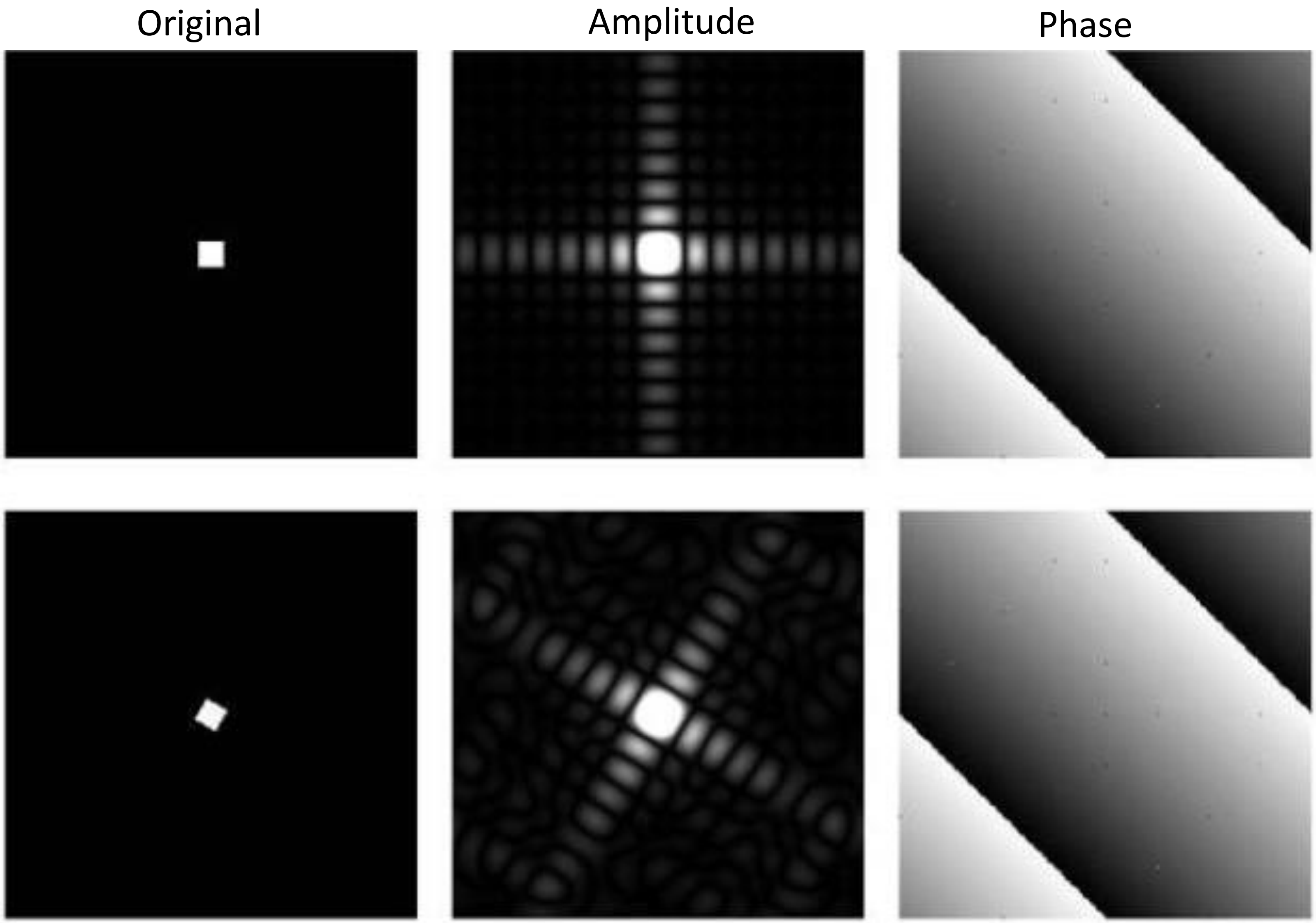


[from wikipedia]

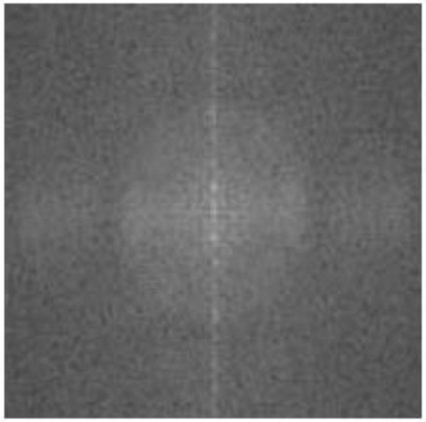
Example: Discrete 2D

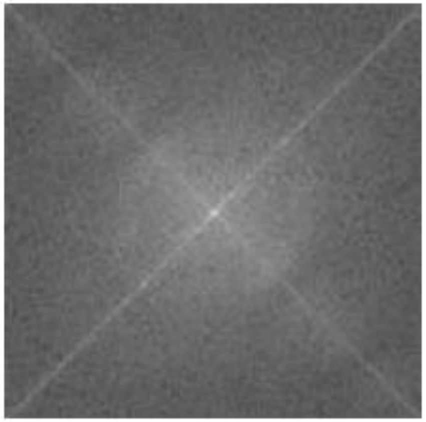


Example: Discrete 2D



Example: Discrete 2D





Fast Fourier Transformation

- Important property: $\mathcal{F}\left(g(x) * h(x)\right) = G(\omega)\, H(\omega)$
- Fast computation:

$$
\begin{tikzcd}
f(x), h(x) \arrow[rr, "\text{O(Nw)}", "\text{convolution}"'] \arrow[d, "\begin{array}{l}\text{O(N logN)}\\ \text{fourier transform}\end{array}"] \arrow[rr, bend right=80, "\textbf{O(N log N)}"'] & & f(x) * h(x) \\
F(\omega), H(\omega) \arrow[rr, "\text{O(N)}", "\text{multiplication}"'] & & F(\omega)\, H(\omega) \arrow[u, "\begin{array}{l}\text{O(N logN)}\\ \text{inverse fourier transform}\end{array}"']
\end{tikzcd}
$$

Roadmap: Basics Digital Image Processing

- Images
- Point operators (Ch. 3.1)
- Filtering: (ch. 3.2, ch 3.3, ch. 3.4) – main focus
  - Linear filtering
  - Non-linear filtering
- Fourier Transformation (ch. 3.4)
- Multi-scale image representation (ch. 3.5)
- Edges (Ch. 4.2)
  - Edge detection and linking
- Lines (Ch 4.3)
  - Line detection and vanishing point detection

Reading for next class

This lecture:

- Chapter 3 (in particular: 3.2, 3.3) - Basics of Digital Image Processing

Next lecture:

- Chapter 3.5: multi-scale representation
- Chapter 4.2 and 4.3 - Edge and Line detection
- Chapter 2 (in particular: 2.1, 2.2) – Image formation process
- And a bit of Hartley and Zisserman – chapter 2