


# Filter

## Industrielle Bildverarbeitung, Vorlesung No. 5<sup>1</sup>

M. O. Franz

07.11.2007

---

<sup>1</sup> falls nicht anders vermerkt, sind die Abbildungen entnommen aus Burger & Burge, 2005. 

# Übersicht

- 1 Lineare Filter
- 2 Formale Eigenschaften linearer Filter
- 3 Nichtlineare Filter

# Übersicht

- 1 **Lineare Filter**
- 2 Formale Eigenschaften linearer Filter
- 3 Nichtlineare Filter

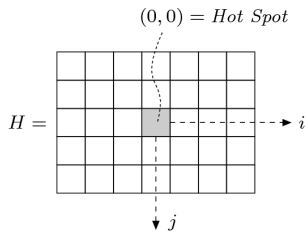
# Lineare Filter

**Lineare Filter:** Wert des Zielpixels wird als gewichtete Summe der Quellpixel berechnet.

Größe und Form der Filterregion und Gewichte des Filter werden durch eine Matrix von *Filterkoeffizienten* spezifiziert, der **Filtermatrix**  $H_{ij}$  oder **Filtermaske**, z.B.

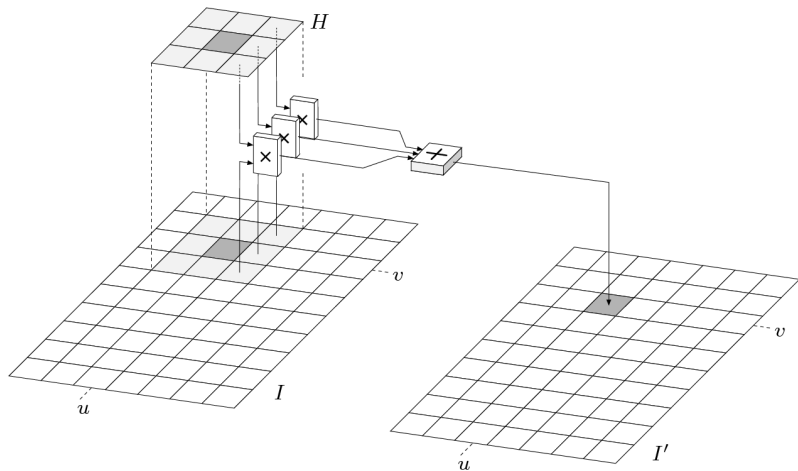
$$H(i, j) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Die Filtermatrix ist - wie ein Bild - eine diskrete zweidimensionale Funktion. Koordinaten werden meist relativ zum Zentrum angegeben ("hot spot").



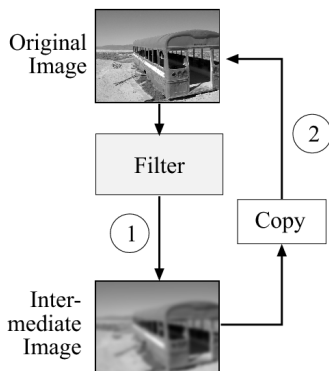
# Anwendung eines Filters

$$I'(u, v) = \sum_i \sum_j I(u+i, v+j)H(i, j)$$

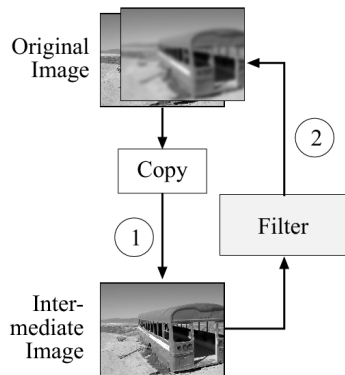


# Praktische Implementierung von Filteroperationen

Im Gegensatz zu Punktoperationen ist bei Filtern keine "in place"-Verarbeitung möglich, da die Quellpixel mehrere Male benötigt werden.



Variante A



Variante B

# Beispiel: $3 \times 3$ -Glättungsfilter mit unterschiedlichen Koeffizienten

Glockenförmiger  
Glättungsfilter:

$$H(i, j) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & \underline{0.200} & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix}$$

```

2      int w = orig.getWidth();
3      int h = orig.getHeight();
4      // 3 x 3 filter matrix
5      double[][] filter = {
6          {0.075, 0.125, 0.075},
7          {0.125, 0.200, 0.125},
8          {0.075, 0.125, 0.075}
9      };
10     ImageProcessor copy = orig.duplicate();
11
12     for (int v=1; v<=h-2; v++) {
13         for (int u=1; u<=w-2; u++) {
14             // compute filter result for position (u, v)
15             double sum = 0;
16             for (int j=-1; j<=1; j++) {
17                 for (int i=-1; i<=1; i++) {
18                     int p = copy.getPixel(u+i, v+j);
19                     // get the corresponding filter coefficient:
20                     double c = filter[j+1][i+1];
21                     sum = sum + c * p;
22                 }
23             }
24             int q = (int) Math.round(sum);
25             orig.putPixel(u, v, q);
26         }
27     }

```

# Ganzzahlige Filterkoeffizienten

Oft ist es vorteilhafter, mit ganzzahligen Filterkoeffizienten zu arbeiten:

- keine Umwandlung und Speicherung des Bildes in Gleitkommaformat notwendig
- auf manchen Rechnerarchitekturen sind Ganzzahloperationen schneller.
- auf FPGAs sind Gleitkommaoperationen extrem aufwendig.

Realisierung über einen Skalierungsfaktor, z.B.

$$H(i, j) = \begin{bmatrix} 0.075 & 0.125 & 0.075 \\ 0.125 & \underline{0.200} & 0.125 \\ 0.075 & 0.125 & 0.075 \end{bmatrix} = \frac{1}{40} \begin{bmatrix} 3 & 5 & 3 \\ 5 & \underline{8} & 5 \\ 3 & 5 & 3 \end{bmatrix}$$



# Beispiel: ganzzahliger $(2K + 1) \times (2L + 1)$ -Filter

Typisch:

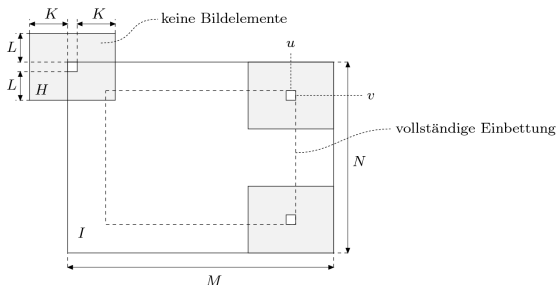
- ungeradzahlige Größe
- zentriert

```

5      // filter matrix of size (2K + 1) × (2L + 1)
6      int[][] filter = {
7          {0,0,1,1,1,0,0},
8          {0,1,1,1,1,0},
9          {1,1,1,1,1,1},
10         {0,1,1,1,1,0},
11         {0,0,1,1,1,0,0}
12     };
13     double s = 1.0/23; // sum of filter coefficients is 23
14
15     int K = filter[0].length/2;
16     int L = filter.length/2;
17
18     ImageProcessor copy = orig.duplicate();
19
20     for (int v=L; v<=N-L-1; v++) {
21         for (int u=K; u<=M-K-1; u++) {
22             // compute filter result for position (u, v)
23             int sum = 0;
24             for (int j=-L; j<=L; j++) {
25                 for (int i=-K; i<=K; i++) {
26                     int p = copy.getPixel(u+i,v+j);
27                     int c = filter[j+L][i+K];
28                     sum = sum + c * p;
29                 }
30             }
31             int q = (int) Math.round(s * sum);
32             if (q < 0) q = 0;
33             if (q > 255) q = 255;
34             orig.putPixel(u,v,q);
35         }
36     }

```

# Behandlung von Randproblemen bei Filtern

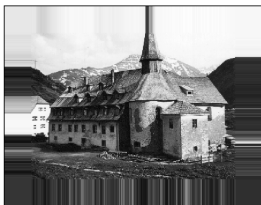


- nur Zentralbereich auswerten, bei dem die Filtermaske ganz ins Bild passt  $\Rightarrow$  Outputbild wird kleiner.
- Zero padding: Inputbild wird um 0 erweitert  $\Rightarrow$  In- und Outputbild gleich groß.
- Gespiegelte Randbedingungen
- Konstante Randbedingungen

# Beispiel: Randbedingungen



(a)

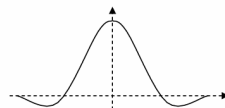
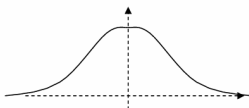
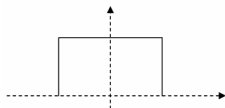
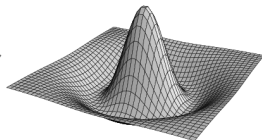
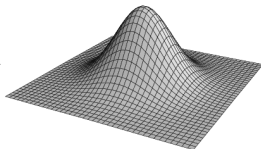
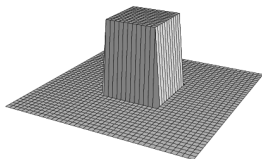


(b)



(c)

# Beispiele für lineare Filter



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

(a)

0	1	2	1	0
1	3	5	3	1
2	5	9	5	2
1	3	5	3	1
0	1	2	1	0

(b)

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

(c)

# Übersicht

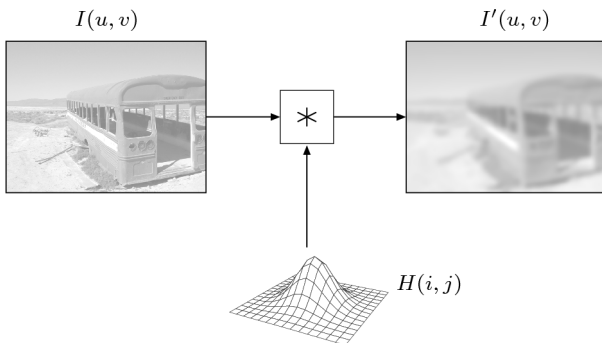
- 1 Lineare Filter
- 2 Formale Eigenschaften linearer Filter**
- 3 Nichtlineare Filter

# Lineare Faltung

Für diskrete 2-dimensionale Funktionen  $I$  und  $H$

$$I'(u, v) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u-i, v-j) \cdot H(i, j)$$

kurz:  $I' = I * H$



# Faltung und Korrelation

Bisherige Notation für Inputbild  $I$  und Filterkern  $H$ : **lineare Korrelation**

$$I'(u, v) = \sum_{i=-\lfloor K/2 \rfloor}^{\lfloor K/2 \rfloor} \sum_{j=-\lfloor L/2 \rfloor}^{\lfloor L/2 \rfloor} I(u+i, v+j) \cdot H(i, j)$$

Definition Faltung (mit  $R$ : Bereich von  $H$  mit  $H(i, j) \neq 0$ ):

$$\begin{aligned} I'(u, v) &= \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(u-i, v-j) \cdot H(i, j) = \sum_{i, j \in R} I(u-i, v-j) \cdot H(i, j) \\ &= \sum_{i, j \in R} I(u+i, v+j) \cdot H(-i, -j) \end{aligned}$$

d.h. Faltung entspricht Korrelation mit **gespiegelter Filtermatrix**.

# Eigenschaften der Faltung

- Kommutativität:

$$I * H = H * I$$

- Linearität:

$$(a \cdot I) * H = I * (a \cdot H) = a \cdot (I * H)$$

$$(I_1 + I_2) * H = I_1 * H + I_2 * H$$

aber:

$$(b + I) * H \neq b + I * H$$

- Assoziativität:

$$A * (B * C) = (A * B) * C$$



# Separable Filter

Aus der Assoziativität ergibt sich, daß ein großer Filter  $H$  in mehrere kleine Filter  $H_i$  zerlegt werden kann:

$$I * H = I * (H_1 * H_2 * \dots) = (\dots ((I * H_1) * H_2) * \dots)$$

Insbesondere bei 2 eindimensionalen Filtern in x- und y-Richtung (**x/y-Separabilität**), z.B

$$H_x = [ 1 \quad 1 \quad 1 \quad 1 \quad 1 ] \quad \text{bzw.} \quad H_y = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

ergibt sich

$$I' = (I * H_x) * H_y = I * (H_x * H_y) = I * H_{xy}$$

z.B.  $H_{xy} = H_x * H_y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$  d.h. 8 statt 15 Operationen

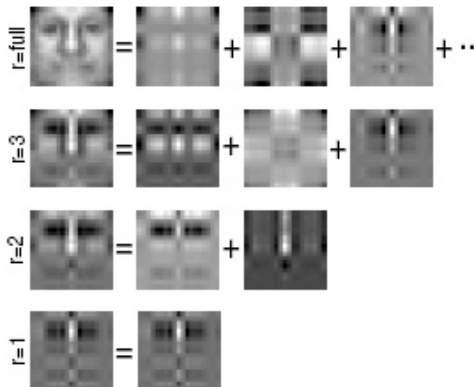
# Beispiele für separable Filter

Zweidimensionale Filter sind separabel, wenn sie als **äußeres Produkt** geschrieben werden können:

$$H_{xy}(i, j) = H_x(i) \cdot H_y(j)$$

Beispiel: Gaußfilter

$$\begin{aligned} G_{xy}(x, y) &= e^{-\frac{x^2+y^2}{2\sigma^2}} \\ &= e^{-\frac{x^2}{2\sigma^2}} \cdot e^{-\frac{y^2}{2\sigma^2}} \\ &= G_x(x) \cdot G_y(y) \end{aligned}$$



[Kienzle et al., 2002]

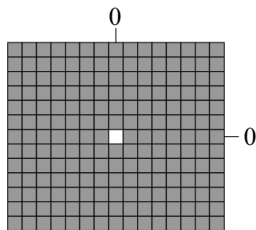
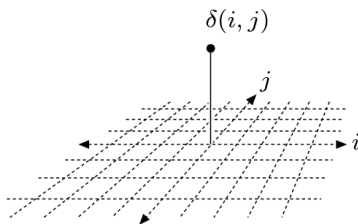
# Dirac-Funktion

Die **Impuls-** oder **Dirac-Funktion**  $\delta$  ist das neutrale Element der Faltung

$$I * \delta = I$$

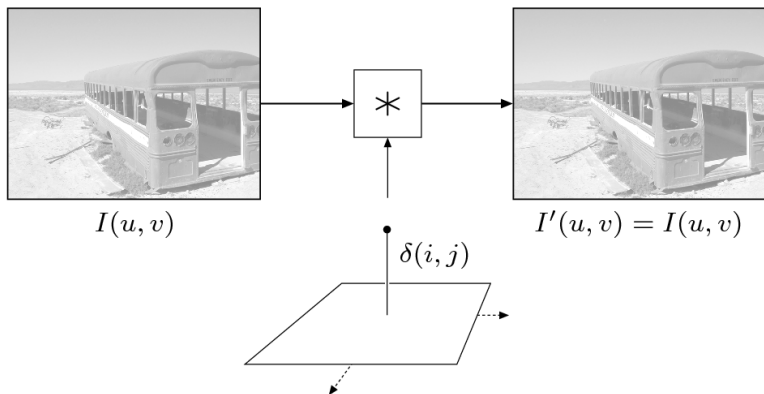
Definition (zweidimensional, diskret):

$$\delta(i, j) = \begin{cases} 1 & \text{für } i = j = 0 \\ 0 & \text{sonst.} \end{cases}$$



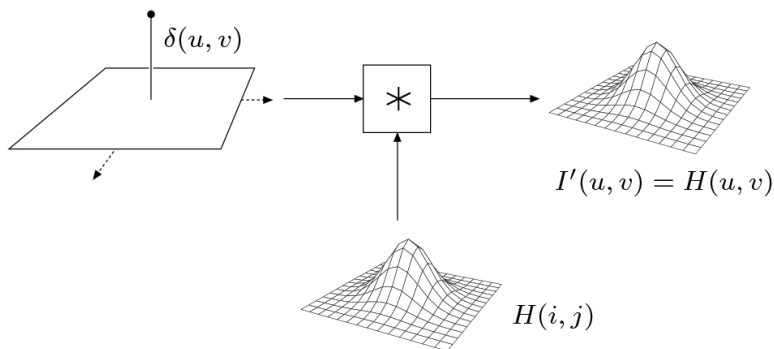
# Dirac-Funktion als neutrales Element der Faltung

Die Faltung mit der Impulsfunktion ergibt wieder das ursprüngliche Bild.



# Impulsantwort

Die Impulsfunktion als Input eines linearen Filters liefert die Filterfunktion  $H$  als Ergebnis.



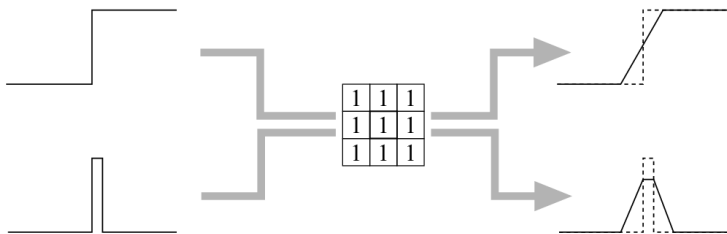
Ein Impuls charakterisiert ein lineares System vollständig!

# Übersicht

- 1 Lineare Filter
- 2 Formale Eigenschaften linearer Filter
- 3 Nichtlineare Filter**

# Rauschunterdrückung mit linearen Filtern

Lineare Glättungsfilter reduzieren zwar das Rauschen im Bild, aber gleichzeitig werden Kanten oder Linien verbreitert und im Kontrast reduziert.



# Nichtlineare Filter

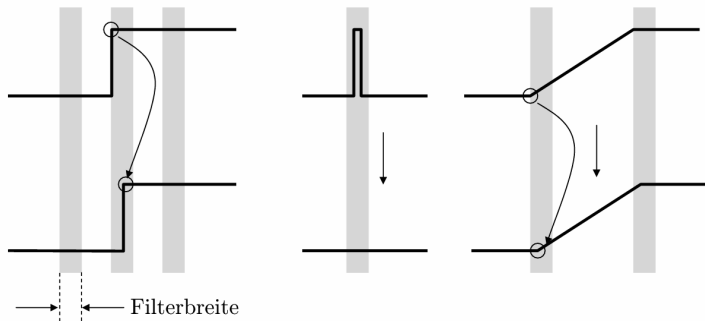
Nichtlineare Filter werden so wie lineare Filter über eine Umgebung  $R$  des Zielpixels mit einer nichtlinearen Funktion  $f : \mathbb{R} \rightarrow \mathbb{R}$  berechnet:

$$I'(u, v) = f(I(u, v), I(u + 1, v), I(u - 1, v), I(u, v + 1), \dots)$$

z.B. **Minimum-** und **Maximumfilter:**

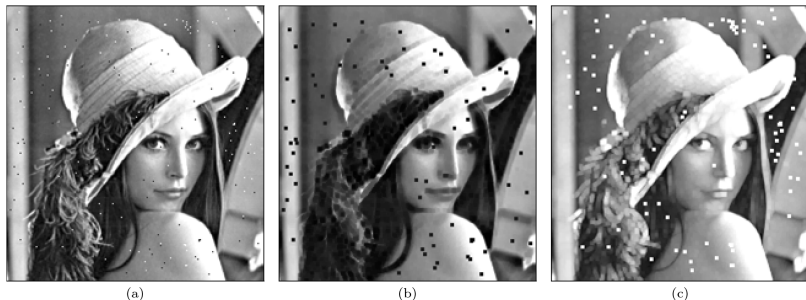
$$I'(u, v) = \min\{I(u + i, v + j) \mid i, j \in R\}$$

$$I'(u, v) = \max\{I(u + i, v + j) \mid i, j \in R\}$$





# Beispiel: Minimum- und Maximumfilter auf Salt-and-Pepper-Rauschen



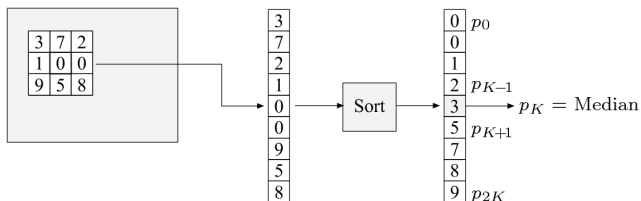
Minimumfilter eliminiert weiße Punkte und verbreitert dunkle Regionen, Maximumfilter macht das Gegenteil.

# Medianfilter

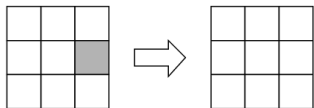
Der Medianfilter ersetzt jeden Pixel durch den **Median** seiner Umgebung  $R$ . Bei  $2k + 1$  aufsteigend sortierten Pixeln ist der Median definiert als

$$\text{median}\{p_0, p_1, \dots, p_k, \dots, p_{2k}\} = p_k,$$

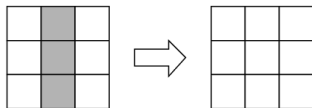
bei  $2k$  aufsteigend sortierten Pixeln  $\{p_0, \dots, p_{2k-1}\}$  als  $(p_{k-1} + p_k)/2$ .



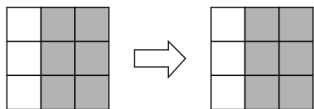
# Beispiel: $3 \times 3$ -Medianfilter



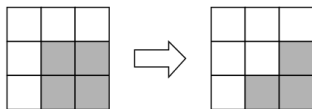
(a)



(b)



(c)



(d)

# Vergleich linearer Glättungs- und Medianfilter



(a)



(b)



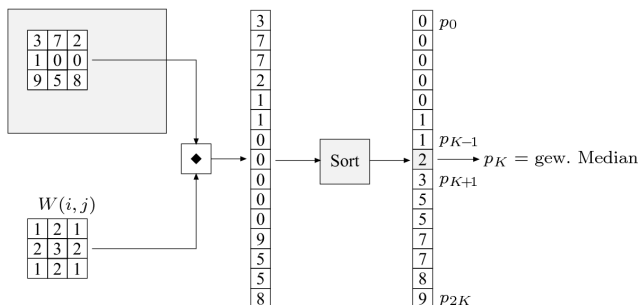
(c)

- Der lineare Filter dämpft das Rauschen, macht aber das Bild unscharf.
- Der Medianfilter eliminiert Spitzen/Höhen, erzeugt örtlich Flecken mit konstanter Intensität.

# Gewichteter Medianfilter

Grundidee: Wert wird in der sortierten Liste so oft wiederholt, wie sein Gewicht ist, d.h. die Liste wird dadurch  $L = \sum_{i,j \in R} W(i,j)$  lang.

Beispiel:



# Weitere nichtlineare Filter...

- Morphologische Filter
- Interest-Point-Detektoren
- Volterra-Filter:

$$\begin{aligned}y(t) = & h^{(0)} + \int_{\mathbb{R}} h^{(1)}(\tau_1)x(t - \tau_1) d\tau_1 \\ & + \int_{\mathbb{R}^2} h^{(2)}(\tau_1, \tau_2)x(t - \tau_1)x(t - \tau_2) d\tau_1 d\tau_2 \\ & + \int_{\mathbb{R}^3} h^{(3)}(\tau_1, \tau_2, \tau_3)x(t - \tau_1)x(t - \tau_2)x(t - \tau_3) d\tau_1 d\tau_2 d\tau_3 \\ & + \dots\end{aligned}$$