Operators Through Convolutions
#1 Smoothing

Cyrill Stachniss

Photogrammetry & Robotics Lab

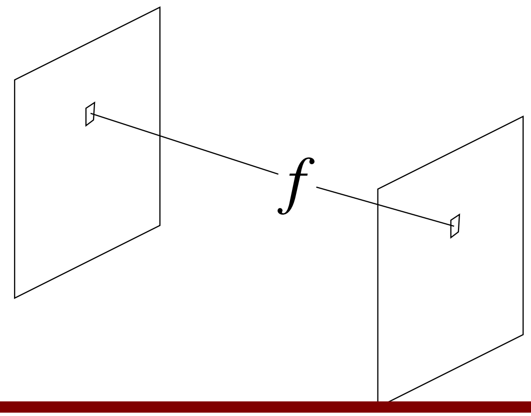
Local Operators Through Convolutions – Part 1 Smoothing Filters

Cyrill Stachniss

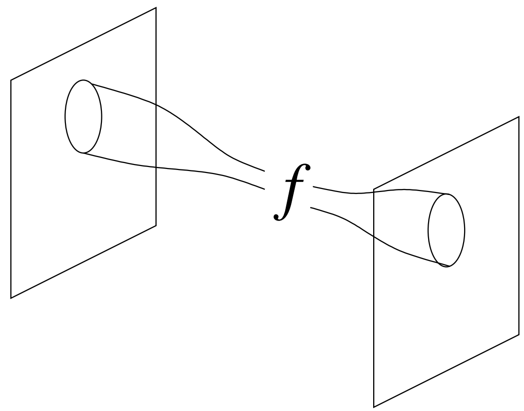
The slides have been created by Cyrill Stachniss.

Three Types of Operators

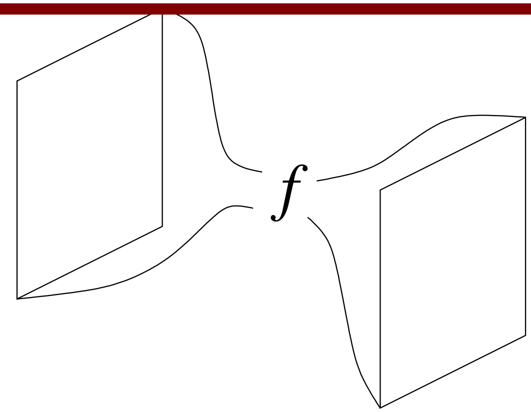
- Point operator



- Local operator



- Global operator



Point Operators

- Cannot deal well with noise
- Cannot deal well with local structures ("one intensity value is not enough")

Today: Local Operators

- **Local** operators are also called **neighborhood** operators
- Convolutions as a framework for specifying such local operators
- We will look in several local operators
  - Noise reduction
  - Binomial filter
  - Gradients
  - …

Box Filter

"Replace an intensity value by the mean intensity value of the neighborhood."

$$g(i) = \frac{1}{K} \sum_k f(i-k)$$

$$g(i,j) = \frac{1}{KL} \sum_{k,l} f(i-k, j-l)$$

DE: "Rechteckfilter (oder gleitende Mittelbildung)"

Box Filter Example

"Replace an intensity value by the mean intensity value of the neighborhood."

$$g(i) = \frac{1}{K} \sum_k f(i-k)$$

$f(i)$ $\qquad k \in \{-1, 0, 1\}$ $\qquad g(i)$

100
126
110
97
99

Box Filter Example

"Replace an intensity value by the mean intensity value of the neighborhood."

$$g(i) = \frac{1}{K} \sum_k f(i-k)$$

$f(i)$ $\qquad$ $k \in \{-1, 0, 1\}$ $\qquad$ $g(i)$

$f(i)$	$g(i)$
100	?
126	112
110	111
97	102
99	?

2D Box Filter Example

$$g(i,j) = \frac{1}{9} \sum_{k=\{-1,0,1\}} \sum_{l=\{-1,0,1\}} f(i-k, j-l)$$

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

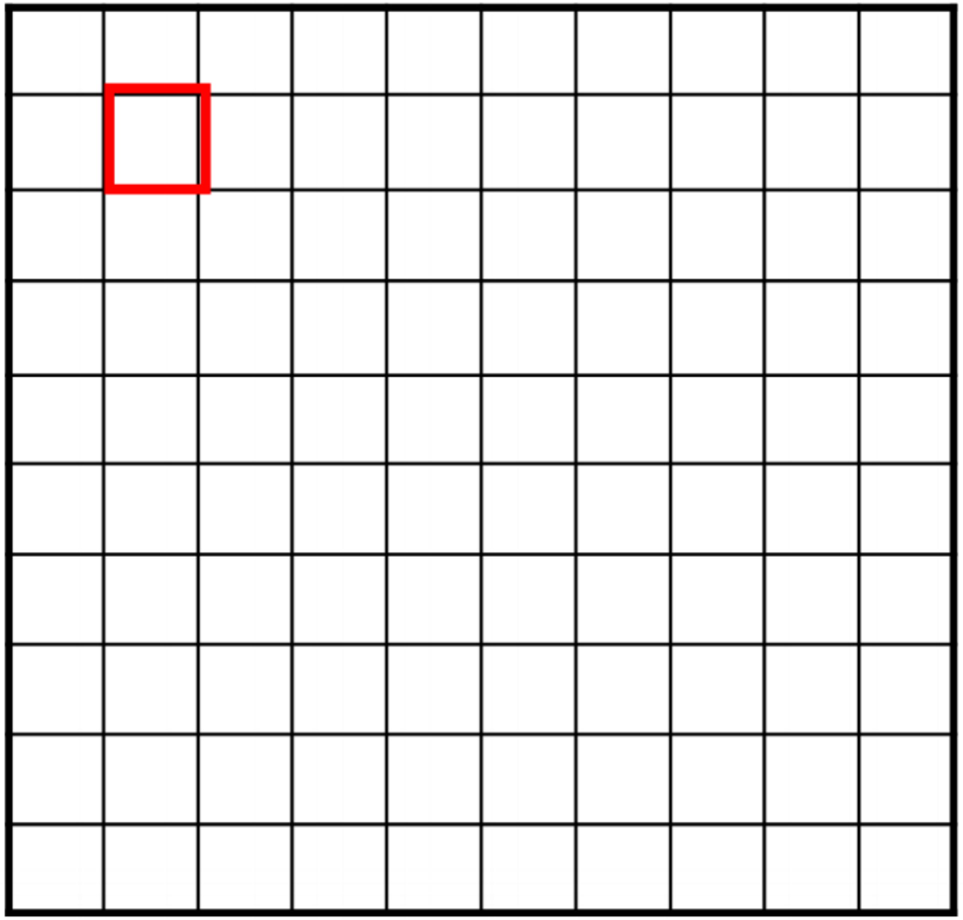


Image courtesy: Seitz

2D Box Filter Example

$$g(i,j) = \frac{1}{9} \sum_{k=\{-1,0,1\}} \sum_{l=\{-1,0,1\}} f(i-k, j-l)$$

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							

Image courtesy: Seitz

2D Box Filter Example

$$g(i,j) = \frac{1}{9} \sum_{k=\{-1,0,1\}} \sum_{l=\{-1,0,1\}} f(i-k, j-l)$$

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						

Image courtesy: Seitz

2D Box Filter Example

$$g(i,j) = \frac{1}{9} \sum_{k=\{-1,0,1\}} \sum_{l=\{-1,0,1\}} f(i-k, j-l)$$

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	30					

Image courtesy: Seitz

2D Box Filter Example

$$g(i,j) = \frac{1}{9} \sum_{k=\{-1,0,1\}} \sum_{l=\{-1,0,1\}} f(i-k, j-l)$$

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	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	

Image courtesy: Seitz

Kernel

- We can formulate the box filter by using a weighting function w

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

- This weighting function is called **kernel** (or kernel function)
- Often, these filtering operators involve **weighted combinations** of intensity values in a neighborhood

Linear Shift Invariant Filters

- A filter L that transforms

$$g(i,j) = L(f(i,j))$$

- is called **linear and shift invariant** if

$$L(\alpha_1 f_1 + \alpha_2 f_2) = \alpha_1 g_1 + \alpha_2 g_2$$

- and

$$L(f(i-k,j-l)) = g(i-k,j-l)$$

Convolution (DE: Faltung)

- Filters of the form

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

- are **convolutions** of the function f and a kernel function w

$$g = w * f$$

Box Filter as a Convolution

- The **box filter** ($\boldsymbol{R}$) is a **convolution**

$$g = \boldsymbol{R}_3^{(2)} * f$$

dimensionality

neighborhood

- of the image function f and a box kernel

(1-dim)

$$\boldsymbol{R}_3^{(1)} = \frac{1}{3}\begin{bmatrix} 1 \\ \underline{1} \\ 1 \end{bmatrix}$$

(2-dim)

$$\boldsymbol{R}_3^{(2)} = \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

element with index zero

1D Box Filter (Neighborhood 3)

$$g(i) \quad = \quad \sum_{k=-1}^{k=1} w(k)\, f(i-k)$$

1D Box Filter (Neighborhood 3)

$$\begin{aligned} g(i) &= \sum_{k=-1}^{k=1} w(k)\, f(i-k) \\ &= w(-1)f(i-(-1)) + w(0)f(i-0) + w(1)f(i-1) \end{aligned}$$

1D Box Filter (Neighborhood 3)

$$
\begin{aligned}
g(i) &= \sum_{k=-1}^{k=1} w(k)\, f(i-k) \\
&= w(-1) f(i-(-1)) + w(0) f(i-0) + w(1) f(i-1) \\
&= \frac{1}{3} f(i+1) + \frac{1}{3} f(i) + \frac{1}{3} f(i-1)
\end{aligned}
$$

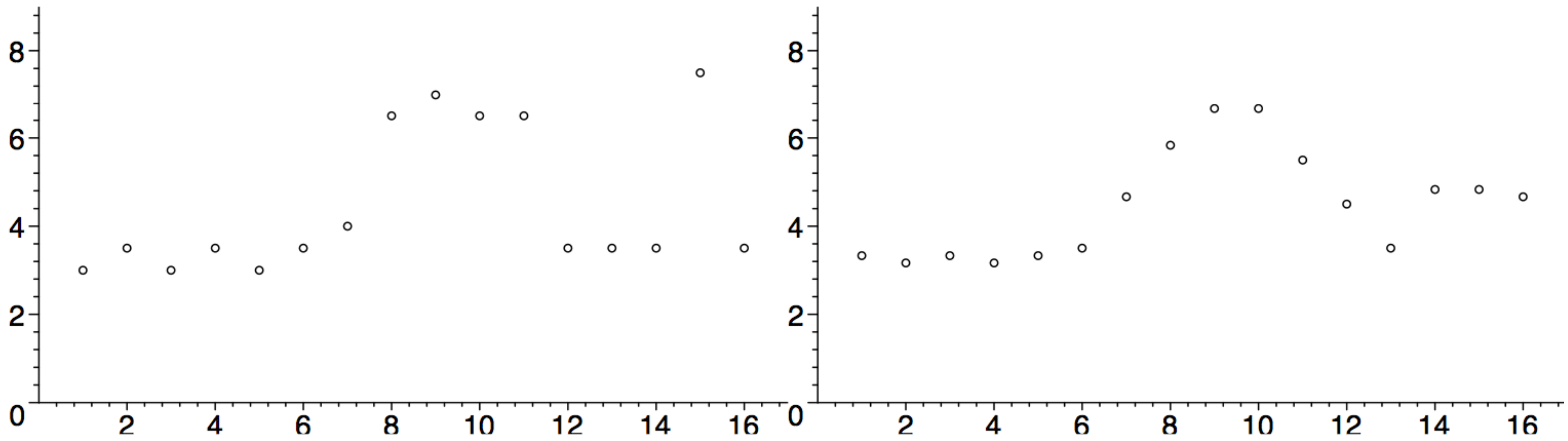
we set $w(i) = \dfrac{1}{3}$

1D Box Filter (Neighborhood 3)

$$
\begin{aligned}
g(i) &= \sum_{k=-1}^{k=1} w(k)\, f(i-k) \\
&= w(-1)f(i-(-1)) + w(0)f(i-0) + w(1)f(i-1) \\
&= \frac{1}{3}f(i+1) + \frac{1}{3}f(i) + \frac{1}{3}f(i-1) \\
&= \frac{1}{3}(f(i+1) + f(i) + f(i-1))
\end{aligned}
$$

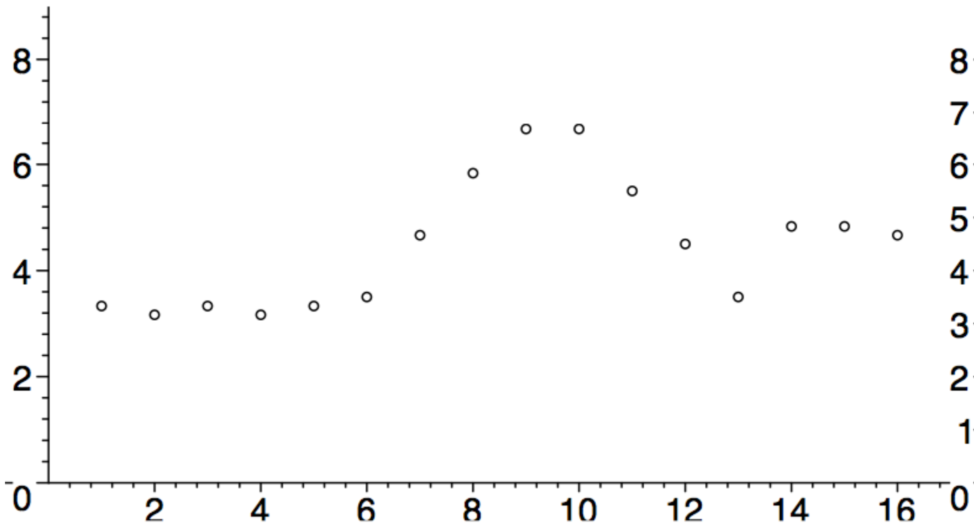
The box filter takes the average of the neighborhood pixels (here 3) and can be expressed as a convolution

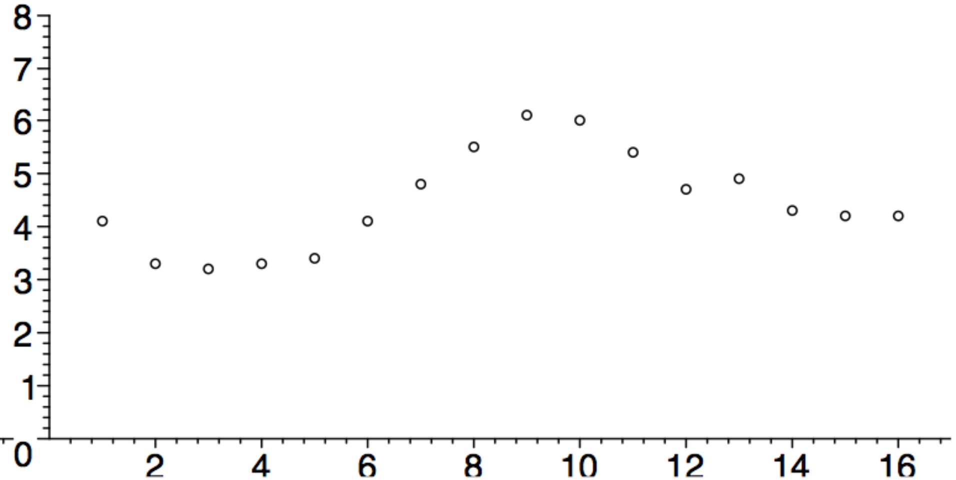
Box Filter Example



$$f \qquad\qquad g = \boldsymbol{R}_3^{(1)} * f$$

Box Filter Example Using Different Neighborhoods





$$g = \boldsymbol{R}_3^{(1)} * f$$

$$\boldsymbol{R}_3^{(1)} = \frac{1}{3}\begin{bmatrix} 1 \\ \underline{1} \\ 1 \end{bmatrix}$$

$$g = \boldsymbol{R}_5^{(1)} * f$$

$$\boldsymbol{R}_5^{(1)} = \frac{1}{5}\begin{bmatrix} 1 \\ 1 \\ \underline{1} \\ 1 \\ 1 \end{bmatrix}$$

Box Kernel

- The sum of all weights of the kernel yields 1. As a result, the mean of the image function does not change

$$\boldsymbol{R}_3^{(2)} = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & \underline{1} & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

- The underlined number is the reference pixel with index (0,0)

Noise Reduction

- Convolutions with such a kernel changes the noise of the input signal
- For the box filter, we have

$$\sigma_{n_g}\left(\boldsymbol{R}_m^{(1)}\right) = \frac{1}{\sqrt{m}}\sigma_{n_f} \qquad \sigma_{n_g}\left(\boldsymbol{R}_m^{(2)}\right) = \frac{1}{m}\sigma_{n_f}$$

- The box filter reduces the noise
- In general, we have

$$\sigma_{n_g}^2 = \sum_i (w(i))^2 \;\; \sigma_{n_f}^2$$

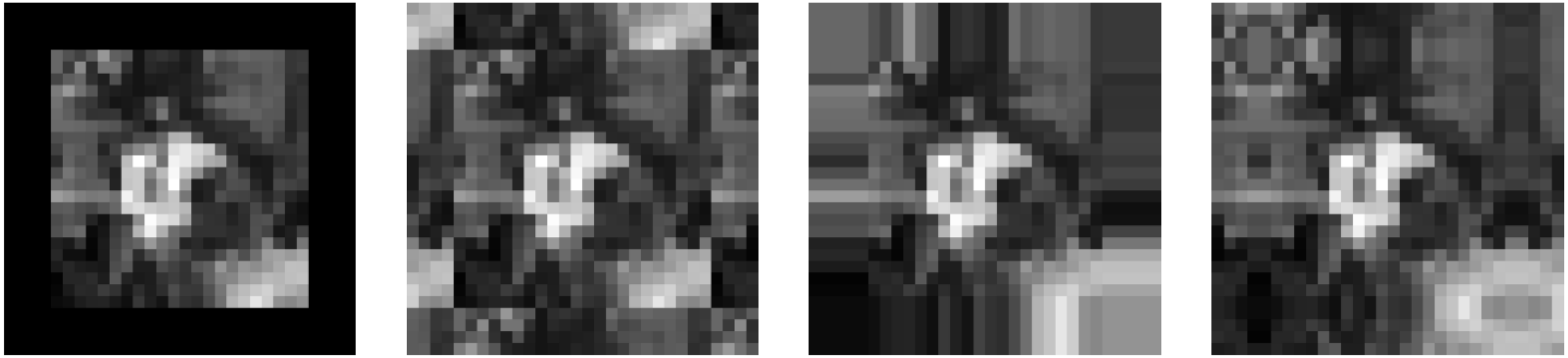
Median Filter

- As an alternative to the box filter, we can compute the median within the neighborhood
- Robust against outliers
- Not a linear filter anymore

How to Deal with the Borders?

- A pixel at the border of an image has no fully defined neighborhood
- **Padding** options
  - **constant value:** for all outside pixels
  - **cyclic wrap:** loop “around” the image
  - **clamp:** repeat edge pixels indefinitely
  - **mirror:** reflect pixels across the image edge

Padding Options



- constant zero: 0 for all outside pixels
- cyclic wrap: loop “around” the image
- clamp: repeat edge pixels indefinitely
- mirror: reflect pixels across the image edge

Image courtesy: Szelinsky

Binomial Filter

DE: “Binomial-Filter”

Binomial Filter

- Performs a smoothing
- Smoothing using an approximation of a Gaussian as the kernel function
- Discrete approximation due to pixels
- The decay of the weights approximates a Gaussian using the coefficients of a Binomial distribution $B(0.5, n)$
- Elements of the Pascal's triangle

Binomial Kernel in 1D

$$\boldsymbol{B}_2^{(1)} = \frac{1}{4}\begin{bmatrix} 1 \\ \underline{2} \\ 1 \end{bmatrix}$$

$$\boldsymbol{B}_4^{(1)} = \frac{1}{16}\begin{bmatrix} 1 \\ 4 \\ \underline{6} \\ 4 \\ 1 \end{bmatrix}$$

Connection to Pascal's Triangle

$$
\begin{array}{ccccccccccc}
 & & & & & 1 & & & & & \\
 & & & & 1 & & 1 & & & & \\
 & & & 1 & & 2 & & 1 & & & \\
 & & 1 & & 3 & & 3 & & 1 & & \\
 & 1 & & 4 & & 6 & & 4 & & 1 & \\
1 & & 5 & & 10 & & 10 & & 5 & & 1
\end{array}
$$

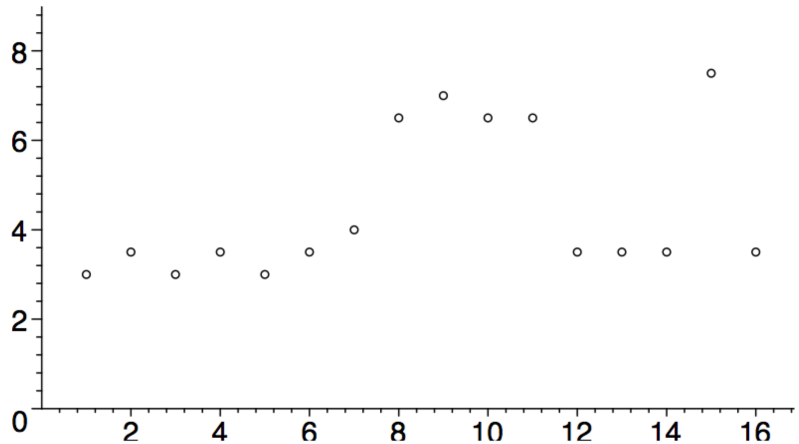
$$\left(\boldsymbol{B}_1^{(1)}\right)^{\top} = \frac{1}{2}[1, 1]$$

$$\left(\boldsymbol{B}_2^{(1)}\right)^{\top} = \frac{1}{4}[1, 2, 1]$$

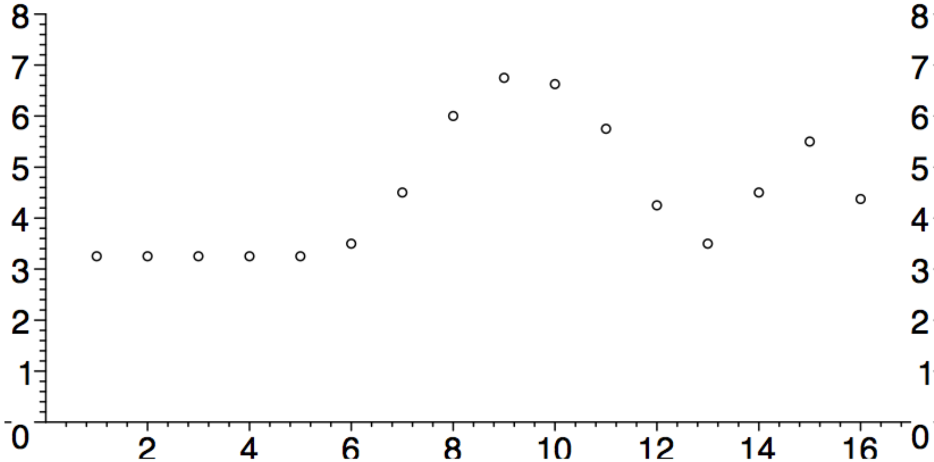
$$\vdots$$

$$\left(\boldsymbol{B}_5^{(1)}\right)^{\top} = \frac{1}{32}[1, 5, 10, 10, 5, 1]$$

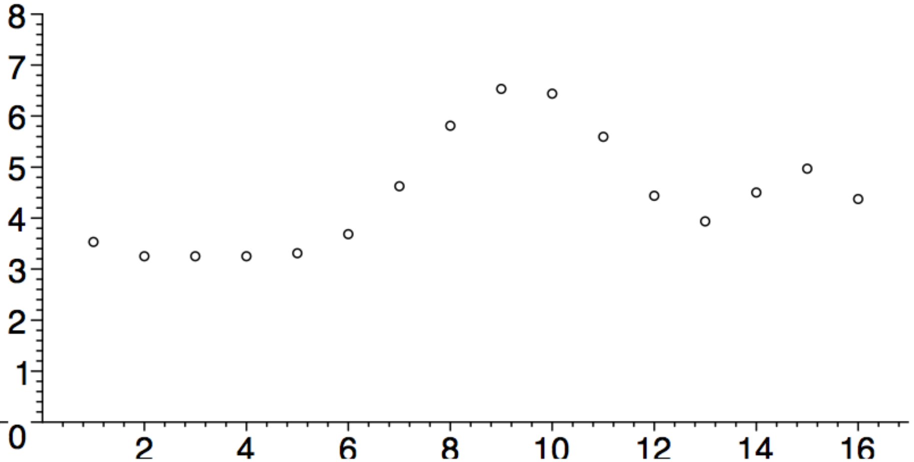
1D Example



$$f$$



$$g = \boldsymbol{B}_2^{(1)} * f$$



$$g = \boldsymbol{B}_4^{(1)} * f$$

Binomial Kernel in 1D and 2D

$$\boldsymbol{B}_2^{(1)} = \frac{1}{4}\begin{bmatrix} 1 \\ \underline{2} \\ 1 \end{bmatrix} \qquad \boldsymbol{B}_2^{(2)} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & \underline{4} & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$\boldsymbol{B}_4^{(1)} = \frac{1}{16}\begin{bmatrix} 1 \\ 4 \\ \underline{6} \\ 4 \\ 1 \end{bmatrix} \qquad \boldsymbol{B}_4^{(2)} = \frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & \underline{36} & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$$

2D Example

f

$g = \boldsymbol{B}_{30}^{(2)} * f$

Noise Reduction

- For the Binomial filter, we obtain for the input-output noise relation

$$\sigma_{n_g}\left(\boldsymbol{B}_m^{(1)}\right) = \frac{1}{\sqrt[4]{\pi m}}\sigma_{n_f} \qquad \sigma_{n_g}\left(\boldsymbol{B}_m^{(2)}\right) = \frac{1}{\sqrt{\pi m}}\sigma_{n_f}$$

- Similar to the box filter, also the Binomial filter reduces the noise
- Less aggressive smoothing than the box filter (for the same neighborhood)

Degree of Smoothing

- **Degree of smoothing** $r = \dfrac{\sigma_{n_g}}{\sigma_{n_f}}$

- We obtain for the kernel size

$$m_R(r) = \frac{1}{r} \qquad m_B(r) = \frac{1}{\pi r^2}$$

- Example for r=1/5:

$$m_R(0.2) = 5 \qquad m_B(0.2) = \frac{25}{\pi} \approx 8$$

- Box filter offers a stronger smoothing than the Binomial filter

Convolution

DE: “Faltung”

Convolution

- We have seen that **linear filters** can be expressed as **convolutions**
- Therefore, we now analyze the **properties of the convolution**

$$g = w * f$$

output image | kernel | input image

Definition

- The discrete convolution of the functions $a(i)$ and $b(i)$ is defined as

$$c(i) = \sum_{k=-\infty}^{+\infty} a(k)\, b(i-k)$$

- and in 2D as

$$c(i,j) = \sum_{k=-\infty}^{+\infty} \sum_{l=-\infty}^{+\infty} a(k,l)\, b(i-k, j-l)$$

- and is written as $\boldsymbol{c} = \boldsymbol{a} * \boldsymbol{b}$

Commutative Property

- The convolution is **commutative**

$$\boldsymbol{a} * \boldsymbol{b} = \boldsymbol{b} * \boldsymbol{a}$$

- because

$$c(i) = \sum_{k=-\infty}^{+\infty} a(k)\, b(i-k)$$

- equals – after replacement $\begin{aligned} j &= i - k \\ k &= i - j \end{aligned}$

$$c(i) = \sum_{j=-\infty}^{+\infty} a(i-j)\, b(j)$$

Further Properties

- **Associative** property

$$\boldsymbol{a} * \boldsymbol{b} * \boldsymbol{c} = (\boldsymbol{a} * \boldsymbol{b}) * \boldsymbol{c} = \boldsymbol{a} * (\boldsymbol{b} * \boldsymbol{c})$$

- **Distributive** property

$$(\boldsymbol{a} + \boldsymbol{b}) * \boldsymbol{c} = \boldsymbol{a} * \boldsymbol{c} + \boldsymbol{b} * \boldsymbol{c}$$

- **Scalar** multiplication

$$\lambda(\boldsymbol{a} * \boldsymbol{b}) = (\lambda \boldsymbol{a}) * \boldsymbol{b} = \boldsymbol{a} * (\lambda \boldsymbol{b})$$

Neutral Element / Unit Impulse

- Unit impulse

$$\delta = \begin{bmatrix} \vdots \\ 0 \\ \underline{1} \\ 0 \\ \vdots \end{bmatrix} \qquad \boldsymbol{\delta} = \begin{bmatrix} \vdots\vdots\vdots & \vdots & \vdots\vdots\vdots \\ \cdots & \underline{1} & \cdots \\ \vdots\vdots\vdots & \vdots & \vdots\vdots\vdots \end{bmatrix}$$

- The convolution with $\boldsymbol{\delta}$ yields the input function, i.e., $\boldsymbol{a} * \boldsymbol{\delta} = \boldsymbol{a}$

Translation/Shift Through Convolution

- A convolution with δ can be used to shift the function f by (x, y)

$$f(i-x, j-y) = f(i,j) * \delta(i-x, j-y)$$

$$\boldsymbol{\delta} = \begin{bmatrix} \vdots\vdots\vdots & \vdots & \vdots\vdots\vdots \\ \cdots & \underline{1} & \cdots \\ \vdots\vdots\vdots & \vdots & \vdots\vdots\vdots \end{bmatrix}$$

Multiplication and Convolution

- Multiplication and convolution can be seen as similar
- Difference: Multiplication uses carry (DE: Übertrag)
- Example:

$$
\begin{array}{r}
1\,2\,1 * 1\,2\,1 \\
\hline
1\,2\,1 \\
2\,4\,2 \\
1\,2\,1 \\
\hline
\underline{1}\,4\,6\,4\,\underline{1}
\end{array}
\qquad
\frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} * \frac{1}{4}\begin{bmatrix}1\\2\\1\end{bmatrix} = \frac{1}{4}\frac{1}{4}\left(\begin{bmatrix}1\\2\\1\end{bmatrix} * \begin{bmatrix}1\\2\\1\end{bmatrix}\right) = \frac{1}{16}\begin{bmatrix}1\\4\\6\\4\\1\end{bmatrix}
$$

corresponds to extended signal

De-Convolution

- If the inverse $a^{-1}(i)$ of $a(i)$ exists and

  $\sum_i a(i) \neq 0$

  we can de-convolute a function

- Given $c(x) = a(x) * b(x)$
  we can recover $b(i)$ given $a^{-1}(i)$ by

$$b(x) = a^{-1}(x) * c(x) = c(x) * a^{-1}(x)$$

Separable Kernels

- A multi-dimensional kernel that can be split into the individual dimensions, is called **separable**.
- Example:

$$\boldsymbol{B}_2^{(2)} = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \frac{1}{4}\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \frac{1}{4}[1\ 2\ 1] = \boldsymbol{B}_2^{(1)} * \left(\boldsymbol{B}_2^{(1)}\right)^{\mathsf{T}}$$

Separable Kernels Allow for Efficient Computations

Two 1D convolutions are more efficient to compute than one 2D convolution

$$\boldsymbol{g} = \underline{\boldsymbol{R}_n^{(2)} * \boldsymbol{f}} = \underline{\boldsymbol{R}_n^{(1)} * \left((\boldsymbol{R}_n^{(1)})^\mathsf{T} * \boldsymbol{f} \right)}$$

$\mathcal{O}(n^2)$ operations $\quad$ $\mathcal{O}(2\,n) = \mathcal{O}(n)$ operations

Multiple Convolutions

- Smoothing filters have the property

$$\sum_i w(i) = 1$$

- The concatenation of smoothing filters is again a smoothing filter

$$\boldsymbol{w} = \boldsymbol{w}_1 * \ldots * \boldsymbol{w}_n$$

- For the Binomial filter holds:

$$\boldsymbol{B}_n = \underbrace{\boldsymbol{B}_1 * \ldots * \boldsymbol{B}_1}_{n \text{ times}}$$

Integral Image

- An integrate image is an image in which each pixel stores a sum of intensity values of the form

$$s(i,j) = \sum_{i'=1}^{i} \sum_{j'=1}^{j} f(i',j')$$

- The integral image can used to efficiently execute a box filter

Integral Image

- Allow for effective computing the sum over intensities in any rectangle

$$
\begin{aligned}
S([i_0, i_1] \times [j_0, j_1]) \quad &= \quad s(i_1, j_1) + s(i_0 - 1, j_0 - 1) \\
&\quad -s(i_0 - 1, j_1) - s(i_1, j_0 - 1)
\end{aligned}
$$

3	2	7	2	3
1	5	1	3	4
5	1	**3**	5	1
4	3	2	1	6
2	4	1	4	8

3	5	12	*14*	17
4	11	19	24	31
9	17	28	38	46
13	24	37	**48**	62
15	30	44	59	81

$$S = 48 + 3 - 14 - 13 = 24$$

Summary

- Linear filters as local operators
- Convolution as a framework
- Introduction of important filters
- Box filter
- Binomial/Gaussian filter
- There are several other operators

Literature

- Szeliski, Computer Vision: Algorithms and Applications, Chapter 3
- Förstner, Scriptum Photogrammetrie I, Chapter “Lokale Operatoren”

Slide Information

- The slides have been created by Cyrill Stachniss as part of the photogrammetry and robotics courses.
- **I tried to acknowledge all people from whom I used images or videos. In case I made a mistake or missed someone, please let me know.**
- The photogrammetry material heavily relies on the very well written lecture notes by Wolfgang Förstner and the Photogrammetric Computer Vision book by Förstner & Wrobel.
- Parts of the robotics material stems from the great Probabilistic Robotics book by Thrun, Burgard and Fox.
- If you are a university lecturer, feel free to use the course material. If you adapt the course material, please make sure that you keep the acknowledgements to others and please acknowledge me as well. To satisfy my own curiosity, please send me email notice if you use my slides.

Cyrill Stachniss, cyrill.stachniss@igg.uni-bonn.de