Video Analysis

LV 188.329, WS 2016/17

Filtering

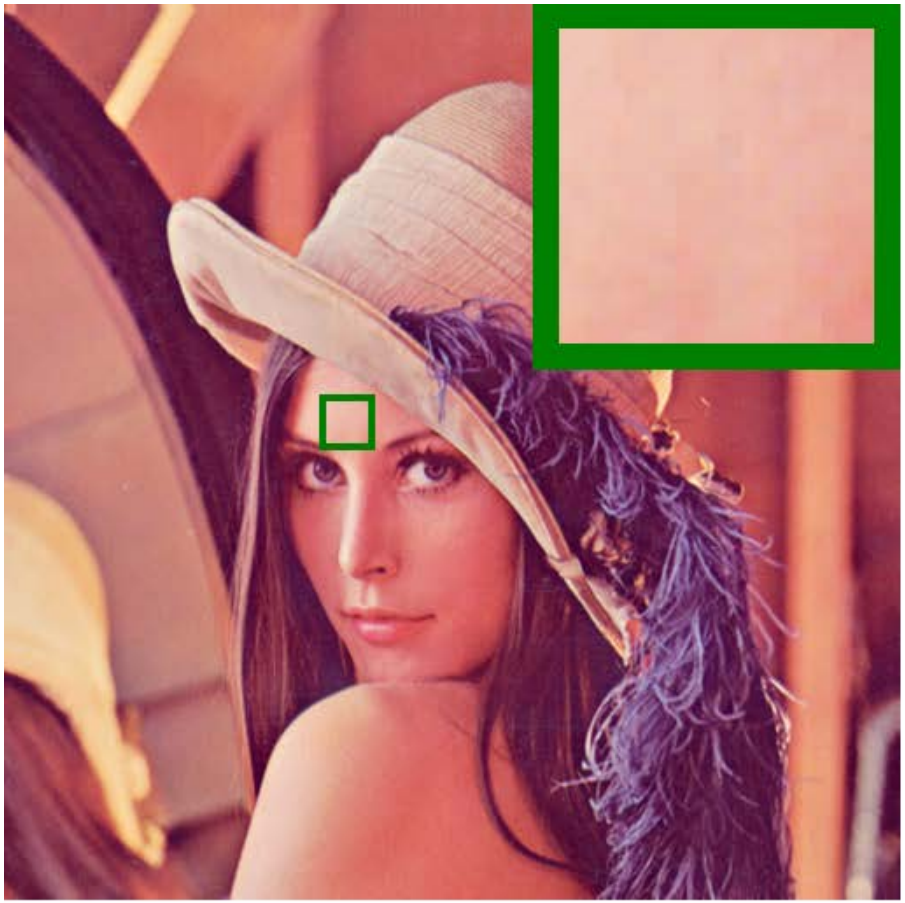
Ao. Univ.-Prof. Margrit Gelautz

margrit.gelautz@tuwien.ac.at

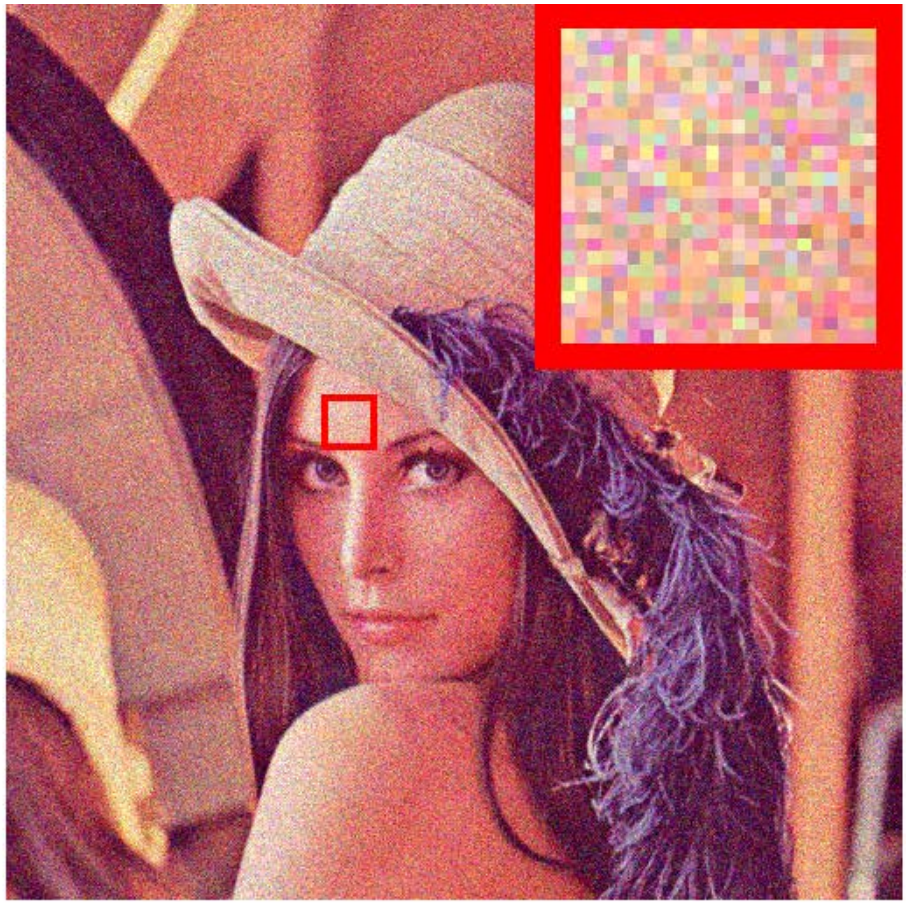
FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

Motivation (1)

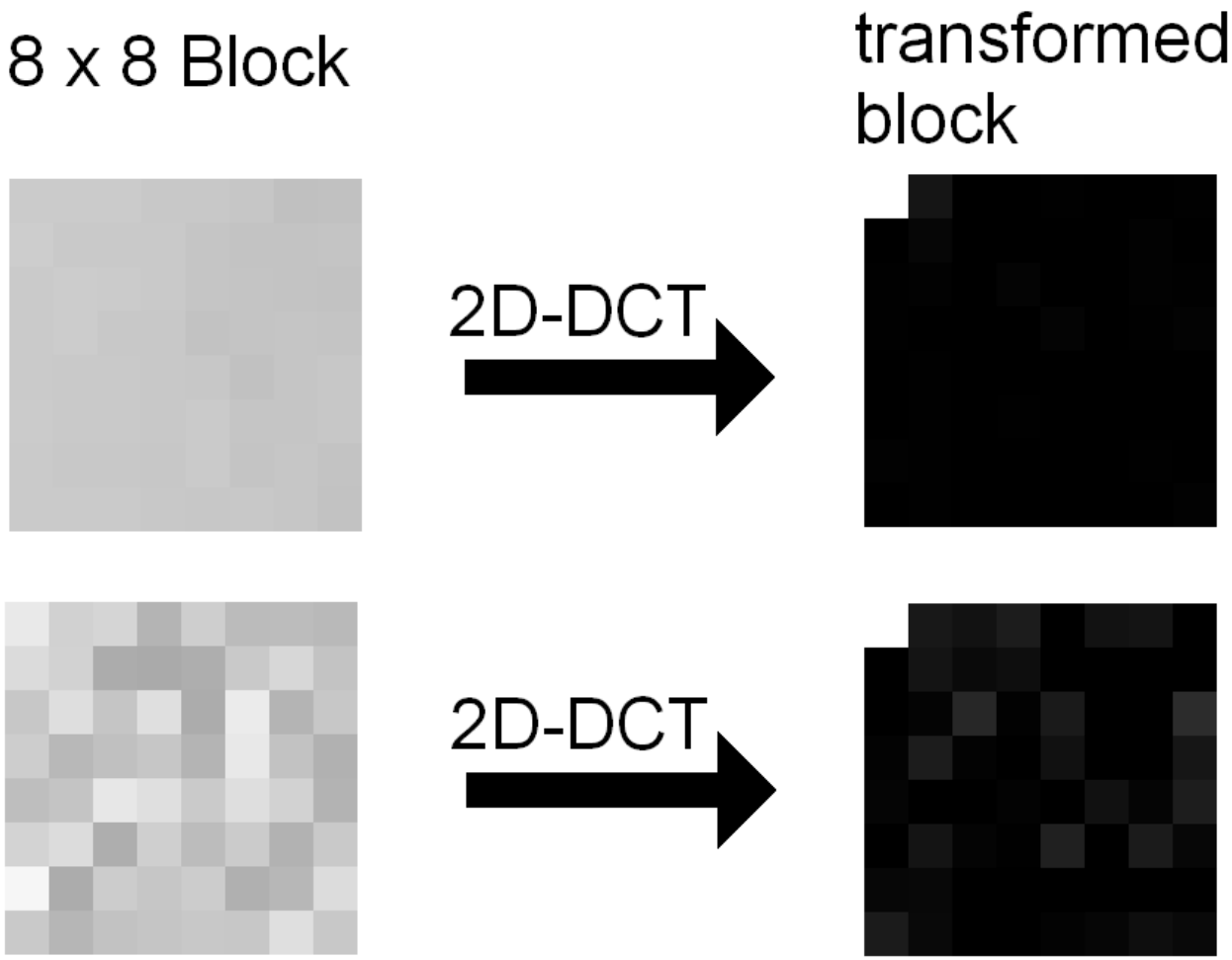


almost no noise

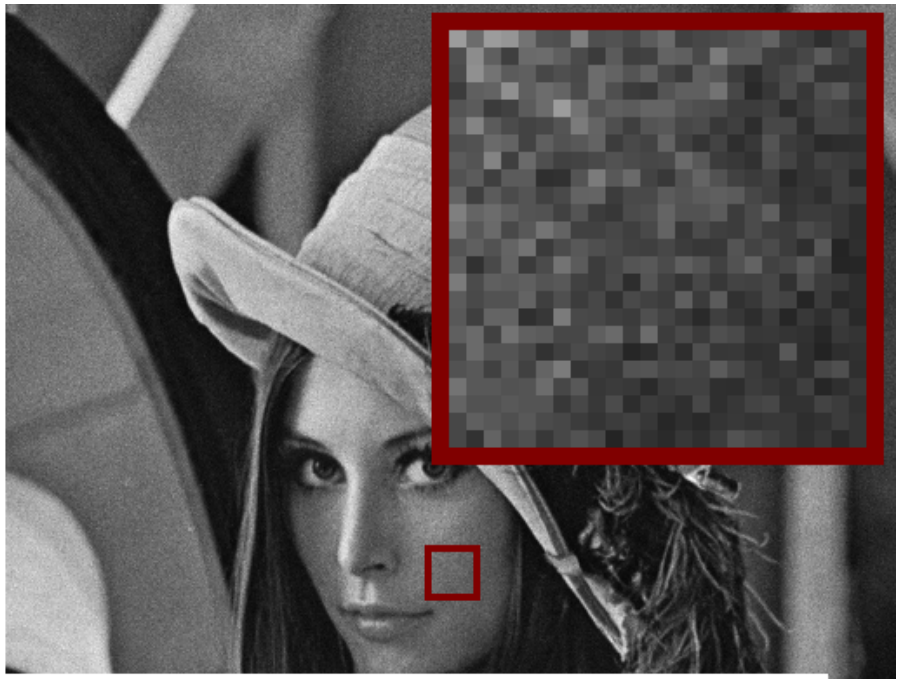


same image with strong noise component

Motivation (2)

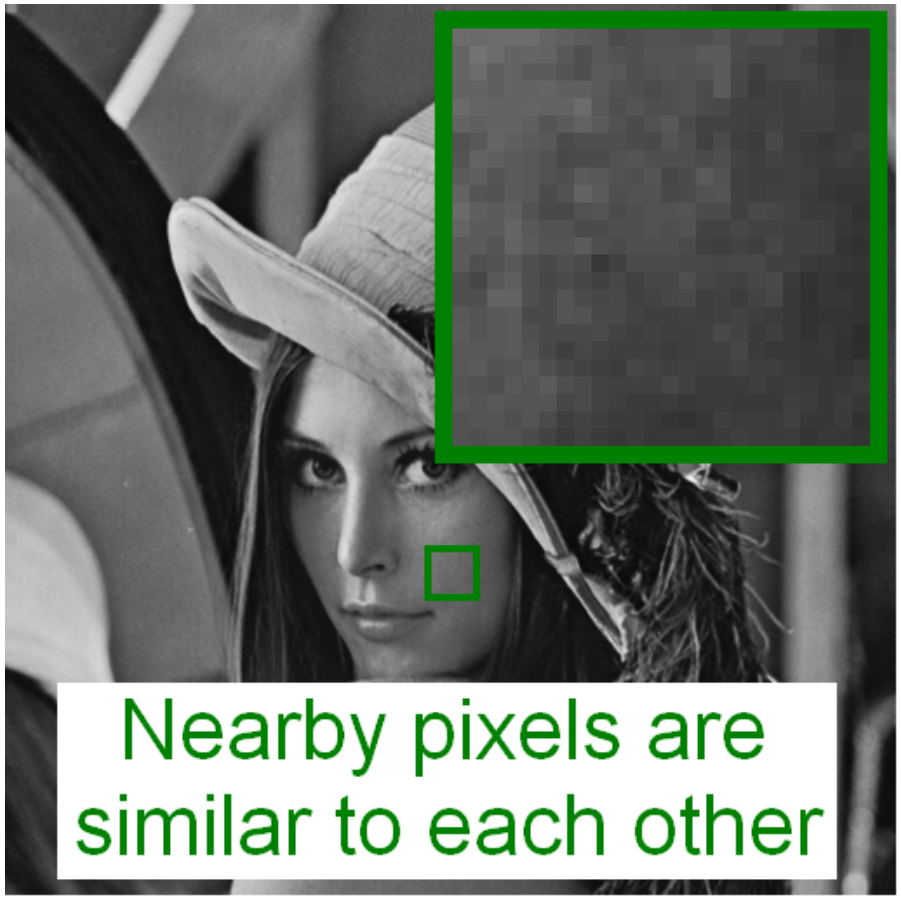


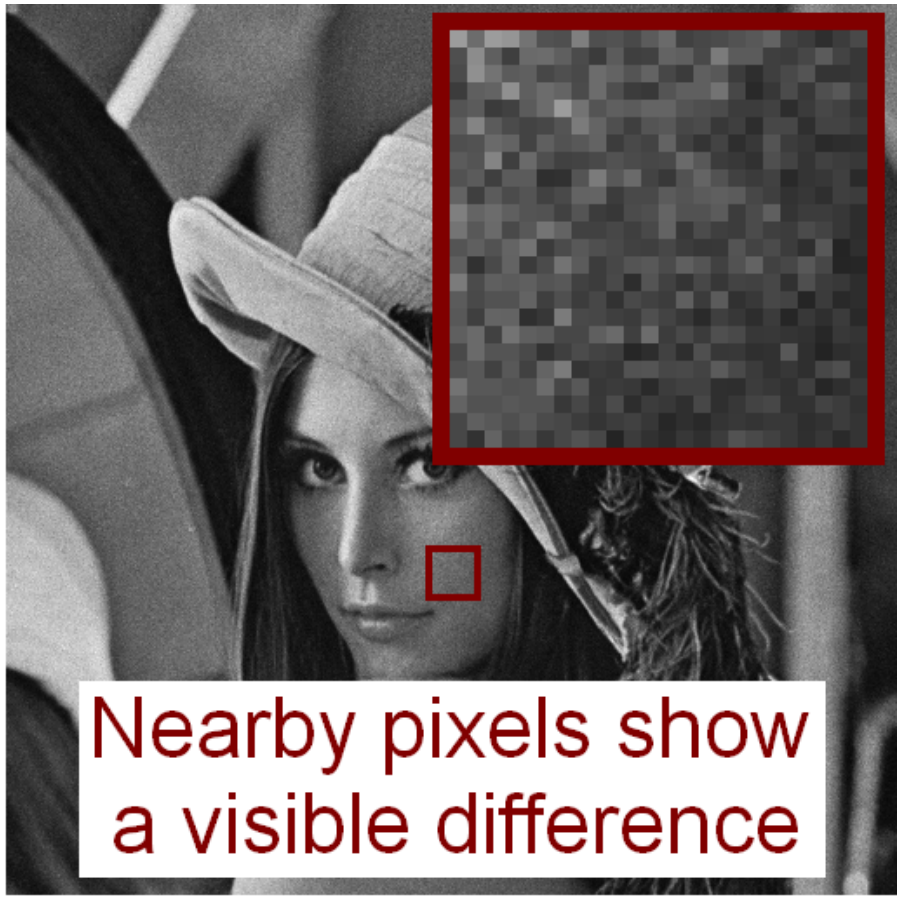
Intuitive approach (1)



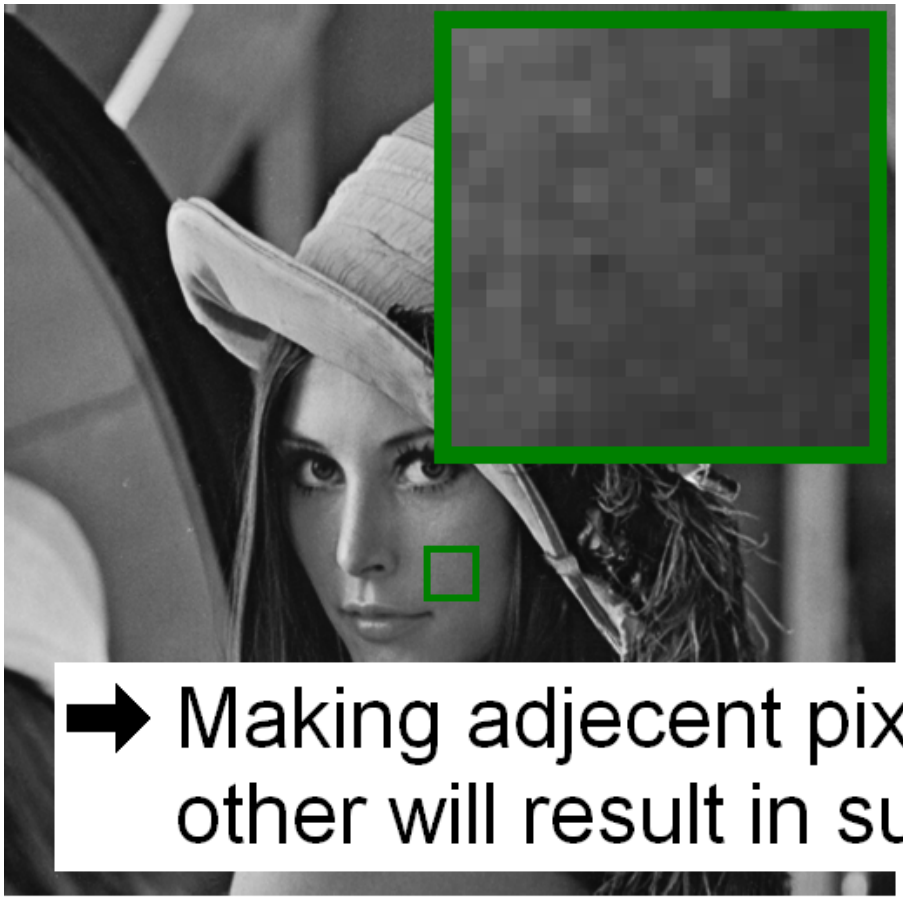
What is the difference between the two magnified image regions?

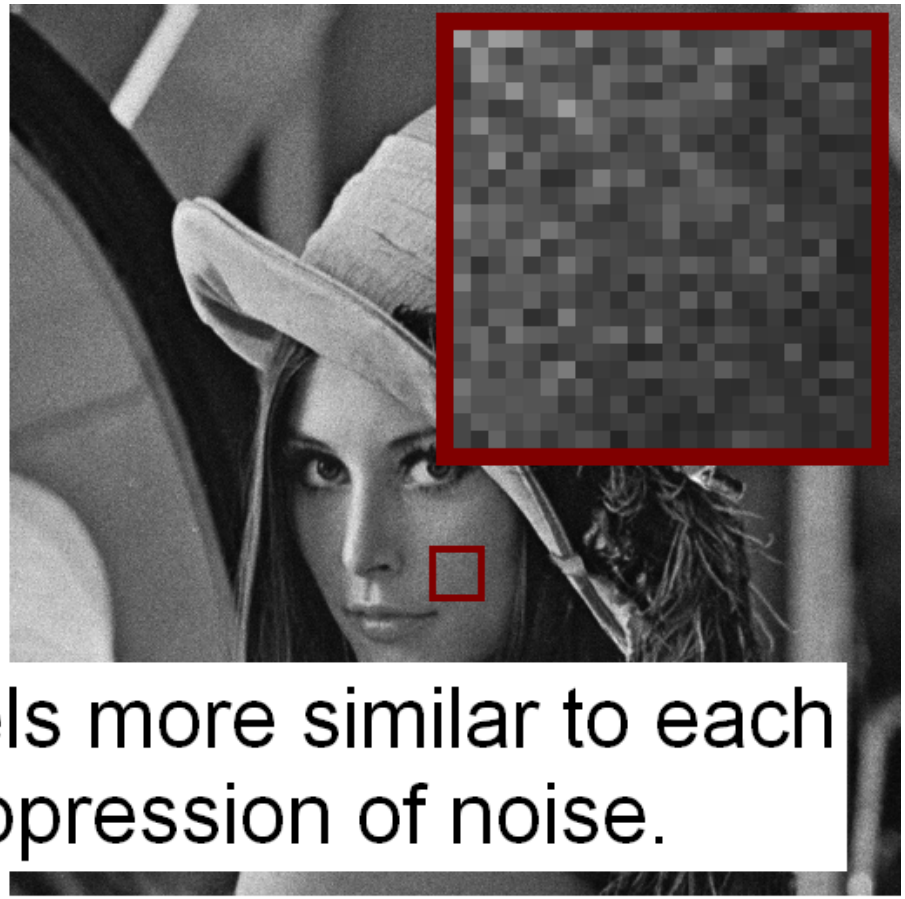
Intuitive approach (2)





Intuitive approach (3)

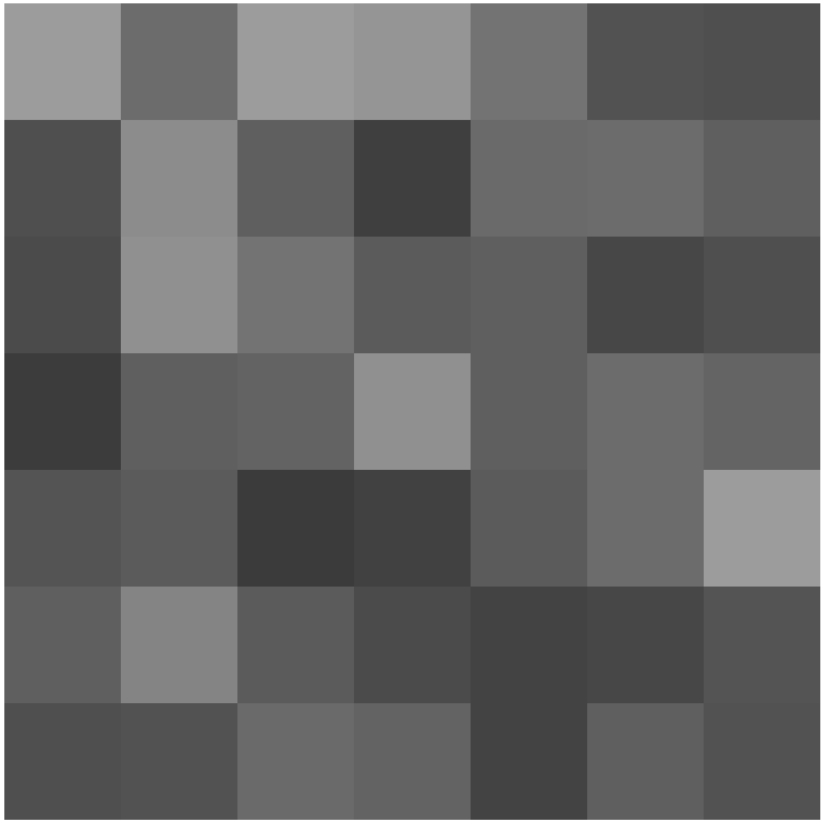




➡ Making adjecent pixels more similar to each other will result in suppression of noise.

Averaging filter (1)

The averaging filter calculates the sum of all pixels in the filter window, then divides the calculated sum by the number of pixels in the filter window.



156	108	156	149	115	82	79
79	140	95	63	106	108	95
75	144	115	91	95	71	79
60	95	99	144	95	108	100
84	91	59	65	91	108	156
95	132	91	75	67	71	84
79	82	106	99	67	95	82

Averaging filter (2)

156	108	156	149	115	82	79
79	140	95	63	106	108	95
75	144	115	91	95	71	79
60	95	99	144	95	108	100
84	91	59	65	91	108	156
95	132	91	75	67	71	84
79	82	106	99	67	95	82

		?				

79 + 140 + 95 ... 75 + 67 = 2346 / 25 = 93.84

Averaging filter (3)

156	108	156	149	115	82	79
79	140	95	63	106	108	95
75	144	115	91	95	71	79
60	95	99	144	95	108	100
84	91	59	65	91	108	156
95	132	91	75	67	71	84
79	82	106	99	67	95	82

		94	?			

140 + 95 + 63 ... 67 + 71 = 2419 / 25 = 96.76

Averaging filter (4)

156	108	156	149	115	82	79
79	140	95	63	106	108	95
75	144	115	91	95	71	79
60	95	99	144	95	108	100
84	91	59	65			
95	132	91	75			
79	82	106	99	67	95	82

		94	97			

After filtering the two pixels are very similar to each other

144 – 99 = 45

97 – 94 = 3

Averaging filter (5)

156	108	156	149	115	82	79
79	140	95	63	106	108	95
75	144	115	91	95	71	79
60	95	99	144	95	108	100
84	91	59	65	91	108	156
95	132	91	75	67	71	84
79	82	106	99	67	95	82

		94	97			

The reason for this is that the kernels overlap each other.

Averaging filter (6)

156	108	156	149	115	82	79
79	140	95	63	106	108	95
75	144	115	91	95	71	79
60	95	99	144	95	108	100
84	91	59	65	91	108	156
95	132	91	75	67	71	84
79	82	106	99	67	95	82

20 out of 25 remain unchanged.

80%

156	108	156	149	115	82	79
79	140	95	63	106	108	95
75	144	115	91	95	71	79
60	95	99	144	95	108	100
84	91	59	65	91	108	156
95	132	91	75	67	71	84
79	82	106	99	67	95	82

6 out of 9 remain unchanged.

67%

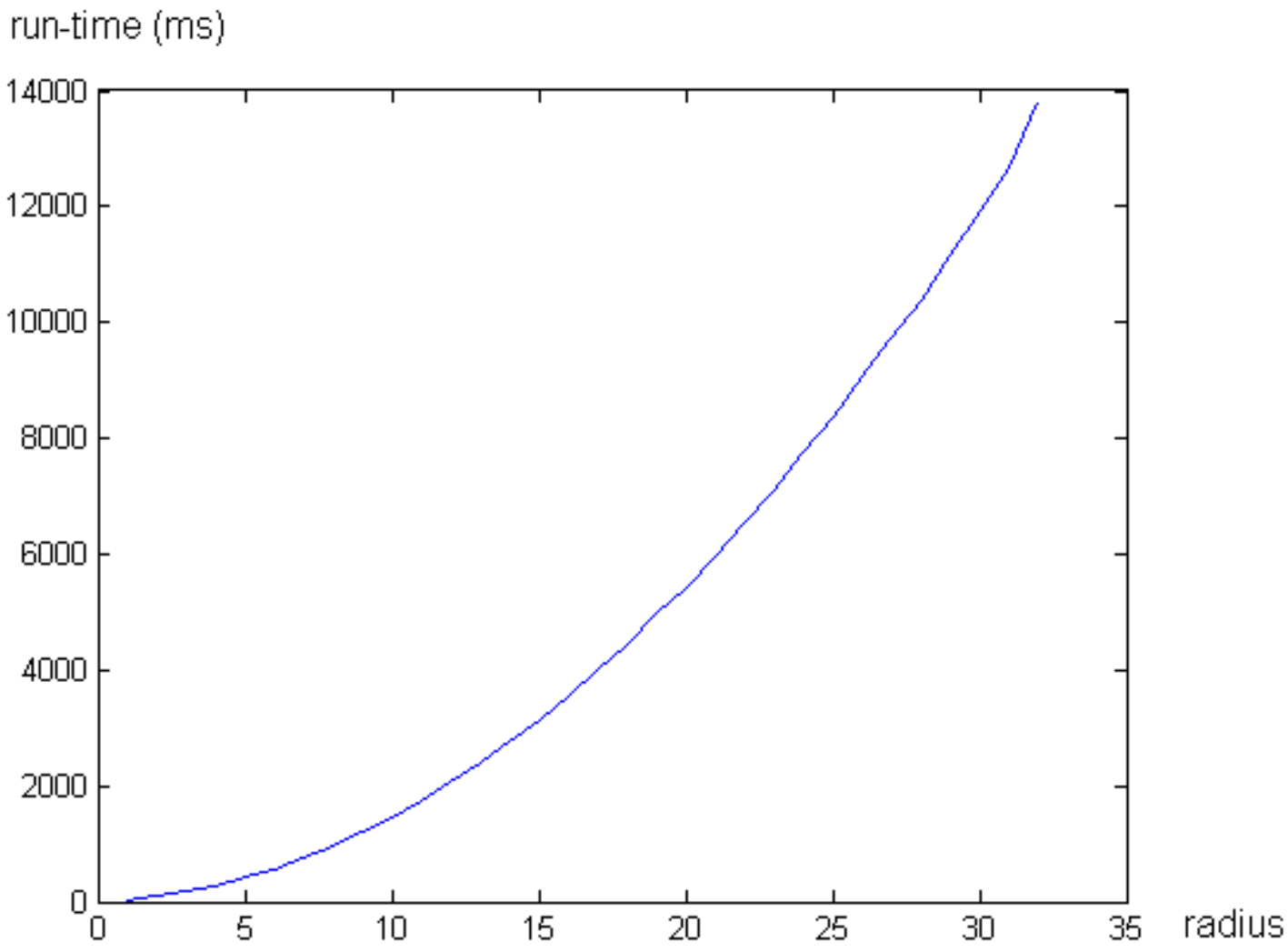
Averaging filter (7)

larger kernel → more smoothing

But the number of computations grows quadratically with kernel size!

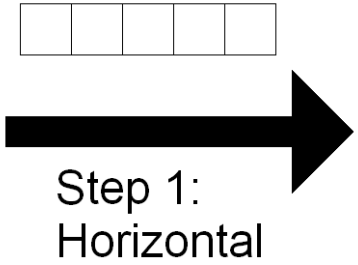
kSize	#ADD	#DIV
3x3	**9**	**1**
5x5	**25**	**1**
7x7	**49**	**1**
9x9	**81**	**1**

Averaging filter (8)

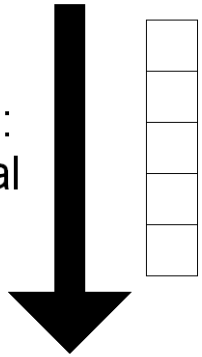


O(n) averaging filter (1)

Idea: Apply two one-dimensional filters sequentially



kSize	#ADD	#DIV
3	3	1
5	5	1
7	7	1
9	9	1



kSize	#ADD	#DIV
3	3	1
5	5	1
7	7	1
9	9	1

Step 1 + Step 2

kSize	#ADD	#DIV
3	3 + 3 = 6	1 + 1 = 2
5	5 + 5 = 10	1 + 1 = 2
7	7 + 7 = 14	1 + 1 = 2
9	9 + 9 = 18	1 + 1 = 2

Computational complexity is linear with respect to kernel size.

O(n) averaging filter (2)

156	108	156	149	115	82	79
79	140	95	63	106	108	95
75	144	115	91	95	71	79
60	95	99	144	95	108	100
84	91	59	65	91	108	156
95	132	91	75	67	71	84
79	82	106	99	67	95	82

Step 1: 5x1 Filter

		?				

60 + 95 + 99 + 144 + 95 = 493 / 5 = <u>98.6</u>

O(n) averaging filter (3)

156	108	156	149	115	82	79
79	140	95	63	106	108	95
75	144	115	91	95	71	79
60	95	99	144	95	108	100
84	91	59	65	91	108	156
95	132	91	75	67	71	84
79	82	106	99	67	95	82

Step 1: 5x1 Filter

		99	?			

95 + 99 + 144 + 95 + 108 = 541 / 5 = 108.2

O(n) averaging filter (4)

156	108	156	149	115	82	79
79	140	95	63	106	108	95
75	144	115	91	95	71	79
60	95	99	144	95	108	100
84	91	59	65	91	108	156
95	132	91	75	67	71	84
79	82	106	99	67	95	82

		99	108			

We finish Step 1 by applying the 5x1 filter to all remaining pixels.
Then we filter the result of Step 1 with a 1x5 filter (Step 2).

O(n) averaging filter (5)

The linear variant of the averaging filter boasts significant speed improvements over its quadratic counterpart

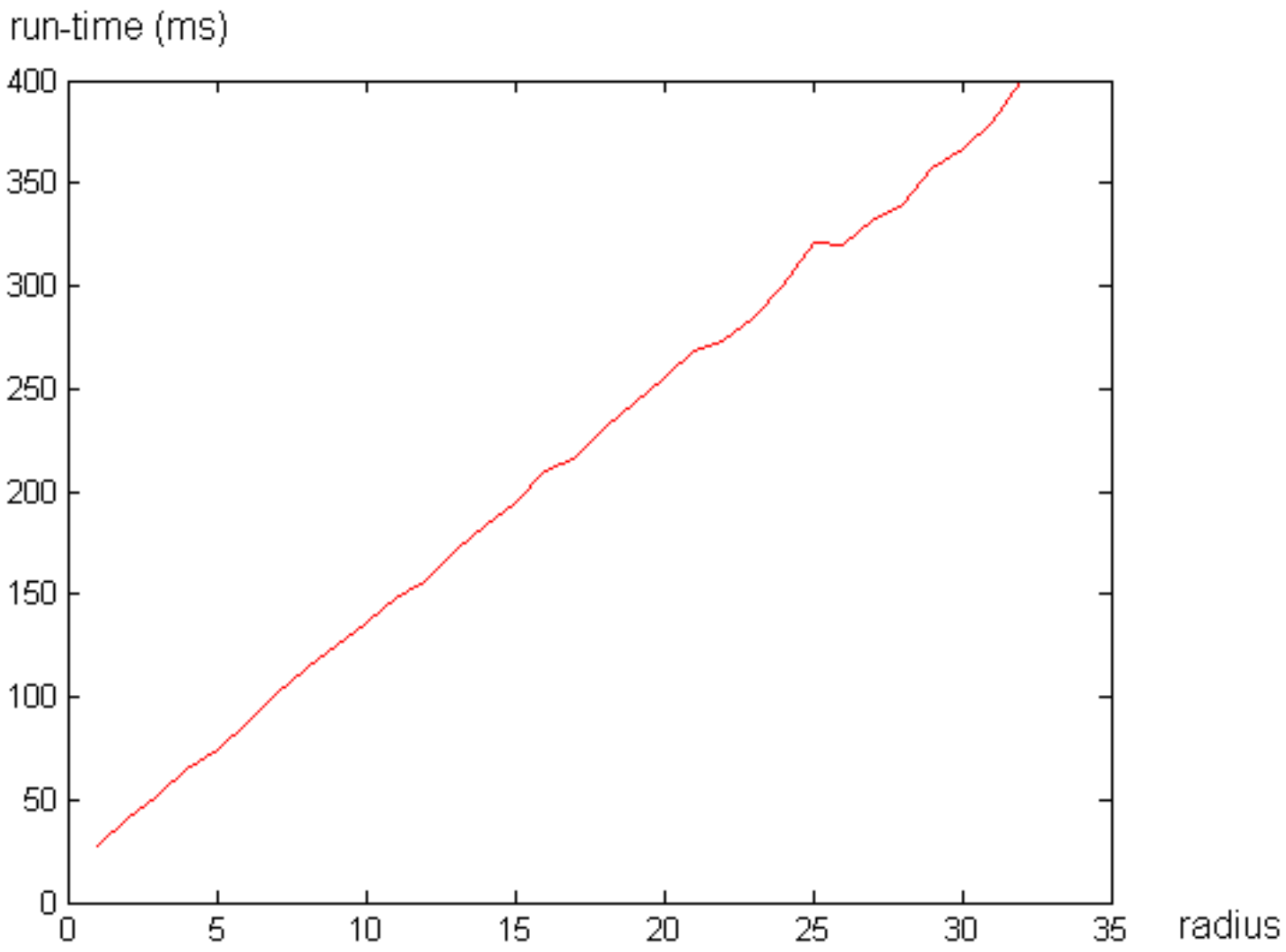
Quadratic Filter

kSize	#ADD	#DIV
3x3	9	1
5x5	25	1
7x7	49	1
9x9	81	1

Linear Filter

kSize	#ADD	#DIV
3	3 + 3 = 6	1 + 1 = 2
5	5 + 5 = 10	1 + 1 = 2
7	7 + 7 = 14	1 + 1 = 2
9	9 + 9 = 18	1 + 1 = 2

O(n) averaging filter (6)



O(n) averaging filter (7)

2	5	7
1	4	3
5	7	8

$$2 + 5 + 7 + 1 + 4 + 3 + 5 + 7 + 8 = 42$$

	42	

2	5	7
1	4	3
5	7	8

	14	
	8	
	20	

	42	

O(1) averaging filter (1)

IN

1	4	3	2	5	7	8

OUT

?	?	15	21	?	?	?

1 + 4 + 3 + 2 + 5 = 15

1	4	3	2	5	7	8

4 + 3 + 2 + 5 + 7 = 21

1	4	3	2	5	7	8

Linear complexity

O(1) averaging filter (2)

IN						
1	4	3	2	5	7	8

OUT						
?	?	15	21	?	?	?

1 + 4 + 3 + 2 + 5 = 15

1	4	3	2	5	7	8

Linear complexity

~~4 + 3 + 2 + 5 + 7 = 21~~ **15 - 1 + 7 = 21**

1	4	3	2	5	7	8

Constant complexity

O(1) averaging filter (3)

The constant averaging filter is a significant improvement!

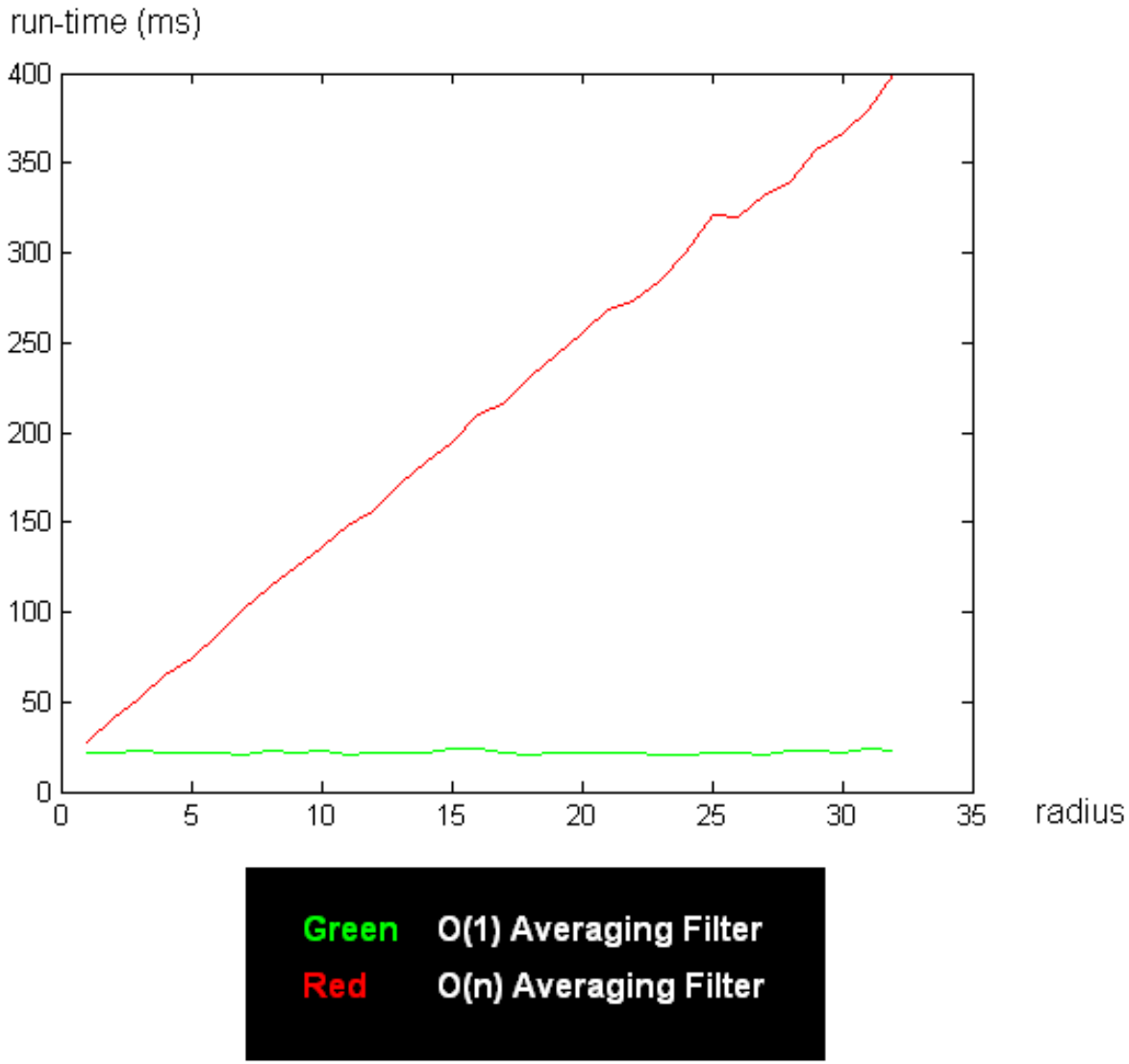
Linear Filter

kSize	#ADD	#DIV
3x1	3	1
5x1	5	1
7x1	7	1
9x1	9	1

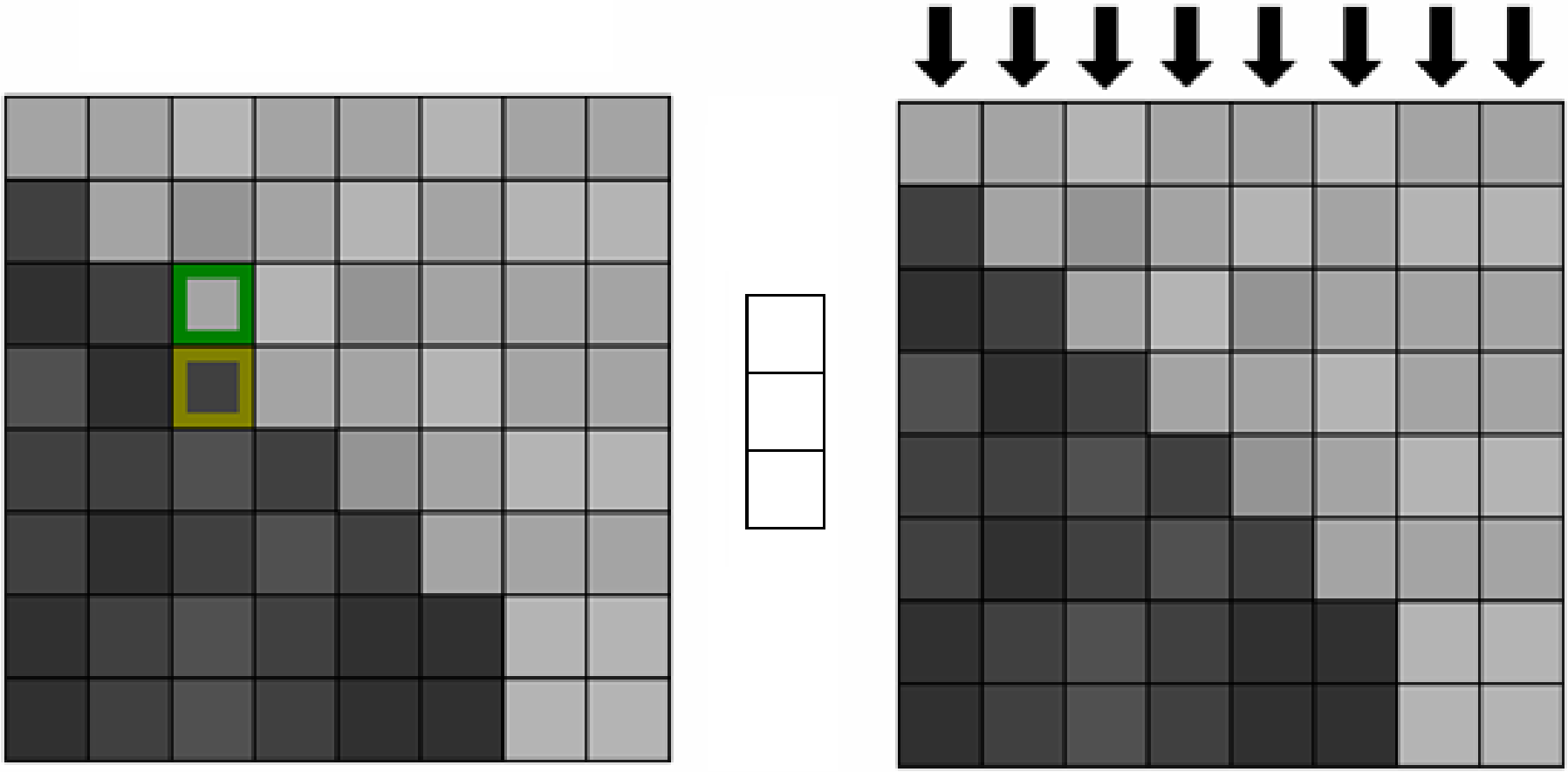
Constant Filter

kSize	#ADD	#DIV
3x1	1 / (3)	1
5x1	1 / (5)	1
7x1	1 / (7)	1
9x1	1 / (9)	1

O(1) averaging filter (4)

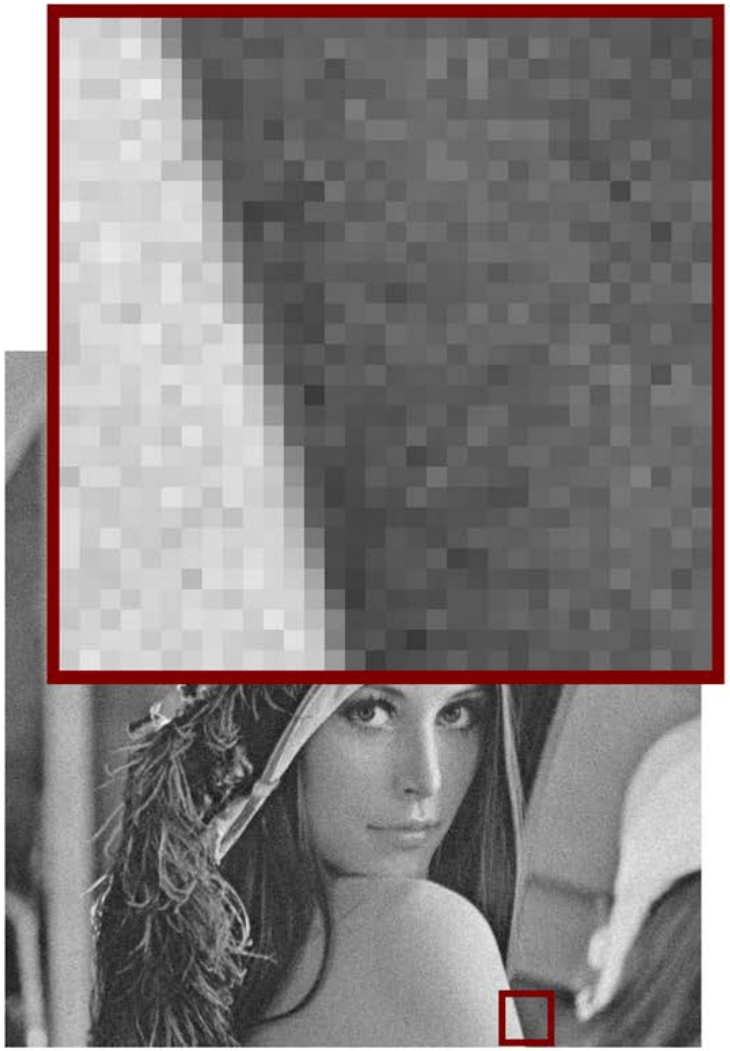


Parallelizing the averaging filter

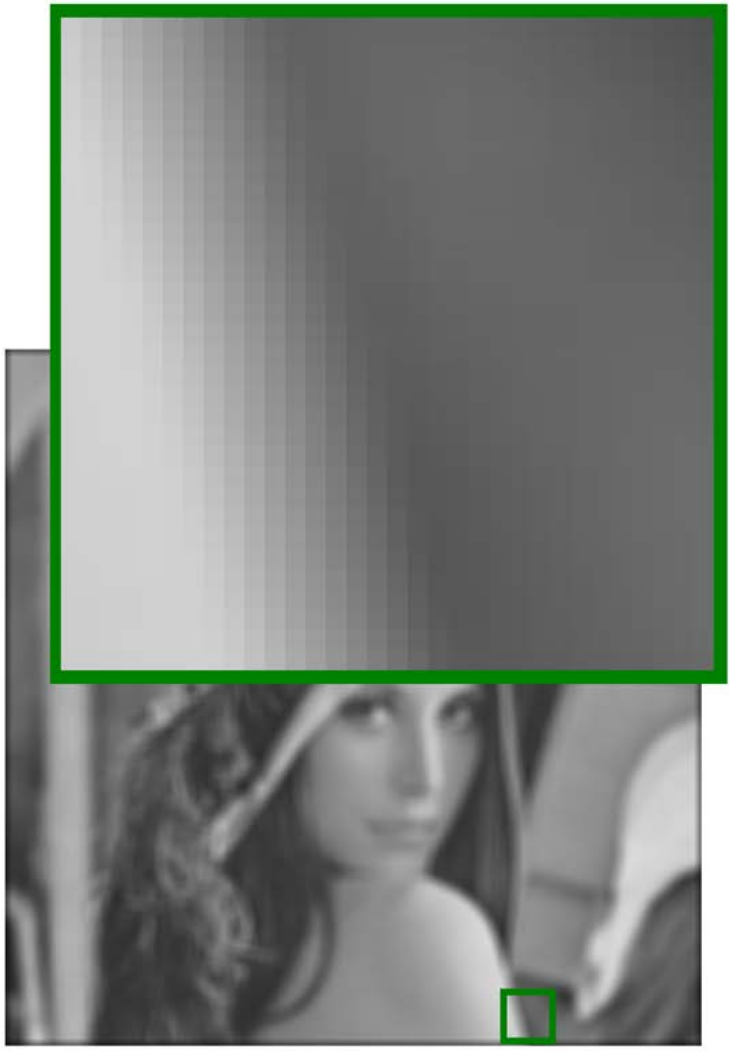


Serial CPU: 22.48ms / Frame
Parallel CUDA: 9.18ms / Frame **(!)**

Edge behaviour (1)

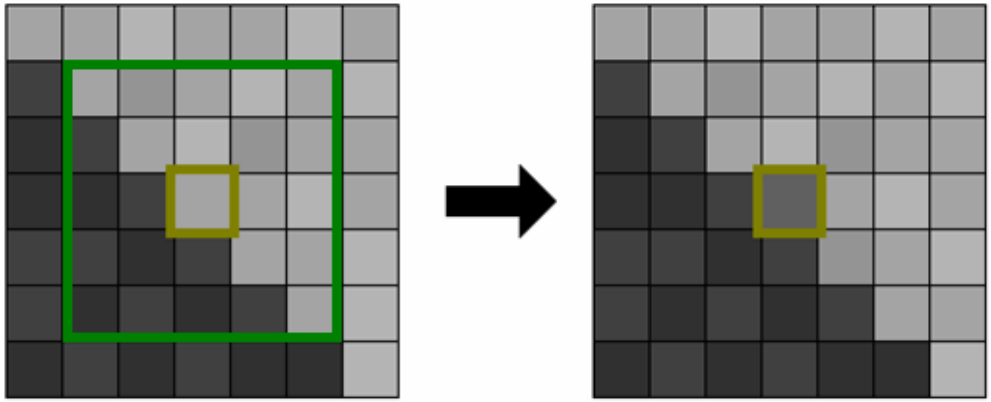


Original

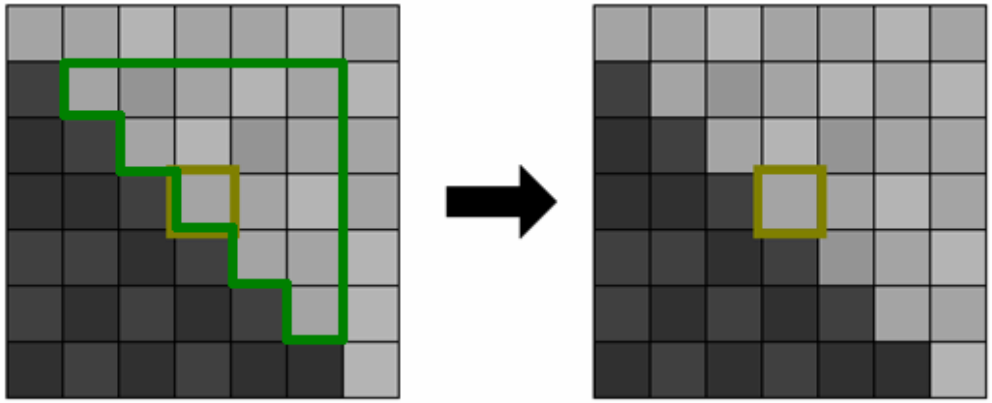


11x11 Average Filter

Edge behaviour (2)



what we have edge is smoothed



what we would like edge is preserved

Bilateral filter (1)

- **We make two assumptions:**
  - **Low intensity changes (e.g., less than 32) are due to noise.**
  - **High intensity changes (e.g., 32 or more) are due to edges.**

- **When filtering a pixel, we only average those pixels which have a low intensity difference to the pixel we want to filter.**

Bilateral filter (2)

66	66	57	66	67	56	208
62	60	62	71	67	215	217
65	64	63	76	218	214	218
60	48	66	206	215	213	218
68	67	207	206	217	220	206
65	211	220	222	221	213	218
217	207	217	211	211	220	209

Bilateral filter (3)

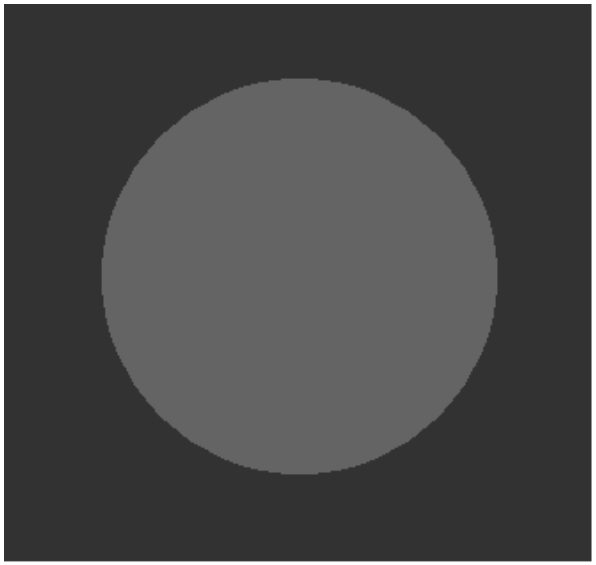
66	66	57	66	67	56	208
62	60	62	71	67	215	217
65	64	63	76	218	214	218
60	48	66	206	215	213	218
68	67	207	206	217	220	206
65	211	220	222	221	213	218
217	207	217	211	211	220	209

		?				

62 + 60 + 62 ... 67 + 65 = 964 / 15 = 64.27

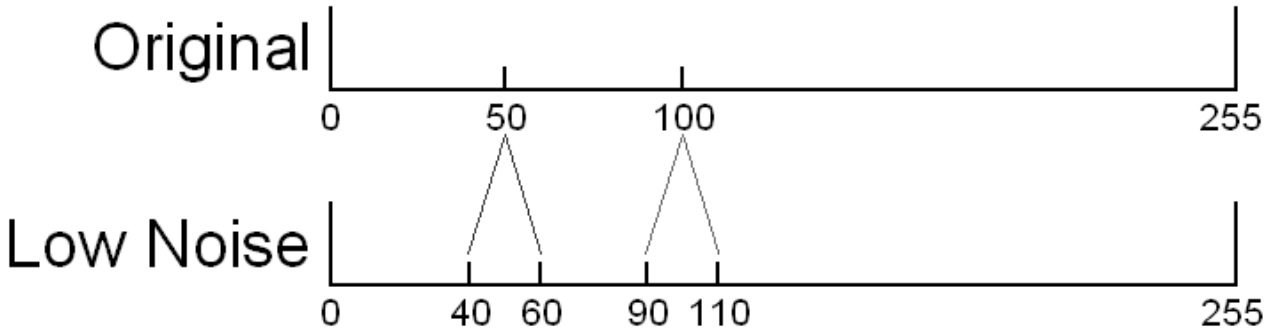
Bilateral filter (4)

- **Low intensity changes are always due to noise?**



Good Case

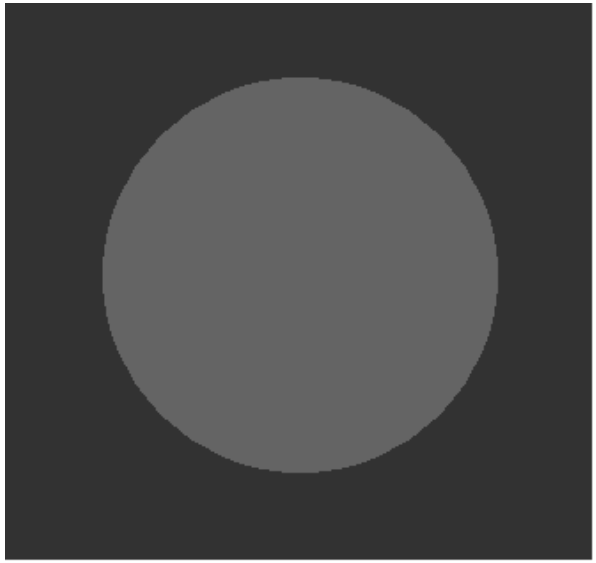
■ = 50 ● = 100




```
60 - 40 = 20
110 - 90 = 20
90 - 60 = 30
```

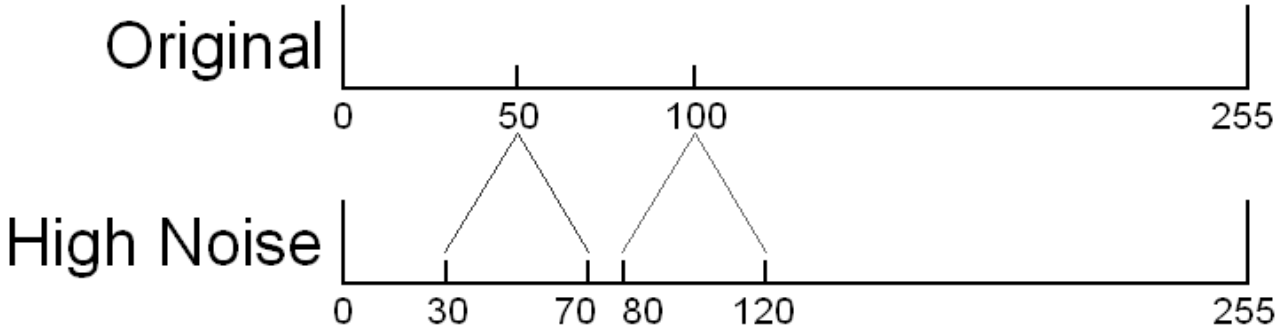
Bilateral filter (5)

- **Low intensity changes are always due to noise?**



Bad Case

■ = 50 ● = 100

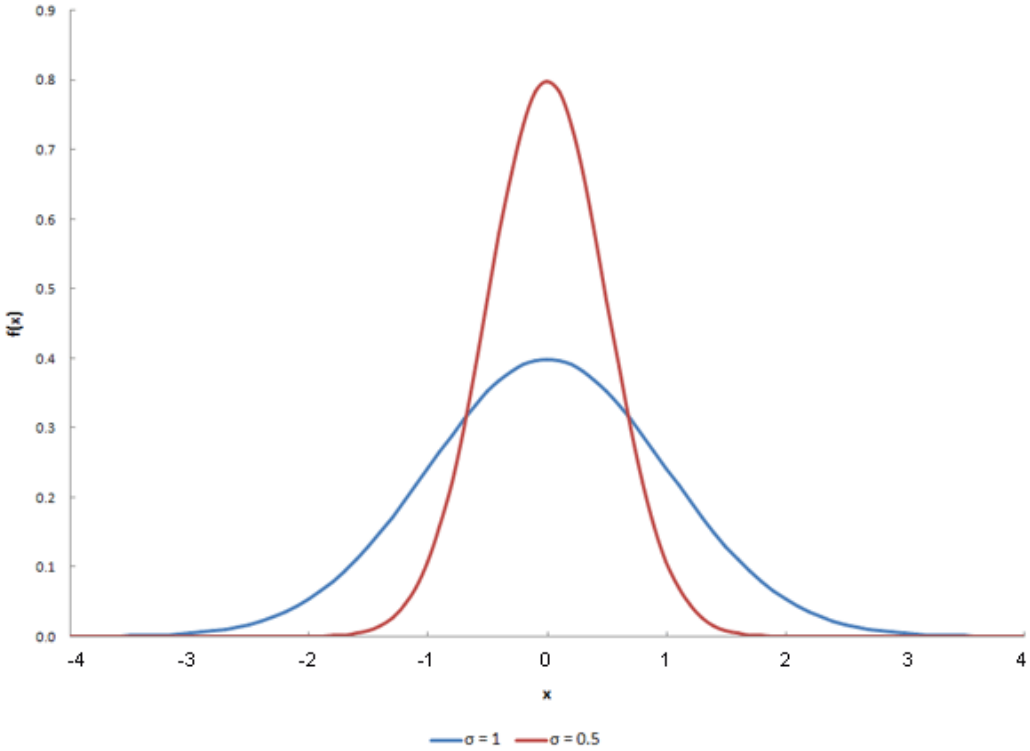



```
70 - 30 = 40
120 - 80 = 40
80 - 70 = 10
```

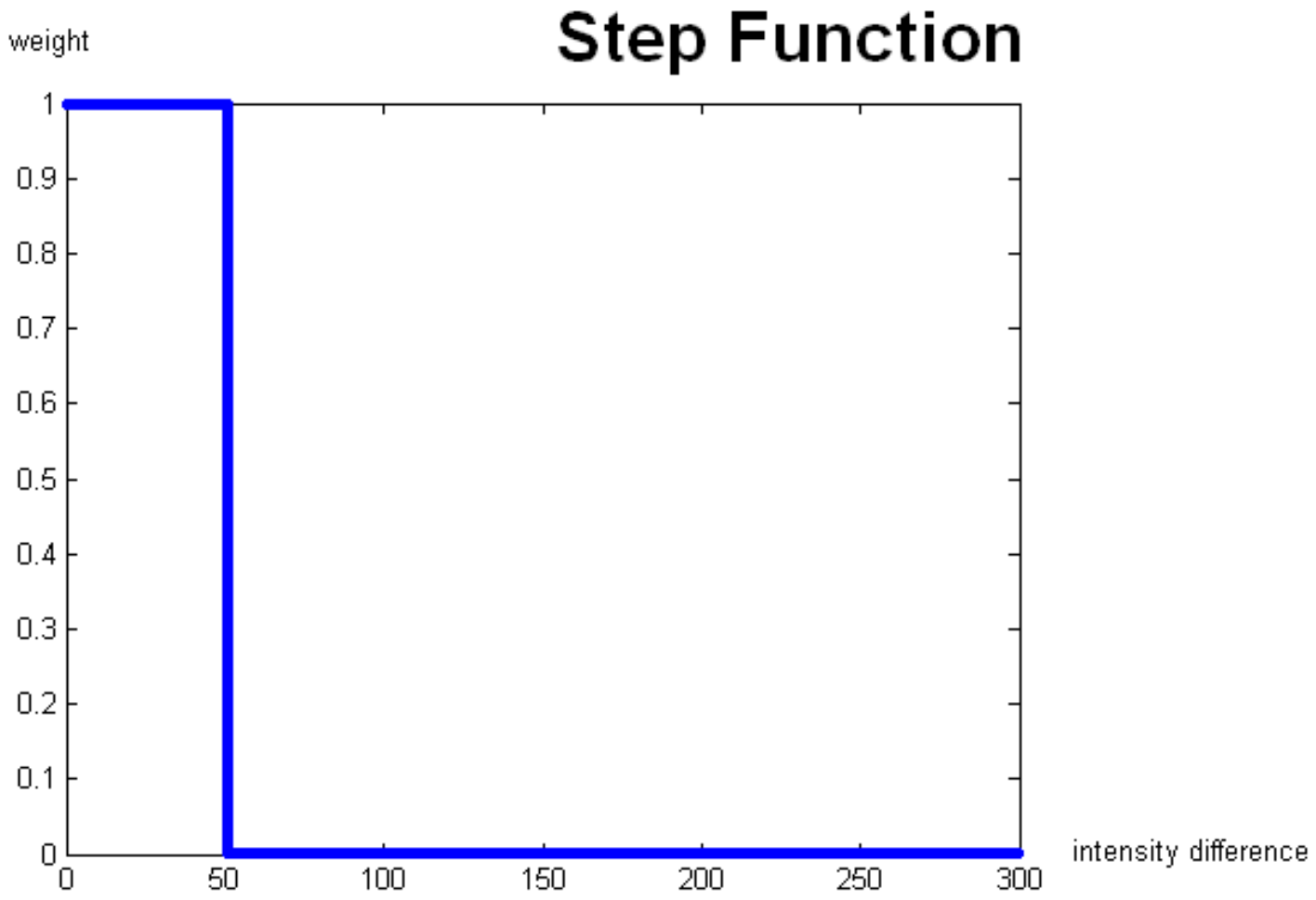
Bilateral filter (6)

- **High intensity changes are always due to edges?**

- **In image processing it is often assumed that the noise has a zero-mean Gaussian distribution**



Bilateral filter (7)



Bilateral filter (8)

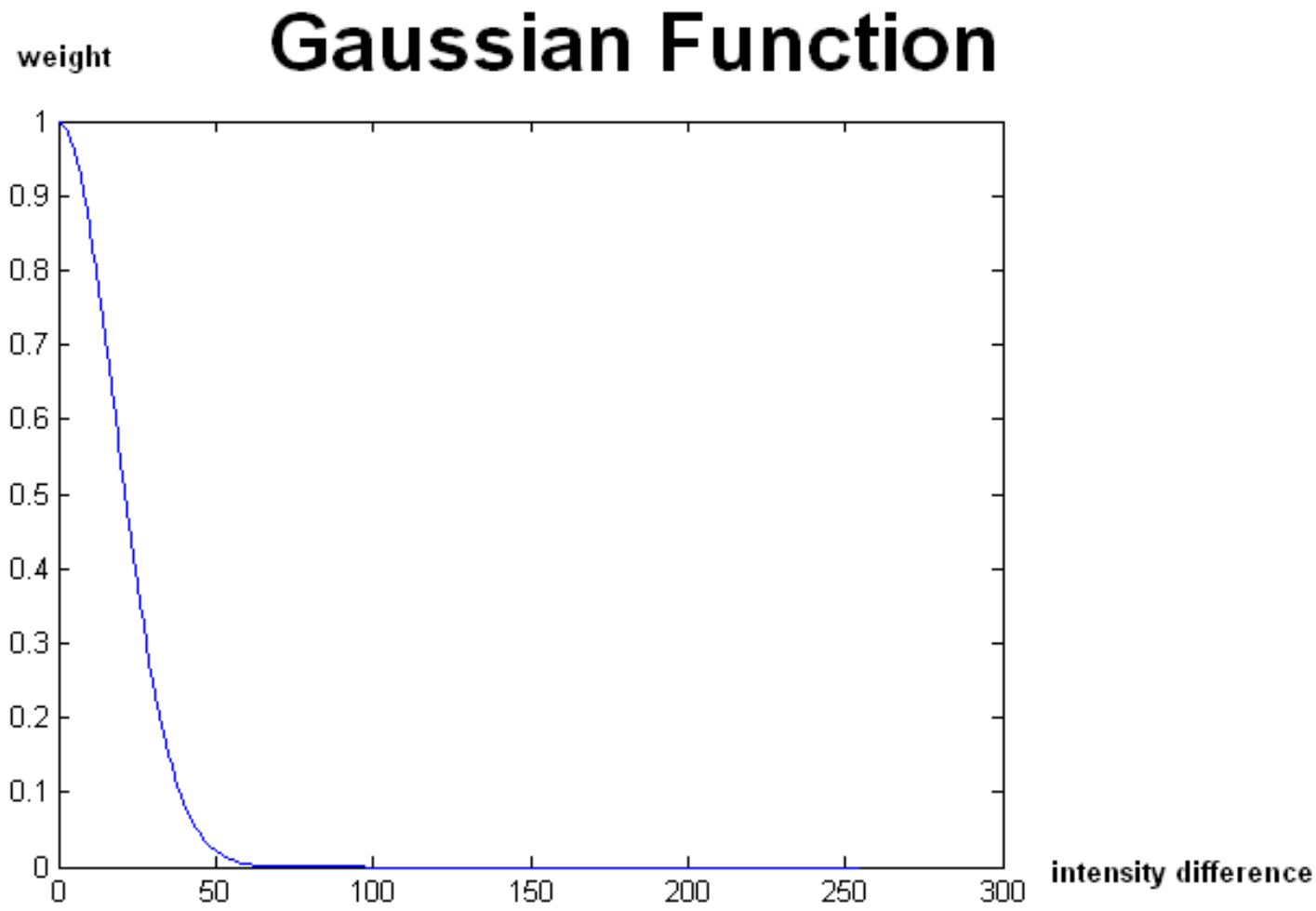
66	66	57	66	67	56	208
62	60	62	71	67	215	217
65	64	63	76	218	214	218
60	48	66	206	215	213	218
68	67	207	206	217	220	206
65	211	220	222	221	213	218
217	207	217	211	211	220	209

```
62 * 1.0 +  60 * 1.0 +  62 * 1.0 +  71 * 1.0 +  67 * 1.0 +
65 * 1.0 +  64 * 1.0 +  63 * 1.0 +  76 * 1.0 + 218 * 0.0 +
60 * 1.0 +  48 * 1.0 +  66 * 1.0 + 206 * 0.0 + 215 * 0.0 +
68 * 1.0 +  67 * 1.0 + 207 * 0.0 + 206 * 0.0 + 217 * 0.0 +
65 * 1.0 + 211 * 0.0 + 220 * 0.0 + 222 * 0.0 + 221 * 0.0 =
                                                        964

1.0 + 1.0 + 1.0 + 1.0 + 1.0 +
1.0 + 1.0 + 1.0 + 1.0 + 0.0 +
1.0 + 1.0 + 1.0 + 0.0 + 0.0 +
1.0 + 1.0 + 0.0 + 0.0 + 0.0 +
1.0 + 0.0 + 0.0 + 0.0 + 0.0 = 15
```

964 / 15 = <u>64.27</u>

Bilateral filter (9)



Bilateral filter (10)

$$I'_p = \frac{1}{W} \sum_{q \in R_p} G(|I_p - I_q|) \cdot I_q$$

$$W = \sum_{q \in R_p} G(|I_p - I_q|)$$

I = input image (the image to filter)

I' = output image (the filtered image)

G = weight function (Step, Gaussian, etc.)

p = position to filter

R_p = window centered on p

The weight function G and the dimension of the window R_p are filter parameters.

Bilateral filter (11)

$$I'_p = \frac{1}{W} \sum_{q \in R_p} G(|I_p - I_q|) \cdot I_q \cdot \boxed{H(\|p - q\|)}$$

spatial weight

$$W = \sum_{q \in R_p} G(|I_p - I_q|) \cdot \boxed{H(\|p - q\|)}$$

I = input image (the image to filter)

I' = output image (the filtered image)

G = weight function (Step, Gaussian, etc.)

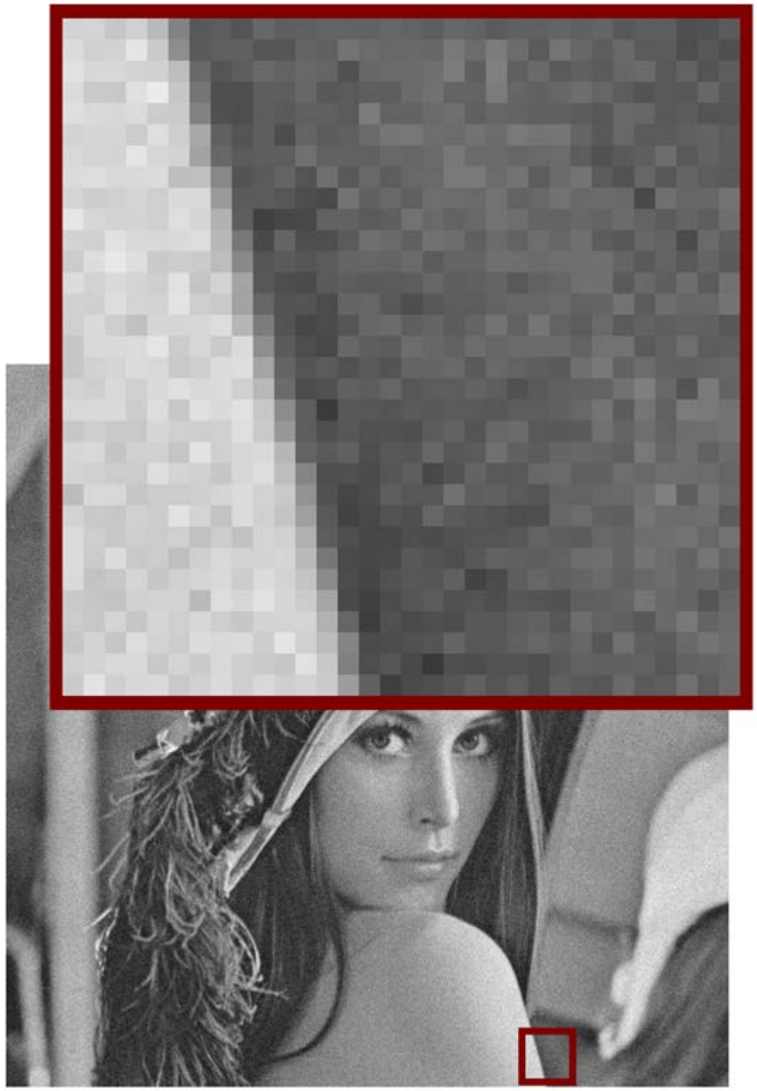
p = position to filter

R_p = window centered on p

The weight function G and the dimension of the window R_p are filter parameters.

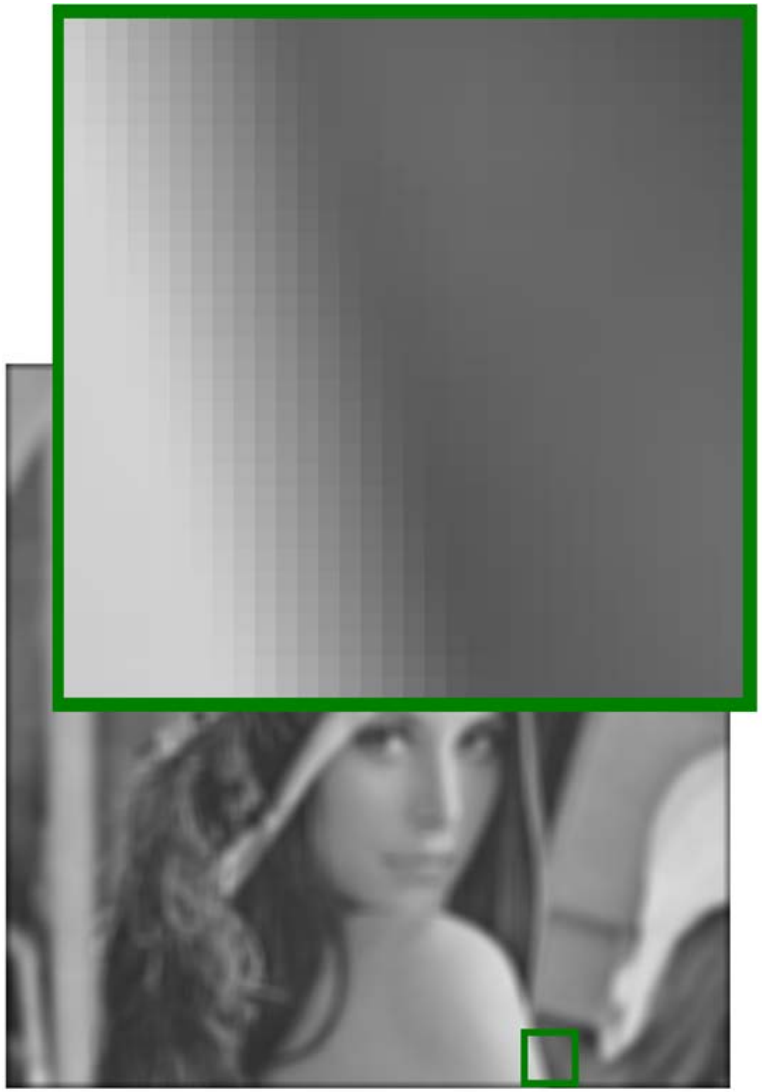
Edge behaviour (1)

Original



Edge behaviour (2)

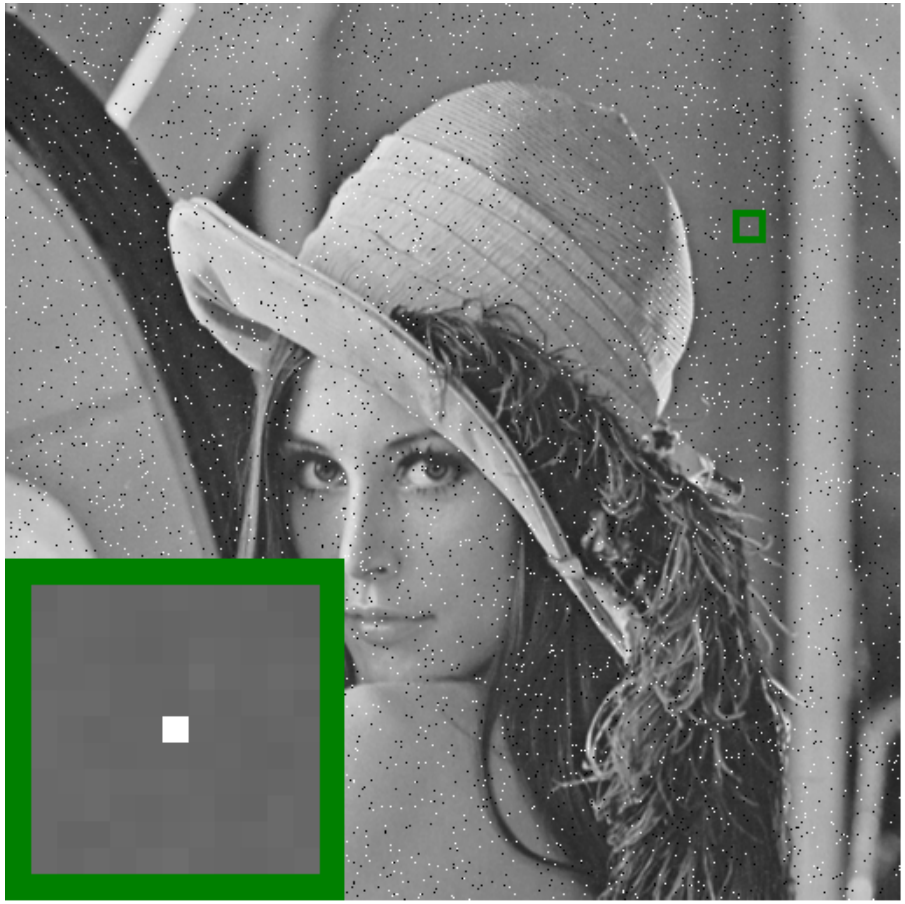
Averaging Filter



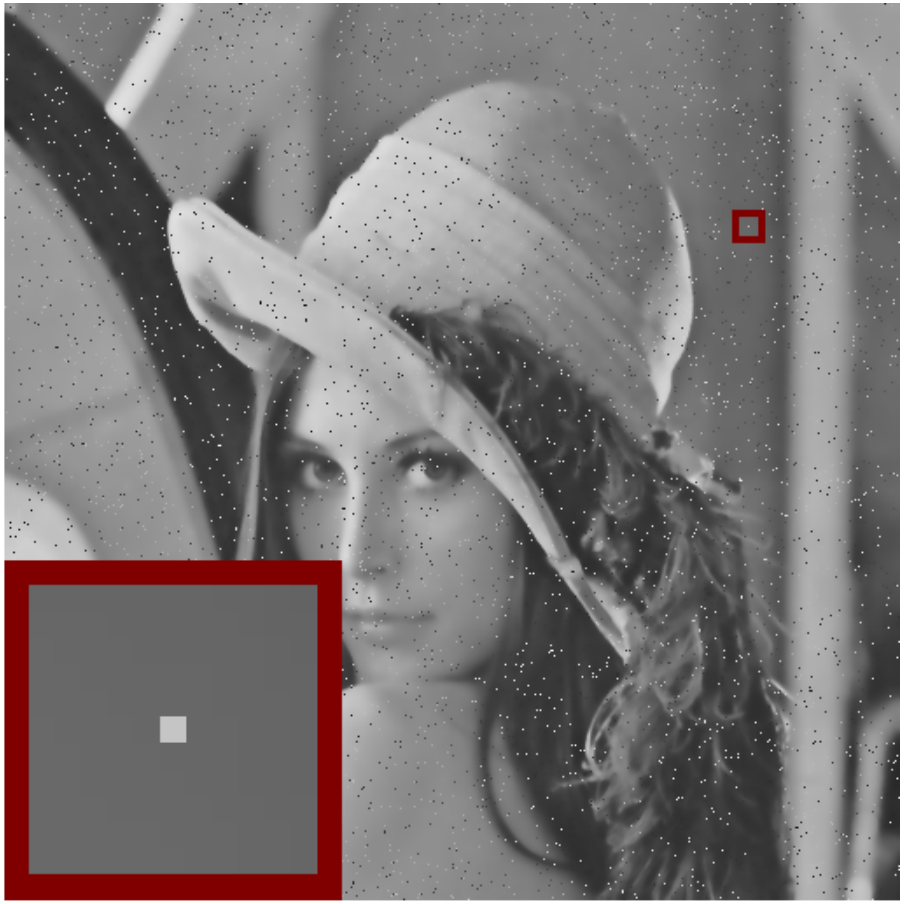
Edge behaviour (3)

Bilateral Filter

Salt & pepper noise

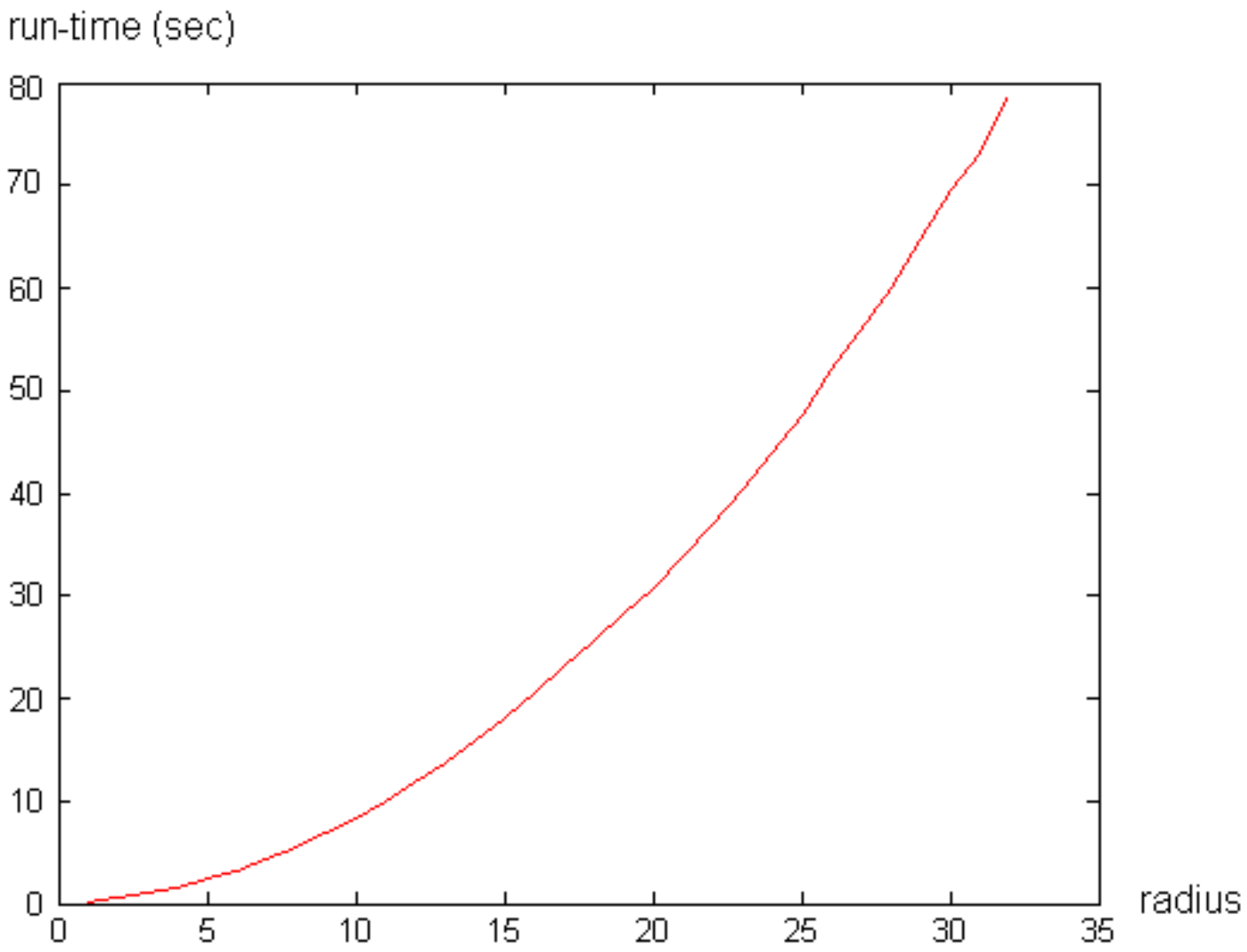


Original



Bilateral Filter

Performance of the bilateral filter



Optimizing the bilateral filter

- **Evaluation of the weight function can be expensive (e.g., Gaussian function).**

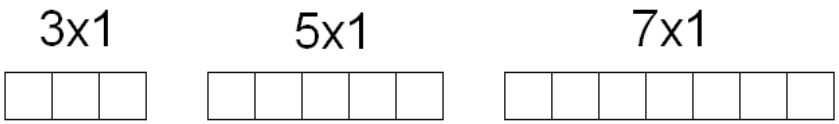
- **In an 8-bit grayscale image there can only be 256 possible intensity differences (0-255).**

- **We can sample the weight function at these 256 points and save the results in a Look-Up Table.**

- **Instead of evaluating the weight function for every pixel we simple use the corresponding value from the Look-Up Table.**

Separating the bilateral filter (1)

- **A one-dimensional bilateral filter is O(n) with respect to kernel size, because the number of pixels inside the kernel grows linearly with kernel size.**



- **From the averaging filter we know: If we apply two one-dimensional filters sequentially, the run-time will still be O(n) with respect to kernel size.**

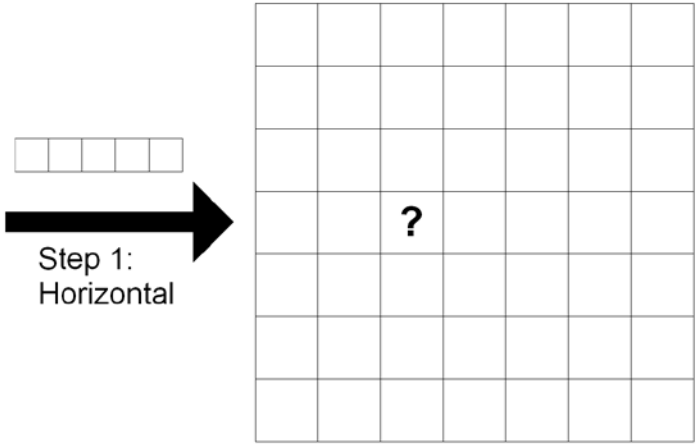
Separating the bilateral filter (2)

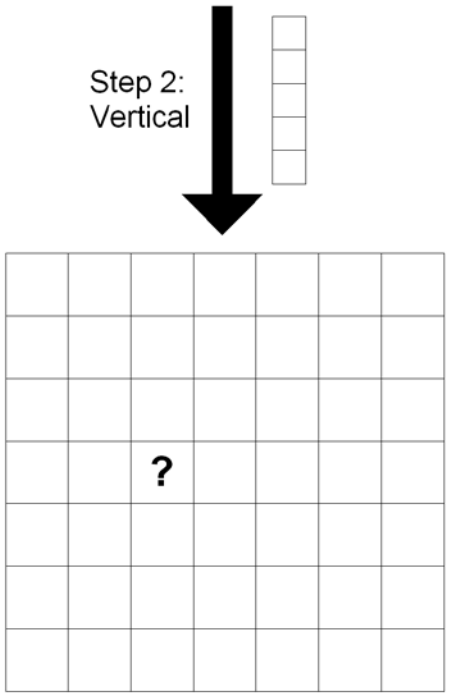
- **Instead of using an m x n bilateral filter we can sequentially apply an m x 1 and 1 x n bilateral filter to achieve O(n) run-time.**

- **We get a (good) approximation of the two-dimensional bilateral filter!**

Separating the bilateral filter (3)

66	66	57	66	67	56	208
62	60	62	71	67	215	217
65	64	63	76	218	214	218
60	48	66	206	215	213	218
68	67	207	206	217	220	206
65	211	220	222	221	213	218
217	207	217	211	211	220	209

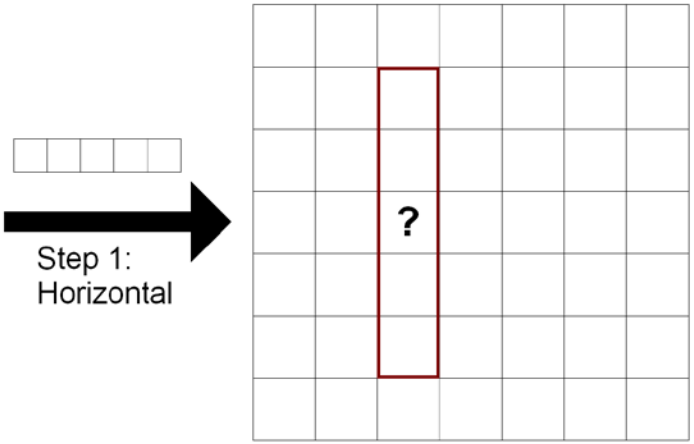


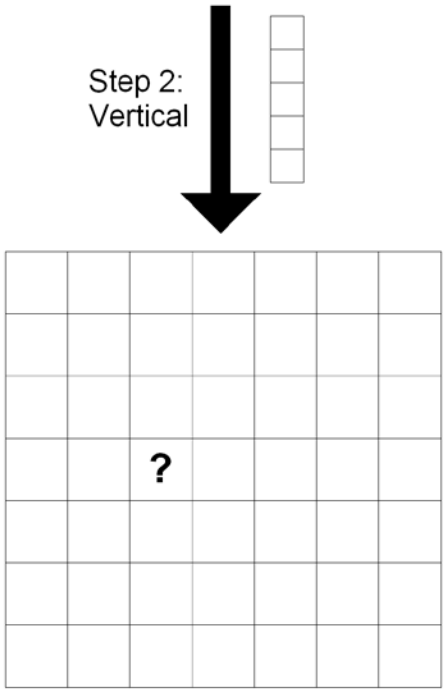


Separated Bilateral Filter
Step 1

Separating the bilateral filter (4)

66	66	57	66	67	56	208
62	60	62	71	67	215	217
65	64	63	76	218	214	218
60	48	66	206	215	213	218
68	67	207	206	217	220	206
65	211	220	222	221	213	218
217	207	217	211	211	220	209

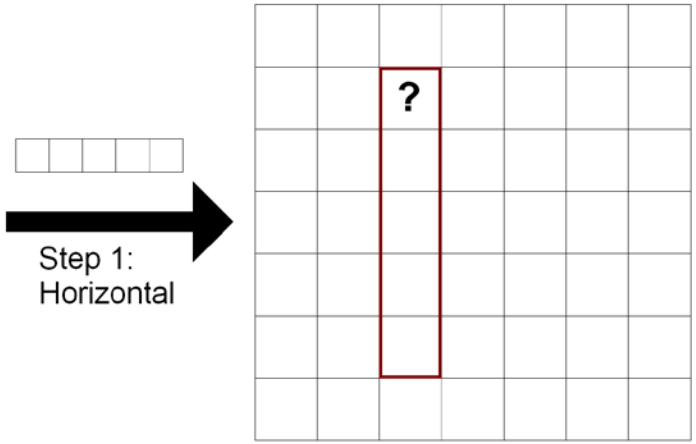




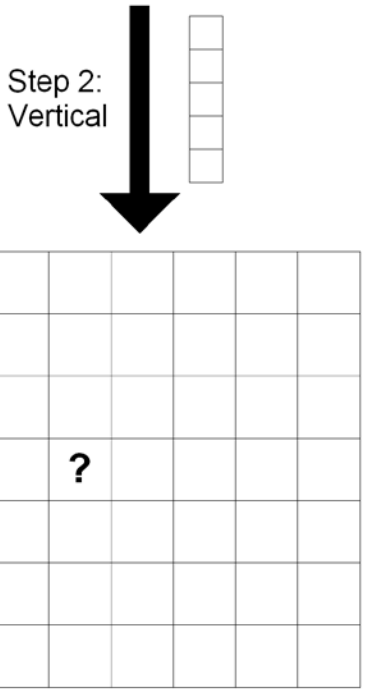
Separated Bilateral Filter
Step 2

Separating the bilateral filter (5)

66	66	57	66	67	56	208
62	60	62	71	67	215	217
65	64	63	76	218	214	218
60	48	66	206	215	213	218
68	67	207	206	217	220	206
65	211	220	222	221	213	218
217	207	217	211	211	220	209



(62 + 60 + 62 + 71 + 67) / 5 = 64.4



Separated Bilateral Filter
Step 3

Separating the bilateral filter (6)

66	66	57	66	67	56	208
62	60	62	71	67	215	217
65	64	63	76	218	214	218
60	48	66	206	215	213	218
68	67	207	206	217	220	206
65	211	220	222	221	213	218
217	207	217	211	211	220	209

Step 1:
Horizontal

		64				
		?				

(65 + 64 + 63 + 76) / 4 = 67

Step 2:
Vertical

Separated Bilateral Filter
Step 4

		?				

Separating the bilateral filter (7)

66	66	57	66	67	56	208
62	60	62	71	67	215	217
65	64	63	76	218	214	218
60	48	66	206	215	213	218
68	67	207	206	217	220	206
65	211	220	222	221	213	218
217	207	217	211	211	220	209

Step 1:
Horizontal

		64				
		67				
		58				
		?				

(207 + 206 + 217) / 3 = 210

Step 2:
Vertical

		?				

Separated Bilateral Filter
Step 5

Separating the bilateral filter (8)

66	66	57	66	67	56	208
62	60	62	71	67	215	217
65	64	63	76	218	214	218
60	48	66	206	215	213	218
68	67	207	206	217	220	206
65	211	220	222	221	213	218
217	207	217	211	211	220	209

Step 1:
Horizontal

		64				
		67				
		58				
		210				
		219				

(64 + 67 + 58) / 3 = 63

Step 2:
Vertical

Separated Bilateral Filter
Step 6

		?				

Separating the bilateral filter (9)

66	66	57	66	67	56	208
62	60	62	71	67	215	217
65	64	63	76	218	214	218
60	48	66	206	215	213	218
68	67	207	206	217	220	206
65	211	220	222	221	213	218
217	207	217	211	211	220	209

Step 1:
Horizontal

		64				
		67				
		58				
		210				
		219				

Step 2:
Vertical

		63				

Separated Bilateral Filter
Step 7

O(1) bilateral filter? (1)

- **The running averaging filter achieves constant run-time with respect to kernel size by updating the previous sum.**

- **Can we do the same for the bilateral filter?**

O(1) bilateral filter? (2)

1:	15	17	19	65	69	72	70	15 17 19	**= 51**
2:	15	17	19	65	69	72	70	65 69 72	**= 206**

1. Remove 15.
2. Check if 17 is similar to 65. Remove 17.
3. Check if 19 is similar to 65. Remove 19.
4. Add 65.
5. Check if 69 is similar to 65. Add 69.
6. Check if 72 is similar to 65. Add 72.

➡ **The performance gets worse!**

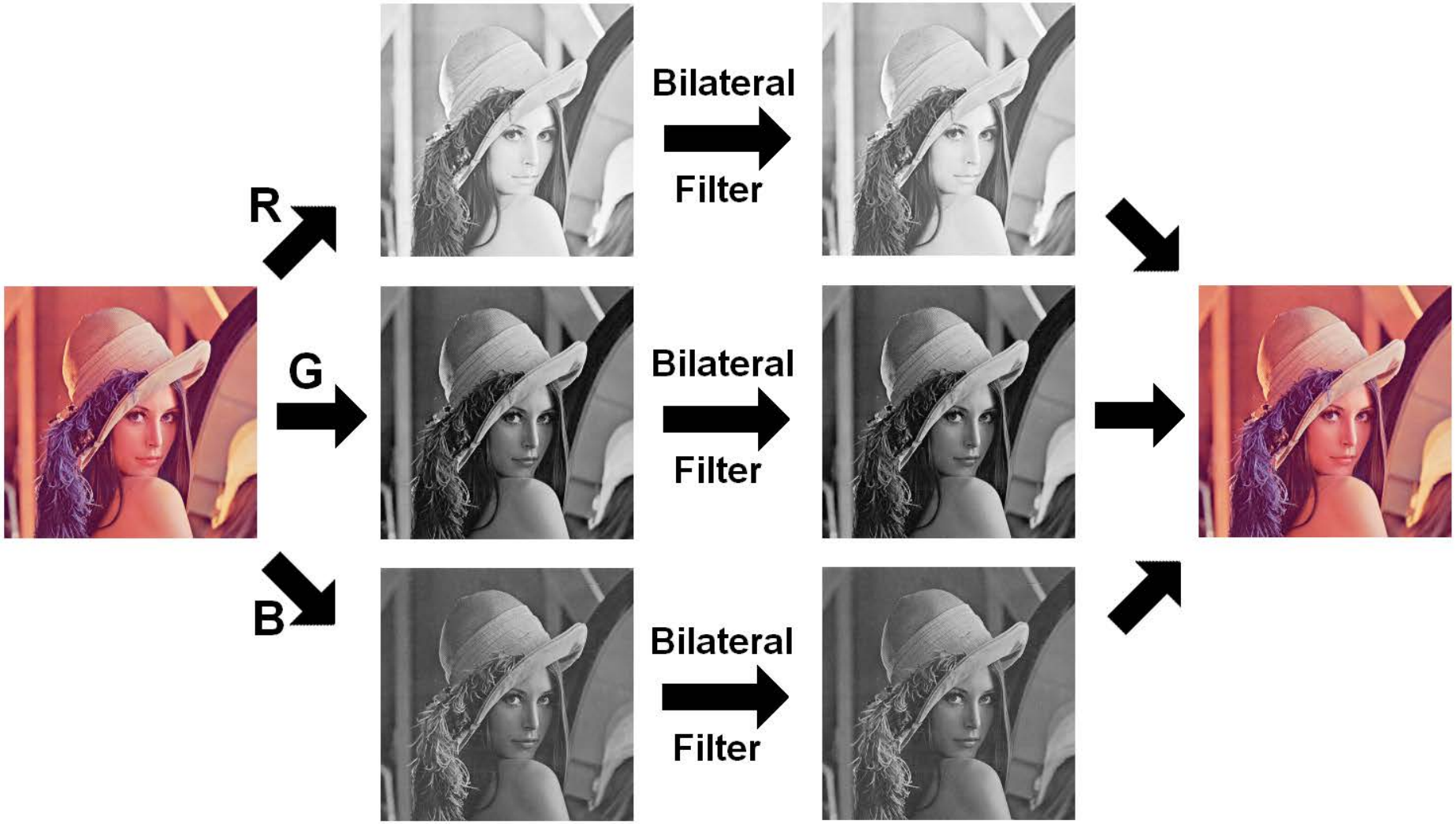
O(1) bilateral filter? (3)

Porikli, F.: “Constant Time O(1) Bilateral Filtering”, IEEE Conference on Computer Vision and Pattern Recognition, 8pp., 2008.

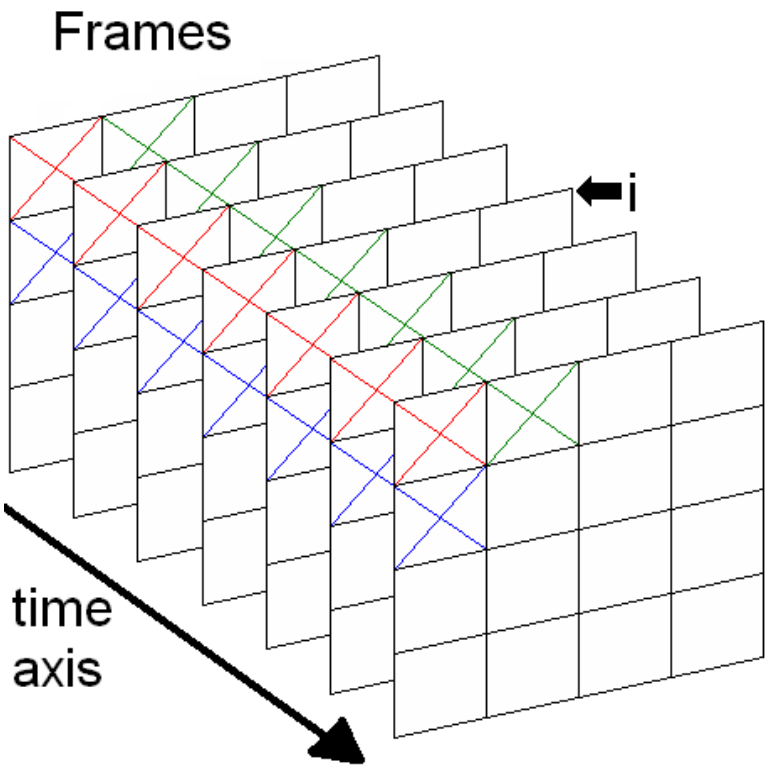
Yang, Q., Tan, K.H., Ahuja, N.: “Real-time O(1) Bilateral Filtering”, Conference on Computer Vision and Pattern Recognition, pp. 557-564, 2009.

Chaudhury, K.N., Sage, D., Unser, M.: “Fast O(1) Bilateral Filtering using Trigonometric Range Kernels”, IEEE Transactions on Image Processing, vol. 20, no. 11, 13pp., 2011.

Color images



Temporal bilateral filter (1)



We want to filter the pixels in Frame i.

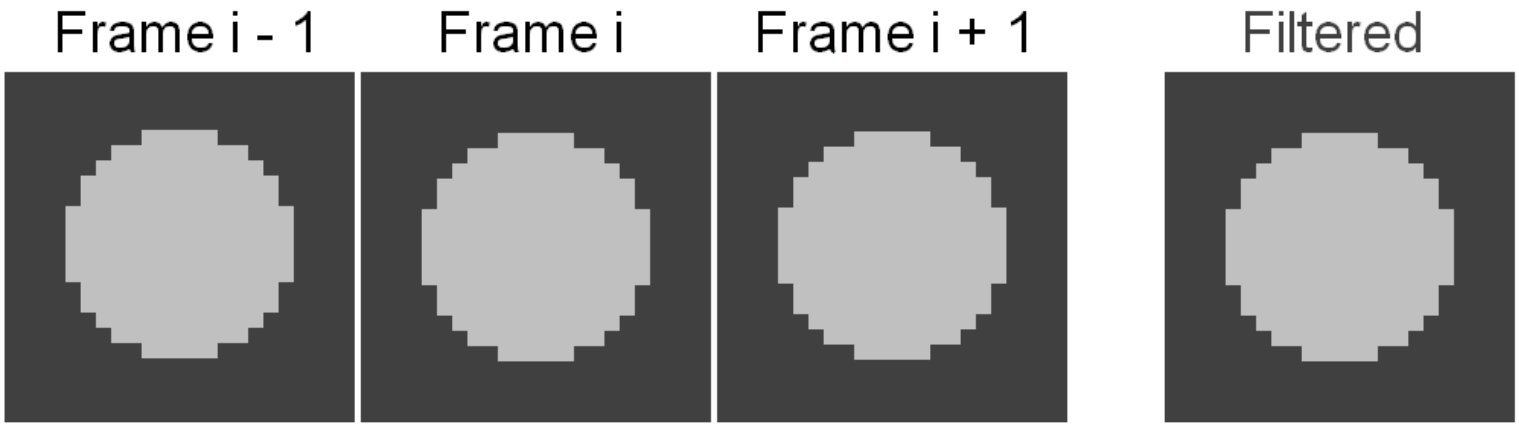
RED: Kernel when filtering position (0,0)

GREEN: Kernel when filtering position (1,0)

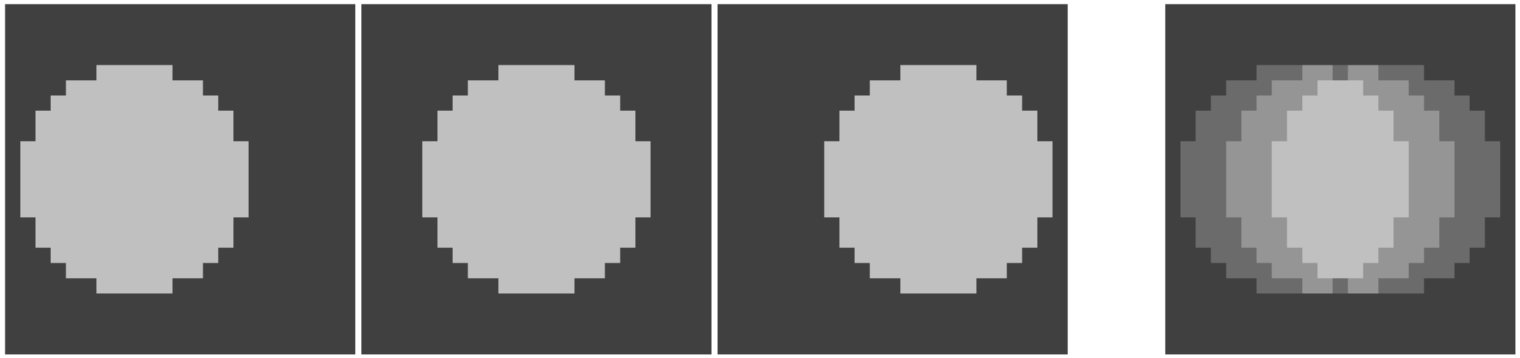
BLUE: Kernel when filtering position (0,1)

Temporal bilateral filter (2)

What if we use the faster averaging filter?



Non-moving objects are filtered fine.



Moving objects have ghosting artifacts.

Spatio-temporal bilateral filter

We have two choices:

1. **Use a volumetric kernel**
2. **Perform spatial and temporal filtering sequentially**

Other edge preserving smoothing filters

Francis, J.J, De Jager, G.: “The Bilateral Median Filter”, Proceedings of the 14th Symposium of the Pattern Recognition Association of South Africa, pp. 57–62, 2003.

He, K., Sun, J., Tang, X.: “Guided Image Filtering”, European Conference on Computer Vision, 14pp., 2010.

Choudhury, P., Tumblin, J.: “The Trilateral Filter for High Contrast Images and Meshes“, Proceedings of the Eurographics Symposium on Rendering, 11pp., 2003.

Median Filter:

16	16	16
16	210	16
16	16	16

value:	16	16	16	16	16	16	16	16	210

Median

16	16	16
16	16	16
16	16	16

Bilateral Median Filter:

16	16	16
16	210	16
16	16	16

value:	16	16	16	16	16	16	16	16	210
weight:	0.05	0.05	0.05	0.05	0.05	0.05	0.05	0.05	1.0

sum: 0.4

Weighted Median

16	16	16
16	210	16
16	16	16

The Bilateral Median Filter is <u>not</u> suitable for suppresion of Salt & Pepper Noise!

Cartoonization

Interactive image segmentation

Rhemann, C., Hosni, A., Bleyer, M., Rother, C., Gelautz, M.: "Fast Cost-Volume Filtering for Visual Correspondence and Beyond", Conference on Computer Vision and Pattern Recognition, 8pp., 2011.

Deblocking

Acknowledgment

Special thanks go to Georg Braun for the preparation of the slides

Thank you!