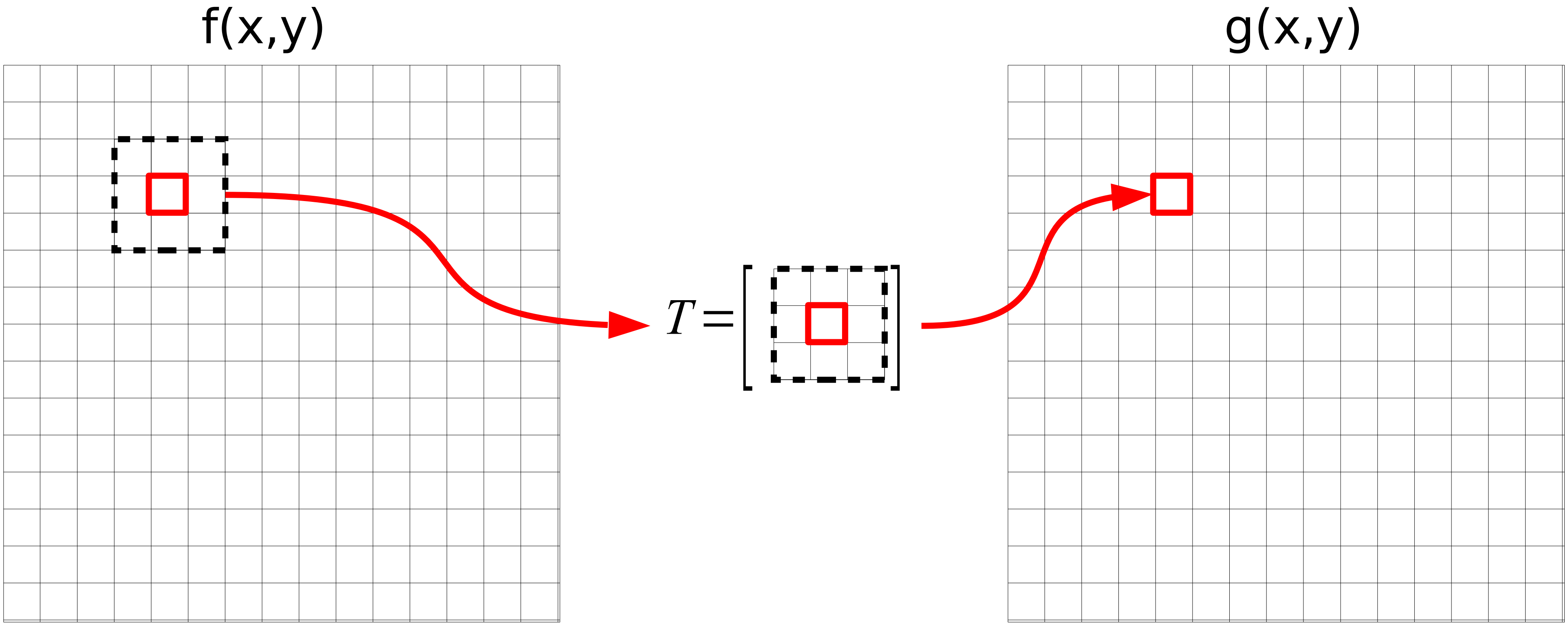
Digital Image Processing

Berlin University of Technology (TUB),
Computer Vision and Remote Sensing Group
Berlin, Germany

Purpose of Digital Image Processing

Image restoration: Improving *objective* image quality
e.g. noise suppression

Sliding Window

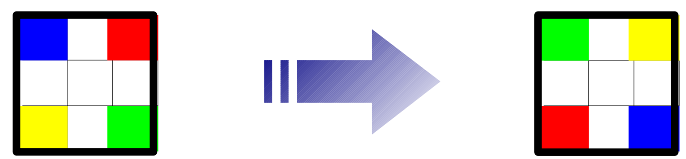


- Operator T takes into account only local information
- Result in g is based on pixel intensity and intensities of neighbours
  - ➔ '*Filter size*' refers to size of neighbourhood (e.g. 3x3 pixels)

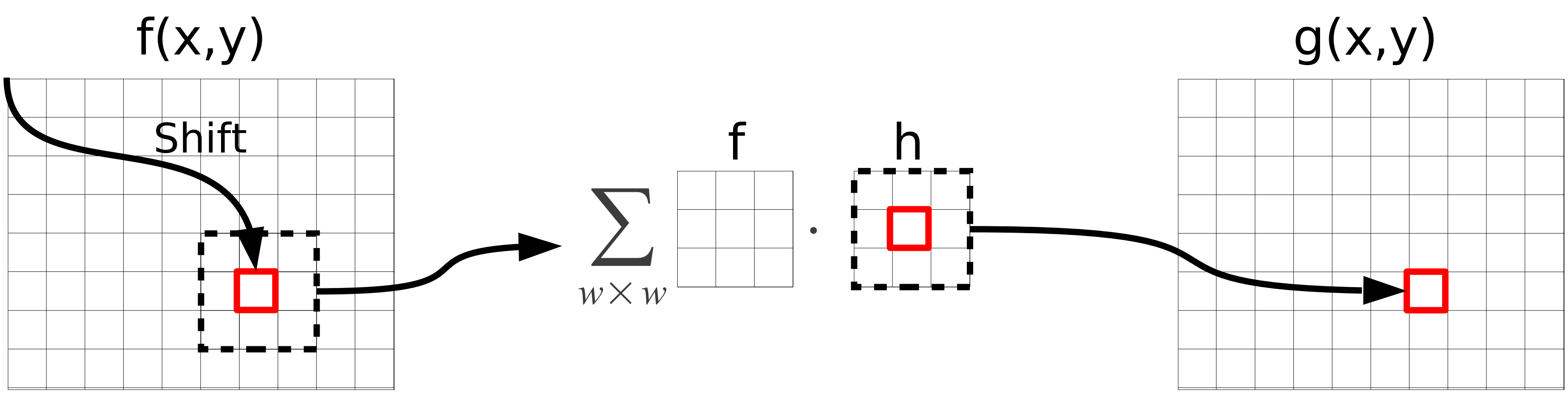
Convolution

$$g(\alpha,\beta)=\sum_{x=1}^{N}\sum_{y=1}^{M} f(x,y)\cdot h(x-\alpha,y-\beta)$$

1. **Flip filter kernel** (about the filter centre)



2. **Shift** (re-centre), **Multiply** and **Integrate**



Convolution

- Filter consists of coefficients and has a centre:

$$h(r,s)=\begin{pmatrix} h(-1,-1) & h(0,-1) & h(1,-1) \\ h(-1,0) & \boxed{h(0,0)} & h(1,0) \\ h(-1,1) & h(0,1) & h(1,1) \end{pmatrix}$$

- Linear filters are applied by *convolution*:

$$g(x,y)=(f*h)(x,y)=$$

$$\sum_{3\times 3}\begin{pmatrix} f(x-1,y-1)h(1,1) & f(x,y-1)h(0,1) & f(x+1,y-1)h(-1,1) \\ f(x-1,y)h(1,0) & f(x,y)h(0,0) & f(x+1,y)h(-1,0) \\ f(x-1,y+1)h(1,-1) & f(x,y+1)h(0,-1) & f(x+1,y+1)h(-1,-1) \end{pmatrix}$$

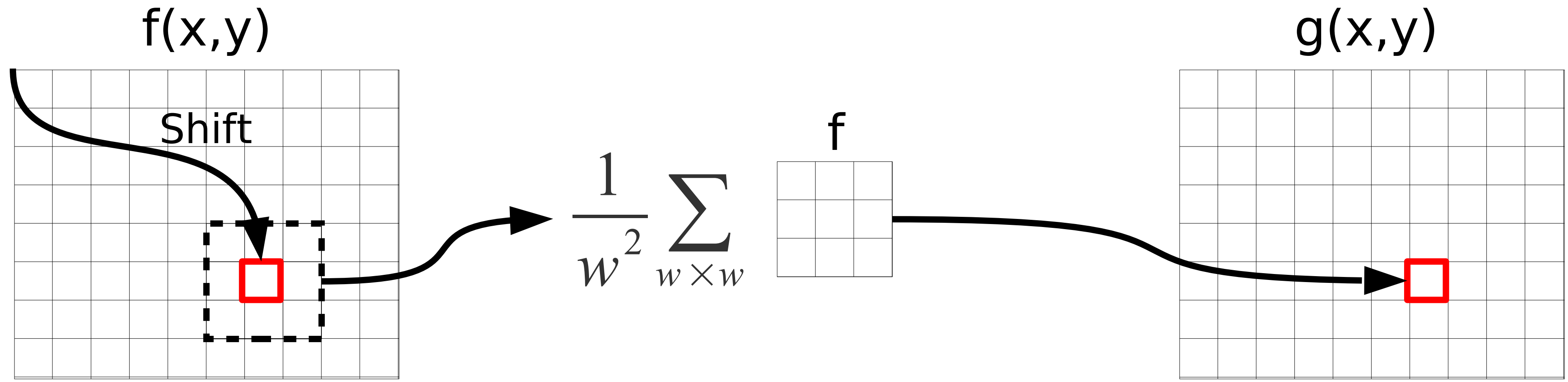
Filter Techniques

Example: Noise Suppression by Moving Average Filter

$$h(x,y) = \frac{1}{w^2} \begin{pmatrix} 1 & 1 & 1 & \cdots \\ 1 & 1 & 1 & \cdots \\ 1 & 1 & 1 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

(w x w Filter Kernel)

- Each pixel intensity is replaced by the local average...



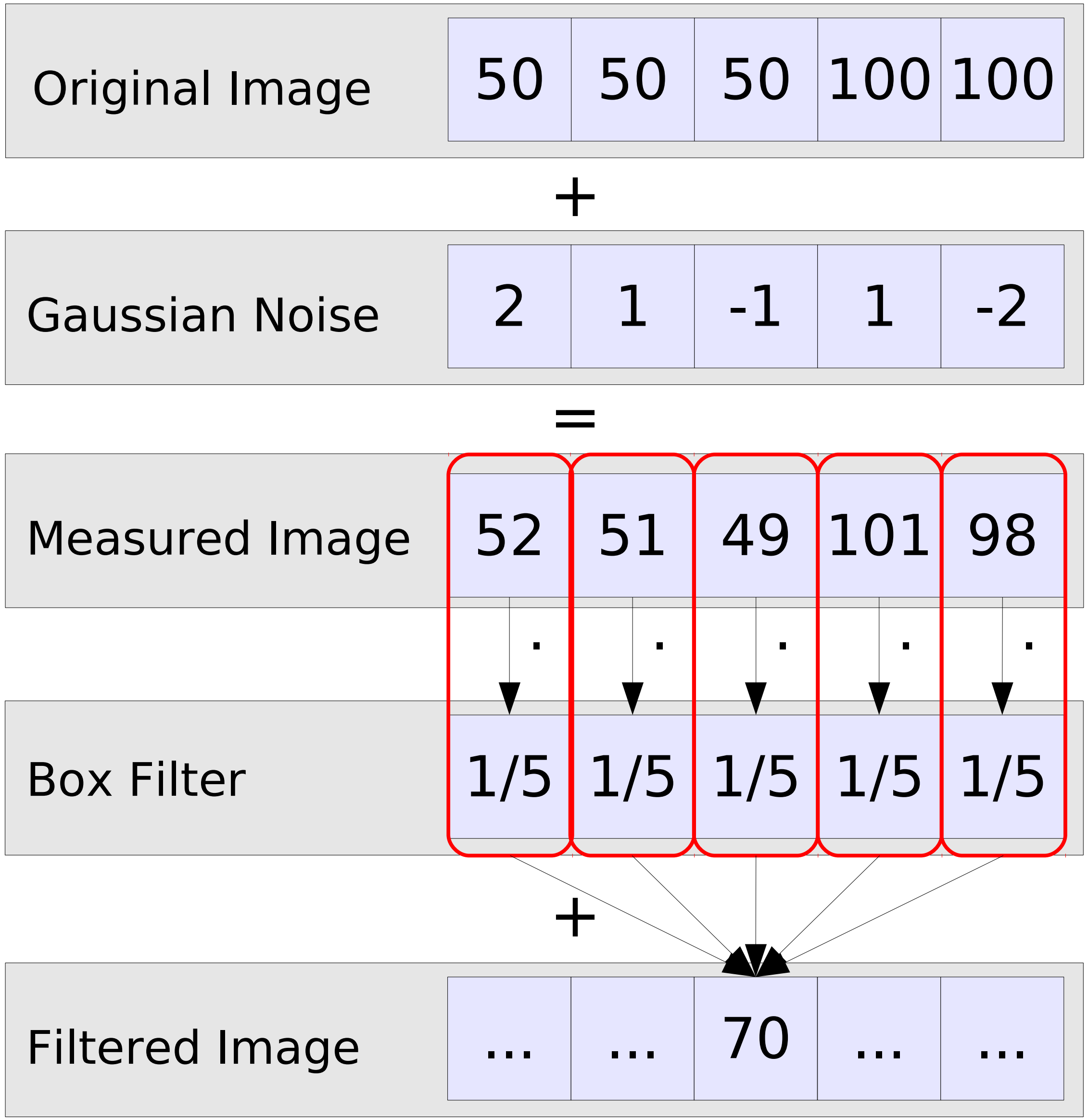
Filter Techniques

Example: Noise Suppression by Moving Average Filter

Original Image	100	100	100	100	100
+					
Gaussian Noise	2	1	-1	1	-2
=					
Measured Image	102	101	99	101	98
	·	·	·	·	·
Box Filter	1/5	1/5	1/5	1/5	1/5
+					
Filtered Image	...	...	100	...	...

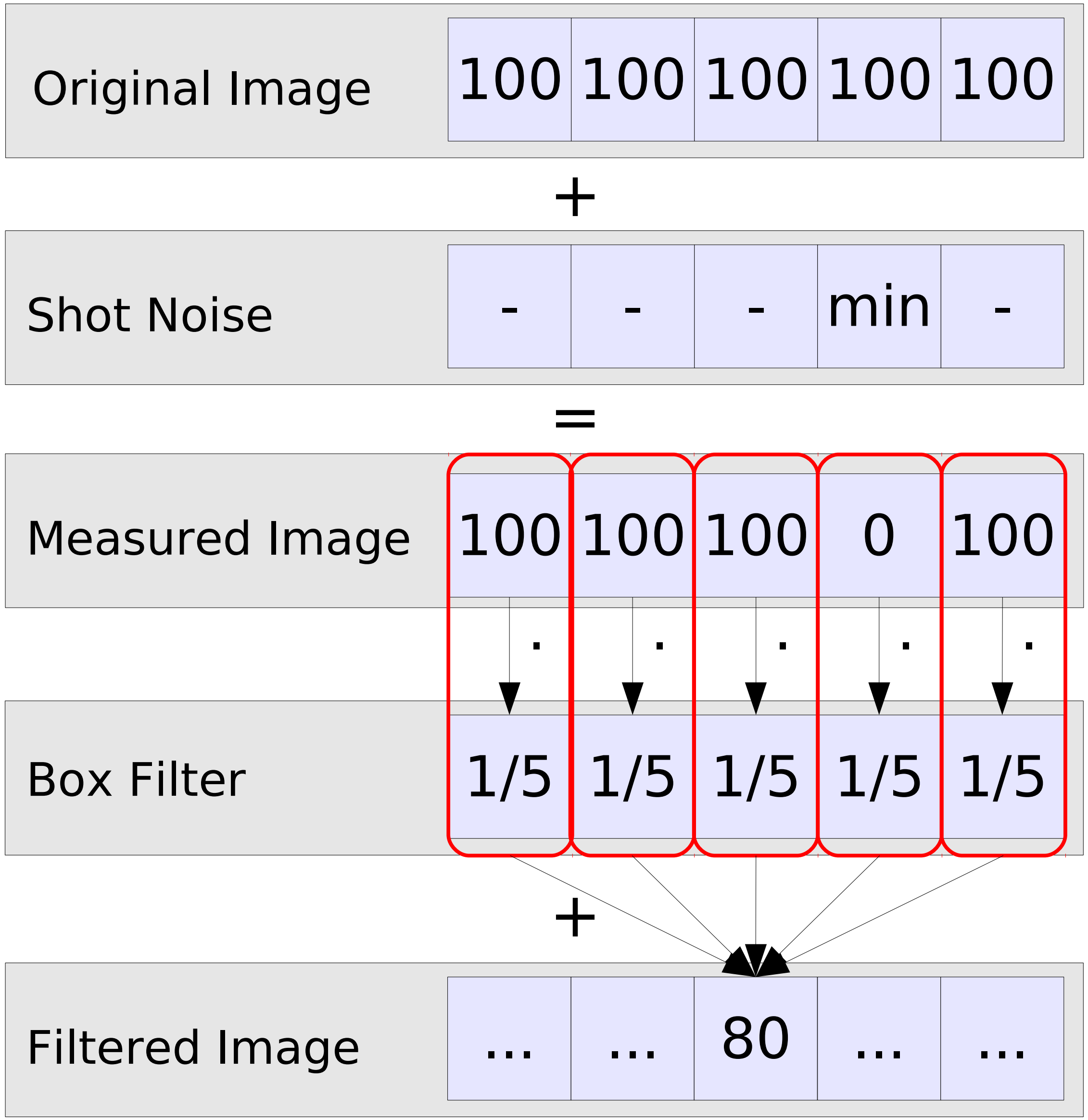
Filter Techniques

Example: Noise Suppression by Moving Average Filter



Filter Techniques

Example: Noise Suppression by Moving Average Filter



Filter Techniques

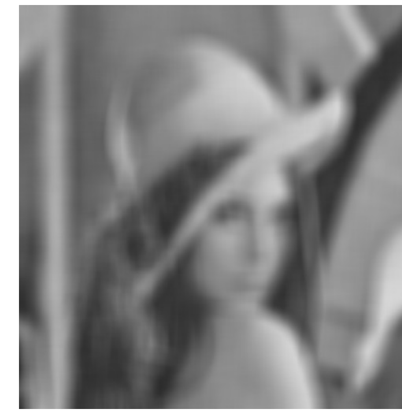
Example: Noise Suppression by Moving Average Filter

Gaussian Noise


f


w=5


w=11

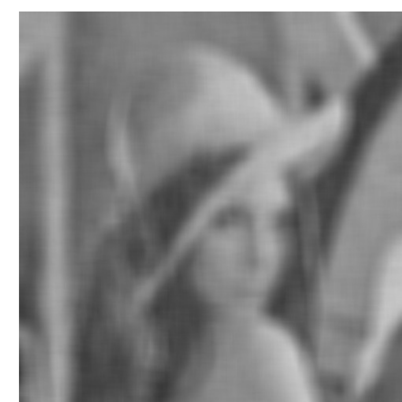

w=25

Shot Noise


f


w=5


w=11

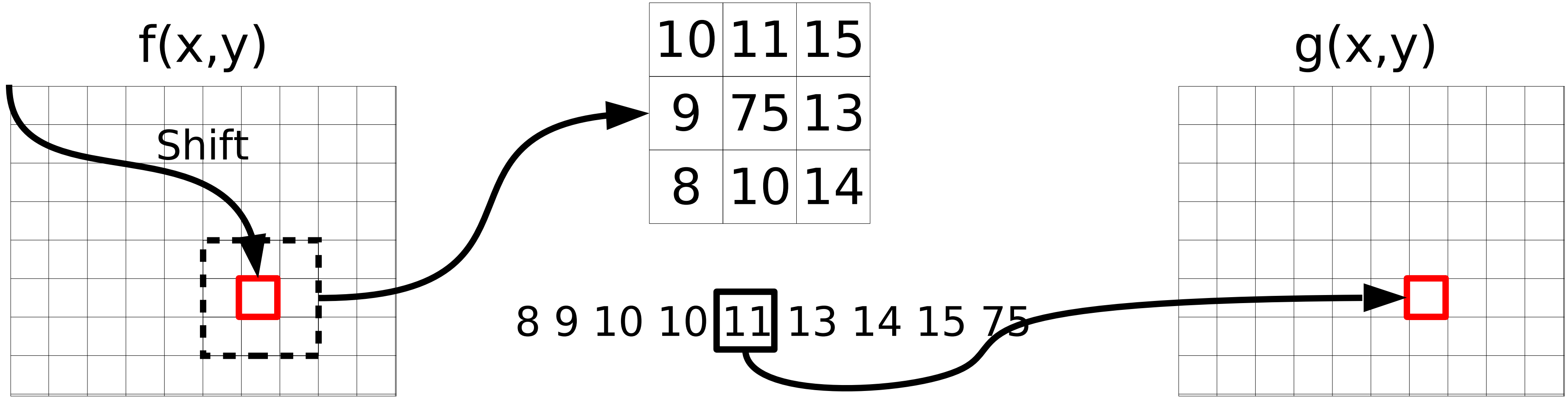

w=25

Filter Techniques

Example: Noise Suppression by Median Filter (NOTE: No convolution)

1. Consider intensities in a local NxN window
2. Sort intensities
3. Select middle value (median) as result

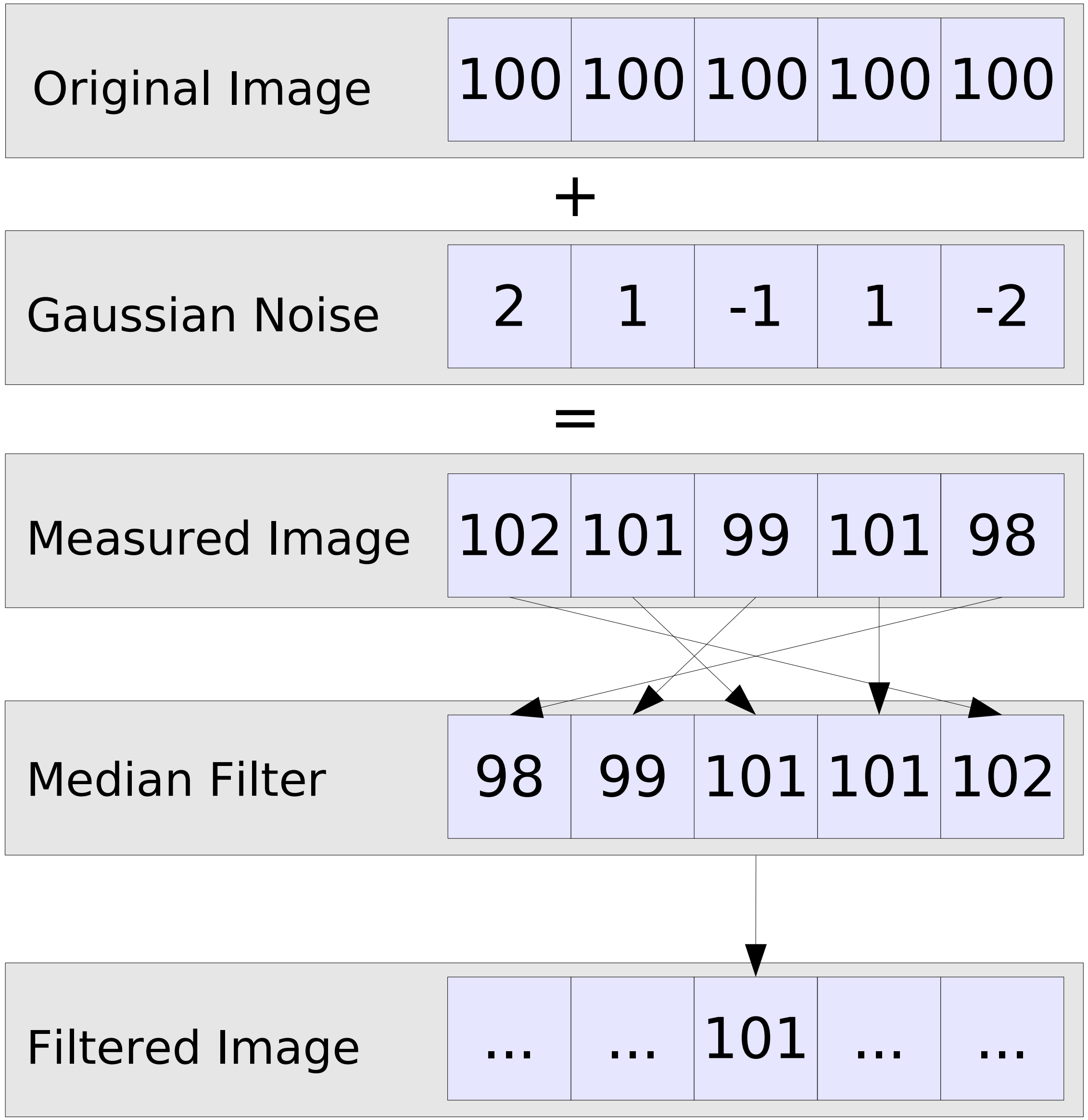
- Each pixel intensity is replaced by the local median...



- Effectively removes outliers
- Preserves sufficiently large (>> wxw) image structures

Filter Techniques

Example: Noise Suppression by Median Filter



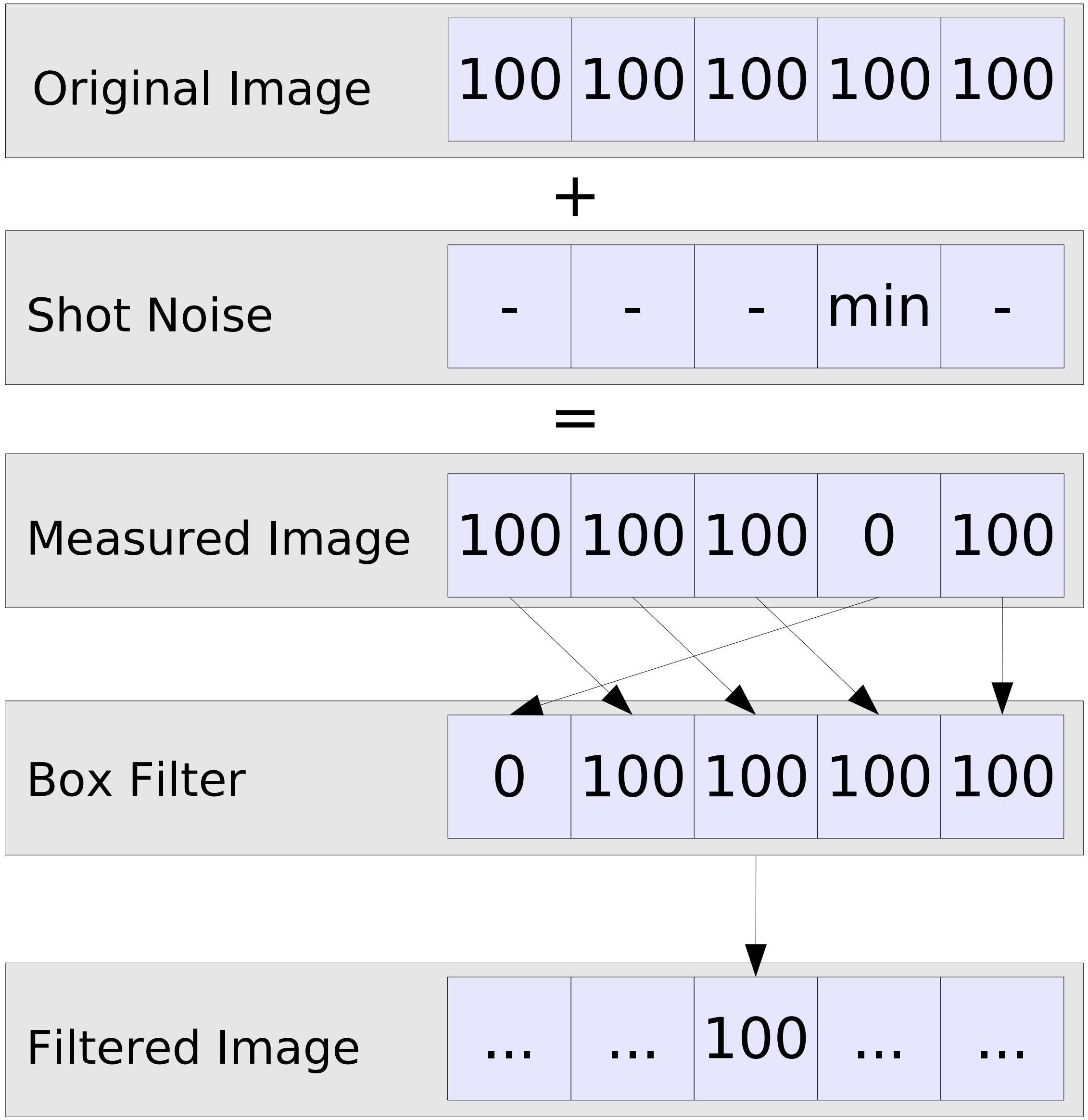
Filter Techniques

Example: Noise Suppression by Median Filter

Original Image	50	50	50	100	100
+					
Gaussian Noise	2	1	-1	1	-2
=					
Measured Image	52	51	49	101	98
Median Filter	49	51	52	98	101
Filtered Image	...	...	52	...	...

Filter Techniques

Example: Noise Suppression by Median Filter



Filter Techniques

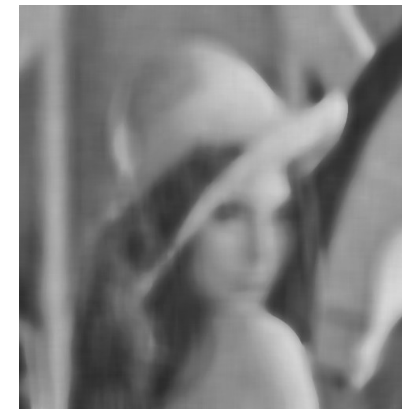
Example: Noise Suppression by Median Filter

Gaussian Noise

f

w=5

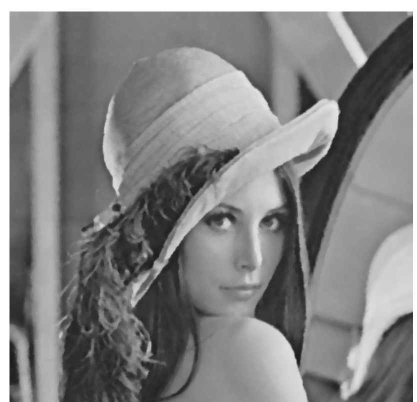
w=11



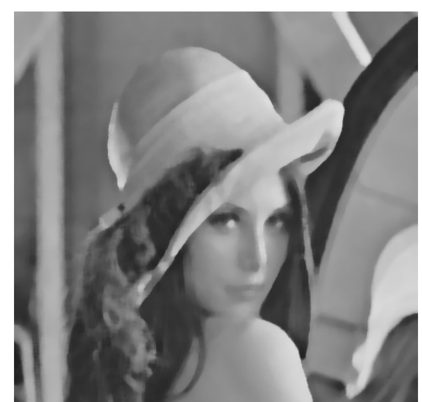
w=25

Shot Noise

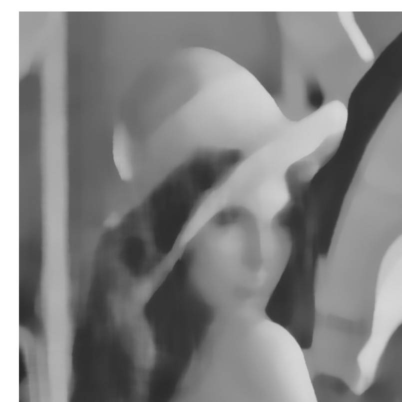
f



w=5



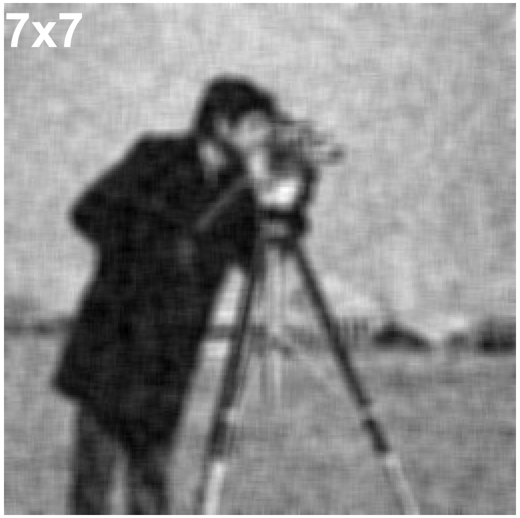
w=11



w=25

Noise Suppression vs. Resolution

Moving average filtering



Adaptive Smoothing

Moving Average Filter: $$m_k(x,y)=\begin{cases}1/k^2 & -k/2\leq x,y<k/2\\ 0 & \text{otherwise}\end{cases}$$

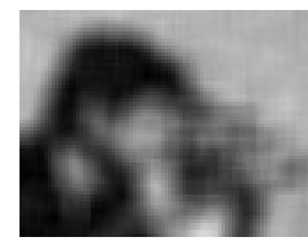
Modified Filtering:

$$g_n(x,y)=\begin{cases}(f*m_n)(x,y) & |(f*m_3)(x,y)-(f*m_n)(x,y)|\leq T\\ (f*m_3)(x,y) & \text{otherwise}\end{cases}$$

- Average unless filtered version departs too far from original
  - Largest discrepancies expected near strong edges
  - Threshold T and size *n* must be specified by the user!

Edge Preservation

$i \otimes (m_3 - m_9) \leq 40$



Excursus: Bilateral Filter

Spatial Weight:
$$h_{spat}(r,s)=\frac{1}{2\pi\sigma^2}\exp\left(-\frac{(r-\mu)^2+(s-\mu)^2}{2\sigma^2}\right)$$

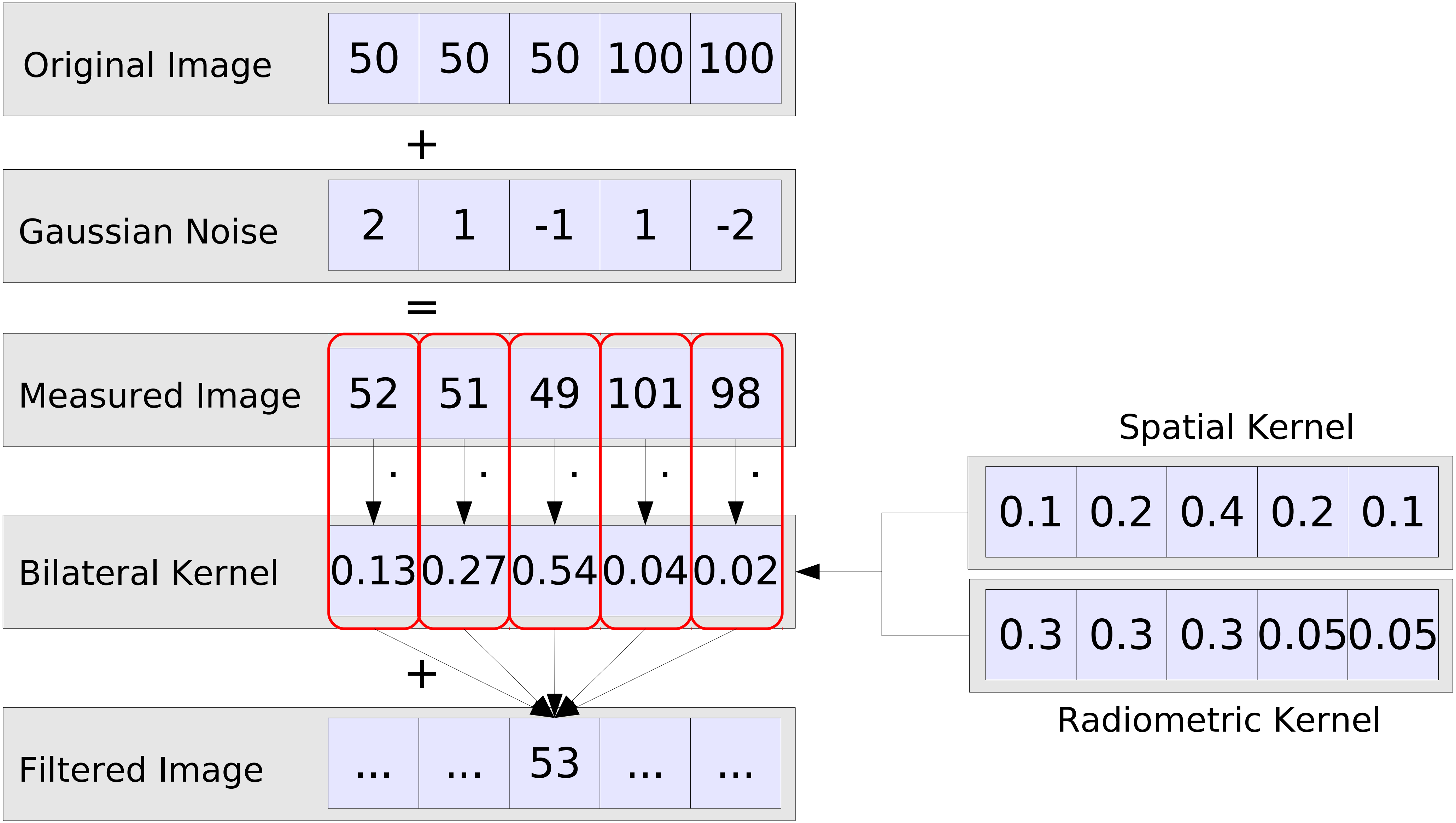
Radiometric Weight:
$$h_{rad}(p,q)=\frac{1}{2\pi\sigma_p^2}\exp\left(-\frac{(p-q)^2}{2\sigma_p^2}\right)$$

Combined:
$$h(r,s,f(x,y))=h_{spat}(r,s)\cdot h_{rad}(f(x,y))$$

Output:
$$g(\vec{x})=\frac{1}{Z(\vec{x})}\sum_{\vec{x}'\in N(\vec{x})}\exp\left(-\frac{(\vec{x}'-\vec{x})^2}{2\cdot\sigma_1^2}\right)\cdot\exp\left(-\frac{(f(\vec{x}')-f(\vec{x}))^2}{2\cdot\sigma_2^2}\right)\cdot f(\vec{x}')$$

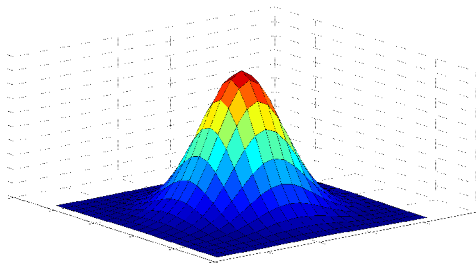
Filter Techniques

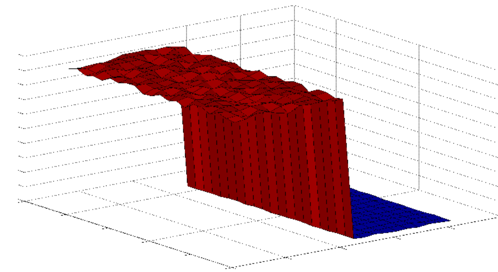
Example: Noise Suppression by Bilateral Filter

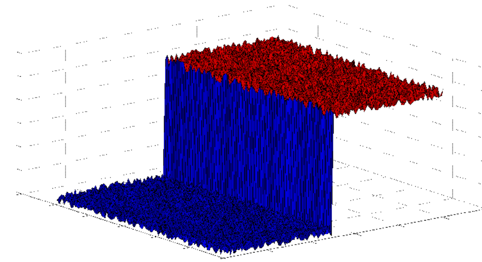


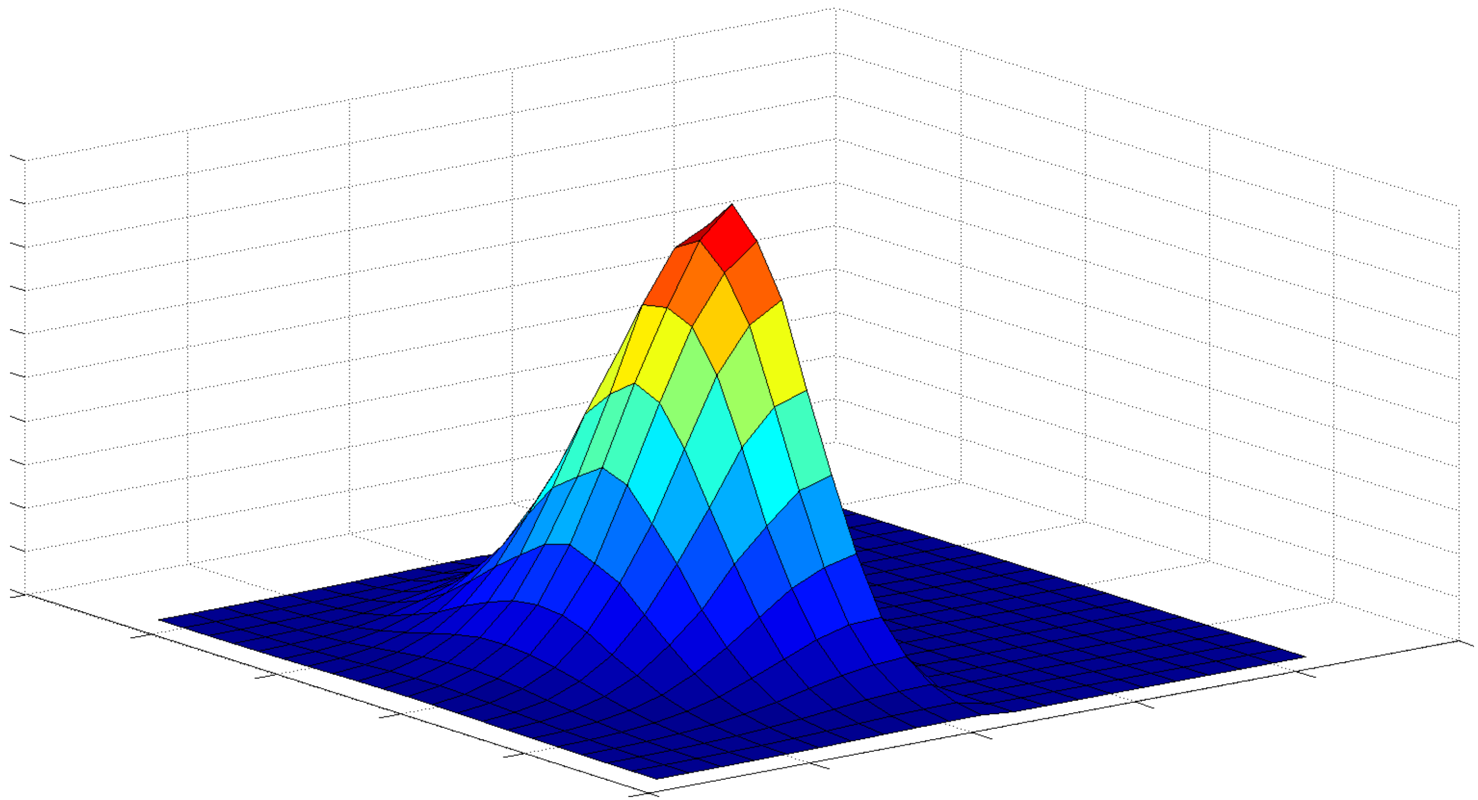
Excursus: Bilateral Filter

$$g(\vec{x}) = \frac{1}{Z(\vec{x})} \sum_{\vec{x}' \in \partial \vec{x}} \exp\left(-\frac{(\vec{x}' - \vec{x})^2}{2 \cdot \sigma_1^2}\right) \cdot \exp\left(-\frac{(f(\vec{x}') - f(\vec{x}))^2}{2 \cdot \sigma_2^2}\right) \cdot f(\vec{x})$$









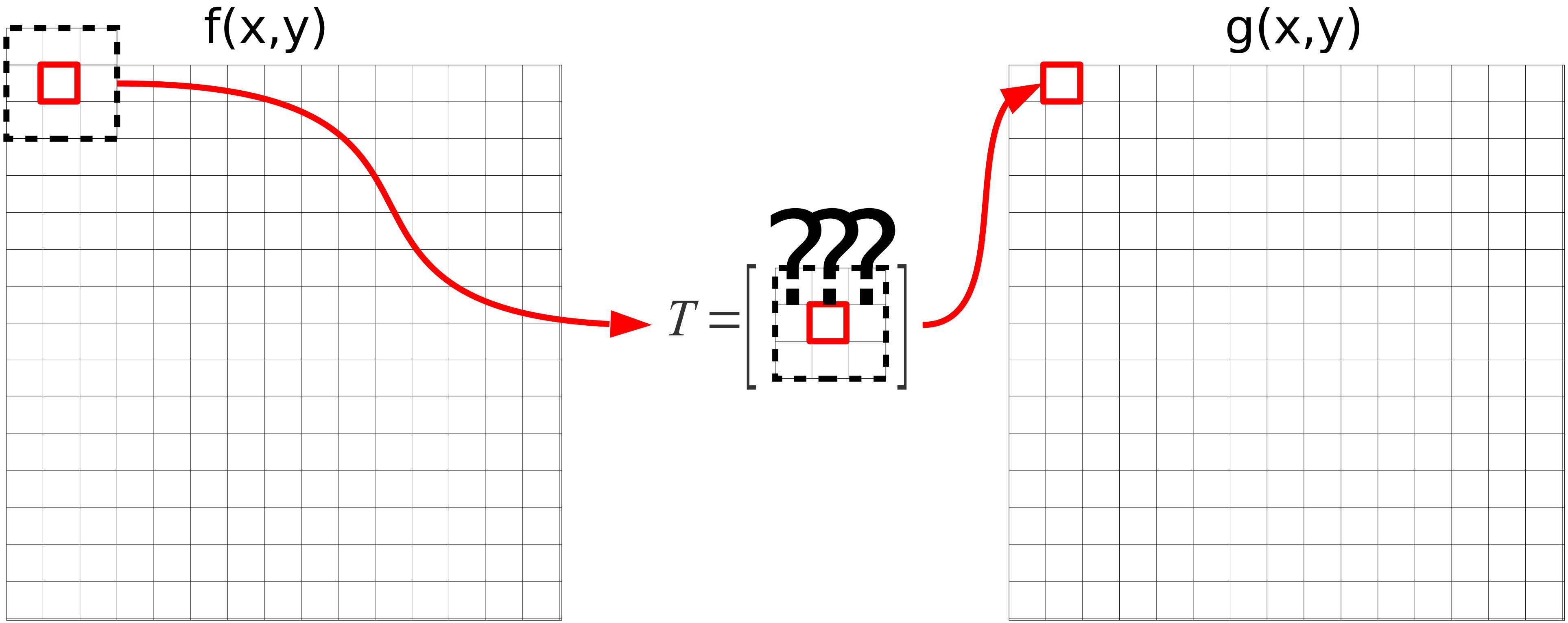
Excursus: Bilateral Filter

Original

Filter Result

C. Tomasi and R. Manduchi, "Bilateral Filtering for Gray and Color Images",
Proceedings of the 1998 IEEE International Conference on Computer Vision, Bombay, India

Border handling



- Problem: Unknown image values beyond image borders
- Possible solution
  - “Shrink” output image using only available information
  - Adapt kernel shape
  - Use “default” values (0, 255)
  - Use other image information (e.g. mirroring, wrapping)

2. Exercise – Noise Suppression

→ Part I - Theoretical

→ Part II - Practical

- Moving Average Filter
- Median Filter
- Adaptive Smoothing

→ <u>OPTIONAL</u>:
- Bilateral Filtering

2. Exercise – Theory

1. When should the median filter be applied to an image and when the moving average filter?

2. Explain your answer to question 1.

3. Is there a **general** better choice than the moving average filter?

4. Explain your answer to question 3.

2. Exercise - Given Functions

```
FILE: main.cpp
int main(int argc, char** argv)
```

- Main function
  - Usage:
    - dip2 generate path_to_original
      - Calls Dip2::generateNoiseImages(...)
      - Generates and saves noisy images
    - dip2 restorate
      - Uncomment Dip2::test(...) for basic testing!
      - Calls Dip2::run(...) for noise reduction

```
FILE: Dip2.cpp/h
void Dip2::generateNoisyImages(string fname)
```

- Applies two noise models to original image
- Saves both images (noiseType_1.jpg and noiseType_2.jpg)

2. Exercise - Given Functions

`void Dip2::run(string fname)`

- Loads both noisy images
- Tries to reduce noise by different methods
  - Calls Dip2::noiseReduction(..)
  - **TO DO:**
    Set parameters of Dip2::noiseReduction(..) according to the type of noise you observe in the two images
- Saves result images

```
FILE: Dip2.cpp
void Dip2::run(void){
    ...
    Mat noise1 = imread("noiseType_1.jpg", 0);
    Mat noise2 = imread("noiseType_2.jpg", 0);
    ...
    Mat restorated1 = noiseReduction(noise1, "", 1);
    Mat restorated2 = noiseReduction(noise2, "", 1);
    ...
}
```

2. Exercise - Given Functions

```
Mat Dip2::noiseReduction(Mat& src, string method,
                         int kSize, double param)
```

- Parameter:
  - **`src`** : noisy source image
  - **`method`** : defines method to be used
    - median, average, adaptive, bilateral
  - **`kSize`** : Kernel size
  - **`param`** : adaptive smoothing: threshold
    bilateral filter: std-dev of radiometric kernel
  - **`return`** : noised reduced output image

- Calls
  - averageFilter(...), medianFilter(...), adaptiveFilter(...), or bilateralFilter(...)

2. Exercise – To Do

```
Mat Dip2::spatialConvolution(Mat& src, Mat& kernel)
```

- Parameter:
  - → **`src`** : source image
  - → **`kernel`** : kernel of the convolution
  - → **`return`** : output image

- Applies convolution in spatial domain

- One method of border handling: size(src) == size(return)

- Do **NOT** use convolution functions of OpenCV

2. Exercise – To Do

`Mat Dip2::averageFilter(Mat& src, int kSize)`

- Parameter:
  - → **`src`** : noisy source image
  - → **`kSize`** : Kernel size
  - → **`return`** : output image

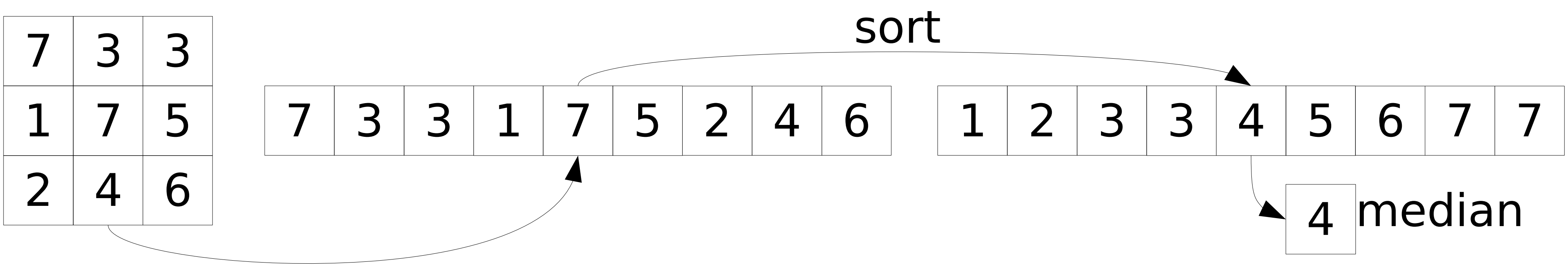
- Uses convolution to calculate local average
- Calls spatialConvolution(...)

`Mat Dip2::medianFilter(Mat& src, int kSize)`

- Parameter:
  - → **`src`** : noisy source image
  - → **`kSize`** : Kernel size
  - → **`return`** : output image

- Applies local median filtering

2. Exercise - Median




```
#include <iostream>
#include <algorithm>

int main() {
  int array[] = { 23, 5, -10, 0, 0, 321, 1, 2, 99, 30 };
  int elements = sizeof(array) / sizeof(array[0]);
  std::sort(array, array + elements);
  for (int i = 0; i < elements; ++i)
     std::cout << array[i] << ' ';
}
```

2. Exercise – To Do

`Mat adaptiveFilter(Mat& src, int kSize, double threshold)`

- Parameter:
  - **`src`** : noisy source image
  - **`kSize`** : kernel size
  - **`threshold`** : smooth only if difference is below this value
  - **`return`** : output image

- Uses moving average filter, but preserves edges

- Calls averageFilter(...)

2. Exercise – Optional

```
Mat bilateralFilter(Mat& src, int kSize, double sigma)
```

- Parameter:
  - **`src`** : noisy source image
  - **`kSize`** : size of spatial kernel
    → Calculate standard-deviation accordingly
  - **`sigma`** : Standard-Deviation of radiometric kernel
  - **`return`** : output image

- Implements bilateral filter

Deadline: 22nd May