Image Filtering Part-1

Dr. Mohamed Waleed Fakhr

2023

2d-convolution, Box and Gaussian Filters, Bilateral Filters, Non-linear filters, Median Filters

Overview of Filtering

- Convolution
- Box Filter
- Gaussian Filtering

Motivation: Noise reduction

- Given a camera and a still scene, how can you reduce noise?

Take lots of images and average them!

What's the next best thing?

Source: S. Seitz

Moving average

- Let's replace each pixel with a *weighted* average of its neighborhood
- The weights are called the *filter kernel*
- What are the weights for the average of a 3x3 neighborhood?

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

"box filter"

Source: D. Lowe

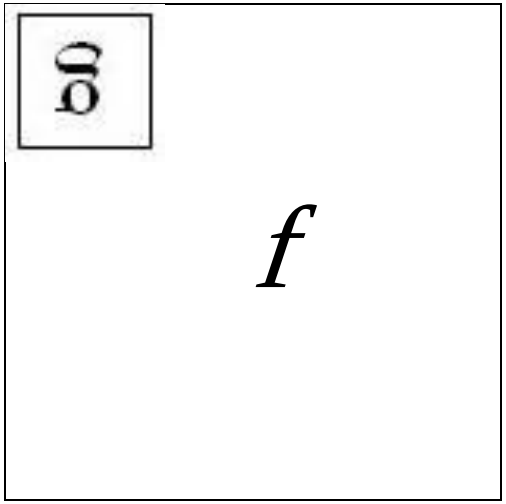
procedure

- It centers the window to the pixel of interest
- It multiplies the weights with the pixel values
- Then, it adds the multiplication values
- Then, puts the result in place of the old pixel value
- This is called convolution

Defining Convolution

- Let f be the image and g be the kernel. The output of convolving f with

$$(f * g)[m,n] = \sum_{k,l} f[m-k, n-l]\, g[k,l]$$

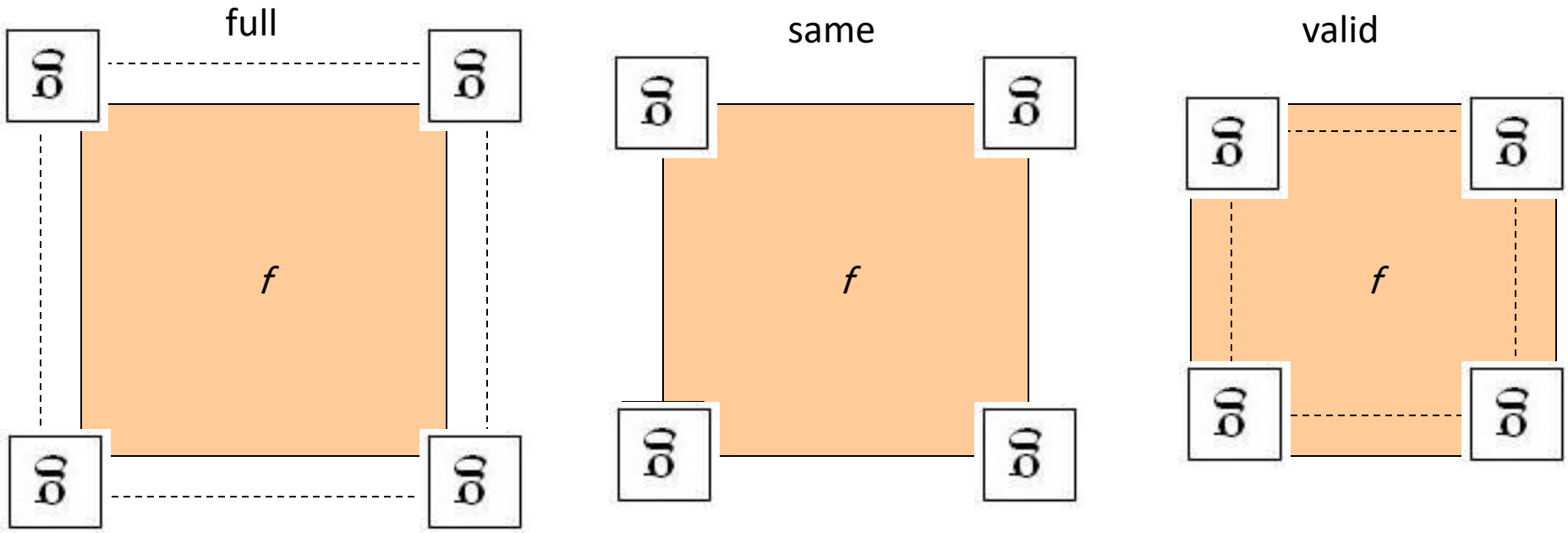


- Convention: kernel is "flipped"
- MATLAB: conv2 (also imfilter)

Source: F. Durand

Important details

- What is the size of the output?
- MATLAB: conv2(f, g,*shape*)
  - *shape* = 'full' : output size is sum of sizes of f and g minus1
  - *shape* = 'same' : output size is same as f
  - *shape* = 'valid' : output size is difference of sizes of f and g plus 1



Cont.

- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods (MATLAB):
    - clip filter (black): imfilter(f, g, 0)
    - wrap around: imfilter(f, g, 'circular')
    - copy edge: imfilter(f, g, 'replicate')
    - reflect across edge: imfilter(f, g, 'symmetric')

Source: S. Marschner

Matlab example: conv2

```
x = imread('baboon.bmp');
xg=rgb2gray(x);
wind=ones(4,4);
wind=wind/16;
xg=im2double(xg);
yfull=conv2(xg,wind,'full');
ysame=conv2(xg,wind,'same');
yvalid=conv2(xg,wind,'valid');
subplot(2,2,1), imshow(xg);
subplot(2,2,2), imshow(yfull);
subplot(2,2,3), imshow(ysame);
subplot(2,2,4), imshow(yvalid);
Size(yfull), size(ysame), size(yvalid)
```

Matlab example using imfilter

```
clear all;

K=16;
SS = K*K;

x = imread('baboon.bmp');
xg=rgb2gray(x);
wind=ones(K,K);
wind=wind/SS;
xg=im2double(xg);

yfull=imfilter(xg,wind,'full','conv');
ysame=imfilter(xg,wind,'same','conv');

%Both used zero padding

%Now we look at different options
yfull1=imfilter(xg,wind,'full','conv');
yfull2=imfilter(xg,wind,'full','conv','symmetric');
yfull3=imfilter(xg,wind,'full','conv','replicate');
yfull4=imfilter(xg,wind,'full','conv','circular');


subplot(2,2,1), imshow(yfull1);
subplot(2,2,2), imshow(yfull2);
subplot(2,2,3), imshow(yfull3);
subplot(2,2,4), imshow(yfull4);
```

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	1	0
0	0	0

Filtered
(no change)

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0

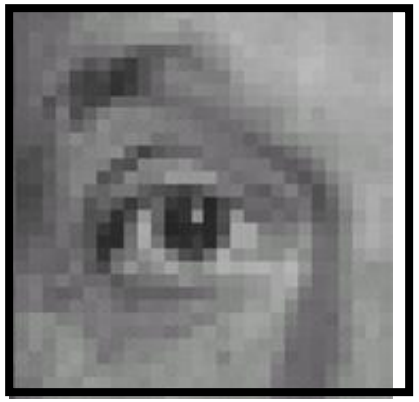
?

Source: D. Lowe

Practice with linear filters

Original

0	0	0
0	0	1
0	0	0



Shifted left
By 1 pixel

Source: D. Lowe

Practice with linear filters

Original

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

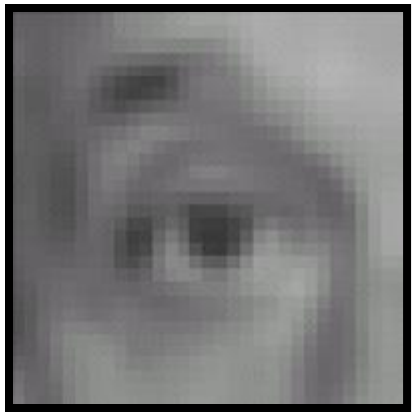
?

Source: D. Lowe

Practice with linear filters

Original

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



Blur (with a box filter)

Source: D. Lowe

Practice with linear filters

Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = ?$$

Source: D. Lowe

Practice with linear filters

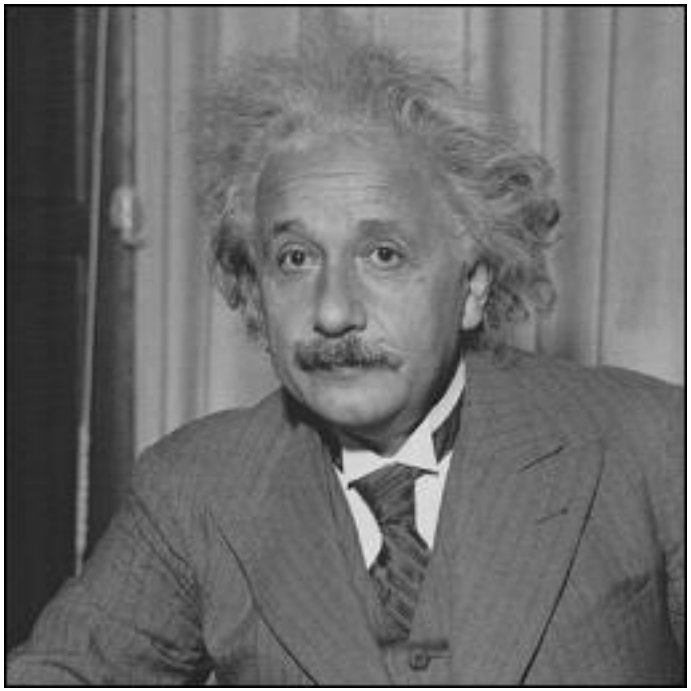
Original

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

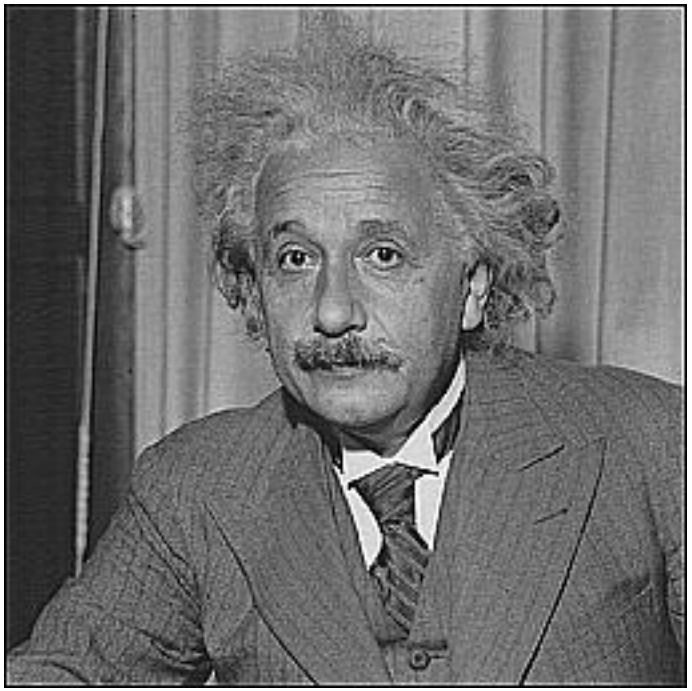
Sharpening filter

Source: D. Lowe

Sharpening



before



after

Slide credit: Bill Freeman

Sources of Image Noises

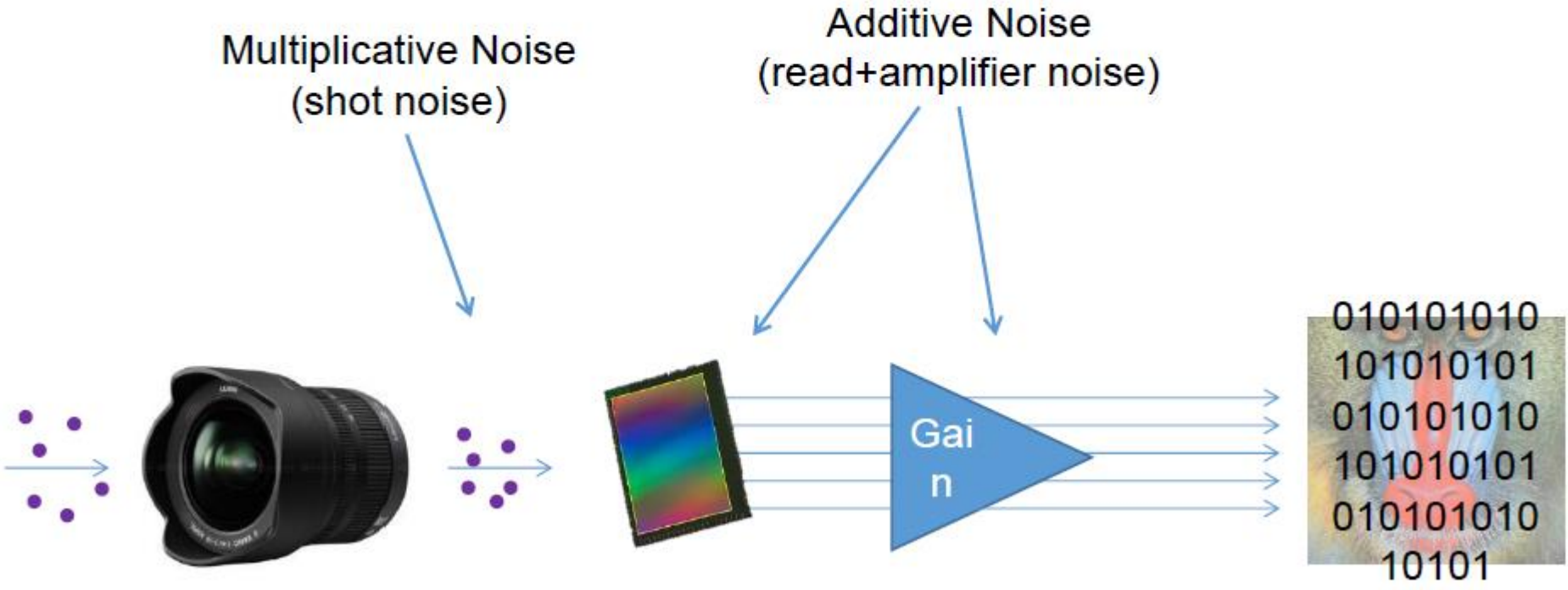
- Metrics

Mean Square Error

$$\text{MSE} = \|\hat{x} - x\|_2^2$$

Peak Signal-to-Noise ratio

$$\text{PSNR} = 20log_{10}\frac{255}{\sqrt{\text{MSE}}}$$



Square Box Generates Defects

- Axis-aligned streaks
- Blocky results

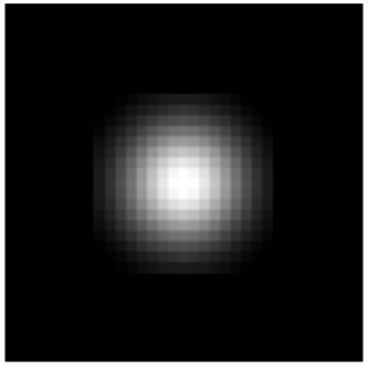
output

input

Strategy to Solve these Problems

- Use an isotropic (*i.e.* circular) window.
- Use a window with a smooth falloff.

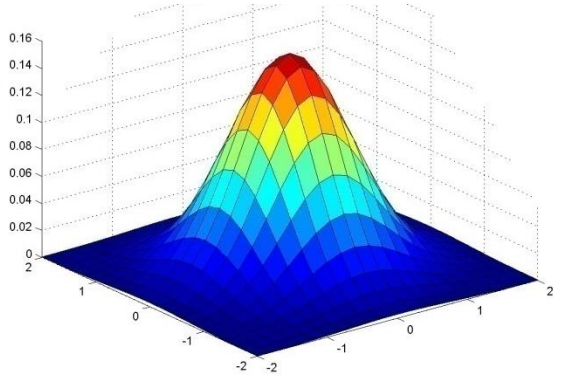
box window

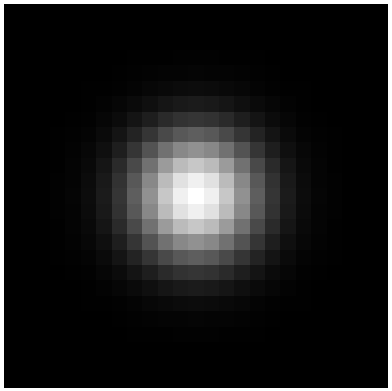


Gaussian window

Gaussian Kernel (aka: Gaussian Blur)

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$





0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

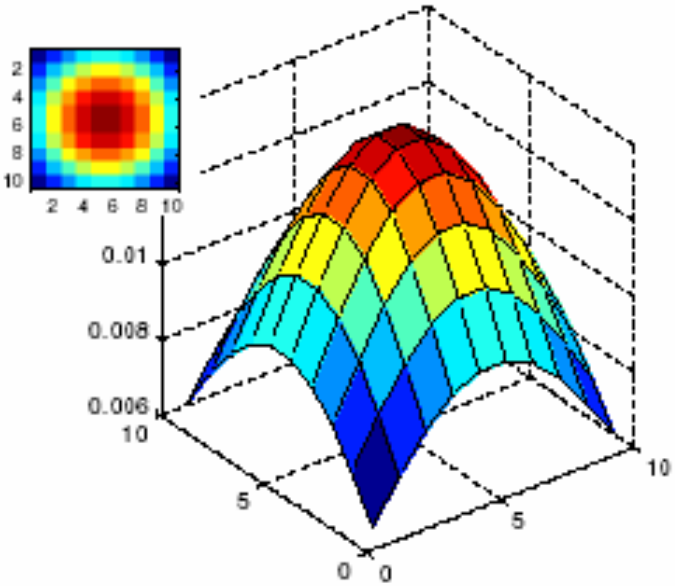
5 x 5, $\sigma = 1$

- Constant factor at front makes volume sum to 1 (can be ignored, as we should re-normalize weights to sum to 1 in any case)

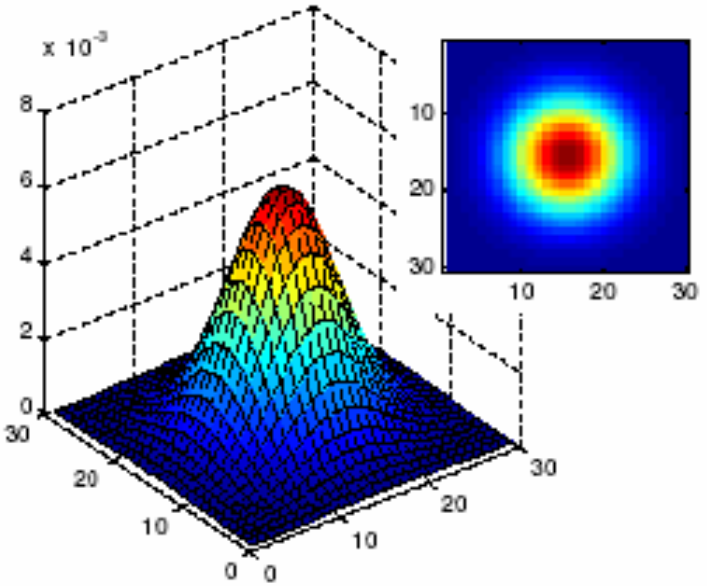
Source: C. Rasmussen

Choosing kernel width

- Gaussian filters have infinite support, but discrete filters use finite kernels



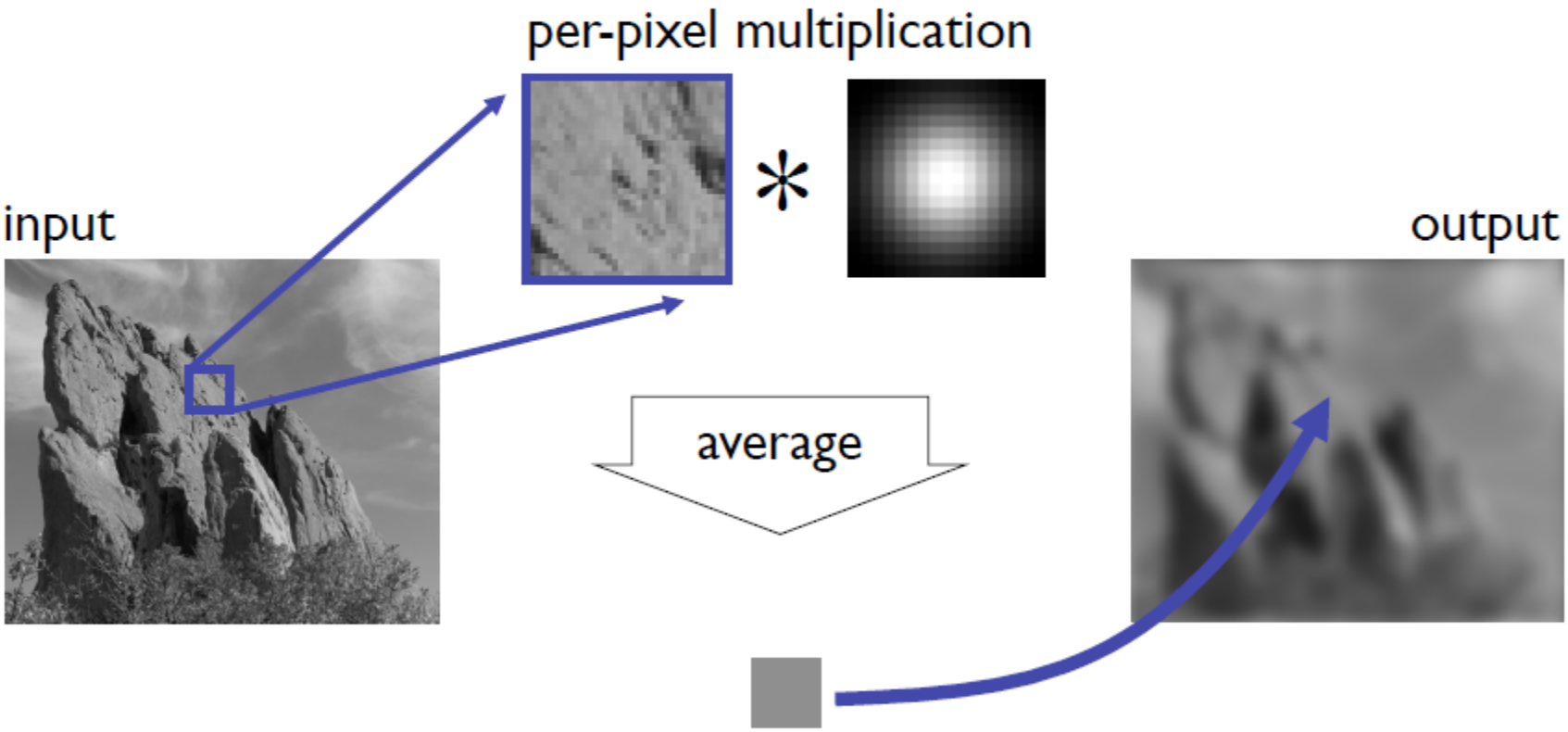
σ = 5 with 10x10 kernel



σ = 5 with 30x30 kernel

Source: K. Grauman

Gaussian Blur




input


box average


Gaussian blur

Gaussian filters

- Remove “high-frequency” components from the image (low-pass filter)
- Usually better than the box (average) filter since it focuses on the most neighboring pixels more than the father ones.

Source: K. Grauman

Gaussian Filter Matlab

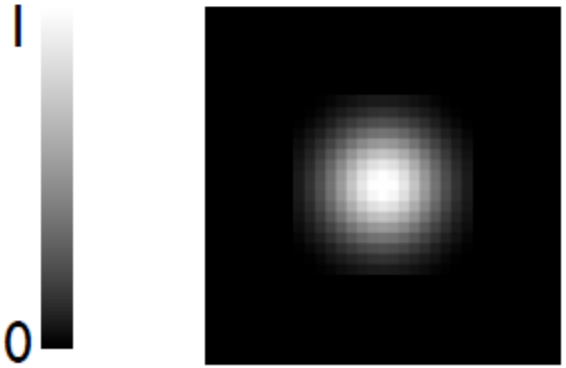
```
clear all;

x = imread('baboon.bmp');
xg=rgb2gray(x);
xg=im2double(xg);



H = fspecial('gaussian',10,3);
y = imfilter(xg,H,'replicate','same','conv');

%OR use directly the im-gaussian-filtering function:
y = imgaussfilt(xg,sigma)   %sigma is the st. dev. Of the Gaussian

subplot(2,1,1), imshow(xg);
subplot(2,1,2), imshow(y);
```

Equation of Gaussian Blur

Same idea: **weighted average of pixels**.

$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in S} G_{\sigma}(\|\mathbf{p}-\mathbf{q}\|)\, I_{\mathbf{q}}$$

normalized
Gaussian function

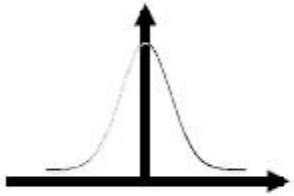


Spatial Parameter

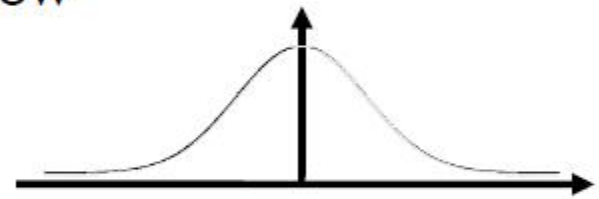
input

$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in S} G_{\sigma}(\| \mathbf{p} - \mathbf{q} \|) I_{\mathbf{q}}$$

size of the window



small *s*

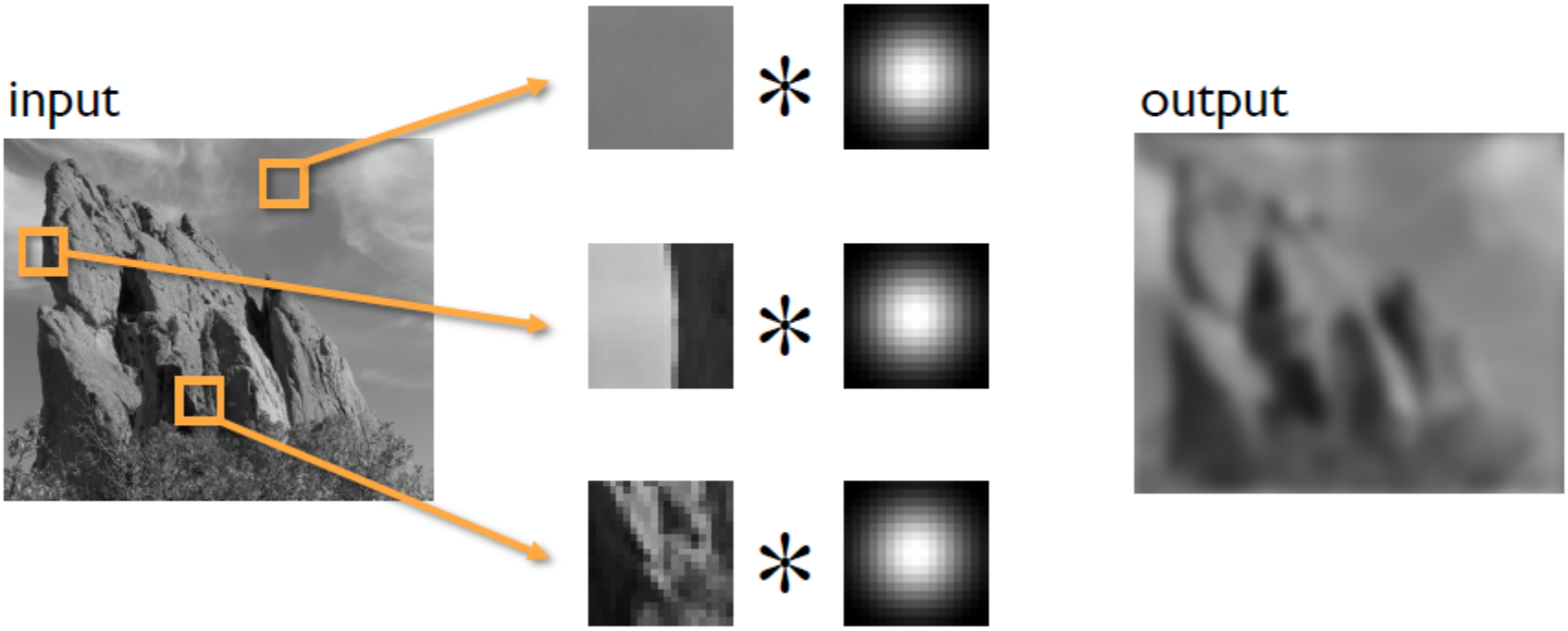


large *s*

limited smoothing

strong smoothing

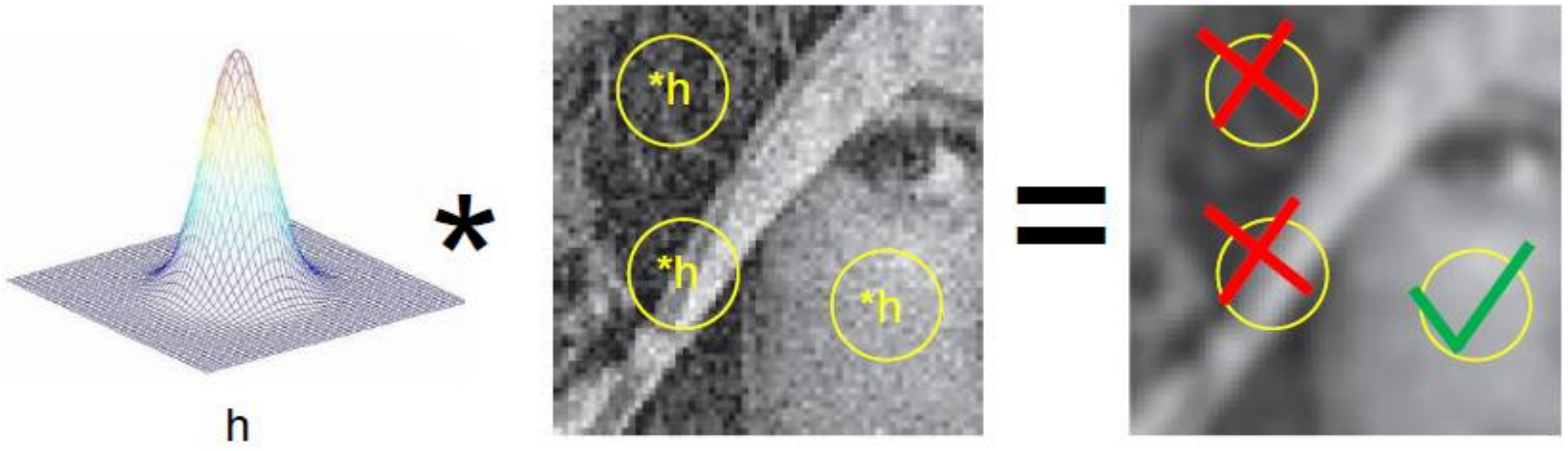
Blur Comes from Averaging across Edges



Same Gaussian kernel everywhere.

Problem with Gaussian Smoothing

- Loss of fine /high freq details at texture rich region



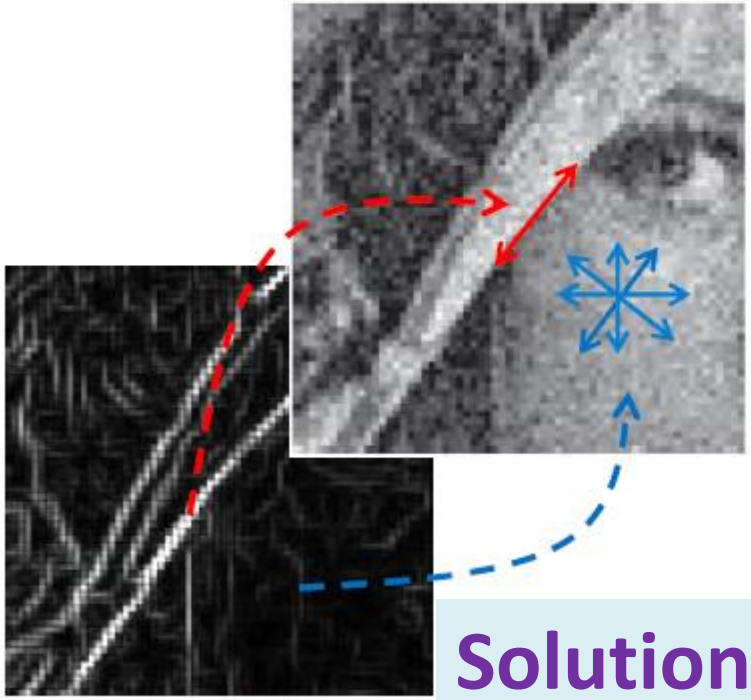
h

$$\hat{x}(i) = \frac{1}{C_i} \sum_j y(j) e^{-\frac{\|i-j\|^2}{2\sigma^2}}$$

Problem: Smoothing only based on spatial closeness

Edge Preserving Filtering

- Edges ⇒ smooth only along edges
- "Smooth" regions ⇒ smooth isotropically



gradient

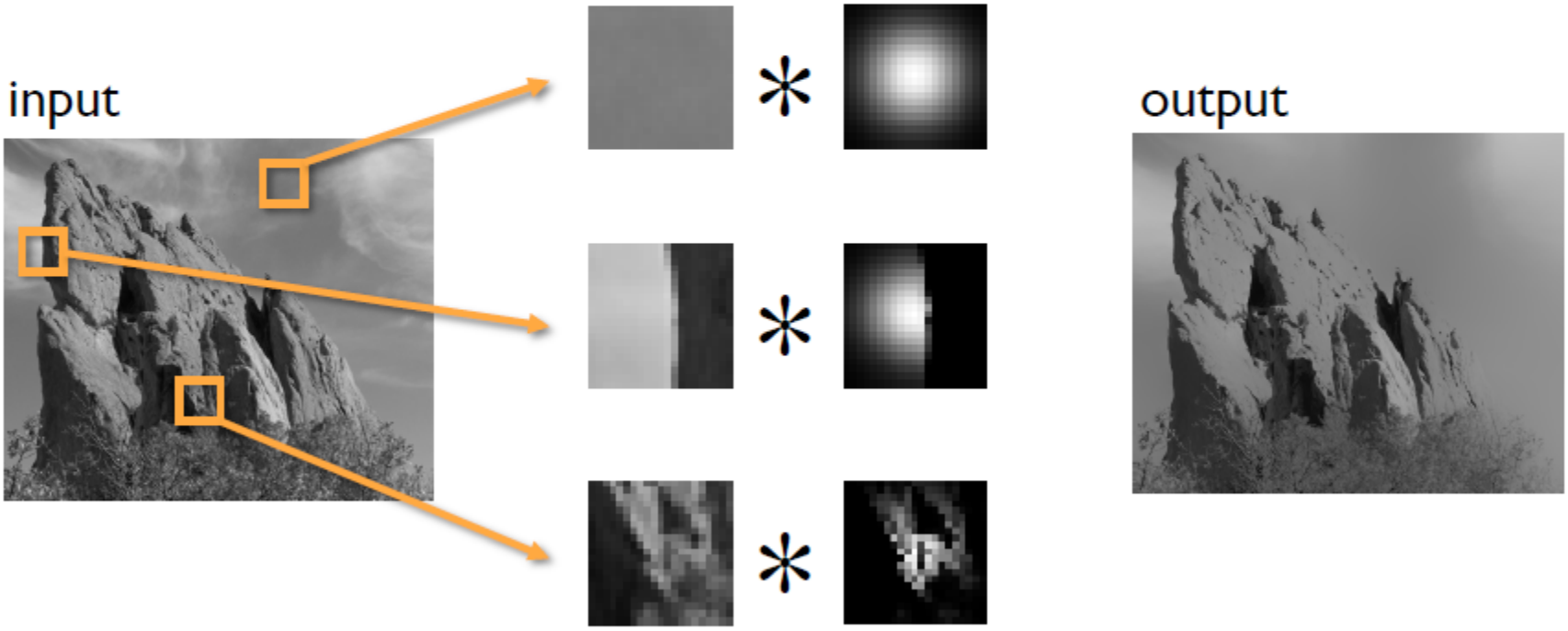
Solution: use Bilateral Filters

What Is Bilateral Filter?

- Bilateral
  - Affecting or undertaken by two sides equally
- Property:
  - Convolution filter
  - Smooth image but preserve edges
  - Operates in the domain (geometry) and the range (photometric) of image

Gaussian filter Bilateral filter

Bilateral Filter
No Averaging across Edges



The kernel shape depends on the image content.

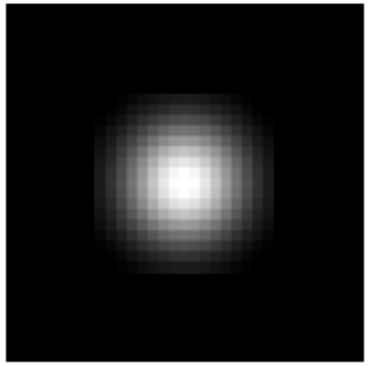
Bilateral Filter Definition: an Additional Edge Term

Same idea: **weighted average of pixels**.

$$BF[I]_{\mathbf{p}} = \underbrace{\frac{1}{W_{\mathbf{p}}}}_{\text{new}} \sum_{\mathbf{q}\in S} \underbrace{G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)}_{\text{not new}} \underbrace{G_{\sigma_r}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)}_{\text{new}} I_{\mathbf{q}}$$

normalization factor

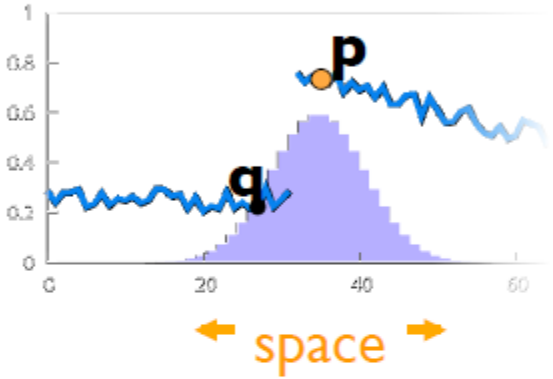
space weight



range weight

Gaussian Blur and Bilateral Filter

Gaussian blur

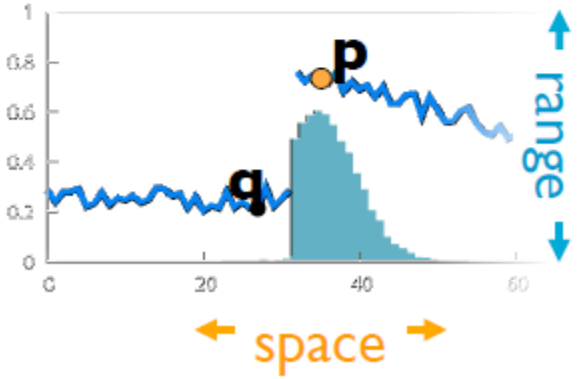


$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q}\in S} G_{\sigma}(\|\mathbf{p}-\mathbf{q}\|) I_{\mathbf{q}}$$

space

Bilateral filter

[Aurich 95, Smith 97, Tomasi 98]



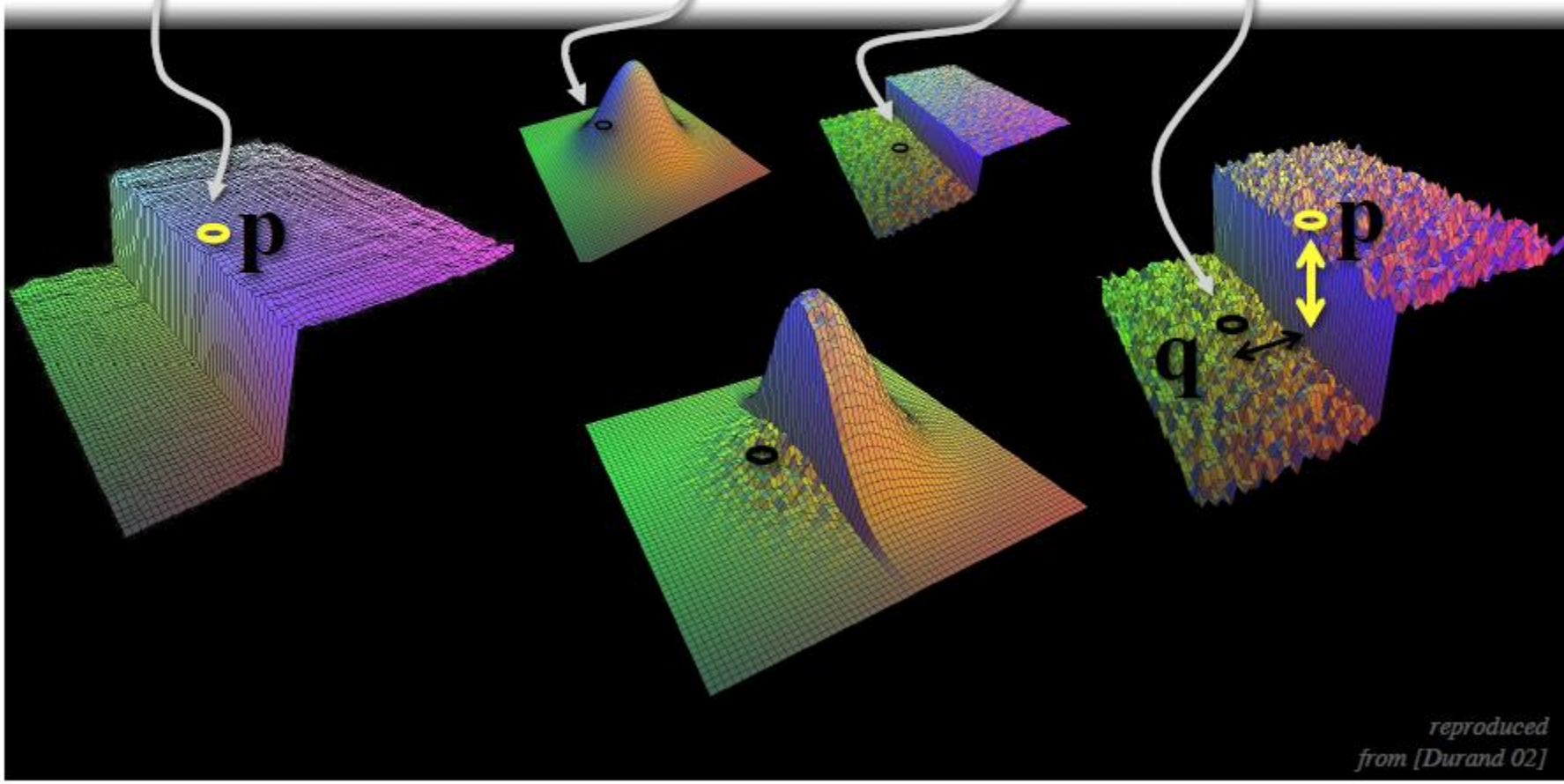
$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|) G_{\sigma_r}(|I_{\mathbf{p}}-I_{\mathbf{q}}|) I_{\mathbf{q}}$$

space range

normalization

Bilateral Filter on a Height Field

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}}-I_{\mathbf{q}}|)\, I_{\mathbf{q}}$$



reproduced from [Durand 02]

Exploring the Parameter Space

input

$s_r = 0.1$

$s_r = 0.25$

$s_r = \infty$ (Gaussian blur)

$s_s = 2$

$s_s = 6$

$s_s = 18$

Varying the Range Parameter

input

$s_r = 0.1$ | $s_r = 0.25$ | $s_r = \infty$ (Gaussian blur)

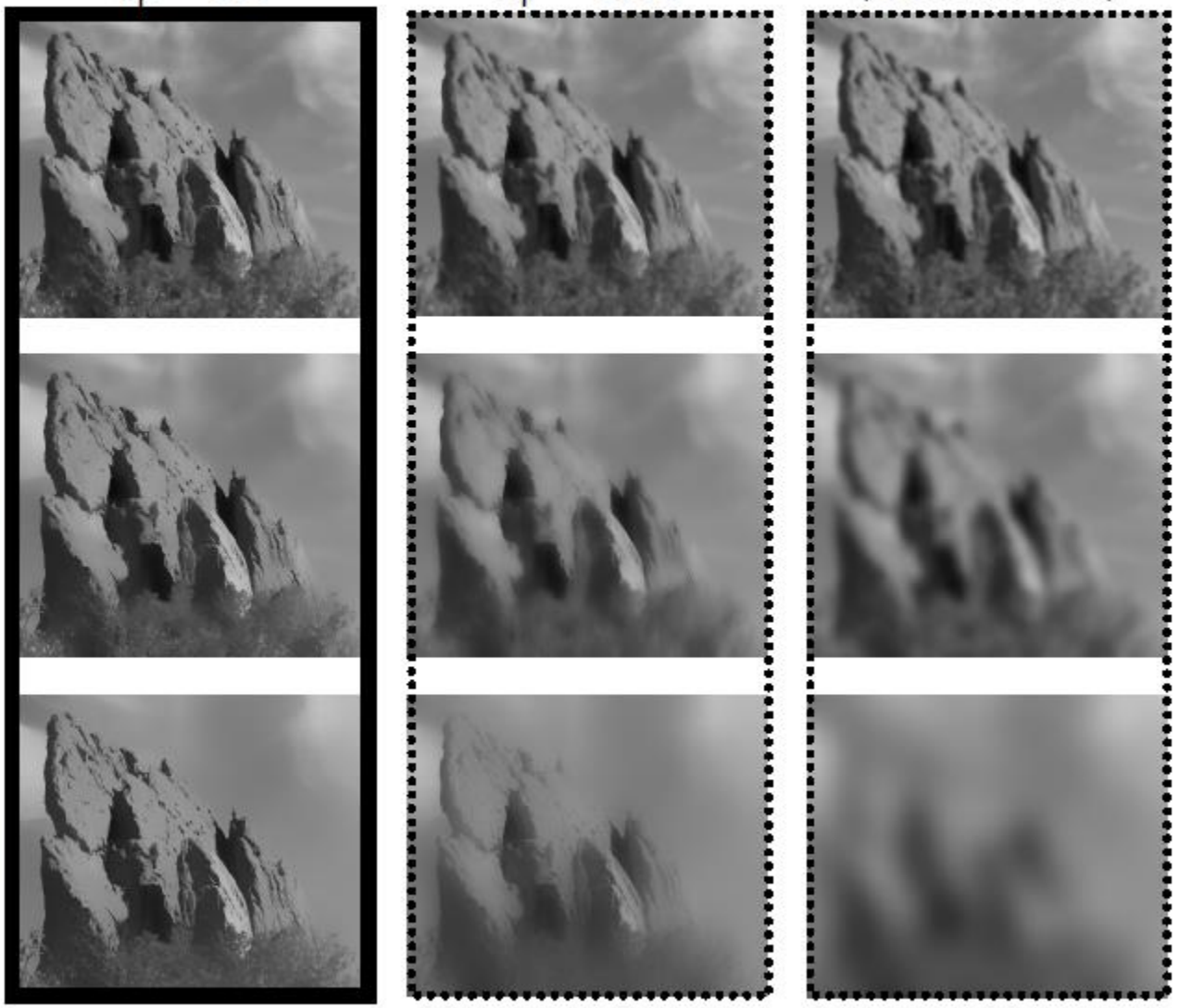
$s_s = 2$

$s_s = 6$

$s_s = 18$

Varying the Space Parameter

input



Other methods for Image De-Noising

- **Non-Local Mean Filtering (NLM)**
- Adaptive Median Filters
- Local Mode Filtering
- Deep neural networks (deep auto-encoders, GAN, etc.)

Non-Local-Means Filter

NL-Means Filter (Buades 2005)

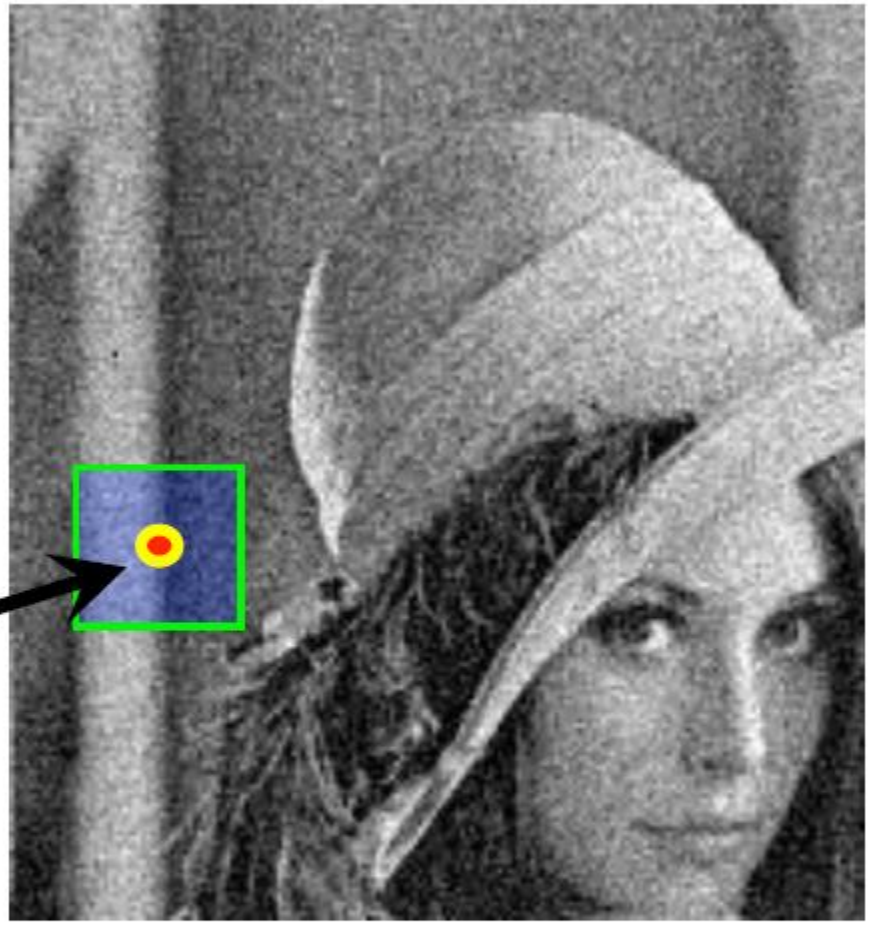
- Same goals: 'Smooth within Similar Regions'

- **KEY INSIGHT**: Generalize, extend 'Similarity'
  - **Bilateral:**
    Averages neighbors with **similar intensities**;
  - **NL-Means:**
    Averages neighbors with **similar neighborhoods!**

NL-Means Method: Buades (2005)

- For each and every pixel **p:**

NL-Means Method: Buades (2005)

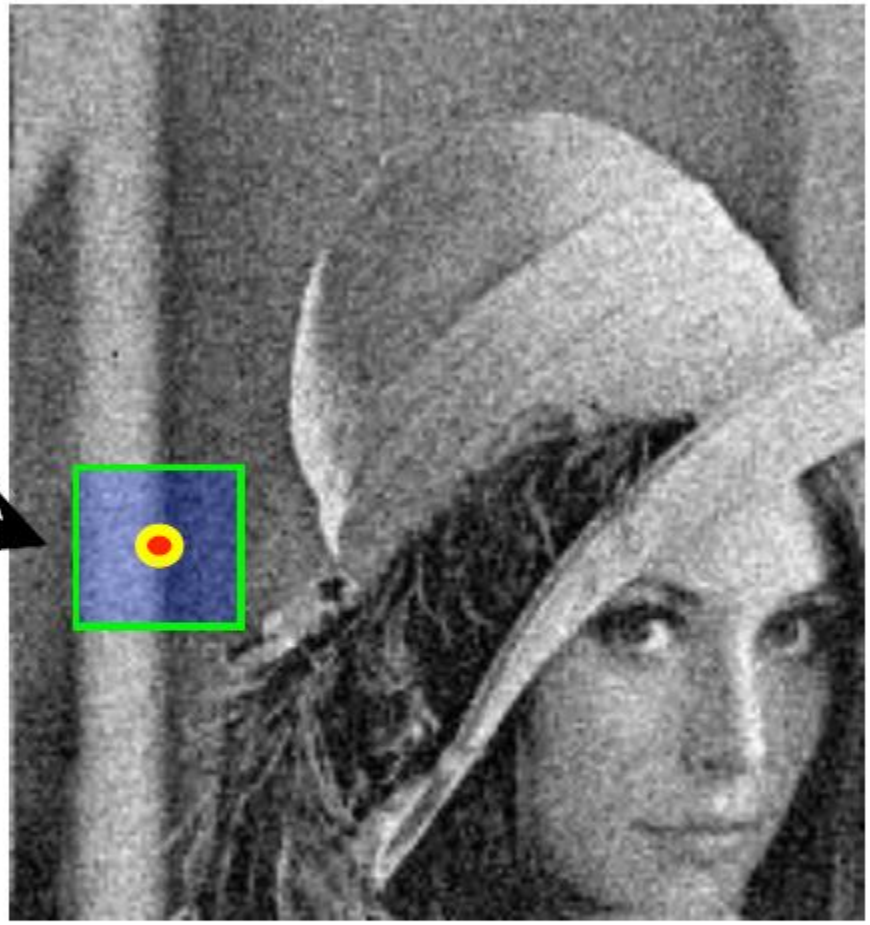


- For each and every pixel **p:**

  – Define a small, simple fixed size neighborhood;

NL-Means Method: Buades (2005)

$$\mathbf{V_p} = \begin{bmatrix} 0.74 \\ 0.32 \\ 0.41 \\ 0.55 \\ \dots \\ \dots \\ \dots \end{bmatrix}$$



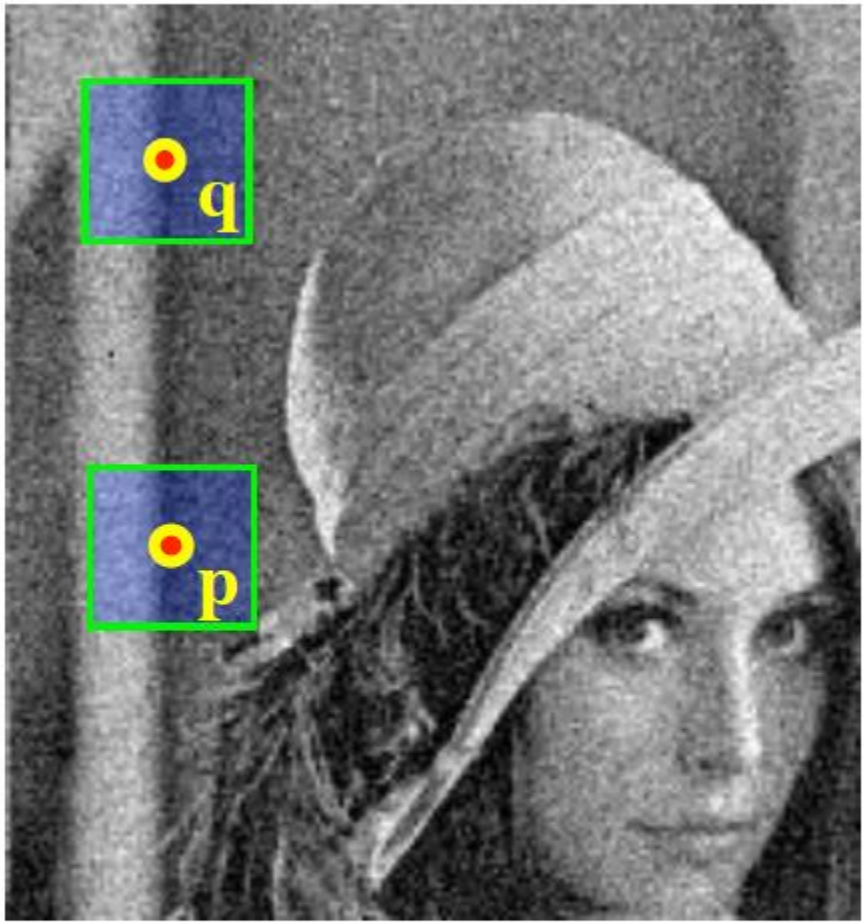
- For each and every pixel **p:**
  - Define a small, simple fixed size neighborhood;
  - Define vector $\mathbf{V_p}$: a list of neighboring pixel values.

NL-Means Method: Buades (2005)

'Similar' pixels **p, q**

→ **SMALL**
vector distance;

$$|| V_p - V_q ||^2$$

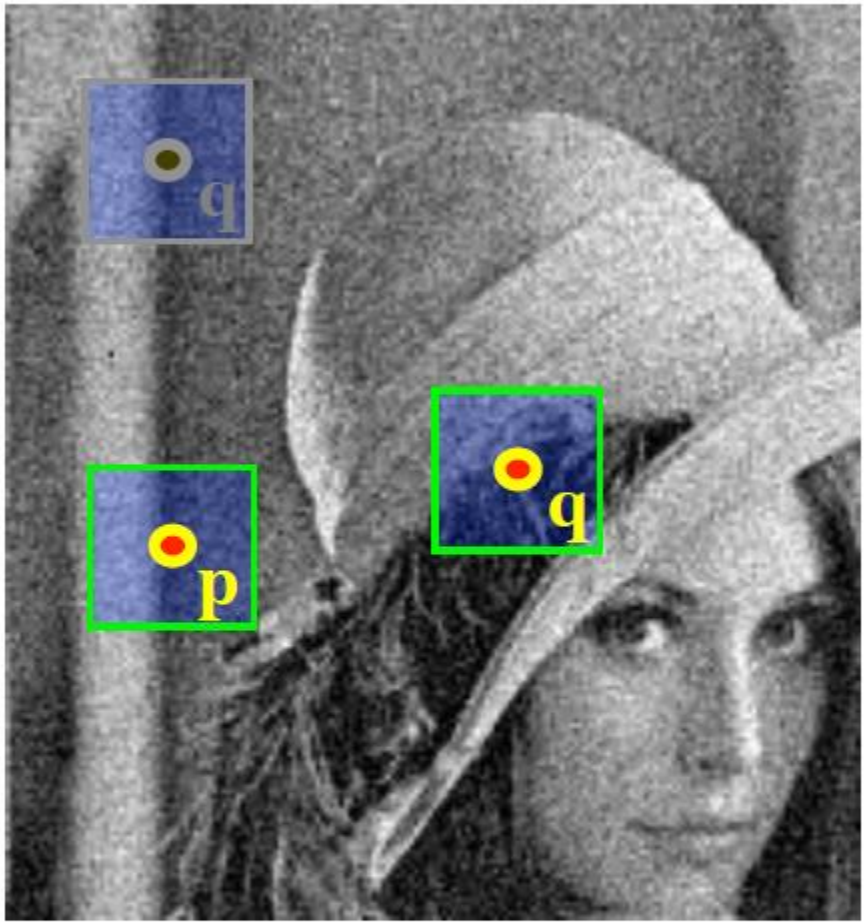


NL-Means Method: Buades (2005)

'Dissimilar' pixels **p, q**

→ **LARGE**
vector distance;

$$|| V_p - V_q ||^2$$



NL-Means Method: Buades (2005)

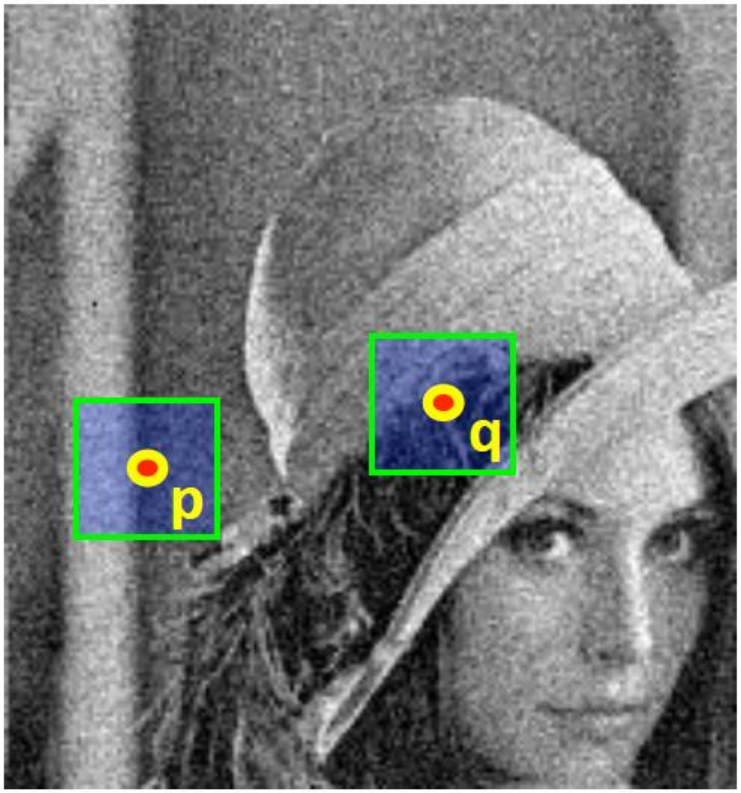
'Dissimilar' pixels **p, q**

→ **LARGE**

vector distance;

$$|| V_p - V_q ||^2$$

Filter with this!



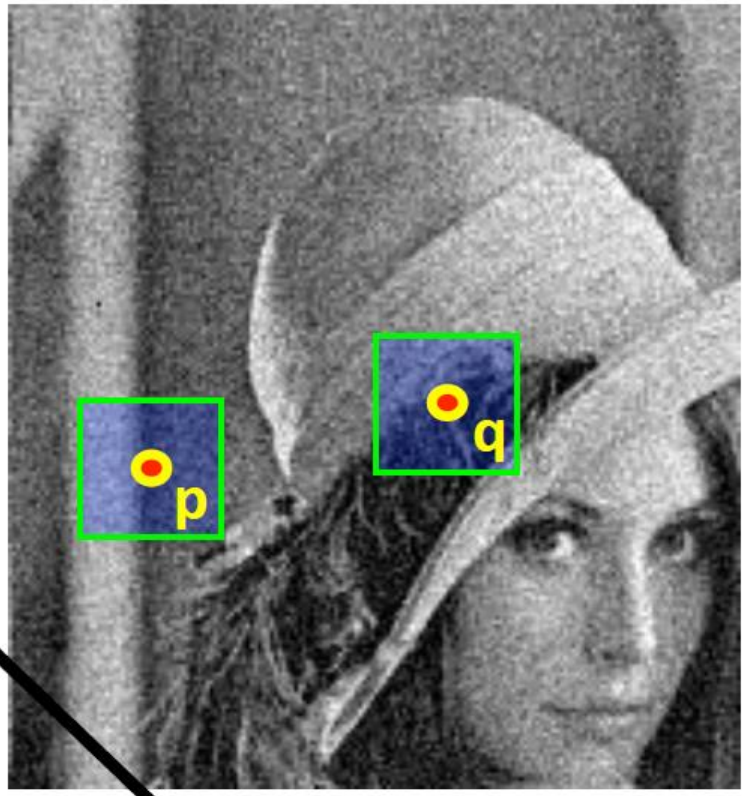
NL-Means Method: Buades (2005)

p, q neighbors define a vector distance;

$$|| V_p - V_q ||^2$$

Filter with this:

No spatial term!



$$NLMF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\| p - q \|) G_{\sigma_r}(\| \vec{V}_p - \vec{V}_q \|^2) I_q$$

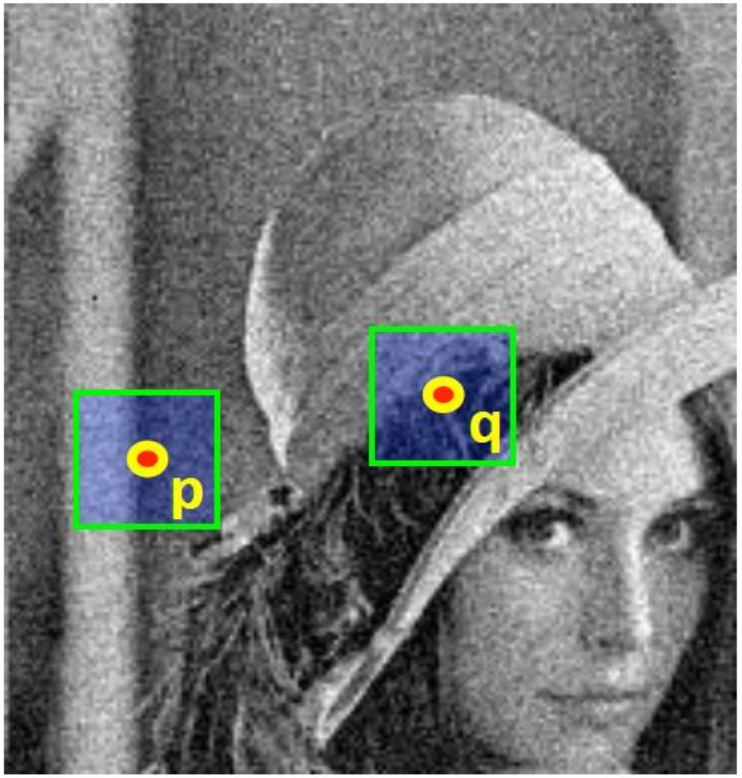
It is a weight based on the distance between the 2 vectors ← more distance == less weight

NL-Means Method: Buades (2005)

pixels **p, q** neighbors
Set a vector distance;

$$|| V_p - V_q ||^2$$

Vector Distance to p sets weight for each pixel q



$$NLMF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_r}\left(|| \vec{V}_p - \vec{V}_q ||^2 \right) I_q$$

To reduce computational complexity a search window can be imposed, for example 31-by-31

NL-Means Filter (Buades 2005)

- Noisy source image:

NL-Means Filter (Buades 2005)

- Gaussian Filter

Low noise,

Low detail

NL-Means Filter (Buades 2005)

- Anisotropic Diffusion

(Note 'stairsteps': ~ piecewise constant)

NL-Means Filter (Buades 2005)

- Bilateral Filter

(better, but similar 'stairsteps' :

NL-Means Filter (Buades 2005)

- NL-Means:

Sharp,
Low noise,
Few artifacts.

Why NLM is Better?

- NLM filter weights precisely captures local edge structure as desired

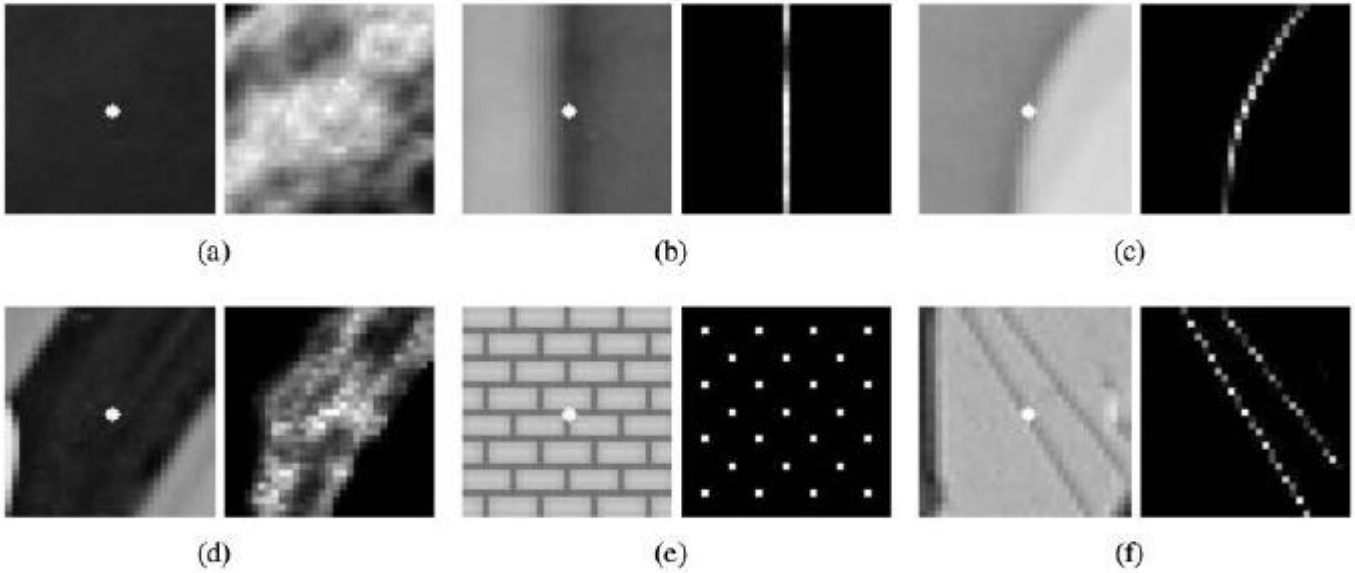


Figure 2. Display of the NL-means weight distribution used to estimate the central pixel of every image. The weights go from 1(white) to zero(black).

Matlab Implementation

- imnlmfilt()
  - [filteredImage,estDoS] = imnlmfilt(noisyImage);

Order Statistics Filters

Spatial filters based on ordering the pixel values that make up the neighbourhood defined by the filter support.

Useful spatial filters include

- Median filter
- Max and min filter
- Midpoint filter
- Alpha trimmed mean filter

Median Filter

Median Filter:

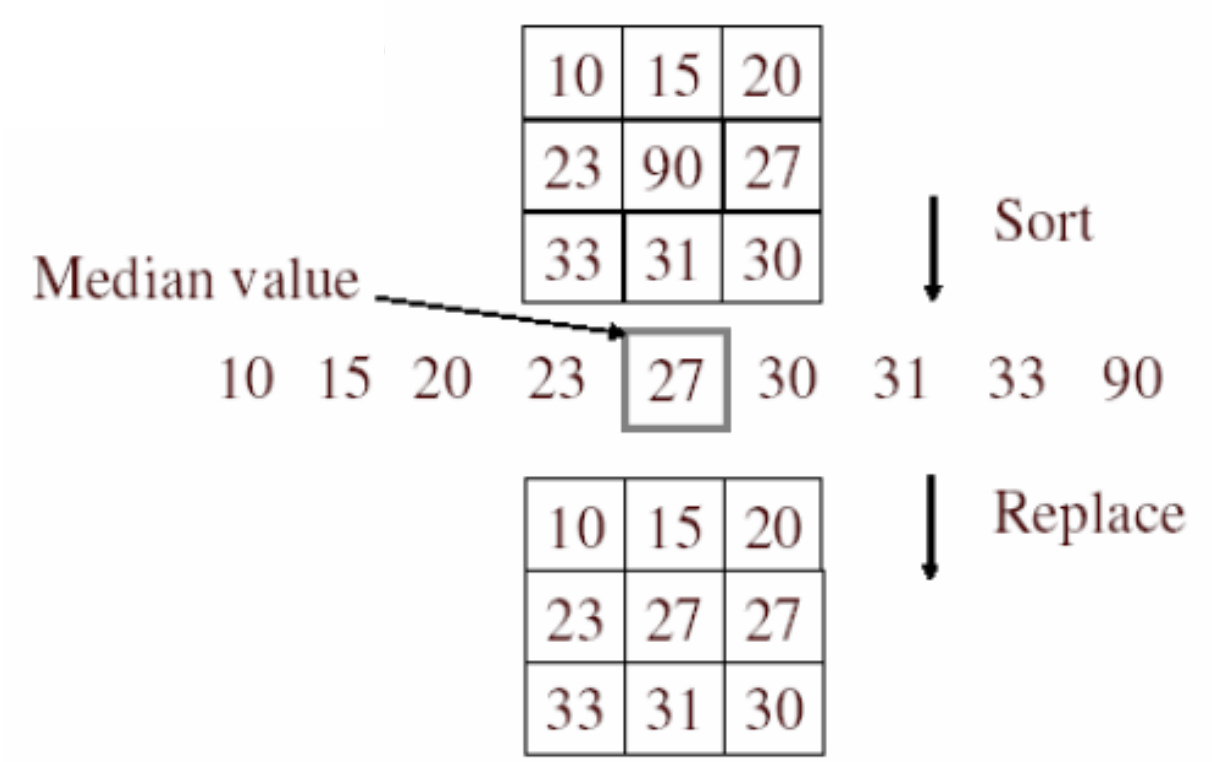
$$\hat{f}(x,y) = \underset{(s,t)\in S_{xy}}{median}\{g(s,t)\}$$

Excellent at noise removal, without the smoothing effects that can occur with other smoothing filters.

Particularly good when salt and pepper noise is present.

Median filtering

- A **median filter** operates over a window by selecting the median intensity in the window



- Is median filtering linear?

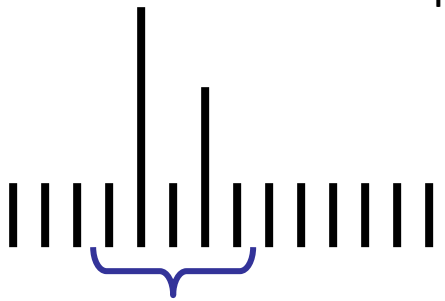
Source: K. Grauman

Median filter

Replace each pixel by the median over N pixels (5 pixels, for these examples).

Median([1 7 1 5 1]) = 1
Mean([1 7 1 5 1]) = 2.8

In:

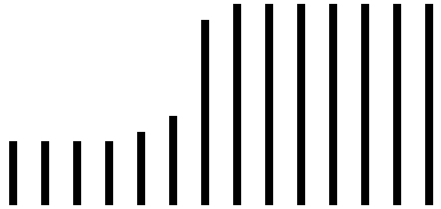


5-pixel neighborhood

Out:

Spike noise is removed

In:



Out:

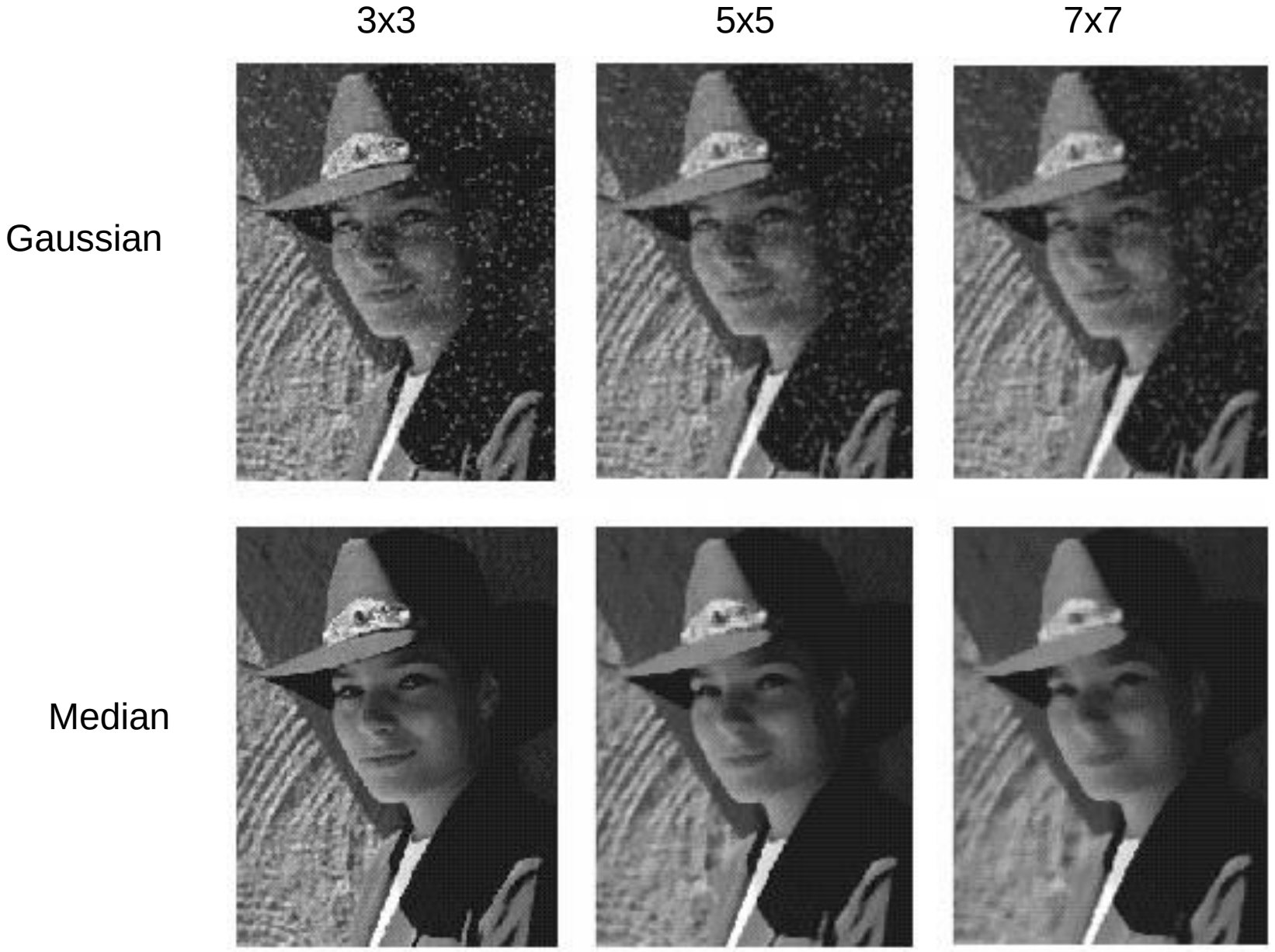
Monotonic edges remain unchanged

Median filtering results

Best for salt and pepper noise

http://homepages.inf.ed.ac.uk/rbf/HIPR2/mean.htm#guidelines

Median vs. Gaussian filtering



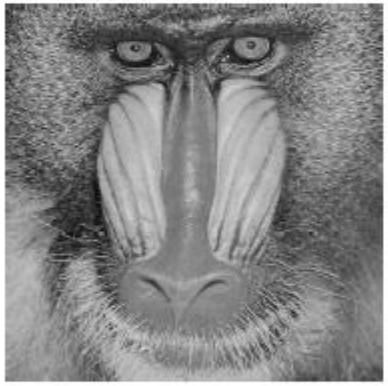
Median filter using medfilt2

```
%The median filter  using the medfilt2 function
ym1 = medfilt2(xg, [4 4]);
ym2 = medfilt2(xg, [10 10]);

subplot(2,2,1), imshow(xg);
subplot(2,2,2), imshow(yg);
subplot(2,2,3), imshow(ym1);
subplot(2,2,4), imshow(ym2);
```

Non Linear Order Statistics Filters

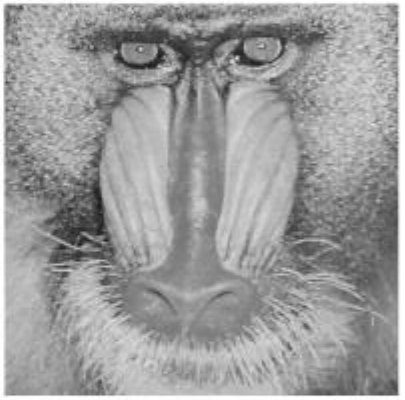
- Order stats filters

$$g(x,y) = \underset{(s,t)\in W_{(x,y)}}{median}\{f(s,t)\}$$

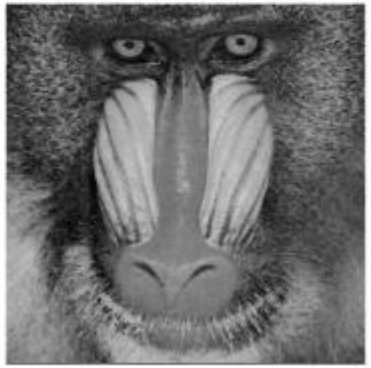


original

$$g(x,y) = \max_{(s,t)\in W_{(x,y)}}\{f(s,t)\}$$



$$g(x,y) = \min_{(s,t)\in W_{(x,y)}}\{f(s,t)\}$$



Median Filter

- Problems with Median Filter

- good at remove outliers like pepper & salt noise
- certain structure is kept (e.g, straight edge)
- details lost, and structure determined by the kernel size

Max and Min Filter

Max Filter:

$$\hat{f}(x,y) = \max_{(s,t)\in S_{xy}} \{g(s,t)\}$$

Min Filter:

$$\hat{f}(x,y) = \min_{(s,t)\in S_{xy}} \{g(s,t)\}$$

Max filter is good for pepper noise and Min filter is good for salt noise.

Midpoint Filter

Midpoint Filter:

$$\hat{f}(x,y) = \frac{1}{2}\left[\max_{(s,t)\in S_{xy}}\{g(s,t)\} + \min_{(s,t)\in S_{xy}}\{g(s,t)\}\right]$$

Good for random Gaussian and uniform noise.

Alpha-Trimmed Mean Filter

Alpha-Trimmed Mean Filter:

$$\hat{f}(x,y) = \frac{1}{mn-d} \sum_{(s,t)\in S_{xy}} g_r(s,t)$$

We can delete the $d/2$ lowest and $d/2$ highest grey levels.

So $g_r(s, t)$ represents the remaining $mn - d$ pixels.

Adaptive Median Filter

The behaviour of **adaptive filters** changes depending on the characteristics of the image inside the filter region.

(e.g. the bilateral filter and the NLM filter);

Now, we will take a look at the **adaptive median filter.**

Adaptive Median Filtering

The median filter performs relatively well on impulse noise as long as the spatial density of the impulse noise is not large.

The adaptive median filter can handle much more spatially dense impulse noise, and also performs some smoothing for non-impulse noise.

Adaptive Median Filtering (cont…)

The key to understanding the algorithm is to remember that the adaptive median filter has three purposes:

- Remove impulse noise
- Provide smoothing of other noise
- Reduce distortion (excessive thinning or thickenning of object boundaries).

Adaptive Median Filtering (cont…)

In the adaptive median filter, the filter size changes depending on the characteristics of the image.

Notation:

- S_{xy} = the support of the filter centerd at (x, y)
- z_{min} = minimum grey level in S_{xy}
- z_{max} = maximum grey level in S_{xy}
- z_{med} = median of grey levels in S_{xy}
- z_{xy} = grey level at coordinates (x, y)
- S_{max} =maximum allowed size of S_{xy}

Adaptive Median Filtering (cont…)

Stage A:

$A1 = z_{med} - z_{min}$

$A2 = z_{med} - z_{max}$

If $A1 > 0$ and $A2 < 0$, Go to stage B

Else increase the window size

If window size $\leq S_{max}$ repeat stage A

Else output z_{med}

Stage B:

$B1 = z_{xy} - z_{min}$

$B2 = z_{xy} - z_{max}$

If $B1 > 0$ and $B2 < 0$, output z_{xy}

Else output z_{med}

Adaptive Median Filtering (cont…)

Stage A: $A1 = z_{med} - z_{min}$

$A2 = z_{med} - z_{max}$

If $A1 > 0$ and $A2 < 0$, Go to stage B

Else increase the window size

If window size $\leq S_{max}$ repeat stage A

Else output z_{med}

Stage A determines if the output of the median filter z_{med} is an impulse or not (black or white).

If it is not an impulse, we go to stage B.

If it is an impulse the window size is increased until it reaches S_{max} or z_{med} is not an impulse.

Note that there is no guarantee that z_{med} will not be an impulse. The smaller the the density of the noise is, and, the larger the support S_{max}, we expect not to have an impulse.

Adaptive Median Filtering (cont…)

Stage B: $B1 = z_{xy} - z_{min}$

$B2 = z_{xy} - z_{max}$

If $B1 > 0$ and $B2 < 0$, output z_{xy}

Else output z_{med}

Stage B determines if the pixel value at (x, y), that is z_{xy}, is an impulse or not (black or white).

If it is not an impulse, the algorithm outputs the unchanged pixel value z_{xy}.

If it is an impulse the algorithm outputs the median z_{med}

.

Adaptive Filtering Example

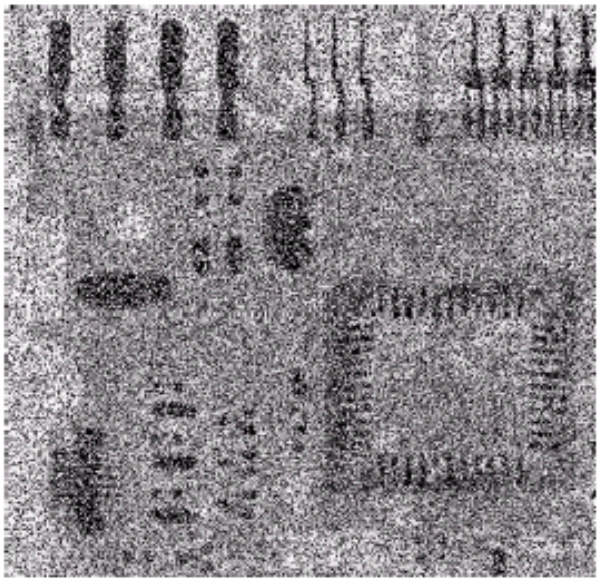
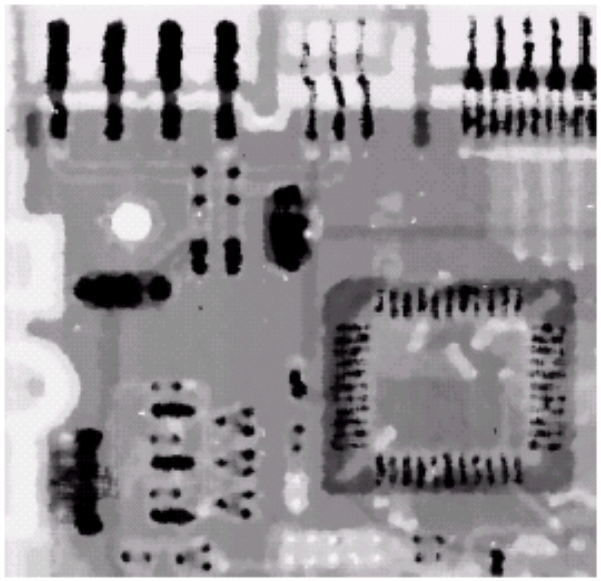
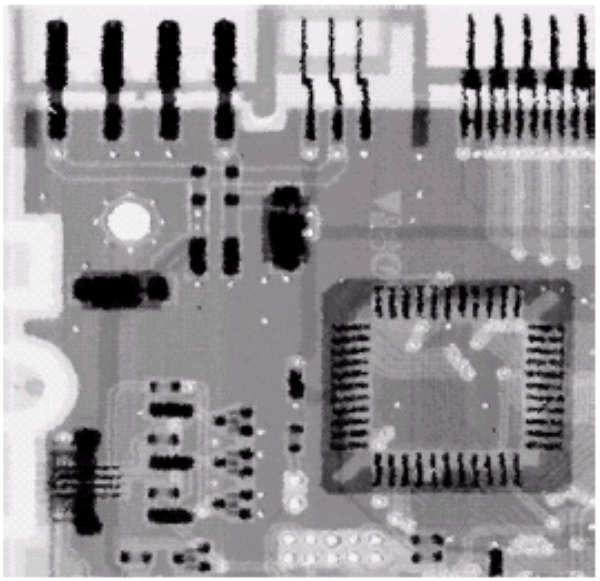


Image corrupted by salt and pepper noise with probabilities $P_a = P_b = 0.25$



Result of filtering with a 7x7 median filter



Result of adaptive median filtering with $S_{max} = 7$

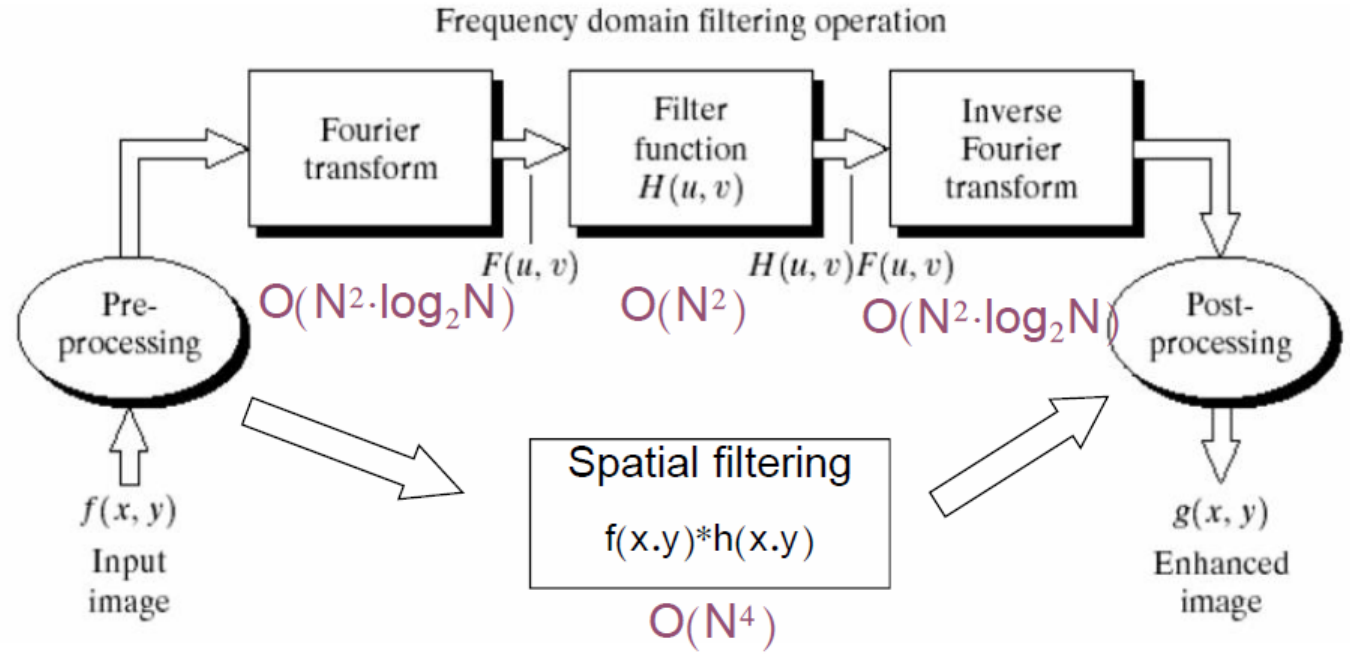
AMF preserves sharpness and details, e.g. the connector fingers.

Frequency Domain Filtering

- Image is first transformed to frequency domain using DFT (using the FFT algorithm): F(u,v)
- The filter transfer function is defined in the frequency domain (LPF, HPF, BPF, notch filter, band reject filter, etc.): H(u,v)
- We get $S(u,v) = F(u,v) * H(u,v)$ (element-wise product)
- We then use IDFT to bring image back to spatial domain: $s(x,y) = IFFT(S(u,v))$

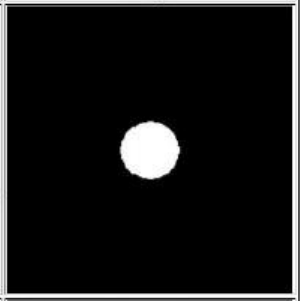
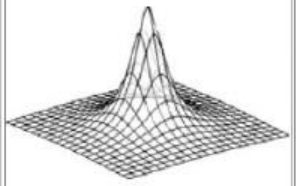
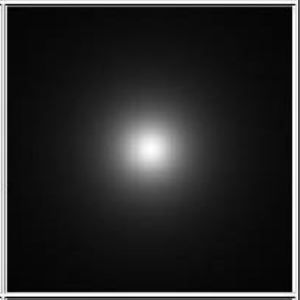
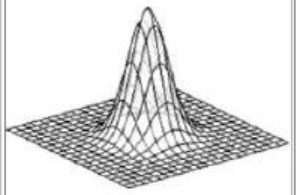
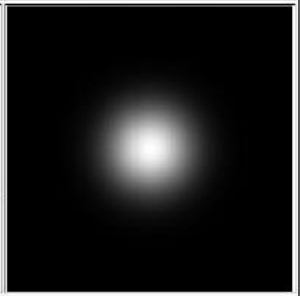
Filtering in Frequency Domain

- Why filtering in Freq Domain ?
  - faster convolution, if involves large kernels



Freq Domain Low Pass Filters

- Cutting off high freq components at D_0:

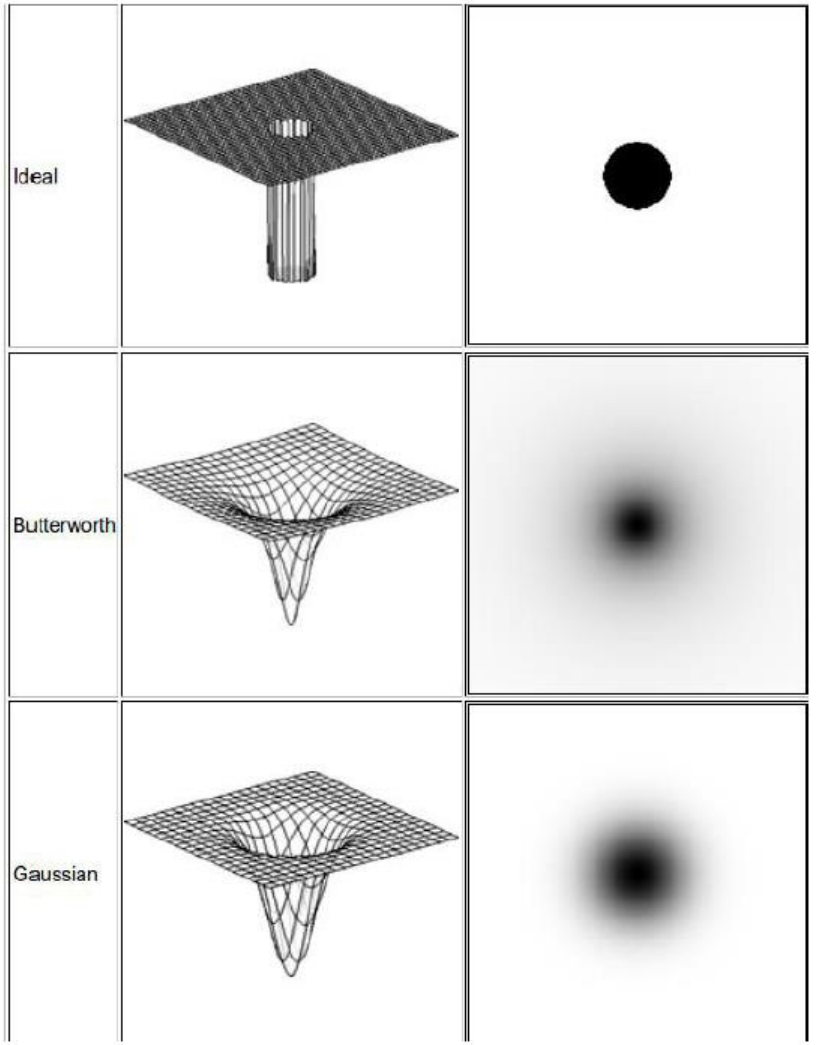
Lowpass Filter	Mesh	Image
Ideal: $H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \le D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases}$		
Butterworth: $H(u,v) = \frac{1}{1+\left[D(u,v)/D_0\right]^{2n}}$		
Gaussian: $H(u,v) = e^{-D^2(u,v)/2D_0^2}$		

High Pass Filtering

- Matlab Implementation

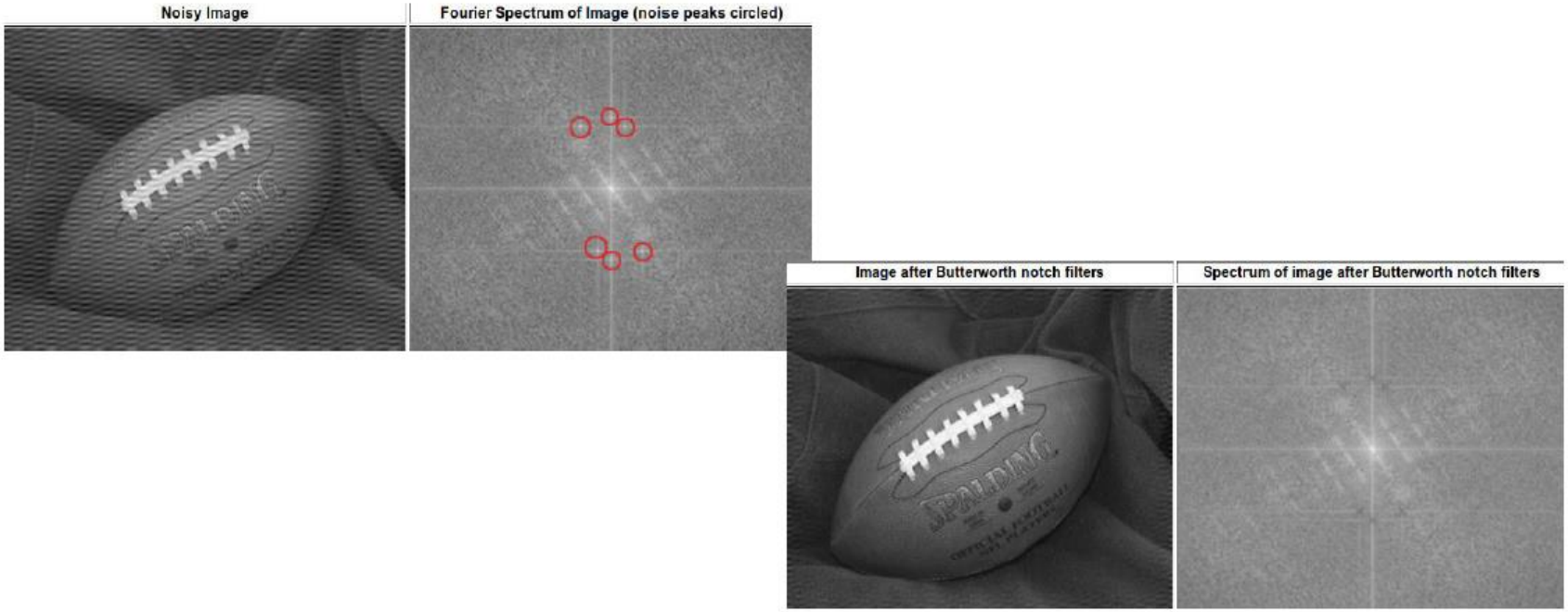
$$H_{hp}(u,v) = 1 - H_{lp}(u,v)$$

```
function H = hpfilter(type, M, N, D0, n)

if nargin == 4
   n = 1; % Default value of n.
end

% Generate highpass filter.
Hlp = lpfilter(type, M, N, D0, n);
H = 1 - Hlp;
```



Bandpass/Notch Filter

- Removing repeatitive patterns
  - Used to remove repetitive "Spectral" noise from an image
  - Act like a narrow highpass filter, but they "notch" out frequencies other than the dc component
  - Attenuate a selected frequency (and some of its neighbors) and leave other frequencies of the Fourier transform relatively unchanged

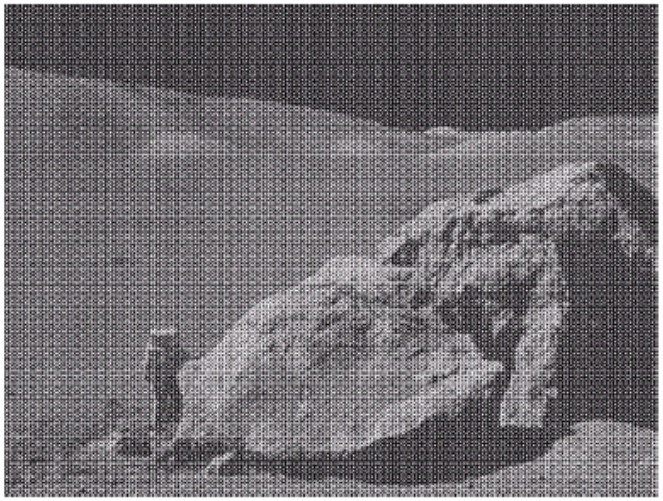


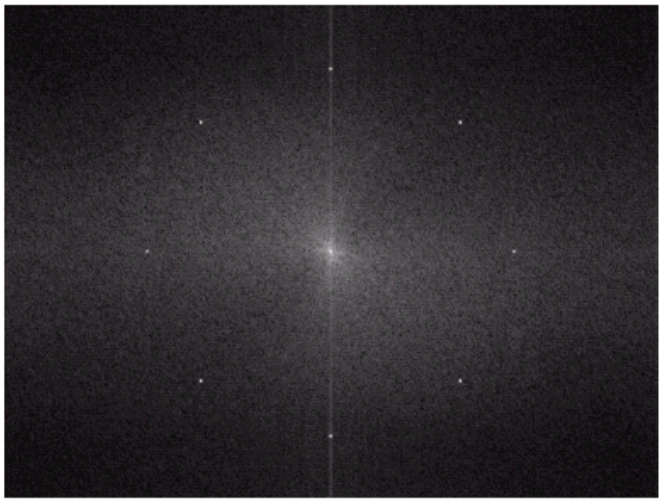
Periodic Noise

Typically arises due to electrical or electromagnetic interference.

Gives rise to regular noise patterns in an image.

Frequency domain techniques in the Fourier domain are most effective at removing periodic noise.





Band Reject Filters

Removing periodic noise form an image involves removing a particular range of frequencies from that image.

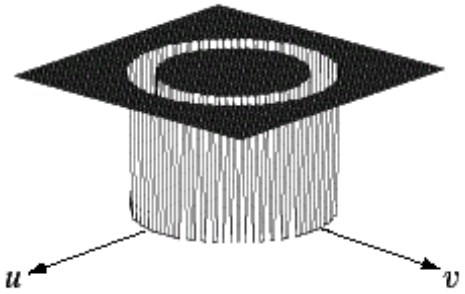
Band reject filters can be used for this purpose

An ideal band reject filter is given as follows:

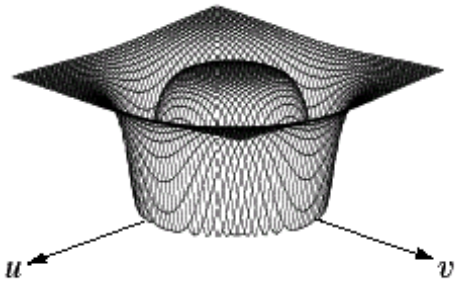
$$H(u,v) = \begin{cases} 1 & if\ D(u,v) < D_0 - \frac{W}{2} \\ 0 & if\ D_0 - \frac{W}{2} \le D(u,v) \le D_0 + \frac{W}{2} \\ 1 & if\ D(u,v) > D_0 + \frac{W}{2} \end{cases}$$

Band Reject Filters (cont…)

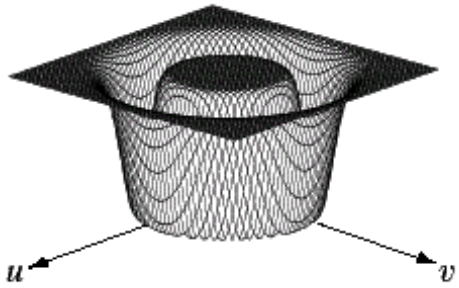
The ideal band reject filter is shown below, along with Butterworth and Gaussian versions of the filter



Ideal Band Reject Filter



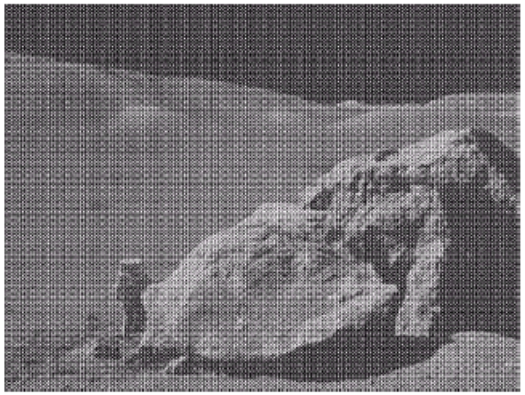
Butterworth Band Reject Filter (of order 1)



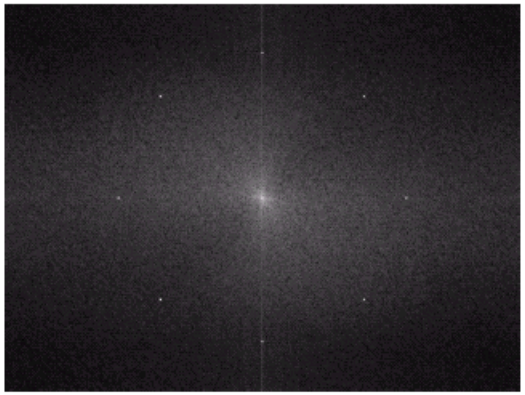
Gaussian Band Reject Filter

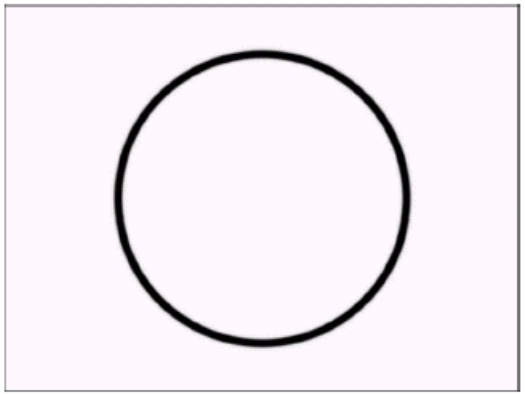
Band Reject Filter Example

Image corrupted by sinusoidal noise



Fourier spectrum of corrupted image

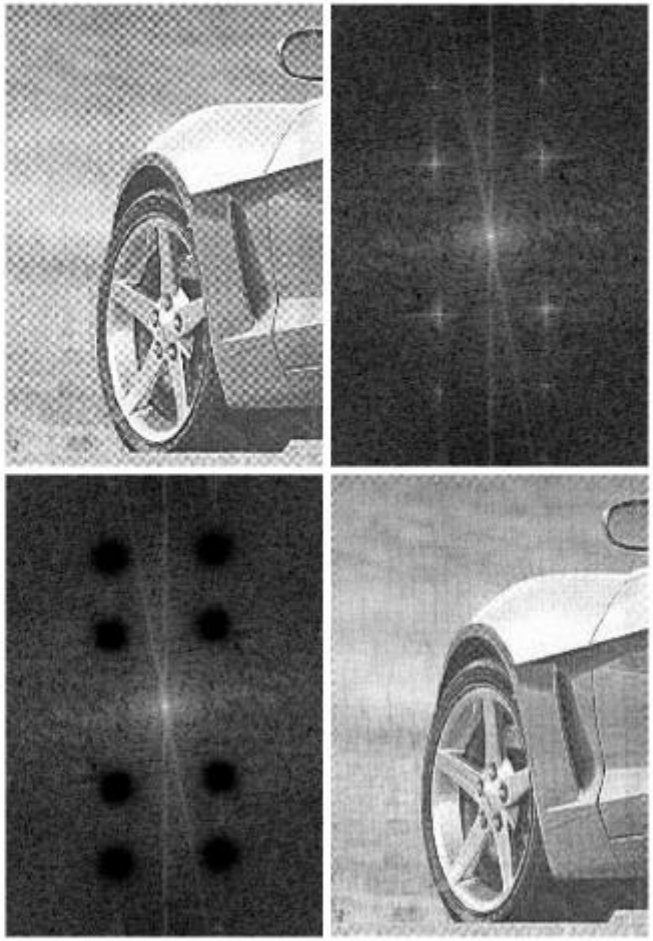




Butterworth band reject filter

Filtered image

notch filter



a b
c d

FIGURE 4.64
(a) Sampled newspaper image showing a moiré pattern. (b) Spectrum. (c) Butterworth notch reject filter multiplied by the Fourier transform. (d) Filtered image.

Image Restoration

Degradations

• original

• optical blur

• motion blur

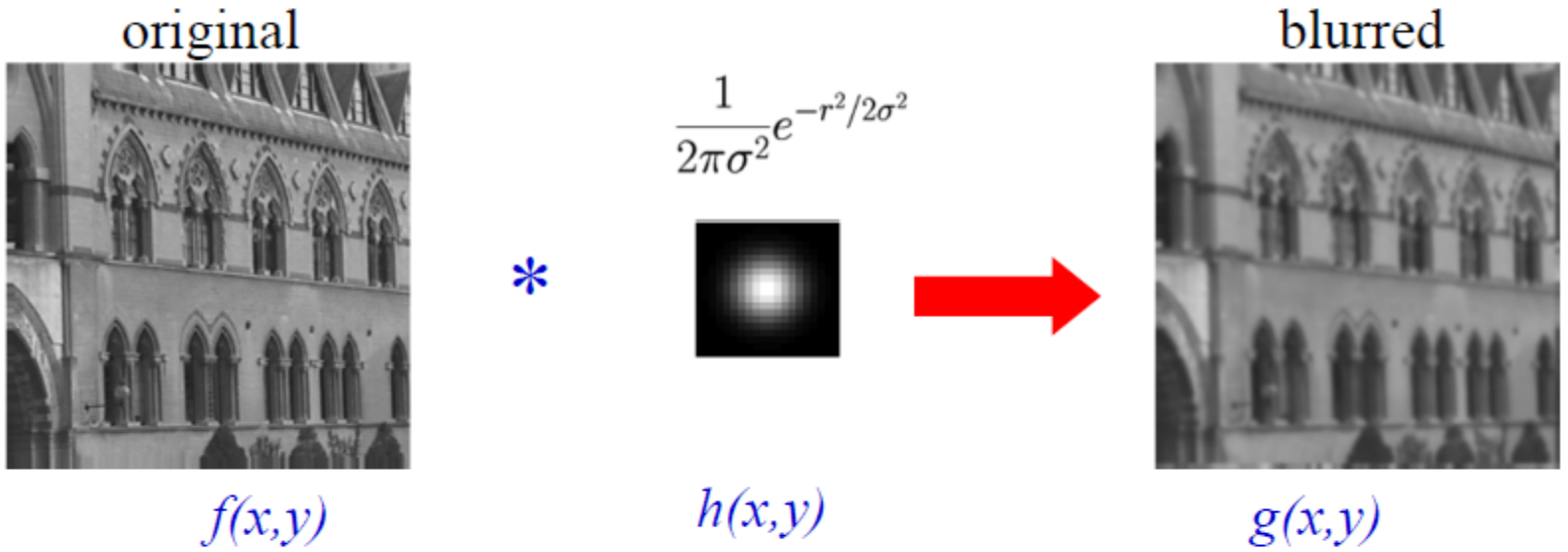
• spatial quantization (discrete pixels)

• additive intensity noise

Degradation model

Model degradation as a convolution with a linear, shift invariant, filter $h(x,y)$

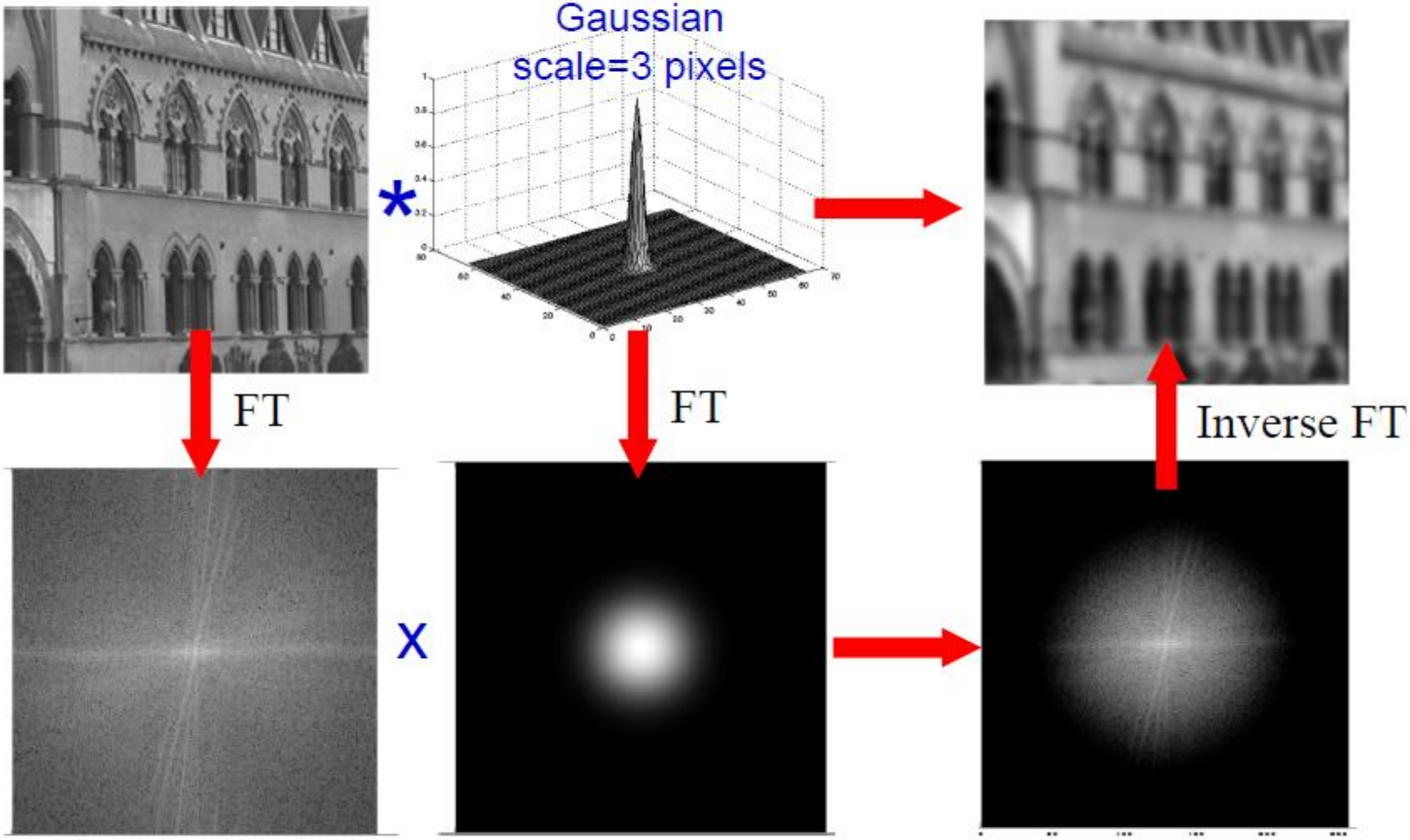
- Example: for out of focus blurring, model $h(x,y)$ as a Gaussian



i.e. : $g(x,y) = h(x,y) * f(x,y)$

$h(x,y)$ is the impulse response or point spread function of the imaging system

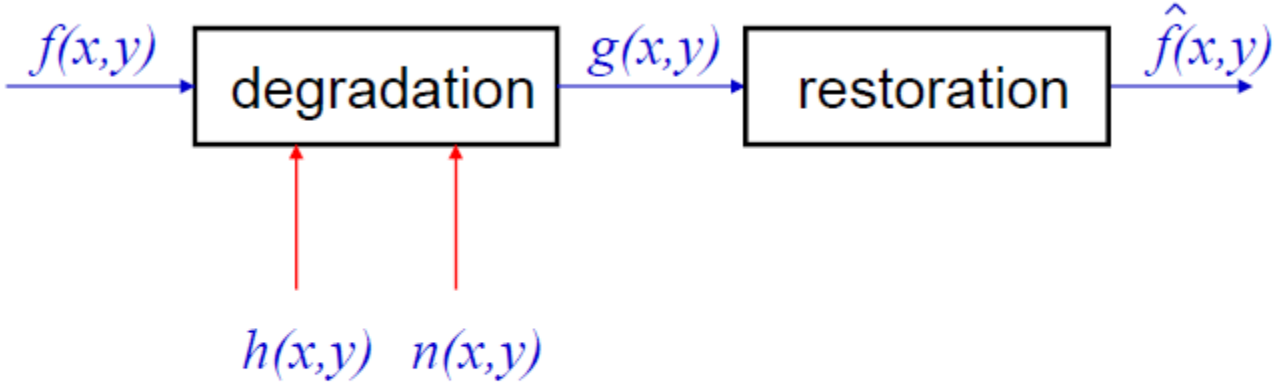
The challenge: loss of information and noise



Blurring acts as a low pass filter and attenuates higher spatial frequencies

Definitions

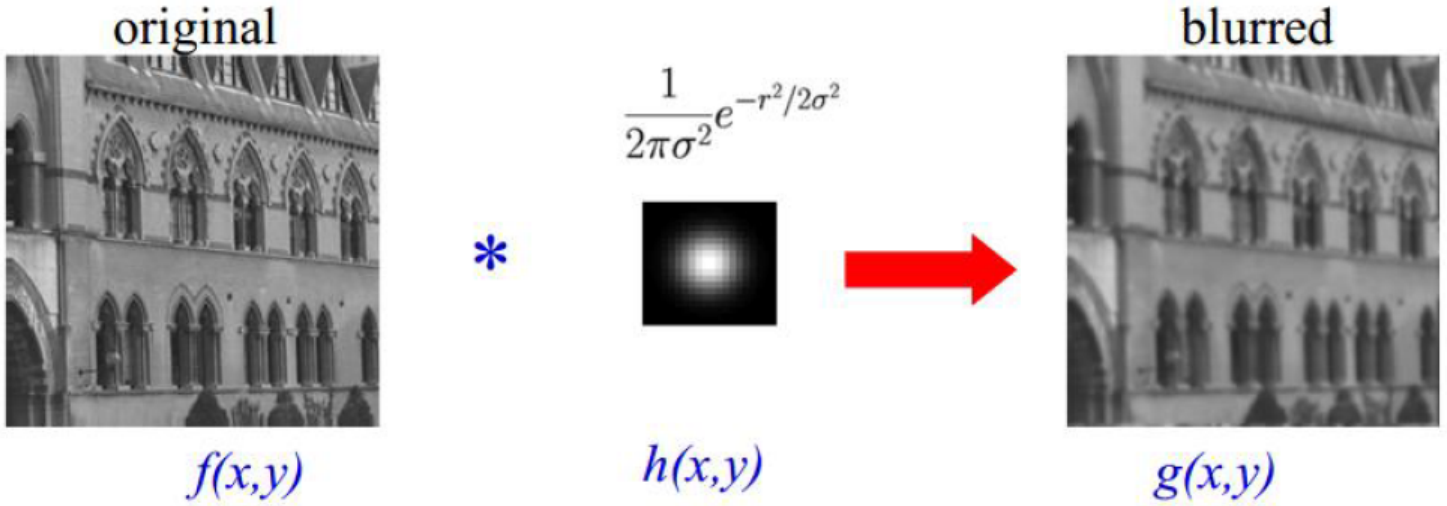
- $f(x,y)$ – image before degradation, 'true image'
- $g(x,y)$ – image after degradation, 'observed image'
- $h(x,y)$ – degradation filter
- $\hat{f}(x,y)$ – estimate of $f(x,y)$ computed from $g(x,y)$
- $n(x,y)$ – additive noise



$$g(x,y) = h(x,y) * f(x,y) + n(x,y) \quad \leftrightarrow \quad G(u,v) = H(u,v)\,F(u,v) + N(u,v)$$

Degradation Model

- Degradation as a linear time invariant process: g = f*h
  - e.g. out of focus blur



i.e. : $g(x,y) = h(x,y) * f(x,y)$

h(x,y) is the impulse response or *point spread function* (PSF) of the imaging system

Degradation Model

- Image enhancement: "improve" an image subjectively.
- Image restoration: remove distortion from image, to go back to the "original" -- objective process

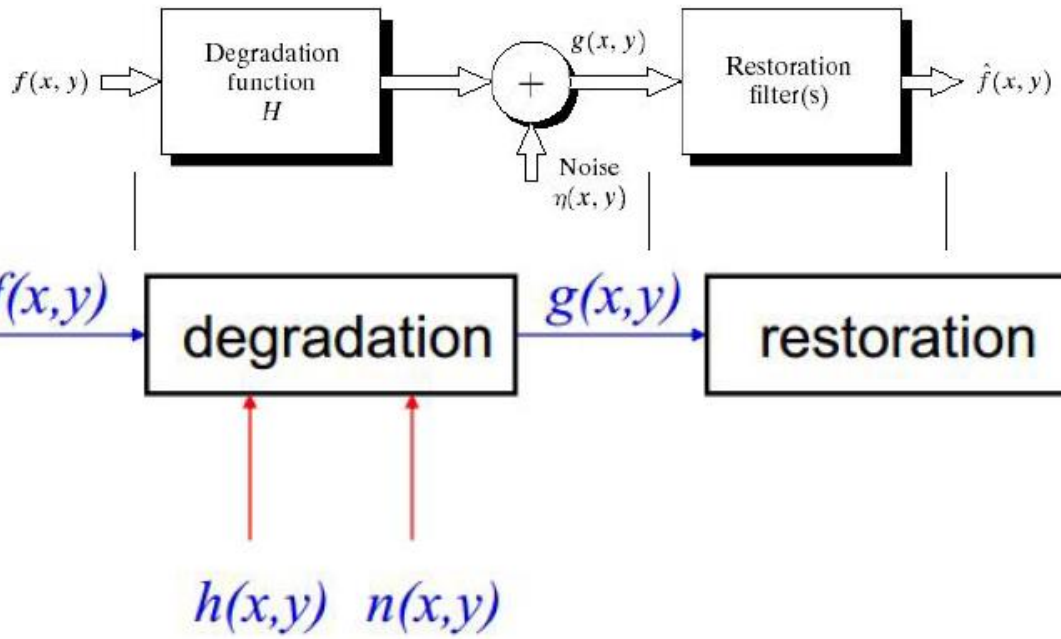


FIGURE 5.1 A model of the image degradation/restoration process.

$$g(x,y) = h(x,y) * f(x,y) + n(x,y) \leftrightarrow G(u,v) = H(u,v)\,F(u,v) + N(u,v)$$

a model for image distortion

- Image restoration
  - Use a priori knowledge of the degradation
  - Modeling the degradation and apply the inverse process
  - Formulate and evaluate objective criteria of goodness

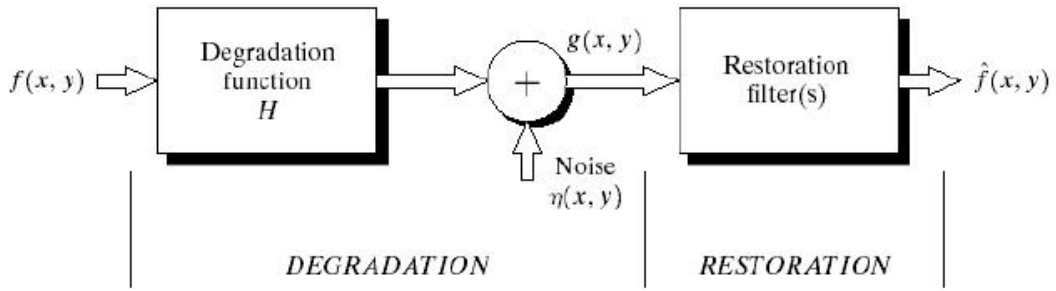


FIGURE 5.1 A model of the image degradation/ restoration process.

$$g(x,y) = H[f(x,y)] + \eta(x,y)$$

$\rightarrow$ design restoration filters such that $\hat{f}(x,y)$ is as close to $f(x,y)$ as possible.

The inverse filter

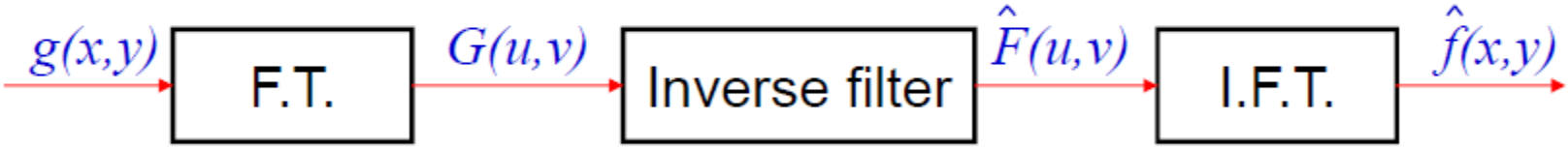
Start from the generative model

$$g(x,y) = h(x,y) * f(x,y) + n(x,y) \quad \Leftrightarrow \quad G(u,v) = H(u,v)\,F(u,v) + N(u,v)$$

and for the moment ignore $n(x,y)$, then an estimate of $f(x,y)$ is obtained from

$$\hat{F}(u,v) = G(u,v) \,/\, H(u,v)$$

Restoration with an inverse filter



Example : Deblurring (deconvolution)

Image blurred with Gaussian point spread function

$h(x,y) = \frac{1}{2\pi\sigma^2} e^{-r^2/2\sigma^2}$ $n(x,y) =$ Normal distribution, mean zero

blur $\sigma = 1.0$ pixels

noise $\sigma = 0.3$ grey levels

$f(x,y)$ $g(x,y)$

Restoration with an inverse filter

$\hat{F}(u,v) = G(u,v) / H(u,v)$ where $H(u,v)$, is the FT of the Gaussian

Deblurring with an inverse filter

noise σ = 0.3 grey levels

$\hat{F}(u,v) = G(u,v) / H(u,v)$

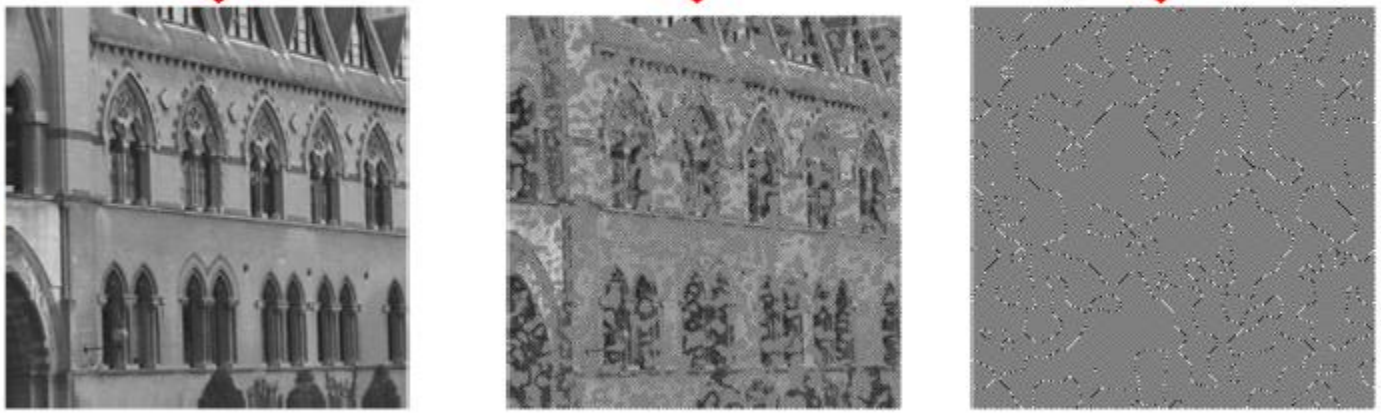
blur σ = 0.5 pixels

blur σ = 1.0pixels

blur σ = 1.5pixels

$g(x,y)$

$\hat{f}(x,y)$



Inverse filtering

- With the estimated degradation function H(u,v)

$$G(u,v)=F(u,v)H(u,v)+N(u,v)$$

Unknown noise

$$=\hat{F}(u,v)=\frac{G(u,v)}{H(u,v)}=F(u,v)+\frac{N(u,v)}{H(u,v)}$$

Estimate of original image

Problem: 0 or small values

Sol: limit the frequency around the origin