

# Module 6

## Image Denoising and Nonlinear Filtering

- **Noise Models**
- **Morphological and Order Statistic Filters**
- **Bilateral Filter**
- **Anisotropic Diffusion**
- **Non-local (NL) Means**
- **Wavelet Soft Thresholding**
- **BM3D**
- **Homomorphic Filtering**

# IMAGE NOISE MODELS

- To proceed further in understanding and **designing** non-linear filters, we will need to deepen our statistical modeling of **noise**.
- Any **channel** (electrical wire, airwaves, optical fiber) is **imperfect**. Images sent over a channel always suffer a **degradation of information**:
  - (a) **static** (high-frequency noise or thermal noise)
  - (b) **bit errors**
  - (c) **sensor noise**
- These errors degrade
  - visual interpretation
  - computer image analysis

# White Noise is Uncorrelated Noise

- Assume the **additive white noise** model:

$$\mathbf{J} = \mathbf{I} + \mathbf{N}$$

where  $\mathbf{I}$  is an original image and  $\mathbf{N}$  a **digital white noise image**.

- **White noise** suggests an (**on average**) a flat spectrum.
- An **equivalent meaning** is that the elements  $N(m, n)$  of  $\mathbf{N}$  are **uncorrelated**: any two elements  $N(m, n)$ ,  $N(p, q)$  of the noise matrix  $\mathbf{N}$  (where  $(m, n) \neq (p, q)$ ) affect one another **little or not at all**.
- Mathematically this is equivalent to the noise being **white**.

# Range of Noise

- The set of values  $\mathfrak{R}_N$  the noise can take. In digital images the noise must be **discrete integers**. However, we will use **approximate continuous models**.
- For **additive noise** the noise range is usually **symmetric**:  $\mathfrak{R}_N = [-Q, Q]$  where possibly  $Q = \infty$ .
- For **multiplicative noise** assume the noise range is **non-negative**:  $\mathfrak{R}_N = [0, Q]$  where possibly  $Q = \infty$ .

# Probability Density Function of Noise

- The **probability density function** or **PDF**  $f_{\mathbf{N}}(q)$  is a **model** of the **probabilities** of  $\mathbf{N}(m, n)$ :

$$\Pr\{a \leq \mathbf{N}(m, n) \leq b\} = \int_a^b f_{\mathbf{N}}(q) dq$$

where  $[a, b] \subseteq \mathfrak{R}_{\mathbf{N}}$ .

- Assume that all elements  $\mathbf{N}(m, n)$  of  $\mathbf{N}$  obey the **identical probability model** - the same PDF.
- So,  $f_{\mathbf{N}}(q)$  is not expressed as a function of  $(m, n)$ .

# Properties of PDF

- **Two** basic properties of the PDF

$$(1) f_{\mathcal{R}_N}(q) \geq 0 \text{ for all } q \in \mathcal{R}_N$$

$$(2) \int_{\mathcal{R}_N} f_{\mathcal{R}_N}(q) dq = 1$$

- Note: (1) and (2) imply that  $f_{\mathcal{R}_N}(q) \leq 1$  for all  $q \in \mathcal{R}_N$ .
- For **symmetric noise**:  $\mathcal{R}_N = [-Q, Q]$  assume that  $f_{\mathcal{R}_N}(q)$  is a **symmetric function**

$$f_{\mathcal{R}_N}(q) = f_{\mathcal{R}_N}(-q)$$

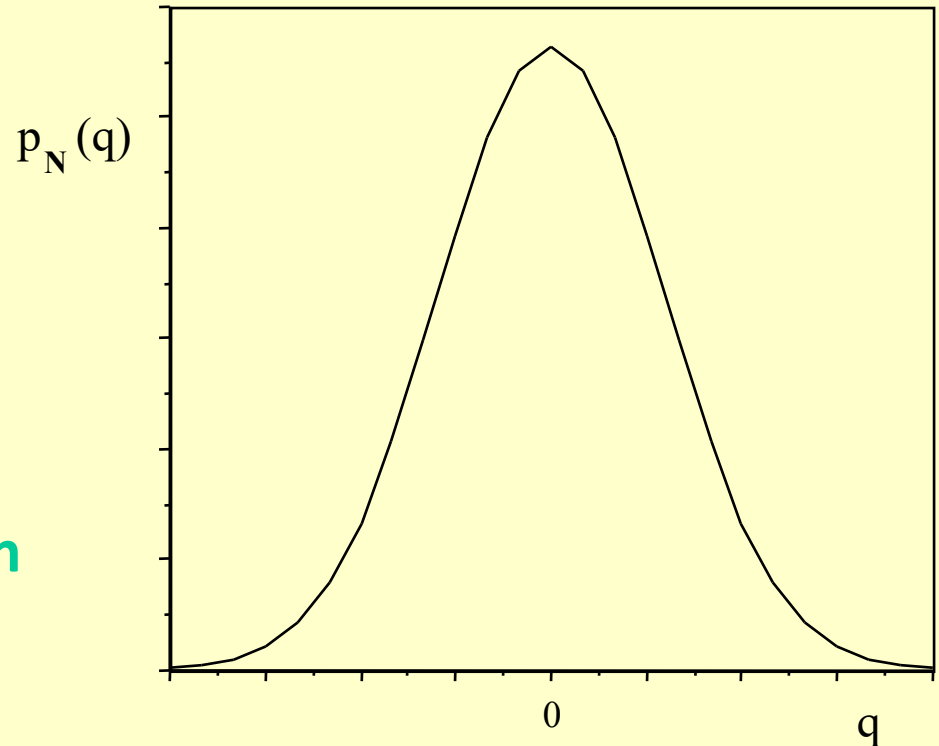
# Gaussian PDF

- Here  $\mathcal{R}_N = [-\infty, \infty]$  and

$$f_N(q) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(q/\sigma)^2}$$

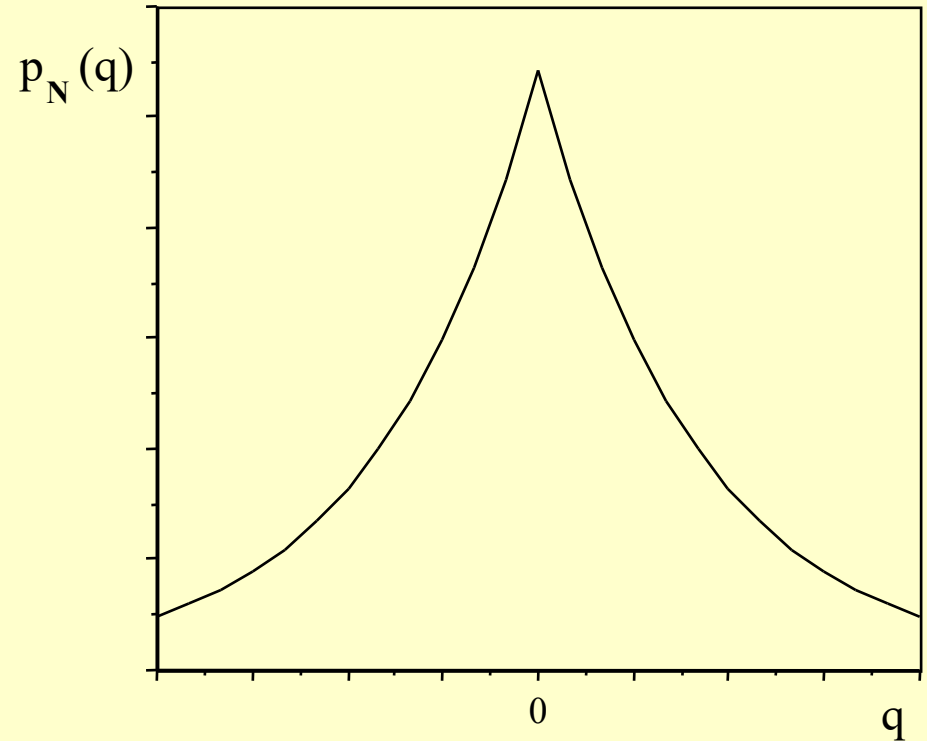
with **scale parameter**  $\sigma$ .

- The parameter  $\sigma^2$  is the **variance** of the PDF
- **The most common** noise that occurs in applications ... and **in nature**.
- The usual noise found in electrical circuits, most communication channels, etc.

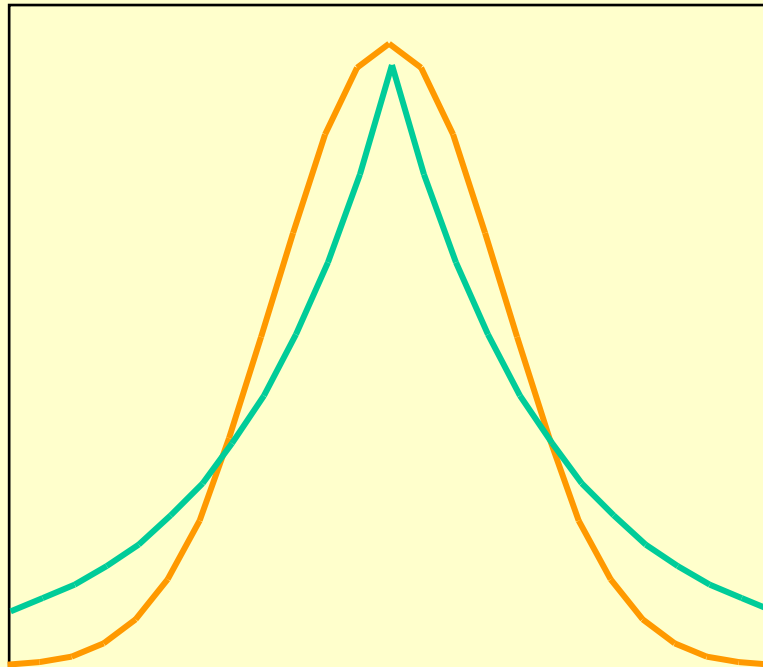


# Laplacian PDF

- Here  $\mathfrak{R}_N = [-\infty, \infty]$  and
$$f_N(q) = \frac{1}{\sqrt{2}\sigma} e^{-\sqrt{2}|q|/\sigma}$$
with **scale parameter**  $\sigma$ .
- Parameter  $\sigma^2$  is **again** the variance.
- Often models **impulse noise**.
- **Large number of small-amplitude values**, and a (relatively) large number of **high-amplitude values**.



# Gaussian vs. Laplacian PDFs



Gaussian  
Laplacian

- Laplacian noise is **impulsive** – a **heavy-tailed** PDF.

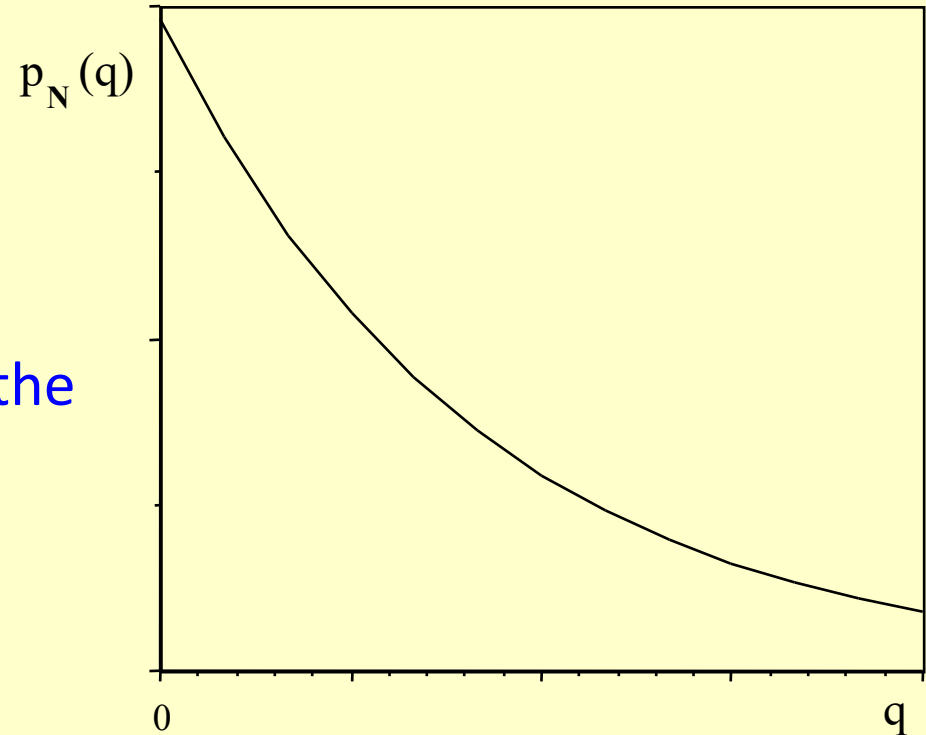
# Exponential PDF

- Here  $\mathfrak{R}_N = [0, \infty]$  and

$$f_N(q) = \frac{1}{\sigma} e^{-q/\sigma}$$

with **scale parameter**  $\sigma$ .

- Parameter  $\sigma^2$  is again equal to the **variance** and  $\sigma$  is the mean.
- Often models **multiplicative noise**.
- A coherent image noise model (radar, laser, electron microscopy, etc)



# Salt-and-Pepper (S&P) Noise

- A model for **significant bit errors**.
- When transmitting digital image data, significant bit errors can occur:
  - '1' → '0'                      ('light' → 'dark')
  - '0' → '1'                      ('dark' → 'light')
- May be due to thermal effects or even gamma rays.
- Salt and pepper noise is **not additive**.

# S&P Noise Model

- Let  $I_{\max}$  and  $I_{\min}$  be the brightest and darkest **allowable** gray levels in image  $\mathbf{I}$  (e.g. 0 & 255)

- Define the simple probabilities

$$p_{s-p}(q) = \begin{cases} (1-p)/2 & ; q = -1 \\ p & ; q = 0 \\ (1-p)/2 & ; q = 1 \end{cases}$$

- If  $\mathbf{J}$  is the noisy image, then

$$J(i, j) = \begin{cases} I_{\min} & ; q = -1 \\ I(i, j) & ; q = 0 \\ I_{\max} & ; q = 1 \end{cases}$$

- This is a **simple** model for the worst case type of bit errors.

# Statistical Mean

- The **statistical mean** of  $\mathbf{N}$  is

$$\mu_{\mathbf{N}} = \int_{\mathfrak{R}_{\mathbf{N}}} q \cdot f_{\mathbf{N}}(q) dq$$

- It measures the **center** about which the noise realizations cluster.
- For a **symmetric** PMF ( $Q$  can be  $\infty$ ),

$$\mu_{\mathbf{N}} = \int_{-Q}^Q q \cdot f_{\mathbf{N}}(q) dq = 0 \quad (\text{why?})$$

# Statistical Variance

- The **variance** of  $\mathbf{N}$  is

$$\sigma_{\mathbf{N}}^2 = \int_{\mathfrak{R}_{\mathbf{N}}} (q - \mu_{\mathbf{N}})^2 \cdot f_{\mathbf{N}}(q) dq$$

- The **standard deviation**  $\sigma_{\mathbf{N}}$  is the **average distance** from  $\mu_{\mathbf{N}}$  that the noise realizations fall.
- For symmetric (zero-mean) noise, the variance is just

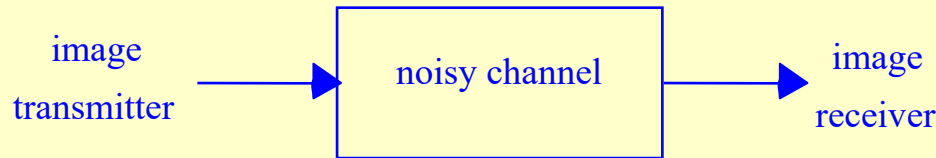
$$\sigma_{\mathbf{N}}^2 = \int_{\mathfrak{R}_{\mathbf{N}}} q^2 \cdot f_{\mathbf{N}}(q) dq$$

# Mean & Variance of PDFs

- **Exercises:** Show that for Gaussian, Laplacian, and exponential PDFs, the **variance** is  $\sigma^2$ .
- Also show that for exponential PDF, the **mean** is equal to  $\sigma$ .

# Denoising

- Consider **denoising** of an image contaminated by **white noise**:



- Two goals of filtering:
  - **Smoothing** - reduce noise (bit errors, channel, etc)
  - **Preservation** - of features: **edges** and **detail**

# Smoothing vs. Preservation

- **Image denoising** contains **conflicting goals**:
  - Smoothing: **high noise frequencies** are attenuated
  - **Broadband images**: both low and high frequencies are important
- Linear filtering **cannot** differentiate between **desirable** high frequencies and **undesirable** high frequencies
- A **linear** low-pass denoising filter will always:
  - **Reduce** high frequency **noise**
  - **Blur** the image

# WHY NONLINEAR FILTERING ?

- Linear filtering has **important** limitations.
- The basis of linear filtering is **frequency** or **spectrum shaping**:
  - **reduce** (attenuate) unwanted frequencies
  - **amplify** or **preserve** desired frequencies

# Motivation for Nonlinear Filtering

(or nonlinear processing of linear filter responses)

- Generally, nonlinear filtering **cannot** be expressed by linear convolution **nor** by frequency shaping.
- Some approaches **nonlinearly modify linear responses**
- Others **combine filter bank responses** nonlinearly
- We will study these in the important context of **image denoising**.
- The common theme: **nonlinear filters** give you capabilities that linear filters **don't have**.

# MORPHOLOGICAL FILTERS

# MORPHOLOGICAL FILTERS

- **Recall:** For image  $\mathbf{I}$ , window  $\mathbf{B}$ , the **windowed set** at  $(m,n)$  is:

$$\mathbf{B} \circ \mathbf{I}(m,n) = \{I(m-p, n-q); (p, q) \in \mathbf{B}\} .$$

- A nonlinear filter  $F$  on a **windowed set** is a nonlinear function of the pixels covered by the window.
- Denote the **nonlinear filter**  $F$  on  $\mathbf{B} \circ \mathbf{I}(m,n)$  by

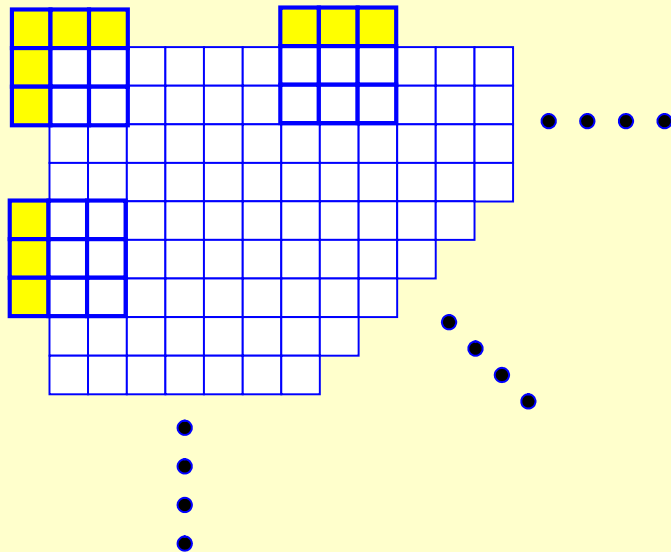
$$J(m,n) = F \{ \mathbf{B} \circ \mathbf{I}(m,n) \} = F \{ I(m-p, n-q); (p, q) \in \mathbf{B} \}$$

- Performing this at every pixel gives a **filtered image**:

$$\mathbf{J} = F[\mathbf{I}, \mathbf{B}] = [J(m,n); 0 \leq m \leq M-1, 0 \leq n \leq N-1]$$

# Boundary-of-Image Processing

- As with binary morphological filters: when a window overlaps "empty space" ...



- Pixel replication is assumed: fill the "empty" window slots by the nearest image pixel.

# MEDIAN FILTER

- The **median filter** is a nonlinear filtering device that is related to the binary majority filter (Module 2).
- Despite being “automatic” (no design), it is **very effective for image denoising**.
- Despite its simplicity, it has an **interesting theory** that justifies its use.
- The median filter is a special member of several classes of filters, including two studied here: gray-level **morphological filters**, and **order statistic filters**.

# Order Statistics

- Definition: Given a set of numbers

$$X = \{X_1, X_2, \dots, X_{2P+1}\}$$

the **order statistics** (or **OS**) of the set are the **same elements** reordered from **smallest** to **largest**.

- The OS of X are denoted

$$X = \{X_{(1)}, X_{(2)}, \dots, X_{(2P+1)}\}$$

such that

$$X_{(1)} \leq X_{(2)} \leq \dots \leq X_{(2P+1)}$$

# Special Order Statistics

- In particular:

$$\text{MIN} \{X_1, X_2, \dots, X_{2P+1}\} = X_{(1)}$$

$$\text{MAX} \{X_1, X_2, \dots, X_{2P+1}\} = X_{(2P+1)}$$

$$\text{MED} \{X_1, X_2, \dots, X_{2P+1}\} = X_{(P+1)}$$

- The **median** MED is the **middle value** in rank.

# Median Filter

- Given image **I** and window **B**, the **median filtered image** is:

$$\mathbf{J} = \text{MED}[\mathbf{I}, \mathbf{B}]$$

- Each output is the **median** of the windowed set:

$$\begin{aligned} J(m,n) &= \text{MED} \{ \mathbf{B} \circ \mathbf{I}(m,n) \} \\ &= \text{MED} \{ I(m-p, n-q); (p, q) \in \mathbf{B} \} \end{aligned}$$

# Properties of the Median Filter

- The median filter **smooths** additive white noise.
- The median filter **does not degrade edges**.
- The median filter is **particularly effective** for **removing** large-amplitude noise **impulses**.

# Example - 1-D Median Filter

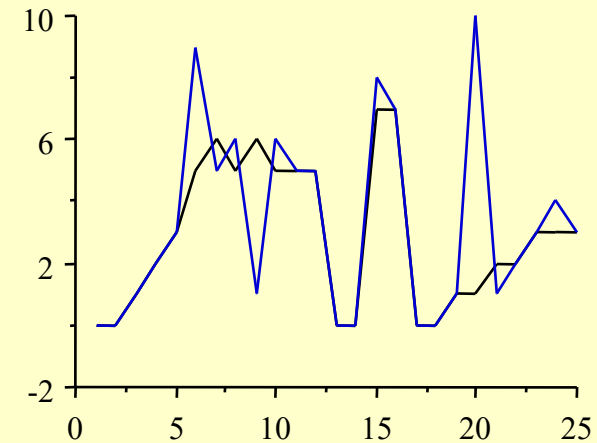
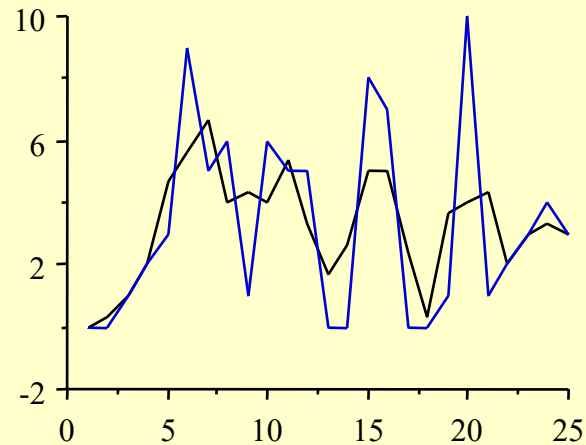
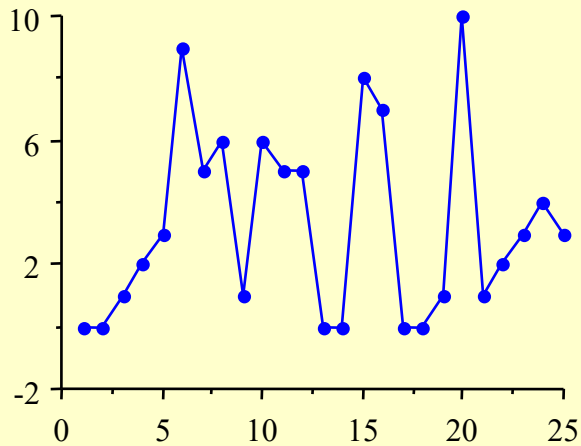


image scan line

$AVE[I, B], B = ROW(3)$

$MED[I, B], B = ROW(3)$

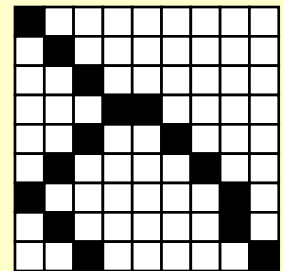
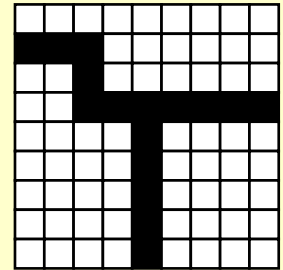
- Both filters smooth, but AVE **blurs** the structures.
- Using MED, the "noise" is **smoothed effectively** - **large noise spikes** are **eradicated**, rather than blurred.
- MED maintains signal structure better - the edges are **sharp**.

# Comments on Window Shape

- **B** = SQUARE smooths noise strongly, but may damage image details of interest.
- **B** = CROSS reduces these effects, if known that the image contains horizontal/vertical features.

Example: Images of dot-matrix characters.

- If the image contains curvilinear details that are diagonally oriented, then **B** = CROSS can perform poorly.
- **Variations**: combine the outputs of multiple CROSS, X-SHAPED, ROW, COL, etc. windows e.g. by taking their MED, AVE, etc. These "**detail-preserving filters**" work pretty well but the theory is **heuristic**.



# Erosion and Dilation

- Given an image **I** and a window **B**:

$$\mathbf{J} = \text{DILATE}[\mathbf{I}, \mathbf{B}]$$

if

$$J(i, j) = \text{MAX} \{ \mathbf{B} \circ \mathbf{I}(i, j) \} \quad (\text{local max})$$

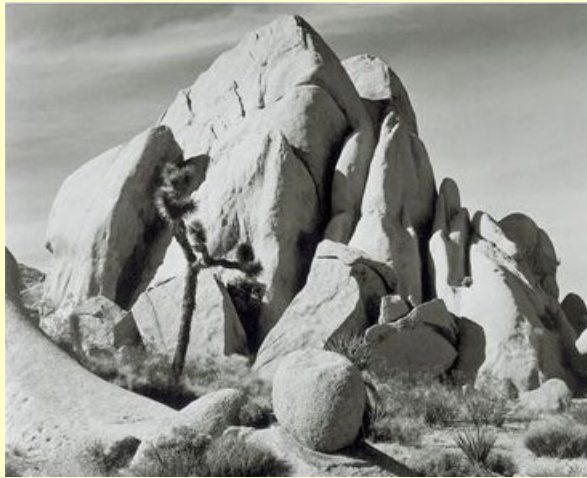
and

$$\mathbf{J} = \text{ERODE}[\mathbf{I}, \mathbf{B}]$$

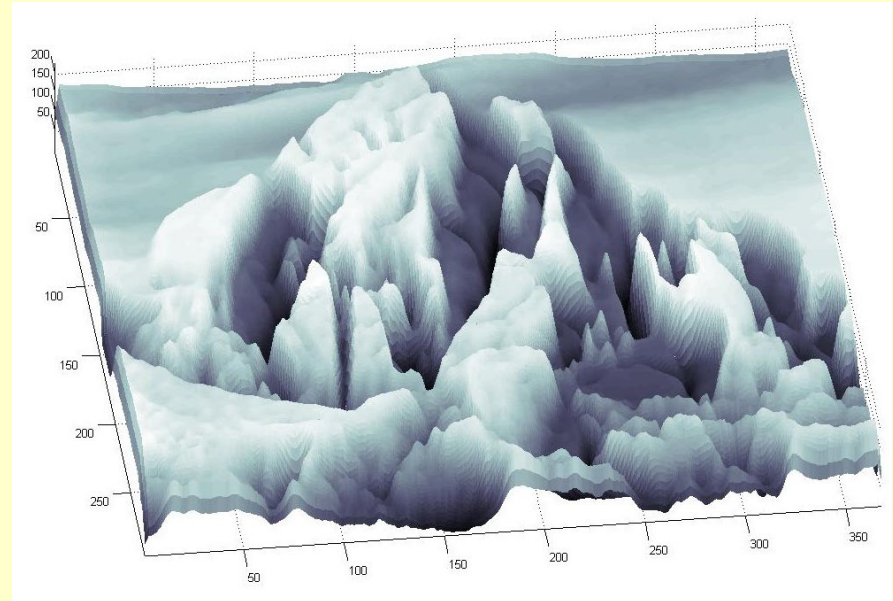
if

$$J(i, j) = \text{MIN} \{ \mathbf{B} \circ \mathbf{I}(i, j) \} \quad (\text{local min})$$

# Qualitative Properties of Morphological Filters

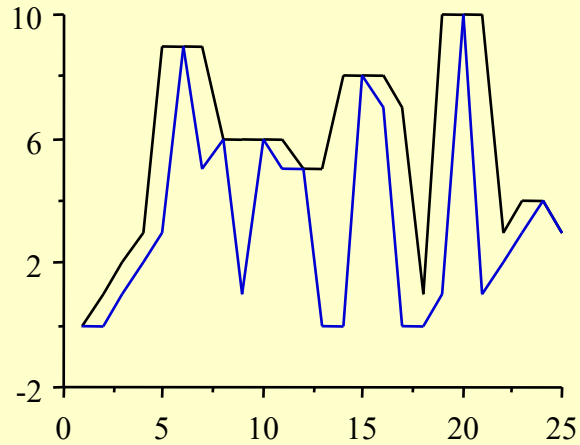


Ansel Adams' *Joshua Tree*

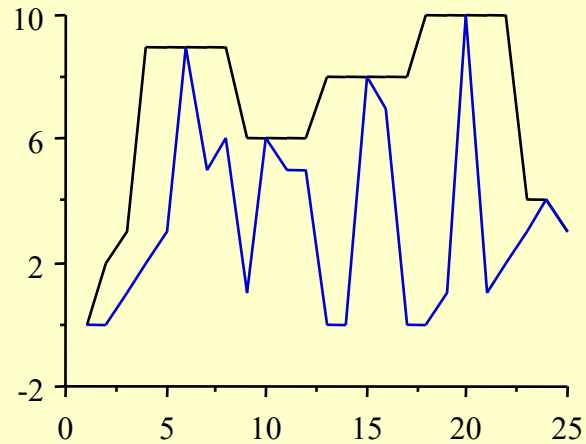


- **Gray-level morphological filters** effect shape – in this case, the shape of the intensity surface of the image.
- The DILATE (ERODE) filters have the **following properties**:
  - Increase the size of peaks (valleys)
  - Decrease the size of, or eliminate, valleys (peaks)

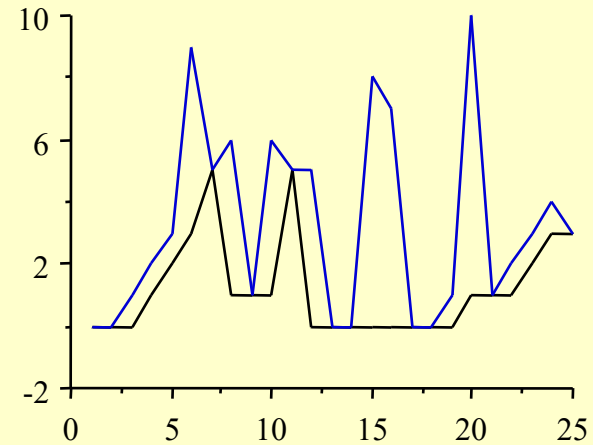
# Dilation and Erosion Examples



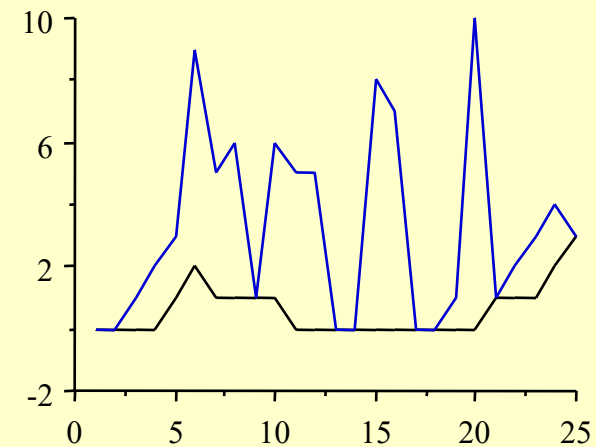
DILATE[I, B], B = ROW(3)



DILATE[DILATE[I, B], B], B = ROW(3)



ERODE[I, B], B = ROW(3)



ERODE[ERODE[I, B], B], B = ROW(3)

Most valleys gone after one pass. All valleys and negative-going impulses gone after two.

Most positive impulses reduced or gone after one pass. After two, all gone.

# CLOSE and OPEN Filters

- The **CLOSE** and **OPEN** filters are defined by:

$$\begin{aligned} \mathbf{J} &= \text{CLOSE}[\mathbf{I}, \mathbf{B}] \\ &= \text{ERODE} [\text{DILATE} [\mathbf{I}, \mathbf{B}], \mathbf{B}] \end{aligned}$$

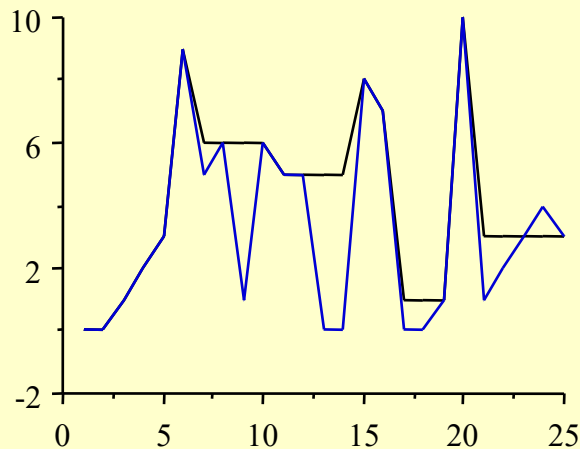
and

$$\begin{aligned} \mathbf{J} &= \text{OPEN}[\mathbf{I}, \mathbf{B}] \\ &= \text{DILATE} [\text{ERODE} [\mathbf{I}, \mathbf{B}], \mathbf{B}] \end{aligned}$$

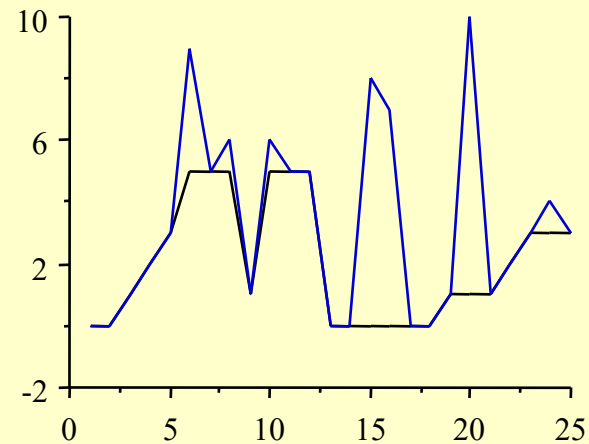
# Properties of CLOSE and OPEN

- Effective smoothing filters similar to the **median filter**.
- The CLOSE (OPEN) filter has the following properties:
  - **smooths noise**
  - preserves **edges**
  - **eradicates** negative- (positive-) going **impulses**
  - leaves the signal at approximately the same **level**

# Close and Open Examples



CLOSE[I, B], B = ROW(3)



OPEN[I, B], B = ROW(3)

- CLOSE preserves the signal except that the negative-going impulses are gone.
- OPEN preserves the signal except the positive-going impulses are gone.

# OPEN-CLOSE & CLOSE-OPEN Filters

- The **OPEN-CLOSE** and **CLOSE-OPEN** filters:

$$\begin{aligned} \mathbf{J} &= \text{OPEN-CLOSE} [\mathbf{I}, \mathbf{B}] \\ &= \text{OPEN} [\text{CLOSE} [\mathbf{I}, \mathbf{B}], \mathbf{B}] \end{aligned}$$

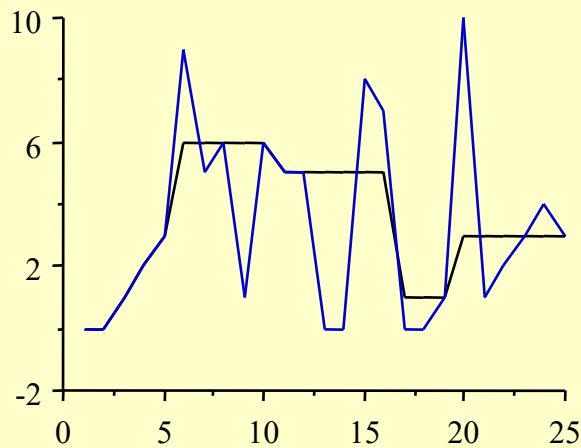
and

$$\begin{aligned} \mathbf{J} &= \text{CLOSE-OPEN} [\mathbf{I}, \mathbf{B}] \\ &= \text{CLOSE} [\text{OPEN} [\mathbf{I}, \mathbf{B}], \mathbf{B}] \end{aligned}$$

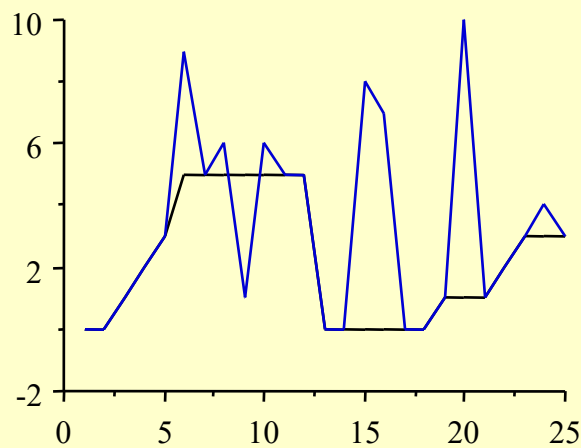
# OPEN-CLOSE & CLOSE-OPEN

- Effective smoothing filters **very** similar to the median filter. Also very similar to **each another**.
- The CLOSE-OPEN and OPEN-CLOSE filters obey the following:
  - both **smooth noise**
  - both **preserve edges**
  - both **eradicate** positive & negative **impulses**

# Close-Open & Open-Close Examples



OPEN-CLOSE[I, B], B = ROW(3)



CLOSE-OPEN[I, B], B = ROW(3)

- Note the **intense smoothing of noise** (especially impulses) and the retention of the global image structure.
- Note the **different possible interpretations** of "correct image structure" under CLOSE-OPEN and OPEN-CLOSE.

# Application: Peak/Valley Detection

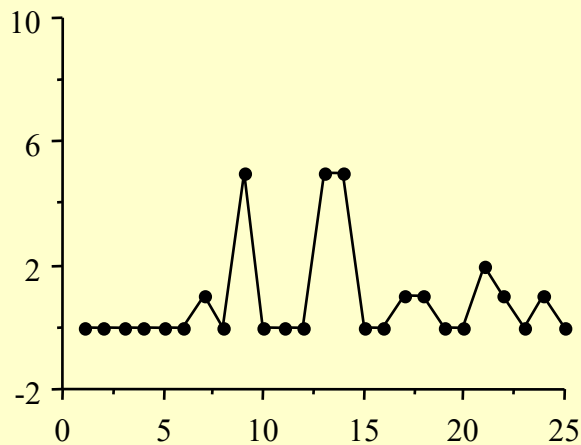
- Suppose we wish to find **bright targets** or **dark spots**.
- **Difference** the OPENed and CLOSEd image **I**:

$$\mathbf{J}_{\text{peak}} = \mathbf{I} - \text{OPEN}[\mathbf{I}, \mathbf{B}]$$

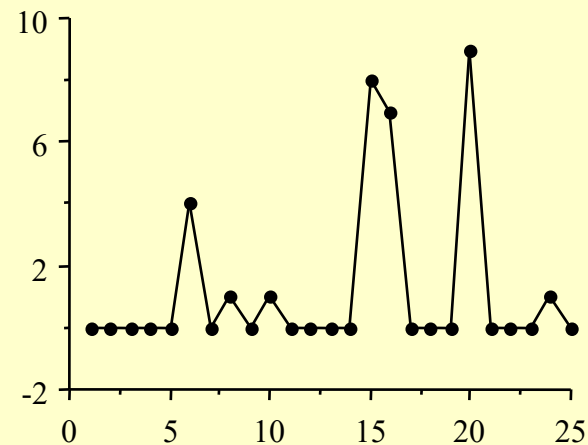
$$\mathbf{J}_{\text{valley}} = \text{CLOSE}[\mathbf{I}, \mathbf{B}] - \mathbf{I}$$

- As we'd expect, these operations highlight the **peaks** and **valleys** that occur.

# Peak/Valley Detection Examples



I-OPEN[I, B], B = ROW(3)

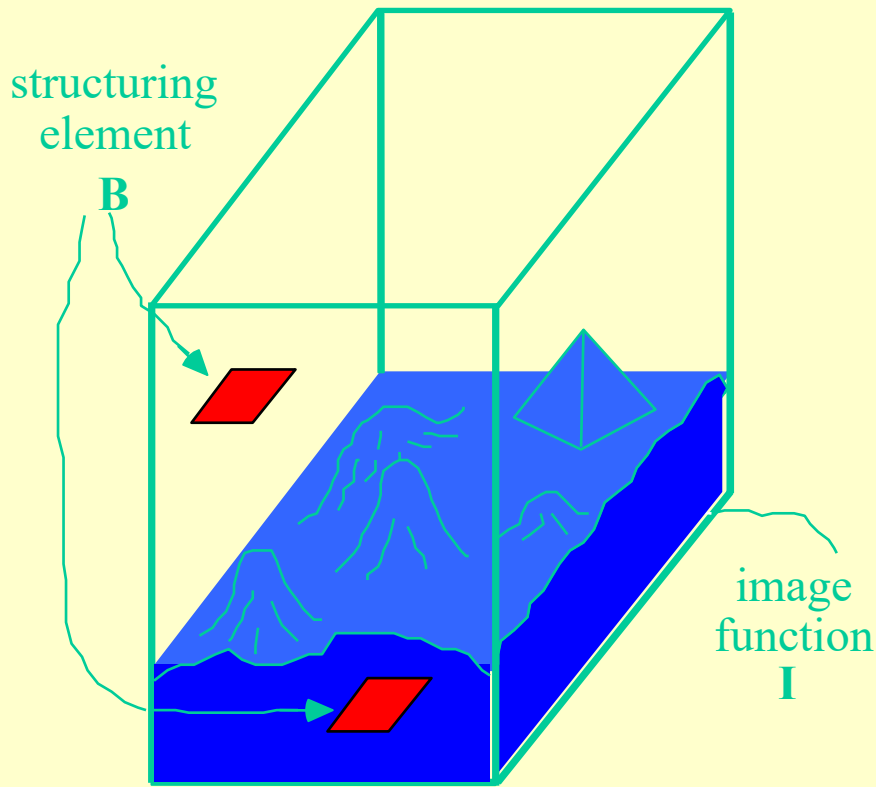


CLOS[I, B]-I, B = ROW(3)

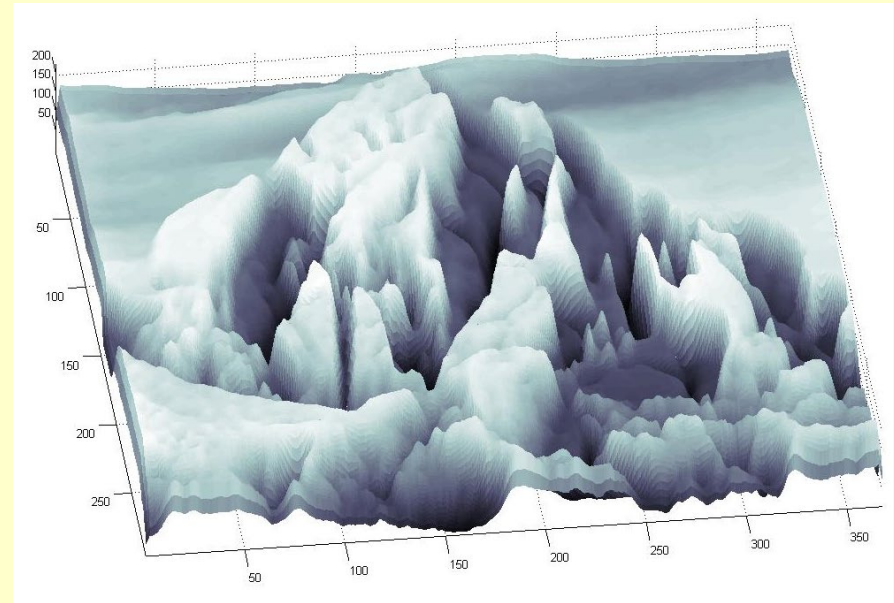
- Simple **thresholding** (binarization) of these could be used to determine the peak and valley **locations**.

# Morphological Interpretation

- DILATE and ERODE are true morphological filters in the sense discussed in Module 2.
- As are CLOSE, OPEN, CLOSE-OPEN, and OPEN-CLOSE.
- They have the **identical interpretation** if we regard
  - **I** as a **3-D binary image** with value '1' below its "plot" and '0' above its "plot"
  - **B** as a floating **2-D structuring element**



Simulated image surface



Real image surface

- Leads to **fast** all-Boolean algorithms & architectures.

# Morphological Interpretation

## ERODE:

- When **B** lies **above I**, the AND of all it touches is '0'.
- When **B** lies **below I**, the AND of all it touches is '1'.
- Whenever **B crosses the boundary** of **I**, the AND of all it touches is '0'.

## DILATE:

- When **B** lies **above I**, the OR of all it touches is '0'.
- When **B** lies **below I**, the OR of all it touches is '1'.
- Whenever **B crosses the boundary** of **I**, the OR of all it touches is '1'.

# Relating Grayscale & Binary Morphological Filters

- For many morphological filters, there is a precise relationship between the grayscale and binary versions.
- As usual, we will consider the 1D case to develop an understanding of this.
- Here, the term **m-ary** is a synonym for **grayscale**.
  - Implies pixels/samples take one of **m** distinct values.
  - For B-bit pixels/samples,  $m = 2^B$  (= K in Modules 1-3).
- Several concepts involved:
  - Threshold decomposition and reconstruction.
  - The **stacking property**.
  - The notion of a **positive boolean function** or PBF.

# Threshold Decomposition

- Let  $x[n]$  be an  $m$ -ary signal.
- Define the binary threshold signal

$$x_{\geq\tau}[n] = \begin{cases} 1, & \text{for all } n \text{ s.t. } x[n] \geq \tau, \\ 0, & \text{for all } n \text{ s.t. } x[n] < \tau. \end{cases}$$

- The **threshold decomposition** of  $x[n]$  is defined by the set of threshold signals  $x_{\geq\tau}[n]$  for  $\tau = [0, 1, 2, \dots, m-1]$ .

# Threshold Decomposition Example

$x[n] =$	5	2	7	0	3	4
$x_{\geq 7}[n] =$	0	0	1	0	0	0
$x_{\geq 6}[n] =$	0	0	1	0	0	0
$x_{\geq 5}[n] =$	1	0	1	0	0	0
$x_{\geq 4}[n] =$	1	0	1	0	0	1
$x_{\geq 3}[n] =$	1	0	1	0	1	1
$x_{\geq 2}[n] =$	1	1	1	0	1	1
$x_{\geq 1}[n] =$	1	1	1	0	1	1
$x_{\geq 0}[n] =$	1	1	1	1	1	1

- Let  $B=3$ , so that  $m = 2^B = 8$ .
- The pixels/samples take values in  $[0, 7]$ .
- Each threshold signal is **binary**.
- In each column, notice that there must be a **stack** of 0's on top of a **stack** of 1's.
- This is called the **stacking property**.

# Threshold Reconstruction

$x[n] =$	5	2	7	0	3	4
$x_{\geq 7}[n] =$	0	0	1	0	0	0
$x_{\geq 6}[n] =$	0	0	1	0	0	0
$x_{\geq 5}[n] =$	1	0	1	0	0	0
$x_{\geq 4}[n] =$	1	0	1	0	0	1
$x_{\geq 3}[n] =$	1	0	1	0	1	1
$x_{\geq 2}[n] =$	1	1	1	0	1	1
$x_{\geq 1}[n] =$	1	1	1	0	1	1
$x_{\geq 0}[n] =$	1	1	1	1	1	1

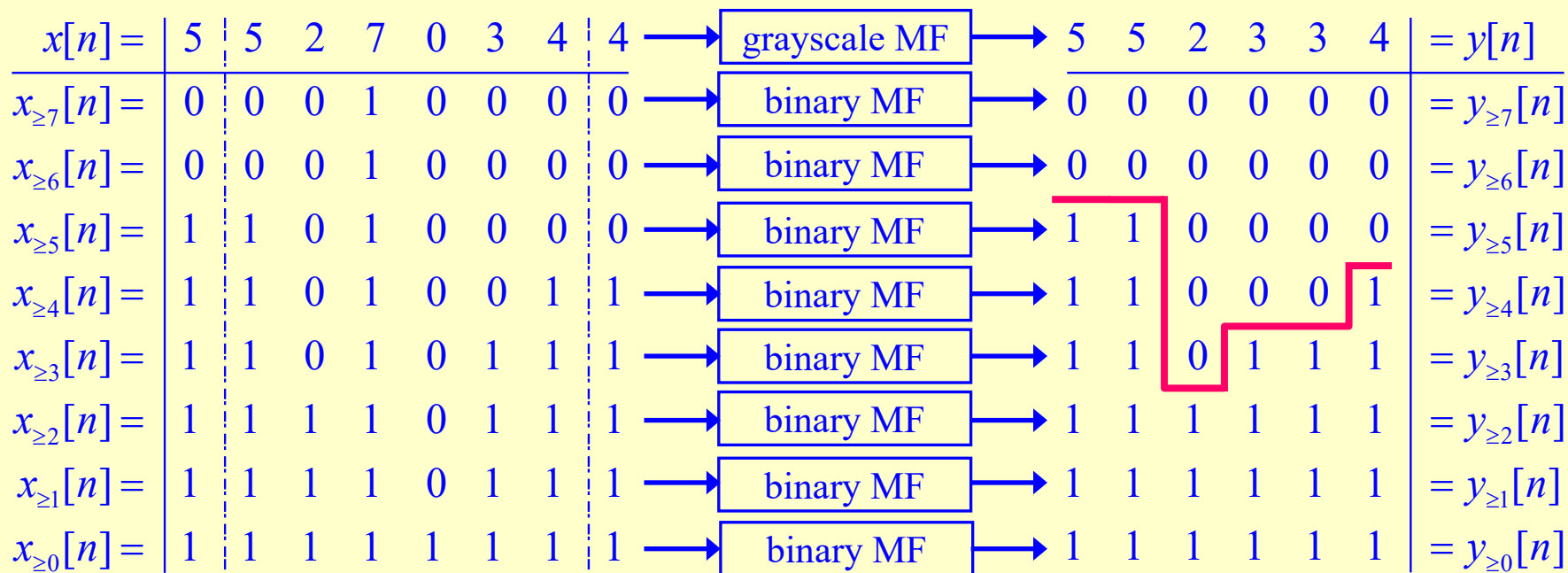
- Note that  $x[n]$  can be reconstructed from any set of threshold signals that obey the stacking property:

$$x[n] = \max \{ \tau \mid x_{\geq \tau}[n] = 1 \}.$$

- This is shown by the red line to the left.

# Median Filtering by Threshold Decomposition

- On the next page, we will apply a **grayscale** 3-point median filter to  $x[n]$ .
- In parallel, we will apply **binary** 3-point median filters to each of the threshold signals  $x_{\geq \tau}[n]$ .
- As is typical for morphological and order statistic filters, we will handle the edge effects by replication.



- The result of applying the grayscale median filter to the m-ary signal is the **same** as the result of applying the binary median filters to the threshold signals!

# When Does this Work?

- Amazing!
- Does this work for other types of filters?
- The answer is YES.
  - It works for any filter that preserves the stacking property at the outputs of the binary filters.
  - Filters that preserve the stacking property are called **stack filters**. They are a **very large** class of nonlinear filters.
- Fact: the stacking property is preserved if the binary version of the filter is a positive boolean function (PBF).
- Def: a PBF is a binary function where the minimum sum of products form contains no complemented variables.
- Ex: 3-pt binary MF:  $y = x_1x_2 + x_1x_3 + x_2x_3$ .

# Stack Filters

- Stack filters are important for two main reasons:
  1. There is a theoretical framework for designing stack filters to minimize the mean absolute error (MAE) in the output signal.
  2. There are techniques for elegant fast hardware implementations.
- Examples of 3-point PBF's that define stack filters:
  - MED:  $y = x_1x_2 + x_1x_3 + x_2x_3$
  - DILATE/MAX:  $y = x_1 + x_2 + x_3$
  - ERODE/MIN:  $y = x_1x_2x_3$

# Morphological Filters as Stack Filters

- Because ERODE and DILATE are stack filters, so are OPEN, CLOSE, Close-Open, and Open-Close.
- This is the “detailed version” of what is meant by the figure on page 6.42.
- Ex: OPEN( $x[n]$ ,  $\mathbf{B}$ ) with  $\mathbf{B}=\text{ROW}(3)$ :
  - Because OPEN( $x[n]$ ,  $\mathbf{B}$ ) = DILATE[ERODE( $x[n]$ ,  $\mathbf{B}$ )], this is actually a 5-point filtering operation.
  - The PBF for the binary filters is given by

$$y = x_1x_2x_3 + x_2x_3x_4 + x_3x_4x_5$$

- It takes a lot more work to derive the PBFs for CLOSE, Close-Open, and Open-Close.

# Fast Implementations

- Fast implementations of grayscale morphological and order statistic filters are challenging because they require **sorting** the pixels in the windowed set.
- But the binary versions of these filters can be implemented with boolean logic: AND, OR, XNOR, which do not require sorting.
- For stack filters, fast integrated circuits can be designed to:
  - perform the threshold decomposition,
  - filter the binary threshold signals,
  - reconstruct the output from the filtered threshold signals.

# Fast Sorting

- Sorting is an expensive operation, especially for large window sizes.
- Let  $M$  be the number of pixels in the windowed set.
- Time complexity of common sorting algorithms:
  - $\mathcal{O}(M^2)$ : quicksort, insertion sort, selection sort...
  - $\mathcal{O}(M \log M)$ : merge sort, heapsort, introsort...
- In hardware, space complexity can often be traded one-for-one for time complexity.
- If the binary version of the filter is a PBF, then there are **very fast** bit-serial realizations that require only  $B$  clock cycles.

# Bit-Serial Median Filtering

- This algorithm is due to Delman (Proc. SPIE, vol. 298, 1981, pp. 184-188).
- For an image  $\mathbf{I}$ , suppose we want to implement the filter  $\mathbf{J} = \text{MED}(\mathbf{I}, \mathbf{B})$  with  $\mathbf{B} = \text{CROSS}(5)$ .
- Suppose  $\mathbf{B}$ =3-bit pixels and let the windowed set be

$$\mathbf{B} \circ \mathbf{I}(i, j) = \{2, 0, 1, 5, 6\}.$$

- The median is 2.
- With good sorting hardware,
  - insertion sort requires **at least** 25 clock cycles.
  - merge sort requires **at least** 15 clock cycles.
- The bit-serial algorithm finds the median in only three clock cycles!

# First Clock Cycle

Original Data

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

Majority → 0--

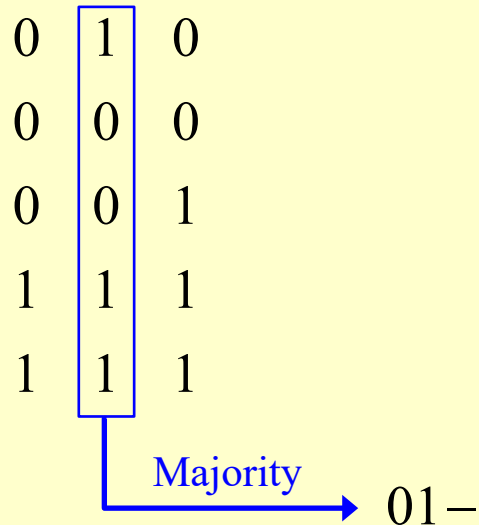
First Change

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

- Take the majority of the hi-order bits. This is the hi-order bit of the median (a zero in this case).
- Any pixel with a hi-order bit of one is **bigger** than the median.
  - Changing it to **all ones** will **not** change the median.
- This is the **First Change**, which is clocked into the registers by the rising edge of the second clock cycle.

# Second Clock Cycle

First Change



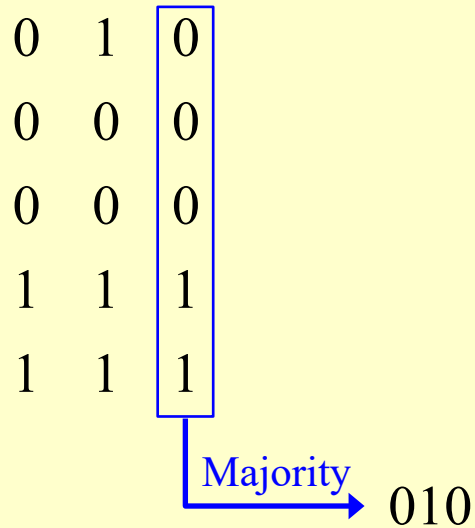
- Take the majority of the second bits. This is the second bit of the median (a one in this case).
- Any pixel with a second bit of zero is **smaller** than the median.
  - Changing it to **all zeros** will **not** change the median.
- This is the **Second Change**, which is clocked into the registers by the rising edge of the third clock cycle.

Second Change

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

# Third Clock Cycle

Second Change



- Take the majority of the lo-order bits. This is the lo-order bit of the median (a zero in this case).
  - We are done! The median is found in only three clock cycles!
  - This approach was generalized by Chen to develop fast bit-serial realizations for **any** stack filter (IEEE Trans. Circuits, Syst., vol. 36, no. 6, June 1989, pp. 785-794).
- 
- For B-bit input pixels, each output pixel is computed in only B clock cycles.

# ORDER STATISTIC FILTERS

# ORDER STATISTIC FILTERS

- Recall the OS of a set  $\mathbf{X} = \{X_1, \dots, X_{2P+1}\}$ :

$$\mathbf{X}_{OS} = \{X_{(1)}, \dots, X_{(2P+1)}\}$$

such that

$$X_{(1)} \leq \dots \leq X_{(2P+1)}.$$

- Define a **linear combination of order statistics**

$$\sum_{p=1}^{2P+1} A_p X_{(p)} = \mathbf{A}^T \mathbf{X}_{OS}$$

where  $\mathbf{A} = [A_1, \dots, A_{2P+1}]^T$ . This defines a class of nonlinear filters called **order statistic (OS) filters**.

# OS Filter Coefficients

- The **filter weights** are assumed to **sum to 1**:

$$\sum_{p=1}^{2P+1} A_p = \mathbf{A}^T \mathbf{e} = 1$$

where  $\mathbf{e} = [1, 1, \dots, 1]^T$ . Then if  $\mathbf{X} = c \cdot \mathbf{e}$  is **constant** then  $\mathbf{A}^T \mathbf{X}_{OS} = c$ , and the filter is then **level-preserving** – like a low-pass linear filter.

- The OS coefficients are assumed **symmetric**:

$$A_i = A_{2P+2-i} \text{ for } 1 \leq i \leq 2P+1$$

# Defining OS filters

- Given an image  $\mathbf{I}$ , a window  $\mathbf{B}$ , and a coefficient set  $\mathbf{A}$ , the OS filter with coefficients  $\mathbf{A}$  is defined by:

$$\mathbf{J} = \text{OS}_{\mathbf{A}}[\mathbf{I}, \mathbf{B}]$$

if

$$J(m,n) = \mathbf{A}^T \{ \mathbf{B} \circ \mathbf{I}(m,n) \}_{\text{OS}}$$

where

$$\{ \mathbf{B} \circ \mathbf{I}(m,n) \}_{\text{OS}}$$

are the OS of

$$\{ \mathbf{B} \circ \mathbf{I}(m,n) \}.$$

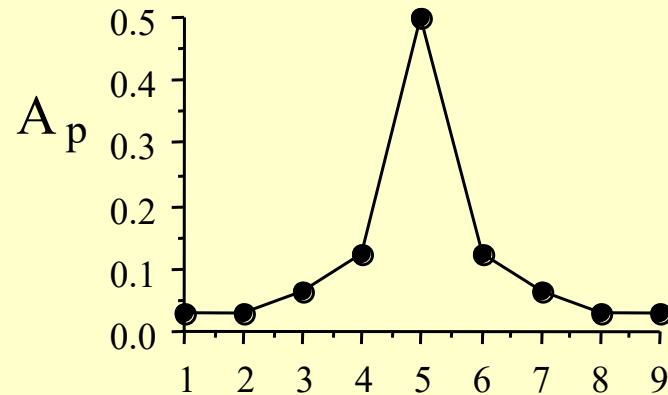
- These are just the **ordered** values of the pixels covered by the window  $\mathbf{B}$  when it's centered at  $(m,n)$ .

# General Properties of OS Filters

- The **behavior** of a filter  $OS_{\mathbf{A}}[\mathbf{I}, \mathbf{B}]$  is largely determined by its coefficients  $\mathbf{A}$ .
- Some important OS filters  $OS_{\mathbf{A}}[\mathbf{I}, \mathbf{B}]$  are:
  - $\mathbf{A} = [0, \dots, 0, 1, 0, \dots, 0]^T$  - **median filter**
  - $\mathbf{A} = \left[ \frac{1}{2P+1}, \dots, \frac{1}{2P+1} \right]^T$  - **average filter**

# Median-Like OS Filters

- Generally, if the coefficients  $\mathbf{A}$  are concentrated near the middle:



then  $OS_{\mathbf{A}}[\mathbf{I}, \mathbf{B}]$  is much like a **median filter**:

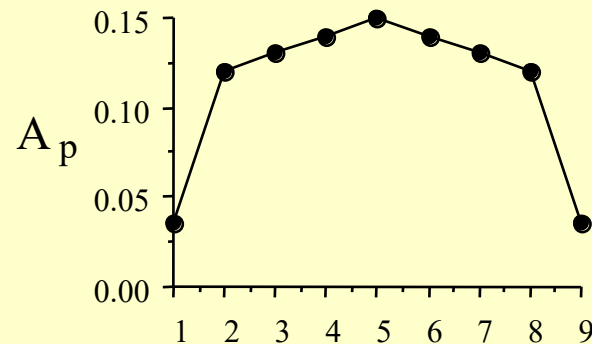
- preserve edges well
- smooth noise
- reduce impulses

**BUT**

- will not "streak" or "blotch" as much
- will suppress non-impulse noise better (more later)

# Average-Like OS Filters

- Generally, if the coefficients are widely distributed:



then  $OS_A[\mathbf{I}, \mathbf{B}]$  will act much like an **average filter**:

- preserve edges poorly (better than AVE)
  - smooth noise
  - blur impulses (not as badly as AVE)
- We will now explore the utility of these filters relative to some of the noise models we have looked at.

# OS Filter Design Criterion

- The goal of a smoothing filter can be stated as **reduce the noise variance.**
- OS filters are best characterized by their ability to reduce noise variance.
- For certain well-known noise PMFs, specific choices of the coefficients  $\mathbf{A}$  of  $OS_{\mathbf{A}}[\mathbf{J}, \mathbf{B}]$  result using the variance reduction criterion.

# Gaussian Noise – Average Filter

- The average filter is **best** among all OS filters at reducing **Gaussian** noise variance. It is also the best linear filter. In fact, it is the best filter of any kind!
- If **B** contains  $2P+1$  elements, then  $AVE[\mathbf{J}, \mathbf{B}]$  reduces (divides) the variance of Gaussian noise by a factor  
 $2P+1$
- In fact,  $AVE[\mathbf{J}, \mathbf{B}]$  reduces the variance of **any white noise** by a factor  
 $2P+1$
- This is pretty good, but for non-Gaussian PMFs other OS filters can do much better.
- Major caveat: The above **does not account** for the **effect on the signal**. Average filtering blurs the signal as it reduces noise, so there exists a **tradeoff** between **noise reduction** and **signal blurring**.

# Laplacian Noise – Median Filter

- The median filter  $\text{MED}[\mathbf{J}, \mathbf{B}]$  is the asymptotically **best** OS filter for reducing the variance of **Laplacian** noise (as window size increases).
- If  $\mathbf{B}$  contains  $2P+1$  elements, then  $\text{MED}[\mathbf{J}, \mathbf{B}]$  reduces Laplacian noise variance by a factor of (approximately)  
 $2 \cdot (2P+1)$  - quite an **improvement!**
- $\text{MED}[\mathbf{J}, \mathbf{B}]$  is **fantastic** for removing salt-and-pepper noise. No better filter!
- If the noise is Gaussian, then  $\text{MED}[\mathbf{J}, \mathbf{B}]$  reduces the noise variance by a factor of (approximately) only  
 $(0.75) \cdot (2P+1)$  - **not** an **improvement!**
- **Extended Caveat:** The median filter is noted to **not** blur images, even as the window size increases.

# Median Filter vs. Average Filter

- Owing to its very important **edge-preserving property**, MED is generally regarded as **superior** to AVE (or any other linear filter!) for **reducing white noise in images**.
- However, as with any application, care should be taken in analyzing the situation rather than blindly applying a filter.
- Both MED and AVE are used **far too often** in situations where a design process might be **more advantageous**.

# Robust OS Filter Coefficients

- A fairly sophisticated **statistical analysis** yields the **best**  $OS_A$  filter for **specific** noise **and** image models.
- Fortunately, a great property of **certain** OS filters is the fact that they are **robust**.
- A filter is **robust** if it gives **close to** optimal performance for a **wide variety** of noise types.
- OS filters  $OS_A$  have been shown to be **remarkably robust** for coefficients  $A$  that resemble **a cross between** MED and AVE.

# Trimmed Mean Filters

- A simple robust OS filter class is the **trimmed mean** OS filters  $TM_Q[\mathbf{J}, \mathbf{B}]$  with coefficients

$$\mathbf{A} = \left[ \underbrace{0, \dots, 0}_Q, \underbrace{\frac{1}{2(P-Q)+1}, \dots, \frac{1}{2(P-Q)+1}}_{2(P-Q)+1}, \underbrace{0, \dots, 0}_Q \right]^T$$

where  $Q < P$  (usually  $Q \approx P/2$ ).

- **Similar to AVE** since the innermost  $2(Q-P)+1$  OS are averaged. If  $Q = 0$ , then  $TM_Q[\mathbf{J}, \mathbf{B}] = AVE[\mathbf{J}, \mathbf{B}]$ .
- **Similar to MED** since *only* inner OS are averaged, the rest of the OS being “trimmed.” Indeed, if  $Q = P$ , then  $TM_Q[\mathbf{J}, \mathbf{B}] = MED[\mathbf{J}, \mathbf{B}]$ .

# Trimmed Mean Filter Performance

- Trimmed mean filters **smooth noise well** (including impulse and Gaussian noise) and **preserve edges well**.
- For **both** Gaussian **and** Laplacian noise,  $TM_{P/2}$  reduces the variance by a factor  
**> (0.9)·OPTIMAL**

where **OPTIMAL** is  $(2P+1)$  for Gaussian noise and  $2 \cdot (2P+1)$  for Laplacian noise.

# Generalized Gaussian PDF

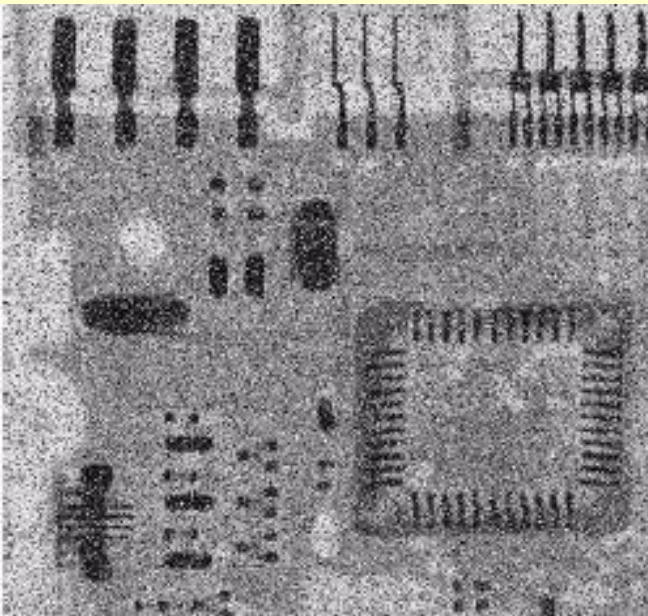
- The **generalized gaussian** PDF is used to model wide behaviors of noise:

$$f_{\gamma}(x) = K e^{-|x/\sigma|^{\gamma}} \quad -\infty < x < \infty$$

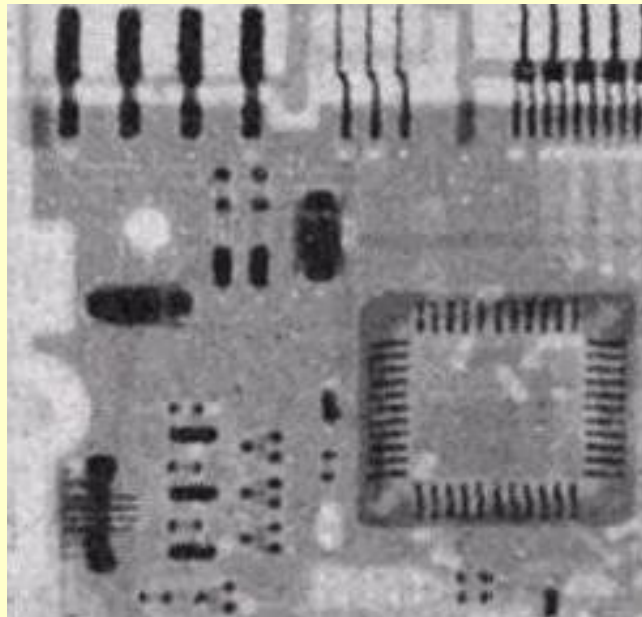
- For  $\gamma < 1$ ,  $f_{\gamma}(x)$  is **very heavy-tailed**.
- For  $\gamma = 1$ ,  $f_1(x)$  is **laplacian**.
- For  $\gamma = 2$ ,  $f_2(x)$  is **gaussian**.
- As  $\gamma \rightarrow \infty$ ,  $f_{\infty}(x)$  is **uniform**.
- For a wide range of values of  $\gamma$

**$TM_{P/2} > (0.9) \cdot \text{OPTIMAL!}$**

# Example – AWGN / S&P Noise



AWGN and S&P Noise



50% Trimmed Mean

Time-consuming!  
Good examples:  
N=3, SQUARE, P=2  
N = 5, CROSS, P=3  
AWGN with S&P

# BILATERAL FILTER

# BILATERAL FILTER

- Another principled approach to **image denoising**.
- Like OS filters, it **modifies linear filtering**.
- Like OS filters, it seeks to smooth while **retaining (edge) structures**.

# Bilateral Filter Concept

- **Observation:** 2-D linear filtering is a method of **weighting by spatial distances** [ $\mathbf{i} = (i, j)$ ,  $\mathbf{m} = (m, n)$ ]:

$$J_G(\mathbf{i}) = \sum_{\mathbf{m}} I(\mathbf{m}) G(\|\mathbf{m} - \mathbf{i}\|)$$

- If linear filter  $G$  is **isotropic**, e.g., a 2-D gaussian LPF\*

$$G(i, j) = K_G \exp\left[-(i^2 + j^2) / \sigma_G^2\right]$$

then  $\|\mathbf{m} - \mathbf{i}\| = \sqrt{(m-i)^2 + (n-j)^2}$  is Euclidean **distance**.

\*The value of  $K_G$  is such that  $G$  has unit volume.

# Bilateral Filter Concept

- **New idea:** Instead of weighting by spatial distance, weight by **luminance similarity**

$$J_H(\mathbf{i}) = \sum_{\mathbf{m}} I(\mathbf{m}) H \left[ I(\mathbf{m}) - I(\mathbf{i}) \right]$$

- **NOT** a linear filter operation.  $H$  could be a 1-D gaussian weighting function\*

$$H(i) = K_H \exp \left[ -i^2 / \sigma_H^2 \right]$$

- **Not very useful by itself.**

\*The value of  $K_H$  is such that  $H$  has unit area.

# Bilateral Filter Definition

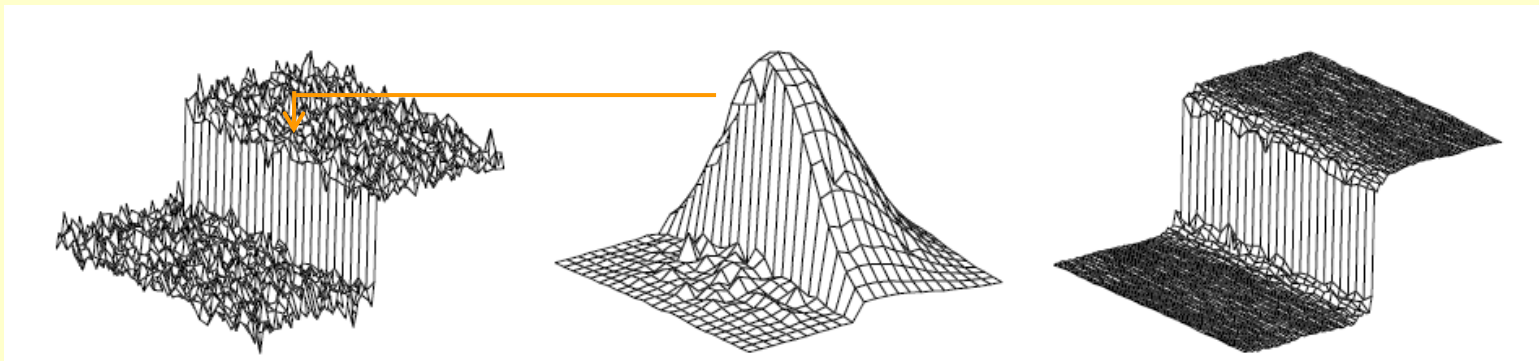
- **Combine** spatial distance weighting with luminance similarity weighting:

$$J(\mathbf{i}) = \sum_{\mathbf{m}} I(\mathbf{m}) G(\|\mathbf{m} - \mathbf{i}\|) H[I(\mathbf{m}) - I(\mathbf{i})]$$

- **Computation** is **straightforward** and about twice as expensive as just spatial linear filtering.
- However, this can be sped up by pre-computing all values of  $H$  for every pair of luminances, and storing them in a **look-up table** (LUT).

# Example

- **Noisy edge** filtered by bilateral filter (Tomasi, ICCV 1998).
- Uses gaussian LPF and **gaussian similarity**



Before

Bilateral weighting 2  
pixels to right of edge

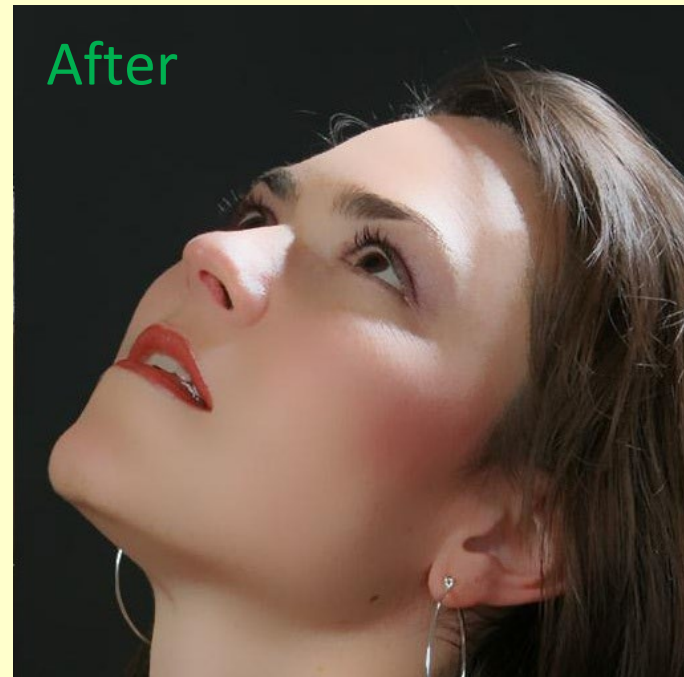
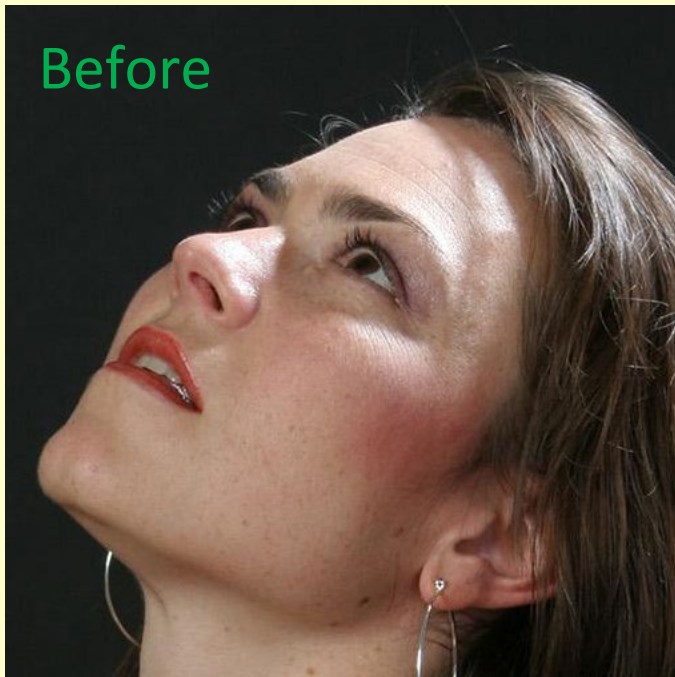
After

# Color Example

- Can extend to **color** using **3-D similarity weighting**.
- If  $\mathbf{I} = [\mathbf{R}, \mathbf{G}, \mathbf{B}]$ , then use

$$H[\mathbf{R}(\mathbf{m})-\mathbf{R}(\mathbf{i}), \mathbf{G}(\mathbf{m})-\mathbf{G}(\mathbf{i}), \mathbf{B}(\mathbf{m})-\mathbf{B}(\mathbf{i})]$$

where  $H(i, j, k) = K_H \exp\left[-(i^2 + j^2 + k^2) / \sigma_H^2\right]$



# Comments on Bilateral Filter

- **Similar to median filter**, emphasizes luminances close in value to current pixel.
- Does not have a statistical (or other) design basis, so **best used with care**, e.g., **retouching** or **low-level noise smoothing**.
- The idea of doing **linear filtering that is decoupled near edges** is very powerful.
- This is also the theme of the **next method**.

# NON-LOCAL (NL) MEANS

# NL Means Concept

- **Idea:** Estimate the current pixel luminance as a weighted average of **all** pixel luminances
- The **weights** are decided by neighborhood **luminance similarity**.
- The **weight changes** from pixel to pixel.
- In all the following,  $\mathbf{i} = (i, j)$ ,  $\mathbf{m} = (m, n)$ ,  $\mathbf{p} = (p, q)$ .
- The method is **expensive** but **effective**.

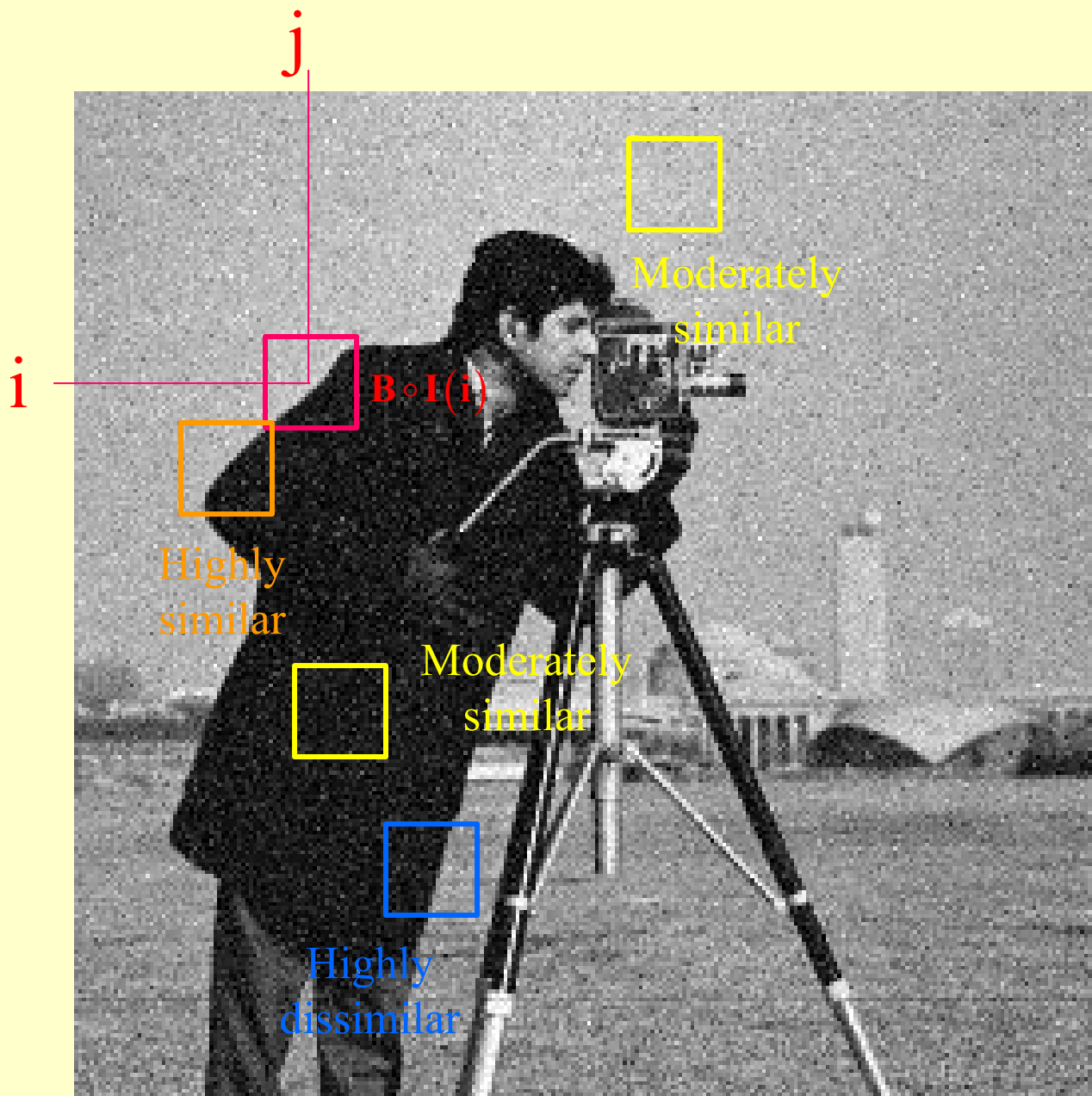
# NL Means Concept

- Given a window  $\mathbf{B}$ , compute the **luminance similarity** of every windowed set  $\mathbf{B} \circ \mathbf{I}(\mathbf{i})$  with every other windowed set  $\mathbf{B} \circ \mathbf{I}(\mathbf{m})$ :

$$W(\mathbf{i}, \mathbf{m}) = K_w \exp \left[ -\frac{\|\mathbf{B} \circ \mathbf{I}(\mathbf{i}) - \mathbf{B} \circ \mathbf{I}(\mathbf{m})\|}{\sigma_w^2} \right]$$

where

$$\begin{aligned} \|\mathbf{B} \circ \mathbf{I}(\mathbf{i}) - \mathbf{B} \circ \mathbf{I}(\mathbf{m})\| &= \sum_{\mathbf{p} \in \mathbf{B}} [I(\mathbf{i} - \mathbf{p}) - I(\mathbf{m} - \mathbf{p})]^2 \\ &= \sum_{(\mathbf{p}, \mathbf{q}) \in \mathbf{B}} [I(\mathbf{i} - \mathbf{p}, \mathbf{j} - \mathbf{q}) - I(\mathbf{m} - \mathbf{p}, \mathbf{n} - \mathbf{q})]^2 \end{aligned}$$



# NL Means Definition

- Given a window  $\mathbf{B}$ , and the **weighting function**  $W$  just defined, the **NL-means filtered image**  $\mathbf{I}$  is

$$\mathbf{J}(\mathbf{i}) = \sum_{\mathbf{m}} W(\mathbf{m}, \mathbf{i}) I(\mathbf{m})$$

- Thus each pixel value is replaced by a **weighted sum of all luminances**, where the **weight increases with neighborhood similarity**.

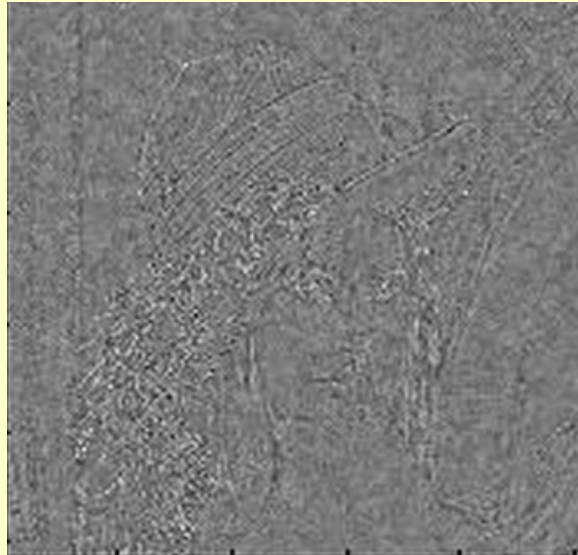
# Example



Before

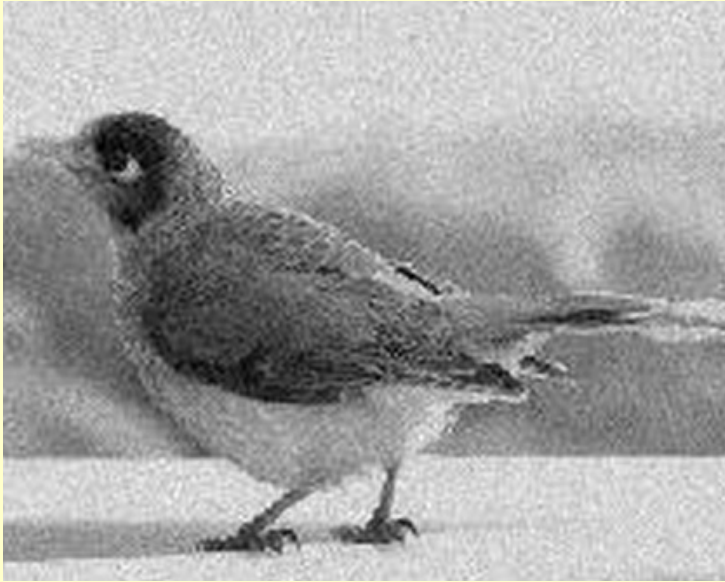


After



Difference

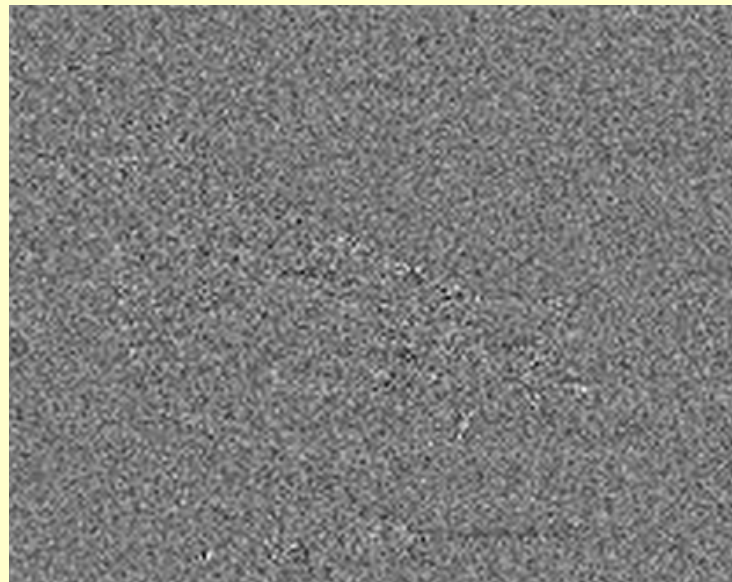
# Example



Before



After



Difference

# Comments

- **NL-Means** is very similar in spirit to **frame averaging**.
- The main drawback is the large computation / search.
- It can be modified to compare/search **less of the image**.
- It **works best** when the image contains a lot of **redundancy** (periodic, large smooth regions, textures).
- It can **fail** at **unique image patches**.

# ANISOTROPIC DIFFUSION

# ANISOTROPIC DIFFUSION

- A powerful approach to both image denoising and **edge detection**.
- Fact: Gaussian blur or smoothing is analogous to physical **diffusion** (e.g., of heat).
- In fact the gaussian function is a solution to a PDE called the **diffusion equation**.

# Concept of Anisotropic Diffusion

- Diffuse (smooth image pixels) where the gradient magnitude is small.
- **Inhibit diffusion** where gradient magnitude is large.
- **Anisotropic** since smoothing **depends on direction**.
- Convolution with a Gaussian is **isotropic diffusion**.
  
- **Rationale:**
  - Smooth image without destroying image detail
  - Inhibit smoothing across edges (inter-region smoothing)
  - Encourage smoothing between edges (intra-region smoothing)

# The Diffusion Equation

- **Continuous Formulation** - a system of partial differential equations:

$$\frac{\partial \mathbf{I}}{\partial t} = \text{div}[\mathbf{c}\nabla \mathbf{I}] = c_{EW} \frac{\partial^2 \mathbf{I}(x, y)}{\partial x^2} + c_{NS} \frac{\partial^2 \mathbf{I}(x, y)}{\partial y^2} \quad \mathbf{c} = \begin{bmatrix} c_{EW} & 0 \\ 0 & c_{NS} \end{bmatrix}$$

where  $\mathbf{I}$  is the image,  $\mathbf{c}$  is a 2x2 matrix of **diffusion coefficients**, and  $\text{div}$  is the **divergence operator**

- The discrete formulation is much easier to understand...

Note: Recall

$$\text{div}(\mathbf{F}) = \frac{\partial f_1}{\partial x} + \frac{\partial f_2}{\partial y} \quad \text{where } \mathbf{F} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

# Approximating Derivatives

$$\frac{\partial \mathbf{I}}{\partial t} \rightarrow I_{t+1}(m, n) - I_t(m, n)$$

where time  $t$  will be iteration number of the diffusion process.

$$\frac{\partial^2 I(x, y)}{\partial x^2} \rightarrow I(m, n+1) - 2I(m, n) + I(m, n-1)$$

$$\frac{\partial^2 I(x, y)}{\partial y^2} \rightarrow I(m+1, n) - 2I(m, n) + I(m-1, n)$$

which approximates twice derivatives by twice differences.

# Diffusion Equation: Discrete Form

- Iterative approximation using **differences**:

$$I_{t+1}(m, n) = I_t(m, n) + \frac{1}{4} \left[ c_{NS}(m, n) \nabla_{NS}^{(2)} I_t(m, n) + c_{EW}(m, n) \nabla_{EW}^{(2)} I_t(m, n) \right]$$

where ...

$$\nabla_{NS}^{(2)} I_t(m, n) = I(m, n+1) - 2I(m, n) + I(m, n-1)$$

$$\nabla_{EW}^{(2)} I_t(m, n) = I(m+1, n) - 2I(m, n) + I(m-1, n)$$

Note: If  $c_{NS} = c_{EW} = c$ , then the diffusion equation becomes the *heat equation*  $\frac{\partial I}{\partial t} = c \nabla^2 I$

# Diffusion Coefficients

- Selection of the diffusion coefficients  $c_{NS}$ ,  $c_{EW}$  is important. Usually  $c_{NS} = c_{EW}$ .
- The idea is to **inhibit smoothing** over edges, **encourage smoothing** on non-edge image regions. So  $c_{NS}$ ,  $c_{EW}$  should be **edge-sensitive**.
- At each pixel  $I(i, j)$ , diffusion proceeds in a given direction according to the diffusion coefficient corresponding to that direction and that location.

# Diffusion Coefficients

- One popular form

$$c_d(m, n) = \exp \left\{ - \left[ \frac{\nabla_d I(m, n)}{k} \right]^2 \right\} \quad d = \text{NS, EW}$$

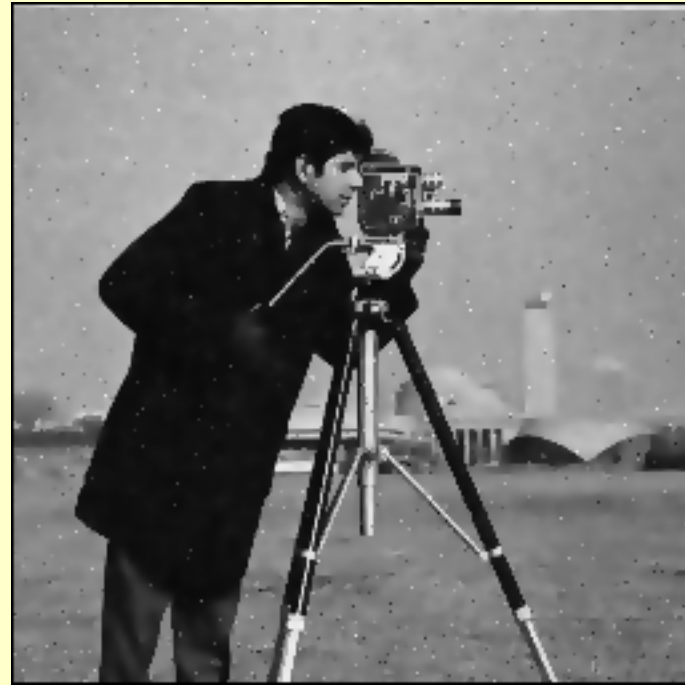
where  $k$  is an edge threshold – as the gradient increases,  $c_d$  rapidly falls.

- Properties of  $c_d(m, n)$ :
  - approaches 0 near edges – **disabling diffusion**
  - approaches 1 over smooth areas – **enabling diffusion**

# Anisotropic Diffusion Example



Laplacian Noise

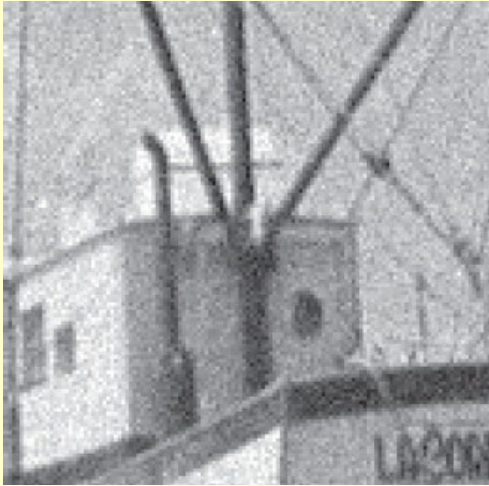


8 iterations of  
anisotropic diffusion

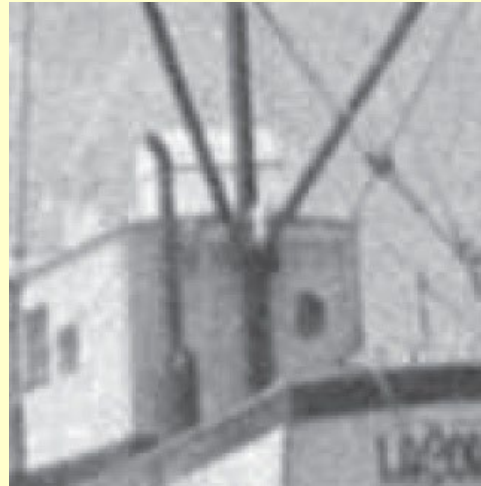
# Comments on Anisotropic Diffusion

- Really a method of alternating edge detection with **edge-decoupled smoothing**.
- Number of iterations controls degree of smoothing (scale)
- Edge detection applied to the smoothed image can be quite effective.

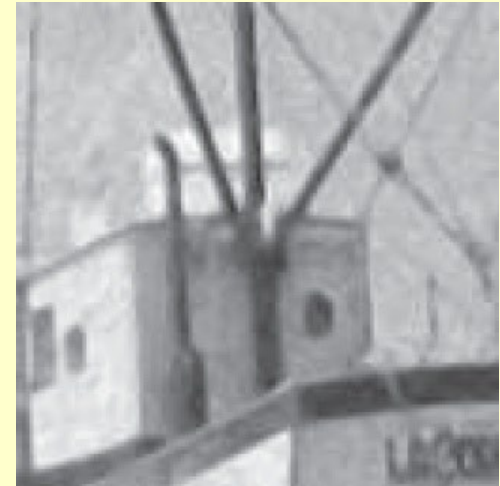
# Comparison



AWGN



Gaussian filtered



Anisotropic diffusion



Bilateral filtered

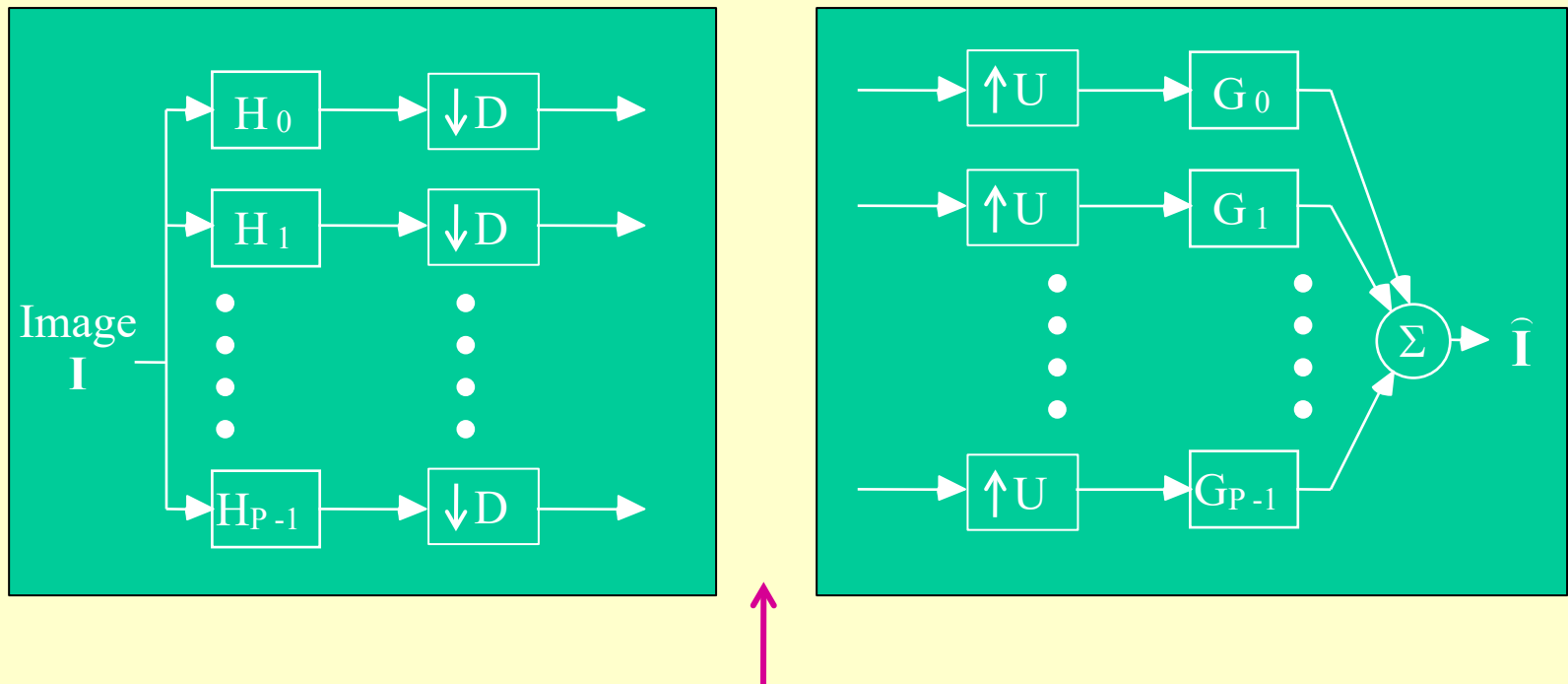


NL-Means

# WAVELET SHRINKAGE

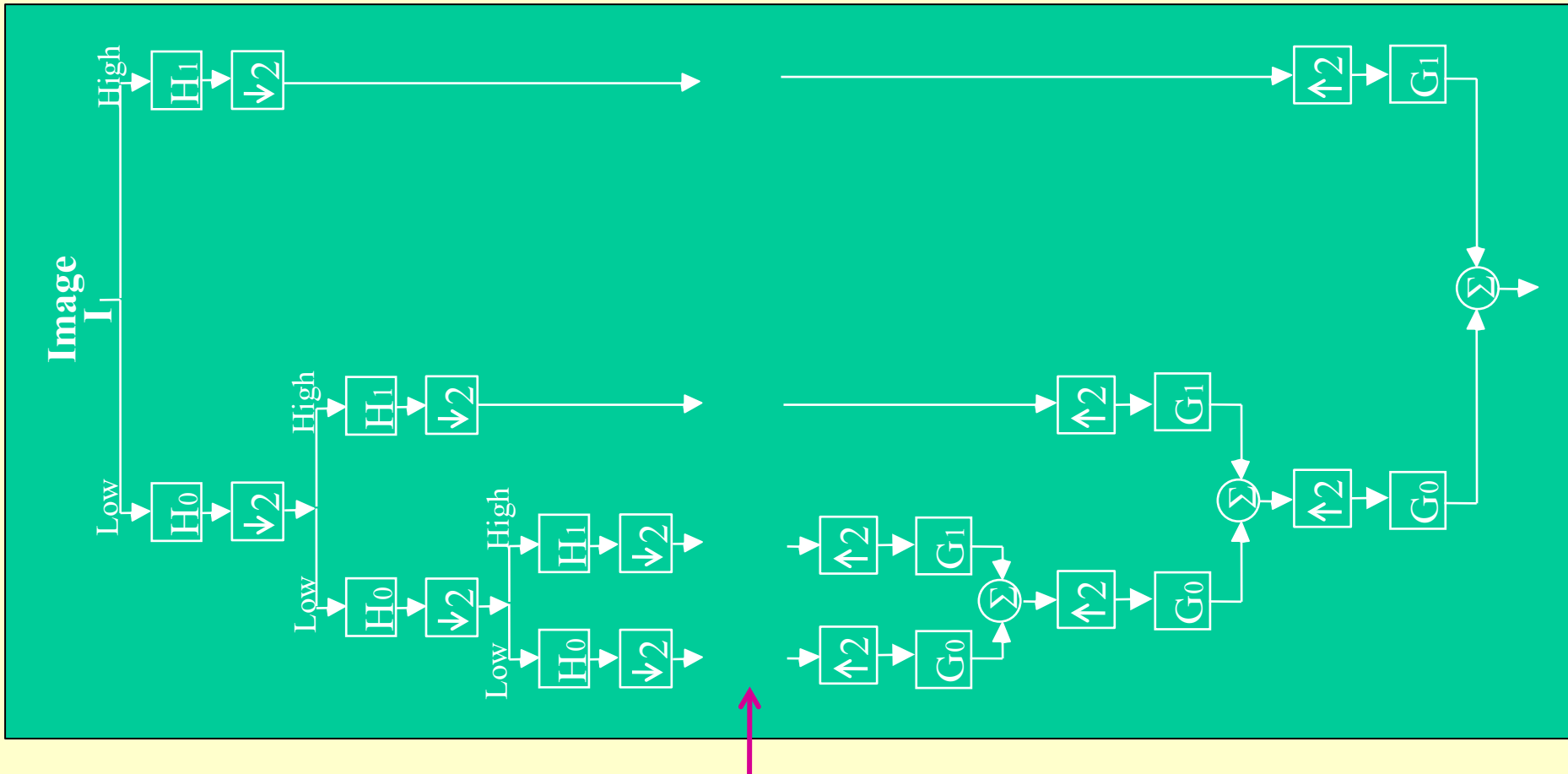
# Wavelet Shrinkage

Perfect reconstruction analysis/synthesis filterbank or DWT



Denoise the DWT coefficients

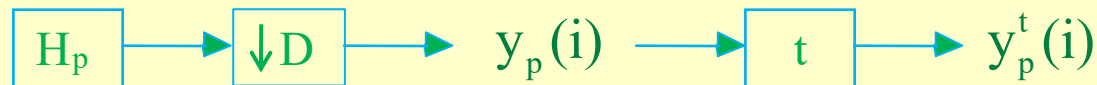
# Wavelet Shrinkage



Denoise the DWT coefficients

# Wavelet Shrinkage

- The process of **denoising** is accomplished simply by **thresholding the DWT coefficients** (filter responses).
- A **threshold is applied** ( $p \neq 0$  since baseband signal is smooth already):

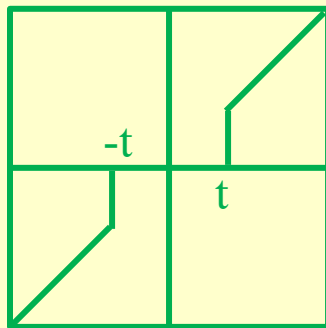


# Wavelet Soft Thresholding

- Two methods of thresholding:
  - **Hard threshold**: signal zeroed if below threshold
  - **Soft threshold**, where in addition to hard threshold, the threshold is subtracted if above threshold.

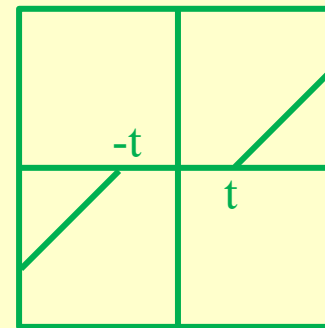
**Hard threshold**

$$y_p^t(i) = \begin{cases} y_p(i) & ; \text{if } |y_p(i)| > t \\ 0 & ; \text{if } |y_p(i)| \leq t \end{cases}$$



**Soft threshold**

$$y_p^t(i) = \text{sgn}[y_p(i)] \max\{0, |y_p(i)| - t\}$$



# Wavelet Shrinkage

- **Soft thresholding** is much more effective. It “shrinks” the wavelet coefficients.
- Works since additive **noise increases coefficient energy**. **Soft thresholding reduces the energy**.
- **Threshold** can be chosen to **optimize** many criteria such as MSE, Bayesian risk, etc.

# Wavelet Shrinkage

- **MMSE threshold (“SureShrink”)**

$$t = \sigma \sqrt{2 \log NM}$$

where  $\sigma^2$  = estimated noise variance, and  $NM$  = # pixels in the image.

- **Bayesian threshold (an SNR) (“BayesShrink”)**

$$t = \sigma_{\text{signal}} / \sigma_{\text{noise}}$$

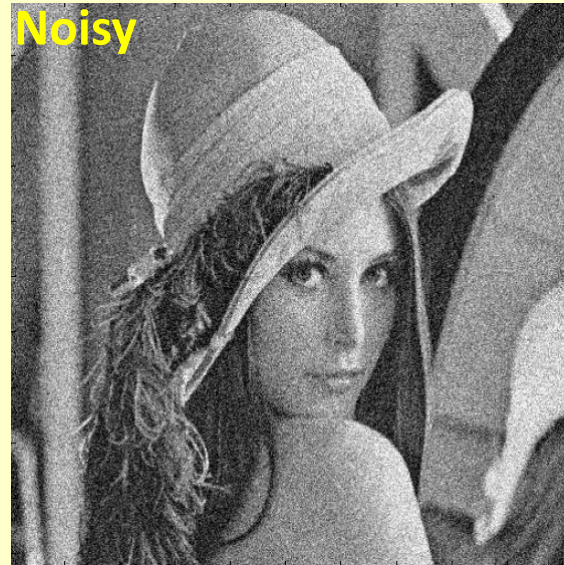
is estimated for each filter response (band), hence **adaptive**. When SNR is large, little thresholding. When large, heavy thresholding.

# Wavelet Shrinkage Examples

Original



Noisy



SureShrink



BayesShrink



# DWT Used

- There is no clear choice (yet) on wavelet filters to use.
- Here we used a 4-level DWT with the “Daubechies 10” orthogonal wavelet (D10)

## Analysis LP

$H_0(0) = 0.1601023980$   
 $H_0(1) = 0.6038292698$   
 $H_0(2) = 0.7243085284$   
 $H_0(3) = 0.1384281459$   
 $H_0(4) = -0.2422948871$   
 $H_0(5) = -0.0322448696$   
 $H_0(6) = 0.0775714938$   
 $H_0(7) = -0.0062414902$   
 $H_0(8) = -0.0125807520$   
 $H_0(9) = 0.0033357253$

## Analysis HP

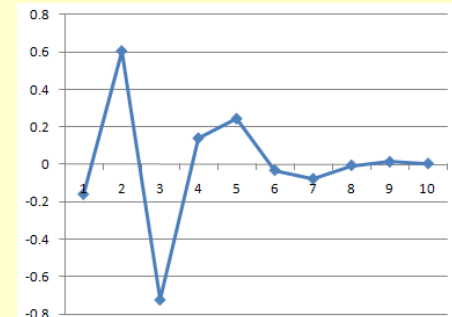
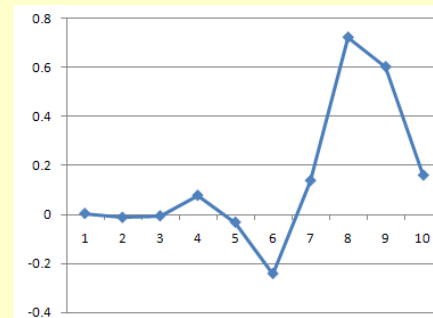
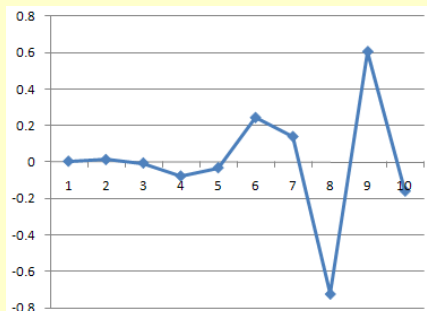
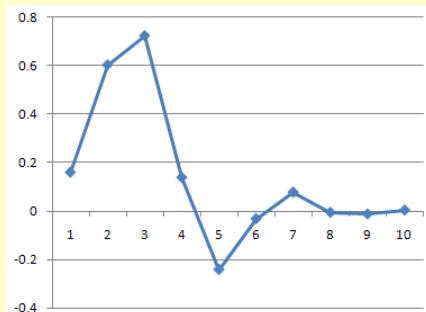
$H_1(0) = 0.0033357253$   
 $H_1(1) = 0.0125807520$   
 $H_1(2) = -0.0062414902$   
 $H_1(3) = -0.0775714938$   
 $H_1(4) = -0.0322448696$   
 $H_1(5) = 0.2422948871$   
 $H_1(6) = 0.1384281459$   
 $H_1(7) = -0.7243085284$   
 $H_1(8) = 0.6038292698$   
 $H_1(9) = -0.1601023980$

## Synthesis LP

$G_0(0) = 0.0033357253$   
 $G_0(1) = -0.0125807520$   
 $G_0(2) = -0.0062414902$   
 $G_0(3) = 0.0775714938$   
 $G_0(4) = -0.0322448696$   
 $G_0(5) = -0.2422948871$   
 $G_0(6) = 0.1384281459$   
 $G_0(7) = 0.7243085284$   
 $G_0(8) = 0.6038292698$   
 $G_0(9) = 0.1601023980$

## Synthesis HP

$G_1(0) = -0.1601023980$   
 $G_1(1) = 0.6038292698$   
 $G_1(2) = -0.7243085284$   
 $G_1(3) = 0.1384281459$   
 $G_1(4) = 0.2422948871$   
 $G_1(5) = -0.0322448696$   
 $G_1(6) = -0.0775714938$   
 $G_1(7) = -0.0062414902$   
 $G_1(8) = 0.0125807520$   
 $G_1(9) = 0.0033357253$



# Comments

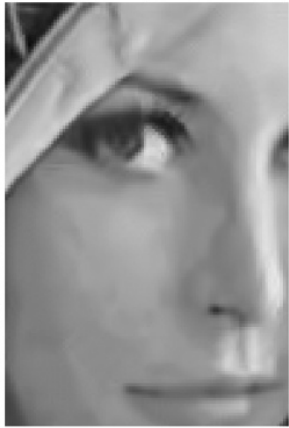
- Since DWTs can be computed **very fast**, so can **soft threshold based denoising**.
- The results are **optimal in wavelet space** and the appearance is **reasonably good**.

**BM3D**

# Overview

- Quite a **complicated** algorithm – too much!  
Uses **many** of the ideas we have covered
- Image broken into **blocks**, blocks **then matched by similarity** into groups.
- Estimates are **collaboratively formed** for each **block** within each group.
- This is done by a **transformation** of the group and a **shrinkage process**.
- Perhaps the best **denoising algorithm extant**.

# Examples



# HOMOMORPHIC FILTERING

- Assume a **multiplicative white noise** model:

$$\mathbf{J} = \mathbf{I} \otimes \mathbf{N}$$

where  $\mathbf{I}$  is the original image and  $\mathbf{N}$  is a **white noise image**.

- Thus

$$J(m,n) = I(m,n)N(m,n) \text{ for } 0 \leq m \leq M-1, 0 \leq n \leq N-1.$$

- Direct linear filtering ineffective, since the noise / image spectra not **added**.

# Multiplicative Noise

- Multiplicative noise is always **positive** since the sensed image intensities must be.
- Multiplicative noise often has an **exponential PMF** in practical applications, such as **radar, laser-based imaging, electron microscopy**, etc.
- **Synthetic aperture radar** images are an excellent example.
- Multiplicative noise appears **much worse** in bright image regions (since it multiplies the gray levels) and can be **hardly noticeable** in dark regions.

# Homomorphic Approach

- Homomorphic filtering is succinctly expressed by the **following diagram**:



- We'll study each step:

# Logarithmic Point Operation

- Idea is similar to log point operations used for histogram improvement: de-emphasize **dominant bright image pixels**.
- More than that! Since  $\log (a \cdot b) = \log (a) + \log (b)$ , then

$$\log [J(m,n)] = \log [I(m,n)] + \log [N(m,n)].$$

- So: the point operation **converts the problem** into that of smoothing

$$\mathbf{J}' = \mathbf{I}' + \mathbf{N}'$$

where

$$I'(m,n) = \log [I(m,n)], N'(m,n) = \log [N(m,n)], J'(m,n) = \log [J(m,n)]$$

- Importantly,  $\mathbf{N}'$  is **still a white noise image**.
- This is just the additive white noise problem studied earlier!

# Filtering

- The filtering problem here is very similar to the linear and nonlinear ones studied earlier in this Module and in Module 5.
- Depending on the noise statistics, AVE, MED,  $OS_A$ , OPEN-CLOSE, CLOSE-OPEN or other linear or nonlinear filter may be used.
- The **theoretical** performance of these various filters in homomorphic systems still remains an open question.
- Most will work reasonably well; in particular MED has been **observed** to be effective if the noise is exponential and the image contains edges.
- The objective is of course to produce a filtered result

$$F[\mathbf{J}', \mathbf{B}] \approx \mathbf{I}'$$

# Exponential Point Operation

- If the filtering operation **succeeds**:

$$F[\mathbf{J}', \mathbf{B}] \approx \mathbf{I}'$$

then the output image  $\mathbf{K}$  will **approximate**

$$\begin{aligned} K(i, j) &\approx \exp [I'(i, j)] \\ &= I(i, j) \end{aligned}$$

# Comments

- We shall now study **some BIG topics** – **image quality and compression...** onward to **Module 7..**