

# COMP527

## COMPUTATIONAL IMAGING

Lecture #05 – Image Filtering



KOÇ  
UNIVERSITY

Aykut Erdem // Koç University // Spring 2023

# Previously on COMP527

- Controlling exposure
- High-dynamic-range imaging
- Tonemapping



# Today's Lecture

- Gaussian filtering
- Sharpening
- Bilateral filter
- Non-local means filter
- RegCov smoothing
- Rolling guidance filter

**Disclaimer:** The material and slides for this lecture were borrowed from

- Ioannis Gkioulekas' 15-463/15-663/15-862 "Computational Photography" class
- Wojciech Jarosz's CS 89.15/189.5 "Computational Aspects of Digital Photography" class
- Steve Marschner's CS6640 "Computational Photography" class
- Kaiming He's slides on Guided Image Filtering

# Filtering

- The name “filter” is borrowed from frequency domain processing
- Accept or reject certain frequency components
- Fourier (1807):

Periodic functions  
could be represented  
as a weighted sum of  
sines and cosines

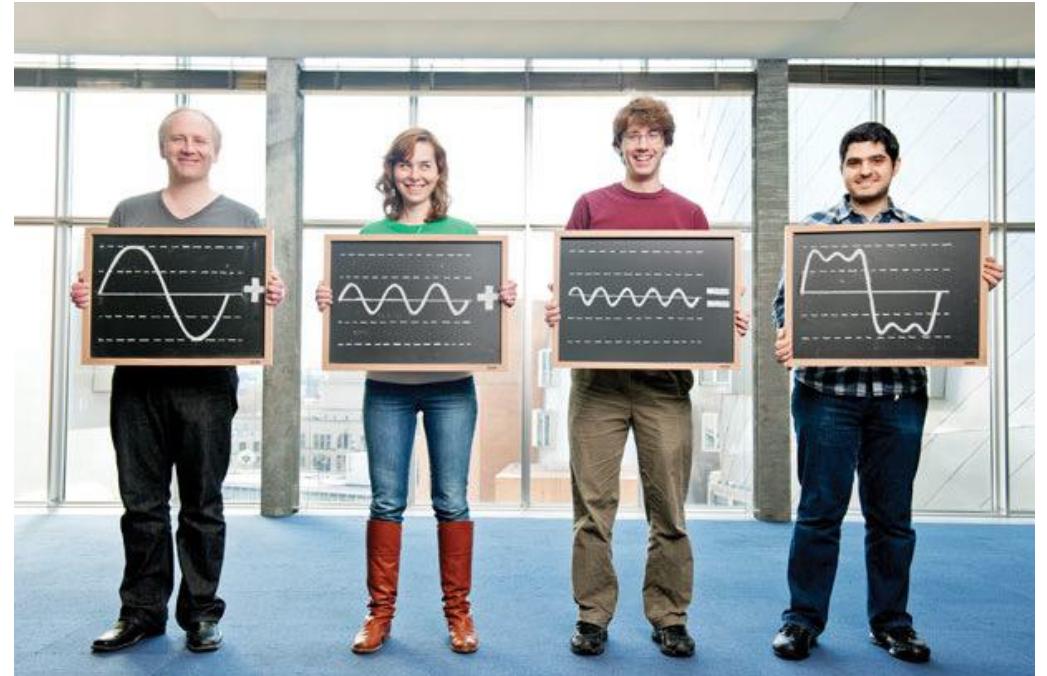
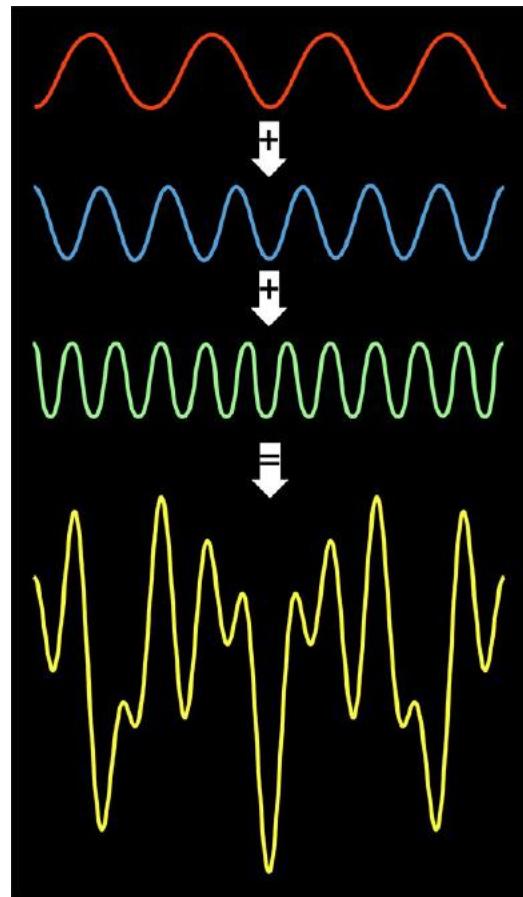


Image courtesy of Technology Review

# Signals

- A signal is composed of low and high frequency components



low frequency components: smooth / piecewise smooth

Neighboring pixels have similar brightness values  
You're within a region

high frequency components: oscillatory

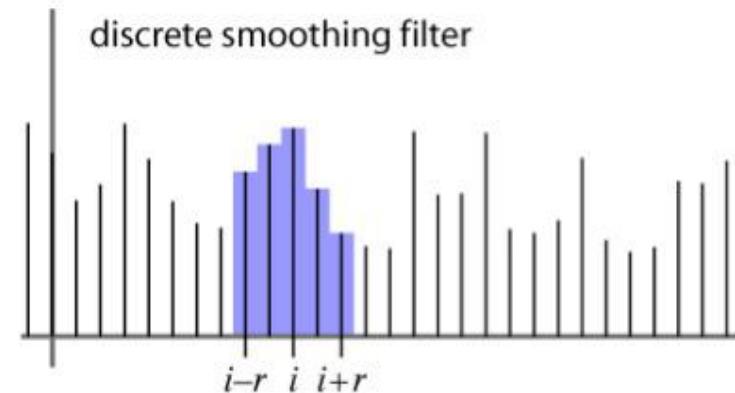
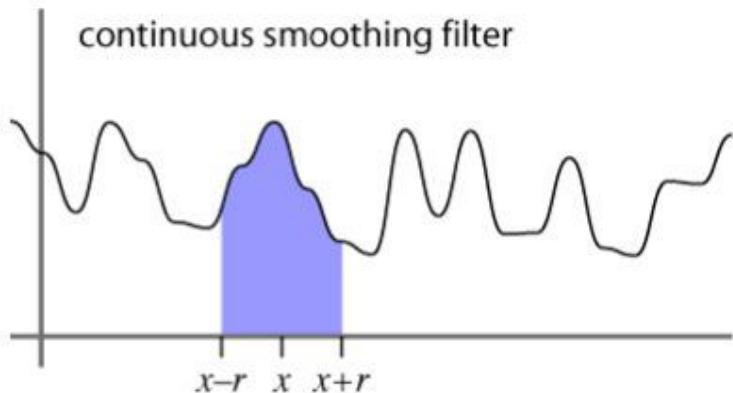
Neighboring pixels have different brightness values  
You're either at the edges or noise points

# Image Filtering

- Idea: Use the information coming from the neighboring pixels for processing
- Design a transformation function of the local neighborhood at each pixel in the image
  - Function specified by a “filter” or mask saying how to combine values from neighbors.
- Various uses of filtering:
  - Enhance an image (denoise, resize, etc)
  - Extract information (texture, edges, etc)
  - Detect patterns (template matching)

# Filtering

- Processing done on a function
  - can be executed in continuous form (e.g. analog circuit)
  - but can also be executed using sampled representation
- Simple example: smoothing by averaging
- Can be modeled mathematically by convolution



# Discrete convolution

- Simple averaging:

$$b_{\text{smooth}}[i] = \frac{1}{2r+1} \sum_{j=i-r}^{i+r} b[j]$$

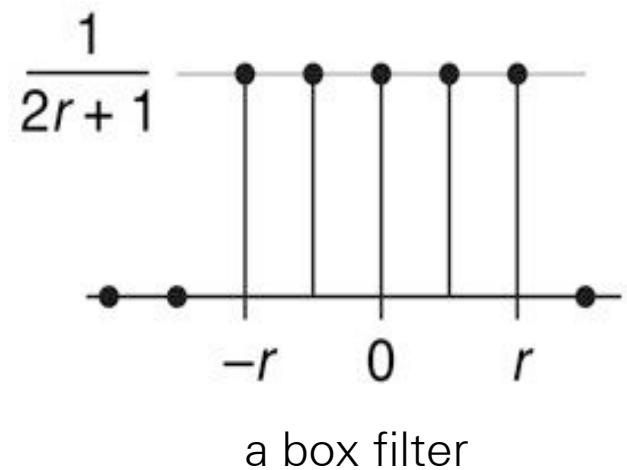
- every sample gets the same weight
- Convolution: same idea but with weighted average

$$(a \star b)[i] = \sum_j a[j]b[i - j]$$

- each sample gets its own weight (normally zero far away)
- This is all convolution is: it is a moving weighted average

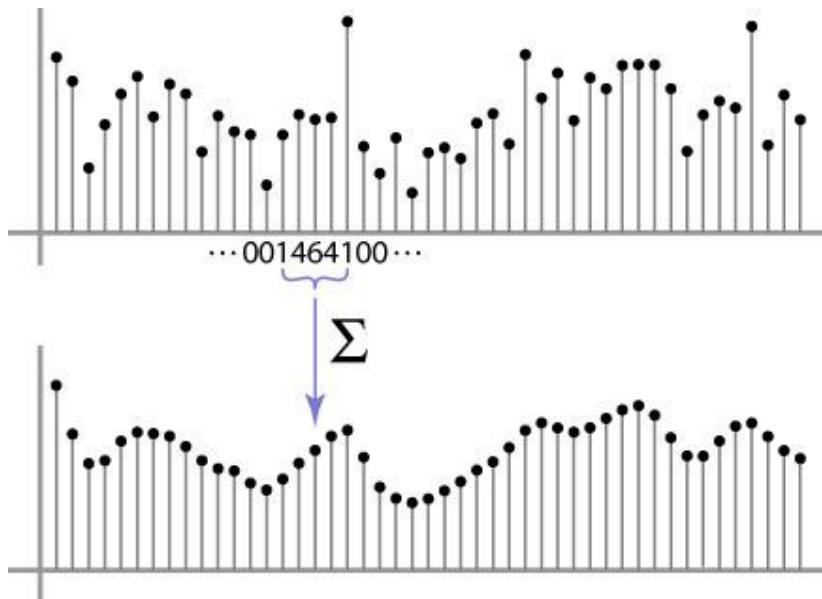
# Filters

- Sequence of weights  $a[j]$  is called a filter
- Filter is nonzero over its region of support
  - usually centered on zero: support radius  $r$
- Filter is normalized so that it sums to 1.0
  - this makes for a weighted average, not just any old weighted sum
- Most filters are symmetric about 0
  - since for images we usually want to treat left and right the same



# Convolution and filtering

- Convolution applies with any sequence of weights
- Example: bell curve (gaussian-like)  $[..., 1, 4, 6, 4, 1, ...]/16$



# Discrete filtering in 2D

- Same equation, one more index

$$(a \star b)[i, j] = \sum_{i', j'} a[i', j']b[i - i', j - j']$$

- now the filter is a rectangle you slide around over a grid of numbers
- Usefulness of associativity
- often apply several filters one after another:  $((a * b_1) * b_2) * b_3$
- this is equivalent to applying one filter:  $a * (b_1 * b_2 * b_3)$

# Moving Average In 2D

$$F[x, y]$$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$$G[x, y]$$


# Moving Average In 2D

$F[x, y]$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$G[x, y]$

0	10									

# Moving Average In 2D

$F[x, y]$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$G[x, y]$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

# Averaging Filter

- What values belong in the kernel H for the moving average example?

F[x, y]									
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

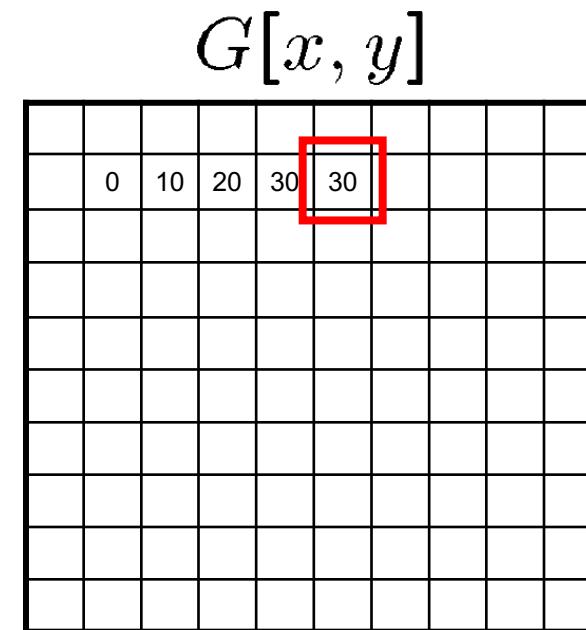
$\otimes$

H[u, v]

$\frac{1}{9}$

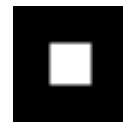
1	1	1
1	?	1
1	1	1

“box filter”



$$G = H \otimes F$$

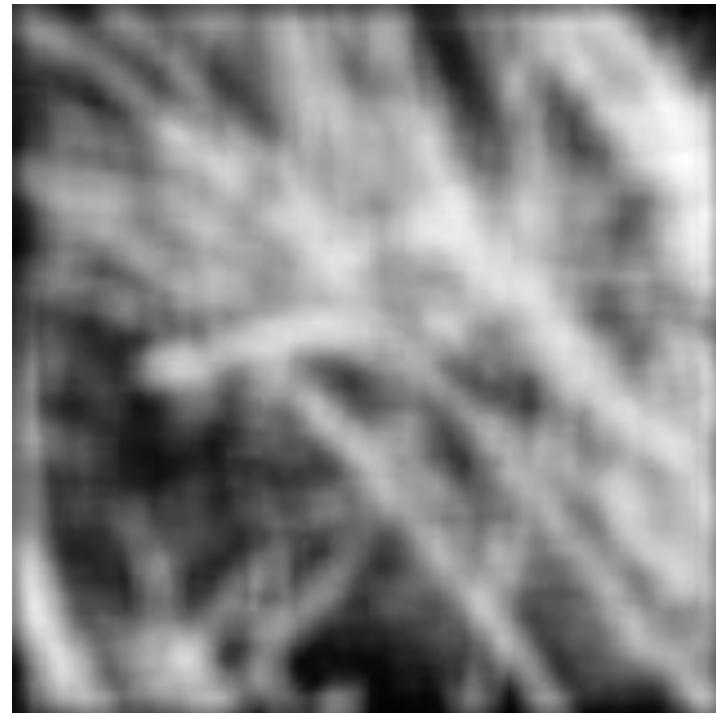
# Smoothing by averaging



depicts box filter:  
white = high value, black = low value



original



filtered

# Gaussian Filtering

# Gaussian Filter

- What if we want nearest neighboring pixels to have the most influence on the output?

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

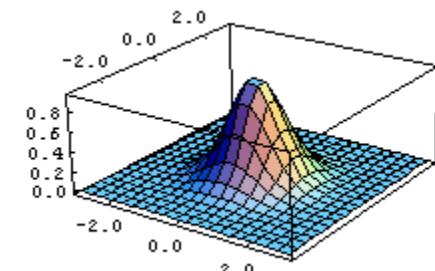
$F[x, y]$

$$\frac{1}{16} \begin{matrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{matrix}$$

$H[u, v]$

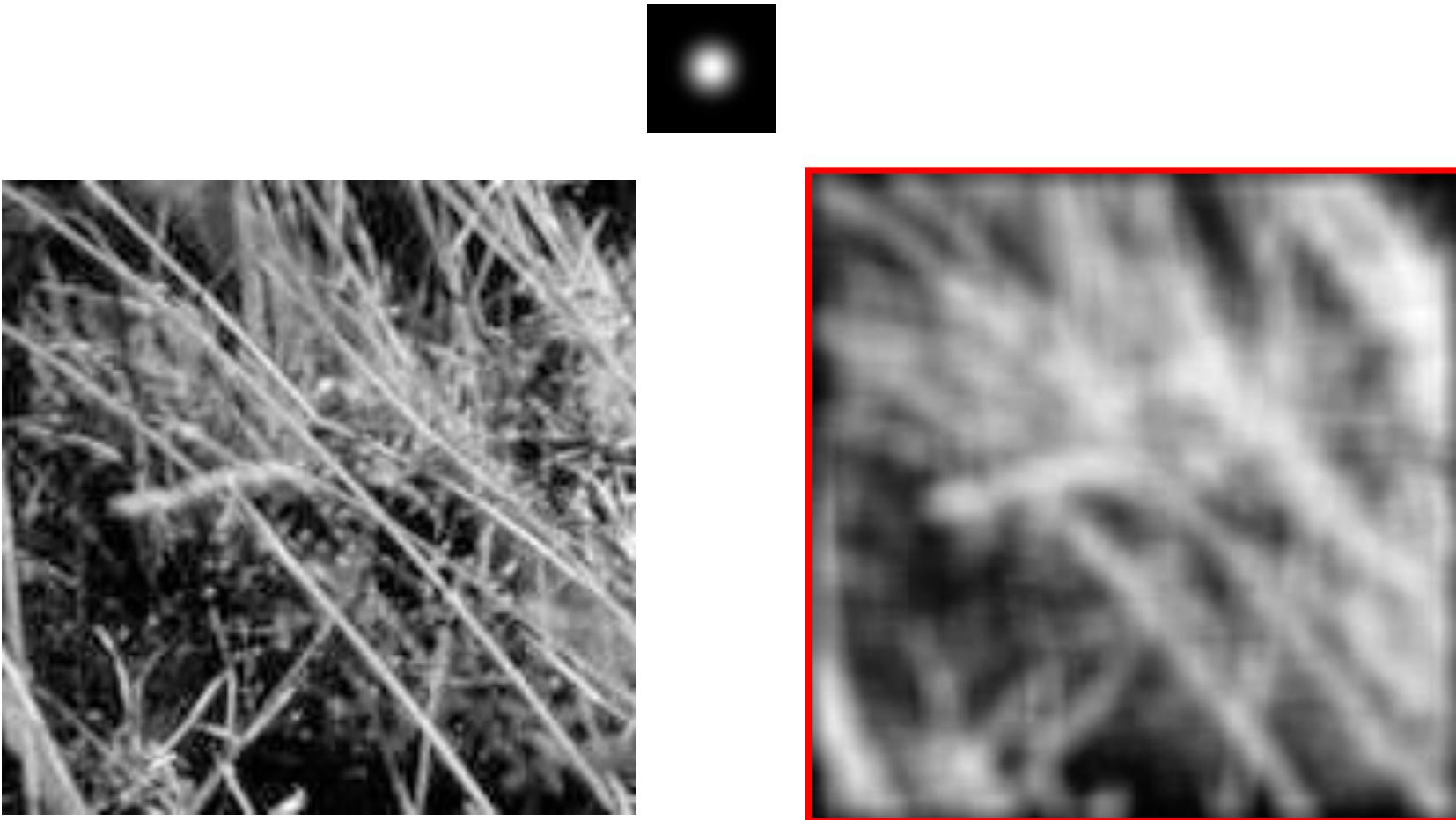
This kernel is an approximation of a 2d Gaussian function:

$$h(u, v) = \frac{1}{2\pi\sigma^2} e^{-\frac{u^2+v^2}{\sigma^2}}$$



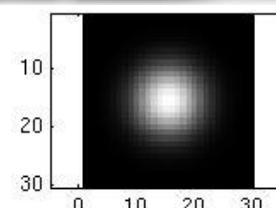
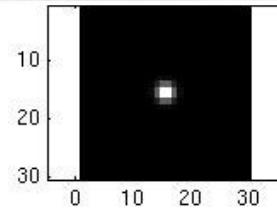
- Removes high-frequency components from the image ("low-pass filter").

# Smoothing with a Gaussian

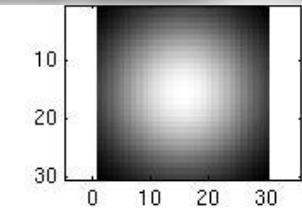


# Smoothing with a Gaussian

Parameter  $\sigma$  is the “scale” / “width” / “spread” of the Gaussian kernel, and controls the amount of smoothing.



...



# Strategy for Smoothing Images

- Images are not smooth because adjacent pixels are different.
- Smoothing = making adjacent pixels look more similar.
- Smoothing strategy  
pixel ~ average of its neighbors

# Sharpening

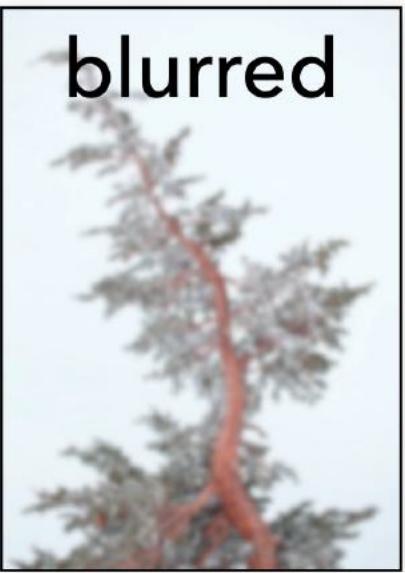
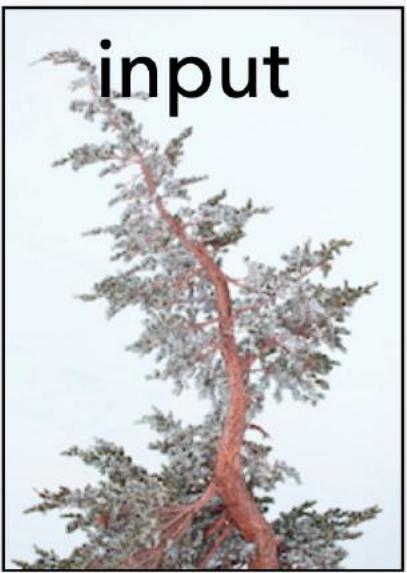
# How can we sharpen?

- Blurring was easy
- Sharpening is not as obvious

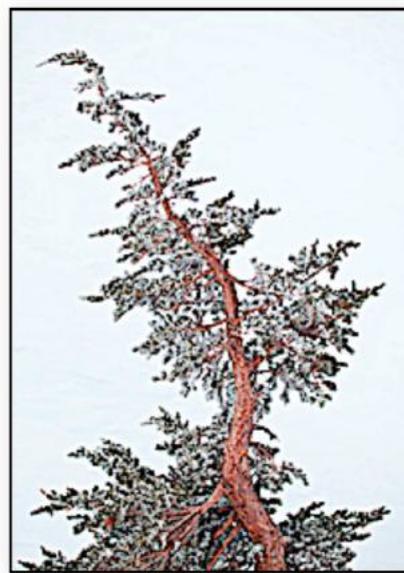
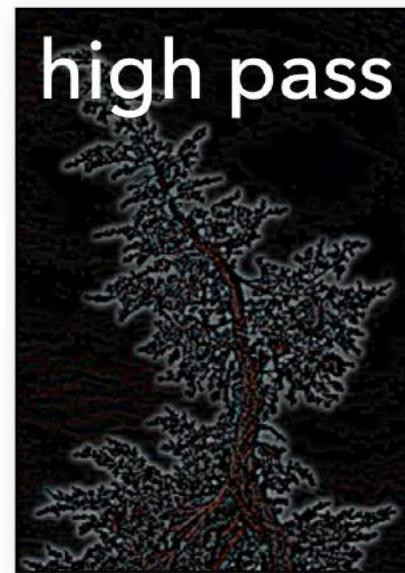
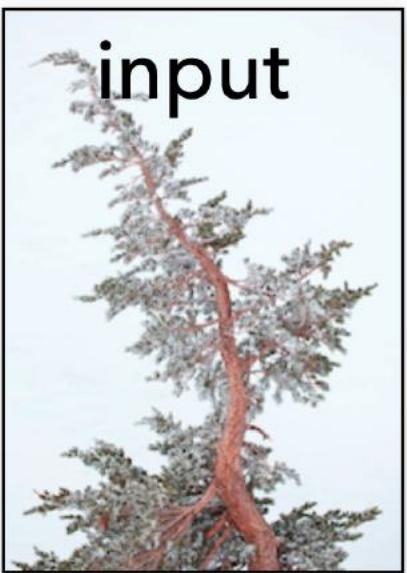
# How can we sharpen?

- Blurring was easy
- Sharpening is not as obvious
- Idea: amplify the stuff not in the blurry image
- $\text{output} = \text{input} + k * (\text{input}-\text{blur}(\text{input}))$

# Sharpening



high pass



sharpened  
image

# Sharpening: kernel view

- Recall

$$f' = f + k * (f - f \otimes g)$$

f is the input

f' is a sharpened image

g is a blurring kernel

k is a scalar controlling the strength of sharpening

# Sharpening: kernel view

- Recall

$$f' = f + k * (f - f \otimes g)$$

- Denote  $\delta$  the Dirac kernel (pure impulse)

$$f = f \otimes \delta$$

# Sharpening: kernel view

- Recall

$$f' = f + k * (f - f \otimes g)$$

$$f' = f \otimes \delta + k * (f \otimes \delta - f \otimes g)$$

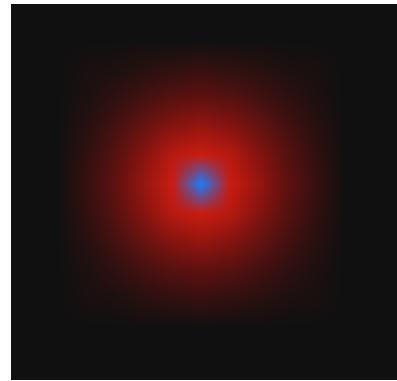
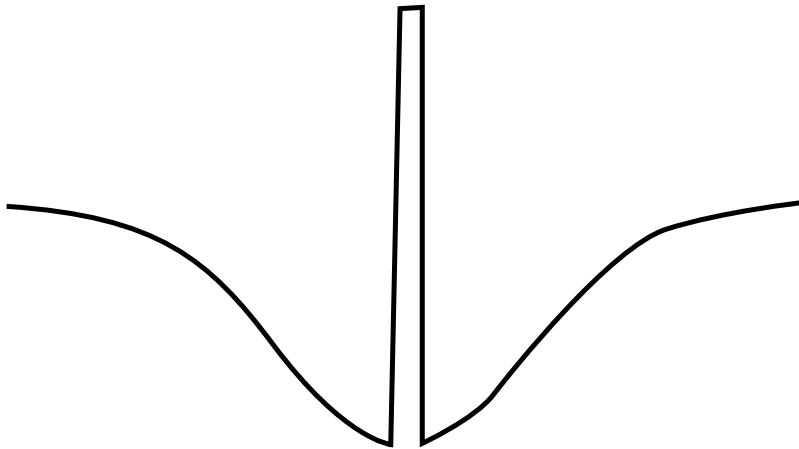
$$f' = f \otimes ((k+1)\delta - g)$$

- Sharpening is also a convolution

# Sharpening kernel

- Note: many other sharpening kernels exist  
(just like we saw multiple blurring kernels)
- Amplify the difference between a pixel and its neighbors

$$f' = f \otimes ((k + 1)\delta - g)$$



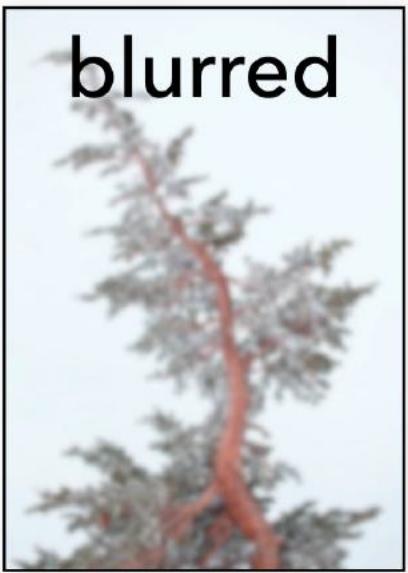
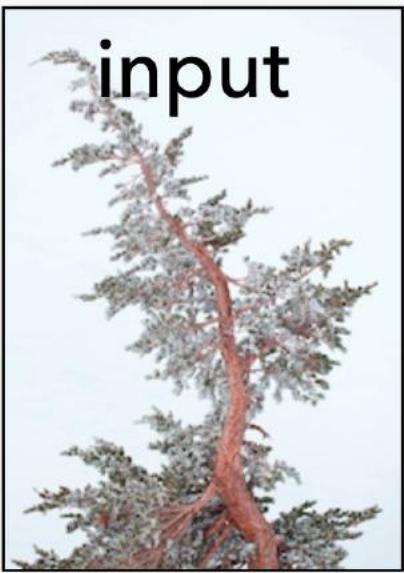
blue: positive  
red: negative

# Alternate interpretation

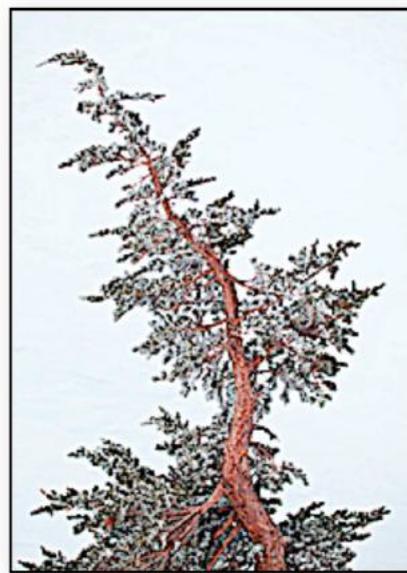
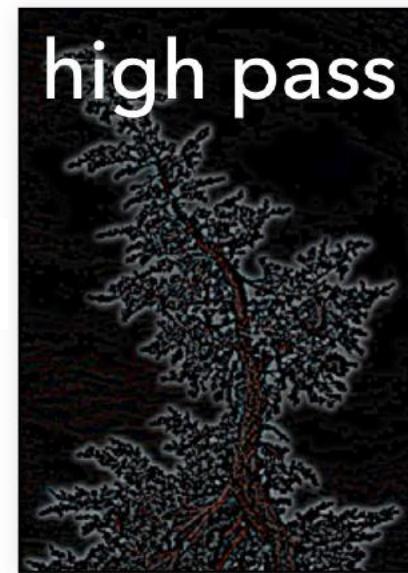
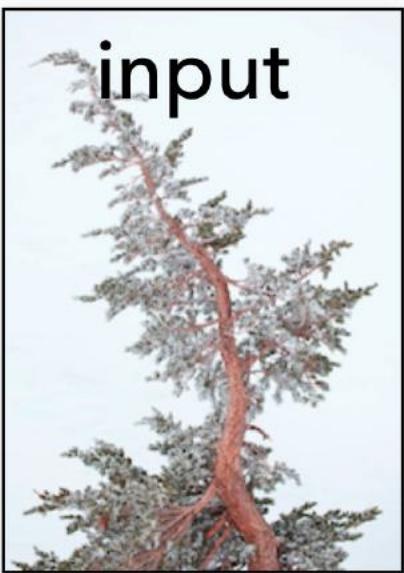
- $\text{out} = \text{input} + k * (\text{input} - \text{blur}(\text{input}))$
- $\text{out} = (1 + k) * \text{input} - k * \text{blur}(\text{input})$
- $\text{out} = \text{lerp}(\text{blur}(\text{input}), \text{input}, 1+k)$

linearly extrapolate from the blurred image “past” the original input image

# Sharpening



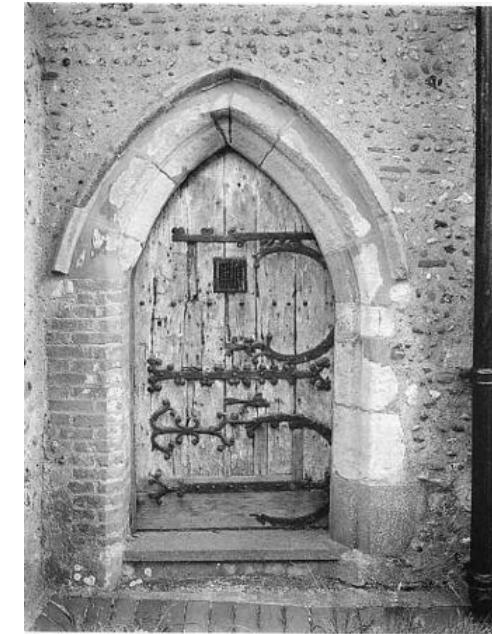
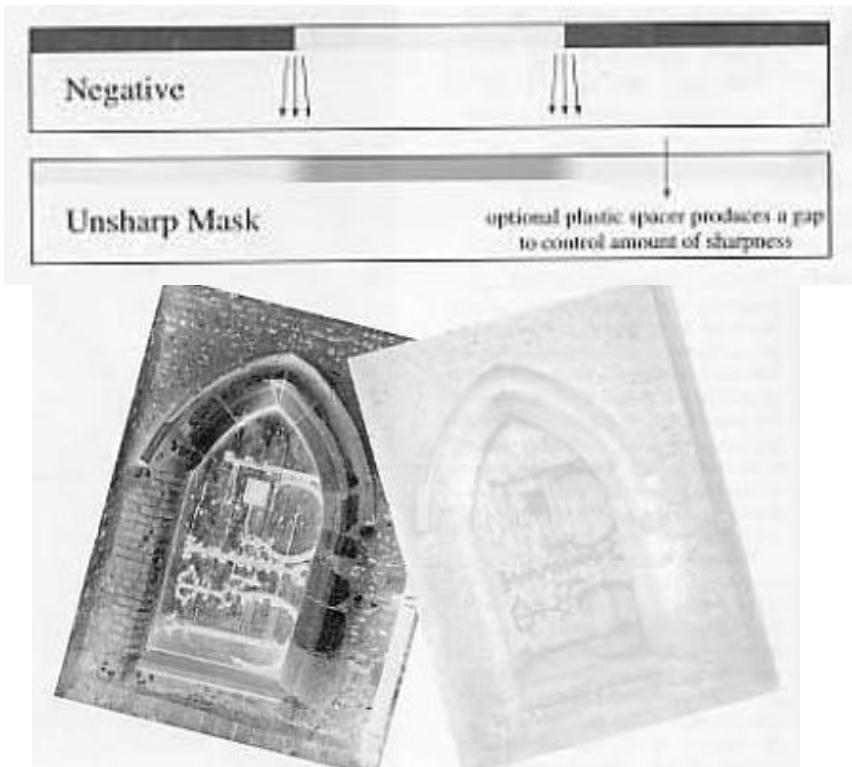
high pass



sharpened  
image

# Unsharp mask

- Sharpening is often called “unsharp mask” because photographers used to sandwich a negative with a blurry positive film in order to sharpen



# Unsharp mask

Fig.4: The two examples here show a detail of the brick-work to the left of the church door. The one on the left was printed with the negative alone – the one on the right was printed with both negative and mask as a sandwich. The increase in local contrast and edge sharpness is minute, but clearly visible. Grade 2.5 was used for the straight print but increased to 4.5 for the sandwiched image to compensate for the reduced contrast.



Fig.5: These two examples show a detail of the lower right hand side of the church door. Here the difference in sharpness is clearly visible between the (left) negative and (right) sandwich prints.



# Problem with excess

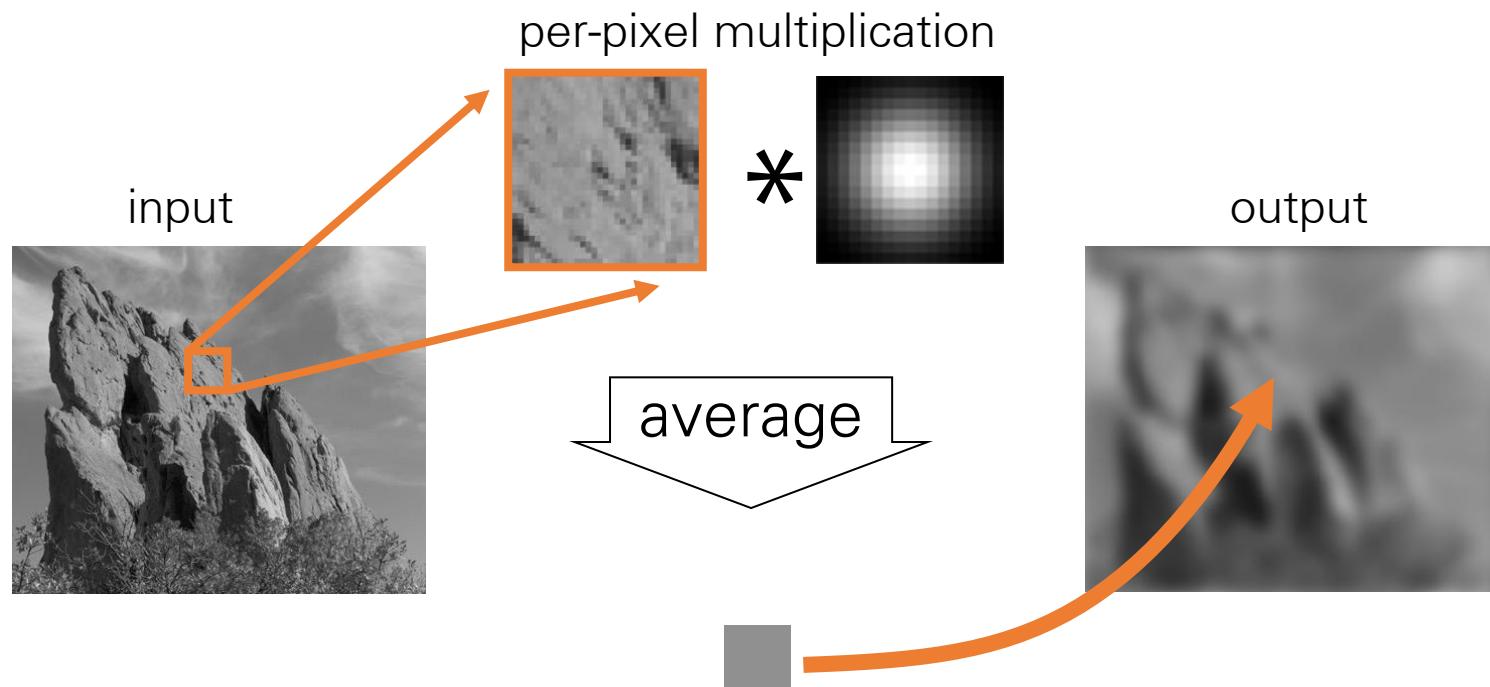
- Haloes around strong edges



# Bilateral Filter

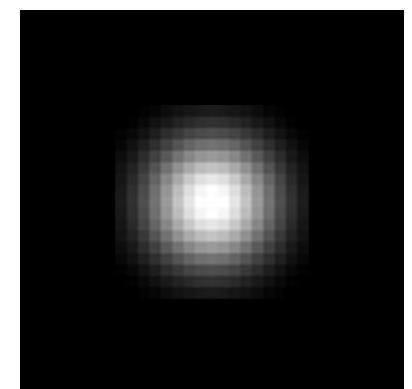
# Gaussian Filter

Idea: weighted average of pixels.



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

normalized  
Gaussian function



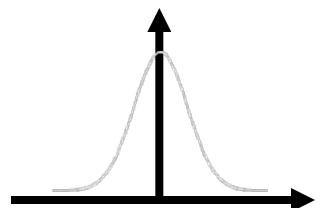
# Spatial Parameter



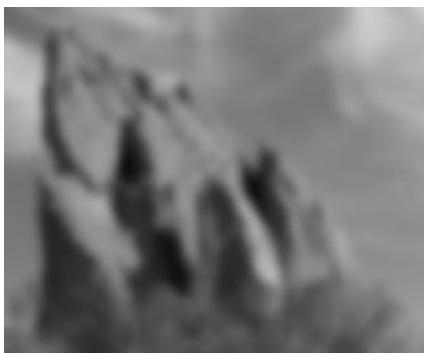
input

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

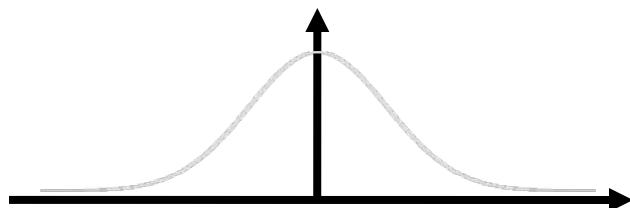
size of the window



small  $\sigma$



limited smoothing



large  $\sigma$

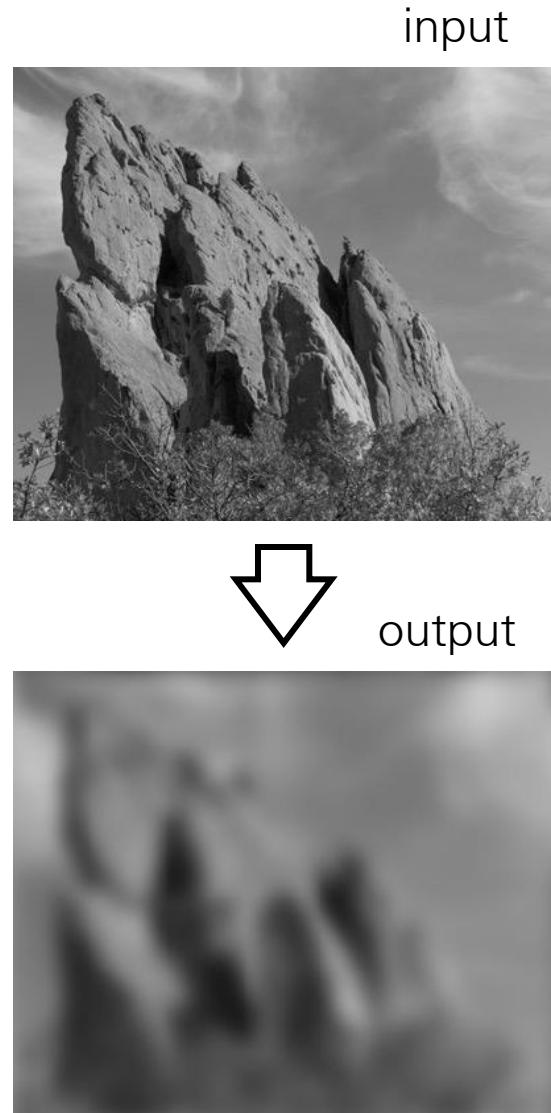


strong smoothing

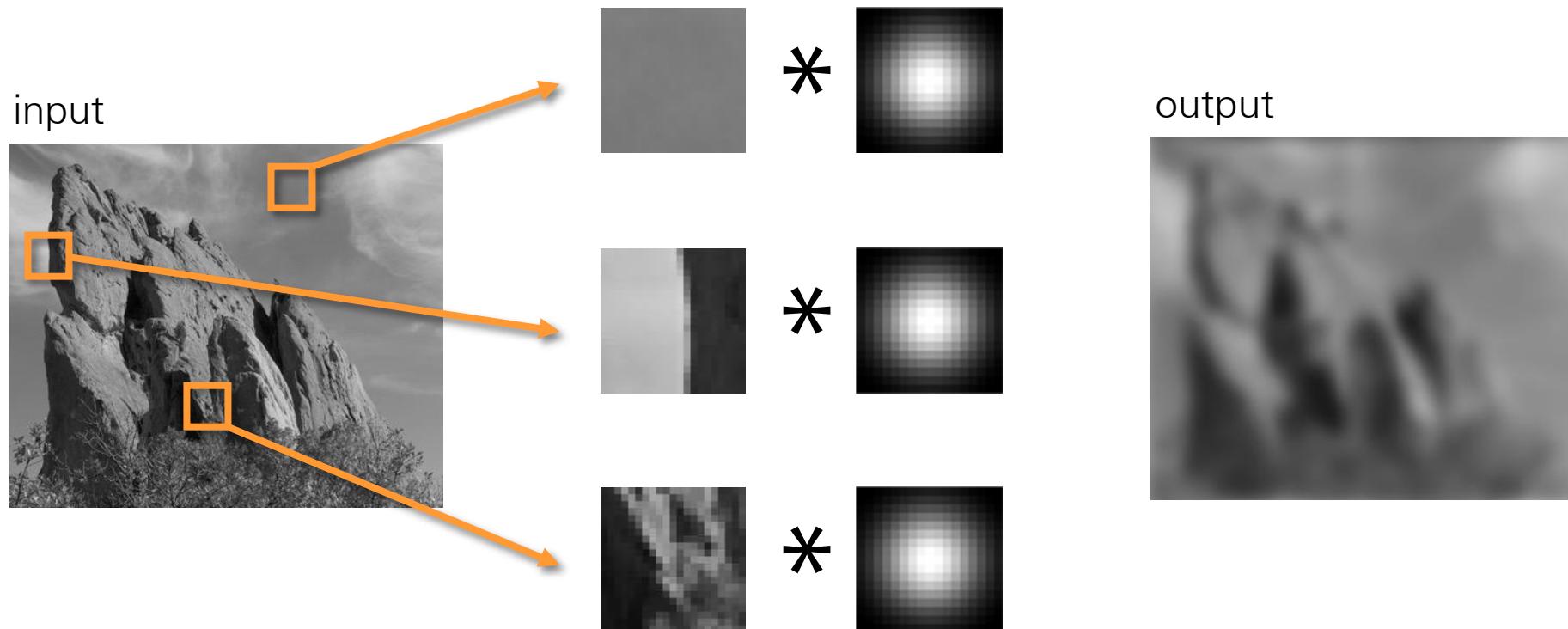
# Properties of Gaussian Blur

- Weights independent of spatial location
  - linear convolution
  - well-known operation
  - efficient computation (recursive algorithm, FFT...)
- Does smooth images
- But smoothes too much:  
edges are blurred.
  - Only spatial distance matters
  - No edge term

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

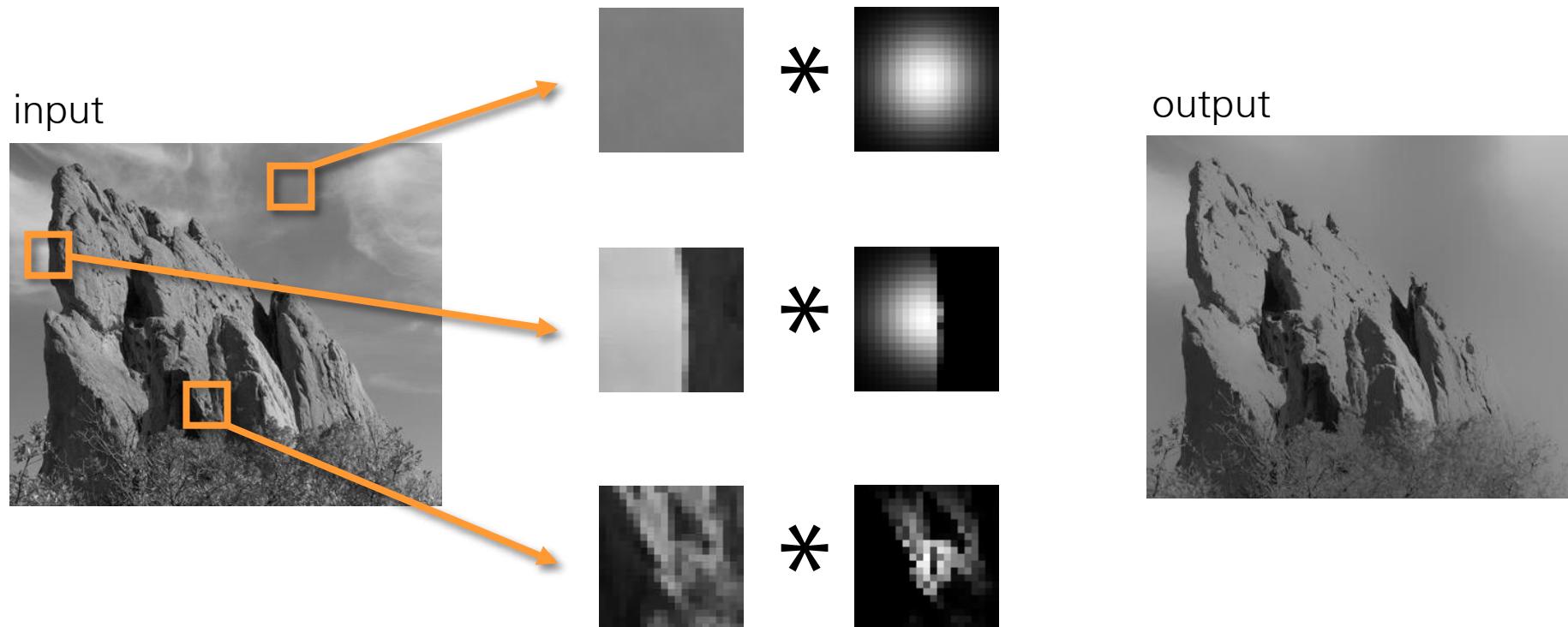


# Blur Comes from Averaging across Edges



Same Gaussian kernel everywhere.

# Bilateral Filter: No Averaging across Edges



The kernel shape depends on the image content.

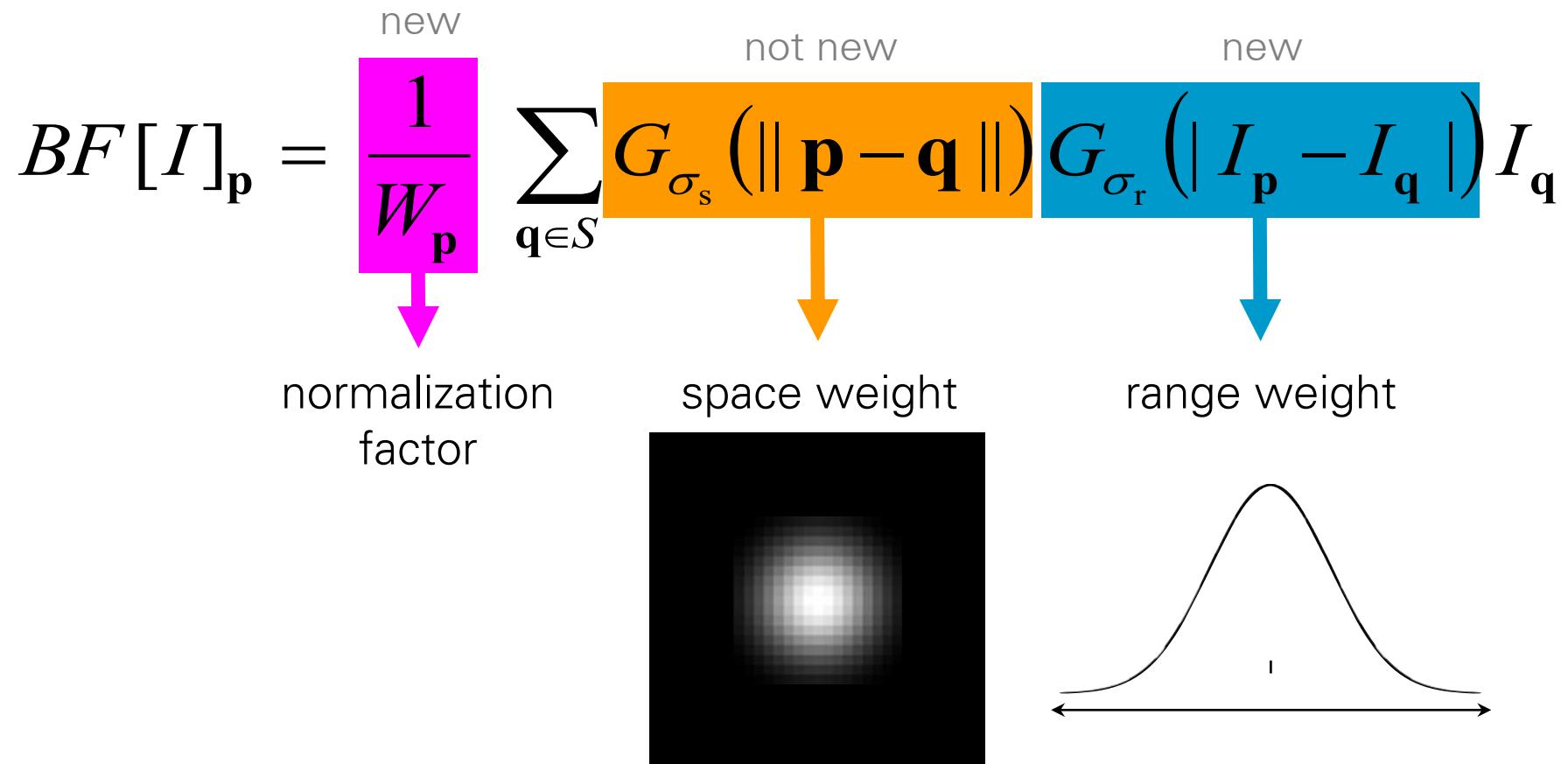
# Bilateral Filter: An Additional Edge Term

Same idea: weighted average of pixels.

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q$$

new  
not new  
new

normalization factor  
space weight  
range weight



# Bilateral Filter: An Additional Edge Term

Same idea: weighted average of pixels.

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q$$

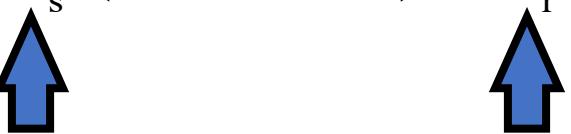
new  
not new  
new

normalization factor  
space weight  
range weight

favor nearby pixels  
favor similar pixels

The diagram illustrates the Bilateral Filter equation. It shows the formula  $BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q$  with three main components: a pink box labeled 'new' containing the normalization factor  $\frac{1}{W_p}$ , an orange box labeled 'not new' containing the space weight  $G_{\sigma_s}(\|p - q\|)$ , and a blue box labeled 'new' containing the range weight  $G_{\sigma_r}(|I_p - I_q|) I_q$ . Below these components are three arrows pointing downwards: a pink arrow from the normalization factor to the text 'normalization factor'; an orange arrow from the space weight to the text 'space weight'; and a blue arrow from the range weight to the text 'range weight'. At the bottom, there are two additional labels: 'favor nearby pixels' under the space weight and 'favor similar pixels' under the range weight. To the right of the range weight, there is a bell-shaped curve representing a Gaussian distribution, with a horizontal double-headed arrow indicating its width, labeled 'range weight'.

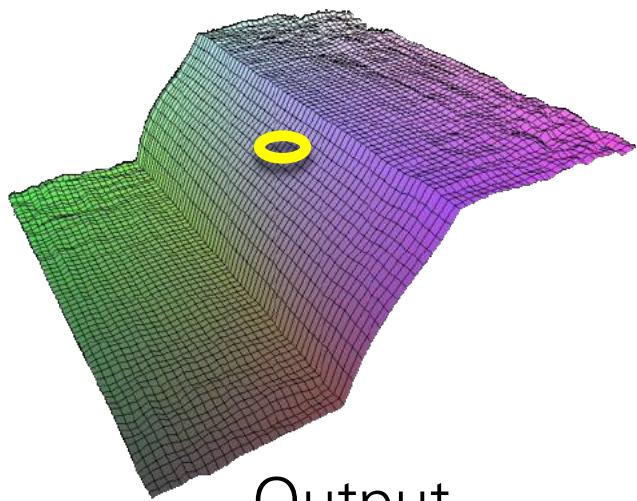
# Space and Range Parameters

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q$$


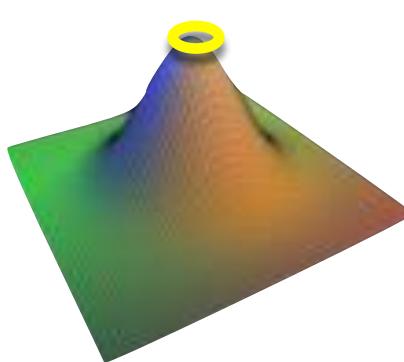
- space  $\sigma_s$  : spatial extent of the kernel, size of the considered neighborhood.
- range  $\sigma_r$  : “minimum” amplitude of an edge

# Gaussian filtering visualization

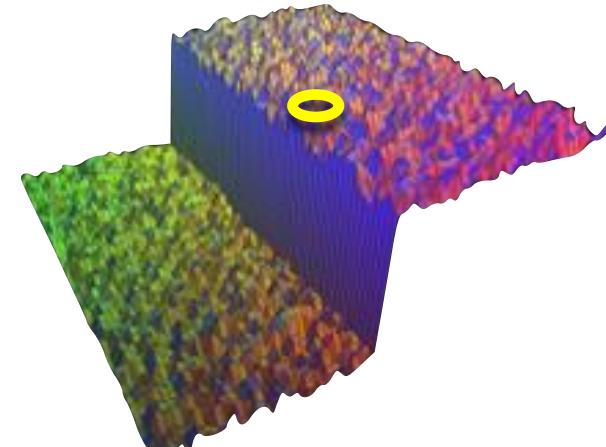
$$h[m, n] = \sum_{k,l} g[k, l] f[m + k, n + l]$$



Output

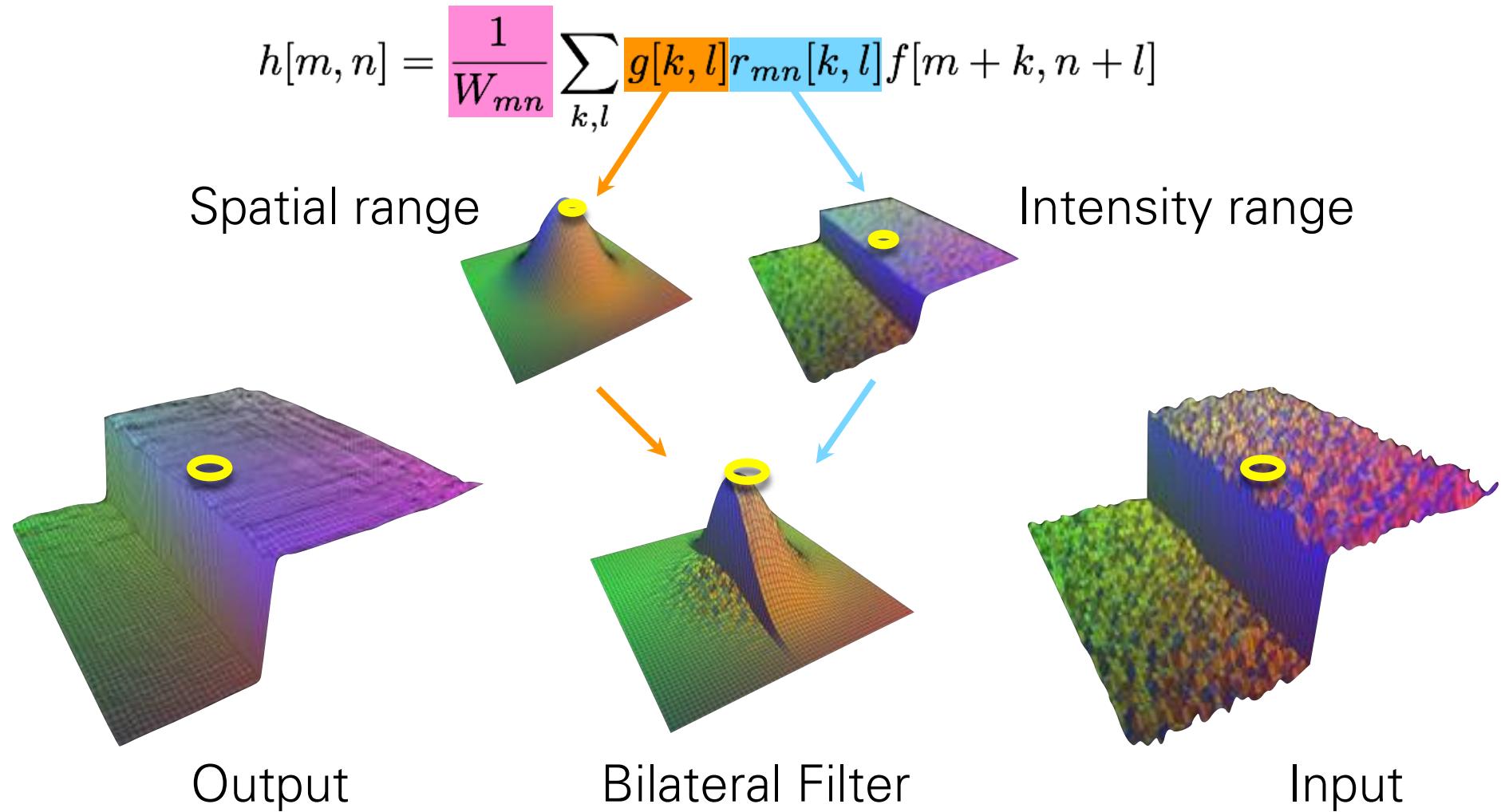


Gaussian Filter



Input

# Bilateral filtering visualization



# Exploring the Parameter Space



input

$\sigma_s = 2$



$\sigma_r = 0.1$

$\sigma_s = 6$

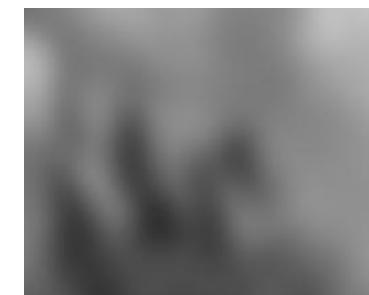


$\sigma_s = 18$

$\sigma_r = 0.25$



$\sigma_r = \infty$   
(Gaussian blur)



# Bilateral Filtering Color Images

For gray-level images

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q$$

intensity difference  
scalar



For color images

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(\|\mathbf{C}_p - \mathbf{C}_q\|) \mathbf{C}_q$$

color difference  
3D vector  
(RGB, Lab)

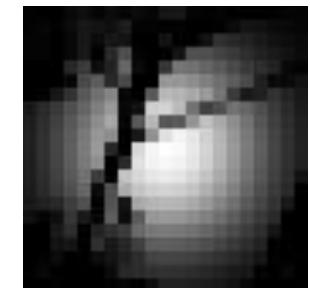
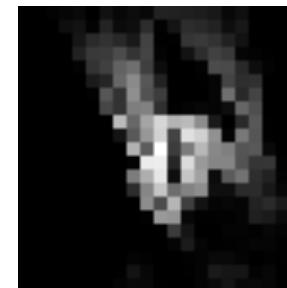
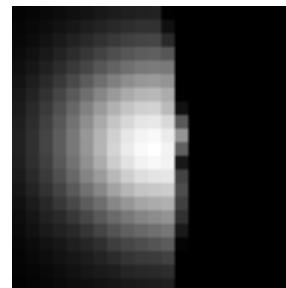
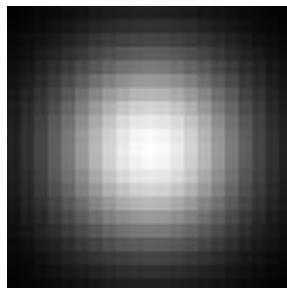


# Hard to Compute

- Nonlinear

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q$$

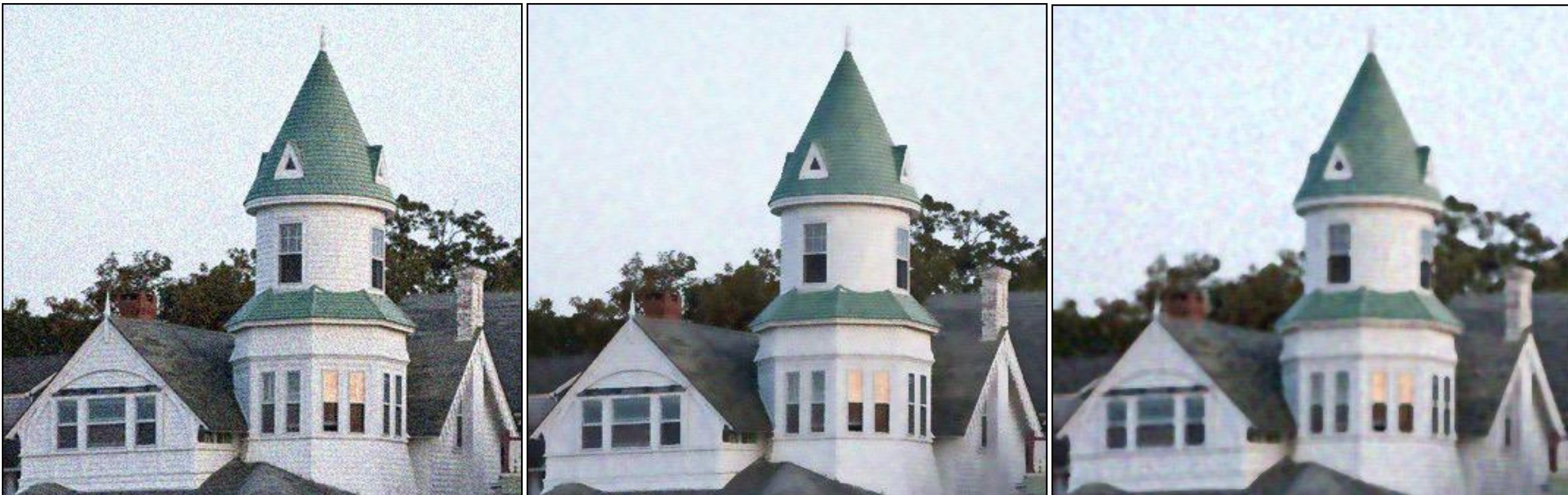
- Complex, spatially varying kernels
- Cannot be precomputed, no FFT...



- Brute-force implementation is slow > 10min

Additional Reading: S. Paris and F. Durand, A Fast Approximation of the Bilateral Filter using a Signal Processing Approach, In Proc. ECCV, 2006

# Denoising



noisy input

bilateral filtering

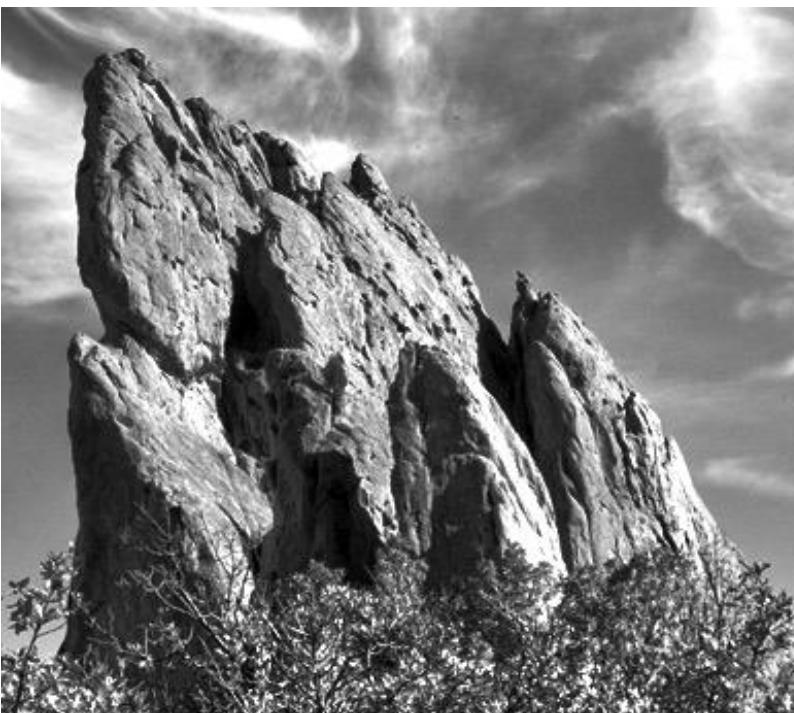
median filtering

# Contrast enhancement

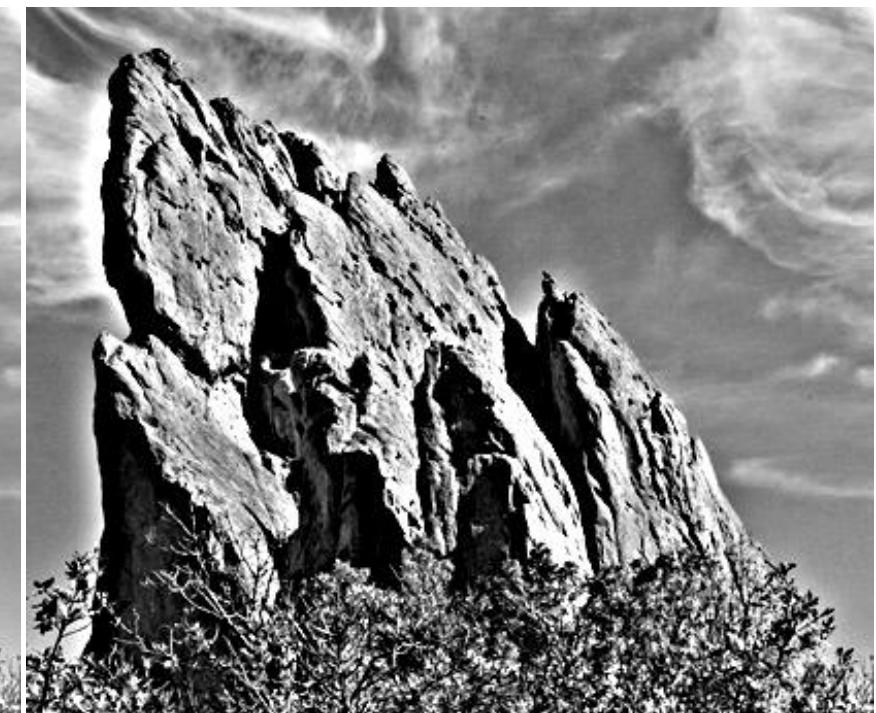
How would you use bilateral filtering for sharpening?



input



sharpening based on  
bilateral filtering



sharpening based on  
Gaussian filtering

# Photo retouching



# Photo retouching



original



digital pore removal (aka bilateral filtering)

# Before



downloaded from pickywallpapers.com

# After



# Close-up comparison



original



digital pore removal (aka bilateral filtering)

# Cartoonization

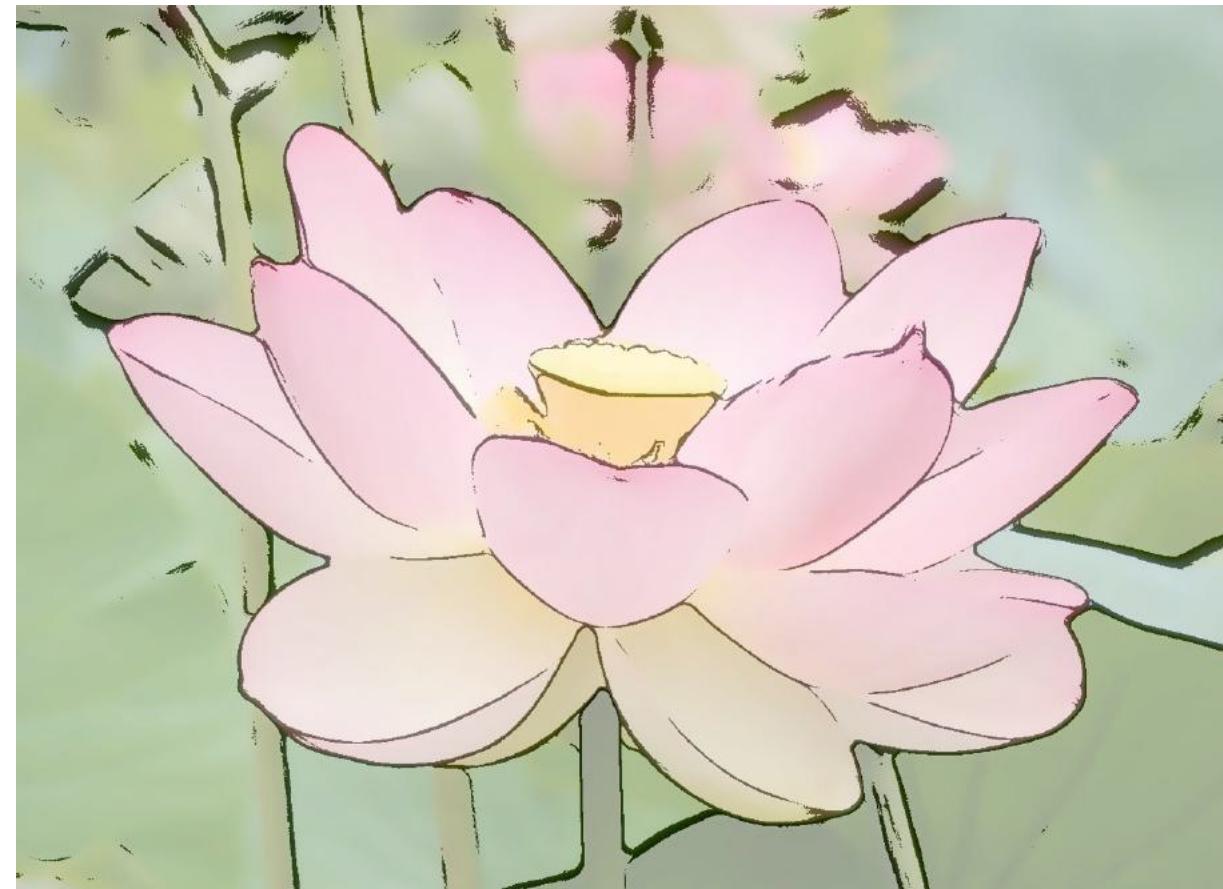


input



cartoon rendition

# Cartoonization



How would you create this effect?

# Cartoonization



edges from bilaterally filtered image



+



bilaterally filtered image

cartoon rendition



=



Note: image cartoonization and abstraction are very active research areas.

# Is the bilateral filter:

Linear?

Shift-invariant?

# Is the bilateral filter:

Linear?

- No.

Shift-invariant?

- No.

Does this have any bad implications?

# The bilateral grid

## Real-time Edge-Aware Image Processing with the Bilateral Grid

Jiawen Chen

Sylvain Paris

Frédo Durand

Computer Science and Artificial Intelligence Laboratory  
Massachusetts Institute of Technology



**Figure 1:** The bilateral grid enables edge-aware image manipulations such as local tone mapping on high resolution images in real time. This 15 megapixel HDR panorama was tone mapped and locally refined using an edge-aware brush at 50 Hz. The inset shows the original input. The process used about 1 MB of texture memory.

Data structure for fast  
edge-aware image  
processing.

# Flash/no-flash photography via bilateral filtering



Red Eye



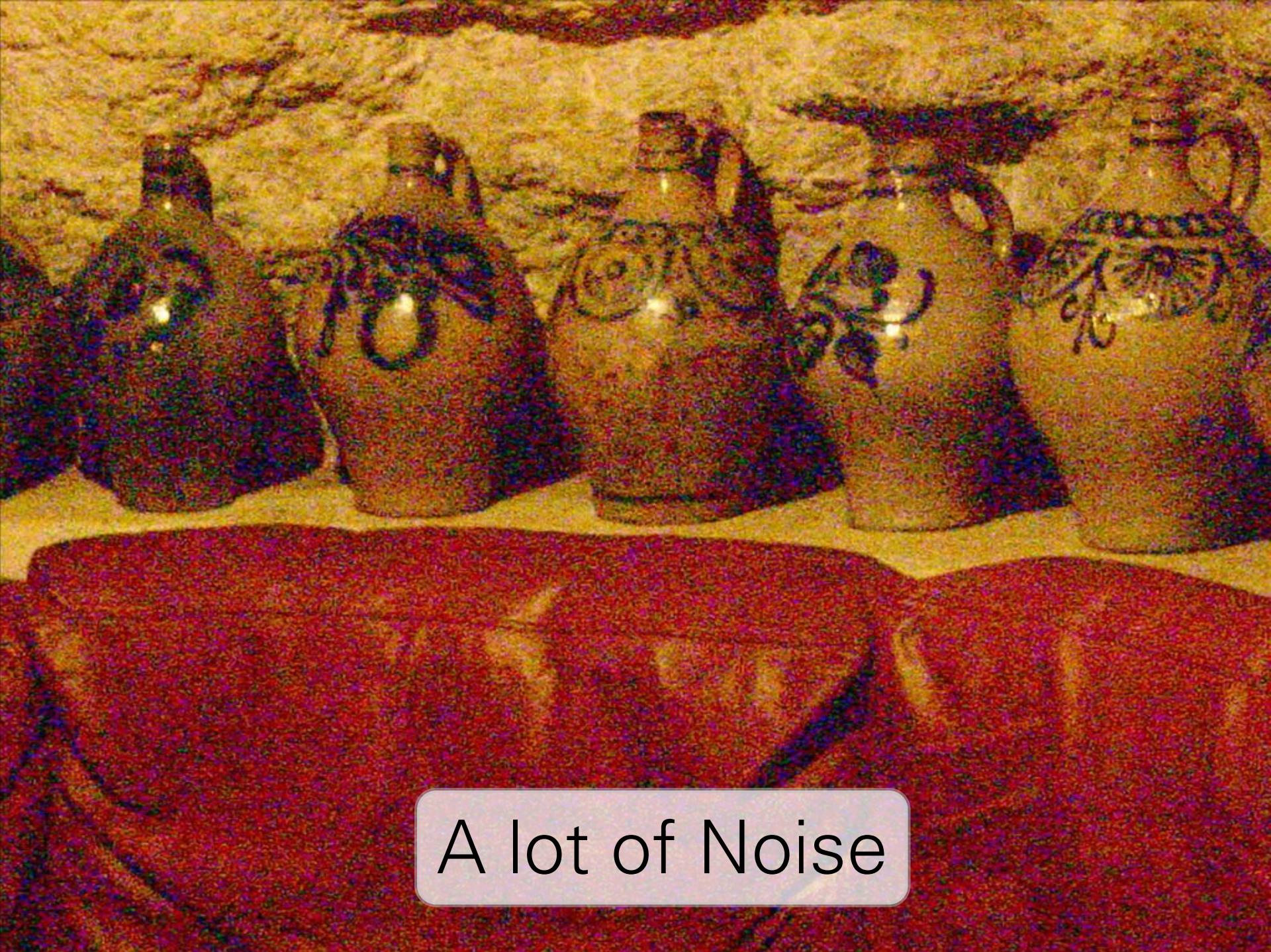
Unflattering Lighting



Motion Blur



Noise

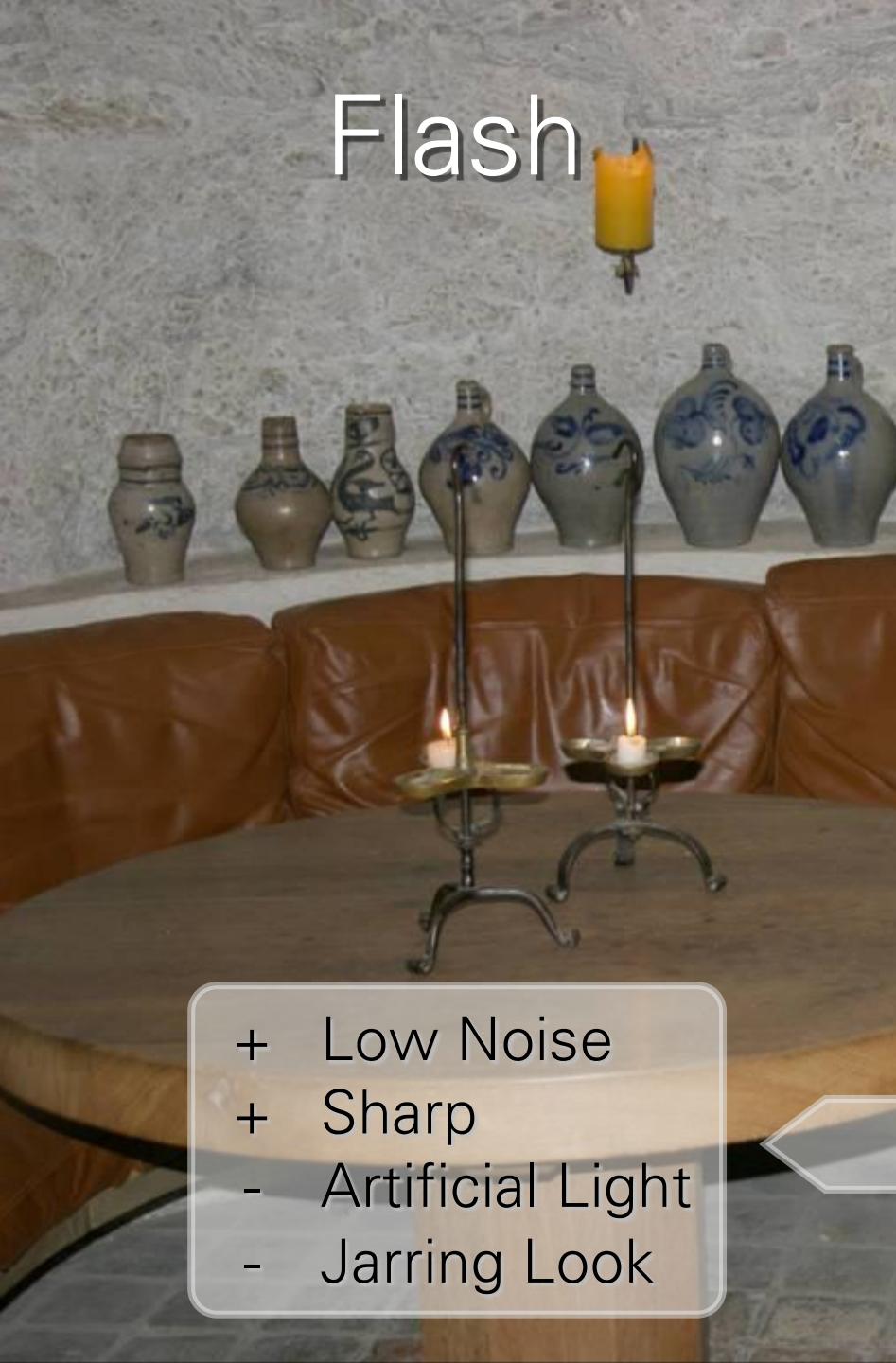


A lot of Noise



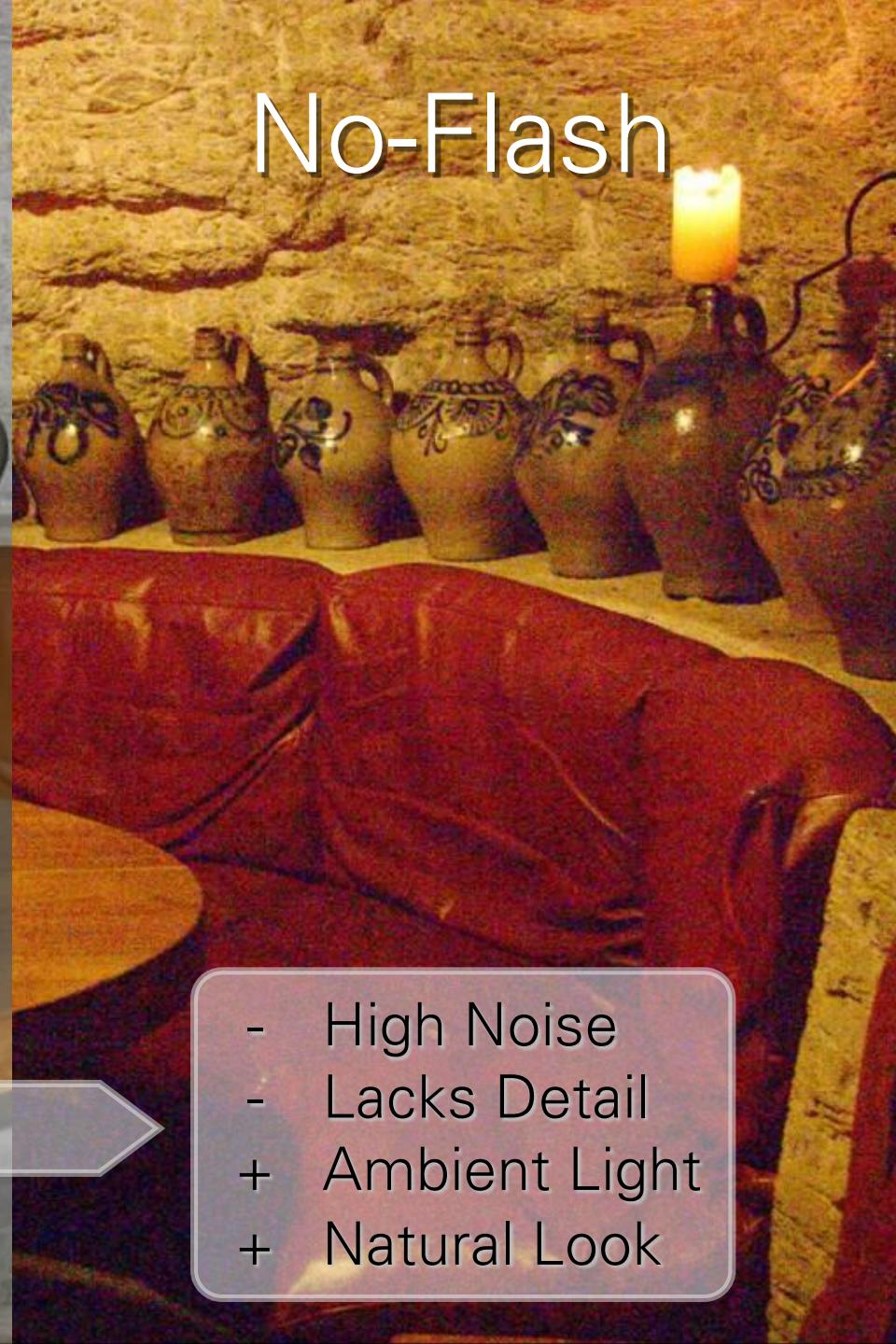
Ruined Ambiance

# Flash



- + Low Noise
- + Sharp
- Artificial Light
- Jarring Look

# No-Flash

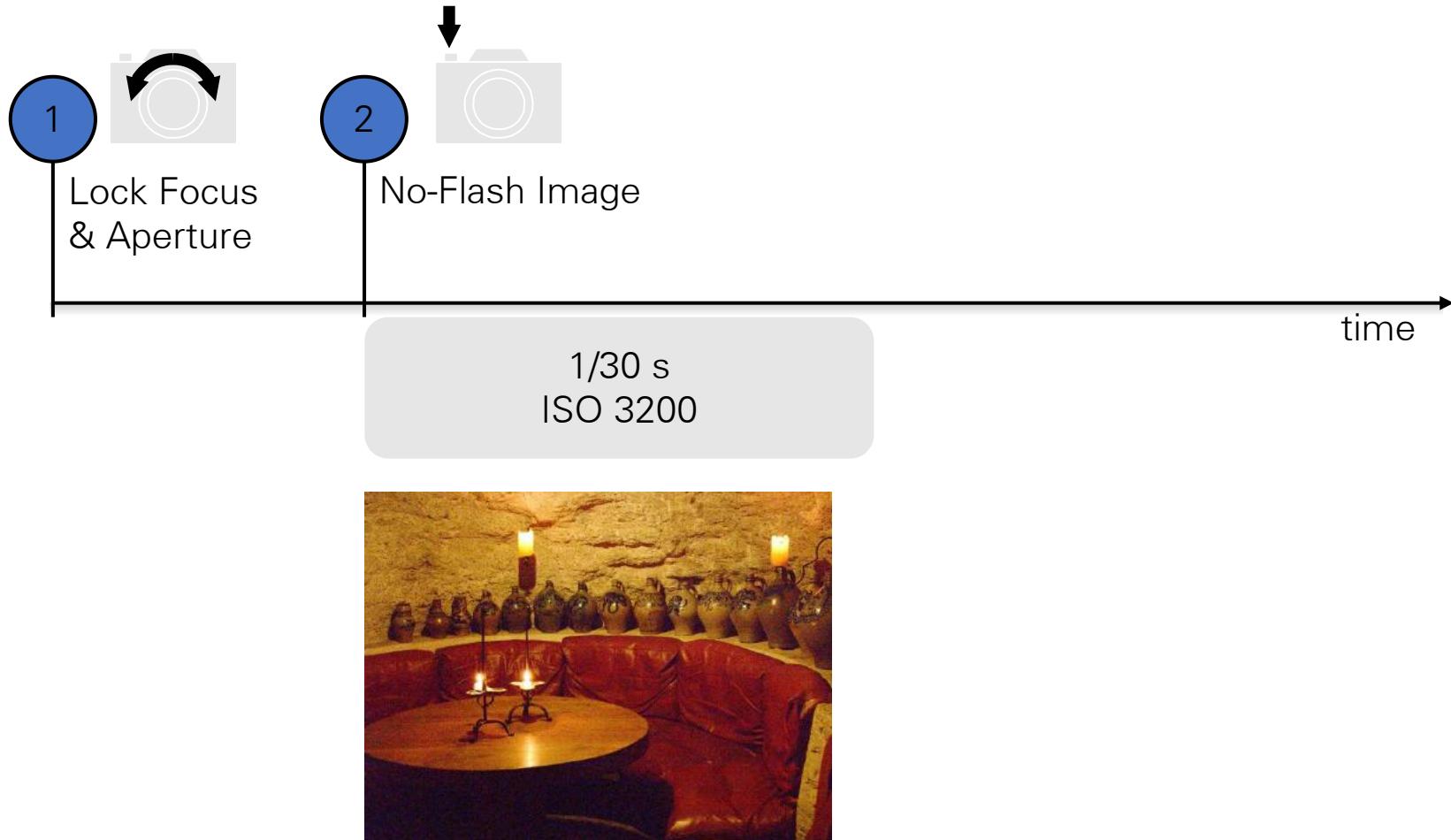


- High Noise
- Lacks Detail
- + Ambient Light
- + Natural Look

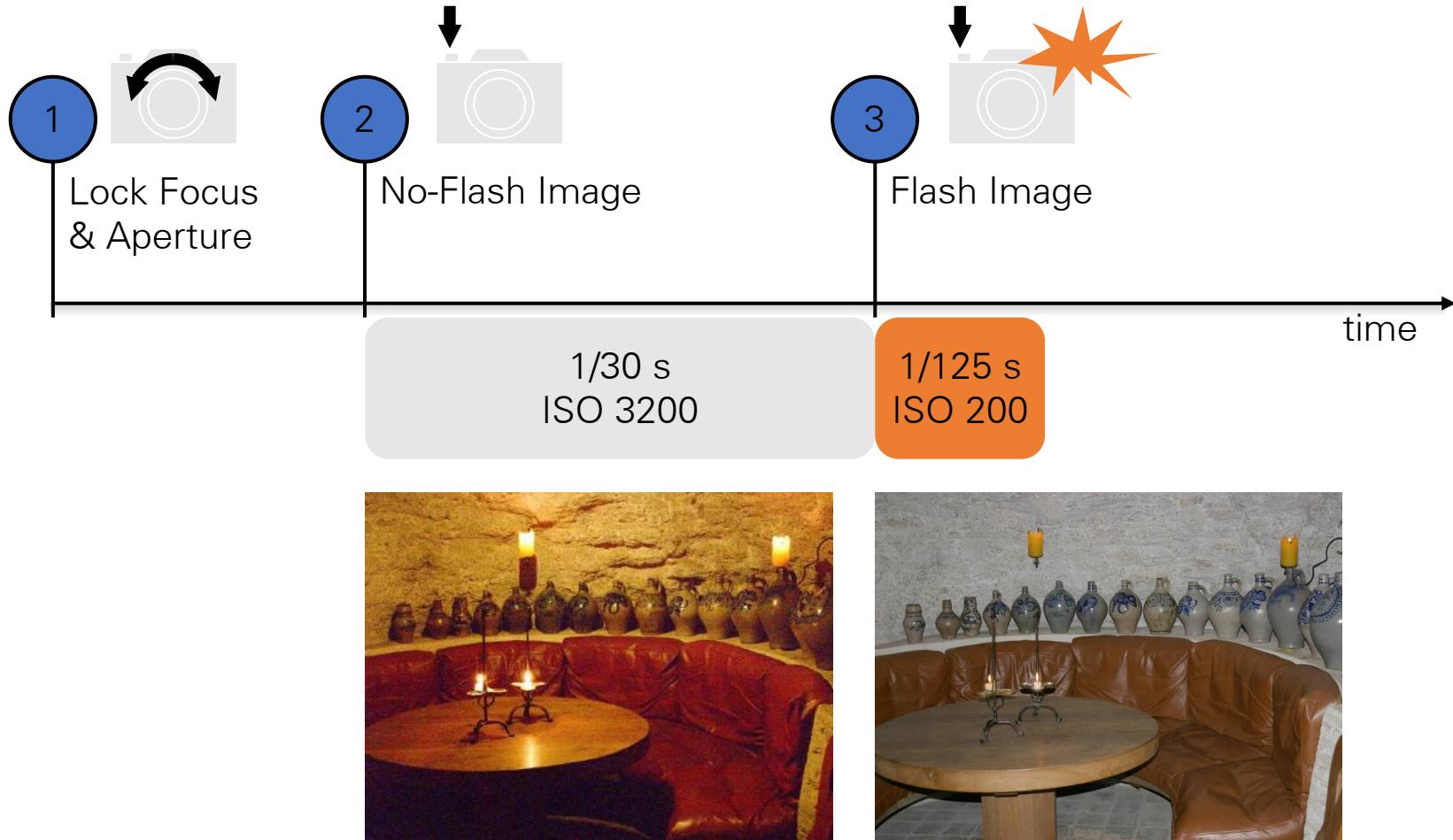
# Image acquisition



# Image acquisition



# Image acquisition





Denoising Result



No-Flash



Denoising Result

# Key idea

Denoise the no-flash image while maintaining the edge structure of the flash image

- How would you do this using the image editing techniques we've learned about?

# Joint bilateral filtering

# Denoising with bilateral filtering



noisy input

bilateral filtering

median filtering

# Denoising with bilateral filtering

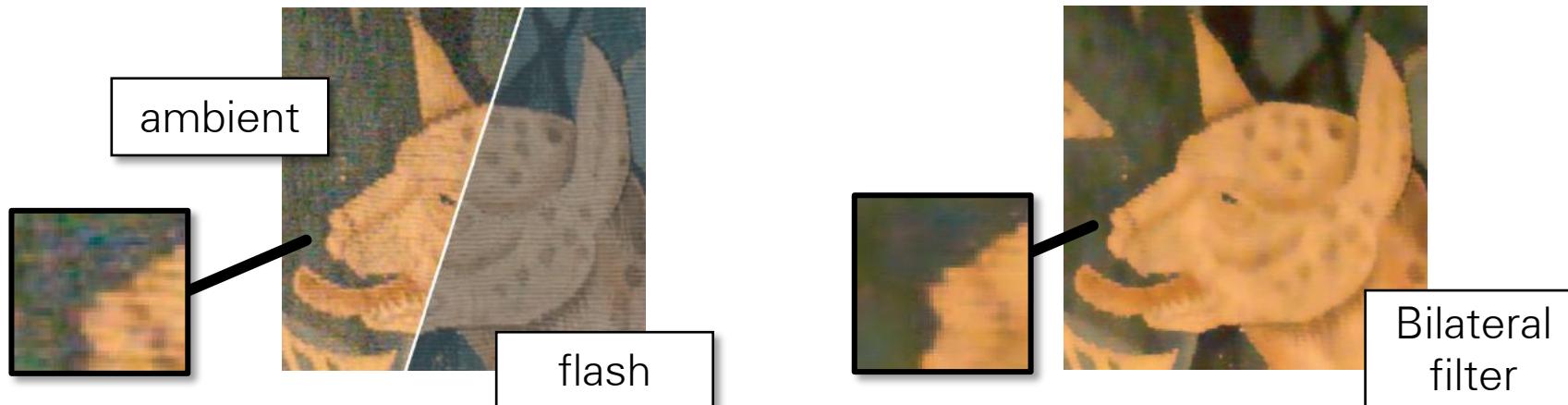
$$A_{p(col)}^{Base} = \frac{1}{k(p(col))} \sum_{p' \in \Omega} g_d(|p - p'|)$$

spatial kernel

$$g_r(A_{p(col)} - A_{p'(col)}) A_{p'(col)}$$

intensity kernel

- However, results still have noise or blur (or both)



# Denoising with joint bilateral filtering

$$A_{p(col)}^{NR} = \frac{1}{k(p(col))} \sum_{p' \in \Omega} g_d(|p - p'|) \\ g_r(F_{p(col)} - F_{p'(col)}) A_{p'(col)}$$

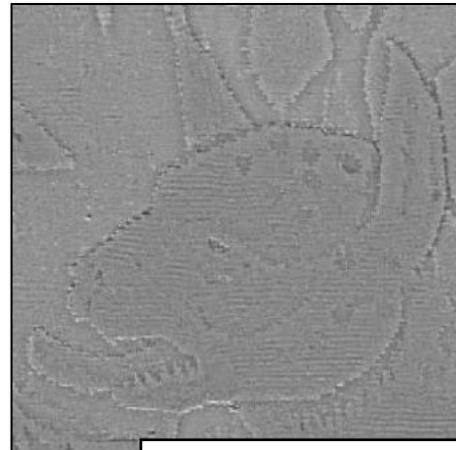
- In the flash image there are many more details
- Use the flash image F to find edges

# Denoising with joint bilateral filtering

$$A_{p(col)}^{NR} = \frac{1}{k(p(col))} \sum_{p' \in \Omega} g_d(|p - p'|) \\ g_r(F_{p(col)} - F_{p'(col)}) A_{p'(col)}$$



Bilateral  
filter



The difference

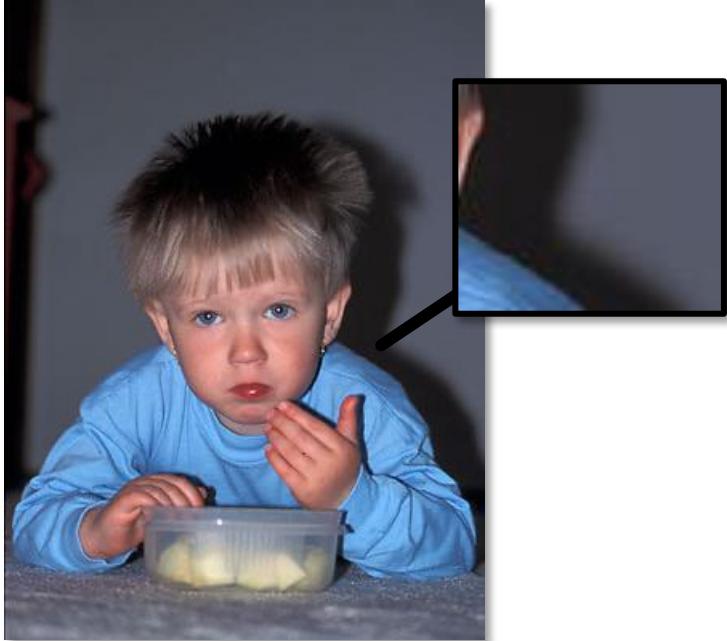


Joint Bilateral  
filter

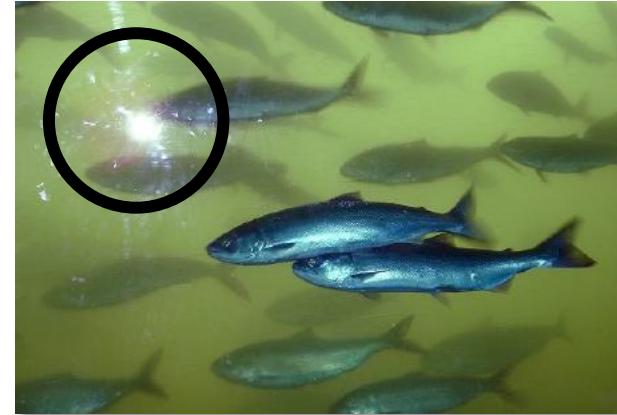
# Not all edges in the flash image are real

Can you think of any types of edges that may exist in the flash image but not the ambient one?

# Not all edges in the flash image are real



shadows



specularities

- May cause over- or under-blur in joint bilateral filter
- We need to eliminate their effect

# Detecting shadows

- Observation: the pixels in the flash shadow should be similar to the ambient image.
- Not identical:
  1. Noise.
  2. Inter-reflected flash.
- Compute a shadow mask.
- Take pixel  $p$  if  $F_{p(col)}^{Lin} - A_{p(col)}^{Lin} \leq \tau_{Shadow}$
- $\tau_{Shadow}$  is manually adjusted
- Mask is smoothed and dilated

# Detecting specularities

- Take pixels where sensor input is close to maximum (very bright).
  - Over fixed threshold  $\tau_{Spec}$
- Create a specularity mask.
- Also smoothed.
- M – the combination of shadow and specularity masks:

Where  $M_p=1$ , we use  $A^{Base}$ . For other pixels we use  $A^{NR}$ .

# Detail transfer

- Denoising cannot add details missing in the ambient image
- Exist in flash image because of high SNR
- We use a quotient image:

$$F_{p(col)}^{Detail} = \frac{F_{p(col)} + \varepsilon}{F_{p(col)}^{Base} + \varepsilon}$$

Reduces the effect of noise in F

- Multiply with  $A^{NR}$  to add the details
- Masked in the same way

Bilateral filtered

Why does this quotient image make sense for detail?

# Detail transfer

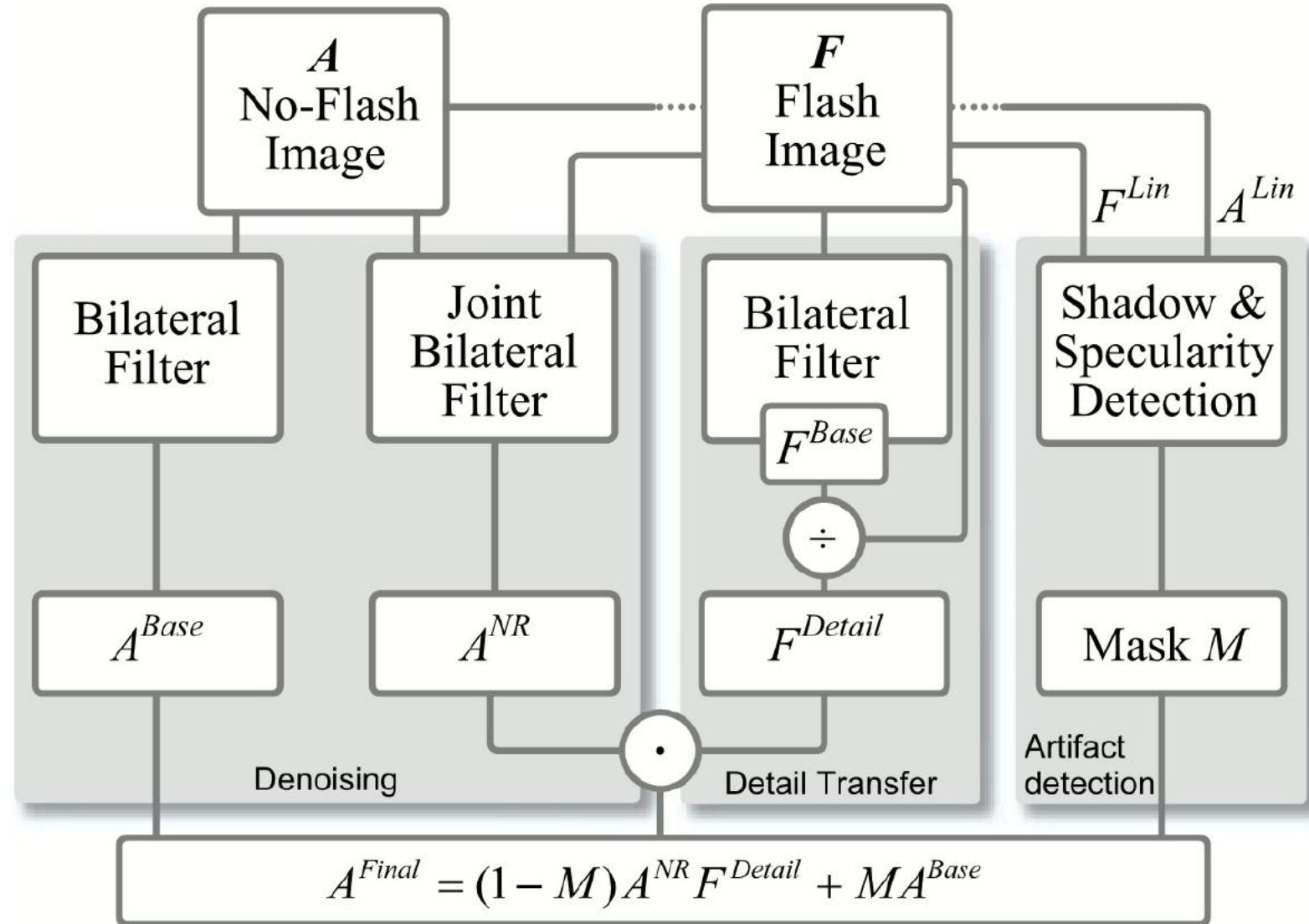
- Denoising cannot add details missing in the ambient image
- Exist in flash image because of high SNR
- We use a quotient image:

$$F_{p(col)}^{Detail} = \frac{F_{p(col)} + \varepsilon}{F_{p(col)}^{Base} + \varepsilon}$$

Reduces the  
effect of  
noise in F



# Full pipeline



# Demonstration



ambient-only



joint bilateral and detail transfer



Flash



No-Flash



No-Flash

Result





Flash



No-Flash





Result



Flash



No-Flash



Flash



No-Flash

Result



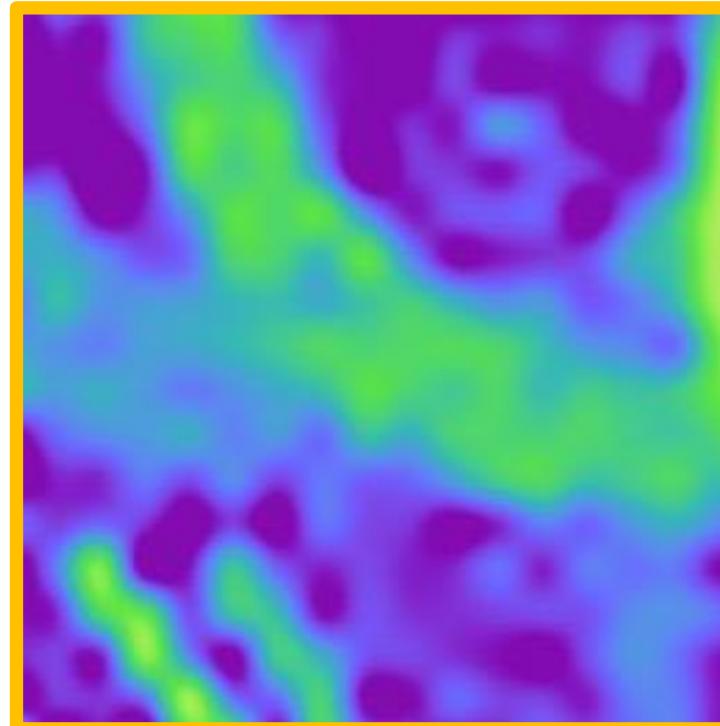
# Joint bilateral filtering other applications

# Edge-aware depth denoising

$$A_{p(col)} = \frac{1}{k(p(col))} \sum_{p' \in \Omega} g_d(|p - p'|) \\ g_r(F_{p(col)} - F_{p'(col)}) A_{p'(col)}$$

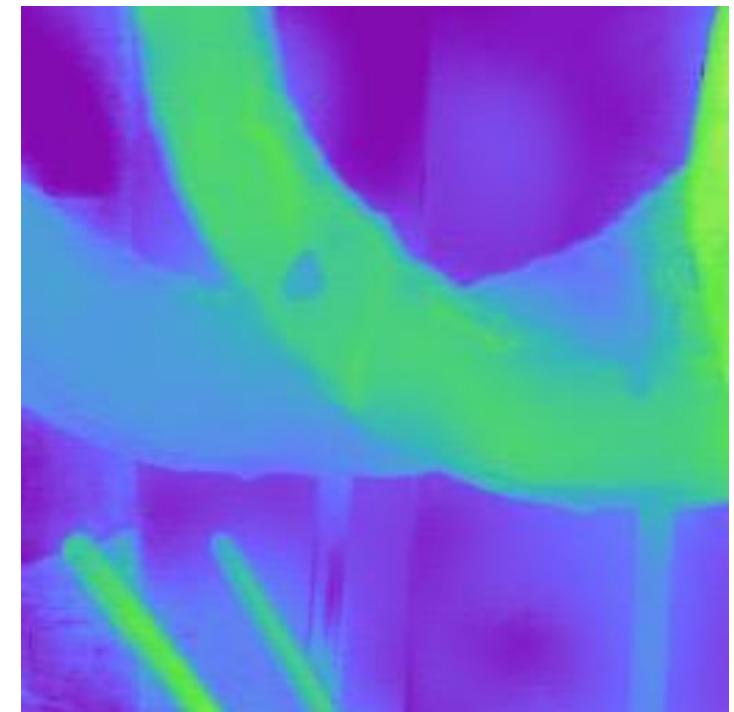


One of two input images



Depth from disparity

Use joint bilateral filtering, with the input image as guide.



Guided filtering

# Other applications of joint bilateral filtering

## Deep Bilateral Learning for Real-Time Image Enhancement

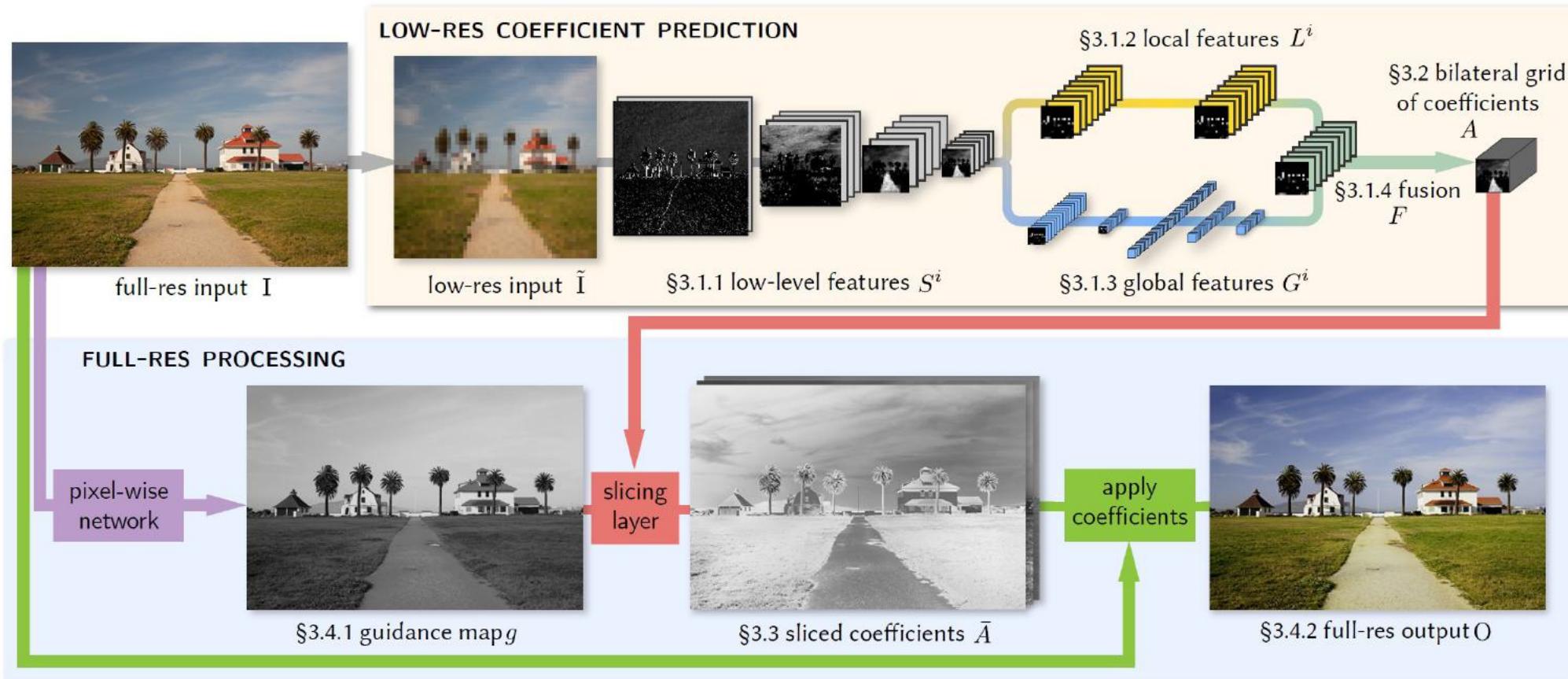
MICHAËL GHARBI, MIT CSAIL

JIAWEN CHEN, Google Research

JONATHAN T. BARRON, Google Research

SAMUEL W. HASINOFF, Google Research

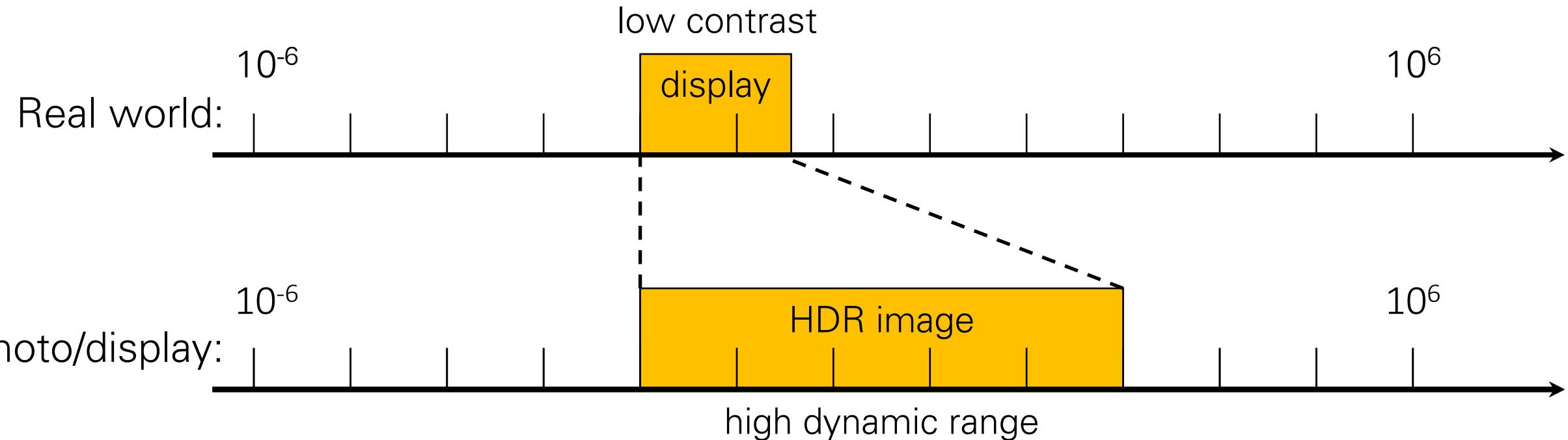
FRÉDO DURAND, MIT CSAIL / Inria, Université Côte d'Azur



# Tonemapping via bilateral filtering

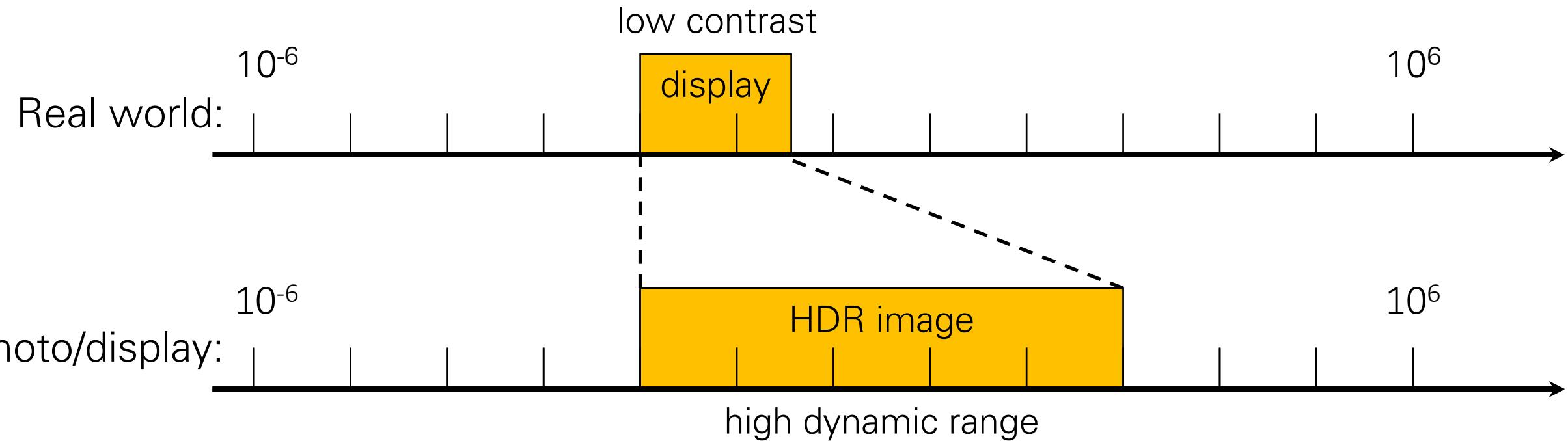
# Display the information

- Recall our HDR class
- Match limited contrast of the medium while preserving details



# Display the information

- Recall our HDR class
- Match limited contrast of the medium while preserving details



## Tonemapping

# Tonemapping

- Called tone mapping operators
- Two general categories:
  - Global (spatially invariant)
  - Local (spatially varying)

# Tone mapping for very HDR scenes

- sun overexposed
- foreground too dark



# Tone mapping for very HDR scenes

- Scene has  $>100,000:1$  dynamic range, JPEG has 255:1
- How can we compress the scene's dynamic range?



# Tone mapping for very HDR scenes

- Scene has  $>100,000:1$  dynamic range, JPEG has 255:1
- How can we compress the scene's dynamic range?
- Scale linearly?
  - If we scaled linearly from 100,000:1 to 255:1, everything but the sun would be black!



# Tonemapping w/ Simple Gamma



- gamma correction,  
applied independently  
on R, G, B:

$$I = I^\gamma$$

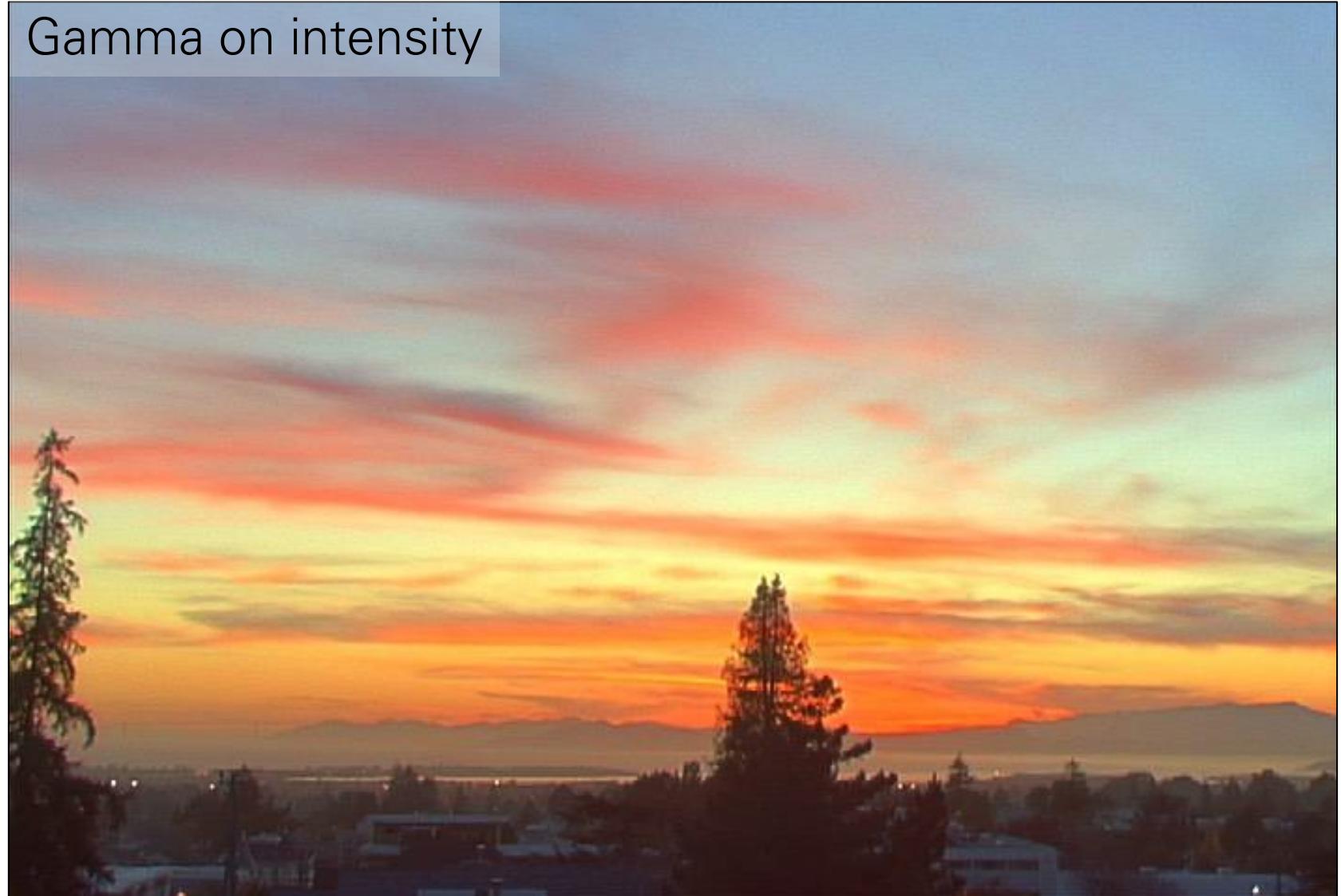
- global tonemapping
- colors are washed out

# Tonemapping w/ Simple Gamma

intensity



Gamma on intensity



color



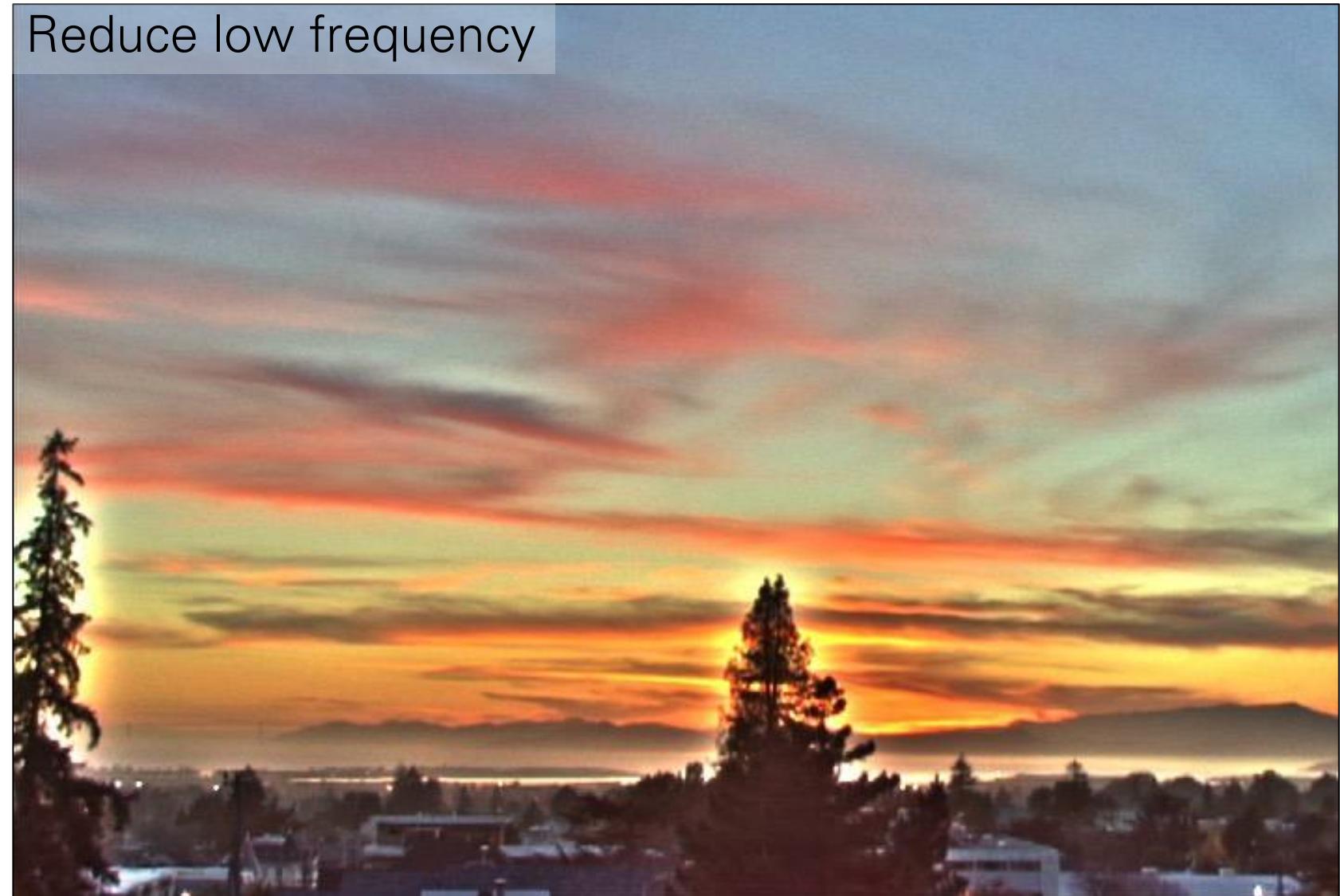
- gamma in intensity only!
- Intensity details lost

# Oppenheim 1968, Chiu et al. 1993

- Reduce contrast of low-frequencies, preserve high frequencies



Reduce low frequency

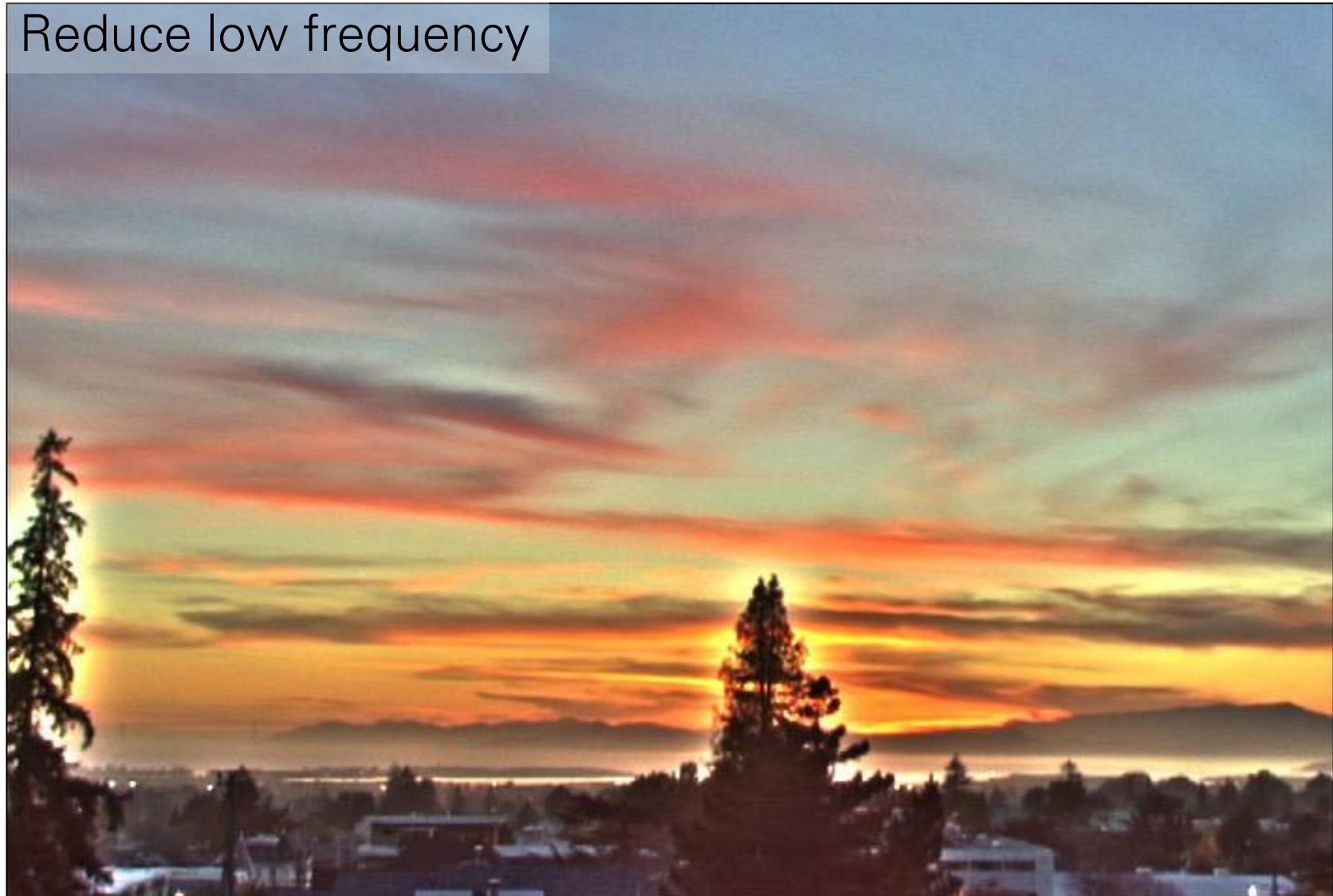


# The halo nightmare

- For strong edges;  
because they contain  
high frequency



Reduce low frequency



# The halo nightmare

- Similar to unsharp mask of luminance in log domain

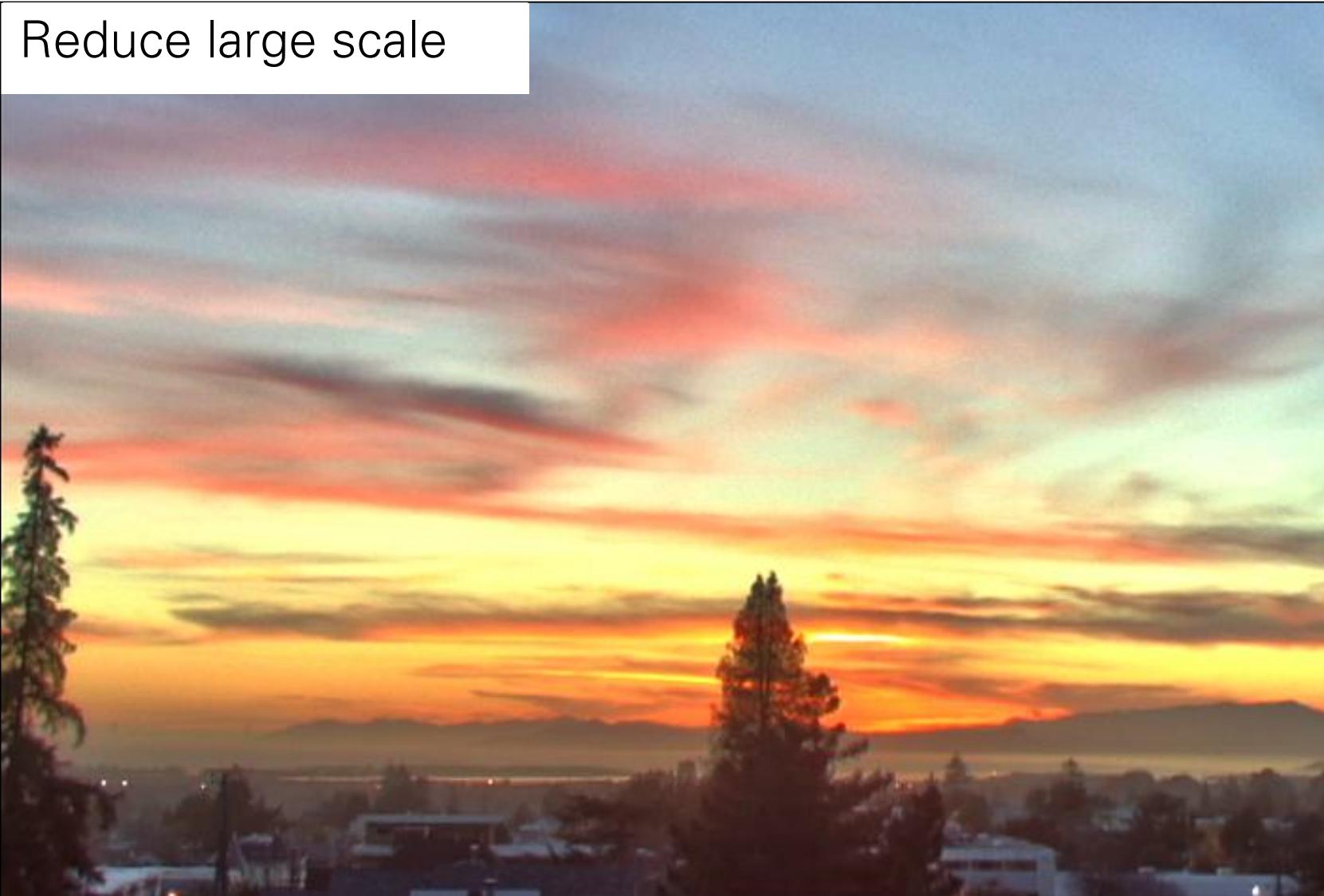
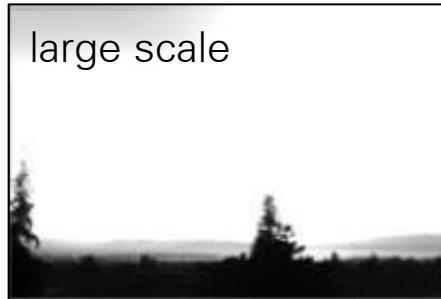


Reduce low frequency



# Tonemapping w/ Bilateral Filter (Durand and Dorsey, 2002)

- Don't blur across edges, decompose using bilateral filter



# Tonemapping w/ Bilateral Filter (Durand and Dorsey, 2002)



Contrast too high!

# Tonemapping w/ Bilateral Filter (Durand and Dorsey, 2002)



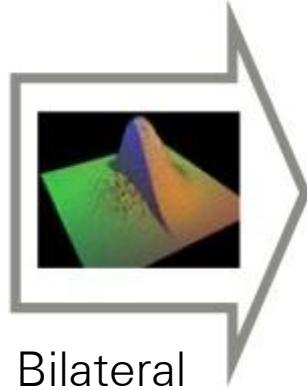
$$\text{Intensity} = 0.3R + 0.6G + 0.1B$$



$$\begin{aligned} R' &= R/\text{intensity} \\ G' &= G/\text{intensity} \\ B' &= B/\text{intensity} \end{aligned}$$

Important to use ratios  
(makes it luminance  
invariant)

# Tonemapping w/ Bilateral Filter (Durand and Dorsey, 2002)



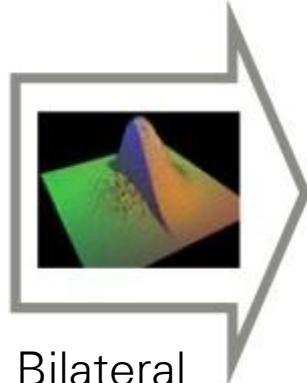
Bilateral  
Filter

in log

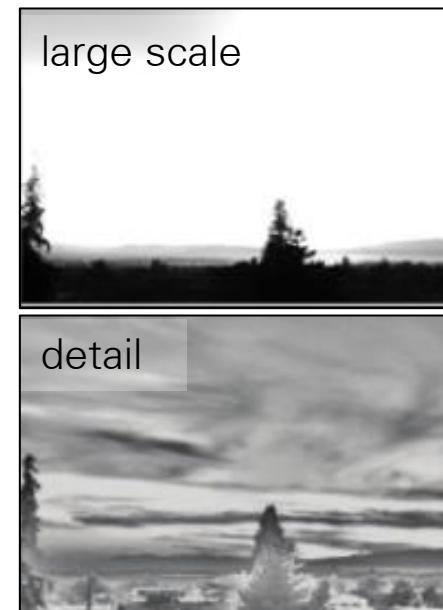


Spatial sigma: 2-5% image size  
Range sigma: 0.4 (in log 10)

# Tonemapping w/ Bilateral Filter (Durand and Dorsey, 2002)

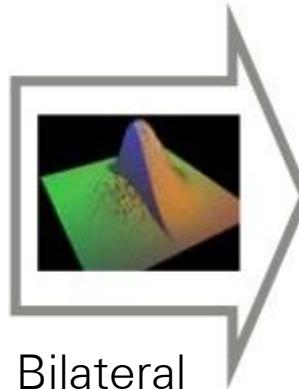


Bilateral  
Filter  
in log

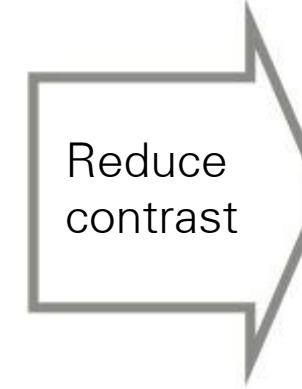
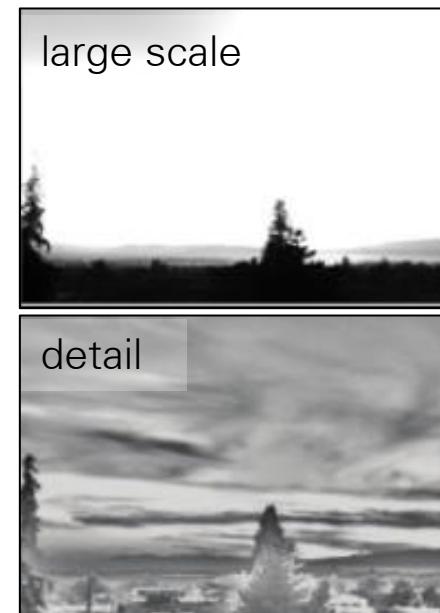


$$\text{detail} = \log \text{intensity} - \text{large scale} \\ (\text{residual})$$

# Tonemapping w/ Bilateral Filter (Durand and Dorsey, 2002)



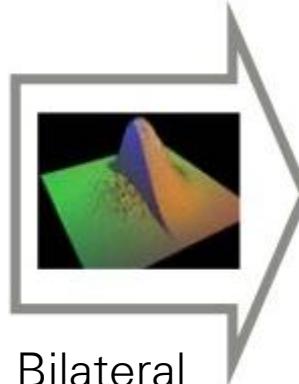
Bilateral  
Filter  
in log



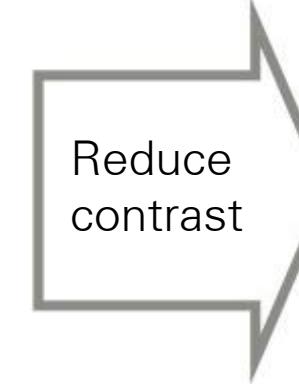
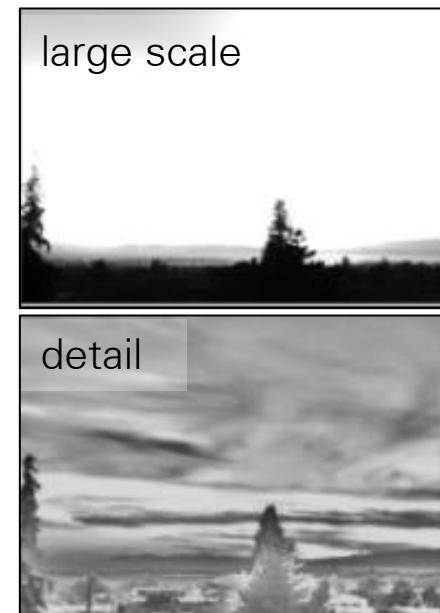
Reduce  
contrast



# Tonemapping w/ Bilateral Filter (Durand and Dorsey, 2002)



Bilateral  
Filter  
in log

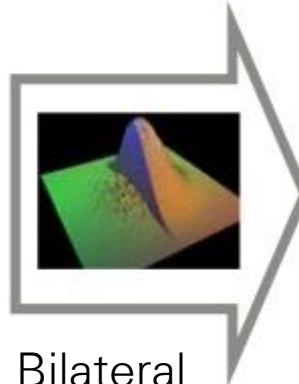


Reduce  
contrast

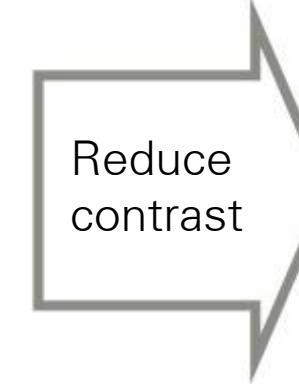
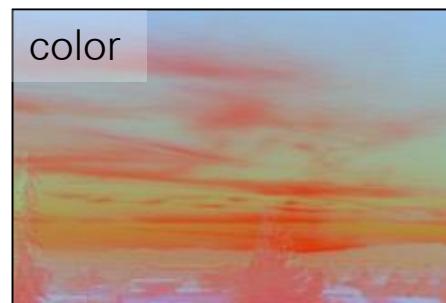
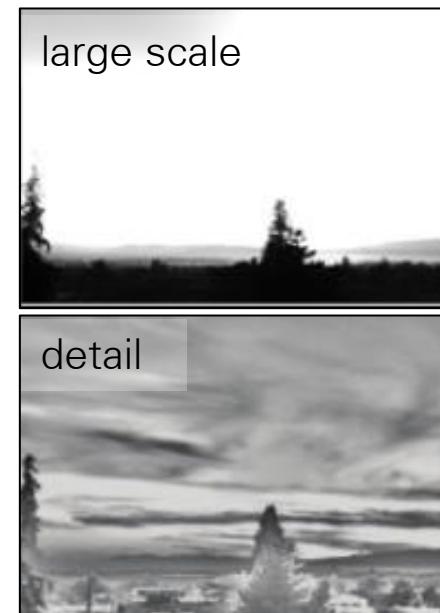
Preserve!



# Tonemapping w/ Bilateral Filter (Durand and Dorsey, 2002)



in log



Preserve!



# Log domain

- Very important to work in the log domain
- Recall: humans are sensitive to multiplicative contrast
- With log domain, our notion of “strong edge” always corresponds to the same contrast

# Scale decomposition in log domain

- $\text{inLog} = \log_{10}(\text{intensity})$
- $\text{inLogLarge} = \text{bilateralFilter}(\text{inLog})$
- $\text{inLogDetail} = \text{inLog} - \text{inLogLarge}$
- hence:
  - $\text{inLog} = \text{inLogDetail} + \text{inLogLarge}$ , or
  - $\text{intensity} = 10^{\text{inLogDetail}} * 10^{\text{inLogLarge}}$
- Now manipulate large-scale and detail separately

# Contrast reduction in log domain

$$\text{outLog} = \text{inLogDetail} + k * (\text{inLogLarge} - \max(\text{inLogLarge}))$$

- Normalize so that the biggest value is 0 in log
- Set target large-scale contrast (e.g.  $\text{targetRange} = \log_{10}(100)$ )
  - i.e. in **linear** output, we want 1:100 contrast for large scale
- Compute range of input's large-scale layer:
  - $\text{largeRange} = \max(\text{inLogLarge}) - \min(\text{inLogLarge})$
- Scale factor  $k = \text{targetRange} / \text{largeRange}$

multiplication in log  
=  $\gamma$  exponent in linear

# Contrast reduction in log domain

```
outLog = detailAmp * inLogDetail + k * (inLogLarge - max(inLogLarge))
```

- Normalize so that the biggest value is 0 in log
- Set target large-scale contrast (e.g.  $\text{targetRange} = \log_{10}(100)$ )
  - i.e. in **linear** output, we want 1:100 contrast for large scale
- Compute range of input's large-scale layer:
  - $\text{largeRange} = \max(\text{inLogLarge}) - \min(\text{inLogLarge})$
- Scale factor  $k = \text{targetRange} / \text{largeRange}$
- **Optional:** amplify detail by  $\text{detailAmp}$

# Contrast reduction in log domain

```
outLog = detailAmp*inLogDetail + k*(inLogLarge – max(inLogLarge))
```

- $\text{outIntensity} = 10^{\text{outLog}}$
- Recall that  $R', G', B'$  is the intensity-normalized RGB color
  - $\text{outR} = \text{outIntensity} * R'$
  - $\text{outG} = \text{outIntensity} * G'$
  - $\text{outB} = \text{outIntensity} * B'$

# What matters

- Spatial sigma: not very important
- Range sigma: quite important
- Use of the log domain for range: **critical**
  - Because HDR and because perception sensitive to multiplicative contrast

# Modern edge-aware filtering: domain transform

## Domain Transform for Edge-Aware Image and Video Processing

Eduardo S. L. Gastal\*      Manuel M. Oliveira†  
Instituto de Informática – UFRGS



(a) *Photograph*



(b) *Edge-aware smoothing*



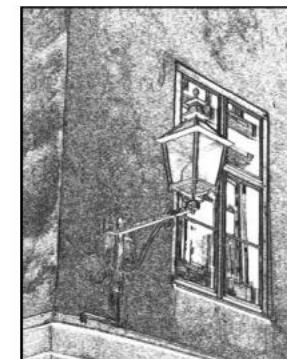
(c) *Detail enhancement*



(d) *Stylization*



(e) *Recoloring*



(f) *Pencil drawing*



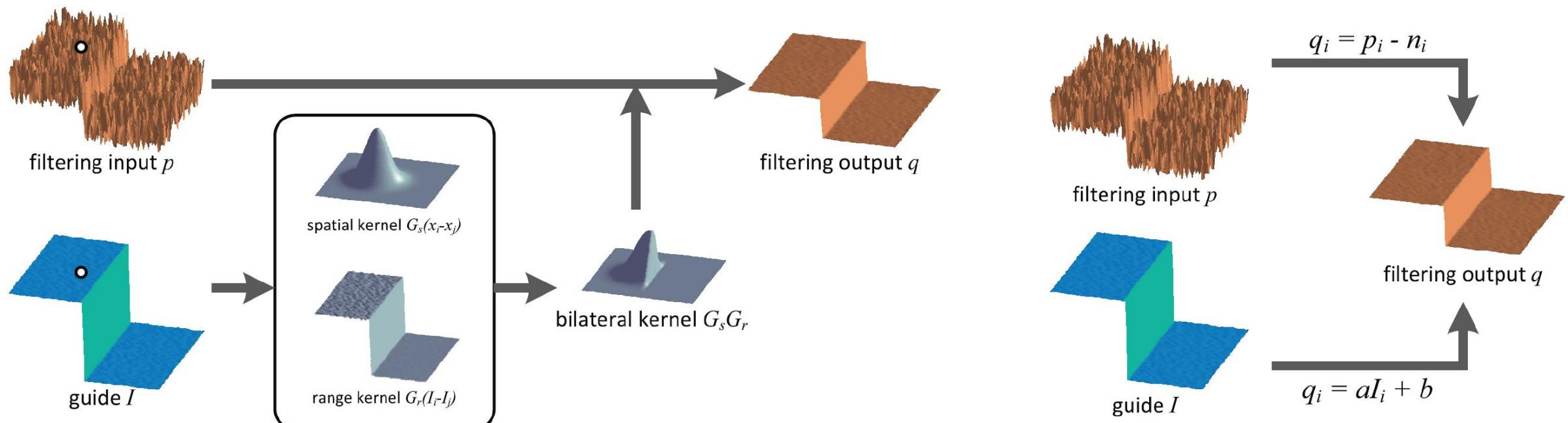
(g) *Depth-of-field*

Lots of great examples at: <https://www.inf.ufrgs.br/~eslgastal/DomainTransform/>

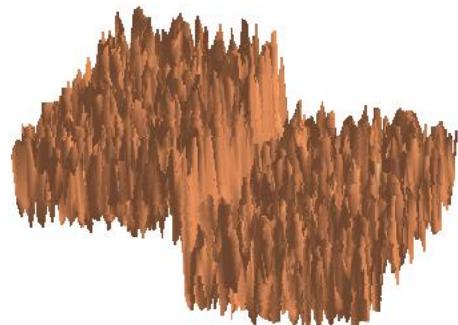
# Modern edge-aware filtering: guided filter

## Guided Image Filtering

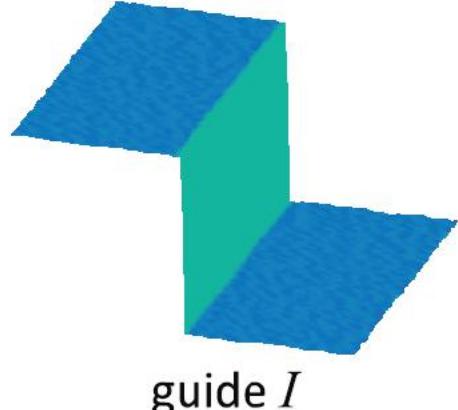
Kaiming He, *Member, IEEE*, Jian Sun, *Member, IEEE*, and Xiaou Tang, *Fellow, IEEE*



# Guided Image Filtering



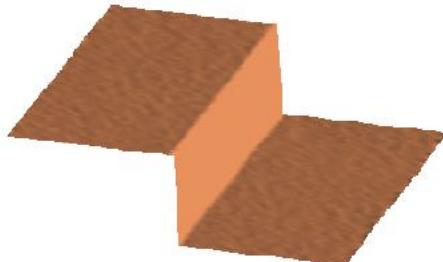
filtering input  $p$



guide  $I$

$$q_i = p_i - n_i$$

$n_i$  : noise / texture



filtering output  $q$

$$\nabla q_i = a \nabla I_i$$

$$q_i = aI_i + b$$

Bilateral/joint bilateral filter does  
not have this linear model

$$\text{minimize } E(a_k, b_k) = \sum_{i \in \omega_k} ((a_k I_i + b_k - p_i)^2 + \epsilon a_k^2)$$

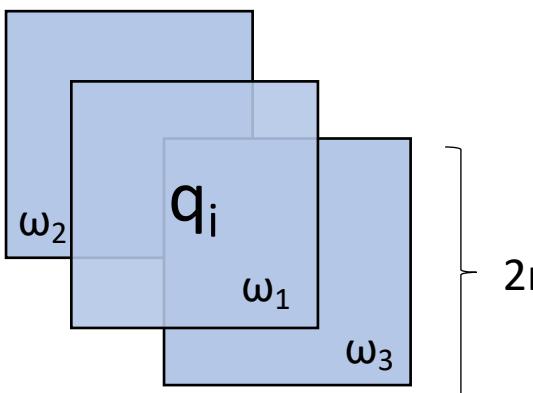
Linear regression

$$a_k = \frac{\frac{1}{|\omega|} \sum_{i \in \omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon},$$

$$b_k = \bar{p}_k - a_k \mu_k.$$

# Guided Image Filtering

- Extend to the entire image
  - In all local windows  $\omega_k$ , compute the linear coefficients
  - Compute the average of  $a_k l_i + b_k$  in all  $\omega_k$  that covers pixel  $q_i$



$$a_k = \frac{\frac{1}{|\omega|} \sum_{i \in \omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon},$$

$$b_k = \bar{p}_k - a_k \mu_k.$$

$$q_i = \frac{1}{|\omega|} \sum_{k | i \in \omega_k} (a_k I_i + b_k)$$

## Definition

### Algorithm 1. Guided Filter.

**Input:** filtering input image  $p$ , guidance image  $I$ , radius  $r$ , regularization  $\epsilon$

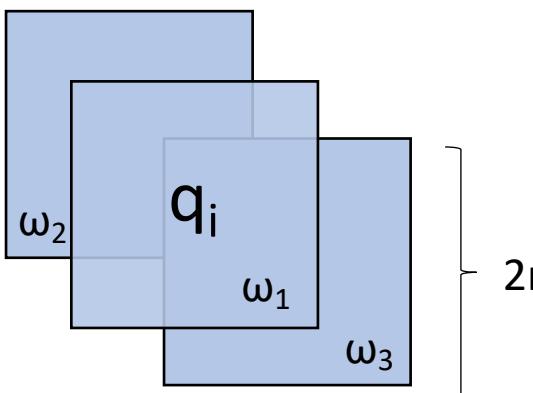
**Output:** filtering output  $q$ .

```
1: meanI = fmean(I)
   meanp = fmean(p)
   corrI = fmean(I.*I)
   corrIp = fmean(I.*p)
2: varI = corrI - meanI.*meanI
   covIp = corrIp - meanI.*meanp
3: a = covIp./(varI + ε)
   b = meanp - a.*meanI
4: meana = fmean(a)
   meanb = fmean(b)
5: q = meana.*I + meanb
```

/\*  $f_{mean}$  is a mean filter with a wide variety of  $O(N)$  time methods. \*/

# Guided Image Filtering

- Edge-preserving filter
- $O(1)$  time, fast, accurate
- Gradient preserving
- Parameters
  - Window radius  $r$
  - Regularization  $\epsilon$



## Definition

$$a_k = \frac{\frac{1}{|\omega|} \sum_{i \in \omega_k} I_i p_i - \mu_k \bar{p}_k}{\sigma_k^2 + \epsilon},$$

$$b_k = \bar{p}_k - a_k \mu_k.$$

$$q_i = \frac{1}{|\omega|} \sum_{k|i \in \omega_k} (a_k I_i + b_k)$$

### Algorithm 1. Guided Filter.

**Input:** filtering input image  $p$ , guidance image  $I$ , radius  $r$ , regularization  $\epsilon$

**Output:** filtering output  $q$ .

```
1: meanI = fmean(I)
   meanp = fmean(p)
   corrI = fmean(I.*I)
   corrIp = fmean(I.*p)
2: varI = corrI - meanI.*meanI
   covIp = corrIp - meanI.*meanp
3: a = covIp./(varI + ε)
   b = meanp - a.*meanI
4: meana = fmean(a)
   meanb = fmean(b)
5: q = meana.*I + meanb
```

/\*  $f_{\text{mean}}$  is a mean filter with a wide variety of  $O(N)$  time methods. \*/

# Example: Edge-Preserving Smoothing



$\sigma_s=2$



$\sigma_s=4$



$\sigma_s=8$



$\sigma_r=0.1$

$\sigma_r=0.2$

Bilateral Filter

$r=2$



$r=4$



$r=8$



$\epsilon=0.1^2$

$\epsilon=0.2^2$

Guided Filter

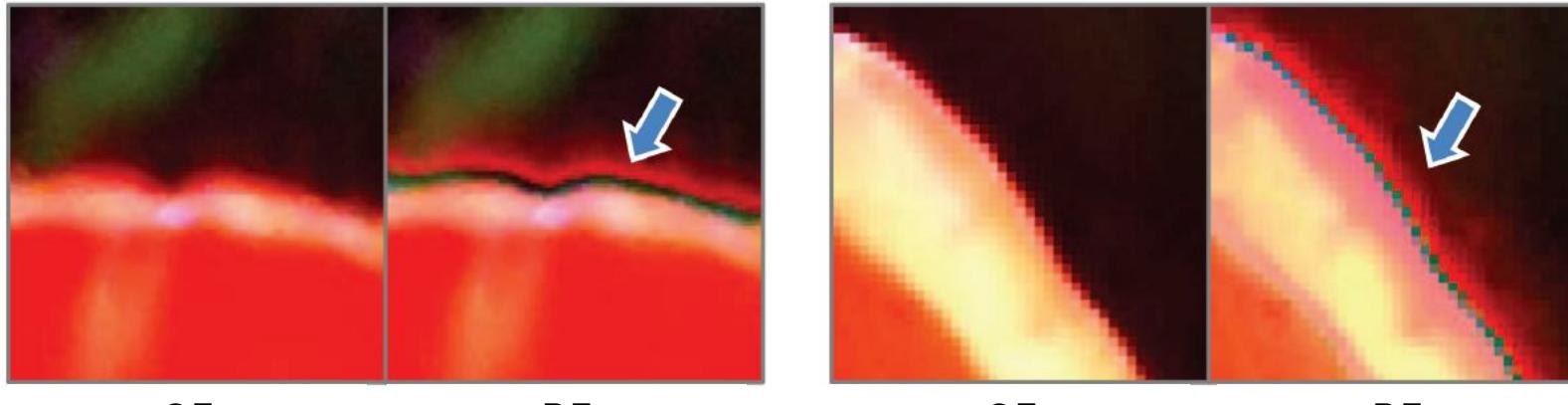
# Example: Detail Enhancement



Original

Guided Filter

Bilateral Filter



GF

BF

GF

BF

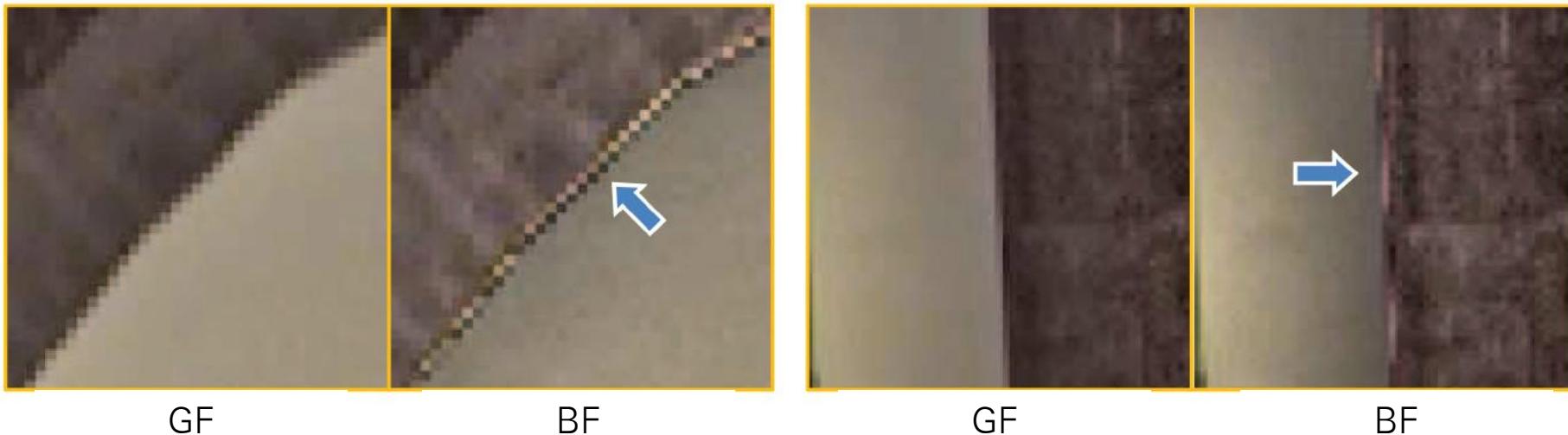
# Example: Tonemapping



Original HDR

Guided Filter

Bilateral Filter



GF

BF

GF

BF

# Example: Flash/No-Flash Photography



Guidance I



Guided Filter



Filter Input p

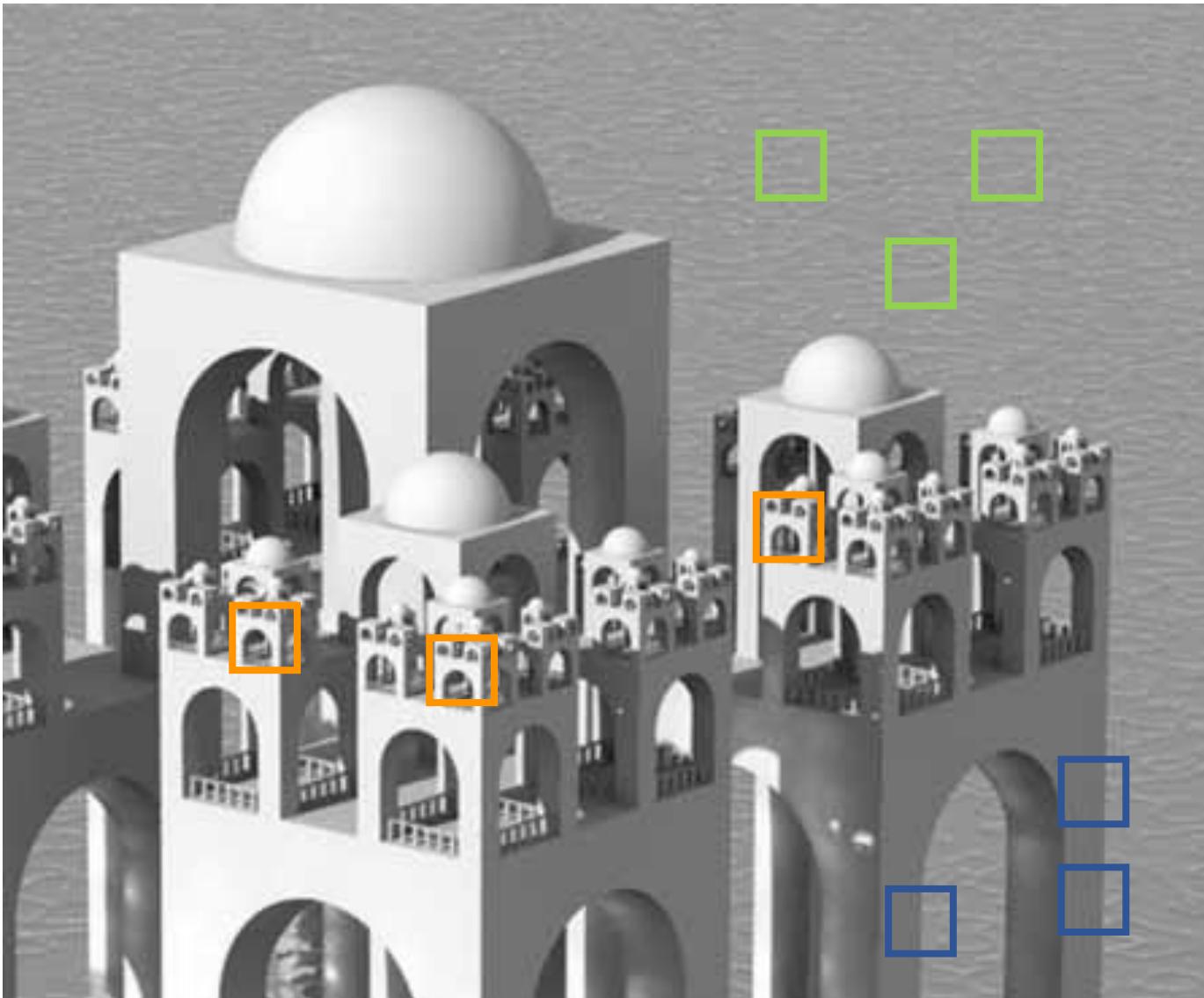


Joint Bilateral Filter



# Non-Local Means Filter

# Redundancy in natural images

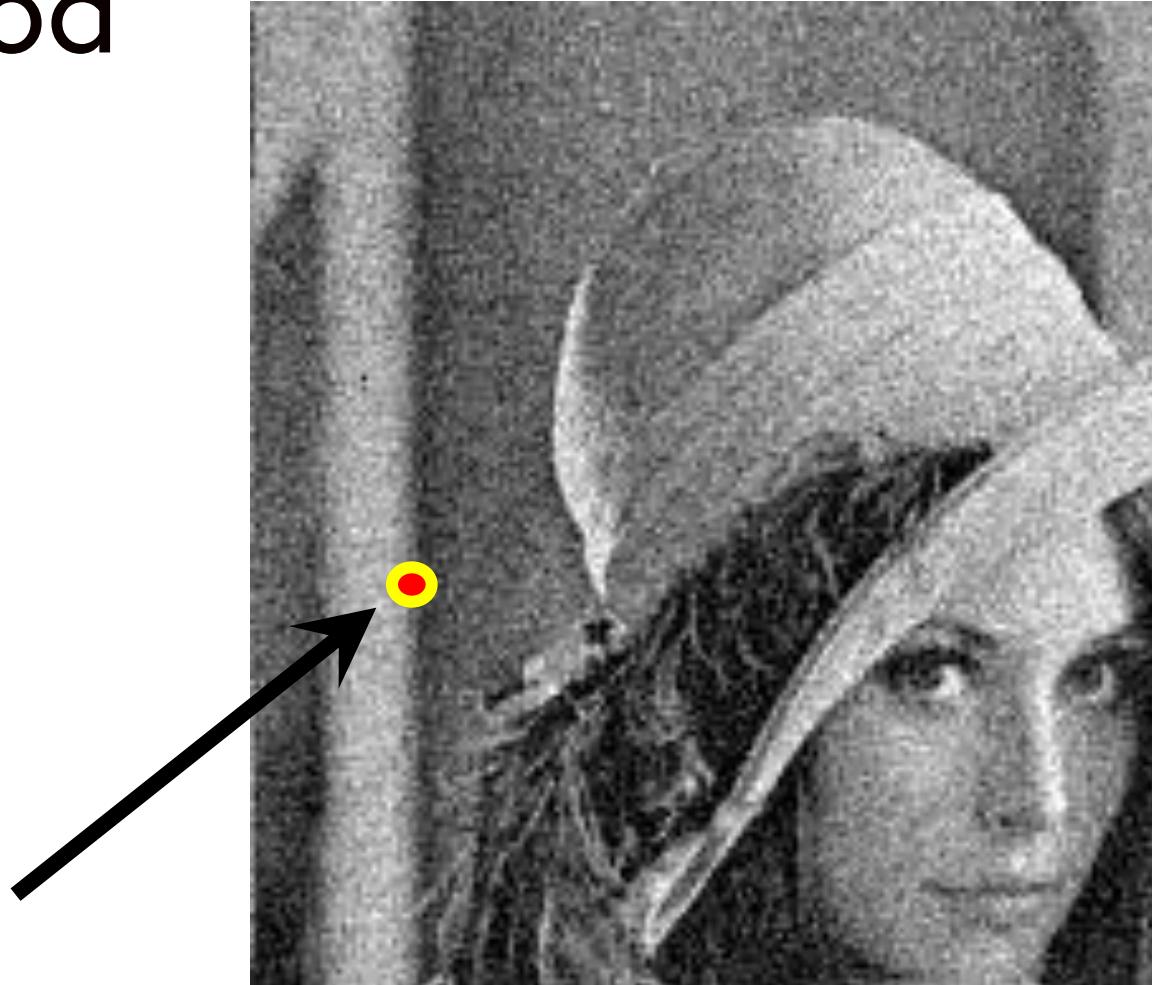


# NL-Means Filter (Buades 2005)

- 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

- For each and every pixel  $p$ :



# NL-Means Method

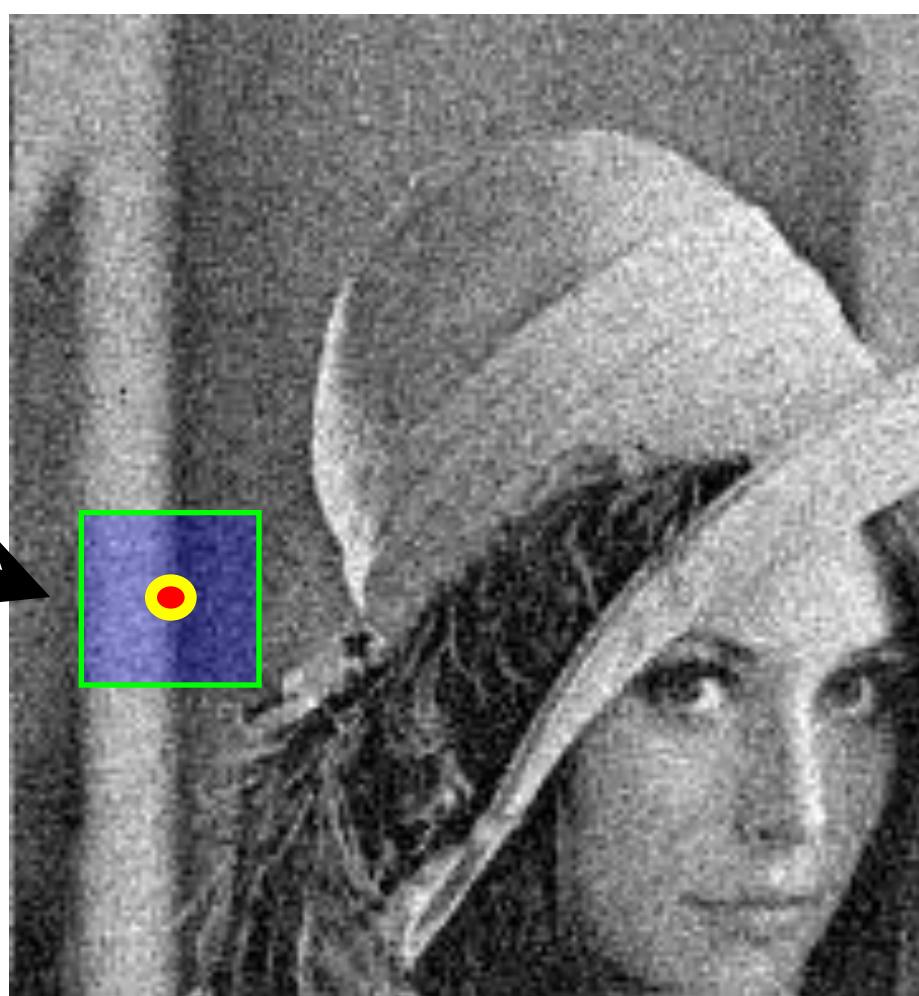
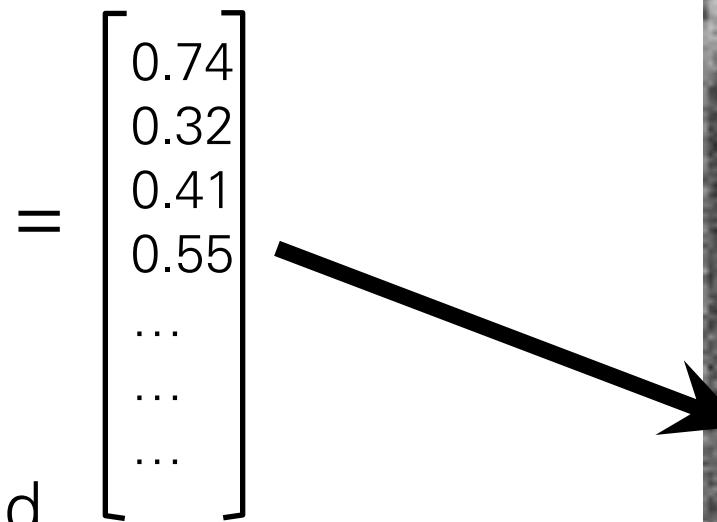
- For each and every pixel  $p$ :
  - Define a small, simple fixed size neighborhood;



# NL-Means Method

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

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



# NL-Means Method

'Similar' pixels p, q

→ SMALL

vector distance;

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

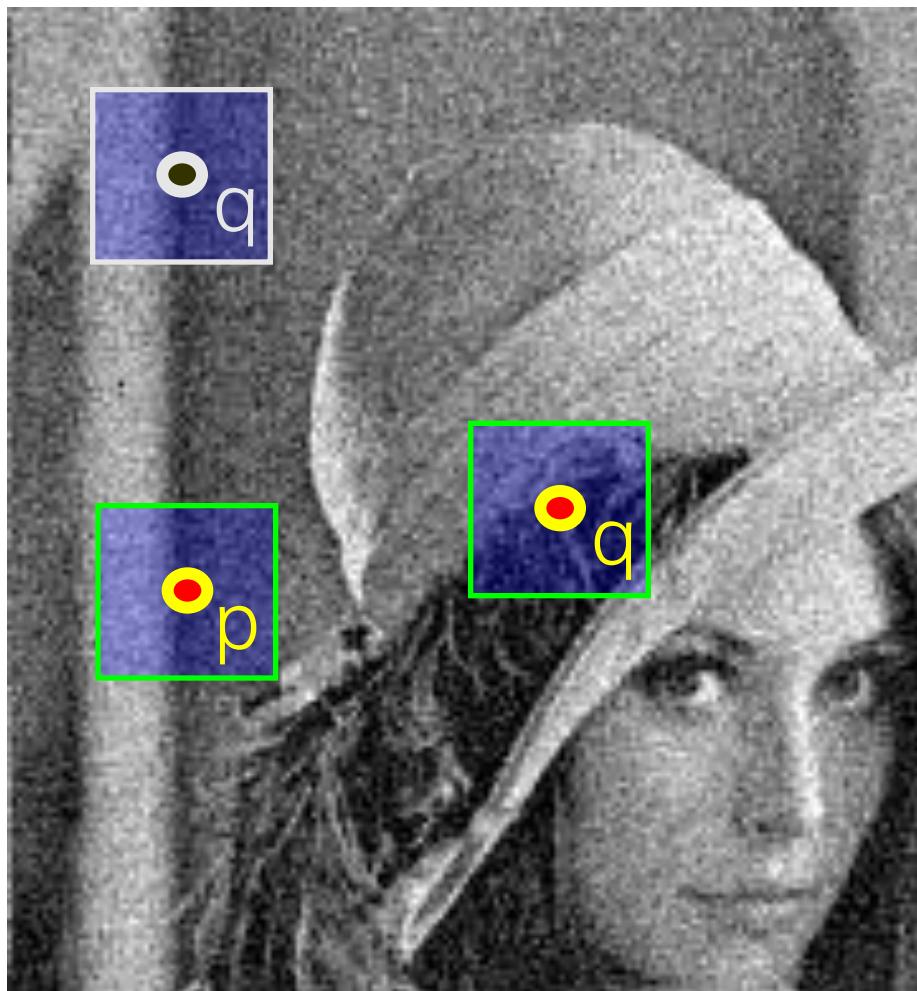


# NL-Means Method

'Dissimilar' pixels  $p, q$

→ LARGE  
vector distance;

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



# NL-Means Method

'Dissimilar' pixels  $p, q$

→ LARGE  
vector distance;

$$\boxed{\| V_p - V_q \|^2}$$

Filter with this!



# NL-Means Method

p, q neighbors define  
a vector distance;

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

Filter with this:

No spatial term!

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



# NL-Means Method

pixels p, q neighbors  
Set a vector distance;

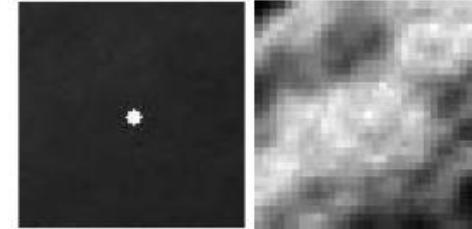
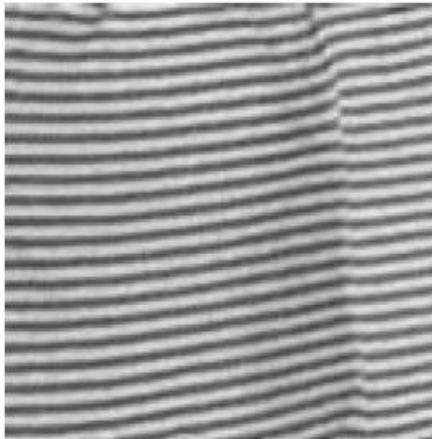
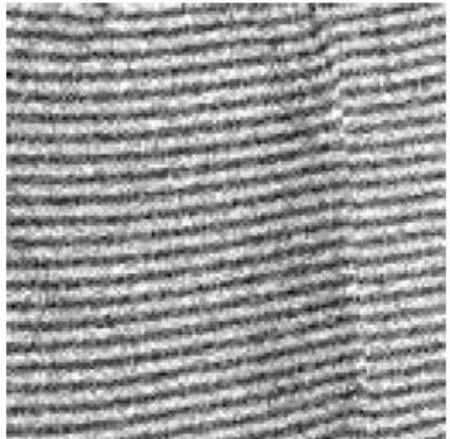
$$\| V_p - V_q \|^2$$



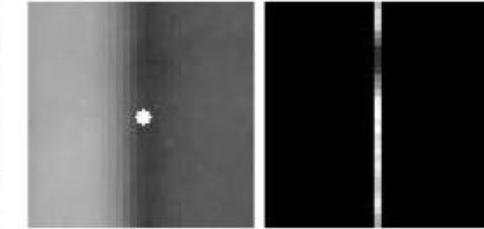
Vector Distance to p sets weight for each pixel q

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

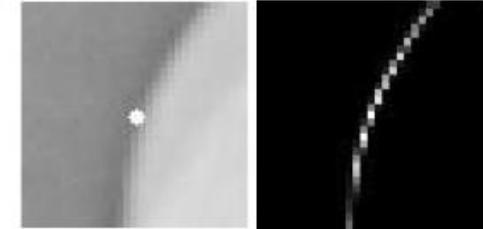
# NL-Means Method



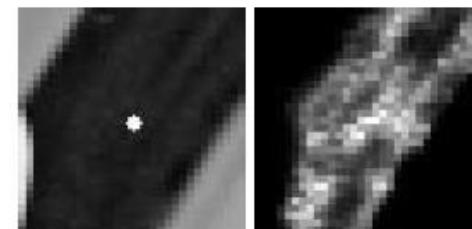
(a)



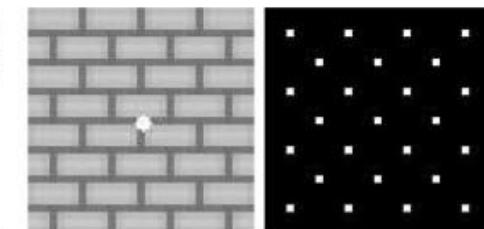
(b)



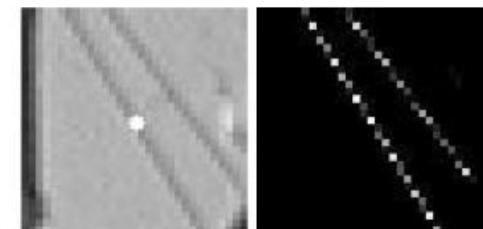
(c)



(d)



(e)



(f)

# NL-Means Method

- Noisy source image:



# NL-Means Method

- Gaussian Filter

Low noise,  
Low detail



# NL-Means Method

- Anisotropic Diffusion

Note 'stairsteps':  
~ piecewise constant



# NL-Means Method

- Bilateral Filter

Better, but similar  
'stairsteps':



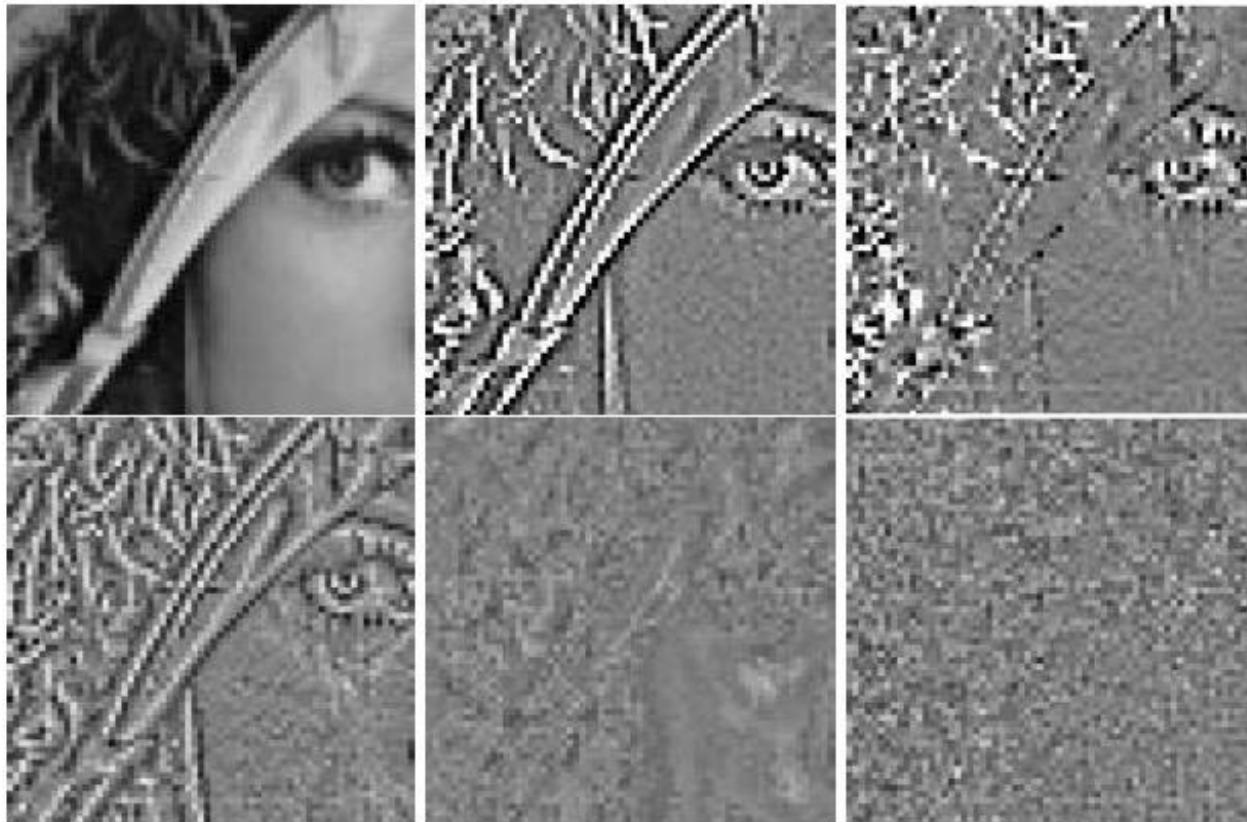
# NL-Means Method

- NL-Means:

Sharp,  
Low noise,  
Few artifacts.

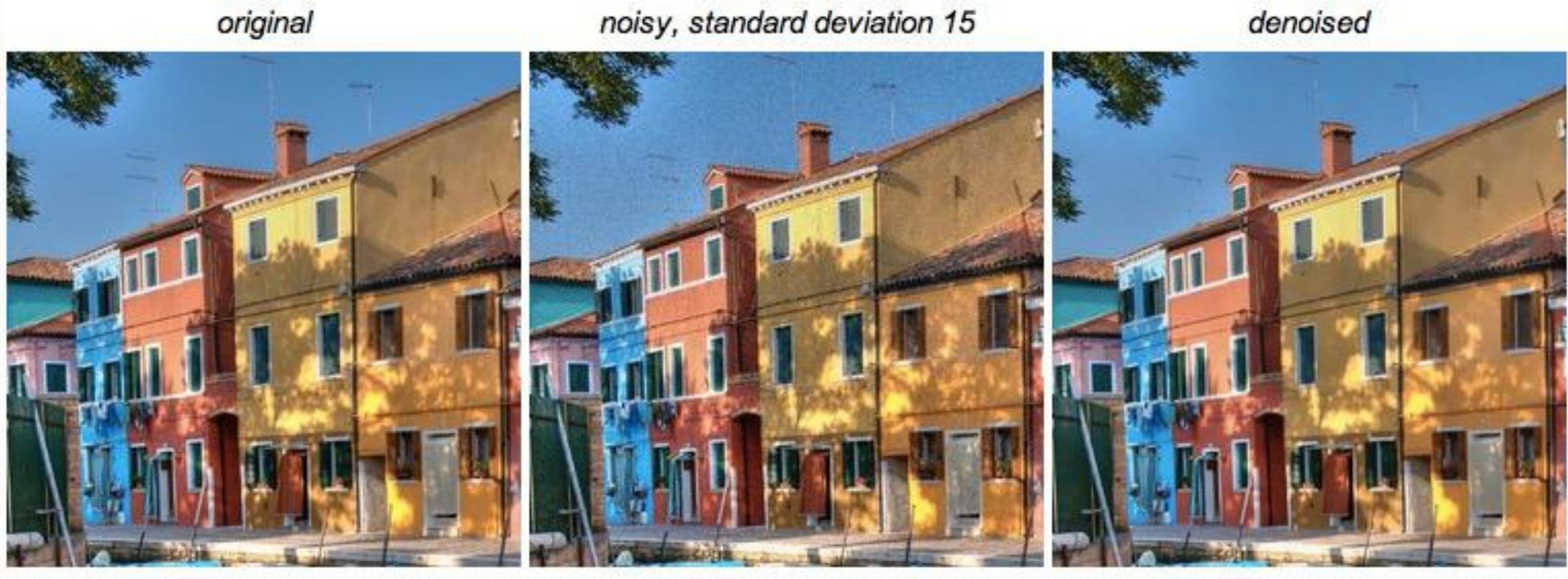


# NL-Means Method



**Figure 4. Method noise experience on a natural image. Displaying of the image difference  $u - D_h(u)$ . From left to right and from top to bottom: original image, Gauss filtering, anisotropic filtering, Total variation minimization, Neighborhood filtering and NL-means algorithm. The visual experiments corroborate the formulas of section 2.**

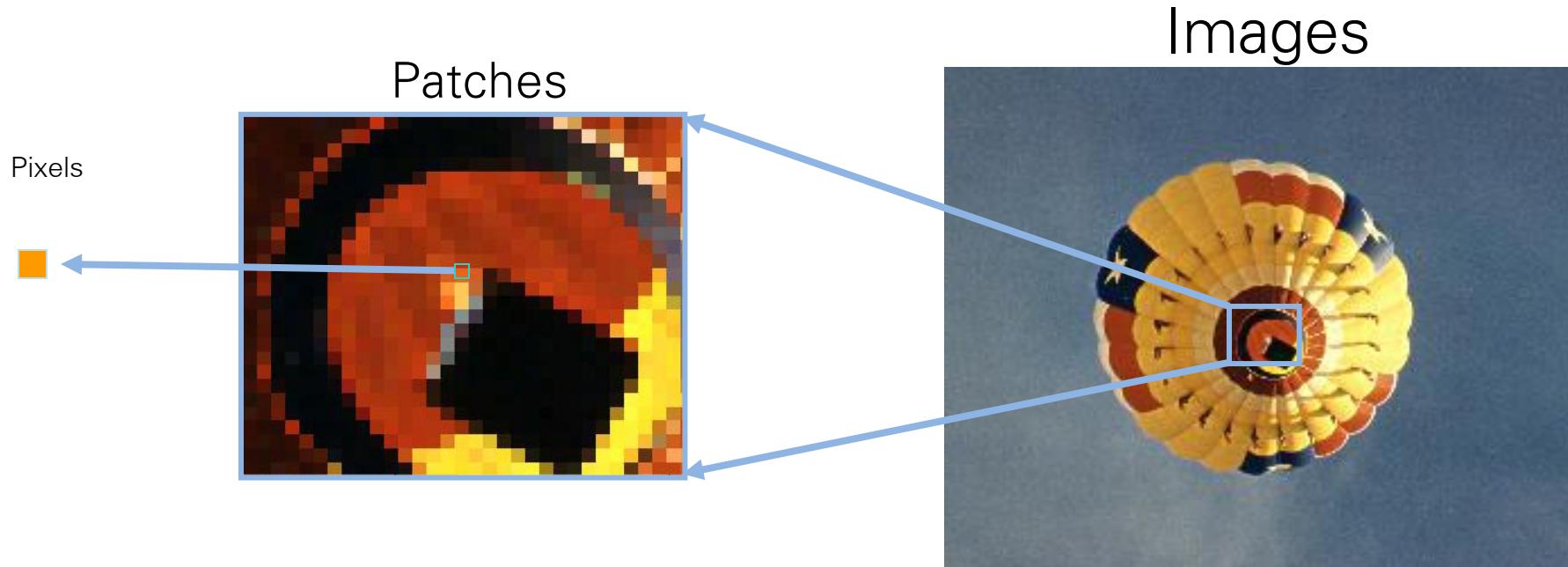
# NL-Means Method



[http://www.ipol.im/pub/algo/bcm\\_non\\_local\\_means\\_denoising/](http://www.ipol.im/pub/algo/bcm_non_local_means_denoising/)

# RegCov Smoothing

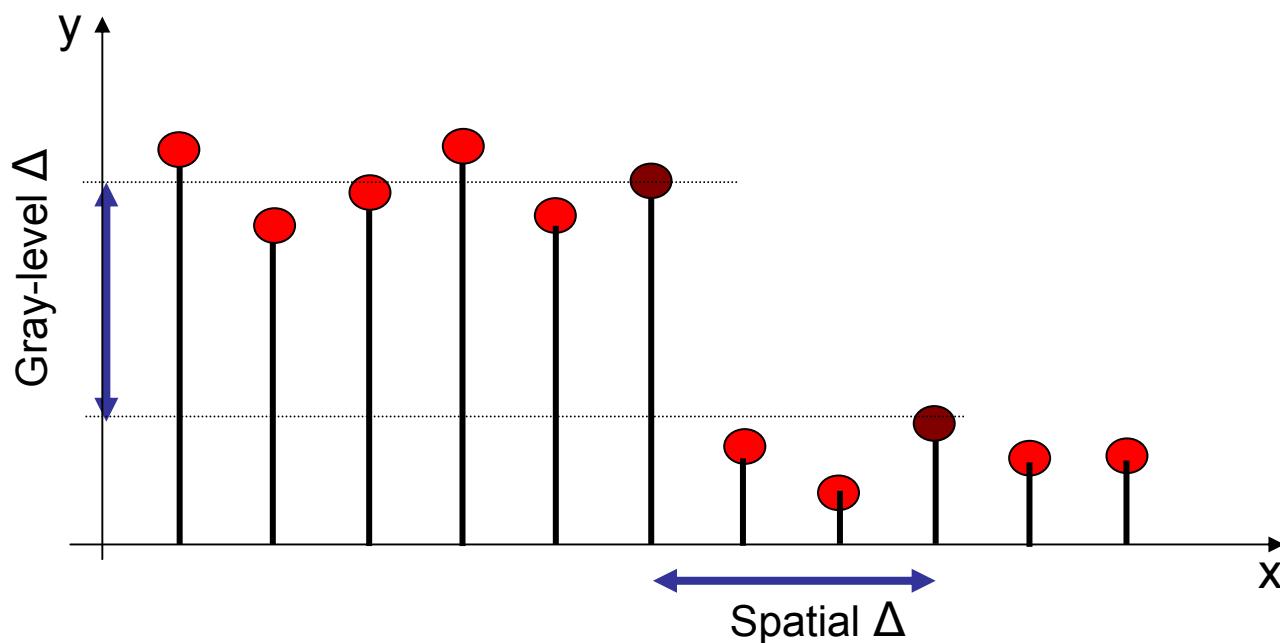
# From pixels to patches and to images



Similarities can be defined at different scales..

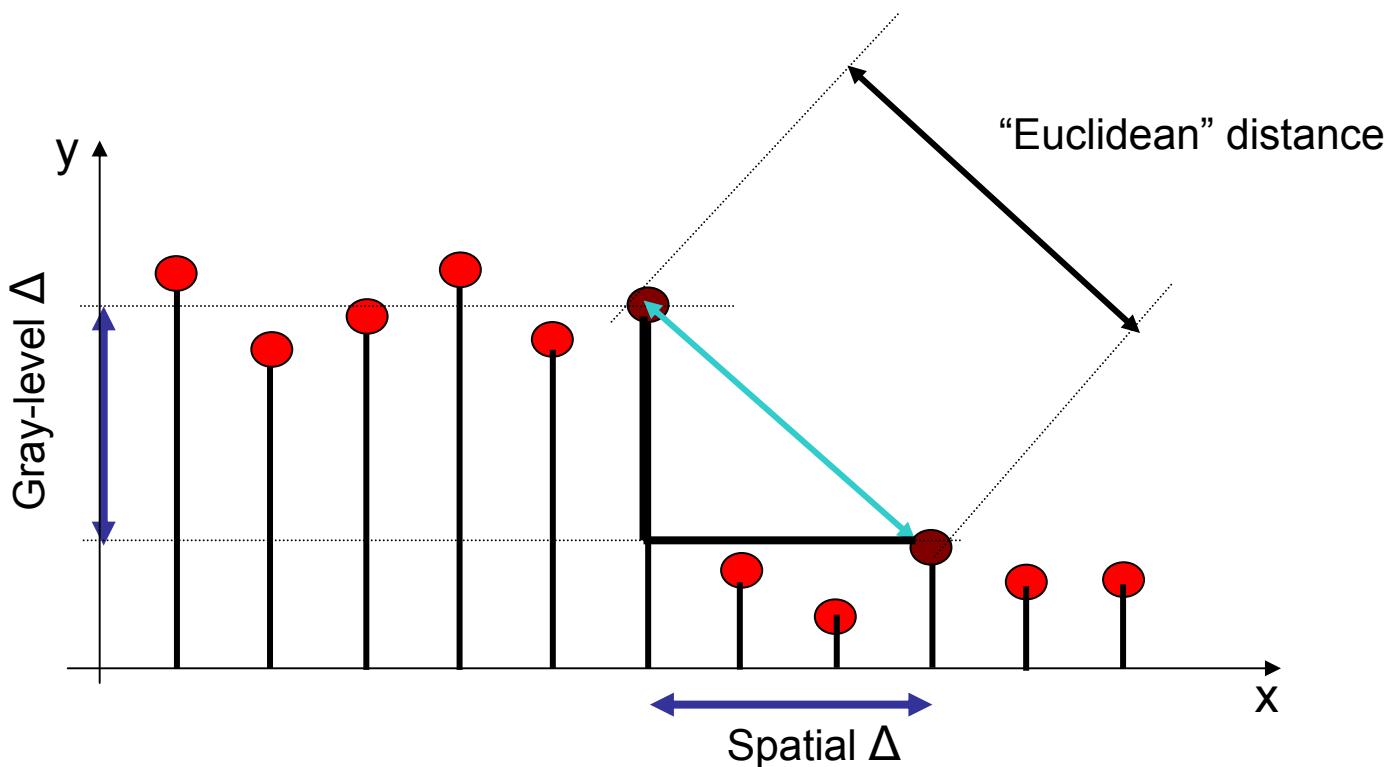
# Pixelwise similarity metrics

- To measure the similarity of two pixels, we can consider
  - Spatial distance
  - Gray-level distance



# Euclidean metrics

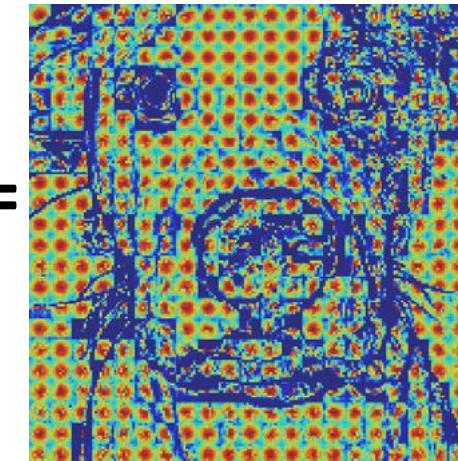
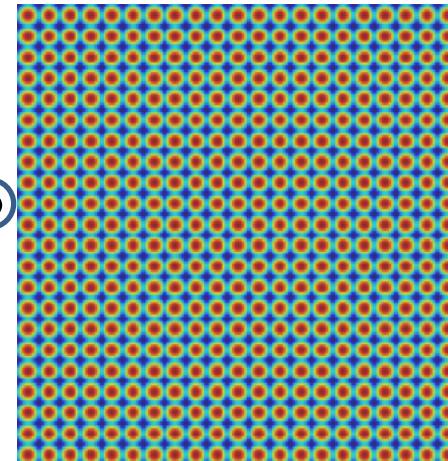
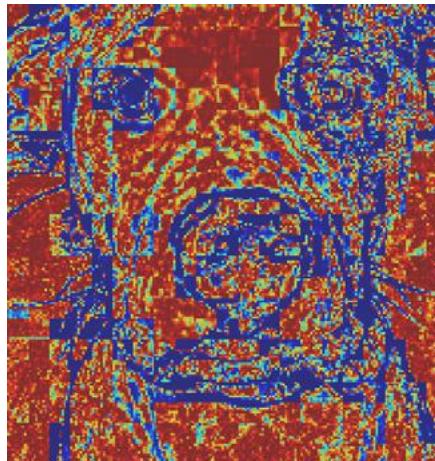
- Natural ways to incorporate the two  $\Delta$ s:
  - Bilateral Kernel [Tomasi, Manduchi, '98] (pixelwise)
  - Non-Local Means Kernel [Buades, et al. '05] (patchwise)



# Bilateral Kernel (BL) [Tomasi et al. '98]

$$K(\mathbf{x}_l, \mathbf{x}, y_l, y) = \exp \left\{ -\frac{\|\mathbf{y}_l - \mathbf{y}\|^2}{h_r^2} - \frac{\|\mathbf{x}_l - \mathbf{x}\|^2}{h_d^2} \right\}$$

Pixels  
↓  
Pixel similarity      ↓  
Spatial similarity



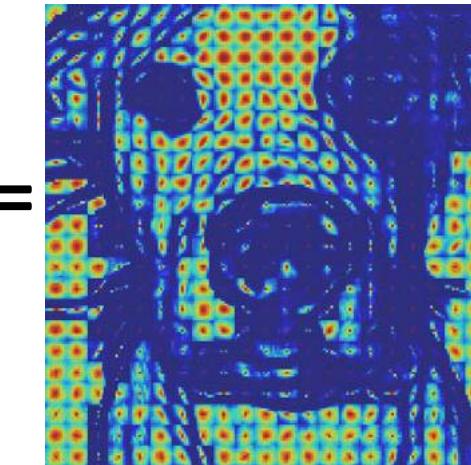
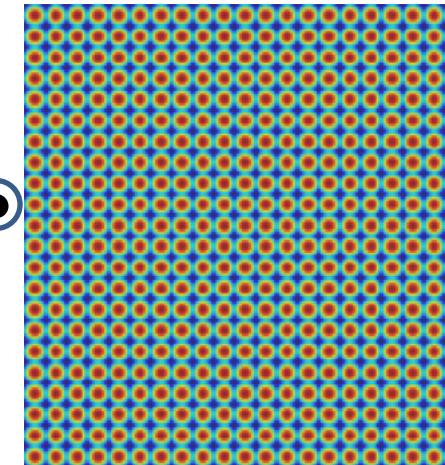
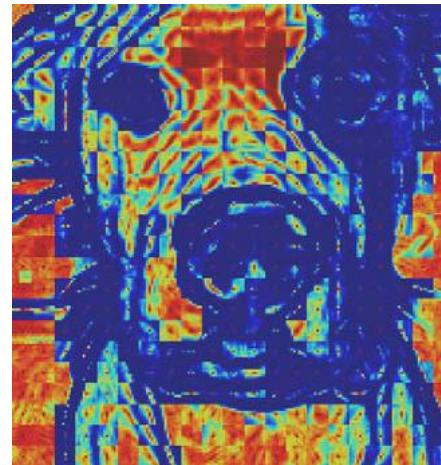
# Non-local Means (NLM) [Buades et al. '05]

$$K(\mathbf{x}_l, \mathbf{x}, \mathbf{y}_l, \mathbf{y}) = \exp \left\{ -\frac{\|\mathbf{y}_l - \mathbf{y}\|^2}{h_r^2} - \frac{\|\mathbf{x}_l - \mathbf{x}\|^2}{h_d^2} \right\}$$

Patches

Patch similarity

Spatial similarity



Smoothing effect

# Structure-Texture Decomposition

- Decomposing an image into structure and texture components

Input Image



# Structure-Texture Decomposition

- Decomposing an image into structure and texture components

Structure Component



# Structure-Texture Decomposition

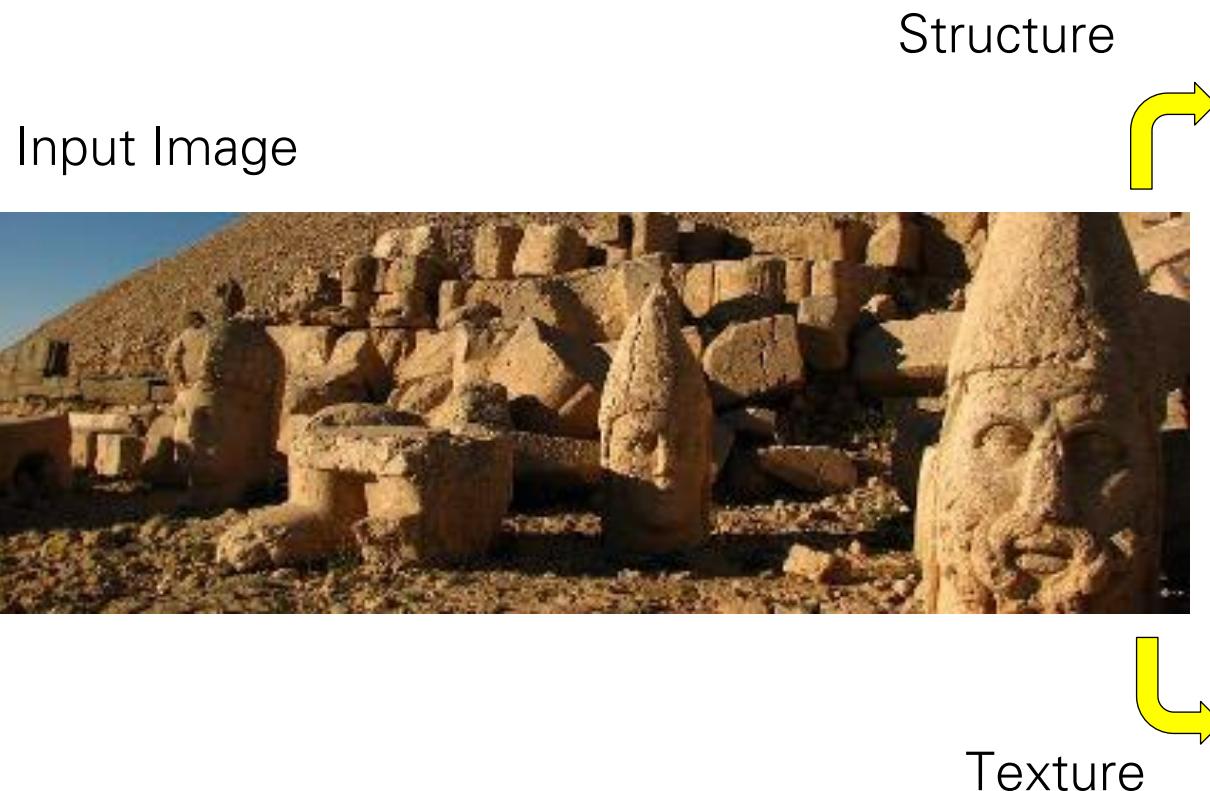
- Decomposing an image into structure and texture components

Texture Component

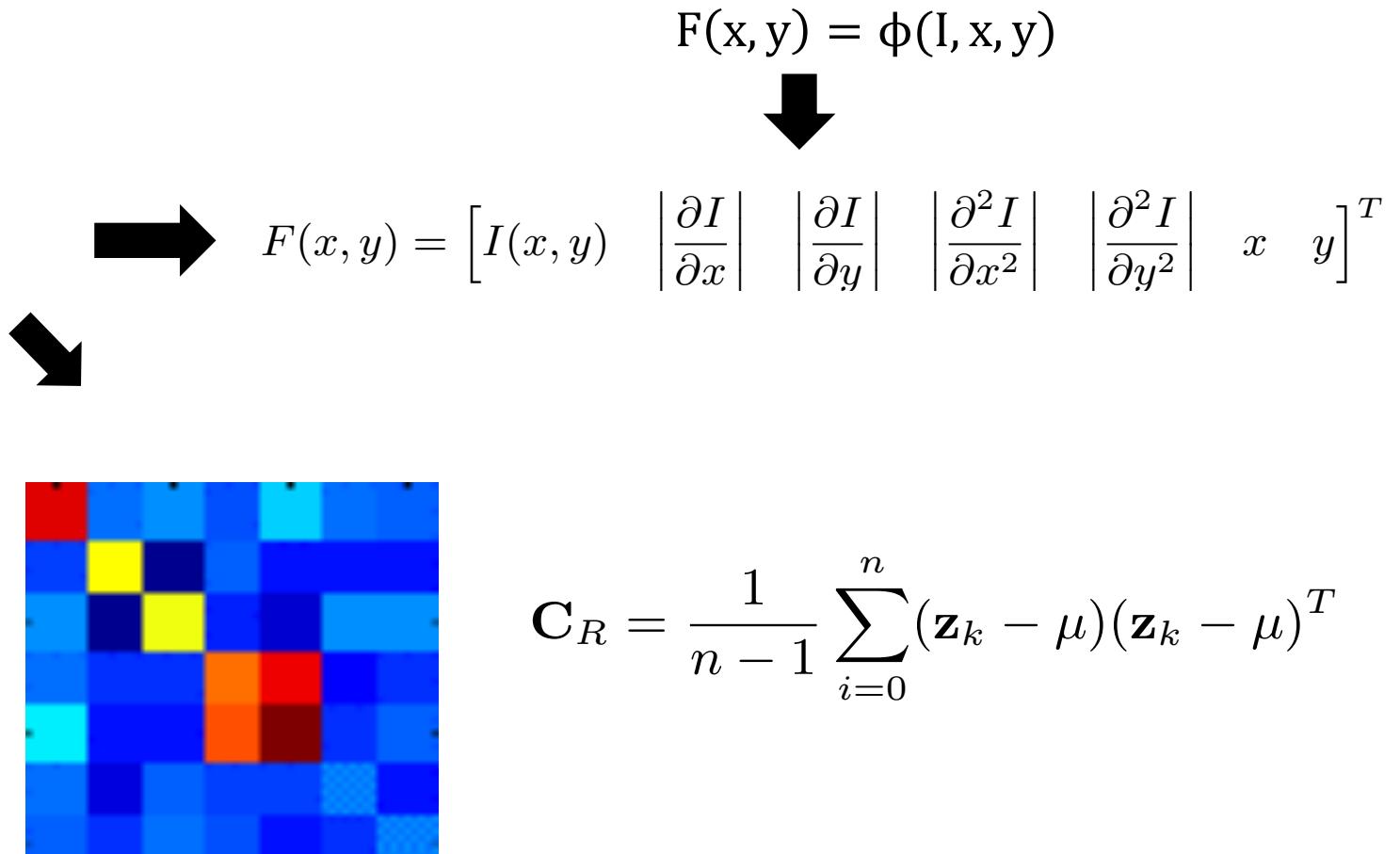


# Structure-Texture Decomposition

- Decomposing an image into structure and texture components



# Structure-Texture Decomposition



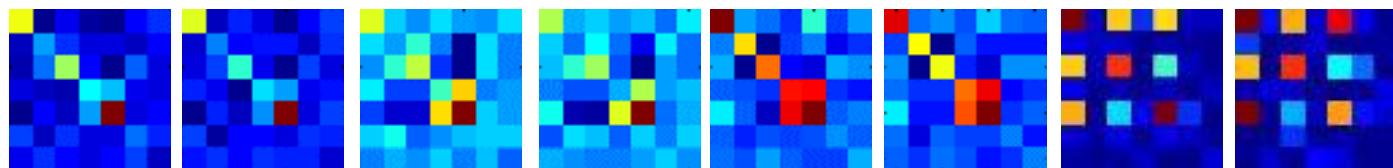
Tuzel et al., ECCV 2006

$$\mathbf{C}_R = \frac{1}{n-1} \sum_{i=0}^n (\mathbf{z}_k - \mu)(\mathbf{z}_k - \mu)^T$$

# Structure-Texture Decomposition



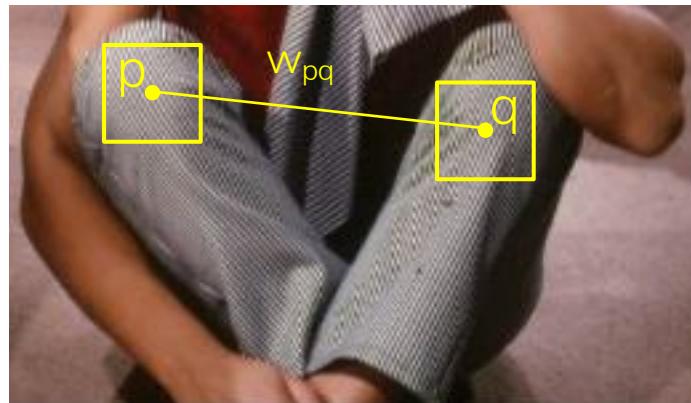
- Region covariances capture local structure and texture information.
- Similar regions have similar statistics.



# RegCov Smoothing - Formulation

$$I = S + T$$

$$S(\mathbf{p}) = \frac{1}{Z_{\mathbf{p}}} \sum_{\mathbf{q} \in N(\mathbf{p}, r)} w_{\mathbf{pq}} I(\mathbf{q})$$



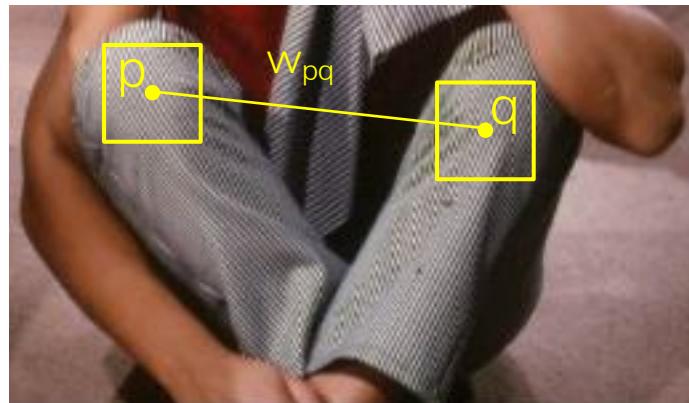
- Structure-texture decomposition via smoothing
  - Smoothing as weighted averaging
  - Different kernels ( $w_{pq}$ ) result in different types of filters.
  - Three novel patch-based kernels for structure texture decomposition.
- 
- L. Karacan, A. Erdem, E. Erdem, "Structure Preserving Image Smoothing via Region Covariances", ACM TOG 2013 (SIGGRAPH Asia 2013)

# RegCov Smoothing – Model 1

- Depends on sigma-points representation of covariance matrices  
(Hong et al., CVPR'09)

$$\mathbf{C} = \mathbf{L}\mathbf{L}^T \quad \text{Cholesky Decomposition}$$

$$\mathcal{S} = \{\mathbf{s}_i\} \quad \text{Sigma Points} \quad \mathbf{s}_i = \begin{cases} \alpha\sqrt{d}\mathbf{L}_i & \text{if } 1 \leq i \leq d \\ -\alpha\sqrt{d}\mathbf{L}_i & \text{if } d+1 \leq i \leq 2d \end{cases}$$



Final representation

$$\Psi(\mathbf{C}) = (\mu, \mathbf{s}_1, \dots, \mathbf{s}_d, \mathbf{s}_{d+1}, \dots, \mathbf{s}_{2d})^T$$

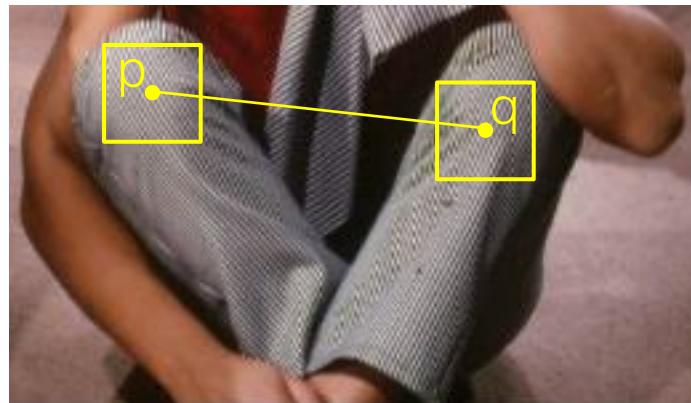
Resulting kernel function

$$w_{\mathbf{p}\mathbf{q}} \propto \exp\left(-\frac{\|\Psi(\mathbf{C_p}) - \Psi(\mathbf{C_q})\|^2}{2\sigma^2}\right)$$

# RegCov Smoothing – Model 2

- An alternative way is to use statistical similarity measures.
- A Mahalanobis-like distance measure to compare to image patches.

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(\mu_{\mathbf{p}} - \mu_{\mathbf{q}}) \mathbf{C}^{-1} (\mu_{\mathbf{p}} - \mu_{\mathbf{q}})^T}$$

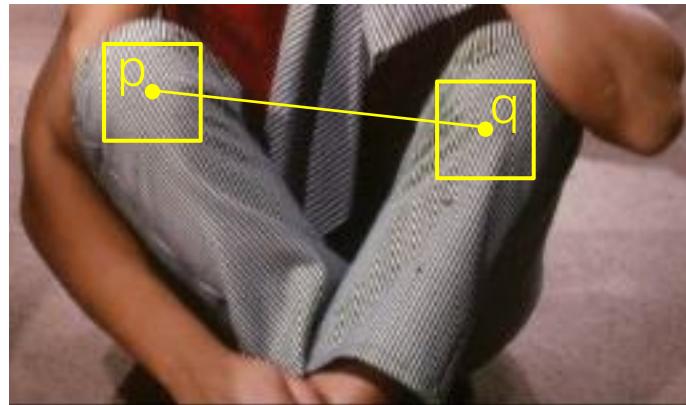


$$\mathbf{C} = \mathbf{C}_{\mathbf{p}} + \mathbf{C}_{\mathbf{q}}$$

Resulting kernel       $w_{\mathbf{pq}} \propto \exp \left( -\frac{d(\mathbf{p}, \mathbf{q})^2}{2\sigma^2} \right)$

# RegCov Smoothing – Model 3

- We use Kullback-Leibler(KL)-Divergence measure from probability theory.
- A KL-Divergence form is used to calculate statistical distance between two multivariate normal distribution



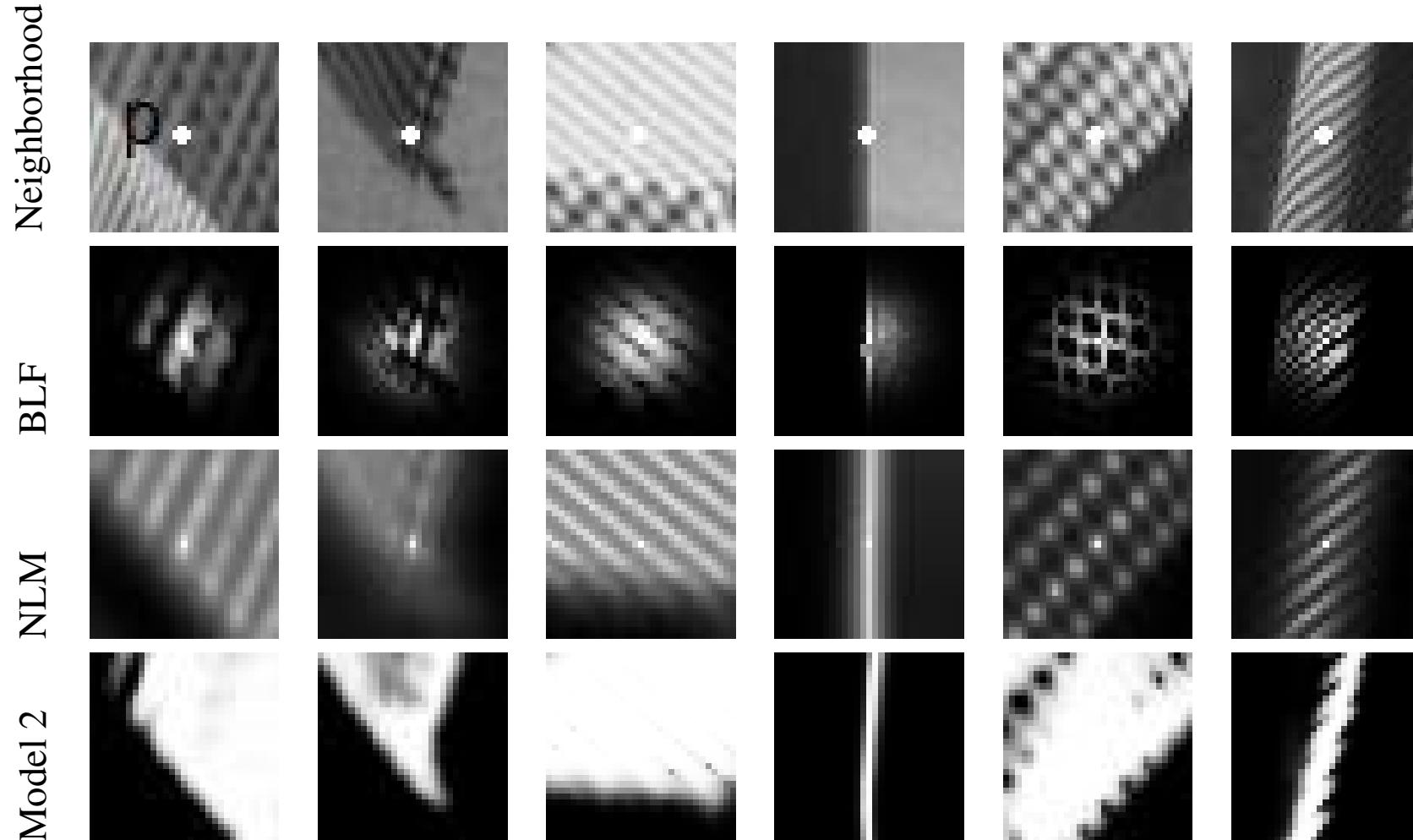
$$d_{KL}(\mathbf{p}, \mathbf{q}) = \frac{1}{2} \left( \text{tr}(\mathbf{C_q}^{-1} \mathbf{C_p}) + (\mu_p - \mu_q)^T \mathbf{C_q}^{-1} (\mu_p - \mu_q) - k - \ln \left( \frac{\det \mathbf{C_p}}{\det \mathbf{C_q}} \right) \right)$$

Resulting kernel

$$w_{pq} \propto \frac{d_{KL}(\mathbf{p}, \mathbf{q})}{2\sigma^2}$$

resulted from a discussion with Rahul Narain (Berkeley University)

# RegCov Smoothing – Smoothing Kernels



# Results



Input

# Results

Model1

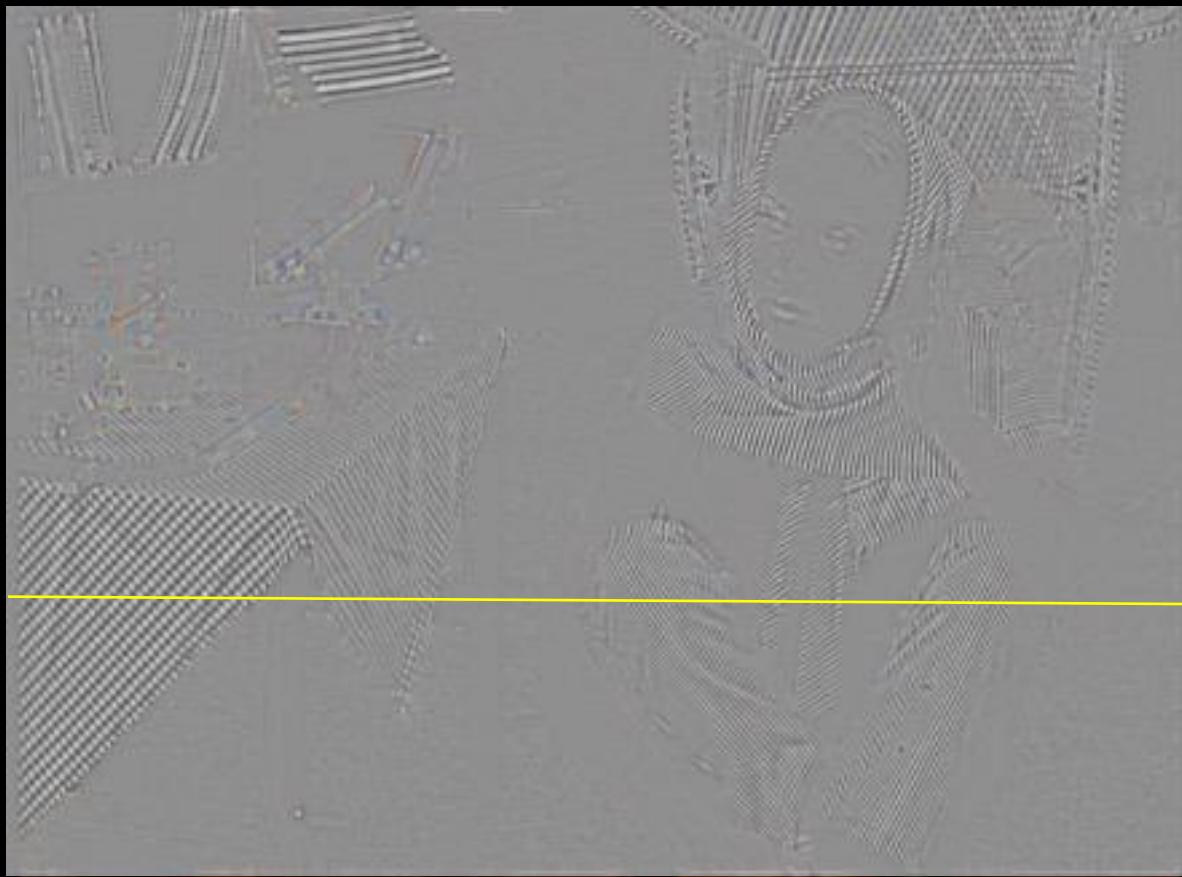


Model2

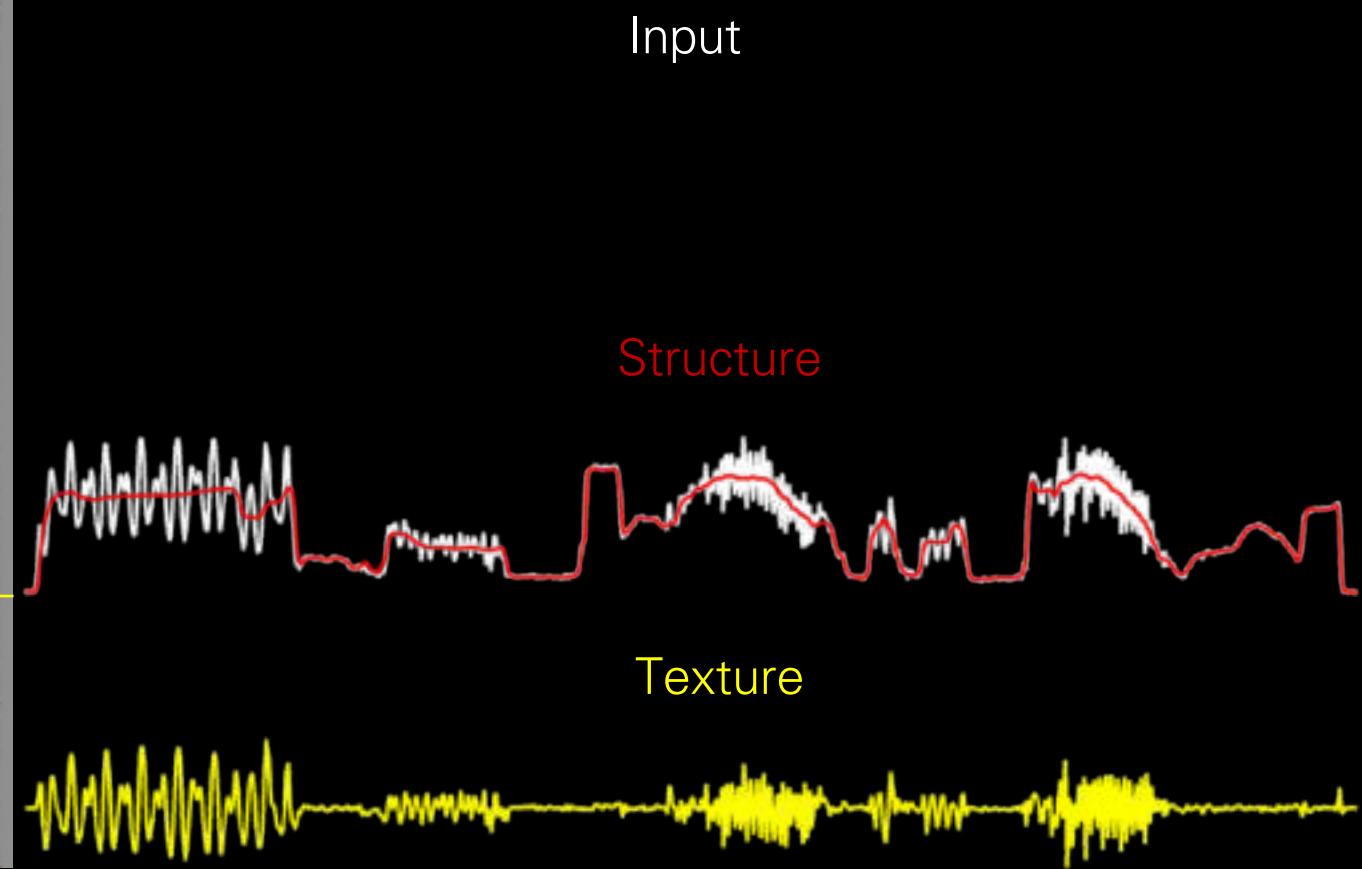


Model3



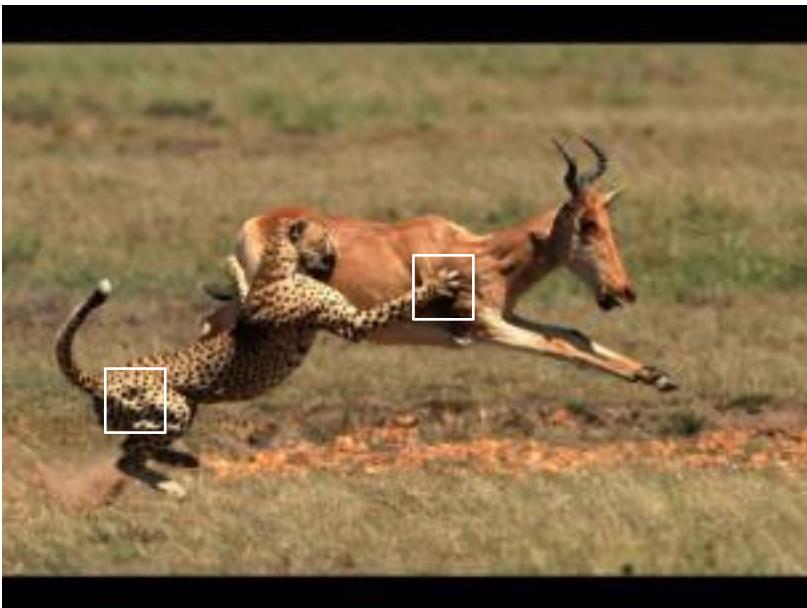


Model Post Texture

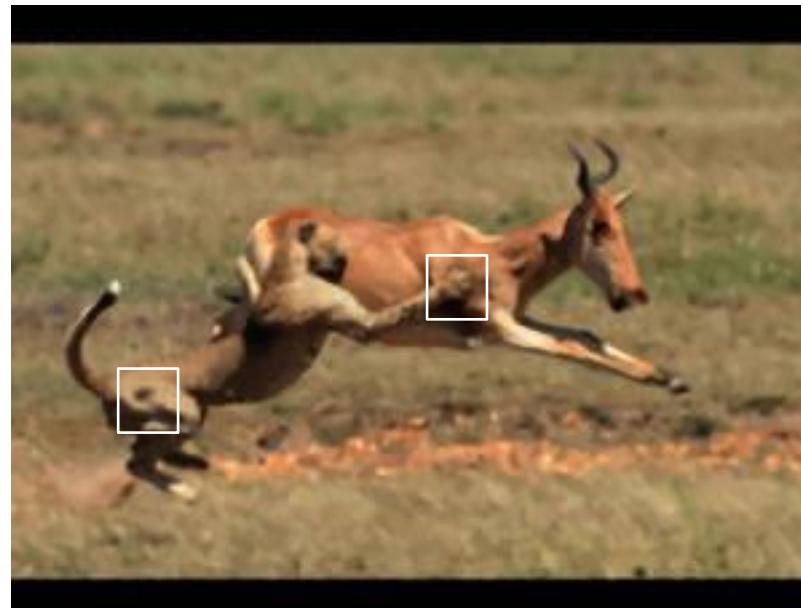


# Results

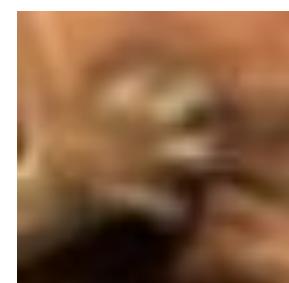
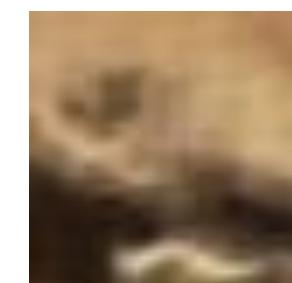
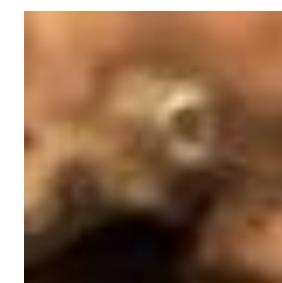
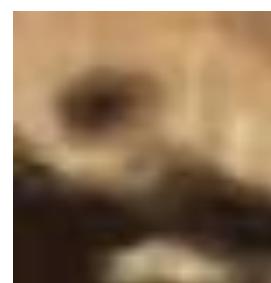
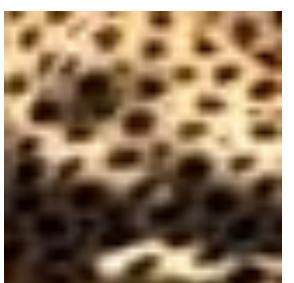
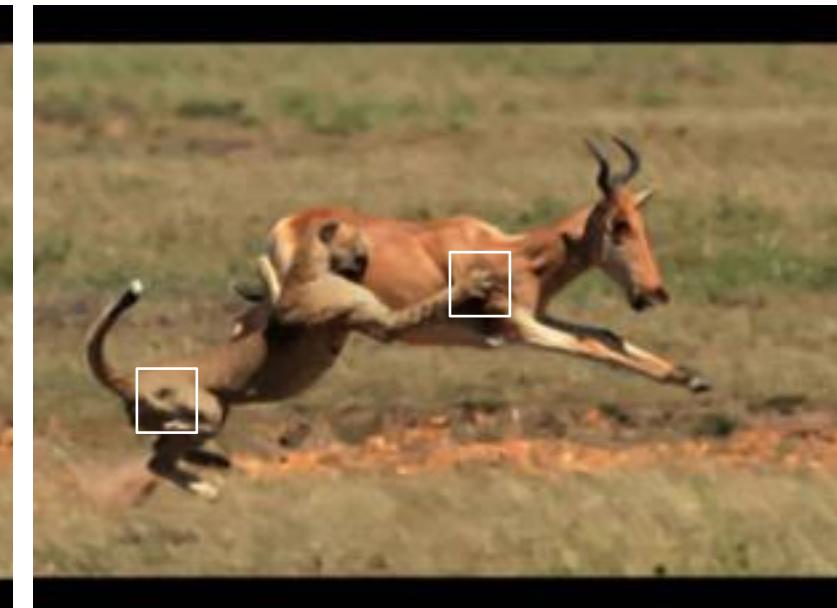
Input



Model2



Model3



# Experimental evaluation



Input

# Experimental evaluation



TV

Rudin et al. 1992

# Experimental evaluation



Bilateral  
Filter

# Experimental evaluation



Envelope  
Extraction  
Subr et al. 2009

# Experimental evaluation



RTV  
Xu et al. 2012

# Experimental evaluation



Model 1

# Experimental evaluation



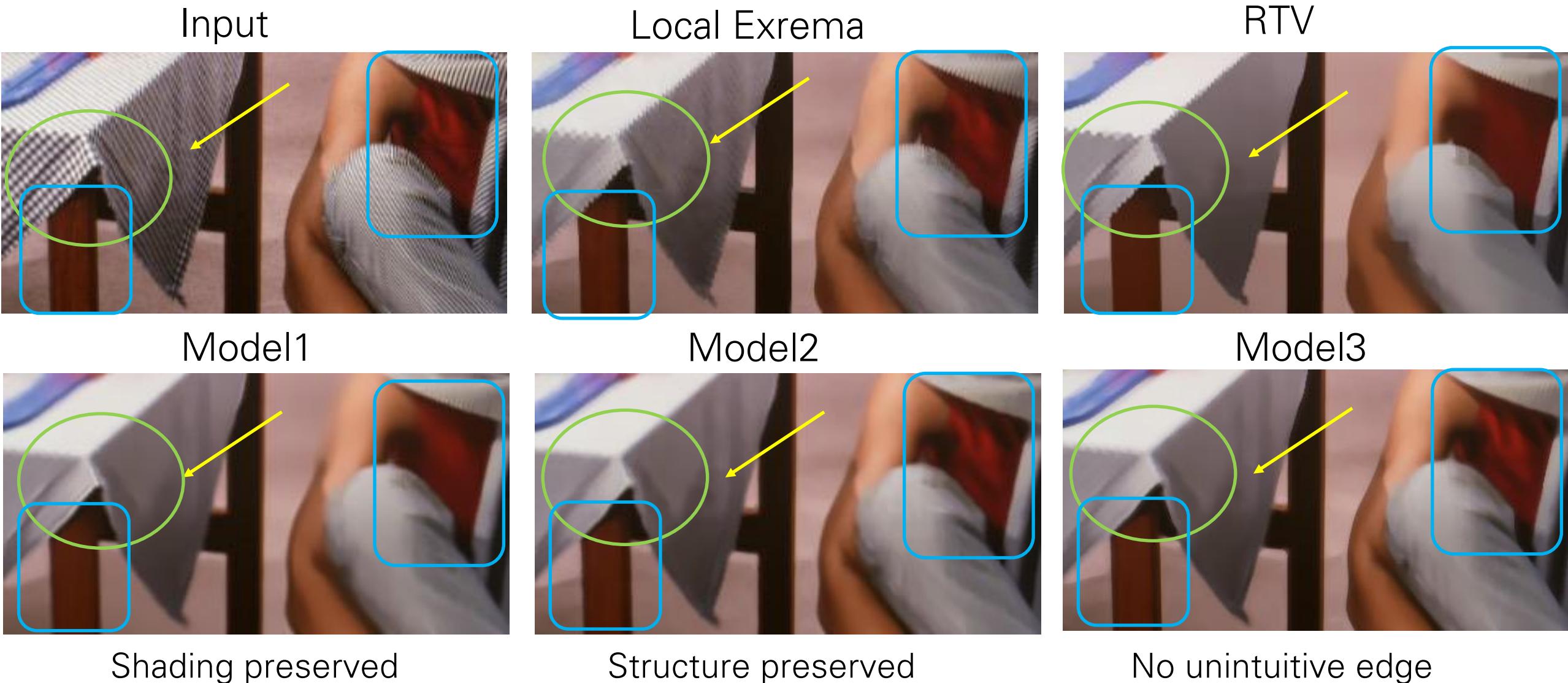
Model 2

# Experimental evaluation

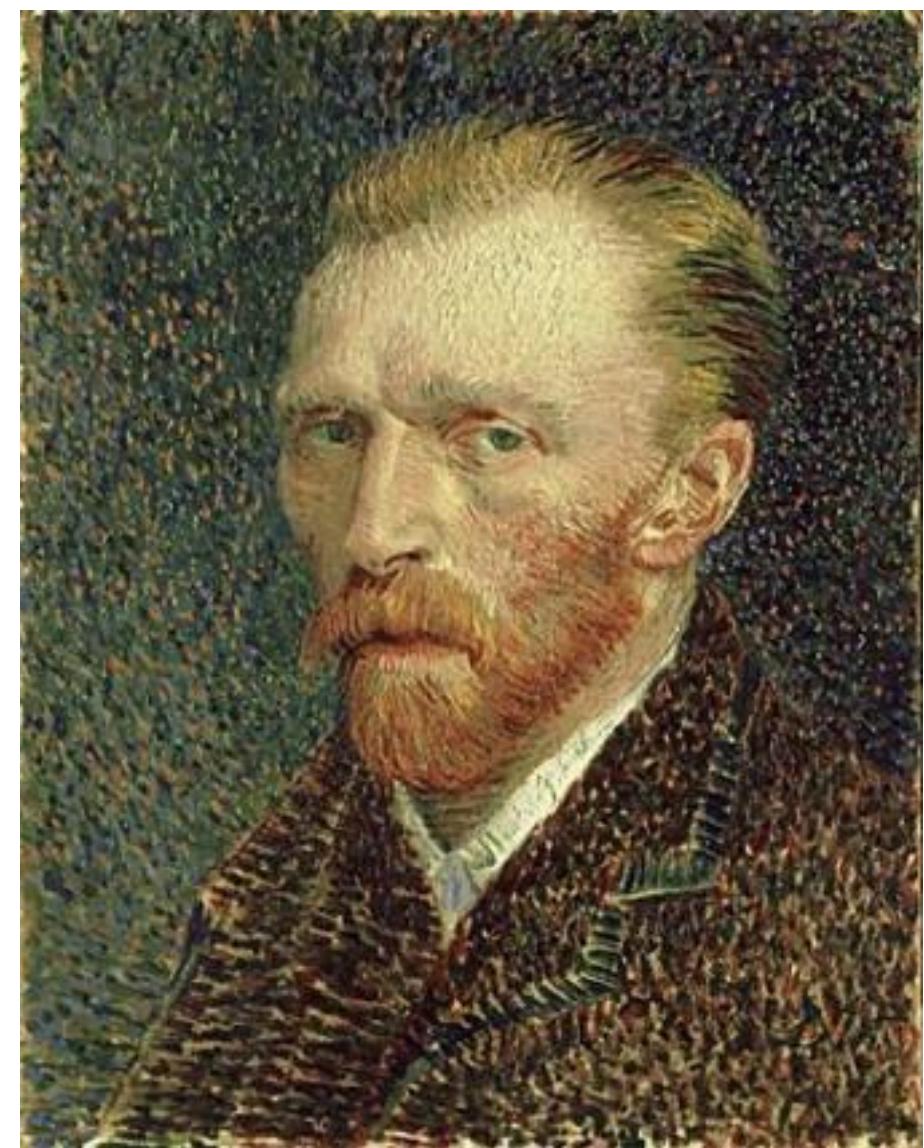


Model 3

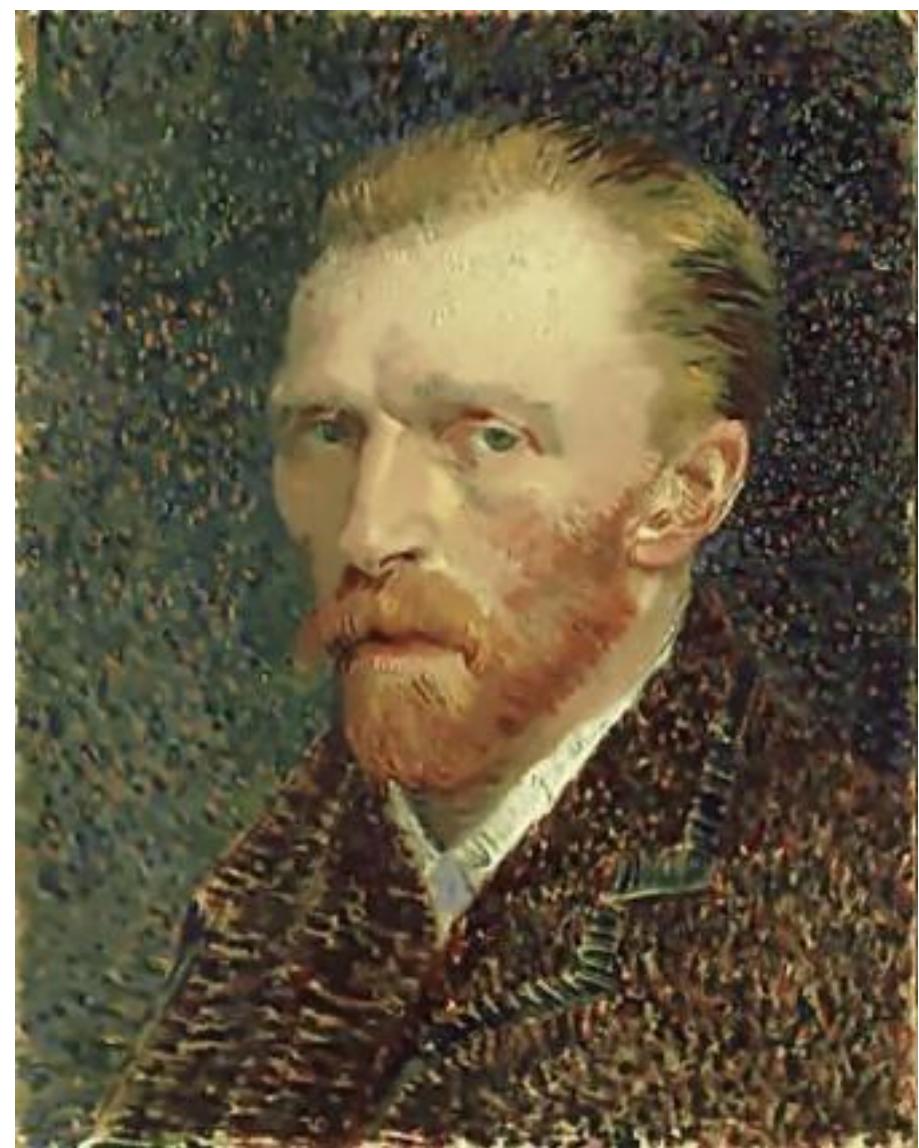
# Experimental evaluation



# Multiscale decomposition



# Multiscale decomposition



$S_1(k = 5)$

# Multiscale decomposition



$S_2(k = 7)$

# Multiscale decomposition



$$S_3(k = 9)$$



# Challenging cases

Input



Model2

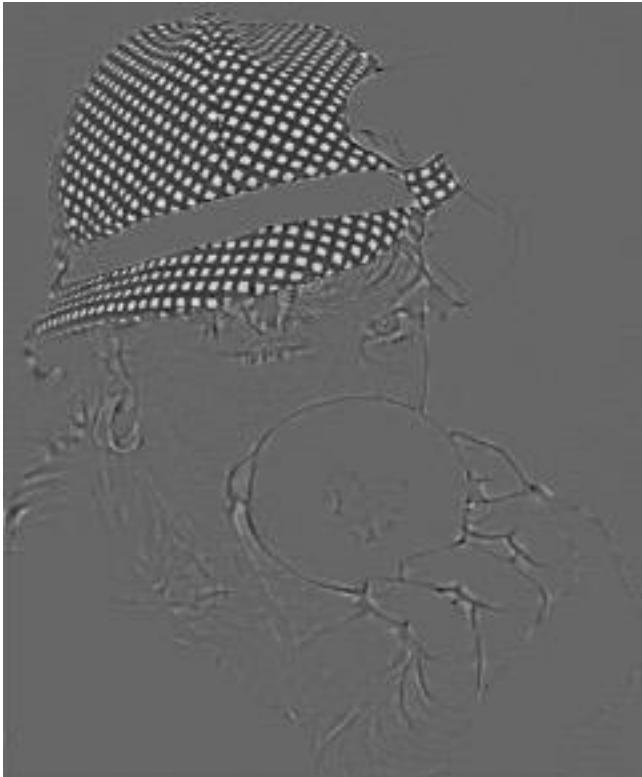


# Challenging cases

Input



Model2 Texture



Model2+Model1



# Edge detection



# Edge detection



# Edge detection

Canny edges of original image



Canny edges of smoothed image



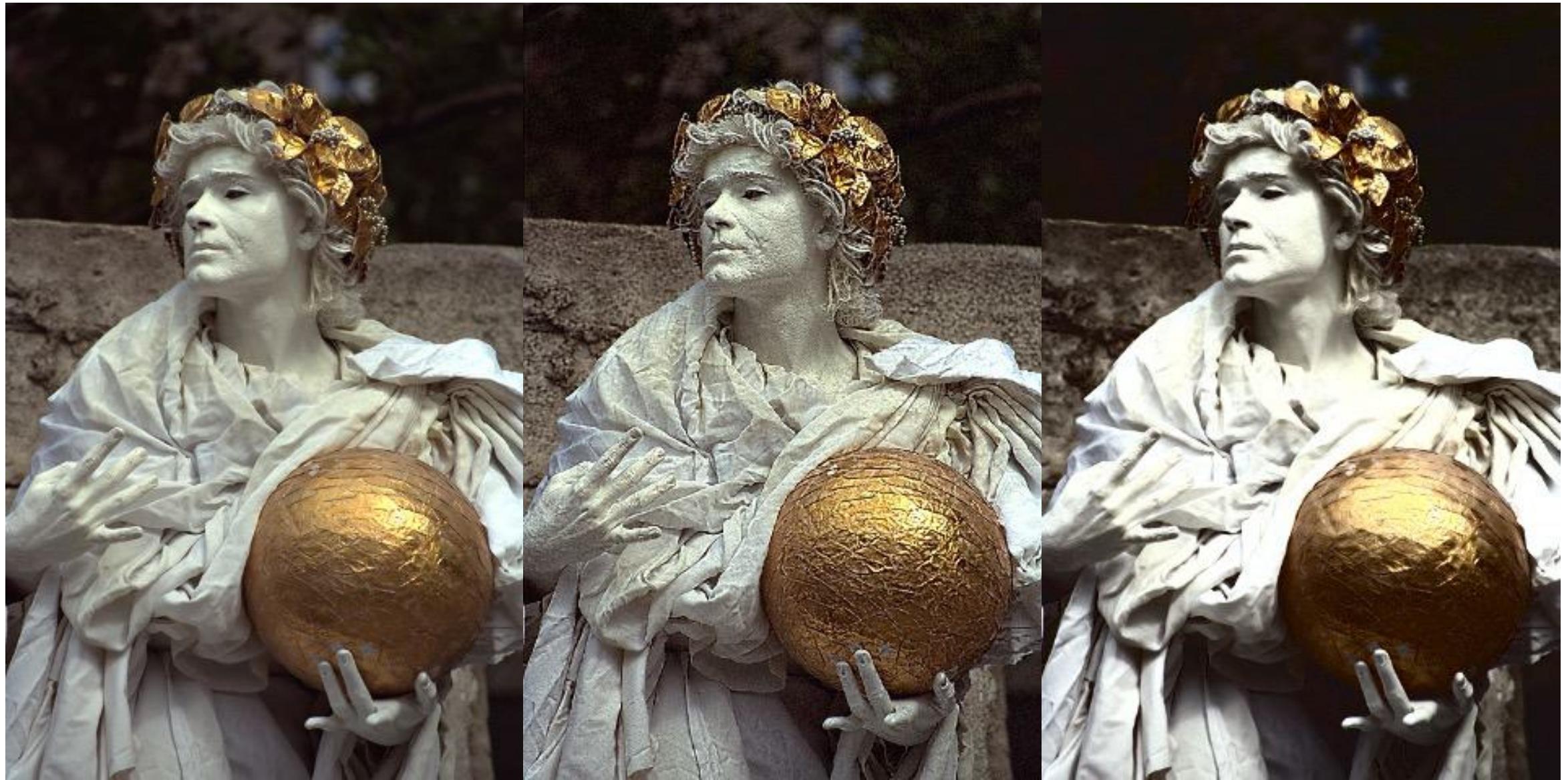
# Image abstraction



# Image abstraction



# Detail boosting



# Image composition



# Inverse Halftoning



Input  
halftone image

# Inverse Halftoning



Smoothed image  
(Model 2)

# Inverse Halftoning



Smoothed image  
(Model 2 + Shock filter)

# Inverse Halftoning



Model2+Shock Filter



Input halftone image

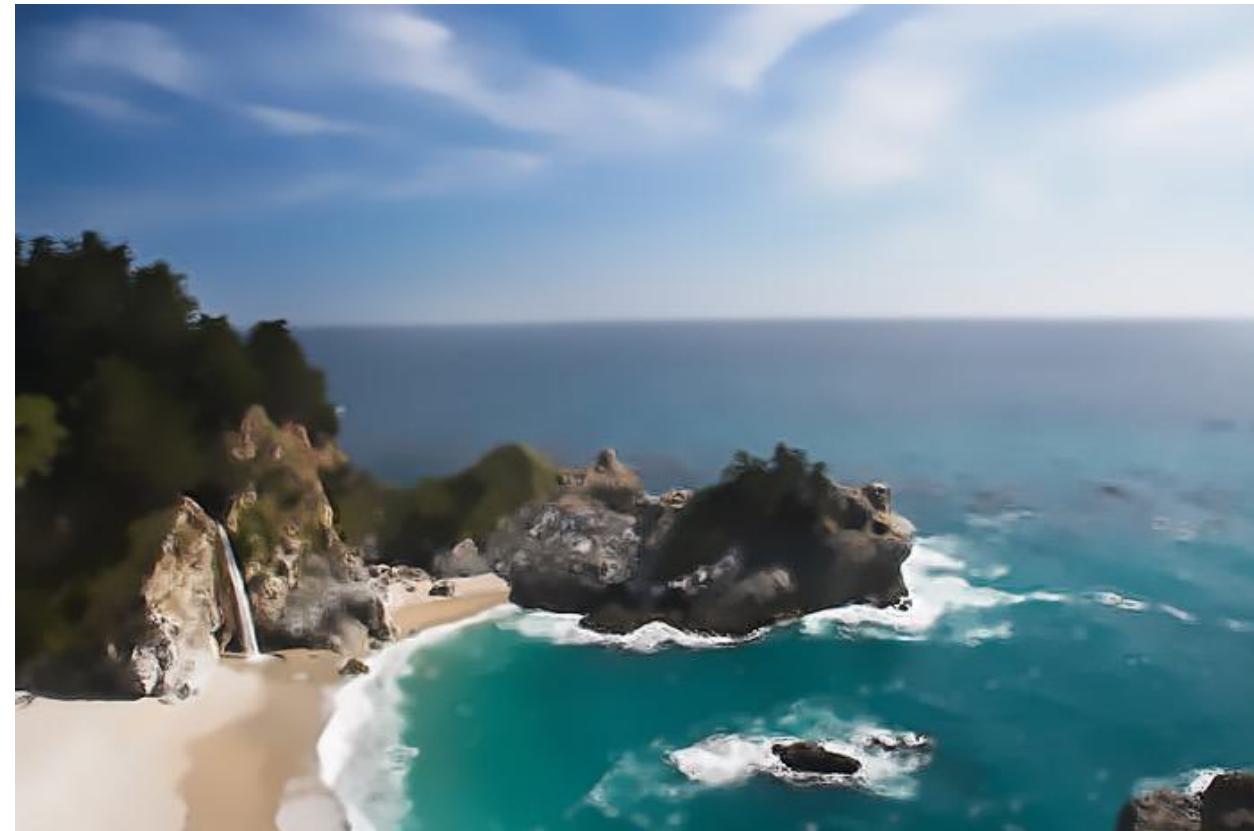


Kopf and Lischinski 2012

# Image Retargeting

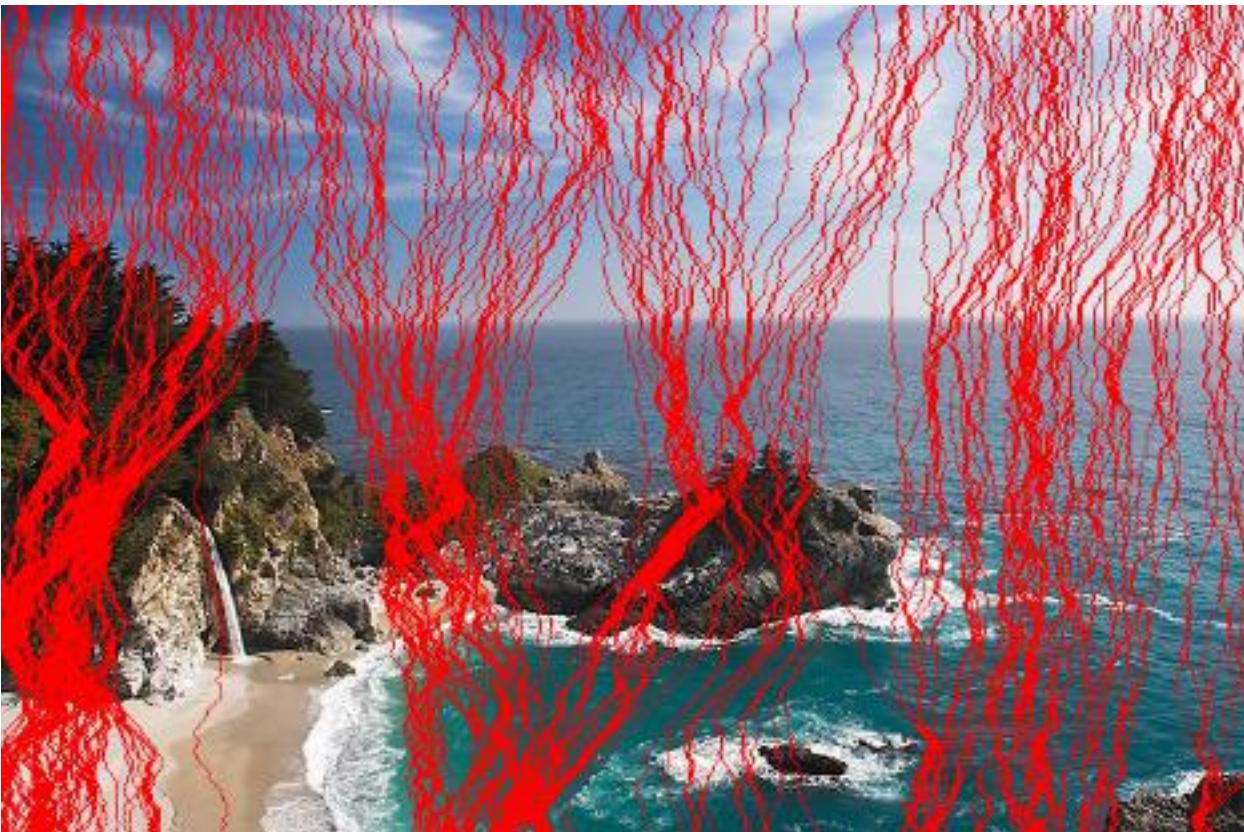


Input image



Smoothed image

# Image Retargeting



Input image  
Extracted Seams



Smoothed image  
Extracted Seams

# Image Retargeting



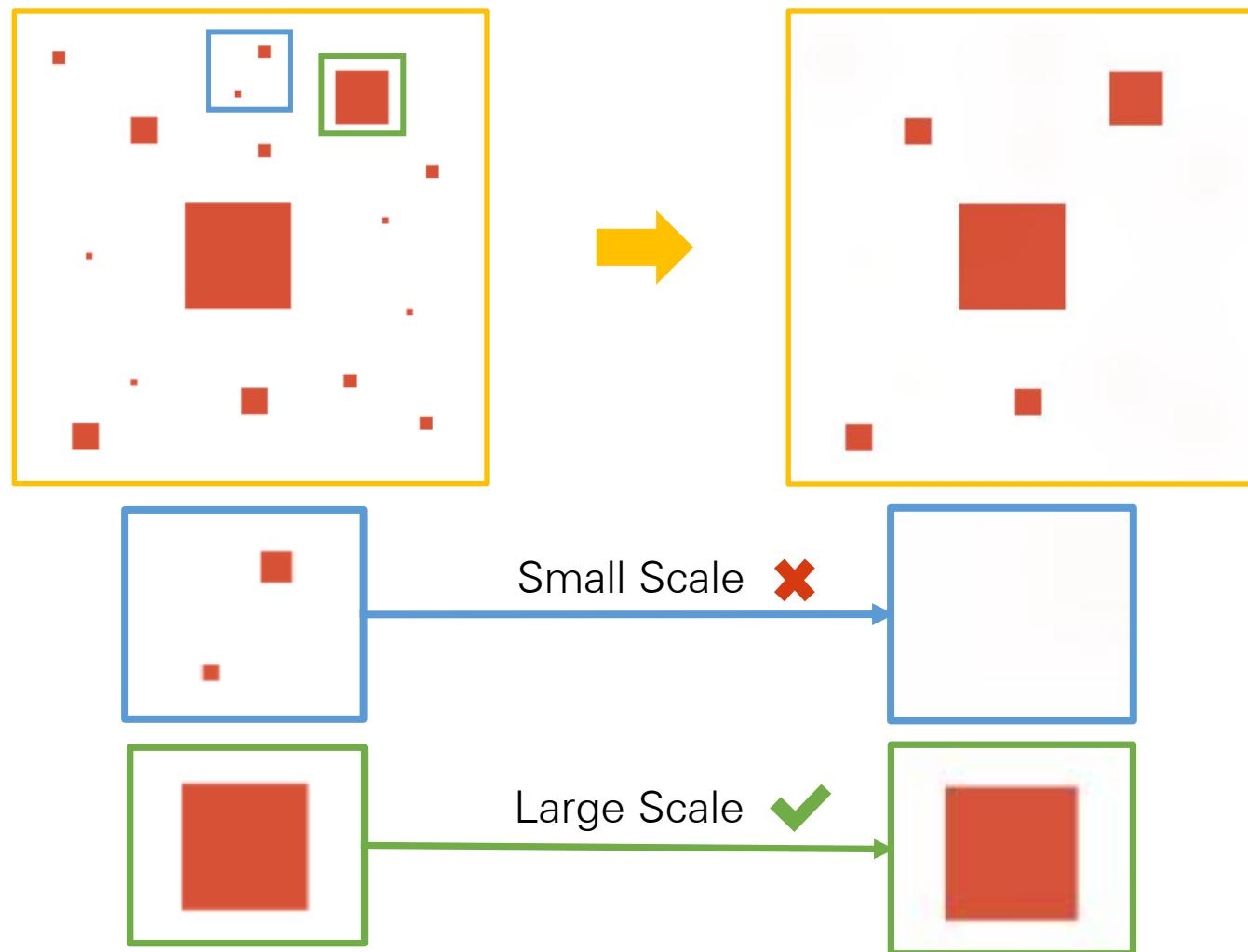
Input image  
Retargeting result



Smoothed image  
Retargeting result

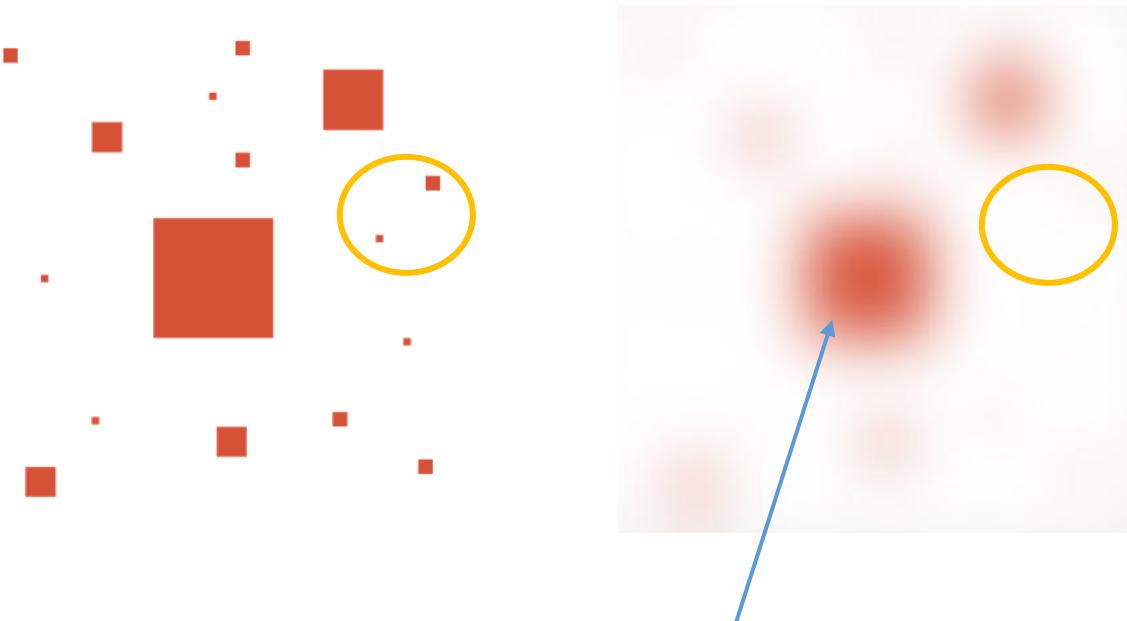
# Rolling Guidance Filter

# Scale-Aware Filtering

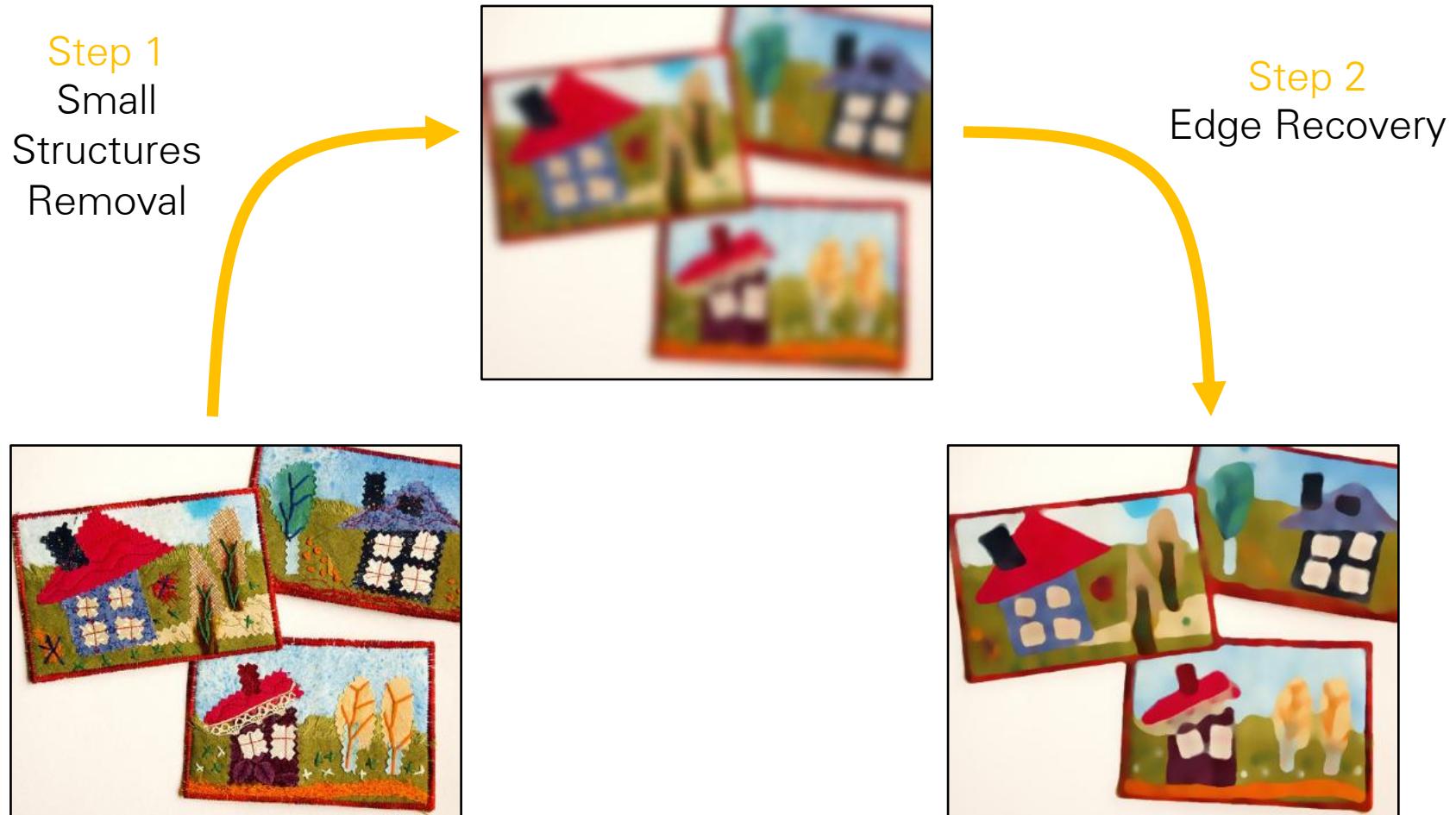


# Main Idea

- Scale Space Theory [Lindeberg, 1994]:
  - An object of size  $t$ , will be largely smoothed away with Gaussian filter of variance  $t^2$ .

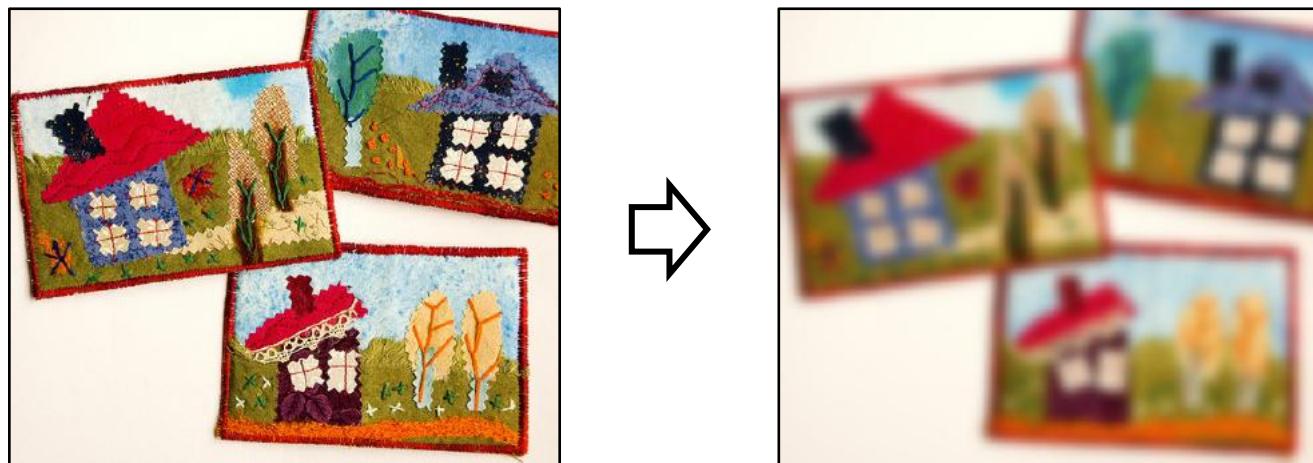


# RGF: A scale-aware Filter



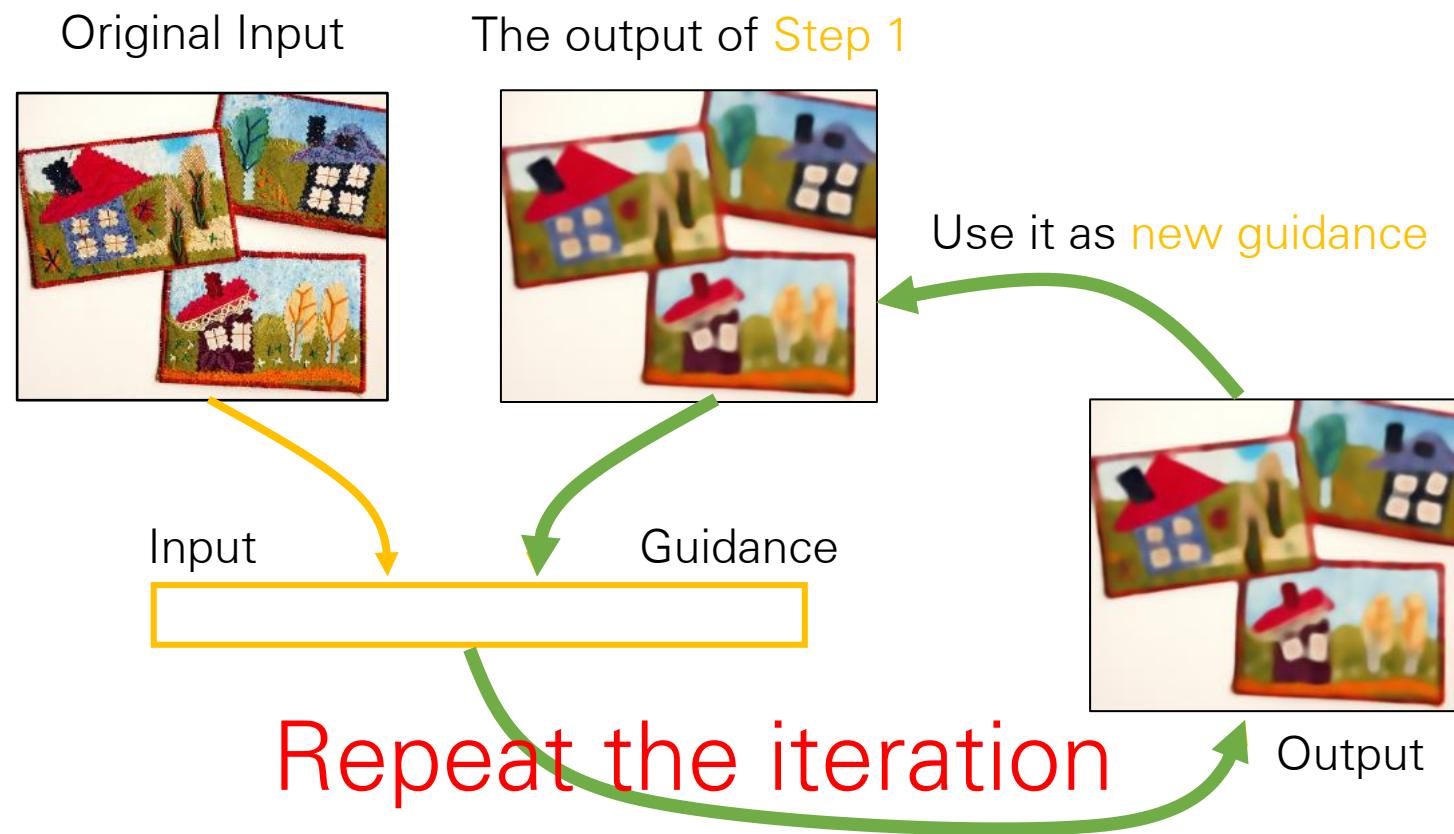
# Step 1: Small Structures Removal

Gaussian Filter

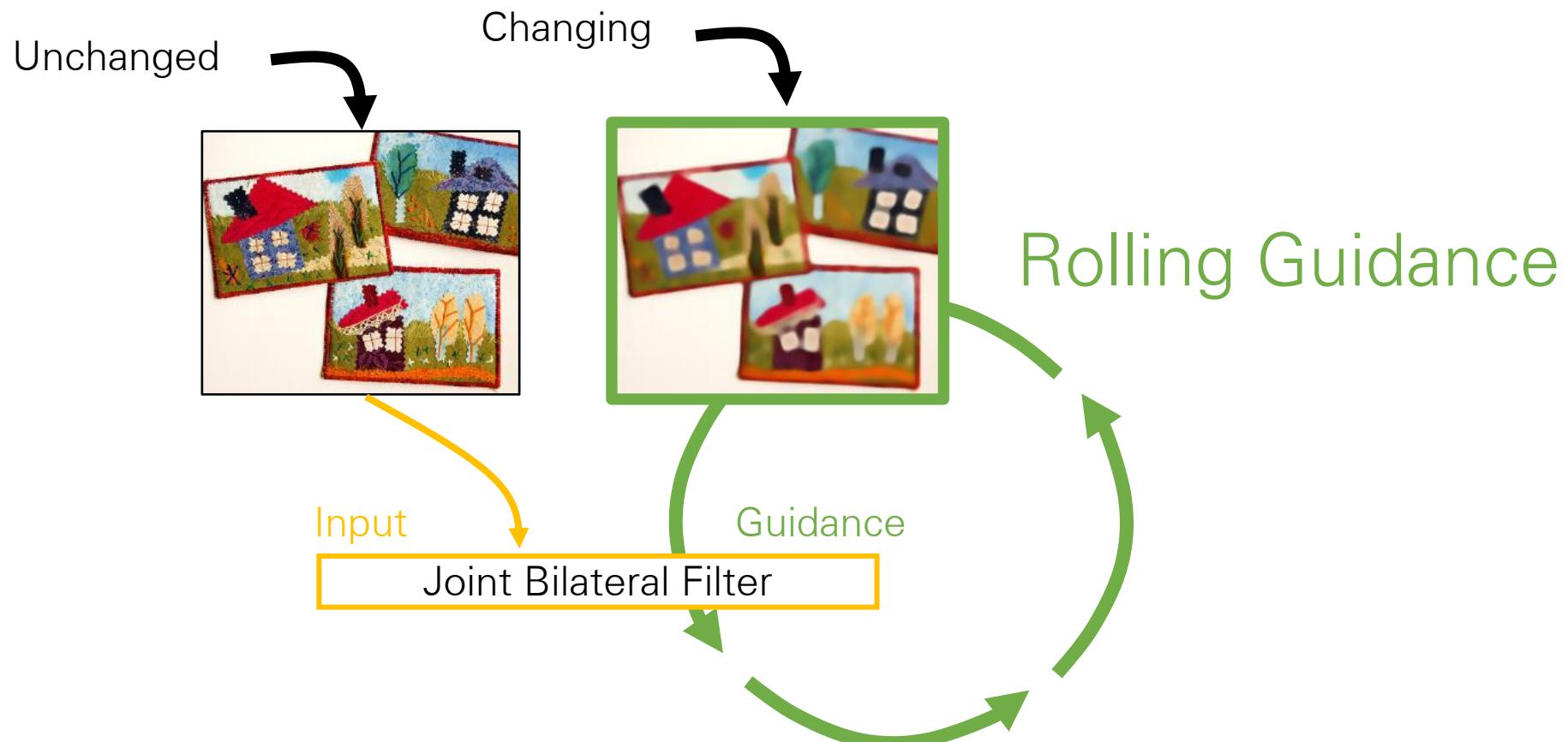


# Step 2: Edge Recovery

- A rolling guidance



# Rolling Guidance



# Rolling Guidance



Guidance for the 1st  
iteration

# Rolling Guidance



Guidance for the 2nd  
iteration

# Rolling Guidance



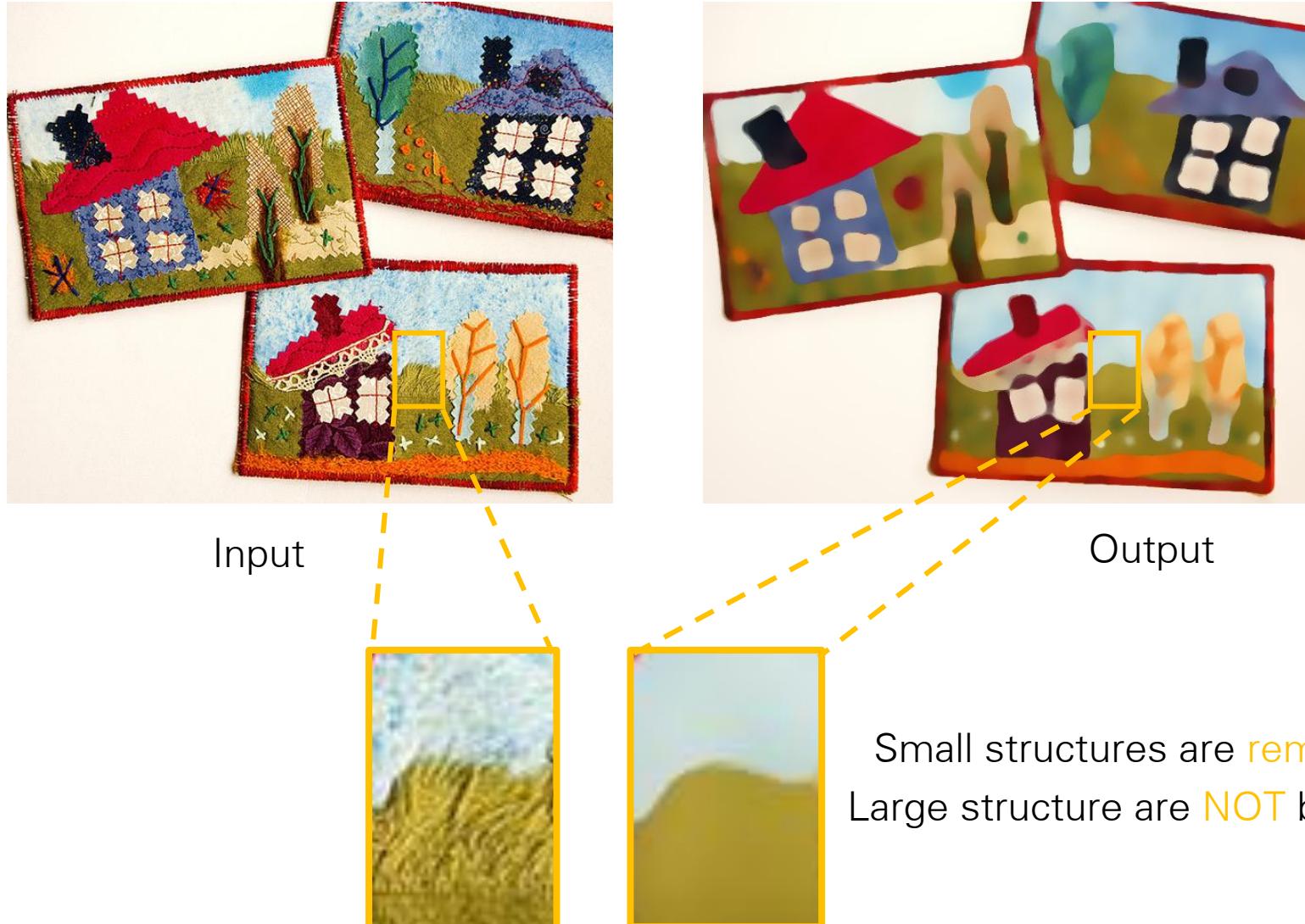
Guidance for the 3rd  
iteration

# Rolling Guidance



Guidance for the 5th  
iteration

# Rolling Guidance

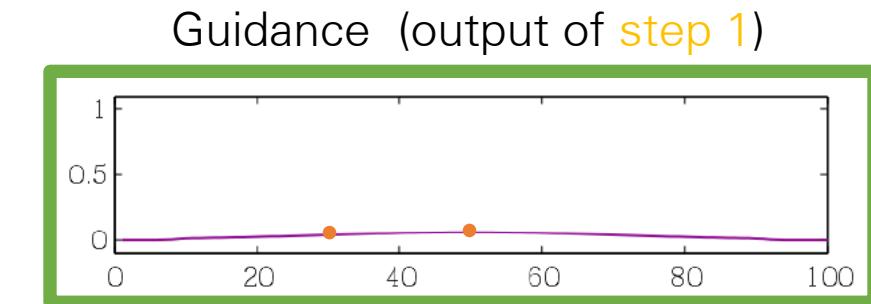
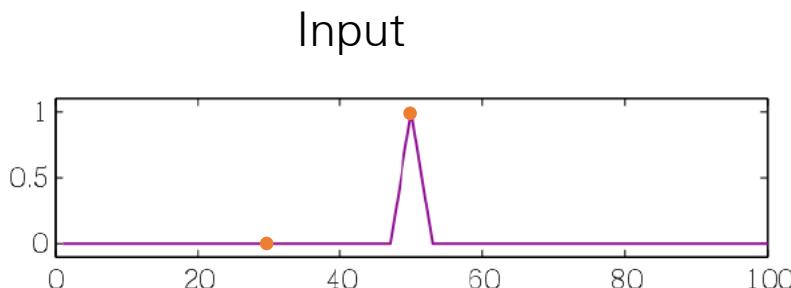


# Implementation

Rolling Guidance Filter (RGF) has only **1 line** of code

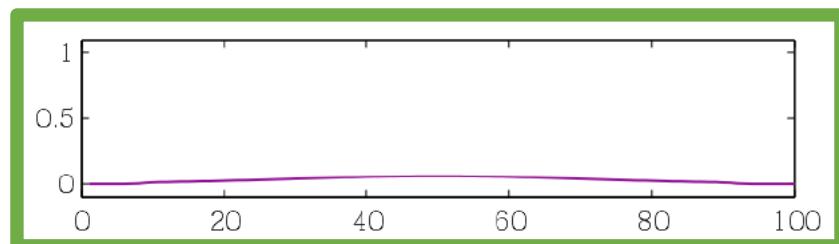
```
1 Mat rollingGuidanceFilter(Mat im, float scale, int iter){  
2     Mat res = im.mul(0);  
3     while(iter--) res = bilateralFilter(im,res,scale,SIGMA_R);  
4     return res;  
5 }
```

# Small Structure



$$J^{t+1}(p) = \frac{1}{K_p} \sum_{q \in N(p)} \exp \left( -\frac{\|p - q\|^2}{2\sigma_s^2} \right)$$

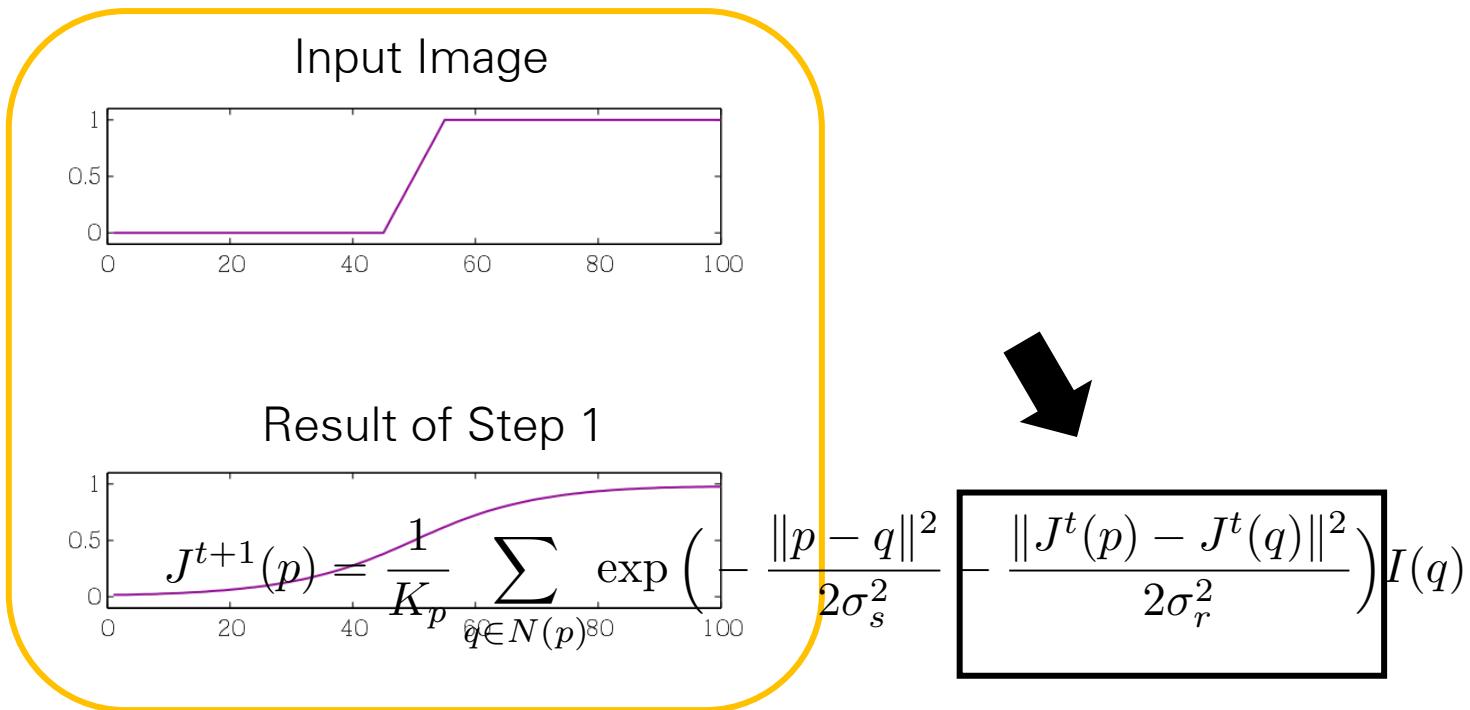
Joint Bilateral Filter



It becomes a Gaussian filter

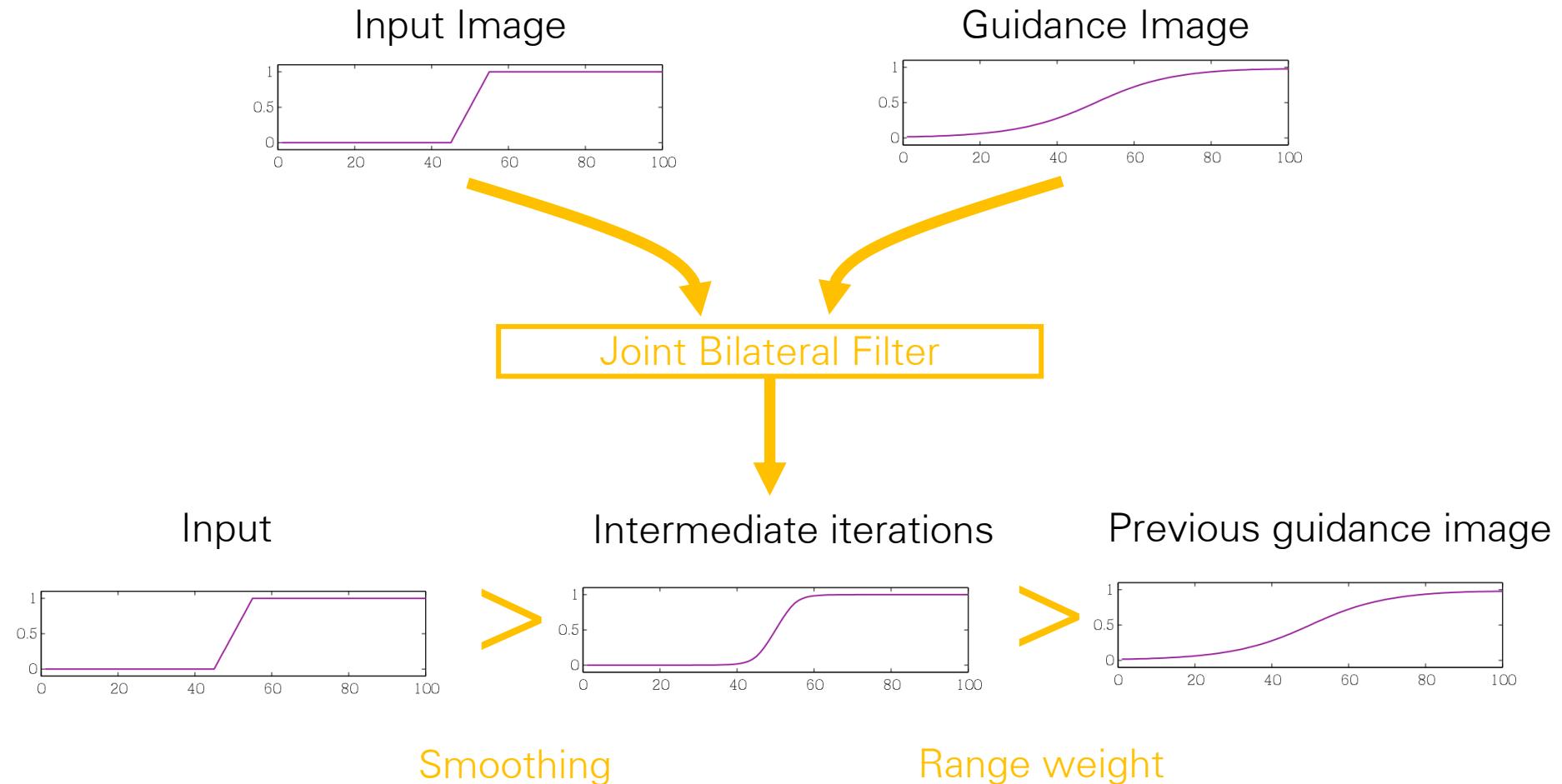
Same

# Large Structure



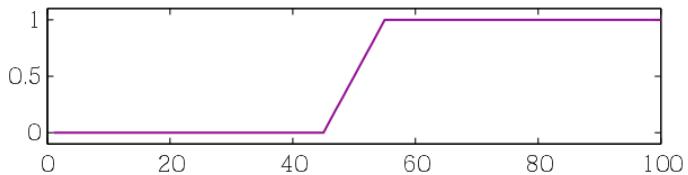
Due to this range weight  
It generates sharper results than Gaussian!

# Processing

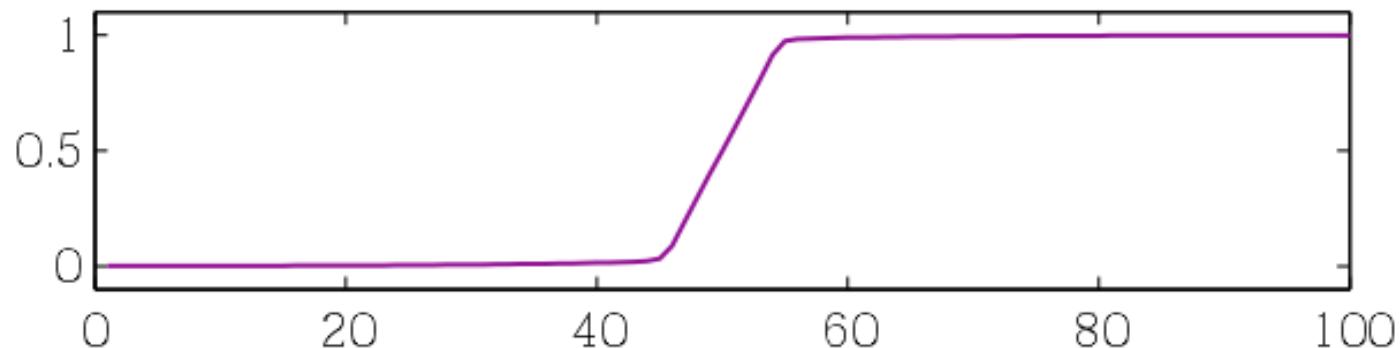


# Processing

Input Image



Guidance Image



3<sup>rd</sup> Iteration

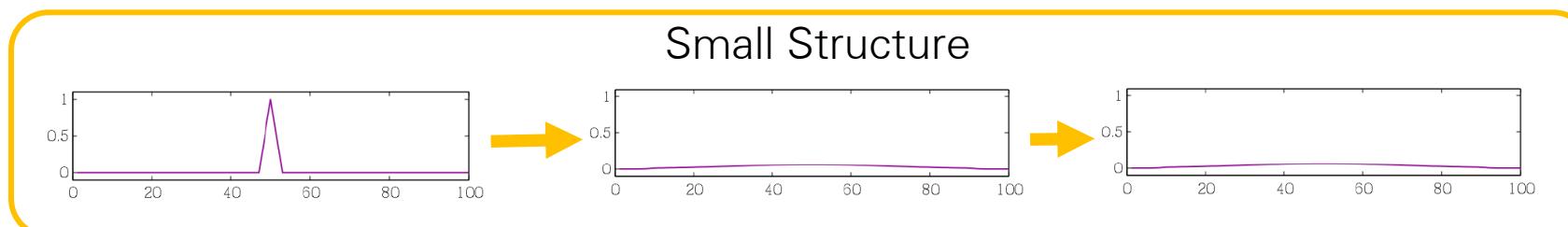
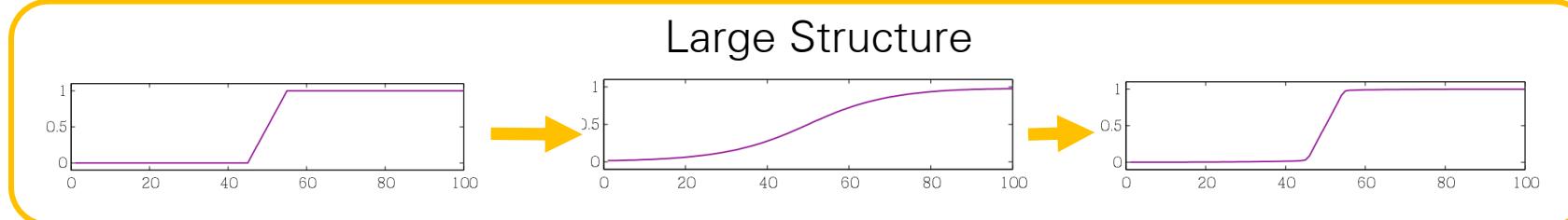
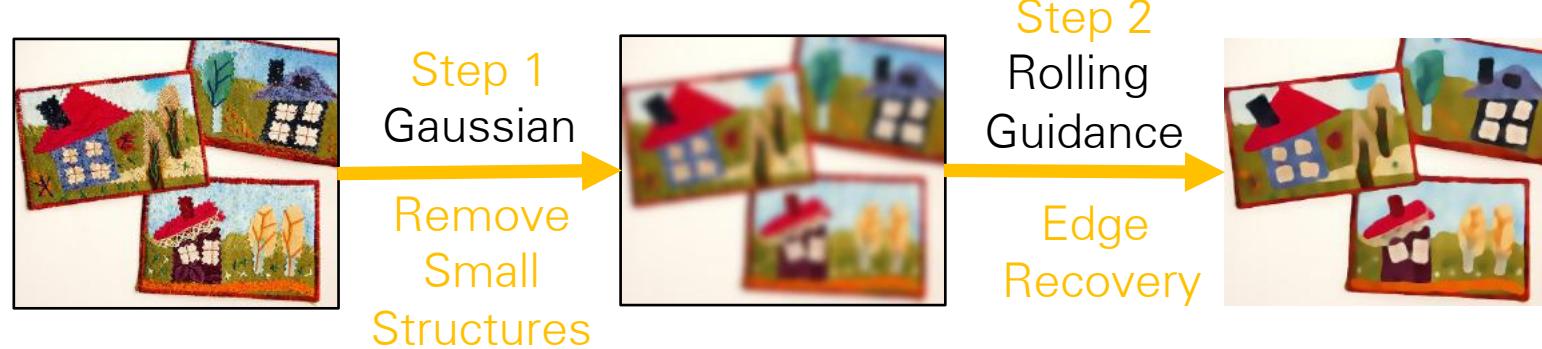
# Large Structure



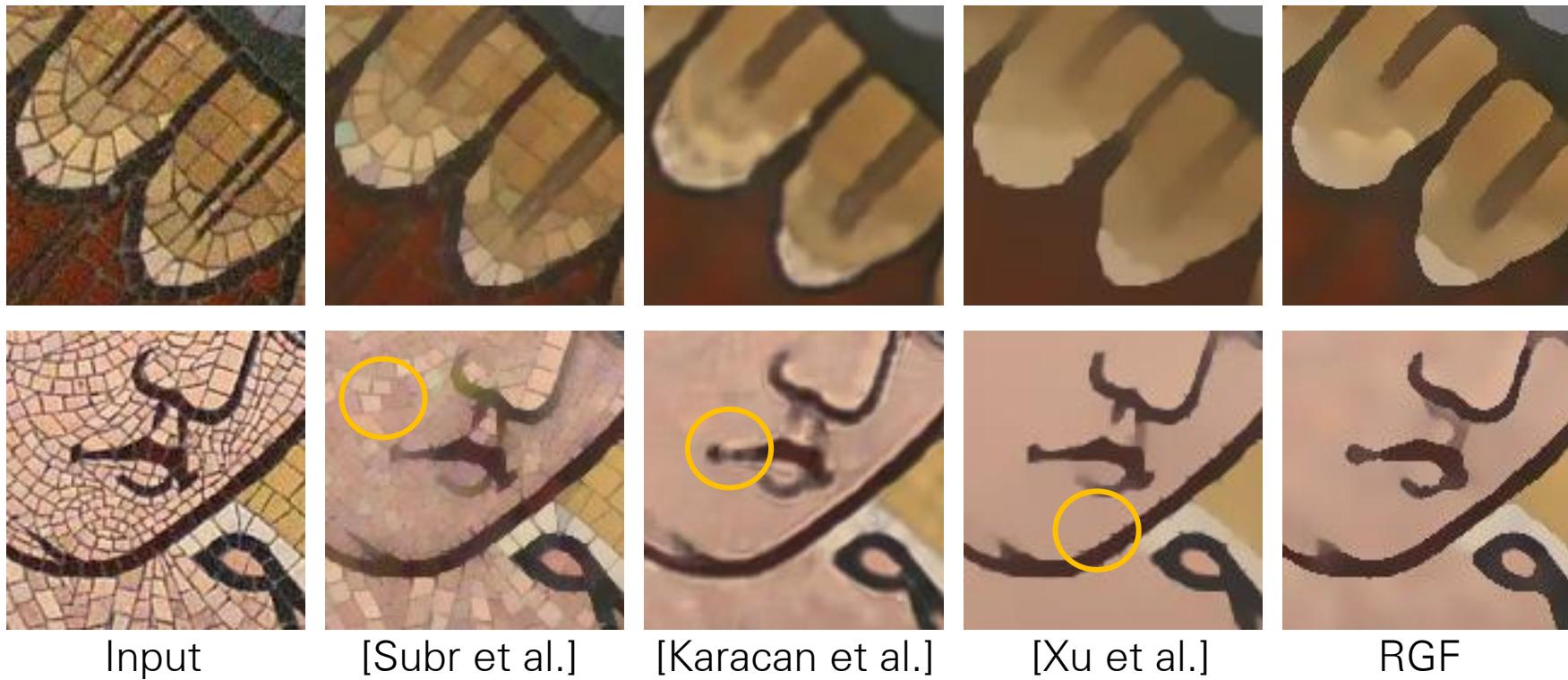
Take-home message

Rolling guidance recovers an edge as long as it still exists in the blurred image after Gaussian smoothing.

# Rolling Guidance Filter



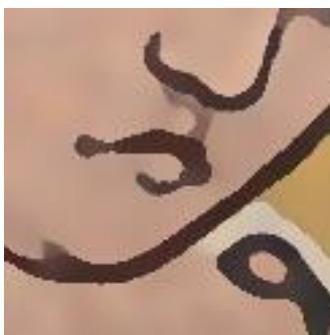
# Result Comparison



# Performance Comparison



Input



RGF  
2013]

For 4 Megapixel Image

2

seconds

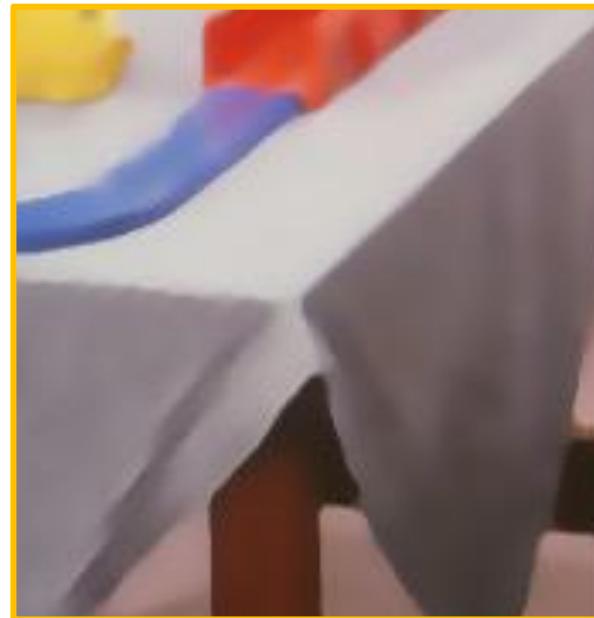
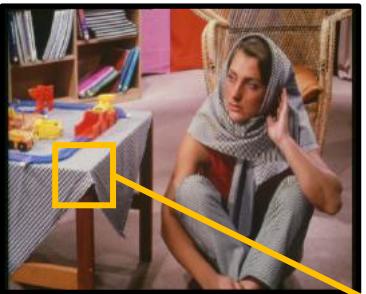
# Performance Comparison

Algorithms	Time (seconds/Megapixel)
Local Extrema [Subr et al., 2009]	95
RTV [Xu et al., 2012]	14
Region Covariance [Karacan et al., 2013]	240
RGF	0.05(Real-time)

# Texture Removal



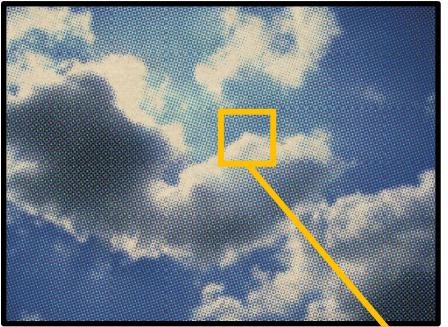
# Texture Removal



# Halftone Image



# Halftone Image



# Boundary detection

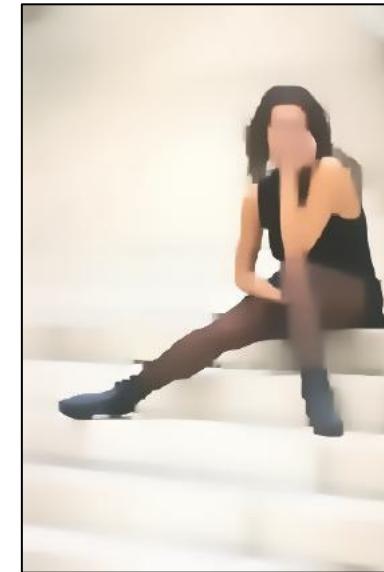
Input



Boundary Detection



Filtered Input



Boundary Detection



# Multi-Scale Filtering



= 30

determine the scale.

# Limitations

- Sharp corners could be rounded
  - It is because sharp corner presents high frequency change.
  - In other words, sharp corners are small-scale structures.

# Recap

- Filtering plays a key role for many applications.
- Filtering by taking into account image content generally gives better results.

# **Next Lecture:**

## Gradient-domain image processing