THE UNIVERSITY OF TEXAS AT DALLAS

Image Processing: Filtering II

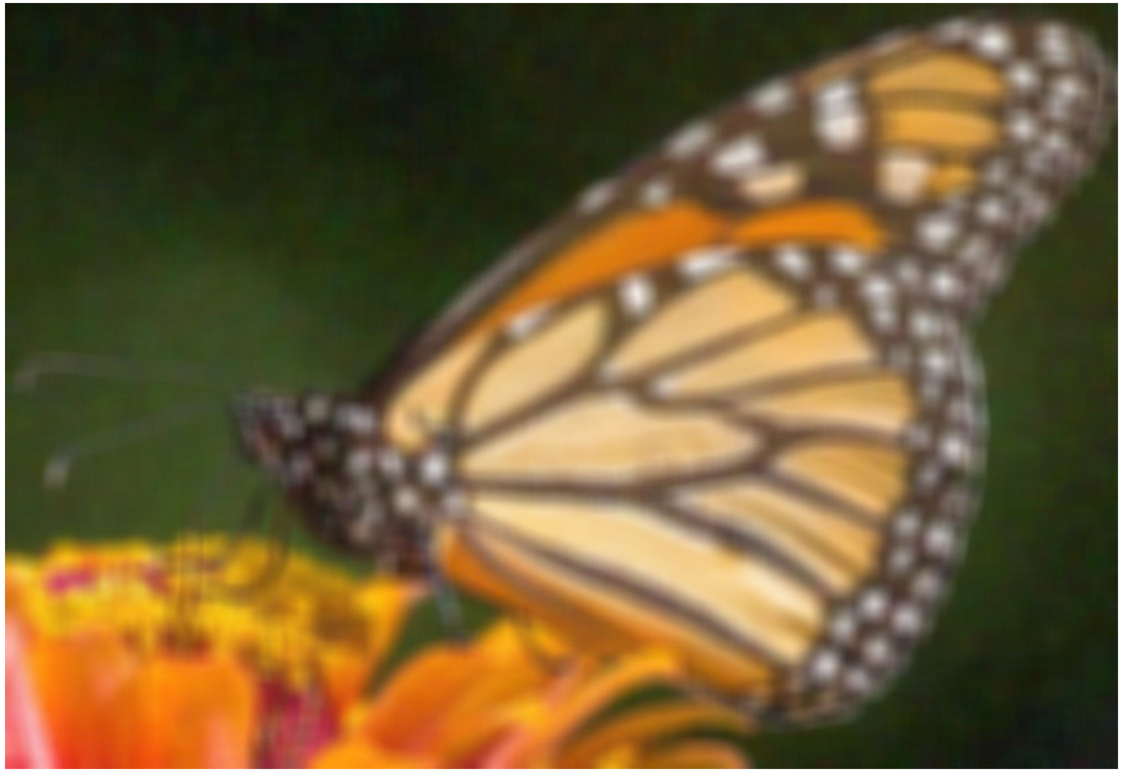
CS 6384 Computer Vision

Professor Yapeng Tian

Department of Computer Science

Many slides in this lecture were inspired or adapted from Ioannis (Yannis) Gkioulekas.

Filtered Image (Gaussian)

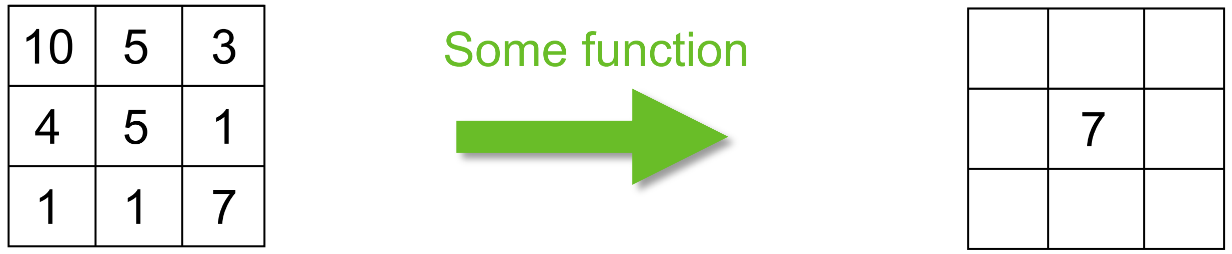


Noisy Image

Question: How to handle blurry artifacts and preserve image edges in the filtered image?

Recap: Image Filtering

Modify the pixels in an image based on some function of a local neighborhood of each pixel



Local image data

Modified image data

Let f be the image, w be the $(2n+1)\times(2n+1)$ kernel weights and h be the filtered output image

$$h[u,v] = \sum_{k=-n}^{n} \sum_{l=-n}^{n} w[k,l] f[u+k, v+l]$$

cross-correlation == convolution, when the kernel is isotropic

Recap: Image Filtering Process

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

kernel

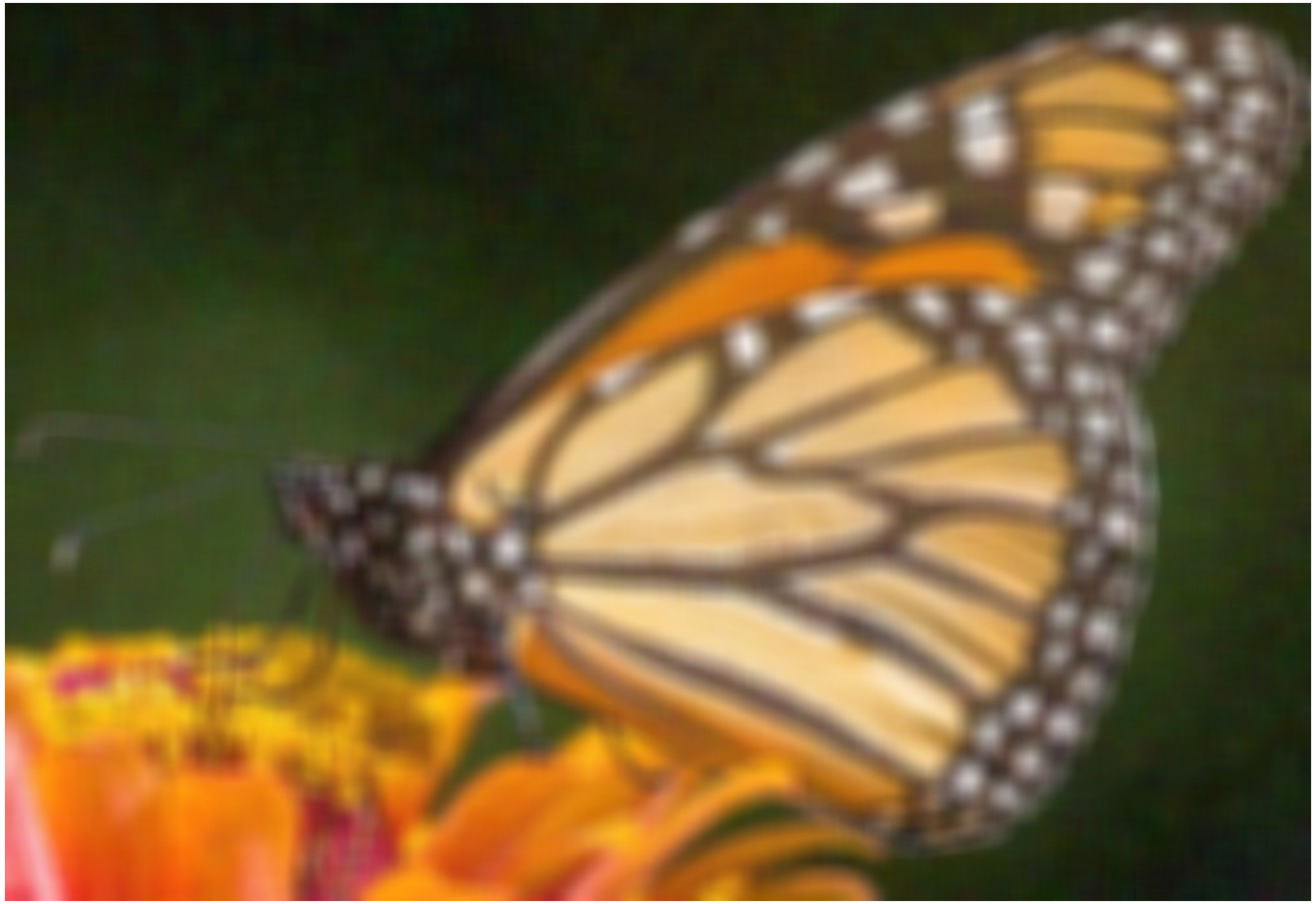
Noisy Image

Apply the filter to every pixel

Recap: Image Filtering Process

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

kernel

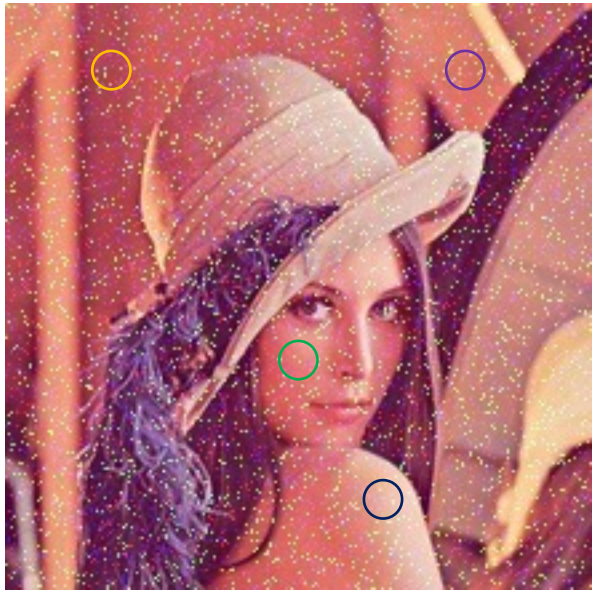


Filtered Image

Apply the filter to every pixel

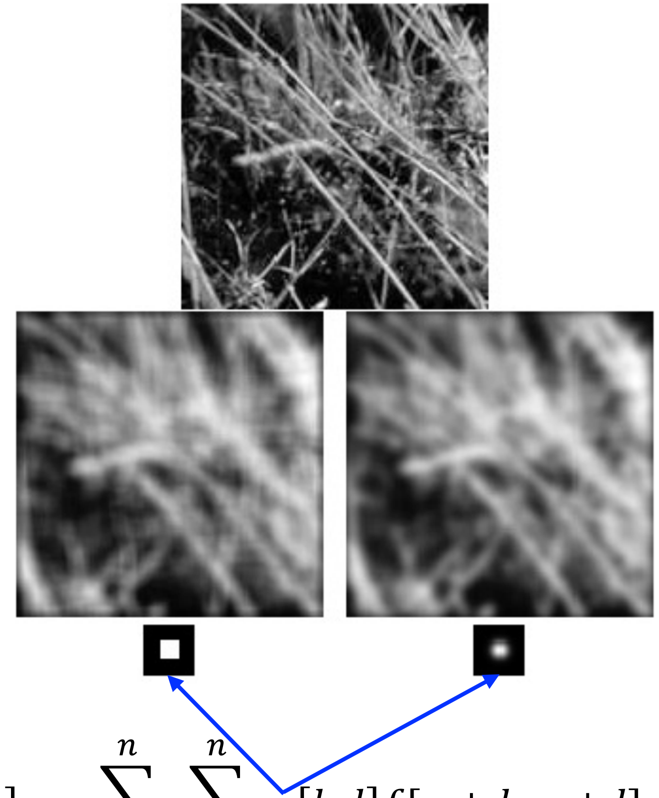
Recap: Image Prior: Local Smoothness

- Local natural image regions are typically smooth or uniform
- The overall structures or texture of a natural image often has a more subtle and gradual variation than image noise



- Image pixels in a small window (e.g., 5x5) usually are similar
- Noise values are dramatically changing at arbitrary directions
- Due to noises, a noisy image have higher local variations than the clean image

Recap: Local Smoothness with Mean vs Gaussian filtering



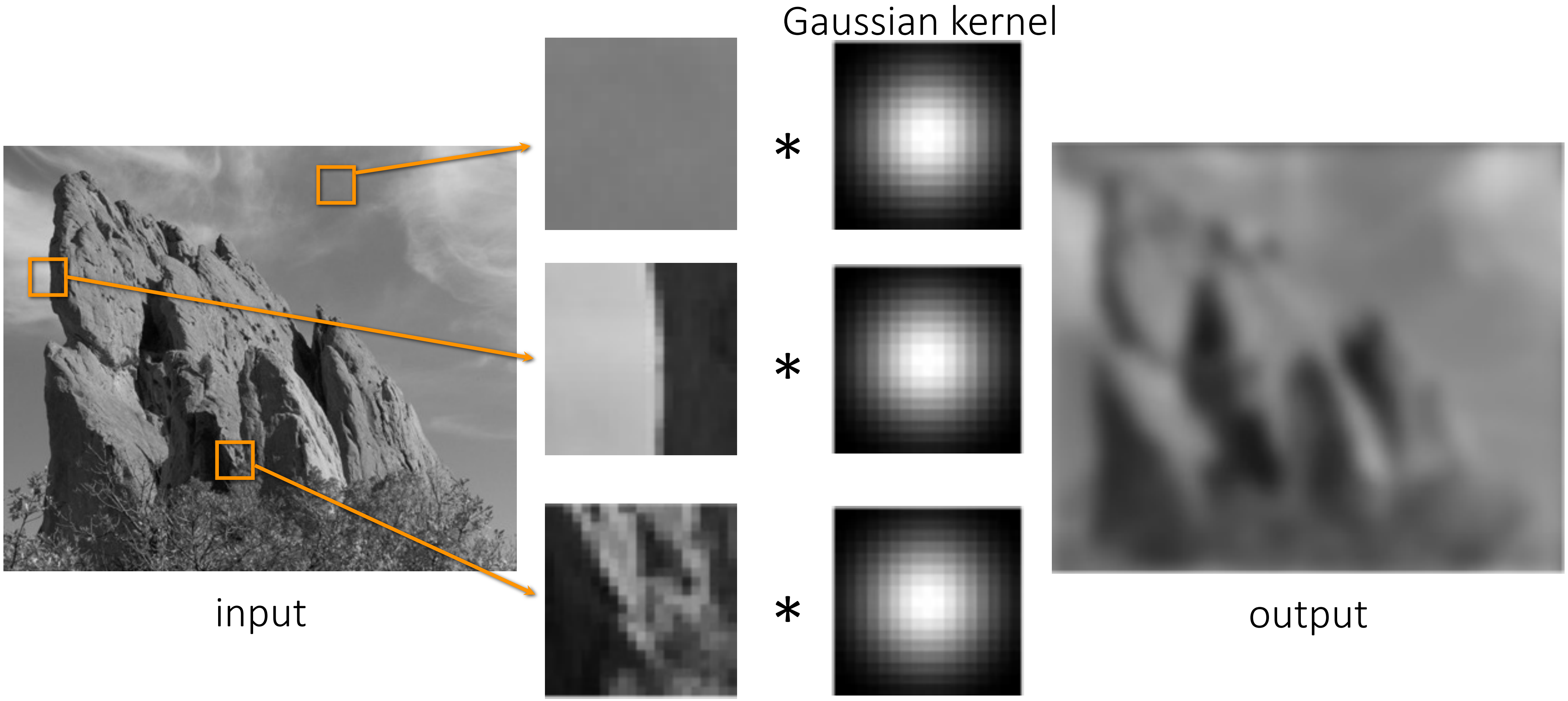
$$h[u,v] = \sum_{k=-n}^{n} \sum_{l=-n}^{n} w[k,l] f[u+k, v+l]$$

Both mean and Gaussian utilize local smoothness prior

- Mean filter assumes all pixels in a local window are equally important
- Gaussian filter assumes pixels that are closer to the target pixel are more important

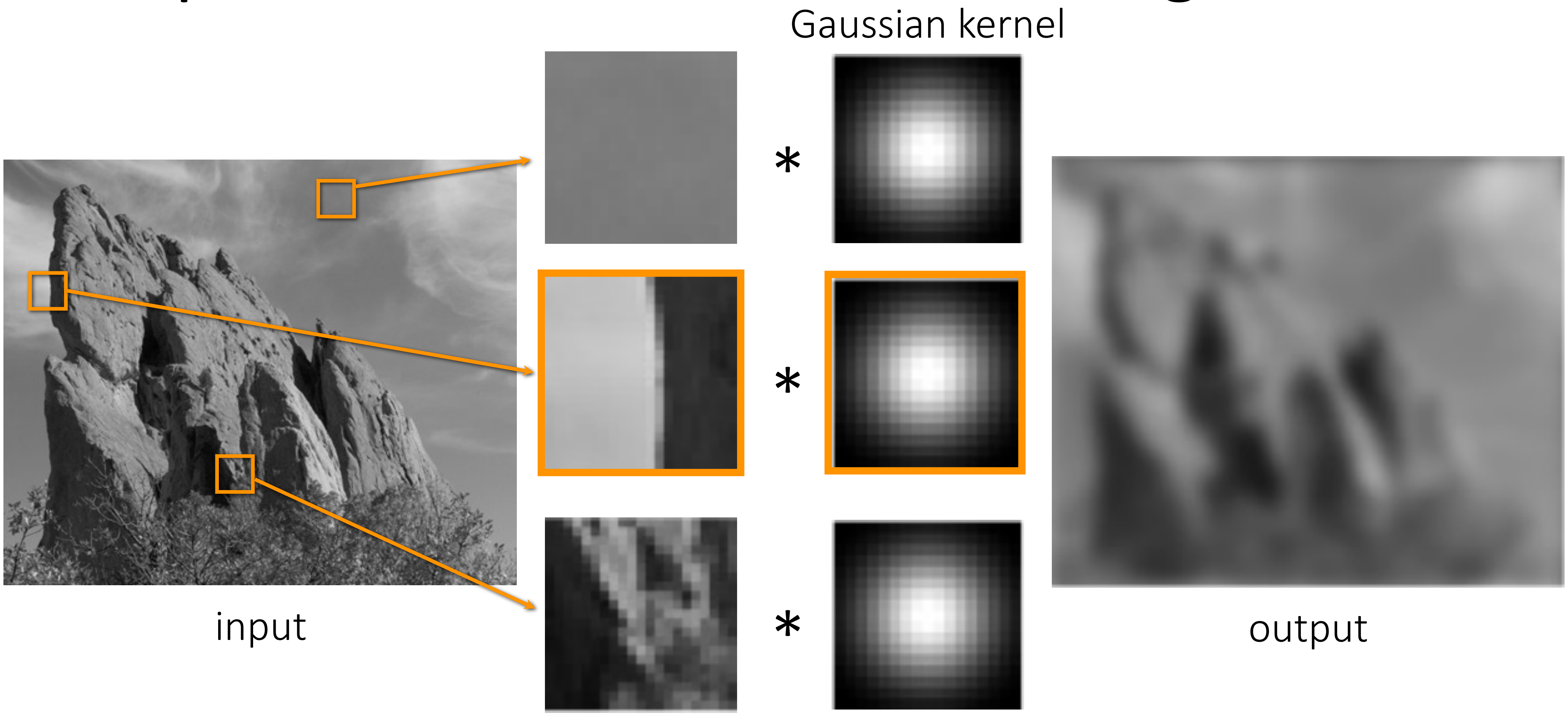
We need to design a better kernel w for improving filtering results.

The problem with Gaussian filtering



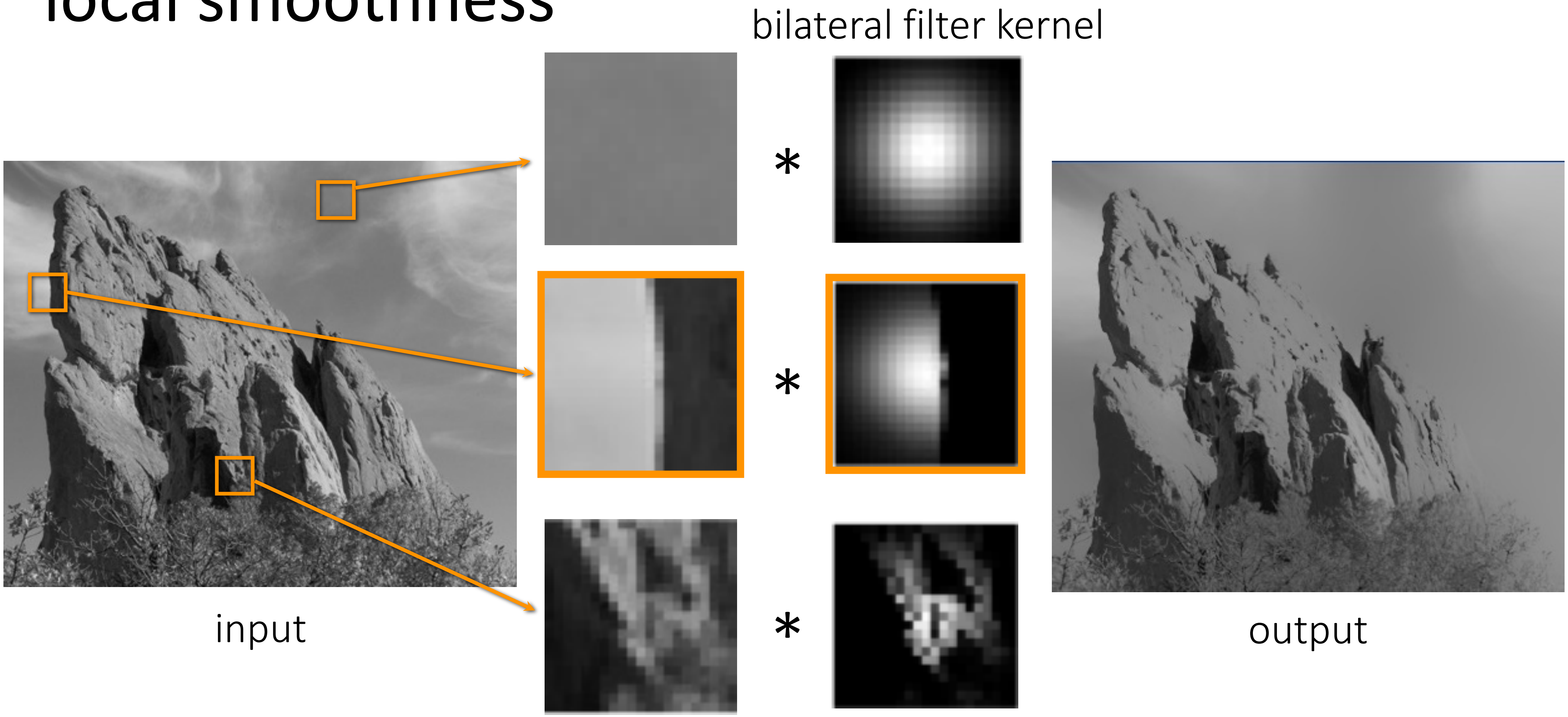
Why is the output so blurry?

The problem with Gaussian filtering



Blur kernel averages across edges

The bilateral filtering solution: Edge-preserving local smoothness



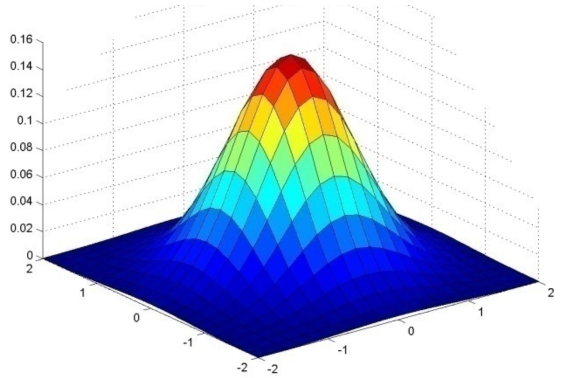
Do not blur if there is an edge! How does it do that?

Bilateral filtering

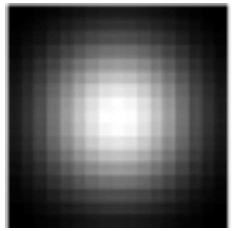
$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$

Bilateral filtering

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$



Spatial weighting



$$g[k,l] = \frac{1}{2\pi{\sigma_s}^2} \exp(-\frac{(k^2 + l^2)}{2{\sigma_s}^2})$$

σ_s

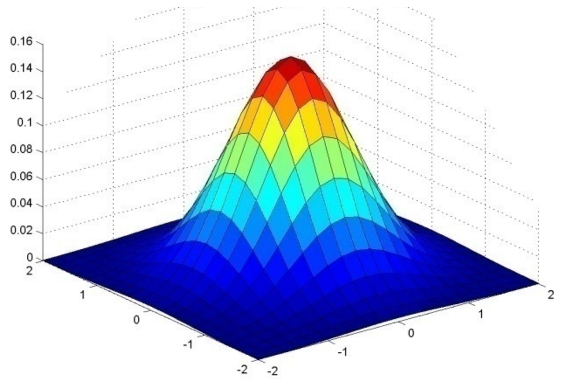
Assign a pixel a large weight if: 1) it's nearby

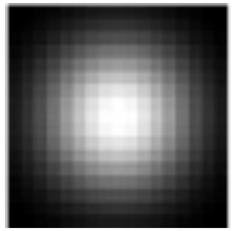
Bilateral filtering

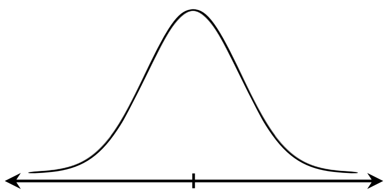
$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l]\, r_{mn}[k,l]\, f[m+k, n+l]$$

Spatial weighting

Intensity range weighting







$$r_{mn} = \frac{1}{\sqrt{2\pi}\sigma_r} e^{-\frac{x^2}{2\sigma_r^2}}$$

$$x = f[m,n] - f[m+k, n+l]$$

$$g[k,l] = \frac{1}{2\pi\sigma_s^2} \exp\left(-\frac{(k^2 + l^2)}{2\sigma_s^2}\right)$$

σ_s σ_r

Assign a pixel a large weight if: 1) it's nearby and 2) it looks like me

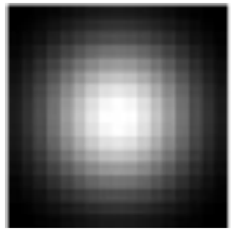
Bilateral filtering

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$

Normalization factor

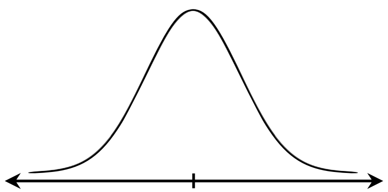
$$W_{mn} = \sum_{k,l} g[k,l] r_{mn}[k,l]$$

Spatial weighting



σ_s

Intensity range weighting



$x = f[m,n] - f[m+k, n+l]$

σ_r

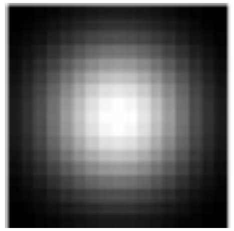
Assign a pixel a large weight if: 1) it's nearby and 2) it looks like me

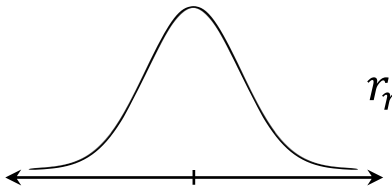
Implementation: Bilateral filtering

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l]\, r_{mn}[k,l]\, f[m+k, n+l]$$

Normalization factor | Spatial weighting | Intensity range weighting

$$W_{mn} = \sum_{k,l} g[k,l]\, r_{mn}[k,l]$$





$$r_{mn} = \frac{\cancel{1}}{\cancel{\sqrt{2\pi\sigma_r}}} e^{-\frac{x^2}{2{\sigma_r}^2}}$$

$$x = f[m,n] - f[m+k, n+l]$$

$$g[k,l] = \frac{\cancel{1}}{\cancel{2\pi{\sigma_s}^2}} \exp\left(-\frac{(k^2 + l^2)}{2{\sigma_s}^2}\right)$$

σ_s σ_r

Bilateral filtering vs Gaussian filtering

Which is which?

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

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$

Bilateral filtering vs Gaussian filtering

Gaussian filtering

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

Bilateral filtering

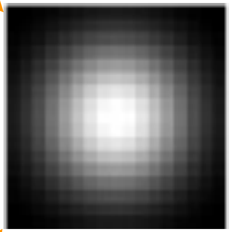
$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$

Bilateral filtering vs Gaussian filtering

Gaussian filtering

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

σ_s



Spatial weighting: favor *nearby* pixels

Bilateral filtering

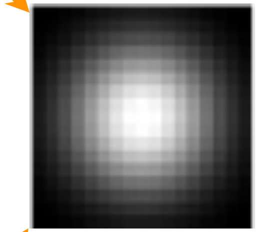
$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$

Bilateral filtering vs Gaussian filtering

Gaussian filtering

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

σ_s

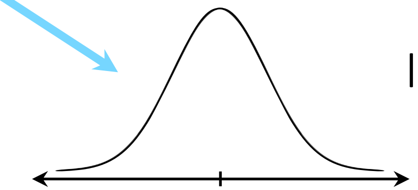


Spatial weighting: favor *nearby* pixels

Bilateral filtering

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$

σ_r



$x = f[m,n] - f[m+k, n+l]$

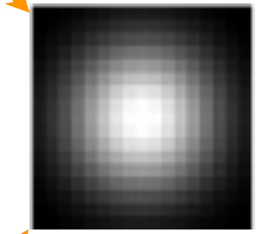
Intensity range weighting: favor *similar* pixels

Bilateral filtering vs Gaussian filtering

Gaussian filtering

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

σ_s



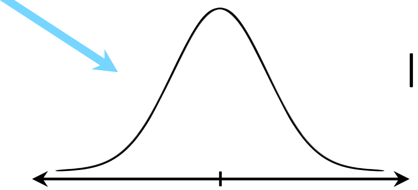
Spatial weighting: favor *nearby* pixels

Bilateral filtering

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$

Normalization factor

σ_r



$x = f[m,n] - f[m+k, n+l]$

Intensity range weighting: favor *similar* pixels

Bilateral filtering vs Gaussian filtering

Gaussian filtering

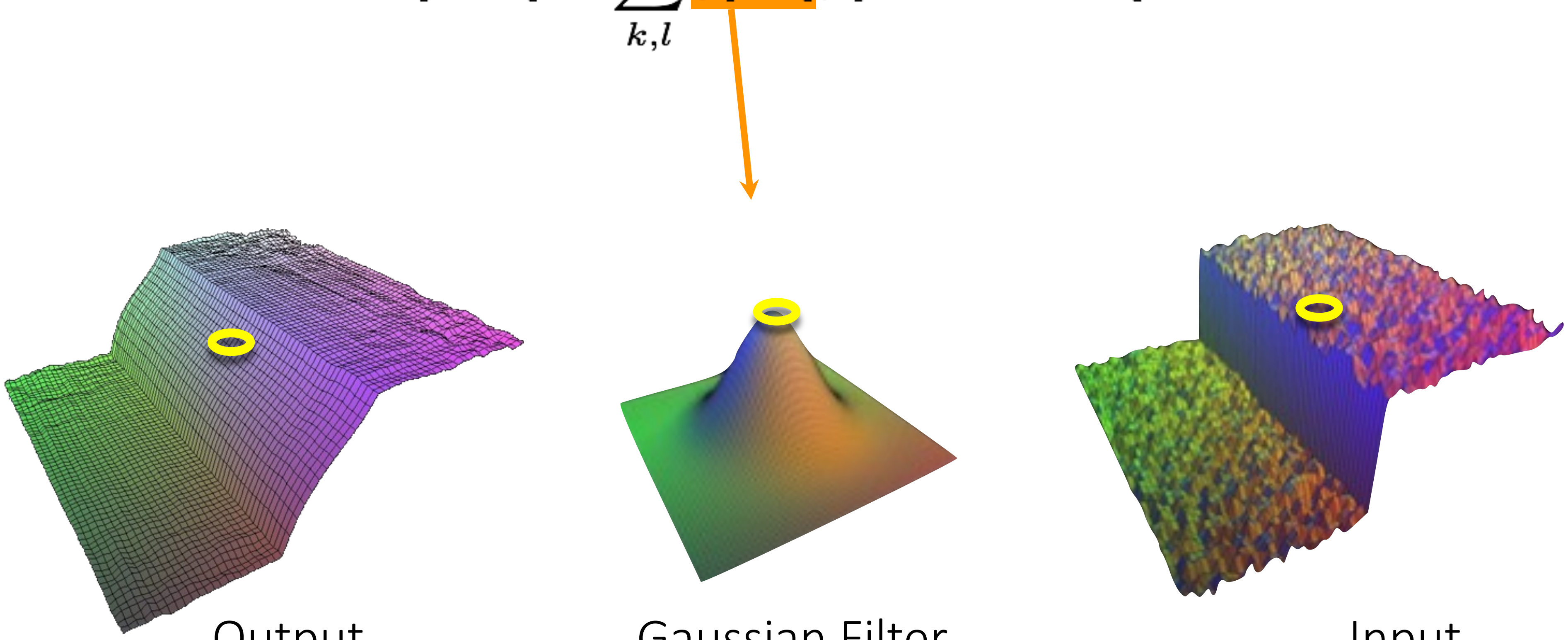
Smooths everything nearby (even edges)
Only depends on *spatial* distance

Bilateral filtering

Smooths 'close' pixels in space and intensity
Depends on *spatial* and *intensity* distance

Gaussian filtering visualization

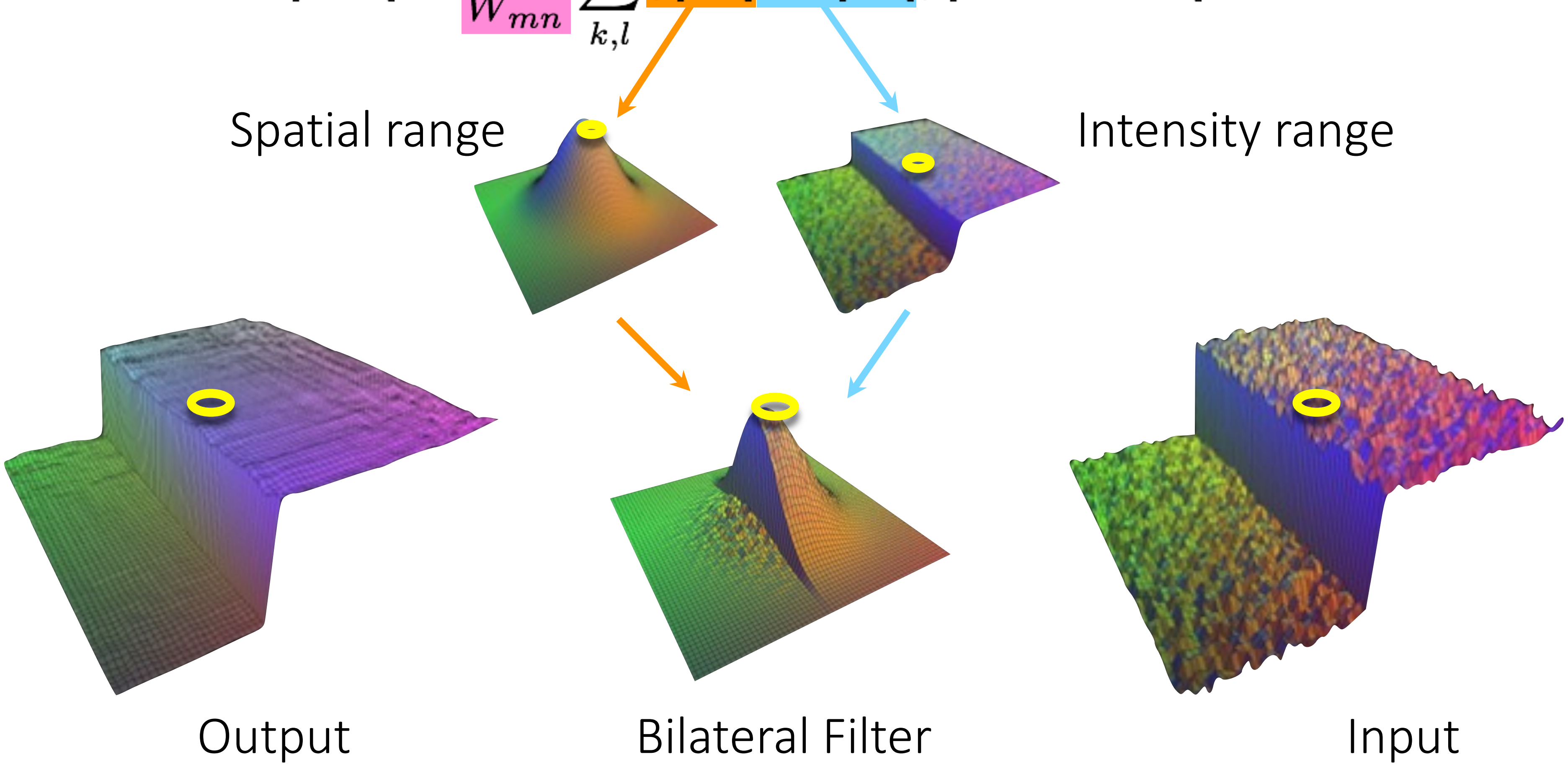
$$h[m,n] = \sum_{k,l} g[k,l] f[m+k, n+l]$$



Output Gaussian Filter Input

Bilateral filtering visualization

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$



Exploring the bilateral filter parameter space

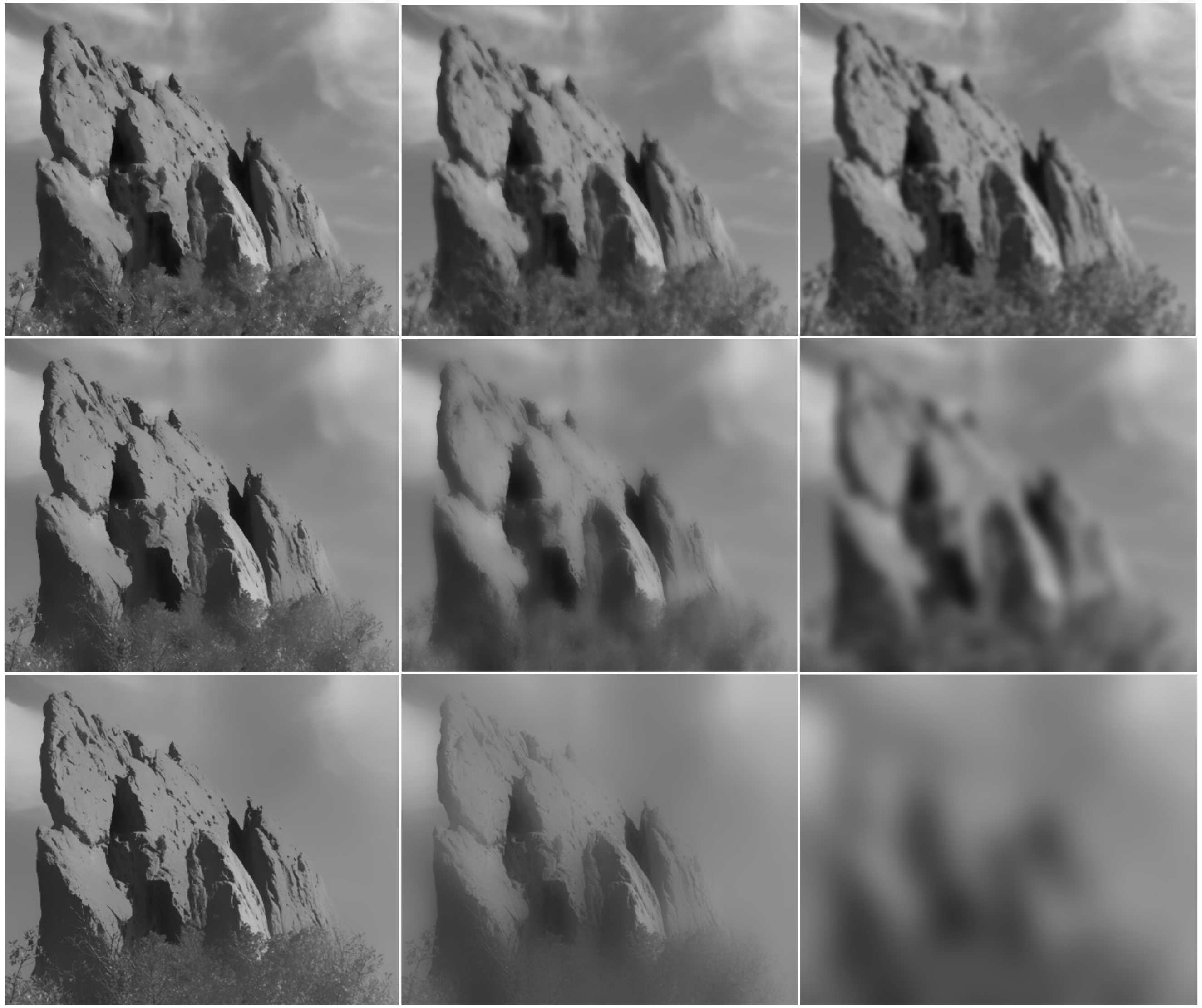
input

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

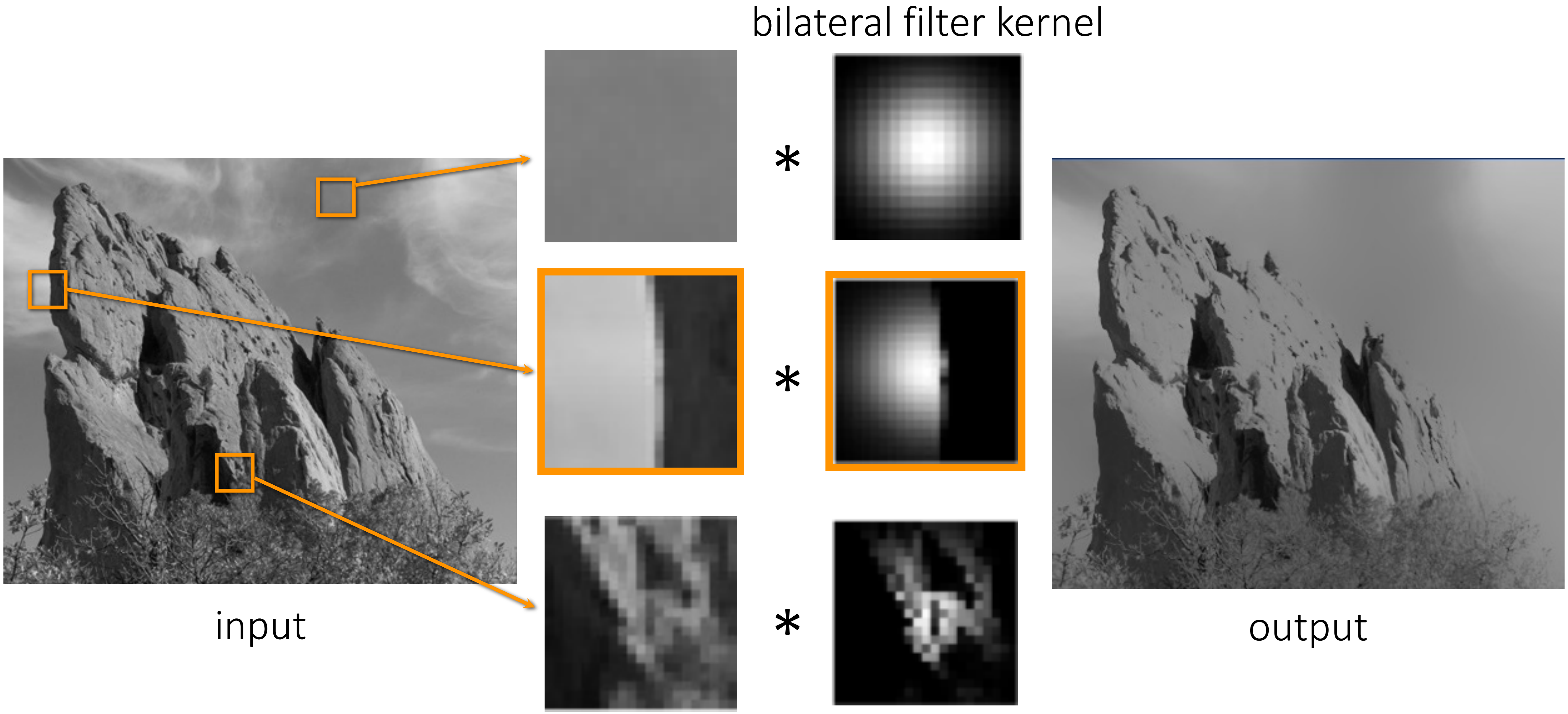
$\sigma_s = 2$

$\sigma_s = 6$

$\sigma_s = 18$

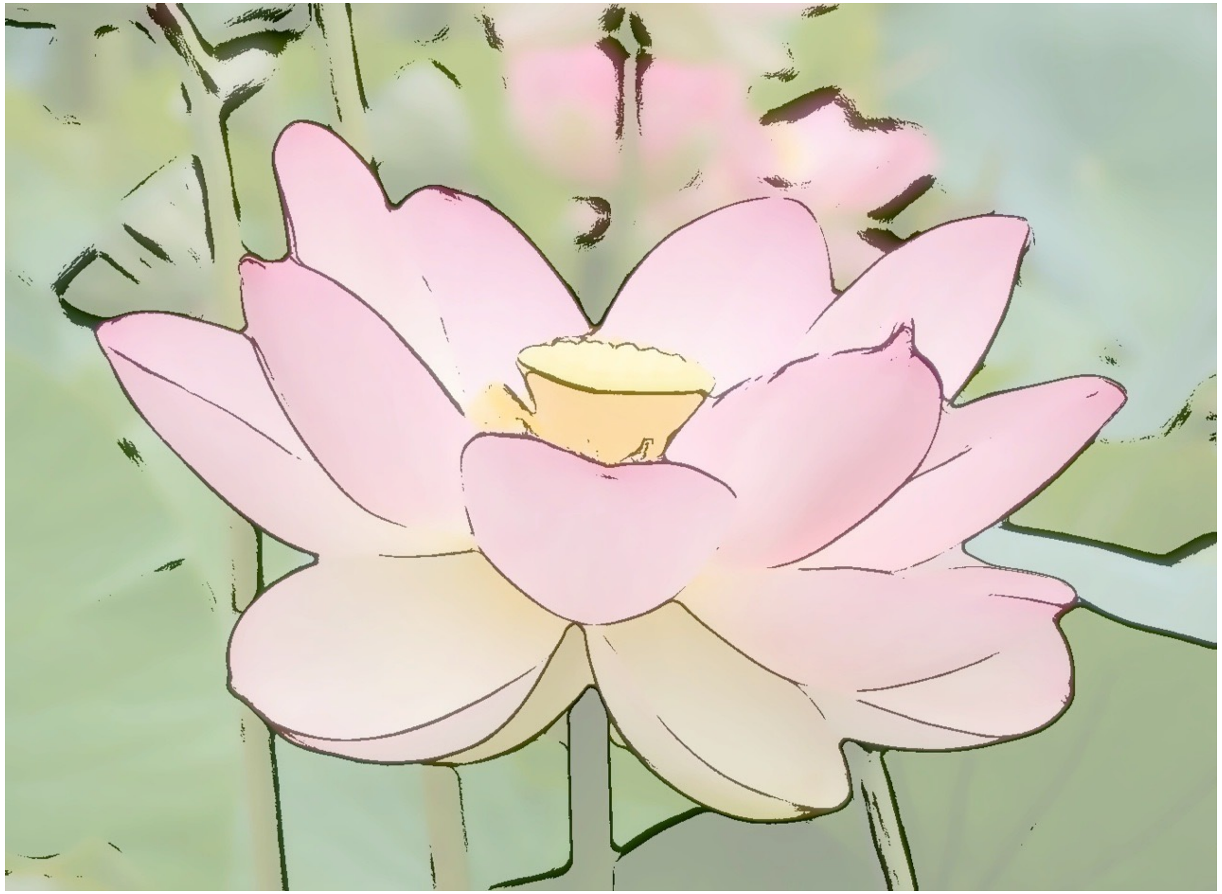


The bilateral filtering solution



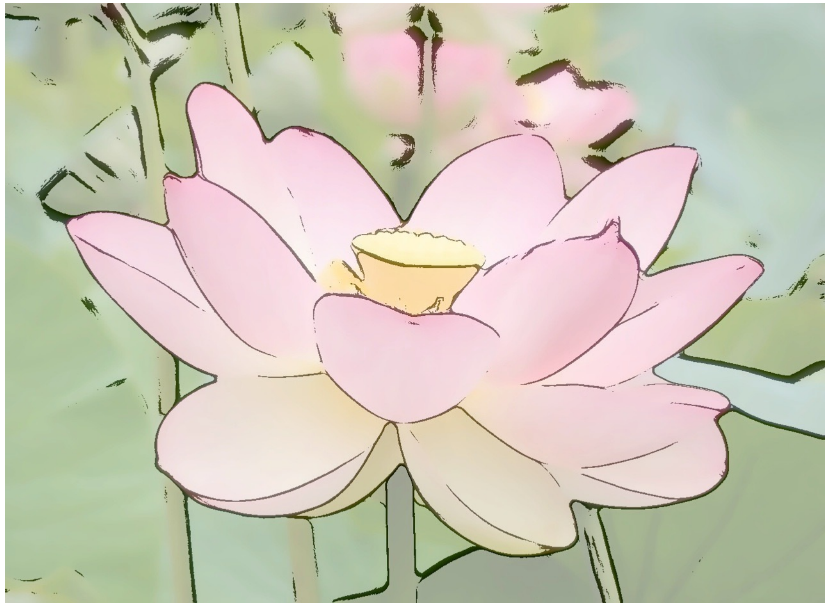
Do not blur if there is an edge!

Application: Cartoonization



How would you create this effect?

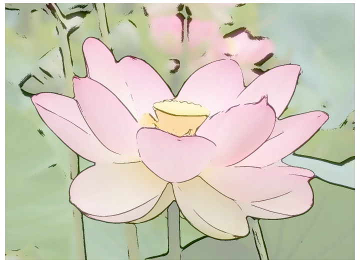
Application: Cartoonization



edges from bilaterally filtered image | bilaterally filtered image | cartoon rendition

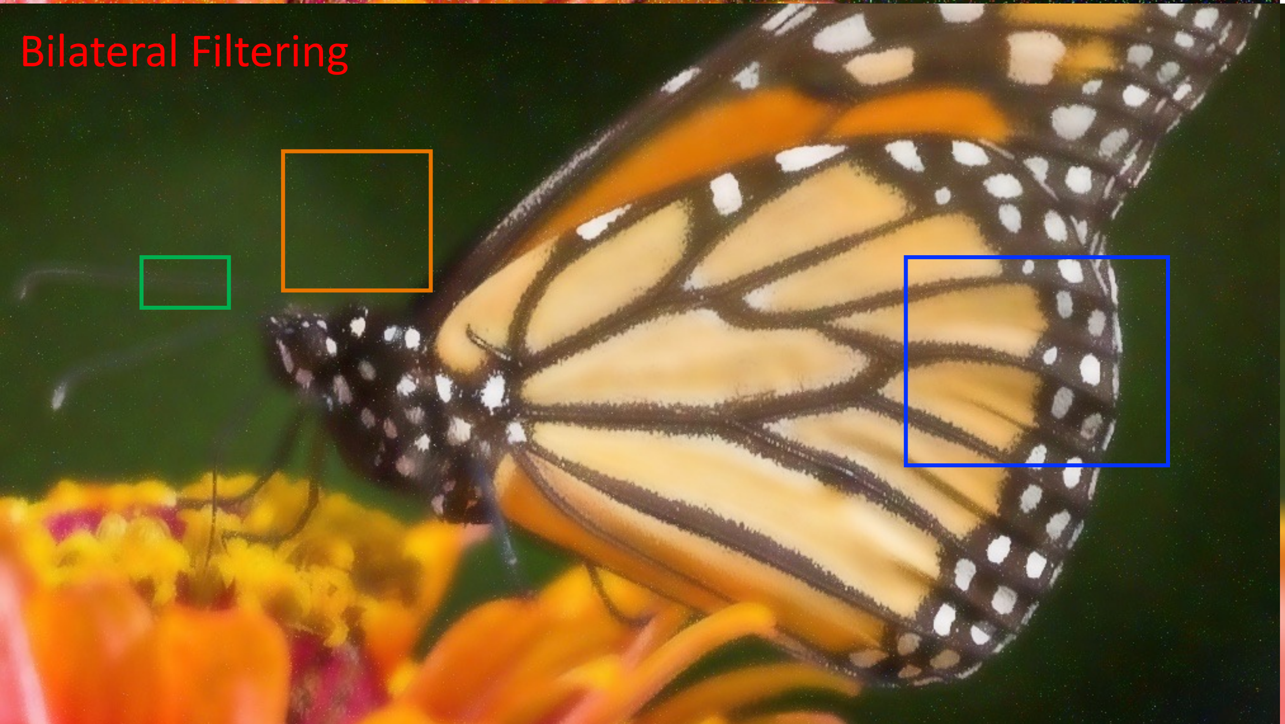
+

=



Application: Image Denoising with Bilateral Filtering

- Sharper edges
- Some thin edges may be reduced
- Flat regions are not fully smoothed



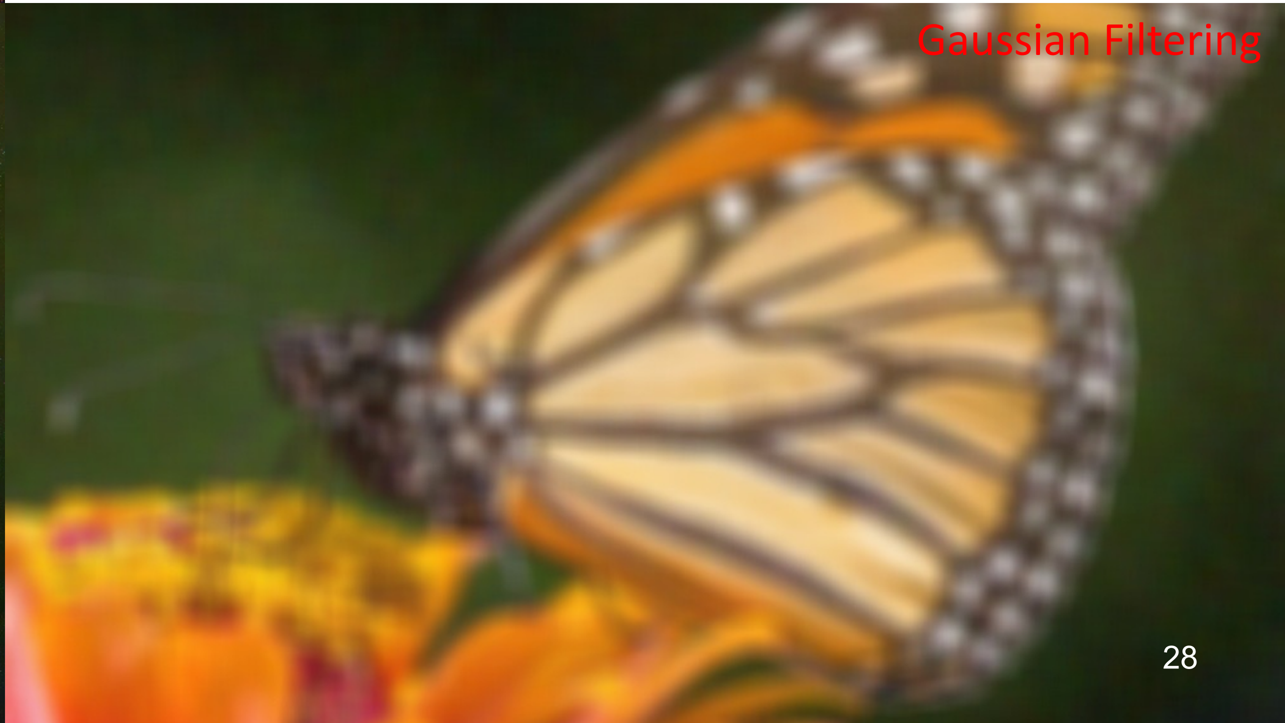
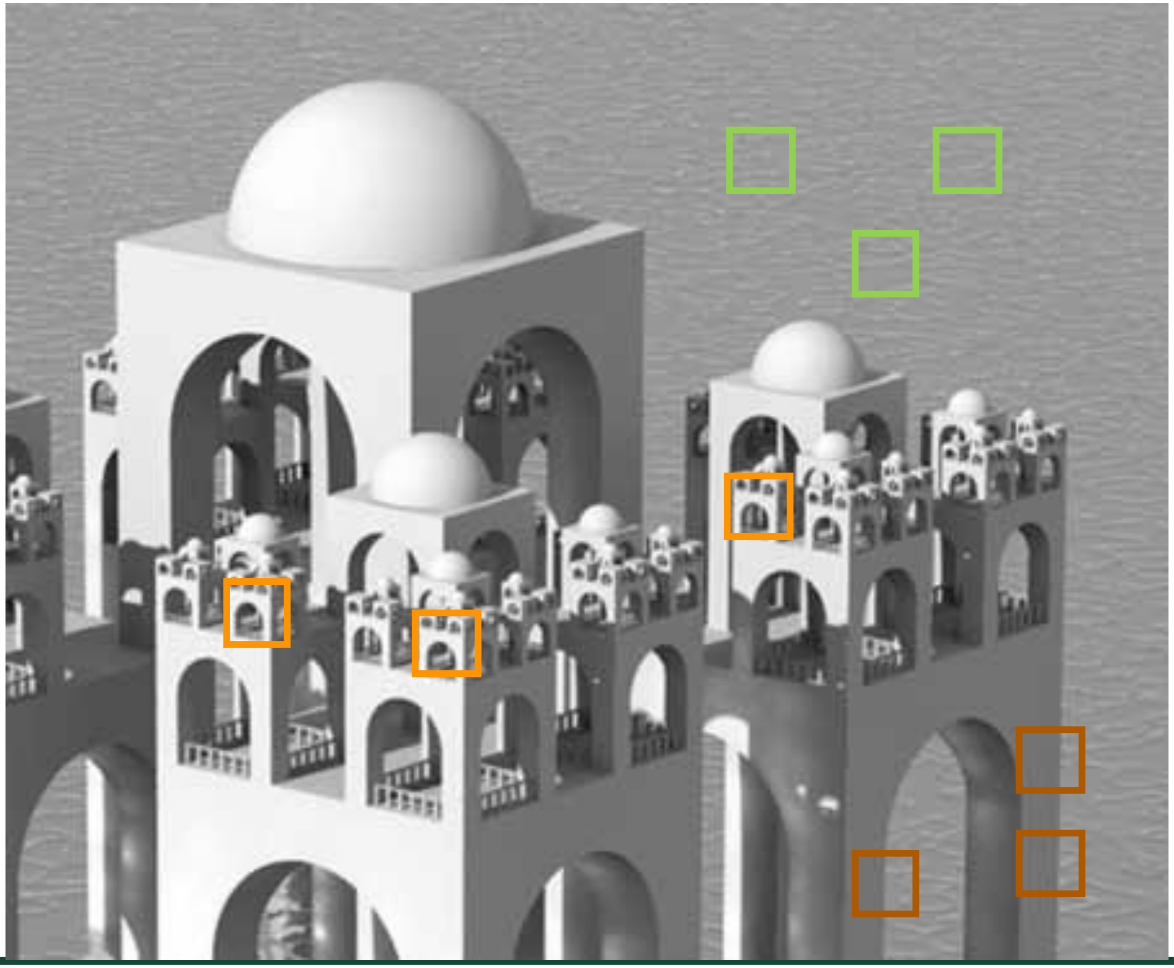


Image Prior: Non-local smoothness/redundancy



Small patches in natural images tend to redundantly appear multiple times

Non-local means Filter

No need to stop at neighborhood. Instead search *everywhere* in the image.

Given a pixel $f(p)$ at position $p = (p_x, p_y)$, the filter uses pixels in the whole image to update $f(p)$

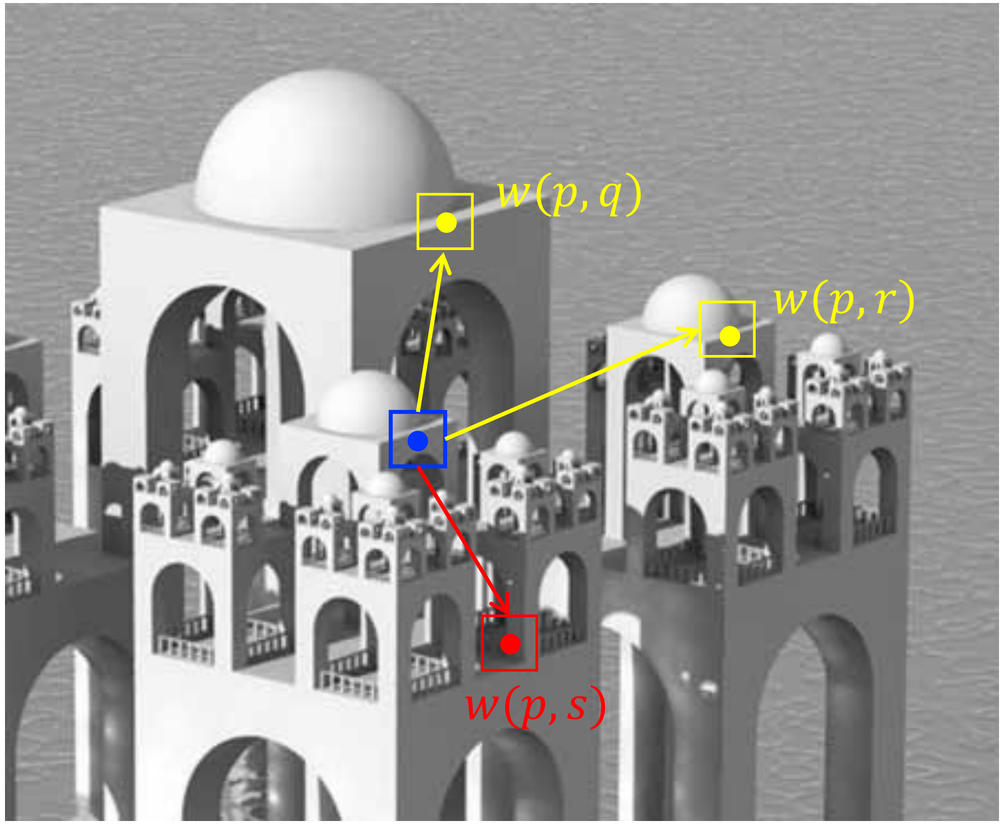
$$h(p) = \frac{1}{W}\sum_{q} w(p,q)f(q)$$

Weight: $w(p,q) = \exp(-\frac{SSD(p,q)}{2\sigma^2})$

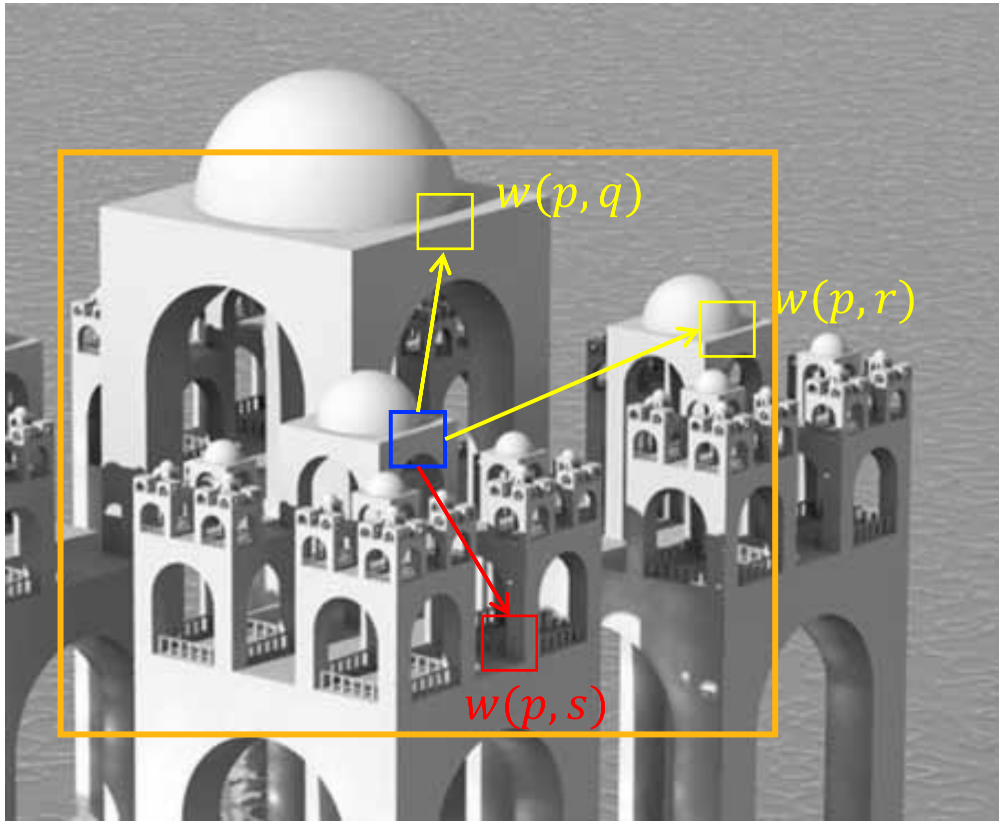
Sum of the squared difference between two patches

$$SSD(p,q) = \sum_{k=-n}^{n}\sum_{l=-n}^{n}\left(f\left(p_x+k, p_y+l\right) - f(q_x+k, q_y+l)\right)^2$$

$W = \sum_q w(p,q)$ is the normalization term



Fast Implementation of Non-local Means



Scan over the whole image to compute weights for each pixel is time-consuming

Implementation:

- set a search window (e.g., 21x21) with the target pixel position as the center
- only use pixels inside the window to compute weights based on patch similarity

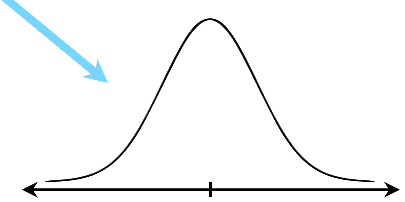
Patch size (e.g., 5x5, 7x7) is much smaller than the window size

Non-local means vs bilateral filtering

Non-local means filtering

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} r_{mn}[k,l] f[m+k, n+l]$$

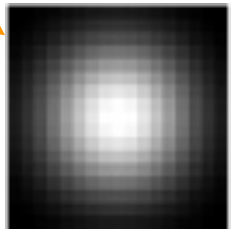
Bilateral filtering



$x = f[m,n] - f[m+k,n+l]$

Intensity range weighting: favor *similar* pixels (patches in case of non-local means)

$$h[m,n] = \frac{1}{W_{mn}} \sum_{k,l} g[k,l] r_{mn}[k,l] f[m+k, n+l]$$



Spatial weighting: favor *nearby* pixels


Nonlocal Means Filtering
Noisy Image
Bilateral Filtering
Gaussian Filtering

Summary

Gaussian filtering

Smooths everything nearby (even edges)
Only depends on *spatial* distance

Bilateral filtering

Smooths ‘close’ pixels in space and intensity
Depends on *spatial* and *intensity* distance

Non-local means

Smooths similar patches no matter how far away
Only depends on *intensity* distance

Further Reading

Chapters 3.3.1 and 3.3.2, Computer Vision: Algorithms and Applications, Richard Szeliski

https://en.wikipedia.org/wiki/Non-local_means