Modern Image Smoothing:
Bilateral Filtering, Non-local Means Denoising, and LARK filter

Erkut Erdem

Review - Smoothing and Edge Detection

- While eliminating noise via smoothing, we also lose some of the (important) image details.
  - Fine details
  - Image edges
  - etc.
- What can we do to preserve such details?
  - Use edge information during denoising!
  - This requires a definition for image edges.

Chicken-and-egg dilemma!

- Edge preserving image smoothing

Today

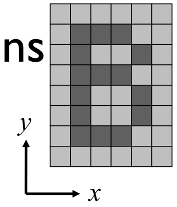
- Bilateral filtering
- Non-local means denoising
- LARK filter

Today

- Bilateral filtering
- Non-local means denoising
- LARK filter

Acknowledgement: The slides are adapted from the course "A Gentle Introduction to Bilateral Filtering and its Applications" given by Sylvain Paris, Pierre Kornprobst, Jack Tumblin, and Frédo Durand (http://people.csail.mit.edu/sparis/bf_course/).

Notation and Definitions

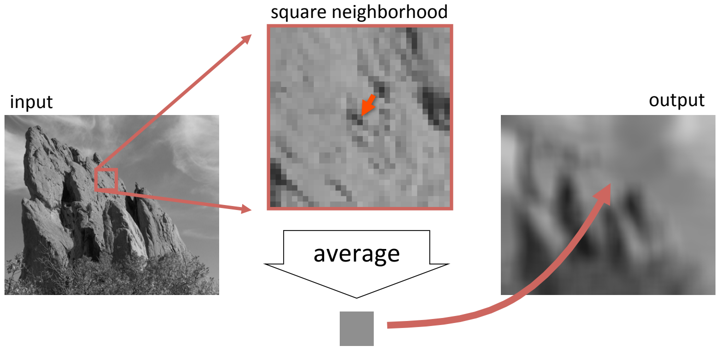


- Image = 2D array of pixels
- Pixel = intensity (scalar) or color (3D vector)
- $I_{\mathbf{p}}$ = value of image I at position: $\mathbf{p} = (p_x, p_y)$
- $F[I]$ = output of filter F applied to image I

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

Box Average



Equation of Box Average

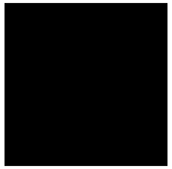
$$BA[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in S} B_\sigma(\mathbf{p}-\mathbf{q})\, I_{\mathbf{q}}$$

result at pixel **p**

sum over all pixels **q**

intensity at pixel **q**

normalized box function

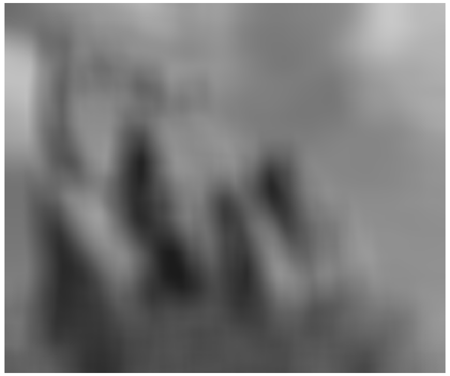


Square Box Generates Defects

- Axis-aligned streaks
- Blocky results

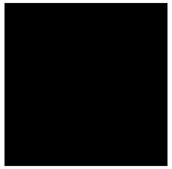
input

output

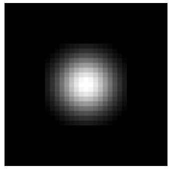


Strategy to Solve these Problems

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

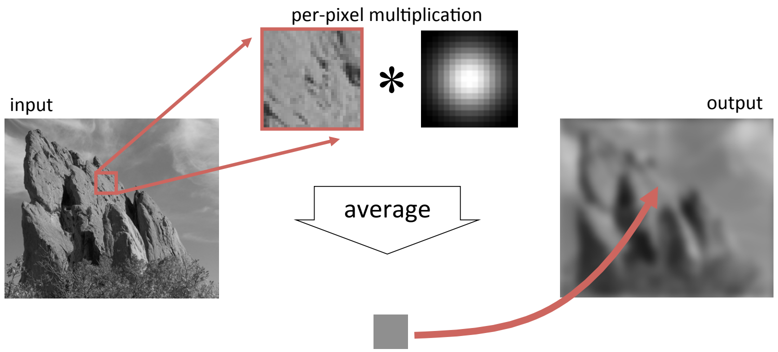


box window



Gaussian window

Gaussian Blur

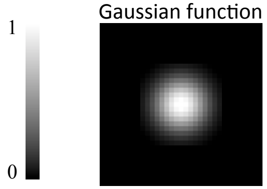


Equation of Gaussian Blur

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

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

normalized Gaussian function

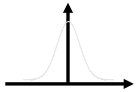


Spatial Parameter

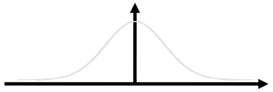
input

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

size of the window

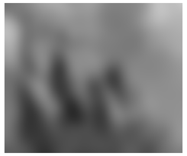


small *s*



large *s*

limited smoothing



strong smoothing

How to set *s*

- Depends on the application.
- Common strategy: proportional to image size
  - e.g. 2% of the image diagonal
  - property: independent of image resolution

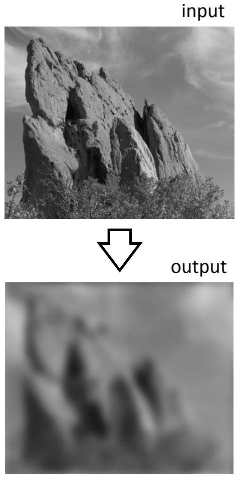
Properties of Gaussian Blur

- Weights independent of spatial location
  - linear convolution
  - well-known operation
  - efficient computation (recursive algorithm, FFT…)

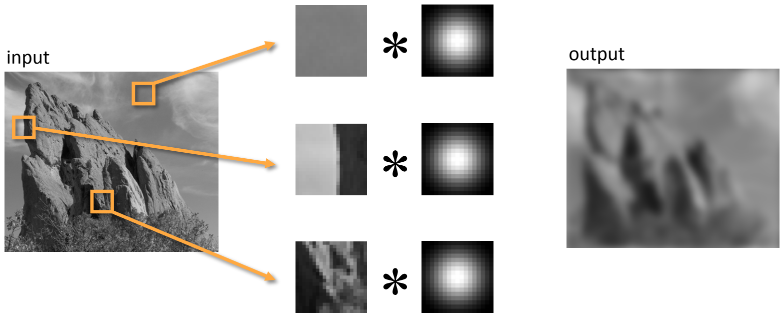
Properties of Gaussian Blur

- Does smooth images
- But smoothes too much: **edges are blurred**.
  - Only spatial distance matters
  - No edge term

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



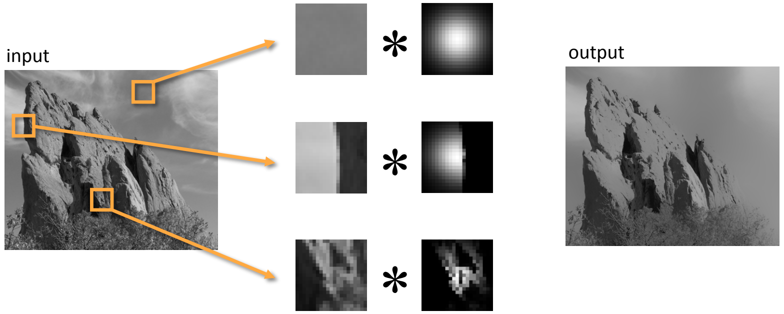
Blur Comes from Averaging across Edges



Same Gaussian kernel everywhere.

Bilateral Filter [Aurich 95, Smith 97, Tomasi 98]

No Averaging across Edges

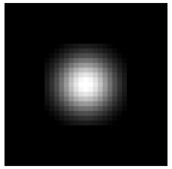


The kernel shape depends on the image content.

Bilateral Filter Definition: an Additional Edge Term

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

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



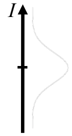
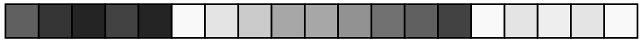
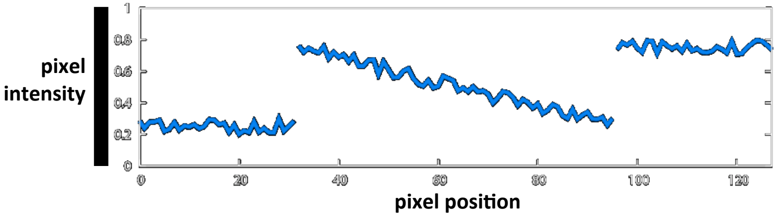


Illustration a 1D Image

- 1D image = line of pixels

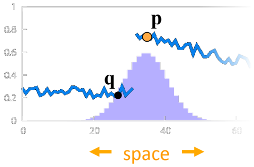


- Better visualized as a plot

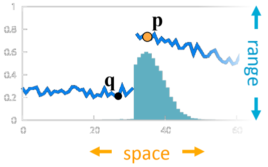


Gaussian Blur and Bilateral Filter

Gaussian blur



Bilateral filter
[Aurich 95, Smith 97, Tomasi 98]

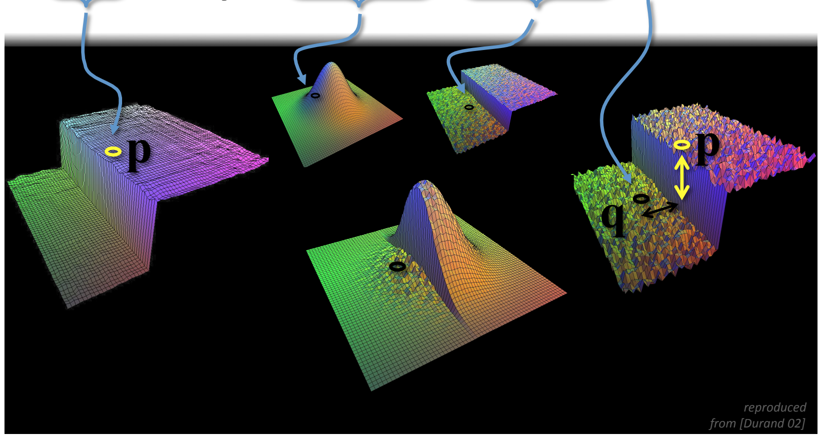


$$GB[I]_{\mathbf{p}} = \sum_{\mathbf{q}\in S} \underbrace{G_{\sigma}(\|\mathbf{p}-\mathbf{q}\|)}_{\text{space}} I_{\mathbf{q}}$$

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

Bilateral Filter on a Height Field

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



reproduced from [Durand 02]

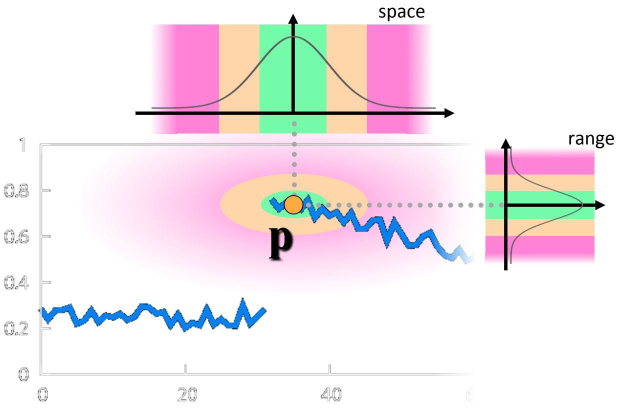
Space and Range Parameters

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

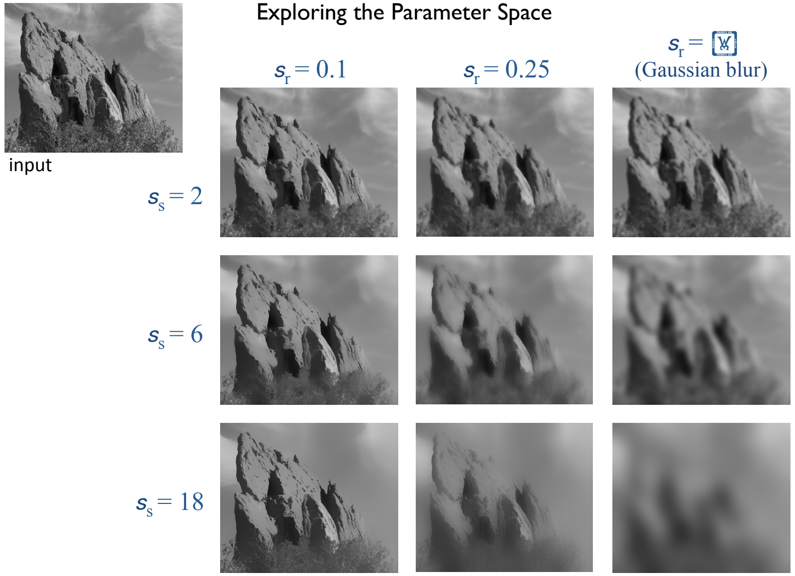
- space s_s : spatial extent of the kernel, size of the considered neighborhood.
- range s_r : “minimum” amplitude of an edge

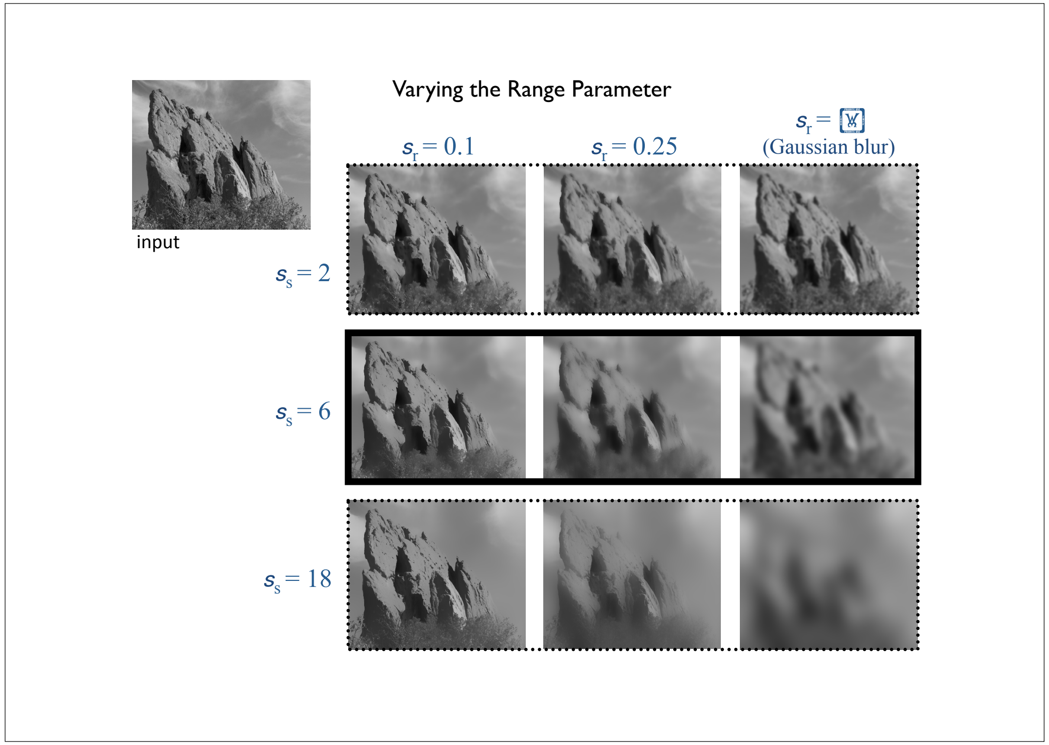
Influence of Pixels

Only pixels close in space and in range are considered.



Exploring the Parameter Space





Varying the Range Parameter

input

$s_r = 0.1$

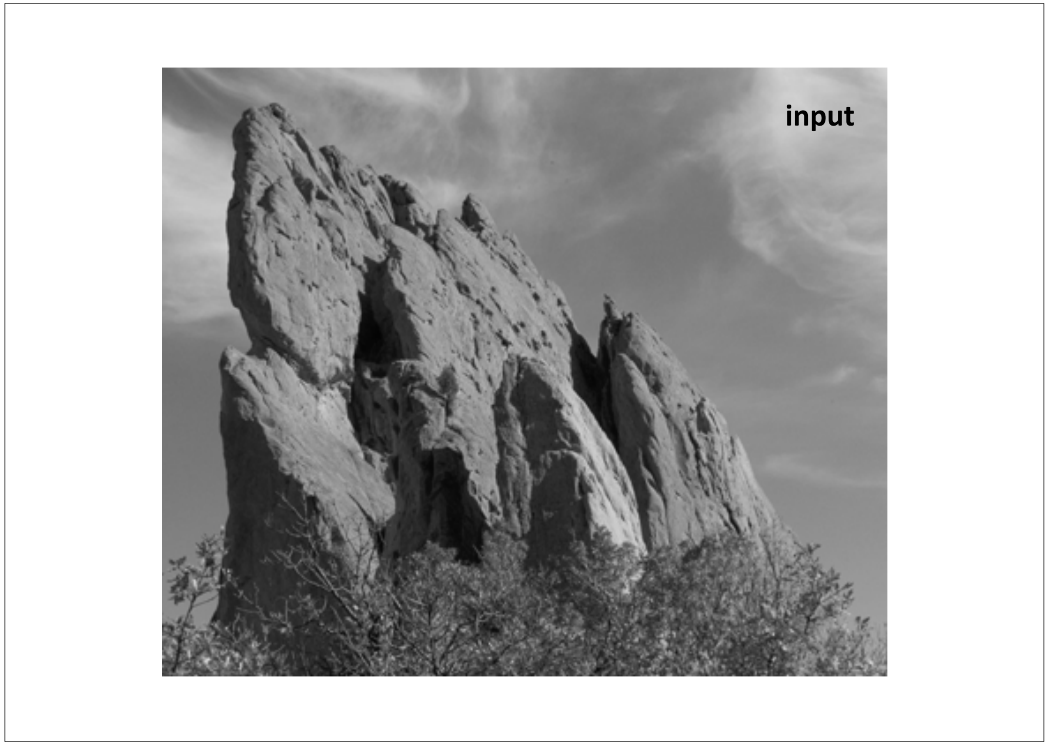
$s_r = 0.25$

$s_r =$ □
(Gaussian blur)

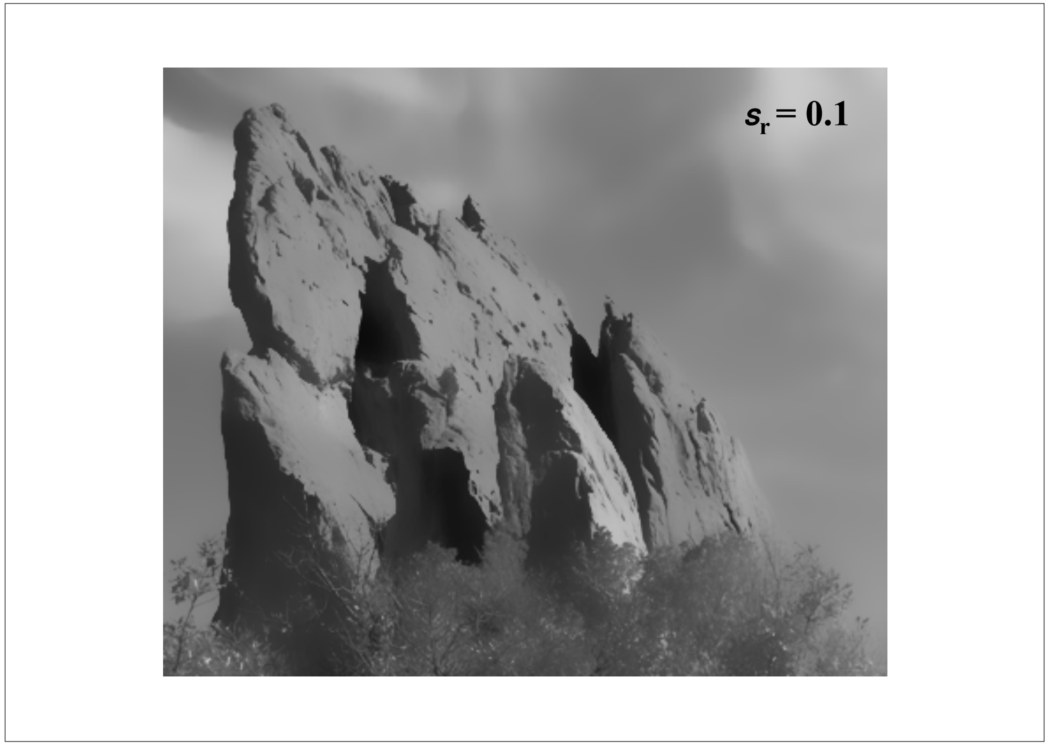
$s_s = 2$

$s_s = 6$

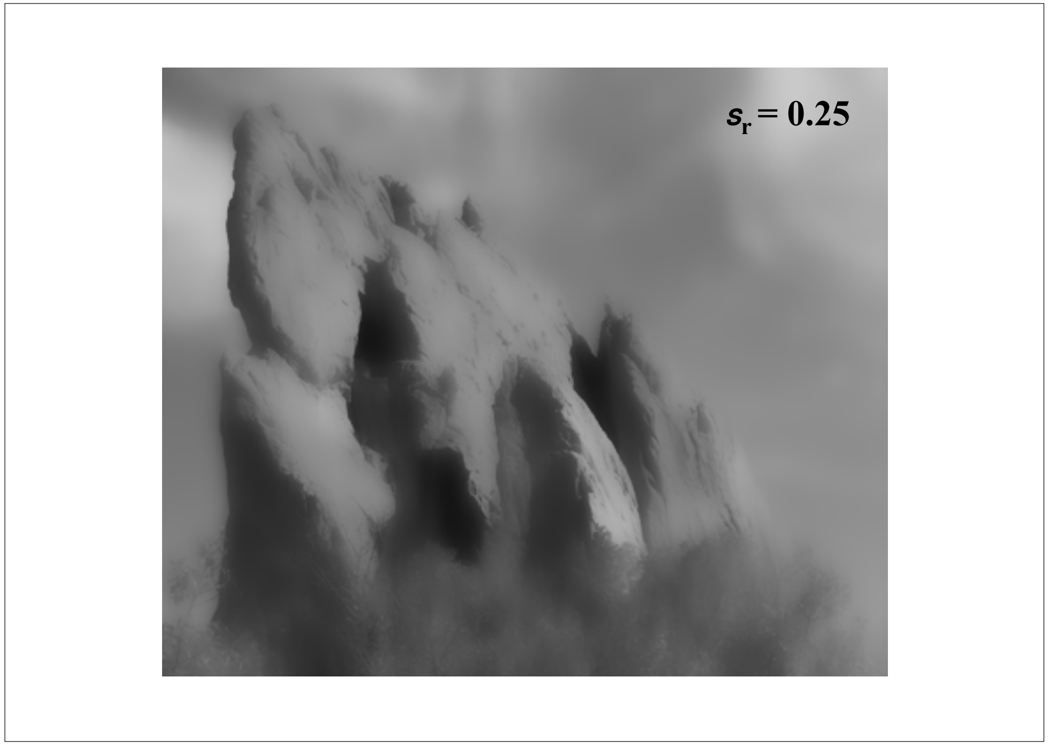
$s_s = 18$



input

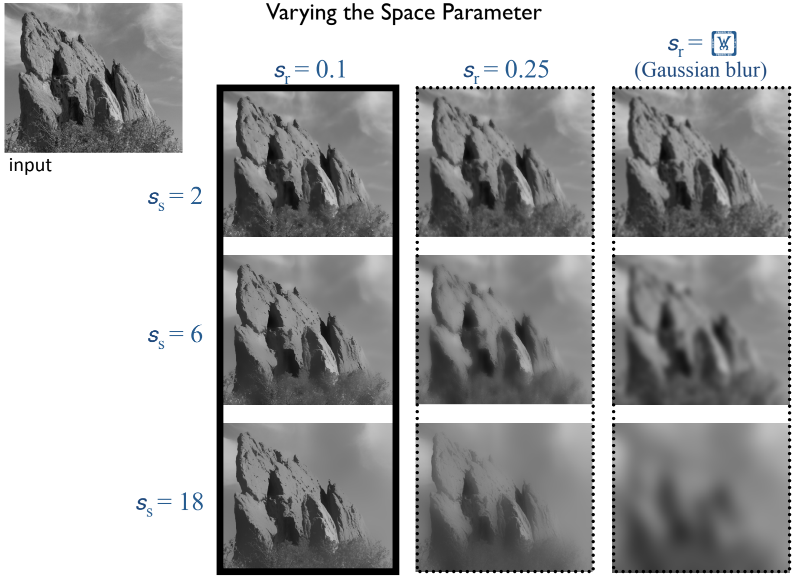


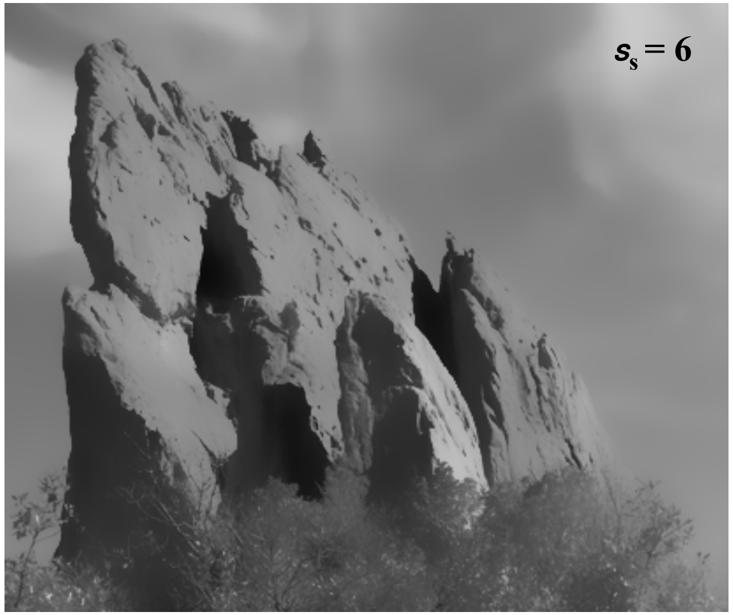
$\mathbf{s_r = 0.1}$

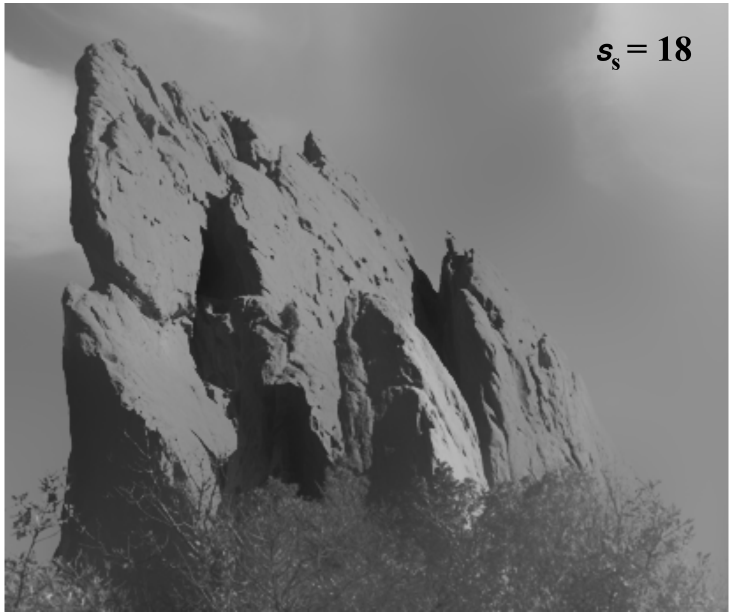


$\mathbf{s_r = 0.25}$

Varying the Space Parameter







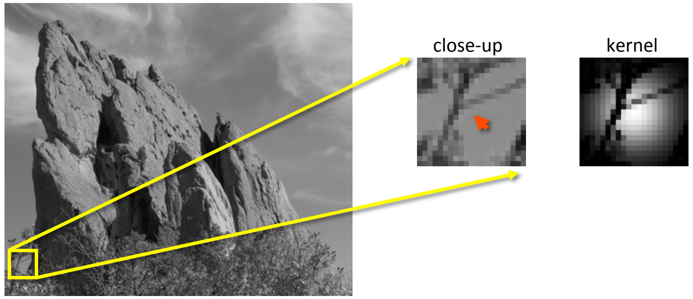
How to Set the Parameters

Depends on the application. For instance:

- space parameter: proportional to image size
  - e.g., 2% of image diagonal
- range parameter: proportional to edge amplitude
  - e.g., mean or median of image gradients
- independent of resolution and exposure

Bilateral Filter Crosses Thin Lines

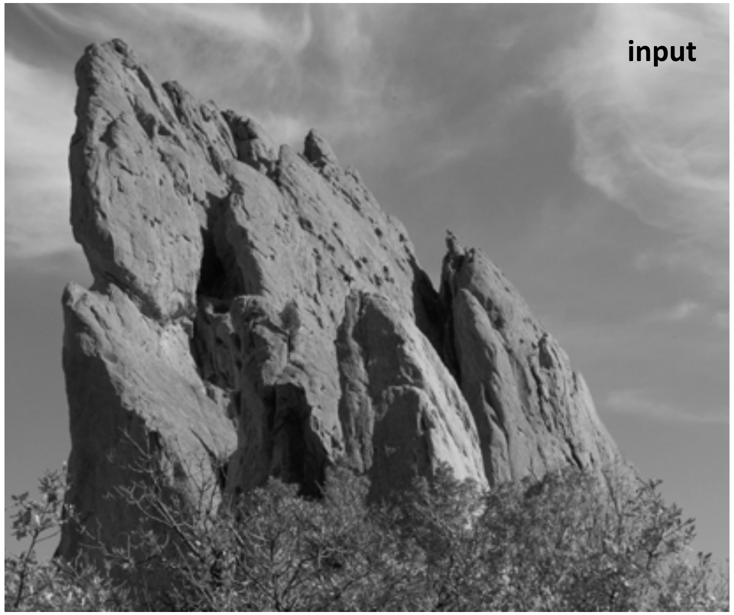
- Bilateral filter averages across features thinner than ~2s_s
- Desirable for smoothing: more pixels = more robust
- Different from diffusion that stops at thin lines

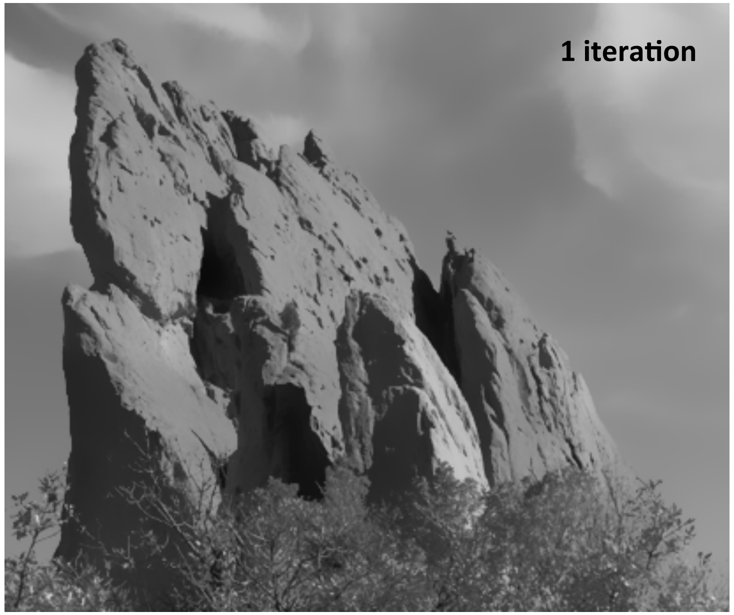


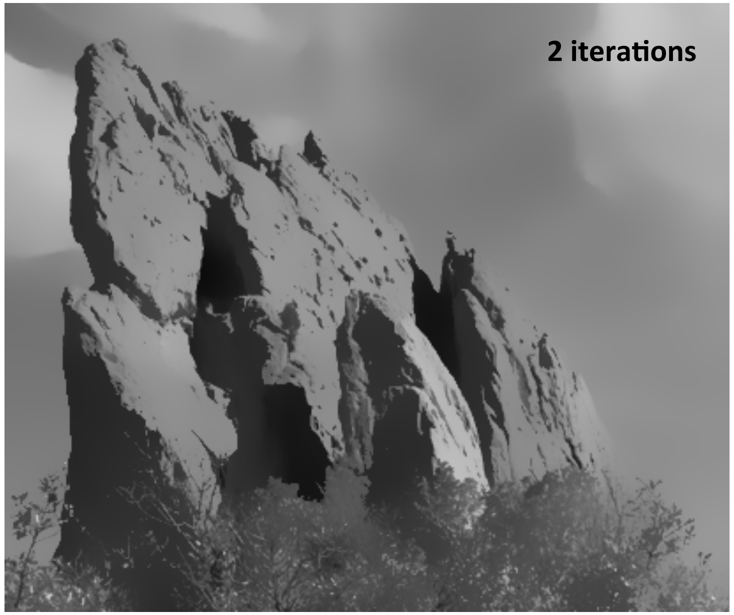
Iterating the Bilateral Filter

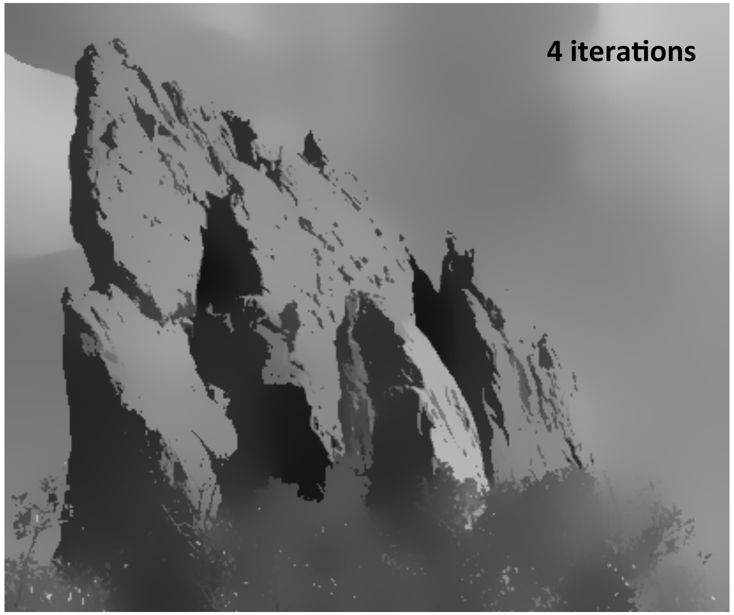
$$I_{(n+1)} = BF\,[I_{(n)}]$$

- Generate more piecewise-flat images
- Often not needed in computational photo.









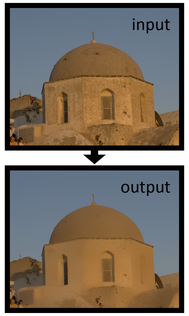
Bilateral Filtering Color Images

For gray-level images

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_r}\big(\overbrace{|I_{\mathbf{p}} - I_{\mathbf{q}}|}^{\text{intensity difference}}\big)\, \underbrace{I_{\mathbf{q}}}_{\text{scalar}}$$

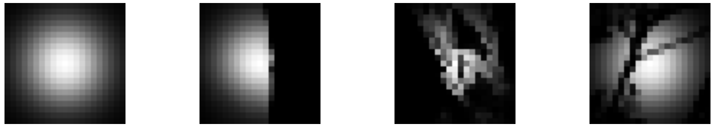
For color images

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)\, G_{\sigma_r}\big(\overbrace{\|\mathbf{C}_{\mathbf{p}} - \mathbf{C}_{\mathbf{q}}\|}^{\text{color difference}}\big)\, \underbrace{\mathbf{C}_{\mathbf{q}}}_{\substack{\text{3D vector}\\ \text{(RGB, Lab)}}}$$



Hard to Compute

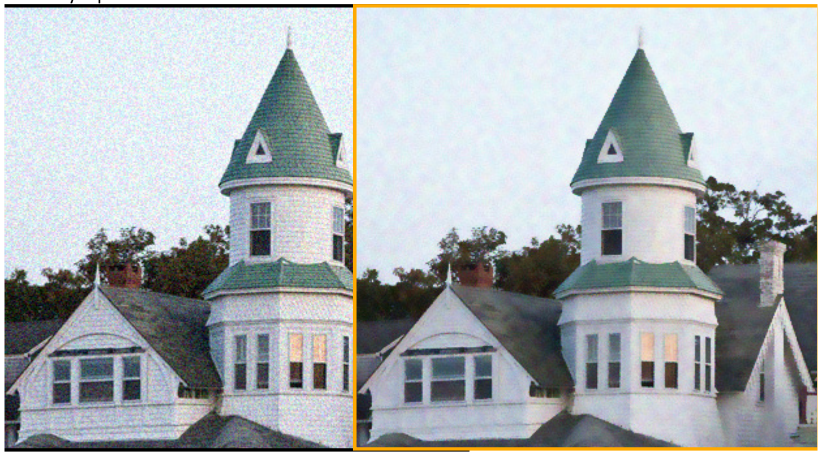
- Nonlinear $BF[I]_{\mathbf{p}} = \boxed{\frac{1}{W_{\mathbf{p}}}} \sum_{\mathbf{q}\in S} G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|) \boxed{G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)} I_{\mathbf{q}}$
- Complex, spatially varying kernels
  - Cannot be precomputed, no FFT…



- 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*

Basic denoising

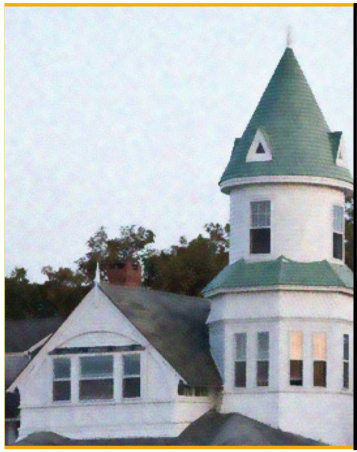


Noisy input

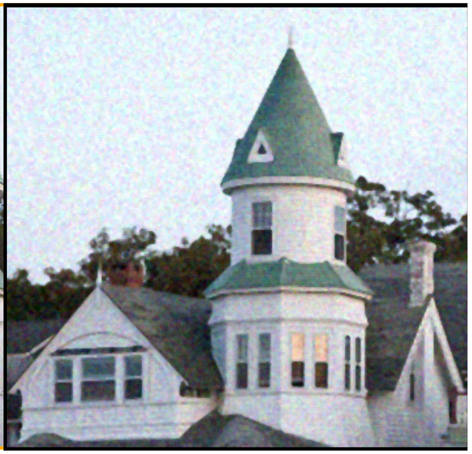
Bilateral filter 7x7 window

Basic denoising

Bilateral filter

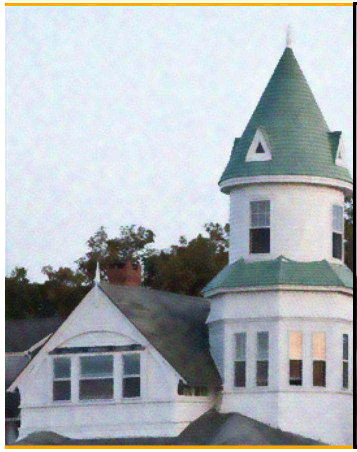


Median 3x3

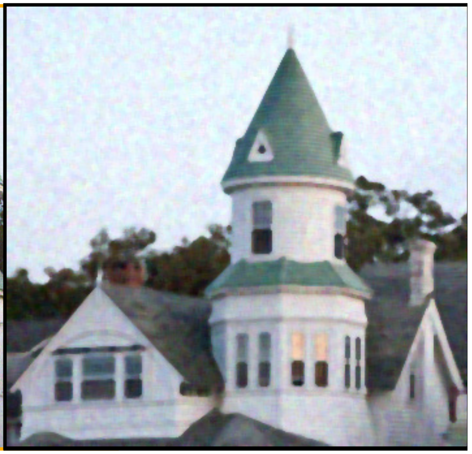


Basic denoising

Bilateral filter

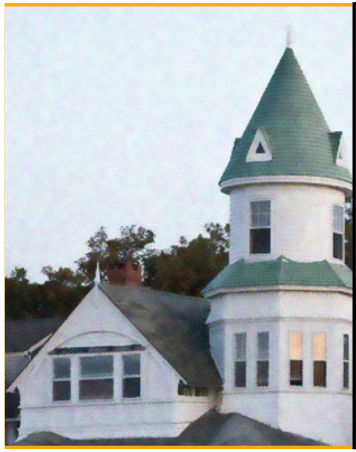


Median 5x5

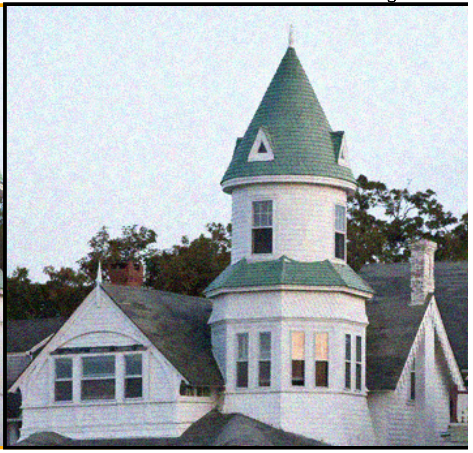


Basic denoising

Bilateral filter

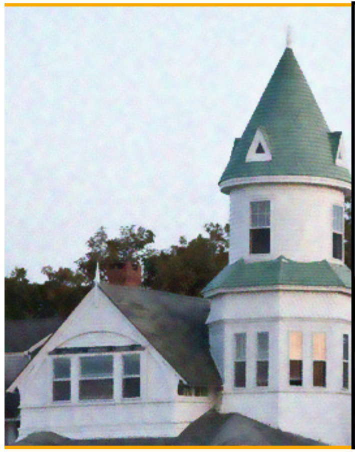


Bilateral filter – lower sigma

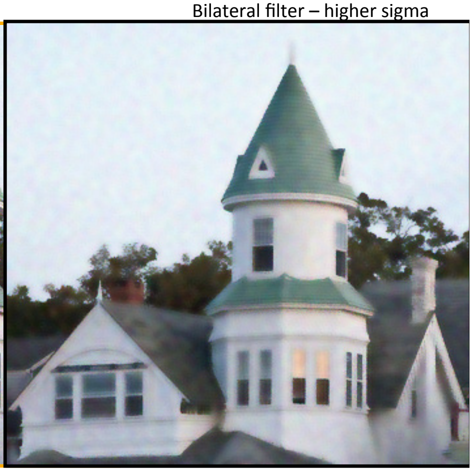


Basic denoising

Bilateral filter



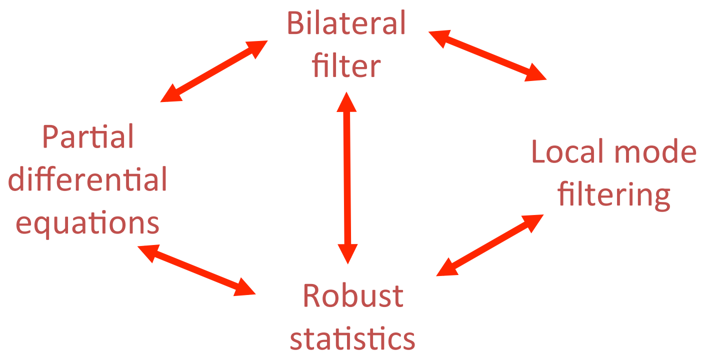
Bilateral filter – higher sigma



Denoising

- Small spatial sigma (e.g. 7x7 window)
- Adapt range sigma to noise level
- Maybe not best denoising method, but best simplicity/quality tradeoff
  - No need for acceleration (small kernel)

Goal: Understand how does bilateral filter relates with other methods



<u>Today's paper:</u> *Generalised Nonlocal Image Smoothing*,
L. Pizarro, P. Mrazek, S. Didas, S. Grewenig and J. Weickert, IJCV, 2010

Today

- Bilateral filtering
- Non-local means denoising
- LARK filter

<u>Acknowledgement:</u> The slides are adapted from the course "A Gentle Introduction to Bilateral Filtering and its Applications" given by Sylvain Paris, Pierre Kornprobst, Jack Tumblin, and Frédo Durand (http://people.csail.mit.edu/sparis/bf_course/).

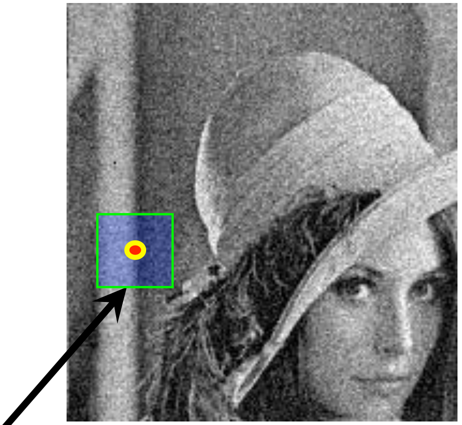
New Idea:
NL-Means Filter (Buades 2005)

- Same goals: 'Smooth within Similar Regions'
- **<u>KEY INSIGHT</u>**: Generalize, extend 'Similarity'
  - **Bilateral:**
    Averages neighbors with **<u>similar intensities</u>**;
  - **NL-Means:**
    Averages neighbors with **<u>similar neighborhoods!</u>**

NL-Means Method:
Buades (2005)

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

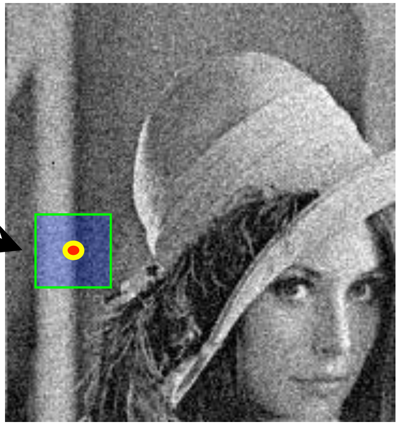
NL-Means Method:
Buades (2005)



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

NL-Means Method:
Buades (2005)

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

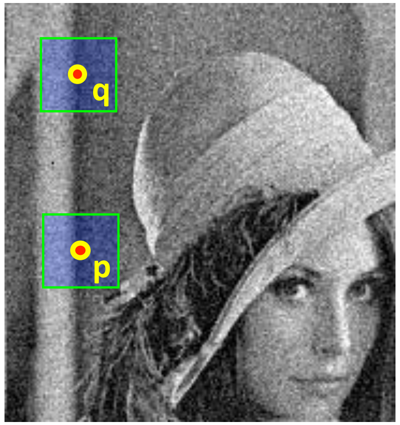


- For each and every pixel **p:**
  - – Define a small, simple fixed size neighborhood;
  - – Define vector $\mathbf{V_p}$: a list of neighboring pixel values.

NL-Means Method:
Buades (2005)

<u>'Similar'</u> pixels **p, q**
→ **SMALL**
vector distance;

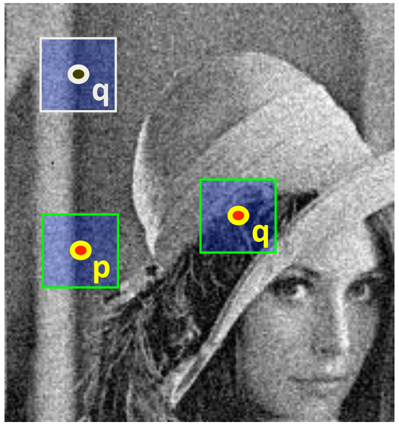
$$|| V_p - V_q ||^2$$



NLM-Means Method: Buades (2005)

‘Dissimilar’ pixels **p, q**
→ **LARGE** vector distance;

$|| V_p - V_q ||^2$

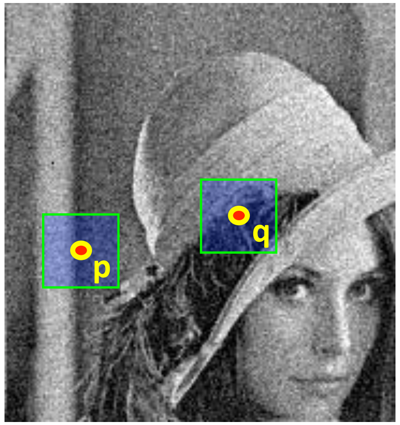


NL-Means Method: Buades (2005)

‘Dissimilar’ pixels **p, q**
→ **LARGE** vector distance;

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

Filter with this!



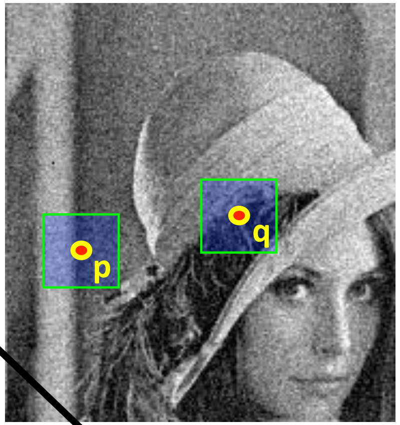
NL-Means Method: Buades (2005)

p, q neighbors define a vector distance;

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

Filter with this:

No spatial term!



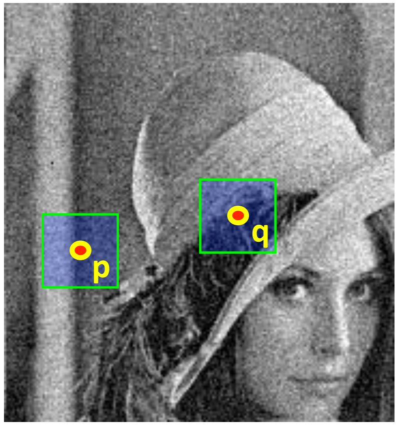
$$NLMF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in S} \cancel{G_{\sigma_s}(\|\mathbf{p}-\mathbf{q}\|)}\, \boxed{G_{\sigma_r}\left(\boxed{\|\vec{V}_{\mathbf{p}} - \vec{V}_{\mathbf{q}}\|^2}\right)} I_{\mathbf{q}}$$

NL-Means Method: Buades (2005)

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

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

Vector Distance to p sets weight for each pixel q



$$NLMF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q}\in S} \boxed{G_{\sigma_r}\left(\boxed{\|\vec{V}_{\mathbf{p}} - \vec{V}_{\mathbf{q}}\|^2}\right)} I_{\mathbf{q}}$$

NL-Means Method: Buades (2005)

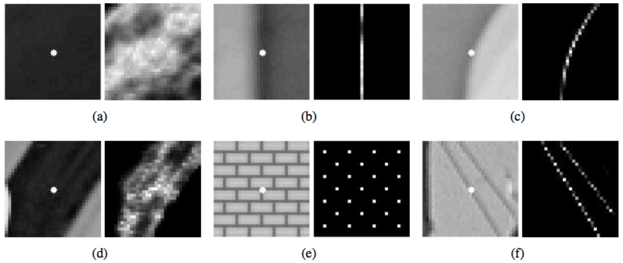


Figure 2. Display of the NL-means weight distribution used to estimate the central pixel of every image. The weights go from 1(white) to zero(black).

NL-Means Method: Buades (2005)

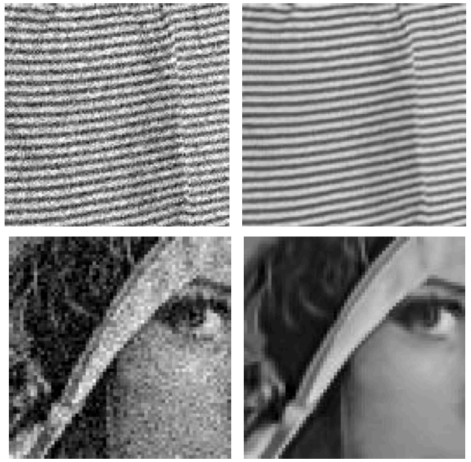
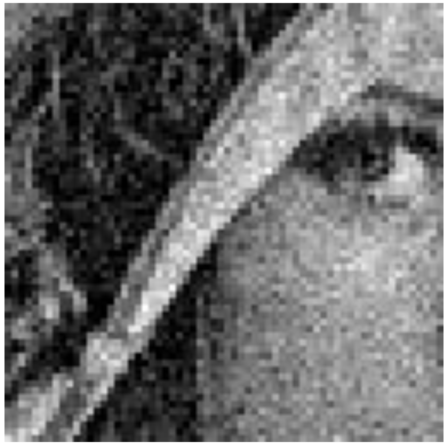


FIG. 9. *NL-means denoising experiment with a natural image. Left: Noisy image with standard deviation 20. Right: Restored image.*

NL-Means Method: Buades (2005)

- Noisy source image:



NL-Means Method: Buades (2005)

- Gaussian Filter

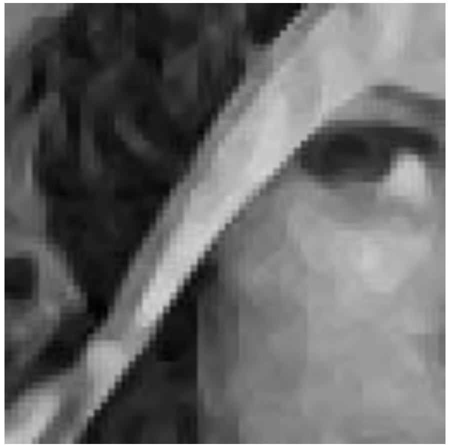
Low noise,

Low detail

NL-Means Method: Buades (2005)

- Anisotropic Diffusion

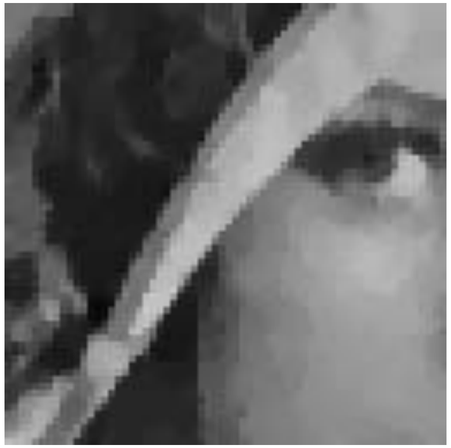
(Note ‘stairsteps’ : ~ piecewise constant)



NL-Means Method: Buades (2005)

- Bilateral Filter

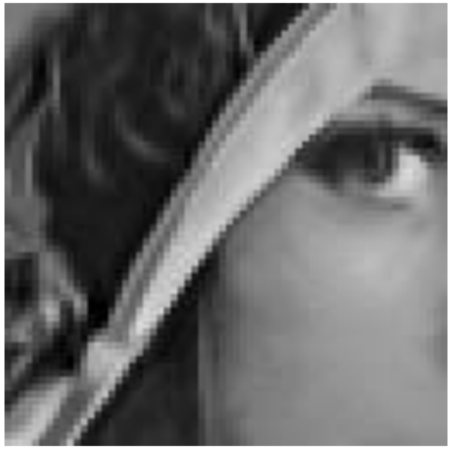
(better, but similar ‘stairsteps’ :



NL-Means Method: Buades (2005)

- NL-Means:

Sharp,
Low noise,
Few artifacts.



NL-Means Method: Buades (2005)

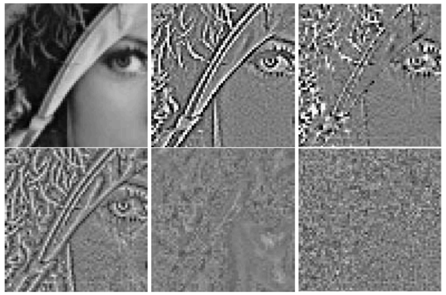
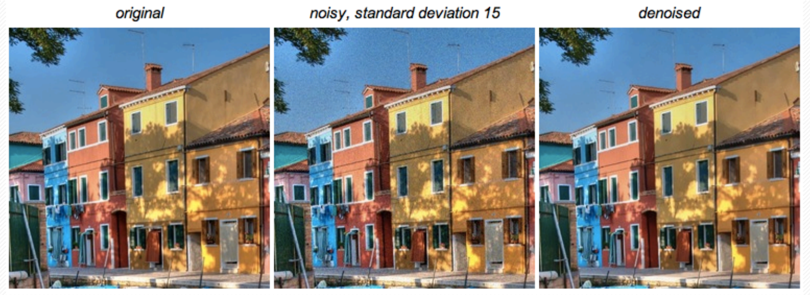


Figure 4. Method noise experience on a natural image. Displaying of the image difference $u - D_h(u)$. From left to right and from top to bottom: original image, Gauss filtering, anisotropic filtering, Total variation minimization, Neighborhood filtering and NL-means algorithm. The visual experiments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

NL-Means Method: Buades (2005)



original

noisy, standard deviation 15

denoised

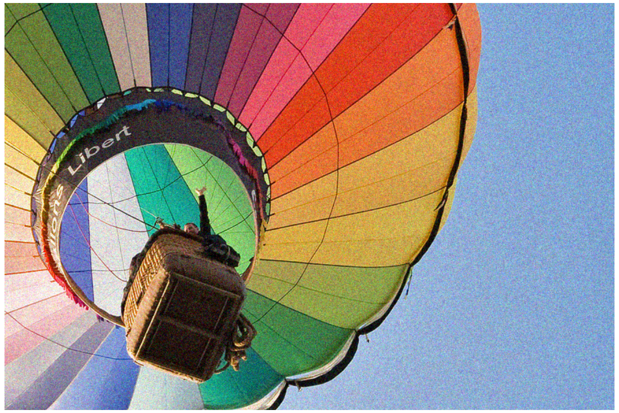
http://www.ipol.im/pub/algo/bcm_non_local_means_denoising/

NL-Means Method: Buades (2005)

original

http://www.ipol.im/pub/algo/bcm_non_local_means_denoising/

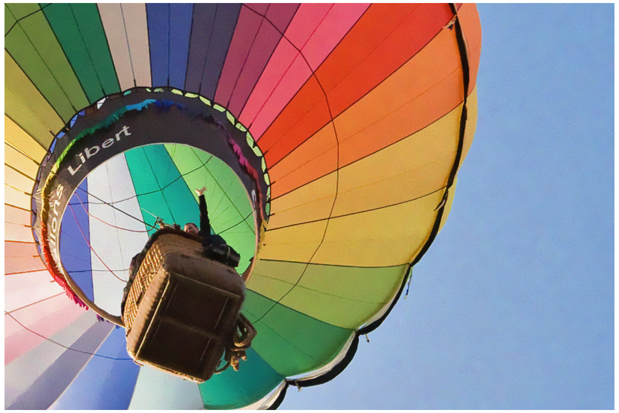
NL-Means Method: Buades (2005)



noisy

http://www.ipol.im/pub/algo/bcm_non_local_means_denoising/

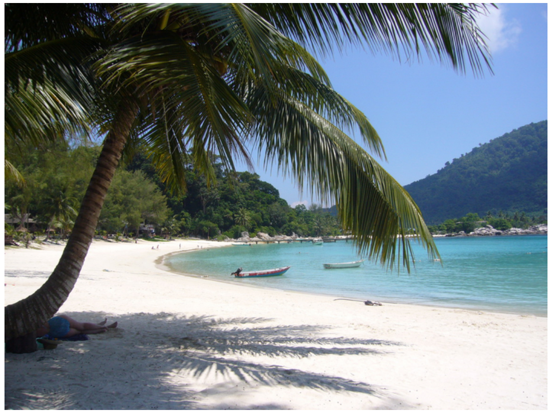
NL-Means Method: Buades (2005)



denoised

http://www.ipol.im/pub/algo/bcm_non_local_means_denoising/

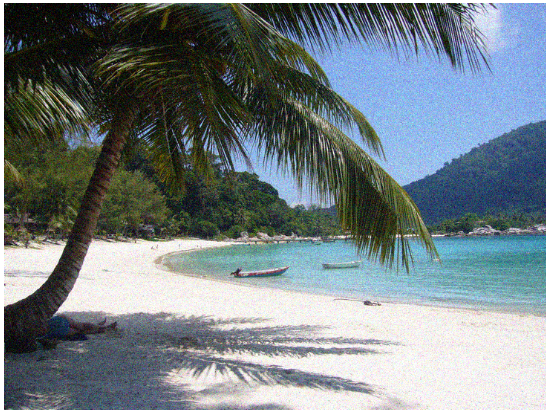
NL-Means Method: Buades (2005)



original

http://www.ipol.im/pub/algo/bcm_non_local_means_denoising/

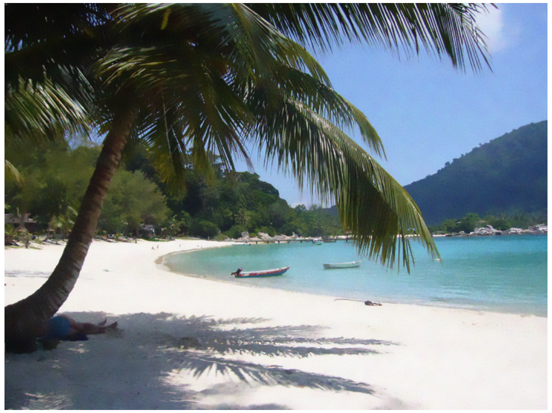
NL-Means Method: Buades (2005)



noisy

http://www.ipol.im/pub/algo/bcm_non_local_means_denoising/

NL-Means Method: Buades (2005)



denoised

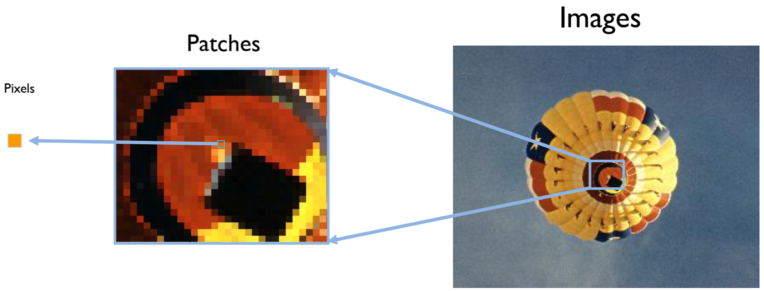
http://www.ipol.im/pub/algo/bcm_non_local_means_denoising/

Today

- Bilateral filtering
- Non-local means denoising
- LARK filter

Acknowledgement: The slides are adapted from the ones prepared by P. Milanfar.

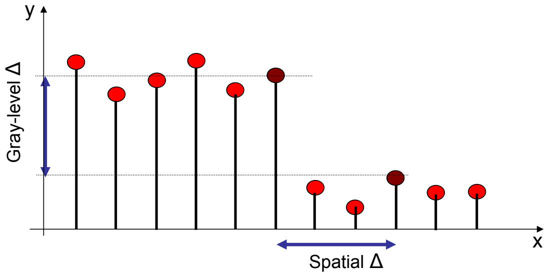
From pixels to patches and to images



Similarities can be defined at different scales..

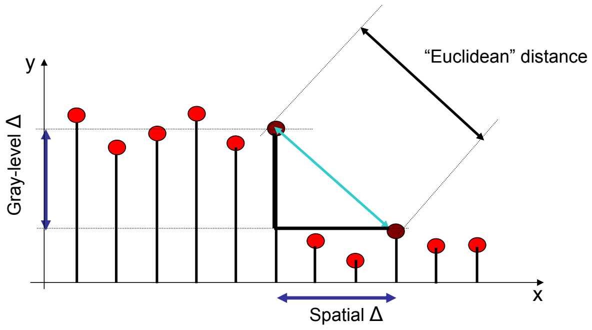
Pixelwise similarity metrics

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



Euclidean metrics

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

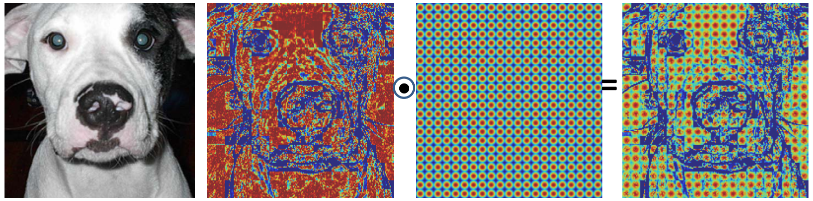


Bilateral Kernel (BL) [Tomasi et al. '98]

Pixels

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

⇩ **Pixel similarity** ⇩ **Spatial similarity**

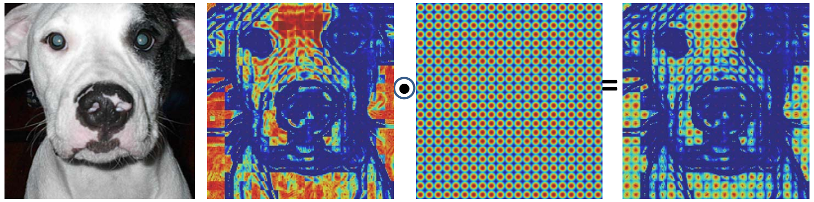


Non-local Means (NLM) [Buades et al.'05]

Patches

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

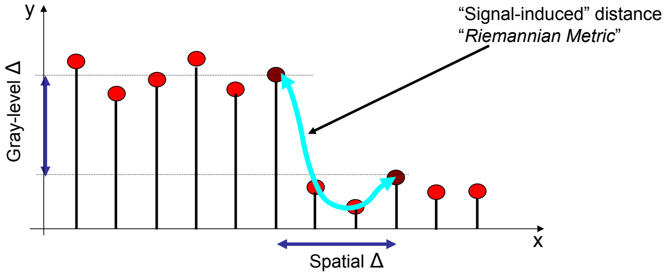
Patch similarity **Spatial similarity**



Smoothing effect

Beyond Euclidean metrics

- Better similarity measures
- More effective ways to combine the two Δ s:
  - LARK Kernel [Takeda, et al.'07]
  - Beltrami Kernel [Sochen, et al.'98]



Non-parametric Kernel Regression

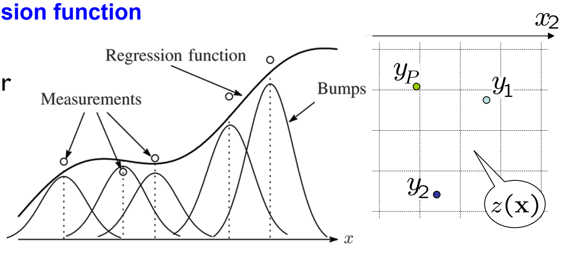
- The data fitting problem **Zero-mean, i.i.d noise (No other assump.)**

$$y_i = z(\mathbf{x}_i) + \varepsilon_i, \quad i = 1, 2, \cdots, P$$

Given samples **The sampling position** **The number of samples**

The regression function

- The particular form of z(x) may remain unspecified for now.



Locality in Kernel Regression

- The data model

$$y_i = z(\mathbf{x}_i) + \varepsilon_i, \quad i = 1, 2, \cdots, P$$

- Local representation (N-term Taylor series expansion)

$$z(x_i) \approx z(x) + z'(x)(x_i - x) + \frac{1}{2!}z''(x)(x_i - x)^2$$
$$+ \cdots + \frac{1}{N!}z^{(N)}(x)(x_i - x)^N$$
$$= \beta_0 + \beta_1(x_i - x) + \beta_2(x_i - x)^2$$
$$+ \cdots + \beta_N(x_i - x)^N.$$

- Note that with a polynomial basis, we only need to estimate the first unknown β_0

Locality in Kernel Regression

- The data model

$$y_i = z(\mathbf{x}_i) + \varepsilon_i, \quad i = 1, 2, \cdots, P$$

- Local representation (N-term Taylor series expansion)

$$\begin{aligned} z(\mathbf{x}_i) &= \boxed{z(\mathbf{x})} + \boxed{\{\nabla z(\mathbf{x})\}^T}(\mathbf{x}_i - \mathbf{x}) + \boxed{\frac{1}{2!}}(\mathbf{x}_i - \mathbf{x})^T \boxed{\{\mathcal{H} z(\mathbf{x})\}}(\mathbf{x}_i - \mathbf{x}) + \cdots \\ &= \boxed{\beta_0} + \boxed{\boldsymbol{\beta}_1^T}(\mathbf{x}_i - \mathbf{x}) + \boxed{\boldsymbol{\beta}_2^T} \text{vech}\left\{(\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T\right\} + \cdots, \end{aligned}$$

Unknowns

- Note that with a polynomial basis, we only need to estimate the first unknown β_0

Finding the unknowns via optimization

- We have a local representation with respect to each sample:

$$\begin{aligned} y_1 &= \beta_0 + \boldsymbol{\beta}_1^T(\mathbf{x}_1 - \mathbf{x}) + \boldsymbol{\beta}_2^T \text{vech}\left\{(\mathbf{x}_1 - \mathbf{x})(\mathbf{x}_1 - \mathbf{x})^T\right\} + \cdots + \varepsilon_1, \\ y_2 &= \beta_0 + \boldsymbol{\beta}_1^T(\mathbf{x}_2 - \mathbf{x}) + \boldsymbol{\beta}_2^T \text{vech}\left\{(\mathbf{x}_2 - \mathbf{x})(\mathbf{x}_2 - \mathbf{x})^T\right\} + \cdots + \varepsilon_2, \\ &\vdots \qquad\qquad\qquad\qquad \vdots \\ y_P &= \beta_0 + \boldsymbol{\beta}_1^T(\mathbf{x}_P - \mathbf{x}) + \boldsymbol{\beta}_2^T \text{vech}\left\{(\mathbf{x}_P - \mathbf{x})(\mathbf{x}_P - \mathbf{x})^T\right\} + \cdots + \varepsilon_P, \end{aligned}$$

- Estimate the parameters $\{\beta_n\}_{n=0}^N$ from the data while giving the nearby samples higher weight than samples farther away.

$$\min_{\{\beta_n\}} \sum_{i=1}^{P} \left[y_i - \beta_0 - \beta_1(x_i - x) - \beta_2(x_i - x)^2 - \cdots - \beta_N(x_i - x)^N\right]^2 \frac{1}{h} K\left(\frac{x_i - x}{h}\right)$$

Finding the unknowns via optimization

- We have a local representation with respect to each sample:

$$\begin{aligned} y_1 &= \beta_0 + \boldsymbol{\beta}_1^T(\mathbf{x}_1 - \mathbf{x}) + \boldsymbol{\beta}_2^T \text{vech}\left\{(\mathbf{x}_1 - \mathbf{x})(\mathbf{x}_1 - \mathbf{x})^T\right\} + \cdots + \varepsilon_1, \\ y_2 &= \beta_0 + \boldsymbol{\beta}_1^T(\mathbf{x}_2 - \mathbf{x}) + \boldsymbol{\beta}_2^T \text{vech}\left\{(\mathbf{x}_2 - \mathbf{x})(\mathbf{x}_2 - \mathbf{x})^T\right\} + \cdots + \varepsilon_2, \\ &\vdots \qquad\qquad\qquad\qquad \vdots \\ y_P &= \beta_0 + \boldsymbol{\beta}_1^T(\mathbf{x}_P - \mathbf{x}) + \boldsymbol{\beta}_2^T \text{vech}\left\{(\mathbf{x}_P - \mathbf{x})(\mathbf{x}_P - \mathbf{x})^T\right\} + \cdots + \varepsilon_P, \end{aligned}$$

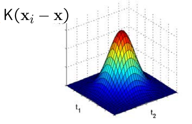
- Optimization

The regression order

$$\min_{\{\beta_n\}_{n=0}^N} \sum_{i=1}^{P} \Big[y_i \overbrace{- \beta_0 - \boldsymbol{\beta}_1^T(\mathbf{x}_i - \mathbf{x}) - \boldsymbol{\beta}_2^T \text{vech}\left\{(\mathbf{x}_i - \mathbf{x})(\mathbf{x}_i - \mathbf{x})^T\right\} - \cdots}^{\text{N+1 terms}}\Big]^2 \text{K}(\mathbf{x}_i - \mathbf{x})$$

This term give the estimated pixel value z(x).

The choice of the kernel function is open, e.g. Gaussian.



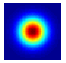
$$\boxed{\hat{z}(\mathbf{x}) = \sum_{i=1}^{P} W_i(\mathbf{x}, K, h, N)\, y_i}$$

Defining Data-Adaptive Kernels

- *Classic* Kernel: Locally Linear Filter:

$$\hat{z}(\mathbf{x}) = \hat{\beta}_0 = \textstyle\sum_i W(\mathbf{x}_i, \mathbf{x}, N)\, y_i$$

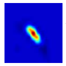
Uses distance x-x_i



- *Data-Adaptive* Kernel:
  Locally Non-Linear Filter:

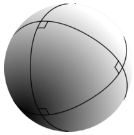
$$\hat{z}(\mathbf{x}) = \hat{\beta}_0 = \textstyle\sum_i W(\mathbf{x}_i, \mathbf{x}, y_i, y, N)\, y_i$$

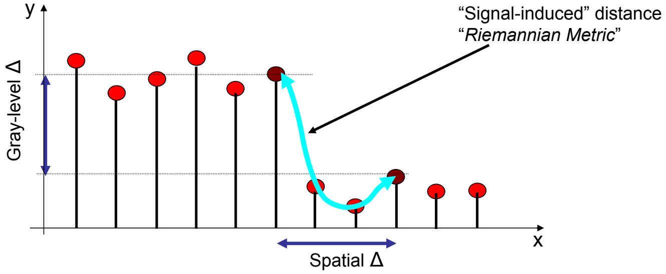
Uses x-x_i and y-y_i



Recall - Beyond Euclidean metrics

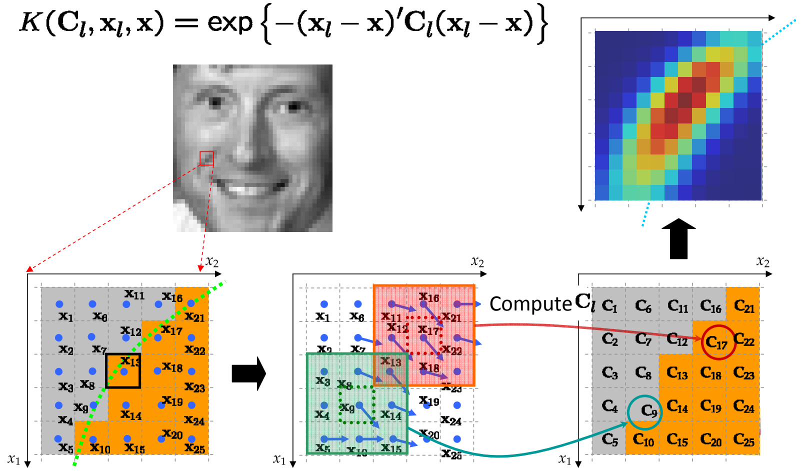
- Better similarity measures
- More effective ways to combine the two Δs:
  - LARK Kernel [Takeda, et al.'07]
  - Beltrami Kernel [Sochen, et al.'98]



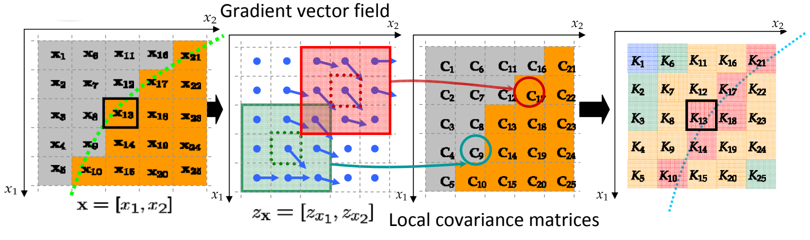


LARK Kernels

$$K(\mathbf{C}_l, \mathbf{x}_l, \mathbf{x}) = \exp\left\{-(\mathbf{x}_l - \mathbf{x})'\mathbf{C}_l(\mathbf{x}_l - \mathbf{x})\right\}$$



LARK Kernels


Gradient vector field
$\mathbf{x} = [x_1, x_2]$
$z_{\mathbf{x}} = [z_{x_1}, z_{x_2}]$
Local covariance matrices
x_1
x_2

Locally Adaptive Regression Kernel: LARK

$$K(\mathbf{C}_l, \mathbf{x}_l, \mathbf{x}) = \exp\left\{-(\mathbf{x}_l - \mathbf{x})'\underline{\mathbf{C}_l}(\mathbf{x}_l - \mathbf{x})\right\}$$

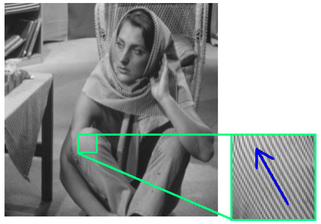
"Structure tensor"

$$\mathbf{C}_l = \sum_{k \in \boxed{\Omega_l}} \begin{bmatrix} z_{x_1}^2(\mathbf{x}_k) & z_{x_1}(\mathbf{x}_k)z_{x_2}(\mathbf{x}_k) \\ z_{x_1}(\mathbf{x}_k)z_{x_2}(\mathbf{x}_k) & z_{x_2}^2(\mathbf{x}_k) \end{bmatrix}$$

Gradient Covariance Matrix and Local Geometry

Gradient matrix over a local patch:

$$\mathbf{C}_l = \sum_{k \in \Omega_l} \begin{bmatrix} z_{x_1}^2(\mathbf{x}_k) & z_{x_1}(\mathbf{x}_k)z_{x_2}(\mathbf{x}_k) \\ z_{x_1}(\mathbf{x}_k)z_{x_2}(\mathbf{x}_k) & z_{x_2}^2(\mathbf{x}_k) \end{bmatrix}$$



$$\mathbf{C}_l = \mathbf{G}^T\mathbf{G}$$

$$\mathbf{G} = \mathbf{USV}^T = \mathbf{U}\begin{bmatrix} s_1 & 0 \\ 0 & s_2 \end{bmatrix}\begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 \end{bmatrix}^T$$

Capturing locally dominant orientations

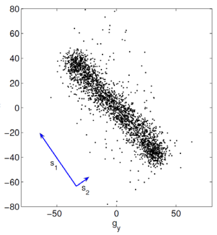
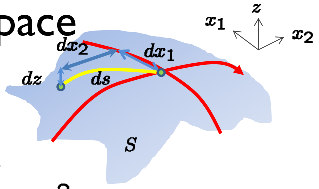


Image as a Surface Embedded in the Euclidean 3-space

$$S(x_1, x_2) = \{x_1, x_2, z(x_1, x_2)\} \in \mathbb{R}^3$$



Arclength on the surface

$$\begin{aligned} ds^2 &= dx_1^2 + dx_2^2 + dz^2 \\ &= dx_1^2 + dx_2^2 + (z_{x_1}dx_1 + z_{x_2}dx_2)^2 \\ &= (1 + z_{x_1}^2)dx_1^2 + 2z_{x_1}z_{x_2}dx_1dx_2 + (1 + z_{x_2}^2)dx_2^2 \end{aligned}$$

Chain rule

$$= (dx_1 \quad dx_2)\begin{pmatrix} 1 + z_{x_1}^2 & z_{x_1}z_{x_2} \\ z_{x_1}z_{x_2} & 1 + z_{x_2}^2 \end{pmatrix}\begin{pmatrix} dx_1 \\ dx_2 \end{pmatrix}$$

$$\Rightarrow (\mathbf{x}_l - \mathbf{x})^T(\mathbf{C}_l + \underset{\uparrow}{\mathbf{I}})(\mathbf{x}_l - \mathbf{x})$$

Riemannian metric

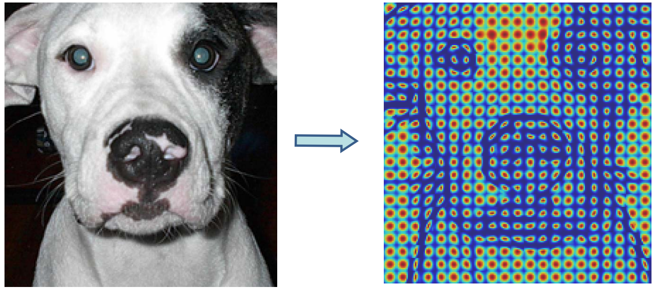
Regularization term

$$K(\mathbf{C}_l, \mathbf{x}_l, \mathbf{x}) = \exp\left\{-(\mathbf{x}_l - \mathbf{x})'\mathbf{C}_l(\mathbf{x}_l - \mathbf{x})\right\}$$

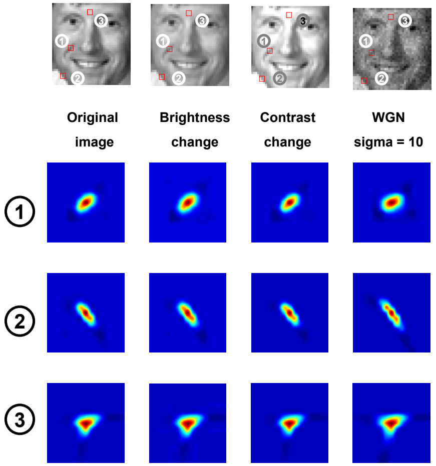
(Dense) LARK Kernels as Visual Descriptors [Seo and Milanfar '10]

$$K(\mathbf{C}_l, \mathbf{x}_l, \mathbf{x}) = \exp\left\{\underbrace{-(\mathbf{x}_l - \mathbf{x})'\mathbf{C}_l(\mathbf{x}_l - \mathbf{x})}\right\}$$

Measure the similarity of pixels using the metric implied by the local structure of the image

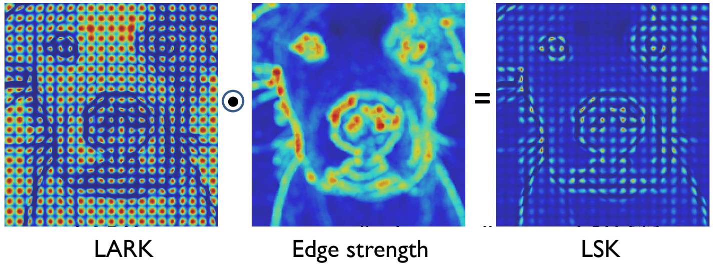


Robustness of LARK Descriptors



A Variant Better-suited for Restoration [Takeda et al. '07]

$$K(\mathbf{C}_l, \mathbf{x}_l, \mathbf{x}) = \boxed{\sqrt{\det \mathbf{C}_l}}\exp\left\{-(\mathbf{x}_l - \mathbf{x})'\mathbf{C}_l(\mathbf{x}_l - \mathbf{x})\right\}$$



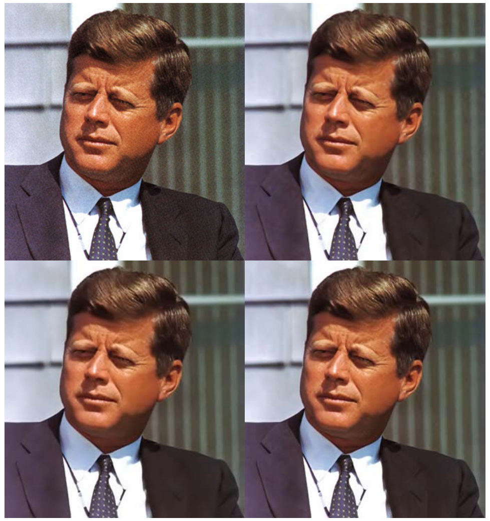
Film Grain Reduction (Real Noise)

Noisy image

Film Grain Reduction (Real Noise)

LARK

Film Grain Reduction (Real Noise)



LARK

KSVD

BM3D

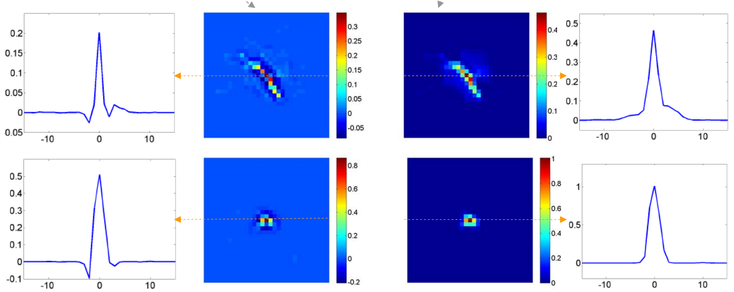
Adaptive Sharpening/Denoising

- Sharpening the LARK Kernel

"Sharpness" parameter

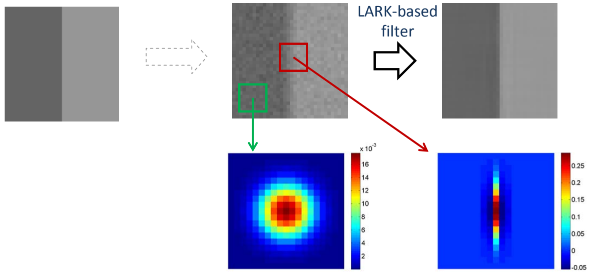
Laplacian operator

$$S = K - \kappa L \otimes K$$



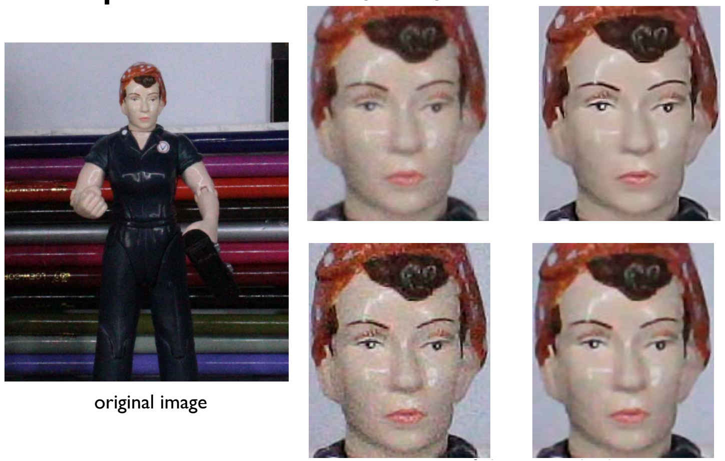
LARK-based Simultaneous Sharpening/ Deblurring/Denoising

- Net effect:
  - aggressive denoising in “flat” areas
  - Selective denoising and sharpening in “edgy” areas



Locally adaptive denoise/deblur filters

Examples



state-of-the-are methods

Examples

Examples