BSB 663
Image Processing

Erkut Erdem
Dept. of Computer Engineering
Hacettepe University

Image Smoothing

Acknowledgement: The slides are mostly 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/)

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 filter (Tomasi et al., 1998)
• NL-means filter (Buades et al., 2005)
• Structure-texture decomposition via region covariances (Karacan et al. 2013)

Notation and Definitions

• Image = 2D array of pixels
• Pixel = intensity (scalar) or color (3D vector)
• $I_p =$ value of image $I$ at position: $p = (p_x, p_y)$
• $F[I] =$ 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 as average of its neighbors

Equation of Box Average

\[ \text{BA}[I]_p = \sum_{q \in S} B_\sigma(p-q) I_q \]

Box Average

Square Box Generates Defects

- Axis-aligned streaks
- Blocky results
**Strategy to Solve these Problems**

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

![box window](image1)
![Gaussian window](image2)

**Gaussian Blur**

- Input
- Per-pixel multiplication
- Average
- Output

![box average](image3)
**Gaussian blur**

**Equation of Gaussian Blur**

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

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

Spatial Parameter

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

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

\[
GB[I]_p = \sum_{q \in S} G_{r^2}(\|p - q\|) I_q
\]

Blur Comes from
Averaging across Edges

Bilateral Filter
No Averaging across Edges

Same Gaussian kernel everywhere.

The kernel shape depends on the image content.

[Aurich 95, Smith 97, Tomasi 98]
**Bilateral Filter Definition:**

**an Additional Edge Term**

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

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

Illustration a 1D Image

- 1D image = line of pixels

- Better visualized as a plot

Illustration a 1D Image

<table>
<thead>
<tr>
<th>pixel position</th>
<th>pixel intensity</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>120</td>
<td>0.04</td>
</tr>
<tr>
<td>128</td>
<td>0.01</td>
</tr>
<tr>
<td>136</td>
<td>0.0</td>
</tr>
</tbody>
</table>

**Gaussian Blur and Bilateral Filter**

- **Gaussian blur**
  \[
  GB[I]_p = \sum_{q \in S} G_{\sigma_s}(\|p - q\|) I_q
  \]

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

Illustration a 1D Image

- Reproduced from [Durand 02]
**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 \( S_s \): 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**

<table>
<thead>
<tr>
<th>( s_s )</th>
<th>( s_r )</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>( \infty )</td>
<td>( \infty )</td>
<td>(Gaussian blur)</td>
</tr>
</tbody>
</table>

**Varying the Range Parameter**

<table>
<thead>
<tr>
<th>( s_s )</th>
<th>( s_r )</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>( \infty )</td>
<td>( \infty )</td>
<td>(Gaussian blur)</td>
</tr>
</tbody>
</table>
\[
s_r = 0.1
\]

\[
s_r = 0.25
\]

\[
s_r = \infty
\]

(Gaussian blur)
Varying the Space Parameter

\[ s_s = 2 \]

\[ s_s = 6 \]

\[ s_s = 18 \]

\( s_s = \infty \) (Gaussian blur)
**Bilateral Filter Crosses Thin Lines**

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

**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

**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

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

intensity difference

scalar

For color images

\[
BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_i} (\| p - q \|) G_{\sigma_i} (\| C_p - C_q \|) C
\]

color difference

3D vector

(RGB, Lab)

**Hard to Compute**

- Nonlinear

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

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

- Brute-force implementation is slow > 10min

**Basic denoising**

- Noisy input
- Bilateral filter 7x7 window

**Basic denoising**

- Bilateral filter
- Median 3x3
Basic denoising

Bilateral filter

Median 5x5

Bilateral filter

Bilateral filter – lower sigma

Bilateral filter

Bilateral filter – higher sigma

Bilateral filter

Basic denoising

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)
  – But the denoising feature in e.g. Photoshop is better
New Idea: NL-Means Filter (Buades 2005)

• Same goals: 'Smooth within Similar Regions'

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

---

**NL-Means Method:** Buades (2005)

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

- Define vector \( V_p \): a list of neighboring pixel values.
**NL-Means Method: Buades (2005)**

'Similar' pixels $p, q$

- SMALL vector distance;

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

'Dissimilar' pixels $p, q$

- LARGE vector distance;

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

Filter with this!

\[
NLMF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_d}( || p - q || ) G_{\sigma_v}( || V_p - V_q ||^2 ) I_q
\]
**NL-Means Method:**
*Buades (2005)*

- pixels $p, q$ neighbors
  - Set a vector distance:

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

Vector Distance to $p$ sets weight for each pixel $q$

$$NLMF(I)_p = \frac{1}{W_p} \sum_{q \in S} G_{g}(\| V_p - V_q \|) I_q$$

---

**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 Filter (Buades 2005)**

- **Gaussian Filter**
  - Low noise,
  - Low detail

**NL-Means Filter (Buades 2005)**

- **Anisotropic Diffusion**
  - (Note ‘stairsteps’: ~ piecewise constant)

**NL-Means Filter (Buades 2005)**

- **Bilateral Filter**
  - (better, but similar ‘stairsteps’)

**NL-Means Filter (Buades 2005)**

- **NL-Means:**
  - Sharp,
  - Low noise,
  - Few artifacts.
Figure 4. Method noise experience on a natural image. Displaying of the image difference $u - D_u(x)$. 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 Filter (Buades 2005)

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

noisy

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

denoised

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

original

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

denoised

http://www.ipol.im/pub/algo/bcm_non_local_means_denoising/
Structure-Texture Decomposition
Karacan et al., SIGGRAPH Asia 2013

Input Image

Structure Component

Texture Component
Region Covariances as Region Descriptors
Tuzel et al., ECCV 2006

\[ F(x, y) = \phi(I, x, y) \]

\[ F(x, y) = \begin{bmatrix} \frac{\partial I}{\partial x} & \frac{\partial I}{\partial y} & \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x \partial y} & \frac{\partial^2 I}{\partial y^2} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}^T \]

\[ C_R = \frac{1}{n-1} \sum_{i=0}^{n} (z_k - \mu)(z_k - \mu)^T \]

Formulation

\[ I = S + T \]

- Structure-texture decomposition via smoothing
- Smoothing as weighted averaging
- Different kernels \( w_{pq} \) result in different types of filters.
- Two novel patch-based kernels for structure-texture decomposition

Main motivation

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

Model I

- Covariance matrices do not live on Euclidean space.
- Hong et al., CVPR’09 suggested a way to transform covariance matrices into Euclidean Space.
- Every covariance matrix has a unique Cholesky decomposition

\[ C = LL^T \]

Cholesky Decomposition

\[ \mathcal{S} = \{ \mathbf{s}_i \} \]

Sigma Points

\[ \mathbf{s}_i = \begin{cases} \alpha \sqrt{d} L_i & \text{if } 1 \leq i \leq d \\ -\alpha \sqrt{d} L_i & \text{if } d + 1 \leq i \leq 2d \end{cases} \]

- First order statistics can be easily incorporated to the formulation.
Equation can be alternatively defined as follows:

\[ \Psi(C) = (\mu, s_1, \ldots, s_d, s_{d+1}, \ldots, s_{2d})^T \]

Final representation

\[ w_{pq} \propto \exp \left( -\frac{\|\Psi(C_p) - \Psi(C_q)\|^2}{2\sigma^2} \right) \]

Resulting kernel

**Model 1**

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

\[ d(p, q) = \sqrt{(\mu_p - \mu_q)^T C^{-1} (\mu_p - \mu_q)^T} \]

\[ C = C_p + C_q \]

Resulting kernel

\[ w_{pq} \propto \exp \left( -\frac{d(p, q)^2}{2\sigma^2} \right) \]

**Model 2**

**Smoothing Kernels**

Model 1 & Model 2

**Illustrative Example**

Input

Model 1 & Model 2
A naive implementation of our structure preserving image smoothing equation can be alternatively defined as follows:

Equation (3) defines a distance measure, which can be seen as an approximation to the similarity between two image pixels. This distance measure is defined as

\[ d(p, q) = \sqrt{\sum_{i=1}^{d} (p_i - q_i)^2} \]

where \( d \) is the dimensionality of the feature vectors, and \( p_i \) and \( q_i \) are the components of the feature vectors at pixel locations \( p \) and \( q \), respectively.

Using the set of pixel locations \( P \) and feature vectors \( C \), we can compute the covariance descriptors for each pixel in \( P \). This enriched feature vector can be defined as

\[ C(p) = \left[ \mu(p), \mu'(p), \mu''(p), \sigma(p), \sigma'(p), \sigma''(p) \right] \]

where \( \mu(p) \), \( \mu'(p) \), and \( \mu''(p) \) are the mean, first and second derivatives of the intensity in the patch centered at pixel location \( p \), respectively, and \( \sigma(p) \), \( \sigma'(p) \), and \( \sigma''(p) \) are the standard deviations, first and second derivatives of the standard deviation in the patch centered at pixel location \( p \), respectively.

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.

The weighted average over the corresponding RGB vectors rather than \( C(p) \) can be used to define the similarity weights.

The decompositions of the input image can be represented by the set of structure components at each iteration as an input for the next iteration. The input image can be expressed as the sum of structure and texture components:

\[ I(p) = \sum_{i=0}^{n} S_i(p) + T_i(p) \]

where \( I(p) \) is the intensity at pixel location \( p \), and \( S_i(p) \) and \( T_i(p) \) are the structure and texture components, respectively.

For each iteration \( i \), the structure component can be defined as

\[ S_i(p) = \sum_{k=0}^{i} w_k(p) T_k(p) \]

where \( w_k(p) \) is the weight at pixel location \( p \) for iteration \( k \).

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.

The decompositions of the input image can be represented by the set of structure components at each iteration as an input for the next iteration. The input image can be expressed as the sum of structure and texture components:

\[ I(p) = \sum_{i=0}^{n} S_i(p) + T_i(p) \]

where \( I(p) \) is the intensity at pixel location \( p \), and \( S_i(p) \) and \( T_i(p) \) are the structure and texture components, respectively.

For each iteration \( i \), the structure component can be defined as

\[ S_i(p) = \sum_{k=0}^{i} w_k(p) T_k(p) \]

where \( w_k(p) \) is the weight at pixel location \( p \) for iteration \( k \).

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.

The decompositions of the input image can be represented by the set of structure components at each iteration as an input for the next iteration. The input image can be expressed as the sum of structure and texture components:

\[ I(p) = \sum_{i=0}^{n} S_i(p) + T_i(p) \]

where \( I(p) \) is the intensity at pixel location \( p \), and \( S_i(p) \) and \( T_i(p) \) are the structure and texture components, respectively.

For each iteration \( i \), the structure component can be defined as

\[ S_i(p) = \sum_{k=0}^{i} w_k(p) T_k(p) \]

where \( w_k(p) \) is the weight at pixel location \( p \) for iteration \( k \).

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.

The decompositions of the input image can be represented by the set of structure components at each iteration as an input for the next iteration. The input image can be expressed as the sum of structure and texture components:

\[ I(p) = \sum_{i=0}^{n} S_i(p) + T_i(p) \]

where \( I(p) \) is the intensity at pixel location \( p \), and \( S_i(p) \) and \( T_i(p) \) are the structure and texture components, respectively.

For each iteration \( i \), the structure component can be defined as

\[ S_i(p) = \sum_{k=0}^{i} w_k(p) T_k(p) \]

where \( w_k(p) \) is the weight at pixel location \( p \) for iteration \( k \).

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.

The decompositions of the input image can be represented by the set of structure components at each iteration as an input for the next iteration. The input image can be expressed as the sum of structure and texture components:

\[ I(p) = \sum_{i=0}^{n} S_i(p) + T_i(p) \]

where \( I(p) \) is the intensity at pixel location \( p \), and \( S_i(p) \) and \( T_i(p) \) are the structure and texture components, respectively.

For each iteration \( i \), the structure component can be defined as

\[ S_i(p) = \sum_{k=0}^{i} w_k(p) T_k(p) \]

where \( w_k(p) \) is the weight at pixel location \( p \) for iteration \( k \).

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.

The decompositions of the input image can be represented by the set of structure components at each iteration as an input for the next iteration. The input image can be expressed as the sum of structure and texture components:

\[ I(p) = \sum_{i=0}^{n} S_i(p) + T_i(p) \]

where \( I(p) \) is the intensity at pixel location \( p \), and \( S_i(p) \) and \( T_i(p) \) are the structure and texture components, respectively.

For each iteration \( i \), the structure component can be defined as

\[ S_i(p) = \sum_{k=0}^{i} w_k(p) T_k(p) \]

where \( w_k(p) \) is the weight at pixel location \( p \) for iteration \( k \).

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.

The decompositions of the input image can be represented by the set of structure components at each iteration as an input for the next iteration. The input image can be expressed as the sum of structure and texture components:

\[ I(p) = \sum_{i=0}^{n} S_i(p) + T_i(p) \]

where \( I(p) \) is the intensity at pixel location \( p \), and \( S_i(p) \) and \( T_i(p) \) are the structure and texture components, respectively.

For each iteration \( i \), the structure component can be defined as

\[ S_i(p) = \sum_{k=0}^{i} w_k(p) T_k(p) \]

where \( w_k(p) \) is the weight at pixel location \( p \) for iteration \( k \).

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.

The decompositions of the input image can be represented by the set of structure components at each iteration as an input for the next iteration. The input image can be expressed as the sum of structure and texture components:

\[ I(p) = \sum_{i=0}^{n} S_i(p) + T_i(p) \]

where \( I(p) \) is the intensity at pixel location \( p \), and \( S_i(p) \) and \( T_i(p) \) are the structure and texture components, respectively.

For each iteration \( i \), the structure component can be defined as

\[ S_i(p) = \sum_{k=0}^{i} w_k(p) T_k(p) \]

where \( w_k(p) \) is the weight at pixel location \( p \) for iteration \( k \).

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.

The decompositions of the input image can be represented by the set of structure components at each iteration as an input for the next iteration. The input image can be expressed as the sum of structure and texture components:

\[ I(p) = \sum_{i=0}^{n} S_i(p) + T_i(p) \]

where \( I(p) \) is the intensity at pixel location \( p \), and \( S_i(p) \) and \( T_i(p) \) are the structure and texture components, respectively.

For each iteration \( i \), the structure component can be defined as

\[ S_i(p) = \sum_{k=0}^{i} w_k(p) T_k(p) \]

where \( w_k(p) \) is the weight at pixel location \( p \) for iteration \( k \).

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.

The decompositions of the input image can be represented by the set of structure components at each iteration as an input for the next iteration. The input image can be expressed as the sum of structure and texture components:

\[ I(p) = \sum_{i=0}^{n} S_i(p) + T_i(p) \]

where \( I(p) \) is the intensity at pixel location \( p \), and \( S_i(p) \) and \( T_i(p) \) are the structure and texture components, respectively.

For each iteration \( i \), the structure component can be defined as

\[ S_i(p) = \sum_{k=0}^{i} w_k(p) T_k(p) \]

where \( w_k(p) \) is the weight at pixel location \( p \) for iteration \( k \).

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.

The decompositions of the input image can be represented by the set of structure components at each iteration as an input for the next iteration. The input image can be expressed as the sum of structure and texture components:

\[ I(p) = \sum_{i=0}^{n} S_i(p) + T_i(p) \]

where \( I(p) \) is the intensity at pixel location \( p \), and \( S_i(p) \) and \( T_i(p) \) are the structure and texture components, respectively.

For each iteration \( i \), the structure component can be defined as

\[ S_i(p) = \sum_{k=0}^{i} w_k(p) T_k(p) \]

where \( w_k(p) \) is the weight at pixel location \( p \) for iteration \( k \).

The set of texture elements can be extended to include other features, like for example rotationally invariant forms of the derivatives, if desired.
More specifically, for two image pixels $p$ and $q$ with respect to first and second-order feature statistics, available in the project website. Our code is publicly available.

A naive implementation of our structure preserving image smoothing algorithm is summarized in Algorithm 1. Our code is publicly available.

\[ I(p) = \sum_{i=0}^{n} T_i(p) + S_n(p) \]

\[ S_1(k = 5) \]

\[ S_2(k = 7) \]

\[ S_3(k = 9) \]

\[ S_2(k = 9) \]

\[ Model 2 + Model 1\]
A new measure based on KL-divergence

Input

Model2

KL-Divergence

suggested by Rahul Narain (Berkeley)