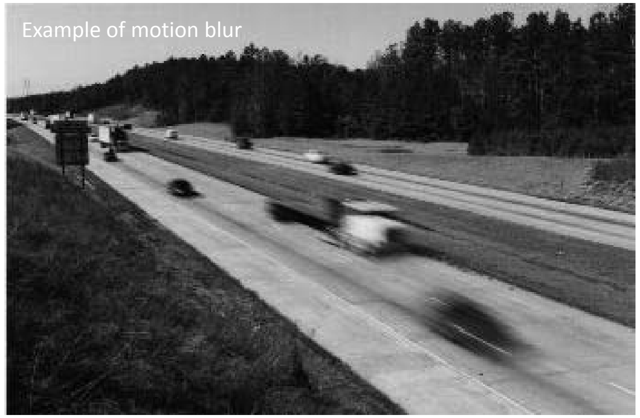
TSBB15
Computer Vision

Lecture 14
Image enhancement

Why image enhancement?

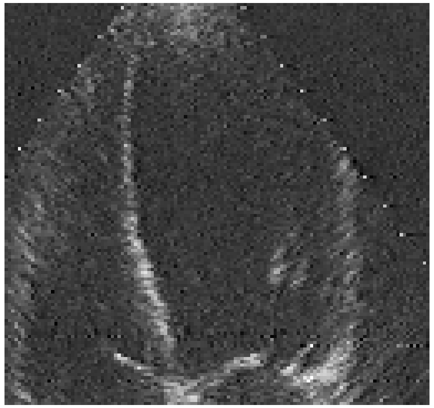
- Example of artifacts caused by image encoding

Why image enhancement?



Why image enhancement?

- Example of an image with sensor noise
  - ultrasound image of a beating heart



Why image enhancement?

- IR-image
  - fixed pattern noise = spatial variations in gain and offset
  - Possibly even variations over time!
  - Hot/dead pixels
- A digital camera with short exposure time
  - Shot noise (photon noise)

Methods for image enhancement

- Inverse filtering: the distortion process is modeled and estimated (e.g. motion blur) and the *inverse* process is applied to the image
- Image restoration: an *objective* quality (e.g. sharpness) is estimated in the image. The image is modified to increase the quality
- Image enhancement: modify the image to improve the visual quality, often with a subjective criterion

Additive noise

- Some types of image distortion can be described as
  - Noise added on each pixel intensity
  - The noise has the identical distribution and is independent at each pixel (i.i.d.)
- Not all type of image distortion are of this type:
  - Multiplicative noise
  - Data dependent noise
  - Position dependent
- The methods discussed here assume additive i.i.d.-noise

What about pixel shot noise?

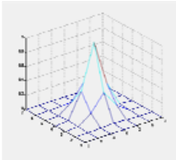
Removing additive noise

- Image noise typically contains higher frequencies than images generally do

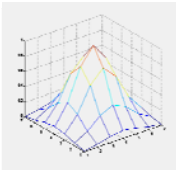
  $\Rightarrow$ a low-pass filter can reduce the noise
- BUT: we also remove high-frequency signal components, e.g. at edges and lines
- HOWEVER: A low-pass filter works in regions without edges and lines

Example: LP filter

Image with some noise



Filter, $\sigma = 1$



Filter, $\sigma = 2$

Basic idea

The problem of low-pass filters is that we apply the same filter on the whole image

We need a filter that locally **adapts** to the image **structures**

A space variant filter

Ordinary filtering / convolution

- Ordinary filtering can be described as a convolution of the signal *f* and the filter *g*:

$$h(\mathbf{x}) = (f * g)(\mathbf{x}) = \int f(\mathbf{x} - \mathbf{y})\, g(\mathbf{y})\, d\mathbf{y}$$

For each **x**, we compute the integral between the filter *g* and a shifted signal *f*

Adaptive filtering

- If we apply an adaptive (or position dependent, or space variant) filter $g_{\mathbf{x}}$, the operation cannot be expressed as a convolution, but instead as

$$h(\mathbf{x}) = \int f(\mathbf{x} - \mathbf{y})\, g_{\mathbf{x}}(\mathbf{y})\, d\mathbf{y}$$

For each **x**, we compute the integral between a shifted signal *f* and the filter $g_{\mathbf{x}}$ where the filter depends on **x**

How to choose $g_{\mathbf{x}}$?

According to the previous discussion,
we choose $g_{\mathbf{x}}$ such that:

– It contains a low-pass component that maintains the local image mean intensity

Independent of **x**

– It contains a high-pass component that depends on the local signal structure

Dependent of **x**

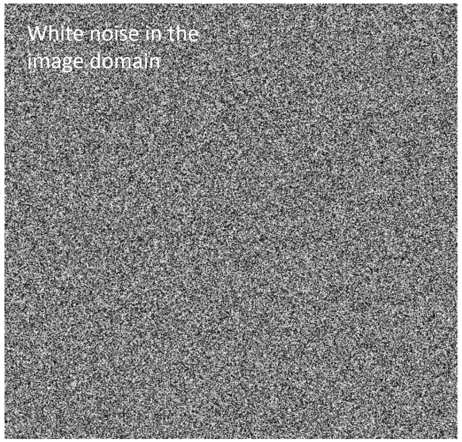
– Also: the resulting operation for computing *h* should be simple to implement

Computational efficient

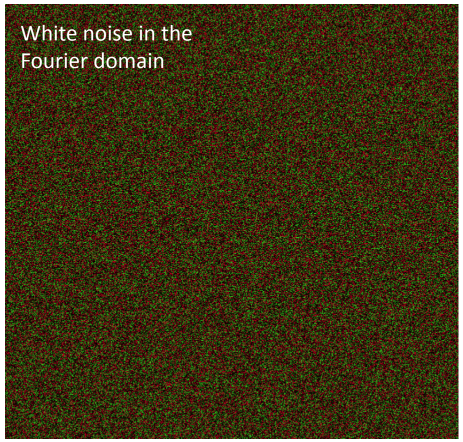
High-frequency components in $g_{\mathbf{x}}$

- If the signal is ≈ i1D the filter can maintain the signal by reducing the frequency components orthogonal to the local structure
- The human visual system is less sensitive to noise along linear structures than to noise in the orthogonal direction
- Results in good subjective improvement of image quality

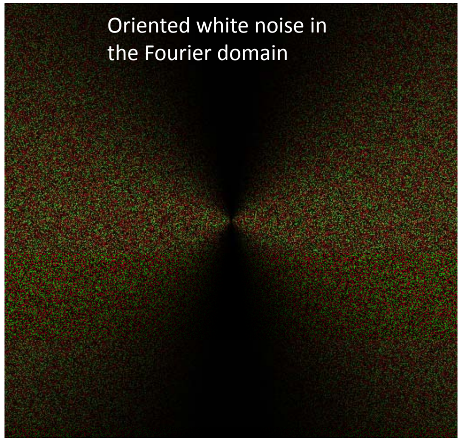
Oriented noise



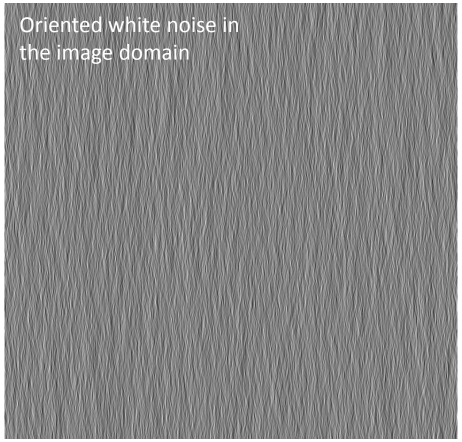
Oriented noise



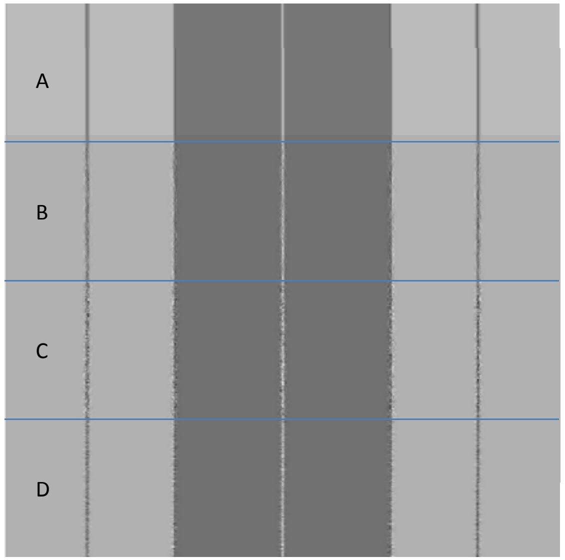
Oriented noise



Oriented noise



Oriented noise



Edges and lines

A. Without noise

B. With oriented noise along

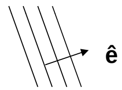
C. With isotropic noise

D. With oriented noise across

Local structure information

- We compute the local orientation tensor **T**(**x**) at all points **x** to control / steer $g_{\mathbf{x}}$
- At a point **x** that lies in a locally i1D region, we obtain

$$\mathbf{T}(\mathbf{x}) = A\,\hat{\mathbf{e}}\hat{\mathbf{e}}^T$$



ê is normal to the linear structure

Ansatz for $g_{\mathbf{x}}$

We apply a filter that is given in the Fourier domain as

$$G_{HP}(\mathbf{u}) = G_\rho(u)\,(\hat{\mathbf{u}}^T\hat{\mathbf{e}})^2 \qquad \mathbf{u} = u\,\hat{\mathbf{u}}$$

– G_{HP} is *polar separable*
– It attenuates frequency components that are $\perp$ to $\hat{\mathbf{e}}$
– It maintains all frequency components that are || to $\hat{\mathbf{e}}$

How to implement $g_{\mathbf{x}}$?

- We know that [EDUPACK – ORIENTATION]

$$(\hat{\mathbf{u}}^T\hat{\mathbf{e}})^2 = \langle \hat{\mathbf{u}}\hat{\mathbf{u}}^T | \hat{\mathbf{e}}\hat{\mathbf{e}}^T \rangle = \langle \hat{\mathbf{u}}\hat{\mathbf{u}}^T | \mathbf{T}(\mathbf{x}) \rangle$$

where $\mathbf{T}(\mathbf{x}) = \hat{\mathbf{e}}\hat{\mathbf{e}}^T$ (assume A = 1!)

- Using a N-D tensor basis $\hat{\mathbf{N}}_k = \hat{\mathbf{n}}_k\hat{\mathbf{n}}_k^T$ and its dual $\tilde{\mathbf{N}}_k$, we obtain:

$$\mathbf{T}(\mathbf{x}) = \sum_{k=1}^N \langle \mathbf{T}(\mathbf{x}) | \tilde{\mathbf{N}}_k \rangle\, \hat{\mathbf{N}}_k$$

How to implement $g_{\mathbf{x}}$?

$$\begin{aligned}(\hat{\mathbf{u}}^T\hat{\mathbf{e}})^2 &= \sum_{k=1}^N \langle \hat{\mathbf{u}}\hat{\mathbf{u}}^T | \hat{\mathbf{N}}_k \rangle\, \langle \mathbf{T}(\mathbf{x}) | \tilde{\mathbf{N}}_k \rangle \\ &= \sum_{k=1}^N \langle \hat{\mathbf{u}}\hat{\mathbf{u}}^T | \hat{\mathbf{n}}_k\hat{\mathbf{n}}_k^T \rangle\, \langle \mathbf{T}(\mathbf{x}) | \tilde{\mathbf{N}}_k \rangle \\ &= \sum_{k=1}^N \underbrace{(\hat{\mathbf{u}}^T\hat{\mathbf{n}}_k)^2}_{\text{depends on } \mathbf{u} \text{ but not on } \hat{\mathbf{e}}}\, \underbrace{\langle \mathbf{T}(\mathbf{x}) | \tilde{\mathbf{N}}_k \rangle}_{\text{depends on } \hat{\mathbf{e}} \text{ but not on } \mathbf{u}}\end{aligned}$$

depends on **u** but not on **ê**

depends on **ê** but not on **u**

How to implement $g_{\mathbf{x}}$?

- Plug this into the expression for G_{HP} :

$$G_{HP}(\mathbf{u}) = \sum_{k=1}^N \underline{\langle \mathbf{T}(\mathbf{x}) | \tilde{\mathbf{N}}_k \rangle}\; \underline{G_\rho(u)\,(\hat{\mathbf{u}}^T\hat{\mathbf{n}}_k)^2}$$

depends on **ê** but not on **u**

depends on **u** but not on **ê**

How to implement $g_{\mathbf{x}}$?

Consequently, the filter G_{HP} is a **linear combination** of N filters, where each filter has a Fourier transform:

$$G_{HP,k}(\mathbf{u}) = G_\rho(u)\,(\hat{\mathbf{u}}^T \hat{\mathbf{n}}_k)^2$$

Independent of **x**

and N scalars:

$$\langle \mathbf{T}(\mathbf{x}) | \tilde{\mathbf{N}}_k \rangle$$

Dependent of **x**

How to implement $g_{\mathbf{x}}$?

Summarizing, the adaptive filter can be written as

$$g_{\mathbf{x}} = g_{LP} + \sum_{k=1}^{N} \langle \mathbf{T}(\mathbf{x}) | \tilde{\mathbf{N}}_k \rangle\, g_{HP,k}$$

A fixed LP-filter

N fixed HP-filters

N position dependent scalars

How to implement $g_{\mathbf{x}}$?

If the filter is applied to a signal, we obtain

$$\begin{aligned} h(\mathbf{x}) &= \int f(\mathbf{x}-\mathbf{y}) \left[g_{LP}(\mathbf{y}) + \sum_{k=1}^{N} \langle \mathbf{T}(\mathbf{x}) | \tilde{\mathbf{N}}_k \rangle\, g_{HP,k}(\mathbf{y}) \right] d\mathbf{y} \\ &= (f * g_{LP})(\mathbf{x}) + \sum_{k=1}^{N} \langle \mathbf{T}(\mathbf{x}) | \tilde{\mathbf{N}}_k \rangle\, (f * g_{HP,k})(\mathbf{x}) \end{aligned}$$

Standard convolution

Standard convolutions

Position dependent scalars

Outline of method, version 1

1. Estimate the local orientation tensor **T**(**x**) at each image point **x**
2. Apply a number of fixed filters to the image; one LP-filter g_{LP} and the N HP-filters $g_{HP,k}$
3. At each point **x**:
   1. Compute the N scalars $\langle \mathbf{T}(\mathbf{x}) | \tilde{\mathbf{N}}_k \rangle$
   2. Form the linear combination of the N HP-filter responses and the N scalars and add the LP-filter response
4. At each point **x**, the result is the filter response $h(\mathbf{x})$ of the locally adapted filter $g_{\mathbf{x}}$

The filter $g_{\mathbf{x}}$ is also called a *steerable filter*

Observation

- **T** can be estimated for any image dimension
- The filters g_{LP} and $g_{HP,k}$ can be formulated for any image dimension

$\Rightarrow$ The method can be implemented for any dimension of the signal (2D, 3D, 4D, …)

Remaining questions

1. What happens in regions that are not i1D, i.e., if **T** has not rank 1?
2. What happens if $A \neq 1$?
3. How to choose the radial function G_ρ?

Non i1D signals

- The tensor's eigenvectors with non-zero eigenvalues span the subspace of the Fourier domain that contains the signal's energy
- Equivalent: For a given local region with orientation tensor **T**, let $\hat{\mathbf{u}}$ define an arbitrary orientation. The product $\hat{\mathbf{u}}^T\mathbf{T}\,\hat{\mathbf{u}}$ is a measure of how much of this orientation the region contains.

Non i1D signals

- But

$$\boxed{\hat{\mathbf{u}}^T\mathbf{T}\,\hat{\mathbf{u}} = \langle \hat{\mathbf{u}}\hat{\mathbf{u}}^T | \mathbf{T} \rangle}$$

which means that the adaptive filtering should work in general, even if the signal is non i1D

How about A = 1?

- Previously we assumed $A = 1$, but normally A depends on the local amplitude of the signal (depends on $\mathbf{x}$)
- In order to achieve $A = 1$, **T** must be pre-processed
- The resulting tensor is called the control tensor **C**

Pre-processing of **T**

- The filter $g_{\mathbf{x}}$ is supposed to vary slowly with **x**, but **T** contains high-frequency noise that comes from the image noise
- This noise can be reduced by an initial LP-filtering of **T** (i.e., of its elements)
- The result is denoted $\mathbf{T}_{\mathrm{LP}}$

Pre-processing of **T**

$\mathbf{T}_{\mathrm{LP}}$ must then be *normalized*:

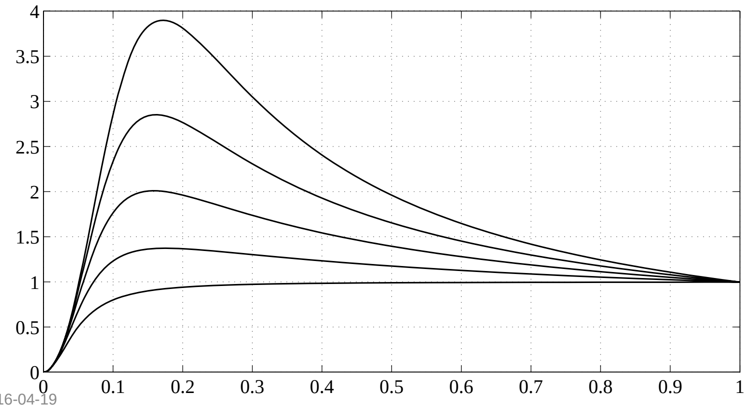
Eigen-decomposition of $\mathbf{T}_{\mathrm{LP}}$

$$\begin{aligned}
\mathbf{T}_{LP} &= \sum_{k=1}^{n} \lambda_k \,\hat{\mathbf{e}}_k \hat{\mathbf{e}}_k^T \qquad \lambda_k \geq \lambda_{k+1} \\
\mathbf{C} &= \sum_{k=1}^{n} \gamma_k \,\hat{\mathbf{e}}_k \hat{\mathbf{e}}_k^T \qquad \gamma_k \geq \gamma_{k+1} \\
\gamma_k &= \gamma_k(\lambda_1, \ldots, \lambda_n)
\end{aligned}$$

Same eigenvectors as $\mathbf{T}_{\mathrm{LP}}$, but different eigenvalues

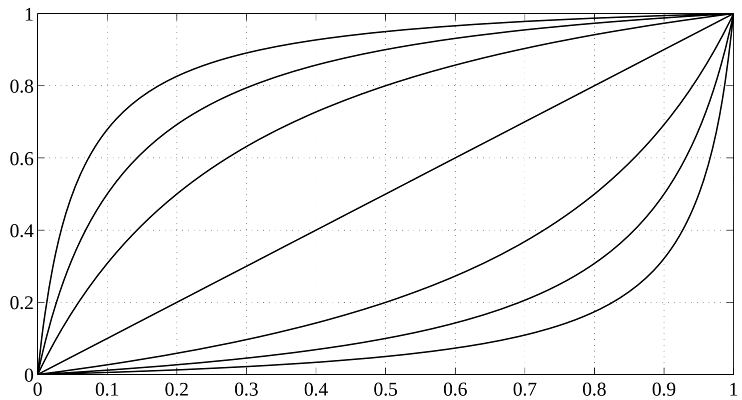
Modification of the eigenvalues

Examples of γ_1 as funcion of, e.g., $\|\mathbf{T}\| = \sqrt{\lambda_1^2 + \lambda_2^2 + \ldots}$



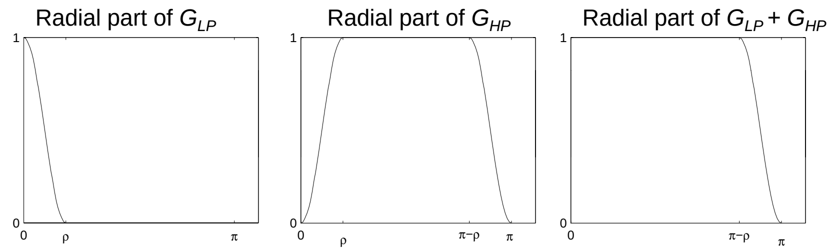
Modification of the eigenvalues

Examples of γ_{k+1}/γ_k as function of λ_{k+1}/λ_k

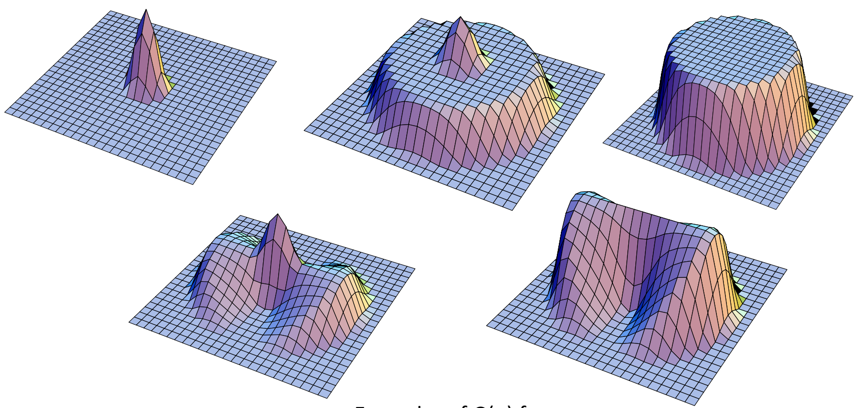


The radial function G_ρ

- Should "mainly" be equal to 1
- Should tend to 0 for u = π
- Together with the LP-filter g_{LP}: an all-pass filter



The adaptive filter in 2D



Examples of G(**u**) for different **C**(**x**)

Outline of method, version 2

1. Estimate the local tensor in each image point: **T**(**x**)
2. LP-filter the tensor: $\mathbf{T}_{LP}(\mathbf{x})$
3. In each image point:
    1. Compute the eigenvalues and eigenvectors of $\mathbf{T}_{LP}(\mathbf{x})$.
    2. Map the eigenvalues λ_k to γ_k.
    3. Re-combine γ_k and the eigenvectors to form the control tensor **C**
    4. Compute the scalars $\langle \mathbf{C} | \tilde{\mathbf{N}}_k \rangle$ for all $k = 1, \ldots, N$
4. Filter the image with g_{LP} and the N HP-filters $g_{HP,k}$
5. In each image point: form the linear combination of the filter responses and the scalars

Example

Original noisy image

Image after enhancement

Example

QuickTime™ and a
YUV420 codec decompressor
are needed to see this picture.

An iterative method

- Adaptive filtering can be iterated for reducing the noise
- If the filter size is reduced at the same time, a close-to continuous transition is achieved (evolution)
- This leads to another method for image enhancement: *anisotropic diffusion*

Scale space recap (from lecture 2)

- The linear Gaussian scale space related to the image f is a family of images $L(x,y;s)$

$$L(x,y;s) = (g_s * f)(x,y)$$

Convolution over (x,y) only!

parameterized by the scale parameter s, where

$$g_s(x,y) = \frac{1}{2\pi s} e^{-\frac{1}{2s}(x^2+y^2)}$$

A Gaussian LP-filter with $\sigma^2 = s$

Note: $g_s(x,y) = \delta(x,y)$ for $s = 0$

Scale space recap (from lecture 2)

- *L*(*x,y;s*) can also be seen as the solution to the PDE

The diffusion equation

Example:

L = temperature

s = time

$$\frac{\partial}{\partial s}L = \frac{1}{2}\nabla^2 L$$

$$\frac{\partial}{\partial s}L = \frac{1}{2}\left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)L$$

with boundary condition *L*(*x,y;*0) = *f*(*x,y*)

Image enhancement based on linear diffusion

- This means that *L*(*x,y;s*) is an LP-filtered version of *f*(*x,y*) for *s* > 0.
- The larger *s* is, the more LP-filtered is *f*
  - High-frequency noise will be removed for larger *s*
- As before: also high-frequency image components (e.g. edges) will be removed
- We need to control the diffusion process such that edges remain
  - How?

Step 1

- Modify the PDE by introducing a parameter μ:

$$\frac{\partial}{\partial s}L = \frac{\mu}{2}\nabla^2 L$$

μ can be seen as a "diffusion speed":

Small μ: the diffusion process is slow when *s* increases

Large μ: the diffusion process is fast when *s* increases

- This PDE is solved by

$$L(x, y; s) = (g_s * f)(x, y)$$

Same as before

$$g_s(x, y) = \frac{1}{2\pi\mu s} e^{-\frac{1}{2\mu s}(x^2+y^2)}$$

Slightly different

Step 2

- We want the image content to control μ
  - In flat regions: fast diffusion (large μ)
  - In non-flat region: slow diffusion (small μ)
- We need to do *space variant* diffusion
  - μ is a function of position (*x,y*)

Compare to the space variant filter g_x in adaptive filtering

Inhomogeneous diffusion

- Perona & Malik suggested to use

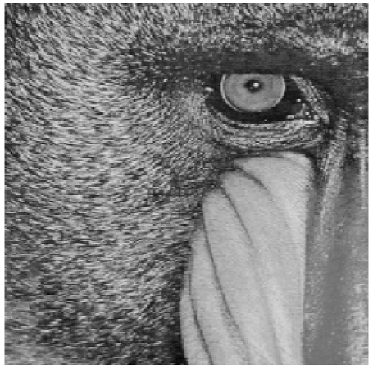
$$\mu(x,y) = \frac{1}{1+|\nabla f|^2/\lambda^2}$$

where ∇f is the image gradient at (*x*,*y*) and λ is fixed a parameter

  – Close to edges: $|\nabla f|$ is large $\Rightarrow$ μ is small
  – In flat regions: $|\nabla f|$ is small $\Rightarrow$ μ is large

Inhomogeneous diffusion

Original image



Edge Enhancing Diffusion

Inhomogeneous diffusion

- Example

Inhomogeneous diffusion

- Noise is effectively removed in flat region 
- Edges are preserved 

- Noise is preserved close to edges 

We want to be able to LP-filter along but not across edges, same as for adaptive filtering

Step 3

- The previous PDEs are all isotropic
  - $\Rightarrow$ The resulting filter *g* is isotropic
- The last PDE can be written:

$$\boxed{\frac{\partial}{\partial s}L = \frac{\mu}{2}\nabla^2 L = \frac{1}{2}\mathrm{div}(\mu \;\mathrm{grad}\; L)}$$

Gradient of *L*, a 2D vector field

Divergence of (...) maps 2D vector field to scalar field

Step 3

- Change μ from a scalar to a 2 $\times$ 2 symmetric matrix **D**

$$\frac{\partial}{\partial s}L = \frac{1}{2}\mathrm{div}(\mathbf{D}\;\mathrm{grad}\; L)$$

- The solution is now given by

$$L(\mathbf{x}; s) \;= (g_s * f)(\mathbf{x})$$

$\Leftarrow$ Same as before

$$g_s(\mathbf{x}) = \frac{1}{2\pi \det(\mathbf{D})^{1/2} s}\, e^{-\frac{1}{2s}\mathbf{x}^T\mathbf{D}^{-1}\mathbf{x}}$$

Ansiotropic diffusion

- The filter *g* is now anisotropic, i.e., not necessary circular symmetric
- The shape of *g* depends on eigensystem of **D**
- **D** is called a *diffusion tensor*
  - Can be given a physical interpretation, e.g. for anisotropic heat diffusion

The diffusion tensor

- Since **D** is symmetric 2 $\times$ 2:

$$\boxed{\mathbf{D} = \alpha_1\mathbf{e}_1\mathbf{e}_1^T + \alpha_2\mathbf{e}_2\mathbf{e}_2^T}$$

where α_1, α_2 are the eigenvalues of **D**, and $\mathbf{e}_1$ and $\mathbf{e}_2$ are corresponding eigenvectors

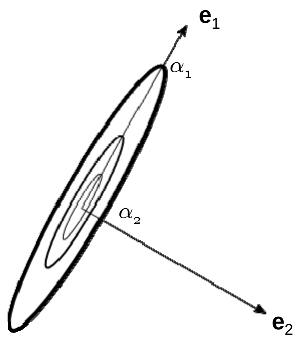
$\mathbf{e}_1$ and $\mathbf{e}_2$ form an ON-basis

The filter *g*

- The corresponding shape of *g* is given by

The width of the filter in direction $\mathbf{e}_k$ is given by α_k

Iso-curves for *g* ⇒



Step 4

- We want *g* to be narrow across edges and wide along edges
- This means: **D** should depend on (*x,y*)
  – A space variant anisotropic diffusion
- This is referred to as *anisotropic diffusion* in the literature
- Introduced by Weickert

Anisotropic diffusion

- Information about edges and their orientation can be provided by an orientation tensor, e.g., the structure tensor **T** in terms of its eigenvalues λ_1, λ_2
- However:
  – We want α_k to be close to 0 when λ_k is large
  – We want α_k to be close to 1 when λ_k is close to 0

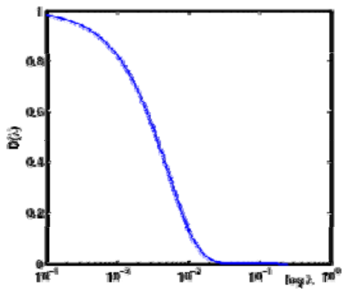
From **T** to **D**

- The diffusion tensor **D** is obtained from the orientation tensor **T** by modifying the eigenvalues and keeping the eigenvectors
  – similar to the control tensor **C**, e.g.

$$\alpha_k = \exp(-\lambda_k / k)$$

For example

k is a control parameter



Anisotropic diffusion: putting things together

1. At all points:
   1. compute a local orientation tensor **T**(**x**)
   2. compute **D**(**x**) from **T**(**x**)
2. Apply anisotropic diffusion onto the image by locally iterating

Left hand side: the change in *L* at (*x*,*y*) between *s* and *s*+∂*s*

$$\frac{\partial}{\partial s} L = \frac{1}{2} \mathrm{div}(\mathbf{D} \nabla L)$$

Right hand side: can be computed locally at each point (x,y)

This defines how scale space level *L*(*x,y;s*+∂*s*) is generated from *L*(*x,y;s*)

Implementation aspects

- The anisotropic diffusion iterations can be done with a constant diffusion tensor field **D**(**x**), computed once from the original image (faster)
- Alternatively: re-compute **D**(**x**) between every iteration (slower)

Regularization

- We assume **D** to have a slow variation with respect to **x** (cf. adaptive filtering)
- This means

$$\frac{\partial}{\partial s} L \approx \frac{1}{2}\, \mathrm{tr}\left[\mathbf{D}\,(\mathrm{div\ grad}\ L)\right] = \frac{1}{2}\, \mathrm{tr}\left[\mathbf{D}\,(\mathbf{H}\ L)\right]$$

The Hessian of *L* = second order derivatives of *L*

$$\mathbf{H}\, L = \begin{pmatrix} \frac{\partial^2}{\partial x^2} L & \frac{\partial^2}{\partial x \partial y} L \\ \frac{\partial^2}{\partial x \partial y} L & \frac{\partial^2}{\partial y^2} L \end{pmatrix}$$

Numerical implementation

- Several numerical schemes for implementing anisotropic diffusion exist
- Simplest one:
  - Replace all partial differentials with finite differences

$$\boxed{L(x,y;s+\Delta s) = L(x,y;s) + \Delta s\, \mathrm{tr}\left[\mathbf{D}(\mathbf{H}L)\right]}$$

The Hessian of *L* can be approximated by convolving *L* with:

$$H_{11} : \begin{pmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{pmatrix} \quad H_{12} : \begin{pmatrix} \frac{1}{4} & 0 & -\frac{1}{4} \\ 0 & 0 & 0 \\ -\frac{1}{4} & 0 & \frac{1}{4} \end{pmatrix} \quad H_{22} : \begin{pmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

Algorithm Outline

1. Set parameters

   e.g.: k, Δs, number of iterations, …

2. Iterate
   1. Compute orientation tensor **T**
   2. Modify eigenvalues $\Rightarrow$ **D**
   3. Computer Hessian **H** *L*
   4. Update *L* according to:

$$\boxed{L(x,y;s+\Delta s) = L(x,y;s) + \Delta s \,\mathrm{tr}\left[\mathbf{D}(\mathbf{H}L)\right]}$$

A note

- The image f is never convolved by the space variant anisotropic filter g
- Instead, the effect of g is generated incrementally based on the diffusion eq.

- In adaptive filtering: we never convolve f with $g_{\mathbf{x}}$, instead several fixed filters are applied onto f and their results are combined in a non-linear fashion

Comparison

Inhomogenous diffusion

Anisotropic diffusion