Applied Medical Image Processing Lecture Notes

1 Geometry Preserving Diffusion Filter:

In this approach we incorporate the geometry information into the smoothing process. Let's replace:

$$\frac{\partial I}{\partial t} = \nabla \cdot (K(\vec{x}, t) \cdot \nabla I)$$

with

$$\frac{\partial I}{\partial t} = \nabla \cdot (D \cdot \nabla I)$$
$$K \in \mathbb{R}$$
$$D \in \mathbb{R}^{2\times 2} D : \mathbb{R}^3 \rightarrow \mathbb{R}^{2\times 2}$$

D here is a symmetric, positive definite 2×2 matrix which represents an oriented, stretched ellipse which controls the local diffusion process. D can be independent of image " I " but it is usually derived from it.

$$\frac{\partial I}{\partial t} = \nabla \cdot [K \cdot \mathbb{I}_2 \cdot \nabla I] = \nabla . \left[\begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix} \nabla I \right]$$

Tschumperle & Derrick aimed at vector/multichannel images, but can applied on single channel images as well. For vector valued Images, assume $I : (I_1, \ldots, I_n)$ consists of n channels then

$$\frac{\partial I_K}{\partial t} = \text{trace}\,(A.H_k)$$

for each channel K. Here H_k is the Hessian matrix for channel K and $A : 2 \times 2$ array for $2D$ image which depends on Image I adopting smoothing to the local geometry.

Recall that if $f : \mathbb{R}^n \rightarrow R$, then Hessian Matrix of f is a $n \times n$ squared matrix of 2^{nd} order partial derivatives. By definition, we will have:

$$H_f : \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & & & \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{pmatrix} = \begin{pmatrix} H_{11} & H_{12} & \cdots H_{1n} \\ \vdots & & \\ H_{n1} & H_{n2} & \cdots H_{nn} \end{pmatrix}$$

Where $H(i,j) = H(j,i)$. The trace of this matrix, by definition is:

$$\begin{aligned} \text{trace}\,(H_f) : & \frac{\partial^2 f}{\partial x_1^2} + \frac{\partial^2 f}{\partial x_2^2} + \cdots \frac{\partial^2 f}{\partial x_n^2} \\ & : \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2} = \nabla^2 f \end{aligned}$$

For a 2D image $I \in \mathbb{R}^2 \to \mathbb{R}$, we could write:

$$H_I : \begin{pmatrix} \frac{\partial^2 I}{\partial x^2} & \frac{\partial^2 I}{\partial x_1 \partial x_2} \\ \frac{\partial^2 I}{\partial x_2 \partial x_1} & \frac{\partial^2 I}{\partial x_2^2} \end{pmatrix} = \begin{pmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{pmatrix}$$

Applying this approach to the Perona-Malik filter, we will have:

$$\begin{aligned} \frac{\partial I}{\partial t} &= \nabla \cdot (K \cdot \nabla I) \\ &= \text{trace}\,((K \cdot \mathbb{I}_2) \cdot H_I) = \text{trac}\,(A \cdot H_I) \end{aligned}$$

With A is defined as:

$$A = K \cdot \mathbb{I}_2 = K \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

A matrix does not represent geometry, instead in a multichannel case, we could

use:

$$A = f_1(\lambda_1, \lambda_2) \cdot \vec{e}_2 \cdot \vec{e}_2^{\top} +$$
$$f_2(\lambda_1, \lambda_2) \cdot (\vec{e}_1 \cdot e_1^{\top})$$
$$\text{where } \begin{cases} \lambda_1 2\lambda_2 \text{ eigenvalues} \\ \vec{e}_1, \vec{e}_2 \text{ eigenvectors (normalized)} \end{cases}$$
$$\text{of the following Matrix: } G = \sum_{k=1}^{k} (\nabla I_k) \cdot (\nabla I_k)^{\top}$$

2 Implementation steps:

Assume I is a multi-channel image (k channels) with size $M \times N$.

$I(\vec{u}) : M \times N \to \mathbb{R}^n \quad \vec{u}_{:}(u, v)$

1. perform $\nabla I_k(u) = \begin{pmatrix} \frac{\partial I_k(\vec{u})}{\partial x} \\ \frac{\partial I_k(\vec{u})}{\partial y} \end{pmatrix}$

$$= \begin{pmatrix} I_{k,x}(\vec{u}) \\ I_{k,y}(\vec{u}) \end{pmatrix} = \begin{pmatrix} (I_k * H_x^{\nabla})(\vec{u}) \\ (I_k * H_y^{\nabla})(\vec{u}) \end{pmatrix}$$

$$H_x^{\nabla} = \begin{bmatrix} -a & 0 & a \\ -b & 0 & b \\ -a & 0 & a \end{bmatrix}, H_y^{\nabla} = \begin{bmatrix} -a & -b & -a \\ 0 & 0 & 0 \\ a & b & a \end{bmatrix}$$
$$a = \frac{2 - \sqrt{2}}{4} \quad b = \frac{(\sqrt{2} - 1)}{2}$$

   Other x and y kernels can be used, but the proposed values here have good rotation invariance.

2. Smooth $I_{k,x} and I_{k,y}$ with an optional isotropic $2D$ Gaussian filter

$$\bar{\nabla} I_k = \begin{pmatrix} \bar{I}_{k,x} \\ \bar{I}_{k,y} \end{pmatrix} = \begin{pmatrix} I_{k,x} * H^{G,\sigma_d} \\ I_{k,y} * H^{G_1 \delta_d} \end{pmatrix}$$

3. Calculate Hessian Matrix

$$H_k : \begin{pmatrix} \frac{\partial^2 I_k}{\partial x^2} & \frac{\partial^2 I_k}{\partial x \partial y} \\ \frac{\partial^2 I_k}{\partial x \partial y} & \frac{\partial^2 I_k}{\partial y^2} \end{pmatrix} = \begin{pmatrix} I_k^* H_{xx}^{\nabla} & I_k * H_{xy}^{\nabla} \\ I_k * H_{xy}^{\nabla} & I_k * H_{yy}^{\nabla} \end{pmatrix}$$

$$H_{xx}^{\nabla} = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}, \quad H_{yy}^{\nabla} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

$$H_{xy}^{\nabla} : \begin{bmatrix} 1/4 & 0 & -1/4 \\ 0 & 0 & 0 \\ -1/4 & 0 & 1/4 \end{bmatrix}$$

4. Calculate the structure Matrix:

$$G = \begin{pmatrix} G_0 & G_1 \\ G_1 & G_2 \end{pmatrix} = \sum_{k=1}^{K} (\overline{\nabla I_k}) \cdot (\overline{\nabla I_k})^{\top} = \sum_{k=1}^{K} \begin{pmatrix} \bar{I}_{k,x}^2 & \bar{I}_{k,x} \bar{I}_{k,y} \\ \bar{I}_{k,x} \bar{I}_{k,y} & \bar{I}_{k,y}^2 \end{pmatrix}$$

5. Smooth structure Matrix G:

$$\bar{G} : \begin{pmatrix} \bar{G}_0 & \bar{G}_1 \\ \bar{G}_1 & \bar{G}_2 \end{pmatrix} = \begin{pmatrix} G_0 * H^{G,\sigma_g} & G_1 * H^{G,\sigma_g} \\ G_1 * H^{G,\sigma_g} & G_2 * H^{G,\sigma_g} \end{pmatrix}$$

6. For each image position (u, v) calculate eigenvalues λ_1, λ_2 form $\bar{G}$ such that $\lambda_1 \geq \lambda_2$. the corresponding eigenvectors are:

$$\vec{e}_1 = \begin{pmatrix} x_1 \\ y_1 \end{pmatrix} \quad \vec{e}_2 = \begin{pmatrix} x_2 \\ y_2 \end{pmatrix} \quad \|\vec{e}_1\| = \|\vec{e}_2\| = 1$$

Note that $\vec{e}_1 \perp \vec{e}_2 \rightarrow$ and $\vec{e}_1$ determines that largest variation in image gradient which is the edge direction. $\vec{e}_2$ represents the tangent to the edge. Since $\vec{e}_1 \cdot \vec{e}_2 = 0$, then we can write $\vec{e}_2 = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \overrightarrow{e_1}$

$$= \begin{pmatrix} -y_1 \\ x_1 \end{pmatrix}$$

7. Define matrix A as:

$$A = \begin{pmatrix} A_0 & A_1 \\ A_1 & A_2 \end{pmatrix} = \underbrace{f_1(\lambda_1, \lambda_2)}_{C_1} \cdot \left(e_2 \cdot e_2^\top\right) + \underbrace{f_2(\lambda_1, \lambda_2)}_{C_2} \cdot \left(e_1 \cdot e_1^\top\right)$$

$$= c_1 \cdot \begin{pmatrix} y_1^2 & -x_1 y_1 \\ -x_1 y_1 & x_1^2 \end{pmatrix} + c_2 \begin{pmatrix} x_1^2 & x_1 y_1 \\ x_1 y & y_1^2 \end{pmatrix}$$

$$= \begin{pmatrix} c_1 y_1^2 + c_2 x_1^2 & (c_2 - c_1) x_1 y_1 \\ (c_2 - c_1) x_1 y_1 & c_1 x_1^2 + c_2 y_1^2 \end{pmatrix}$$

$$c_1 = \frac{1}{(1 + \lambda_1 + \lambda_2)^{a_1}} \quad c_2 = \frac{1}{(1 + \lambda_1 + \lambda_2)^{a_2}}$$

a_1 &$a_2 > 0$ and control non-isotropy of the filter.

$$a_1 = 0.5 \quad a_2 = 0.9$$

$a_1 \longrightarrow$ smoothing along contours $a_2 \rightarrow$ perpendicular to the contour

8. Each image channel I_k is updated using:

$$I_k \longleftarrow I_k + \alpha \cdot \text{trace}\left(A \cdot H_k\right) = I_k + \alpha\beta_k$$
$$\beta_k = \left(A_0 I_{k,xx} + A_1 I_{k,xy} + A_1 I_{k,yx} + A_2 I_{kyy}\right)$$

Note that A is common across all channels indicating that we smooth along common image geometry.

$$\text{update } \alpha = \frac{dt}{\max \beta_x} = \frac{dt}{\max_{k,u} \left|\text{trace}\left(AH_k\right)\right|}$$

α is small as long as image gradients (Vector field velocities) are large.

Algorithm 1 summarizes the implementation steps.

3 Edges and Contours:

Edge is a position where local intensity changes distinctly along a particular orientation. The amount of change in intensity with respect to spatial distance is

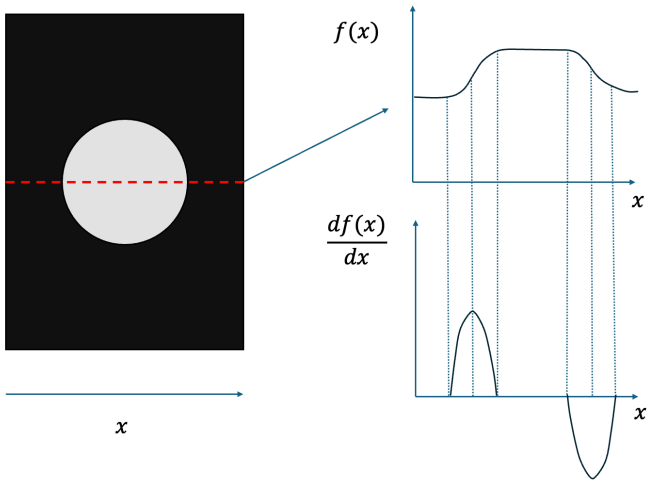

Figure 1: First order derivative of image cross section at the location marked with red doted line

known as first derivative. One way to identify edge is to use image gradient (see Fig. 1). Since derivative in continuous domain is the slope of the tangent line at a particular point, we can discretize it as follows:

$$\frac{df(u)}{du} \approx \frac{f(u+1) - f(u-1)}{2}$$

As described previously, for $2D$ image I, we will define gradient as:

$$\nabla I(u,v) = \begin{bmatrix} \frac{\partial I(u,v)}{\partial u} \\ \frac{\partial I(u,v)}{xv} \end{bmatrix}$$

And its magnitude as:

$$|\nabla I(u,v)| : \sqrt{\left(\frac{\partial I(u,v)}{\partial u}\right)^2 + \left(\frac{\partial I(u,v)}{\partial v}\right)^2}$$

Which is invariant under the rotation.

3.1 Derivative Filters:

Historically several different kernels were proposed to use discrete derivatives for the edge detection process. Examples are:

1. Prewitt Filter:

$$H_x^P = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad H_y^p : \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

note:

$$H_x^p = \underbrace{\begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}}_{\text{averaging operator}} \cdot \underbrace{\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}}_{\text{gradient operator}}$$

$$H_y^P = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}$$

2. Sobel Filter

$$H_x^S = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad H_y^S = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

$$H_x^s = \underbrace{\begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}}_{\text{smoothing with more weight to the center pixel}} \cdot \underbrace{\begin{bmatrix} -1 & 0 & 1 \end{bmatrix}}_{\text{gradient filter}},$$

$$\nabla I(u,v) = 1/6 \begin{bmatrix} I* & H_x^P(u,v) \\ I* & H_y^P(u,v) \end{bmatrix}$$

$$\nabla I(u,v) = 1/8 \begin{bmatrix} I * H_x^s(u,v) \\ I * H_y^s(u,v) \end{bmatrix}$$

$$D_x(u,v) = I * H_x$$

$$D_y(u,v) = I * H_y$$

other Sobel alternative with better performance:

$$H_x^{S'} = 1/32 \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} H_y^{s'} = 1/32 \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}$$

3. Kirsh Filter

$$H_0^k : \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad H_4^k = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$
$$H_1^k = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix} \quad H_5^k = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}$$
$$H_2^k = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \quad H_6^k = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$
$$H_3^k = \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix} \quad H_7^k = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}$$

Only 4 of 8 Filters is required due to the symmetry of kernels. For example: ${H_4}^k = -H_6^k$

$$I * H_4^k = I * -H_0^k = -\left(I * H_0^k\right)$$
$$D_0 = I * H_0^k \quad D_1 = I * H_1^k \quad D_2 = I * D_2^k \quad D_3 = H_3^k$$
$$D_4 = -D_0 \quad D_5 = -D_1 \quad D_6 = -D_2 \quad D_7 = -D_3$$
$$E^k(u, v) = \max\left(\left(D_0(u, v)\,|, D_1(u, v), \ldots |\, D_7(u, v)\right)\right)$$
$$\phi^K(u, v) \triangleq \frac{\pi}{4} \text{ as } j : \operatorname{argmax} D_2 \cdot (u, v)$$
$$0 \leqslant i \leqslant 7$$

3.2 Fundamental problem:

The gradient method generates edge which is as wide as underlying intensity transition (see Fig. 2). We could use 2^{nd} order derivative instead (Laplacian operator). Edges are identified by zero-crossing of the 2^{nd} derivative. However, second order derivatives could generally represent

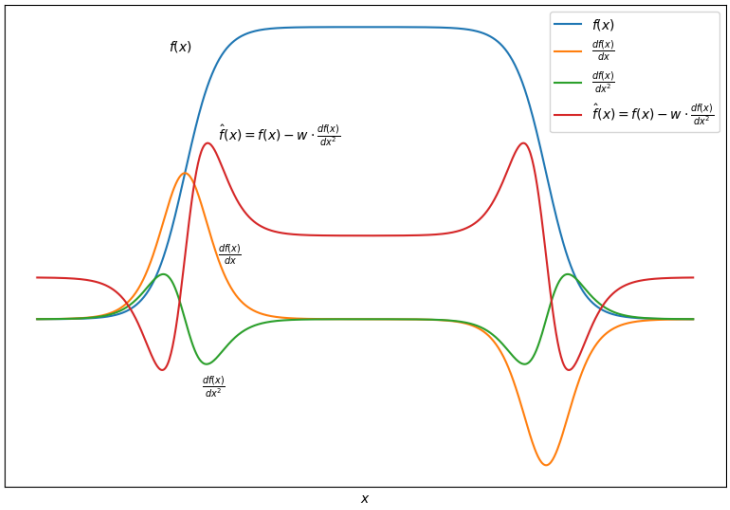

Figure 2: First and second order derivatives of image cross section at the location marked with red doted line in Fig. 1. The zero-crossing of second derivative can be used to identify the location of an edge in the image. Additionally, second derivatives can be used to generate overshooting across both sides of the edge (red curve).

image noise, therefore a pre-smoothing filter can be included (Ex. Laplacian of Gaussian)

Few examples of Laplacian operator are listed bellow:

$$H^L : \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

$$H^L_8 : \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$H^L_{12} : \begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

By subtracting a weighted second derivative image from the original image ($\hat{I} = I - \omega \left(I * H^L\right)$) we could enhance pixels across the edge boundary (see Fig. 2)

3.3 Unsharp Masking (USM)

Unsharp masking is a technique used in image processing to enhance the sharpness and clarity of an image. The process involves subtracting a smoothed version of image from itself. The steps are:

1. generate a mask:
$$M = I - (I * H) = I - \tilde{I}$$
Where H is a normalized smoothing kernel such as Gaussian.
2. Generate the new image as a weighted difference between the original and smoothed image.
$$\hat{I} = I + \alpha \cdot M = I + \alpha(I - \tilde{I}) = (1 + \alpha)I - \alpha\tilde{I}$$

Advantage: $\downarrow$ ed sensitivity to noise.

$$\hat{I}(u, v) = \begin{cases} I(u, v) + \alpha M(u, v) & \text{if } |\nabla I(u, v)| \geqslant \text{th.} \\ I(u, v) & \text{otherwise} \end{cases}$$

Note that Laplacian filter is a specific case of USM as shown bellow:

$$H^L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} - 5 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

$$= 5(1/5[\underbrace{\begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix}}_{\tilde{H}} - \underbrace{\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}}_{\delta}) = 5(\tilde{H} - \delta)$$

$$\tilde{I}_L = I - \omega \cdot \left(H^L * I\right) = I - \omega \cdot \left(5(\tilde{H} - \delta) * I\right)$$

$$= I - 5\omega \cdot (\tilde{H} * I - I) = I + 5\omega(I - \tilde{H} * I)$$

$$= I + 5\omega \cdot \tilde{M}$$

$$M = \tilde{M} = (I - \tilde{H} * I)$$

$$\tilde{H} = 1/5 \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & \\ 0 & 1 & 0 \end{bmatrix} \quad \alpha = 5\omega.$$

Algorithm 1 TschumperleDericheFilter(I, T, d_t, σ_g, σ_s, a_1, a_2)

Input: $\mathbf{I} = (I_1, ..., I_K)$, image of size $M \times N$ with K channels;
T, number of iterations; d_t, time increment;
σ_g, width of Gaussian for smoothing the gradient; σ_s, width of Gaussian for smoothing the structure matrix; a_1, a_2, diffusion parameters for directions of min./ max. variation, respectively.
Returns the modified image I.

Create maps:
$\mathrm{D} : K \times M \times N \to \mathbb{R}^2$
$\mathrm{H} : K \times M \times N \to \mathbb{R}^{2\times 2}$
$\mathrm{G} : M \times N \to \mathbb{R}^{2\times 2}$
$\mathrm{A} : M \times N \to \mathbb{R}^{2\times 2}$
$\mathrm{B} : K \times M \times N \to \mathbb{R}$
for $t \leftarrow 1, \ldots, T$ **do**
  for $k \leftarrow 1, \ldots, K$ and **all** coordinates $(u, v) \in M \times N$ **do**
    $\mathrm{D}(k, u, v) \leftarrow \begin{pmatrix} (I_k * H_x^{\nabla})(u, v) \\ (I_k * H_y^{\nabla})(u, v) \end{pmatrix}$
    $\mathrm{H}(k, u, v) \leftarrow \begin{pmatrix} (I_k * H_{xx}^{\nabla})(u, v) & I_k * H_{xy}^{\nabla})(u, v) \\ (I_k * H_{xy}^{\nabla})(u, v) & I_k * H_{yy}^{\nabla})(u, v) \end{pmatrix}$
  end for
  $\mathrm{D} \leftarrow \mathrm{D} * H_G^{\sigma_d}$
  for all coordinates $(u, v) \in M \times N$ **do**
    $\mathrm{G}(k, u, v) \leftarrow \sum_{k=1}^{K} \begin{pmatrix} (\mathrm{D}_x(k, u, v))^2 & \mathrm{D}_x(k, u, v) \cdot \mathrm{D}_y(k, u, v) \\ \mathrm{D}_x(k, u, v) \cdot \mathrm{D}_y(k, u, v) & (\mathrm{D}_y(k, u, v))^2 \end{pmatrix}$
  end for
  $\mathrm{G} \leftarrow \mathrm{G} * H_G^{\sigma_g}$
  for all coordinates $(u, v) \in M \times N$ **do**
    $(\lambda_1, \lambda_2, e_1, e_2) \leftarrow EigenValues(\mathrm{G}(u, v))$
    $\hat{e}_1 \leftarrow \begin{pmatrix} \hat{x}_1 \\ \hat{y}_1 \end{pmatrix} = \frac{e_1}{\|e_1\|}$
    $c_1 \leftarrow \frac{1}{(1+\lambda_1+\lambda_2)^{a_1}}$, $c_2 \leftarrow \frac{1}{(1+\lambda_1+\lambda_2)^{a_2}}$
    $\mathrm{A}(u, v) \leftarrow \sum_{k=1}^{K} \begin{pmatrix} c_1 \cdot \hat{y}_1^2 + c_2 \cdot \hat{x}_1^2 & (c_2 - c_1) \cdot \hat{x}_1 \cdot \hat{y}_1 \\ (c_2 - c_1) \cdot \hat{x}_1 \cdot \hat{y}_1 & c_1 \cdot \hat{x}_1^2 + c_2 \cdot \hat{y}_1^2 \end{pmatrix}$
  end for
  $\beta_{max} \leftarrow -\infty$
  for $k \leftarrow 1, \ldots, K$ and **all** coordinates $(u, v) \in M \times N$ **do**
    $\mathrm{B}(k, u, v) \leftarrow trace(\mathrm{A}(u, v) \cdot \mathrm{H}(k, u, v))$
    $\beta_{max} \leftarrow max(\beta_{max}, |\mathrm{B}(k, u, v)|)$
  end for
  $\alpha \leftarrow d_t / \beta_{max}$
  for $k \leftarrow 1, \ldots, K$ and **all** coordinates $(u, v) \in M \times N$ **do**
    $I_k(u, v) \leftarrow I_k(u, v) + \alpha \cdot \mathrm{B}(k, u, v)$
  end for
end for
return I

Bibliography

[1] Wilhelm Burger, Mark J. Burge , Principles of Digital Image Processing: Advanced Methods, Chapter 5 - Edge-Preserving Smoothing Filters, 2013

[2] D. Tschumperlé, Fast Anisotropic Smoothing of Multi-Valued Images using Curvature-Preserving PDE's. International Journal of Computer Vision, Vol. 68, No. 1, pp. 65–82, 2006, ISSN : 0920-5691.

[3] Handbook of Medical Image Processing and Analysis Book, Chapter 1 - Fundamental Enhancement Techniques and Chapter 5, Section 5-1 to 5-5., Second Edition 2009