REAL-TIME ANISOTROPIC FILTERING BASED ON LINE INTEGRAL CONVOLUTION APPROXIMATIONS

Benoît Vandame[1]

[1]Canon Research Center France,
Rue de la touche Lambert, F-35510 Cesson Sévigné, FRANCE, benoit.vandame@crf.canon.fr

ABSTRACT

Regularization of noisy or corrupted data has been successfully addressed using Partial Differential Equations (PDE's). Many formalisms have been proposed, most of them define a local smoothing along one or several directions in order to preserve contours and sharp details. The drawback of these methods is their very high computation time.

Our interest is to achieve comparable results in a real-time context. To do so, the formalism must be drastically simplified. Starting with a quick review of PDE's, and especially about the Line Integral Convolution, we point out aspects that could be simplified. Then, we defined an anisotropic local filtering based on a one dimensional kernel, such as its shape has a small *crossing intensity* versus the local gradients of the image to restore. This notion allows to define, for a given pixel, an adequate kernel shape which will not affect contours and corners during filtering stage.

Our algorithm offers an efficient anisotropic smoothing being one to several magnitudes faster than previous methods based on PDE's. Our results show that we can process video sequence in real-time. Besides the efficient smoothing; this diffusion method preserves contours and corners even when the anisotropic filtering is applied many times.

1. INTRODUCTION

Recently, regularization of noisy or corrupted image data is an important field of image processing. It is useful to restore a signal as a pre-processing step before further analysis. Regularization is a critical step for many complex processes, and must be achieved in real-time for embedded solutions. In this context, the dedicated computation time is often very small and is not compatible with sophisticated approaches: a huge constraint is imposed to regularization algorithms. This issue must be addressed either by using faster computers, or by modifying the algorithm. It is often a combination of both approaches that conduct to fast and efficient solutions.

Many formalisms have been proposed in the literature. Since the pioneering work of Perona-Malik[1] in the early 90's, anisotropic diffusion based on Partial Differential Equations (PDE's) has become popular: these equations define a nonlinear way to smooth images and preserve discontinuities. Many variants of diffusion PDE's have been proposed [1, 2, 3]. The wide variety of PDE regularization share the same basic property: they define local smoothing along one or several directions for each pixel of an image. The principal smoothing directions are defined parallel to image contours. It results an anisotropic regularization preserving the edges. The drawback of those methods is their very high computation time.

An interesting approach to anisotropic smoothing have been introduced by Cabral which defines the Line Integral Convolution[4, 5] (LIC). The anisotropic smoothing is defined along a streamline which is constraint by a vector field. This algorithm defines a one dimension convolution mask. This formalism as been used by Tschumperlé to obtain faster PDE filtering[6].

In parallel to the PDE's other methods based on anisotropic smoothing have been developed. It is worth mentioning the work around the bilateral filter[7]. Basically, a convolution kernel is designed for each pixel depending on the value and distance of the surrounding pixels. Those approaches offer shorter computation time than PDE's, but nevertheless a real-time computation is still not available. The relationship between PDE's and bilateral filtering has been detailed in [8].

Anisotropic filtering is achieved in two steps: First, local features around a pixel are analyzed; Then, a local filtering is applied following the preliminary analysis. Local features analysis is commonly based on gradients or similarity computation. Local filtering is made using elongated Gaussian function, weighted Gaussian, or other means. The problem of adapting the kernel shape is extensively studied, the following articles [3, 9, 10, 11, 12, 13] illustrate different approaches comparable to the proposed method.

In a real-time context, both steps must be optimized. Therefore, we have defined an anisotropic local filtering based on a one dimensional warped kernel, such as its shape has a small *crossing intensity* versus the local gradients of the image. This notion is defined to compute how much a streamline (the shape of the filtering kernel) is crossing the local gradient field. This notion makes the selection of a kernel shape, versus a family of shapes, very efficient. We have built a denoising algorithm preserving local features like contours, corners, which is a magnitude faster than existing strategies. Ours results shows that real-time restoration is feasible for still and video images with a good quality.

In this article we first make a quick review of regularization methods based on PDE's, LIC and Bilateral Filter (section 2) pointing advantages and drawbacks. We then propose an approximation to the anisotropic smoothing based on LIC which is compatible with real-time processing (section 3). Results on simulated and real images are shown in section 4. Finally, section 5 gives the conclusion.

2. BACKGROUND

Typical image restoration based on PDE's are organized in three steps: the gradients and/or Laplacians of an image are computed for different orientations and organized into a tensor; Then, the tensors are filtered to decrease the level of noise and increase their internal coherence; And finally, for each pixel a convolution kernel is built and then applied. The coefficients of the kernel are deduced from a Gaussian function which is characterized by the tensor associated to that pixel. For an homogeneous area, the kernel is made of a large symmetric Gaussian function. Around a contour, the Gaussian function is very elongated parallel to that contour. For other areas such as textures, the kernel is almost symmetric with a small radius. The diffusion achieved by theses kernels keeps contours intact. The computation time associated is very high because for each pixel one needs to compute all coefficient of the two dimensional kernel.

An interesting approach has been proposed by Cabral[4]. The filtering is made by a one dimensional kernel which passes through a pixel to filter, and then follows the vector field. The vector field is defined perpendicular to the gradient of the image, such as the one dimensional kernel is designed parallel to contours. This approach has the great advantage to decrease the size of the kernel compared to two dimensional kernels. The drawback of this method is the time used to compute the shape of the kernel which must follow precisely the vector field. A solution has been proposed by [5] to speed-up the shape computation.

The Bilateral Filter is another way to build anisotropic filtering. A convolution kernel is built for each pixel to filter. Each coefficient of the kernel is computed depending on: the spatial distance; And, the values of the central pixel and the ones covered by the coefficient. For homogeneous areas, convolution kernels are close to a Gaussian function. For contours or corner areas, the convolution kernel looks like a truncated Gaussian function which keeps intact the sharp details. The computation time needed to build the kernel is high, bilateral filters are not compatible with real-time processing. Nevertheless, a simpler version has been designed as presented in section 4.

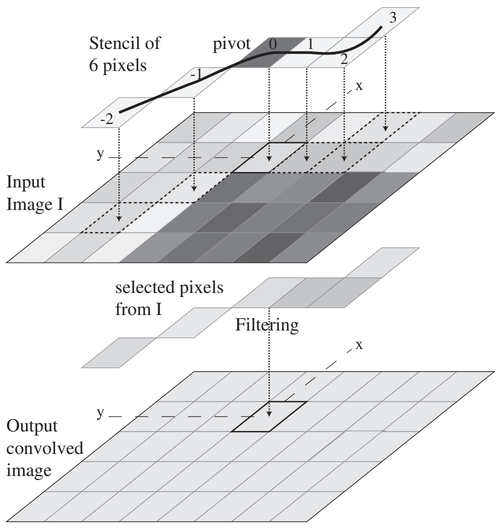

Figure 1. Example of a stencil made of 6 pixels, and its application to convolve one pixel of an input image.

3. LINE INTEGRAL CONVOLUTION APPROXIMATION

3.1. Motivations and Terminology

The LIC defines, for each pixel to process, a streamline which corresponds to a 1 dimensional convolution kernel made of L pixels for instance. This anisotropic smoothing is made using fewer pixels than a classical 2 dimensional convolution kernel of $L \cdot L$ pixels. This characteristic makes LIC appealing for real-time computation. Obviously this 1D convolution kernel defines a softer smoothing than 2D convolution mask.

The notions of: Shape of the convolution kernel (the streamline); And, convolution coefficients, are defined independently to the streamline. One could use whatever convolution coefficients, or non-linear filtering (such as median filtering). As pointed above: the complexity resides in computing, for each pixel of the input image, a streamline according to a given vector field. Our motivation is to focus on this step which is the most time consuming. In our case the vector field is the gradient of the image to filter.

The path followed by a streamline must be computed very precisely especially around contours. If not, sharp edges will be smoothed by the filtering. On the other hand, the path followed by a streamline in an homogeneous area does not need to respect precisely the vector field. Indeed, the smoothing for those areas should be isotropic anyway. In other words, the computation of a streamline according to the vector field for a given pixel can be simplified by finding one streamline among others which will not cross any contours. With this new constraint, several streamlines are valid for a given pixel.

Let us use the word *stencil* to characterize a shape (streamline) including the pixels used for a filtering. A stencil is a group of L pixels labeled p_i with $i \in \mathbb{Z}$, such as a pixel p_i is 8-connected with the previous p_{i-1} and next p_{i+1} ones. A stencil defines the selected pixels for a filtering. The pixel of index zero of the shape is the pivot, it defines how a stencil is applied to a given pixel of an input image. The nature of the filtering, the way pixels are filtered linearly or not, is not defined by the stencil. Figure 1 illustrates a stencil made of 6 pixels with $i \in [-2, 3]$, its pivot ($i = 0$) is aligned to the the pixel $[x, y]$ of the input image I. The 6 pixels selected by the stencil are then used for the filtering, the result is injected back on the output image at location $[x, y]$. One stencil is defined for each pixel of the input image.

The constraint: finding one streamline among others which will not cross any contours, is reformulated using the notion of stencil. The filtering of a given pixel is achieved by one stencil, among others, such that it does not cross any contours. In Figure 1, the input pixel $[x, y]$ is properly filtered by any stencil which does not encompass one of the darker pixels. The one illustrated is therefore prone to de-noise properly that pixel. One needs to define a notion of crossing intensity in order to quantify how much the pixels of a stencil are crossing the contours of an image.

3.2. Crossing intensity

Using the constraint mentioned above, we characterize a stencil S which is computed for a pixel X according to a vector field W defined for each pixel of the input image I. W is the gradient of I: $W = \nabla I$. Let $S^X_{(a)}$ be the stencil starting from X and indexed by $a \in \mathbb{R}$:

$$\left\{ \begin{array}{rcl} S^X_{(0)} & = & X \\ C & = & \displaystyle\int_{-\infty}^{+\infty} g\left(\vec{S}_{\perp(a)}, \vec{W}(S^X_{(a)})\right) \cdot w(a) \cdot da < e \end{array} \right. \tag{1}$$

When $a \to +\infty$ the stencil $S^X_{(a)}$ is traced forward, and backward when $a \to -\infty$. $\vec{S}_{\perp(a)}$ is the vector orthogonal to the stencil at the point (a). $\vec{W}(S^X_{(a)})$ is the vector of W at the point $S^X_{(a)}$. The function g is used to compute an orientation error between vector $\vec{S}_{\perp(a)}$ and the vector at point $\vec{W}(S^X_{(a)})$. The function w is used to weight the function g depending on a. C is integral of $g \cdot w$ through a, and must be inferior to the criteria named e. C is named the crossing intensity, it quantifies how much a stencil S is crossing the vector field W.

If e is null, then the stencil S is equal to the integral curve of W as defined by the LIC. The function g may have several forms. The simplest one g_o compares the orientation of two vectors u and v:

$$g_o(u, v) = \left| \arccos\left(\frac{u \cdot v}{\|u\| \, \|v\|}\right) \right| \bmod \frac{\pi}{2} \tag{2}$$

The function g can be defined to be more sensitive to orientation error for points where the vector field w is strong:

$$g_N(\vec{u}, \vec{v}) = g_o(\vec{u}, \vec{v}) \cdot \|\vec{v}\|^N \tag{3}$$

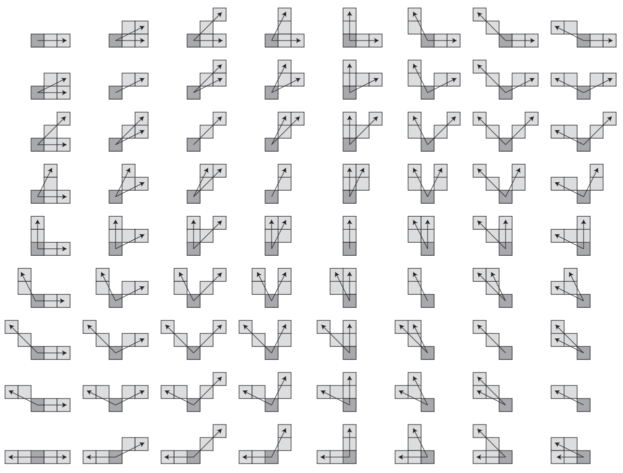
Figure 2. Subset of the possible stencils made of 5 8-connected pixels according to the “Two branches model”. The two arrows of each stencil represent the orientations θ_1 and θ_2 of the model. The darker pixel is the pivot of the stencil. Stencils with $\theta_i \in [0, \pi[$ are represented, others are deduced using vertical and horizontal mirror.

With $N = 1, 2 \ldots$ The orientation error is almost null for homogeneous areas whatever is the stencil S. On another hand, for strong gradient of W corresponding to contours, a small orientation error on S will be maximized. Experiences show that the choice of $N = 2$ leads to the best definition of crossing intensity.

It is also possible to weight the effect of g with the function of w. If w is chosen constant, the instant crossing intensity at point a is independent of a. Function w might be chosen Gaussian, it means that an instant crossing intensity is lowered for points of S far from the starting pixel X (the pivot of S).

As for LIC, the vector field W is defined as the gradient of image I: $W = \nabla I$. Following Equation 1 for a given pixel X, several stencils respect the given criteria. The goal is to find one which can be computed as quickly as possible. In order to simplify Equation 1, we define a model including a finite number of stencils.

3.3. The two branches model

Our “two branches model” defines a stencil made of 2 straight branches of same size centered in pixel X as defined bellow:

$$\left\{ \begin{array}{rcl} S^X_{(0)} & = & X \\ \dfrac{\partial S^X_{(a>0)}}{\partial a} & = & \vec{\theta}_1 \\ \dfrac{\partial S^X_{(a<0)}}{\partial a} & = & \vec{\theta}_2 \end{array} \right. \tag{4}$$

$\vec{\theta}_1$ is an unitary vector of orientation θ_1. The orientation of $\vec{S}_{\perp(a)}$ becomes either: $\theta_1 + \frac{\pi}{2}$ or $\theta_2 + \frac{\pi}{2}$ depending on the sign of a. The computation of g is made simpler with this model. From Equation 1 we can write $C = C_1 + C_2$ such as C_1 corresponds to the crossing intensity for $a < 0$ and C_2 to crossing intensity for $a > 0$.

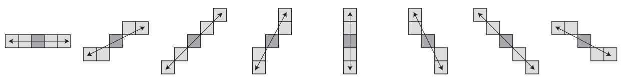
Figure 3. Stencil aligned used for the first guess of the iterative process. The three last stencils being symmetrical to the first ones.

In discrete space, the model is defined by the two orientations of the two branches, and the length in pixel of the stencils. If L is the length of the stencil, the total number of orientations of a branches is equal to $4(L-1)$, the total number of stencils is equal to $16(L-1)^2$. Figure 2 represents 64 of the 256 stencils for length $L = 5$. We notice that this family of stencils is able to track precisely straight contours and corners.

This model gives a finite number of stencils. For each pixel to filter, one needs to select one stencil, among others, according to the Equation 1. The selection is done iteratively as described in next sub-section.

3.4. Iterative selection of stencils

For a given pixel to filter, one stencil from the model must be chosen. The crossing intensity C_1 and C_2 of the two branches are independent and could be computed independently for all possible orientations. This strategy is heavy and inadequate for real-time processing. Instead an iterative strategy is defined: a first guess of the stencil is chosen, and then improved according to the crossing intensity of the two branches. The main idea is to find a stencil that is: oriented perpendicular to the strong gradients being crossed; and, with a minimum crossing intensity.

As a first guess, one selects the aligned stencil perpendicular to the vector field at the given pixel X. In other words: $\theta_1 = (\theta_{\vec{W}(X)} + \frac{\pi}{2}) \bmod 2\pi$ and $\theta_2 = (\theta_1 + \pi) \bmod 2\pi$. $\theta_{\vec{W}(X)}$ being the orientation of the vector field at the pixel X.

It is worth mentioning that this approach is defined by [14], and detailed in [4]. The number of aligned stencils is equal to $2(L-1)$. Figure 3 illustrates the 8 aligned stencils for $L = 5$.

This first guess is adequate for straight contours, but has a large crossing intensity around corners or curved contours. While error $C_1$and C_2 are computed, one computes for each branch the vector V_1 and V_2 such as:

$$\begin{cases} \vec{V_1} &= \sum_{a=-L/2}^{a=0} \vec{W}(S^x_{(a)}) \cdot \left\| \vec{W}(S^x_{(a)}) \right\|^N \\ \vec{V_2} &= \sum_{a=0}^{a=L/2} \vec{W}(S^x_{(a)}) \cdot \left\| \vec{W}(S^x_{(a)}) \right\|^N \end{cases} \quad (5)$$

In other words, V_i is the weighted sum of the vectors, from the vector field W, being crossed by the branch i of the stencil S. N is used to improve the impact of strong vectors, it is chosen equal to 1 or 2. The stencil is re-oriented per branch if its corresponding crossing intensity is higher than e. The stencil is reoriented as follow: if $C_1 > e$ then $\theta_1 = (\theta_{\vec{V_1}} + \frac{\pi}{2}) \bmod 2\pi$, and if $C_2 > e$ then $\theta_2 = (\theta_{\vec{V_2}} - \frac{\pi}{2}) \bmod 2\pi$.

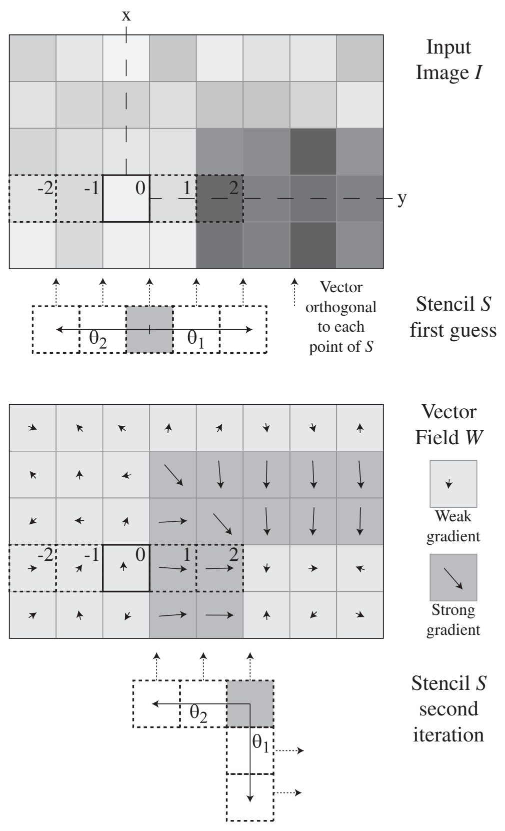

Figure 4. Illustration of the re-orientation of a straight stencil associated to one pixel.

In fact, only the orientations of vectors V_i are considered, their modules being useless. A vector V_i is computed in order to derive a new orientation prone to decrease the crossing intensity of branch i. This strategy allows to pick a good stencil candidate without computing the crossing intensity of all possible stencils.

This iteration is applied a couple of times. For each iteration, one keeps the $\theta_{1min} = \theta_1$ and $\theta_{2min} = \theta_2$ that provides the smaller crossing intensity respectively C_{1min} and C_{2min}. After the couple of iterations, the stencil chosen for the filtering is the one oriented by θ_{1min} and θ_{2min}.

Figure 4 illustrates the computation of the stencil S for the pixel X at position $[x, y]$ of the image I. The vector field W is the gradient of I. It is illustrated with arrows. One notices the strong gradients oriented perpendicular to the contours of the dark pixels, and the weak gradients influenced by the noise of I. The first guess of S is chosen perpendicular to the vector field at position X (as illustrated by θ_1 and θ_2). The crossing intensity C_2 is small as the second branch of S is just crossing weak gradients. By opposition, the crossing intensity of C_1 is important as S is crossing the strong gradients and $\vec{S}_{\perp(1)}$ and $\vec{S}_{\perp(2)}$ are almost orthogonal to respectively $\vec{W}(S^X_{(1)})$ and $\vec{W}(S^X_{(2)})$. θ_2 is then redefined perpendicular to $\vec{W}(S^X_{(1)})$ and $\vec{W}(S^X_{(2)})$. The second estimate of S is now optimal: the crossing intensities are small on the two branches. A good number of iterations is 2 or 3.

This iterative selection of a S does not guaranty to respect Equation 1 ($C < e$) for two reasons. First: none of the stencils from the two branches model might respect the constraint, and second: only few stencils are tested. Nevertheless, the two branches model and the iterative approach lead to a robust choice and an efficient filtering preserving contours and corners (as illustrated in section 4). In practice the parameter e might be omitted and one looks for the stencil having minimum crossing intensity between the one tested during the couple of iterations.

3.5. Anisotropic filtering

When a stencil has been selected for each pixel, one can apply the anisotropic smoothing to the image I. One chooses a filtering made on the L pixels for each stencil. It can be a linear smoothing using convolution coefficients, or non linear such as median filtering. It is also possible to apply a bilateral filtering. Section 4 illustrates different type of filters.

It worth mentioning that the vector field W can be filtered by the same principle. One selects a filter able to filter the L vectors selected by a stencil. Filtering W is useful to enhance its coherence. This is a typical step which is mentioned by [15, 16]. The advantage of this approach is that the vector field is smoothed in an anisotropic way ([15, 16] defined a Gaussian smoothing).

3.6. The algorithm

3.6.1. Step by Step

1. Computation of the W the vector field of the input image I. Let F_∇ be the filter chosen to compute the gradient.
2. Stencil computation.
   (a) Section of S_X perpendicular to $\vec{W}(X)$.
   (b) For each S_X, loop N_r times:
      i. Computation of crossing intensity C_1 and C_2.
      ii. Computation of the vector V_1 and V_2.
      iii. Check if C_1 and C_2 are minimum compared to the previous iterations: keep the corresponding $\theta_{1min} = \theta_1$ and $\theta_{2min} = \theta_2$.
      iv. Re-orientation of branch i if $C_i > e$ such as $\vec{\theta}_i \perp \vec{V}_i$.
   (c) Define S_X oriented by θ_{1min} and θ_{2min}.
3. Filtering
   (a) If needed the vector field W is filtered by F_W to be defined. Stencils must be recomputed: loop to step 2.
   (b) Compute the output image O: the filtered version of I according to F_I to be defined. The process can be iterated N_f times by looping on step 1 with $I = O$.

The complete algorithm is made of 3 major steps: The first one is a classical gradient filtering, and is very fast to compute; The second is the computation of stencils, the most demanding step; The third is the anisotropic filtering applied to the input image or the vector field. This last step is fast because few pixels are used for the computation despite the various shapes of the stencils.

3.6.2. Parameter choice

Several parameters have to be defined. The following list gives the typical values for those ones:

F_∇ One can use the Sobel or Prewitt filters. Shorter convolution kernels can be used such as: $[\ -1\ \ 0\ \ 1\ ]$ and $[\ -1\ \ 0\ \ 1\ ]^T$ for respectively horizontal and vertical convolution. Theses masks are faster to apply, but filtering with Prewitt of Sobel filters provides W with greater signal to noise.
In case of larger noise, a more extended version of this filter can be chosen. For instance $F_{\nabla 25} = [-1, -2, 0, 1, 2] \cdot [1, 2, 4, 2, 1]^T$ includes a smoothing. This filter is made of 5 by 5 pixels.

L The length of the stencil, typical values are 5, 7, 9, 11, 15.

g This function is used to compute the instant crossing intensity of two vectors (see Equation 1). Experiences show the function g_N with $N = 2$ (see Equation 3) gives the best results. By opposition, the function g_0 gives to much weight to small gradients.

w This function is used to weight the effect of g depending on the distance of the pixel of a stencil to the pivot. For faster computation, this function is not used. In other word this we choose $w = 1$.

e This parameter can be omitted as described in subsection 3.4.

N_r The number of iteration used to re-orientation the stencil is typically set to 3. Larger values do not lead to better re-orientation.

N_f The number of times the filtering is applied to the input image. This parameter correspond to the "temperature" of the heat equation. In practice only one iteration is used. A larger number of iteration may be desirable for noisy images.

F_l It corresponds to a filter having as input the L pixels of a stencil. The resulting value of the filter is the output pixel. F_l might have many different forms. One distinguishes: First, the linear filtering where F_l is defined with L coefficients as for a convolution; Second non linear filtering where the output pixel is equal to the median of the L input pixels; And last, bilateral filtering where each coefficient of the mask c_a with $a \in [-L/2, L/2]$ is computed depending on the value of the pixel p_{pivot} covered by the pivot minus the value of the pixel p_a covered by a. It corresponds to the application of the bilateral filter, except

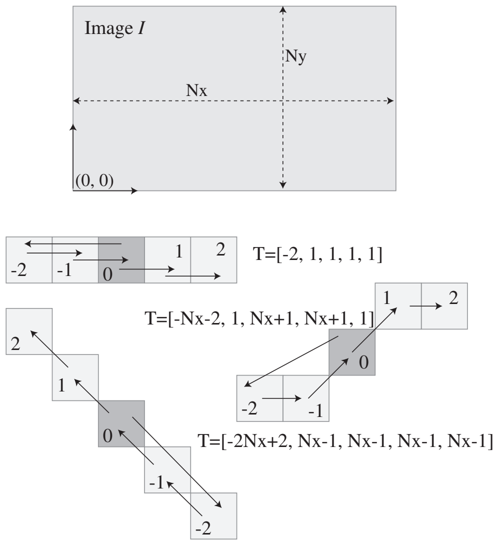

Image I
Ny
Nx
(0, 0)
-2
-1
0
1
2
T=[-2, 1, 1, 1, 1]
T=[-Nx-2, 1, Nx+1, Nx+1, 1]
T=[-2Nx+2, Nx-1, Nx-1, Nx-1, Nx-1]

Figure 5. Illustration of different tables of indexes, each one describing a stencil. The indexes in the table of stencil is function of the width $N_x$of the input image.

that the support for the convolution kernel is defined by the stencil.
This last version is certainly the one offering the best restoration. In order to have an efficient implementation, the coefficient c_a of the filter is equal to 1 if: $|p_{pivot} - p_a| < R$ and 0 else. R being the resolution which is chosen depending on the noise amplitude. This Equation is simplified taking advantage of the integer division defined by the CPU: $(p_{pivot} - p_a)/R = 0$. If the parameter R is a power of 2, the division can be optimize further. This filter $F_{l,R}$ is a sharp approximation of the bilateral filter.

F_W It is the filter applied to the vector field W. It is used to enhance the coherence of W in case of strong noise. The filter has the L vectors covered by a stencil as input. The result is the filtered vector. The implemented version defines a convolution where each coefficient c_a with $a \in [-L/2, L/2]$ is equal to $c_a = \left\|\vec{W}(S^X_{(a)})\right\|^2$.

3.6.3. Implementation details

The implementation is an important issue for real-time algorithm. The computation of anisotropic filters is time consuming as a filtering kernel must be computed for each pixel. The present algorithm has been designed to have a finite number of filtering kernels (following the two branches model describe in sub-section 3.3). To prevent computing the shape of a stencil each time it is used, all stencils are first pre-computed.

The relative position from one point of a stencil to the next can be seen as one dimensional shift in the input image. One assumes that an image I of size $[N_x, N_y]$ is also seen as a vector V made of $N_x \cdot N_y$ contiguous pixels. The position of a pixel X is characterized by two coordinates $[x, y]$, it can also be referred versus V with one coordinate $[x + y \cdot N_x]$. The translation from a pixel $X_1(x_1, y_1)$ to a pixel $X_2(x_2, y_2)$ can be interpreted as one shift of $x_2 - x_1 + N_x(y_2 - y_1)$ pixels in the referential of V.

Using this property, a stencil of L points can be described by a table T of L values indicating the shift from one point to the next. Figure 5 illustrates several tables for different stencils. The first value of a table gives the shift from the pivot p_0 to point p_{-2}, the second value gives the shift between point p_{-2} to point p_{-1} and so on. One notices that only the horizontal stencil does not depend on the width of I. The $16(N-1)^2$ stencils are precomputed once, knowing the width of the input image, and used for each stencil re-orientation.

Pre-computing all the stencils allows to factorize the computation on-demand during stencil re-orientation (typically 3 times) and image filtering (at least 1 time). Also from a CPU point of view, computing the stencils on-demand monopolizes CPU resources (such as registers) which can not be used for other purposes (loops, local variables of the filter...). Therefore a pre-computation is mandatory for a real-time algorithm.

The pre-computation leads to $16(N-1)^2$ tables T corresponding to all the stencils. The tables are arranged in a two dimensional array of size $4(N-1)$ such as the two coordinates of a table correspond to the orientation θ_1 and θ_2. It results in a direct correspondence between: the index of a table/stencil in the array, and the two parameters characterizing that stencil. This relation is used to optimize the code.

Parameters (simulated case)	Parameters (real case)
$F_\nabla = F_{\nabla 25}$	$F_\nabla = F_{\nabla 25}$
$L = 9$	$L = 17$
$w = Cste$	$w = Cste$
$g = g_{N=2}$	$g = g_{N=2}$
$N_r = 3$	$N_r = 3$
$F_I = [1, 2, 4, 8, 16, 8, 4, 2, 1]$	$F_I = F_{l,R=64}$
$N_f = 1, 5, 10, 100$	$N_f = 1$

Table 1. Parameters set used for the restoration of the synthetic images (left column) and real case (right column).

4. APPLICATIONS

4.1. Simulation images

In order to test the algorithm, synthetic images with different levels of Gaussian noise have been created. The noise-free version consists in a disk of amplitude 1, with a rectangle of amplitude 0.5. Several version with additive Gaussian noise have been created such as the signal to noise of the disk is equal to $\frac{S}{N} = 16, 8, 4, 2, 1$. The parameters selected to restore the synthetic images are given in Table 1 (left column).

Figure 6 illustrates the restoration of the different synthetic images depending on the number of filtering iterations (N_f). The last row illustrates the Restoration achieved for the noisiest synthetic image ($\frac{S}{N} = 1$), with

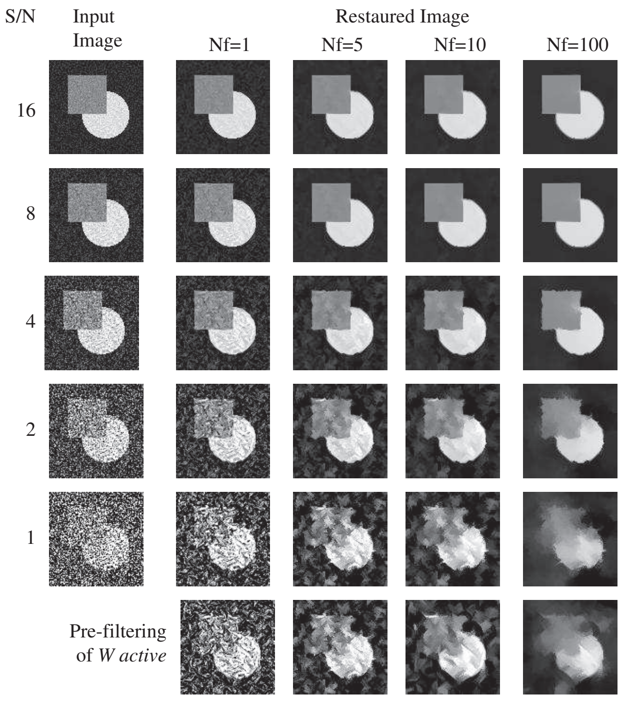

Figure 6. Application of the algorithm applied on synthetic images with different noise amplitudes (on per row), and number of filtering iterations (one per column).

a filtering stage of the vector field W using the filter F_W defined above. The filter F_l is a linear filtering made of 9 coefficients.The following observations can be made:

The noisy input image with S/N of 2 has a PSNR of 6. The processed images with the number of iterations equal to 5, 10 and 100, have respectively a PSNR of 17, 17.9, 18.9.

The Restoration is stable whatever is the number of filtering iteration N_f: the anisotropic diffusion stops and preserves the sharp contours and corners. N_f is also the "time" parameter of the heat equation [2, 3, 1]: when the time is converging to infinite, the solution of the heat equation is converging to a constant. Existing PDE's share the same properties. By opposition the restoration defined in this paper makes restored images converging to a stable state where sharp details and corners are kept intact. This is a major distinction versus prior anisotropic diffusion.

The last row (Figure 6) shows the effect of the pre-filtering of the vector field W. It is worth pre-filtering W for poor signal to noise ratio. The stencil which are re-defined after the pre-filtering are more coherent. The restored images have sharper contours compared to the version with no pre-filtering. The gray rectangle is not properly restored, its S/N is equal to half of the one of white disk.

The input image has $1MPixel$ (1024×1024). The computer used for computation is an Intel Core2 CPU running at $2.4GHz$. The implementation of the code is parallelized and is therefore twice faster. For this test, the image are processed in float, faster computation is achieved using integer format (as for the real case test). The computation time organized as follows: $45ms$ for the computation of the vector field; $180ms$ for the re-orientation of the stencils (including the 3 re-orientation steps); and $35ms$ per cycle of filtering. The computation time for the complete algorithm with 5 iterations of filtering is equal to: $451ms$ which corresponds to a computation rate of $2.2MPixel/sec$.

This algorithm is fast because its internal complexity is small: The re-orientation step relies on a simple criteria; The filtering steps is made fast because the kernel size is small; The table of stencils offers an efficient implementation. This algorithm is by nature fast and designed for real-time processing.

4.2. Real Cases

The algorithm is applied on a well known image. The Table 1 (right column) gives the parameters used for this real case image. The filter F_l used for the filtering is the one described in sub-section 3.6.2. The parameter R is set to 64, this value is chosen in comparison to the noise of the degraded image.

Figure 7 illustrated the restoration on 2 areas of the same image. For each area is shown in row: First, the input image; Second, the degraded version with an additive Gaussian noise $\sigma = 20$; Third, the restored version; And last, the difference between the input image and the restored one. The two following observations can be made:

The global quality of the restoration is good, residual noise are still visible on the restored image. Also, on homogeneous area smoothing is not strong, this is due to the constant number of pixels per stencil. The PSNR of the full restored image is equal to 26.6.

The total computation time is equal to $150ms$, the size of the image is $0.43MPixel$. The computation rate is therefore $2.9Mpixel/sec$. This rate is equal to the rate of a video sequence made of 25 frames per second, each frame having a size of 400×300 pixels. By comparison, another algorithm based on PDE filtering on the same computer offers a processing speed of $0.20Mpixel/sec$ for comparable restoration (same restored PSNR).

5. CONCLUSION

We have designed an anisotropic filter taking advantages of PDE's and LIC formalism. The method defines an anisotropic diffusion which preserves sharp details while decreasing the noise. Also, if the filter is iterated indefinitely to an image, the solution converges to a stable state. This property contrasts to other filters based on PDE's. Finally, the filter is one to several magnitudes faster than other methods based on PDE's. The algorithm has a computation rate which is comparable to the rate of standard resolution video sequences, thus offering a real-time processing.

This model is based on few parameters, it establishes a clear distinction between the shape of a kernel and type of the filtering itself. This approach is therefore very flexible. This model can be expanded further to color images by

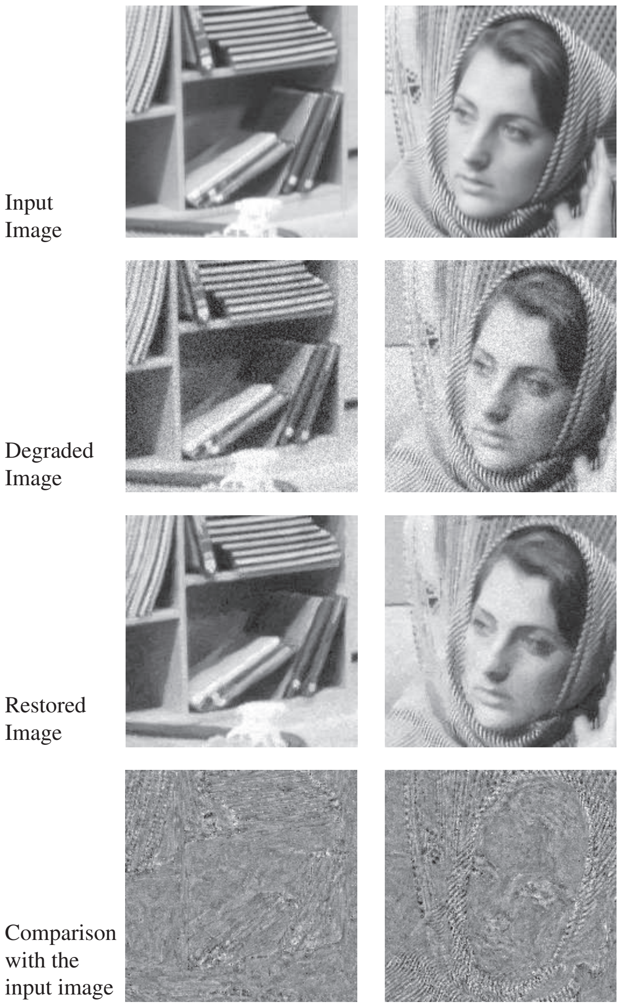

Figure 7. Illustration of the algorithm on 2 images. The first row represents the input image, the second row shows the degraded version by a Gaussian noise ($\sigma = 20$), the third row gives the restored version, and finally, the last row points out the difference between the input image and the restored one. The PSNR of the complete restored frame is equal to 26.6.

computing the vector field from the gradients of each color plan of the image (as suggested in [17]).

6. REFERENCES

[1] P. Perona and J. Malik, "Scale-space and edge detection using anisotropic diffusion," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 12, no. 7, pp. 629–639, 1990.

[2] G. Sapiro, *Geometric partial differential equations and image analysis*, Cambridge University Press, New York, NY, USA, 2001.

[3] J. Weickert, *Anisotropic Diffusion in Image Processing*, Teubner-Verlag, 1998.

[4] B. Cabral and L. Leedom, "Imaging vector fields using line integral convolution," in *Proceedings of SIGGRAPH*. 1993, pp. 263–270, ACM Press.

[5] Detlev Stalling and Hans-Christian Hege, "Fast and resolution independent line integral convolution," in *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, New York, NY, USA, 1995, pp. 249–256, ACM.

[6] D. Tschumperlé, "Fast anisotropic smoothing of multi-valued images using curvature-preserving PDE's," *Int. J. Comput. Vision*, vol. 68, no. 1, pp. 65–82, 2006.

[7] C. Tomasi and R. Manduchi, "Bilateral filtering for gray and color images," in *ICCV '98: Proceedings of the Sixth International Conference on Computer Vision*, Washington, DC, USA, 1998, p. 839, IEEE Computer Society.

[8] Danny Barash, "A fundamental relationship between bilateral filtering, adaptive smoothing, and the nonlinear diffusion equation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, no. 6, pp. 844–847, 2002.

[9] A. Foi, V. Katkovnik, K. Egiazarian, and J. Astola, "A novel anisotropic local polynomial estimator based on directional multiscale optimizations," in *Proc. of the 6th IMA Int. Conf. Math. in Signal Processing, Cirencester (UK)*, 2004, pp. 79–82.

[10] J. Polzehl and V. G. Spokoiny, "Adaptive weights smoothing with applications to image restoration," *Journal Of The Royal Statistical Society Series B*, vol. 62, no. 2, pp. 335–354, 2000.

[11] J. Polzehl and V. Spokoiny, "Image denoising: Pointwise adaptive approach," Tech. Rep., 1998.

[12] J. Portilla, V. Strela, M.J. Wainwright, and E.P. Simoncelli, "Image denoising using scale mixtures of gaussians in the wavelet domain," *Image Processing, IEEE Transactions on*, vol. 12, no. 11, pp. 1338–1351, Nov. 2003.

[13] M.I. Gurelli and L. Onural, "A class of adaptive directional image smoothing filters," vol. 29, no. 12, pp. 1995–2004, December 1996.

[14] J. Bresenham, *Algorithm for Computer Control of a Digital Plotter.*, 1965.

[15] D. Tschumperle and R. Deriche, "Constrained and unconstrained pdes for vector image restoration," 2001, pp. O–M5B.

[16] D. Tschumperle and R. Deriche, "Diffusion pdes on vector-valued images," *Signal Processing Magazine, IEEE*, vol. 19, no. 5, pp. 16–25, Sep 2002.

[17] Silvano Di Zenzo, "A note on the gradient of a multi-image," *Comput. Vision Graph. Image Process.*, vol. 33, no. 1, pp. 116–125, 1986.