Fast GPU-based denoising filter using isoline levels

Gilles Perrot[1], Stéphane Domas[1], Raphaël Couturier[1], Nicolas Bertaux[2]
[1]FEMTO-ST institute
Rue Engel Gros, 90000 Belfort, France.
forename.name@univ-fcomte.fr
[2] Institut Fresnel, CNRS, Aix-Marseille Université, Ecole Centrale Marseille,
Campus de Saint-Jérôme, 13013 Marseille, France.
nicolas.bertaux@ec-marseille.fr

***Abstract*—In this study, we propose to address the issue of image denoising by means of a GPU-based filter, able to achieve high-speed processing by taking advantage of the parallel computation capabilities of modern GPUs. Our approach is based on the level sets theory first introduced by [1] in 1975 but little implemented because of its high computation costs. What we actually do is try to guess the best isoline shapes inside the noisy image. At first, our method involved the polyline modelling of isolines; then we found an optimization heuristics which very closely fits the capabilities of GPUs. So far, though our proposed hybrid PI-PD filter has not achieved the best denoising levels, it is nonetheless able to process a 512x512 image in about 11 ms.**

***Keywords*-GPU; denoising; filter; isoline;level line;**

I. Introduction

Denoising has been a much studied research issue since electronic transmission was first used. The wide range of applications that involve denoising makes it uneasy to propose a universal filtering method. Among them, digital image processing is a major field of interest as the number of digital devices able to take pictures or make movies is growing fast and shooting is rarely done in optimal conditions. Moreover, the increase in pixel density of the CCD or CMOS sensors used to measure light intensity leads to higher noise effects and imposes high output flow rates to the various processing algorithms.

In addition, it is difficult to quantify the quality of an image processing algorithm, as visual perception is subject to high variation from one human to another. So far, the advent of GPUs has brought high speedups to a lot of algorithms, and many researchers and developpers have successfully adressed the issue of implementing existing algorithms on such devices. For example in [2], [3] and [4], authors managed to design quite fast median filters. Bilateral filtering has also been successfully proposed in [5]. Still, most high quality algorithms, like NL-means [6] or BM3D [7] make use of non-local similarities and/or frequency domain transforms. However, speedups achieved by their current GPU implementations, though quite sigificant (as shown for example with NL-means in [8]), do not come near those achieved by local methods such as gaussian, median or neighborhood filters, as they have not originally been designed against GPU architecture. In order to fully benefit from the capabilities of GPUs, it is important that the approach to designing algorithms be more hardware-oriented, keeping in mind, from the very beginning, the intrinsic constraints of the device which is actually going to run those algorithms. Consequently, this often results in unusual options and even apparently sub-optimal solutions, but the considerable speed benefits obtained would possibly make it at least a good compromise or even the only current way to real-time high-definition image processing.

II. Contribution

As early as 1975 [1], it was found that, under the conditions mentioned in section V, an image can be decomposed into a set of level lines. Accordingly, real-life images fulfill the above conditions and since then, with the increase of computing capabilities, researchers have succeded in implementing such level-lines based algorithms as in [9] and [10]. A few years ago, in [11], authors proposed an original method which significantly reduces speckle noise inside coherent images, using the level lines in the image to constrain the minimization process. Those level lines are actually *iso-gray-level* lines, which are called *isolines*. In [11], isolines consist in neighborhoods of polyline shapes determined by maximum likelihood optimization. This method proved not only to bring good enhancement but also to preserve edges between regions. Nevertheless, the costs in computation time, though not prohibitive, did not allow real-time image processing; as an example, the authors of [11] managed to process an almost 2Mpixel image within a minute on an old PIII-1GHz.

Our work started by designing a set of GPU implementations with various optimization heuristics, in order to find out which tracks could be followed towards minimizing loss in quality and preserve admissible execution times. Those algorithms have been tested with reference images taken from [12] for which various processing results have been published. Some of the more interesting ones are listed and compared in [13]. Statistical observations (to be detailed below) made on the output images produced by the method proposed in [11], led us to propose a very fast and simple parallel denoising method which gives good results in terms of average gray-level error, but also avoids the blurring of edges.

On the basis of the BM3D timings listed in [7] and with our own measurements, our proposed GPU-based filter runs around 350 times faster and thus is able to process high definition images at over 16fps. It also achieves good denoising quality.

III. PLAN

In the following, section IV briefly focuses on recent Nvidia GPU characteristics. Section V will introduce the theory and notations used to define isolines. Then, in section VI, we will describe the two isoline based models that led to the final hybrid model, while section VII details the parallel implementation of the proposed algorithm. Finally, we present our results in section VIII before drawing our conclusions and outlining our future works in section IX.

IV. NVIDIA'S GPU ARCHITECTURE

GPUs are multi-core, multi-threaded processors, optimized for highly parallel computation. Their design focuses on a SIMT model (Single Instruction Multiple Threads) that devotes more transistors to data processing rather than data-caching and flow control (see [14] for more details). For example, a C2070 card features 6 GBytes global memory and a total of 448 cores bundled in several Streaming Multiprocessors (SM). An amount of shared memory, much faster than global memory, is avalaible on each SM (up to 48 KB for a C20xx card)

Writing efficient code for such architectures is not obvious, as re-serialization must be avoided as much as possible. Thus, code design requires one pays attention to a number of points, among which:

- the CUDA model organizes threads by a) thread blocks in which synchronization is possible, b) a grid of blocks with no possible synchronization between them.
- there is no way to know how blocks are scheduled during one single kernel execution.
- data must be kept in GPU memory, to reduce the overhead generated by copying between CPU and GPU.
- the total amount of threads running the same computation must be as large as possible.
- the number of execution branches inside one block should be as small as possible.
- global memory accesses should be coalescent, *i.e.* memory accesses done by physically parallel threads (2 x 16 at a time) must be consecutive and contained in a 128 Bytes range.
- shared memory is organized in 32x32 bit-wide banks. To avoid bank conflicts, each parallel thread (2 x 16 at a time) must access a different bank.

All the above characteristics always make designing efficient GPU code all the more constraining as non-suited code would probably run even slower on GPU than on CPU.

V. ISOLINES

In the following, let I be the reference noiseless image (assuming we have one), I' the noisy acquired image corrupted by Independent and Identically Distributed (IID) additive white gaussian noise of zero mean value and standard deviation σ. Let $\widehat{I}$ be the denoised image. Each pixel of I' of coordinates (i, j) has its own gray level $z(i, j)$.

As introduced above and since most common images are continuous and contain few edges, they can be decomposed into a set of constant gray level lines called *isolines*. Then our goal is to find, for each single pixel of a noisy image, the isoline it belongs to. The generalized likelihood criterion (GL) is used to select the best isoline among all the considered ones, all of which must have the same number of pixels in order to be compared.

A. Fixed-length isolines

For each pixel (i, j) of the corrupted image, we look for the gray level of the isoline it belongs to, inside a rectangular window ω centered on (i, j). Inside ω, let S^n be the isoline part which the center pixel belongs to. S^n is a set of n pixel positions (i_q, j_q) ($q \in [0; n[$).
The gray levels z along S^n follow a gaussian probability density function whose parameters μ_{S^n} (mean value of isoline part) and σ (standard deviation brought by gaussian noise) are unknown.
Let $\overline{S^n}$ be defined by $\omega = S^n \cup \overline{S^n}$.
For each pixel, the mean values μ_{ij} of gray levels z over $\overline{S^n}$ are unknown and supposed independant .
Let Z be the gray levels of pixels in ω and $\{\mu_{ij}\}_{\overline{S^n}}$ the mean values of pixels in $\overline{S^n}$. The likelihood is given by:

$$P\left[Z|S^n, \mu_{S^n}, \{\mu_{ij}\}_{\overline{S^n}}, \sigma\right]$$

When separating contributions from regions S^n and $\overline{S^n}$, it becomes:

$$\prod_{(i,j)\in S^n} P\left[z(i,j)|\mu_{S^n}, \sigma\right]. \prod_{(i,j)\in \overline{S^n}} P\left[z(i,j)|\left\{\mu_{ij}\right\}_{\overline{S^n}}, \sigma\right] \quad (1)$$

The goal is then to estimate the value of the above expression, in order to find the boundaries of S^n that maximize expression (1).
Let us consider that, on $\overline{S^n}$, the values $z(i, j)$ are the likelihood estimations $\widehat{\mu_{ij}}$ for μ_{ij}. The second term of expression (1) becomes:

$$\prod_{(i,j)\in \overline{S^n}} P\left[z(i,j)|\left\{\widehat{\mu_{ij}}\right\}_{\overline{S^n}}, \sigma\right] = 1 \quad (2)$$

which leads to the generalized likelihood expression:

$$\prod_{(i,j)\in S^n} P\left[z(i,j)|\mu_{S^n}, \sigma\right] \quad (3)$$

As we know the probability density function on S^n, (3) can then be developped as

$$\prod_{(i,j)\in S^n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(z(i,j)-\mu_{S^n})^2}{2\sigma^2}} \quad (4)$$

The log-likelihood is then given by:

$$-\frac{n}{2}log\left(2\pi\right) - \frac{n}{2}log\left(\sigma^2\right) - \frac{n}{2} \quad (5)$$

inside which the vector of parameters (μ_{S^n}, σ) is determined by maximum likelihood estimation

$$\left(\begin{array}{l} \widehat{\mu_{S^n}} = \frac{1}{n} \sum_{(i,j)\in S^n} z(i,j) \\ \widehat{\sigma^2} = \frac{1}{n} \sum_{(i,j)\in S^n} \left(z(i,j) - \widehat{\mu_{S^n}}\right)^2 \end{array}\right.$$

The selection of the best isoline is done by searching which one maximizes the expression of equation (5).

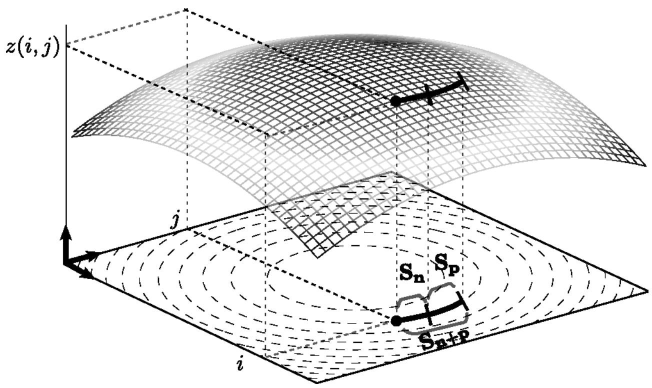


Fig. 1. Determination and lengthening of an isoline: The gray level z of each pixel is seen as an elevation value. S^n is the n pixel length isoline for pixel of coordinates (i, j). The elongation of S^n by S^p (p pixel length) is submitted to the GLRT condition (see eq. (8)).

B. Lengthenable isolines

Searching for larger isolines should lead to better filtering as a larger number of pixels would be involved. However, processing all possible isolines starting from each pixel would be too costly in computing time, even in the case of a small GPU-processed 512x512 pixel image. Therefore, we chose to build large isolines inside an iterative process including a mandatory validation stage between each lengthening iteration, so as to reduce the number of pixel combinations to be examined and keep the estimation of deviation σ within a satisfactory range of values.

Let S^n be a previously selected isoline part and S^p connected to S^n in such a way that S^p could be seen as an addition to S^n so as to define a possible valid isoline S^{n+p}. Figure 1 illustrates this situation with a very simple example image. In this figure, the gray level of each pixel is used as its corresponding height (z) in order to visualize isolines easily. Some of the orthogonal isoline projections have been drawn in dotted line in the $(\vec{i}, \vec{j})$ plane. Both labeled parts S^p and S^n are represented in the $(\vec{i}, \vec{j})$ plane and in the 3D associated plot.

In order to decide whether S^{n+p} can be considered as an actual isoline, we compare the log-likelihood of both hypothesis below by using GLRT (Generalized Likelihood Ratio Test):

First, assuming that S^{n+p} is an isoline, the gray levels of its pixels share the same mean value μ_{n+p}. According to (5), its log-likelihood is

$$-\frac{(n+p)}{2}\left(log\left(2\pi\right)+1\right)-\frac{(n+p)}{2}log\left(\widehat{\sigma_1}^2\right) \tag{6}$$

where $\widehat{\sigma_1}$ is the estimation of the standard deviation along S^n.

Second, considering S^n and S^p as two separate isoline parts connected together, the gray levels of their pixels have two different mean values μ_n and μ_p. The log-likelihood is the sum of both log-likelihoods, given by

$$-\frac{(n+p)}{2}\left(log\left(2\pi\right)+1\right)-\frac{n}{2}log\left(\widehat{\sigma_2}^2\right)-\frac{p}{2}log\left(\widehat{\sigma_2}^2\right) \tag{7}$$

where $\widehat{\sigma_2}$ is the estimation of the standard deviation along S^n and S^p.

The difference between (6) and (7) leads to the expression of $GLRT(S^{n+p}, S^n, S^p, T_{max})$:

$$T_{max}-(n+p).\left[log\left(\widehat{\sigma_1}^2\right)-log\left(\widehat{\sigma_2}^2\right)\right] \tag{8}$$

The decision to validate lengthening from S^n to S^{n+p} depends whether $GLRT(S^{n+p}, S^n, S^p, T_{max})$ is higher or lower than 0. Value T_{max} is the GLRT threshold.

VI. ISOLINE MODELS

The most obvious model considers isolines as polylines. Each isoline can then be curved by allowing a direction change at the end of each segment; we shall call such isolines *poly-isolines*.

In order to keep the number of candidate isolines within reasonable range, we chose to build them by combinating segments described by simple pre-computed patterns. Each pattern $p_{l,d}$ describes a segment of length l and direction d. For one given l value, all $p_{l,d}$ patterns are grouped into a matrix denoted P_l. Figure 8 shows an example of such a pattern matrix for $l = 5$.

To fit the GPU-specific architecture, we define regularly distributed D primary directions ($D = 32$ in our examples).

A. Poly-isolines with limited deviation angle (PI-LD)

At one stage we implemented an algorithm parsing the tree of all possible polyline configurations, but the process proved far too slow regarding our goal, even on GPU, because of the amount of memory involved (and consequent memory accesses) and because of the necessary reduction stage for which GPU efficiency is not maximum. So we focused on a variant inspired by [11] in which the selected direction of the next segment depends on the whole of the previously built and validated poly-isoline.

Let us consider a poly-isoline S^n under construction, starting from pixel (i, j) and made of K validated segments s_k ($k \in [1; K]$) of length l, each of them having its own direction d_k. The coordinates of the ending pixel of each segment s_k are denoted (i_k, j_k). Both of the following sums

$$C_x\left(Z(S^n)\right) = \sum_{(i,j)\in S^n} z(i,j) \tag{9}$$

$$and \;\; C_{x^2}\left(Z(S^n)\right) = \sum_{(i,j)\in S^n} z(i,j)^2 \tag{10}$$

have been obtained during the previous lengthening steps.

Let us examine now how to decide wether to add a new segment to S^n or to stop the lenghtening process. The main idea is to apply each pattern $p_{l,d}$ to the ending pixel (i_k, j_k), on the condition that its direction is contained within the limits of maximum deviation Δd_{max}. Maximum deviation Δd_{max} prevents poly-isolines from beeing of circular shape (or backward-oriented) which would possibly generate supplementary artefacts in the output image. Another of its benefits is to reduce the number of combinations to be evaluated.

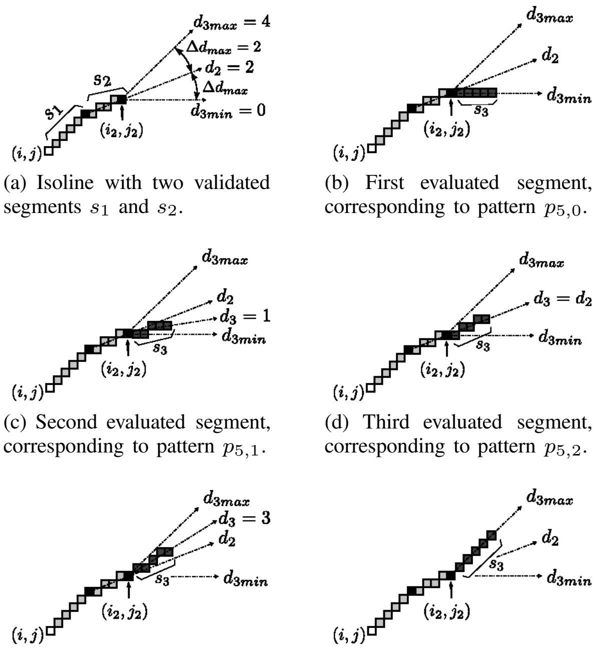
For each allowed pattern, GLRT is performed in order to decide if the corresponding segment could likely be added to the end of the poly-isoline S^n. If none is validated by GLRT, the poly-isoline S^n is stopped.

If at least one segment has been accepted by GLRT, the one that leads to the maximum likelihood (ML) value of the lengthened poly-isoline S^{n+l} is selected and integrated to S^{n+l} as s_{K+1}.

In order to avoid critical situations where the first selected segment would not share the primary direction of the actual poly-isoline, no selection is performed on the level of the first segment; D poly-isolines are kept and submitted to the lengthening process. To ensure isotropy, each of them shares the direction of one pattern $p_{l,d}$ $(d \in [0; D])$.

Eventually, the poly-isoline with the maximum likelihood value is selected among the longest ones.

Figure 2 illustrates one stage of the lengthening process with the example of a two-segment poly-isoline at the beginning of stage ($l = 5$ and $\Delta d_{max} = 2$).



(a) Isoline with two validated segments s_1 and s_2.

(b) First evaluated segment, corresponding to pattern $p_{5,0}$.

(c) Second evaluated segment, corresponding to pattern $p_{5,1}$.

(d) Third evaluated segment, corresponding to pattern $p_{5,2}$.

(e) Fourth evaluated segment, corresponding to pattern $p_{5,3}$.

(f) Fifth evaluated segment, corresponding to pattern $p_{5,4}$.

Fig. 2. Example of lengthening process starting with a two-segment poly-isoline ($l = 5$, $\Delta d_{max} = 2$). The initial situation is shown in 2a, while 2b to 2f represent the successive candidate segments. The direction index of the last validated segment is $d_2 = 2$ (2a). It implies that direction indexes allowed for the third segment range from $d_2 - \Delta dmax = 0$ to $d_2 + \Delta dmax = 4$ (2b to 2f). The lengthening of the poly-isoline is accepted if at least one segment has a positive GLRT. If there are several, the one which minimizes the standard deviation of the whole poly-isoline is selected.

B. *Poly-isolines with precomputed directions (PI-PD)*

Though much faster, the PI-LD-based filter may be considered a bit weak compared to *state-of-the-art* filters like BM3D family algorithms [7]. Furthermore, we saw that this way of building poly-isolines requires the alternate use of two different types of validation at each lengthening stage: GLRT and maximum likelihood minimization. In order to be performed, each of them generates numerous branches during kernel execution, which does not fit GPU architecture well and leads to execution times that we hoped would be more impressive.

Within the PI-LD model, at each pixel (i, j), as no selection is done at the first stage, D poly-isolines are computed and kept as candidate though, obviously, only one follows the actual isoline at (i, j). So, if we assume we can achieve a robust determination of the direction at any given pixel of this isoline, it becomes unnecessary to perform the selection at each lenghtening step. Thus, at each pixel (i, j), only the first segment has to be determined in order to obtain the local direction of the isoline. This leads to an important reduction of the work complexity: the above PI-LD model needs to evaluate $D.\left(2.\Delta_{dmax} + 1\right)^{K-1}$ segments at each pixel position, while only $D.K$ evaluations are needed in the second case. For example, with a maximum of $K = 5$ segments and a maximum deviation of $\Delta_{dmax} = 2$, the PI-LD needs to evaluate up to 20000 segments per pixel where only 160 should be enough.

On the basis of these observations, we propose a new model that we shall call PI-PD, that completely separates the validation stages performed in the PI-LD model implementation mentioned above:

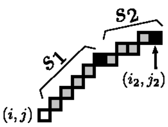
A first computation stage selects the best first segment s_1 starting at each pixel (i, j) of the input image. Its direction index $d_1(i, j)$ is then stored in a reference matrix denoted I_Θ; sums C_x and C_{x2} along $s_1(i, j)$ are also computed and stored in a dedicated matrix I_Σ. It can be noticed that this selection method of s_1 segments is a degraded version of PI-LD constrained by $K = 1$.

A second stage manages the now independant lengthening process. For one given state of a poly-isoline where the last added segment has been s_K, the pattern whose direction index is given by $d = I_\Theta(i_K, j_K)$ defines the only segment to be evaluated. Both corresponding sums C_x and C_{x2} are read from matrix I_Σ and used in GLRT evaluation. The last point is to prevent poly-isolines from turning back.

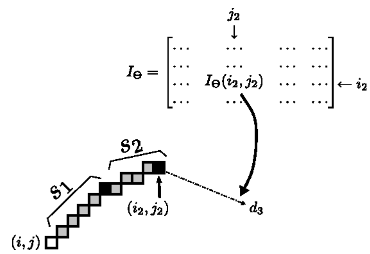
Figure 3 details this process, starting from the same initial state as in figure 2 with the noticeable difference that no deviation limit is needed.

Thus, as introduced above, work complexity is considerably reduced, as each pattern is only applied once at one given pixel (i, j), and associated values are computed only once; they are re-used every time one poly-isoline's segment ends at pixel (i, j). Also, this fits GPU constraints better, as it avoids multiple branches during kernel execution. It remains that, the building of poly-isolines is done without global likelihood optimization.

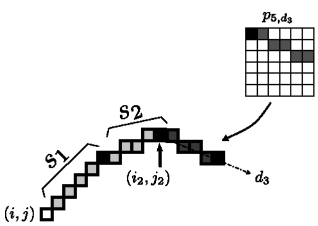
Eventually, the model has been improved by adding to it the ability to thicken poly-isolines from one pixel up to three which allows to achieve higher PSNR values by increasing the number of pixels of poly-isolines in addition to the lengthening process. This may apply to large images which do not contain small relevant details, as it may blur small significant details or objects present in the noisy image. Still, this feature makes PI-PD more versatile than our reference BM3D, which has prohibitive computation times when processing large images (over 5 minutes for a 4096x4096 pixel image) and thus should require a slicing stage prior to processing them, causing some overhead.



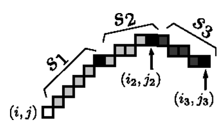
(a) Poly-isoline with two validated segments.



(b) Next direction is read from element (i_2, j_2) of I_Θ.



(c) Pattern p_{l,d_3} is then applied at (i_2, j_2) and GLRT is performed. Both sums needed to perform GLRT are read from element (i_2, j_2) of I_Σ.



(d) If accepted by GLRT, segment s_3 is added to poly-isoline.

Fig. 3. Example of PI-PD lengthening process starting with a two-segment poly-isoline ($l = 5$). The initial situation is represented in 3a, while 3a to 3d represent the successive processing steps. The end pixel of the last validated segment is (i_2, j_2) (3a). Reference matrices I_Θ and I_Σ provide the values needed to select the pattern to be applied on (i_2, j_2) (3b and 3c). GLRT is performed to validated lengthening or not. This process goes on until one submitted segment does not comply with GLRT.

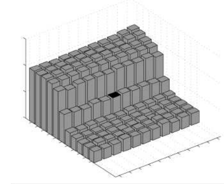
C. Hybrid PI-PD

As the determination of each segment's direction only involves a few pixels, the PI-PD model may not be robust enough in regions where the surface associated with Z has a low local slope value regarding power of noise σ^2. We shall call those regions Low Slope Regions (LSR). Figure 4 shows this lack of robustness with an example of two drawings of additive white gaussian noise applied on the same reference image (Figure 6). Within this image, we focused on a small 11x11 pixel window containing two LSR with one sharp edge between them.

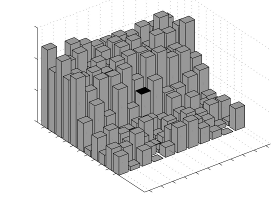
Figures 4d and 4e show that the directions computed by PI-PD are identical from one drawing to the other near the edge (lines 5-7), while they vary in LSR (lines 1-4, 8-11).

Within such regions, our speed goals forbid us to compute isoline directions with the PI-LD model, more robust but far too slow. Instead we propose a fast solution which implies designing an edge detector whose principle is to re-use the segment patterns defined in section VI and to combine them by pairs in order to detect any possible LSR around the center pixel. If a LSR is detected, the output gray-level value is the average value computed on the current square window, otherwise, the PI-PD output value is used.

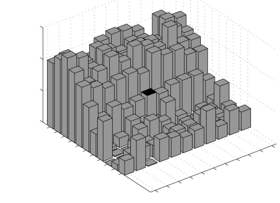
In order to further simplify computation, only the patterns that do not share any pixel are used. These patterns have a direction which is a multiple of $45°$.



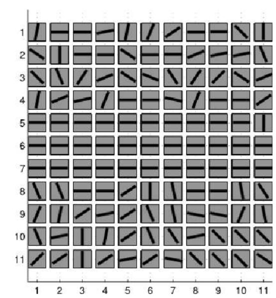
(a) Reference image



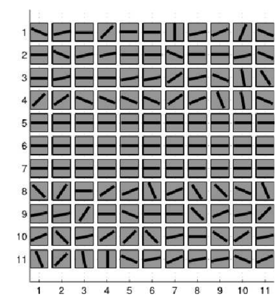
(b) Image corrupted by random drawing $n°1$



(c) Image corrupted by random drawing $n°2$



(d) Isoline directions for random drawing $n°1$



(e) Isoline directions for random drawing $n°2$

Fig. 4. Zoom on a small square window of the airplane image. 4a reproduce the zoom on the window, taken from the reference image of Figure 6. 4b, 4c and 4a and are 3D views where each bar represents a pixel whose gray-level corresponds to the height of the bar. Figures 4d and 4e are 2D top views of the window. The chosen window shows an edge between two regions of low slope. The images 4b and 4c are corrupted with two different random drawings of the same additive white gaussian noise (AWGN) of power σ^2 and mean value 0. 4d and 4e show, for each pixel of the window, the direction of the isoline found by PI-PD. In regions of low slope (the two regions at the top and the bottom), the determination of the direction is not robust. But near the edge, directions do not vary from one drawing to another.

Each base direction (Θ_i) and its opposite $(\Theta_i + \pi)\,[2\pi]$ define a line that separates the square window in two regions (top and and bottom regions, denoted T and B). We assume that segments on the limit belong to the T region which includes pixels of orientation from Θ_i to $\Theta_i + \pi$. This region comprises three more segments of directions $(\Theta_i + \frac{\pi}{4})$, $(\Theta_i + \frac{2\pi}{4})$ and $(\Theta_i + \frac{3\pi}{4})$. The other region (B) only includes three segments of directions $(\Theta_i + \frac{5\pi}{4})$, $(\Theta_i + \frac{6\pi}{4})$ and $(\Theta_i + \frac{7\pi}{4})$.

Figure 5 illustrates this organization for $\Theta_i = \Theta_4 = 45°$. Each bar represents a pixel in the detector's window. Pixels with null height are not involved in the GLRT. Pixels represented by higher bars define the T region and those represented by shorter bars define the B region.

For each Θ_i, one GLRT is performed in order to decide whether the two regions T and B defined above are likely to be seen as a single region or as two different ones, separated by an edge as shown in figure 5. The center pixel is located on the edge. Equations (6), (7) and (8) lead to a similar GLRT

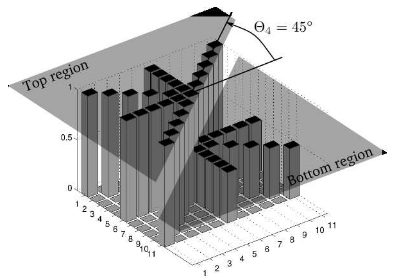


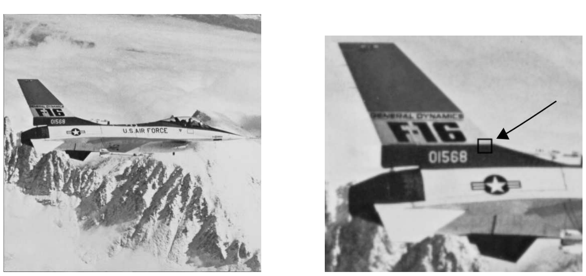
Fig. 5. Edge detector. 3D view representing an example square 11x11 pixel window ($l = 5$) used in the edge detector for $\Theta_4 = 45°$ around a center pixel colored in black. Each pixel is represented by a bar. Bars of height value 0 are for pixels that are not involved in the detector. Top region is defined by five pattern segments and includes the center pixel. Bottomp region only includes three pattern segments. The different height values are meant to distinguish between each of the three different sets of pixels and their role.

expression:

$$T2_{max} - (8.l + 1). \left[log\left(\widehat{\sigma_3}^2\right) - log\left(\widehat{\sigma_4}^2\right) \right] \tag{11}$$

where σ_3 is the standard deviation considering that the two regions are likely to define a single one and σ_4 the standard deviation if an edge is more likely to separate the two regions. $T2_{max}$ is the decision threshold. With equation (11), a negative result leads to an edge detection, oriented towards direction Θ_i. When GLRT is known for each Θ_i, we apply the following hybridation policy:

a) more than one negative GLRT: the PI-PD output value is used.
b) only one negative GLRT: the center pixel is likely to be on a well-defined edge, and only the region it belongs to is considered. The average value of its pixel gray levels is then used.
c) no negative GLRT: the window around the center pixel is likely to be a LSR. The average value on the whole square window is used (11x11 pixels in the example of Figure 5).

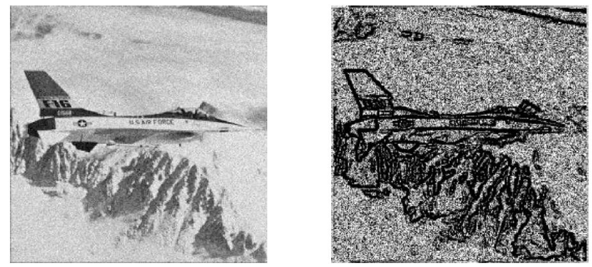


(a) Reference noiseless airplane image

(b) Location of the example window in the reference image.

Fig. 6. Location of the example window inside the reference image. Figure 6a shows the whole reference image and 6b zooms on the part where the example 11x11 pixel window is.

It must be noticed that point b) has been introduced in order to achieve smoother transitions between regions to which PI-PD is applied and those in which the plain average value is used. Figure 7 shows an example of such a classification achieved by the edge detector. The detector has been applied on the top noisy airplane image with a GLRT threshold value $T2_{max} = 2$. Black pixels represent pixel classified as *on an edge*, while white ones are those which belong to LSR.



(a) Noisy airplane image

(b) Pixel classification performed by the edge detector.

Fig. 7. Pixel classification inside the noisy image. Figure 7a shows the noisy input image and 7b reproduces the output classification of pixels, as a black and white image, obtained with threshold value $T2_{max} = 2$. Black pixels are supposed to be near an edge, while white pixels belong to Low Slope Regions.

VII. Hybrid PI-PD filter Implementation: details

All implementation details that will be given here are relative to the proposed PI-PD models and Nvidia© GPU devices.

A. Segment patterns

The first kernel to be run is `kernel_genPaths()` which generates matrix P_l. Its elements $(\Delta i; \Delta j)$ are the relative coordinates of the pixels which define segment patterns $p_{l,d}$. The dimensions of matrix P_l are D rows $\times$ l columns. To fit GPU architecture as closely as possible, we chose $D = 32$ patterns. Each segment s_k of a poly-isoline can then be seen as a pattern $p_{l,d}$ applied on the starting pixel (i, j) of this segment, denoted $p_{l,d}(i, j)$.

The example in figure 8 shows the first quarter of P_5 and the corresponding eight discrete segment patterns in the first quadrant. The three remaining quarters of the matrix are easily deduced by applying successive rotations of angle $\frac{\pi}{2}$ to the above elements.

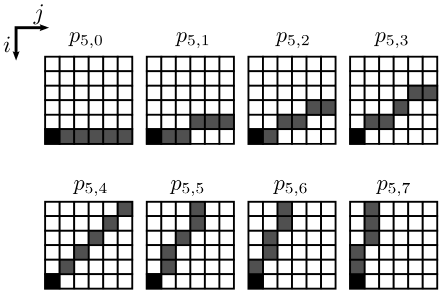
B. Generation of reference matrices I_Σ and I_Θ

In order to generate both matrices, a GPU kernel `kernel_precomp()` computes, in parallel for each pixel (i, j):

- the direction δ of the most likely segment $s_1 = p_{l,\delta}(i, j)$ among the D possible ones. This value is stored in matrix I_Θ at position (i, j).
- values $C_x(s_1)$ and $C_{x^2}(s_1)$ defined in equations (9) and (10). This vector of values is stored in matrix I_Σ at position (i, j).

In order to reduce processing time, the input image is first copied into texture memory (see algorithm 1 for initializations and memory transfer details), thus taking advantage of the 2D optimized caching mechanism.

This kernel follows the *one thread per pixel* rule. Consequently, each value of P_l has to be accessed by every thread



$$P_5 = \begin{bmatrix} (0,1) & (0,2) & (0,3) & (0,4) & (0,5) \\ (0,1) & (0,2) & (-1,3) & (-1,4) & (-1,5) \\ (0,1) & (-1,2) & (-1,3) & (-2,4) & (-2,5) \\ (-1,1) & (-1,2) & (-2,3) & (-3,4) & (-3,5) \\ (-1,1) & (-2,2) & (-3,3) & (-4,4) & (-5,5) \\ (-1,1) & (-2,1) & (-3,2) & (-4,3) & (-5,3) \\ (-1,0) & (-2,1) & (-3,1) & (-4,2) & (-5,2) \\ (-1,0) & (-2,0) & (-3,1) & (-4,1) & (-5,1) \\ \cdots & \cdots & \cdots & \cdots & \cdots \end{bmatrix}$$

Fig. 8. Top: example segment patterns $p_{5,d}$ for $d \in [0;7]$; the black pixel represents the center pixel (i,j), which does not belong to the pattern. The gray ones define the actual pattern segments. Bottom: the first 8 lines of corresponding matrix P_5 whose elements are the positions of segment pixels with respect to the center pixel.

of a block. That led us to load it from texture memory first, then copy it into all shared memory blocks. This has proved to be the fastest scheme.

Algorithm 2 summarizes the computations achieved by `kernel_precomp()`. Vector (C_x, C_{x2}) stores the values of $C_x(s_1)$ and $C_{x^2}(s_1)$ associated with the current tested pattern. Vector $(C_{x-best}, C_{x2-best})$ stores the values of $C_x(s_1)$ and $C_{x^2}(s_1)$ associated with the best previously tested pattern.

In the same manner, σ and σ_{best} are deviation values for current and best tested patterns.

The selection of the best pattern is driven by the value of the standard deviation of candidate isolines. Lines 2 and 3 compute both sums for the first pattern to be evaluated. Line 4 computes its standard deviation. Then, lines 5 to 14 loop on each pattern and keep values associated with the best pattern found. These values are eventually stored in matrices I_Θ and I_Σ on lines 16 and 17.

Algorithm 1: Initializations in GPU memory

$l \leftarrow$ step size;
$D \leftarrow$ number of primary directions;
$I_n \leftarrow$ noisy image;
$I_{ntex} \leftarrow I_n$; `/* copy to texture mem. */`
$P_l \leftarrow$ kernel_genPaths ; `/* pattern matrix */`
$P_{ltex} \leftarrow P_l$; `/* copy to texture mem. */`
$T_{max} \leftarrow$ GLRT threshold (lengthening);
$T2_{max} \leftarrow$ GLRT threshold (edge detection);

C. PI-PD lengthening process: `kernel_PIPD()`

This parallel kernel is run in order to obtain the image of the *isolines*. It is detailed in algorithm 3, (see section VI-B for process description).

Algorithm 2: generation of reference matrices, kernel `kernel_precomp()`

1: **foreach** *pixel* (i,j) **do** `/* in parallel */`
2: $C_{x-best} \leftarrow \sum_{(y,x)\in p_{l,0}(i,j)} I_{ntex}(i+y, j+x)$;
3: $C_{x2-best} \leftarrow \sum_{(y,x)\in p_{l,0}(i,j)} I^2_{ntex}(i+y, j+x)$;
4: $\sigma_{best} \leftarrow$ standard deviation along $p_{l,0}(i,j)$;
`/* loop on each pattern */`
5: **foreach** $d \in [1; D-1]$ **do**
6: $C_x \leftarrow \sum_{(y,x)\in p_{l,d}(i,j)} I_{ntex}(i+y, j+x)$;
7: $C_{x2} \leftarrow \sum_{(y,x)\in p_{l,d}(i,j)} I^2_{ntex}(i+y, j+x)$;
8: $\sigma \leftarrow$ standard deviation along $p_{l,d}(i,j)$;
9: **if** $\sigma_d < \sigma_{best}$ **then** `/* keep the best */`
10: $C_{x-best} \leftarrow C_x$;
11: $C_{x2-best} \leftarrow C_{x2}$;
12: $\Theta_{best} \leftarrow d$;
13: **end**
14: **end**
15: $I_\Sigma(i,j) \leftarrow [C_{x-best}, C_{x2-best}]$; `/* stores */`
16: $I_\Theta(i,j) \leftarrow \Theta_{best}$; `/* in matrices */`
17: **end**

Lines from 2 to 11 perform allocations for the first lengthening to evaluate. More precisely, (i_1, j_1) represents the starting pixel of the current segment; (i_2, j_2) is both its ending pixel and the starting pixel of the next segment; d_1 and d_2 are their directions, read from precomputed matrix I_Θ. C_x^1 and C_{x2}^1 are the gray-level sums along the current poly-isoline; C_x^2 and C_{x2}^2 are the gray-level sums of the candidate segment. The current poly-isoline ends at (i_1, j_1) and is made of l_1 pixels (already accepted segments); its standard deviation is σ_1. The loop extending from lines 12 to 21 performs the allocations needed to proceed one segment forward, as long as GLRT is true. If the lengthening has been accepted, the length of the poly-isoline is updated in line 13, and the same is done with C_x and C_{x2} which are read from precomputed matrix I_Σ (see equations (9) and (10) for definition). Finally, using direction value d_2, it translates the coordinates (i_1, j_1) to the end of the newly elongated poly-isoline, and (i_2, j_2) to the end of the next segment to be tested. As soon as the GLRT condition becomes false, line 23 eventually produces the output value of the denoised image at pixel (i,j), that is, the average gray-level value along the poly-isoline.

D. Hybrid PI-PD : `kernel_edge_detector()`

As introduced in section VI-C, the aim of kernel `kernel_edge_detector()` is to divide pixels into two classes according to their belonging to a LSR or not. Algorithm 4 explains the detailled procedure. Lines 2 to 6 initialize values of the direction index (Θ), the number of edges detected ($edgeCount$), the gray-level sum along the pixels that defines the H half-plane ($sumEdge$) and the number of pixels that

Algorithm 3: PI-PD lengthening process `kernel_PIPD()`

foreach *pixel* (i, j) **do** `/* in parallel */`
  $(C_x^1, C_{x2}^1) \leftarrow z(i, j)$; `/* starting pixel */`
  $(i_1, j_1) \leftarrow (i, j)$; `/* first segment */`
  $(C_x^1, C_{x2}^1) \leftarrow I_\Sigma(i_1, j_1)$; `/* read matrix */`
  $d_1 \leftarrow I_\Theta(i, j)$; `/* read matrix */`
  $l_1 \leftarrow l$; `/* isoline length */`
  $\sigma_1 \leftarrow (C_{x2}^1/l_1 - C_x^1)/l_1$;
  $(i_2, j_2) \leftarrow end\ of\ first\ segment$;
  $(C_x^2, C_{x2}^2) \leftarrow I_\Sigma(i_2, j_2)$; `/* 2`nd `segment */`
  $d_2 \leftarrow I_\Theta(i_2, j_2)$;
  $\sigma_2 \leftarrow (C_{x2}^2/l - C_x^2)/l$;
  while $GLRT(\sigma_1, \sigma_2, l_1, l) < T_{max}$ **do**
    $l_1 \leftarrow l_1 + l$; `/* lengthening */`
    $(C_x^1, C_{x2}^1) \leftarrow (C_x^1, C_{x2}^1) + (C_x^2, C_{x2}^2)$;
    $\sigma_1 \leftarrow (C_{x2}^1/l_1 - C_x^1)/l_1$; `/* update */`
    $(i_1, j_1) \leftarrow (i_2, j_2)$; `/* step forward */`
    $d_1 \leftarrow d_2$;
    $(i_2, j_2) \leftarrow end\ of\ next\ segment$; `/* next segment */`
    $(C_x^2, C_{x2}^2) \leftarrow I_\Sigma(i_2, j_2)$;
    $d_2 \leftarrow I_\Theta(i_2, j_2)$;
    $\sigma_2 \leftarrow (C_{s2}^2/l - C_s^2)/l$;
  end
end
$\widehat{I}(i, j) \leftarrow C_x^1/l_1$; `/* isoline value */`

Algorithm 4: edge detector and pixel classifier `kernel_edge_detector()`

1: **foreach** *pixel* (i, j) **do** `/* in parallel */`
2:   $\Theta \leftarrow 0$; `/* direction index */`
3:   $edgeCount \leftarrow 0$;
4:   $sumEdge \leftarrow 0$;
5:   $nH \leftarrow 5l + 1$;
6:   $nL \leftarrow 3l$;
7:   **while** $(\Theta < 32)$ **do**
8:     $sumH \leftarrow (I_{ntex}(i, j), I_{ntex}^2(i, j))$;
9:     $sumL \leftarrow (0, 0)$;
10:     **for** $(\alpha = \Theta$ *to* $\alpha = \Theta + 16$ *by step* $4)$ **do**
11:       $sPat \leftarrow \sum_{(y,x)\in P_{l,\alpha}(i,j)} I_{ntex}(i + y, j + x)$;
12:       $sPat2 \leftarrow \sum_{(y,x)\in P_{l,\alpha}(i,j)} I_{ntex}^2(i + y, j + x)$;
13:       $sumH \leftarrow sumH + (sPat, sPat2)$;
14:     **end**
15:     **for** $(\alpha = \Theta + 20$ *to* $\alpha = \Theta + 28$ *by step* $4)$ **do**
16:       $sPat \leftarrow \sum_{(y,x)\in P_{l,\alpha}(i,j)} I_{ntex}(i + y, j + x)$;
17:       $sPat2 \leftarrow \sum_{(y,x)\in P_{l,\alpha}(i,j)} I_{ntex}^2(i + y, j + x)$;
18:       $sumL \leftarrow sumL + (sPat, sPat2)$;
19:     **end**
20:     **if** $(GLRT(sumH, nH, sumL, nL) > T2_{max})$ **then**
21:       $edgeCount \leftarrow edgeCount + 1$;
22:       $sumEdge \leftarrow sumH.x$;
23:     **end**
24:     $\Theta \leftarrow \Theta + 4$;
25:   **end**
  `/* outputs isoline value */`
26:   **if** $(edgeCount == 0)$ **then**
27:     $\widehat{I}(i, j) \leftarrow \frac{(sumH.x + sumL.x)}{nH + nL}$; `/* LSR */`
28:   **end**
29:   **if** $(edgeCount == 1)$ **then**
30:     $\widehat{I}(i, j) \leftarrow \frac{(sumEdge)}{nH}$
31:   **end**
32:   **if** $(edgeCount > 1)$ **then**
33:     $\widehat{I}(i, j) \leftarrow \widehat{I_{PIPD}}(i, j)$; `/* PI-PD */`
34:   **end**
35: **end**

defines both half-planes H and L (nH, nL). Then the loop starting at line 7 performs the GLRT for every considered direction index Θ. Values $sumH$ and $sumL$ are vectors of two parameters x and y, parameter x being the sum of gray-level values and y the sum of square gray-level values. Value $sumH$ is computed along the pixels of half-plane H and is obtained by loop at lines 10 to 14; Value $sumL$ is computed along the pixels of half-plane L and is obtained by loop at lines 15 to 19. Value $I_{ntex}(i, j)$ refers to the gray-level value at pixel (i,j) previously stored in texture memory. Eventually, the isoline level value is output at line 27, 30 or 33 depending on the situation (see VI-C for details about the decision process).

VIII. RESULTS

The proposed hybrid PI-PD model has been evaluated with the 512x512 pixel sample images used by [12] in order to make relevant comparisons with other filtering techniques. As we aim to address image processing in very noisy conditions (as in [15]), we focused on the noisiest versions, degraded by AWGN of standard deviation $\sigma = 25$.

Quality measurements of the denoised images in comparison with reference images have been obtained by the evaluation of:

a) Peak Signal to Noise Ratio (PSNR) that quantifies the mean square error between denoised and reference images: $MSE(I, \widehat{I})$. We used the following expression:

$$PSNR = 10.log_{10}\left(\frac{max(\widehat{I})}{MSE(I, \widehat{I})}\right)$$

PSNR values are given in dB and highest values mean best PSNR.

b) The Mean Structure Similarity Index (MSSIM, defined in [16]), which quantifies local similarities between denoised and reference images inside a sliding window.

MSSIM values belong to an interval $[0; 1]$; the closer to 1 the better.

PSNR is widely used to measure image quality but can be misleading when used by itself: as demonstrated in [16], the processing of noisy images can bring a high PSNR value but very bad visual quality. This is avoided by the use of the MSSIM index along with the PSNR value: when both of them show high values, the overall quality can be considered high.

Result figure 9 provides the PSNR and MSSIM of every image, denoised with three different filters: average 5x5, hybrid PI-PD and BM3D. The *noisy* column shows the values for each image before denoising. BM3D ([7]) is taken as a reference in terms of denoising quality, while the average filter is taken as a reference in terms of processing time. The window size of 5x5 pixels has been choosen to achieve PSNR values similar to those obtained by PI-PD.

BM3D code is run on a quad-core Xeon E31245 at 3.3GHz and 8GByte RAM under linux kernel 3.2 (64bits), while PI-PD as well as average filter codes is run on a Nvidia C2070 GPU hosted by a PC running linux kernel 2.6.18 (64bits). The average filter used is an efficient parallel GPU implementation that we developped. It is a generic and versatile separable convolution kernel that outputs more than 700MPixels per second in the 5x5 averaging configuration.

Hybrid PI-PD measurements were performed with $n = 25$, $l = 5$, $T_{max} = 1$ and $T2_{max} = 2$. BM3D measurements have been performed with the freely available BM3D software proposed in [7].

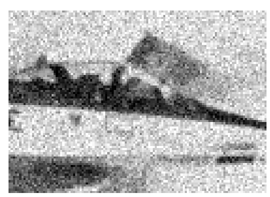
The hybrid PI-PD model proves much faster than BM3D and better than the average 5x5 filter. Processing the thirteen images of the database reveals that hybrid PI-PD brings an average improvement of 1.5dB (PSNR) and 7.2% (MSSIM) against the average filter at the cost of 35 times its computational duration. Against hybrid PI-PD, BM3D achieves an average improvement of 2.4dB and 4.6% at the cost of 350 times as much duration. Actually, the 5x5 average filter takes around **0.35 ms** to process an image while hybrid PI-PD needs around **11 ms** and BM3D around **4.3 s**.

It must be noticed that experimental optimization show that the vector of parameter values $T_{max} = 1$ and $T2_{max} = 2$ is optimal for 11 of the 13 images of the database. Better results are obtained with a slightly different value of $T2_{max}$ for *peppers* or *zelda* whose denoised images can obtain a MSSIM index of 0.90. Most of the computational time of hybrid PI-PD is spent by the edge detector, which clearly does not fit GPU requirements to achieve good performance. For information, the simple PI-PD model runs in less than 4 ms in the same conditions.

Figure 10 shows denoised images produced by hybrid PI-PD model compared with the output of the BM3D and the average 5x5 filters. The figure illustrates the merits and drawbacks of each model: edges are well preserved by hybrid PI-PD, but a *staircase* effect is visible, a well-known artefact inherent to this type of neighborhood filters. Our recent GPU-implementation of the regression method proposed in [17] brings a mean improvement of 1dB at the cost of 0.4 ms.

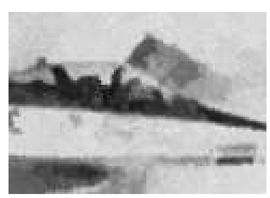
Image	Noisy	average 5x5	hybrid PI-PD	BM3D
airplane	19.49dB 0.58	26.39dB 0.84	28.46dB 0.88	30.88dB 0.93
barbara	20.04dB 0.70	22.76dB 0.76	24.26dB 0.83	30.60dB 0.94
boat	20.33dB 0.66	25.58dB 0.81	27.54dB 0.87	30.02dB 0.91
couple	20.28dB 0.69	25.25dB 0.79	27.33dB 0.87	29.77dB 0.91
elaine	19.85dB 0.59	28.71dB 0.86	28.94dB 0.87	30.60dB 0.91
fingerprint	20.34dB 0.93	23.33dB 0.87	26.07dB 0.95	27.93dB 0.96
goldhill	19.59dB 0.67	26.47dB 0.82	27.43dB 0.87	29.22dB 0.88
lena	19.92dB 0.60	27.99dB 0.84	29.14dB 0.88	31.80dB 0.93
man	20.38dB 0.71	24.74dB 0.80	26.74dB 0.86	28.14dB 0.87
mandrill	19.34dB 0.77	20.34dB 0.69	22.38dB 0.83	24.75dB 0.88
peppers	19.53dB 0.61	27.30dB 0.86	28.68dB 0.87	30.87dB 0.92
stream	20.35dB 0.80	23.23dB 0.78	25.35dB 0.87	26.34dB 0.88
zelda	17.71dB 0.58	23.13dB 0.87	27.71dB 0.88	30.49dB 0.93

Fig. 9. Comparison between hybrid PI-PD, average and BM3D filters. PI-PD parameter values: $n = 25$, $l = 5$, $T_{max} = 1$ and $T2_{max} = 2$. The *noisy* column correspond to the noisy input images, before denoising.
Timings: average filter in around 0.35 ms hybrid PI-PD in around 11.0 ms and BM3D in around 4.3 s

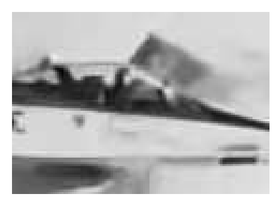


(a) Noisy image $\sigma = 25$

(b) Average 5x5 filter, in $0.35\ ms$



(c) PI-PD hybrid filter, $n = 25$, $l = 5$, $T_{max} = 1$, $T2_{max} = 2$, in $11\ ms$



(d) BM3D filter, in $4.3s$

Fig. 10. Comparison of 512x512 images denoised from noisy airplane image (10a) with a PI-PD filter (10b), PI-PD hybrid filter (10c) and BM3D filter (10d). Only zoomed parts of images are shown in order to ensure better viewing.

IX. Conclusion, future work

From the start, our approach, unlike quite a few others, has been to base this study on the conception and characteristics of the targeted hardware (Nvidia Graphic cards).

So as to get high execution speeds, we chose, for example, to find a method that remains local (concentrating on the immediate neighborhood of the center pixel), but still provides very significant benefits, using our technique of progressive lengthening.

Nevertheless, our method has proved slightly sub-optimal and lacking robustness in *flat* regions (see above, Low Slope Regions), even if the actual visual effect may be considered quite satisfactory.

As a first step to address the above drawbacks, we have devised a hybrid method that detects and applies distinct processing to LSR regions (see above). Processing speeds remain fast, and much higher than the BM3D implementation taken as quality reference. This is very promising, and opens the perspective of real-time high definition image sequence processing at 25 fps, provided we improve the edge detector, which currently limits the HD frame rate at 16fps (High Definition: 1920x1080 pixels).

To further improve the quality of output images, we also implemented a efficient parallel implementation of the staircase effect reduction technique presented in [17]. With this method, searching for best improvement factors leads to different parameters values for each image processed, which prompts to studying some way of overriding such parameters.

Our study so far has been based on additive noise; we are currently working on transposing criteria to various multiplicative noise types. We also extended the process to color images with very interesting visual results to be confirmed by the experimental measurement currently in progress.

References

[1] G. Matheron, *Random sets and integral geometry*. Wiley, 1975.

[2] M. McGuire, “A fast, small-radius gpu median filter,” in *ShaderX6*, February 2008. [Online]. Available: http://graphics.cs.williams.edu/papers/MedianShaderX6

[3] W. Chen, M. Beister, Y. Kyriakou, and M. Kachelries, “High performance median filtering using commodity graphics hardware,” in *Nuclear Science Symposium Conference Record (NSS/MIC), 2009 IEEE*, 24 2009-nov. 1 2009, pp. 4142–4147.

[4] R. M. Sanchez and P. A. Rodriguez, “Bidimensional median filter for parallel computing architectures,” in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, march 2012, pp. 1549–1552.

[5] Q. Yang, K.-H. Tan, and N. Ahuja, “Real-time o(1) bilateral filtering,” in *CVPR*. IEEE, 2009, pp. 557–564. [Online]. Available: http://doi.ieeecomputersociety.org/10.1109/CVPRW.2009.5206542

[6] B. Coll, J.-M. Morel, and A. Buades, “Non-local Means Denoising,” *Image Processing On Line*, 2011.

[7] K. Dabov, R. Foi, V. Katkovnik, and K. Egiazarian, “Bm3d image denoising with shape-adaptive principal component analysis,” in *Proc. Workshop on Signal Processing with Adaptive Sparse Structured Representations (SPARS’09*, 2009.

[8] F. Palhano Xavier De Fontes, G. Andrade Barroso, P. Coupé, and P. Hellier, “Real time ultrasound image denoising,” *Journal of Real-Time Image Processing*, May 2010. [Online]. Available: http://hal.inria.fr/inria-00476122

[9] V. Caselles, B. Coll, and J.-M. Morel, “Scale space versus topographic map for natural images.” Springer, 07 1997, pp. 29–49.

[10] P. Monasse and F. Guichard, “Scale-space from a level lines tree,” in *Scale-Space Theories in Computer Vision*, ser. Lecture Notes in Computer Science, M. Nielsen, P. Johansen, O. Olsen, and J. Weickert, Eds. Springer Berlin / Heidelberg, 1999, vol. 1682, pp. 175–186, 10.1007/3-540-48236-9_16. [Online]. Available: http://dx.doi.org/10.1007/3-540-48236-9_16

[11] N. Bertaux, Y. Frauel, P. Réfrégier, and B. Javidi, “Speckle removal using a maximum-likelihood technique with isoline gray-level regularization,” *J. Opt. Soc. Am. A*, vol. 21, no. 12, pp. 2283–2291, Dec 2004. [Online]. Available: http://josaa.osa.org/abstract.cfm?URI=josaa-21-12-2283

[12] *DenoiseLab Philosophy: A Standard Test Set and Evaluation Method to Compare Denoising Algorithms*, Oct. 2007.

[13] A. Buades, B. Coll, and J. Morel, “A review of image denoising algorithms, with a new one,” *Multiscale Modeling and Simulation*, vol. 4, no. 2, pp. 490–530, 2005.

[14] *NVIDIA CUDA C Programming Guide v3.1.1*, NVIDIA Corporation, 7 2010.

[15] G. Perrot, S. Domas, R. Couturier, and N. Bertaux, “Gpu implementation of a region based algorithm for large images segmentation,” in *Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on*, 31 2011-sept. 2 2011, pp. 291 –298.

[16] Z. Wang, A. C. Bovik, H. R. Sheikh, S. Member, E. P. Simoncelli, and S. Member, “Image quality assessment: From error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, pp. 600–612, 2004.

[17] A. Buades, B. Coll, and J.-M. Morel, “The staircasing effect in neighborhood filters and its solution,” *IEEE Transactions on Image Processing*, vol. 15, no. 6, pp. 1499–1505, 2006.