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

April 9, 2014

Abstract

In this study, we describe a GPU-based filter for image denoising, whose principle rests on Matheron's level sets theory first introduced in 1975 but rarely implemented because of its high computation cost. We use the fact that, within a natural image, significant contours of objects coincide with parts of the image level-lines. The presented algorithm assumes an *a priori* knowledge of the corrupting noise type and uses the polygonal level-line modeling constraint to estimate the gray-level of each pixel of the denoised image by local maximum likelihood optimization. Over the 512×512 pixel test images, the freely available implementation of the state-of-the-art BM3D algorithm achieves 9.56 dB and 36% of mean improvement in 4.3 s, respectively for peak signal to noise ratio (PSNR) and mean structural similarity index (MSSIM). Over the same images, our implementation features a high quality/runtime ratio, with a mean improvement of 7.14 dB and 30% in 0.09 s, which is 470 times as fast and potentially allows processing high-definition video images at 19 fps.

1 Introduction

For the past few years, because of a fast growth in digital devices able to take pictures or make movies, digital processing has become more and more important. Moreover, the wide range of applications requiring noise removal makes it difficult to design a universal filter and the increase in pixel density of the CCD or CMOS sensors leads to higher noise levels and requires high data rates in the processing algorithms. In addition, it is difficult to quantify the quality of an image processing algorithm, as visual perception varies significantly from one person to another.

To date, many researchers have successfully sped up image processing algorithms by implementing them on GPUs. For example Mc Guire [9], Chen *et al.* [4] and Sanchez [13] reported quite fast median filters. Sanchez's filter is able to output up to 300 million pixels per second, *i.e* 150 fps of high definition video images. Bilateral filtering has also been successfully proposed by Yang [17].

However, most high quality algorithms, such as NL-means [5] or BM3D [6] make use of non-local similarities and/or frequency domain transforms, so the speedups achieved by their current GPU implementations (see NL-means [11]) do not come near those achieved by local methods such as Gaussian, median or neighborhood filters.

Considering that a *natural* image, *i.e* any photograph of an outdoor or indoor scene taken by a standard camera, is of bounded variation and thus can be decomposed into a set of level-lines [3, 8], researchers have implemented algorithms that make use of level-lines properties, dedicated, in particular, to segmentation and classification [3, 7, 10, 14, 15].

Bertaux *et al.* described a method which significantly reduces speckle noise inside coherent images by using the properties of the level-lines in the image to constrain the minimization process. In [1], isolines (iso-gray-level lines) consist in neighborhoods of polyline shapes determined by maximum likelihood optimization.

This method proved that is was able not only to bring good enhancement but also to preserve edges between regions. Nevertheless, the costs in computation time, one minute to process a 2 Mpixel image on a PIII-1GHz CPU, do not allow real-time image processing.

We started by implementing Bertaux *et al.* algorithm on GPU and testing various optimization heuristics, in order to find out which tracks could be followed towards both minimizing loss in quality and preserving admissible execution times. Our tests were carried out with reference images taken from the denoiseLab site [1].

As will be detailed further down, statistical observations of the output images produced by the Bertaux *et al.* method enabled us to propose a very fast and simple parallel GPU-based method with good results in terms of PSNR and edge preservation.

In terms of processing speed, on the basis of the BM3D timings listed in [6] and with our own measurements, our GPU-based filter runs around 470 times faster and is thus able to process high definition images at 19 fps. It also achieves good denoising quality.

In the following, section 2 briefly focuses on recent GPU characteristics. Section 3 introduces the theory and notations used to define isolines. Then, in section 4, we describe the two isoline-based models that led to our final hybrid model, while section 5 details the parallel implementation of the proposed algorithm. Results are presented in section 6.

[1]http://www.stanford.edu/∼slansel/denoiseLab/ is apparently no longer accessible from the site.

2 GPU architecture

GPUs are multi-core, multi-threaded processors, optimized for highly parallel computation. Their design focuses on the *single instruction multiple threads* (SIMT) model that devotes more transistors to data processing rather than data caching and flow control. Three main manufacturers produce GPUs : Intel, Nvidia and ATI.

We only have had access to two Nvidia cards, respectively models C1060 and the more recent C2070, both designed for general purpose computing. The C2070 card features 6 GBytes global off-chip memory and a total of 448 cores (1.15 GHz) bundled in several *streaming multiprocessors* (SM). An amount of on-chip memory (much faster), is also available: thread registers (63 per thread) and shared memory (up to 48 KB per thread block). The memory bandwidth claimed by the manufacturer is 144 GB per second.

Moreover, Nvidia provides CUDA (for compute unified device architecture), which makes it quite easy to write parallel code for their target GPUs. However, writing efficient code is not trivial and requires to pay special attention to a number of points, among which:

1. CUDA organizes threads by a) thread blocks in which synchronization is possible, b) a grid of blocks with no possible synchronization between them.

2. The order in which threads are scheduled is not defined. Threads are run in parallel by groups called *warps*. One execution engine runs the odd-indexed threads of each warp and another runs the even-indexed threads.

3. Data must be kept in GPU memory, to reduce the overhead generated by copying between CPU and GPU.

4. The total amount of threads running the same computation must be as large as possible.

5. The number of execution branches inside one block should be as small as possible.

6. Global memory accesses have to be coalescent, *i.e.* memory accesses done by physically parallel threads (2 x 16 at a time) must be consecutive and contained in an 128 Byte aligned block.

7. Shared memory is organized in 32x32 bit-wide banks. A bank conflict occurs when two threads in a half-warp map to the same bank at different locations.

Among the various heuristics presented in the following sections, the PI-LD model implementation can hardly comply with the above constraint n°5 and does not allow high performances. That led us to design the presented hybrid PI-PD of section 4.3, which gets rid of this particular constraint. The use of shared memory is restricted and does not have any noticeable impact on the implementation's performances.

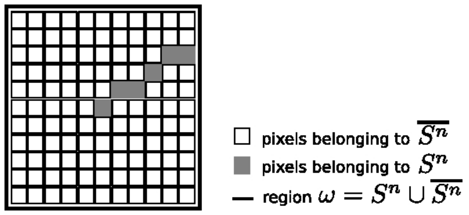
3 Isolines

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

Within the noisy image, our goal is to select, for each pixel, the set of neighboring pixels that best fits the underlying *level line* of the *reference image* I (in Matheron's meaning). From now on, we shall refer to such sets of pixels as *isolines*. An *isoline* can be either one single *isoline-segment* or one polyline composed by at least two connected *isoline-segments*. The generalized likelihood criterion (GL) is used to select the best isoline among all the considered ones. The output gray level value of the considered pixel is the average of all gray level values along the selected isoline.

3.1 Isoline-segments

For each pixel (i,j) of the corrupted image, we estimate the likelihood of each candidate isoline-segment inside a rectangular window ω centered on (i,j). Inside ω, let S^n be the isoline-segment; the center pixel belongs to it. S^n is a set of n pixel positions (i_q,j_q) $(q \in [0..n[)$.
The gray level z along S^n follows a gaussian probability density function whose parameters μ_{S^n} (mean value of isoline-segment) and σ (standard deviation brought by gaussian noise) are unknown.
Let $\overline{S^n}$ be defined by $\omega = S^n \cup \overline{S^n}$.
Figure 1 shows an example of such a ω region with its two separate sub-regions S^n and $\overline{S^n}$. The allowed patterns of the *isoline-segment* S^n, among all the possible ones in ω, are predefined and grouped into a look-up table p_{n-1} presented in figure 2. We defined $D = 32$ directions, each corresponding to one pattern. All of them have the same number of pixels, in order to be compared by maximum likelihood optimization. For each pixel, the mean values μ_{ij} of



□ pixels belonging to $\overline{S^n}$
■ pixels belonging to S^n
— region $\omega = S^n \cup \overline{S^n}$

Figure 1: Example of organization of the set of pixels to which maximum likelihood optimization is applied. The whole window ω is divided into 2 sets of pixels, candidate isoline-segment S^n and $\overline{S^n}$. All isoline-segments patterns share the same number of pixels and are predefined as shown in Figure 2

gray levels z over $\overline{S^n}$ are unknown and supposedly independent.
Let Z be the set of gray levels of pixels in ω and $\{\mu_{ij}\}_{\overline{S^n}}$ the mean values of pixels in $\overline{S^n}$. With the above assumptions, the likelihood of Z 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 re-written 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:

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

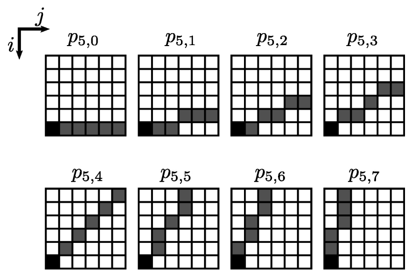
inside which the vector of parameters (μ_{S^n},σ) is determined by maximum likelihood estimation

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

The selected isoline-segment is the one which maximizes equation (5).

3.2 Extendable isolines

Searching for longer 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 512×512 pixel image. Therefore, we chose to build large isolines inside an iterative process including a mandatory validation stage between each extension 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.



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

Figure 2: **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.

Let S^n be a previously selected isoline part (composed of isoline-segments) and S^p connected to S^n in such a way that S^p could be an extension to S^n so as to define an isoline S^{n+p}. Figure 3 illustrates this situation with parts of an image on the left side, showing a supposedly validated isoline-segment S^n and two possible candidates for its extension: $S^{p'}$ and $S^{p''}$. The right side of the figure represents both situations in another system of reference coordinates, with the gray level value on the ordinate axis and the linear pixel index along the abscissa. In this example, one can imagine that $S^{p''}$ may represent a valid extension to S^n.

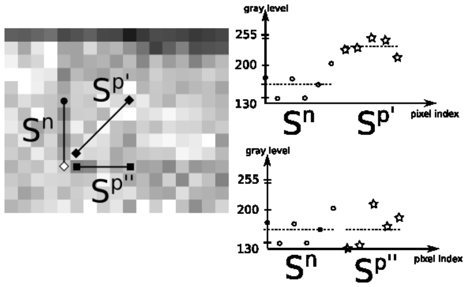


Figure 3: Extension of an isoline: S^n is supposed to be a valid n-pixel isoline, while $S^{p'}$ and $S^{p''}$ are two sample candidates for S^n extension by p pixels length. This extension of S^n is submitted to the GLRT condition (see eq. (8)). Left: zoom on a small window of the airplane image, featuring S^n, $S^{p'}$ and $S^{p''}$. Right: gray levels of all pixels *vs* the 1-D pixel index. Dotted lines represent the average values of the gray levels of S^n, $S^{p'}$ and $S^{p''}$. In this particular example, with a $T_{max} = 2$ (which is often near the optimal value in our benchmarks), $S^{p'}$ would not be accepted as an extension of S^n, while $S^{p''}$ would be.

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

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

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

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

Alternately, if S^n and S^p are two separate isolines 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

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

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 following expression of $GLRT(S^{n+p}, S^n, S^p, T_{max})$:

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

The decision to validate the extension from S^n to S^{n+p} depends on whether $GLRT(S^{n+p}, S^n, S^p, T_{max})$ is superior or inferior to 0. Value T_{max} is the GLRT threshold.

4 Isoline models

We construct *isolines* as polylines. Each isoline can then be curved by allowing a direction change at the end of each *isoline-segment*.

In order to keep the number of candidate isolines within reasonable range, we chose to build them by combining segments described by simple precomputed 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 2 shows an example of such a pattern matrix for $l = 5$.

4.1 Isolines with limited deviation angle (PI-LD)

Due to the amount of memory involved and because the necessary reduction stage involves variable size vectors, parsing the tree of all possible polyline configurations proved far too slow regarding our goal. So, we focused on a variant inspired by Bertaux *et al.* [1] in which the selected direction of the next segment depends on the whole of the previously built and validated isoline.

Let us consider an 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). During the previous extension steps, both of the following sums have been obtained:

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

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

Let us now examine how to decide whether to add a new segment to S^n or to stop the extension 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 direction change Δd_{max}. The above condition is meant to prevent the artifacts generated by:

- isolines rolling onto themselves.
- isolines being backward-oriented.
- isolines segments crossing each other.

Another of its benefits is to reduce the number of combinations to be evaluated.

For each pattern, we use GLRT to decide if the segment can extend isoline S^n. If no segment satisfies the test, S^n is terminated.

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

Figure 4 illustrates one stage of the extension process with the example of a two-segment isoline at the beginning of stage ($l = 5$ and $\Delta d_{max} = 2$).

We simultaneously extend D isolines, each of them having the same primary direction as pattern $p_{l,d}$ ($d \in [0..D]$). Eventually, the isoline with the maximum likelihood value is selected among the longest ones. That prevents the selected isoline from being critically sub-optimal, which could be the case if the first selected isoline-segment did not follow the primary direction of the optimal isoline, in which case the limiting deviation angle would forbid the isoline under construction to come near the optimal one.

Though much faster, the PI-LD-based filter may still be considered inferior to *state-of-the-art* filters such as the BM3D family of algorithms [6]. Furthermore, this way of building isolines requires the alternate use of two different types of validation at each extension stage: GLRT and maximum likelihood minimization, each of them generating numerous divergent branches, sequentially run by each thread warp and leading to unwanted overhead. This led us to propose the optimization heuristics detailed in the following section.

4.2 Isolines with precomputed directions (PI-PD)

Within the PI-LD model, at each starting pixel (i, j), D isolines are computed and kept as candidates though only one follows the actual isoline at (i, j).

It needs to evaluate $(2.\Delta_{dmax} + 1)$ segments at each extension stage (Δ_{dmax} segments on both sides of the previous direction angle). Allowing a maximum of K extension stages and mandatorily evaluating the first D directions lead to the evaluation of $D.\left(2.\Delta_{dmax} + 1\right)^{K-1}$ segments at each pixel position.

If we assume we can achieve a robust determination of the direction of this isoline at (i, j), it becomes unnecessary to perform the selection at each extension step. Thus, at each pixel (i, j), only the direction of the first segment has to be determined in order to obtain the local direction of the isoline. This drastically reduces work complexity as only $D.K$ evaluations are needed.

For example, with a maximum of $K = 5$ segments and a maximum direction change of $\Delta_{dmax} = 2$, the PI-LD and the above equations lead to a count of up to $20,000$ 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 independent extension process. For one given state of an 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.

It remains necessary to prevent isolines from generating artifacts much in the same way as in PI-LD. Figure 5 details this process, starting from the same initial state as in figure 4 with the noticeable difference that no direction change limit is needed.

This optimization heuristics avoids multiple divergent branches during kernel execution, which better fits GPU constraints. It remains, however, that the building of isolines is done without global likelihood optimization.

In addition to the above extension process, the model has been further improved by adding the ability to thicken isolines from one to three pixels, allowing higher PSNR values. This may preferably apply to large images that do not contain small relevant details, liable to be slightly blurred. Still, this feature makes PI-PD more versatile than our reference BM3D, which has prohibitive computation times when processing such large images (over 5 minutes for a 4096x4096 pixel image).

4.3 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 such regions Low Slope Regions (LSR). Figure 7 shows that lack of robustness with an example of two drawings of additive white gaussian noise applied to our reference image (Figure 6). We focused on a small 11×11 pixel window containing two LSRs with one sharp edge between them.

Figures 7d and 7e 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 4 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 use GLRT correctly and without adding too many computations, we only involve patterns that do not share any pixel between them. They are those whose direction is a multiple of 45°.

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 bottom regions, denoted T and B). We assume that segments on the limit belong to the T region which includes pixels of direction 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 8 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 taller bars define the T region and those represented by shorter bars define the B region.

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

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

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 8).

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 9 shows an example of such classification achieved by the edge detector. The detector has been applied to the noisy airplane image with GLRT threshold value $T2_{max} = 2$. Darker dots belong to an edge, whiter dots belong to a LSR.

5 Hybrid PI-PD filter Implementation: details

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

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

5.2 Generation of reference matrices I_Σ and I_Θ

In order to generate both matrices, a GPU kernel named `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 of a block. That led us to load it from texture memory first, then copy it into all shared memory blocks, which 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, that are eventually stored in matrices I_Θ and I_Σ on lines 16 and 17.

Algorithm 1: Initializations in GPU memory

```
1 $l \leftarrow$ step size;
2 $D \leftarrow$ number of primary directions;
3 $I_n \leftarrow$ noisy image;
4 $I_{ntex} \leftarrow I_n$;      /* copy to texture mem.  */
5 $P_l \leftarrow$ kernel_genPaths ; /* pattern matrix */
6 $P_{ltex} \leftarrow P_l$;      /* copy to texture mem.  */
7 $T_{max} \leftarrow$ GLRT threshold (extension);
8 $T2_{max} \leftarrow$ GLRT threshold (edge detection);
```

5.3 PI-PD extension 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 4.2 for process description).

Lines from 2 to 11 perform the allocations needed by the evaluation of the first extension. 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 isoline; C_x^2 and C_{x2}^2 are the gray-level sums of the candidate segment. The current 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 extension has been accepted, the length of the 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 definitions). Finally, using direction value d_2, it translates the coordinates (i_1, j_1) to the end of the newly extended isoline and coordinates (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 isoline.

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
```

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

```
1  foreach pixel $(i,j)$ do                      /* in parallel */
2      $(C_x^1, C_{x2}^1) \leftarrow z(i,j)$ ;   /* starting pixel */
3      $(i_1, j_1) \leftarrow (i,j)$ ;           /* first segment */
4      $(C_x^1, C_{x2}^1) \leftarrow I_\Sigma(i_1, j_1)$ ;   /* read matrix */
5      $d_1 \leftarrow I_\Theta(i,j)$ ;          /* read matrix */
6      $l_1 \leftarrow l$ ;                      /* isoline length */
7      $\sigma_1 \leftarrow (C_{x2}^1/l_1 - C_x^1)/l_1$;
8      $(i_2, j_2) \leftarrow$ end of first segment;
9      $(C_x^2, C_{x2}^2) \leftarrow I_\Sigma(i_2, j_2)$ ;   /* 2nd segment */
10     $d_2 \leftarrow I_\Theta(i_2, j_2)$;
11     $\sigma_2 \leftarrow (C_{x2}^2/l - C_x^2)/l$ ;
12     while $GLRT(\sigma_1, \sigma_2, l_1, l) < T_{max}$ do
13         $l_1 \leftarrow l_1 + l$ ;            /* extension */
14         $(C_x^1, C_{x2}^1) \leftarrow (C_x^1, C_{x2}^1) + (C_x^2, C_{x2}^2)$;
15         $\sigma_1 \leftarrow (C_{x2}^1/l_1 - C_x^1)/l_1$ ;   /* update */
16         $(i_1, j_1) \leftarrow (i_2, j_2)$ ;  /* step forward */
17         $d_1 \leftarrow d_2$;
18         $(i_2, j_2) \leftarrow$ end of next segment;
                                                 /* next segment */
           $(C_x^2, C_{x2}^2) \leftarrow I_\Sigma(i_2, j_2)$;
19         $d_2 \leftarrow I_\Theta(i_2, j_2)$;
20         $\sigma_2 \leftarrow (C_{s2}^2/l - C_s^2)/l$ ;
21     end
22 end
23 $\widehat{I}(i,j) \leftarrow C_x^1/l_1$ ;     /* isoline value */
```

5.4 Hybrid PI-PD : `kernel_edge_detector()`

As introduced in section 4.3, the aim of the kernel named `kernel_edge_detector()` is to divide pixels into two classes according to their belonging to a LSR or not. Algorithm 4 explains the detailed 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 defines both half-planes H and L (nH, nL). Then the loop starting at line 7 uses 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 the loop at lines 10 to 14. Value *sumL* is computed along the pixels of half-plane L and is obtained by the 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 4.3 for details about the decision process).

6 Results

The proposed hybrid PI-PD model has been evaluated with the 512x512 pixel sample images of the S. Lansel denoiseLab, in order to make relevant comparisons with other filtering techniques. As we aim to address image processing in very noisy conditions (as in [12]), 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 decibels (dB) and the higher values mean better 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 as only quality assessment: as demonstrated in [16], a high PSNR value does not necessarily mean good visual quality. This can be avoided by using the MSSIM index along with the PSNR value: when both of them show high values, the overall visual quality can be considered high.

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

```
foreach pixel $(i, j)$ do                                  /* in parallel */
    $\Theta \leftarrow 0$;                                 /* direction index */
    $edgeCount \leftarrow 0$;
    $sumEdge \leftarrow 0$;
    $nH \leftarrow 5l + 1$;
    $nL \leftarrow 3l$;
    while ($\Theta < 32$) do
        $sumH \leftarrow (I_{ntex}(i, j), I^2_{ntex}(i, j))$;
        $sumL \leftarrow (0, 0)$;
        for ($\alpha = \Theta$ to $\alpha = \Theta + 16$ by step 4) do
            $sPat \leftarrow \sum_{(y,x) \in P_{l,\alpha}(i,j)} I_{ntex}(i+y, j+x)$;
            $sPat2 \leftarrow \sum_{(y,x) \in P_{l,\alpha}(i,j)} I^2_{ntex}(i+y, j+x)$;
            $sumH \leftarrow sumH + (sPat, sPat2)$;
        end
        for ($\alpha = \Theta + 20$ to $\alpha = \Theta + 28$ by step 4) do
            $sPat \leftarrow \sum_{(y,x) \in P_{l,\alpha}(i,j)} I_{ntex}(i+y, j+x)$;
            $sPat2 \leftarrow \sum_{(y,x) \in P_{l,\alpha}(i,j)} I^2_{ntex}(i+y, j+x)$;
            $sumL \leftarrow sumL + (sPat, sPat2)$;
        end
        if ($GLRT(sumH, nH, sumL, nL) > T2_{max}$) then
            $edgeCount \leftarrow edgeCount + 1$;
            $sumEdge \leftarrow sumH.x$;
        end
        $\Theta \leftarrow \Theta + 4$;
    end
    /* outputs isoline value */
    if ($edgeCount == 0$) then
        $\widehat{I}(i, j) \leftarrow \frac{(sumH.x + sumL.x)}{nH + nL}$;   /* LSR */
    end
    if ($edgeCount == 1$) then
        $\widehat{I}(i, j) \leftarrow \frac{(sumEdge)}{nH}$
    end
    if ($edgeCount > 1$) then
        $\widehat{I}(i, j) \leftarrow \widehat{I_{PIPD}}(i, j)$;   /* PI-PD */
    end
end
```

The BM3D code is run on a quad-core Xeon E31245 at 3.3GHz and 8GByte RAM under linux kernel 3.2 (64bits), while both hybrid PI-PD and average filter codes are run on a Nvidia C2070 GPU hosted by a PC running linux kernel 2.6.18 (64bits). The average filter we used is our own implementation of a generic and versatile convolution kernel, able to output over $1,200$ million pixels per second in the 5×5 averaging configuration.

Result figure 10 provides the PSNR improvement and MSSIM values of reference images, denoised with average 5×5, hybrid PI-PD and BM3D filters. The *noisy* column shows values prior to denoising. BM3D ([6]) 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 5×5 pixels has been chosen to achieve PSNR values similar to those obtained by PI-PD. The 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 [6].

The hybrid PI-PD model proves much faster than BM3D and better than the average 5×5 filter. Processing the thirteen images of the database reveals that hybrid PI-PD brings an average improvement of 1.52 dB (PSNR) and 7.26 % (MSSIM) against the average filter at the cost of 128 times its computational duration. Against the hybrid PI-PD, BM3D achieves an average improvement of 2.41 dB and 4.64 % at the cost of 475 times as much duration. The 5×5 average filter takes **0.07 ms** to process an image while the hybrid PI-PD needs **9 ms** and BM3D **4.3 s**. Data transfers from CPU to GPU texture memory and from GPU global memory to CPU pinned memory need 0.15 ms additional time. The use of pinned memory saves 0.09 ms for each 8 bit-coded 512×512 pixel image.

It must also be noticed that measurements 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 the 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 11 shows denoised images produced by the hybrid PI-PD model compared with the output of the BM3D and the average 5×5 filters. It illustrates the merits and drawbacks of each model: edges are well preserved by the hybrid PI-PD, but a *staircase* effect is visible, a well-known artifact inherent to this type of neighborhood filters. Our GPU-implementation of the regression method proposed in [2] brings interesting quality improvement over processing time with an average of 1.00 dB at the cost of 0.2 ms.

7 Conclusion

Generally speaking, our chosen approach has been to focus on what a GPU can do fast to design efficient and robust elementary kernels that can be re-used in complex programs that achieve signal processing functions, without necessarily trying to parallelize any existing CPU algorithms or designing CPU versions to validate results.

Instead of only searching to obtain speedups from existing CPU algorithms, we have tried to design a specific GPU-based high-speed denoising filter. For that purpose, we chose to design a method we called *hybrid PI-PD* that both remains local and provides very significant benefits, thanks to our technique of progressive isoline extension.

Processing speeds are much higher than the BM3D implementation taken as quality reference. This is very promising and currently allows real-time high definition image sequence processing at 19 fps (High Definition: 1920x1080 pixels). We also implemented a parallel version of the staircase effect reduction technique presented by Buades *et al.* [2] which further improves the quality of output images without significant performance loss.

Though our research has been so far focused on additive white Gaussian noise, we are currently transposing the criterion to various multiplicative noise types. We also extended the process to color images with very interesting visual results to be confirmed by the experimental measurements currently in progress.

References

[1] Bertaux N, Frauel Y, Réfrégier P, Javidi B (2004) Speckle removal using a maximum-likelihood technique with isoline gray-level regularization. J Opt Soc Am A 21(12):2283–2291, DOI 10.1364/JOSAA.21.002283, URL http://josaa.osa.org/abstract.cfm?URI=josaa-21-12-2283

[2] Buades A, Coll B, Morel JM (2006) The staircasing effect in neighborhood filters and its solution. IEEE Transactions on Image Processing 15(6):1499–1505

[3] Caselles V, Coll B, Morel JM (1997) Scale space versus topographic map for natural images. Springer, pp 29–49

[4] Chen W, Beister M, Kyriakou Y, Kachelries M (2009) High performance median filtering using commodity graphics hardware. In: Nuclear Science Symposium Conference Record (NSS/MIC), 2009 IEEE, pp 4142–4147, DOI 10.1109/NSSMIC.2009.5402323

[5] Coll B, Morel JM, Buades A (2011) Non-local means denoising. Image Processing On Line

[6] Dabov K, Foi R, Katkovnik V, Egiazarian K (2009) Bm3d image denoising with shape-adaptive principal component analysis. In: Proc. Workshop on Signal Processing with Adaptive Sparse Structured Representations (SPARS'09

[7] Froment J (1999) A compact and multiscale image model based on level sets. In: Proceedings of the Second International Conference on Scale-Space Theories in Computer Vision, Springer-Verlag, London, UK, UK, SCALE-SPACE '99, pp 152–163, URL http://dl.acm.org/citation.cfm?id=647082.715239

[8] Matheron G (1975) Random sets and integral geometry. Wiley

[9] McGuire M (2008) A fast, small-radius gpu median filter. In: ShaderX6, URL

http://graphics.cs.williams.edu/papers/MedianShaderX6

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

[11] Palhano Xavier De Fontes F, Andrade Barroso G, Coupé P, Hellier P (2010) Real time ultrasound image denoising. Journal of Real-Time Image Processing DOI 10.1007/s11554-010-0158-5, URL http://hal.inria.fr/inria-00476122

[12] Perrot G, Domas S, Couturier R, Bertaux N (2011) Gpu implementation of a region based algorithm for large images segmentation. In: Computer and Information Technology (CIT), 2011 IEEE 11th International Conference on, pp 291 –298, DOI 10.1109/CIT.2011.60

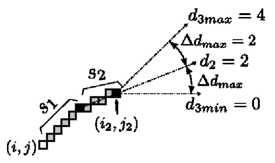
[13] Sanchez RM, Rodriguez PA (2012) Bidimensional median filter for parallel computing architectures. In: Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on, pp 1549–1552, DOI 10.1109/ICASSP.2012.6288187

[14] Varshney KR, Willsky AS (2010) Classification using geometric level sets. J Mach Learn Res 11:491–516, URL http://dl.acm.org/citation.cfm?id=1756006.1756020

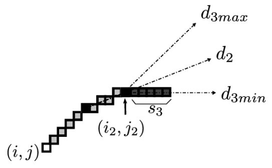
[15] Vese LA, Chan TF, Tony, Chan F (2002) A multiphase level set framework for image segmentation using the mumford and shah model. International Journal of Computer Vision 50:271–293

[16] Wang Z, Bovik AC, Sheikh HR, Member S, Simoncelli EP, Member S (2004) Image quality assessment: From error visibility to structural similarity. IEEE Transactions on Image Processing 13:600–612

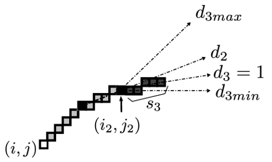
[17] Yang Q, Tan KH, Ahuja N (2009) Real-time o(1) bilateral filtering. In: CVPR, IEEE, pp 557–564, URL http://doi.ieeecomputersociety.org/10.1109/CVPRW.2009.5206542



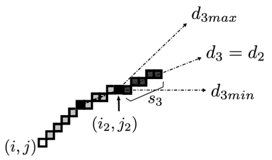
(a) Isoline with two validated segments s_1 and s_2.



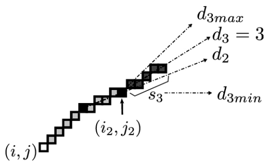
(b) First evaluated segment, corresponding to pattern $p_{5,0}$.



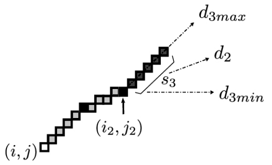
(c) Second evaluated segment, corresponding to pattern $p_{5,1}$.



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

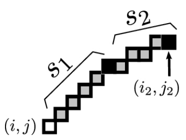


(e) Fourth evaluated segment, corresponding to pattern $p_{5,3}$.

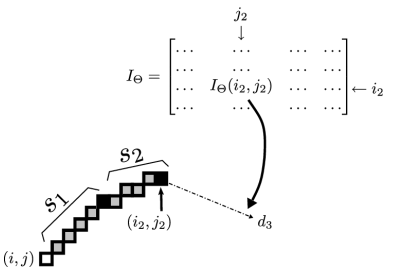


(f) Fifth evaluated segment, corresponding to pattern $p_{5,4}$.

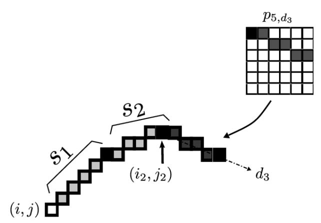
Figure 4: Example of an extension process starting with a two-segment isoline ($l = 5$, $\Delta d_{max} = 2$). The initial situation is shown in 4a, while 4b to 4f represent the successive candidate segments. The direction index of the second validated segment is $d_2 = 2$ (4a). This implies that direction indices d_3 allowed for the third segment range from $d_2 - \Delta dmax = 0$ to $d_2 + \Delta dmax = 4$ (4b to 4f). The extension of the 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 gray levels of the whole isoline is selected.



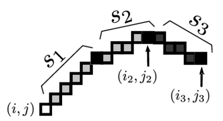
(a) Isoline with two validated segments.



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

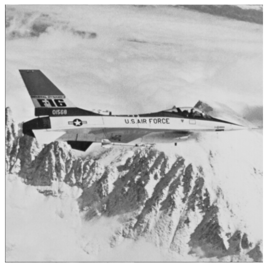


(c) Pattern p_{l,d_3} is then applied at (i_2, j_2) and GLRT is used. Both sums needed to use GLRT are read from element (i_2, j_2) of I_Σ.



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

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


(a) Reference noiseless airplane image

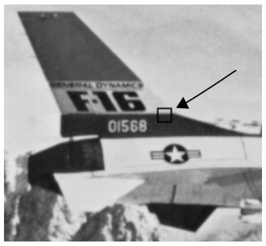
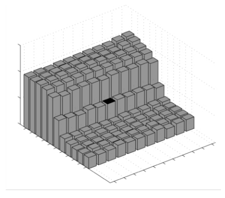
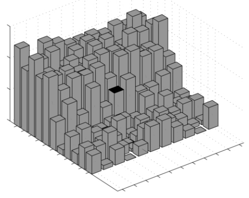
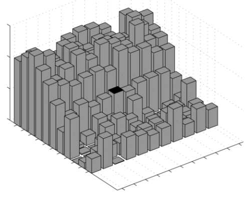
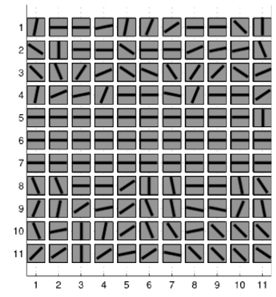

(b) Location of the example window in the reference image.

Figure 6: Location of the example window inside the reference image. Figure 6a shows the whole reference image and 6b zooms on the part which the example 11×11 pixel window is taken from.

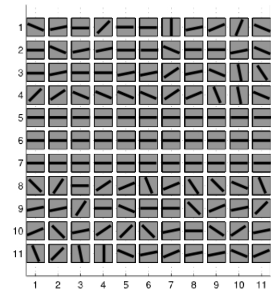

(a) Reference image


(b) Image corrupted by random drawing $n^{\circ}1$


(c) Image corrupted by random drawing $n^{\circ}2$



(d) Isoline directions for random drawing $n^{\circ}1$



(e) Isoline directions for random drawing $n^{\circ}2$

Figure 7: Zoom on a small square window of the airplane image. 7a reproduces the zoom on the window, taken from the reference image of Figure 6. 7b, 7c and 7a are 3D views where each bar represents a pixel whose gray-level corresponds to the height of the bar. Figures 7d and 7e are 2D top views of the window. The chosen window shows an edge between two LSRs. Images 7b and 7c are corrupted with two different random drawings of the same additive white gaussian noise (AWGN) of power σ^2 and mean value 0. 7d and 7e show the direction of the isoline found by PI-PD, for each pixel of the window. In low slope regions (top and bottom regions), the determination of the direction is not robust. Near the edge, directions do not vary from one drawing to another.

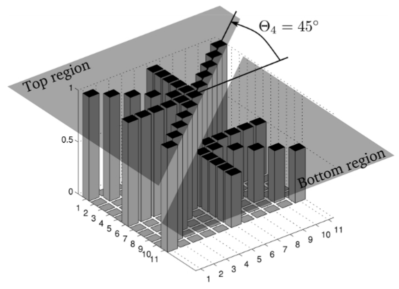
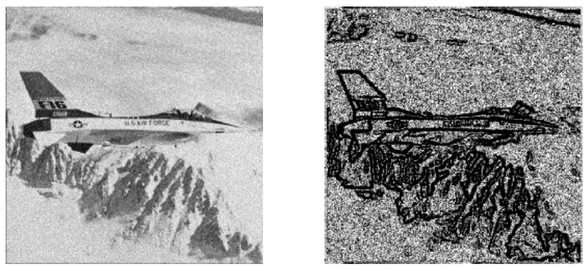


Figure 8: Edge detector. 3D view representing an example square 11x11 pixel window ($l = 5$) used in the edge detector for $\Theta_4 = 45°$. Each pixel is represented by a bar. Bars of 0 height value are for pixels that are not involved in the detector. The top region is defined by five pattern segments and includes the center pixel. The bottom 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. The edge detector uses one GLRT for each value of Θ_i, from $O°$ to $315°$ by step of $45°$.



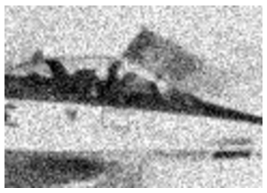
(a) Noisy airplane image

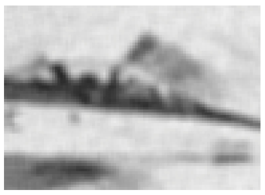
(b) Pixel classification performed by the edge detector.

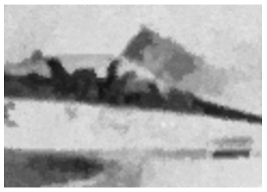
Figure 9: Pixel classification inside the noisy image. Figure 9a shows the noisy input image and 9b reproduces the output classification of pixels, as a black and white image, obtained with threshold value $T2_{max} = 2$. Black pixels are supposed to be near an edge, while white pixels belong to Low Slope Regions.

Image	**Noisy**	**average** 5×5	**hybrid PI-PD**	**BM3D**
	PSNR (dB)	gain (dB)	–	–
	MSSIM	MSSIM	–	–
airplane	19.49	6.90	8.97	11.39
	0.58	0.84	0.88	0.93
barbara	20.04	2.72	4.22	10.56
	0.70	0.76	0.83	0.94
boat	20.33	5.25	7.21	9.69
	0.66	0.81	0.87	0.91
couple	20.28	4.97	7.05	9.49
	0.69	0.79	0.87	0.91
elaine	19.85	8.86	9.09	10.75
	0.59	0.86	0.87	0.91
fingerprint	20.34	2.99	5.73	7.59
	0.93	0.87	0.95	0.96
goldhill	19.59	6.88	7.84	9.63
	0.67	0.82	0.87	0.88
lena	19.92	8.07	9.22	11.88
	0.60	0.84	0.88	0.93
man	20.38	4.36	6.36	7.76
	0.71	0.80	0.86	0.87
mandrill	19.34	1.00	3.04	5.41
	0.77	0.69	0.83	0.88
peppers	19.53	7.77	9.15	11.34
	0.61	0.86	0.87	0.92
stream	20.35	2.88	5.00	5.99
	0.80	0.78	0.87	0.88
zelda	17.71	10.42	10.00	12.78
	0.58	0.87	0.88	0.93

Figure 10: Comparison between hybrid PI-PD, average and BM3D filters. The PSNR values represent the improvement brought to the noisy image by the filters, the MSSIM values are absolute values. 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 0.07 ms hybrid PI-PD in 9.0 ms and BM3D in 4.3 s. Data transfers between CPU and GPU need 0.15 ms additional time.


(a) Noisy image $\sigma = 25$


(b) Average 5×5 filter, in 0.07 ms


(c) PI-PD hybrid filter, $n = 25$, $l = 5$, $T_{max} = 1$, $T2_{max} = 2$, in 9 ms

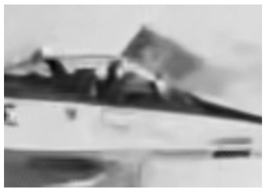

(d) BM3D filter, in 4.3s

Figure 11: Comparison of 512x512 images denoised from noisy airplane image (11a) with a PI-PD filter (11b), PI-PD hybrid filter (11c) and BM3D filter (11d). Only zoomed parts of images are shown in order to ensure better viewing.