Published in Image Processing On Line on 2013–07–11.
Submitted on 2012–12–11, accepted on 2013–03–01.
ISSN 2105–1232 © 2013 IPOL & the authors CC–BY–NC–SA
This article is available online with supplementary materials,
software, datasets and online demo at
https://doi.org/10.5201/ipol.2013.53

A Finite Difference Scheme for the Stack Filter Simulating the MCM

Marco Mondelli[1]

[1] École Polytechnique Fédérale de Lausanne, Switzerland (marco.mondelli@epfl.ch)

Communicated by Pascal Monasse *Demo edited by* Pascal Monasse

Abstract

The paper presents an algorithm that applies a stack filter simulating the Mean Curvature Motion equation via a finite difference scheme.

Source Code

An ANSI C implementation is provided and available from the web page of this article[1]. The code is distributed under GPL license.

Supplementary Material

An online demo of the algorithms is available. The demo allows uploading an image (or using the default one), running the Stack Filter for the MCM and original FDS on the whole picture or on a zoomed detail, and visualizing and comparing the results.

Keywords: finite difference scheme, stack filter, mean curvature motion, smoothing, scale space

1 Overview

Wertheimer [1] noticed that the local values of gray levels in an image might not be relevant information for the human visual system. Our perception of pictures is to some degree insensitive to contrast changes even when they are flat on an interval, which implies that some information gets lost as several gray levels melt together. In other words, if we want image analysis to be coherent with our sensory perception, it must achieve in a certain measure *contrast invariance*.

Let $u : \mathbb{R}^n \to \mathbb{R}$ be a continuous function ($u \in C^0$) and $g : \mathbb{R} \to \mathbb{R}$ be a continuous *contrast change*, i.e. a surjective and nondecreasing function. A function operator $T : C^0 \to C^0$ is said to be *contrast invariant*, if

$$T(g(u)) = g(T(u)). \tag{1}$$

[1] https://doi.org/10.5201/ipol.2013.53

Furthermore, T is said to be monotone, if for $u, v \in C^0$

$$u(x) \geq v(x) \Rightarrow T(u(x)) \geq T(v(x)) \qquad \forall\, x \in \mathbb{R}^n. \tag{2}$$

It has been shown by Guichard et al. [6] that all the contrast-invariant monotone function operators can be expressed as *stack filters*, a particular class of filters introduced by Wendt et al. [2]. Motivated by the ease of implementation and remarkable denoising qualities of the median filter, they decomposed the M-valued input signal into the set of its $M-1$ cross sections obtained thresholding the initial function at value $k = 1, 2, \ldots, M-1$. Then, they defined *stack filters* applying to these signals a translation invariant, increasing, binary operator and finally reconstructing the output via the superposition principle, i.e. "superposing" its evolved level sets. A characterization by means of positive Boolean Functions was provided and the convergence behavior analyzed. The authors determined the edge preserving characteristics of stack filters, finding conditions under which monotone regions are left untouched.

Maragos and Schafer [3] established the functional notation and set the link between stack filters and the framework of mathematical morphology introduced by Serra [4]. In particular they showed that stack filters represent the class of all discrete *translation invariant* (*TI*), *function-and-set-processing* (*FSP*) filters that can be expressed as a *finite maximum of local minima* or as a *minimum of local maxima* and that *commute with thresholds* (an operator satisfies the *commutation with threshold* property if to apply the operator and then consider the generic level set is the same as considering the level set and then applying the operator). Thus, discrete erosions, dilations, closings, openings, median and order-statistic (OS) filters [4] that commute with thresholds are all special cases of stack filters.

In particular, the aim of this work lies in the implementation of a stack filter based on a finite difference scheme (FDS) for the numerical simulation of the Mean Curvature Motion (MCM) equation [5],

$$\frac{\partial u}{\partial t} = \mathrm{curv}(u)|Du|, \tag{3}$$

where Du stands for the gradient of the function u and $\mathrm{curv}(u)$ for its scalar curvature, that can be evaluated as

$$\mathrm{curv}(u) = \frac{1}{|Du|^3} D^2u(Du^\perp, Du^\perp), \tag{4}$$

in which $Du^\perp$ denotes the vector orthogonal to Du and with its same norm.

The basic algorithm applied to the cross sections of the input image was introduced by Alvarez and Morel [7] and has been implemented in C language by Ciomaga and Mondelli [8].

The proposed scheme is a compromise between the simple FDS described by Ciomaga et al. [8, section *Algorithms*] and the Level Lines Shortening (LLS), a fully contrast invariant procedure simulating a subpixel evolution of an image by affine curvature motion [10, 11]. On the one hand, the Level Lines Shortening algorithm is highly accurate, but much more complex; on the other hand, FDSes are not monotone, contrast invariant, and affine invariant. So, a stack filter based on FDSes keeps the relative easiness of finite difference schemes while achieving a higher degree of contrast invariance. In addition, the common median filter, which is also a stack filter, is not to be preferred to the proposed scheme, as it lacks all the isotropic invariances that characterize FDSes for curvature motions.

In the following, we briefly outline the structure of the algorithm, pointing out that it is by definition *stable* and satisfies the *semigroup property* with good approximation. Then, we discuss the choice of the parameters and present some experimental results that focus on the behavior of simple *linear* and *radial functions*, and on the degree of *contrast invariance* and *Euclidean invariance* achieved by the stack filter and by the original FDS. Since in literature [2] stack filters are well

known for their denoising properties, a comparison is made between the proposed algorithm and the scheme [8] as concerns the reconstruction of an image corrupted by uniformly distributed noise. The time step which guarantees the best possible trade off between computational time and performances is chosen and two techniques to gain on CPU time are analyzed. An ANSI C implementation of the algorithm is provided. Lastly, we present a rich gallery of examples taken from geometry and everyday life, comparing the behavior of the proposed stack filter with the original FDS and the LLS algorithm.

2 Description of the Algorithm

We consider a discrete image $u(i,j)$ as an array of $n_{row} \times n_{col}$ integer samples taken from the continuous function u. The index i represents the current row number and j the current column number. The program requires three arguments, i.e.

- the desired normalized scale R;
- an input image to be processed;
- the name of the output image to be created.

At normalized scale R every disk with radius less or equal than R disappears. Solving the MCM equation for the particular case of a circumference, it is possible to show that the number of iterations n_{iter} is linked to the time step Δt and to the normalized scale R by this formula,

$$n_{\text{iter}} = \frac{R^2}{2\Delta t}. \tag{5}$$

The proposed algorithm acts on gray-level images, and if an image has multiple color channels, they are dealt with independently.

The proposed finite difference scheme for the numerical simulation of the MCM takes advantage of the diffusive interpretation of the equation, i.e. the mean curvature can be expressed as the second derivative of the input function u in the direction ξ orthogonal to the gradient,

$$u_t = u_{\xi\xi}. \tag{6}$$

More precisely, we evaluate numerically $u_{\xi\xi}$ with a linear scheme based on a 3×3 stencil and then we define iteratively the sequence $u_n(i,j)$ by

$$u_{n+1}(i,j) = u_n(i,j) + \Delta t \cdot (u_{\xi\xi})_n(i,j). \tag{7}$$

Since the mean curvature evolution is undefined when $Du = 0$, we diffuse by half Laplacian whenever the gradient is below a threshold T_g,

$$u_{n+1}(i,j) = u_n(i,j) + \frac{1}{2}\,\Delta t \cdot \Delta u_n(i,j). \tag{8}$$

Indeed, the threshold T_g is required to ensure that local extrema evolve quick enough.

If the modulus of the gradient is greater than the threshold T_g, a linear scheme based on a 3×3 stencil whose coefficients are functions of the direction of the gradient is adopted for the evaluation of $u_{\xi\xi}$,

$$\begin{aligned}(u_{\xi\xi})_n(i,j) = {} & -4\lambda_0 \cdot u_n(i,j) + \lambda_1 \cdot (u_n(i,j+1) + u_n(i,j-1)) + \lambda_2 \cdot (u_n(i+1,j) + u_n(i-1,j)) \\ & + \lambda_3 \cdot (u_n(i-1,j-1) + u_n(i+1,j+1)) + \lambda_4 \cdot (u_n(i-1,j+1) + u_n(i+1,j-1)),\end{aligned} \tag{9}$$

where the choice of λ_i ($i \in \{0, 1, 2, 3, 4\}$) has been discussed in [8].

Nevertheless, FDSes do not commute with increasing contrast changes, which implies that new gray levels can appear in the output image, adding to it further spurious information not present in the input. In order to restore a full contrast invariance, one needs to extract all input upper level sets, apply the finite difference scheme hinted at above to each of them and eventually reconstruct the output image from the stack of the evolved level sets. More formally, let

$$\mathcal{X}_l u = \{(i, j) \mid u(i, j) \geq l\}, \tag{10}$$

be the set at gray level l of the input function u. Then, the scheme can be schematized as in algorithm 1.

Algorithm 1: Stack Filter for the simulation of the MCM equation via a finite difference scheme

decide the time step Δt and the threshold for the gradient T_g;
read the input image and store its values in the array u;
decide the number of iterations needed;
for *each color channel* **do**
  for $l = 1$ *to* 255 **do**
    if *the gray level l is present in the image* **then**
      define the binary image v_l as

$$v_l(i, j) = \begin{cases} 255 & \text{if } (i, j) \in \mathcal{X}_l u \\ 0 & \text{otherwise} \end{cases}; \tag{11}$$

      apply the FDS for the MCM (7) to v_l, obtaining the array w_l;
  evaluate the output image from *superposition principle*, $u(i, j) = \sup \{l \mid w_l(i, j) \geq 127.5\}$;
write the output image from the array u;

Notice that it is useless to consider the 0−level set, since, by definition, $u(i, j)$ is non negative for all (i, j). Therefore, we start our calculations from $l = 1$. It is also a waste of time to process gray levels not present in the image. In fact, if it does not exist a pixel attaining a certain value k,

$$\mathcal{X}_k u = \mathcal{X}_{k+1} u, \tag{12}$$

as the gray levels are quantized. In addition, remember that the filter is constructed to be *contrast invariant*, so when the input does not contain a specific value, such a value must not appear also in the output. With this trick, we make the algorithm really fast in the particular case of binary inputs. However in most real images, almost all the gray levels are present in the picture to be processed.

As pointed out in the description of the FDS for the MCM [8], the application of the scheme to pixel (i, j) involves the knowledge of neighboring pixel values. Nevertheless, when we are considering the borders of the picture, some of them could not be defined. To solve this issue, we operate a symmetrization on the borders of the image, as in the basic finite difference scheme. This solution avoids sharp changes at the edges, but yields an unpleasing *mirroring effect* that makes the figure evolve as if it were the central block of a 3×3 bigger matrix of overlapped copies.

3 Choice of the Parameters and Experimental Testing

First of all, the algorithm satisfies the property of *stability*. Indeed, if this was a serious issue for the original FDS for the MCM, stack filters are by definition *stable*. In fact they achieve a full contrast invariance and therefore they do not allow pixel values to go out of the initial range [0, 255].

The λ_i of the 3×3 stencil on which the finite difference scheme is based are the same as in the final online version of the MCM algorithm [8].

The threshold for the gradient T_g is set to 4 and allows us to have an uncertainty in the evaluation of $|Du|$ smaller than 15% of the real value in the worst case (see Section *Gradient Threshold* of [8] for further details).

When analyzing the original FDS, we showed that a good trade off between performance and processing time was to set $\Delta t = 0.1$, since higher time steps yield a significantly more irregular behavior while smaller ones increase CPU time and do not produce remarkable variations in the output. Instead, as regards the stack filter, we prefer to choose $\Delta t = 0.5$, i.e. the theoretical upper bound due to the application of the heat equation if $|Du| < T_g$, in order to obtain a $5\times$ gain in CPU time, without noticing a definite worsening in performances. The comparative tests that follow confirm this assumption.

3.1 Semigroup Property Check

A function operator T_s processing an image at scale s satisfies the semigroup property if

$$T_n(T_m(u)) = T_{n+m}(u), \tag{13}$$

for every continuous function u.

In particular, for the current implementation of the stack filter the semigroup property is achieved with very good approximation, as similar results are obtained if

1. we apply the algorithm to the input image u with $n_{\text{iter}} = n_1$, getting the output v and we apply it again to v with $n_{\text{iter}} = n_2$;

2. we apply the algorithm to the input image u with $n_{\text{iter}} = n_1 + n_2$.

The validity of this property is shown in figure 1 for three different choices of the numbers of iterations:

- $n_1 = 5$ and $n_2 = 3$;
- $n_1 = 10$ and $n_2 = 15$;
- $n_1 = 30$ and $n_2 = 40$.

Even if the images on the left appear extremely similar to those on the right, the numerical differences between these two sets of pictures are different from zero, due to the forced quantization before the application of the superposition principle. In other words, if we apply the stack filter with $n_{\text{iter}} = n_1$ and then with $n_{\text{iter}} = n_2$, the output image has been quantized two times, while if we apply the stack filter directly with $n_{\text{iter}} = n_1 + n_2$ there is a single quantization operation. As a result, the semigroup property may not be satisfied locally and the number of colors may change. For example, if the original image has 3 different gray levels, smoothing it with $n_{\text{iter}} = n_1$ may convert it to a 2-color image, while smoothing it with $n_{\text{iter}} = n_1 + n_2$ may keep the three colors. Thus, it is impossible to recover the result with $n_{\text{iter}} = n_1 + n_2$ from the result with $n_{\text{iter}} = n_1$, because the stack filter is contrast invariant and cannot add new gray levels. Anyway, it is important to remark again that this phenomenon is visually irrelevant.

(a) Original image

(b) Stack filter with $n_{\text{iter}} = 5$ and then with $n_{\text{iter}} = 3$

(c) Stack filter with $n_{\text{iter}} = 8$

(d) Stack filter with $n_{\text{iter}} = 10$ and then with $n_{\text{iter}} = 15$

(e) Stack filter with $n_{\text{iter}} = 25$

(f) Stack filter with $n_{\text{iter}} = 30$ and then with $n_{\text{iter}} = 40$

(g) Stack filter with $n_{\text{iter}} = 70$

Figure 1: The semigroup property is achieved by the proposed implementation of the stack filter with excellent approximation: compare (b) and (c), (d) and (e), and (f) and (g).

3.2 Linear and Radial Checks

Assessment tests are to see if a linear function stands still and a radial function remains radial after the application of the stack filter. To this end, we create digital images whose pixel values vary linearly and radially with the current row and column numbers. Then, in the first case, we evaluate the numerical difference between the original linear image and its mean curvature evolutions. On the contrary, if the input is radial, the difference image is useless and we focus on the distance between evolved level lines and real circles. No changes in the numerical results can be noticed when time step varies in the range $\{0.1, 0.2, 0.3, 0.4, 0.5\}$, so that we can take the highest possible value for Δt. Some visual outcomes where R goes from 0 to 100 are to be found in figures 2 and 3. The gray levels of input images have been quantized in order to emphasize the *mirroring effect*, which is responsible for the deformations appearing on the borders of the evolutions.

3.3 Relationship between Normalized Scale and Number of Iterations

This relationship has been treated extensively when dealing with the original finite difference scheme for the Mean Curvature Motion. In that context, we wanted to see how the formula

$$n_{\text{iter}} = \frac{R^2}{2\Delta t} \tag{14}$$

works out in practice. Therefore we created digital circles of various radii r and we applied the numerical scheme, comparing the theoretical value with the actual number of iterations needed to make the circle vanish after a threshold with $\lambda = 127.5$. In other words, we were implicitly employing a stack filter.

When $\Delta t = 0.1$, we noticed that numerical results stay close to theoretical ones for normalized scales in [2, 50]. In particular, in the range [14, 46] the difference is hardly noticeable and the error does not exceed 1%. For higher values of R the curve of experimental data started to increase faster and faster.

As can be seen in figure 4, similar outcomes are obtained when $\Delta t = 0.5$. In the range [16, 46] the percent error stays below 1% and if R belongs to [0, 44] the absolute error is < 6. In conclusion, the approximation is highly reliable when $R < 50$.

3.4 Contrast Invariance Check

Stack filters are by definition *contrast invariant*, so it is interesting to check this property, comparing the performances achieved with different values of time step and when employing the original FDS.

Given a digital picture u_0 and a contrast change g, the idea is to make the difference between the two evolutions obtained applying first the scheme for the MCM and afterwards the function g and vice versa. The *Root Mean Square Error* (RMSE) between these two images is to be minimized,

$$\text{RMSE}(v, w) = \sqrt{\frac{\sum_{i=1}^{n_{row}} \sum_{j=1}^{n_{col}} (v(i,j) - w(i,j))^2}{n_{row} \cdot n_{col}}}. \tag{15}$$

We start employing the simplest contrast change possible, i.e. a straight line, being careful to take an increasing function and to respect the inequalities $0 \leq g(u_0(i,j)) \leq 255$. A possibility is to set $g(u_0) = 0.5 \cdot u_0 + 10$.

Notice that the numerical difference between the two evolutions can be done before or after passing from float arrays to the usual representation of images as made by 8-bit unsigned integers. Obviously,

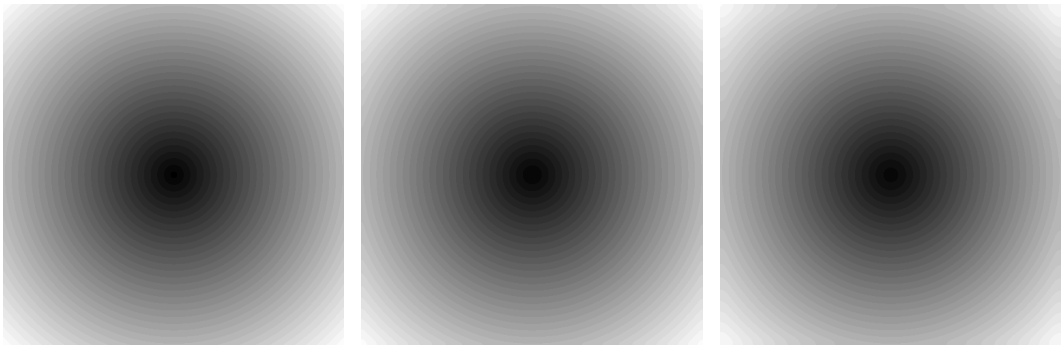
(a) $R = 0$ (b) $R = 5$ (c) $R = 10$

(d) $R = 20$ (e) $R = 30$ (f) $R = 40$

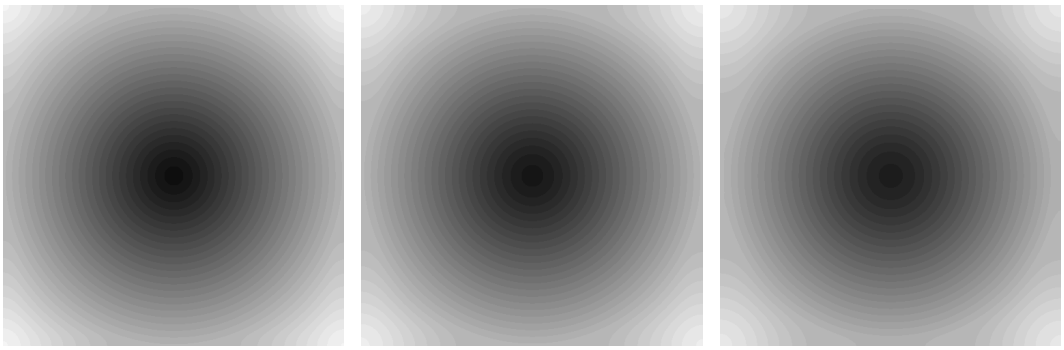
(g) $R = 50$ (h) $R = 60$ (i) $R = 70$

(j) $R = 80$ (k) $R = 90$ (l) $R = 100$

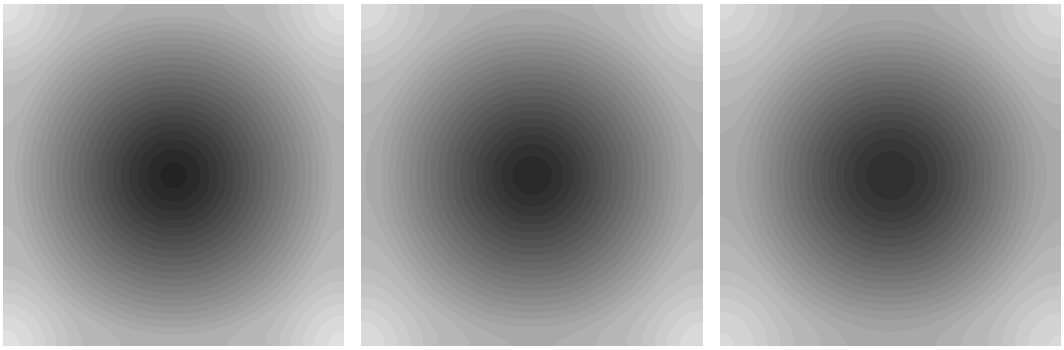
Figure 2: Evolution of a linear function with quantization of gray levels.



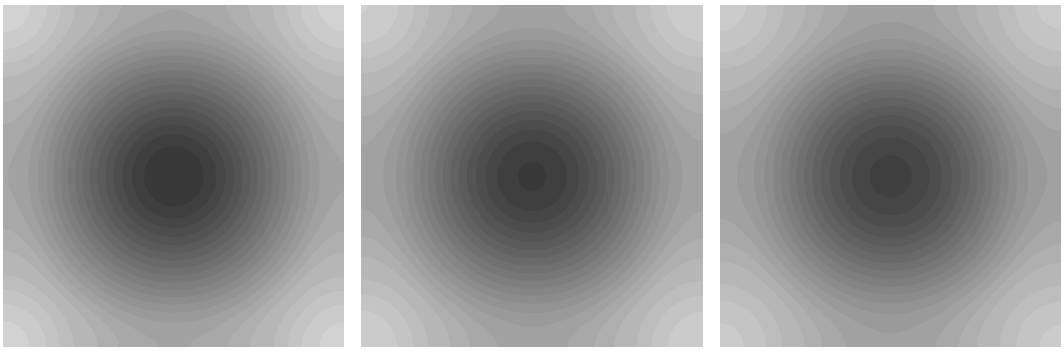
(a) $R = 0$ (b) $R = 5$ (c) $R = 10$



(d) $R = 20$ (e) $R = 30$ (f) $R = 40$



(g) $R = 50$ (h) $R = 60$ (i) $R = 70$



(j) $R = 80$ (k) $R = 90$ (l) $R = 100$

Figure 3: Evolution of a radial function with quantization of gray levels.

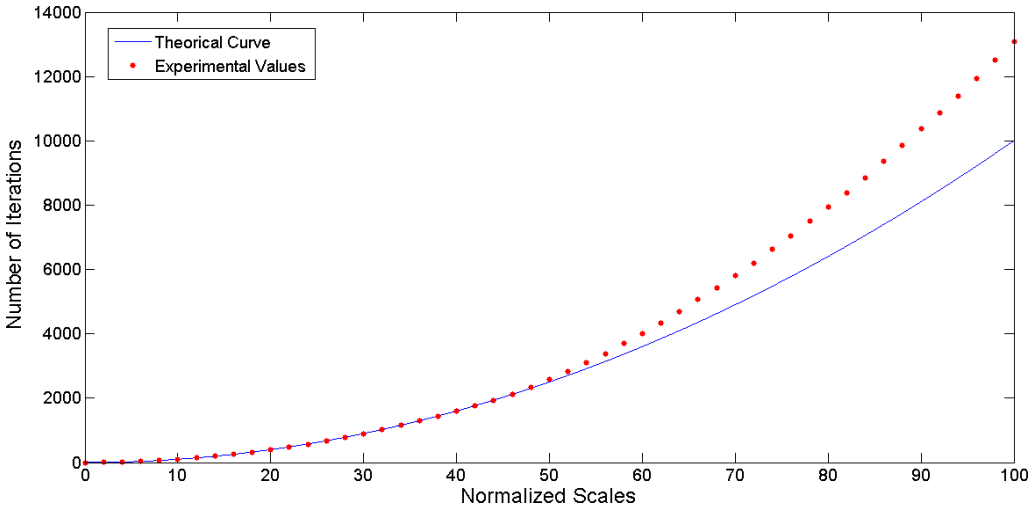


Figure 4: Relationship between normalized scale and number of iterations when $R \in [0, 100]$

in the second case the RMSE is bigger, because of the inevitable rounding and approximation errors in the conversion from float to integer values. However also in the first case RMSE $\neq 0$. Indeed, when applying a contrast change, some of the input gray levels might be lost. Thus, the stack filter will process only the remaining ones and the output image will be less accurate. Theoretically speaking, a perfect invariance can be achieved if in the implementation of the stack filter the quantization step l between two successive level sets satisfies

$$l \leq \inf_{x \in [0,255]} g'(x), \tag{16}$$

where g' denotes the first derivative.

This equation has sense only when g is at least C^1. Otherwise, if g is not regular enough or if we do not have a closed form for it, the condition to be checked is instead

$$l \leq \min_{a \in \{0,1,\cdots,254\}} (g(a+1) - g(a)). \tag{17}$$

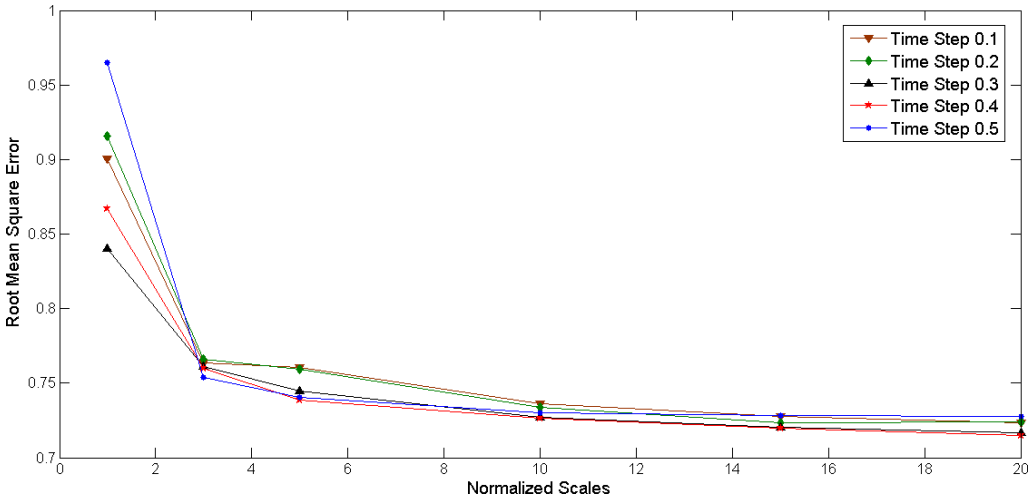
The analysis of figure 5 make us conclude that the RMSE decreases but does not change its trend if we consider float arrays instead of images. Therefore the successive tests will take into consideration only RMSEs between images.

When the normalized scale is too small, the errors are due to the loss in level sets. Then, for $R \geq 3$ we notice how the differences in RMSE are really scarce while time step varies in the range $\{0.1, 0.2, 0.3, 0.4, 0.5\}$. As a consequence we can take $\Delta t = 0.5$ for CPU time reasons.

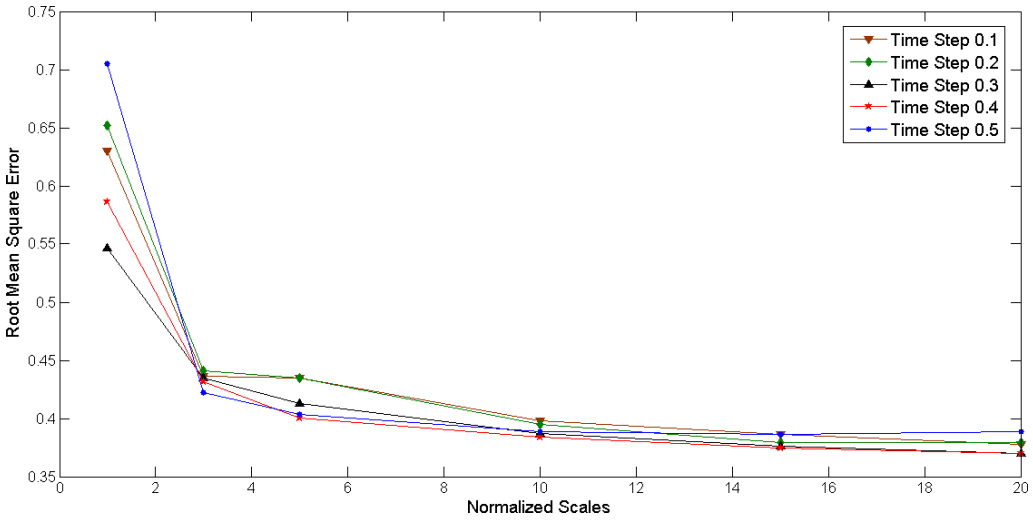
Afterwards, we vary the normalized scale, comparing the behavior of the original finite difference scheme ($T_g = 4$ and $\Delta t = 0.1$) with that of the stack filter with $\Delta t = 0.5$.

As shown in figure 6, the RMSE produced by the stack filter remains practically constant when R varies and is due to the loss in gray levels subsequent to the application of the contrast change g. Instead, if we consider the original FDS, we always notice bigger values of RMSE, which attains a maximum of almost 3 in proximity of $R = 20$ and then starts decreasing. In short, the stack filter is decidedly more contrast invariant than the simple FDS for the MCM.

Ulterior tests can be performed on other images employing different contrast changes. Once again, the aim is to compare the outcomes obtained applying the stack filter with $\Delta t = 0.5$ and the



(a) Test performed on RGB images



(b) Test performed on RGB float arrays

Figure 5: Influence of time step in the contrast invariance of RGB images and float arrays when the contrast change is linear.

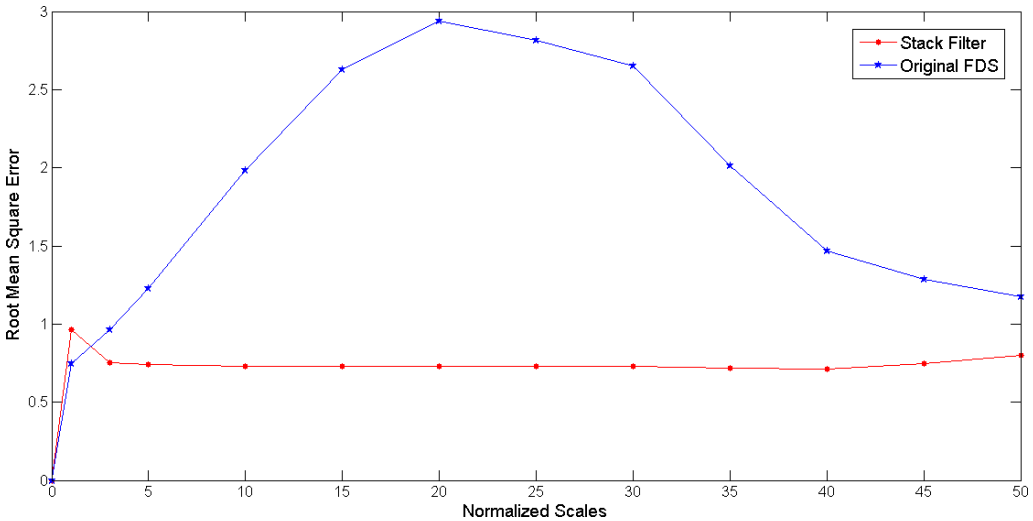


Figure 6: Contrast invariance test for the stack filter and the original FDS when g is a linear function.

original FDS with $\Delta t = 0.1$ and $T_g = 4$.

Quadratic contrast change

$$g(u_0) = \frac{u_0^2}{255} \tag{18}$$

This is a convex function satisfying $g(0) = 0$ and $g(255) = 255$. As regards the stack filter, the RMSE decreases slowly passing from 1.2 to 1.05 after the initial peak of 1.7 when R runs in the range [3, 50]. Instead, the original FDS lacks contrast invariance and the RMSE is a strictly increasing function of the normalized scale, attaining a maximum of 6.5 when $R = 50$ (see figure 7).

Square root contrast change

$$g(u_0) = \sqrt{255 \cdot u_0} \tag{19}$$

This is a concave function satisfying $g(0) = 0$ and $g(255) = 255$. The conclusion is the same as in the two previous tests: the numerical scheme for the stack filter is practically contrast invariant and the RMSE is a slowly decreasing function of the normalized scale that passes from 1.2 to 0.8. On the other hand, the basic FDS has an irregular behavior and the RMSE increases quickly attaining a maximum of 3.6 when $R = 20$ (see figure 8).

Uniform equalization

Let $F^{(-1)}$ be the left pseudo-inverse of F, M be the number of pixels of the image u_0 and H_{u_0} its cumulative histogram defined as

$$H_u(l) = |\{x \mid u(x) \leq l\}|, \tag{20}$$

where $|A|$ denotes the cardinality of the set A. Indeed, the contrast change needed to modify the cumulative histogram of u_0, making it as close as possible to a linear function is

$$g(l) = \lfloor M \cdot l/255 \rfloor^{(-1)} \circ H_{u_0}(l). \tag{21}$$

Such an operation is called *uniform equalization* and is particularly relevant in everyday applications and common photo-retouching programs. Notice that the contrast change taken into account maps integer pixels values in integer pixel values. Thus, we need to round gray levels to the nearest integer before applying uniform equalization. As shown by figure 9, this example is characterized by a definitely more irregular behavior of both schemes. As regards the stack filter, the RMSE increases monotonically until $R = 40$, attaining a maximum value close to 12. On the contrary, when dealing with the original FDS, the RMSE increases until $R = 50$, where it is equal to 19.5.

3.5 Euclidean Invariance

The Mean Curvature Motion equation is invariant with respect to transformations that preserve distances. Thus, it is interesting to analyze the Euclidean invariance properties of the numerical algorithms simulating the MCM via the basic finite difference scheme and via the stack filter. During the numerical experiments we have considered pictures after the application of

1. the finite difference scheme (in its original or stack filter version) for the simulation of the MCM, and then a transformation f;

2. a transformation f first, and then the finite difference scheme.

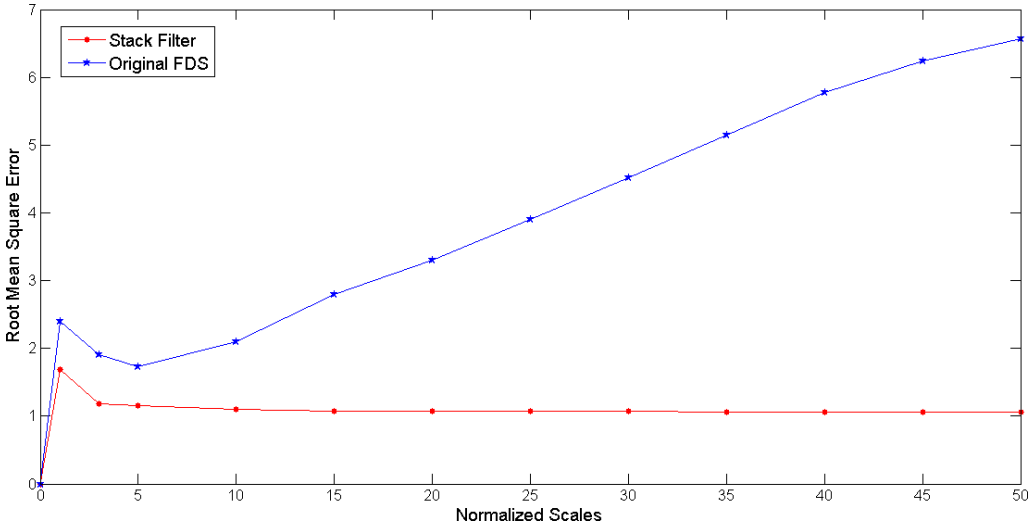


Figure 7: Contrast invariance test for the stack filter and the original FDS when g is a quadratic function.

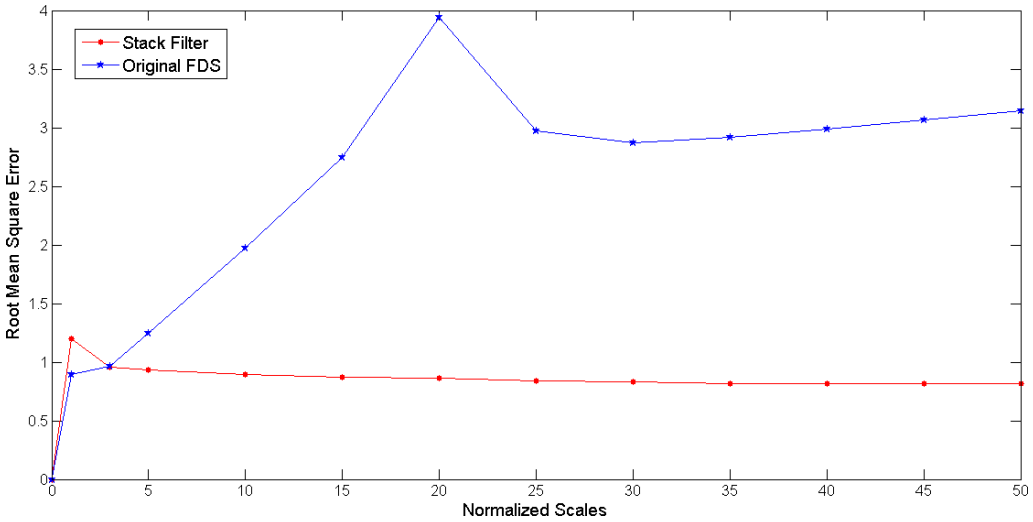


Figure 8: Contrast invariance test for the stack filter and the original FDS when g is a square root function.

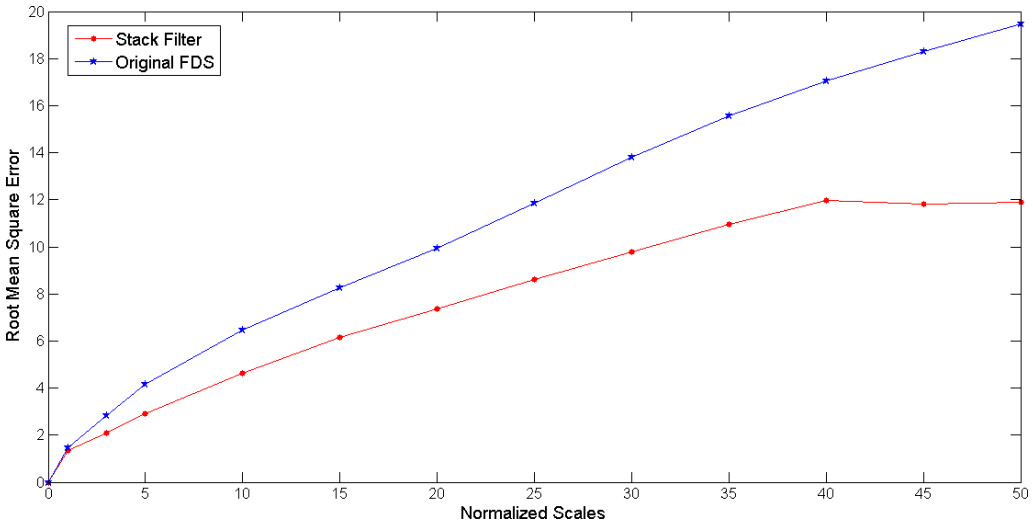


Figure 9: Contrast invariance test for the stack filter and the original FDS if we apply a uniform equalization.

The aim is to check if and to what degree these pairs of images are equal. To do so, we have considered as significant metrics of comparison both the RMSE (15) and the maximum of the absolute difference between the pixel values of the images, i.e.

$$\mathrm{diff}_{\max}(v, w) = \max_{i,j} |v(i, j) - w(i, j)|. \tag{22}$$

It is well known that no single norm can be used in general for comparing digital pictures, and therefore using several norms tends to be much more informative. The results for different choices of the transform f can be summarized as follows.

- Both the FDS and the stack filter for the MCM are invariant with respect to translations. Indeed, since the algorithm is a pixel-based finite difference scheme, if we consider an image with a sufficiently big homogeneous background in order to eliminate border effects, applying a rigid translation to the input or to the output of the algorithm does not result in any difference.

- When f is a rotation, the stack filter is able to improve the Euclidean invariance of the scheme, especially for input images that present a high number of intermediate gray levels. Notice that during the transformation a linear interpolation is employed. Figure 10 deals with a rotation of $\pi/6$ applied to an image with a small number of gray levels, i.e. that is almost binary. In this case, for both the algorithms the RMSE does not exceed the value of 2.4 and the maximum absolute difference stays below 70. On the other hand, figure 11 considers an input image which contains a significant number of intermediate color levels and shows that the stack filter yields a more regular behavior than the original FDS. In particular, the RMSE for the former scheme stays below 4, while for the latter increases till a maximum > 16. This effect is mostly due to the evolution of the image borders (see figure 12): if we apply first the rotation and then the FDS, the smoothing is uniform on the whole border of the picture; on the other hand, if we apply first the FDS and then the rotation, the smoothing is concentrated on the angles. Since the stack filter does not yield any shading of the contour, the RMSE for the latter algorithm remains low. If the computation of the RMSE is restricted to the central part of the image neglecting the boundaries, significantly smaller values of this indicator can be obtained. Indeed, according to figure 13, the RMSE for the FDS does not exceed a maximum of 5, and for the stack filter remains below 2.5. No significant changes in the maximum of the absolute

differences can be noticed when we consider only the central crop of the picture. It is also interesting to remark that the particular choice of the angle of rotation does not affect in a significant way the outcomes of the experiment.

- The original FDS is not invariant with respect to zooms, while the stack filter presents a more regular behavior. As for the previous case of a rotation, during the zooming transformation a linear interpolation is employed. According to the numerical results of figure 14 which refers to a 0.5× zoom and to an almost binary input image, the RMSE for the basic finite difference scheme increases monotonously, while the RMSE for the stack filter stays below 2.5. As concerns the maximum absolute difference, both the algorithms present fairly big values of this indicator (around 90 when $R = 5$ and $R = 50$ for the stack filter and about 60 at the same normalized scales for the original FDS). Similar results are to be found in figure 15 for an input image that contains a high number of intermediate gray levels. Remember that when applying the smoothing to the zoomed image we need to multiply the normalized scale by the zooming factor (i.e. by 0.5 in this case) to obtain fair results. Indeed, by definition, the normalized scale represents the amount of smoothing necessary to shrink to a point a circle of radius R.

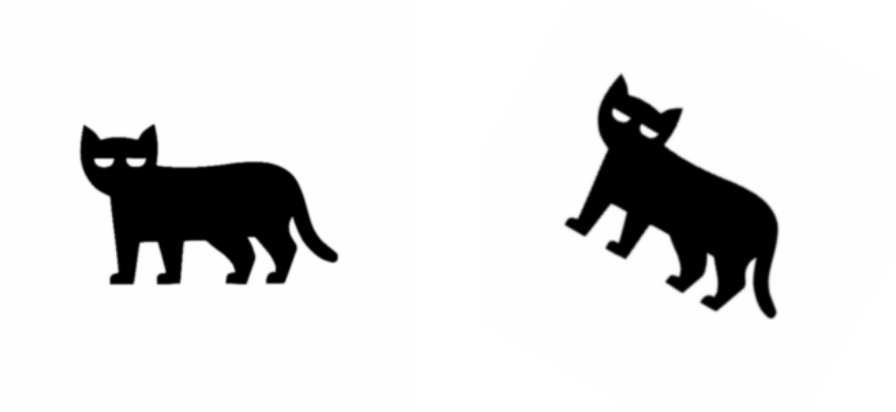
3.6 Denoising Properties

Stack filters display interesting denoising qualities and are really good tools as concerns the reconstruction of severely corrupted images. Our aim is to verify this statement and to evaluate numerically the process, comparing the performances obtained changing the value of time step or employing the original FDS rather than the stack filter.

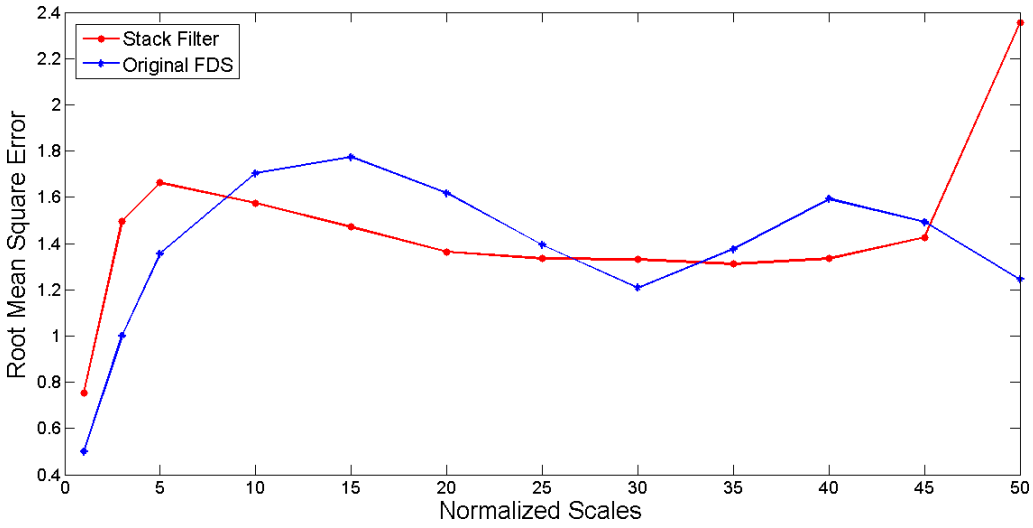
The first thing to do is to create a simple digital image, in our case a black square of 100×100 pixels on a white background (500×500 pixels). Then, we apply different percentages p of *salt and pepper* noise using the pseudo-random number generator known as *Mersenne Twister* in order to get a random variable uniformly distributed in [0, 1]. This means that a pixel assumes the value of a random variable uniformly distributed in $\{0, 1, \ldots, 255\}$ with probability p. Once obtained the noisy image, we apply the stack filter at different normalized scales, evaluating the RMSE with respect to the original test image. We expect that such a value decreases monotonically until it attains a global minimum and starts increasing. Indeed, in the descending part the smoothing shows its denoising qualities, i.e. it modifies the image but the most evident change lies in the removal of *salt and pepper* noise. Then, when noise is completely eliminated (global minimum point), the filter continues to smooth and the RMSE increases because the output images become more and more different from the original one. The behavior of the original FDS should be similar, but we expect higher values of RMSE.

First of all, we examine the numerical results when $p = 30\%$, R goes from 0 to 5 with a step of 0.5 and Δt belongs to $\{0.1, 0.2, \ldots, 0.5\}$ (consider figure 16). The minimum in RMSE is always obtained at a normalized scale of 3.5 and its value oscillates between 3.3 and 3.95. From a visual point of view, the evolutions performed with different time step values are indistinguishable. Therefore, we have decided to put in the final online version of the algorithm $\Delta t = 0.5$.

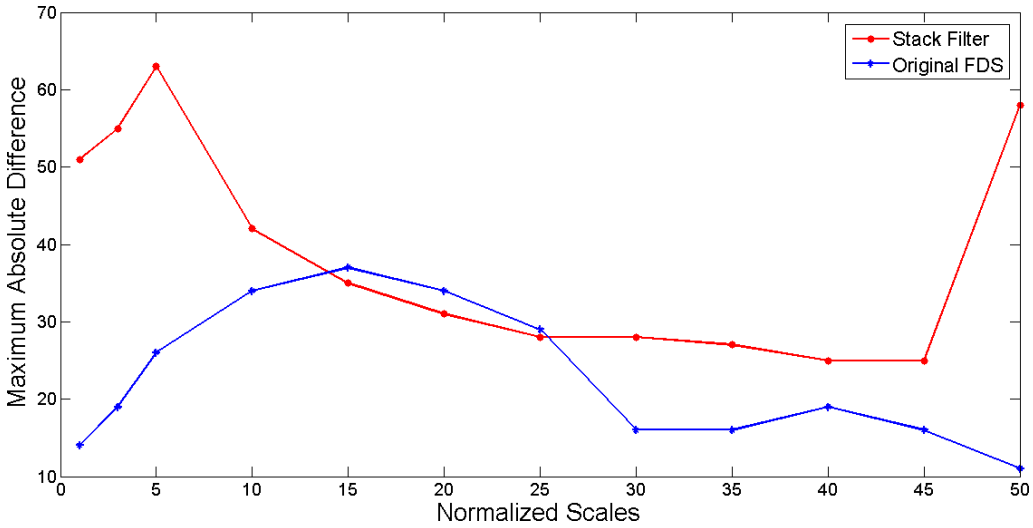
Subsequently, we compare the behavior of the stack filter and of the original finite difference scheme, noticing that the performances of the first are always better. In particular, the improvement both in numerical and visual results becomes more and more evident as the percentage of noise increases. As regards the stack filter, we conclude that the denoising process is perfect if $p = 30\%$, starts to have some problems when $p = 40\%$ and loses much of its accuracy if $p = 50\%$. Remember that the processed images shown in the figures below correspond to the minimum in RMSE and not to the normalized scale in which we see that noise is vanished. Indeed, especially when $p = 30\%$,



(a) Original image (b) Image after a rotation of $\pi/6$



(c) Comparison for the RMSE metric

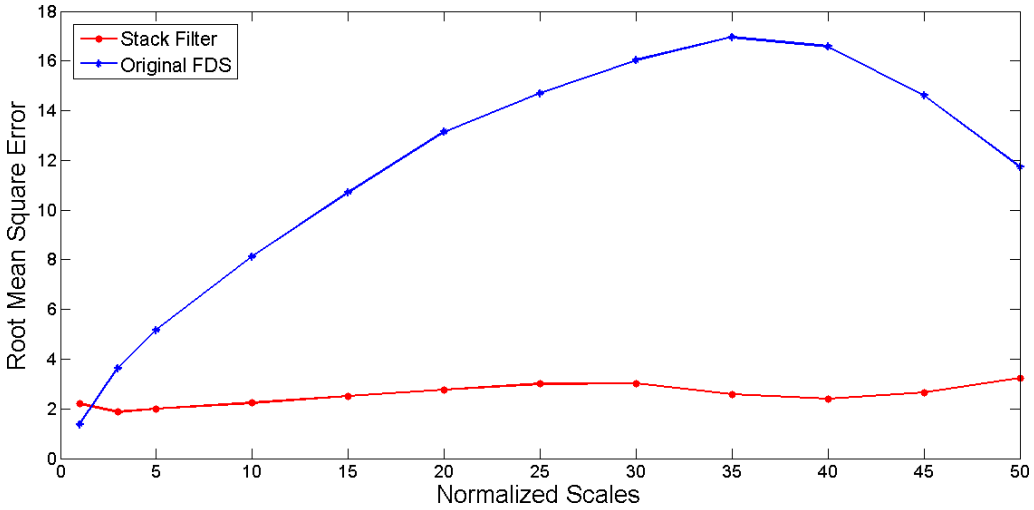


(d) Comparison for the $\text{diff}_{\max}$ metric

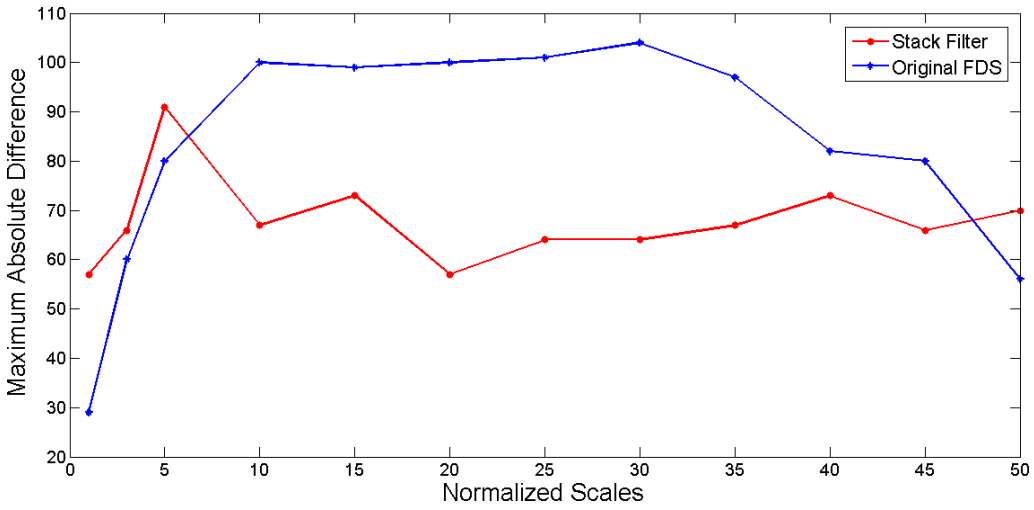
Figure 10: Euclidean invariance test for the stack filter and the original FDS when the transformation f is a rotation of $\pi/6$ and the input image is almost binary.

(a) Original image

(b) Image after a rotation of $\pi/6$



(c) Comparison for the RMSE metric

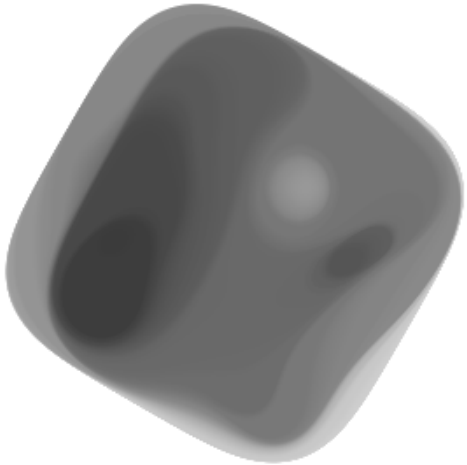


(d) Comparison for the $\text{diff}_{\max}$ metric

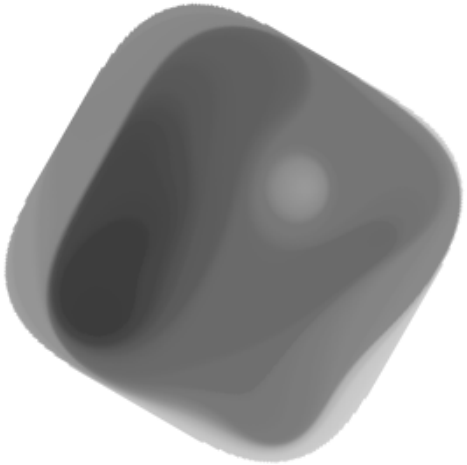
Figure 11: Euclidean invariance test for the stack filter and the original FDS when the transformation f is a rotation of $\pi/6$ and the input image contains a high number of intermediate gray levels.

(a) Image after a rotation of $\pi/6$ followed by the application of the FDS with $R = 35$

(b) Image after the application of the FDS with $R = 35$ followed by a rotation of $\pi/6$



(c) Image after a rotation of $\pi/6$ followed by the application of the stack filter with $R = 35$

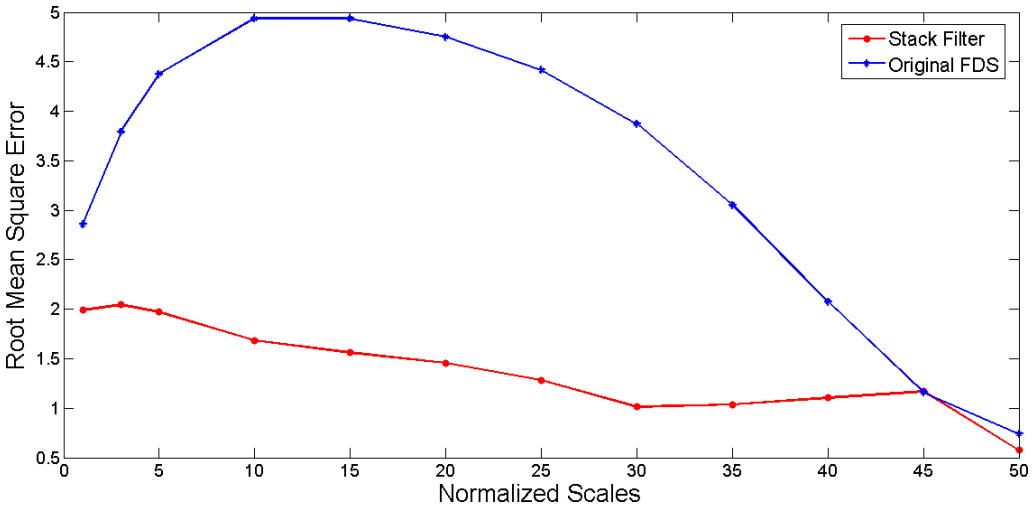


(d) Image after the application of the stack filter with $R = 35$ followed by a rotation of $\pi/6$

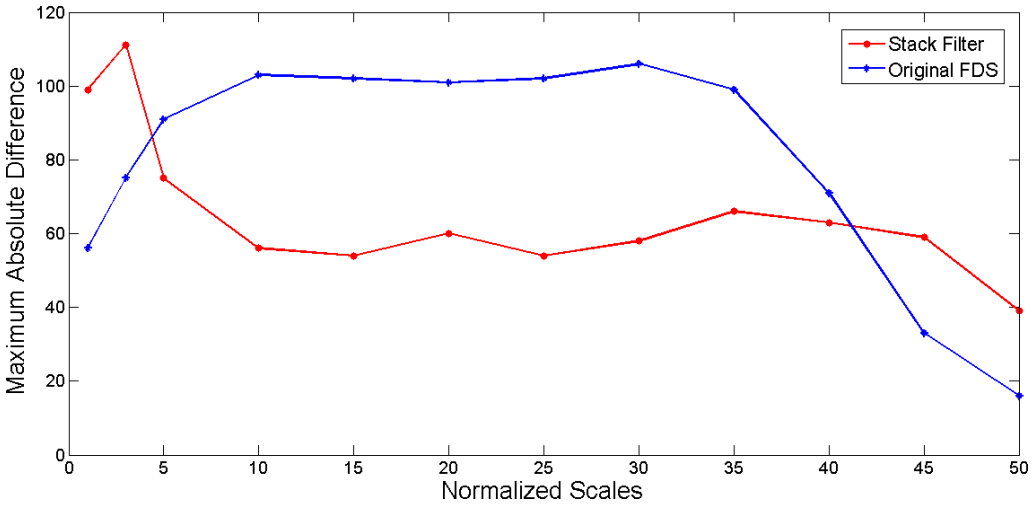
Figure 12: Boundary evolutions for the stack filter and the original FDS.

(a) Original image

(b) Image after a rotation of $\pi/6$



(c) Comparison for the RMSE metric

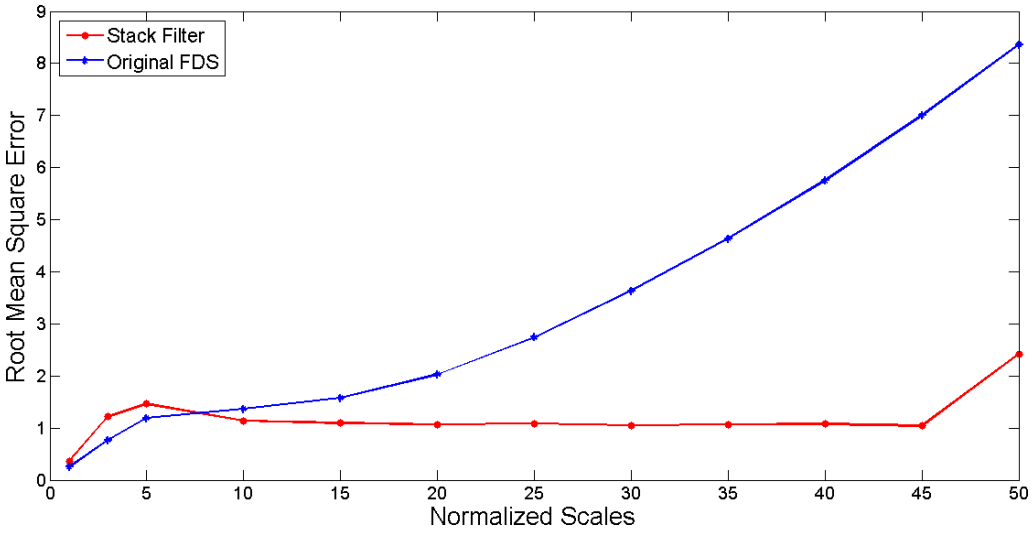


(d) Comparison for the $\text{diff}_{\max}$ metric

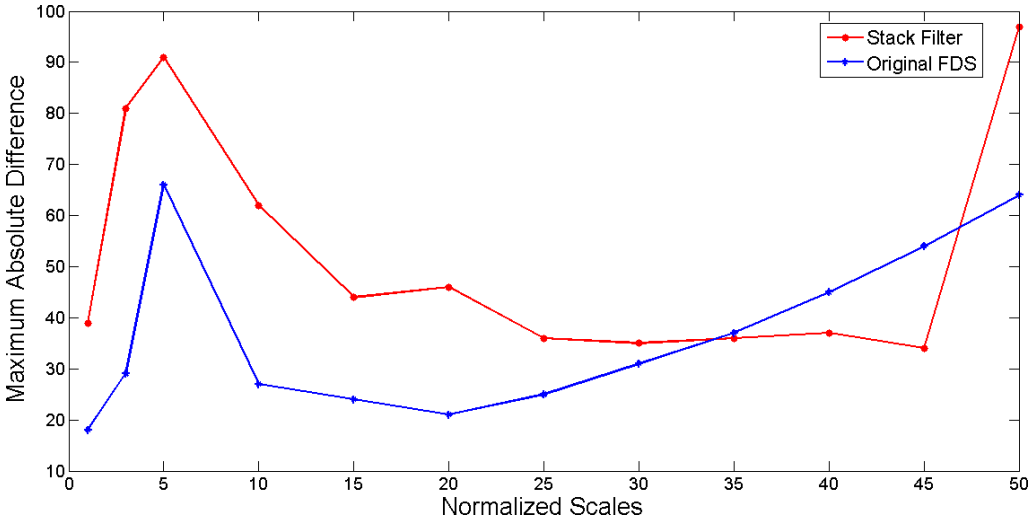
Figure 13: Euclidean invariance test for the stack filter and the original FDS when the transformation f is a rotation of $\pi/6$ and the computations are restricted to the central part of the image.

(a) Original image

(b) Image after a 0.5× zoom



(c) Comparison for the RMSE metric

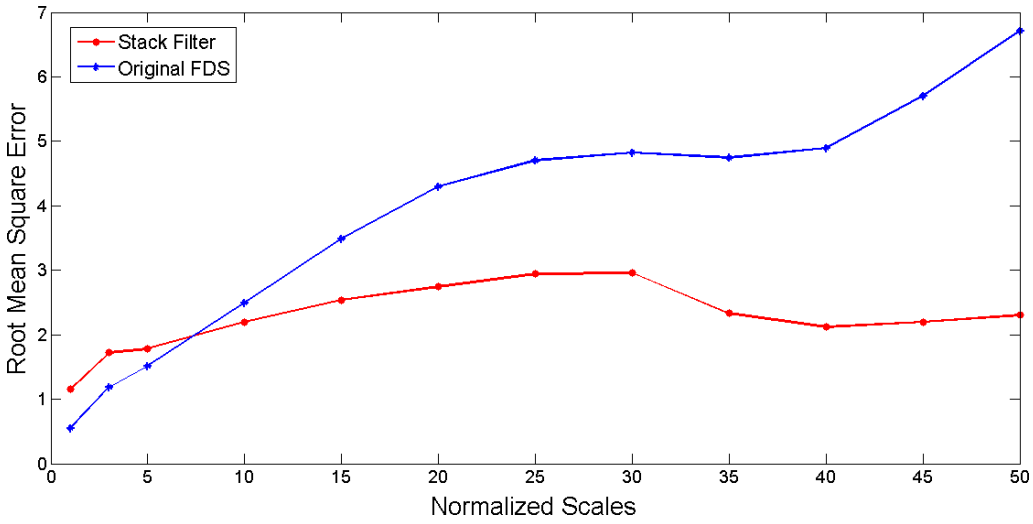


(d) Comparison for the $\text{diff}_{\max}$ metric

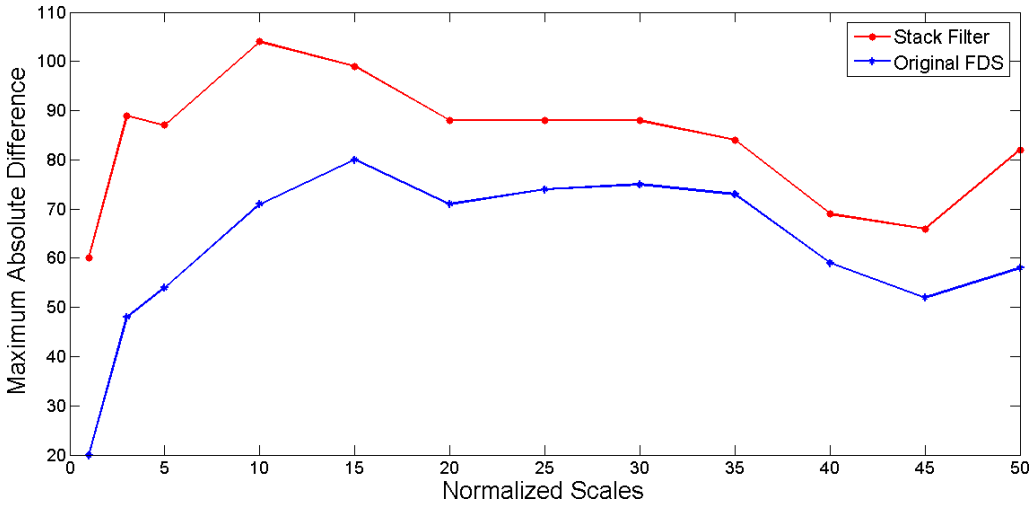
Figure 14: Euclidean invariance test for the stack filter and the original FDS when the transformation f is a 0.5× zoom and the input image is almost binary.

(a) Original image

(b) Image after a 0.5× zoom



(c) Comparison for the RMSE metric



(d) Comparison for the $\text{diff}_{\max}$ metric

Figure 15: Euclidean invariance test for the stack filter and the original FDS when the transformation f is a 0.5× zoom and the input image contains a high number of intermediate gray levels.

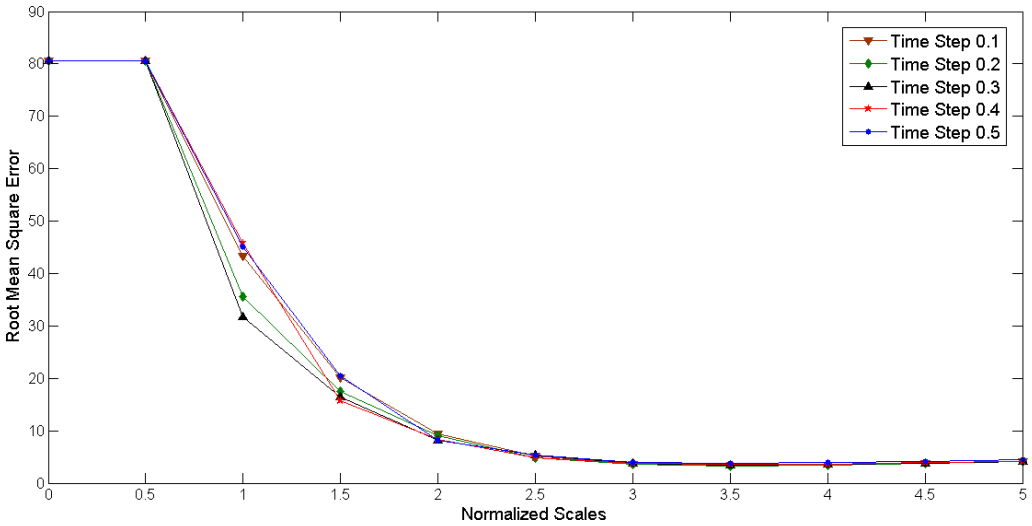


Figure 16: Influence of time step in the denoising properties of stack filter (30% of salt and pepper noise).

higher values of R are needed if we want to eliminate the unpleasant gray spots that affect noisy images. Evidently, these stains yield a smaller RMSE than that produced by a further smoothing of the black square. In particular,

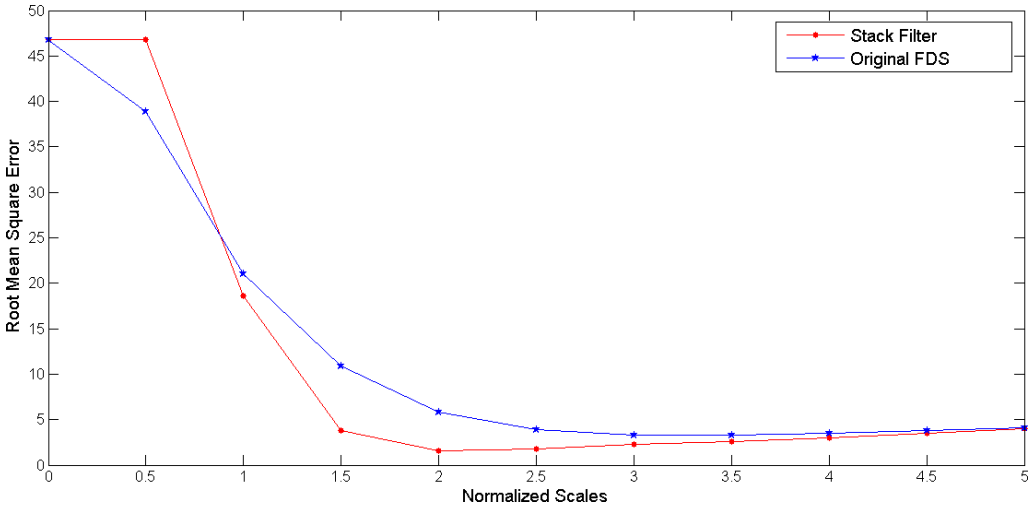
- $p = 10\%$ is a percentage of noise too low to appreciate a definite visual improvement between the two algorithms. Nevertheless, a careful observation shows that the image processed by the stack filter is cleaner and less smooth. Indeed, the minimum of about 1.5 is attained at a normalized scale of 2, while if we use the original FDS we have a minimum of 3.3 obtained when $R = 3.5$ (see figure 17).

- When $p = 20\%$ the visual improvement starts to be more evident. The image denoised with the original FDS is full of gray spots, while, if we process the level sets, the outcome seems to be perfect. Numerically, as can be seen in figure 18, this reflects on a smaller value of the minimum RMSE (2.5 against 7.8) attained at a smaller normalized scale (2.5 against 5).

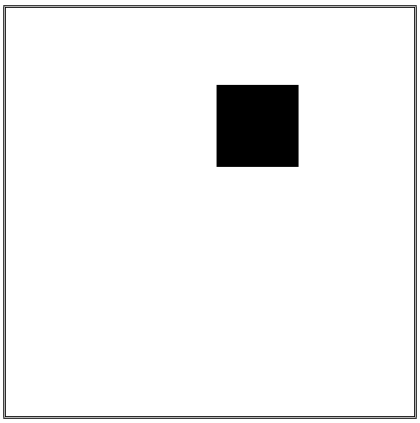
- If $p = 30\%$, the minimum in RMSE reached processing the level sets is about five times smaller (3.7 against 18.6). This affects visual outcomes, making the original FDS fail, while the stack filter performs in an excellent way (see figure 19).

- The difference in performances between the two algorithm increases quickly with the percentage of noise. When $p = 40\%$ a minimum RMSE < 6 attained when $R = 5.5$ is opposed to a minimum of > 33 obtained when $R = 8.5$. From a visual point of view, if we use the original FDS the result is a black square with rounded corners immersed in a gray fog. However, as shown by figure 20, sporadic spots start to appear also in the picture processed by the stack filter.

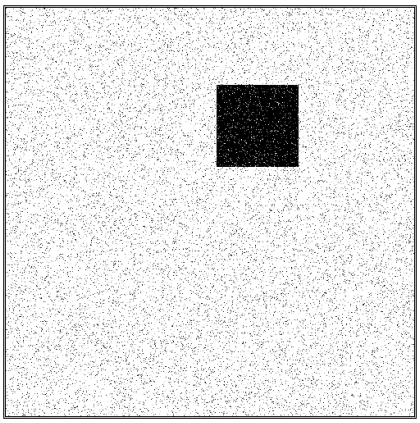
- If $p = 50\%$, the difference in RMSE minima increases in absolute value but not in percentage, as we have 13.7 against 49. Furthermore, such a minimum is attained at a normalized scale of 11.5 against a value of 9 in the case of the original FDS. The results obtained applying the stack filter are still decidedly better, but the percentage of noise starts to be too high for the algorithm in question (see figure 21).



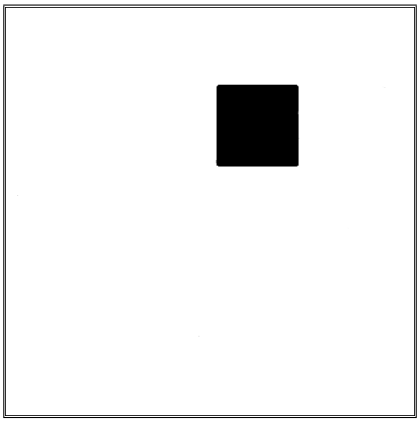
(a)



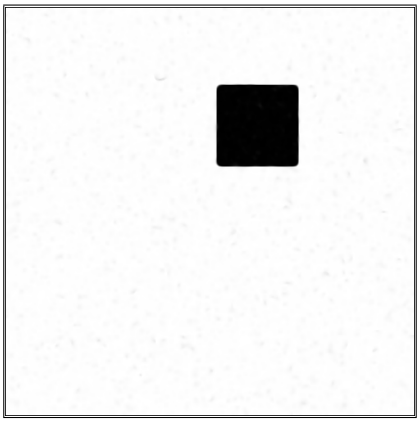
(b) Original image



(c) Noisy image

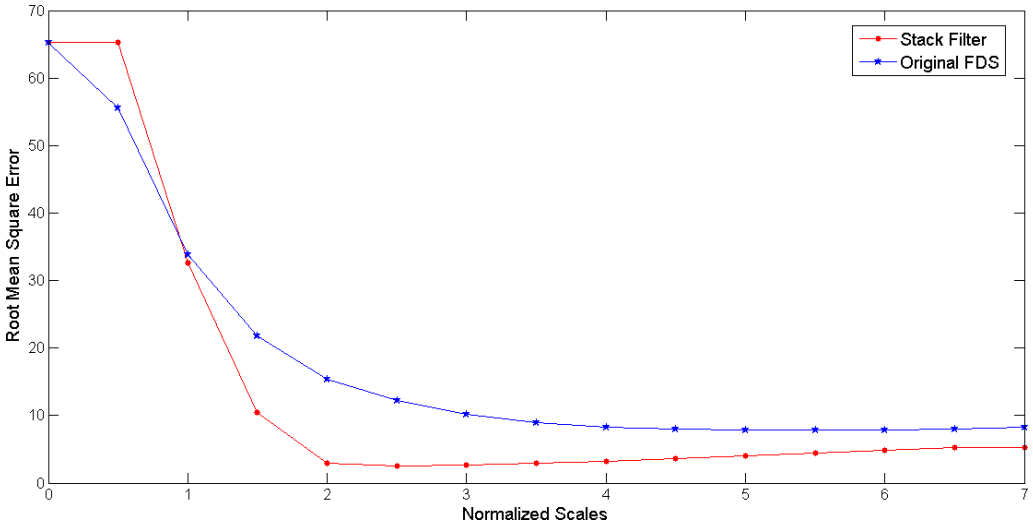


(d) Denoising via the stack filter

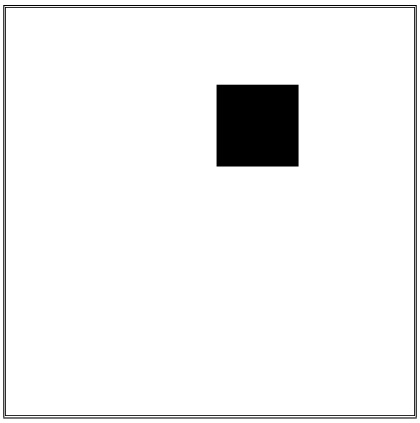


(e) Denoising via the original FDS

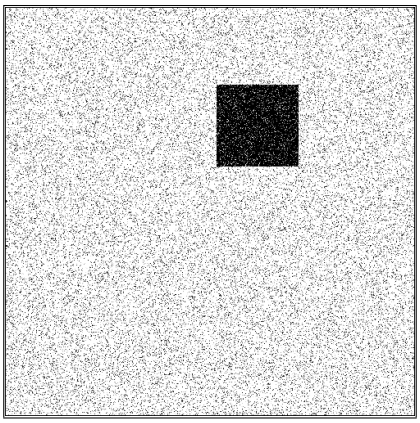
Figure 17: Denoising properties of the stack filter and of the original FDS when $p = 10\%$.



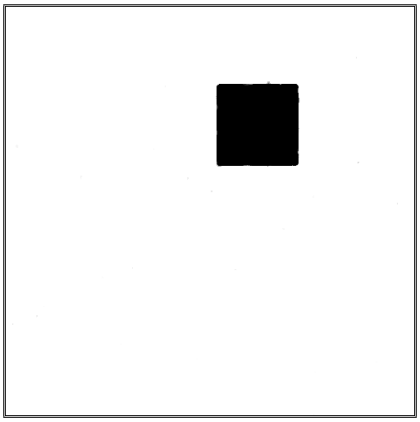
(a)



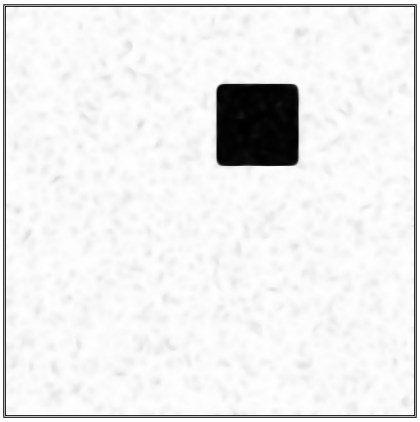
(b) Original image



(c) Noisy image

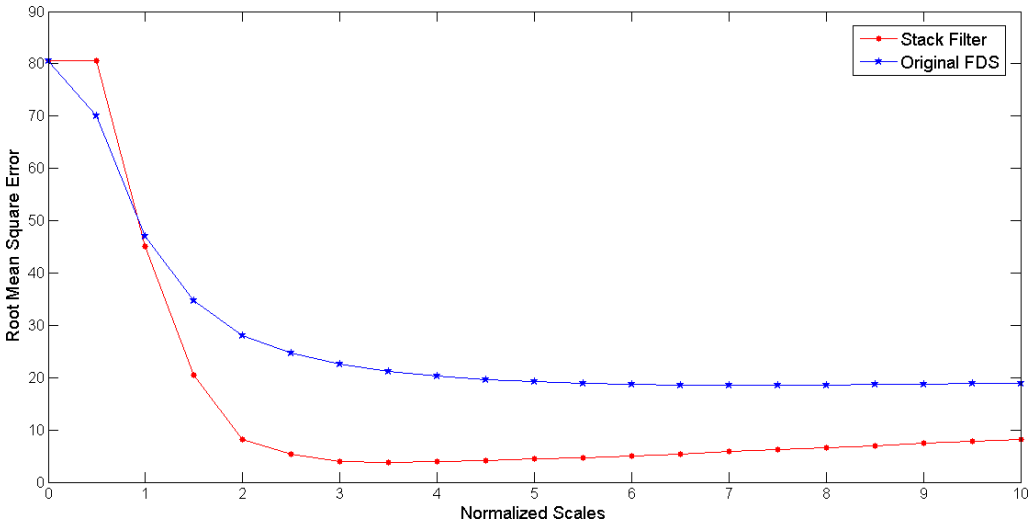


(d) Denoising via the stack filter

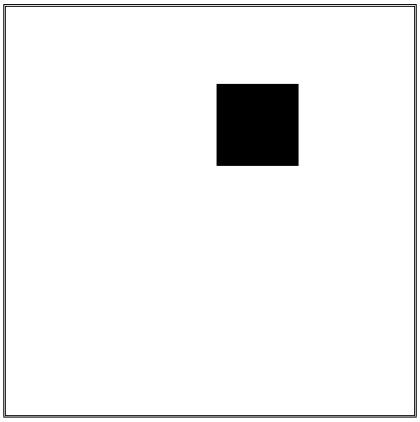


(e) Denoising via the original FDS

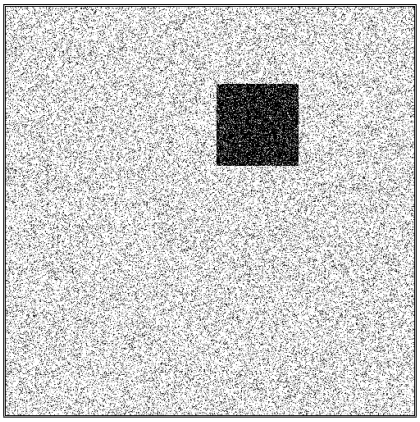
Figure 18: Denoising properties of the stack filter and of the original FDS when $p = 20\%$.



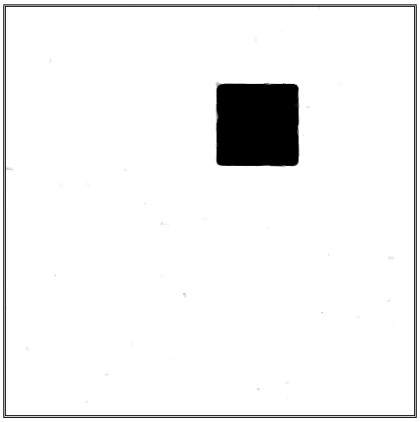
(a)



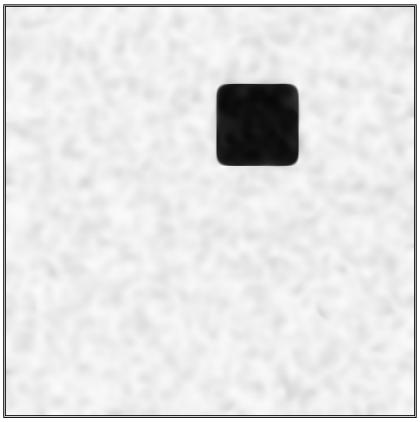
(b) Original image



(c) Noisy image

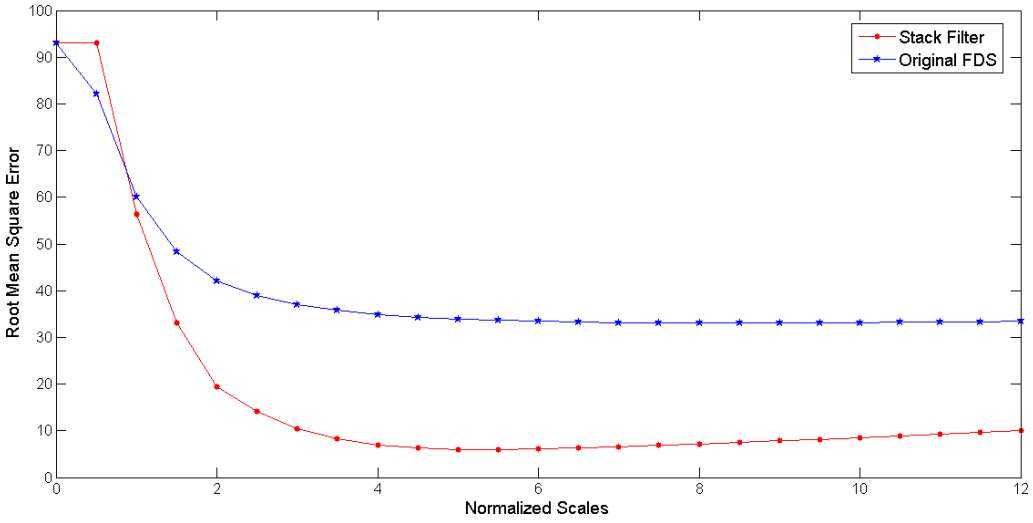


(d) Denoising via the stack filter

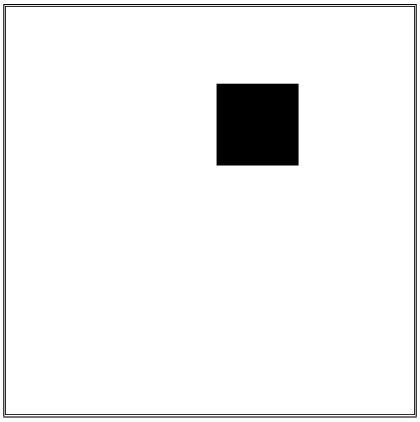


(e) Denoising via the original FDS

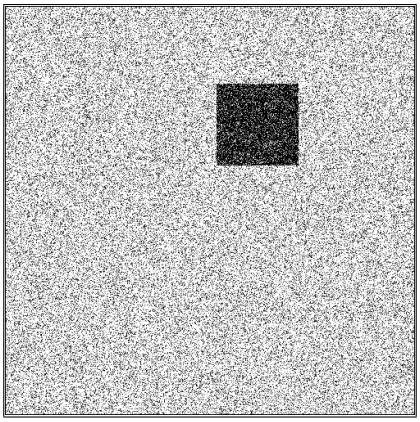
Figure 19: Denoising properties of the stack filter and of the original FDS when $p = 30\%$.



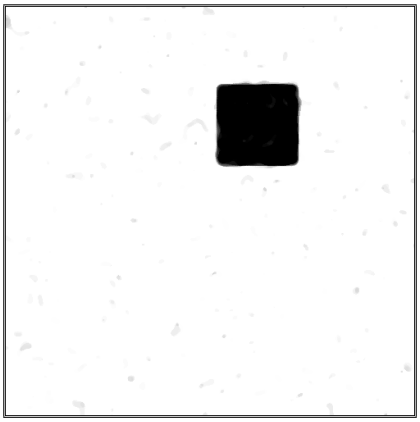
(a)



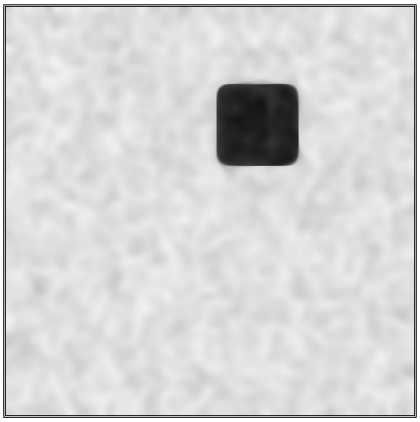
(b) Original image



(c) Noisy image

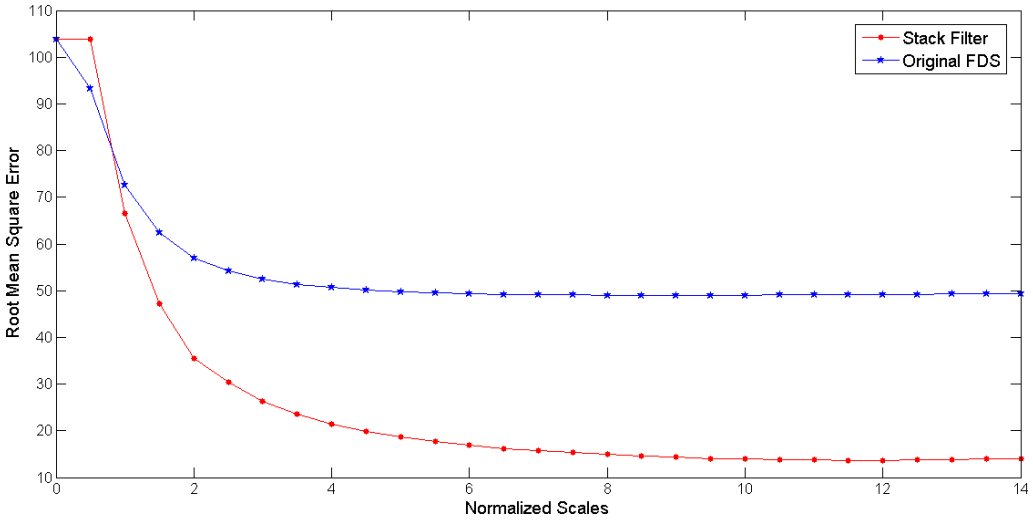


(d) Denoising via the stack filter

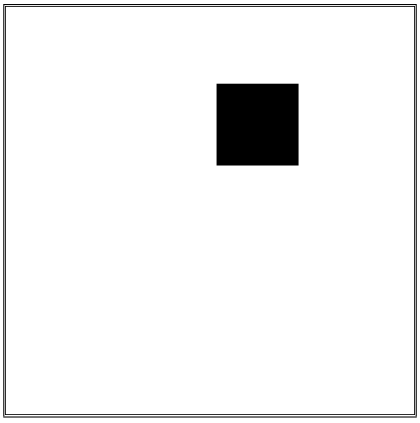


(e) Denoising via the original FDS

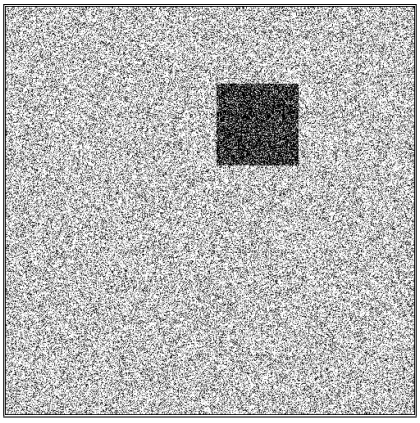
Figure 20: Denoising properties of the stack filter and of the original FDS when $p = 40\%$.



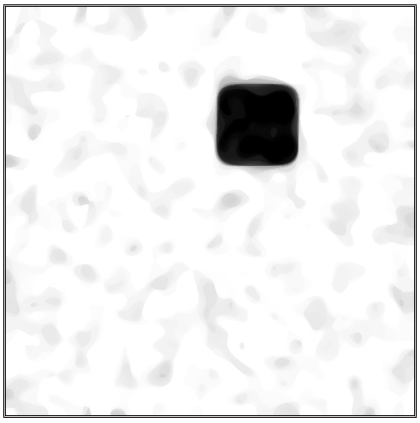
(a)



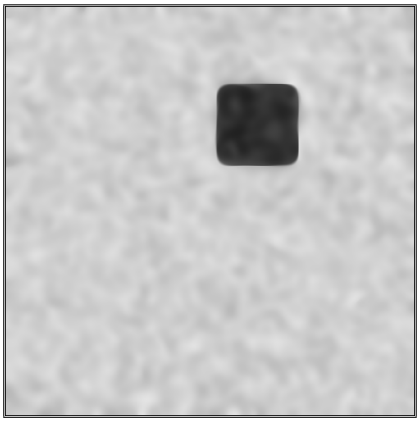
(b) Original image



(c) Noisy image



(d) Denoising via the stack filter



(e) Denoising via the original FDS

Figure 21: Denoising properties of the stack filter and of the original FDS when $p = 50\%$.

4 Gaining on CPU Time

The main drawback of our algorithm lies in CPU time, as it is N times slower with respect to the function actually applied to the arrays v_l, where $N = 255$ is the number of level sets. To fix the ideas, the computational time needed to evolve a 500×500 pixels RGB image when $R = 5$ and $\Delta t = 0.5$ is approximately 4 minutes. The test has been performed on a laptop with processor *Intel Core i5 430 M* (2.27 GHz) and 4 Gb of *RAM*.

4.1 Parallelization

To improve the timing performances, we have to resort to some forms of *parallel computing*. In plain words, the idea is to save on computational time carrying out simultaneously several operations.

In particular, we parallelize loops in order to distribute the data across different computing nodes, assigning to different threads entire sections of code by means of the OpenMP[2] application programming interface.

In order to minimize the number of *fork-join* operations and avoid data racing, the best solution is to parallelize when dealing with level sets which are in sufficient number (256) to exploit the advantages of a machine with several independent processing units. The only problematic part lies in the final reconstruction by superposition principle, which is included in a *critical* directive.

Instead, if we parallelize at a lower level, for instance when we apply the finite difference scheme to the pixels of the function v_l, the processing time for the evolution of the usual RGB image at a normalized scale of 5 goes from 129 to 115 seconds, depending on the type of allocation used (better *dynamic* than *static*) and on the choice of the amount of iterations to be assigned each time (1000 is the optimal number). On the other hand, parallelizing the *for* cycle that iterates on the level sets, we have a processing time of 95 seconds if the schedule is dynamic and the amount of iterations assigned each time is ≤ 8. It is useless to parallelize at the channels level, as we obtain no improvement for B/W images and we have only 3 iterations when processing RGB pictures.

4.2 Reduction of Gray Levels

If, for instance, our machine has a single processing unit or we have to perform tests with high values of normalized scale and, above all, we do not need quantitative results but only visual outcomes, a possibility to reduce even further processing time is to consider only a part of the 256 level sets of the digital input image. Such a reduction of gray levels can be performed in multiple ways.

Biased uniform reduction. We fix the number of levels needed, say k, therefore the step scale is $m = 256/k$, which means that for all n in $\{0, 1, \ldots, k-1\}$ each value belonging to $\{n \cdot m, n \cdot m + 1, \ldots, (n+1) \cdot m - 1\}$ will be set to $n \cdot m$. This implies an error of approximation which in the worst case is $m - 1$ and whose mean is $(m-1)/2$.

Unbiased uniform reduction. If in the previous case we quantized level sets only by shortcoming, in order to reduce the mean and worst case error we can approximate both by shortcoming and excess, dividing into two parts the interval of eliminated gray values. Nevertheless, if k is a power of 2, the number of eliminated level sets between two preserved ones is odd, which means that the approximation will not be correctly balanced, i.e. the mean error will not be 0. Instead, if we decide an even width of the interval of discarded values, say w, then the mean error is 0 and the worst case error is $w/2$. The preserved gray levels have the form $m \cdot (w+1)$. A problem arises when 255 has to be approximated by excess since pixel values go out from

[2] http://openmp.org/

the range $\{0, 1, \ldots, 255\}$. Such an eventuality occurs when $255 > g_{\text{last}} + w/2$, where g_{last} is the highest gray level of the quantized image. If $w \leq 20$, $w = 12$ is the only case in which this undesirable effect takes place.

Elimination of the smallest isolevel sets. A completely different approach consists in discarding the less relevant gray values, i.e. those attained only by a small number of pixels. However if we fix a threshold on the cardinality of isolevel sets, we cannot know *a priori* the number of levels of the quantized image and therefore the improvement in computational time achieved. To solve this problem, we can decide to eliminate, for instance, the k less relevant level sets, in order to obtain a gain of $256/(256-k)$ times. Nevertheless, a relevant disadvantage of such an approximation technique lies in the fact that real information could be hidden in those gray levels which are assumed by a small amount of pixels.

In the final online version of the algorithm no reduction of gray levels is performed, as the conditions on CPU time have been satisfied with parallelization techniques.

5 Examples

Remark: the normalized scale R is always given with respect to the original image. To find out the normalized scale for the zoomed detail, the value of R must be multiplied by the zooming factor.

5.1 Geometrical Figures

Our analysis starts with the simplest possible pictures, i.e. those taken from the geometrical world: straight and bent lines, circles and polygons.

As figures 22 and 23 clearly show, the original FDS and the stack filter share the complete elimination of the staircase effect due to pixelization. In addition, under a mean curvature evolution, straight contours practically remain still, while angles gradually bend, tending to become arcs of circles. When two or more figures overlap, the finite difference scheme seems to mix them, forming an undefined union and making difficult to recognize the boundaries. Nevertheless, the contours of images processed by the stack filter are precise and well defined, as the blurring produced by the scale space is attenuated by the application of a *contrast invariant* algorithm. In the RGB example, the differences between the two algorithms are less evident, as a higher number of gray level is present in the input. This makes the transition of the stack filter evolution between black lines and colored slices more gradual.

After that, we focus on the evolutions of two pictures which show how the stack filter behaves when dealing with two important disadvantages typical of the basic finite difference scheme: the disappearance of curves in space with small thickness and the shading of contours noticed when the normalized scale assumes high values. As we can see in figure 24, the adoption of the stack filter does not prevent thin ribbons from collapsing. Indeed, since the input image is binary, applying such an algorithm instead of the original FDS means only to threshold the result with $\lambda = 127.5$. As a result, the incapability of pixel-based schemes to deal with narrow contours turns out to be enhanced. In other words, the elliptical contour is not thick enough to be evolved properly by the numerical algorithm. On the other hand, the stack filter evolution represented in figure 25 appears decidedly better than that obtained using the original FDS and perfectly coherent with theoretical results. According to Grayson's theorem [9], an embedded curve first becomes convex and then shrinks tending asymptotically to a circle.

(a) Original image (b) 5× zoomed detail

(c) Stack Filter, $R = 8$ (d) MCM, $R = 8$

Figure 22: Intersection between a hexagon and a circle.

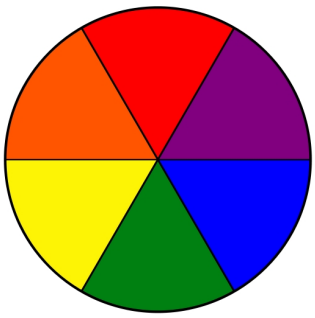
5.2 Everyday Pictures

First of all, it is important to remark that finite difference schemes simulating the MCM are to be considered essentially as an introductory step preceding further numerical analysis or feature extraction. The aim of this preprocessing part is to eliminate the low scale details which are not considered meaningful and free the images from their aliasing, JPEG, and noise artifacts. Collaterally, it is also possible to use these schemes to obtain an immediate improvement in the quality of pictures. In the examples that follow, we are going to present a brief comparison among these image processing algorithms:

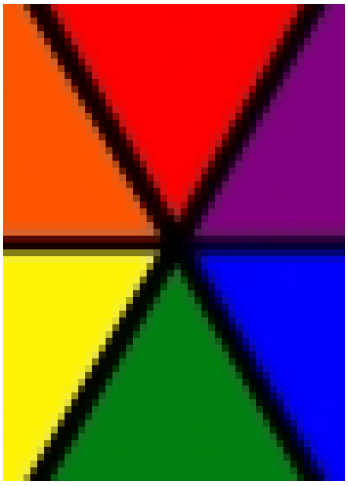
- **original FDS** for the MCM [8];
- proposed **Stack filter** based on a finite difference scheme for the MCM;
- **Level Lines Shortening**, which adopts a completely different approach, as it is not *pixel-based*, but samples all image level lines at a sub-pixel fine resolution smoothing them separately [10].

5.2.1 Fingerprints

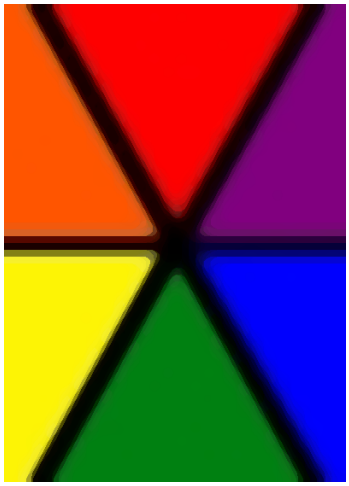
An attempt to restore the original smoothness of fingerprint ridges is to be found in figure 26. Mean curvature evolutions eliminate the effects of pixelization in this 5× zoomed detail. As lots of intermediate gray values appear in the input image, the transition between black and white cannot be sharp even if we use the stack filter. Consequently, the differences between the two FDS evolutions are scarce. The image processed by the LLS algorithm is more regular, but some of the fingerprint contours still seem to melt, because of the low quality of the original picture.



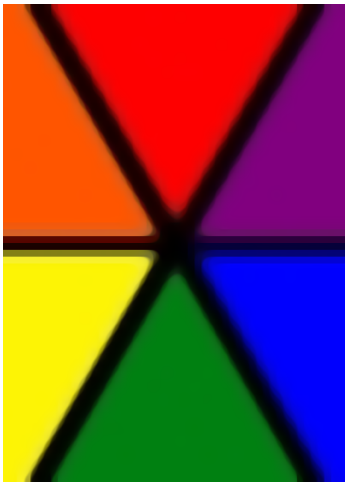
(a) Original image



(b) 8× zoomed detail



(c) Stack Filter, $R = 7$



(d) MCM, $R = 7$

Figure 23: Center of a multicolor circle.

(a) Stack Filter, $R = 1$ (b) MCM, $R = 1$

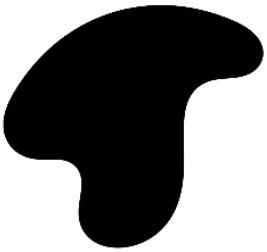
(c) Stack Filter, $R = 2$ (d) MCM, $R = 2$

(e) Stack Filter, $R = 3$ (f) MCM, $R = 3$

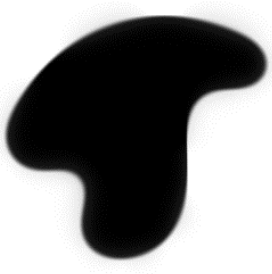
(g) Stack Filter, $R = 4$ (h) MCM, $R = 4$

(i) Stack Filter, $R = 5$ (j) MCM, $R = 5$

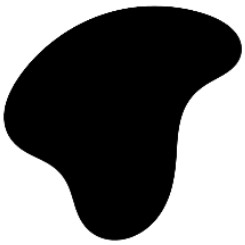
Figure 24: Evolution of a thin ribbon by the stack filter and the original FDS.



(a) Stack Filter, $R = 20$

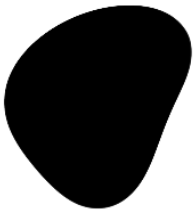


(b) MCM, $R = 20$

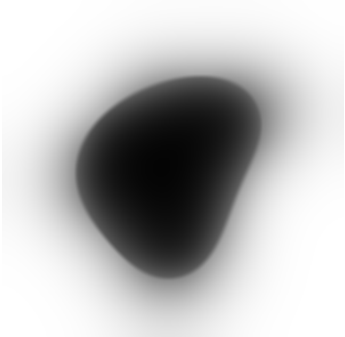


(c) Stack Filter, $R = 30$

(d) MCM, $R = 30$

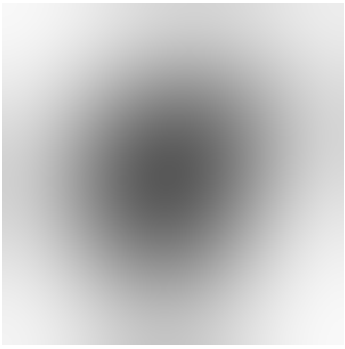


(e) Stack Filter, $R = 50$



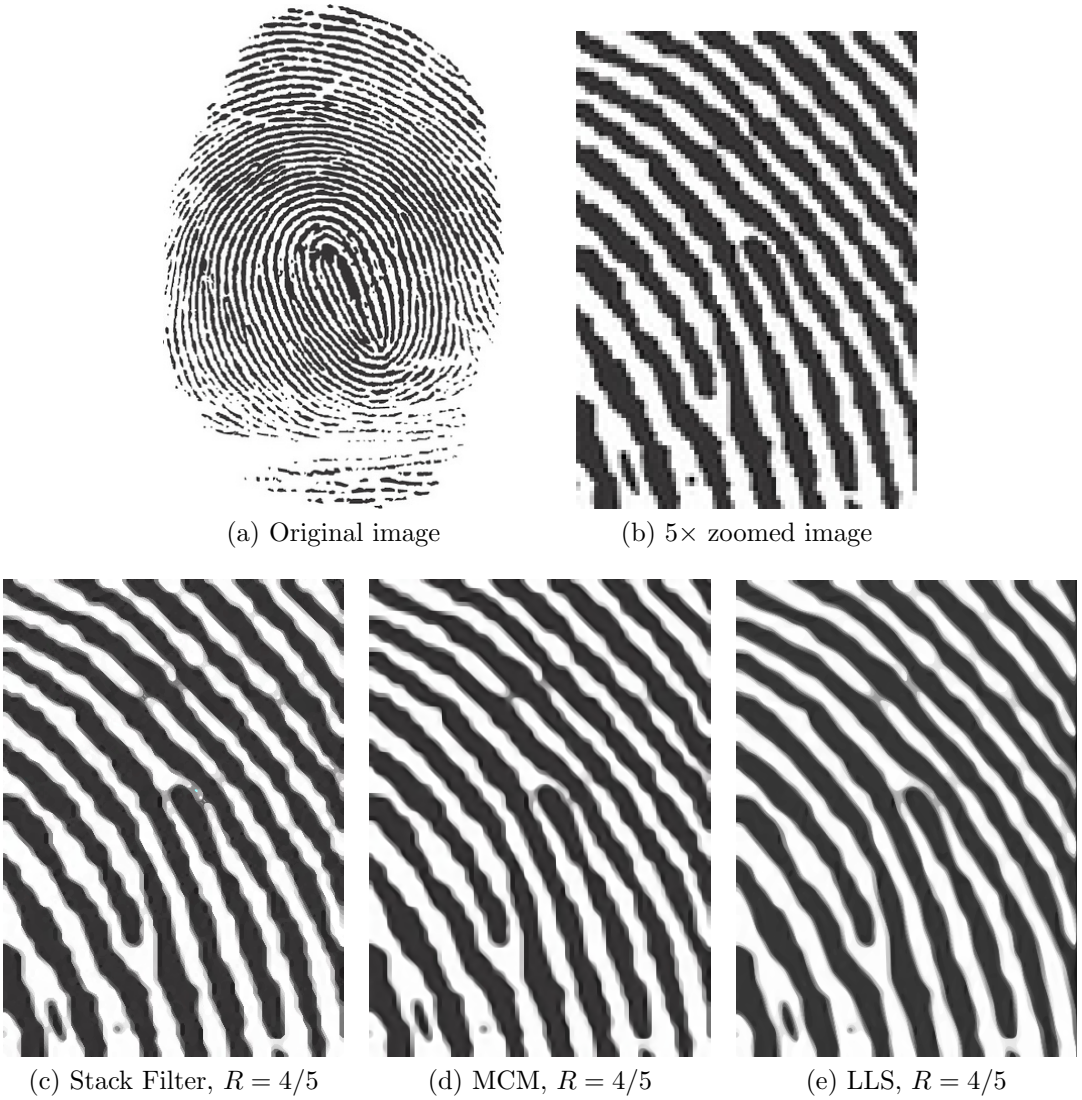
(f) MCM, $R = 50$

(g) Stack Filter, $R = 90$



(h) MCM, $R = 90$

Figure 25: Evolution of the interior of an embedded curve.



(a) Original image (b) 5× zoomed image

(c) Stack Filter, $R = 4/5$ (d) MCM, $R = 4/5$ (e) LLS, $R = 4/5$

Figure 26: Fingerprints.

5.2.2 Flower and Text Corrupted by JPEG Compression

The well known JPEG coding is a lossy technique of image compression that may yield artifacts, such as Gibbs's oscillations, staircase noise or checker boarding. An algorithm simulating the MCM can be used to reduce these unwanted effects. In this case the differences between the stack filter, the original FDS and the level lines shortening are not easily noticeable (see figure 27).

5.3 Denoising Features

We are not interested in a quantitative analysis, so there is no need in taking simple geometrical examples. Therefore we have processed an everyday picture containing details at various scales.

Following the usual procedure, we add to the inputs different percentages of *salt and pepper noise* and then we evolve these noisy images with the stack filter and the original finite difference scheme at various normalized scales, looking for the optimal trade off between denoising and smoothing of small scale particles. The choice of the best value of R is based essentially on visual standards, so it is somewhat arbitrary, but useful to give an intuitive idea of the denoising properties achieved by the stack filter.

5.3.1 B/W example

When $p = 10\%$ (see figure 28), we obtain a good denoising also using the basic scheme, even if a bigger value of normalized scale is needed (2 against 1.5). This reflects on the higher amount of well defined details present in the output picture on the left (notice, for instance, the texture of the Eiffel tower or the trees in the background).

As the percentage of noise increases ($p = 30\%$), the application of the basic FDS yields decidedly worse results, while the reconstruction by means of the stack filter is still visually excellent. As shown by figure 29, the noise has been completely eliminated and the fundamental informative content of the picture has been preserved.

If $p = 50\%$ (see figure 30), the basic finite difference scheme fails completely, while the application of the stack filter yields still good results: the people walking in the street are disappearing, but the geometry of the scene, with the Eiffel Tower, the road, the lamps and the trees in the background is left untouched.

5.3.2 RGB Example

If the percentage of noise is too low ($p = 10\%$), we have difficulty in noticing differences between the two mean curvature evolutions. Nevertheless, when employing the stack filter, the branches and leaves on the two sides of the image appear more definite and precise, as we can see in figure 31. Indeed, if we consider the original finite difference scheme, the normalized scale needed to eliminate the uniform noise is to be doubled ($R = 3$ against $R = 1.5$).

In the intermediate example of figure 32 ($p = 30\%$) we notice a remarkable difference between the performances of the two algorithms: the basic mean curvature evolution creates an evident blurring and cannot eliminate completely the noise; on the other hand, the stack filter returns an output picture extremely similar to the original one.

When $p = 50\%$ (see figure 33), the denoising process is practically impossible for the basic finite difference scheme. On the contrary, acting on level sets instead of considering directly input images, allows the stack filter algorithm to maintain the essential features of the original picture: fence, grass and trees are clearly recognizable in the lower left figure. Inevitably, the output appears more blurred with respect to the previous examples, as a higher value of R is required for the denoising process to be successful.

(a) Original image

(b) Stack Filter, $R = 4$

(c) MCM, $R = 4$

(d) LLS, $R = 4$

Figure 27: Flower and text corrupted by JPEG compression.

(a) Input image

(b) Noisy image

(c) Denoising with the stack filter ($R = 1.5$)

(d) Denoising with the original FDS ($R = 2$)

Figure 28: Use of the stack filter and of the original FDS to eliminate uniform noise from a B/W image when $p = 10\%$.

(a) Input image

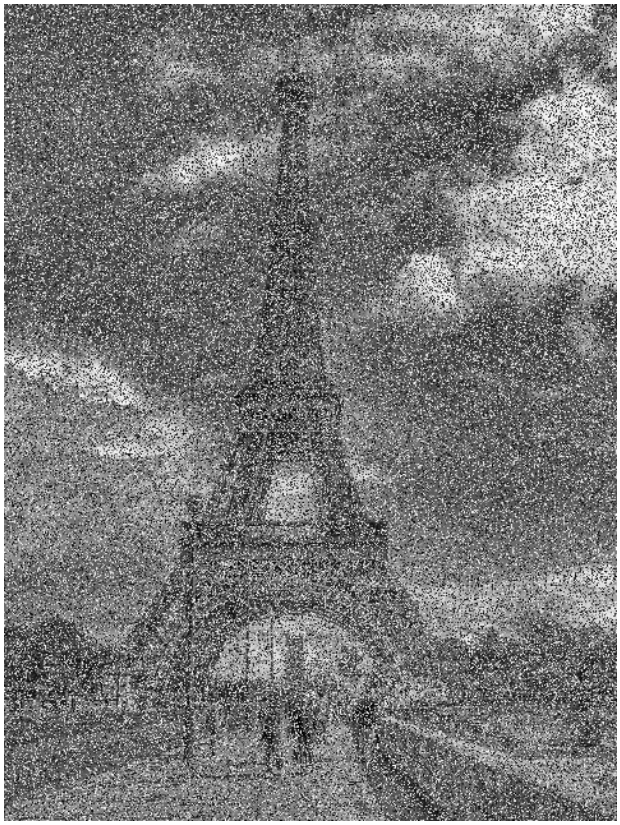
(b) Noisy image

(c) Denoising with the stack filter ($R = 2.5$)

(d) Denoising with the original FDS ($R = 3$)

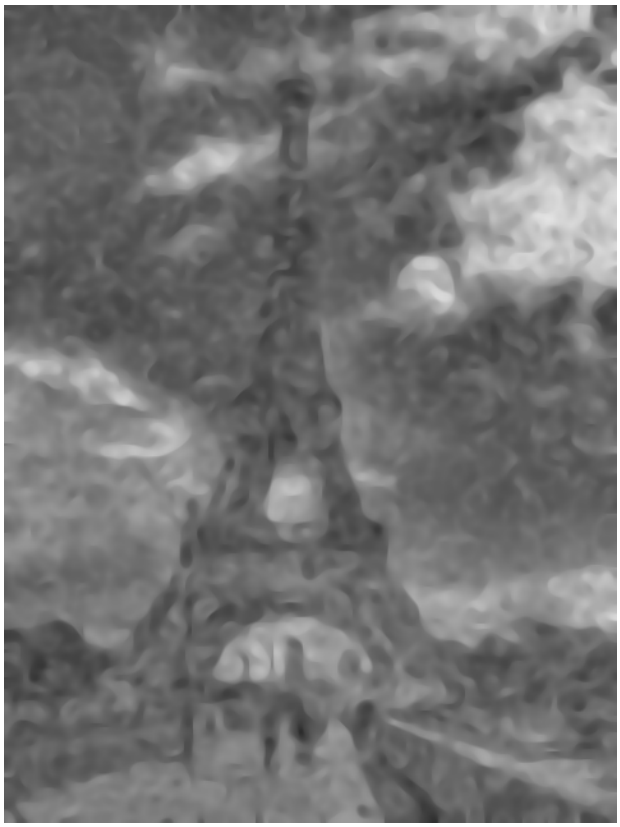
Figure 29: Use of the stack filter and of the original FDS to eliminate uniform noise from a B/W image when $p = 30\%$.

(a) Input image



(b) Noisy image

(c) Denoising with the stack filter ($R = 3$)



(d) Denoising with the original FDS ($R = 5$)

Figure 30: Use of the stack filter and of the original FDS to eliminate uniform noise from a B/W image when $p = 50\%$.

(a) Input image

(b) Noisy image

(c) Denoising with the stack filter ($R = 1.5$)

(d) Denoising with the original FDS ($R = 3$)

Figure 31: Use of the stack filter and of the original FDS to eliminate uniform noise from an RGB image when $p = 10\%$.

(a) Input image

(b) Noisy image

(c) Denoising with the stack filter ($R = 2.5$)

(d) Denoising with the original FDS ($R = 4$)

Figure 32: Use of the stack filter and of the original FDS to eliminate uniform noise from a RGB image when $p = 30\%$.

(a) Input image

(b) Noisy image

(c) Denoising with the stack filter ($R = 4$)

(d) Denoising with the original FDS ($R = 5$)

Figure 33: Use of the stack filter and of the original FDS to eliminate uniform noise from a RGB when $p = 50\%$.

5.4 Pros, Cons and Future Work

The implementation of a stack filter based on finite difference schemes for the Mean Curvature Motion equation allows us to obtain requisites of regularity, even when adopting the highest possible value for time step ($\Delta t = 0.5$).

The algorithm is, by definition, *stable*, it satisfies the *semigroup property* and is *contrast invariant* with good approximation, as it does not act directly on the input image, but on the characteristic functions of its level sets, conveniently scaled by a multiplicative factor. Pixel values cannot exceed the initial range [0, 255] and no further gray level, i.e. no further spurious information, can appear in the output. Unfortunately, a full contrast invariance is not achievable because of the restriction due to quantization, as the contrast changes may not be exactly representable on quantized images. Anyway, since an image is regarded as an array of integers $\in \{0, 1, \ldots, 255\}$, the pixelization is more a problem of image representation than an algorithmic issue. In addition, even if stack filters are below *state of the art* denoising methods, the proposed algorithms presents some interesting denoising properties that has been successfully tested with percentages of uniform noise up to 50%.

The main drawback of the numerical scheme lies in a notable computational time, as the amount of operations performed has to be multiplied by the number of level sets, i.e. by 255. To overcome such a difficulty, on the one hand we have increased the value of time step as much as possible in order to reduce the number of iterations, without noticing a considerable worsening in performances. On the other hand we have resorted to some techniques of code acceleration, focusing in particular on *vectorization* and *multiprocessing programming*. If a further gain in CPU time is required, a possibility is to resort to a completely different approach, making use of fast marching algorithms to implement the stack filter [12].

Nevertheless, since the stack filter is based on a weighted linear combination of current pixel's first neighbors, it inevitably shares some common defects of FDSes, such as, for example, the *lack of monotonicity*. Furthermore, the algorithm is *pixel-based* and therefore blind to small curvatures, as when we evolve sets sampled on a fixed grid, boundaries either jump by a positive integer number of pixels or do not move at all. An example of continuous, grid independent evolution of a digital image by curvature motion is the *Level Lines Shortening* [10].

Acknowledgment

I would like to thank the reviewer Enric Meinhardt whose remarks and fruitful comments have improved the present article.

Credits

 LucasKrech.com, CC-BY-SA[3]

 Glenn J. Mason, CC-BY-SA[4]

 Wikipedia user - Lulu of the Lotus-Eaters, CC-BY-SA[5]

 Pier-Luc Bergeron, CC-BY-SA[6]

[3] http://lucaskrech.com/blog/index.php/tag/stanley-mccandless
[4] http://www.flickr.com/photos/glennji/3558118429/
[5] http://en.wikipedia.org/wiki/File:Sego_lily_cm-150.jpg
[6] http://www.flickr.com/photos/burgtender/3531915799/in/photostream/


Rodney Burton, CC-BY-SA[7]

References

[1] M. Wertheimer, *Untersuchungen zur Lehre von der Gestalt, II*, Psychologische Forschung, 4 (1923), no. 1, 301–350, http://dx.doi.org/10.1007/BF00410640.

[2] P. D. Wendt, E. J. Coyle and N. C. Gallagher, *Stack Filters*, IEEE Transactions on Acoustics, Speech and Signal Processing, 34 (1986), 898–911, http://dx.doi.org/10.1109/TASSP.1986.1164871.

[3] P. Maragos, and R. W. Schafer, *Morphological Filters - Part II: Their Relations to Median, Order-Statistic, and Stack Filters*, IEEE Transactions on Acoustics, Speech and Signal Processing, 35 (1987), no. 8, 1170–1184 http://dx.doi.org/10.1109/TASSP.1987.1165254.

[4] J. Serra, *Introduction to mathematical morphology*, Computer Vision, Graphics and Image Processing, 35 (1986), 283–305, http://dx.doi.org/10.1016/0734-189X(86)90002-2.

[5] S. Osher, and J. A. Sethian, *Fronts Propagating with Curvature Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations*, Journal of Computational Physics, 79, 12–49, 1988, http://dx.doi.org/10.1016/0021-9991(88)90002-2.

[6] F. Guichard, J-M. Morel, and R. Ryan *Contrast invariant image analysis and PDE's*, to appear, http://mw.cmla.ens-cachan.fr/~morel/JMMBookOct04.pdf

[7] L. Alvarez, and J.M. Morel, *Formalization and computational aspects of image analysis*, Acta Numerica, 1994, 1–59, http://dx.doi.org/10.1017/S0962492900002415.

[8] A. Ciomaga, and M. Mondelli, *Finite Difference Schemes for MCM and AMSS*, Image Processing On Line, 2011, http://dx.doi.org/10.5201/ipol.2011.cm_fds.

[9] M.A. Grayson, *The heat equation shrinks embedded plane curves to round points*, J. Differential Geom., 26 (1987), no. 2, 285–314, http://projecteuclid.org/euclid.jdg/1214441371.

[10] A. Ciomaga, P. Monasse, and J-M. Morel, *Image Visualization and Restoration by Curvature Motions*, SIAM Multiscale Modeling and Simulation, 9 (2011), 834–871, http://dx.doi.org/10.1137/100791269.

[11] A. Ciomaga, L. Moisan, P. Monasse, and J-M. Morel, *Image Curvature Microscope*, Image Processing On Line, to appear, http://www.ipol.im/pub/pre/H4/.

[12] J. A. Sethian, *Level Set Methods and Fast Marching Methods. Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*, Cambridge University Press, 1999, ISSN: 02635747.

[7] http://www.geograph.org.uk/photo/64921