**Journal of Real-Time Image Processing manuscript No.**
(will be inserted by the editor)

**Andrés Frías-Velázquez · Josep Ramon Morros · Mario García · Wilfried Philips**

# Hierarchical stack filtering: a bitplane-based algorithm for massively parallel processors

Received: date / Accepted: date

**Abstract** With the development of novel parallel architectures for image processing, the implementation of well-known image operators needs to be reformulated to take advantage of the so-called massive parallelism. In this work, we propose a general algorithm that implements a large class of nonlinear filters, called stack filters, with a 2D-array processor. The proposed method consists of decomposing an image into bitplanes with the bitwise decomposition, and then process every bitplane hierarchically. The filtered image is reconstructed by simply stacking the filtered bitplanes according to their order of significance. Owing to its hierarchical structure, our algorithm allows us to trade-off between image quality and processing time, and to significantly reduce the computation time of low-entropy images. Also, experimental tests show that the processing time of our method is substantially lower than that of classical methods when using large structuring elements. All these features are of interest to a variety of real-time applications based on morphological operations such as video segmentation and video enhancement.

**Keywords** Stack filters · Array processors · Bitwise decomposition · Morphological operators · Smart camera

## 1 Introduction

Platform-aware implementations of basic image operations are essential to build efficient applications on massively-parallel processors. In this paper, we propose a novel implementation of *stack filters* using a 2D-array processor. These filters compose a large class of nonlinear filters that return local rank statistics of an image. Classic filters like median, erosion, and dilation belong to this class of filters; they are widely used in image denoising, pattern recognition, and object segmentation. The naive implementation of these filters consists of first sorting the input data, and then selecting the desired ranked element. Alternative implementations [2–5, 8, 17] decompose the input data into binary signals, and then apply a simple binary filter to these signals, thus avoiding expensive sorting.

The baseline implementation of stack filters via binary decomposition consists of three steps: First, the input data is decomposed into binary signals by thresholding the data for every level of

The final publication is available at Springer via http://dx.doi.org/10.1007/s11554-017-0681-8

A. Frías-Velázquez · W. Philips
Ghent University
Department of Telecommunications and Information Processing (TELIN-IPI-IMEC)
Sint-Pietersnieuwstraat 41-B
9000 Gent – Belgium
E-mail: Andres.FriasVelazquez@ugent.be

J.R. Morros · M. García
Technical University of Catalonia
Jordi Girona 1-3 – 08034 Barcelona – Spain

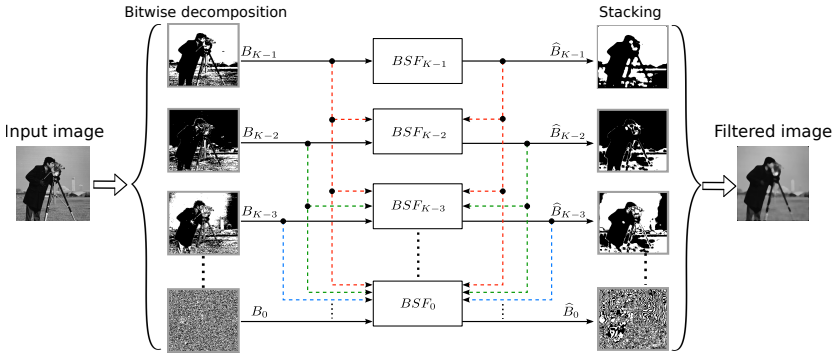



**Fig. 1** Block diagram of the Bitplane Stack Filter (BSF) algorithm.

the data range. This step is known as Threshold Decomposition (TD) [18]. Second, a binary filter is applied to the binary signals of each level of decomposition. Third, the output is obtained by summing up the filtered data of all levels of decomposition.

In the literature we can find several implementations of stack filters using different architectures. For instance, Chakrabarti and Lucke [3] proposed an implementation that applies the threshold decomposition to each input sample in a serial fashion, and then filters the binary words of all levels of decomposition in parallel. On the other hand, Gevorkian et al. [8] proposed an implementation that decomposes all input samples in parallel using Fibonacci $p$-codes to reduce the number of decomposing levels, and then filters the binary words of every level of decomposition in a serial fashion. Avedillo et al. [2] proposed a binary decomposition based on an ordering matrix resulting from the comparison between input samples. In this way, the number of decomposing levels not longer depends on the gray-level range of the input data, but on the number of samples. After applying such decomposition, the binary words of all levels are filtered in parallel. Meanwhile, Mertzios and Tsirikolias [11] proposed the use of coordinated logic filters to implement classical morphological operations for applications that range from noise removal to edge extraction. These kind of filters also rely on the bitwise decomposition, and allow to compute a filter approximation by only processing the most significant bits to improve the computation speed. However, they do not easily generalize to other stack filters. Finally, Spiliotis and Boutalis [16] also exploited the idea of separating an image in bitplanes by using the bitwise decomposition, although their goal was to accelerate the computation of image moments. This strategy enabled them to save significant computation time while achieving a high accuracy of the computed moments.

In general, stack filter implementations sequentially process a pixel neighborhood that slides over the image using the TD method. In this paper, we follow a different approach by using the image bitwise decomposition with a bitplane architecture. The bitwise decomposition allows us to filter an image hierarchically, while the bitplane architecture helps us to exploit the very fast processing of both logical operations and binary morphological filters in a Focal Plane Processor (FPP) [20]. Primarily, the proposed method consists of decomposing the input image using the bitwise decomposition. In this way, an 8-bit image is decomposed into 8 bitplanes arranged from the most to the least significant bit. Then, we filter every bitplane hierarchically by using our stack filter algorithm. Finally, the filtered image is assembled by simply stacking the filtered bitplanes according to their order of significance. A block diagram that outlines the proposed method is presented in Fig. 1. To test our approach, we implemented it in the Eye-RIS vision system [12], which is a smart camera that contains a mixed-signal FPP called Q-Eye.

It is worth to remark that an early development of the method described above was presented in our previous paper [6]. This prototype method also benefits from the bitwise decomposition and the bitplane architecture; however, it was specifically tailored to the erosion filter. In another paper [7], we developed a preliminary version of our stack filter algorithm. In this paper, we laid the groundwork of the current approach by outlining the basics of our algorithm, and by informally introducing the relationship between bitwise and threshold decomposition. In the present work, we include several contributions apart from extending and detailing the description of our algorithm: *(i)* We formally derive the Boolean functions that relate the bitwise decomposition and the threshold decomposition.

*(ii)* We derive two optimization criteria that reduce the processing time of our algorithm depending on the gray-level distribution of the image and the size of the structuring element. Thanks to these criteria, we can speed up our algorithm not only by discarding the computation of the less significant bitplanes, but also by actively discarding superfluous computations without compromising the image fidelity. *(iii)* We present a complexity analysis and a new set of simulations that test key features of our algorithm.

The rest of the paper is organized as follows: in Sect. 2, we derive the equations that relate the threshold decomposition and bitwise decomposition, which are the basis of our approach. Meanwhile, in Sect. 3, we formally describe the proposed stack filter algorithm. The architecture of the massively-parallel processor where we demonstrate our algorithm is described in Sect. 4. In Sect. 5, we present some experimental results. Finally, the conclusions of this work are stated in Sect. 6.

## 2 Relationship between bitwise and threshold decomposition

In this section we will derive expressions to obtain the threshold decomposition of an image from its bitwise decomposition, and vice versa. These expressions will allow us to later propose a stack filter algorithm implemented with the image bitwise decomposition.

Let $I(n,m)$ be a digital image with spatial indices $n, m \in \mathbb{Z}$, and $L$ grayscale levels. The image can be decomposed into $L$ threshold bitplanes with the threshold decomposition function $\mathcal{T}$ as follows:

$$T_l(n,m) = \mathcal{T}_l(I(n,m)) = \begin{cases} 1, \text{ if } I(n,m) \geq l, \\ 0, \text{ if } I(n,m) < l, \end{cases} \tag{1}$$

where $T$ represents the threshold bitplane at the $l$ threshold level. On the other hand, the bitwise decomposition function $\mathcal{B}$ splits an image into $K = \lceil \log_2(L) \rceil$ bitwise bitplanes as follows:

$$B_k(n,m) = \mathcal{B}_k(I(n,m)) = \left\lfloor \frac{I(n,m)}{2^k} \right\rfloor \bmod 2, \tag{2}$$

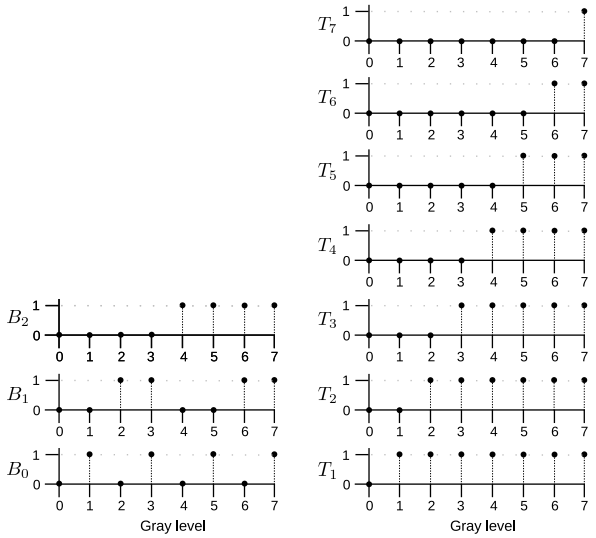
where $B$ represents the bitwise bitplane at the $k$ bit of significance. To recompose a multilevel image from its threshold decomposition we evaluate:

$$I(n,m) = \sum_{l=1}^{L} T_l(n,m), \tag{3}$$

whereas from its bitwise decomposition we evaluate:

$$I(n,m) = \sum_{k=0}^{K-1} B_k(n,m) 2^k. \tag{4}$$

Note that monotone functions like the threshold function (1) preserve the order of the decomposed data, which is key to implement stack filters using binary filters [18]. By contrast, non-monotone functions like the bitwise decomposition do *not* satisfy this order preserving property. In Fig. 2 we can observe the differences between these decomposition functions. Although the bitwise decomposition does not fulfill the monotonicity property, it possesses two important advantages over the threshold decomposition: 1. The bitwise decomposition is obtained by simply reading the bit fields of the data, whereas the TD requires many comparisons. 2. The image information is conveyed hierarchically, which allow to reduce the processing time of both the exact filtered output and its approximation by removing the computation of the least significant bitplanes.




(a) (b)

**Fig. 2** Examples using: (a) bitwise, and (b) threshold decompositions. Note that in (a) $B_0$ and $B_1$ are not monotone, whereas in (b) all functions are monotone

### 2.1 From threshold to bitwise decomposition

To derive an expression of the bitwise decomposition from the threshold decomposition, we manipulate the definition of the bitwise decomposition in (2) as described in Appendix A:

$$B_k = \bigvee_{j=0}^{N_{k+1}} b_{j,k}, \quad b_{j,k} = T_{(2j+1)2^k} \wedge \neg T_{(j+1)2^{k+1}} \tag{5}$$

where $b_{j,k}$ is the $j$-th partial output of the $k$-th bitplane and $N_k = 2^{K-k} - 1$. For simplicity, the spatial indices $n$ and $m$ of the bitplanes have been omitted. From (5) we conclude that every $k$-th bitplane can be reconstructed with $2^{K-k}$ threshold bitplanes. Thus, the number of thresholdings increases exponentially as the bit of significance decreases. This means that coarse information of the image can be generated with a couple thresholdings, whereas fine details require many of them.

### 2.2 From bitwise to threshold decomposition

To compute the threshold decomposition from the bitwise decomposition (the inverse relationship of (5)), we prove in Appendix B that the threshold bitplane at level $l$ can be reconstructed with the $q$ most significant bitplanes as follows:

$$T_l = (B_{K-1} \underset{\lambda_{K-1}}{\bowtie} (B_{K-2} \underset{\lambda_{K-2}}{\bowtie} \cdots (B_{K-q+1} \underset{\lambda_{K-q+1}}{\bowtie} B_{K-q}))) \tag{6}$$

where $(\lambda_{K-1}\lambda_{K-2} \ldots \lambda_1\lambda_0)_2$ is the binary representation of $l$, and every bit controls the following conditional expression

$$\underset{x}{\bowtie} = \begin{cases} \vee, \text{ if } x = 0, \\ \wedge, \text{ if } x = 1. \end{cases} \tag{7}$$

The number of the most significant bitplanes is $q = K - z$, where $z$ is the number of least significant bits of $l$ set to zero. For instance, to compute $T_{168}$, we express 168 as $(\texttt{10101000})_2$, which yields $z = 3$ and $q = 5$. By analyzing (6), we conclude that the threshold decomposition can be generated from the bitwise decomposition using simple Boolean operations.

## 3 The bitplane stack filter algorithm

In this section, we will present two algorithms to perform the bitplane stack filtering. In Sect. 3.1 we describe the general procedure to filter an image from the most to the least significant bitplane, while in Sect. 3.2 we describe optimization criteria that depend on the image distribution, and that reduce the computation of the least significant bitplanes. Both algorithms return the exact filter output, but the computation time of the optimized algorithm is usually faster depending on the image distribution.

In general, the filtered response $\widehat{B}_k$ of a bitwise bitplane $\mathcal{B}_k(I)$ is defined by $\widehat{B}_k = \Psi_{s,f}[\mathcal{B}_k(I)]$, where $\Psi$ is a binary morphological filter with a structuring element $s$, and a positive boolean function $f$. Therefore, in light of (4), the filtered response $I^*$ of a grayscale image $I$ using the bitwise decomposition is obtained as follows

$$I^* = \sum_{k=0}^{K-1} \widehat{B}_k \, 2^k = \sum_{k=0}^{K-1} \Psi_{s,f}[\mathcal{B}_k(I)] \, 2^k \tag{8}$$

By using (5), the filtering of every bitwise bitplane can be derived from threshold bitplanes that are filtered with the binary stack filter $\Phi$. Thus, it follows that:

$$\widehat{B}_k = \bigvee_{j=0}^{N_{k+1}} \widehat{b}_{j,k}, \text{ where } \widehat{b}_{j,k} = \Phi_{s,f}\left[T_{u_{j,k}}(\mathbf{B})\right] \wedge \neg \Phi_{s,f}\left[T_{v_{j,k}}(\mathbf{B})\right] \tag{9}$$

and the threshold bitplanes at levels $u_{j,k} = (2j+1)2^k$ and $v_{j,k} = (j+1)2^{k+1}$ are generated from the set of bitwise bitplanes $\mathbf{B} = (B_0, ..., B_{K-1})$ using (6). Finally, we can express a filtered threshold bitplane as $\widetilde{T} = \Phi_{s,f}[T]$ to later rewrite (9) more compactly as follows:

$$\widehat{B}_k = \bigvee_{j=0}^{N_{k+1}} \widehat{b}_{j,k}, \text{ where } \widehat{b}_{j,k} = \widetilde{T}_{u_{j,k}} \wedge \neg \widetilde{T}_{v_{j,k}} \tag{10}$$

### 3.1 The proposed algorithm

Our algorithm consists of filtering the bitwise bitplanes of an image using (10). Its architecture is described in Fig. 1. Each processing block, denoted as $BSF_k$, represents the implementation of (10) for the $k$-th bitplane of significance. The algorithm runs from the most to the least significant bitplane because each processing block depends on bitplanes of greater or equal significance than the $k$-th block. This hierarchization is related to the number of bitwise bitplanes required to generate $T_u$ and $T_v$ using (6). For instance, to compute $T_u$ using bitwise bitplanes, we need the $p$ most significant bitplanes such that $p = K - k$. On the other hand, to compute $T_v$ we need the $q$ most significant bitplanes such that $q \leq K - k - 1$. We conclude that $q \leq p - 1$, which suggests that $T_v$ can be determined with at least one bitwise bitplane less than $T_u$. This allows us to compute $\widetilde{T}_v$ directly from output bitplanes of greater significance than the level to be processed. Thus, we can see in Fig. 1 that the output bitplanes are eventually input into processing blocks of lower significance. For $\widetilde{T}_u$, $T_u$ is first computed using the input bitwise bitplanes, and then the binary stack filter is applied.

Algorithm 1 details the bitplane stack filtering. It is composed of a nested for-loop. The outer loop controls the order in which the bitplanes are processed (from most to least significant bitplanes). The inner loop performs the filtering of the $k$-th bitwise bitplane by computing and accumulating the $2^{K-k-1}$ partial outputs as stated in (10).

### 3.2 Optimized algorithm

The optimized algorithm follows a similar structure as the general algorithm. The main difference is that we include criteria that avoid the computation of partial outputs as the bitplanes are processed

**Algorithm 1** StackFilter($(B_0,\ldots,B_{K-1})$,$s$,$f$)

$\widehat{B}_K \leftarrow \mathbf{0}$
**for** $k = (K-1)$ down to 0 **do**
  $\widehat{B}_k \leftarrow \mathbf{0}$
  **for** $j = 0$ to $N_{k+1}$ **do**
    $u \leftarrow (2j+1)2^k$
    $v \leftarrow (j+1)2^{k+1}$
    $(\mu_{K-1}\mu_{K-2}\ldots\mu_{K-p}\mathtt{0}\ldots\mathtt{0})_2 \leftarrow (u)_{10}$
    $(\nu_K\nu_{K-1}\ldots\nu_{K-q}\mathtt{0}\ldots\mathtt{0})_2 \leftarrow (v)_{10}$
    $T_u \leftarrow (B_{K-1} \underset{\mu_{K-1}}{\bowtie} (B_{K-2} \underset{\mu_{K-2}}{\bowtie} \cdots (B_{K-p+1} \underset{\mu_{K-p+1}}{\bowtie} B_{K-p}))) (B_{K-p+1} \underset{\mu_{K-p+1}}{\bowtie} B_{K-p}))$
    $\widetilde{T}_u \leftarrow \Phi_{s,f}[T_u]$
    $\widetilde{T}_v \leftarrow (\widehat{B}_K \underset{\nu_K}{\bowtie} (\widehat{B}_{K-1} \underset{\nu_{K-1}}{\bowtie} \cdots (\widehat{B}_{K-q+1} \underset{\nu_{K-q+1}}{\bowtie} \widehat{B}_{K-q})))$
    $\widehat{b}_{j,k} \leftarrow \widetilde{T}_u \wedge \neg\widetilde{T}_v$
    $\widehat{B}_k \leftarrow \widehat{B}_k \vee \widehat{b}_{j,k}$
  **end for**
**end for**
**return** $(\widehat{B}_0,\ldots,\widehat{B}_{K-1})$

hierarchically. A partial output is defined by $\widehat{b}_{j,k} = \widetilde{T}_{u_{j,k}} \wedge \neg\widetilde{T}_{v_{j,k}}$, which represents a rectangle mapping from the grayscale domain to the Boolean domain as shown in Fig. 3. In this figure, the mapping at the top represents a partial output from the $k$-th level of significance. The mapping at the bottom represents the logical sum of two partial outputs from the $(k-1)$-th level of significance, where each rectangle function represents a partial output. As a result, a partial output from a given level is split into two partial outputs in the next lower level within the same grayscale domain. This pattern is repeated in all mappings from the bitwise decomposition, as shown in Fig. 2a.

Let us consider Fig. 3 as a model to derive the optimization criteria. The first criterion states that if the partial output $\widehat{b}_{j,k}$ returns a bitplane of zeros, then the computation of $\widehat{b}_{j'+1,k-1}$ can be avoided since it also yields a bitplane of zeros. This can be verified in Fig. 3 because if none of the pixels of the filtered image have a gray level between $u_{j,k}$ and $v_{j,k}$, then no pixel will have a gray value in the subinterval between $u_{j'+1,k-1}$ and $v_{j'+1,k-1}$. On the other hand, the second criterion states that if the threshold bitplane $\widetilde{T}_{u_{j,k}}$ returns a bitplane of ones, then the computation of $\widehat{b}_{j',k-1}$ can be avoided since it yields a bitplane of zeros. This criterion can be also verified in Fig. 3 because if all pixels of the filtered image have a gray value greater or equal than $u_{j,k}$, then no pixel has a gray value between $u_{j',k-1}$ and $v_{j',k-1}-1$. By incorporating these two criteria into the stack filter algorithm, we avoid computing partial outputs in the lower levels of significance, notably reducing the processing time. As excluded partial outputs are evaluated in grayscale intervals where there are no pixels, we expect to process low-contrast images much faster than those with full dynamic range.

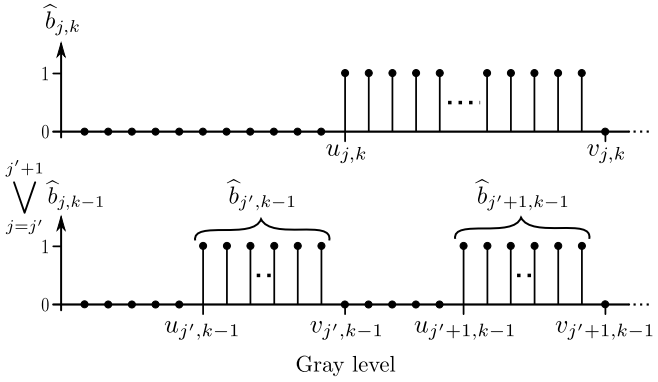



**Fig. 3** Mappings from grayscale to Boolean domain of partial outputs of consecutive bitplanes. Top: mapping of the partial output $\widehat{b}_{j,k}$. Bottom: mapping of $\widehat{b}_{j',k-1} \vee \widehat{b}_{j'+1,k-1}$. Variables $u$,$v$ indicate the threshold levels used to derive the partial outputs.

**Algorithm 2** OPTIMIZED_SF($(B_0, \ldots, B_{K-1})$,$s$,$f$)

1: $\widehat{B}_K \leftarrow \mathbf{0}$
2: $J_{K-1} \leftarrow \{0\}$
3: **for** $k = (K-1)$ down to 0 **do**
4: $\quad \widehat{B}_k \leftarrow \mathbf{0}$
5: $\quad J_{k-1} \leftarrow \{\emptyset\}$
6: $\quad$ **for all** $j \in J_k$ **do**
7: $\quad\quad u \leftarrow (2j+1)2^k$
⋮ ⋮
14: $\quad\quad \widehat{B}_k \leftarrow \widehat{B}_k \vee \widehat{b}_{j,k}$
15: $\quad\quad J_{k-1} \leftarrow$EVAL_CRITERIA$(j, \widehat{b}_{j,k}, \widetilde{T}_u, J_{k-1})$
16: $\quad$ **end for**
17: **end for**
18: **return** $(\widehat{B}_0, \ldots, \widehat{B}_{K-1})$

**Algorithm 2b**

1: **function** EVAL_CRITERIA$(j, \widehat{b}, \widetilde{T}_u, J)$
2: $\quad$ **if** $\widehat{b} = \mathbf{0}$ **then**
3: $\quad\quad J \leftarrow J \cup \{2j\}$
4: $\quad$ **else if** $\widetilde{T}_u = \mathbf{1}$ **then**
5: $\quad\quad J \leftarrow J \cup \{2j+1\}$
6: $\quad$ **else**
7: $\quad\quad J \leftarrow J \cup \{2j, 2j+1\}$
8: $\quad$ **end if**
9: $\quad$ **return** $J$
10: **end function**

In Algorithm 2 we present the optimized algorithm that includes the simplification criteria described above. This algorithm follows the same structure as Algorithm 1. The main differences occur in lines 6 and 15. In line 6, the $j$-indexed loop computes only the partial outputs that are linked to the indices stored in $J_k$. These indices are derived from the optimization criteria (evaluated with function EVAL_CRITERIA). Given that every partial output of index $j$ yields two partial outputs of indices $2j$ and $2j+1$ in the next level of significance, the criteria stated in lines 2 and 4 of Algorithm 2b define the partial output to be computed in the next level for each case.

### *3.2.1 Algorithm complexity*

From the algorithms described above, we can see that the most demanding operations are basic logical and binary morphological operations, both evaluated at a bitplane level. Therefore, the computational complexity of our algorithms can be estimated by enumerating these operations. Recall that FPPs compute logical and binary morphological operations in parallel at a blitplane level. In Table 1, we compare the complexity of the classical TD approach with the algorithms proposed. The operation count is done per bitplane, from the least to the most significant bitplane, given an 8-bit image.

First, we focus on analyzing the number of morphological operations. Both the TD approach and the general algorithm (GA) require 255 morphological operations, although their distribution per bitplane is different. That is, the TD approach applies a morphological operation per threshold level, whereas the general algorithm applies $2^{7-k}$ morphological operations per bitplane depending on its level of significance $k$. This implies that the most significant bitplanes, which convey the coarse information of an image, are computed much faster than the least significant bitplanes, which convey

**Table 1** Number of logical and morphological operations required to filter each bitplane of significance.

| | Morphological operator | | | | | Logical operator | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | *TD* | *GA* | *OA* | | | *TD* | *GA* | *OA* | | |
| $k$ | | | $s_{1\times3}$ | $s_{5\times5}$ | $s_{15\times15}$ | | | $s_{1\times3}$ | $s_{5\times5}$ | $s_{15\times15}$ |
| 0 | | 128 | 128 | 87 | 36 | | 1,921 | 2,177 | 1,477 | 611 |
| 1 | | 64 | 64 | 45 | 19 | | 833 | 961 | 676 | 282 |
| 2 | | 32 | 32 | 23 | 10 | | 353 | 417 | 299 | 130 |
| 3 | | 16 | 16 | 12 | 5 | | 145 | 177 | 134 | 55 |
| 4 | | 8 | 8 | 6 | 3 | | 57 | 73 | 56 | 28 |
| 5 | | 4 | 4 | 4 | 2 | | 21 | 29 | 28 | 14 |
| 6 | | 2 | 2 | 2 | 2 | | 7 | 11 | 11 | 10 |
| 7 | | 1 | 1 | 1 | 1 | | 2 | 4 | 4 | 4 |
| Total | 255 | 255 | 255 | 180 | 78 | 13,778 | 3,339 | 3,849 | 2,685 | 1,134 |

**Table 2** Approximate number of logic gates per pixel depending on the filtering method.

| | Sort-and-select | | TD | | GA | |
|---|---|---|---|---|---|---|
| $n$ | best | worst | erosion | median | erosion | median |
| 3 | 455 | 585 | 14,816 | 14,818 | 86 | 88 |
| 5 | 1,105 | 1,625 | 17,894 | 17,908 | 144 | 158 |
| 7 | 1,820 | 3,185 | 20,972 | 21,030 | 202 | 260 |
| 9 | 2,600 | 5,265 | 24,050 | 24,264 | 260 | 474 |
| 11 | 3,445 | 7,865 | 27,128 | 27,851 | 318 | 1,401 |
| 13 | 4,335 | 10,985 | 30,206 | 32,674 | 376 | 2,844 |
| 15 | 5,330 | 14,625 | 33,284 | 41,902 | 434 | 9,052 |

image details and texture. Meanwhile, the optimized algorithm (OA) follows the same hierarchical distribution, but its complexity actually varies with the image content. In other words, its performance depends on the gray-level distribution of the image, and the size of the structuring element. For instance, in Table 1, we enumerate the required operations to filter a uniformly-distributed image with structuring elements of different size. Particularly, with a $1 \times 3$ structuring element, the number of operations is the same as the general algorithm. By contrast, with a $5 \times 5$ or a $15 \times 15$ structuring element, the number of operations is drastically reduced in the least significant bitplanes. This reduction occurs because the simplification criteria (Algorithm 2) discard numerous partial outputs in the less significant bitplanes as the size of the structuring element increases. This behavior is expected given that the filtered image is usually smoother and coarser.

In terms of the number of logical operations, the TD approach needs 4 times more operations than the general algorithm. This overhead is caused by thresholdings and full additions, which are required to decompose and recompose the image. Meanwhile, the optimized algorithm reports a 15% overhead when using a small structuring element of $1 \times 3$ points. In this case, the overhead comes from the implementation of the simplification criteria established in Algorithm 2. Nevertheless, as the size of the structuring element increases, these criteria pay off by drastically reducing the number of logical operations, as shown in Table 1. In conclusion, the optimized algorithm may filter an image much faster than the general algorithm without compromising the fidelity of its output. Also, the proposed algorithms can be speeded up while running in a *coarse mode* operation by discarding the computation of the least significant bitplanes. For reference, a further comparison between the proposed methods will be discussed in Sect. 5.2.

Although our algorithms filter images in a bitplane fashion, it is interesting to compare the complexity of the sort-and-select approach with the decomposition-based methods at a pixel level. In Table 2, we present an estimation of the number of gates required to filter a pixel using an $n$-point structuring element. First, we evaluate the complexity of the sort-and-select filtering method. In this case, the complexity depends mostly on the number of comparisons to sort the data. In general, current sorting methods need between $2n \log n$ and $n^2$ comparisons in the best and worst case, respectively. Assuming that an 8-bit comparator requires 65 logic gates, the complexity bounds of the method can be estimated in terms of logic gates, as shown in Table 2. These bounds move apart as the size of the structuring element increases. Please note that this is a baseline estimation of the method's complexity since sorting also requires numerous data-swap operations. On the other hand, the complexity of the decomposition-based methods not only depends on the size of the structuring element, but also on the type of filter applied and the decomposition method. For instance, for the TD approach, the main burden is the implementation of decomposition/recomposition steps, which demand thresholdings and full adders. Conversely, our approach minimizes the decomposition/recomposition burden, but still depends on the filter complexity. Note that the erosion and the median filter are the extremes in the scale of complexity.

## 4 The Eye-RIS vision system

Recent advances in CMOS technology have allowed integrating sophisticated image processors into compact cameras. In particular, the so-called *smart cameras* [1,15,22] integrate image processors

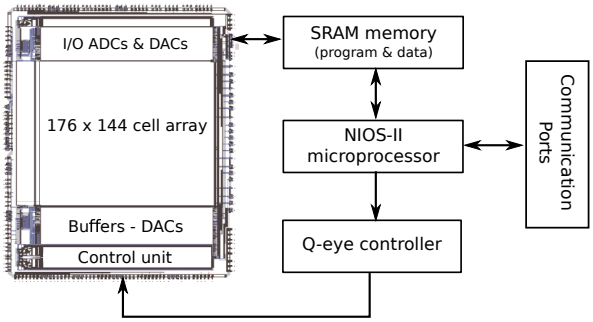



**Fig. 4** Conceptual block diagram of the Eye-RIS vision system.

capable of recognizing image patterns and making autonomous decisions. This sort of camera stands out from conventional cameras because of its low power consumption, high processing power, and autonomy. Smart cameras with a high silicon integration incorporate a parallel processor and the optical sensors into a single chip known as Focal Plane Processor (FPP). The architecture of an FPP consists of an interconnected array in which every cell integrates an optical sensor (pixel), a memory, and a processing unit [20]. This array architecture is known as Massively-Parallel SIMD (MP-SIMD), which permits us to execute low-level image operations at pixel/block level in parallel. To efficiently exploit the parallelism of an MP-SIMD architecture, sequential implementations of well-known vision algorithms need to be redesigned. For instance, Wu et al. [19] mapped several computer vision algorithms into an MP-SIMD architecture for an application of gesture analysis. Similarly, Zarandy et al. [20] presented a collection of algorithms implemented in an FPP for applications such as laser beam control, finger tracking, and traffic sign detection. The high computational power of smart cameras has been also exploited to tackle problems that involve multiple view cameras, such as the 3D pose reconstruction of humans in real time [21].

To implement our algorithm, we use the Eye-RIS vision system [12]. A conceptual block diagram of this system is presented in Fig. 4. This smart camera is composed of a mixed-signal FPP (Q-Eye), a digital microprocessor (NIOS II), plus some hardware interfaces, resulting in a low cost system that can make autonomous decisions and control external devices. The Q-Eye is a mixed-signal MP-SIMD processor with a computational power of 250 GOPS (Giga Operations Per Second) and a power consumption of 4 mW/GOPS. This processor is composed of a 176 x 144 cell array that processes binary and grayscale images. Eight grayscale and four binary image memory banks are internally available in the Q-Eye to buffer intermediate results. As the optical sensors and the Q-Eye are embedded in the same chip, grayscale images are directly processed in the analog domain. Thus, these images are only converted to digital when sent to an external SRAM memory to be manipulated by the microprocessor. On the other hand, the NIOS-II is a 32-bit general purpose microprocessor that performs 75 MIPS at 70 MHz. It acts like the "brain" of the system by controlling the Q-Eye, the SRAM memory, and the communication ports. The average power consumption of the camera is 1.5 W.

The implementation of our algorithm in the Eye-RIS vision system is distributed in two parts: on the one hand, the image decomposition/recomposition is performed in the microprocessor by simply reading/writing the bits of the input/output image. On the other hand, the stack filter algorithm is executed in the Q-eye to take advantage of its fast implementation of logical and morphological operations since they are the basis of our algorithm. Note that thanks to the massive parallelism of the Q-eye, *all of its cells* perform the same operation at the same time. In particular, note that each cell contains a combination of analog and digital processing modules that interact between them. For our implementation, two of these digital processing modules play a relevant role. One of them is the Local Logic Unit (LLU), which is a two-input logic block that performs logic operations between binary images in 2.5 $\mu$s. Thanks to a programmable truth table, different types of logical operations can be defined. Meanwhile, the other module is known as the Hit-and-Miss Unit (HMU), which checks whether the $3 \times 3$ neighborhood of a cell matches a specific pattern or not. This pattern or structuring element can be programmed to define which pixels are included in the morphological operation. Based on this hit-and-miss operator, many other binary morphological operators such

as the erosion, dilation, opening, and closing can be programmed in the Q-Eye. Every hit-and-miss operation is executed in approximately 10 $\mu$s. Another advantage of the morphological unit is that large structuring elements can be quickly performed with a minimal overhead by applying the predefined $3 \times 3$ structuring element multiple times with a single instruction. This type of operation largely differs from that of sequential implementations whose complexity rapidly increases with the size of the structuring element.

In order to develop applications with the Eye-RIS, we need to understand its execution flow. Primarily, note that NIOS microprocessor is programmed with C/C++ code, whereas the Q-Eye is programmed with a simple C-like programming language developed by AnaFocus called CFPP code. Therefore, any application in the Eye-RIS is composed of both types of code. The main control flow is performed by the NIOS microprocessor, which makes calls of CFPP code to be executed in the Q-Eye. In our algorithms, the image decomposition and recomposition is performed by the NIOS microprocessor because the Q-Eye cannot cope with them given its mixed-signal nature. Such decomposition and recomposition takes approximately 14 ms and 25 ms, respectively. Meanwhile, Algorithms 1 and 2 are programmed with CFPP code and computed in the Q-Eye. Unfortunately, the Q-Eye contains only 12 image memories internally. For our implementation, eight of these memories are reserved for the input bitplanes, and only 4 are available to store the output bitplanes and the intermediate results. Consequently, we are forced to transfer some of these bitplanes to the external SRAM memory. Unfortunately, transferring data between the Q-Eye and the SRAM memory is very slow, thus representing a bottleneck of the implementation. In some experiments, we found that such memory transfers can reach up to 65% of the total processing time. As a result, the coarse mode operation and the optimization criteria not only help to reduce the number of bitplane operations, but also the number of memory transfers. In total, the computation time of an image with the optimized algorithm ranges from 46.27 ms to 84.0 ms depending on its gray-level distribution and the size of the structuring element. These values correspond to a frame rate that goes from 11.9 fps to 21.6 fps. Note that there are two bottlenecks in the Eye-RIS that hinder the full potential of our algorithm. One of them is the decomposition and recomposition steps, which generate a baseline cost of 39 ms. The other is the limited number of image memories in the Q-Eye, and the slow transfers between internal and external memories. Despite these limitations, the time overhead of our approach is much lower than the TD implementation by considering that the image decomposition and recomposition also need to be performed in the NIOS processor. In this case, such operations can take more than 3 seconds, which is prohibitive. The problem is that image thresholding in the Q-eye is performed with analog comparators, which are quite inaccurate due to noise when the threshold level is set below 127.

## 5 Experimental results

To validate the proposed algorithms, we implemented them with the Eye-RIS vision system [12]. In the following sections, we will present an evaluation of our algorithms and a comparison with the baseline method based on threshold decomposition. In particular, the baseline method consists of applying the threshold decomposition to an 8-bit input image, which generates 255 bitplanes. Then, we apply a binary stack filter to each bitplane, and the output image is generated by summing up the 255 filtered bitplanes.

### 5.1 Evaluation of the hierarchical processing

To compare the hierarchical processing of our algorithm with the linear processing of the baseline approach, we measured the percentage of time to compute a certain number of bitplanes and the quality of the resulting image. For this experiment, we applied a median filter with a cross-shaped structuring element of width 3 to a test image of uniformly distributed random numbers by using Algorithm 1 and the baseline method.

In Fig. 5(a), we present the percentage of time spent in filtering a number of threshold bitplanes in ascending threshold level using the baseline approach. This graphic reveals that the processing time linearly increases with the number of bitplanes, which means that the computational cost of

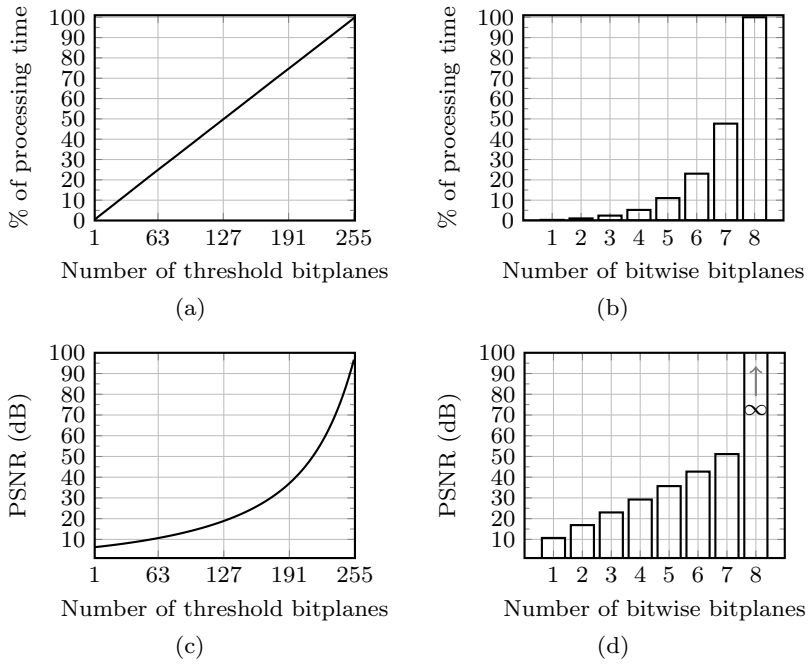
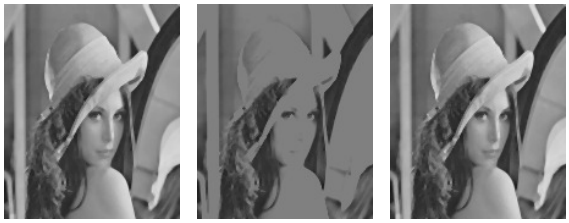



**Fig. 5** In (a) and (b): relative processing time of a certain number of threshold and bitwise bitplanes, respectively. In (c) and (d): quality of a reconstructed image with a certain number of threshold and bitwise bitplanes, respectively.

every threshold bitplane is the same. In Fig. 5(b), we present the percentage of time spent in filtering a certain number of bitwise bitplanes in descending order of significance. Note that the processing time exponentially increases in a factor of 2 as the number of bitplanes increases. This implies that the computational cost of every bitplane is not the same, and increases as the bit of significance decreases. In other words, the most significant bitplanes are computed much faster than the least significant bitplanes. This behavior is expected, since the most significant bitplanes are computed with a couple logical and morphological operations, whereas the least significant bitplanes need many of them, as stated in Table 1.

In Fig. 5(c), we evaluate the quality of an image reconstructed with a certain number of threshold bitplanes that are filtered in ascending order. The image quality is measured with the Peak Signal-to-Noise Ratio (PSNR) [9], between the filtered image $I^*$ and the reconstructed image $I^*_L$ with $L$ threshold bitplanes. Fig. 5(c) reveals that, by only processing the first threshold bitplane, the PSNR is around 6 dB; this value increases exponentially at a rate of 1.2% until all threshold bitplanes are processed, and the reconstruction is perfect. By contrast, in Fig. 5(d), we can see the evolution of the image quality as we filter the bitwise bitplanes from the most to the least significant bitplane. The image quality is also measured using PSNR, but now $L$ represents the number of bitwise bitplanes. The bar chart shows that the reconstructed image with the most significant bitplane yields a PSNR of 10.7 dB; this value increases linearly at a rate of 6.63 dB per bitplane until all bitplanes are processed and the reconstruction is perfect.

In general, to filter $K$ bitwise bitplanes in descending order of significance, we need to filter $2^K - 1$ threshold bitplanes to derive the corresponding partial outputs. This relation can give us an approximate equivalence of the complexity between our algorithm and the baseline approach. For instance, by processing the seven most significant bitplanes, the PSNR of the reconstructed image is 51 dB, as shown in Fig. 5(d). Equivalently, by processing 127 threshold bitplanes with the baseline approach, the PSNR is only 19 dB as shown in Fig. 5(c). Thanks to the hierarchical structure of our algorithm, we can reconstruct an image of much better quality than with the baseline approach using the same number of threshold bitplanes. This statement is visually checked in Fig. 6 by comparing a median filtered image (Fig. 6(a)) with their partial reconstructions resulting from both filtering methods. In Fig. 6(b), we can see the reconstructed image with 127 threshold bitplanes using the



(a) (b) (c)

**Fig. 6** (a) Median filtered image, (b) Reconstruction of the filtered image with 127 threshold bitplanes (PSNR = 19 dB), (c) Reconstruction of the filtered image with the seven most significant bitplanes (PSNR = 51dB).

baseline approach, which gives a darker and coarser image than Fig. 6(a). By contrast, Fig. 6(c) shows the reconstructed image with the seven most significant bitplanes using our approach, which gives a very similar image to Fig. 6(a). In conclusion, by not filtering the least significant bitplane of an image, we can reduce the computation time by half without significantly compromising the quality of the reconstructed image. This filter approximation may become attractive to many real-time applications based on stack filters.

### 5.2 Comparative analysis between the general algorithm and the optimized algorithm

In Sect. 3.2, we proposed some optimization criteria that reduce the computation time of Algorithm 1 depending on the image distribution. Therefore, we expect to compute images with little contrast much faster than those with high contrast. This relationship is further examined by testing our optimized algorithm with three images of different entropy, which are depicted in Fig. 7(a)-(c). The corresponding histograms of these images are presented in Fig. 7(d)-(f). The stack filter employed in the test is the erosion filter with a square structuring element.

In Fig. 8(a) we plot the number of bitplane erosions required to filter every image using the general and the optimized algorithm with square structuring elements of different width. With the general algorithm, the number of bitplane erosions is 255 regardless of the image entropy. By contrast, with the optimized algorithm, the number of bitplane erosions decreases as the entropy of the image decreases. Moreover, the number of bitplane erosions decreases as the width of the structuring element increases. In this regard, as the structuring element increases, the filtered image tends to have many pixels with the same gray tone, and thus less contrast and entropy. As a result, many partial outputs are not computed in lower levels of significance, which reduces the number of bitplane erosions.

In Fig. 8(b), we evaluate the processing time of the images of low, medium, and high entropy using the general and the optimized algorithm. With the general algorithm, the processing time increases with the size of the structuring element because the computation of the bitplane erosions gets slower as the structuring element increases. Moreover, the total number of bitplane erosions remains the same regardless of the neighborhood size. On the other hand, with the optimized algorithm, the processing time decreases as the image entropy decreases and the structuring element increases. This reduction in time is closely related to the exponential decay of the number of bitplane erosions. Moreover, this reduction is larger than the time overhead of filtering with large structuring elements. In conclusion, with the optimized algorithm, we can significantly reduce the computation time of an image depending on its entropy and the size of the structuring element employed.

### 5.3 Rank-order filter implementation and performance

Rank Order Filters (ROFs), also called order-statistic filters [10], compose a stack-filter subclass that return local statistics of an image. A ROF of rank $r$, denoted as $ROF_{(r)}$, selects the $r$-th largest element from a list of pixel values defined by a structuring element. As $r$ goes from 1 to $N$, where $N$ is the number of elements, the filters $ROF_{(1)}$ and $ROF_{(N)}$ correspond to the morphological dilation and erosion operators, respectively. Another special case occurs when $r = (N + 1)/2$, which corresponds to the median filter when $N$ is an odd value.

*5.3.1 Implementation in the Eye-RIS vision system*

The implementation of rank order filters can be performed with the naive sort-and-select method or with our optimized algorithm by using the appropriate binary filter. Unfortunately, the Eye-RIS system does not provide any sorting algorithm implemented in the Q-Eye, and the only binary rank filters available are the erosion and dilation. As a result, the rest of the binary rank filters need to be derived from these morphological operations. According to Wendt et al. [18], any binary rank filter can be represented as a Positive Boolean Function (PBF). For instance, the erosion ($\epsilon$) and dilation ($\delta$) of the binary sequence $X = (x_1, x_2, \cdots, x_N)$ can be expressed with the following PBFs:

$$\epsilon(X) = x_1 x_2 \cdots x_N, \tag{11}$$

$$\delta(X) = x_1 + x_2 + \cdots + x_N, \tag{12}$$

where the addition represents the OR operation and the multiplication the AND operation. According to [4], the PBF of any binary rank-order filter can be implemented as a sum-of-products:

$$ROF_{(r)}(X) = \sum_{1 \leq n_1 < n_2 < \cdots < n_r \leqslant N} x_{n_1} x_{n_2} \cdots x_{n_r}, \tag{13}$$

where the indices $n_1, n_2, \cdots, n_r$ are defined by the set of combinations that meet the inequality $1 \leq n_1 < n_2 < \cdots < n_r \leqslant N$. By combining (11) with (13), we can rewrite the latter as follows:

$$ROF_{(r)}(X) = \sum_{1 \leq n_1 < n_2 < \cdots < n_r \leqslant N} \epsilon(x_{n_1} x_{n_2} \cdots x_{n_r}). \tag{14}$$

As a result, every product term of the PBF can be implemented as an erosion, and its output is added up yielding the rank filter desired. The number of erosions required to derive a $r$-th rank

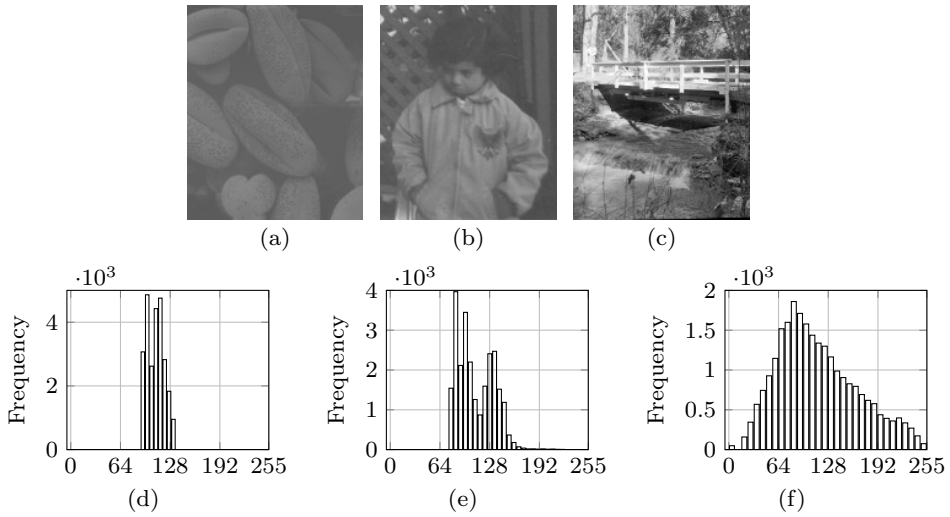



**Fig. 7** Test images with: (a) low entropy (5.2 bits), (b) medium entropy (6.2 bits), and (c) high entropy (7.6 bits). The histogram of these images are depicted in (d), (e), and (f), respectively.

**Table 3** Number of erosion operations required to derive each ROF with a 5-point neighborhood.

| | ***Number of erosions*** | |
|---|---|---|
| ***Rank Order Filter*** | $\binom{5}{r}$ | *duality* |
| $ROF_{(1)}$(Dilation) | 5 | 1 |
| $ROF_{(2)}$ | 10 | 5 |
| $ROF_{(3)}$(Median) | 10 | 10 |
| $ROF_{(4)}$ | 5 | 5 |
| $ROF_{(5)}$(Erosion) | 1 | 1 |

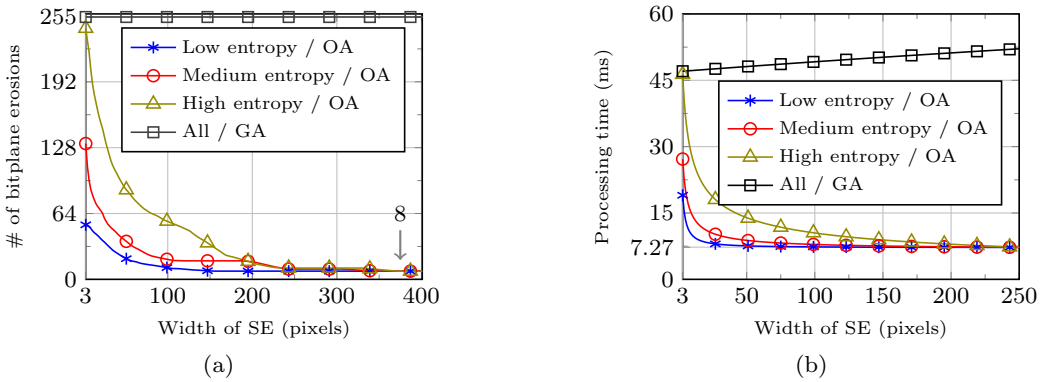



**Fig. 8** (a) Number of bitplane erosions, and (b) processing time required to filter the images of low, medium, and high entropy with square structuring elements (SE) of different width. Results are provided for the general algorithm (GA) and for the optimized algorithm (OA).

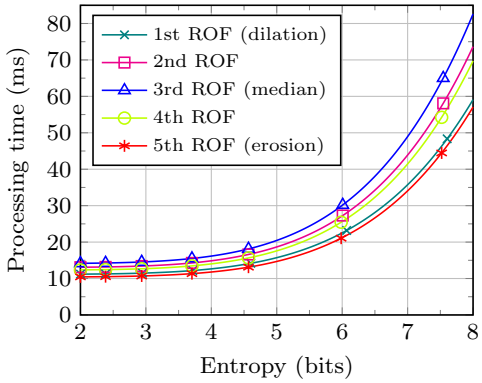



**Fig. 9** ROF performance on the Eye-RIS vision system.

filter is determined by the binomial coefficient $\binom{N}{r}$, where $N$ is the number of elements of the structuring element. In Table 3, we show the number of erosions required by each rank-order filter with a structuring element of size 5. Also, this table shows that the number of erosions for $r < 3$ can be reduced by using the duality property [10], which states that $ROF_{(r)}(X) = \neg ROF_{(N-r+1)}(\neg X)$. Therefore, $ROF_{(1)}$ can be derived in terms of $ROF_{(5)}$, while $ROF_{(2)}$ can be derived in terms of $ROF_{(4)}$. In particular, the number of erosions $N_e$ using the duality property is computed as follows

$$N_e = \begin{cases} \binom{N}{N-r+1}, & \text{if } 1 \leq r \leq \frac{N+1}{2}, \\ \binom{N}{r}, & \text{if } \frac{N+1}{2} < r \leq N. \end{cases} \tag{15}$$

To evaluate the processing time of rank-order filters using our stack filter algorithm, we filtered the image depicted in Fig. 7(c) with a cross-shaped structuring element of 5 points. Also, we tested our algorithm with images of different entropy values. These images are obtained by linearly mapping the image with different dynamic ranges that run from 1 to 255 gray tones. The performance of the 5 possible rank filters is shown in Fig. 9. In general, the processing time decreases as the image entropy decreases. This behavior was previously discussed in Sect. 5.2 for the case of the erosion. Now, we can see that other rank-order filters follow a similar behavior. In Fig. 9, we can also see that low-entropy images such as synthetic images can be processed in less than 20 ms since their entropy is usually less than 5 bits. On the other hand, natural images have an entropy between 5 and 8 bits, thus giving a processing time between 15 and 80 ms approximately. In particular, note that the curves of dual filters such as erosion/dilation and $ROF_{(2)}/ROF_{(4)}$ have similar performance. This is expected because dual rank filters require the same number of erosions, as shown in Table 3. Moreover, the processing time of the median filter is slower than the other filters because it requires more binary

erosions. Therefore, if a larger structuring element is used, the gap in time between the median and erosion filter would be larger. In summary, our stack filter algorithm allows us to implement any type of rank-order filter based on either bitplane erosions or its direct PBF implementation. Moreover, by integrating the optimization criteria into our algorithm, we can reduce its computation time depending on the image distribution without compromising the quality of the filtered image.

## 6 Conclusion

In this work, we proposed a stack filter algorithm based on the image bitwise decomposition for a smart camera with a 2D-array image processor. Thanks to its hierarchical structure, this algorithm prioritizes the processing of the most informative bitplanes because they are filtered much faster than the least informative bitplanes. As a result, we can approximate a stack filtered image by skipping the computation of the least informative bitplane without significantly compromising the quality of the output image, while reducing the processing time by half. On the other hand, our optimized algorithm exhibits a close relationship between processing time and image distribution by computing low-entropy images up to 5 times faster than high-entropy images. Another interesting feature is that our approach drastically reduces the processing time when using large structuring elements. All these features make our algorithm attractive to applications that need to run in real-time.

## A Derivation of equation (5)

The bitwise decomposition function, stated in (2), can be expressed in terms of floor functions using the modulo operator $x \bmod y = x - y \lfloor x/y \rfloor$ and the identity $\lfloor \lfloor x/p \rfloor / q \rfloor = \lfloor x/pq \rfloor$ as follows

$$B_k = \left\lfloor \frac{I}{2^k} \right\rfloor - 2 \left\lfloor \frac{I}{2^{k+1}} \right\rfloor . \tag{16}$$

In general, a floor function has a staircase shape, where every step can be expressed in terms of threshold functions. Therefore, we can state that

$$\left\lfloor \frac{I}{p} \right\rfloor = \sum_j j \left( T_{jp} - T_{(j+1)p} \right) , \tag{17}$$

where $p$ is an integer number that determines the stair width. Consequently, the floor functions in (16) can be expressed as

$$\left\lfloor \frac{I}{2^k} \right\rfloor = \sum_{j'=0}^{N_k} j' \left( T_{j'2^k} - T_{(j'+1)2^k} \right) , \tag{18}$$

$$\left\lfloor \frac{I}{2^{k+1}} \right\rfloor = \sum_{j=0}^{N_{k+1}} j \left( T_{j2^{k+1}} - T_{(j+1)2^{k+1}} \right) , \tag{19}$$

where $N_k = 2^{K-k} - 1$ and $K$ represents the bit depth of the image. As the stair width in (19) is two times larger than in (18), the indices $j$ and $j'$ point to different thresholding levels. Therefore, to use a common index, we express (18) with the index $j$ as follows

$$\left\lfloor \frac{I}{2^k} \right\rfloor = \sum_{j=0}^{N_{k+1}} 2j \left( T_{j2^{k+1}} - T_{(2j+1)2^k} \right) + (2j+1) \left( T_{(2j+1)2^k} - T_{(j+1)2^{k+1}} \right) . \tag{20}$$

By substituting (19) and (20) into (16) we obtain the relation between decompositions as

$$B_k = \sum_{j=0}^{N_{k+1}} T_{(2j+1)2^k} - T_{(j+1)2^{k+1}} . \tag{21}$$

Equivalently, we can express this relation in terms of Boolean operators as shown below

$$B_k = \bigvee_{j=0}^{N_{k+1}} \left( T_{(2j+1)2^k} \wedge \neg T_{(j+1)2^{k+1}} \right). \tag{22}$$

## B Derivation of equation (6)

As stated in (1), the threshold function is based on the comparison of the image $I$ and a gray level $\ell$. The equality comparator, on the other hand, is a more fundamental operation defined as shown below

$$C_\ell(n,m) = \begin{cases} 1, & \text{if } I(n,m) = \ell, \\ 0, & \text{otherwise.} \end{cases} \tag{23}$$

If we apply the bitwise decomposition to $I$ and $\ell$, we get $(B_{K-1}, ..., B_0)_2$ and $(\beta_{K-1,\ell}, ..., \beta_{0,\ell})_2$, respectively. By comparing their $K$ bit levels of decomposition, we can also express the equality comparator as follows

$$C_\ell(n,m) = \bigwedge_{k=0}^{K-1} B_k(n,m) \leftrightarrow \beta_{k,\ell}. \tag{24}$$

Based on equality comparisons we can reformulate the threshold function (1) as shown below

$$T_l(n,m) = \bigvee_{\ell=l}^{2^K-1} C_\ell(n,m), \tag{25}$$

where $C_\ell$ can be substituted by (24) leading to

$$T_l(n,m) = \bigvee_{\ell=l}^{2^K-1} \bigwedge_{k=0}^{K-1} \left( B_k(n,m) \leftrightarrow \beta_{k,\ell} \right). \tag{26}$$

This Boolean function has the canonical form of a Sum of Products (SoP) [13], and it can be minimized yielding a simplified representation. In the following paragraphs, we will perform the minimization of (26) by assuming that $T_l$ returns the minimized expression and $F_l(B_{K-1}, B_{K-2}, \ldots, B_0)$ is the sum-of-products representation. In this way, we start applying the Shannon's expansion theorem [14] to $F_l$ as follows

$$F_l(B_{K-1}, B_{K-2}, \ldots, B_0) = (B_{K-1} \wedge F_l(\mathbf{1}, B_{K-2}, \ldots, B_0)) \vee (\neg B_{K-1} \wedge F_l(\mathbf{0}, B_{K-2}, \ldots, B_0)), \tag{27}$$

where $\mathbf{1}$ and $\mathbf{0}$ represent bitplanes of ones and zeros, respectively. This expression can be reduced for two possible cases:

*Case* 1. If $l < 2^{K-1}$, then $F_l(\mathbf{1}, B_{K-2}, \ldots, B_0) = \mathbf{1}$ by the *uniting theorem* [13]. As a result, we can reduce (27) to $F_l(B_{K-1}, B_{K-2}, \ldots, B_0) = B_{K-1} \vee (\neg B_{K-1} \wedge F_l(\mathbf{0}, B_{K-2}, \ldots, B_0))$, and then use the *elimination theorem* [13] to get

$$F_l(B_{K-1}, B_{K-2}, \ldots, B_0) = B_{K-1} \vee F_l(\mathbf{0}, B_{K-2}, \ldots, B_0). \tag{28}$$

*Case* 2. If $l \geq 2^{K-1}$, then $F_l(\mathbf{0}, B_{K-2}, \ldots, B_0) = \mathbf{0}$. For this case, the equation (27) is reduced to

$$F_l(B_{K-1}, B_{K-2}, \ldots, B_0) = B_{K-1} \wedge F_l(\mathbf{1}, B_{K-2}, \ldots, B_0). \tag{29}$$

By gathering (28) and (29) into a single function, we can express (27) as follows

$$T_l = \begin{cases} B_{K-1} \vee F_l(\mathbf{0}, B_{K-2}, \ldots, B_0), & \text{if } \lambda_{K-1} = 0, \\ B_{K-1} \wedge F_l(\mathbf{1}, B_{K-2}, \ldots, B_0), & \text{if } \lambda_{K-1} = 1, \end{cases} \tag{30}$$

where $\lambda_{K-1}$ is the most significant bit of $l$ given that $(l)_{10} = (\lambda_{K-1}\lambda_{K-2}...\lambda_0)_2$. Note that $\lambda_{K-1} = 0$ implies that $l < 2^{K-1}$, while $\lambda_{K-1} = 1$ implies that $l \geq 2^{K-1}$. This piecewise function can be redefined more compactly as follows

$$T_l = B_{K-1} \underset{\lambda_{K-1}}{\bowtie} F_l(\boldsymbol{\lambda}_{K-1}, B_{K-2}, \ldots, B_0), \tag{31}$$

where $\boldsymbol{\lambda}_{K-1}$ represents a bitplane of either zeros or ones depending on the value of $\lambda_{K-1}$, while the *bow tie* operator is defined by

$$\underset{x}{\bowtie} = \begin{cases} \vee, \text{ if } x = 0, \\ \wedge, \text{ if } x = 1. \end{cases} \tag{32}$$

The expression in (31) can be further worked out by applying the Shannon's expansion theorem to $F_l(\boldsymbol{\lambda}_{K-1}, B_{K-2}, \ldots, B_0)$, which leads to

$$F_l(\boldsymbol{\lambda}_{K-1}, B_{K-2}, \ldots, B_0) = (B_{K-2} \wedge F_l(\boldsymbol{\lambda}_{K-1}, \mathbf{1}, B_{K-3}, \ldots, B_0)) \vee (\neg B_{K-2} \wedge F_l(\boldsymbol{\lambda}_{K-1}, \mathbf{0}, B_{K-3}, \ldots, B_0)). \tag{33}$$

This expression can be simplified for four possible cases:

*Case* 1. If $l < 2^{K-2}$, then $F_l(\mathbf{0}, \mathbf{1}, B_{K-3}, \ldots, B_0) = \mathbf{1}$, which after substituted in (33) we get

$$F_l(\mathbf{0}, B_{K-2}, \ldots, B_0) = B_{K-2} \vee F_l(\mathbf{0}, \mathbf{0}, B_{K-3}, \ldots, B_0). \tag{34}$$

*Case* 2. If $l \geq 2^{K-2}$, then $F_l(\mathbf{0}, \mathbf{0}, B_{K-3}, \ldots, B_0) = \mathbf{0}$, which reduces (33) to

$$F_l(\mathbf{0}, B_{K-2}, \ldots, B_0) = B_{K-2} \wedge F_l(\mathbf{0}, \mathbf{1}, B_{K-3}, \ldots, B_0). \tag{35}$$

*Case* 3. If $l < 3 \cdot 2^{K-2}$, then $F_l(\mathbf{1}, \mathbf{1}, B_{K-3}, \ldots, B_0) = \mathbf{1}$, and the expansion is simplified as

$$F_l(\mathbf{1}, B_{K-2}, \ldots, B_0) = B_{K-2} \vee F_l(\mathbf{1}, \mathbf{0}, B_{K-3}, \ldots, B_0). \tag{36}$$

*Case* 4. If $l \geq 3 \cdot 2^{K-2}$, then $F_l(\mathbf{1}, \mathbf{0}, B_{K-3}, \ldots, B_0) = \mathbf{0}$, which after substituted in (33) returns

$$F_l(\mathbf{1}, B_{K-2}, \ldots, B_0) = B_{K-2} \wedge F_l(\mathbf{1}, \mathbf{1}, B_{K-3}, \ldots, B_0). \tag{37}$$

All these cases are put together in the following form:

$$T_l = (B_{K-1} \underset{\lambda_{K-1}}{\bowtie} (B_{K-2} \underset{\lambda_{K-2}}{\bowtie} F_l(\boldsymbol{\lambda}_{K-1}, \boldsymbol{\lambda}_{K-1}, B_{K-3}, \ldots, B_0))) \tag{38}$$

In light of (31) and (38), we can easily deduce the last Shannon's expansion as follows

$$T_l = (B_{K-1} \underset{\lambda_{K-1}}{\bowtie} (B_{K-2} \underset{\lambda_{K-2}}{\bowtie} \cdots (B_0 \underset{\lambda_0}{\bowtie} F_l(\boldsymbol{\lambda}_{K-1}, \boldsymbol{\lambda}_{K-2}, \ldots, \boldsymbol{\lambda}_0)))) \tag{39}$$

Given that $F_l(\boldsymbol{\lambda}_{K-1}, \boldsymbol{\lambda}_{K-2}, \ldots, \boldsymbol{\lambda}_0) = \mathbf{1}$, then equation (39) can be simplified if the least significant bits of $l$ are set to zero. For instance, if $z$ is the number of the least significant bits of $l$ set to zero, then $\lambda_{z-1} = \lambda_{z-2} = \cdots = \lambda_0 = 0$ and the rightmost part of (39) is reduced as follows $B_{z-1} \vee B_{z-2} \vee \cdots \vee \mathbf{1} = \mathbf{1}$. Finally, the minimization of the sum of products stated in (26) is shown below

$$T_l = (B_{K-1} \underset{\lambda_{K-1}}{\bowtie} (B_{K-2} \underset{\lambda_{K-2}}{\bowtie} \cdots (B_{K-q+1} \underset{\lambda_{K-q+1}}{\bowtie} B_{K-q}))) \tag{40}$$

where $q = K - z$.

**Acknowledgements** This work has been supported by IMEC and by the Flemish Fund for Scientific Research (FWO), and project TEC2016-75976-R, financed by the Spanish Ministerio de Economía y Competitividad and the European Regional Development Fund (ERDF)

## References


1. Abbo, A.A., Kleihorst, R.P., Schueler, B.: Xetal-II: A low-power massively-parallel processor for video scene analysis. Signal Processing Systems **62**(1), 17–27 (2011)
2. Avedillo, M., Quintana, J., Alami, H., Jimenez-Calderon, A.: A practical parallel architecture for stacks filters. Journal of VLSI signal processing systems for signal, image and video technology **38**(2), 91–100 (2004)
3. Chakrabarti, C., Lucke, L.: VLSI architectures for weighted order statistic (WOS) filters. In: IEEE International Symposium on Circuits and Systems (ISCAS), vol. 2, pp. 320–323 vol.2 (1998)
4. Chen, K.: Bit-serial realizations of a class of nonlinear filters based on positive boolean functions. IEEE Transactions on Circuits and Systems **36**(6), 785–794 (1989)
5. Droogenbroeck, M., Buckley, M.: Morphological erosions and openings: Fast algorithms based on anchors. Journal of Mathematical Imaging and Vision **22**(2-3), 121–142 (2005)
6. Frías-Velázquez, A., Morros, J.: Gray-scale erosion algorithm based on image bitwise decomposition: Application to focal plane processors. In: IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP, pp. 845–848 (2009)
7. Frías-Velázquez, A., Philips, W.: Bit-plane stack filter algorithm for focal plane processors. In: IEEE International Conference on Image Processing ICIP, pp. 3741–3744. IEEE (2010)
8. Gevorkian, D., Egiazarian, K., Agaian, S., Astola, J., Vainio, O.: Parallel algorithms and VLSI architectures for stack filtering using Fibonacci p-codes. IEEE Transactions on Signal Processing, **43**(1), 286–295 (1995)
9. Gonzalez, R.C., Woods, R.E.: Digital Image Processing. Prentice-Hall, Inc. (2006)
10. Maragos, P., Schafer, R.: Morphological filters–Part II: Their relations to median, order-statistic, and stack filters. IEEE Transactions on Acoustics, Speech and Signal Processing **35**(8), 1170–1184 (1987)
11. Mertzios, B.G., Tsirikolias, K.: Coordinate logic filters and their applications in image processing and pattern recognition. Circuits, Systems and Signal Processing **17**(4), 517–538 (1998)
12. Rodríguez-Vázquez, A., Domínguez-Castro, R., Jiménez-Garrido, F., Morillas, S., Listán, J., Alba, L., Utrera, C., Espejo, S., Romay, R.: The Eye-RIS CMOS vision system. In: Analog Circuit Design, pp. 15–32. Springer Netherlands (2008)
13. Roth Jr, C., Kinney, L.: Fundamentals of logic design. Cengage Learning (2013)
14. Shannon, C., et al.: The synthesis of two-terminal switching circuits. Bell System Tech. Journal **28**(1), 59–98 (1949)
15. Shi, Y.: Smart cameras for machine vision. In: Smart Cameras, pp. 283–303. Springer (2010)
16. Spiliotis, I., Boutalis, Y.: Parameterized real-time moment computation on gray images using block techniques. Journal of Real-Time Image Processing **6**(2), 81–91 (2011)
17. Urbach, E., Wilkinson, M.H.F.: Efficient 2-D grayscale morphological transformations with arbitrary flat structuring elements. IEEE Transactions on Image Processing **17**(1), 1–8 (2008)
18. Wendt, P., Coyle, E., Gallagher N.C., J.: Stack filters. IEEE Transactions on Acoustics, Speech and Signal Processing **34**(4), 898–911 (1986)
19. Wu, C., Aghajan, H., Kleihorst, R.: Mapping vision algorithms on SIMD architecture smart cameras. In: First ACM/IEEE International Conference on Distributed Smart Cameras, pp. 27–34 (2007)
20. Zarándy, Á.: Focal-plane sensor-processor chips. Springer (2011)
21. Zivkovic, Z.: Wireless smart camera network for real-time human 3D pose reconstruction. Computer Vision and Image Understanding **114**(11), 1215 – 1222 (2010)
22. Zivkovic, Z., Kleihorst, R.: CHAPTER 21 - smart cameras for wireless camera networks: Architecture overview. In: H. Aghajan, A. Cavallaro (eds.) Multi-Camera Networks, pp. 497 – 510. Academic Press, Oxford (2009)