Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2007, Article ID 97845, 16 pages
doi:10.1155/2007/97845

Research Article

Lightweight Object Tracking in Compressed Video Streams Demonstrated in Region-of-Interest Coding

Robbie De Sutter,[1] Koen De Wolf,[1] Sam Lerouge,[2] and Rik Van de Walle[1]

[1] *Multimedia Lab, Department of Electronics and Information Systems, Ghent University - IBBT, Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium*

[2] *Regionale Media Maatschappij, Kwadestraat 151b, B-8800 Roeselare, Belgium*

Received 25 January 2006; Revised 28 September 2006; Accepted 11 October 2006

Recommended by Dimitrios Tzovaras

Video scalability is a recent video coding technology that allows content providers to offer multiple quality versions from a single encoded video file in order to target different kinds of end-user devices and networks. One form of scalability utilizes the *region-of-interest* concept, that is, the possibility to mark objects or zones within the video as more important than the surrounding area. The scalable video coder ensures that these regions-of-interest are received by an end-user device before the surrounding area and preferably in higher quality. In this paper, novel algorithms are presented making it possible to automatically track the marked objects in the regions of interest. Our methods detect the overall motion of a designated object by retrieving the motion vectors calculated during the motion estimation step of the video encoder. Using this knowledge, the region-of-interest is translated, thus following the objects within. Furthermore, the proposed algorithms allow adequate resizing of the region-of-interest. By using the available information from the video encoder, object tracking can be done in the compressed domain and is suitable for real-time and streaming applications. A time-complexity analysis is given for the algorithms proving the low complexity thereof and the usability for real-time applications. The proposed object tracking methods are generic and can be applied to any codec that calculates the motion vector field. In this paper, the algorithms are implemented within MPEG-4 fine-granularity scalability codec. Different tests on different video sequences are performed to evaluate the accuracy of the methods. Our novel algorithms achieve a precision up to 96.4%.

1. INTRODUCTION

Video content providers want to target as many kinds of end-user devices without the need to create a large simulstore of multiple versions of the same video fragment. As a consequence, recent video encoding techniques embed *scalability* such that multiple versions of an encoded video can be derived on the fly, that is, without the need for decoding and reencoding the video. Within the Moving Picture Experts Group (MPEG), the *fine-granularity scalability* (FGS) technology was developed to add scalability to the MPEG video coding technologies. The FGS technology can be used to implement the *region-of-interest* (ROI) concept. An ROI identifies area(s) within the video scene as more interesting or important than the remaining area(s). Consider for example a news broadcast: the newsreader is more important than the surrounding news studio setting, thus the newsreader can be identified as the ROI for this scene. If an end-user device is not capable of receiving the full-quality video feed (due to, e.g., restrictive bandwidth capacity), the streaming server or the network can choose to send only the ROI in high quality and the remaining areas in low quality. As such, less bandwidth is required, however the most important regions (i.e., the ROI) can be viewed in high quality.

An issue that arises with this technique is the identification of the important regions in the video scene and the tracking of the object(s) in these important regions throughout the video sequence. In this paper, we focus on the latter issue. We present new methods that can be used to improve the handling of regions of interest. A novel concept and novel algorithms are introduced in order to track objects within an ROI by reusing the motion vectors calculated during the motion estimation step of the video encoder. Nevertheless, the performance of our algorithms depends on the motion estimation algorithms. Note that the methods we introduce in this paper are not restricted to this particular use case, they can also be applied in other scenarios if object tracking is needed. While other comparable object tracking algorithms work in the uncompressed domain and have a very high computational complexity, our algorithms work during the actual encoding of the video in the compressed domain and use the information directly available from the encoder itself. This results in an object tracking scheme that is very fast and that can be used for real-time and streaming applications.

The outline of the paper is as follows. First, we discuss how our approach differs from existing techniques. Next, we briefly explain the principles of MPEG-4 FGS encoding and its possibilities to handle regions of interest. In Section 4, we explain our algorithms to enable object tracking and to handle object resizing by using the motion vector field. Next, a test setup to evaluate the methods is outlined in Section 5. Section 6 presents and discusses the results. Finally, conclusions are drawn in Section 7.

2. RELATED WORK

Many algorithms, systems, and techniques have been described in the literature to identify and to track objects in video streams. The identification of objects is usually called *object segmentation* or *object classification*.

Most discussed techniques for object segmentation use object models and ontologies [1], color information (e.g., flesh tone for human face recognition) [2, 3], semantic and probabilistic decomposition of the video frames with learning capabilities [4, 5], temporal comparisons between consecutive images of a video stream [6], leveled watershed techniques [7], or a combination of the previous techniques as in [8, 9]. These techniques could be used by themselves to track an object over the consecutive frames of a video stream, but this implies a computational burden.

Object tracking is following a moving object over consecutive video frames. By itself, the object tracking techniques can be subdivided into object tracking in the pixel domain and object tracking in the compressed domain. The former technique requires that the video stream be (partially) decoded before it can be processed. Working in the pixel domain allows to apply advanced techniques like stochastic algorithms [10], but the necessary decoding step decelerates the tracking process and makes it infeasible for real-time applications.

The techniques for object tracking in the compressed domain take advantage of the fact that the video encoders calculate the motion vector field during the encoding process of the sequence. A motion vector field is a mathematical representation of the displacement of a group of pixels between related (previous and successive) frames of a video stream. In [11], trajectory estimation is made based on the motion vector field and partially decoded DCT coefficients. While this technique promises to be "fast enough for real-time applications," it assumes that there is "no camera motion." A similar technique is discussed in [12], but fails when "the object was small or the object was nonrigid or was changing a lot in shape and size." In [13], the camera motion is estimated by using a Hough transform for the overall motion and a mean-shift algorithm based on the motion vector field. The technique has good results for shorter sequences, unfortunately the computations are rather complex and time-consuming, making this technique difficult to implement for real-time applications. Finally, we want to mention the work of Favalli et al. [14]. In this work, how object tracking can be done is described, solely by using the information of the motion vectors and applying very simple calculations, hence adding "as little additional processing as possible to the complexity of a standard decoder."

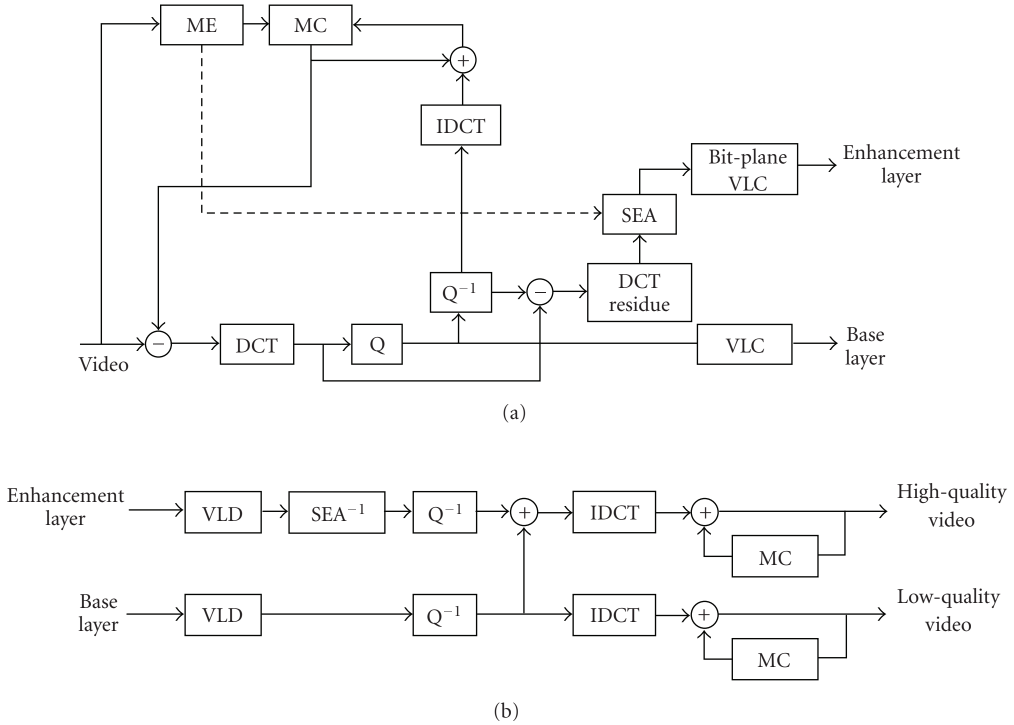
The work presented in this paper is different from all the previously discussed techniques, and [14] in particular, as it adds no complexity to the decoder, but only very little additional computations with low time complexity to the encoder of the video stream. By doing so, the techniques introduced in this paper can be used for real-time and streaming applications in contrast to [13]. Also, our techniques allow the tracking of any kind of object of any arbitrary shape and are capable of handling camera motion, object resizing, and object deformation, addressing the main shortcomings of [11, 12]. Furthermore, although the performance of our algorithms depends on the motion estimation algorithms, our techniques are independent of the video encoding specification itself; for demonstration and testing purpose, we used the MPEG-4 FGS codec and exploited its ROI capabilities to visualize the results of the object tracking algorithms. Finally, the novel methods for object tracking in the compressed domain introduced in this paper are detached from the macroblocks by introducing two independent layers on top of the macroblock grid. This enables the tracking of fine-detailed objects.

Note that the techniques described in this paper do not perform object segmentation nor object classification. It is assumed that the object that has to be followed is determined beforehand, either by manual selection (e.g., by a human operator) or automatically by one of the available segmentation techniques.

3. REGION-OF-INTEREST IN MPEG-4 FGS

MPEG is offering scalability in several of its video compression standards, such as the FGS standard [15] and the scalable video coding (SVC) standard [16]. In this paper, we use the FGS standard to demonstrate our algorithms, but these could also be implemented in other (scalable) video compression models such as SVC. Basically, FGS creates two video layers: a base layer that contains a low-quality version of the video that can be streamed and decoded under any circumstances by any FGS-compliant device, and an enhancement layer that improves the quality of the base layer video. The enhancement layer can be truncated at any bit location, resulting in a loss of visual quality, but reducing the needed bit rate.

The base layer is traditionally encoded with MPEG-4 visual simple profile [17, 18]. However, other encoding schemes for the base layer are suggested in [19–22]. As such, FGS can be seen as an enhancement scheme on top



(a)

(b)

(I)DCT: (inverse) discrete cosine transformation
$Q(^{-1})$: (inverse) quantization
MC: motion compensation
ME: motion estimation
VLC: variable-length coding
VLD: variable-length decoding
$SEA(^{-1})$: (inverse) selective enhancement algorithms (bit-plane manipulations)

Figure 1: (a) MPEG-4 FGS encoder and (b) MPEG-4 FGS decoder with shifting logic.

of existing codecs delivering additional quality and enabling various additional features.

When referring to FGS encoding, we actually refer to the encoding of the enhancement layer. For the encoding of the FGS enhancement layer, different techniques are possible. Within MPEG, bit-plane DCT residue encoding [23] was chosen.

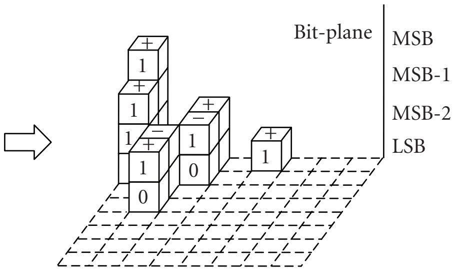
The enhancement layer receives as input the DCT residue values from the macroblock of a frame,[1] that is, the values obtained after subtracting the dequantized coded DCT coefficients from the original DCT coefficients (Figure 1(a)). The resulting residue matrix inherits all characteristics of a DCT matrix. The difference with more traditional video encoding schemes is a novel approach to encode the residual values. The FGS bit-plane DCT residue value encoding occurs by zig-zag scanning the values and by placing them in their *binary* form in a matrix (Figure 2). Note that the sign of a value is stored separately, so only the absolute value is used for the binary representation. A *bit plane* is one row in this matrix, thus a sequence of 256 bits (in case of 16 × 16 macroblocks as used in MPEG-4 visual simple profile). Finally, the bit planes are translated to unique symbols encoded by a *variable-length coding* (VLC) technique.

It is possible to (partially) *drop* bit planes, that is, not completely encoding or transmitting the bit plane, and as a result, using less bits for the enhancement layer. If one or more bit planes are dropped, the reconstructed values at the decoder side are less precise. These less accurate values can still be used to rebuild the macroblocks of the frame, but with a drop in visual quality. When dropping bit planes, large residue values are more likely to have a more correct reconstructed value than small residue values as the larger values will have bits in the upper, nondiscarded, bit planes.

MPEG-4 FGS natively supports the selective improvement of the visual quality within a frame, hence the ROI coding. This is realized by executing an additional operation on the binary matrix representation before bit-plane dropping and works on the complete DCT residue matrix of a macroblock. A *shift* operation is performed on all matrix values, that is, multiplying the matrix coefficient values with 2^{α} (α is the number of bit planes). Because the binary representation

[1] MPEG-4 traditionally uses the term video object plane (VOP) instead of a frame. Because our algorithms are independent of the actual used video encoding technology, we prefer to use the generic term “frame” throughout this paper.

10	0	3	0	1	0	0	⋯
7	0	−2	0	0	0	0	
0	−2	0	0	0	0	0	
0	2	0	0	0	0	0	
0	0	0	0	0	0	0	
0	0	0	0	0	0	0	
0	0	0	0	0	0	0	
⋮							⋱



MSB	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	⋯	0	0
MSB-1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	⋯	0	0
MSB-2	1	0	1	0	0	1	0	1	1	0	0	1	0	0	0	0	⋯	0	0
LSB	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	⋯	0	0

Figure 2: Bit-plane representation of a DCT residue matrix.

of these shifted values appears in higher bit planes, they are placed more in front during encoding. This improves the probability that they will be present in the received—and probably truncated—enhancement layer. Hence, the probability increases for all values within this DCT matrix to be reconstructed with a higher precision. As a result, this operation can be used to support ROI. Obviously, the adequate information about the shifting is added to the resulting bitstream so the decoder can perform the inverse transformation. This is represented by the SEA^{-1} block in Figure 1(b).

The algorithms described in the next section are generically applicable and are not solely usable in MPEG-4 FGS, as they can be implemented in other codecs. We have chosen to use MPEG-4 FGS merely to clearly demonstrate and visualize the results of the object tracking algorithm presented in this paper.

4. OBJECT TRACKING

4.1. Introduction

One or more objects in a video scene can be identified manually or automatically through object segmentation techniques and the macroblocks can be placed into an ROI. Usually, these objects will move in successive frames. We want to follow the movement of the objects so that we can automatically relocate the ROI. This process is called *object tracking*.

During the encoding of the base layer with an MPEG-4 visual simple profile encoder, the motion vectors are calculated. This information is used to determine the motion of the tracked object, represented by the *object motion vector*. This is illustrated by the dashed arrow in Figure 1(a). The object motion vector is used to move the ROI mask accordingly and, as such, to follow the motion of the object. Furthermore, as objects become larger or smaller, for example due to camera zooming, the mask must grow or shrink simultaneously with the object within.

In this section, new algorithms are introduced to enable object tracking and object motion. It works on a frame per frame basis; for each frame, the following steps are executed.

(i) First, the macroblocks that are part of the ROI are identified as explained in Section 4.2.

(ii) Next in Section 4.3, the motion of the object within the ROI is determined, using the motion estimation information from the identified macroblocks. This results in the translation of the ROI.

(iii) Then, the ROI is resized automatically for optimal object fitting by using the motion estimation information. This is explained in Section 4.4.

(iv) Finally, to demonstrate the algorithm in MPEG-4 FGS, the relevant macroblocks of the updated ROI are identified and these are shifted by the shift value α. This is done by reusing the techniques explained in Section 4.2.

To end this section, a *cloaking* technology is introduced in Section 4.5 and the time complexity of the algorithms is given in Section 4.6. Pseudocode listings as an illustration to these algorithms are given in the appendix of this paper.

4.2. Selecting the macroblocks

The existing object tracking techniques in the compressed domain as discussed in Section 2 work on complete macroblocks. Also the default ROI functionality in MPEG-4 FGS utilizes a mask that is aligned with the macroblock boundaries of a frame. This results in a direct mapping that determines whether the macroblock is inside or outside the ROI. However, as macroblocks are, for example, 16 × 16 pixels, these rather large blocks of pixels are not suitable for fine-detailed object tracking.

We use a different and novel approach as we utilize an *enhanced ROI mask* that works as a separate floating layer on top of the frame and the macroblocks within. As a result, this enhanced mask is no longer aligned with the boundaries of the macroblocks. As such, our algorithms are independent of the size of a macroblock which is determined by the video encoding specification.

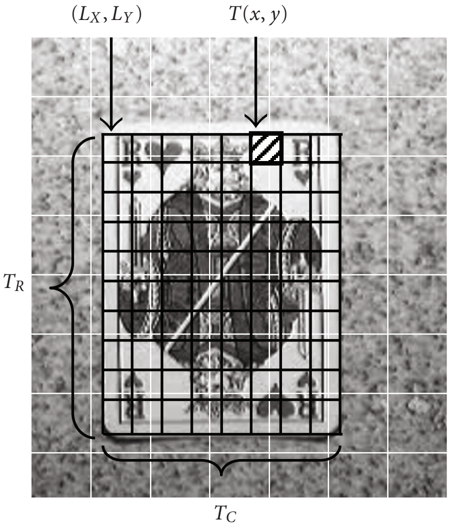


FIGURE 3: A part of a frame where the card is selected as the ROI. The ROI mask has dimensions $T_C \times T_R = 8 \times 10$. An element of the ROI mask is notated as $T(x, y)$. The upper left corner of the ROI mask has the coordinates (L_X, L_Y). The ROI mask is a floating layer on top of the frame and it is not aligned to the boundaries of the macroblock (represented by the white grid).

The enhanced ROI mask is represented by the matrix T with dimensions $T_C \times T_R$; T_C is the number of columns and T_R is the number of rows in the enhanced ROI mask. The value of an element in matrix T—referred to as $T(x, y)$—represents the shift value α that has to be applied to the corresponding macroblock(s). Note that if $T(x, y) = 0$, no shifting will occur, allowing to create an arbitrary-shaped ROI holding arbitrary-shaped objects. Further, an element $T(x, y)$ can represent any block size; here we use 8×8 pixels as block size which is a quarter of the default 16×16 pixels block size of macroblocks in MPEG-4 FGS. This creates a more fine-grained ROI mask, allowing a more neatly fitting selection of objects. The discussed algorithms can easily be modified to cope with any other size. In the remainder of the paper, we will assume 8×8 pixels as block size for the ROI mask elements and frame macroblocks of 16×16 pixels.

As the enhanced ROI mask floats on top of the frame, its location is also required. This is specified in pixel coordinates (denoted as L_X and L_Y) and locates the upper left corner of the mask. Note that this location does not need to be aligned to a macroblock boundary. A visual representation of the ROI mask as layer on top of the frame is shown in Figure 3.

Because of the decoupling of the ROI mask from the macroblocks, a mapping algorithm is required to determine which macroblocks are overlapped by the ROI. Each element $T(x, y)$ in the enhanced ROI mask matrix T overlaps with at least one macroblock, denoted as $m_{i,j}$. The indices i and j are calculated by the formulas

$$i = \left\lfloor \frac{L_X + 8x}{16} \right\rfloor, \qquad j = \left\lfloor \frac{L_Y + 8y}{16} \right\rfloor, \tag{1}$$

where $i = 0, 1, \dots, M - 1$ and $j = 0, 1 \dots, N - 1$ (M being the number of macroblocks in a frame horizontally and N being the number of macroblocks in a frame vertically).

The computed macroblock $m_{i,j}$ is tagged as part of the ROI and receives the shift value $T(x, y)$. As $T(x, y)$ represents a block of 8×8 pixels, it is possible that $T(x, y)$ overlaps with other macroblocks, namely $m_{i+1,j}$, $m_{i,j+1}$, and $m_{i+1,j+1}$ (Figure 4). In Figure 4(b), the 8×8 pixel block represented by $T(x, y)$ overlaps with macroblock $m_{i,j}$ and macroblock $m_{i+1,j}$. This can be mathematically expressed by the conditions $(L_X + 8x) \bmod 16 > 8$ and $(L_Y + 8y) \bmod 16 \le 8$. Other overlaps can be determined in a similar way. All macroblocks that are overlapped receive the shift value $T(x, y)$, hence in case of Figure 4(b), the macroblocks $m_{i,j}$ and $m_{i+1,j}$ are set to this shifting value. If a macroblock already received a shifting value, then the highest shifting value is used. In case of Figure 4(b), the macroblock $m_{i,j}$ receives the highest shifting value from the ROI mask elements $T(x - 2, y - 1)$, $T(x - 1, y - 1)$, $T(x, y - 1)$, $T(x - 2, y)$, $T(x - 1, y)$, $T(x, y)$, $T(x - 2, y + 1)$, $T(x - 1, y + 1)$, and $T(x, y + 1)$ assuming that $x - 2 \ge 0$, $y - 1 \ge 0$, and $y + 1 < T_C$.

The time complexity of the described algorithm to determine which macroblocks are overlapped by one ROI mask element $T(x, y)$ is $\mathcal{O}(1)$: the calculation of i and j with the given formulas and the determination of overlaps with any neighboring macroblocks are done with a constant time complexity. Note that all multiplication and division operations to select and to shift the macroblocks can be executed by a (fast) binary bit-shift operation.

4.3. Object motion

Object tracking by using the motion vectors of the object is only possible if the motion vector field is available. Traditional base layer encoders, like implementations of the MPEG-4 visual simple profile specification, do not calculate the motion vector field of intracoded frames. As our algorithms need this vector field, this limitation of the encoder implementation can be circumvented by different strategies. For example, the encoder can be slightly modified, with minor overhead, in such a way that motion estimation is performed—hence the determination of the motion vector field—even for intracoded frames. The motion estimation can use the latest available frame as reference. As such, the dashed arrow in Figure 1 always provides the object motion vector field, also for intracoded frames. Note that the motion vector field for intracoded frames does not need to be stored into the resulting bit stream, so this modification of the encoder implementation does not affect the resulting encoded video bit stream. Because there is at least one solution available, this and following sections assume that the motion vector field for the current frame is available.

In order to determine the motion of the object, the object motion vector (OMV) is calculated and is used to translate the upper left corner of the ROI mask resulting in adjusting the L_X and L_Y values of the ROI mask. The OMV is calculated using the steps explained in the following paragraphs.

For each $T(x, y)$ element, four motion vectors are collected from the motion estimation. Which motion vectors

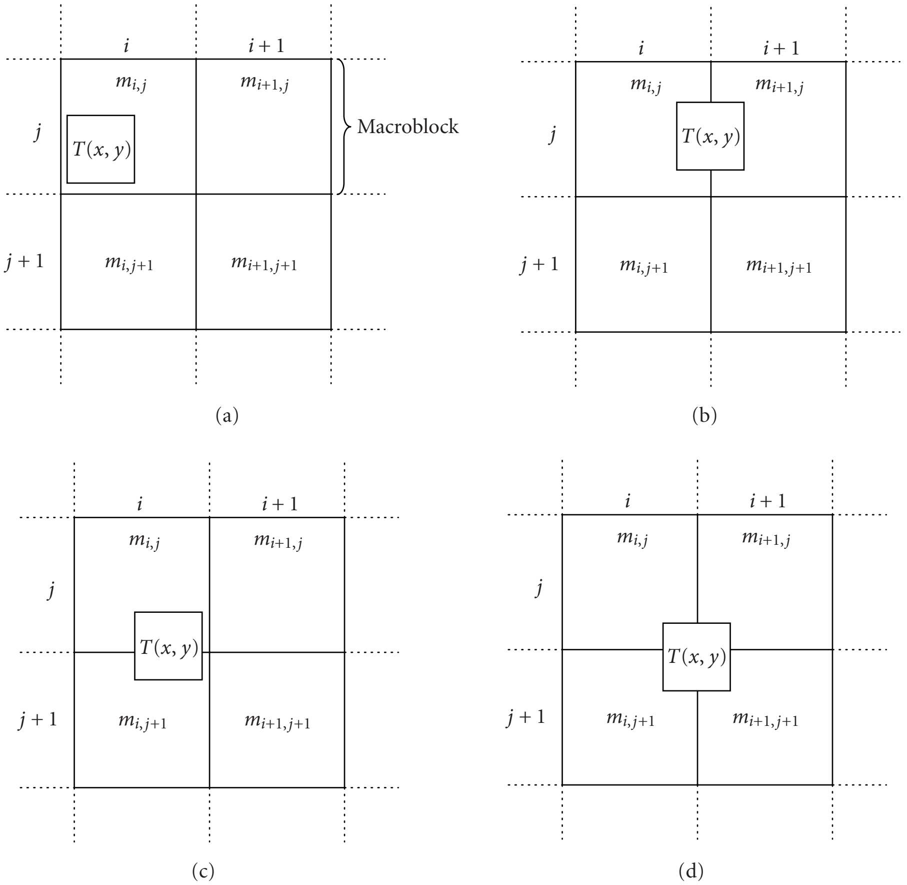


FIGURE 4: Mapping of the ROI mask element $T(x, y)$ onto the macroblock $m_{i,j}$. (a) The ROI mask element fits completely in the macroblock $m_{i,j}$. (b), (c), and (d) the ROI mask element overlaps with neighboring macroblocks.

are chosen depends on the location of $T(x, y)$ as shown in Figure 5 (motion estimation is in 4MV modus[2]). As a result, four subblocks are selected and the motion vectors thereof are denoted as the vectors $MV^1_{x,y}$ to $MV^4_{x,y}$. The selection of the motion vectors is independent of the accuracy of the motion vector; the implementation in MPEG-4 FGS uses half-pixel-element (*pel*) precision as MPEG-4 visual simple profile is used for the base layer.

Next, four *overlapping percentages* (P^1 to P^4) are calculated. They express the overlap of an element $T(x, y)$ with the four selected subblocks. In 4MV mode, first the horizontally (d_x) and vertically (d_y) overlaps of $T(x, y)$ with the first subblock are determined. This is illustrated by Figure 6 and is calculated using the formulas

$$\begin{aligned} d_x &= 8 - ((L_X + 8x) \bmod 8) = 8 - (L_X \bmod 8), \\ d_y &= 8 - ((L_Y + 8y) \bmod 8) = 8 - (L_Y \bmod 8). \end{aligned} \tag{2}$$

The d_x and d_y values allow the calculation of the overlapping percentages P^γ:

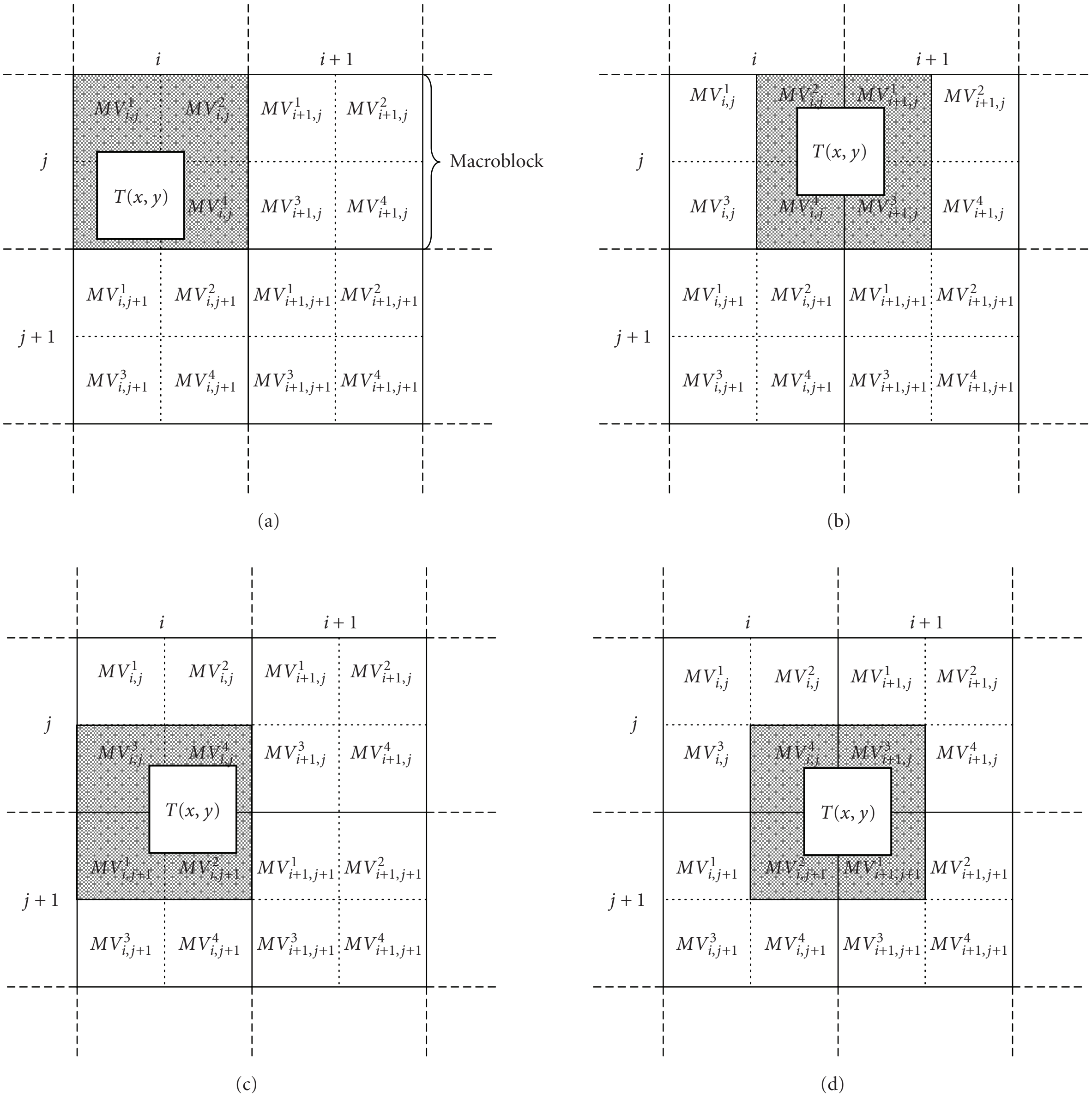
$$\begin{aligned} P^1 &= \frac{d_x d_y}{64}, \\ P^2 &= \frac{(8 - d_x) d_y}{64}, \\ P^3 &= \frac{d_x (8 - d_y)}{64}, \\ P^4 &= \frac{(8 - d_x)(8 - d_y)}{64}. \end{aligned} \tag{3}$$

These percentages P^γ are identical for all elements of the ROI mask T. This is because we work with a block size of 8 × 8 pixels for the ROI mask elements.

Next, the object motion vector for the $T(x, y)$ element is determined ($\text{OMV}_{x,y}$) as the weighted sum of the motion vectors $MV^\gamma_{x,y}$ and the overlapping percentages P^γ by

$$\text{OMV}_{x,y} = \frac{\sum_{\gamma=1}^{4} (MV^\gamma_{x,y}) P^\gamma}{\sum_{\gamma=1}^{4} P^\gamma}. \tag{4}$$

[2] The 4MV mode divides a macroblock in 4 8 × 8 blocks (*subblocks*). The motion vectors of the four subblocks are calculated. For the macroblock $m_{i,j}$, these are notated as $MV^\gamma_{i,j}$, $\gamma = 1, 2, 3, 4$. 1MV mode—where only one motion vector per macroblock is calculated—can also be used by the algorithm.



(a)

(b)

(c)

(d)

FIGURE 5: Selecting the four appropriate motion vectors for ROI mask element $T(x, y)$, based on the overlap with the macroblocks.

Finally, the OMV is calculated as the average of all $\text{OMV}_{x,y}$ vectors:

$$\text{OMV} = \frac{\sum_{x=0}^{T_C-1} \sum_{y=0}^{T_R-1} \text{OMV}_{x,y}}{T_C T_R}. \quad (5)$$

The resulting motion vector is the overall object motion vector of the ROI mask. This vector is used to adjust the L_X and L_Y values.

Formula (4) has two additional constraints. These constraints were added after evaluating and testing the algorithms and affect the overall results (see Section 6). The first constraint removes the influence of motion vectors of too small overlapping areas. If P^y is smaller than a threshold λ, the value of P^y is substituted by zero. This explains the reason of the divisor in (4).

The second constraint removes the influence of the motion vectors being part of the border of an ROI mask as these could become less reliable after some iterations of the algorithm. If $T(x, y)$ is part of the border of the ROI mask, the vector $\text{OMV}_{x,y}$ is set to the null vector. In this case, the divisor in (5) is decreased with the number of ROI mask elements that fulfill with constraint (C.2).

These constraints can be expressed as follows:

(C.1) $P^y = 0$ if $P^y < \lambda$;

(C.2) $\text{OMV}_{x,y} = \vec{0}$ if ($x = 0$ or $x = T_C - 1$) and $T_C > 2$;
$\text{OMV}_{x,y} = \vec{0}$ if ($y = 0$ or $y = T_R - 1$) and $T_R > 2$.

With regard to the time complexity to compute the OMV vector, the sums in formula (5) are decisive and result in $\mathcal{O}(n)$ with n being the total number of elements of the ROI mask matrix T, that is, the calculation is linear to the number of ROI mask matrix elements. Indeed, an $\text{OMV}_{x,y}$ vector in formula (4) is calculated in $\mathcal{O}(1)$ as all $MV^{\gamma}_{x,y}$ and all P^{γ} (with $\gamma = 1, \ldots, 4$) are determined in $\mathcal{O}(1)$ for arbitrary $T(x, y)$. For $MV^{\gamma}_{x,y}$, this is proven by the time-complexity analysis in Section 4.2. The P^{γ} values are calculated by simple straightforward computations, hence $\mathcal{O}(1)$. In addition, these P^{γ} values are actually calculated only once as they are equal for all elements of the ROI mask.

4.4. *Object resizing*

Object motion, as discussed in the previous subsection, captures the global motion of the object inside the ROI. During this motion, it is also possible that the "size" of the object changes, due to for example camera zooming or a change in the relative distance of the object to the camera. As a result, the ROI must also change in size, otherwise the ROI will be too small or too large for the larger, respectively, smaller object. We call this *ROI resizing*.

In total, there are six operations the size of an ROI can undergo:

(1.a) reduction in width;
(1.b) stay equal in width;
(1.c) enlarge in width;

and

(2.a) reduction in height;
(2.b) stay equal in height;
(2.c) enlarge in height.

For every frame, one action is selected from (1.a), (1.b), and (1.c) and one action is selected from (2.a), (2.b), and (2.c). Both actions are executed as explained further in this section.

The first step to enable automatic resizing of the ROI is to determine the two appropriate actions. To do so, the motion of the boundaries of the ROI is calculated using the formulas

$$
\begin{aligned}
M_L &= \sum_{y=0}^{T_R-1} \frac{\text{OMV}^1_{0,y}}{T_R}, \\
M_R &= \sum_{y=0}^{T_R-1} \frac{\text{OMV}^1_{T_C-1,y}}{T_R}, \\
M_H &= M_L - M_R, \qquad (6)\\
M_U &= \sum_{x=0}^{T_C-1} \frac{\text{OMV}^2_{x,0}}{T_C}, \\
M_D &= \sum_{x=0}^{T_C-1} \frac{\text{OMV}^2_{x,T_R-1}}{T_C}, \\
M_V &= M_U - M_D, \qquad (7)
\end{aligned}
$$

$\text{OMV}^1_{x,y}$ represents the first vector component of the vector $\text{OMV}_{x,y}$; $\text{OMV}^2_{x,y}$ is the second vector component of this vector. M_L represents the horizontal motion of the left-hand side ROI boundary, M_R is the horizontal motion of the right-hand side ROI boundary, M_U is the vertical motion of the topmost boundary, and M_D denotes the vertical motion of the bottom boundary. M_H is the overall horizontal motion and M_V is the overall vertical motion.

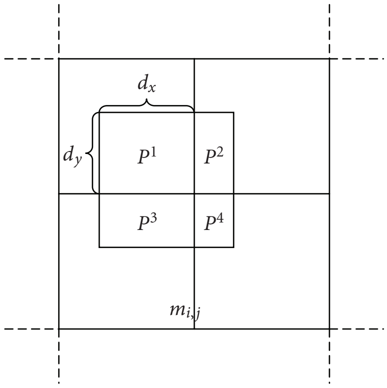


Figure 6: A macroblock $m_{i,j}$ divided into four 8 × 8 subblocks is overlapped by an $T(x, y)$ element. The figure shows how to determine d_x and d_y and the overlapping percentages P^1 to P^4.

Next, the two ROI resize actions are determined by the formulas

$$
\begin{aligned}
\text{Action}_1 &= \begin{cases} (1.\text{a}) : M_H < -\delta, \\ (1.\text{b}) : -\delta \leq M_H \leq \delta, \\ (1.\text{c}) : M_H > \delta, \end{cases} \\
\text{Action}_2 &= \begin{cases} (2.\text{a}) : M_V < -\delta, \\ (2.\text{b}) : -\delta \leq M_V \leq \delta, \\ (2.\text{c}) : M_V > \delta, \end{cases}
\end{aligned}
\qquad (8)
$$

δ is a threshold the motion of the ROI boundaries has to reach before the reducing or enlarging of the ROI mask takes place. A value of half the size of the block that a $T(x, y)$ element represents proves to be a good value (see Section 6).

Knowing the required actions, the resizing of the ROI can take place. In case of action (1.a), the ROI matrix T is replaced by a new matrix T' with dimensions $((T_C - 1) \times T_R)$. The shifting values of the new matrix T' are a linear combination of the values of matrix T, expressed by the formula

$$
T'(x, y) = \frac{(T_C - x - 1)T(x, y) + (x + 1)T(x + 1, y)}{T_C}, \qquad (9)
$$

where

$$
x = 0, 1, \ldots, T_C - 2, \qquad y = 0, 1, \ldots, T_R - 1. \qquad (10)
$$

This formula reduces the ROI matrix T with one column, however it is possible that this is insufficient. If $M_H + \delta < -\delta$, we repeat the reduction formula (9). A similar procedure is executed in case of action (2.a).

In case of operations (1.b) and (2.b), the ROI matrix T remains invariant.

Finally, in case of an enlarging size, the ROI matrix T is replaced by a new matrix T' with dimensions $((T_C+1)\times T_R)$ in case of (1.c) and $(T_C\times(T_R+1))$ in case of (2.c). The values of new matrix T' are a linear combination of the values of matrix T, analog to the formula (9). This enlarging must also be repeated if necessary.

After applying the appropriate resizing actions, a surplus between $-\delta$ and δ remains. This value is added to the results of the calculation of (6) and (7) for the next frame, in other words we cumulate the motion of the ROI boundaries over multiple frames.

The time complexity of the algorithm to resize the ROI is determined as follows. First, the algorithm calculates the horizontal and vertical motions (i.e., M_H and M_V) according to the formulas (6) and (7). These formulas aggregate over, respectively, the number of rows and the number of columns of the ROI mask. This results in a time complexity of $\mathcal{O}(T_R)$ for formula (6) and $\mathcal{O}(T_C)$ for formula (7). The outcome of the formulas is used to determine the resizing action of the ROI according to the formula (8). With regard to the time complexity, this finding does not alter the time complexity. Next, the actual resizing of the ROI matrix is performed, dependent on the type of action determined in the previous step. If no resizing occurs (actions (1.b) and (2.b)), no additional steps must be performed. In case of a vertical reduction (action (1.a)), the formula (9) must be executed. This formula creates a new matrix in $\mathcal{O}((T_C-1)\cdot T_R)=\mathcal{O}(T_C\cdot T_R)=\mathcal{O}(n)$, that is, linear to the number of elements in de ROI matrix. For all other actions ((1.c), (2.a), and (2.c)), the time complexity can be deduced in a similar way, thus $\mathcal{O}(n)$ for each action. When combining two resize actions or a resize action that must be repeated several times, the overall time complexity remains $\mathcal{O}(n)$.

Adding the time complexity $\mathcal{O}(T_R)$ to calculate formula (6) and the $\mathcal{O}(T_C)$ to calculate formula (7) to the time complexity for the resizing actions ($\mathcal{O}(n)$), the time complexity for the ROI resizing algorithm is $\mathcal{O}(n)$. In the optimal case (if no resizing is done), the time complexity is $\Omega(m)$, $m=\max(T_C,T_R)$, thus linear to the maximum number of rows or columns of the ROI mask matrix.

4.5. Cloaking

The algorithms discussed in this section enable the creation of an ROI mask T, containing elements $T(x,y)$ that represent blocks of 8×8 pixels. By setting a shift value to zero, it is possible to create arbitrary-shaped ROI masks. However to determine the object motion as described in Section 4.3, the motion vectors for all $T(x,y)$ elements are determined and used in the overall object motion vector calculations. If the object inside the ROI is not rectangular, the calculation of the OMV uses all motion vectors within this ROI, even those that are not associated with this object. This is not desirable as these are associated with another object which might exhibit another motion trajectory. In addition, sometimes it is desirable to differentiate between the area to improve the visual quality and the object that is being tracked as the creation of a region slightly larger than the object itself improves visual perception.

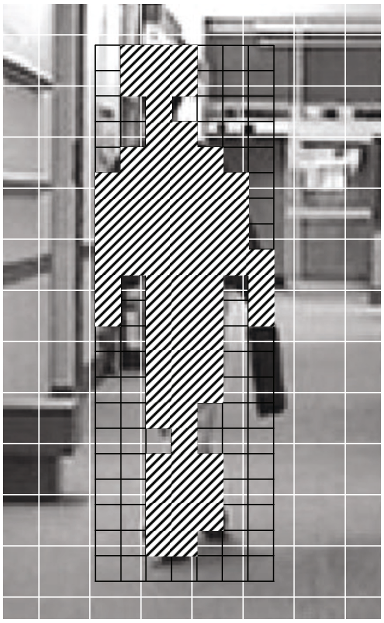


Figure 7: A part of a frame of the video sequence “hall monitor.” The black grid represents the cloaking layer which coincides with the ROI mask matrix T. The hatched parts indicate the cloaking matrix elements larger than zero. The motion vectors of the parts that are not hatched will not be used to determine the overall object motion OMV.

To solve these concerns, an additional *cloaking* layer is used that allows full separation of the visually important region and the object that is being tracked.

The cloaking layer is represented by a new matrix C which has identical dimensions as matrix T and consists of binary values indicating if the $\text{OMV}_{x,y}$ vector must be taken into account in the overall object motion vector calculations. The determination of the values in the matrix C is done by a manual selection, similar to the determination of the object that must be followed, that is, the initial determination of matrix T and the coordinates (L_x,L_y). This results in a third constraint for formula (4):

(C.3) $\text{OMV}_{x,y}=\vec{0}$ if $C(x,y)=0$.

Similar as for constraint (C.2), the divisor in (5) is decreased with the number of the matrix element that fulfills constraint (C.3).

In Figure 7 a part of a frame is shown of the “hall monitor” sequence (see Figure 8 for the complete frame). The white grid represents the macroblocks, the black grid is the enhanced ROI mask matrix T and the cloaking matrix C, which coincide. The hatched parts indicate the cloaking matrix elements larger than zero. Only the motion vectors of the hatched parts will be used to determine the overall object motion OMV.

For the macroblock selection and object tracking algorithms, nothing changes when adding the cloaking matrix C, except for the overall object motion formula which receives the additional constraint (C.3). The ROI resizing algorithm

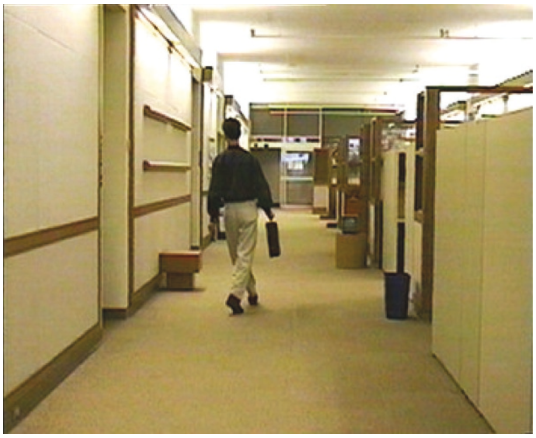


Figure 8: A picture from the "hall monitor" test sequence.

is extended in such a way that not only matrix T is adjusted to the new size, but also cloaking matrix C. This is done identical to the creation of matrix T' as discussed in the previous section resulting in a new matrix C'.

The cost in terms of time complexity for adding an additional cloaking layer is relevant during the resize operation. As the same resizing formulas are executed as for the ROI mask matrix T, the time complexity is the same. Hence, the time complexity of resizing the ROI matrix T and the cloaking matrix C is $\mathcal{O}(2n) = \mathcal{O}(n)$.

4.6. *Time-complexity analysis*

To conclude this section, an overall time-complexity analysis of our algorithms is presented. Our algorithms enabling automatic object tracking in the compressed domain are based on two principles: *object motion* as described in Section 4.3 and *object resizing*—resulting in *ROI resizing*—as described in Section 4.4.

For every frame, both algorithms are executed. As demonstrated in the previous sections, both have a (worst-case) time complexity of $\mathcal{O}(n)$, meaning that the time complexity is linear to the total number of elements n in the ROI mask matrix. Because both algorithms are sequential, the global time complexity for our algorithms is also $\mathcal{O}(n)$.

As such, using our lightweight object tracking algorithms implies adding an additional time cost that is linear to the size of the object we want to track. Note that the size of the object is normally smaller than a complete frame. Furthermore, performance analysis of the MPEG-4 reference software encoder and MPEG-4 FGS reference software encoder in [24, 25] shows that the encoder needs tens of millions of operations per second. As such, our lightweight algorithms are only a very small fraction of the total required encoding time. Hence, object tracking in the compressed domain by using the presented algorithms is feasible, even for real-time and streaming applications, on the condition that the object is known through the matrices T and C, and the initial coordinates L_x and L_y.

Finally, it must be noted that the object tracking algorithms do not add any complexity to the video decoder. All algorithms, and associated time complexity, are part of the video encoder.

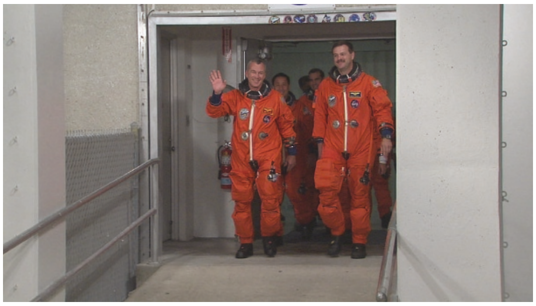


Figure 9: A picture from the "crew" test sequence.

5. MATERIALS AND METHODS

In the remainder of the paper, we evaluate the accuracy of our lightweight object tracking algorithms. First, we created a video of a playing card moving from right to left against a nonuniform background, while the camera is zooming in. The first frame shows the playing card at the right-hand side. Throughout the frames, the card moves towards and outside the left boundary of the frame, until only a small piece is visible in the last frame. Meanwhile, the card is more than doubled in size due to the camera zooming operation. The video has a resolution of 320×240 resulting in 20×15 macroblocks and has a length of 501 frames. A picture from the video sequence is shown in Figure 3.

We also used two well-know test sequences, namely "hall monitor" (consisting of 101 frames with a CIF resolution of 352×288) and "crew" (consisting of 50 frames with an HD resolution of 1280×720). One can see a picture of the former in Figure 8 and a picture of the latter in Figure 9. The first sequence was chosen because the object being tracked is scaled down, the second sequence was chosen because of the larger resolution and the many moving objects.

To test the algorithms, we need to validate them. We asked four different persons who have no visual defects to indicate for each frame the smallest possible rectangular region containing the object, that is, the card for the first test sequence, the man on the left in "hall monitor" sequence and the man on front right in the "crew" sequence. All macroblocks that hold (a part of) this region are marked and stored as the *object indication* for the given frame. The object indication for the first frame was given as an input, the remaining frames were tagged manually. From the four resulting sets of object indications, we distilled one reference set for each test sequence. In the next section, the construction of these sets are discussed.

The same video feed is used as input video sequence for the object tracking algorithm. It receives a matrix T with the initial location (L_X, L_Y) so that the selection of macroblocks as explained in Section 4.2 results in an object indication identical to the object indication that was given for the first frame for the manual marking. In the first sequence, the playing card fits perfectly in the ROI mask, hence the cloaking matrix C is completely filled with values larger than 0. For the other two sequences, the cloaking matrix C is used so that only the motion vectors of the human form are taken into account. Figure 7 depicts the cloaking matrix for the "hall monitor" sequence. All values of the matrix T are $\alpha = 9$. Next, different parameters were used to investigate their optimal settings and their influence on the results:

(i) *(C.1)*: $\lambda = 0.00$, $\lambda = 0.05$, $\lambda = 0.10$, $\lambda = 0.15$, $\lambda = 0.20$, $\lambda = 0.25$;
(ii) δ *in formula* (8): $\delta = 2$, $\delta = 4$, $\delta = 8$, and $\delta = 16$;
(iii) *applying and not applying (C.2)*.

It is necessary that $\lambda \leq 0.25$ in constraint (C.1). If not and the ROI mask is located in such a way that none of the overlapping percentages are larger than λ, then constraint (C.1) is never met, and consequently OMV will be the null vector. As a result, the values L_X and L_Y do not change, and the ROI mask will not move. For all subsequent frames, the same event will occur, keeping the ROI mask at the same location.

All different combinations are encoded with an I(P*) GOP[3] structure. During the encoding, for each frame the macroblocks that are marked for selective enhancement by the algorithms are logged.

6. RESULTS AND DISCUSSION

6.1. Results

First, we elaborate on the results of the manual object indication for the given video, and this by comparing the different indications given by the four persons. To create the reference set for the first test sequence, 473 object indications can be selected in a trivial manner as there is a majority in the four-object-indication sets. In order to have for each of the 500 frames one-object indication, 27 object indications must be added to the reference set. To do this, we determine the person that contributed the most indications to the 473 object indications and we select the remaining 27 object indications from this person. This results in a reference set of 500 object indications, for each frame one indication.

The test reference sets for the "crew" and "hall monitor" sequences are determined in a similar way.

It can be observed that selecting the "correct" region is, even for humans, a difficult task. Further investigation of the results shows that most often, the four opinions only differ in one row or one column of macroblocks.

Next, the logs that were created during the encoding of the video are compared to the constructed reference set and compared to the reference set accepting a minor dissimilarity of one row or one column. The criterion for evaluation is the percentage of identical object indications: the higher this value is, the better the automatic algorithm performs. This result in two measurements for each of the different test sequences:

(M.1) compare to the reference set;
(M.2) compare to the reference set and accept a dissimilarity of one row or one column as correct.

The results for the three sequences can be found in Table 1.

6.2. Discussion

The relationship of the results for (M.1) and (M.2) is as expected. Obviously, the results that accept a dissimilarity of one row or one column as correct (M.2) are always better than the cases were such dissimilarity is not accepted as correct (M.1).

We observe that the value of λ does not have a significant influence on the results. Because formula (4) weights the value of the motion vector $MV^{\gamma}_{x,y}$ with overlapping percentage P^{γ}, the influence of small overlapping areas is automatically reduced.

We also observe that our algorithms are also insensitive to sudden motion changes. Indeed, if the object stops, the calculation of the object motion vector will result in a null vector as the motion vectors associated with the object will also be the null vector. Hence, the ROI mask will not be displaced. If the object starts moving again, the motion vectors will reflect this motion and the object motion vector will reflect the displacement of the object, regardless of the direction of this last motion. In fact, the object that is being tracked in the "hall monitor" sequence stops for a few frames and continues to move in another direction.

The table also shows the influence of parameter δ. The parameter δ determines how soon the matrix T must be resized. The necessity of this parameter is proven by the fact that the results for $\delta = 4$ are superior for all sequences. Hence, this parameter is independent of the kind of resizing, namely an object enlarging in the "card" and "crew" sequences and an object reduction in the "hall monitor" sequence. The biggest difference is shown when (C.2) is enabled. For the "card" sequence, the improvement for using $\delta = 4$ instead of $\delta = 8$ ranges from 56.6% to 61.8%. When comparing $\delta = 4$ to $\delta = 16$, the difference is even more clear. Setting the parameter $\delta = 8$ or $\delta = 16$ makes the resizing algorithm too slow to react properly. In the case of the "card" test video, this usually means that the matrix T is too small as the algorithm does not counter the zoom-in operation immediately. δ should be set to the optimal value of 4. In case of the "crew" sequence, the *enlarging* of the traced object itself occurs more steadily and slowly, when compared to the playing card of the first test sequence. Hence the need to resize in two consecutive frames is rare; the resizing is more distributed over multiple frames. As such, if the algorithm misses a resize for one frame, it can correct this in the next frame. This explains the smaller influence of δ parameter. Setting δ too small (e.g., $\delta = 2$) results in an unsteady resizing.

[3] The GOP structure specifies the sequence of frame types. In this notation, I(P*) means that the encoded video starts with an I-frame, followed by only P-frames.

Table 1: Test results (%).

Parameters		Card				Hall monitor				Crew			
		C.2		Not C.2		C.2		Not C.2		C.2		Not C.2	
δ	λ	M.1	M.2	M.1	M.2	M.1	M.2	M.1	M.2	M.1	M.2	M.1	M.2
2	0.00	9	18	8	13	5	36	5	37	18	68	14	70
	0.05	8	18	8	13	5	36	5	36	16	68	14	66
	0.10	10	19	8	14	5	36	5	36	16	68	14	68
	0.15	10	19	7	14	5	36	5	37	16	70	8	48
	0.20	11	20	7	14	4	38	5	38	8	46	8	42
	0.25	11	20	7	19	4	38	6	44	14	68	14	70
4	0.00	75	96	30	59	11	85	10	85	14	78	14	78
	0.05	75	96	30	60	16	87	10	85	14	78	14	78
	0.10	76	96	31	61	12	86	10	85	14	78	14	78
	0.15	76	96	32	60	21	88	10	85	14	76	14	76
	0.20	74	96	30	60	20	86	11	86	14	72	14	72
	0.25	71	96	33	60	20	87	12	86	14	72	14	72
8	0.00	15	34	14	36	2	34	2	42	10	72	10	72
	0.05	15	34	14	36	2	34	2	42	10	72	10	72
	0.10	15	34	14	36	2	34	2	42	10	72	10	72
	0.15	15	36	14	36	2	34	2	44	10	72	10	72
	0.20	15	35	14	35	2	34	2	42	10	72	10	72
	0.25	15	34	14	34	2	34	2	44	10	72	10	72
16	0.00	14	28	14	29	2	20	2	26	10	48	10	48
	0.05	14	28	14	29	2	20	2	26	10	48	10	48
	0.10	14	28	14	30	2	20	2	25	10	48	10	48
	0.15	14	28	13	30	2	20	4	60	10	48	10	48
	0.20	14	28	13	30	2	21	2	25	10	46	10	46
	0.25	14	28	13	30	2	20	2	25	10	46	10	46

```
for each frame do
    // Start procedure
    // Initialization frame
    Algorithm 2
    OMV = calculate OMV using Algorithm 3

    // Translated mask
    $L_X = L_X + \mathrm{OMV}^1$
    $L_Y = L_Y + \mathrm{OMV}^2$

    // Perform resize actions
    Algorithm 5
    Algorithm 6

    // Determine macroblock shifting
      Values
    Algorithm 7
end
```

Algorithm 1: Main algorithm.

```
// Calculate $d_x$, $d_y$
$d_x = 8 - (L_X \bmod 8)$
$d_y = 8 - (L_Y \bmod 8)$

// Calculate $P^y$ and $\sum_{y=1}^{4} P^y$
$P^1 = (d_x * d_y)/64$
$P^2 = (d_x * (8 - d_y))/64$
$P^3 = ((8 - d_x) * d_y)/64$
$P^4 = ((8 - d_x) * (8 - d_y))/64$
if $P^1 < \lambda$ then $P^1 = 0$
if $P^2 < \lambda$ then $P^2 = 0$
if $P^3 < \lambda$ then $P^3 = 0$
if $P^4 < \lambda$ then $P^4 = 0$
$P_{\mathrm{sum}} = P^1 + P^2 + P^3 + P^4$
```

Algorithm 2: Initialization for each frame.

The table also reveals the big impact of constraint (C.2) for the "card" sequence, in particular when δ is set to 4. It is clear from the test that the border of an ROI mask is less reliable, especially after some iterations. To avoid the same event to occur when $\lambda > 0.25$ as explained in Section 5, the constraint (C.2) is never applied if the ROI mask is too small in size. This is realized by (C.2), its conditions $T_C > 2$, and $T_R > 2$. Constraint (C.2) should always be enabled in combination with $\delta = 4$ as this always generates better results.

However the latter depends on the content of the video itself. As the test video has a growing object due to a camera

```
// For all matrix elements $T$
for $x = 0$ to $T_C - 1$ do
    for $y = 0$ to $T_R - 1$ do
        // Get $MV^1_{x,y}$ using Algorithm 4
        // Get $MV^2_{x,y}$ using Algorithm 4
        // Get $MV^3_{x,y}$ using Algorithm 4
        // Get $MV^4_{x,y}$ using Algorithm 4

        // Calculate formula (4)
        $\mathrm{OMV}_{x,y} = (MV^1_{x,y} + MV^2_{x,y} + MV^3_{x,y} +$
        $MV^4_{x,y})/P_{\mathrm{sum}}$

        // Apply constraint (C.2)
        if $(((x = 0)$ or $(x = T_C - 1))$ and
        $(T_C > 2))$ or $(((y = 0)$ or
        $(y = T_R - 1))$ and $(T_R > 2))$ then
            $\mathrm{OMV}_{x,y} = \vec{0}$
            C2Counter++
        end
    end
end

// Calculate overall object motion
   result.
for $x = 0$ to $T_C - 1$ do
    for $y = 0$ to $T_R - 1$ do
        $\mathrm{OMV} += \mathrm{OMV}_{x,y}$
    end
end
$\mathrm{OMV} = \mathrm{OMV}/((T_C * T_R) - \mathrm{C2Counter})$
```

Algorithm 3: Calculate OMV.

```
// Determine $MV_{x,y}$
if $(L_X + 8 * d_x) \bmod 16 \le 8$ and
$(L_Y + 8 * d_y) \bmod 16 \le 8$ then
    // Figure 5(a)
    $MV^1_{x,y} = MV^1_{i,j}$
    $MV^2_{x,y} = MV^2_{i,j}$
    $MV^3_{x,y} = MV^3_{i,j}$
    $MV^4_{x,y} = MV^4_{i,j}$
else if $(L_X + 8 * d_x) \bmod 16 > 8$ and
$(L_Y + 8 * d_y) \bmod 16 \le 8$ then
    // Figure 5(b)
    $MV^1_{x,y} = MV^2_{i,j}$
    $MV^2_{x,y} = MV^1_{i+1,j}$
    $MV^3_{x,y} = MV^4_{i,j}$
    $MV^4_{x,y} = MV^3_{i+1,j}$
else if $(L_X + 8 * d_x) \bmod 16 \le 8$ and
$(L_Y + 8 * d_y) \bmod 16 > 8$ then
    // Figure 5(c)
    $MV^1_{x,y} = MV^3_{i,j}$
    $MV^2_{x,y} = MV^4_{i,j}$
    $MV^3_{x,y} = MV^1_{i,j+1}$
    $MV^4_{x,y} = MV^2_{i,j+1}$
else if $(L_X + 8 * d_x) \bmod 16 > 8$ and
$(L_Y + 8 * d_y) \bmod 16 > 8$ then
    // Figure 5(d)
    $MV^1_{x,y} = MV^4_{i,j}$
    $MV^2_{x,y} = MV^3_{i+1,j}$
    $MV^3_{x,y} = MV^2_{i,j+1}$
    $MV^4_{x,y} = MV^1_{i+1,j+1}$
end
```

Algorithm 4: Determine $MV_{x,y}$.

zooming operation, setting δ larger than the optimal value will result in a too small matrix T. Therefore, if the matrix T is smaller than the genuine object, adding the motion vectors of the elements on the edges of matrix T means incorporating the motion vectors of the actual object being tracked; this is good. For video sequences where the matrix T is larger than the actual object, not enabling constraint (C.2) gives a worse result as the motion vectors outside the actual object are taken into account.

The reason we do not observe this large influence for the two other sequences is because of the usage of the cloaking matrix C. Indeed, because the used cloaking matrix—such as the one depicted in Figure 7—removes most macroblocks that are part of the border, the enabling or disabling of constraint (C.2) hardly influences the results for these sequences.

To conclude, the most optimal settings for a video sequence are

(i) *(C.1)*: $\lambda \le 0.25$;

(ii) *(C.2)*: enabled or use a cloaking matrix C;

(iii) δ: 4.

Our automatic algorithm reaches near perfection when comparing to the reference set and allowing one row or column mismatch (M.2) for the "card" sequence: 96.1% on average over the six possible values for λ. Comparing to the reference set without allowing one row or column mismatch (M.1) gives an average result of 74.5% for this first test sequence.

For the "hall monitor" sequence, using the optimal settings, our algorithms achieve on average 86.5% when allowing one row or column mismatch (M.2). However, if a mismatch is not allowed (M.1), the results are much lower in comparison to the first sequence. The good results for (M.2) indicate that mostly there is only one row or one column mismatch. Looking more into detail, we see that the automatic algorithm marks the object too large, hence one column or one row too much. While this restrains the percentages for (M.1), the visual perception is not negatively influenced by enhancing a little more than the object. Furthermore, the mismatch does not propagate over successive frames, so the algorithms are still useful.

Finally, the "crew" sequence achieves also a good result when using the optimal settings: on average 75.7% when allowing one row or column mismatch (M.2). While these

```
$M_H$ = calculate formula (6)

// Perform resize actions horizontal
if $M_H < -\delta$ then
    // Action (1.a)
    repeat
        for $x = 0$ to $T_C - 2$ do
            for $y = 0$ to $T_R - 1$ do
                $T'(x, y) = \frac{(T_C - x - 1)T(x, y) + (x + 1)T(x + 1, y)}{T_C}$
            end
        end
        $M_H = M_H + \delta$
    until $M_H > -\delta$
end
if $M_H > \delta$ then
    // Action (1.c)
    repeat
        for $x = 0$ to $T_C$ do
            for $y = 0$ to $T_R - 1$ do
                $T'(x, y) = \frac{(T_C - x)T(x, y) + (x)T(x - 1, y)}{T_C}$
            end
        end
        $M_H = M_H - \delta$
    until $M_H \leq \delta$
end
```

Algorithm 5: Perform resize actions horizontal.

```
$M_V$ = calculate formula (7)

// Perform resize actions vertical
if $M_V < -\delta$ then
    // Action (2.a)
    repeat
        for $x = 0$ to $T_C - 1$ do
            for $y = 0$ to $T_R - 2$ do
                $T'(x, y) = \frac{(T_R - y - 1)T(x, y) + (y + 1)T(x, y + 1)}{T_R}$
            end
        end
        $M_V = M_V + \delta$
    until $M_V > -\delta$
end
if $M_V > \delta$ then
    // Action (2.c)
    repeat
        for $x = 0$ to $T_C - 1$ do
            for $y = 0$ to $T_R$ do
                $T'(x, y) = \frac{(T_R - y)T(x, y) + (y)T(x, y - 1)}{T_R}$
            end
        end
        $M_V = M_V - \delta$
    until $M_V \leq \delta$
end
```

Algorithm 6: Perform resize actions vertical.

results are lower than the "card" sequence—particularly when comparing to (M.1)—, it must be noted that this sequence is far more complex as it contains more (moving) objects, a moving background with similar texture and colorization of the object that is being tracked, abrupt luminance changes due to flash photography, and so on.

Nevertheless, these results illustrate the usability of our algorithms to enable a lightweight and real-time object tracking in the compressed domain.

7. CONCLUSIONS

In this paper, we have discussed novel algorithms allowing a video encoder to automatically track an object. These algorithms have a very low time complexity which is linear to the size of the object, making them very useful for real-time and streaming applications. We make use of the motion vector field calculated by the encoder's motion estimation algorithms to capture the overall motion of the tracked object. We also introduced an algorithm that allows an encoder to cope with the "enlargement" and "shrinking" of an object. All our algorithms are capable of tracking any kind of objects in a video stream. Furthermore, our algorithms are not bound to the (relatively large) size of a macroblock; we use a fine grid on top of a frame so the algorithms can track any object in a more detailed way.

All our novel methods presented in this paper are generic and can be implemented in any codec that calculates the motion vector field. However, the results depend on the specific kind of the motion estimation algorithm of the video encoder. For testing and evaluating purposes, we have implemented the algorithms within the MPEG-4 FGS reference software codec. We used the region-of-interest capabilities of this codec to visualize the results of the algorithms. A second layer was introduced enabling us to differentiate between the ROI and the (possibly smaller) objects within. Three test sequences were used to evaluate the influence of various parameters. The results of our algorithms were compared to manually constructed reference sets, so an optimal parameter set was determined.

From these results, we can conclude that these novel lightweight algorithms are capable of tracking objects in complex scenes, can handle scaled down or scaled up objects, and are independent of the resolution of the video stream.

APPENDIX

PSEUDOCODE ALGORITHMS

The pseudocode listings in Algorithms 1, 2, 3, 4, 5, 6, and 7 are given as an illustration of the above mentioned algorithms.

```
// For all matrix elements $T$
for $x = 0$ to $T_C - 1$ do
    for $y = 0$ to $T_R - 1$ do
        // Determine $i$ and $j$
        $i = \text{trunc}((L_X + 8 * x)/16)$
        $j = \text{trunc}((L_Y + 8 * y)/16)$

        // Set shifting values to
           macroblocks
        if $(L_X + 8 * d_x) \bmod 16 \leq 8$ and
        $(L_Y + 8 * d_y) \bmod 16 \leq 8$ then
            // Figure 4(a)
            $m_{i,j} = \max(m_{i,j}, T(x, y))$
        else if $(L_X + 8 * d_x) \bmod 16 > 8$ and
        $(L_Y + 8 * d_y) \bmod 16 \leq 8$ then
            // Figure 5(b)
            $m_{i,j} = \max(m_{i,j}, T(x, y))$
            $m_{i+1,j} = \max(m_{i+1,j}, T(x, y))$
        else if $(L_X + 8 * d_x) \bmod 16 \leq 8$ and
        $(L_Y + 8 * d_y) \bmod 16 > 8$ then
            // Figure 5(c)
            $m_{i,j} = \max(m_{i,j}, T(x, y))$
            $m_{i,j+1} = \max(m_{i,j+1}, T(x, y))$
        else if $(L_X + 8 * d_x) \bmod 16 > 8$ and
        $(L_Y + 8 * d_y) \bmod 16 > 8$ then
            // Figure 5(d)
            $m_{i,j} = \max(m_{i,j}, T(x, y))$
            $m_{i+1,j} = \max(m_{i+1,j}, T(x, y))$
            $m_{i,j+1} = \max(m_{i,j+1}, T(x, y))$
            $m_{i+1,j+1} = \max(m_{i+1,j+1}, T(x, y))$
        end
    end
end
```

Algorithm 7: Determine shifting values for all macroblocks.

ACKNOWLEDGMENTS

The authors would like to thank Tom Caljon of the Vrije Universiteit Brussel, Department of Electronics and Informatics for his valuable input and testing during the writing of this paper. The research activities that have been described in this paper were funded by Ghent University, the Interdisciplinary Institute for Broadband Technology (IBBT), the Institute for the Promotion of Innovation by Science and Technology in Flanders (IWT), the Fund for Scientific Research-Flanders (FWO-Flanders), the Belgian Federal Science Policy Office (BFSPO), and the European Union.

REFERENCES

[1] A. J. Lipton, H. Fujiyoshi, and R. S. Patil, "Moving target classification and tracking from real-time video," in *Proceedings of the 4th IEEE Workshop on Applications of Computer Vision (WACV '98)*, pp. 8–14, Princeton, NJ, USA, October 1998.

[2] M. van der Schaar and Y.-T. Lin, "Content-based selective enhancement for streaming video," in *Proceedings of IEEE International Conference on Image Processing (ICIP '01)*, vol. 2, pp. 977–980, Thessaloniki, Greece, October 2001.

[3] H. Wang and S.-F. Chang, "A highly efficient system for automatic face region detection in MPEG video," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 4, pp. 615–628, 1997.

[4] C. Bregler, "Learning and recognizing human dynamics in video sequences," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 568–574, San Juan, Puerto Rico, USA, June 1997.

[5] A. Cavallaro, O. Steiger, and T. Ebrahimi, "Semantic video analysis for adaptive content delivery and automatic description," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 10, pp. 1200–1209, 2005.

[6] O. Sukmarg and K. R. Rao, "Fast object detection and segmentation in MPEG compressed domain," in *Proceedings of IEEE Region 10 Annual International Conference on TENCON (TENCON '00)*, vol. 3, pp. 364–368, Kuala Lumpur, Malaysia, September 2000.

[7] S.-Y. Chien, Y.-W. Huang, and L.-G. Chen, "Predictive watershed: a fast watershed algorithm for video segmentation," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 5, pp. 453–461, 2003.

[8] S. Dasiopoulou, V. Mezaris, I. Kompatsiaris, V.-K. Papastathis, and M. G. Strintzis, "Knowledge-assisted semantic video object detection," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 15, no. 10, pp. 1210–1224, 2005.

[9] V. Mezaris, I. Kompatsiaris, N. V. Boulgouris, and M. G. Strintzis, "Real-time compressed-domain spatiotemporal segmentation and ontologies for video indexing and retrieval," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 14, no. 5, pp. 606–621, 2004.

[10] M. Isard and A. Blake, "Contour tracking by stochastic propagation of conditional density," in *Proceeding of 4th European Conference on Computer Vision (ECCV '96)*, vol. 1, pp. 343–356, Cambridge, UK, April 1996.

[11] W.-N. Lie and R.-L. Chen, "Tracking moving objects in MPEG-compressed videos," in *Proceedings of IEEE International Conference on Multimedia and Expo (ICME '01)*, pp. 1172–1175, Tokyo, Japan, August 2001.

[12] R. Achanta, M. Kankanhalli, and P. Mulhem, "Compressed domain object tracking for automatic indexing of objects in MPEG home video," in *Proceedings of IEEE International Conference on Multimedia and Expo (ICME '02)*, vol. 2, pp. 61–64, Lusanne, Switzerland, August 2002.

[13] S.-M. Park and J. Lee, "Object tracking in MPEG compressed video using mean-shift algorithm," in *Proceedings of the Joint Conference of the 4th International Conference on Information, Communications and Signal Processing and the 4th Pacific-Rim Conference on Multimedia (ICICS-PCM '03)*, vol. 2, pp. 748–752, Singapore, December 2003.

[14] L. Favalli, A. Mecocci, and F. Moschetti, "Object tracking for retrieval applications in MPEG-2," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, no. 3, pp. 427–432, 2000.

[15] W. Li, "Overview of fine granularity scalability in MPEG-4 video standard," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 11, no. 3, pp. 301–317, 2001.

[16] J.-R. Ohm, "Advances in scalable video coding," *Proceedings of the IEEE*, vol. 93, no. 1, pp. 42–56, 2005.

[17] F. Pereira and T. Ebrahimi, Eds., *The MPEG-4 Book*, Prentice-Hall, Englewood Cliffs, NJ, USA, 2002.

[18] A. Puri and T. Chen, Eds., *Multimedia Systems, Standards and Networks*, Marcel Dekker, New York, NY, USA, 2000.

[19] J. Ascenso and F. Pereira, "Drift reduction for a H.264/AVC fine grain scalability with motion compensation architecture," in *Proceedings of International Conference on Image Processing (ICIP '04)*, vol. 4, pp. 2259–2262, Singapore, October 2004.

[20] M. Domański, L. Blaszak, and S. Maćkowiak, "AVC video coders with spatial and temporal scalability," in *Proceedings of Picture Coding Symposium (PCS '03)*, pp. 41–46, Saint Malo, France, 2003.

[21] K. Ugur, G. Louizis, P. Nasiopoulos, and R. Ward, "Extremely fast selective enhancement method for fine granular scalable enabled H.264 video," in *Proceedings of IEEE Canadian Conference on Electrical and Computer Engineering (CCECE '03)*, vol. 2, pp. 1103–1106, Montreal, Canada, May 2003.

[22] K. Ugur and P. Nasiopoulos, "Combining bitstream switching and FGS for H.264 scalable video transmission over varying bandwidth networks," in *IEEE Pacific Rim Conference on Communications Computers and Signal Processing (PACRIM '03)*, vol. 2, pp. 972–975, Victoria, BC, Canada, August 2003.

[23] F. Ling, W. Li, and H. Sun, "Bitplane coding of DCT coefficients for image and video compression," in *Visual Communications and Image Processing*, vol. 3653 of *Proceedings of SPIE*, pp. 500–508, San Jose, Calif, USA, January 1999.

[24] F. Cavalli, R. Cucchiara, M. Piccardi, and A. Prati, "Performance analysis of MPEG-4 decoder and encoder," in *Proceedings of 4th EURASIP-IEEE International Symposium on Video/Image Processing and Multimedia Communications (VIPromCom '02)*, pp. 227–231, Zadar, Croatia, June 2002.

[25] O. Lehtoranata and T. D. Hämäläinen, "Complexity analysis of spatially scalable MPEG-4 encoder," in *IEEE International Symposium on System-on-Chip*, pp. 57–60, Tampere, Finland, November 2003.

Robbie De Sutter received the M.S. degree in computer science from Ghent University, Belgium, in 1999. He joined the Multimedia Lab in 2001, where he obtained his Ph.D. degree in 2006. His research interests include video coding technologies, usage-context modeling and negotiation, and content adaptation.

Koen De Wolf received the M.S. degree in computer science from Ghent University, Belgium, in 2003. In that year, he joined the Multimedia Lab, where he is currently working towards the Ph.D. degree. His research interests include video coding technologies, more in particular scalable video coding, interlayer prediction, and scalable motion information.

Sam Lerouge received his M.S. degree in computer science from Ghent University, Belgium in 2001. Since then, he started working towards the Ph.D. degree in the Multimedia Lab, which he obtained in 2005. His research focused on maximizing the visual quality in constrained environments. Since 2006, he is working as a Project Manager for the Regionale Media Maatschappij, which supports two local television channels, where he is focusing on digital TV applications.

Rik Van de Walle received his M.S. and Ph.D. degrees in engineering from Ghent University, Belgium in 1994 and 1998, respectively. After a visiting scholarship at the University of Arizona (Tucson, USA), he returned to Ghent University, where he became Professor of multimedia systems and applications, and Head of the Multimedia Lab. His current research interests include multimedia content delivery, presentation and archiving, coding and description of multimedia data, content adaptation, and interactive (mobile) multimedia applications.