Multimed Tools Appl
DOI 10.1007/s11042-011-0831-0

Sprite-based generation of side information for multi-view Distributed Video Coding

Lucian Ciobanu · Luís Côrte-Real

Abstract Generation of side information for multi-view Distributed Video Coding in multi-camera environments (e.g., video surveillance) poses challenges in scenarios with temporary non-overlapping among views and consequently, no resources for generating the side information at some time instants. In this paper we extend our previous work (Ciobanu and Corte-Real, Multimed Tools Appl 48(3):411–436, 2010) (for scenarios with permanent complete-overlapping among views) and propose a solution to this problem by exploiting the past visual data associated with each view, gathered over time as a panoramic image (sprite). The entire collection of temporal data from all the cameras is subsequently used for generating the side information. We tackle several topics related to these scenarios and propose solutions for the encountered issues. Optimization techniques are also discussed, e.g., temporal tags and block alternatives associated with the sprite contents for an improved generation of side information. This paper also presents a post-processing technique for additional refinement of generated side information. Practical results show an overall significant enhancement of side information by over 2 dB.

Keywords Distributed Video Coding (DVC) · Wyner-Ziv (WZ) · Side information generation · Scale-Invariant Feature Transform (SIFT) · SIFT-based multi-view registration (SMVR) · Global motion estimation (GME) · Sprites · Multi-view registration metric (MVRM) · Structural similarity (SSIM)

L. Ciobanu (✉) · L. Côrte-Real
Faculdade de Engenharia da Universidade do Porto/INESC Porto, Porto, Portugal
e-mail: lciobanu@inescporto.pt

L. Côrte-Real
e-mail: lreal@inescporto.pt

Published online: 21 June 2011

1 Introduction

Distributed Video Coding (DVC), as a particular paradigm of Distributed Source Coding (DSC), has been one of the most active research areas in the signal processing community in the last years, providing a revolutionary new perspective over the conventional video compression (e.g., the MPEGx, H.26x families). It has arisen in the favourable context of the ever increasing use of distributed architectures due to the emerging technical advances in the last decade, thus enabling the deployment of cheap, low-power sensing devices widely spread over large areas, from hand-held digital cameras to the omnipresent multimedia cellular phones.

The roots of Distributed Source Coding date back to the 1970s when the information-theoretic results of Slepian-Wolf in 1973 for lossless coding with side information at decoder side [18], and then in 1976 extended by Wyner-Ziv for lossy coding [21], have shown the conceptual importance of this distributed paradigm. As stated in theory, the same coding efficiency can be approached for architectures with independent (non-linked) encoders and joint decoding, as for the ideally joint encoding, at the cost of a vanishing error probability for long sequences. Consequently, when applied to Distributed Video Coding, it is an essentially reversed paradigm that enables the shift of the computational burden from encoder to decoder, as opposed to the conventional (e.g., H.26x, MPEGx), non-distributed coding.

Among the many challenging topics of Distributed Video Coding is the generation of side information, usually based on an elaborated correlation model that is essentially more complex when it comes to multi-view architectures [9, 10, 13, 14, 22]. There are currently two main techniques for generating the side information: trajectory-based motion interpolation (TMI) and hash-based motion estimation (HME) [4]. The former uses past and future reference frames, usually conventionally coded, in order to determine the motion vectors and finally to interpolate the frame in between. It proves to be highly reliable as long as the motion is sufficiently smooth and predictable, otherwise other methods are needed in addition. As for works implementing the latter technique [1, 2, 4, 16], the hash-based motion estimation, the encoder sends a signature or hash in order to feed the decoder with the necessary data for the motion estimation process, at the cost of an extra-load on encoder and communication channel.

In the multi-view setting the above mentioned trajectory-based motion interpolation is called intra-camera interpolation. Additionally, it is usually performed an inter-camera interpolation (spatial interpolation) between neighbour cameras. Most works use a fusion of both, intra-camera and inter-camera interpolation, or use mode decisions in order to achieve optimal results on side information generation [3, 13, 14].

Multi-view registration has been recently an important topic of research in computer vision, as fundamental task in distributed video processing applications. A few solutions towards automatic, generic and flexible camera registration were proposed, usually assuming some context-specific knowledge [5, 6, 11, 20].

The sprite coding, as proposed in [8, 12, 17, 19] and standardized in MPEG-4 [15], relies on pre-processing the video content by segmenting it in two parts: background object and foreground objects. A background sprite image is generated by global motion estimation (GME) at encoder, then encoded and sent to the decoder along with the independently encoded foreground objects. At decoder, the reconstructed

sprite image is merged with the decoded foreground objects and consequently, the whole sequence is eventually decoded.

Video surveillance is an important topic of research in computer vision with many uses in today's world, mainly to guarantee the safety of persons and sites of particular interest (e.g., airports, train stations, warehouses, shopping centers, school campuses, public places, etc.). As the number of employed cameras and their mobility play a critical role in the flexible monitoring of the environment, a system with high density and free motion of the cameras is of particular interest in video surveillance. The design of such a system is far from trivial and is expected to exploit the visual redundancy among views, by adapting the Distributed Video Coding to this scenario where the generation of side information plays a major role. One solution was proposed in [7]. However, it requires at least one reference camera to permanently capture the entire scene, in order to generate side information at any time instant.

In this paper we extend our previous work [7] and propose a solution for this limitation, based on additional exploiting of the temporal visual redundancy accumulated in time from all the cameras. It increases the flexibility of the multi-view setup (e.g., video surveillance, remote monitoring) in order to properly handle the temporary absence of overlapping among views and consequently, no resources for generating the side information at some time instants. The lack of other approaches in the literature to this scenario motivated the authors to propose this solution. To this end, the decoder gets an additional assignment of collecting and managing the past visual data for each individual view (target or reference), as a panoramic image (sprite) and some additional data. The decoder then uses the entire stored collection from all the cameras in order to provide an adequate side information for decoding a Wyner-Ziv (WZ) bitstream from a given target camera. However, the proposed solution is focused on the generation of side information and is codec-independent, i.e., the overall performance of the codec is beyond the scope of this paper.

Section 2 presents several techniques designed to manage the visual data collection in this scenario, including the sprite-based generation of side information. Section 3 describes a post-processing technique for additional refinement of generated side information. Section 4 shows the achieved results. Finally, the conclusions are drawn in Section 5.

2 Sprite-based techniques for multi-view distributed video coding

In this paper we added more capabilities to the decoder, i.e., it generates over time a panoramic image (sprite) associated with each view (target or reference). It collects each frame received from a given view by inserting it into the associated sprite, in the right location. Section 2.1 fully describes the updating process of each sprite. Figure 1 illustrates an example of generated sprite for the reference view. Section 4 shows more examples of generated sprites. The sprite generation is achieved regardless of the actual contents of the frames, i.e., without object segmentation.

Figure 2 depicts the new architecture, the added capabilities for sprite management are indicated by shaded blocks. Each individual sprite, corresponding to one

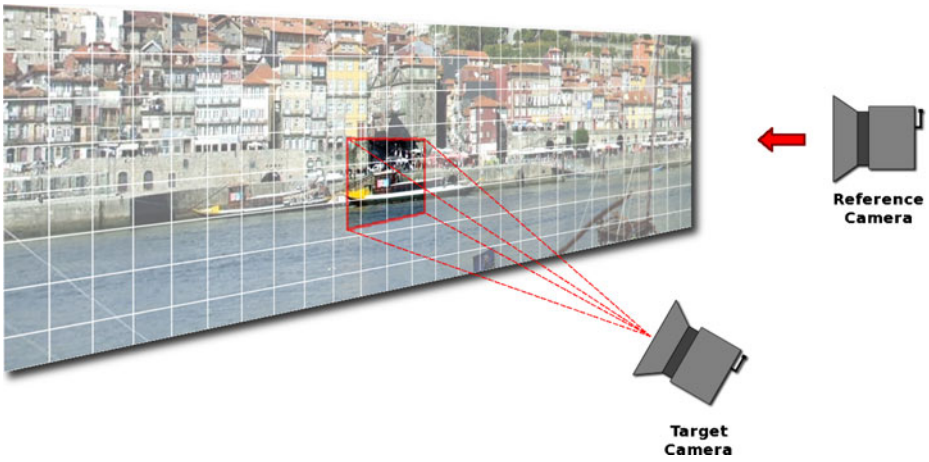


Fig. 1 Example of generated sprite image associated with the reference view. Correlation area corresponding to the current target view (the indicated *quadrilateral*)

of the views, is individually managed by the same processing technique, discussed in Section 2.1. Therefore, we illustrated two different instances corresponding to either the target view (see instance *Sprite Management* (TV_i)) or the reference view (see instance *Sprite Management* (RV_i)). For simplicity we represented only two cameras (one target and one reference). However, the system is designed for any number and combination of cameras.

The reference camera performs conventional encoding (e.g., H.26x, MPEGx) of its perceived view (the "scene" view), while the target camera conventionally encodes only the first frame and applies Wyner-Ziv (WZ) encoding to the remaining frames. Accordingly, the updating process of the target sprite is performed in two situations, as follows. First, when the decoder receives the first (conventionally coded) target frame (see the instance *Sprite Management* (TV_i) from upper-left area of Fig. 2). Secondly, each WZ decoded frame is inserted into the target sprite by the same instance, illustrated in the lower area.

The reference sprite is updated before the iterative decoding (see instance *Sprite Management* (RV_i) from upper-right area), i.e., a new reference frame is inserted into the sprite before the first multi-view registration. Moreover, the designed system can be set to perform in another mode: the instance *Sprite Management* (RV_i) may continuously update the reference sprite without any subsequent target frame decoding, e.g., in a first phase only the process flow of the reference view is functional, up to the block *Sprite Management* (RV_i), thus inserting into the reference sprite a specified number of frames (e.g., 1,000 frames) and then, the rest of the architecture can be activated in order to decode the frames from target view. At that time, the reference sprite already contains a panoramic image of the scene. Then, the same instance (*Sprite Management* (RV_i)) continues to insert one reference frame at a time, in synchronization with the rest of the architecture.

The second functional mode of the architecture was designed to demonstrate the proposed scenario discussed in Section 1. In a complex multi-camera environment (e.g., comprising dozens of cameras), after the initial generation of the reference

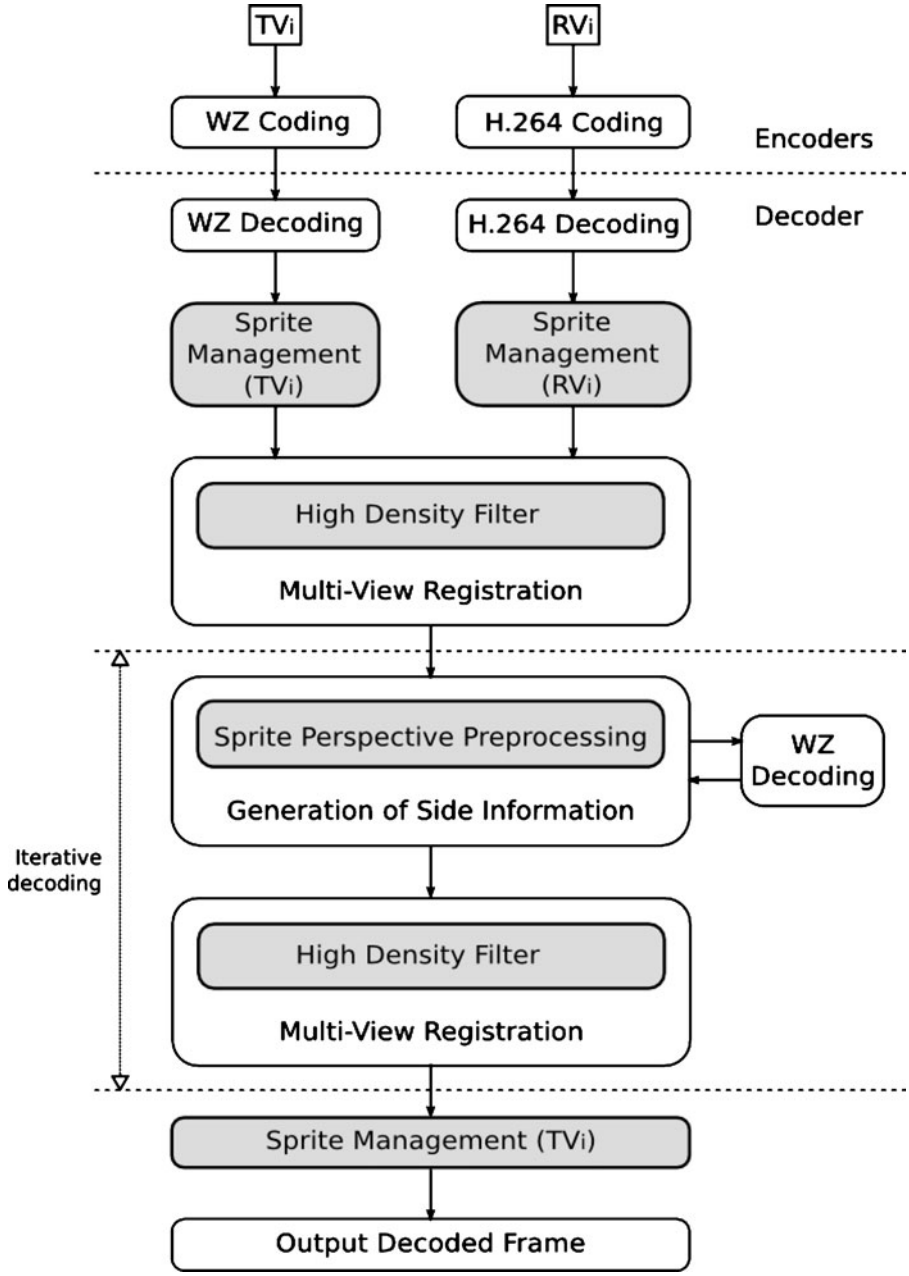


Fig. 2 Multi-camera architecture embedding the sprite management capabilities, as indicated by the *shaded blocks*. Illustrated process flow for each individual target view (TV_i) and reference view (RV_i)

sprites, the system may easily find overlapping between the current target view to be decoded and (at least) one of the reference sprites. Over time, the sprites associated with previous decoded target sequences may also be used.

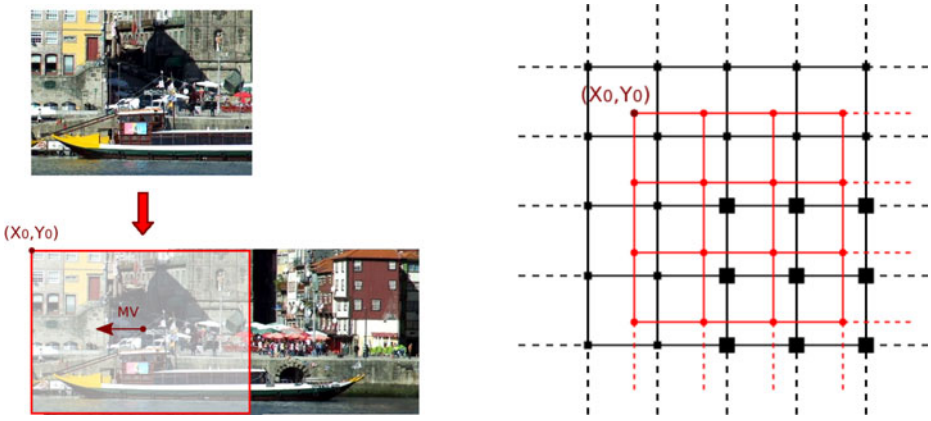
Concluding, the already built reference sprites are used for generation of side information and, the target cameras can also contribute to the generation of side information, i.e., their associate sprites are also used. There is one sprite associated with each camera and one generated side information for each sprite.

The next subsections provide more details and discussions, in order to describe the developed sprite-based techniques corresponding to the shaded modules from the architecture (see Fig. 2), as follows. Section 2.1 presents the generation of sprites. Section 2.2 discusses some additional processing specific to the target sprites. Section 2.3 details an appropriate filtering technique for the SIFT-generated keypoint matches, as part of the multi-view registration. Section 2.4 discusses a technique for optimization of generated side information by additionally storing past visual data associated with each sprite. Finally, Section 2.5 presents the actual generation of side information for each sprite.

2.1 Generation of sprite image

The process described in this subsection is performed by the module *Sprite Management*, illustrated in Fig. 2 as two separate instances: *Sprite Management* (TV_i) and *Sprite Management* (RV_i), as follows. A panoramic image (sprite) is generated on-the-fly at decoder for each camera, i.e., each frame received from a given camera is inserted into the corresponding sprite, as illustrated in Fig. 3a. The decoder management of each sprite assumes the following: (1) storage of the sprite image, (2) separately storage (as an image) of the last received frame and, (3) memorize the position (e.g., (X_0, Y_0)) in sprite where the last frame was inserted (the upper-left conner, see Fig. 3a). The latter two are considered additional data associated to the sprite image and they are updated when a new received frame is inserted into the sprite, as described in the following steps:

1. apply global motion estimation (GME) between the new frame to be inserted and the last stored one; the determined motion vector $MV(X, Y)$ is used to



(a) Sprite management at decoder (b) Sub-pixel insertion of a new frame

Fig. 3 Addition of a new frame to the corresponding sprite

update the memorized position (X_0, Y_0), as follows: $X_0 = X_0' + X$, $Y_0 = Y_0' + Y$, where (X_0', Y_0') is the previous memorized position
2. insert the new frame into the sprite, in the new position (X_0, Y_0) determined in Step 1(see Fig. 3a); the insertion process is detailed below
3. the new frame is also stored separately as an image (the previous stored frame is removed)

The accuracy of sprite generation has a direct impact on the subsequent generation of side information based on it. Therefore, for each newly added frame it is determined a motion vector with sub-pixel precision. Consequently, the determined coordinates of the pixels of the new frame are float values, including (X_0, Y_0). They are illustrated as round-shaped points in Fig. 3b. The actual pixels of the sprite, represented as square-shaped, have integer coordinates.

The round-shaped pixels are considered virtual and serve only as a guidance, since they cannot be stored into the sprite image (as float values). Consequently, are selected those square-shaped pixels from sprite which have at least sixteen round-shaped neighbours (e.g., the bigger square-shaped pixels from Fig. 3b). Each such pixel is initialized by bicubic interpolation applied on its sixteen (round-shaped) neighbours and therefore, the insertion process of a new frame into the sprite is accomplished.

Note that the (X_0, Y_0) coordinates are always stored as float values and serve only as a guidance to the next frame to be added. It is considered the virtual upper-left conner of the inserted frame. The actual upper-left conner initialized in sprite is the bigger upper-left square-shaped pixel from Fig. 3b.

Initially, the sprite image is blank and the (X_0, Y_0) coordinates are set to a predefined value (e.g., $(0, 0)$). The Step 1 is skipped for the first received frame. During the sprite generation process, every new frame may partially overwrite previous contents from the sprite image.

Section 4 presents two examples of sprites generated with several sub-pixel precisions. As concluded, a precision of up to 1/10 of a pixel is considered adequate for all the video sequences from the dataset.

2.2 Enhancement of target sprites based on additional Intra-blocks

The reference camera provides high quality frames at decoder due to the conventional encoding (e.g., H.26x, MPEGx) and consequently, better quality of the generated sprite (see Figs. 12 and 13). However, the sprites associated to target cameras have poorer quality due to the WZ coding involved. In order to enhance these particular sprites, we propose a solution, as described next.

The target camera (encoder) sends to decoder a percentage of the blocks per frame as conventional Intra-coded, instead of DVC (e.g., for a 320×240 frame size, 8×8 block size, and 1% amount per frame, there are $\frac{320}{8} \cdot \frac{240}{8} \cdot \frac{1}{100} = 12$ Intra-blocks per frame). Additionally, are also sent the positions of these Intra-blocks within the frame. At decoder, each Intra-block is inserted into the corresponding position in the decoded frame. The WZ decoding is only performed for the remaining block positions. The Intra-block positions are randomly chosen by encoder with the requirement to avoid repetitions in consecutive frames as much as possible (e.g., for a 1% amount of Intra-blocks per frame, there is no repetition in the first 100 frames).

(a) First frame (b) Second frame

Fig. 4 Example of consecutive decoded frames of the "Serralves" sequence: the first frame in **a** and the second frame in **b**

Figure 4 shows the first two decoded frames of the "Serralves" video sequence. The target camera encodes the first frame as conventional Intra, illustrated in Fig. 4a. The second decoded frame (see Fig. 4b) is composed by both Intra and WZ blocks. For illustration purposes it was chosen a lower quality WZ coding and 10% of Intra-blocks per frame. The Intra-blocks can be easily noticed in Fig. 4b, mainly in the lower half of the frame (in the grass area).

Every new frame added to a sprite may overwrite already existent sprite contents. In order to preserve older Intra-blocks in sprite from one added frame to another, as they provide more quality than WZ blocks, we propose the following technique. A temporal tag (counter) is associated to each pixel from sprite. Initially (when the sprite is blank), all the counters are set to a predefined value (e.g., 10,000). For each added frame all the sprite counters are incremented and therefore, the higher is the counter value the older is the sprite pixel, as illustrated in Fig. 5. A low counter value is perceived as a high priority of the pixel, i.e., the pixel will be persistent in sprite for a longer time.

Another set of counters is also associated by decoder to the pixels of the added frame, as follows: 0 (zero) for pixels from Intra-blocks and a predefined value (e.g., 10,000) for pixels from WZ blocks. When adding the new frame into the sprite, the overwrite of a sprite pixel is only permitted if the counter of the frame pixel is lower (higher priority) than the counter of the sprite pixel (lower priority). If so, the counter of the frame pixel is also copied into the sprite. However, given the sub-pixel handling of the added frame and the subsequent bicubic interpolation of the sprite's overwritten pixel (see Section 2.1), the counter of the "frame pixel" is actually computed as the maximum of the counters of the four nearest neighbours from the added frame (illustrated as round-shaped points in Fig. 3b), i.e., the shorter persistence among the four nearest neighbours is chosen. For instance, the counter 0

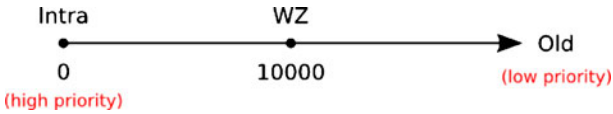


Fig. 5 Timeline for temporal tags (counters) associated with the sprite pixels

(zero) is associated with a sprite pixel only if it is surrounded by four Intra-pixels, otherwise the four neighbours belong to a mixture of Intra and WZ blocks and therefore, the lower priority among them is advised (the higher counter value of the four nearest neighbours).

The target camera conventionally encodes the first frame as Intra and consequently, at decoder side the frame will be added to the corresponding sprite with all counters set to 0 (see Fig. 19a). The percentage of Intra-blocks is only applied for the remaining WZ frames.

This technique assures the persistence of the Intra-blocks in sprite (e.g., an Intra-block will be persistent for 10,000 added frames, unless is overwritten by other Intra-block). Over time, given a small amount of Intra-blocks per frame, the overall quality of the sprite is significantly enhanced depending on the camera movement (e.g., a better quality is achieved for slower movement of the camera or for repeatedly reviewing the same scene areas). Also, the random distribution of the Intra-blocks and the requirement to avoid repetitions, provide a uniform enhancement of the sprite.

In this subsection we described some additional processing specific only to the target sprites, i.e., it is performed only by the *Sprite Management* (TV_i) instance (see Fig. 2). Nevertheless, the rest of the techniques discussed in this paper are designed for both the target and reference sprites.

Section 4 presents a few examples of generated sprites based on Intra-blocks (see Figs. 18 and 19). As concluded, a percentage of up to 10% of Intra-blocks is considered adequate for all the video sequences from the dataset.

2.3 Sprite-based multi-view registration

In this paper we use a sprite to substitute the reference frame in the multi-view registration process described in [7]. Figure 1 illustrates an example of overlapping area on a sprite.

In the scenario described in this paper there is a significant discrepancy in size between the target frame and an ordinary sprite, and continuous change in sprite's size and shape. In order to improve the accuracy of the multi-view registration, we propose in this subsection an appropriate filter technique for SIFT-generated keypoint matches.

In sprite-based multi-view registration, the end-points (from the sprite) of the correct keypoint matches use to be isolated in a small area (see Fig. 1). Consequently, we developed a technique to track the highest density of end-points in the sprite. To this end, we propose an iterative method with a predefined maximum number of iterations (e.g., 3). Each iteration brings more accuracy on locating the assumed high density of correct end-points, as follows. All end-points are initially selected on the first iteration. On each iteration are selected the end-points that will participate in the next iteration, regardless of the selection from the previous iteration. These are the steps for each iteration:

1. determine the density point (the point having the average coordinates of the previously selected end-points)
2. calculate the distance from each end-point to the density point
3. determine the maximum distance, divide it into a predefined number of equal units (e.g., 3) and calculate the unit length; then

4. for each distance calculate the minimum (integer) number of such units so as the sum of their lengths is greater or equal to the distance; associate the calculated number of units to the end-point
5. determine the number of units (N_U) that is mostly encountered among end-points
6. select the end-points that have associated the determined number of units (N_U) or less (closer to the density point); they will participate in the next iteration
7. if it's not the third iteration then go to Step 1 (start a new iteration); otherwise, stop iterations!

Figure 6 illustrates an example of one iteration. It is considered the first iteration and consequently, all the end-points are previously selected. After determining the density point (the cross-shaped point) in Step 1, and calculating each distance in Step 2, the maximum distance corresponds to the far distant point from the right. This distance is divided into 3 equal units (in Step 3). It is determined in Step 5 that most points (7 out of 9) are close within distance of up to 1 unit length from the density point. These points are selected in Step 6. For illustration purposes, the area around the density point surrounding the selected points is indicated as dashed circle (with radius of 1 unit length). In the next iteration, the two unselected points from the right are ignored.

The number of selected end-points decreases from one iteration to another. The set of end-points selected in the last iteration is considered the result of the filter method: the correct end-points. The remaining ones are assumed outliers.

The filtering of the SIFT-generated keypoint matches is considered a pre-processing step before the multi-view registration (SMVR) and is performed by the *High Density Filter* module (see Fig. 2). An instance of the same SMVR processing is independently performed for each sprite, i.e., the same target view is simultaneously tracked in each sprite, as illustrated in Fig. 1.

2.4 Sprite-associated block alternatives

As previously discussed in this section, a collection of past visual data may be achieved by generating a sprite for each view. As a result, various large perspectives of the same scene are gradually generated over time. However, the scene contents may change at any time due to temporary occlusions (e.g., objects passing through the scene) which may affect some of the views (e.g., a reference camera may be occluded while a target camera is not; consequently, for decoding the target

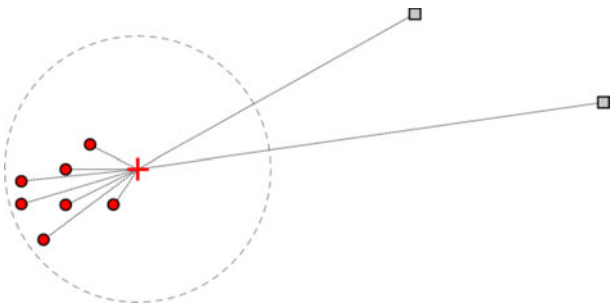


Fig. 6 One iteration of the method for tracking the highest density of end-points. The density point is indicated as *cross-shaped point*, the selected end-points as *round-shaped points*, the unselected end-points as *square-shaped points*

sequence, the reference sprite is not suitable for generating the side information due to the contained occlusion). Section 4 discusses such an example (see Fig. 11).

In this subsection we present an optimization technique to be applied on each individual sprite, in order to exploit better the eventual scene changes. Some additional data is attached to each sprite, as follows. First, a fixed grid is associated to the sprite. It virtually divides the sprite in equal square blocks of predefined size (e.g., 8×8), as shown in Fig. 1. For illustration purposes, in this paper we used a larger block size for representation of a grid.

Figure 7 illustrates a new frame inserted into the sprite, as in Fig. 3a, except for the rest of the sprite which is not shown. The sprite's fixed grid is shown by dashed lines. The grid divides the frame (depending on its exact position within the sprite) into equal square blocks. Only the full blocks (as divided by the grid) will be processed in this technique. One such block is outlined by solid line (see Fig. 7).

Secondly, a set of square blocks is associated to each block position from the grid. Each set can hold up to a predefined maximum number of blocks (e.g., 5 blocks). The technique aims to maintain various block instances for each block position from sprite, as much varied as possible, in order to optimally cover the scene variations that may occur over time. The goal is to optimize the generation of side information, discussed in Section 2.5, i.e., to provide for each block position from sprite various block alternatives previously found in that position (e.g., a block representing part of a wall can still be used as a reliable source for generation of side information although it will be occluded later on, as a block alternative from the set of blocks associated to the respective block position from sprite).

Therefore, each full block (as divided by the sprite's grid) from the newly inserted frame is independently processed as follows. It is considered for adding to the set of blocks associated to the respective block position from sprite. Figure 8 illustrates an example of a set of block alternatives for a given block position.

Several considerations are initially assumed before adding a new block to a set. The Peak Signal-to-Noise Ratio (PSNR) metric is used to measure similarity between two blocks. For illustration purposes, a closer distance between two blocks in Fig. 8 represents a higher PSNR value between them (more similar blocks). There

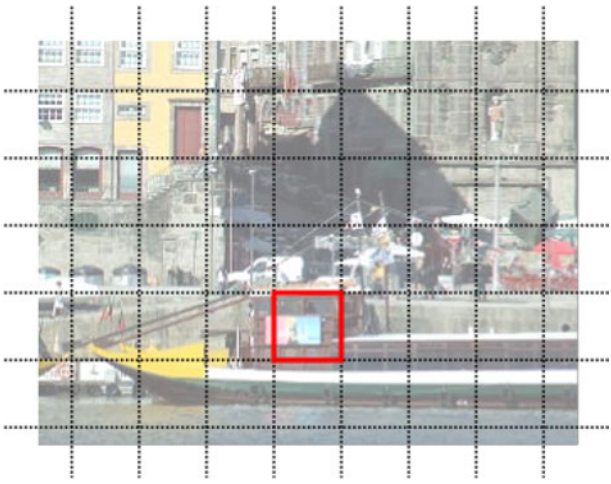


Fig. 7 The sprite associated grid (*dashed lines*). Example of one block position in grid (outlined by *solid line*)

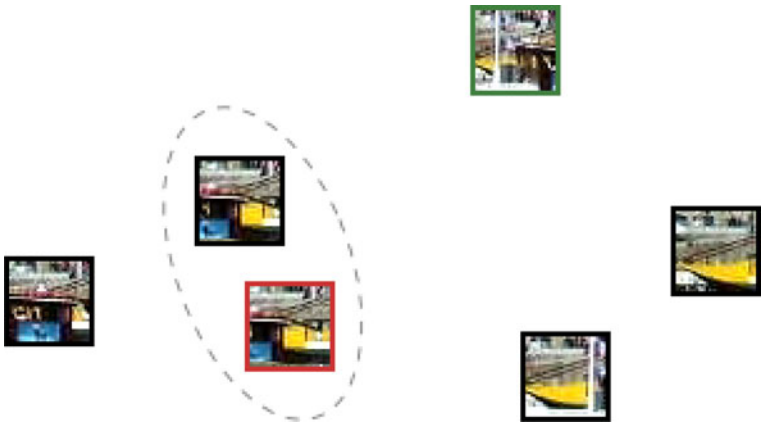


Fig. 8 Example of a set of alternative blocks for a given block position: the upmost block is added, the selection of two candidates for block removal is indicated by *dashed line*. The lower candidate will be finally removed

will be used a predefined maximum number of alternative blocks per set (e.g. 5). Additionally, each set holds an extra alternative block that is the last encountered block for that position. Also, every new block to be added will be discarded if it's too similar with the last successfully added alternative block (PSNR difference above a predefined threshold, e.g., 25 dB). Initially, before adding the first frame to sprite, all the sets are empty.

The following are the steps for maintenance of a set of alternative blocks when a new block is considered for adding:

1. if the set is not full then add the block; go to next block
2. if the set is full then one block needs to be eliminated; execute the following steps:
3. add the block to the set (it will hold now N+1 alternatives, N is the predefined maximum dimension of the set)
4. determine the maximum PSNR between each two blocks (a pair of blocks most similar one to another, see the selected two blocks from Fig. 8); one of them has to be eliminated
5. between the selected two blocks, eliminate the one that is generally most similar with rest of the blocks (having the highest sum of PSNR values with the rest of the blocks).

In the particular case of $N = 1$, the new block is added to the set and the two candidates selected for block removal are the only ones left in the set, i.e., there are no other blocks for PSNR comparison. In this case the previously stored block is automatically removed and the new one remains in the set. Consequently, the last encountered block for this block position will always be stored into the set, unless it's too similar with the last successfully added block.

The higher is the predefined maximum number of alternative blocks (N), the richer is the temporal data associated with the sprite and therefore, the better is the generated side information based on sprite. For scenes with high amount of movement, N can be set higher to better exploit the expected scene variations.

Section 4 presents an evaluation for two different values of N. Figure 20 illustrates an example of associated block alternatives to a sprite.

Concluding, the technique is applied when a new frame is inserted into the sprite, as a secondary step after the actual insertion discussed in Section 2.1, within the *Sprite Management* module (see Fig. 2). It only processes the new frame (and not the rest of the sprite), as it was positioned in sprite, i.e., are updated the sets of alternative blocks corresponding to the block positions overlapped by the new frame.

2.5 Sprite-based generation of side information

Section 2.3 discussed the sprite-based multi-view registration process, i.e., a window is continuously tracked in each sprite according to a given target view. This subsection describes the actual generation of side information based on the current window from sprite and the associated block alternatives of the sprite.

In this paper the resource for generating the side information is not a frame (image) as in [7] but a large collection of blocks, i.e., a set of alternative blocks per each block position from grid. Consequently, in order to solve this challenge of exploiting the block alternatives, we propose a three stage technique for generating the side information, as illustrated in the example from Fig. 9. First, we create a smaller distorted version of the previous decoded frame to match the exact size and shape of the current window (called *perspective frame*, as it represents the perspective of the target view into the sprite), as shown in Fig. 9. It is virtually fitted into the sprite in the exact area indicated by the window. Then, an estimation of the *perspective frame* is created by selecting the best matching block alternative for each block position (of the grid) overlapped (either partially or totally) by the *perspective frame*, resulting in a new image, seen as virtually overlapped over the same area from sprite. Finally, the side information is generated by bicubic interpolation applied on this image estimation, as indicated by the window.

Figure 9 also illustrates two examples of sets of block alternatives (each containing 20 block alternatives), corresponding to two of the block positions overlapped by the *perspective frame*. For illustration purposes the shape of the *perspective frame* is presented as a rectangle (not "deformed"). In most cases however, the inner area representing the actual contents appears slightly deformed, as a quadrilateral, although close to a rectangle.

The detailed steps of the technique are the following:

1. retrieve the current estimated window for this sprite (as the result of the multi-view registration)
2. select the minimal rectangular area from sprite, composed by full blocks according to the sprite's grid, which contains the window
3. transform the previous decoded frame by bicubic interpolation (as indicated by the window) into a smaller deformed image, called *"perspective frame"* (as illustrated in Fig. 9); the size of the generated *perspective frame* is that of the selected area from Step 2, the inner deformed area corresponding to the actual contents of the previous decoded frame is illustrated as a quadrilateral image over a black background, it has the exact size, shape and position of the window; the perspective frame can be virtually divided into full blocks as indicated by the sprite's grid (see Fig. 9)

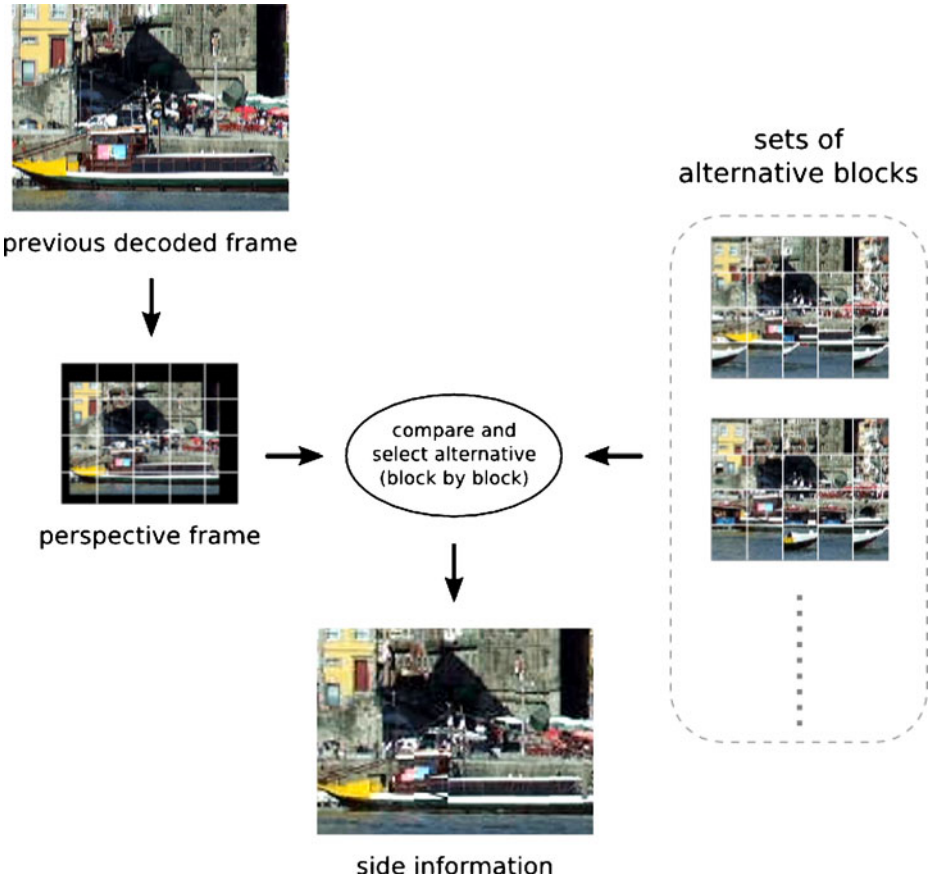


Fig. 9 Sprite-based generation of side information: the previous decoded frame, the generated *perspective frame*, two examples of sets of alternative blocks, the generated side information

4. compose a new image, block by block, with the same size as the perspective frame: for each block from the perspective frame, compare it with all the blocks from the set of alternative blocks for that grid position, select the most similar alternative block (maximum PSNR value); insert the selected alternative block in that block position in the new image; finally, a new image is created, resembling the perspective frame (with the same size), but without black borders
5. transform the new image by bicubic interpolation, as indicated by the window, into another new image with the same size as the target frame (an estimation of the target frame); we consider this final image the side information based on a sprite (see Fig. 9).

The edge blocks from the *perspective frame* are only partially compared, as indicated by the window, i.e., the PSNR comparison is performed only for the partial contents and not for the black areas.

The two image transformations by bicubic interpolation are mandatory in order to exploit the associated temporal data of the sprite (the alternative blocks), as the

sprite was generated from a different perspective (viewpoint of the scene) than the target view. The deformation degree of the *perspective frame* is a direct consequence of the angle between the target view (TV) and the view to which the sprite is associated (e.g., the reference view (RV) in Fig. 1), therefore the inner deformed area from the *perspective frame* is usually an arbitrary-shaped quadrilateral reflecting this angle.

The steps from 1 to 4 are performed by the *Sprite Perspective Preprocessing* submodule (see Fig. 2). They are considered sprite-based preprocessing steps prior to the actual generation of side information, executed in Step 5 by the module *Generation of Side Information*. In this paper we use the *perspective frame* to substitute the reference frame, as source for the generation of side information described in [7].

Concluding, the multi-view registration estimates the window by using the sprite itself, then the generation of side information is achieved based exclusively on this window and the associated block alternatives of the sprite (not including the sprite itself).

3 Side information refinement

In [7] we introduced a successive refinement of side information performed in various iterations per each decoded frame. The previous architecture was improved in this paper by adding an intermediate processing step on each iteration, performed by the *SI Refinement* module illustrated in Fig. 10. This technique aims to additionally refine the side information within the current iteration by exploiting the decoded

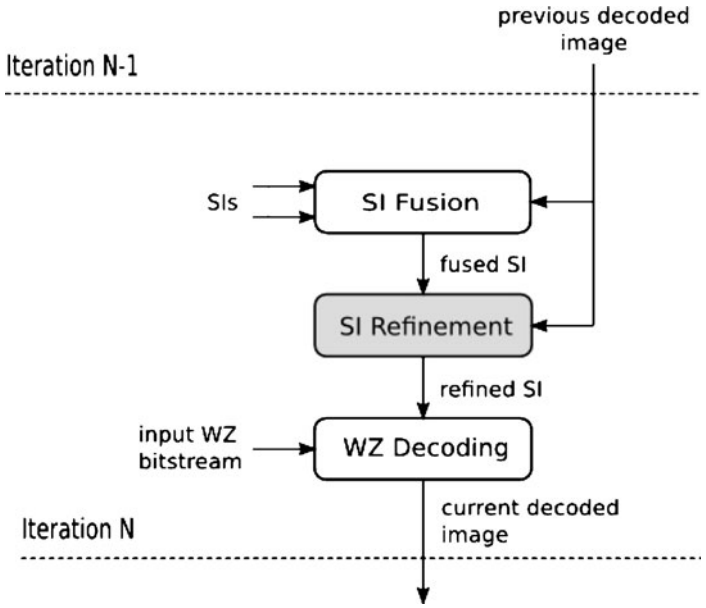


Fig. 10 The new *SI Refinement* module

image from the previous iteration (not to be confused with the successive refinement technique from [7]). The implementation is also included in the extended architecture presented in this paper (see Fig. 2).

First, the *SI Fusion* module provides the side information (SI) by fusion of several SIs obtained from different sources, i.e., all the reference cameras (and also all the target cameras in this paper). Then, the resulted side information is refined by this technique as follows: each block from the previous decoded image (e.g., 8×8 block) is used to perform a block matching in the side information, around the same block position and limited to a predefined search area (e.g., 125×50 pixels). Finally, the block (from side information) with the minimum Mean Squared Error (MSE) is inserted into the newly generated side information (identified as *refined SI* in Fig. 10), in the same block position. Thus, the technique produces a rearrangement of the blocks in the initial side information and furthermore, allows repetitions of the same block, as long as the entire image produced approximates better the previous decoded frame.

This technique is designed as a secondary post-processing step, after the SI fusion, for the optimization of side information before the decoding process (see Fig. 10).

(a) reference frame (b) target frame

(c) generated side information

Fig. 11 A pair of corresponding frames from the "Ribeira" sequence in **a** and **b**. The generated side information in **c**

Practical results show an overall significant enhancement of the side information when compared with the original target frames (see Section 4).

4 Results

We present the evaluation results for a dataset composed by seven (pairs of) video sequences, as follows: "Ribeira" (320 × 240, 150 frames, 15 fps), "Ribeira 2" (320 × 240, 100 frames, 15 fps), "Ribeira 3" (320 × 240, 100 frames, 15 fps), "Serralves" (320 × 240, 100 frames, 10 fps), "Castelo de Queijo" (320 × 240, 75 frames, 15 fps), "Castelo de Queijo 2" (320 × 240, 100 frames, 15 fps), "Parque da Cidade" (320 × 240, 100 frames, 15 fps). Each pair contains the two corresponding sequences captured from the target and reference camera. We aimed to demonstrate the robustness of the proposed architecture for scenarios with temporary non-overlapped views by providing diverse video sequences with progressive movement of the cameras, both unidirectional and constantly changing. Due to the low-processing constraints of the cameras in Distributed Video Coding, as presented in Section 1, we evaluated the proposed techniques using small resolution sequences (320 × 240 frame size).

Figure 11 shows a pair of corresponding frames from the "Ribeira" sequence. Figure 11c illustrates the generated (unrefined) side information. The reference sequence (see Fig. 11a) contains a partial occlusion when compared to the target sequence (see Fig. 11b), i.e., another passing boat in foreground. The occlusion is present for 90 frames and aims to test the sprite's associated block alternatives and the side information refinement technique, as discussed below.

(a) generated sprite for "Castelo de Queijo 2"

(b) generated sprite for "Ribeira 2"

Fig. 12 Examples of generated sprites for reference cameras: "Castelo de Queijo 2" in **a** and "Ribeira 2" in **b**

Due to specific artefacts produced by the presented techniques (e.g., distortion, blocking effect), we also used the perceptual measure SSIM in our evaluations.

Figures 12 and 13 illustrate a few examples of generated sprites for reference cameras.

Figures 14 and 16 illustrate a few examples of generated sprites with varied sub-pixel precision. The better sprite quality is obtained for 1/20 pixel precision (see Figs. 14d and 16d). For lower precisions however, the sprite generation (discussed in Section 2.1) may accumulate small mismatch errors at sub-pixel level for each added frame, i.e., a new frame that is inserted into the sprite may seem slightly displaced

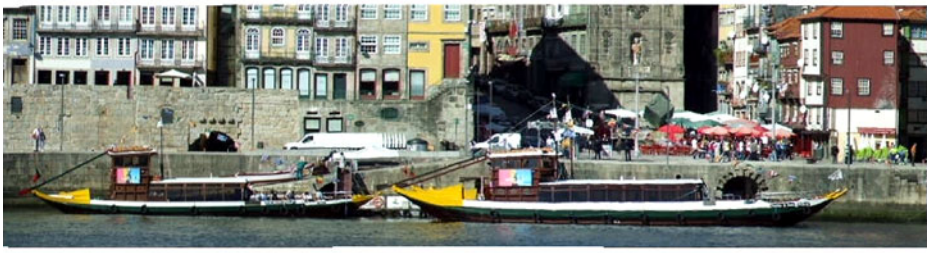
(a) generated sprite for "Parque da Cidade"

(b) generated sprite for "Serralves"

Fig. 13 Examples of generated sprites for reference cameras: "Parque da Cidade" in **a** and "Serralves" in **b**

relative to the other pixels already present in the sprite. These displacements occur on the borders of the currently added frame and may accumulate over time all over the sprite, from one added frame to another. Consequently, the overall sprite may be

(a) Pixel precision



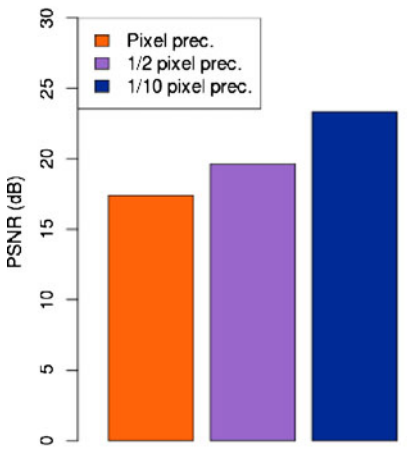
(b) 1/2 pixel precision

(c) 1/10 pixel precision

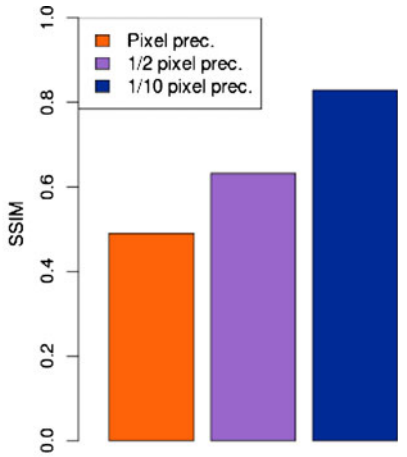
(d) 1/20 pixel precision

Fig. 14 Examples of generated sprites with varied sub-pixel precision for "Ribeira" sequence: pixel precision in **a**, 1/2 pixel precision in **b**, 1/10 pixel precision in **c** and 1/20 pixel precision in **d**

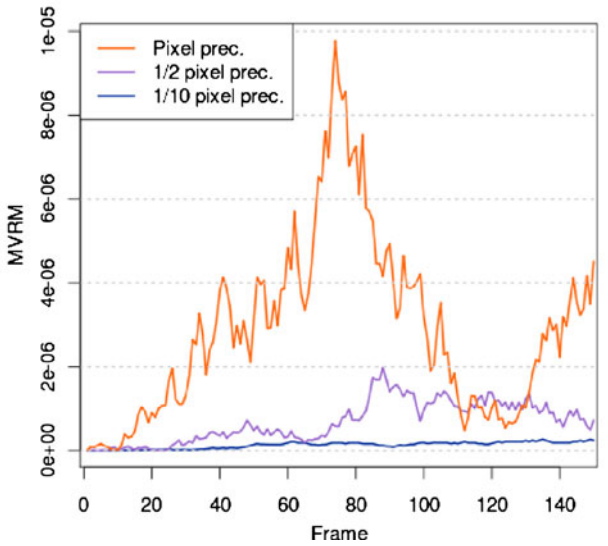
uniformly distorted, although not immediately noticeable at a quick visual inspection (e.g., although two sprites generated with different precisions seem identical and without evident deformations, one object may still be displaced in the two sprites, for instance with up to 10 pixels for a sprite with 100 added frames). Practical experiments have shown that a more detailed evaluation is necessary. For illustration purposes we used the original target sequences for this particular evaluation, i.e., we aimed to show sprites with many details and high quality (see Figs. 14 and 16) for easily noticing the deformations.



(a) PSNR evaluation



(b) SSIM evaluation



(c) MVRM evaluation

Fig. 15 Evaluation of generated sprites with sub-pixel precision for "Ribeira" sequence, relative to the 1/20 pixel precision: PSNR in **a**, SSIM in **b** and MVRM in **c**

Figure 14a shows some evident distortions, a slight deformation of the horizontal edges from the right side of the sprite (e.g., along the boat from the right side, the rooftops, etc.). Figure 16a also contains a slight deformation of the vertical edges of the houses, in the center-left area of the sprite.

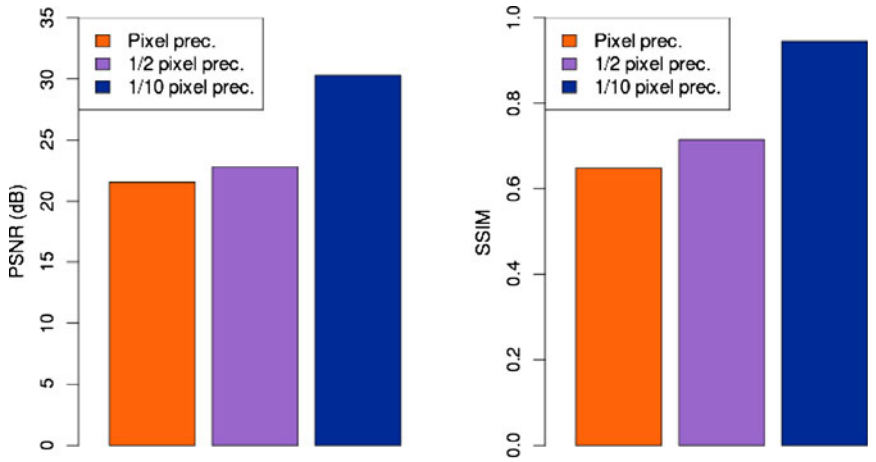
Practical experiments have shown that beyond a certain precision (e.g., 1/10 pixel precision) the generated sprite is accurate enough. We used the sprite with 1/20 pixel precision (see Figs. 14d and 16d) to evaluate the other precisions, by comparing the generated sprite images. Figures 15a, b and 17a, b show the gradual improvement of the sprite when raising the precision, for both PSNR and SSIM metrics.

Considering each added frame to sprite as an actual window within a "frame", there can be used the metric from [7] to compare two sprites, i.e., given a ground-truth sprite (the sprite with 1/20 pixel precision in this case) we can evaluate the accuracy of sprite generation for another (the precision of positioning each frame into the sprite). Figures 15c and 17c show the significant improvement of sprites for raising the precision. For 1/10 pixel precision the metric has all values close to 0 (zero)

(a) Pixel precision (b) 1/2 pixel precision

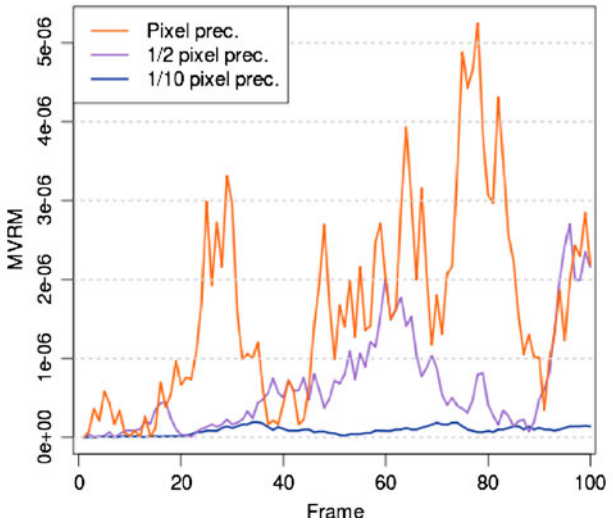
(c) 1/10 pixel precision (d) 1/20 pixel precision

Fig. 16 Examples of generated sprites with varied sub-pixel precision for "Ribeira 3" sequence: pixel precision in **a**, 1/2 pixel precision in **b**, 1/10 pixel precision in **c** and 1/20 pixel precision in **d**



(a) PSNR evaluation

(b) SSIM evaluation



(c) MVRM evaluation

Fig. 17 Evaluation of generated sprites with sub-pixel precision for "Ribeira 3" sequence, relative to the 1/20 pixel precision: PSNR in **a**, SSIM in **b** and MVRM in **c**

and consequently, no significant improvements can be achieved for higher precisions. The sprites from Figs. 14c, d and 16c, d, respectively, also seem identical. We considered a 1/10 pixel precision for all the other evaluations from this section. In this paper the metric is referred as Multi-View Registration Metric (MVRM)(see Figs. 15c and 17c).

As discussed above, for lower precisions the generated sprite image may constantly accumulate displacements, from one added frame to another. The accuracy of sprite generation over time is essential due to the large collection of pixels with different temporal tags gathered in this process. For instance, it prevents the eventual

(a) 0% Intra-blocks per frame

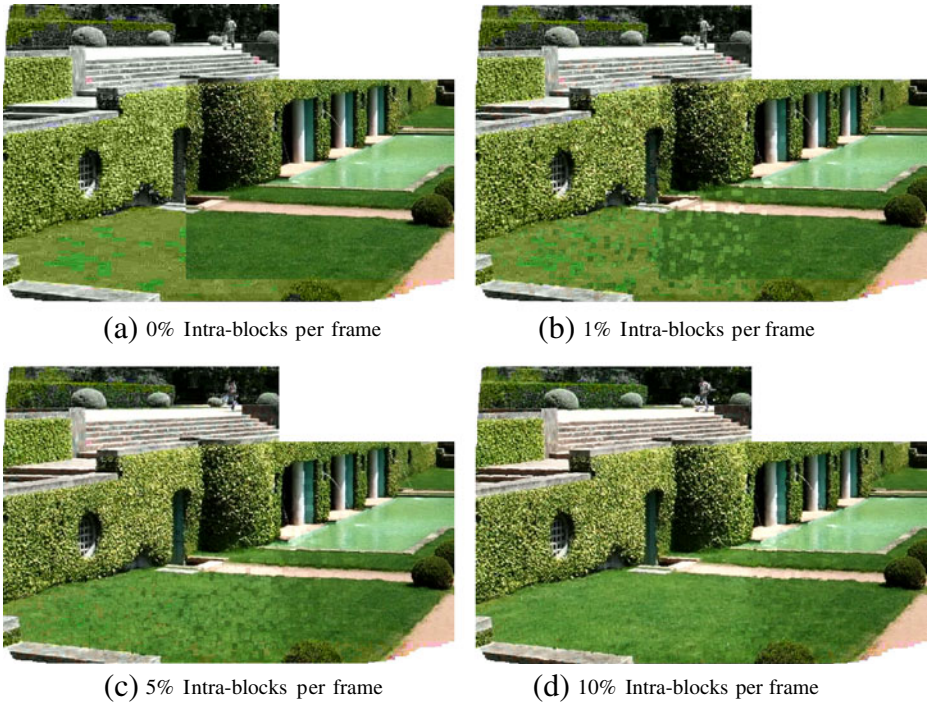
(b) 1% Intra-blocks per frame

(c) 5% Intra-blocks per frame

(d) 10% Intra-blocks per frame

Fig. 18 Examples of generated sprites for target camera (the "Ribeira" sequence): with 0% Intra-blocks per frame in **a**, 1% Intra-blocks per frame in **b**, 5% Intra-blocks per frame in **c** and 10% Intra-blocks per frame in **d**

mismatches between Intra and WZ blocks (see Figs. 18 and 19) as they use to have different temporal tags, i.e., an Intra-block is inserted into the sprite and will be persistent for a long time then, after many added frames, the neighbour WZ



(a) 0% Intra-blocks per frame (b) 1% Intra-blocks per frame

(c) 5% Intra-blocks per frame (d) 10% Intra-blocks per frame

Fig. 19 Examples of generated sprites for target camera (the "Serralves" sequence): with 0% Intra-blocks per frame in **a**, 1% Intra-blocks per frame in **b**, 5% Intra-blocks per frame in **c** and 10% Intra-blocks per frame in **d**

blocks already reflect the accumulated displacements. Consequently, a mismatch may appear between the Intra-block and the neighbour WZ blocks. It becomes more evident when lowering the precision of sprite generation. The sprite examples from Figs. 18 and 19 were generated with 1/10 pixel precision. A slight mismatch between the Intra-blocks and WZ blocks is noticeable on the edges (e.g., see the white stripe of the boat from the right, in Fig. 18).

As stated in Section 2.2, each reference camera performs conventional encoding (e.g., H.26x, MPEGx) of the captured frames, providing high quality frames at decoder and consequently, the high quality of the generated sprites.

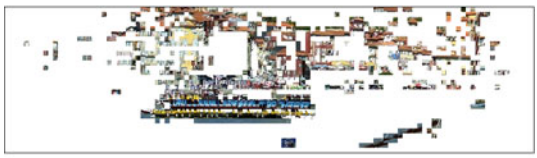
In Section 2.2 we proposed a technique based on the injection of a small amount of random selected Intra-blocks, in order to enhance the poor quality of sprites for target cameras. Figures 18 and 19 show a few examples of generated sprites for various percentages of Intra-blocks. Independently of the chosen percentage, the first target frame is always coded Intra and it appears as a large compact area with high quality in all the illustrated sprites. In Figs. 18a and 19a (for 0% Intra-blocks), the quality difference between the Intra-blocks (the high quality rectangular-shaped area from the right side) and the WZ blocks (the remaining areas) can be easily noticed. By raising the percentage of Intra-blocks up to 10%, the overall quality of the sprite is gradually improved. Both the random selection of Intra-blocks and the requirement to avoid repetition in consecutive frames, contribute to a uniformization of the sprite quality (see Figs. 18d and 19d).

(a) Sprite image

(b) Extra alternative blocks

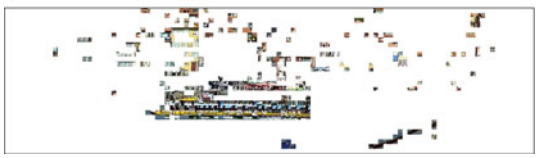
(c) The 1^{st} alternative blocks

(d) The 2^{nd} alternative blocks



(e) The 3^{rd} alternative blocks

(f) The 4^{th} alternative blocks



(g) The 5^{th} alternative blocks

Fig. 20 Generated sprite image (**a**) and its associated block alternatives for each grid position (**b**–**g**), for "Ribeira" sequence (reference camera)

In Fig. 19 the large compact area corresponding to the first target frame appears slightly shaded when compared with the rest of the sprite. This is caused by the self adjusting brightness of the target camera, along the captured sequence. As opposed to the gradual change in the sprite's brightness, from one added frame to another, illustrated in Fig. 13b, in this case there is a high discrepancy between the temporal tags of the area pixels and the neighbour pixels around this area, i.e., the first frame is inserted into the sprite and is persistent for a long time, then, after many frame overwrites the neighbour pixels around this area belong to a much distant frame, captured considerably later by the camera and already with significantly changed brightness. As noticed in Fig. 19d the same uniformization of the sprite quality also displayed a gradually changed brightness in the target sprite.

Figure 20 illustrates an example of generated sprite image and its associated block alternatives for "Ribeira" sequence. Up to five block alternatives are used per grid position (8×8 block size).

As presented in Section 2.5, the alternative blocks are used to generate the side information. In the evaluation from Fig. 21 we considered two cases, for maximum one and five, respectively, alternative blocks per grid position. We aimed to demonstrate the impact of raising the number of alternative blocks on the quality of generated side information, when compared with the original target sequence. The better performance is achieved in the latter case (five alternative blocks). A significant improvement can be noticed for the first 90 frames where the occlusion (foreground boat) is present, i.e., the side information gained up to 2.08 dB by raising the number of alternative blocks from one to five.

In the particular case of one alternative block per grid position, no PSNR comparison is needed for block removal (see Section 2.4). The only two candidates are the block already stored and the one to be currently added. Consequently, the old one is removed and the new one remains, i.e., each alternative block is always the last block encountered in that block position. Therefore, in this case the generated side information for the first 90 frames carries the occlusion.

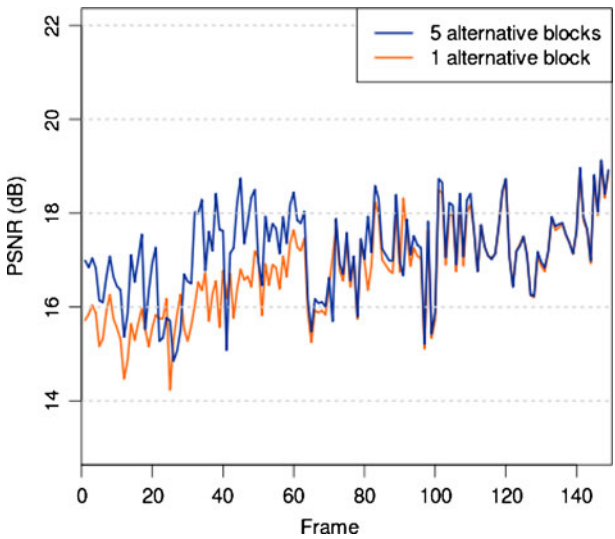


Fig. 21 Evaluation of generated side information based on the number of alternative blocks per grid position, for "Ribeira" sequence

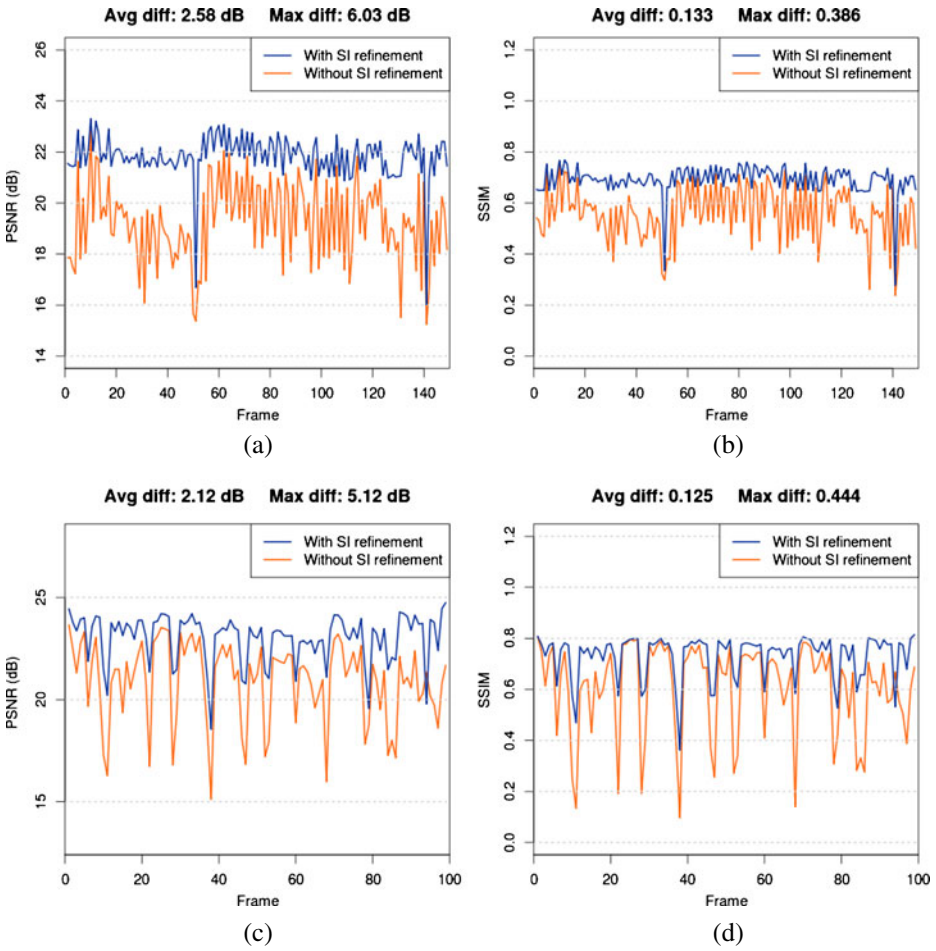


Fig. 22 Evaluation of side information refinement: for "Ribeira" sequence in **a**, **b** and "Ribeira 2" sequence in **c**, **d**

Figures 22 and 23 present the evaluation results for side information refinement. The technique provides a significant overall improvement of side information by over 2 dB on average.

Figure 24 illustrates an example of side information, before and after applying the refinement technique. Note the blocking effect produced by the technique (see Fig. 24b). It is caused by the rearrangement of the blocks in the side information in order to approximate better the last decoded image, as discussed in Section 3. Nevertheless, the perceptual quality (SSIM) is also improved (see Figs. 22 and 23).

In the example illustrated in Fig. 24 the side information was generated for decoding the second target frame of "Ribeira" sequence. Consequently, the initial (unrefined) side information carries the occlusion (foreground boat) in the lower side (see Fig. 24a). In this particular case, the refinement technique also conceals the occlusion by block padding the lower side, according with the last decoded image, i.e., the first target frame that was sent as conventional Intra.

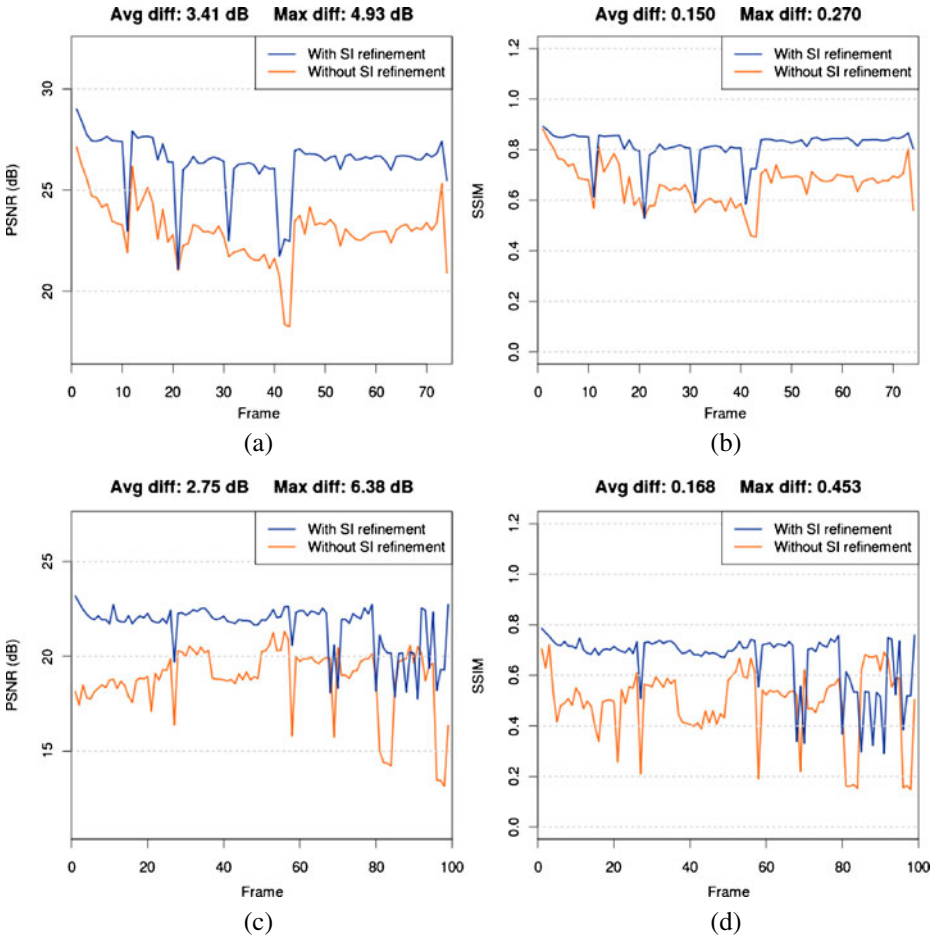


Fig. 23 Evaluation of side information refinement: for "Castelo de Queijo" sequence in **a**, **b** and "Parque da Cidade" sequence in **c**, **d**

(a) without SI refinement (b) with SI refinement

Fig. 24 Example of refined side information for "Ribeira" sequence: the initial side information in **a** and the refined side information in **b**

5 Conclusions

In this paper we described a technique for generation of side information in multi-view systems with temporary non-overlapping among views. In this difficult scenario we are challenged by the possible lack of real-time resources for generating the side information at some time instants. In order to overcome this issue we exploit the past visual data associated with each view, previously generated as a panoramic image (sprite). Over time, various perspectives of the scene may be built in an efficient manner, holding both the static features of the scene and the eventual relevant variations as well (e.g., object movement, occlusions). This procedure enables the generation of side information later on, as required for decoding the Wyner-Ziv bitstreams from target cameras. In the complex multi-camera environment described in Section 1, exploiting as many sprites as possible from different viewpoints and with different qualities, not only provides the means (later on) for generating the side information for decoding a given target view, but also contributes to the optimization of the respective (fused) side information.

The efficient storing and management of the past visual data has a direct impact on the quality of the generated side information and therefore, besides providing a solution for these scenarios, we also discussed a few optimization techniques (e.g., diversification of the stored data, gradual uniformization of the high quality data, eliminating distortion effects, etc.). Practical experiments show significant improvements (both objectively and subjectively) of the collected visual data. Nevertheless, this complex topic presented in this paper needs further study towards optimization and flexibility of the system (e.g., dynamic allocation of the number of alternative blocks depending on the scene variations, gradually discarding very old contents from both the sprite and the associated block alternatives, etc.). Additional evaluation methods will be investigated.

We also presented a scenario-independent technique for additional refinement of the side information, designed as a post-processing step before the Wyner-Ziv decoding. Practical results show an overall significant enhancement of side information by over 2 dB.

Acknowledgement The first author acknowledges the *Fundação para a Ciência e a Tecnologia, Portugal*, for the financial support.

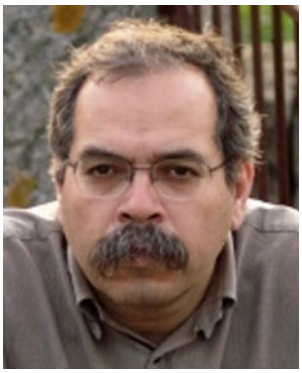
References

1. Aaron A, Girod B (2004) Wyner-ziv video coding with low encoder complexity. In: Proc. international Picture Coding Symposium, PCS'04, San Francisco, CA
2. Aaron A, Rane S, Girod B (2004) Wyner-ziv video coding with hash-based motion-compensation at the receiver. In: Proc. IEEE intl. conference on image processing, Singapore
3. Artigas X, Angeli E, Torres L (2006) Side information generation for multiview distributed video coding using a fusion approach. In: 7^{th} Nordic Signal Processing Symposium, NORSIG'06, Reykjavik, Iceland, 7–9 June 2006
4. Ascenso J, Pereira F (2007) Adaptive hash-based side information exploitation for efficient Wyner-Ziv video coding. In: International Conference on Image Processing (ICIP)—2007, San Antonio, Texas, USA
5. Benedek C, Havasi L, Sziranyi T, Szlavik Z (2005) Motion-based flexible camera registration. In: IEEE conference on Advanced Video and Signal Based Surveillance, 2005, AVSS 2005, pp 439–444

6. Bergevin R, Soucy M, Gagnon H, Laurendeau D (1996) Towards a general multi-view registration technique. IEEE Trans Pattern Anal Mach Intell 18(5):540–547
7. Ciobanu L, Corte-Real L (2010) Successive refinement of side information for multi-view distributed video coding. Multimed Tools Appl 48(3):411–436
8. Dufaux F, Moscheni F (1996) Background mosaicking for low bit rate video coding. In: Proceedings of international conference on image processing, vol 1, pp 673–676
9. Dufaux F, Ouaret M, Ebrahimi T (2007) Recent advances in multiview distributed video coding. In: SPIE Defense and Security Symposium (DSS 2007), Orlando, Florida, USA, 9–13 Apr 2007
10. Guo X, Lu Y, Wu F, Gao W, Li S (2006) Distributed multi-view video coding. In: Proceedings of SPIE-IS&T Electronic Imaging, SPIE, vol 6077. San Jose, California, USA, 15–19 Jan 2006
11. Izquierdo E (2003) Efficient and accurate image based camera registration. IEEE Trans Multimedia 5(3):293–302
12. Lee M-C, Chen W-G, Lin C, Gu C, Markoc T, Zabinsky S, Szeliski R (1997) A layered video object coding system using sprite and affine motion model. IEEE Trans Circuits Syst Video Technol 7(1):130–145
13. Ouaret M, Dufaux F, Ebrahimi T (2006) Fusion-based multiview distributed video coding. In: ACM International Workshop on Video Surveillance and Sensor Networks, Santa Barbara, CA, USA, 27 Oct 2006
14. Ouaret M, Dufaux F, Ebrahimi T (2007) Multiview distributed video coding with encoder driven fusion. In: European Conference on Signal Processing (EUSIPCO), Poznan, Poland, 3–7 Sept 2007
15. Pereira FC, Ebrahimi T (2002) The MPEG-4 Book. Prentice Hall PTR, Upper Saddle River, NJ, USA
16. Puri R, Ramchandran K (2003) PRISM: a "reversed" multimedia coding paradigm. In: Proc. Intl. Conference on Image Processing (ICIP), Barcelona, Spain
17. Sikora T (2005) Trends and perspectives in image and video coding. In: Proceedings of the IEEE vol 93(1), pp 6–17
18. Slepian D, Wolf JK (1973) Noiseless coding of correlated information sources. IEEE Trans Inf Theory 19:471–480
19. Smolic A, Sikora T, Ohm J-R (1999) Long-term global motion estimation and its application for sprite coding, content description, and segmentation. IEEE Trans Circuits Syst Video Technol 9(8):1227–1242
20. Szlávik Z, Szirányi T, Havasi L (2007) Video camera registration using accumulated co-motion maps. ISPRS J Photogramm Remote Sens 61(5):298–306
21. Wyner D, Ziv J (1976) The rate-distortion function for source coding with side information at the decoder. IEEE Trans Inf Theory 22:1–10
22. Yeo C, Ramchandran K (2007) Robust distributed multi-view video compression for wireless camera networks. In: VCIP 2007, San Jose, California, USA, 28 Jan–1 Feb 2007

Lucian Ciobanu was born in Iasi, Romania, in 1978. He graduated in Software Engineering at the Faculty of Automatic Control and Computer Engineering—the "Gheorghe Asachi" Technical University of Iasi, Romania. He is researcher at the Institute for Systems and Computer Engineering

(INESC) Porto, Portugal, since 2002 and also PhD student of the Faculdade de Engenharia da Universidade do Porto, Portugal, since 2005. His research interests include image/video coding and processing.



Luís Côrte-Real was born in Vila do Conde, Portugal, in 1958. He graduated in Electrical Engineering from the Faculdade de Engenharia, Universidade do Porto, Portugal, in 1981. He received the M.Sc. degree in Electrical and Computers Engineering in 1986 from Instituto Superior Técnico, Universidade Técnica de Lisboa, Lisbon, Portugal and the Ph.D degree from the Faculdade de Engenharia, Universidade do Porto, in 1994. In 1984 he joined Universidade do Porto as a lecturer of telecommunications. He is currently associate professor at the Departamento de Engenharia Electrotécnica e de Computadores da Faculdade de Engenharia da Universidade do Porto. He is researcher at the Institute for Systems and Computer Engineering (INESC) Porto, Portugal since 1985. His research interests include image/video coding and processing.