OPTIMAL PARTITIONING OF VIDEO SEQUENCES FOR MPEG-4 SPRITE ENCODING

Dirk Farin[1], Peter H. N. de With[2], Wolfgang Effelsberg[1]

[1] Dept. of Computer Science IV, University of Mannheim, Germany
[2] LogicaCMG / Univ. Technol. Eindhoven, 5600 MB Eindhoven, Netherlands

This paper shows that the concept of merging several views of a non-changing scene-background into a single background sprite is not necessarily the most efficient way to transmit the background image. We have found that the counter-intuitive approach of splitting the background into several independent parts can reduce the overall amount of data. For this reason, we propose an algorithm that provides an optimal partitioning of a video sequence into independent background sprites, resulting in the minimum coding cost. Moreover, our algorithm results in background sprites with better quality by ensuring that the sprite resolution has at least the final display resolution throughout the sequence.

INTRODUCTION

One video-encoding tool defined in the MPEG-4 standard is the coding of a scene background as a single image. If the camera is moving, this background image (*sprite*) is larger than the actual video format. The decoder can reconstruct the current background view from the sprite based on a small set of transmitted camera parameters. Hence, the sprite itself needs to be transmitted only once, which can result in a large improvement of coding efficiency [1]. A sprite is typically synthesized in the encoder by first applying a robust global-motion estimator to determine camera-motion. The current input frame is then warped into the coordinate system of a reference frame to get a seamless mosaic. To our knowledge, optimal selection of the reference coordinate system has not been discussed previously, although this has direct implications for the generated sprite size and resolution. Interestingly, transmitting the background in a single sprite is not generally the most efficient approach. This paper shows that transmitting the background as several separate sprites can result in an overall reduced amount of data. Besides this, we also present an algorithm that computes the optimal partitioning of a video sequence in terms of *minimum coding cost*. Additionally, our algorithm can incorporate a constraint ensuring that no input frame is warped to a size smaller than the input resolution. As a result, sprite coding will not cause any loss of resolution.

PROBLEM CASES

The MPEG-4 sprite decoding algorithm is based on the perspective motion model, which can compensate arbitrary camera motion if the scene background is planar or as long as the camera does not change its location in the 3-D world. The current background view $b(x', y')$ is reconstructed from the sprite image $s(x, y)$ using the transformation

$$x' = \frac{a_{00}x + a_{01}y + t_x}{p_x x + p_y y + 1}, \quad y' = \frac{a_{10}x + a_{11}y + t_y}{p_x x + p_y y + 1}. \tag{1}$$

This transformation is actually a plane-to-plane mapping (a homography [2]). Thus, the background sprite image can be envisioned as the projection of the 3-D world onto a plane. This induces one direct limitation of the MPEG-4 sprite model: only 180 degrees field of view can be represented in a single sprite image. In practice, the usable viewing angle is even smaller since the perspective distortion increases rapidly when the camera rotates clearly away from the frontal view position (see Figure 1). Consequently, the required sprite size increases quickly during a camera pan with short focal length. Even though some input images are projected to a larger area in the sprite than their original size, this does not result in an increased resolution at the decoder output, since the image will be scaled down to its original resolution again at the decoder. Hence, the sprite-coding is *inefficient* in the sense that it uses high resolution for transmitting the sprite although this extra resolution is never displayed.

A comparable problem occurs for sequences containing zoom-out operations. Since each image covers a larger view than the last one, the image size in the sprite is constantly increasing. At first, the sprite-mode is advantageous, because most of the image was already visible in the last image. However, when the zoom continues, a point occurs where the increase of total sprite size outweighs the reuse of existing background and it would be better to start with a new sprite.

In the sequel, we propose a more efficient coding technique based on splitting the video sequence into several partitions and calculating a separate background sprite for each partition. Although some parts of the background may be transmitted twice, the overall sprite area to be transmitted is reduced. This counter-intuitive property results from the fact that the geometric distortions from camera operations do not accumulate as much in this case, so that larger parts of the sprite can be transmitted in a resolution close to the lowest possible value.

A complementary problem case are zoom-in operations. If the camera performs a zoom-in after the reference frame, the pictures are mapped onto a smaller

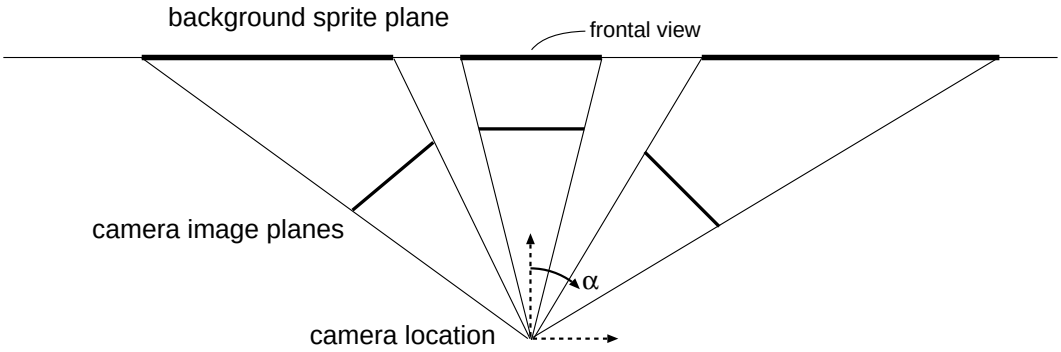


Figure 1: Top-view of projecting the input frames onto the background sprite plane. The more the camera rotates away from the frontal view ($|\alpha|$ increases), the larger the projection area on the sprite plane.

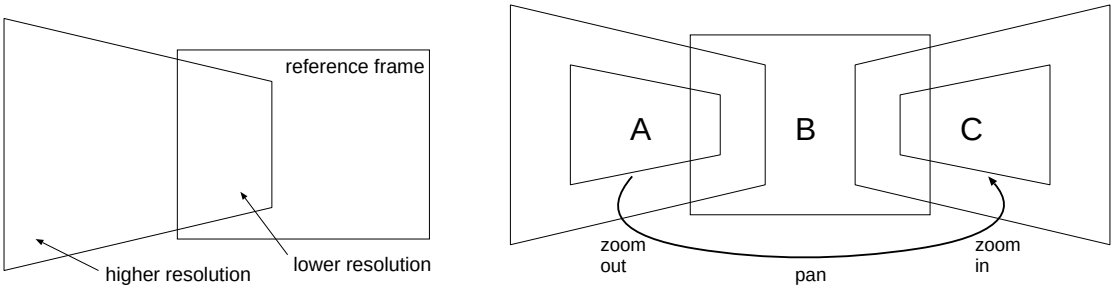
area of the sprite than the reference frame. Hence, the resolution in the sprite is lower than the input resolution. This results in a reduced decoder output quality. To solve this problem, we have to ensure that the sprite resolution is never lower than the corresponding input resolution. We have designed the algorithm such that it can integrate this additional constraint in its optimization process.

ALGORITHM

To find the best sequence partitioning, the algorithm has to determine the optimal segment boundaries and, additionally, for each segment the optimal reference frame. The reference frame has to be chosen such that the required sprite area for this segment is minimal under the extra constraint that the sprite resolution is never below the input resolution. However, there are cases where both criteria cannot be satisfied at the same time. Therefore, we introduce a segment scaling-factor for increasing the total sprite resolution. As an example, let us consider the case depicted in Fig. 2b. For optimality, frame B should be selected as reference and the scaling factor has to be chosen such that A and C still have full input resolution.

Definition of coding cost

Consider a sprite, synthesized from frames i to k and using the reference frame $r \in [i;k]$. We approximate the sprite area by computing the bounding box $S^r_{i;k}$ around the image-corner pixel positions in the sprite coordinate system (Figure 3). This approximation is valid since an optimal selection of the reference frame results in a sprite that almost completely fills a rectangular area. In cases where no background sprite can be constructed (angle of view > 180 degrees), $S^r_{i,k}$ is set to the special value ∞, denoting a bounding box of infinite size. We further



(a) resolution change during pan

(b) zoom out, pan, zoom in

Figure 2: (a) Camera rotation leads to warping parameters that transform the input with non-constant scaling. (b) Example sequence for which it is not possible to choose a reference frame while meeting the minimum resolution constraint. An additional scaling factor has to be introduced.

select a specific bounding box $S^*_{i;k}$ according to $||S^*_{i;k}|| = \min_r ||S^r_{i;k}||$ where $||\cdot||$ denotes the bounding-box area. The parameter r, that minimizes the expression, indicates the optimal reference frame for the sequence range $[i;k]$ and $||S^*_{i;k}||$ is its approximate sprite size. As an approximation, we assume that the coding cost is proportional to the sprite area.

Since a direct computation of all $S^*_{i;k}$ would be too complex, we split the calculation of $S^r_{i;k}$ into two steps by combining $S^r_{i;r}$ and $S^r_{r;k}$ (note that the reference is fixed at the end and start of the range, respectively) into a bounding box enclosing both. The bounding boxes $S^r_{i;r}$ (backwards from the reference frame) can be computed iteratively by enlarging $S^r_{i+1;r}$ using the four corner points of frame i. As initialization, $S^r_{r;r}$ is set to the input frame area. In the forward direction, $S^r_{r;i}$ can be computed similarly by extending $S^r_{r;i-1}$.

Optimal sequence partitioning

Let $P = \big((1, p_1 - 1), (p_1, p_2 - 1), (p_2, p_3 - 1), \ldots, (p_{n-1}, N)\big)$ be a partitioning of the video sequence of length N into n sub-sequences. We now minimize the total sprite size over all possible partitions by determining

$$P^* = \arg\min_P \sum_{(a,b)\in P} \frac{1}{m_{a;b}} ||S^*_{a;b}||. \tag{2}$$

The magnification factor $m_{a;b}$ will be introduced in the next section. For the time being, it can be assumed unity. The minimization problem can be computed

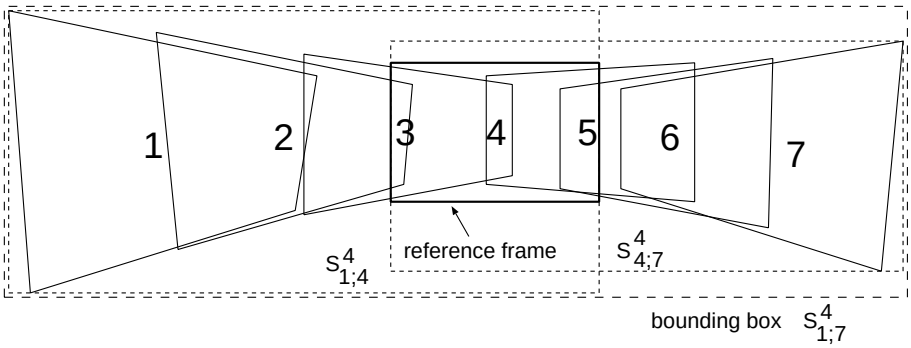


Figure 3: Sprite for frames 1 to 7 with frame 4 as the reference frame. In this case, the bounding-box total $S^4_{1;7}$ can be determined as combined bounding-box using $S^4_{1;4}$ and $S^4_{4;7}$.

efficiently with an iterative algorithm. For each image i, we compute the minimum cost c_i ($c_0 = 0$) of a partitioning ending in image i as

$$c_i = \min_{k \in [1,i]} \left\{ c_{k-1} + \frac{1}{m_{a;b}} ||S^*_{k;i}|| \right\}. \tag{3}$$

The index k denotes the beginning of the last segment in the partitioning up to frame i. For each image, we store the k for which the minimum was reached. Tracing back these stored k-values, starting at frame N, results in the optimal partitioning with respect to total sprite size.

Determining the scaling factor

The algorithm described so far does not enforce a constraint on the minimum sprite resolution. This results in undesirable behaviour if the camera zooms into the scene. Since the algorithm tries to minimize the total sprite area, it will select the frame at the beginning of the zoom as reference. However, this will lead to a poor quality for the decoded images at the end of the zoom. Hence, we have to constrain the solution such that the resolution never falls below the input frame resolution. This is achieved by calculating the magnification factor $m_{a;b}$, that equals the minimum factor by which a frame will be scaled when building the sprite for frame range a to b.

Since the motion model includes perspective distortion, the scaling factor is not constant over a single input frame (see Fig. 4). The local scaling factor can be computed using the Jacobian determinant of the transformation mapping the

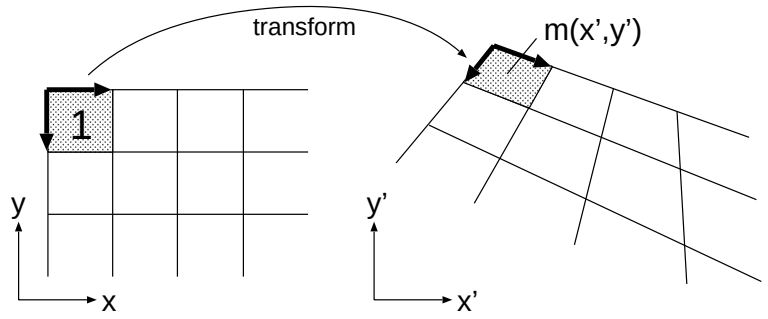


Figure 4: Change of local resolution. The input image (left) is warped to the sprite coordinate system (right). This transformation can change the size of a pixel.

input coordinate system to the reference coordinate system:

$$m(x',y') = \begin{vmatrix} \frac{\partial x'}{\partial x} & \frac{\partial x'}{\partial y} \\ \frac{\partial y'}{\partial x} & \frac{\partial y'}{\partial y} \end{vmatrix} = \frac{1}{D^2}\left[\begin{vmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{vmatrix} - \begin{vmatrix} a_{00} & a_{01} \\ p_x & p_y \end{vmatrix} y' + \begin{vmatrix} a_{10} & a_{11} \\ p_x & p_y \end{vmatrix} x' \right], \quad (4)$$

where $D = p_x x + p_y y + 1$ is the denominator of the motion model Equation (1) [1]. Since $m(x',y')$ is linear in x' and y', its minimum value over the image area can be found in one of the image corners. Hence, we compute $m(x',y')$ for all four image corners and take the minimum. Finally, we set $m_{a;b}$ to the minimum scale of all frames between a and b.

To regain full input resolution in the sprite, we have to increase the sprite resolution by a factor of $1/m_{a;b}$ in segment $(a;b)$. This increased coding cost is reflected in the optimal partitioning algorithm presented earlier by integrating the scaling factor into the cost definition.

EXPERIMENTS AND RESULTS

This section describes the algorithm behaviour for the two sequences *table-tennis* and *stefan*. From the *table-tennis* sequence, the first part consisting of a long zoom-out has been selected (first 132 frames). The algorithm prevents the sprite from getting too large by splitting the sequence into three parts (Fig. 5). It also selects the highest-resolution image as reference to get optimal reconstruction quality.

A single sprite-image, generated from the MPEG-4 sequence *stefan* without using the described algorithm is shown in Figure 6. It is not possible to synthesize

[1]For affine transformations, p_x, p_y are zero and $D = 1$. Therefore, the pixel scale is simply the determinant of the affine matrix, which is 1 for orthonormal transformations like rotations.

(a) frames 1–51 (b) frames 52–77 (c) frames 78–132

Figure 5: Multi-sprite synthesized from a long zoom-out operation. The sequence is partitioned into three separate sprites of almost the same size. The center image has been selected as the reference coordinate system (shown in a darker shade).

a single background sprite for the whole sequence, because the total viewing angle is too large. Instead, we used only the first 255 frames for building the sprite. Alternatively, the new algorithm splits the complete sequence of 300 frames into four partitions as shown in Figure 7. We have measured that the total required sprite size for the multiple-sprite approach is a factor of 2.7 smaller than for the single sprite case, which did not even cover the whole input sequence.

CONCLUSIONS

This paper showed that transmitting a background sprite in several independent parts can result in clearly reduced coding cost and better resolution at the same time. A proposed algorithm computes the optimal partitioning of a sequence, the reference frame for each partition, and associated scaling factors. The reduction of coding cost depends on the type of camera motion in the sequence. For the *stefan* sequence, a reduction by a factor of 2.7 has been achieved.

REFERENCES

[1] Hiroshi Watanabe, Kumi Jinzenji. "Sprite Coding in Object-based Video Coding Standard: MPEG-4". In *World Multiconf. on SCI 2001*, Vol. XIII, pp. 420–425.

[2] Richard Hartley, Andrew Zisserman. "Multiple View Geometry in Computer Vision". *Cambridge University Press*, 2000.

[3] Frédéric Dufaux and Fabrice Moscheni. "Background Mosaicking for Low Bit Rate Video Coding". In *IEEE International Conference on Image Processing 1996*, Vol. 1, pp. 673–676.

[4] Yan Lu, Wen Gao, Feng Wu. "Sprite generation for frame-based video coding". In *IEEE International Conference on Image Processing 2001*, Vol. 1, pp. 473–476.

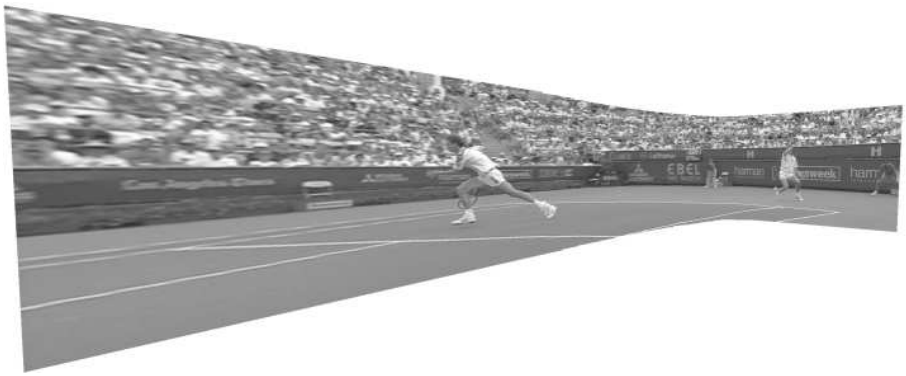
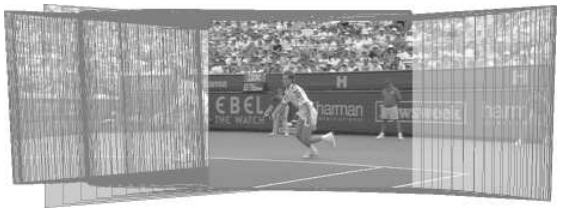
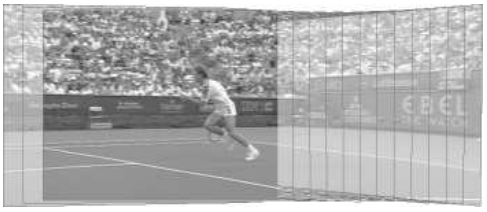


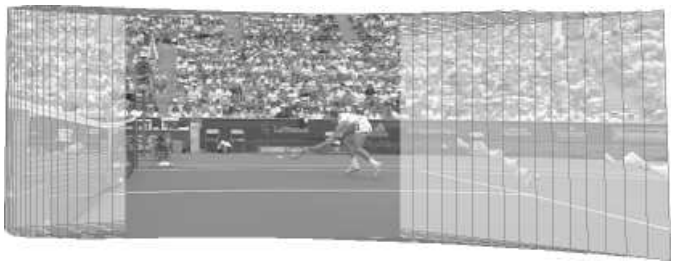
Figure 6: Sprite synthesized from *stefan* sequence. Only the first 255 frames can be used since it is impossible to create the sprite for the complete sequence if the first frame is selected as reference. Sprite resolution is 2445×1026 pixels.



(a) 1–241, 926×339



(b) 242–255, 699×296



(c) 256–292, 830×318

(d) 293–300, 431×350

Figure 7: Sprite partitions synthesized using the described algorithm. The respective reference frames are depicted in a darker shade. Note that the long camera pan is broken up into two separate sprites and that the zoom-in at the end of the sequence is put into a separate sprite. Total sprite area of all partitions is 2.7 times smaller than in the single sprite case (which covered not all input frames).