International Journal of Pattern Recognition
and Artificial Intelligence
Vol. 23, No. 2 (2009) 331–353

World Scientific
www.worldscientific.com

A FAST MULTISPRITE GENERATOR WITH NEAR-OPTIMUM CODING BIT-RATE

I-SHENG KUO* and LING-HWEI CHEN†

Department of Computer Science
National Chiao Tung University, 1001 Ta Hsueh Rd.
Hsinchu, Taiwan 30050, R. O. C.
**sasami@debut.cis.nctu.edu.tw*
†*lhchen@cc.nctu.edu.tw*

Sprite coding, which can increase the coding efficiency of backgrounds greatly, is a novel technology adopted in MPEG-4 object-based coding. Due to the geometric transformation applied to each nonreference frame in the procedure of sprite coding, the generated sprite is distorted and the available view angles relative to the reference frame are restricted. These problems can be resolved by using multiple sprites. An optimal multisprite generator has been proposed by Farin *et al.*, but the optimal method requires high computation in the sprite coding cost. To treat this disadvantage, a fast multisprite partition algorithm is proposed in this paper based on frame translations and scalings. The proposed algorithm divides a video sequence into several subsequences, and a sprite is generated from each subsequence. The experimental results show that the executing speed of the proposed method is increased by 10 to 190 times than the optimal method, with the total size of generated sprites is slightly higher and the qualities of generated sprites are preserved.

Keywords: Multiple sprites; reference frame; sequence splitting; sprite generation; background mosaic; MPEG-4.

1. Introduction

MPEG-4[4] had adopted a novel technique to code a series of backgrounds belonging to a scene into a single panoramic image, which is often denoted as a "sprite". The constructed sprite and some specified parameters are transmitted to the receiver, and then a decoder can reconstruct the series of backgrounds by the transmitted information. Since the sprite is transmitted only once, this technique can achieve very low bit-rate with good quality. Aside from the high coding efficiency of sprite coding, the generated sprite is also useful for segmenting moving objects.[3] The extracted moving objects and the sprite itself can be used in video summarization.[6]

A sprite is constructed in the encoder by a sequence of complex algorithms called a "sprite generator". Many sprite generators[7,8,10–12] have been proposed, most of

†Author for correspondence

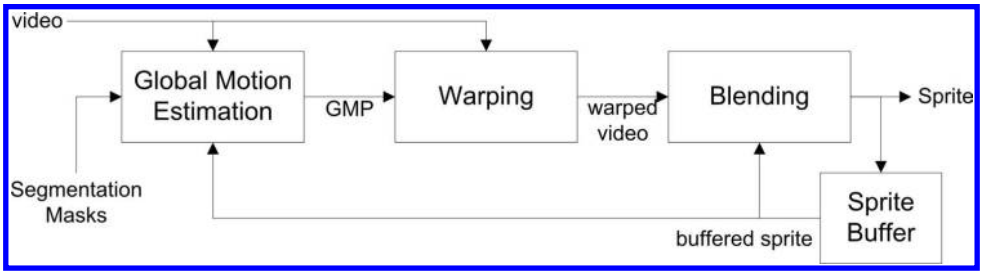


Fig. 1. The framework of the sprite generator in MPEG-4 VM.

them are based on a framework provided by MPEG-4 VM.[5] In the framework shown in Fig. 1, a sprite generator consists of three parts: a global motion estimator, a frame warper and a sprite blender. The global motion estimator estimates the camera motion of each input frame relative to the camera view of a selected reference frame, and the camera motion is represented by parameters of a certain geometric model. These parameters are known as global motion parameters (GMP) and will be sent to the receiver along with the generated sprite. Then each input frame is geometrically transformed (also called warped) to the reference coordinate system of the sprite that is also the coordinate system of the selected reference frame to eliminate the effect of the camera motion. Each warped frame is then blended into the sprite sequentially by the sprite blender.

The performance of a sprite generator is limited to the perspective motion model applied in the MPEG-4 VM's framework.[2] The perspective model projects each frame of a video sequence into a planar reference coordinate system, which is the coordinate system of the reference frame, as shown in Fig. 2. The focal length of the camera is f and the reference imaging coordinate system for a sprite is assumed to be the first frame that is denoted as frame A in Fig. 2. All the following frames must be projected to this reference system by geometric transformation. As the camera rotates, transformed frames are geometrically distorted, this phenomenon can be

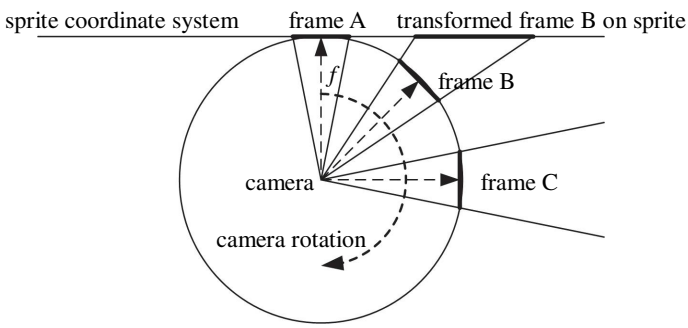
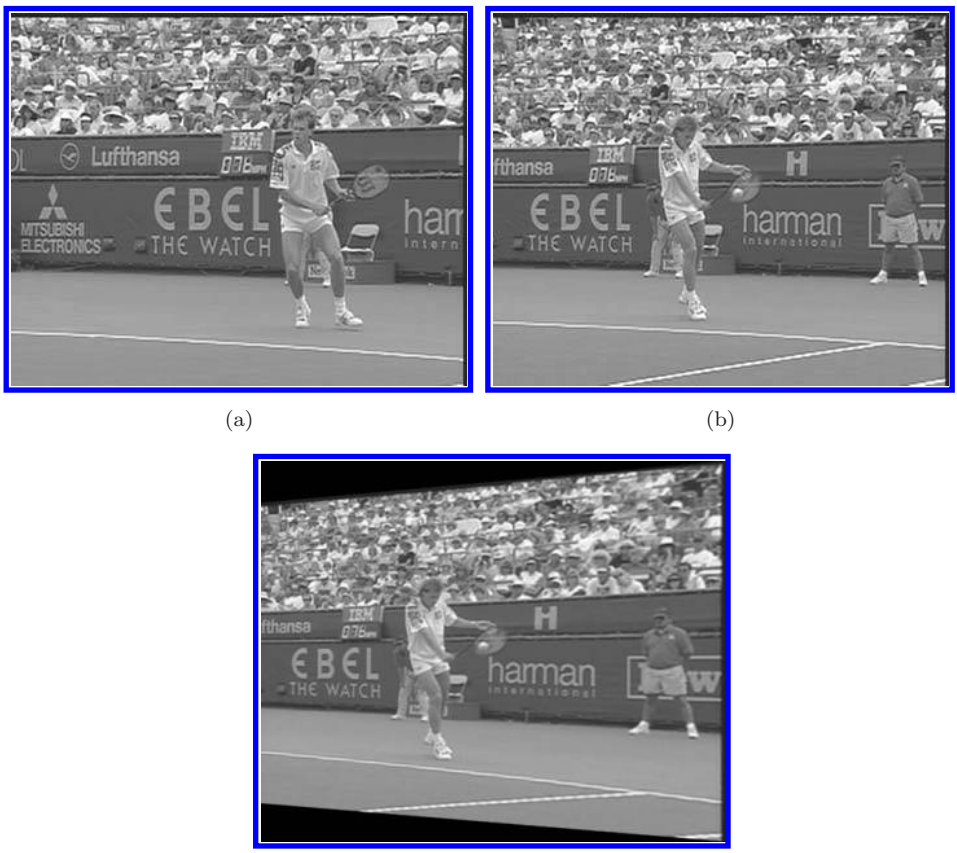


Fig. 2. A sprite coordinate system and the geometric distortions of transformed frames.

found between frame B and transformed frame B. If camera rotation continues, from Fig. 2, we can see that frame C cannot be projected to the reference system. Figure 3 shows a geometric distorted frame. Frame A shown in Fig. 3(a) is employed as the reference frame of the sprite coordinate system. Figure 3(b) shows a frame B, and the transformed frame B relative to frame A is shown in Fig. 3(c).

Theoretically, the perspective model employed in MPEG-4 VM can cover 180° of view. However, the useable viewing angle is much smaller in practice, since the geometric distortion increases rapidly as the camera rotates away from the reference frame. Frames away from the reference frame are forced to be recorded by extremely large resolution, but this resolution is useless because the sprite must be scaled down to display by the decoder.

In order to overcome the resolution-increasing effect, Massey and Bender proposed to use the middle frame of a video sequence as the reference frame.[9] The generated sprite will be much symmetric and the boundary area of the generated



(a) (b)

(c)

Fig. 3. Demonstration of geometric distorted frame due to camera rotation. (a) Original frame A. (b) Original frame B. (c) Transformed frame B.

sprite becomes much smaller if the background of the frames in the video sequence pans toward only one direction. On the other hand, this method only slows down the increasing effect of the sprite size, but the range of view angle is not extended.

Apart from the geometric distortion, the effect of camera zoom-in and zoom-out will highly affect the generated sprite and the reconstructed frames.[2] Figure 4 illustrates how camera zoom-in operation affects generated sprite and reconstructed frame. Since the reference coordinate system is based on the reference frame, the zoomed-in frame has to be scaled down in order to be merged into the sprite. Details of the zoomed-in frame are lost forever during the down-sampling and the reconstructed frame is degraded. In contrast to the camera zoom-in, the camera zoom-out operation makes the generated sprite looks blur after the zoomed-out frame is blended into the sprite. As Fig. 5 shows, the zoomed-out frame has to be up-sampled before blending into the sprite, this will make the blended sprite

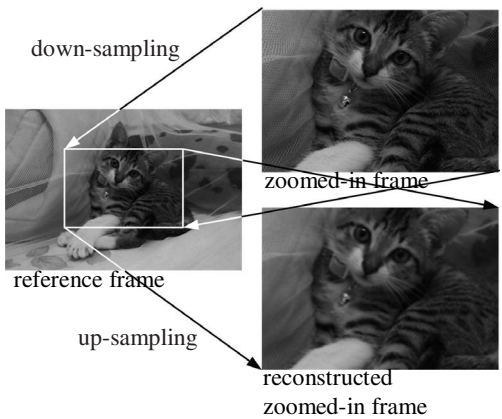


Fig. 4. Effect of camera zoom-in with the details of the reconstructed frame lost.

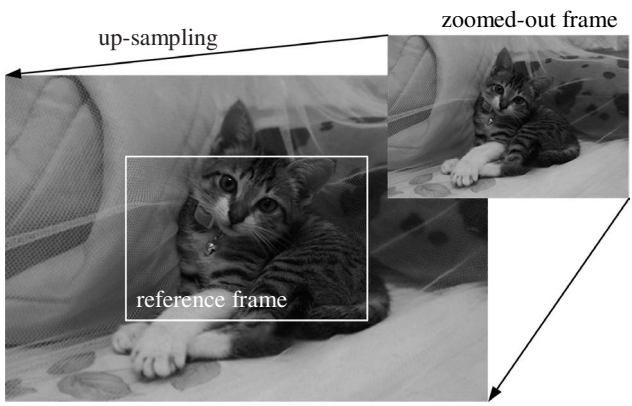


Fig. 5. Effect of camera zoom-out with the sprite blurred.

blurred. Furthermore, the up-sampled frame will occupy a very large area in the sprite, this causes the area of the generated sprite to be expanded rapidly.

In order to solve the above-mentioned problems, a technique using multiple sprites was proposed by Farin *et al.*[2] In their work, the background of a scene is stored by multiple sprites. In order to fit the MPEG-4 standard, a video sequence is divided into several subsequences, and sprites of all subsequences are generated independently. Figure 6 shows the geometric distortions using two sprites. In contrast to the geometric distortion using only one sprite shown in Fig. 2, the geometric distortion of frame B in sprite #2 becomes smaller. Furthermore, frame C, which is unable to be projected into sprite #1 can be projected into sprite #2 now. Full 360° of camera view may be covered if more sprites are used. Note that any single sprite must not cover 90° or more of camera rotation over any direction to prevent an effect called "degeneration".[1]

Farin *et al.*[2] have shown that using multiple sprites not only benefits the wider range of camera view angles but also reduces storage for the generated sprites. This means that storage required for multiple sprites is smaller than that for only one sprite.

As mentioned above, before generating multiple sprites, a video sequence must be divided into several subsequences. Each frame in the video sequence can be a partition position from which the video sequence is divided into subsequences. Thus, a partition algorithm is needed. For a video sequence with N frames, there will be 2^{N-1} combinations of partitions. Not only the partition position but also the reference frame of each subsequence must be selected by the partition algorithm. The selection of reference frames greatly affects the size of generation sprites. Each frame in a subsequence can be selected as the reference frame of the subsequence. If the sequence is divided into K subsequences, each subsequence has M_i frames, where i is the index of a subsequence. There will be $\prod_{i=1}^{K} M_i$ selections for reference frames. Farin *et al.* proposed an optimal multiple sprite partition algorithm. A cost

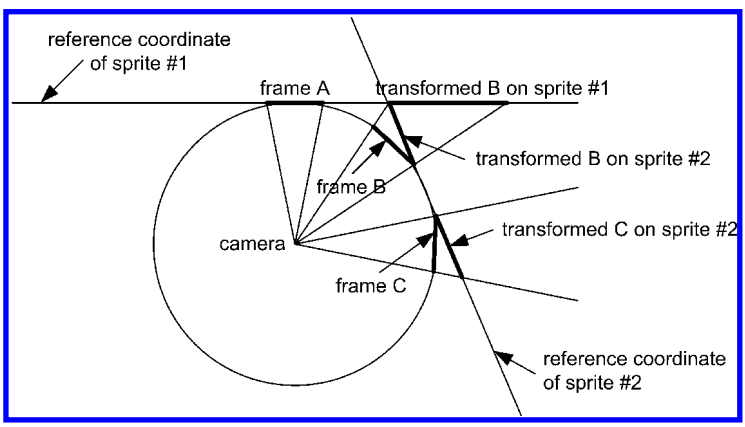


Fig. 6. Geometric distortions using two sprites.

function representing the total area of all generated sprites was defined, and a smart exhaustive search through the entire partition positions and reference frame possibilities was proposed. Since the partition algorithm is an exhaustive search, it finds the optimal solution. However, it is very time-consuming.

In our paper, a fast multiple sprite partition method will be proposed. The proposed method reduces the searching time for finding an applicable partition for multiple sprite generation, and the memory required during the searching is also decreased in contrast to the optimal partition method. Experimental results show that the coding cost of the generated sprites using the proposed method is near-optimum, i.e. only slightly higher than that in the optimal method.

The proposed method consists of two algorithms: video partition algorithm and a reference frame selection algorithm. The video partition algorithm is developed based on the characteristics of frame translations and scaling. The frame translation, which is caused by camera motion and also denoted as global translation, represents the movement of the background of a frame in the x- and y-directions relative to a reference frame. The global translations across frames are accumulated to represent the estimated position of a frame projected in a sprite. Since the geometric distortion depends on the accumulated global translation relative to the reference frame, the accumulated global translation provides a good measurement on the distortion. The effect of frame scaling caused by camera zoom-in or zoom-out can be employed in a similar way.

The reference frame selection algorithm is developed based on the idea of Messey and Bender's work.[9] In their work, the middle frame of a video sequence is suggested as the reference frame, since its background has higher possibility to be located at the center of a generated sprite. The proposed algorithm extends this idea by taking the frame with its background most likely being at the center of the corresponding sprite as the reference frame.

The rest of the paper is organized as follows. Section 2 describes the Farin *et al.*'s optimal partition method. In Sec. 3, the effect of frame translations and scalings are carefully analyzed and appropriate partition points and reference frames are explained. The proposed fast multisprite algorithm is presented in Sec. 4. Section 5 shows the experimental results and comparisons with the optimal method. Comparison of time and space complexity are made in Sec. 6. Conclusions are presented in Sec. 7.

2. Farin *et al.*'s Optimal Partition Algorithm

In order to find the optimal partition of a video sequence, an evaluation of partition results must be selected. In order to reduce the computational complexity, the area of the bounding box around a sprite is chosen to be the evaluation cost function in Farin *et al.*'s work. Their optimal partition algorithm is divided into two steps. The first step computes the minimal costs for coding sprites of all possible

subsequences and finds the optimal reference frames of every possible subsequence. The second step decides the optimal partition positions which minimize the coding costs computed in the first step. In the following, we will give a brief review for their optimal partition algorithm.

2.1. *Coding costs computing and reference frames finding*

A coding cost matrix holding the coding costs of all possible combinations of subsequences are computed in this step. For a subsequence beginning at frame i and ending at frame k, a sprite $S^r_{i;k}$ can be generated for a reference frame r with $i \leq r \leq k$. The coding cost for coding $S^r_{i;k}$ is denoted as $\|S^r_{i;k}\|$. $\|S^r_{i;k}\|$ of all possible combinations of i, k and r are computed. The number of possible combinations is huge and this computation will take a lot of time. After computing all $\|S^r_{i;k}\|$, the optimal reference frame $r^*_{i;k}$ of a subsequence beginning at frame i and ending at frame k can be selected by

$$r^*_{i;k} = \arg\min_r \|S^r_{i;k}\|. \tag{1}$$

The minimized coding cost of this subsequence, $\|S_{i;k}\| = \min_r \|S^r_{i;k}\|$, is also kept.

The optimal reference frames for every possible subsequence are found and stored in upper triangular matrixes indexed by i and k.

2.2. *Optimal partitioning*

In order to obtain the optimal partition, not only the starting frame (partition position) of each subsequence but also the number of subsequences must be decided. If the video is partitioned into n subsequences, $n-1$ starting frames need to be decided since the first subsequence always starts at frame 1. For a video with N frames, a partition for the video can be represented by

$$P = \{(1, p_1 - 1), (p_1, p_2 - 1), (p_2, p_3 - 1), \ldots, (p_{n-1}, N)\}, \tag{2}$$

where p_i is the starting frame of subsequence $i+1$. The sprite coding cost of a video sequence using a partition P is the summation of sprite coding costs of all partitioned subsequences, and the optimal partition P^* is selected as

$$P^* = \arg\min_P \sum_{(i,k)\in P} \|S_{i;k}\|. \tag{3}$$

The minimization problem is solved efficiently. If the video contains only the first frame, there is only one possible partition $P = \{(1,1)\}$ and the optimal sprite coding cost of the video is $\|S_{1;1}\|$ which is denoted as c_1. If the video contains more than one frame, the remaining frames are then added one by one. When adding

frame k, the optimal sprite coding cost c_k for the sequence ending at frame k can be calculated as

$$\begin{aligned} c_k &= \min_{i\in[1,k]}\{c_{i-1} + \|S_{i;k}\|\} \\ p_k &= \arg\min_{i\in[1,k]}\{c_{i-1} + \|S_{i;k}\|\}, \end{aligned} \tag{4}$$

where c_0 is set to zero. The frame p_k is the best partition point to obtain the minimal cost for each frame k.

After calculating c_N which is the minimal sprite coding cost for the entire sequence, the optimal partition can be obtained by back tracking the stored p-values from p_N. That is, the entire sequence is best partitioned at frame p_N to form two subsequences $(1, p_N - 1)$ and (p_N, N). The former subsequence is further divided at $p_{(p_N-1)}$, which is the best partition point of subsequence $(1, p_N - 1)$, and so on.

3. Proposed Candidate Partition Points Selection and Reference Frames Finding Methods

Farin *et al.*'s method achieved optimal partition result with high computational complexity, even if their efficient algorithm is applied. If we can reduce the possible combinations of subsequences and reference frames, the computational complexity will be reduced. In the following sections, we will first analyze the accumulated translations and scalings. Based on the accumulated translations and scaling, some candidate partition points and reference frames are then located first. Finally, a method is provided to get a near-optimal partition with similar bitrate to Farins'.

3.1. *Analysis of accumulated translation*

As mentioned previously, the geometric projection distortion comes from the camera motion, which is illustrated in Fig. 2. Farin *et al.*'s experiments[2] also show that the sprite area grows exponentially as camera panning angle increases. Thus the selecting of partition frames must be highly related to the effect of camera panning. In order to capture the effect of camera panning, the global translations between video frames are calculated and analyzed.

The global translation between two frames is a measurement of background displacement. Let frames i and j be the two frames to be measured and $\vec{p} = (x, y)$ be a pixel in frame j, the displacement of $\vec{p}$ relative to frame i is defined as

$$\vec{d}_p = T_{ji}(\vec{p}) - \vec{p}, \tag{5}$$

where T_{ji} is the geometric transformation applied in the frame warping which converts locations of pixels from the coordinate of frame j to frame i. Due to the effect of geometric transformation, the displacements of all pixels in frame j are

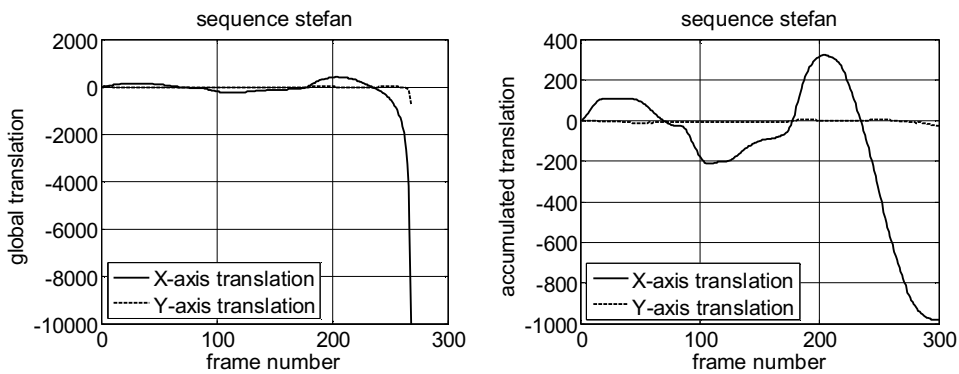
not consistent. In order to get a fast estimation of the frame displacement, the average of four corner displacements is used, that is,

$$\vec{t}_{ji} = \frac{1}{4} \sum_{P \in \{LT,RT,LB,RB\}} \vec{d}_P, \tag{6}$$

where LT, RT, LB, and RB are the left-top, right-top, left bottom and right bottom pixel of frame j. The $\vec{t}_{ji}$ is considered as the global translation between frames i and j.

The global translations of sequence "stefan" are illustrated in Fig. 7(a). The first frame is set to be the reference frame. The calculated translations of frames show their background displacements relative to the reference frame. A positive translation in the x-axis represents a frame displacement in the right direction, i.e. the frame is warped to the right side of the reference frame. A negative translation in the x-axis denotes a displacement in the left direction, and the y-axis translations can be denoted similarly. In Fig. 7(a), one can see that the view of background moves toward the right direction in the first 30 frames. Then it moves left and crosses the first frame in the next 90 frames. And then it moves toward the right again until frame #205, finally it moves toward the left in the rest of frames.

The figure also shows that the view of background moves toward the left quickly after frame #260. The magnitudes of global translations increase quickly from 100 to over 10,000 pixels. Actually, the magnitudes can be million pixels in short frames and tend to be infinity when a frame is unable to be projected into the first frame. The huge difference of magnitudes between frames is the result of huge geometric distortions when projecting a frame into the first frame with a view angle far away from the projecting frame. The huge difference makes it difficult to analyze the effect of camera translations from the global translations.



(a) (b)

Fig. 7. Background displacements of stefan. (a) Global translations. (b) Accumulated translations.

In order to analyze the camera translations efficiently, accumulated translation is proposed. The accumulated translations are calculated from the local translations, which represent the translation from one frame to its adjacent previous frame and can be denoted as $\vec{t}_{j(j-1)}$ in Eq. (6). Since local translations are less geometric-distorted than the global translations, they can be combined into an accumulated translation, $\vec{a}_{ji}$, to represent a less-distorted global translation between frames j and i, that is,

$$\vec{a}_{ji} = \sum_{k=i+1}^{j} \vec{t}_{k(k-1)}. \tag{7}$$

Note that $\vec{a}_{ji}$ can be computed by a recursive procedure:

$$\vec{a}_{ji} = \vec{a}_{(j-1)i} + \vec{t}_{j(j-1)}, \tag{8}$$

where $\vec{a}_{ii}$ is defined to be $(0, 0)$.

The accumulated translations for sequence "stefan" are illustrated in Fig. 7(b). In contrast to global translation, magnitudes of accumulated translations are limited into a reasonable range. The details of camera movements are still preserved, and the translations of all frames can be calculated, even for those frames that cannot be projected into the first frame.

3.2. *Accumulated translation based feasible partition point finding method*

One of the goals of a multisprite partition algorithm is to find some partition points to split the video sequence into several subsequences. Since the geometric distortions are the major issue of using multiple sprites, camera motion must be considered first. The following paragraph demonstrates the finding of feasible partition points, FPX, based on accumulated translations in the x-axis direction. In a similar way, we can also find the feasible partition points, FPY, based on accumulated translation in y-axis direction.

Figure 8 shows the x-axis accumulated translations which have been shown in Fig. 7(b). The camera pans to the right from the first frame to frame 29, then pans to the left until frame 107. When the camera begins the left-panning, the backgrounds of frames from 29 to 69 go back through an area that has been recorded into the current sprite. Since the background area already exists in the sprite, merging these frames into the sprite will not expand the sprite area. Thus, frames from 29 to 69 must not be selected as candidate partition points, and frames 70 to 107 are considered as candidate partition points.

Now, the camera pans to the right from frame 107 to 204, and backgrounds of frames from 107 to 183 have been recorded, thus they will not be considered as candidate partition points. For similar reasons, frames 204 to 244 are not considered as candidate partition points, and frames after 245 are considered as candidates. The candidates of partition points are illustrated by thick lines in Fig. 8. The

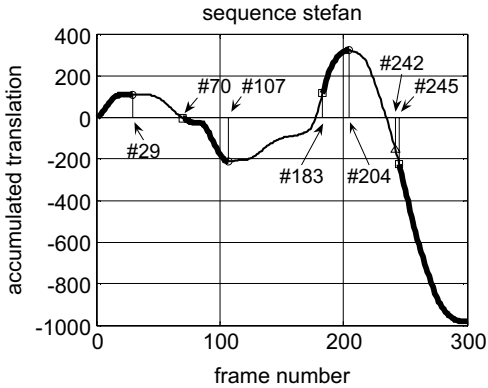


Fig. 8. Finding feasible partition points.

candidates of partition points can be grouped into several pieces, and each piece covers a small range of view angles. Since the covered view angle range in a piece is small, frames in the same piece should be merged into a sprite. The first frame in each piece is considered as a feasible partition point. If the candidates are grouped into K pieces, there will be $(K-1)$ feasible partition points, this will produce 2^{K-1} combination of possible partitions. In Fig. 8, feasible partition points are frames 70, 183 and 245. Comparing with Farin *et al.*'s result, which has an optimum partition point at frame 242, we have a feasible partition point at frame 245 which is very close to frame 242.

The above finding method is applied to both x- and y-axis directions, and two sets of feasible partition points (x- and y-axes) are found. The final feasible partition points are the union of the x- and y-axis partition points (FPX and FPY). Experimental result of a testing sequence based on accumulated translations in both directions is given in Sec. 5.

3.3. *The proposed scaling factor based feasible partition point finding method*

Using an accumulated translation exploits the effect of camera panning to partition a sequence. Therefore, if a sequence does not have camera panning, it will not be partitioned. Figure 9 shows the accumulated translations of sequence "tabletennis" from frame #1 to #131. The frames of the sequence are continually zoomed out with no camera panning. One can see that the values of accumulated translations fall into a very narrow range, that is, from +4 to −6. In this case, the accumulated translations are useless, and the effect of scaling must be considered.

The effect of scaling between two geometric transformed frames can be directly measured by the ratio of width or height of the transformed frames. However, due to the effect of geometric transformation, the width ratio and height ratio of two frames are not consistent. Instead of width or height ratio, ratio of area of two

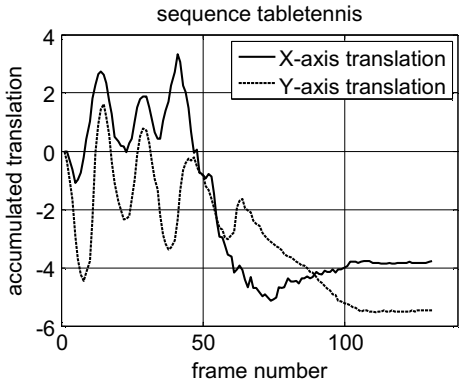


Fig. 9. Accumulated translations of "tabletennis".

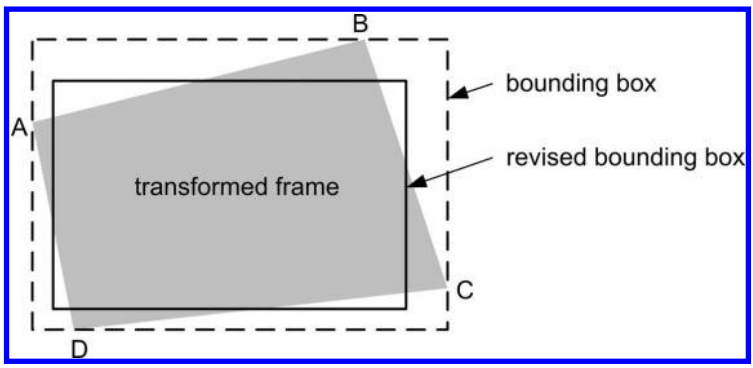


Fig. 10. Frame area calculation of a geometric-transformed frame.

frames is chosen in this paper. Area is the product of width and height, and area ratio will combine both the effect of width and height ratios. This makes area ratio more robust than width or height ratio.

An ordinary geometric transformed frame is illustrated in Fig. 10. The four corners of the transformed frame are A, B, C and D and the corresponding coordinates are denoted as (x_P, y_P), where $P \in \{A, B, C, D\}$. A fast approximation of the transformed frame area is to calculate the area of its bounding box, which is illustrated in a dash-lined rectangle in Fig. 10. However, one can see that the transformed frame is completely inside its bounding box. This makes the approximated transformed frame area oversized. For the purpose of making a better approximation, the bounding box is revised by correcting the boundaries of the bounding box as follows:

$$\text{left} = \frac{x_A + x_D}{2}$$

$$\text{right} = \frac{x_B + x_C}{2}$$

$$\text{top} = \frac{y_A + y_B}{2}$$
$$\text{bottom} = \frac{y_C + y_D}{2}. \tag{9}$$

The revised bounding box is shown by the solid-lined rectangle in Fig. 10. Clearly, the revised bounding box is closer to the actual area. Now the approximated area of the transformed frame j with frame i as a reference frame can be calculated by

$$\text{AREA}_{ji} = (\text{right} - \text{left}) \times (\text{bottom} - \text{top}). \tag{10}$$

After obtaining the areas of the transformed frames, the scaling factor of the transformed frame j versus the transformed frame i is defined as

$$s_{ji} = \sqrt{\frac{\text{AREA}_{ji}}{\text{AREA}_{ii}}}. \tag{11}$$

Since area is the product of width and height, a square root is applied to make the scaling factor linear. Similar to the computation of accumulated translations, the accumulated local scaling factors are employed to increase the robustness of scaling factor. The accumulated scaling factor of the transformed frame j versus the transformed frame i is defined as

$$as_{ji} = \prod_{k=i+1}^{j} s_{k(k-1)} = as_{(j-1)i} \times s_{j(j-1)}, \tag{12}$$

where as_{ii} is defined to be one. The accumulated scaling factors of the sequence "tabletennis" are illustrated in Fig. 11.

As mentioned previously, the quality degradation of a reconstructed frame is higher as the scaling between the frame and its reference frame increases. Therefore, the accumulated scaling of frames in a single sprite should be limited. This can be

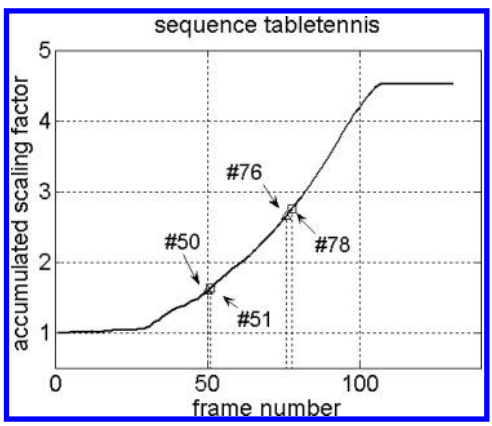


Fig. 11. Accumulated scaling factors of "tabletennis".

done by limiting the ratio of the maximum and minimum accumulated scaling factors of frames in a sprite generated from frame k to h. That is,

$$\frac{\max_i\{as_{ik}|k < i \leq h\}}{\min_i\{as_{ik}|k < i \leq h\}} < t, \tag{13}$$

where t is a threshold. According to Eq. (13), we process frames in a video sequence sequentially. When Eq. (13) is not satisfied at a certain frame h, frame h is considered as a feasible partition point. We keep processing the remaining frames with frame h as the starting frame to find all feasible partition points based on Eq. (13). In our experiment, the threshold t is set to 1.8.

The fixed threshold t in Eq. (13) has a disadvantage: the scaling ratio of the last partition will be smaller than that of the other ones. In the other words, the last partition covers less scaling range than others. This can be solved by applying an adjusted threshold

$$t' = \sqrt[L]{\max_i\{as_{i1}|1 \leq i \leq N\}}, \tag{14}$$

where L is the number of feasible partition points found based on Eq. (13) and threshold t, and N is the number of frames in the video sequence. The feasible partition points, FPS, based on the accumulated scaling factors are searched again by recalculating Eq. (13) with the adjusted threshold t'.

Figure 11 shows the feasible partition points found based on the accumulated scaling factors. The selected feasible partition points of sequence "tabletennis" are frames #51 and #78, which are marked by squares in Fig. 11. The optimal partition method[2] splits the sequence in frames #50 and #76, which are marked by triangles in Fig. 11.

3.4. *The proposed reference frame finding method*

The second goal of a multisprite partition algorithm is to find a good reference frame for each partitioned subsequence. In order to cover a larger view angle from both directions of panning, the frame with its view at the center of the view in the subsequence would be a good reference frame. Massey and Bender [11] suggested using the middle frame in a sequence as the reference frame, but the middle frame is not always at the center of view in a sequence. Here, we provide a method to get the reference frame with view near the center.

When a subsequence is partitioned, its maximum and minimum values of x-axis accumulated translations can be found. The maximum value represents the right view boundary, and the minimum value represents the left view boundary. The mean of the maximum and the minimum values will represent the view center, and the frame with accumulated translation closest to the mean value is selected as the reference frame.

4. The Complete Algorithm

The proposed multisprite partition algorithm is based on the methods described in Sec. 3. First, the accumulated translations and accumulated scaling factors are calculated and the feasible partition points based on accumulated translations and accumulated scaling factors are found separately. Then all feasible partition points based on translation and scaling are considered as candidate partition points. After finding the candidate partition points, the reference frames of all possible partitions are found. Based on these candidate partition points and reference frames, the second step of Farin *et al.*'s method is applied to obtain the final result.

4.1. *Candidate partition points and reference frames finding*

Let (X, Y) be the size of frame i and $T_{i(i-1)}$ be the geometric transformation converting locations of pixels from the coordinates of frame i to $i-1$. Let $(1, 1)$, $(1, Y)$, $(X, 1)$, (X, Y) be its four corners. Using Eqs. (5) and (6), we can obtain the local translation $\vec{t}_{i(i-1)}$. Based on the local translation, we can get the accumulated translations using Eqs. (7)–(8). Meanwhile, the locations of transformed corners in each transformed frame are used to find the revised bounding box via Eq. (9). The approximated area and accumulated scaling factors are calculated by applying Eqs. (10)–(12). Then three sets of feasible partition points (FPX, FPY, FPS) are found by the method described in Secs. 3.2 and 3.3. And the candidate partition points is set to be the union of these three sets of feasible partition points.

After obtaining the candidate partition points, the reference frames of all subsequences in all possible partitions are found using the method described in Sec. 3.4.

4.2. *Reference frame validation*

Although the reference frames are obtained in the previous section, the selected reference frame may be 1–3 frames away from the reference frame found by Farin *et al.* Since the size of a generated sprite is heavily affected by its reference frame, to get a more accurate reference frame, the neighboring frames of the selected reference frame are checked. If any of them achieves better result than the selected reference frame, the reference frame will be replaced by it.

Here, two methods using different validations for sprite area are proposed to find better reference frames. The first method is called normal validation. The bounding box of the corresponding sprite using each neighboring frame is calculated. Then the neighboring frame resulting in a minimal size of bounding box is selected as the validated reference frame. Since every frame in the current subsequence must be geometrically transformed toward each neighboring frame, this validation step achieves better performance with higher computational complexity.

The second method called fast validation tries to take a shortcut. Four boundary frames in the current subsequence are selected according to their accumulated translations: the frames with maximum and minimum values of x-axis translation, and

the frames with maximum and minimum values of y-axis translation. For each neighboring frame, only these four boundary frames are geometric transformed toward the neighboring frame. Then a bounding box covering these four transformed frames is found. The neighboring frame resulting in the corresponding bounding box with minimum area is taken as the validated reference frame.

4.3. *Sequence partition*

In order to find the proper partition points, the optimal partition algorithm[2] described in Sec. 2.2 is applied. The algorithm is originally developed to find the optimal partition from a coding-cost matrix made up by costs of all combinations of possible subsequences. In our method, the sequence is partitioned only in the candidate partition points found in Sec. 4.1. Therefore, a much smaller coding-cost matrix can be made up.

Assume that M candidate partition points $\{v_1, \ldots, v_M\}$ are found in Sec. 4.1. The first and the last frame N are added by setting $v_0 = 1$ and $v_{M+1} = N + 1$ to form a node set $V = \{v_0, \ldots, v_{M+1}\}$ of size $M + 2$. The area of bounding box $b_{i;j}$ of subsequence $(v_i, v_j - 1)$ beginning at frame v_i and ending at frame $v_j - 1$ for all possible $v_i, v_j \in V$ and $i < j$ can be obtained from the minimal area bounding box (the bounding box with validated reference frame) found in Sec. 4.2.

An upper triangle coding-cost matrix C with size $(M + 2) \times (M + 2)$ can be made by assigning

$$c_{i;j} = \begin{cases} b_{i;j} & \text{if } i < j \\ 0 & \text{otherwise} \end{cases}, \tag{15}$$

where i and j are indices of two elements in V.

The matrix C is applied to Eq. (3) as the sprite coding-cost matrix $\|S_{i;k}\|$. Then based on Eqs. (3) and (4), the partition of sequence can be found using the method described in Sec. 2.2.

5. Experimental Results

Identical global motion parameters of testing sequences should be used in all competitive methods to show the performance. However, the estimated global motion parameters will be different according to various estimation methods, different feature points and initial guesses used in the gradient descent algorithm. Since it is impossible to acquire the same estimated global motion parameters as those in Farins' paper, we choose to implement their optimal method. The global motion parameters are generated in advance by using our previous work on a high visual quality sprite generator,[7] and the same parameters are used in both the proposed and the optimal methods. The testing platform is an IBM laptop with mobile PentiumIII 800 MHz CPU and 640 MB of RAM. Both methods are implemented and simulated by Matlab. The results of using a single sprite are also included as a comparison.

Table 1. Experimental results of sequence "stefan" (perspective).

	Partitions (Reference Frames)	Total Sprite Size (Bytes)	Executing Time (Seconds)
Using a single sprite (frame #1∼#250)	—	2,862,240	—
Farin *et al.*'s optimal method	1–242 (57), 243–300 (265)	766,350	780
Proposed method with normal validation	1–244 (60), 245–300 (266)	777,160	44
Proposed method with fast validation	1–244 (53), 245–300 (265)	793,529	4.1

Table 2. Experimental results of sequence "stefan" (affine).

	Partitions (Reference Frames)	Total Sprite Size (Bytes)	Executing Time (Seconds)
Using a single sprite (frame #1∼#300)	—	1,450,446	—
Farin *et al.*'s optimal method	1–245 (81), 246–300 (283)	604,214	766
Proposed method with normal validation	1–244 (58), 245–300 (261)	608,685	44
Proposed method with fast validation	1–244 (57), 245–300 (259)	633,953	4.1

Table 3. Experimental results of sequence "tabletennis".

	Partitions (Reference Frames)	Total Sprite Size (Bytes)	Executing Time (Seconds)
Using a single sprite	—	620,044	—
Farin *et al.*'s optimal method	1–49 (48), 50–75 (52), 76-131 (76)	177,766	95
Proposed method with normal validation	1–51 (9), 52–77 (57), 78–131 (83)	190,150	9.3
Proposed method with fast validation	1–51 (4), 52–77 (52), 78–131 (78)	220,964	2.7

Tables 1 and 2 show the results of sequence "stefan" using perspective and affine motion model respectively. Table 3 shows the results of sequence "tabletennis" in perspective model. Note that the experimental results of the optimal method are different from the results described in their paper[2] because the global motion parameters of sequences are not the same. In all tables, we can see that Farins' optimal method achieves excellent performance. The total sprite sizes of all sequences by the optimal method are superior to the sizes of using a single sprite. The performance of using multiple sprites is obvious. Results also show that using affine motion model slightly reduces execution time because the affine transformation is a bit faster than perspective transformation, but the execution time of the optimal method is still very slow.

The results of the proposed method with normal validation and fast validation are also listed in tables. The proposed method divides the sequence "stefan" into partitions at frame 245 and divides the sequence "tabletennis" into three partitions at frames 52 and 78. The partition points of the proposed method are very close to the partition points of using the optimal method, which is frame 243 in "stefan" and frames 50 and 76 in "tabletennis".

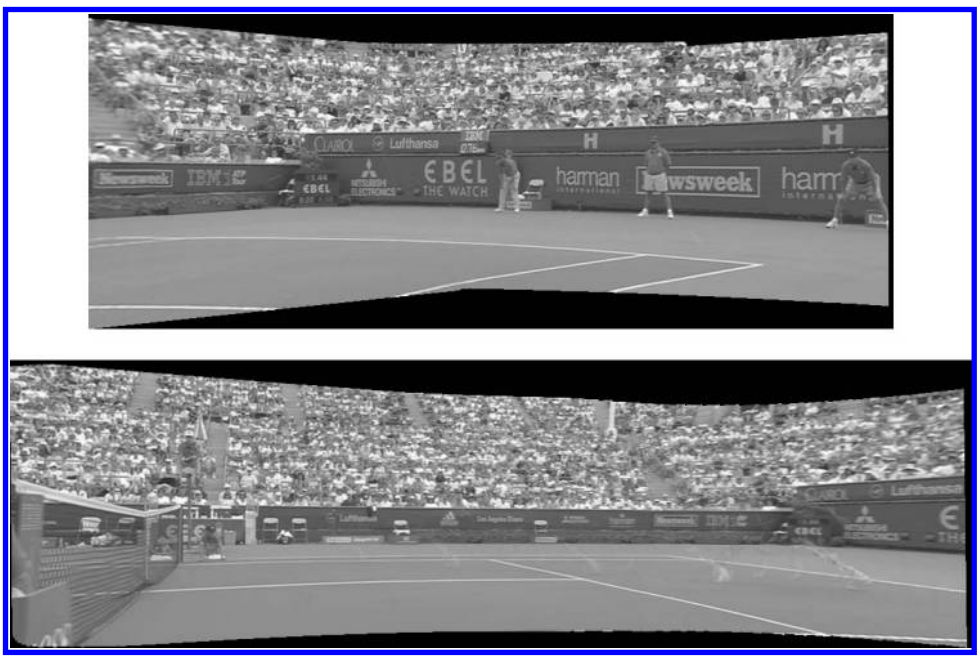
The total sprite sizes using the proposed method are only slightly higher than those using the optimal method, but the executing time of the proposed method are greatly reduced. The total sprite sizes of two testing sequences using the proposed method with normal validation are 777,160 and 190,150 pixels respectively, which are only 1.41% and 6.97% higher than total sprite sizes of using the optimal method. The executing times are reduced from 780 seconds to 44 seconds, and 95 seconds to 9.3 seconds. The execution speed is increased over ten times. If the fast validation method is applied, the executing times can be further decreased to 4.1 s and 2.7 s, which is 35–190 times faster than that of the optimal method. In contrast to the reduction of executing time, the total sprite size of using fast validation method is not increased much.

The generated sprites of sequence "stefan" by the optimal method and the proposed methods are shown in Fig. 12 respectively. We can see that the generated sprites are perceptually similar, excepting for the dimensions of sprites. The effects of geometric distortions are lightened and the qualities of generated sprites are

(a)

Fig. 12. Generated sprites of sequence "stefan" by different methods. (a) Farin *et al.*'s optimal method. (b) Proposed method with normal validation. (c) Proposed method with fast validation.

(b)



(c)

Fig. 12. (*Continued*)

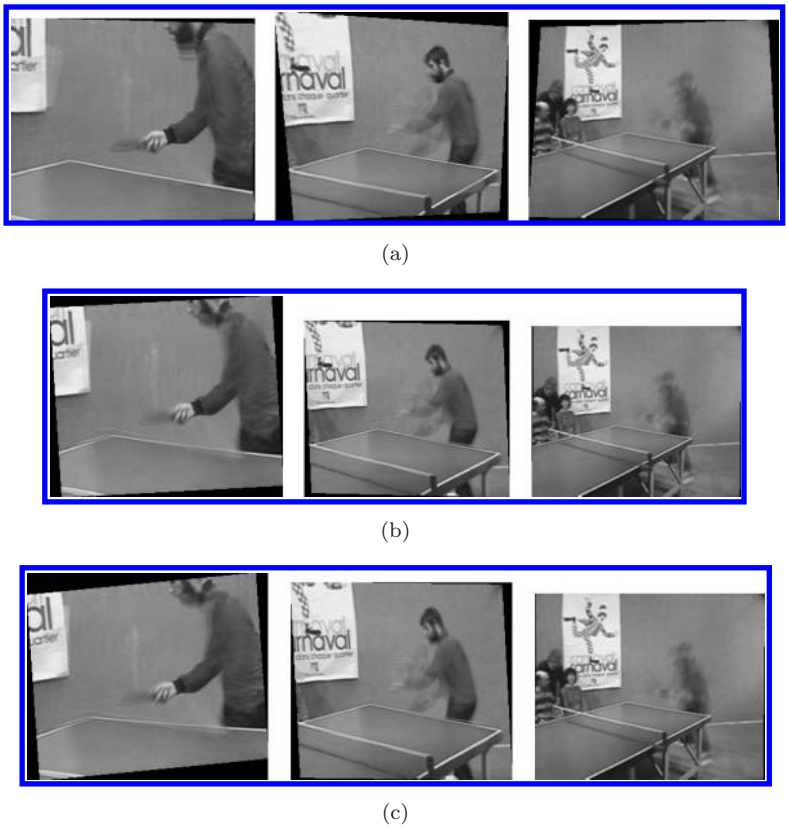


Fig. 13. Generated sprites of sequence "tabletennis" by different methods. (a) Farin *et al.*'s optimal method. (b) Proposed method with normal validation. (c) Proposed method with fast validation.

preserved. The generated sprites of sequence "tabletennis" by different methods are shown in Fig. 13.

The experimental results of sequence "building" are listed in Table 4. The sequence consists of wide camera movements in y-axis direction and continuous panning in x-axis direction. From Table 4, one can see that the proposed method works well. Figure 14 shows the generated sprites of the sequence.

Table 4. Experimental results of sequence "building".

	Partitions (Reference Frames)	Total Sprite Size (Bytes)	Executing Time (Seconds)
Farin *et al.*'s optimal method	1–12 (10), 13–39 (31), 40–65 (56)	607,265	22
Proposed method with fast validation	1–29 (10), 30–41 (37), 42–65 (56)	614,052	2.2

(a)

(b)

Fig. 14. Generated sprites of sequence "building" by different methods. (a) Farin *et al.*'s optimal method. (b) Proposed method with fast validation.

6. Complexity Analysis

Complexity can be discussed in two different ways: time and space. Both complexities of the proposed and optimal methods are discussed.

The complexity of Farins' optimal method is divided into two parts: the building of coding cost matrix described in Sec. 2.1, and the optimal partition algorithm described in Sec. 2.2. While building the cost matrix, the coding cost of all subsequences beginning at frame i and ending at frame k with reference frame r must be computed. Suppose that the sequence has N frames, the time and space complexity of building a cost matrix will be N^3. However, a method was developed to reduce the space required for N^2. The optimal partition algorithm using Eq. (4) to find the best partition frame-by-frame takes N^2 time.

The proposed method calculates the accumulated translation and scaling first, and both of them take linear time. The finding of candidate partition points also requires linear time because it only observes the changes of accumulated translation and scaling once. Let M be the number of candidate partition points found in Sec. 4.1, finding reference frame for all possible subsequences takes $M^2 \times N$ time. Finally, the Farins' optimal partition is applied. Since only M candidate partition points are involved, it takes only M^2 time. The accumulated translations and scalings must be held in memory and the coding-cost matrix must be generated. These will need $2N + M^2$ space.

Table 5 shows the complexity of both methods. The proposed method takes $N + N + M^2 \times N + M^2$ time, i.e. $O(M^2 \times N)$ in time and $O(M^2) + O(N)$ in space.

Table 5. Complexity comparison.

	Time	Space
Farin *et al.*'s optimal method	$O(N^3)$	$O(N^2)$
Proposed method	$O(M^2N)$	$O(M^2)+O(N)$

Since M is usually very small in contrast to N in practice, for example, $M = 6$ and $N = 300$ in the sequence "stefan", this makes the complexity of the proposed method better than the optimal method.

7. Conclusions

An efficient and fast method for generating multiple sprites is proposed. In contrast to the conventional sprite generating method that uses a single sprite, using multiple sprites reduces the storage space of sprites. Conventional multiple sprite generation method uses an exhaustive search to find the optimal subsequence partition and optimal reference frame of each partition. However, the exhaustive search costs a lot of time. The proposed method consists of a subsequence partition algorithm and a fast reference frame selection algorithm, which are developed based on the frame accumulated translations and scalings. By using the proposed methods, a video sequence can be partitioned and reference frame can be selected in a very short time. In order to increase the performance of the selected reference frames, two reference frame validation methods are also proposed. The proposed validation methods searches frames close to the result of the fast reference frame selection method and checks if better reference frame exists. The experimental results also show that the proposed method greatly increases the executing speed from 10 to 190 times in contrast to the Farins' optimal method, the total sprite size is slightly higher and the qualities of generated sprites are preserved.

Acknowledgments

This work is supported in part by National Science Council under the contract number NSC-96-2221-E-009-201-. The authors would like to thank the anonymous reviewers for their many valuable suggestions which have greatly improved the presentation of the paper.

References

1. D. Farin and P. H. N. de With, Enabling arbitrary rotational camera motion using multisprites with minimum coding cost, *IEEE Trans. Circuits Syst. Vid. Technol.* **16**(4) (2006) 492–506.
2. D. Farin, P. H. N. de With and W. Effelsberg, Minimizing MPEG-4 sprite coding-cost using multisprites, *SPIE Vis. Commun. Imag. Process.* **5308** (2004) 234–245.
3. D. Farin, P. H. N. de With and W. Effelsberg, Video-object segmentation using multi-sprite background subtraction, *IEEE Int. Conf. Multimedia and Expo (ICME '04)* (June 2004), pp. 343–346.
4. ISO/IEC MPEG Video Group, MPEG-4 video international standard with amd. 1, ISO/IEC 14496-2 (Jully 2000).

5. ISO/IEC MPEG Video Group, MPEG-4 video verification model version 18.0, N3908 (January 2001).
6. S. Kopf, T. Haenselmann, D. Farin and W. Effelsberg, Automatic generation of video summaries for historical films, *IEEE Int. Conf. Multimedia and Expo (ICME '04)* (June 2004), pp. 2067–2070.
7. I. S. Kuo and L. H. Chen, A high visual quality sprite generator using intelligent blending without segmentation masks, *Int. J. Patt. Recogn. Artif. Intell.* **20**(8) (2006) 1139–1158.
8. Y. Lu, W. Gao and F. Wu, Efficient background video coding with static sprite generation and arbitrary-shape spatial prediction techniques, *IEEE Trans. Circuits Syst. Vid. Technol.* **13**(5) (2003) 394–405.
9. M. Massey and W. Bender, Salient stills: process and practice, *IBM Syst. J.* **35**(3–4) (1996).
10. A. Smolić, T. Sikora and J-R. Ohm, Long-term global motion estimation and its application for sprite coding, content description, and segmentation, *IEEE Trans. Circuits Syst. Vid. Technol.* **9** (1999) 1227–1242.
11. R. Szeliski, Video mosaics for virtual environments, *IEEE Comput. Graph. Appl.* **16**(2) (1996) 22–30.
12. H. Watanabe and K. Jinzenji, Sprite coding in object-based video coding standard: MPEG-4, *World Multiconference on Systemics, Proc. Cybernetics and Informatics (SCI)* **XIII** (July 2001) 420–425.

I-Sheng Kuo received the B.S. and M.S. degrees in electrical engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 1997 and National Cheng Kung University, Tainan, Taiwan, in 1999, respectively. He is currently a Ph.D. student in the Institute of Computer Science and Engineering at National Chiao Tung University, Hsinchu, Taiwan.

His research interests include image processing, video and image compression, MPEG-4 and H.264.

Ling-Hwei Chen received the B.S. degree in mathematics and the M.S. degree in applied mathematics from National Tsing Hua University, Hsinchu, Taiwan in 1975 and 1977, respectively, and the Ph.D. degree in computer engineering from National Chiao Tung University, Hsinchu, Taiwan in 1987.

From August 1977 to April 1979 she worked as a research assistant in the Chung-Shan Institute of Science and Technology, Taoyan, Taiwan. From May 1979 to February 1981 she worked as a research associate in the Electronic Research and Service Organization, Industry Technology Research Institute, Hsinchu, Taiwan. From March 1981 to August 1983 she worked as an engineer in the Institute of Information Industry, Taipei, Taiwan. She is now a Professor in the Department of Computer Science at the National Chiao Tung University.

Her current research interests include image processing, pattern recognition, multimedia compression, content-based retrieval and multimedia steganography.