國立交通大學

資訊科學與工程研究所

博 士 論 文

一個關於切割影片以產生單一/多重場景背景之研究

A Study on Single/Multiple Sprite Generation and Partition for Videos

研 究 生：郭萓聖

指導教授：陳玲慧　教授

中 華 民 國 九 十 七 年 七 月

一個關於切割影片以產生單一/多重場景背景之研究
A Study on Single/Multiple Sprite Generation and Partition for Videos

研 究 生：郭萓聖　　　Student：I-Sheng Kuo

指導教授：陳玲慧　　　Advisor：Ling-Hwei Chen

國 立 交 通 大 學
資 訊 科 學 與 工 程 研 究 所
博 士 論 文

A Dissertation
Submitted to Institute of Computer Science and Engineering
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in

Computer Science

July 2008

Hsinchu, Taiwan, Republic of China

中華民國九十七年七月

一 個 關 於 切 割 影 片 以 產 生 單 一 / 多 重 場 景 背 景 之 研 究

學生：郭萓聖　　　　　　　　　　　　指導教授：陳玲慧　教授

國立交通大學資訊科學與工程研究所博士班

摘　　要

以物件為基礎編碼的 MPEG-4 採用了一種創新的場景背景編碼方式，該方法可以提高背景部分的編碼效率。MPEG-4 所提出的場景背景產生系統，使用算數平均將所有影像疊合以產生場景背景影像，然而這樣的方式會使得產生出的場景背景影像中某些區域變得模糊，特別是曾經被移動物件佔據過的位置。為了防止產生背景模糊的狀況，MPEG-4 建議使用者提供一個物件分割遮罩，標示畫面中屬於移動物件的位置，以避免物件被混入場景背景之中。我們依照 MPEG-4 所提出的架構，建立一個場景背景產生系統。但手動產生所有畫面的物件分割遮罩是不切實際的，因此在所建立的系統中，我們提出一個自動化物件分割遮罩產生方法。該方法先以不使用物件遮罩的方式產生粗糙場景背景，而後以粗糙背景為參考影像產生物件分割遮罩，最後再以所產生的遮罩重新產生較佳的場景背景。實驗結果顯示所提的系統產生之場景背景，具有良好的視覺品質。

自動化影像分割方法所產生的物件分割遮罩，不可能非常完美的將所有移動物件與背景區分。未正確區分的物件分割遮罩，會使得部分移動物件被混合入背景影像之中。導致所產生的場景背景影像中，出現如鬼影般的移動物件殘骸。為了解決這個問題，我們將提出一個不需要物件分割遮罩的場景背景產生系統。所提出的系統包含兩個新方法：均勻化特徵點擷取方法與智慧型影像疊合方法。提出的特徵點擷取方法估計背景的運動向量，利用該向量將特徵點中屬於移動物件的點予以排除。同時以均勻化的擷取方式平均分散所有特徵點的位置。提出的智慧型影像疊合方法使用一種計數方法，使得只有屬於背景的點被混合入場景背景影像之中。實驗結果顯示所提出的均勻化特徵點擷取方法能有效的提高全域運動估計的準確

性，因而提高場景背景影像之品質。提出的智慧型影像疊合方法則能夠將物件排除在疊合過程以外，使得以提出之方法產生的場景背景影像，不存在分割失誤可能導致的鬼影現象。其視覺品質接近使用人工產生之物件分割遮罩產生的場景背景影像，並優於 Smolic *et al*.提出之使用自動化物件分割之場景背景產生方法。

場景背景產生系統中，應用了幾何轉換將非參考畫面轉換至參考畫面的座標系統。進行幾何轉換會使轉換後畫面，以及根據轉換後畫面疊合的場景背景影像變的歪曲。這使得場景背景影像所需要的儲存空間增加，同時亦限制了場景背景影像所能夠涵蓋的視角。對此 Farin *et al*.提出了使用多重場景背景的方式解決問題。使用多張場景背景影像所需的儲存空間總和，有可能較使用單一場景背景影像來的小，同時亦能涵蓋較大範圍的視角。然而 Farin *et al*.所提出的方法，利用暴力搜尋法找出最佳的影片分割位置。若有 N 個畫面，這樣的方法需要 $O(N^3)$的執行時間與 $O(N^2)$的儲存空間。為了降低運算的複雜度，我們提出一個快速的多重場景背景影片分割方法。該方法包含一個可能的分割位置選取方法以及一個快速參考畫面選擇方法。利用測量畫面之間的移動與縮放，以找出影片中有可能的分割位置。並由這些可能的分割位置尋得最終的分割位置，將影片分割為數個子影片，最後每一個子影片將產生一個場景背景影像。若所提出的方法找到 M 個可能的分割位置，則所提出的方法僅需要 $O(M^2N)$的執行時間以及 $O(M^2)+O(N)$的儲存空間。同時所產生的數個場景背景影像的總儲存空間僅較暴力搜尋法所產生的略高。

A Study on Single/Multiple Sprite Generation and Partition for Videos

Student：I-Sheng Kuo　　　　Advisors：Dr. Ling-Hwei Chen

Institute of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

Sprite coding, which can increase the coding efficiency of backgrounds greatly, is a novel technology adopted in MPEG-4 object-based coding. The sprite generator introduced in MPEG-4 blends frames by averaging blending, this will make some places, which are ever occupied by moving objects, look blurring. Thus, providing segmented masks for moving objects is suggested. We build a sprite generation system based on MPEG-4's framework, but we find that using manual segmentation masks in a sprite generation system is impractical. An automatic segmentation mask generation method is proposed and is applied in the sprite generation system. The sprite generation system produces a coarse sprite first by MPEG-4's method without segmentation masks. Then the coarse sprite is employed as the reference image in the proposed segmentation mask generation method. After generating the segmentation masks, a better sprite is re-generated again with generated segmentation masks. Experimental results show the sprite generated by the proposed system has good quality.

Automatic image segmentation can not produce perfect object segmentation masks. Segmentation faults in segmentation masks causes some moving objects being blended into a sprite. This makes some ghost-like shadows appear in a generated sprite. To treat this problem, a sprite generation without segmentation masks is proposed in this dissertation. The proposed sprite generator consists of two novel methods: a balanced feature point extraction method and an intelligent blending method. The feature point extraction method estimates the motion vector of background pixels, and excludes pixels of moving objects from the feature points. Proposed intelligent blending method blends only background pixels into a sprite by a simple counting schema. Experimental results show the feature points extracted by the proposed method increases the accuracy of global motion estimation, and the quality of generated sprites is increased. The proposed intelligent blending method excludes pixels of moving objects directly in the blending procedure. Thus ghost-like shadows caused by segmentation faults is not exist in the sprite generated by our method. The visual quality of our sprite is close to that using manually segmented masks and is better than that generated by Smolic *et al.*'s method.

Due to the geometric transformation applied to each non-reference frame in the procedure of sprite coding, the generated sprite is distorted and the available view angles relative to the reference frame are restricted. This makes multiple sprites used be necessary. An optimal multiple sprite generation method has been proposed by Farin *et al.*, but it uses an exhaustive search to find the optimal partition and reference frames. Let N be the number of frames, Frains' method requires $O(N^3)$ time and $O(N^2)$ space to perform the search. In order to reduce the complexity, a fast multiple sprite partition method is proposed in this dissertation. The proposed method includes a fast partition point finding method and a fast reference frame finding method. The proposed partition point finding method measures translation and scaling between frames and finds candidate partition points by the measured values. The final partition positions are decided from these candidate points, and reference frames of each partition are found by the proposed fast reference frame selecting method. Let M candidate partition points are found, the proposed method requires only $O(M^2N)$ in time and $O(M^2)+O(N)$ in space. The total size of generated sprites is only slightly higher than that of Farin's method.

誌　　　謝

這篇論文能夠完成，首先要感謝指導教授陳玲慧博士的細心指導與協助。於陳教授身上，我不僅學習到豐富的學業知識與嚴謹的研究態度，同時亦對待人接物的道理以及為人處事應有的堅持有更深的瞭解。誠摯的感謝陳教授多年來的指導。感謝口試委員莊仁輝教授、荊宇泰教授、陳永昇教授、張隆紋教授、鍾國亮教授、尹邦嚴教授及李建興教授對論文與研究提供許多寶貴的指導與建議，使得本論文更加完善。

接著要謝謝自動化資訊處理實驗室一同研究的伙伴們。建興學長、遠坤學長、坤龍學長、昭玲學姊以及瑞祥學長，學長姊們在研究過程中給予的指導與叮嚀，提供很大的幫助。感謝博士班同學民全、文超、惠龍及俊旻，大自學術上的研討，小至生活上的閒談，都讓我在交大的生活更加充實。瓊雪、佩瑩、立人、芳如、信嘉、薰瑩、子翔、偉全等幾位學弟妹，有你們在，實驗室總是充滿歡樂與愉悅的氣氛。同時也要感謝系上助理耀萱，因為她的大力幫忙，我才能夠如期完成畢業口試。

此外我也要感謝從小到大求學期間認識的好朋友們：一同通車上下學，經常能夠仗義執言的君燦；經常聊天聊到深夜，互相通報八卦，一起想鬼點子，無話不談的好朋友怡慧；細心體貼、總是為人著想的肚魚以及點子很多、經常帶來歡笑的世懷。特別謝謝草莓公主，這幾年妳的關心與體諒，讓我能夠沒有壓力的完成學位。謝謝三年來陪伴在身邊並慷慨提供自拍照於論文中的百香果。還有中山大學滋青社的社團伙伴們，讓我在大學時期經歷許多美好的體驗與成長；經常給予我人生中許多中肯建議的韶玲老師，以及諮商中心志工團的朋友們。

最後我要感謝一路栽培我的父母、舅舅以及兄長於生活上的諸多照料，讓我無需擔心經濟壓力，專心攻讀博士學位。謹以最誠摯的心意將本篇論文獻給我的父母，也獻給曾經在我的人生中給予我鼓勵及協助的每一個人。

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATION

FPX	Feasible Partition Point based on *X*-axis Translation
FPY	Feasible Partition Point based on *Y*-axis Translation
FPS	Feasible Partition Point based on Scaling
GME	Global Motion Estimation
GMP	Global Motion Parameter
MPEG	Moving Picture Expert Group
MPEG-4 VM	MPEG-4 Verification Model
MSE	Mean-Squared Error
PSNR	Peak Signal-to-Noise Ratio

CHAPTER 1
INTRODUCTION

1.1 Motivation

MPEG-4 [1] 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’ or a ‘background mosaic’ [2-6]. 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 [7-11]. The extracted moving objects and the sprite itself can be used in video summarization [12].

A sprite is constructed in the encoder by a sequence of complex algorithms called a ‘sprite generator’. MPEG-4 VM [13-14] has provided a framework of sprite generator as shown in Fig. 1.1. The framework contains three parts: global motion estimation (GME), frame warping and frame blending. The GME aims at finding the spatial location variation, which is caused by the camera motion, of the background in the current frame relative to the current sprite. The camera motion can be represented by parameters of some geometric models often denoted as global motion parameters (GMP). Gradient descent based algorithms [15-17] are widely used in the estimator. These algorithms usually need a good initial guess to avoid the solution being local optimum, and an error function is required to evaluate the performance of different GMPs. The squared error between the current frame and the current sprite is usually employed as the error function. In order to raise speed, only some points are selected as feature points and involved in the computation of the error function [18-21]. The selection of feature points decides the estimation accuracy, especially when the number of feature points is small. Thus, how to select few representative feature points is an important issue. Most existing methods [18-21] take those points with higher variations in spatial or temporary domain; this will lose some important ones with moderate variations and will include some moving object points with higher variations. To avoid this disadvantage, in this dissertation, a balanced feature point extractor is provided to increase the estimation precision for GMPs.

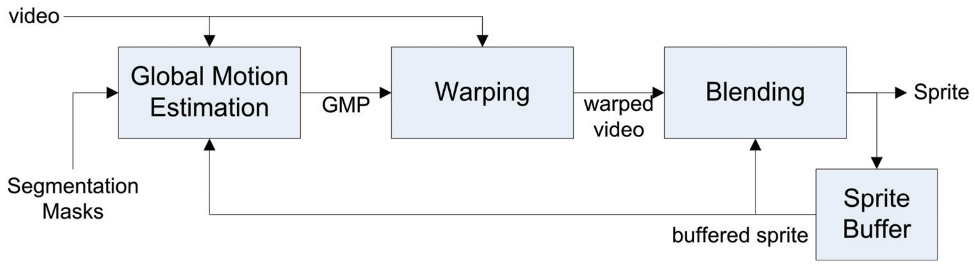


Fig. 1.1 The framework of the sprite generator in MPEG-4 VM.

After obtaining GMPs, the current frame is first geometrically transformed (also called warped) to a warped one with the same camera view as the current sprite. The warped frame is then blended into the current sprite by a blending strategy. In MPEG-4's framework, averaging blending is employed as the blending function. This will make some places, which are ever occupied by moving objects, blurred. To avoid the disadvantage, providing manually-segmented masks have been suggested. The segmentation masks are used to distinguish moving objects from background such that moving objects will not be involved in blending and the quality of the generated sprite can be significantly improved. However, segmenting moving objects manually from the video frames is impractical. In this dissertation, two approaches will be provided to increase the blending quality. The first approach generates these segmentation masks from a coarsely generated sprite without these masks. Then a better sprite is generated using the automatically generated masks. The second approach tries to get rid of the segmentation masks by developing a new blending method that excludes moving objects from being blended into the sprite.

The video sequence is inputted frame by frame into the sprite generator. The sprite buffer holds the sprite generated so far and provides the current sprite to the GME process as the reference image.

1.2 Basic of Sprite Generation

As Fig. 1.1 shows, a sprite generator can be divided into three parts. The global motion estimation registers pixels between two frames and estimates the camera motion between two frames. The warping transforms a frame into another frame's coordinate according to the camera motion estimated in the global motion estimation. Finally, the blending process blends the transformed frames and merges them into a single sprite. These parts will be introduced as followings.

1.2.1 Global motion estimation

The aim of global motion estimation is to obtain an accurate estimation of background motion between the current frame and a reference image, which is often the previous frame or the current sprite. Many global motion estimation methods have been proposed [22-26]. Before estimating the background motion, a motion model must be chosen to describe the motion of background [27]. Affine transformation and perspective transformation are widely used as the motion model in the generation of a sprite. Then an image registration method [28] is applied to find corresponding pixel pairs that belong to the same location of background in both images, and the global motion parameters are estimated by iterative minimization methods.

1.2.1.1 Motion model selection

The motion of background comes from camera motion, like zooming, panning, and rotation. These motions can be modeled by a geometric transformation. One of the two transformations is often chosen as the camera motion model: the affine transformation and the perspective transformation. Both of them are defined by two equations with a set of parameters and as follows:

$$\text{Affine Transformation:} \quad \begin{cases} x' = m_1 x + m_2 y + m_3 \\ y' = m_4 x + m_5 y + m_6 \end{cases} \tag{1.1}$$

$$\text{Perspective Transformation:} \quad \begin{cases} x' = \dfrac{m_1 x + m_2 y + m_3}{m_7 x + m_8 y + 1} \\ y' = \dfrac{m_4 x + m_5 y + m_6}{m_7 x + m_8 y + 1} \end{cases} \tag{1.2}$$

where (x,y) and (x',y') denote the coordinates of a pixel before and after the camera motion respectively. $m_1, m_2, \ldots, m_8$ are the transformation parameters also referred as global motion parameters (GMPs). We can see that the affine transformation with six parameters is a special case of the perspective transformation with $m_7=m_8=1$. The perspective transformation can describe more complicated camera motions. However, the higher computational complexity of the perspective transformation limits its usability. The affine transformation is quite simple, but it has a lower accuracy. Both of them are adopted as standard tools in MPEG-4. In most of sprite generators, the perspective transformation is selected since the sprite is usually generated first before coding the video sequence itself and real-time processing is not required.

1.2.1.2 Parameter estimation

By using the selected transformation, the global motion estimation can be converted to a minimization problem:

$$P^* = \arg\min_P E(I, I', T_P) \tag{1.3}$$

where I and I' are the current frame and the reference image respectively. T_P is the transformation function with global motion parameters P. P^* is the estimated parameters. $E(I,I',T_P)$ is an error function defined by user. The estimation registers pixels in the current frame into the reference image by finding the parameters which minimize the error between the current frame and the reference image. In this dissertation, the squared error is chosen as the error function, that is,

$$E(I, I', T_P) = \sum_{x,y \in I} \left(I(x,y) - I'(T_P(x,y)) \right)^2 \tag{1.4}$$

Due to the complication of global motion parameters, Least-Mean-Square minimization method is used to find the parameters P^*. For example, the Levenberg-Marquardt algorithm [15] based on the gradient descent method can be used. Gradient descent based methods search for better parameters around the current parameters, and refine the parameters iteratively. Thus they have a risk of being trapped into a local minimum. A starting point called an ‘initial guess’ must be provided. A good initial guess can reduce the risk of being trapped into a local minimum and can also speed up the refinement process.

In order to provide a good and robust initial guess, global motion estimation is usually processed in two stages [18]. To generate a sprite from a video sequence, the first frame in the sequence is copied into the sprite directly. Then camera motions between the following frames and the first frame must be estimated by the global motion estimation. In the beginning of the sequence, it is easy to find a good initial guess because the camera does not move too far. It becomes hard to find a good initial guess when the camera motion of the current frame relative to the current sprite is large. This problem is solved by a two-stage GME schema shown in Fig. 1.2. The first stage estimates the motion parameters called the local parameters between the current frame and its previous frame, i.e., the frame before the current frame. Finding an initial guess of the local parameters is easy because the variation of camera motion between two successive frames is small. Based on the estimated local parameters, the initial guess of global motion parameters can be computed in the second stage by combining the local parameters and the global motion parameters of the previous frame. Then the gradient descent method is employed to estimate the global motion parameters.

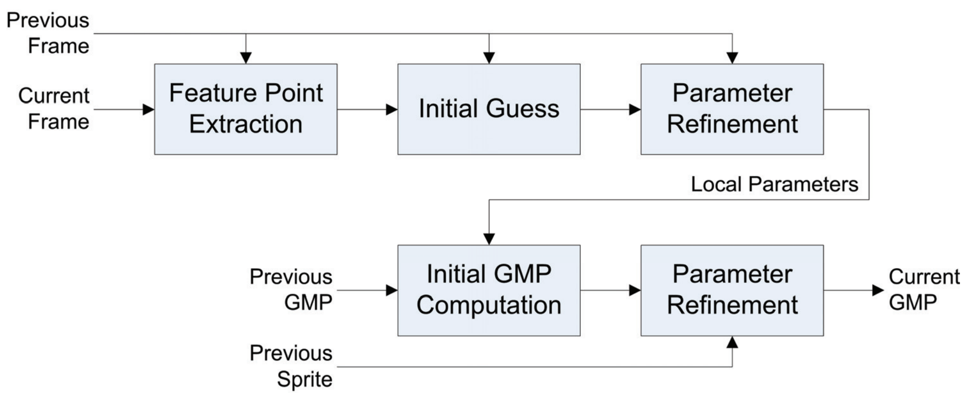


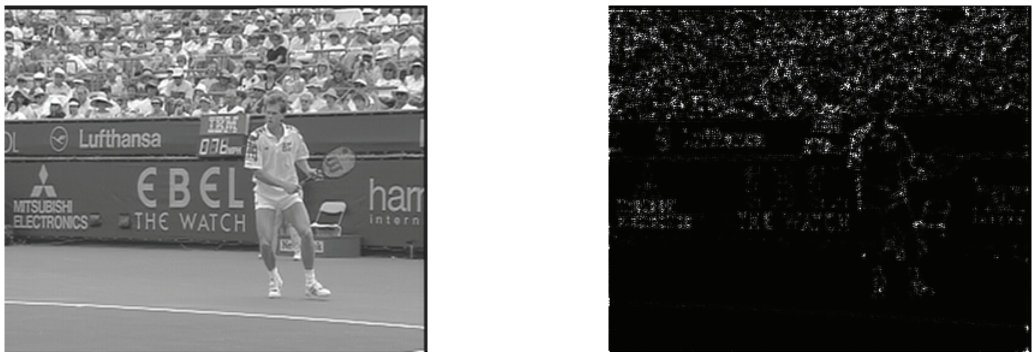
Fig. 1.2 The two-stage GME schema.

1.2.1.3 Feature points selection

The iterative minimization of the gradient descent method is time consuming. To reduce the time complexity, only some selected feature points in the current frame are employed while computing the registration error [18]. In order to avoid the aperture problem [29], the Hessian value [30], defined by

$$H(x,y)=\left(\frac{d^2I(x,y)}{dx^2}\cdot\frac{d^2I(x,y)}{dy^2}-\left(\frac{d^2I(x,y)}{dxdy}\right)^2\right) \qquad (1.5)$$

is employed to find feature pixels in many previous researches [18-21, 23-24]. Those points with Hessian values being local maximum or minimum are considered as feature points. An example of using Hessian value to extract feature points is shown in Fig. 1.3. The grayscale of each pixel in Fig. 1.3(b) represents the absolute Hessian value of the corresponding pixel in Fig. 1.3(a).



(a) (b)

Fig. 1.3 Image of Hessian value of a frame.

Except for speeding up the iterative minimization, feature points can be carefully selected to avoid pixels of moving objects from affecting the parameter estimation. If segmentation masks are provided by the user, it is simple to exclude the pixels of moving objects according to the information in the masks. However, if the segmentation masks are not provided, a sprite generator should detects pixels of moving objects and excludes them automatically.

1.2.1.4 Feature points and minimization

Each feature point (x,y) in the current frame and its corresponding point (x',y') in the reference image form a feature point pair, which are used to find the initial guess for camera motion. The corresponding point is defined to be the motion-estimated point of the feature point, i.e., $(x',y')=(x+dx,y+dy)$, where (dx,dy) is the motion vector.

As mentioned previously, the perspective transformation expressed in Eq. (1.2) has eight parameters $m_1,m_2,\dots m_8$. By substituting each pair of (x,y) and (x',y') into Eq. (1.2) respectively, two equations will be built. Thus, four feature point pairs are sufficient to solve the eight parameters. However, in practical, the corresponding points found by motion estimation are not precise enough to provide a correct solution. Instead of using four feature point pairs, all feature point pairs found are applied to form an over-determined set of equations. A least

Mean-Square-Error minimization method is employed here. The error function required in the gradient descent method, is slightly different from Eq. (1.4). Only the errors of the feature points are counted in the error function, that is,

$$E(I, I', T_P) = \sum_{(x,y)\in \text{feature\_points}} \left(I(x,y) - I'(T_P(x,y))\right)^2 . \qquad (1.6)$$

The gradient descent method is applied in the estimation of the local parameters and the global parameters. While estimating the local parameters, the reference image is defined as the previous frame. And the reference image is defined as the current sprite in the case of estimating the global parameters.

1.2.2 Warping

The current frame is warped toward the sprite coordinate using the camera model selected and the parameters estimated in the global motion estimation. Let *I* be the current frame, the warped frame I_W can be found by geometric transforming the current frame as

$$I_W(x,y) = I_F(T_P(x,y)) \qquad (1.7)$$

where *T* is the transformation and *P* is the estimated global motion parameters. Since the transformed coordinates are not integers, bilinear interpolation [31] is applied while generating the warped frame. Since the frame is warped toward the sprite coordinate, the warped frame can be blended directly into the current sprite.

1.2.3 Blending

The warped frame is blended into the current sprite. The simplest blending method is the averaging blending. Let X, S_C and S_U are the intensities of the current frame, the current sprite and the updated sprite respectively. The averaging blending can be expressed as:

$$S_U = \frac{N_C * S_C + X}{N_C + 1} \tag{1.8}$$

where N_C is the number of pixels blended in the current sprite.

Since the averaging blending simply blends every pixel into the sprite, pixels of moving objects must be excluded from attending the averaging blending. Similar to the selection of feature points, excluding these object pixels is easy if segmentation masks are provided. If these object pixels are not fully excluded and some object pixels are blended into the sprite, the blended object pixels will leave some shadows in the generated sprite.

1.3 Existing Single Sprite Generators

Many sprite generators [18-20, 32-35] have been proposed. Most of them are based on a framework provided by MPEG-4 VM [14]. Among these existing sprite generators, the generator proposed by Smolić *et al.* [18] is a milestone. They proposed a hierarchal long-term global motion estimator and a reliability-based blending strategy to generate a sprite. The reliability-based blending tries to prevent the segmentation faults in segmentation masks from affecting the generated sprite by introducing a warning zone between objects and background.

1.3.1 Smolić *et al.*'s reliability-based blending

The traditional sprite generation methods produce the sprite by warping and averaging all frames of the video sequence. However, pixels belonging to the foreground objects will also be blended into the generated sprite. Using segmentation masks can resolve this problem. However, automatically generated segmentation masks are always not perfect, and segmentation faults always exist. These segmentation faults makes some pixels of moving objects be blended into the sprite and makes the generated sprite blur.

Reliability-based blending is developed to recover the segmentation faults. A segmentation mask is split into reliable, unreliable and undefined regions to form a reliability mask. The object pixels in the segmentation mask are defined as undefined region. The background pixels near an object pixel or the boundary of the frame are defined as unreliable region. The rest of background pixels are defined as reliable region. Fig. 1.4 shows a segmentation mask and its reliability mask. The reliable, unreliable and undefined regions are colored black, gray and white in Fig. 1.4(b), respectively.

Pixels in the undefined region belong to the moving objects, and definitely must not be blended into a sprite. Pixels in the reliable region are the background pixels, which are safe to be blended into a sprite. Pixels in the unreliable region are special. They are denoted as background pixels in the segmentation mask. However, due to the possibility of segmentation faults, background pixels near an object pixel have a higher possibility of being wrongly classified. Thus these pixels, which belong to the unreliable region, should be treated carefully.

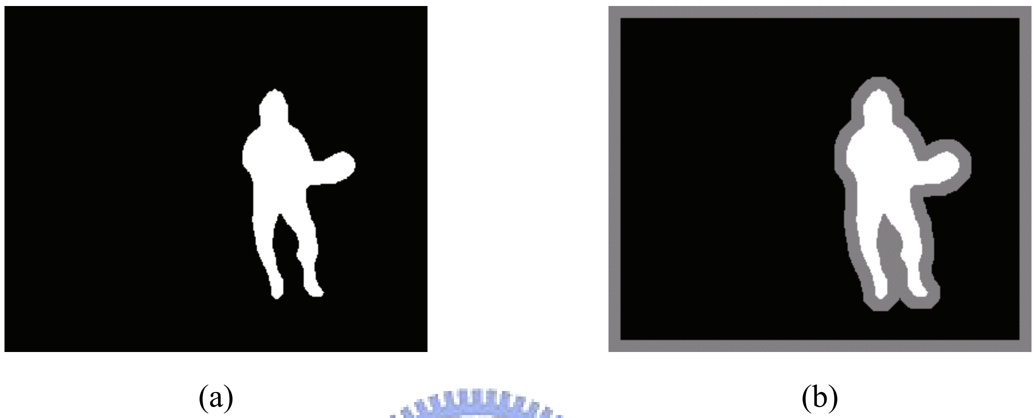


Fig. 1.4 Reliability mask used in the reliability-based blending.
(a) Segmentation mask. (b) Reliability mask derived from (a).

While blending a pixel into the current sprite, the reliable level is also recorded. The reliable level of the current pixel is compared to the reliable level of the current sprite. If the reliable level of the current pixel is lower than the sprite, the current pixel will not be blended into the sprite and will be discarded. If the reliable levels of both pixels are identical, the current pixel is blended into the current sprite by averaging blending. If the reliable level of the current pixel is higher than the sprite, the sprite pixel is discarded and is replaced by the current pixels. The discard and replace strategy ensures only pixels with highest reliable level are blended into the sprite.

1.3.2 Watanabe and Jinzenji's generator

Watanabe and Jinzenji [34] presented a sprite generator with two-passed blending and automatic foreground object extraction. In the first pass of sprite blending, a provisional sprite is constructed using the temporal median. Then the foreground objects are extracted automatically based on the provisional sprite. A difference image of the current frame from the provisional sprite is calculated, it is used to classify the pixels of the current frame into foreground and background ones. Then the current frame is divided into blocks, and each block is classified as either a foreground one or a background one according to the number of foreground pixels inside the block. The foreground blocks are excluded in the second pass of sprite blending. There are two disadvantages. One is that getting a perfect segmentation is impossible due to that a good threshold is needed in block classification and that block used as the classification unit will make segmentation roughly. The other is that the additional pass doubles the blending time.

1.3.3 Lu *et al.*'s generator

Lu *et al.* [19-21] used a more precise segmentation method proposed by Meier and Ngan [36] to obtain moving object masks. Based on the obtained masks, a modified reliability-based blending strategy inspired from Smolić *et al.* [18] work is developed to generate sprite. However, the qualities of generated sprites of these methods are still relying on the precision of segmentation masks.

1.4 Geometric Distortion and Multiple sprites

The performance of a sprite generator is also limited to the perspective motion model applied in the global motion estimation. The perspective model projects each frame of a video sequence into a planar reference coordinate system, which is usually the coordinate system of the first frame. Theoretically, the perspective model employed in MPEG-4 VM can cover 180 degrees 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.

Fig. 1.5 illustrates the effect of geometric distortion. 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. 1.5. 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 found between frame B and transformed frame B. If camera rotation continues, from Fig. 1.5, we can see that frame C can not be projected to the reference system.

Fig. 1.6 shows a geometric distorted frame. Frame A which shown in Fig. 1.6(a) is employed as reference frame of the sprite coordinate system. Fig. 1.6(b) shows a frame B, and the transformed frame B relative to frame A is shown in Fig. 1.6(c). On can see that the transformed frame B is geometric distorted and its size is larger than the original frames. The distortion causes the 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. This useless large resolution increases the memory usage and storage space required to hold the sprite.

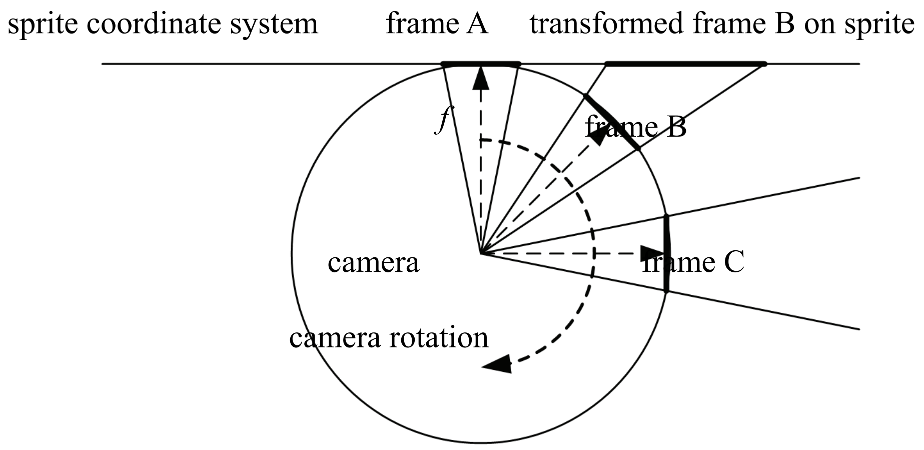
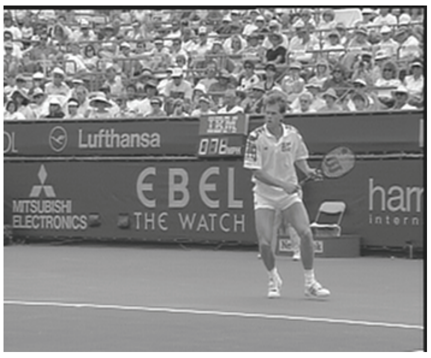
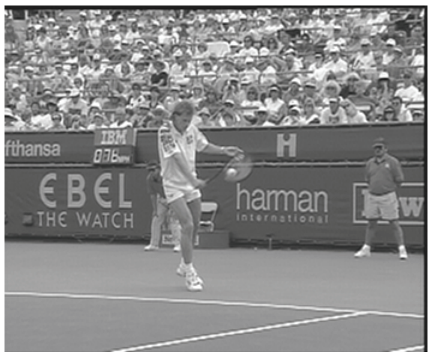


Fig. 1.5 Sprite coordinate system and geometric distortions of transformed frames.

In order to overcome the resolution-increasing effect, Massey and Bender proposed a method using the middle frame of a video sequence as the reference frame [37]. The generated sprite will be much symmetric and the boundary area of the generated 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.


(a)


(b)

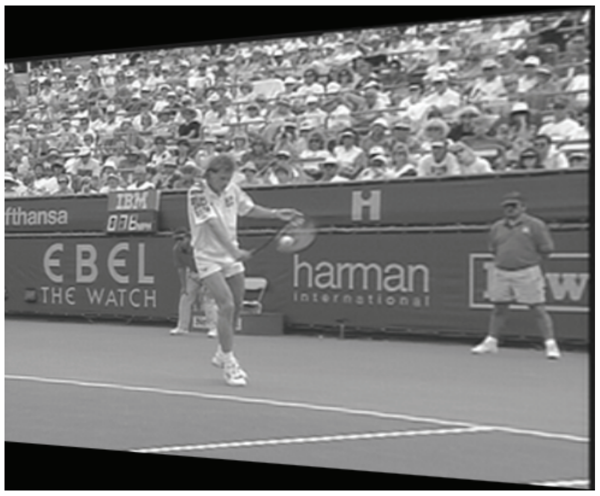
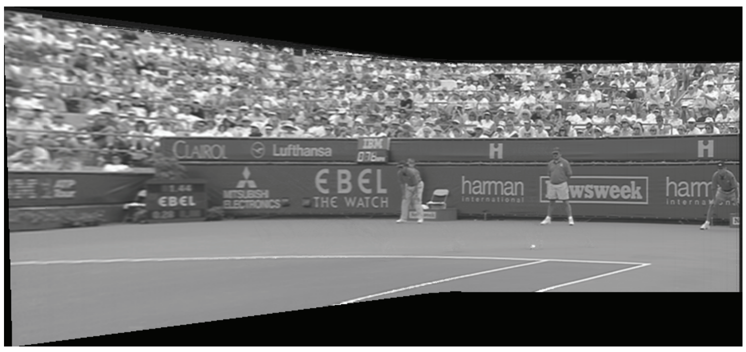
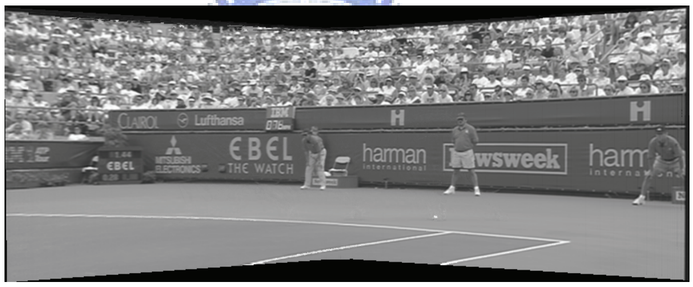

(c)

Fig. 1.6 Demonstration of geometric distorted frame due to camera rotation. (a) Original frame A. (b) Original frame B. (c) Transformed frame B.

Fig. 1.7 shows two sprites generated from the same portion of sequence 'stefan'. Fig. 1.7(a) uses the rightmost frame as the reference frame, and Fig. 1.7(b) uses the middle frame as the reference frame. One can see that the geometric distortion gets worse in the left part of sprite in Fig. 1.7(a) since the left part is away to the reference frame. The geometric distortion in Fig. 1.7(b) is evenly spread to both left and right part of the sprite. It is obviously that the sprite using the middle frame as the reference frame is smaller than that using the rightmost frame.



(a)



(b)

Fig. 1.7 Sprites with different reference frames.
(a) Rightmost frame as reference frame. (b) Middle frame as reference frame.

A technique using multiple sprites was proposed by Farin *et al*. [38] to solve the problem. In their works, 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.

Fig. 1.8 shows the geometric distortions using two sprites. In contrast to the geometric distortion using only one sprite shown in Fig. 1.5, 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 degrees of camera view can be covered if more sprites are used. Note that any single sprite must not cover 90 degree or more of camera rotation over any direction to prevent an effect called ‘degeneration’ [39].

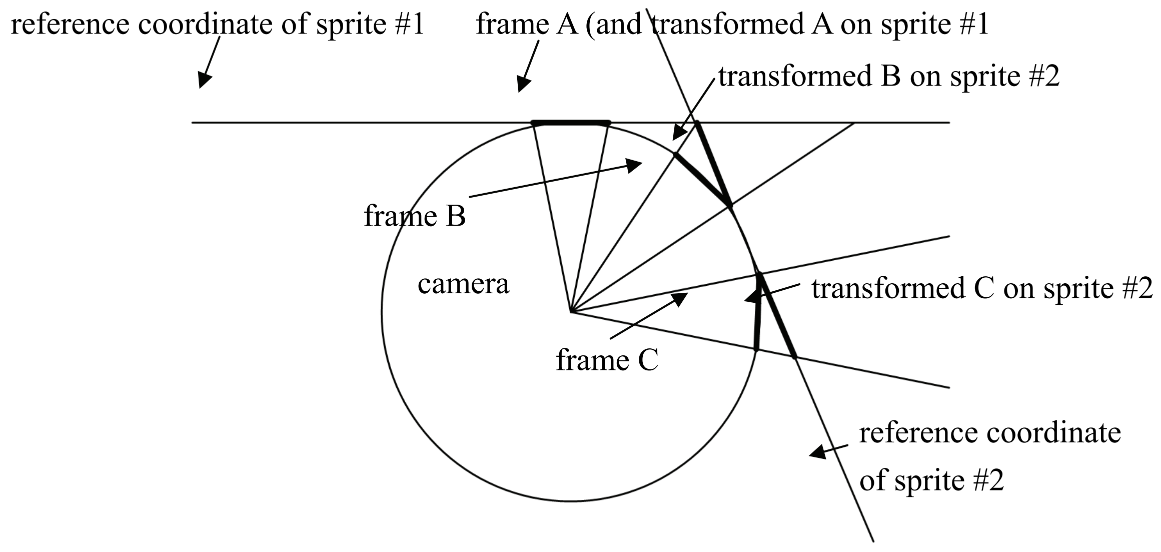

reference coordinate of sprite #1
frame A (and transformed A on sprite #1
transformed B on sprite #2
frame B
camera
transformed C on sprite #2
frame C
reference coordinate
of sprite #2

Fig. 1.8 Geometric distortions using two sprites.

Farin *et al.* [38] 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. However, the Farin *et al.*’s method uses exhaustive searches to find the partition points of sub-sequences and the reference frame of each sub-sequence. The exhaustive searches make the method very time-consuming.

Some sprite generators are proposed to employ the multiple sprite technique. Chen *et. al.* intergrates a frame skipping techniques and the multiple sprite to speed up the overall computation time of sprite generation [40-41]. Kunter *et al.* proposed a experimental framework [42] to employ multiple sprite into H.264/AVC. None of them is discussing to speed up the multiple sprite partition.

1.5 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 sub-sequences and finds the optimal reference frames of every possible sub-sequence. 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.

1.5.1 Coding Costs Computing and Reference Frames Finding

A coding cost matrix holding the coding costs of all possible combinations of sub-sequences are computed in this step. For a sub-sequence beginning at frame *i* and ending at frame *k*, a sprite $S_{i;k}^{r}$ can be generated for a reference frame *r* with $i \le r \le k$. The coding cost for coding $S_{i;k}^{r}$ is denoted as $\left\|S_{i;k}^{r}\right\|$. $\left\|S_{i;k}^{r}\right\|$ of all possible combination 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 $\left\|S_{i;k}^{r}\right\|$, the optimal reference frame $r_{i;k}^{*}$ of a sub-sequence beginning at frame *i* and ending at frame *k* can be selected by

$$r_{i;k}^{*} = \arg\min_{r}\left\|S_{i;k}^{r}\right\|. \qquad (1.9)$$

The minimized coding cost of this sub-sequence, $\left\|S_{i;k}\right\| = \min_{r}\left\|S_{i;k}^{r}\right\|$, is also kept.

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

1.5.2 Optimal Partitioning

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

$$P = \{(1, p_1 - 1), (p_1, p_2 - 1), (p_2, p_3 - 1), \ldots, (p_{n-1}, N)\}, \qquad (1.10)$$

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

$$P^* = \arg\min_P \sum_{(i,k)\in P} \left\| S_{i;k} \right\|. \tag{1.11}$$

The minimization problem is solved efficiently. If the video contains only the first frame, there is only 1 possible partition $P = \{(1,1)\}$ and the optimal sprite coding cost of the video is $\left\| S_{1;1} \right\|$ 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

$$\begin{aligned} c_k &= \min_{i\in[1,k]} \left\{ c_{i-1} + \left\| S_{i;k} \right\| \right\} \\ p_k &= \arg\min_{i\in[1,k]} \left\{ c_{i-1} + \left\| S_{i;k} \right\| \right\}, \end{aligned} \tag{1.12}$$

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 sub-sequences $(1, p_N - 1)$ and (p_N, N). The former sub-sequence is further divided at $p_{(p_N - 1)}$, which is the best partition point of sub-sequence $(1, p_N - 1)$, and so on.

1.6 The Main Problems and Current Status

The main problems that this dissertation tries to address are based on the above-mentioned segmentation masks related problems and the time complexity problem of multiple sprites generating. These will be described in the following.

In this dissertation, we will propose methods to deal with the problems of single and multiple sprite generation. For the traditional single sprite generation, a new generation method without using segmentation masks will be proposed to avoid the segmentation faults from affecting the generated sprite. For the multiple sprite generation, a fast generation method will be proposed to increase the search speed of sub-sequences and selecting of reference frames.

1.6.1 Sprite Generation without Segmentation Masks

The framework of MPEG-4 VM shown in Fig. 1.1 requires segmentation masks while generating sprites. Segmentation masks are binary maps indicating whether a pixel belongs to moving objects or not. In order to avoid sprite being blurred, pixels of moving objects must be excluded from being blended into the sprite. If the segmentation is perfect, the averaging blending provided in MPEG-4 can achieve excellent quality; otherwise, the generated sprite will be blurred around moving object boundary due to that some pixels of moving objects are considered as background.

The segmentation masks can be manually provided before the sprite generation or generating automatically during the sprite generation process. No matter how the segmentation masks are provided, they must be generated automatically to makes the sprite generation practical. However, it is almost impossible to generate segmentation masks perfectly automatically. Thus segmentation faults always exist, and the generated sprite always looks blurry.

On the purpose of reducing the blur caused by segmentation faults, precise segmentation methods [36] are employed [34] and a new blending strategy denoted as reliability-based blending [18] is developed. The reliability-based blending is adopted by several sprite generators [18-21]. In the reliability-based blending strategy, a frame is divided into reliable, unreliable, and undefined regions according to the segmented masks. Pixels denoted as objects in the segmented masks are classified as undefined pixels, and pixels near mask borders or frame borders (within a given distance) are classified as unreliable ones. The rest of pixels are classified as reliable ones. The reliable and unreliable pixels are average-blended separately, and the blended pixels with the highest reliability are chosen into the sprite. The undefined pixels do not contribute to the sprite blending. The given distance from the mask border must be large enough to cover all segmentation faults, or the generated sprite will have ghost-like shadows in some places. However, it is hard to decide the distance automatically. Thus ghost-like shadows still can be found in the generated sprite. In order to overcome this problem, in this dissertation, we try to develop a novel blending strategy without using segmentation masks.

Segmentation masks are not only used in the blending process, but also in the global motion estimation process. The global motion estimation uses segmentation masks to avoid objects affecting the accuracy of generated global motion parameters. For removing the usage of segmentation masks completely from the entire sprite generation process, the global motion estimation must also be modified.

1.6.2 Fast Multiple Sprites Generation

The perspective model employed in the global motion estimation process makes the transformed frames geometric distorted, as Fig. 1.6 shows. This distortion become more seriously as the camera view of frame 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.

Massey and Bender proposed to use the middle frame of a video sequence as the reference frame to overcome the resolution-increasing effect. The generated sprite will be much symmetric and the boundary area of the generated sprite becomes much smaller if the background of the frames in the video sequence pans toward only one direction. However, this method only slows down the increasing effect of the sprite size, but the range of view angle is not extended.

Using multiple sprites can solve this problem. 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 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.

Apart from the geometric distortion, the effect of camera zoom-in and zoom-out will highly affect the generated sprite and the reconstructed frames [38]. Fig. 1.9 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.

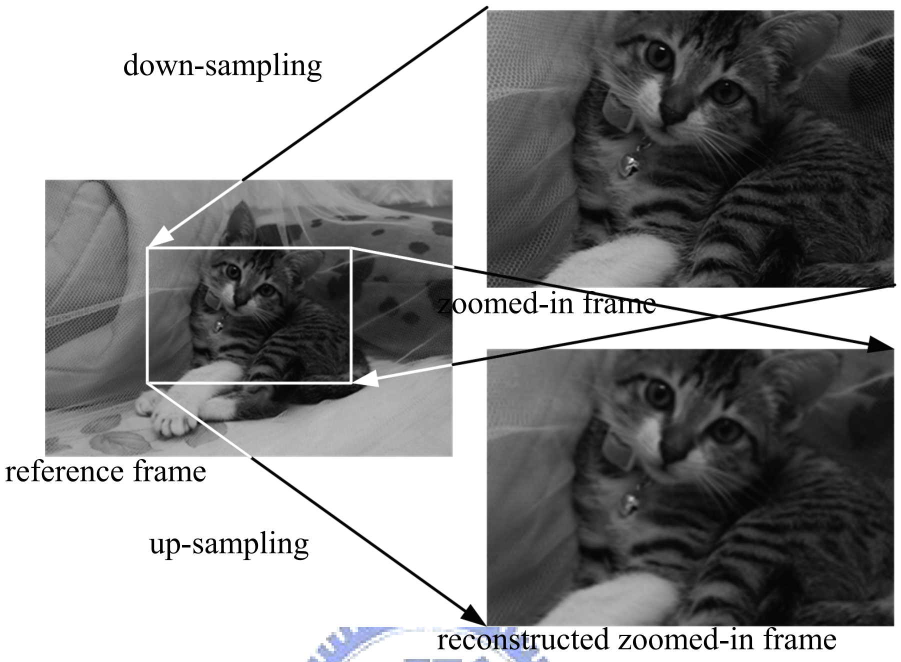


Fig. 1.9 Effect of camera zoom-in with the details of the reconstructed frame lost.

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. 1.10 shows, the zoomed-out frame has to be up-sampled before blending into the sprite. This will make the blended sprite blurred. Furthermore, the up-sampled frame will occupy a very large area in the sprite. This causes the area of generated sprite being expanded rapidly.

In this dissertation, we try to develop a fast multiple sprites partition method and fast reference frame selection method with acceptable total sprite areas.

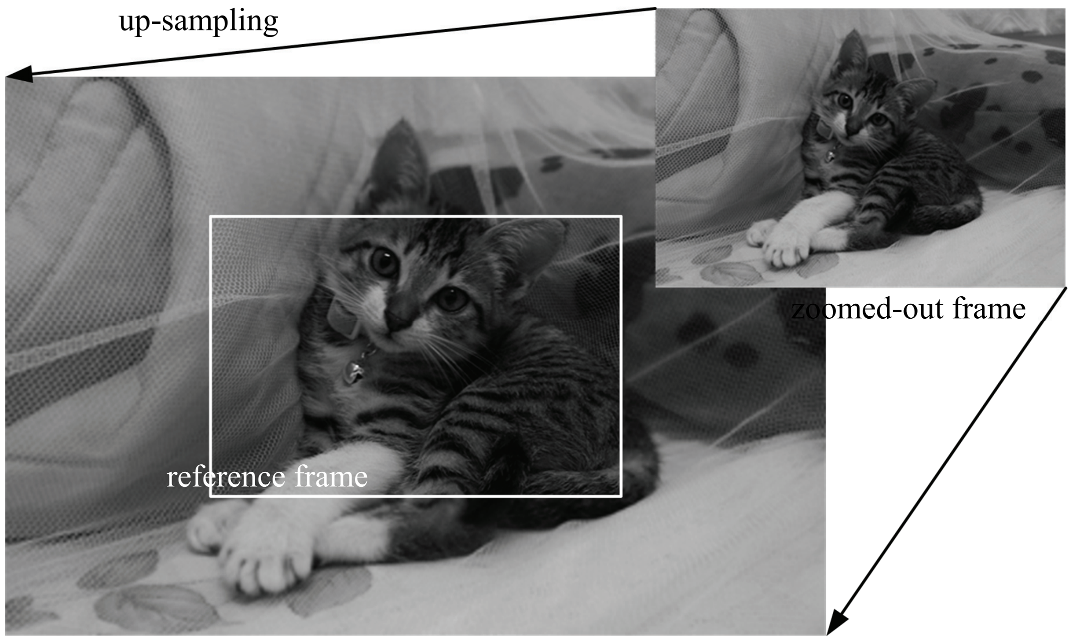


Fig. 1.10 Effect of camera zoom-out with the sprite blurred.

1.7 Synopsis of the Dissertation

The rest of the dissertation is organized as follows. Chapter 2 describes the proposed sprite generation method with automatically generated segmentation masks. In Chapter 3, the proposed sprite generation method without using segmentation masks will be introduced. The fast multiple sprite partition and reference frame selection methods are proposed in Chapter 4. Some conclusions and future research directions are drawn in Chapter 5.

CHAPTER 2
AUTOMATIC GENERATION OF SEGMENTATION MASKS FOR SPRITE GENERATION

In this chapter, we will propose a sprite generator with an automatic generation of segmentation masks. The generation process contains two passes. The first pass generates a coarse sprite by conventional averaging blending method. Then we generate segmentation masks of every frame automatically from the coarse sprite. In the second pass of sprite generation, the final sprite is generated with the generated segmentation masks to reduce the effect of moving objects. The details of the proposed generator are described as follows.

2.1 Proposed Two-Pass Sprite Generation

As Fig. 1.1 shows, MPEG-4's sprite generation framework requires auxiliary segmentation masks to reduce the effect of moving objects. Segmentation masks are used in global motion estimation to raise the estimation accuracy. These masks can also be used to avoid moving objects attending the sprite blending. It is impractical to build these masks manually. Thus an automatic generation of segmentation masks is necessary. Since the sprite is a merged background in the video sequence, it can be used as a reference background for moving object detection.

2.1.1 Object blurring effect of averaging blending

The averaging blending used in sprite generation has a blurring effect to every pixel in the sprite. If there does not have segmentation masks, the averaging blending blends pixels of moving objects and background together. In case of enough frames are blended, the moving objects will be blurred and only shadows of objects are left in a generated sprite. Fig. 2.1 demonstrates the blurring of moving objects. Figs. 2.1(a) and (b) are an original frame and its reconstructed background from generated sprite, respectively. One can see that the moving player is blended into the sprite, and leave white shadows in the reconstructed background. Although the quality of reconstructed background is degraded by these shadows, the backgrounds in the original frame are still visible in the reconstructed background.

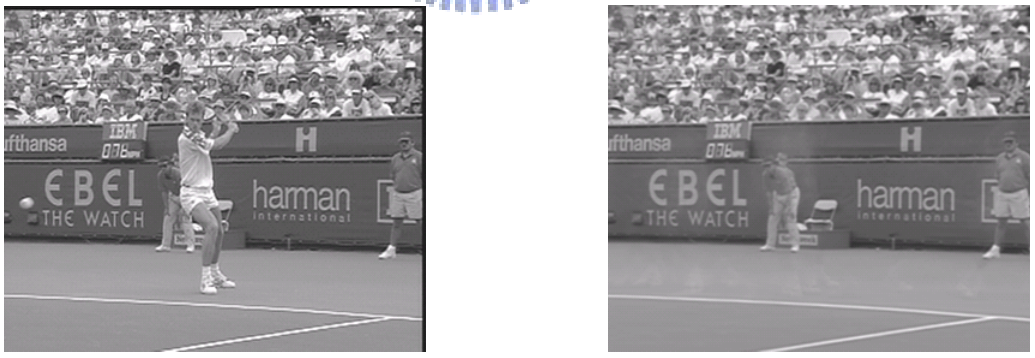


Fig. 2.1 Object blurring effect of averaging blending.
(a) Original frame. (b) Reconstructed background of (a).

2.1.2 Frame segmentation in sprite generation

Since reconstructed backgrounds in a sprite generated by averaging blending without segmentation masks are only blurred by moving objects, the reconstructed backgrounds can be used as reference backgrounds to detect moving objects. A two-pass sprite generation with automatic segmentation masks generation is proposed and shown in Fig. 2.2.

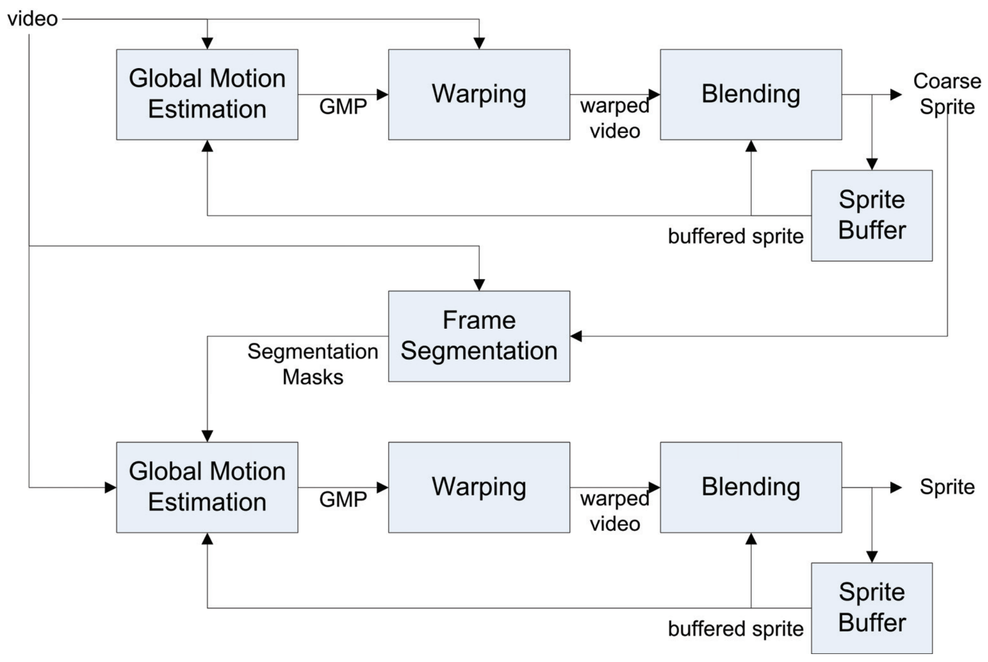


Fig. 2.2 The proposed two-pass sprite generator.

In the first pass of the proposed generator, a coarse sprite is generated first by the MPEG-4's sprite generator without segmentation masks. The coarse sprite will contain shadows of moving objects definitely. Then the reconstructed backgrounds of the coarse sprite are employed as reference backgrounds to detect the moving objects in the video and generate segmentation masks automatically. Finally the sprite is re-generated in the second pass by the MPEG-4's sprite generator with the generated segmentation masks.

2.2 The First Pass of Sprite Generation

The first pass of sprite generation is almost identical to the MPEG-4's framework. In the global motion estimation, some feature points are extracted by selecting the pixels with larger Hessian value. Then global motion parameters are estimated by a least mean-square-error minimization method, the Levenberg-Marquardt algorithm. An averaging blending method is employed as the blending method. No segmentation masks are applied in this pass of sprite generation. Thus moving objects will be blended and the generated coarse sprite will be blurred. The coarse sprite will be used to extract the segmentation masks automatically.

2.3 Automatic Generation of Segmentation Masks

Despite of the blurred areas, the reconstructed backgrounds still carry most of background information. By subtracting the original frame by the reconstructed background, we can get an image of the moving objects. In order to remove the effect of peak noise, the block difference is applied instead of the pixel difference.

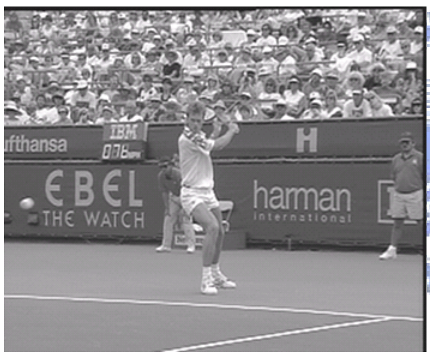
The pixel difference D is defined as the magnitude of the difference between the original frame I and the reconstructed background R, i.e. $D = |I - R|$. A threshold t_1 is set to find out the candidates of object pixels. Pixels with D value larger than t_1 are considered as candidates. For each candidate, a 5×5 block B centered on the candidate is taken. The block difference D_B, is defined as

$$D_B = \sum_{(i,j)\in B} D(i,j)\,. \tag{2.1}$$

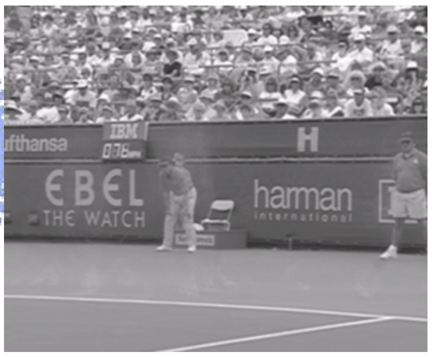
The candidate with block difference larger than a preset threshold t_2 is considered as an object pixel. The two-stage thresholding technique computes the block differences only for those pixels with higher possibility to be objects. It reduces the complexity of computing block difference for each pixel.

There are two problems while extracting the object pixels. First, the object regions are often ill-shaped with holes. Second, there are some small-sized regions which are misclassified as objects. These problems can be solved using morphological processing and region selecting. Let O be a binary image representing the results of thresholding in the previous step. Pixels judged as objects will be set to one and others will be set to zero. Two binary images called seed and base images are computed. The seed image is produced by applying morphological erosion to O using a disk shaped structure element of radius 2, and the base image is produced by applying morphological dilation to O using the same shaped structure element of radius 5. The region selecting is applied on the base image. An object region is selected if any of its pixels have a value of one in the seed image. The segmentation mask is defined as the union of all regions selected.

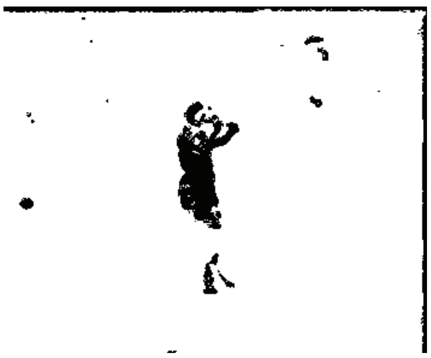
Fig. 2.3 gives an example of generating a segmentation mask. The original frame and the reconstructed background of the frame are shown in Figs. 2.3(a) and (b) respectively. By subtracting the original frame by the reconstructed background and performing the two-stage thresholding, the image of the extracted object pixels is shown in Fig. 2.3(c). The seed image shown in Fig. 2.3(d) and the base image shown in Fig. 2.3(e) are generated by applying the morphological processing to Fig. 2.3(c). Finally, the segmentation mask is produced by region selecting and shown in Fig. 2.3(f). The object regions are colored black.



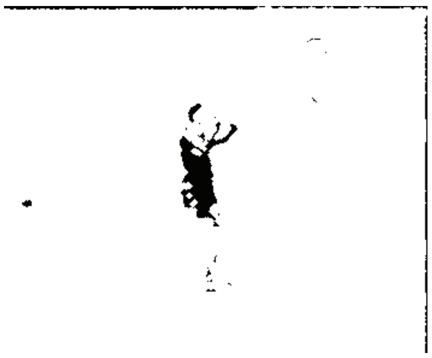
(a)



(b)

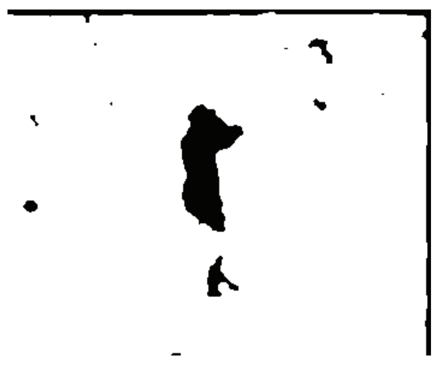


(c)

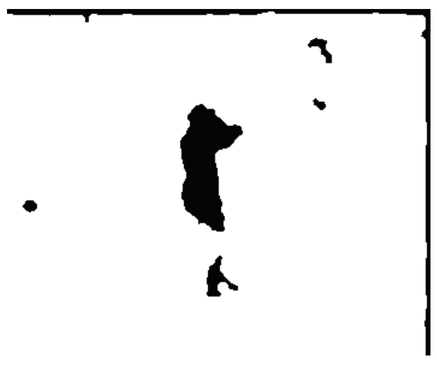


(d)

Fig. 2.3 (*Continued*) The generation of a segmentation mask. (a) The original image. (b) The reconstructed background. (c) The object pixels extracted by using two-stage thresholding. (d) The seed image. (e) The base image. (f) The generated segmentation mask.



(e)



(f)

Fig. 2.3 The generation of a segmentation mask. (a) The original image. (b) The reconstructed background. (c) The object pixels extracted by using two-stage thresholding. (d) The seed image. (e) The base image. (f) The generated segmentation mask.

Most part of the moving objects was extracted correctly, except two unclassified parts: the upper part of bat and the player's legs. The upper part of bat is nearly transparent hence the background is visible through the bat; the legs of the player have similar intensities to the background. Thus, both misclassified parts do not affect the blending result. Moreover, the top and right borders are also classified as object; this will eliminate the black line shadows in the generated sprite. Note that the tennis ball is also classified as an object. These generated segmentation masks will be employed in the second pass of sprite generation.

2.4 The Second Pass of Sprite Generation

The generated segmentation masks are employed in the second pass of sprite generation. The second pass sprite generation is similar to the first pass with some modifications. The automatically generated segmentation masks are employed in the global motion estimation and the blending process. These modifications remove the effect caused by considering the object pixels as background, and increase the fidelity of the generated sprite.

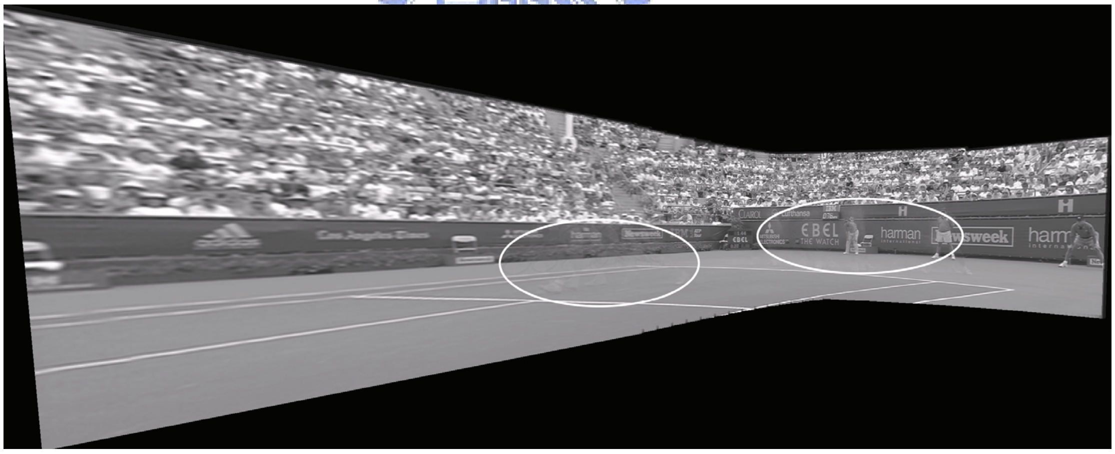
In the global motion estimation, the generated segmentation masks are employed as a classification of object pixels. All feature points are checked with the masks. Feature points which are classified as moving objects in the masks are removed from the feature points. Then the global motion parameters are also estimated by the Levenberg-Marquardt algorithm. The accuracy of estimated parameters should be increased since the effect of object pixels is reduced.

The sprite is then blended using the newly estimated parameters. Since generated segmentation masks are available in the second pass, the reliability-based blending is adopted instead of the averaging blending employed in the first pass. The reliability-based blending prevents some of moving objects that not segmented correctly from being blended into the final sprite. The generated sprite in the second pass is outputted as the final sprite.

2.5 Experimental Results

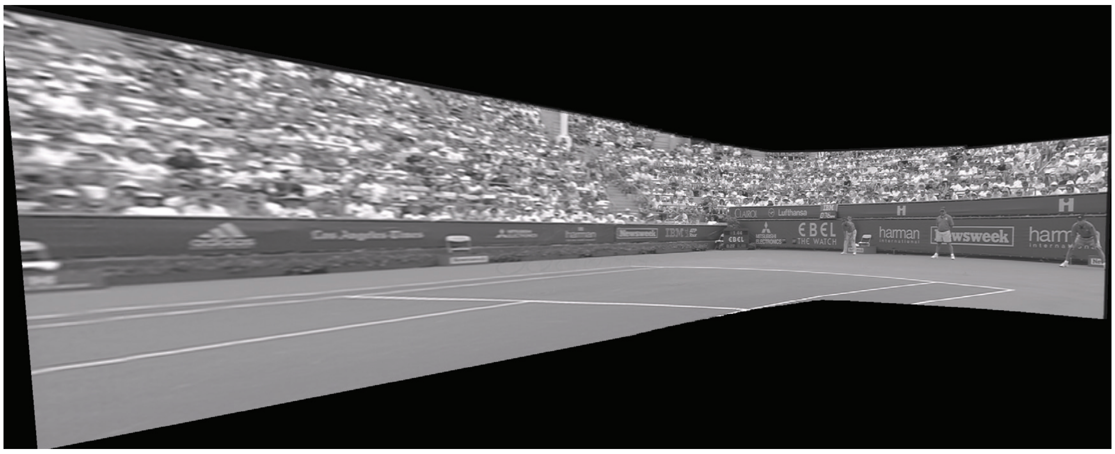
Fig. 2.4 shows the generated sprite of the video sequence ‘stefan’ by different methods. Fig. 2.4(a) is generated by the MPEG-4’s method without using segmentation masks, that is also the coarse sprite generated from the first pass of sprite generation. The masks used to generate Fig. 2.4(b) are obtained automatically by the proposed segmentation schema.

Fig. 2.5 shows one of the reconstructed frames by different methods respectively. Like we stated before, the sprite generated without using masks contains shadows, which are circled in Fig. 2.4(a), caused by wrongly blending the player into sprite. These shadows are successfully removed in the sprite generated using the masks generated automatically by our method. Manually segmented masks are employed in Fig. 2.4(c) and Fig. 2.5(c) for comparisons. Both sprites generated using automatically or manually segmented masks are perceptually the same by human eyes.

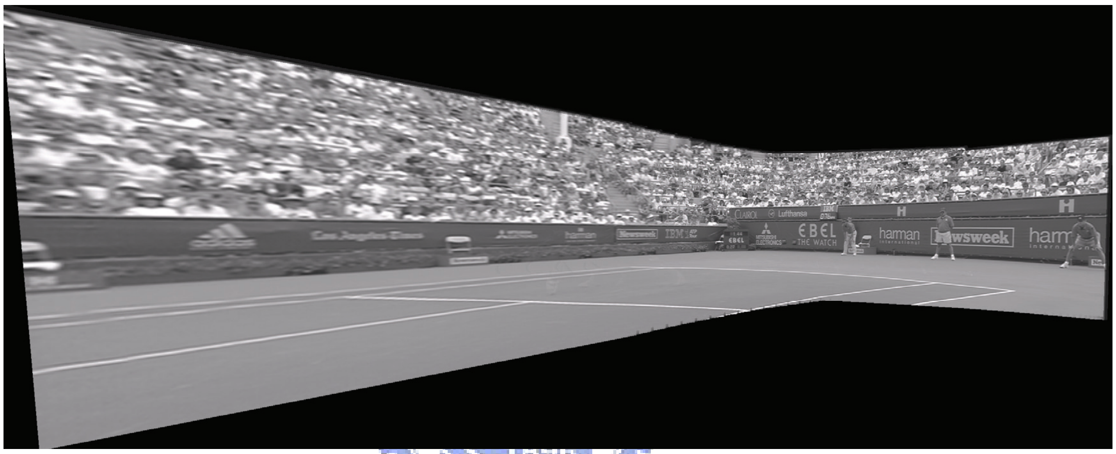


(a)

Fig. 2.4 (*Continued*) Generated sprites of different methods. (a) The first pass.
(b) Two pass generation with automatic generated segmentation masks.
(c) MPEG-4 with manually segmented masks.

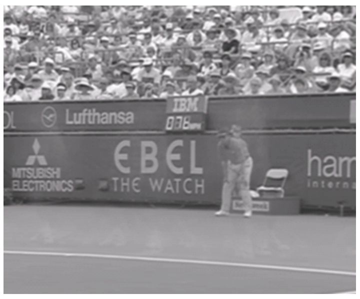


(b)

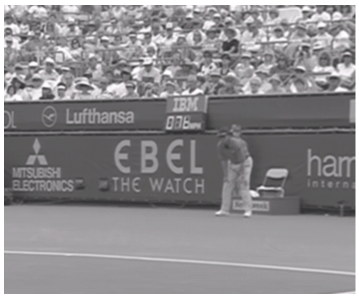


(c)

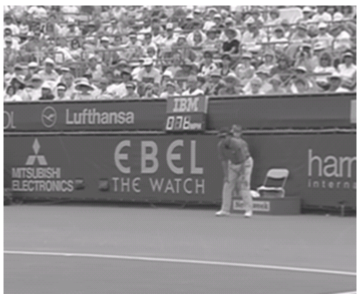
Fig. 2.4 Generated sprites of different methods. (a) The first pass. (b) Two pass generation with automatic generated segmentation masks. (c) MPEG-4 with manually segmented masks.



(a)



(b)



(c)

Fig. 2.5 Reconstructed frames of different methods. (a) The first pass. (b) Two pass generation with automatic generated segmentation masks. (c) MPEG-4 with manually segmented masks.

CHAPTER 3
A NEW APPROACH FOR SPRITE GENERATION WITHOUT SEGMENTATION MASKS

In this chapter, we will propose a sprite generator without using segmentation masks. It consists of a modified feature point selection method and a novel intelligent blending method.

3.1 Problems of Segmentation Masks

To avoid sprite being blurred, pixels of moving objects must be excluded from being blended into the sprite. If the segmentation is perfect, the averaging blending provided in MPEG-4 can achieve excellent quality. Otherwise, the generated sprite will be blurred around moving object boundary due to that some pixels of moving objects are considered as background. However, a perfect segmentation is impossible, conventional sprite generation methods often use a reliability-based blending concept provided in [18] to solve this problem. In the reliability-based blending strategy, a frame is divided into reliable, unreliable, and undefined regions according to the segmented masks. Pixels denoted as objects in the segmented masks are classified as undefined pixels, and pixels near mask borders or frame borders (within a given distance) are classified as unreliable ones. The rest of pixels are classified as reliable ones. The reliable and unreliable pixels are average-blended separately, and the blended pixels with the highest reliability are chosen into the sprite. The undefined pixels do not contribute to the sprite blending. The given distance from the mask border must be large enough to cover all segmentation faults, or the generated sprite will have ghost-like shadows in some places. However, it is hard to decide the distance automatically. In this dissertation, we will provide a sprite generator to avoid above-mentioned problems.

The proposed method is based on the MPEG-4's framework shown in Fig. 1.1, but the demand of segmentation masks is removed. A balanced feature point extractor with object point removing is proposed. With the proposed feature points, the precision of estimated global motion parameters are increased significantly. A new blending strategy that does not need segmentation masks is also proposed. The moving objects are excluded from blending by a counting schema. The proposed method provides higher visual quality of the generated sprite than those existing methods, and the average PSNR of reconstructed backgrounds is increased slightly.

3.2 The Proposed Sprite Generator

In the proposed sprite generator, a two-stage GME is provided with a novel feature point extraction method to get accurate global motion parameters. With the estimated parameters, each input frame is warped. An intelligent blending strategy is then presented to blend the warped frame to form a sprite. The details of the proposed generator are described as follows.

3.2.1 Global motion estimation

The aim of global motion estimation is to obtain an accurate estimation of camera motion between the current frame and a reference image, e.g. the current sprite. In this dissertation, we take the perspective transformation to model camera motion as follows:

$$\begin{cases} x' = \dfrac{m_1 x + m_2 y + m_3}{m_7 x + m_8 y + 1} \\ y' = \dfrac{m_4 x + m_5 y + m_6}{m_7 x + m_8 y + 1} \end{cases}, \tag{3.1}$$

where (x,y) and (x',y') denote the coordinates of a pixel before and after the camera motion respectively. $m_1, m_2, \ldots m_8$ are the transformation parameters referred as global motion parameters.

The global motion parameter is estimated by the two-stage GME schema described in Section 1.2.1. The minimization problem described in Section 1.2.1.2 is solved by the Levenberg-Marquardt algorithm [15]. In order to increase the estimation speed, only selected feature points are attending the minimization. Pixels of moving objects should be avoided to be selected into the feature points. Since we want to propose a sprite generator that do not need segmentation masks, a novel feature point selection method that excludes the pixels of moving objects must be developed.

3.2.2 Feature point extraction

The iterative minimization of the gradient descent method is time consuming. To reduce the time complexity, only some selected feature points in the current frame are employed while computing the registration error. Like other sprite generation methods, our method selects feature points according to their Hessian values defined by

$$H(x,y)=\left(\frac{d^2I(x,y)}{dx^2}\cdot\frac{d^2I(x,y)}{dy^2}-\left(\frac{d^2I(x,y)}{dxdy}\right)^2\right). \qquad (3.2)$$

Those points with Hessian values being local maximum or minimum are considered as feature points. An example of using Hessian value to extract feature points is shown in Fig. 3.1. The grayscale of each pixel in Fig. 3.1(b) represents the absolute Hessian value of the corresponding pixel in Fig. 3.1(a).

Conventional methods choose pixels with largest absolute Hessian values as feature points. However, the distribution of pixels with large absolute Hessian values does not spread uniformly, as shown in Fig. 3.1(b). Fig. 3.1(c) shows the feature points extracted by these methods. The extracted feature points are concentrated in the half-upper of the image. This will degrade the accuracy of the estimated parameters because the registration will be focused only on the half-upper of the image. This degradation will become more serious, when the number of feature points is small. The sprite generated based on these feature points is shown in Fig. 3.1(a). Although the half-upper of the sprite looks well, the white lines in the half-bottom of the sprite are not fitted correctly such that they look blurred (see Fig. 3.1(c)); the reason is that no white line points are considered as feature points. To overcome this problem, the feature points must be selected uniformly. A balanced feature point extraction method is proposed and described as follows.

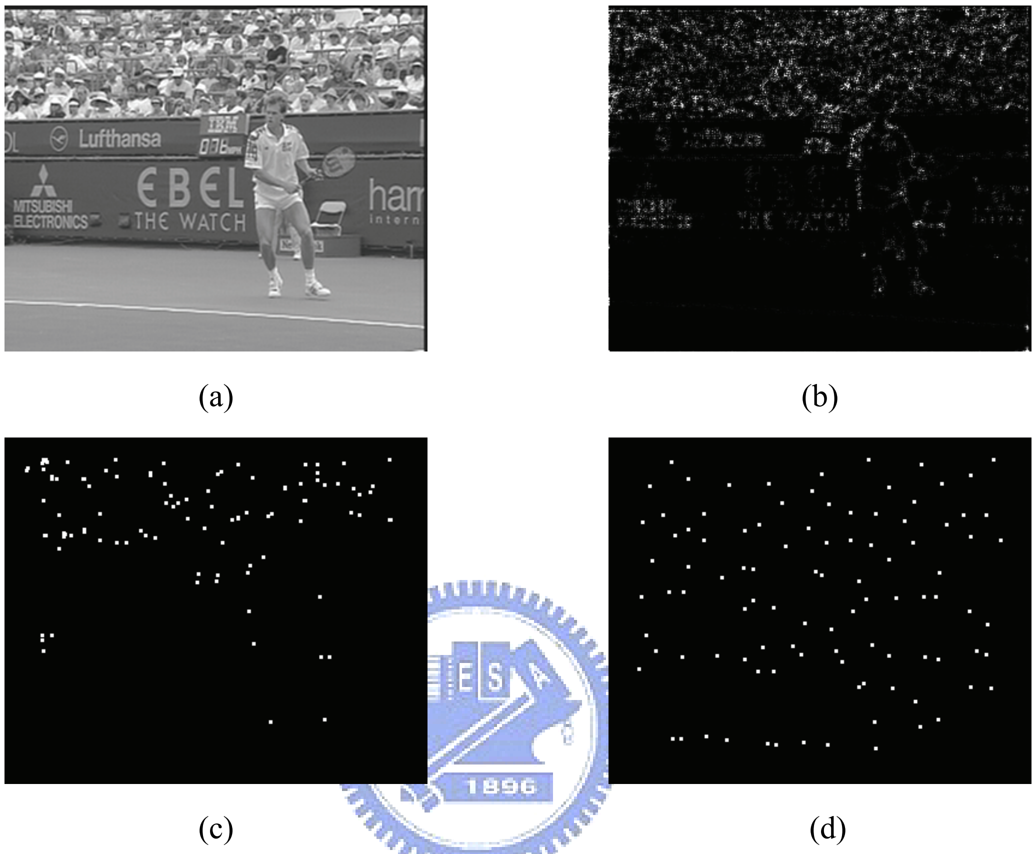


Fig. 3.1 Feature point extraction based on Hessian value. (a) Original image. (b) absolute Hessian values of (a). (c) Feature points extracted by conventional methods. (d) Feature points extracted by the proposed method.

For an image of width W and height H, its border area of width B is excluded first. The rest is divided into 256 non-overlapping blocks. For each block, the gray value variance is calculated to test its homogeneity. The block will be classified as a homogeneous one if its variance is smaller than a preset threshold T_V. The feature points are extracted uniformly in the non-homogeneous blocks to avoid the aperture problem. Suppose that we want to extract

N feature points from *K* non-homogenous blocks, *N*/*K* pixels with largest absolute Hessian values are chosen in each non-homogeneous block. Feature points extracted from Fig. 3.1(a) using the proposed method are shown in Fig. 3.1(d). In contrast to the result of using the conventional method (see Fig. 3.1(c)), the distribution of feature points using the proposed method is more balanced than MPEG-4's method. Several points on the white line in the half-bottom of the frame are extracted as feature points, this will make the white line registered well and will significantly improve the visual quality of the generated sprite. However, from Fig. 3.1(d), we also find some points on the player located; this will reduce the accuracy of estimated GMPs. As mentioned previously, we should avoid taking moving objects as feature points. Since the motions of moving objects usually differ from the motion of background, this provides us a clue to remove these outliers.

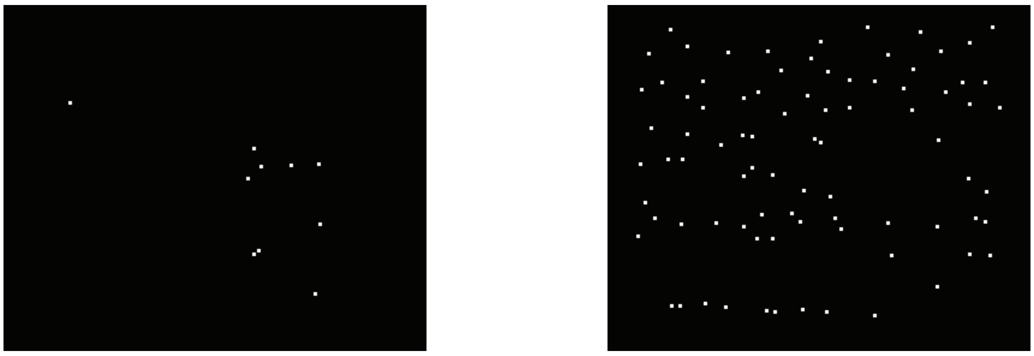
Traditional translation-based motion estimation is applied on each feature point to find the motion vector relative to the previous frame. In order to reduce the searching time, a global translation is found based on some selected feature points first, then a full search around the global translation for each feature point is preformed. A 17×17 block centered at each selected feature point is used to find the global translation. A full-searched motion estimation with a large search window (64×64) is proceeded on the 17×17 block. To raise up the searching speed, only 100 pixels with the largest absolute Hessian values among all feature points found previously are employed. The occurrences of the estimated 100 motion vectors are counted, and the motion vector with the highest occurrence is considered as the global translation. The motion vectors of all feature points are found by searching around the global translation with a smaller search window (17×17).

Let (*dx*,*dy*) be the motion vector estimated, the feature point is considered as an outlier if its mean-squared-error (MSE) between the original and the motion-estimated blocks is larger than a preset threshold T_O, i.e.,

$$\frac{1}{N_B}\sum_{x,y\in B}\left(I(x,y)-I'(x+dx,y+dy)\right)^2 > T_O\,, \qquad (3.3)$$

where B is the block centered at the feature point and N_B is the number of pixels in the block, $I(x,y)$ and $I'(x,y)$ are the current frame and the previous frame respectively. Since objects are assumed to have different motions from the background, their best motion vectors are usually not around the global translation (a roughly approximation of the background motion), and their MSEs are likely to be higher with inaccuracy motion vectors. They will be considered as outliers in Eq. (3.3).

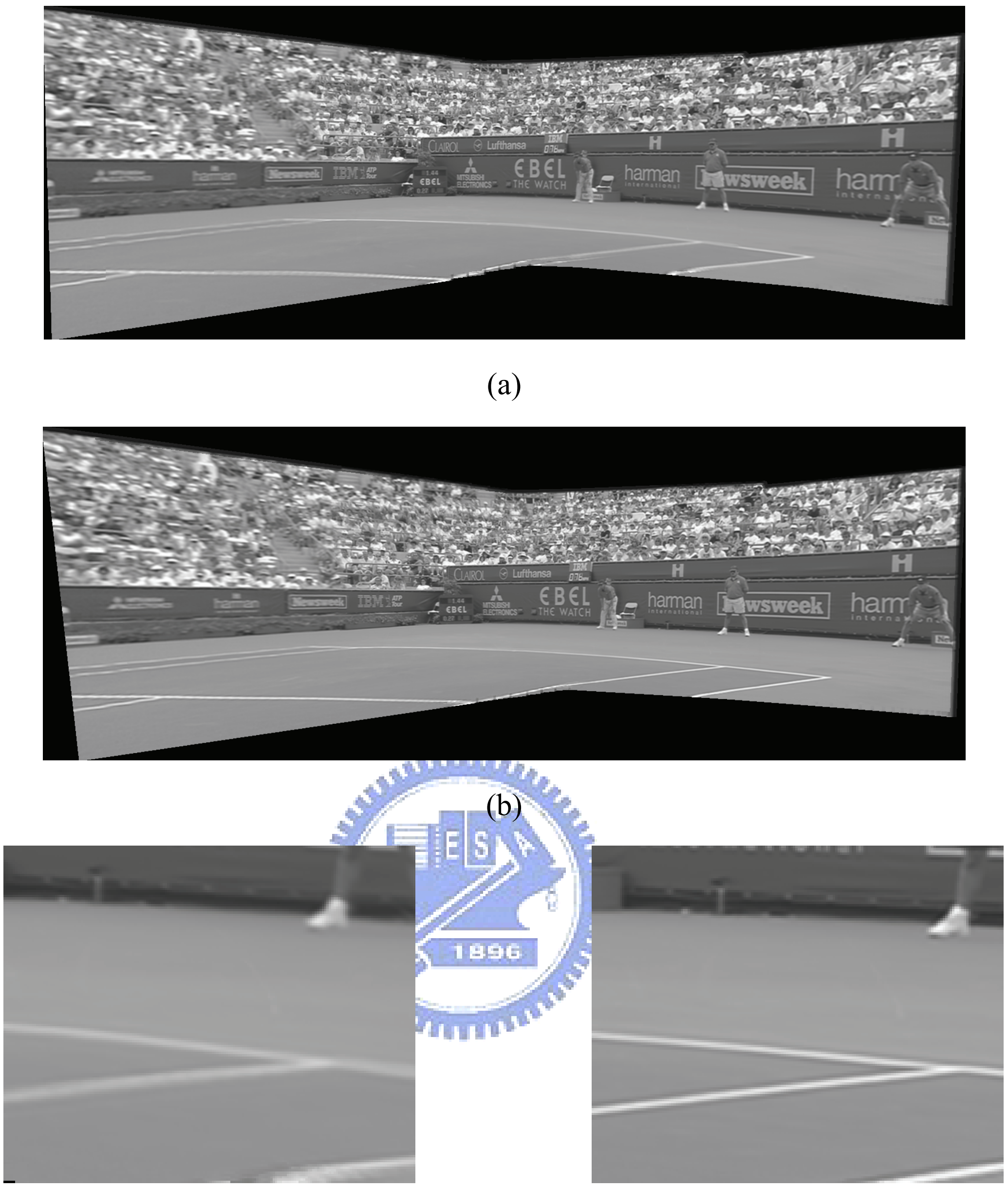
Fig. 3.2(a) illustrates the object pixels found from the feature points shown in Fig. 3.1(d). The feature points on the player are detected successfully. These object pixels are removed from the original feature points and the final feature points are shown in Fig. 3.2(b).



(a) (b)

Fig. 3.2 An example for outlier removing. (a) Detected object pixels in Fig. 3.1(d). (b) The feature points after removing outliers from Fig. 3.1(d).

Figs. 3.3(a) and (b) show the sprite generated using the conventional method and proposed balanced feature point extraction method, respectively. The same number of feature points is used in both methods. A close view of the white lines in Figs. 3.3(a) and (b) are shown in Figs. 3.3(c) and (d), respectively. From the figures, we can see that those white lines in the sprite generated by the proposed method are registered very well. While the same place in sprite generated by conventional method looks blurred. The average PSNR of the reconstructed backgrounds using the non-balanced and balanced feature points are both 26.25dB. Although the PSNR is the same, the proposed method achieves much better visual quality.


(a)

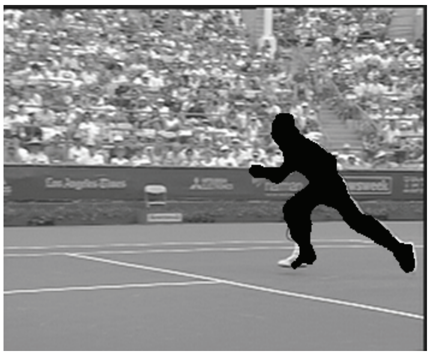
(b)

(c) (d)

Fig. 3.3 Two examples to show sprites generated using different feature points with the same number.
(a) The sprite generated using feature points extracted by conventional methods.
(b) The sprite generated using feature points extracted by the proposed method.
(c) A close look of the white line in (a). (d) A close look of the white line in (b).

3.2.3 Intelligent blending

The precision of segmentation mask affects the quality of the generated sprite. Although the unreliable region around segmentation mask boundary reduces the segmentation error in the reliability-based blending, some errors still can not be covered. Fig. 3.4 shows two frames with segmentation errors. The left foot of the player in both frames is not completely segmented; this will leave some ghostlike shadows after blending. A close view of the shadows in the blended sprite is shown in Fig. 3.5(a). Increasing the distance from the mask border given in the reliability-based blending schema may solve this problem, but other problem will occur. More segmentation errors can be covered using a larger distance, but it also increases the number of pixels being classified as unreliable. Thus the opportunity of an unreliable pixel being replaced by a reliable one is decreased. The reliable and unreliable pixels are blended by averaging separately. If an unreliable pixel is not replaced by a reliable one, the blending acts like the normal averaging. Fig. 3.5(b) shows a close view of the blended sprite using a larger distance. We can see that the right border, which is the boundary of reliable and unreliable regions, is blurred. Thus, how to give a suitable distance is a hard job. To avoid this problem, an intelligent blending strategy without requiring segmentation masks is proposed here.


(a)

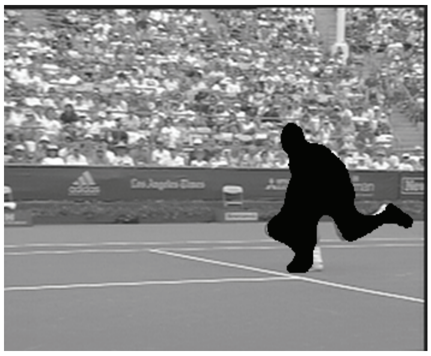
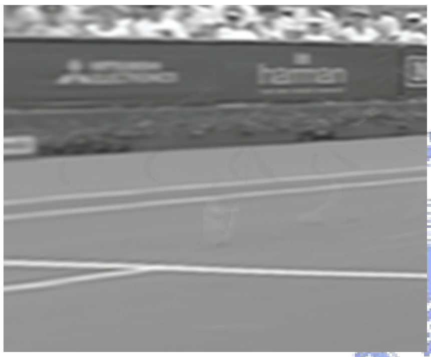
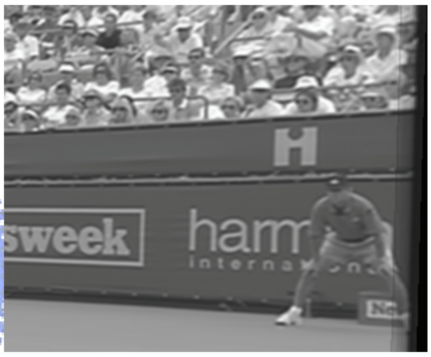
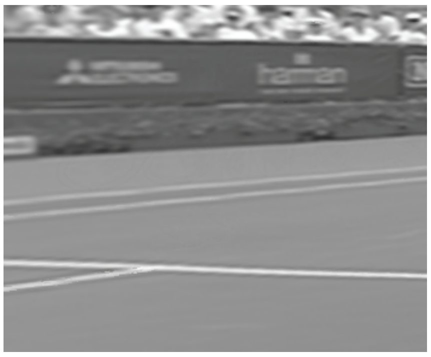

(b)

Fig. 3.4 Segmentation errors in player's feet. (a) Frame 255 with left foot incompletely segmented. (b) Frame 258 with both feet incompletely segmented.


(a)


(b)


(c)

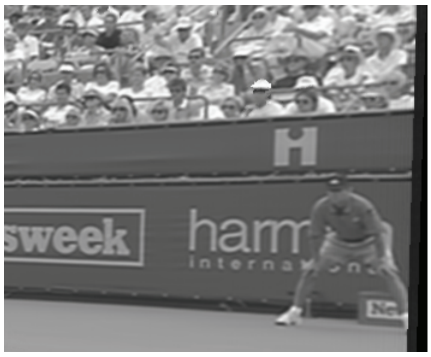

(d)

Fig. 3.5 Two examples to show the blended sprites using different methods.
(a) The first example of the generated sprite based on the reliability-based blending.
(b) The second example of the generated sprite based on the reliability-based blending.
(c) The first example of the generated sprite based on the intelligent blending.
(d) The second example of the generated sprite based on the intelligent blending.

The proposed intelligent blending strategy is based on a fact that for a series of pixels in video frames corresponding to the same location of a sprite, most of these pixels will be background; only few pixels are moving objects. Since objects are moving, those object pixels will come from different positions of an object, or even different objects, their intensities will have larger variation. On the contrary, intensities of those background pixels standing for the same background point in the real world should be similar, and can be found out by counting their occurrence.

Fig. 3.6 shows a flowchart of the proposed intelligent blending schema. Let X be the incoming pixel, S be the current sprite pixel. A candidate pixel C is used to store a candidate of incoming background pixel. Two counters C_S and C_C are used to store the number of pixels being blended into S and C, respectively. Initially, S and C are undefined and both counters are set as zero. A similarity check is performed on the incoming pixel by calculating two absolute differences:

$$\begin{aligned} D_S &= |X - S| \\ D_C &= |X - C| \end{aligned} . \tag{3.4}$$

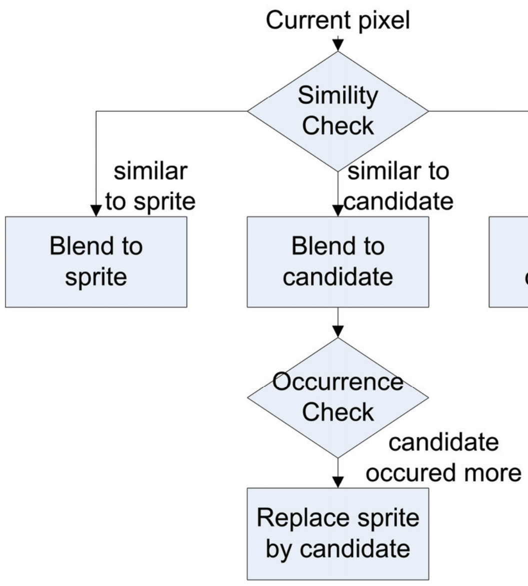


Fig. 3.6 Flowchart of the intelligent blending.

In the starting of the blending, since S is undefined, it is set to be gray value of the first incoming pixel and C_S is set to one. After filling S, the similarity check is conducted. If an incoming pixel is unlike S (i.e. D_S is greater than a preset threshold T) and C is undefined, C is set to the gray value of the incoming pixel and C_C is set to one; otherwise, if D_S is smaller than T and D_C, the incoming pixel is considered as a background pixel and is blended into S, C_S is increased by 1. If D_C is smaller than T and D_S, the incoming pixel is blended into the candidate C, C_C is increased by 1. Two counters are compared when a pixel is blended into the candidate C. If the candidate counter is larger than the sprite counter, the sprite and the sprite counter are replaced by the candidate and the candidate counter. Then the candidate and its counter are reset to undefined and zero, respectively. This replacement is based on the fact that being described before. Since the candidate appears more frequently than the current sprite, the candidate is more likely to be the background.

If both D_S and D_C are larger than T, the candidate is replaced by the incoming pixel, and the candidate counter is set to one. With a series of incoming object pixels, the candidate is continuously replaced by the incoming pixels until a background pixel is replaced into the candidate. If the background pixels appear continuously, the accumulation of candidate counter will begin.

The boundaries of frames are often non-reliable and should be removed from blending. In the proposed method, the affects of boundaries are eliminated by counting the boundaries pixels once. The pixels near the frame border within a preset distance W are defined as boundary pixels. The boundary pixels are checked and blended into the sprite or the candidate as normal pixels, but the corresponding counter is not increased. This will ensure the boundary pixels to be replaced quickly when normal pixels are inputted.

Two examples of the blending results using the proposed intelligent blender are shown in Fig. 3.5(c) and 3.5(d). In contrast to the results using reliability-based blender shown in Fig. 3.5(a) and 3.5(b), the ghostlike shadows are eliminated and the sprite border is clear and sharp. In order to examine the results easier, a contrast-enhanced version of Fig. 3.5 is provided in Fig. 3.7.

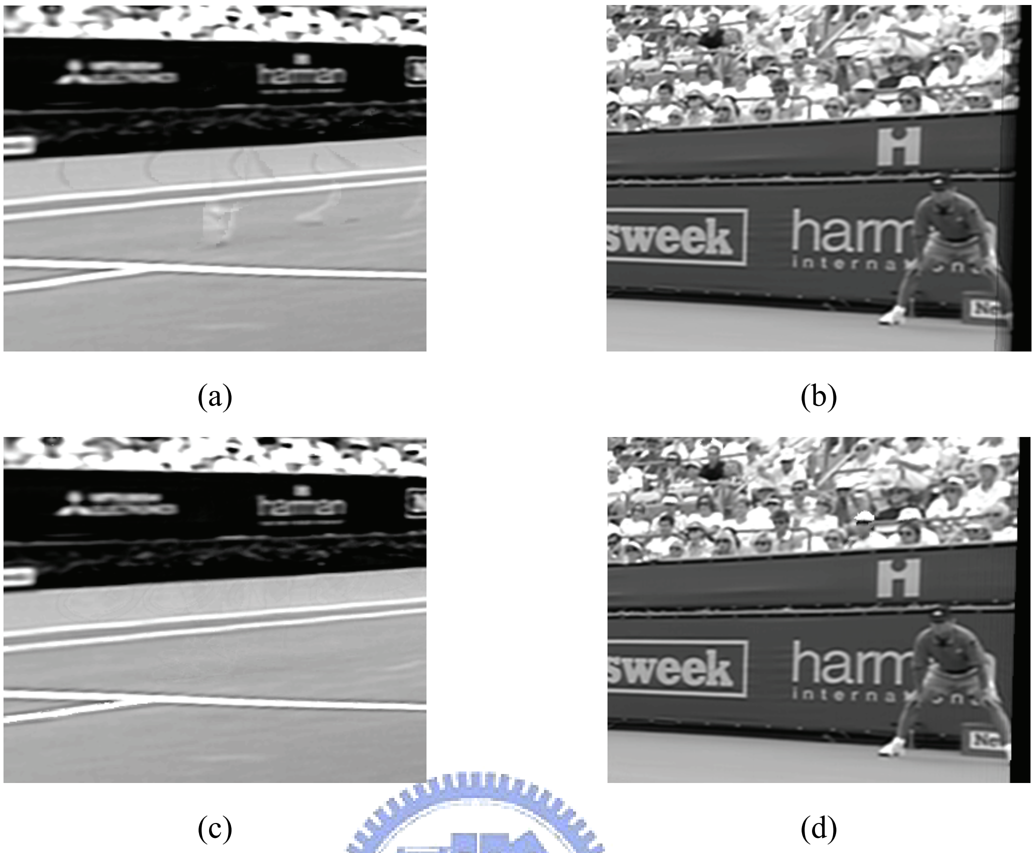
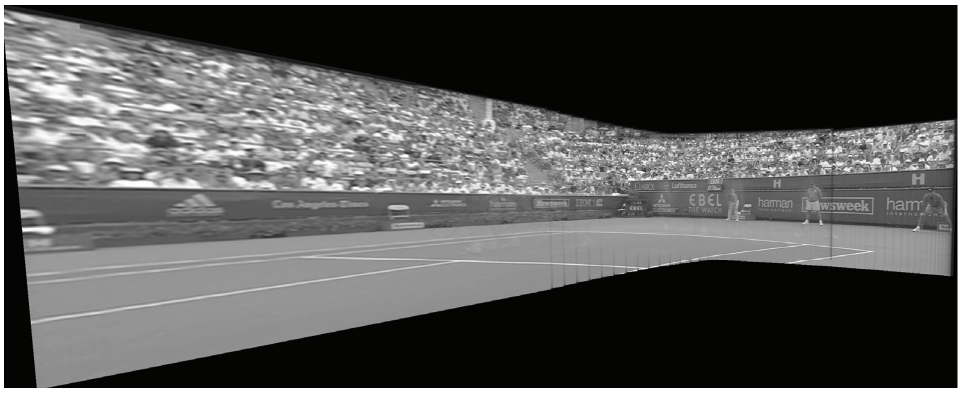


Fig. 3.7 Contrast-enhanced version of Fig. 3.5. (a) Contrast-enhanced version of Fig. 3.5(a). (b) Contrast-enhanced version of Fig. 3.5(b). (c) Contrast-enhanced version of Fig. 3.5(c). (d) Contrast-enhanced version of Fig. 3.5(d).

3.3 Experimental Results

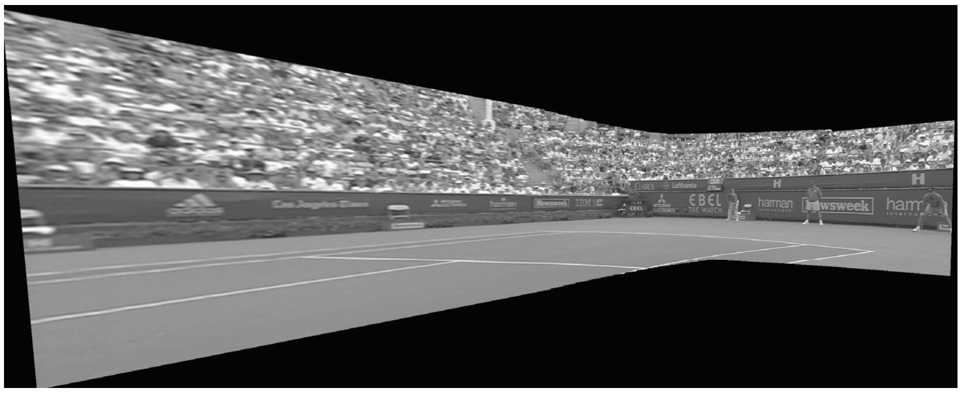
The aim of sprite generation is to reconstruct the background from the generated sprite perfectly. The quality of the reconstructed background for each frame is often measured by PSNR. In most cases, the PSNR is a good measurement to describe the quality. However, the PSNR is fooled in seldom cases. A comparison in visual quality of the generated sprites is also performed.

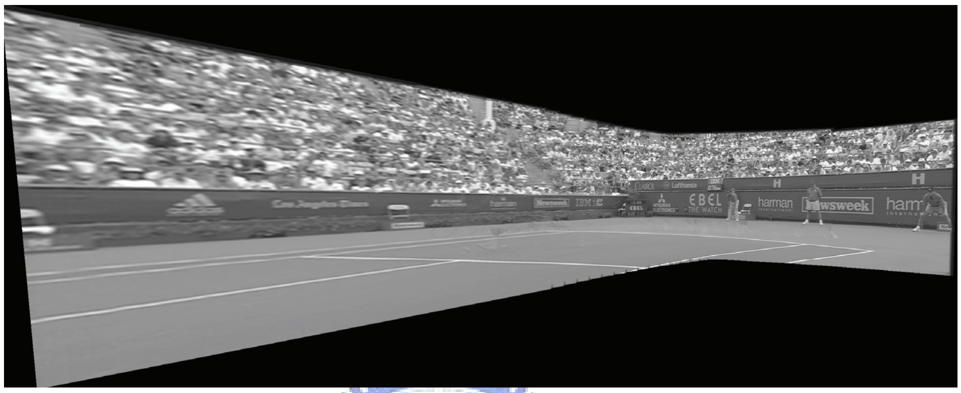
The GMPs are estimated using the two-staged GME with the proposed balanced feature points described in Sections 3.2.1 and 3.2.2. Then sprites are mixed by different blending strategies with the same GMPs. Backgrounds are reconstructed from the generated sprites and their PSNRs are calculated. Pixels of moving objects are excluded when calculating the PSNR since we are measuring the qualities of the reconstructed backgrounds. The qualities of generated sprites are measured by computing the averaging PSNR of the reconstructed backgrounds. The generated sprites are shown in Fig. 3.8. Fig. 3.8(a) is generated by the averaging blending strategy employed in the MPEG-4 VM without segmentation masks. The result using the averaging blending strategy with manually segmented masks is shown in Fig. 3.8(b). Fig. 3.8(c) shows the sprite generated using the reliability-based blending strategy based on the rough segmentation masks extracted via the method developed by Lu *et al.* [19]. Finally, the sprite generated using the proposed intelligent blender is shown in Fig. 3.8(d). Fig. 3.9 shows one of the reconstructed backgrounds using three different strategies respectively.



(a)

Fig. 3.8 (*Continued*) The generated sprites. (a) Sprite generated using averaging blending without segmentation masks. (b) Sprite generated using averaging blending based on manually segmented masks. (c) Sprite generated using reliability-based blending. (d) Sprite generated using intelligent blending.


(b)


(c)

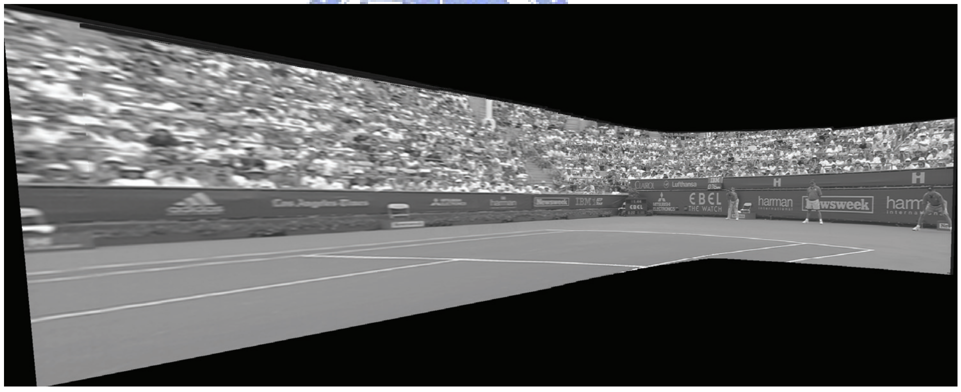

(d)

Fig. 3.8 The generated sprites. (a) Sprite generated using averaging blending without segmentation masks. (b) Sprite generated using averaging blending based on manually segmented masks. (c) Sprite generated using reliability-based blending. (d) Sprite generated using intelligent blending.

Since all moving objects are blended, the sprite generated by averaging blending will have shadows in several places, which are obvious in a reconstructed frame shown in Fig.

3.9(a). However, if perfect manually masks are provided, the shadows are eliminated completely and the averaging blending can achieve excellent results, as shown in Fig. 3.9(b). Most shadows can be removed using the reliability-based blending except some ill-segmented parts shown in the half-bottom of Fig. 3.9(c). The reconstructed background using the proposed intelligent blending is shown in Fig. 3.9(d). We can see that the sprite generated by our method is perceptually the same as the result using the average blending with perfect manually masks provided.

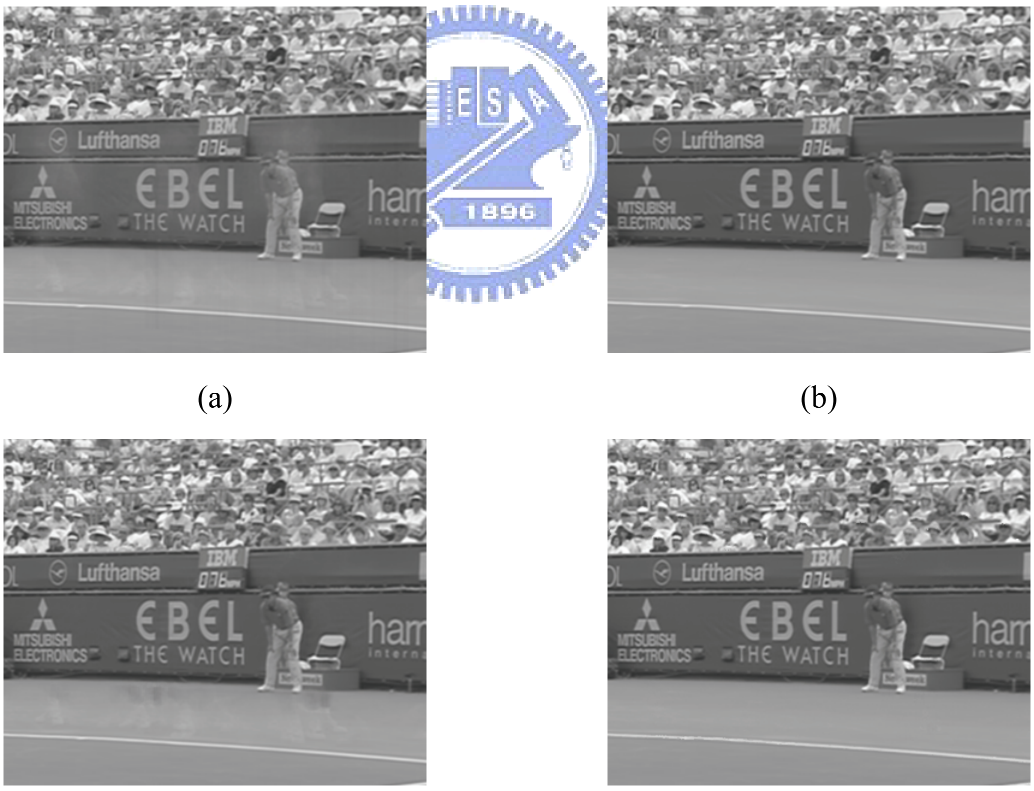


Fig. 3.9 The reconstructed backgrounds. (a) Averaging without segmentation masks. (b) Averaging with manually segmented masks.
(c) Reliability-based blending. (d) Intelligent blending.

A quantitative comparison in PSNR is performed and illustrated in Fig. 3.10. While calculating the PSNR of a frame, only the background parts in the frame are attending the calculation because the sprite contains only the information of backgrounds in a video sequence. Manually segmented masks are employed to exclude object parts in the frame.

The results of the average blending with and without manually segmented masks are plotted in dash-dotted and dotted line respectively in Fig. 3.10. The result without masks is degraded by the shadows of moving objects and the frame borders, and has low average PSNR of 26.23dB. With manually segmented masks, the averaging blending shows superior results not only in the visual quality but also in the measured PSNR. The average PSNR is 28.38dB and is the best result in our tests.

The reliability-based and intelligent blending strategies are plotted in dashed line, and normal line respectively. The reliability-based blending has average PSNRs 28.20dB. The proposed intelligent blending has average PSNRs 28.29dB, which is slightly higher than that of reliability-based blending and close to that of the average blending with perfect masks. These experiments show that the proposed method can generate high visual quality sprite without needing any segmentation mask.

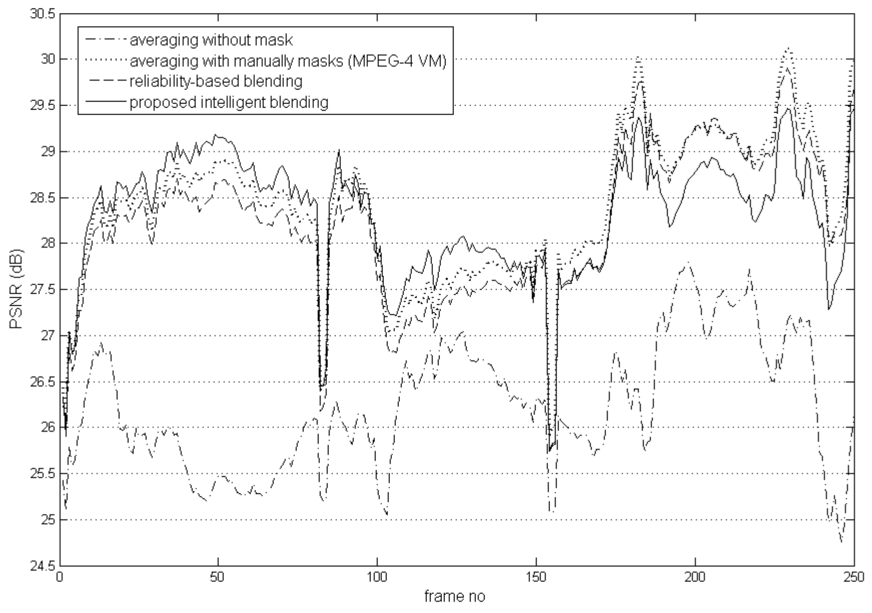


Fig. 3.10 PSNR comparison of different blending strategies.

The result of Lu *et al.*'s sprite generator [21], which is shown in Fig. 3.11, is also quoted as a comparison. In contrast to the result of the proposed generator shown in Fig. 3.8(d), the proposed generator can generate better results. The average PSNR of Lu's generator is 23.1dB, which is much lower than that of the proposed generator (28.29dB). To make comparison more clear, we take close views of three parts from the generated sprites in Fig. 3.11 and show them in Fig. 3.12. From part 1 shown in Figs. 3.12(a) and (b), we can see that the generated sprite using [21] skews seriously. Figs. 3.12(c) and (d) show part 2, the white lines in Fig. 3.12(d) are registered well, but in Fig. 3.12(c) are not. The above two faults are due to the inaccuracy of the estimated GMP, and do not exist in the result of our generator. Part 3 shown in Figs. 3.12(e) and (f) also demonstrates that our method is superior to Lu *et al.*'s.

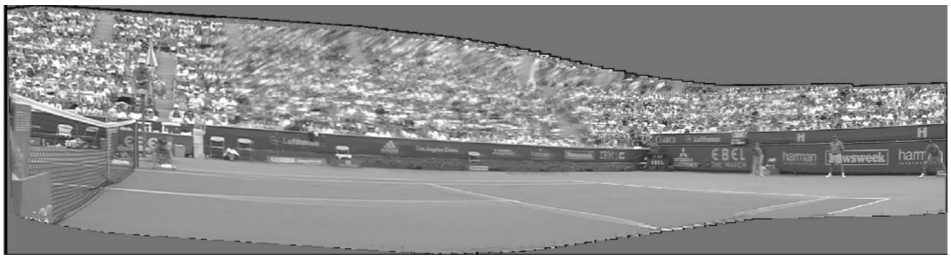
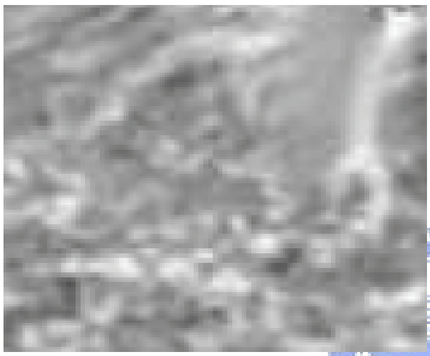
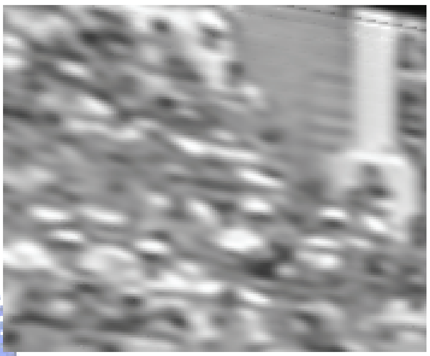


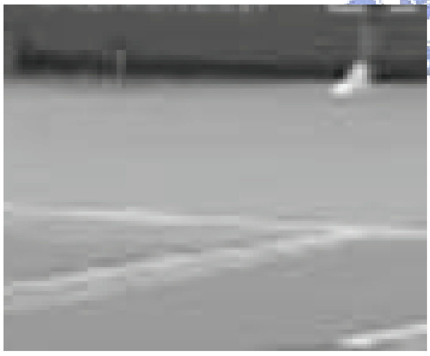
Fig. 3.11 The generated sprite of Lu *et al.*'s work [21].



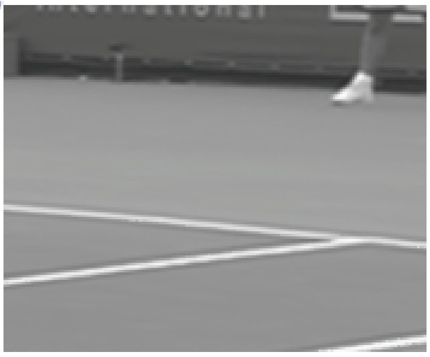
(a)



(b)

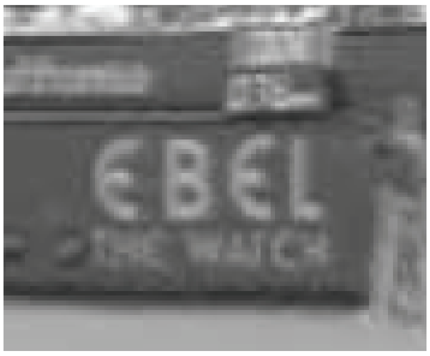


(c)

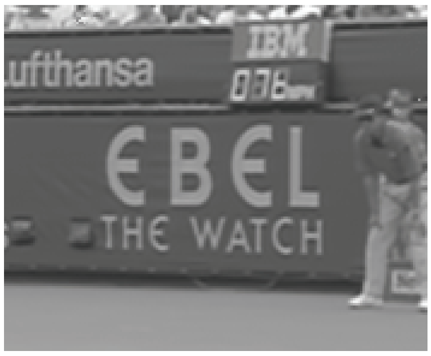


(d)

Fig. 3.12 (*Continued*) Close views of the generated sprites.
(a) Part 1 of Lu *et al.*'s work [21]. (b) Part 1 of the proposed method.
(c) Part 2 of Lu *et al.*'s work [21]. (d) Part 2 of the proposed method.
(e) Part 3 of Lu *et al.*'s work [21]. (f) Part 3 of the proposed method.



(e)



(f)

Fig. 3.12 Close views of the generated sprites.
(a) Part 1 of Lu *et al*.'s work [21]. (b) Part 1 of the proposed method.
(c) Part 2 of Lu *et al*.'s work [21]. (d) Part 2 of the proposed method.
(e) Part 3 of Lu *et al*.'s work [21]. (f) Part 3 of the proposed method.

CHAPTER 4
A NEW APPROACH FOR FAST MULTIPLE SPRITES GENERATION

Farin *et al*. have proposed a novel technique denoted as multiple sprites or multi-sprites that divides a video sequence into several sub-sequences. The backgrounds of each sub-sequence are stored by its own sprite. Using multiple sprites reduces the geometrical distortions and the storage required by a single large sprite. However, Farins' technique uses exhaustive searches to find the optimum sub-sequences and optimum reference frame of each sub-sequence. This makes the algorithm very time-consuming.

In our dissertation, 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* direction 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.

A reference frame selection algorithm is developed based on the idea of Messey and Bender's work [37]. 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.

4.1 Proposed Feasible Partition Points Selecting and Reference Frames Finding Methods

Farin *et al.*'s method achieves optimal partition results with high computational complexity, even if their efficient algorithm is applied. If we can reduce the possible combinations of sub-sequences 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 total sprite area to Farins'.

4.1.1 Analysis of Accumulated Translation

As mentioned previously, the geometric projection distortion comes from the camera rotation. Farin *et al.*'s experiments [38] also show that the sprite area grows exponentially as camera pan angle increases. Thus the selecting of partition frames must be highly related to the effect of camera rotation. In order to capture the effect of camera rotation, the global translations between video frames are calculated and analyzed.

The global translation between two frames is a measurement of background displacements. Let frame *i* and frame *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

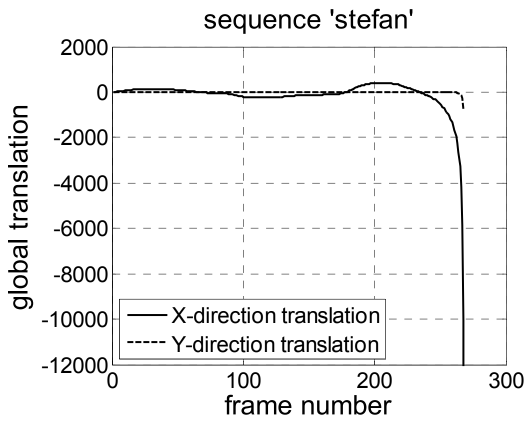
$$\vec{d}_p = T_{ji}(\vec{p}) - \vec{p}\,, \tag{4.1}$$

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,

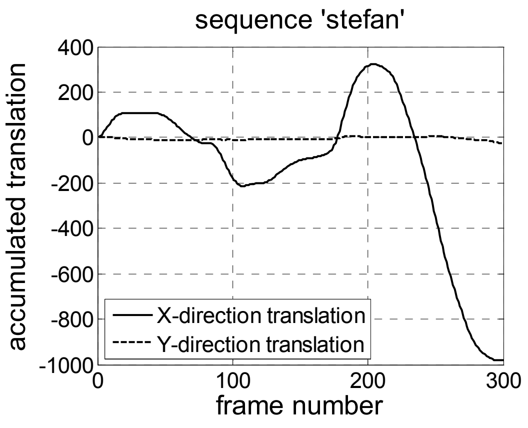
$$\vec{t}_{ji} = \frac{1}{4}\sum_{P\in\{LT,RT,LB,RB\}} \vec{d}_P \,, \tag{4.2}$$

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 frame *i* and frame *j*.

The global translations of sequence ‘stefan’ are illustrated in Fig. 4.1(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. 4.1(a), one can see that the view of background moves toward the right direction in the first thirty frames. Then it moves left and crosses the first frame in the next ninety frames. And then it moves toward the right again until frame 205, finally it moves toward the left in the rest of frames.



(a)



(b)

Fig. 4.1 Background displacements of ‘stefan’.
(a) Global translations. (b) Accumulated translations.

The figure also shows that the view of background moves toward left quickly after frame 260. The magnitudes of global translations increase quickly from hundred to over ten thousand 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.

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. (3.6). 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 frame j and frame i, that is

$$\vec{a}_{ji} = \sum_{k=i+1}^{j} \vec{t}_{k(k-1)} \,. \tag{4.3}$$

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

$$\vec{a}_{ji} = \vec{a}_{(j-1)i} + \vec{t}_{j(j-1)} \,, \tag{4.4}$$

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

The accumulated translations for sequence ‘stefan’ are illustrated in Fig. 4.1(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 can not be projected into the first frame.

4.1.2 Accumulated Translation Based Feasible Partition Point Finding Method

One of the goals of a multi-sprite partition algorithm is to find some partition points to split the video sequence into several sub-sequences. 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 *x*-axis direction. By the similar way, we can also find the feasible partition points, FPY, based on accumulated translation in *y*-axis direction.

Fig. 4.2 shows the *x*-axis accumulated translations which have been shown in Fig. 4.1(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 are going 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.

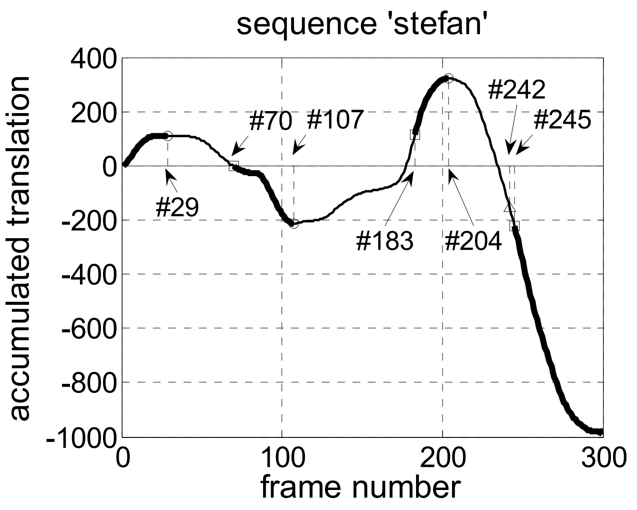


Fig. 4.2 Finding feasible partition points.

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. By 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. 4.2. The 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} combinations of possible partitions. In Fig. 4.2, feasible partition points are frame 70, 183, and 245. Comparing to 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*-axis and *y*-axis) are found. The final feasible partition points are the union of the *x*- and *y*-axis partition points (FPX and FPY).

4.1.3 Scaling Factor Based Feasible Partition Point Finding Method

Using 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. Fig. 4.3 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 frames is chosen in this dissertation. Area is the product of width and height, and area ratio will combine both the effect of width and height ratio. This makes area ratio more robust than width or height ratio.

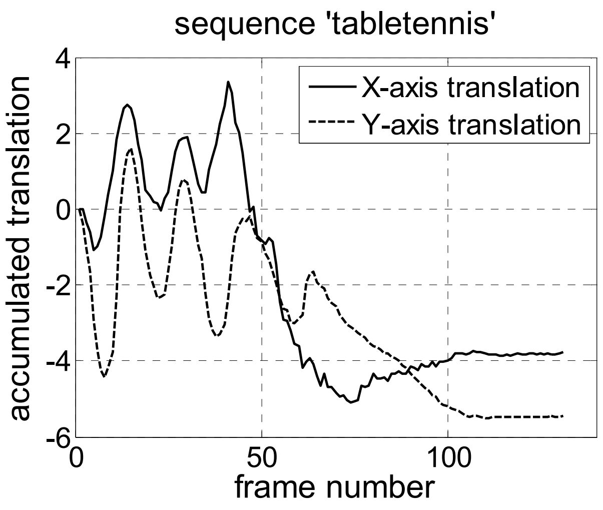


Fig. 4.3 Accumulated translations of ‘tabletennis’.

An ordinary geometric transformed frame is illustrated in Fig. 4.4. 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. 4.4. 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:

$$\begin{aligned} left &= (x_A + x_D)/2 \\ right &= (x_B + x_C)/2 \\ top &= (y_A + y_B)/2 \\ bottom &= (y_C + y_D)/2. \end{aligned} \tag{4.5}$$

The revised bounding box is shown by the solid-lined rectangle in Fig. 4.4. 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

$$AREA_{ji} = (right - left) \times (bottom - top)\,. \tag{4.6}$$

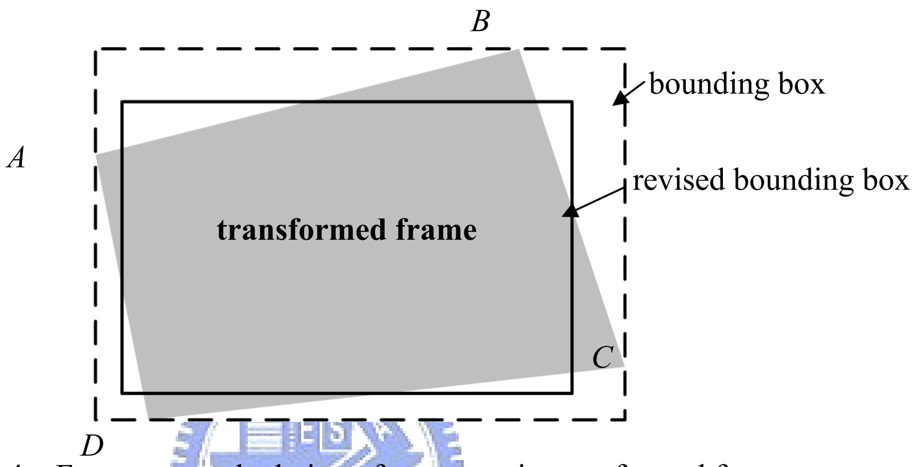


Fig. 4.4 Frame area calculation of a geometric-transformed frame.

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

$$s_{ji} = \sqrt{AREA_{ji} / AREA_{ii}}\,. \tag{4.7}$$

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 factors. The accumulated scaling factor of the transformed frame *j* versus the transformed frame *i* is defined as

$$as_{ji} = \prod_{k=i+1}^{j} s_{k(k-1)} = as_{(j-1)i} \times s_{j(j-1)}\,, \tag{4.8}$$

where as_{ii} is defined to be one. The accumulated scaling factors of the sequence 'tabletennis' are illustrated in Fig. 4.5.

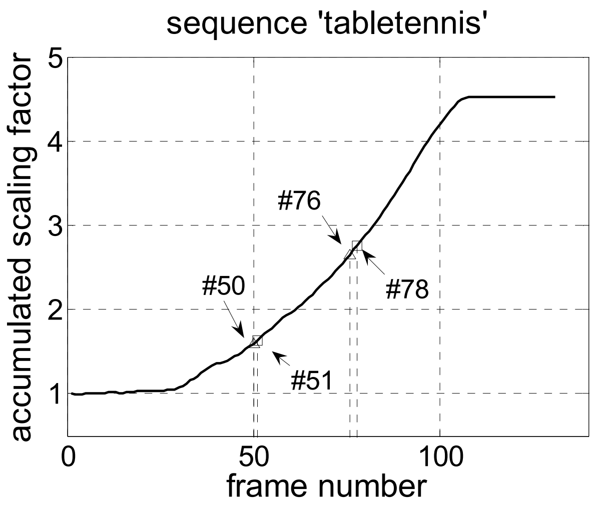


Fig. 4.5 Accumulated scaling factors of 'tabletennis'.

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

$$\frac{\max_i\{as_{ik} \mid k < i \le h\}}{\min_i\{as_{ik} \mid k < i \le h\}} < t, \tag{4.9}$$

where t is a threshold. According to Eq. (4.9), we process frames in a video sequence sequentially. When Eq. (4.9) 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. (4.9). In our experiment, the threshold t is set to 1.8.

The fixed threshold t in Eq. (4.9) 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

$$t' = \sqrt[L]{\max_i \{as_{i1} \mid 1 \le i \le N\}}, \tag{4.10}$$

where L is the number of feasible partition points found based on Eq. (4.9) 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. (4.9) with the adjusted threshold $t^{'}$.

Fig. 4.5 shows the feasible partition points found based on the accumulated scaling factors. The selected feasible partition points of sequence 'tabletennis' are frame 51 and frame 78, which are marked by squares in Fig. 4.5. The optimal partition method [38] splits the sequence in frame 50 and 76, which are marked by triangles in Fig. 4.5.

4.2 Proposed Reference Frame Finding Method

The second goal of a multi-sprite partition algorithm is to find a good reference frame for each partitioned sub-sequence. In order to cover 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 [37] 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 center.

When a sub-sequence 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.

4.3 The Complete Algorithm

The proposed multi-sprite partition algorithm is based on the methods described in Sections 4.1 and 4.2. 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.

4.3.1 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 frame $i-1$. Let (1,1), (1,Y), (X,1), (X,Y) be its four corner. Using Eqs. (4.1)-(4.2), we can obtain the local translation $\vec{t}_{i(i-1)}$. Based on the local translation, we can get the accumulated translations using Eqs. (4.3)-(4.4). Meanwhile, the locations of transformed corners in each transformed frame are used to find the revised bounding box via Eq. (4.5). The approximated area and accumulated scaling factors are calculated by applying Eqs. (4.6)-(4.8). Then three sets of feasible partition points (FPX, FPY, and FPS) are found by the method described in Sections 4.1.2 and 4.1.3. And the candidate partition points is set to be the union of these three sets of feasible partition points.

After obtaining the feasible partition points, the reference frames of all subsequences in all possible partitions are found using the method described in Section 4.2.

4.3.2 Reference frame validation

Although the reference frames are obtained in the previous section, the selected reference frame may have 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 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 sub-sequence 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 sub-sequence 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.

4.3.3 Sequence partition

In order to find the proper partition points, the optimal partition algorithm [38] described in Section 1.5.2 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 sub-sequences. In our method, the sequence is partitioned only in the candidate partition points found in Section 4.3.1. Therefore, a much smaller coding-cost matrix can be made up.

Assume that *M* candidate partition points $\{v_1,...,v_M\}$ are found in Section 4.3.1. 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 $V = \{v_0,...,v_{M+1}\}$ of size *M*+2. The area of bounding box $b_{i;j}$ of sub-sequence $(v_i, v_j - 1)$ 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 Section 4.2.

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

$$c_{i;j} = \begin{cases} b_{i;j} & if\, i < j \\ 0 & otherwise \end{cases}, \qquad (4.11)$$

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

The matrix *C* is applied to Eq. (1.9) as the sprite coding-cost matrix $\left\|S_{i;k}\right\|$. Then based on Eqs. (1.11) and (1.12), the partition of sequence can be found using the method described in Section 1.5.2.

4.4 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 the sprite generator with intelligent blending proposed in Section 2, and the same parameters are used in both the proposed and the optimal methods. The testing platform is an IBM laptop with mobile PentiumIII 800MHz CPU and 640MB of RAM. Both methods are implemented and simulated by Matlab. The results of using a single sprite are also included as a comparison.

Table 4.1 and Table 4.2 show the results of sequence 'stefan' using perspective and affine motion model respectively. Table 4.3 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 [38] 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.

Table 4.1 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 4.2 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

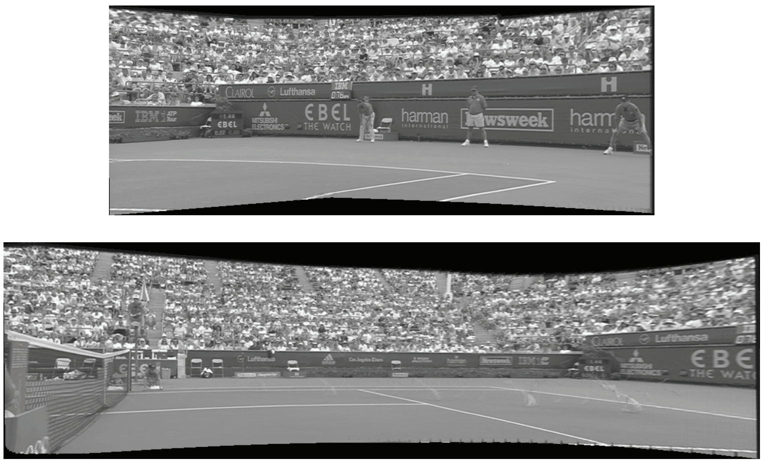
Table 4.3 Experimental results of sequence ‘tabletennis’.

	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

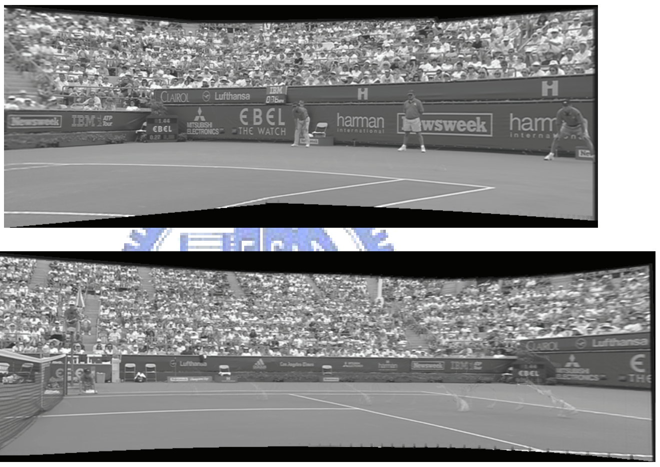
The results of the proposed method with normal validation and fast validation are also listed in tables. The proposed method divides the sequence ‘stefan’ into partitions at frame 245 and divides the sequence ‘tabletennis’ into three partitions at frame 52 and 78. The partition points of the proposed 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 10 times. If the fast validation method is applied, the executing times can be further decreased to 4.1 and 2.7 seconds, which is 35-190 times fast 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.

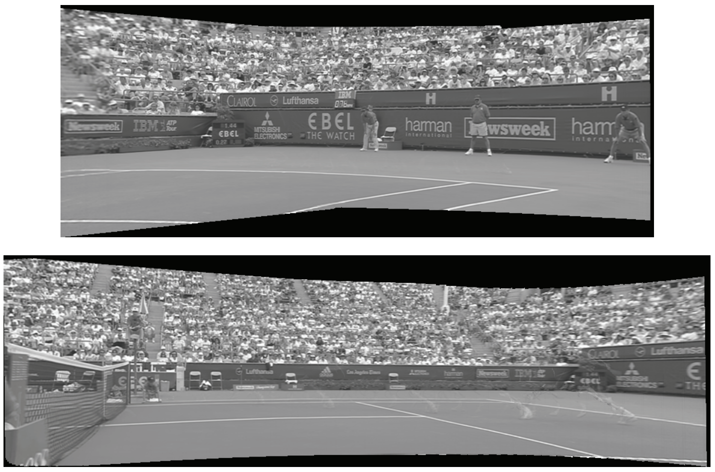
The generated sprites of sequence ‘stefan’ by the optimal method and the proposed methods are shown in Fig. 4.6 respectively. We can see that the generated sprites are perceptually similar, excepting for the dimensions of sprites. The effects of geometric distortions are lightened and the qualities of generated sprites are preserved. The generated sprites of sequence ‘tabletennis’ by different methods are shown in Fig. 4.7.



(a)

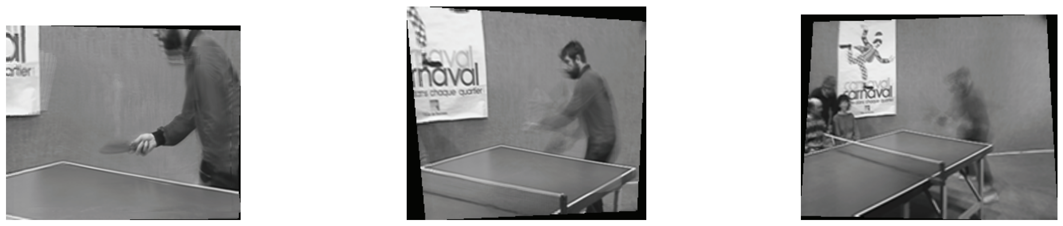


(b)

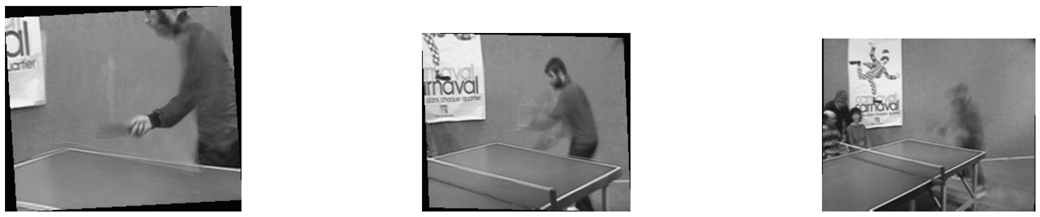


(c)

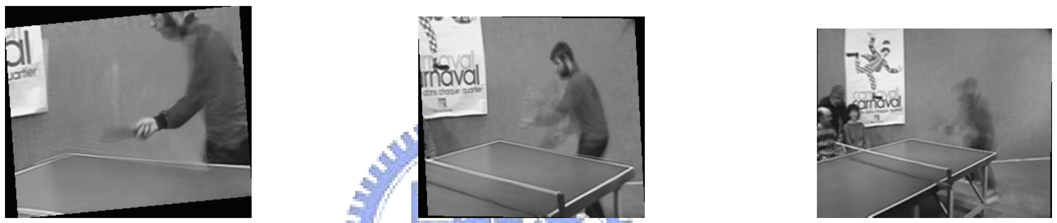
Fig. 4.6 Generated sprites of sequence ‘stefan’ by different methods.
(a) Farin *et al*.’s optimal method. (b) Proposed method with normal validation.
(c) Proposed method with fast validation.



(a)



(b)



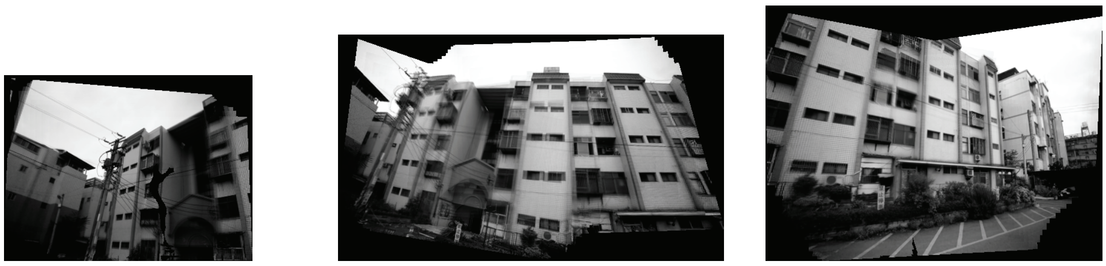
(c)

Fig. 4.7 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.

The experimental results of sequence ‘building’ are listed in Table 4.4. The sequence consists of wide camera movements in *y*-axis direction and continuous panning in *x*-axis direction. From Table 4.4, one can see that the proposed method works well. Fig. 4.8 shows the generated sprites of the sequence.

Table 4.4 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)

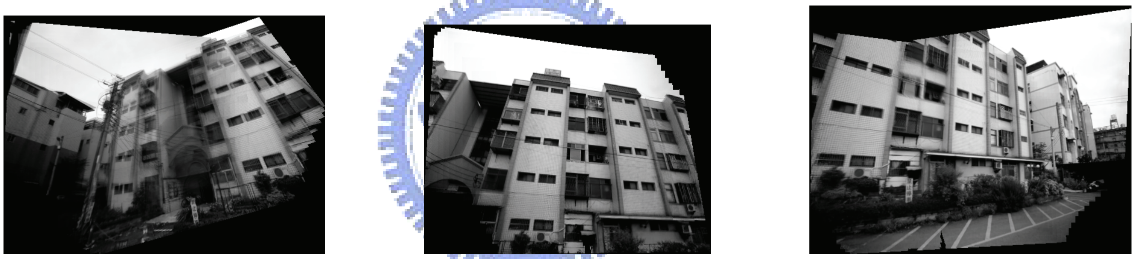

(b)

Fig. 4.8 Generated sprites of sequence ‘building’ by different methods.
(a) Farin *et al.*’s optimal method. (b) Proposed method with fast validation.

4.5 Complexity Analysis

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

The complexity of Farins’ optimal method is divided into two parts: the building of coding cost matrix described in Section 1.5.1, and the optimal partition algorithm described in

Section 1.5.2. While building the cost matrix, the coding cost of all sub-sequences 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, they had developed a method to reduce the space required to N^2. The optimal partition algorithm using Eq. (1.12) 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 is also 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 Section 4.3.1, finding reference frame for all possible sub-sequences 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 hold in memory and the coding-cost matrix must be generated. These will need $2N+M^2$ space.

Table 4.5 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. Since M is usually very small in contrast to N in practical, for example, M=9 and N=300 in the sequence 'stefan', this makes the complexity of the proposed method better than the optimal method.

Table 4.5 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)

CHAPTER 5
CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

5.1 Conclusions

An effective sprite generator without segmentation masks is proposed. The method is based on MPEG-4's framework. A balanced feature point extraction with object removing is introduced to increase the precision of estimated global motion parameters. In order to remove the demand of segmentation masks, an intelligent blending strategy is proposed. It counts the occurrence of pixels and chooses the pixels with higher count to be blended into the sprite. The ghostlike shadows in the sprite generated by conventional reliability-based blending are eliminated, and the average PSNR is increased slightly.

An efficient and fast method for generating multiple sprites is also proposed. In contrast to the conventional sprite generating method that using 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 sub-sequence 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 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 check 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.

5.2 Future Research Directions

A sprite is a 'pure' background generated from a video sequence. This makes generated sprites very useful. For example, in a change-detection-mask based video object detection system, a sprite can be applied as the reference background instead of previous frames to achieve better results. We will work on the moving object detection methods based on generated sprites.

REFERENCES

[1] ISO/IEC MPEG Video Group, "MPEG-4 video international standard with amd. 1," ISO/IEC 14496-2, Jul. 2000.

[2] F. Dufaux and F. Moschieni, "Background mosaicking for low bit rate coding," Proc. IEEE Int. Conf. Image Processing, vol. 1, Sept. 1996, pp.673-676.

[3] M. Irani and P. Anandan, "Video indexing based on mosaic representations," Proc. IEEE, 16(5) , pp. 905-921, May 1998.

[4] R. Szeliski, "Video mosaics for virtual environments," IEEE Computer Graphics and Applications, 16(2) , pp. 22-30, Mar. 1996.

[5] R. Szeliski and H. Y. Shum, "Creating full view panoramic mosaics and environment maps," Proc. ACM Conf. SIGGRAPH 97, Aug. 1997, pp. 251-258.

[6] M. Kourogi, T. Kurata, J. Hoshino and Y. Muraoka, "Real-time image mosaicing from a video sequence," IEEE International Conference on Image Processing (ICIP'99), Oct. 1999, pp. 133-137.

[7] S. Ji and H.W. Park, "Moving object segmentation with adaptive sprite for DCT-based video coder," IEEE International Conference on Image Processing, Oct. 2001, pp. 566-569.

[8] Y. Lu, W. Ga and F. Wu, "Automatic video segmentation using a novel background model," IEEE International Symposium on Circuits and Systems. (ISCAS), May 2002, pp. 807-810.

[9] S.Y. Chien, Y.W. Huang, B.Y. Hsieh, S.Y. Ma, and L.G. Chen, "Fast video segmentation algorithm with shadow cancellation, global motion compensation, and adaptive threshold techniques," IEEE Transactions on Multimedia, vol. 6, no. 5, Oct. 2004, pp. 732–748.

[10] D. Farin, P.H.N. de With, and W. Effelsberg, "Video-object segmentation using multi-sprite background subtraction," IEEE International Conference on Multimedia and Expo, ICME '04, pp. 343-346, Jun. 2004.

[11] A. Krutz, M. Kunter, T. Sikora, M. Mandal and M. Frater, "Motion-based object segmentation using sprites and anisotropic diffusion," International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS), Jun. 2007, pp. 35-38.

[12] S. Kopf, T. Haenselmann, D. Farin, and W. Effelsberg, "Automatic generation of video summaries for historical films," IEEE International Conference on Multimedia and Expo, ICME '04, pp. 2067-2070, Jun. 2004.

[13] T. Sikora, "The MPEG-4 video standard verification model," IEEE Transactions on Circuits and Systems for Video Technology, vol. 7, no. 1, Feb. 1997, pp. 19-31.

[14] ISO/IEC MPEG Video Group, "MPEG-4 video verification model version 18.0," N3908, Jan. 2001.

[15] J.J. Moré, "The Levenberg-Marquardt algorithm: implementation and theory," Numerical Analysis, ed. G. A. Watson, Lecture Notes in Mathematics 630, Springer Verlag, pp. 105-116, 1977.

[16] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns, IEEE Trans. on Pattern Analysis and Machine Intelligence, vol. 13, no. 4, Apr. 1991, pp. 376–380.

[17] G.B. Rath and A. Makur, "Iterative least squares and compression based estimations for a four-parameter linear global motion model and global motion compensation," IEEE Trans. on Circuits and System for Video Technology, vol. 9, no. 7, Oct. 1999, pp.1075-1099.

[18] 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 and system for video technology, vol. 9, pp. 1227-1242, Dec. 1999.

[19] Y. Lu, W. Gao, and F. Wu, "Fast and robust sprite generation for MPEG-4 video coding," Proc. IEEE Pacific-Rim Conf. Multimedia, Beijing, China, pp. 118-125, Oct. 2001.

[20] Y. Lu, W. Gao, and F. Wu, "Sprite generation for frame-based video coding," Proc. IEEE Int. Conf. Image Processing, vol. 3, Thessaloniki, Greece, pp. 473-476, Oct. 2001.

[21] Y. Lu, W. Gao, and F. Wu, "Efficient background video coding with static sprite generation and arbitrary-shape spatial prediction techniques," IEEE Trans. Circuits and Systems for Video Technology, vol. 13, no. 5, pp. 394-405, May 2003.

[22] F. Dufaux and J. Konrad, "Efficient, robust and fast global motion estimation for video coding," IEEE Trans. on Image Processing, vol. 9, no. 3, Mar. 2000, pp. 497-501.

[23] A. Smolić and J-R. Ohm, "Robust global motion estimation using a simplified M-estimator approach," International Conference on Image Processing (ICIP), Sep. 2000, pp. 868-871.

[24] H. Richter, A. Smolić, B. Stabernack and E. Müller, "Real time global motion estimation for an MPEG-4 video encoder," Proceedings of Picture Coding Symposium (PCS '01), Apr. 2001, pp. 401-404.

[25] Y. Keller and A. Averbuch, "Fast gradient methods based on global motion estimation for video compression," IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, no. 4, Apr. 2003, pp.300-309.

[26] H, Alzoubi and W.D. Pan, "Reducing the complexity of MPEG-4 global motion estimation using pixel subsampling," Electronics Letters, vol. 44, Jan. 2008, pp.20-22.

[27] A. Smolić, K. Müller and J.-R. Ohm, "Global motion compensation and video mosaicing using different 2-D motion models," Proceedings of Picture Coding Symposium (PCS '01), Apr. 2001, pp. 331-334.

[28] L.G. Brown, "A survey of image registration techniques," ACM Computing Surveys, vol. 24, no. 4, Dec.1992, pp. 325-376.

[29] E.C. Hildreth, "The analysis of visual motion: from computational theory to neural mechanisms," Annual Review of Neuroscience 10, Dec. 1986, pp. 477-533.

[30] A. Azarbayejani, T. Starner, B. Horowitz, and A. Pentland, "Visually controlled graphics," IEEE Trans. Pattern Anal. Machine Intell., vol. 15, Jun. 1993, pp. 602-605.

[31] R. C. Gonzalez and R. E. Woods, "Digital Image Processing," 2nd Edition, Prentice Hall, Jan. 2002.

[32] N. Grammalidis, D. Beletsiotis and M.G. Strintzis, " Sprite generation and coding in multiview image sequences," IEEE Trans. on Circuits and Systems for Video Technology, vol. 10, no. 2, Mar. 2000, pp. 302-311.

[33] C.T. Hsu and Y.C. Tsan, "Mosaics of video sequences with moving objects," International Conference on Image Processing 2001, Oct. 2001, pp.387-390.

[34] H. Watanabe and K. Jinzenji, "Sprite Coding in Object-based Video Coding Standard: MPEG-4," World Multiconference on Systemics, Cybernetics and Informatics (SCI) 2001, Proc. Vol.XIII, pp.420-425, Jul. 2001.

[35] H.K. Cheung, W.C. Siu, "Robust global motion estimation and novel updating strategy for sprite generation," IET Image Processing, vol. 1, no. 1, Mar. 2007, pp. 13-20.

[36] T. Meier and K.N. Ngan, "Video segmentation for content-based coding," IEEE Transactions on Circuits and Systems for Video Technology, vol. 9, no. 8, Dec. 1999, pp. 1190–1203.

[37] M. Massey and W. Bender, "Salient stills: Process and practice," IBM Systems Journal, vol.35, no. 3-4, 1996, pp.557-573.

[38] D. Farin, P.H.N. de With, W. Effelsberg, "Minimizing MPEG-4 Sprite Coding-Cost Using Multi-Sprites," SPIE Visual Communications and Image Processing, Vol. 5308, pp. 234-245, Jan. 2004.

[39] D. Farin and P.H.N. de With, “Enabling arbitrary rotational camera motion using multisprites with minimum coding cost,” IEEE Trans. Circuits and system for video technology, vol. 16, no. 4, pp.492-506, Apr. 2006.

[40] S.Y. Chen, C.Y. Chen, Y.W. Huang and L.G. Chen, “Multiple sprites and frame skipping techniques for sprite generation with high subjective quality and fast speed,” IEEE International Conference on Multimedia and Expo 2002 (ICME ’02), Aug. 2002, pp. 785-788.

[41] S.Y. Chen, C.Y. Chen; W.M. Chao, C.W. Hsu, Y.W Huang and L.G. Chen, “A fast and high subjective quality sprite generation algorithm with frame skipping and multiple sprites techniques,” International Conference on Image Processing. 2002 (ICIP ’02), vol. 1, Sep. 2002, pp. 193-196.

[42] M. Kunter, A. Krutz, M. Drose, M. Frater and T. Sikora, “Object-based multiple sprite coding of unsegmented videos using H.264/AVC,” International Conference on Image Processing 2007 (ICIP ’07), Sep. 2007, pp. 65-68.

PUBLICATION LIST

We summarize the publication status of the proposed methods and our research status in the following.

(1) I-Sheng Kuo and Ling-Hwei Chen, "**High Visual Quality Sprite Generator using Automatic Segmentation and Intelligent Blending**," Visual Communications & Image Processing 2005 (VCIP '05), Proceedings of the SPIE, vol. 5960, Jul. 2005, pp. 2128-2139.

(2) I-Sheng Kuo and Ling-Hwei Chen, "**A High Visual Quality Sprite Generator using Intelligent Blending without Segmentation Masks,**" International Journal of Pattern Recognition and Artificial Intelligence, vol. 20, no. 8, Dec. 2006, pp. 1139–1158.

(3) I-Sheng Kuo and Ling-Hwei Chen, "**A Fast Multi-Sprite Generator with Near-Optimum Coding Bit-Rate,**" accepted by International Journal of Pattern Recognition and Artificial Intelligence.