Journal of Marine Science and Technology

Volume 18 | Issue 6 Article 13

FAST BLOCK MOTION ESTIMATION WITH EDGE ALIGNMENT ON H.264 VIDEO CODING

Chi-Han Chuang
Department of Computer Science and Engineering, National Taiwan Ocean University, 2, Pei-Ning Road, Keelung 20224, Taiwan, R.O.C.

Chien-Hua Su
Department of Computer Science and Engineering, National Taiwan Ocean University, 2, Pei-Ning Road, Keelung 20224, Taiwan, R.O.C.

Shyi-Chyi Cheng
Department of Computer Science and Engineering, National Taiwan Ocean University, 2, Pei-Ning Road, Keelung 20224, Taiwan, R.O.C., csc@mail.ntou.edu.tw

FAST BLOCK MOTION ESTIMATION WITH EDGE ALIGNMENT ON H.264 VIDEO CODING

Acknowledgements

This work was supported in part by National Science Council, Taiwan under Grant: NSC 96-2221-E-019-047-.

FAST BLOCK MOTION ESTIMATION WITH EDGE ALIGNMENT ON H.264 VIDEO CODING

Chi-Han Chuang*, Chien-Hua Su*, and Shyi-Chyi Cheng*

Key words: fast block matching, edge alignment, H.264, motion estimation.

ABSTRACT

This paper presents a novel block matching scheme with edge alignment strategy on H.264 video coding, which uses multiple references and multiple block sizes for motion estimation in order to improve the rate-distortion performance. In H.264, the computational complexity is linearly dependent on the number of allowed reference frames and block sizes using the full exhaustive search. Many fast block-matching algorithms reduce the computational complexity of motion estimation by carefully designing the search patterns with different shapes or sizes which have significant impact on the search speed and distortion performance. However, the search speed and the distortion performance conflict often with each other for these methods. In this paper, given a block in the current frame, we first apply a fast approximate method based on edge alignment to obtain a good initial motion vector as well as a tight initial bound of distortion measure. Then, considering the edge orientation of the block, a modified hexagon search is used to fine tune the motion vector in low computational complexity. The proposed algorithm also pays attentions to the characteristics of multiple reference frames and multiple block sizes in H.264. Computer simulation results show that the proposed method gives good performance and spans a new way to design a cost-effective real-time video coding system.

I. INTRODUCTION

As the Internet advances, the demand of new ways to represent, integrate, store and exchange multimedia information (such as text, image, audio, and video) has increased. A remarkable change occurred in the video-driven applications such as teleconference, videophone, and image-based multimedia services, because of the increased role of synthetic information and new two-way communication systems. Advanced video coding techniques that yield reconstructed images with good subjective image quality to make efficient use of the available bandwidth facilitate the development of video-based applications over the Internet. Among these methods, H.264 is an emerging international video coding standard, made by Joint Video Team (JVT) consisting of ITU-T Video Coding Experts Group (VCEG) and ISO/IEC MPEG Video Coding Group [5]. Comparing to MPEG-4 [9], H.263 [23], and MPEG-2 [10], H.264 achieves 39%, 49%, and 64% of bit-rate reduction, respectively [12].

The performance improvement achieved by H.264 comes mainly from the prediction part [5], [24] involving the motion estimation (ME) at quarter-pixel with variable block sizes and multiple reference frames. These novel techniques greatly reduce the prediction errors. In H.264, there are seven kinds of block size (16×16, 16×8, 8×16, 8×8, 8×4, 4×8, 4×4) and up to 5 reference frames for motion compensation (MC). The simulation results of the reference software of H.264 [13] show that it achieves more than 15% of bit rate reduction compared with only using block size 16×16 using variable block sizes and 5%-10% of bit rate reduction using multiple reference frames [22]. However, the complexity of ME in developing H.264 is obviously very high. Hence, it is a crucial issue to reduce the complexity of ME for H.264-based real-time video applications. In H.264, the factors to affect the performance of ME include: (1) the complexity of mode decision when doing motion estimation [25]; (2) the complexity of reference frames when doing motion estimation [8]; (3) the number of search points to complete a motion estimation process. As reported by Huang *et al.* [8], it takes more than 80% of execution time on doing motion estimation for the reference software of H.264/AVC, JM [13] to encode a video sequence.

The block-matching algorithm (BMA) is adopted by H.264. Among all BMAs, the well-known full-search block-matching algorithm (FSBMA) aims at finding the position with minimal block distortion from all possible candidate motion vectors over a predetermined neighborhood search window. Although FSBMA produces the best quality, it demands the most computation. Many fast BMAs, such as three-step search (TSS) [15], Diamond Search (DS) [28], and Hexagon Based Search (HEXBS) [22, 27] have been proposed to speed up the FSBMA with acceptable distortion performance. In [26], Xu and He introduce an additional early termination scheme with adaptive thresholds for Hybrid Unsymmetrical-Cross Multi-Hexagon-Grid Search (UMHES) [2]. Gonzalez-Diaz and

Paper submitted 03/12/09; revised 07/13/09; 08/20/09; accepted 11/12/09.
Author for correspondence: Shyi-Chyi Cheng (e-mail: csc@mail.ntou.edu.tw).
**Department of Computer Science and Engineering, National Taiwan Ocean University, 2, Pei-Ning Road, Keelung 20224, Taiwan, R.O.C.*

Diaz-de-Maria propose a motion classification-based search (MCS), which use a classifier based on some motion cues to choose the best search patterns [7]. These algorithms can save a lot of search points compared with FSBMA through carefully designing the search patterns.

Although the computing power of processors has been rapidly improved for the past decade, software-based real-time video encoders remain as a challenge even with the help of fast BMAs. Given a block in the current frame, the process to find the best match from previous frames using a fast BMA cannot be interrupted; traditional BMAs stop only when all the search points confined by a specific search pattern are examined. Thus, the search process might violate the real-time constraint –the time the process allowed to complete its work. A computation-aware scheme for software-based block motion estimation was proposed by Tai *et al*. [20] to allow the search process of block matching to stop once a specific amount of computation has been performed. However, the computation-aware BMA faces two problems. First, it brings another time-consuming operator –to dynamically determine the target amount of computation power allocated to a frame, and then allocates this to each block on a computation-distortion-optimization manner [20]. To solve the problem, some heuristic approaches, which offer faster decisions on real-time constraints but lead to larger distortion, were used in [20]. Second, as indicated in [20], Tsai's computation aware BMA scheme allocates more computation power to the macroblock with the largest distortion among the entire frame in a step-by-step fashion. This implies that random access of macroblocks is required. The random access flow requires a huge amount of memory accesses for all macroblocks to store the up-to-date minimum distortions, best motions vectors, and search steps. Chen *et al*. [1] proposed an adaptive computation-aware BMA to reduce the memory size requirement and further speed up tradition BMAs with better computation-distortion performance.

Traditional BMAs perform motion estimation with every block in the current frame as the basic unit, which independently search the best location in the search window from the reference frame. FSBMA checks 1089 points in P-frame when referring to one frame for MPEG-4 but needs 223245 search points in P-frame when referring to five reference frames and seven block sizes for H.264. Considering two neighboring blocks in the current frame, most of the search points for individual blocks are the same, since their search windows are obviously overlapped. In H.264, the search window and the centers of all seven block sizes are all the same, and thus the matching errors of 4×4 blocks can be reused to calculate the matching errors of larger block sizes. In this way, the computation load of variable block sizes can be reduced. More concisely, we can estimate the matching errors for all the 4×4 blocks in a macroblock first, and then estimate the matching errors for larger sizes.

Let it contain k sub-blocks in a macroblock. For each sub-block, we aim at finding the best match from the search window using block matching. To estimate motion vectors for all the sub-blocks simultaneously, it is possible to first apply a *k*-nearest neighbors searching scheme to quickly find the *k* candidates by removing a large amount of non-relevant search points. Then, for each block, we can easily find its corresponding best match from the candidate set of small size. The *k*-nearest neighbors searching problem has been studied extensively for a wide range of scientific and engineering applications including pattern recognition [21], object recognition [19], data clustering [11], vector quantization [16], and content-based image retrieval [4]. Many algorithms are also proposed to speed up the searching process [14, 17, 18]. Unfortunately, these fast *k*-nearest neighbors searching algorithms cannot speed up the block matching much since they have a common characteristic: the query point is not a set of blocks.

Edge features, which are recognized as an important aspect of human visual perception, are commonly used in shape analysis. Decomposition of images into low-frequency blocks and blocks containing visually important features (such as edges or lines) suggest visual continuity and visual discontinuity constraints. A block is visually continuous if the values of all the pixels in the block are almost the same. In contrast, if the variations of the pixel values in the block are noticeable, it is a visually discontinuous block. The mean of a visually continuous block is enough to represent the block. If a block is visually discontinuous and if a strong orientation is associated with it, then it should be coded as a kind of visually important feature. Using coded edges, we can represent the structure of an image without explicitly extracting visual features.

In this paper, a fast block motion estimation algorithm is proposed based on the application of the moment-preserving technique to detect a visually important feature, namely an edge in a given image block. Each given image is divided into non-overlapping square blocks and coded block by block. Edge features used to code an ordinary image, produce excellent image quality according to human perception and provide a promising approach for the representation of the image content with a compact code [3]. Simple and analytical formulae to transform the block content into the corresponding visual pattern code are derived in our previous work [3], which makes the computation very fast. Many fast block-matching algorithms reduce the computational complexity of motion estimation by carefully designing the search patterns with different shapes or sizes which have significant impact on the search speed and distortion performance. However, the search speed and the distortion performance conflict often with each other for these methods. The proposed method offers a fast solution for block motion estimation on H.264 video coding systems. For each basic 4×4 block of a macroblock, we first apply a fast approximate method based on edge alignment to obtain a good initial motion vector as well as a tight initial bound of distortion measure. Then, considering the edge orientation of the block, a hexagon search modified from [27] is used to fine tune the motion vector in low computational complexity. Finally, we design a fast intermode decision method to achieve better rate-distortion performance for

H.264. Computer simulation results show that the proposed method gives good performance and spans a new way to design a cost-effective real-time video coding system.

The remainder of this paper is organized as follows. In Section II, a fast block matching based on edge alignment is first discussed. In Section III, the proposed block motion estimation algorithm for H.264 is presented. Experimental results are shown in Section IV. Finally, Section V gives a brief conclusion.

II. BLOCK MATCHING USING EDGE ALIGNMENT

In H.264, each frame is divided into multiple nonoverlapping blocks of size 16 × 16 pixels. Each block is further divided into several sub-blocks of variable sizes according to a rate-distortion optimization scheme. Each of the sub-blocks then performs block motion estimation to remove the temporal redundancy of the video sequence. In this work, a 16 × 16 block is initially divided into 16 4 × 4 sub-blocks with their motion vectors found based on a fast block matching using edge alignment.

Block matching for motion estimation is an approach to shift or warp image blocks relative to each other and to look at how much the pixels agree according to certain criteria. On such function, often used in video coding because of its speed, is the sum of absolute differences (SAD) metric, i.e.,

$$SAD_{x,y}(u,v) = \sum_{i=1}^{N-1}\sum_{j=1}^{N-1} | I_t(x+i, y+j) - I_{t-1}(x+u+i, y+v+j) | \qquad (1)$$

where $\vec{u} = (u,v)$ is the displacement and $| I_t(x+i,y+j) - I_{t-1}(x+u+i,y+v+j) |$ is called the displaced frame difference. Given a block in the current frame, the full searching technique tries all possible alignments and is too slow in practice. Besides block matching techniques with full or partial pixel information, feature-based alignment methods are possible to speed up the process of motion estimation.

At the lowest level of computer vision, potentially useful visual patterns such as edges and line segments can be extracted from an image without any priori knowledge of the image content. In our approach, segmentation and detailed object representation are not required. A given image is partitioned into a set of non-overlapping square blocks. Each block is coded as either a uniform block or an edge block. The edge in each block is detected by the moment-preserving edge detection technique which was proposed in our previous work [3], and the image can be reconstructed according to the parameters of these blocks. The continuous two-dimensional edge model specified by four parameters, two representative gray (color) values h_1 and h_2, an edge translation l, and an orientation angle θ for an edge in square block B is shown in Fig. 1. The edge translation l is defined as the length from the center of the edge model to the transition, and is confined within the range of $-\sqrt{2}$ to $+\sqrt{2}$. The parameter θ specified the direction of the edge and is confined within the range of 0 to 180 degrees. The solution to the edge detection problem in a given block is analytic and this means that the edge detection process can be performed very fast for large-database applications with no need for special hardware.

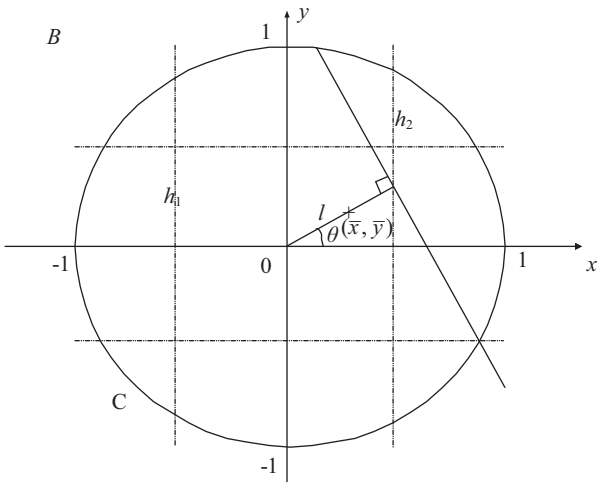

Fig. 1. An edge model in a 4 × 4 block *B*. The circle *C* is inscribed in *B*, and $(\bar{x}, \bar{y})$ are the coordinates of the centers of gravity of the gray (color) values inside *C*.

In our approach, there are two cases to align the edge patterns of two blocks. First, given a pair of edge blocks (*B, B'*) with the same edge orientation, the edge of *B* is said to be aligned properly with that of *B'* by translating *B'* if necessary such that the edge pattern of *B'* exactly coincides with that of *B*, shown in Fig. 2(a). What would happen to the process of edge alignment if the edge orientations for *B* and *B'* are different? In this case, the edge pattern of *B* is first translated and then rotated by an angle which is equal to the difference between the edge orientations of *B* and *B'* in order to overlap the two edge patterns, shown in Fig. 2(b). Obviously, the former is a special case of the second. The problem is what is the amount of translation for the second case?

Given two edge blocks *B* and *B'* characterized with the edge orientation and translation as (θ, l) and (θ', l'), respectively, we would like to align the edge pattern of *B* with that of *B'* by rotating and translating the edge pattern of *B*, shown in Fig. 3. The proposed edge alignment process consists of two steps. First, we translate the center of *B'* to the center of *B*, which results in the first displacement for *B'*:

$$\vec{u}_1 = (x_C - x'_C, y_C - y'_C) \qquad (2)$$

where (x_C, y_C) and (x'_C, y'_C) are the center coordinates of *B* and *B'*, respectively. Then, we further translate *B'* by

$$\vec{u}_2 = (Nd_l \cos\theta, Nd_l \sin\theta) \qquad (3)$$

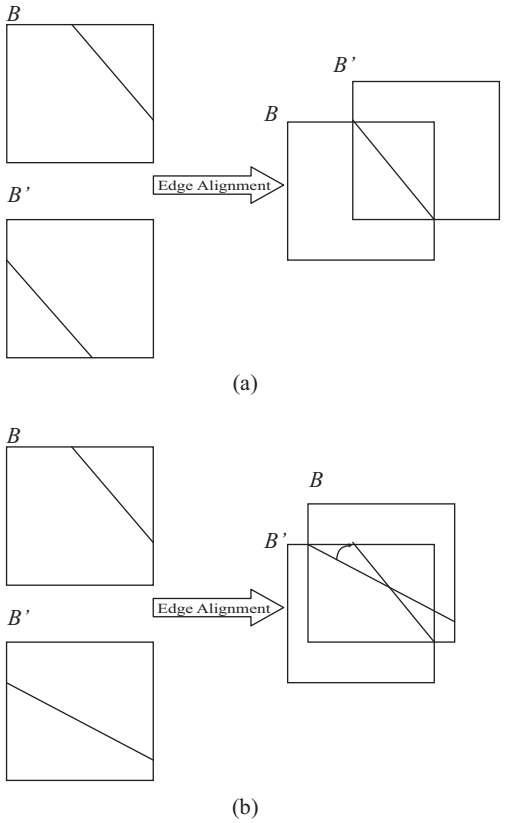

Fig. 2. The process of edge alignment is to transform the edge pattern of a block to that of the other by translation and rotation: (a) the edge orientations for blocks *B* and *B'* are the same; (b) the edge orientations for blocks *B* and *B'* are different.

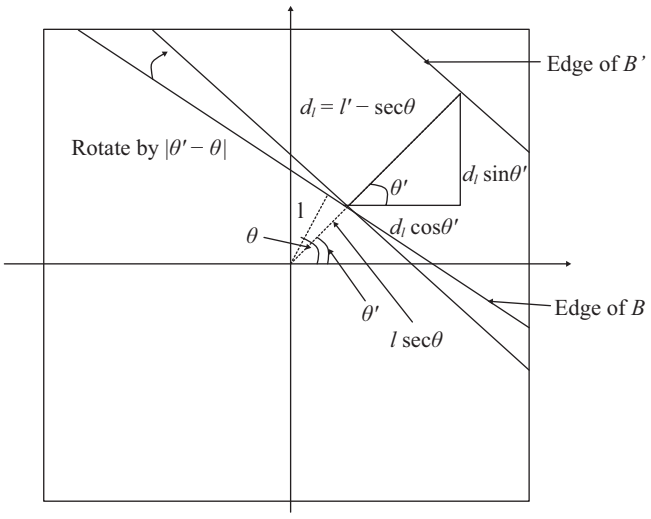

Fig. 3. Given two edge blocks *B* and *B'* charactering with the edge orientation and translation as (θ, *l*) and (θ', *l'*), respectively, the edge patterns of *B'* and *B* are aligned by rotating and translating the edge pattern of *B'*.

where $d_l = |l' - l\sec\theta|$ is the translation distance to align the edge pattern of *B* with that of *B'* when the centers of *B* and *B'* are coincided and *N* is the size of block. Combining (2) and (3), we obtain the final motion vector

$$\vec{u} = \vec{u}_1 + \vec{u}_2 . \tag{4}$$

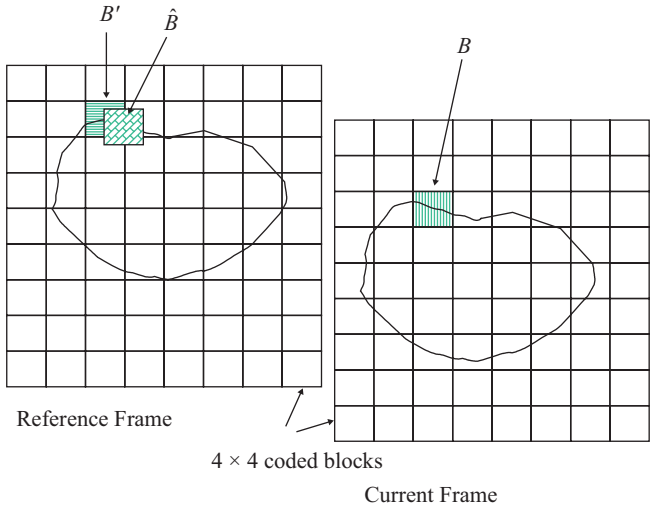

Fig. 4. The boundary of the matches $\hat{B}s$ for a given edge block *B* in the current frame might not be coincided with that of coded blocks in the reference frames.

Given a reference frame, the edge blocks play a role as the references to search the possible matches for each edge block of the current frame. For each edge block in the current frame, we scan the edge blocks within the search window from the reference image one-by-one in the fashion of left-to-right and up-to-down to find its possible best match. Let *B* be an edge block in the current frame. Given an edge block *B'* corresponding to one of the search point of *B* in the reference frame, we could locate the possible match block $\hat{B}$ in the reference frame for *B* using (4) when aligning the edge patterns of *B* and *B'*, shown in Fig. 4. The SAD value between *B* and $\hat{B}$ are then computed to judge the match degree. Assuming the size of searching window is 32 × 32, the number of search points to find the best match block is 8 × 8 for a 4 × 4 edge block using the proposed edge alignment technique. This dramatically reduces the number of check points using the full searching technique when doing block motion estimation.

In practice, it is not necessary to align the edge patterns for a pair of edge blocks of absolutely different orientations because these two blocks are impossible to be the best match with each other. The process of edge alignment can be further simplified if we assume the orientations for a match pair of blocks are very similar. Let the edge block $B = (h_1, h_2, \theta, l)$ centered at (x_B, y_B) in the current frame, and the edge block $\hat{B} = (\hat{h}_1, \hat{h}_2, \hat{\theta}, \hat{l})$ centered at $(x_{\hat{B}}, y_{\hat{B}})$ be the candidate best match of *B* in the reference frame. Suppose that *B'* is one of the regularly partitioned blocks in the reference frame nearby $\hat{B}$, $B' = (h_1', h_2', l', \theta')$ plays the role as the referencing point to

find the parameters of $\hat{B}$, shown in Fig. 4. Obviously, the orientation of $\hat{B}$ equals to that of B', that is $\hat{\theta} = \theta'$. Furthermore, we should have $\hat{l} = l$ for $\hat{B}$ to be one of the possible matches of B. The actual problem is: where should the center $(x_{\hat{B}}, y_{\hat{B}})$ of $\hat{B}$ be? Obviously, the location of $\hat{B}$ can be obtained by solving the following linear system

$$l' - \frac{x_{\hat{B}} - x_{B'}}{N}\cos\theta' - \frac{y_{\hat{B}} - y_{B'}}{N}\sin\theta' = l$$

$$y_{\hat{B}} - y_{B'} = \tan\theta' \times (x_{\hat{B}} - x_{B'}) \tag{5}$$

That is

$$\begin{bmatrix} x_{\hat{B}} \\ y_{\hat{B}} \end{bmatrix} = \begin{bmatrix} x_{B'} + N(l'-l)\cos\theta' \\ y_{B'} + N(l'-l)\sin\theta' \end{bmatrix} \tag{6}$$

Then, the displacement vector for B is easily obtained from

$$\vec{u} = (x_{\hat{B}} - x_B, y_{\hat{B}} - y_B) \tag{7}$$

if $\hat{B}$ is used to predict B.

To complete the discussion of edge alignment for block matching, we can not ignore the problem of fast block matching for uniform blocks. Before answering the problem, we first define the rule to decide the type of a block --an edge type or a uniform type. An image block B is defined to be an edge block if

$$| h_2 - h_1 | > \tau \tag{8}$$

where $| h_2 - h_1 |$ are the block contrast of B and τ is a predefined threshold. Obviously, many image blocks will be classified as edge blocks if we use a small value of τ. In this work, we set the value of τ to be a half of the average of the block contrasts from current frame in order to avoid resulting too many uniform blocks. However, we still have a lot of uniform blocks whose motion vectors are not obtainable through employing the process of edge alignment. As a matter of fact, for a small block size, the correlation between neighboring motion vectors is high. For each uniform block U, its motion vector $\vec{u}_U$ is simply predicted by the motion vectors of the edge blocks within the same macroblock as follows

$$\vec{u}_U = \sum_{B \in E} w_B \vec{u}_B, \sum_{B \in E} w_B = 1 \tag{9}$$

where E is the set of edge blocks of the macroblock containing U, $\vec{u}_B$ is the motion vector of the edge block B, and w_B is the weighting of B. The edge block which is far away from the uniform block U is supposed to give little impact on the motion vector of U. Thus, the value of w_B is computed by

$$w_B = a_B \Big/ \sum_{B' \in E} a_{B'}, a_B = 1 - d(B,U)/d_{\max} \tag{10}$$

where $d(B,U)$ is the Euclidean distance between B and U in terms of center coordinates and d_{max} is the maximal distance to U for all edge blocks in E. Similarly, we can interpolate the potential gradient orientation of the uniform block U by $\theta_U = \sum_{B \in E} w_B \theta_B$, which is further used to fine tune the motion vector.

The advantages of the proposed edge alignment for block matching are threefold: (1) it quickly determines the motion vectors with lesser SAD values for an image block in the current frame; (2) motion vectors of sub-pixel accuracy are obtained; (2) edge information which is visually important to human perception is preserved during video compression. Without loss of generality, suppose we have an ideal object of a single color moving on a uniform background. Given a block B on the boundary of the object in the current frame, the edge pattern of B can then be modeled as a step edge which classifies the pixel values of B into two classes –one of them is represented by h_1^B and the other is represented by h_2^B ($h_1^B < h_2^B$). Let $\hat{B}$ be the corresponding block of B in the reference frame characterized with two representative pixel values $h_1^{\hat{B}}$ and $h_2^{\hat{B}} (h_1^{\hat{B}} < h_2^{\hat{B}})$, too. Then, we have $h_1^B \approx h_1^{\hat{B}}$ and $h_2^B \approx h_2^{\hat{B}}$. As shown in Fig. 5(a), the value of SAD for B and $\hat{B}$ with their edge patterns aligned properly is

$$SAD_1 = n_1 | h_1^B - h_1^{\hat{B}} | + n_2 | h_2^B - h_2^{\hat{B}} | \tag{11}$$

where n_1 and n_2 are the numbers of pixels of B represented by h_1^B and h_2^B, respectively. Figures 5(b) and 5(c) show two non-edge alignment cases and their SAD values are

$$\begin{aligned} SAD_2 &= n_{11} | h_1^B - h_1^{\hat{B}} | + n_{12} | h_1^B - h_2^{\hat{B}} | \\ &\quad + n_2 | h_2^B - h_2^{\hat{B}} |, n_1 = n_{11} + n_{12}, \\ SAD_3 &= n_1 | h_1^B - h_1^{\hat{B}} | + n_{21} | h_2^B - h_1^{\hat{B}} | \\ &\quad + n_{22} | h_2^B - h_2^{\hat{B}} |, n_2 = n_{21} + n_{22}. \end{aligned} \tag{12}$$

Comparing (11) and (12), it is easy to prove that the value of SAD_1 is less than that of SAD_2 or SAD_3. Thus, the edge alignment process generates motion vectors with lesser SAD values.

In H.264, motion vectors are required to achieve sub-pixel accuracy and this results in high computational complexity in general. The line equation for an image block obtained by the moment-preserving edge detector is itself with the sub-pixel accuracy property, and then the motion vectors obtained through the process of the proposed edge alignment process meet the special requirement of H.264 without demanding additional computations.

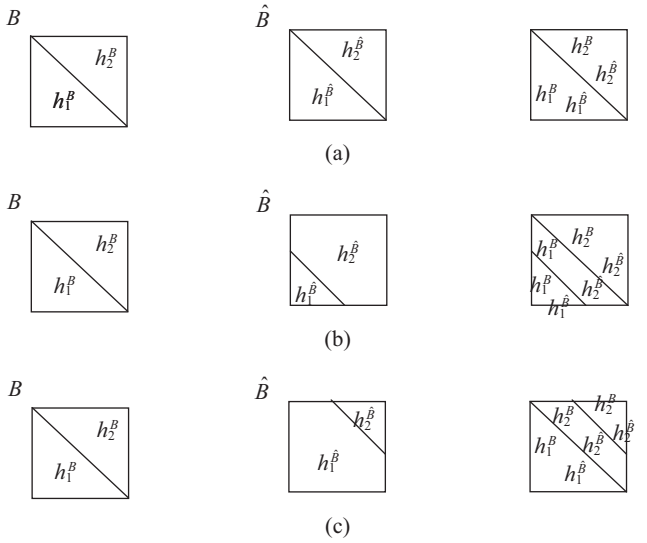

Fig. 5. Aligning two ideal step edges: (a) the edge patterns of *B* and $\hat{B}$ are aligned properly; (b) and (c) are two cases that do not align the edge patterns of *B* and $\hat{B}$ properly.

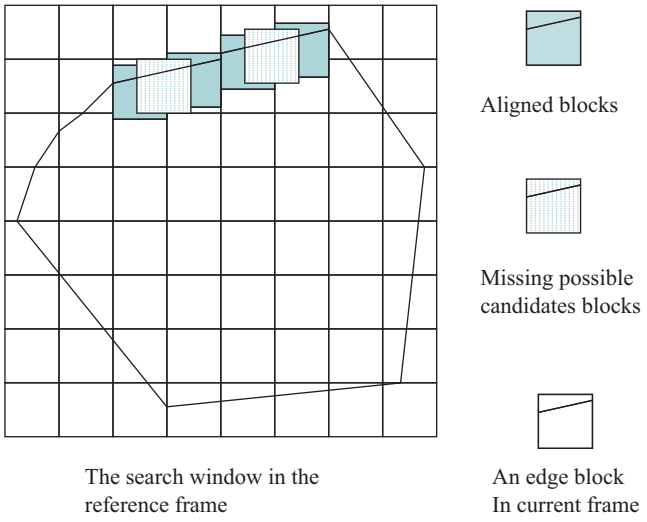

Fig. 6. Possible candidate blocks for the best match might be missing when doing block motion estimation using the proposed edge alignment strategy.

III. THE PROPOSED FAST BLOCK MOTION ESTIMATION

In combination of above analyses, given a block *B*, we propose a new searching process which predicts the motion vector of *B* through the process of edge alignment mentioned above. The edge orientation of *B* is used to adaptively select suitable hexagonal search pattern to accurately estimate the final motion vector of *B*.

1. Predictive Motion Estimation Using Edge Alignment

Given an edge block in the current frame, the predictive motion vector can be obtained by performing the proposed edge alignment process. For the sake of illustration, the edge alignment for block motion estimation is summarized as the following algorithm.

Algorithm. Edge Alignment for Motion Estimation.
Input: a 4 × 4 block *B*.
Output: the motion vector $\vec{u}_B$ of *B*.
Method:

(1) Perform the moment-preserving edge detection to detect the orientation θ and translation *l* of the edge pattern in *B*.
(2) If B is an edge block then
  (2.1) Initialize SAD_B as a very large value.
  (2.2) For each edge block B' with translation *l'* and orientation θ' within the search window of *B* in the reference frame do
    (2.2.1) If $|\theta - \theta'| < \xi$//ξ is a predefined threshold.
      (2.2.1.1) Use (6) to determine the location of $\hat{B}$ which is supposed to be the block whose edge pattern match well with that of *B*.
      (2.2.1.2) Compute the *SAD* value $SAD_{\hat{B}}$ between *B* and $\hat{B}$ using (1).
      (2.2.1.3) If $(SAD_{\hat{B}} < SAD_B)$ then $SAD_B = SAD_{\hat{B}}$ and $\vec{u}_B = (x_{\hat{B}} - x_B, y_{\hat{B}} - y_B)$ where (x_B, y_B) and $(x_{\hat{B}}, y_{\hat{B}})$ are the center coordinates of *B* and $\hat{B}$, respectively.
(3) If *B* is a uniform block, interpolate $\vec{u}_B$ based on the motion vectors of the edge blocks nearby *B* in the current frame using (9).

Actually, the algorithm is separated into two phases. In the first phase, we determine the motion vectors for all the 4 × 4 edge blocks in the current frame. Then, in the second phase, the motion vectors of remaindering uniform blocks are interpolated. These motion vectors cannot be the final motion vectors for further processing since we just align the edge patterns along the direction normal to the edge of *B*, shown in Fig. 4. Some possible candidate points for finding the best match might be missing if we do not consider aligning the blocks along the edge direction, shown in Fig. 6. Thus, incorporating the edge alignment process, an orthogonal search pattern, shown in Fig. 7, is added to fine tune the motion vector of a block in the current frame in order to obtain better image quality. In this work, we design a new search pattern to work as the motion vector fine tune procedure.

2. Synchronization Schemes

The predictive hexagon search (PHS) patterns are configured according to the types of blocks, which are obtained from the orientations of blocks. The predictive hexagon search

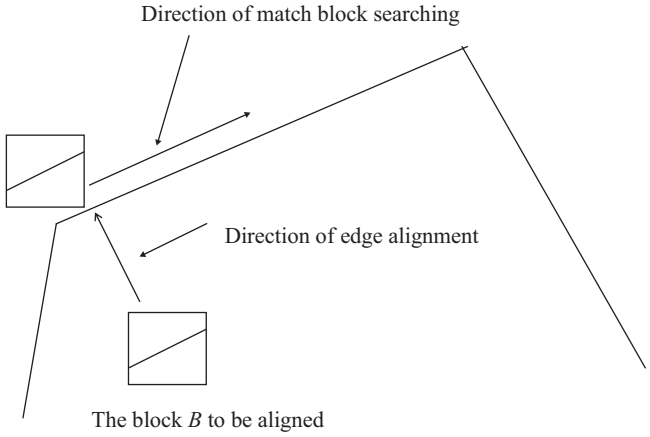

Fig. 7. An orthogonal block searching strategy: given a block *B* in the current frame, the initial motion vector is obtained using the proposed edge alignment process and then fine tune the motion vector by matching the blocks along the direction of edge pattern of *B*.

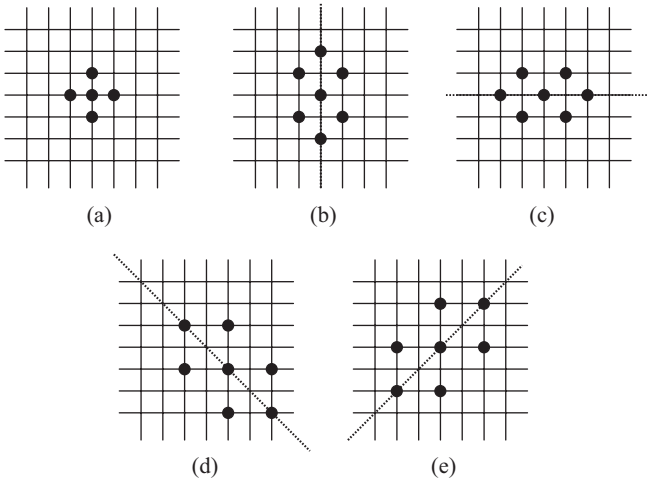

Fig. 8. Predictive hexagon search patterns: (a) diamond search pattern (DSP); (b) vertical edge search pattern (VESP); (c) horizontal edge search pattern (HESP); (d) diagonal edge search pattern (DESP); (e) anti-diagonal edge search pattern (ADESP).

pattern is shown in Fig. 8. Figure 8(a) shows a diamond search pattern (DSP) which contains five checking points (left, right, up, down dots with distance 1 around the center pot). DSP is first applied to start our searching flow. Considering the searching along edge directions of image blocks, four search patterns –vertical edge search pattern (VESP), horizontal edge search pattern (HESP), diagonal edge search pattern (DESP), and anti-diagonal edge search pattern (ADESP) shown in Figs. 8(b)-8(e), respectively, are used as the subsequent steps. In this paper, the block size 4 × 4 is adopted, which can be assumed to be small for high-resolution images. Instead of representing edges in any directions, the detected edges are mapping to 4 different directions. This assumes that the possible directions of an edge in a 4 × 4 block are limited to

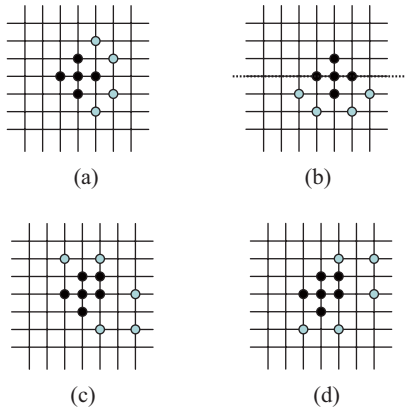

Fig. 9. Four cases for the first two steps of the block searching with (a) vertical edge type, (b) horizontal edge type, (c) diagonal edge type, and (d) anti-diagonal edge type.

multiples of 45°, or equivalently, $i \times 45°$, i = 0, 1, …, 3. If the actual direction of an edge is not a multiple of 45°, it is quantized to be the nearest multiple of 45°. After the direction of an edge is given, we use the corresponding search pattern to search the final best match block in the reference frames.

The purpose of the searching process is to find a search point with minimum rate-distortion cost (RD-Cost) which is defined as

$$J(\vec{u}) = SAD + \lambda \times Rate(\vec{u}) \tag{13}$$

where λ is a regulation parameter. Given an edge block B with the edge orientation θ in the current frame, the type of B is determined according to the following rule

$$B_{type} = \begin{cases} V, & if \quad 0^\circ \le \theta < 22.5^\circ \, or \, 157.5^\circ \le \theta < 180^\circ \\ D, & if \quad 22.5^\circ \le \theta < 67.5^\circ \\ H, & if \quad 67.5^\circ \le \theta < 112.5^\circ \\ A, & if \quad 112.5^\circ \le \theta < 157.5^\circ \end{cases} \tag{14}$$

where V, D, H, and A represent the vertical type, diagonal type, horizontal type, and anti-diagonal type, respectively. Figure 9 illustrates four cases to start the block searching along the edge directions for $B_{type} = V$, $B_{type} = H$, $B_{type} = D$, and $B_{type} = A$. For each case, the block searching procedure starts from applying the DSP which calculates the first 5 search points. The search pattern for the successive steps to be applied depends on the block type – VESP, HESP, DESP, and ADESP for $B_{type} = V$, $B_{type} = H$, $B_{type} = D$, and $B_{type} = A$, respectively. Suppose that the minimum RD-Cost point is one of the corner points in the first step, the minimum RD-Cost point will be the center point in next searching step. It would calculate 4 new points for $B_{type} \in \{V, H\}$, shown in Figs. 9(a) and 9(b) and 5 new points for $B_{type} \in \{D, A\}$, shown in Figs. 9(c) and 9(d). It

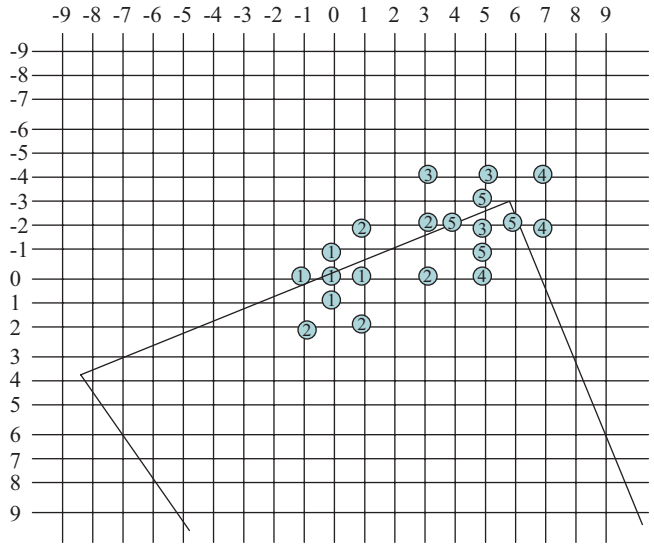

Fig. 10. Search path example for an image block with edge type *A* to find the motion vector (5, -2) in five steps.

is interesting to notice that no new points will be added to calculate the RD-Cost using the block searching procedure if the minimum RD-Cost point is located on the center point of the edge search patterns. In this case, we apply the DSP as the final step of the searching procedure. Figure 10 shows an example of the block searching procedure for demonstration. The performance of the searching procedure depends on the length of a line in the search window with its orientation similar to that of an input block.

The algorithm of the block searching procedure is summarized as follows.

Algorithm. Proposed Block Searching Procedure.

Input: a 4 × 4 block B with center coordinates $\vec{c}_B$ and the predictive motion vector $\vec{u}_B$.

Output: the final motion vector $\vec{u}_B$ of B.

Method:

(1) The DS with $\vec{c}_B + \vec{u}_B$ as the center point is used.

(2) If the minimum RD-Cost point is located on the center point of DS, then return the center point as the final point of the motion vector.

(3) Else do the following sub-steps:

(3.1) Decide B_{type} (the type of B) using (14).

(3.2) When the searching points are within the search window of B do

(3.2.1) In case $B_{type} = V$, perform the VESP with the minimum RD-Cost point in the previous step as the center point. If the minimum RD-Cost point is located on the center point of VESP, then go to Step (3.3), else go back to (3.2).

(3.2.2) In case $B_{type} = H$, perform the HESP with the minimum RD-Cost point in the previous step as the center point. If the minimum RD-Cost point is located on the center point of VESP, then go to Step (3.3), else go back to (3.2).

(3.2.3) In case $B_{type} = D$, perform the DESP with the minimum RD-Cost point in the previous step as the center point. If the minimum RD-Cost point is located on the center point of DESP, then go to Step (3.3), else go back to (3.2).

(3.2.4) In case $B_{type} = A$, perform the ADESP with the minimum RD-Cost point in the previous step as the center point. If the minimum RD-Cost point is located on the center point of ADESP, then go to Step (3.3), else go back to (3.2).

(3.3) Perform the DS with the minimum RD-Cost point in the previous step as the center point, and return the final minimum RD-Cost point as the final point of the motion vector.

3. Extension Method

At the beginning of mode decision in the reference software baseline encoder [13], SADs of the 16 4 × 4 blocks for a macroblock are calculated at all search points in all reference frames. These SAD values are then reused for other blocks of larger sizes because they share the same search window. In order to support fast algorithms for multiple frames motion estimation, for the first reference frame, we calculate the motion vectors of the 16 4 × 4 blocks for a macroblock first. Then, we predict the motion vector for other 6 types of block sizes. For other reference frames, we predict the motion vector as the motion vector of the same block size at previous reference frame.

Figure 11 shows the correlation between motion vectors of variable block sizes. The motion vectors of other blocks of larger sizes can be predicted using the motion vectors of the 16 4 × 4 blocks for a macroblock. For the block size 16 × 16, we use all the motion vectors of 4 × 4 blocks to calculate the predictive motion vector as expressed by

$$\vec{u}^{16\times16} = \frac{1}{16}\sum_{i=0}^{3}\sum_{j=0}^{3}\vec{u}_{ij}^{4\times4}\ . \tag{15}$$

For the upper (bottom) 16 × 8 block, we use 4 motion vectors of upper (bottom) two rows of 4 × 4 blocks for calculation as shown in Fig. 11. Similarly, for the left (right) 8 × 16 block, we use 4 motion vectors of left (right) two columns of 4 × 4 blocks for calculation. The 16 4 × 4 blocks are divided into four quarter parts, each of them consisting of 4 4 × 4 blocks is used to predict the motion vector for a 8 × 8 block. For each block of size 8 × 4 or 4 × 8, the motion vectors for the 2 4 × 4 blocks

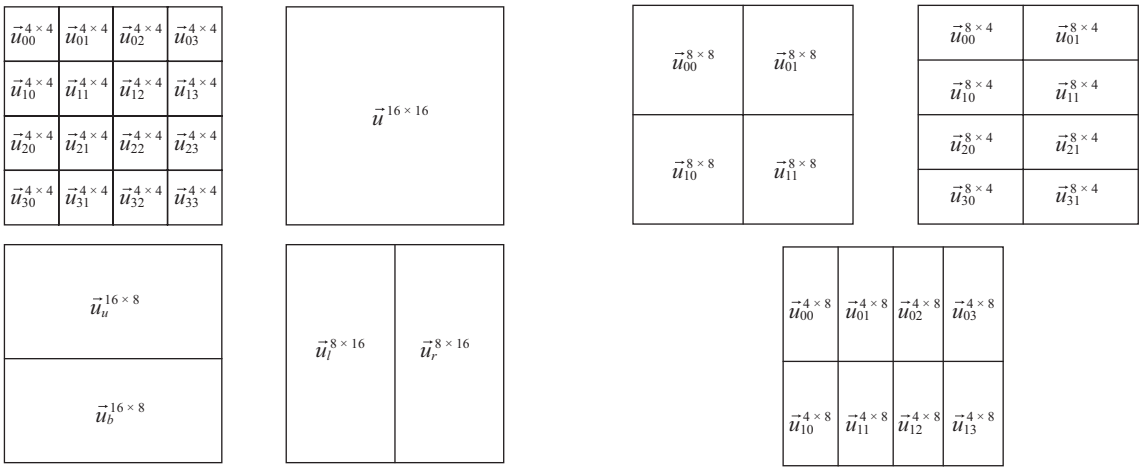

Fig. 11. The correlation between motion vectors of variable block sizes.

covered by the block are used for calculation as shown in Fig. 11. The equations are listed in (16)-(19) and expressed as:

$$\begin{cases} \vec{u}_u^{16\times8} = \dfrac{1}{8}\displaystyle\sum_{i=0}^{1}\sum_{j=0}^{3}\vec{u}_{ij}^{4\times4} \\ \vec{u}_b^{16\times8} = \dfrac{1}{8}\displaystyle\sum_{i=2}^{3}\sum_{j=0}^{3}\vec{u}_{ij}^{4\times4} \end{cases} \tag{16}$$

$$\begin{cases} \vec{u}_l^{8\times16} = \dfrac{1}{8}\displaystyle\sum_{i=0}^{3}\sum_{j=0}^{1}\vec{u}_{ij}^{4\times4} \\ \vec{u}_r^{8\times16} = \dfrac{1}{8}\displaystyle\sum_{i=0}^{3}\sum_{j=2}^{3}\vec{u}_{ij}^{4\times4} \end{cases} \tag{17}$$

$$\vec{u}_{ij}^{8\times8} = \frac{1}{4}\sum_{s=2i}^{2i+1}\sum_{t=2j}^{2j+1}\vec{u}_{st}^{4\times4},\ 0 \le i,\ j \le 1 \tag{18}$$

$$\begin{cases} \vec{u}_{ij}^{8\times4} = \dfrac{1}{2}\displaystyle\sum_{t=j}^{j+1}\vec{u}_{it}^{4\times4},\ 0 \le i \le 3,\ 0 \le j \le 1 \\ \vec{u}_{ij}^{4\times8} = \dfrac{1}{2}\displaystyle\sum_{s=i}^{i+1}\vec{u}_{it}^{4\times4},\ 0 \le i \le 1,\ 0 \le j \le 3 \end{cases} \tag{19}$$

In addition, we calculate the orientation for each block of larger size by averaging the orientation values of the 4 × 4 blocks covered by the block. Combining the predictive motion vectors listed in (15)-(19) and the proposed block search procedure, we obtain one motion vector for a 16 × 16 block, two motion vectors for 16 × 8 blocks, two motion vectors for 8 × 16 blocks, four motion vectors for 8 × 8 blocks, eight motion vectors for 8 × 4 blocks, eight motion vectors for 4 × 8 blocks, and 16 motion vectors for 4 × 4 blocks. The SAD values for all blocks of variable block sizes are also obtained.

IV. SIMULATION RESULT

In order to evaluate the proposed approach, a series of experiments was conducted on a 1.8 GHz PC with 960 MB main memory. For motion estimation, the search window is from -16 to 16, the number of reference frame is 5, and the number of block types is 7. The methods simulated for performance comparison with Full Search (FS) using seven test video sequences –'Container,' 'Foreman,' 'News,' 'Silent,' 'Paris,' 'Mobile,' and 'Tempete' include UMHEX [2], Xu and He's Method [26], Motion Classification-based Search (MCS) [7], and the proposed method. All the methods are implemented into the H.264/AVC reference software JM11.0 [13]. In our environment the UMHEX is adopted with the early termination here. All the test sequences are in QCIF format.

Table 1 illustrates the number of search points with different methods and different video sequences. For block-matching, the number of search points decides the computational complexity of motion estimation. Based on the simulation results of Table 1, except MCS, the number of search points by the proposed method is much smaller than the compared methods. The proposed edge alignment process is used to generate the predictive motion vector (PMV) quickly and the proposed predictive hexagon search (PHS) is used to further enhance the accuracy of the motion vector. We include the experimental results obtained from the proposed method with and without PHS to demonstrate the effectiveness of the edge alignment process. In our method, the information (h_1, h_2, *and* θ) for every block in the previous frames is properly indexed, which is used to locate the most similar block for a block in the current frame. Thus, the predictive motion vector based on the edge alignment process does not introduce additional search points. Figure 12 shows the performance comparison in terms of search points using the 7 test video sequences. Table 2 shows the average PSNR per frame. In order to have a fair comparison, the PSNR values for the reconstructed frames using the compared methods are almost the same. Accordingly, the proposed method has better capability in filtering unnecessary search points.

Table 1. The number of search points with different methods and different video sequences. The search range is from -16 to 16, the number of reference frame is 5, and the number of block types is 7.

	Container	Foreman	News	Silent	Paris	Mobile	Tempete
UMHEX [2]	118513207	140448962	125966243	136739720	550826115	620525963	628025257
Xu&He's method [26]	36739094	43539178	39049535	42389313	170756095	192363048	194687830
MCS [7]	7347819	25280813	15745780	19827259	76014004	108592044	72222905
Proposed	29318729	34797289	30636668	31907496	154862912	155006181	164159478

Table 2. Average PSNR per frame with different methods and different video sequences. The search range is from -16 to 16, the number of reference frame is 5, and the number of block types is 7.

	Container	Foreman	News	Silent	Paris	Mobile	Tempete
Full Search	36.03	35.54	36.48	35.71	34.92	33.19	33.68
UMHEX [2]	36.01	35.41	36.46	35.70	34.91	33.11	33.61
Xu & He's method [26]	35.99	35.41	36.45	35.71	34.90	33.10	33.60
MCS [7]	35.93	35.51	36.37	35.73	34.88	33.15	33.63
Proposed (without PHS)	34.90	34.57	35.31	34.96	34.41	33.16	33.53
Proposed	35.89	35.28	36.31	35.65	34.73	33.18	33.67

Table 3. Total execution time (ms) per video sequence with different methods and different video sequences. The search range is from -16 to 16, the number of reference frame is 5, and the number of block types is 7.

	Container	Foreman	News	Silent	Paris	Mobile	Tempete
Full Search	1842132	1994310	1549468	2454158	8795171	9715372	9283706
UMHEX [2]	399481	368286	380669	352663	2793489	3267793	3151528
Xu & He's method [26]	314032	284943	270503	262064	2035057	2616522	2209211
MCS [7]	282034	262588	304535	265435	2128639	2320133	2146191
Proposed (without PHS)	197736	248529	210508	231333	1892125	2120844	2106194
Proposed	263367	259730	239658	264434	2064770	2311310	2304615

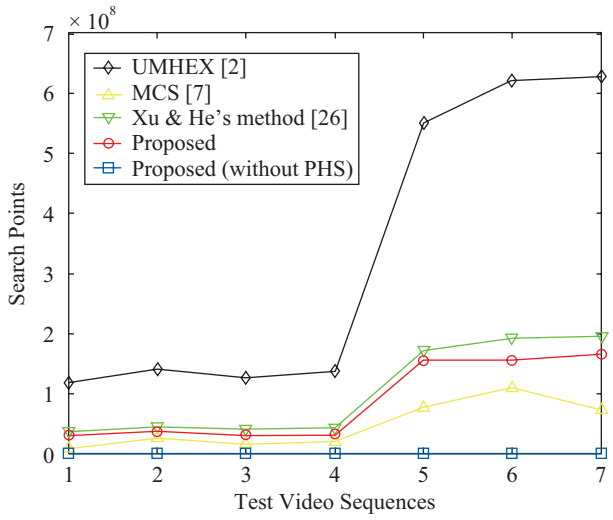

Fig. 12. Performance comparison in terms of average search points per macro block using 10 test video sequences.

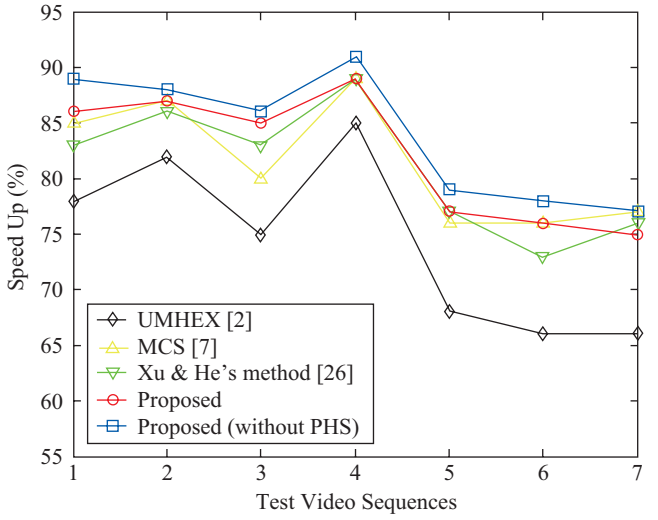

Fig. 13. Performance comparison in terms of speedup improvement using 10 test video sequences.

Table 3 shows the total execution time in terms of micro seconds (ms) to encode a video sequence with different methods and different video sequences. Figure 13 shows the performance comparison in terms of speedup improvement which is defined as

$$\text{Speedup}(A) = (1 - f(A)/f(\text{FSBMA})) * 100\%$$

where $f(A)$ is the execution time to encode a video sequence using method A. In average, the proposed method has 84% speedup improvement as compared with FS. However,

(a) (b) (c) (d) (e)

Fig. 14. Reconstructed frame 100 of 'Foreman': (a) original image; (b) FSBMA; (c) UMHEX; (d) proposed method (e) proposed method without PHS.

UMHEX, MCS and Xu and He's method have 74%, 82% and 81% speedup improvement, respectively. The proposed method is obviously faster than the compared methods.

The effectiveness of edge alignment affects the performance of the proposed video coding scheme. The quality of the reconstruction sequences (up to PSNR 40 db) for all compared methods is good. Figure 14 shows an example of reconstructed frame 100 for 'Forman' using FSBMA, UMHEX, the proposed method, and the proposed method without PHS. According to the experimental results, the proposed method achieves good performance when the input data contain obvious edges. In this case, the edge information extracted from the proposed moment-preserving edge detector is relatively accurate and produce a better predictive motion vector which reduces the complexity of further PHS operators. For example, the proposed method outperforms the compared methods using the test sequence 'Mobile' which contains many human-made objects. On the other hand, the performance of the proposed method is slightly degraded for ill-defined edges which can be founded in low-resolution video frames. The accuracy of edge detection affects the effectiveness of edge alignment. An improper edge alignment process leads to quality degradation in the proposed video coding scheme. Fortunately, this problem is not serious to our coding scheme since the color information in a block is also used to make sure the effectiveness of edge alignment. Although the proposed method has 0.1db PSNR degradation in the quality of reconstructed images, it does not produce noticeable artifacts in the reconstructed images. Hence, the proposed method meets the requirement of encoding high quality videos quickly. As shown in Fig. 14 (e), the reconstructed frame 100 of 'Foreman' using the proposed method without PHS is good and this demonstrates that the proposed edge alignment method has the benefits of the low complexity and still keeps good quality.

V. CONCLUSION

Motion estimation is the most critical part to realize H.264 which is the latest standard for video compression to support video applications with high video quality and low bit rates. In this paper we have presented a fast block matching for video coding based on edge alignment. A video encoding strategy based on the low-level computer vision also makes the coded results following the human perception without the main disadvantage of high-computational complexity for traditional block motion estimation. It applies some simple equations to calculate the predictive motion vector of a current block based on the concept of edge alignment. Then, considering the edge orientation of the block, a modified hexagon search is used to fine tune the motion vector in low computational complexity. The proposed algorithm also pays attentions to the characteristics of multiple reference frames and multiple block sizes in H.264. The proposed method is not only fast but also encoding video sequences in very high quality.

ACKNOWLEDGMENTS

This work was supported in part by National Science Council, Taiwan under Grant: NSC 96-2221-E-019-047-.

REFERENCES

1. Chen, C.-Y., Huang, Y.-W., Lee, C.-L., and Chen, L.-G., "One-pass computation-aware motion estimation with adaptive search strategy," *IEEE Transactions on Multimedia*, Vol. 8, No. 4, pp. 698-706 (2006).
2. Chen, Z., Zhou, P., and He, Y., "Fast integer pel and fractional pel motion estimation for AVC," presented at *the 6th JVT-Fo17 Meeting*, Awaji Island, Japan (2002).
3. Cheng, S.-C., "Content-based image retrieval using moment-preserving edge detection," *Image and Vision Computing*, Vol. 21, No. 9, pp. 809-826 (2003).
4. Cheng, S.-C. and Wu, T.-L., "Fast indexing method for image retrieval using k nearest neighbors searches by principal axis analysis," *Journal of Visual Communication and Image Representation*, Vol. 17, pp. 42-56 (2006).
5. *Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification*. ITU-T Rec. H.264 and ISO/IEC 14 496-10 AVC, Joint Video Team (2003).
6. Faloutsos, C., Ranganthan, M., and Manolopoulos, Y., "Fast subsequent matching in time-series databases," *Proceedings of the ACM SIGMOD International Conference on Management of Data*, Minneapolis, Minnesota, pp. 419-429 (1994).
7. Gonzalez-Diaz, I. and Diaz-de-Maria, F., "Adaptive multipattern fast block-matching algorithm based on motion classification techniques," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 18, No. 10, pp. 1369-1382 (2008).
8. Huang, Y. H., Hsieh, B. Y., Wang, T. C., Chen, S. Y., Ma, S. Y., Shen, C. F., and Chen, L. G., "Analysis and complexity reduction of multiple reference frames motion estimation in H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 16, No. 4, pp. 507-522 (2006).
9. *Information Technology—Coding of Audio-Visual Objects—Part 2: Visual*, ISO/IEC 14 496-2 (1999).

10. *Information Technology—Generic Coding of Moving Pictures and Associated Audio Information: Video*, ISO/IEC 13 818-2 and ITU-T Rec.H. 262 (1996).
11. Jain, A. K. and Dubes, R. C., *Algorithms for Clustering Data*, Prentice Hall, NJ (1988).
12. Joch, A., Kossentini, F., Schwarz, H., Wiegand, T., and Sullivan, G. J., "Performance comparison of video coding standards using lagragian coder control," *IEEE International Conference on Image Processing*, New York, pp. 501-504 (2002).
13. Joint Video Team Reference Software JM11.0 Aug. (2007) [Online]. Available: http://bs.hhi.de/~suehring/tml/download/.
14. Kim, T., "Side match and overlap match vector quantizers for images," *IEEE Transactions on Image Processing*, Vol. 1, No. 2, pp. 170-185 (1992).
15. Koga, T., Iinuma, K., Hirano, A., Iijima, Y., and Ishiguro, T., "Motion compensated interframe coding for video conferencing," *Proceeding of the National Telecommunications Conference*, New Orleans, LA, pp. C9.6.1-C9.6.5 (1981).
16. Lai, J. Z. C. and Liaw, Y.-C., "Fast searching algorithm for vector quantization using projection and triangular inequality," *IEEE Transactions on Image Processing*, Vol. 13, No. 12, pp. 1554-1558 (2004).
17. Lai, J. Z. C., Liaw, Y.-C., and Liu, J., "Fast k-nearest-neighbor search based on projection and triangular inequality," *Pattern Recognition*, Vol. 40, pp. 351-359 (2007).
18. McNames, J., "A fast nearest-neighbor algorithm based on a principal axis search tree," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 23, No. 9, pp. 964-976 (2001).
19. Murase, H. and Nayar, S. K., "Visual learning and recognition of 3D objects from appearance," *International Journal of Computer Vision*, Vol. 14, No. 1, pp. 5-24 (1995).
20. Tai, P.-L., Huang, S.-Y., Liu, C.-T., and Wang, J.-S., "Computation-aware scheme for software-based block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 9, pp. 901-913 (2003).
21. Theodridis, S. and Koutroumbas, K., *Pattern Recognition*, second Ed., Academic Press, Salt Lake City, USA (2003).
22. Tsai, T.-H. and Pan, Y. N., "A novel 3-D hexagon search algorithm for fast block motion estimation on H.264 video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 16, No. 12, pp. 1542-1549 (2006).
23. *Video Coding for Low Bit Rate Communication*, ITU-T Rec. H.263 (1998).
24. Wiegand, T., Sullivan, G. J., Bjntegaard, G., and Luthra, A., "Overview of the H.264/AVC video coding standard," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 13, No. 7, pp. 560-576 (2003).
25. Wu, D., Pan, F., Lim, K. P., Wu, S., Li, Z. G., Lin, X., Rahardja, S., and Ko, C. C., "Fast intermode decision in H.264/AVC video coding," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 15, No. 7, pp. 953-958 (2005).
26. Xu, X. and He, Y., "Improvements on fast motion estimation strategy for H.264/AVC," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 18, No. 3, pp. 285-293 (2008).
27. Zhu, C., Lin, X., and Chau, L. P., "Hexagon-based search pattern for fast block motion estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 12, No. 5, pp. 349-355 (2002).
28. Zhu, S. and Ma, K. K., "A new diamond search algorithm for fast block-matching motion estimation," *IEEE Transactions on Image Processing*, Vol. 9, No. 2, pp. 287-290 (2000).