Video Block Motion Estimation Based on Gray-Code Kernels

Yair Moshe, *Member, IEEE*, and Hagit Hel-Or, *Member, IEEE*

***Abstract*—Motion in modern video coders is estimated using a block matching algorithm that calculates the distance and direction of motion on a block-by-block basis. In this paper, a novel fast block-based motion estimation algorithm is proposed. This algorithm uses an efficient projection framework that bounds the distance between a template block and candidate blocks. Fast projection is performed using a family of highly efficient filter kernels—the gray-code kernels—requiring only 2 operations per pixel per kernel. The projection framework is combined with a rejection scheme which allows rapid rejection of candidate blocks that are distant from the template block. The tradeoff between computational complexity and quality of results can be easily controlled in the proposed algorithm; thus, it enables adaptivity to image content to further improve the results. Experiments show that the proposed adaptive algorithm outperforms other popular fast motion estimation algorithms.**

***Index Terms*—Block matching, gray-code kernels (GCK), motion estimation, video coding, Walsh–Hadamard transform (WHT).**

I. Introduction

VIDEO coders compress digital video sequences by removing redundancies. The most important redundancy—temporal redundancy—is typically reduced by motion estimation and motion compensation which encodes the differences between intensity values in the current frame and those of their counterparts in the reference frame that has been translated by an estimated motion vector. In most video coding standards, motion estimation is block based. A video frame is divided into nonoverlapping macroblocks, typically of size 16 $\times$ 16 pixels. Each macroblock is compared to candidate blocks within a search area in the reference frame. This process is referred to as the block matching algorithm (BMA). Various distortion measures could be used for finding the best match for a macroblock in the motion estimation process. Mean squared error (MSE), mean absolute error (MAE), and sum of absolute differences (SAD) are commonly used. Recently, a new distortion measure for motion estimation has been proposed—the sum of absolute transformed differences (SATD) [1]. This measure sums the frequency transform coefficients, typically the Hadamard transform, of the differences between the pixels in the template macroblock and the corresponding pixels in a candidate block. SATD is considerably slower than the SAD but it more accurately predicts quality from the viewpoints of both objective and subjective metrics. Therefore, it is used in the H.264 reference model software [2], as well as in other new video encoders.

Motion estimation, although efficient in reducing temporal redundancy, incurs high computational complexity. A full search technique for finding the best matching region within the search area in the reference frame is usually impractical for real-time applications due to the large number of comparisons required. Thus, many alternative "fast search" motion estimation algorithms have been proposed in the literature. The main concepts of these fast algorithms can be classified into six categories: reduction in search positions, predictive search, simplification of matching criterion, bitwidth reduction, hierarchical search, and fast full search [3]. The most popular category is the reduction in search positions. Algorithms in this category reduce search complexity by limiting the number of candidate blocks. These algorithms rely on the assumption that the matching error monotonically increases with the distance from the optimal position (having minimum distortion). This assumption is not always valid and the process may converge to a local minimum on the error surface rather than to the global minimum as in the full search algorithm. Well-known algorithms in this category are the 2-D logarithmic search [4], three-step search [5], four-step search [6], cross search [7], diamond search [8], and center-biased diamond search [9]. Diamond search based algorithms have significantly better performance in speed and quality than prior algorithms [3]. However, due to its simplicity, three-step search is still commonly used.

Predictive motion estimation, for example [10] and [11], utilizes the motion information in the spatial and/or temporal neighboring macroblocks to form an initial estimate of the current motion vector; thus, it can effectively reduce the search area as well as the computation. Another approach for fast motion estimation is to speed up the calculation of matching error for each candidate block independently. This is usually achieved by subsampling the pixels in the template and candidate blocks [12], [13]. Finding the optimal match with minimum matching error using this technique is, however, not guaranteed. This approach may be combined with the former two techniques to limit the number of search positions and to predict the current motion vector.

Manuscript received October 17, 2008; revised May 18, 2009. First published June 16, 2009; current version published September 10, 2009. This work was supported in part by the Israeli Ministry of Science and by a grant from A.M.N. fund for the promotion of science, culture and arts in Israel. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Srdjan Stankovic.

The authors are with the Computer Science Department, University of Haifa, 31905, Haifa, Israel (e-mail: yair@ee.technion.ac.il; hagit@cs.haifa.ac.il).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIP.2009.2025559

A hierarchical search uses a multiresolution pyramid structure. Conventional block matching (either full search or any other fast search method), is first applied to the highest level of the pyramid. The detected motion vector is further refined iteratively in the lower levels [14]. Similar to the previously described approach, this technique may also converge to a local minimum. In spite of this fact, it has been regarded as one of the most efficient methods for motion estimation with very large frames and search areas.

A different approach for fast motion estimation uses simple matching criteria to reject search positions while ensuring the global minimal matching error can still be attained. Only candidate blocks that have not been disqualified are further processed using more precise distortion calculations. Using an appropriate test, many search positions may be excluded from being further considered in the motion vector search, thus reducing search complexity significantly. Some well-known examples of this approach are the successive elimination algorithm [15], [16] and the block sum pyramid [17]. An improvement of the block sum pyramid based on a winner-update strategy is presented in [18].

Orthogonal transforms have also been shown to be useful for block motion estimation. However, only very few algorithms using the Walsh–Hadamard transform (WHT) for block motion estimation have been proposed in the literature. In [19] a hierarchical motion estimation algorithm is proposed in which the SAD of the WHT coefficients is used as a distortion measure of four search levels. In [20] a fast full search algorithm using the MSE is proposed. The MSE is calculated using the WHT coefficients with the low frequency coefficients considered first. An early termination criterion based on the successive elimination algorithm [15] is used for early exclusion of inappropriate candidates. Efficient calculation of the transform coefficients is performed based on the overlapping nature of search regions. This approach is not generalized and suffers from scalar constants that degrade the overall algorithm performance significantly. Recently, two fast motion estimation algorithms using the WHT have been proposed [21], [22]. Both algorithms considerably benefit from the addition of a standard predictive motion estimation scheme; however, these studies use the projection framework described in [23] and [24] (which was later shown to be less efficient than the fast projection technique described in [25] and [26]).

In this paper, a novel fast block-based motion estimation algorithm is described. This algorithm uses the WHT coefficients as a special case of an efficient projection framework that bounds the distance between macroblocks and their corresponding candidate blocks. A family of highly efficient filter kernels—the gray-code kernels (GCK)—is used for projection using only 2 operations per pixel per kernel. The projection framework is combined with a rejection scheme which allows rapid rejection of candidate blocks that are distant from the template macroblock. The algorithm enables adaptivity to image content in order to tune the tradeoff between computational complexity and quality of results and to further improve the results. An initial prototype of the proposed algorithm was first described in [27].

This paper is organized as follows: Fast pattern matching algorithms using WH projection kernels and GCK are first described in Section II and Section III, respectively. The proposed fast block motion estimation algorithm, based on these fast pattern matching algorithms, is presented in Section IV. Complexity analysis and results are given in Section V and Section VI, respectively. The proposed algorithm is further refined for adaptivity in Section VII. Finally, conclusions are drawn in Section VIII.

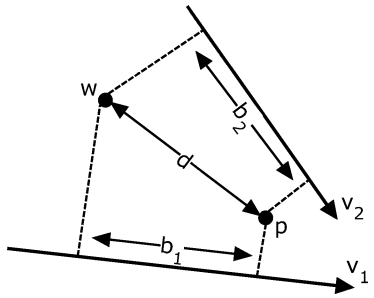


Fig. 1. Projection of $p - w$ onto vector v produces a lower bound on distance $\|d\| = \|p - w\|$.

II. Fast Pattern Matching Using Walsh–Hadamard Projection Kernels

The block motion estimation approach suggested in this paper is based on a novel pattern matching technique, introduced in [23], [24] and briefly reviewed here. The suggested approach uses an efficient WH projection scheme which bounds the distance between a pattern and an image window using very few operations on average. The projection framework is combined with a rejection scheme which allows rapid rejection of image windows that are distant from the pattern.

The pattern matching problem involves finding a particular pattern in an image where the pattern is usually much smaller than the image. This can be performed naively by scanning the entire image and evaluating the similarity between the pattern and a local 2-D window about each pixel. Assume a 2-D $k \times k$ pattern p is to be matched within an image I of size $n \times n$. For each pixel location (x, y) in the image, the Euclidean distance may be calculated

$$d_E^2(p, I_{x,y}) = \sum_{\{i,j\}=0}^{k-1} \left(I(x+i, y+j) - p(i,j)\right)^2 \qquad (1)$$

where $I_{x,y}$ denotes a local window of I at coordinates (x, y). In the context of motion estimation, this procedure is equivalent to full search block matching of a $k \times k$ template block to a set of candidate blocks in a search area of size $n \times n$ with the MSE criterion. Referring to the pattern p and window w as vectors in $\mathcal{R}^{k^2}$, $d = p - w$ is the difference vector between p and w. The Euclidean distance can then be rewritten in vectorial form

$$d_E(p, w) = \|d\| = \sqrt{d^T d}. \qquad (2)$$

Now assume, as illustrated in Fig. 1, that p and w are not given but only the values of their projection onto a vector v. Let

$$z = v^T d = v^T p - v^T w \qquad (3)$$

be the projected distance value. Since the Euclidean distance is a norm, it follows from the Cauchy–Schwartz inequality that a lower bound on the actual Euclidean distance can be inferred from the projection values. Using Cauchy–Schwartz inequality for norms, it follows that:

$$||v||||d|| \geq ||v^T d||. \tag{4}$$

This implies

$$d_E(p,w) = ||p-w|| = ||d|| \geq \frac{\left|\left|v^T(p-w)\right|\right|}{||v||} = \frac{||v^T p - v^T w||}{||v||} \tag{5}$$

and

$$d_E^2(p,w) \geq z^2/||v||^2. \tag{6}$$

If a collection of projection vectors are given $v_1 \ldots v_m$ along with the corresponding projected distance values $z_i = v_i^T d$, the lower bound on the distance can then be tightened

$$d_E^2(p,w) \geq Z^T (V^T V)^{-1} Z = LB_m^2(p,w) \tag{7}$$

where $V = [v_1 \ldots v_m]$ and $Z = (z_1 \ldots z_m)^T$ so that $Z = V^T d$. As the number of projection vectors increases, the lower bound on the distance $d_E(p,w)$ becomes tighter. In the extreme case when the rank of $\mathbf{V}$ equals k^2, the lower bound reaches the Euclidean distance. An iterative scheme for calculating the lower bound is also possible. Given an additional projection vector v_{m+1} and projection value z_{m+1}, the previously computed lower bound can be updated without recalculating the inverse of the entire system $(V^T V)^{-1}$ (see [24] for details). If the m projection vectors are orthonormal, the distance lower bound after m projections is reduced to

$$LB_m^2(p,w) = Z^T (V^T V)^{-1} Z = Z^T Z. \tag{8}$$

Returning to pattern matching, a window can be determined as being dissimilar to the pattern if the lower bound is above a certain threshold. Windows can be rejected as nonpatterns without actually computing the true distance. In the context of this problem, since lower bounds are only compared to each other, the actual value of the lower bound is irrelevant. Thus, even if the projection vectors are orthogonal and not orthonormal, the normalizing factor in the lower bound calculation can be discarded. In order for this approach to be efficient, vectors should be chosen according to the following two necessary requirements.

- The projection vectors should be highly probable of being parallel to the vector $d = p - w$.
- Projections of image windows onto the projection vectors should be fast to compute.

The first requirement implies that, on average, the first few projection vectors produce a tight lower bound on the pattern-window distance. This, in turn, will allow rapid rejection of image windows that are distant from the pattern. The second requirement arises from the fact that the projection calculations are performed many times for each window of the image. Thus, the complexity of calculating the projection plays an important role when choosing the appropriate projection vectors.

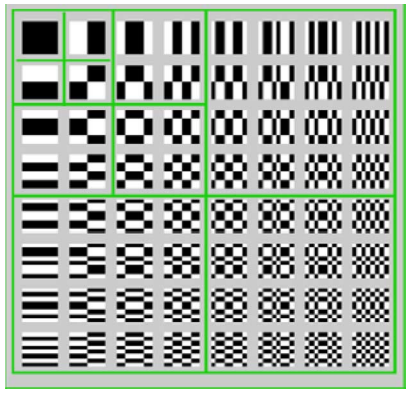


Fig. 2. Projection vectors of the WHT of order $k = 8$. Projection vectors are ordered with increasing spatial sequency. White represents the value $+1$ and black represents the value -1.

A set of projection vectors shown in [23] and [24] to satisfy the above two requirements are the WH basis vectors. For natural images, these vectors, ordered in increasing sequency (the number of sign changes along rows and columns of the basis vector), capture a large portion of the pattern-window distance with few projections on average. In addition, an efficient method for calculating the projection values for these vectors is presented in the next section. The WHT has long been used for image representation under numerous applications [28]. The elements of the WH (nonnormalized) basis vectors are orthogonal and contain only binary values (± 1). Thus, computation of the transform requires only integer additions and subtractions. The WHT of an image window of size $k \times k$ (with k a power of 2) is obtained by projecting the window onto k^2 WH basis vectors. In the case of pattern matching within an image, it is required to project each $k \times k$ window of an $n \times n$ image onto the vectors. This results in a highly overcomplete image representation. The projection vectors associated with the 2-D WHT of order $k = 8$ are displayed in order of increasing sequency in Fig. 2. Each basis vector is of size 8×8. White represents the value $+1$ and black represents the value -1.

Finally, we note that the projection approach described above was introduced for the Euclidean distance; however, it is applicable to any distance measure that forms a norm. The correctness of the iterative scheme for norm-2 is proven in [23], [24]. However, in this paper, the iterative projection scheme is used with the well-known SAD (norm-1) distance measure. This is applicable since as additional projections are performed the lower bound on the SATD is tightened. In [22], the correctness of the iterative projection scheme with the SAD as the distance measure is proven.

III. Gray-Code Kernels

In [25] and [26], a family of filter kernels—the GCK—is introduced. Filtering an image with a sequence of GCK is highly efficient and requires only 2 operations per pixel for each filter kernel, independent of the size or dimension of the kernel. This

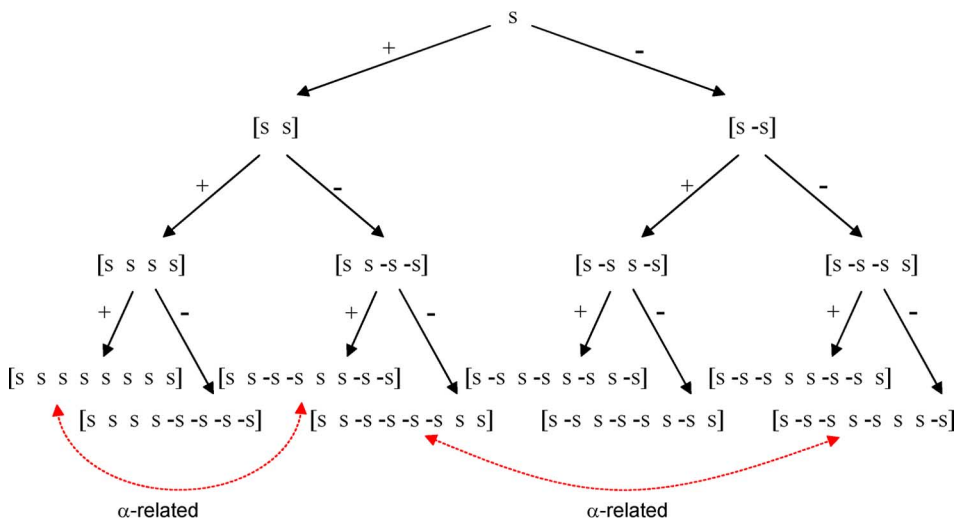


Fig. 3. Set of Gray Code Kernels and their recursive definition visualized as a binary tree. In this example, the tree is of depth $k = 3$ and creates $2^3 = 8$ kernels of length 8. Arrows indicate examples of pairs of kernels that are α-related.

family of kernels includes the WH kernels among others; thus, it enables very efficient projection onto the WH basis vectors.

Consider first the 1-D case where signal and kernels are 1-D vectors. Denote by $V_s^{(k)}$ a set of 1-D filter kernels expanded recursively from an initial seed vector s, as follows:

$$\begin{aligned} V_s^{(0)} &= s \\ V_s^{(k)} &= \left\{ \left[v_s^{(k-1)} \quad \alpha_k v_s^{(k-1)} \right] \right\} \quad s.t. \\ v_s^{(k-1)} &\in V_s^{(k-1)}, \alpha_k \in \{+1 \quad -1\} \end{aligned} \tag{9}$$

where $\alpha_k v$ indicates the multiplication of kernel v by the value α_k and $[\ldots]$ denotes concatenation. The set of kernels and the recursive definition can be visualized as a binary tree of depth k. An example is shown in Fig. 3 for $k = 3$. The nodes of the binary tree at level i represent the kernels of $V_s^{(i)}$. The leaves of the tree represent the eight kernels of $V_s^{(3)}$. The branches are marked with the values of α used to create the kernels (where $+/-$ indicates $+1/-1$).

Denote $|s| = y$ the length of s. It is easily shown that $V_s^{(k)}$ is an orthogonal set of 2^k kernels of length $2^k t$. Furthermore, given an orthogonal set of seed vectors $s_1 \ldots s_n$, it can be shown that the union set $V_{s_1}^{(k)} \bigcup \cdots \bigcup V_{s_n}^{(k)}$ is orthogonal with $2^k n$ vectors of length $2^k t$. If $n = t$ the set forms a basis. Fig. 3 also demonstrates the fact that the values $\alpha_1 \ldots \alpha_n$ along the tree branches uniquely define a kernel in $V_s^{(k)}$. The sequence $\alpha = \alpha_1 \ldots \alpha_n$, $\alpha_i \in \{+1, -1\}$ that uniquely defines a kernel $v \in V_s^{(k)}$ is called the *α-index* of v. Two kernels $v_i, v_j \in V_s^{(k)}$ are defined to be *α-related* if and only if the hamming distance between their α-index (the number of positions for which their α-indices differ) is one. Without loss of generality, let the α-indices of two α-related kernels be $(\alpha_1 \ldots \alpha_{r-1}, +1, \ldots \alpha_k)$ and $(\alpha_1 \ldots \alpha_{r-1}, -1, \ldots \alpha_k)$. We denote the corresponding kernels as v_+ and v_-, respectively. Since $\alpha_1 \ldots_{r-1}$ uniquely defines a kernel in $V_s^{(r-1)}$, two α-related kernels always share the same prefix vector of length $2^{r-1} t = \Delta$. The arrows of Fig. 3 indicate examples of α-related kernels in the binary tree of depth $k = 3$.

Of special interest are sequences of kernels that are consecutively α-related. An ordered set of kernels $v_1 \ldots v_n \in V_s^{(k)}$ that are consecutively α-related form a sequence of GCK. The sequence is called a *gray-code sequence* (GCS). The term gray code relates to the fact that the series of α-indices associated with a GCS form a gray code [29]. The kernels at the leaves of the tree in Fig. 5 in a left to right scan, are consecutively α-related, thus forming a GCS. Note, however, that this sequence is not unique and that there are many possible ways of reordering the kernels to form a GCS. The main idea presented in [25] and [26] relies on the fact that two α-related kernels share a special relationship. Given two α-related kernels $v_+, v_- \in V_s^{(k)}$, then

$$\begin{aligned} v_+(i) &= +v_+(i-\Delta) + v_-(i) + v_-(i-\Delta) \\ v_-(i) &= -v_-(i-\Delta) + v_+(i) - v_+(i-\Delta). \end{aligned} \tag{10}$$

Equation (10) is the core principle behind an efficient filtering scheme. Let b_+ and b_- be the signals resulting from convolving a signal x with filter kernels v_+ and v_-, respectively. Then, by linearity of the convolution operation and by (10), we have the following:

$$\begin{aligned} b_+(i) &= +b_+(i-\Delta) + b_-(i) + b_-(i-\Delta) \\ b_-(i) &= -b_-(i-\Delta) + b_+(i) - b_+(i-\Delta). \end{aligned} \tag{11}$$

This forms the basis of an efficient scheme for convolving a signal with a set of GCK. Given the result b_+ (b_-) of convolving the signal with the filter kernel v_+ (v_-), the result b_- (b_+) of convolving with the filter kernel v_- (v_+) requires only two operations per pixel independent of the kernel size. This scheme is illustrated in Fig. 4.

Considering definition (9), and setting the prefix string to $s = [1]$, we obtain that $V_s^{(k)}$ is the WH basis set of order 2^k. A binary tree can be designed such that its leaves are the WH kernels ordered in dyadic (or Paley) order [28] of increasing sequency and they form a GCS (i.e., are consecutively α-related). An example for $k = 2$ is shown in Fig. 5 where every two consecutive kernels are α-related. Thus, given the result of filtering an image with the first WH kernel, filtering with the consecutive

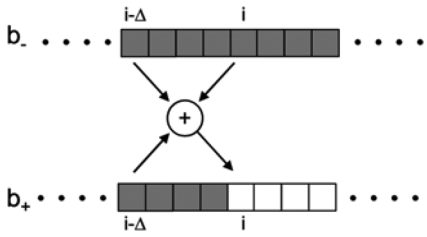


Fig. 4. Efficient filtering using GCK. Given b_- (convolution of a signal with the filter kernel v_-), the convolution result b_+ can be computed using 2 operations per pixel regardless of kernel size. In this example: $b_+(i) = b_+(i-\Delta) + b_-(i) + b_-(i-\Delta)$.

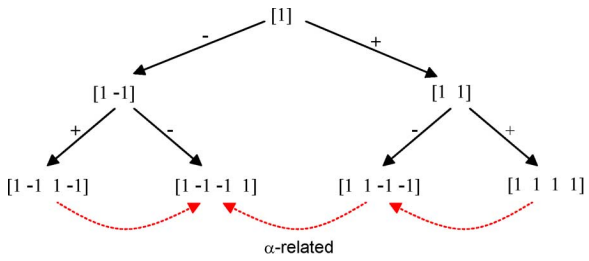


Fig. 5. GCK with initial vector $s = [1]$ creates the WH basis set. Using initial vector $s = [1]$ and depth $k = 2$, a binary tree creates the WH basis set of order 4. Consecutive kernels are α-related, as shown by the arrows.

kernels can be performed using only 2 operations per pixel per kernel regardless of signal and kernel size. For separable kernels, such as the WHT, the previous definitions and results can be generalized to two (and higher) dimensions. The computation cost remains at two operations per pixel per kernel regardless of the dimension. For more details, the reader is referred to [26].

It was shown that successive filtering with α-related kernels can be applied efficiently. However, the efficiency of using the GCK in a particular application is determined not only by the computational complexity of applying each kernel, but also by the total number of kernels taking part in the process. This, in turn, depends upon the order in which the kernels are applied. In the context of pattern matching, ordering the 2-D WH kernels in order of increasing sequency (the number of sign changes along each dimension of the kernel - analogous to frequency), is known to perform well on natural images due to energy compaction in the low order sequencies [30]. However, consecutive kernels in the 2-D WH sequency order are not necessarily α-related; thus, they do not form a GCS. Fortunately, horizontally or vertically neighboring kernels in the 2-D WH array are α-related, so "snake" ordering is possible, as depicted by overlaid arrows in Fig. 6. The "snake" ordering forms a GCS and, although not exactly according to sequency, captures the increase in spatial frequency.

The linear "snake" forms a GCS, thus filtering with any kernel in the snake-ordered sequence requires maintaining the projection onto the previous kernel in the sequence. However, by allowing "nonlinear orders," it is possible to select an ordering of kernels such that consecutive kernels are not necessarily α-related rather every kernel is α-related to a kernel that precedes it anywhere in the sequence. This still allows an efficient projection scheme, but incurs higher memory complexity since several preceding projections must be maintained in memory. One

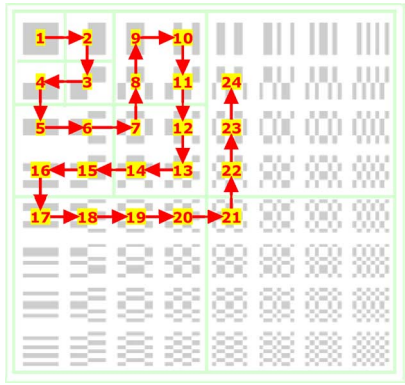


Fig. 6. "Snake" ordering of WH kernels. The projection vectors of the WHT of order $n = 8$. "Snake" ordering is depicted by overlaid arrows and numbers.

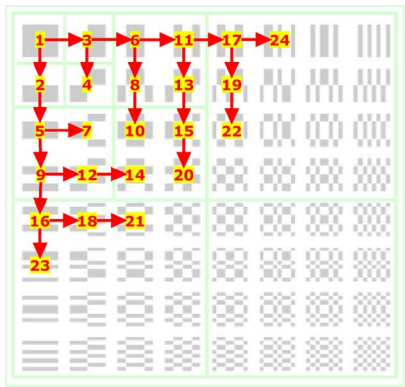


Fig. 7. Increasing frequency ordering of WH kernels. The projection vectors of the WHT of order $n = 8$. Increasing frequency ordering is depicted by overlaid arrows and numbers.

such ordering is the nonlinear "increasing frequency" ordering depicted in Fig. 7. The kernels in this order are arranged in increasing spatial frequency, that has better energy compaction in the first kernels compared to "snake" ordering [31]. In the algorithm presented in the next section, memory complexity is not an issue; thus, the nonlinear increasing frequency order is used.

IV. FME-GCK Algorithm

We propose a novel fast block motion estimation algorithm—FME-GCK—based on the fast pattern matching technique described above. The FME-GCK algorithm inherits all the advantages of the fast pattern matching technique described above as well as exploiting additional redundancies inherent to the block motion estimation process. The proposed scheme is fast and efficient, involves only integer computations and uses sequential memory access. In contrast with classical motion estimation algorithms, the FME-GCK enables online adaptivity to image content (see Section VII). In block motion estimation, every block of the reference frame forms a matching candidate for several neighboring macroblocks in the current frame. Additionally, the current frame forms the reference frame of the consecutive frame in the video sequence. The FME-GCK exploits these additional redundancies in the block motion estimation process.

Assume a video sequence is composed of images $I_0, I_1, I_2, \ldots$ of size $n_1 \times n_2$. Further assume that macroblocks are of size $k \times k$, and search areas are of size $n \times n$. Also assume that a set of m WH basis vectors $\{v_i\}_{i=0}^{m-1}$ is given such that every basis vector is α-related to at least one basis vector that precedes it in the sequence. Denote by $b_i^{(j)}$ the array of projection values of all blocks of image I_j onto the i-th WH basis vector v_i. Denote by $p_{x1,y1}^{(j)}$ the $k \times k$ macroblock of image I_j located at coordinates $(x1, y1)$, and denote by $w_{x2,y2}^{(j-1)}$ the candidate region of size $k \times k$ located at coordinates $(x2, y2)$ in image I_{j-1}. Denote by $SA(p)$ the set of all candidate regions inside the search area around macroblock p in the preceding image. The FME-GCK operates as follows. First, each image is projected onto the m WH basis vectors $\{v_i\}_{i=0}^{m-1}$ using the fast GCK scheme. The resulting projections are stored in memory. For each macroblock, candidate blocks in the appropriate search area are tested. This is performed by calculating the norm-1 lower bound on the distance between the template macroblock and each candidate using the stored projection values and (8). The q candidate blocks with minimal lower bound are selected and the actual SAD values between these candidates and the template macroblock are calculated. The candidate block with the minimal SAD is selected as the best matching block.

The FME-GCK algorithm

For each image I_j

1) Project I_j onto $\{v_i\}_{i=0}^{m-1}$ to obtain $\{b_i^{(j)}\}_{i=0}^{m-1}$ and store the resulting projections in memory.
2) For each macroblock $p_{x1,y1}^{(j)}$
   - 2.1) For each candidate block $w_{x2,y2}^{(j-1)} \in SA(p_{x1,y1}^{(j)})$
     - 2.1.1) Calculate the norm-1 lower bound on the distance between $p_{x1,y1}^{(j)}$ and $w_{x2,y2}^{(j-1)}$ using $\{b_i^{(j)}\}_{i=0}^{m-1}$ and $\{b_i^{(j-1)}\}_{i=0}^{m-1}$ and (8).
   - 2.2) Select the q candidate blocks $\{w_{x2,y2}^{(j-1)}\}_{i=0}^{q-1}$ with the minimal lower bound. Calculate the actual SAD between them and $p_{x1,y1}^{(j)}$.
   - 2.3) Of the q candidate blocks $\{w_{x2,y2}^{(j-1)}\}_{i=0}^{q-1}$, select that with the minimal SAD as the best matching block.

A block diagram of the FME-GCK algorithm is shown in Fig. 8. Note that image projections (step 1 in the algorithm), is performed on both Inter and Intra frames, whereas the remaining steps are applied only on Inter frames, where motion information is required. Image projections are stored in memory since they are required for motion estimation of the following frame in the video sequence.

In order to perform efficient GCK calculations, each basis vector should be α-related to at least one basis vector that precedes it in the sequence (from within the projection values stored in memory). The order of kernels used within the FME-GCK is depicted in Fig. 7 as overlaid arrows. This "increasing frequency" ordering has been chosen due to its good energy compaction property [31]. Step 1 of the algorithm is performed using GCK with only 2 operations per pixel for each WH kernel. An exception in this efficient calculation is the first kernel (DC component) which has no preceding kernel in the sequence. The DC component can be calculated using 4 operations per pixel as described in [30].

Notice that the GCK approach cannot be used efficiently for projecting macroblocks on the top and left image boundaries due to the necessity of accessing preceding pixels Δ units away in either dimensions [see (11)]. This limitation, although seemingly minor, may increase algorithm complexity substantially. In an experiment with the Foreman video sequence at CIF (352×288) resolution, boundary macroblock projections were performed by direct filtering with WH basis vectors and non-boundary macroblock projections were performed using GCK filtering. At CIF resolution, only about 0.7% of the candidate regions are top or left boundary blocks, and yet projections of these blocks were found to require about 55% of the actual calculation time spent on projections. A solution to this problem is to zero-pad the upper and left boundaries of the image with $\Delta + k - 1$ rows and $\Delta + k - 1$ columns, respectively. The size of the projection images $\{b_i^{(j)}\}_{i=0}^{m-1}$ increases accordingly. However, the upper Δ rows and left columns of these images $\{b_i^{(j)}\}_{i=0}^{m-1}$ can be shown to equal zero (since projecting a zero macroblock onto any kernel results in zero). The projection of image blocks using the efficient GCK method is initiated starting from the $\Delta + 1$ row and $\Delta + 1$ column.

Step 2.1.1 of the algorithm is based on the projection framework described in Section II. Although the WH basis vectors are not orthonormal, they are orthogonal. Therefore, the term $(V^T V)^{-1}$ in (7) can be ignored. The projection scheme is used with norm-1 rather than norm-2 since this forms a partial calculation of the SATD distance measure shown to be effective in block matching [32]. As additional projections are applied, a better approximation of the SATD is obtained [22].

The FME-GCK algorithm gives good time-quality tradeoff compared to classical fast block motion estimation techniques. Usually, only a few projections are required for highly accurate motion estimation. However, if $m = k^2$ the algorithm results are guaranteed to be identical to that of full search, though this is not a common configuration. In this sense, FME-GCK can be considered an approximation of a fast full search motion estimation algorithm.

V. Complexity Analysis

The FME-GCK algorithm uses two parameters that affect the tradeoff between complexity and accuracy of motion estimation. These parameters are m, the number of projections to perform for each image, and q, the number of candidate macroblocks for which the SAD value is calculated. Larger m produces more accurate results at the cost of higher time and memory complexity. Memory complexity is affected since m projections of image I_j and m projections of image I_{j-1} must be stored in memory; thus, memory complexity is approximately $2(m+1)$ times the size of the video frame. Larger q also produces more accurate

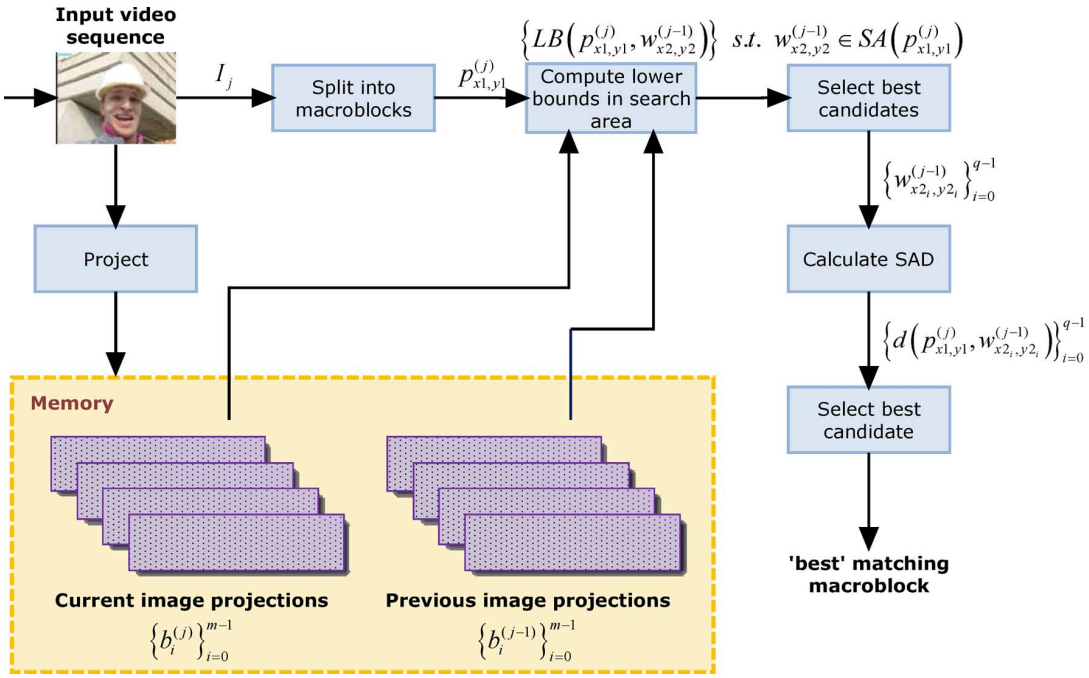


Fig. 8. FME-GCK algorithm.

TABLE I
COMPARISON OF ALGORITHM COMPLEXITY FOR MACROBLOCKS OF SIZE 16 × 16 AND SEARCH AREA OF SIZE 15 × 15. FME-GCK COMPLEXITY DEPENDS ON m, THE NUMBER OF PROJECTIONS AND q, THE NUMBER OF CANDIDATE MACROBLOCKS FOR WHICH THE SAD VALUE IS CALCULATED

Algorithm	Complexity [ops. per MB]
Full search	172,799
Three-step search	19,199
Diamond search	11,903
FME-GCK	1187m+993q+286

TABLE II
VIDEO SEQUENCES USED FOR SIMULATION EXPERIMENTS. FOR EACH RESOLUTION, VIDEO SEQUENCES ARE SORTED BY ASCENDING ORDER OF ESTIMATED CODING DIFFICULTY

QCIF	CIF
Akiyo	Akiyo
Miss America	Silent
Trevor	Foreman
Carphone	Tempete
Coastguard	Mobile
Foreman	Stefan

results at the cost of higher time complexity; it does not, however, affect memory.

Let us assume 1 time unit for each operation of addition, subtraction, multiplication, absolute value, and minimum of two numbers. Further assume a block of size $k \times k$ and search area of size $n \times n$. It is shown in [33] that FME-GCK requires $2k^2(m+1) + (3m-1)n^2 + qn^2 + 3k^2q - 1$ time units per template macroblock. Table I shows a comparison of full search, three-step [5], diamond search [8], [9], and FME-GCK, for $k = 16$, $n = 15$.

Note that when comparing the number of time units to perform FME-GCK with the time units to perform three-step search or diamond search, a factor γ should multiply FME-GCK's complexity. The factor γ is added since in FME-GCK both Inter and Intra macroblocks must be projected, in contradiction to zero calculations for Intra macroblocks incurred by the three-step search and the diamond search. The value of γ depends on the intra periodicity in the video sequence. A typical value for γ is 1.10.

Considering the results in Table I, we obtain that $\{m = 11, q = 4\}$ and $\{m = 5, q = 13\}$ are FME-GCK configurations similar in their computational complexity to three-step search and that $\{m = 5, q = 4\}$ is an FME-GCK configuration similar in its computational complexity to diamond search. Note that it has been verified by real-time code profiling that these configurations are indeed similar to their counterparts (see Section VI for details). These configurations allow comparison of the accuracy of motion information produced by FME-GCK, three-step search, and diamond search under the same computational constraints. It is important to note that the theoretical complexity comparison of FME-GCK to three-step search and to diamond search does not take into account the fact that FME-GCK allows sequential memory access whereas three-step search and diamond search incur many unpredictable branches and memory accesses. This difference may have a significant effect on run times in favor of the FME-GCK algorithm, depending on the specific hardware configuration. For example, sequential memory access is highly beneficial in digital signal processor (DSP) chips which are widely used in many signal processing applications.

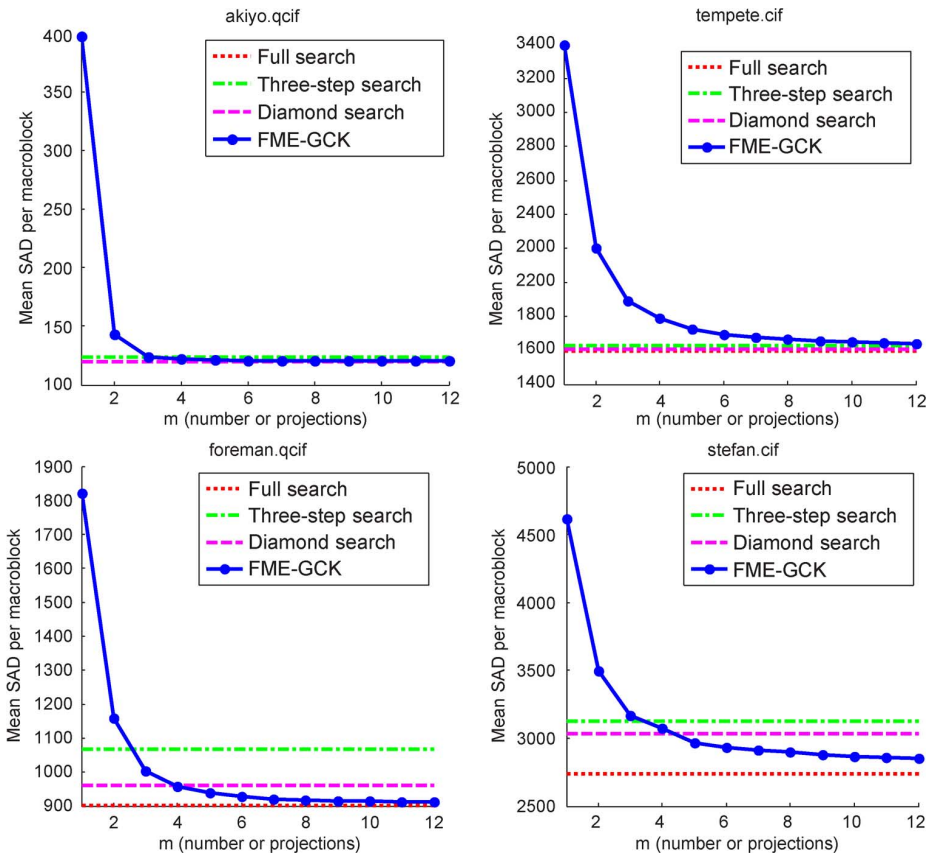


Fig. 9. Effect of different values of the parameter m on motion estimation accuracy. Results are for a constant $q = 4$. Full search, three-step search, and diamond search results are displayed as a reference. FME-GCK algorithm results converge to the optimal. FME-GCK significantly outperforms three-step search and is comparable to diamond search, when computational complexity is considered (see text).

VI. FME-GCK Results

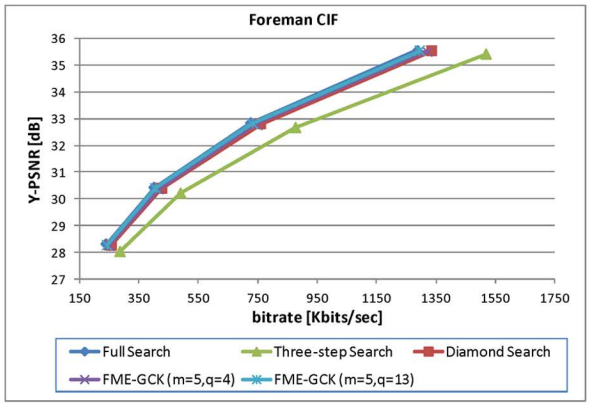
FME-GCK was implemented using a highly efficient ANSI-C code, together with its full search, three-step, and diamond search counterparts, in order to enable fair time and quality comparison. Implementation was performed and measured on a Pentium 4 PC at 3 GHz running Windows XP. In general, computational complexity was found to coincide with theoretical complexity calculation as described in Section V. Both diamond search and FME-GCK $\{m = 5, q = 4\}$ execute on this hardware configuration at the speed of about 110 CIF frames per second. First, an extensive set of simulations was performed. Then, FME-GCK and its counterparts were integrated with a video encoder in order to measure the effect on real video encoding. In both simulation and video encoding tests, motion estimation was performed with GOP size of 15, macroblocks of size 16×16, and search area of size 15×15.

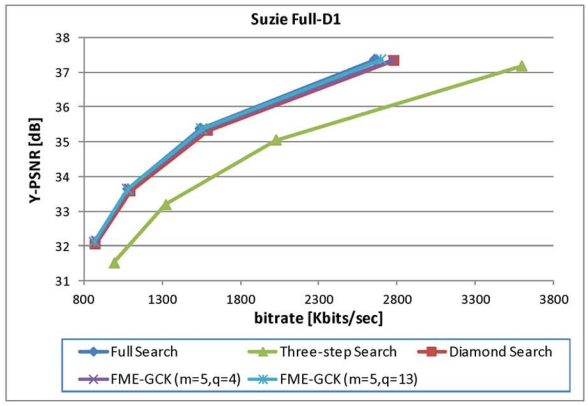
A. Simulation Results

All simulation results were obtained using the video sequences listed in Table II. In Fig. 9, the effect of different values of the parameter m on FME-GCK motion estimation accuracy with a constant $q = 4$ is depicted for a few representative video sequences (the complete set of results for all video sequences can be found in [33]). Motion estimation accuracy is measured in mean SAD per macroblock between macroblocks and their "best" matching counterparts. Full search, three-step search and diamond search results are displayed as reference. As expected, FME-GCK algorithm results converge to the optimal, namely increasing the number of projections produces lower SAD values, thus approaching full search SAD values. For all video sequences except one (Tempete CIF), FME-GCK outperforms three-step search for $m = 5$. For 9 out of the 12 video sequences, $m = 4$ is sufficient to outperform three-step search. Note that an FME-GCK configuration equal in its computational complexity to three-step search is $\{m = 11, q = 4\}$; thus, for the same motion accuracy, the gain in computation time by using FME-GCK compared to three-step is significant. A configuration of FME-GCK $\{m = 5, q = 4\}$ comparable in its computational time to diamond search, outperforms diamond search only in some of the video sequences. This will further improve in favor of FME-GCK with the introduction of an adaptive FME-GCK in Section VII.

It is also possible to maintain the parameter m constant and select different values of the parameter q. In this case as well, the FME-GCK algorithm results converge to the optimal, namely increasing the number of SAD calculation per macroblock produces lower SAD values, thus approaching full search SAD values (see [33]).

We summarize the simulation results section by stating that the FME-GCK algorithm significantly outperforms three-step search and produces motion information that is almost as accurate as diamond search. This will further improve in favor of FME-GCK with the introduction of an adaptive FME-GCK.





	FME-GCK {m=5,q=13}		FME-GCK {m=5,q=4}		Diamond Search		Three-step Search		Full Search		
	Y-PSNR	bitrate	Y-PSNR	bitrate	Y-PSNR	bitrate	Y-PSNR	bitrate	Y-PSNR	bitrate	QP
Foreman CIF	35.53	1295.45	35.50	1318.74	35.51	1335.48	35.39	1517.26	35.54	1286.80	28
	32.82	733.34	32.80	748.11	32.80	761.02	32.66	878.20	32.84	727.59	32
	30.41	406.69	30.38	416.81	30.39	425.67	30.21	492.51	30.42	403.07	36
	28.29	244.27	28.28	252.10	28.26	257.27	28.04	289.39	28.30	241.83	40
	-0.05	1.17%	-0.17	4.02%	-0.25	5.87%	-1.02	26.22%			Δ
Suzie Full-D1	37.36	2699.54	37.34	2774.16	37.33	2780.00	37.17	3595.87	37.35	2659.90	28
	35.37	1555.97	35.36	1584.05	35.32	1592.68	35.05	2026.72	35.36	1544.84	32
	33.64	1082.04	33.66	1092.80	33.58	1098.10	33.20	1323.81	33.63	1081.84	36
	32.14	870.98	32.17	880.74	32.05	875.46	31.53	994.14	32.11	873.24	40
	-0.01	0.26%	-0.09	1.94%	-0.15	3.37%	-1.41	37.22%			Δ

Fig. 10. FME-GCK rate-distortion video encoding results for Foreman CIF and Suzie Full-D1. Bitrate is given in Kbits/sec; Y-PSNR is given in decibels. Full search, three-step search, and diamond search results are displayed as a reference. Average difference in PSNR and in bitrate is computed according to [35]. For these two video sequences, FME-GCK outperforms both three-step search and diamond search.

B. Video Encoding Results

FME-GCK and its counterparts were integrated into a real video encoder. The standard JVT H.264/AVC reference software [2] was used for all tests. Several code modules were simplified or degenerated to comply with the FME-GCK implementation. These include: B pictures, motion estimation of submacroblocks smaller than 16 × 16 in size, subpixel motion estimation, and multiple reference frames for motion estimation. All experiments were performed according to the common testing conditions recommended in [34]. Therefore, the video sequences that appear in Table III were coded with QP values of 28, 32, 36, 40.

Rate-distortion results for Foreman CIF (352 × 288) and Suzie Full-D1 (720 × 480) can be found in Fig. 10. Rate-distortion results for three-step search and diamond search are displayed as reference. For every QP value in these figures, distortion (PSNR) was maintained roughly constant and mean $\Delta\text{bitrate}$ results are computed relative to full search according to [35]. Smaller $\Delta\text{bitrate}$ indicates more accurate motion estimation. For ten out of the eleven video sequences that appear in Table III, FME-GCK outperforms three-step search. For five out of these eleven video sequences, FME-GCK also outperforms diamond search.

TABLE III
VIDEO SEQUENCES USED FOR VIDEO CODING EXPERIMENTS. FOR EACH RESOLUTION, VIDEO SEQUENCES ARE SORTED BY ASCENDING ORDER OF ESTIMATED CODING DIFFICULTY

QCIF	CIF	Full D1
Container	Paris	Suzie
Silent Voice	Foreman	Waterfall
Foreman	Tempete	Football
	Mobile	Tempete

The video coding results corroborate the simulation results from Section VI-A. The FME-GCK algorithm significantly outperforms three-step search and produces motion information that is almost as accurate as diamond search. This too, will further improve in favor of FME-GCK with the introduction of an adaptive FME-GCK in Section VII.

VII. ADAPTIVE FME-GCK

An important advantage of FME-GCK compared to classical fast block motion estimation techniques is its ability to adapt to image content, producing a varying complexity block motion estimation algorithm. For some video coding applications, controlling the tradeoff between complexity and quality is a necessity. For example, software video codec (computational

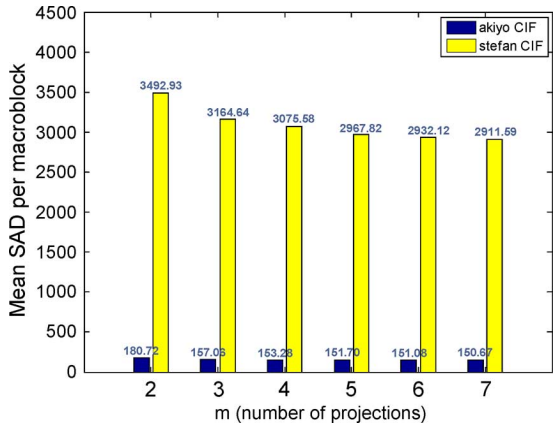


Fig. 11. Distortion when using FME-GCK with $q = 4$ and different values of m. Results are for the Akiyo and Stefan video sequences in CIF resolution. Akiyo is easy-to-code while Stefan is difficult-to-code. For both sequences, larger values of m (more projections) result in smaller distortion, but the expected improvement in coding efficiency is substantially larger for Stefan compared to Akiyo.

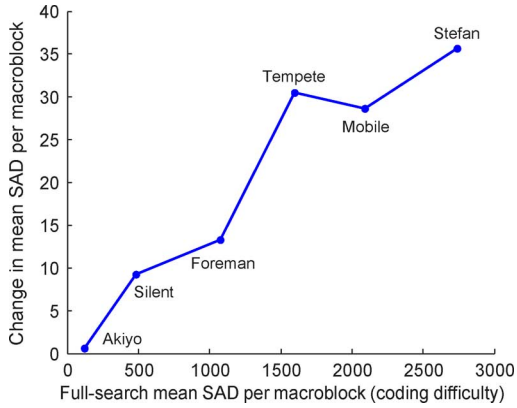


Fig. 12. Change in distortion when transitioning from $m = 5$ projections to $m = 6$ projections with constant $q = 4$. Results are for different video sequences in CIF resolution. More difficult-to-code video sequences result in larger change in distortion.

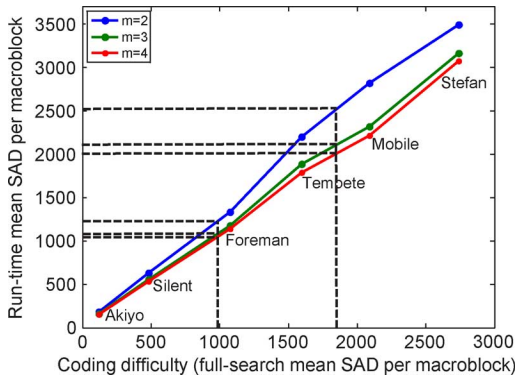


Fig. 13. Coding difficulty can be evaluated using run-time SAD. Mean SAD values using FME-GCK of sequences of varying coding difficulty are displayed versus mean full-search SAD values. Plots shown are for $m = 2, 3, 4$ and $q = 4$. Dashed lines represent two full-search SAD values and their corresponding run-time SAD for the three values of m.

complexity depends on the available processing resources), power-limited video codec (computational complexity depends on the power consumption budget), and multichannel video coding (computational resources are divided between different coding processes) [36]. Furthermore, even if not a necessity, an adaptive varying complexity may improve motion estimation accuracy compared with a nonadaptive method. In all scenarios, desired algorithm complexity may depend on external parameters, on the characteristics of the input video sequences, or on both. Since external parameters are application specific, we focus on adaptively changing FME-GCK parameters based only on the characteristics of the input video sequence.

Some video scenes are more difficult-to-code than others; Material containing an abundance of spatial detail and/or rapid, possibly nontranslational, movement generally requires more encoded bits than material containing little detail and/or simple motion. The more difficult-to-code material is not modeled well by the translational block-based motion model used in modern video coders, thus resulting in relatively large values in the residual signal, which, in turn, require many bits to code. Thus, the coding efficiency of these video scenes is relatively low. Increasing the computational resources for motion estimation of difficult-to-code scenes, if performed wisely, should improve their coding efficiency.

FME-GCK uses two parameters that affect the tradeoff between complexity and accuracy of resulting motion vectors. These parameters are m, the number of projections to perform for each image, and q, the number of candidate macroblocks for which the SAD value is calculated (Step 2.2 in algorithm). Larger m and larger q produce more accurate results at the cost of higher (time and memory) complexity.

Fig. 11 shows the mean SAD between macroblocks and their "best" matching regions in the previous frame for the sequences Akiyo and Stefan of length 300 frames each in CIF resolution. Akiyo is a "talking head" sequence with a small amount of simple motion whereas the Stefan sequence comprises of complex local and global motions. The matching regions were found by FME-GCK with constant $q = 4$ and with different values of m. Since Akiyo is easy-to-code, its residual signal is small, and since Stefan is difficult-to-code, its residual signal is substantially larger. The difference is more than an order of magnitude. As expected, for both sequences, larger values of m (more projections), produce a smaller residual signal. More importantly, however, is the fact that increasing the number of projections produces a reduction in SAD substantially greater for the Stefan sequence than for the Akiyo sequence. For example, increasing the number of projections from 2 to 3 reduces the mean SAD per macroblock in the Stefan sequence by 328.29. On the other hand, reducing the number of projections from 3 to 2 increases the mean SAD per macroblock sequence in the Akiyo sequence only by 23.66. Thus, using more projections for Stefan is much more effective in raising mean coding efficiency than using more projections for Akiyo. The same observation is also depicted in Fig. 12 which displays the change in mean SAD when transitioning from $m = 5$ projections to $m = 6$ projections (maintaining constant $q = 4$) for different CIF video sequences. As in Fig. 11, more difficult-to-code video sequences result in a larger change in distortion. Note that since mean SAD is a measure of subjective image quality (though not a very good one), using more projections for difficult-to-code video sequences increases their subjective quality and assists in

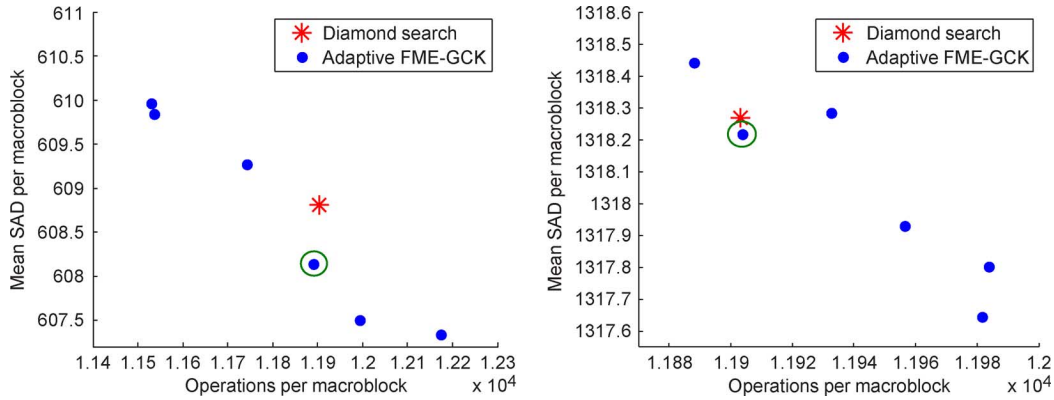


Fig. 14. Adaptive FME-GCK results for a concatenation of the six QCIF (left) and six CIF (right) video sequences listed in Table II. Different time-accuracy tradeoffs are produced according to threshold selection. For the same computational complexity, adaptive FME-GCK outperforms diamond search.

equating image quality across different video scenes. Similar conclusions can be deduced when q varies and m is maintained constant.

Thus, when a large set of video sequences with varying coding difficulties are to be coded, maintaining a balance of subjective quality across sequences can be obtained by selecting larger values of m and q for more difficult-to-code video sequences at the expense of lower parameter values for easy-to-code sequences. Transition thresholds on the coding difficulty can be determined when the number of projections m, or the number of SAD computations q, are to increase/decrease.

In order to change m and q dynamically during coding, an estimate of the coding difficulty of the current frame/macroblock must be determined. The average SAD per macroblock of a sequence obtained using full-search is a measure of coding difficulty (as used in Figs. 12 and 13). Unfortunately, this measure is not available during coding. An estimate of the full-search SAD (and, thus, of the coding difficulty) can be obtained from the residual of the previous frame, i.e. from the SAD available during coding using FME-GCK. However, this SAD value is dependant of the values of m and q as shown in Fig. 13. The figure depicts the average SAD values obtained during coding using FME-GCK for various video sequences as a function of the coding difficulty (measured as full-search SAD). Three plots are shown corresponding to $m = 2$, 3, and 4, with $q = 4$ in all cases. Using this figure, given the run-time SAD and given the number of projections m, the coding difficulty of the current frame can be determined. In Fig. 13, two transition thresholds are marked ($t = 1000$ and $t = 1800$ corresponding to transitions from 2 to 3 projections and from 3 to 4 projections, respectively). For example, if $m = 4$ and resulting SAD equals 2000, then the frame coding difficulty is evaluated as 1800 whereas if $m = 2$ then a SAD of 1500 would be associated with this level of coding difficulty. A similar figure can be used for varying q and constant m.

We note that the change in m, the number of projections to perform, is associated with a complete frame. On the other hand, adaptivity of q, the number of candidate macroblocks for which the SAD value is calculated, is applied at the macroblock level with q varying as a function of coding difficulty of each macroblock. Thus, controlling q requires a local estimate of coding difficulty. We further note that whereas the change in m affects both memory and time complexity, change in q does not affect memory at all. Additionally, note that, since computation of lower bounds requires current and previous frames, a change in m takes affect with one frame of delay.

Following are adaptive FME-GCK simulation results with a constant $q = 4$ and with variable values of the parameter m. The size of the residual of every frame was used as a simple coding difficulty estimate of its consecutive frame. This coding difficulty estimate is used to determine the transition between m values. As before, macroblocks are of size 16×16 and search area is of size 15×15. Fig. 14 shows time, measured in operations per macroblock, versus motion accuracy, measured in mean SAD per macroblock, for a video sequence that is a concatenation of all six QCIF (left) and CIF (right) video sequences listed in Table II. Results are plotted for different configurations of transition thresholds. In all configurations, more projections are preformed for more difficult-to-code scenes and fewer projections are performed for easier-to-code video scenes. Using adaptivity, the mean SAD for the concatenated video sequence is reduced. One adaptive FME-GCK configuration of m-threshold (marked with circle in Fig. 14) is of similar computation complexity as the diamond search (marked with asterisk in Fig. 14), yet outperforms it. We conclude that if thresholds are appropriately selected, adaptive FME-GCK outperforms diamond search on average. It should be noted that a residual based coding difficulty estimate was used to produce Fig. 14. A more sophisticated estimate is expected to improve adaptive FME-GCK performance. Such an estimate can also be used to adaptively control the parameter q to further improve FME-GCK performance.

VIII. Conclusion

In this paper, a novel fast block motion estimation algorithm, FME-GCK, was presented. FME-GCK uses an efficient projection framework that bounds the distance between a template block and candidate blocks using highly efficient filter kernels. Candidate regions that are distant from the template macroblock are quickly rejected using a rapid computation of lower bounds. For the few remaining candidate blocks, the SAD distortion measure is used. The FME-GCK algorithm enables flexibility in the tradeoff between coding efficiency and computational complexity by allowing adaptivity of the motion estimation process

based on image content and complexity limitations. Algorithm results are guaranteed to converge to the optimal (full search) with the increase of allowed computation. When tuned to computational complexity equal to that incurred by three-step search or by diamond search, and when adaptivity parameters are appropriately selected, the FME-GCK algorithm significantly outperforms both three-step search and diamond search. In addition, FME-GCK incurs only integer arithmetic and sequential memory access; thus, it is appropriate for embedded systems or for any other application where the constraints on memory complexity are not very tight. FME-GCK currently supports only a fixed block size. In order to use FME-GCK in video coders that support adaptive block size motion estimation (e.g., H.264/AVC), the algorithm should be extended with some fast heuristics to ignore nonreasonable block sizes.

REFERENCES

[1] K.-P. Lim, G. Sullivan, and T. Wiegand, Text Description of Joint Model Reference Encoding Methods and Decoding Concealment Methods, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG Doc. JVT-X101, Jul. 2007.

[2] H.264/AVC Reference Software ver. 11.1, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG [Online]. Available: http://iphome.hhi.de/suehring/tml/, Aug. 2006

[3] Y.-W. Huang, C.-Y. Chen, C.-H. Tsai, C.-F. Shen, and L.-G. Chen, "Survey on block matching motion estimation algorithms and architectures with new results," *J. VLSI Signal Process.*, vol. 42, no. 3, pp. 297–320, Mar. 2006.

[4] J. R. Jain and A. K. Jain, "Displacement Measurement and Its Application in Interframe Image Coding," *IEEE Trans. Commun.*, vol. 29, no. 12, pp. 1799–1808, Dec. 1981.

[5] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing," in *Proc. Nat. Telecommun. Conf.*, 1981, pp. G5.3.1–G5.3.5.

[6] L. M. Po and W. C. Ma, "A novel four-step search algorithm for fast block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 3, pp. 313–337, Jun. 1996.

[7] M. Ghanbari, "The cross-search algorithm for motion estimation," *IEEE Trans. Commun.*, vol. 38, no. 7, pp. 950–953, Jul. 1990.

[8] S. Zhu and K.-K. Ma, "A new diamond search algorithm for fast block matching motion estimation," in *Proc. IEEE Int. Conf. Inf., Commun., Signal Process.*, 1997, pp. 292–29.

[9] J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A novel unrestricted center-biased diamond search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 8, no. 4, pp. 369–377, Apr. 1998.

[10] C. H. Hsieh, P. C. Lu, J. S. Shyn, and E. H. Lu, "Motion estimation algorithm using interblock correlation," *Electron. Lett.*, vol. 26, no. 5, pp. 276–277, Mar. 1990.

[11] J. Chalidabhongse and C. C. J. Kuo, "Fast motion vector estimation using multiresolution-spatio-temporal correlations," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, no. 3, pp. 477–488, Jun. 1997.

[12] A. Zaccarin and B. Liu, "Fast algorithms for block motion estimation," in *Proc. IEEE Int. Conf. Acoust., Speech, Signal Process.*, 1992, vol. 3, pp. 449–452.

[13] B. Liu and A. Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 3, no. 2, pp. 148–157, Apr. 1993.

[14] D. Tzovaras, M. G. Strintzis, and H. Sahinolou, "Evaluation of multiresolution block matching techniques for motion and disparity estimation," *Signal Process.: Image Commun.*, vol. 6, no. 1, pp. 59–67, Mar. 1994.

[15] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Trans. Image Process.*, vol. 4, no. 1, pp. 105–107, Jan. 1995.

[16] M. Yang, H. Cui, and K. Tang, "Efficient tree structured motion estimation using successive elimination," *Proc. IEEE Vis., Image, Signal Process.*, vol. 151, no. 5, pp. 369–377, 2004.

[17] C.-H. Lee and L.-H. Chen, "A fast motion estimation algorithm based on the block sum pyramid," *IEEE Trans. Image Process.*, vol. 6, no. 11, pp. 1587–1591, Nov. 1997.

[18] Y.-S. Chen, Y.-P. Hung, and C.-S. Fuh, "A fast block matching algorithm based on the winner-update strategy," *IEEE Trans. Image Process.*, vol. 10, no. 8, pp. 1212–22, Aug. 2001.

[19] S.-Y. Choi and S.-I. Chae, "Hierarchical motion estimation in Hadamard transform domain," *Electron. Lett.*, vol. 35, no. 25, pp. 2187–2188, 1999.

[20] M. Brunig and B. Menser, "A fast exhaustive search algorithm using orthogonal transforms," in *Proc. 7th Int. Workshop on Syst., Signals Image Process.*, 2000, pp. 111–114.

[21] S.-W. Liu, S.-D. Wei, and S.-H. Lai, "Winner update on Walsh–Hadamard domain for fast motion estimation," in *Proc. 18th Int. Conf. Pattern Recognition*, 2006, vol. 3, pp. 794–797.

[22] N. Li, C.-M. Mak, and W.-K. Cham, "Fast block matching algorithm in Walsh–Hadamard domain," in *Proc. 7th Asian Conf. Comput. Vision*, 2006, pp. 712–721.

[23] Y. Hel-Or and H. Hel-Or, "Real-time pattern matching using projection Kernels," in *Proc. 9th IEEE Int. Conf. Comput. Vision*, 2003, pp. 1486–1493.

[24] Y. Hel-Or and H. Hel-Or, "Real-time pattern matching using projection Kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 9, pp. 1430–1445, Sep. 2005.

[25] G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or, "Filtering with Gray-Code Kernels," in *Proc. 17th Int.Conf. Pattern Recognition*, 2004, vol. 1, pp. 556–559.

[26] G. Ben-Artzi, H. Hel-Or, and Y. Hel-Or, "The Gray-Code filter Kernels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 3, pp. 382–393, Mar. 2007.

[27] Y. Moshe and H. Hel-Or, "A fast block motion estimation algorithm using Gray Code Kernels," in *Proc. 6th IEEE Int. Symp. Signal Processing and Information Technology*, Vancouver, BC, Canada, 2006, pp. 185–190.

[28] K. G. Beauchamp, *Applications of Walsh and Related Functions*. New York: Academic, 1984.

[29] M. Gardner, "The Binary Gray Code," in *Knotted Doughnuts and Other Mathematical Entertainments*. New York: W. H. Freeman, 1986, pp. 11–27.

[30] P. Simard, L. Bottou, P. Haffner, and Y. LeCun, "Boxlets: A fast convolution algorithm for neural networks and signal processing," *Adv. Neural Inf. Process. Syst.*, 1999.

[31] H. Kitajima, "Energy packing efficiency of the Hadamard transform," *IEEE Trans. Commun.*, vol. 24, no. 11, pp. 1256–1258, Nov. 1976.

[32] I. E. G. Richardson, *H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia*. Hoboken, NJ: Wiley, 2003.

[33] Y. Moshe, "Fast Block Motion Estimation Using Gray-Code Kernels," M.S. Thesis, Dept. Comput. Sci., Univ. Haifa, Haifa, Israel, 2007.

[34] T. Tan, G. Sullivan, and T. Wedi, Recommended Simulation Common Conditions for Coding Efficiency Experiments, ITU-T Q.6/SG16, Document VCEG-AA10d1, Oct. 2005.

[35] G. Bjontegaard, Calculation of Average PSNR Differences Between RD-Curves ITU-T Q.6/SG16, Document VCEG-M33, Apr. 2001.

[36] I. E. G. Richardson, *Video Codec Design: Developing Image and Video Compression Systems*. Chichester, U.K.: Wiley, 2002.

Yair Moshe (M'02) received the B.Sc. degree in computer and software engineering from The Technion—Israel Institute of Technology, Haifa, in 2002, and the M.Sc. degree in computer science from the University of Haifa, Israel, in 2008. He is currently pursuing the Ph.D. degree in computer science in the University of Haifa.

Since 2002, he has been a Senior Engineer in the Signal and Image Processing Laboratory (SIPL), Israel Institute of Technology. His research interests include image and video analysis, image and video coding, and pattern recognition.

Hagit Hel-Or received the Ph.D. degree in computer science from the Hebrew University of Jerusalem, Israel, in 1994.

From 1996–1998, she was with the Department of Mathematics and Computer Science, Bar-Ilan University, Israel. Since 1998, she has been a faculty member in the Department of Computer Science, University of Haifa, Israel. She has held visiting scholar positions in the Vision Group, Department of Psychology and in the Department of Statistics, both at Stanford University, Stanford, CA. Her research interests in the area of image processing and computer vision include image enhancement, pattern recognition, color vision, imaging technologies, and computational and human vision.