Fast Block Matching Algorithm Based on the Winner-Update Strategy

Yong-Sheng Chen, Yi-Ping Hung, and Chiou-Shann Fuh

Abstract— **Block matching is a widely-used method for stereo vision, visual tracking, and video compression. Many fast algorithms for block matching have been proposed in the past, but most of them do not guarantee that the match found is the globally optimal match in a search range. This paper presents a new fast algorithm based on the winner-update strategy which utilizes an ascending lower bound list of the matching error to determine the temporary winner. Two lower bound lists derived by using partial distance and by using Minkowski's inequality are described in this paper. The basic idea of the winner-update strategy is to avoid, at each search position, the costly computation of matching error when there exists a lower bound larger than the global minimum matching error. The proposed algorithm can significantly speed up the computation of the block matching because**

1) computational cost of the lower bound we use is less than that of the matching error itself;

2) an element in the ascending lower bound list will be calculated only when its preceding element has already been smaller than the minimum matching error computed so far;

3) for many search positions, only the first several lower bounds in the list need to be calculated.

Our experiments have shown that, when applying to motion vector estimation for several widely-used test videos, 92% to 98% of operations can be saved while still guaranteeing the global optimality. Moreover, the proposed algorithm can be easily modified either to meet the limited time requirement or to provide an ordered list of best candidate matches. Our source codes of the proposed algorithm are available at http://smart.iis.sinica.edu.tw/html/winup.html.

Index Terms— **Block matching, fast algorithm, Minkowski's inequality, motion estimation, winner-update strategy.**

I. Introduction

BLOCK matching technique has been widely used for finding the corresponding points in stereo vision, visual tracking, and video compression. In this paper, we focus our discussion of fast block matching algorithm on its application to motion vector estimation for video compression. However, the proposed fast algorithm can be easily applied to other applications which need block matching.

Video compression is important for efficient transmission and storage of video data. In order to achieve a high compression ratio, motion-compensated predictive coding is commonly used for reducing the temporal redundancy existing in video sequences. The value of a pixel in the current image is predicted as the value of its corresponding pixel in the reference image. The displacement between these two corresponding pixels, referred to as the motion vector, and the prediction residual are coded in the data stream. Usually, each image is divided into nonoverlapping blocks. The motion vectors for all the pixels in one block are treated as one single vector to be estimated. This kind of block-based motion compensation is widely adopted by video coding standards [1], [2], [3], [4] due to its simplicity.

The block-based motion vectors can be estimated by using block matching, which minimizes a measure of matching error. The matching error between the block at position (x, y) in the current image, I_t, and the candidate block at position $(x+u, y+v)$ in the reference image, I_{t-1}, is usually defined as the sum of absolute difference (SAD)

$$\mathrm{SAD}_{(x,y)}(u,v) \equiv \sum_{j=0}^{B-1}\sum_{i=0}^{B-1} |I_t(x+i,y+j) - I_{t-1}(x+u+i,y+v+j)|, \quad (1)$$

where the block size is $B \times B$. SAD will be referred to as the *complete matching error* for its summation over the "complete" block or the "complete" summation set. In this paper, the best estimate of the motion vector, $(\hat{u}, \hat{v})$, is defined to be the (u, v) which minimizes $\mathrm{SAD}_{(x,y)}(u, v)$. This estimate, $(\hat{u}, \hat{v})$, can be obtained by using the full-search (FS) algorithm which calculates and compares the SADs for all the search positions, $\{(x+u, y+v)\}$, in the reference image, I_{t-1}. That is,

$$(\hat{u},\hat{v}) \equiv \arg\min_{(u,v)\in S} \mathrm{SAD}_{(x,y)}(u,v), \quad (2)$$

where $S = \{(u, v) | -R \le u, v \le R$ and $(x+u, y+v)$ is a valid position in $I_{t-1}\}$ is the *search set* and R is an integer which determines the search range. This straightforward method takes extremely large amount of computation, although it can find the motion vector which gives the global minimum of matching error.

A. Literature Review

In the past, many techniques have been developed to accelerate the block matching process. We classify these techniques into three categories. The techniques in the first category save the computations by reducing the number of the positions searched. Therefore, the obtained minimum of the matching error may only be a local minimum within the search set, S. The techniques in the second category, on the other hand, try to reduce the computational cost of the matching error for each search position. Whether this kind of techniques can obtain the global minimum or not depends on how we compute and compare the matching errors. The techniques in the first and second categories can be combined to further improve the efficiency and this kind of hybrid methods are classified as the third category.

Yong-Sheng Chen is currently a Ph.D. candidate in the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, and a research assistant in the Institute of Information Science, Academia Sinica, Taipei, Taiwan. E-mail: yschen@iis.sinica.edu.tw. Tel.: +886-2-27883799 ext. 1518. Fax: +886-2-27824814.

Yi-Ping Hung is a research fellow in the Institute of Information Science, Academia Sinica, Taipei, Taiwan, and an adjunct professor in the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan. E-mail: hung@iis.sinica.edu.tw. Tel.: +886-2-27883799 ext. 1718. Fax: +886-2-27824814.

Chiou-Shann Fuh is a professor in the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan. E-mail: fuh@csie.ntu.edu.tw. Tel.: +886-2-23625336 ext. 327. Fax: +886-2-23628167.

A.1 Category 1: Partial-Search-Set Techniques

Techniques in this category perform the matching error calculation and comparison within a partial search set, S', which is a subset of the complete search set, S, defined in Equation (2). The efficiency of these techniques depends on the number of the selected search positions, while the resulted minimum matching error depends on how the search positions are selected. Consequently, the design issue of these techniques is how to find the motion vector with small minimum matching error by examining as smaller number of search positions as possible.

The gradient descent techniques, based on the unimodal error surface assumption, belong to this category. Starting from the initial value, the motion vector with the local minimum matching error, SAD, can be found by using pixel-by-pixel search along the conjugate directions or simply the horizontal and vertical directions [5]. The number of the search positions examined depends on the distance between the initial position and the "valley" position with the local minimum SAD. The minimum SAD obtained is sometimes larger than the global minimum SAD because the search process may be trapped at a local minimum.

Some other gradient descent techniques use the multiresolution search space of motion vectors for the coarse-to-fine search. These techniques divide the search process into several search steps. Starting from the origin position, $(u, v) = (0, 0)$, SADs of several coarsely-spaced search positions (i.e., in coarse resolution) are calculated and the one with the minimum SAD is selected as the new starting position of the next step. This procedure is repeated several times with smaller and smaller spacing between the search positions (i.e., in finer resolution) until the search positions with spacing of one pixel are examined. The final search position with the minimum SAD is selected as the search result. Well-known examples of this kind of techniques include the three-step search (TSS) algorithm [6], the two-dimensional logarithmic search algorithm [7], and the cross-search algorithm [8]. The number of the search positions examined is roughly constant. Thus the computational cost of motion estimation is also roughly fixed. This is a good feature especially for on-line video compression applications, for example, video conference, which prefer static video compression speed.

Another important technique in this category is to restrict the search region in a smaller region determined by the predicted motion vector. The value of the motion vector can be predicted from the motion vector of the spatially and temporally adjacent blocks together with the hierarchical related blocks because the motion vectors of these blocks are statistically coherent [9].

A.2 Category 2: Partial-Matching-Error Techniques

Techniques in this category accelerate the calculation of matching error for each search position. Instead of calculating the complete matching error, SAD, the technology in this category calculates a *partial matching error* which needs less computation than SAD and whose value is less than or equal to SAD. From Equation (1), for example, we can define the partial sum of absolute difference (PSAD) of l pixels, $l = 1, 2, \ldots, B^2$, as

$$\mathrm{PSAD}^{l}_{(x,y)}(u,v) \equiv \sum_{m=0}^{l-1} |I_t(x+i(m), y+j(m)) - I_{t-1}(x+u+i(m), y+v+j(m))|, \quad (3)$$

where $\{(i(m), j(m)) | m = 0, \ldots, B^2 - 1\}$ is the index set of all the pixels in the block. Here, PSAD can be treated as a partial matching error. The index set determines the positions and the order of the pixels in the matching block used for the accumulation of PSAD. Suitable positions and order may be raster scan, spiral, checkered, or any other predefined order.

One simple technique in this category is to subsample the pixels in the matching blocks. For example, $\mathrm{PSAD}^{\frac{B^2}{4}}_{(x,y)}(u,v)$ can be calculated by using a quarter of pixels regularly subsampled in each matching block [10]. The motion vector is determined by choosing the one with minimum $\mathrm{PSAD}^{\frac{B^2}{4}}_{(x,y)}(u,v)$. This kind of techniques cannot guarantee that the global minimum SAD to be obtained.

Another technique, called the *partial distance* method [11], [12], can be used for speeding up the computation. For each search position, the PSADs are calculated from $\mathrm{PSAD}^{1}_{(x,y)}(u,v)$ to $\mathrm{PSAD}^{B^2}_{(x,y)}(u,v)$. During the calculation process, if one of these PSADs is larger than the minimum matching error computed so far (which will be referred to as the so-far-minimum matching error), the calculation for this search position can be terminated and the calculation of the PSADs behind can be saved. If the last PSAD, $\mathrm{PSAD}^{B^2}_{(x,y)}(u,v)$ ($= \mathrm{SAD}_{(x,y)}(u,v)$), is computed and it is still smaller than the so-far-minimum matching error, the so-far-minimum matching error will be updated to be this $\mathrm{PSAD}^{B^2}_{(x,y)}(u,v)$. All the search positions are examined one by one and the calculation of many PSADs can be saved. Obviously, this technique can find the global minimum matching error.

One variation of the partial distance method is the so-called early jump-out technique [13]. This technique interrupt the calculation of PSADs for each search position when one of the PSADs is large enough compared to a threshold sequence instead of the so-far-minimum matching error. In general, this technique can save more computations than the partial distance method. However, it can not guarantee the global minimum matching error.

Another variation of the partial distance method defines a new kind of partial matching error which utilizes the projections (or partial sums) of the pixel values instead of the pixel values themselves [14], [15], [16]. It can be proved by using Minkowski's inequality that the SADs of the projections are smaller than or equal to the matching error. Hence, the SADs of the projections can be treated as a partial matching error. This kind of techniques first calculate the complete matching error at the predicted position and use it as the initial value of the so-far-minimum matching error. For each search position other than the predicted one, the partial matching errors (the SADs of the projections) are first calculated. The calculation of the complete matching error can be avoided if any one of the partial matching errors is larger than the so-far-minimum matching error. Otherwise, the complete matching error is calculated and

then the so-far-minimum matching error is updated to be this newly computed matching error if it is indeed smaller than the so-far-minimum matching error. The major advantage of this technique is that it guarantees the global optimality. However, if the predicted position is not accurate enough (for example, when the image sequences contains objects with abrupt motion), the initial minimum matching error may be quite large. Hence a lot of useless calculation of the partial and complete matching errors will be performed until a position with small complete matching error is examined. In the worst case, the complete matching errors are monotonically decreasing with respect to the visiting order of the search positions. For such case, the computational cost will even be higher than that of using the FS algorithm.

A.3 Category 3: Hybrid Techniques

To speed up the block matching, the techniques in Category 1 reduces the number of search positions by choosing a smaller partial search set. In contrast, the techniques in Category 2 gain their efficiency by reducing the frequency of calculating the complete matching errors. The techniques in Category 3 combine the techniques in the above two categories, as described in [10], [17], to further improve the efficiency. Another technique in this category is the hierarchical method [18] which first estimates the coarse result of the motion vector in the lower resolution image, and then refines the result in the higher resolution image within a small search region centering at the coarse result.

B. Overview of Our Work

For the application in video compression, the major drawback of not guaranteeing the global minimum is that the compression ratio is generally lower due to the higher matching error (which can be thought of as the prediction residual to be coded). In this paper, we propose an efficient and simple block matching algorithm, named the winner-update algorithm. This algorithm, belonging to Category 2, can accelerate the block matching process while still ensuring that the global minimum of the matching error can be obtained. This algorithm utilizes an ascending lower bound list of the matching error to save the computation. The two lower bound lists described in this paper are derived by using the partial distance and by using Minkowski's inequality. The cost of computing a lower bound of the matching error should be less than that of computing the matching error itself. The basic idea is to avoid, at each search position, the computation of the more costly matching error when there exists a lower bound larger than the global minimum matching error. The proposed algorithm uses a comparison strategy, called the winner-update strategy, to switch the computation among the search positions depending on the comparison of the updated partial matching error of the temporary winner with the *so-far-minimum partial matching error*. The computational efficiency of the winner-update algorithm is irrelevant to the visiting order of the search positions. By using the proposed algorithm, only a small subset of lower bounds and matching errors are actually calculated. Consequently, the total computational cost can be significantly reduced.

For further speedup, the proposed algorithm can be easily combined with the techniques in Category 1 which reduce the number of the search positions. The result of the combination is that the guarantee of global optimality is given up, which is inevitable for all hybrid techniques. In this paper, we shall also present the result of combining the winner-update algorithm and the TSS algorithm, as an example of the combination.

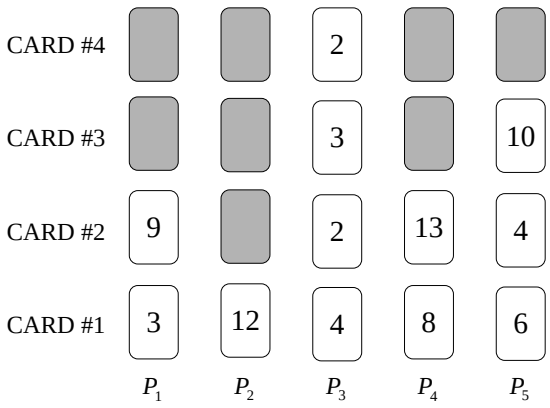


Fig. 1. A simple game for illustrating the concept of the winner-update algorithm. There are five players, P_1, ..., P_5, and each player is dealt four cards. Player P_3 has finished the accumulation of the values in his/her hand and becomes the winner. The temporary accumulations of the others are all greater than the final accumulation result of P_3. Hence their remaining calculation can be saved.

This paper is organized as follows. At first, we introduce the concept of the winner-update strategy and the proposed algorithm in Section II. Then, the combination of the proposed algorithm and the TSS algorithm is presented in Section III. Section IV presents the experimental results of the proposed algorithm as well as some previously developed algorithms and compares their performance. Finally, some discussions and conclusions are stated in Sections V and VI.

II. Winner-Update Algorithm

A. Concept

In this section, we use a simple game of poker cards to illustrate the concept of the winner-update algorithm. Suppose there are five players in the game and each player is dealt four cards. The value of each card ranges from 1 to 13. The penalty score of each player is the sum of the values of his/her hand and the player with the minimum penalty score is the winner. The basic idea is that one does not have to calculate the summation of all the card values for each player when determining the winner. If the intermediate summation, or the lower bound of the total penalty score, for one player has already been greater than the total penalty score of the winner, then he/she has no chance to win and hence we can stop calculating his/her penalty score to save some computation. An example is shown in Figure 1. Of course, the penalty score of the winner is not known in advance. But the winner-update strategy explained below can help to solve the problem.

At the beginning, we lay all the cards face down except the first one of each hand. The lower bound of the penalty score for each player is initialized as the value of the first card. Among all the players, only the temporary winner who has the minimum lower bound is allowed to turn up the face of the next card of

TABLE I
STEP BY STEP CALCULATION OF THE PENALTY SCORE OF EACH PLAYER. THE SCORE NUMBERS IN BOLDFACE ARE THE TEMPORARY MINIMA AFTER EACH STEP.

operation	P_1	P_2	P_3	P_4	P_5
initialize	**3**	12	4	8	6
turn up CARD #2 of P_1	12	12	**4**	8	6
turn up CARD #2 of P_3	12	12	**6**	8	6
turn up CARD #3 of P_3	12	12	9	8	**6**
turn up CARD #2 of P_5	12	12	9	**8**	10
turn up CARD #2 of P_4	12	12	**9**	21	10
turn up CARD #4 of P_3	12	12	11	21	**10**
turn up CARD #3 of P_5	12	12	**11**	21	20
choose P_3 as the winner	12	12	**11**	21	20

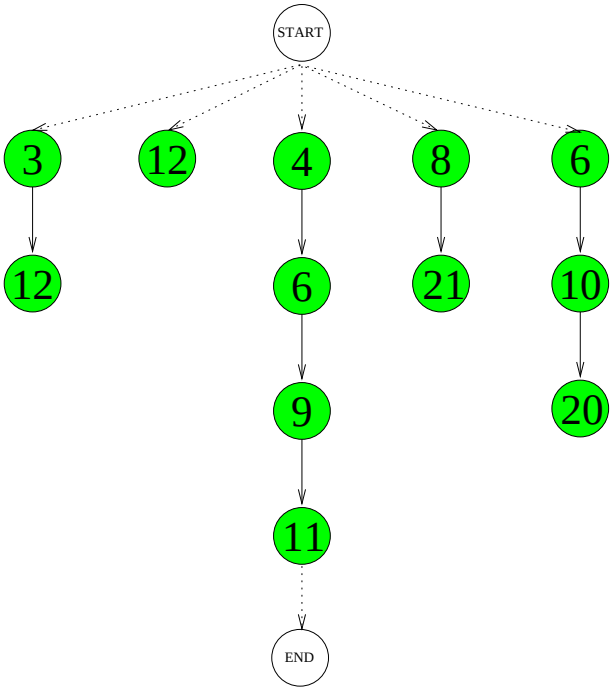


Fig. 2. The winner-update strategy is a special case of the branch-and-bound strategy with only one branch for each node except the "START" node. This figure illustrates the search process conducted in Table I.

his/her hand and update (increase) his/her intermediate lower bound of the penalty score. A new temporary winner is then selected and this process is repeated until the temporary winner has no card laid facedown and becomes the final winner. Table I lists the operations step by step to demonstrate the process for the example given in Figure 1. This comparison strategy, called the winner-update strategy, is a special case of the branch-and-bound strategy [19] (1-branch only) as shown in Figure 2.

In block matching for motion vector estimation, the matching errors between the template block and most of the candidate matching blocks are usually very large compared with the minimum matching error. That is, most of the players hold cards with relatively large values except the winner and a few tough competitors. Therefore, by using this winner-update strategy, the number of the cards laid facedown, which implies the saved computations, can be enormous.

B. Lower Bound Derived from Partial Accumulation

The process of choosing the winner having the minimum penalty score in the previously mentioned game resembles the process of finding the corresponding block having the minimum matching error in block matching. Each player in the game stands for a search position and the penalty score is the absolute difference between the values of the corresponding pixels. Thus, PSAD defined in Equation (3) represents the temporarily accumulated penalty score. Obviously, the following inequality relationship holds true (subscript (x, y) is dropped for simplicity):

$$\begin{aligned} \mathrm{PSAD}^1(u,v) \le \mathrm{PSAD}^2(u,v) \le \cdots &\le \mathrm{PSAD}^{B^2}(u,v) \\ &= \mathrm{SAD}(u,v). \end{aligned}$$

As a result, the list of the partial accumulation, $\{\mathrm{PSAD}^l(u,v), l = 1, \ldots, B^2\}$, can be used as the lower bound list needed by the winner-update strategy. That is,

$$\mathrm{LB}^l(u,v) = \mathrm{PSAD}^l(u,v), l = 1, \ldots, K,$$

where $K = B^2$. This lower bound will be referred to as the partial-sum lower bound. Notice that the last element in this list, $\mathrm{LB}^K(u,v)$, equals the complete matching error, $\mathrm{SAD}(u,v)$. These K lower bounds are in ascending order:

$$\mathrm{LB}^1(u,v) \le \mathrm{LB}^2(u,v) \le \cdots \le \mathrm{LB}^K(u,v).$$

Furthermore, the number of pixels used for summing up these lower bounds are ascending from 1 to B^2.

Next, we use a practical example to give the intuitive feeling of how the computation can be saved with the winner-update strategy. As shown in Figure 3, the template block with size 16×16 in image I_t is matched in the search range $([-16, 16] \times [-16, 16])$ in image I_{t-1}. When the optimal match is found, the number of pixels, l, used for calculating the lower bound, $\mathrm{LB}^l(u,v)$, is relatively small for most search positions, as shown in Figure 4(a). Even with such a small number of pixels, its lower bound has already been larger than the global minimum error, as can be seen in Figure 4(b). Consequently, the block matching process can be terminated and the area between the dashed and solid lines in Figure 4(a), which implies the amount of saved computation, can be quite large.

C. General Winner-Update Algorithm

The winner-update algorithm for fast block matching is described in this section. Consider a template block at position (x, y). For each search position $(x + u, y + v)$, we let $\mathrm{LB}(u,v)$ denote the currently-used lower bound of the matching error between this template block and the candidate matching block at position $(x + u, y + v)$ in the reference image. At the beginning, $\mathrm{LB}(u,v)$ for each search position is initialized as the first element in its lower bound list, $\mathrm{LB}^1(u,v)$. An additional variable, $l(u,v)$, is used to record the index of the element to which $\mathrm{LB}(u,v)$ is assigned. Initially, $l(u,v)$ is set to be 1. Let $(\hat{u}, \hat{v})$ be the search position that has the minimal $\mathrm{LB}(u,v)$ among all

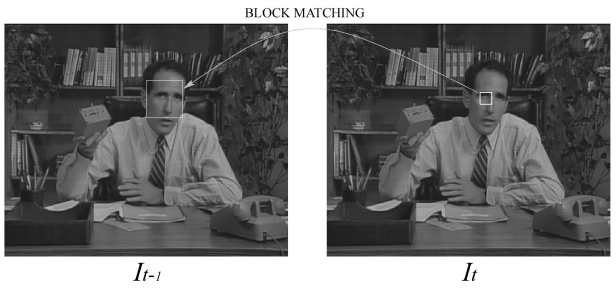


Fig. 3. A template block with size 16×16 in image I_t is matched in the search range ($[-16, 16] \times [-16, 16]$) in image I_{t-1}.

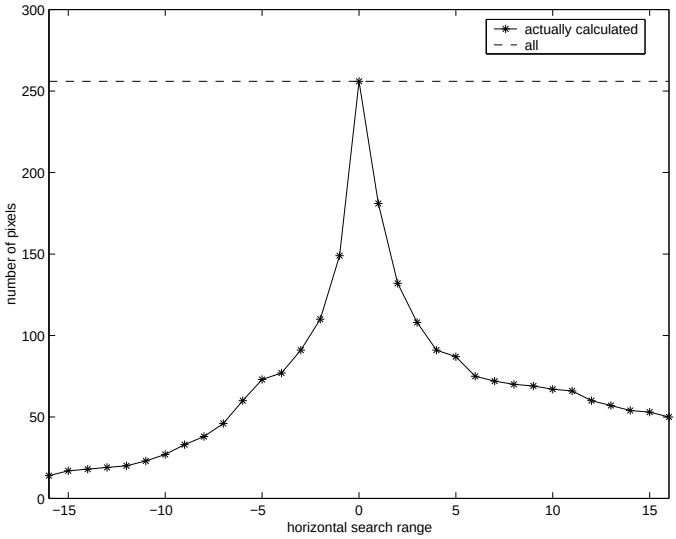


Fig. 4. (a) Number of pixels used for calculating the partial matching error (solid line) and the complete matching error (dashed line). The area between these two lines implies the amount of saved computation. Only the search positions along the horizontal search line passing through the best estimate, $(\hat{u}, \hat{v})$, are shown for simplicity. (b) This figure illustrates the complete matching error, SAD, (dashed line) and the partial matching error, LB, (solid line) computed up to the moment when the global minimum is found.

search positions. Then the temporary winner, $(\hat{u}, \hat{v})$, updates its lower bound $\mathrm{LB}(\hat{u}, \hat{v})$ with the next element in the lower bound list, $\mathrm{LB}^{l(\hat{u},\hat{v})+1}(\hat{u}, \hat{v})$, and updates the new index number. A new temporary winner is then chosen, again as the search position $(\hat{u}, \hat{v})$ having the minimal $\mathrm{LB}(u, v)$. This process is repeated until $l(\hat{u}, \hat{v})$ of the new winner equals K, the number of the last element in the lower bound list. Remember that the last lower bound, $\mathrm{LB}^K(\hat{u}, \hat{v})$, is actually the matching error $\mathrm{SAD}(\hat{u}, \hat{v})$ at position $(\hat{u}, \hat{v})$. The block matching process can now stop because the lower bounds of the matching errors for other search positions are all larger than the matching error of the winner.

The proposed algorithm is summarized below:

The Winner-Update Algorithm

Given a template block at position (x, y) in I_t
begin
  for each (u, v) in the search range **do**
    begin (initialization)
      calculate $\mathrm{LB}^1(u, v)$
      $\mathrm{LB}(u, v) := \mathrm{LB}^1(u, v)$
      $l(u, v) := 1$
    end
  select $(\hat{u}, \hat{v})$ having the minimal $\mathrm{LB}(u, v)$ to be the temporary winner
  while $l(\hat{u}, \hat{v}) < K$ **do**
    begin
      $l(\hat{u}, \hat{v}) := l(\hat{u}, \hat{v}) + 1$
      calculate $\mathrm{LB}^{l(\hat{u},\hat{v})}(\hat{u}, \hat{v})$
      $\mathrm{LB}(\hat{u}, \hat{v}) := \mathrm{LB}^{l(\hat{u},\hat{v})}(\hat{u}, \hat{v})$
      select $(\hat{u}, \hat{v})$ having the minimal $\mathrm{LB}(u, v)$ to be the new temporary winner
    end
  output $(\hat{u}, \hat{v})$
end

D. Lower Bound Derived from Minkowski's Inequality

For each search position, the number of lower bounds, PSADs, defined in Equation (3) is B^2. We want to reduce the number of lower bounds because the winner-update algorithm has to select a new temporary winner each time when a new lower bound is calculated. One possibility is to rewrite the calculation of PSAD defined in Equation (3) as follows:

$$\mathrm{RSAD}^l_{(x,y)}(u, v) \equiv \sum_{j=0}^{l-1} \sum_{i=0}^{B-1} |I_t(x + i, y + j) - I_{t-1}(x + u + i, y + v + j)|,$$

for $l = 1, \ldots, B$ (instead of $l = 1, \ldots, B^2$). These lower bounds, $\mathrm{RSAD}^1_{(x,y)}(u, v)$, ..., $\mathrm{RSAD}^B_{(x,y)}(u, v)$, constitute another ascending lower bound list, which is in fact a partial list of the original PSAD list. In this lower bound list, the l-th element, $\mathrm{RSAD}^l_{(x,y)}(u, v)$, means the sum of absolute difference of the pixels in the first l rows. Other kind of lower bound can be used in the winner-update algorithm. In the following, we will describe the lower bound derived from Minkowski's inequality. This kind of lower bound is proposed by Lee and Chen [16] in

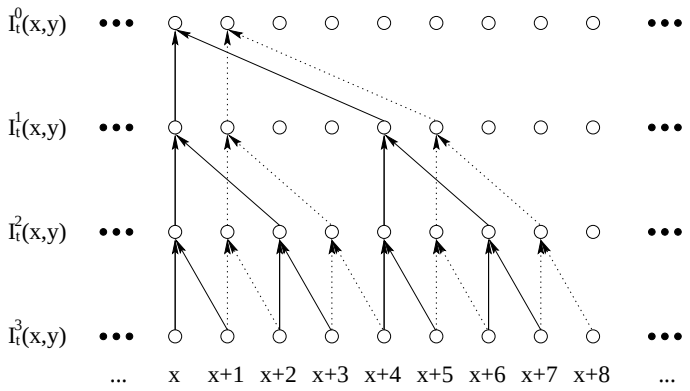


Fig. 5. Four-level images constructed from an original image, where each level is of full-resolution. Only one dimension (the x-axis) is shown for simplicity. Two pyramids at positions x and $x+1$ are depicted in solid line and dotted line, respectively.

their block sum pyramid (BSP) algorithm for fast motion vector estimation. The number of lower bounds for each search position is only $\log_2 B + 1$, instead of B^2.

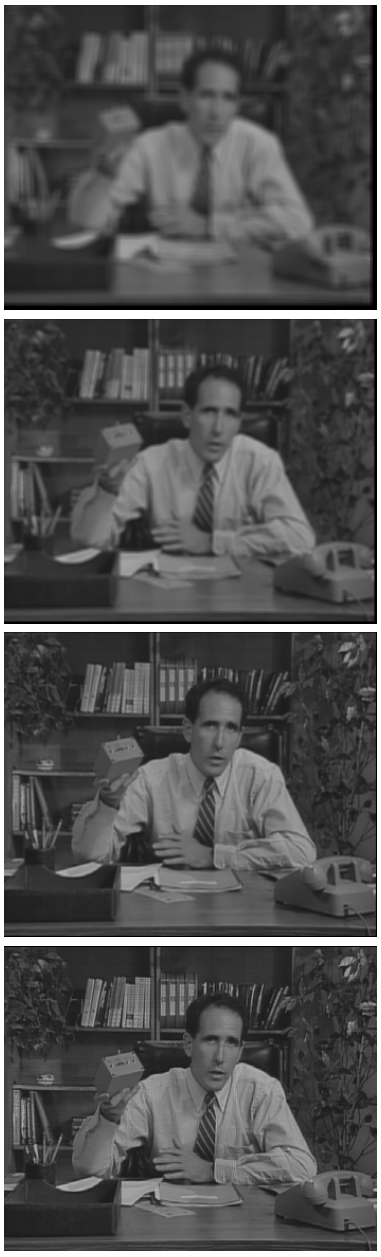
Assume the size of the matching block is $2^K \times 2^K$, we construct $K+1$ levels of images, where each level is of full-resolution, for each image in the video sequence by using the following way. Consider a four-level example as illustrated in Figure 5. For each pixel, ${I_t}^l(x,y)$, at level l and at time t, its value is assigned to be the sum of its four corresponding pixels at level $l+1$:

$$\begin{aligned} {I_t}^l(x,y) &= {I_t}^{l+1}(x,y) + {I_t}^{l+1}(x+2^{K-l-1},y) + \\ &\quad {I_t}^{l+1}(x,y+2^{K-l-1}) + \\ &\quad {I_t}^{l+1}(x+2^{K-l-1},y+2^{K-l-1}), \end{aligned} \tag{4}$$

where l is the level number, $l = 0,\ldots,K-1$, and ${I_t}^K$ is the original image at time t. Notice that the value of the pixel ${I_t}^l(x,y)$ at level l is in fact the sum of the corresponding $2^{K-l} \times 2^{K-l}$ pixels of the original image ${I_t}^K(x,y)$. By using Equation (4) to calculate ${I_t}^l(x,y)$ from level $l = K-1$ to 0, we can construct block sum pyramids for all the positions in the whole image, except for the last $2^{K-l}-1$ pixels at each row and each column due to the boundary condition. Figure 6 shows an example constructed from an image in the Salesman sequence.

Notice that there is overlap between neighboring block sum pyramids. Instead of calculating each block sum pyramid independently, our method can remove the redundant calculation existing in the construction of overlapping block sum pyramids. In Equation (4), it costs three operations to calculate the summation of four pixel values. Consequently, the overall computational cost to construct the K levels of images is about $K \times 3 \times W \times H$, where W and H are the width and height of the image, respectively. Notice that the computational cost is about $\frac{2^K \times 2^K - 1}{3} \times 3 \times W \times H$ if the block sum pyramids at all positions in the whole image are calculated independently.

Let the level-l matching error, $\mathrm{LSAD}^l_{(x,y)}(u,v)$, be the matching error between the template block at position (x,y) in image ${I_t}^l$ and the candidate matching block at position $(x+u, y+v)$ in image ${I_{t-1}}^l$. That is,


level-0 image, $I_t^0(x,y)$

level-1 image, $I_t^1(x,y)$

level-2 image, $I_t^2(x,y)$

level-3 image, $I_t^3(x,y)$

Fig. 6. Construction of the four-level images for the Salesman test sequence. Notice that the intensity values are normalized to $[0, 255]$ for each image.

$$\mathrm{LSAD}^l_{(x,y)}(u,v) \equiv \sum_{j=0}^{2^l-1}\sum_{i=0}^{2^l-1} |{I_t}^l(x+2^{K-l}i, y+2^{K-l}j) - {I_{t-1}}^l(x+u+2^{K-l}i, y+v+2^{K-l}j)|, \tag{5}$$

where $l = 0,\ldots,K$.

By using Minkowski's inequality, the following inequality relationship among the matching errors for different levels can be derived [16] (subscript (x,y) is dropped for simplicity):

$$\begin{aligned} \mathrm{LSAD}^0(u,v) \le \mathrm{LSAD}^1(u,v) \le \cdots &\le \mathrm{LSAD}^K(u,v) \\ &= \mathrm{SAD}(u,v). \end{aligned} \tag{6}$$

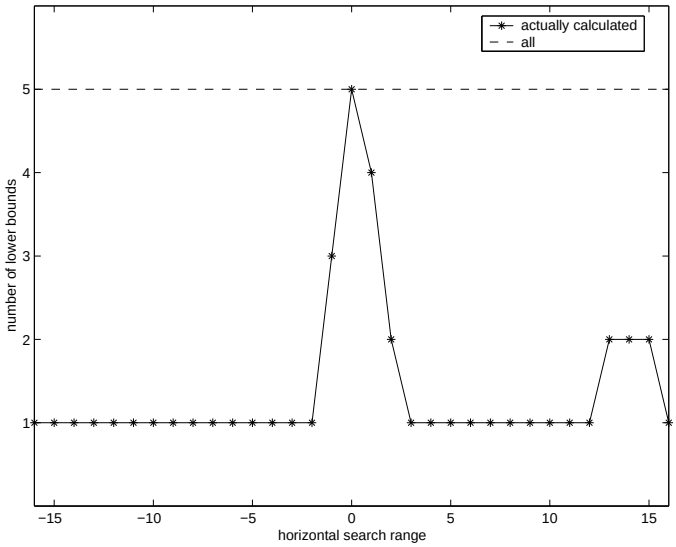


Fig. 7. Total number of lower bound (dashed line) and the number of lower bounds actually calculated (solid line).

The number of pixels used for calculating $\mathrm{LSAD}^l(u,v)$ at level l is $2^l \times 2^l$. More specifically, the number of pixels used for calculation is $1, 4, \ldots$, and $2^K \times 2^K$ at level $0, 1, \ldots$, and K, respectively. As a result, LSAD can be used as the lower bound of SAD and this kind of lower bound is referred to as the *pyramidal lower bound*. In the following, we use $\mathrm{LSAD}^l(u,v)$ as the lower bound $\mathrm{LB}^l(u,v)$ because there are only $K+1$ lower bounds in each lower bound list. Notice that the index of the first element in the lower bound list is changed to 0. The initialization process of the proposed winner-update algorithm has to be modified as the same. Also, the construction process of multilevel images has to be inserted into the proposed algorithm.

To give an intuitive feeling of how the computation can be greatly saved by using the pyramidal lower bound, we show the number of lower bounds computed with the example used in Figure 3. For this case, $K = 4$ (i.e., there are five lower bounds for each search position), but we can see from Figure 7 that the number of lower bounds needed to be computed is only one or two for most search positions. Most importantly, the computation required for computing the first lower bound is as simple as one-pixel operation. Consequently, the computational saving can be enormous.

III. Combination with the Three-Step Search Algorithm

The proposed winner-update algorithm can be easily combined with other fast algorithms for further speedup, but the guarantee of achieving the global minimum will be sacrificed. In this section, we shall use the well-known TSS algorithm as an example, where nine coarsely spaced search positions are examined at first. Then the position having the minimum matching error is selected and eight search positions which are less coarsely spaced around this selected position are examined. The matching errors of these eight positions and the previously selected position are compared and a new position having the minimum matching error is selected again. This process is repeated until the required spacing (one pixel) of the examined search positions is achieved.

At each iteration, the TSS algorithm needs to calculate and compare the matching errors of nine or eight search positions. Meanwhile the winner-update algorithm can be applied to efficiently find the one having the minimum matching error among these search positions. Because the winner-update algorithm can find the best matching position at each iteration, the accuracy of this combined algorithm is the same as that of the TSS algorithm while the computational efficiency will be much better.

IV. Experiments

A. Implementation Issues

A.1 Memory Storage

To utilize the inequality relationship in Equation (6), multilevel images for both the current and reference images have to be constructed. If the block size used is $2^K \times 2^K$, then $2K$ more images need to be stored in memory. Usually, one byte is sufficient to store the original pixel value ranging from 0 to 255. However, as Equation (4) shows, the value of each pixel in level l, $l = 0, \ldots, K-1$, is the sum of the $2^{K-l} \times 2^{K-l}$ pixel values in the block at level K. Two or more bytes has to be used to store the sum value. Thus more memory has to be allocated. Memory usage can be reduced by using one byte to store the pixel value at each level if the average, instead of the sum, of the pixel values in the block is stored. This can be accomplished by modifying Equations (4) and (5) into:

$$\begin{aligned} {I_t}^l(x,y) &= \frac{1}{4}({I_t}^{l+1}(x,y) + {I_t}^{l+1}(x+2^{K-l-1},y) + \\ &\quad {I_t}^{l+1}(x,y+2^{K-l-1}) + \\ &\quad {I_t}^{l+1}(x+2^{K-l-1},y+2^{K-l-1})), \end{aligned} \tag{7}$$

$$\begin{aligned} &\mathrm{LSAD}^l_{(x,y)}(u,v) \equiv \\ &\quad 2^{K-l}2^{K-l}\sum_{j=0}^{2^l-1}\sum_{i=0}^{2^l-1}|{I_t}^l(x+2^{K-l}i,y+2^{K-l}j) - \\ &\qquad {I_{t-1}}^l(x+u+2^{K-l}i,y+v+2^{K-l}j)|. \end{aligned}$$

However, this implementation may increase the minimum matching error a little because of the truncation error induced from Equation (7). For this reason, we did not use this modification in our experiments to ensure the global optimality.

A.2 Minimum Lower Bound Selection

The major overhead of the winner-update algorithm is the selection of the so-far-minimum lower bound at each iteration. A heap data structure can be used for selecting the minimum, which requires $O(\log_2 n)$ operations to maintain the heap structure that keeps the element having the smallest lower bound in the root of the heap tree.

The $O(\log_2 n)$ operations for minimum selection can be reduced to constant time by using a hashing method instead. If we use the SAD value as the address of the hashing table, the table size will be very large because the SAD value ranges from

0 to $256 \times 2^K 2^K$. To reduce the size of the hashing table from $256 \times 2^K 2^K$ to 256, the mean absolute difference (MAD) is used, instead of the SAD, and the pyramidal lower bound is calculated as

$$\mathrm{LB}^l_{(x,y)}(u,v) = \mathrm{MAD}^l_{(x,y)}(u,v) = \frac{1}{2^K 2^K}\mathrm{LSAD}^l_{(x,y)}(u,v).$$

Notice that the denominator for normalizing $\mathrm{LSAD}^l_{(x,y)}(u,v)$ in the above equation is the same for all ls. Thus, the increasing order of Equation (6) is preserved. That is, we can divide Equation (6) by $2^K 2^K$ and obtain the following inequality relationship:

$$\begin{aligned}\mathrm{MAD}^0(u,v) \le \mathrm{MAD}^1(u,v) \le \cdots &\le \mathrm{MAD}^K(u,v)\\ &= \mathrm{MAD}(u,v).\end{aligned}$$

Because $\mathrm{MAD}^l_{(x,y)}(u,v)$ ranges from 0.0 to 255.0, we only need a table of 256 elements and use the integer part of $\mathrm{MAD}^l_{(x,y)}(u,v)$ as the table address. Separate chaining is used in our implementation to resolve the collision. The first element in the first non-empty table entry is chosen to be the temporary winner. After updating the lower bound corresponding to this temporary winner, this element is put back to the corresponding table entry according to its new lower bound if it is not the minimum anymore. This operation requires only constant time. Once the element of the so-far-minimum lower bound reaches level K, all the remaining elements, if any, with the same table address (i.e., in the same chain) are examined to determine the final winner.

The cost of selecting the so-far-minimum lower bound can be further reduced by reducing the number of competitors. First, we calculate the complete matching error at the predicted search position (in our experiments, (u,v) is predicted as $(0,0)$). Then, the other search position can be avoided to be inserted into the hash table if its first lower bound has already been larger than the complete matching error calculated at the predicted search position. The less search positions join the competition, the faster selection can be obtained.

B. Experimental Results

To evaluate the performance of the winner-update algorithm, we implemented five algorithms: the full-search (FS) algorithm, the block sum pyramid (BSP) algorithm [16], the proposed winner-update algorithm with the lower bound derived from Minkowski's inequality (WinUpMI), the three-step search (TSS) algorithm [6], and the combination of the winner-update algorithm and the three-step search algorithm (WinUpTSS). Notice that the last two algorithms do not guarantee the global optimality. We evaluate the above five algorithms by comparing three performance indices:

1) the peak signal-to-noise ratio (PSNR) between the reconstructed motion-compensated image and the original image;

2) the number of absolute operations for calculating the matching error;

3) the execution time with our implementation and hardware environment.

These algorithms were implemented in C language on a Sun Ultra-1 workstation. The execution time includes reading images, multilevel images construction if necessary, and motion vector estimation. Five commonly-used image sequences, as shown in Figures 6 and 8, were used for comparing the performance of the algorithms. Each image was divided into 16×16 nonoverlapping blocks and the search range was set to $[-16,16] \times [-16,16]$.

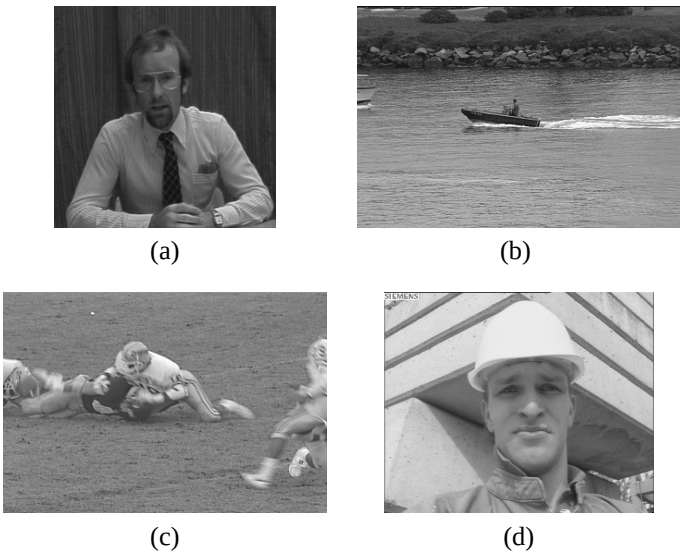


Fig. 8. Salesman test image sequence is shown in Figure 6. The other four test image sequences are (a) Trevor, (b) Coastguard, (c) Football, and (d) Foreman.

Table II shows the performance comparison of the above-mentioned algorithms with the Salesman image sequence which contains 100 images of size 352×288. The image in this sequence contains complex background as shown in Figure 6 and the PSNR value is high. Thus, only a few competitors with small matching error exist and the minimum matching error is small. Most search positions obtained larger lower bounds than the small minimum matching error and withdrew from the competition. Only 2% of the absolute operations are needed to calculate the matching errors, compared to the absolute operations for the FS algorithm. Even considering the overhead of constructing the multilevel images and switching the calculation among different search positions, the total execution time for estimating the motion vectors with the winner-update algorithm remain to be very small. Our experiments shows that the WinUpMI algorithm costs only 4% execution time of what the FS algorithm costs. As for the WinUpTSS algorithm, further speedup can be achieved without decreasing the PSNR value of the TSS algorithm. The number of the absolute operations decreases from 3.1% to 0.6% and the actual execution time decreases from 3.3% to 1.6%.

Table III shows the experimental results obtained with the Trevor image sequence which contains 99 images of size 256×256. As shown in Figure 8(a), the background of the images contains periodic strips. The blocks in the background area needed more computation to find the final corresponding block having the minimum matching error. Thus, higher operation ratio and execution time ratio were obtained compared to those for the Salesman image sequence.

The experiments described below use three image sequences containing larger motion. The first image sequence Coastguard

TABLE II
PERFORMANCE COMPARISON OF FIVE ALGORITHMS WITH SALESMAN IMAGE SEQUENCE

Algorithm	PSNR (dB)	Operations		Exec. Time	
		Number	%	sec.	%
FS	35.44	99847168	100.0	533.6	100.0
BSP	35.44	2823837	2.8	30.8	5.8
WinUpMI	35.44	1967831	2.0	21.2	4.0
TSS	35.15	3103744	3.1	17.8	3.3
WinUpTSS	35.15	627416	0.6	8.6	1.6

TABLE III
PERFORMANCE COMPARISON OF FIVE ALGORITHMS WITH TREVOR IMAGE SEQUENCE

Algorithm	PSNR (dB)	Operations		Exec. Time	
		Number	%	sec.	%
FS	34.42	62980096	100.0	359.7	100.0
BSP	34.42	2716395	4.3	31.5	8.8
WinUpMI	34.42	2445738	3.9	28.0	7.8
TSS	34.01	1971453	3.1	12.5	3.5
WinUpTSS	34.01	495151	0.8	7.0	2.0

contains 300 images of size 360×240, the second image sequence Football contains 60 images of size 352×240, and the third image sequence Foreman contains 400 images of size 360×288. Tables IV, V, and VI show the experimental results using the three image sequences. Because of the lower PSNR value, the efficiency improvement of the WinUpMI algorithm comparing to the FS algorithm is not as significant as that in the experiments of Salesman and Trevor image sequences. The minimum matching errors are larger, on the average, and more computations are required for most search positions before they are out of the competition for the winner. It is obvious that the proposed WinUpMI algorithm outperforms the BSP algorithm in these experiments because there is a lot of large motion in the image sequence (which makes it harder to have a good prediction).

Although the TSS algorithm and the WinUpTSS algorithm are faster among the five algorithms, they cannot guarantee the global optimality. From the above experiments, the TSS algorithm and the WinUpTSS algorithm decrease the PSNR values in a range from 0.29 dB to 1.26 dB. That is, they increase 6.9% to 33.7% of mean square error. Thus, more compensation effort is needed in video compression application.

TABLE IV
PERFORMANCE COMPARISON OF FIVE ALGORITHMS WITH COASTGUARD IMAGE SEQUENCE

Algorithm	PSNR (dB)	Operations		Exec. Time	
		Number	%	sec.	%
FS	28.51	83206656	100.0	1339.7	100.0
BSP	28.51	16157373	19.4	372.7	27.8
WinUpMI	28.51	6948447	8.4	178.8	13.4
TSS	28.07	2623309	3.2	44.6	3.3
WinUpTSS	28.07	1171328	1.4	37.4	2.8

TABLE V
PERFORMANCE COMPARISON OF FIVE ALGORITHMS WITH FOOTBALL IMAGE SEQUENCE

Algorithm	PSNR (dB)	Operations		Exec. Time	
		Number	%	sec.	%
FS	23.86	82258432	100.0	268.1	100.0
BSP	23.86	9649071	11.7	49.5	18.5
WinUpMI	23.86	5796655	7.1	32.1	12.0
TSS	23.10	2573771	3.1	8.8	3.3
WinUpTSS	23.10	1013051	1.2	6.8	2.5

TABLE VI
PERFORMANCE COMPARISON OF FIVE ALGORITHMS WITH FOREMAN IMAGE SEQUENCE

Algorithm	PSNR (dB)	Operations		Exec. Time	
		Number	%	sec.	%
FS	31.28	100998144	100.0	2209.1	100.0
BSP	31.28	10891459	10.8	366.8	16.6
WinUpMI	31.28	6456348	6.4	221.8	10.0
TSS	30.02	3169292	3.1	71.6	3.2
WinUpTSS	30.02	1257482	1.3	54.5	2.5

V. DISCUSSIONS

The major advantages of the winner-update algorithm include conceptual simplicity, easy implementation, and high efficiency. Also, it has many variations for adapting to different applications. For example, in the applications of on-line video compression, such as video conference and video phone, satisfying the time constraint is more important than acquiring the minimum matching error. For all the fast block matching techniques which guarantee the global optimality, the computational costs are all data dependent. When many search positions have the matching errors close to the minimum one, it needs much effort to identify the final winner from these tough competitors. This kind of bad situation occurs when the content of the matching block has no obvious features. On the other hand, the fast block matching techniques that examines only a portion but a fixed number of the search positions can give the motion vector in constant time, but the matching error may not be the global minimum. The winner-update algorithm can be slightly modified to provide a suboptimal match within a specified time interval. Once the time limit for performing the computation is met, the chain at the first non-empty entry of the hash table is checked and the element whose lower bound has been calculated to the finest level (i.e., the element (u, v) whose $\mathrm{LB}^{l(u,v)}(u, v)$ has the

largest $l(u, v)$) can be chosen as the matching result (i.e., the up-to-date best match).

In stereo vision and visual tracking applications, the winner-update algorithm can also be very helpful, but with another kind of modification. The matching process can be terminated earlier when the so-far-minimum lower bound has already been larger than a specified threshold value. The reason is that the matching result is unreliable (e.g. due to occlusion) and can be ignored. Moreover, an ordered list of the best candidate matches (the k-nearest neighbors) can be provided if we continue the matching process after the best match is found and removed from the search space. Further post-processing can be applied to the set of candidates for more accurate results.

The winner-update algorithm is suitable for serial computation. When multiple CPUs or computers are available, the task of block matching can be divided to enable parallel processing. In video compression application, for example, the motion vectors have to be estimated for all the blocks in the image. These blocks can be divided into groups so that each group is processed by a CPU or a computer. In this way, we can take the advantages of parallel processing and achieve further speedup.

VI. Conclusions

In this paper, we have proposed a new fast algorithm, named the winner-update algorithm, which can speed up the computation of block matching while still guaranteeing that the minimum matching error can be obtained. According to our experiments, the proposed winner-update algorithm (WinUpMI) can save 91.6% to 98.0% of the absolute operations needed by the FS algorithm, depending on which test image sequence is used. If the global optimum is not required, then the winner-update algorithm can combine with the TSS algorithm, and boost the saving up to 98.6% to 99.4% with a small decrease of PSNR value. The important thing is this combination (WinUpTSS) will speed up the TSS algorithm without decrease its PSNR value. When comparing the execution time of a fast algorithm, overhead needs to be considered. It is true that the WinUpMI algorithm requires a preprocessing overhead to construct multilevel images. Also, additional operations are needed to switch among the search positions. However, after including all the overhead, the efficiency of the winner-update algorithm (WinUpMI) remains still very good. According to our experiments, 86.6% to 96.0% of the execution time can be saved, compared to that of using the FS algorithm. Moreover, the proposed algorithm can be easily modified either to meet the limited time requirement or to provide an ordered list of the best candidate matches. Our source codes of the proposed algorithm are available at http://smart.iis.sinica.edu.tw/html/winup.html.

In addition to block matching, we are currently investigating a fast algorithm for k-nearest neighbor search by using the winner-update strategy. In this case, preprocessing is allowed and the overhead of constructing the multilevel images can be ignored. Another our research direction is to adopt Haar wavelet transform, instead of block sum pyramid method, to construct the multilevel images.

References

[1] ITU-T Recommendation H.261, "Video codec for audiovisual services at p×64 kbit/s," Mar. 1993.

[2] ITU-T Recommendation H.263, "Video coding for low bit rate communication," Feb. 1998.

[3] ISO/IEC 11172-2, "Information technology—coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s - part 2: video," 1993.

[4] ISO/IEC 13818-2 and ITU-T Recommendation H.262, "Information technology—generic coding of moving pictures and associated audio information: video," 1996.

[5] Ram Srinivasan and K. R. Rao, "Predictive coding based on efficient motion estimation," *IEEE Transactions on Communications*, vol. COM-33, no. 8, pp. 888–896, 1985.

[6] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion-compensated interframe coding for video conferencing," in *Proceedings of National Telecommunications Conference*, New York, 1981, vol. 4, pp. G5.3.1–G5.3.5.

[7] Jaswant R. Jain and Anil K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Transactions on Communications*, vol. COM-29, no. 12, pp. 1799–1808, 1981.

[8] M. Ghanbari, "The cross-search algorithm for motion estimation," *IEEE Transactions on Communications*, vol. 38, no. 7, pp. 950–953, 1990.

[9] Junavit Chalidabhongse and C.-C. Jay Kuo, "Fast motion vector estimation using multiresolution-spatio-temporal correlations," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 3, pp. 477–488, 1997.

[10] Bede Liu and André Zaccarin, "New fast algorithms for the estimation of block motion vectors," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 3, no. 2, pp. 148–157, 1993.

[11] Chang-Da Bei and Robert M. Gray, "An improvement of the minimum distortion encoding algorithm for vector quantization," *IEEE Transactions on Communications*, vol. COM-33, no. 10, pp. 1132–1133, 1985.

[12] Allen Gersho and Robert M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, London, 1992.

[13] Ho-Chao Huang and Yi-Ping Hung, "Adaptive early jump-out technique for fast motion estimation in video coding," *Graphical Models and Image Processing*, vol. 59, no. 6, pp. 388–394, 1997.

[14] W. Li and E. Salari, "Successive elimination algorithm for motion estimation," *IEEE Transactions on Image Processing*, vol. 4, no. 1, pp. 105–107, 1995.

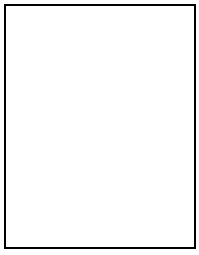
[15] Yih-Chuan Lin and Shen-Chuan Tai, "Fast full-search block-matching algorithm for motion-compensated video compression," *IEEE Transactions on Communications*, vol. 45, no. 5, pp. 527–531, 1997.

[16] Chang-Hsing Lee and Ling-Hwei Chen, "A fast motion estimation algorithm based on the block sum pyramid," *IEEE Transactions on Image Processing*, vol. 6, no. 11, pp. 1587–1591, 1997.

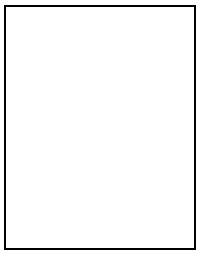
[17] Y. Q. Shi and X. Xia, "A thresholding multiresolution block matching algorithm," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, no. 2, pp. 437–440, 1997.

[18] Kwon Moon Nam, Joon-Seek Kim, Rae-Hong Park, and Young Serk Shim, "A fast hierarchical motion vector estimation algorithm using mean pyramid," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 5, no. 4, pp. 344–351, 1995.

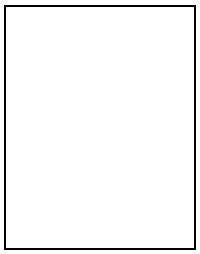
[19] Patrick Henry Winston, *Artificial Intelligence*, Addison-Wesley, Reading, Massachusetts, third edition, 1992.



Yong-Sheng Chen was born in Taiwan in 1971. He received his B.S. degree in computer and information science from National Chiao Tung University, Hsinchu, Taiwan, in 1993, and M.S. degree in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 1995. He is currently a Ph.D. candidate in the Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan, and a research assistant in the Institute of Information Science, Academia Sinica, Taipei, Taiwan. His research interests include fast template matching, pattern recognition, object detection, visual tracking, and video surveillance.



Yi-Ping Hung received his B.S. in electrical engineering from National Taiwan University in 1982. He received an M.S. from the Division of Engineering, an M.S. from the Division of Applied Mathematics, and a Ph.D. from the Division of Engineering, all at Brown University, in 1987, 1988 and 1989, respectively. He then joined the Institute of Information Science, Academia Sinica, Taiwan, and became a research fellow in 1997. He served as the deputy director of the Institute of Information Science from 1996 to 1997, and received the Outstanding Young Investigator Award given by Academia Sinica in 1997. He has been teaching in the Department of Computer Science and Information Engineering at National Taiwan University since 1990, where he is now an adjunct professor. Dr. Hung has published more than 70 technical papers in the fields of computer vision, pattern recognition, image processing, and robotics. In addition to the above topics, his current research interests also include visual surveillance, virtual reality, and human-computer interface.



Chiou-Shann Fuh received the B.S. degree in computer science and information engineering from National Taiwan University, Taipei, Taiwan, in 1983, the M.S. degree in computer science from the Pennsylvania State University, University Park, PA, in 1987, and the Ph.D. degree in computer science from Harvard University, Cambridge, MA, in 1992. He was with AT&T Bell Laboratories and engaged in performance monitoring of switching networks from 1992 to 1993. He was an associate professor in Department of Computer Science and Information Engineering, National Taiwan University, Taipei, Taiwan from 1993 to 2000 and then promoted to a full professor. His current research interests include digital image processing, computer vision, pattern recognition, and mathematical morphology.