In Proceedings of the Fourth Asian Conference on Computer Vision, Taipei, Taiwan, Jan. 2000, Volume 2, pages 977–982

A Fast Block Matching Algorithm Based on the Winner-Update Strategy

Yong-Sheng Chen†‡ Yi-Ping Hung†‡ Chiou-Shann Fuh‡

†Institute of Information Science, Academia Sinica, Taipei, Taiwan
‡Department of Computer Science and Information Engineering,
National Taiwan University, Taipei, Taiwan

ABSTRACT

Block matching is a popular and powerful technique for stereo vision, visual tracking, object recognition, and video compression. This paper presents a new fast algorithm, which is called the winner-update algorithm, for block matching. We utilize an ascending list of the lower bounds of the matching error for each search position. The calculation of the matching error can be avoided if one of its lower bound is larger than the globally minimum matching error. The winner-update algorithm computes the lower bounds only when the previous lower bounds in the same list are smaller than the globally minimum matching error. This algorithm can significantly speed up the computation of the block matching because (1) the computational cost of each lower bound is less than that of the matching error; and (2) for many search positions, only the first several lower bounds in the list need to be calculated. In the application to motion vector estimation, our experiments on test image sequences show that up to 98% of operations can be reduced while still guaranteeing the minimum matching error.

Keywords: block matching, fast algorithm, motion estimation.

1. INTRODUCTION

Block matching is a popular and powerful technique for stereo vision, visual tracking, object recognition, and video compression. In this paper, we focus on its application to motion vector estimation for video compression. Motion-compensated predictive coding has been widely used in the transmission and storage of video data [5, 6, 3, 4] for reducing the temporal redundancy. For this purpose, the image is usually divided into blocks and the displacement (motion vector) of each block from its corresponding block in the reference image is estimated and coded. The residual matching error between these two corresponding blocks are also coded in the data stream to retain the video quality.

Block matching technique is much used for estimating the block motion vector due to its simplicity. 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 often defined as the sum of absolute difference (SAD):

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

where B is the block size. The best estimate of the block motion vector, $(\hat{u}, \hat{v})$, locates the block at position $(x+\hat{u}, y+\hat{v})$ having the minimum matching error, $SAD_{(x,y)}(\hat{u}, \hat{v})$. This motion vector $(\hat{u}, \hat{v})$ can be obtained by using the full-search (FS) algorithm to calculate and compare the matching error for each search position in the reference image:

$$(\hat{u}, \hat{v}) = \arg\min_{(u,v)} SAD_{(x,y)}(u,v),$$

where $(u,v) \in \{(u,v) | -R \le u, v \le R\}$, R is the search range, and $(x+u, y+v)$ is a valid position in the reference image, I_{t-1}. This straightforward method takes extremely large amount of computation, although it can find the best matching block.

In the literature, many techniques have been developed to reduce the computational cost of block matching. These techniques can be classified into three categories. The techniques in the first category concern the search strategy, that is, they save the computations by reducing the number of positions searched. The gradient descent techniques, based on the unimodal error surface assumption, belong to this category. Well-known examples include the following: the three-step search (TSS) algorithm [8], the two-dimensional logarithmic search algorithm [7], and the conjugate direction search algorithm [15]. For another example, the search region can also be restricted to a small region located by the motion vectors of the spatially and temporally adjacent blocks together with the hierarchical related blocks [1].

The techniques in the second category, on the other hand, speed up the calculation of matching error for each search position. One simple way is to subsample the pixels in the matching blocks [12], thus only a part of the pixels are used for calculating the matching error. Another example is the so-called early jump-out technique [2], which can interrupt the process of matching error accumulation when the accumulated matching error is large enough comparing to a threshold sequence.

The techniques in the first and second categories are not exclusive and they can be combined to further improve the efficiency as described in [12, 14]. For another example, the hierarchical method [13] estimates the coarse result of the motion vector in the lower resolution image. Then the result is refined in the higher resolution image within a small search region centering at the coarse result. The major drawback of the techniques of the first two categories is that they cannot guarantee to achieve the minimum matching error because only a part of information is examined. Consequently, the best coding quality cannot be obtained.

The techniques in the third category use some matching criteria to eliminate search positions while still ensuring that the minimum matching error can be obtained [10, 11, 9]. These matching criteria are induced from Minkowski's inequality and their measures are smaller than or equal to the matching error. At first, the minimum matching error is initialized as the matching error calculated at the predicted position. For each search position other than the predicted one, the calculation of the matching error can be avoided if one of the measures of the matching criteria is larger than the up-to-date minimum matching error. Otherwise, the matching error is calculated and the up-to-date minimum matching error is updated if necessary. Because calculating the measure of the matching criteria costs less than calculating the matching error, the total amount of computation can be reduced.

When the motion vector is hard to predict, for example, if the image sequences contains objects with abrupt motion, the initial minimum matching error may be large. Hence much useless calculation of the matching criteria and the matching errors will be performed until a position with small matching error is examined. In the worst case, the matching errors are monotonically decreasing in the order that the search positions are examined. All the matching criteria are useless and the calculation of their measures is only waste of time. In such an unfortunate case, the computational cost will be higher than that of using the FS algorithm in fact.

In this paper, we propose an efficient and simple algorithm, named the winner-update algorithm, which can accelerate the computation of the block matching while still ensuring that the minimum matching error can be obtained. For each search position, an ascending list of the lower bounds of the matching error can be established. The calculation of the matching error can be avoided if one of the lower bounds is larger than the globally minimum matching error. The lower bound of the matching error can be derived from partial accumulation or Minkowski's inequality and its computational cost is less than that of the matching error. By using the proposed algorithm, very few lower bounds and matching errors are actually calculated. Consequently, the total computational cost can be significantly reduced. The algorithm does not need the prediction of the motion vector and it can avoid the useless calculation of the matching error which the techniques in the third category may suffer.

Moreover, this algorithm can be easily combined with the techniques in the first two categories for further speedup, but the guarantee of the minimum matching error is given up, of course. As an example, the combination of the three-step search algorithm and the proposed algorithm is also presented in this paper.

2. THE WINNER-UPDATE ALGORITHM

Concept

In this section, we use a simple game of poker cards to illustrate the concept of the winner-update algorithm. Suppose there are r players in the game and each player is dealt d cards. An example is shown in Figure 1 where $r = 5$ and $d = 4$. 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 before he can determine 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. 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.

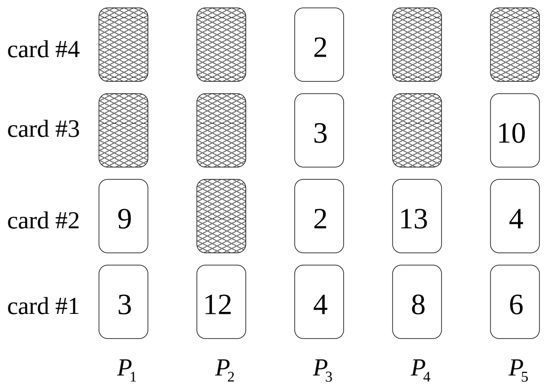

Figure 1. A simple game for illustrating the concept of the winner-update algorithm. There are five players, P_1, ..., P_5, and each player is dealt four cards. Player 3 has finished the accumulation of the values in his/her hand and becomes the winner. The temporary accumulations of others are all greater than the third player's final accumulation result. Hence their remaining calculation can be saved.

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. Only the temporary winner among the players having the minimum lower bound is allowed to turn up the face of the next card of 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 1 lists the operations step by step to demonstrate the process for the example given in Figure 1.

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 us-

operation/status	P_1	P_2	P_3	P_4	P_5
initialization	**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
P_3 is the winner	12	12	**11**	21	20

Table 1. Step by step calculation of the penalty score of each player. The score numbers in boldface are the temporary minima after each step.

ing this winner-update strategy, the amount of the cards laid facedown which represents the saved computations can be enormous.

Partial Accumulation

The process of choosing the winner having minimum penalty score in the previously mentioned game resembles the process of finding the corresponding block having minimum matching error in block matching. Each player in the game stands for a search position and the penalty score is the difference between the values of the corresponding pixels. From Eq.(1), we can define the partial accumulation of l pixels as:

$$PSAD^l_{(x,y)}(u,v) = \sum_{m=0}^{l} |I_t(x+i_m, y+j_m) - I_{t-1}(x+u+i_m, y+v+j_m)|,$$

where $\{(i_m, j_m)|m = 0, \ldots, B^2-1\}$ is the index set of all the pixels in the block. Obviously, the following inequality relationship holds true (subscript (x,y) is dropped for simplicity):

$$PSAD^0(u,v) \leq PSAD^1(u,v) \leq \cdots \leq PSAD^{B^2-1}(u,v) = SAD(u,v).$$

As a result, the partial accumulation of l pixels, $PSAD^l(u,v)$, can be defined as the lth element in the lower bound list, $LB^l(u,v)$. Notice that the last element in the list, $LB^{B^2-1}(u,v)$, equals the matching error, $SAD(u,v)$. These B^2-1 lower bounds are in ascending order:

$$LB^0(u,v) \leq LB^1(u,v) \leq \cdots \leq LB^{B^2-1}(u,v).$$

Furthermore, the computational cost of these lower bounds are ascending from 1 to B^2.

General Winner-Update Algorithm

For each block at position (x,y) in the current image, the lower bound of the matching error, $LB(u,v)$, between this block and the candidate matching block at position $(x+u, y+v)$ in the reference image is initialized as the first element in the lower bound list, $LB^0(u,v)$. An additional variable, $l(u,v)$, is used to record the index in the list that $LB(u,v)$ is assigned and it is initialized to 0. At first, the search position $(\hat{u},\hat{v})$ is chosen to be the temporary winner having minimum $LB(\hat{u},\hat{v})$. Then the temporary winner updates its lower bound $LB(\hat{u},\hat{v})$ with the next element in the lower bound list, $LB^{l(\hat{u},\hat{v})+1}(\hat{u},\hat{v})$, and records the new index number. The new search position $(\hat{u},\hat{v})$ having minimum $LB(\hat{u},\hat{v})$ is chosen again as the new temporary winner. This process is repeated until $l(\hat{u},\hat{v})$ of the new winner equals K, which denotes the number of the last element in the lower bound list. Remember that the last lower bound, $LB^K(\hat{u},\hat{v})$, equals the matching error $SAD(\hat{u},\hat{v})$ 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 given below:

The Winner-Update Algorithm

```
given template block at position $(x,y)$ in $I_t$
begin
  for each $(u,v)$ in the search range do
    begin (initialization)
      calculate $LB^0(u,v)$
      $LB(u,v) := LB^0(u,v)$
      $l(u,v) := 0$
    end
  select $(\hat{u},\hat{v})$ having minimum $LB(\hat{u},\hat{v})$ 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 $LB^{l(\hat{u},\hat{v})}(\hat{u},\hat{v})$
      $LB(\hat{u},\hat{v}) := LB^{l(\hat{u},\hat{v})}(\hat{u},\hat{v})$
      select $(\hat{u},\hat{v})$ having minimum $LB(\hat{u},\hat{v})$ to be the
        new temporary winner
    end
  output $(\hat{u},\hat{v})$
end
```

Minkowski's Inequality

Recently, Lee and Chen [9] developed the block sum pyramid (BSP) algorithm for fast motion vector estimation. They constructed a pyramid structure for each matching block and defined a matching criterion for each layer in the pyramid as mentioned in Section 1. This kind of matching criteria can be used as the lower bound we need and is presented below.

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

$$ {I_t}^l(x,y) = {I_t}^{l+1}(x,y) + {I_t}^{l+1}(x+2^{K-l-1}, y) + {I_t}^{l+1}(x, y+2^{K-l-1}) + {I_t}^{l+1}(x+2^{K-l-1}, y+2^{K-l-1}), \qquad (2)$$

where l is the layer 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 layer l is in fact the sum of the corresponding

$2^{K-l} \times 2^{K-l}$ pixels of the original image ${I_t}^K(x,y)$. In this way, we can construct the 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 3 shows an example of 4-layer images.

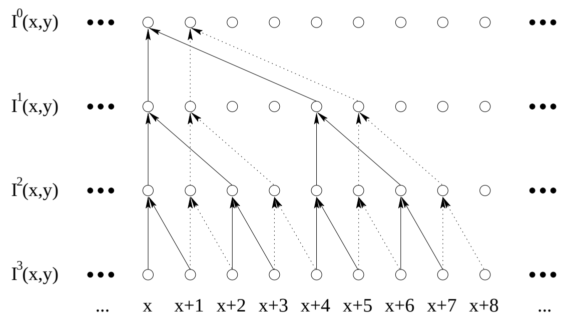

Figure 2. This figure illustrates the 4-layer images constructed from an original image, here each layer is of full-resolution. Only one dimension (the x-axis) is shown for simplicity. Two pyramids at position x and $x+1$ are depicted in solid line and dotted line, respectively.

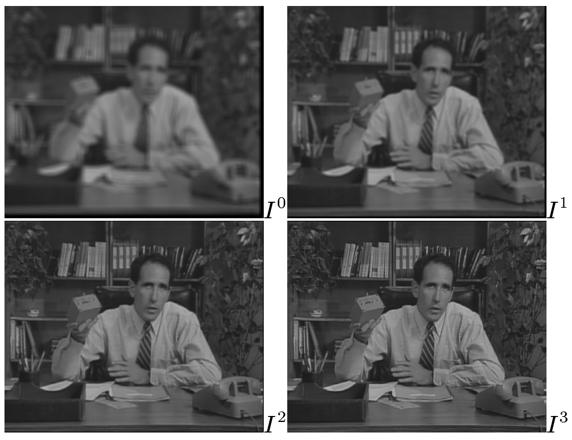

Figure 3. This figure presents the construction of 4-layer images.

At layer l, the matching error ${SAD_{(x,y)}}^l(u,v)$ between the block at position (x,y) in image ${I_t}^l$ and the candidate matching block at position $(x+u, y+v)$ in image ${I_{t-1}}^l$ is defined as:

$$
{SAD_{(x,y)}}^l(u,v) = \sum_{i=0}^{2^l-1}\sum_{j=0}^{2^l-1} |{I_t}^l(x+2^{K-l}i, y+2^{K-l}j) - {I_{t-1}}^l(x+u+2^{K-l}i, y+v+2^{K-l}j)|, \qquad (3)
$$

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

According to Minkowski's inequality (1-norm case here), Lee and Chen derived and proved the following inequality relationship between the matching errors computed at each layer:

$$
{SAD_{(x,y)}}^0(u,v) \le {SAD_{(x,y)}}^1(u,v) \le \cdots \le {SAD_{(x,y)}}^K(u,v). \qquad (4)
$$

The number of pixels used for calculating ${SAD_{(x,y)}}^l(u,v)$ at layer l is $2^l \times 2^l$, that is, the number is $1, 4, \ldots, 2^K \times 2^K$ when l is $0, 1, \ldots, K$, respectively. We use ${SAD_{(x,y)}}^l(u,v)$ defined in Eq.(3) as the lower bound $LB^l(u,v)$ of the matching error $SAD_{(x,y)}(u,v)$ at position (x,y).

3. 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 the minimum matching error is given up. Considering the well-known TSS algorithm, nine search positions which are coarsely spaced are examined at first. Then the position having 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 minimum matching error is selected again. This process is repeated until the required resolution (spacing) 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 minimum matching error among these search positions. The accuracy of this combined algorithm is the same as that of the TSS algorithm because the winner-update algorithm can find the best matching position at each iteration.

4. EXPERIMENTS

Implementation Issues

The major overhead of the winner-update algorithm is the selection of the minimum lower bound at each iteration. A heap data structure can be used in this algorithm and $O(\log_2 n)$ operations are used to maintain the heap condition and keep the element with the smallest lower bound in the root of the heap tree at each iteration.

The selection overhead can be reduced to constant time by using hashing method. To reduce the size of hashing table, the mean absolute difference (MAD) is used instead of SAD and the lower bound can be calculated as (subscript (x,y) is dropped for simplicity):

$$
\begin{aligned}
LB^l(u,v) &= MAD^l(u,v) \\
&= \frac{1}{2^K 2^K} SAD^l(u,v).
\end{aligned}
$$

Because $MAD^l(u,v)$ ranges from 0.0 to 255.0, a table with 256 elements is used and the table address is assigned with the integer part of $MAD^l(u,v)$. Separate chaining is used in our implementation to resolve the collision. The first block in the first non-empty table entry is chosen to be the temporary winner. After updating this block's new lower bound, it is moved, if necessary, to the corresponding table entry according to its new lower bound in constant time. When the selected block with minimum lower bound reaches layer 0, all the blocks with the same table address, that is, in the same chain, are examined to determine the final winner with the minimum matching error.

Experimental Results

This section shows the experimental results of applying the winner-update algorithm to some test image sequences. Five algorithms—the full-search (FS) algorithm, the block sum pyramid (BSP) algorithm [9], the proposed winner-update algorithm with the lower bound defined from Minkowski's inequality (WinUpMI), the three-step search (TSS) algorithm [8], and the combination of the winner-update algorithm and the three-step search algorithm (WinUpTSS)—were implemented. Notice that the last two algorithms do not guarantee that the global minimum can be found. We evaluate these algorithms by comparing three performance issues: (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; and (3) the execution time according to our implementation and hardware environment. These algorithms were implemented in C language on a Sun Ultra-1 workstation. The execution time includes reading images, multi-layer images construction if necessary, and motion vector estimation. As shown in Figure 4, five image sequences—Salesman, Trevor, Coastguard, Football, and Foreman—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 $[-16, 16] \times [-16, 16]$.

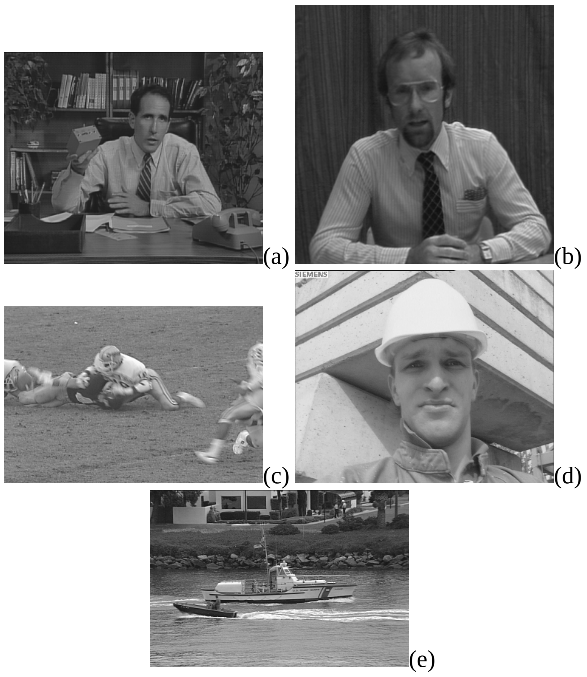

Figure 4. Five test image sequences: (a) Salesman, (b) Trevor, (c) Football, (d) Foreman, and (e) Coastguard.

Table 2 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 4(a) 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 1.97% of absolute operations are needed to calculate the matching errors, comparing to the absolute operations for the FS algorithm. Considering the overhead of constructing the multi-layer 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 3.97% 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 absolute operations decreases from 3.11% to 0.63% and the actual execution time decreases from 3.33% to 1.61%.

Algorithm	PSNR (dB)	Operations		Execution Time	
		number	%	sec.	%
FS	35.44	99847168	100.00	533.6	100.00
BSP	35.44	2823837	2.83	30.8	5.77
WinUpMI	35.44	1967831	1.97	21.2	3.97
TSS	35.15	3103744	3.11	17.8	3.33
WinUpTSS	35.15	627416	0.63	8.6	1.61

Table 2. Performance comparison of five algorithms with Salesman image sequence.

Table 3 shows the experimental results obtained with the Trevor image sequence which contains 99 images of size 256×256. As shown in Figure 4(b), 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 comparing to those for the Salesman image sequence.

Algorithm	PSNR (dB)	Operations		Execution Time	
		number	%	sec.	%
FS	34.42	62980096	100.00	359.7	100.00
BSP	34.42	2716395	4.31	31.5	8.76
WinUpMI	34.42	2445738	3.88	28.0	7.78
TSS	34.01	1971453	3.13	12.5	3.48
WinUpTSS	34.01	495151	0.79	7.0	1.95

Table 3. Performance comparison of five algorithms with Trevor image sequence.

The experiments described below use three image sequences containing larger motion. The first image sequence Coastguard 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 4, 5, and 6 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 and more computations are required to determine the best matching blocks. It is obvious that the proposed WinUpMI algorithm outperforms the BSP algorithm in these experiments especially when there is a lot of large motion in the image sequence.

Algorithm	PSNR (dB)	Operations		Execution Time	
		number	%	sec.	%
FS	28.51	83206656	100.00	1339.7	100.00
BSP	28.51	16157373	19.42	372.7	27.82
WinUpMI	28.51	6948447	8.35	178.8	13.35
TSS	28.07	2623309	3.15	44.6	3.33
WinUpTSS	28.07	1171328	1.41	37.4	2.79

Table 4. Performance comparison of five algorithms with Coastguard image sequence.

Algorithm	PSNR (dB)	Operations		Execution Time	
		number	%	sec.	%
FS	23.86	82258432	100.00	268.1	100.00
BSP	23.86	9649071	11.73	49.5	18.46
WinUpMI	23.86	5796655	7.05	32.1	11.97
TSS	23.10	2573771	3.13	8.8	3.28
WinUpTSS	23.10	1013051	1.23	6.8	2.54

Table 5. Performance comparison of five algorithms with Football image sequence.

Algorithm	PSNR (dB)	Operations		Execution Time	
		number	%	sec.	%
FS	31.28	100998144	100.00	2209.1	100.00
BSP	31.28	10891459	10.78	366.8	16.60
WinUpMI	31.28	6456348	6.39	221.8	10.04
TSS	30.02	3169292	3.14	71.6	3.24
WinUpTSS	30.02	1257482	1.25	54.5	2.47

Table 6. Performance comparison of five algorithms with Foreman image sequence.

5. 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 the minimum matching error. According to our experiments, the proposed WinUpMI algorithm can save 91.7% to 98% of absolute operations, depending on the image sequence. If the global optimum is not required, then WinUpTSS can save 98.6% to 99.4% of absolute operations without decreasing the PSNR value of using TSS algorithm. The WinUpMI algorithm requires a preprocessing stage to construct multi-layer images. Additional operations are also required to switch among the search positions. After considering these two kinds of overhead, the efficiency of the proposed algorithm is still very good. According to our experiments, 86.7% to 96% of the execution time can be saved, compared to that using the FS algorithm.

References

[1] J. Chalidabhongse and C.-C. J. Kuo. Fast motion vector estimation using multiresolution-spatio-temporal correlations. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(3):477–488, 1997.

[2] H.-C. Huang and Y.-P. Hung. Adaptive early jump-out technique for fast motion estimation in video coding. *Graphical Models and Image Processing*, 59(6):388–394, 1997.

[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.

[5] ITU-T Recommendation H.261. Video codec for audiovisual services at p$\times$64 kbit/s. Mar. 1993.

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

[7] J. R. Jain and A. K. Jain. Displacement measurement and its application in interframe image coding. *IEEE Transactions on Communications*, COM-29(12):1799–1808, 1981.

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

[9] C.-H. Lee and L.-H. Chen. A fast motion estimation algorithm based on the block sum pyramid. *IEEE Transactions on Image Processing*, 6(11):1587–1591, 1997.

[10] W. Li and E. Salari. Successive elimination algorithm for motion estimation. *IEEE Transactions on Image Processing*, 4(1):105–107, 1995.

[11] Y.-C. Lin and S.-C. Tai. Fast full-search block-matching algorithm for motion-compensated video compression. *IEEE Transactions on Communications*, 45(5):527–531, 1997.

[12] B. Liu and A. Zaccarin. New fast algorithms for the estimation of block motion vectors. *IEEE Transactions on Circuits and Systems for Video Technology*, 3(2):148–157, 1993.

[13] K. M. Nam, J.-S. Kim, R.-H. Park, and Y. S. Shim. A fast hierarchical motion vector estimation algorithm using mean pyramid. *IEEE Transactions on Circuits and Systems for Video Technology*, 5(4):344–351, 1995.

[14] Y. Q. Shi and X. Xia. A thresholding multiresolution block matching algorithm. *IEEE Transactions on Circuits and Systems for Video Technology*, 7(2):437–440, 1997.

[15] R. Srinivasan and K. R. Rao. Predictive coding based on efficient motion estimation. *IEEE Transactions on Communications*, COM-33(8):888–896, 1985.