Pergamon

Pattern Recognition, Vol. 31, No. 7, pp. 945–952, 1998
© 1998 Pattern Recognition Society. Published by Elsevier Science Ltd
All rights reserved. Printed in Great Britain
0031-3203/98 $19.00 + 0.00

PII: S0031-3203(97)00077-0

FAST FULL SEARCH IN MOTION ESTIMATION BY HIERARCHICAL USE OF MINKOWSKI'S INEQUALITY (HUMI)[†]

JING-YI LU, KUANG-SHYR WU and JA-CHEN LIN*

Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan 30010, R.O.C.

(*Received* 21 *March* 1996; *in revised form* 19 *November* 1996)

Abstract—In this paper, we extend the idea of successive elimination algorithm (SEA) to obtain a fast full search (FS) algorithm accelerating the block matching procedure of motion estimation. Based on the monotonic relation between the accumulated absolution distortions (AAD) obtained for distinct layers of a pyramid structure, the proposed method successfully rejects many impossible candidates considered in the FS. The derivation of the monotonicity relation repeatedly uses in a four-dimensional vector space the l_1-version of Minkowski's inequality, an inequality which is quite well-known in the field of mathematics. Simulation results show that the processing speed is faster than that of several well-known fast full search methods, including the SEA that uses just once the Minkowski's inequality (in a vector space of 256 dimension when the block size is 16×16). The processing speed of the proposed method is also competitive with that of the three-step search (TSS), which is often used for block matching in interframe video coding, although the visual quality performance of TSS is usually a little poorer than that of the FS. © 1998 Pattern Recognition Society. Published by Elsevier Science Ltd. All rights reserved

Motion estimation Block matching Position vector Motion vector
Accumulated absolute distortion Minkowski's inequality l_1-norm

1. INTRODUCTION

The block-matching algorithm (BMA) is popularly used for the motion estimation of interframe video coding. The ideal approach to BMA is the FS, which exhaustively searches for a block whose, say, accumulated absolute distortion (AAD) to an input block, is minimal among all possible blocks within a search window. Although the FS can find the optimal soltuion, i.e. the best-matched block mentioned above, the serious computation burden needed will make the hardware implementation difficult[(1)] if the window size becomes large (in the application to HDTV or super high-resolution TV). Many existing methods save computation but degrade the visual quality; examples include the TSS,[(2)] the orthogonal search,[(3)] or the two algorithms introduced in reference (4) which use alternating patterns. On the other hand, an elegant algorithm called the successive elimination algorithm (SEA) was developed recently in reference (5) by Li and Salari to alleviate the computation burder of FS while the visual quality was kept to be identical to that of the FS. Note that SEA used directly in a 256-dimensional vector space (if each block has size 16×16 and is treated as a 256-dimensional vector) the l_1-version of Minkowski's inequality—an inequality quite well-known in the field of mathematics[(6)]—to eliminate impossible candidate vectors. Another technique widely used in industry is to use the partial distortion elimination[(7)] (PDE) to eliminate impossible candidates at certain stages before completing the evaluation of the distortions from those impossible candidates to the discussed block. The PDE technique could also yield the visual quality identical to that of the FS.

In this paper, we extend the idea of SEA to obtain a pyramid based fast searching algorithm for the block matching. The monotonic relation between the AAD of distinct layers of the pyramid will be derived in Appendix A using in a four-dimensional vector space the l_1-version of Minkowski's inequality. With the derived monotonic relation, many impossible candidates can be kicked out without doing time-consuming computation. The proposed method keeps the good visual quality, i.e. the minimized AAD error, of the FS (and hence of the PDE and SEA), while the processing speed is greatly improved.

2. DEFINITIONS AND THE PROPOSED ALGORITHM

In this section, we define the AAD that will be used to measure the matching error for the block matching. For a specified block **G** of size $2^N \times 2^N$ [**G** will be referred to hereafter as the incoming (or input) block] in the current frame t, the corresponding AAD(x, y) is defined as

$$\mathrm{AAD}(x, y) = \sum_{i=1}^{2^N} \sum_{j=1}^{2^N} |f^t(i,j) - f^{t-1}(i+x, j+y)|. \tag{1}$$

* Author to whom correspondence should be addressed.
[†] This study was supported by the National Science Council, Republic of China, under contract number: NSC86-2213-E009-108.

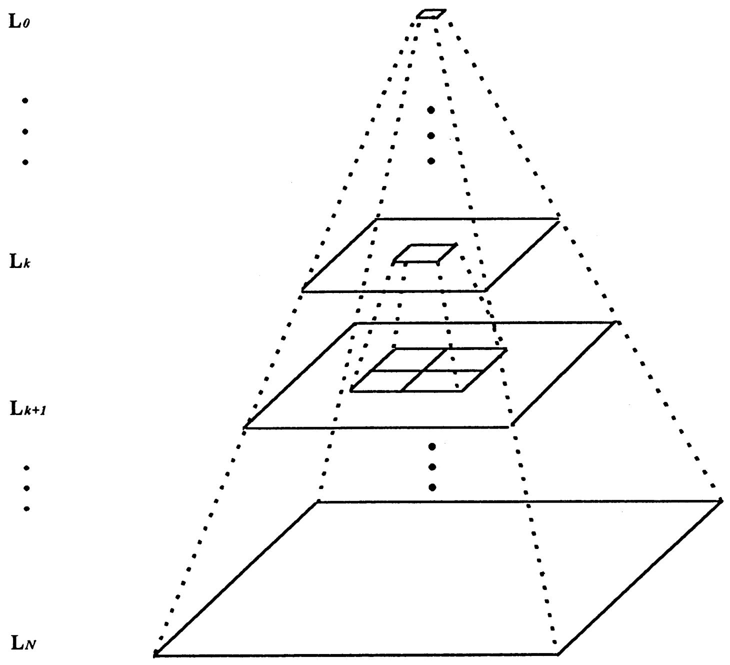

Fig. 1. The structure of the pyramid.

Here, $f^t(i,j)$ represents the gray value at pixel (i,j) of $\mathbf{G}$ in frame t, and $f^{t-1}(i+x, j+y)$ denotes the gray value at pixel $(i+x, j+y)$ in frame $t-1$. As for the vector (x, y), which is called the position vector, denotes the relative displacement of a candidate block (in the previous frame $t-1$) to the specified block $\mathbf{G}$ (in the current frame t).

In the BMA, the current frame t is often divided into non-overlapping blocks of size $2^N \times 2^N$ (16×16 in this paper, i.e. $N = 4$). The goal of the BMA is to find for each block $\mathbf{G}$ its best position vector (m, n) within a search window $\mathbf{W}$ (in this paper the window size is $(2^{N+1}+1) \times (2^{N+1}+1) = 33 \times 33$) such that

$$\mathrm{AAD}(m, n) = \min_{-2^N \le x,\, y \le 2^N} \mathrm{AAD}(x, y). \tag{2}$$

Hereafter, the position vector which is the best among all 33×33 position vectors $\{(x, y)\}$ will be called the "motion vector" (for the block $\mathbf{G}$). The basic concept to obtain the motion vector (m, n) quickly is to eliminate from the candidate space $\mathbf{W}$ those position vectors (x, y) which are impossible to be the best. We first review below a pyramid structure so that those impossible candidates can be eliminated layer by layer.

For every block $\mathbf{B}$, the corresponding pyramid of $(N+1)$ layers (from layer 0 to layer N) is a sequence $\{\mathbf{L}_0, \mathbf{L}_1, \ldots, \mathbf{L}_N\}$, with each layer defined as follows: the layer $\mathbf{L}_N$ of size $2^N \times 2^N$ is exactly the original block $\mathbf{B}$, the $\mathbf{L}_{N-1}$ of size $2^{N-1} \times 2^{N-1}$ is a size-reduced layer derived from $\mathbf{L}_N$, the $\mathbf{L}_{N-2}$ of size $2^{N-2} \times 2^{N-2}$ is a size-reduced layer derived from $\mathbf{L}_{N-1}, \ldots,$ and finally, the layer $\mathbf{L}_0$, which consists of a single pixel only, is a size-reduced layer derived from $\mathbf{L}_1$ (see Fig. 1). For each layer $\mathbf{L}_{k-1}$ (where $1 \le k \le N$), each pixel value $f_{k-1}(i,j)$ (where $1 \le i \le 2^{k-1}$ and $1 \le j \le 2^{k-1}$) is obtained by a summation of the 2×2 corresponding pixel values of $\mathbf{L}_k$. In other words, the pixel values are computed by

$$\begin{aligned} f_{k-1}(i,j) = {} & f_k(2i-1, 2j-1) + f_k(2i-1, 2j) \\ & + f_k(2i, 2j-1) + f_k(2i, 2j) \\ & \text{for } 1 \le i,j \le 2^{k-1} \text{ and } 1 \le k \le N. \end{aligned} \tag{3}$$

[To save computation, we do not want to divide the right-hand side of equation (3) by 4, although the division will make the value of $f_{k-1}(i,j)$ staying in the range 0–255.] Since each layer $\mathbf{L}_k$ can be considered as a (low-resolution) image (except that the pixel values are in fact the so-called "generalized" gray values because they are not in the range 0–255), the $\mathrm{AAD}_k(x, y)$ between layer $\mathbf{L}_k$ of an input $\mathbf{G}$, and layer $\mathbf{L}_k$ of the block corresponding to a motion vector candidate (x, y), can be defined by generalizing equation (1), namely,

$$\mathrm{AAD}_k(x, y) = \sum_{i=1}^{2^k} \sum_{j=1}^{2^k} \left| f_k^t(i,j) - f_k^{[t-1;(x,y)]}(i,j) \right| \quad \text{for } 0 \le k \le N. \tag{4}$$

Here, the $f_k^t(i,j)$ represents the (generalized) gray value at the (low-resolution) pixel (i,j) in layer $\mathbf{L}_k$ constructed from the $2^N \times 2^N$ block $\mathbf{G}$ in frame t; and $f_k^{[t-1;(x,y)]}(i,j)$ denotes the (generalized) gray value at the (low resolution) pixel (i,j) in layer $\mathbf{L}_k$ constructed from a $2^N \times 2^N$ block in frame $t-1$, with the center of that $2^N \times 2^N$ block being (x, y) units away from the center of the $2^N \times 2^N$ block $\mathbf{G}$.

The relation among the AAD of distinct layers is introduced below. For two arbitrarily given sets $\mathbf{P} = \{p_1, p_2, \ldots, p_z\}$ and $\mathbf{Q} = \{q_1, q_2, .., q_z\}$ of non-negative real numbers, where **P** and **Q** have the same number of elements, the l_1-version of the very famous Minkowski's inequality

$$\|\mathbf{P} - \mathbf{Q}\| \geq |\|\mathbf{P}\| - \|\mathbf{Q}\||$$

[see Theorem 8.13 of reference (6) for the definition and proof of the infinitely many versions l_1–l_∞ of the Minkowski's inequality] implies that

$$\sum_{i=1}^{z} |p_i - q_i| = \|\mathbf{P} - \mathbf{Q}\| \geq |\|\mathbf{P}\| - \|\mathbf{Q}\|| = \left|\sum_{i=1}^{z} |p_i| - \sum_{i=1}^{z} |q_i|\right| = \left|\sum_{i=1}^{z} p_i - \sum_{i=1}^{z} q_i\right|. \quad (5)$$

This well-known fact can also be interpreted geometrically as: in the l_1-norm space (also know as the city-block-norm space), the triangular inequality still holds (the length of the third edge ($p\varpi q\varpi$) is not shorter than the difference of the lengths $\|p\varpi\|_{l_1}$ and $\|q\varpi\|_{l_1}$ of the two edges $p\varpi = (p_1, p_2, \ldots, p_z)$ and $q\varpi = (q_1, q_2, \ldots, q_z)$. The assumption $p_i \geq 0$ and $q_i \geq 0$ for all i makes $\|p\varpi\|_{l_1} = \sum |p_i| = \sum p_i$ and $\|q^{\varpi}\|_{l_1} = \sum |q_i| = \sum q_i$ when we evaluate the l_1-norm).

According to inequality (5), the inequality

$$\mathrm{AAD}_k(x, y) \geq \mathrm{AAD}_{k-1}(x, y) \quad \text{for } 1 \leq k \leq N \quad (6)$$

can be proved (the proof is given in Appendix A). From the pyramid structure and equation (6), we can efficiently search for the motion vector, i.e. the best position vector. Without loss of generality, we assume that a part of the $33 \times 33 = 1089$ position vectors in **W** have been inspected; the "so far" best position vector is stored in (m, n); and the corresponding distortion AAD(m, n) is stored in $\mathrm{AAD}_{\min}$. Then the following corollary is always true:

Corollary. If $\mathrm{AAD}_k(x, y) \geq \mathrm{AAD}_{\min}$ for some $k \in \{0, 1, \ldots, N\}$, then

$$\mathrm{AAD}(x, y) \geq \mathrm{AAD}_{\min},$$

i.e., in the previous frame $t - 1$, the block obtained by translating the input block (x, y) units will not be better than the block obtained by translating the input block (m, n) units.

This corollary comes directly from equation (6): because $\mathrm{AAD}_N(x, y)$ is in fact $\mathrm{AAD}(x, y)$ and $\mathrm{AAD}_N(x, y) \geq \mathrm{AAD}_{N-1}(x, y) \geq \ldots \geq \mathrm{AAD}_0(x, y)$ always holds by equation (6), we therefore obtain

$$\mathrm{AAD}(x, y) = \mathrm{AAD}_N(x, y) \geq \mathrm{AAD}_k(x, y) \geq \mathrm{AAD}_{\min}.$$

Hence, (x, y) could not be better than (m, n), and thus, (x, y) should be rejected. Note that the computation of $\mathrm{AAD}_k(x, y)$ grows when k increases. More precisely, the computation load of $\mathrm{AAD}_k(x, y)$ is about four times heavier than that of $\mathrm{AAD}_{k-1}(x, y)$. Therefore, the proposed algorithm uses the "top-down" approach (from $\mathbf{L}_0$ to $\mathbf{L}_N$) to reject impossible candidates (x, y) layer by layer. Some are rejected in $\mathbf{L}_0$, some are rejected in $\mathbf{L}_1$, etc. Of course, the overhead to construct the pyramid (the construction is bottom-up, however) should also be considered. But, we found that, even with this overhead, the total CPU time is still saved somewhat using the above corollary.

Note that if we let $z = 2^N \times 2^N = 2^{2N}$ in equation (5), then

$$\mathrm{AAD}(x, y) = \sum_{1 \leq i, j \leq 2^N} |f^t(i, j) - f^{t-1}(i + x, j + y)| \geq \left| \sum_{1 \leq i, j \leq 2^N} |f^t(i, j)| - \sum_{1 \leq i, j \leq 2^N} |f^{t-1}(i + x, j + y)| \right|$$

by the definition (1) of AAD(x, y). SEA therefore eliminates a position vector (x, y) if

$$\left| \sum_{1 \leq i, j \leq 2^N} |f^t(i, j)| - \sum_{1 \leq i, j \leq 2^N} |f^{t-1}(i + x, j + y)| \right| \geq \mathrm{AAD}_{\min}$$

[whereas we eliminate (x, y) if $\mathrm{AAD}_k(x, y) \geq \mathrm{AAD}_{\min}$ for some $k \in \{0, 1, \ldots, N\}$].

For the reader's benefit, we give below the algorithm that utilizes the proposed method to eliminate those "impossible position vectors" for a given **G**. Note that the motion vector obtained for the corresponding block (i.e. the block having identical location as **G**) in frame $t - 1$ is used as the initial guess for the motion vector (m, n) of **G** of the current frame t.

Algorithm

Goal: Find the motion vector, i.e. the best position vector within a search window **W** in the previous frame $t - 1$ for an incoming block **G** in the current frame t.

Input: 1. An incoming block **G** of size $2^N \times 2^N = 16 \times 16$ in frame t.
2. $33 \times 33 = 1089$ candidates, i.e. 1089 position vectors, in **W** of frame $t - 1$.

Output: The motion vector (m, n), i.e. the position of the best-matched block.

Initial conditions: (1) Assume that the pyramids for all blocks in frame $t - 1$ have been established. (The overhead is only about 12 MH operations for the frame of size $M \times H$, as is shown in Appendix B).
(2) Let the initial guess of the motion vector (m, n) be the motion vector obtained for the corresponding block of **G** in frame $t - 1$.

Table 1. Performance comparison among the five algorithms using different image sequences. NOAE meant the "number of AAD evaluations" needed for each block, although CPU time was for the whole sequence (150, 300, and 168 frames, respectively)

Images	Algorithms	NOAE* per block	PSNR(db) per block	AAD value per block	Total CPU time†
Suzie	FS	1089	38.804	650.2	343′30″
	PDE	227	38.804	650.2	123′47″
	SEA	316	38.804	650.2	108′16″
	ours	97	38.804	650.2	38′45″
	TSS	33	38.188	721.4	22′40″
Salesman	FS	1089	37.553	703.6	855′5″
	PDE	237	37.553	703.6	188′13″
	SEA	253	37.553	703.6	216′54″
	ours	22	37.553	703.6	34′45″
	TSS	33	37.457	718.8	55′18″
Claire	FS	1089	47.671	231.7	470′0″
	PDE	282	47.671	231.7	126′10″
	SEA	176	47.671	231.7	88′17″
	ours	32	47.671	231.7	26′5″
	TSS	33	47.65	233.5	30′11″

* Preprocessing overhead (e.g. building the pyramids) was not included in NOAE, although the overhead for elimination test (e.g. Step 5 of our algorithm) was included.
† CPU time included all overheads (preprocessing, elimination test, etc.).

Steps:

Step 1: Use Formula (3) to construct the pyramid for **G**. [This construction uses additions only, and the number of additions used is not more than the additions used in equation (1). Hence, doing Step 1 takes shorter CPU time than doing Step 2.]

Step 2: Evaluate the AAD for the initial guess (m, n) and let $\text{AAD}_{\min} = \text{AAD}(m, n)$.

Step 3: If all position vectors have been checked, then go to Step 8.

Step 4: Pick up from the search window **W** a position vector (x, y) which is not processed yet by Step 5.

Step 5: For $k = 0$ to N do
  if $(\text{AAD}_k(x, y) \geq \text{AAD}_{\min}$ for the current value of $k)$ then delete (x, y) from **W** and go to Step 3.

Step 6: Replace the contents of $\text{AAD}_{\min}$ and (m, n) by $\text{AAD}_N(x, y)$ and (x, y), respectively. In symbol, $\text{AAD}_{\min} \leftarrow \text{AAD}_N(x, y)$, $m \leftarrow x$, and $n \leftarrow y$. (This step updates the "so far" minimal AAD and the "so far" best position vector.)

Step 7: Go to Step 3.

Step 8: Print out the final value of (m, n) because its content now is indeed the motion vector.

3. EXPERIMENTAL RESULTS

In this section, the FS, PDE, SEA, TSS, and the proposed method are compared. Note that the partial distortion elimination (PDE) technique widely used in industry is to check the rth partial AAD

$$\sum_{i=1}^{r} \sum_{j=1}^{2^N} |f^t(i,j) - f^{t-1}(i + x, j + y)| \tag{7}$$

for $r = 1, 2, 3, \ldots$ [see equation (1)] against the current $\text{AAD}_{\min}$ during the matching; should there exists an $r < 2^N$ making the partial AAD exceed the current $\text{AAD}_{\min}$, then quit the summation process in equation (1) and kick out the impossible candidate vector (x, y) because it is trivial that the full sum in equation (1) would certainly be larger than equation (7), and hence larger than the current $\text{AAD}_{\min}$. Tables 1 and 2 illustrate the performance of the FS, PDE, SEA, TSS and the proposed method by comparing the NOAE (number-of-AAD-evaluations), total CPU time, AAD and PSNR values. Table 1 used three 288×352 luminance video sequences (Suzie, Salesman, and Claire), whereas Table 2 used three 176×144 luminance video sequences (Car phone, Foreman, and Mother-and-daughter) grabbed from the International Telecommunication Union. The total CPU time shown in Tables 1 and 2 includes the overhead. (Note that the FS, PDE and TSS have no preprocessing while the SEA and the proposed method have.) It is easy to see that the proposed method outperforms the other four methods. As for the values of AAD or PSNR, it can be seen that PDE, SEA and the proposed method produce the same values as those of the FS. This should be of no surprise because they are just some kinds of fast implementation of the FS. The TSS, however, has worse values in both AAD and PSNR, because TSS is not a kind of FS. (The TSS uses a hard rule to by-pass many candidates without even giving these candidates a chance of being checked. Unfortunately, it happens quite often that some of these candidates by-passed by TSS are the ones that yield optimized match.)

In the column NOAE in Tables 1 and 2, the works to evaluate the AAD_k [defined in equation (4), $0 \leq k \leq N$] for our method were all collected together and expressed in terms of the work needed to

Table 2. Same as Table 1 except that other data were used. CPU time was for the whole sequence (382, 400, and 961 frames, respectively)

Images	Algorithms	NOAE* per block	PSNR(db) per block	AAD value per block	Total CPU time†
Car phone	FS	1089	36.32	829.44	273′38″
	PDE	227	36.32	829.44	55′6″
	SEA	190	36.32	829.44	51′11″
	ours	21	36.32	829.44	11′55″
	TSS	33	36.1	865.28	16′44″
Foreman	FS	1089	36.43	793.6	286′21″
	PDE	237	36.43	793.6	61′54″
	SEA	186	36.43	793.6	54′38″
	ours	27	36.43	793.6	12′42″
	TSS	33	35.8	896.1	17′44″
Mother & daughter	FS	1089	42.74	419.84	602′10″
	PDE	182	42.74	419.84	114′14″
	SEA	115	42.74	419.84	86′47″
	ours	29	42.74	419.84	35′44″
	TSS	33	42.71	424.96	43′42″

evaluate AAD_N. (If we consider the number of pixels in each layer, we can find that the work of each evaluation of AAD_N is approximately the work $4^1 = 4$ evaluations of AAD_{N-1}, or, the work of $4^2 = 16$ evaluations of AAD_{N-2}, etc.) Since AAD_N is in fact the traditional AAD defined in equation (1), the collection we just obtained can also be said to be expressed in terms of the work needed to evaluate the traditional AAD. Therefore, this number was entered for out method in the column with the title NOAE. As for the NOAE of the PDE method, analogous procedure had been used to express the work for the rth partial AAD [see equation (7)] in terms of that for the full AAD [see equation (1)]. The detail is trivial and hence not explained here.

Note especially that, for the image sequence Salesman listed in Table 1, averagely speaking [the average was taken not only between the $(288/16)\times(352/16)$ blocks $\mathbf{G}$ contained in a frame, but also between the 300 frames of the image sequence], only 22 evaluations of the traditional AAD, i.e. AAD_4, were needed. This number is even smaller than that of the TSS. The advantage still exists even if the overhead is also included. (In fact, for 300 frames, the total CPU time used in our workstation was 34 min for the proposed method, and, 55 min for the TSS.) We explain below how the number "22" was obtained for the image sequence Salesman. For a given block $\mathbf{G}$, there were $(33)^2 = 1089$ candidate blocks. One of them was used as the initial guess, and the remaining 1088 candidates were examined layer by layer. Averagely speaking, the number of candidates kicked out in layers 0, 1, 2, and 3, were 837, 195 40, and 11, respectively. In other words, only $1088 - 837 - 195 - 40 - 11 = 5$ candidates reached Layer 4. Note that, by Step 5 of the algorithm, if a candidate needs to evaluate AAD_k, then, this candidate also needs to evaluate $\{\mathrm{AAD}_0, \mathrm{AAD}_1, \ldots, \mathrm{AAD}_{k-1}\}$. Therefore,

$837 + 195 + 40 + 11 + 5 = 1088$ candidates needed to evaluate AAD_0;

$195 + 40 + 11 + 5 = 251$ candidates needed to evaluate AAD_1;

$40 + 11 + 5 = 56$ candidates needed to evaluate AAD_2;

$11 + 5 = 16$ candidates needed to evaluate AAD_3;

and 5 candidates needed to evaluate AAD_4.

Since an evaluation of AAD_4 equals, as stated in the previous Section, $4^1 = 4$ evaluations of AAD_3, or equivalently, $4^2 = 16$ evaluations of AAD_2, etc., we have

$$1088/4^4 + 251/4^3 + 56/4^2 + 16/4 + 5 = 21$$

evaluations of AAD_4. Together with the AAD evaluation used for the initial guess (see Step 2 of the algorithm), we have $21 + 1 = 22$ evaluations of AAD_4.

For the reader's benefit, we also listed in Table 3 the computational cost needed for each frame. Here, we assume that there are $B = (M/16)\times(H/16)$ blocks in the frame; and hence, there are about $1089B$ position vectors to be processed. Since $MH = 256B$, $4MH$ and $12MH$ appearing in Table 3 for SEA and our methods equal $1024B$ and $3072B$, respectively. Also note that if the image sequence is the Salesman, then the $\{m_j\}_{j=0}^{4}$ mentioned in Table 3 for our method are $m_0 = 1088B$, $m_1 = 251B$, $m_2 = 56B$, $m_3 = 16B$, and $m_4 = 5B + 1B = 6B$, respectively, by the paragraph right above. (The $1B$ appearing in m_4 is due to the $\mathrm{AAD}[= \mathrm{AAD}_4]$ evaluation of the initial guess, as stated at the end of last section.) Therefore,

$$\sum_{j=0}^{4} 4^j m_j = \cdots = 256\left[\frac{1088B}{256} + \frac{251B}{64} + \frac{56B}{16} + \frac{16B}{4} + \frac{5B + 1B}{1}\right]$$

Table 3. Computation cost per frame (Assume there are $B = \frac{1}{16}M + \frac{1}{16}H$ blocks; hence, about $1089B$ position vectors and each vector is 256-dim.)

Methods	$-$	Abso. value	$+$	Compare	Remarks
FS	$256 \times 1089B$	$256 \times 1089B$	$255 \times 1089B$	1088B	
PDE	$16 \sum_{j=1}^{16} jn_j$ $<256 \times 1089B$	$16 \sum_{j=1}^{16} jn_j$ $<256 \times 1089B$	$\sum_{j=1}^{16} (16j-1)n_j$ $<255 \times 1089B$	$\sum_{j=1}^{16} jn_j$ $>1089B$	(1)
SEA	0	0	$4MH$	0	(2α)
	$+1088B$	$+1088B$	$+0$	$+1088B$	(2β)
	$+256P$	$+256P$	$+255P$	$+1P$	(2γ)
ours	0	0	$12MH$	0	(3α)
	$+\sum_{j=0}^{4} 4^j m_j$	$+\sum_{j=0}^{4} 4^j m_j$	$+\sum_{j=0}^{4} (4^j-1)m_j$	$+\sum_{j=0}^{4} 4^j m_j$	(3β)
TSS	$256 \times 33B$	$256 \times 33B$	$255 \times 33B$	$32B$	(4)

Note. (1) Note $\sum_{j=1}^{16} n_j = 1089B$. Here, n_j $(1 \le j \le 15)$ is the number of position vectors that was eliminated by the PDE test after summing up only j of the 16 rows, and n_{16} corresponds to the number of vectors that need the total evaluations of AAD.
(2α) For l_1-norm set-up ($4MH = 1024B$).
(2β) For l_1-norm elimination test.
(2γ) For the AAD evaluations of the P position vectors not eliminated by the l_1-norm test. (Note $P \le 1089B$. In fact, P seldom exceeds $350B$.)
(3α) For pyramids set-up ($12MH = 256 \times 12B$)
(3β) For the kick-out test ($\mathrm{AAD}_j, j = 0, 1, 2, 3$) and traditional AAD ($= \mathrm{AAD}_4$) evaluation. Note m_j is the number of position vectors that need to evaluate AAD_j. Also note $m_0 = (1089 - 1)B = 1088B > m_1 > m_2 > m_3 > m_4$. (See the Salesman example given in the last two section of the text. Note $m_1 = 251B$, $m_2 = 56B$, $m_3 = 16B$, $m_4 = 5B + 1B = 6B$, and $\sum_{j=0}^{4} 4^j m_j \approx 256 \times 22B$ there by the text around equation (8).)
(4) The authors of reference (5) used 33 of the 1089 position vectors as TSS candidates for each block. We also used 33 here.

$$= 256[4.25B + 3.92B + 3.5B + 4B + 6B] \approx 256 \times 22B \ll 256 \times 1089B \tag{8}$$

indicates that the subtractions, absolute value evaluations, and additions needed for ours are much less than those for FS. As for the comparison operations, although FS used fewer comparisons, this advantage of FS is nothing to the whole work if we notice that just the number of subtractions used by FS ($256 \times 1089B$) is much greater than the total operations (including the comparisons) needed by ours $[256 \times (22B \times 4 + 12B) = 256 \times 100B]$.

4. CONCLUSIONS

In this paper, the idea of SEA is extended to obtain a fast searching algorithm for the block matching of motion estimation. With the image reconstruction quality identical to that of the full search, the proposed method uses the monotonicity property (6) of the AAD relation between adjacent layers of the pyramid structure to alleviate the search burden of the FS. Note that the derivation of the monotonicity property (6) used in a four-dimensional vector space the l_1-version of Minkowski's inequality which is quite well-known in the field of mathematics. The experiment results showed that the proposed method really reduced the computations needed. The method outperformed both the SEA method (which applied the l_1-version of Minkowski's inequality to a 256-dimensional vector space if block size is 16×16) and the PDE method commonly used in industry. Moreover, the computation speed was competitive with (often even better than) that of the TSS, although the TSS usually obtained non-optimal position vector only. As a final remark, note that the AAD we tried to minimize [see equation (1)] is in fact a kind of l_1-norm which is easier-to-compute than the mean square error (MSE) that many researchers used.

Acknowledgement—The authors wish to thank the reviewer for giving helpful suggestions.

APPENDIX A

The statement $\mathrm{AAD}_{k+1}(x, y) \ge \mathrm{AAD}_k(x, y)$ holds for all $k \in \{0, 1, \ldots, N-1\}$.

Proof. By (3)–(5), we get

$$\mathrm{AAD}_{k+1}(x, y) = \sum_{i=1}^{2^{k+1}} \sum_{j=1}^{2^{k+1}} |f_{k+1}^{t}(i, j) - f_{k+1}^{[t-1;(x,y)]}(i, j)| \tag{A1}$$

$$= \sum_{i=1}^{2^k} \sum_{j=1}^{2^k} \left(\sum_{a=0}^{1} \sum_{b=0}^{1} |f_{k+1}^{t}(2i-a, 2j-b) - f_{k+1}^{[t-1;(x,y)]}(2i-a, 2j-b)| \right) \tag{A2}$$

$$\ge \sum_{i=1}^{2^k} \sum_{j=1}^{2^k} \left| \sum_{a=0}^{1} \sum_{b=0}^{1} f_{k+1}^{t}(2i-a, 2j-b) - \sum_{a=0}^{1} \sum_{b=0}^{1} f_{k+1}^{[t-1;(x,y)]}(2i-a, 2j-b) \right| \tag{A3}$$

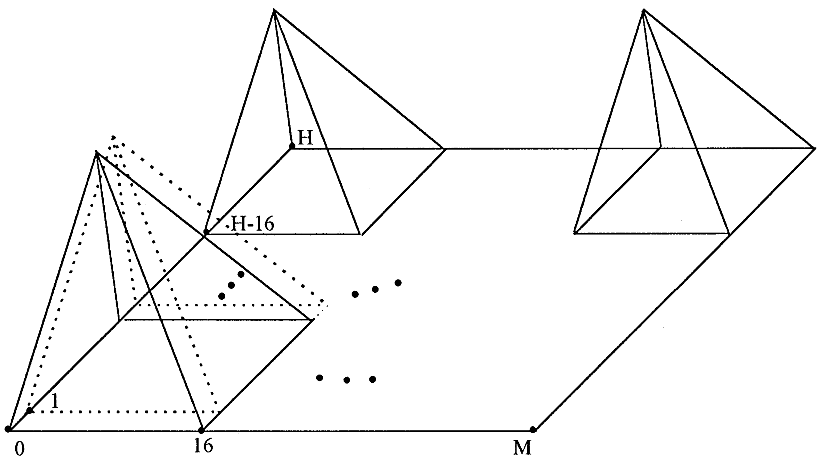

Fig. 2. Some of the $(M-15)\times(H-15)$ pyramids in the previous frame $t-1$. Assume that the image size is $M\times H$, and the block size (the size of the pyramid base) is 16×16.

$$= \sum_{i=1}^{2^k}\sum_{j=1}^{2^k} |f_k^t(i,j) - f_k^{[t-1;(x,y)]}(i,j)|$$
$$= \mathrm{AAD}_k(x,y).$$

The derivation of equation (A2) from equation (A1) is just a change of the indices so that the $2^{k+1}\times 2^{k+1}$ pixels in equation (A1) are decomposed into $2^k\times 2^k$ groups with each group having four adjacent pixels. The derivation of equation (A3) from equation (A2) is again by the Minkowski's inequality (in the four-dimensional vector space). □

APPENDIX B

If each image frame is of size $M\times N$, and if each block **G** is of size 16×16, then the overhead of constructing all the needed pyramids in frame $t-1$ is about $12MH$ operations (more definitely, about $12MH$ additions).

Proof. We discuss here the overhead to construct "all" pyramids in the previous frame $t-1$ (each pyramid will be used by some blocks **G** of the current frame t). As shown in Fig. 2, if the base of each pyramid contains 16×16 pixels, then, to construct the $(M-15)\times(H-15)$ pyramids (some pyramids might have partially overlapping bases, see Fig. 2) for an $M\times H$ image frame (frame $t-1$), the computations of the (generalized) pixel values of the four layers (L_0–L_3) are required. (The pixel values of each "base" layer L_4 has no need to compute, because each base layer coincides with a portion of the given image frame $t-1$, and each pixel value is thus known.) Note that in each layer $\mathbf{L}_{k-1}$ (where $1\le k\le 4$), each pixel value $f_{k-1}(i,j)$ (where $1\le i\le 2^{k-1}$ and $1\le j\le 2^{k-1}$) is obtained by a summation of the 2×2 corresponding pixel values of $\mathbf{L}_k$, and the summation of the 2×2 pixel values needs only three additions [see equation (3)]. Therefore, the total computation to obtain these (generalized) pixel values for the $M\times H$ image frame is $3\times$(total number of generalized pixels).

If we consider all generalized pixels in layer 3, then, due to the overlap of pyramid bases, there are $(M-1)\times(H-1)$ generalized pixels (all pyramids together) in layer L_3 for frame $t-1$. Similarly, there are $(M-3)\times(H-3)$, $(M-7)\times(H-7)$, $(M-15)\times(H-15)$ generalized pixels (all pyramids together) in L_2, L_1, L_0, respectively. The total number of generalized pixels is therefore $(M-1)(H-1)+(M-3)(H-3)+(M-7)(H-7)+(M-15)(H-15)$, and the total number of operations to get their (generalized) pixel values are thus

$$\begin{aligned} U &= 3[(M-1)(H-1)+(M-3)(H-3)\\ &\quad +(M-7)(H-7)+(M-15)(H-15)]\\ &= 12MH-78M-78H+852\\ &\approx 12MH. \end{aligned}$$ □

REFERENCES

1. L. W. Lee, J. F. Wang, J. Y. Lee and J. D. Shie, Dynamic search-window adjustment and interlaced search for block-matching algorithm, *IEEE Trans. Circuits System Video Technol.* **3**(1), 85–87 (1993).
2. T. Koga, K. Iinuma, A. Hirano, Y. Iijima and T. Ishiguro, Motion compensated interframe coding for video conf. *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, pp. G5.3.1–G.5.3.5 (1981).
3. A. Puri, H. M. Hang and D. L. Schilling, An efficient block-matching algorithm for motion compensated coding, *Proc. IEEE ICASSP*, pp. 25.4.1–25.4.4 (1987).
4. B. Liu and A. Zaccarin, New fast algorithms for the estimation of block motion vectors, *IEEE Trans. Circuits System Video Technol.* **3**(2), 148–157 (1993).
5. W. Li and E. Salari, Successive elimination algorithm for motion estimation, *IEEE Trans. Image Process.* **4**(1), 105–107 (1995).
6. R. L. Wheeden and A. Zygmund, *Measure and Integral: An Introduction to Real Analysis*, Vol. 131. Marcel Dekker, New York (1977).
7. ITU-T Recommendation H.263 software implementation, Digital Video Coding Group at Telenor R&D (1995).

About the Author–JING-YI LU was born in 1971 in Taiwan, Republic of China. She received her B.S. degree in 1994 and M.S. degree in 1996, both from the Computer and Information Science Department of National Chiao Tung University, Taiwan. Her recent research interests include image processing and signal processing.

About the Author–KUANG-SHYR WU was born in 1963 in Taiwan, Republic of China. He received his B. S. degree in 1988 from National Chung Cheng Institute of Technology, Taiwan. Since 1994 he has been studying toward the Ph.D. degree and he is currently a Ph.D. candidate in the Computer and Information Science Department of Chiao Tung University. His recent research interests include image processing and document analysis.

About the Author—JA-CHEN LIN was born in 1955 in Taiwan, Republic of China. He received his B.S. degree in computer science in 1977 and M.S. degree in applied mathematics in 1979, both from National Chiao Tung University, Taiwan. In 1988 he received his Ph.D. degree in mathematics from Purdue University, U.S.A. In 1981–1982, he was an instructor at National Chiao Tung University. From 1984 to 1988, he was a graduate instructor at Purdue University. He joined the Department of Computer and Information Science at National Chiao Tung University in August 1988, and is currently an Associate Professor there. His recent research interests include pattern recognition, image processing, and parallel computing. Dr Lin is a member of the Phi-Tau-Phi Scholastic Honor Society, the Image Processing and Pattern Recognition Society, and the IEEE Computer Society.