IEICE TRANS. FUNDAMENTALS, VOL. E80-A, NO. 8 AUGUST 1997

PAPER Special Section on Digital Signal Processing

Bits Truncation Adaptive Pyramid Algorithm for Motion Estimation of MPEG2

Li JIANG†, *Nonmember*, Kazuhito ITO††, *and* Hiroaki KUNIEDA†, *Members*

SUMMARY In this paper, a new bits truncation adaptive pyramid (BTAP) algorithm for motion estimation is presented. The method employs bits truncation of the gray level from 8 bits to much less bits in the searching algorithm. Compared with conventional fast block matching algorithms, this method drastically improves speed for motion estimation on reduced gray-level images and preserves reasonable performance and algorithm reliability. Bits truncation concept is well combined with hierarchical pyramid algorithm in order to truncate adaptively according to image characteristics. The computation complexity is much less than that of pyramid algorithm and 3-Step motion estimation algorithm because of bit-truncated search and low overhead adaptation. Nevertheless, the PSNR property is also comparable with these two algorithms for various video sequences.

key words: *bits truncation, adaptive, pyramid, motion estimation, MPEG2*

1. Introduction

Motion estimation is the most time consuming task in encoding video of today's hybrid video coding standards such as MPEG2. In real time video coding applications, such as HDTV, the reduction of high computation complexity of motion estimation algorithm while preserving reasonable performance is always of great concern. Many fast algorithms as well as some real time hardware designs have been proposed [1], [2], [4], [6]. However, in order to be a general purpose standard such as MPEG2, its High Level standards (HP@HL and MP@HL) are being frustrated by the drastic computation complexity incurred by motion estimation. This is because the video is of image size 1920 × 1152 with frame rate 60 frame/s. The required number of operations per second is more than 5.7 BOP if a search range of 60 in both direction is required (as in HDTV).

Recently, many hardware implementations for MPEG2 are proposed. But most of them are restricted to the Main Level standard [5]–[7]. High Level standard implementation is still viewed as a difficulty and dedicated design is considered as a cost efficient solution. In [4], a single chip dedicated implementation of motion estimation for MP@HL is presented, which uses 256 PEs, and 12.7 mm × 13.7 mm of area. This example, indicates the difficulties in the design of MPEG2 HL standard. Firstly, the PE size may be very large. 256 PE's array will occupy very large area [4]. Secondly, to increase PE speed, pipeline stages are usually inserted into PE. Thus, the increased new latency will decrease the number of search points and degrade the performance. Hence, it is still quite necessary to develop faster motion estimation algorithm and to reduce the hardware size, to increase the cost-efficiency.

In order to overcome such difficulties, we exploit another level of hierarchy, that is the gray-level hierarchy. We enjoy extreme fast search on images of reduced gray levels. By bits truncation, the hardware is correspondently reduced and critical signal path is greatly shorten. To avoid uniform bits truncation scheme, we combined bits truncation with pyramid algorithm to adaptively adjust truncation threshold value. It is derived from information of data distribution between pyramid levels. The algorithm has the advantages such as high speed, and excellent performance and high reliability.

2. Basic Idea

Here, we shall describe a new motion estimation algorithm which aims at further reduction of the computation complexity of motion estimation at less expense of video quality. Our method is to perform motion estimation for bits truncated image while preserving algorithm reliability as well as image quality. We combine it with pyramid algorithm [1] adaptively to find a better bits truncation for every macroblock matching.

The widely used mean absolute difference (MAD) criterion is written as

$$S(m,n) = \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}(m,n) \tag{1}$$

$$d_{ij}(m,n) = |\, f_k(i,j) - f_{k-1}(i+m, j+n)\,| \tag{2}$$

where N is the macroblock size in one dimension. (m,n) is the distance in coordinates between a reference block and candidate blocks. f_k is the pixel value of current frame. f_{k-1} is the pixel value of previous frame. $S(m,n)$ is calculated for every candidate macroblock within the search window. The (m,n) with minimum $S(m,n)$ is chosen as a motion vector. Here,

Manuscript received December 11, 1996.
Manuscript revised March 6, 1997.

†The authors are with the Department of Electrical and Electronic Engineering, Tokyo Institute of Technology, Tokyo, 152 Japan.
††The author is with the Department of EE System Engineering, Saitama University, Urawa-shi, 338 Japan.

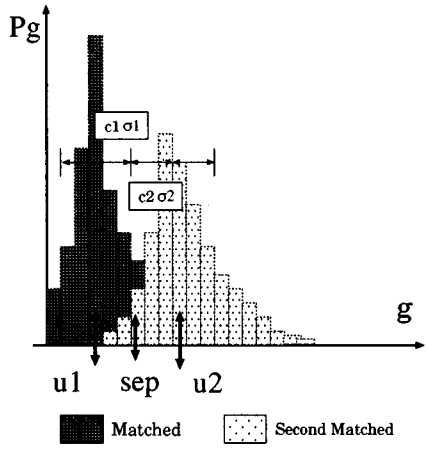


Fig. 1 Distribution of $d_{ij}(m,n)$.

$d_{ij}(m,n)$s are represented by the same number of bits as the original image pixels. For 8-bit image case, it is also 8 bit.

Figure 1 shows a typical example of the distribution of $d_{ij}(m,n)$ of the best and second best matched macroblock. The vertical axis is the probability while the horizontal axis is the gray-level. For the best matched block, the grey levels to represent $d_{ij}(m,n)$ is usually much less than 256. This is intuitively understood and has been demonstrated by computer simulations. If $d_{ij}(m,n)$ is represented by using less than 8 bits, then the summation of Eq. (2) becomes easier and hence the motion estimation speeds up.

The *bits truncation* is the procedure to reduce the number of bits to represent $d_{ij}(m,n)$ values. The simplest bits truncation is just to ignore less significant bits of $d_{ij}(m,n)$. However, such bits truncation may lose important characteristics of the original $d_{ij}(m,n)$. For example, $d_{ij}(m,n)$ can be bit-truncated to 1 if $d_{ij}(m,n) \geqq 128$, or 0 if $d_{ij}(m,n) < 128$. In other words, $d_{ij}(m,n)$ is bit-truncated to only one bit by using a threshold value of 128. In case when all the values of $d_{ij}(m,n)$ are less than 128, there is no way to identify the motion vector (m,n) with the minimum $S(m,n)$ since all $S(m,n)$ become zero. Therefore, an adaptive algorithm is used to choose appropriate threshold values which preserve the characteristics of the $d_{ij}(m,n)$ necessary to search the motion vector (m,n). However, the calculation of appropriate threshold values directly requires the similar amount of computations to conventional one. In order to reduce this computation overhead. Our algorithm combines bits truncation with pyramid algorithm so as to get information of low level threshold value from high pyramid level search. We adjust the threshold value adaptively for every macroblock.

3. Bits Truncation Threshold

Now we consider the above idea from the viewpoint of data distribution. $d_{ij}(m,n)$ is the absolute difference value between pixels of candidate blocks and reference block. Let the distribution function of $d_{ij}(m,n)$ of each matching within search window be denoted as $p_g(m,n)$, assuming $\sum_{g=0}^{255} p_g(m,n) = N^2$. Let gray-level be denoted as g. Then, the mean absolute difference (MAD) can be rewritten by using $p_g(m,n)$ as

$$S(m,n) = \sum_{g=0}^{255} g \cdot p_g(m,n). \tag{3}$$

For each matching in the search window, $d_{ij}(m,n)$ distributes with mean and variance, denoted as $u(m,n)$ and $\sigma(m,n)$. They are defined by $u(m,n) = S(m,n)/N^2$ and $\sigma(m,n) = E((d_{ij}(m,n) - u(m,n))^2)$ with ensemble operation. According to our investigation, the mean and variance play an important role to derive the efficient separation threshold. Let u_1, u_2 and σ_1, σ_2 be the mean and variance of $d_{ij}(m,n)$ for matched block and second matched block respectively. Without loss of generality, $u(m,n)$ of different candidate blocks have a relation in their magnitude as

$$u_1 < u_2 < u_3 \ldots . \tag{4}$$

Figure 1 shows a typical distribution of the first and second matched block data and their separation. The motion estimation is to find the vector of the minimum mean, u_1 within search window. In other words, it is to separate u_1 from all other u, or just from u_2.

Equation (3) can also be expressed by four parts or more parts according to the range of gray-level. In 2-bit approximation case, we rewrite it into four parts as

$$S(m,n) = \sum_{g=0}^{sep_1-1} g \cdot p_g(m,n) + \sum_{g=sep_1}^{sep_2-1} g \cdot p_g(m,n) + \sum_{g=sep_2}^{sep_3-1} g \cdot p_g(m,n) + \sum_{g=sep_3}^{255} g \cdot p_g(m,n)$$

where $sep_k \in [0,255]$, $(k = 1,2,3)$, and $sep_1 < sep_2 < sep_3$.

One of truncations of $d_{ij}(m,n)$ to 4 kinds of gray-level is to approximate values within $0 \leqq d_{ij}(m,n) < sep_1$ as 0, within $sep_1 \leqq d_{ij}(m,n) < sep_2$ as 1, within $sep_2 \leqq d_{ij}(m,n) < sep_3$ as 2, and within $sep_3 \leqq d_{ij}(m,n) < 256$ as 3. Magnitudes 0, 1, 2, 3 of the approximated values are not important because our main purpose is to compare MAD among all candidate blocks. In this case, $S(m,n)$ can be approximated to $S_2'(m,n)$ as

$$S_2'(m,n) = \sum_{k=1}^{3} k \times \sum_{g=sep_k}^{sep_{k+1}-1} p_g(m,n) \tag{5}$$

where $sep_4 = 256$.

In the same way, we use $S_1'(m,n)$ for 1-bit approximation as

$$S_1'(m,n) = \sum_{g=sep}^{255} p_g(m,n). \tag{6}$$

Thus, $d_{ij}(m,n)$ smaller than the threshold value sep is approximated to 0, and larger $d_{ij}(m,n)$ become 1. The matched block will have more 0 while the unmatched block will have more 1 in approximated values. Then, the block with the smallest S' will become the matched block. Instead of $S(m,n)$, $S_2'(m,n)$ and $S_1'(m,n)$ can be used as criterion to find motion vector.

In the above, we have pointed out that the choice of an appropriate threshold value is critical for an effective approximation of $d_{ij}(m,n)$ and separating u by S'. Intuitively, as shown in Fig. 1, we select the threshold value sep in between u_1 and u_2 as

$$sep = (u_1 + u_2)/2. \tag{7}$$

Let $p_g(m_1,n_1)$ and $p_g(m_2,n_2)$ denote the distribution of $d_{ij}(m,n)$ for matched block and second matched block respectively. Under the condition of

$$u_2 - u_1 > 2(\sigma_1^2 + \sigma_2^2)^{1/2}, \tag{8}$$

we can derive an inequality

$$\sum_{g=sep}^{255} p_g(m_2,n_2) > \sum_{g=sep}^{255} p_g(m_1,n_1). \tag{9}$$

This inequality means that the matched block can be separated from the unmatched blocks by setting sep as Eq. (7).

proof: If Z is random variable, with its mean as u_z and variance as σ, Chebyshev's Inequality

$$P(|Z - u_z| \geqq c\sigma) \leqq 1/c^2 \tag{10}$$

holds. c can be any number. $|Z - u_z|$ is the absolute distance between Z and u_z. The inequality describes that the possibility of this distance larger than $c\sigma$ is less than $1/c^2$.

Considering $d_{ij}(m,n)$ as random variable, we derive

$$1/c_1^2 \geqq P(|d_{ij}(m_1,n_1) - u_1| \geqq c_1\sigma_1) \tag{11}$$

$$1 - P(|d_{ij}(m_2,n_2) - u_2| \geqq c_2\sigma_2) \geqq 1 - 1/c_2^2. \tag{12}$$

As shown in Fig. 1, when $c_1\sigma_1 = sep - u_1$ and $c_2\sigma_2 = u_2 - sep$, we obtain

$$P(|d_{ij}(m_1,n_1) - u_1| \geqq c_1\sigma_1) \geqq \frac{1}{N^2} \sum_{g=sep}^{255} p_g(m_1,n_1) \tag{13}$$

$$1 - P(|d_{ij}(m_2,n_2) - u_2| \geqq c_2\sigma_2) \leqq \frac{1}{N^2} \sum_{g=sep}^{255} p_g(m_2,n_2). \tag{14}$$

Under condition Eq. (8) and by selecting sep to be in the middle of u_1 and u_2 as in Eq. (7), we obtain

$$1 > \frac{1}{c_1^2} + \frac{1}{c_2^2}. \tag{15}$$

From above equations, we can finally get

$$\sum_{g=sep}^{255} p_g(m_2,n_2) > \sum_{g=sep}^{255} p_g(m_1,n_1) \tag{16}$$

or $S_1'(m_2,n_2) > S_1'(m_1,n_1)$.

Therefore, if we set the threshold value in the middle of u_1 and u_2 (not necessary condition), we can separate them.

The condition Eq. (8) implies that the larger $u_2 - u_1$ become, the better the separation is achieved. According to practical simulation results, most images satisfy the inequality for the mean and variance. The variance of $d_{ij}(m,n)$ becomes small for matched cases. In case the image is not so plain, the $u_2 - u_1$ is much larger than u_1. It is easier to estimate a motion vector. On the contrary, in case the image is plain, $u_2 - u_1$ becomes small. Thus, even condition Eq. (8) is not satisfied, the estimation error will be also small.

4. Adaptive Threshold

We perform bits truncation to use less gray levels to approximate $d_{ij}(m,n)$. However, the selection of sep requires the probability density function of $d_{ij}(m,n)$ for each matching. It can only be directly calculated after motion estimation when we know u_1 and u_2. This contradiction will make bits truncation no meaning at all. Further more, every macroblock has different characteristics. We should adaptively choose sep according to statistic information of every macroblock.

Thus, our adaptive method combines bits truncation with pyramid search algorithm.

1. In the highest pyramid level, we perform search in the same way as original algorithm using 8-bit criterion. Since block size is quite small compared with low level, the computation complexity becomes also low. We estimate the position for u_1 and u_2, and the sep for low level according to the statistic information.
2. In the low level, we use derived sep to truncate $d_{ij}(m,n)$ and perform search on these bit-truncated $d_{ij}(m,n)$.

We discuss herein how to decide adaptive threshold sep in the mean pyramid algorithm and in simple down-sampling algorithm.

4.1 Threshold for Mean Pyramid

The mean pyramidal images [1] are constructed by simply averaging four neighboring pixels of the lower level to construct a pixel data in the higher level as

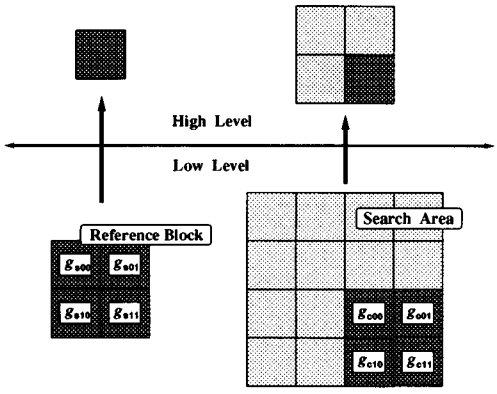


Fig. 2 High level and low level of pyramid algorithm.

$$g_{L+1}(p,q) = \lceil 1/4 \sum_{u=0}^{1} \sum_{v=0}^{1} g_L(2p+u, 2q+v) \rfloor, \quad (17)$$

where $g_{L+1}(p,q)$ represents the gray level at the position (p,q) of the $(L+1)$th level and $g_1(p,q)$ denotes the original image. The number of pixels in the next upper level is reduced one fourth as small as the lower level. The means are denoted as u_H and u_L for higher and lower level respectively. As shown in Fig. 2, g_s and g_c denote the pixel values of the reference block and candidate blocks respectively.

Now we want to find relation between u_H and u_L.

One pixel's absolute difference at high level is expressed by

$$d_H = \frac{1}{4}\big|(g_{s00} - g_{c00}) + (g_{s01} - g_{c01}) + (g_{s10} - g_{c10}) + (g_{s11} - g_{c11})\big|, \quad (18)$$

which corresponds to four pixels difference in low level. The corresponding sum of absolute difference is described by

$$d_L = \frac{1}{4}\big(|g_{s00} - g_{c00}| + |g_{s01} - g_{c01}| + |g_{s10} - g_{c10}| + |g_{s11} - g_{c11}|\big). \quad (19)$$

Triangular inequality of $|a+b| \leqq |a| + |b|$ leads to the relation, $d_H \leqq d_L$. Since these inequalities hold for every pixel in the high level, $u_H \leqq u_L$ holds for their average.

Now, we compare $4d_H^2$ with d_L^2 as,

$$4d_H^2 = \frac{1}{4}\big((g_{s00} - g_{c00})^2 + (g_{s01} - g_{c01})^2 + (g_{s10} - g_{c10})^2 + (g_{s11} - g_{c11})^2 + \boldsymbol{I}_B\big)$$

$$d_L^2 = \frac{1}{16}\big((g_{s00} - g_{c00})^2 + (g_{s01} - g_{c01})^2 + (g_{s10} - g_{c10})^2 + (g_{s11} - g_{c11})^2 + \boldsymbol{I}_C\big)$$

$$\boldsymbol{I}_B = 2 \sum_{ij \neq mn} (g_{sij} g_{smn} + g_{cij} g_{cmn} - g_{smn} g_{cij} - g_{sij} g_{cmn}) \quad (20)$$

Table 1 Simulation results for u_H and u_L relation.

Mean Pyramid	level3 → level2	level2 → level1
$u_{H1} \leqq u_{L1} \leqq 2u_{H1}$	95.7%	99.8%
$u_{H2} \leqq u_{L2} \leqq 2u_{H2}$	99.4%	99.9%

Simple Down-sampling	level3 → level2	level2 → level1
$0.7u_{H1} \leqq u_{L1} \leqq 1.3u_{H1}$	66.8%	73.2%
$0.7u_{H2} \leqq u_{L2} \leqq 1.3u_{H2}$	56.8%	58.7%
$0.5u_{H1} \leqq u_{L1} \leqq 1.5u_{H1}$	84.1%	89.9%
$0.5u_{H2} \leqq u_{L2} \leqq 1.5u_{H2}$	82.3%	84.9%

$$\boldsymbol{I}_C = 2 \sum_{ij \neq mn} (|g_{sij} - g_{cij}| * |g_{smn} - g_{cmn}|) \quad (21)$$

where $ij, mn \in (00, 01, 10, 11)$.

From inequality $a^2 + b^2 \geqq 2ab$, we obtain

$$(g_{sij} - g_{cij})^2 + (g_{smn} - g_{cmn})^2 \geqq 2(|g_{sij} - g_{cij}| * |g_{smn} - g_{cmn}|) \quad (22)$$

Thus, only if $\boldsymbol{I}_B \geqq 0$, we can get $4d_H^2 \geqq d_L^2$, or $2d_H \geqq d_L$.

In order to show that $\boldsymbol{I}_B \geqq 0$ holds statistically, we investigate $E(\boldsymbol{I}_B)$. As defined in [8], the correlation of two pixels within one frame is called spatial correlation. On the contrary, the correlation of two pixels of different frames at the same position is called temporal correlation. $E(\boldsymbol{I}_B)$ includes $E(g_{sij}g_{smn})$ and $E(g_{cij}g_{cmn})$, $E(g_{sij}g_{cmn})$ and $E(g_{cij}g_{smn})$. Since pixel pairs (g_{sij}, g_{smn}) and (g_{cij}, g_{cmn}) belong to one frame, $E(g_{sij}g_{smn})$ and $E(g_{cij}g_{cmn})$ are spatial correlation. However, pixel pairs (g_{smn}, g_{cij}) and (g_{sij}, g_{cmn}) are not only in different frames but also at different pixels position. Thus, $E(g_{sij}g_{cmn})$ and $E(g_{cij}g_{smn})$ have both certain spatial distance as well as certain temporal distance. This indicates that the correlation of $E(g_{sij}g_{cmn})$ and $E(g_{cij}g_{smn})$ are smaller than $E(g_{sij}g_{smn})$, $E(g_{cij}g_{cmn})$. As a result,

$$E(\boldsymbol{I}_B) \geqq 0 \quad (23)$$

holds statistically. Therefore, we can show that statistically $2d_H \geqq d_L$ and thus $2u_H \geqq u_L$ holds.

From the above two points, we derive the inequality as

$$u_H \leqq u_L \leqq 2u_H. \quad (24)$$

In order to confirm Eq. (24), we perform the computer simulation for Flower Garden video sequences. In Table 1, it is shown how many blocks among the total of 1350 blocks of one frame (720 × 480) satisfies the relation $u_H \leqq u_L \leqq 2u_H$ for each pyramid hierarchy level $i \rightarrow$ level $(i-1)$. u_{H1} and u_{L1} are corresponding to matched block, while u_{H2} and u_{L2} are corresponding to the second matched block. It is clear that more than 95-percent of blocks within one frame satisfy the above relation. Also, we simulated Mobile & Calendar video sequences and find 93-percent of blocks within one frame satisfy this relation.

From the statistic information we derive from high pyramid level, we select a proper threshold value *sep* for the low level as follows.

The relation (24) also holds for the first and second matched blocks cases as

$$u_{H1} \leqq u_{L1} \leqq 2u_{H1} \tag{25}$$

$$u_{H2} \leqq u_{L2} \leqq 2u_{H2}. \tag{26}$$

The minimum mean difference of high level u_{H1} may move in the low level to u_{L1}. However its moving range is limited from u_{H1} to $2u_{H1}$. The same things applies to u_{L2} and u_{H2} of the second matched block. For 1-bit approximation, we select $sep = (u_{L1} + u_{L2})/2$ for bits truncation in the lower level. Since

$$(u_{H1} + u_{H2})/2 \leqq sep \leqq (u_{H1} + u_{H2}) \tag{27}$$

holds, we can set *sep* in the middle of this range as

$$sep = 0.75(u_{H1} + u_{H2}). \tag{28}$$

It tells that the lower level bits truncation threshold is decided by the average calculated in higher level analysis. In 2-bit approximation case, we select $sep_2 = 0.75(u_{H1} + u_{H2})$, and $sep_1 = 0.5sep_2$, $sep_3 = 1.5sep_2$.

4.2 Threshold for Simple Down-Sampling

The simple down-sampling algorithm is just to select one pixels from every four pixels of low level to construct high level. Since no low pass filtering is performed, it is sometimes suffered from noise effects. But, due to its simplicity, it is also widely used.

When the image's frequency is not so high and the aliasing is small, d_H and d_L can be estimated to be almost the same.

$$u_L \cong u_H. \tag{29}$$

This relation is inherited by average of the best matched and the second best matched block as

$$u_{L1} \cong u_{H1}\ u_{L2} \cong u_{H2}. \tag{30}$$

So, *sep* can be selected as $\frac{u_{H1}+u_{H2}}{2}$ in 1-bit approximation case. In 2-bit case, we select $sep_2 = 0.5(u_{H1}+u_{H2})$, and $sep_1 = 0.5sep_2$, $sep_3 = 1.5sep_2$.

4.3 Threshold on 2's Power

The above selections have given satisfactory results for simulated video sequences. However, for further reduction of the computation complexity, we can select *sep* on 2's power as 2^n. This is shown in Fig. 3. Figure 3 (a) shows the 1-bit approximation case. The example is for $sep = 4$. When $d_{ij} < 4$, d_{ij} is converted to 0, otherwise 1. Figure 3 (b) shows the 2-bit case with $sep1 = 2$, $sep2 = 4$ and $sep3 = 6$. When $d_{ij} < 2$, it is converted to 00, when $2 \leqq d_{ij} < 4$, it is converted to 01, and so on. In fact, no comparison is needed any more for this threshold value. The computation complexity is greatly reduced and the algorithm becomes a real bits truncation algorithm.

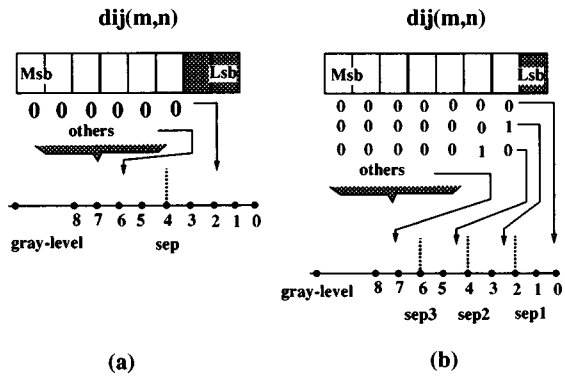


Fig. 3 Select *sep* from 2's power.

In this way, we have chosen threshold only on 0, 2, 4, 8, 16, 32, 64, 128. The mapping of original arbitrary value *sep* to 2's power should make original *sep* around 2^n symmetrically. For example, when original *sep* equals to 3 or 4 or 5, we map them all to 4. When original *sep* equals to 6, 7, 8, 9 or 10, it is mapped to 8, and so on. Thus, for small *sep*, the mapping error is small. For large *sep*, mapping error becomes large but the the distribution of $d_{ij}(m,n)$ is also sparse.

5. Bits Truncation Adaptive Hierarchical Search

Now we summarize the 3-level bit-truncated adaptive algorithm as a whole with practical figures. The step is:

1. Construct the 3-level pyramidal search window and macroblocks
2. At the highest pyramid level, the block size is 4 pixels by 4 pixels. Search motion vector with 8-bit MAD criterion. Obtain u_{H1} and u_{H2}. Set sep_k according to the adaptive method.
3. At the middle level, the block size is 8×8. Search motion vector using bit-truncated $S'(m,n)$ criterion. Get corresponding motion vectors and sep_k for lower level.
4. At the low level, the block size is 16×16. Search motion vector using bit-truncated $S'(m,n)$ criterion.
5. Calculate the motion vector as $mv = \sum_{n=1}^{l} mv_n * 2^{n-1}$, where l is number of levels (l=3).

The ratio of the pixel number is 16 : 64 : 256 from highest to lowest level of macroblock size 16×16. For the highest level, we perform just the same search as

original algorithm. This is quite necessary to keep image quality because the top level determines the most important motion vector information. Computation complexity is also quite low here compared with the other two levels.

6. Performance Simulation Results

The proposed algorithm is applied to four video sequences of 50 frames each to demonstrate the performance properties of our algorithm. They are Flower Garden, Mobile & Calendar, Claire of size 720 × 480 and Table Tennis of size 352 × 240. Satisfactory results have been obtained. The performance is evaluated using $PSNR$ defined as

$$PSNR = 10\log\frac{ImageSize * 255^2}{\sum pixel\_error^2}$$

Table 2 shows the simulated results. For example, $2\,bitMean2^n$ means the proposed algorithm of 2-bit approximation with mean pyramid algorithm and *sep* is set to 2's power.

proposed 1 and proposed 2 are results of bits truncation with mean pyramid algorithm. We can compare them with *Mean Pyramid* item in Table 2. Though the four video sequences are quite different in property, our algorithm are always catching up the $PSNR$ performance with mean pyramid algorithm. Even when 1-bit approximation and *sep* sets to 2's power, the performances are still satisfactory. Also, if we comparing with 3-step algorithm, we find its performance is similar to us. The performance difference of our algorithm with full search algorithm is related with the performance difference of pyramid algorithm with full search algorithm.

proposed 3 and proposed 4 are results of bits truncation with simple down-sampling pyramid algorithm. We can compare them with *Simple Pyramid* item.

When frames'spatial frequency is high and object's move is slow, u_L and u_H 's relationship may fail for some blocks and this results in search error. Also, high spatial frequency means high penalty ($pixel\_error^2$) for search error. We find that in the Mobile & Calendar video sequences' simulation, it is 0.43 dB for $2\,bitMean$ algorithm and 0.56 dB for $1\,bitMean2^n$ algorithm worse than *Mean Pyramid* algorithm, which is not as good as Flower Garden's case. But from the reconstructed frames, we find most parts of image's search have preserved the performance of *Mean Pyramid* algorithm, Only in the calendar part, the "4" has vanished. But the overall performance is still satisfactory.

The result of no adaption *NoAdapt* indicates the necessity of the adaptation for keeping the reliability of algorithm.

Table 2 Average PSNR Property.

Conventional	FlowerG	MobileC	TableT	Claire
FullSearch	26.23	24.06	27.92	45.78
MeanPyramid	25.58	23.52	27.49	45.66
3 − Step	25.29	23.21	27.32	45.72
SimplePyramid	24.27	21.01	24.06	39.58
Proposed 1	FlowerG	MobileC	TableT	Claire
$2\,bitMean$	25.30	23.09	27.29	45.33
$1\,bitMean$	25.23	23.01	27.22	45.14
Proposed 2				
$2\,bitMean2^n$	25.19	23.01	27.20	45.21
$1\,bitMean2^n$	25.11	22.96	27.08	45.04
Proposed 3				
$2\,bitSim$	23.91	20.65	23.78	39.27
$1\,bitSim$	23.84	20.52	23.75	38.91
Proposed 4				
$2\,bitSim2^n$	23.82	20.58	23.76	39.02
$1\,bitSim2^n$	23.80	20.49	23.56	38.80
NoAdapt	22.34	17.59	21.43	34.45

7. Computation Complexity

Since our algorithm is based on bits truncation, it is convenient for us to compare the computation complexity based on bit-serial operation. We compare how many clock cycle needed for all operations. We assume that the computation complexity of summation is linear to the number of bits. For example, the computation complexity of 16-bit addition is twice as large as that of 8-bit addition and so on.

For the low level, the macroblock size is 16 × 16. Thus, to calculate the 256 $d'_{ij}(m,n)$ of n-bit, we perform 128 n-bit additions, then 64 (n+1)-bit additions, and so on. As a result, the total number of additions becomes

128	×	addition	(n)
64	×	addition	(n+1)
32	×	addition	(n+2)
16	×	addition	(n+3)
8	×	addition	(n+4)
4	×	addition	(n+5)
2	×	addition	(n+6)
1	×	addition	(n+7)
		Sum	(n+8)

The number of operations is scaled to 8-bit add/sub operation. In our algorithm of $1\,bitMean2^n$ and $2\,bitMean2^n$, with *sep* setting on 2's power, no comparison operation is needed, and we assume only 1/8 of operation is added to 8 bit subtraction of $d_{ij}(m,n) = |f_k(i,j) - f_{k-1}(i+m, j+n)|$. For original algorithm, the number of operation $N_l(b8)$ in the lowest level of 256 pixels becomes

$$\begin{aligned} N_l(b8) &= 256 + \sum_{k=7}^{0} 2^k(n+7-k)/8 \\ &= 256 + (255n + 245)/8. \end{aligned} \tag{31}$$

In the middle level, the pixels is 64, the number of operation $N_m(b8)$ is

$$N_m(b8) = 64 + \sum_{k=5}^{0} 2^k(n+5-k)/8 = 64 + (63n+57)/8. \quad (32)$$

In the high level, the pixels is 16, the number of operation $N_h(b8)$ becomes

$$N_h(b8) = 16 + \sum_{k=3}^{0} 2^k(n+3-k)/8 = 16 + (15n+11)/8. \quad (33)$$

The total number of operation $N(b8)$ for all three levels can be expressed as

$$N(b8) = 336 + (333n+313)/8. \quad (34)$$

Since $n = 8$, we have the number of operation is 542, 135, 33 from low to high level respectively, and the total of operation is 709.

For our algorithm, the number of operation $N_l(bt)$ is calculated in low level as

$$N_l(bt) = 256(1+1/8) + \sum_{k=7}^{0} 2^k(n+7-k)/8 = 288 + (255n+245)/8. \quad (35)$$

In, the middle level as

$$N_m(bt) = 64(1+1/8) + \sum_{k=5}^{0} 2^k(n+5-k)/8 = 72 + (63n+57)/8. \quad (36)$$

And in the high level the same as original algorithm. Totally, the number of operation $N(bt)$ becomes

$$N(bt) = 360 + (318n+302)/8 + 33. \quad (37)$$

The results of the number of operations are almost the same in the function for n bit. Our algorithm uses $n = 1$ or 2, while original algorithm uses $n = 8$. For 2 bits approximation, $n = 2$, and the number of operation is 383, 95 and 33 respectively from low to high level. The total is 499. For 1 bit approximation, $n = 1$, and the number of operation is 351, 87 and 33 respectively from low to high level. The total number of operation becomes 471.

The calculation of *sep* has not been taken into consideration in the comparison, since it is calculated only once for the whole search area, this overhead is very low (0.1-percent) and can be neglected.

In Table 3, we compared the number of operations required for algorithms. It is assumed that the maximum motion vector is 7 in both direction and thus 9 points is searched for each step of 3-Step algorithm or each pyramid level. *Prep.* means the preparation of higher pyramid levels for searching. The number of operation is three times less than 3-Step algorithm, and because the neighboring candidate block data within search area can be reused during search [5], it may require less I/O access than 3-step algorithm.

Table 3 Number of operations.

Conventional	Prep.	First	Second	Third	Total
FS					121950
3 – Step		9×542	9×542	9×542	13550
Mean3level	240	9×33	9×135	9×542	6630

Proposed	Prep.	First	Second	Third	Total
1 *bitMean*2^n	240	9×33	9×87	9×352	4488
2 *bitMean*2^n	240	9×33	9×95	9×383	4839

The computation complexity is two third of Mean pyramid algorithm.

In fact, it is difficult to compare them only by number of operation.

As we have analyzed, the high level standards of MPEG-2 requires the hardware running at very high speed. The hardware's critical signal path should be quite short for such a purpose. In the above comparison, we have assumed that the computation complexity of addition to be linear with the number of bits. This is only true for CPA (carry-propagate-adder). When PE working at very high speed, CLA (carry-look ahead-adder) may be required, or pipeline stage may be inserted. In all cases, the computation complexity is not linear with number of bits. From our practical investigation, we find that when the clock frequency is high (for example, 200 MHz), the area to realize a 16-bit adder may require 3 times the area as a 8-bit adder.

More bits of operation usually means more bits of saving. The intermediate saving are not taken into consideration in the comparison yet. For the original algorithm, the accumulator may require 16-bit register. While in our case, it is much less. Since the registers are prepared in every PE for array implementation, we save these registers greatly. The hardware cost is once more reduced in our case.

Furthermore, by taking advantage of the feature of motion estimation, we can change the operations of absolute difference (Eq. (2)) and comparison with *sep* into two comparison

$$| f_k - f_{k-1} | \leqq sep \Rightarrow f_k - sep \leqq f_{k-1} \leqq f_k + sep$$

Note that $f_k \pm sep$ are operation with reference block and need only to be calculated once for whole search area. A PE thus contains only two comparators. One comparator can be designed with 22 gates of 10 gate delay, which results in critical signal path of only 10 gate delay. Comparing with conventional designs [4], the critical single path is usually 40 gate delay when CPA is used. We estimate our PE size will be 5 times as small as or smaller than a conventional one.

8. Conclusion

In this paper, a new bits truncation adaptive algorithm for motion estimation is proposed. This method improves speed of motion estimation by reducing the computation complexity for searching motion vectors on reduced gray-level images, while it preserves reasonable performance and algorithm reliability. Because of bit-truncated search and low cost adaptation, the computation complexity is greatly reduced. The PSNR property is also comparable with the other algorithms for the tested video sequences.

References

[1] K.M. Nam, J.-S. Kim, R.-H. Park, and Y.S. Shim, "A fast hierarchical motion vector estimation algorithm using mean pyramid," IEEE Trans. CAS for Video Tech., vol.5, pp.344–351, Aug. 1995.

[2] J.R. Jain and A.K. Jain, "Displacement estimation and its application in interframe image coding," IEEE Trans. Commun., vol.COM-29, pp.1799–1808, Dec. 1981.

[3] T. Koga, et al., "Motion compensated interframe coding for video conferencing," Proc. Nat. Telecommun. Conf., pp.G5.3.1-5.3.3, Nov. 1981.

[4] T. Onoye, G. Fujita, M. Takatsu, I. Shirakawa, and N. Yamai, "Single chip implementation of motion estimator dedicated to MPEG2 MP@HL," IEICE trans., vol.E79-A, no.8, pp.1210–1216, Aug. 1996.

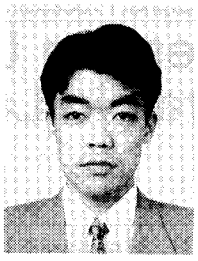
[5] E.G. Tzeng and C.-Y.Lee, "An efficient memory architecture for motion estimation processor design," Proc. IEEE Int'l Symp. Circuits and Systems, pp.712–715, May 1995.

[6] P. Pirsch, N. Demassieux, and W. Gehrke, "VLSI architectures for video compensation—A survey," Proc. IEEE, vol.83, pp.220–246, Feb. 1995.

[7] H.D. Lin, A. Anesko, B. Petryna, and G. Pavlovic, "A programmable motion estimator for a class of hierarchical algorithms," Proc. IEEE Signal Processing Society Workshop on VLSI Signal Processing, Sakai, Japan, pp.411–420, Oct. 1995.

[8] F. Rocca and S. Zanoletti, "Bandwidth reduction via movement compensation on a model of the random video process," IEEE Trans. Commun., vol.COM-20, no.10, pp.960–963, Oct. 1972.

Li Jiang was born in Baoji, China on Dec. 1967. He received the B.E. degree and M.E.degree from Department of Electronic Engineering, Fudan Univ., Shanghai, China, in 1990, 1993 respectively. He is currently a MONBUSHO scholarship student and working toward Ph.D. degree in Tokyo Institute of Technology. His research interests include image processing and DSP design



Kazuhito Ito was born in Niigata, Japan on May 29, 1964. He received the BS, the MS, and the Ph.D. degrees in Electrical Engineering from Tokyo Institute of Technology, Tokyo, Japan, in 1987, 1989, and 1992, respectively. He was with the Tokyo Institute of Technology from 1992 and visited the University of Minnesota from April 1993 to July 1994. He is now an associate professor of the Department of Electrical and Electronic Systems, Saitama University, Saitama, Japan. His research interests include high-level synthesis in digital signal processing, VLSI signal processing, and asynchronous systems. Dr. Ito is a member of IEEE.

Hiroaki Kunieda was born in Yokohama, Japan in 1951. He received B.E., M.E. and Dr. Eng. degrees from Tokyo Institute of Technology in 1973, 1975 and 1978, respectively. He was Research Associate in 1978 and Associater Professor in 1985, at Tokyo Institute of Technology. He is currently Professor at EE Department of Tokyo Institute of Technology. He has been engaged in researches on Distributed Circuits, Switched Capacitor Circuits and IC Circuit Simulation. His current research focuses on VLSI Signal Processing, Parallel Processing, VLSI Microarchitecture Design, VLSI CAD and Parallel Image Processing. Dr. Kunieda is a member of IEEE CAS and SP society and IPSJ.