IEICE Electronics Express, Vol.16, No.1, 1–12

LETTER

An adaptive hash-based search for integer motion estimation in SCC

Youngkyu Choi[a)], **Hyuk-Jae Lee, and Soo-Ik Chae**

Department of Electrical and Computer Engineering, Seoul National University, 1 Gwanak-ro, Gwanak-gu, Seoul 08826, Korea

a) *cloud0@snu.ac.kr*

Abstract: As a new approach to finding integer motion vectors in SCC, hash-based searches have been proposed recently. In the paper we propose an improved search algorithm for integer motion estimation (IME) that employs hash-based search only for 8×8 coding units (CUs) and bottom-up search for larger CUs. Furthermore, it updates the hash function and hash table for each CTU to improve coding efficiency and reduce search complexity. According to simulation results, the proposed algorithm provides similar or better coding performance for screen contents while keeping the complexity at a lower level and constant compared with the reference algorithms.

Keywords: screen contents coding (SCC), integer motion estimation (IME), hash-based search, adaptive hash function

Classification: Integrated circuits

References

[1] Recommendation ITU-T H.265, MPEG H — Part 2: High efficiency video coding, ISO/IEC 23008-2 (2013).

[2] I. K. Kim, *et al.*: "High efficiency video coding (HEVC) test model 13 (HM13) encoder description," Document JCTVC-O1002 of JCT-VC, Geneva, CH (2013).

[3] R. Joshi, *et al.*: "Screen content coding test model 7 (SCM 7)," Document JCTVC-W1014 of JCT-VC, San Diego, USA (2016).

[4] B. Li, *et al.*: "A unified framework of hash-based matching for screen content coding," IEEE Visual Communications and Image Processing Conference (2014) 530 (DOI: 10.1109/VCIP.2014.7051623).

[5] W. Zhu, *et al.*: "Hash-based block matching for screen content coding," IEEE Trans. Multimed. **17** (2015) 935 (DOI: 10.1109/TMM.2015.2428171).

[6] H. Jegou, *et al.*: "Product quantization for nearest neighbor search," IEEE Trans. Pattern Anal. Mach. Intell. **33** (2011) 117 (DOI: 10.1109/TPAMI.2010.57).

[7] T. S. Kim, *et al.*: "Fast integer motion estimation with bottom-up motion vector prediction for an HEVC encoder," IEEE Trans. Circuits Syst. Video Technol. **28** (2018) 3398 (DOI: 10.1109/TCSVT.2017.2759245).

[8] HEVC Reference Software [Online]. Available at: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.9+SCM8.0.

[9] H. Yu, *et al.*: "Common test conditions for screen content coding," Document JCTVC-T1015 of JCT-VC, Geneva, Switzerland (2015).

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019

[10] G. Bjontegaard: "Calculation of average PSNR differences between RD-curves," Document VCEG-M33 of ITU-T, Austin, USA (2001).

1 Introduction

Integer motion estimation (IME) is the most essential coding tool for inter prediction to achieve high compression performance in High Efficiency Video Coding (HEVC) [1]. The test zone search (TZS) algorithm [2] adopted in the HM reference software as the search algorithm for IME adaptively employs basically several diamond search rounds and a raster search, if necessary. Although it is suitable for finding small motions, the TZS algorithm has a limitation in finding large motions and not effective for screen contents with repeating patterns, sharp edges and long scroll motions.

Recently, hash-based searches have been proposed for IMEs on screen content [3, 4, 5], where the best-matching block is searched only among reference blocks whose hash key is equal to that of the current prediction unit (PU). It is crucial to find a hash function suitable for maintaining search quality and reducing search complexity. A hash-based IME search algorithm [3] that adopts a hash function based on product-quantization is used to find intra block copy mode of 8×8 PUs for screen content coding. Another hash-based search [4] that employs a random hash function based on cyclic redundancy code is used only for inter prediction mode of all $2N \times 2N$ PUs, while other PUs are searched with the TZS algorithm. The hash-based search in [5] employs a hash function utilizing both CRC and quantization code.

All of the above can give better coding performance for screen contents, but they have two problems. First, they use fixed hash functions that do not change according to the video contents, although the search complexity is very sensitive to the content of the input sequence. Second, their search algorithms are inefficient because all reference blocks are searched in a reference picture, but most of the best matching-blocks are located near to the current PU.

In this paper, we propose a new hash-based IME search algorithm that addresses these two issues. First, the proposed algorithm selectively changes hash functions in the CTU level to keep the search complexity constant. To generate CTU-level hash functions, we employ two methods to reduce the search complexity in finding hash functions: finding a hash function based on product quantization (PQ) [6] and reusing, if necessary, the hash function for the previous CTU with similar statistics. To improve the coding efficiency, secondly, we generate hash keys for all 4×4 block within each PU and use all blocks whose hash keys are equal to any of one of these hash keys in the IME process as search candidate.

The rest of this paper is organized as follows. In Section 2, we briefly describe a typical hash-based search algorithm. Then a new hash-based search algorithm is proposed in Section 3, focusing on its computational complexity control. We describe a method to adaptively find a hash function for each CTU in Section 4. The IME algorithm using hash-based search is proposed in details in Section 5 and experimental results are presented in Section 6, which is followed by the conclusion in Section 7.

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019

IEICE Electronics Express, Vol.16, No.1, 1–12

2 Hash-based search scheme

Integer motion estimation (IME) is a complex and essential algorithm for searching blocks that can compress the current PU, PU_{cur}, most efficiently in video coding. A hash-based search algorithm uses all blocks whose hash key are equal to that of the PU_{cur}. Thus, reprocessing for calculating hash keys for all PUs in the reference pictures is performed in advance in order to utilize a hash-based search algorithm.

Fig. 1 shows a typical hash-based search scheme, which consists of two steps: generating hash tables in the preprocessing step and evaluating search candidates in the IME search step. First, the hash key, key_{ref}, of all blocks within reference pictures are calculated for a pre-selected hash function before encoding each picture. After then, by classifying the reference blocks according to their hash keys, hash tables are generated. And its location information is stored into a bucket identified with its hash key in a hash table. Thus, each bucket in the hash table is implemented a set of singly-linked lists, which includes all reference blocks with the same hash key.

In IME search step, a hash key for the current PU, key_{cur}, is first calculated with the same hash function used for the hash table generation. Then the bucket identified with the hash key is fetched from the hash table and all blocks in the bucket corresponds to search candidates for finding the best matching block, PU_{best}, which is a block with the minimum rate distortion cost [2].

3 Proposed hash-based search algorithm

Problem formulation: The computational complexity of real-time video encoders should be limited to be within a fixed level. Assuming that C_{target} is the target upper bound of the search complexity for IME, our goal is to maximize the coding gain while satisfying the complexity constraint with a hash-based search algorithm.

The total computational complexity C_{total} of the hash-based search algorithm for a CTU includes three components like the following.

$$C^k_{total} = C^k_{search} + C^k_{buildHT} + C^k_{findHF}, \tag{1}$$

where C_{search}, $C_{buildHT}$ and C_{findHF} represent the computational complexity for IME search of all possible sized PUs, for building a hash table, and for finding a hash function, respectively for each CTU. Here C^k represents the average complexity for a CTU in the k-th picture. In this paper, the subscripts k, and j are used to represent the indices of picture and CTUs in a picture, which can be omitted, if possible,

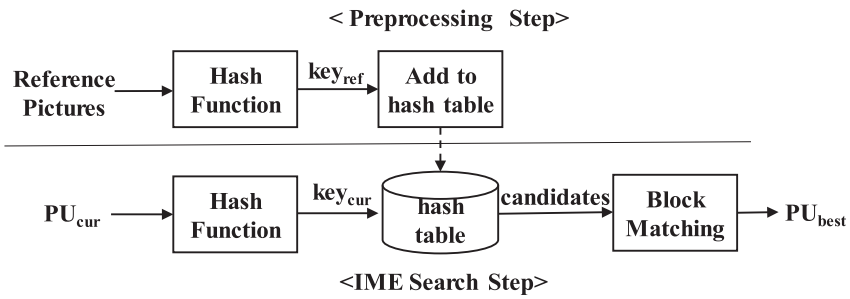


Fig. 1. A typical hash-based search algorithm

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019

IEICE Electronics Express, Vol.16, No.1, 1–12

without confusion. C_{search} can be measured as the number of block-matching for IME of all PUs in a CTU, N_{BM}, as follows.

$$C_{search} = \sum_{PU_i \in CTU} C_{BM} * S_i * N_{BM}, \tag{2}$$

where C_{BM} is the computational complexity of block-matching for one block and S_i is a scaling factor of the computational complexity for a PU relative to that for an 8×8 PU considering the various-sized PUs in a CTU. For example, S_i for a 16×16 PU is equal to 4.

Assuming that B, N_{HT}, $N_{hash\text{-}key}$, and N_{total} represent the bit length of hash keys, the number of hash tables used for IME search, the number of hash keys used for IME search per a PU, and the total number of 4×4 blocks stored in the hash table, respectively, the expectation value of the number of 4×4 blocks explored to find the best matching block, N_{BM}, can be calculated as follows.

$$N_{BM} = N_{HT} * N_{hash\text{-}key} * \left(\sum_{z=0}^{2^B-1} N_{total} * p_r(z) * p_c(z) \right), \tag{3}$$

where $p_r(z)$ is the probability for key_{ref} being equal to z, and $p_c(z)$ is the probability for key_{cur} being equal to z.

The computational complexity for building a hash table, $C_{buildHT}$, is proportional to the number of blocks in hash tables as follows.

$$C_{buildHT} = C_{hashing} * N_{total}, \tag{4}$$

where $C_{hashing}$ is the computational complexity of generating a hash key for one block and inserting it into hash tables, which depends on the hash function.

Lastly, C_{findHF} depends on the algorithm that adaptively determines a hash function which will be explained in Section 4.

Proposed approach: To alleviate the problems described in Section 1, we propose a new hash-based search algorithm with three steps: 1) finding hash functions to generate hash tables for the current picture, 2) building hash tables for the current pictures to be referenced by the following pictures and 3) performing IME search for the current picture, as shown in Fig. 2, where the additional blocks to a typical hash-based search are represented as shaded-boxes.

First, the proposed algorithm finds hash functions for each CTU while keeping C_{total} less than and close to C_{target}. In estimating C_{total}, we assume that C_{search}, $C_{buildHT}$ and C_{findHF} for the current picture are replaced with those for its immediately previous picture in the same temporal layer or their adjusted values according to the bit length of the hash key B and a threshold value that determines whether to find a hash function through data learning or to reuse the hash function. We will discuss it more in the Section 4.

After determining the hash function for the current CTU, a hash table is generated for the current CTU. For example, a hash key for each 4×4 block in the CTU_{jk} is generated with its hash functions h_{jk} and is stored into a corresponding bucket in the CTU-level hash table HT_{jk}. Therefore, all 4×4 blocks in the same CTU are mapped into the same hash table. Note that not all possible hash keys are generated and the distribution of hash keys depends on the video contents. Through experiments, we found that only about 30–50% of all possible hash keys actually

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019

IEICE Electronics Express, Vol.16, No.1, 1–12

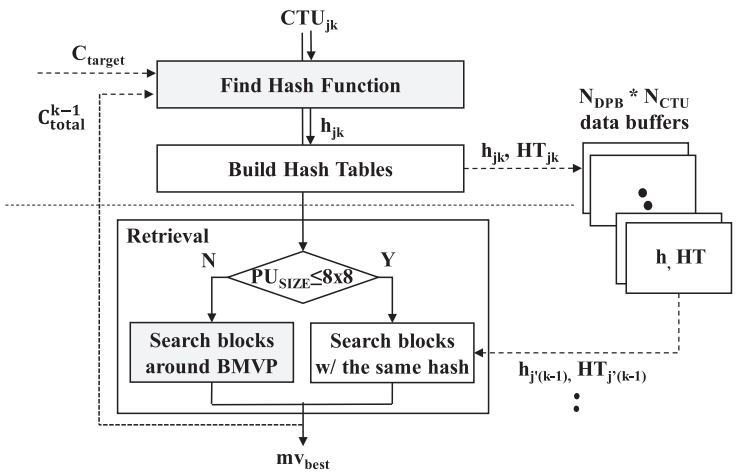


Fig. 2. Flowchart of the proposed hash-based search algorithm

are generated. If all pixels in a block are the same, they will not be stored in the hash table because they can be coded using other prediction modes of the HEVC. Experimental results show that it can reduce IME search complexity by approximately 50–70% with negligible coding loss (0.01∼0.02%).

For the IME search algorithm referencing CTU-level hash tables, its hash function and hash table for each CTU must be stored together into data buffers considering the reference by the following pictures. The size of the buffer needs to be equal to the product of the number of CTU in a picture, N_{CTU}, and the number of the decoded picture buffer (DPB), N_{DPB}. Furthermore, they should be kept until the reference pictures used for building a hash table are removed from the DPB.

In the IME search step, the best-matching block is searched among all search candidates through block matching. To efficiently reduce IME search complexity, two methods are employed here. First, we employ the bottom-up MVPs (BMVPs) [7] for PUs of size larger than 8×8. Note that when the depth of the current PU is d in the BMVP algorithm, its MVP candidates consists of the best motion vectors of its corresponding PUs with depth of $d + 1$. Second, we employ hash-based search only for PUs of size 4×8, 8×4, and 8×8 where search candidates consist of all blocks with a hash key equal to any of one of 4×4 blocks in the current PU.

Consequently, we can divide the complexity of hash-based IME search C_{search} into two terms such as $C_{bottom\text{-}up}$ for find the BMVPs and $C_{hash\text{-}search}$ for finding the best matching block among blocks with the same hash key in the hash table, which is represented as follows.

$$C_{search} = C_{hash\text{-}search} + C_{bottom\text{-}up} \tag{5}$$

4 Searching an optimal hash function for each CTU

Hash function: In this paper we focus on using product quantization [6] in determining hash functions suitable for IME search. The hash functions based on PQ have a form of the Cartesian product of multiple quantized feature components, each of which can be quantized independently to minimize its expected distortion.

Assuming that X is a 4×4 block, the input of the hash function h is m tuple $(f_1(X), \ldots, f_m(X))$, each of which is a feature-value for m-th feature calculated from the block. And the output of the hash function h is m tuple $(q_1(f_1(X)), \ldots,$

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019

IEICE Electronics Express, Vol.16, No.1, 1–12

$q_m(f_m(X)))$, each of which is a quantized value of the feature-value. Thus, it can be written as follows.

$$(q_1(f_1(X)), \ldots, q_m(f_m(X))) = h(f_1(X), \ldots, f_m(X)), \tag{6}$$

where m features are selected in a predefined feature set. Features included in the feature set used for experiments in this paper are summarized in Table I.

How to determine the parameter of the hash function: C_{search} is the sum of $C_{hash\text{-}search}$ and $C_{botton\text{-}up}$ as shown in Eq. (5). Even though the number of BMVPs to be used in the IME process can be slightly changed according to the input content, it does not really make any significant difference in $C_{botton\text{-}up}$. Thus it is assumed to be approximately the same as the previous pictures of the same temporal layer for simplicity. As a result, any change of C_{search} due to the input content is mostly from that of $C_{hash\text{-}search}$.

In the Eq. (3) for $C_{hash\text{-}search}$, N_{HT}, $N_{hash\text{-}key}$ and $p_r(z)$ are values that can be obtained at the time of determining the parameter of the hash function for the current CTU. Because N_{HT} is determined from the number of CTUs in the IME search area which is generally fixed. As $N_{hash\text{-}key}$ is equal to the number of 4×4 blocks in a PU, $N_{hash\text{-}key}$ is fixed to 4 for 8×8 PUs and 2 for 8×4 or 4×8 PUs. And the distribution for $p_r(z)$ can be calculated from the hash table generated in the previous pictures.

On the other hand, N_{total} and $p_c(z)$ are not. Especially, $p_c(z)$ can't be determined until all hash-based IME searches are completed for the current CTU. To determine the $C_{hash\text{-}search}$, we just assume that the image change between two consecutive pictures is not large. Consequently, N_{total} and $p_c(z)$ can be approximated as the average value of N_{total} and $p_r(z)$ of previous pictures, respectively. Based on these assumption, the $C_{hase\text{-}search}$ can be approximated as follows.

$$C_{hash\text{-}search} \approx \sum_{PU_i \in CTU} S^*{}_i * \left(\sum_{z=0}^{2^B-1} p_r(z)^2 \right) = \sum_{PU_i \in CTU} S^*{}_i * F = \alpha_{est} * F, \tag{7}$$

where $S^*{}_i$ is a kind of constant corresponding to the product of C_{BM}, S_i, N_{HT}, $N_{hash\text{-}key}$ and N_{total}. For the convenience, the sum of squares of $p_r(z)$ in Eq. (7) is represented with F, a complexity factor, which can be interpreted as a measure of the hash-based search complexity for a hash key distribution. Note that the complexity of the hash-based search is proportional to F, which is the smallest if the distribution

Table I. Features included in the feature set

features	descriptions	equations
DC	Average value of luminance pixels	$\frac{1}{W \cdot H} \cdot \sum_{x=0}^{W} \sum_{y=0}^{H} p_{cur}(x, y)$
SATD	Sum of absolute transformed differences	$\lvert H \cdot (P\text{-}DC) \cdot H^T \rvert$
ACTx	Horizontal activity	$\frac{1}{(W-1) \cdot H} \cdot \sum_{x=0}^{W-1} \sum_{y=0}^{H} \lvert p(x+1, y) - p(x, y) \rvert$
ACTy	Vertical activity	$\frac{1}{W \cdot (H-1)} \cdot \sum_{x=0}^{W} \sum_{y=0}^{H-1} \lvert p(x, y+1) - p(x, y) \rvert$
Gx	Average horizontal edges	3×3 Sobel filter
Gy	Average vertical edges	

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019

IEICE Electronics Express, Vol.16, No.1, 1–12

of hash keys is uniform. The complexity factor of the hash-based search F can be determined from several combinations of B and how to make a hash function. Among them, we choose a B and a hash function that minimizes the difference of C_{target} and the expectation of C_{total}. The hash function is determined based on PQ.

Then, assuming that C_{findNC}, $N_{iteration}$ and B_d are the computational complexity of finding the nearest centroid to the feature-value of the block when a block is inputted, the number of iteration of the k-means clustering algorithm [6] and the bit length of hash key of the d-th feature, the search complexity to find a hash function based on PQ, C_{findHF}, is as follows.

$$C_{findHF} = \sum_{d=1}^{m} C_{findNC} * N_{total} * N_{iteration} * 2^{B_d}, \tag{8}$$

where the sum of B_d for m features is equal to B. In the Eq. (8), $N_{iteration}$ is assumed to be equal to that of the average of the previous picture, for simplicity. As a result, the C_{findHF} depends on only B and can be estimated as follows.

$$C^{k}_{findHF} \approx \left(\sum_{d=1}^{m} 2^{B^{k-1}{}_d} \Big/ \sum_{d=1}^{m} 2^{B^{k}{}_d} \right) * C^{k-1}_{findHF} = \beta_{est} * C^{k-1}_{findHF}, \tag{9}$$

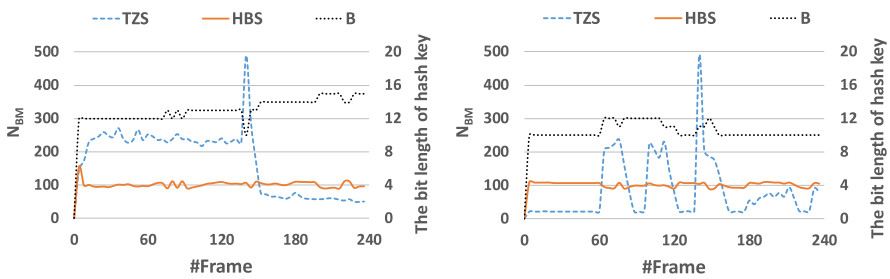
where $B^{k-1}{}_d$ and $B^{k}{}_d$ are the bit length of hash key of the d-th feature for (k-1)-th and k-th pictures in the same temporal layer, respectively.

On the other hand, $C_{botton\text{-}up}$ and $C_{buildHT}$ does not depends on B. Consequently, C_{total} can be estimated with the Eq. (7) and (9) as follows.

$$C^{k}_{total} \approx \alpha_{est} * C^{k-1}_{hash\text{-}search} + \beta_{est} * C^{k-1}_{findHF} + C^{k-1}_{bottom\text{-}up} + C^{k-1}_{buildHT}, \tag{10}$$

where the α_{est} and β_{est} are determined from the average of α_{real} and β_{real} the previous 3 pictures, respectively. The α_{real} and β_{real} are values calculated from the real search complexity from obtained after encoding a picture. And B and m are initially set to 12 and 3, respectively, and B_d is equally assigned for 3 features.

Fig. 3 shows the average number of search points for an 8×8 PU in TZS and hash-based IME search (HBS) picture by picture, where the bit length of hash key of the k-th picture means that used hash functions of the k-th picture. This shows that by adjusting B the search complexity of a hash-based IME search can be kept almost constant. And it is still working well even the scene-change point where the average number of search candidates for the TZS algorithm reduces drastically. For the experiment, we set the number of search points for 8×8 PUs in the hash-based



(a) Kimono sequence at QP22 (b) sc_map sequence at QP22

Fig. 3. The average number of search points for an 8×8 PU in TZS and HBS

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019

IEICE Electronics Express, Vol.16, No.1, 1–12

search to a fixed 100. Thus, the average of search points of HBS may be larger than that of TZS in scenes where there is little change in the input image.

How to find a hash function: Fig. 4 shows a simplified flowchart of the algorithm to find a hash function suitable for each CTU, which consists of two paths: reusing and finding hash functions. To reduce the computational complexity for finding hash function, first it determines in the reusing step whether or not one of hash functions for the previous coded CTUs is used for a hash function of the current CTU, CTU_{cur}. Otherwise, it tries to find a new hash function that minimizes similarity distortion.

We employ a heuristic that there exists a positive linear correlation between the similarity of two CTUs and the similarity of their hash functions to reduce the computational complexity C_{findHF}. Therefore, we first find the CTU (CTU_{MS}) most similar to the CTU_{cur} among the previous coded CTUs by using a similarity measure between two CTUs and another similarity measure of their hash functions which are defined as follows, respectively.

$$\text{similarity } (CTU_1, CTU_2) = |\mu_1 - \mu_2| + |\sigma_1 - \sigma_2| \tag{11}$$

$$\text{similarity } (h_1, h_2) = |F_1 - F_2|, \tag{12}$$

where μ_{jk} and σ_{jk} are the mean and the standard deviation of DC of blocks in the CTU_{jk}, respectively, and F_{jk} is a complexity factor of the distribution of hash keys for CTU_{jk} with its corresponding hash function h_{jk}.

After finding CTU_{MS}, its hash function h_{MS} is selected as the initial hash function of the CTU_{jk}, h^i_{jk}. The complexity factor F^i_{jk} is calculated from the distribution of hash keys, which are generated from sampled blocks in CTU_{cur} with the hash function h^i_{jk} to further reduce C_{findHF}. Experiments show that it does not degrade in performance if the sampling ratio is larger than 1/8. If F^i_{jk} is close to F_{MS}, h^i_{jk} is used as the hash function of CTU_{cur}, h_{jk}. Otherwise, the finding step is performed.

It is practically impossible to find the optimal set of important features for hash keys at runtime because there exist too many search candidates and the coding performance and quality for each search must be evaluated through encoding. Thus, we decided to determine the number of features off-line just to reduce the search complexity of hash functions at runtime. As summarized in Fig. 5, when the target

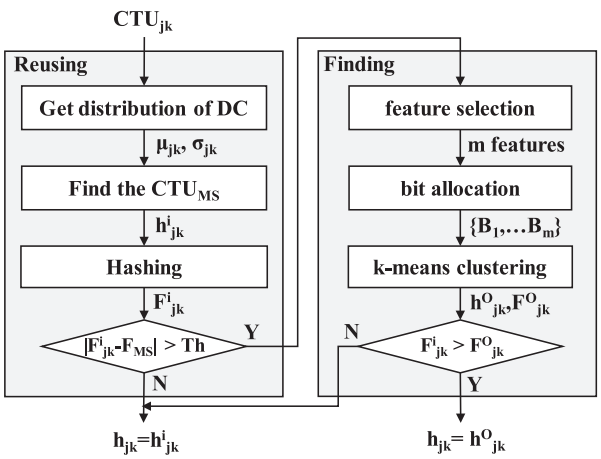


Fig. 4. A simplified flowchart for finding a hash function suitable for each CTU.

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019

complexity is set to 100, 200, 300, 500 and 1000, and the number of features are 2, 3, and 4, the total complexity and the coding performance are compared after encoding the first 32 frames of all the sequences in Table II Through this experiment, we found that the number of features m depends on the C_{target}. If the C_{target} is roughly less than $400\,K*C_{BM}$, it is appropriate to set it to 3, otherwise 2. Consequently, the number of features is limited to be 3, based on the assumption than C_{target} is less than $400\,K*C_{BM}$.

At runtime, the hash function is adaptively determined through three processes such as feature selection, bit allocation, and k-means clustering based on PQ. In the feature selection, one feature with the greatest variance is first chosen from the pre-selected feature set to distinguish blocks in the current CTU. Then features with smaller correlation with already selected features are chosen to have a set of selected features with more expressive power. In the bit allocation, the bit-length for each feature are determined in order of the variances based on the greedy algorithm. Finally, a hash function is determined through the k-means clustering algorithm based on PQ, which minimizes the sum of the difference between feature-values and its nearest centroids as follow.

$$\text{minimize}\left(\sum_{d=1}^{m}\sum_{e=1}^{N_{total}}|f_d(X_e) - C(q(f_d(X_e)))|\right), \tag{13}$$

where X, f(X), q(f(X)) and C(q(f(X))) are a 4×4 input block, a feature-value of X, a quantized value of f(X) and the nearest centroid of f(X), respectively.

After finding a hash function, the complexity factor F^O is calculated. Mostly, the F^O is smaller than F^i, but it is not always true. Thus, in case of $F^O > F^i$, the hash function h^i is chosen finally.

Both C_{findHF} and $C_{hash\text{-}search}$ are dependent on the threshold value Th to determine the ratio for reusing hash functions. Let's assumed that $p_{finding}$ is the number of CTU of performing the finding step divided by the total number of CTU. As $p_{finding}$ increases, C_{findHF} is increased but $C_{hash\text{-}search}$ is mostly decreased because F^O is on the average smaller than F^i. Therefore, it is necessary to find the optimal threshold that minimizes C_{total}.

According to the experiment results, there are two cases. In the first case, C_{total} is minimized when the threshold is 0 as shown in Fig. 6(a), which is mainly

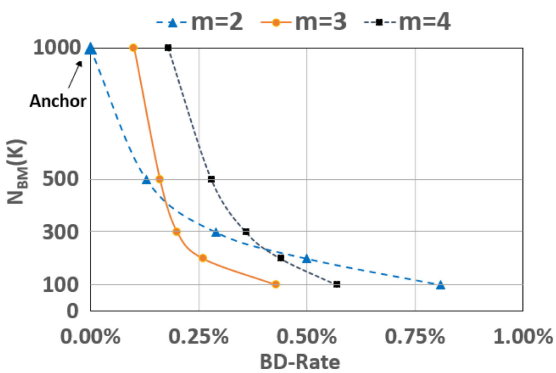
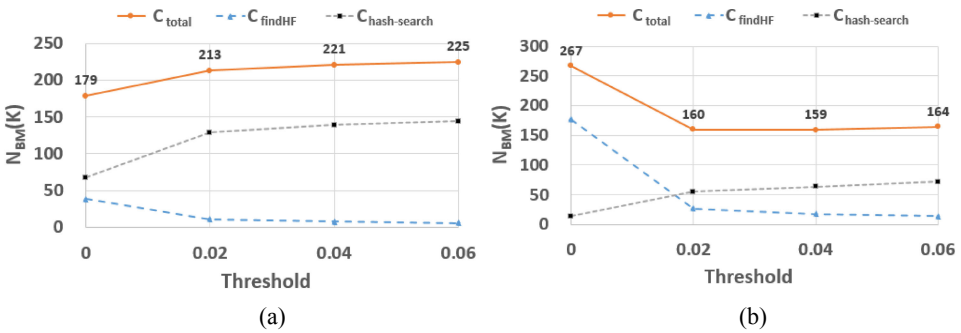


Fig. 5. The total complexity and the coding performance when the target complexity is set to 100, 200, 300, 500 and 1000 for three different number of features.

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019

IEICE Electronics Express, Vol.16, No.1, 1–12



(a) (b)

Fig. 6. C_{total}, C_{findHF} and $C_{hash\text{-}search}$ by the threshold when (a) B = 9 (b) B = 15

occurred when the bit length of the hash key is short and F is large. This is because C_{findHF} is much smaller than $C_{hash\text{-}search}$ so that there is not much room for further reduction of the C_{findHF}. In the second case, there is a minimum value of C_{total} at a threshold value other than zero as shown in Fig. 6(b). Note that the first case occurs if C_{findHF} is smaller than $C_{hash\text{-}search}$ when the threshold is 0, otherwise the second case. In the second case the minimum of C_{total} occurs when the threshold value is between 0.02 and 0.05, and the variation of C_{total} in the range is not large. Based on these observations, the threshold of the k-th picture is determined as follows.

$$Th^k = \begin{cases} 0 & \text{if } C^{k\text{-}1}_{findHF(Th=0)} < C^{k\text{-}1}_{hash\text{-}search(Th=0)} \\ 0.03 & \text{otherwise} \end{cases}, \tag{14}$$

where $C^{k\text{-}1}{}_{findHF(Th=0)}$ and $C^{k\text{-}1}{}_{hash\text{-}search(Th=0)}$ are the average value of C_{findHF} and $C_{hash\text{-}search}$ of the (k-1)-th picture when the threshold is 0. But thresholds used for all CTUs of the (k-1)-th picture are not always equal to 0, thus they are not available. In this case, $C^{k\text{-}1}{}_{findHF(Th=0)}$ and $C^{k\text{-}1}{}_{hash\text{-}search(Th=0)}$ are estimated by assuming that the complexity of the reusing step is zero as follows, respectively.

$$C^{k\text{-}1}_{findHF(Th=0)} \approx C^{k\text{-}1}_{findHF}/p^{k\text{-}1}_{finding}, \quad C^{k\text{-}1}_{hash\text{-}search(Th=0)} \approx C^{k\text{-}1}_{hash\text{-}search} \tag{15}$$

In Eq. (14), we selected the fixed threshold to 0.03, based on the observations because there is no large difference in the value of C_{total} according to the variation of Th, which can be negligible compared to the estimation error of the complexity of the current picture with the complexity of the previous picture.

5 Proposed hash-based integer motion estimation

Fig. 7 shows the pseudo code for the proposed hash based IME of each PU, which are consisted of 2 stages: hash-table selection stage and hash-based search. The hash-table selection stage is performed before hash-based search. At first, we select a CTU with the smallest lower bound of the CTU-level RD cost which can be calculated by finding the motion vector of block with the shortest distance to AMVP among all PUs in the CTU_{jk}, assuming that distortion is zero. If CTU_{jk} is included in the IME search range and if its lower bound of the RD cost is smaller than the current minimum RD cost, CTU_{min}, its corresponding hash table HT_{jk} and hash function h_{jk} are fetched for block matching.

Secondly, hash-based search is performed where a hash key is generated for each 4×4 subblock. A hash-key, key_s is calculated with h_{jk} for each 4×4 subblock in the current PU where s is the index of subblocks in the raster-scan

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019

hash-table selection stage

```
:: doIntegerMotionEstimation (PU_cur, #Ref.Picture)
Cost_min = 0
for (k=0; k<#Ref.Picture; k++)
    AMVP_k = getAMVP(k, PU_cur)
    CTU_jk = getCTUwithSmallestCost(k, AMVP_k)
    if ( isInSearchArea(CTU_jk) && CTU-Cost(CTU_jk) < Cost_min )
        HT_jk = getHashTable(CTU_jk) , h_jk = getHashFunction(CTU_jk)
        Cost_min = doHashbasedSearch (PU_cur, h_jk , HT_jk , AMVP_k )
```

hash-based search stage

```
:: doHashbasedSearch (PU_cur, h_jk , HT_jk , AMVP_k )
  for (s=0; s<4; s++)
    key_s = h_jk (PU_cur, s)
    candidate-list_s = getCandidateList(key_s, HT_jk )
    while ( isNonEmpty(candidate-list_s))
      refPU_st = getReferencePU(candidate-list_s)
      if ( isNonReferenced(refPU_st)) D_st = doBlockMatching( refPU_st, PU_cur)
      if (D_st = 0 ) stop hash-based IME
      Cost_min = updateRDCost (Cost_min, D_st, AMVP_k, refPU_st )
```

Fig. 7. The pseudo code of the proposed hash-based IME for a PU

order in PU_{cur}. The subblocks with hash key, key_s in HT_{jk} are included in the s-th candidate list for PU_{cur}. Denoting that the t-th subblock is taken from the s-th candidate list and the position of the top-left pixel of the subblock is (x_{st}, y_{st}), the position of the top-left pixel of the reference PU for block-matching, $refPU_{st}$, can be derived by aligning both the current PU and the reference PU at the s-th subblock position corresponding to key_s and is calculated as follows.

$$(x_{st} - \mathrm{mod}(s,2) * 4, \quad y_{st} - \mathrm{floor}(s,2) * 4), \tag{16}$$

where floor(s, 2) and mod(s, 2) are the quotient and remainder of dividing s by 2.

Note that it is possible that a referenced PU can be referenced again from different candidate lists. To avoid duplication computation, it can be checked whether the block is referenced before performing block matching. Experimental results show that the number of bock matching can increase about 15 to 20% without the duplication check. Note that if the distortion calculated between the current PU and a reference PU is zero, the hash-based IME can be terminated because a perfectly matching PU is already found. Otherwise, this process is repeated until all reference PUs in the candidate-lists are searched and all reference CTUs in the search area are referenced and all reference pictures are referenced.

6 Experimental results

The proposed IME search algorithm was integrated into HM-16.9+SCM8.0 [8]. And 12 video sequences for screen contest [9] listed up in Table II were used for simulation tests. The coding performance is measured by the BD-rate [10], which is obtained with the weighted average of the Y, U and V components of 6, 1 and 1 for 4 quantization parameter values: 22, 27, 32 and 37 under lowdelayB common test condition. As an anchor for BD-rate calculation, the algorithm in [2] is used.

Table II shows comparison of total search complexity and BD-rate of reference algorithms and our algorithm. The "Avg. C_{total}" means the average search complexity of the full sequence. For comparison, [2], [4], [7] are used. They employ TZS, TZS+HBS and TZS+bottom-up search (BTS) for IME, respectively. Note that the search range for TZS and HBS are set to ±64 and the full-range, respectively. The total search complexity of [2] and [7] adopted TZS is considered IME search

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019

Table II. Comparison of total search complexity and BD-rate

	Avg. C_{total} (K)						BD-rate (%)				
sequence name	**[2]**	**[4]**	**[7]**	**120**	**150**	**200**	**[4]**	**[7]**	**120**	**150**	**200**
sc_flyingGraphics	441	656	88	124	150	199	−0.92	0.35	1.91	1.04	−0.73
sc_desktop	97	102	19	120	150	185	−18.8	0.16	−15.6	−19.1	−19.3
sc_console	248	260	50	121	150	191	−19.6	0.23	−13.7	−18.7	−19.5
MissonControlClip3	90	148	18	120	150	182	−8.78	0.10	−8.83	−12.0	−12.3
sc_web_browsing	61	39	12	113	113	113	−31.0	0.10	−24.7	−31.8	−32.2
sc_map	30	51	6	120	123	123	−0.40	0.28	−0.43	−0.56	−0.59
sc_programming	82	186	16	118	150	175	−1.33	0.12	−0.94	−3.04	−3.21
SlideShow	304	325	61	124	150	195	−0.1	0.14	−0.91	−0.81	−0.79
sc_robot	139	321	28	124	150	210	0.00	0.26	0.77	0.62	0.60
Basketball_Screen	225	289	45	122	150	191	−3.42	0.29	−3.18	−4.11	−4.15
MissionControlClip2	208	250	44	125	150	194	−5.63	0.20	−4.30	−7.94	−8.01
ChinaSpeed	545	780	104	130	150	203	−0.22	0.31	1.50	0.91	0.54
screen contents avg.	**206**	**284**	**41**	**122**	**145**	**180**	**−7.52**	**0.21**	**−5.70**	**−7.96**	**−8.30**

complexity only. And the total search complexity of [4] is included the complexity for IME search and building a hash table. On the other hand, ours combines HBS and BTS for IME and the search range is set to ±320. And the target complexity is set to the level equivalent to the complexity of 120 K*C_{BM}, 150 K*C_{BM} and 200 K*C_{BM}, which are denoted as 120, 150 and 200 in the table, respectively. However, when the target complexity is 120 K*C_{BM} and 200 K*C_{BM}, even though the complexity is controlled, the total search complexity is not precisely controlled. This is because the total search complexity of most sequences is ranging from 100 to 200. And, *sc_map* and *sc_web_ browsing* doesn't change the total complexity even though the target complexity is changed. Because, in that sequences, the IME search is early terminated due to lots of perfectly matching blocks, thus there is not much room to control search complexity. According to our experiment, the proposed algorithm gives an additional coding gain of about 5∼8% relative to [2] and [7]. It can reduce the overall search complexity to about half on average, while keeping the coding efficiency similar to [4].

7 Conclusion

We have proposed a hash-based search for integer motion estimation in this paper. The proposed method selectively changes hash functions in the CTU level to keep the search complexity constant. To generate CTU-level hash functions, we employ two methods: finding a hash function based on PQ and reusing the hash function for the previous CTU. To improve the coding efficiency, we generate hash keys for all 4×4 block within each PU and use all blocks whose hash keys are equal to any of one of these hash keys in the IME process as search candidate. Consequently, the proposed algorithm provides similar or better coding performance for screen contents while keeping the complexity at a lower level and constant compared with the reference algorithms.

DOI: 10.1587/elex.15.20181019
Received November 14, 2018
Accepted November 26, 2018
Publicized December 12, 2018
Copyedited January 10, 2019