A Unified Framework of Hash-based Matching for Screen Content Coding

Bin Li [#1], Jizheng Xu [#2], Feng Wu [*3]

[#] *Microsoft Research*
No. 5 Danling Street, Haidian District, Beijing, China
[1,2] {libin, jzxu}@microsoft.com

[*] *Dept. of EEIS, University of Science and Technology of China*
No. 96 Jinzhai Road, Hefei, Anhui, China
[3] fengwu@ustc.edu.cn

***Abstract*—This paper introduces a unified framework of hash-based matching method for screen content coding. Screen content has some different characteristics from camera-captured content, such as large motion and repeating patterns. Hash-based matching is proposed to better explore the correlation in screen content, thus, improving the coding efficiency. The proposed method can handle both intra picture and inter picture block matching with variable block sizes in a unified framework. The proposed framework is also easy to be extended to handle other motion models to further improve the coding efficiency of screen content. We also develop fast encoding algorithms to make full use of the hash results. The experimental results show the proposed algorithm achieves about 12% bit saving while saving more than 25% encoding time. The bit saving is up to 57% and the encoding time saving is up to 60% for the proposed method.**

***Index Terms*—High Efficiency Video Coding (HEVC), Screen Content Coding (SCC), hash, block matching, motion estimation**

I. Introduction

High Efficiency Video Coding (HEVC) version 1 has been finalized since Jan. 2013 [1]. To meet the requirements on screen content coding, ITU-T and ISO/IEC issued a Call for Proposal (CfP) for screen content coding [2].

Different with camera-captured content, screen content has several new features, such as sharp content, large motion, unnatural motion and repeating patterns. The different characteristics between screen content and camera-captured content make the encoding of screen content different from camera-captured content. To achieve high coding efficiency for screen content, new coding tools should be considered to explore the correlation in screen content.

The characteristics of screen content, such as large motion and repeating patterns, require the screen content coding to explore non-local correlations to achieve good compression ratio. Thus, conventional motion estimation and intra block copy estimation, which usually perform a search in a small range, may not work well for screen content, as only local correlations can be explored by the conventional block search methods. Moreover, conventional fast motion estimation methods usually assumes the smoothness of the content, while it is no longer true for screen. Thus, conventional fast motion estimation algorithms are not suitable for screen content too. To utilize non-local correlations, we propose a unified framework of block matching in the whole picture.

A straightforward full picture block matching method is full search, which means that for every target block, an encoder needs to compare it with every possible blocks in the whole picture and then select the block which best predicts the current block. For every $m \times n$ block (m and n are the width and height of the block respectively) in the a picture, the encoder needs to check $w \times h$ block candidate (w and h are the picture width and height respectively. Strictly speaking, the number of block candidates should be $(w-m+1)\times(h-n+1)$. But considering that usually w is much larger than m and h is much larger than n, we just use $w \times h$ for simple.). For each block candidate, the encoder needs to compare $m \times n$ pixels. Thus, the complexity of full search is about $O(l \cdot m \cdot n \cdot w \cdot h)$, where l is the number of blocks in the whole picture.

For example, if we want to perform full picture search for all the 64x64 blocks in 1080p video, the overall operations for full search are about $500 \times 64 \times 64 \times 1920 \times 1080 \approx 4T$. (Actually, there are not integer number 64x64 blocks in 1080p picture. We use the number of 500 here and in the rest of paper to simplify the discussion.) The complexity of full picture search is too high for an encoder. Thus, we propose hash-based method to make the large scale block matching practical.

It is noted that hash-based methods were used in some other work to encode screen contents, e.g., Lempel-Ziv based methods [3][4], 2-D dictionary coding [5]. Compared with previous works, we provide a framework to use hash-based methods for block based screen content coding, which has the following contributions:

- We can handle both intra block estimation and inter motion estimation in a unified way.
- We can easily handle variable block sizes in our frame-

IEEE VCIP'14, Dec. 7 - Dec. 10, 2014, Valletta, Malta.
978-1-4799-6139-9/14/$31.00

work.

- Other complicated motion models can be easily integrated.
- We propose to use line-based hashing to reduce the complexity of hash table generation.
- We identify blocks that do not benefit from hash-based method and thus reduce the complexity of hash table generation and search.
- We also present fast encoding algorithms based on hash-based search.

We have integrated the proposed framework into the latest HEVC reference software[6]. Experimental results show that for screen contents, the proposed algorithm not only improves the coding efficiency, but also saves the encoding time significantly.

The rest of this paper is organized as follows. Sec. II introduces the proposed hash-based matching algorithm in details. Experimental results are provided in III Sec. IV concludes the whole paper.

II. Hash-based Matching

The screen content has a characteristics of noiseless, which means that we can perform exact block match rather than approximate block match. Hash has a good feature that if the hash values of two inputs are different, the two inputs are absolutely different. Thus, we can use hash value to check whether two blocks are identical. According to the noiseless of screen content, this paper proposes hash-based block matching for screen content. There are mainly two steps in hash-based block matching, hash table generation and block matching. They will be introduced in Sec. II-A and Sec. II-B respectively.

A. Hash Table Generation

We choose CRC (Cyclic Redundancy Check) as the hash function to generate the hash value for every block. The CRC calculation only involves table checking, bit shifting and other bit operation. The complexity of calculating CRC value relates to the magnitude of input, i.e., the block size. The total complexity of generating all hash values of all the blocks in a picture is about $O(m \cdot n \cdot w \cdot h)$, relating to the block size and picture size. For example, to calculate all the hash values of 64x64 blocks in 1080p video, about 8G operations are required.

Although 8G operations for hash table generation is much less than 4T operations for full picture search, we still want to further reduce the complexity by reuse the intermediate results. For one block, instead of calculating hash value H_{block} for the whole block, we calculate the hash values for each row $H_{row}[i]$ first. And then we group the row hash values in a block together to calculate a new hash value of the grouped row hash values. This hash value will be used as the block hash value H_{block}.

Fig. 1 shows the hash value calculation process for a 64x64 block. First the hash values of each rows are generated, and then we group the row hash values together as a 1D array $H_{row}[0]H_{row}[1].....H_{row}[63]$. A new hash value will be calculated for the 1D array and this hash value will be used as the block hash value. When we need to calculate the hash value for the block one row below the current block (Row[1] to Row[64]), 63 of the intermediate data could be reused ($H_{row}[1]$ to $H_{row}[63]$). Thus, the complexity of hash table generation will be reduced from $O(m \cdot n \cdot w \cdot h)$ to $O(m \cdot w \cdot h + n \cdot w \cdot h)$, where the first part is the complexity of generating intermediate results and the second part is the complexity of calculating the final hash values. In such a case, only about 256M operations are required to calculate all the hash values of 64x64 blocks in 1080p video.

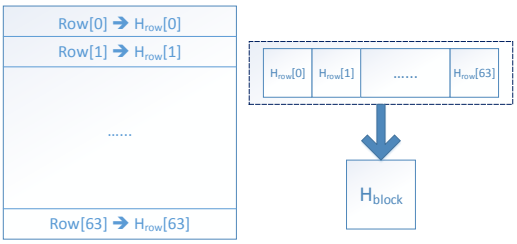

Fig. 1. Block Hash Value Calculation

B. Block Matching

We can use the hash values to check whether the two blocks have identical content or not quickly. A straightforward method is that for each block, we only need to calculate the hash value, whose complexity is about $O(m \cdot n)$, and compare the hash value with the hash values of all possible blocks, whose complexity is about $O(w \cdot h)$. Taken the number of blocks in a picture into consideration, the overall complexity is about $O(l \cdot (m \cdot n + w \cdot h))$. Using the same example as above, about 1G operations are required for all the 64x64 blocks in 1080p video.

To further reduce the complexity of block matching step, we re-arrange the hash table in form of the inverted index. With the help of inverted index, the encoder does not need to compare the hash values block by block. Instead, it only needs to check all the blocks having the same hash value and select one block to predict the current block. In such a case, the overall complexity for the block matching step is about

$$O(l \cdot (m \cdot n + C)) \tag{1}$$

where $m \cdot n$ is the complexity of calculating the hash value of the current block and C is the number of blocks having the same hash value. When C is relatively small compared with $m \cdot n$, it is negligible. For example, if 16-bit hash value is used, there are 31.6 blocks corresponding to a same hash value on average. Compared with $m \cdot n = 4096$ for 64x64 blocks, C is negligible. In the same example as above, only about 2M operations are required for the block matching step.

The overall complexity of the proposed hash-based block matching method is about $O(m \cdot w \cdot h + n \cdot w \cdot h + l \cdot m \cdot n)$, for both hash table generation step and block matching step. In the example of 64x64 blocks in 1080p video, the overall operation

Fig. 2. Homogeneous Region in the First Picture of Map Sequence

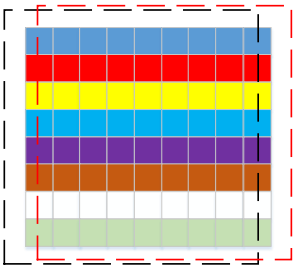

Fig. 3. Every Row Has a Same Pixel Value

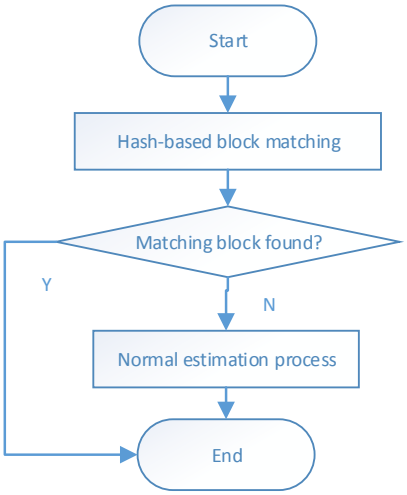

Fig. 4. Hash-based block match and normal search

is reduced from 4T (full picture search) to 258M (hash-based block matching), which is more than $15000\times$ speedup. Some other implementation details are provided in the next part.

C. Other Considerations and Implementation Details

As shown in Eq. (1), the overall search complexity relates to C. When C is not large, it is negligible. Thus, we should avoid the case of leading to very large C for some hash values.

Usually, the homogeneous region, such as background region, may lead to one hash value corresponding to many blocks. For example, as shown in Fig. 2, the a large region only contains one color. When block 1, 2, 3, 4 and all the blocks in this region will have the same hash value as exactly identical pixels values are contained in all these blocks. Thus, to avoid very large C values, the blocks satisfying at least one of the following conditions are not included in the hash table.

- Every row has only single pixel value. In this case, as shown in Fig. 3, the black box is the current block and every row in the current block has a single pixel value. When the block moves to the right by one pixel, as in the red box, most likely the new block has identical pixel values as the previous block, thus leading to identical hash values.
- Every column has only single pixel value. The situation is similar to the above case.

As these blocks are not included in the hash table, we need to evaluate the impact on the coding efficiency if removing them from the hash table. Actually, the blocks excluded from the hash table are not difficult to predict. As every row or column of the blocks only has single pixel values, horizontal prediction or vertical prediction can almost predict the blocks perfectly. Thus, excluding these blocks from hash table only benefit for the block matching step and will not harm the coding efficiency significantly.

In our implementation, the hash table for different block sizes are maintained in a unified hash table. 18-bit hash value is used. The higher 2 bits are determined by the block size and the lower 16 bits are determined by the CRC values. Using only 16-bit CRC value will not cost too much memory. But as there are only 65536 different values within 16-bit, hash collision cannot be refrained. Thus, we need to further check whether the two blocks are identical even if they have the same hash value. A simple way to check whether two blocks are identical is to compare the two blocks pixel by pixel. However, it will introduce additional complexity. We use a second 24-bit CRC value to check whether the two blocks are identical. If both the 16-bit CRC value and the second 24-bit CRC values are the same for two blocks, we treat the two blocks having identical contents (although there is still possibility that they have different content, but it is very low). Please note that we only use the first hash value (plus the block size bits) to build inverted index table, such that the memory cost for the table index is about 256K (18-bit).

The hash-based block matching can work together with the conventional intra BC estimation process and inter motion estimation process. If exact match is found, the normal estimation process will be skipped. Otherwise, the normal estimation process is also invoked to find approximate match as usual, as shown in Fig. 4.

D. Fast Encoding Algorithms

Besides skipping the conventional block estimation process if an exact match is found by hash-based block matching as described above, some other mode checking can also be skipped if we are confident that we have already found a block good enough to predict the current one. In our implementation, if all the following conditions are satisfied, the remaining mode checking process including further splitting current CU

(Coding Unit) into sub-CUs are skipped.

- Exact match is found.
- The quality of the block used for prediction is no worse than the expected quality of the current block.
- The current CU depth is 0.

E. Extendability

As mentioned previously, complicated motion may exists in screen content. Thus, how to find a block to predict the current block well is very important for screen content. Block matching with new motion model is easy to incorporated into the proposed hash-based block matching scheme. For example, if we want to perform block matching considering block flipping, instead of calculating the hash values of the flipped blocks in the reference region and add them into the hash table (which may double the memory requirement of the hash table), we only need to calculate the hash value of the flipped current block and compare this hash value with the hash values of the original blocks in the reference region. It is not difficult to take other motion model into consideration under the hash-based block matching framework, without significant impact on the complexity and memory requirement.

III. Experimental Results

The proposed algorithm is implemented on HM-13.0+RExt-6.0 [6]. The HM-13.0+RExt-6.0 is used as the anchor of all the comparisons. A total of 12 test sequences specified in the screen content coding CfP [2] are used for testing. All the sequences are in both RGB format and YUV format. Three coding structures, All Intra (AI), Random Access (RA), and Low Delay (LD) are tested. Both lossy coding and lossless coding are tested. Fix QP encoding is applied for lossy coding. The coding efficiency is measured as average Y (or G, the first component in encoding) BD-Rate [7] for lossy coding (negative number means performance gain) and average bit saving for lossless coding (positive number means performance gain). The experimental results are provided in Table I and Table II.

From the tables we can know that, the proposed hash-based matching algorithm improves the coding efficiency by 12.1%, 12.0%, and 11.5% for AI, RA, and LD lossy coding, respectively. Meanwhile, the proposed algorithm saves about 33% and 26% encoding time for RA and LD lossy coding, respectively. The results for lossless coding is similar to those of lossy coding. The maximum bit saving for lossy coding is about 57% for WebBrowsing sequence in RGB format under LD coding structure. The maximum encoding time saving about 60% for WebBrowsing sequence in YUV format under RA coding structure. It is very clear that the proposed unified hash-based matching framework can brings both coding efficiency improvement and encoding complexity reduction.

TABLE I
Average BD-Rate for Lossy Coding

	AI	RA	LD
RGB, text & graphics, 1080p	−23.3%	−23.6%	−23.1%
RGB, text & graphics, 720p	−13.7%	−13.7%	−15.1%
RGB, mixed content, 1440p	−7.0%	−3.7%	−1.8%
RGB, mixed content, 1080p	−5.2%	−5.6%	−3.6%
RGB, Animation, 720p	0.0%	0.0%	0.0%
YUV, text & graphics, 1080p	−22.1%	−23.2%	−22.3%
YUV, text & graphics, 720p	−12.4%	−12.7%	−14.0%
YUV, mixed content, 1440p	−7.5%	−4.8%	−2.5%
YUV, mixed content, 1080p	−5.5%	−6.5%	−4.3%
YUV, Animation, 720p	0.0%	0.0%	0.0%
Average	−12.1%	−12.0%	−11.5%
Encode Time[%]	104%	67%	74%
Decode Time[%]	94%	101%	99%

TABLE II
Average Bit Saving for Lossless Coding

	AI	RA	LD
RGB, text & graphics, 1080p	21.2%	24.1%	24.7%
RGB, text & graphics, 720p	8.2%	8.5%	8.5%
RGB, mixed content, 1440p	5.9%	1.4%	0.8%
RGB, mixed content, 1080p	3.0%	1.1%	0.6%
RGB, Animation, 720p	0.0%	0.0%	0.0%
YUV, text & graphics, 1080p	22.0%	24.2%	24.4%
YUV, text & graphics, 720p	9.9%	10.1%	10.2%
YUV, mixed content, 1440p	6.2%	1.5%	0.9%
YUV, mixed content, 1080p	3.3%	1.2%	0.7%
YUV, Animation, 720p	0.0%	0.0%	0.0%
Average	9.9%	9.6%	9.5%
Encode Time[%]	111%	73%	75%
Decode Time[%]	95%	100%	99%

IV. Conclusion

This paper introduces a unified framework of hash-based matching algorithm for screen content coding. The proposed algorithm not only improves the coding efficiency by 12% on average for lossy coding and up to 57%, but also saves encoding time significantly. Other motion models are also easy to be incorporated into proposed framework to further improve the coding efficiency. For screen content, the proposed framework is a very useful tool explore the non-local correlation and to improve the coding efficiency.

References

[1] G. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, no. 12, pp. 1649–1668, 2012.

[2] ITU-T Q6/16 Visual Coding and ISO/IEC JTC1/SC29/WG11 Coding of Moving Pictures and Audio, "Joint Call for Proposals for Coding of Screen Content," ISO/IEC JTC1/SC29/WG11 MPEG2014/N14175, Jan. 2014.

[3] C. Lan, J. Xu, and F. Wu, "Compression of compound images by combining several strategies," in *Multimedia Signal Processing (MMSP), 2011 IEEE 13th International Workshop on*, Oct 2011, pp. 1–6.

[4] T. Lin, P. Zhang, S. Wang, K. Zhou, and X. Chen, "Mixed chroma sampling-rate high efficiency video coding for full-chroma screen content," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 23, no. 1, pp. 173–185, Jan 2013.

[5] W. Zhu, W. Ding, J. Xu, Y. Shi, and B. Yin, "2-D Dictionary Based Video Coding for Screen Contents," in *Data Compression Conference (DCC), 2014*, March 2014, pp. 43–52.

[6] HM, HEVC test Model, [Online], available at: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/.

[7] G. Bjontegaard, "Improvements of the BD-PSNR model," Document VCEG-AI11, Berlin, Germany, Jul. 2008.