An HEVC-based Screen Content Coding Scheme

Bin Li and Jizheng Xu

***Abstract*—This document presents an efficient screen content coding scheme based on HEVC framework. The major techniques in the scheme includes hash-based large-scale block matching, dictionary mode, palette mode, adaptive color space coding, and several improvements to intra block copy mode. This scheme was submitted as our response to the joint Call for Proposals (CfP) for coding of screen content issued by ISO/IEC JCT1/SC29/WG11, i.e. MPEG and ITU-T Q6/16, i.e. VCEG. Compared with other coding schemes in response to the joint CfP, the proposed scheme shows clearly advantages - 1) It is the only one that shows less encoding time compared with the HEVC reference scheme; 2) it has the best lossless coding performance; 3) it is one of the two best performing schemes in terms of lossy coding. Compared with HM-13.0+RExt-6.0, which is the state-of-art screen content coding system when responding to the joint CfP, the proposed coding scheme for screen content achieves the average bit saving of 20.3% $\sim$ 26.4% for lossy coding. The average bit saving for losslss coding is 15.5% $\sim$ 23.7% for different coding structures.**

***Index Terms*—High Efficiency Video Coding (HEVC), Screen Content Coding (SCC), hash, dictionary mode, adaptive color space coding.**

I. Introduction

HIGH Efficiency Video Coding (HEVC) version 1 was finalized in January 2013 [1]. Compared with its predecessor, H.264/MPEG-4 AVC [2], HEVC doubles the compression ratio when achieving similar visual quality [3]. To meet the requirements on more applications, several extensions to HEVC version 1, including range extensions, scalable extensions, 3D video extensions, have also been developed [4]. Besides the above extensions, screen content coding extension is also an extension of HEVC which is under development by the Joint Collaborative Team on Video Coding (JCT-VC). The Call for Proposal (CfP) of HEVC Screen Content Extension was issued in January 2014 [5].

Screen content has several new features not previously available in camera-captured content, including

- Sharp Content. Compared with camera-captured content, screen content usually has with sharp edges. The sharp edges make the transform, which works well to reduce redundancy for camera-captured content, not suitable for screen content. To help the encode sharp content, transform skip [6] [7] [8] has been designed for screen content.
- Large motion. Screen content may contain large motions. For example, when browsing a web page, a large motion exists when scrolling the page. Conventional motion estimation algorithm may not be able to handle large motion well. Thus, new motion estimation algorithms to handle the large motions for screen content may be required.

Emails: {libin, jzxu}@microsoft.com

- Unnatural motion. Screen content may contain unnatural motions, such as fading in and fading out. The conventional motion model, which usually only takes translational motions into consideration, may not be able to handle screen content well.
- Repeating patterns. Repeating patterns are also very common in screen content. For example, the screen content may contain the same letter many times. The same letters belong to a repeating pattern and using previous letters to predict the current letter may be quite efficient. To utilize the correlation among repeating patterns, Intra Block Copy (IBC) [9] has been developed. With the help of IBC, similar blocks can be predicted and encoded efficiently.

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. Several new coding tools are developed to improve the coding efficiency for screen content in the proposed coding scheme.

Hash-based large scale block matching is designed to handle the repeating patterns and large motions in screen content. Although repeating patterns are very common in screen content, it is still not easy to find a matching block in a narrow search area. Also, the conventional motion estimation strategy only performs motion estimation in a narrow range. When the motion is very large, the conventional motion estimation strategy may not be able to find a matching block, even if there does exist an exact match block in its reference picture. Hash-based block matching can do IBC estimation and motion estimation in a very large range. Thus, hash-based block matching can better utilize the correlation in screen content and improve the coding efficiency. Furthermore, several fast encoding algorithms according to the results from hash-based block matching are also developed to further speed up the encoding process.

Dictionary mode is also supported in the proposed coding scheme to help encode screen content, considering the repeating pattern is very common in screen content. Unlike IBC, which removes the redundancy of repeating patterns at the block level, dictionary mode removes the redundancy of repeating patterns at the pixel level, which could be more flexible. A hash-based search strategy is also developed to help the encoder quickly make decisions.

The native color space of screen-captured content is usually RGB, as it is captured from the screen directly. Also, the screen content may also be directly encoded in the RGB color space. Compared with another commonly used color space YUV, there exists stronger correlations among different color components in the RGB color space. To solve the problem,

the proposed coding scheme uses adaptive color space coding, which adaptively selects from the RGB color space and the YCoCg color space, to improve the coding efficiency of screen content.

Several other improvements are also included in the proposed coding scheme, such as palette mode and several improvements to the IBC mode. Also fast encoding algorithms are also developed to speed up the encoder. The experimental results show the proposed coding scheme can improve the coding efficiency of screen content significantly with reduced encoding time.

The proposed coding framework has already been submitted as a response to the joint CfP [10]. Compared with other coding schemes in response to the joint CfP, the proposed scheme has many clearly advantages - 1) It is the only one that shows less encoding time compared with the HEVC reference scheme; 2) It shows the best lossless coding performance; 3) It is one of the two best performing schemes in terms of lossy coding [11].

The rest of this technical report is organized as follows. Section II overviews the proposed scheme in general. Section III $\sim$ Section V describe the newly introduced coding tools used in the CfP. Section VI introduces other modifications and fast encoding algorithms. The experimental results are provided in Section VII, followed by the conclusion in Section VIII.

II. Scheme Overview

This section briefly introduces the framework of the proposed HEVC-based screen content coding scheme.

Similar to HEVC, a hybrid coding framework is used in the proposed scheme. Fig. 1 shows the overall framework of the proposed coding scheme. Basically, it is indeed an HEVC coding framework plus several new designed coding modes. Considering the characteristics of screen content, several new coding tools are designed for intra prediction. They will be described in details in the rest of the paper. Besides intra coding tools designed for screen content, the motion estimation strategy is also modified to better explorer the correlation in screen content.

To achieve the best coding efficiency, Rate-Distortion Optimization (RDO) as shown in Eq. 1 [12] is applied to select the best coding mode.

$$minD + \lambda R \tag{1}$$

In Eq. 1, D is the distortion, R is the bit cost, and λ is the Lagrange multiplier, which is determined by the optimization target.

With the help of RDO, all the possible modes, including the coding modes existing in HEVC and the newly added coding modes designed for screen content, are tested using Eq. 1. And the mode with the smallest RD Cost ($D + \lambda R$) is selected as the best mode for the current Coding Unit (CU).

III. Hash Based Large Scale Block Matching

As mentioned in Section I, screen content has different characteristics than camera-captured 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, which is no longer true for screen content. Thus, conventional fast motion estimation algorithms are not suitable for screen content too. To utilize non-local correlations, large-scale block matching in the whole picture is proposed in this paper.

A straightforward full picture block matching method is full search, which means that for every target block, an encoder needs to compare it with all possible blocks in the whole picture and then selects 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 $w \times h$ to simplify.). 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 a full picture search for all the 64x64 blocks in 1080p picture, 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 the 1080p picture. We use the number of 500 here and in the rest of paper to simplify the discussion.) The complexity of the full picture search is too high for an encoder. Thus, we need to find some methods to reduce the encoding complexity to make the block matching practical. Sec. III-A will introduce the two main parts of the proposed hash-based block matching. Sec. III-B will describe some implementation details. And Sec. III-D will provide some analysis on the extendability of incorporating new motion models into the proposed hash-based block matching method.

A. Hash Based Block Matching

The screen content is a characterized by noiseless, which means that we can perform exact block matching rather than approximate block matching. If the hash values of two inputs are different, the two inputs are absolutely different. Thus, we can use the hash value to check whether two blocks are identical. According to the noiseless characteristics 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 III-A1 and III-A2 respectively.

1) Hash Table Generation: We choose CRC as the hash function to generate the hash value for every block. The CRC calculation only involves table checking and simple shifting and bit operation. The complexity of calculating the CRC value relates to the block size. The total complexity of

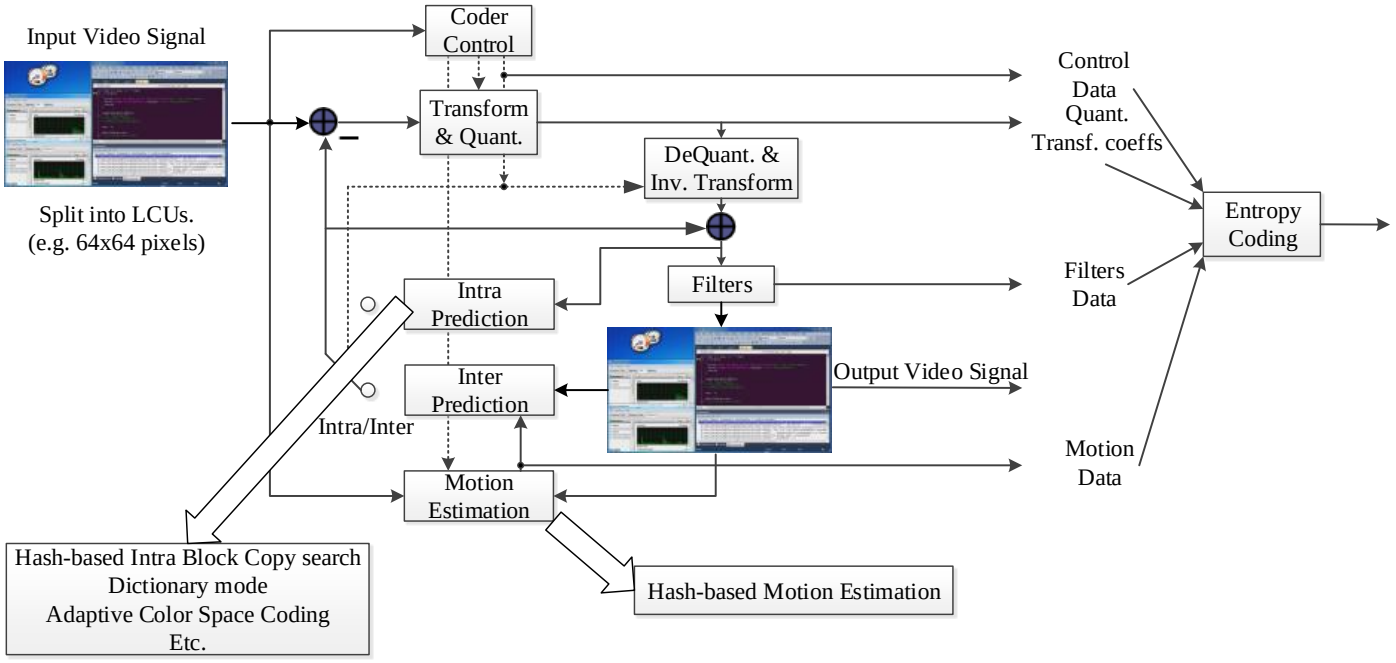


Fig. 1: Framework of the Proposed Coding Scheme

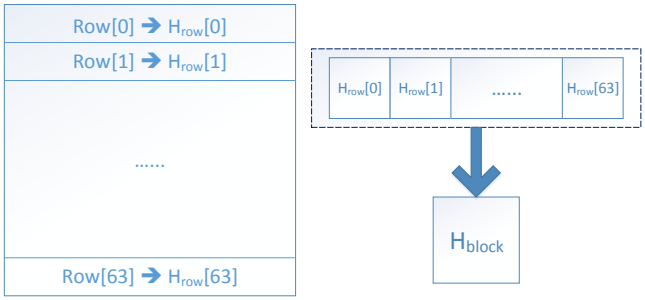


Fig. 2: Block Hash Value Calculation

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

For example, Fig. 2 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 of 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 which 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 can 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 the 1080p video.

2) Block Matching: We can use the hash values to quickly check whether the two blocks have identical content or not. 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 $O(w \cdot h)$. Taking 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 the 1080p video.

To further reduce the complexity of block matching step, we re-arrange the hash table in form of an inverted index. With the help of the inverted index, the encoder does not need to compare the hash values block by block. Instead, it only need to check all the blocks that have the same hash value and select one block to predict the current block (for example, the encoder can select the block nearest to the current block used as prediction because it will cost fewer bits to represent motion vectors or block vectors). In such a case, the overall complexity for the block matching step is about

$$O(l \cdot (m \cdot n + C)) \quad (2)$$

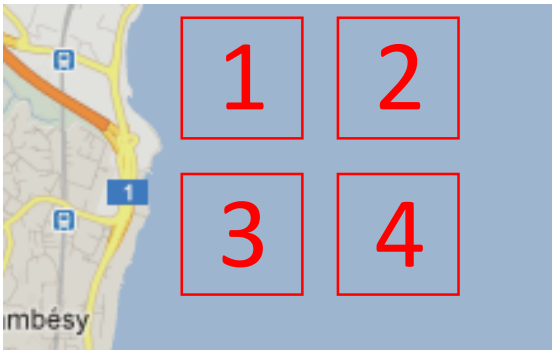


Fig. 3: Homogeneous region

where $m \cdot n$ is the complexity of calculating the hash value of the current block and C is the number of blocks with 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 that same hash value on average. Compared with $m \cdot n = 1024$ 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 the 1080p video, the overall operation is reduced from 4T (full picture search) to 258M (hash-based block matching), which is more than $15000\times$ speedup. Thus, the large scale block matching is practical with the help of hashes. Some other implementation details are provided in the next part.

B. Other Considerations and Implementation Details

As shown in Eq. (2), the overall search complexity relates to C. When C is not large, it is negligible. Thus, we should avoid getting a very large value of C for some hash values.

Usually, the homogeneous region, such as a background region, may lead to one hash value corresponding to many blocks. For example, as shown in Fig. 3, the whole region contains only one color. Blocks 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 a single pixel value. In this case, as shown in Fig. 4, 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 a only single pixel value. The situation is similar to the above case.

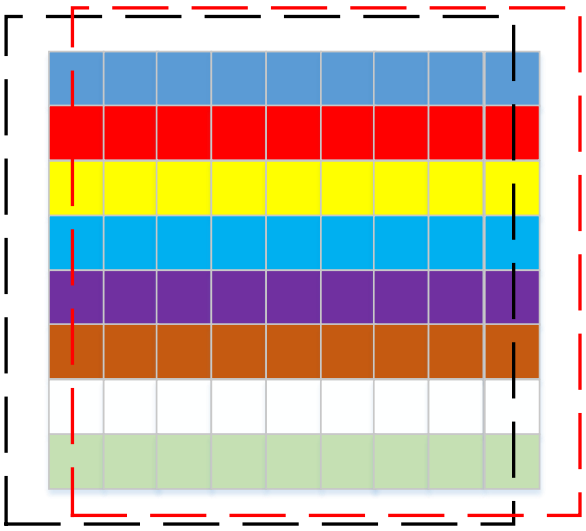


Fig. 4: Every Row Has the Same Pixel Value

As these blocks are not included in the hash table, we need to evaluate the impact of removing them from the hash table. The blocks not included in the hash table are not difficult to predict. As every row or column of the blocks only has singular pixel values, horizontal prediction or vertical prediction can almost predict the blocks perfectly. Removing these blocks from hash table only benefits the block matching step and will not harm the coding efficiency.

In our implementation, the hash tables for different block sizes are maintained in a unified hash table. An 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 a 16-bit CRC value will not cost too much memory. But as there are only 65536 different values within the 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, this 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 as having identical contents (although there is still the possibility that they have different content, but this is very low). Please note that we only use the first hash value (plus the block size bits) to build an inverted index table, such that the memory cost for the table index is about 256K (18-bit).

The proposed hash-based block matching algorithm is enabled for both Intra BC searching and Inter motion estimation. The hash-based block matching can work together with the conventional Intra BC estimation process and Inter motion estimation process. If an exact match is found, the conventional estimation process will be skipped. Otherwise, the normal estimation process is also invoked to find an approximate

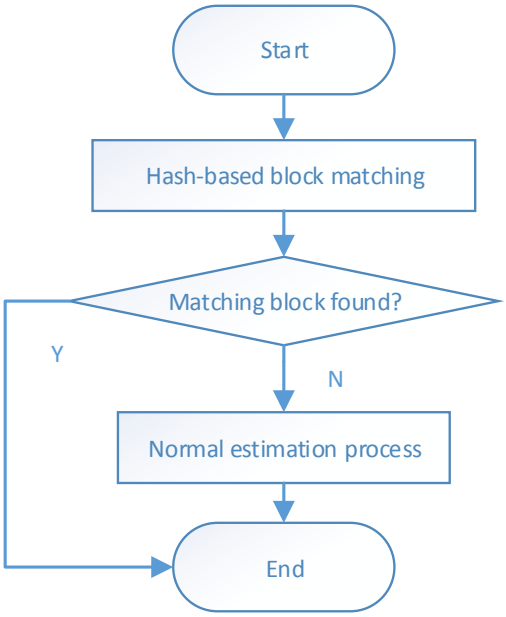


Fig. 5: Hash-based block match and normal search

match as usual, as shown in Fig. 5.

C. Fast Encoding Algorithms

Besides skipping the conventional block estimation process if an exact match is found by hash-based block matching as described in Sec. III, 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 following mode checking process including further splitting current CU into sub-CUs are skipped.

- An exact match is found.
- The quality of the block used for prediction is no worse than the expected quality of the current block. The situation is similar to the above case.
- The current CU depth is 0, as we are more confident if we find an exact match for large blocks.

D. Further Extension

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 models is easy to incorporated into the proposed hash-based block matching scheme. For example, if we want to perform block matching considering vertical flipping, instead of calculating the hash values of the flipped blocks in the reference region and adding 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 models into consideration under the hash-based block matching framework, without significant impact on the complexity and memory requirements.

IV. Dictionary Mode

Dictionary mode is supported in the proposed framework. There have been several attempts made on the dictionary mode [13], [14], [15]. The main work in this paper is the encoder side decision for dictionary mode. The basic idea of dictionary mode will be briefly described in Sec. IV-A. Sec. IV-B and IV-C describe the encoder side decision for the dictionary mode and fast encoding algorithms for dictionary mode.

A. Basic Idea for Dictionary Mode

In dictionary mode, 2-D pixels are converted to 1-D signals according to a predefined scanning order. In our implementation, both horizontal scan and vertical scan are supported. The previously reconstructed pixels are used to predict the current CU. For every pixel, there are two modes, predict mode and direct mode. In the predict mode, offset and matching lengths are signaled. The offset is used to indicate the start position to predict the current pixel. The offset can be 1-D or 2-D, associating two types of dictionary modes supported, which will be explained later. The matching length is used to indicate the number of pixels sharing the same offset. In the direct mode, the pixel values will be singled directly, without any prediction. Two types of dictionary modes are supported in our implementation.

The first type is normal dictionary mode, which is similar to Lempel-Ziv coding [16], [17]. A virtual dictionary is maintained to stored the previously reconstructed pixels values. All the reconstructed pixel values coded in dictionary mode will be stored in the virtual dictionary. 1-D offset relative to the position of the current pixel in the dictionary is signaled to indicate the pixel used for prediction, as shown in Fig. 6. When the current dictionary size reaches a predetermined size, some of the oldest elements will be removed from the dictionary. To simply the dictionary shrink operation, the removing process is only invoked after encoding/decoding one CTU. In our implementation, when the dictionary size is greater than or equal to $(1 << 18) + (1 << 17)$ after encoding/decoding one CTU, the oldest $(1 << 17)$ elements are removed from the dictionary.

The second type is reconstruction based dictionary mode. No virtual dictionary needs to be maintained in this dictionary mode. All the previously reconstructed pixels belonging to the same slice and tile can be used for prediction. 2-D offset relative to the position of the current pixel in the picture is signaled to indicate the pixel used for prediction. Fig. 7 shows an example of an 8x8 CU using a (horizontal scan) reconstruction based dictionary mode. The first matching length is 3, followed by a second matching length of 17. The relative position is kept when performing prediction.

The syntax elements in the CU coded with the dictionary mode are generally parsed as shown in Table I. If the current CU is coded with dictionary mode, the dictionary_type_flag

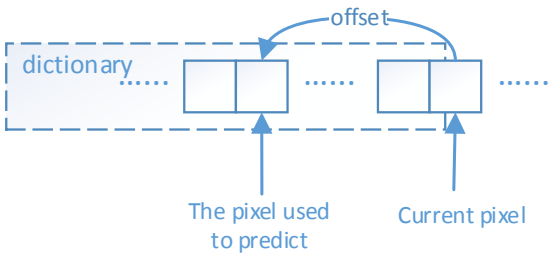


Fig. 6: Normal Dictionary Mode

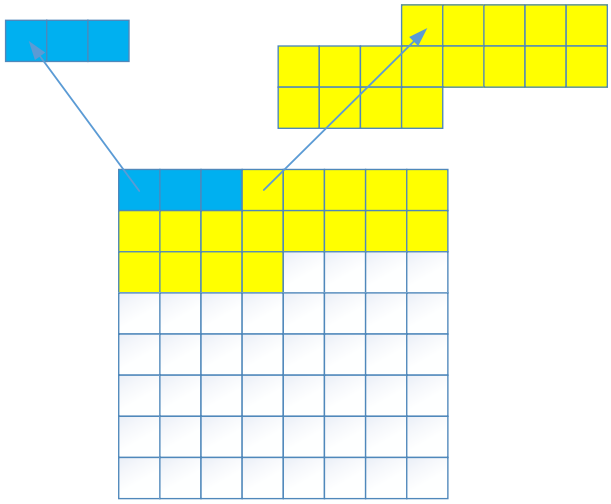


Fig. 7: Reconstruction Based Dictionary Mode

(to distinguish between normal dictionary mode or reconstruction based dictionary mode) and dictionary_scan_flag (to distinguish horizontal or vertical scanning) are read from the bitstream first. For every pixel, a dictionary_pred_mode_flag is read from the bitstream to distinguish whether the current pixel uses prediction mode or direct mode. If prediction mode is used, offset (1-D or 2-D) and the matching length are read from the bitstream. Otherwise, the pixel value is read from the bitstream.

For both dictionary modes, hash based searching is used on the encoder side to speed up the encoding process, which will be described in the following section.

B. Encoder Decision for Dictionary Mode

We use normal dictionary mode as an example to demonstrate the encoder side decision. The encoder side decision for reconstruction based dictionary mode is similar to that of normal dictionary mode. The data structure used for normal dictionary mode is organized as in Fig. 8. We maintain a hash table to indicate the position of the first match for 1 pixel in the dictionary. For every pixel in the dictionary, we maintain four positions to indicate the following:

TABLE I: Parsing Process for Dictionary Mode

```
dictionary_mode(cuSize) {
    totalPixel = cuSize*cuSize
    currPixel = 0
    dictionary_type_flag
    dictionary_scan_flag
    while (currPixel < totalPixel ) {
        dictionary_pred_mode_flag
        if( dictionary_pred_mode_flag ) {
            dictionary_pred_offset
            dictionary_pred_length_minus1
            currPixel += (dictionary_pred_length_minus1 + 1)
        }
        else {
            dictionary_direct_pixel
            currPixel += 1
        }
    }
}
```

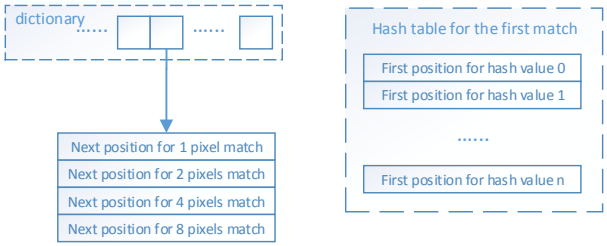


Fig. 8: Data Structure for Normal Dictionary Mode

- The next position in the dictionary with 1 pixel matching (Table 1).
- The next position in the dictionary with 2 continuous pixels matching (Table 2).
- The next position in the dictionary with 4 continuous pixels matching (Table 3).
- The next position in the dictionary with 8 continuous pixels matching (Table 4).

For every pixel, we try to find the maximum matching length in the current position. The algorithm for the encoder side decision of normal dictionary mode to find the maximum matching length is shown in Algorithm 1, and some terminology used for algorithm description is provided in Table II. The basic method for finding the maximum matching length is as follows:

- Determine the first position candidate by hash values and the other positions from the table.
- At each position candidate, check the matching length.
- If the current matching length is greater than or equal to the matching length of the next table, use the next matching table for future matching.

In our implementation, T_{max} is set to 1000. If the L_m returned by Algorithm 1 is 0, it means that no matching is found. Thus, direct mode will be used to encode the current pixel. The hash tables will be updated after one or more pixels have been encoded.

TABLE II: Terminology for Normal Dictionary Search Algorithm Description

Symbol	Definition
Pix	Pixels to be encoded.
N	Number of pixels to be encoded.
T_{max}	Maximum number of test times for one pixel.
H	Hash table to find first match.
$H[i]$	The matching position of the i-th element in H.
D	The current dictionary.
$D[i]$	The i-th pixel in the dictionary.
$D[i].N$	The next matching position tables for the i-th pixel in the dictionary.
$D[i].N[j]$	The next matching position for the j-th element in $D[i].N$. Different j corresponds to different matching length.
L_m	Maximum matching length.
P_m	The position where L_m is obtained.
L_c	Current matching length.
P_c	The position where L_c is obtained.

Algorithm 1 Find Maximum Matching Length for Normal Dictionary Mode

Input:
Pix, N, T_{max}, H, D;
Output:
L_m, P_m;
$M[4] = \{1, 2, 4, 8\}$
$L_m = 0$
$Idx = 0$
Current test times $T_c = 0$
Hash value of the current pixel $h = Hash(Pix[0])$
$P_c = H[h]$
while $T_c < T_m$ **and** $P_c \geq 0$ **and** $L_m < N$ **do**
  Determine current matching length L_c from P_c
  if $L_c > L_m$ **then**
    $L_m = L_c$
    $P_m = P_c$
  end if
  if $Idx < 3$ **and** $L_c \geq M[Idx + 1]$ **and** $D[P_c].N[Idx + 1] \geq 0$ **then**
    $Idx++$
    $P_c = D[P_c].N[Idx]$
  else
    $P_c = D[P_c].N[Idx]$
  end if
  T_c++
end while

C. Early Termination for Dictionary Mode

The dictionary mode is efficient if the matching length is large. The worst case for dictionary mode is that every pixel should be directly coded or the matching length is only 1. Thus, we can check that the average matching length of all the pixels have already been coded and terminate early if the average matching length is very small. For the pixels coded in direct mode, the matching length is treated as 1. In our implementation, if the following criterion is satisfied, the RDO process for dictionary mode is terminated and dictionary mode will not be used as the optimal mode for the current CU.

$$P_{coded} > T_{coded} \quad \&\& \quad L_{avg} < L_{thres} \tag{3}$$

TABLE III: Threshold for Dictionary Mode Early Termination

	T_{coded}	L_{thres}
Normal Dictionary Mode	64	2
Reconstruction based Dictionary Mode	20	3

where P_{coded} and T_{coded} are the number of pixels that have already been coded and the threshold for the number of pixels, respectively. L_{avg} and L_{thres} are the average matching length and the threshold of the matching length, respectively. In our implementation, the value of T_{coded} and L_{thres} are provided in Table III.

V. Adaptive Color Space Coding

Adaptive color space coding is designed for RGB coding. For most applications, video is encoded in YCbCr format, in which the correlation among different color components has been removed. But for some professional video applications (especially when the expected video quality is very high), RGB coding is applied. It has been analyzed in [18] that the distortion introduced by converting RGB to YCbCr, and then converting back is not negligible. Although there have already been several attempts to remove the redundancy among different color components in RGB coding, such as cross-component prediction [18], there is still correlation among different color components for RGB coding. Thus, it is worth considering how to further utilize the correlation among different color components to improve coding efficiency. The overall framework of adaptive color space coding will be introduced in Sec. V-A. This paper will then introduce two kinds of adaptive color space coding. The first one is color components reordering, which will be described in Sec. V-B. The second one is RGB to YCoCg conversion, which will be described in Sec. V-C.

A. Framework

In our design, adaptive color space coding is only applied to Intra coded CUs. There will be two kinds of color spaces in the encoding/decoding process. The *Main Color Space* is the color space where all the pixels will be stored in the loop. For example, in the common test condition [19] for RGB sequences, the *Main Color Space* will be GBR. The *Current Color Space* is the color space that the current CU uses. The *Current Color Space* may be the same or different from the *Main Color Space*.

Fig. 9 shows the decoding process when adaptive color space coding is applied. If the *Current Color Space* is not the same as the *Main Color Space*, the surrounding pixels (at most $4N + 1$ pixels, where N is the current CU size) will be converted from *Main Color Space* to *Current Color Space*. The original (unconverted) surrounding pixels in *Main Color Space* will also be stored, and after decoding the current CU, they will be restored. Intra prediction and reconstruction are performed in the *Current Color Space*. After the current CU has been reconstructed (before loop filters), it will be converted back to the *Main Color Space* and stored in the picture buffer. With this design, the additional memory required to support adaptive color space coding is only to store the $4N + 1$

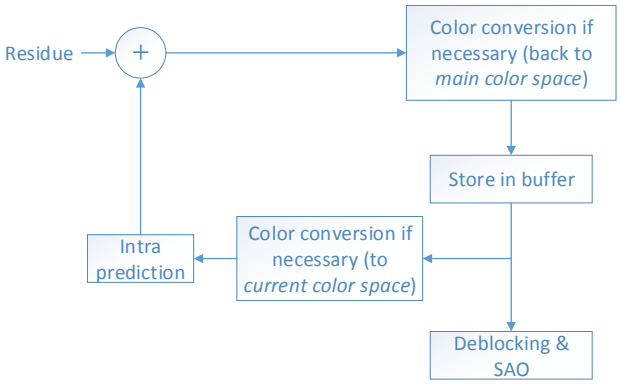
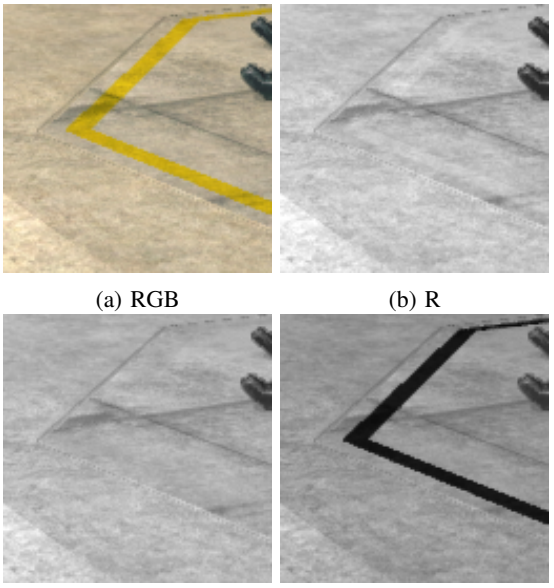


Fig. 9: Decoding Process when Adaptive Color Space Coding is Used



(a) RGB (b) R

(c) G (d) B

Fig. 10: Clip of Robot Sequence (128x128 clip in the first picture, starting from (576, 512))

pixels. For HEVC, the maximum CU size is 64. Thus, only the memory to store 257 pixel values is required to support adaptive color space coding.

B. Color Components Reordering

Color components reordering is one kind of adaptive color space coding in our design. In the color components reordering, besides GBR, an additional two color spaces, RGB and BGR are supported. The reason to support color components reordering is that all the 35 intra prediction directions could be applied to the first component in HEVC. But for the second and third components, only a few (5) intra prediction directions could be used. Thus, it is straightforward to let the most complicated color component be the first color component.

For example, Fig. 10 shows a 128x128 region in the first picture of the Robot sequence. The clear edge exists in the B component but not in the R and G components. Thus, in this case, using component B as the first component is helpful in improving coding efficiency.

In our design, color components reordering can be applied to both lossy coding and lossless coding.

C. YCoCg Coding

YCoCg color space introduced in [20] shows good compression performance. We also apply YCoCg color space in our design. The conversions between RGB and YCoCg are shown in Eq. (4) and (5).

$$\begin{bmatrix} Y \\ Co \\ Cg \end{bmatrix} = \begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 0 & -1/2 \\ -1/4 & 1/2 & -1/4 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 1 & -1 \\ 1 & 0 & 1 \\ 1 & -1 & -1 \end{bmatrix} \begin{bmatrix} Y \\ Co \\ Cg \end{bmatrix} \tag{5}$$

The Co and Cg obtained from Eq. (4) are not in the same dynamic range with input. To adjust the dynamic range, Co and Cg are further modified by adding $1 << (BitDepth-1)$. For example, if 8-bit encoding is applied, Co and Cg are further added by 128. To match with the forward transformation, Co and Cg are subtracted by the same magnitude before being used as the input in Eq. (5).

As a different color space is used in the encoding/decoding process, the quantization parameter (QP) used in the YCoCg color space should be modified according to the norm of the inverse conversion matrix as shown in Eq. (5). One unit distortion of Y brings about three units distortion on RGB, according to the norm that $(1)^2 + (1)^2 + (-1)^2 = 3$, meaning the distortion of Y will be amplified by 3 times after the inverse quantization. It is also easy to know that the distortion of Co and Cg will be amplified 2 times and 3 times respectively after the inverse color space conversion. We would like to let the magnitude of amplified distortion Y have a level similar to the distortion in the unchanged color space. It is similar to Co and Cg components. Thus, we need to adjust the QP used in the YCoCg color space a little smaller than that used in the RGB color space.

The relationship between QP and Q_{step} (Quantization step size) for HEVC is

$$Q_{step} = 2^{(QP-4)/6} \tag{6}$$

The quantization error for a given QP relates to Q^2_{Step},

$$Dist \sim Q^2_{Step} \sim 2^{(QP-4)/3} \tag{7}$$

where $Dist$ is the distortion. $Dist_{RGB}$, $Dist_Y$, $Dist_{Co}$, and $Dist_{Cg}$ are used to represent the distortion of RGB, Y, Co, and Cg, respectively.

Suppose QP_{RGB} is the QP used in the RGB color space. We want to derive QP_Y, QP_{Co}, and QP_{Cg}, which are the QP used for Y, Co, and Cg, respectively. We need to have

$$\begin{cases} Dist_{RGB} = 3 \times Dist_Y \\ Dist_{RGB} = 2 \times Dist_{Co} \\ Dist_{RGB} = 3 \times Dist_{Cg} \end{cases} \tag{8}$$

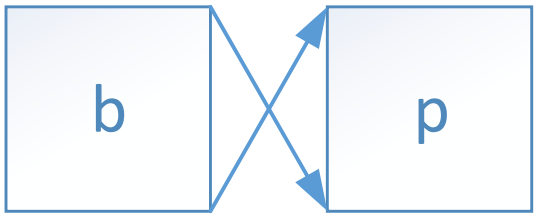


Fig. 11: An Example of Intra BC Flipping

From Eq. (8), we can derive that,

$$\begin{cases} 2^{(QP_{RGB}-4)/3} = 3 \times 2^{(QP_Y-4)/3} \\ 2^{(QP_{RGB}-4)/3} = 2 \times 2^{(QP_{Co}-4)/3} \\ 2^{(QP_{RGB}-4)/3} = 3 \times 2^{(QP_{Cg}-4)/3} \end{cases} \tag{9}$$

And finally, we can have that,

$$\begin{cases} QP_Y &= QP_{RGB} - 3 \times log_2 3 \approx QP_{RGB} - 5 \\ QP_{Co} &= QP_{RGB} - 3 \times log_2 2 = QP_{RGB} - 3 \\ QP_{Cg} &= QP_{RGB} - 3 \times log_2 3 \approx QP_{RGB} - 5 \end{cases} \tag{10}$$

From Eq. 10 we can know that when YCoCg color space is used, the QP for Y, Co, and Cg should be set to be equal to $QP_{RGB} - 5$, $QP_{RGB} - 3$, and $QP_{RGB} - 5$, respectively.

In our design, as we do not increase the bit depth in the YCoCg color space, the conversion is not revisable. Thus, YCoCg color space is only used in lossy coding.

VI. Other Coding Tools and Fast Encoding Algorithms

A. Modifications to Intra BC mode

This paper proposes several modifications to Intra BC mode in the following areas:

- Intra BC skip mode. Intra BC skip mode means the current CU uses Intra BC prediction and the partition size is 2Nx2N. There is no residue signaled for Intra BC skip mode.
- Intra BC merge mode. Intra BC merge mode is used to signal the block vector. Intra BC merge mode is only enabled for Intra BC CU with a partition size of 2Nx2N. Two merge candidates are enabled. The first merge candidate comes from the block vector used for the left block and the second merge candidate comes from the block vector used for the above block.
- Intra BC flipping. Vertical flipping is enabled for the CU using Intra BC prediction. Fig. 11 shows an example where Intra BC flipping is very efficient.

B. Palette Mode

The palette mode described in [21], with color table sharing [22], four-neighbor major color index prediction [23] and transition copy mode [24] as described in [25], is used in our response to the CfP.

C. Motion Estimation Start Point

In the motion estimation of HM-13.0+RExt-6.0 [26], one of the MVPs (Motion Vector Predictor) will be used as the start point of motion estimation. In this paper, for every reference picture, the MV for the PU with the size of 2Nx2N determined by the motion estimation process is stored on the encoder side. When performing motion estimation for the other PU sizes (non-2Nx2N PUs), the MV of the 2Nx2N PU using a same reference picture is used as another candidate motion estimation start point. The encoder compares the new motion estimation start point candidate with the original MVP and then selects the one leading to a smaller prediction cost as the motion estimation start point.

D. Fast Encoding Algorithms

To further speed up the encoding process, the following fast encoding algorithms are enabled to achieve a better trade between the coding complexity and coding efficiency.

- Less search candidates for dictionary mode. T_{max} is set to 100 for lossy coding and 500 for lossless coding.
- Early termination based on number of different colors. If the number of different colors of the first component contained in one CU is less than 50 (for lossy coding, and 100 for lossless coding), we skip checking Intra BC mode and dictionary mode. When the adaptive color space coding introduced in Sec. V is enabled, if the number of different colors of the first component in one CU is less than 10 (for lossy coding and 3 for lossless coding), skip checking the current color space. If the number of different colors of the first component in one CU is less than 20, we skip the RDO process for non-2Nx2N Intra BC partitions.
- If a hash-based Intra BC match is found, we skip all the following RDO process except checking Intra BC merge mode.
- We relax the restrictions of the fast encoding algorithms introduced in Sec. III-C. The last condition (CU depth is equal to 0) is removed.
- Fast algorithm for normal intra mode checking. In the RDO process for normal intra mode, besides MPM (Most Probable Mode), we check (2, 2, 2, 1, 1) candidates for (4x4, 8x8, 16x16, 32x32, 64x64) PU. The original checking number is (8, 8, 8, 3, 3). We disable the RDOQ (Rate-Distortion Optimized Quantization) when selecting the best Intra prediction direction.
- We disable the RDOQ for transform skip mode.

VII. Experimental Results

The proposed algorithms are implemented based on HM-13.0+RExt-6.0 [26]. The HM-13.0+RExt-6.0 is used as the

anchor for all the comparisons. It should be noted that HM-13.0+RExt-6.0 is not the anchor used for the screen content coding CfP [27]. [27] uses HM-12.1+RExt-5.1 as the anchor, but HM-13.0+RExt-6.0 improves quite a lot over HM-12.1+RExt-5.1. Considering HM-13.0+RExt-6.0 achieves the best coding efficiency when responding to the screen content coding CfP, we use HM-13.0+RExt-6.0 as the anchor for all the comparisons.

The detailed experimental conditions are described in Sec. VII-A, followed by the experimental results in Sec. VII-B. Sec. VII-C provides some discussions on the experimental results. Sec. VII-D provides the results of different trade-offs between encoding complexity and coding efficiency.

A. Test Condition

A total of 12 test sequences are used for testing. All the sequences are in both RGB format and YUV format. The characteristics of the test sequences are shown in Table IV.

Three coding structures are used in the tests.

- All Intra (AI). All the pictures are coded as Intra picture.
- Random Access (RA). Hierarchical-B coding structure is applied. Random Access is supported roughly once a second.
- Low Delay (LD). Only the first picture is coded as Intra picture and there is no delay in the encoding.

Both lossy coding and lossless coding are tested. Fix QP Encoding is applied for lossy coding. The QP setting for each sequence is provided in Table IV. It should be noted that in the screen content coding CfP, fix bitrate coding is applied as subjective testing is performed. We only use BD-Rate as the measurement in this paper. Thus, fix QP encoding is applied for simple.

B. Detailed Results

We provide the testing results for every individual tool first, including

- Hash. The experimental results of the hash-based block matching introduced in Sec. III are provided.
- Dictionary (Dict). The experimental results of dictionary mode introduced in Sec. IV are provided.
- Adaptive color space coding (ACS). The experimental results of adaptive color space coding introduced in Sec. V are provided.

Besides the experimental results of every individual tool, the results of the whole framework of screen content coding are also provided, including

- All. All the coding tools designed for screen content coding are enabled.
- Fast. All the coding tools with the fast encoding algorithms introduced in Sec. VI-D are enabled.

When testing the coding efficiency of the whole framework (not individual tool testing), the QP refinement algorithm introduced in [28] is enabled for lossy coding. The coding efficiency is measured as Y (or G, the first component in encoding) BD-Rate (for lossy coding) and bit saving (for lossless coding). The experimental results for different coding structures are provided in Table V to Table X. Several example R-D curves are shown in Fig. 12.

TABLE V: Experimental Results for AI, Lossy Coding

	Hash	Dict	ACS	All	Fast
RGB, T, 1080p	−23.3%	−37.1%	−7.1%	−45.2%	−43.2%
RGB, T, 720p	−13.7%	−21.5%	−10.3%	−32.0%	−29.3%
RGB, M, 1440p	−7.0%	−10.6%	−12.9%	−24.6%	−22.0%
RGB, M, 1080p	−5.2%	−8.5%	−8.7%	−18.7%	−15.5%
RGB, A, 720p	0.0%	0.0%	−20.2%	−19.8%	−19.2%
YUV, T, 1080p	−22.1%	−34.0%	0.0%	−41.5%	−39.7%
YUV, T, 720p	−12.4%	−16.6%	0.0%	−21.4%	−20.0%
YUV, M, 1440p	−7.5%	−10.4%	0.0%	−14.8%	−11.9%
YUV, M, 1080p	−5.5%	−8.1%	0.0%	−12.3%	−7.4%
YUV, A, 720p	0.0%	0.0%	0.0%	−0.2%	0.7%
Average	−12.1%	−18.4%	−5.3%	−26.4%	−24.1%
Encode Time[%]	104%	166%	143%	230%	138%
Decode Time[%]	94%	99%	98%	97%	101%

TABLE VI: Experimental Results for RA, Lossy Coding

	Hash	Dict	ACS	All	Fast
RGB, T, 1080p	−23.6%	−24.0%	−3.6%	−37.9%	−34.3%
RGB, T, 720p	−13.7%	−20.4%	−11.1%	−33.4%	−31.0%
RGB, M, 1440p	−3.7%	−7.3%	−16.4%	−28.4%	−26.8%
RGB, M, 1080p	−5.6%	−6.5%	−6.8%	−18.1%	−16.2%
RGB, A, 720p	0.0%	0.0%	−14.0%	−16.7%	−16.4%
YUV, T, 1080p	−23.2%	−21.4%	0.0%	−35.3%	−32.3%
YUV, T, 720p	−12.7%	−14.9%	0.0%	−23.4%	−22.1%
YUV, M, 1440p	−4.8%	−7.0%	0.0%	−14.9%	−13.2%
YUV, M, 1080p	−6.5%	−6.6%	0.0%	−13.4%	−11.1%
YUV, A, 720p	0.0%	0.1%	0.0%	−4.1%	−3.8%
Average	−12.0%	−13.8%	−4.8%	−25.7%	−23.6%
Encode Time[%]	67%	144%	107%	89%	73%
Decode Time[%]	101%	101%	98%	102%	108%

C. Discussion

From the results above, it is clear that the proposed coding scheme for screen content coding outperforms the existing coding scheme. The following discussions use Low Delay lossy coding as an example. The entire framework saves about 20.3% bits on average for all the sequences. Besides the significant bit saving, about 9% encoding time saving is also achieved, which demonstrates that the proposed framework improves the coding efficiency significantly while reducing the encoding complexity. When fast encoding algorithms are enabled, 18.6% bit saving is achieved with 22% encoding time reduction, compared with the anchor. Among the proposed coding tools, hash-based block matching and dictionary mode have the most significant performance gain. The adaptive color space coding only helps the encoding of RGB sequences, which brings up to 12.8% bit saving for *RGB, M, 1080p* sequences. The all frame works outperforms the existing coding framework significantly.

D. Different Trade-offs Between Encoding Complexity and Coding Efficiency

To further evaluate the coding efficiency of the proposed coding scheme, we developed several other fast encoding algorithms to show the performance at different trade-offs between the encoding complexity and coding efficiency. Five

TABLE IV: Test Sequences

Categrary	Sequence	Resolution	FPS	Frames	QP for lossy coding
Text & Graphics with motion (T)	FlyingGraphics	1920x1080 (1080p)	60	600	27, 32, 37, 42
	Desktop	1920x1080 (1080p)	60	600	27, 32, 37, 42
	Console	1920x1080 (1080p)	60	600	27, 32, 37, 42
	WebBrowsing	1280x720 (720p)	30	300	27, 32, 37, 42
	Map	1280x720 (720p)	60	600	22, 27, 32, 37
	Programming	1280x720 (720p)	60	600	22, 27, 32, 37
	SlideShow	1280x720 (720p)	20	500	27, 32, 37, 42
Mixed content (M)	BasketballScreen	2560x1440 (1440p)	60	300	22, 27, 32, 37
	MissionControlClip2	2560x1440 (1440p)	60	300	22, 27, 32, 37
	MissionControlClip3	1920x1080 (1080p)	60	600	22, 27, 32, 37
	SocialNetworkMap	1920x1080 (1080p)	60	300	22, 27, 32, 37
Animation (A)	Robot	1280x720 (720p)	30	300	22, 27, 32, 37

TABLE VII: Experimental Results for LD, Lossy Coding

	Hash	Dict	ACS	All	Fast
RGB, T, 1080p	−23.1%	−19.0%	−2.7%	−32.8%	−29.2%
RGB, T, 720p	−15.1%	−15.9%	−6.8%	−27.5%	−25.5%
RGB, M, 1440p	−1.8%	−4.4%	−12.8%	−20.8%	−19.9%
RGB, M, 1080p	−3.6%	−3.2%	−5.0%	−12.4%	−10.9%
RGB, A, 720p	0.0%	0.0%	−5.5%	−7.0%	−6.8%
YUV, T, 1080p	−22.3%	−15.7%	0.0%	−29.7%	−26.3%
YUV, T, 720p	−14.0%	−10.7%	0.0%	−19.8%	−18.5%
YUV, M, 1440p	−2.5%	−4.1%	0.0%	−9.8%	−9.5%
YUV, M, 1080p	−4.3%	−3.2%	0.0%	−8.1%	−6.8%
YUV, A, 720p	0.0%	0.2%	0.0%	−1.6%	−1.6%
Average	−11.5%	−10.0%	−3.2%	−20.3%	−18.6%
Encode Time[%]	74%	134%	104%	91%	78%
Decode Time[%]	99%	101%	95%	100%	107%

TABLE VIII: Experimental Results for AI, Lossless Coding

	Hash	Dict	ACS	All	Fast
RGB, T, 1080p	21.2%	42.6%	0.8%	45.5%	44.1%
RGB, T, 720p	8.2%	20.3%	0.8%	21.6%	20.9%
RGB, M, 1440p	5.9%	10.3%	0.5%	12.8%	12.1%
RGB, M, 1080p	3.0%	14.1%	0.7%	15.6%	14.1%
RGB, A, 720p	0.0%	0.0%	3.3%	3.3%	3.0%
YUV, T, 1080p	22.0%	46.0%	0.0%	48.6%	47.2%
YUV, T, 720p	9.9%	21.7%	0.0%	22.4%	21.7%
YUV, M, 1440p	6.2%	10.3%	0.0%	13.0%	12.4%
YUV, M, 1080p	3.3%	11.0%	0.0%	12.4%	10.9%
YUV, A, 720p	0.0%	0.0%	0.0%	0.1%	−0.1%
Average	9.9%	21.9%	0.5%	23.7%	22.8%
Encode Time[%]	111%	226%	124%	268%	164%
Decode Time[%]	95%	106%	99%	97%	101%

TABLE IX: Experimental Results for RA, Lossless Coding

	Hash	Dict	ACS	All	Fast
RGB, T, 1080p	24.1%	33.7%	0.7%	40.4%	38.1%
RGB, T, 720p	8.5%	12.7%	0.6%	14.9%	14.4%
RGB, M, 1440p	1.4%	1.8%	0.2%	2.9%	2.6%
RGB, M, 1080p	1.1%	6.2%	0.3%	6.8%	6.2%
RGB, A, 720p	0.0%	0.0%	0.3%	0.3%	0.3%
YUV, T, 1080p	24.2%	36.7%	0.0%	42.8%	40.6%
YUV, T, 720p	10.1%	14.7%	0.0%	16.3%	15.9%
YUV, M, 1440p	1.5%	1.7%	0.0%	2.8%	2.6%
YUV, M, 1080p	1.2%	3.7%	0.0%	4.2%	3.6%
YUV, A, 720p	0.0%	0.0%	0.0%	0.0%	0.0%
Average	9.6%	14.5%	0.2%	17.0%	16.1%
Encode Time[%]	73%	147%	103%	94%	80%
Decode Time[%]	100%	103%	98%	100%	107%

TABLE X: Experimental Results for LB, Lossless Coding

	Hash	Dict	ACS	All	Fast
RGB, T, 1080p	24.7%	30.3%	0.4%	38.5%	35.8%
RGB, T, 720p	8.5%	11.5%	0.5%	13.3%	12.8%
RGB, M, 1440p	0.8%	0.9%	0.2%	1.7%	1.4%
RGB, M, 1080p	0.6%	5.3%	0.2%	5.7%	5.2%
RGB, A, 720p	0.0%	0.0%	0.1%	0.1%	0.1%
YUV, T, 1080p	24.4%	32.9%	0.0%	40.4%	37.9%
YUV, T, 720p	10.2%	13.4%	0.0%	14.7%	14.3%
YUV, M, 1440p	0.9%	1.0%	0.0%	1.6%	1.4%
YUV, M, 1080p	0.7%	2.7%	0.0%	3.0%	2.6%
YUV, A, 720p	0.0%	0.0%	0.0%	0.0%	0.0%
Average	9.5%	12.9%	0.2%	15.5%	14.6%
Encode Time[%]	75%	140%	102%	90%	77%
Decode Time[%]	99%	102%	96%	101%	106%

different $bit\ saving - complexity$ points are provided in Fig. 13. They are

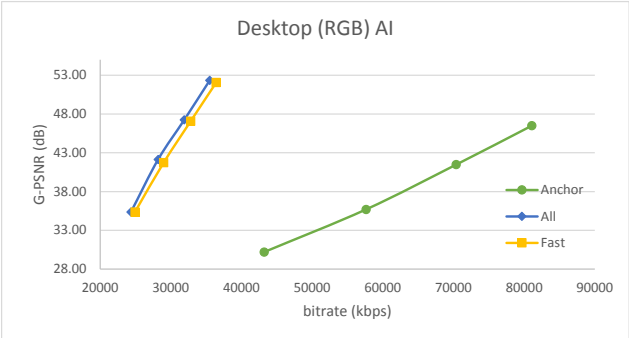
- Point 1: The full coding scheme described above.
- Point 2: The fast encoding solution described above.
- Point 3: Point 2 with the following modifications. Disable AMP (Asymmetrical Motion Partition) in the RDO process. Disable lossy search for Intra BC. Disable adaptive color space coding. Adjust the same encoding parameters used in the fast encoding algorithms for lossless coding. The number of search candidates for dictionary mode is set to 100 and if the number of different colors is more than 50, we skip the RDO process for dictionary mode.
- Point 4: Point 3 with the following modification. For normal intra mode, we only perform RDO on the direction leading to the smallest SATD (Sum of Absolute Transformed Differences) cost. We only check DM for chroma component in normal intra mode. We set the maximum transform depth to 1.
- Point 5: Point 4 with the disabling of non-2Nx2N inter partitions.

It should be noticed that Point 4 and Point 5 are identical for All Intra Coding.

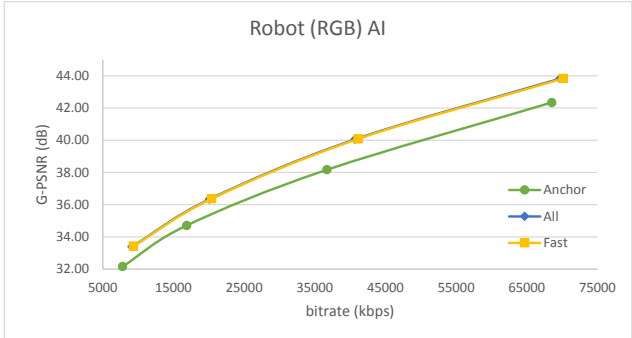
The figures show that different trade-offs could be achieved by adjusting encoding parameters. Compared with the anchor, large bit savings (more than 10%) may also be achieved even if the encoding complexity is significantly reduced (less than half of the original encoding time).

VIII. Conclusions

This paper has introduced a hybrid coding scheme for screen content coding. Considering the characteristics of screen con-

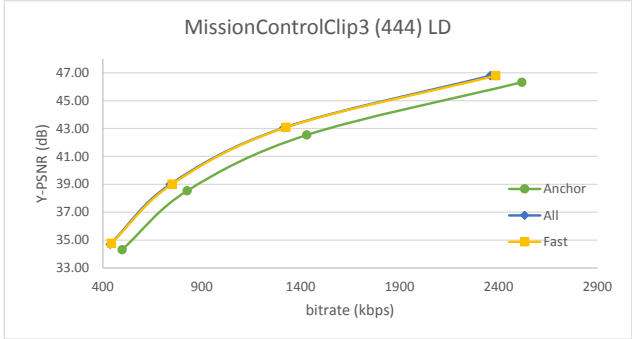


(a) Desktop, RGB, All Intra



(b) Robot, RGB, All Intra

(c) WebBrowsing, 444, Random Access



(d) MissionControlClip3, 444, Low Delay

Fig. 12: Example R-D Curves

tent, new coding tools, such as hash-based large scale block matching, dictionary mode, adaptive color space coding, etc., are designed. The experimental results show that the proposed coding scheme can significantly improve the coding efficiency for screen content while reducing complexity.

ACKNOWLEDGMENT

The authors would like to thank Dr. Feng Wu, Dr. Gary J. Sullivan and Dr. Xun Guo for their help and discussions during the development of the proposed coding scheme.

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] T. Wiegand, G. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard ," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, pp. 560 –576, Jul. 2003.

[3] J. Ohm, G. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards - including high efficiency video coding (hevc)," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 22, pp. 1669–1684, Dec 2012.

[4] G. Sullivan, J. Boyce, Y. Chen, J.-R. Ohm, C. Segall, and A. Vetro, "Standardized extensions of high efficiency video coding (hevc)," *Selected Topics in Signal Processing, IEEE Journal of*, vol. 7, pp. 1001–1016, Dec 2013.

[5] 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.

[6] C. Lan, J. Xu, G. J. Sullivan, and F. Wu, "Intra transform skipping." Document JCTVC-I0408, Geneva, CH, Apr. - May 2012.

[7] X. Peng, C. Lan, J. Xu, and G. J. Sullivan, "Inter transform skipping." Document JCTVC-J0237, Stockholm, SE, Jul. 2012.

[8] X. Peng, J. Xu, B. Li, L. Guo, J. Sole, and M. Karczewicz, "Non-RCE2: Transform skip on large TUs." Document JCTVC-N0288, Vienna, AT, Jul. - Aug. 2013.

[9] M. Budagavi and K. D.-K., "AHG8: Video coding using Intra motion compensation." Document JCTVC-M0350, Incheon, KR, Apr. 2013.

[10] B. Li, J. Xu, F. Wu, X. Guo, and G. J. Sullivan, "Description of screen content coding technology proposal by Microsoft." Document JCTVC-Q0035, Valencia, ES, Mar. - Apr. 2014.

[11] R. Joshi, R. Cohen, and H. Yu, "BoG report on summary of objective performance and tools for SCC CfP responses." Document JCTVC-Q0239, Valencia, ES, Mar. - Apr. 2014.

[12] G. Sullivan and T. Wiegand, "Rate-distortion optimization for video compression," *Signal Processing Magazine, IEEE*, vol. 15, pp. 74 –90, Nov. 1998.

[13] T. Lin, S. Wang, Z. P., and Z. K., "AHG7: Full-chroma (YUV444) dictionary+hybrid dual-coder extension of HEVC." Document JCTVC-K0133, Shanghai, CN, Oct. 2012.

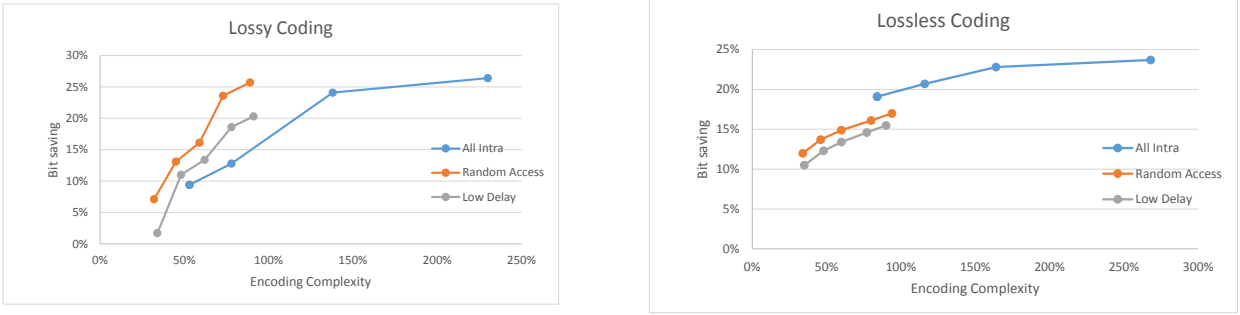


Fig. 13: Rate-Distortion-Complexity Trade-offs

[14] P. Zhang, T. Lin, X. Chen, K. Zhou, and X. Jin, "AHG7: HM software implementation and source code for JCTVC-K0133." Document JCTVC-K0134, Shanghai, CN, Oct. 2012.

[15] 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, pp. 173–185, Jan 2013.

[16] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *Information Theory, IEEE Transactions on*, vol. 23, pp. 337–343, May 1977.

[17] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *Information Theory, IEEE Transactions on*, vol. 24, pp. 530–536, Sep 1978.

[18] W.-S. Kim and T. Nguyen, "RCE1: Summary Report of HEVC Range Extensions Core Experiment 1 on Inter-Component Decorrelation Methods." Document JCTVC-O0035, Geneva, CH, Oct. - Nov. 2013.

[19] C. Rosewarne, K. Sharman, and F. D., "Common test conditions and software reference configurations for HEVC range extensions." Document JCTVC-P1006, San José, CA, USA, Jan. 2014.

[20] H. S. Malvar, G. J. Sullivan, and S. Srinivasan, "Lifting-based reversible color transformations for image compression," *Proc. SPIE*, vol. 7073, pp. 707307–707307–10, 2008.

[21] X. Guo, Y. Lu, and S. Li, "RCE4: Test1. Major-color-based screen content coding." Document JCTVC-P0108, San José, CA, USA, Jan. 2014.

[22] L. Lai, S. Liu, T.-D. Chuang, Y.-W. Huang, and S. Lei, "Non-RCE4: Major color table (palette) sharing." Document JCTVC-P0153, San José, CA, USA, Jan. 2014.

[23] T.-D. Chuang, Y.-C. Sun, Y.-W. Chen, Y.-W. Huang, and S. Lei, "Non-RCE4: Four-neighbor major color index prediction." Document JCTVC-P0098, San José, CA, USA, Jan. 2014.

[24] C. Gisquet, G. Laroche, and O. P., "Non-RCE4: Transition copy mode for Palette mode." Document JCTVC-P0115, San José, CA, USA, Jan. 2014.

[25] Y.-C. Sun, T.-D. Chuang, Y.-W. Chen, Y.-W. Huang, and S. Lei, "Non-RCE4: A combination of the four-neighbor major color index prediction in JCTVC-P0098 and a simplified transition copy mode from JCTVC-P0115 on top of RCE4 Test1." Document JCTVC-P0249, San José, CA, USA, Jan. 2014.

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

[27] 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.

[28] B. Li, L. Li, J. Zhang, J. Xu, and H. Li, "Encoding with fixed Lagrangian multipliers." Document JCTVC-J0242, Stockholm, SE, Jul. 2012.