Advanced Screen Content Coding Using Color Table and Index Map

Zhan Ma, *Member, IEEE*, Wei Wang, Meng Xu, and Haoping Yu

***Abstract*— This paper presents an advanced screen content coding solution using color table and index map (cTIM) method. This cTIM scheme is implemented in the range extensions of high-efficiency video coding (HEVC-RExt) as an additional tool of intracoding to complement conventional spatial angular prediction to better exploit the screen content redundancy. For each coding unit, a number of major colors will be selected to form the color table, then the original pixel block is translated to the corresponding index map. A 1D or hybrid 1D/2D string match scheme is introduced to derive matched pairs of index map for better compression. Leveraging the color distribution similarity between neighboring image blocks, color table merge is developed to carry it implicitly. For those blocks that color table has to be signaled explicitly, intertable color sharing and intratable color differential predictive coding are applied to reduce the signaling overhead. Extensive experiments have been performed and they have demonstrated the significant coding efficiency improvement over conventional HEVC-RExt, resulting in 26%, 18%, and 15% bit rate reduction at lossless case and 23%, 19%, and 13% Bjontegaard Delta-rate improvement at lossy scenario of typical screen content with text and graphics, for respective all intra, random access, and low-delay using B picture encoder settings. Detailed performance study and complexity analysis (as well as the comparison with other algorithms) have been included as well to evidence the efficiency of proposed algorithm.**

***Index Terms*— Color table, Index map, Screen content, HEVC.**

I. Introduction

A. Motivation and Attempts

RECENT networked applications, such as cloud computing, shared screen collaboration, virtual desktop interface, etc., have drawn more and more attentions in practice. Nowadays instead of providing everyone a powerful computing device (such as a personal computer), an emerging trend is to let people connect to the super computer (locally or globally) via their light-weighted ultra book, or even a touch screen display to manage and process the daily work. Such a solution enables the pervasive computing as long as one has stable network access. It also provides more secure workplace since it is no longer needed to distribute information to hundreds and thousands of individual clients but caching all of them at super computers (or data center). This framework could be realized in different ways, either proprietary or standardized schemes. This paper will focus on the technology to provide standardized resolution for such service enabling.

Intuitively, it could be seen as the pipeline of compressing screens at server side and interacting with remotely connected clients. This introduces new opportunities and challenges on how to compress high-definition computer generated screen content efficiently for limited network bandwidth. This is especially true for connecting the servers from long distance access through the Internet. Apparently, we can directly reuse the video compression technology, such as the state-of-the-art HEVC or HEVC-RExt [1], [2], to fulfill the screen compression purpose. However, screen content does have its distinct characteristics compared with the natural camera-captured video content. Therefore, it might suffer from the inferior compression performance if we use the existing video coding standards without modification, since these standards are mainly developed with the dedication to the camera-captured natural content.

More specifically, a typical computer screen picture mixes discontinuous-tone content such as text, icon, and graphics, and continuous-tone content such as video sequences and images with natural content. As also revealed in [3], continuous-tone and discontinuous-tone content have quite different statistical distributions. For instance, pixels in continuous-tone content evolve smaller intensity changes among local neighbors while neighboring pixel intensity could vary unexpectedly for discontinuous-tone samples. Furthermore, discontinuous-tone content possibly contains less distinct colors than the rich color distribution in continuous-tone content. Here, we refer pixel "color" to the pixel intensity for convenience. On the other hand, local samples of continuous-tone content typically present more complicate textures and orientations compared with the discontinuous-tone content. For instance, HEVC employs up-to 35 modes for spatial intra prediction to encode the natural content [1]. It might be good enough to have few angular modes (such as horizontal, vertical) since screen content typically contains well structured local orientation.

Moreover, chroma component sub-sampled contents, such as popular YCbCr 4:2:0, have been widely used for camera-captured natural videos, where the intuition is that YCbCr 4:4:4 is not really required due to the insensitivity impact of chroma components on human visual system. However, chroma sub-sampling would introduce noticeable

Manuscript received May 5, 2014; revised July 31, 2014; accepted August 1, 2014. Date of publication August 12, 2014; date of current version September 5, 2014. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Anthony Vetro.

Z. Ma is with the Nanjing University, Nanjing 210093, China (e-mail: zhan.ma@outlook.com).

W. Wang, M. Xu, and H. Yu are with Futurewei Technologies, Santa Clara, CA 95050 USA (e-mail: rickwang101@gmail.com; mxu02@students.poly.edu; haoping.yu@huawei.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIP.2014.2346995

visual artifacts for the discontinuous-tone region in screen content [4]. Hence, full chroma resolution is normally used for screen content representation, for instance, YCbCr 4:4:4 or RGB 4:4:4.[1] Full chroma resolution content compression would result in more bit rate consumption compared with the conventional chroma sub-sampled natural video. It also motivates the development of efficient tools to leverage the characteristics of screen content which are not exploited in natural camera-captured videos to compress the screen content more efficiently, even at full chroma resolution. Given that majority displays and screens are 8-bit resolution for each color component, we will emphasize this work considering 8-bit 4:4:4 screen content only.

To well understand the natural statistics of the screen content and to develop new technology for its efficient compression, in addition to the early efforts from both academia and industry [3]–[5], ad-hoc study group has been organized within the HEVC standardization committee to investigate the promising tools. Standard committee has issued a joint Call for Proposal (CfP) with the focus on the high efficiency screen content coding under the current HEVC framework[2] [6] in January 2014, and evaluated seven technical responses in April 2014, with the conclusion to launch the screen content extension development officially. These CfP responses share three common tools which provide the substantial gains, including intra block copy (IntraBC) [7]–[9] with limited or extended full frame search range, dictionary coding [10]–[13], and color table/palette coding [14]–[17].

B. Our Contribution

In this paper, we introduce an advanced color Table and Index Map coding (cTIM) scheme to compress screen content more efficiently. It is developed as a part of screen content coding CfP response submitted to the standard committee [16], [18]. Due to its promising coding performance improvement, it is selected into the Core Experiment to do extensive evaluation and test for the consideration of potential standard adoption. This cTIM tool is implemented on top of the HEVC-REext reference software and fully harmonized with HEVC recursive quad-tree structure from largest coding unit (LCU or CTU - coding tree-block unit) to the smallest coding unit. It is introduced into the HEVC-REext as an additional tool for intra coding to complement the conventional spatial angular prediction. Therefore, this tool is also useful for image or single frame processing [1].

For each coding unit (CU), we first derive the color table by ordering the pixels[3] (or colors as defined for the pixel intensity) according to its frequency of occurrence; Few frequent colors will be selected to form the color table while original pixels are then converted to the best matched index value; Residuals between original pixels and corresponding index-mapped colors are encoded using the HEVC-REext residual engine without modification; A 1D or hybrid 1D/2D string search is applied to the index map to generate matched pairs for compact representation and efficient compression. It has been observed in our experiments that neighboring CU might share the similar color distribution, hence, color table merge is included to improve the performance. For CUs that are not using color merge, inter-table color sharing and intra-table color differential predictive coding (DPCM) are employed to reduce the overhead. This cTIM scheme belongs to the color palette coding in general, but string search has been realized for index map coding with better compression efficiency than conventional run-length coding, line copy, adaptive index prediction using causal neighbors, etc. [17].

All screen content videos selected by the standard committee are used to evaluate the performance of proposed solution in comparison to the default settings of HEVC-REext 6.0 reference model [6], [19]. As demonstrated by the extensive simulations, cTIM shows 26%, 18%, 15% bit rate reduction at lossless case and 23%, 19%, 13% Bjontegaard Delta (BD)-Rate [20] improvement at lossy scenario of typical screen content with text and graphics, for respective All Intra (AI), Random Access (RA), and Low-delay using B picture (LB) encoder settings, respectively. Detailed performance comparison with other algorithms and complexity analysis have been provided to further evidence the efficiency of our proposed algorithms.

Even though color table or palette algorithm for image coding was studied decades ago [21]–[23], we have revisited the problem, introduced the color table and index map coding in the state-of-the-art HEVC framework and made several contributions to improve performance for screen compression:

- Color table and index map processing is decoupled to handle both lossy and lossless scenarios, and fully harmonized with the HEVC recursive quad-tree structure;
- A 1D or hybrid 1D/2D string search is introduced for index map processing to leverage the efficiency in string search with its advantages proven in well-known algorithms such as LZMA [3], [13], [24];
- Color table merge is developed to signal the table implicitly; For those tables that have to be carried explicitly, inter-table color sharing and intra-table color DPCM are implemented to reduce the bit rate overhead.
- Color table and index map coding are carefully designed with the consideration of hardware implementation cost, especially for the additional on-chip memory space and memory bandwidth requirement.

The rest of this paper is organized as follows: A very short review of HEVC is presented in Section II; Section III then briefs the technologies for screen content compression that are mainly developed and discussed in the HEVC standard committee followed by details of proposed cTIM in Section IV, V and VI. Experiments are carried out and discussed extensively in Section VII, and concluding remarks are drawn in Section VIII.

II. A Glance of High-Efficiency Video Coding

HEVC is the latest video coding standard developed under the efforts of Joint Collaborative Team on Video

[1]For simplification, we will use RGB 4:4:4 in the paper to exemplify the ideas. YCbCr 4:4:4 follows the similar processing manner.

[2]HEVC extensions are built upon the HEVC version 1 framework. We will use "HEVC framework" for simplicity in this paper.

[3]Note that here we use "pixel" or "color" to represent the triplet of its three color components.

Coding (JCT-VC), which demonstrates the 50% bit rate reduction compared with the well-known H.264/AVC [25], [26] at same perceptual quality, and promises the huge potential for massive market adoption to replace existing H.264/AVC or MPEG-2 products. It still falls into the same hybrid motion compensation and transform coding framework as its predecessors, but with quite noticeable improvements in many aspects to improve the coding efficiency and reduce the implementation complexity (especially for high throughput parallel processing) [27], [28].

Macroblock in the previous standards is extended in the HEVC to recursive quad-tree based *Coding Unit* (CU), *Prediction Unit* (PU) and *Transform Unit* (TU) [29]. Each CU can be recursively split into four sub-CUs and at each CU (or sub-CU) level, it can be further split into one or multiple PUs. HEVC also adopts recursive TUs for residual coding. Besides, larger block transforms, fine-grain spatial intra prediction, high precision interpolation filter, sample adaptive offset, etc., are introduced to improve the coding efficiency.

After the HEVC version 1, several extensions are under development to enable more functionalities on top of the HEVC framework, such as scalability and range extensions [2]. HEVC based screen content coding extension has been officially launched in April 2014 to study cost-efficient tools for standard adoption, and is expected to be finalized in October 2015 [30]. This paper will describe the techniques developed under HEVC framework for screen content coding.

III. Screen Content Coding: A Brief Review

There are three major technologies developed for screen content coding. One category is the intra block copy [7]–[9], [31], [32], the second one is string search based dictionary coding [3], [9]–[13], [24], [33]–[35] and the third one is the palette or color table coding [14]–[16], [18].

A. Intra Block Copy

Intra block copy (or intra motion compensation mentioned in the first place) was studied a decade ago [36]. Recently, it was brought up again and introduced into HEVC-RExt [7], [8] to enable inter-alike motion estimation and compensation technology using fixed block size for better coding efficiency. Instead of searching the reference in previously (temporally) reconstructed frame, it searches the reconstructed region in current frame and carries the block vector and compensation residual to the decoder. This technology does not show impressive performance gains for camera-captured content but significant gains for screen content. Various refinements, such as 1D and 2D block search, vector coding, padding, rectangle PU [31], [32] and even full frame search [9], are further investigated to improve the performance but the principle behind stays unchanged (i.e., block search and compensation).

B. String Search Based Dictionary Coding

String based dictionary coding for screen content (DictSCC) has been extensively studied over recent years [3], [10]–[13], [24], [33]–[35]. DictSCC sends the matched position offset (i.e., current position minus matched position) and length of a group of matched pixels, where the decoder simply copies from the matched location for reconstruction. If a pixel is not associated with any matched points, it will be encoded independently by itself, or predictively using the corresponding color component of its previously reconstructed pixel. DictSCC noticeably differs from intra mode in H.264/AVC or HEVC. It could be treated as the inter mode, i.e., matched position offset can be translated as the motion vector displacement. However, matched length gives more flexibility of the reference 1D or 2D string without fixed shape constraint.

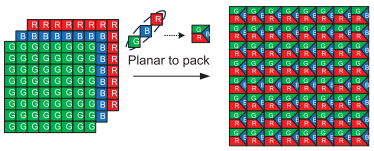


Fig. 1. Interleaved planar color components in pack fashion for coding unit color table and index derivation. GBR is used as an example. YCbCr will follow the samme fashion for processing.

Please note that HEVC and its predecessors normally process each color component (i.e., Y, Cb, Cr or R, G, B) sequentially for every block. Oppositely, DictSCC takes three color channels, and then converts to interleaved pattern pixel-by-pixel as shown in Fig. 1. The encoder then searches in sequential order to obtain pairs of matched position offset and length.

C. Color Table/Palette Coding

Color table/palette method was studied almost two decades ago [21]. It was revisited and found to be another attractive technology for screen content coding [14]–[18], [37], given that non-camera captured content typically contains a limited number of distinct colors, rather than the continuous color tone in natural videos. Neighboring colors in screen content generally have larger difference in terms of the intensity, hence conventional spatial intra prediction could not compress efficiently (since the residual is still quite large). By applying the color mapping using derived table or palette, original image block will be converted to a block of indices with reduced dynamic range, which will be easier for prediction and compression. Many attempts have been made to improve the performance of color table coding, including color table differential coding, index map coding using run-length code, line based direct copy, etc. It is worth to mention that *escape color mechanism* is utilized in [17] and relevant technical solutions to categorize those colors that have large intensity difference with the colors in the table. These escape colors are then represented by the most significant bits after quantization while other colors are indicated by the corresponding indices. With such a method, HEVC residual coding is completely bypassed.

D. Our cTIM Solution

Leveraging the advantages from both palette coding and string search, our cTIM scheme is introduced into the current

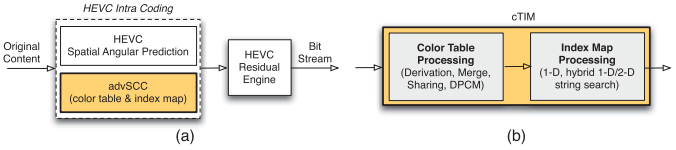


Fig. 2. Proposed algorithm: (a) cTIM as an additional intra tool; (b) Two step pipeline of cTIM.

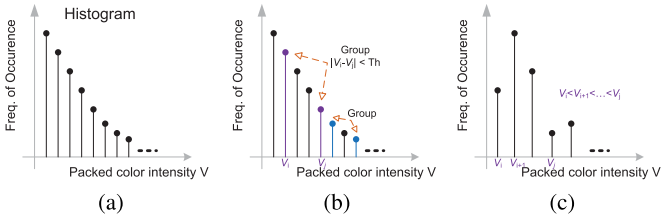


Fig. 3. Illustration of color table derivation: (a) histogram collection according to the frequency of occurrence in descending order, x-axis is dis-ordered packed color intensity V; (b) color grouping if their intensity difference is less than pre-defined threshold Th; (c) Re-order colors according to the intensity of packed pixel V in ascending order.

HEVC framework and integrated with HEVC-REXt software to efficiently exploit the screen content redundancy. It is carried out as an additional intra mode as shown in Fig. 2a. Therefore, our cTIM could be used for image or single frame coding in addition to the video sequence compression. Overall, cTIM contains two major steps in its processing pipeline, i.e., color table processing and index map compression, as illustrated in Fig. 2b. Residuals between original pixels and index-mapped colors are encoded using the HEVC residual engine without modification [38], [39]. Respective details will be shown as follows.

IV. Color Table Processing

To encode the 4:4:4 full chroma sampling content using the range extensions of HEVC or H.264/AVC, it typically processes component by component at block level, i.e., in the sequence of Y, Cb, Cr or G, B, R. Instead, we propose to interleave the color components for each pixel to *packed* fashion as exemplified in Fig. 1 for screen content coding. Note that such interleaving pattern is also applied in string search based dictionary coding [3], [9], [12], [13]. This packed coding unit (pCU) is then used to derive the color table and corresponding index map. Since we apply the algorithms upon the pCU, pixel V itself includes three 8-bit packed components (i.e., $V = \mathrm{G} \ll 16 + \mathrm{B} \ll 8 + \mathrm{R}$ or $V = \mathrm{Y} \ll 16 + \mathrm{Cb} \ll 8 + \mathrm{Cr}$). Unless pointed out otherwise, we will use "pixel color" or "pixel intensity", "pixel" or "color" interchangablely to represent a full resolution pixel with all components included, defined as $\mathbf{I}$ with $\mathbf{I}(0)$ for G or Y, $\mathbf{I}(1)$ for B or Cb and $\mathbf{I}(2)$ for R or Cr component of current pixel, respectively. Each pixel or color is associated with a corresponding $p_{\mathbf{I}}$ for its frequency of occurrence in this pCU. Please also note that u used in $\mathbf{I}_u$ indicates the position offset in a block.

A. Color Table Derivation

Color quantization has been studied for many years [40]. Most existing algorithms developed for video coding mainly focus on camera-captured content, thus not suitable for screen content coding directly. Given that typical screen content has limited colors, we present a simple yet effective method to derive the dominant colors.

For each pCU, we first collect the pixel histogram according to the frequency of the occurrence in descending order as shown in Fig. 3a to form the ordered color array, and then group the colors together if their intensity difference is less than a pre-defined error allowance Th as exemplified in Fig. 3b. Th = 9 is used in this paper. For two colors to be clustered together, each component difference should be less than this allowance, i.e., $|\mathbf{I}_u(k) - \mathbf{I}_v(k)| < \mathrm{Th}, k = 0, 1, 2$ with u and v indicating the color location in ordered array. After each grouping, color array will be updated with the occurrence frequency as the sum of the occurrence frequency of grouped colors.

Obviously, grouping two different colors together will produce residuals. To reduce the energy of residual, we apply a simple criteria to group colors where one color $\mathbf{I}_u$ with smaller frequency of occurrence is grouped to another color $\mathbf{I}_v$ with larger frequency of occurrence, i.e., $p_{\mathbf{I}_u} < p_{\mathbf{I}_v}$, yielding less grouping error. As aforementioned, error residuals will be encoded using the HEVC-REXt residual engine without modification.

After performing the grouping, N colors will be selected to form the color table. Here, we use 128 as the maximum number of colors allowed for all CU size. If actual number of colors $N \leq 128$, the exact N will be used to signal the color table size; If $N > 128$, only first 128 colors will be chosen with rest of colors are mapped to related colors in the table according to the least sum of absolute error criteria.

Re-ordering is then applied to selected colors to convert frequency descending order to intensity ascending order as illustrated in Fig. 3c. It is designed for better compression purpose as will be explained in color table encoding. Note that since the color table and the index map processing is de-coupled, color table reordering have negligible impact on the index map processing.

B. Color Table Signaling

There are two ways to carry the color table in the bitstream to the decoder side. One is the *explicit* signaling, and the other is the *implicit* signaling. Implicit signaling is also called *color table merge* using derived color table from available neighbors, while explicit signaling is encoding each color in table one by one. For the explicit signaling, we also introduce inter-table color sharing and intra-table color DPCM to further reduce the bit rate overhead. For simplicity, we use $\mathbb{C}$ to represent the color table with total N colors. $\mathbb{C}_n$ is the n-th color element $\mathbf{I}$ in the table with $n \in [0, N-1]$.

1) Color Table Merge: This is motivated by the observation that neighboring blocks could share the similar colors. Especially for the screen content, it normally has a large amount of blocks (or regions) with the similar color distribution. To further exploit the local correlation, we also introduce

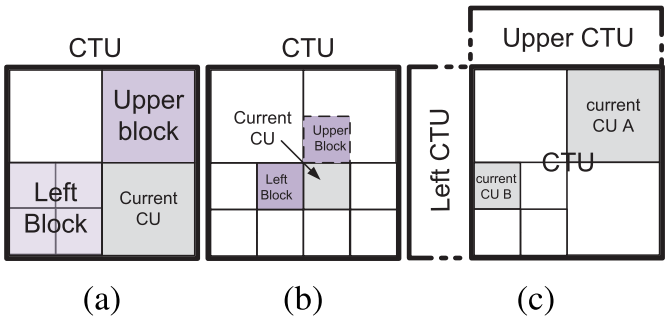


Fig. 4. Color table merge using available neighbor blocks: (a) current CU size > neighbor CU size; (b) current CU size < neighbor CU size; (c) neighbor CU outside current CTU.

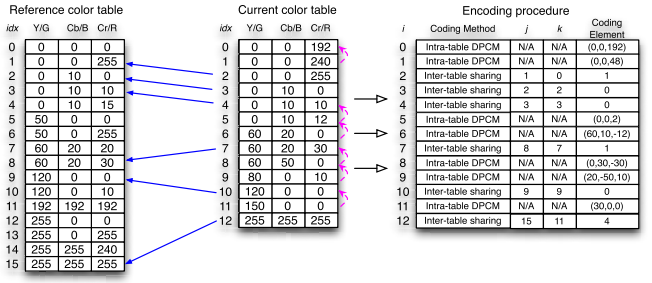


Fig. 5. Example of inter-table color sharing and intra-table color DPCM: i is the index in current color table, j is matched index in reference table, and k is the prediction of matched index j in reference table.

the possibility to merge the colors from either left or upper neighbors.

As shown in Fig. 4, available reconstructed neighbor blocks of current CU are used to derive the color table for merge. Unlike motion vector merge process in HEVC where current CU could have different block size (or split depth in quad-tree structure), we propose to use the neighbors with the same block size for color table merge. For example, instead of using left 16×16 CU which is quarter size of current 32×32 CU (shown in Fig. 4a), we use the 32×32 "left block" to derive the color table. Similarly, upper CU has the larger size than current CU, but we will use pixels in the 16×16 "upper block" for color merge shown in Fig. 4b.

So far, we have shown the available neighbors and current CU are all inside current CTU where color tables could be derived at both encoder and decoder on-the-fly. Besides, there are the cases that current CU lies at the boundary of current CTU, such as current CU A and current CU B in Fig. 4c. Rather than fetching the reconstructed data block from upper and left CU (probably from off-chip memory), we propose to apply the color table of upper and left CTU directly. Therefore, we only need to generate the color table for each CTU after completing its encoding or decoding process and cache the table itself for next CTU processing.

In total, we only need two binary bits (or bins) to carry the information to decoder side to indicate that whether current CU uses the color merge mode and which merge direction is applied, i.e., either left or upper merge.

2) Color Sharing and DPCM: We see the color table overhead could be reduced largely by introducing the color merge from available reconstructed neighbors. However, neighbor blocks may not share similar color distribution (for majority colors) in reality. Hence, it is inevitable that color table should be carried explicitly for some CUs.

Even though color merge mode is not used for current CU, we have observed that there are still some identical colors between the color table $\mathbb{C}$ of current CU and the reference table $\hat{\mathbb{C}}$ of neighboring block. Thus, we can share these identical colors by signaling the relative index in the reference table to reduce the overhead. Relative index is calculated between the exact-matched index j and the predicted index k of matched index j in reference table. k is obtained when $\hat{\mathbb{C}}_k(0) \geq \mathbb{C}_{i-1}(0)$ is first satisfied. Here, $\hat{\mathbb{C}}_k(0)$ and $\mathbb{C}_{i-1}(0)$ are the first component (i.e., G or Y) of k-th element in the reference table and $(i-1)$-th element in the current table, respectively. We assume that i-th color is the one for encoding and $(i-1)$-th element is previously coded color. This is so-called *inter-table color sharing*.

Besides, if the color does not exist in the reference table (which is fairly common in practice that new colors are gradually introduced from one spatial region to another), we further develop the differential prediction between successive colors, which is defined as the *intra-table color DPCM*. As discussed in Section IV-A, colors are placed in ascending order according to the intensity of packed pixel V. Difference between successive colors would result in much smaller dynamic range compared with the original 8-bit (i.e., 0-255) component. Ideally, less bins are required to encode low dynamic range predictive difference than high dynamic range original component level.

Fig. 5 shows the step-by-step procedure for color table sharing and DPCM. Reference table (far-left) is derived from the left CTU. Starting from the first entry of the current color table, i.e., $(\text{idx}, \text{G}, \text{B}, \text{R}) = (0, 0, 0, 192)$, it does not have any exactly matched item in the reference table and it does not have previous reference in the same table as the first item, therefore it is binarized using 8-bit fixed-length code and encoded using bypass mode directly. For the 2nd entry, it does not have a matched item in the reference table either. Given that we have coded the first entry already, only predictive difference is carried in the bitstream, i.e., $(0-0, 0-0, 240-192) = (0, 0, 48)$. For the 3rd entry of the current table, it finds the exact match in the reference table when $j = 1$, meanwhile the predicted index k is 0 when first meeting $\hat{\mathbb{C}}_{k=0}(0) = \mathbb{C}_{i-1=1}(0)$, resulting in $j - k = 1$ to be carried in bitstream. The similar encoding procedure is applied until completing the last entry of the current table (i.e., idx = 12 in this example). Since $\mathbb{C}_{12}$ of the current table already reaches the last element of the reference table for sharing, if there are more elements than the total thirteen ones displayed, all the rest elements will be coded using color DPCM.

V. Color Index Map Processing

Index map is generated using the color table for each packed pixel in current CU. For those colors that are exactly placed in the color table, corresponding index will be assigned directly;

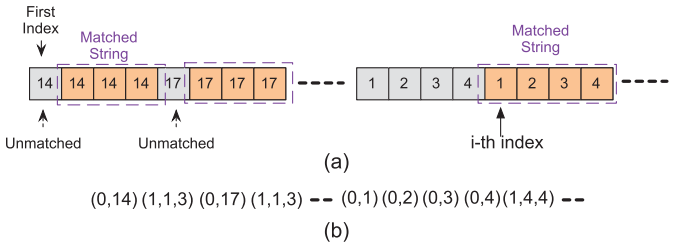


Fig. 6. Illustrative example of (a) 1D string search to derive matched pairs of encoding; (b) encoded matched pairs (1, dist, len) or unmatched pairs (0, idx);

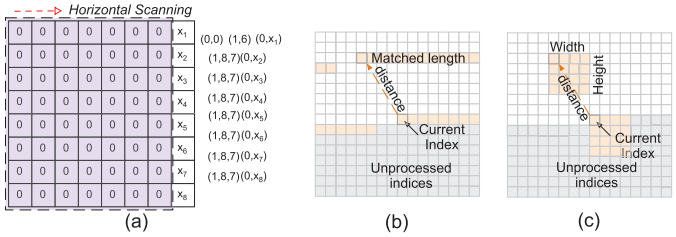


Fig. 7. Examples of (a) a 8 × 8 index map and its encoded pairs after 1D string search: $x_i, i \in [1, 8]$ are other index values different from 0; (b) 1D string match distance and length; (c) 2D string match distance and block width and height.

For those colors that are not included in the color table (due to the maximum color number allowance and error allowance in lossy scenario), we have applied the least error criteria to find the best match. A 1D and hybrid 1D/2D string search are developed to derive matched or unmatched pairs for high-efficiency index map compression.

A. 1D String Search

Intuitively, we will convert each 2D index map to a 1D string from the first position (top-left) to the last one (bottom-right) in a raster scan order to derive matched/unmatched pairs.

1) Adaptive Index Map Scanning: Unidirectional search is indeed sufficient for native 1D string (such as text string). However, this 1D string search is performed on the pseudo 1D pixel string which is converted from a 2D image block through a pre-defined scanning pattern. Only vertical scanning is applied in [3] and [10] to process the pixels. As revealed in numerous video coding works, video block contains noticeable angular patterns. Moreover, typical screen element contains large proportion of both horizontal and vertical patterns (i.e., bars, edges, etc.). Therefore, we propose to use both horizontal and vertical scanning for 1D string search, where the optimal scanning direction is determined using a simple bit consumption estimation by calculating the logarithmic values of matched distance and length. On the other hand, extending horizontal and vertical directions with more angular patterns would probably improve the coding efficiency, however, we have observed that horizontal and vertical patterns dominate the typical screen content and more angular directions will certainly require more computing resource.

2) 1D String Match Paris: For each scanning pattern, A 1D string search is applied from the first index (the most top left) to the last index (the most bottom right) to derive the matched pairs. Previously processed indices will be used as the reference in the successive index search. If current index could not find its exact match from previously processed indices, it is encoded as the unmatched pair by sending a flag (signaling no-match) and the index value itself, i.e., (MatchOrNot = 0, idx) or (0, idx) for simplicity.[4] If current index finds its match from the reference buffer, the search continues for the next index, and so on so forth. The matched pairs are encoded by sending a flag (signaling matched string), the distance (offset between the current position and reference position) and the length of successive matched indices, i.e., (MatchOrNot = 1, dist, len) or (1, dist, len) for simplicity. If there are multiple matches found for the current index, the pair providing the best Rate-Distortion performance is chosen. To reduce the complexity, the current implementation uses a heuristic algorithm to select the pair with the longest matched length. Please note that we have constrained such 1D string search within current CU for our design. As will be demonstrated in later experiments, 1D string search based index map coding already provides decent performance improvement.

To well understand the 1D string search in the paper, Fig. 6 gives a piece of 1D segment after performing the horizontal scanning of a 2D index map. On top of this 1D color index vector, 1D string match is applied as follows:

- For the first position of each index map, say 14 as shown in Fig. 6. Since there is no buffered reference yet, this very first index is treated as an "unmatched pair", where we give the −1 and 1 to its corresponding distance and length, noted as (dist, len) = (−1, 1) and signal the index 14 itself. MatchOrNot flag is inferred as 0.
- For the second index, again another "14", we have the first index coded as reference, therefore the dist = 1. Because it continues "14" at 3rd and 4th positions, the length should be 3, i.e., len = 3 with the corresponding codeword (MatchOrNot, dist, len) = (1, 1, 3);
- For the "17" first appeared, it is coded as another "unmatched pair" by signaling (MatchOrNot, idx) = (0, 17);
- Moving forward to the i-th position, we will have dist = 4 and len = 4 (i.e., (MatchOrNot, dist, len) = (1, 4, 4)) since we can find a completely matched string "1234" from previously processed indices.

B. Hybrid 1D/2D String Search

Fig. 7a shows an index map collected from the real test sequences, where encoded pairs after the 1D string search are exemplified as well. There is still redundancy among matched pairs for encoding from the second row to the bottom, i.e., except the last position of each row using the unmatched pair (0, x_i) $i \in [2, 8]$, all rows are encoded using matched pair (1, dist, len) = (1, 8, 7). Apparently, it will require less bits if we can signal the 8 × 7 zero block

[4]Please note that the first index is always an unmatched index in current design. Hence, we don't need to signal the match flag for it.

TABLE I
BINARIZATION AND CABAC MODEL OF SYNTAX ELEMENTS IN cTIM FOR 2N × 2N CU

Syntax Element	Description	Binarization	Model
cTabIdxFlag	cTIM mode flag	1-bin FLC	adaptive
cTableSize	color table size	8-bin FLC	bypass
cMergeFlag	color merge flag	1-bin FLC	adaptive
cMergeDir	color merge direction	1-bin FLC	adaptive
cShareflag	color sharing flag	1-bin FLC	adaptive
cRelativeIdx	$j-k$	unary code	bypass
cDynRgBit[3]	bits for color dynamic range	4-bin FLC	bypass
cDpcmValue	DPCM difference	d_i-bin FLC	bypass
cDpcmSign	DPCM sign value	1-bin FLC	bypass
MatchOrNot	match flag	1-bin FLC	adaptive
idx	unmatched index	m-bin FLC	bypass
dist	matched distance	$2n$-bin FLC	adaptive
len	matched length	$2n$-bin FLC	adaptive
2Dflag	2D search flag	1-bin FLC	adaptive
dist*	2D matched distance	x-bin FLC	adaptive
width	2D matched width	n-bin FLC	adaptive
height	2D matched height	n-bin FLC	adaptive

d_i = cDynRgBit[i], $i = 0, 1, 2$; $n = \log_2(2N)$;
$m = ceil(\log_2(\texttt{cTableSize}))$; $x = 2n + \log_2(w)$

using the 2D representation, i.e., (1, dist, width, height) rather than multiple 1D matched pairs. Such a case could be quite often if the content exhibits similar block patterns, and motivates the hybrid 1D/2D string search design to further improve the coding efficiency.

1) Hybrid 1D/2D Matched Pairs: Both 1D and 2D string searches are performed for the indices in current CU. For the 1D case, search engine records matched length and corresponding distance or location as shown in Fig. 7b. For the 2D case, search engine records matched block width and height and distance as well, as illustrated in Fig. 7c. The same as the 1D search, if the current index could not find its match from previously processed indices, it is encoded as the unmatched pair, i.e., (0, idx); If current index finds its match from previous coded samples, besides MatchOrNot = 1 flag, an additional 2D search enabling flag, i.e., 2DFlag, is used to indicate whether the match type is 1D string or 2D block. Whether to choose 1D string or 2D block depends on the size of 1D matched string and 2D matched block,[5] i.e., if len ≥ width×height, 1D string is selected by signaling (1, 2DFlag = 0, dist, len), otherwise, 2D block is chosen by sending (1, 2DFlag = 1, dist, width, height) to the decoder. In case that multiple 1D or 2D matches found for the current index, the one providing the largest size will be chosen and encoded into the stream.

2) Extended Reference Buffers: As discussed in previous sections, we have constrained the 1D string search within current CU without requiring additional on-chip memory cache (except for the color table storage). It targets the low-complexity profile of our cTIM algorithm. For hybrid 1D/2D string search, we extend the search range to include previously coded CTUs and CUs (in current CTU). Note that different search range settings will have various impact on both coding efficiency and complexity (such as computational resource and memory bandwidth). More detailed discussion regarding the trade-off between coding efficiency and complexity will be presented in next section. Also note that extended reference buffers are only used for 2D search, 1D search is still constrained within current CU to support the backward compatibility with pure 1D search scenario. It is also worth to point out that we have constrained the hybrid 1D/2D search for 64 × 64 or CTU size only and pure 1D search for all CU size in current design.

[5]Apparently, rate-distortion based 1D or 2D decision could be developed to improve the performance compared with current heuristic scheme. It is our future focus of study.

VI. SYNTAX ELEMENTS AND ENCODING

In this section, we briefly summarize the syntax elements as well as the (CABAC) entropy encoding schemes introduced in cTIM method as shown in Table I. Context coded binary cTabIdxFlag is used to indicate whether current CU is coded using cTIM method.

First, cTableSize is binarized using 8-bin (as of maximum 128 colors) fixed-length code (FLC) and encoded using bypass model. cMergeFlag and cMergeDir describe the color table merge option and merge direction respectively. Both of them are encoded using adaptive contexts. Without color merge, context coded binary cShareflag and cRelativeIdx signal the color entry sharing and relative index between reference and current color table. Note that cRelativeIdx is binarized using unary code and encoded using the bypass context. Moreover, cDynRgBit[i] ($i = 0, 1, 2$) captures the number of bits required for the dynamic range representation of i-th color component in the table. cDpcmValue and cDpcmSign represent the absolute value and sign of difference between successive colors respectively. All of them are binarized using FLC and encoded using bypass model. Note that bins number for cDynRgBit[i] ($i = 0, 1, 2$) binarization is bounded to 4 (as of signaling "8" for 8-bit video), while bins number for cDpcmValue is bounded by the value of cDynRgBit[i] for i-th component.

As for a 2N×2N CU, MatchOrNot describes whether current index is matched or not: For unmatched pair, index itself is binarized using 8-bit FLC and encoded with bypass model; For 1D matched pair, both dist and len are binarized using $\log_2$(2N×2N)-bin FLC. Each bin is coded using individual adaptive context model. Context coded 2Dflag signals whether the string match using a 2D block. If so, both width and height are binarized using $\log_2$(2N)-bin FLC and encoded using adaptive context for each bin. Note that for 2D case, because of the extended CTU buffer, dist is binarized using x-bin FLC where $x = \log_2(w \times 2N \times 2N)$ with w as the number of CTUs used for hybrid search.

VII. EXPERIMENTAL STUDY AND DISCUSSION

Experiments are carried out using the screen content sequences selected by the experts in Joint Collaborative Team on Video Coding group [6]. These sequences are chosen to represent popular and typical screen content application scenarios. For instance, one category is for the text and graphics with motion (TGM) representing the typical remote desktop applications, including "FlyGraphicsText" (FGT), "Desktop",

TABLE II

PERFORMANCE EVALUATION AND COMPARISON ON AI ENCODING CONFIGURATION FOR LOSSLESS SCREEN CONTENT CODING OF 1D, HYBRID I, HYBRID II, IBC I, IBC II, IBC FF, CP, CP-EXT, P2M, AND P2M-EXT AGAINST HEVC-REXT 6.0

		Bit Rate Reduction Against RExt 6.0 (percentile)									
		Proposed cTIM			Other Algorithms						
		1D	Hybrid I	Hybrid II	IBC I	IBC II	IBC FF	CP	CP-Ext	P2M	P2M-Ext
RGB	FGT	13.83%	15.62%	16.90%	7.57%	18.49%	21.84%	2.99%	4.58%	17.60%	29.43%
	Desktop	34.43%	44.52%	48.28%	15.25%	25.64%	35.71%	21.68%	21.84%	37.53%	47.42%
	Console	20.91%	27.05%	29.94%	4.27%	8.61%	10.06%	9.77%	10.16%	22.49%	25.72%
	WebB	22.89%	30.78%	33.28%	10.87%	18.74%	25.06%	15.04%	15.05%	23.03%	30.18%
	Map	26.87%	27.67%	28.08%	0.39%	2.42%	1.50%	17.75%	22.96%	19.77%	19.87%
	PGM	8.56%	9.07%	9.54%	2.82%	6.29%	7.51%	1.22%	1.42%	11.33%	16.17%
	SlideShow	1.38%	1.55%	1.58%	0.48%	0.70%	0.82%	0.16%	0.16%	0.59%	0.69%
	BbScreen	4.46%	5.15%	5.69%	2.58%	5.47%	7.02%	0.53%	0.72%	5.37%	8.89%
	MCClip2	5.00%	5.12%	5.28%	0.63%	1.67%	13.98%	1.42%	2.27%	2.06%	4.68%
	MCClip3	3.73%	4.37%	4.77%	2.35%	5.67%	10.01%	0.77%	0.90%	4.35%	8.24%
	SNM	4.87%	4.92%	4.94%	-0.01%	1.66%	1.07%	0.36%	0.42%	10.70%	14.26%
	Robot	0.00%	0.00%	0.00%	-0.01%	0.01%	0.01%	0.00%	0.00%	0.00%	0.00%
YCbCr 444	FGT	13.57%	15.75%	17.41%	7.58%	18.76%	22.01%	2.50%	4.04%	17.55%	28.11%
	Desktop	41.37%	51.28%	54.82%	16.37%	27.81%	38.23%	29.01%	29.45%	44.19%	52.61%
	Console	30.99%	36.77%	39.33%	5.15%	9.36%	10.94%	10.41%	15.84%	27.19%	29.18%
	WebB	30.30%	39.00%	41.69%	13.50%	23.06%	30.67%	22.18%	22.22%	31.51%	39.18%
	Map	24.88%	25.83%	26.31%	0.52%	2.58%	1.61%	14.34%	20.08%	16.47%	16.01%
	PGM	7.59%	8.35%	9.00%	2.81%	6.34%	7.52%	0.62%	0.73%	11.21%	14.95%
	SlideShow	1.48%	1.69%	1.73%	0.61%	0.82%	0.94%	0.14%	0.13%	0.51%	0.60%
	BbScreen	4.87%	5.68%	6.35%	3.02%	6.37%	8.11%	0.57%	0.82%	6.47%	9.88%
	MCClip2	0.80%	0.92%	1.13%	0.71%	1.97%	13.83%	0.12%	0.07%	1.41%	4.09%
	MCClip3	3.14%	3.58%	4.11%	2.39%	6.36%	10.78%	0.35%	0.41%	4.29%	8.55%
	SNM	2.68%	2.73%	2.76%	-0.05%	1.54%	1.05%	0.04%	0.04%	7.24%	8.28%
	Robot	0.07%	0.07%	0.07%	-0.01%	0.01%	0.00%	0.00%	0.00%	0.00%	0.00%
Average	TGM-G	18.41%	22.32%	23.94%	5.95%	11.56%	14.65%	9.80%	10.88%	18.91%	24.21%
	MIX-G	4.52%	4.89%	5.17%	1.39%	3.62%	8.02%	0.77%	1.08%	5.62%	9.02%
	AMT-G	0.00%	0.00%	0.00%	-0.01%	0.01%	0.01%	0.00%	0.00%	0.00%	0.00%
	TGM-Y	21.45%	25.52%	27.19%	6.65%	12.68%	15.99%	11.31%	13.21%	21.23%	25.81%
	MIX-Y	2.87%	3.23%	3.59%	1.52%	4.06%	8.44%	0.27%	0.33%	4.85%	7.70%
	AMT-Y	0.07%	0.07%	0.07%	-0.01%	0.01%	0.00%	0.00%	0.00%	0.00%	0.00%

Fig. 8. Sample frames of typical screen content test sequences.

"Console", "WebBrowsing" (WebB), "Map", "Programming" (PGM) and "SlideShow". Another category is the animation content (AMT), i.e., "Robot", standing for the applications such as cloud gaming. Yet another category is the mixed content (MIX) for the most common screen in our daily life that contains texts, graphics as well as the camera-captured video, such as "MotionControlClip2" (MCClip2), "MotionControlClip3" (MCClip3), "BasketballScreen" (BbScreen) and "SocialNetworkMap" (SNM). All sequences have both 8-bit RGB and YCbCr (or YUV) version with full chroma sampling resolution. For convenience, we note RGB format sequence of TGM category as TGM-G while YCbCr format as TGM-Y. The same abbreviation rule applies to other categories as well. Sample frames of all test sequences are presented in Fig. 8.

A. Performance Evaluation of Proposed cTIM

We have implemented the proposed cTIM on top of the HEVC-REXt 6.0 reference software [19]. Both lossless and lossy scenarios are evaluated with three popular encoder settings, i.e., All Intra (AI), Random Access (RA) and Low-delay with B picture (LB). Detailed coding efficiency improvement will be presented for each test sequence using AI configuration, given that cTIM is developed to explore the correlation within current frame, while categorized averaged performance data is shown for LB and RA cases. Bit rate reduction is used to show the gains for lossless encoding while BD-Rate improvement [20] is for the lossy scenario. Both are calculated against the anchor data produced by HEVC-REXt 6.0. Please note that *positive number of bit rate reduction and negative number of BD-Rate indicate the performance gain*.

All other parameters (mainly quantization parameters, intra smoothing option, transform quantization bypass option, intra frame period, etc.) follow the screen content coding CfP description [6].

We have performed simulations step by step to show incremental gains against the anchor, i.e.,

- 1D: current CU constrained 1D string search is enabled for index map coding;
- Hybrid I: Hybrid 1D/2D string searches are supported, and 2D search range is extended to include three left CTUs (i.e., four CTUs in total by including current CTU);
- Hybrid II[6]: In addition to the left three CTUs, upper four CTUs are used for 2D search (i.e., 2× 4 CTU window).

1) Results for TGM and MIX Content:

a) AI: Detailed experimental results (against anchor HEVC-REXt 6.0) for AI lossless and lossy encoding are shown Table II and Table III.

[6]For both Hybrid I and Hybrid II, 1D search is still limited in current CU.

TABLE III

PERFORMANCE EVALUATION AND COMPARISON ON ALL INTRA ENCODING CONFIGURATION FOR LOSSY SCREEN CONTENT CODING OF 1D, HYBRID I, HYBRID II, IBC I, IBC II, IBC FF, CP, CP-EXT, P2M, AND P2M-EXT AGAINST HEVC-REXT 6.0

		BD-Rate [20] Reduction Against RExt 6.0 (percentile)									
		Proposed cTIM			Other Algorithms						
		1D	Hybrid I	Hybrid II	IBC I	IBC II	IBC FF	CP	CP-Ext	P2M	P2M-Ext
RGB	FGT	-11.57%	-14.22%	-17.31%	-6.73%	-18.95%	-22.76%	-9.11%	-8.48%	-6.64%	-8.06%
	Desktop	-23.70%	-37.69%	-43.99%	-14.39%	-27.12%	-37.05%	-12.48%	-11.41%	-19.27%	-31.55%
	Console	-20.58%	-28.63%	-33.55%	-4.80%	-12.37%	-15.30%	-14.58%	-14.30%	-13.40%	-16.86%
	WebB	-23.82%	-45.62%	-53.17%	-16.07%	-28.86%	-41.23%	-11.68%	-11.40%	-18.27%	-37.57%
	Map	-7.51%	-8.72%	-9.45%	-1.40%	-5.07%	-4.33%	-6.22%	-6.03%	-4.34%	-5.62%
	PGM	-12.41%	-14.20%	-16.00%	-3.66%	-9.55%	-11.32%	-4.61%	-4.14%	-2.72%	-6.97%
	SlideShow	-3.65%	-4.47%	-4.74%	-3.19%	-4.61%	-5.17%	-4.00%	-3.99%	-1.10%	-1.79%
	BbScreen	-5.23%	-7.04%	-9.15%	-5.94%	-13.18%	-15.79%	-1.41%	-1.09%	-1.02%	-3.68%
	MCClip2	-2.02%	-2.52%	-3.72%	-2.20%	-7.55%	-10.65%	-0.90%	-0.81%	-0.69%	-1.44%
	MCClip3	-6.52%	-7.90%	-10.33%	-3.50%	-13.74%	-17.15%	-2.89%	-2.38%	-1.35%	-2.80%
	SNM	-0.77%	-0.78%	-0.79%	0.07%	-2.22%	-1.46%	-0.23%	-0.18%	-0.10%	-0.11%
	Robot	0.04%	0.04%	0.04%	0.00%	-0.11%	-0.06%	0.07%	0.07%	0.03%	0.04%
YCbCr 444	FGT	-8.00%	-9.27%	-11.27%	-6.00%	-17.01%	-20.55%	-6.98%	-6.90%	-5.36%	-6.55%
	Desktop	-19.88%	-32.22%	-39.34%	-13.85%	-26.56%	-35.95%	-11.88%	-11.63%	-13.76%	-24.52%
	Console	-19.75%	-28.45%	-34.48%	-6.15%	-14.74%	-18.47%	-14.48%	-14.55%	-11.15%	-15.06%
	WebB	-12.59%	-23.89%	-33.43%	-14.05%	-26.03%	-36.72%	-7.02%	-7.02%	-8.05%	-17.18%
	Map	-5.34%	-6.85%	-7.60%	-1.76%	-5.45%	-4.74%	-3.83%	-3.71%	-3.87%	-5.50%
	PGM	-6.92%	-8.50%	-10.91%	-3.73%	-9.98%	-11.45%	-2.56%	-2.43%	-1.11%	-2.04%
	SlideShow	-1.78%	-1.98%	-2.07%	-3.58%	-4.92%	-5.88%	-1.65%	-1.65%	-0.20%	-0.51%
	BbScreen	-4.88%	-7.40%	-10.68%	-6.43%	-14.35%	-17.22%	-1.05%	-0.59%	-0.01%	-0.92%
	MCClip2	-1.89%	-2.60%	-4.64%	-2.35%	-8.18%	-11.60%	-0.81%	-0.69%	-0.49%	-0.84%
	MCClip3	-6.09%	-8.21%	-11.85%	-3.55%	-14.49%	-18.26%	-2.58%	-2.21%	-0.77%	-1.28%
	SNM	0.37%	0.37%	0.36%	0.07%	-2.35%	-1.65%	0.05%	0.04%	0.01%	0.00%
	Robot	0.07%	0.07%	0.08%	-0.03%	-0.13%	-0.09%	0.08%	0.08%	0.06%	0.06%
Average	TGM-G	-14.75%	-21.94%	-25.46%	-7.18%	-15.22%	-19.59%	-8.95%	-8.54%	-9.39%	-15.49%
	MIX-G	-3.64%	-4.56%	-6.00%	-2.89%	-9.17%	-11.26%	-1.36%	-1.12%	-0.79%	-2.01%
	AMT-G	0.04%	0.04%	0.04%	0.00%	-0.11%	-0.06%	0.07%	0.07%	0.03%	0.04%
	TGM-Y	-10.61%	-15.88%	-19.87%	-7.02%	-14.96%	-19.11%	-6.91%	-6.84%	-6.21%	-10.19%
	MIX-Y	-3.12%	-4.46%	-6.70%	-3.07%	-9.84%	-12.18%	-1.10%	-0.86%	-0.32%	-0.76%
	AMT-Y	0.07%	0.07%	0.08%	-0.03%	-0.13%	-0.09%	0.08%	0.08%	0.06%	0.06%

As we can see, cTIM has shown impressive performance improvement for TGM content at both lossless and lossy scenarios, against the anchor. For TGM content, 1D cTIM already provides approximately 20% lossless bit rate reduction and 13% lossy BD-Rate improvement (averaged between TGM-G and TGM-Y). Hybrid I further improves 1D by another averaged ≈4% and 5% gains for respective lossless and lossy cases. Hybrid II expands the reference area (for 2D string search) by including both left and upper CTUs, averaged relative gain is still more than 2% over Hybrid I (both against HEVC-REXt 6.0). For YCbCr format "WebBrowsing" sequence, Hybrid II can bring up to almost 10% relative gain over Hybrid I for lossy AI configuration shown in Table III.

Noticeable gains have been observed for MIX content as well (but not as large as TGM content). For instance, 1D cTIM gives averaged ≈5% lossless bit rate reduction and ≈4% lossy BD-Rate improvement, but only marginal improvements have been observed by using Hybrid I and II over 1D method.

b) RA and LB: Because of the superior performance provided by the cTIM for TGM content AI encoding, even for the RA and LB coding configurations, it still gives about 10% lossless bit rate reduction (up to 14.3% for TGM-Y content encoded using RA setting) and 8% lossy BD-Rate improvement (up to 11.6% for TGM-G content using RA configuration) on average using 1D method, as briefed in Table IV. Another relative 3% gain can be further obtained by enlarging the search range from constrained CU of 1D to the inclusion of three more left CTUs of Hybrid I, while slightly larger than 2% gain is shown from Hybrid II over Hybrid I.

TABLE IV

PERFORMANCE EVALUATION ON RA AND LB CONFIGURATIONS OF 1D, HYBRID I AND II AGAINST HEVC-REXT 6.0

		RA			LB		
		1-D	Hybrid I	Hybrid II	1-D	Hybrid I	Hybrid II
Lossless	TGM-G	11.2%	14.5%	16.3%	9.1%	12.1%	14.0%
	MIX-G	1.0%	1.0%	1.1%	0.5%	0.5%	0.6%
	AMT-G	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
	TGM-Y	14.3%	17.8%	19.5%	11.9%	15.1%	17.0%
	MIX-Y	0.5%	0.5%	0.6%	0.3%	0.3%	0.3%
	AMT-Y	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
Lossy	TGM-G	-11.6%	-17.5%	-20.9%	-8.1%	-13.1%	-16.3%
	MIX-G	-4.0%	-4.6%	-5.5%	-3.2%	-3.8%	-4.2%
	AMT-G	0.1%	0.1%	0.1%	0.1%	0.0%	0.1%
	TGM-Y	-7.9%	-12.3%	-16.1%	-4.3%	-6.8%	-10.5%
	MIX-Y	-3.3%	-4.3%	-5.8%	-2.6%	-3.2%	-4.0%
	AMT-Y	0.2%	0.2%	0.2%	0.2%	0.3%	0.3%

There is almost no difference regarding the coding efficiency of 1D, Hybrid I and Hybrid II cTIM algorithms on MIX content for RA and LB scenarios, with very marginal gain over HEVC-REXt 6.0 at lossless scenario. But for lossy encoding, cTIM provides over averaged 3% improvement of MIX content.

2) Results for AMT Content: It is worth to note that cTIM (including its three options) does not show any performance improvement for AMT sequences. This is due to the fact that AMT sequence is very close to the camera-captured natural content, where normal HEVC coding technologies can compress it very well. Also our cTIM is developed with dedication to the high contrast discontinuous tone content, thus it does not benefit natural scene very much. As will be

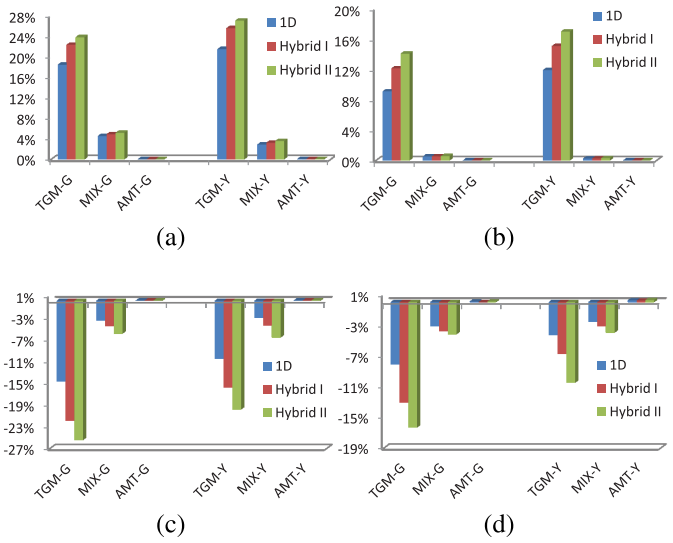


Fig. 9. Averaged performance improvement of 1D, Hybrid I and Hybrid II cTIM over HEVC-REXt 6.0. (a) AI lossless. (b) LB lossless. (c) AI lossy. (d) LB lossy.

unfolded in later sections, other techniques, such as IntraBC and string search based dictionary coding, have the similar behavior without noticeable gains on AMT sequence.

3) Summary: Our cTIM has shown the largest performance improvement (i.e., with averaged 26%, 18%, and 16% lossless bit rate reduction and 23%, 18% and 13% BD-Rate improvement of Hybrid II scheme for AI, RA and LB encoder settings, respectively) to encode the typical screen content with text and graphics (TGM category). Moderate gain (i.e., with respective 4.4%, 1.0% and 0.4% lossless bit rate reduction and 6.4%, 5.7%, 4.1% lossy BD-Rate improvement of Hybrid II for AI, RA and LB) can be obtained for mixed content where camera-captured natural content and typical screen content are both presented in the same scene. However, cTIM does not benefit camera-captured natural sequence according to the simulations performed in this paper. Fig. 9 pictures the averaged performance gains for each content category over HEVC-REXt 6.0. Results for AI and LB encoding settings are presented here while RA has the similar trend.

We also collect the encoding run-time for cTIM method as a relative measurement of the computational complexity against the anchor. In order to obtain trustworthy run-time, simulations are dispatched among similar computing nodes powered by Windows Server 2008 R2 as much as possible. On average, 1D cTIM requires 9% more run-time while Hybrid I and II require 20% and 31% more run-time compared with the HEVC-REXt 6.0 for AI configuration. The encoding run-time increase numbers are 3%, 5% and 8% for RA, and 2%, 3% and 6% for LB, of 1D, Hybrid I and II, respectively. The computational complexity increase is mainly due to the string search, pixel-to-index conversion and neighbor block color table derivation. As will be discussed in conclusion, better trade-off between complexity and performance is now studied in core experiment to further improve the cTIM.

B. Performance Evaluation of Other Algorithms

In addition, we have also conducted the simulations for the IntraBC, state-of-the-art palette coding method and the latest string search based dictionary coding. For IntraBC, we take the latest implementation in [41] (which is the same implementation as in the well performed CfP response [9]) to include three simulation points:

- IBC I: IntraBC search is constrained within current CTU and left three CTUs (4 CTUs in total);
- IBC II: IntraBC search is constrained within current CTU, left 3 CTUs and upper 4 CTUs (2×4 CTU window);
- IBC FF: IntraBC search is extended to full frame (as the default configuration provided by [41]).

Color palette coding method is selected from [17] which represents the state-of-the-art performance and is recommended by the standard ad-hoc group as test model for investigation. It is developed and harmonized from multiple technical proposals, such as [15] and [37], over several standardization meeting cycles. Two simulations are performed for color palette mode, i.e.:

- CP: default color palette mode described in [17] is applied on top of the HEVC-REXt 6.0 reference software, where the maximum palette size is 32 (by including escape color);
- CP-Ext: ColorPalette method [17] with the maximum palette size extended from 32 to 128 is used.

String search based dictionary coding is chosen from the latest implementation on top of the HEVC-REXt 6.0 [12] with two tests considering the different cache size of the dictionary:

- P2M: Default implementation of [12] is applied with the dictionary cache setting as level 4 (corresponding to the size of 64 Kbyte or KB);
- P2M-Ext: Dictionary cache level is upgraded to level 6 with corresponding size of 1 Mbyte or MB; other parameters are the same as [12].

According to the extensive simulations, these additional seven tests have also shown the larger gains for TGM sequences, relative smaller gains for MIX content while almost no gain for AMT sequence. Detailed comparative study regarding the coding efficiency as well as the complexity concerns are presented in the following section.

C. Space Complexity Analysis and Performance Comparison

This section attempts to analyze the algorithm complexity with the emphasis on the additional on-chip memory capacity and bandwidth requirement for hardware implementation. Meanwhile, we also perform the performance comparison for the algorithms having similar memory requirement.

From the decoder point of view, we can see the following various scenarios, i.e.,

1) Scenario #0: cTIM 1D [16], CP and CP-Ext [17] require marginal increase of on-chip buffer to cache color table at respective $128 \times 3 = 384$, $32 \times 3 = 96$ and 384 bytes. All perspective processing is limited within current CU. Referring to the 124 KB on-chip memory required by the state-of-the-art 4K Ultra-HD HEVC decoder chip implementation [42],

TABLE V
PERFORMANCE EVALUATION AND COMPARISON OF 1D cTIM AND CP-EXT AGAINST HEVC-REXT 6.0

		AI		RA		LB	
		1-D	CP-Ext	1-D	CP-Ext	1-D	CP-Ext
Lossless	TGM-G	18.4%	10.9%	11.2%	4.1%	9.1%	1.5%
	MIX-G	4.5%	1.1%	1.0%	0.1%	0.5%	0.0%
	AMT-G	0.0%	0.0%	0.0%	0.0%	0.0%	0.0%
	TGM-Y	21.5%	13.2%	14.3%	5.6%	11.9%	2.4%
	MIX-Y	2.9%	0.3%	0.5%	0.0%	0.3%	0.0%
	AMT-Y	0.1%	0.0%	0.0%	0.0%	0.0%	0.0%
Lossy	TGM-G	-14.7%	-8.5%	-11.6%	-5.7%	-8.1%	-1.6%
	MIX-G	-3.6%	-1.1%	-4.0%	-4.3%	-3.2%	-1.2%
	AMT-G	0.0%	0.1%	0.1%	0.1%	0.1%	0.0%
	TGM-Y	-10.6%	-6.8%	-7.9%	-4.3%	-4.3%	-1.4%
	MIX-Y	-3.1%	-0.9%	-3.3%	-1.2%	-2.6%	-0.9%
	AMT-Y	0.1%	0.1%	0.2%	0.2%	0.2%	0.1%

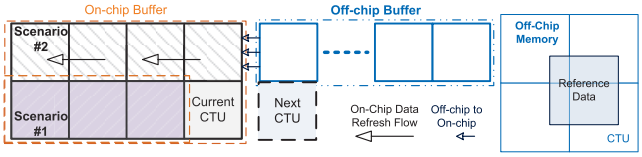
additional on-chip buffer increase is less than 0.3%. Note that this percentage number just gives a rough idea about the relative buffer increase. Exact on-chip buffer size would depend on the actual architectures of implementation.

Since we are using 128 as the maximum colors allowed in the table, we extend the default 32 used in CP method [17] to 128 for fair comparison. Except up to 1.9% gain is observed for TGM-Y lossless AI encoding, CP-Ext does not show noticeable gain over CP, but even slightly loss for lossy encoding. This is mainly because of the index map coding schemes applied in CP method, where index "run" and index "copy above" modes are less efficient when increasing the maximum color size. For example, if the successive pixels are quite similar with small intensity difference, in the case that 32 is used as the maximum table size, many pixels can be represented using a single index, which increases the possibility for longer length of "run"; however, if alternative 128 is used as the maximum table size, pixels which are merged to a single index can be differentiated using various index values, resulting in more bits to encode them one by one rather than the single "run" representation aforementioned.

Table V demonstrates the averaged coding efficiency improvement of 1D cTIM and CP-Ext against the anchor. Meanwhile we also measure the *relative gain (or improvement)* by calculating the direct difference of the bit rate or BD-Rate percentage between 1D and CP-Ext algorithms. For TGM content, about 8% relative gain is shown for lossless bit rate reduction of 1D over CP-Ext, while approximately 5% lossy BD-Rate improvement is recorded; For MIX content, 1D presents another 3% and 2% improvement over CP-Ext for lossless and lossy encoding using AI configuration, while averaged 1% (up to 3%) improvement for RA and LB.

2) Scenario #1: For Hybrid I and IBC I, left three more CTUs are used to derive the reference for prediction as shown in Fig. 10a (reference area marked as "scenario #1"). Each 64 × 64 CTU need 12 KB for 8-bit 4:4:4 content. It is approximately another 36 KB for both methods,[7] which is about 30% on-chip memory space increase on top of the reference point [42]. We also include P2M of string search based dictionary coding into this category where it requires

[7]Hybrid-cTIM I actually requires (384 + 36K) bytes.



(a) (b)

Fig. 10. Illustration of decoder on-chip memory structure (a) sliding window (2×4 as exemplified) based on-chip buffer data refresh and updating; (b) CTU memory non-aligned reference data (worst case estimation).

TABLE VI
PERFORMANCE EVALUATION AND COMPARISON OF HYBRID I, IBC I AND P2M AGAINST HEVC-REXT 6.0

			TGM-G	MIX-G	AMT-G	TGM-Y	MIX-Y	AMT-Y
Lossless	AI	Hybrid I	22.3%	4.9%	0.0%	25.5%	3.2%	0.1%
		IBC I	5.9%	1.4%	0.0%	6.6%	1.5%	0.0%
		P2M	18.9%	5.6%	0.0%	21.2%	4.9%	0.0%
	RA	Hybrid I	14.5%	1.0%	0.0%	17.8%	0.5%	0.0%
		IBC I	4.9%	0.3%	0.0%	5.5%	0.3%	0.0%
		P2M	12.0%	1.8%	0.0%	14.4%	1.2%	0.0%
	LB	Hybrid I	12.1%	0.5%	0.0%	15.1%	0.3%	0.0%
		IBC I	4.5%	0.2%	0.0%	5.0%	0.2%	0.0%
		P2M	9.9%	1.4%	0.0%	12.0%	0.9%	0.0%
Lossy	AI	Hybrid I	-21.9%	-4.6%	0.0%	-15.9%	-4.5%	0.1%
		IBC I	-7.2%	-2.9%	0.0%	-7.0%	-3.1%	0.0%
		P2M	-9.4%	-0.8%	0.0%	-6.2%	-0.3%	0.1%
	RA	Hybrid I	-17.5%	-4.6%	0.1%	-12.3%	-4.3%	0.2%
		IBC I	-5.2%	-1.7%	0.0%	-4.9%	-2.1%	0.0%
		P2M	-7.8%	-1.4%	0.0%	-5.3%	-0.6%	0.2%
	LB	Hybrid I	-13.1%	-3.8%	0.0%	-6.8%	-3.2%	0.2%
		IBC I	-4.1%	-0.8%	-0.1%	-3.6%	-1.0%	0.1%
		P2M	-4.7%	-1.0%	0.1%	-2.4%	-0.4%	0.2%

64 KB (about 60% increase) for its on-chip cache requirement at level 4 [12]. Note that there is no extra memory bandwidth (between on-chip and off-chip) by assuming the additional 30% (or 60% of P2M) on-chip memory increase.

Table VI summarizes the averaged lossless and lossy performance of Hybrid I, IBC I and P2M, respectively. For example, for TGM content, Hybrid I demonstrates averaged 17.6%, 10.9% and 8.9% relative gain of lossless AI, RA and LB, and 11.8%, 9.8% and 2.0% of lossy AI, RA and LB, over IBC I, while corresponding 3.9%, 3.0% and 2.7% relative gain of lossless AI, RA and LB, and 11.1%, 8.4% and 6.4% of lossy AI, RA and LB, over P2M, respectively.

P2M has shown the better lossless coding performance for MIX content than proposed cTIM Hybrid I, as revealed in Table VI. One possible reason is due to the fairly larger on-chip caching size. However, P2M does not show consistent superior performance for lossy encoding. This is because of the *lossless* match design behind this particular P2M implementation, even for the lossy scenario. P2M will search lossy reconstructed neighbors to find matches for the original current block, resulting in the lossless P2M coded blocks surrounded by the lossy HEVC coded blocks (with quantization noise).

3) Scenario #2: For Hybrid II and IBC II, in addition to left three more CTUs, upper four CTUs are included for reference. Typically, we would like to add whole line of reference inside the chip to avoid frequent I/O that will result in significant memory bandwidth increase. However, including additional one CTU line into the on-chip buffer may not be possible for practical implementation since it is

TABLE VII
PERFORMANCE EVALUATION AND COMPARISON OF HYBRID II AND IBC II AGAINST HEVC-REXT 6.0

			TGM-G	MIX-G	AMT-G	TGM-Y	MIX-Y	AMT-Y
Lossless	AI	Hybrid II	23.9%	5.2%	0.0%	27.2%	3.6%	0.1%
		IBC II	11.6%	3.6%	0.0%	12.7%	4.1%	0.0%
	RA	Hybrid II	16.3%	1.1%	0.0%	19.5%	0.6%	0.0%
		IBC II	9.4%	1.0%	0.0%	10.3%	1.1%	0.0%
	LB	Hybrid II	14.0%	0.6%	0.0%	17.0%	0.3%	0.0%
		IBC II	8.8%	0.7%	0.0%	9.5%	0.7%	0.0%
Lossy	AI	Hybrid II	-25.5%	-6.0%	0.0%	-19.9%	-6.7%	0.1%
		IBC II	-15.2%	-9.2%	-0.1%	-15.0%	-9.8%	-0.1%
	RA	Hybrid II	-20.9%	-5.5%	0.1%	-16.1%	-5.8%	0.2%
		IBC II	-10.6%	-5.7%	-0.1%	-10.2%	-6.7%	-0.1%
	LB	Hybrid II	-16.3%	-4.2%	0.1%	-10.5%	-4.0%	0.3%
		IBC II	-8.4%	-3.2%	-0.1%	-7.5%	-3.5%	0.1%

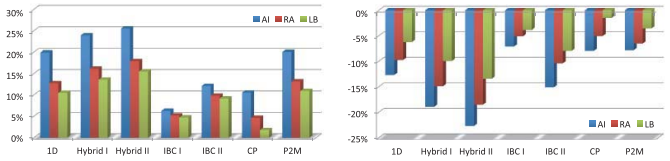
TABLE VIII
PERFORMANCE EVALUATION AND COMPARISON OF HYBRID II, IBC FF, AND P2M-EXT AGAINST HEVC-REXT 6.0

			TGM-G	MIX-G	AMT-G	TGM-Y	MIX-Y	AMT-Y
Lossless	AI	Hybrid II	23.9%	5.2%	0.0%	27.2%	3.6%	0.1%
		IBC FF	14.6%	8.0%	0.0%	16.0%	8.4%	0.0%
		P2M-Ext	24.2%	9.0%	0.0%	25.8%	7.7%	0.0%
	RA	Hybrid II	16.3%	1.1%	0.0%	19.5%	0.6%	0.0%
		IBC FF	12.2%	1.9%	0.0%	13.4%	2.1%	0.0%
		P2M-Ext	15.8%	2.9%	0.0%	17.9%	1.8%	0.0%
	LB	Hybrid II	14.0%	0.6%	0.0%	17.0%	0.3%	0.0%
		IBC FF	11.3%	1.1%	0.0%	12.3%	1.3%	0.0%
		P2M-Ext	13.4%	2.3%	0.0%	15.2%	1.2%	0.0%
Lossy	AI	Hybrid II	-25.5%	-6.0%	0.0%	-19.9%	-6.7%	0.1%
		IBC FF	-19.6%	-11.3%	-0.1%	-19.1%	-12.2%	-0.1%
		P2M-Ext	-15.5%	-2.0%	0.0%	-10.2%	-0.8%	0.1%
	RA	Hybrid II	-20.9%	-5.5%	0.1%	-16.1%	-5.8%	0.2%
		IBC FF	-14.2%	-7.1%	-0.1%	-13.6%	-8.3%	-0.1%
		P2M-Ext	-12.4%	-2.9%	0.0%	-8.3%	-1.2%	0.1%
	LB	Hybrid II	-16.3%	-4.2%	0.1%	-10.5%	-4.0%	0.3%
		IBC FF	-11.9%	-4.0%	-0.1%	-10.8%	-4.5%	0.0%
		P2M-Ext	-7.9%	-1.9%	0.1%	-3.9%	-0.7%	0.2%

about 4K × 64 × 3 = 712 KB for a 4K content. On the other hand, *sliding window* based scheme could be a realistic solution where we put $M \times N$ (i.e., 2×4 in our case) CTUs inside the chip and update the reference data CTU by CTU, as illustrated in Fig. 10a. Although it is doable, it still requires extra (12K × 7) = 84 KB to host the 2×4 buffer window and the memory bandwidth requirement is doubled ($2\times$) since we need to fetch the off-chip data (of reconstructed upper CTUs).

As revealed in Table VII, Hybrid II still outperforms the IBC II with quite large performance gap for TGM contents with averaged 13.4%, 8.1%, 3.7% and 7.6%, 8.1%, 5.5% gains for lossless and lossy AI, RA and LB respectively. However, IBC II gives better coding gains over Hybrid II for MIX content, particularly for AI lossy settings. This is mainly due to the fixed error allowance threshold 9 used in this paper to group the colors during table derivation phase. Rate-distortion optimized color table derivation will be the focus for next step.

4) Scenario #3: For IBC FF, it is impractical to host a whole frame inside the chip. Since IBC FF enables the full frame search range, it is possible to have the reference data anywhere as long as it is within current frame. It is not practical to have additional on-chip memory window to buffer a large chunk of prediction data (continuously) since the next block may refer to a region very far away. Memory bandwidth requirement



(a) (b)

Fig. 11. Overall averaged performance improvements over HEVC-REXt 6.0 for AI, RA and LB coded TGM contents. (a) Lossless. (b) Lossy.

TABLE IX
ESTIMATION OF EXTRA ON-CHIP MEMORY BUFFER (B OR BYTES) AND BANDWIDTH INCREASE REQUIREMENT

Scenario	Algorithm	Extra Buffer	Bandwidth
#0	1-D	384 B	1x
	CP [17]	96 B	1x
	CP-Ext [17]	384 B	1x
#1	Hybrid I	(384 B + 36 KB)	1x
	IBC I [9]	36 KB	1x
	P2M [12]	64 KB	1x
#2	Hybrid II	(384 B + 84 KB)	2x
	IBC II [9]	84 KB	2x
#3	IBC FF [9]	0	4x
	P2M-Ext [12]	1 MB	1x

could increase forth times ($4\times$) at worst if reference data is not aligned with CTU memory as shown in Fig. 10b.

P2M-Ext with additional 1 MB on-chip cache requirement is included into this category as well. Different from Hybrid II and IBC II, where the reference is constructed via 2×4 CTU window by including both left and upper CTUs close-by, P2M-Ext just caches the previously reconstructions.

We perform the comparison between Hybrid II, IBC FF and P2M-Ext and summarize the gains in Table VIII. As expected, IBC FF enlarges the gains for MIX content, and reduce the loss for TGM content, over Hybrid II, respectively. Also note that P2M-Ext shows quite impressive gains for lossless coded MIX content, with up to 4.1% relative gains over Hybrid II.

Another improved dictionary coding method (iDict) presented in [13] and [24] demonstrates even better performance over P2M-Ext with full frame search range, adaptive lossy and lossless match, etc. For lossy case, Hybrid II gives averaged 2.8%, 1.4% and 2.4% BD-Rate gains for TGM content while 5.4%, 5.1% and 4.3% gains for MIX content, at AI, RA and LB settings respectively. Oppositely for lossless encoding, Hybrid II increases respective 4.5%, 4.5% and 4.4% bit rate for TGM content while 3.0%, 0.2% and 0.1% for MIX content. Note that lossless bit rate increase means performance loss. For AMT videos, Hybrid II shares the similar performance with this iDict method. Note that the numbers reported here are using this iDict as the anchor. It is different from the results in other sections using HEVC-REXt 6.0 as the anchor.

D. Brief Summary

We have summarized the estimation for on-chip memory requirement in Table IX. Memory bandwidth is estimated against the decoder without including the target algorithm, i.e., $1\times$ means no extra bandwidth required and $2\times$ means

bandwidth requirement doubled. Excluding Scenario #3 where both full frame search and 1 MB on-chip cache are very difficult (or even impossible) to be realized in current silicon technology due to the trade-off consideration between cost and performance, we further picture the averaged coding gains over HEVC-RExt 6.0 for TGM content in Fig. 11.

Hybrid II gives the best performance with averaged 26%, 18%, 15% lossless bit rate reduction, and 23%, 19%, 13% BD-Rate improvement over HEVC-RExt 6.0, for AI, RA and LB coded TGM contents. For MIX content, Hybrid II still shows the best performance but with smaller numbers compared with the TGM sequences, i.e., 4%, 1%, and 0% of lossless AI, RA, and LB, and 6%, 6%, 4% of lossy AI, RA, and LB respectively.

Averaged relative gain from 1D to Hybrid I, and to Hybrid II decreases slightly, with about 3%, 5% of Hybrid I over 1D, and 2%, 4% of Hybrid II over Hybrid I, for respective lossless and lossy cases. For lossy encoding, even Hybrid I outperforms other algorithms, which is more favored in the practical implementation without memory bandwidth increase (with assumption that 4 CTUs can be buffered on-chip).

VIII. Conclusion

In this paper, we have proposed an advanced screen content coding method using color table and index map. We apply a 1D or hybrid 1D/2D string search for the compact representation of index map. Considering the color correlation among neighbors, we have also introduced color table merge to signal the table implicitly. Additional inter-table color sharing and intra-table color DPCM are developed for explicit table encoding.

cTIM Hybrid II provides averaged 26%, 18%, 15% lossless bit rate reduction, and 23%, 19%, 13% BD-Rate improvement over HEVC-RExt 6.0, for AI, RA and LB coded TGM contents. For MIX content, it still shows averaged 4%, 1%, and 0% improvement of lossless AI, RA, and LB, and 6%, 6%, 4% of lossy AI, RA, and LB respectively. Same as other algorithms, there is no clear evidence cTIM benefits camera-captured natural content encoding.

Meanwhile, we have also performed extensive simulations for algorithm benchmark, including the intra block copy, conventional color palette coding and string search based dictionary coding. Except for the full frame intra block copy, cTIM clearly outperforms all other algorithms. Even compared with full frame intra block copy, we demonstrate the noticeable gains for TGM sequences, but slightly loss for MIX content.

cTIM can be further improved in the following directions such as rate-distortion optimized adaptive color table derivation and matched pair selection, efficient tool for mixed content, etc. On the other hand, computational complexity is another concern for next phase development where better trade-off between complexity and performance will be carefully evaluated.

Moreover, cTIM demonstrates the noticeable coding gains for screen content. Its string search based index coding method is now under investigation in the core experiment formed by the standardization committee on top of the screen content reference software version 2 (i.e., SCM-2.0). Not only performance but also the complexity (such as the entropy coding throughput) will be carefully studied to further improve the overall algorithm design. Meanwhile, other core experiments, such as the palette coding improvement, etc., are also created to enhance the existing palette coding method in SCM-2.0.

References

[1] G. J. Sullivan, J. Ohm, W.-J. Han, and T. Wiegand, "Overview of the high efficiency video coding (HEVC) standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[2] G. J. Sullivan, J. M. Boyce, Y. Chen, J.-R. Ohm, A. Segall, and A. Vetro, "Standardized extensions of high efficiency video coding (HEVC)," *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 6, pp. 1001–1016, Dec. 2013.

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

[4] Y. Lu, S. Li, and H. Shen, "Virtualized screen: A third element for cloud-mobile convergence," *IEEE Multimedia*, vol. 18, no. 2, pp. 4–11, Feb. 2011.

[5] Z. Pan, H. Shen, Y. Lu, S. Li, and N. Yu, "A low-complexity screen compression scheme for interactive screen sharing," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 6, pp. 949–960, Jun. 2013.

[6] *Joint Call for Proposals for Coding of Screen Content*, Standard ISO/IEC JTC1/SC29/WG11 MPEG, MPEG N14715, Jan. 2014.

[7] D.-K. Kwon and M. Budagavi, *RCE3: Results of Test 3.3 on Intra Motion Compensation*, document JCTVC-N0205, Jul. 2013.

[8] C. Pang, J. Sole, L. Guo, M. Karczewicz, and R. Joshi, *Non-RCE3: Intra Motion Compensation With 2-D MVs*, document JCTVC-N0256, Jul. 2013.

[9] J. Chen *et al.*, *Description of Screen Content Coding Technology Proposal by Qualcomm*, document JCTVC-Q0031, Apr. 2014.

[10] S. Wang and T. Lin, *4:4:4 Screen Content Coding Using Macroblock-Adaptive Mixed Chroma-Sampling-Rate*, document JCTVC-H0073, Feb. 2012.

[11] W. Zhu, J. Xu, and W. Ding, *Screen Content Coding Using 2-D Dictionary Mode*, document JCTVC-O0357, Oct. 2013.

[12] J. Ye, S. Liu, S. Lei, X. Chen, L. Zhao, and T. Lin, *Improvements on 1D Dictionary Coding*, document JCTVC-Q0124, Apr. 2014.

[13] B. Li, J. Xu, F. Wu, X. Guo, and G. J. Sullivan, *Description of Screen Content Coding Technology Proposal by Microsoft*, document JCTVC-Q0035, Apr. 2014.

[14] L. Guo, M. Karczewicz, and J. Sole, *RCE3: Results of Test 3.1 on Palette Mode for Screen Content Coding*, document JCTVC-N0247, Jul. 2013.

[15] W. Zhu, J. Xu, and W. Ding, *RCE3 Test 2: Multi-Stage Base Color and Index Map*, document JCTVC-N0287, Jul. 2013.

[16] Z. Ma, W. Wang, M. Xu, X. Wang, and H. Yu, *Description of Screen Content Coding Technology Proposal by Huawei Technologies (USA)*, document JCTVC-Q0034, Apr. 2014.

[17] W. Pu, X. Guo, P. Onno, P. Lai, and J. Xu, *AHG10: Suggested Software for Palette Coding based on RExt6.0*, document JCTVC-Q0094, Apr. 2014.

[18] W. Wang, Z. Ma, M. Xu, X. Wang, and H. Yu, *AHG8: String Match in Coding of Screen Content*, document JCTVC-Q0176, Apr. 2014.

[19] M. Naccari, C. Rosewarne, K. Sharman, and G. J. Sullivan, *HEVC Range Extensions Test Model 6 Encoder Description*, document JCTVC-P1013, Jan./Feb. 2014.

[20] G. Bjontegaard, *Calculation of Average PSNR Differences Between R-D Curves*, document VCEG-M33, ITU-T VCEG, Apr. 2001.

[21] A. Zaccarin and B. Liu, "A novel approach for coding color quantized images," *IEEE Trans. Image Process.*, vol. 2, no. 4, pp. 442–453, Oct. 1993.

[22] W. Zeng, J. Li, and S. Lei, "An efficient color re-indexing scheme for palette-based compression," in *Proc. IEEE ICIP*, Dec. 2000, pp. 476–479.

[23] X. Li, "Palette-based image compression method, system and data file," U.S. Patent 7 162 077, Oct. 11, 2001.

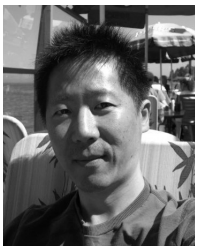
[24] B. Li and J. Xu, *SCCE4: Result of Test 3.1*, document JCTVC-R0098, Jul. 2014.

[25] J. Ohm, G. J. Sullivan, H. Schwarz, T. K. Tan, and T. Wiegand, "Comparison of the coding efficiency of video coding standards—Including high efficiency video coding (HEVC)," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1669–1684, Dec. 2012.
[26] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H.264/AVC video coding standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 7, pp. 560–576, Jul. 2003.
[27] C. C. Chi *et al.*, "Parallel scalability and efficiency of HEVC parallelization approaches," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1827–1838, Dec. 2012.
[28] V. Sze and M. Budagavi, "High throughput CABAC entropy coding in HEVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1778–1791, Dec. 2012.
[29] I.-K. Kim, J. Min, T. Lee, W.-J. Han, and J. Park, "Block partitioning structure in the HEVC standard," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1697–1706, Dec. 2012.
[30] *Standardization Plan for HEVC Extensions for Screen Content Coding*, Standard ITU-T Q6/16 and ISO/IEC JTC1/SC29/WG11, MPEG N14520, Apr. 2014.
[31] T.-S. Chang *et al.*, *RCE3: Results of Subtest B.1 on Nx2N/2NxN Intra Block Copy*, document JCTVC-P0176, Jan. 2014.
[32] C. Pang, J. Sole, L. Guo, and M. Karczewicz, *RCE3: Subtest B.3—Intra Block Copy With NxN PU*, document JCTVC-P0145, Jan. 2014.
[33] W. Zhu, J. Xu, W. Ding, Y. Shi, and B. Yin, "Adaptive LZMA-based coding for screen content," in *Proc. Picture Coding Symp.*, Dec. 2013, pp. 373–376.
[34] T. Lin, K. Zhou, X. Chen, and S. Wang, "Arbitrary shape matching for screen content coding," in *Proc. Picture Coding Symp.*, Dec. 2013.
[35] B. Li, J. Xu, and F. Wu, *Screen Content Coding Using Dictionary Based Mode*, document JCTVC-P0214, Jan. 2014.
[36] S.-L. Yu and C. Chrysafis, *New Intra Prediction Using Intra-Macroblock Motion Compensation*, document JVT-C151, May 2002.
[37] L. Guo, M. Karczewicz, J. Soel, and R. Joshi, *Non-RCE3: Modified Palette Mode for Screen Content Coding*, document JCTVC-N0249, Jul. 2013.
[38] J. Soel *et al.*, "Transform coefficient coding in HEVC," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 22, no. 12, pp. 1765–1777, Dec. 2012.
[39] T. Nguyen *et al.*, "Transform coding techniques in HEVC," *IEEE J. Sel. Topics Signal Process.*, vol. 7, no. 6, pp. 1765–1777, Dec. 2013.
[40] X. Wu, "Color quantization by dynamic programming and principal analysis," *ACM Trans. Graph.*, vol. 11, no. 4, pp. 348–372, Oct. 1992.
[41] K. Rapaka, C. Pang, J. Sole, and M. Karczewicz, *Software for the Screen Content Coding Model*, document JCTVC-Q0243, Apr. 2014.
[42] M. Tikekar, C.-T. Huang, C. Juvekar, V. Sze, and A. P. Chandrakasan, "A 249-Mpixel/s HEVC video-decoder chip for 4K ultra-HD applications," *IEEE J. Solid-State Circuits*, vol. 49, no. 1, pp. 61–72, Jan. 2014.

Zhan Ma (S'06–M'11) received the B.S.E.E. and M.S.E.E. degrees from the Huazhong University of Science and Technology, Wuhan, China, in 2004 and 2006, respectively, and the Ph.D. degree from the New York University Polytechnic School of Engineering, Brooklyn, NY, USA, in 2011.

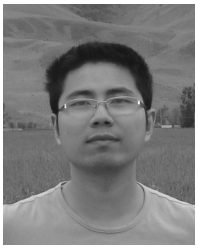
He was with Samsung Research America, Dallas, TX, USA, from 2011 to 2013.

Dr. Ma is currently with Futurewei Technologies, Inc., Santa Clara, CA, USA, and Nanjing University, Nanjing, China. His current research focuses on the next-generation video coding standardization, screen content coding and sharing, and video signal modeling.



Wei Wang received the B.S. and M.S. degrees in electronic engineering from Fudan University, Shanghai, China, in 1995 and 1998, respectively.

He founded his company in Vancouver, BC, Canada, from 2003 to 2009, with a focus on the research and development of JPEG2000 software and hardware product. Since 2011, he has been with Huawei Technologies, Santa Clara, CA, USA, and Futurewei Technologies, Santa Clara, as a Video Architect. His research interests include image, video compression, and screen content coding.



Meng Xu received the B.S. degree in physics from Nanjing University, Nanjing, China, in 2006, and the M.S. and Ph.D. degrees in electrical engineering from the New York University Polytechnic School of Engineering, Brooklyn, NY, USA, in 2009 and 2014, respectively. His research interests include video coding and its applications. During pursuing the Ph.D. degree, he interned at the Dialogic Media Laboratory, Eatontown, NJ, USA, in 2010, Samsung Telecommunications America Inc., Richardson, TX, USA, in 2013, and Huawei Technologies, Santa Clara, CA, USA, in 2013.

He has been with Huawei Technologies, since 2014, as a Video Coding Engineer and Researcher. His current research focuses on the standardization of HEVC screen content coding.

Haoping Yu received the B.S. and M.S. degrees from the University of Science and Technology of China, Hefei, China, in 1984 and 1987, respectively, and the Ph.D. degree from the University of Central Florida, Orlando, FL, USA, in 1995, all in electrical engineering.

He was a member/Senior Member/Principal Member of Technical Stuff of the Corporate Research with Thomson Consumer Electronics Inc., Indianapolis, IN, USA, and Thomson Multimedia, Paris, France, from 1995 to 2008. Since 2008, he has been with Huawei Technologies, Santa Clara, CA, USA, and Futurewei Technologies, Santa Clara, as the Head of the Video Technology Research Laboratory. His research interests include video compression, processing, and delivery for both consumer and professional applications.