Pseudo 2D String Matching Technique for High Efficiency Screen Content Coding

Liping Zhao, Tao Lin, Kailun Zhou, Shuhui Wang, and Xianyi Chen

***Abstract*—This paper proposes a pseudo 2D string matching (P2SM) technique for high efficiency screen content coding (SCC). The technique uses a primary reference buffer (PRB) and a secondary reference buffer (SRB) for string matching and string copying. In the encoder, optimal reference string searching is performed in both PRB and SRB, and either a PRB or an SRB string is selected as an optimal reference string on a string-by-string basis. If no reference string of at least one pixel is founded for a current pixel, then the current pixel is coded as an unmatched pixel. Compared with HM-16.4+SCM-4.0 reference software, the proposed P2SM technique achieves up to 37.7% Y BD-rate reduction for a screen snapshot of a spreadsheet. On average, using HEVC SCC common test condition and YUV test sequences in text and graphics with motion category, the proposed technique achieves Y BD-rate reduction of 7.7%, 5.0%, 2.6% for all intra (AI), random access (RA) and low-delay B (LB) configurations, respectively in lossy coding with both intra block copy (IBC) and P2SM having the same 4 coding tree units (CTUs) searching range, and bit-rate saving of 6.0%, 3.9%, 3.1% for AI, RA, LB configurations, respectively in lossless coding with IBC having full frame searching range while P2SM having only 2 CTUs searching range, at very low additional encoding and decoding complexity.**

***Index Terms*—Block matching, hash-table, high efficiency video coding (HEVC), screen content coding (SCC), string matching.**

I. Introduction

With continuous and rapid advancements in communications, networking, computers, semiconductors, and displays technologies, screen content coding (SCC) becomes a key technology for some emerging popular applications such as cloud computing, cloud-mobile computing, WIFI display, smart-phone and tablet second display, ultra-thin client, remote desktop, and screen sharing [1]–[3]. In recent years, SCC has attracted increasing attention of researchers from both academia and industry [4]–[37].[1] In response to the demand on SCC, ISO/IEC MPEG and ITU-T VCEG are jointly developing an SCC version of high efficiency video coding (HEVC) standard [4].

Screen content has some characteristics quite different from camera-captured content, such as sharp edges, few colors in a coding unit (CU), repeated identical patterns with a variety of shapes and sizes. Conventional video coding techniques such as block matching based intra/inter prediction and DCT-based transform coding [5] cannot code screen content effectively. To achieve high coding efficiency for screen content, new coding tools are necessary.

Two of the major tools adopted in HEVC SCC draft up to now are Intra Block Copy (IBC) coding [6]–[8] and palette coding (PLT) [9], [10]. IBC is efficient to code repeated identical patterns of a few fixed sizes with rectangle or square shapes in a picture, but is not flexible enough to code repeated identical patterns of varying sizes from a few pixels to a few thousands of pixels with a variety of shapes. PLT can code repeated identical patterns inside a CU using two types of intra-CU pixel-index string matching, but it cannot exploit non-local repeated identical patterns.

To address these issues and improve the coding efficiency of repeated identical patterns with a variety of sizes and shapes, a few approaches based on generic string matching instead of block matching or intra-CU string matching were proposed [12]–[29]. These string matching approaches are all variations of early dictionary coding technique [12]–[16], which applies LZ algorithm [11] to CU-based picture coding. The basic operation of string matching is to break a current CU being coded into multiple strings of variable size and to search for longest reference matching strings one by one in a predetermined searching range, which is also called a dictionary in some literatures and is actually a 1D or 2D sliding window reference buffer in the current reconstructed picture. Previous string matching approaches include CTU or macro-block based 1D string matching [12]–[16], string matching of palette index [17], arbitrary shape 2D string matching [18].

However, these string matching based approaches do not show sufficient coding performance improvement on top of the IBC and PLT tools. This is mainly due to that only a single sliding window is used as the reference buffer and the coding of string matching parameters such as reference string offset and string length is not optimized.

In this paper, Pseudo 2D String Matching (P2SM) technique is proposed for high efficiency screen content coding.

The P2SM technique has a generic, flexible and uniform coding system architecture to exploit both local and non-local repeated identical patterns with a variety of sizes and/or shapes in a wide range of generic screen content. In the P2SM coding

Manuscript received August 3, 2015; revised November 18, 2015; accepted December 16, 2015. Date of publication December 24, 2015; date of current version February 18, 2016. This work was supported in part by the National Natural Science Foundation of China under Grant 61201226 and Grant 61271096 and in part by the Specialized Research Fund for the Doctoral Program of Higher Education under Grant 20130072110054. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Zhu Li. *(Corresponding author: Tao Lin.)*

The authors are with the Institute of VLSI, College of Electronics and Information Engineering, Tongji University, Shanghai 200092, China (e-mail: Lintao@tongji.edu.cn).

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/TMM.2015.2512539

[1][Online]. Available: https://hevc.hhi.fraunhofer.de/svn/svn_HEVCSoftware/tags/HM-16.4+SCM-4.0

system architecture, two reference buffers are used. One is primary reference buffer (PRB) which is typically a part of the traditional reconstructed picture buffer to provide reference string pixels for the current pixels being coded. The other is secondary reference buffer (SRB) which is a dynamically updated lookup table (LUT) storing a few of recently and frequently referenced pixels for repetitive reference by the current pixels being coded. In the encoding process, for any starting pixel being coded, searching of optimal reference string is performed in both PRB and SRB. As a result of the searching, either a PRB string or an SRB string is selected as an optimal reference string on a string-by-string basis. For a PRB string, an offset and a length are coded into the bitstream. For an SRB string, an SRB index and a length are coded into the bitstream. If no reference string of at least one pixel is found in PRB or SRB, the starting pixel is coded directly into the bitstream as an unmatched pixel preceded by a special SRB index. In lossless P2SM, the absolute difference between a current pixel and a reference pixel is zero. In lossy P2SM, the absolute difference between a current pixel and a reference pixel is within a predetermined limit and no P2SM prediction residual is coded. Moreover, in P2SM, string matching parameter coding is also optimized.

Section II introduces the background and related work of P2SM. Section III proposes a general architecture of P2SM enhanced coding system using both PRB and SRB, discusses some basic operations of P2SM, describes major syntax elements and semantics of P2SM, and the relations of P2SM, IBC, PLT, traditional string matching. Section IV presents some technical details of P2SM including hash-table based PRB string searching strategy, optimal reference string selection, matching parameter coding optimization, and memory access bandwidth and context-adaptive binary arithmetic coding (CABAC) throughput. Some experimental results are given in Section V to show that compared with HM-16.4+SCM-4.0 reference software,[1] the proposed technique can achieve significant Bjøntegaard delta rate (BD-rate) [30], [31] reduction in lossy coding and bit-rate saving in lossless coding. Section VI is conclusions and future work. This paper is partially based on JCT-VC documents [27]–[29] and [32]–[35], in which P2SM is renamed as Intra String Copy (ISC) or Palette with Pixel Copy (PPC).

II. Background and Related Work

A. Characteristics of Screen Content

Screen content is video or picture captured from a computer screen. Typically, screen content is captured by indirectly reading frame buffers of computer graphics devices or directly capturing digital display output signals of computer graphics devices. As an important and comprehensive class of video or picture, screen content exhibits very different characteristics from natural video or picture captured by cameras. A typical screen content picture is shown in Fig. 1, which is part of sc_web_browsing sequence, one of SCC test sequences in HEVC SCC common test condition (CTC) [37].

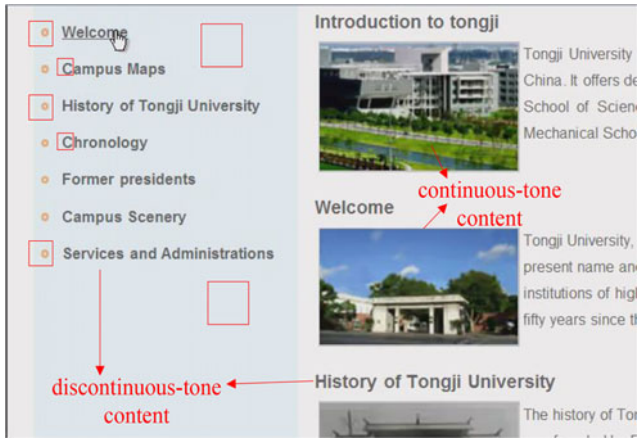

Fig. 1. Typical screen content.

Screen content has at least the following characteristics.

1) As shown in Fig. 1, screen content is mainly a mix of two types of essentially different contents: (1) discontinuous-tone content such as texts, charts, graphs, menus, window frames, slider bars, control buttons, and icons, and (2) continuous-tone content such as photographs, movie/TV clips, natural pictures, natural video sequences, and sophisticated light-shaded and texture-mapped virtual-reality scenes.
2) For discontinuous-tone content, colors within one CU usually concentrate on a few values with high occurrence count. In other words, the number of colors in a local block of screen content is limited.
3) There are usually many repeated identical patterns in a screen picture. The red square boxes shown in Fig. 1 illustrate some repeated identical patterns. These repeated identical patterns may have varying sizes from a few pixels to a few thousands of pixels with a variety of shapes.
4) Most non-edge regions in natural video or picture are smooth, while most non-natural video or picture regions have very sharp edges and very thin lines, even one-pixel-wide single-color lines. When represented in frequency domain, energy of natural video or picture is concentrated at low frequency coefficients, however the energy of non-natural video or picture is scattered over coefficients in all frequencies [9]. Transform process may blur sharp edges and thin lines.

B. IBC and PLT Tools

Two of the major SCC tools adopted in HEVC SCC draft up to now are IBC [6]–[8] and PLT [9], [10].

IBC: Repeated identical patterns are often observed on the same picture of screen content. IBC is designed to find the repeated identical patterns at block level. IBC performs block matching in a part of the reconstructed region of current picture to find the best prediction for the current CU. IBC is a direct extension of traditional inter prediction technique to the current picture.

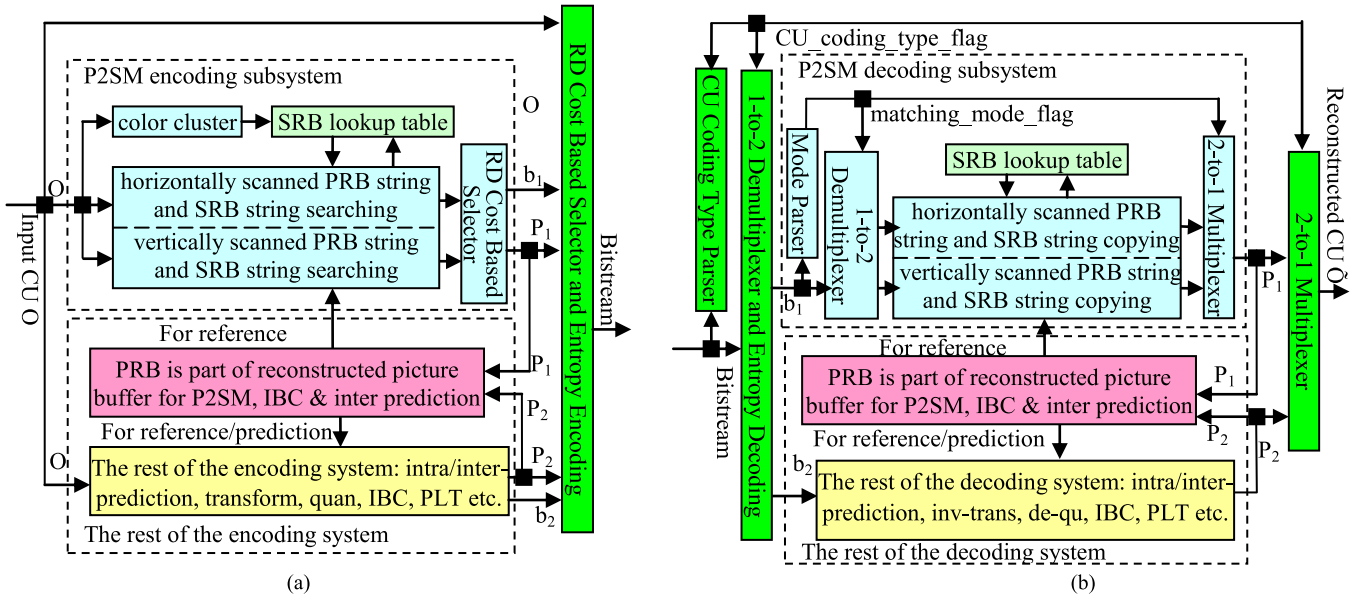

Fig. 2. P2SM enhanced coding system architecture. (a) Encoder (b) Decoder.

PLT: Based on the characteristics of screen content, colors within a CU usually concentrate on a few values with high occurrence count. PLT introduces a color palette of these color values. After getting the palette, pixels within one CU are mapped to the palette indices. Then, an effective two-mode run length coding method is applied to effectively compress the repeated identical pattern in the mapped indices of the CU. No transform process is invoked in PLT to avoid blurring sharp edges that can significantly impact the visual quality of screen content.

III. Pseudo 2D String Matching

A. P2SM Enhanced Coding System Architecture

A general architecture and major components of P2SM enhanced coding system are shown in Fig. 2.

In the encoder of the P2SM enhanced coding system, the P2SM encoding subsystem performs color clustering, PRB string searching, and SRB string searching. The rest of the encoding system performs traditional encoding operations such as intra prediction, inter prediction, transform, quantization, entropy encoding, IBC, and PLT. The input CU O is fed to both P2SM encoding subsystem and the rest of the encoding system. The P2SM encoding subsystem codes O to generate coded bitstream b_1 and reconstructed CU P_1. The rest of the encoding system also codes O to generate coded bitstream b_2 and reconstructed CU P_2. O, $\mathrm{P}_1, \mathrm{b}_1, \mathrm{P}_2$, and b_2 are sent to rate-distortion (RD) cost based selector that calculates the RD cost of two results and selects the one with the best RD performance as the final coding result for the CU. The corresponding coded bitstream b_1 or b_2 is selected and put into the output bitstream. Also, the corresponding reconstructed CU P_1 or P_2 is stored in reconstructed picture buffer, which is shared by both P2SM encoding subsystem as PRB for P2SM and the rest of the encoding system for inter prediction and IBC. The input CU O is also fed to a color cluster unit to obtain a few of most frequently occurred pixel colors. Some of the colors are put into the SRB LUT for SRB string searching. Therefore, a new SRB is generated after the color cluster operation for every CU. The maximum number of colors in an SRB is 32. Any new SRB color that is not in an SRB color first in first out (FIFO) buffer holding the SRB colors of previously P2SM-coded CUs will be put into the bitstream to be transferred to P2SM decoder. If an SRB color happens to be in the SRB color FIFO buffer which is maintained in both encoder and decoder, then the position of the SRB color in the SRB color FIFO buffer, instead of the SRB color itself, will be put into the bitstream to be transferred to P2SM decoder. The number of bits spent to signal a position is usually much less than that spent to transfer an SRB color.

In the decoder, the input bitstream is firstly parsed by CU coding type parser to get the CU_coding_type_flag that indicates if the current CU being decoded is coded by the proposed P2SM technique or by traditional coding tools. If the CU is coded by P2SM technique, then the CU layer bitstream is sent to the P2SM decoding subsystem that decodes and reconstructs the CU P_1, which is stored in reconstructed picture buffer and is also the final reconstructed CU output Õ. If the CU is coded by traditional coding tools, then the CU layer bitstream is sent to the rest of the decoding system that performs traditional decoding tasks such as intra prediction, inter prediction, inverse-transform, de-quantization, IBC, PLT decoding, and eventually reconstructs the CU P_2, which is stored in reconstructed picture buffer and is also the final reconstructed CU output Õ.

B. P2SM Coding Subsystem

The P2SM coding subsystem has two CU level matching modes: 1) horizontally scanned 2D-shape-preserved matching and 2) vertically scanned 2D-shape-preserved matching. In 2D-shape-preserved matching, the reference pixels form a reference string which has exactly the same 2D shape and length

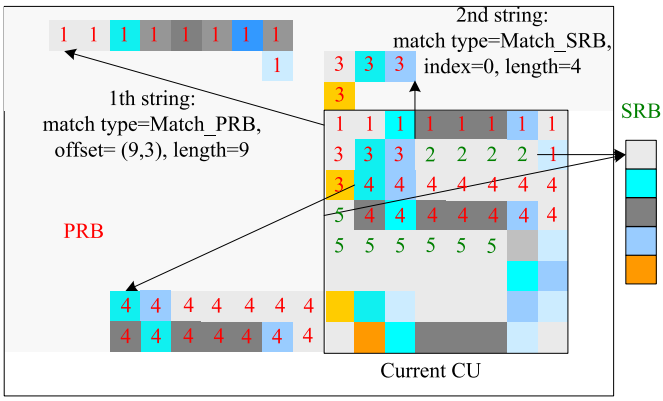

Fig. 3. Example of letter S coded by P2SM.

(number of pixels) as the current string. There are two common types of pixel scanning methods available inside a CU. One is raster-scan where a CU is scanned line by line, either horizontally or vertically, and all lines have the same scan direction. The other is traverse-scan where a CU is also scanned line by line, either horizontally or vertically, but odd lines and even lines have opposite scan direction. Traverse-scan mode is applied in this paper. A CU is pre-coded by both modes and the mode minimizing RD cost is selected as the final P2SM mode for the CU.

In horizontally scanned 2D-shape-preserved matching or vertically scanned 2D-shape-preserved matching of P2SM, a current string being coded can get reference pixels from either PRB or SRB. When a current string gets the reference pixels from PRB, the reference string can be in any valid location inside PRB, and an offset and a length are coded into the bitstream. Offset specifies the position displacement vector between the current string to be encoded and the reference string. On the other hand, when a current string gets the reference pixels from SRB, it actually gets only one single reference pixel color from SRB LUT and all pixels of the entire string have the same color. For an SRB string, an SRB index and a length are coded into the bitstream. If no reference string of at least one pixel is found in PRB or SRB, the current pixel is coded directly into the bitstream as an unmatched pixel preceded by a special SRB index. Each pixel in the current string and the corresponding pixel in the reference string have exactly the same color value in lossless P2SM, but may have slightly different color value in lossy P2SM. The color value difference, i.e., prediction residual is not coded in P2SM.

To reduce encoding complexity, the searching range of PRB strings is limited to either 2 coding tree units (current and left CTUs) or 4 CTUs (current and left three CTUs).

Thus, a CU coded by the P2SM technique has three match types: Match_PRB (PRB string), Match_SRB (SRB string), Match_NONE (unmatched pixel). An example of letter S coded by P2SM technique is shown in Fig. 3. The size of the current CU is 8×8. Horizontally scanned 2D-shape-preserved matching mode with traverse-scan pixel scanning method is used in the example. The following is five cases of PRB strings or SRB strings in the example.

TABLE I
CU LEVEL SYNTAX OF P2SM CODING

```
cu_p2sm_coding (cu_width, cu_height) {
  scan_mode_flag
  strScanPos = 0
  while(strScanPos < cu_width * cu_height) {
    match_type
    if(match_type = = Match_PRB) {
      offset
      length
    }
    else{
    index
    if(index ! = numSRBcolor)
      length
    else{
      length = 1
      for(cIdx = 0; cIdx < numComps; cIdx++)
        unmatched_pixel_val
      }
    }
    strScanPos + = length
  }
}
```

The 1st string marked with red "1" shown in Fig. 3 is a 9-pixel PRB string. Its reference string in PRB with offset (9, 3) has the same horizontal traverse-scan 2D shape and length of 9 pixels. The match type (Match_PRB), the offset (9, 3), and the length (9) will be coded into the bitstream to reconstruct the 1st string of the CU in P2SM decoder. The 2nd string marked with green "2" shown in Fig. 3 is a 4-pixel SRB string. Its reference pixels are all the 1st SRB pixel color duplicated four times. The match type (Match_SRB), the SRB index (0), and the length (4) will be coded into the bitstream to reconstruct the 2nd string of the CU in P2SM decoder. The 3nd string and the 4th string are PRB strings, and the 5th string is an SRB string.

C. P2SM Syntax and Semantics

Table I is CU level syntax of P2SM.

In Table I, cu_width and cu_height are the width and height, respectively, of the current CU. Match_PRB specifies that the current string being coded is a PRB string. numSRBcolor is the number of SRB colors of the current CU. numComps is the number of color components of a pixel and equal to 3 for YUV or RGB color format. There are mainly six syntax elements, which have the following semantics:

1) *scan_mode_flag* equal to 0 specifies that the horizontally scanned 2D-shape-preserved matching mode is applied to the associated pixels of the current CU. scan_mode_flag equal to 1 specifies that the vertically scanned 2D-shape-preserved matching mode is applied to the associated pixels of the current CU;
2) *match_type* equal to 1 specifies a PRB string in the current CU. match_type equal to 0 specifies either an SRB string or an unmatched pixel in the current CU;
3) *offset* specifies the position displacement vector (offsetX, offsetY) between the current string to be coded and the reference string, where offsetX or offsetY specifies the

TABLE II
MATCHING PATTERN DIFFERENCES BETWEEN THREE TECHNIQUES

	P2SM	IBC	PLT
Shape	String	Block	String
Size	Flexible	Fixed	Flexible
Location	Anywhere in reference buffer	Anywhere in reference buffer	Inside the same CU

horizontal or vertical offset between the first pixel of current string to be encoded or decoded and the first pixel of reference string in the PRB;

4) *length* specifies the length of a PRB string or an SRB string;
5) *index* specifies the SRB index for an SRB string or indicates that an unmatched pixel follows; and
6) *unmatched_pixel_val* specifies a color component of an unmatched pixel.

D. Relations of P2SM, IBC, PLT, and Traditional String Matching

P2SM, IBC, PLT, and traditional string matching have one thing in common: searching for matching patterns as reference pixels to code the current pixels. The differences between P2SM, IBC, and PLT are in the shape, size, and location of the matching patterns as shown in Table II.

It can be seen that IBC is actually a special case of P2SM that each CU has only one string equivalent to a prediction unit (PU) of 2Nx2N case in IBC or two equal-sized strings equivalent to two PUs of Nx2N (or Nx2N) case in IBC. PLT is also a special case of P2SM that all reference matching strings are in the same CU as the current strings being coded, thus every pixel in the CU is mapped to an index of a palette and pixel strings are equivalent to index strings. The major differences between P2SM and traditional string matching are that SRB is used and string matching parameter coding is optimized in P2SM.

IV. TECHNICAL DETAILS OF P2SM

A. Hash-Table Based PRB String Searching Strategy

The coding performance of P2SM heavily depends on the PRB searching range. The larger the searching range is, the better the coding performance can achieve, but also the longer the searching time is and the more the computing power requires. Hash-table is an effective tool to speed up reference string searching in a large searching range. A hash-table chains all pixel coordinates with the same hash values together to provide a limited candidate set of pixel coordinates for starting pixels of all potential reference strings.

The role of a hash-table is to quickly find the first matching reference pixel in the PRB searching range by table-look-up. First of all, we need to define a pixel-order for all pixels in the PRB searching range. Naturally, we use the order defined in the horizontally scanned string matching mode, that is, all pixels are ordered one CTU followed by another CTU in CTU coding order, and within a CTU, horizontal raster scanning is applied to order pixels. A hash-table actually has two parts: hash-head and hash-body. In the hash-head, the first coordinates of a hash chain is stored. In the hash-body, all coordinates of the pixels with the same hash-value are chained together one-by-one following the defined horizontally scanned pixel-order. Finally, $(-1, -1)$ is stored in hash-body to indicate the end of each chain.

For any current pixel being coded, the hash-value of the current pixel is calculated. For a pixel $\mathrm{P} = (\mathrm{Y}, \mathrm{U}, \mathrm{V})$ or (G, B, R), where Y, U, V (or G, B, R) are three 8-bit color components of P, a 12-bit hash_value is computed. For lossy coding, hash_value is computed by concatenating 4-bit MSB of Y, U, V (or G, B, R) to obtain a 12-bit value using (1a) or (1b). For lossless coding, hash_value is computed by CRC-12 algorithm [38] to obtain a 12-bit value

$$\text{hash\_value} = ((\mathrm{Y} \;\&\; 0\text{xf0}) \ll 4) | (\mathrm{U} \;\&\; 0\text{xf0}) | (\mathrm{V} \gg 4) \quad \text{(1a)}$$

$$\text{hash\_value} = ((\mathrm{G} \;\&\; 0\text{xf0}) \ll 4) | (\mathrm{B} \;\&\; 0\text{xf0}) | (\mathrm{R} \gg 4). \quad \text{(1b)}$$

A hash-table is organized into many hash-chains. Each hash-chain starts from an entry of the hash-head, whose entries are all initialized to the values of $(-1, -1)$. The hash-table and its hash-chains are built in the following steps.

1) Given a pixel $\mathrm{P}(i, j)$ with coordinates (i, j), its hash value hash_value (i, j) is computed.
2) The value at address hash_value(i, j) of the hash-head is written into the location (i, j) of the hash-body.
3) The value (i, j) is written into the hash-head entry at address hash_value(i, j) to overwrite the previous value.

For example, if the pixel at location (0, 0) has hash-value 1743, then the value at address 1743 of the hash-head initialized to $(-1, -1)$ is written into the location (0, 0) of the hash-body, and (0, 0) is written into the hash-head entry at address 1743 to overwrite the previous value $(-1, -1)$. Next, if the pixel at location (1, 0) has the same hash-value 1743, then the value (0, 0) at address 1743 of the hash-head is written into the location (1, 0) of the hash-body, and (1, 0) is written into the hash-head entry at address 1743 to overwrite the previous value (0, 0).

The details of the hash based PRB string searching algorithm is illustrated in Fig. 4, which has three hash chain examples, as follows.

1) Hash chain marked with magenta color squares connected by blue arrowed lines is a chain of 4 coordinates with the same pixel hash value 1743. The first coordinates of pixel with hash value 1743 is (63, 1) which is stored in address 1743 of the hash-head. The next three coordinates are (63, 0), (1, 0), (0, 0), which are stored in locations (63, 1), (63, 0), (1, 0), respectively, of the hash-body. Finally, $(-1, -1)$ is stored in location (0, 0) of the hash-body to indicate the end of the hash chain.
2) Hash chain marked with purple color squares connected by green arrowed lines is a chain of 4 coordinates with the same pixel hash value 2556. The first coordinates of pixel with hash value 2556 is (65, 1) which is stored in address 2556 of the hash-head. The next three coordinates are (64, 1), (65, 0), (64, 0), which are stored in locations (65, 1), (64, 1), (65, 0), respectively, of the hash-body. Finally,

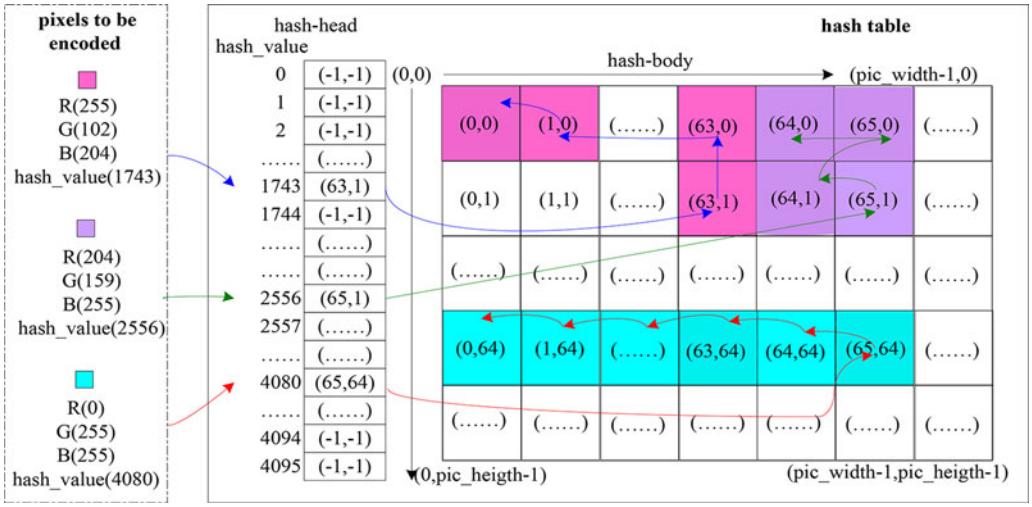

Fig. 4. Hash-table-based PRB string searching.

$(-1, -1)$ is stored in location (64, 0) of the hash-body to indicate the end of the hash chain.

3) Hash chain marked with cyan color squares connected by red arrowed lines is a chain of 66 coordinates with the same pixel hash value 4080. The first coordinates of pixel with hash value 4080 is (65, 64) which is stored in address 4080 of the hash-head. The next sixty-five coordinates are (64, 64), (63, 64), $(\ldots\ldots)$, (1, 64), (0, 64), which are stored in locations (65, 64), (64, 64), $(\ldots\ldots)$, (2, 64), (1, 64), respectively, of the hash-body. Finally, $(-1, -1)$ is stored in location (0, 64) of the hash-body to indicate the end of the hash chain.

If a current pixel to be coded has color values of (G, B, R) equal to (102, 204, 255) and corresponding hash_value of 1743 for lossy coding, then, hash-table based PRB string searching begins at coordinates (63, 1) obtained by access to the hash-head with the address 1743. The coordinates (63, 1) are the first entry in the hash chain with the hash value of 1743 and give the first candidate for starting pixels of potential reference PRB strings. Then, coordinates (63, 0) are obtained by access to the hash-body with address, i.e., coordinates, (63, 1). The coordinates (63, 0) are the second entry in the hash chain and give the second candidate for starting pixels of potential reference PRB strings. In the same way, coordinates (1, 0) and (0, 0) are obtained by access to the hash-body with addresses, i.e., coordinates, (63, 0) and (1, 0). The coordinates (1, 0) and (0, 0) are the third and fourth entries in the hash chain and give the third and fourth candidates for starting pixels of potential reference PRB strings. Finally, $(-1, -1)$ are obtained by access to the hash-body with address, i.e., coordinates, (0, 0) and indicate the end of the hash chain with the hash value of 1743.

From the above examples, it is obvious that a hash chain of all coordinates with the same pixel hash value provides a small and efficient candidate set of reference pixels for starting pixels of potential reference PRB strings. In this way, a hash-table can significantly speed-up the searching process of optimal PRB strings.

TABLE III
NOTATIONS USED IN THIS SUBSECTION

Notations	Descriptions
$P_{m,n}$	pixel at coordinates ($P_{m,n}$) of a picture
com($P_{m,n}$)	sample value of a component for $P_{m,n}$. *com* is one of Y, U, V, R, G, B
curS(*m*, *n*, *l*)	current string starting at coordinates (*m*, *n*) and having length *l*
refSP(*i*, *j*, *l*)	reference PRB string starting at coordinates (*i*, *j*) and having length *l*
refSB(*index*, *l*)	reference SRB string with index *index* and length *l*
P_k *curS*(*m*, *n*,*l*)	the *k*-th pixel of *curS*(*m*, *n*, *l*), $1 \le k \le l$
P_k *refSP*(*i*,*j*,*l*)	the *k*-th pixel of *refSP*(*i*, *j*, *l*), $1 \le k \le l$
E_k	maximum of three absolute differences calculated from three color components of P_k *curS*(*m*, *n*, *k*) and P_k *refS*(*i*, *j*,*k*), as given by (2)
Bits(*se*)	number of bits to code syntax element *se*
Dist(*s*1, *s*2)	sum of squared error (SSE) between strings *s*1 and *s*2
$E_{\max}$	a predetermined error threshold for E_k
λ	a defined weighting factor[1]

B. Optimal Reference String Selection

Table III presents notations used in this subsection.

For a picture with YUV format, E_k in Table III is calculated using (2). For a picture with RGB format, E_k is calculated using (2) with Y, U, V replaced by G, B, R respectively

$$\begin{aligned} E_k = \max(&|Y(P_k curS(m,n,k) - Y(P_k refSP(i,j,k))|, \\ &|U(P_k curS(m,n,k) - U(P_k refSP(i,j,k))|, \\ &|V(P_k curS(m,n,k) - V(P_k refSP(i,j,k))|). \end{aligned} \tag{2}$$

$E_{\max}$ in Table III is 0, 7, 11, 16, and 24 for QP equal to 0, 22, 27, 32, and 37, respectively.

In P2SM encoder, the optimal reference string selection from PRB or SRB for a starting pixel $P_{m,n}$ being coded is performed by the following steps.

Step 1. Searching of the optimal reference PRB string is performed. Hash table provides a limited candidate set of pixel coordinates for starting pixels of all potential reference PRB

TABLE IV
OPTIMAL REFERENCE PRB STRING SEARCHING SUBSTEPS

Input	$P_{m,n}$
Output	*opt_prbLength*, *opt_offsetX*, *opt_offsetY*, *minAvgRDcost_PRB* for the optimal reference PRB string
Substep 1	Calculating the hash value of $P_{m,n}$: *hash_value*
Substep 2	Initialization: *opt_prbLength* = 0; *opt_offsetX* = −1; *opt_offsetY* = −1; *minAvgRDcost_PRB* = MAX_DOUBLE; *i* = 0

Substep 3	Using the hash chain $(x_0, x_0,)$, $(x_1, x_1,)$ . . . . , (−1,−1) with *hash_value* to find the optimal reference PRB string:
	while(x_i != −1 && y_i != −1){
	prbLength = 0; *offsetX* = *m*− x_i; *offsetY* = *n*− y_i; *k* = 1
	while($E_k \le E_{\max}$ && $p_k refSP(i, j, k)$) is in the searching range)
	prbLength ++; *k*++
	Computing *avgRDcost_PRB* using (3)
	if(*avgRDcost_PRB* < *minAvgRDcost_PRB*){
	opt_prbLength = *prbLength*;
	opt_offsetX = *offsetX*;
	opt_offsetY = *offsetY*;
	minAvgRDcost_PRB = *avgRDcost_PRB*
	}
	i++;
	}

strings. From these candidates, the optimal reference PRB string is selected based on average RD cost evaluation *avgRDcost_PRB*. For a reference PRB string with offset of (*offsetX*, *offsetY*) and length of *prbLength*, *avgRDcost_PRB* is calculated using (3)

$$\begin{aligned} &avgRDcost\_PRB \\ &= (\lambda(Bits(\textit{offsetX}) + Bits(\textit{offsetY}) \\ &\quad + Bits(prbLength)) \\ &\quad + Dist(curS(m, n, prbLength), \\ &\qquad refSP(m - \textit{offsetX}, n - \textit{offsetY}, \\ &\qquad prbLength)))/prbLength. \end{aligned} \tag{3}$$

Table IV shows the optimal reference PRB string searching substeps. As a result, *opt_prbLength*, *opt_offsetX*, *opt_offsetY*, *minAvgRDcost_PRB* for the optimal reference PRB string are obtained from these substeps.

Step 2. Searching of the optimal reference SRB string is performed. Since a reference SRB string is simply duplication of an SRB pixel color, searching for the optimal (i.e., longest) reference SRB string is straightforward. If such a reference SRB string with an index of *index* and a length of *srbLength* is found in SRB, the average RD cost for the SRB string is calculated using (4)

$$\begin{aligned} avgRDcost\_SRB = &(\lambda\,(Bits(index) + Bits(srbLength)) \\ &+ Dist(curS(m, n, srbLength), \\ &\quad refSB(index, srbLength)))/srbLength. \end{aligned} \tag{4}$$

Thus, *srbLength*, *index*, *avgRDcost_SRB* for the optimal reference SRB string are obtained.

Step 3. Either the optimal reference PRB string or the optimal reference SRB string is selected as the final optimal reference string based on average RD cost evaluation, i.e., if *minAvgRDcost_PRB* is less than *avgRDcost_SRB*, then the optimal reference PRB string is selected, otherwise, the optimal reference SRB string is selected.

Step 4. If neither PRB string nor SRB string of at least one pixel is found (i.e., *srbLength* equal to 0 and *prbLength* equal to 0) for the current $P_{m,n}$ being coded, then the pixel $P_{m,n}$ itself is directly coded as an unmatched pixel.

TABLE V
SIX REGIONS DEFINITION FOR TWO SCAN MODES

	Horizontal scan		Vertical scan	
Region	offsetX	offsetY	offsetX	offsetY
Above	0	1	1	0
Above Ext	0	>1	>1	0
Left && Left Ext	!= 0	0	0	!= 0
x < 0 && y > 0	<0	>0	>0	<0
x > 0 && y > 0	>0	>0	>0	>0
x > 0 && y < 0	>0	<0	<0	>0

C. Offset Syntax Coding Optimization

To improve coding performance, more frequently occurred values of offset should be assigned shorter binary code. Hence, the statistic distribution of offset should be well investigated and understood.

A version of P2SM implementation [32] based on HM-16.4+SCM-4.0 software is used in an experiment to investigate the statistic distribution of offset. In the experiment, the searching range of IBC is full frame and the searching range of P2SM is current CTU and left CTU. Lossy coding with QP equal to 22, 27, 32, and 37 is used in the experiment. All frames of twenty-six HEVC SCC CTC sequences [37] are used in the simulation.

The value range of offset is divided into six regions as shown in Table V. The offset frequency distribution histogram across the six regions is shown in Fig. 5 for two scan modes.

From Fig. 5, we observed two obvious statistic characteristics of offset. One is that one single offset value of Above occurs much more frequently than any other offset values in both horizontal and vertical scan modes. The other is that for other five regions, offset frequency distribution in horizontal scan mode is quite different from that in vertical scan mode, thus, different coding methods for offset should be applied to different scan modes.

Besides these observations, due to the coding order of CTUs and CUs, and traverse-scan coding order of pixels inside a CU, offsetX and offsetY have the following correlations.

1) In horizontal scan mode, when offsetY is equal to 0, offsetX is always greater than 0 if the starting pixel of the current string being coded is in an even row; when offsetY is less than 0, offsetX is always greater than 0.
2) In vertical scan mode, when offsetY is equal to 0, offsetX is always greater than 0; when offsetY is less than 0, offsetX is always greater than or equal to 0 if the starting pixel of the current string being coded is in an odd column,

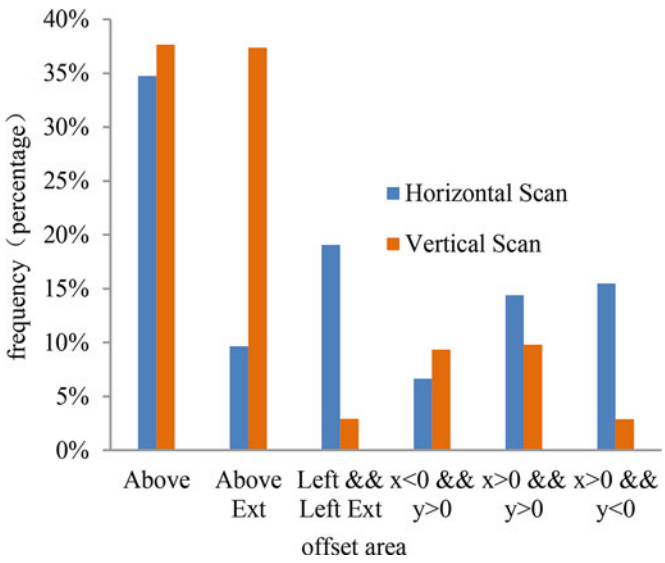

Fig. 5. Offset frequency distribution across six regions.

TABLE VI
SYNTAX OF OPTIMIZED OFFSET CODING

p2sm_offset_coding(){	Descriptor
above_string_flag	**ae(v)**
if(!above_string_flag){	
offsetY_zero_flag	**ae(v)**
if(!offsetY_zero_flag){	
if(!scan_mode_flag && isOddRow)	
offsetX_sign	**u(1)**
offsetX_abs_minus1	**ae(v)**
}	
else{	
offsetY_sign	**u(1)**
offsetY_abs_minus1	**ae(v)**
if(offsetY_sign){	
if(scan_mode_flag && isOddColumn)	
offsetX_abs	**ae(v)**
else	
offsetX_abs_minus1	**ae(v)**
}	
else{	
offsetX_zero_flag	**ae(v)**
if (offsetX_zero_flag){	
offsetX_sign	**u(1)**
offsetX_abs_minus1	**ae(v)**
}	
}	
}	
}	
}	

or is always greater than 0 if the starting pixel of the current string being coded is in an even column.

Based on these observations and correlations, an offset coding optimization scheme is proposed and the syntax of the optimized offset coding is shown in Table VI.

In Table VI, the variable isOddRow equal to 1 specifies the starting pixel of the current string being coded is in an odd row in horizontal scan mode. The variable isOddColumn equal to 1 specifies the starting pixel of the current string being coded is in an odd column in vertical scan mode. ae(v) specifies that a syntax element is context-adaptive arithmetic entropy-coded using variable bits. u(1) specifies that a syntax element is coded as an unsigned integer using 1 bit.

The semantics of the syntax elements shown in Table VI are as follows:

1) *above_string_flag* equal to 1 specifies (offsetX, offsetY) equal to (0, 1) when scan_mode_flag equal to 0 and (offsetX, offsetY) equal to (1, 0) when scan_mode_flag equal to 1. above_string_flag equal to 0 specifies (offsetX, offsetY) not equal to (0, 1) when scan_mode_flag equal to 0 and (offsetX, offsetY) not equal to (1, 0) when scan_mode_flag equal to 1;
2) *offsetY_zero_flag* equal to 0 specifies the value of an offsetY equal to 0. offsetY_zero_flag equal to 1 specifies the value of an offsetY not equal to 0;
3) *offsetX_zero_flag* equal to 0 specifies the value of an offsetX equal to 0. offsetX_zero_flag equal to 1 specifies the value of an offsetX not equal to 0;
4) *offsetY_sign* equal to 0 specifies the value of an offsetY has a positive value. offsetY_sign equal to 1 specifies the value of an offsetY has a negative value;
5) *offsetX_sign* equal to 0 specifies the value of an offsetX has a positive value. offsetX_sign equal to 1 specifies the value of an offsetX has a negative value;
6) *offsetY_abs_minus1* plus 1 specifies the absolute value of an offsetY;
7) *offsetX_abs_minus1* plus 1 specifies the absolute value of an offsetX; and
8) *offsetX_abs* specifies the absolute value of an offsetX.

D. Memory Access Bandwidth and CABAC Throughput

The worst case per-pixel memory access bandwidth (denoted by P) [36] with respect to various memory access patterns of m by n pixels is measured by using (5)

$$P = \frac{\left\lceil \frac{m-1+M+L-1}{m} \right\rceil \cdot \left\lceil \frac{n-1+N+L-1}{n} \right\rceil \cdot m \cdot n}{M \cdot N} \tag{5}$$

where $M \times N$ denotes the size of a target block, $m \times n$ represents the size of a given memory access pattern, and L is the interpolation filter length.

In P2SM decoding, the worst case memory access bandwidth occurs when all strings in a CU are one pixel PRB strings. In this case, M is equal to 1 and N is equal to 1. Also, in P2SM, L is equal to 1 because no interpolation filtering is used. Thus, P is equal to $m \times n$. P is usually larger in P2SM than in IBC and traditional inter-prediction. For example, when $m \times n = 4 \times 4$, P is 16 in P2SM, is 3 in IBC for 8×4 or 4×8 prediction unit (PU), and is 10 in traditional inter-prediction for 8×4 or 4×8 PU of 8-tap interpolation filter. Two solutions can be used to address the worst case memory access bandwidth issue:

1) using SRAM to implement PRB. A SRAM access takes one clock cycle. Therefore, as long as SRAM clock rate is designed to be greater than pixel access rate, memory access bandwidth is not a bottleneck in IC implementation of P2SM decoder; and

TABLE VII
CONTEXT-MODEL CODED BINS PER 8 × 8 CU IN P2SM

	Syntax Elements	Number
Mode Flag	cu_transquant_bypass_flag*	1
	cu_skip_flag*	1
	palette_mode_flag*	1
	p2sm_mode_flag	1
	scan_mode_flag#	1
	srb_color_present_flag	1
	unmatched_pixel_present_flag	1
Partition	N/A	
String Parameter	pixel_string_flag#	64
	above_string_flag#	64
	offsetX_zero_flag#	64
	offsetY_zero_flag#	64
	length#	64
Residual	N/A	
Number of Context-coded Bins		327

*The syntax element is specified in HEVC Screen Content Coding: Draft 4 [4].
The syntax element is specified in Table I and Table VI.

2) limiting the size of a PRB string. If the minimum size of a PRB string is 32 pixels, then P2SM has almost the same worst case memory access bandwidth as IBC having the minimum size PU of 8 × 4 or 4 × 8 pixels.

A typical P2SM decoder design usually uses both solutions. A small portion of PRB, e.g., the current CTU and left CTU, is implemented in SRAM and outside of the small portion of PRB, the size of PRB strings is at least 32 pixels.

In P2SM coding, CABAC throughput measured by counting the number of context-model coded bins per 8 × 8 CU is shown in Table VII.

Other syntax elements have the following semantics:

1) *p2sm_mode_flag* equal to 1 or 0 specifies that the current CU is coded by P2SM or not, respectively;
2) *srb_color_present_flag* equal to 1 specifies that the current CU has at least one SRB string, otherwise, srb_color_present_flag equal to 0 specifies that there is no SRB string in the current CU; and
3) *unmatched_pixel_present_flag* equal to 1 specifies the current CU has at least one unmatched pixel, otherwise, unmatched_pixel_present_flag equal to 0 specifies that there is no unmatched pixel in the current CU.

On the other hand, the number of context-model coded bins per 8 × 8 CU in IBC is 484 [36].

V. EXPERIMENT

A. Description of Test Sequences

Thirteen HEVC SCC CTC sequences [37] shown in Table VIII are used in the experiment. CTC sequences can be classified into four categories: Text and graphics with motion(TGM), Mixed Content (M), Camera Captured (CC), and Animation (A). Each test sequence has a RGB color format version and a YCbCr (i.e., YUV) color format version. Besides the CTC sequences, one typical and commonly seen screen snapshot picture sc_excel [33] is also used in the experiment. Part of sc_excel is shown in Fig. 6.

TABLE VIII
TEST SEQUENCES FROM FOUR CATEGORIES

Resolution	Sequence name	Category	fps	No. of frames
1920 × 1080	sc_flyingGraphics	TGM	60	300
	sc_desktop	TGM	60	600
	sc_console	TGM	60	600
	MissionControlClip3	M	60	600
	EBURainFruits	CC	50	250
	Kimono1	CC	24	120
1280 × 720	sc_web_browsing	TGM	30	300
	sc_map	TGM	60	600
	sc_programming	TGM	60	600
	sc_SlideShow	TGM	20	500
	sc_robot	A	30	300
2560 × 1440	Basketball_Screen	M	60	300
	MissionControlClip2	M	60	300

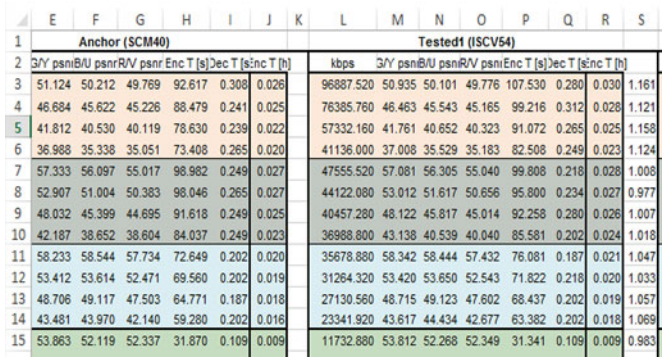

Fig. 6. Part of sc_excel picture.

TABLE IX
TEST CONDITIONS FOR SEARCHING RANGE

Name	Searching range	
	SCM	SCM-P2SM
Test condition 1 (TC1)	full frame for IBC	full frame for IBC and 2CTUs for P2SM
Test condition 2 (TC2)	4CTUs for IBC	4CTUs for both IBC and P2SM

B. Experiment Settings

The experiment evaluates P2SM coding performance gain by comparing the following two coding implementations:

1) HM-16.4+SCM-4.0 reference software (SCM); and
2) HM-16.4+SCM-4.0 integrated with P2SM (SCM-P2SM).

All experimental results were generated under the CTC and configurations defined in [37] for HEVC SCC project. Two test conditions TC1 and TC2 with different searching ranges shown in Table IX are used to evaluate the coding performance. Both TC1 and TC2 are used to evaluate lossy coding performance and only TC1 is used to evaluate lossless coding performance.

TABLE X
LOSSY CODING PERFORMANCE COMPARISON (BD-RATE REDUCTION%) BETWEEN SCM AND SCM-P2SM

			AI			RA			LB		
Test condition	Color format	Test sequences category	G/Y	B/U	R/V	G/Y	B/U	R/V	G/Y	B/U	R/V
TC1	RGB	TGM (1080p & 720p)	−3.8	−4.1	−4.1	−2.5	−2.7	−2.7	−2.0	−2.0	−1.9
		M (1440p & 1080p)	−1.2	−1.7	−1.7	−0.7	−1.0	−1.1	0.1	−0.7	−0.6
		A (720p)	0.0	0.0	0.0	0.0	−0.1	0.0	−0.1	0.1	0.1
		CC (1080p)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	YUV	TGM (1080p & 720p)	−4.2	−4.1	−4.1	−2.4	−2.3	−2.5	−1.6	−1.7	−1.8
		M (1440p & 1080p)	−1.5	−2.0	−2.3	−0.9	−1.4	−1.9	−0.3	−1.2	−1.6
		A (720p)	0.0	0.0	−0.1	−0.1	−0.3	0.1	0.0	0.2	0.0
		CC (1080p)	0.0	0.0	0.0	0.1	0.0	−0.1	−0.1	0.0	0.1
TC2	RGB	TGM (1080p & 720p)	−7.4	−8.0	−7.9	−5.1	−5.4	−5.3	−3.1	−3.3	−3.3
		M (1440p & 1080p)	−2.4	−3.2	−3.2	−1.2	−1.9	−1.9	−0.4	−1.2	−1.1
		A (720p)	0.0	0.0	0.0	0.0	0.1	0.1	0.0	0.0	−0.1
		CC (1080p)	0.0	0.0	0.0	−0.1	0.0	0.0	0.0	0.0	0.0
	YUV	TGM (1080p & 720p)	−7.7	−7.7	−7.9	−5.0	−4.9	−5.0	−2.6	−2.5	−2.8
		M (1440p & 1080p)	−3.2	−4.1	−4.3	−1.8	−2.2	−2.7	−0.6	−1.3	−1.7
		A (720p)	0.0	0.0	0.0	0.0	0.2	0.1	0.0	0.0	0.2
		CC (1080p)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.1

TABLE XI
LOSSLESS CODING PERFORMANCE COMPARISON (BIT-RATE SAVING%) BETWEEN SCM AND SCM-P2SM

			AI				RA				LB			
Test condition	Color format	Test sequences category	total	average	max	min	total	average	max	min	total	average	max	min
TC1	RGB	TGM (1080p & 720p)	−4.8	−5.6	−14.9	−0.9	−2.8	−3.5	−10.2	−0.4	−2.6	−2.8	−6.6	−0.2
		M (1440p & 1080p)	−0.8	−0.9	−1.6	−0.2	−0.1	−0.1	−0.2	−0.1	−0.1	−0.1	−0.1	−0.1
		A (720p)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		CC (1080p)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	YUV	TGM (1080p & 720p)	−5.2	−6.0	−15.4	−1.0	−2.8	−3.9	−10.5	−0.3	−2.6	−3.1	−7.5	−0.2
		M (1440p & 1080p)	−0.9	−1.0	−1.7	−0.2	−0.1	−0.1	−0.2	−0.1	−0.1	−0.1	−0.1	−0.1
		A (720p)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
		CC (1080p)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

To evaluate overall coding performance improvement for CTC sequences, BD-rate metric [30], [31] is used for lossy coding and bit-rate saving metric is used for lossless coding. In lossy coding, for each color format and category, an average BD-rate reduction is calculated for three color components G/Y, B/U, and R/V separately. Lossy coding uses four QPs of 22, 27, 32, and 37. In lossless coding, for each color format and category, four bit-rate saving values, total, average, minimum, and maximum, are calculated. Three configurations of All Intra (AI), Random Access (RA), and Low delay B (LB) are tested in the experiment. To evaluate encoder and decoder complexity, encoding and decoding software runtime are also compared.

To evaluate coding performance improvement of the screen snapshot picture sc_excel, Y-PSNR-bps curves and BD-rate metric are used for lossy coding. The Y-PSNR-bps curves show the PSNR and bit-rate values for Y component of four QPs. BD-rate is also calculated for Y component.

C. Experimental Results

R-bps curves and BD-rate reduction percentage of sc_excel for AI configuration. For CTC sequences, lossy coding performance comparison (BD-rate reduction percentage in negative numbers) between SCM and SCM-P2SM is shown in the first 8 rows of Table X for TC1 and in the second 8 rows of Table X for TC2. Each row of data corresponds to a combination of one color format and one category. There are total 8 combinations, each of which contains from 1 to 7 sequences. Lossless coding performance comparison (bit-rate saving percentage in negative numbers) is shown in Table XI for TC1. Table XII shows the encoding runtime ratio and decoding runtime ratio. Fig. 7 shows the Y-PSN.

The experimental results can be summarized as follows.

1) For CTC sequences, the overall coding performance is notably improved. For YUV TGM category, in lossy coding case, P2SM can achieve up to 4.2% for AI, 2.4% for RA, 1.6% for LB average BD-rate reduction in TC1, and can achieve up to 7.7% for AI, 5.0% for RA, 2.6% for LB average BD-rate reduction in TC2. In lossless coding case, P2SM can achieve up to 15.4 for AI, 10.5% for RA, 7.5% for LB maximum bit-rate saving and 6.0% for AI, 3.9% for RA, 3.1% for LB average bit-rate saving.
2) For a commonly seen screen snapshot sc_excel, the coding performance is significantly improved. In lossy coding case, as shown in Fig. 7(b) for TC2, AI configuration, and QP of 22, both bitrate and Y-PSNR are significantly

TABLE XII
COMPLEXITY COMPARISON BETWEEN SCM AND SCM-P2SM

	Lossy coding						Lossless coding		
Test condition	TC1			TC2			TC1		
Configuration	AI	RA	LB	AI	RA	LB	AI	RA	LB
SCM-P2SM/SCM encoding time ratio [%]	105	106	103	106	104	100	118	106	104
SCM-P2SM/SCM decoding time ratio [%]	99	102	103	98	101	100	100	103	102

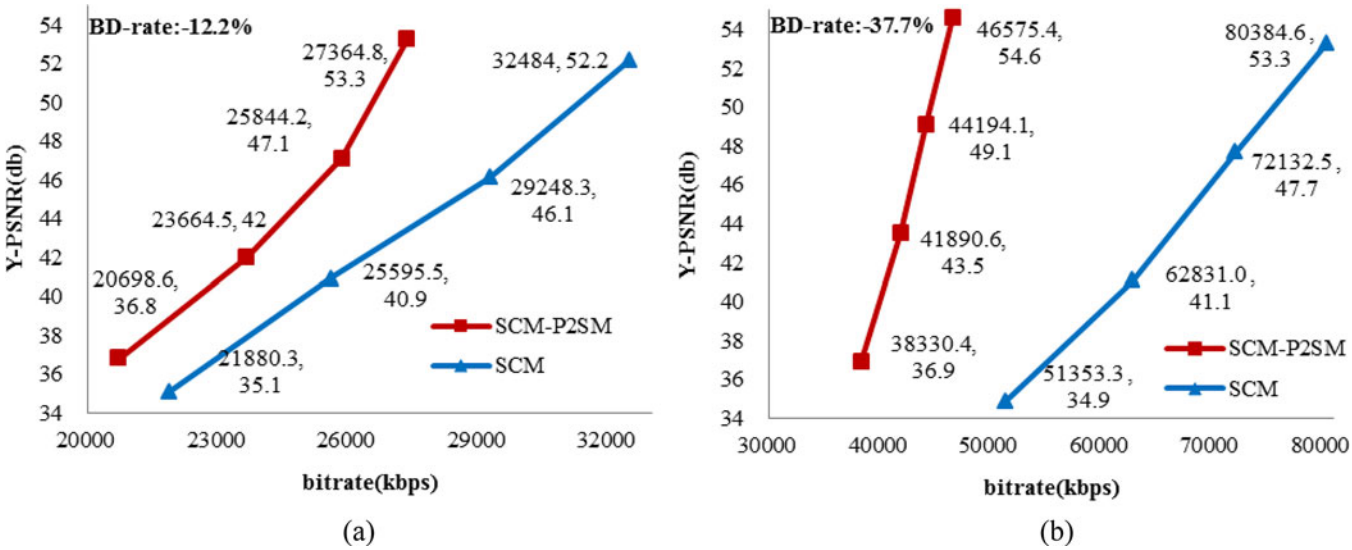

Fig. 7. Y-PSNR-bps curves and BD-rate reduction percentage of sc_excel for AI configuration. (a) TC1. (b) TC2.

improved from 80384.6 kbps and 53.3 dB in SCM to 46575.4 kbps and 54.6 dB in SCM-P2SM. Fig. 7 also shows that in terms of BD-rate reduction percentage for Y component, P2SM can achieve up to 12.2% in TC1 and up to 37.7% in TC2.

3) The coding performance improvement by P2SM depends on the contents. As shown in Tables X and XI, P2SM has good BD-rate reduction for TGM and M categories in all configurations, but no coding performance improvement for CC and A categories.
4) As shown in Table XII, in lossy coding case, encoding runtime increase of P2SM is about 5%–6% for AI, 4%–6% for RA, 0%–3% for LB. In lossless coding case, encoding runtime increase is about 18% for AI, 6% for RA, 4% for LB. Therefore, encoding runtime increase is very minor. Table XII also shows that P2SM decoding runtime increase is negligible.

VI. CONCLUSION AND FUTURE WORK

In order to exploit both local and non-local repeated identical patterns with a variety of sizes and/or shapes in a wide range of generic screen content, this paper presents a generic, flexible, and uniform P2SM technique for SCC. Both PRB and SRB are used for string-matching. The resulting bitstream is a combination of PRB string coding parameters, SRB string coding parameters, and unmatched pixel mixed on a string-by-string basis. Some technical details of P2SM are also discussed in this paper. Experimental results show that P2SM improves coding performance significantly in both lossy coding case and lossless coding case at very low additional encoding and decoding complexity. Future work includes: 1) Optimizing P2SM technique to improve coding performance and reduce coding complexity further and implementing rate-control in P2SM enhanced coding system; 2) Integrating P2SM technique with other video coding standards such as AVS2; 3) Achieving idempotent (reloss-free) [39] coding of P2SM in repetitive (multi-generation) compression.

REFERENCES

[1] T. Lin, K. Zhou, and S. Wang, "Cloudlet-screen computing: A client-server architecture with top graphics performance", *Int. J. Ad Hoc Ubiquitous Comput.*, vol. 13, no. 2, pp. 96–108, Jun. 2013.
[2] 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, Apr. 2011.
[3] T. Lin and S. Wang, "Cloudlet-screen computing: A multi-core-based, cloud-computing-oriented, traditional-computing-compatible parallel computing paradigm for the masses," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jul. 2009, pp. 1805–1808.
[4] *High Efficiency Video Coding (HEVC) Screen Content Coding: Draft 4*, JCTVC-U1005, Jun. 2015.

[5] G. J. Sullivan, J. R. 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, Sep. 2012.

[6] *Intra Motion Compensation With Variable Length Intra MV Coding*, JCTVC-N0206, Jul. 2013.

[7] *Intra Motion Compensation With 2-D MVs*, JCTVC-N0256, Jul. 2013.

[8] W. Zhu *et al.*, "Hash-based block matching for screen content coding," *IEEE Trans. Multimedia*, vol. 17, no. 7, pp. 935–944, Jul. 2015.

[9] W. Zhu, W. Ding, and J. Xu, "Screen content coding based on HEVC framework," *IEEE Trans. Multimedia*, vol. 16, no. 5, pp. 1316–1326, Aug. 2014.

[10] L. Guo *et al.*, "Color palette for screen content coding," in *Proc. IEEE Int. Conf. Image Process.*, Oct. 2013, pp. 5556–5560.

[11] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.

[12] S. Wang and T. Lin, "A unified LZ and hybrid coding for compound image partial-lossless compression," in *Proc. IEEE Int. Conf. Image Signal Process.*, Oct. 2009, pp. 1–5.

[13] S. Wang and T. Lin, "United coding for compound image compression," in *Proc. IEEE Int. Conf. Image Signal Process.*, Oct. 2010, pp. 566–570.

[14] *Intra and Inter Coding Tools for Screen Contents*, JCTVC-E145, Mar. 2011.

[15] S. Wang and T. Lin, "Compound image compression based on unified LZ and hybrid coding," *IET Image Process.*, vol. 7, no. 5, pp. 484–499, May 2013.

[16] S. Wang and T. Lin, "United coding method for compound image compression," *Multimedia Tools Appl.*, vol. 71, no. 3, pp. 1263–1282, 2014.

[17] *CE7: Summary Report for Core Experiment 7 on String Matching for Palette Index Coding*, JCTVC-S0027, Oct. 2014.

[18] *CE3 Test B.3: 2-D Intra String Copy in HEVC SCC*, JCTVC-T0126, Feb. 2015.

[19] *CE1-Related: Palette Coding With Inter-Prediction*, JCTVC-U0064, Jun. 2015.

[20] W. Zhu *et al.*, "Adaptive LZMA-based coding for screen content," in *Proc. IEEE Picture Coding Symp.*, Dec. 2013, pp. 373–376.

[21] B. Li, J. Xu, and F. Wu, "A unified framework of hash-based matching for screen content coding," in *Proc. IEEE Visual Commun. Image Process. Conf.*, Dec. 2014, pp. 530–533.

[22] T. Lin *et al.*, "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.

[23] T. Lin *et al.*, "Arbitrary shape matching for screen content coding," in *Proc. IEEE Picture Coding Symp.*, Dec. 2013, pp. 369–372.

[24] T. Lin, X. Chen, and S. Wang, "Pseudo-2-D-matching based dual-coder architecture for screen contents coding," in *Proc. IEEE Int. Conf. Multimedia Expo*, Jul. 2013, pp. 1–4.

[25] *Improvements on 1D Dictionary Coding*, JCTVC-Q0124, Apr. 2014.

[26] *SCCE4: Results of Subtest 3.3*, JCTVC-R0060, Jul. 2014.

[27] *Non-CE10: Improvement on Coding of ISC Parameters and Comparison to Palette Coding*, JCTVC-S0250, Oct. 2014.

[28] *CE3: Results of Test B.4.2 (Minimum String Length of 20 Pixels) on Intra String Copy*, JCTVC-T0136, Feb. 2015.

[29] *Non-CE3: Improvement on Intra String Copy*, JCTVC-T0139, Feb. 2015.

[30] *Calculation of Average PSNR Differences Between RD-Curves*, ITU-T SG16 Q.6, Doc. VCEG-M33, Apr. 2001.

[31] *Improvements of the BD-PSNR Model*, ITU-T SG16 Q.6, Doc. VCEG-AI11, Jul. 2008.

[32] *Non-CE1: Enhancement to Palette Coding by Palette With Pixel Copy (PPC) Coding*, JCTVC-U0116, Jun. 2015.

[33] *Flexible Coding Tools to Significantly Improve SCC Performance in Cloud and Mobile Computing*, JCTVC-U0186, Jun. 2015.

[34] *Crosscheck of Non-CE1: Enhancement to Palette Coding by Palette With Pixel Copy (PPC) Coding*, JCTVC-U0173, Jun. 2015.

[35] *Cross-Check Report of U0116*, JCTVC-U0189, Jun. 2015.

[36] *JCT-VC AHG Report: Complexity of IBC, Intra Line and Intra String Copy Coding (AHG10)*, JCTVC-S0010, Oct. 2014.

[37] *Common Conditions for Screen Content Coding Tests*, JCTVC-S1015, Oct. 2014.

[38] T. V. Ramabadran and S. S. Gaitonde, "A tutorial on CRC computations," *IEEE Micro*, vol. 8, no. 4, pp. 62–75, Aug. 1988.

[39] T. Lin, "Achieving re-loss-free video coding," *IEEE Signal Process. Lett.*, vol. 16, no. 4, pp. 323–326, Apr. 2009.

Liping Zhao received the B.S. degree in computer science and technology from Hengyang Normal University, Hunan, China, in 2006, the M.S. degree in computer science and technology from Hunan University, Changsha, China, in 2009, and is currently working toward the Ph.D. degree in control science and engineering at the VLSI Laboratory, Tongji University, Shanghai, China.

Her current research interests include the areas of screen content coding, and video coding.

Tao Lin received the B.S. degree from East China Normal University, Shanghai, China, in 1982, and the M.S. and Ph.D. degrees from Tohoku University, Sendai, Japan, in 1985 and 1989, respectively.

From 1988 to 2002, he was with the University of California at Berkeley, Berkeley, CA, USA, as a Postdoctoral Researcher; for Integrated Device Technology, Inc., PMC-Sierra Inc., Cypress Semiconductor Corp., NeoMagic Corp., all located in San Jose, CA, USA. He has been with the VLSI Laboratory, Tongji University, Shanghai, China, since 2003. He has been granted 24 U.S. patents and 14 China patents. His current research interests include cloud-mobile computing, digital signal processing, audiovisual coding, and multimedia SoC design.

Prof. Lin was the recipient of the "Chang Jiang Scholars," the highest honor given by the China Ministry of Education, in 2005.

Kailun Zhou received the B.S. and M.S. degrees from Shanghai Jiaotong University, Shanghai, China, in 2000 and 2003, respectively, and is currently working toward the Ph.D. degree at Tongji University, Shanghai, China.

His current research interests include embedded system design and video coding.

Shuhui Wang received the B.S. and M.S. degrees from the Shandong University of Science and Technology, Jinan, China, in 1995 and 1998, respectively, and the Ph.D. degree from Shanghai Jiaotong University, Shanghai, China, in 2007.

She is currently with the VLSI Laboratory, Tongji University, Shanghai, China. Her current research interests include image and video coding.

Xianyi Chen received the B.S. degree from Harbin Engineering University, Harbin, China, in 2004, the M.S. degree from the Shandong University of Science and Technology, Jinan, China, in 2010, and is currently working toward the the Ph.D. degree at Tongji University, Shanghai, China.

His current research interests include video coding and multimedia processing.