Multimedia Tools and Applications (2023) 82:23693–23708
https://doi.org/10.1007/s11042-023-14673-x

Line-based self-referencing string prediction technique for screen content coding in AVS3

Liping Zhao[1,2] · Qingyang Zhou[3] · Keli Hu[1] · Sheng Feng[1] · Kailun Zhou[3] · Weixing Wang[1] · Tao Lin[3]

Received: 16 November 2021 / Revised: 2 June 2022 / Accepted: 3 February 2023 /
Published online: 28 February 2023

Abstract
String Prediction (SP) is a very efficient screen content coding (SCC) tool. In SP, the self-referencing string plays an important role to improve coding efficiency. But general self-referencing string has the problem of very low pixel copying throughput and is prohibited in the non-self-referencing based SP which has been adopted in the third-generation Audio Video Standard (AVS3). To overcome the problem and bring back the coding gain of self-referencing string, a line-based self-referencing string (LSRS) enabled SP technique is proposed. Moreover, to keep the pixel copying throughput and coding complexity of LSRS enabled SP the same as non-self-referencing based SP, an unbroken-line decomposition algorithm is presented to decompose an LSRS into multiple non-self-referencing strings. In this way, LSRS can be treated in the same way as a non-self-referencing string with the best trade-off between coding efficiency and complexity. Compared with non-self-referencing based SP, using AVS3 reference software HPM, for twelve SCC common test condition YUV test sequences in text and graphics with motion category and mixed content category, the proposed LSRS technique achieves the average Y BD-rate reduction of 0.81% and 0.59% as well as the maximum Y BD-rate reduction of 2.04% and 1.31% for All Intra and Low Delay configurations, respectively, with almost no additional encoding and decoding complexity. *The proposed LSRS enabled SP technique has been adopted in AVS3.*

Keywords Audio-video coding standard · Screen content coding · String prediction · Self-referencing · String decomposition

✉ Tao Lin
lintao@tongji.edu.cn

[1] Department of Computer Science and Engineering, Shaoxing University, Shaoxing 312000, China

[2] Information Technology R&D Innovation Center of Peking University, Shaoxing 312000, China

[3] College of Electronics and Information Engineering, VLSI Lab, Tongji University, Shanghai 200092, China

1 Introduction

With applications using the computer screen as an interface for daily remote human-machine and human-human interactions like online education, teleconferencing, telecommuting and remote entertainment becoming more and more popular, almost all of recent video coding standards have included SCC tools. The video coding standards with SCC tools include High Efficiency Video Coding (HEVC) [13], Versatile Video Coding (VVC) [12], the second-generation Audio Video Standard (AVS2) [28], the third-generation AVS (AVS3) [20], AV1 [20], Essential Video Coding (EVC) [21].

In various standards, the adopted SCC tools are quite different [20]. In VVC, five main SCC tools have been adopted [12]: transform skip residual coding, block-based differential pulse-code modulation (BDPCM), intra block copy (IBC) [21], adaptive color transform (ACT), and palette [9, 11]. In AVS3, also five main SCC tools have been adopted in the draft [20]: IBC, frequency-based intra mode coding (FIMC), implicit selection of transform skip (ISTS), adaptive control of deblocking type (ACDT), and string prediction (SP) [34]. There are several other SCC techniques [2, 17, 18] developed for HEVC-SCC standard.

Unlike IBC applied to a few fixed rectangular shapes, the major merit of SP is to divide a coding unit (CU) into multiple strings to take full advantage of matching patterns with a variety of sizes, shapes, and positions in screen content. SP, also known as string matching (SM) or intra string copy (ISC), has quite a few early versions including macro-block or coding tree unit (CTU)-based 1D SM [5, 6, 14], 2D SM [35], Pseudo 2D SM (P2SM) [24, 25, 31] and universal string matching (USM) [26, 32] implemented in HEVC, USP in AVS2 [27–29] and SP in AVS3 [15, 16, 22, 33, 34].

Pixel copying in SP is more flexible than block copying in IBC, but also brings higher hardware complexity. In SP, all reconstructed pixels of a current string are copied from its reference string. Therefore, the basic operation of SP is to copy pixels from reference positions to current positions continuously. To solve the hardware complexity problem of SP, a low complexity implementation of SP mode is proposed and adopted in AVS3 [16] by introducing a variety of string constraints. These constraints include 1) only traverse-horizontal scanning direction is allowed, 2) SP has the same reference range as IBC, 3) the total number of strings in a CU cannot exceed a quarter of the total number of pixels in the CU, 4) Only strings whose length is divisible by 4 are allowed, 5) Only non-self-referencing string (non-SRS) is allowed and self-referencing string (SRS) is prohibited.

SRS plays an important role in SP to improve coding efficiency but has the problem of very low pixel copying throughput. SRS is a string that at least one pixel of its reference string is located in the current string itself. Because generally the reference pixels as reconstructed pixels in the current string must be copied from their reference positions to their current positions before they are copied again as reference pixels of other pixels, a general string copy implementation has to copy pixels one by one sequentially, resulting in the lowest throughput of only one pixel per copying operation and significant difficulty of real-world SP implementation. Due to the problem, SRS is not allowed in early versions of SP (non-SRS based SP), thus partial coding gain of SP is lost.

To overcome the problem and bring back the coding gain of SPS, a line-based self-referencing string (LSRS) enabled SP technique is proposed in this paper. Detailed analysis reveals that not only a special type of SRS with the reference string is above the current string in horizontal scan mode, does not really have the problem of low pixel copying throughput, but also a majority of SRS coding gain actually comes from this special type of SRS. In this special type of SRS, string copy can always be performed at least line-by-line instead of pixel-by-pixel, thus it is named LSRS. Furthermore, to keep the coding complexity of LSRS the same as non-SRS based SP, decomposition algorithms are presented to convert an LSRS into multiple non-SRSs. In this way, LSRS can be simply treated in the same way as non-SRS with no additional effort and cost.

The major contributions of this paper are as follows.

1) *An LSRS-enabled SP technique is proposed to further improve the coding efficiency of non-SRS based SP.*
2) *To keep the same low complexity implementation of non-SRS based SP, three decomposition algorithms with the different trade-offs between coding efficiency and complexity based on the six principles are presented and analyzed.*
3) *An LSRS-enabled SP with the unbroken-line decomposition algorithm is implemented in AVS3 and has been adopted in AVS3.*

This paper is partially based on AVS documents [15, 33]. The remainder of the paper is organized as follows. Section 2 describes background and motivation. Section 3 elaborates the technical details of the line-based self-referencing string prediction. Experimental results are provided and discussed in Section 4. Compared with the AVS3 reference software HPM "https://gitlab.com/AVS3_Software/hpm/-/tree/HPM-9.1", for the sequences from the AVS3 SCC Common Test Condition (CTC) [19], the proposed line-based self-referencing string prediction technique can achieve a Bjøntegaard delta rate (BD-rate) [1] reduction up to 2.04% with almost no additional encoding and decoding complexity. Conclusions and future work are given in Section 5.

2 Background and motivations

In the non-SRS based SP adopted in the AVS3, the encoder searches for the best reference string in the reference range for a current string. As shown in Fig. 1a, each reference string has two string copy parameters: a string vector SV = (svX, svY) representing the coordinate offset between the current string and the reference string and a string length parameter StrLen representing the number of pixels in the string. If no reference pixel is found for a pixel being coded, the pixel value is written into the bitstream as an unmatched pixel.

As shown in Fig. 1b, an SRS is a string with the last part of the reference string partially overlappings the first part of the current string. Consecutive repeated identical patterns occur frequently in screen content. Thus, SP with SRS enabled can further improve coding efficiency

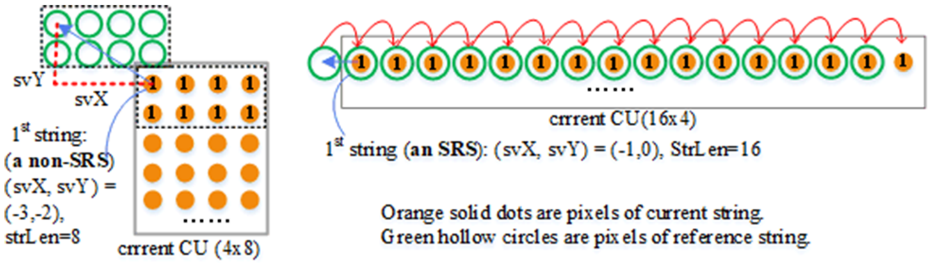
svY
svX
1st string:
(a non-SRS)
(svX, svY) =
(-3,-2),
strLen=8
crrent CU (4x8)
1
......
crrent CU(16x4)
1st string (an SRS): (svX, svY) = (-1,0), StrLen=16
Orange solid dots are pixels of current string.
Green hollow circles are pixels of reference string.

Fig. 1 Examples of non-SRS and SRS. **a** An example of a non-SRS. **b** An example of a pixel-by-pixel copied SRS

significantly. However, due to the low throughput hardware implementation problem of SRS, SRS is not allowed in early versions of SP and partial coding gain of SP is lost. For an SRS, all pixels on the overlap are both reconstructed pixels of themselves and reference pixels of others. Therefore, these pixels have to be copied from their reference positions first before they are used as reference pixels of other pixels to be copied to other positions. To minimize SP hardware implementation cost, string copy is usually implemented using basic and straightforward n-to-n pixel copying logic which inputs n pixels from source and outputs the same n pixels to destination in one clock cycle of operation. Typically, n is at least 4 to achieve pixel copying throughput of at least 4 pixels per operation. It can be seen that SRS requires 1-to-1 pixel copying logic to handle the worst case scenario. As illustrated in Fig. 1b, for an SRS with (svX, svY) = (−1, 0) and StrLen = 16, sixteen pixels of the current string marked by orange solid dots have to be copied one pixel by one pixel sequentially, resulting in the lowest throughput of only one pixel per copying operation.

To solve the problem and regain the coding gain of SRS, characteristics of SRS are investigated. In particular, statistical analysis of SRS string vector distribution reveals that for a majority of SRS in horizontal scan mode, the reference string is in a position above the current string (i.e. $svY < 0$).

The experiment in the analysis uses AVS3 SCC common test condition (CTC) [19] in lossy coding ($QP = 27$) and All Intra configuration. Twelve AVS3 CTC SCC sequences (see X axis of Fig. 2 for the abbreviation of the sequence names) are used in the experiment. From Fig. 2, there are at least the following two findings.

1) For each sequence, the ratio of SRS with $svY < 0$ over all SRS is very high, ranging from 83.9% to 98.8%. The reason is that the repeated identical patterns with a variety of sizes and shapes with $svY < 0$ occur much more frequently than the other offsets [24] for the screen content.
2) The average ratio is as high as 94.8%.

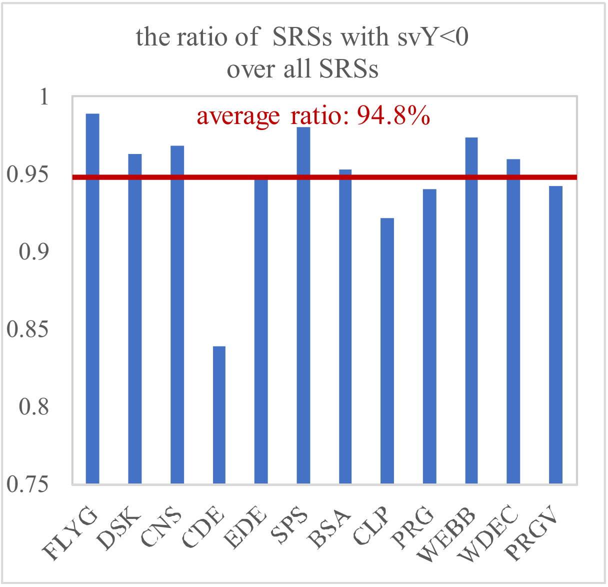

Fig. 2 The ratio of SRSs with $svY < 0$ over all SRSs

Moreover, SRS with $svY < 0$ has a special feature: the reference pixels are always at least one line above the current pixels and can be reconstructed at least one line before reconstruction of the current pixels. Thus, for SRS with $svY < 0$, string copy can be properly managed to have high pixel copying throughput as to be discussed in the next section. Because of the special feature, SRS with $svY < 0$ in horizontal scan mode is named line-based self-referencing string (LSRS).

Motivated by these observations and analysis, it can be expected that LSRS has the potential to regain most of the coding gain lost by prohibiting SRS and also to achieve high pixel copying throughput at the same time.

3 Line-based self-referencing string prediction technique

3.1 LSRS-enabled SP coding architecture and subsystem

Figure 3 shows the AVS3 encoding architecture with LSRS-enable SP included.

The encoding architecture consists of an LSRS-enable SP module in orange boxes and other operational modules such as intra prediction, IBC, FIMC, ISTS, ACDT, inter prediction, transform, and quantization modules. Modules in yellow boxes are other SCC coding tools in AVS3. Modules in green boxes are traditional coding tools in AVS3.

As illustrated in Fig. 3, the input CU is fed to the LSRS-enable SP encoding subsystem and traditional block prediction encoding subsystem including IBC, FIMC, and Intra/Inter modes. Different modes generate different reconstructed pixels and rate-distortion (RD) costs. Then, the one with the best RD cost is selected as the CU's final coding mode by the RD cost-based CU mode selector. After that, the coded bitstream is put into the output bitstream and the reconstructed CU pixels are stored in reconstructed YUV 4:2:0 picture buffer, which is shared by LSRS-enable SP encoding subsystem, and the rest of the encoding system for inter prediction and IBC. LSRS-enable SP has no residual coding including transform, quantization, residual coefficient coding and deblocking steps.

Figure 4 illustrates the LSRS-enabled SP coding subsystem including four functional modules: string prediction module, string copy parameters coding module, string decomposition module, and string copy module. *The main differences between LSRS-enable SP and non-*

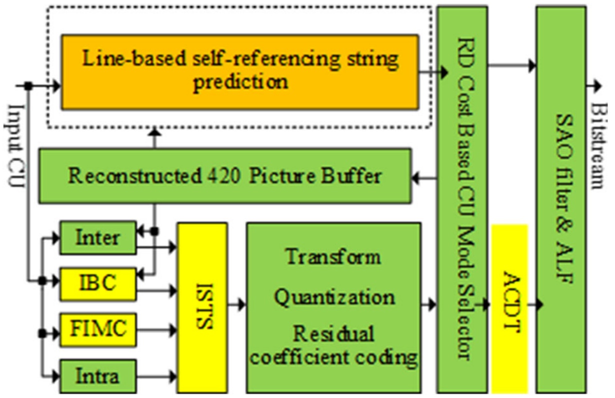
Line-based self-referencing string prediction
Input CU
Reconstructed 420 Picture Buffer
Inter
IBC
FIMC
Intra
ISTS
Transform
Quantization
Residual coefficient coding
RD Cost Based CU Mode Selector
ACDT
SAO filter & ALF
Bitstream

Fig. 3 AVS3 encoding architecture with LSRS-enable SP included

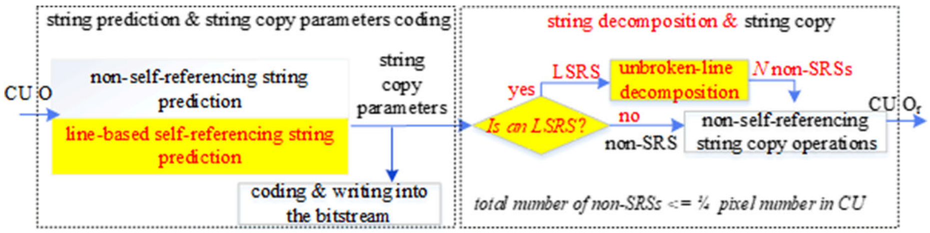
string prediction & string copy parameters coding
CU O
non-self-referencing string prediction
line-based self-referencing string prediction
string copy parameters
coding & writing into the bitstream
string decomposition & string copy
yes
LSRS
unbroken-line decomposition
N non-SRSs
Is an LSRS?
no
non-SRS
non-self-referencing string copy operations
CU O_r
total number of non-SRSs <= ¼ pixel number in CU

Fig. 4 Line-based self-referencing string prediction coding subsystem

SRS based SP are the new line-based self-referencing string prediction in the string prediction module and the new string decomposition module. Both are marked with yellow bounding boxes and red texts.

As shown in Fig. 4, for an input CU O, in the string prediction module, if svY < 0, both LSRS prediction and non-SRS prediction are performed in the encoder, otherwise, only non-SRS prediction is performed. In the string copy parameters coding module, the string copy parameters for both non-SRS and LSRS are coded and written into the bitstream using the same way. In the string decomposition module in both encoder and decoder, an LSRS is decomposed into multiple non-SRSs as described in subsection B. In the string copy module, only non-SRSs need to be processed and copied in the same way as in a non-SRS SP coding system.

In LSRS-enabled SP, after decomposition, the total number of non-SRS in a CU cannot exceed a quarter of the total number of pixels in the CU. The constraint is the same as in non-SRS SP to keep the hardware complexity of LSRS-enabled SP the same as non-SRS SP.

3.2 Decomposition algorithms

For string copy of LSRS to have the same pixel copying throughput and implementation complexity as string copy of non-SRS, an LSRS needs to be decomposed into multiple substrings of non-SRS and the total number of non-SRS after decomposition in a CU must meet the same constraint as in original non-SRS SP. There are at least three algorithms to decompose an LSRS into multiple non-SRSs. Figure 5 illustrates how an LSRS with (svX, svY) = (−3, −2) and StrLen = 20 is decomposed into different *N* non-SRSs using three different ways. In Fig. 5, orange solid dots are pixels of current string and green hollow circles are pixels of the reference string.

1) ***Minimum decomposition***: Each decomposed non-SRS has the maximum length, i.e. if the length is increased by one, the non-SRS becomes an LSRS. In minimum decomposition, the number of decomposed non-SRSs is minimized.

As shown in Fig. 5b, the original LSRS is decomposed into two non-SRSs by the minimum decomposition algorithm. The first non-SRS has (svX, svY) = (−3, −2) and StrLen = 11. Obviously, if the length of the first non-SRS is increased by one, the first non-SRS becomes an LSRS because the last pixel of the extended reference string is located in the current string itself. Actually, the last pixel of the extended reference string and the first pixel of the current string are at the same location. The second non-SRS has (svX, svY) = (−3, −2) and StrLen = 9.

Let (svX, svY), StrLen, and *H* be the string vector, length, and height (i.e. number of lines in the string) of the original LSRS, respectively. |svY| and H always satisfy |svY| $\leq$ H.

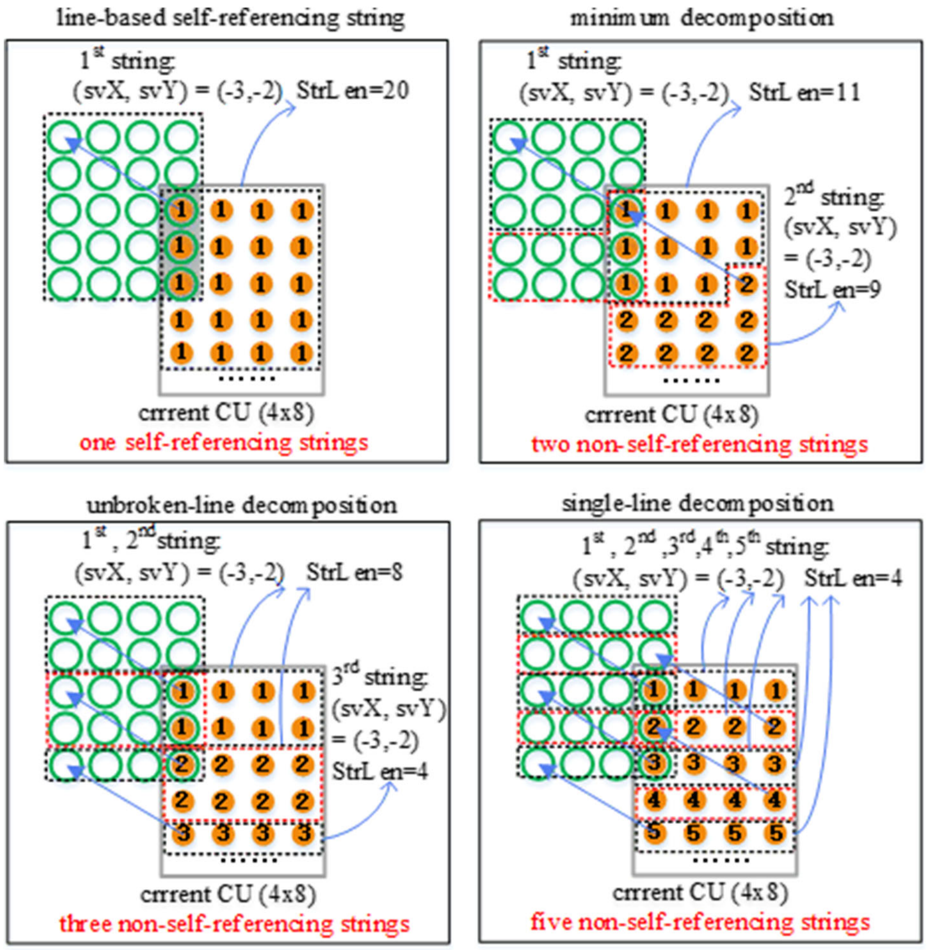
line-based self-referencing string
1st string
(svX, svY) = (-3,-2) StrLen=20
crrent CU (4x8)
one self-referencing strings
minimum decomposition
1st string
(svX, svY) = (-3,-2) StrLen=11
2nd string
(svX, svY) = (-3,-2)
StrLen=9
crrent CU (4x8)
two non-self-referencing strings
unbroken-line decomposition
1st, 2nd string
(svX, svY) = (-3,-2) StrLen=8
3rd string
(svX, svY) = (-3,-2)
StrLen=4
crrent CU (4x8)
three non-self-referencing strings
single-line decomposition
1st, 2nd, 3rd, 4th, 5th string
(svX, svY) = (-3,-2) StrLen=4
crrent CU (4x8)
five non-self-referencing strings

Fig. 5 Three self-referencing string decomposition algorithms. **a** an example of LSRS, **b** minimum decomposition, **c** unbroken-line decomposition, **d** single-line decomposition

In minimum decomposition, the length of the first non-SRS is given by StrLen1 = StrLen1a + StrLen1b, where StrLen1a is the number of pixel in the first |svY| lines of the original LSRS and StrLen1b is the number of pixels located on the (|svY| + 1)-th line (i.e. the line following the first |svY| lines) of the reference string of the original LSRS but not located on the first line of the current string of the original LSRS.

After obtaining the first non-SRS and its length, the same method of obtaining the first non-SRS and its length can be applied to the remaining part of the original LSRS to obtain the next non-SRS and its length recursively until the remaining part of the original LSRS is a non-SRS. An extensive case study reveals that among all possible decomposition, the minimum of the total number of decomposed non-SRSs $N_{\min}$ is given by

$$N_{min} = ceil(H/|SvY|) - 1. \quad (1)$$

where ceil(x) is the smallest integer greater than or equal to x.

In Fig. 5b), $N_{\min}$ is equals to ceil(5 / 2) – 1 = 3–1 = 2.

2) ***Unbroken-line decomposition***: In minimum decomposition, a line in the original LSRS may be broken into two segments separately located in two decomposed non-SRSs. Any broken line will reduce pixel copying throughput and should be avoided. Unbroken-line decomposition can fix the problem. In unbroken-line decomposition, each decomposed non-SRS starts at the beginning of a line except the first non-SRS, ends at the end of a line except the last non-SRS, and has |svY| lines except the last non-SRS.

As shown in Fig. 5c, the original LSRS is decomposed into three non-SRSs by unbroken-line decomposition algorithm. The first and the second non-SRSs have (svX, svY) = (−3, −2) and StrLen = 8. The third non-SRS has (svX, svY) = (−3, −2) and StrLen = 4.

The total number of decomposed non-SRSs $N_{\text{unbroken-line}}$ is given by

$$N_{unbroken-line} = ceil(H/|SvY|). \quad (2)$$

In Fig. 5c $N_{\text{unbroken-line}}$ is equals to ceil(5 / 2) = ceil(2.5) = 3.

3) ***Single-line decomposition***: Each decomposed non-SRS is exactly one line except the first and last decomposed non-SRSs which may be equal to or less than one line.

As shown in Fig. 5d, the original LSRS is decomposed into five non-SRSs by single-line decomposition algorithm. Each non-SRS has (svX, svY) = (−3, −2) and StrLen = 4.

In single-line decomposition, the total number of decomposed non-SRSs $N_{\text{single-line}}$ is equal to the height (i.e. the number of lines) of the original LSRS and is given by

$$N_{single-line} = H. \quad (3)$$

To evaluate the decomposition algorithms, at least six principles should be considered when an LSRS is decomposed into N non-SRSs.

1) ***principle 1, SV invariant principle*:** The SV of an LSRS and SVs of N non-SRSs after decomposition are invariant.
2) ***principle 2, Length sum principle***: The sum of the length of N non-SRSs is equal to the length of the original LSRS.
3) ***principle 3, String constraints invariant principle***: The string constraints are the same as the string constraints of non-SRS based SP after decomposition.
4) ***principle 4, Minimizing N principle***: The less the N is, the higher the coding gain gets and the better the decomposition algorithm is.
5) ***principle 5, Maximizing pixel copying throughput principle***: The higher throughput the decomposition algorithm results in, the more favorable the decomposition algorithm is.
6) ***principle 6, Simple decomposition principle***: The simpler the decomposition algorithm is, the more attractive the decomposition algorithm is.

Table 1 lists the comparisons of three decomposition algorithms based on the six principles. N_1, N_2 and N_3 represent the decomposition number for the minimum decomposition, the

Table 1 Comparisons of three decomposition algorithms based on the six principles

	Principle 1	Principle 2	Principle 3	Principle 4	Principle5	Principle 6
Minimum decomposition	satisfy	satisfy	not satisfy	$N_1 \le N_2 \le N_3$	$T_1 < T_2 \approx T_3$	complex
unbroken-line decomposition	***satisfy***	***satisfy***	***satisfy***	$\boldsymbol{N_2 \le N_3}$	$\boldsymbol{T_2 \approx T_3}$	***simple***
Single-line decomposition	satisfy	satisfy	satisfy	N_3	T_3	simple

unbroken-line decomposition and the single-line decomposition algorithm, respectively. T_1, T_2 and T_3 represent the pixel copying throughput for the minimum decomposition, the unbroken-line decomposition and the single-line decomposition algorithm, respectively.

Based on the six principles, all three decomposition algorithms satisfy the first second principles, only the last four principles need to be considered in evaluating the decomposition algorithms. Among the three decomposition algorithms, the unbroken-line decomposition algorithm is the best trade-off due to the following reasons.

1) Although the minimum decomposition algorithm may get the minimum decomposition number N_1 of non-SRSs and has the highest coding gain, it may have a broken line thus have the minimum pixel copying throughput T_1 and has a quite complex decomposition procedure. What's more, since the string length is not always divisible by 4 after decomposition, thus the string constraints are not the same as the string constraints of non-SRS based SP after the minimum decomposition.
2) The unbroken-line decomposition algorithm is able to avoid the broken line to achieve higher pixel copying throughput than the minimum decomposition algorithm and also has a simple decomposition procedure. In particular, according to formulas (1), (2), in most cases its total number of decomposed non-SRSs is equal to the minimum decomposition algorithm and only in very limited cases the total number of decomposed non-SRSs is one more than the minimum decomposition, thus it has coding gain very close to the minimum decomposition algorithm.
3) Both single-line decomposition algorithm and unbroken-line decomposition algorithm have similar and very simple decomposition procedures. Both have no broken line and provide similar pixel copying throughput. It is obvious that $N_{\text{unbroken-line}}$ is much smaller than $N_{\text{single-line}}$ for $|\text{SvY}| > 1$ or equal to $N_{\text{single-line}}$ for $|\text{SvY}| = 1$, thus the unbroken-line decomposition algorithm can achieve higher coding gain than the single-line decomposition algorithm.

Overall, ***the unbroken-line decomposition algorithm is the best trade-off and is discussed in more details in the rest of the paper and used in the experiments section***.

The pseudocode for the unbroken-line decomposition algorithm is described in Table 2. Let *cuWidth* be the width of a CU. For an LSRS in a CU, let (*svX*, *svY*) be the SV of the string, *SRSLen* be the length of the string, *xStartPos*, *yStartPos* be the x, y position of the first pixel of the string relative to the top-left position of the CU. *nonSRSLen[N]* stores the length for each of *N* non-SRSs. Since the traverse-horizontal scanning direction (neighboring rows have opposite direction) is used, the length of the first non-SRS depends on the parity (even or odd number) of *yStartPos*. The first non-SRS can be divided into three parts: the first line, the middle *abs(svY) – 2* lines and the final line. If *yStartPos* is in the odd line of the CU (*yStartPos & 0 × 1* is true), the length of the first line (*lengthF*) for an LSRS is *xStartPos + 1*, otherwise the length of the first line of an LSRS is *cuWidth – xStartPos*. The length of the middle lines

Table 2 The unbroken-line decomposition algorithm

Input	***SRSLen*, *cuWidth*, *xStartPos*, *yStartPos*, *svY***
Output	***nonSRSLen[N]*, *N***
Substep 1	**Initialization:** *N =1* *RemainingStrLen = SRSLen*

Substep 2	**Calculating each length of the nonSRSs**:
	*nonSRSLen [1] = min((abs(svY) – 1) * cuWidth + ((yStartPos & 0x1)? xStartPos + 1: cuWidth – xStartPos), RemainingStrLen)*
	while (*RemainingStrLen* > *nonSRSLen* [*N*]) {
	    RemainingStrlen - = *nonSRSLen* [*N*]
	    N ++
	    nonSRSLen [*N*] = *min* (*abs*(*svY*) * *cuWidth*, *RemainingStrLen*)
	}

(*lengthM*) for an LSRS is always equal to *cuWidth** (*abs(svY) – 2*). The length of the final line for an LSRS is *min*(*cuWidth, RemainingStrLen- lengthF- lengthM*). Thus, the sum of the length for the three parts is *nonSRSLen* [13] given in Table 2. The length of the next *N*-2 non-SRSs is always *abs*(*svY*)**cuWidth*. The length of the last non-SRS is the final *RemainingStrLen* that is less than or equal to *abs*(*svY*)**cuWidth*. After decomposition, the string copy operation of LSRS can be performed at least line-by-line.

3.3 Performance analysis

The proposed LSRS-enabled SP technique can further improve the coding efficiency with almost no additional encoding and decoding complexity as analyzed below.

Coding efficiency analysis In the LSRS-enabled SP coding system, the string copy parameters of one LSRS rather than multiple non-SRS are coded and written into the bitstream. For example, as shown in Fig. 5, if an LSRS with SV = (−3, −2) and StrLen =20 is found in the encoder, only one SV and one StrLen will be coded and written into the bitstream for twenty pixels. But in the non-SRS SP coding system, LSRS is not allowed, thus the encoder will generate two or more non-SRSs to code the same twenty pixels. Obviously, two or more SV and two or more StrLen with other string copy parameter coding overhead such as match_type_flag for each non-SRS have to be coded and written into the bitstream. For example, the encoder may search and find the following three strings: two with SV = (−3, −2) and StrLen = 8 and one with SV = (−3, −2) and StrLen = 4, resulting in a lot more bits to be coded and written into the bitstream. It is obvious that with LSRS, much fewer string copy parameters are coded and written into the bitstream and the coding efficiency is much improved.

Coding complexity analysis In the LSRS-enabled SP coding system, any LSRS is decomposed into multiple non-SRSs and the total number of non-SRSs in a CU has the same constraint as in the traditional non-SRS SP coding system. Therefore, the LSRS-enabled SP coding system has the same string copying throughput and coding complexity as the traditional non-SRS SP coding system. The only additional function needed by the LSRS-enabled SP coding system is the LSRS-to-non-SRS decomposition function. As shown in Table I, the unbroken-line decomposition function is very simple, straightforward, and easy to be implemented with little logic and cost.

As a result, *the LSRS technique can notably improve coding efficiency with almost no additional coding complexity*.

4 Experiments

4.1 Description of test images

Fourteen AVS3 SCC CTC [19] sequences shown in Table 3 are used in the experiment. CTC sequences can be classified into four categories: Text and Graphics with Motion (TGM), Mixed Content (MC), Gaming (G), and Camera Captured (CC). All test sequences are in YUV 4:2:0 format. Four of CTC sequences, sc_desktop, sc_console, BitstreamAnalyzer, program_vidyo are partially shown in Fig. 6a, b, c, d, respectively.

4.2 Experimental settings

The experiments evaluate and compare the coding efficiency and coding complexity of LSRSP using the following two CODECs.

1) HPM, the AVS3 reference software HPM9.1 "https://gitlab.com/AVS3_Software/hpm/-/tree/HPM-9.1".
2) HPM-LSRSP, the proposed line-based self-referencing string prediction technique implemented on HPM9.1.

All experimental results are generated under the CTC and configurations defined in [19] for AVS3 SCC. To evaluate overall coding efficiency improvement, the BD-rate metric [1] is used. Fourteen YUV 4:2:0 test sequences classified into Text and Graphics with Motion (TGM), Mixed Content (MC), Gaming (G), and Camera Captured (CC) categories are used. For each

Table 3 Fourteen AVS3-SCC CTC images used in the experiment

Resolution	Sequence name	Abbreviation	Category	fps	No. of frames
1920 × 1080	sc_flyingGraphics	FLYG	TGM	60	600
	sc_desktop	DSK	TGM	60	600
	sc_console	CNS	TGM	60	600
	ChineseDocumentEditing	CDE	TGM	30	300
	EnglishDocumentEditing	EDE	TGM	30	300
	Spreadsheet	SPS	TGM	30	300
	BitstreamAnalyzer	BSA	TGM	30	300
	CircuitLayoutPresentation	CLP	TGM	30	300
	program	PRG	TGM	60	600
	web_en	WEBB	TGM	60	600
	word_excel	WDEC	TGM	60	600
	program_vidyo	PRGV	MC	60	600
	ArenaOfValor	AOV	A	60	600
	BQTerrace	BQT	CC	60	600

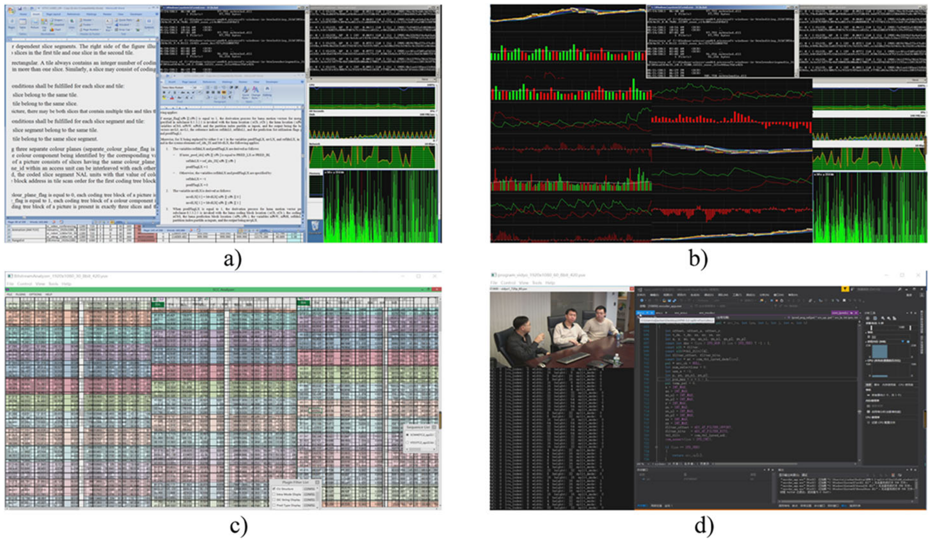

Fig. 6 Examples of AVS3 SCC test sequence. **a** a part of sc_desktop. **b** a part of sc_console. **c** a part of BitstreamAnalyzer. **d** a part of program_vidyo

sequence, BD-rate reduction is calculated separately for Y, U, and V color components. Four QPs of 27, 32, 38, and 45 and two configurations of AI and LDB are tested. To evaluate encoder and decoder complexity, encoding and decoding software runtimes are also compared.

4.3 Experimental results and conclusions

To evaluate the coding efficiency of LSPSP, the coding efficiency comparisons between HPM and HPM-LSPS for AI and LDB configurations are shown in Table 4. The PSNR-bps curves and BD-rate reduction percentage of CNS for AI configuration are shown in Fig. 7. The encoding runtime ratio and decoding runtime ratio between HPM and HPM-LSRSP are shown in Table 4.

The experimental results can be summarized as follows.

1) *For CTC sequences, the overall coding performance of HPM-LSRSP is notably improved.* For TGM and MC category, as shown in Table 4, HPM-LSRSP can achieve an average Y-BD-rate of −0.81%, −0.59% for AI and LDB. In particular, for CNS sequence, HPM-LSRSP can achieve YUV BD-rate of ***−2.04%, −2.04%, −1.75%*** for AI. As shown in Fig.7a, from the Y-PSNR-bps curves of CNS for AI configuration and QP of 22, the bitrate and Y-PSNR are improved from 35,989.5 kbps and 51.5 dB in HPM to 35,423.7 kbps and 51.6 dB in HPM-LSRSP. Figure 7b shows the U-PSNR-bps curves and BD-rate reduction percentage of CNS for AI configuration. For the CLP sequence, HPM-LSRSP can achieve a Y-BD-rate of −1.31% for LDB.

2) The coding performance improvement by HPM-LSRSP depends on the contents. As shown in Table 4, HPM-LSRSP has good BD-rate reduction for almost all the sequences

Table 4 Lossy coding performance comparison (BD-rate %) between HPM and HPM-LSPSP

	Test sequences		AI			LDB		
	Category	Abbreviation	Y	U	V	Y	U	V
HPM vs. HPM-LSRSP	TGM	FLYG	−0.37	−0.53	−0.49	−0.18	−0.07	−0.28
	TGM	DSK	−1.22	−1.17	−1.19	−0.91	−0.64	−0.44
	TGM	CNS	***−2.04***	***−2.04***	***−1.75***	−0.86	−1.03	−1.12
	TGM	CDE	−0.31	−0.20	−0.43	−1.23	−1.07	−1.38
	TGM	EDE	−0.22	−0.55	−0.65	−0.31	0.11	−0.09
	TGM	SPS	−0.77	−0.66	−0.09	−0.25	−0.13	0.18
	TGM	BSA	−1.02	−0.88	−1.04	−0.77	−0.35	−0.15
	TGM	CLP	−0.93	−1.04	−1.12	***−1.31***	***−1.12***	***−1.65***
	TGM	PRG	−0.65	−0.65	−0.91	0.45	0.09	1.07
	TGM	WEBB	−0.68	−0.74	−0.71	−0.86	−1.05	−1.14
	TGM	WDEC	−0.47	−0.79	−0.82	−0.23	0.88	0.00
	MC	PRGV	−1.09	−1.31	−1.07	−0.68	−1.39	−1.33
	G	AOV	0.0	0.0	0.0	0.0	0.0	0.0
	CC	BQT	0.0	0.0	0.0	0.0	0.0	0.0
	Average of TGM and MC		***−0.81***	***−0.88***	***−0.86***	***−0.59***	***−0.48***	***−0.53***

of TGM and M categories in all configurations, except the PRG sequence in the LDB configuration. The PRG LDB issue probably means that the optimal string selection method based on RD cost may not work as expected in this particular case and have room for further optimization. No coding performance improvement for CC and A categories because string prediction is disabled by picture-level early termination as the original SP in HPM does.

3) As shown in Table 5, *the encoding runtime and decoding runtime of HPM-LSRSP are almost the same as HPM.*

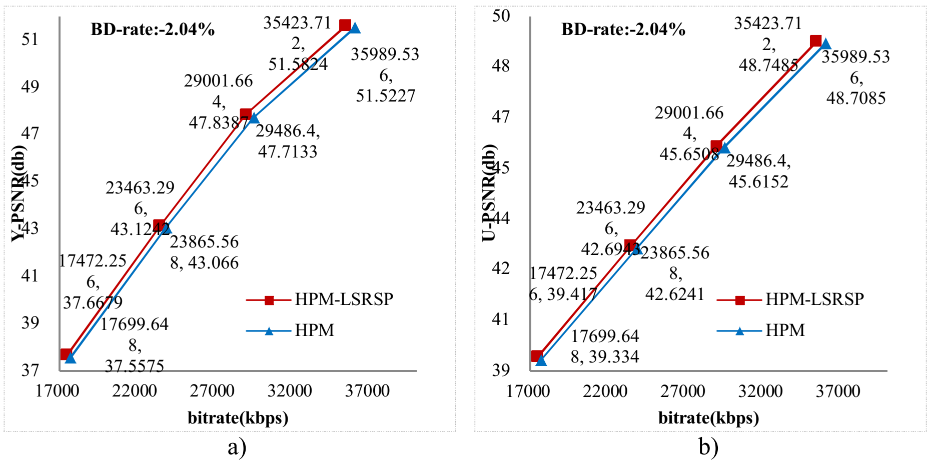

Fig. 7 The PSNR-bps curves and BD-rate reduction percentage of CNS for AI configuration. **a** Y-PSNR-bps curves, **b** U-PSNR-bps curves

Table 5 Complexity Comparison between HPM and HPM-LSRSP

	encoding time ratio [%]		decoding time ratio [%]	
	AI	LDB	AI	LDB
HPM vs. HPM-LSRSP	98.09	99.67	103.38	100.29

5 Conclusion and future work

Self-referencing strings play an important role in SP but are prohibited in early versions of SP to avoid very low pixel copying throughput problems. Statistical analysis shows that a majority of self-referencing strings are line-based self-referencing strings (LSRS). LSRS really does not have the low pixel copying throughput problem because it can always be decomposed into non-SRSs. Thus, an LSRS-enabled SP coding system is proposed. An LSRS is decomposed into multiple non-SRS to perform at least line-by-line string copy operations. Three decomposition algorithms are compared and the best trade-off is the unbroken-line decomposition algorithm. Finally, theoretical analysis and experimental results show that LSRS-enabled SP can notably improve coding efficiency with almost no additional encoding and decoding complexity.

Future work includes 1) Improving encoder RD cost-based optimal string selection algorithm, in particular, for LDB configuration; 2) Exploring other string prediction techniques with a better trade-off between coding efficiency and coding complexity; 3) Applying SP to the lossless or near-lossless coding for screen content [23], alpha channel images [30] and key-point sequences [7, 8]; 4) Applying deep learning and neural network approaches [7] to video analysis [8, 10], video processing [3, 4] and SP based video coding.

Acknowledgments This work is supported by the National Natural Science Foundation of China (No. 62271321, No. 61871289), the Science and Technology Plan Project in Basic Public Welfare class of Shaoxing city (No. 2022A11002), the Social Sciences and Humanities Youth Foundation of Ministry of Education (No. 21YJCZH039), Zhejiang Provincial Postdoctoral Science Foundation No. ZJ2022066, Zhejiang Provincial Natural Science Foundation of China (No. LTY22F020003), the Ministry of Education Industry University Cooperation Collaborative Education Project (No. 202101011025), and the Humanities and Social Sciences Project of Shaoxing University (No. 2021LJ001).

Data availability The data that support the findings of this study are available from ftp://47.93.196.121/Public/Seqs/Video/ but restrictions apply to the availability of these data, which were used under licence for the current study, and so are not publicly available. Data are however available from the authors upon reasonable request and with permission of the Audio Video Standard Workgroup of China (http://www.avs.org.cn/english/).

Declarations

Conflict of interest The authors declare that there are no conflicts of interests.

References

1. Bjøntegaard G (2001) Calculation of average PSNR differences between RD-Curves, ITU-T SG16 Q.6 Document, VCEG-M33, Austin, USA
2. Chen C, Peng W (2017) Intra Line Copy for HEVC Screen Content Intra-Picture Prediction. IEEE Trans Circuits Syst Video Technol 27(7):1568–1579

3. Feng S, Hu K, Fan E, Zhao L, Wu C (2021) Kalman filter for spatial-temporal regularized correlation filters. IEEE Trans Image Process 30:3263–3278
4. Hu K, Ye J, Fan E, Shen S, Huang L, Pi J (2017) A novel object tracking algorithm by fusing color and depth information based on single valued neutrosophic cross-entropy. J Intell Fuzzy Syst 32(3):1775–1786
5. Li B, Xu J, Wu F (2014). 1-D dictionary mode for screen content coding. In: IEEE Visual Communications and Image Processing Conference, 189–192.
6. Lin T, Zhang P, Wang S, Zhou K, Chen X (2013) Mixed Chroma Sampling-Rate High Efficiency Video Coding for Full-Chroma Screen Content. IEEE Trans Circuits Syst Video Technol 23(1):173–185
7. Lin W, He X, Dai W, See J, Shinde T, Xiong H, Duan L (2020) Key-point sequence lossless compression for intelligent video analysis. IEEE MultiMedia 27(3):12–22
8. Lin W, Shinde T, Dai W, Liu M, He X, Tiwari A, Xiong H (2020) Adaptive lossless compression of skeleton sequences. Signal Process Image Commun 80:11567
1–14
9. Liu Y, Fang C, Sun J, Huang X (2019) Fast Palette Mode Decision Methods for Coding Game Videos With HEVC-SCC. IEEE Trans Circuits Syst Video Technol 29(10):3061–3067
10. Ma S, Zhang X, Jia C, Zhao Z, Wang S, Wang S (2020) Image and Video Compression with Neural Networks: A Review. IEEE Trans Circuits Syst Video Technol 30(6):1683–1698
11. Ma Z, Wang W, Xu M et al Advanced Screen Content Coding Using Color Table and Index Map. IEEE Trans Image Process 23(10):4399–4412
12. Nguyen T, Xu X, Henry F et al (2021) Overview of the screen content support in VVC: applications, coding tools, and performance. IEEE Trans Circuits Syst Video Technol 31:3801–3817. https://doi.org/10.1109/TCSVT.2021.3074312
13. Peng W, Walls F, Cohen R et al (2016) Overview of screen content video coding: technologies, standards, and beyond. IEEE J Emerg Sel Top Circuits Syst 6(4):393–408
14. Wang S, Lin T (2014) United coding method for compound image compression. Multimed Tools Appl 71(3):1263–1282
15. Wang Y, Zhou Q, Zhao L et al (2020) SCC: Unit Basis Vector String for String Prediction, *AVS M5994,* China
16. Wang Y, Xu X, Liu S (2021) Low complexity implementation of intra string copy in AVS3. In: IEEE International Conference on Multimedia & Expo Workshops (ICMEW), 1-4
17. Xiao W, Shi G, Li B, Xu J, Wu F (2018) Fast Hash-Based Inter-Block Matching for Screen Content Coding. IEEE Trans Circuits Syst Video Technol 28(5):1169–1182
18. Xiao W, Bin L, Xu J, Shi G, Wu F (2018) Weighted Rate-Distortion Optimization for Screen Content Coding. IEEE Trans Circuits Syst Video Technol 28(2):499–512
19. Xu X (2020) AVS3-P2 Common Test Conditions for screen content coding, *AVS Document*, AVS M5928, China
20. Xu X, Liu S Overview of screen content coding in recently developed video coding standards. IEEE trans Circuits Syst Video Technol. https://doi.org/10.1109/TCSVT.2021.3064210
21. Xu X, Liu S, Chuang T et al (2016) Intra block copy in HEVC screen content coding extensions. IEEE J Emerg Sel Top Circuits Syst 6(4):409–418
22. Yang Y, Zhou K, Zhao L, Lin T (2021) An ultralow complexity string matching approach to screen content coding in AVS3[J]. IEEE Trans Circuits Syst Video Technol 31(9):3714–3718
23. Yang Y, Lin T, Zhao L, Zhou K, Wang S (2021) A string matching based ultra-low complexity lossless screen content coding technique. Multimed Tools Appl 81:2043–2063. https://doi.org/10.1007/s11042-021-11418-6
24. Zhao L, Lin T, Zhou K, Wang S, Chen X (2016) Pseudo 2D String Matching Technique for High Efficiency Screen Content Coding. IEEE Trans Multimedia 18(3):339–350
25. Zhao L, Lin T, Zhou K (2017) An efficient ISC offset parameter coding algorithm in screen content coding. Chin J Comput 40(5):1–11
26. Zhao L, Zhou K, Guo J, Wang S, Lin T (2018) A Universal String Matching Approach to Screen Content Coding. IEEE Trans Multimedia 20(4):796–809
27. Zhao L, Zhou K, Lin T (2018) Pixel string matching for full-chroma screen and mixed content coding in AVS2. Chin J Comput 41(11):2482–2495
28. Zhao L, Zhou K, Lin T et al (2019) A universal string prediction approach and its application in AVS2 mixed content coding. Chin J Comput 42(41):1–15
29. Zhao L, Lin T, Guo J et al (2019) Universal string prediction-based inter coding algorithm optimization in AVS2 mixed content coding. Chin J Comput 42(28):1–13
30. Zhao L, Lin T, Zhang D, Zhou K, Wang S (2020) An Ultra-Low Complexity and High Efficiency Approach for Lossless Alpha Channel Coding. IEEE Trans Multimedia 22(3):786–794

31. Zhou K, Zhao L, Lin T (2016) A hardware decoder architecture for general string matching technique. IEEE J Emerg Sel Top Circuits Syst 6(4):560–572
32. Zhou K, Zhao L, Lin T (2018) A flexible and uniform string matching technique for general screen content coding. Multimed Tools Appl 77:23751–23775
33. Zhou Q, Zhao L, Wang H et al (2020) CE_SCC_related: Encoding and Decoding Optimization for String Vector based on the Inherent Characteristics of Several Above Strings. *AVS M5950*, China
34. Zhou Q, Zhao L, Zhou K, Lin T, Wang H, Wang S, Jiao M (2021) String prediction for 4:2:0 format screen content coding and its implementation in AVS3. IEEE Trans Multimedia 23:3867–3876
35. Zou F, Chen Y, Karczewicz M, Seregin V (2015) Hash based intra string copy for HEVC based screen content coding. In: ICMEW, pp. 1–4.