Variable-Length Segment Copy for Compressing Index Map of Palette Coding in Screen Content Coding

Yao-Jen Chang[1], Ching-Chieh Lin[1], Chao-Hsiung Hung[1], Jih-Sheng Tu[1], Chun-Lung Lin[1(✉)], and Pei-Hsuan Tsai[2]

[1] Industrial Technology Research Institute, Hsinchu, Taiwan
{britpablo, JackLin, chhung, sunriseJSTu, Chunlung}@itri.org.tw
[2] Institute of Manufacturing Information and Systems, National Cheng Kung University, Tainan, Taiwan
peihsuan.tsai@gmail.com

Abstract. With the emerging applications, such as screen mirroring, and remote play, screen contents coding (SCC) plays important role in video coding recently. Since the characteristics of screen contents are different from nature contents, palette coding is adopted in the current draft standard of HEVC-SCC. The basic idea of the palette coding is to represent the colors of a coding unit (CU) by the indices of selected representative colors. This paper presents that the produced index maps exhibit considerably high spatial correlation. To utilize the spatial correlation among indices, a general 2-D search method is proposed firstly for index map compression. To reduce the memory access and implementation complexity, three simplified search schemes are proposed to balance the coding performance and complexity. The experimental results show that the three simplified methods can achieve 0.6 %, 0.5 % and 0.9 % BD-rate saving respectively, as compared to HM-13.0 + RExt-6.0 test model.

Keywords: Segment matching mode · Line-based · Palette · Screen content coding

1 Introduction

With rapid development of technologies and increased internet bandwidth, such as GPON, 3G and 4G etc., people are no longer satisfied with only communicating with each other via text or speech. More and More applications have started to provide services with many still images or videos to attract more users. These multimedia signals not only improve working efficiency but also enrich our daily life. In recent years, wireless displaying, cloud computing and gaming, screen sharing and collaboration, remote desktop, etc. are booming and have attracted a lot of attention. These emerging use cases can obliviously re-use the existing video compression technologies such as the state-of-the-art AVC [1] or HEVC [2] to fulfill purposes.

However, screen contents are fully or partial generated by computers. Console desktop, spreadsheet, slides and web browsers are typical examples for the screen

T. Bräunl et al. (Eds.): PSIVT 2015, LNCS 9431, pp. 148–159, 2016.
DOI: 10.1007/978-3-319-29451-3_13

contents with mixing lots of text, graphics, nature still images and nature video sequences. In addition, the screen contents usually contain sharp edges of the objects, background with simple colors and many thin lines. On the other hand, nature contents captured by video camcorders generally have rich colors, complex texture/shape and smooth-edged objects. Therefore, HEVC provides up to 35 spatial directions to code the nature contents. [2].

Since the state-of-the-art AVC and HEVC standards are mainly developed for the nature contents, the coding efficiency may not be adequate for the screen contents which have dissimilar characteristics from the nature contents. Therefore, ISO/IEC Moving Picture Experts Group (MPEG) and ITU-T Video Coding Experts Group (VCEG) issued a joint Call for proposal (CfP) for screen content coding (SCC) [3] in January 2014. Several responses were proposed and evaluated by the Joint Collaborative Team on Video Coding (JCT-VC) group in April 2014.

These responses all proposed two major coding tools for SCC: Intra Block Copy (IBC) [4] and Palette mode [5, 6]. IBC mode uses an estimated block vector to locate a similar block for the current partition in the current frame. The IBC mode is similar to the Inter Motion Estimation; however, the IBC mode only uses the current frame as a reference frame. The palette coding conceptually uses palette tables and index maps to represent the current CU. Overall, the palette coding has two major processing steps: palette table generation and index map compression. A palette table is generated by classifying the colors in the current CU and then reordering them according to the frequency of occurrence. After that, an index map is created by converting each sample to indices via the established palette table. Both palette tables and index maps need to be transmitted to the decoder.

Palette mode was proposed by Microsoft and Qualcomm in the 16th JCT-VC meeting [7, 8]. They are named PM and PQ respectively in this paper. In order to code the index map efficiently, we introduce advance segment matching mode on top of PM which is a line-based palette coding. Since many blocks or lines repeatedly appear in one frame for screen contents, similar or identical blocks/lines have high opportunities to be found from the previous decoded indices in the current coding CU for the palette coding. Splitting a line into several segments with various lengths plus 2-D searching is proposed first in this paper. However, using 2-D searching with various length segments not only requires a huge amount of data access but also increases the difficulties of the hardware implementation. Therefore, this paper further presents three simplified methods to reduce the complexity. The maximum BD-rate saving can achieve 0.6 %, 0.5 % and 0.9 % for "YUV, text & graphics with motion, 1080p" class for All Intra test condition.

The rest of this paper is organized as follows. Section 2 gives the overview of the palette coding and its related works. The proposed advanced segment matching tool and the three simplified methods are presented in Sect. 3. The performance evaluations are carried out and discussed in Sect. 4. The concluding remarks and possible future works are drawn in Sect. 5.

2 Overview

Palette coding has been studied in JCT-VC for many meeting cycles. It analyzes the colors in the current CU at first to generate the palette table and transmit the converted color indices to the decoder. Since neighboring samples in the screen contents generally have high contrasts such as sharp edges, conventional spatial coding tools may not be able to compress the screen contents efficiently. In addition, the screen contents contain limited colors. Therefore, the palette mode does not need to maintain a huge palette table for a color-index mapping process. Meanwhile, less numbers of the colors in the palette table reduce the coded bits to represent the palette indices. It improves the index map coding performance.

During the investigation of the palette mode, there were two main solutions with different implementations. The first one was a run-based palette coding PQ [8]. Compared with the conventional HEVC coding tools, the PQ packs the three components, e.g. YCbCr or GBR, for the palette table generation. The encoder first generates a color histogram according to the frequency of the color appearance and then groups similar colors together as major colors via an error allowance threshold value. Since the palette table needs to be transmitted to the decoder, the last decoded palette table is used as a palette table predictor in order to avoid resending the identical major colors. This is done by simply signaling reuse flags to indicate which colors in the last decoded palette table are reused in the current coding CU. Then, un-predicted major colors need to be signaled directly to the decoder. After applying the color to index mapping, the encoder can use two copy-run methods to compress the index map. When the current and the following color indices are the same with the above indices, Copy Above Run mode (CAR) is valid. If the current coding color index and the followed indices are the same, the Copy Left Run mode (CLR) is available. The number of the identical indices is defined as run value. In addition, CLR is also used for coding index when the run value equals one. Both the mode flag for the two modes and their run value are signaled. For the decoding process, the decoder parses the reuse flags and the signaled major colors to regenerate the palette table for the current decoding CU at first. After that, the index map is reconstructed by inversing the indices encoding process. The last step is to convert the decoded indices to color samples.

The line-based palette coding PM basically shares the same ideas for palette table generation and index map compression. Instead of packing all the three components together for selecting the major colors, PM generates triple palette tables for each component. In order to signal the major colors more efficiently, PM also uses the last decoded palette tables as predictors. After converting the color samples in the current coding CU by referencing the palette tables individually, triple index maps are created. For index maps compression, PM provides three copy modes: Vertical, Horizontal and Normal modes. The Vertical mode is similar to CAR in PQ. The main difference is that Vertical mode is valid if the current line is the same as the above line. The Horizontal mode is available when the whole color indices of the current line are all the same. For Normal mode, there are three sub-modes inside. They are Above, Left and No Prediction modes. When Normal mode is applied, each color indices will be coded in a pixel-by-pixel manner. Above mode and Left mode use the above and the left

neighboring index of the current coding index as the predictor respectively. No prediction handles unpredictable indices. On the decoder side, the decoder regenerates the three palette tables and the procedure is similar to PQ. Then, the triple index maps are reconstructed by parsing the three modes. Lastly, the decoder converts the index map to color samples by looking up the palette tables.

3 Proposed Algoriths

In the current PM design, Vertical mode is only allowed to use the above one neighboring row to perform matching process. In addition, it also requires that every index in the above row should be identical to the current line. Owing to the special characteristics of screen contents such as repeatedly occurred objects/lines and simple background, the decoded indices excluding the above row may have useful information that can improve the coding efficiency. In the following sections, we first report the analysis about the correlations between the current line and the previous decoded lines in the current CU. After that, we present four novel methods for index map compression.

3.1 Previous Line Matching Analysis and General 2-D Index-Segment Copy Scheme

Our analyzing result is illustrated in Table 1. In this table, "occur once" means that only one matched line is found. "Occur twice" and "occur repeatedly" represent two and more than two matched lines, respectively.

As an obvious consequence, once the Vertical mode has been selected, it is highly possible to find an exactly matched line in the same CU (more than 90 % in Table 1). Hence, we can make a hypothesis based on the results in Table 1. If the current line differs from its above one, it still may find a matched line in the current CU. We can further extend this hypothesis by supposing that partial concatenation indices called a "segment" in the current line may find a matched segment in the already decoded part of the current CU. According to this hypothesis, a palette coding improvement algorithm is proposed. This algorithm includes two parts:

Extremely Search. For a current line, it can be considered as a combination result of segments. And there may be many possible combination results for the current line. For the segments in one combination result, the encoder could find their matched segments in the previous lines. If a matched segment is found, a shift vector and a segment length need to be recorded. The vector and the length are used to indicate where and how long do the start position of a segment can copy the indices from the reconstructed frame. If the current start position can't find any matched index, then it is coded as Normal mode.

Calculate the Cost of the Coded-Bins. If several search results are found in the search stage, the encoder would calculate the cost of the coded-bins of each pair's vector and length.

Table 1. The analysis result of Vertical mode

Category	Test Sequence	QP	Occur once (%)	Occur twice (%)	Occur repeatedly (%)
RGB, text & graphics with motion, 1080p	Flyinggraphicstext	27	4.80	4.65	90.54
		37	4.33	4.16	91.51
	Desktop	27	5.27	5.10	89.63
		37	4.50	4.50	91.00
	Console	27	3.87	3.87	92.26
		37	3.49	3.44	93.07
RGB, text & graphics with motion,720p	WebBrowsing	27	4.34	4.34	91.32
		37	3.27	3.27	93.45
	Map	27	15.48	12.50	71.43
		37	5.63	4.93	89.44
	Programming	27	7.33	7.33	85.34
		37	6.47	6.47	87.06
	SlideShow	27	10.00	10.00	83.33
		37	5.45	5.45	90.91
RGB, mixed content, 1440p	BasketballScreen	27	5.97	5.60	88.43
		37	4.92	4.92	90.15
	MissionControlClip2	27	6.94	6.36	86.71
		37	4.04	4.04	91.93
YUV, text & graphics with motion, 1080p	Flyinggraphicstext	27	4.82	4.65	90.53
		37	3.86	3.86	92.27
	Desktop	27	4.41	4.35	91.19
		37	3.89	3.89	92.22
	Console	27	3.14	3.14	93.68
		37	2.97	2.97	94.06
YUV, text & graphics with motion,720p	WebBrowsing	27	4.44	4.44	91.24
		37	2.50	2.50	95.01
	Map	27	9.33	8.00	81.33
		37	4.49	4.49	89.89
	Programming	27	7.82	7.41	84.77
		37	7.04	7.04	85.92
	SlideShow	27	8.57	8.57	85.71
		37	2.78	2.78	93.06
YUV, mixed content, 1440p	BasketballScreen	27	5.73	5.73	88.89
		37	4.65	4.65	90.70
	MissionControlClip2	27	4.48	4.48	91.42
		37	4.59	4.59	89.91

The pairs of the vector and the length with the minimum cost for the current line would be the final search result. Figure 1 illustrates an example of this search method.

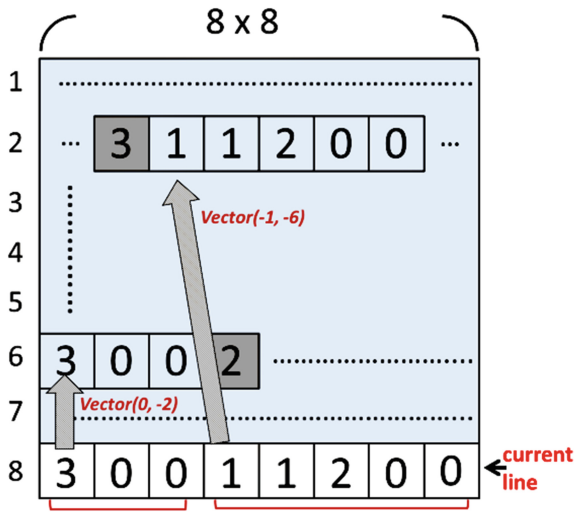

Fig. 1. An example of "segment search" result.

In Fig. 1, it is an 8 × 8 block and the 8th line is the current line. The current line is separated into two segments after the search stage. For these two segments, the matched segments locate in the 6th and the 2nd line. The shift vector and length of the first segment are (0, –2) and 3. Those of second segment are (–1, –6) and 5. These information all need to be coded.

However, there is no doubt that the high complexity of the aforementioned algorithm would lead to a huge overhead in computing time or memory usage. Therefore, we provide three simplified designs for the proposed algorithm. In the next sections, we will introduce these methods in detail.

3.2 1st Simplified Mode: Copy 2nd Above Mode

As the analysis shown in Sect. 3.1, the encoder and the decoder may find an identical line in the decoded indices for the current line easily. It is straightforward to force the encoder and the decoder to search one more line instead of only using the above line. Compared with the original three mode in PM for index map compression, we introduce an additional mode: copy 2nd above mode. When the current line and the above line are different, the encoder will bypass the Vertical mode and use the second line above the current line as a reference in our proposed mode. If the second line above the current line is identical to the current line, the proposed mode is valid. The decoder can parse the mode flag and copy the second line above the current line to the current line.

Figure 2 is an example for the first simplified mode. For the original Vertical mode, the 5th and the 6th lines are different. Therefore, the Vertical mode is unavailable. Meanwhile, in our proposed copy 2nd above mode, the 4th line is the same as the current 6th line. In this case, our proposed mode can be used here to improve the coding efficiency.

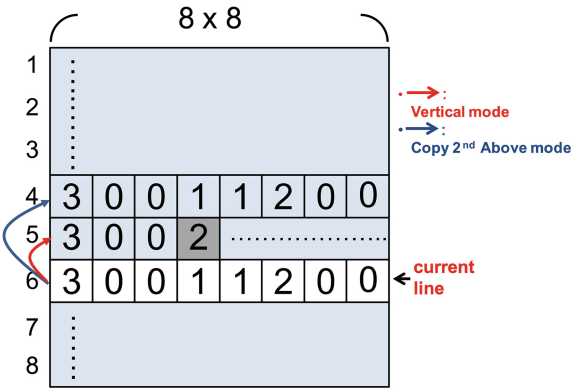

Fig. 2. An example of "Copy 2nd Above" mode.

The extra hardware complexity for the copy 2nd above mode is almost negligible. However, this mode may not be able to use the full information in the decoded indices. To approach the general 2-D searching, we propose the following two simplified designs.

3.3 2nd Simplified Mode: Sliding Half Line Mode

Our previous analysis shows a line has matched one in above processed region with high probability. Thus we partition a Normal-processed line into small segments and find matched ones. We provide two more simplified modes based on this idea.

Figure 3(a) shows an example of sliding half line mode based on this idea. The half-length segment *L1* is located arbitrarily within the current line. This simplified mode is also called match mode. This is a new model and proposed to replace original Normal mode.

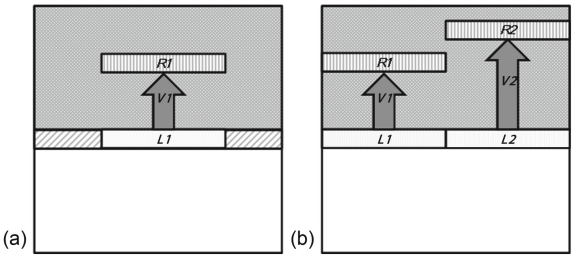

Fig. 3. (a) Sliding half line mode (b) splitting half line mode.

Figure 4 illustrates the operation of the match mode. In the match mode, each current segment *L1* has many reference segments *R1* ∼ *Ri* above in Fig. 4(a). In Fig. 4 (b), the match mode will evaluate whether the current segment *L1* matches one of the

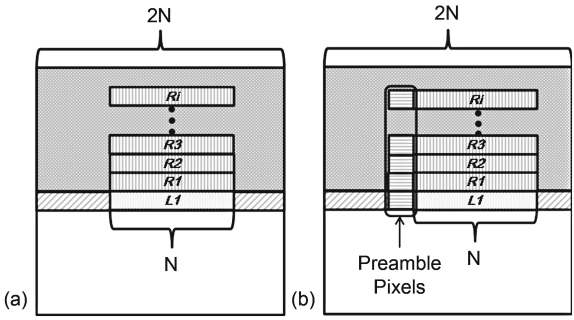

Fig. 4. Match mode. (a) Reference segments *Ri* above current segment *L1*. (b) Operation of match mode.

above reference segments. In order to simplify and accelerate the matching process, we take previous two indices before each segment for pre-matching process. This group of indices is preamble indices in Fig. 4(b). The preamble indices and the reference segments $R1 \sim Ri$ are already decoded while the current segment *L1* is going to be coded now.

If the current line is processed by the match mode, there are four steps to code *L1* in Fig. 4(b).

Step 1: For the preamble indices before *L1*, it finds the matched preamble indices of *Ri* above. If the preamble indices have no matched ones, the first index of *L1* will be processed by three sub-modes in original Normal mode. Then it goes to next index and re-starts the step 1.

Step 2: If the matched preamble indices of *Ri* exist, it then checks the corresponding *Ri* and *L1*.

Step 3: If no *Ri* matches *L1*, it transmits a flag "0" and codes the first index of *L1* by three sub-modes in original Normal mode. Then it goes to the next index and re-starts the step 1.

Step 4: If at least one *Ri* match *L1*, it labels these reference segments by matching indices starting from 0. It then transmits the flag "1" and the corresponding matching index. Then it goes to the index next to the end of *L1* and re-starts the step 1.

For the decoding process, the decoder has to parse the specific flag and the corresponding matching index at first. With the inversed encoding steps, the decoder then can use the matching index to find the preamble. After that, the decoder can copy the half line width indices after the preamble to the current decoding position.

3.4 3rd Simplified Mode: Splitting Half Line Mode

Previous simplified mode requires the additional pre-matching process which may complicate the hardware implementation. In consequence, we propose another simplified mode, splitting half mode in Fig. 3(b). This simplification is a modified Normal mode.

It splits the Normal mode line into two segments, *L1* and *L2* at first. The width of *L1* and *L2* are equal. Then it finds a matched segment *R1* for *L1* vertically in above processed region. If *R1* exists, the shift vector *V1* will be coded. Otherwise, a zero vector will be coded and *L1* will be processed by the original Normal mode. The other half segment *L2* is processed similarly.

On the decoder side, if the parsed mode flag is the half line mode, the decoder decodes the shift vector at first. If the shift vector is zero, the decoder decodes this half segment by the original Normal mode. Otherwise, the decoder finds the identical segment according to the shift vector and then copies the indices to the current decoding position.

4 Experimental Results

We integrated the line-based palette coding PM [2] into MPEG reference software HM-13.0+RExt-6.0. We then compare the coding performance of integrated software (PM+HM-13.0+RExt-6.0) and the reference software (HM-13.0+RExt-6.0). The experimental conditions are subject to the test conditions issued by JCT-VC in the 16th meeting [9]. Each test sequences must be tested under All Intra, Random Access, and Low-delay B conditions.

Since the palette coding is one of the intra-prediction coding tools, we are more interested in evaluating the All Intra experimental results. Our proposed methods are tested only under All Intra test conditions. The integrated software and the four simplified methods run on the different simulation platform. Encoding time and decoding time may not be reliable. Therefore, the comparisons of the averaged encoding and decoding time are not reported in our experimental results.

The improvement of the integrated software is investigated at first. Table 2 gives the All Intra results in BD-rate. The more negative value means better coding performance. Table 2 shows that the palette coding improves the coding performance, especially for the screen content test sequences.

Table 2. All Intra experimental results. The test algorithm and the anchor are the integrated software and the reference software respectively.

	AllIntra		
	Y	U	V
RGB, text & graphics with motion, 1080p	-8.3%	-8.0%	-8.4%
RGB, text & graphics with motion,720p	-3.8%	-3.5%	-3.7%
RGB, mixed content, 1440p	-1.0%	-0.8%	-1.0%
RGB, mixed content, 1080p	-1.3%	-1.1%	-1.2%
RGB, Animation, 720p	0.1%	-0.1%	0.0%
YUV, text & graphics with motion, 1080p	-8.7%	-6.7%	-7.4%
YUV, text & graphics with motion,720p	-3.0%	-2.8%	-4.0%
YUV, mixed content, 1440p	-1.2%	-1.4%	-1.4%
YUV, mixed content, 1080p	-1.1%	-0.9%	-0.9%
YUV, Animation, 720p	0.1%	0.0%	0.0%

The integrated software is the platform for implementation of our proposed modes. Tables 3, 4 and 5 give the All Intra experimental results. The anchors are the integrated software. It shows that the proposed simplified modes improve the palette coding performance.

Table 3. All Intra experimental results. The test algorithm and the anchor are the copy 2nd above mode and the integrated software respectively.

	All Intra		
	Y	U	V
RGB, text & graphics with motion, 1080p	-0.5%	-0.5%	-0.5%
RGB, text & graphics with motion,720p	0.0%	0.0%	-0.1%
RGB, mixed content, 1440p	0.0%	0.0%	0.0%
RGB, mixed content, 1080p	0.0%	0.0%	0.0%
RGB, Animation, 720p	0.0%	0.0%	0.0%
YUV, text & graphics with motion, 1080p	-0.6%	-0.3%	-0.4%
YUV, text & graphics with motion,720p	0.0%	0.0%	0.0%
YUV, mixed content, 1440p	0.0%	0.0%	0.0%
YUV, mixed content, 1080p	0.0%	0.0%	0.0%
YUV, Animation, 720p	0.0%	0.0%	0.0%

Our proposed copy 2nd above mode achieves 0.5 % and 0.6 % BD-rate saving in "RGB, text & graphics with motion, 1080p" and "YUV, text & graphics with motion, 1080p" classes with minor change. However, there is no coding gain in the rest of the testing classes. The reason may be caused by not using the full information provided by the decoded indices in the current CU.

Meanwhile, the sliding half mode provides much adaption of the current segment location with maximum 0.5 % coding gain as shown is Table 4. Comparing to the splitting half mode, the slightly worse coding performance may be caused by the pre-matching process. The size of the preamble indices is two in our current design to reduce the computations and accelerate the matching process. However, the preamble indices, which can be considered as predictors, cannot perfectly reflect the matching process. It is a tradeoff between coding performance and complexity.

Table 4. All Intra experimental results. The test algorithm and the anchor are the sliding half line mode and the integrated software respectively.

	AllIntra		
	Y	U	V
RGB, text & graphics with motion, 1080p	-0.5%	-0.5%	-0.5%
RGB, text & graphics with motion,720p	-0.2%	-0.2%	-0.2%
RGB, mixed content, 1440p	-0.1%	0.0%	-0.1%
RGB, mixed content, 1080p	-0.1%	-0.1%	-0.1%
RGB, Animation, 720p	0.0%	0.0%	0.0%
YUV, text & graphics with motion, 1080p	-0.5%	-0.3%	-0.3%
YUV, text & graphics with motion,720p	-0.2%	-0.3%	-0.2%
YUV, mixed content, 1440p	-0.1%	-0.1%	-0.1%
YUV, mixed content, 1080p	-0.1%	-0.1%	-0.1%
YUV, Animation, 720p	0.0%	0.0%	0.0%

Table 5. All Intra experimental results. The test algorithm and the anchor are the splitting half line mode and the integrated software respectively.

	All Intra		
	Y	U	V
RGB, text & graphics with motion, 1080p	-0.8%	-0.8%	-0.8%
RGB, text & graphics with motion,720p	-0.3%	-0.3%	-0.3%
RGB, mixed content, 1440p	-0.1%	-0.1%	-0.1%
RGB, mixed content, 1080p	-0.1%	-0.1%	-0.1%
RGB, Animation, 720p	0.0%	0.0%	0.0%
YUV, text & graphics with motion, 1080p	-0.9%	-0.5%	-0.6%
YUV, text & graphics with motion,720p	-0.3%	-0.2%	-0.3%
YUV, mixed content, 1440p	-0.1%	-0.2%	-0.2%
YUV, mixed content, 1080p	-0.1%	-0.1%	-0.1%
YUV, Animation, 720p	0.0%	0.1%	0.0%

The final proposed mode, the splitting half mode, provides better coding performance with the simple and clear design. It handles each line with the simple process and consumes less computing resources. In Table 5, the splitting half mode can achieve 0.8 % and 0.9 % BD-rate saving in " RGB, text & graphics with motion, 1080p" and " YUV, text & graphics with motion, 1080p" classes. For all the three proposed methods, there is no improvement for the animation classes. It may be caused by too many rich colors and complicated textures in the class.

5 Conclusions

In this paper, we have analyzed the probability of finding above matched lines for current processing line at first. The analysis shows that there exists a high correlation between the current line and the previous lines. Then, we have proposed a general 2-D palette indices search method and three simplified methods from Sects. 3.2 to 3.4. The first simplification use an additional mode to check the second line above the current line. Coding improvement is only shown in the "text & graphics with motion, 1080p" classes. The second simplified method uses our observed feature to provide obvious coding benefit. Then, the third simplified method further reduces the syntax overhead in coded-bins to achieve better coding gain. All of our proposed methods improve the coding efficiency significantly on top of the line-based palette coding. As the simulation results show, the third proposed simplified method gets 0.8 % to 0.9 % coding gain at the "RGB, text & graphics with motion, 1080p" and the "YUV, text & graphics with motion, 1080p" test sequence categories individually. Since the run-based and the line-based palette coding share the similar ideas for compression, integrating our proposed methods on top of the run-based palette coding will be our next step.

References

1. Wiegand, T., Sullivan, G.J., Bjontegaard, G., Luthra, A.: Overview of the H.264/AVC video coding standard. IEEE Trans. Circuits Syst. Video Technol. **13**(7), 560–576 (2003)
2. Sullivan, G.J., Ohm, J., Han, W.-J., Wiegand, T.: Overview of the high efficiency video coding (HEVC) standard. IEEE Trans. Circuits Syst. Video Technol. **22**(12), 1649–1668 (2012)
3. Joint call for proposals for coding of screen content, ISO/IEC JTC1/SC29/WG11 MPEG, MPEG N14715 (2014)
4. Chang, T.-S., Liao, R.-L., Chen, C.-C., Peng, W.-H., Hang, H.-M., Lin, C.-L., Jou, F.-D.: RCE3: results of subtest B.1 on Nx2N/2NxN intra block copy. Document JCTVC-P0176 (2014)
5. Guo, L., Karczewicz, M., Sole, J.: RCE3: results of Test 3.1 on palette mode for screen content coding. Document JCTVC-N0247 (2013)
6. Zhu, W., Xu, J., Ding, W.: RCE3 Test 2: multi-stage base color and index map. Document JCTVC-N0287 (2013)
7. Guo, X., Lu, Y., Li, S.: RCE4: Test 1. Major-color-based screen content coding. Document JCTVC-P0108 (2014)
8. Guo, L., Pu, W., Karczewicz, M., Sole, J., Joshi, R., Zou, F.: RCE4: results of Test 2 on palette mode for screen content coding. Document JCTVC-P0198 (2014)
9. Rosewarne, C., Sharman, K., Flynn, D.: Common test conditions and software reference configurations for HEVC range extensions. Document JCTVC-P1005 (2014)