Algorithm for Coding Unit Partition in 3D Animation Using High Efficiency Video Coding Based on Canny Operator Segment

ZHAO Hong¹,², LI Jing-Bo¹,², ZENG Xiang-Yan³
¹ School of Computer and Communication, Lanzhou University of Technology, Lanzhou, 730050, China
² Information Center, Lanzhou University of Technology, Lanzhou, 730050, China
³ Department of Mathematics and Computer Science, Fort Valley State University, Fort Valley, GA, 31030, USA
Corresponding author, email: zhaoh@lut.edu.cn

ABSTRACT: The involvement of a huge volume of man-made models and frames in 3D digital animations results in high coding costs using high efficiency video coding (HEVC). Studies identified the unique features of textures in 3D models that aids in predicting the depth of coding units (CU). The fast algorithm for CU partition, which was based on the texture features of the 3D models, was proposed to decrease coding complexity and shorten coding time. Canny operator was introduced to facilitate the segmentation of key frames in 3D animation. The optimal depth of the current CU was predicted based on its location and that of adjacent CUs, as well as the CU located at the current position of the reference frame. To reduce prediction errors in complex areas, rate-distortion optimization method was used to conduct precise partition. Compared with the performance of the original high efficiency video coding algorithm, the proposed algorithm shortened encoding time by 39.20% when the peak-signal-to-noise ratio of brightness was decreased by 0.07 dB and the encoding bit rate was increased by 0.84%. These results indicate the practical utility of the proposed algorithm and its excellent performance in improving coding time.

Subject Categories and Descriptors
I.4.2 [Image Processing and Computer Vision]: Compression (Coding); I.4.6 [Image Processing and Computer Vision]: Segmentation – Edge and feature detection; H.5.1 [Information Interfaces and Presentation]: Multimedia Information Systems – Animations

General Terms
Algorithms, Performance, Verification

Keywords: High Efficiency Video Coding, Texture features of 3D animation, Depth prediction, Fast partition, Coding unit

Received: 19 May 2016, Revised: 7 July 2016, Accepted: 17 July 2016

1. Introduction

Unlike videos produced by camera, 3D digital animations include a large number of models and animation frames. The demand for realistic visual effects increased the degree of model refinement and data quantity of videos. High efficiency video coding (HEVC) is the most important method in coding 3D digital animations. HEVC involves costly and highly complicated computation of rate-distortion optimization method and requires long encoding time for dividing coding unit (CU). The huge data quantity and poor encoding performance of HEVC significantly restrict digital video experience and impair the management of multimedia information. Huge volumes of video data require vast storage spaces and cause difficulty in retrieving multimedia contents and transmitting them in the network. These issues indicate that studying the texture features of 3D digital animation, decreasing coding complexity, and shortening the encoding time of HEVC are crucial to the management of multimedia information.

2. State of The Art

The process of creating 3D digital animations involves the simulation of 3D space; in this process, the model and
scene are established according to the shape and size of the real object using professional software; the color and texture of the surface are assigned and the movement and deformation are designed according to the script; designs for intensity of light, location and movement are followed and the shooting scale of the virtual camera is established [1]. This process produces a series of consecutive video frames that are available for dynamic, real-time playback. The software such as 3D Max, Maya, Rhino and Direct3D, can be used to implement these works. Movies such as Madagascar, Finding Nemo and Monsters Inc. are typical examples of 3D animation. At present, the demand for realistic visual effects increased the degree of model refinement and expanded the volume of video data.

Since the introduction of HEVC in 2013, this algorithm has become the most important coding method in 3D animation; HEVC is mainly used in high definition (1920×1080) and ultra high definition (7680×4320) videos; the classic block-based, hybrid coding mode is still used; this process adopts a flexible quadtree structure to organize CU, prediction units (PU), and transform units (TU) [2]. CU, which is the basic unit, is squared-shaped. The width and height of this unit are raised to the power of 2 thereby resulting in four sizes, namely, 64×64, 32×32, 16×16, and 8×8. Each size respectively corresponds to a depth that ranges from 0 to 3 degrees. Given the quadtree structure of this unit, the recursive method is used in HEVC to divide CUs; this approach significantly improves coding efficiency but increases coding complexity [3]. In HM16.6, the HEVC standard test model is used to partition the 64×64 CU by recursively traversing the depth of each of unit according to the quadtree structure; all possible prediction models are then used to calculate the cost of rate-distortion; the minimum cost of rate-distortion will then be selected as the best CU division scheme to obtain the best coding performance; however, this approach leads to computational complexity and long coding time and therefore not feasible in practice [4].

The algorithms described in [5] and [6] decides whether further partition should be executed by setting the threshold and comparing the costs of rate-distortion when partitioning CUs; this approach does not consider the features of video texture and significantly damages the quality of video compression. In [7] and [8], the depth of CU is predicted according to intra prediction and inter prediction algorithms. However, these two kinds of algorithm ignore the early termination of CU partition at a certain depth. The present study introduces a CU partition algorithm for 3D animation that employs HEVC and Canny operator. This algorithm can fully utilize texture features to reduce coding complexity and shorten coding time.

The rest of the study is organized as follows. The texture features of 3D animations are measured in Section 3.1 by utilizing the luminance histogram. Texture smoothness is defined and the texture features acquired through sufficient animation frames. Section 3.2 introduces the Canny operator, which is applied to improve the speed of CU partition according to the distinct video features of 3D digital animation. Section 3.3 describes our algorithm in detail. Section 4 discusses the experimental results, including the comparison of different resolution sequences, quantization parameter (QP) values, and computational performance. Section 5 concludes our advantages and disadvantages of the algorithm and confirms our research targets.

3. Methodology

3.1 Features of 3D animation videos

The surface details or textures of objects used in 3D models are created using professional applications. For example, the construction of slope models using Direct3D involves bitmaps of grass, soil, and rock textures that are placed on the surface of graphic primitives to form a realistic vision of a mountain [9]. These artificial textures designed by computer applications have their unique texture features. This study defines texture smoothness as the degree of aggregation of pixels distributed around the highest frequency level of brightness.

Luminance histogram represents the probability of each intensity level in the frame [10]. The highest frequency level of brightness \( B_{\text{popular}} \) must exist in luminance histogram. The ratio of a certain range of the pixels distributed around \( L \) to the total number of pixels is defined as the texture smoothness of largest coding unit (LCU), which is denoted as \( S \) in Equation (1):

\[
S = \frac{1}{M} \times \sum_{i=B_{\text{popular}}-10}^{B_{\text{popular}}+10} N_i \times 100, \quad i \in 0,1,\ldots,L-1
\]

where \( M \) means the total number of pixels within LCU; \( N_i \) is the total number of pixels at brightness level \( i \) in the LCU; \( L \) is the level of total brightness. \( L \) is equal to 256 when the 8-bit quantization of histogram statistics is used. Figure 1 shows that the texture smoothness is 80 for the sky model area in the 460th frame of the video BigBuckBunny. Texture is relatively simple. Texture smoothness is 24 in the cross-region of the bird model and the branches, which indicates complex textures. Texture smoothness is 36 in the cross-area of the butterfly and sky models of the 2647th frame. The textures are relatively complicated. Texture smoothness is 41 and textures are relatively simple in the region of the rabbit model.

Figure 1 shows LCUs with high texture smoothness have highly similar internal textures. Other LCUs with low texture smoothness in the cross-regions of bird and sky, bird and branches, and branches and leaves have low internal similarity. \( TS \) is then assigned as the degree of similarity of a CU texture similarity. \( TS \) calculation is shown as Equation (2):

\[
TS_{\text{position}} = \sum \frac{S_{\text{position}}}{T_{\text{position}}}, \quad \text{position} \in [0,1,2]
\]
where \( \text{position} \) represents different statuses; \( S_{\text{position}} \) is the texture smoothness of position relations; \( T_{\text{position}} \) represents the number of this type of position relation. \( \text{position} = 0 \) means that CU textures are located in the same model; \( \text{position} = 1 \) means that CU textures are at the intersecting regions of the two models; \( \text{position} = 2 \) means that CU textures are in the cross area of more than two models.

As shown in Table 1, the 5th to 11730th frames of BigBuckBunny are used to complete our statistics. \( \text{position} = 0 \) indicates \( TS \) of 82, which means that textures in the same models have highly similar internal texture. \( \text{position} = 1, 2 \) respectively indicates \( TS \) of 46 and 23, which means that textures in different models have low textural similarity.

The statistics indicate that the complexity of texture of the internal models tends to be similar and the connection areas of different models evidently differ with the surrounding area and their similarity is low.

In HEVC, CUs with simple textures are allocated with low levels of depth, whereas CUs with complex textures have high levels of depth [11]. This conclusion can be used to estimate the initial depth of CU in coding.

### 3.2 Prediction of CU depth based on texture features

In HM16.6, the HEVC test model, the depth of CU ranges from 0 to 3. High levels of depth are suitable for processing image areas that contain a high number of complex textures. Low levels of depth are used to process smooth image areas. The depth of current CU (Cur_CU) is closely related to the CU adjacent to the left border of Cur_CU (Left_CU), the CU adjacent to the up border of Cur_CU (Up_CU), and the CU in the reference frame with the same position as Cur_CU (Col_CU). Figure 2 shows that the optimum depth of current CU has a close relationship with Left_CU’s left_dep, which is the minimum depth of CU symmetrical to the left line of the 8×8 pixel block, Up_CU’s up_dep, which is the depth of the minimum CU symmetrical to the up boundary line of the 8×8 pixel block, and Col_CU’s col_dep, which is the minimum depth of CU located in the same location in the reference frame of the 8×8 pixel block [12].

To predict Cur_CU depth, a Canny operator for edge detection is initially introduced to detect the edge of models.
in the key frames. Edge is the end of a region and the beginning of another area. Edge detection can be used to partition the models of the frame and to determine the location relationship between Cur_CU, Left_CU, and Up_CU. As shown in Figure.3, Canny operators can disregard areas with smooth texture because they are not important in contour features changes; thus, only the edge of the models is retained; the advantages of Canny operators improve detection rate in texture-rich areas and identification accuracy of edge-point location [13]. The Canny algorithm requires complex computation, but it has strong denoising ability, highly accurate edge location, continuous advantage, and ability to separate the frame model [14].

The Canny operator can separate the models. When the first step of this process is completed, the following prediction approach is applied:

1. Determine whether Cur_CU, Left_CU, and Up_CU are located in the same model. If they are located in the same mode as shown as Figure.4, determine whether left_dep=up_dep=n. If this condition is obtained, the textures of Cur_CU are considered similar to these CUs. The initial depth of Cur_CU can be predicted as n. The division of Cur_CU is terminated.

2. If Cur_CU and Left_CU are located in the same model but Up_CU is not, the position relationship between Cur_CU and Up_CU is considered relatively complex as shown in Figure.5.

The edge boundary of the model may be intruded into Up_CU area or Cur_CU area. The complexity of textures
If \( \text{left}_\text{dep} = \text{col}_\text{dep} = n \), the textures of \( \text{Cur}_\text{CU} \) are similar to that of \( \text{CU} \). The initial depth of \( \text{Cur}_\text{CU} \) is predicted as \( n \) and \( \text{Cur}_\text{CU} \) division is terminated.

If \( \text{left}_\text{dep} \neq \text{col}_\text{dep} \), the textures around \( \text{Cur}_\text{CU} \) change significantly.

The initial depth of \( \text{Cur}_\text{CU} \) is predicted as 
\[
\frac{\text{left}_\text{dep} + \text{col}_\text{dep}}{2}
\]

to ensure the video high quality.

(3) If \( \text{Cur}_\text{CU} \) and \( \text{Left}_\text{CU} \) are not located in the same model but \( \text{Up}_\text{CU} \) is included as shown in Figure 7, \( \text{Cur}_\text{CU} \) and \( \text{Up}_\text{CU} \) are compared.

If \( \text{up}_\text{dep} = \text{col}_\text{dep} = m \), complexity of texture in \( \text{Cur}_\text{CU} \) is similar to that in \( \text{CU} \). The initial depth of \( \text{Cur}_\text{CU} \) is predicted as \( m \), and \( \text{Cur}_\text{CU} \) division is terminated.
If $\text{up}_\text{dep} \neq \text{col}_\text{dep}$, textures around Cur_CU change significantly. The initial depth of Cur_CU is predicted as 

$$\left\lfloor \frac{\text{up}_\text{dep} + \text{col}_\text{dep}}{2} \right\rfloor$$

to ensure the video quality.

(4) If Cur_CU, Left_CU, and Up_CU are not located in the same model shown as in Figure 8, rate-distortion optimization method is used to ensure precise partition.

Figure 8. Cur_CU, Left_CU and Up_CU in different model

All judgments and predictions that make full use of the texture features found in this study have been completed.

3.3 Complete process of the fast algorithm for CU partition

Figure 9 shows the flow chart of the partition algorithm for CU based on Canny operator. Suppose the depth of Left_CU is n, the depth of Up_CU is m and the prediction process for the initial depth of CU is:

Step 1: Canny operator is introduced to detect the edge of the model in LCU. Determine whether Cur_CU, Left_CU and Up_CU are located in the same model. If they are, implement Step2, otherwise go to Step3.

Step 2: Determine whether $\text{left}_\text{dep} = \text{up}_\text{dep} = n$. If they are equal, the depth of Cur_CU is predicted as n then proceed to Step8. If they are not equal, the initial depth of Cur_CU is predicted as 

$$\left\lfloor \frac{\text{left}_\text{dep} + \text{up}_\text{dep}}{2} \right\rfloor$$

then proceed to Step8.

Step 3: Determine whether Cur_CU and Left_CU are located in the same model. If they are, proceed to Step4, otherwise, go to Step5.

Step 4: Determine whether $\text{left}_\text{dep} = \text{col}_\text{dep} = n$. If they are equal, the depth of Cur_CU is predicted as n then proceed to Step8. If the two are not equal, the initial depth of Cur_CU is predicted as 

$$\left\lfloor \frac{\text{left}_\text{dep} + \text{col}_\text{dep}}{2} \right\rfloor$$

then proceed to Step7.

Figure 9. Flow chart for the depth prediction of CU
Step 5: Determine whether Cur_CU and Up_CU are located in the same model. If they are, proceed to Step 6; otherwise, go to Step 7.

Step 6: Determine whether up\_dep=col\_dep=m. If they are equal, the depth of Cur_CU is predicted as m, then proceed to Step 8; if they are not equal, the initial depth of Cur_CU is predicted as \[ \frac{up_{\_dep} + col_{\_dep}}{2} \], then proceed to Step 7.

Step 7: Use the rate-distortion optimization method to ensure precise judgment.

Step 8: End of process

The algorithm first introduces the Canny operator to split the texture of animation. If Cur_CU, Left_CU and Up_CU are in the same model according to their location relationship, the depth of Cur_CU can be predicted as similar to that of Left_CU and Up_CU. If Cur_CU is located in the same model as Left_CU or Up_CU, the depth of Cur_CU cannot be easily predicted simply because such a level of depth should be predicted according the depth of Col_CU. The rate-distortion optimization method is used to obtain precise partition of areas with complex texture.

4. Result Analysis And Discussion

The optimized algorithm was integrated with HM16.6 to verify the validity of the algorithm. Performance of the algorithm was compared with that of the original coding algorithm at the mode of All Intra (AI) frame in encoder_intra_main configuration file. The two algorithms were compared in terms of coding performance and coding time. The platform configurations for the experimental test were Intel Core i5 processors, 2.30 GHz main frequency, 6 GB memory, 64-bit Windows 7 operating system, and development tools Microsoft Visual Studio 2010. The experiment adopted four standard resolution YUV test sequences, namely, ElephantsDream_704×576, BigBuckBunny_1024×768, ElephantsDream_1920×1080, and the low-resolution test sequence BigBuckBunny_352×288 provided by the Joint Collaborative Team on Video Coding (JCT-VC). The QP value adopted in the experiment was set as 22, 27, 32, and 37 respectively. Coding performance was measured by bit rate (BR) and peak signal-to-noise ratio (PSNR). Encoding time was measured by encoding time (ET). The increment of PSNR (\( \Delta PSNR(Y) \)), coding bit rate increment (\( \Delta BR \)), and encoding time (\( \Delta ET \)) were used as evaluation index contrasted with the original coding algorithm for HM16.6 [15]. These equations are given as:

\[ \Delta PSNR(Y) = PSNR(Y)_P - PSNR(Y)_{HM16.6} \quad (3) \]

\[ \Delta BR = \frac{BR_P - BR_{HM16.6}}{BR_{HM16.6}} \times 100\% \quad (4) \]

\[ \Delta ET = \frac{ET_P - ET_{HM16.6}}{ET_{HM16.6}} \times 100\% \quad (5) \]

<table>
<thead>
<tr>
<th>Test sequence</th>
<th>QP</th>
<th>( \Delta PSNR(Y)(dB) )</th>
<th>( \Delta BR(%) )</th>
<th>( \Delta ET(%) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElephantsDream_1920×1080</td>
<td>22</td>
<td>-0.05</td>
<td>0.71</td>
<td>-30.87</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>-0.07</td>
<td>0.47</td>
<td>-36.52</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>-0.07</td>
<td>0.15</td>
<td>-41.12</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>-0.08</td>
<td>0.21</td>
<td>-43.17</td>
</tr>
<tr>
<td>BigBuckBunny_1024×768</td>
<td>22</td>
<td>-0.06</td>
<td>1.02</td>
<td>-36.69</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>-0.07</td>
<td>0.94</td>
<td>-39.14</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>-0.08</td>
<td>0.81</td>
<td>-42.29</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>-0.09</td>
<td>0.63</td>
<td>-47.43</td>
</tr>
<tr>
<td>ElephantsDream_704×576</td>
<td>22</td>
<td>-0.07</td>
<td>1.25</td>
<td>-32.15</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>-0.07</td>
<td>1.10</td>
<td>-36.54</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>-0.08</td>
<td>1.04</td>
<td>-41.11</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>-0.09</td>
<td>0.76</td>
<td>-43.34</td>
</tr>
<tr>
<td>BigBuckBunny_352×288</td>
<td>22</td>
<td>-0.06</td>
<td>1.31</td>
<td>-33.63</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>-0.07</td>
<td>1.12</td>
<td>-36.98</td>
</tr>
<tr>
<td></td>
<td>32</td>
<td>-0.09</td>
<td>1.05</td>
<td>-42.37</td>
</tr>
<tr>
<td></td>
<td>37</td>
<td>-0.08</td>
<td>0.89</td>
<td>-43.87</td>
</tr>
<tr>
<td>Average</td>
<td>-</td>
<td>-0.07</td>
<td>0.84</td>
<td>-39.20</td>
</tr>
</tbody>
</table>

Table 2. Comparison of performances of HM16.6 coding algorithms
where $PSNR(Y)$, $BR$, and $ET$, are the $PSNR$, $BR$ and $ET$ of the proposed algorithm respectively. $PSNR(Y)_{HM16.6}$, $BR_{HM16.6}$, and $ET_{HM16.6}$ are $PSNR$, $BR$ and $ET$ of the original HM16.6 algorithm.

As shown in Table 2, the different QP values of the proposed algorithm are compared with that of the original HM16.6 encoding algorithm. With the QP value increased, the fluctuation range of encoding bit rate decreased, and the saved encoding time gradually increased at the same test sequence. In terms of overall coding efficiency, encoding time was reduced by an average of 39.20%, and the fluctuation range of the $PSNR$ was reduced by only 0.07 dB. Coding complexity was reduced by the application of this algorithm. Encoding time was shortened with minimal effect on video quality.

Coding efficiency was significantly improved by the fast algorithm for CU partition based on Canny operator. The depth of CU was predicted through the segments, judgments, and predictions of the fast algorithm. Calculations for the cost of rate-distortion are skipped partly and additional calculations are avoided.

5. Conclusion

Focusing on coding problems involving HEVC which does not make full use of texture features when partitioning CUs and coding 3D animations thereby resulting in highly complex and inefficient computations, this study examined texture features and predicted the depth of CU in HEVC of 3D animations. The conclusions obtained in this study are as follows.

(1) This study examined sufficient animation frames and found that the man-made textures of 3D models are composed of unique texture features. CUs located in the same model tend to have similar depths. However, CUs located in different models have distinct depths.

(2) Texture features were applied in the partition of CUs, which is the main of this study. The Canny operator was introduced to separate the edge of the models and divide texture. The location relationship of Cur_CU, Left_CU, and Up_CU was determined, and the depth of Cur_CU was predicted. Unnecessary CU traversals were skipped and comparisons of rate-distortion were reduced.

(3) This algorithm considered all kinds of the location relationships. The depth of CU can be predicted effectively in both flat and complex area. This algorithm significantly reduced coding time, however, with the increase of encoding bit rate and decrease of $PSNR$, which was indicated acceptable overall performance. This result suggests that the algorithm has high practical utility in 3D digital animations. This algorithm offers a trade-off between the qualities of video compression and coding time. The algorithm predicts the depth of CU partition according to the texture characteristics of 3D animations. However, these predictions cannot consistently provide accurately predictions and certain probability of misjudgments exists. Low levels of depth in complex CU areas will result in coding distortion, which will affect the quality of videos. However, these effects are acceptable according to experimental results. Given that the current study only measured the algorithm on HM16.6, future studies should verify the effects of this algorithm when applied with other HEVC models. Coding for 3D animation should also be optimized from other aspects.

Acknowledgements

This work was supported by the Natural Science Foundation of China under Grant No. 61262016, the University Foundation of Gansu Province under Grant No. 04-056001, the Natural Science Foundation of Gansu under Grant No. 1208RJZA239, and the Technology Project of Lanzhou under Grant No. 2012-2-64.

References


