ARBITRARY-SIZED MOTION DETECTION IN SCREEN VIDEO CODING

1Tao Zhang, 2Xun Guo, 2Yan Lu, 3Shipeng Li, 3Siwei Ma, 4Debin Zhao

1Harbin Institute of Technology, Harbin, China 150006
2Microsoft Research Asia, Beijing, China 100080
3Peking University, Beijing, China 100871

ABSTRACT

In real-time screen remoting system, frame rate is one of essential factors that affect user experience. Therefore, how to compress diversity of screen contents fast and efficiently is very important. Existing video codecs such as H.264 are always used in such a system for screen compression. However, arbitrary-sized regions with large motion always exist in typical screen content videos, which lead to a lower encoding speed and higher bit-rate, thereby decrease the frame rate. This paper proposes an efficient motion detection algorithm, which is fast and efficient for large motion regions. In specific, a region-based motion detection is used to find motion vectors instead of traditional block based motion estimation. Then the motion vectors are utilized by H.264 encoder for normal motion compensated prediction. Experimental results show that the proposed algorithm can reduce both encoding time and bit-rate significantly.

Index Terms—Screen video coding, motion detection, H.264

1. INTRODUCTION

With the development of Internet and digital devices, more and more remoting applications occur, such as screen sharing and interaction. In typical application, screen content in remote computer is captured and transmitted to local users instantly to realize desktop sharing and duel-side interaction. In our previous work, a screen virtualization in cloud computing [1] is proposed, in which the screen rendering is performed in the cloud and delivered as images to the client for interactive display. As a result, a fast and efficient encoder is always needed at the remote computer for compressing and transmitting the screen video in real-time with high quality and low bit-rate.

In general, the screen contents can be classified into two categories: text and image. Much efforts has been made on processing these two types of contents separately. Existing codecs such as JPEG and H.264 are always used to exploit spatial and temporal correlations among frames. Said et al. [2] proposed a block-based method which classifies blocks into text blocks and picture blocks, then compresses text blocks using JPEG-LS and picture blocks using JPEG. Li et al. [3] proposed a visually lossless method to compress scanned documents. Querioz et al. [4] described a layer based method, in which compound image is segmented into multiple layers according to content and processed with different coding algorithms. To fully leverage existing video codec, Wang et al. proposed a hybrid video compression scheme [5], in which text coding method is combined with H.264 to process different layers of compound video. All these methods aim at improving the efficiency of existing codecs from the viewpoint of spatial characteristics.

Considering the general requirement of thin-client, it would be better if different screen contents can be efficiently processed by a single compression method. This paper proposes a H.264-compatible screen video coding scheme so that the existing well optimized H.264 encoders and decoders can be fully leveraged in terms of both software and hardware. Different from the previous methods, the proposed scheme aims at solving a temporal problem other than spatial ones. That is, the arbitrary-sized large motion regions, which always occur in screen video contents as shown in Fig. 1, will lower the performance of H.264. On the one hand, if the search range of motion estimation (ME) is very large, the encoding time would increase. On the other hand, if the search range is small, inter-frame correlation exploitation will be inefficient, which leads to a high bit-rate.

To tackle this problem, a motion detection algorithm is proposed to help the motion compensated prediction of H.264. Specifically, a very fast region-based motion detection is used to find motion vectors instead of block matching in H.264, so that large motions can be efficiently found with a low complexity.

The rest of this paper is organized as follows. Section 2 describes motions in screen video content. Section 3 explains...
the proposed screen video coding scheme and arbitrary-sized motion detection. The experimental results are shown in section 4 and section 5 draws conclusions.

2. MOTIONS IN SCREEN VIDEO

In traditional nature video sequences, the motion is usually generated by camera panning and object moving. And the captured images are always affected by sampling noise and other conditions like lightening. As a consequence, the true motion in nature video is usually not RD-optimal. However, for screen video content such as webpage, documents and slides, the situation is very different.

![Fig. 2 Temporal correlated regions in consecutive frames. (a) The previous frame. (b) The current frame. (c) Part of the previous frame in motion region. (d) Part of the current frame in motion region.](image)

There are several different characteristics for the motion in typical screen contents compared to traditional nature video. Firstly, the motion in screen contents is most likely generated by user interaction such as page scrolling, dragging and touching. Given the high screen resolution these days, the motion gap between two consecutive frames is relatively big, especially when network condition is bad and some frames have to be dropped. Secondly, the motion is usually region-based and arbitrary-sized. And the region always has regular shape such as rectangle. Thirdly, the screen images are rendered by system and captured directly from the screen. There is no introduced noise like that of the nature video. Therefore, it’s very likely that the corresponding motion regions in consecutive frames have the same pixel values. Although in some system like Windows 8, temporally correlated regions may have different values due to rendering mechanism, i.e. regions within red boxes in Fig. 2, the difference can be handled more easily than nature noise.

The above characteristics lead to a fact that the motion search in screen content video can be faster and more efficient than that of nature video. Firstly, motion search can be performed in original pixels, which can guarantee true motion is also RD-optimal motion. Secondly, hierarchical motion detection with pixel level, block level and region level search is possible so that the complexity of motion estimation can be significantly reduced. Thirdly, to a large extent, motion detection can be decoupled with the search range of ME so that large motion would not lead to a high encoding complexity.

3. ARBITRARY-SIZED MOTION DETECTION

3.1 The proposed screen video coding scheme

Fig. 3 depicts the pipeline of the proposed screen video coding scheme, which consists of host end and client end. Screen video sequences are captured and transmitted from the host to the client for interactive display. In the host, motion detection is performed between the current frame and the previous frame. Then the detected motions are used by H.264 encoder for the inter-prediction of the current frame.

![Fig. 3 The pipeline of the proposed screen video coding.](image)

3.2 Motion detection

We propose an efficient motion detection scheme, which hierarchically uses pixel level, block level and region level matching to find the motion regions with arbitrary sizes. In specific, a couple of feature points are firstly detected in each 16x16 macroblock (MB) in the current frame. The corresponding feature points are then searched in the previous frame. If the corresponding feature points are found, a region-based motion search is used to find the final motion regions around the feature points. The details of the whole process are described in the following sub-sections.

3.2.1 Feature point matching

A very fast and efficient method is used to find the feature points. In specific, each pixel of a MB, e.g. the black point in Fig. 4, is checked and compared with its surrounding pixels, i.e. the orange points, which are left or up to it. If the differences between the current pixel and the surrounding pixels are larger than a threshold Td and the differences among the surrounding pixels are smaller than another threshold Tc, the pixel will be selected as a feature point.

![Fig. 4 Feature point selection.](image)
When a pixel is determined as a feature point, several pixels right to the current pixel, i.e. blue points in Fig. 4, are used together with this pixel to calculate a Hash value for later feature point matching process.

After the feature point selection, the feature point matching process begins. For each feature point in the current frame, all the feature points within a search range in the previous frame are checked to find out if it has a pair. Specifically, if there is only one feature point in the previous frame which has the same hash value with the feature point in current frame, they are counted as a feature point pair.

3.2.2 Motion region detection

If there are feature point pairs, motion regions will be grown around the feature points gradually. In specific, two initial cores, i.e. the black box in Fig. 5, are firstly chosen around the two corresponding feature points. Then the initial cores are extended region by region toward four directions. When the corresponding extended regions are the same, they are merged to the initial cores. As shown in Fig. 5, the initial core is extended to the left first with a step size of 8-pel until there are no matched extended regions any more. Then other directions perform the same process.

In this process, the basic check unit is 8x8 block. Only if all the 8x8 blocks in the extended regions are matched blocks, the extended regions are taken as matched regions. To lower the complexity, only half of the 64 pixels are compared for each 8x8 block. In even rows, only pixels in odd columns are compared, and in odd rows, only pixels in even columns are compared. The matching process for each 8x8 block is described as follows.

**Step 1** Initialize the difference count as 0, which will be used to judge whether the two corresponding 8x8 blocks are matched.

**Step 2** Check the difference between the pixels of a pixel pair. If the difference exceeds a threshold, the matching process will be early terminated and this block will be marked as unmatched block. Otherwise, continue to check whether they are similar. If not, the difference count is increased by 1.

**Step 3** If the difference count is larger than 5, the matching process is stopped and the current 8x8 block is marked as unmatched block. If it is not the last pixel to be checked, step 1 is performed to check the next pixel in the current block. While if it is the last one, the matching process is stopped and the current block is marked as matched block.

Two pixels are taken as similar if their difference is lower than a threshold. In some system like Windows 8, high differences always occur in the pixels with high gradient, although they are temporally correlated. Therefore, a simple gradient detection will be performed for each pixel pair. A higher difference threshold is used if their gradient is high, while a lower difference threshold is used if their gradient is low.

3.3 H.264 encoding

When the global motions for the current frame are detected, they are passed to H.264 encoder to help the motion estimation. In a normal motion estimation process of H.264, different partitions, i.e. 16x16, 16x8, 8x16, 8x8, 8x4, 4x8 and 4x4, will be separately searched to achieve the best motion compensated prediction. In the proposed algorithm, MBs in the detected motion regions and MBs out of the detected motion regions use different mode decision processes. For the MBs in the detected motion regions, the motion vectors are set as the detected motion. In this case, only 16x16 partition is considered due to the consistency of the motion. For the MBs out of the detected motion regions, normal motion estimation and mode decision process are performed.

4. EXPERIMENTAL RESULTS

In order to justify the proposed algorithm, a couple of experiments are conducted. Because the screen video should be captured and compressed in real-time, we use the well optimized X264 video codec [6] as the benchmark, as well as the base for the proposed algorithm. Four typical screen scenarios are tested: pure document (pdf), compound images (webpage), touch UI in Windows 8 (metro) and full screen natural video (video). All of the test sequences have the resolution of 1280x768, and each sequence contains 150 frames. In pdf and webpage, the motion mainly comes from scrolling up and down. In metro, the motion is caused by finger slipping, which is relatively slow and smooth. While in video, the motions are traditional ones. Considering the balance of the encoding complexity and RD performance, we choose a pre-defined parameter set for X264 with a structure of IPPP and a fast motion estimation of UMH, which has been tested in a real screen sharing system. In order to reach the goal of real-time encoding, searching ranges for different test sequences are different. For webpage and pdf, the search ranges are set as 32 and 64, respectively. And for metro and video, search ranges are both 16.
Rate-distortion performances of H.264 and the proposed method

Fig. 6 shows the comparison of RD performance for X264 and proposed algorithm. For the sequences with scrolling motions like webpage and pdf, the efficiency of the proposed algorithm is very high, because the corresponding regions have exactly the same pixel values. Over 30% – 40% bit-rate reduction is observed. For metro sequence, the gain is not so significant, which has about 20% bit-rate reduction. That is mainly because the difference between pixel pairs mentioned in Section 2 lowers the detection accuracy. For video sequence, there is no gain as expected, because the proposed algorithm is not designed for nature video.

Table I shows the comparison of average encoding time for each frame, which is collected in the PC with CPU E5400 @ 2.0 GHz and 4 GB RAM. For metro, webpage and pdf, 10% – 30% encoding time reduction can be achieved. While for video, encoding time is a little bit increased, because the proposed motion detection doesn’t work at all. It should be noted that the encoding time is only for compression part. In practical system, there are also other operations which consume time such as capturing, transmission and controlling. Therefore, the compression needs to be as fast as possible to guarantee a high frame rate. 30% encoding time reduction may mean 30% frame rate increase, which can significantly improve the user experiences.

5. CONCLUSIONS

In this paper, a novel arbitrary-sized motion region detection algorithm was proposed to be used together with H.264 encoder to improve the performance for screen video coding. Experimental results show that, compared with the well optimized X264 encoder, the proposed algorithm can achieve significant performance gain in terms of both bit-rate reduction and encoding complexity.

6. REFERENCES