EE669 Homework #5

Yifan Wang
wang608@usc.edu

November 6, 2019

1 Written questions

MPEG2 has three type of frames (I, B, P). In each frame, it uses Y, Cb, Cr to represent a pixel. There are several modes of different sample rate for each attribute like ($4:4:4$, $4:2:0$ etc.). For I frames, we use 8×8 discrete cosine transform to compress them which is the same way as jpeg compression, these frames would have relative higher bit rate and higher quality than P or B frames. They will be used as reference frames.

For P frame, it would be used as a reference frame, as well. On the other hand it also relays on I frame to encode. To get it, we first need to reconstruct I frame. And the P frame is divided into 16×16 blocks and search the best match block in reconstructed I frame. The offset is called motion vector (generally it is zero). Two matched blocks would not match perfectly in most cases, so that there is a residual vector which records the difference of corresponding pixels in block. It would be appended after motion vector and represent this block.

For B frame, it would relative similar to P frame, only difference is that B frames use the picture in a successive reference frame as well as the previous reference frame. So that it would have a much higher compression ratio. And it would never be used as reference frames.

1.1 Integer transformation

Unlike MPEG2 using 8×8 DCT, H.264 use integer transformation (similar to DCT but has lower complexity) which would have higher compression ratio. Besides, it would have an effect of reduce ringing effect cause by Gibbs phenomenon from truncating high frequency components during quantization step. Using a smaller block size (4×4) and simple integer processing provides an exactly specified decoded result. So that each correlated region would be compressed more efficiently. Adaptive encoder selection between the 4×4 and 8×8 transform block sizes for the integer transform operation.

1.2 Inter Picture Prediction

1.2.1 Multiple Reference Frame

H.264 uses previously encoded frame as references in a more flexible way than MPEG2. It would support at least 16 reference frames. While in previous MPEG2 we can only use one reference frame for encoding P frames while two for B frames. Allowing to use more reference frames can make it easier to find some highly matched blocks and use a weighted sum of these blocks from each reference block to get block prediction. So that it can use multiple motion vector from different reference frame for one block to make prediction more accurate especially in some scenes where occurs changing of plane, zoom, new object occurs or some periodic motion .

1.2.2 Variable Block Size Motion Compensation

It can use block size from 16×16 to 4×4 (others like 16×8, 8×16, 4×8 etc. can be used as well, 9 modes for intra 4×4 and 4 modes for intra 4×4 luma blocks) which can help to segment moving object

more precisely. A small block can also achieve a low difference with reference block, resulting less data to be compressed in motion vector. While in MPEG2 it only uses 16×16 blocks find matched blocks. Motion vector for each partition region can point to different blocks in different reference frame.

1.2.3 Quarter Bit

MPEG2 would accept half bit to find matched block which is realized by using interpolation. In H.264 it can accept quarter pixel while finding matched blocks. So that prediction on motion vector would be more precise especially in moving regions resulting better video quality and smaller file size. However, it would increase the complexity and time during encoding. Besides, the way of interpolation is improved in H.264. Previously MPEG2 use bilinear interpolation while H.264 uses a five-tap filter which can increase accuracy of predicted interpolated pixel.

1.2.4 Enhanced Direct/Skip Macro Block

Skip and direct mode frequently occurs in B frames and can significantly reduce number of bits to be coded. It acts through following way, one block would use same motion vector as successive frame while successive fame use motion vectors from previous reference frames. This method is super nice when meeting some scenes without any moving objects, it would help to reduce final compressed file size.

1.3 Intra Prediction

In MPEG2 it do not have any intra predictions, while H.264 has 9 modes of intra predictions. It can better utilize information inside a frame and reduce final compressed file size.

1.4 CABAC and CABAC

In MPEG2, it uses Huffman coding as entropy coding methods. It is changed to context-adaptive binary arithmetic coding (CABAC) and context-adaptive variable-length coding (CAVLC) which can encoding more efficiently. CABAC can achieve higher compression ratio than CAVLC with the cost of more time and memory. But we are flexible to choose which one to use while doing encoding.

1.5 De-blocking Filter

H.264 uses an in-loop de-blocking filter which helps to prevent blocking artifacts and result a higher image quality and better visual experience. While MPEG2 does not have such de-blocking filters, it would have more blocking effect especially in some videos including fast moving scenes. Obviously, this part would increase image quality by increase complexity of algorithm.

1.6 Two Pass

In x264, it has a two pass mode to control rate. It would run encoding algorithm two times, the first pass would examine the whole file to get statics. Second pass would base on previous statics to better assign bitrate under certain conditions. It would slightly increase the video quality, but time for compression would be doubled.

1.7 Extra Motion Vector Initialization

In x264, it would use up to 10 motion vector from neighbourhood and reference frame to be candidates for initialization motion vector for current micro-block. This method would help to better find a lowest-cost motion vector and resulting a better estimation of motion vector. Otherwise, optimal motion vector estimation would likely to be stacked in local minima.

1.8 More Parameters

H.264 offers more parameters that can be self-set during encoding process. These parameters can give user more control to final bit rate and video quality.

2 Motion estimation in x264

2.1

2.1.1 Diamond

1. Start with large search pattern (center pixel with step size 2).
2. Find the pixel which minimize the cost function in the search pattern.
3. If the minimized one is not the center pixel, move center pixel to this pixel and search the other pixels which have never searched before using same way as ($Step2$).
4. If minimized pixel is the center pixel, then change to a smaller search pattern (half the step size) repeat $Step2$ and $Step3$.

Step size here means Hamming Distance between pixel's location with center pixel. It would achieve similar searching result as exhausted searching while saving lots of searching power and time.

2.1.2 Hexagon

Hexagon search using the same searching rule as Diamond search, only difference is that it change the searching patten from diamond shape to a hexagon shape which has less pixels than diamond, so that the searching result may not be the best one but would close to optimization. However, due to the less pixels, it would save more time than diamond search.

2.1.3 Uneven Multi-Hexagon

1. Initial search point prediction as spatial median prediction, upper layer prediction, neighboring reference frame prediction and temporal prediction. Use them to predict current block's motion vector.
2. Use asymmetrical cross search with an early termination scheme to search these pixels.
3. Use uneven multi-hexagon grid search. It would include two search patterns one is square search pattern and a 16 point hexagon search pattern;
4. Then it would extend hexagon based search. Include one hexagon search pattern and a diamond search pattern along with early termination.
5. Using this method would slightly increase running time while increase searching accuracy as well.

2.1.4 Successive Elimination Search

Let $P^t(i,j)$ be a pixel locate (i,j) at frame t and $P^{t-1}(i-x,j-y)$ as its reference pixel in previous frame. Motion vector is (x,y). For the in-equality we can have

$$\|P^t(i,j)\| - \|P^{t-1}(i-x,j-y)\| \leq \|P^t(i,j) - P^{t-1}(i-x,j-y)\| \tag{1}$$

For all pixels in a $N \times N$ blocks, we can get

$$\sum_{i=1}^{N}\sum_{j=1}^{N}\|P^t(i,j)\| - \sum_{i=1}^{N}\sum_{j=1}^{N}\|P^{t-1}(i-x,j-y)\| \leq \sum_{i=1}^{N}\sum_{j=1}^{N}\|P^t(i,j) - P^{t-1}(i-x,j-y)\| \tag{2}$$

The right part of $Equation(2)$ is a summation of absolute difference of micro block in current frame and reference frame. Mean of absolute difference (MAD) is the normalized version of $Equation(2)$.

$$\frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\|P^t(i,j)\| - \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\|P^{t-1}(i-x,j-y)\| \leq \frac{1}{N^2}\sum_{i=1}^{N}\sum_{j=1}^{N}\|P^t(i,j) - P^{t-1}(i-x,j-y)\| \quad (3)$$

In $Equation(3)$, first part of left side is mean of pixel value in this block, R. While second part is mean of similar block in reference frame, $M(x,y)$. We can get two in-equality equations

$$R - M(x,y) \leq MAD(x,y) \quad (4)$$

$$M(x,y) - R \leq MAD(x,y) \quad (5)$$

.

We start from a initial motion vector $M(m,n)$ and need to find a smaller $MAD(x,y)$. Then we can solve equation by change position in $Equation(4,5)$

$$R - MAD(x,y) \leq M(x,y) \leq R + MAD(x,y) \quad (6)$$

Only find motion vector that meets $Equation(6)$ and do it repeatedly until find the best one.

2.2

Motion search would cost lots of time in encoding, in x264 it provides several motion search methods by using prefix $--me$ which supports DIA, HEX, UMH, ESA and TESA. They are used for different speed and quality of encoding. For fast compression while low quality DIA is used in its presetting. While for median speed and quality UHM is used to provide a reasonable speed and quality. For slowest compression but higher quality TESA is used.

All the searching methods use a modified motion vector initialization. They would consider 10 candidates consists of 4 neighbours locations and 3 motion vectors from reference frame, $(0,0)$ MV, median MV and temporal direct MV. And compare the sum of absolute differences of each candidate and choose the best one. One step of diamond search is applied to $(0,0)$ MV and median MV. If block size is 4×4 apply hexagon search rather than UMH. Then apply diamond search to best MV to 4 nearest neighbours. Early termination would happened when previous step did not find a new best MV. But before break, it would enlarge the step size to 2 and do another diamond search if remains cannot find a new best MV, then break.

Adaptive radius depends on magnitude of best SAD and the smoothness of motion field. Default is 16 and it would decrease to 12 if all predictors are similar, or increase to 24 on the other hand. Then it would run a 5×5 exhaustive search.

The is similar to change learning rate in ML which can accept a large learning rate resulting a faster speed, when nearly to converge, it would change to a small learning rate when approaching the optimal value to fine tune the result.

2.3

```
# time command is used to calculate running time.
time ./x264 --output video.mkv \
        --input-res 352x288 foreman_cif.yuv \
        --psnr --me $ME_METHOD
```

Early termination happens in $encoder/me.c$ ($Figure1(a)$) where it is under the switch in case when using UMH. Besides, in x264's documentation, it said under prefix $--subme$ if set value as 11 it would disable all early terminations. (-11: Full RD: disable all early terminations,)

Extra initialization ($Figure1(b)$) for motion vector would not be used if $--subme$ is less than three, however, $--subme$ would affect other part of encoding and resulting a lower quality as well. So that we need to modify the if condition to change the way of initialization for a larger $--subme$.

```
            /* early termination */
#define SAD_THRESH(v) ( bcost < ( v >> pixel_size_shift[i_pixel] ) )
            if( bcost == ucost2 && SAD_THRESH(2000) )
            {
                COST_MV_X4( 0,-2, -1,-1, 1,-1, -2,0 );
                COST_MV_X4( 2, 0, -1, 1, 1, 1,  0,2 );
                if( bcost == ucost1 && SAD_THRESH(500) )
                    break;
                if( bcost == ucost2 )
                {
                    int range = (i_me_range>>1) | 1;
                    CROSS( 3, range, range );
                    COST_MV_X4( -1,-2, 1,-2, -2,-1, 2,-1 );
                    COST_MV_X4( -2, 1, 2, 1, -1, 2, 1, 2 );
                    if( bcost == ucost2 )
                        break;
                    cross_start = range + 2;
                }
            }
```

(a) Early termination

```
    /* Try extra predictors if provided.  If subme >= 3, check subpel predictors,
     * otherwise round them to fullpel. */
    if( h->mb.i_subpel_refine >= 3 )
    {
        /* Calculate and check the MVP first */
        int bpred_mx = x264_clip3( m->mvp[0], SPEL(mv_x_min), SPEL(mv_x_max) );
        int bpred_my = x264_clip3( m->mvp[1], SPEL(mv_y_min), SPEL(mv_y_max) );
        pmv = pack16to32_mask( bpred_mx, bpred_my );
        pmx = FPEL( bpred_mx );
        pmy = FPEL( bpred_my );

        COST_MV_HPEL( bpred_mx, bpred_my, bpred_cost );
        int pmv_cost = bpred_cost;

        if( i_mvc > 0 )
        {
            /* Clip MV candidates and eliminate those equal to zero and pmv. */
            int valid_mvcs = x264_predictor_clip( mvc_temp+2, mvc, i_mvc, h->mb.mv_limit_fpel, pmv );
            if( valid_mvcs > 0 )
            {
                int i = 1, cost;
                /* We stuff pmv here to branchlessly pick between pmv and the various
                 * MV candidates. [0] gets skipped in order to maintain alignment for
                 * x264_predictor_clip. */
                M32( mvc_temp[1] ) = pmv;
                bpred_cost <<= 4;
                do
```

(b) Extra Initialization

Figure 1: Early termination and Extra initialization

		DIA_x264	HEX_x264	UMH_x264	UMH_no_emv	ESA_x264
	Y	37.585	37.613	37.613	37.492	37.627
	U	42.284	42.304	42.315	42.219	42.310
PSNR	V	44.287	44.275	44.271	44.185	44.258
	Avg	38.774	38.799	38.800	38.683	38.811
	Global	38.656	38.680	38.681	38.568	38.692
Time		4.254	4.525	5.572	5.287	9.194
		DIA_no_emv	HEX_no_emv	UMH_no_ETRA	UMH_no_both	ESA_no_emv
	Y	37.506	37.497	37.624	37.496	37.506
	U	42.238	42.225	42.315	42.199	42.227
PSNR	V	44.189	44.191	44.267	44.186	44.258
	Avg	38.697	38.688	38.810	38.685	38.697
	Global	38.579	38.573	38.692	38.569	38.578
Time		4.049	4.392	6.163	5.921	8.639

Table 1: PSNR on foreman_cif.yuv using different settings.

As the introduction of each motion estimation search methods says, ESA would give best performance while using much longer time ($Table1$). Using extra motion vector would help a little bit on PSNR. Early termination in UMH would saving some time for encoding at the cost of some drop in PSNR.

3 Rate control in x264

3.1

3.1.1 Video buffer verifier compliant constant bitrate (CBR)

As its name shows, the output bitrate for the video should be a constant. It aims to be used on limited capacity channels for streaming multimedia content and can make full use of the channel capacity. It would not be best for storing data, since its quality would not be nice.

3.1.2 Average Bitrate (ABR)

In the mode, it would achieve an average bit rate close to a target, however, since encoder cannot predict what would happen in next frame, bit rate trends to vary greatly, especially in the starting point of the video and the end of video. It would likely have either a too large or too small bit rate to achieve target bit rate. So that quality would vary a lot even in a short clip.

3.1.3 Constant rate-factor mode (CRF)

This mode is default rate control method for x264. It would have a constant quality while various bit rate which can solve the shortage of CQP. Constant rate factor would define how much information to be ignored from a micro block. So that for a relative simple scene, it would have a lower bit rate, and for complex scene it would have higher bit rate. However, both scene would discard same amount of information resulting a more even quality and lower file size.

3.1.4 Two pass (2pass)

Using two pass estimation would allow encoder to get estimation for the targeted video. It can calculate the cost of encoding a frame in first pass and using this estimation in second path to more efficiently use the bits. It would ensure best quality under a certain bit rate.

3.1.5 Constant quantizer mode (CQP)

QP controls amount of compression in every frame, larger value indicates higher quantization, more compression, and lower quality. While constant means that the QP for compress the whole video is fixed. However using it bitrate would vary strongly depending on how complex each fame is and is not efficient for video. Since for a frame with relative simple scene, a relative low bit rate would be enough to achieve nice visual experience, while a complex one need more. So that if set QP as a constant for a video includes both, we need to either tolerate high bit rate for some unnecessary part or lose quality for complex frames.

3.2

```
# time command is used to calculate running time.
# CBR
time ./ffmpeg -f image2 \
    -s 640x480 -start_number 1 -i ED-360-png/%05d.png -vframes 15691 \
    -c:v libx264 -x264-params "nal-hrd=cbr:force-cfr=1" \
    -b:v 800k -minrate 800k -maxrate 800k -bufsize 2M out.mp4 -psnr

# ABR
time ./ffmpeg -f image2 \
    -s 640x480 -start_number 1 -i ED-360-png/%05d.png -vframes 15691 \
    -c:v libx264 -b:v 800k out.mp4 -psnr

# CRF
time ./ffmpeg -f image2 \
    -s 640x480 -start_number 1 -i ED-360-png/%05d.png -vframes 15691 \
```

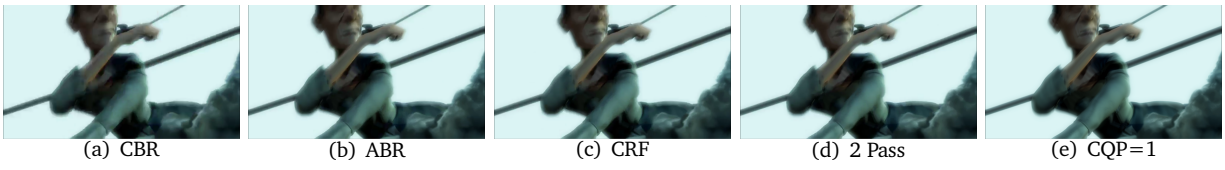
```
    -c:v libx264 -crf 21 out.mp4 -psnr

# 2 pass
time ./ffmpeg -f image2 \
    -s 640x480 -start_number 1 -i ED-360-png/%05d.png -vframes 15691 \
    -c:v libx264 -b:v 800k -pass 1 -f null /dev/null
time ./ffmpeg -f image2 \
    -s 640x480 -start_number 1 -i ED-360-png/%05d.png -vframes 15691 \
    -c:v libx264 -b:v 800k -pass 2 out.mp4 -psnr

# CQP
time ./ffmpeg -f image2 \
    -s 640x480 -start_number 1 -i ED-360-png/%05d.png -vframes 15691 \
    -c:v libx264 -qp 25 out.mp4 -psnr
```

		CBR	ABR	CRF	2pass
PSNR	Y	45.689	44.450	43.751	43.909
	U	55.870	55.568	55.363	55.345
	V	55.594	55.254	55.011	54.994
	Avg	48.078	46.920	46.293	46.441
	Global	42.602	43.587	44.386	44.494
Time		10m25.652s	10m5.949s	10m3.967s	4m1.041s+9m38.407s
BitRate		799.68	783.37	776.63	796.75

Table 2: PSNR of different rate control mode on elephant_dream sequence.

QP		1	10	20	25	30	40
PSNR	Y	62.740	53.159	46.166	42.846	39.487	33.394
	U	62.894	59.399	56.137	54.613	53.381	51.493
	V	62.947	59.326	55.840	54.245	52.939	51.040
	Avg	60.298	54.123	48.290	45.390	42.545	37.209
	Global	57.714	52.182	46.643	43.768	40.925	35.572
Time		20m2.488s	13m56.521s	10m36.983s	9m24.759s	8m8.114s	6m17.801s
BitRate		12391.77	4171.27	1392.38	770.54	408.60	115.82

Table 3: PSNR of different QP values on elephant_dream sequence using CQP mode.



(a) CBR (b) ABR (c) CRF (d) 2 Pass (e) CQP=1

(f) CQP=10 (g) CQP=20 (h) CQP=25 (i) CQP=30 (j) CQP=40

Figure 2: Compression result. 726^{th} frame

From $Table2\&3$ and $Figure2$, it can be found that CBR would perform poorly under bit rate we want. Frames with complex scenes would loss more information resulting poor quality. Lots of details are lost and edges are blurred. For ABR it would slightly perform better than CBR which comes from slight adaptiveness

among neighbours frames. However, due to the 1 pass encoding process, information loss for each frame is not optimal which causes the low PSNR. CRF gives better performance than previous 2 methods, the default value for CRF is 23. To achieve $800k$ bitrate, it was set at 21. PSNR is much better, but visual difference is not so obvious from $Figure2$. Two pass method would have better performance with a better adjusted bitrate for each frame.

Considering CQP, $QP = 1$ gives the highest performance while having a high bitrate and cost more time to encode it. With the increase of QP, video quality decreases. For $QP = 40$ this frame would have lots of blur and blocking effect. For $QP = 25$ which achieve bitrate at $800k$, PSNR is not comparable to CRF or 2 pass which would provide a more smooth transition between 2 different colors.