EE669 2015 Spring

PROJECT 4
Video Coding using H.264

Name: Shanglin Yang **ID:** 3795329308

Email: shangliy@usc.edu

Table of Contents

Problem 1: Written questions

Compare H.264 with MPEG2, and describe the new techniques adopted in H.264 that improve the rate-distortion performance.

1.1. The comparison between H.264 and MPEG2

The comparison involves application, performance and methods used. Detail seen in the table 1 below.

Table 1 Comparison between H.264 and MPEG2

	MPEG2	**H.264**
The main aiming application	It is widely used for the trans-mission of standard definition (SD) and high definition (HD) TV signals over satellite, cable, and terrestrial emission and the storage of high-quality SD video signals onto DVDs	High quality for all bitrate;better compresision performance on the higher quality multimedia;3G mobile networks, digital cinema, real time data transmission
The performance	MPEG-2 • 4:2:0MP@HL, 4:2:2MP@HL, 8-80Mbps • Good HD picture quality from as little as 12Mbps (MPEG-2) • Contribution quality at ~18Mbps • Low Latency ~42mS (decoder dependent)	Advantage: Save up to 50% in bit rate compare to H.263+ or MEPG-4 simple profile *Offer consistently high video quality at all bit rates * Operate in both low-delay mode (e.g. Real-time videoconferencing) and long processing delay mode (e.g. Video storage) * Provide necessary tools for error resilience *Separate Video Coding Layer (VCL) and Network Adaptation *Layer (NAL) for network friendliness
		Disadvantage:H.264 codec has very high demand on memory and computation power: Encoding: more than 4 times of MPEG-2 encoding Decoding: around 4 times relative to MPEG-2 decoding
Methods and Technology	Fixed macroblocks	Intra block prediction
	Block DCT transform	4x4 exact integer DCT transform
	motion estimation Up tp ½-pel	Advanced motion estimationHigher precision (1/4-pel)
	Fixed Block motion	Various block sizes and shapes
	reference frame I, P, B coding types	Multiple reference frames New predictive frames: SP frame
	No deblocking	In-loop de-blocking filter
	Fixed Scalar quantization	various Scalar quantization(51)
	entropy coding: Huffman coding	New entropy coding tools Universal VLC (UVLC) Content-based adaptive binary arithmetic coding (CABAC)
	NO NAL unit syntax structure	NAL unit syntax structur

1.2. The new techniques adopted in H.264

(1) Different models for Intra Prediction and Coding

Rather than using just the DC prediction, H.264 provides different models to do the intra prediction using Directional spatial prediction to reduce the spatial redundancy. So there are fewer bits to represent the coded macroblock as compared to applying transform directly.There are 9 modes for Intra 4x4 luma blocks,4 modes for Intra 16x16 luma blocks and 4 modes for Intra chroma blocks.This improves the quality of the prediction signal, and also allows prediction from neighboring areas that were not coded using intra coding.

(2) Inter Prediction and Coding

-Variable block-size motion compensation with small block sizes: H.264 supports more flexibility in the selection of motion compensation block sizes and shapes than any previous standard, with a minimum luma motion compensation block size as small as 4x4;

-Tree-structure Motion Segmentation: Partition of a macroblock based on tree-structure motion segmentation to get the Optimum choice of partitions;

-High-precision sub-pel motion vectors: H.264 use quarter-sample motion vector accuracy rather than integral and half-pel,thus it generates more accurate prediction for the target position may not be integral;

-Motion vectors over picture boundaries: The picture boundary extrapolation technique is included in H.264/AVC;

-Multiple reference picture motion compensation: H.264 extends upon the enhanced reference picture selection technique found in to enable efficient coding by allowing an encoder to select, for motion compensation purposes, among a larger number of pictures that have been decoded and stored in the decoder. Assign a picture reference parameter to each block;

-Decoupling of referencing order from display order:H.264/AVC allows the encoder to choose the ordering of pictures for referencing and display purposes with a high degree of flexibility constrained only by a total memory capacity bound imposed to ensure decoding ability;

-Weighted prediction: H.264/AVC allows the motion-compensated prediction signal to be weighted and offset by amounts specified by the encoder. This can dramatically improve coding efficiency for scenes containing fades, and can be used flexibly for other purposes as well;

-In-the-loop Deblocking filtering: Block-based video coding produces artifacts known as blocking artifacts. The deblocking filter in the H.264/AVC design is brought within the motion-compensated prediction loop, so that this improvement inquality can be used in inter picture prediction to improve the ability to predict other pictures as well. In addition to improved prediction methods, other parts of the design were also enhanced for improved coding efficiency,including the following.

(3) Block Transform

-Small block-size transform: All major prior video coding standards used a transform block size of 8 8,while the new H.264/AVC design is based primarily on a 4x4 transform. This allows the encoder to represent signals in a more locally-adaptive fashion, which reduces artifacts known colloquially as "ringing";

-Hierarchical block transform: There are some signals that contain sufficient correlation to call for some method of using a representation with longer basis functions. The H.264/AVC standard enables this in two ways: 1) by using a hierarchical transform to extend the effective block size use for low-frequency chroma information to an 8 8 array and 2) by allowing the encoder to select a special coding type for intra coding, enabling extension of the length of the luma transform for low-frequency information to a 16x16 block size in a manner very similar to that applied to the chroma;

(4) Entropy Coding

-Context-Adaptively switched Variable Length Codes (CAVLC)

CAVLC is the baseline entropy coding method of H.264. It assigns shorter codewords to symbols with higher probabilities of occurrences. The symbols and associated codewords are organized in look-up tables stored at both the encoder and decoder. A total number of 32 different VLCs are used in CAVLC.CAVLC achieves 2-7% bit rate reduction compared to conventional run length scheme;

-Context-based Adaptive Binary Arithmetic Coding (CABAC)

CABAC uses adaptive probability models and exploits symbol correlations by using contexts. It is adopted in H.264 main profile and CABAC provides 5-15% bit rate reduction compared to CAVLC.

(5) Error Resilience

Error resilience tools usually increase the data rate at the same quality.

-**Slice structure coding**

Flexible Macroblock Ordering (FMO): A new ability to partition the picture into regions called slice groups has been developed, with each slice becoming an independently-decodable subset of a slice group. When used effectively, flexible macroblock ordering can significantly enhance robustness to data losses by managing the spatial relationship between the regions that are coded in each slice

Arbitrary Slice Order (ASO): Since each slice of a coded picture can be (approximately) decoded independently of the other slices of the picture, the H.264/AVC
design enables sending and receiving the slices of the picture in any order relative to each other;

-Data partitioning (up to 3 partitions)

Since some coded information for representation of each region (e.g., motion vectors and other prediction information) is more important or more valuable than other information for purposes of representing the video contentH.264/AVC allows the syntax of each slice to be separated into up to three different partitions for transmission, depending on a categorization of syntax elements;

-Redundant intra coded slice

The slices are randomly selected channel adaptive R-D optimization;

-Multiple reference frames with feedback delay

Problem 2: Motion estimation in x264

2.1 Abstract and Motivation

Recently established H.264/AVC is the newest video coding standard. The main goals of the H.264/AVC standardization effort have been to enhance compression performance and provide a“network-friendly” video representation. The key methods adopted by the standardization is the more accurate motion estimation. H.264 has various motion-compensation units in sizes of 16×16,16×8, 8×16, 8×8, and sub8×8. For sub8×8, there are further four sub-partitions of sub8×8, sub8×4, sub4×8, and sub4×4. Meanwhile, H.264 also implements the quarter-pel accurate motion vectors which require interpolation of pictures by a factor of four,which is done by a 2-tap bilinear filter and a 6-tap FIR filter.

The subsequent coding gain is significant. However, such wide block choices greatly improve coding efficiency but at the cost of largely increased motion estimation time. The computational complexity becomes even worse when larger search ranges, bi-directional and/or multiple reference frames are used. Such high computational complexity is often a bottle-neck for real-time conversational applications. It also causes inconvenience for researchers during code optimization or evaluation process.

Thus, there have been many fast motion estimation techniques proposed and adopted by the H.264.Two popular approaches can be chosen to reduce computation in block matching motion estimation. The approach reduces the number of candidate blocks in the search window (fast searching techniques). Well-known examples are 2-D logarithmic search (LOGS), three-step search (TSS), block-based gradient descent search (BBGDS), new three-step search (N3SS), diamond search (DS), hexagon-based search (HEXBS).There methods show good speed gain but have relatively larger rate-distortion (R-D) performance degradation.

2.2 Approach and Procedures

2.2.1 Four different integer-pixel motion estimation methods

A. Diamond (DIA)

(1) Search Pattern

The Pattern is shown in Figure 1.

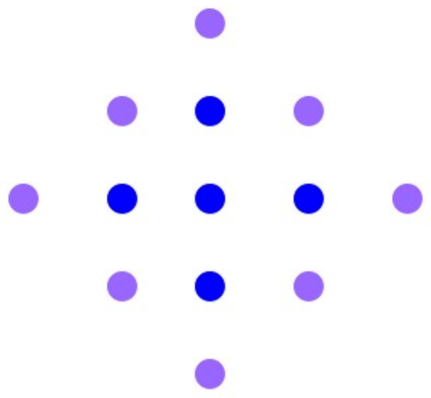


Figure 1 Diamond (DIA) Search Pattern

(2) Search Steps

Step 1) The initial LDSP (Large Diamond Search Pattern) is centered at the origin of the search window, and the 9 checking points of LDSP are tested (labeled '1'). If the MBD (minimum block distortion) point calculated is located at the center position, go to Step 3; otherwise, go to Step 2.

Step 2) The MBD point found in the previous search step is re-positioned as the center point to form a new LDSP (labeled '2') . If the new MBD point obtained is located at the center position(labeled '3'), go to Step 3; otherwise, recursively repeat this step (labeled '2').

Step 3) Switch the search pattern from LDSP to SDSP(labeled '4'). The MBD point found in this step is the final solution of the motion vector which points to the best matching block.

The process of the methods is shown in Figure 2.

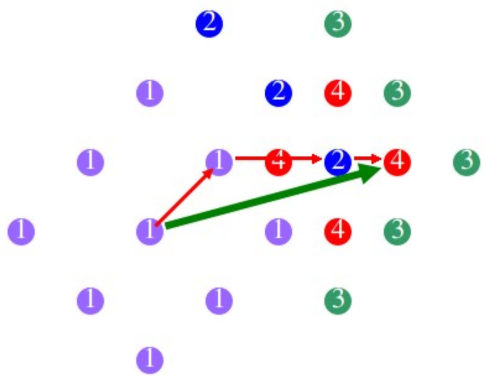


Figure 2 Diamond (DIA) Search Steps

B. Hexagon (HEX)

(1) Search Pattern

The Pattern is shown in Figure 3.

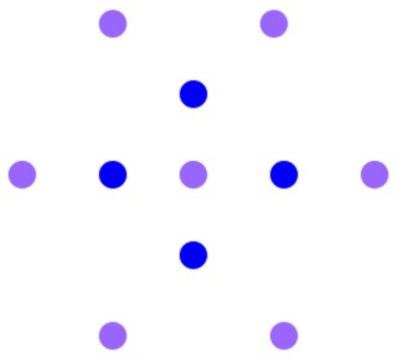


Figure 3 Hexagon (HEX) Search Pattern

(2) Search Steps

Step 1) The large hexagon with 7 check points is centered at (0,0) (labeled '1') , the center of predefined search window in the motion field. If the MBD point is found to be at the center of the hexagon, proceed to Step 3); otherwise, proceed to Step 2) .

Step 2) With the MBD point in the previous search step as the center, a new large hexagon is formed. Three new candidate points are checked, and the MBD point is again identified. If the MBD point is still the center point of the newly formed hexagon(labeled '3'), then go to Step3) ; otherwise, repeat this continuously(labeled '2').

Step 3) Switch the search pattern from the large size of hexagon to the small size of hexagon (labeled '4'). The four points covered by the small hexagon are evaluated to compare with the current MBD point. The new MBD point is the final solution of motion vector.

The process of the methods is shown in Figure 4.

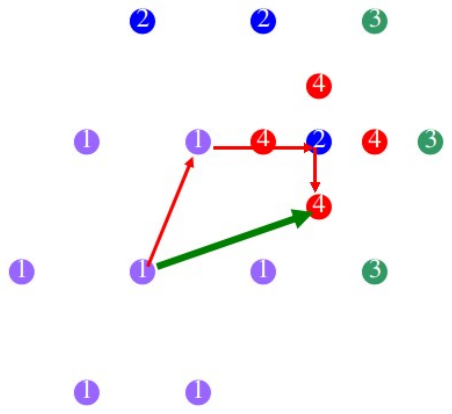
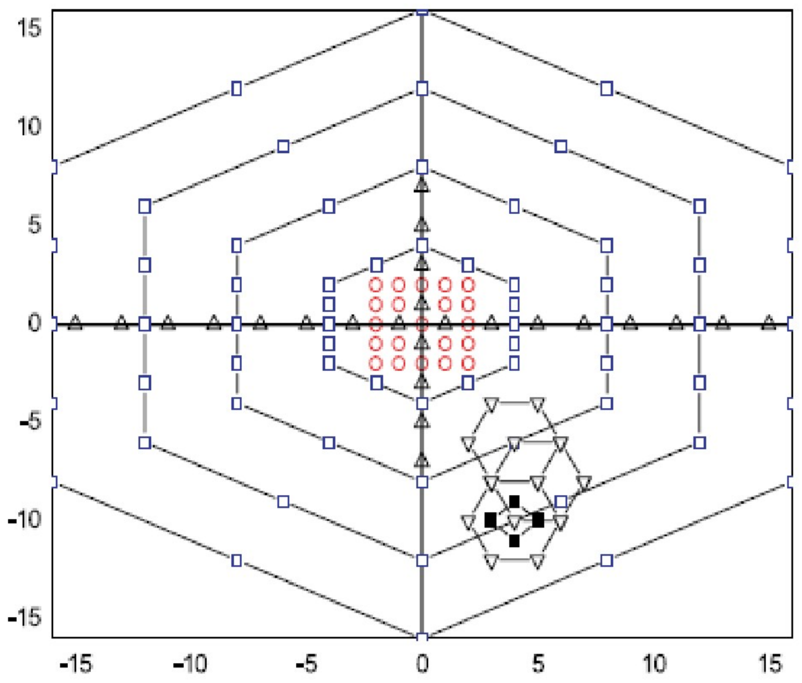


Figure 4 Hexagon (HEX) Search Steps

C. Uneven Multihexagon (UMH)

(1) Search Pattern

UMHexagonS is a hybrid pattern algorithm for it includes four steps with different kinds of search pattern: **unsymmetrical-cross search**; **small rectangular full search**; **uneven multi-hexagon-grid search**; **extended hexagon based search**; **diamond search**. Figure 5 demonstrates the different search pattern under the conditions of the assumption that the start search point to be (0, 0).



Δ-unsymmetrical-cross search ◯-small rectangular full search
□-uneven multi-hexagon-grid search ∇-extended hexagon based search
■- diamond search

Figure 5 Uneven Multihexagon (UMH) Search Pattern

(2) Search Steps

A. For the search steps, first do the integer-pel search, a full fractional pel search is performed for 8 types of blocks (Fig 6), and then implement a fast sub-pel search.

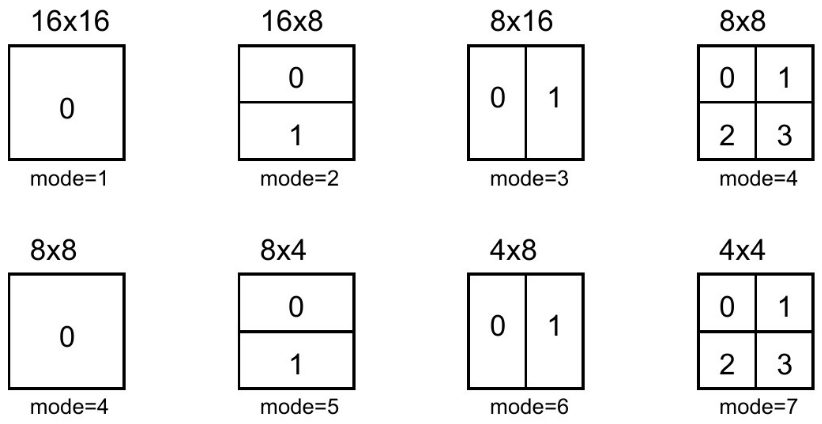


Figure 6 Different partition sizes in a macroblock

B. Considering the integer-pel search, the detail of the process is shown in the figure 7 :

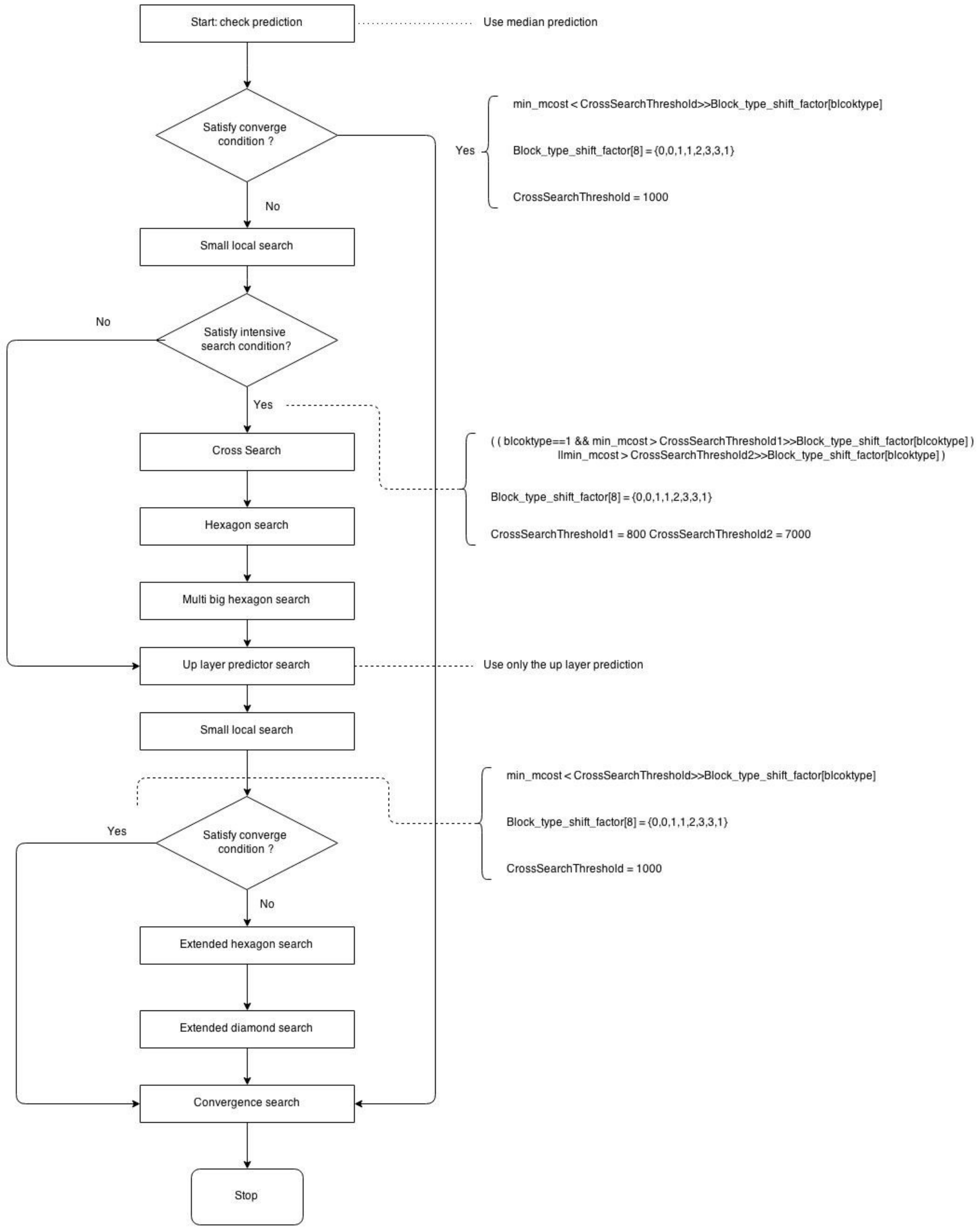

Start: check prediction
Use median prediction
Satisfy converge condition ?
Yes
min_mcost < CrossSearchThreshold>>Block_type_shift_factor[blcoktype]
Block_type_shift_factor[8] = {0,0,1,1,2,3,3,1}
CrossSearchThreshold = 1000
No
Small local search
No
Satisfy intensive search condition?
Yes
((blcoktype==1 && min_mcost > CrossSearchThreshold1>>Block_type_shift_factor[blcoktype]) llmin_mcost > CrossSearchThreshold2>>Block_type_shift_factor[blcoktype])
Block_type_shift_factor[8] = {0,0,1,1,2,3,3,1}
CrossSearchThreshold1 = 800 CrossSearchThreshold2 = 7000
Cross Search
Hexagon search
Multi big hexagon search
Up layer predictor search
Use only the up layer prediction
Small local search
min_mcost < CrossSearchThreshold>>Block_type_shift_factor[blcoktype]
Block_type_shift_factor[8] = {0,0,1,1,2,3,3,1}
CrossSearchThreshold = 1000
Yes
Satisfy converge condition ?
No
Extended hexagon search
Extended diamond search
Convergence search
Stop

Figure 7 Uneven Multihexagon (UMH) integer-pel search steps

Key points during the proces:

1): Initial search point prediction, For MV prediction, we only use spatial median and up layer predictors. median predictor is calculated anyway. For SAD prediction, we only use up layer predictor, namely, pred_SAD_uplayer as opposed to the four predictions used by UMHexagonS.

2): The convergence condition is based on the primary set and the block type as shown below.

C. Following the integer-pel search, a full fractional pel search is performed,the detail of the process is shown in the figure 8 :

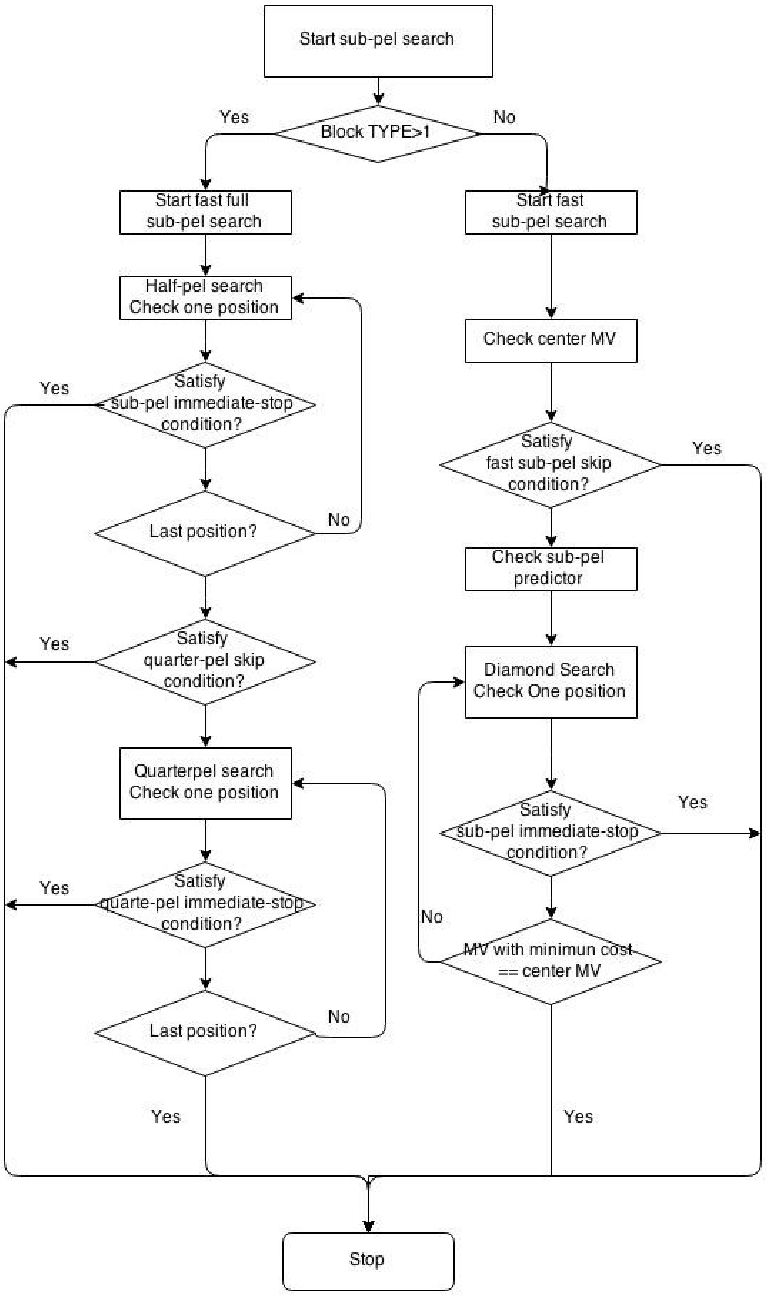

Start sub-pel search
Block TYPE>1
Yes
No
Start fast full sub-pel search
Start fast sub-pel search
Half-pel search Check one position
Satisfy sub-pel immediate-stop condition?
Yes
Last position?
No
Satisfy quarter-pel skip condition?
Yes
Quarterpel search Check one position
Satisfy quarte-pel immediate-stop condition?
Yes
Last position?
No
Yes
Check center MV
Satisfy fast sub-pel skip condition?
Yes
Check sub-pel predictor
Diamond Search Check One position
Satisfy sub-pel immediate-stop condition?
Yes
MV with minimun cost == center MV
No
Yes
Stop

Figure 8 Uneven Multihexagon (UMH) sub-pel search steps

Key points during the proces:

1) For block partitionof 16 × 16, a fast full sub-pel search is used because 16 × 16 block is utilized

for prediction for the smaller partition blocks. For all the other partition blocks, a fast sub-pel search is applied.

2) satisfying sub-pel immediate-stop condition which is defined as

(min_mcost < (SubPelThreshold2>>block_type_shift_factor[blocktype]))

the quarter-pel skip condition is defined as the following:

```
( (*mv_x == 0) && (*mv_y == 0) &&
    (pred_mv_x == 0 && pred_mv_y == 0) &&
        (min_mcost < (SubPelThreshold1>>block_type_shift_factor[blocktype])) )
```

the fast sub-pel skip condition is defined as

```
( ((*mv_x) == 0) && ((*mv_y) == 0) &&
    (pred_frac_mv_x == 0 && pred_frac_up_mv_x == 0) &&
        (pred_frac_mv_y == 0 && pred_frac_up_mv_y == 0) &&
            (min_mcost < (SubPelThreshold1>>block_type_shift_factor[blocktype])) )
```

D. Successive elimination exhaustive search (ESA)

(1) Basic idea

The main aim of these algorithms is to reduce the number of the search locations in the search windows. If the checked block is a better one, the matching error should be stored for the next matching decision. Otherwise, it will not be used. In fact, the former case appears occasionally compared with the later one. The sum norms of the blocks were used to decide whether matching computation should be performed at a search position in the search window and thus to reduce the number of matching evaluations.

(2) Approach and Improvement based on multi-level SEA

Baisc Approach:

Let f_c (i,j) and f_r (i,j) denote the intensity of the pixel with coordinate (i,j) in the current frame and the reference frame respectively.

Assume that the size of a block is N x N pixels, the motion vector search window size is (2M + 1) x (2M + 1), and the matching criteria function is mean absolute difference (MAD). Let $B_c^{(i,j)}$ and $B_r^{(i,j,x,y)}$ denote the two blocks in the two frames with the top left corners at (i, j) and (i - x; j - y) respectively, where (x,y) represent a candidate motion vector.

The MAD(x,y) between two blocks is defined as

$$MAD(x,y)=\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}\left|B_C(m,n)-B_r(m,n)\right| \tag{1}$$

Then , we could get the relations using mathematic:

$$\left|R-M(x,y)\right|\leqslant MAD(x,y) \tag{2}$$

where R and M(x,y) represent the sum norms of the two blocks defined as

$$R=\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}B_C(m,n) \tag{3}$$

$$M(x,y)=\sum_{m=0}^{N-1}\sum_{n=0}^{N-1}B_r(m,n) \tag{4}$$

if we define the (x*,y*) as the optimal motion vector,we replace it with current (x,y) only if M(x*,y*) > M(x,y). Based on the relation above, we can get the relation that is stratified (but not necessary) :

$$\left|R-M(x,y)\right|\leqslant MAD(x^{'*'},y^{'*'}) \tag{5}$$

If the relation is satisfied, we replace (x*,y*) with (x,y);otherwise, we skip it the reduce the cose.

Improvement based on multi-level :

To improve the performance ,we need more tighter decision boundary than equ.5. To achieve this aim,it partition a block into several sub-blocks and then use the sum norms of the sub-blocks to generate such a tighter decision value. Thus it partition the blocks in a multi-level manner to offer tighter and tighter decision boundaries. First, the block is partitioned into four subb-locks with size N=2 x N=2. Then each sub-block is partitioned into four sub-blocks with size N=4 x N=4. This procedure can be repeated until the size of the sub-blocks becomes 2 x 2. The maximum level of such partition is $L_{max} = \log_2 N - 1$ for the blocks with size N x N . The MSEA with L-level partition, $0 \leq L \leq L_{max}$, will be called L -level MSEA, and the SEA corresponds to the 0-level MSEA.

At l_{th} level of the L-level partition, where $0 \leq l \leq L$, the number of the sub-blocks is $S_l = 2^{2l}$, and each subblock is of size N_l x N_l, where $N_l = N/2^l$. We define

$$R_l^{(u,v)} = \sum_{m=0}^{N_l-1} \sum_{n=0}^{N_l-1} B_c(m+uN_l, n+vN_l) \tag{6}$$

$$M_l^{(u,v)}(x,y) = \sum_{m=0}^{N_l-1} \sum_{n=0}^{N_l-1} B_r(m+uN_l, n+vN_l) \tag{7}$$

$$MAD_l^{(u,v)}(x,y) = \sum_{m=0}^{N_l-1} \sum_{n=0}^{N_l-1} \left| B_c(m+uN_l, n+vN_l) - B_r(m+uN_l, n+vN_l) \right| \tag{8}$$

where (u,v) is the index of a sub-block. Then, with the respect of the whole block, we can get :

$$MAD(x,y) = \sum_{u=0}^{2^l-1} \sum_{v=0}^{2^l-1} MAD_L^{(u,v)}(x,y) \tag{9}$$

$$SMAD_l = \sum_{u=0}^{2^l-1} \sum_{v=0}^{2^l-1} \left| R_L^{(u,v)} - M_L^{(u,v)}(x,y) \right| \tag{10}$$

Then we get the relation as above:

$$SMAD_l \leq SMAD_{l+1} \tag{11}$$

$$SMAD_L \leq MAD(x,y) \tag{12}$$

In (11), $0 \leq l \leq L - 1$. If we arrange all the sum norms of the sub-blocks in a block at l_{th} level partition into a vector, $SMAD_l$ can be considered as the distance between the sum norm vectors of the reference block and the current block. The inequality (11) and (12) mean that $SMAD_l$ becomes larger as l increases, and all the $SMAD_l$ are less than or equal to the MAD between the two blocks. Assume that all the sum norms of the blocks and sub-blocks are known. Based on inequality (11) and (12), the L-level MSEA can be described as:

step1) At each search point, $SMAD_0$ is calculated firstly. If it is not less than MAD(x * ; y*), we can decide that the checked point is not a better matching candidate and no other operation is necessary at this point. Otherwise, $SMAD_1$ is calculated.

Step2) If $SMAD_1$ is not less than MAD(x* ; y*), we can decide that the checked point is not a better matching candidate. Otherwise, $SMAD_2$ is calculated. This procedure is repeated until $SMAD_L$ is calculated. If $SMAD_L$ is not less than MAD(x* ; y*), we can decide that the checked point is not a better matching candidate. Otherwise, the MAD between the two blocks MAD(x; y) is calculated.

Step3) Finally, if MAD(x; y) is less than MAD(x*; y*), then MAD(x* ; y*) and (x* ; y*) are replaced with MAD(x; y) and (x; y) respectively.

Since the gap between $SMAD_l$ and MAD(x; y) becomes smaller and smaller as l increases from 0 to L, more and more search points can be eliminated. In this way, the MSEA with L > 0 can save the matching computation cost more significantly than the SEA.

(3) Fast computation

The sum norms of all the blocks and subblocks can be efficiently calculated on the whole image before the block matching motion estimation. Since the subblocks of a block is related to the block by a quadtree structure, we can calculate the sum norms from the bottom to the top of the quad-tree. For a block in the current frame, the recursive computing formulas are the following:

$$R_L^{(u,v)} = \sum_{u'=0}^{N_L-1} \sum_{v'=0}^{N_L-1} B_c(m+u', n+v') \tag{13}$$

$$R_l^{(u,v)} = \sum_{u'=0}^{1} \sum_{v'=0}^{1} R_{l+1}^{(2u+u', 2v+v')} \qquad 0 \le l \le L-1 \tag{14}$$

Suppose that the size of an image frame is $W_f * H_f$. We rearrange the sum norms of all the possible sub-blocks with size N_l x N_l in the whole reference image into an image f_r^l (i, j) according to their relative locations of the sub-blocks, where $0 \le i \le H_f - N_l$; $0 \le j \le W_f - N_l$. For L-level MSEA, there are L+1 images to be generated. If f_r^l(i, j) is known, f_r^{l-1} (i, j) can be obtained by filtering fr l (i; j) using a separable two-dimensional filter. Which also save the large calculation while decrease the calculation complexity.

2.2.2 Motion search strategy implemented in x264

A. Modified initialization scheme

We consider up to 10 candidate motion vectors (MVs):

4 candidate motion vectors: the motion vector of neighboring locations in the current frame (A,B,C,D);

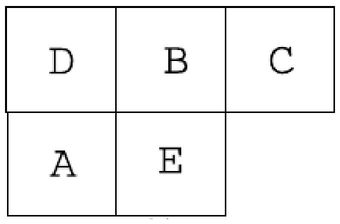


Figure 9 the neighboring locations in the current frame
E represents the current macroblock location

3 candidate motion vectors: three MVs in the previous frame (X,Y,Z),The MVs from the previous frame are scaled based upon the amount of time they span.;

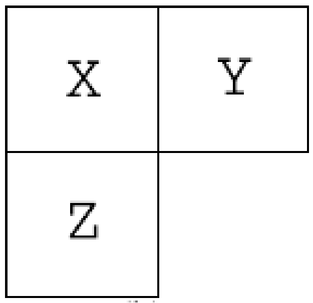


Figure 10 the locations in the previous frame
Z represents the current macroblock location

1 candidate motion vectors: the (0,0) MV;
1 candidate motion vectors: the median MV;
1 candidate motion vectors: the temporal direct MV.

We compare the sum of absolute differences (SAD) of each candidate, and choose the best SAD.

B. Early termination and range adaptive (ETRA) algorithm

a) The whole process flowchart

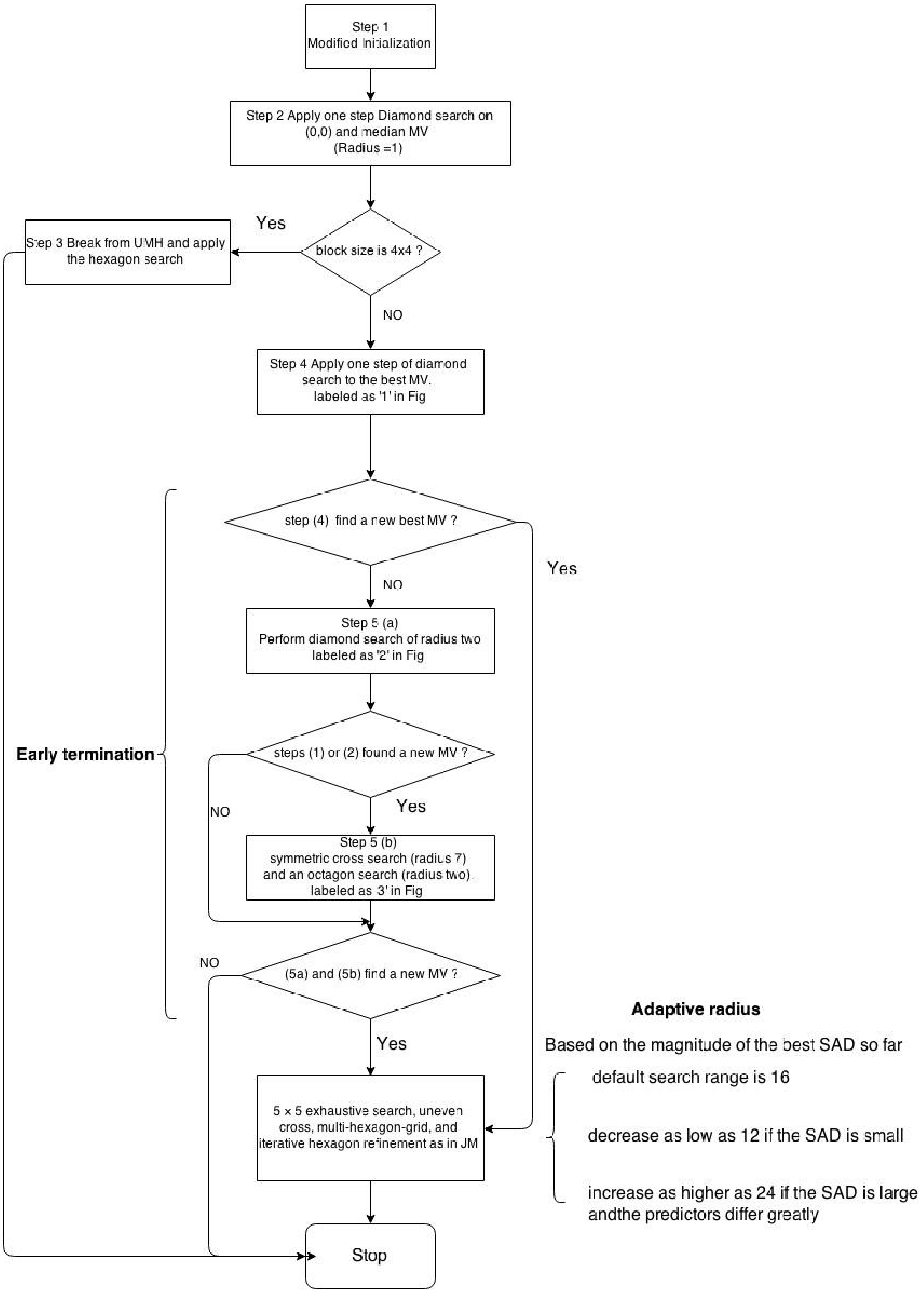

Step 1
Modified Initialization
Step 2 Apply one step Diamond search on (0,0) and median MV (Radius =1)
block size is 4x4 ?
Yes
Step 3 Break from UMH and apply the hexagon search
NO
Step 4 Apply one step of diamond search to the best MV. labeled as '1' in Fig
step (4) find a new best MV ?
Yes
NO
Step 5 (a) Perform diamond search of radius two labeled as '2' in Fig
steps (1) or (2) found a new MV ?
NO
Yes
Step 5 (b) symmetric cross search (radius 7) and an octagon search (radius two). labeled as '3' in Fig
Early termination
NO
(5a) and (5b) find a new MV ?
Yes
5 × 5 exhaustive search, uneven cross, multi-hexagon-grid, and iterative hexagon refinement as in JM
Stop
Adaptive radius
Based on the magnitude of the best SAD so far
default search range is 16
decrease as low as 12 if the SAD is small
increase as higher as 24 if the SAD is large andthe predictors differ greatly

Figure 11 the process flowchart adopted with ETRA

b) Early Termination

The Early termination is firstly based on the one-radius diamond search around the prediction (Step 4), if we do not get new prediction , meaning the prediction is accurate, we have the chance to do the early termination,then we implement step 5 . otherwise, we would not do the early termination. It is the first judgment.

After the first judgment,we need a tighter decision to make sure whether to do the early termination ,which is based on the step 5 (a) and (b). In step 5(a),with radius 2 of diamond we have more chance to find another better prediction; In step 5 (b) is based on the results of the small diamond search around the (0,0) and median which give more chance to have the better prediction. If both 5 (a) and (b) find no new MV, we say the prediction is used with high accuracy and break from the search.

The Early termination provides the chance of leaving the search process when we get prediction with high accuracy. It save the calculation time as well as keep the performance.

c) Adaptive Radius

The radius adopted by the search in the JM methods is based on the best SAD so far. The default value is 16 while it could varies from 12 to 24.

When best SAD is small ,meaning the difference between the current and reference is small ,which could be the frame change lightly and/or we get good prediction, thus ,we can have a small search window;

When best SAD is large or the prediction differ greatly ,meaning the difference between the current and reference is large ,which could be the frame change fast and/or we get poor prediction, thus ,we can have a big search window;

Thus Adaptive Radius realize the balance between the performance and the search time.

2.2.3 The performance of motion search methods in x264 with and without optimizations

A. Extra MV's initialization and the ETRA usage in the x264 source code

a. The extra mv's initialization in the file "**me.c**" under the folder"**x264/encoder/me.c**"and the file "**mvpred.c**" under the folder "**x264/common/mvpred.c**".

In the "**mvpred.c**", the methods are implemented in the function :

"void x264_mb_predict_mv_ref16x16(x264_t *h, int i_list, int i_ref, int16_t mvc[9][2], int *i_mvc)"

which calculate and set the extra MV candidate,include the the b-direct search candidate , spatial predictor and temporal predictor --three MVs in the previous frame (X,Y,Z),The MVs from the previous frame are scaled based upon the amount of time they span.

```
/* spatial predictors */
        if( SLICE_MBAFF
        {
           SET_IMVP( h->mb.i_mb_left_xy[0] );
           SET_IMVP( h->mb.i_mb_top_xy );
           SET_IMVP( h->mb.i_mb_topleft_xy );
           SET_IMVP( h->mb.i_mb_topright_xy );
        }
        else
```

```
        {
           SET_MVP( mvr[h->mb.i_mb_left_xy[0]] );
           SET_MVP( mvr[h->mb.i_mb_top_xy] );
           SET_MVP( mvr[h->mb.i_mb_topleft_xy] );
           SET_MVP( mvr[h->mb.i_mb_topright_xy] );
        }
#undef SET_IMVP
#undef SET_MVP
```

```
/* temporal predictors */
        if( h->fref[0][0]->i_ref[0] > 0 )
        {
            x264_frame_t *l0 = h->fref[0][0];
            int field = h->mb.i_mb_y&1;
            int curpoc = h->fdec->i_poc + h->fdec->i_delta_poc[field];
            int refpoc = h->fref[i_list][i_ref>>SLICE_MBAFF]->i_poc;
            refpoc += l0->i_delta_poc[field^(i_ref&1)];
```

```
#define SET_TMVP( dx, dy ) \
        { \
            int mb_index = h->mb.i_mb_xy + dx + dy*h->mb.i_mb_stride; \
            int scale = (curpoc - refpoc) * l0->inv_ref_poc[MB_INTERLACED&field]; \
            mvc[i][0] = (l0->mv16x16[mb_index][0]*scale + 128) >> 8; \
            mvc[i][1] = (l0->mv16x16[mb_index][1]*scale + 128) >> 8; \
            i++; \
        }

        SET_TMVP(0,0);
        if( h->mb.i_mb_x < h->mb.i_mb_width-1 )
            SET_TMVP(1,0);
        if( h->mb.i_mb_y < h->mb.i_mb_height-1 )
            SET_TMVP(0,1);
#undef SET_TMVP
```

The next step of the extra motion candidate is implemented in the "me.c"with the function : "***void x264_me_search_ref (x264_t *h, x264_me_t *m, int16_t (*mvc)[2], int i_mvc, int *p_halfpel_thresh)*** "

Besides the candidate provided below, there are added with other candidate including the median predictor and zero predictor. Which is set two part based on the h->mb.i_subpel_refine ,which kind of represents the accuracy required from the frame. The key part of it is the do-while part,whcih calculate the "cost"/SAD for each MV candidate and then choose the best candidate as below:

```
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
            {
                int mx = mvc_temp[i+1][0];
                int my = mvc_temp[i+1][1];
                COST_MV_HPEL( mx, my, cost );
                COPY1_IF_LT( bpred_cost, (cost << 4) + i );
            } while( ++i <= valid_mvcs );
```

b. ETRA usage is implemented in the in the "me.c"with the function :"***void x264_me_search_ref (x264_t *h, x264_me_t *m, int16_t (*mvc)[2], int i_mvc, int *p_halfpel_thresh)*** "

the code is seen below:

```
/* early termination */
#define SAD_THRESH(v) ( bcost < ( v >> x264_pixel_size_shift[i_pixel] ) )
        if (ETAR_enable==1)
        {
              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;
                    }
                }
        }

 if( i_mvc && ETAR_enable==1 )
        {
```

```
/* range multipliers based on casual inspection of some statistics of
 * average distance between current predictor and final mv found by ESA.
 * these have not been tuned much by actual encoding. */
static const uint8_t range_mul[4][4] =
{
    { 3, 3, 4, 4 },
    { 3, 4, 4, 4 },
    { 4, 4, 4, 5 },
    { 4, 4, 5, 6 },
};
int mvd;
int sad_ctx, mvd_ctx;
int denom = 1;

if( i_mvc == 1 )
{
    if( i_pixel == PIXEL_16x16 )
        /* mvc is probably the same as mvp, so the difference isn't meaningful.
         * but prediction usually isn't too bad, so just use medium range */
        mvd = 25;
    else
        mvd = abs( m->mvp[0] - mvc[0][0] )
            + abs( m->mvp[1] - mvc[0][1] );
}
else
```

```
    {
        /* calculate the degree of agreement between predictors. */
        /* in 16x16, mvc includes all the neighbors used to make mvp,
         * so don't count mvp separately. */
        denom = i_mvc - 1;
        mvd = 0;
        if( i_pixel != PIXEL_16x16 )
        {
            mvd = abs( m->mvp[0] - mvc[0][0] )
                + abs( m->mvp[1] - mvc[0][1] );
            denom++;
        }
        mvd += x264_predictor_difference( mvc, i_mvc );
    }
```

B. Usage code and command :

code using the ffmpeg:

```
ffmpeg -f rawvideo -vcodec rawvideo -s 352x288 -r 25 -threads 2 -pix_fmt yuv420p -i foreman_cif.yuv -c:v libx264 -me_method dia  -benchmark foreman_264_dia.mp4
```

code using the x264:

```
x264 --me umh --psnr --tune psnr -o foreman_264_dia.256 --input-res 352x288 foreman_cif.yuv
```

C. The change of code in original source

Using the flag **int emv_enable** for the no_emv first disable the extra candidate in the **mvpredict.c** then , disable the zero vector in the mv.c , considering the median is also based on the neighboring candidate and it is based on smaller luma rather than 16x16 macroblock, so it is kept.

The codes are seen below:

```
/* This just improves encoder performance, it's not part of the spec */
void x264_mb_predict_mv_ref16x16( x264_t *h, int i_list, int i_ref, int16_t mvc[9][2], int *i_mvc )
{
    int16_t (*mvr)[2] = h->mb.mvr[i_list][i_ref];
    int i = 0;


    if (emv_enable==1)
    {

            #define SET_MVP(mvp) \
            { \
                CP32( mvc[i], mvp ); \
                i++; \
            }

        #define SET_IMVP(xy) \
            if( xy >= 0 ) \
            { \
                int shift = 1 + MB_INTERLACED - h->mb.field[xy]; \
                int16_t *mvp = h->mb.mvr[i_list][i_ref<<1>>shift][xy]; \
                mvc[i][0] = mvp[0]; \
                mvc[i][1] = mvp[1]<<1>>shift; \
                i++; \
            }

                /* b_direct */
            if( h->sh.i_type == SLICE_TYPE_B
                && h->mb.cache.ref[i_list][x264_scan8[12]] == i_ref )
            {
                SET_MVP( h->mb.cache.mv[i_list][x264_scan8[12]] );
            }

            if( i_ref == 0 && h->frames.b_have_lowres )
            {
                int idx = i_list ? h->fref[1][0]->i_frame-h->fenc->i_frame-1
                                 : h->fenc->i_frame-h->fref[0][0]->i_frame-1;
                if( idx <= h->param.i_bframe )
                {
                    int16_t (*lowres_mv)[2] = h->fenc->lowres_mvs[i_list][idx];
                    if( lowres_mv[0][0] != 0x7fff )
                    {
                        M32( mvc[i] ) = (M32( lowres_mv[h->mb.i_mb_xy] )*2)&0xfffeffff;
                        i++;
                    }
                }
            }

    /* spatial predictors */
    if( SLICE_MBAFF )
    {
        SET_IMVP( h->mb.i_mb_left_xy[0] );
        SET_IMVP( h->mb.i_mb_top_xy );
        SET_IMVP( h->mb.i_mb_topleft_xy );
        SET_IMVP( h->mb.i_mb_topright_xy );
    }
    else
    {
        SET_MVP( mvr[h->mb.i_mb_left_xy[0]] );
        SET_MVP( mvr[h->mb.i_mb_top_xy] );
        SET_MVP( mvr[h->mb.i_mb_topleft_xy] );
        SET_MVP( mvr[h->mb.i_mb_topright_xy] );
    }
#undef SET_IMVP
#undef SET_MVP

    /* temporal predictors */
    if( h->fref[0][0]->i_ref[0] > 0 )
    {
        x264_frame_t *l0 = h->fref[0][0];
        int field = h->mb.i_mb_y&1;
        int curpoc = h->fdec->i_poc + h->fdec->i_delta_poc[field];
        int refpoc = h->fref[i_list][i_ref>>SLICE_MBAFF]->i_poc;
        refpoc += l0->i_delta_poc[field^(i_ref&1)];

#define SET_TMVP( dx, dy ) \
        { \
            int mb_index = h->mb.i_mb_xy + dx + dy*h->mb.i_mb_stride; \
            int scale = (curpoc - refpoc) * l0->inv_ref_poc[MB_INTERLACED&field]; \
            mvc[i][0] = (l0->mv16x16[mb_index][0]*scale + 128) >> 8; \
            mvc[i][1] = (l0->mv16x16[mb_index][1]*scale + 128) >> 8; \
            i++; \
        }
```

```
            SET_TMVP(0,0);
            if( h->mb.i_mb_x < h->mb.i_mb_width-1 )
                SET_TMVP(1,0);
            if( h->mb.i_mb_y < h->mb.i_mb_height-1 )
                SET_TMVP(0,1);
    #undef SET_TMVP
        }

}
else
{

        #define SET_MVP(mvp) \
            { \
                CP32( mvc[i], mvp ); \
            i++; \
            }

        #define SET_IMVP(xy) \
        if( xy >= 0 ) \
        { \
            int shift = 1 + MB_INTERLACED - h->mb.field[xy]; \
            int16_t *mvp = h->mb.mvr[i_list][i_ref<<1>>shift][xy]; \
            mvc[i][0] = mvp[0]; \
            mvc[i][1] = mvp[1]<<1>>shift; \
            i++; \
```

```
        }

            /* spatial predictors */
        if( SLICE_MBAFF )
        {
            SET_IMVP( h->mb.i_mb_left_xy[0] );
            SET_IMVP( h->mb.i_mb_top_xy );
            SET_IMVP( h->mb.i_mb_topleft_xy );
            SET_IMVP( h->mb.i_mb_topright_xy );
        }
        else
        {
            SET_MVP( mvr[h->mb.i_mb_left_xy[0]] );
            SET_MVP( mvr[h->mb.i_mb_top_xy] );
            SET_MVP( mvr[h->mb.i_mb_topleft_xy] );
            SET_MVP( mvr[h->mb.i_mb_topright_xy] );
        }
        #undef SET_IMVP
        #undef SET_MVP
    }



    *i_mvc = i;
}
```

For the using ETAR_enable to disabel the function. Codeseen below :

```
            if (ETAR_enable==1)
                {
                    ----
            }
          if( i_mvc && ETAR_enable==1 )
                {
             ----
            }
```

For the time recrd , using the function , Code seen below :

```
struct timeval  tv1, tv2;
  gettimeofday(&tv1, NULL);
/*------serach process---*/
gettimeofday(&tv2, NULL);
time_spent+= (double) (tv2.tv_usec - tv1.tv_usec) / 1000000 + (double) (tv2.tv_sec – tv1.tv_sec);
```

2.3 Results

Table 2 Performance Comparison of different motion search methods with and without optimizations

Method	PSNR (Mean)(dB)							$time_{ME}$(s)
	Y	U	V	Avg	Δ_{osnr}	Global	Δ_{osnr}	
DIA_x264	39.105	42.548	44.43	40.107	-0.021	39.986	-0.021	1.46
DIA_no_emv	39.07	42.45	44.348	40.063	-0.065	39.962	-0.045	1.43
HEX_x264	39.122	42.556	44.454	40.124	-0.004	40.003	-0.004	1.56
HEX_no_emv	39.082	42.473	44.375	40.077	-0.051	39.968	-0.039	1.52
UMH_x264	39.127	42.547	44.45	40.127	-0.001	40.006	-0.001	2.34
UMH_no_ETRA	39.135	42.573	44.463	40.137	0.009	40.016	0.009	3.09
UMH_no_emv	39.092	42.48	44.362	40.085	-0.043	39.978	-0.029	2.43
UMH_no_both	39.092	42.474	44.365	40.084	-0.044	39.978	-0.029	3.01
ESA_x264	39.127	42.553	44.451	40.128	0	40.007	0	4.69
ESA_no_emv	39.081	42.491	44.363	40.076	-0.052	39.968	-0.039	4.96

2.4 Discussion

2.4.1 The comparison between different search methods

From the table above,we can see for PSNR, the performance :ESA>UMH>HEW>DIA,But the performance time for search is shown that (better when using less time):DIA>HEX>UMH>ESA.

In detail, the we can find the DIA although spends least time on search ,it's performance (PSNR) is largely lower than other search methods; while , the ESA although get the best performance on PSNR ,but it take much more time, almost 3 times of time used in DIA and 2 times of the time of UMH;

The reason for that is because the the search time is related to the performance. It is rational when using more time on search,there is higher possibility that we could find better motion predictor which increase the PSNR. The DIA take one radius to search which is simple and fast but may not find the optimal predictor,while the HEX and UMH take more complexity search pattern and more steps to search ,thus increase the PSNR as well as the search time. For the ESA, it takes the full search with thresholds ,thus it has largest possibility to get the best predictor to increase the PSNR ,however,it takes more time;

Specially, we can find the PSNR difference between the UMH/HEX and ESA is very small(0.001),but the time taken by the ESA is two times of used by the UMH,thus ,it is not worthy to waste time using ESA while get little improvements on the PSNR;

2.4.2 The comparison between using or not the extra Mvs

PSNR: We can find in general, the using of extra Mvs improves the performance of PSNR, and the PSNR decrease when choose not to use the extra Mvs;

The reason for that is that the extra candidates provides us more probability to find better predictor ,for some extra mv candidates may have better representation of the true motion vector. Thus , we get better motion prediction and the PSNR when using the extra Mvs;

Specially , the influence of the extra Mvs is related with the motion search methods, for HEX,UMH and ESA,whose original PSNR is hicher,the imfluense is smaller comparing to the DIA, whose original PSNR lower.

The reason for that is the DIA using the simple and small search pattern which have less chance to find better motion predictor,thus it is highly depend on the original start points which is decided by the motion predictor. While for HEX,UMH and ESA ,which have more complex and reliable search methods,they can still get good motion predictor even if the start point is not good. So, when we choose not to use the extra Mvs, it is more possible that we get bad predictor , then for DIA ,it is more likely unable to get good motion vector ,while for HEX,UMH and ESA , they have more chance to still get good results;

Search time: Find for DIA and HEX, the search time decrease when we choose not to use extra Mvs candidates(no_emv); However, for UMH and ESA, the search time increase when we choose not to use extra Mvs candidates(no_emv);

The reason for that is because: for DIA and HEX, the search pattern and methods are simple comparing to UMH and ESA ,thus the initialization process takes large part of the whole search time. So when we do not use the extra candidates, we spend less time on initialization,although we get less accurate predictor which leads to more time on search, the time is saved more considering the initialization ; However, for UMH and ESA,the search pattern and methods are complex ,thus the search process takes large part of the whole search time. So when we do not use the extra candidates, we get less accurate predictor which leads to more time on search ,although we spend less time on initialization, the time is increased more considering more time to search,thus the search time increase when we choose not to use extra Mvs candidates(no_emv);

2.4.3 The comparison between using or not ETRA

PSNR: The PSNR , we get better performance and larger PSNR;

The reason for that is we choose not to use the early termination , thus we have more chance to find a better motion vector;

Search time: The search time increase almost 1.25 times when we use ETRA;

The reason for that is we choose early termination ,thus we get chance to stop search earlier, which leads the time saving;

Then, if we disable both,we get worsen PSNR and more time to search, the reason is same for two methods. In special ,without initialization , the improvement of PSNR from disabling ETRA is small , for less accurate predictor , there is less chance to get better performance;

2.4.4 Conclusion

From the discussion above, we can see that when we use more time on search , in general we get better performance on PSNR. Thus, for different search methods, the PSNR performance: ESA>UMH>HEW>DIA,while the performance time for search is shown that (better when using less time):DIA>HEX>UMH>ESA. In practical,we should make a balance between PSNR and search time. For example, we can find the PSNR difference between the UMH/HEX and ESA is very small(0.001),but the time taken by the ESA is two times of used by the UMH,thus ,it is not worthy to waste time using ESA while get little improvements on the PSNR;

Then, we can find Extra initialization can improve the PSNR performance ,while its effect on the search time depends on the method we choose;

The ETRA save the search time ,while it's PSNR effect is tolerable.

Thus , the strategy of applying EMV and ETRA generate better performance in practical use.

Problem 3: Rate control in x264

3.1 Abstract and Motivation

Rate control allows selection of encoding parameters to maxi-mize quality under bitrate and decoder video buffer constraints. The rate control in H.264 can be performed at three different granularities - group of pictures level, picture level, and macroblocks level. At each level, the rate control algorithm selects the quantization parameter (QP) values that determine the quantization of the transformed coefficients. As the QP increases, the quantization step size increases and the bitrate decreases. The rate control in x264 is based in part upon libav-codec's implementation, which is mostly empirical. It includes five different modes, one two-pass and four one-pass modes, that are described below. In the constant bitrate mode, each macroblock is allowed to have a different QP, while in other modes, the QP is determined for an entire frame.

3.2 Approach and Procedures

3.2.1 Five different modes in x264

A. Two pass (2pass)

In this approach, data obtained from each frame during a first pass are used for allocating bits globally over the file during the second pass. Using data from the first pass, we can achieve optimal results in the second coding, the flowchart is shown in the figure 12.

Step1: applying a one-pass mode in the first pass, the mode could be ABR or using constant QP;

Step2: The relative number of bits to be allocated between P-frames is selected independently of the total number of bits and it is calculated empirically by

$$\text{bits} \propto \quad complexity^{0.6} \tag{15}$$

where complexity is the predicted bit size of a given frame at constant QP.

Step3: Considering the bits allocation to decide the qscale and QP for the P-frames; The I- and B-frames use the QP from nearby P-frames with a constant offset;;

Step4: Scale the results considering the target file size and the requested file size,these process involves all frames together. Besides, if the VBV adopted, the buffer size should also be considered;

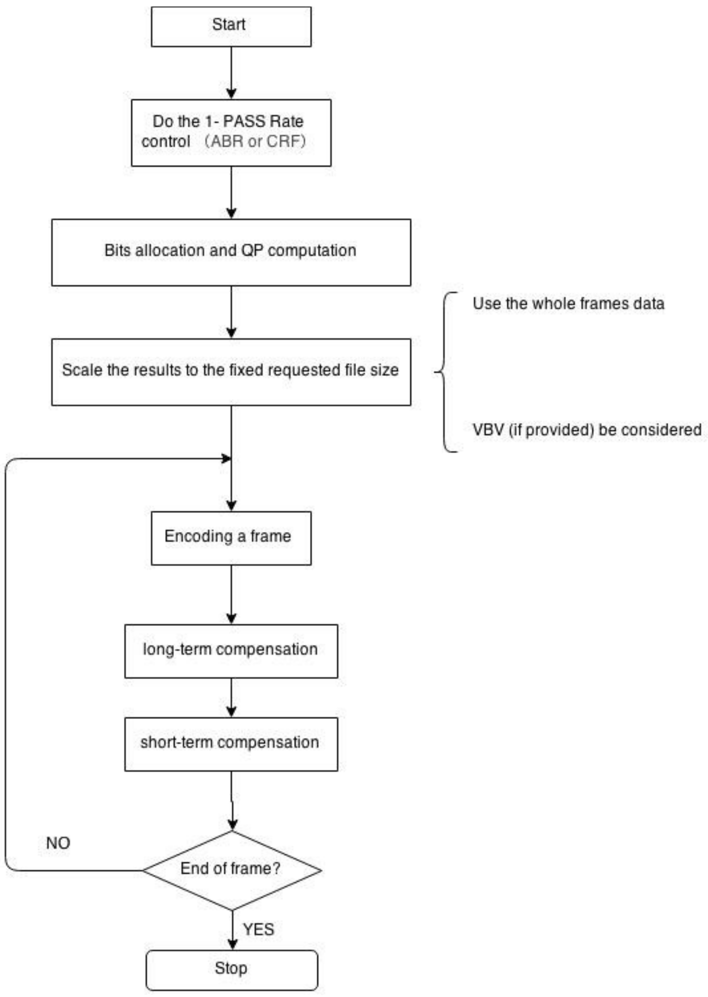

Start
Do the 1- PASS Rate control （ABR or CRF）
Bits allocation and QP computation
Scale the results to the fixed requested file size
Use the whole frames data
VBV (if provided) be considered
Encoding a frame
long-term compensation
short-term compensation
NO
End of frame?
YES
Stop

Figure 12 the process flowchart adopted with 2-pass

Step5: Encode the current frame;

Step6: The long-term compensation: If the second pass is consistently off from the predicted size, then the target size of all future frames is multiplied by a factor $\frac{prediced\,filesize}{real\,filesize}$ and their QPs are recomputed.

Step7: the short-term compensation to prevent x264 from deviating too far from the desired file size near the beginning (when there are less data for long-term compensation) and near the end (when long-term compensation does not have time to react). In short-term compensation, the compensation factor is defined as $C^{(real\,filesize\, -\, predicted\,filesize)}$, where C is a user defined constant.

B. Average Bitrate (ABR)

This is a one-pass scheme which produces near-constant quality within a specified file size. Since the rate control must be done without the knowledge of the future frames, ABR can not exactly achieve the target file size. the flowchart is shown in the figure 13.

Step1: Calculate the SATD for the current frame, SATD represents the "sum of absolute transformed Difference",which is the sum of the Hadmard transformed SAD; run a fast motion estimation algorithm over a half-resolution version of each frame, and use SATD as the complexity using the equation below:

$$blurred-comlexity[i]=\frac{cplxsum[i]}{cplxcount[i]} \qquad (16)$$

cplxsum and cplxcount represent the long-term computation of the complexity.

$$cplxsum[i+1]=cplxsum[i]*0.5+SATD[i] \qquad (17)$$

$$cplxcount[i+1]=cplxsum[i]*0.5+1 \qquad (18)$$

Step2: Calculate the qscale and scale it based on the past frame:

$$qscale[i]=blurred-comlexity[i]^{1-qcompress} \qquad (19)$$

$$qscale[i]=\frac{qscale[i]}{rate-factor} \qquad (20)$$

$$rate-factor=\frac{wanted-bits-window[i]}{cplxrsum[i]} \qquad (21)$$

where `wanted-bits-window[i] is the sum of all bits until current frame`
there is also overflow compensation:

$$qscale[i]=qscale[i]*overflow \qquad (22)$$

where overflow is decided by the total bits,target bits and the buffer size;

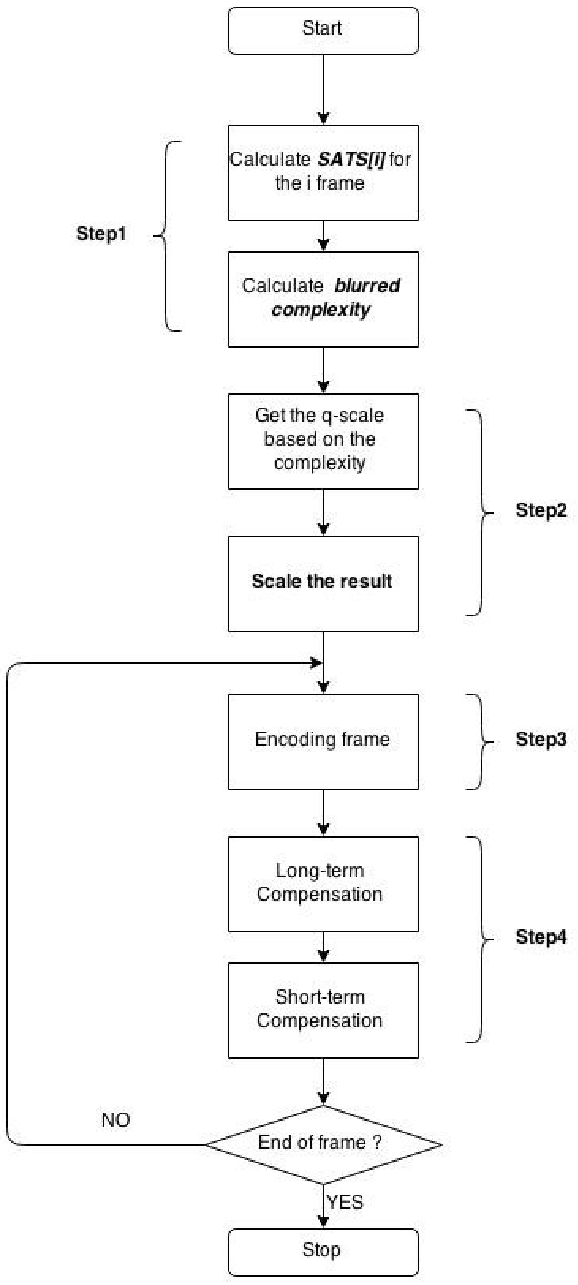


Figure 13 the process flowchart adopted with ABR

Step3: Encode the current frame;

Step4: Long-term compensation and short-term compensation same as 2-pass;

C. Video buffer verifier compliant constant bitrate (CBR)

This is a one-pass mode designed for real-time streaming. It uses the same complexity estimation for computing bit size as ABR does. But there is VBV buffer set to affect the results of the scale and overflow compensation, the flowchart is shown in the figure 14.

Step1: Calculate the SATD for the current frame, same as ABR;

Step2: Calculate the qscale and scale it based on the past frame. Basic ideas are same as ABR, but its scaling factor used for achieving the requested bi-trate is based on a local average (dependent on the buffer size of the video buffer verifier) instead of all past frame, and the overflow step is not included in this step;

$$blurred-comlexity[i]=\frac{cplxsum[i]}{cplxcount[i]} \quad (23)$$

where cplxsum and cplxcount are calculated based on the local values;

Step3: Encode each macroblock and do the overflow compensation. The overflow compensation is tighter, for it runs after each row of macroblocks instead of per-frame. The calculation is same as ABR;

Step4: Long-term compensation and short-term compensation same as 2-pass;

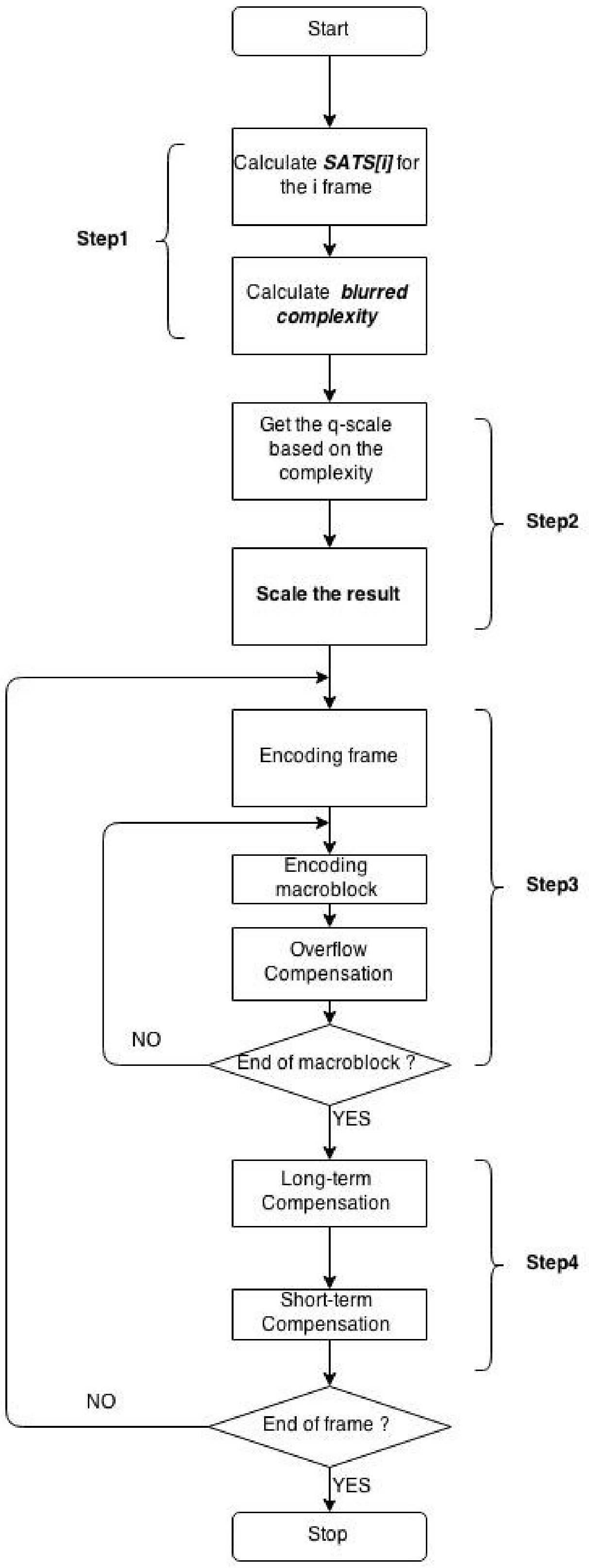


Figure 14 the process flowchart adopted with CBR

D. Constant rate-factor mode (CRF)

The constant rate-factor mode (CRF) is a one-pass mode that is optimal if the user specifies quality instead of bitrate. It is the same as ABR, except that the scaling factor is a user-defined constant and no overflow compensation is done. The flowchart is shown in the figure 15.

Step1: Set the CRF value;

Step2: Calculate the SATD for the current frame, same as ABR;

Step3: Calculate the qscale and scale it based on the past frame. Basic ideas are same as ABR, but for the equation (19) , the rate factor is set by user rather than computed by the program;

Step4: Step3: Encode the current frame without the overflow compensation;

Step5: Long-term compensation and short-term compensation same as 2-pass;

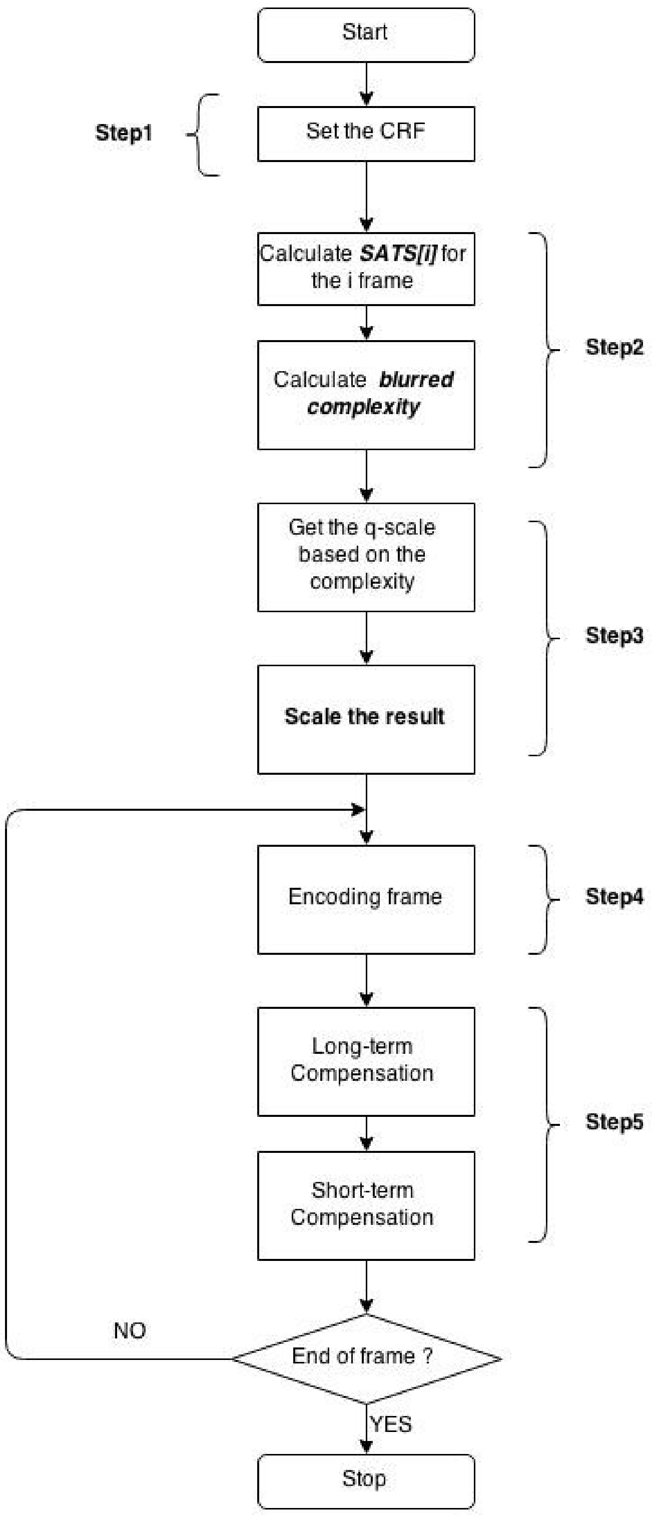


Figure 15 the process flowchart adopted with CBR

E. Constant quantizer mode (CQP)

The constant quantizer mode (CQP) is a one-pass mode where QPs are simply based on whether the frame is I-, P- or B-frame. This mode is used when the rate control option is disabled. The flowchart is shown in the figure 16.

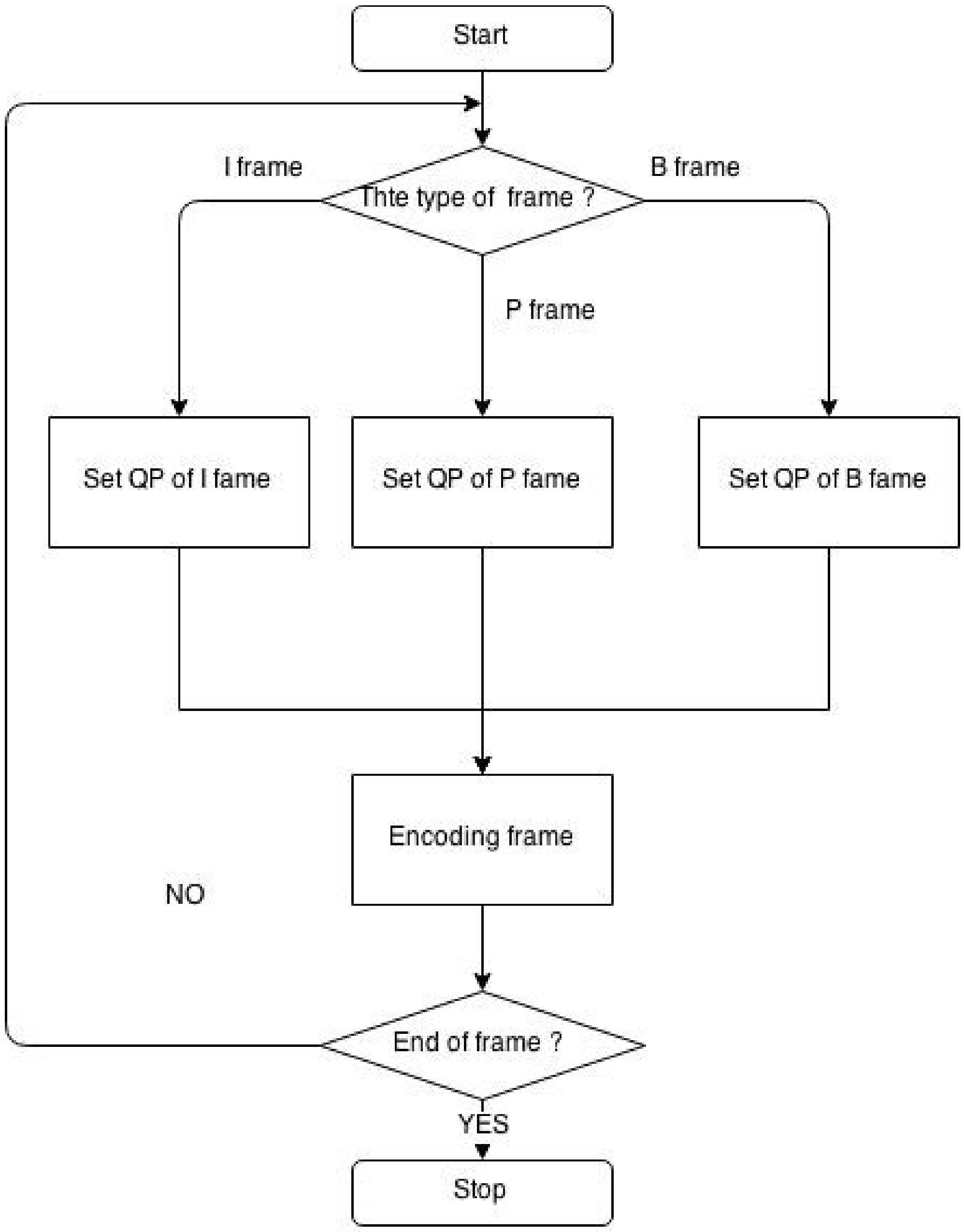


Figure 16 the process flowchart adopted with CQP

3.2.2 Command for rate control methods

Test each rate control schemes with the Elephants Dream sequence,use the command below:
Fisrt use the **png2yuv** - Convert PNG images to the YUV4MPEG stream format.The command is

```
png2yuv  -b 00001  -I  p  -j result.yuv
```

Then, use x264 command to do the encoding;

Video buffer verifier compliant constant bitrate (CBR):

```
x264 --demuxer y4m --bitrate 800 --vbv-maxrate 800 --vbv-bufsize 400 --fps 25 --psnr --tune psnr -o ele.mkv --input-res  640x480 result.yuv
```

Average Bitrate (ABR):

```
x264 --demuxer y4m --bitrate 800 --fps 25 --psnr --tune psnr -o ele.mkv --input-res  640x480 result.yuv
```

Constant rate-factor mode (CRF):

```
x264 --demuxer y4m --fps 25 --crf 23 --psnr --tune psnr -o ele.mkv --input-res  640x480 result.yuv
```

Constant quantizer mode (CQP):

```
x264 --demuxer y4m --qp 20 --fps 25 --psnr --tune psnr -o ele.mkv --input-res  640x480 result.yuv
```

Two pass (2pass):

```
x264 --demuxer y4m --pass 1 --crf 25 --fps 25 --psnr --tune psnr -o ele_pass1.mkv --input-res 640x480  result.yuv
x264 --demuxer y4m --pass 2 --bitrate 800 --fps 25 --psnr --tune psnr -o ele_pass2.mkv --input-res 640x480   result.yuv
```

3.3 Results

Table 3   Performance Comparison of different control methods for a target bitrate of 800 kb/s

Methods	Bitrate (kb/s)	PSNR(mean)/dB				
		Y	U	V	Avg	Global
CBR	**688.34**	45.957	56.352	56.224	46.969	**40.831**
ABR	**810.01**	45.112	56.066	55.914	45.142	**42.24**
CRF(23)	**761.09**	44.113	55.484	55.47	45.162	**43.351**
CQP(25)	**770.25**	43.167	54.834	54.503	44.185	**42.693**
2-Pass	**797.35**	44.301	55.456	55.202	45.346	**43.48**

Table 4 Performance Comparison of CQP with different Qps

Methods	Bitrate (kb/s)	PSNR(mean)/dB				
		Y	U	V	Avg	Global
CQP(10)	**4170.55**	53.33	60.181	60.282	53.923	**52.482**
CQP(20)	**1393.29**	46.481	57.005	56.772	47.443	**45.99**
CQP(25)	**770.25**	43.167	54.834	54.503	44.185	**42.693**
CQP(30)	**399.23**	39.8	53.34	52.954	40.973	**39.353**
CQP(40)	**104.8**	33.656	50.512	49.955	35.029	**33.332**

Table 5 Performance Comparison of CRF with different crfs

Methods	Bitrate (kb/s)	PSNR(mean)/dB				
		Y	U	V	Avg	Global
CRF(18)	**1388.5**	47.4	57.093	57.093	48.321	**46.572**
CRF(22)	**861.94**	44.77	55.701	55.438	45.782	**43.993**
CRF(23)	**761.09**	44.113	55.484	55.47	45.162	**43.351**
CRF(30)	**304.13**	39.52	53.12	52.55	40.715	**38.894**

3.4 Discussion

3.4.1 Performance Comparison of different control methods

From the table 3 ,we can see

--2-pass get the best performance both at the bitrates and the PSNR;

--ABR methods get good performance on the bitrate (810 kb/s is very close to the target 800 kb/s),but not very good at the PSNR performance compared to CQP and CRF;

--CQP and CRF do not get good output bitrates, but they generate good performance on PSNR;

--CBR get the bitrates far off the target as well as the bad PSNR results;

The reason for the performance above is that:

(1) 2-pass is based on the the first-pass results, thus it is a global optimization to choose the QP, thus it is most reliable to allocates the bits and generate QPs to get the best quality with the satisfying output bitrates;

(2)ABR focus on the output bitrates mainly. It tries best to make a good prediction on the bits and generates the QP as well as pertain the quality; Thus , it could generate good performance on the output bitrates,but it is still just the prediction based on the past datas , so it get as good as results as 2-pass, meanwhile, it's PSNR can not beat the CQP and CRF,whose focus are on the quality;

(3) CQP and CRF are both focusing on the quality by set parameter to decide the Qps. So, the bitrates can not be set directly. In the table , I set 23 for CRF and 25 for CQP , which based on the experiment; Because they care more on the quality,we can see the performance on PSNR is good , but meanwhile , the output bitrates are not satisfying;

(4)CBR set the max_bitrate and buffersize , as we know ,it have tighter requirement on the overflow , which leads the bitrate to be lower. Thus , the output bitrates of CBR tends to be far lower than the target. To get lower bitrates , the QP needs to the large , which course the poor quality and smaller PSNR;

3.4.2 Performance Comparison of different QPs and crfs

From the table 4 and table 5 ,we can see as Qps or crfs increase, the output bitrates decrease and the PSNR also decrease;

The reason for that is the larger Qps, the more information are removed ,which leads to smaller bits and lower bitrates , but meanwhile it leads the poor quality and lower PSNR;

In general , The QP and crf have basic effect, but the CRF are more adaptive to the data compared the CQP , for the CRF do the constant factor to the calculated qscale, which represents the complexity of the frame. In fact the CRF decrease more information the “less important frame”.Thus although CRF and CQP have similar average PSNR, psychovisually CRF performs better than CQP.

Besides, we can be the relation between the QP and bitrates is not linear;From the figure 17, we can see the relation is more like a Quadratic function or log function;

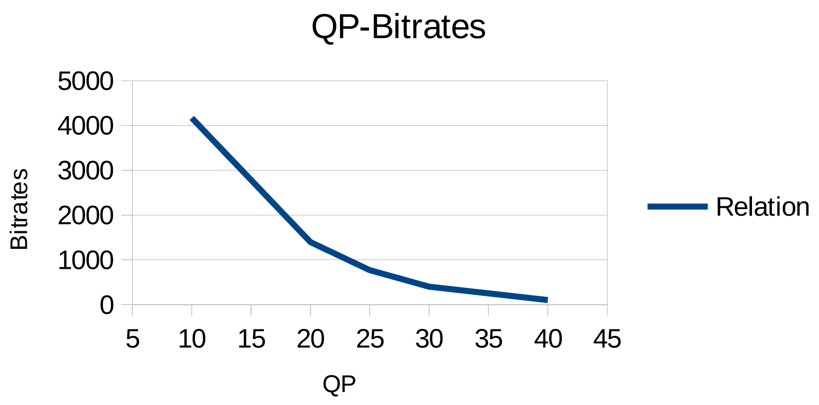


Figure 17   the relation between the QP and bitrates

3.4.3 Conclusions

From the talk above, we can see the 2-pass get the best performance, but it take much time., especially for real time media, we can choose only the 1-pass methos;

Among the different x264 one pass approaches, CRF provides the best quality. Meanwhile , the CQP although get similar results ,for both methods focus on the quality performance , but CQP loose more bits on the less important frame , thus , in practical , we are more willing to use the CRF. Besides, we can see the target bitrates are unable to be set directly through CRF and CQP ,and the relation between the QP and bitrates is not linear,so we can set the relation experimentally or use the bitrates limitation. In fact , we can see in general,we add and minus 6 on the QP or CRF, we get half or twice of the bitrates, so we can make assumption at first and then refine the results;

ABR focus on the bitrates , it is qualified when we have strong requirements on the output bitrates ,while CBR is the safe methods for overflow , but we should be careful about the underflow problem. We set the buffer size and refine the algorithm to get better performance.