Project 2 – H.264-based Motion Compensation Residual Coding

In this project, you will implement a simplified H.264/AVC inter-frame encoder with coding tools like motion estimation (ME), 4×4 integer discrete cosine transform (DCT), quantization and entropy coding, etc. You only need to encode the prediction residual information with entropy coding.

Objective

- To investigate the H.264/AVC 4×4 integer DCT algorithm
- To study the H.264/AVC quantization method
- To implement the H.264/AVC Exp-Golomb Entropy coding

1. Introduction

The H.264/AVC video coding standard was together finalized by two groups, the MPEG (Moving Pictures Experts Group) from ISO and the VCEG (Video Coding Experts Group) from ITU (International Telecommunication Union). Unlike the popular 8×8 discrete cosine transform used in previous standards, the 4×4 transforms in the H.264/AVC can be calculated exactly in integer arithmetic, thus avoiding inverse transform mismatch problems. The new transforms can also be computed without multiplications, just additions and shifts, in 16-bit arithmetic, thus minimizing computational complexity, especially for low-end processors. Fig. 1 shows the framework used in the H.264/AVC.

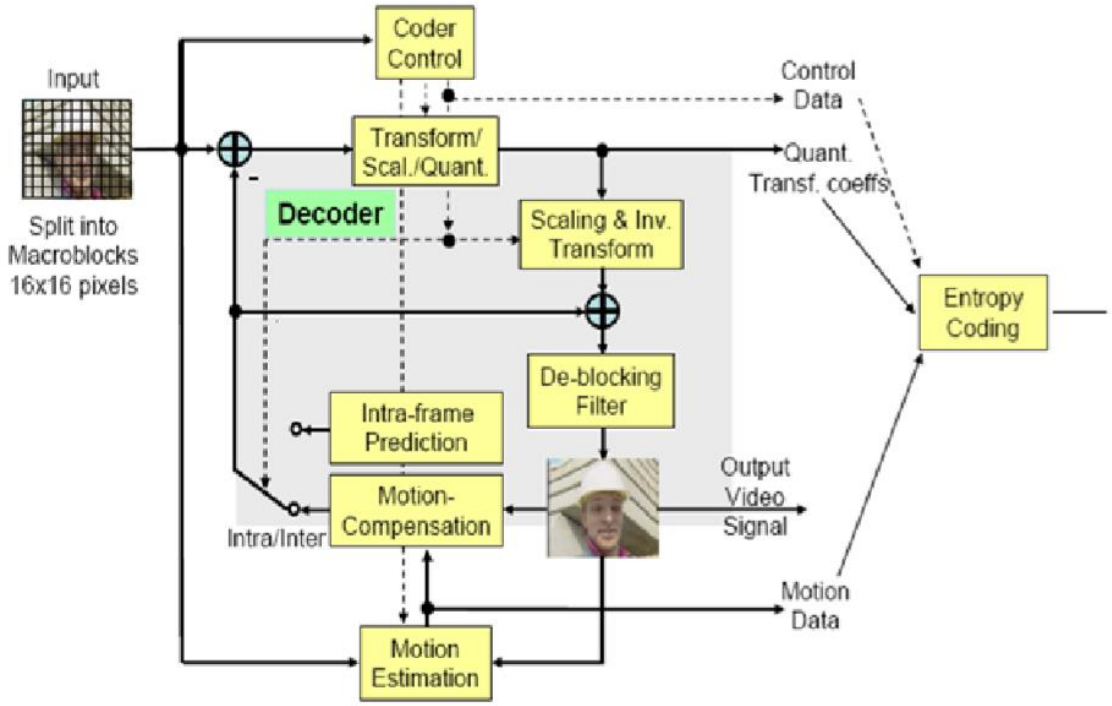

Figure 1: H.264/AVC Encoder.

In this project, you will implement a simplified H.264/AVC inter-frame encoder with coding tools like motion estimation (ME), 4×4 integer discrete cosine transform (DCT), quantization and entropy coding, etc.

2. Inter frame coding

In video coding, P frame refers to the inter-coded frame that references the previous reconstructed frames for motion estimation. In this section, you are asked to encode 10 original “Foreman (352×288)” Y component frames via inter frame coding. The framework to be implemented is shown in Fig. 2. Modules DCT, inverse DCT, Quantization, Dequantization, Zigzag scanning and Exp-Golomb Entropy coding will be introduced as follows.

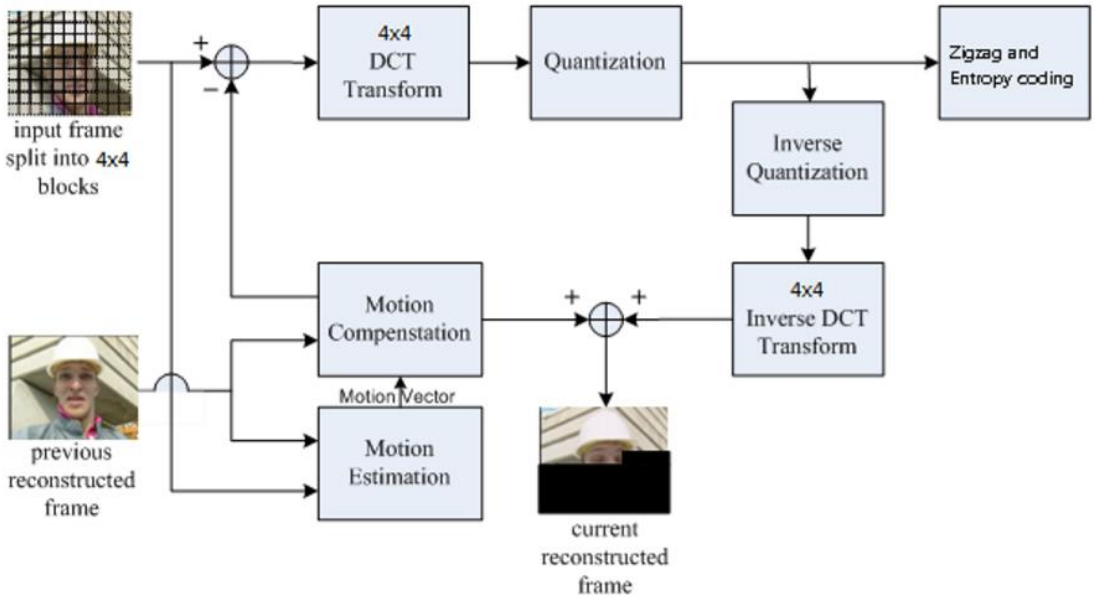

Figure 2: The framework of inter frame coding.

Fig. 3 shows the frame reference structure used in our project. For the motion estimation of frame 1, the first frame directly encoded will be used as the reference. For the frame 2, the reconstructed frame 1 will be used as the reference, so on and so forth. Full-search block matching algorithm (FSBMA) (Integer-pixel) is used in our project.

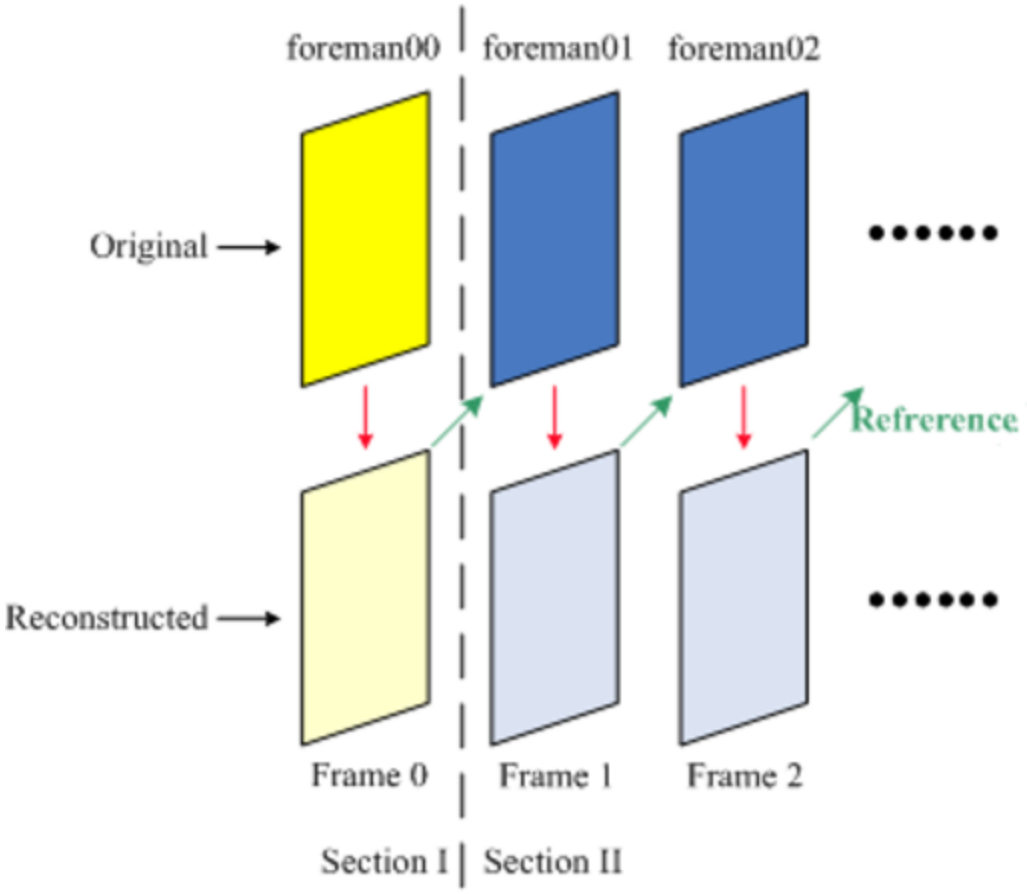

Figure 3: The framework of inter frame coding.

Given the motion vector, the motion compensation module locates and extracts the 4×4 matching block from the reference frame. The matching block will be subtracted from the current 4×4 block and the resultant residual block will be processed by the DCT module. Also the matching block will be added to the reproduced residual blocks to reconstruct the current frame, as shown in Fig. 2.

2.1 Forward/Inverse integer DCT

The integer DCT operates on a 4×4 block of residual data after motion-compensated prediction. The core part of the transform can be implemented using only additions and shifts. To reduce the total number of multiplications, the scaling-multiplication is integrated into the quantizer. The inverse quantization (scaling) and inverse transform operations can be carried out using 16-bit integer arithmetic with only a single multiply per coefficient, without any loss of accuracy.

The conventional DCT basis is given by

$$A_{ij} = C_i \cos\frac{(2j+1)i\pi}{2N} C_i = \sqrt{\frac{1}{N}}(i=0), C_i = \sqrt{\frac{2}{N}}(i>0) \quad (1)$$

When the input block is X, then X is transformed as

$$Y = AXA^T \quad (2)$$

where

$$A = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos(\frac{\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8}) & -\sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8}) & -\sqrt{\frac{1}{2}}\cos(\frac{\pi}{8}) \\ \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ \sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8}) & -\sqrt{\frac{1}{2}}\cos(\frac{\pi}{8}) & \sqrt{\frac{1}{2}}\cos(\frac{\pi}{8}) & -\sqrt{\frac{1}{2}}\cos(\frac{3\pi}{8}) \end{bmatrix}$$

.

Let a=1/2, b=$\sqrt{\frac{1}{2}}\cos(\pi/8)$ and c=$\sqrt{\frac{1}{2}}\cos(3\pi/8)$. To simplify the implementation of the transform, and to ensure that transform remains orthogonal, the forward transform process Eq. (1) is modified

$$\begin{aligned} Y' &= (C_f X C_f^T) \otimes E_f \\ &= \left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} X \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix} \right) \otimes \begin{bmatrix} a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \\ a^2 & \frac{ab}{2} & a^2 & \frac{ab}{2} \\ \frac{ab}{2} & \frac{b^2}{4} & \frac{ab}{2} & \frac{b^2}{4} \end{bmatrix} \end{aligned} \quad (3)$$

where a=1/2 and b=$\sqrt{\frac{2}{5}}$. E_f is integrated into the quantizer. The inverse transform process is

$$\begin{aligned} Y &= C_i^T (Y \otimes E_i) C_i \\ &= \begin{bmatrix} 1 & 1 & 1 & \frac{1}{2} \\ 1 & \frac{1}{2} & -1 & -1 \\ 1 & -\frac{1}{2} & -1 & 1 \\ 1 & -1 & 1 & -\frac{1}{2} \end{bmatrix} \left(X \otimes \begin{bmatrix} a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \\ a^2 & ab & a^2 & ab \\ ab & b^2 & ab & b^2 \end{bmatrix} \right) \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \frac{1}{2} & -\frac{1}{2} & -1 \\ 1 & -1 & -1 & 1 \\ \frac{1}{2} & -1 & 1 & -\frac{1}{2} \end{bmatrix} \end{aligned} \quad (4)$$

H.264/AVC employs the fast DCT by only additions and shifts. Fig. 4 shows flowgraphs of the fast forward and inverse integer DCT, which are applied to rows and columns of each 4×4 block. x(i) is the input pixel and X(i) is output transform coefficient. Correspondingly, Xr(i) is the input coefficient and xr(i) is the reconstructed pixel.

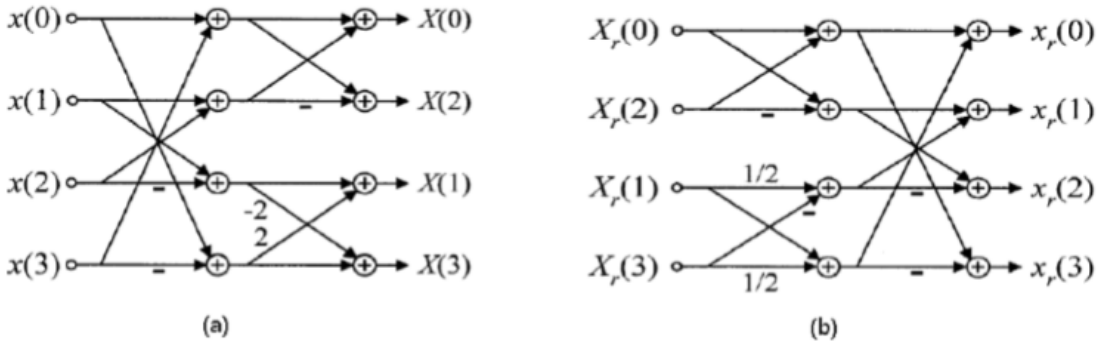

Figure 4: Fast implementation of the H.264/AVC forward transform (a) and inverse transform (b). No multiplications are needed, only additions and shifts.

2.2 Quantization/Dequantization

Assuming that $W = C_f X C_f^T$ is the transform coefficient block, the H.264/AVC employs the following quantization process.

$$Z = \text{round}(W \times \frac{PF}{\text{Qstep}}) \quad (5)$$

where *PF* is the post-scaling factor and *Qstep* is the quantization step, Y_{ij} is a transform coefficient described above, *Qstep* is a quantizer step size and Z_{ij} is a quantized coefficient.

QP	Q_{step}	*QP*	Q_{step}	*QP*	Q_{step}	*QP*	Q_{step}
0	0.625	7	1.375	18	5	⋮	
1	0.6875	8	1.625	⋮		42	80
2	0.8125	9	1.75	24	10	⋮	
3	0.875	10	2	⋮		48	160
4	1	11	2.25	30	20	⋮	
5	1.125	12	2.5	⋮		51	224
6	1.25	⋮		36	40		

Figure 5: Quantization step sizes in the H.264/AVC.

A total of 52 values of Qstep are supported by the standard and these are indexed by a quantization parameter(QP). The values of Qstep corresponding to each QP are shown in Fig. 5. Note that Qstep doubles in size for every increment of six in QP; Qstep increases by 12.5% for each increment of one in QP. The wide range of quantizer step sizes makes it possible for an encoder to accurately and flexibly control the trade-off between bit rate and quality.

To replace the division by the shift operator, H.264/AVC adopts the following conversion

$$\frac{PF}{\text{Qstep}} = \frac{\text{MF}}{2^{\text{qbits}}} \tag{6}$$

where MF is the multiplication factor and qbits = 15 + floor(QP/6). The first 6 values of MF is shown in Fig. 6, depending on QP and the coefficient position (*i,j*).

QP	Positions (0,0),(2,0),(2,2),(0,2)	Positions (1,1),(1,3),(3,1),(3,3)	Other positions
0	13107	5243	8066
1	11916	4660	7490
2	10082	4194	6554
3	9362	3647	5825
4	8192	3355	5243
5	7282	2893	4559

Figure 6: Multiplication factor MF in the H.264/AVC.

In integer arithmetic, Eq. (5) can be implemented as follows:

$$|Z_{ij}| = (|W_{ij}| \times \text{MF} + f) >> \text{qbits} \tag{7}$$

where >> indicates a binary shift right, f is the offset and

$$f = \begin{cases} \frac{2^{qbits}}{3} & \text{intra block} \\ \frac{2^{qbits}}{6} & \text{inter block} \end{cases}.$$

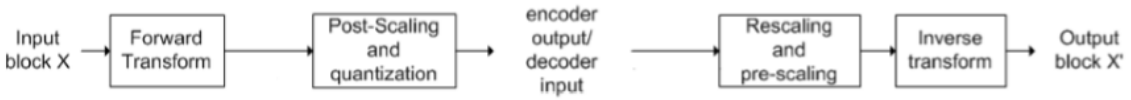

Figure 7: Transform, quantization, dequantization and inverse transform workflow.

Fig. 7 shows the transform coding of the H.264/AVC. The complete forward transform and quantization are

Encoding

1. Input : 4×4 resisdual sample :X

2. Forward transform : $W = C_f X C^T_f$

3. Post−Scaling and quantization : Eq.(7)

Note that you can choose this decoding methods as follows.

Decoding

1. Decodingscaling: $W' = Z \times Q_{step} \times PF \times 64$

2. inverse Transform: $X' = C_i^T(W')C_i$

3. Post−scaling : X'',

Output : 4×4 residual samples : X''

2.3 Zigzag scanning and Exp-Golomb Entropy Coding

The zigzag scanning pattern for run-length coding of the quantized DCT coefficients was established in the original MPEG standard. Fig. 8 shows the scanning pattern.

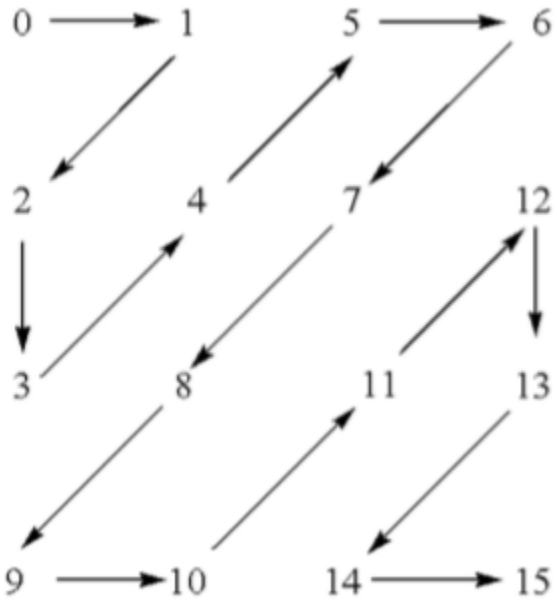

Figure 8: Zigzag table.

Exp-Golomb codes are adopted by current video standards to perform entropy coding. Traditionally, Exp-Golomb (EG) code is used in the coding of generalized Gaussian sources and both are claimed to be nearly optimal.

Exp-Golomb codes are constructed in a logical way:

$$[\text{Mzeroes}][1][\text{INFO}] \tag{8}$$

where where INFO is an M-bit field carrying information.

The value of M is a function of the index code num:

$$M = floor(\log_2 [code\_num + 1]) \tag{9}$$

where INFO = $code\_num+1-2^M$. The length of each Exp-Golomb codeword is $(2M+1)$ bits.

The Exp-Golomb codeword can be decoded as follows:

1. Read in M leading zeroes followed by 1.
2. Read M−bit INFO field.
3. Find $code\_num = 2^M + \text{INFO} - 1$.

[hints]: if the coefficient is minus, you should add a sign bit. Please look for related documents on website.

3. Highlights on Implementation Issues

(a) To implement the fast integer DCT algorithm in a Matlab function;

(b) For intra frame coding (frame 0), set the QP=22 and directly encode it by the DCT without motion prediction. For inter frame coding (frame 1 to frame 9), set the QP=24; [Notice]: Your program should also works at QP ∈ [1,51]

(c) The search range for motion estimation algorithm (FSBMA) should be set to [-8, +8], [-16, +16] and [-32, +32], respectively; [Notice]: You can also provide more search range results in your report.

(d) To find a matching block, a criterion to measure similarity of blocks is required. You should use Sum of Square Difference (SSD);

(e) To show PSNR curves of the decoded frames under different motion estimation conditions, i.e., different search ranges;

(f) To show PSNR curves of the decoded frames under different quantization parameters (QP=18, 22, 24, 26, 30, and 36) for inter frames with search range [-8, 8];

(g) To calculate the binary code length of each frame;

(h) To show the computational complexity of your encoder, i.e., the running time of encoding each frame;

(i) To evaluate your source code, explicitly show the intra coding results of the first 4x4 block (frame 0), i.e, quantized/dequantized transform coefficients at QP=17-22;

(j) A report with source codes must be submitted before the ??. The report should include the following parts but not limited to: Objective, Introduction, Methodology, Simulation Results, Discussion, Conclusion, References (optional), etc. Useful Functions (use lower-case instead of capital letters for function names, e.g. 'ZEROS' should be 'zeros').

TIC and TOC functions work together to measure elapsed time.

B = PADARRAY(A,PADSIZE,PADVAL) pads array A with PADVAL (a scalar) instead of with zeros.

ZEROS(M,N) or ZEROS([M,N]) is an M-by-N matrix of zeros.

IMSHOW(I, ⋯) displays the grayscale image I.

XLABEL('text') adds text beside the X-axis on the current axis.

YLABEL('text') adds text beside the Y-axis on the current axis.

TITLE('text') adds text at the top of the current axis.

More "help" can be found from the MATLAB online document:
http://www.mathworks.com/help/matlab/

$$MSE = \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I(i,j) - R(i,j)]^2$$

$$SSE = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [I(i,j) - R(i,j)]^2$$

$$SAD = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |I(i,j) - R(i,j)|$$

$$PSNR = 10 \log_{10} \left(\frac{Max_I^2}{MSE} \right)$$

where $I(i,j)$ is the original data and $R(i,j)$ is the estimated one, *M* and *N* are the width and height respectively, and Max_I is the maximum value in I.