UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI

Multimedia Representation and Compression

Project Hybrid Video Coder

Supervisor : Anissa MOKRAOUI

[1] *Université Sorbonne Paris Cité*

[2] *University of Science and Technology of Hanoi*

Student : VU Anh Tuan

Master ICT, USTH

Hanoi, June 7, 2015

Contents

List of Figures

Reduction of Spatial Redundancy: Still Image

1

Q.1. *Complete the implementation of the basic JPEG version (that we started together) as provided in the document (unit2.3 part b slides 16 and 17) without using arithmetic encoder. Work on still image "Lena" and "Peppers".*

To implement a basic version of JPEG compression algorithm without using arithmetic encoder, for encoding I implement three first steps of JPEG encoding scheme (figure 1.1).

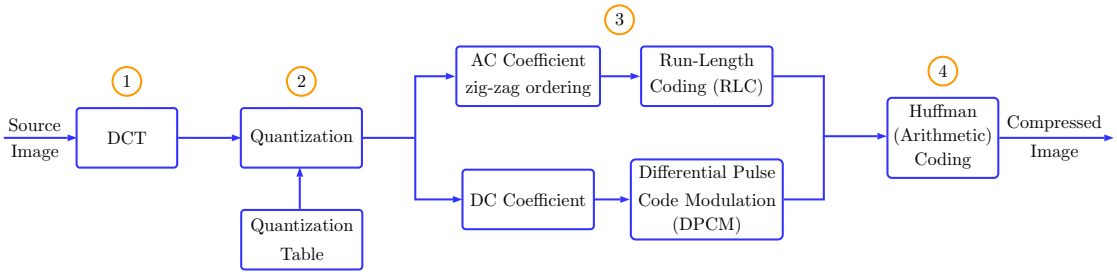


Figure 1.1: JPEG Encoding Scheme

The first and second steps are quite simple and are done in the class then in this section I explain only in detail the third step (figure 1.1). In this step, DC coefficients are coded by using difference encoding, called Differential Pulse Code Modulation (DPCM). An example of DC coefficients encoding is shown in the table 1.1.

DC coeffs	22	→	21	→	33	→	39	→	41	→	42	→	41	→	30	...
Difference		-1		12		6		2		1		-1		-11		...
Encoded	22	→	-1	→	12	→	6	→	2	→	1	→	-1	→	-11	...

Table 1.1: Example of DC coefficients encoding

Moreover in this step, AC coefficients of each block are read following zig-zag way to create a 1-D sequence. Then sequences of block are combined into a large 1-D sequence of all AC coefficients. In the Run-Length Coding (RLC) part, a sequence of block is encoded in a collection of 2-tuples `(skip, value)`, where `skip` indicates the number of zeros in the run and `value` is the next non-zero coefficient. In the large sequence of all AC coefficients it emerges the need to have a mark in order to separate sequences of block. In my code I use tuple `(0,0)` to mark the end of a block because in the run of encoding this tuple never appears. For example, a sequence of block: `9, 7, 0, 3, 2, 0, -2, -3, 0, 0, 0, 1, 0 ... 0` *(51 zeros at the end)* is encoded to tuples `(0,9),(0,7),(1,3),(0,2),(1,-2),(0,-3),(3,1),(0,0)`.

For decoding, I implement three last steps of JPEG decoding scheme (figure 1.2) because we do not realize arithmetic encoding. The second step of this process is the inverse of the third step in the encoding process. In this step, from the encoded DC, AC coefficients I reconstruct DC, AC coefficients then combine them to quantized DCT matrix. Then I do inverse quantization (step 3 in figure 1.2) on this matrix, after that do inverse block DCT (step 4 in figure 1.2) to obtain decoded image.

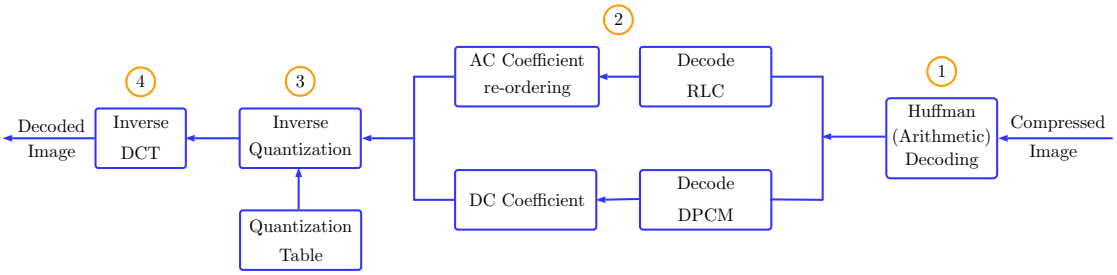


Figure 1.2: JPEG Decoding Scheme

As the demand of question, I use two images in the figure 1.3 to test my basic JPEG version.

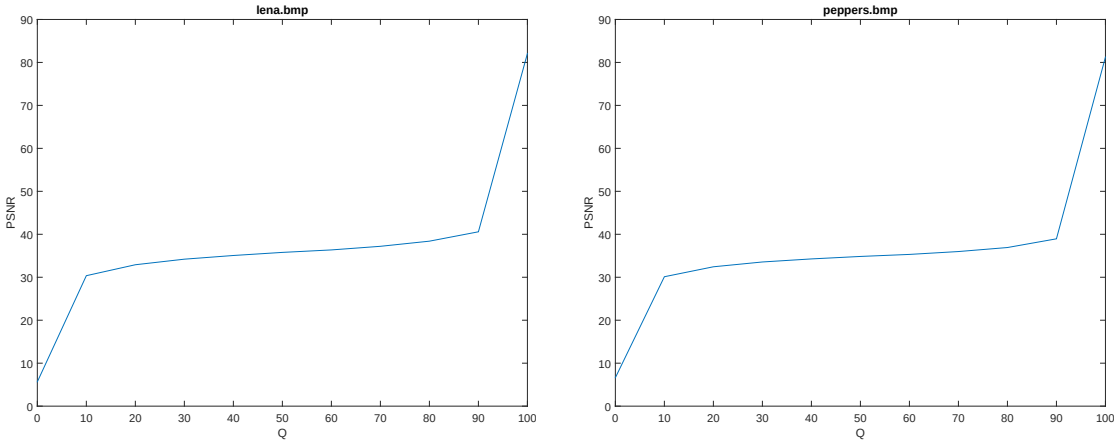
(a) Lena (b) Peppers

Figure 1.3: Images to work on

Q.2. *Discuss the behavior of the quality factor with respect to the PSNR (provide a graph).*

The influence of quality factor on PSNR of two images (figure 1.3) is shown in the figure 1.4. From this figure we can see that higher quality makes higher PSNR. This is easy

understand because PSNR measures the quality of reconstruction of lossy compression. PSNR is computed by the ratio between the square of maximum possible value and the mean square error. With higher quality of compression, error between reconstructed image and source image is small, thus PSNR gets high value. With lower quality of compression, this error is high, thus PSNR gets small value.



(a) Lena (b) Peppers

Figure 1.4: Influence of quality factor Q on PSNR

Moreover, in the figure 1.4 we can see that the PSNR increases gradually in case of quality factor increases from 10 to 90. The PSNR rises sharply in case of quality factor increase from 0 to 10 and from 90 to 100. Therefore, to keep quality of reconstructed image, the quality factor needs to be higher than 10 and it is much better if the quality factor takes value from 90 to 100. In this case, the reconstructed image is nearly the same the source image.

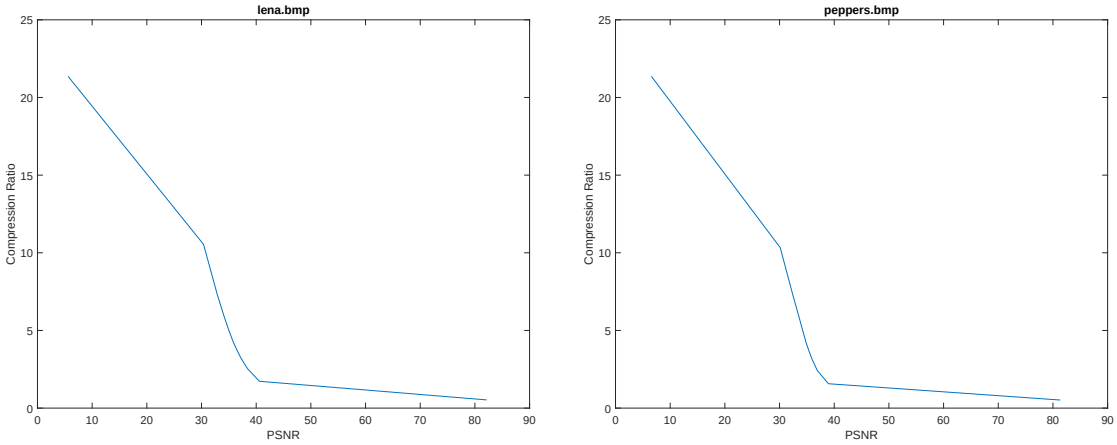
Q.3. *Discuss the PSNR behavior with respect to the compression ratio for a given quality factor (provide a graph).*

Since in this basic JPEG version we do not realize arithmetic coding then to compute the compression ratio I consider each element of encoded DC coefficients uses 8 bits, and each tuple `(skip, value)` of encoded AC coefficients uses 16 bits. Therefore, if we define *rows, cols* are number of rows and columns of image; *length(DC), length(AC)* are the number of elements in encoded DC, AC coefficients respectively, the compression ratio is then computed by the formula 1.1 below.

$$\text{Compression Ratio} = \frac{rows * cols}{length(DC) + length(AC)} \tag{1.1}$$

The influence of PSNR on compression ratio of two images `Lena` and `Peppers` in the figure 1.3 is shown in the figure 1.5. This figure shows that the compression ratio slumps

when PSNR varies from 5 to 40. The compression ratio decreases slowly when PSNR varies from 40 to 80. This means that the higher PSNR makes lower compression ratio. The compression ratio has significant change if the PSNR varies in [5; 40], and has slight change if PSNR is higher than 40. In my JPEG version, the maximum compression ratio obtained is about 20 times when the quality equal or less than 1. The minimum compression ratio is about 0.5 times. This is obtained when quality factor equal 100. It means that in this case there is no compression.



(a) Lena (b) Peppers

Figure 1.5: Influence of PSNR on compression ratio

Reduction of Temporal Redundancy: Video Sequence

2

Q.1. *Implement the generic block matching algorithm (studied in document part b) with equal block size as a parameter using two consecutive frames of any test video sequence. Explain which mode of prediction you have implemented.*

In my code I implement the Block Matching Algorithm (BMA) in forward prediction mode. In this mode, a frame is encoded based on its previous frame. In forward prediction mode, motion vectors and residual errors between predicted and original frame are sent to the decoder. The decoder bases on motion vectors and residual error frame to reconstruct the original frame.

There are three functions in my implementation: `getBestMatching, encodeFrame` and `decodeFrame`. I will explain these function below.

Function `getBestMatching`: This function is to find the best matching of a block of frame to encode in reference frame. This function takes 6 parameters.

- `B`: this is block to find the best matching.
- `refFr`: this is reference frame in which we find the best matching of `B`.
- `bcor`: this is vector of coordinates ([ycor xcor]) of top-left corner of block `B` in the frame to encode.
- `mspd`: this is vector of moving speed following horizontal and vertical direction to move next block to compare in the reference frame. By default, `mspd = [1 1]`.
- `sreg`: this is region of searching the best matching of block `B` in the reference frame. This is a vector of horizontal and vertical direction. By default, `sreg = [5 5]`.
- `eps`: this is up-bound MSE between the block `B` and its matched block. This means that if it found a block having MSE between this block and `B` is less or equal `eps`, it will finish the process of searching. I add this parameter to reduce the time of searching. By default, `eps = 0`.

Function `encodeFrame`: This function is to encode a frame to motion vectors and residual error frame based on the BMA in forward prediction mode. This function takes 3 mandatory parameters: frame to encode `Fr2`, reference frame `Fr1` and block size `bsz`; and 3 optional parameters: moving speed `mspd`, region of searching `sreg`, and up-bound MSE `eps`. By default, `mspd = [1 1], sreg = [5 5]` and `eps = 0`.

Function `decodeFrame`: This function decodes motion vectors and residual error frame to predicted frame following the BMA in forward prediction mode. This functions take 4 mandatory parameters: reference frame `RefFr`, motion vectors `MV`, residual error frame `RE` and the block size `bsz`.

Q.2. *Implement the predicted current frame using the information provided by question 1 for different equal block sizes:* 8x8; 16x16 *and* 32x32.

When testing with different block sizes (8x8, 16x16, 32x32), a problem regarding the block size appears. If the width (or height) of frame is not an integer times of the width (or height) of block (case of 32x32), the result is not good. Then I improve functions `encodeFrame` and `decodeFrame`. If the width (or height) of frame is not an integer times of the width (or height) of block I expand frames (frame to encode and reference frame) following horizontal (or vertical) direction by using zeros padding in order to have integer times of the width (or height) of block. After encoding or decoding frame I cutoff zeros padding. Figure 2.1 shows result of my test with different block sizes.

(a) reference frame (b) current frame

(c) 8x8 (d) 16x16 (e) 32x32

Figure 2.1: Predicted frames with different block sizes

Q.3. *Compare the predicted image with its original version. Specify the metric that you have used.*

To compare the predicted image with its original version I use MSE and PSNR. The result of tests shows that MSE between predicted image and its original version is 0 and the PSNR between them is infinitive. This is because of motion compensation in residual error frame.

Q.4. *Make sure your programs are working properly (questions 1, 2 and 3).*

Yes, it works well in tests. The figure 2.1 shows predicted frames with different block sizes. In all cases, the MSE between predicted frame and current frame equals 0. The PSNR between them is infinitive.

Q.5. *Propose a strategy to encode the motion vectors (lossless or lossy compression?) You should justify your choice?*

In my program I do not encode motion vectors. However, if it is encoded, the motion vectors must be encoded in lossless encoding because if we lose information of motion vectors, we are not able to get the position of the block.

Reduction of Spatial and Temporal Redundancies: Video Sequence 3

Q.1. *By relying only on the results provided in Parts I and II, propose a video encoder for the following three cases:*

a) *GOP = I (using part I)*

b) *GOP = IB (using part I and II)*

c) *GOP = IBB (using part I and II)*

a) GOP = I: In this case of GOP all frames in the video are encoded as I-frames. I use the basic version of JPEG which is implemented in the section 1 to encode these frames.

b) GOP = IB: In this case of GOP the odd frame is seen as I-frame, and the even frame is seen as B-frame. Since B-frame can be predicted from a previous frame (`list0` mode), or a subsequent frame (`list1` mode), or all of two previous and subsequent frames (`bipredictive` mode), I then use the prediction mode `list0` to facilitate using implemented codes in the section 2. This prediction mode works with all four sizes of block 16 by 16, 16 by 8, 8 by 16 and 8 by 8.

c) GOP = IBB: In this case of GOP, the first frame is seen as I-frame, the second and third frames are seen as B-frames, the fourth frame is seen as I-frame, etc. Since a B-frame must be predicted based on I- or P- frame(s) then two consecutive B-frames are predicted based on its previous I-frame because of using the prediction mode `list0`.

Q.2. *Implement the proposed solution (given to the previous question).*

a) GOP = I: In this case, I implement a function called `compressIframe` to return a compressed frame based on the basic JPEG version which is implemented in the section 1. This function does tasks of all encoder and decoder. It encodes frame to compress with given quality factor to DC, AC coefficients and decodes these coefficients to obtain compressed frame. I use this function to compress Y, U, V frames in a video to get compressed Y, U, V frames then use the given function `yuv_export` to create video file.

b) GOP = IB: In this case, I implement two functions. Since in the prediction of B-type, the encoder do not send residual errors to the decoder, then I implement the function `decodeBframe` in order to accord to this process. Function `decodeBframe` is a modified version of the function `decodeFrame` which is implemented in the section 2. After that, I implement function `compressBframe`. This function also does all tasks of encoder and decoder. Firstly, it gets motion vectors between I- and B-frame *(task*

of encoder) by using the function `encodeFrame` implemented in the section 2. Then these motion vectors and compressed I-frame are sent to the function `decodeBframe` to obtain compressed B-frame *(task of decoder)*. Since the encoder uses I-frame as reference frame and the decoder uses compressed I-frame as reference frame then function `compressBframe` needs four parameters: B-frame, I-frame, compressed I-frame and block size of prediction mode.

c) GOP = IBB: This case is very similar to the previous case (GOP = IB). There is only a change, it is in reading frames of a video. In the previous case, following an I-frame is a B-frame. In this case, following an I-frame is two B-frames.

Q.3. *Discuss the PSNR behavior with respect to the bit-rate for different quality factors and block sizes (provide a graph).*

To evaluate the influence of quality factor, block size and GOP on PSNR I do tests on the video `foreman.yuv` in two cases: fixing block size to assess the influence of quality factor and GOP on PSNR; fixing GOP to judge the influence of quality factor and block size on PSNR.

In the first case, I fix the block size to 16 by 16 *(with GOP = IB and GOP = IBB)* and evaluate the influence of quality factor and GOP on PSNR. The graph of evaluation is shown in the figure 3.1.

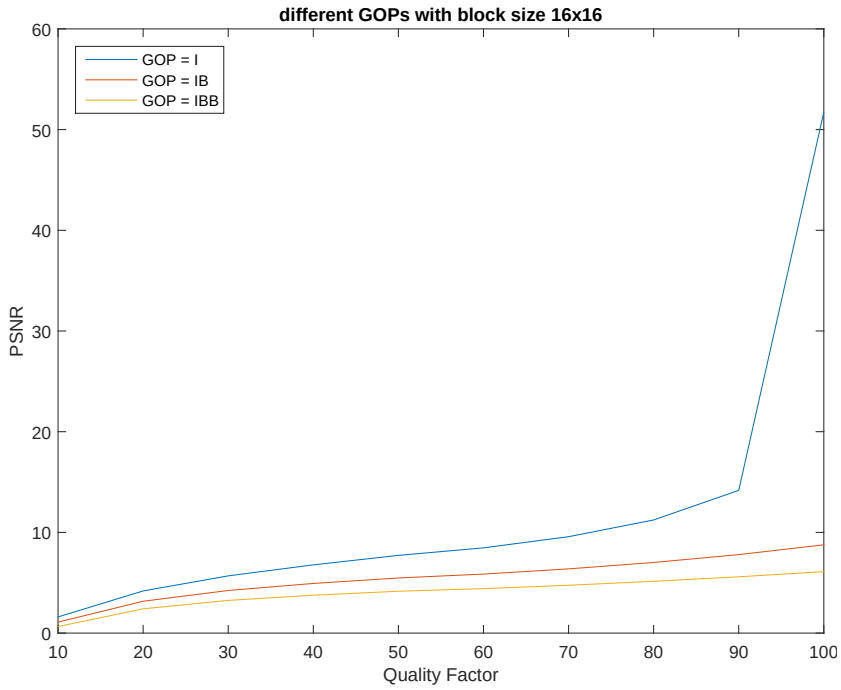


Figure 3.1: Influence of quality factor, GOP on PSNR

In the figure 3.1, we can see that in a GOP, the higher quality factor makes better PSNR. This is easy to understand, because higher quality factor makes smaller distortion between predicted I-frames and its original version. The quality factor has significant influence on I-frame, and the quality of decoded B-frames depends on the one of respective compressed I-frames. That is why the PSNR of all cases of GOP is influenced by the quality factor.

The figure 3.1 also shows that at the same quality factor, the GOP = I has the best PSNR and the worst PSNR belongs to the case of GOP = IBB. This is because the quality of I-frame is better than the one of B-frame. Thus, the quality of the prediction of GOP = I is better than the one of GOP = IB and the one of GOP = IB will better than the one of GOP = IBB.

In the second case of tests, I fix the GOP = IB and evaluate the influence of quality factor and block size on PSNR. The graph of evaluation is shown in the figure 3.2.

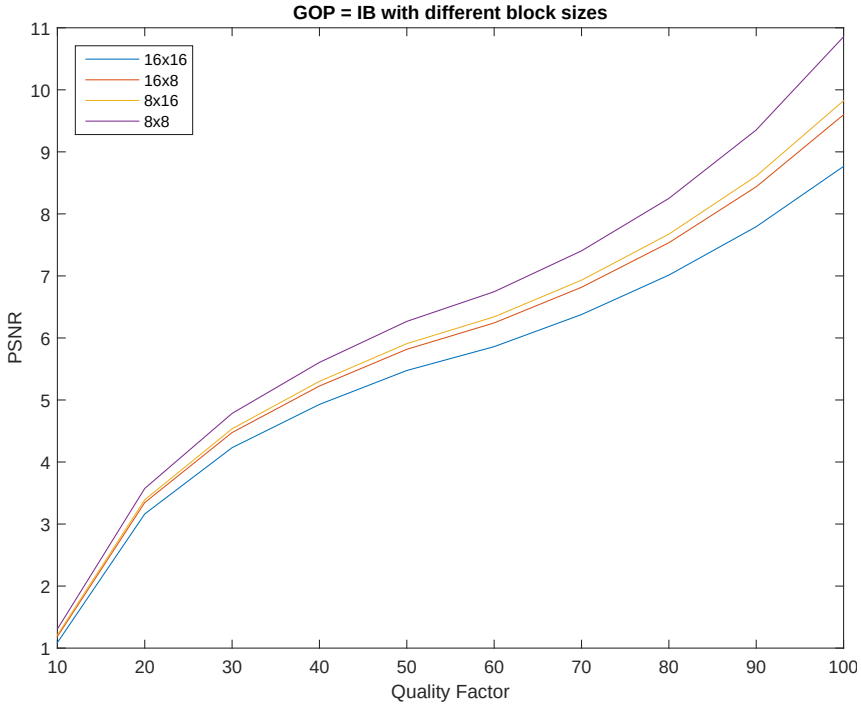


Figure 3.2: Influence of quality factor, block size on PSNR

In the figure 3.2 we can see that in a block size, the higher quality factor makes better PSNR. The reason is very similar to the one of the previous case. That is, the higher quality factor makes smaller distortion between compressed frame with its original version. Thus, in the same condition, as the greater the quality factor is, as the better the PSNR is.

The figure 3.2 also shows that at the same quality factor, the block size 16x16 makes the worst PSNR and the best PSNR belongs to the case of block size 8x8. This is because the larger block size makes higher distortion then in the same condition, the PSNR of block size 8x8 is the best and the one of block size 16x16 is the worst. The PSNR of two block sizes 16x8 and 8x16 are nearly the same. In this case of video and of GOP, the one of 8x16 is better than the one of 16x8 (figure 3.2). However, I think that in a case of other videos or of other GOP of this video this order can be changed.

In conclusion, after testing I found that at the same GOP or block size the higher quality factor makes the better PSNR. In case of the same block size and the same quality factor, the PSNR of GOP = I is better than the one of GOP = IB and the PSNR of GOP = IB is better than the one of GOP = IBB. In case of the same GOP and the same quality factor, the PSNR of block size 8x8 is the best in four kinds of block size (16x16, 16x8, 8x16, 8x8) and the one of block size 16x16 is the worst. The PSNR of block sizes 16x8 and 8x16 are nearly the same and smaller than the one block size 8x8 but higher than the one of block size 16x16.

Bibliography

[1] Anissa MOKRAOUI. *Course Multimedia Representation and Compression.* University of Science and Technology of Hanoi (USTH), May, 2015.

[2] Padma Mundur, Fang Huang. *Course Distributed Multimedia Systems, Lecture 3.* University of Maryland, Baltimore County (UMBC), Spring, 2001.

[3] Zhou Wang, Bovik, A.C. *Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures.* Signal Processing Magazine, IEEE, Volume: 26, Issue: 1, Page(s): 98 - 117, Jan. 2009.