Project : Video Compression

UE Data compression

Groupe
SCHNEIDER Adrien
HENI Yahia
BECKER Gonzalo

Date d'édition : 2 avril 2024
Version : 1.0

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Index

1. Introduction

Video compression is crucial for efficiently storing and transmitting video data. It reduces the amount of data needed to represent a video while maintaining an acceptable level of quality. Several techniques are employed in video compression, and some of the key ones include Discrete Cosine Transform (DCT), quantization, Run-Length Encoding (RLE), and color space conversion such as YUV or RGB. All of these methods will be used in this work, following the guide defined in the flux diagram of Figure **??**.

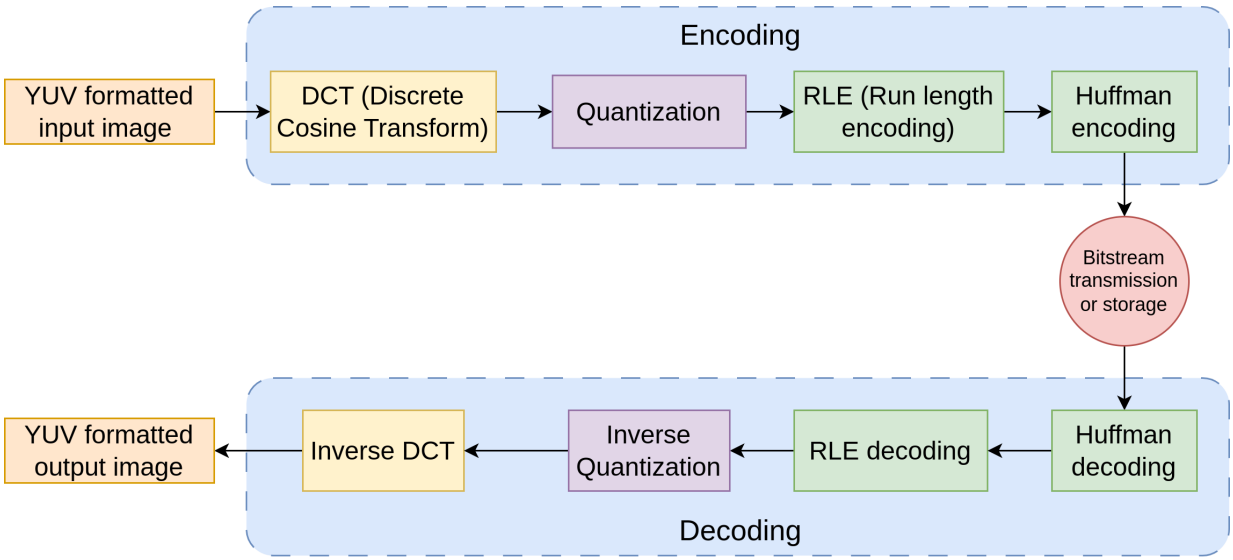


Figure 1 – Flux diagram of the compression and decompression scheme

In short, this algorithm will take the individual images and transform them to the frequency domain by using DCTs. Only the main components of this decomposition will be retained. Once this is done, a matrix of coefficients will be obtained. It is by smartly walking through this matrix that 0 value aggregation is ensured, allowing then to efficiently encode the sequence by using the RLE and Huffman methods consecutively.

Concerning the MATLAB codes, JPEG compression for a single image can be found at test_script.m, which is then implemented inside JPEG.m and JPEG_Decomp.m. Video compression is implemented in VideoComp.m. Predictive coding is partially coded in Predictive.m.

2. YUV decomposition

YUV decomposition is a color space representation where an image is separated into luminance (Y) and chrominance (U and V) components (see Figure 2). The motivation behind it lies in the human visual system's higher sensitivity to changes in brightness (luminance) compared to changes in color (chrominance). By separating luminance and chrominance, more efficient compression and processing algorithms can be developed, catering to human perception characteristics. This separation was crucial in the development of early color television systems and continues to be widely used in digital video encoding and processing.

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.09991 & -0.33609 & 0.436 \\ 0.615 & -0.55861 & -0.05639 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$
$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1.28033 \\ 1 & -0.21482 & -0.38059 \\ 1 & 2.12798 & 0 \end{bmatrix} \begin{bmatrix} Y' \\ U \\ V \end{bmatrix}$$

FIGURE 4 – Enter Caption

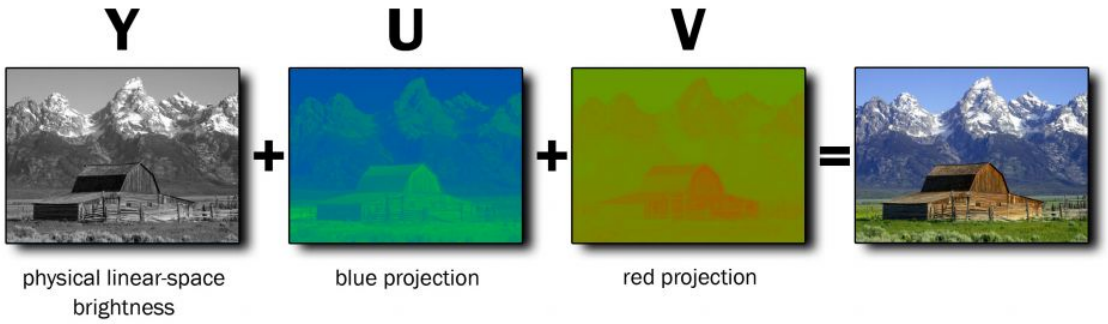
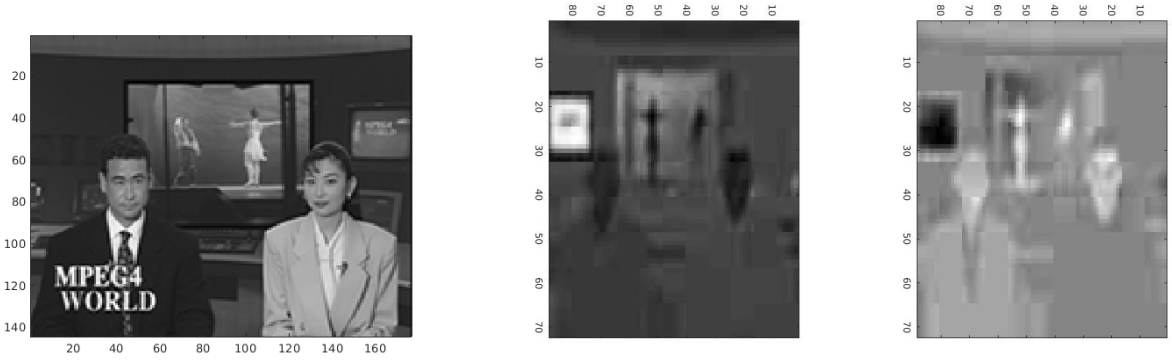


FIGURE 2 – Enter Caption

The videos used in this project are in format .qcif (176x144 pixels) and are given in the YUV format. For instance, the decomposition is observed in Figure 3.



(a) Y (b) U (c) V

FIGURE 3 – YUV decomposition

An RGB representation can easily be obtained by applying a simple reconstruction : Note, however, that a downsampling by a factor of 2 was performed on the U and V components, so that an upsampling is necessary for rescaling.

3. Transform coding : Discrete Cosine Transforms (DCT)

DCT is the basis for many popular video compression standards, such as MPEG. DCT is applied to blocks of pixels in each frame of the video. In this case, each block has a dimension of 8x8. By transforming spatial data into frequency data, it becomes easier to discard less perceptually significant information. For instance, when the DCT is applied over each block, $8x8 = 64$ coefficients are obtained. In what follows, tests have been carried out by selecting the 16 and 32 frequency components with the highest energy, and discarding the others.

In Figure 5, the number of components is varied between 16 (5a), 32 (5b) and 64 (5c). In any case quantization was performed. While at 16 components the degradation of the image is very noticeable, at 32 the compression improves significantly to the human eye. At 64 components, the image is completely recovered. From now on, all of the 64 components will be used.

(a) 16 components (b) 32 components (c) 64 components

Figure 5 – Effect of the number of components

4. Quantisation

After the DCT is applied, a quantisation is performed to further compress the data. Since the DCT gives the coefficients for each 8x8 block, the same quantisation matrix is applied over each one of these blocks. Then, each coefficient will be divided by the corresponding coefficient in the following matrix :

$$M = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix}$$

In JPEG compression, the psycho-visual matrix, also known as the quantisation matrix, plays a crucial role in determining how visual information is discarded or preserved during the compression process. The psycho-visual matrix accounts for the limitations of human visual perception by allocating fewer bits to represent image details that are less noticeable to the human eye.

Once this operation is performed, the resulting values are rounded to integers, so that effectively the number of levels of the discrete representation will have now decreased. Now, a zigzag operation will be carried out, as shown in Figure 6, so as to obtain a flat representation of the DCT coefficients for all blocks.

This zigzag will tend to group higher values (top-left) and 0 values (down-right) together, leading to a more efficient RLE (Run Length Encoding).

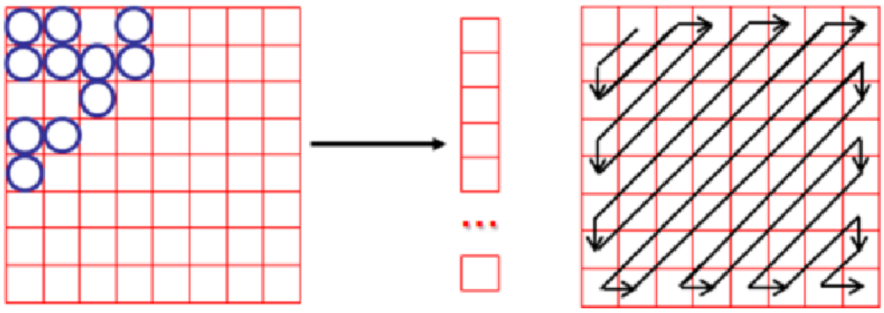


Figure 6 – Zigzag in DCT blocks

Finally, the quality factor Q is considered. For every coefficient q_0 in M, the following transformation is made :

$$q_Q = q_0 + \frac{1 - q_0}{100} Q$$

As Q becomes higher, M gets closer to an all-one matrix, in which case no weighting would be made over the selected DCT coefficients.

5. Run Length Encoding (RLE)

Run-length encoding (RLE) is a simple form of data compression where consecutive identical data values are replaced by a single value and a count of how many times it repeats. In this way, a lossless compression is achieved in an efficient and fast way. In this case, the format of the compression will be the following :

1. **r** : Number of zeros before next non-zero element
2. **s** : Number of bits that the following non-zero element will use
3. **c** : The non-zero number of s bits

This coding is performed for each non zero element.

6. Huffman Encoding

Huffman encoding works by assigning variable-length codes to input characters, with shorter codes assigned to more frequently occurring characters. In our case, the idea is to link each (r, s) couple to a single, uniquely decodable code. This implies that the probability of ocurrence will have to be computed for each (r, s) pair by performing a count of ocurrences in our sequence. This is shown in the histogram of Figure 7, where we observe that there exists a significant variance in the probability of different couples, what justifies the usage of an entropy based encoding.

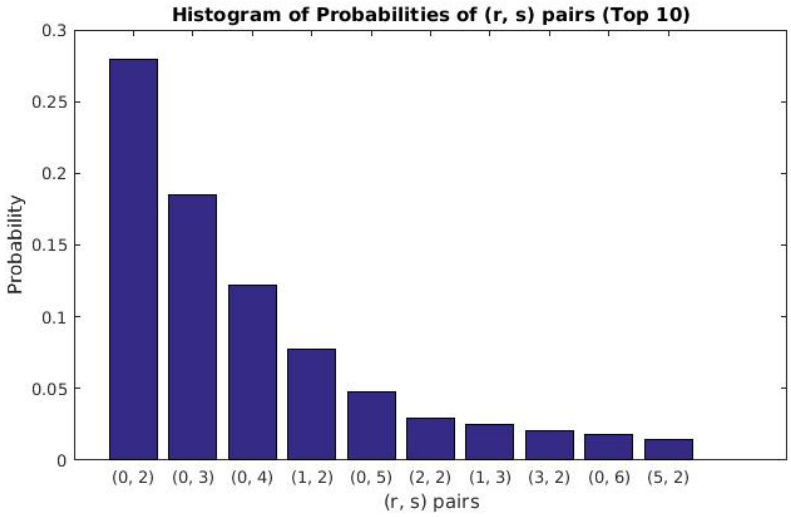


FIGURE 7 – Histogram of (r, s) couples

Once the code is generated, the whole sequence is coded and sent for transmission and/or storage.

7. Metrics

Different metrics can be computed between a compressed image and the original one :

1. **Compression Rate** : Total number of bits used in original image divided by total number of bits in compressed image.
2. **Bits per pixel (bpp)** : Number of average bits used per pixel.
3. **PSNR** : Peak Signal to Noise Ratio

Metrics can be found on table 1.

TABLE 1 – CAPTION HERE

Nb composantes	Q	0	10	30	60	90	100
16	Compression rate	10,75	10,30	9,30	7,63	5,35	3,89
	Bits per pixel	0,74	0,78	0,86	1,05	1,50	2,06
	PSNR	9,31	9,36	9,53	10,06	12,97	23,08
32	Compression rate	7,73	7,33	6,55	5,23	3,41	2,24
	Bits per pixel	1,03	1,09	1,22	1,52	2,35	3,57
	PSNR	9,33	9,37	9,56	10,10	13,15	27,63
64	Compression rate	6,87	6,50	5,74	4,41	2,63	1,40
	Bits per pixel	1,17	1,23	1,39	1,82	3,04	5,68
	PSNR	9,33	9,39	9,56	10,11	13,21	58,96

Overall, we can extract the following conclusions :

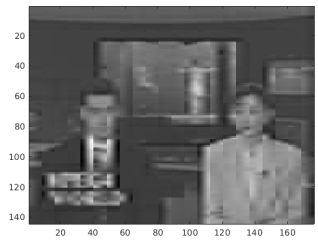
1. As Q increases, the compression rate decreases, and bits per pixel and the PSNR increase. Equivalently, as Q increases the image is less compressed and so more bits are needed and a better quality is achieved.

2. Similarly, as the number of DCT considered components increases, the compression rate decreases and the bits per pixel increase. However, for low values of Q the PSNR shows very small variations for different values of selected coefficients. Otherwise said, if a low Q is to be used, we might as well choose a small number of DCT coefficients since there is not a great advantage of selecting lots of them.

8. Reconstruction

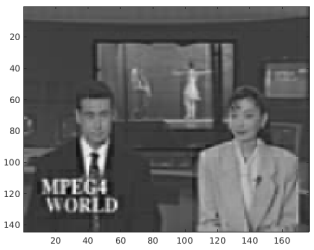
At some point, the compressed image will be retrieved and process for visualization and processing, in which case all previous steps will have to be undone. For this purpose, the decoding algorithm will need to know the Huffman coding table and some other variables, such as the dimensions of the original image or the size of blocks used in the DCT.

Then, step by step, all operations are reversed and the expected image is obtained. If Q < 100 or not all all coefficients of the DCT have been considered, some degree of data loss is expected. This is observed in Figures **??**.

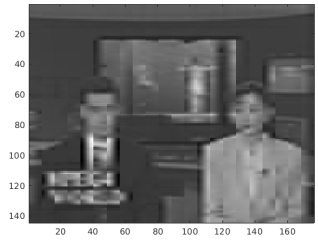


(a) Q=10, 16 DCT components

(b) Q=10, 32 DCT components

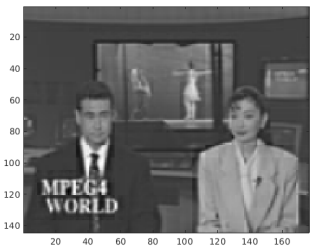


(c) Q=10, 64 DCT components

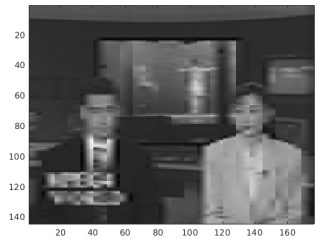


(d) Q=60, 16 DCT components

(e) Q=60, 32 DCT components



(f) Q=60, 64 DCT components



(g) Q=100, 16 DCT components

(h) Q=100, 32 DCT components

(i) Q=100, 64 DCT components

FIGURE 8 – Images with different Q and comp values

9. Video Compression

JPEG compression is independently applied to all frames in a given video. As can be observed in Figure 9, where $Q = 100$ and all the 64 components are used, the results are satisfying. Aditionally, in this case the YUV decomposition is processed so as to display the image in color.

FIGURE 9 – Decompressed video

10. Predictive Coding

Predictive coding exploits data redundancy to achieve video compression with acceptable quality deterioration. To use it, the following steps should be followed. An incomplete implementation can be found at file `predictive.m`.

1. **Load Video**
2. **Choose a Period : IBBPBBPBBPBBI** :
   - **I frames (Intra-coded frames)** : These frames are encoded independently of other frames and serve as reference points for the subsequent frames.
   - **P frames (Predicted frames)** : These frames are encoded based on the previous I or P frames, using motion compensation to predict their content.
   - **B frames (Bi-directional frames)** : These frames are encoded using information from both past and future reference frames, providing more efficient compression.
3. **Apply JPEG to I Frames (After Choosing a Period k=10)** :
   - I frames are typically compressed using techniques similar to JPEG compression.
4. **Calculate P Frames Using Surrounding I or P Frames (Motion Compensation)** :
   - **Prediction by block similarity** : P frames are predicted from surrounding I or P frames by finding similar blocks of pixels in the reference frames.
   - **Motion compensation** : The predicted frame is then modified by taking into account the motion of objects in the scene. Differences between the original frame and the predicted frame (error image) are encoded using techniques such as DCT, quantization, and entropy coding.
5. **Calculate B Frames Using Surrounding I or P Frames** :
   - B frames are predicted from surrounding I or P frames using two predictions based on block similarity. The error image with respect to either the previous frame, the next frame, or the average of the two predictions is encoded using DCT, quantization, and entropy coding.