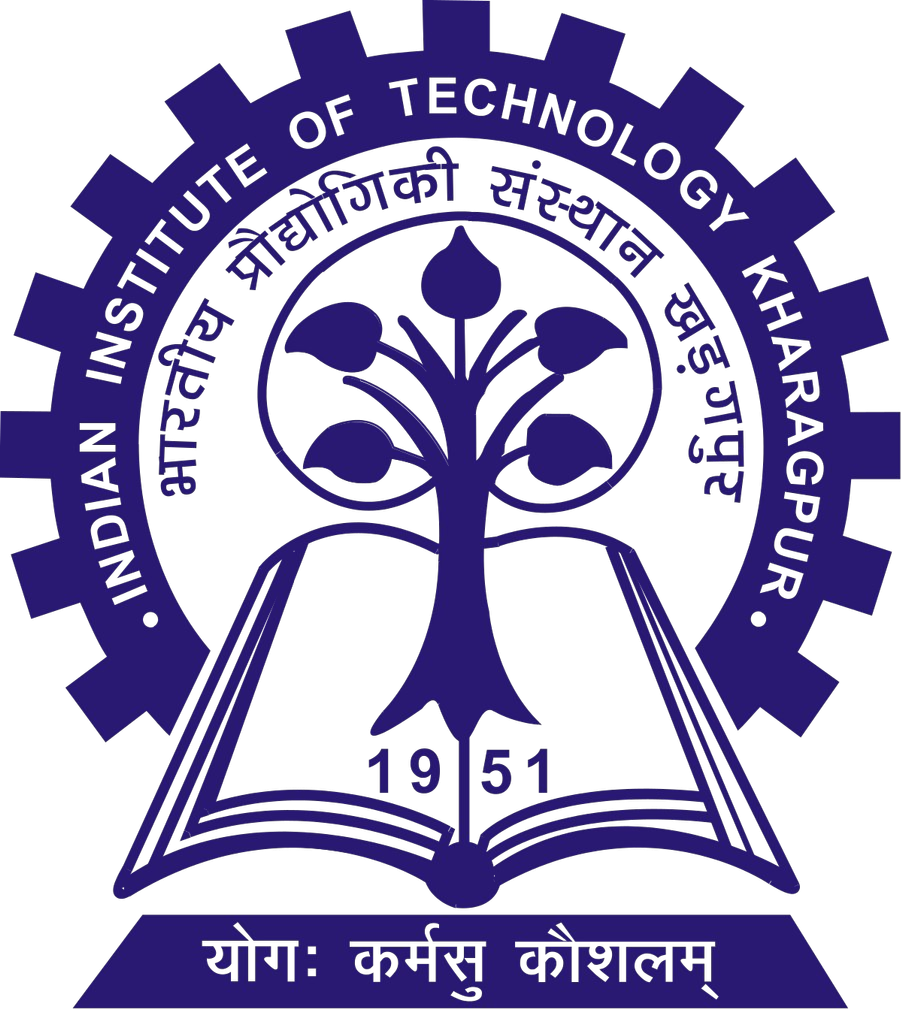
**INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR**



**Department of Electronics & Eletrical Communication Engineering Vision and Intelligent Systems**

**EC69211 – Image and Video Processing Laboratory**

**Mini Project**

**JPEG Compression**

**Submitted by –**

Saurabh Mehra (23EC65R06)

Anshu Pal (23EC65R04)

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sl No** | **Content** | **Pg No** |
| 1 | Cover Page | 1 |
| 2 | Contents | 2 |
| 3 | Objective | 3 |
| 4 | Theory | 3-5 |
| 5 | Algorithms | 5-6 |
| 6 | Results | 7-12 |
| 7 | Discussion | 12-13 |
| 8 | References | 13 |

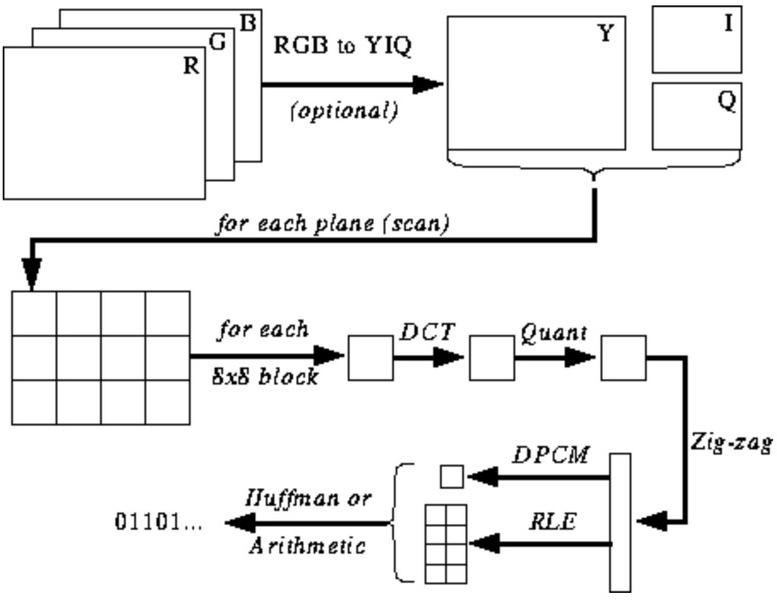
# Objective:

1. Designing and implementing entire JPEG pipeline
2. Reading image in any raw format, passing it through JPEG encoder and storing output encoded image on disk

# Theory:

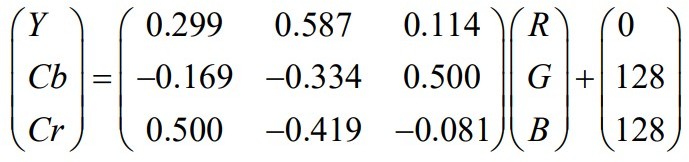
Overall JPEG pipeline

The JPEG encoding consists of multiple steps which are pictorially shown below



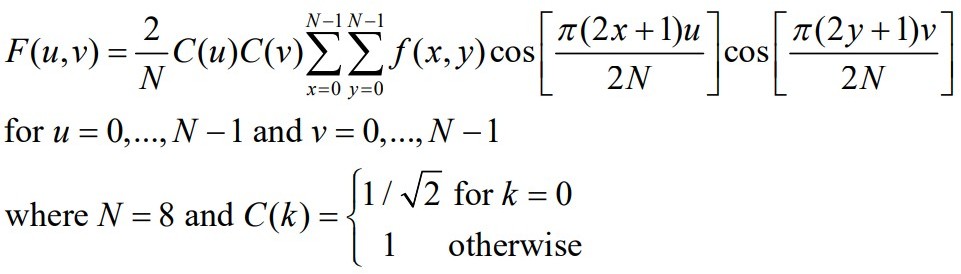
Color Specification

The YUV colour coordinate defines Y, Cb, and Cr components of one color image, where Y is commonly called the luminance and Cb, Cr are commonly called the chrominance. Describing of a colour in terms of its luminance and chrominance content separately enable more efficient processing and transmission of colour signals in many applications.

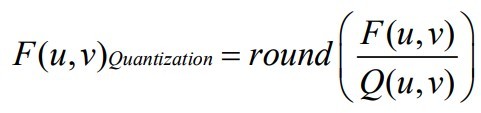


Discrete Cosine Transform

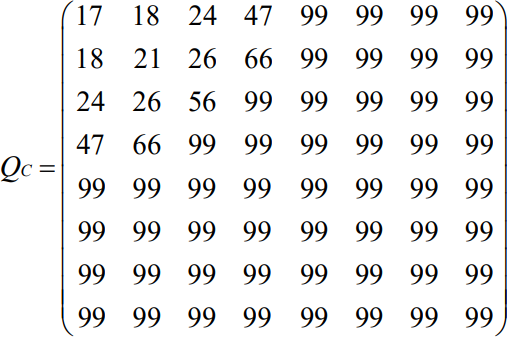
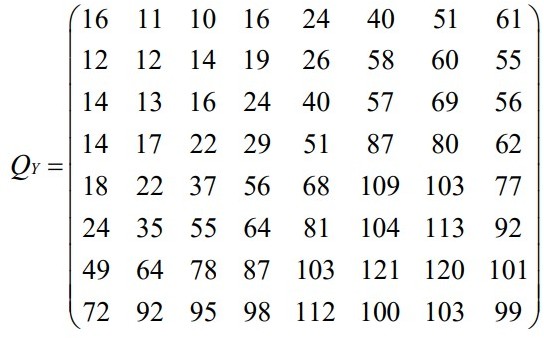
After colour coordinate conversion, the next step is to divide the three colour components of the image into many 8×8 blocks. For an 8-bit image, in the original block each element falls in the range [0,255]. Data range that is centred around zero is produced after subtracting The mid-point of the range (the value 128) from each element in the original block, so that the modified range is shifted from[0,255] to [-128,127]. Images are separated into parts of different frequencies by the DCT.



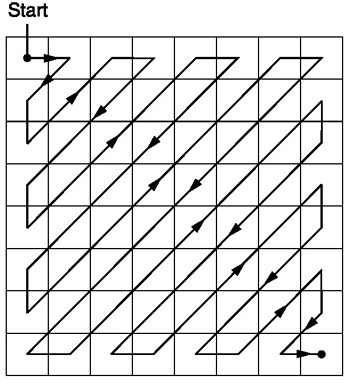
Quantization

We actually throw away data through the Quantization step. We obtain the Quantization by dividing transformed image DCT matrix by the quantization matrix used . Values of the resultant matrix are then rounded off.

There are standard quantization tables for JPEG compression and they are constructed in a way that when we divide each value by two and use the resultant matrix, one cannot notice significant degradation in quality of the output image. Standard Qunatization matrices for luminance and chrominance components respectively are as follows:

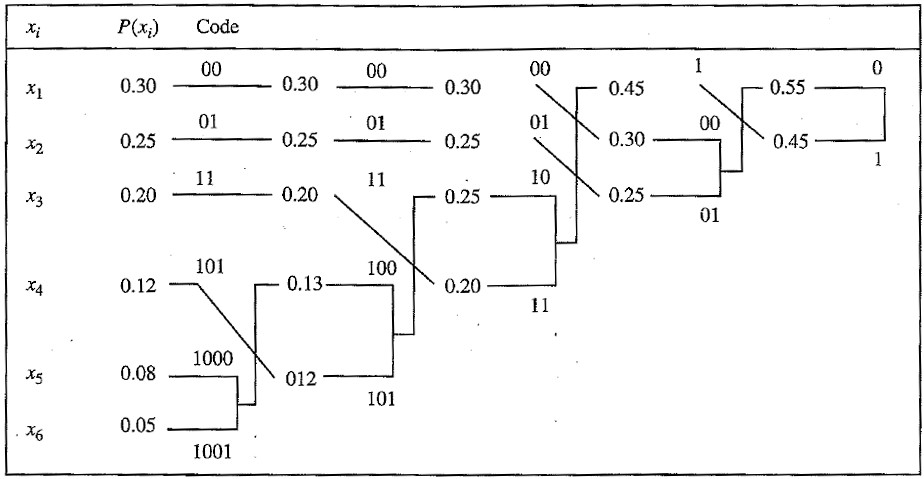


Zig-zag ordering

After quantization, the "zig-zag" sequence orders all of the quantized coefficients. In the "zig-zag" sequence, firstly it encodes the coefficients with lower frequencies (typically with higher values) and then the higher frequencies (typically zero or almost zero).

Huffman Encoding

Entropy Coding achieves more lossless compression by encoding more compactly the quantized DCT coefficients. Huffman coding is used in the baseline sequential codec, but all modes of operation use Huffman coding and arithmetic coding.



# Algorithms:

Following is the algorithm used to perform JPEG compression:

* 1. Read the RGB pixel arrays of input image using OpenCV
  2. Pad each channel to make the dimensions a mulitple of 8
  3. Convert RGB to YCbCr using the matrix transformations
  4. Break the image into 8x8 blocks and carry out the steps below for each block
  5. Compute DCT of Y, Cb, Cr matrix
  6. Divide each element of DCT matrix with corresponding element of quantization matrix and round each element to nearest integer
  7. Apply zigzag ordering on each of quantized matrices
  8. Subtract DC coefficient of current block from that of previous block and let the intermediate symbol be (class)(amplitude). Apply huffman encoding on class and store amplitude as Variable Length Code (VLI)
  9. For AC coefficient, apply run length encoding to get intermediate symbols of the form (number of preceding zeros, class)(amplitude) with the first part being huffman encoded and the second part as VLI
  10. After each occurence of 0xFF in the data add a 0x00 as mentioned in JPEG standard
  11. Write all the encoded data, Huffman tables, Quantization tables and other specifications in appropriate segments starting with corresponding segment marker

Following is the algorithm used to do run-length encoding:

1. Find the last index after which all the entries are 0s in the data stream.
2. Maintain 2 pointers, one for the starting index and the next for the current index to find the length of 0s.
3. Loop the current index till the last index and if the current index value is not 0 or if the length of the 0s is 16 then store the run length code. Otherwise just move the current index to the next index.
4. Keep storing the run length code in a list and break out once you reach the last index.
5. Append [0,0] at the end to the run length code to indicate that all the terms are 0 after this.
6. Return the obtained list and this is the run length code.

Following is the algorithm used to generate Huffman code:

1. Calculate the frequency of each unique value from the data stream and then divide it by the total number of values to get the probability of occurrence of the value.
2. Sort the values by their probability from lowest to highest.
3. Choose 2 of the values with the lowest probability and merge them to create a new value with the probability as the sum of those 2 values.
4. Sort the new set of values again based on the new probability distribution.
5. Repeat this process till there is only a single value left.
6. The merged nodes would have formed a tree structure, which is the Huffman tree.
7. Traverse the tree by taking the left child as “0” and right child as “1” and append consecutive values till you reach the leaf node and this will be the Huffman code.

Algorithm used to compute Fast DCT is given in [[4]](#_bookmark0)

# Results:

The output images along with their attributes are given below:

Cameraman 0.5 Size ---12.3kb



Cameraman 1 size-----------------8.24kb



Cameraman 10 size------------5.61kb



Cameraman 25 size---------------------4.31kb



Cameraman 50 size-----4.30kb



Corn 0.5 size---------44.8kb



Corn 1 size----------------28kb



Corn 2 size-------------17.7kb



Corn 8 size----------------6.33kb



Corn 10 size-----------------5.35kb



Corn 25 size-----------------3.47kb



Corn 50 size-----------------3.46kb



Lena 0.5 size-----------------14.3kb



Lena 1 size-----------------10.6kb



Lena 2 size-----------------6.33kb



Lena 10 size-----------------2.21kb



Lena 25 size-----------------1.68kb



Lena 50 size-----------------1.68kb



From the above images and their properties, we got to see the trade-off between compression and quality of output image. Depending on the application, one needs to choose the quality factor so that the quality is good enough for their purpose while getting some compression.

# Discussion:

* While encoding the image, we always consider 8x8 blocks. Whenever the image dimensions are not multiples of 8, we must pad the image by some strategy to make its dimensions multiple of 8. In JPEG encoding, zero padding is not a good strategy because it will add lots

of high frequency components and compression will not be good. Instead, it is a good choice to pad using the same values as that at the edges and corners of the original image. Also, we mention the height and width of the image in SOF segment due to which the image decoder can always display the image with original dimensions.

* The end of block code (0,0) must be added only if the length of the zigzag length sequence is less than 64. It was observed that when the length was equal to 64, the decoder automatically breaks from current block and scans the next block.
* The quantization matrix is a measure of how much compression we want in the encoded image. If the matrix contains higher values, there will be lesser distinct symbols to be encoded and hence compression will be higher. There are some standard quntization tables

which can be used to get good compression while not sacrificing much on the quality of the output image. When the values of the standard matrix are halved, there was no significant degradation in image quality.

* It is interesting to note that JPEG decoder reconstructs the Huffman tree just from the information mentioned in the Huffman segment which contains the number of codes having

particular length and the values associated with codes of each length. It can decode unambiguously because there is predefined pattern in which Huffman codes must be used. For Huffman codes, it is the length of the code which matters and not the exact code. So in JPEG, Huffman codes of a particular length are generated as consecutive numbers (ex – 000, 001, 010, 011, ...) and whenever there is an increase in length we must multiply 2 with the current code (ex – 010, 0110, 0111). There is another restriction that no code must have all 1’s in it which can be handled by skipping the code containing all 1’s and going to next code following the pattern mentioned above.

* All segment markers in JPEG starts with 0xFF. As per JPEG standards whenever actual data contains the byte 0xFF we must add a dummy byte 0x00 so that decoder can distinguish whether is a segment marker or it is actual data.
* JPEG compression is a lossy compression and hence must only be used when we can afford to lose some information from the image. For example, if the image is used for entertainment purpose then JPEG can be used as we found that the low compressed image of

lena had no noticeable degradation in quality. However, JPEG must not be used in certain medical imaging processes as it might need high details of the image and losing information might give bad results.