Lebanese University
Faculty of Sciences 1

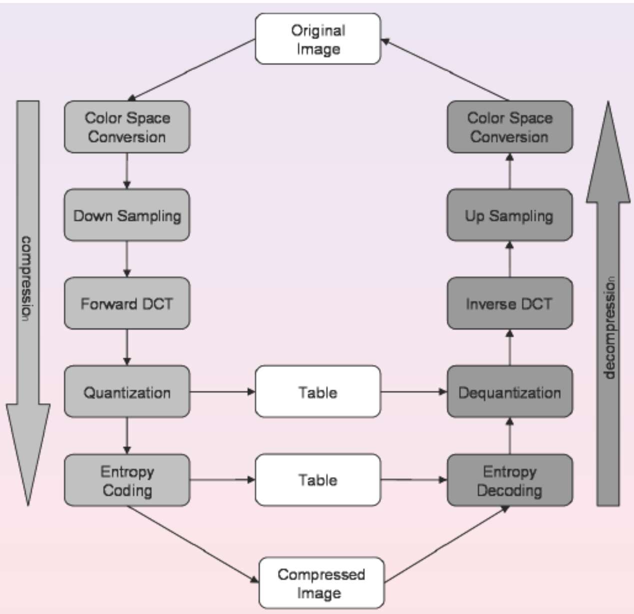
INFO 430

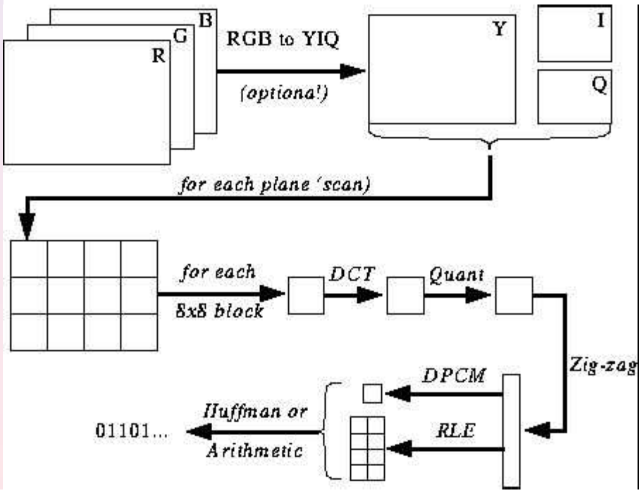
Multimedia

Mini Project 19.20
Duration: 2 weeks

Implement the JPEG standard for image compression

This is to code the JPEG standard with Matlab, of course we will make a primary version. You are asked to program the compression and decompression algorithm while progressively performing the requested functions. DO NOT use the functions ready in MATLAB like rgb2ycbcr to realize rvb2yuv for example. If not you will not be rated. Each of the following functions must be coded independently, taking the data as input parameter and not global variables, and the results also as output, to facilitate testing on several images.





1. resize.m	14. vect2RLE.m
2. rgb2yCbCr.m	15. RLE2vect.m
3. yCbCr2rgb.m	16. DC2DPCM.m
4. downsample.m	17. DPCM2DC.m
5. upsample.m	18. vect2Huffman.m
6. data2DCT.m	19. Huffman2vect.m
7. DCT2data.m	20. save2file.m
8. DCT2quant.m	21. loadfile.m
9. quant2DCT.m	22. im2JPEG.m
10. mat2ZigZag.m	23. JPEG2im.m
11. zigzag2mat.m	24. GUI main.m
12. mat2DC.m	25. small report
13. DC2mat.m	

1- Function *p_resize.m* :

   a. input = 1 matrix (example one color component R, G or B of the image)

   b. output = 1 matrix reduced

   c. remplace each 4×4 bloc of each component by the *mean;* resize the image so that Matlab can process quickly, Matlab is a simulation enviroment much slower than C env., for each channel (r,g,b) resize the matrix, take the average of every 4x4 pixels

   d. test = output the image before and after and check the visual content as well as the reduced dimension to verify the success of the operation

2- Function *p_rgb2yCbCr.m* :

   a. input = 3 matrices (R, G and B)

   b. output = 3 matrices (Y, Cb and Cr)

   c. color conversion as seen in lesson,

   - Y= (R+2G+B)/4
   - Cb=B-G
   - Cr=R-G

3- Function *p_yCbCr2rgb.m* :

   a. convert back to RGB

   b. test = transform the image forth and back and view it to make sure both functions (2 and 3) work properly.

   - G=Y-(Cb+Cr)/4
   - R=G+Cr
   - B=G+Cb

4- Function *p_downsample.m* :

   a. input = 1 matrix

   b. output = 1 matrix reduced

   c. like resize but replace 2×2 blocs.

   d. used in chroma-subsampling : use the 4:2:2 model (4Y:2Cb:2Cr)

pixel_{11}	Y_{11} $(C_{b11}+C_{b12})/2$ $(C_{r11}+C_{r12})/2$	Y_{12}	pixel_{12}
pixel_{21}	Y_{21} $(C_{b21}+C_{b22})/2$ $(C_{b21}+C_{b22})/2$	Y_{22}	pixel_{22}

5- Function *p_upsample.m* :

   a. inverse operation

   b. test

6- Function *p_data2DCT.m* :

   a. input = 1 matrix

   b. output = 1 matrix of DCT coefficients

   c. calculate the DCT coefficients (by blocs of 8×8)

   - DCT(A)= $C = P^t \times A \times P$
   - with P=

$$P = \begin{pmatrix} \frac{1}{\sqrt{8}} & \frac{1}{2}\cos(\frac{\pi}{16}) & \frac{1}{2}\cos(\frac{2\pi}{16}) & \cdots & \frac{1}{2}\cos(\frac{7\pi}{16}) \\ \frac{1}{\sqrt{8}} & \frac{1}{2}\cos(\frac{3\pi}{16}) & \frac{1}{2}\cos(\frac{3*2\pi}{16}) & \cdots & \frac{1}{2}\cos(\frac{3*7\pi}{16}) \\ \cdots & & & & \\ \frac{1}{\sqrt{8}} & \frac{1}{2}\cos(\frac{15\pi}{16}) & & \cdots & \frac{1}{2}\cos(\frac{15*7\pi}{16}) \end{pmatrix}$$

   - elements pij: $p_{ij} = \alpha_j \sqrt{\frac{2}{N}} \cos\left(\frac{(2i+1)j\pi}{2N}\right)$
   - with $\alpha(u) = \begin{cases} \sqrt{\frac{1}{2}} \text{ pour u} = 0 \\ 1 \text{ pour u} = 1,\ldots,N-1 \end{cases}$

   d. **Note** in Matlab π is constant *pi*.

7- Function *p_DCT2data.m* :

a. inverse operation

$$f(x,y)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\alpha(u)\alpha(v)C(u,v)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

b. test both functions by searching for the maximum of difference between the two matrices (original and recovered). It must be small for a low frequency bloc, as seen in course.

8- Function p_DCT2quant.m :

a. input = 1 matrix of DCT coefficients

b. output = 1 quantized matrix

c. use the matrices seen in lesson. Note that this step permits to choose the compression degree/quality of JPEG compressed images, and Q differes according to the chosen compression rates, as well as according to luma/chroma bloc type.

d. Example :

16*Q=

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

9- Function p_quant2DCT.m :

a. inverse operation

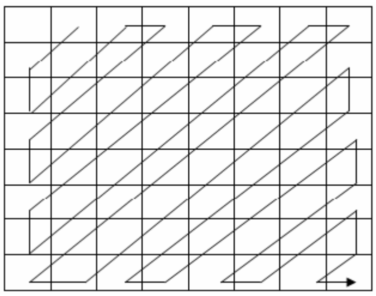
b. test = test functions 8 & 9 together checking the difference especially for DC coeffiecients (before/after), it must stay almost untouched.

10- Function p_mat2ZigZag.m :

a. input = 1 matrix

b. output = 1 vector = matrix read in ZigZag

c. attention DC coefficients DC not included, only AC.



11- Function p_ZigZag2mat.m :

a. inverse operation

b. test

12- Function p_mat2DC.m:

a. input = 1 DCT matrix

b. output = 1 vector containing one DC coefficient

13- Function p_DC2mat.m :

a. input = 1 matrix (output of ZagZag2mat) + 1 DC

b. output = reconstructed matrix

c. test

14- Function p_vect2RLE.m :

a. input = 1 vector

b. output = 1 vector coded in RLE version DCT. (count the number of zeros preceding each non-zero value)

15- Function p_RLE2vect.m :

a. inverse operation
b. test functions 14 & 15, it must be a lossless transformation.

16- Function *p_DC2DPCM.m* :
a. input = vector of DC (output from *mat2DC)*
b. output = vector coded in DPCM
c. coding of DC coefficients

17- Function *p_DPCM2DC.m* :
a. inverse operation
b. test = also 16 & 17 are lossless operations

The two functions 18 and 19 which follow are reserved for the more advanced, those who find difficulties realizing them can omit them.

18- Function *p_vect2Huffman.m* :
a. input = 1 vector (output of *vect2RLE.m* or 1 vector output from *DC2DPCM.m*)
b. output = 1 vector coded Huffman

19- Function *p_Huffman2vect.m* :
a. inverse operation
b. test = lossless

20- Function *p_save2file.m* :
a. input = 6 coded vectors (2 for each componants Y, Cb and Cr)
b. output = text file *.txt*
c. It is your compressed image !!!
d. Remember to concatenate the six vectors one after the other, and use the **save** command of Matlab which saves a vector, since the **load** command reads a vector from a .txt file without needing to know its size (in number of elements) beforehand.
e. e. Attention insert at the beginning of your file information facilitating the proofreading, that is to say, the 6 respective sizes of your 6 vectors…

21- Function *p_loadfile.m* :
a. input = txt file
b. output = 6 vectors
c. inverse operation
d. test = compare the 6 vectors before and after saving, they must be identical.

22- Function *p_im2JPEG.m* :
a. input = image name, for example ' lena_std.tif '
b. output = compressed file

23- Function *p_JPEG2im.m* :
a. inverse operation
b. test of course

24- … and finally… Create a graphical interface in which
a. choose the image to compress and we compress it
b. read and display a compressed image

25- Do not forget to indicate in your report the compression ratio that you achieved for Lena.

Good luck

The Lenna Story - www.lenna.org