CS 576 – Assignment 2

Assigned on Monday 09/28/2009,
Solutions due on Thursday 10/15/2009 (noon)

Part 1: Non Programming

In this question you will try to understand the working of DCT in the context of JPEG. Below is an 8x8 *luminance* block of pixel values and its corresponding DCT coefficients.

188	180	155	149	179	116	86	96
168	179	168	174	180	111	86	95
150	166	175	189	165	101	88	97
163	165	179	184	135	90	91	96
170	180	178	144	102	87	91	98
175	174	141	104	85	83	88	96
153	134	105	82	83	87	92	96
117	104	86	80	86	90	92	103

D	**A**	-6.8	-27.2	29.3	-20.8	-11.2	8.0
B	52.6	-93.5	-7.3	34.0	-18.8	-11.3	10.6
-45.9	-49.2	13.9	53.8	11.1	-24.7	-0.1	8.4
8.8	38.1	47.9	**C**	-17.9	-10.9	4.2	3.7
-1.3	-5.9	-1.2	-4.7	0.8	6.6	4.8	0.2
-4.5	-1.2	3.3	8.1	7.0	6.1	-0.2	1.2
-2.9	-2.1	0.9	-1.5	0.0	-3.4	-0.9	-1.2
-0.8	-3.4	-0.6	-1.8	-4.2	-1.3	2.3	1.6

- Using the 2D DCT formula, compute the values of **A, B, C**, and **D**.
- In the JPEG pipeline, the DCT values are quantized and then further scanned in a zigzag order. For simplicity, assume that you have a uniform quantization table with each value in the table given by Q. Show the resulting zigzag scan when Q=100.

For this zigzag AC sequence, write down the intermediary notation and hence the resulting JPEG bit stream. For the bit stream you may consult standard JPEG VLC and VLI code tables.

Part 2: Programming

The goal of this assignment is to understand a lossy compression technique – *vector quantization.* Although we did not discuss this in detail in class, it is an important technique and a general assignment on this topic will increase you overall understanding of compression. We will use vector quantization to compress color images. Color images, as you know, are represented using three channels R, G and B and in our case each channel uses 8 bits (= 24 bits per pixel). You could consider every pixel as a three-channel vector. The goal then becomes to take the entire sample space of vectors and decide on a few vectors that form a codebook for your image. Each pixel in the image can be represented by the best approximating (or least error) codebook vector. The vectors in the codebook can be represented by an index. Thus if you have a *w*x*h* color image and decide on *n* vectors in your code book, you will compress your image down to $whlog_2n$ bits (down from 24*wh* bits).

You are free to use the code given in the assignment1 as a start for the assignment2. The input to your program will be three parameters where

- The first parameter is the name of the image, which is provided in an 8 bit per channel RGB format (Total 24 bits per pixel). All images are of fixed size.
- The second parameter *n* controls the number of vectors in your codebook.
- The third parameter *m* describes the type of algorithm that you will be implementing - *m* =1 for uniform axis quantization, *m* = 2 for an optimized vector quantization of your choice.

To invoke your program we will compile it and run it at the command line as

YourEncoder.exe C:/myDir/myImage.rgb n m

where *C:/myDir/myImage.rgb* is the image and *n m* are the parameters as described above. As output you will be required to show the compressed image and a channel-by-channel difference between the original and the output. So there will be four images shown as output, the first one will be color followed by 3 gray images.

Implementation Details

The algorithm proceeds as 4 steps.

1. Sampling the original image for vector statistics.
2. Choosing a codebook depending on *n* and *m*.
3. Mapping the image color vectors to a representative vector in the codebook.
4. Quantizing and drawing the new vector indexed image.

For step 1 - Consider the space of all color vectors. There are 3 channels, which can be represented on three axes. Each axes has a possible variation of 256 values (8 bits per channel). You can sample the input image and create a 3D histogram that gives you, which vectors are present in your image and how many of each is present in your image.

For step 2 - Next, for *m*=1, you are expected to do uniform quantization. You can decide based on *n*, how many bits need to be allocated per axis (or channel). Eg if you are using *n*=64, which needs 6 bits per vector, each axis can be 2 bits. If you are using a number of bits per vector that is non divisible by 3, you will need to decide how to split your bits intelligently based on your 3D histogram. In uniform quantization, you will cut your 3D space in uniform volume cells. The number of cells should be equal to *n*. In the suggested example of *n*=64, and each axis which has 2 bits can be partitioned into 4 equal parts dividing the whole cube into 64 equal smaller cells.

Now that you have partitioned your vector space into cells, you need to decide on a representative vector for each cell. Give this some thought - it could be the vector representing the center of the cell or it could be the average of all the vectors in the cell. Either ways, each cell is represented by one vector, which forms a vector in you codebook. This should give you a codebook of *n* vectors

For step 3 – You now have a codebook of *n* vectors, where each cell represents a vector. For each cell, all the vectors then can be mapped to its corresponding codebook vector. Each pixel in the output image is thus mapped in a straightforward lookup (depending on which cell it lies in) to a codebook vector.

For step 4 – As output, your first image will need to display every pixel mapped to its indexed vector. As such there will be a maximum of *n* colors in your output image, since there are *n* codebook vectors. You will also need to display three other images, which represent for each pixel a channel-by-channel difference from the original. Remember all these images will be single channel images and hence gray. This breakdown will allow you to analyze which channel is most distorted. You may choose or modify your algorithms so as to minimize distortions of a chosen channel so as to get the best perceived visual quality of the output.

Lastly, for *m*=2 you will need to implement a better cell division algorithm. In uniform quantization there could be some cells, which have no vector samples in them, or there could be some with a very large number of samples in them, which results in an ineffective assignment of codebook vectors. *Give this some thought and research*. You will need to move your division boundaries per axis based on statistical distributions as represented by your histogram.

Extra credit

The quality of your output depends on how effectively you decide your cells and hence codebook vectors. Although you are asked to come up with just one effective algorithm when *m*=2, you could experiment with a variety of different ways of dividing space for *m*=2. Explain your algorithms and perform a quantitative analysis of how good each one is. You should show per channel error quantifier (which may be the sum of all the differences per channel) and an overall error quantifier.

What should you submit?

- Your source code, and your project file or makefile, if any, using the submit program. Please do not submit any binaries. We will compile your program and execute our tests accordingly.
- Along with the program, also submit an electronic document in pdf format using the submit program that includes your non-programming assignment part. If you chose to do an analysis for extra credit, please include sample output images, comparison tables, error analysis and so on - explaining your algorithms, results etc.