Lab - 9 Solutions: Achromatic Baseline JPEG Encoding

ECE 637: Digital Image Processing 1 - Spring 2017

Aarti Ghatkesar

April 28, 2017

Question 1. DCT Block Transforms and Quantization

MATLAB Code: Q1_Soln.m

```
%% Lab 9: Achromatic Baseline JPEG Encoding Lab
% Author; Aarti Ghatkesar
% Exercise 2.1: Block DCT transforms followed by Quantization - Determine
% effect of gamma


%%

clc
clear
close all

%% Reading Image and converting to double and level shifting by 128

img = double(imread('img03y.tif'));
img = img - 128;

%% Executing script 'Qtables.m' for variables Quant and Zig
run('jpeg\Qtables.m');

%% 8 x 8 Block DCT and Quantization

gm = 4;

fcn = @(x)round(dct2(x.data,[8,8])./(Quant*gm));
dct_blk = blockproc(img,[8,8],fcn);


%% Writing to file

fileID =fopen('img03y_4.dq','w');
fwrite(fileID,size(img),'integer*2');
fwrite(fileID,dct_blk','integer*2');
```

```
fclose(fileID);

%% Reading from Binary file

fileID = fopen('img03y_4.dq','r');
A= fread(fileID,'integer*2');
fclose(fileID);
rows = A(1);
cols = A(2);
quant_dct_coeff = reshape(A(3:end),cols,rows);
quant_dct_coeff = quant_dct_coeff';

%% Inverse operations

fcn = @(x)(x.data.*Quant*gm);
dct_blk = blockproc(quant_dct_coeff,[8,8],fcn);

fcn = @(x)round((idct2(x.data,[8,8])));
restored_img = blockproc(dct_blk,[8,8],fcn);
figure
imshow(restored_img,[]);
title(sprintf('Restored Image gamma = %f',gm),'Interpreter','LaTex');

diff_img = (img - restored_img).*10 +128;
figure
imshow(diff_img,[]);
% title('Difference Image');
```

Figure 1: Original Image

Restored Image gamma = 0.250000

Figure 2: Restored Image for $\gamma = 0.25$

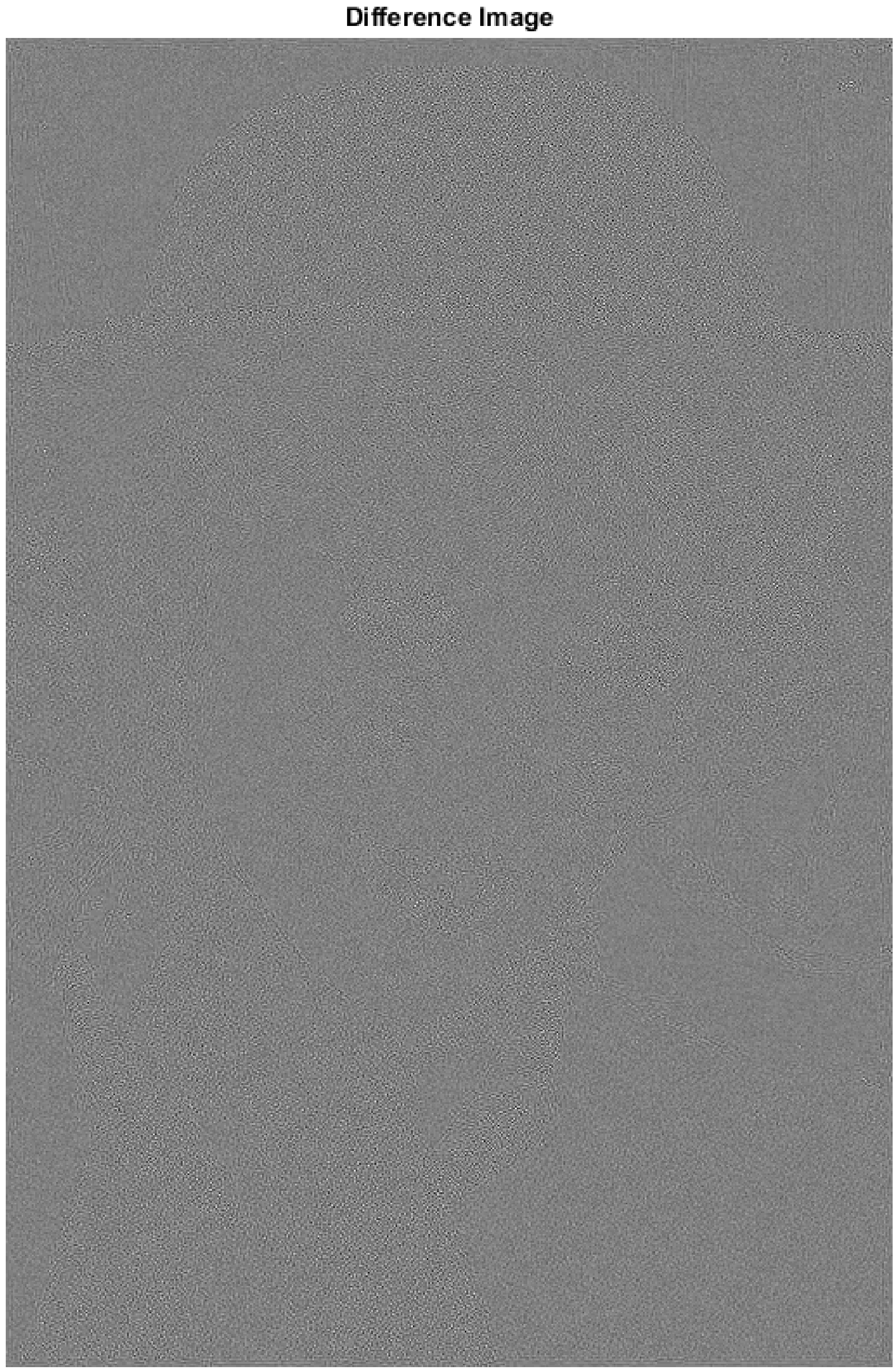

Figure 3: Difference Imagee for $\gamma = 0.25$

Figure 4: Restored Image for $\gamma = 1$

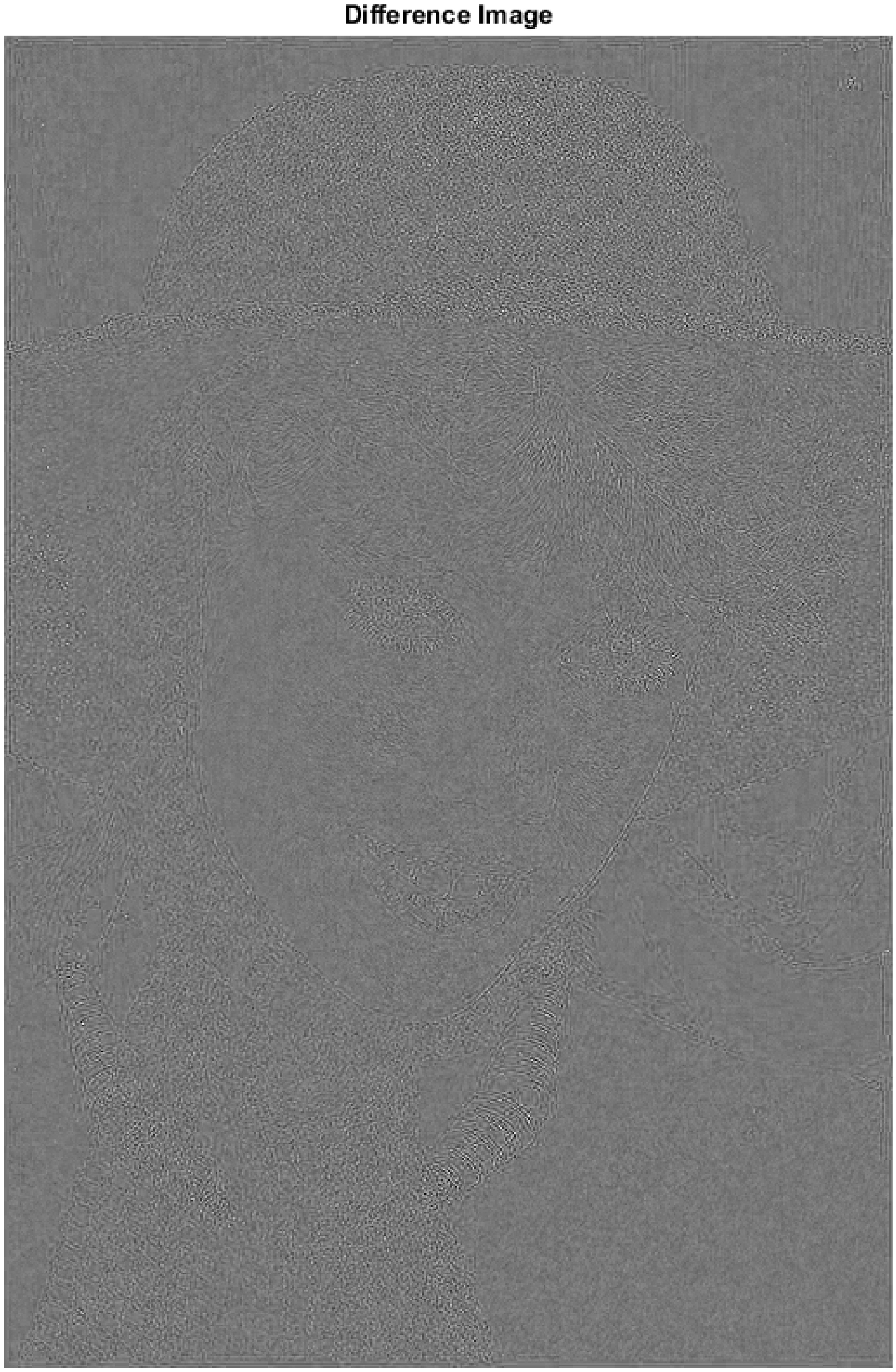

Figure 5: Difference Imagee for $\gamma = 1$

Restored Image gamma = 4.000000

Figure 6: Restored Image for $\gamma = 4$

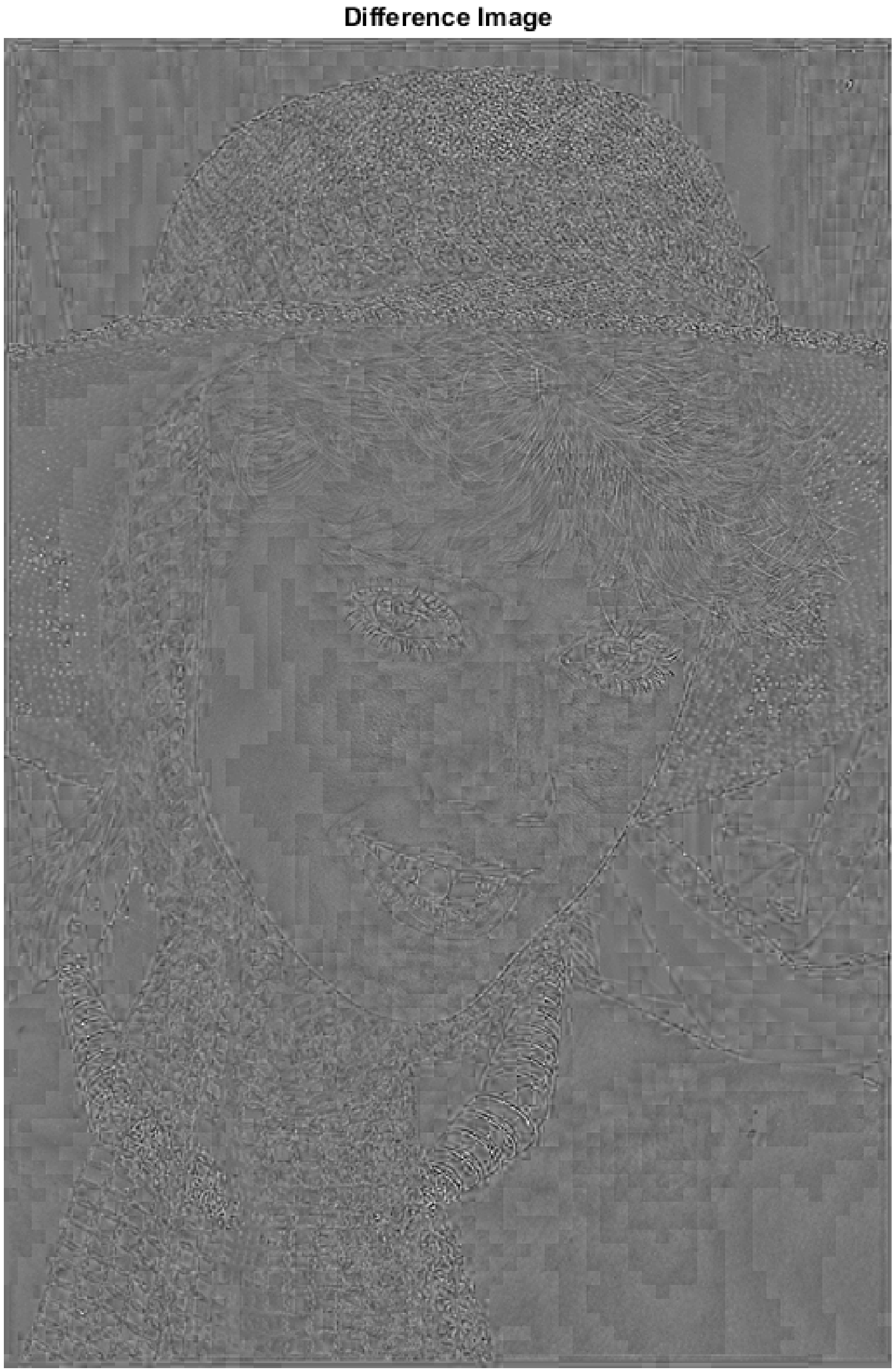

Figure 7: Difference Image for $\gamma = 4$

Comment on γ based on Results As the results show, as the value of γ increases, the quality of restored image deteriorates. This is since, the higher the value of γ, the higher would be the quantization error, since the most prominent DCT coefficients are mapped to a lower range of values and this causes loss of information. This can be seen in the 'blocky' effect observed in the restored image when $\gamma = 4$

Question 2. Differential Encoding and Zig-Zag Scan Pattern

MATLAB Code: Q2_Soln.m

```
%% Lab 9: Achromatic Baseline JPEG Encoding Lab
% Author; Aarti Ghatkesar
% Exercise 2.3: Observe Properties of DC and AC coefficients

%%

clc
clear all
close all

%% Reading DCT coefficients from 'img03y_1.dq' file obtained from previous
    section

run('jpeg\Qtables.m');
fileID = fopen('img03y_1.dq','r');
A= fread(fileID,'integer*2');
fclose(fileID);
rows = A(1);
cols = A(2);
quant_dct_coeff = reshape(A(3:end),cols,rows);
quant_dct_coeff = quant_dct_coeff';


%% Displaying DC coefficients
fcn = @(x)(x.data(1));
dct_blk = blockproc(quant_dct_coeff,[8,8],fcn);
dct_blk = dct_blk + 128;
imshow(dct_blk,[]);

%% Extracting AC coefficients

AC_coeff = zeros(rows*cols/64,63);
count =1;

for  i = 1:1:rows/8
    for j = 1:1:cols/8

        blk = quant_dct_coeff((i-1)*8+1:i*8,(j-1)*8+1:j*8);
        blk = blk(Zig);
```

```
            blk=blk(2:end);
            AC_coeff(count,:) = blk;
            count = count + 1;


        end
end

avg = mean(abs(AC_coeff));
figure
plot([2:64]',avg);
xlabel('AC coefficient index in Zig Zag order');
ylabel('Mean absolute value of AC coefficients across all blocks');
title('Mean absolute value of AC coefficients in zig zag order');
```

Figure 8: Image formed by DC coefficients. Note that this image been enlarged for demonstration purpose

Comments on Image formed by DC coefficient: The image formed by DC coefficients of the DCT transform resembles closely to the original image. This is expected since most of the energy of the image is concentrated in the DC coefficients, hence we can still figure out the image with just the DC coefficients. Notice that the image formed by DC coefficients appears blurry. This is since the edges that correspond to high frequency content of an image is lost, hence the image appears blurred and not sharp.

Why DC coefficients of adjacent blocks are correlated:

The reason for this is since the adjacent gray level of adjacent image blocks are likely to be similar. Since the image size is very very large compared to the 8 x 8 blocks formed, the image gray level does not change drastically from one block to another.

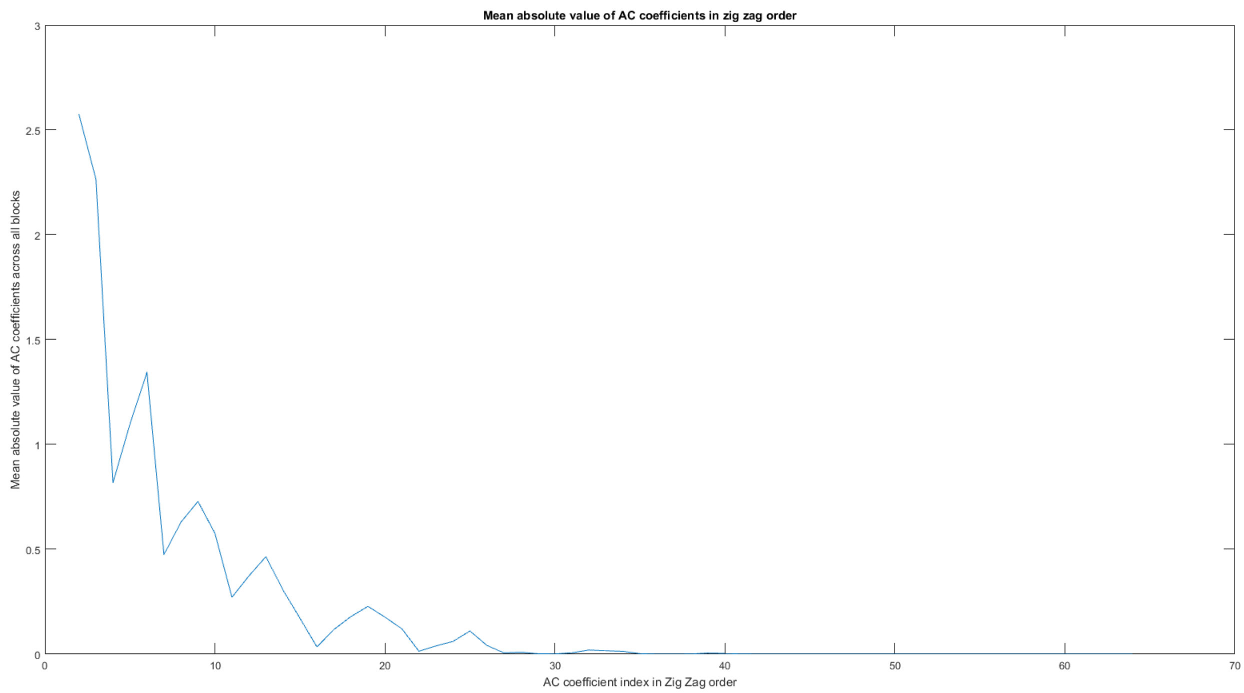

Figure 9: Plot of mean value of the magnitude of the AC coefficients for $\gamma = 1.0$. The AC coefficients have first been scanned in a zig zag fashion and the mean is then computed

Explanation of plot: As can be seen from the plot, the average DC coefficients value among blocks is highest when compared to all other AC coefficients.This is expected since most of the energy of the Image would be concentrated in the DC coefficient.

The remaining AC coefficients have been ordered according to a zig zag scan pattern. Arranging the AC coefficients in zig zag pattern has two reasons. One being that the energy in the coefficient (7,7) which corresponds to a highest frequency coefficent in both u and v plane is practically pretty close to zero and can be ignored. Secondly, human eye is less sensitive to high frequencies which can be obtained using visual MTF function. Hence reordering the Ac coefficients according to zig-zag scan pattern ensures that the Ac coefficients are arranged so that the high frequency components are towards the end of the array.

As mentioned earlier, the energy in these high frequency components is low, hence the average value of these coefficients among the blocks is low. This can be seen from the plot that as the frequency increases, the value of the corresponding coefficient reduces.

JPEG encoding makes use of this fact and codes the AC coefficients using Run length encoding.

C Code - BitSize, VLI_encode, ZigZag, DC_encode, AC_encode, Block_encode, Convert_encode, Zero_pad

jpegFunction.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include "Htables.h"

/****************************************/
int BitSize(int value)
{
        // Function to find the size 'm'
        int bitsize;
        bitsize = ceil(log2(abs(value )+ 1));
        return(bitsize);

}

/****************************************/
void VLI_encode(int bitsz, int value, char *block_code)
{
        // Function to perform Variable Length Integer Encoding

        printf("In VLI_Encode\n");
        int i;

        char *temp = malloc(sizeof(char)* (bitsz + 1));

        for (i = 0; i <= bitsz; i++)
        {
                temp[i] = '\0';
        }

        if (value < 0)
        {
                value = value - 1;
        }

```

```
        for (i = bitsz - 1; i >= 0; i--)
        {
                if (value & 1)
                        temp[i] = '1';
                else
                temp[i] = '0';

                value = value >> 1;
        }
        strcat(block_code, temp);
        free(temp);
}

/****************************************/

void ZigZag(int ** img, int y, int x, int *zigline)
{
        int i, j;

        for (i = 0; i < 8; i++)
        {
                for (j = 0; j < 8; j++)
                {
                        zigline[Zig[i][j]] = img[i + y][j + x];
                }
        }

}

/****************************************/
void DC_encode(int dc_value, int prev_value, char *block_code)
{
        int diff,bitsize;
        diff = dc_value - prev_value;
        bitsize = BitSize(diff);
    strcat(block_code, dcHuffman.code[bitsize]);
        VLI_encode(bitsize, diff, block_code);

}

/****************************************/
void AC_encode(int *zigzag, char *block_code)
{
        /*Init Variables*/

        int idx = 1; // starts from 1 as idx = 0  is DC
        int zerocnt = 0; // Counts runs of zeros
        int bitsize;

        while (idx < 64)
        {
                if (zigzag[idx] == 0)
                {
                        zerocnt++;
```

```
            }
            else
            {
                /* ZRL coding - enters this when encounters first non zero AC coeffient. So perfom Run length coding with obtained zerocount*/
                for (; zerocnt > 15; zerocnt=zerocnt - 16) // take 16 at a time to use ZRL for code
                {
                    strcat(block_code, acHuffman.code[15][0]); // Append code for ZRL : VLC ac for ZRL
                }

                bitsize = BitSize(zigzag[idx]); // Get size required for value
                strcat(block_code, acHuffman.code[zerocnt][bitsize]); // VLC ac for pair (run,size) for AC
                VLI_encode(bitsize, zigzag[idx], block_code);
                zerocnt = 0; // Resetting zero count for next run length
            }
            idx++;
        }
        // EOB coding -  End of block run length of zeros
        if (zerocnt)
        {
            strcat(block_code, acHuffman.code[0][0]);
        }
}

/****************************************/
void Block_encode(int prev_value, int *zigzag, char *block_code)
{
        DC_encode(zigzag[0], prev_value, block_code);
        AC_encode(zigzag, block_code);
        // NULL character taken care by strcat
}

/****************************************/
int Convert_encode(char *block_code, unsigned char *byte_code)
{

        // Converts Block_code to individual byte code
        // Takes care of byte stuffing
        // Final block_code contains less than 8 number of elements

        char tempArr[8] = { '\0' };
        int length=0;
        int len; // Holds number of bytes
        int i,j,k,temp,totLen;
        int value = 0;
        int rem = 0;

        totLen = strlen(block_code);
```

```
        len = strlen(block_code);
        rem = len % 8; // Remaining number of elements in block_code
        len = len / 8;

        for (i = 0; i <= 8 * len - 1; i = i + 8)
        {
                for (j = 0; j < 8; j++)
                {
                        // 0 in ASCII is 48 i.e  0011 0000
                        // 1 in ASCII is 49 i.e  0011 0001 Notice the last bit
                            only is different

                        temp = block_code[j + i] & 1;
                        value = pow(2, 7 - j) * temp + value;
                }

                byte_code[length] = value;
                length = length + 1;

                if (value == 0xFF)
                {
                        byte_code[length] = 0x00;
                        length = length + 1;
                }

                value = 0;
        }

        // Get remaining bits in block_code

        for (k = 0; k < 8; k++)
        {
                tempArr[k] = block_code[k + 8*len];
        }

        // Set all elements in block_code to NULL so that only the reamining
            elements can be copied
        memset(block_code, '\0', strlen(block_code));
        strcat(block_code, tempArr);

        return(length);

}

/****************************************/
unsigned char Zero_pad(char *block_code)
{

        // Zeropadding
        unsigned char byte_value;
        int length = 0;
        int i,value=0,temp;
        length = strlen(block_code);

```

```
        if (length > 8)
        {
                printf("Exiting From Zero Pad");
                exit(-1);
        }

        for(i = length; i < 8;i++)
        {
                block_code[i] = '0';
        }


        for (i = 0; i < 8; i++)
        {

                temp = block_code[i] & 1;
                value = pow(2, 7 - i) * temp + value;
        }

        byte_value = value;

}
```

JPEG_encode.c

```
/******************************************************/
/* JPEG_encoder     By Jinwha Yang and Charles Bouman  */
/* Apr. 2000.       Built for EE637 Lab.               */
/* All right reserved for Prof. Bouman                 */
/******************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "Htables.h"
#include "JPEGdefs.h"
#include "allocate.h"



int main(int argc, char* argv[])
{
  int   **input_img;  /* Input set of DCT coefficients read from matlab file */
  FILE  *outfp;       /* File pointer to output JPEG image */
  int     row;        /* height of image */
  int     column;     /* width of image */
  double gamma;       /* scaling factor for quantizer */
  int bitsize;
  int i = 0;

  /* Use command line arguements to read matlab file, and return */
  /* values of height, width, quantizer scaling and file pointer */
  /* to output JPEG file.                                        */
```

```
  input_img = get_arguments(argc,argv,&row,&column,&gamma,&outfp) ;

  /* scale global variable for quantization matrix */
  if( gamma > 0 )
    change_qtable(gamma) ;
  else {
    fprintf(stderr, "\nQuantizer scaling must be > 0.\n") ;
    exit(-1) ;
  }



  /* Encode quantized DCT coefficients into JPEG image */
  jpeg_encode(input_img,row,column,outfp) ;

}


void change_qtable(double scale)
{
  int     i,j ;
  double  val ;

  for(i=0;i<8;i++){
    for(j=0;j<8;j++){
      val = Quant[i][j]*scale ;
      /* w.r.t spec, Quant entry can be bigger than 16 bit */
      Quant[i][j] = (val>65535) ? 65535 : (int)(val+0.5) ;
    }
  }
}


int **get_arguments(int argc,
                    char *argv[],
                    int *row,
                    int *col,
                    double *gamma,
                    FILE **fp )
{
  FILE *    inp ;
  short**   img ;
  int   **  in_img ;
  short     tmp ;
  int       i,j ;


  /* needs at least 2 argument */
  switch(argc){
  case 0:
  case 1:
  case 2:
  case 3: usage(); exit(-1) ; break ;
  default:
```

```

    /* read Quant scale */
    sscanf(argv[1],"%lf",gamma) ;

    /* prepare output file */
    *fp = fopen(argv[3],"wb") ;
    if(*fp==NULL) {
      fprintf(stderr,
            "\n%s file error\n",argv[3]) ;
      exit(-1) ;
    }

    /* read input file */
    inp = fopen(argv[2],"rb") ;
    if( inp == NULL ) {
      fprintf(stderr,
              "\n%s open error\n",argv[2]) ;
      exit(-1) ;
    }
    /* input file has 2 16 bit(short) row, column info */
    /* valid 2-D array follows */
    fread(&tmp,sizeof(short),1,inp) ;
    *row = (int) tmp ;
    fread(&tmp,sizeof(short),1,inp) ;
    *col = (int) tmp ;

    img = (short **)get_img(*col,*row,sizeof(short)) ;
    fread(img[0],sizeof(short),*col**row,inp) ;
    fclose(inp) ;

    break ;
  }

  in_img = (int **)get_img(*col,*row,sizeof(int)) ;
  for( i=0 ; i<*row; i++ ){
    for( j=0 ; j<*col; j++ ){
      in_img[i][j] = (int) img[i][j] ;
    }
  }
  free_img((void**)img) ;
  return( in_img ) ;
}



void jpeg_encode(int **img, int h, int w, FILE *jpgp)
{
  int     x, y, length ;
  int     prev_dc = 0 ;
  unsigned char val ;
  static int     zigline[64] ;
  static char    block_code[8192] = {'\0'} ;
  static unsigned char byte_code[1024] ;

```

```
  printf("\n JPEG encode starts...") ;

  /* JPEG header writes */
  put_header(w,h,Quant,jpgp) ;

  printf("\n Header written...\n Image size %d row  %d column\n",h,w) ;
  /* Normal block processing */
  for( y = 0 ; y < h ; y += 8) {
    for( x = 0 ; x < w ; x += 8 ){
      /* read up 8x8 block */
      ZigZag(img,y,x,zigline) ;
      Block_encode(prev_dc,zigline,block_code) ;
      prev_dc = zigline[0] ;
      length = Convert_encode(block_code,byte_code) ;
      fwrite(byte_code,sizeof(char),length,jpgp) ;
    }
    printf("\r (%d)th row processing    ",y) ;
  }
  printf("\nEncode done.\n") ;
  /* Zero padding */
  if( strlen(block_code) ){
    val = Zero_pad(block_code) ;
    fwrite(&val,sizeof(char),1,jpgp) ;
  }

  /* EOI */
  put_tail(jpgp) ;
  fclose(jpgp) ;
  free_img((void **)img) ;
}


void usage(void)
{
  fprintf(stderr,"\nJPEG_encode <Quant scale> <in_file> <out_file>");
  fprintf(stderr,"\n<Quant scale> - gamma value in eq (1)");
  fprintf(stderr,"\n<in_file> - output file using section 2.1");
  fprintf(stderr,"\n<out_file> - JPEG output file");
}
```

Figure 10: JPEG Encoded Image when $\gamma = 0.25$

Figure 11: JPEG Encoded Image when $\gamma = 1$

Figure 12: JPEG Encoded Image when $\gamma = 4$