ECE 637 Digital Image Processing Laboratory: Achromatic Baseline JPEG Encoding Lab

Yang Wang

April 25, 2016

1 Introduction

Nothing due for report.

2 DCT Block Transforms and Quantization

In this section, discrete cosine transform (DCT) is introduced. Quantizations using DCT coefficients are also explored in this section.

2.1 Exercises

2.1.1 MATLAB Code for Storing the File img03y.dq

```
function writeDq(file, img, gamma)
  run('Qtables.m');
  img = double(img) - 128;
  dct_blk = blockproc(img, [8 8], ...
    @(x) round(dct2(x.data, [8 8])./(Quant*gamma)));

  f = fopen(file, 'w');
  fwrite(f, size(dct_blk, 1), 'integer*2');
  fwrite(f, size(dct_blk, 2), 'integer*2');
  fwrite(f, dct_blk', 'integer*2');
end
```

2.1.2 MATLAB Code for Reading the File img03y.dq

```
function [img] = readDq(file, gamma)
  run('Qtables.m');
  f = fopen(file, 'r');
  data = fread(f, 'integer*2');
  img = reshape(data(3:end), [data(2) data(1)])';
  img = blockproc(img, [8 8], ...
    @(x) round(idct2(x.data.*Quant*gamma, [8 8])));
  img = img + 128;
  img = uint8(img);
end
```

2.1.3 Orginal, Restored and Difference Images for γ=0.25, 1, and 4

When γ=4, the image has the biggest distortion since the difference image is the most noisy out of three gamma values.

(a) Original img03y.tif

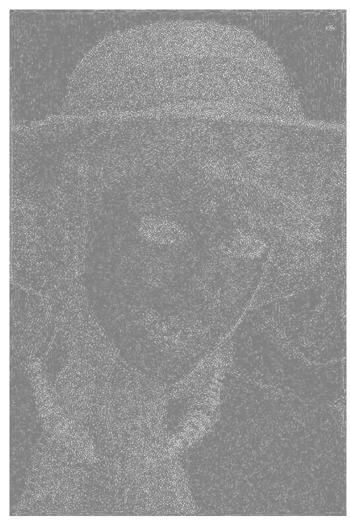
(b) Restored Image

(c) Diff Image

Figure 1: Original vs. Restored vs. Difference Images, γ=0.25

(a) Original img03y.tif

(b) Restored Image



(c) Diff Image

Figure 2: Original vs. Restored vs. Difference Images, γ=1

(a) Original img03y.tif (b) Restored Image (c) Diff Image

Figure 3: Original vs. Restored vs. Difference Images, γ=4

2.2 Differential Encoding and the Zig-Zag Scan Pattern

Nothing due for report.

2.3 Exercises

2.3.1 Image Formed By the DC Coefficients

The image below is formed by the DC components of the original image; it looks similar to the original image but the resolution is lowered since it is an window-average of the original image.

Figure 4: DC Coefficients Image

2.3.2 Explanation

The DC coefficients of adjacent blocks are correlated because the average intensity of neighboring pixels is usually close in terms of values for a natural image.

2.3.3 Plot of Mean Value of AC Coefficients for γ=1.0

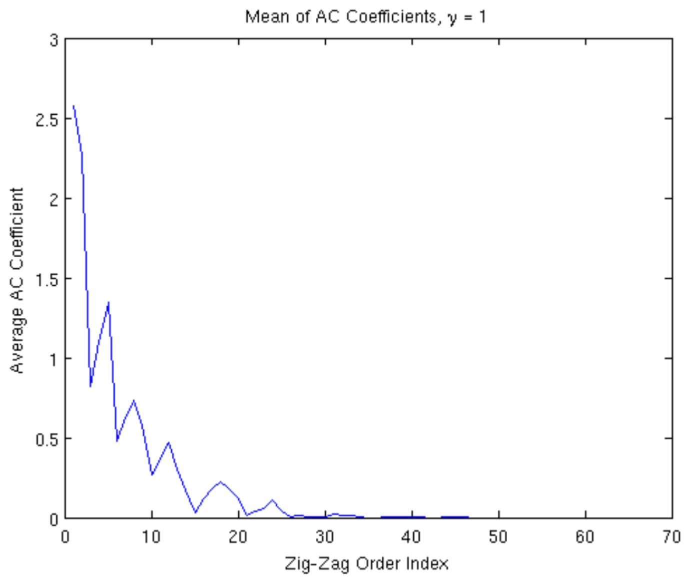


Figure 5: Average DCT AC Value

3 Entropy Encoding of Coefficients

3.1 C Code for Subroutines

3.1.1 JPEGdef.c

```
#include "JPEGdefs.h"
#include "Htables.h"

int BitSize(int value) {
  fprintf(stdout, "in BitSize()\n");
  int bitsize = 0;

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

  while (value > 0) {
    bitsize++;
    value >>= 1;
  }

  printf("bitsize = %d\n", bitsize);
  return bitsize;
```

```
}

void VLI_encode(int bitsz, int value, char *block_code) {
  fprintf(stdout, "in VLI_encode()\n");
  int i;
  char VLI[13] = { 0 };

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

  for (i = bitsz - 1; i >= 0; i--) {
    VLI[i] = (value & 1) ? '1' : '0';
    value >>= 1;
  }

  strcat(block_code, VLI);
}

void ZigZag(int ** img, int y, int x, int *zigline) {
  fprintf(stdout, "in ZigZag()\n");
  int i, j;

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

void DC_encode(int dc_value, int prev_value, char *block_code) {
  fprintf(stdout, "in DC_encode()\n");
  int diff, size;

  diff = dc_value - prev_value;
  printf("diff = %d\n", diff);
  size = BitSize(diff);

  strcat(block_code, dcHuffman.code[size]);
  VLI_encode(size, diff, block_code);
}

void AC_encode(int *zigzag, char *block_code) {
  fprintf(stdout, "in AC_encode()\n");
  int idx = 1;
  int zerocnt = 0;
  int bitsize;

  while (idx < 64) {
    if (zigzag[idx] == 0) {
      zerocnt++;
    } else {
      for (; zerocnt > 15; zerocnt -= 16) {
        strcat(block_code, acHuffman.code[15][0]);
```

```
      }

      bitsize = BitSize(zigzag[idx]);
      strcat(block_code, acHuffman.code[zerocnt][bitsize]);
      VLI_encode(bitsize, zigzag[idx], block_code);

      zerocnt = 0;
    }

    idx++;
  }

  if (zerocnt) {
    strcat(block_code, acHuffman.code[0][0]);
  }
}

void Block_encode(int prev_value, int *zigzag, char *block_code) {
  fprintf(stdout, "in Block_encode()\n");
  DC_encode(zigzag[0], prev_value, block_code);
  AC_encode(zigzag, block_code);
}

int Convert_encode(char *block_code, unsigned char *byte_code) {
  fprintf(stdout, "in Convert_encode()\n");
  int len = strlen(block_code);
  int bytes = len / 8;
  int idx;
  int i, j;

  idx = 0;
  for (i = 0; i < bytes; i++) {
    for (j = 0; j < 8; j++) {
      byte_code[idx] <<= 1;

      if (block_code[i*8 + j] == '1') {
        byte_code[idx]++;
      }
    }

    if (byte_code[idx] == 0xff) {
      byte_code[++idx] == 0x00;
      bytes++;
    }

    idx++;
  }

  strcpy(block_code, block_code + len / 8 * 8);

  return bytes;
}

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

```
  fprintf(stdout, "Zero_pad()\n");
  unsigned char val;
  int len;
  int i;

  len = strlen(block_code);
  for (i = 0; i < len; i++) {
    val <<= 1;

    if (block_code[i] == '1') {
      val++;
    }
  }

  val <<= (8 - len);

  return val;
}
```

3.2 C Code for Main Program

3.2.1 JPEGencode.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 */

  /* 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) ;

  return 1 ;
}


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 */
```

```c
  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\n");
}
```

3.3 Email the Encoded Image

The encoded image was submitted.

3.4 Three Plots from xv

(a) Encode Image γ=0.25

(b) Encoded Image γ=1

(c) Encoded Image γ=4

Figure 6: Encoded JPEG Images for Different γ Values