KWANGWOON UNIVERSITY

2D Block-based DCT

2018.10.9

Seoungjun Oh(sjoh@kw.ac.kr)
Wooju Lee (krosea@kw.ac.kr)

Multimedia LAB
VIA-Multimedia Center, Kwangwoon University

Contents

- Discrete Cosine Transform
  - 2-D block-based DCT
- Example
- Assignment

Discrete Cosine Transform (DCT)

❖ Discrete Cosine Transform (DCT)

- Transform a signal into a different representation
  - 1-D DCT : time domain → frequency domain
  - 2-D DCT : spatial domain → frequency domain

- 1-D DCT

$$\text{Forward DCT}: X[k] = \sqrt{\frac{2}{N}}\beta[k]\sum_{n=0}^{N-1} x[n]\cos\left(\frac{(2n+1)\pi k}{2N}\right), \quad 0 \le k \le N-1$$

$$\text{Inverse DCT}: x[n] = \sqrt{\frac{2}{N}}\sum_{k=0}^{N-1} \beta[k]X[k]\cos\left(\frac{(2n+1)\pi k}{2N}\right), \quad 0 \le n \le N-1$$

$$\text{where } \beta[k] = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & \text{otherwise} \end{cases}$$

Discrete Cosine Transform (DCT)

❖ Discrete Cosine Transform (DCT)

- 2-D DCT

$$\text{Forward DCT}: F_{x,y} = \frac{c(x)c(y)}{N/2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_{i,j}\cos\left(\frac{(2i+1)\pi x}{2N}\right)\cos\left(\frac{(2j+1)\pi y}{2N}\right), \quad 0 \le x, y \le N-1$$

$$\text{Inverse DCT}: f_{i,j} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\frac{c(x)c(y)}{N/2} F_{x,y}\cos\left(\frac{(2i+1)\pi x}{2N}\right)\cos\left(\frac{(2j+1)\pi y}{2N}\right), \quad 0 \le i, j \le N-1$$

$$\text{where } c(l) = \begin{cases} \frac{1}{\sqrt{2}}, & l = 0 \\ 1, & \text{otherwise} \end{cases}$$

< Original image >

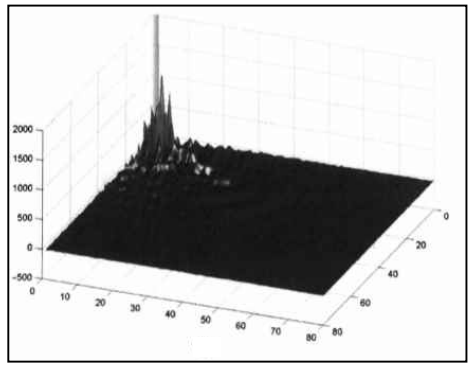
< DCT coefficient >

Discrete Cosine Transform (DCT)

KWANGWOON UNIVERSITY

- 2-D DCT in digital image processing
  - Energy compactness
    - Transform image data to be easily compressed

< 80×80 pixel image >



< Coefficients of 2-D DCT >

  - Independent basis
  - Separability
  - Fast algorithms

VIAME

Discrete Cosine Transform (DCT)

KWANGWOON UNIVERSITY

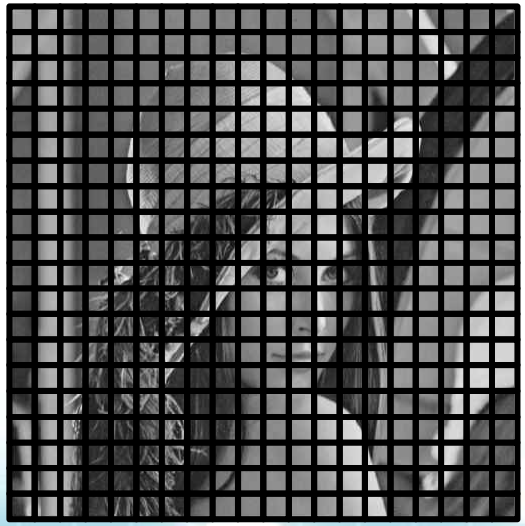
❖ Consideration on the block size of DCT

- Increase of block size of DCT
  - Better energy compaction and decorrelation performance
  - Increase of computational complexity

  → Block-based DCT & Separability

❖ Block-based DCT

< Original image >



< Original image with block partitioning >

< Block-based DCT coefficient >

VIAME

Block-based DCT

- Block-based DCT is commonly used (JPEG, MPEG)
  - Compromise between compression efficiency and computational efficiency

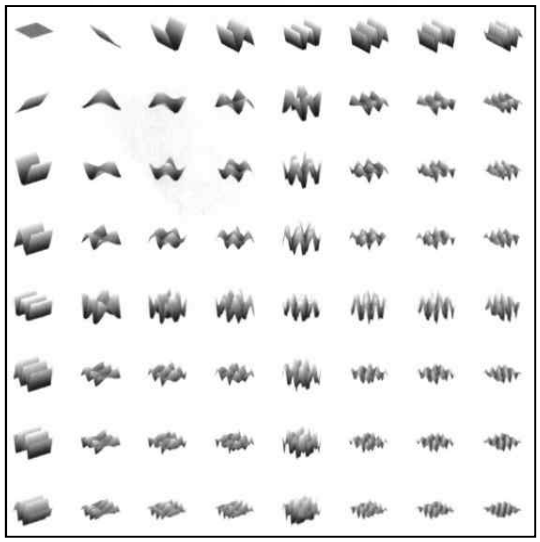
$$\text{Forward 8x8 DCT}: F_{x,y} = \frac{c(x)c(y)}{4}\sum_{i=0}^{7}\sum_{j=0}^{7} f_{i,j}\cos\left(\frac{(2i+1)\pi x}{16}\right)\cos\left(\frac{(2j+1)\pi y}{16}\right), \quad 0 \le x, y \le 7$$

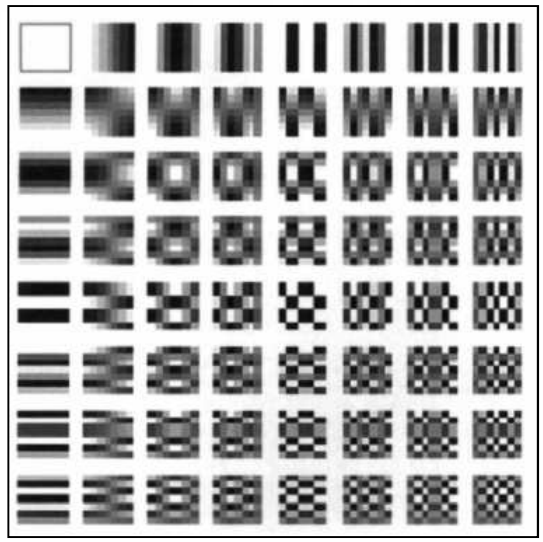
$$\text{Inverse 8x8 DCT}: f_{i,j} = \sum_{x=0}^{7}\sum_{y=0}^{7} \frac{c(x)c(y)}{4} F_{x,y}\cos\left(\frac{(2i+1)\pi x}{16}\right)\cos\left(\frac{(2j+1)\pi y}{16}\right), \quad 0 \le i, j \le 7$$

$$\text{where } c(l) = \begin{cases} \frac{1}{\sqrt{2}}, & l = 0 \\ 1, & \text{otherwise} \end{cases}$$

Block-based DCT

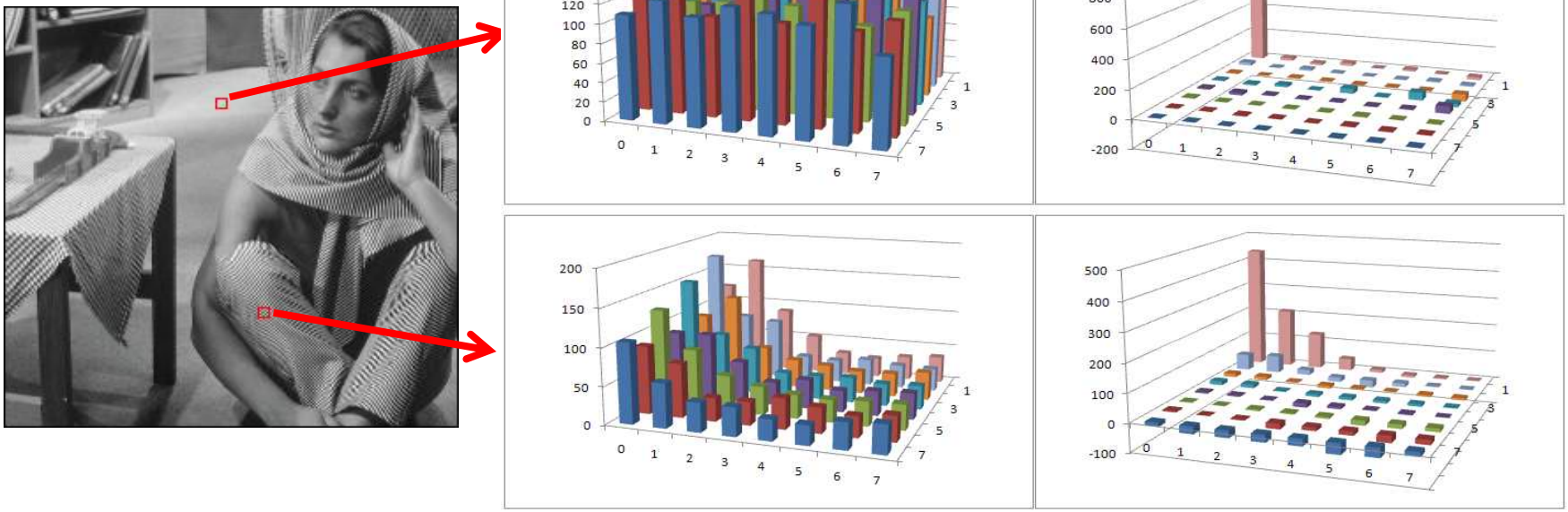
❖ 8x8 DCT basis functions





- Top-left : the lowest frequency
- Moving to right : increase in horizontal frequency
- Moving down : increase in vertical frequency
- Moving to right and down : increase in both horizontal and vertical frequency

Block-based DCT



< Original block >

< DCT coefficient >

Discrete Cosine Transform

- Forward DCT
  - 2-D 8x8 DCT transform

$$F_{x,y} = \frac{c(x)c(y)}{N/2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_{i,j}\cos\left(\frac{(2i+1)\pi x}{2N}\right)\cos\left(\frac{(2j+1)\pi y}{2N}\right)$$

$$c(l) = \begin{cases} 1/\sqrt{2} & \text{for } l = 0 \\ 1 & \text{otherwise} \end{cases}$$

46	42	32	27	27	29	98	168
44	41	32	33	29	37	140	174
37	32	29	29	31	55	162	180
38	37	34	31	33	88	173	177
34	32	28	34	47	129	164	166
32	34	31	35	81	146	151	158
46	44	59	70	105	141	137	147
83	76	94	100	112	112	123	139

< input image >

384.5	23.2	12.9	11.5	8.3	-5.0	1.7	-1.1
-30.3	32.9	28.2	20.0	4.6	24.0	1.6	-1.9
40.5	29.8	26.1	6.6	10.7	-12.5	-2.8	4.3
3.3	39.0	5.5	12.0	-1.0	9.0	2.3	-0.6
15.3	-8.2	16.9	8.5	5.0	-2.4	-3.5	3.0
-6.3	15.7	-1.8	-4.7	-2.0	4.0	2.5	-2.0
5.1	-9.5	-1.8	4.0	0.4	-3.2	-1.1	1.1
-1.7	4.0	4.7	-3.5	-3.8	5.4	-1.2	-1.4

< DCT coefficients >

Discrete Cosine Transform

❖ Inverse DCT

- 2-D 8x8 Inverse DCT transform with 2x2 low frequency

$$f_{i,j} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} \frac{c(x)c(y)}{N/2} F_{x,y} \cos\left(\frac{(2i+1)\pi x}{2N}\right) \cos\left(\frac{(2j+1)\pi y}{2N}\right)$$

$$c(l) = \begin{cases} 1/\sqrt{2} & \text{for } l = 0 \\ 1 & \text{otherwise} \end{cases}$$

54	52	49	45	40	36	32	30
54	52	49	45	41	37	34	32
53	52	49	46	43	40	37	36
52	51	50	48	45	43	42	41
51	51	50	49	48	47	47	46
50	50	50	50	51	51	51	51
49	50	50	51	53	54	54	55
49	50	51	52	54	55	56	57

< input image >

384.5	23.2	0	0	0	0	0	0
-30.3	32.9	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

< DCT coefficients >

Discrete Cosine Transform

❖ Inverse DCT

- 2-D 8x8 Inverse DCT transform with 6x6 high frequency

$$f_{i,j} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} \frac{c(x)c(y)}{N/2} F_{x,y} \cos\left(\frac{(2i+1)\pi x}{2N}\right) \cos\left(\frac{(2j+1)\pi y}{2N}\right)$$

$$c(l) = \begin{cases} 1/\sqrt{2} & \text{for } l = 0 \\ 1 & \text{otherwise} \end{cases}$$

14	1	0	0	0	0	3	6
0	1	0	0	3	0	0	2
0	0	8	7	0	0	0	0
0	0	5	6	1	1	0	0
1	0	0	2	0	2	6	0
0	0	1	4	3	6	0	0
0	0	3	2	0	0	0	1
7	6	0	0	4	0	0	10

< input image >

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	26.1	6.6	10.7	-12.5	-2.8	4.3
0	0	5.5	12.0	-1.0	9.0	2.3	-0.6
0	0	16.9	8.5	5.0	-2.4	-3.5	3.0
0	0	-1.8	-4.7	-2.0	4.0	2.5	-2.0
0	0	-1.8	4.0	0.4	-3.2	-1.1	1.1
0	0	4.7	-3.5	-3.8	5.4	-1.2	-1.4

< DCT coefficients >

Block-based DCT

KWANGWOON UNIVERSITY

❖ Separability

- 2-D DCT can be conducted by performing two 1-D DCTs
- Simplifies the hardware requirements at the expense of a slight increase in the overall operations count

< Horizontal 1-D DCT >

< Vertical 1-D DCT >

< 2-D DCT >

VIAME

Block-based DCT

- Separability with 8x8 DCT
  - 2-D DCT can be conducted by performing two 1-D DCTs
  - Simplifies the hardware requirements at the expense of a slight increase in the overall operations count

$$y_{kl} = \frac{c(k)}{2}\sum_{i=0}^{7}\left[\frac{c(l)}{2}\sum_{j=0}^{7} x_{ij}\cos\left(\frac{(2j+1)l\pi}{16}\right)\right]\cos\left(\frac{(2i+1)k\pi}{16}\right), \quad 0 \le k,l \le 7$$

$$z_{il} = \frac{c(l)}{2}\sum_{j=0}^{7} x_{ij}\cos\left(\frac{(2j+1)l\pi}{16}\right), \quad 0 \le i,l \le 7$$

$$\mathbf{Z} = \mathbf{TX}$$

$$\mathbf{Y} = (\mathbf{TZ}^T)^T = \mathbf{TXT}^T$$

where
- $\mathbf{T}$: 8x8 DCT transform matrix
- $\mathbf{X}$: input matrix
- $\mathbf{Y}$: coefficient matrix

$$A\begin{bmatrix}1&2&3\\4&5&6\\7&8&9\end{bmatrix} \quad A^T\begin{bmatrix}1&4&7\\2&5&8\\3&6&9\end{bmatrix}$$

$$A\begin{bmatrix}1&4&3\\8&2&6\\7&8&3\\4&9&6\\7&8&1\end{bmatrix} \quad A^T\begin{bmatrix}1&8&7&4&7\\4&2&8&9&8\\3&6&3&6&1\end{bmatrix}$$

Example

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4
5  #define WIDTH 512
6  #define HEIGHT 512
7  #define DCT_BlockSize   8                                          // Definition for DCT macro block size
8  #define PI 3.14159265358979323846
9  typedef unsigned char BYTE;
10
11 unsigned char** MemAlloc_2D(int width, int height);                                  // 2D memory allocation
12 void MemFree_2D(unsigned char** arr, int height);                                   // 2D memory free
13 void FileRead(char* filename, unsigned char** img_in, int width, int height);       //  read data from a file
14 void FileWrite(char* filename, unsigned char** img_out, int width, int height);     //  write data to a file
15 float GetPSNR(unsigned char** img_ori, unsigned char** img_dist, int width, int height);  //  PSNR calculation
16
17 void FDCT(BYTE** img_in, double** img_coeffi, int blockSize, int height, int width);
18 void IDCT(double** img_coeffi, BYTE** img_recon, int blockSize, int lowFreqBlock, int highFreqBlock, int height, int width);
19 // lowFreqBlockSize : coefficient block size to remain in low frequency
20 // highFreqBlockSize : coefficient block size to remain in high frequency
21 int main()
22 {
23     BYTE** img_in, **img_recon;
24     double** img_coeffi;
25
26     int i,j;
27     double temp;        // Variables for operations
28     BYTE data;
29
30     FILE* fp_fdct_out = fopen("[8x8_DCT]Lena(512x512).raw", "wb");      // File stream to write DCT coefficients
31
32     img_in = MemAlloc_2D(WIDTH,HEIGHT);
33     img_recon = MemAlloc_2D(WIDTH,HEIGHT);
34
35     img_coeffi = (double**)malloc(sizeof(double*)*HEIGHT);
36     for(i=0 ; i<HEIGHT ; i++){
37         img_coeffi[i] = (double*)malloc(sizeof(double)*WIDTH);
38     }
```

Example

```
40      FileRead("Lena(512x512).raw", img_in, WIDTH, HEIGHT);
41      FDCT(img_in, img_coeffi, DCT_BlockSize, HEIGHT, WIDTH);     // Forward DCT
42
43      for(i = 0 ; i < HEIGHT ; i++){      // DCT data save
44          for(j = 0 ; j < WIDTH ; j++){
45              temp = img_coeffi[i][j];
46              if(temp < 0)        // Clipping
47                  temp = 0;
48              else if(temp > 255)
49                  temp = 255;
50              data = (BYTE)floor(temp + 0.5);
51              fwrite(&data, 1, 1, fp_fdct_out);
52          }
53      }
54
55      IDCT(img_coeffi, img_recon, DCT_BlockSize, DCT_BlockSize, 0, HEIGHT, WIDTH);            // Inverse DCT by all coefficients
56      FileWrite("[8x8_IDCT]Lena(512x512).raw",img_recon, WIDTH, HEIGHT);
57      printf("PSNR (Reconstruction by all coefficients) : %fdB₩n₩n", GetPSNR(img_in, img_recon,WIDTH,HEIGHT));        // Print the PSNR value
58
59      IDCT(img_coeffi, img_recon, DCT_BlockSize, 6, 0, HEIGHT, WIDTH);                        // Inverse DCT by low6x6 coefficients
60      FileWrite("[8x8_IDCT_Low6x6]Lena(512x512).raw",img_recon, WIDTH, HEIGHT);
61      printf("PSNR (Reconstruction by low6x6 coefficients) : %fdB₩n₩n", GetPSNR(img_in, img_recon,WIDTH,HEIGHT));
62
63      IDCT(img_coeffi, img_recon, DCT_BlockSize, 2, 0, HEIGHT, WIDTH);                        // Inverse DCT by low2x2 coefficients
64      FileWrite("[8x8_IDCT_Low2x2]Lena(512x512).raw",img_recon, WIDTH, HEIGHT);
65      printf("PSNR (Reconstruction by low2x2 coefficients) : %fdB₩n₩n", GetPSNR(img_in, img_recon,WIDTH,HEIGHT));
66
67      IDCT(img_coeffi, img_recon, DCT_BlockSize, 0, 6, HEIGHT, WIDTH);                        // Inverse DCT by high6x6 coefficients
68      FileWrite("[8x8_IDCT_High6x6]Lena(512x512).raw",img_recon, WIDTH, HEIGHT);
69      printf("PSNR (Reconstruction by high6x6 coefficients) : %fdB₩n₩n", GetPSNR(img_in, img_recon,WIDTH,HEIGHT));
70
71      MemFree_2D(img_in, HEIGHT);
72      MemFree_2D(img_recon, HEIGHT);
73
74      for(i=0 ; i<HEIGHT ; i++){
75          free(img_coeffi[i]);
76      }
77      free(img_coeffi);
78
79      fclose(fp_fdct_out);
80
81      return 0;
82  }
```

```
PSNR (Reconstruction by all coefficients) : 49.843594dB

PSNR (Reconstruction by low6x6 coefficients) : 40.266804dB

PSNR (Reconstruction by low2x2 coefficients) : 28.182249dB

PSNR (Reconstruction by high6x6 coefficients) : 5.718628dB
계속하려면 아무 키나 누르십시오 . . .
```

Assignment 1

KWANGWOON UNIVERSITY

❖ Forward DCT

$$F_{x,y} = \frac{c(x)c(y)}{N/2} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f_{i,j} \cos\left(\frac{(2i+1)\pi x}{2N}\right) \cos\left(\frac{(2j+1)\pi y}{2N}\right)$$

```
void FDCT(BYTE** img_in, double** img_coeffi, int blockSize, int height, int width)     // Operating forward DCT
{
    int x,y,u,v,i,j;
    double coeffi, cn, cm;
    for(x = 0 ; x < height ; x += blockSize){       // (x,y) : left top position of current block on operation
        for(y = 0 ; y < width ; y += blockSize){

            for(u = 0 ; u < blockSize ; u++){       // (u,v) : coefficients coordinates
                for(v = 0 ; v < blockSize ; v++){

                    coeffi = 0;
                    for(i = 0 ; i < blockSize ; i++){       // (i,j) : image data coordinates
                        for(j = 0 ; j < blockSize ; j++){
                            cn = u == 0 ? 1 / sqrt(2.0) : 1;        // FDCT operation
                            cm = v == 0 ? 1 / sqrt(2.0) : 1;
                            coeffi += cn * cm * (2 / (double)blockSize) * (double)img_in[x + i][y + j] * cos(((2*i+1) * u*PI)
                                / (double)(2*blockSize)) * cos(((2*j+1) * v*PI) / (double)(2*blockSize));
                        }
                    }
                    img_coeffi[x + u][y + v] = coeffi;      // coefficient save
                }
            }

        }
    }
}
```

Assignment 1

KWANGWOON UNIVERSITY

❖ Inverse DCT

$$f_{i,j} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} \frac{c(x)c(y)}{N/2} F_{x,y} \cos\left(\frac{(2i+1)\pi x}{2N}\right) \cos\left(\frac{(2j+1)\pi y}{2N}\right)$$

```
void IDCT(double** img_coeffi, BYTE** img_recon, int blockSize, int lowFreqBlockSize, int highFreqBlockSize, int height, int width)
{



}
```

?

VIAME

Assignment 1

❖ Result

Original image

8x8 FDCT coefficient image

8x8 IDCT image

Assignment 1

❖ Result

IDCT image using low 2x2 coefficient
PSNR : 28.2dB

IDCT image using low 6x6 coefficient
PSNR : 40.3dB

IDCT image using high 6x6 coefficient
PSNR : 5.72dB

Assignment 2

KWANGWOON UNIVERSITY

```c
#include <stdio.h>      // Header file
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

typedef unsigned char BYTE;

#define WIDTH       512     // image size
#define HEIGHT      512
#define BLOCK_SIZE  8       // DCT block size

// memory management
unsigned char** MemAlloc_UC_2D(const int width, const int height);  // 2D memory allocation for unsigned char
double** MemAlloc_D_2D(const int width, const int height);          // 2-D memory allocation for double type
void MemFree_UC_2D(unsigned char** arr, const int height);          // 2-D memory free for unsigned char type
void MemFree_D_2D(double** arr, const int height);                  // 2-D memory free for double type

// image file management
void ImgRead(const char* filename, unsigned char** img_in, const int width, const int height);          // image file read
void ImgWrite(const char* filename, const unsigned char** img_out, const int width, const int height);  // image file write

// Discrete Cosine Transform (DCT)
// separable block-based forward DCT
void SeparableBlockFDCT_2D(const double** input, double** coeff, const int width, const int height, const int block_size);
// separable block-based inverse DCT
void SeparableBlockIDCT_2D(const double** coeff, double** output, const int width, const int height, const int block_size);

void FDCT_1D(const double* input, double* coeff, const int N);          // N-point 1-D forward DCT
void IDCT_1D(const double* coeff, double* output, const int N);         // N-point 1-D inverse DCT

// matrix operation
void MatTranspose(double **mat, const int size);        // matrix transpose

// block-based 2-D forward DCT
void BlockFDCT_2D(const double** input, double** coeff, const int width, const int height, const int block_size);
// block-based 2-D inverse DCT
void BlockIDCT_2D(const double** coeff, double** output, const int width, const int height, const int block_size);
```

VIAME

Assignment 2

```
void main()
{
    int i,j,cnt;
    clock_t start;

    // memory allocation
    BYTE **img_in = MemAlloc_UC_2D(WIDTH, HEIGHT);
    BYTE **img_out_sep = MemAlloc_UC_2D(WIDTH, HEIGHT);
    BYTE **img_out_2D = MemAlloc_UC_2D(WIDTH, HEIGHT);
    double **input = MemAlloc_D_2D(WIDTH, HEIGHT);
    double **output = MemAlloc_D_2D(WIDTH, HEIGHT);
    double **coeff_sep = MemAlloc_D_2D(WIDTH, HEIGHT);
    double **coeff_2D = MemAlloc_D_2D(WIDTH, HEIGHT);

    // image read
    ImgRead("Lena(512x512).raw", img_in, WIDTH, HEIGHT);
    // type casting
    for(i = 0 ; i < HEIGHT ; i++){
        for(j = 0 ; j < WIDTH ; j++){
            input[i][j] = (double)img_in[i][j];
        }
    }
```

Assignment 2

```
// separable block-based forward DCT
start = clock();
SeparableBlockFDCT_2D(input, coeff_sep, WIDTH, HEIGHT, BLOCK_SIZE);
printf("%dx%d separable block-based FDCT to %dx%d image : %.2f ms\n",
    BLOCK_SIZE, BLOCK_SIZE, WIDTH, HEIGHT, (double)1000*(clock()-start) / CLOCKS_PER_SEC);
// type casting and clipping
for(i = 0 ; i < HEIGHT ; i++){
    for(j = 0 ; j < WIDTH ; j++){
        double temp = coeff_sep[i][j];
        temp = temp > 255 ? 255 : temp < 0 ? 0 : temp;
        img_out_sep[i][j] = (BYTE)floor(temp + 0.5);
    }
}
// image write
ImgWrite("[SeparableFDCT]Lena(512x512).raw", img_out_sep, WIDTH, HEIGHT);

// separable block-based inverse DCT
start = clock();
SeparableBlockIDCT_2D(coeff_sep, output, WIDTH, HEIGHT, BLOCK_SIZE);
printf("%dx%d separable block-based IDCT to %dx%d image : %.2f ms\n",
    BLOCK_SIZE, BLOCK_SIZE, WIDTH, HEIGHT, (double)1000*(clock()-start) / CLOCKS_PER_SEC);

// type casting and clipping
for(i = 0 ; i < HEIGHT ; i++){
    for(j = 0 ; j < WIDTH ; j++){
        double temp = output[i][j];
        temp = temp > 255 ? 255 : temp < 0 ? 0 : temp;
        img_out_sep[i][j] = (BYTE)floor(temp + 0.5);
    }
}
// image write
ImgWrite("[SeparableIDCT]Lena(512x512).raw", img_out_sep, WIDTH, HEIGHT);
```

Assignment 2

```
// block-based 2-D forward DCT
start = clock();
BlockFDCT_2D(input, coeff_2D, WIDTH, HEIGHT, BLOCK_SIZE);
printf("\n%dx%d block-based 2-D FDCT to %dx%d image : %.2f ms\n",
    BLOCK_SIZE, BLOCK_SIZE, WIDTH, HEIGHT, (double)1000*(clock()-start) / CLOCKS_PER_SEC);

// type casting and clipping
for(i = 0 ; i < HEIGHT ; i++){
    for(j = 0 ; j < WIDTH ; j++){
        double temp = coeff_2D[i][j];
        temp = temp > 255 ? 255 : temp < 0 ? 0 : temp;
        img_out_2D[i][j] = (BYTE)floor(temp + 0.5);
    }
}
// image write
ImgWrite("[BlockFDCT]Lena(512x512).raw", img_out_2D, WIDTH, HEIGHT);

// block-based 2-D inverse DCT
start = clock();
BlockIDCT_2D(coeff_2D, output, WIDTH, HEIGHT, BLOCK_SIZE);
printf("%dx%d block-based 2-D IDCT to %dx%d image : %.2f ms\n",
    BLOCK_SIZE, BLOCK_SIZE, WIDTH, HEIGHT, (double)1000*(clock()-start) / CLOCKS_PER_SEC);

// type casting and clipping
for(i = 0 ; i < HEIGHT ; i++){
    for(j = 0 ; j < WIDTH ; j++){
        double temp = output[i][j];
        temp = temp > 255 ? 255 : temp < 0 ? 0 : temp;
        img_out_2D[i][j] = (BYTE)floor(temp + 0.5);
    }
}
// image write
ImgWrite("[BlockIDCT]Lena(512x512).raw", img_out_2D, WIDTH, HEIGHT);
```

Assignment 2

```
    // check whether two results are same
    cnt = 0;
    for(i = 0 ; i < HEIGHT ; i++){
        for(j = 0 ; j < WIDTH ; j++){
            if(img_out_2D[i][j] != img_out_sep[i][j])   cnt++;
        }
    }
    if(cnt == 0)    printf("₩nSeparable-DCT and 2D-DCT are same₩n");
    else            printf("₩nSeparable-DCT and 2D-DCT are different₩n");

    // memory free
    MemFree_UC_2D(img_in, HEIGHT);
    MemFree_UC_2D(img_out_sep, HEIGHT);
    MemFree_UC_2D(img_out_2D, HEIGHT);
    MemFree_D_2D(input, HEIGHT);
    MemFree_D_2D(output, HEIGHT);
    MemFree_D_2D(coeff_sep, HEIGHT);
    MemFree_D_2D(coeff_2D, HEIGHT);
}
```

Assignment 2

KWANGWOON UNIVERSITY

```
// 2-D memory allocation for unsigned char type
unsigned char** MemAlloc_UC_2D(const int width, const int height)
{
    unsigned char** arr;
    int i;
    arr = (unsigned char**)malloc(height * sizeof(unsigned char*));
    for(i = 0 ; i < height ; i++)   arr[i] = (unsigned char*)malloc(width * sizeof(unsigned char));
    return arr;
}

// 2-D memory allocation for double type
double** MemAlloc_D_2D(const int width, const int height)
{
    double** arr;
    int i;
    arr = (double**)malloc(height * sizeof(double*));
    for(i = 0 ; i < height ; i++)   arr[i] = (double*)malloc(width * sizeof(double));
    return arr;
}

// 2-D memory free for unsigned char type
void MemFree_UC_2D(unsigned char** arr, const int height)
{
    int i;
    for(i = 0 ; i < height ; i++)   free(arr[i]);
    free(arr);
}

// 2-D memory free for double type
void MemFree_D_2D(double** arr, const int height)
{
    int i;
    for(i = 0 ; i < height ; i++)   free(arr[i]);
    free(arr);
}

// image file read
void ImgRead(const char* filename, unsigned char** img_in, const int width, const int height)
{
    FILE* fp_in;
    int i;
    fopen_s(&fp_in, filename, "rb");
    for(i = 0 ; i < height ; i++)   fread(img_in[i], sizeof(unsigned char), width, fp_in);
    fclose(fp_in);
}

// image file write
void ImgWrite(const char* filename, const unsigned char** img_out, const int width, const int height)
{
    FILE* fp_out;
    int i;
    fopen_s(&fp_out, filename, "wb");
    for(i = 0 ; i < height ; i++)   fwrite(img_out[i], sizeof(unsigned char), width, fp_out);
    fclose(fp_out);
}
```

Assignment 2

```
// separable block-based forward DCT
void SeparableBlockFDCT_2D(const double** input, double** coeff, const int width, const int height, const int block_size)
{
    int i,j;
    double **temp_hor = MemAlloc_D_2D(block_size, block_size);
    double **temp_ver = MemAlloc_D_2D(block_size, block_size);

                                                    ?


    MemFree_D_2D(temp_hor, block_size);
    MemFree_D_2D(temp_ver, block_size);
}

// separable block-based inverse DCT
void SeparableBlockIDCT_2D(const double** coeff, double** output, const int width, const int height, const int block_size)
{
    int i,j;
    double **temp_hor = MemAlloc_D_2D(block_size, block_size);
    double **temp_ver = MemAlloc_D_2D(block_size, block_size);

                                                    ?


    MemFree_D_2D(temp_hor, block_size);
    MemFree_D_2D(temp_ver, block_size);
}
```

Assignment 2

```
// N-point 1-D forward DCT
void FDCT_1D(const double* input, double* coeff, const int N)
{
    const double PI = 3.1415926535;
    int n,k;

    for(k = 0 ; k < N ; k++){
        double beta = k == 0 ? 1/sqrt(2.0) : 1;
        double temp = 0;
        for(n = 0 ; n < N ; n++){
            double basis = cos( ((2*n+1)*PI*k) / (2.0*N) );
            temp += input[n] * basis;
        }
        temp *= sqrt(2/(double)N) * beta;

        coeff[k] = temp;
    }
}

// N-point 1-D inverse DCT
void IDCT_1D(const double* coeff, double* output, const int N)
{
    const double PI = 3.1415926535;
    int n,k;

    for(n = 0 ; n < N ; n++){
        double temp = 0;
        for(k = 0 ; k < N ; k++){
            double beta = k == 0 ? 1/sqrt(2.0) : 1;
            double basis = cos( ((2*n+1)*PI*k) / (2.0*N) );
            temp += beta * coeff[k] * basis;
        }
        temp *= sqrt(2/(double)N);

        output[n] = temp;
    }
}
```

$$\text{Forward DCT}: X[k] = \sqrt{\frac{2}{N}}\beta[k]\sum_{n=0}^{N-1} x[n]\cos\left(\frac{(2n+1)\pi k}{2N}\right), \quad 0 \le k \le N-1$$

$$\text{where } \beta[k] = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & \text{otherwise} \end{cases}$$

$$\text{Inverse DCT}: x[n] = \sqrt{\frac{2}{N}}\sum_{k=0}^{N-1} \beta[k]X[k]\cos\left(\frac{(2n+1)\pi k}{2N}\right), \quad 0 \le n \le N-1$$

$$\text{where } \beta[k] = \begin{cases} \frac{1}{\sqrt{2}}, & k = 0 \\ 1, & \text{otherwise} \end{cases}$$

Assignment 2

KWANGWOON UNIVERSITY

```
// matrix transpose
void MatTranspose(double **mat, const int size)
{
    int i,j;
    for(i = 0 ; i < size ; i++){
        for(j = i+1 ; j < size ; j++){
            double temp = mat[i][j];
            mat[i][j] = mat[j][i];
            mat[j][i] = temp;
        }
    }
}

// block-based 2-D forward DCT
void BlockFDCT_2D(const double** input, double** coeff, const int width, const int height, const int block_size)
{
    const double PI = 3.1415926535;
    int m,n, i,j,x,y;
    for(m = 0 ; m < width ; m += block_size){
        for(n = 0 ; n < height ; n += block_size){

            for(x = 0 ; x < block_size ; x++){
                double cx = x == 0 ? 1/sqrt(2.0) : 1;
                for(y = 0 ; y < block_size ; y++){
                    double cy = y == 0 ? 1/sqrt(2.0) : 1;
                    double temp = 0;
                    for(i = 0 ; i < block_size ; i++){
                        for(j = 0 ; j < block_size ; j++){

                            double basis = cos( ((2*i+1)*PI*x) / (2.0*block_size) ) * cos( ((2*j+1)*PI*y) / (2.0*block_size) );
                            temp += input[i][j] * basis;
                        }
                    }
                    temp *= (cx*cy) / (block_size/2.0);
                    coeff[x][y] = temp;
                }
            }
        }
    }
}
```

$$\text{Forward DCT}: F_{x,y} = \frac{c(x)c(y)}{N/2}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f_{i,j}\cos\left(\frac{(2i+1)\pi x}{2N}\right)\cos\left(\frac{(2j+1)\pi y}{2N}\right), \quad 0 \le x, y \le N-1$$

$$\text{where } c(l) = \begin{cases} \frac{1}{\sqrt{2}}, & l = 0 \\ 1, & \text{otherwise} \end{cases}$$

VIAME

Assignment 2

```
// block-based 2-D inverse DCT
void BlockIDCT_2D(const double** coeff, double** output, const int width, const int height, const int block_size)
{

}
```

Inverse DCT : $f_{i,j} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} \frac{c(x)c(y)}{N/2} F_{x,y} \cos\left(\frac{(2i+1)\pi x}{2N}\right)\cos\left(\frac{(2j+1)\pi y}{2N}\right), \quad 0 \le i, j \le N-1$

where $c(l) = \begin{cases} \frac{1}{\sqrt{2}}, & l = 0 \\ 1, & \text{otherwise} \end{cases}$

?

Assignment 2

< Original image >

< 8x8 block-based DCT coefficient >

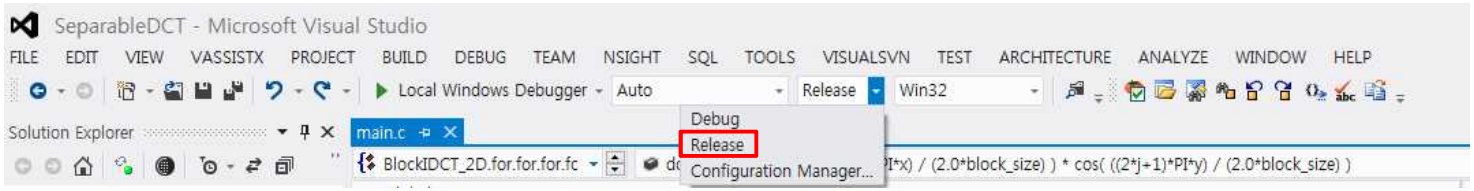
Assignment 2

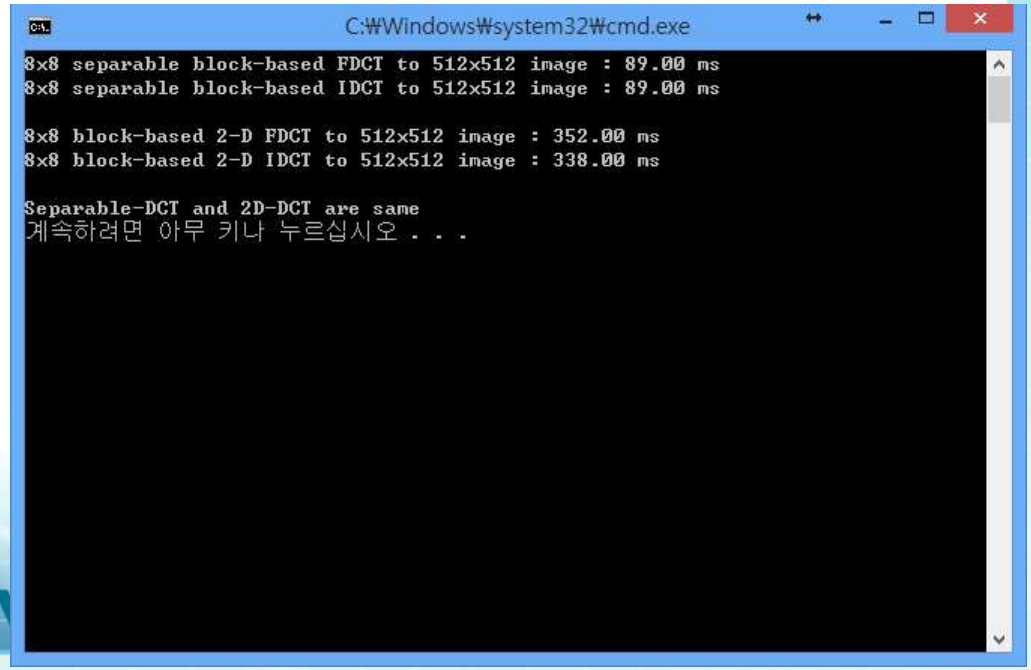
< 8x8 block-based DCT coefficient >

< Inverse DCT image >

Assignment 2

- Execution time comparison
  - CPU : Intel i7-4770 3.4 GHz
  - Windows 8 with Visual Studio 2012





KWANGWOON UNIVERSITY

Assignment

VIAME

Programming Guide

```
// block-based 2-D inverse DCT
void BlockIDCT_2D(const double** coeff, double** output, const int width, const int height, const int block_size)
{

}
```

Inverse DCT : $f_{i,j} = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} \frac{c(x)c(y)}{N/2} F_{x,y} \cos\left(\frac{(2i+1)\pi x}{2N}\right)\cos\left(\frac{(2j+1)\pi y}{2N}\right), \quad 0 \le i, j \le N-1$

where $c(l) = \begin{cases} \frac{1}{\sqrt{2}}, & l = 0 \\ 1, & \text{otherwise} \end{cases}$

?

Assignment

- Implement the separable DCT example
- Complete the separable FDCT & IDCT source code

```
// separable block-based forward DCT
void SeparableBlockFDCT_2D(const double** input, double** coeff, const int width, const int height, const int block_size)
{
    int i,j;
    double **temp_hor = MemAlloc_D_2D(block_size, block_size);
    double **temp_ver = MemAlloc_D_2D(block_size, block_size);

                                                    ?

    MemFree_D_2D(temp_hor, block_size);
    MemFree_D_2D(temp_ver, block_size);
}

// separable block-based inverse DCT
void SeparableBlockIDCT_2D(const double** coeff, double** output, const int width, const int height, const int block_size)
{
    int i,j;
    double **temp_hor = MemAlloc_D_2D(block_size, block_size);
    double **temp_ver = MemAlloc_D_2D(block_size, block_size);

                                                    ?

    MemFree_D_2D(temp_hor, block_size);
    MemFree_D_2D(temp_ver, block_size);
}
```