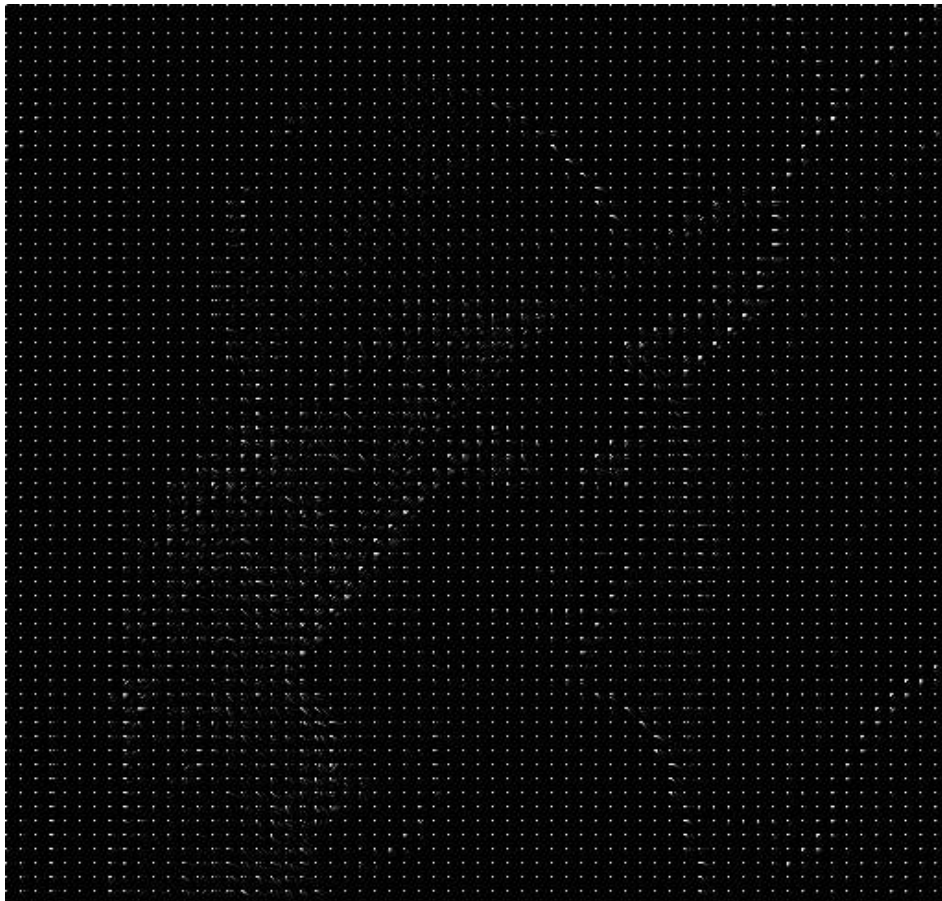


정경주

The purpose of this assignment is to implement DCT algorithm for a given still image. In order to evaluate the performance, we can compare the subjective quality reconstructed by inverse transform as well as the objective criterion such as MSE (mean square error). Furthermore, we can exploit the frequency spectrum in which the energy compaction performance could be compared. For this purpose, carry out the following assignments.

1. For the given 512x512 image 'Lena',
 - (a) Perform 8x8 forward DCT and plot the frequency spectrum on the monitor in proper scale for easy observation.



<OutImg_DCT_Lena>

- (b) Perform 8x8 inverse DCT to obtain the reconstructed image. Compute the MSE.



<OutImg_Inverse_Lena>

=====

1.Lena 2. Boat 3. exit : 1

Lena DCT is created.
Lena Inverse is created.

MSE of Lena is 0.083988

=====

<실행창>

2. For the given 512x512 image 'Boat',
 - (a) Perform 8x8 forward DCT and plot the frequency spectrum on the monitor

in proper scale for easy observation..



<OutImg_DCT_Boat>

(a) Perform 8x8 inverse DCT to obtain the reconstructed image. Compute the MSE.



<OutImg_Inverse>

=====

1.Lena 2. Boat 3. exit : 2

Boat DCT is created.
Boat Inverse is created.

MSE of Boat is 0.083633

=====

<실행창>

```
=====
1.Lena  2. Boat  3. exit  :  4
잘못 입력 하셨습니다.
=====
1.Lena  2. Boat  3. exit  :  3
=====
계속하려면 아무 키나 누르십시오 . . .
<다른 입력 시 결과>
```

3. Discuss the results.

이번 과제는 오리지날 영상을 DCT를 통해 주파수 축에서 어떤 분포를 가지고 있는지 확인하고 역변환 했을 때의 오차를 알아보는 것이었다. 우선 이중 for문을 이용해서 아래 그림과 같이 NxN 을 전체적으로 읽고 그 안에서 8x8 씩 오리지날 영상을 읽어서 ix에 저장해 준다.

```
for(i=0; i<N; i=i+8)
{
    for(j=0; j<N; j=j+8)
    {
        for(ii=0; ii<B_size; ii++){
            for(jj=0; jj<B_size; jj++){
                ix[ii][jj]=InImg[i+ii][j+jj];
            }
        }
    }
}
```

ix는 다음 for문에서 s에 임시 저장하고 클램핑을 한 후 다시 8x8만큼 저장해 주는데 OutImg[i+k][j+m] 형태로 인덱스가 안에 두개씩 있기 때문에 8x8씩 겹치지 않고 누적되서 저장될 수 있다.

```
dct8x8 (ix);

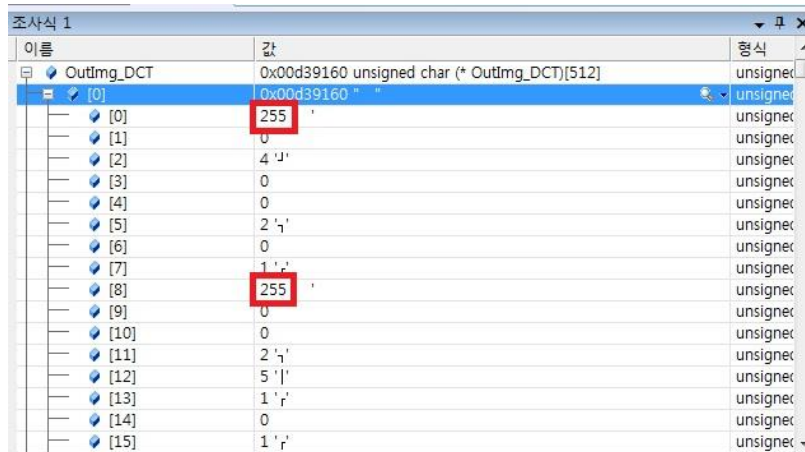
for(k=0; k<B_size; k++){
    for(m=0; m<B_size; m++){
        s=ix[k][m];
        OutImg_DCT[i+k][j+m]= s>255 ? 255 : s<0 ? 0 : s;
    }
}
```

ix는 바로 idct함수로 들어가서 연산이 되고 연산된 값 또한 OutImg_Inverse에 누적되서 저장이 된다. DCT결과를 확인하는 영상은 255가 넘어가는 값이 존재해서 클램핑을 해주는 것이고 idct를 할때는 원래 값이 존재하는 matrix를

Inverse해줘야 오리지널 값을 제대로 복원해 낼 수가 있다.

DCT는 이론적으로는 Lossless지만 프로그램이 돌아가면서 형변환도 되고 소수점이 무시되는 부분이 있기 때문에 오리지널과 처리된 영상간의 차이가 발생하게 되지만 차이값을 알 수 있는 MSE는 HW3에 비하면 엄청 작고 실제로 구현해보면 Lossy라는 것도 확인할 수 있다.

DCT는 저주파 부분으로 모아주는 효과가 있는데 아래 그림이 디버깅을 통해서 OutImg_DCT처럼 제대로 값이 나왔는지 확인한 과정이다.



이름	값	형식
OutImg_DCT	0x00d39160 unsigned char (* OutImg_DCT)[512]	unsigned
[0]	0x00d39160 " 255 "	unsigned
[1]	0	unsigned
[2]	4 'd'	unsigned
[3]	0	unsigned
[4]	0	unsigned
[5]	2 'h'	unsigned
[6]	0	unsigned
[7]	1 'e'	unsigned
[8]	255	unsigned
[9]	0	unsigned
[10]	0	unsigned
[11]	2 'h'	unsigned
[12]	5 'l'	unsigned
[13]	1 'e'	unsigned
[14]	0	unsigned
[15]	1 'e'	unsigned

You can refer to the following 8x8 forward and inverse DCT algorithm.

```
#define B_size 8
#define nint(x)      ( (x)<0. ? (int)((x)-0.5) : (int)((x)+0.5) )
/*-----
  8x8 block DCT
  -----*/
dct8x8 (ix)
int ix[][B_size];

{
    static float pi=3.141592653589793238;
    float x[B_size][B_size],z[B_size][B_size],y[B_size],c[40],s[40],
          ft[4],fxy[4],yy[B_size],zz;
    int i,ii,jj;

    for (i=0; i<40; i++ ) {
        zz = pi * (float)(i+ 1) / 64.0;
        c[i] = cos(zz);
        s[i] = sin(zz);
    }

    for (ii=0; ii<B_size; ii++ )
        for (jj=0; jj<B_size; jj++ )
            x[ii][jj] = (float)ix[ii][jj];

    for (ii=0; ii<B_size; ii++ ) {
        for (jj=0; jj<B_size; jj++ )
            y[jj] = x[ii][jj];

        for (jj=0; jj<4; jj++ )
            ft[jj] = y[jj] + y[7-jj];

        fxy[0] = ft[0] + ft[3];
        fxy[1] = ft[1] + ft[2];
```

```

    fxy[2] = ft[1] - ft[2];
    fxy[3] = ft[0] - ft[3];

    ft[0] = c[15] * (fxy[0] + fxy[1]);
    ft[2] = c[15] * (fxy[0] - fxy[1]);
    ft[1] = s[7] * fxy[2] + c[7] * fxy[3];
    ft[3] = -s[23] * fxy[2] + c[23] * fxy[3];

    for (jj=4; jj<8; jj++)
        yy[jj] = y[7-jj] - y[jj];

    y[4] = yy[4];
    y[7] = yy[7];
    y[5] = c[15] * (-yy[5] + yy[6]);
    y[6] = c[15] * (yy[5] + yy[6]);

    yy[4] = y[4] + y[5];
    yy[5] = y[4] - y[5];
    yy[6] = -y[6] + y[7];
    yy[7] = y[6] + y[7];

    y[0] = ft[0];
    y[4] = ft[2];
    y[2] = ft[1];
    y[6] = ft[3];
    y[1] = s[3] * yy[4] + c[3] * yy[7];
    y[5] = s[19] * yy[5] + c[19] * yy[6];
    y[3] = -s[11] * yy[5] + c[11] * yy[6];
    y[7] = -s[27] * yy[4] + c[27] * yy[7];

    for (jj=0; jj<B_size; jj++)
        z[ii][jj] = y[jj];

}

for (ii=0; ii<B_size; ii++) {

```



```

for (jj=0; jj<B_size; jj++)
    y[jj] = z[jj][ii];

for (jj=0; jj<4; jj++)
    ft[jj] = y[jj] + y[7-jj];

fxy[0] = ft[0] + ft[3];
fxy[1] = ft[1] + ft[2];
fxy[2] = ft[1] - ft[2];
fxy[3] = ft[0] - ft[3];

ft[0] = c[15] * (fxy[0] + fxy[1]);
ft[2] = c[15] * (fxy[0] - fxy[1]);
ft[1] = s[7] * fxy[2] + c[7] * fxy[3];
ft[3] = -s[23] * fxy[2] + c[23] * fxy[3];

for (jj=4; jj<8; jj++)
    yy[jj] = y[7-jj] - y[jj];

y[4] = yy[4];
y[7] = yy[7];
y[5] = c[15] * (-yy[5] + yy[6]);
y[6] = c[15] * (yy[5] + yy[6]);

yy[4] = y[4] + y[5];
yy[5] = y[4] - y[5];
yy[6] = -y[6] + y[7];
yy[7] = y[6] + y[7];

y[0] = ft[0];
y[4] = ft[2];
y[2] = ft[1];
y[6] = ft[3];
y[1] = s[3] * yy[4] + c[3] * yy[7];
y[5] = s[19] * yy[5] + c[19] * yy[6];
y[3] = -s[11] * yy[5] + c[11] * yy[6];

```

```

        y[7] = -s[27] * yy[4] + c[27] * yy[7];

        for (jj=0; jj<B_size; jj++)
            y[jj] = y[jj] / 4.0;

        for (jj=0; jj<B_size; jj++)
            z[jj][ii] = y[jj];
    }

    for (ii=0; ii<B_size; ii++)
        for (jj=0; jj<B_size; jj++)
            ix[ii][jj] = nint(z[ii][jj]);
}

/*-----
  8x8 block inverse DCT
  -----*/
idct8x8 (ix)

int ix[][B_size];

{
    static float pi=3.141592653589793238;
    float x[B_size][B_size],z[B_size][B_size],y[B_size],c[40],s[40],
        ait[4],aixy[4],yy[B_size],zz;
    int i,ii,jj;

    for (i=0; i<40; i++) {
        zz = pi * (float)(i+1) / 64.0;
        c[i] = cos(zz);
        s[i] = sin(zz);
    }

    for (ii=0; ii<B_size; ii++)
        for (jj=0; jj<B_size; jj++)
            x[ii][jj] = (float)ix[ii][jj];

```

```

for (ii=0; ii<B_size; ii++ ) {
    for (jj=0; jj<B_size; jj++ )
        y[jj] = x[jj][ii];

    ait[0] = y[0];
    ait[1] = y[2];
    ait[2] = y[4];
    ait[3] = y[6];

    aaxy[0] = c[15] * (ait[0] + ait[2]);
    aaxy[1] = c[15] * (ait[0] - ait[2]);
    aaxy[2] = s[7] * ait[1] - s[23] * ait[3];
    aaxy[3] = c[7] * ait[1] + c[23] * ait[3];

    ait[0] = aaxy[0] + aaxy[3];
    ait[1] = aaxy[1] + aaxy[2];
    ait[2] = aaxy[1] - aaxy[2];
    ait[3] = aaxy[0] - aaxy[3];

    yy[4] = s[3] * y[1] - s[27] * y[7];
    yy[5] = s[19] * y[5] - s[11] * y[3];
    yy[6] = c[19] * y[5] + c[11] * y[3];
    yy[7] = c[3] * y[1] + c[27] * y[7];

    y[4] = yy[4] + yy[5];
    y[5] = yy[4] - yy[5];
    y[6] = -yy[6] + yy[7];
    y[7] = yy[6] + yy[7];

    yy[4] = y[4];
    yy[7] = y[7];
    yy[5] = c[15] * (-y[5] + y[6]);
    yy[6] = c[15] * (y[5] + y[6]);

    for (jj=0; jj<4; jj++ )

```

```

        y[jj] = ait[jj] + yy[7-jj];

    for (jj=4; jj<8; jj++)
        y[jj] = ait[7-jj] - yy[jj];

    for (jj=0; jj<B_size; jj++)
        z[jj][ii] = y[jj];

}

for (ii=0; ii<B_size; ii++) {
    for (jj=0; jj<B_size; jj++)
        y[jj] = z[ii][jj];

    ait[0] = y[0];
    ait[1] = y[2];
    ait[2] = y[4];
    ait[3] = y[6];

    aaxy[0] = c[15] * (ait[0] + ait[2]);
    aaxy[1] = c[15] * (ait[0] - ait[2]);
    aaxy[2] = s[7] * ait[1] - s[23] * ait[3];
    aaxy[3] = c[7] * ait[1] + c[23] * ait[3];

    ait[0] = aaxy[0] + aaxy[3];
    ait[1] = aaxy[1] + aaxy[2];
    ait[2] = aaxy[1] - aaxy[2];
    ait[3] = aaxy[0] - aaxy[3];

    yy[4] = s[3] * y[1] - s[27] * y[7];
    yy[5] = s[19] * y[5] - s[11] * y[3];
    yy[6] = c[19] * y[5] + c[11] * y[3];
    yy[7] = c[3] * y[1] + c[27] * y[7];

    y[4] = yy[4] + yy[5];
    y[5] = yy[4] - yy[5];

```

```
y[6] = -yy[6] + yy[7];
```

```
y[7] = yy[6] + yy[7];
```

```
yy[4] = y[4];
```

```
yy[7] = y[7];
```

```
yy[5] = c[15] * (-y[5] + y[6]);
```

```
yy[6] = c[15] * (y[5] + y[6]);
```

```
for (jj=0; jj<4; jj++)
```

```
    y[jj] = ait[jj] + yy[7-jj];
```

```
for (jj=4; jj<8; jj++)
```

```
    y[jj] = ait[7-jj] - yy[jj];
```

```
for (jj=0; jj<B_size; jj++)
```

```
    z[ii][jj] = y[jj] / 4.0;
```

```
}
```

```
for (ii=0; ii<B_size; ii++)
```

```
    for (jj=0; jj<B_size; jj++)
```

```
        ix[ii][jj] = nint(z[ii][jj]);
```

```
}
```