# Lab02

| | |
|---|---|
| 유형 | |
| 교수님 | |
| 학년/학기 | |

**코드에 대한 설명은 코드에 안의 주석으로 설명해놨습니다.**

**약간의 쉬운 내용이나 중요도가 낮은 내용들은 코드가 아예 없거나 주석이 없습니다.**

## 분석에 대한 결과

해당 이미지를 바꾸는 과정에 대해서는 YCrCb값으로의 변화 이후 block DCT를 하고 quantization을 한 후 이를 히스토그램으로 만들었다.

이후에 IDCT를 통해서 RGB 변환을 하여서 결과값으로 다시 원본값을 복원했지만, DCT과정에서 고주파의 일부를 버리고 양자화 과정에서 뒷자리를 버리는 과정에서 값이 손실되기 때문에 원본값을 복원하는 과정에서 데이터가 손실된다.

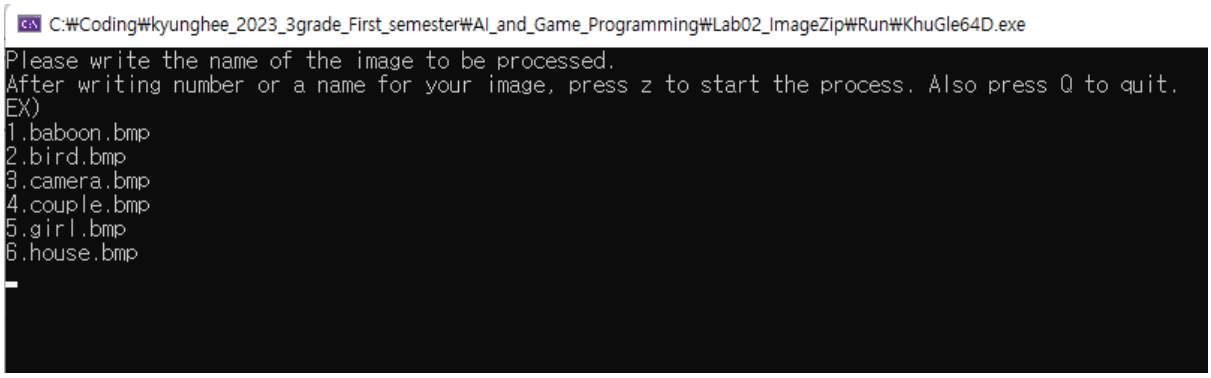
때문에 정확한 색상이 표시가 되지 않고, 고주파의 엣지값이 손실이 되어서 약간 흐리고 엣지값이 부드러운 결과값을 우리는 확인할수 있다.

## 시작

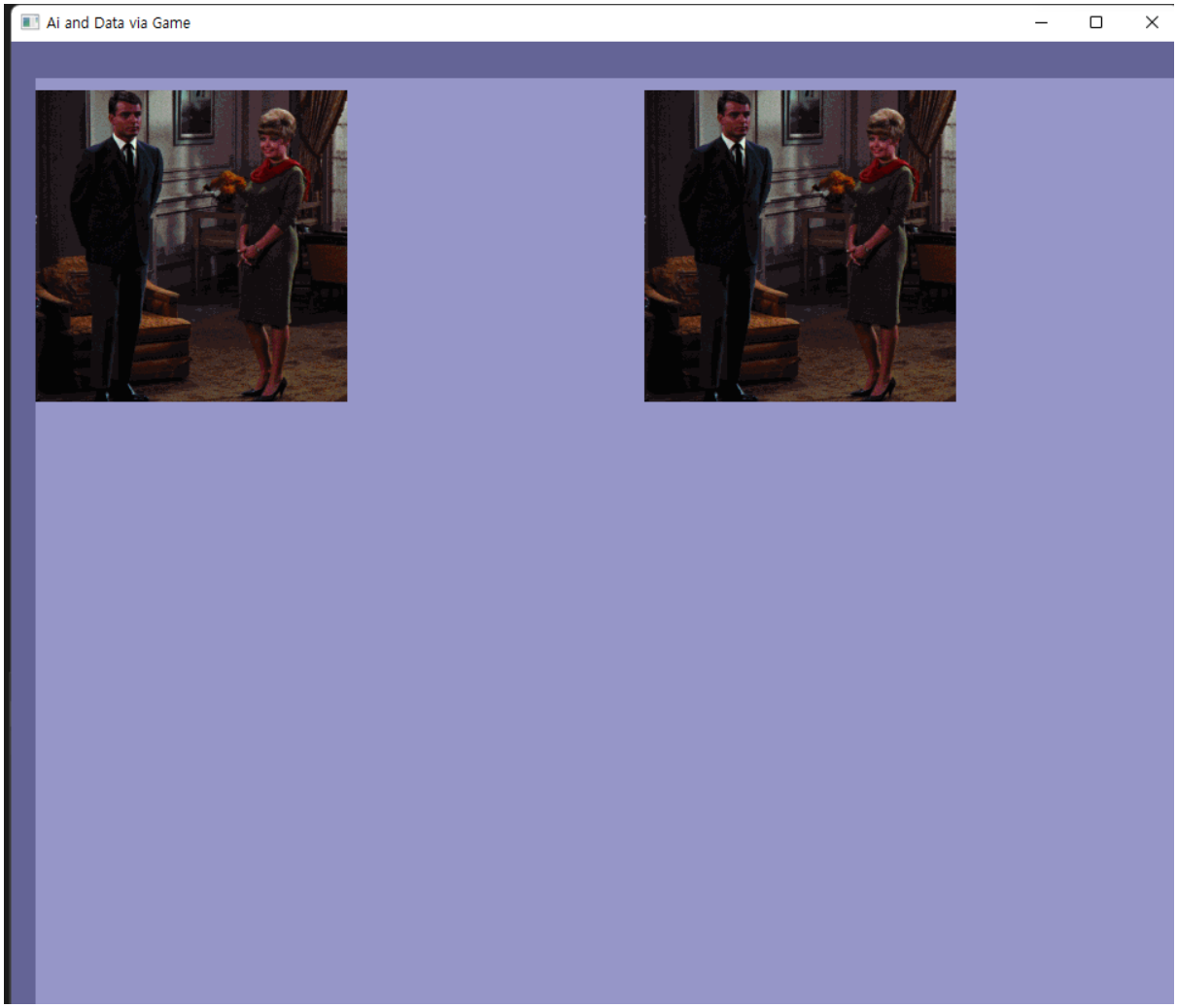
프로세스를 진행할 이미지의 이름을 입력하세요

이미지의 이름을 입력하거나 아래의 예시의 숫자번호를 입력해도 됩니다.

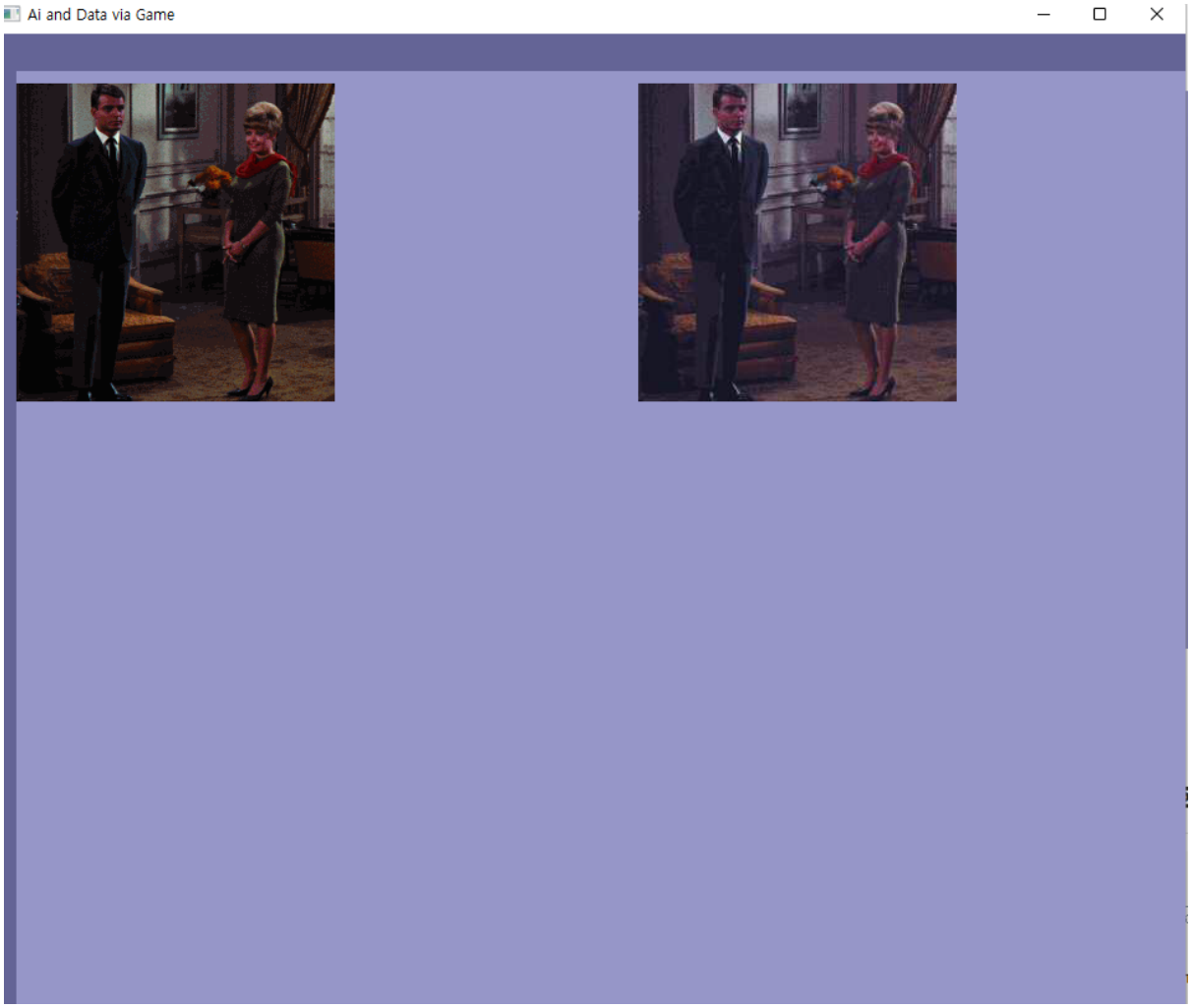
z를 누르면 시작되고, q를 누르면 종료됩니다.




## 원본값 :




## 결과값




## 터미널 출력 결과 (0의 개수 + 세로축 히스토그램 + 엔트로피 + PSNR)

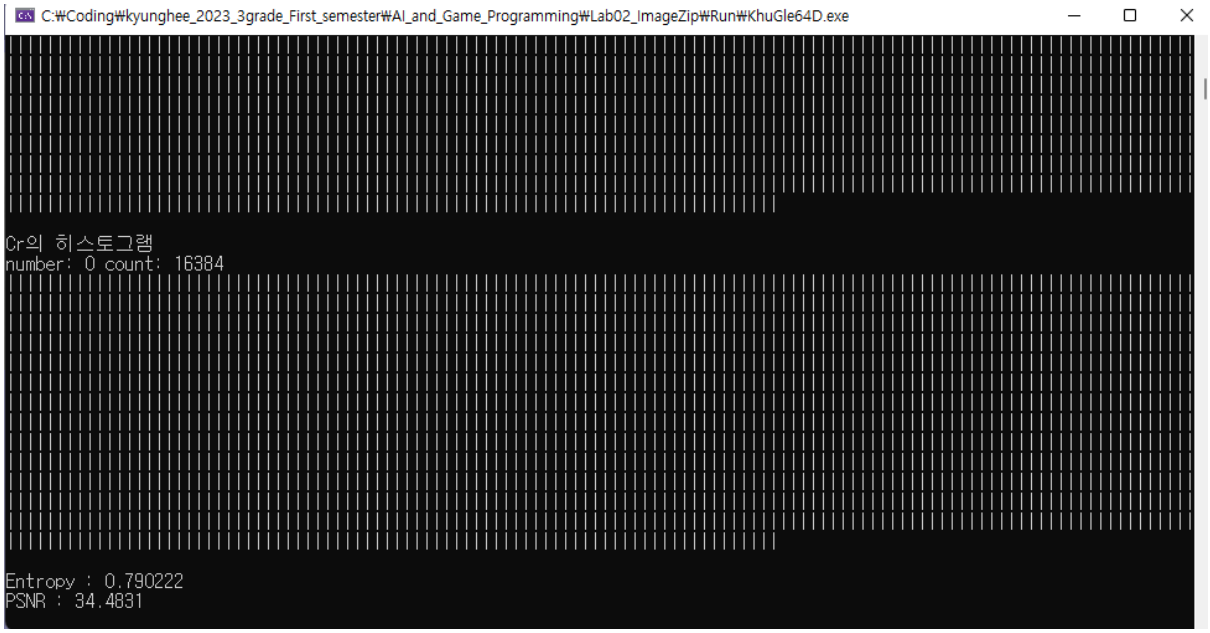
1. 처음에 0의 갯수를 출력한다.
2. 히스토그램의 데이터를 출력한다.

   형식은 히스토그램의 범위 (number) 와 개수(count)를 출력한다.

   추가적으로 |를 출력하는데 이는 10단위로 |를 출력한다.
3. 엔트로피를 출력한다.
4. PSNR를 출력한다.


채널별(Y, Cb, Cr) 0의 개수
Y: 57203 Cb: 16098 Cr: 15827
Y 히스토그램
number: -39 count: 1
number: -38 count: 1
number: -37 count: 0
number: -36 count: 0
number: -35 count: 1
number: -34 count: 1
number: -33 count: 0
number: -32 count: 1
number: -31 count: 0
number: -30 count: 0
number: -29 count: 0
number: -28 count: 2
number: -27 count: 0
number: -26 count: 3
number: -25 count: 2
number: -24 count: 1
number: -23 count: 2
number: -22 count: 3
number: -21 count: 1
number: -20 count: 2
number: -19 count: 2
number: -18 count: 0
number: -17 count: 4
number: -16 count: 8
number: -15 count: 4
number: -14 count: 4
number: -13 count: 7
number: -12 count: 16
number: -11 count: 14
number: -10 count: 15


number: -7 count: 49
number: -6 count: 62
number: -5 count: 91
number: -4 count: 150
number: -3 count: 238
number: -2 count: 571
number: -1 count: 2280
number: 0 count: 57203
number: 1 count: 2295
number: 2 count: 648
number: 3 count: 308
number: 4 count: 214
number: 5 count: 123
number: 6 count: 98
number: 7 count: 83
number: 8 count: 93
number: 9 count: 74


**PSNR값 34가 나옴 (이미지 사용값: bird.bmp)**

## 초기값 init + RGB값에 대해서 배열 초기화 + YCrCb 배열 형성 및 이미지 생성

```
// 입력된 RGB값, YCrCb값 init YCbCr값은 2x2 공간으로 묶을 것이기 때문에
// /2를 해서 공간을 4배 줄인다.
double** InputR = dmatrix(m_pImageLayer->m_Image.m_nH, m_pImageLayer->m_Image.m_nW);
double** InputG = dmatrix(m_pImageLayer->m_Image.m_nH, m_pImageLayer->m_Image.m_nW);
double** InputB = dmatrix(m_pImageLayer->m_Image.m_nH, m_pImageLayer->m_Image.m_nW);

double** InputY = dmatrix(m_pImageLayer->m_Image.m_nH, m_pImageLayer->m_Image.m_nW);
double** InputCb = dmatrix(m_pImageLayer->m_Image.m_nH/2, m_pImageLayer->m_Image.m_nW/2);
double** InputCr = dmatrix(m_pImageLayer->m_Image.m_nH/2, m_pImageLayer->m_Image.m_nW/2);
```

```
double** DCT_Y = dmatrix(m_pImageLayer->m_Image.m_nH, m_pImageLayer->m_Image.m_nW);
double** DCT_Cb = dmatrix(m_pImageLayer->m_Image.m_nH / 2, m_pImageLayer->m_Image.m_nW / 2);
double** DCT_Cr = dmatrix(m_pImageLayer->m_Image.m_nH / 2, m_pImageLayer->m_Image.m_nW / 2);

double** Quan_Y = dmatrix(m_pImageLayer->m_Image.m_nH, m_pImageLayer->m_Image.m_nW);
double** Quan_Cb = dmatrix(m_pImageLayer->m_Image.m_nH / 2, m_pImageLayer->m_Image.m_nW / 2);
double** Quan_Cr = dmatrix(m_pImageLayer->m_Image.m_nH / 2, m_pImageLayer->m_Image.m_nW / 2);

double** OutR = dmatrix(m_pImageLayer->m_Image.m_nH, m_pImageLayer->m_Image.m_nW);
double** OutG = dmatrix(m_pImageLayer->m_Image.m_nH, m_pImageLayer->m_Image.m_nW);
double** OutB = dmatrix(m_pImageLayer->m_Image.m_nH, m_pImageLayer->m_Image.m_nW);

// 1. RGB 값 matrix init
for (int y = 0; y < m_pImageLayer->m_Image.m_nH; ++y)
  for (int x = 0; x < m_pImageLayer->m_Image.m_nW; ++x){
    InputR[y][x] = m_pImageLayer->m_Image.m_Red[y][x];
    InputG[y][x] = m_pImageLayer->m_Image.m_Green[y][x];
    InputB[y][x] = m_pImageLayer->m_Image.m_Blue[y][x];
  }

// 입력 영상이 WXH이면, Y는 동일하고 CbCr은 (W/2)X(H/2)
// 2. Y값 Cr값 Cb값 matrix init
// Y = R*0.299+G*0.587+B*0.114
for (int y = 0; y < m_pImageLayer->m_Image.m_nH; ++y)
  for (int x = 0; x < m_pImageLayer->m_Image.m_nW; ++x)
  {
    InputY[y][x] = InputR[y][x] * 0.299 + InputG[y][x] * 0.587 + InputB[y][x] * 0.144;

    // 짝수값만 clamp해서 2x2 값으로 설정함.
    if (y % 2 == 0 && x % 2 == 0)
    {
      InputCb[y / 2][x / 2] = (- 0.16874) * InputR[y][x] + (-0.33126) * InputG[y][x] + 0.5 * InputB[y][x];
      InputCr[y / 2][x / 2] = 0.5 * InputR[y][x] + (- 0.41869) * InputG[y][x] + (- 0.08131)* InputB[y][x];
    }
```

## DCT 작업 (이는 이미 KHU라이브러리에 구현된 함수가 존재)

```
// block DCT에서 block 크기를 8X8
// 3. DCT 작업 -> 8X8 block으로 만들기
// void DCT2D(double **Input, double **Output, int nW, int nH, int nBlockSize)
DCT2D(InputY, DCT_Y, m_pImageLayer->m_Image.m_nW, m_pImageLayer->m_Image.m_nH, 8);
DCT2D(InputCb, DCT_Cb, m_pImageLayer->m_Image.m_nW / 2, m_pImageLayer->m_Image.m_nH / 2, 8);
DCT2D(InputCr, DCT_Cr, m_pImageLayer->m_Image.m_nW / 2, m_pImageLayer->m_Image.m_nH / 2, 8);}
```

## 양자화 작업

```
// Quantization후에 각 채널별(Y, Cb, Cr)
// 1) 0의 개수, 2) 히스토그램 계산, 3) entropy 계산
// 4. Quantization 설정
for (int y = 0; y < m_pImageLayer->m_ImageOut.m_nH; ++y) {
  for (int x = 0; x < m_pImageLayer->m_ImageOut.m_nW; ++x)
  {
    // 해당 과정에서 데이터의 손실이 발생하는데 복구해도 해당과정에서
    //사라진 정보에 의해 완벽하게 복원될 수 없음
    Quan_Y[y][x] = round(DCT_Y[y][x] / QuantizationTable[y % 8][x % 8]);
```

```
    // Zero값 count 및 MAX 값과 MIN값을 찾아서 나중에 범위를 통해
    // 이미지 압축에 활용
    if (Quan_Y[y][x] == 0)
      Y_Zero++;
    if (Quan_Y[y][x] > Y_MAX)
      Y_MAX = Quan_Y[y][x];
    if (Quan_Y[y][x] < Y_MIN)
      Y_MIN = Quan_Y[y][x];

    // 2X2 배열로 만든 것에 대해서 clamp
    if (x % 2 == 0 && y % 2 == 0) {
      Quan_Cb[y/2][x/2]=round(DCT_Cb[y/2][x/2]/QuantizationTable_2[y/2%8][x/2%8]);
      Quan_Cr[y/2][x/2]=round(DCT_Cr[y/2][x/2]/QuantizationTable_2[y/2%8][x/2%8]);

      if (Quan_Cb[y / 2][x / 2] == 0)
        Cb_Zero++;
      if (Quan_Cb[y / 2][x / 2] > Cb_MAX)
        Cb_MAX = Quan_Cb[y / 2][x / 2];
      if (Quan_Cb[y / 2][x / 2] < Cb_MIN)
        Cb_MIN = Quan_Cb[y / 2][x / 2];

      if (Quan_Cr[y / 2][x / 2] == 0)
        Cr_Zero++;
      if (Quan_Cr[y / 2][x / 2] > Cr_MAX)
        Cr_MAX = Quan_Cr[y / 2][x / 2];
      if (Quan_Cr[y / 2][x / 2] < Cr_MIN)
        Cr_MIN = Quan_Cr[y / 2][x / 2];
    }
}
```

## 엔트로피 계산

```
// 엔트로피 계산 세팅
double delta;
double entropy = 0;

// 데이터 구조 초기화
int* ArrayA = (int*)calloc(Y_MAX - Y_MIN + 1, sizeof(int));
int* ArrayCb = (int*)calloc(Cb_MAX - Cb_MIN + 1, sizeof(int));
int* ArrayCr = (int*)calloc(Cr_MAX - Cr_MIN + 1, sizeof(int));

// YCrCb에 대해서 데이터 집합 생성
for (int y = 0; y < m_pImageLayer->m_ImageOut.m_nH; ++y) {
  for (int x = 0; x < m_pImageLayer->m_ImageOut.m_nW; ++x)
  {
    ArrayA[int(Quan_Y[y][x]) - Y_MIN] += 1;

    if (y % 2 == 0 && x % 2 == 0) {
      ArrayCb[int(Quan_Cb[y / 2][x / 2]) - Cb_MIN] += 1;
      ArrayCr[int(Quan_Cr[y / 2][x / 2]) - Cr_MIN] += 1;
    }
  }
}

std::cout << " 채널별(Y, Cb, Cr) 0의 개수\n";
std::cout << "Y: " << Y_Zero << " Cb: " << Cb_Zero << " Cr: " << Cr_Zero << "\n";

// 5. Y히스토그램 생성
std::cout << "Y 히스토그램 \n";
for (int i = 0; i < Y_MAX - Y_MIN + 1; i++) {

  std::string name_number = std::to_string(i + Y_MIN);
```

```
  std::string name_number2 = std::to_string(ArrayA[i]);
  std::cout << "number: " << name_number << " count: "<< name_number2 << "\n";

  // 히스토그램 출력
  int histogram_number = ArrayA[i] / 10;
  for (int k = 0; k < histogram_number; k++)
    std::cout << "|";
  std::cout << '\n';

  // 엔트로피 계산
  if (ArrayA[i] != 0) {
    delta = ArrayA[i] / double(m_pImageLayer->m_ImageOut.m_nH * m_pImageLayer->m_ImageOut.m_nW);
    entropy += (delta * log2(delta));
  }
}
std::cout << "\n";
```

## 이미지 복원 작업 → 지금까지의 작업 역순으로 진행

```
// 원래 이미지 복원 7~11

// 7. Quantization된 정보 다시 원상복구
for (int y = 0; y < m_pImageLayer->m_ImageOut.m_nH; ++y) {
for (int x = 0; x < m_pImageLayer->m_ImageOut.m_nW; ++x)
{
  DCT_Y[y][x] = Quan_Y[y][x] * QuantizationTable[y % 8][x % 8];

  if (y % 2 == 0 && x % 2 == 0) {
    DCT_Cb[y / 2][x / 2] = Quan_Cb[y / 2][x / 2] * QuantizationTable_2[y / 2 % 8][x / 2 % 8];
    DCT_Cr[y / 2][x / 2] = Quan_Cr[y / 2][x / 2] * QuantizationTable_2[y / 2 % 8][x / 2 % 8];
  }
}
}

// 8. IDCT2D 역변환
IDCT2D(DCT_Y, InputY, m_pImageLayer->m_Image.m_nW, m_pImageLayer->m_Image.m_nH, 8);
IDCT2D(DCT_Cb, InputCb, m_pImageLayer->m_Image.m_nW / 2, m_pImageLayer->m_Image.m_nH / 2, 8);
IDCT2D(DCT_Cr, InputCr, m_pImageLayer->m_Image.m_nW / 2, m_pImageLayer->m_Image.m_nH / 2, 8);

// 9. YCbCr을 RGB로 변환함
for (int y = 0; y < m_pImageLayer->m_ImageOut.m_nH; ++y) {
for (int x = 0; x < m_pImageLayer->m_ImageOut.m_nW; ++x)
{
  InputR[y][x] = InputY[y][x] + InputCr[y/2][x/2] * 1.40200;
  InputG[y][x] = InputY[y][x] + InputCb[y/2][x/2] * -0.34414 + InputCr[y/2][x/2] * -0.71414;
  InputB[y][x] = InputY[y][x] + InputCb[y/2][x/2] * 1.77200;
}
}


// 10. 이미지 복원 결과값, 전체 수정값 다시 출력
// 해당 결과를 MIN과 MAX로 범위로 재구현 하는 것이 아닌 0~255값을 사용하고,
// 이 범위를 넘어가는 값은 0과 255로 바꾼다.
for (int y = 0; y < m_pImageLayer->m_ImageOut.m_nH; ++y)
for (int x = 0; x < m_pImageLayer->m_ImageOut.m_nW; ++x)
{

  if (InputR[y][x] <= 255 && InputR[y][x] >= 0)
    m_pImageLayer->m_ImageOut.m_Red[y][x] = InputR[y][x];
  else if (InputR[y][x] < 0)
    m_pImageLayer->m_ImageOut.m_Red[y][x] = 0;
  else
    m_pImageLayer->m_ImageOut.m_Red[y][x] = 255;
```

```
  if (InputG[y][x] <= 255 && InputG[y][x] >= 0)
    m_pImageLayer->m_ImageOut.m_Green[y][x] = InputG[y][x];
  else if (InputG[y][x] < 0)
    m_pImageLayer->m_ImageOut.m_Green[y][x] = 0;
  else
    m_pImageLayer->m_ImageOut.m_Green[y][x] = 255;

  if (InputB[y][x] <= 255 && InputB[y][x] >= 0)
    m_pImageLayer->m_ImageOut.m_Blue[y][x] = InputB[y][x];
  else if (InputB[y][x] < 0)
    m_pImageLayer->m_ImageOut.m_Blue[y][x] = 0;
  else
    m_pImageLayer->m_ImageOut.m_Blue[y][x] = 255;

}
```

## 이미지 생성 및 출력 함수

```
// 이미지 생성 함수
void CKhuGleImageLayer::DrawImage() {
  int OffsetX, OffsetY;

  for (int y = 0; y < m_nH; y++)
    for (int x = 0; x < m_nW; x++)
    {
      m_ImageBgR[y][x] = KgGetRed(m_bgColor);
      m_ImageBgG[y][x] = KgGetGreen(m_bgColor);
      m_ImageBgB[y][x] = KgGetBlue(m_bgColor);
    }

  // 원본 이미지 출력
  if (m_Image.m_Red && m_Image.m_Green && m_Image.m_Blue)
  {
    for (int y = 0; y < m_Image.m_nH && y < m_nH; ++y)
      for (int x = 0; x < m_Image.m_nW && x < m_nW; ++x)
      {
        m_ImageBgR[y + 10][x] = m_Image.m_Red[y][x];
        m_ImageBgG[y + 10 ][x] = m_Image.m_Green[y][x];
        m_ImageBgB[y + 10][x] = m_Image.m_Blue[y][x];
      }
  }

  // 결과값 이미지 출력
  if (m_ImageOut.m_Red && m_ImageOut.m_Green && m_ImageOut.m_Blue)
  {
    OffsetX = 500;
    OffsetY = 10;
    for (int y = 0; y < m_ImageOut.m_nH && y + OffsetY < m_nH; ++y)
      for (int x = 0; x < m_ImageOut.m_nW && x + OffsetX < m_nW; ++x)
      {
        m_ImageBgR[y + OffsetY][x + OffsetX] = m_ImageOut.m_Red[y][x];
        m_ImageBgG[y + OffsetY][x + OffsetX] = m_ImageOut.m_Green[y][x];
        m_ImageBgB[y + OffsetY][x + OffsetX] = m_ImageOut.m_Blue[y][x];
      }
  }
}
```

## PDF에 나오는 양자화 테이블 목록

```
// pdf에 나오는 테이블
double QuantizationTable[8][8] = {
{16, 11, 10, 16, 24, 40, 51, 61},
{12, 12, 14, 19, 26, 58, 60, 55},
{14, 13, 16, 24, 40, 57, 69, 56},
{14, 17, 22, 29, 51, 87, 80, 62},
{18, 22, 37, 56, 68, 109, 103, 77},
{24, 35, 55, 64, 81, 104, 113, 92},
{49, 64, 78, 87, 103, 121, 120, 101},
{72, 92, 95, 98, 112, 100, 103, 99}
};

double QuantizationTable_2[8][8] = {
{17, 18, 24, 47, 99, 99, 99, 99},
{18, 21, 26, 66, 99, 99, 99, 99},
{24, 26, 56, 99, 99, 99, 99, 99},
{47, 66, 99, 99, 99, 99, 99, 99},
{99, 99, 99, 99, 99, 99, 99, 99},
{99, 99, 99, 99, 99, 99, 99, 99},
{99, 99, 99, 99, 99, 99, 99, 99},
{99, 99, 99, 99, 99, 99, 99, 99}
};
```