

note: 원본 코드에서는 CV_LOAD_IMAGE_COLOR, CV_RGB2GRAY, CV_RGB2YUV를 사용하고 있으나, OpenCV v4.0 이상에서는 IMREAD_COLOR, COLOR_RGB2GRAY, COLOR_RGB2YUV를 사용하므로 전체 프로그램에서 해당 flag를 모두 수정하였다.

- 1) CV_LOAD_IMAGE_COLOR -> IMREAD_COLOR
- 2) CV_RGB2GRAY -> COLOR_RGB2GRAY
- 3) CV_RGB2YUV -> COLOR_RGB2YUV

hist.func.h

hist.func.h는 PDF, CDF 생성 및 히스토그램 stretching, equalization, matching에 사용할 변수와 함수들을 포함한다. cal_PDF_RGB(), cal_CDF_RGB() 함수를 완성해 컬러 이미지에서 PDF와 CDF를 추출할 수 있게 하였으며, plot() 함수와 drawGraph() 함수를 추가해 각 프로그램에서 각자의 PDF, CDF로 호출해 사용할 수 있게끔 hist.func.h를 수정하였다.

```
#define L 256 // # of intensity levels
```

이미지의 intensity는 0부터 255까지의 값을 가지므로 L=256으로 설정한다.

1. *cal_PDF(Mat &input), *cal_CDF(Mat &input)

cal_PDF(), cal_CDF() 함수는 input에 대해 각각 PDF와 CDF를 계산하는 함수이며, 이때 input은 grayscale(single channel) 이미지이다.

$$F_X(x) = \int_{-\infty}^x f_X(t) dt = P(X \leq x)$$

에서 $f_X(t)$ 는 확률 변수 X 의 분포를 나타내는 PDF(Probability Density Function, 확률밀도함수)이고, PDF를 적분한 값인 $F_X(x)$ 는 X 가 x 이하일 확률을 나타내는 CDF(Cumulative Distribution Function, 누적분포함수)이다. 1D discrete form에서 PDF(r)은 intensity=r인 픽셀이 존재할 확률이며, CDF(r)은 전체 픽셀 중 intensity $k \leq r$ 인 픽셀의 비율이다.

$$PDF(r) = \frac{n_r}{N}, CDF(r) = \sum_{k=0}^r PDF(k) = \sum_{k=0}^r n_k / N$$

1.1 cal_PDF()

```
int count[L] = { 0 };
float *PDF = (float*)calloc(L, sizeof(float));
```

크기가 256(L=256)이고 0으로 초기화된 count 배열과 256개의 float 데이터를 저장할 PDF 배열을 생성한다. count 배열에는 각 intensity의 빈도를, PDF에는 각 intensity가 등장할 확률을 저장한다.

```
// Count
for (int i = 0; i < input.rows; i++)
    for (int j = 0; j < input.cols; j++)
        count[input.at<G>(i, j)]++;
```

input 이미지의 모든 픽셀을 순회하며, 해당 픽셀이 가진 intensity의 등장 횟수를 계산한다.

```
// Compute PDF
for (int i = 0; i < L; i++)
    PDF[i] = (float)count[i] / (float)(input.rows * input.cols);
```

```
return PDF;
```

for문을 돌며 intensity i 가 존재할 확률($\text{count}[i]/\text{전체 픽셀 수}$)를 계산해 $\text{PDF}[i]$ 에 저장하고, 완성된 PDF 배열을 반환한다.

1.2 cal_CDF()

```
float *cal_CDF(Mat &input) {  
  
    int count[L] = { 0 };  
    float *CDF = (float*)calloc(L, sizeof(float));  
  
    // Count  
    for (int i = 0; i < input.rows; i++)  
        for (int j = 0; j < input.cols; j++)  
            count[input.at<G>(i, j)]++;  
  
    // Compute CDF  
    for (int i = 0; i < L; i++) {  
        CDF[i] = (float)count[i] / (float)(input.rows * input.cols);  
  
        if (i != 0)  
            CDF[i] += CDF[i - 1];  
    }  
  
    return CDF;  
}
```

count 배열 계산까지는 배열명을 $\text{CDF}[i]$ 로 사용하는 것을 제외하면 $\text{cal_PDF}()$ 와 같은 원리로 작동된다.

```
// Compute CDF  
for (int i = 0; i < L; i++) {  
    CDF[i] = (float)count[i] / (float)(input.rows * input.cols);  
  
    if (i != 0)  
        CDF[i] += CDF[i - 1];  
}  
  
return CDF;
```

이후 for문을 돌며 CDF 데이터를 계산한다. $i=0$ 일 때는 이전에 누적된 값이 없으므로 $\text{CDF}[i] = \text{PDF}[i] = (\text{float})\text{count}[i] / (\text{float})(\text{input.rows} * \text{input.cols})$ 로 계산하지만, 1 이상의 i 에 대해서는 $\text{CDF}[i] = \text{PDF}[i]$ 에 더불어 이전 인덱스의 CDF 값인 $\text{CDF}[i-1]$ 을 더해 주어야 한다.

2. **cal_PDF_RGB(Mat &input), **cal_CDF_RGB(Mat &input)

$\text{cal_PDF_RGB}()$, $\text{cal_CDF_RGB}()$ 함수는 컬러 이미지인 input 에 대해 각 색상별 PDF와 CDF를 계산하는 함수이다. grayscale 이미지에서 DF와 CDF를 구하는 것과 동일한 원리이며, 컬러 이미지는 3개의 channel을 사용하므로 이들 R, G, B에 대해 각각의 PDF 또는 CDF를 계산하면 된다.

2.1 cal_PDF_RGB()

```
int count[L][3] = { 0 };  
float **PDF = (float**)malloc(sizeof(float*) * L);  
  
for (int i = 0; i < L; i++)  
    DF[i] = (float*)calloc(3, sizeof(float));
```

3 channel 이미지이므로 count와 PDF 모두 2차원 배열을 사용한다.

```
// Count
for (int i = 0; i < input.rows; i++) {
    for (int j = 0; j < input.cols; j++) {
        // [0] = B intensity, [1] = G intensity, [2] = R intensity
        for (int k = 0; k < 3; k++) { // RGB
            count[input.at<Vec3b>(i, j)[k]][k]++; // count[intensity][channel]
        }
    }
}
```

CV_8UC3 타입의 행렬 A에서 $A(i, j, k) = A.at<Vec3b>(i, j)[k]$ 이므로, 3중 for문을 순회하며 count 배열에 intensity별 빈도를 저장한다. for (int k = 0; k < 3; k++)에서 k가 0, 1, 2일 때 input.at<Vec3b>(i, j)[k]는 (i, j) 픽셀의 B, G, R intensity값이므로, 위 코드를 사용해 해당 intensity를 가지는 B, G, R의 빈도를 계산할 수 있다.

```
// Compute PDF
for (int i = 0; i < L; i++) {
    for (int k = 0; k < 3; k++) {
        PDF[i][k] = (float)count[i][k] / (float)(input.rows * input.cols);
    }
}

return PDF;
```

계산한 count 배열을 바탕으로 PDF를 계산해 반환한다. PDF[i][k]는 k channel의 i intensity를 가지는 픽셀이 존재할 확률로, count[i][j]를 전체 픽셀 수로 나누어 계산한다.

2.2 cal_CDF_RGB()

```
int count[L][3] = { 0 };
float **CDF = (float**)malloc(sizeof(float*) * L);

for (int i = 0; i < L; i++)
    CDF[i] = (float*)calloc(3, sizeof(float));

for (int i = 0; i < input.rows; i++) {
    for (int j = 0; j < input.cols; j++)
        for (int k = 0; k < 3; k++) {
            count[input.at<Vec3b>(i, j)[k]][k]++;
        }
}

}
```

1.2와 마찬가지로 CDF[i]를 계산하는 방법을 제외하면 cal_PDF_RGB() 함수와 유사하다.

```
// Compute CDF
for (int k = 0; k < 3; k++) {
    for (int i = 0; i < L; i++) { // kth channel intensity i
        CDF[i][k] = (float)count[i][k] / (float)(input.rows * input.cols);

        if (i != 0) {
            CDF[i][k] += CDF[i - 1][k];
        }
    }
}

return CDF;
```

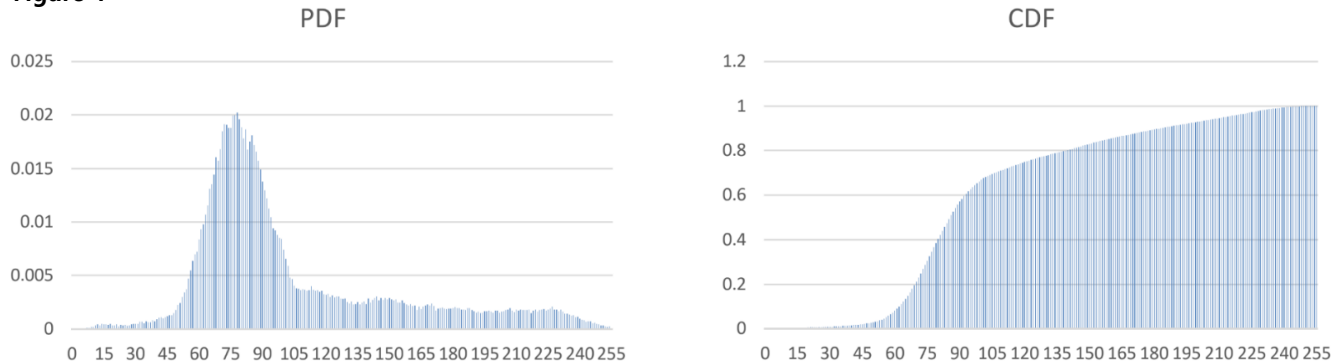
1.2와 동일하게 $CDF[i][k] = PDF[i][k]$ 로 계산하고, i가 1 이상인 경우에만 이전 CDF 값 $CDF[i-1][k]$ 를 더한다. 이때 CDF[i][k]는 색상 k 중 intensity가 i 이하인 픽셀이 존재할 확률이다.

3. plot()

```
void plot(float* function, string opt, string title) {  
...
```

plot() 함수는 opt 옵션에 따라 PDF 또는 CDF 데이터를 그래프화하여 별도의 창으로 출력하는 함수이다. opt는 function이 PDF 데이터인지 CDF 데이터인지를 나타내는 인자, title은 그래프 제목으로 표시될 문자열 인자임에 주의해야 한다.

Figure 1



OSP-Lec03-Pixel의 34쪽 그래프 예시를 참고하여(그림1) PDF 그래프는 0.005 간격, CDF 그래프는 0.2 간격으로 y축을 라벨링하였으며, x축은 두 그래프 모두 15 간격으로 라벨링하였다.

3.1 그래프 설정

x, y축과 윈도우 크기를 설정하고, opt에 따라 y축 최댓값 maxSize를 저장한다..

```
int x = 512; // x: 0~255  
int y = 250; // y  
int margin = 50;  
int w = x + margin * 2; // width of window  
int h = y + margin * 2; // height~  
  
Mat graph = Mat::zeros(h, w, CV_8UC3);  
float maxSize;
```

그래프 크기는 512*250으로 설정하였으며, margin=50으로 설정해 전체 창과 그래프가 표시되는 부분 사이에 50만큼의 간격을 둔 뒤 이를 합친 전체 창의 가로, 세로 크기는 각각 w와 h변수에 저장한다. 이를 바탕으로 w*h 크기 Mat형 graph를 생성하고 초기화한다.

```
if (opt == "PDF") {  
    maxSize = 0.025f;  
  
    for (int i = 0; i < L; i++) {  
        if (function[i] > maxSize) {  
            maxSize = function[i]; // 초과하는 값 있을 경우 변경  
            h += maxSize;  
        }  
    }  
}  
else {  
    maxSize = 1.2f;  
}
```

y축 최댓값 변수 maxSize는 (그림1)과 동일하게 PDF에서는 0.025, CDF에서는 1.2를 default로 설정하였다. 주어진 "input.jpg"이외 다른 이미지가 입력되어 PDF 최댓값이 "input.jpg"의 최댓값과 달라지는 경우,

해당 이미지의 최대값으로 maxSize를 업데이트하고 윈도우 크기 또한 변경한다.

3.2 PDF, CDF 값 그리기

line() 함수를 사용해 function[] 값을 직선으로 표시한다.

```
for (int i = 0; i < L; i++) {
    int pointX = margin + cvRound((i / 255.0) * x);
    int pointY = margin + y - cvRound((function[i] / maxSize) * y);

    line(graph, Point(pointX, y + margin), Point(pointX, pointY), Scalar(255, 255, 255));
}
```

for 루프를 돌며 intensity i의 PDF 또는 CDF 값 function[i](=PDF[i] or CDF[i])를 그래프에 표시한다. pointX에는 x축의 길이(0~255) 대비 i의 비율을 계산해 좌표평면에 점이 표시될 위치를 저장하고, pointY에는 y축의 길이(maxSize) 대비 function[i]의 비율을 저장하며, 이후 graph의 (pointX, y+margin)부터 (pointX, pointY)까지 흰색(255, 255, 255) 선을 그려 그래프가 출력되게 한다.

```
drawGraph(graph, x, y, w, h, margin, maxSize, opt, title);

namedWindow(title, WINDOW_AUTOSIZE);
imshow(title, pdf_graph);
```

이후 drawGraph() 함수를 호출해 제목, 좌표축, 라벨을 추가하고 별도의 창에 결과를 출력한다.

4. drawGraph()

```
void drawGraph (Mat & graph, int x, int y, int w, int h, int margin, float maxSize, const
string & opt, string title) {
...
}
```

plot() 함수에서 호출해 사용하는 drawGraph() 함수는 5.1에서 설정한 그래프 설정(x, y, w, h, margin), 그래프 옵션(PDF or CDF, opt)과 그에 따른 데이터 최대값(maxSize=PDFmaxSize or CDFmaxSize), 그래프에 표시될 제목(title)을 전달받고, 그래프에 좌표축과 라벨 눈금을 추가해 가독성을 개선한 것을 출력하는 함수이다.

4.1 x축 표시

```
line(graph, Point(margin, margin + y), Point(margin + x, margin + y), Scalar(255, 255, 255), 1);
for (int i = 0; i <= 255; i += 15) { // label
    int pointX = margin + cvRound((i / 255.0) * x);

    putText(graph, to_string(i), Point(pointX - 10, margin + y + 10), FONT_ITALIC, 0.4,
    Scalar(255, 255, 255), 1);
}
```

line() 함수를 사용해 여백을 제외한 출력 창 영역에 x축을 표시하고, 축의 색상은 흰색(255, 255, 255), 굵기는 1로 설정한다. (그림1)과 같이 x축은 15 간격으로 라벨링되므로, (i+=15)로 for문을 반복하며 적당한 위치(pointX-10, margin+y+10)에 15 간격의 인덱스를 표시한다. pointX 변수 설정은 5.2와 동일하게 전체 축 길이 대비 해당 지점의 길이 비율로 한다.

4.2 y축 표시

(그림1)에서 PDF의 y축 라벨은 0부터 0.025까지 0.005 간격, CDF의 y축 라벨은 0부터 1.2까지 0.2 간격이므로, 전달된 opt에 따라 구현을 달리해야 한다.

```
if (opt == "PDF") { // PDF
    vector<float> label = {};
    float iPos = maxSize / 6;
```

```

for (float i = 0.000; i <= maxSize; i += iPos) {
    label.push_back(i);
}

for (int i = 0; i < label.size(); i++) { // 라벨 표시
    int pointY = margin + y - cvRound((label[i] / maxSize) * y);
    putText(graph, to_string(label[i]).substr(0, 5), Point(margin - 40, pointY + 5),
        FONT_ITALIC, 0.4, Scalar(255, 255, 255), 1);
}
}

```

PDF 그래프의 y축 라벨을 label에 저장하고, putText() 함수를 사용하여 해당 배열의 원소를 적당한 위치에 표시한다. maxSize에 따라 표시될 인덱스의 개수와 종류가 달라지므로, label을 vector로 선언해 필요한 만큼 값을 추가하여 사용할 수 있게 하며, for문에서 i를 증가시킬 iPos=maxSize/6을 설정해 iPos 간격으로 6개의 라벨을 삽입할 수 있도록 한다.

```

else {
    float label[] = { 0.0f, 0.2f, 0.4f, 0.6f, 0.8f, 1.0f, 1.2f }; // 0~1.2

    for (int i = 0; i < 7; i++) {
        int pointY = margin + y - cvRound((label[i] / maxSize) * y);

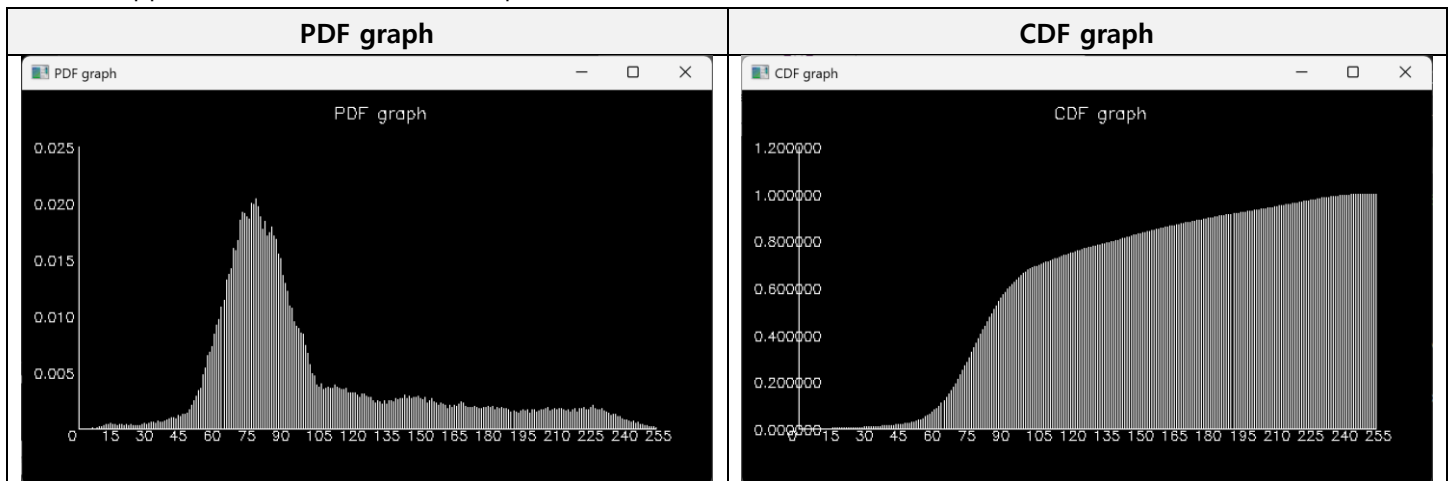
        putText(graph, to_string(label[i]), Point(margin - 40, pointY + 5), FONT_ITALIC,
            0.4, Scalar(255, 255, 255), 1);
    }
}

```

CDF 또한 PDF 그래프와 유사하게 구현하며, 사용하는 라벨만 0.0, 0.2, 0.4, 0.6, 0.8, 1.0, 1.2로 달리하여 label 배열에 저장한다. PDF와 달리 라벨 인덱스가 변하지 않으므로 label은 배열로 선언한다.

5. 실행 결과

PDF_CDF.cpp에서 PDF, CDF를 전달받아 plot() 함수를 실행한 결과, 아래와 같은 그래프가 출력된다.



PDF_CDF.cpp

PDF_CDF.cpp는 OpenCV와 "hist_func.h"에 정의된 cal_PDF(), cal_CDF() 함수를 사용하여 주어진 "input.jpg"의 PDF와 CDF를 계산하고, 이를 각각 "PDF.txt"와 "CDF.txt" 파일에 저장하는 코드이다.

1. 이미지 read 및 grayscale 변환

```
Mat input = imread("input.jpg", IMREAD_COLOR);
Mat input_gray;

cvtColor(input, input_gray, COLOR_RGB2GRAY); // convert RGB to Grayscale
```

imread() 함수와 IMREAD_COLOR를 사용해 "input.jpg"를 컬러 이미지로 읽어 온 것을 input에 저장한다. 이후 cvtColor() 함수와 COLOR_RGB2GRAY를 사용해 input을 grayscale로 변환하고 이를 input_gray에 저장한다.

2. PDF, CDF 계산 및 파일로 저장

```
// PDF, CDF txt files
FILE *f_PDF, *f_CDF;

fopen_s(&f_PDF, "PDF.txt", "w+");
fopen_s(&f_CDF, "CDF.txt", "w+");
```

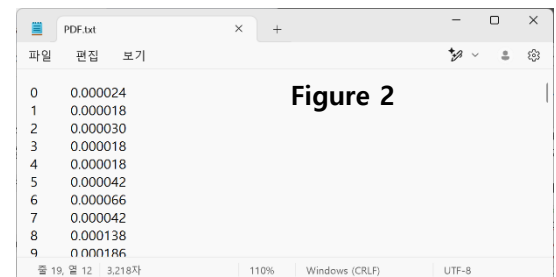
PDF, CDF 데이터를 저장할 파일 포인터 f_PDF, f_CDF를 선언한 후, fopen_s() 함수를 사용하여 "PDF.txt"와 "CDF.txt" 파일을 쓰기 모드로 open한다. 만약 파일이 존재하지 않는다면 파일을 생성한다.

```
// each histogram
float *PDF = cal_PDF(input_gray); // PDF of Input image(Grayscale) : [L]
float *CDF = cal_CDF(input_gray); // CDF of Input image(Grayscale) : [L]

for (int i = 0; i < L; i++) {
    // write PDF, CDF
    fprintf(f_PDF, "%d\t%f\n", i, PDF[i]);
    fprintf(f_CDF, "%d\t%f\n", i, CDF[i]);
}

plot(PDF, "PDF", "PDF graph");
plot(CDF, "CDF", "CDF graph");
```

cal_PDF(), cal_CDF() 함수로 input_gray의 PDF와 CDF를 계산한 것을 각각 PDF, CDF 포인터에 저장하며, for문을 사용해 L=256만큼 반복하며 i번째 level의 PDF와 CDF를 f_PDF와 f_CDF 파일에 write한다. 저장 형태는 (그림1)과 같이 "i(level) PDF[i]" 형태이다. 이후 "hist_func.h" 헤더의 plot() 함수를 호출해 히스토그램을 작성하며, PDF, CDF 포인터의 메모리를 release하고 f_PDF와 f_CDF 파일 포인터를 close한다.



```
// memory release
free(PDF);
free(CDF);
fclose(f_PDF);
fclose(f_CDF);
```

3. 결과 출력

```
namedWindow("RGB", WINDOW_AUTOSIZE);
imshow("RGB", input);
```

```
namedWindow("Grayscale", WINDOW_AUTOSIZE);
imshow("Grayscale", input_gray);

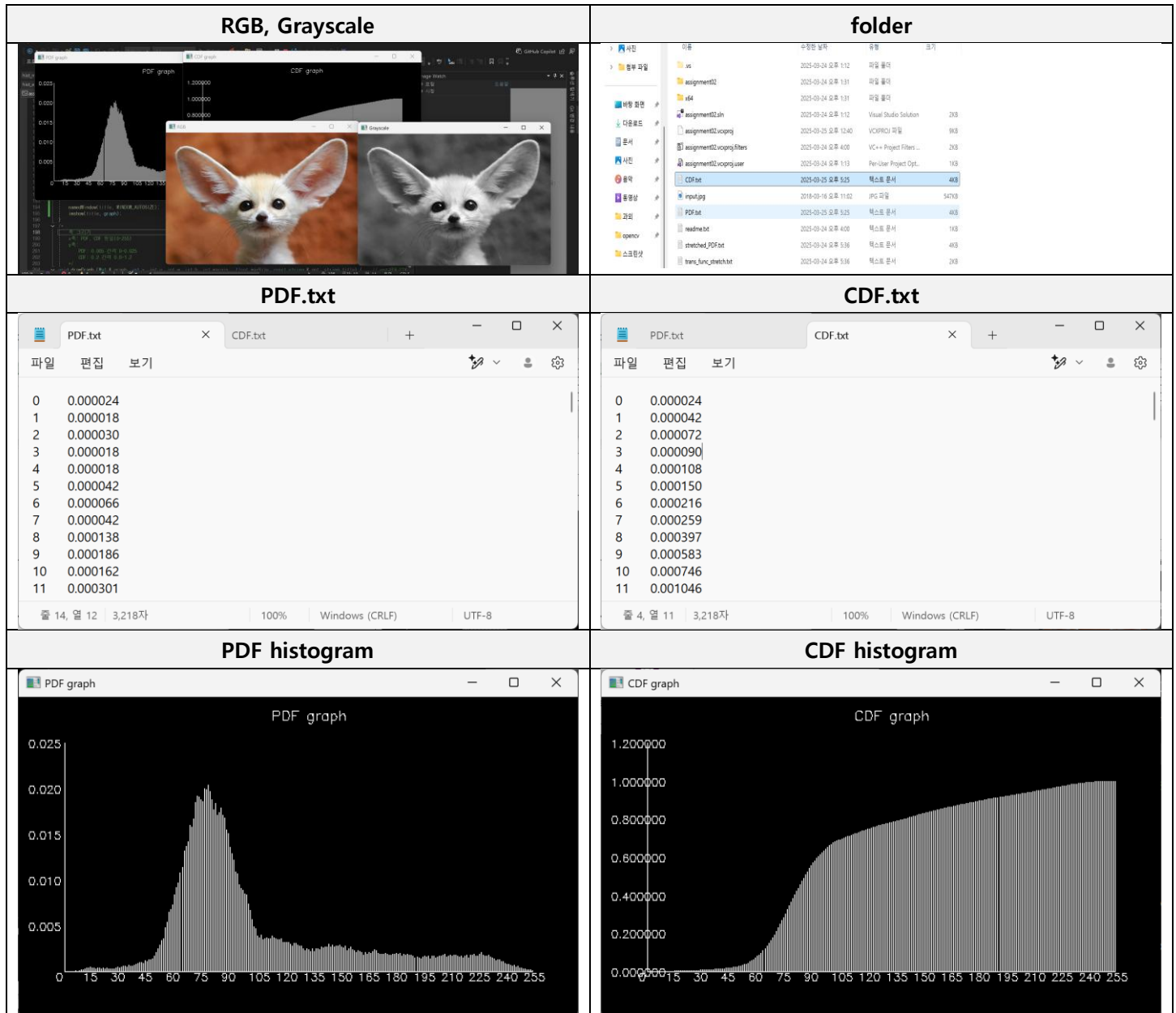
waitKey(0);

return 0;
```

namedWindow(), imshow() 함수를 사용해 컬러 이미지와 grayscale 이미지를 별도의 결과 창에 출력하고 프로그램을 종료한다.

4. 결과 분석

visual studio IDE에서 OpenCV 라이브러리, C++을 사용해 코드를 컴파일 및 실행하였으며, 입력 이미지로는 "input.jpg"를 사용하였다.



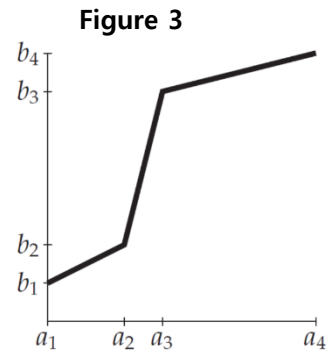
실행 결과 컬러 이미지, grayscale 이미지와 PDF/CDF 그래프가 별도의 창에 정상적으로 출력되었으며, 파일 폴더에 PDF.txt와 CDF.txt가 생성되어 이미지의 PDF, CDF 정보를 저장하고 있는 것을 확인할 수 있다.

hist_stretching.cpp

hist_stretching.cpp는 histogram stretching을 통해 "input.jpg"의 contrast를 개선하는 코드로, 이때 histogram stretching은 이미지의 히스토그램이 균형 있게 나타나도록 이미지를 변환하는 기법이다.

히스토그램이 특정한 구간에 집중적으로 분포하는, 명암비가 낮은 이미지에 대해 linear stretching function을 적용하면 해당 구간의 히스토그램이 "stretch"되는데, 이때 linear stretching function의 general form은 아래와 같으며, (그림3)과 같이 나타낼 수 있다.

$$y = \frac{b_{i+1} - b_i}{a_{i+1} - a_i}(x - a_i) + b_i$$



PDF_CDF.cpp의 실행 결과 intensity가 50과 110 사이인 구간에서 PDF가 가장 높게 나타나고 있으므로, 해당 구간을 stretching하기 위해 hist_stretching.cpp에서는 $(a_2, b_2)=(50, 10)$ 과 $(a_3, b_3)=(110, 110)$ 지점에서 구간을 분할하고 있다. stretching된 이미지에서는 기존에 (50, 110) 구간이던 히스토그램이 (10, 110)구간으로 늘어났으며, 이로 인해 명암비가 높아져 이미지가 이전보다 더 선명해진다.

a, b 설정을 달리 하여 stretching할 구간, stretching할 정도 등을 변경할 수 있다.

1. main()

1.1 이미지 read 및 grayscale 변환

```
Mat input = imread("input.jpg", IMREAD_COLOR);
Mat input_gray;

cvtColor(input, input_gray, COLOR_RGB2GRAY);           // convert RGB to Grayscale

Mat stretched = input_gray.clone();
```

"input.jpg"를 grayscale로 변환한 것을 input_gray에 저장한 후 이를 복제(clone())하여 stretched에 저장한다.

1.2 파일 불러오기 및 PDF, stretching 계산

```
FILE *f_PDF;
FILE *f_stretched_PDF;
FILE *f_trans_func_stretch;

fopen_s(&f_PDF, "PDF.txt", "w+");
fopen_s(&f_stretched_PDF, "stretched_PDF.txt", "w+");
fopen_s(&f_trans_func_stretch, "trans_func_stretch.txt", "w+");
```

PDF_CDF.cpp에서 작성한 "PDF.txt"를 f_PDF 포인터로 open하고, histogram stretching된 PDF를 저장할 f_stretched_PDF와 linear stretching function 값을 저장할 f_trans_func_stretch 포인터를 선언한 후 "stretched_PDF.txt", "trans_func_stretch.txt" 파일을 open한다.

```
G trans_func_stretch[L] = { 0 };

float *PDF = cal_PDF(input_gray);
```

stretching function을 기록할 L=255 크기의 trans_func_stretch 배열을 생성하고, cal_PDF() 함수를 호출하여 input_gray 이미지의 PDF를 계산해 PDF에 저장한다.

```
linear_stretching(input_gray, stretched, trans_func_stretch, 50, 110, 10, 110);
float *stretched_PDF = cal_PDF(stretched);
```

linear_stretching() 함수를 호출하여 input_gray를 stretching한 것을 stretched에 저장한 후 stretched 이미지의 PDF를 계산하여 stretched_PDF에 저장한다.

1.3 파일에 출력 및 메모리 할당 해제

```
for (int i = 0; i < L; i++) {
    // write PDF
    fprintf(f_PDF, "%d\t%f\n", i, PDF[i]);
    fprintf(f_stretched_PDF, "%d\t%f\n", i, stretched_PDF[i]);

    // write transfer functions
    fprintf(f_trans_func_stretch, "%d\t%d\n", i, trans_func_stretch[i]);
}
```

for루프를 순회하며 PDF 데이터, stretching한 후의 PDF 데이터, linear stretching function 값을 각각 파일에 출력한다.

```
plot(PDF, "PDF", "PDF");
plot(stretched_PDF, "PDF", "PDF(stretching)");
```

plot() 함수를 호출해 원본 이미지와 stretching한 이미지의 히스토그램을 출력한다.

```
free(PDF);
free(stretched_PDF);
fclose(f_PDF);
fclose(f_stretched_PDF);
fclose(f_trans_func_stretch);
```

할당된 메모리들을 해제하고, 파일을 close한다.

1.4 히스토그램 파일 생성

```
FILE* f_hist;
FILE* f_stretched_hist;

fopen_s(&f_hist, "hist.txt", "w+");
fopen_s(&f_stretched_hist, "stretched_hist.txt", "w+");
```

원본 히스토그램과 stretching된 이미지의 히스토그램을 저장할 파일 포인터와 파일을 생성한다.

```
int histSize = L;
float ran[] = { 0, 256 };
const float* range = { ran };
Mat hist, stretched_hist;
calcHist(&input_gray, 1, 0, Mat(), hist, 1, &histSize, &range);
calcHist(&stretched, 1, 0, Mat(), stretched_hist, 1, &histSize, &range);
```

cv::calcHist()를 사용해 히스토그램을 계산하기 위해 함수에 전달할 변수들을 설정한다. 히스토그램의 가로 축 크기 histSize는 L로 설정하고, ran[] 배열에는 히스토그램의 최솟값과 최댓값을 저장한다. 원본 히스토그램과 stretching 히스토그램을 저장할 Mat형 hist, stretched_hist를 선언하고, 이 변수들을 인자로 전달해 각각 히스토그램을 계산한다.

```
for (int i = 0; i < histSize; i++) {
    fprintf(f_hist, "%d\t%f\n", i, hist.at<float>(i));
    fprintf(f_stretched_hist, "%d\t%f\n", i, stretched_hist.at<float>(i));
}
```

L만큼 for문을 순회하며 원본 히스토그램 값 hist와 stretching한 히스토그램 값 stretched_hist를 파일에

출력한다.

```
fclose(f_hist);
fclose(f_stretched_hist);
```

사용한 파일을 close한다.

1.5 결과 출력

```
namedWindow("Grayscale", WINDOW_AUTOSIZE);
imshow("Grayscale", input_gray);

namedWindow("Stretched", WINDOW_AUTOSIZE);
imshow("Stretched", stretched);
```

grayscale 이미지와 stretching된 이미지를 별도의 창에 출력한다.

2. linear_stretching(Mat &input, Mat &stretched, G *trans_func, G x1, G x2, G y1, G y2)

linear_stretching 함수는 지정된 구간에 대해 linear stretching을 수행한다. x1과 x2를 기준으로 intensity를 세 구간으로 분할하며, 각 구간에 대해 다른 linear stretching function을 적용해 히스토그램이 stretching되는 정도를 다르게 한다.

2.1 transfer function 계산

```
float constant = (y2 - y1) / (float)(x2 - x1);

for (int i = 0; i < L; i++) {
    ...
}
```

(x1, y1)에서 (x2, y2)까지의 기울기를 계산해 constant에 저장한다. 이후 for문으로 반복하며 모든 지점에서 trans_func[i] 값을 계산한다.

```
if (i >= 0 && i <= x1)
    trans_func[i] = (G)(y1 / x1 * i);
```

intensity가 0 이상 x1 이하일 때는 $trans_func[i] = \frac{y_1}{x_1}i$ 가 적용된다.

```
else if (i > x1 && i <= x2)
    trans_func[i] = (G)(constant * (i - x1) + y1);
```

intensity가 x1 초과 x2 이하일 때 linear stretching이 가장 강하게 적용되며, $trans_func[i] = \frac{y_2 - y_1}{x_2 - x_1}(i - x_1) + y_1$ 가 적용된다.

```
else
    trans_func[i] = (G)((L - 1 - x2) / (L - 1 - y2) * (i - x2) + y2);
```

intensity가 x2 초과일 경우, $\frac{(L-1)-x_2}{(L-1)-y_2}(i - x_2) + y_2$ 가 적용된다.

2.2 transfer function 적용

```
for (int i = 0; i < input.rows; i++)
    for (int j = 0; j < input.cols; j++)
        stretched.at<G>(i, j) = trans_func[input.at<G>(i, j)];
```

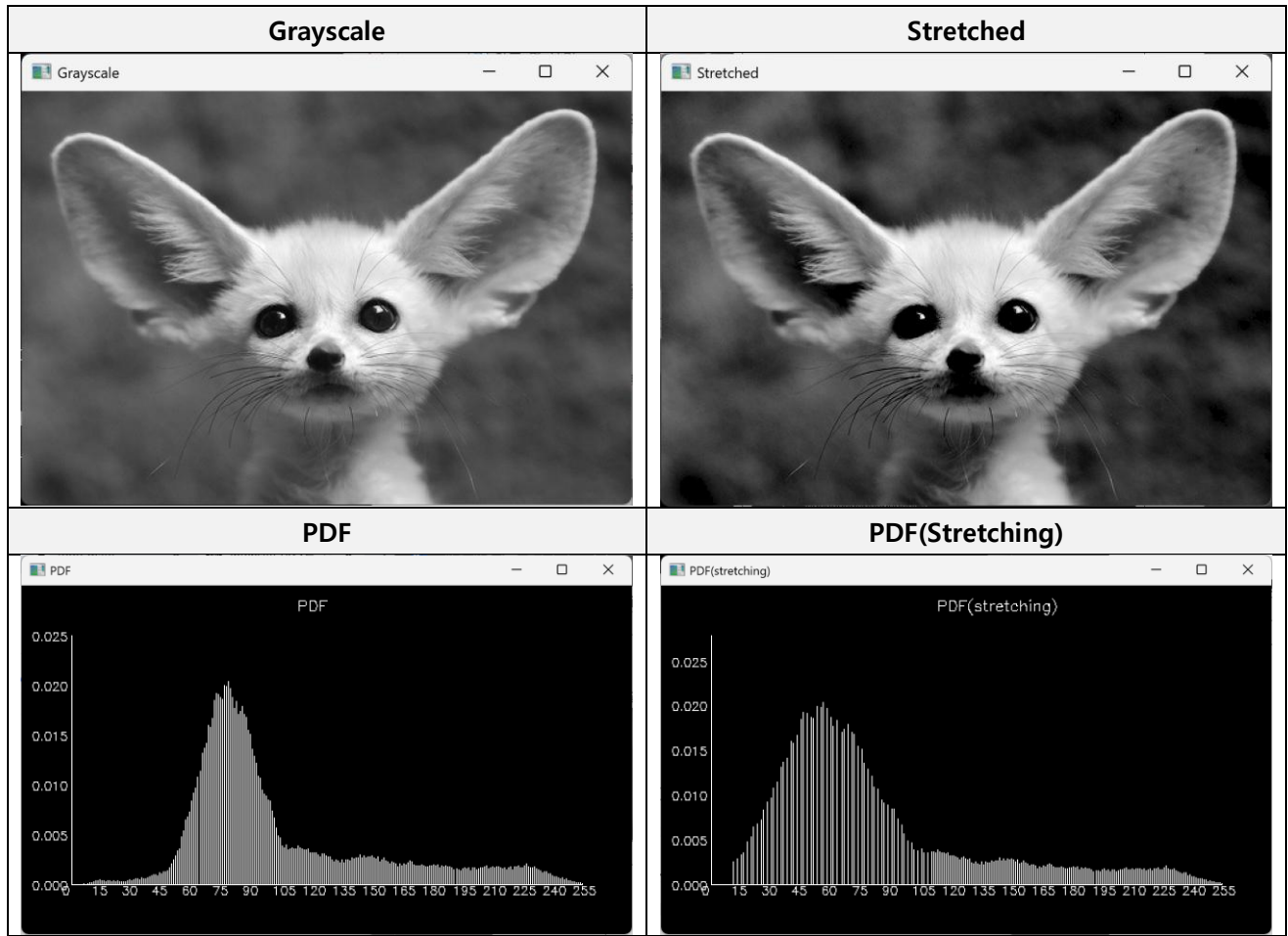
이중 for문을 순회하며, 2.1에서 변경된 intensity를 stretched 이미지에 할당한다.

3. 결과 분석

visual studio IDE에서 OpenCV 라이브러리, C++을 사용해 코드를 컴파일 및 실행하였으며, 입력 이미지로는 "input.jpg"와 "lena.jpg" 2개를 사용하였다.

3.1 input.jpg

입력으로 "input.jpg"를 사용하고, `linear_stretching(input_gray, stretched, trans_func_stretch, 50, 110, 10, 110)`을 호출했다.



실행 결과 원본 이미지와 stretching한 이미지, 각각의 PDF 그래프가 별개의 창에 출력되었다. 원본의 (50~110) 구간 값들이 (10~110) 구간에 넓게 분포하게 되었으며, 그 결과로 이미지의 대비가 더욱 강해진 것을 확인할 수 있다.

PDF.txt	stretched_PDF.txt	trans_func_stretch.txt	hist.txt	stretched_hist.t
<pre> 0 0.000024 1 0.000018 2 0.000030 3 0.000018 4 0.000018 5 0.000042 6 0.000066 7 0.000042 8 0.000138 9 0.000186 10 0.000162 11 0.000301 12 0.000355 13 0.000379 14 0.000463 </pre>	<pre> 0 0.027850 1 0.000000 2 0.000000 3 0.000000 4 0.000000 5 0.000000 6 0.000000 7 0.000000 8 0.000000 9 0.000000 10 0.000000 11 0.002543 12 0.000000 13 0.002898 14 0.000000 </pre>	<pre> 0 0 1 0 2 0 3 0 4 0 5 0 6 0 7 0 8 0 9 0 10 0 11 0 12 0 13 0 14 0 </pre>	<pre> 0 4.000000 1 3.000000 2 5.000000 3 3.000000 4 3.000000 5 7.000000 6 11.000000 7 7.000000 8 23.000000 9 31.000000 10 27.000000 11 50.000000 12 59.000000 13 63.000000 14 77.000000 </pre>	<pre> 0 4632.000000 1 0.000000 2 0.000000 3 0.000000 4 0.000000 5 0.000000 6 0.000000 7 0.000000 8 0.000000 9 0.000000 10 0.000000 11 423.000000 12 0.000000 13 482.000000 14 0.000000 </pre>

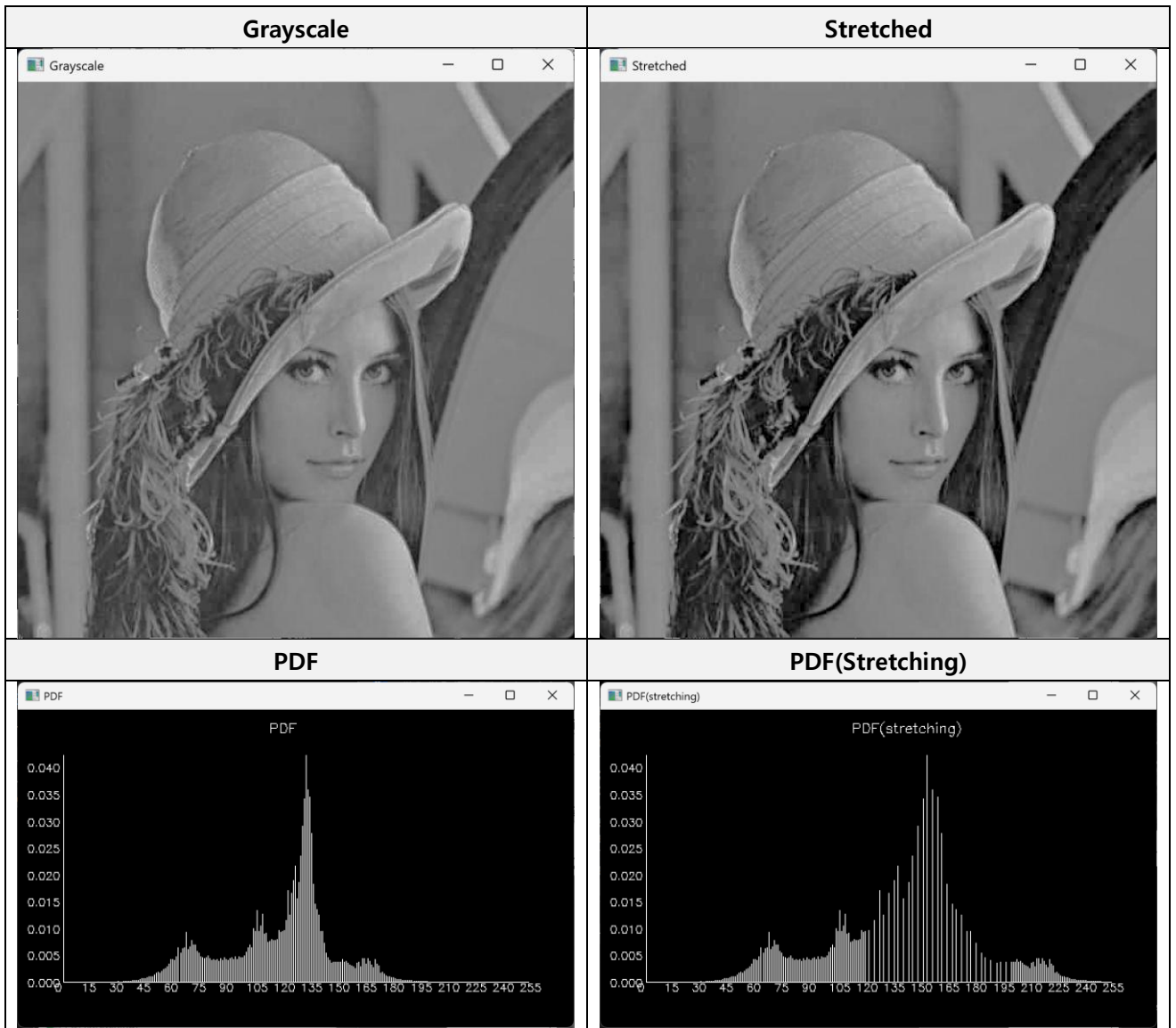
이미지의 PDF, 히스토그램, transfer function 데이터 텍스트 파일이 생성되었으며, `trans_func_stretch.txt`에 $x_1=50, x_2=110$ 을 기준으로

$$trans_func[i] = \frac{y_1}{x_1}i, \quad trans_func[i] = \frac{y_2-y_1}{x_2-x_1}(i-x_1) + y_1, \quad trans_func[i] = \frac{(L-1)-x_2}{(L-1)-y_2}(i-x_2) + y_2$$

가 각각 저장되었다.

3.2 lena.jpg

"lena.jpg"를 사용해 프로그램을 테스트했으며, `linear_stretching(input_gray, stretched, trans_func_stretch, 120, 150, 120, 200)`을 호출해 (120, 150) 구간의 값들을 (120, 200)으로 확장했다.



실행 결과 각 이미지와 그래프가 별도의 창에 정상적으로 출력되었으며, 입력한 대로 120~150 사이의 값들이 넓은 구간에 분포하고 있는 것을 확인할 수 있다. 마찬가지로 stretching된 이미지가 원본 이미지보다 강한 대비를 가진다.

PDF.txt	stretched_PDF.txt	trans_func_stretch.txt	hist.txt	stretched_hist.t
<pre> 0 0.000000 1 0.000000 2 0.000000 3 0.000000 4 0.000000 5 0.000000 6 0.000000 7 0.000000 8 0.000000 9 0.000000 10 0.000000 11 0.000000 12 0.000000 13 0.000000 14 0.000000 </pre>	<pre> 0 0.000034 1 0.000019 2 0.000015 3 0.000008 4 0.000015 5 0.000011 6 0.000011 7 0.000011 8 0.000011 9 0.000011 10 0.000004 11 0.000004 12 0.000004 13 0.000000 14 0.000011 </pre>	<pre> 0 0 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 11 11 12 12 13 13 14 14 </pre>	<pre> 0 0.000000 1 0.000000 2 0.000000 3 0.000000 4 0.000000 5 0.000000 6 0.000000 7 0.000000 8 0.000000 9 0.000000 10 0.000000 11 0.000000 12 0.000000 13 0.000000 14 0.000000 </pre>	<pre> 0 9.000000 1 5.000000 2 4.000000 3 2.000000 4 4.000000 5 3.000000 6 3.000000 7 3.000000 8 3.000000 9 3.000000 10 1.000000 11 1.000000 12 1.000000 13 0.000000 14 3.000000 </pre>

텍스트 파일들 또한 정상적으로 생성되었다.

hist_eq.cpp

hist_eq.cpp는 "input.jpg"에 대해 histogram equalization을 적용하는 프로그램이다. histogram equalization은 이미지의 히스토그램이 최대한 균등하게 분포하도록 하는 것으로, 히스토그램이 특정 영역에 집중되어 있을 때 분포를 균등하게 만들어 contrast를 개선하는 기법이다. 이 과정에서 히스토그램 사이의 간격뿐 아니라 히스토그램 크기(y축)도 변화하므로, 히스토그램 간격만 변화하는 histogram stretching과 차이가 있다.

입력 intensity s 와 변환 후 intensity r 에 대해, equalization의 이상적인 결과는 (1)모든 intensity의 PDF가 uniform한 상태인 경우이, 이때 동일한 intensity를 가진 픽셀들은 equalization 후에도 서로 동일한 intensity를 가지고 있어야 한다. 이를 수식으로 나타내면 아래와 같다.

$$1) \quad PDF_s(s) = \frac{1}{L-1} (L = 256)$$

$$2) \quad PDF_s(s)ds = PDF_r(r)dr$$

이때 transfer function $s = T(r) = (L-1)CDF_r(r)$ 이며, 증명은 다음과 같다.

$$1) \quad \frac{ds}{dr} = \frac{d}{dr}T(r) = \frac{d}{dr}(L-1)CDF_r(r) = (L-1)\frac{d}{dr}CDF_r(r) = (L-1)PDF_r(r)$$

$$2) \quad \frac{ds}{dr} = \frac{d}{dr}T(r) = \frac{d}{dr}(L-1)CDF_r(r) = (L-1)\frac{d}{dr}CDF_r(r) = (L-1)PDF_r(r)$$

$$3) \quad \frac{d}{dr}T(r) = (L-1)PDF_r(r) \text{ 여기서, } dT(r) = (L-1)PDF_r(r)dr$$

$$4) \quad \text{양변을 0에서 } r \text{ 까지 적분하면 } \int_0^r dT(r) = T(r) - T(0) = T(r), \int_0^r (L-1)PDF_r(t)dt = (L-1)CDF_r(r)$$

$$5) \quad T(0) = 0 \text{ 을 사용할 때 } s = T(r) = (L-1)CDF_r(r)$$

따라서 $(L-1)CDF_r(r)$ 을 사용해 equalization 계산을 수행할 수 있다.

1. main()

```
Mat input = imread("input.jpg", IMREAD_COLOR);
Mat input_gray;

cvtColor(input, input_gray, COLOR_RGB2GRAY); // convert RGB to Grayscale

Mat equalized = input_gray.clone();
```

"input.jpg"를 읽어 오고 grayscale로 변환한 것을 input_gray에 저장한 후 이를 복제해 equalized에 저장한다.

```
FILE *f_PDF;
FILE *f_equalized_PDF_gray;
FILE *f_trans_func_eq;

fopen_s(&f_PDF, "PDF.txt", "w+");
fopen_s(&f_equalized_PDF_gray, "equalized_PDF_gray.txt", "w+");
fopen_s(&f_trans_func_eq, "trans_func_eq.txt", "w+");
```

f_PDF, f_equalized_PDF_gray, f_trans_func_eq 파일 포인터를 사용해 원본 PDF 데이터와 equalization한 PDF, transfer function 데이터를 저장할 파일 "PDF.txt", "equalized_PDF_gray.txt", "trans_func_eq.txt"을 생성한다.

```
float *PDF = cal_PDF(input_gray); // PDF of Input image(Grayscale) : [L]
float *CDF = cal_CDF(input_gray); // CDF of Input image(Grayscale) : [L]
```

cal_PDF(), cal_CDF() 함수를 호출해 이미지의 PDF와 CDF 정보를 추출한다.

```
G trans_func_eq[L] = { 0 }; // transfer function
```

```
hist_eq(input_gray, equalized, trans_func_eq, CDF); // histogram equalization on grayscale
float *equalized_PDF_gray = cal_PDF(equalized); // equalized PDF (grayscale)
```

hist_eq() 함수를 호출해 histogram equalization을 수행하고, 이 과정에서 $s = T(r) = (L - 1)CDF_r(r)$ 공식을 사용한다. 이후 equalization한 이미지의 PDF를 계산해 equalized_PDF_gray에 저장한다.

```
for (int i = 0; i < L; i++) {
    // write PDF
    fprintf(f_PDF, "%d\t%f\n", i, PDF[i]);
    fprintf(f_equalized_PDF_gray, "%d\t%f\n", i, equalized_PDF_gray[i]);

    // write transfer functions
    fprintf(f_trans_func_eq, "%d\t%d\n", i, trans_func_eq[i]);
}
```

계산한 데이터를 텍스트 파일에 기록한다.

```
plot(PDF, "PDF", "PDF");
plot(equalized_PDF_gray, "PDF", "PDF(Hist_eq)");

// memory release
free(PDF);
free(CDF);
fclose(f_PDF);
fclose(f_equalized_PDF_gray);
fclose(f_trans_func_eq);

////////// Show each image //////////

namedWindow("Grayscale", WINDOW_AUTOSIZE);
imshow("Grayscale", input_gray);

namedWindow("Equalized", WINDOW_AUTOSIZE);
imshow("Equalized", equalized)
```

원본과 equalization한 이미지의 PDF 그래프를 생성해 출력하고, 메모리 할당 해제 및 파일 닫기 후에 원본과 결과 이미지를 별도의 창에 출력한다.

2. hist_eq(Mat &input, Mat &equalized, G *trans_func, float *CDF)

```
// compute transfer function
for (int i = 0; i < L; i++)
    trans_func[i] = (G)((L - 1) * CDF[i]);
```

각 intensity i 에 대해 $(L - 1)CDF_i(i)$ 를 계산하고, 그 결과를 trans_func 배열에 저장한다.

```
// perform the transfer function
for (int i = 0; i < input.rows; i++)
    for (int j = 0; j < input.cols; j++)
        equalized.at<G>(i, j) = trans_func[input.at<G>(i, j)];
```

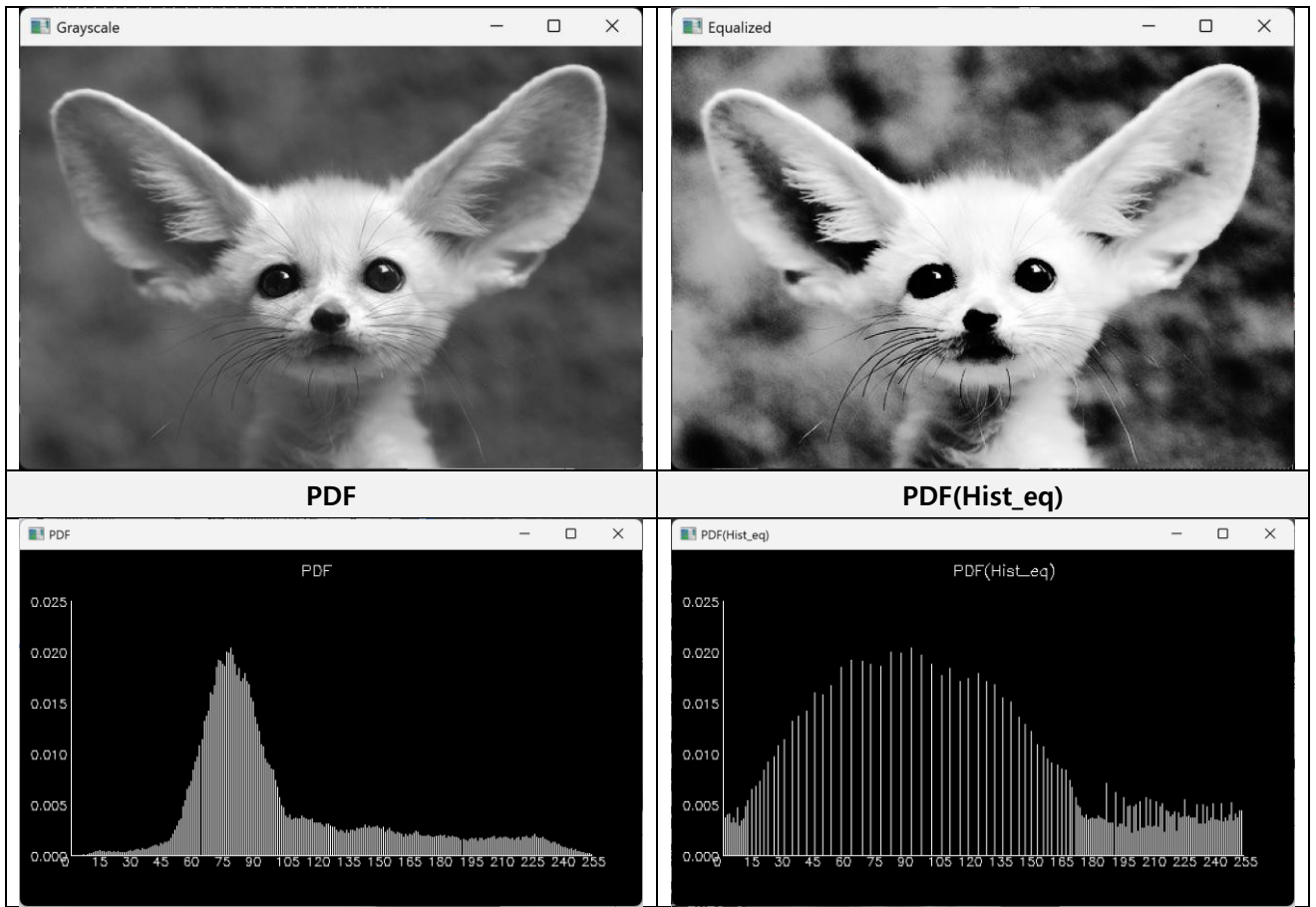
원본 이미지 (i, j) 픽셀의 intensity를 변환한 값을 equalized 이미지 (i, j) 픽셀의 intensity로 설정한다.

3. 결과 분석

visual studio IDE에서 OpenCV 라이브러리, C++을 사용해 코드를 컴파일 및 실행하였으며, 입력 이미지로는 "input.jpg"와 "apples.jpg" 2개를 사용하였다. histogram equalization은 사용자가 매개변수를 조정하지 않아도 되는 fully automatic한 프로그램이므로, 두 경우에서 입력 이미지를 변경하는 것 이외의 다른 조건은 동일하다.

3.1 input.jpg

Grayscale	Equalized
-----------	-----------



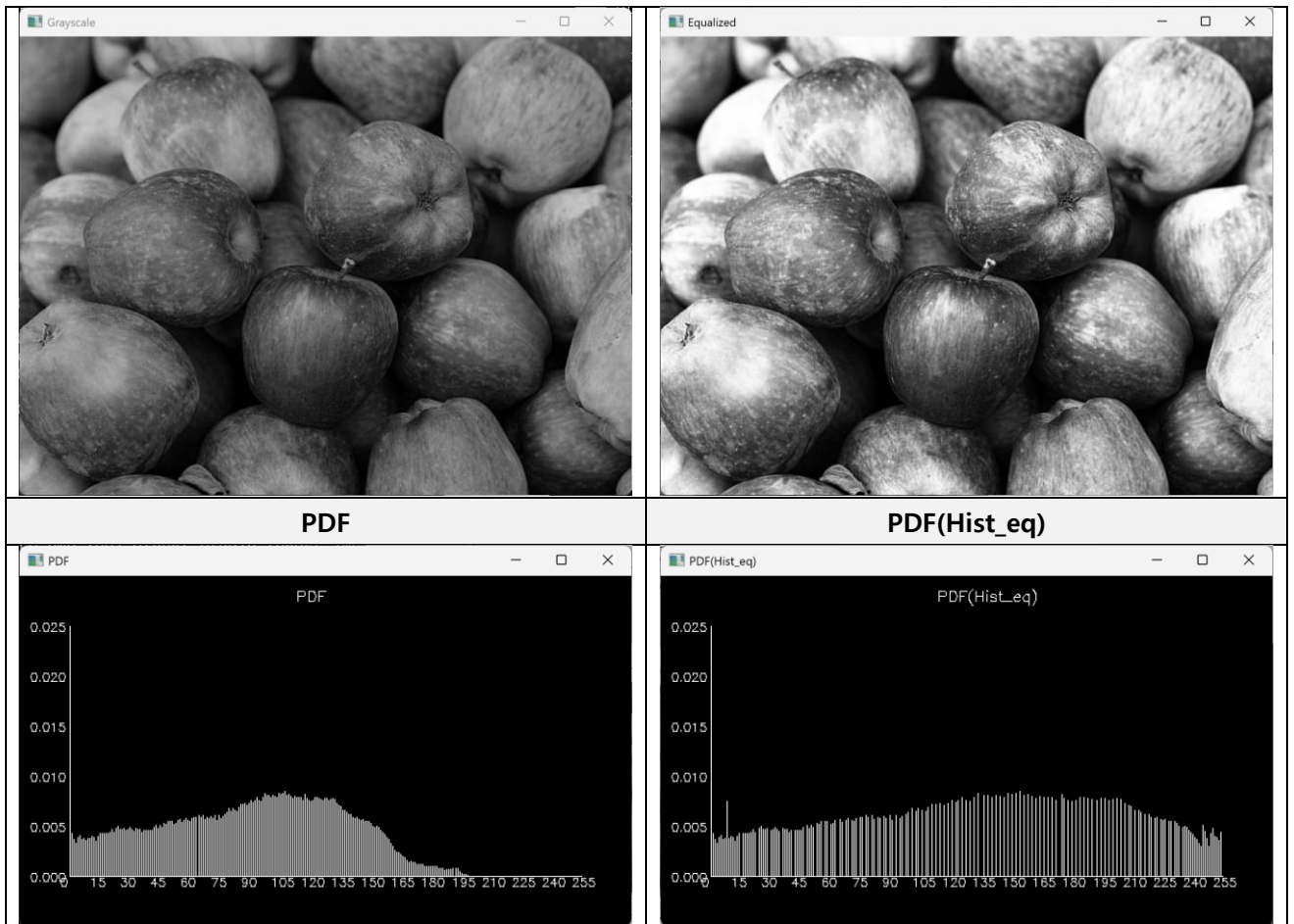
실행 결과 0~50, 110~255 구간의 PDF 값이 증가하여 원본보다 결과 이미지의 PDF가 균등하게 분포하고 있으며, Equalized 이미지의 contrast가 강해진 것을 확인할 수 있다.

PDF.txt	trans_func_eq.txt	equalized_PDF_gray.txt
<div> <div>PDF.txt</div> <div> <div>파일</div> <div>편집</div> <div>보기</div> </div> <div> <div>줄 26, 열 12 3,218자</div> <div>100%</div> <div>Window</div> <div>UT</div> </div> </div> <pre> 0 0.000000 1 0.000000 2 0.000000 3 0.000000 4 0.000000 5 0.000000 6 0.000000 7 0.000004 </pre>	<div> <div>trans_func_eq.txt</div> <div> <div>파일</div> <div>편집</div> <div>보기</div> </div> <div> <div>줄 1, 열 1 1,802자</div> <div>100%</div> <div>Window</div> <div>UT</div> </div> </div> <pre> 0 0 1 0 2 0 3 0 4 0 5 0 6 0 7 0 </pre>	<div> <div>equalized_PDF_gray.txt</div> <div> <div>파일</div> <div>편집</div> <div>보기</div> </div> <div> <div>줄 1, 열 1 3,218자</div> <div>100%</div> <div>Window</div> <div>UT</div> </div> </div> <pre> 0 0.003567 1 0.003242 2 0.003315 3 0.005157 4 0.002689 5 0.003315 6 0.003906 7 0.005085 </pre>

텍스트 파일 또한 정상적으로 생성된다.

3.2 apples.jpg

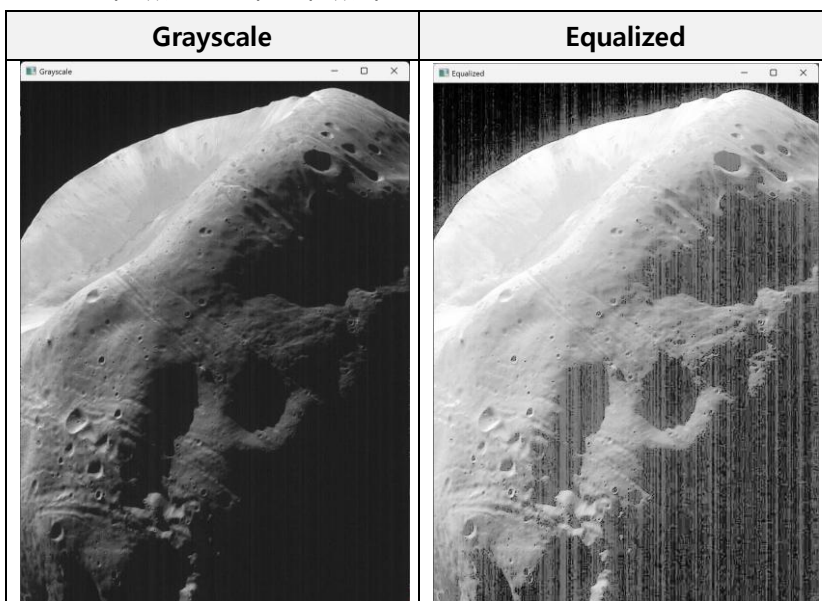
Grayscale	Equalized
-----------	-----------



3.1과 마찬가지로 이미지의 contrast가 개선되고 품질이 향상되었으며, PDF 그래프가 모든 구간에서 보다 균등한 높이로 분포하고 있다.

3.3 equalization 문제점

histogram equalization은 전체 픽셀에 대해 transfer function을 일괄적으로 적용하기 때문에, 아래와 같이 원본 이미지에서 이미 밝은 픽셀이 더욱 밝아지고 이로 인해 디테일이 희석되어 washed-out한 이미지가 생성될 수 있는 문제점이 있다.



hist_eq_RGB.cpp

hist_eq_RGB.cpp는 channel이 3개인 RGB 이미지에 histogram equalization을 적용하는 프로그램이고, 이때 각 채널 R, G, B에 독립적으로 equalization을 수행한다.

1. main()

hist_eq_RGB.cpp에서는 R, G, B에 독립적으로 연산을 수행하므로, PDF/CDF 배열과 transfer function 배열 등을 모두 2차원 배열로 선언해야 한다. 이를 제외하면 hist_eq.cpp의 main() 함수와 유사하다.

```
float **PDF_RGB = cal_PDF_RGB(input); // PDF of Input image(RGB) : [L][3]
float **CDF_RGB = cal_CDF_RGB(input); // CDF of Input image(RGB) : [L][3]
```

이후 이들 정보를 텍스트 파일에 write할 때도 R, G, B 픽셀 값을 각각 기록할 수 있게 형식을 수정한다.

```
vector<Mat> bgr;
split(input, bgr);
```

split() 함수를 사용해 input 이미지의 channel을 분리한 것을 bgr에 저장한다.

```
Mat histB, histG, histR;
calcHist(&bgr[0], 1, 0, Mat(), histB, 1, &histSize, &range); //b
calcHist(&bgr[1], 1, 0, Mat(), histG, 1, &histSize, &range); //g
calcHist(&bgr[2], 1, 0, Mat(), histR, 1, &histSize, &range); //r
```

B, G, R 히스토그램을 각각 저장할 Mat형 histB, histG, histR을 선언하고, calcHist() 함수를 사용해 히스토그램을 계산한다. 각 channel에 대해 계산하기 위해, 인자로 bgr[0], bgr[1], bgr[2]를 전달한다.

```
vector<Mat> bgr_equalized;
split(equalized_RGB, bgr_equalized);

Mat histB_eq, histG_eq, histR_eq;
calcHist(&bgr_equalized[0], 1, 0, Mat(), histB_eq, 1, &histSize, &range);
calcHist(&bgr_equalized[1], 1, 0, Mat(), histG_eq, 1, &histSize, &range);
calcHist(&bgr_equalized[2], 1, 0, Mat(), histR_eq, 1, &histSize, &range);
```

equalization한 이미지에 대해서도 동일한 작업을 수행한다.

```
for (int i = 0; i < histSize; i++) {
    fprintf(f_hist_RGB, "%d\t\t%f\t\t%f\t\t%f\n", i, histR.at<float>(i), histG.at<float>(i),
    histB.at<float>(i));
    fprintf(f_equalized_hist_RGB, "%d\t\t%f\t\t%f\t\t%f\n", i, histR_eq.at<float>(i),
    histG_eq.at<float>(i), histB_eq.at<float>(i));
}
```

for루프를 반복하며 histR, histG, histB와 histR_eq, histG_eq, histB_eq를 텍스트 파일에 기록한다.

2. hist_eq_Color(Mat &input, Mat &equalized, G(*trans_func)[3], float **CDF)

hist_eq_Color() 함수에서는 histogram equalization 공식 $(L - 1)CDF_r(r)$ 을 사용해 RGB 각각의 transfer function을 계산하고, 이를 equalized에 매핑해 결과를 생성한다.

```
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < L; j++) {
        trans_func[j][i] = (G)((L - 1) * CDF[j][i]); // equalization
    }
}
```

컬러 이미지에서 PDF와 CDF는 채널(i) 별로 j를 intensity로 가지는 픽셀의 등장 확률과 누적 합이므로 $trans_func[j][i] = (G)((L - 1) * CDF[j][i])$ 로 계산한다.

```
for (int i = 0; i < input.rows; i++) {
    for (int j = 0; j < input.cols; j++) {
        ...
    }
}
```

이후 2중 for문을 사용해 모든 픽셀에 대해 순회하며, trans_func 배열의 값들을 equalized에 매핑한다.

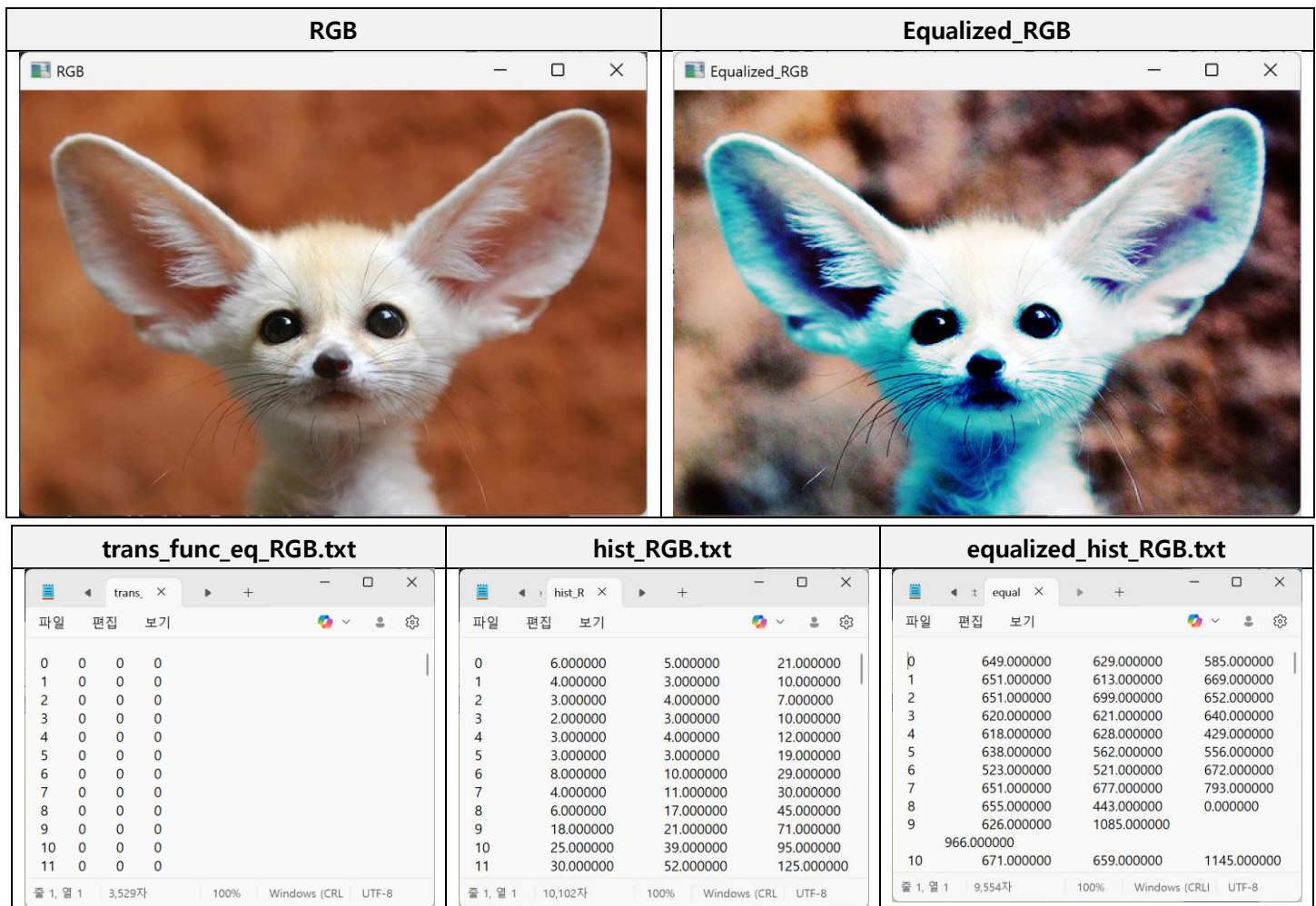
```
Vec3b temp;
```

Vec3b형 temp를 선언한다. temp에는 p의 세 channel을 각각 equalization한 값이 저장된다.

```
for (int k = 0; k < 3; k++) { // RGB
    temp[k] = trans_func[input.at<Vec3b>(i, j)[k]][k];
}
equalized.at<Vec3b>(i, j) = temp;
```

input.at<Vec3b>(i, j)[k]는 (i, j) 픽셀의 BGR 성분 값 중 k channel의 데이터(=intensity)이며, 이것을 trans_func[input.at<Vec3b>(i, j)[k]][k]; 한 것을 temp에 저장한다. 3개의 channel에 대해 temp[0], temp[1], temp[2]를 모두 계산한 후, 내부 for문을 벗어나 temp 값을 equalized에 매핑한다.

3. 결과 분석



원본 이미지와 equalization한 이미지가 별도의 창에 출력되었으며, transfer function과 히스토그램 텍스트 파일도 정상적으로 생성되었다.

hist_eq_RGB.cpp는 별도의 매핑 과정 없이 R, G, B를 독립적으로 계산하였지만, 채널들은 correlated된 관계이므로 Equalized_RGB 이미지와 같이 기존의 분포가 왜곡될 수 있다.

hist_eq_YUV.cpp

hist_eq_YUV.cpp는 RGB를 독립적으로 계산하던 hist_eq_RGB.cpp의 문제를 개선해 컬러 이미지를 equalization하는 프로그램이며, 그 과정에서 YUV 형식을 사용한다. 과정은 다음과 같다.

- 1) RGB를 YUV로 변경
- 2) Y 채널과 U, V 채널을 분리
- 3) Y 채널에 대해서만 histogram equalization 수행
- 4) YUV를 다시 RGB로 복원

RGB 형식은 빛의 삼원색을 사용해 색을 표현하는 반면, YUV 형식은 명암 정보를 나타내는 휘도(Y)와 색상 정보를 나타내는 U, V로 색을 표현하므로 (2), (3)과 같이 Y 채널에 대해서만 equalization을 수행한다.

이와 같은 방법으로 RGB의 correlation 문제를 해결할 수 있으며, 원본 이미지의 색상 왜곡 없이 equalization을 수행할 수 있다.

1. main()

RGB를 YUV 형식으로 변환하고 Y 채널에 대해서만 equalization한다. 그 외 파일 설정과 read/write는 hist_eq_RGB.cpp와 유사하다.

```
cvtColor(input, equalized_YUV, COLOR_RGB2YUV); // RGB -> YUV
```

cvtColor() 함수와 COLOR_RGB2YUV flag를 사용해 input 이미지(RGB)를 YUV 형식으로 변경하고, 이를 equalized_YUV에 저장한다.

```
// split each channel(Y, U, V)
Mat channels[3];
split(equalized_YUV, channels);
Mat Y = channels[0];
```

Y, U, V 채널들을 분리하기 위해 Mat형 channel 배열을 선언하고, split() 함수를 이용해 equalized_YUV의 채널을 분리하여 channels에 저장한 후 Y채널 데이터가 존재하는 channels[0]를 Y에 저장한다. 이때 channels[1]은 채널 U, channels[2]는 채널 V이다.

```
float **PDF_RGB = cal_PDF_RGB(input); // PDF of Input image(RGB) : [L][3]
float *CDF_YUV = cal_CDF(Y); // CDF of Y channel image
```

cal_PDF_RGB()와 cal_CDF() 함수를 사용해 RGB의 PDF와 Y 채널의 CDF를 계산한다.

```
G trans_func_eq_YUV[L] = { 0 }; // transfer function
```

```
// histogram equalization on Y channel
hist_eq(Y, channels[0], trans_func_eq_YUV, CDF_YUV);
```

Y 채널에 대해 equalization한 transfer function을 저장할 배열 trans_func_eq_YUV를 선언하고, hist_eq() 함수에 Y를 전달해 Y 채널을 equalization한다.

```
// merge Y, U, V channels
merge(channels, 3, equalized_YUV);

// YUV -> RGB (use "CV_YUV2RGB" flag)
cvtColor(equalized_YUV, equalized_YUV, COLOR_YUV2RGB);
```

merge() 함수를 호출해 equalization된 Y 채널과 U, V 채널을 병합하고, cvtColor() 함수와 COLOR_YUV2RGB flag를 사용해 equalized_YUV를 RGB 형식으로 변환한다.

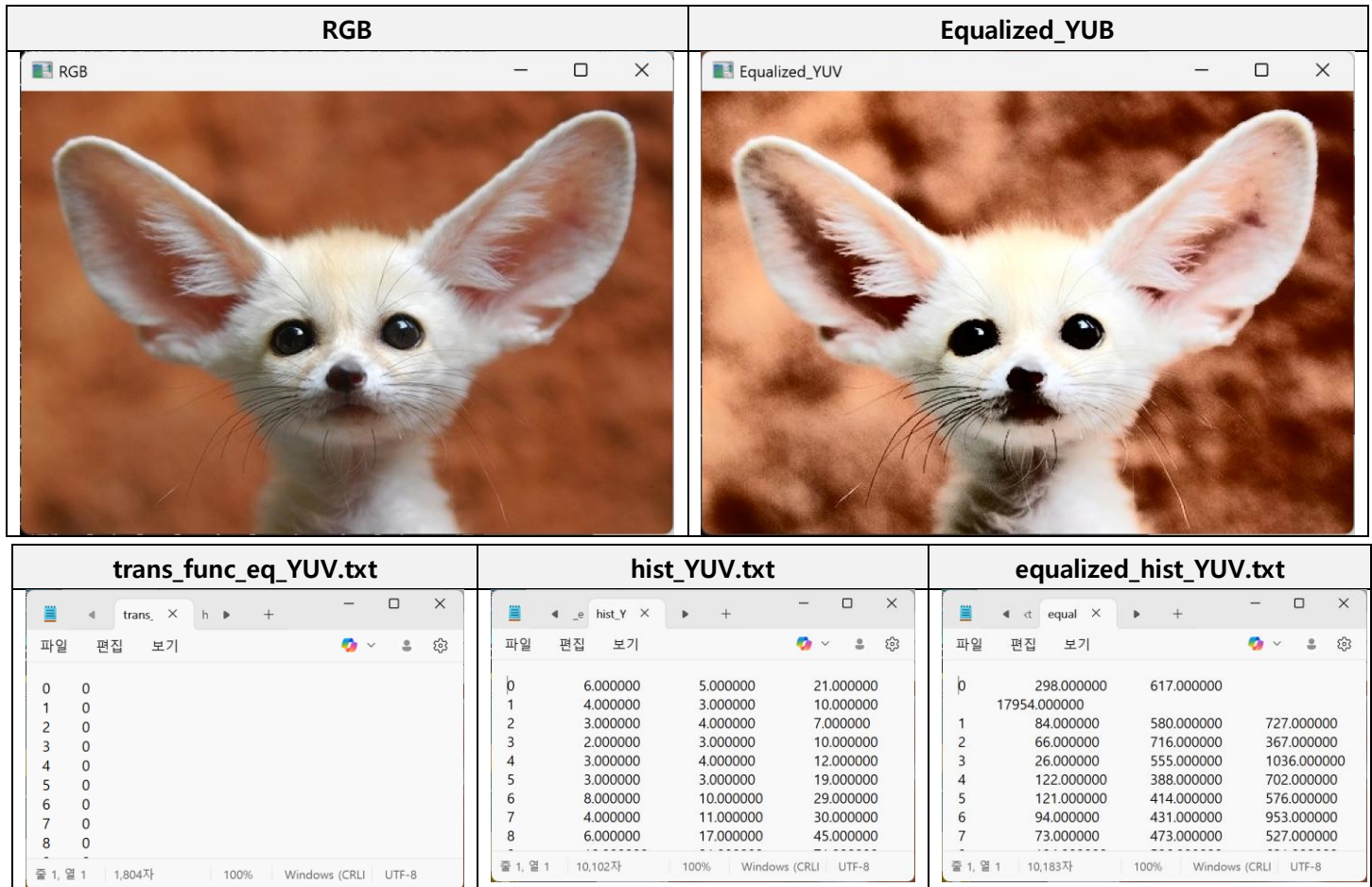
2. void hist_eq(Mat &input, Mat &equalized, G *trans_func, float *CDF)

```
void hist_eq(Mat &input, Mat &equalized, G *trans_func, float *CDF) {
    // compute transfer function
    for (int i = 0; i < L; i++)
        trans_func[i] = (G)((L - 1) * CDF[i]);

    // perform the transfer function
    for (int i = 0; i < input.rows; i++)
        for (int j = 0; j < input.cols; j++)
            equalized.at<G>(i, j) = trans_func[input.at<G>(i, j)];
}
```

단일 채널에 대해 equalization하므로 1차원 배열 trans_func를 사용하며, hist_eq.cpp 코드의 hist_eq()와 동일하다.

3. 결과 분석



원본 이미지와 equalization한 이미지가 별도의 창에 출력되었으며, transfer function, 히스토그램 텍스트 파일도 정상적으로 생성되었다. YUV로 변경하지 않고 RGB 각 채널에 대해 독립적으로 연산했던 hist_eq_RGB.cpp와 달리 원본 이미지의 색상 왜곡 없이 contrast만 개선된 것을 확인할 수 있다.

hist_matching_gray.cpp

hist_matching_gray.cpp는 grayscale 이미지인 input.jpg와 reference.jpg에 대해 히스토그램 matching을 수행하는 프로그램이다.

히스토그램 matching은 입력 이미지의 히스토그램을 reference 이미지의 히스토그램 분포와 유사하게 만드는 것으로, stretching 또는 equalization과 달리 결과 이미지의 히스토그램을 넓게 퍼뜨리거나 균등하게 변경할 필요가 없다. 히스토그램 matching을 하는 방법은 아래와 같다.

- 1) 입력 이미지 I_r 에 대해 equalization을 수행하고, equalization transfer function s 를 저장한다.

$$s = T(r) = (L - 1)CDF_r$$

- 2) reference 이미지 I_z 에 대해 equalization을 수행하고, equalization transfer function s 를 저장한다.

$$s = G(z) = (L - 1)CDF_z$$

- 3) (2)에서 $z = G^{-1}(s)$ 이므로, r 에서 z 로의 intensity mapping 공식은 아래와 같다.

$$z = G^{-1}(s) = G^{-1}(T(r))$$

1. main

```
Mat input = imread("input.jpg", IMREAD_COLOR);
Mat reference = imread("lena.jpg", IMREAD_COLOR);

Mat input_gray, reference_gray;      // Convert to grayscale
cvtColor(input, input_gray, COLOR_RGB2GRAY);
cvtColor(reference, reference_gray, COLOR_RGB2GRAY);
```

입력 이미지와 reference 이미지를 읽어 와 각각 input과 reference에 저장하고, cvtColor() 함수와 COLOR_RGB2GRAY flag를 하용해 grayscale로 변환한다.

```
Mat equalized_input = input_gray.clone();
Mat equalized_ref = reference_gray.clone();

float* CDF_input = cal_CDF(input_gray);
float* CDF_ref = cal_CDF(reference_gray);

G trans_func_input[L] = { 0 };
G trans_func_ref[L] = { 0 };

hist_eq(input_gray, equalized_input, trans_func_input, CDF_input);
hist_eq(reference_gray, equalized_ref, trans_func_ref, CDF_ref);
```

hist_eq() 함수를 호출해 input_gray와 reference_gray로 히스토그램 equalization을 수행한다. hist_eq.cpp 프로그램에서 수행했던 것과 동일하다.

```
Mat matched = input_gray.clone();
G trans_func_z[L] = { 0 };

hist_matching(equalized_input, matched, trans_func_z, trans_func_input, trans_func_ref);
```

matching할 이미지를 저장할 matched와 matching transfer function을 저장할 배열 trans_func_z를 생성한다. 이후 hist_matching() 함수를 호출해 matching을 수행하며, 이때 입력 이미지, reference 이미지의 transfer function을 인자로 전달받아 이들을 inverse mapping한 것을 trans_func_z 배열에 저장하고, 이를 matched에 mapping한다.

이후 원본 이미지, reference 이미지와 matching한 이미지를 별도의 창에 출력하고, matching transfer function,

원본과 결과 이미지의 히스토그램 데이터를 텍스트 파일에 기록한다. 이 과정은 앞선 프로그램들에서 수행했던 것과 동일하다.

2. void hist_eq(Mat& input, Mat& equalized, G* trans_func, float* CDF)

```
void hist_eq(Mat& input, Mat& equalized, G* trans_func, float* CDF) {  
    // compute transfer function  
    for (int i = 0; i < L; i++)  
        trans_func[i] = (G)((L - 1) * CDF[i]);  
  
    // perform the transfer function  
    for (int i = 0; i < input.rows; i++)  
        for (int j = 0; j < input.cols; j++)  
            equalized.at<G>(i, j) = trans_func[input.at<G>(i, j)];  
}
```

입력 이미지 input, equalization한 이미지 equalized, transfer function배열 trans_func, 이미지의 CDF를 인자로 전달받아 히스토그램 equalization을 수행하는 함수이다. hist_eq.cpp의 hist_eq()와 동일하다.

3. hist_matching(Mat& input, Mat& matched, G* trans_func_z, G* trans_func_ref)

hist_matching() 함수는 input 이미지, matching한 이미지를 저장할 matched와 입력 이미지, reference 이미지의 equalization transfer function인 trans_func_input, trans_func_ref를 인자로 전달받아 히스토그램 matching을 수행하고 matching transfer function을 trans_func_z 포인터에 저장하는 함수이다.

$s = T(r) = (L - 1)CDF_r = trans_func_input$ 이고 $s = G(z) = (L - 1)CDF_z = trans_func_ref$ 이므로, trans_func_z는 trans_func_ref의 역함수이다. 따라서 r에서 z로 mapping해야 하지만, 두 함수가 일대일 대응이 아니므로 이 경우 여러 대응값 중 하나를 임의로 선택해야 한다. 이때 equalization transfer function들은 $(L - 1)CDF$ 이고, CDF는 단조 증가 함수이므로 trans_func_ref 또한 단조 증가 함수이다.

이 프로그램에서는 아래와 같이 대응값이 여러 개라면 그 중 가장 작은 값을 선택하도록 구현하였다. intensity가 큰 곳에서부터 탐색하기 위해 별도의 인덱스 변수 idx1과 idx2를 선언하였으며, 각각 254와 255로 초기화한다. 255에서부터 역함수를 탐색함으로써 가장 작은 값을 선택하는 조건을 만족할 수 있다.

```
int idx1 = L - 2;  
int idx2 = L - 1;  
  
while (idx1 >= 0) {  
    for (int i = trans_func_ref[idx1]; i <= trans_func_ref[idx2]; i++) {  
        trans_func_z[i] = idx2;  
    }  
    idx2 = idx1--;  
}
```

인덱스가 범위를 벗어나지 않을 때(idx1>=0)까지 while문을 수행한다. 이후 trans_func_ref[idx1](=G[idx1])로 i를 초기화하고, trans_func_ref[idx2]까지 반복하여 idx1과 idx2 사이의 모든 범위를 탐색한다.

for문을 반복하며 각 i에 대해 trans_func_z[i]에 idx2를 저장하며, 이는 역함수를 단조 증가 함수로 유지하기 위해 해당 범위 내의 높은 인덱스, 즉 가장 작은 값을 선택하기 위함이다.for문이 종료되면 idx1을 1 감소시켜 이를 idx2에 대입함으로써 다음 인덱스에 대해 역함수 값을 찾는다. 이후 후위 연산자를 사용해 idx2를 idx1로 변경한 후 idx1을 1 감소시키며, 다음 반복에서 이전 범위를 탐색한다.




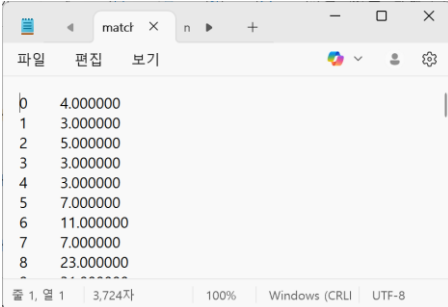
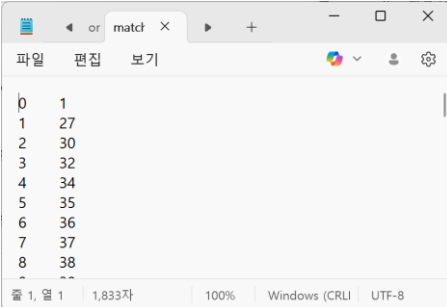
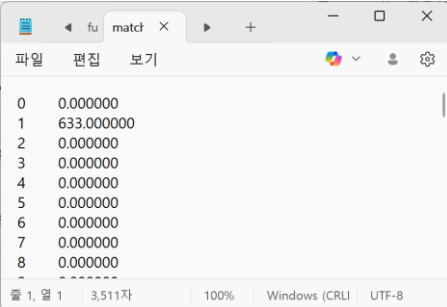
hist_matching() 함수를 이와 같이 구현해 중복되거나 불연속적인 mapping 없이 z의 각 구간마다 하나의 입력 값을 할당할 수 있다.

```
for (int i = 0; i < input.rows; i++)
    for (int j = 0; j < input.cols; j++)
        matched.at<G>(i, j) = trans_func_z[(input.at<G>(i, j))];
```

이후 2중 for문으로 모든 픽셀에 대해 반복하여 trans_func_z 배열에서 (i, j) 픽셀의 intensity와 대응되는 값을 matched의 (i, j) 픽셀에 저장한다.

4. 결과 분석

입력 이미지 input으로는 "input.jpg"를, reference 이미지 reference로는 "lena.jpg"를 사용하였다. histogram matching은 histogram equalization 코드에 기반하고 있으므로, hist_eq.cpp와 마찬가지로 입력 이미지 이외에 추가로 파라미터를 조정할 필요는 없다.

Input	Reference	Matched
		
matching_original_hist_gray.txt	matching_function_gray.txt	matched_hist_gray.txt
		

실행 결과 input 이미지, reference 이미지, matching한 결과 이미지가 별도의 창에 출력되었으며, 원본과 결과 이미지의 히스토그램, matching transfer function 텍스트 파일 또한 정상적으로 생성되었다.

```
float* input_PDF = cal_PDF(input_gray);
float* ref_PDF = cal_PDF(reference_gray);
float* matched_PDF = cal_PDF(matched);

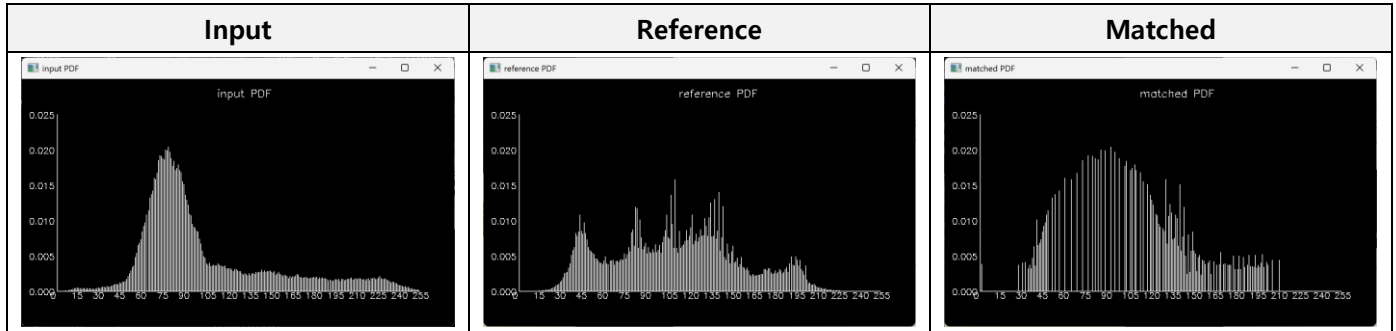
plot(input_PDF, "PDF", "input PDF");
plot(ref_PDF, "PDF", "reference PDF");
plot(matched_PDF, "PDF", "matched PDF");

free(input_PDF);
free(matched_PDF);
```



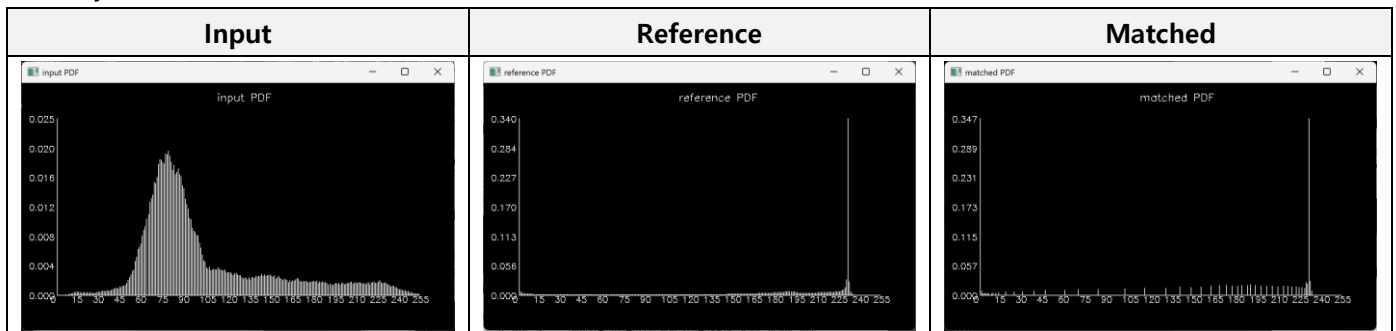
```
free(ref_PDF);
```

위와 같이 cal_PDF()와 plot() 함수를 사용해 PDF 그래프를 작성하는 코드를 추가하여 PDF 변화를 확인하였다.



실행 결과 이미지 그래프의 분포, intensity 별 PDF값이 변화하였다.

이후 input 이미지 PDF가 reference 이미지 PDF와 비슷하게 변화하는지 확인하기 위해 픽셀의 대부분이 high intensity를 가지는 극단적인 reference 이미지를 사용해 다시 한번 실행하였으며, 결과는 아래와 같다.



matched PDF가 reference PDF와 유사한 분포를 가지게 되어 대부분의 PDF 값이 intensity가 높은 구간에 분포하는 것을 확인할 수 있다.

hist_matching_color.cpp

hist_matching_color.cpp는 컬러 이미지에 대해 히스토그램 matching을 수행하는 함수이다. 프로그램의 전체적인 흐름은 grayscale matching인 hist_matching_gray.cpp와 유사하지만, 컬러 이미지를 다루고 있으므로 hist_eq_YUV와 같이 RGB space를 YUV space로 변환한 후 Y 채널에 대해서만 matching하는 과정이 추가된다.

1. main

```
Mat input = imread("input.jpg", IMREAD_COLOR);
Mat reference = imread("lena.jpg", IMREAD_COLOR);

Mat equalized_input, equalized_ref; // YUV
cvtColor(input, equalized_input, COLOR_RGB2YUV);
cvtColor(reference, equalized_ref, COLOR_RGB2YUV);
```

grayscale matching과 동일하게 읽어 온 이미지들을 input과 reference에 저장하지만, cvtColor() 함수와 COLOR_RGB2YUV flag를 사용해 RGB를 YUV로 변환한 것을 equalized_input과 equalized_ref에 저장한다.

```
Mat channels_input[3], channels_ref[3];
split(equalized_input, channels_input);
split(equalized_ref, channels_ref);

Mat Y_input = channels_input[0];
Mat Y_ref = channels_ref[0];
```

split() 함수를 이용해 equalized_input과 equalized_ref의 채널을 분리하고, 이를 channels_input과 channels_ref 배열에 각각 저장한다. 이후 각 channels 배열의 인덱스 0값인 Y 채널을 Y_input과 Y_ref에 저장한다.

```
float* CDF_input = cal_CDF(Y_input);
float* CDF_ref = cal_CDF(Y_ref);
```

transfer function 계산에 사용할 CDF 데이터를 추출하되, 각 이미지의 Y채널에 대해서만 계산한다.

```
G trans_func_input[L] = { 0 };
G trans_func_ref[L] = { 0 };

hist_eq(Y_input, channels_input[0], trans_func_input, CDF_input);
hist_eq(Y_ref, channels_ref[0], trans_func_ref, CDF_ref);
```

이후 입력 이미지와 reference 이미지의 equalization transfer function을 저장할 배열을 선언하고, 이를 hist_eq() 함수에 전달해 equalization transfer function을 계산한다.

```
Mat matched = equalized_input.clone();
Mat channels_matched[3];
split(matched, channels_matched);

G trans_func_z[L] = { 0 };

hist_matching(channels_input[0], channels_matched[0], trans_func_z, trans_func_ref);
```

matching한 이미지를 저장할 matched를 선언하고, equalized_input을 복제한 것으로 초기화한다. matched도 마찬가지로 channels_matched 배열과 split() 함수를 사용해 채널을 분리하였으며, matching transfer function을 저장할 배열을 선언하고 이들을 hist_matching 함수에 전달해 히스토그램 matching을 수행한다.

```
merge(channels_matched, 3, matched);

cvtColor(matched, matched, COLOR_YUV2RGB);

free(CDF_input);
```

```
free(CDF_ref);
```

merge() 함수를 사용해 matching한 Y채널과 U, V 채널을 병합하고, cvtColor() 함수와 COLOR_YUV2RGB flag를 사용해 YUV로 변환되었던 matched 이미지를 다시 RGB로 변환한다. 이후 CDF 계산에 사용했던 CDF 포인터들을 할당 해제한다.

다른 프로그램들과 마찬가지로 transfer function과 원본, 결과 이미지의 히스토그램을 작성하고 각 이미지들을 별도의 창에 출력하며, 이때 hist_eq_RGB.cpp에 서술한 것과 동일한 방식으로 RGB 히스토그램을 추출해 텍스트 파일로 작성한다.

2. hist_eq(Mat& input, Mat& equalized, G* trans_func, float* CDF)

```
void hist_eq(Mat& input, Mat& equalized, G* trans_func, float* CDF) {  
    // compute transfer function  
    for (int i = 0; i < L; i++)  
        trans_func[i] = (G)((L - 1) * CDF[i]);  
  
    // perform the transfer function  
    for (int i = 0; i < input.rows; i++)  
        for (int j = 0; j < input.cols; j++)  
            equalized.at<G>(i, j) = trans_func[input.at<G>(i, j)];  
}
```

입력 이미지 input, equalization한 이미지 equalized, transfer function배열 trans_func, 이미지의 CDF를 인자로 전달받아 히스토그램 equalization을 수행하는 함수이다. hist_eq.cpp의 hist_eq()와 동일하다.

3. hist_matching(Mat& input, Mat& matched, G* trans_func_z, G* trans_func_ref)

전달받은 input 이미지와 transfer function들을 사용하여 r에서 z로 inverse mapping 값을 matching transfer function 배열에 저장하고, 이를 matched 이미지에 다시 할당하는 함수이다.

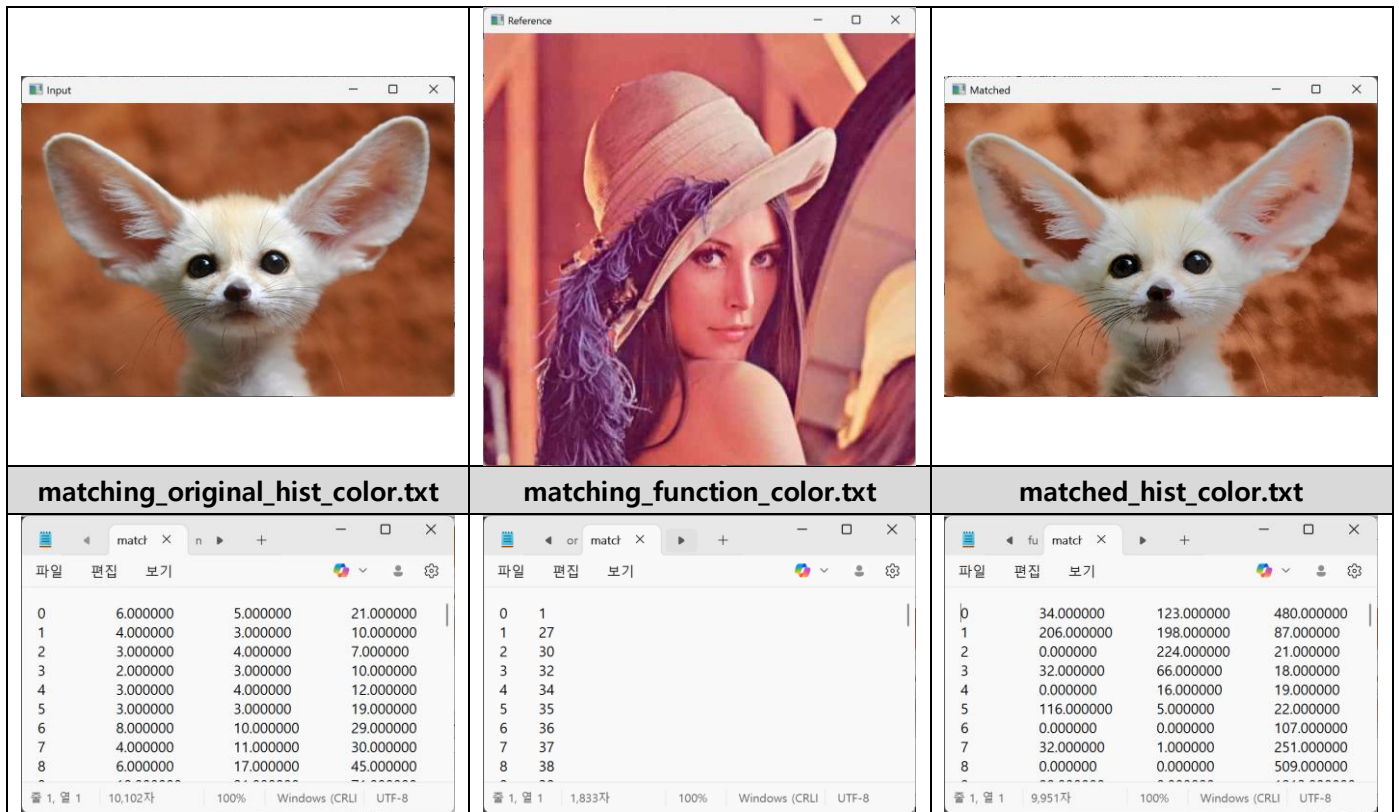
```
void hist_matching(Mat& input, Mat& matched, G* trans_func_z, G* trans_func_ref) {  
    int idx1 = L - 2;  
    int idx2 = L - 1;  
  
    while (idx1 >= 0) {  
        for (int i = trans_func_ref[idx1]; i <= trans_func_ref[idx2]; i++) {  
            trans_func_z[i] = idx2;  
        }  
        idx2 = idx1--;  
    }  
  
    for (int i = 0; i < input.rows; i++)  
        for (int j = 0; j < input.cols; j++)  
            matched.at<G>(i, j) = trans_func_z[input.at<G>(i, j)];  
}
```

hist_matching_gray.cpp의 hist_matching() 함수와 동일하다.

4. 결과 분석

입력 이미지로 "input.jpg"를, reference 이미지로 "lena.jpg"를 사용하였다.

Input	Reference	Matched
-------	-----------	---------



실행 결과 Matched 이미지의 색상 분포가 변화한 것을 확인할 수 있으며, transfer function, original 히스토그램, reference 히스토그램의 텍스트 파일 또한 정상적으로 생성된 것을 확인할 수 있다.