1. hist_func.h

코드 목적:

이번 프로젝트에서 공통적으로 사용되는 헤더파일, 변수, 함수들을 정의한다.

PDF, CDF 를 구하는 함수를 포함하고 있기에 다른 모든 코드에서 이 함수를 불러온다.

설명:

1) cal_PDF (Mat &input)

매개변수: input: 입력 이미지 행렬의 주소값

함수 목적: input 이미지의 PDF 를 리턴한다.

PDF 는 0 부터 255 까지에 해당하는 색 데이터에 해당하는 픽셀의 개수를 전체 픽셀수로 나눈 것으로, 각 색이 이미지를 구성하는 비율을 나타낸다.

```
int count[L] = { 0 };
```

count[L] 배열에 색 L 을 가지는 픽셀의 개수를 저장한다.

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

이미지의 모든 픽셀을 돌며 (i,j)의 색 데이터를 인덱스로 하는 count 배열의 요소에 +1 한다. 즉, 색 L 이 나오면 count[]의 L 번 요소에 +1 을 하며 count 를 진행한다.

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

PDF 는 전체에 대한 해당 색의 비율이므로, count 값을 전체 필셀 수로 나눈다.

2) cal_PDF_RGB (Mat &input)

매개변수: input: 입력 이미지 행렬의 주소값. 이때 이미지는 컬러이다.

함수 목적: 컬러 이미지의 PDF 를 리턴한다

컬러 이미지는 R,G,B 세 채널을 이용해 이미지를 표현한다. PDF 를 구할때도 R,G,B 따로따로 픽셀을 세고, PDF 를 계산해주어야 한다.

```
int count[L][3] = { 0 };
```

OpenCV 는 색을 BGR 순서로 표현한다. 인덱스 0 은 파랑, 1 은 초록, 2 는 빨강 채널이다.
k 채널의 데이터가 L 일 때, count[L][k]에 +1 을 할 것이다.

```
for (int i = 0; i < input.rows; i++){
        for (int j = 0; j < input.cols; j++){
            count[input.at<Vec3b>(i, j)[0]][0]++;
            count[input.at<Vec3b>(i, j)[1]][1]++;
            count[input.at<Vec3b>(i, j)[2]][2]++;
    }
```

방식은 (1)과 같지만, 컬러이므로 input 이미지 (i,j) 픽셀의 [0], [1], [2]에 대해 따로 따로 count 한다.

```
for (int i = 0; i < L; i++){
        PDF[i][0] = (float)count[i][0] / (float)(input.rows * input.cols);
        PDF[i][1] = (float)count[i][1] / (float)(input.rows * input.cols);
        PDF[i][2] = (float)count[i][2] / (float)(input.rows * input.cols);
}
```

PDF 계산 또한 따로따로 해준다. 즉, R G B 각각의 전체 PDF 의 합이 1 이다. 서로 다른 세 데이터라고 생각하면 된다.

3) cal_CDF (Mat &input)

매개변수: input: 입력 이미지 행렬의 주소값

함수 목적: input 이미지의 CDF 를 리턴한다.

CDF 는 누적분포함수이다. CDF[i]는 0 부터 i 까지 모든 PDF 값을 더해주면 된다.
우선 PDF 를 위해 색상 별 픽셀 수를 구한다. 이는 (1)의 과정과 크게 다르지 않다.

```
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];
      }
```

i 색을 count 한 픽셀을 전체로 나눠 PDF 를 구한 뒤 이전 CDF[i-1]에 PDF 를 더해 CDF[i]를 구한다.

4) cal_CDF_RGB (Mat &input)

매개변수: input: 입력 이미지 행렬의 주소값. 이때 이미지는 컬러이다.

함수 목적: 컬러 이미지의 PDF 를 리턴한다

(2)에서, (1)과 같은 작업을 하되, 각 R G B 에 대해 따로 계산해준 것처럼, (3)의 과정을 RGB 를 고려하며 따라가주면 된다.

```
for (int i = 0; i < input.rows; i++){
  for (int j = 0; j < input.cols; j++){
    count[input.at<Vec3b>(i, j)[0]][0]++; count[input.at<Vec3b>(i, j)[1]][1]++;  count[input.at<Vec3b>(i, j)[2]][2]++;
  }
}
for (int i = 0; i < L; i++){
  CDF[i][0] = (float)count[i][0] / (float)(input.rows * input.cols);
  CDF[i][1] = (float)count[i][1] / (float)(input.rows * input.cols);
  CDF[i][2] = (float)count[i][2] / (float)(input.rows * input.cols);
  if (i != 0){
    CDF[i][0] += CDF[i - 1][0];  CDF[i][1] += CDF[i - 1][1];  CDF[i][2] += CDF[i - 1][2];
  }
}
```

2. PDF_CDF.cpp

코드 목적: 흑백 이미지를 입력받아 PDF 와 CDF 를 계산하고 계산 결과를 출력하는 텍스트 파일을 생성한다.

코드 흐름:

1. 이미지 파일을 읽고 흑백으로 전환한다. 텍스트 파일을 연다. (준비)
2. cal_PDF(), cal_CDF()를 호출해 PDF 와 CDF 를 구한다.
3. 리턴받은 계산 결과를 텍스트 파일에 적는다.
4. 텍스트 파일 닫는다. input 이미지와 흑백으로 전환한 이미지 창을 띄운다.(마무리)

설명:

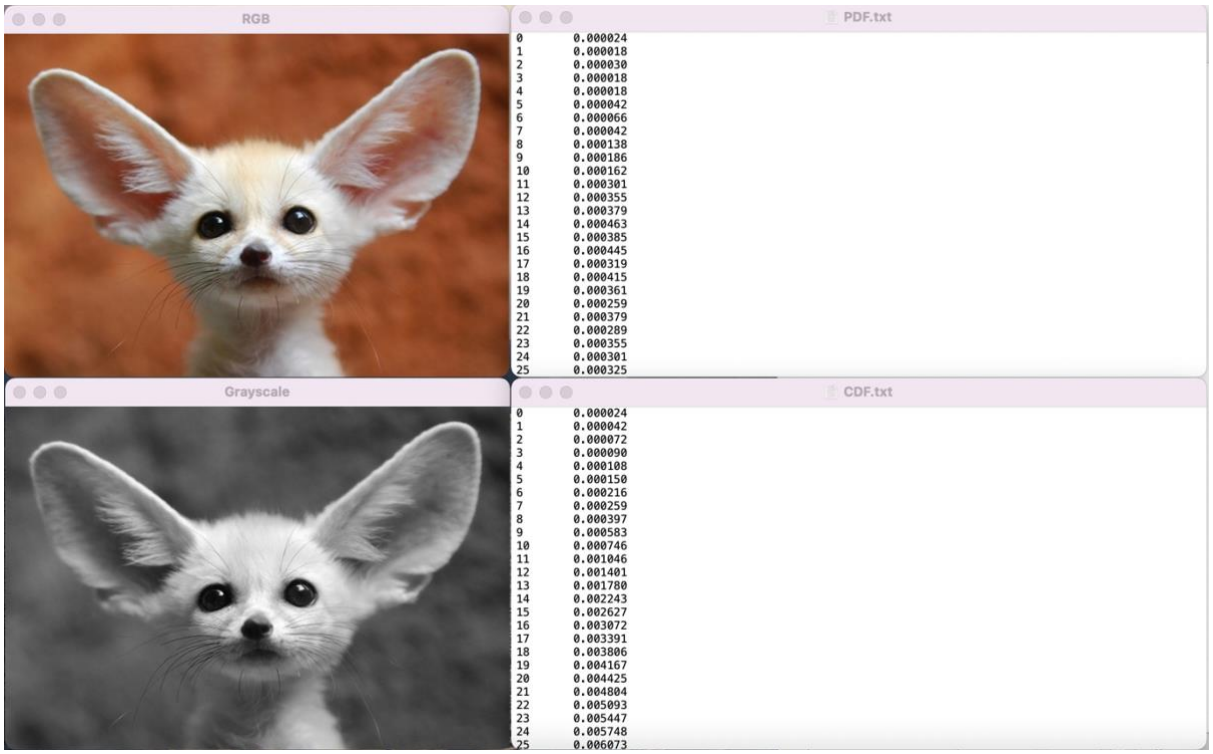
```
cvtColor(input, input_gray, CV_RGB2GRAY);
```

input 이미지를 "CV_RGB2GRAY"flag 를 이용해 흑백으로 전환해 input_gray 에 넣는다.

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

input_gray(흑백 이미지)를 헤더 파일 hist_function.h 에 있는 함수의 매개변수로 보내 PDF 와 CDF 를 구한다.

실행 결과:



3. hist_stretching.cpp

코드 목적:

linear strectching function 을 통해 histogram stretching 한 결과 이미지를 새로운 창으로 띄우고, 입력 이미지의 PDF, streching 한 이미지의 PDF, transfer function(mapping 되는 값)을 쓴 텍스트파일을 생성한다.

코드 흐름:

1. 이미지를 입력받아 흑백으로 전환한다. 텍스트파일을 생성한다.
2. linear_stretching()함수를 호출해 histogram stretching한다.
3. input 이미지의 PDF, stretched 이미지의 PDF, transfer function을 파일에 적는다.
4. 입력 이미지와 histogram stretching 완료한 이미지를 새로운 창에 띄운다

함수 설명:

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

매개변수:

input: 입력 이미지. 흑백이다

stretched: histogram stretching 의 결과를 저장할 Mat

trans_func: mapping 될 색 저장할 배열

x1,x2: pixel 몰린 구간

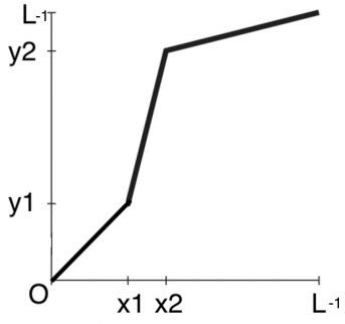
y1,y2: pixel 을 집중적으로 나눠주고싶은 구간

함수 목적: input 이미지를 histogram stretching 해 stretched 와 trans_func 에 결과 값을 넣어준다

코드 설명:

현재 이미지의 색 데이터 x 를 y 로 mapping 시켜줄 것이다.

픽셀이 많이 모여있는 x1~x2 구간의 색을 y1~y2 로 늘려주면 contrast 가 증가한 이미지가 될 것이다.



o~x1 구간: 기울기 $\frac{y1}{x1}$ 이고 (0,0)를 지나는 일차 함수-> y=(y1/x1)*x

x1~x2 구간: 기울기 $\frac{y2-y1}{x2-x1}$이고 (x1,y1)를 지나는 일차 함수-> $y = \frac{y2-y1}{x2-x1}(x-x1)+y1$

x2~L-1 구간: 기울기 $\frac{L-1-y2}{L-1-x2}$ 이고 (x2,y2)를 지나는 일차 함수-> $y = \frac{L-1-y2}{L-1-x2}(x-x2)+y2$

```
for (int i = 0; i < L; i++) {
        if (i >= 0 && i <= x1)
            trans_func[i] = (G)(y1 / x1 * i);
        else if (i > x1 && i <= x2)
            trans_func[i] = (G)(constant * (i - x1) + y1);
        else
            trans_func[i] = (G)((L - 1 - x2) / (L - 1 - y2) * (i - x2) + y2);
    }
```

```
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)];
}
```

input 의 모든 픽셀에 접근해 얻은 색 데이터 x 를 transfer function 에 넣어 mapping 되는 y 를 찾는다. stretched 의 (i,j)픽셀에 y 를 넣어준다.

실행 결과:

4. hist_eq.cpp

코드 목적:

Histogram equalization 을 통해 이미지의 contrast 를 증가시킨다. 결과 이미지를 새로운 창으로 띄우고, 입력 이미지의 PDF, equalization 한 이미지의 PDF, PDF, transfer function(mapping 되는 값)을 쓴 텍스트파일을 생성한다.

코드 흐름:

1. 이미지를 입력받아 흑백으로 전환한다. 텍스트파일을 생성한다.
2. hist_eq()함수를 호출해 Histogram equalization한다.
3. input 이미지의 PDF, equalization한 이미지의 PDF, transfer function을 파일에 적는다.
4. 입력 이미지와 Histogram equalization완료한 이미지를 새로운 창에 띄운다

함수 설명:

매개변수: hist_eq(Mat &input, Mat &equalized, G *trans_func, float *CDF)

input: 입력 이미지. 흑백이다

equalized: Histogram equalization 의 결과를 저장할 Mat

trans_func: mapping 될 색 저장할 배열

CDF: 입력 이미지의 CDF

함수 목적: 입력받은 이미지에 대해 histogram equalization 수행

코드 설명:

hist_eq():

입력 이미지의 색 데이터를 r 이라고 하면, Histogram equalization 을 하는 식은 다음과 같다.

$T(r) = (L-1) * CDF(r)$

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

CDF[i] 배열에 i 색상의 CDF 가 들어있으므로, 위의 식 T(r)을 모든 색에 대해 적용해준다.

trans_func[i]에는 색상 i 가 Histogram equalization 될 때, 어떤 색에 mapping 되는지 적혀있다.

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

input 의 모든 픽셀에 접근해 얻은 색 데이터 x 를 transfer function 에 넣어 mapping 되는 y 를 찾는다. equalizedop[의 (i,j)픽셀에 y 를 넣어준다.

실행 결과:

5. hist_eq_RGB.cpp

코드 목적:

Histogram equalization 을 통해 컬러 이미지의 contrast 를 증가시킨다. 결과 이미지를 새로운 창으로 띄우고, 입력 이미지의 PDF, equalization 한 이미지의 PDF, PDF, transfer function(mapping 되는 값)을 R,G,B 에 대해 각각 모두 적은 텍스트파일을 생성한다.

코드 흐름:

1. 이미지를 입력받는다. (컬러) 텍스트파일을 생성한다.
2. hist_eq_RGB()함수를 호출해 Histogram equalization한다.
3. input 이미지의 PDF, equalization한 이미지의 PDF, transfer function을 파일에 적는다.
4. 입력 이미지와 Histogram equalization완료한 이미지를 새로운 창에 띄운다

함수 설명: hist_eq_Color(Mat &input, Mat &equalized, G(*trans_func)[3], float **CDF)

매개변수:

input: 입력 이미지. 흑백이다

equalized: Histogram equalization 의 결과를 저장할 Mat

trans_func: mapping 될 색 저장할 배열

CDF: 입력 이미지의 CDF

함수 목적: RGB(컬러) 이미지 데이터에 대해 Histogram Equalization 을 진행한다

코드 설명:

코드의 흐름과 역할은 4 번 코드와 같다. 이 코드에서는 컬러 이미지를 다루기 때문에, 4 번에서 흑백 데이터에 한 작업을 컬러이미지의 R,G,B 에 대해 각각 해주면 된다.
변수가 R, G, B 에 대한 정보를 모두 저장할 수 있어야 하기에 자료형의 크기가 달라진다.

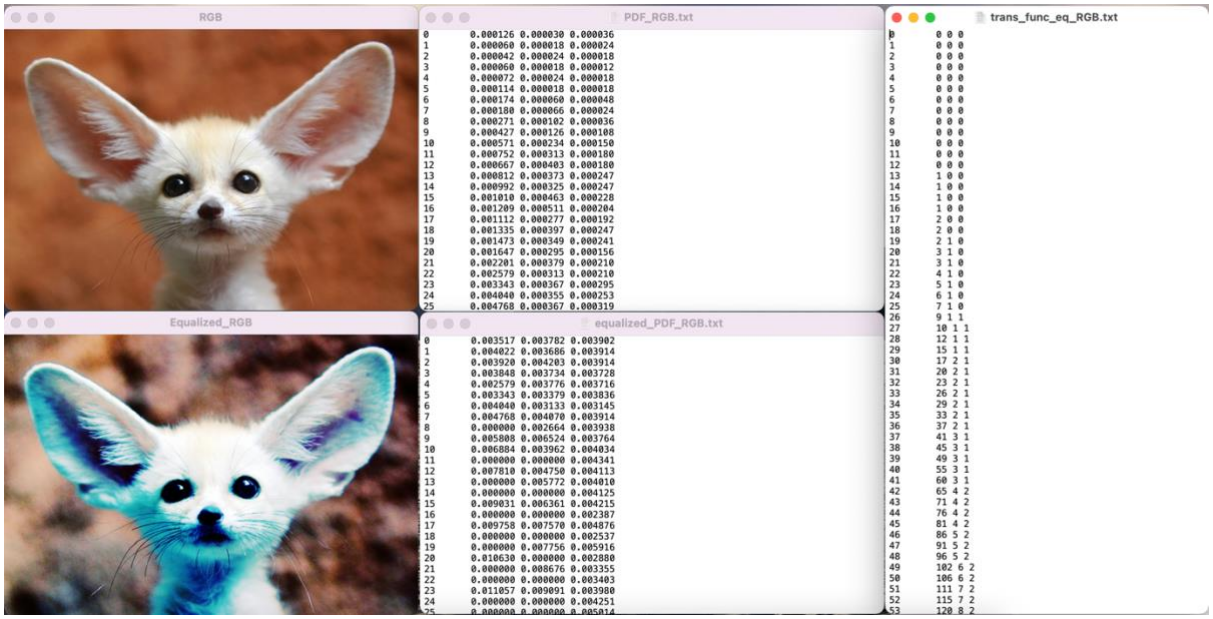
```
float **PDF_RGB = cal_PDF_RGB(input);
float **CDF_RGB = cal_CDF_RGB(input);
G trans_func_eq_RGB[L][3] = { 0 };
```

PDF 와 CDF 는 기존에 1 차원 배열로, 인덱스에 해당하는 색 데이터의 비율을 나타냈는데, 컬러 이미지에서는 2 차원배열로 표현해, [a][b]인 경우, b 채널 의 a 데이터의 비율을 저장하게 된다.
transfer function 는 trans_func_eq_RGB[L][b]=k 인 경우, b 채널의 값이 L 일 때, k 로 mapping 된다는 의미이다.

따라서, histogram equalization 을 각 R,G,B 에 대해 따로따로 계산해준다

```
for (int i = 0; i < L; i++){
   trans_func[i][0] = (G)((L - 1) * CDF[i][0]);   trans_func[i][1] = (G)((L - 1) * CDF[i][1]);   trans_func[i][2] = (G)((L - 1) *
CDF[i][2]);
}
for (int i = 0; i < input.rows; i++){
   for (int j = 0; j < input.cols; j++){
                equalized.at<Vec3b>(i, j)[0] = trans_func[input.at<Vec3b>(i, j)[0]][0];
                equalized.at<Vec3b>(i, j)[1] = trans_func[input.at<Vec3b>(i, j)[1]][1];
                equalized.at<Vec3b>(i, j)[2] = trans_func[input.at<Vec3b>(i, j)[2]][2];
   }
}
```

실행 결과:



R, G, B 의 값을 equal 하게 바꾼것 뿐인데, 사막여우의 색이 어색하게 보이는 것은, R, G, B 가 독립적인 값이 아니라, 서로 correlative 한 데이터기 때문이다. 각각의 색을 따로 계산하는 순간 correlation 관계가 무너지는 것이다

6. hist_eq_YUV.cpp

코드 목적:

컬러 이미지의 Histogram Equalization 을 진행한다. RGB 의 이미지 데이터를 YUV 로 변환한 뒤, Y 값에 대해서만 Histogram Equalization 해준다. 결과 이미지를 새로운 창으로 띄우고, 입력 이미지의 PDF, equalization 한 이미지의 PDF, Y 에 대한 transfer function(mapping 되는 값)을 적은 텍스트파일을 생성한다.

코드 흐름:

1. 컬러 이미지를 입력받는다.
2. RGB의 데이터를 YUV로 변환한다.
3. 텍스트 파일을 연다.
4. hist_eq()함수를 호출해 Y채널에 대해 Histogram Equalization을 진행한다.
5. YUV의 이미지를 다시 RGB로 바꿔준다.
6. 텍스트 파일에 데이터를 적는다.
7. 입력 이미지와 Histogram Equalization완료한 이미지를 새로운 창에 띄운다.

함수 설명: hist_eq(Mat &input, Mat &equalized, G *trans_func, float *CDF)

매개변수:

input: Histogram Equalization 할 데이터. 여기선 Y 채널의 값만이 전달될 것이다.

equalized: Histogram equalization 의 결과를 저장할 Mat

trans_func: Y 가 mapping 될 값 저장할 배열

CDF: Histogram Equalization 할 데이터의 CDF

함수 목적: 채널 개수가 1 개인 데이터에 대해 Histogram Equalization 을 진행한다.

코드 설명:

```
cvtColor(input, equalized_YUV, CV_RGB2YUV);
Mat channels[3];
split(equalized_YUV, channels);
Mat Y = channels[0];
```

input 이미지를 "CV_RGB2YUV"flag 를 이용해 YUV 데이터로 전환해 equalized_YUV 에 넣는다. split()함수를 이용해 equalized_YUV 를 channels[3]에 채널별로 분리해 저장한다. Y 채널의 데이터는 channels[0], U 채널의 데이터는 channels[1], V 채널의 데이터는 channels[2] 에 저장되는 것이다.

```
float *CDF_YUV = cal_CDF(Y);                    // CDF of Y channel image
```

cal_CDF 는 하나의 채널의 데이터의 CDF 를 구해주는 함수이다. Y 채널의 값만을 보내 Y 의 CDF 를 구한다.

```
hist_eq(Y,channels[0],trans_func_eq_YUV,CDF_YUV);
```

U, V 는 그대로이고, Y 의 값만 equalization 하면 되므로, Y 의 데이터와 Y 의 CDF 를 매개변수로 보낸다.

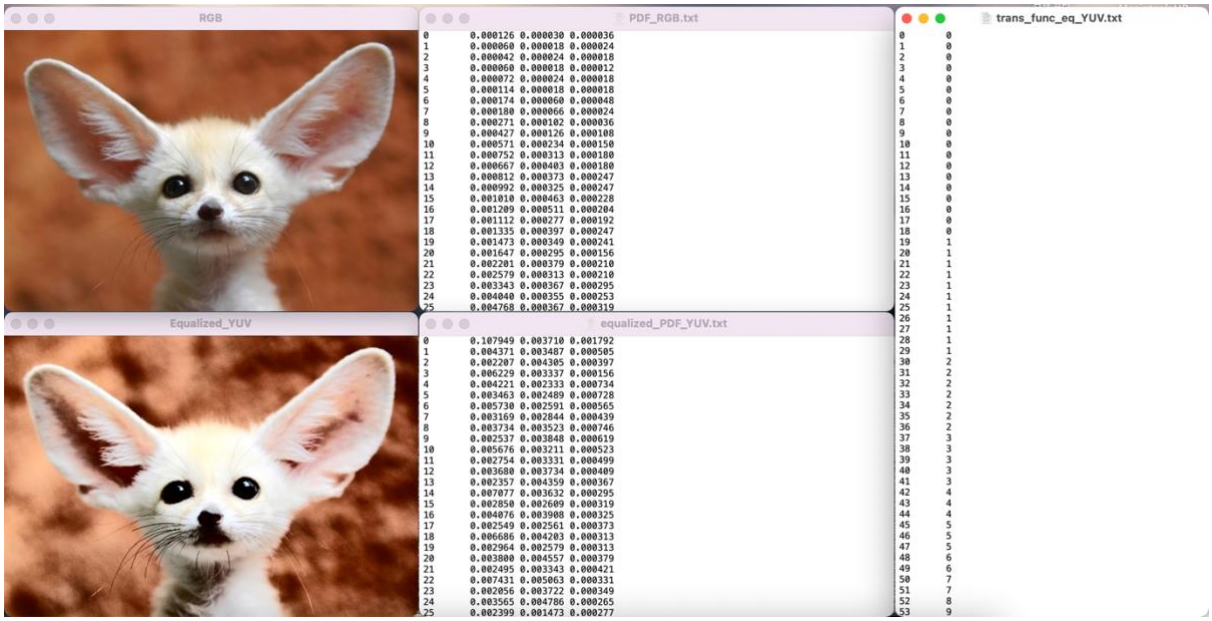
```
void hist_eq(Mat &input, Mat &equalized, G *trans_func, float *CDF) {

  for (int i = 0; i < L; i++)
    trans_func[i] = (G)((L - 1) * CDF[i]);

  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)];
}
```

Y 한 채널에 대해서만 equalization 을 진행하므로, 흑백의 데이터에 대해 equalization 을 진행하는 것과 똑같은 상황이다. hist_eq.cpp 의 hist_eq()함수와 똑같다.

```
merge(channels, 3, equalized_YUV);
cvtColor(equalized_YUV,equalized_YUV, CV_YUV2RGB);
```

Y 에만 적용하는 계산을 마쳤으므로 Y, U, V 의 데이터를 담은 channels 의 세 요소를 merge()함수를 이용해 equalized_YUV 에 합쳐준다. cvtColor 함수를 통해 이미지를 다시 RGB 로 바꿔준다.

실행 결과:



7. hist_hm_YUV.cpp

코드 목적:

컬러 이미지의 Histogram Matching 을 진행한다. RGB 의 이미지 데이터를 YUV 로 변환한 뒤, Y 값에 대해서만 Histogram Matching 해준다. 결과 이미지를 새로운 창으로 띄우고, 입력 이미지의 PDF, matching 한 이미지의 PDF, Y 에 대한 transfer function(mapping 되는 값)을 적은 텍스트파일을 생성한다.

코드 흐름:

1. input 이미지와 reference 이미지를 입력받는다.
2. RGB의 데이터를 YUV로 변환한다.
3. 텍스트 파일을 연다.
4. hist_ma() 함수를 호출해 Y채널에 대해 Histogram Matching을 진행한다.
5. YUV의 이미지를 다시 RGB로 바꿔준다.
6. 텍스트 파일에 데이터를 적는다.
7. 입력 이미지와 레퍼런스 이미지, HM완료한 이미지를 새로운 창에 띄운다.

함수 설명:

hist_ma(Mat &input, Mat &reference, Mat &equalized, G *trans_func, float *CDF,float *CDF_reference)

매개변수:

input: HM 할 데이터. 여기선 Y 채널의 값만이 전달될 것이다.
reference: HM 의 reference 이미지
equalized: HM 의 결과를 저장할 Mat
trans_func: Y 가 mapping 될 값 저장할 배열
CDF: HM 할 데이터의 CDF
CDF_reference: 레퍼런스 이미지의 CDF

함수 목적: input 이미지를 reference 이미지의 히스토그램에 가까워지도록 Histogram Matching 을 수행한다

코드 설명:

Histogram Matching 은 input r을 HE 한 $s = T(r)$을 별도의 reference image $s = G(z)$에 가까운 색분포를 가지게 하는 것이 목표로, input r, output z에 대해 $z = G^{-1}(s) = G^{-1}(T(r))$을 구하면 된다. 그 과정과 구현은 다음과 같다.

첫번째, input 이미지를 HE 한 결과를 구한다. $s = T(r) = (L-1) * \mathrm{CDF}$

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

두번째, 함수 $s = G(z) = (L-1) * \mathrm{CDF}$를 구한다.

```
for (int i = 0; i < L; i++){
        trans_func_G=(G)((L - 1) * CDF_reference[i]);
        trans_func_G_reverse[trans_func_G]=i;
}
```

$s = (L-1) * \mathrm{CDF}[z]$ 로, (z, s)순서쌍을 구했다.

세번째, $z = G^{-1}(s)$를 구한다.

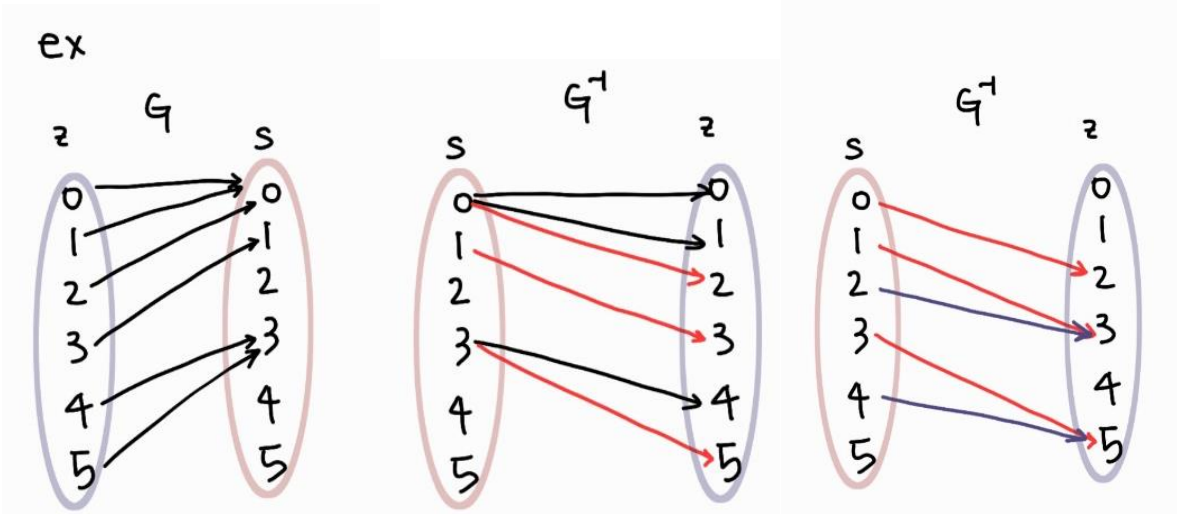
```
for (int i = 0; i < L; i++){
        trans_func_G=(G)((L - 1) * CDF_reference[i]);
        trans_func_G_reverse[trans_func_G]=i;
}
```

HM 을 위해서는 G 의 역함수가 필요하므로 trans_func_G_reverse[] 배열에 (s, z) 순서쌍을 저장해준다.

$z = G^{-1}(s)$: trans_func_G_reverse[]의 인덱스 s 로 z 에 접근이 가능한 것이다.

```
for (int i = 0; i < L; i++){
   if(trans_func_G_reverse[i]==0){
                trans_func_G_reverse[i]=trans_func_G_reverse[i-1];
}
```

G 가 일대일 대응이 아니므로, 역함수가 온전하지 못하다. 연결이 없는 부분을 이전 인덱스의 값으로 채워준다.

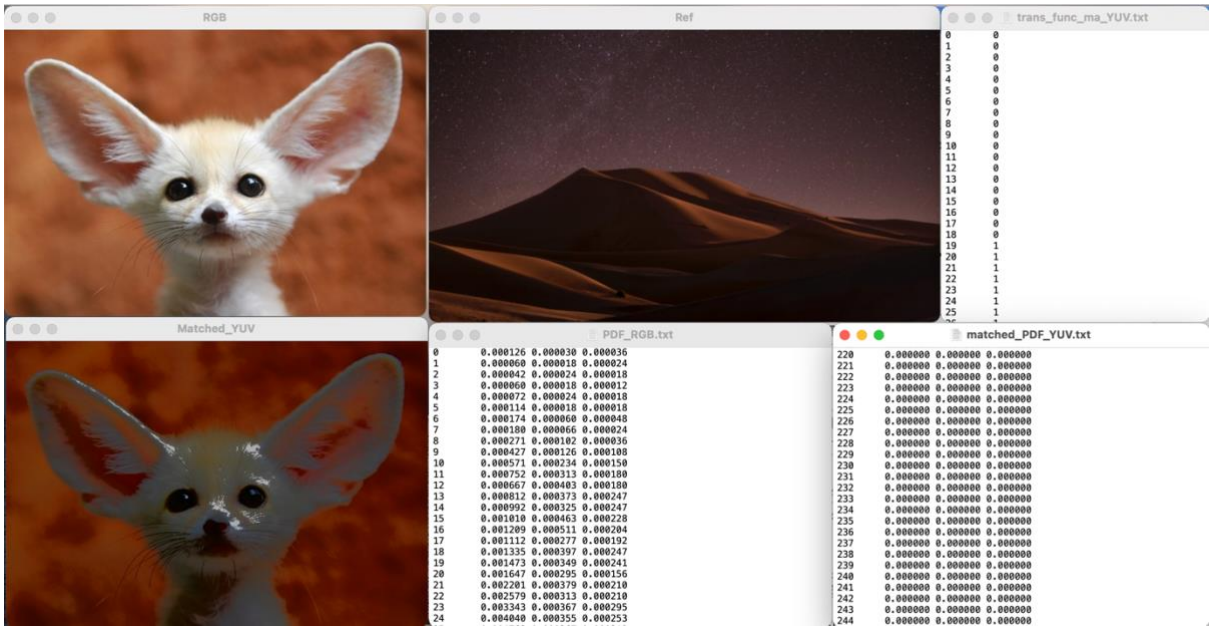


네번째, 첫번째에서 구한 $s = T(r)$를 $z = G^{-1}(s)$에 넣어준다.

```
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)]; //s=T(r)
            equalized.at<G>(i, j) = trans_func_G_reverse[equalized.at<G>(i, j)]; //s=G^-1(s)
        }
}
```

실행 결과:

사막여우를 밤 사막과 같은 색감으로 만들고 싶다면 reference 로 밤 사막 이미지를 "reference.jpeg"의 이름으로 저장하면 된다.



참고자료:

오픈 SW 프로젝트 Lec03 수업자료