

영상처리 실습과제1

16010811

김범수

Image & Histogram






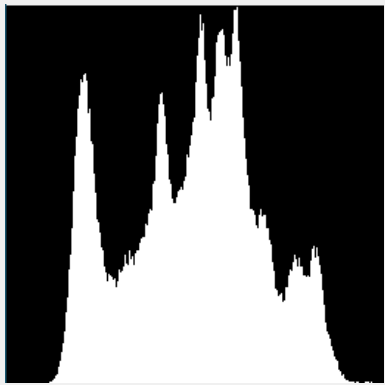
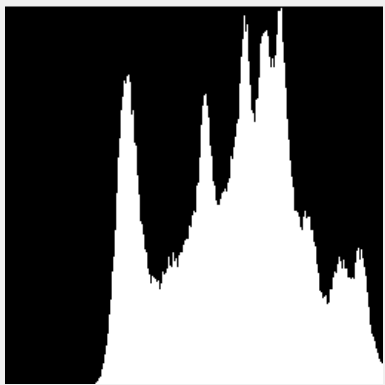
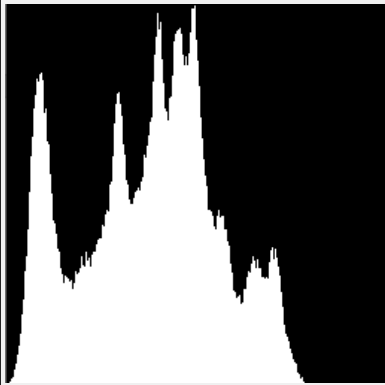
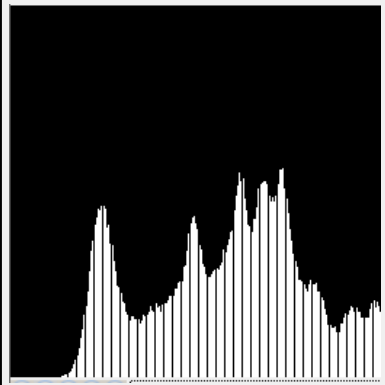
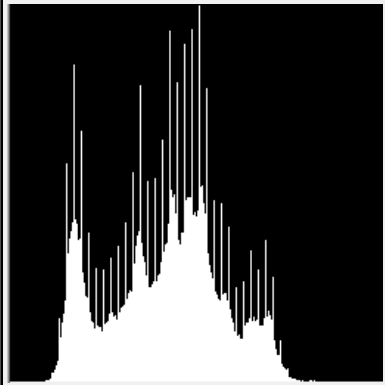





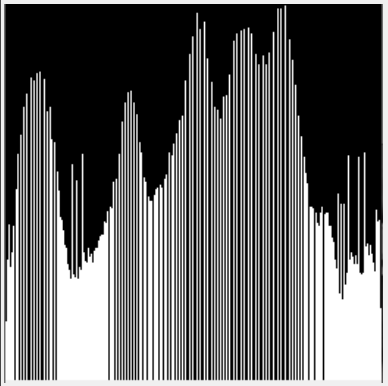
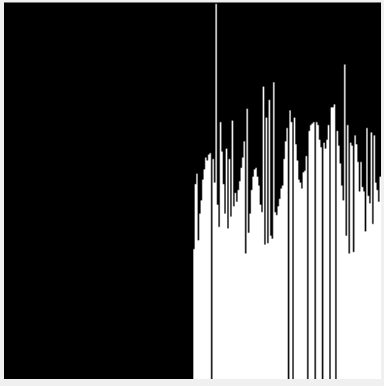
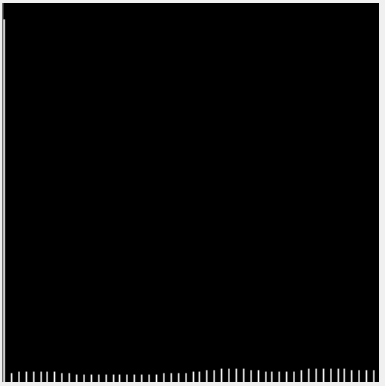
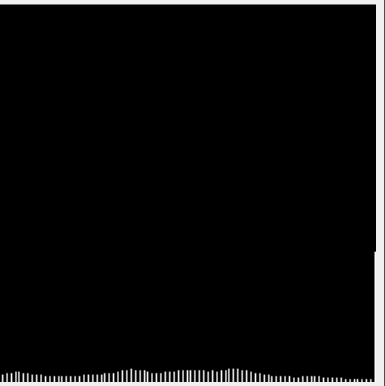
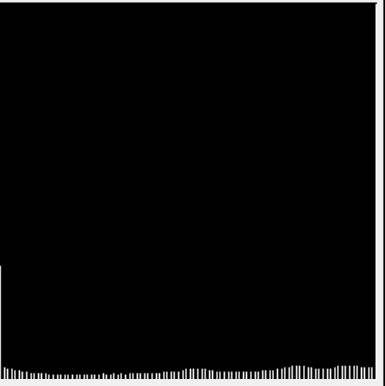
	Original	Addition	Subtraction	Multiplication	Division
Image					
Histogram					

Image & Histogram

End-in Search					
	Equalization	Specification	Low30, High30	Low30, High10	Low10, High30
Image					
Histogram					

Arithmetic Code

```
void ARITHMETIC_OPERATION(img_buf* img)
{
    for (int i = 0; i < img->info.width * img->info.height; i++)
    {
        //alpha
        if (img->Ori_img[i] + alpha > 255) { // +alpha 255 Clipping
            img->Arith_Addition[i] = 255;
        }
        else if (img->Ori_img[i] + alpha < 0) { // +alpha 0 Clipping
            img->Arith_Addition[i] = 0;
        }
        else { // +alpha
            img->Arith_Addition[i] = img->Ori_img[i] + alpha;
        }
        //beta
        if (img->Ori_img[i] - beta > 255) { // -beta 255 Clipping
            img->Arith_Subtraction[i] = 255;
        }
        else if (img->Ori_img[i] - beta < 0) { // -beta 0 Clipping
            img->Arith_Subtraction[i] = 0;
        }
        else { // -beta
            img->Arith_Subtraction[i] = img->Ori_img[i] - beta;
        }
        //gamma
        if ((double)img->Ori_img[i] * (double)gamma + 0.5 > 255) { // *gamma, 반올림 255 Clipping
            img->Arith_Multiplication[i] = 255;
        }
        else if ((double)img->Ori_img[i] * (double)gamma + 0.5 < 0) { // *gamma, 반올림 0 Clipping
            img->Arith_Multiplication[i] = 0;
        }
        else { // *gamma, 반올림
            img->Arith_Multiplication[i] = (double)img->Ori_img[i] * (double)gamma + 0.5;
        }
        //delta
        if ((double)img->Ori_img[i] / (double)delta + 0.5 > 255) { // /delta, 반올림 255 Clipping
            img->Arith_Division[i] = 255;
        }
        else if ((double)img->Ori_img[i] / (double)delta + 0.5 < 0) { // /delta, 반올림 0 Clipping
            img->Arith_Division[i] = 0;
        }
        else { // /delta, 반올림
            img->Arith_Division[i] = (double)img->Ori_img[i] / (double)delta + 0.5;
        }
    }
}
```

Arithmetic operation에서 덧셈, 뺄셈 연산에서는 0미만 -> 0, 255초과 -> 255에 대한 양방향Clipping을 진행했으며 곱셈, 나눗셈 연산에서는 자료형의 특성을 이용하여 반올림 연산 이후 같은 Clipping을 진행하였습니다.

Stretch Code

```
//Histogram Stretch

int Low_pt = 0; // Low, High 좌표 선언
int High_pt = Max_Pix;
int Low_th = 10; // Low, High 비율 정의
int High_th = 30;

char Name_Stret[30] = "Stret.";
UChar* STERT_IMG = (UChar*)calloc((Wid*Hei), sizeof(UChar)); // STERT_IMG 메모리 할당

for (int i = 0; i < Pix_Range; i++) { //Low 좌표 생성
    if (Accum_Sum[i] < (double)Wid * (double)Hei * (double)Low_th / (double)100 {
        Low_pt = i;
    }
}

for (int i = Pix_Range - 1; i >= 0; i--) { //High 좌표 생성
    if (Accum_Sum[i] > (double)Wid * (double)Hei * (double)(100 - High_th) / (double)100 {
        High_pt = i;
    }
}

for (int i = 0; i < Hei * Wid; i++) {
    if (Data[i] <= Low_pt) { //Low 이하 0
        STERT_IMG[i] = 0;
    }
    else if (Data[i] >= High_pt) { //High 이상 Max_Pix
        STERT_IMG[i] = Max_Pix;
    }
    else {
        STERT_IMG[i] = (double)Max_Pix * ( (double)(Data[i] - Low_pt) / (double)(High_pt - Low_pt) ) + 0.5; //균일하게 곱함
    }
}

strcat(Name_Stret, String);
HISTOGRAM(STERT_IMG, Wid, Hei, Name_Stret);

strcat(Name_Stret, Name_IMG);
strcat(Name_Stret, Name_extension);
fopen_s(&wp, Name_Stret, "wb");
fwrite(STERT_IMG, sizeof(UChar), (Wid*Hei), wp);
fclose(wp);
```

Stretch에서 Low_pt, High_pt라는 하위 Low_th%, 상위 High_th%를 나타내는 밝기 값의 좌표를 Accum_sum을 이용해 정의하였고 두 좌표를 이용하여 Low_pt 이하의 값들은 0, High_pt 이상의 값들은 Max_pix로 할당 후 Low_pt, High_pt 사이의 값들은 pdf 수식을 활용하여 균일하게 분포시켜 End-in search를 수행하였습니다.

Histogram에서 최대 빈도 수를 기준으로 정규화가 진행되므로 빈도 수가 많은 0, 255이외 숫자에서는 그래프가 작게 나타남을 확인할 수 있었습니다.

또한, Low_th가 낮을수록 High_th가 높을수록 이미지가 밝게 나타나고 Low_th가 높을수록 High_th가 낮을수록 이미지가 어둡게 나타남을 확인할 수 있었습니다.

Equalization Code

```
for (int i = 0; i < Wid * Hei; i++)  
    LUT[Data[i]]++;  
  
//Histogram Equalization  
Accum_Sum[0] = LUT[0];  
for (int i = 1; i < Pix_Range; i++)  
    Accum_Sum[i] = Accum_Sum[i - 1] + LUT[i];  
  
for (int i = 0; i < Pix_Range; i++)  
    EQUAL_LUT[i] = ((double)Max_Pix / ((double)Wid * (double)Hei)) * (double)Accum_Sum[i] + 0.5;  
  
for (int i = 0; i < Wid * Hei; i++)  
    EQUAL_IMG[i] = EQUAL_LUT[Data[i]];  
  
strcpy(Name_Equal, String);  
HISTOGRAM(EQUAL_IMG, Wid, Hei, Name_Equal);  
  
strcpy(Name_Equal, Name_IMG);  
strcpy(Name_Equal, Name_extension);  
fopen_s(&wp, Name_Equal, "wb");  
fwrite(EQUAL_IMG, sizeof(UChar), (Wid * Hei), wp);  
fclose(wp);  
  
////////////////////////////////////
```

LUT는 화소 별 빈도 수를 나타내며 Accum_Sum은 빈도 수의 누적합을 나타냅니다.

Accum_Sum의 최댓값이 총 픽셀 수인 $Wid * Hei$ 를 나타내므로 $Max_Pix / (Wid * Hei)$ 즉, 정규화 작업(반올림 포함)을 통해 최댓값을 Max_Pix 로 맞춰 Equalization이 수행됩니다.

Specification Code

```
////////////////////////////////////
//Histogram Specification
int LUT_BUF[Pix_Range] = { 0 };
int Speci_Accum_Sum[Pix_Range] = { 0 }; //누적 빈도수
int SPECI_LUT[Pix_Range] = { 0 };
int INV_EQUAL_LUT[Pix_Range] = { 0 };
char Name_Speci[30] = "Speci_";

UChar* SPECI_IMG = (UChar*)calloc((Wid*Hei), sizeof(UChar)); //Specification 메모리 할당

for (int i = Pix_Range / 2; i < Pix_Range; i++) //128부터 2048 값을 가지는 LUT
    LUT_BUF[i] = 2048;

Speci_Accum_Sum[0] = LUT_BUF[0];
for (int i = 1; i < Pix_Range; i++) //128부터 2048 * (n - 127) 값는 누적합
    Speci_Accum_Sum[i] = Speci_Accum_Sum[i - 1] + LUT_BUF[i];

for (int i = 0; i < Pix_Range; i++) //원하는 Histogram의 Equalization
    INV_EQUAL_LUT[i] = ((double)Max_Pix / ((double)Wid * (double)Hei)) * (double)Speci_Accum_Sum[i] + 0.5;

int Val1, Val2;
for (int i = 0; i < Pix_Range - 1; i++) //역평활화 수행
{
    Val1 = INV_EQUAL_LUT[i];
    Val2 = INV_EQUAL_LUT[i + 1];
    if (Val1 != Val2) //Equalization된 누적값 -> 밝기값
    {
        for (int j = Val1; j < Val2; j++) //밝기값 -> 역평활화 값
            SPECI_LUT[j] = i + 1;
        else
            SPECI_LUT[Val1] = 0;
    }
}
SPECI_LUT[Pix_Range - 1] = Max_Pix;

for (int i = 0; i < Wid * Hei; i++) //Equalization 이후 Specification 진행
    SPECI_IMG[i] = SPECI_LUT[EQUAL_IMG[i]];

strcpy(Name_Speci, String);
HISTOGRAM(SPECI_IMG, Wid, Hei, Name_Speci);

strcpy(Name_Speci, Name_IMG);
strcpy(Name_Speci, Name_extension);

fopen_s(&wp, Name_Speci, "wb");
fwrite(SPECI_IMG, sizeof(UChar), (Wid*Hei), wp);
fclose(wp);
```

LUT_BUF에 원하는 Histogram을 생성하기 위해 128부터 Pix_Range-1까지 2048을 갖는 LUT를 생성 후 누적 합을 통해 128부터 Pix_Range-1까지 $2048 * (i - 127)$ 값을 나타내는 Speci_Accum_Sum이 생성됨을 확인할 수 있었습니다. (Pix_Range-1 에는 Wid * Hei 값)

이후 Equalization을 진행 후의 누적값을 Val에 할당하여 밝기 값이 되고, 밝기 값을 역평활화 값으로 사용하여 SPECI_LUT를 만들어 기존 EQUAL_IMG를 기반으로 Specification을 진행됨을 확인할 수 있었습니다.