



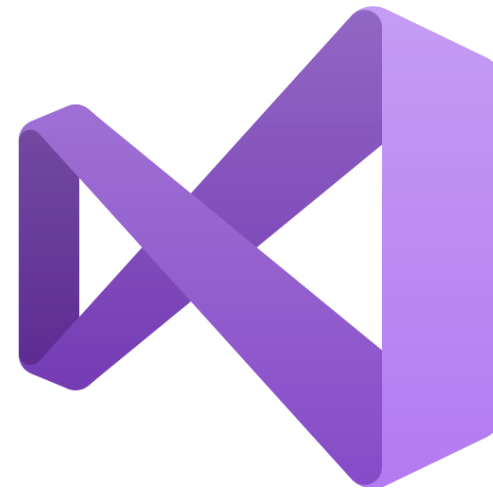
# C언어 기반의 GrayScale Image Processing

이윤혁



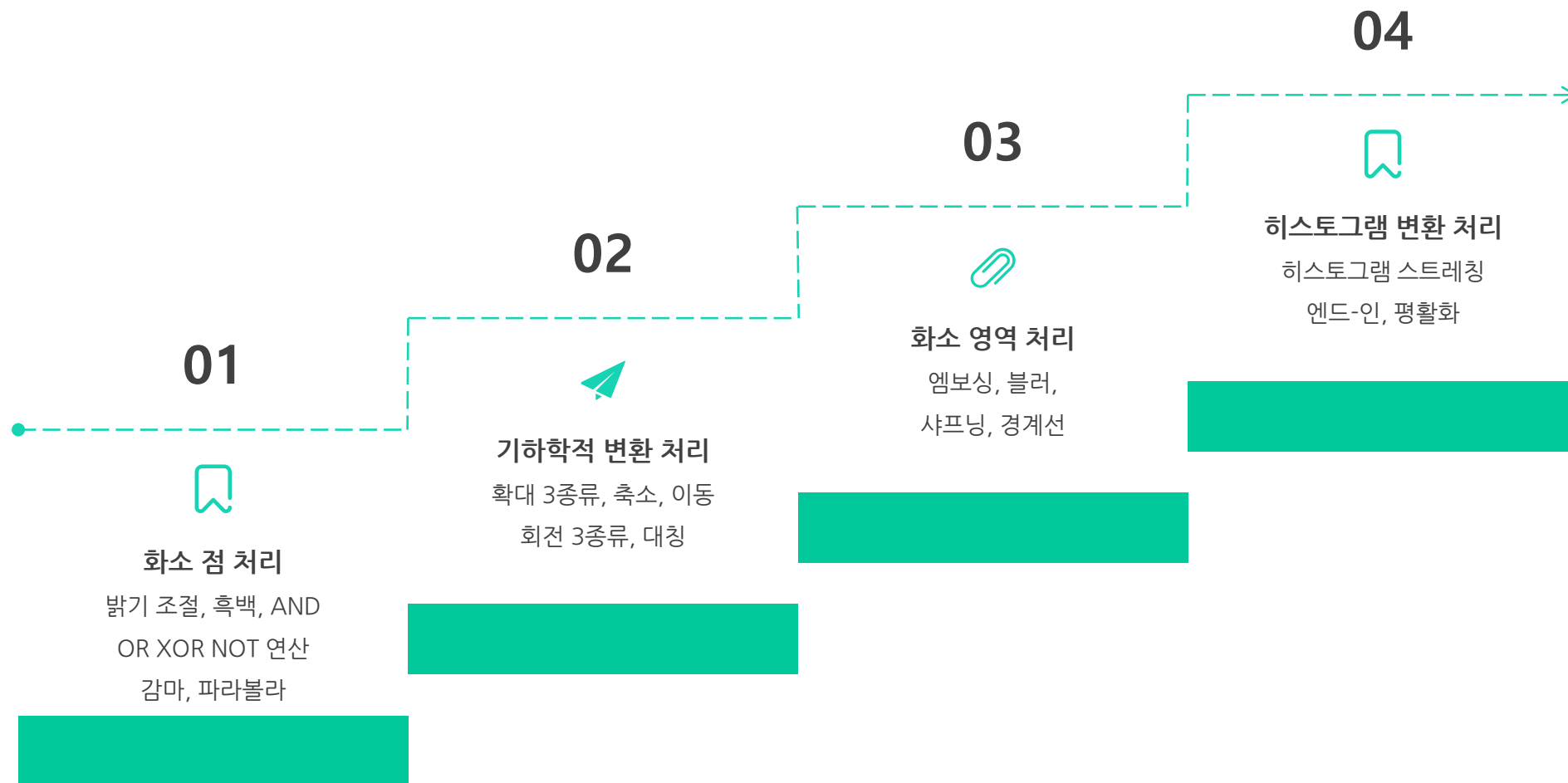
### 개요

C언어를 기반으로 한  
다양한 영상처리 기능을 활용한  
GrayScale Image Processing



### 구축 환경

Windows 11  
Visual Studio ver. 17.9.2





```
## GrayScale Image Processing (Beta 4) ##
```

0.열기 1.저장 9.종료

Q. 동일 / W.밝게 / E.어둡게 / R.반전 / T.흑백 / Y.이동  
 U.감마 / I.파라볼라CAP / O.파라볼라CUP / P.AND 처리 / [.OR 처리 / ].XOR 처리  
 A.확대(포워딩) / S.확대(백워딩) / D.축소 / F.확대(양선형 보간)  
 G.회전 / H.회전(중양, 백워딩) / J.회전+확대  
 Z.엠보싱 / X.블러 / C.샤프닝 / V.경계선1 / B.엔드-인  
 N.히스토그램 스트레칭 / M.평활화 / L.대칭

```
char inKey = 0;
while (inKey != '9') { // 메뉴 호출
    printMenu();
    inKey = _getch();
    system("cls");

    switch (inKey)
    {
        case '0': loadImage(); break;
        ....
    }
}
```

While 문과  
 Switch - Case문을 사용하여  
 메뉴 호출 함수 구축.  
 PrintMenu() 함수는 puts함수를  
 사용하여 따로 정의함



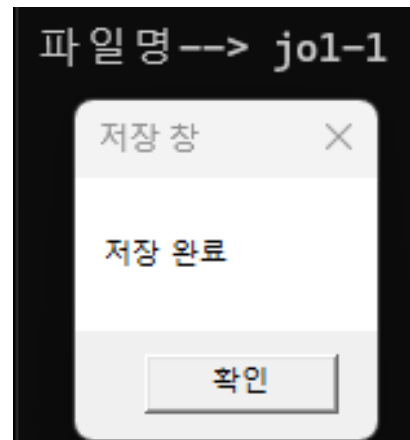
파일명--> jo1  
원본 이미지 파일

파일명--> jo23  
잘못된 파일명입니다. 다시 입력해주세요.  
파일명-->

파일명을 잘못 입력할 시  
재입력 유도



RAW 파일을  
GrayScale로 변환한 모습



변환한 파일을  
RAW 형태로 저장

```
while (!isValidFileName) {
    printf("파일명--> "); // ex) flower!
    scanf("%s", tmpName);
    strcat(fullName, tmpName);
    strcat(fullName, ".raw");
    strcpy(fileName, fullName);
    // (중요!) 이미지의 폭과 높이를 결정
    rfp = fopen(fileName, "rb");
}
```

파일명을 입력 받은 후  
이미지를 GrayScale로 불러옴

```
// 메모리 --> 파일 (한 행씩)
for (int i = 0; i < outH; i++)
    fwrite(outImage[i], sizeof(unsigned char), outW, wfp);
fclose(wfp);
```

fwrite함수 사용하여  
RAW형태로 저장



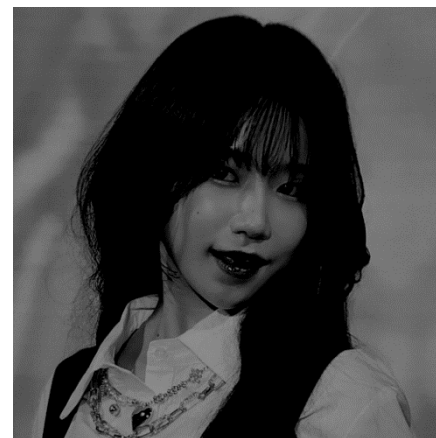
## N 화소 점 처리 알고리즘 (1)



정수 값 --> 100  
밝게 처리

```
if (inImage[i][k] + val < 255)
    outImage[i][k] = inImage[i][k] + val;
else
    outImage[i][k] = 255;
```

- ★ inImage(원본)에 입력 받은 정수 값을 더 함.
- ★ outImage 값이 255를 넘을 때를 고려하여 if 문 사용



정수 값 --> 100  
어둡게 처리

```
if (inImage[i][k] - val > 0)
    outImage[i][k] = inImage[i][k] - val;
else
    outImage[i][k] = 0;
```

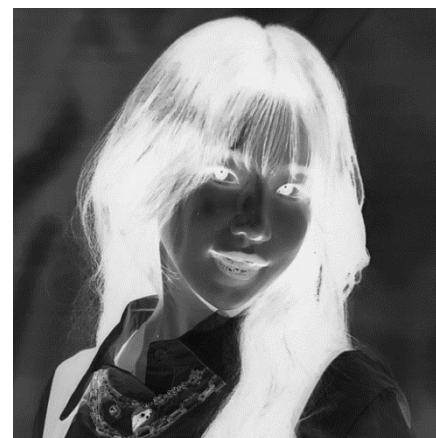
- ★ inImage(원본)에 입력 받은 정수 값을 뺌.
- ★ outImage 값이 0를 미만이 될 때를 고려하여 if 문 사용



흑백 처리

```
if (inImage[i][k] < 127)
    outImage[i][k] = 0;
else
    outImage[i][k] = 255;
```

- ★ 중앙값 127을 기준으로 각 픽셀을 0 or 255 값으로 설정



반전 처리

```
outImage[i][k] = 255 - inImage[i][k];
```

- ★ 각 픽셀의 값을 255에 대한 역수로 대입



## N 화소 점 처리 알고리즘 (2)



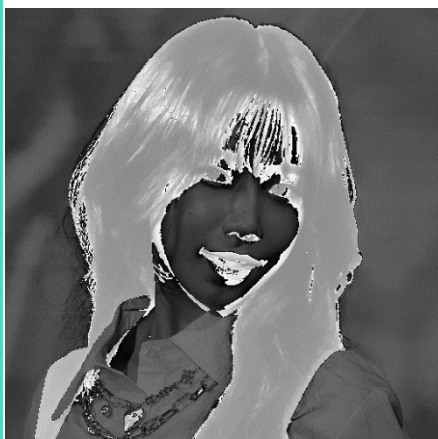
정수 값 --> 128  
And 처리 알고리즘



```
outImage[i][k] =  
(unsigned char)(inImage[i][k] & (unsigned char)val);
```

0~255 범위 안에 있는 경우  
& 연산자를 사용하여 연산

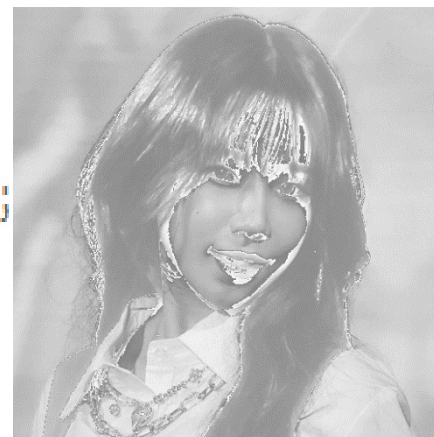
정수 값 --> 128  
XOR 처리 알고리즘



```
outImage[i][k] =  
(unsigned char)(inImage[i][k] ^ (unsigned char)val);
```

0~255 범위 안에 있는 경우  
XOR 연산자를 사용하여 연산

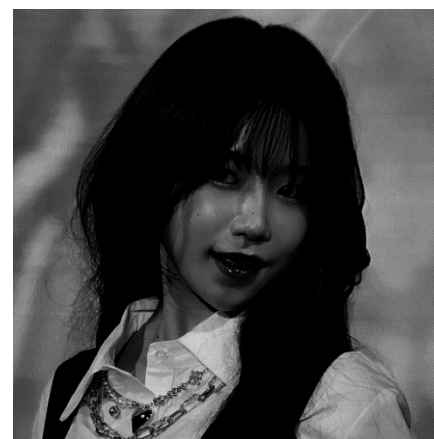
정수 값 --> 128  
Or 처리 알고리즘



```
outImage[i][k] =  
(unsigned char)(inImage[i][k] | (unsigned char)val);
```

0~255 범위 안에 있는 경우  
OR 연산자를 사용하여 연산

감마 처리 (0~10) --> 4



```
float normalized_pixel = (float)inImage[i][k] / 255.0;  
float corrected_pixel = pow(normalized_pixel, gamma);
```

픽셀 값을 0~1로 정규화 한 후  
감마 함수 적용 후 보정된 값 계산



### 파라볼라 CAP 처리

```
outImage[i][k] = 255 - 255 * pow((inImage[i][k] / 128 - 1), 2);
```

Parabola CAP 함수를 사용하여 연산



### 파라볼라 CUP 처리

```
outImage[i][k] = 255 * pow((inImage[i][k] / 128 - 1), 2);
```

Parabola CUP 함수를 사용하여 연산





## N 기하학적 변환 알고리즘 (1)



배율 입력  
정수 값 --> 2  
확대 (포워딩)



```
outImage[(int)(i * scale)][(int)(k * scale)] = inImage[i][k]; outImage[i][k] = inImage[(int)(i / scale)][(int)(k / scale)];
```

1024\*1024 사진.  
(원본은 512\*512)  
확대 (포워딩)을 적용하여  
화질이 깨진 모습

배율 입력  
정수 값 --> 2  
확대 (백워딩)



1024\*1024 사진.  
(원본은 512\*512)  
확대 (백워딩)을 적용하여  
화질을 유지한 채 확대



## N 기하학적 변환 알고리즘 (2)



배율 입력  
정수 값 --> 2  
확대 (양선형 보간)



```
outImage[(int)(i * scale)][(int)(k * scale)] = inImage[i][k]; outImage[i][k] = inImage[(int)(i / scale)][(int)(k / scale)];
```

1024\*1024 사진.  
(원본은 512\*512)  
확대 (양선형 보간법)을 적용하여  
화질이 유지 됨

정수 값 --> 2  
축소



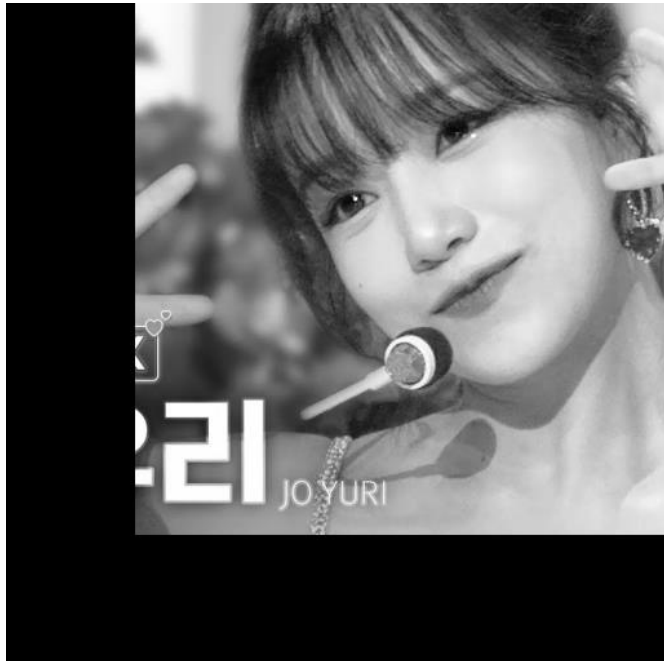
216\*216 사진.  
(원본은 512\*512)  
축소 알고리즘을 적용하여  
입력한 정수 값에 따라 축소



## N 기하학적 변환 알고리즘 (3)



```
정수 값--> 100  
정수 값--> 100  
이미지 이동
```



```
outImage[i][k] = inImage[i + a][k - b];
```

이동 알고리즘 사용.  
정수 값 가로 100 세로 100 입력.  
경계를 벗어나는 경우는  
if문을 사용하여 처리



## N 기하학적 변환 알고리즘 (4)



정수 값 --> 50  
회전



```
int xd = cos(radian) * xs - sin(radian) * ys;  
int yd = sin(radian) * xs + cos(radian) * ys;
```

입력 받은 각도 만큼 회전.

if ((0 <= xd && xd < outH) && (0 <= yd && yd < outW))

if문을 사용하여 벗어난 이미지에 대한 예외처리.

하지만 중심이 맞지 않고, 화질 저하

정수 값 --> 50  
회전 + 중심 + 백워딩



```
int cx = inH / 2;  
int cy = inW / 2;
```

```
int xs = (int)(cos(radian) * (xd - cx) + sin(radian) * (yd - cy));  
int ys = (int)(-sin(radian) * (xd - cx) + cos(radian) * (yd - cy));
```

입력 받은 각도 만큼 회전.

중심 맞추는 코드 추가.

백워딩을 적용하여 화질 보존





회전 각도 입력  
정수 값 --> 50  
확대 배율 입력  
정수 값 --> 2  
회전 및 확대 완료



```
int xd = i / scale;  
int yd = k / scale;  
int xs = (int)(cos(radian) * (xd - cx) - sin(radian) * (yd - cy) + cx);  
int ys = (int)(sin(radian) * (xd - cx) + cos(radian) * (yd - cy) + cy);
```

1024\*1024 사진.  
(원본은 512\*512)  
입력 받은 각도 만큼 회전.  
입력 받은 배율 만큼 확대.  
.회전과 동시에 확대를 시켰음.  
확대에는 백워딩 기법을 적용하여  
화질 저하 방지



## N 기하학적 변환 알고리즘 (6)



1 = 상하 반전 or 2 = 좌우 반전  
정수 값 --> 1  
상하 대칭



```
if (a == 1)
    outImage[i][k] = inImage[inH - 1 - i][k];
```

1or2를 입력 받아 대칭 알고리즘 수행  
1을 입력 받았으므로 상하 대칭 수행

1 = 상하 반전 or 2 = 좌우 반전  
정수 값 --> 2  
좌우 대칭



```
else
    outImage[i][k] = inImage[i][inW - 1 - k];
```

1or2를 입력 받아 대칭 알고리즘 수행  
2를 입력 받았으므로 상하 대칭 수행



## 엠보싱

```
double mask[3][3] =  
{ {-1.0, 0.0, 0.0},  
  { 0.0, 0.0, 0.0},  
  { 0.0, 0.0, 1.0} };
```

```
// 마스크 (3x3) 와 한점을 중심으로한 3x3을 곱하기  
S = 0.0; // 마스크 9개와 입력값 9개를 각각 곱해서 합한 값.  
....  
    S += tmpInImage[i + m][k + n] * mask[m][n];  
}  
tmpOutImage[i][k] = S;
```



## 블러

```
double mask[3][3] =  
{ {1. / 9, 1. / 9, 1. / 9},  
  {1. / 9, 1. / 9, 1. / 9},  
  {1. / 9, 1. / 9, 1. / 9} };
```

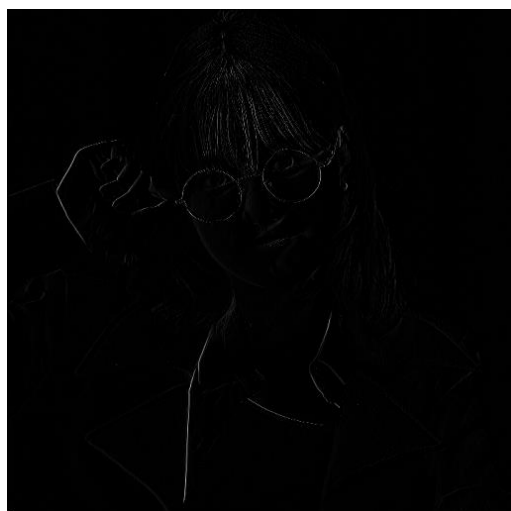
```
tmpInImage[i + 1][k + 1] = inImage[i][k];  
임시로 이미지의 가장자리를 확장.  
이후 후처리가 필요  
tmpOutImage[i][k] += 127.0;
```



### 샤프닝

```
double mask[3][3] =  
{ {0.0, -1.0, 0.0},  
  {-1.0, 5.0, -1.0},  
  {0.0, -1.0, 0.0}};
```

```
// 마스크 (3x3) 와 한점을 중심으로한 3x3을 곱하기  
S = 0.0; // 마스크 9개와 입력값 9개를 각각 곱해서 합한 값.  
....  
    S += tmpInImage[i + m][k + n] * mask[m][n];  
}  
tmpOutImage[i][k] = S;
```



### 경계선 처리

```
double mask[3][3] =  
{{0.0, 0.0, 0.0},  
 {-1.0, 1.0, 0.0},  
 {0.0, 0.0, 0.0}};
```

```
tmpInImage[i + 1][k + 1] = inImage[i][k];  
임시로 이미지의 가장자리를 확장.  
이후 후처리가 필요  
tmpOutImage[i][k] += 127.0;
```





### 히스토그램 스트레칭

```
if (inImage[i][k] < low)
    low = inImage[i][k];
if (inImage[i][k] > high)
    high = inImage[i][k];
```

히스토그램의 최저 명도 값과  
최고 명도 값을 구하는 함수를 삽입 후  
기본 명암대비 스트레칭 수행 공식을 코드에 삽입

```
old = inImage[i][k];
new = (int)((double)(old - low) / (double)(high - low) * 255.0);
```

### 엔드-인

```
old = inImage[i][k];
new = (int)((double)(old - low) / (double)(high - low) * 255.0);
if (new > 255)
    new = 255;
if (new < 0)
    new = 0;
```

일정한 양의 화소를 흰색이나 검정색으로 지정하여  
히스토그램의 분포를 좀더 균일하게 만들  
엔드-인 탐색 수행 공식을 사용





### 히스토그램 평활화



```
int histo[256] = { 0. };
```

```
....
```

```
histo[inImage[i][k]]++;
```

1단계 : 히스토그램 빈도수 세기

```
int sumHisto[256] = { 0, };
```

```
sumHisto[0] = histo[0];
```

```
...
```

```
sumHisto[i] = sumHisto[i - 1] + histo[i];
```

2단계 : 누적 히스토그램 생성

```
double normalHisto[256] = { 1.0, };
```

```
.... {
```

```
normalHisto[i] = sumHisto[i] * (1.0 / (inH * inW)) * 255.0;
```

3단계 : 정규화 된 히스토그램 생성

```
outImage[i][k] = (unsigned char)normalHisto[inImage[i][k]];
```

4단계 : inImage를 정규화 된 값으로 치환



C언어로 본격적으로 이렇게 프로그램을 만들어 본 것은 처음이었다. 그 동안 포토샵 같은 프로그램들이 어떻게 작동하는지 원리를 전혀 몰랐는데 이 프로젝트를 진행하면서 많이 알아가게 되는 것 같다.

아직은 남아 있는 수 많은 영상 처리 알고리즘들을 다 담아내진 못하였다. 내가 코딩 실력이 부족함에 따라 시간이 남들보다 많이 걸렸기 때문이라 생각한다. 출발이 늦은 만큼 더더욱 노력 해야겠다.

향후에 이 프로그램을 더 발전 시킨다면, 우선 시간 관계상 코딩하지 못했던 추가 기능들을 추가한다음, 궁극적으로 GrayScale이 아닌 풀컬러 영상에서도 동작하게 하는 프로그램을 만들어 보고 싶다.