C++로 만드는 Image Processing

[Intel] AI SW 아카데미
객체지향 프로그래밍
김기범

프로젝트 개요

- 프로젝트 과정
- 실행 모습
- 코드 설명
- 마치면서

프로젝트 과정

목표

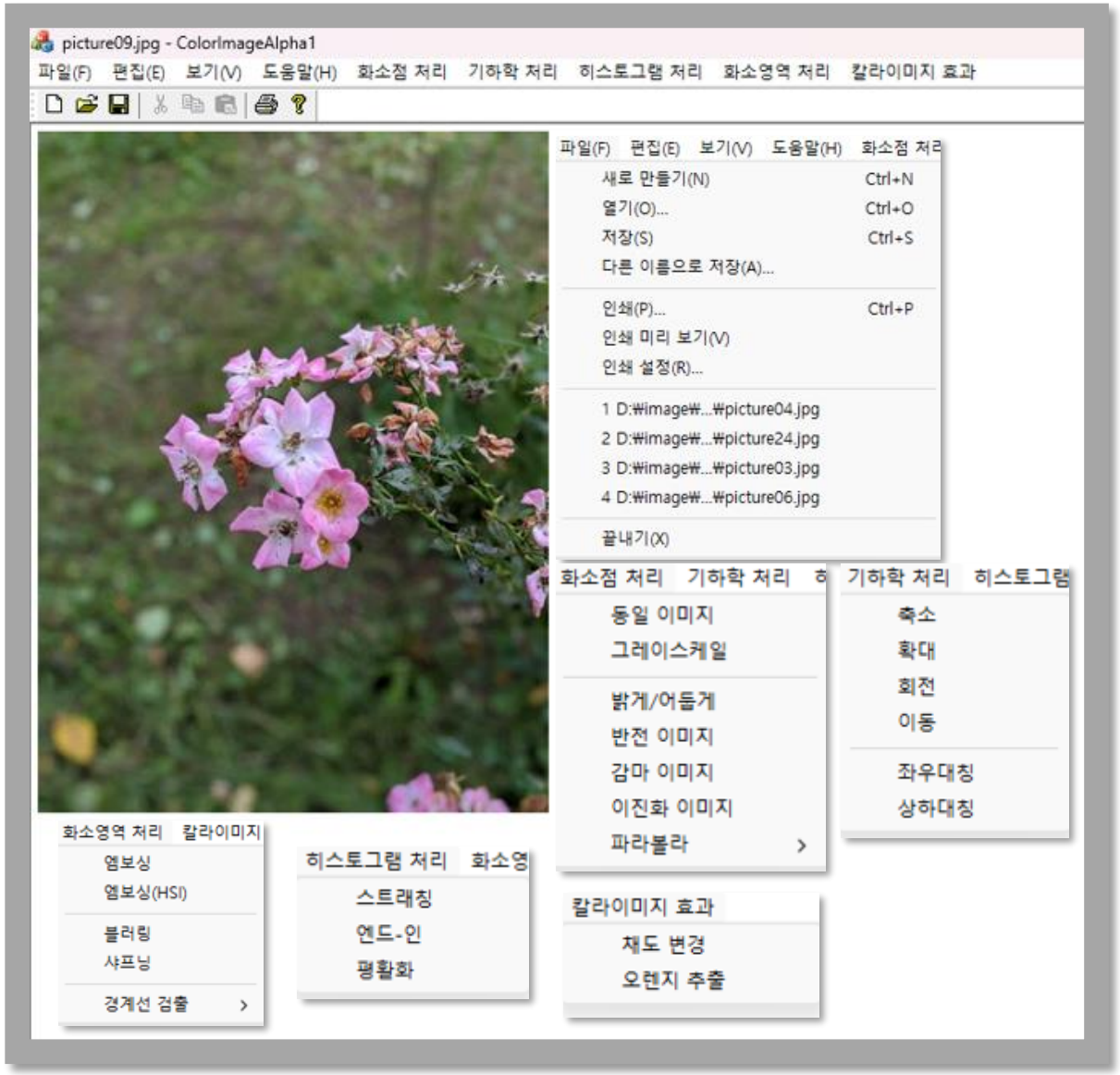
C++을 통해 디지털 영상처리 구현

개발 환경

C++, MFC, WINDOW11

진행 기간

2024.03.25~2024.04.03



실행 창

열기를 통해 이미지 불러오기

화소점 처리

기하학 처리

히스토그램 처리

화소영역 처리

컬러이미지 효과 등등

여러 이미지 효과 구현

화소점 처리

- 화소 점의 원래 값이나 화소 점의 위치를 기반으로 화소 값 변경

- 밝기조절, 반전, 감마, 이진화, 파라볼라

그레이 스케일

```
avg = (m_inImageR[i][k] + m_inImageG[i][k] +
m_inImageB[i][k]) / 3.0;
m_outImageR[i][k] = m_outImageG[i][k] =
m_outImageB[i][k] = (unsigned char) avg;
```

RGB값의 평균값을
각각의 RGB
출력값에 대입

밝기 조절(밝게/어둡게)

```
if (m_inImageR[i][k] + value > 255)
    m_outImageR[i][k] = 255;
else if (m_inImageR[i][k] + value < 0)
    m_outImageR[i][k] = 0;
else
    m_outImageR[i][k] = m_inImageR[i][k] + value;
```

RGB값 각각 적용

반전 이미지/감마 이미지

RGB값 각각 적용

```
m_outImageR[i][k] = 255 - m_inImageR[i][k];
m_outImageG[i][k] = 255 - m_inImageG[i][k];
m_outImageB[i][k] = 255 - m_inImageB[i][k];
```

```
float m = m_inImageR[i][k];
m_outImageR[i][k] = (unsigned char)255.0 * pow(m / 255.0, gamma);
```

캡 파라볼라/컵 파라볼라

RGB값 각각 적용

Cap para
m_outImageR[i][k] = (unsigned char)255.0 - 255.0 *
pow((m_inImageR[i][k] / 128.0 - 1.0), 2);

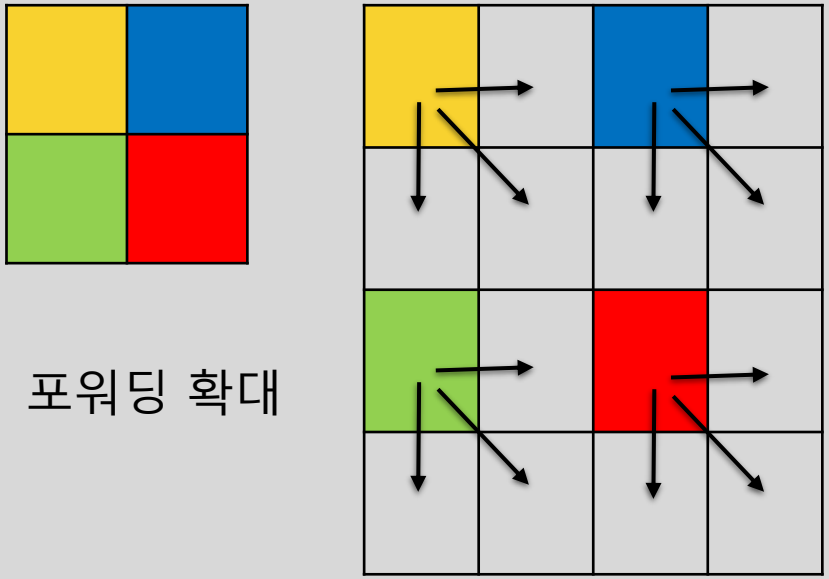
Cup para
m_outImageR[i][k] = (unsigned char)255.0 *
pow((m_inImageB[i][k] / 128.0 - 1.0), 2);

기하학 처리

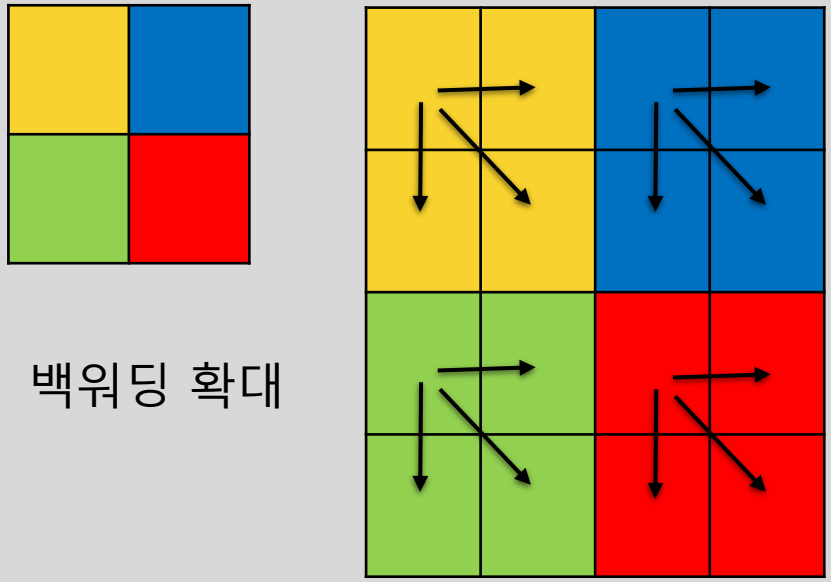
- 화소들의 위치나 배열을 변화시킴

- 축소, 확대(백워딩), 회전(중앙백워딩),
  이동, 좌우대칭, 상하대칭

기하학 처리(포워딩과 백워딩)

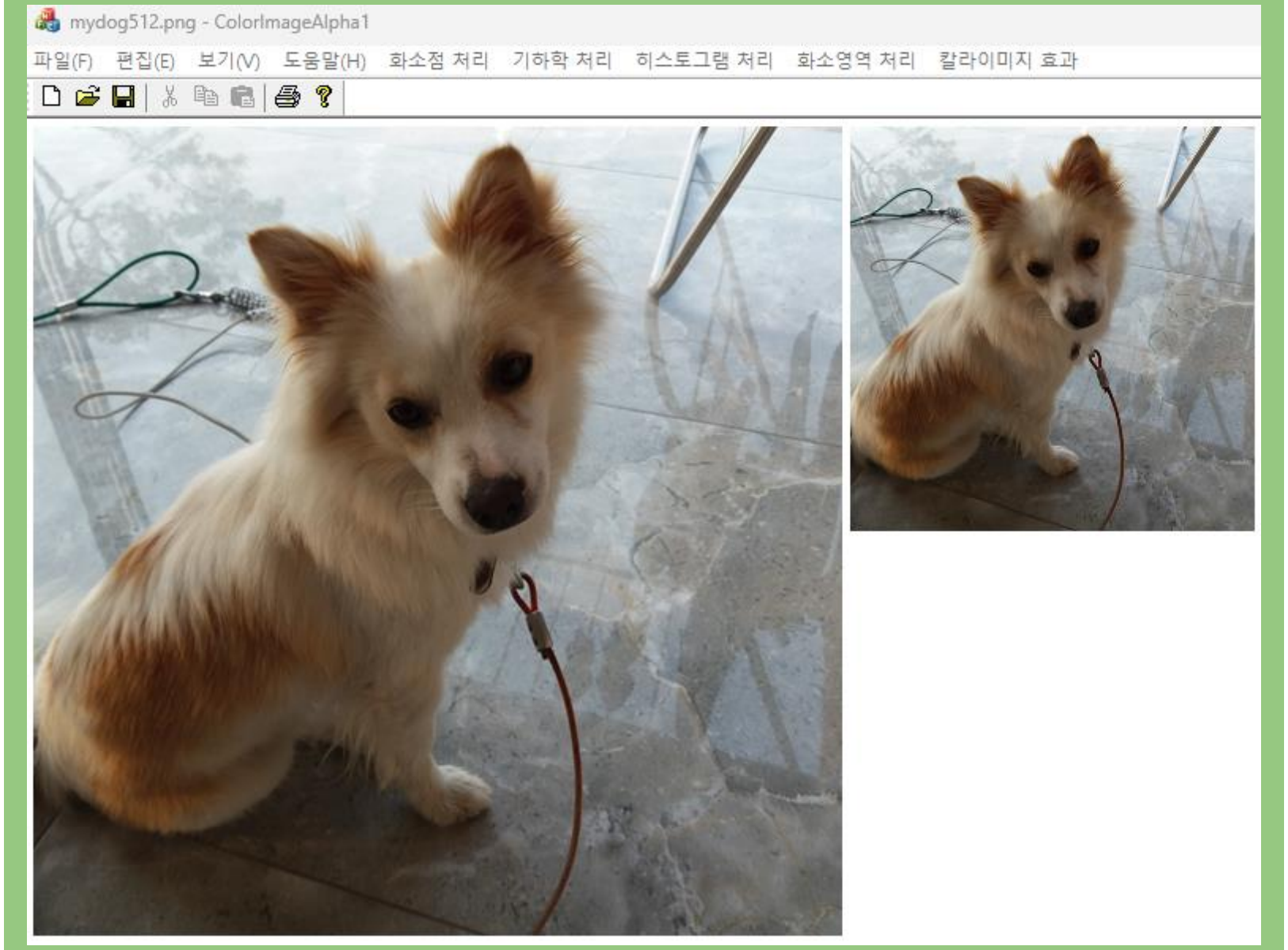


포워딩 확대



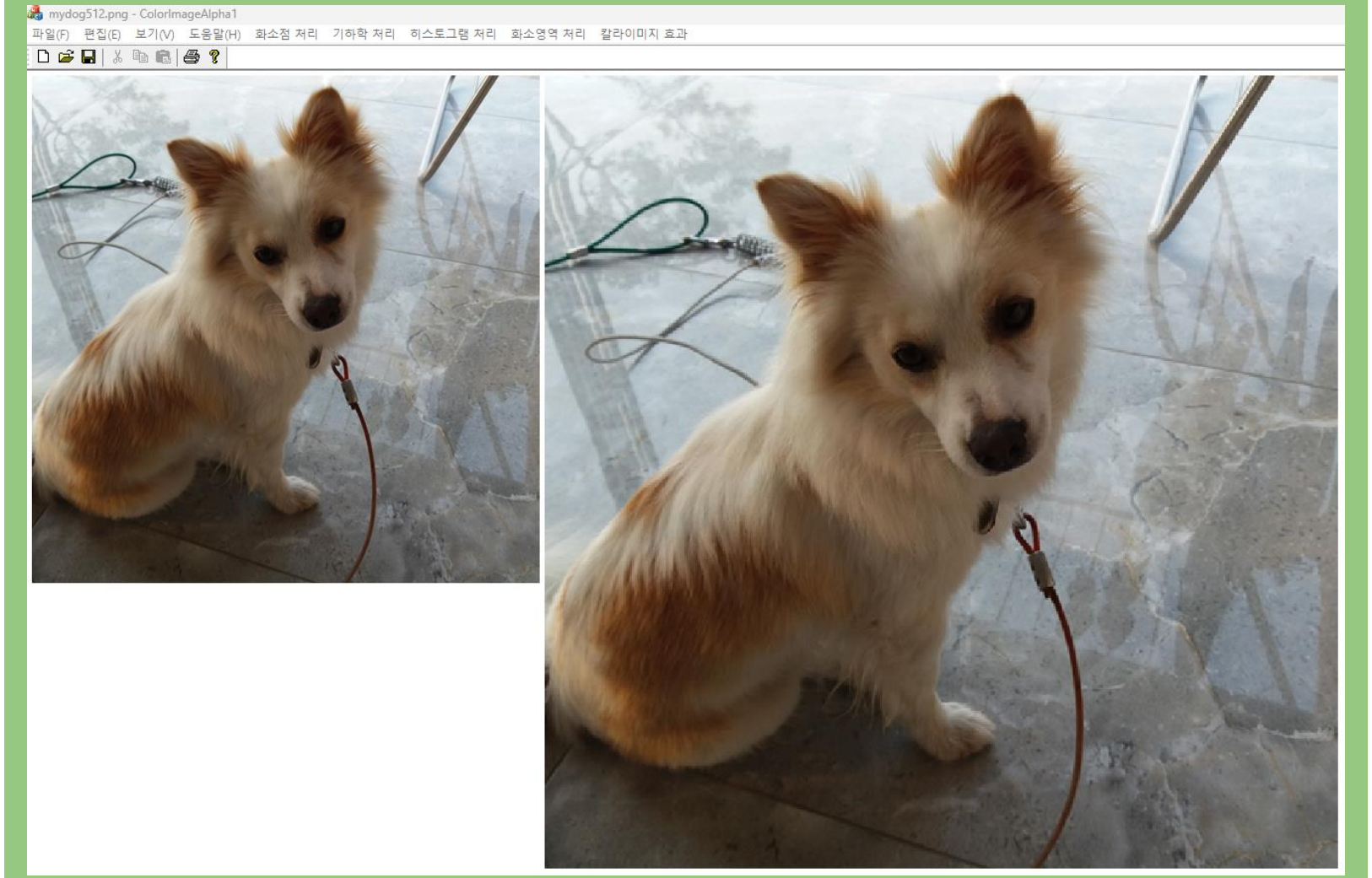
백워딩 확대

축소 시에는 문제가 없지만,
확대 시 포워딩 방식은 인접 목적 화소에 원시 화소 값을 넣지 못함
그래서 백워딩 방식으로 넣어야 홀 문제 없이 확대가 가능



축소

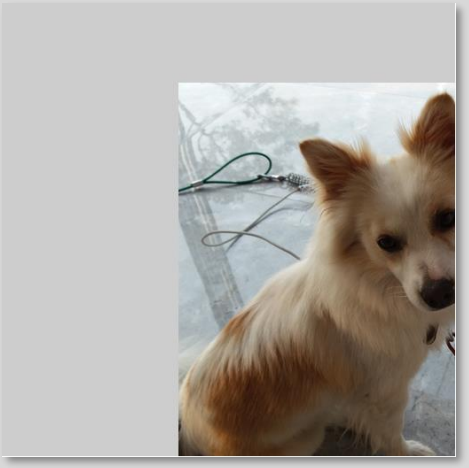
```
m_outImageR[(int)(i / value)][(int)(k / value)] = m_inImageR[i][k];
m_outImageG[(int)(i / value)][(int)(k / value)] = m_inImageG[i][k];
m_outImageB[(int)(i / value)][(int)(k / value)] = m_inImageB[i][k];
```



확대

```
m_outImageR[i][k] =
m_inImageR[(int)(i /
value)][(int)(k / value)];
m_outImageG[i][k] =
m_inImageG[(int)(i /
value)][(int)(k / value)];
m_outImageB[i][k] =
m_inImageB[(int)(i /
value)][(int)(k / value)];
```

회전/이동



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

```
int mv_i = i + D;
int mv_k = k + R;
if (0 <= mv_i && mv_i < m_outH && 0 <= mv_k && mv_k
< m_outW)
   m_outImageR[mv_i][mv_k] = m_inImageR[i][k],
   m_outImageG[mv_i][mv_k] = m_inImageG[i][k],
   m_outImageB[mv_i][mv_k] = m_inImageB[i][k];
```

좌우대칭/상하대칭

```
for (int i = 0; i < m_inH; i++) {
  for (int k = 0; k < m_inW; k++) {
    m_outImageR[i][m_outW - k - 1] = m_inImageR[i][k],
    m_outImageG[i][m_outW - k - 1] = m_inImageG[i][k],
    m_outImageB[i][m_outW - k - 1] = m_inImageB[i][k];
```

```
for (int i = 0; i < m_inH; i++) {
  for (int k = 0; k < m_inW; k++) {
    m_outImageR[m_outH - i - 1][k] = m_inImageR[i][k],
    m_outImageG[m_outH - i - 1][k] = m_inImageG[i][k],
    m_outImageB[m_outH - i - 1][k] = m_inImageB[i][k];
```

히스토그램 처리

- 관찰한 데이터의 특징을 알아볼 수 있도록 데이터를 막대그래프 모양으로 나타낸 것 디지털 영상에 대한 많은 정보를 제공함.

- 히스토그램 스트래칭, 엔드-인, 평활화

스트레칭/엔드-인/평활화

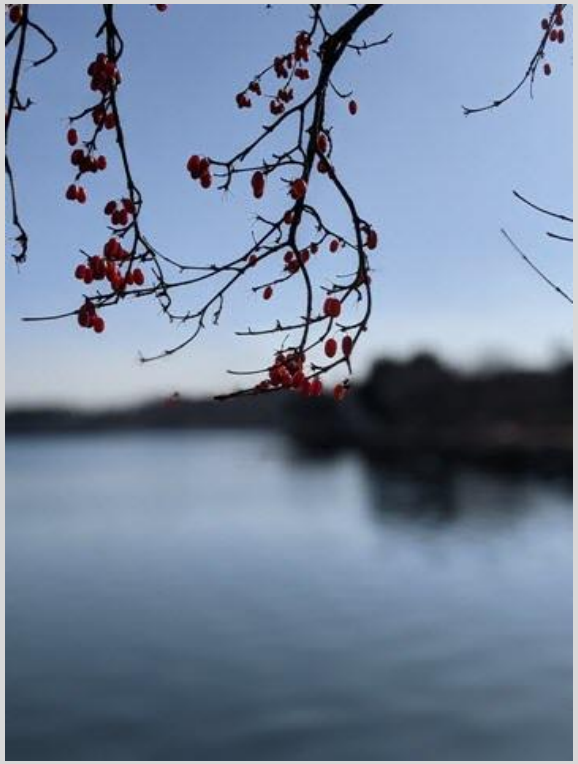
```
int oldthingR = m_inImageB[i][k];
Int newthingR =
(int)((double)(oldthingR – lowR)/
(double)(highR – lowR) * 255.0);
If (newthingR > 255)
  newthingR = 255;
if (newthingR < 0)
  newthingR = 0;
m_outImageR[i][k] = newthingR;
```

```
highR -= 50, highG -= 50, highB -= 50;
lowR += 50, lowG += 50, lowB += 50;

그외에 스트레칭과 동일
```

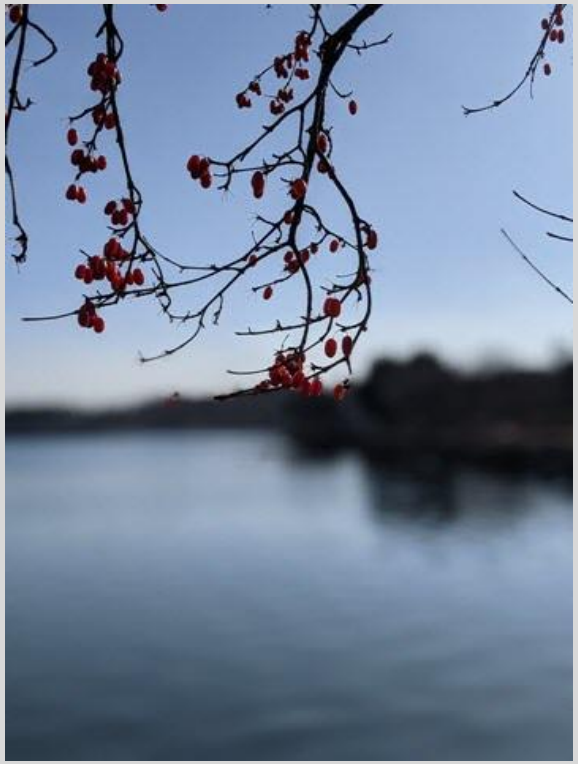
```
1단계 :빈도수 세기(=히스토그램)
histo[256]
2단계 :누적 히스토그램 생성
3단계 :정규화 히스토그램 생성
normalHisto = sumHisto * ( 1 /
(inH*inW) ) * 255.0
4단계 :inImage를 정규화 값 치환
```

화소영역 처리



- 화소의 원래 값과 이웃하는 화소의 값을 기반으로 화소 값 변경
- 엠보싱, 블러링, 샤프닝, 경계선 검출

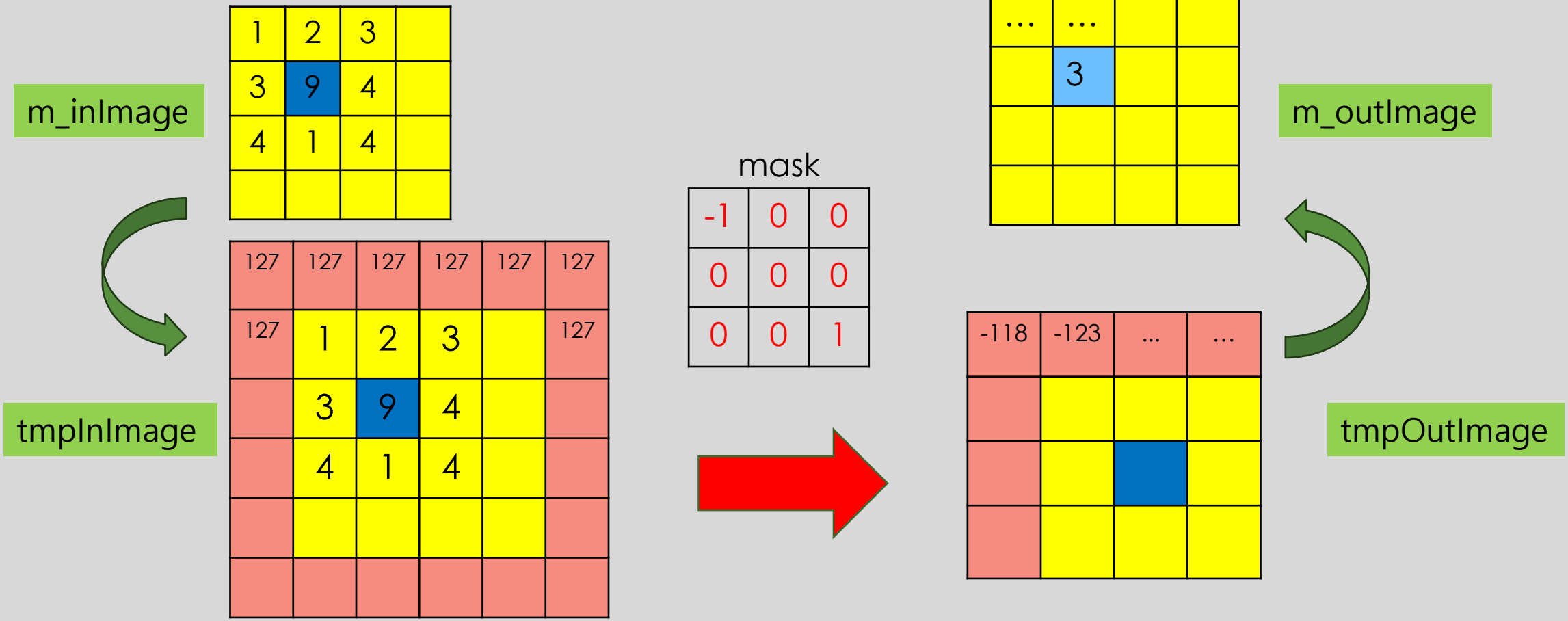
마스크 원리



- 회선 마스크를 사용해 원시 화소와 이웃한 각 화소에 가중치를 곱한 합을 출력 화소로 생성

$$Output\_pixel[x,y] = \sum_{m=(x-k)}^{x+k} \sum_{n=(y-k)}^{y+k} \left(I[m,n] \times M[m,n] \right)$$

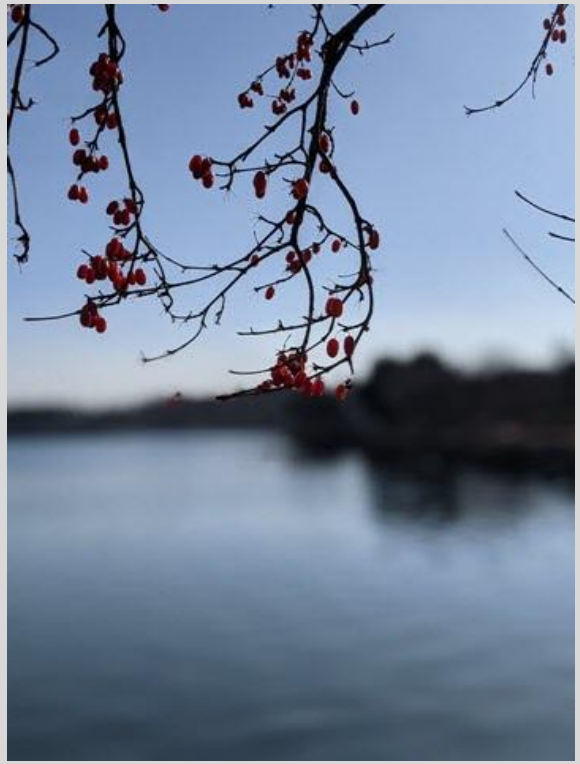
마스크 원리

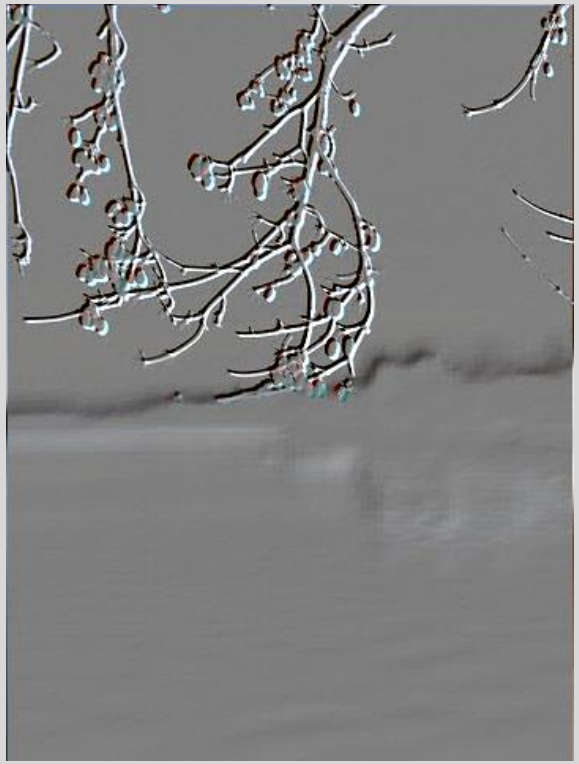

m_inImage
1 2 3
3 9 4
4 1 4
tmpInImage
127 127 127 127 127 127
127 1 2 3 127
3 9 4
4 1 4
mask
-1 0 0
0 0 0
0 0 1
... ...
3
m_outImage
-118 -123 ... ...
tmpOutImage

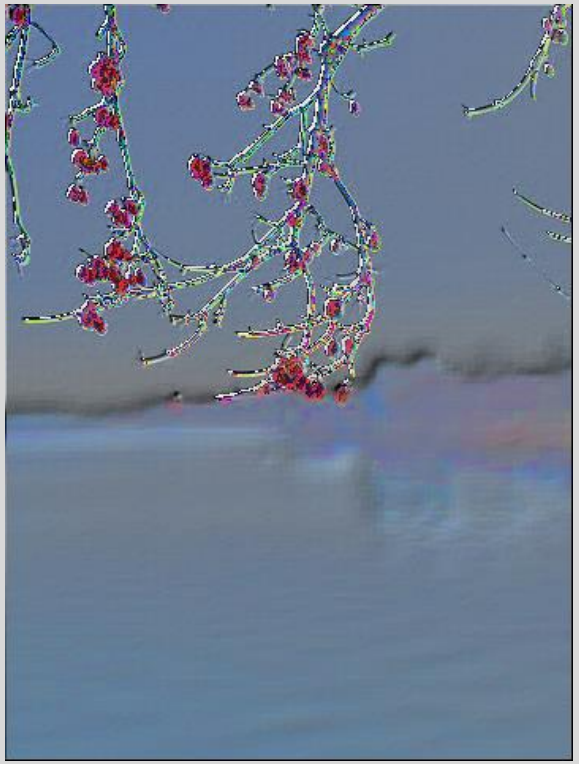
엠보싱/엠보싱(HSI 컬러모델)

{-1.0, 0.0, 0.0},
{ 0.0, 0.0, 0.0},
{ 0.0, 0.0, 1.0}

RGB 모델 -> HSI모델
회선 연산 : 마스크로 긁어가면서 계산
HSI 모델 -> RGB 모델



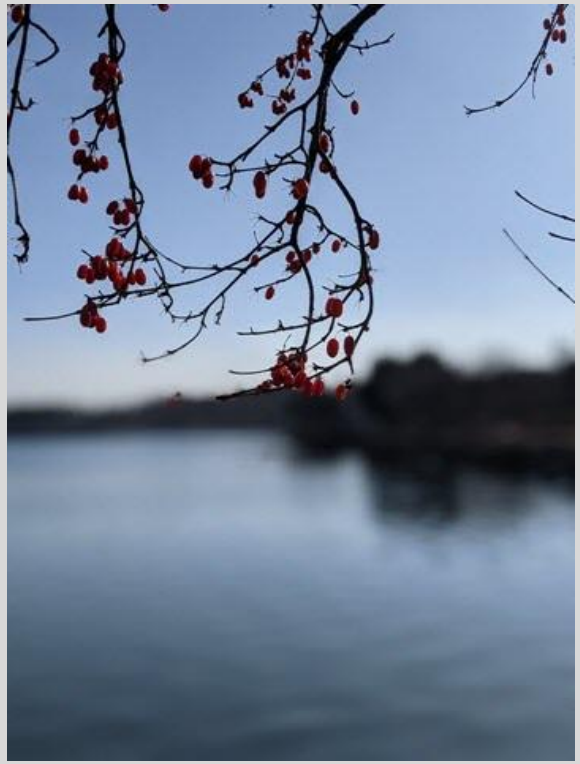


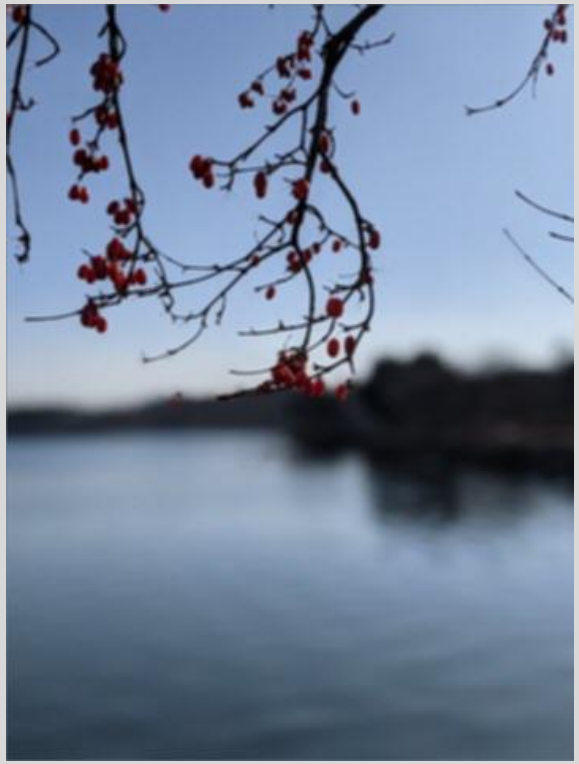


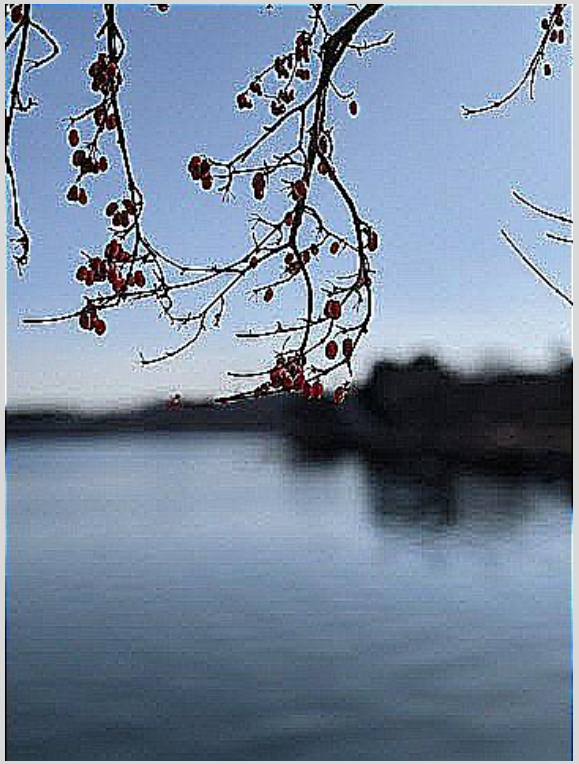
블러링/샤프닝

```
{1. / 9, 1. / 9, 1. / 9},
{1. / 9, 1. / 9, 1. / 9},
{1. / 9, 1. / 9, 1. / 9}
```

```
{-1.0, -1.0, -1.0},
{-1.0,  9.0, -1.0},
{-1.0, -1.0, -1.0}
```



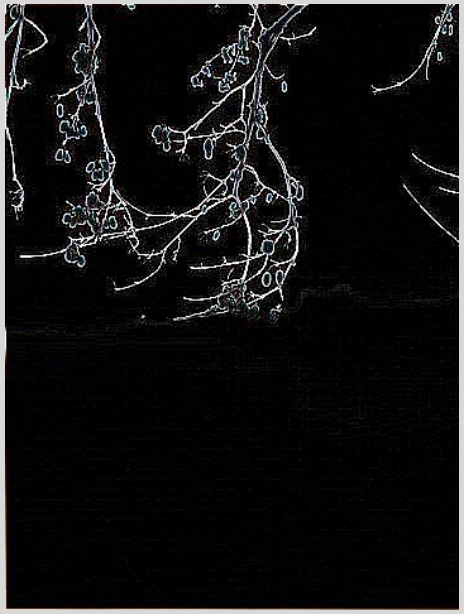




경계선 검출

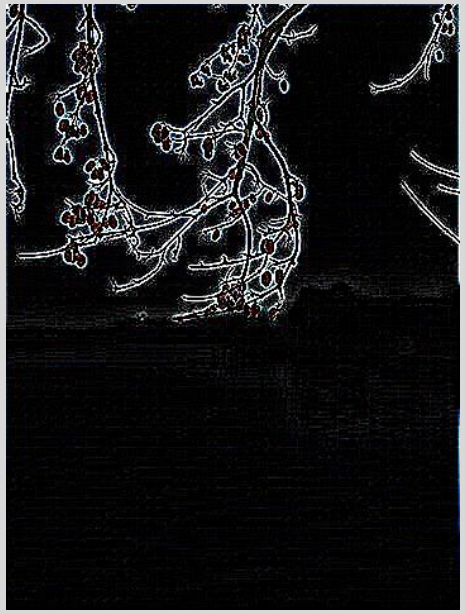
수직

{ 0.0, 0.0, 0.0},
{-1.0, 1.0, 0.0},
{ 0.0, 0.0, 0.0}



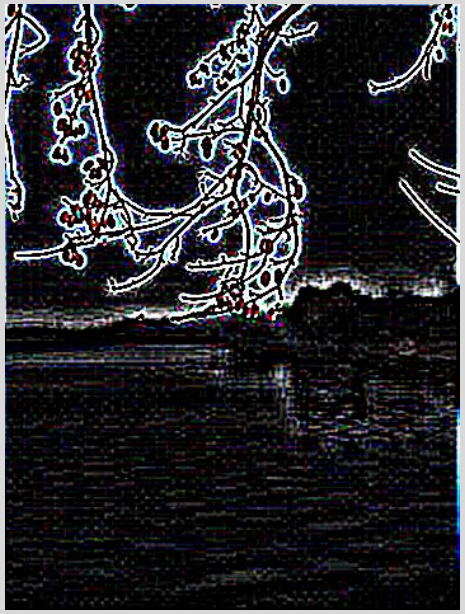
라플라시안

{ 1.0, 1.0, 1.0},
{ 1.0,-8.0, 1.0},
{ 1.0, 1.0, 1.0}



LoG

{ 0.0, 0.0, -1.0, 0.0, 0.0 },
{ 0.0,-1.0, -2.0,-1.0, 0.0 },
{-1.0,-2.0,16.0,-2.0,-1.0 },
{ 0.0,-1.0, -2.0,-1.0, 0.0 },
{ 0.0, 0.0, -1.0, 0.0, 0.0 }



DoG

{ 0.0, 0.0,-1.0,-1.0,-1.0, 0.0, 0.0},
{ 0.0,-2.0,-3.0,-3.0,-3.0,-2.0, 0.0},
{-1.0,-3.0, 5.0, 5.0, 5.0,-3.0,-1.0},
{-1.0,-3.0, 5.0,16.0, 5.0,-3.0,-1.0},
{-1.0,-3.0, 5.0, 5.0, 5.0,-3.0,-1.0},
{ 0.0,-2.0,-3.0,-3.0,-3.0,-2.0, 0.0},
{ 0.0, 0.0,-1.0,-1.0,-1.0, 0.0, 0.0}

컬러 이미지 효과

```
double* hsi=RGB2HSI(R, G, B);
H = hsi[0]; S=hsi[1]; I = hsi[2];
S = S - 0.2;
if (S < 0)
   S = 0.0;
unsigned char* rgb=HSI2RGB(H, S, I);
```

```
if (8 <= H && H <= 30) {
m_outImageR[i][k] = m_inImageR[i][k];
m_outImageG[i][k] = m_inImageG[i][k];
m_outImageB[i][k] = m_inImageB[i][k];}
else {
double avg = (m_inImageR[i][k]+m_inImageG[i][k]
+m_inImageB[i][k]) / 3.0;
m_outImageR[i][k] = m_outImageG[i][k] =
m_outImageB[i][k] = (unsigned char)avg;}
```

마치면서.....

좋았던 점 : 흑백영상만 하다가 컬러 영상을 구현한 점

아쉬운 점 : 컬러 이미지 효과를 다양하게 넣어보지 못한 것

향후 계획 : 다양한 컬러 이미지 효과 구현