

# OpenCV 없이 구현한 디지털 영상처리 Ver.2.0

[Intel] 엣지 AI SW 아카데미  
객체지향 프로그래밍  
／ using C++, MFC

이름

김정대

# 목차

- 01 프로젝트 개요
- 02 업데이트 사항
- 03 영상처리 알고리즘
- 04 크로마키
- 05 크로마키 보정 알고리즘
- 06 마치며

# 01 프로젝트 개요

## 목표

- OpenCV 없이 디지털 영상처리 프로그램 구현
- 화소점 처리, 기하학 처리, 히스토그램 처리, 화소영역 처리 알고리즘 구현

## 개발 환경

- OS: Windows 11, Linux
- Tool: Visual Studio 2022 (64-bit), Pycharm 2023, MFC
- 언어: C, Python, C++

# 01 프로젝트 개요 - 전체적 구성



픽셀 데이터 로드



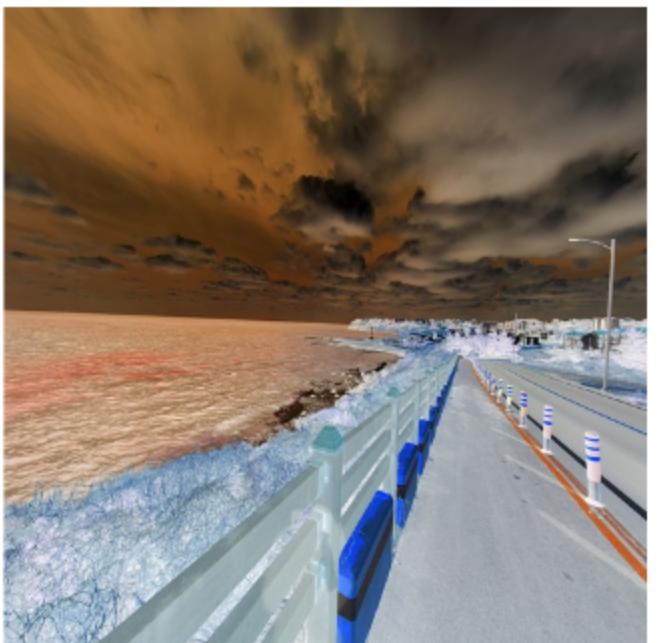
220	46	54	86	186
244	4	206	239	186
194	161	142	215	189
249	230	239	88	195
4	106	171	188	121

메모리 저장



영상 처리 알고리즘

```
void reverseImage() {  
    reallocOutputMemory(inH, inW);  
    for (int i = 0; i < inH; i++) {  
        for (int k = 0; k < inW; k++) {  
            outImage[i][k] = 255 - inImage[i][k];  
        }  
    }  
    printImage();  
}
```



화면 출력



35	209	201	169	69
11	251	49	16	69
61	94	113	40	66
6	25	16	167	60
251	149	84	67	134

메모리 저장



# 02 업데이트 사항

## 1. 칼라 이미지 모델

- 기존의 그레이 스케일 모델이 아닌 RGB 칼라 이미지로 변경

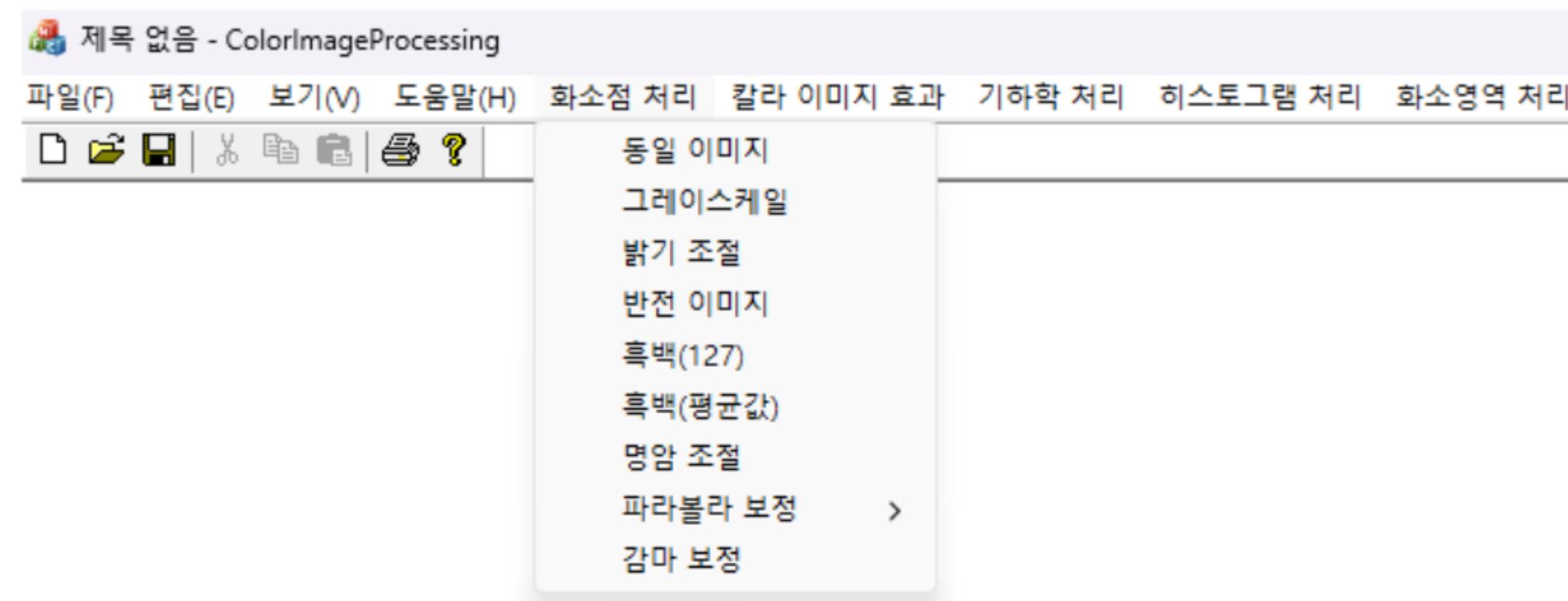


# 02 업데이트 사항

## 2. UI 구현 (feat.MFC)

- 기존의 터미널에서 실행하지 않고 MFC를 활용한 UI 구현

```
## 화소 점 처리 ##
0. 열기 1. 저장 9. 뒤로
A. 동일 B. 밝게 C. 어둡게 D. 반전 E. 흑백(127)
F. 흑백(평균) G. 감마 H. 곱셈 I. 나눗셈
J. 파라cap K. 파라cup L. AND M. OR N. XOR
```



# 02 업데이트 사항

## 3. 크로마키

- HIS 색 탐지를 활용하여 크로마키 기능 구현
- 영상처리 알고리즘을 활용하여 노이즈 보정 기능 구현



# 03 영상처리 알고리즘 - 화소점 처리

- 화소 점의 원래 값이나 위치를 기준으로 화소 값을 변경
- 산술연산, 논리연산 등



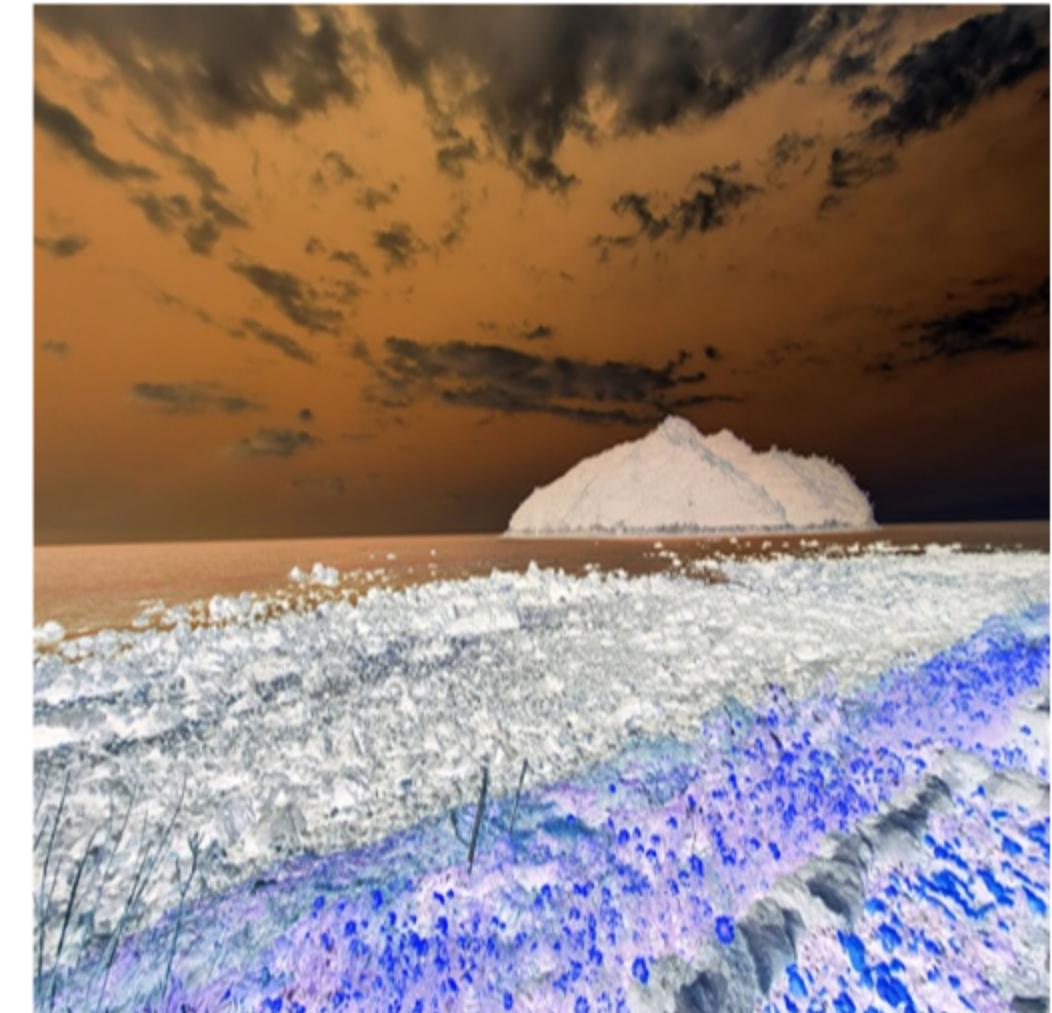
# 03 영상처리 알고리즘 - 화소점 처리



동일 이미지

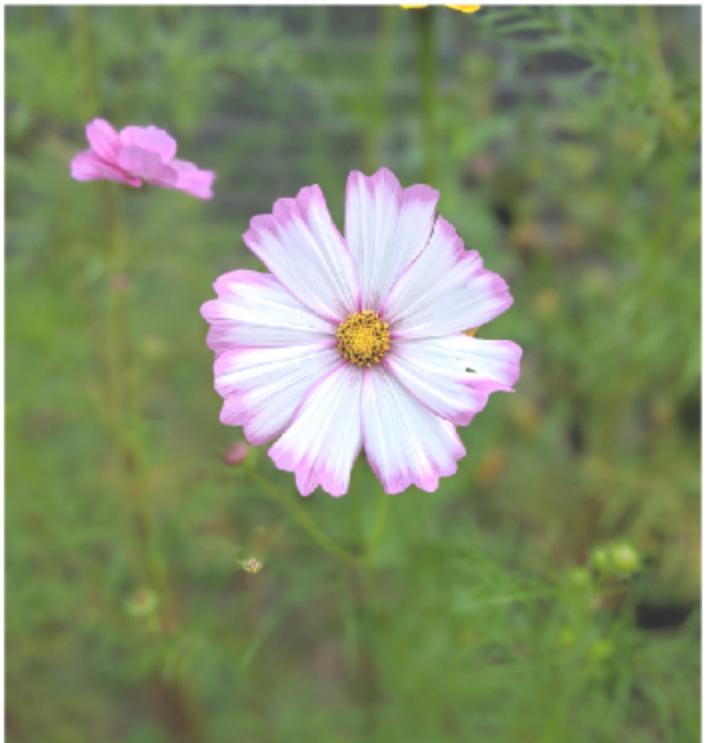


흑백 이미지



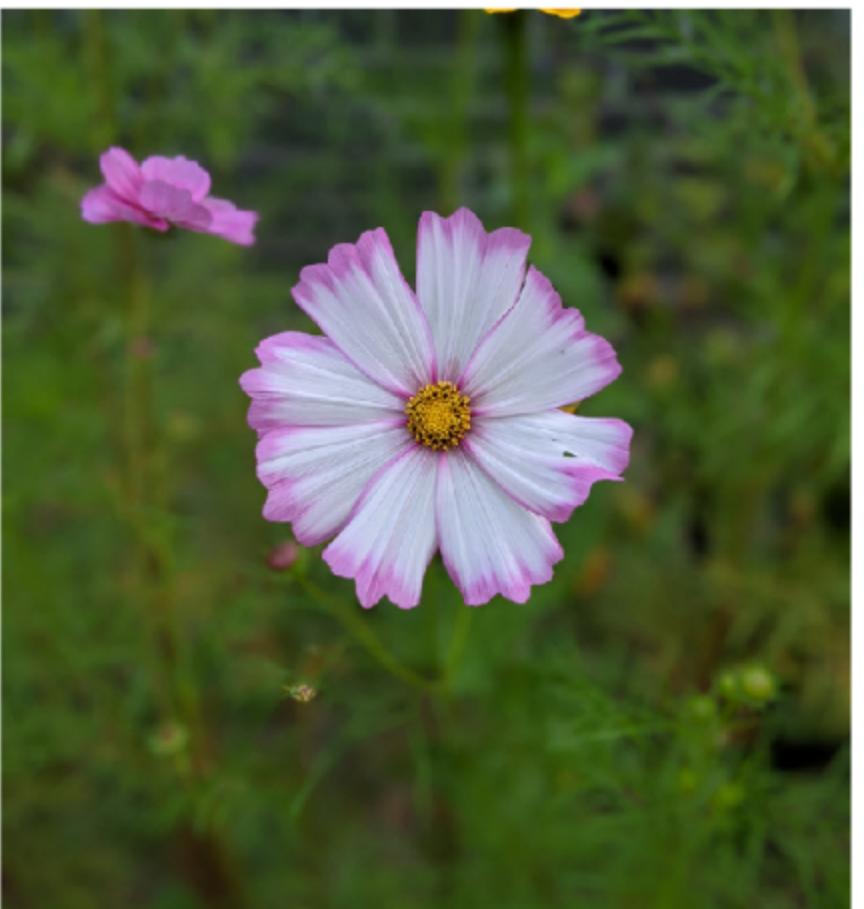
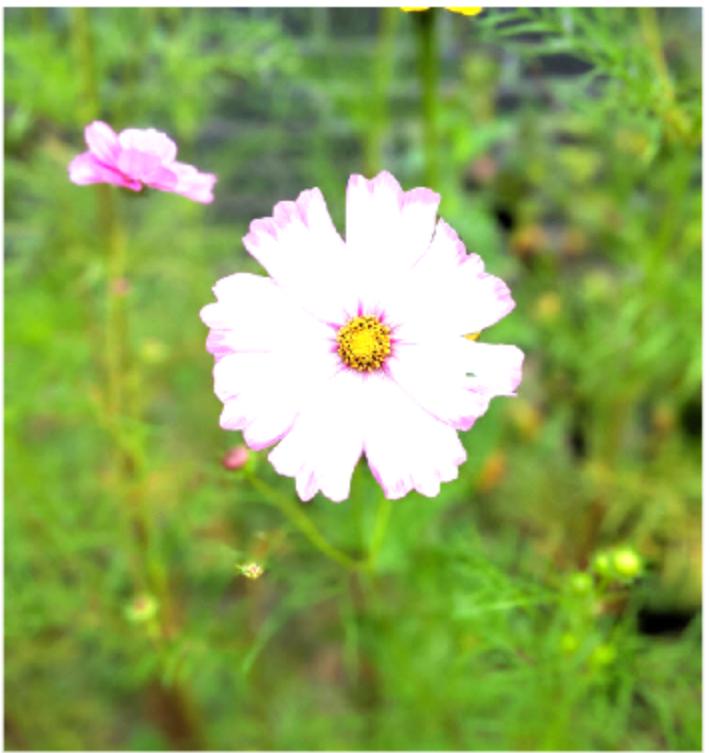
반전 이미지

# 03 영상처리 알고리즘 - 화소점 처리



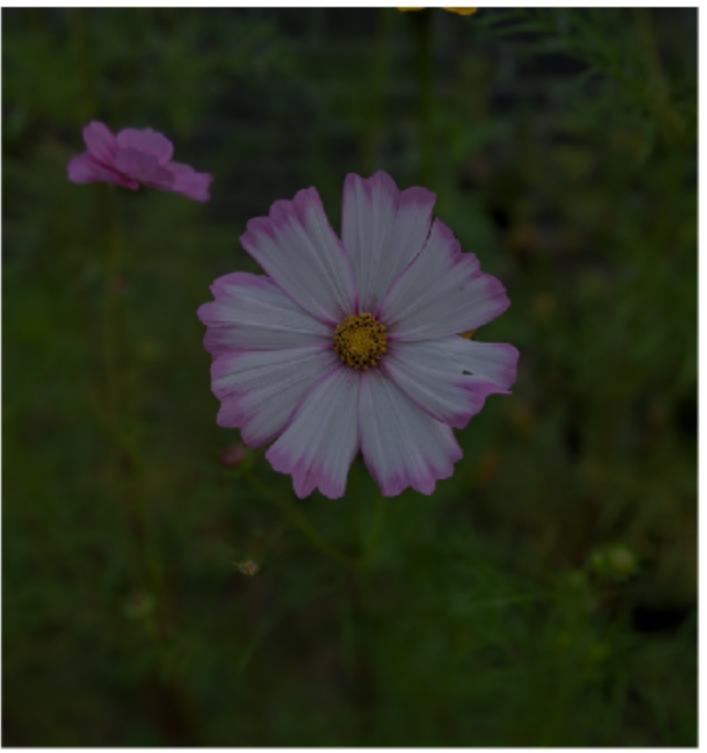
+50

x2



-50

÷2



# 03 영상처리 알고리즘 - 화소점 처리

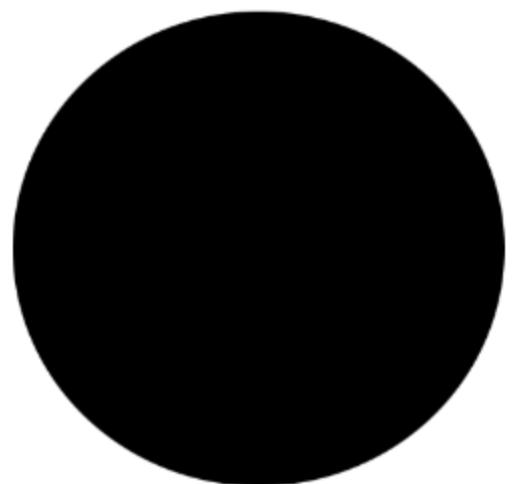
inImage



outImage



mask



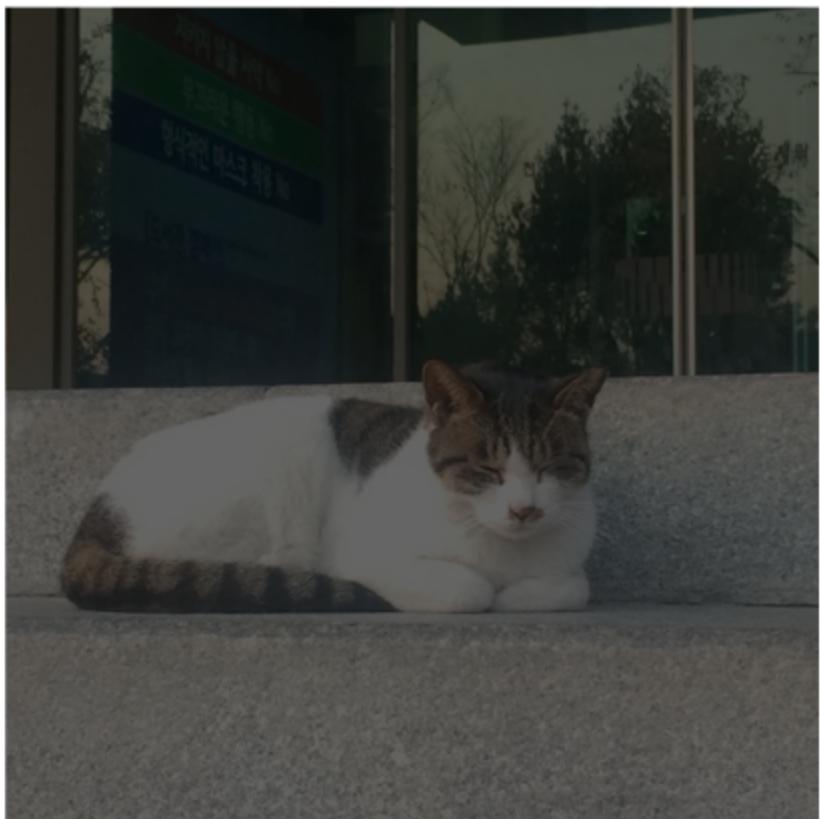
AND

OR

XOR

# 03 영상처리 알고리즘 - 화소점 처리

## 감마 보정



gamma  
 $= 1.2$

## 파라볼라 보정



gamma  
 $= 0.8$

cap

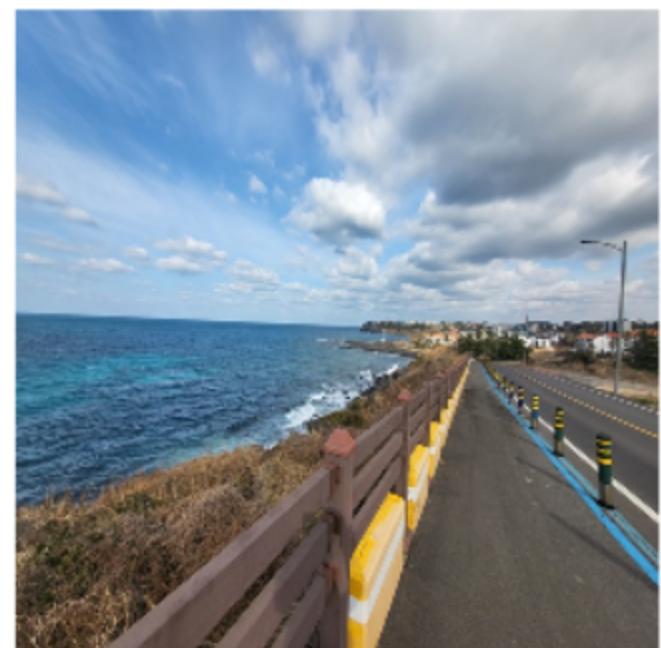
cup

# 03 영상처리 알고리즘 - 기하학 처리

- 화소의 위치나 화소의 모임인 배열을 변화
- 확대, 축소, 회전, 이동 등



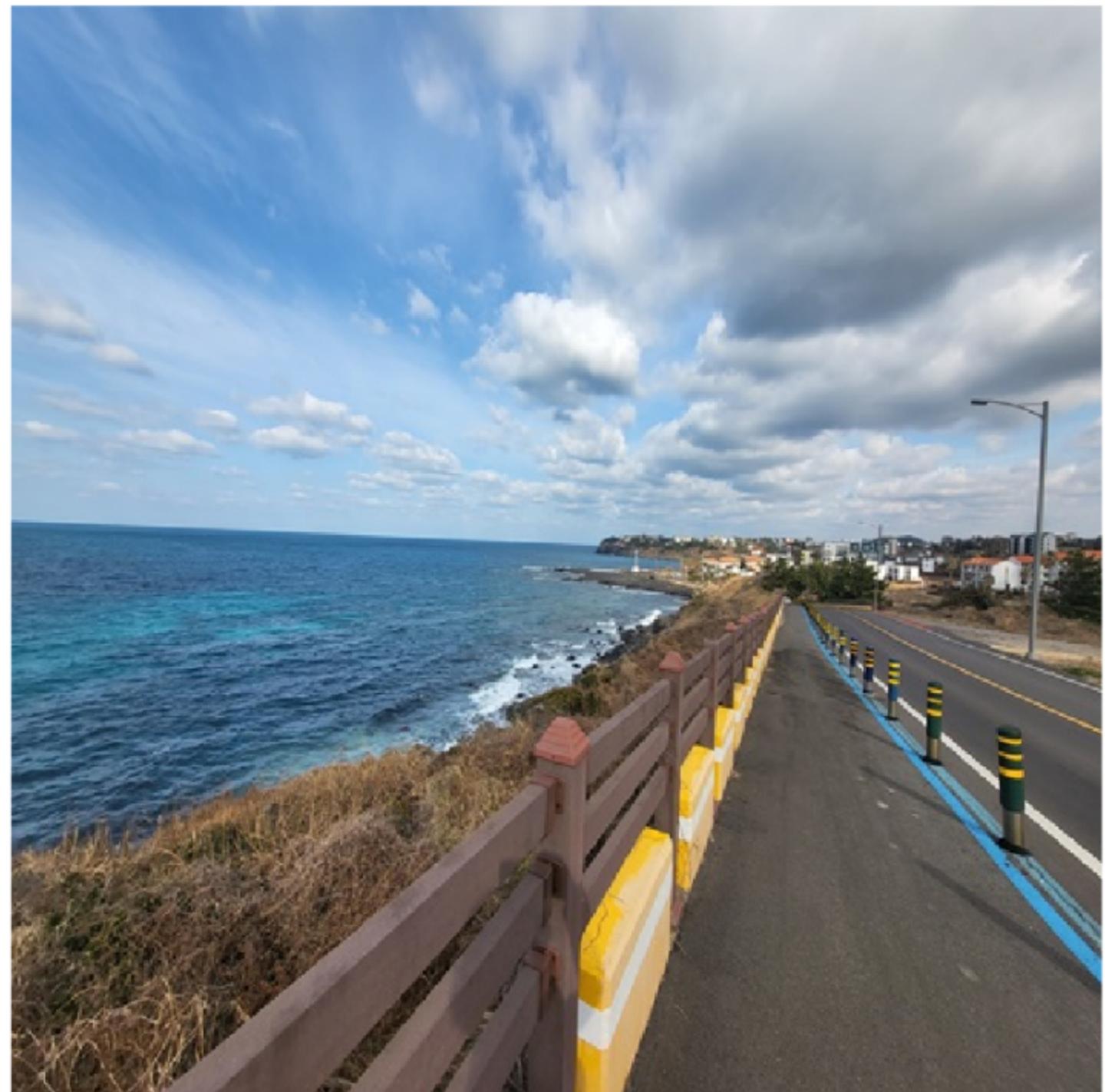
# 03 영상처리 알고리즘 - 기하학 처리



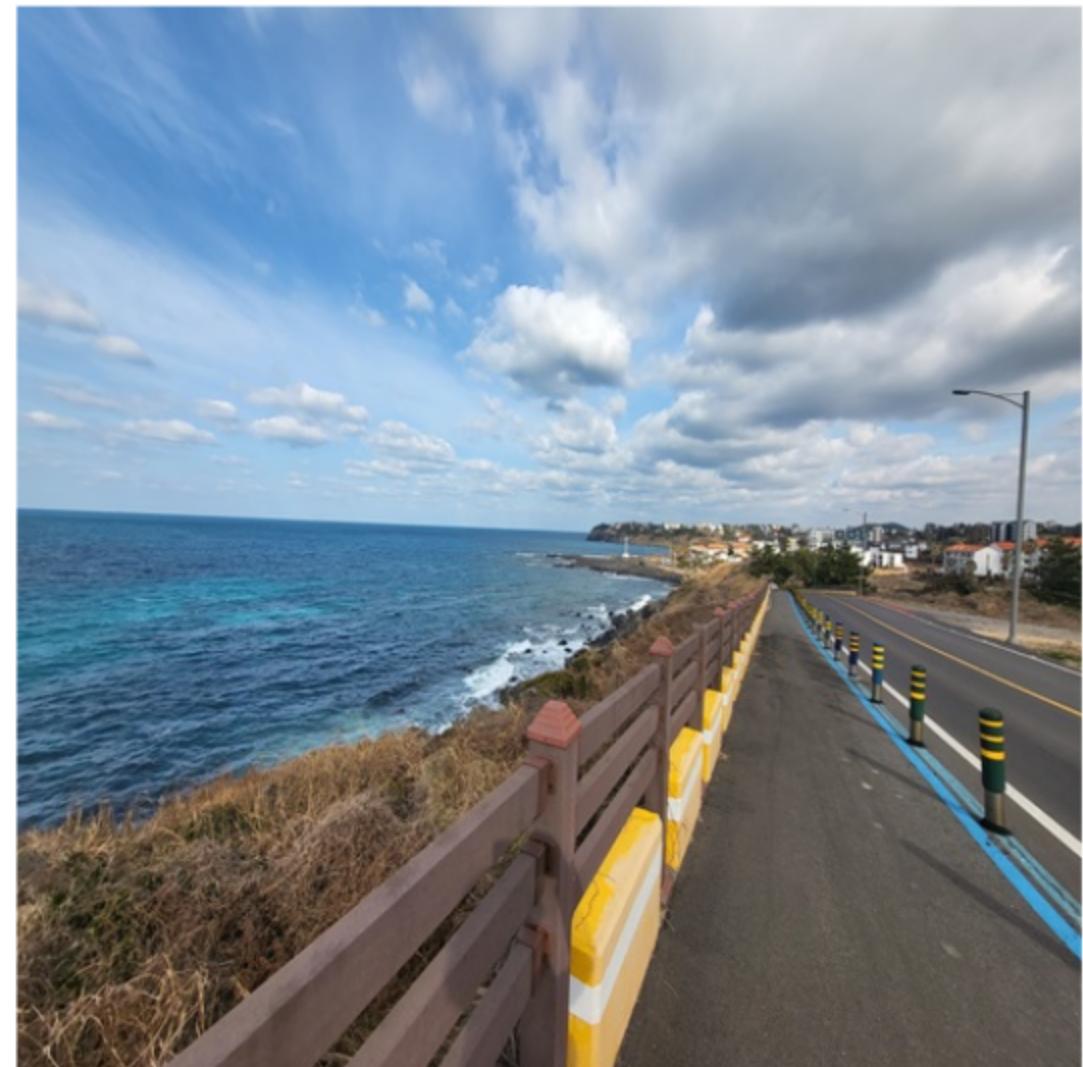
2배 축소  
←



→ 2배 확대



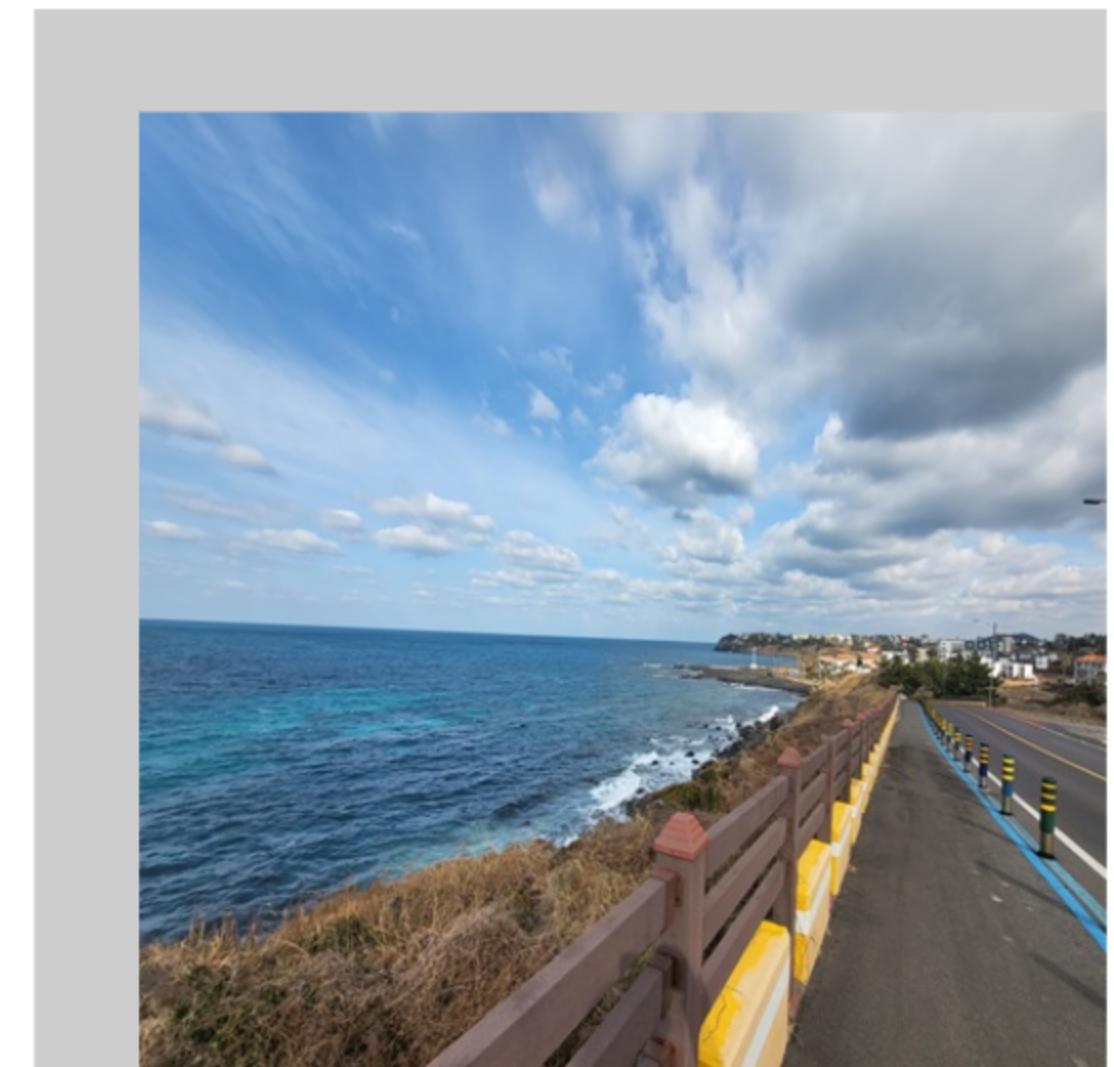
# 03 영상처리 알고리즘 - 기하학 처리



원본

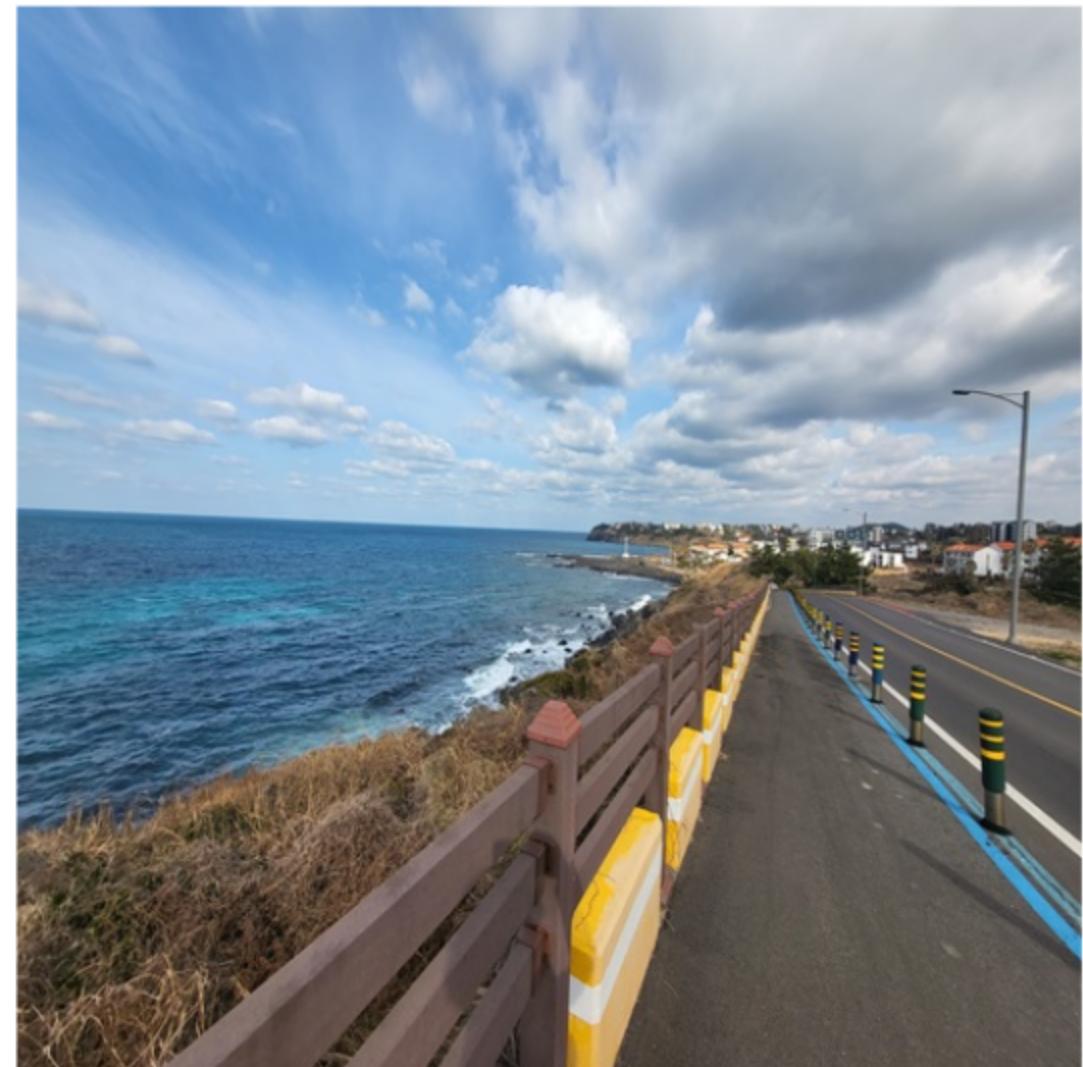


회전



이동

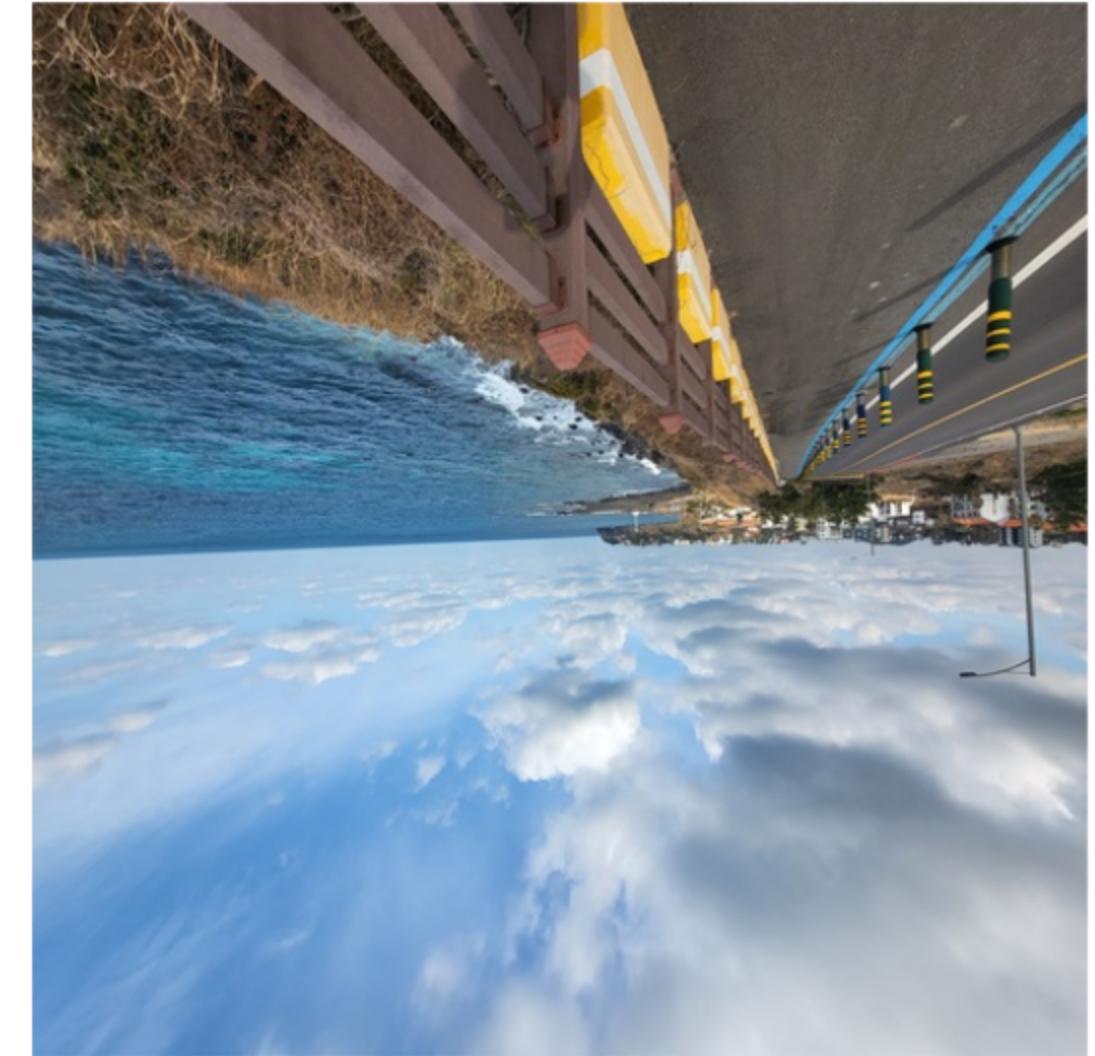
# 03 영상처리 알고리즘 - 기하학 처리



원본



좌우대칭



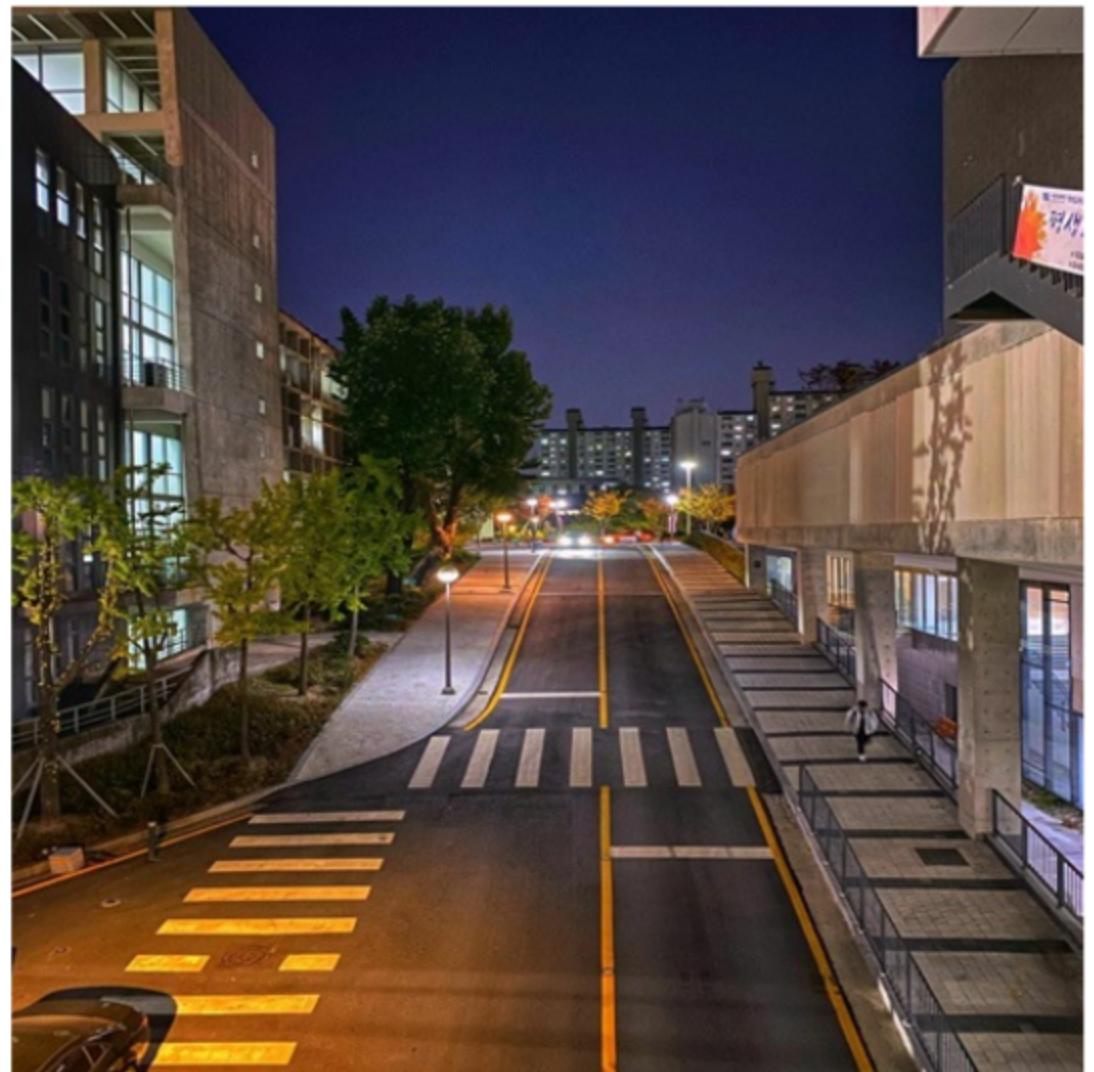
상하대칭

# 03 영상처리 알고리즘 - 히스토그램 처리

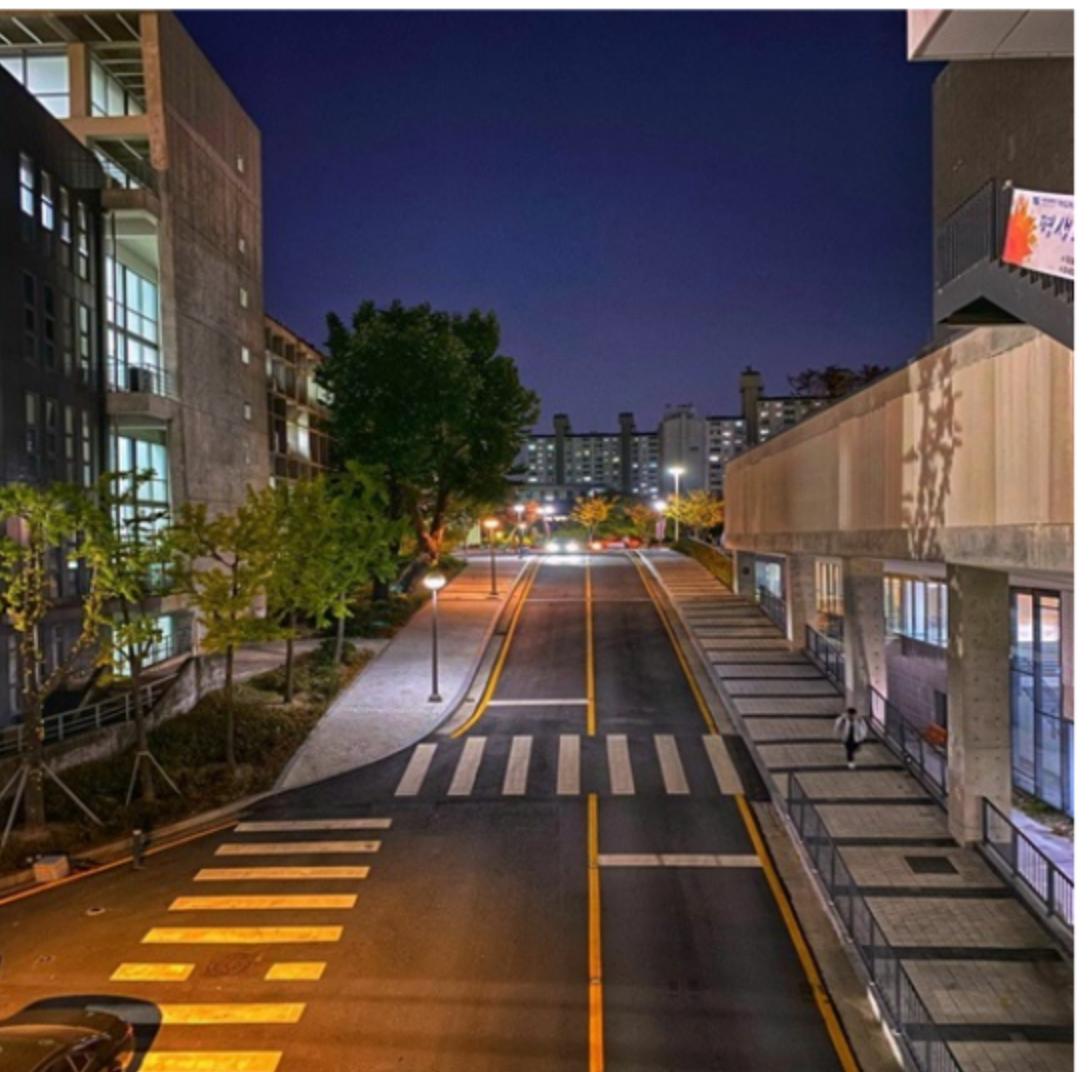
- 모든 범위의 화소값을 포함하도록 히스토그램 분포 확장
- 명암비가 낮은 영상의 명암비 향상



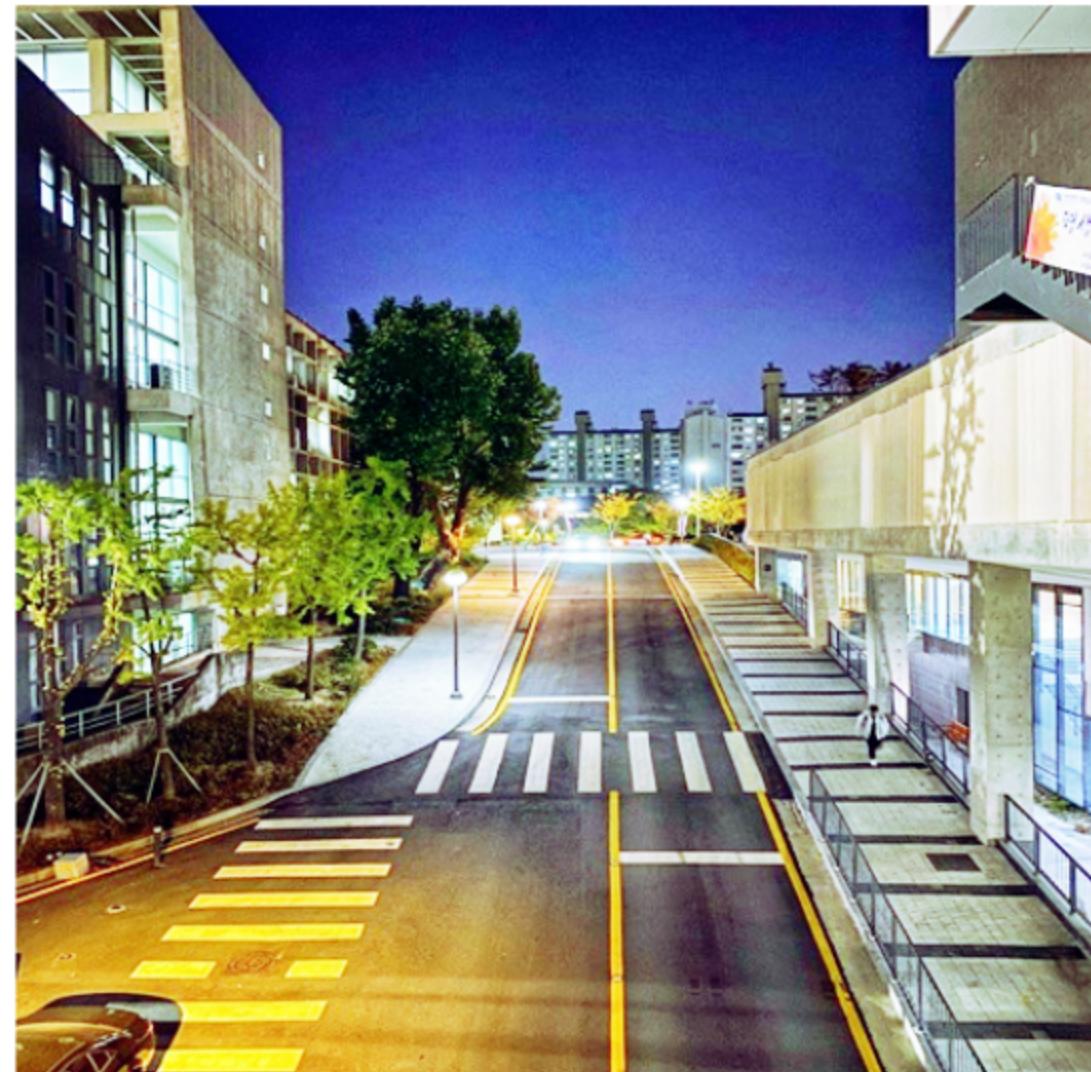
# 03 영상처리 알고리즘 - 히스토그램 처리



명암대비 스트레칭



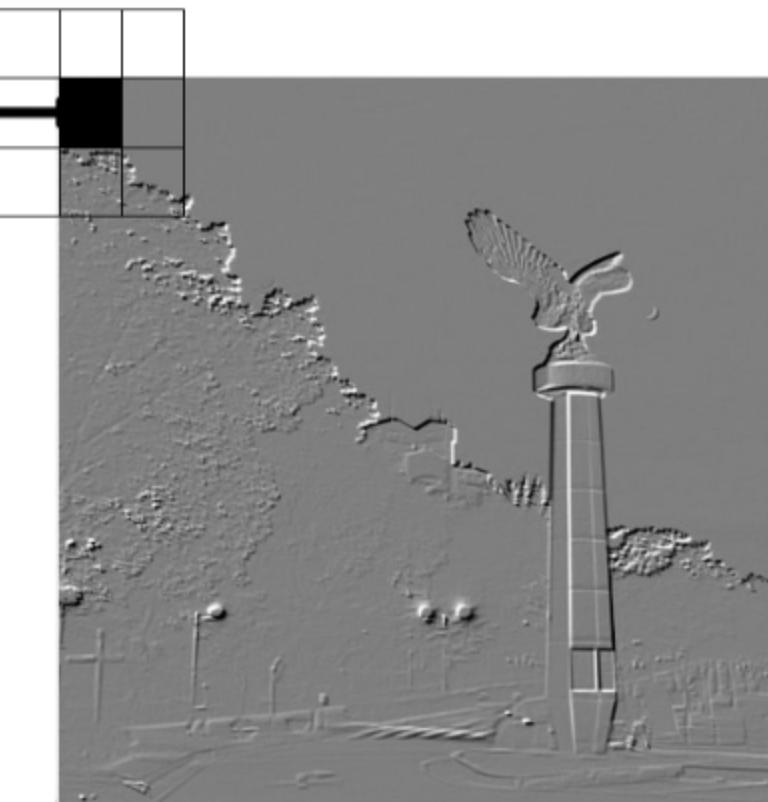
엔드-인



평활화

# 03 영상처리 알고리즘 - 화소영역 처리

- 입력된 화소의 값과 위치를 기반으로 주변 화소 값을 고려하여 조정
- 회선 기법으로 영역 처리



# 03 영상처리 알고리즘 - 화소영역 처리

## 회선 기법

- 이웃한 화소의 가중치 합을 출력화소로 생성
- 가중치를 포함한 마스크가 이동하면서 수행

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

- **Output\_pixel[x, y]**: 회선 처리로 출력한 화소
- **I[m, n]**: 입력 영상의 화소
- **M[m, n]**: 입력 영상의 화소에 대응하는 가중치

### # 마스크 설정

```
mask = [[-1, 0, 0],  
        [ 0, 0, 0],  
        [ 0, 0, 1]]
```

### # 임시 메모리 할당

```
tmpInImage = malloc2D(inH + 2, inW + 2)
```

### # 회선 처리

```
S += tmpInImage[i+m][k+n] * mask[m][n]  
tmpOutImage[i][k] = S  
outImage[i][k] = int(tmpOutImage[i][k])
```

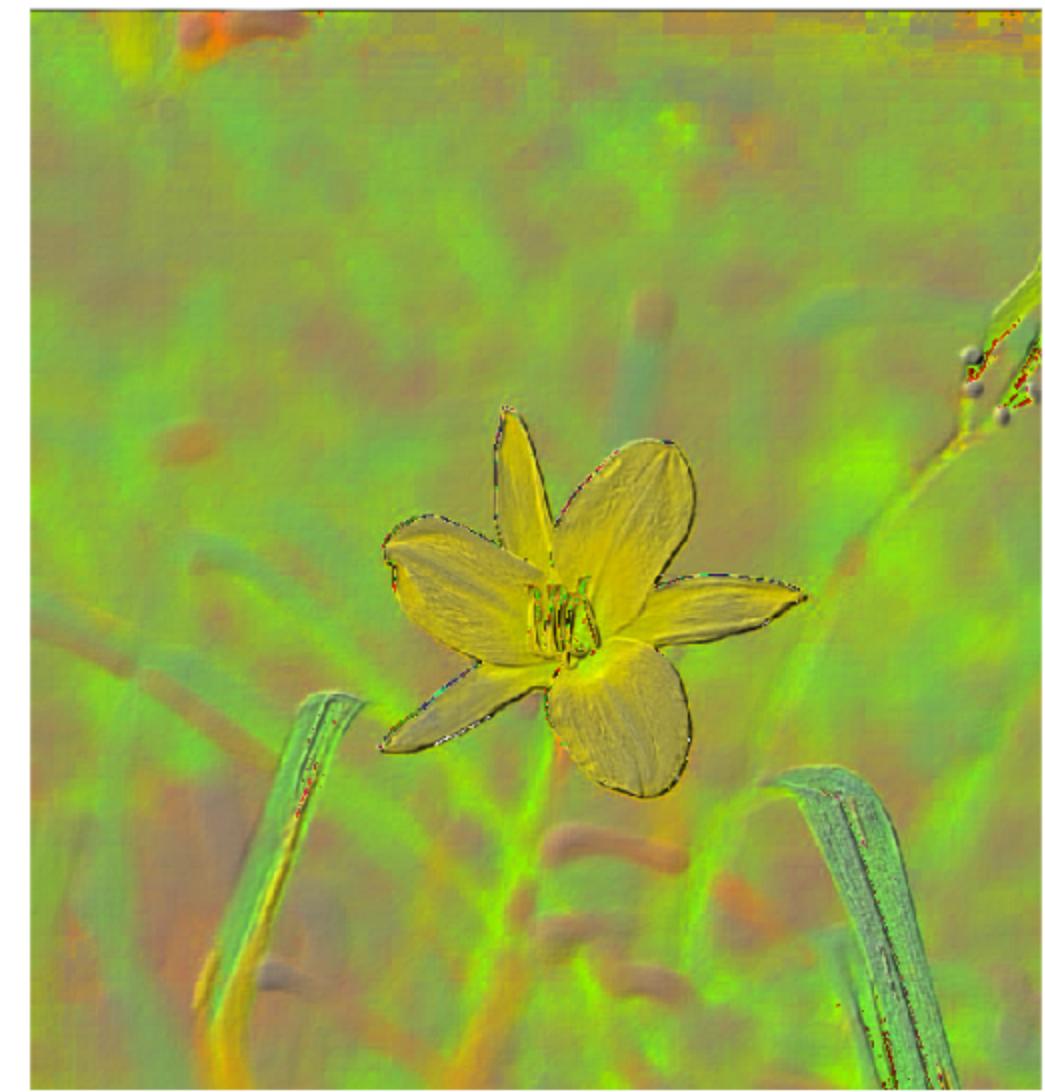
# 03 영상처리 알고리즘 - 화소영역 처리



원본



엠보싱(RGB)



엠보싱(HSI)

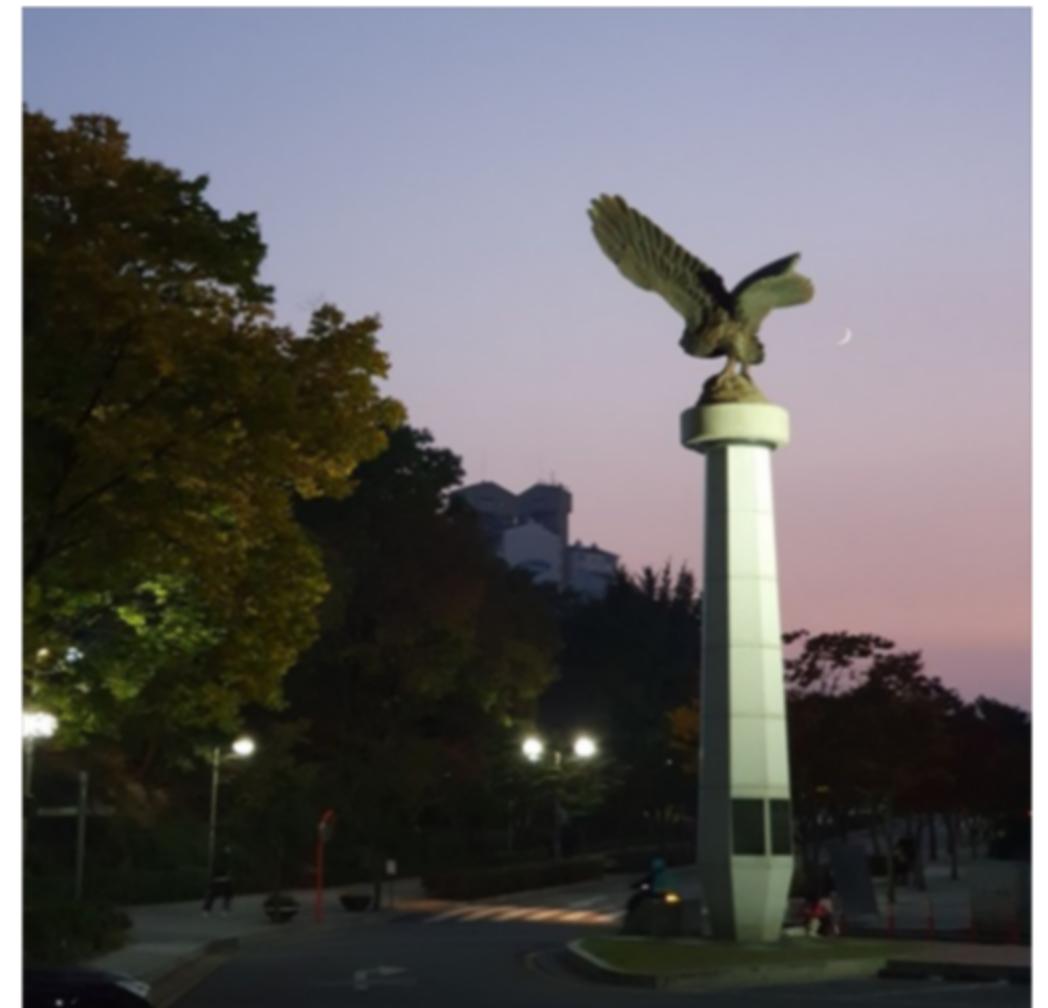
# 03 영상처리 알고리즘 - 화소영역 처리



원본

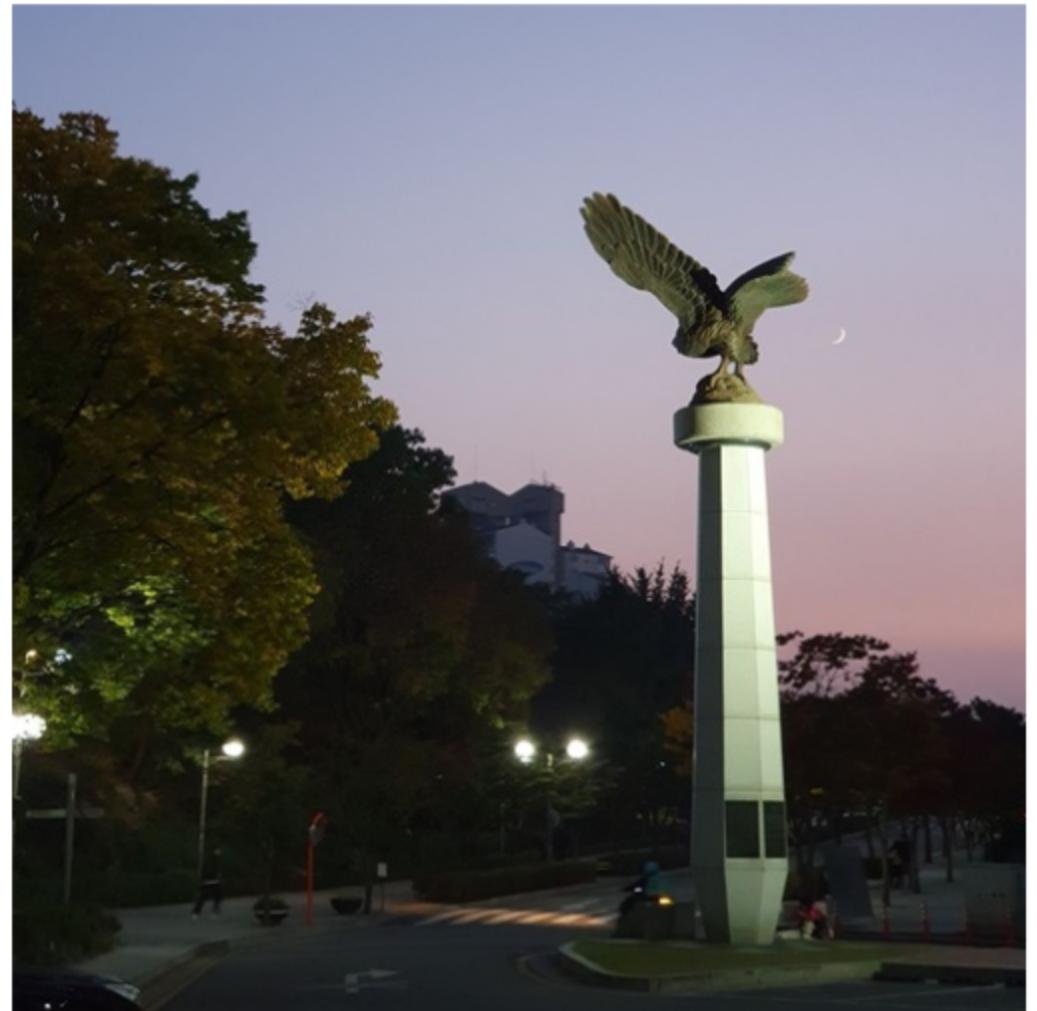


블러링

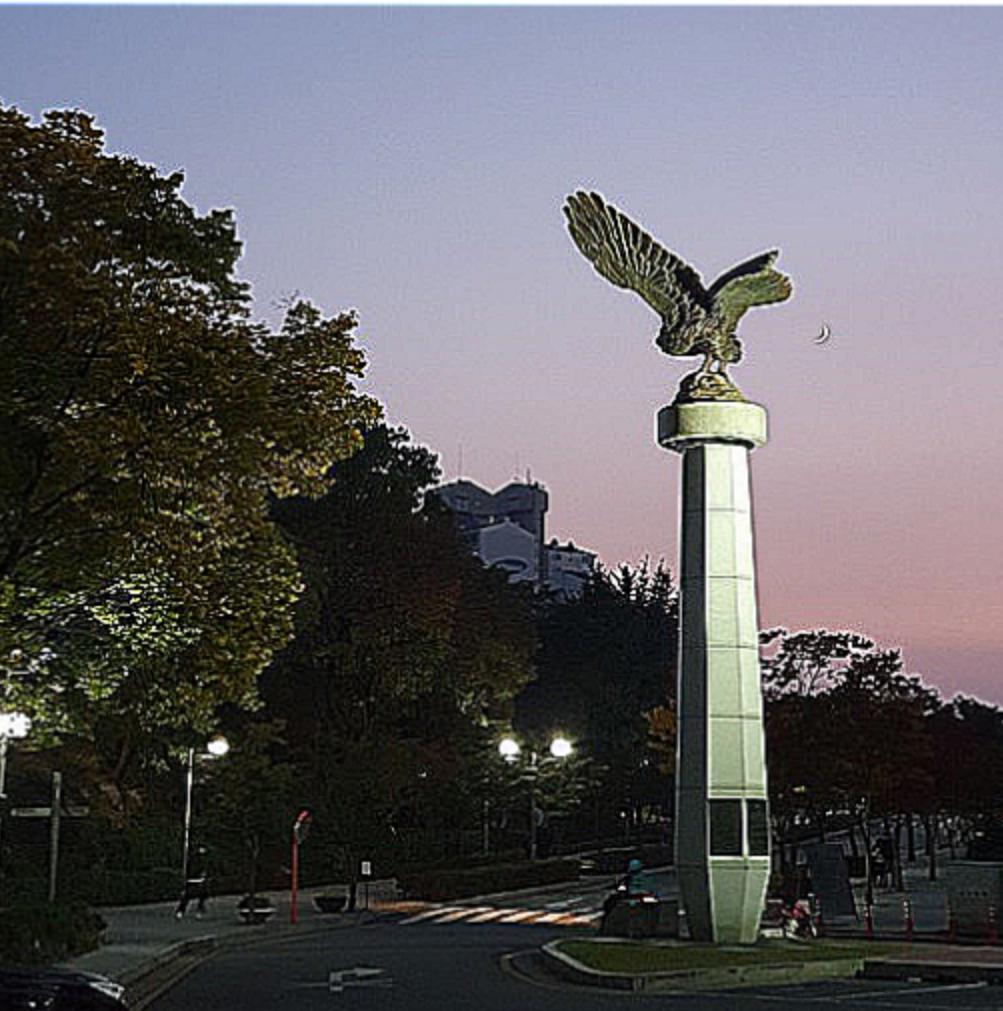


가우시안

# 03 영상처리 알고리즘 - 화소영역 처리



원본

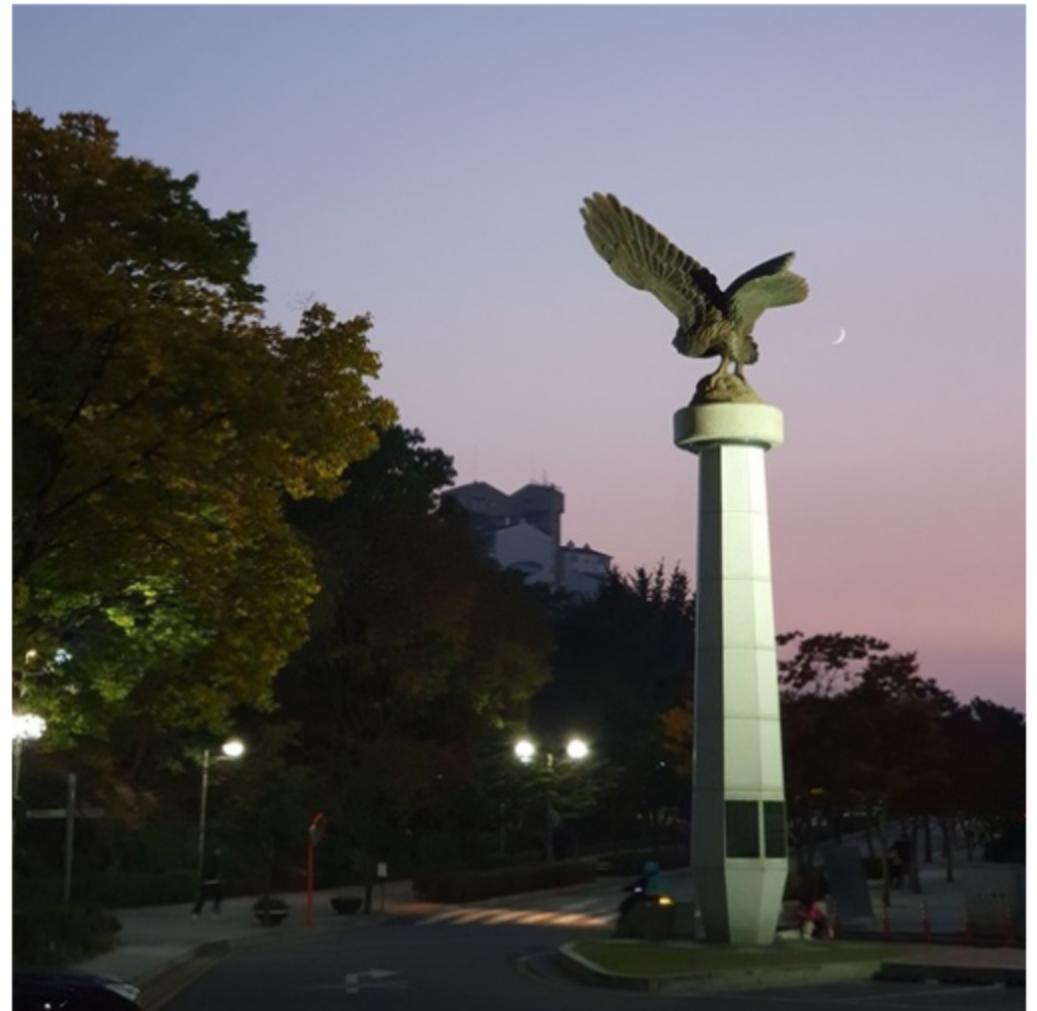


샤프닝



고주파 필터 샤프닝

# 03 영상처리 알고리즘 - 화소영역 처리



원본

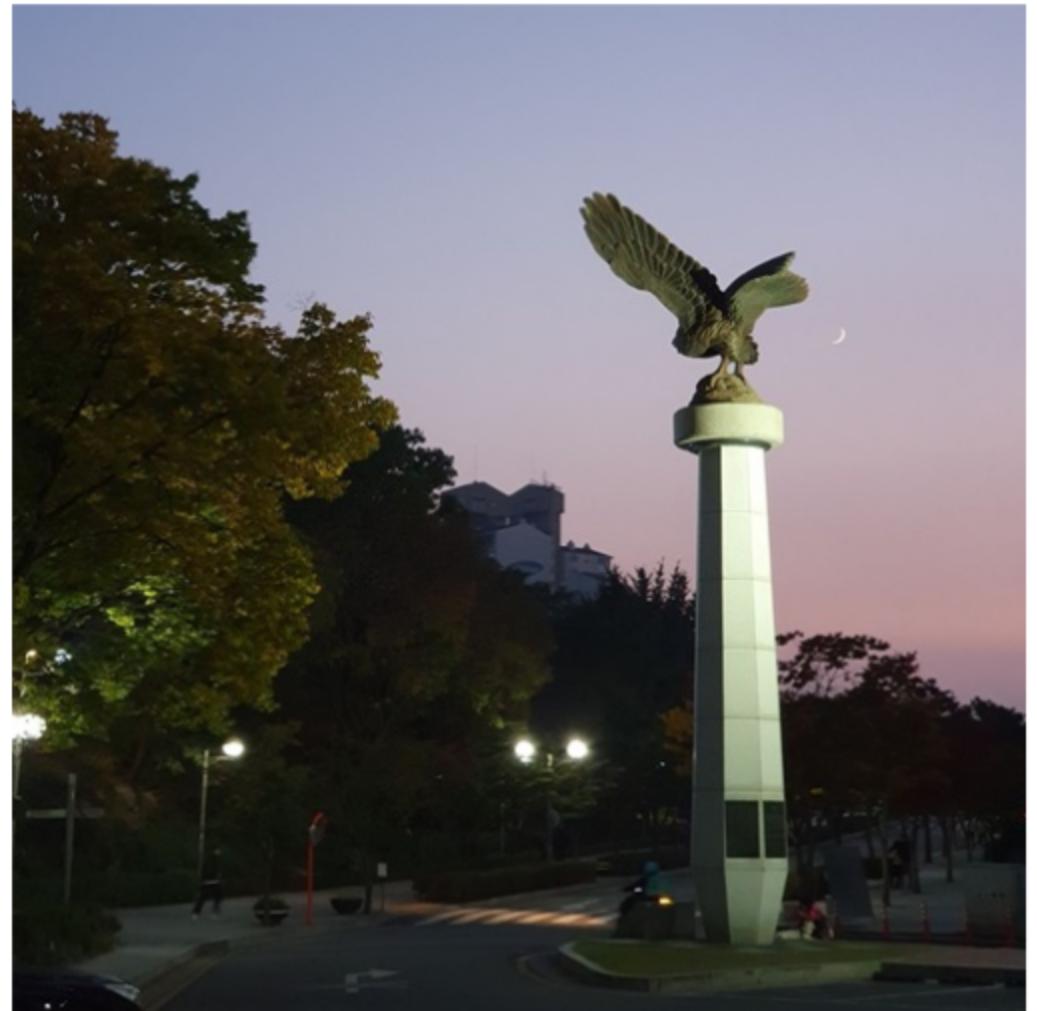


수직 에지



수평 에지

# 03 영상처리 알고리즘 - 화소영역 처리



원본

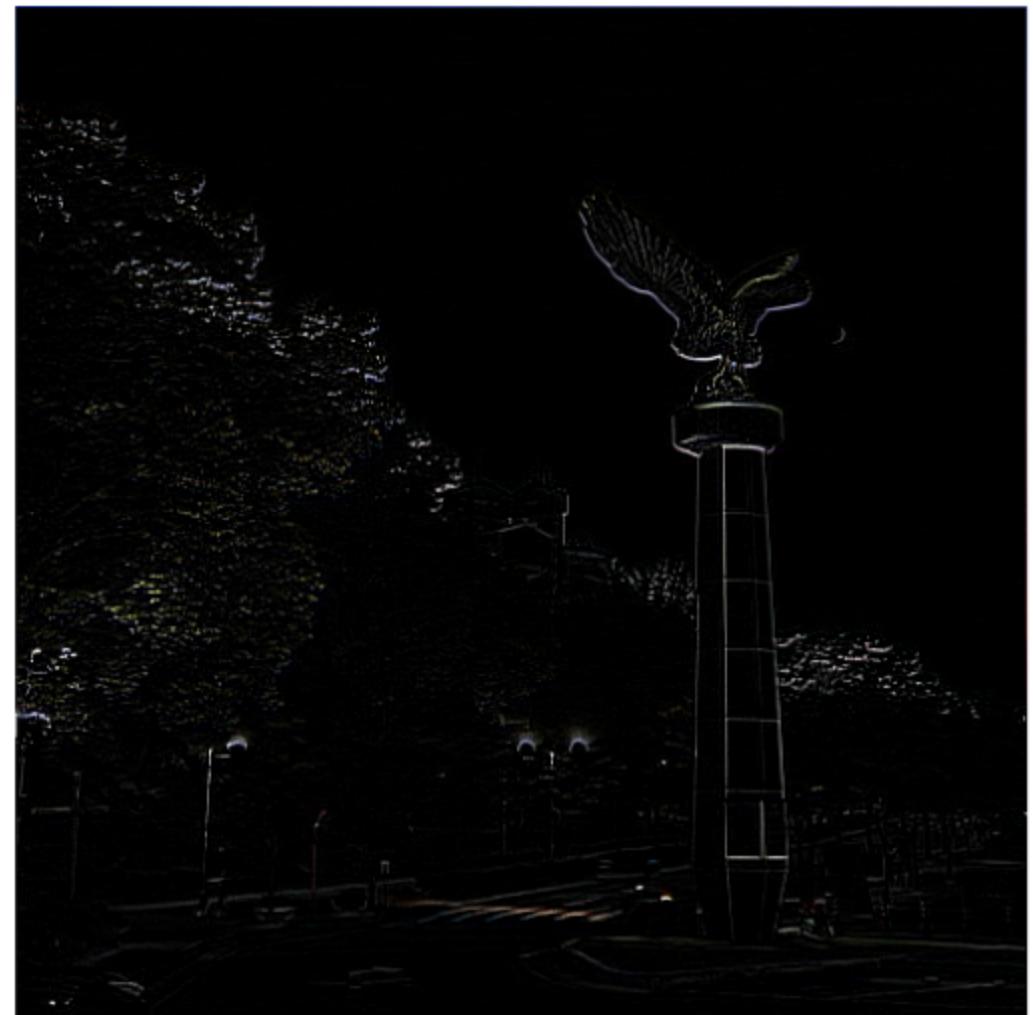


유사 연산자



차 연산자

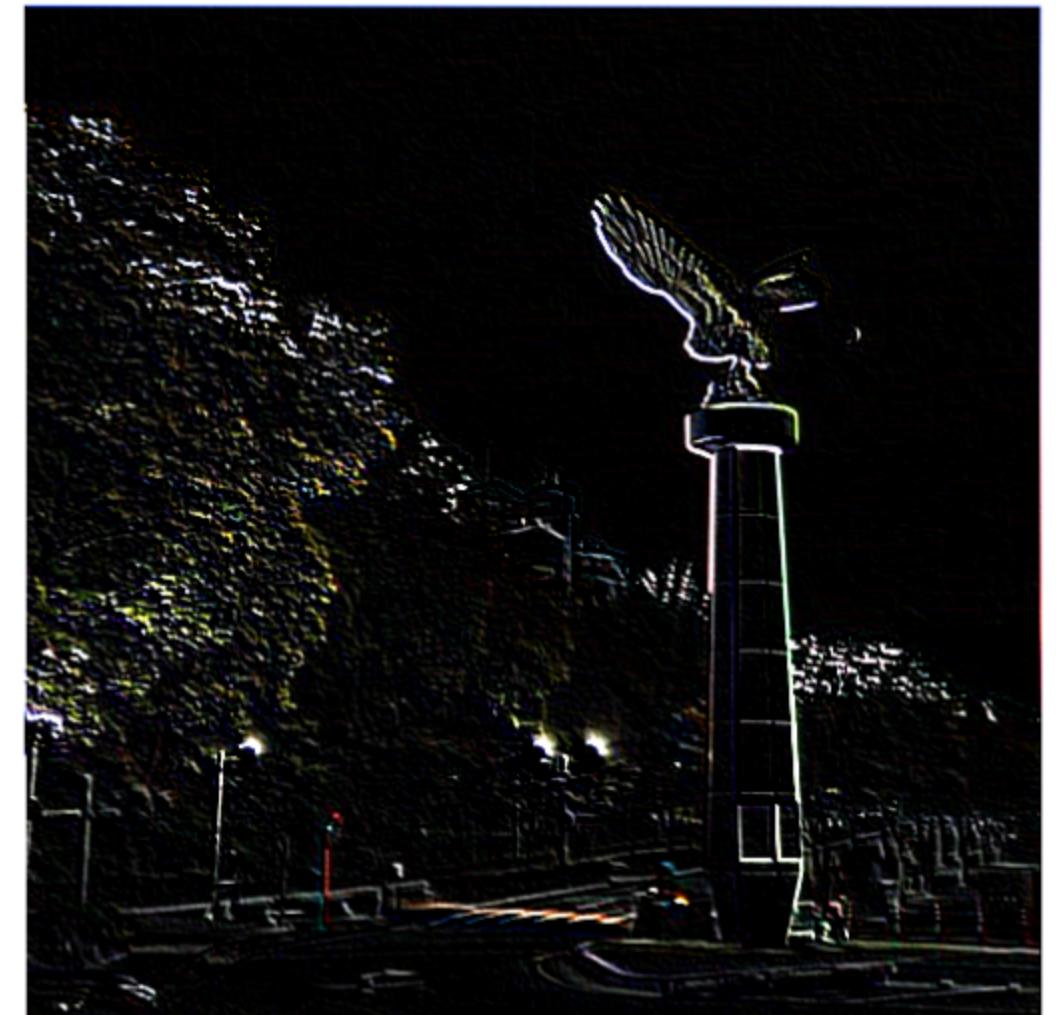
# 03 영상처리 알고리즘 - 화소영역 처리



로버츠



프리윗



소벨

# 03 영상처리 알고리즘 - 화소영역 처리



라플라시안



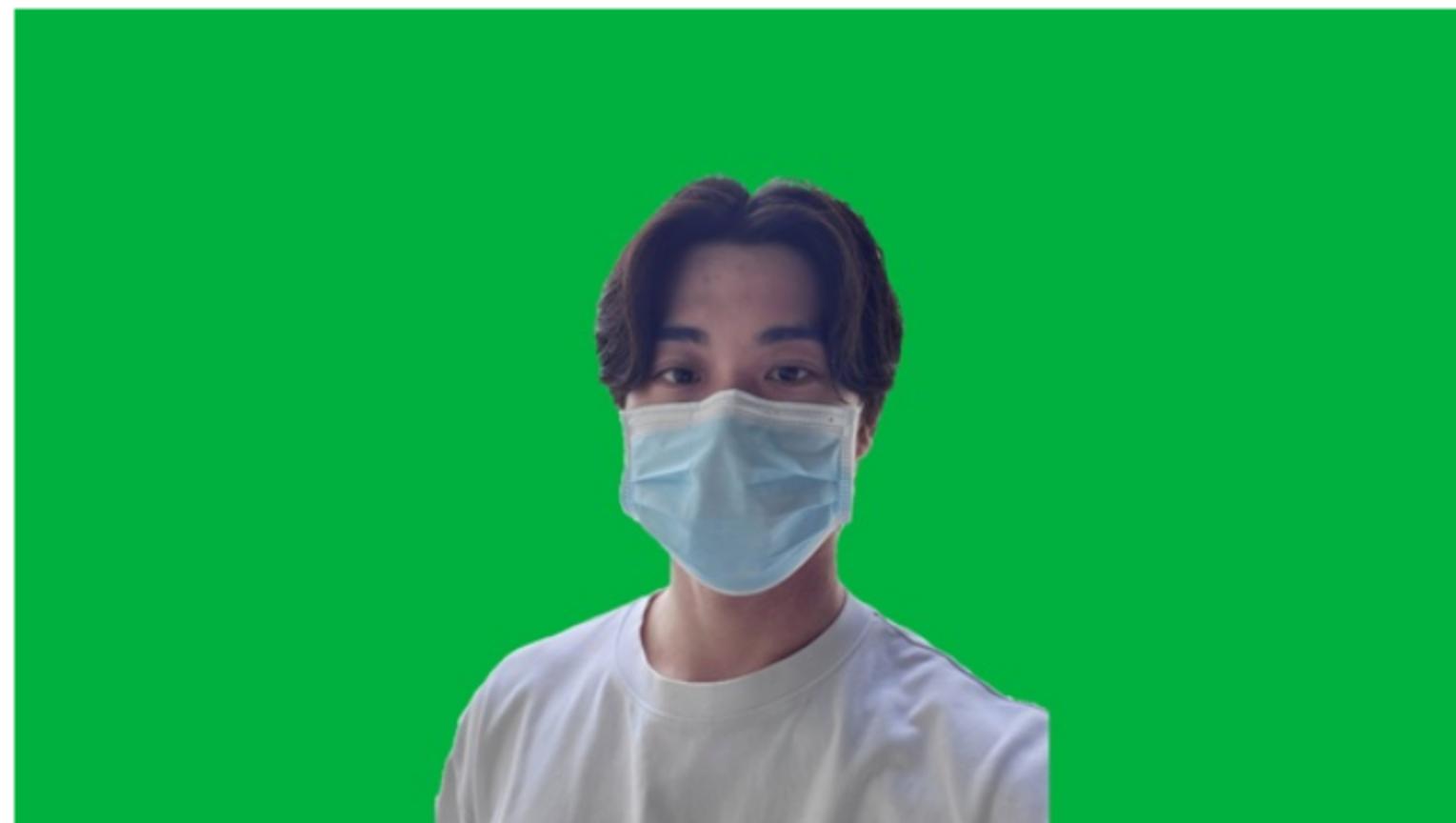
LoG



DoG

# 04 크로마키

- 녹색 배경에서 촬영한 영상에 다른 배경을 합성하는 기술
- 녹색에 해당되는 색 영역을 탐지하여 배경 이미지로 변환



# 04 크로마키 - 알고리즘 구현

## HSI 변환

- 녹색 배경에 해당되는 색 영역을 탐지하기 위함
- 색상 값(Hue)만을 통해 분류하기 때문에 빛의 변화에 효과적

$$H = \cos^{-1} \left[ \frac{0.5[(R-G)+(R-B)]}{\sqrt{(R-G)^2 + (R-B)(G-B)}} \right] \quad (1)$$

$$I = \frac{1}{3}(R + G + B) \quad (2)$$

$$S = 1 - \frac{3}{(R+G+B)} [\min(R, G, B)] \quad (3)$$

# 녹색 색상 최소값

```
const int hue_green_min = 90;
```

# 녹색 색상 최대값

```
const int hue_green_max = 150;
```

# 채도 최소값

```
const double saturation_min = 0.3;
```

# HSI 변환

```
double* hsi = RGB2HSI(R, G, B);
```

```
double H = hsi[0];
```

```
double S = hsi[1];
```

```
double I = hsi[2];
```

# 04 크로마키 - 알고리즘 구현

## 필터링

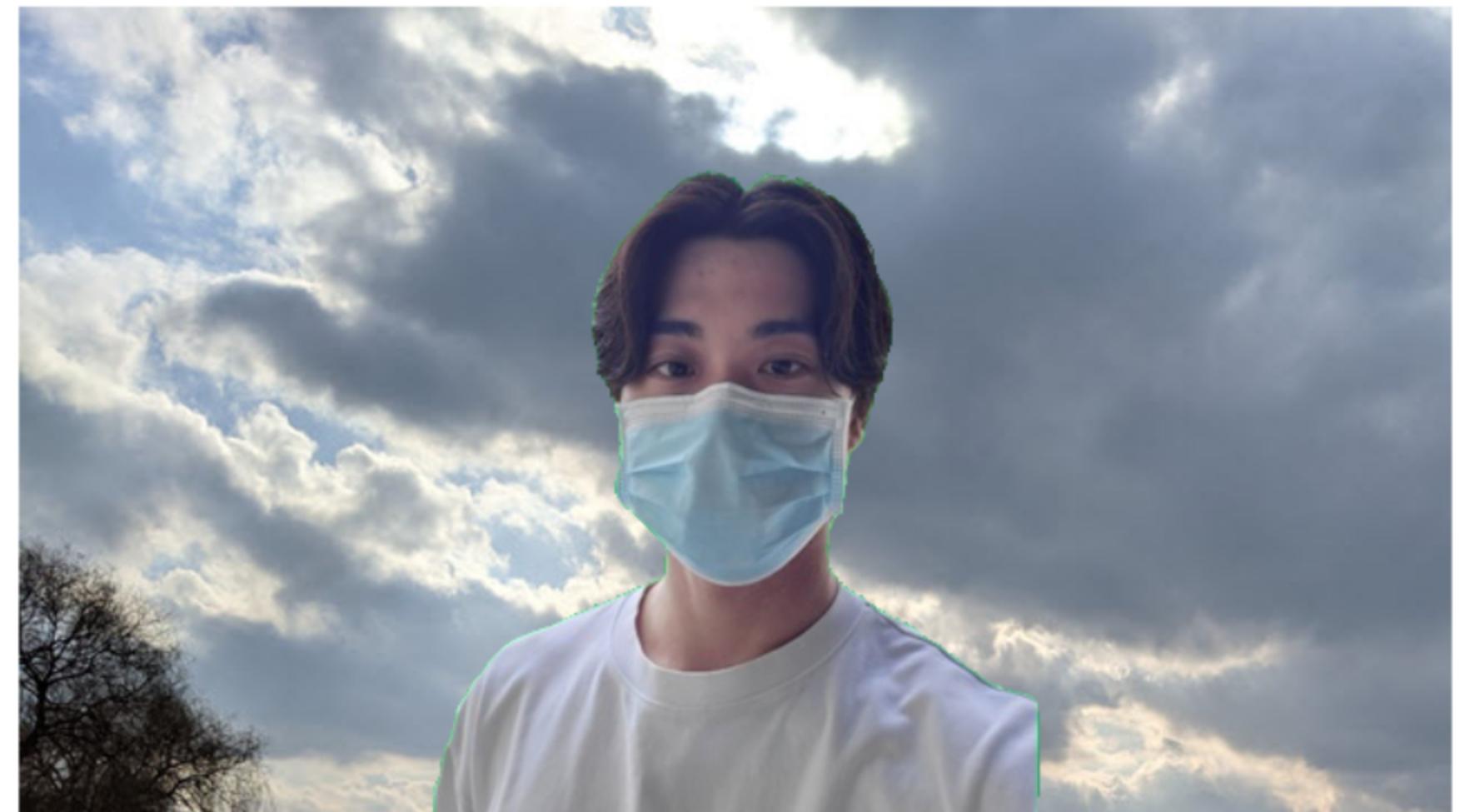
- 녹색 배경에 해당되는 픽셀을 배경 이미지로 대입
- 범위 이외의 픽셀들은 원본 이미지 유지
- 배경 이미지는 CFileDialog를 통해 불러옴

```
if (H >= hue_green_min  
    && H <= hue_green_max  
    && S >= saturation_min) {  
    outImageR = newImageR;  
    outImageG = newImageG;  
    outImageB = newImageB;  
}  
else{  
    outImageR = inImageR;  
    outImageG = inImageG;  
    outImageB = inImageB;  
}
```

# 04 크로마키 - 알고리즘 구현

## 중간 결과

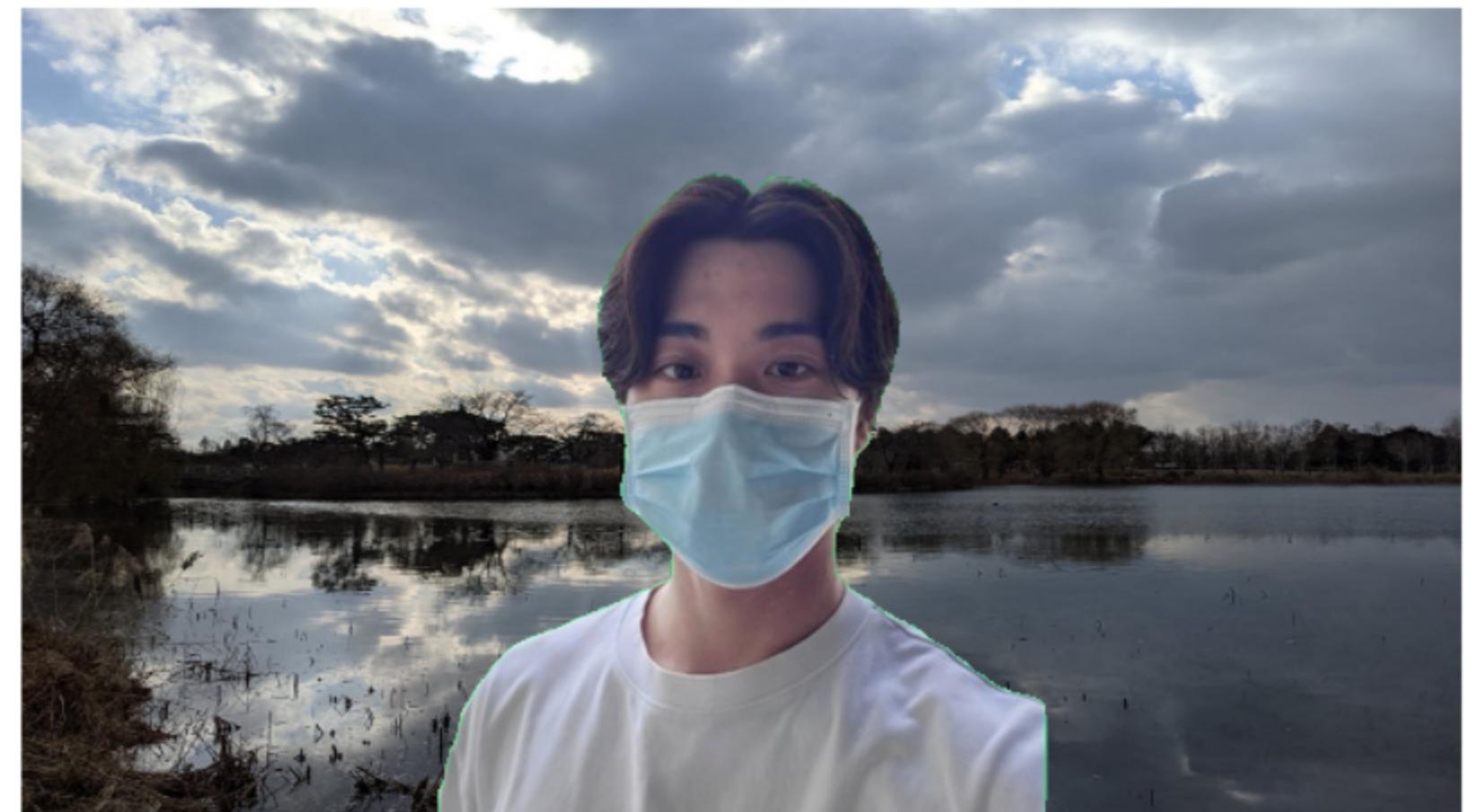
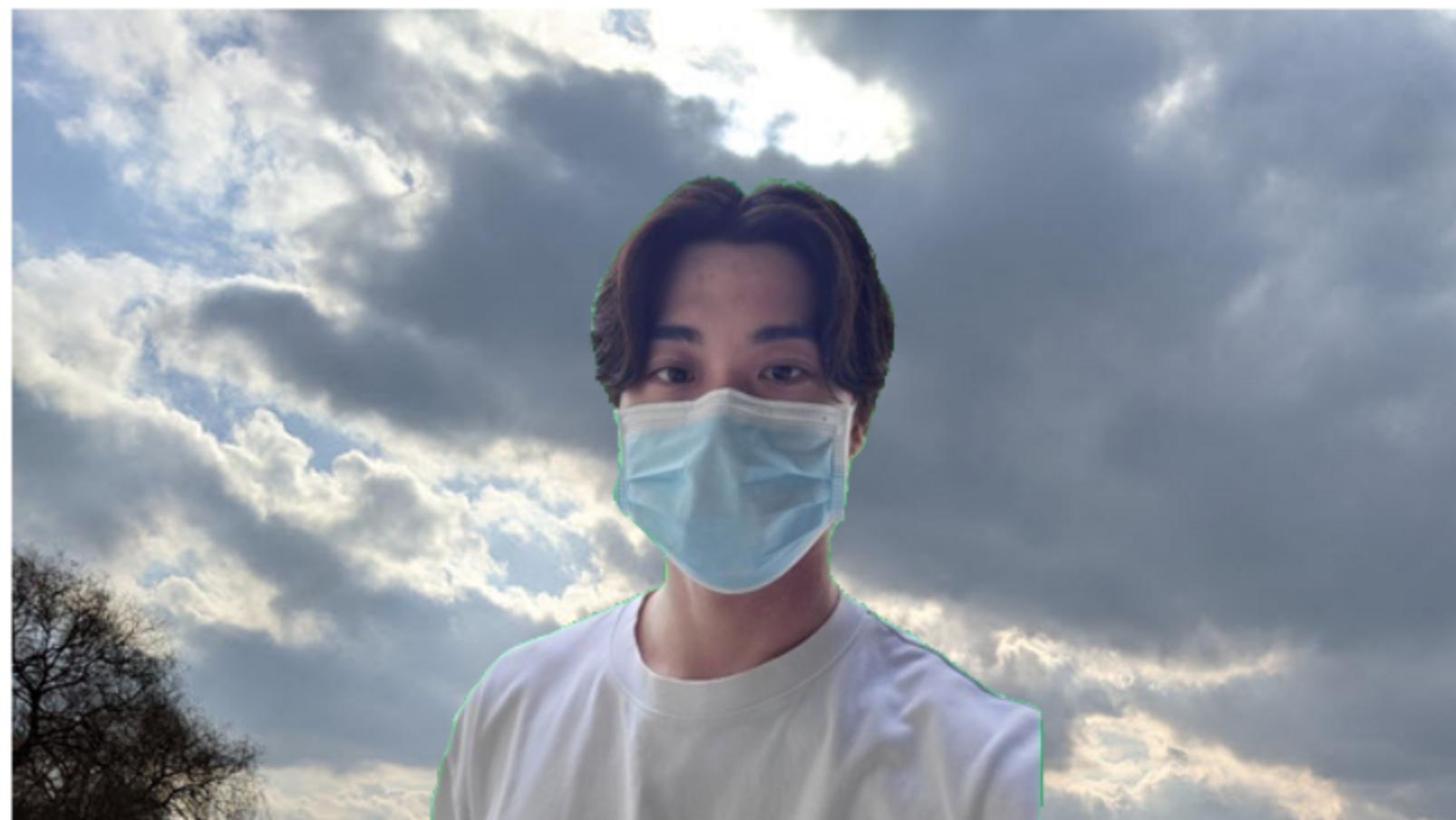
- 크기 호환성 오류
- 객체 주변 노이즈 발생



# 04 크로마키 - 알고리즘 구현

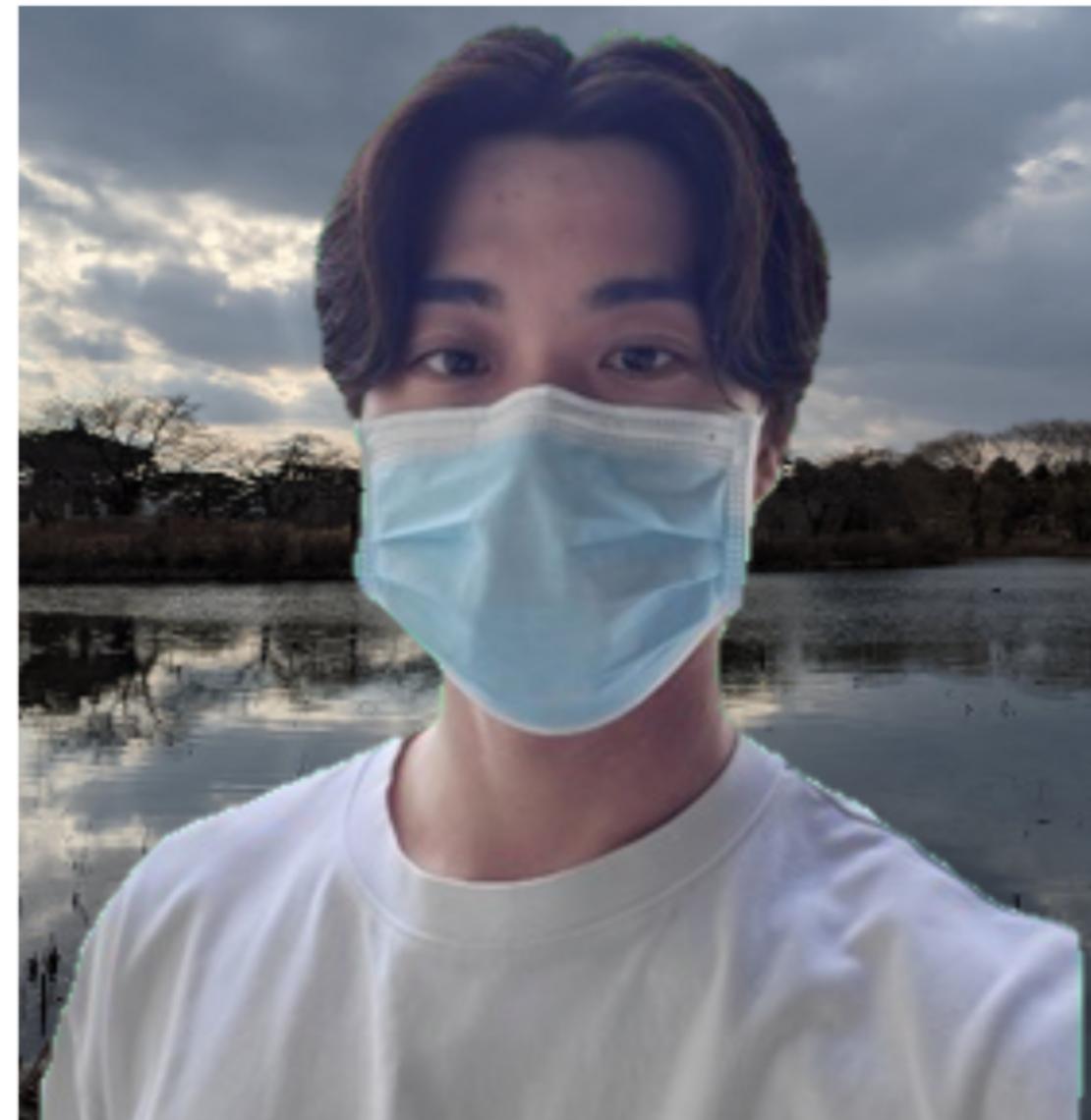
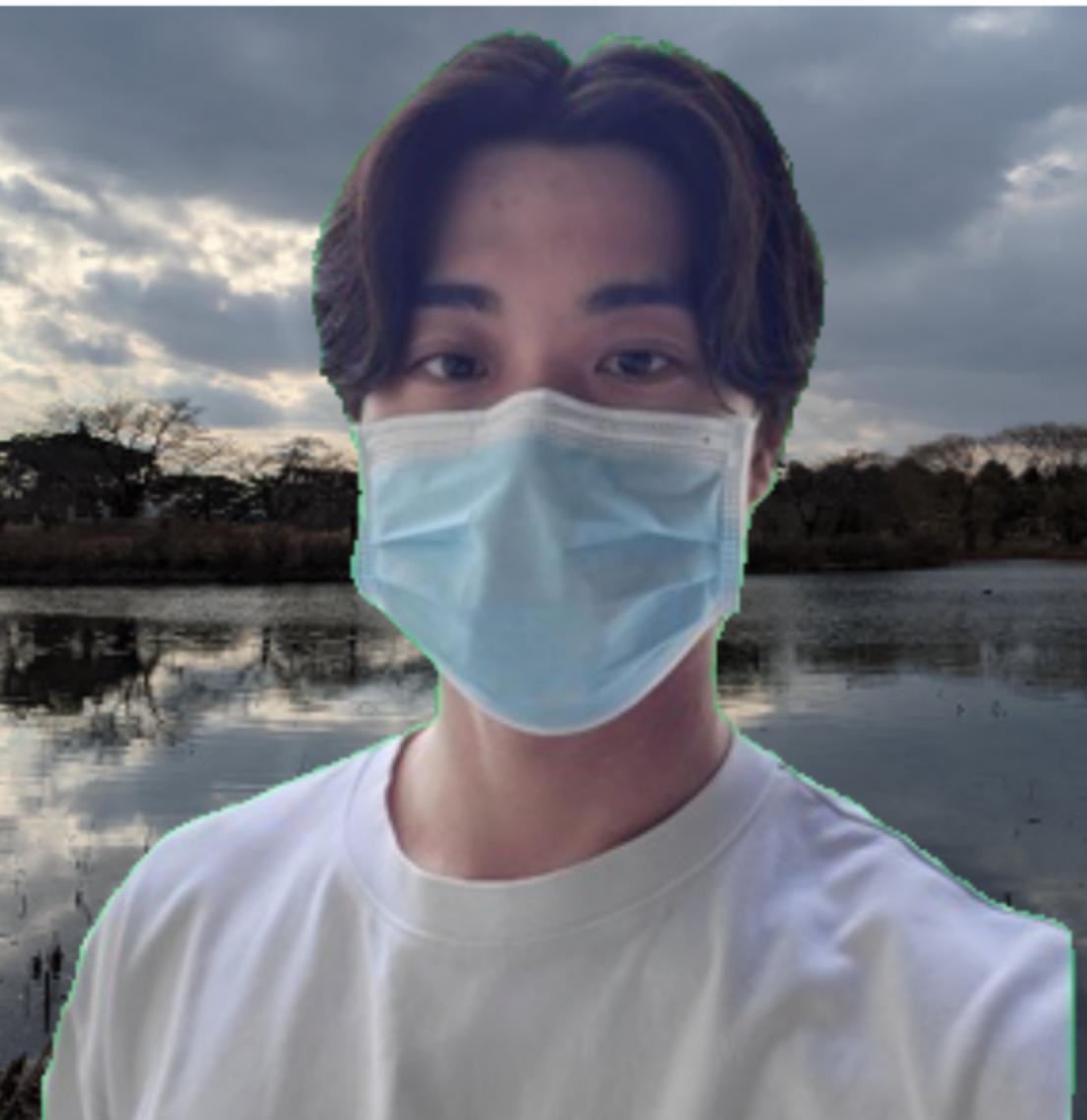
## 양선형 보간법 크기 조정

- 크기 호환성 오류를 해결하기 위함
- 크기 조정시 왜곡 최소화



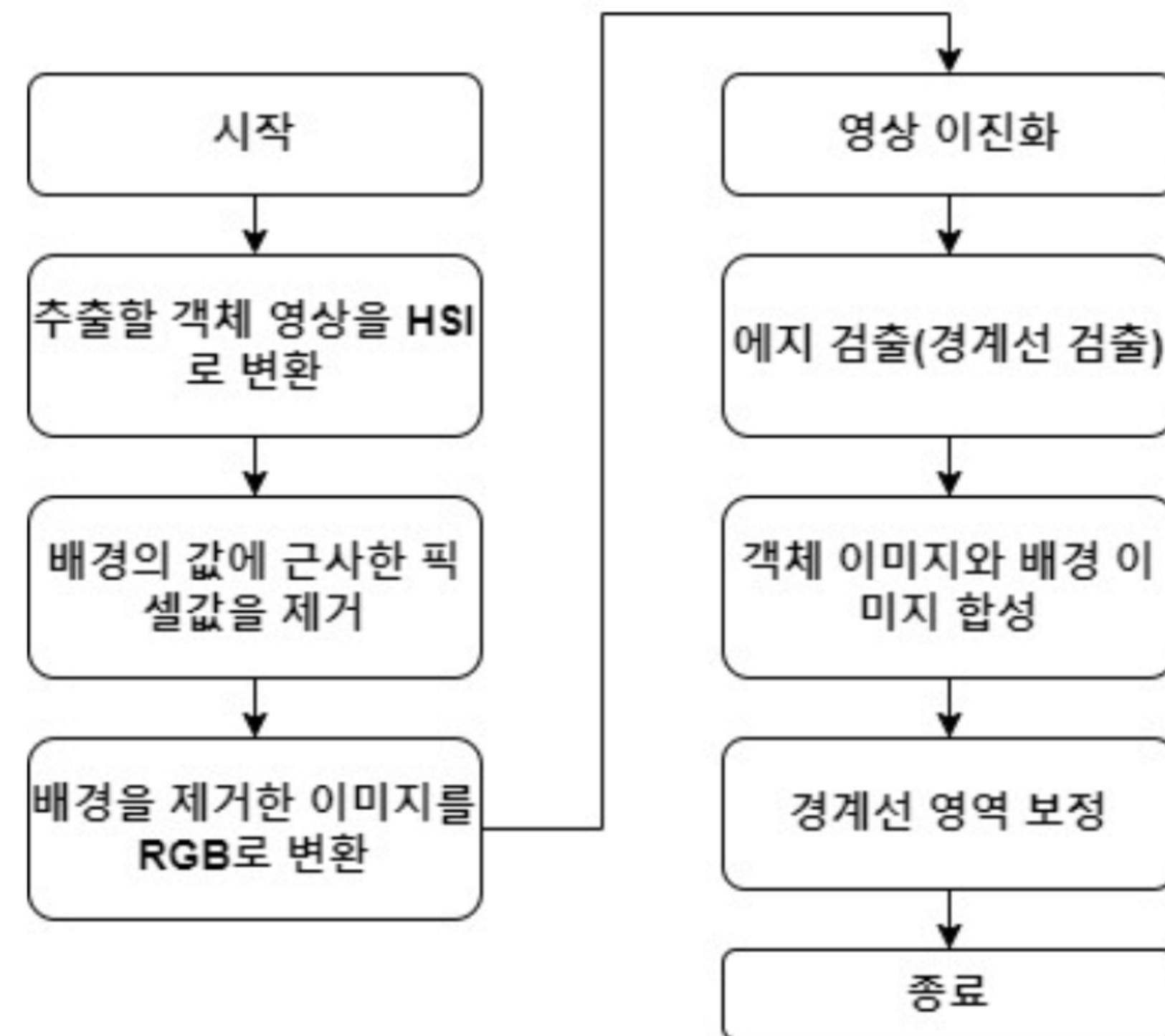
# 05 크로마키 보정 알고리즘

- 크로마키 알고리즘 적용 이후 객체 주변 노이즈 발생
- 노이즈를 제거하기 위한 보정 알고리즘 적용 필요



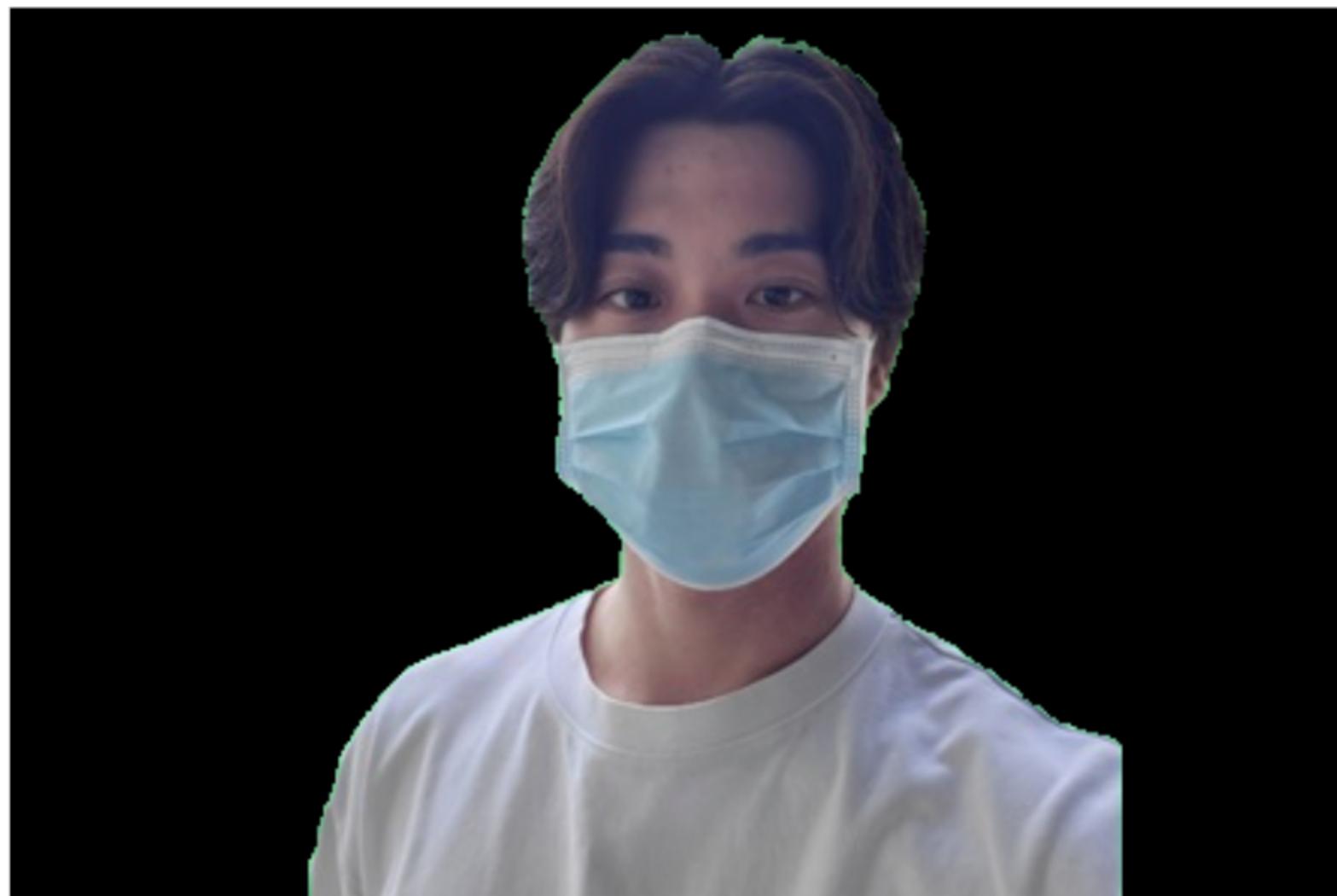
# 05 크로마키 보정 알고리즘

## 알고리즘 순서도



# 05 크로마키 보정 알고리즘 - 배경 픽셀 제거

- 배경과 객체의 이진화를 위함
- 배경에 해당되는 픽셀은 전부 검정으로 설정



```
if (H >= hue_green_min  
    && H <= hue_green_max  
    && S >= saturation_min) {  
    tmplImageR = 0;  
    tmplImageG = 0;  
    tmplImageB = 0;  
}  
else{  
    tmplImageR = inImageR;  
    tmplImageG = inImageG;  
    tmplImageB = inImageB;  
}
```

# 05 크로마키 보정 알고리즘 - 영상 이진화

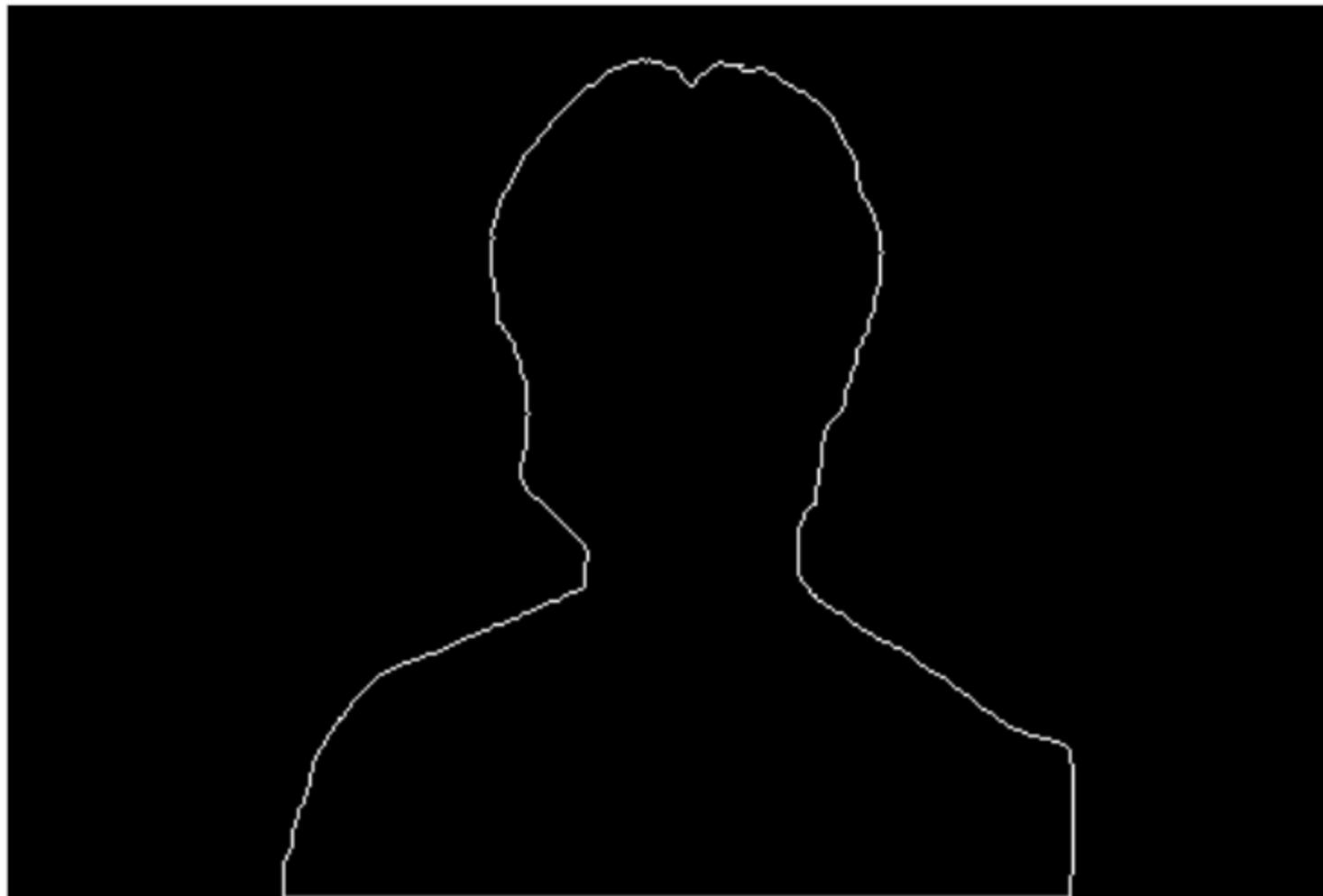
- 배경과 객체의 에지 검출을 위함
- 객체에 해당되는 영역은 전부 흰색으로 설정



```
# 평균값 계산  
double avg = sum / (m_inH * m_inW);  
  
# 이진화  
if (avgRGB > avg)  
    avgRGB = 255;  
else  
    avgRGB = 0;  
  
tmplImageR = tmplImageG = tmplImageB  
= (unsigned char)avgRGB;
```

# 05 크로마키 보정 알고리즘 - 에지 검출

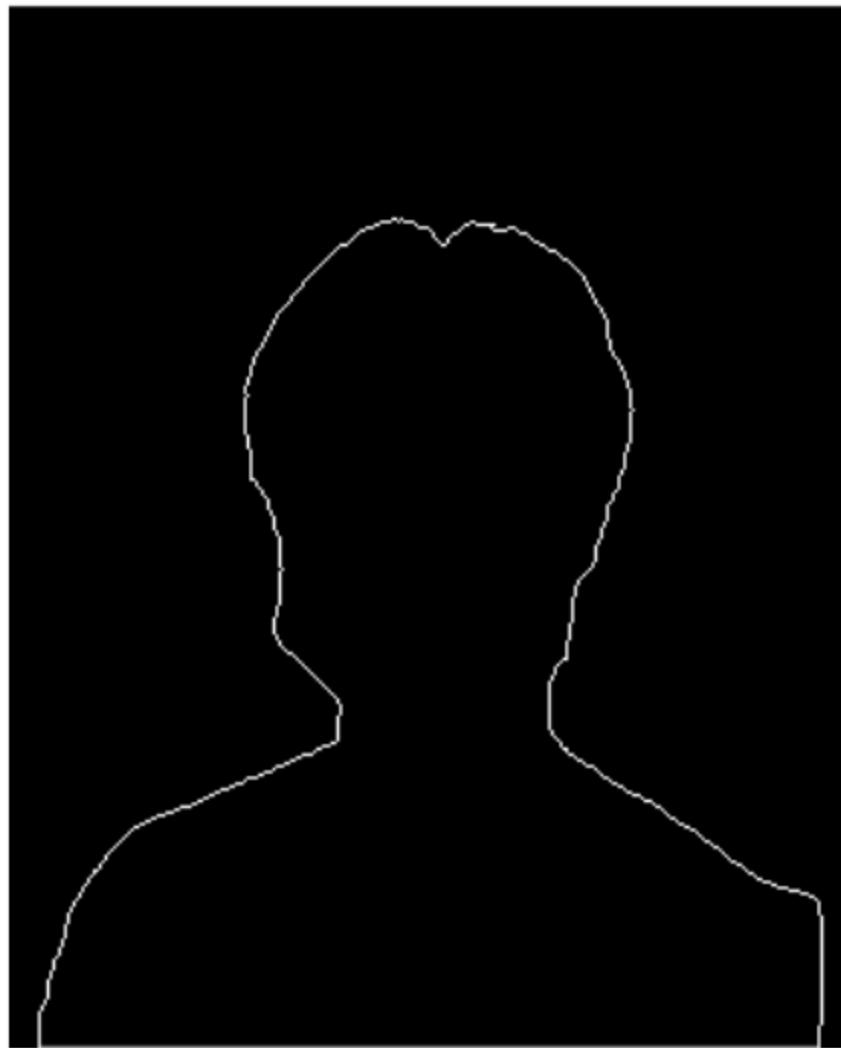
- 보정 처리의 기준이 되는 경계선 검출
- 에지 검출 기법은 라플라시안으로 설정



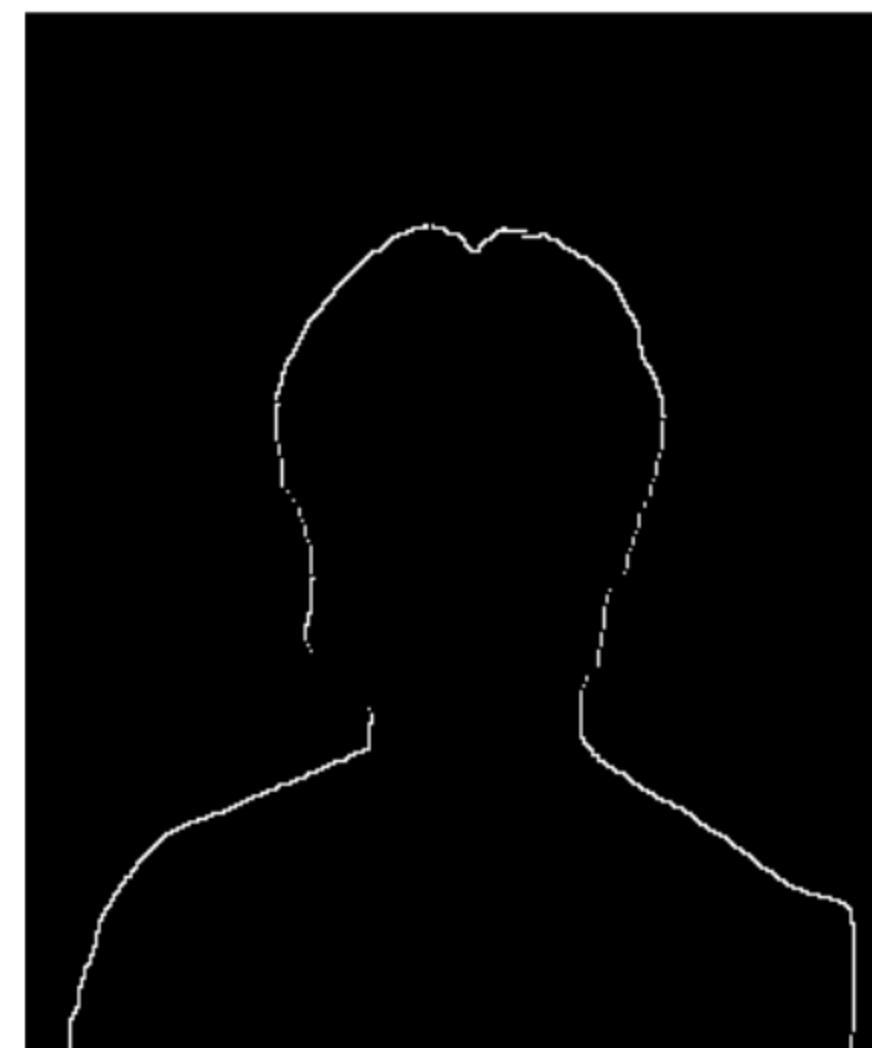
```
# 라플라시안 에지 검출  
double mask[3][3] = {  
    { 0.0,-1.0, 0.0},  
    {-1.0, 4.0,-1.0},  
    { 0.0,-1.0, 0.0} };
```

```
# 회선 처리  
S += edgeInImage[i+m][k+n] * mask[m][n]  
edgeOutImage[i][k] = S  
tmplImage[i][k] = int(edgeOutImage[i][k])
```

# 05 크로마키 보정 알고리즘 - 에지 검출



라플라시안



로버츠

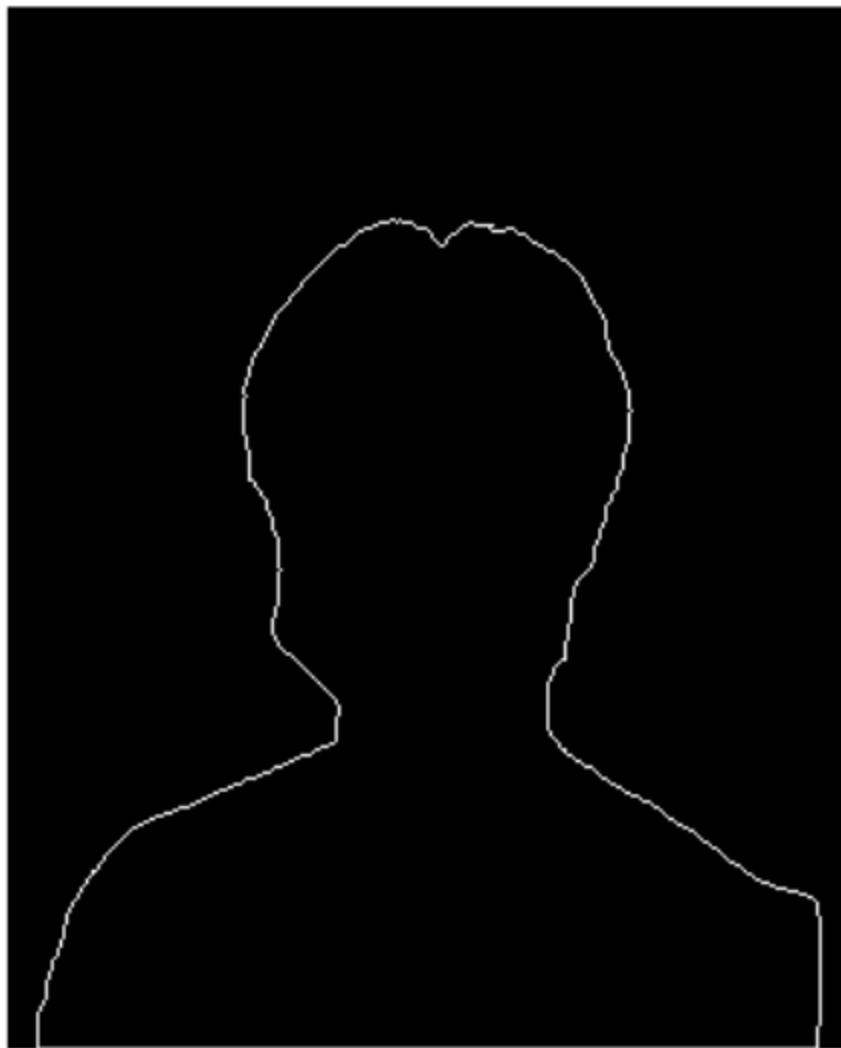


프리윗

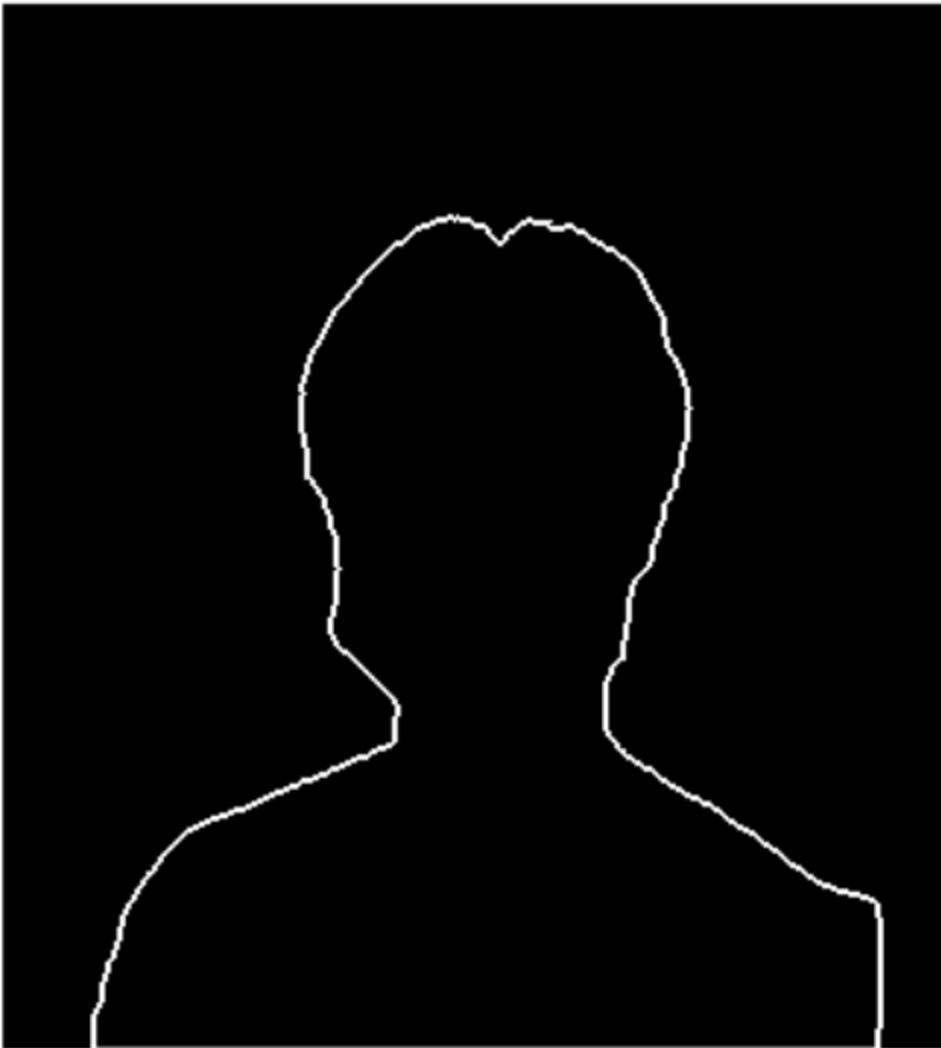


소벨

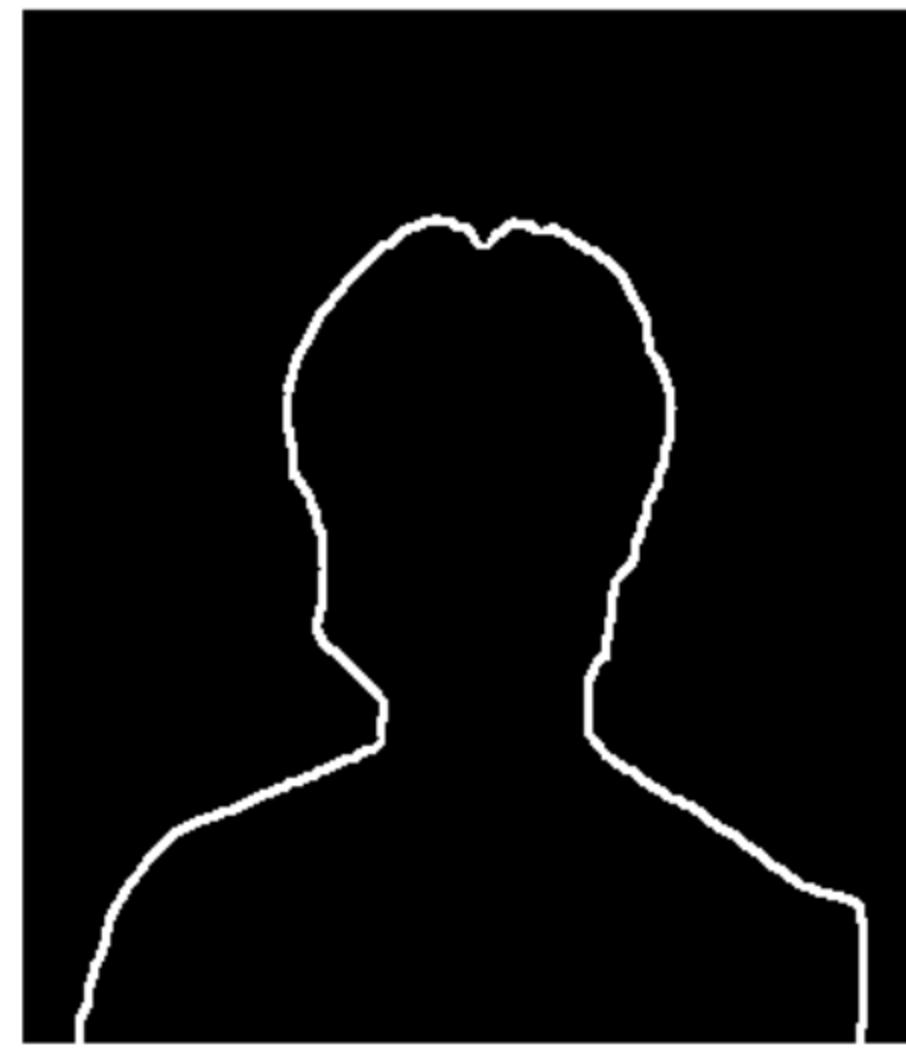
# 05 크로마키 보정 알고리즘 - 에지 검출



라플라시안



LoG

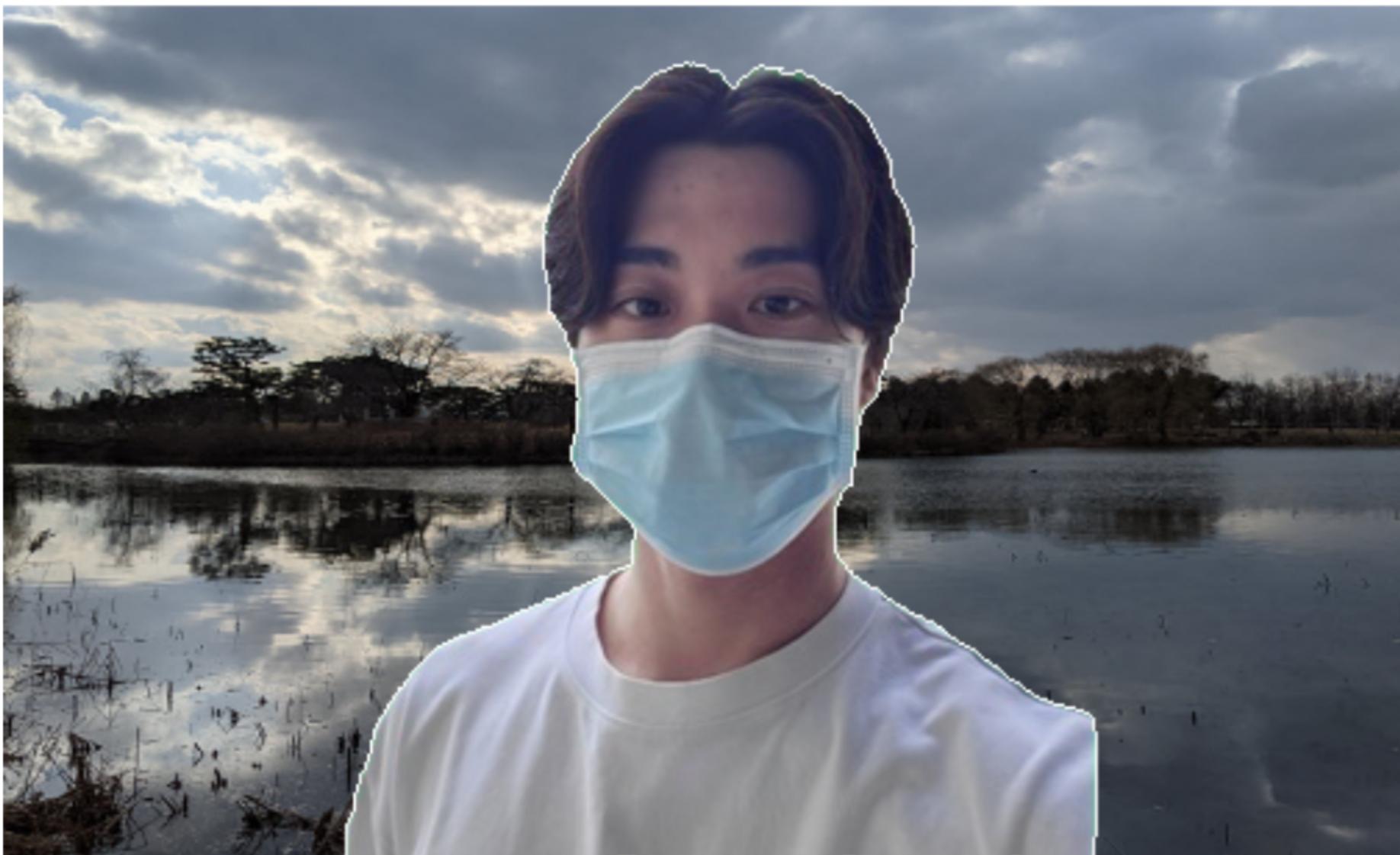


DoG

# 05 크로마키 보정 알고리즘 - 에지 검출

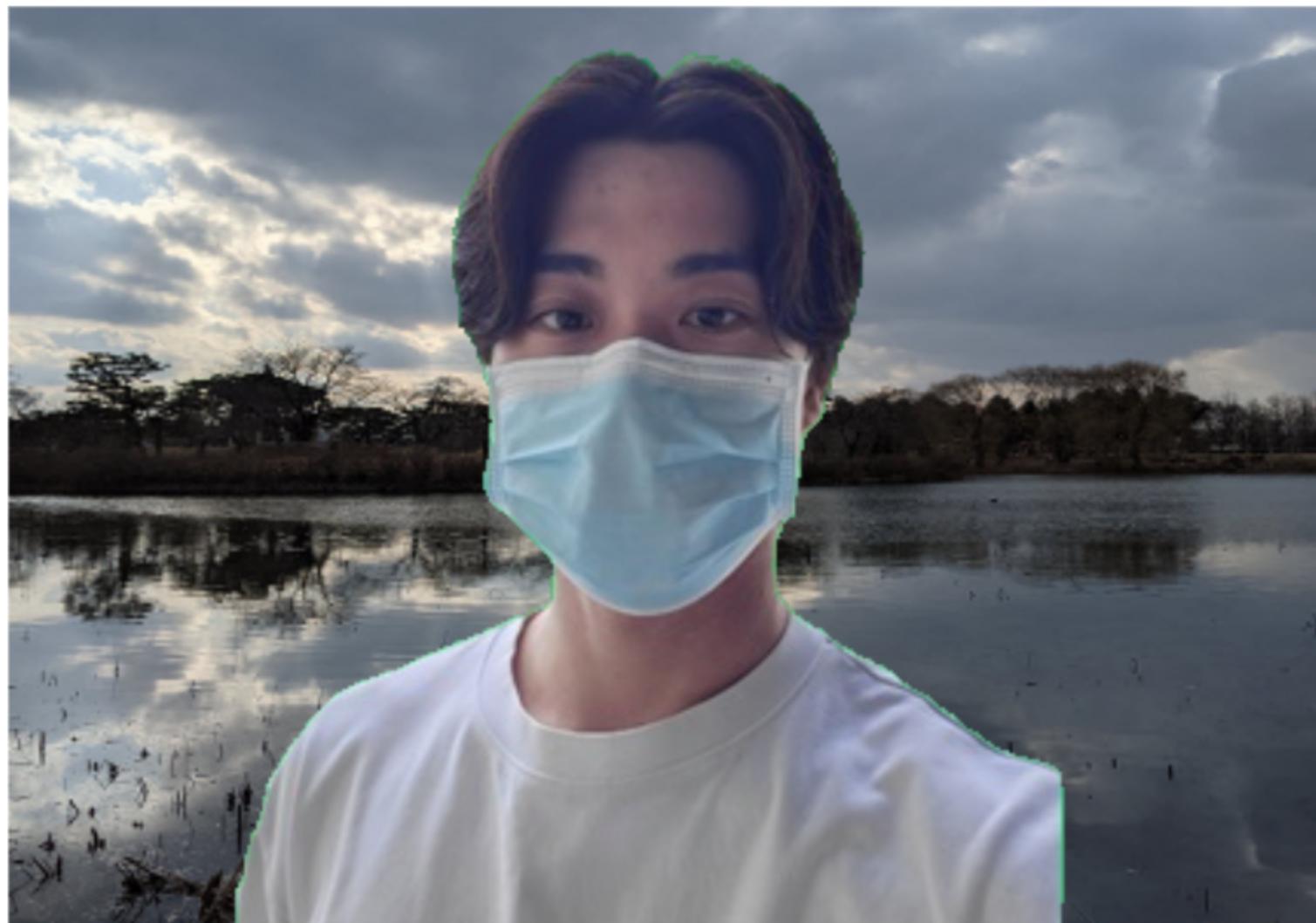
## 중간 결과

- 노이즈 영역과 경계선 영역이 중복된 것을 확인



# 05 크로마키 보정 알고리즘 - 이미지 합성

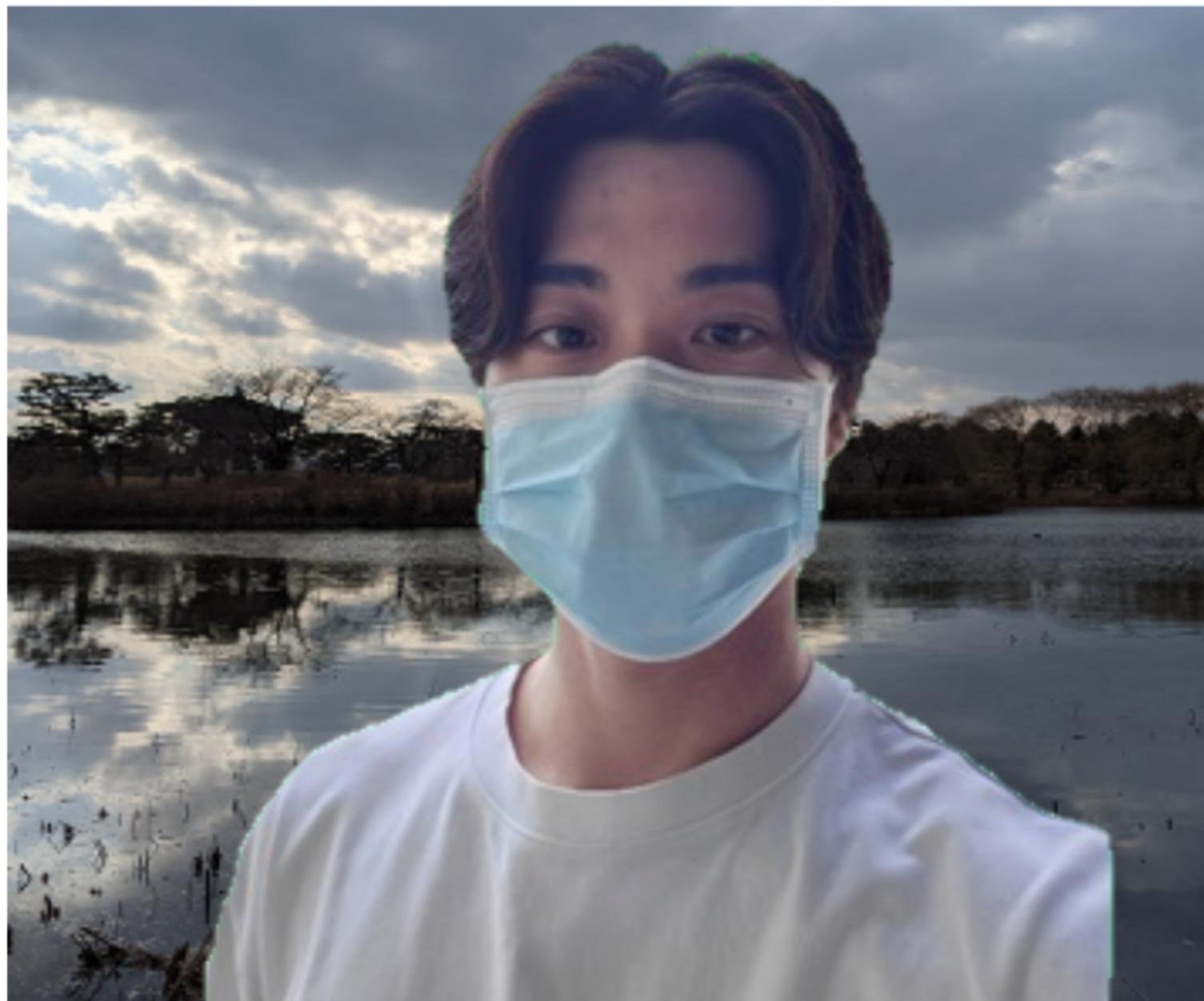
- 객체 이미지와 배경 이미지 합성
- 경계선에 해당되는 픽셀 변경



```
# 경계선 영역에 대하여  
if (tmpImageR[i][k] == 255  
    && tmpImageG[i][k] == 255  
    && tmpImageB[i][k] == 255) {  
    # 배경 이미지로 대입  
    tmpOutImageR[i][k] = newImageR[i][k];  
    tmpOutImageG[i][k] = newImageG[i][k];  
    tmpOutImageB[i][k] = newImageB[i][k];  
}
```

# 05 크로마키 보정 알고리즘 - 경계선 영역 보정

## 블러링 기법 적용



# 블러링 마스크

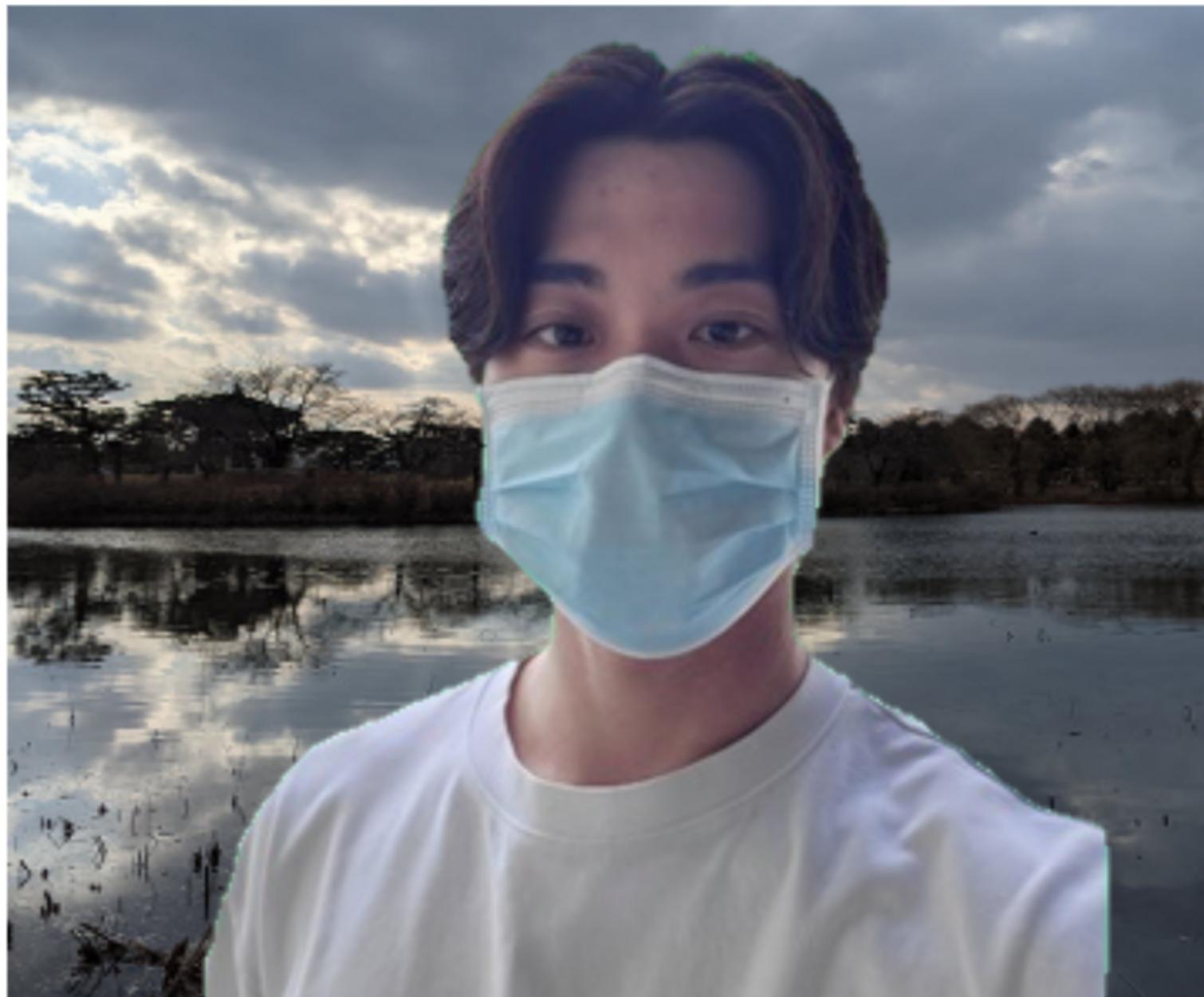
```
double maskB[3][3] = {  
    { 1. / 9, 1. / 9, 1. / 9},  
    { 1. / 9, 1. / 9, 1. / 9},  
    { 1. / 9, 1. / 9, 1. / 9} };
```

# 경계선 영역 회선 처리

```
if (tmplImageR[i][k] == 255  
&& tmplImageG[i][k] == 255  
&& tmplImageB[i][k] == 255){  
    ....  
}
```

# 05 크로마키 보정 알고리즘 - 경계선 영역 보정

## 중간값 필터링 기법 적용

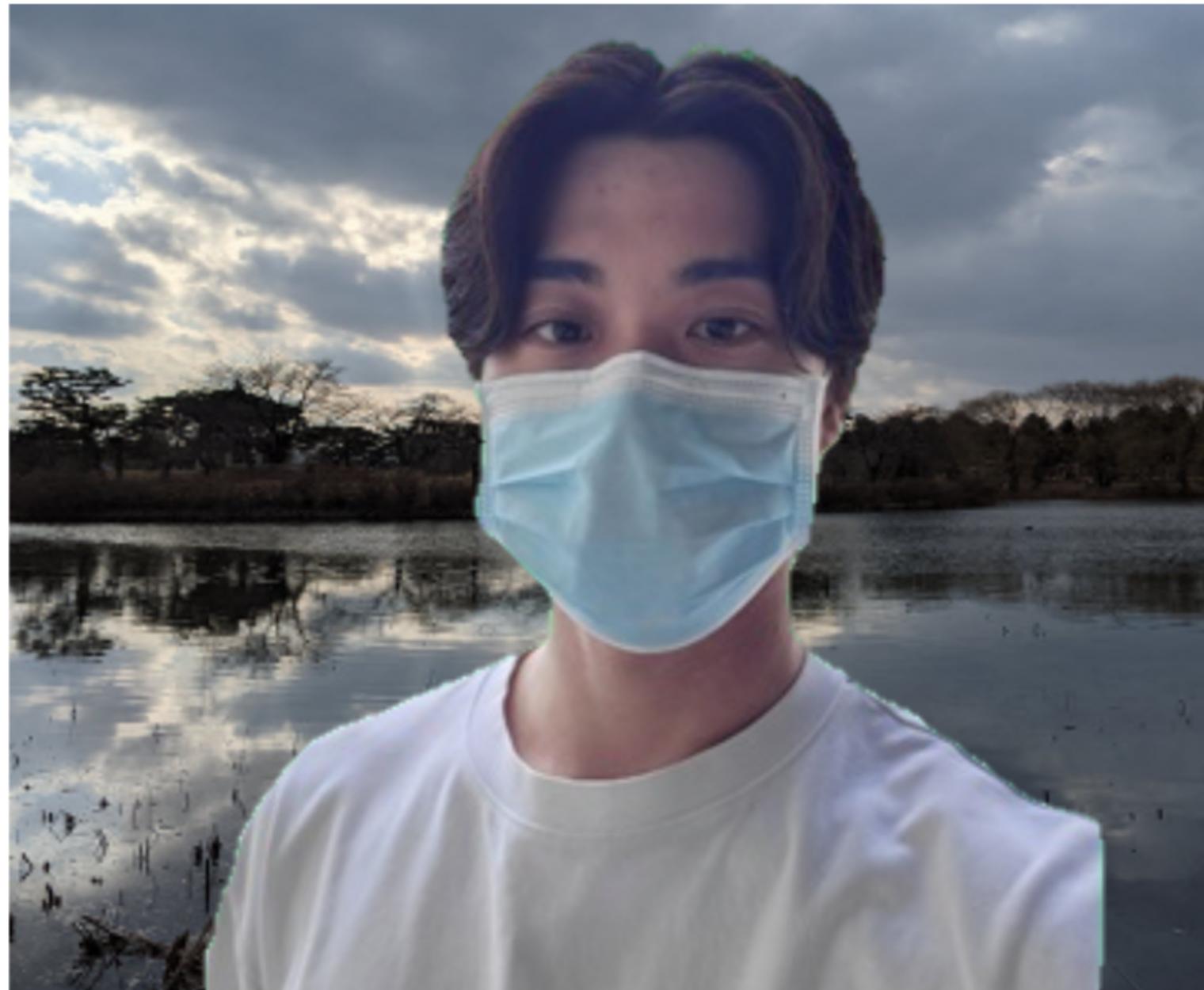


# 경계선 영역에 대하여

```
if (R == 255 && G == 255 && B == 255){  
    for (int x = -1; x <= 1; x++) {  
        for (int y = -1; y <= 1; y++) {  
            mR[index] = m_inImageR[i + x][k + y];  
            ...  
            index++;  
            ...  
            std::sort(mR, mR + 9);  
            tmpOutImageR[i][k] = mR[4];  
    }  
}
```

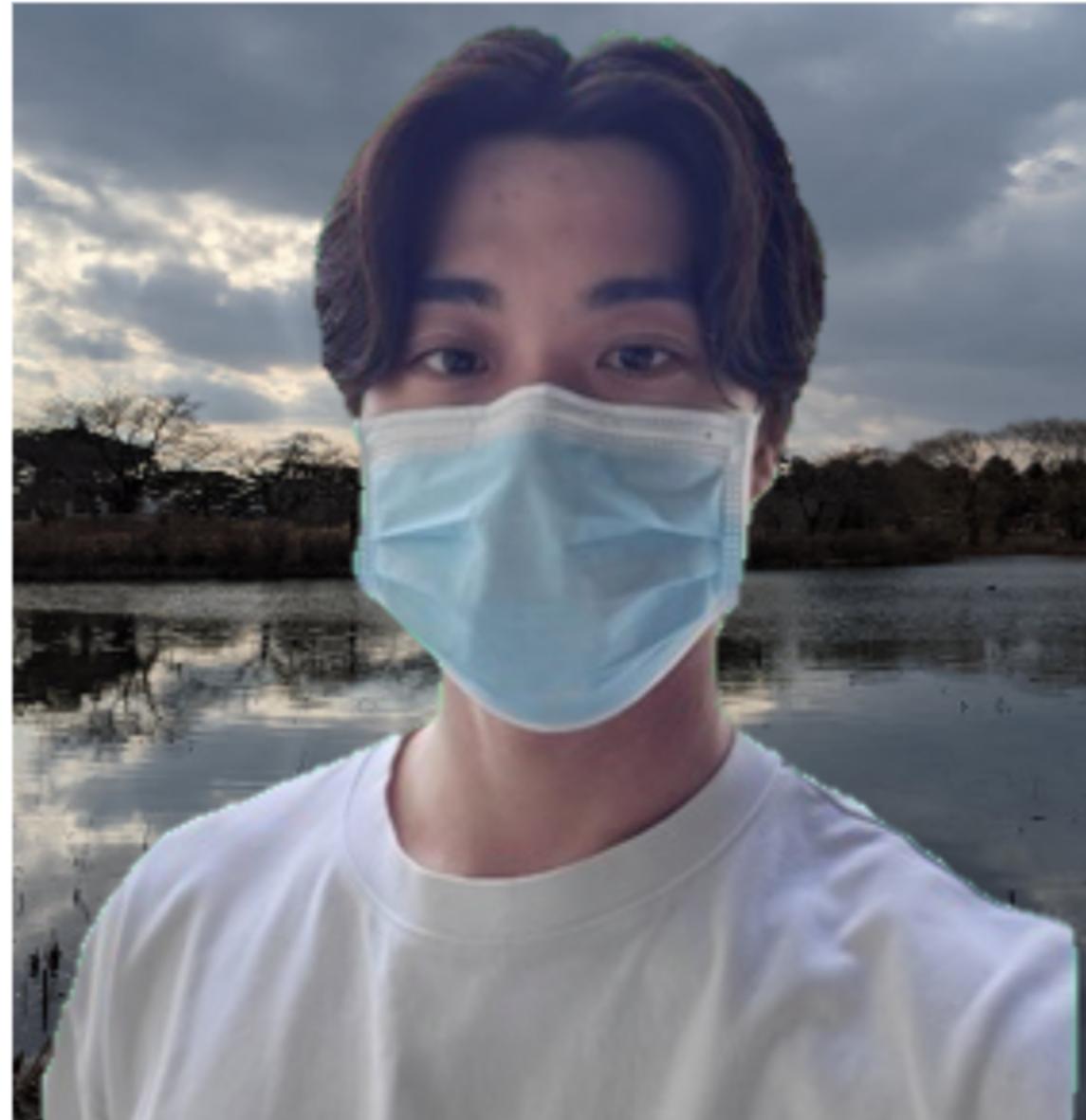
# 05 크로마키 보정 알고리즘 - 경계선 영역 보정

## 평균값 필터링 기법 적용

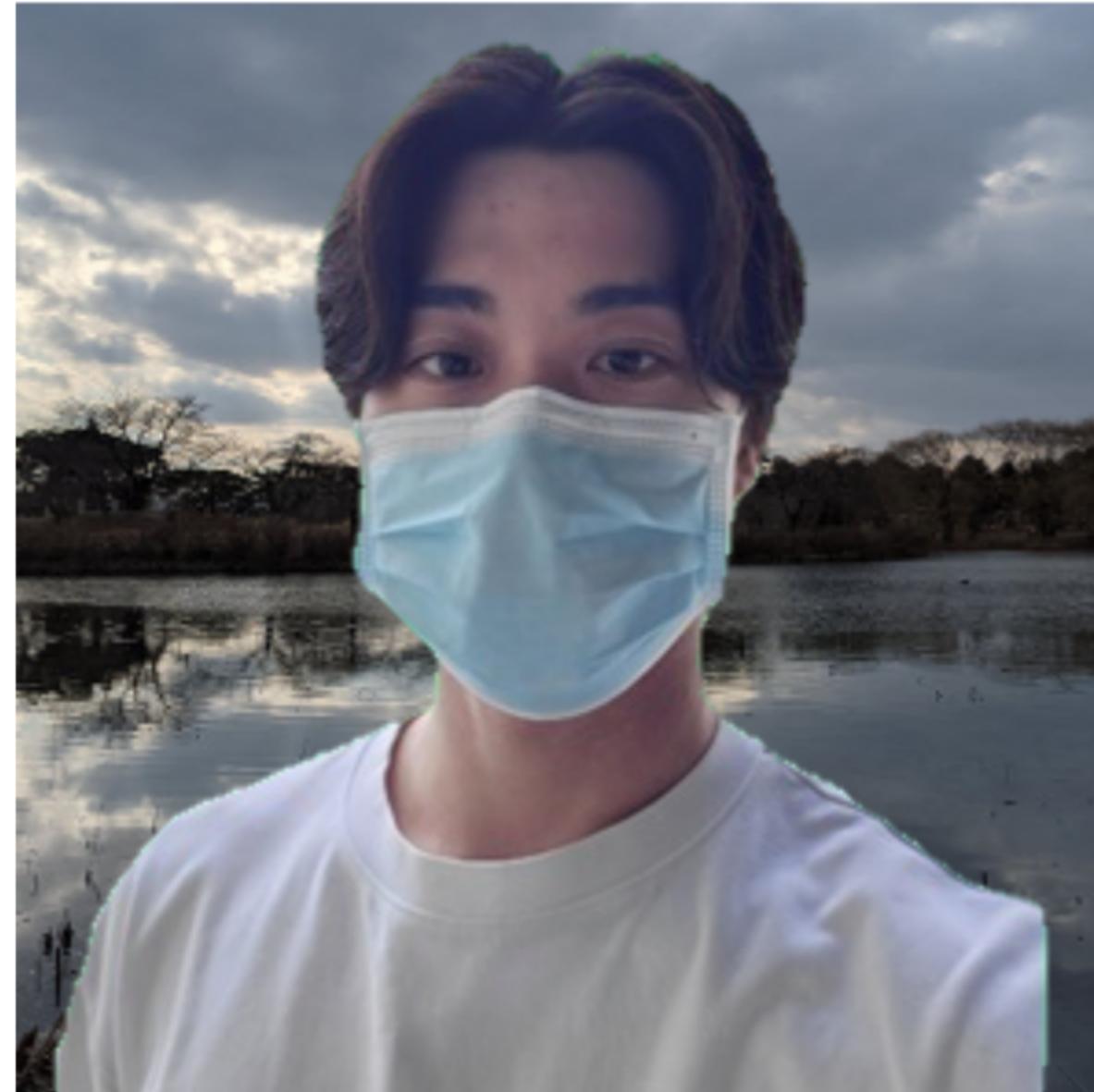


```
# 경계선 영역에 대하여
if (R == 255 && G == 255 && B == 255){
    for (int x = -1; x <= 1; x++) {
        for (int y = -1; y <= 1; y++) {
            sumR += tmpOutImageR[i + x][k + y];
        }
    }
    ...
    double avgR = sumR / 9.0;
    tmpOutImageR[i][k] = avgR;
}
```

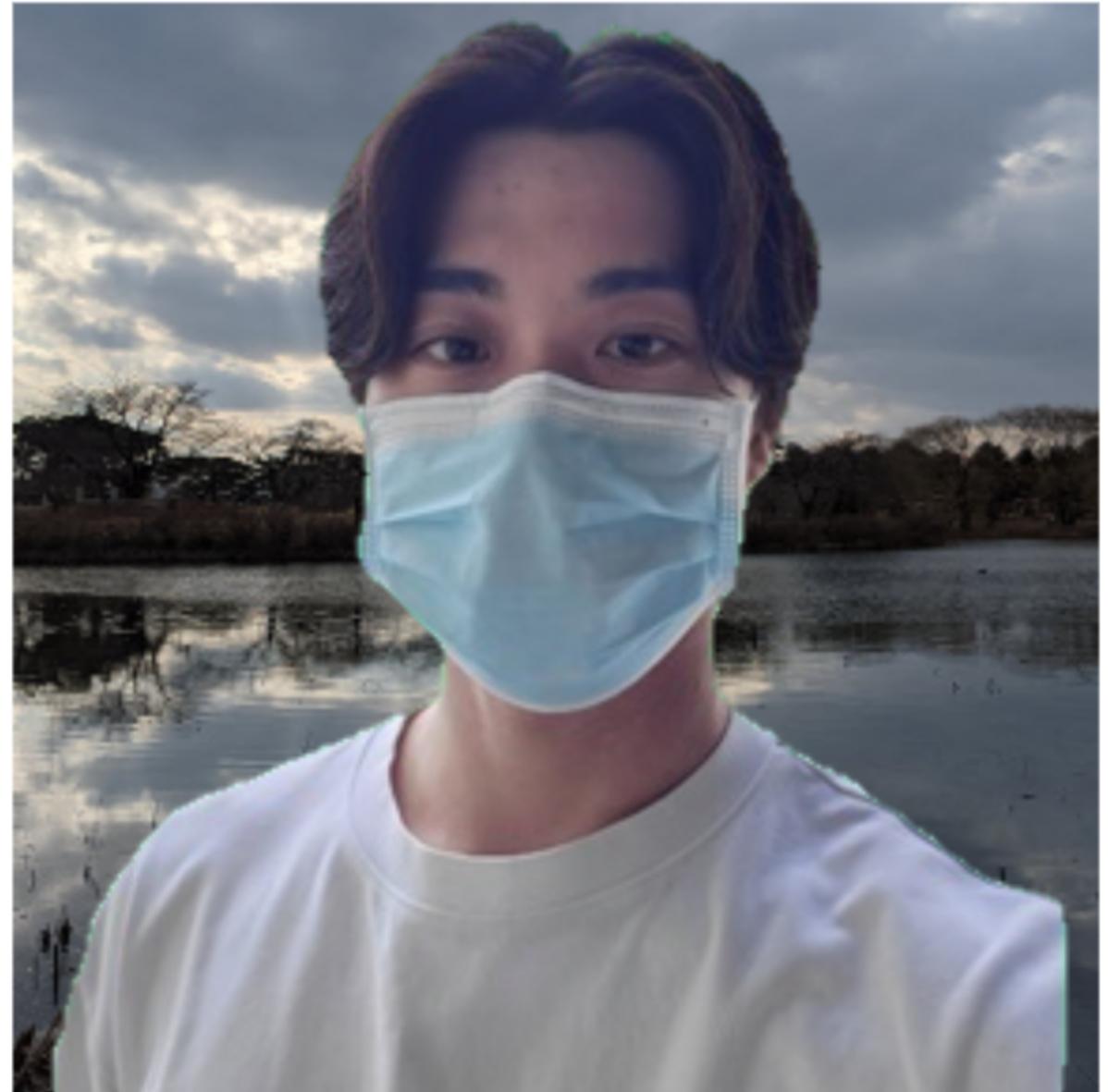
# 05 크로마키 보정 알고리즘 - 결과 비교



블러링

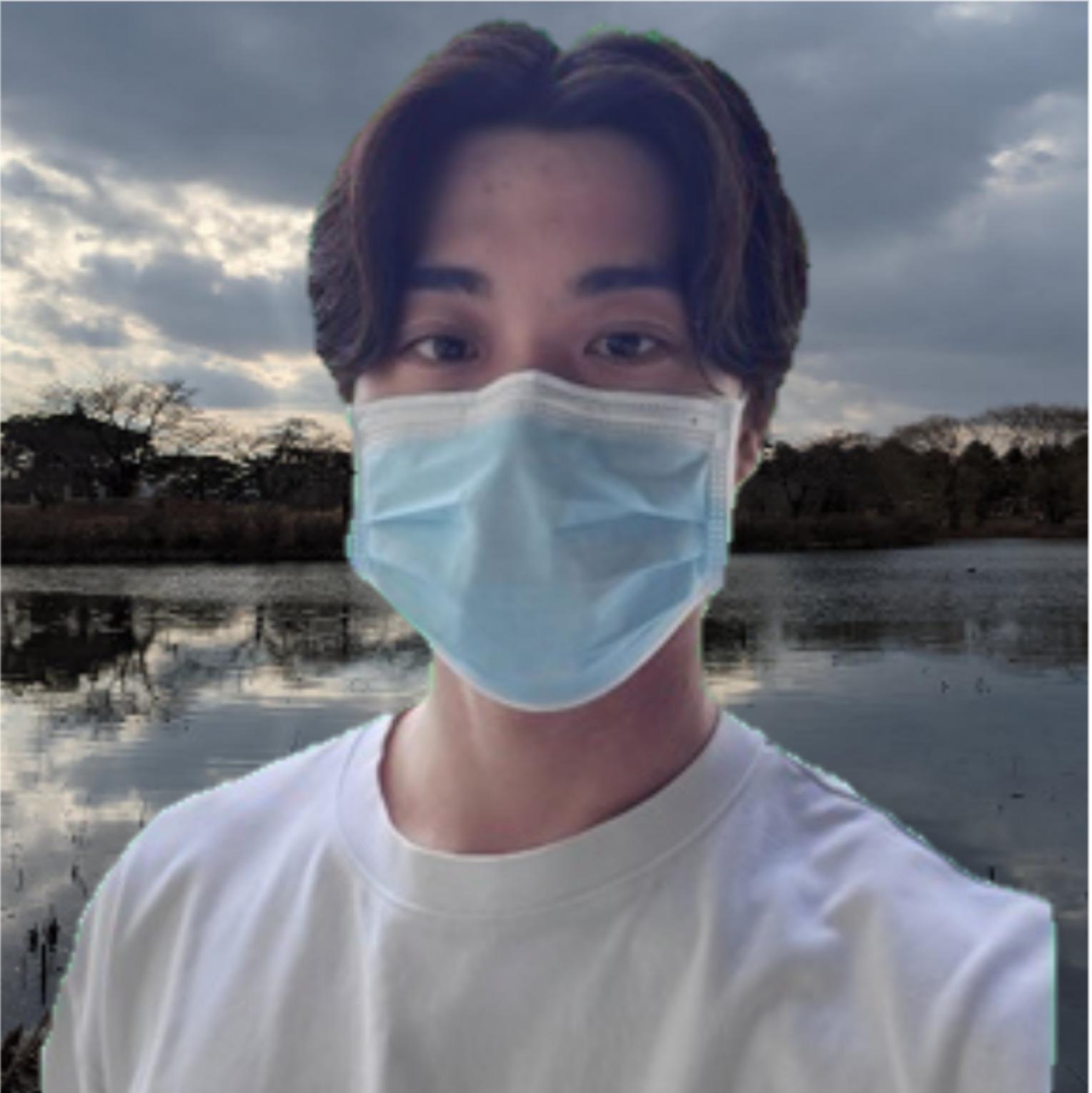
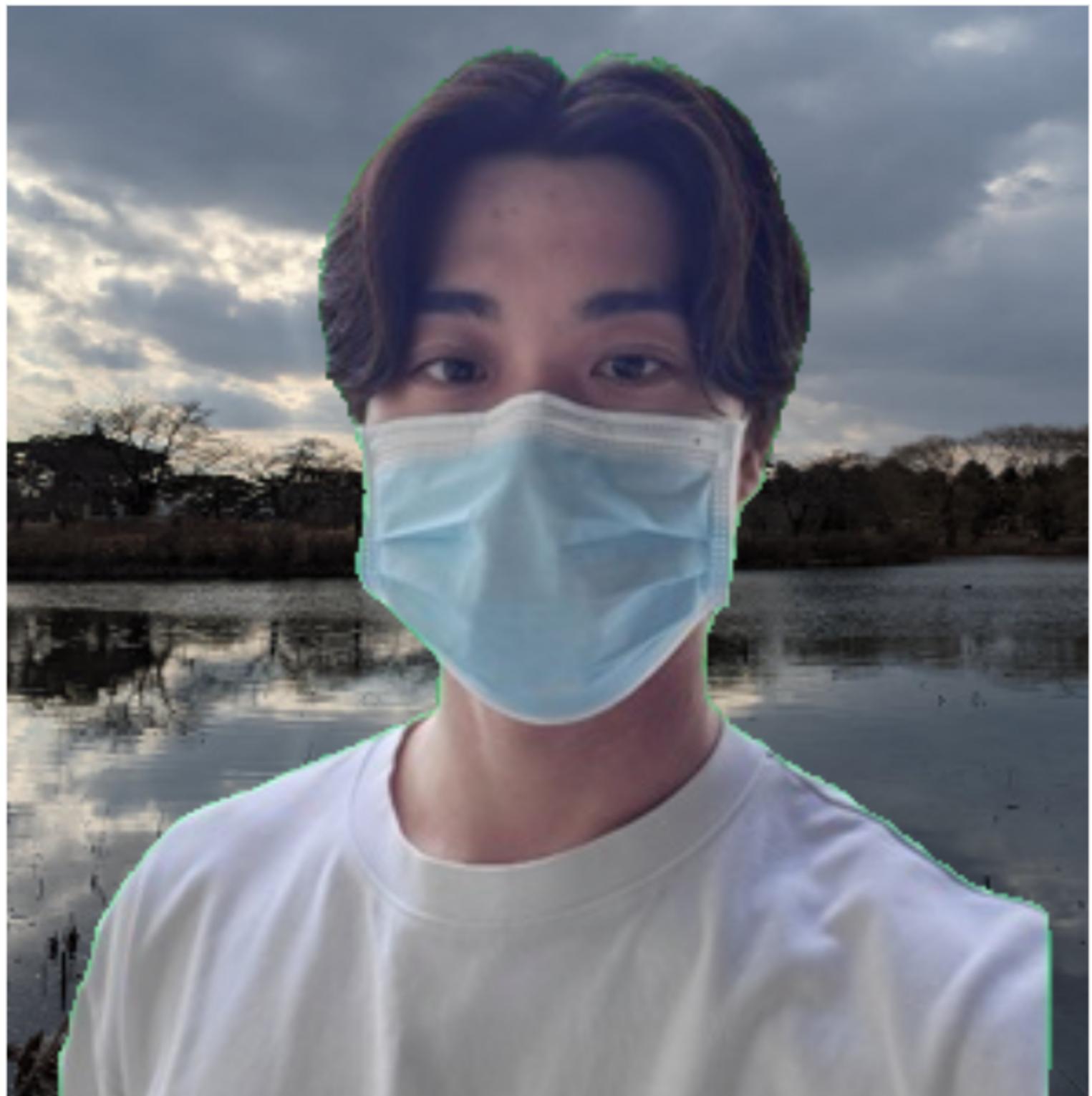


중간값



평균값

# 05 크로마키 보정 알고리즘 - 결과 비교



# 05 크로마키 보정 알고리즘 - 결과 비교



# 06 마치며

## 느낀점

- 노이즈를 보정하기 위한 기법을 고민하면서 영상처리에 대한 이해도를 높일 수 있었음
- UI를 설계하는 과정에서 MFC를 활용하는 경험을 할 수 있었음
- C++을 다루면서 객체지향 언어에 대한 이해를 높일 수 있었음

## 한계점

- 객체 탐지의 어려움 → 배경이 녹색 배경이 아니더라도 객체를 탐지할 수 있는가
- 노이즈 완전 제거의 어려움 → 에지 검출의 정확도 개선 필요
- 참고 자료의 부족 → 동일한 기능의 참고 자료는 OpenCV로 구현

# 06 마치며

## 향후 발전 방향

- 배경화면이 녹색이 아니어도 크로마키 기능 구현
- 효과 중첩 적용 기능
- Undo, Redo 기능
- 크로마키 보정 알고리즘 개선
- 코드 최적화

# 06 마치며

## 참고 문헌

- 박정주, 낭종호. (2015). "인페인팅 기법을 활용한 크로마키 구현". 한국정보과학회 2015년 동계학술발표회, pp.1,180 - 1,182.
- 유길상. (2021). "배경 모델링을 이용한 비디오 크로마키 생성기법" 한국융합학회논문지, 12(10), pp.1 - 8.

# 06 마치며

감사합니다

QnA