미니프로젝트 버전 2.0

OpenCV 없이 구현한
컬러 영상처리 프로젝트 using C++

과정명: [intel] 엣지 AI SW 아카데미(4기)

과목명: 객체지향 프로그래밍

발표자: 정용재

프로젝트 요약

- **프로젝트 목표**
  - ✓ **영상처리 알고리즘의 원리 이해**
  - ✓ 영상처리 기능구현(with C++)
  - ✓ MFC로 GUI 기능구현
- **개발 환경**
  - ✓ C++
  - ✓ Visual Studio Community 2022
- **지원 환경**
  - ✓ Windows

기능 요약

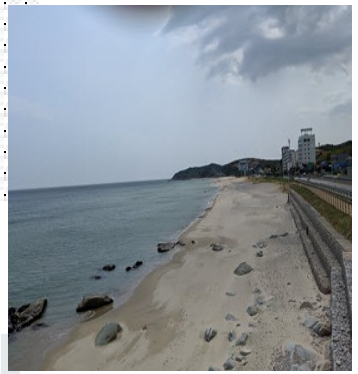
- **화소점 처리**
  - ✓ 산술 연산 (+, -, *, /)
  - ✓ 논리 연산 (AND, OR, XOR)
  - ✓ 감마 보정
  - ✓ 이진화, 반전
  - ✓ 파라볼라 CAP, CUP
- **히스토그램 화소점 처리**
  - ✓ 히스토그램 스트레치
  - ✓ 히스토그램 엔드-인
  - ✓ 히스토그램 평활화
- **화소영역 처리**
  - ✓ 블러링
  - ✓ 엠보싱
  - ✓ 고주파 샤프닝
  - ✓ 가우시안 스무딩
- **기하학 처리**
  - ✓ 확대, 축소
  - ✓ 배경을 고려한 회전
- **컬러 효과(HSI 모델)**
  - ✓ 채도 편집
  - ✓ 색상 추출
- **그레이스케일 에지 검출**
  - ✓ 1차원 프리윗
  - ✓ 2차원 LoG

화소점 처리 - 산술연산

더하기, 빼기

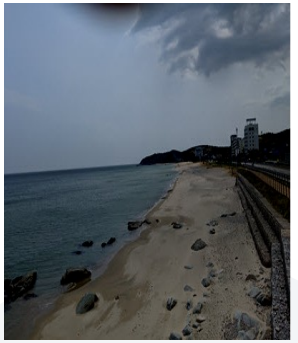
화면

원본



+50

-50



핵심 코드

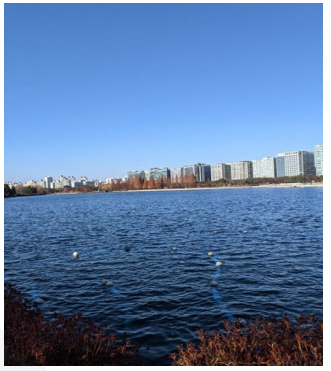
```
// 라디오 버튼 -> 연산이 바뀜   // 입력상자의 입력값 받아오기    // 초기값은 '+'
int index = arith.m_radioIndex; double value = arith.m_arithVal; char select = '+';
switch (index) {
case 0:
    select = '+'; break;
case 1:
    select = '-'; break;
case 2:
    select = '*'; break;
case 3:
    select = '/'; break;
}
// 연산자에 따라 그에 맞는 연산 실행
OnArithCal(value, select);
```

세부 사항

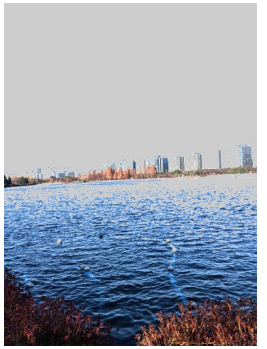
1) 이미지의 히스토그램의 분포가 그대로 좌,우로 이동

곱하기, 나누기

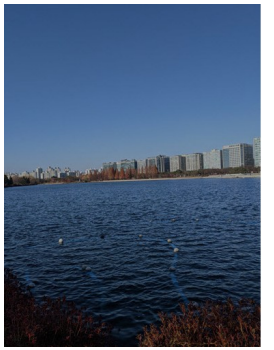
원본



*1.5



/1.5



```
case '/':
    if (value == 0.0) {
        value = 1.0;
    }
    for (int h = 0; h < m_inH; h++) {
        for (int w = 0; w < m_inW; w++) {
            double R, G, B;
            R = double(m_inImageR[h][w]);
            G = double(m_inImageG[h][w]);
            B = double(m_inImageB[h][w]);

            R /= value;
            G /= value;
            B /= value;
```

1) 곱하기 – 어두운 화소는 조금 밝아지고, 밝은 화소는 많이 밝아짐

2) 나누기 – 밝은 화소는 많이 어두워지고, 어두운 화소는 조금 어두워짐

화소점 처리 - 논리연산

AND

화면

원본

& 128

핵심 코드

```
switch (select) {
case '&':
    for (int h = 0; h < m_inH; h++) {
        for (int w = 0; w < m_inW; w++) {
            int R, G, B;
            R = int(m_inImageR[h][w]);
            G = int(m_inImageG[h][w]);
            B = int(m_inImageB[h][w]);
            R &= value;
            G &= value;
            B &= value;
```

세부 사항

특정 비트를 의도적으로 0으로 만들 수 있다.

예) 128을 입력 -> 128이상의 화소만 128로 표현된다

OR

원본

| 127

```
case '|':
    for (int h = 0; h < m_inH; h++) {
        for (int w = 0; w < m_inW; w++) {
            int R, G, B;
            R = int(m_inImageR[h][w]);
            G = int(m_inImageG[h][w]);
            B = int(m_inImageB[h][w]);

            R |= value;
            G |= value;
            B |= value;
```

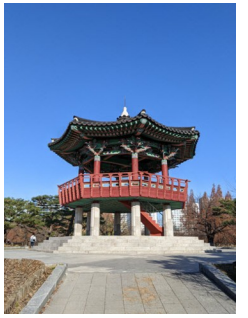
특정 비트를 의도적으로 1으로 만들 수 있다.

예) 127을 입력 -> 128이상의 화소가 255로 표현된다

또한 127이하의 화소는 전부 127이 된다.

XOR

원본



^ 127

```
case '^':
    for (int h = 0; h < m_inH; h++) {
        for (int w = 0; w < m_inW; w++) {
            int R, G, B;
            R = m_inImageR[h][w];
            G = m_inImageG[h][w];
            B = m_inImageB[h][w];

            R ^= value;
            G ^= value;
            B ^= value;
```

특정 비트를 의도적으로 반전 시킬 수 있다.

예) 128을 입력 -> 128이상의 화소는 128을 뺄셈

127이하의 화소는 128을 더하는 효과

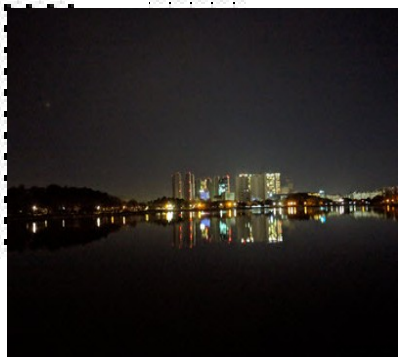
화소점 처리 - 감마보정

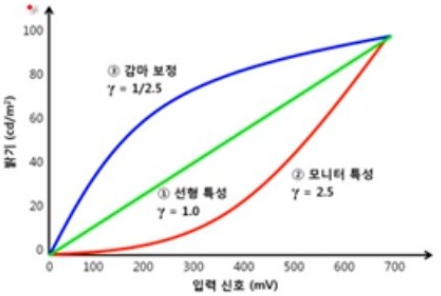
원본

γ = 1.5

γ = 0.5

화면





← 감마커브

핵심코드

```
double R, G, B;
// 연산을 위해 double 형 변환
R = double(m_inImageR[h][w]);
G = double(m_inImageG[h][w]);
B = double(m_inImageB[h][w]);

R = 255 * pow(R / 255, 1.0 / gammaVal);
G = 255 * pow(G / 255, 1.0 / gammaVal);
B = 255 * pow(B / 255, 1.0 / gammaVal);
```

$$y = 255 * (\frac{x}{255})^{1/\gamma}$$

감마 보정의 수식

세부사항

아주 어두운 부분과 아주 밝은 부분을 제외하고, 중앙값 부근의 밝기를 조절하여

1) 대비가 좋아지는 결과 2) 굴절된 모니터에서 왜곡을 감소시키는 역할을 할 수 있다.

화소점 처리 - 이진화, 반전

이진화

화면

원본

흑백

핵심 코드

```
// 127을 기준으로 극단값으로 보내기
for (int h = 0; h < m_inH; h++) {
    for (int w = 0; w < m_inW; w++) {
        if (m_inImageR[h][w] > 127) m_outImageR[h][w] = 255;
        else m_outImageR[h][w] = 0;

        if (m_inImageG[h][w] > 127) m_outImageG[h][w] = 255;
        else m_outImageG[h][w] = 0;

        if (m_inImageB[h][w] > 127) m_outImageB[h][w] = 255;
        else m_outImageB[h][w] = 0;
    }
}
```

세부 사항

입력값을 기준으로 크면 최댓값, 작으면 최솟값을 저장

반전

원본

반전

```
// 각 화소점에 NOT연산으로 반전하기
for (int h = 0; h < m_inH; h++) {
    for (int w = 0; w < m_inW; w++) {
        m_outImageR[h][w] = ~m_inImageR[h][w];
        m_outImageG[h][w] = ~m_inImageG[h][w];
        m_outImageB[h][w] = ~m_inImageB[h][w];
    }
}
```

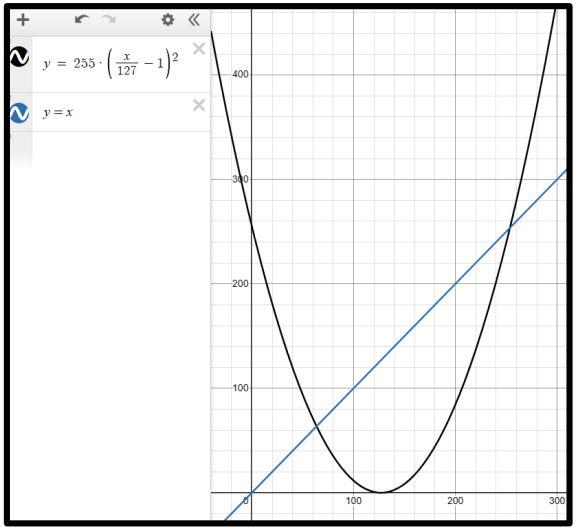
각 화소점에 NOT 논리연산

화소점 처리 - 파라볼라 CAP

화면

원본

파라볼라CAP



← 파라볼라 CAP 그래프

핵심 코드

```
// y = 255 * pow((화소/127 - 1),2)
R = 255 * pow((R / 127 - 1), 2);
G = 255 * pow((G / 127 - 1), 2);
B = 255 * pow((B / 127 - 1), 2);
```

$$y = 255 * (\frac{x}{127} - 1)^2$$

세부 사항

1) 어두운 화소들이 밝은 값으로 반전 -> 급격히 어두워짐

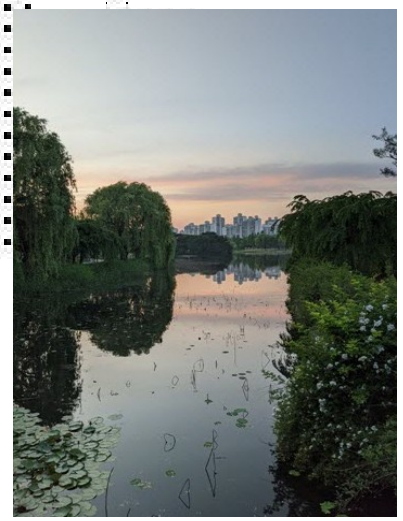
2) 중앙값 부근의 화소들이 실제보다 어두워짐 -> 급격히 밝아짐

3) 밝은 화소들이 실제보다 어두워짐 -> 급격히 밝아짐

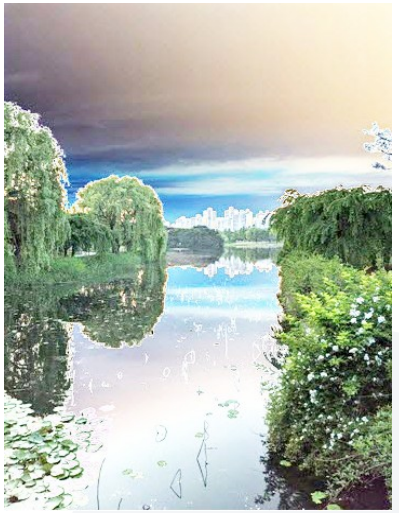
화소점 처리 - 파라볼라 CUP

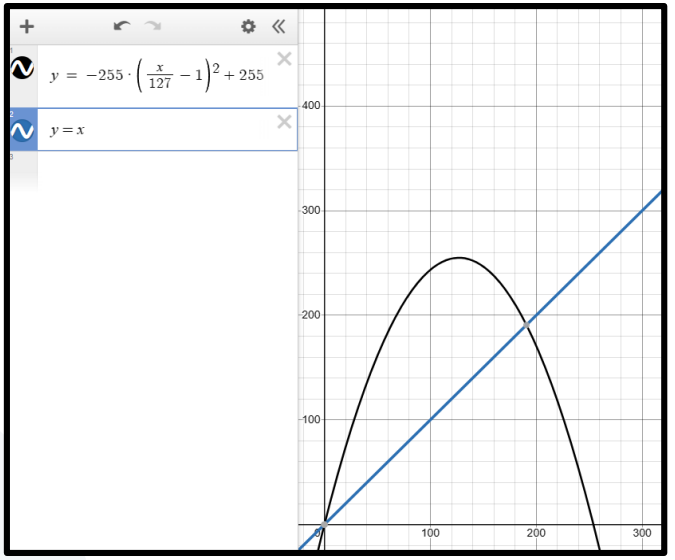
화면

원본



파라볼라CUP





← 파라볼라 CUP 그래프

핵심 코드

```
// 파라볼라 CUP 적용
R = -255 * pow((R / 127 - 1), 2) + 255;
G = -255 * pow((G / 127 - 1), 2) + 255;
B = -255 * pow((B / 127 - 1), 2) + 255;
```

$$y = -255 * (\frac{x}{127} - 1)^2 + 255$$

세부 사항

1) 어두운 화소들이 급격하게 밝은 값으로 바뀌는 효과

2) 중앙값 부근의 화소들이 실제보다 밝아지는 효과

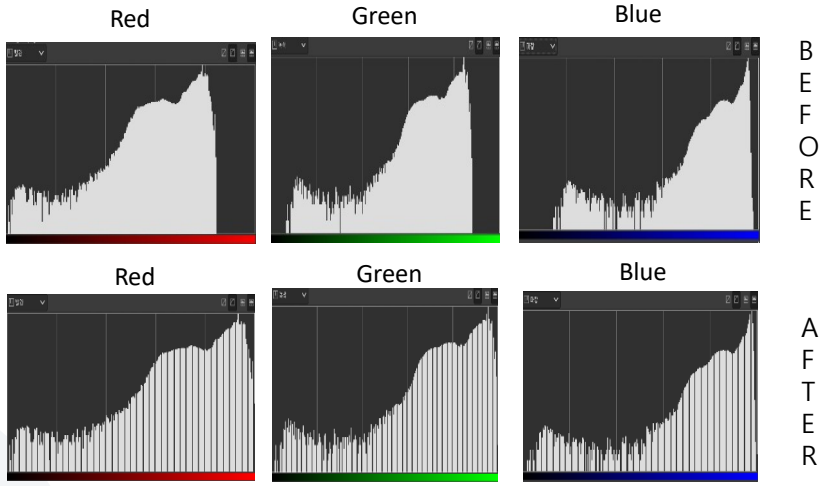
3) 밝은 화소들이 급격하게 어두운 값으로 바뀌는 효과

화소점 처리 - 히스토그램 스트레치

화면

원본

히스토그램 스트레치



핵심 코드

```
// 히스토그램 스트레치
for (int h = 0; h < m_outH; h++) {
    for (int w = 0; w < m_outW; w++) {
        int oldVal = m_inImageR[h][w];
        int newVal = int(((oldVal - lowestR) / (highestR - lowestR)) * 255.0);
        m_outImageR[h][w] = newVal;
    }
}
```

$$new\ pixel = \frac{old\ \ pixel - low}{high - low} \times 255$$

세부 사항

1) 편향된 화소의 분포를 정규화하는 효과 → RGB의 대비가 증가

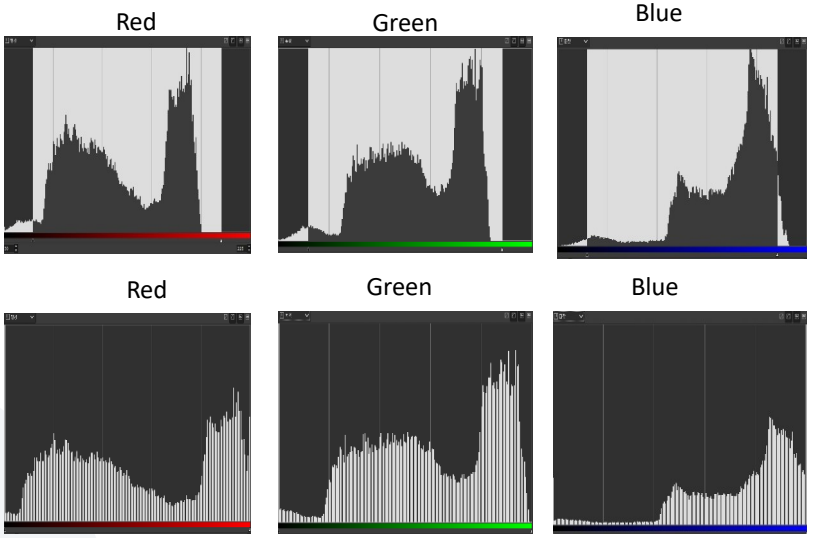
2) 문제점: 분포가 완만한 이미지는 더 이상 스트레치 불가

화소점 처리 - 히스토그램 엔드-인

화면

원본

히스토그램 엔드-인 (30)



핵심 코드

```
// 상한값과 하한값을 입력값으로 조정
highestG -= endinVal;
lowestG += endinVal;

// 조정된 값을 바탕으로 히스토그램 스트레치
for (int h = 0; h < m_outH; h++) {
    for (int w = 0; w < m_outW; w++) {
        int oldVal = m_inImageG[h][w];
        int newVal = int(((oldVal - lowestG) / (highestG - lowestG)) * 255.0);
```

$$new\ pixel = \begin{cases} 0 & old\ pixel \le low \\ \dfrac{old\ pixel - low}{high - low} \times 255 & low \le old\ pixel \le high \\ 255 & high \le old\ pixel \end{cases}$$

세부 사항

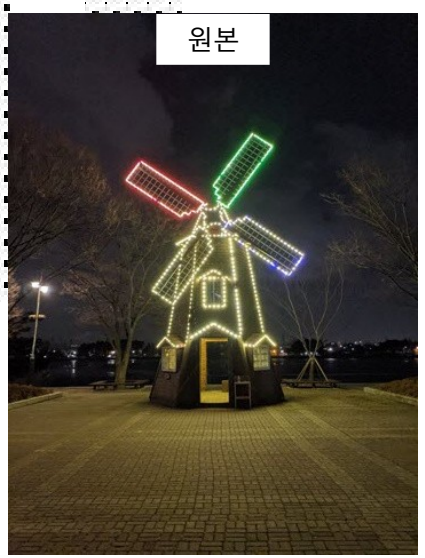
1) 편향된 화소의 분포를 정규화하는 효과 → RGB의 대비가 증가

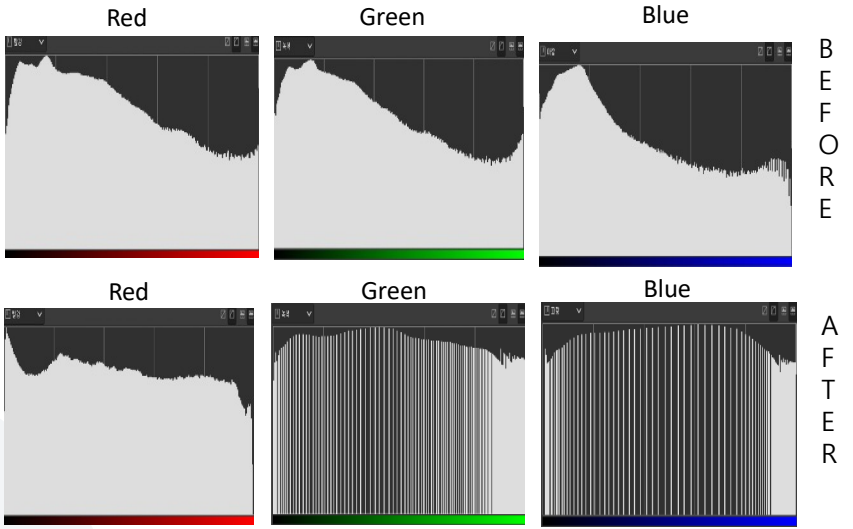
2) 문제점: 일부 픽셀 값의 정보를 손실

화소점 처리 - 히스토그램 평활화

히스토그램 평활화

화면





핵심 코드

```
// 기본 히스토그램
for (int h = 0; h < m_inH; h++) {
    for (int w = 0; w < m_inW; w++) {
        if (m_inImageR[h][w] > highestR) {
            // 최대값 구하기
            highestR = m_inImageR[h][w];
        }
        // 기본 히스토그램 (도수분포표) 작성
        hist[m_inImageR[h][w]] += 1;
    }
}
```

```
// 누적 히스토그램 첫 요소만 복사
sum_hist[0] = hist[0];
// 누적 도수분포표 작성
for (int idx = 1; idx <= 255; idx++) {
    sum_hist[idx] = sum_hist[idx - 1] + hist[idx];
}
//// 정규화 된 누적 히스토그램 작성
for (int idx = 0; idx <= 255; idx++) {
    norm_hist[idx] = sum_hist[idx] * (1.0 / pixel_n) * highestG;
}
```

$$sum[i] = \sum_{j=0}^{i} hist[j]$$

$$n[i] = sum[i] \times \frac{1}{N} \times I_{max}$$

세부 사항

1) 편향된 화소의 분포를 정규화하는 효과 → RGB의 대비를 효과적으로 증가

화소 영역 처리 - 엠보싱,블러링

엠보싱

블러링

화면

원본

엠보싱

고주파 샤프닝

블러링

핵심 코드

```
// 엠보싱 마스크
const int MSIZE = 3;
double mask[MSIZE][MSIZE] = {
    {-1.0, 0.0, 0.0},
    {0.0, 0.0, 0.0},
    {0.0, .0, 1.0} };
```

```
//회선 연산
for (int i = 0; i < m_inH; i++) {
    for (int k = 0; k < m_inW; k++) {
        double S_VALUE = 0.0;
        for (int m = 0; m < MSIZE; m++)
            for (int n = 0; n < MSIZE; n++)
                S_VALUE += tmpInImage[i + m][k + n] * mask[m][n];
        tmpInImage[i][k] = S_VALUE;
    }
}
```

```
const int MSIZE = 3;
// 블러링 마스크
double mask[MSIZE][MSIZE] = {
    {1. / 9, 1. / 9, 1. / 9},
    {1. / 9, 1. / 9, 1. / 9},
    {1. / 9, 1. / 9, 1. / 9} };
```

```
//회선 연산
for (int i = 0; i < m_inH; i++) {
    for (int k = 0; k < m_inW; k++) {
        double S_VALUE = 0.0;
        for (int m = 0; m < MSIZE; m++)
            for (int n = 0; n < MSIZE; n++)
                S_VALUE += tmpInImage[i + m][k + n] * mask[m][n];
        tmpInImage[i][k] = S_VALUE;
    }
}
```

세부 사항

마스크의 총 합이 0이 되기 때문에 후처리(+127)를 수행.

마스크의 총 합이 1이 되기 때문에 후처리를 수행하지 않음

화소 영역 처리 - 가우시안 스무딩, 고주파 샤프닝

가우시안 스무딩

고주파 샤프닝

화면

원본

가우시안 스무딩

고주파 샤프닝

블러링

핵심 코드

```
// 가우시안 스무딩 마스크
const int MSIZE = 3;
double mask[MSIZE][MSIZE] = {
    {1. / 16, 1. / 8, 1. / 16},
    {1. / 8, 1. / 4, 1. / 8},
    {1. / 16, 1. / 8, 1. / 16} };
```

```
//회선 연산
for (int i = 0; i < m_inH; i++) {
    for (int k = 0; k < m_inW; k++) {
        double S_VALUE = 0.0;
        for (int m = 0; m < MSIZE; m++)
            for (int n = 0; n < MSIZE; n++)
                S_VALUE += tmpInImage[i + m][k + n] * mask[m][n];
        tmpInImage[i][k] = S_VALUE;
    }
}
```

```
// 고주파 마스크
const int MSIZE = 3;
double boundary = -1.0 / 9;
double center = 8.0 / 9;
double mask[MSIZE][MSIZE] = {
    {boundary , boundary , boundary },
    {boundary , center   , boundary },
    {boundary , boundary , boundary } };
```

```
// 회선 연산 - 고주파 필터는 대상 영상에 값을 더하여
// 고주파 성분을 두드러지게 만듦
for (int i = 0; i < m_inH; i++) {
    for (int k = 0; k < m_inW; k++) {
        double S_VALUE = 0.0;
        for (int m = 0; m < MSIZE; m++)
            for (int n = 0; n < MSIZE; n++)
                S_VALUE += tmpInImage[i + m][k + n] * mask[m][n];
        double temp = tmpInImage[i][k] + S_VALUE;
        if (temp > 255.0)
            tmpInImage[i][k] = 255.0;
        else if (temp < 0.0)
            tmpInImage[i][k] = 0.0;
        else
            tmpInImage[i][k] = temp;
    }
}
```

세부 사항

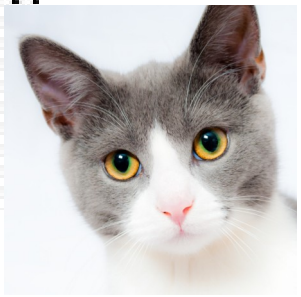
고주파 마스크는 고주파만 걸러내기 때문에
영상에 더하기 연산을 하여 고주파를 두드러지게 만듦

기하학 처리 - 축소, 확대

축소

화면

원본



축소

핵심 코드

```
// 출력 이미지 영상 처리
for (int i = 0; i < m_inH; i++) {
    for (int k = 0; k < m_inW; k++) {
        m_outImageR[(int)(i / scale)][(int)(k / scale)] = m_inImageR[i][k];
        m_outImageG[(int)(i / scale)][(int)(k / scale)] = m_inImageG[i][k];
        m_outImageB[(int)(i / scale)][(int)(k / scale)] = m_inImageB[i][k];
    }
}
```

세부 사항

1)출력 영상 크기를 n으로 나누고

2)입력좌표 / 2로 출력좌표를 구함

확대

원본

확대

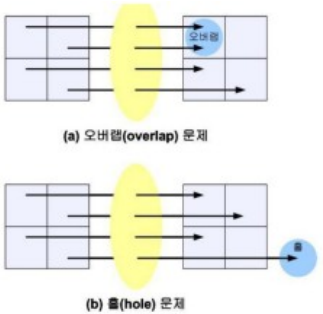
```
// 출력 이미지 영상 처리
for (int i = 0; i < m_outH; i++) {
    for (int k = 0; k < m_outW; k++) {
        m_outImageR[i][k] = m_inImageR[(int)(i / scale)][(int)(k / scale)];
        m_outImageG[i][k] = m_inImageG[(int)(i / scale)][(int)(k / scale)];
        m_outImageB[i][k] = m_inImageB[(int)(i / scale)][(int)(k / scale)];
    }
}
```

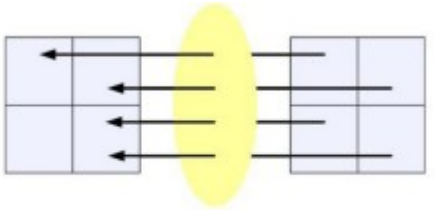
1)출력 영상 크기에 n을 곱하고

2)출력 영상의 좌표 / 2를 입력영상에 대입하여

3)백워딩으로 확대 -> 정보 손실이 감소

포워딩, 백워딩





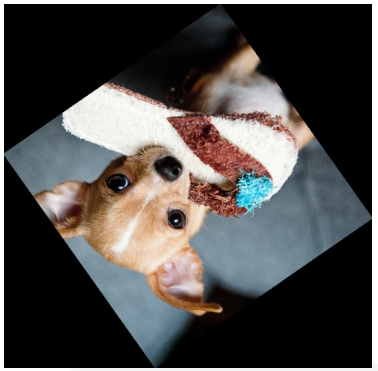
[그림 8-10] 역방향 사상의 동작

기하학 처리 - 배경을 고려한 회전

화면

원본

회전 후

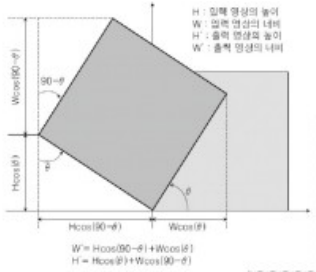


$$\begin{bmatrix} x_{dest} \\ y_{dest} \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_{source} - C_x \\ y_{source} - C_y \end{bmatrix} + \begin{bmatrix} C_x \\ C_y \end{bmatrix}$$

⊖ 만큼 회전한 이후 좌표의 위치를 구하는 공식

핵심 코드

```
// 입력한 각도 -> 라디안 값을 저장할 변수 , 배경의 크기 변수
double radian = 0.0; double radian_0 = 0.0; double radian_90 = 0.0;
// radian_0 , radian_90은 (90-radian) 값을 저장하는 변수
// 배경의 높이와 너비 값을 저장하는 변수
m_backH = 0; m_backW = 0;
// 각도 -> 라디안
radian = m_degree * 3.141592 / 180.0;
if (m_degree > 0 && m_degree <= 90) {
    radian_0 = radian;
    radian_90 = (90 - m_degree) * 3.141592 / 180.0;
}
```



배경의 크기를 구하는 공식

세부 사항

1) 높이와 너비가 다른 이미지는 둘 중 더 큰 값을 사용해서 정사각형 배경을 만듦

컬러효과 (HSI 모델)

화면

채도 변경

원본

+0.1

원하는 색상 추출

원본

빨강 추출

핵심 코드

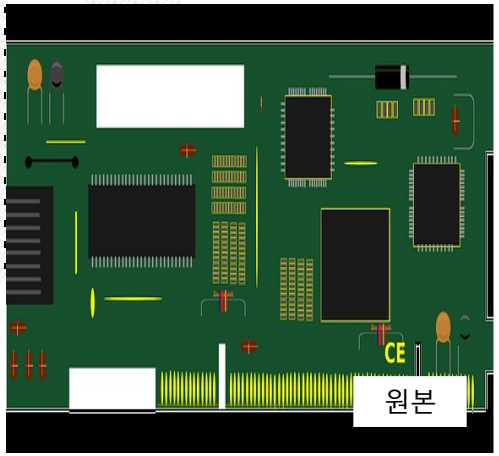
```
// 채도 변경
S += satDlg.m_saturation;
if (S < 0.0)
    S = 0.0;
else if (S > 1.0)
    S = 1.0;
// 다시 RGB로
unsigned char* rgb = OnHSI2RGB(H, S, I);
R = rgb[0];     G = rgb[1];     B = rgb[2];
```

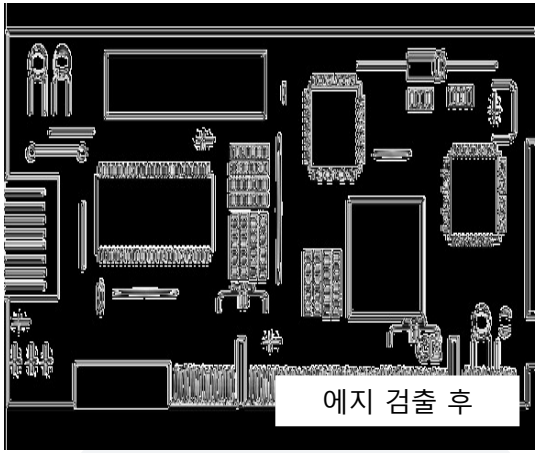
```
// 빨강 추출 (h 값 0~20, 330~360)
int left = extractDlg.m_extractLeft;
int right = extractDlg.m_extractRight;

// 지정된 범위 사이의 값은 그대로 나머지는 Gray Sclae로 출력
if (right < left) {
    if ((right >= H && 0 <= H) || (H >= left && H <= 360)) {
        m_outImageR[h][w] = m_inImageR[h][w];
        m_outImageG[h][w] = m_inImageG[h][w];
        m_outImageB[h][w] = m_inImageB[h][w];
    }
    else {
        double avg = (m_inImageR[h][w] + m_inImageG[h][w] + m_inImageB[h][w]) / 3.0;
        m_outImageR[h][w] = unsigned char(avg);
        m_outImageG[h][w] = unsigned char(avg);
        m_outImageB[h][w] = unsigned char(avg);
    }
}
```

세부 사항

그레이스케일 에지 검출 - 1차원 프리윗

화면





핵심 코드

```
const int MSIZE = 3;
// 행 검출 마스크
double mask_col[MSIZE][MSIZE] = { {-1.0,-1.0,-1.0},
                                  {0.0,0.0,0.0},
                                  {1.0,1.0,1.0} };

// 열 검출 마스크
double mask_row[MSIZE][MSIZE] = { {1.0,0.0,-1.0},
                                  {1.0,0.0,-1.0},
                                  {1.0,0.0,-1.0} };
```

```
// 회선 연산
for (int i = 0; i < m_outH; i++) {
    for (int k = 0; k < m_outW; k++) {
        double C_VALUE = 0.0;
        double R_VALUE = 0.0;
        for (int m = 0; m < MSIZE; m++)
            for (int n = 0; n < MSIZE; n++) {
                C_VALUE += tmpoutImageCol[i + m][k + n] * mask_col[m][n];
                R_VALUE += tmpoutImageRow[i + m][k + n] * mask_row[m][n];
            }
        m_outImageR[i][k]= m_outImageG[i][k] = m_outImageB[i][k] = C_VALUE + R_VALUE;
    }
}
```

세부 사항

행 검출 마스크 결과와 열 검출 마스크 결과를 더하여 수직, 수평 에지를 모두 검출

프리윗

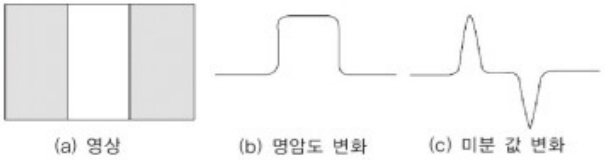
행 검출 마스크

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

열 검출 마스크

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

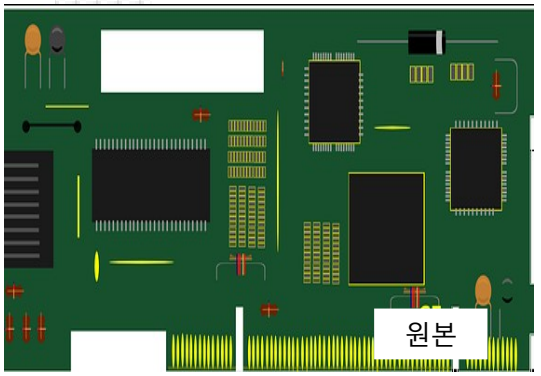
1. 위, 아래 주변 행과 차이가 클 수록 차이가 명확해진다.
2. 양 옆의 열과 차이가 클 수록 차이가 명확해진다.

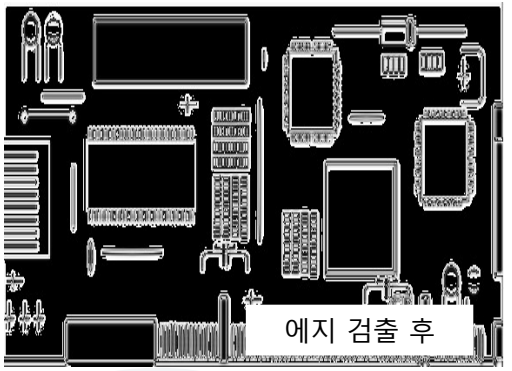


1차 미분으로 에지를 검출하는 원리

그레이스케일 에지 검출 - 2차원 LoG

화면:





1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

가우시안 필터

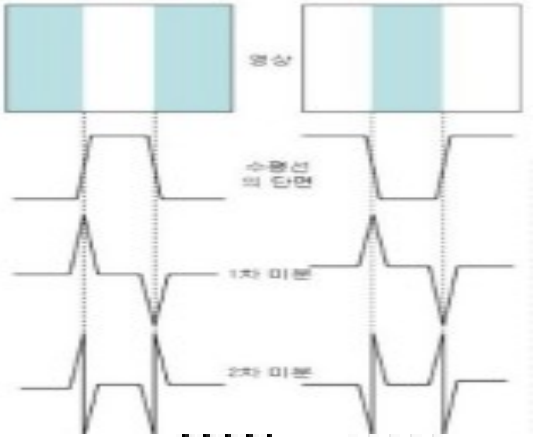
$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

라플라시안 마스크

핵심 코드

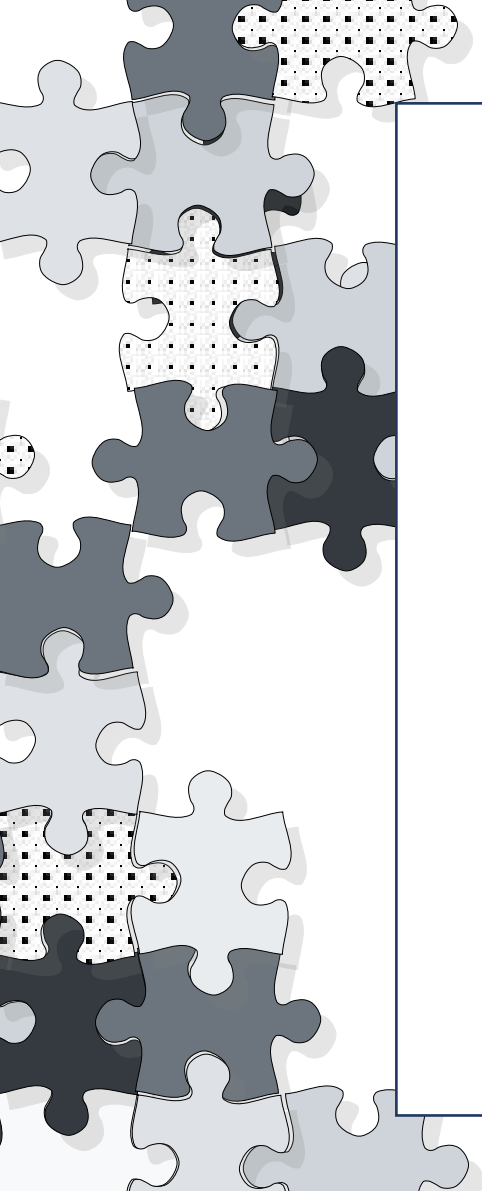
```
// 회선 연산: 가우시안 스무딩
for (int i = 0; i < m_outH; i++) {
    for (int k = 0; k < m_outW; k++) {
        double G_VALUE = 0.0;
        for (int m = 0; m < MSIZE; m++)
            for (int n = 0; n < MSIZE; n++) {
                G_VALUE += tmpoutImage[i + m][k + n] * mask_gau[m][n];
            }
        tmpoutImage[i + 1][k + 1] = G_VALUE;
    }
}
```

```
// 회선 연산: 라플라시안
for (int i = 0; i < m_outH; i++) {
    for (int k = 0; k < m_outW; k++) {
        double L_VALUE = 0.0;
        for (int m = 0; m < MSIZE; m++)
            for (int n = 0; n < MSIZE; n++) {
                L_VALUE += tmpoutImage[i + m][k + n] * mask_lap[m][n];
            }
        m_outImageR[i][k] = L_VALUE;
        m_outImageG[i][k] = L_VALUE;
        m_outImageB[i][k] = L_VALUE;
    }
}
```



세부 사항

1차원 에지 검출보다 덜 민감하고 예리한 결과



마무리

- **느낀점**
  - ✓ 영상처리 알고리즘을 계산해보고 그래프도 그려보면서 알고리즘의 정확한 역할을 제대로 알 수 있었고 다양하게 응용할 수 있다고 느낌
  - ✓ 잘 만들어진 S/W, 코드, 문서를 경험해보는 일이 중요하다고 느낌

- **보완하고 싶은 점**
  - ✓ 더 성능이 좋은 화소 보간법을 구현해보고 싶다.
  - ✓ AI 컴퓨터 비전에 중요하게 사용되는 에지검출에 대해 더 공부해보고 싶다.
  - ✓ 보고서를 더 깔끔하게 만들고 싶다.