[Intel] 엣지 AI SW 아카데미 _ 절차 지향 프로그래밍

C 언어로 구현한 GrayScale Image Processing

임지원

프로젝트 개요

목표

OpenCV 없이 C 언어를 기반으로 영상 처리 프로그램을 구현

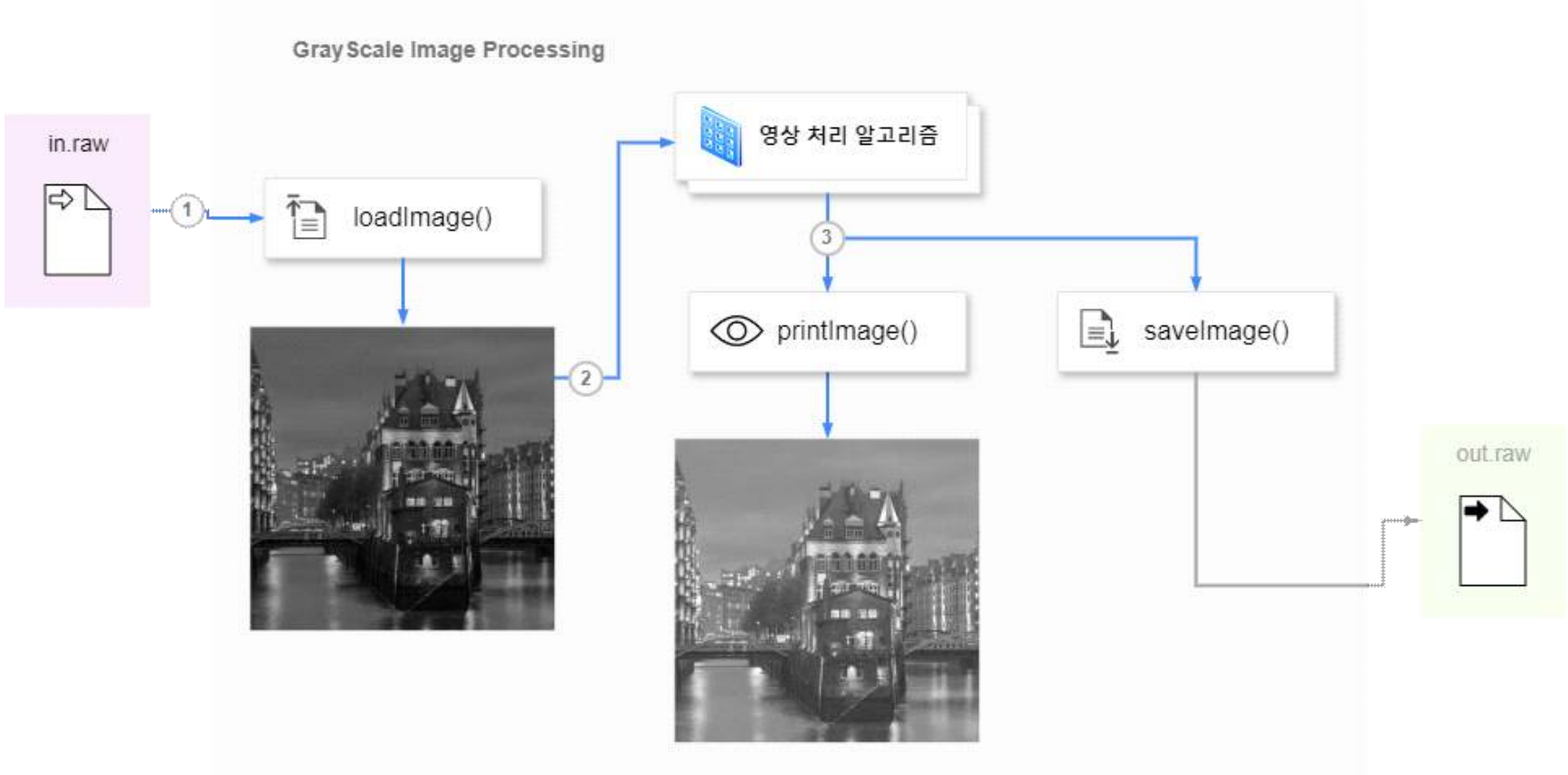
개발 환경

OS : Windows 11 Pro
Tool : Microsoft Visual Studio 2022

코드

https://github.com/Jday4612/GrayScale-Image-Processing

프로젝트 구조도


Gray Scale Image Processing
in.raw
1
loadImage()
2
영상 처리 알고리즘
3
printImage()
saveImage()
out.raw

메인 화면

```
## GrayScale Image Processing (Version 1.0) ##

            0.열기 1.저장 9.종료

+-----------------------------------------------+

A.동일(Null) B1.밝게(덧셈) B2.어둡게(뺀셈) B3.곱셈 B4.나눗셈
C1.AND C2.OR C3.XOR C4.NOT(반전) D.감마 E.파라볼라
F1.히스토그램 - 스트레칭(명암 - 대비 - 스트레칭) F2.endIn F3.평활화
G.포스터라이징 H.강조(에신) I.모핑
J1.흑백 J2.흑백(평균값) J3.흑백(중앙값)

K1.확대(포워딩) K2.확대(백워딩) K3.확대(양선형 보간)
L1.축소 L2.축소(평균값) L3.축소(중앙값)
M1.미러링(상하) M2.미러링(좌우)
N1.회전(기본) N2.회전(중앙 + 백워딩) N3.회전(확대) N4.회전(확대 + 양선형) O.이동

P1.엠보싱 P2.블러링(3x3) P3.블러링(9x9) P4.샤프닝 P5.샤프닝(고주파) P6.샤프닝(저주파) P7.가우시안
Q1.경계선(수직) Q2.경계선(수평) Q3.경계선(유사-연산자) Q4.경계선(차-연산자)
R1.경계선(로버츠) R2.경계선(프리윗) R3.경계선(소벨) R4.경계선(라플라시안) R5.경계선(LoG) R6.경계선(DoG)
```

열기 및 저장

열어 볼 파일명-->house256

열기

[0]을 누르고 파일명을 입력 시,
해당 파일을 열고 print함

저장할 파일명-->저장예시

[저장 완료]

저장예시.raw

저장

[1]을 누르고 파일명을 입력 시,
raw 파일이 저장됨

화소 점 처리

: 원 화소의 값이나 위치를 바탕으로 단일 화소 값을 변경하는 기술

Input

Output

동일

[A]를 누를 시,
입력 값을 그대로 출력 값으로 변환

```
outImage[i][j] = inImage[i][j];
```

화소 점 처리

Input

Output

밝게

[B1]를 누르고 정수를 입력 시,
해당 값을 더해 밝기 증가

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

화소 점 처리

Input

Output

어둡게

[B2]를 누르고 정수를 입력 시,
해당 값을 빼 밝기 감소

```
if (0 <= inImage[i][j] - val)
    outImage[i][j] = inImage[i][j] - val;
else
    outImage[i][j] = 0;
```

화소 점 처리

Input

Output

곱셈

[B3]를 누르고 정수를 입력 시,
해당 값을 곱해 전체적으로 밝기 증가

```
if (inImage[i][j] * val < 0)
    outImage[i][j] = 0;
else if (255 < inImage[i][j] * val)
    outImage[i][j] = 255;
else
    outImage[i][j] = inImage[i][j] * val;
```

화소 점 처리

Input

Output

나눗셈

[B4]를 누르고 정수를 입력 시,
해당 값을 나눠 전체적으로 밝기 감소

```
if (inImage[i][j] / val < 0)
    outImage[i][j] = 0;
else if (255 < inImage[i][j] / val)
    outImage[i][j] = 255;
else
    outImage[i][j] = inImage[i][j] / val;
```

화소 점 처리

Input

Output

AND

[C1]를 누를 시,
특정 데이터와 AND 연산 수행

```
if ((inImage[i][j] & dataImage[i][j]) < 0)
    outImage[i][j] = 0;
else if (255 < (inImage[i][j] & dataImage[i][j]))
    outImage[i][j] = 255;
else
    outImage[i][j] = inImage[i][j] & dataImage[i][j];
```

화소 점 처리

Input

Output

OR

[C2]를 누를 시,
특정 데이터와 OR 연산 수행

```
if ((inImage[i][j] | dataImage[i][j]) < 0)
    outImage[i][j] = 0;
else if (255 < (inImage[i][j] | dataImage[i][j]))
    outImage[i][j] = 255;
else
    outImage[i][j] = inImage[i][j] | dataImage[i][j];
```

화소 점 처리

Input

Output

XOR

[C3]를 누를 시,
특정 데이터와 XOR 연산 수행

```
if ((inImage[i][j] ^ dataImage[i][j]) < 0)
    outImage[i][j] = 0;
else if (255 < (inImage[i][j] ^ dataImage[i][j]))
    outImage[i][j] = 255;
else
    outImage[i][j] = inImage[i][j] ^ dataImage[i][j];
```

화소 점 처리

Input

Output

NOT

[C4]를 누를 시,
화소 비트가 반전됨

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

화소 점 처리

Input

Output

감마

[D]를 누르고 실수를 입력 시,
해당 값에 따라 밝기 조절

```
x = inImage[i][j];
outImage[i][j] = 255.0 * pow(x / 255.0, gamma);
```

화소 점 처리

Input

파라볼라

[E]를 누를 시,
파라볼라 CAP/CUP 수식으로 처리

```
x = inImage[i][j];
outImage[i][j] = -255.0 * pow(x / 127.0 - 1.0, 2) + 255.0;
```

```
x = inImage[i][j];
outImage[i][j] = 255.0 * pow(x / 127.0 - 1.0, 2);
```

화소 점 처리

Input

Output

포스터라이징

[G]를 누르고 정수를 입력 시,
해당 값으로 명암 값 수를 변경

```
for (int k = 1; k < bit; k++) { // input 간격 : bit - 1개, output 간격 : bit - 2개
    if (k == (bit - 1)) // 마지막 구간
        outImage[i][j] = 255;
    else if (inImage[i][j] <= 255 / (bit - 1) * k) {
        if (k == 1) // 첫 번째 구간
            outImage[i][j] = 0;
        else // 나머지 구간
            outImage[i][j] = 255 / (bit - 2) * (k - 1);

        break;
    }
}
```

화소 점 처리

Input

Output

강조

[H]를 누르고 정수 2개를 입력 시,
해당 범위 내의 화소만 강조

```
if (val1 <= inImage[i][j] && inImage[i][j] <= val2)
    outImage[i][j] = 255;
else
    outImage[i][j] = inImage[i][j];
```

화소 점 처리

Input

Output

흑백

[J1]을 누를 시,
이진화 적용

```
if (inImage[i][j] < 128)
    outImage[i][j] = 0;
else
    outImage[i][j] = 255;
```

화소 점 처리

Input

Output

흑백 (평균값)

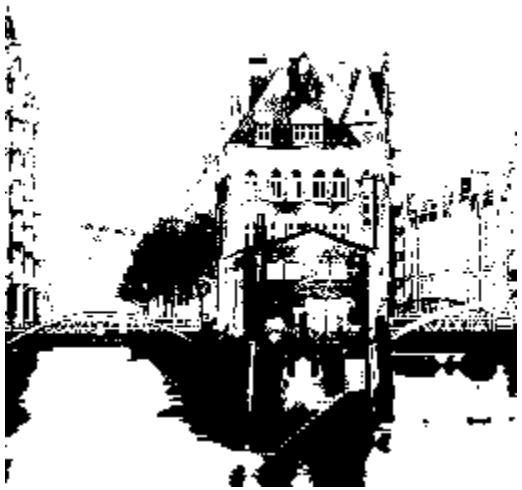
[J2]를 누를 시,
평균값을 기준으로 이진화 적용

```
int avg = 0;

for (int i = 0; i < inH; i++)
    for (int j = 0; j < inW; j++)
        avg += inImage[i][j];

avg /= inH * inW;

for (int i = 0; i < inH; i++) {
    for (int j = 0; j < inW; j++) {
        if (inImage[i][j] < avg)
            outImage[i][j] = 0;
        else
            outImage[i][j] = 255;
    }
}
```

화소 점 처리

Input

Output



흑백 (중앙값)

[J3]을 누를 시,
중앙값을 기준으로 이진화 적용

```
for (int i = 0; i < inH; i++) {
    for (int j = 0; j < inW; j++) {
        arr[cnt] = inImage[i][j];
        cnt++;
    }
}

qsort(arr, cnt, sizeof(unsigned char), compare);

med = arr[inH * inW / 2];

for (int i = 0; i < inH; i++) {
    for (int j = 0; j < inW; j++) {
        if (inImage[i][j] < med)
            outImage[i][j] = 0;
        else
            outImage[i][j] = 255;
    }
}
```

기하학 처리

: 영상을 구성하는 화소의 공간적 위치를 재배치하는 기술

Input

Output

축소

[L1]을 누르고 정수를 입력 시,
포워딩으로 (해당 값)배 축소

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

기하학 처리

Input

Output

축소 (평균값)

[L2]을 누르고 정수를 입력 시,
평균값을 기준으로 (해당 값)배 축소

```
long long avg = 0;

for (int n = 0; n < scale; n++)
    for (int m = 0; m < scale; m++)
        avg += inImage[i + n][j + m];

outImage[(int)(i / scale)][(int)(j / scale)] = avg / (scale * scale);
```

기하학 처리

Input

Output

축소 (중앙값)

[L3]을 누르고 정수를 입력 시,
중앙값을 기준으로 (해당 값)배 축소

```
int med, cnt = 0;

for (int n = 0; n < scale; n++) {
    for (int m = 0; m < scale; m++) {
        arr[cnt] = inImage[i + n][j + m];
        cnt++;
    }
}

qsort(arr, cnt, sizeof(unsigned char), compare);
med = arr[scale * scale / 2];
outImage[(int)(i / scale)][(int)(j / scale)] = med;
```

기하학 처리

Input

Output

확대 (포워딩)

[K1]를 누르고 정수를 입력 시,
포워딩으로 (해당 값)배 확대

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

기하학 처리

Input

Output

확대 (백워딩)

[K2]를 누르고 정수를 입력 시,
백워딩으로 (해당 값)배 확대

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

기하학 처리

Input

Output

확대 (양선형 보간)

[K3]를 누르고 정수를 입력 시,
양선형 보간으로 (해당 값)배 확대

```
double rH = (double)i / scale;
double rW = (double)j / scale;

int iH = (int)floor(rH);
int iW = (int)floor(rW);

double sH = rH - iH;
double sW = rW - iW;

if (iH < 0 || inH - 1 <= iH || iW < 0 || inW - 1 <= iW)
    outImage[i][j] = 255;
else {
    double C1 = (double)inImage[iH][iW];
    double C2 = (double)inImage[iH][iW + 1];
    double C3 = (double)inImage[iH + 1][iW + 1];
    double C4 = (double)inImage[iH + 1][iW];

    outImage[i][j] = (unsigned char)(C1 * (1 - sH) * (1 - sW)
        + C2 * sW * (1 - sH) + C3 * sW * sH + C4 * (1 - sW) * sH);
}
```

기하학 처리

Input

Output

미러링 (상하)

[M1]를 누를 시,
상하 대칭 적용

```
outImage[i][j] = inImage[inH - 1 - i][j];
```

기하학 처리

Input

Output

미러링 (좌우)

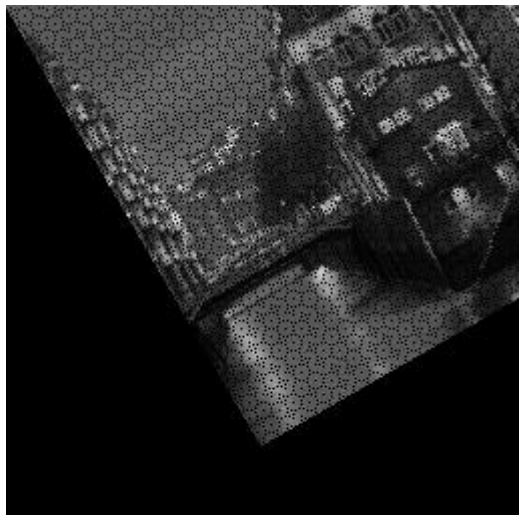
[M2]를 누를 시,
좌우 대칭 적용

```
outImage[i][j] = inImage[i][inH - 1 - j];
```

기하학 처리

Input

Output



회전

[N1]를 누르고 정수를 입력 시,
해당 값만큼 회전

```
for (int i = 0; i < outH; i++)
    for (int j = 0; j < outW; j++)
        outImage[i][j] = 0;

int degree = getIntValue();
double radian = degree * 3.141592 / 180.0;

// 입력 배열 --> 출력 배열
for (int i = 0; i < inH; i++) {
    for (int j = 0; j < inW; j++) {
        int xs = i;
        int ys = j;

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

        if ((0 <= xd && xd < outH) && (0 <= yd && yd < outW))
            outImage[xd][yd] = inImage[xs][ys];
    }
}
```

기하학 처리

Input

Output

회전 (중앙 + 백워딩)

[N2]를 누르고 정수를 입력 시,
중앙 + 백워딩으로 해당 값만큼 회전

```
for (int i = 0; i < outH; i++)
    for (int j = 0; j < outW; j++)
        outImage[i][j] = 0;

int degree = getIntValue();
double radian = degree * 3.141592 / 180.0;

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

// 입력 배열 --> 출력 배열
for (int i = 0; i < outH; i++) {
    for (int j = 0; j < outW; j++) {
        int xd = i;
        int yd = j;

        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;

        if ((0 <= xs && xs < outH) && (0 <= ys && ys <= outW))
            outImage[xd][yd] = inImage[xs][ys];
    }
}
```

기하학 처리

Input

회전 (확대)

[N3]을 누르고 정수 두 개를 입력 시,
해당 값만큼 회전 및 확대

```
// 확대
for (int i = 0; i < outH; i++) {
    for (int j = 0; j < outW; j++)
        tmpImage[i][j] = inImage[(int)(i / scale)][(int)(j / scale)];
}

// 회전
for (int i = 0; i < outH; i++)
    for (int j = 0; j < outW; j++)
        outImage[i][j] = 0;

int cx = tmpH / 2;
int cy = tmpW / 2;

for (int i = 0; i < outH; i++) {
    for (int j = 0; j < outW; j++) {
        int xd = i;
        int yd = j;

        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;

        if ((0 <= xs && xs < outH) && (0 <= ys && ys <= outW))
            outImage[xd][yd] = tmpImage[xs][ys];
    }
}
```

기하학 처리

Input

회전 (확대 + 양선형)

[N4]을 누르고 정수 두 개를 입력 시,
양선형 보간으로 해당 값만큼 회전 및 확대

기하학 처리

Input

Output

이동

[O]를 누르고 정수 2개를 입력 시,
기존 위치에서 해당 값만큼 이동

```
for (int i = 0; i < outH; i++)
    for (int j = 0; j < outW; j++)
        outImage[i][j] = 0;

// 입력 배열 --> 출력 배열
for (int i = 0; i < inH - posH; i++) {
    for (int j = 0; j < inW - posW; j++) {
        int nx = i + posH;
        int ny = j + posW;

        if ((0 <= nx && nx < outH) && (0 <= ny && ny < outW))
            outImage[nx][ny] = inImage[i][j];
    }
}
```

기하학 처리

Input

Output

모핑

[]을 누르고 다른 파일명을 입력 시,
모핑 변환 (두 파일을 혼합)

```
u = ((i * inH) + (j + 1.0)) / (inH * inW);
tmp = (int)((1.0 - u) * inImage[i][j] + u * tmpImage[i][j]);

if (255 < tmp)
    outImage[i][j] = 255;
else if (tmp < 0)
    outImage[i][j] = 0;
else
    outImage[i][j] = (unsigned char) tmp;
```

히스토그램 처리

히스토그램 : 데이터를 막대 그래프 모양으로 나타낸 것

Input

Output

히스토그램 스트레칭

[F1]를 누를 시,
특정 부분에 집중된 히스토그램을
모든 영역으로 확장

```
int high = inImage[0][0], low = inImage[0][0];

for (int i = 0; i < inH; i++) {
    for (int j = 0; j < inW; j++) {
        if (inImage[i][j] < low)
            low = inImage[i][j];

        if (high < inImage[i][j])
            high = inImage[i][j];
    }
}

int old, new;

for (int i = 0; i < inH; i++) {
    for (int j = 0; j < inW; j++) {
        old = inImage[i][j];
        new = (int)((double)(old - low) / (double)(high - low) * 255.0);

        if (new < 0)
            new = 0;

        if (255 < new)
            new = 255;

        outImage[i][j] = new;
    }
}
```

히스토그램 처리

Input

Output

endIn

[F2]를 누를 시,
일정한 양의 화소를 흰색이나 검정색으로 지정

```
int high = inImage[0][0], low = inImage[0][0];

for (int i = 0; i < inH; i++) {
    for (int j = 0; j < inW; j++) {
        if (inImage[i][j] < low)
            low = inImage[i][j];
        if (inImage[i][j] > high)
            high = inImage[i][j];
    }
}

high -= 50;
low += 50;

int old, new;
for (int i = 0; i < inH; i++) {
    for (int j = 0; j < inW; j++) {
        old = inImage[i][j];

        new = (int)((double)(old - low) / (double)(high - low) * 255.0);

        if (new > 255)
            new = 255;

        if (new < 0)
            new = 0;

        outImage[i][j] = new;
    }
}
```

히스토그램 처리

Input

Output

평활화

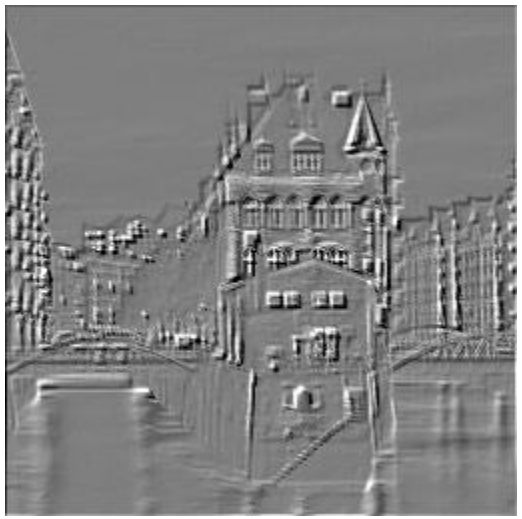
[F3]를 누를 시,
영상의 밝기 분포를 재분배하여 명암 대비를 최대화

```
// 1단계 : 빈도 수 세기 (=히스토그램) histo[256]
int histo[256] = { 0 };

for (int i = 0; i < inH; i++)
    for (int j = 0; j < inW; j++)
        histo[inImage[i][j]]++;

// 2단계 : 누적 히스토그램 생성
int sumHisto[256] = { 0 };
sumHisto[0] = histo[0];

for (int i = 1; i < 256; i++)
    sumHisto[i] = sumHisto[i - 1] + histo[i];

// 3단계 : 정규화된 히스토그램 생성 normalHisto = sumHist* (1.0 / (inH * inW)) * 255.0
double normalHisto[256] = { 0 };

for (int i = 0; i < 256; i++)
    normalHisto[i] = sumHisto[i] * (1.0 / (inH * inW)) * 255.0;

// 입력 배열 --> 출력 배열
for (int i = 0; i < inH; i++) {
    for (int j = 0; j < inW; j++) {
        outImage[i][j] = (unsigned char)normalHisto[inImage[i][j]];
    }
}
```

영역 처리

Input

Output



엠보싱

[P1]를 누를 시,
양각 형태로 변환

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

```
for (int i = 0; i < inH; i++) {
    for (int j = 0; j < inW; j++) {
        // 마스크(3x3)와 한 점을 중심으로 한 3x3 곱하기
        S = 0.0; // 마스크 9개와 입력값 9개를 각각 곱해서 합한 값

        for (int m = 0; m < 3; m++)
            for (int n = 0; n < 3; n++)
                S += tmpInImage[i + m][j + n] * mask[m][n];

        tmpOutImage[i][j] = S;
    }
}

// 후처리 (마스크 값의 합계에 따라서)
for (int i = 0; i < outH; i++)
    for (int j = 0; j < outW; j++)
        tmpOutImage[i][j] += 127.0;
```

영역 처리

Input

Output

블러링 (3x3)

[P2]를 누를 시,
3x3 마스크로 영상의 세밀한 부분을 제거

```
double mask[3][3] = { { 1.0 / 9, 1.0 / 9, 1.0 / 9 },
    { 1.0 / 9, 1.0 / 9, 1.0 / 9 },
    { 1.0 / 9, 1.0 / 9, 1.0 / 9 } }; // 블러링 마스크
```

영역 처리

Input

Output

블러링 (9x9)

[P3]를 누를 시,
9x9 마스크로 영상의 세밀한 부분을 제거

```
double mask[9][9]; // 블러링 마스크

for (int i = 0; i < 9; i++)
    for (int j = 0; j < 9; j++)
        mask[i][j] = 1.0 / 81;
```

영역 처리

Input

Output

샤프닝

[P4]를 누를 시,
영상의 상세한 부분을 더욱 강조

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

영역 처리

Input

Output

샤프닝 (고주파)

[P5]를 누를 시,
고주파 통과 필터 적용

```
double mask[3][3] = { { -1.0 / 9, -1.0 / 9, -1.0 / 9 },
    { -1.0 / 9, 8.0 / 9, -1.0 / 9 },
    { -1.0 / 9, -1.0 / 9, -1.0 / 9 } }; // 고주파 샤프닝 마스크
```

```
// 마스크(3x3)와 한 점을 중심으로 한 3x3 곱하기
S = 0.0; // 마스크 9개와 입력값 9개를 각각 곱해서 합한 값

for (int m = 0; m < 3; m++)
    for (int n = 0; n < 3; n++)
        S += tmpInImage[i + m][j + n] * mask[m][n] * 20;
```

영역 처리

Input

Output

샤프닝 (저주파)

[P6]을 누르고 정수를 입력 시,
(원 영상) – (저주파 통과 필터링 결과 영상)

```
double mask[3][3] = { { 1.0 / 9, 1.0 / 9, 1.0 / 9 },
    { 1.0 / 9, 1.0 / 9, 1.0 / 9 },
    { 1.0 / 9, 1.0 / 9, 1.0 / 9 } }; // 저주파 샤프닝 마스크
```

```
        unsharp = alpha * inImage[i][j] - tmpOutImage[i][j];
```

영역 처리

Input

Output

가우시안

[P7]을 누를 시,
영상의 세세한 부분을 제거

```
double mask[3][3] = { { 1.0 / 16, 1.0 / 8, 1.0 / 16 },
                      { 1.0 / 8, 1.0 / 4, 1.0 / 8 },
                      { 1.0 / 16, 1.0 / 8, 1.0 / 16 } }; // 가우시안 마스크
```

영역 처리

Input

Output

경계선 (수직)

[Q1]을 누를 시,
수직 경계선 검출

```
double mask[3][3] = { { 0.0, 0.0, 0.0 },
    { -1.0, 1.0, 0.0 },
    { 0.0, 0.0, 0.0 } }; // 수직 에지 검출 마스크
```

영역 처리

Input

Output

경계선 (수평)

[Q2]를 누를 시,
수평 경계선 검출

```
double mask[3][3] = { { 0.0, -1.0, 0.0 },
    { 0.0, 1.0, 0.0 },
    { 0.0, 0.0, 0.0 } }; // 수평 에지 검출 마스크
```

영역 처리

Input

Output

경계선 (유사 연산자)

[Q3]을 누를 시,
화소를 감산한 값에서 최대값
을 결정하여 경계선 검출

```
S = 0.0;

for (int m = 0; m < 3; m++) {
    for (int n = 0; n < 3; n++) {
        if (S < fabs(tmpInImage[i + 1][j + 1] - tmpInImage[i + m][j + n]))
            S = fabs(tmpInImage[i + 1][j + 1] - tmpInImage[i + m][j + n]);
    }
}

tmpOutImage[i][j] = S;
```

영역 처리

Input

Output

경계선 (차 연산자)

[Q4]를 누를 시,
뺄셈 연산이 유사 연산자와는 달리 화소당
네 번만 사용되어 빠르게 경계선 검출

```
S = 0, 0;

for (int n = 0; n < 3; n++) {
    if (S < fabs(tmpInImage[i][j + n] - tmpInImage[i + 2][j + - n + 2]))
        S = fabs(tmpInImage[i][j + n] - tmpInImage[i + 2][j + - n + 2]);
}

if (S < fabs(tmpInImage[i + 1][j + 2] - tmpInImage[i + 1][j]))
    S = fabs(tmpInImage[i + 1][j + 2] - tmpInImage[i + 1][j]);

tmpOutImage[i][j] = S;
```

영역 처리

Input

Output

경계선 (로버츠)

[R1]을 누를 시,
로버츠 경계선 검출

```
double maskRow[3][3] = { { -1.0, 0.0, 0.0 },
    { 0.0, 1.0, 0.0 },
    { 0.0, 0.0, 0.0 } }; // 로버츠 행 에지 검출 마스크
double maskCol[3][3] = { { 0.0, 0.0, -1.0 },
    { 0.0, 1.0, 0.0 },
    { 0.0, 0.0, 0.0 } }; // 로버츠 열 에지 검출 마스크
```

영역 처리

Input

Output

경계선 (프리윗)

[R2]을 누를 시,
프리윗 경계선 검출

```
double maskRow[3][3] = { { -1.0, -1.0, -1.0 },
    { 0.0, 0.0, 0.0 },
    { 1.0, 1.0, 1.0 } }; // 프리윗 행 에지 검출 마스크
double maskCol[3][3] = { { 1.0, 0.0, -1.0 },
    { 1.0, 0.0, -1.0 },
    { 1.0, 0.0, -1.0 } }; // 프리윗 열 에지 검출 마스크
```

영역 처리

Input

Output

경계선 (소벨)

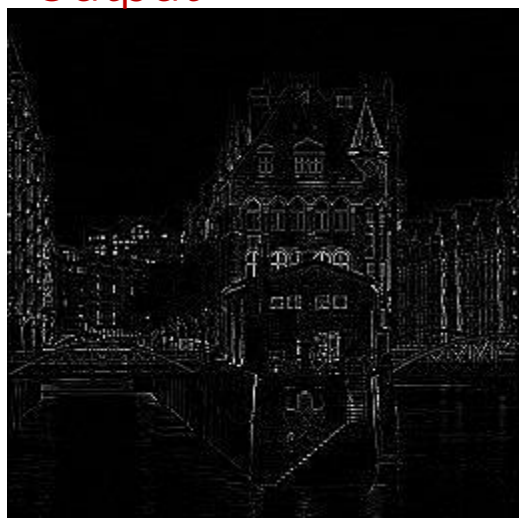
[R3]을 누를 시,
소벨 경계선 검출

```
double maskRow[3][3] = { { -1.0, -2.0, -1.0 },
    { 0.0, 0.0, 0.0 },
    { 1.0, 2.0, 1.0 } }; // 소벨 행 에지 검출 마스크
double maskCol[3][3] = { { 1.0, 0.0, -1.0 },
    { 2.0, 0.0, -2.0 },
    { 1.0, 0.0, -1.0 } }; // 소벨 열 에지 검출 마스크
```

영역 처리

Input

Output



경계선 (라플라시안)

[R4]을 누를 시,
라플라시안 경계선 검출

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

영역 처리

Input

Output

경계선 (LoG)

[R5]을 누를 시,
LoG 경계선 검출

```
double mask[5][5] = { { 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 } }; // LoG 에지 검출 마스크
```

영역 처리

Input

Output

경계선 (DoG)

[R6]을 누를 시,
DoG 경계선 검출

```
double mask[7][7] = { { 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 } }; // DoG 에지 검출 마스크
```

오류 처리

이미지를 열지 않고 다른 기능 실행 시

```
** 열려 있는 이미지가 존재하지 않음 **
```

-> 메인 화면

열린 이미지가 없는 상태로 저장 시도 시

```
** 저장할 이미지가 존재하지 않음 **
```

-> 메인 화면

정리

느낀점

직접 영상 처리 프로그램을 구현해 봄으로써 영상 처리에 있어서 어떠한 기능들이 있고, 각각의 기능들이 어떤 과정을 통해 작동하는지를 보기만 했을 때보다 더 잘 알고, 이해할 수 있는 기회가 되었던 것 같다.

향후 발전 방향

- UI 개선
- 특정 입력 값으로 인한 오류 처리
- 더 다양한 기능 추가
- 필터 크기 입력 받아 적용