# [Intel] 엣지 AI S/W 아카데미

절차 지향 프로그래밍

조수환

OpenCV 없이 구현한 C언어 영상처리

# 프로젝트 개요

# 프로젝트 개요

## 목적

- 엣지 디바이스 최적화를 위한 경량 영상처리 솔루션 개발
- C언어 프로그래밍 역량 강화

## 접근 방식

- C 언어로 영상처리 파이프라인 직접 구현
  - 이미지 파일 입출력
  - 영상처리 알고리즘 (밝기 조절, 필터링, 에지 검출 등)

## 개발 환경

- 운영 체제 : Windows 11
- 프로그래밍 언어 : C
- 개발 도구 : Visual Studio 2022

# 구현 기능

# 구현 기능

## 파일 입출력

- 파일에서 이미지 불러오기 (loadImage)
- 이미지를 파일로 저장 (saveImage)

## 입력/출력

- 사용자로부터 값 입력 받기 (getInValue, getFlValue)

## 유틸리티

- 이미지 데이터 출력 (printImage)
- 애플리케이션 메뉴 출력 (printMenu)

## 메모리 관리

- 메모리 할당 (mallocInputMemory, mallocOutputMemory, mallocDoubleMemory)
- 메모리 해제 (freeInputMemory, freeOutputMemory, freeDoubleMemory)

# 구현 기능

## 점 변환

- 밝기 조절 (addImage)
- 영상 반전 (reverseImage)
- 이진화 (binaryImage)
- 감마 보정 (gammaImage)
- 파라볼릭 변환 (paraImage)
- 포스터라이징 (postImage)
- 범위 강조 (emphImage)

## 히스토그램 처리

- 히스토그램 스트레칭 (histoStretch)
- 히스토그램 평활화 (histoEqual)

## 기하학적 변환

- 크기 조절 (scaleImage)
- 이동 (moveImage)
- 회전 (rotatImage)
- 미러링 (mirrorImage)

## 영역 변환

- 엠보싱 / 블러링 / 스무딩 / 샤프닝 (emboss)
- 에지 검출(LOG) (edgeLOG)
- 에지 검출(DOG) (edgeDOG)

# 기능 설명
## (점 변환)

# 점 변환 :원 화소의 값이나 위치를 바탕으로 단일 화소 값을 변경하는 기술

## 1. 밝기 조절 :각 픽셀에 일정한 값을 더하여 영상의 전체 밝기를 균등하게 조정

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

-50

원본

+100

결과 값이 최대값과 최소값을 벗어나지 않게 클래핑(Clapping) 기법 적용

```
if (inImage[i][j] + val >= 0) // val이 음수면
    outImage[i][j] = 0;
if (inImage[i][j] + val <= 255)
    outImage[i][j] = 255;
```

## 2. 영상 반전 : 각 화소의 값이 영상 내에 대칭이 되는 값으로 변환

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

반전

원본

# 점 변환 :원 화소의 값이나 위치를 바탕으로 단일 화소 값을 변경하는 기술

## 3. 이진화 :특정 임계값을 사용하여 영상을 흑백(이진) 형식으로 변환


t = 50


원본


t = 200


t = 127

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

## 4. 감마 보정 : 함수의 감마 값에 따라 영상을 밝게 하거나 흐리게 조절

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


r = 0.5


원본


r =2.0

# 점 변환 :원 화소의 값이나 위치를 바탕으로 단일 화소 값을 변경하는 기술

## 5. 파라볼릭 변환 :이미지의 강도 분포를 수정하여 특정 강도 범위를 강조하거나 억제

```
outImage[i][j] = 255.0 * pow(((double)inImage[i][j] / 127.0 - 1.0), 2.0); // CAP 파라볼라 : 밝은 곳이 입체적으로 보임
outImage[i][j] = 255.0 - 255.0 * pow(((double)inImage[i][j] / 127.0 - 1.0), 2.0); // CUP 파라볼라 : 어두운 곳이 입체적으로 보임
```

CAP

원본

CUP

## 6. 포스터라이징 :영상에서 화소에 있는 명암 값의 범위를 경계 값으로 축소

원본

포스터라이징

```
if (inImage[i][j] >= 0 && inImage[i][j] <= 31)
    outImage[i][j] = 31;
else if (inImage[i][j] > 31 && inImage[i][j] <= 63)
    outImage[i][j] = 63;
else if (inImage[i][j] > 63 && inImage[i][j] <= 95)
    outImage[i][j] = 95;
else if (inImage[i][j] > 95 && inImage[i][j] <= 127)
    outImage[i][j] = 127;
else if (inImage[i][j] > 127 && inImage[i][j] <= 159)
    outImage[i][j] = 159;
else if (inImage[i][j] > 159 && inImage[i][j] <= 191)
    outImage[i][j] = 191;
else if (inImage[i][j] > 191 && inImage[i][j] <= 223)
    outImage[i][j] = 223;
else if (inImage[i][j] > 223 && inImage[i][j] <= 255)
    outImage[i][j] = 255;
```

# 기능 설명
## (히스토그램 처리)

# 히스토그램 처리 :이미지의 픽셀 강도 분포를 분석하고 수정

## 1. 히스토그램 스트레칭 :낮은 명암 대비를 보이는 영상의 화질을 향상

원본

히스토그램 스트레칭

```
// End-In 탐색 : 최대 최소값 사이를 좁혀 스트레칭 효과를 극대화
// new = (old - low) / (high - low) * 255
old = inImage[i][j];
new = (int)((old - low) / (double)(high - low) * 255.0);
if (new > 255)
    new = 255;
if (new < 0)
    new = 0;
outImage[i][j] = new;
```

## 2. 히스토그램 평활화 :영상의 밝기 분포를 재분배하여 명암 대비를 최대화

히스토그램 평활화

원본

```
// 1단계 : 빈도수 세기(=히스토그램) histo[256]
histo[inImage[i][j]]++;
// 2단계 : 누적히스토그램 생성
sumHisto[i] = sumHisto[i - 1] + histo[i];
// 3단계 : 정규화된 히스토그램 생성
normalHisto[i] = sumHisto[i] * (1.0 / (inH * inW)) * 255.0;
// 4단계 : inImage를 정규화된 값으로 치환
outImage[i][j] = (unsigned char)normalHisto[inImage[i][j]];
```

# 기능 설명
## (기하학적 변환)

# 기하학적 변환 :영상을 구성하는 화소의 공간적 위치를 재배치

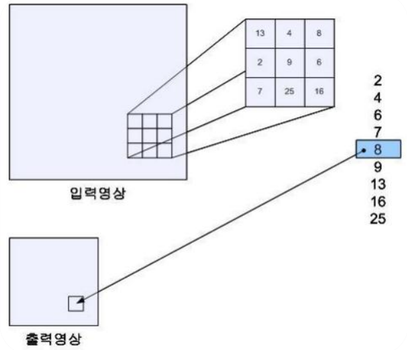
## 1. 크기 조절 :영상의 모양은 변화시키지 않은 채 크기만을 확대하거나 축소


2배 축소

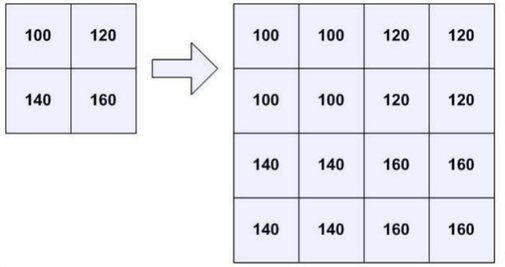

원본


2배 확대

```
// 히스토그램 계산
histo[inImage[k][q]] += 1;
count += histo[a];
// 중간 값 계산
if (count >= (scale * scale) / 2) {
    median = a;
    break;
}
outImage[i][j] = median;
```




**축소** : 미디언 표현




**확대** : 인접 픽셀 보간법 (백워딩)

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

## 2. 이동 :영상의 모양은 변화시키지 않은 채 위치만을 이동


-100


원본


+50

```
// 우측 하단
if ((0 <= i && i < outH) && (0 <= j && j < outW))
    outImage[i + move][j + move] = inImage[i][j];
```

```
// 좌측 상단
if ((0 - move <= i && i < outH) && (0 - move <= j && j < outW))
    outImage[i + move][j + move] = inImage[i][j];
```

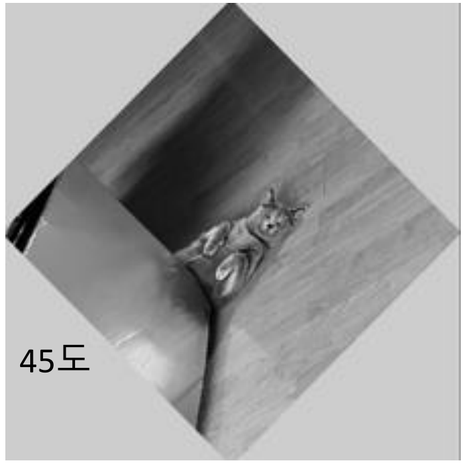
# 기하학적 변환 :영상을 구성하는 화소의 공간적 위치를 재배치

$$W' = H\cos(90-\theta) + \cos\theta$$
$$H' = H\cos\theta + W\cos(90-\theta)$$

## 3. 회전 :영상을 임의의 방향으로 특정한 각도만큼 회전


원본


90도


45도

```
double tmp_radian = degree % 90 * PI / 180.0;
double tmp_radian90 = (90 - degree % 90) * PI / 180.0;
freeOutputMemory();
// 회전 각도에 따라 출력 영상을 확대
outH = (int)(inH * cos(tmp_radian90) + inW * cos(tmp_radian));
outW = (int)(inW * cos(tmp_radian) + inH * cos(tmp_radian90));
double angle = degree * PI / 180.0;
```

```
// 출력 크기의 임시 영상, 중앙에 (0,0) 오도록 이동
tmpImage[i + dx][j + dy] = inImage[i][j];
```

```
// 백워딩 + 중앙으로 이동
xs = (int)(cos(angle) * (xd - cx) + sin(angle) * (yd - cy)) + cx;
ys = (int)(-sin(angle) * (xd - cx) + cos(angle) * (yd - cy)) + cy;
if ((0 <= xs && xs < outH) && (0 <= ys && ys < outW))
    outImage[xd][yd] = tmpImage[xs][ys];
```

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

## 4. 미러링 :영상을 가로축이나 세로축으로 단순히 뒤집는 것


상/하


원본


좌/우

```
// 좌-우
outImage[i][-(j - outW + 1)] = inImage[i][j];
```

```
// 상-하
outImage[-(i - outH + 1)][j] = inImage[i][j];
```

# 기능 설명
## (영역 변환)

# 영역 변환 :해당 입력 화소뿐만 아니라 그 주위의 화소 값도 함께 고려하는 공간 영역 연산

## 1. 엠보싱 :입력 영상을 양각 형태로 보이게 하는 기술

원본

3x3 마스크

5x5 마스크

```
// 엠보싱 마스크 생성
if (i == 0 && j == 0)
    mask[i][j] = -1.0;
else if (i == size - 1 && j == size - 1)
    mask[i][j] = 1.0;
else
    mask[i][j] = 0.0;
```

| | | |
|---|---|---|
| -1 | 0 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 1 |

## 2. 블러링 :영상의 세밀한 부분을 제거하여 영상을 흐리게 하거나 부드럽게 하는 기술

원본

3x3 마스크

5x5 마스크

```
// 블러링 마스크 생성
mask[i][j] = 1 / (double)(size * size);
```

| | | |
|---|---|---|
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |
| 1/9 | 1/9 | 1/9 |

# 영역 변환 :해당 입력 화소뿐만 아니라 그 주위의 화소 값도 함께 고려하는 공간 영역 연산

## 3. 스무딩 :영상의 세세한 부분을 제거하여 부드럽게 하므로 스무딩(Smoothing) 처리라고도 함

```
// 가우시안 마스크 생성
double x = sqrt((pow((i - center), 2)+ pow((j - center), 2)));
gaussian = exp(-(x * x) / (2.0 * sigma * sigma)) / (sigma * sqrt(2.0 * PI));
mask[i][j] = gaussian;
```

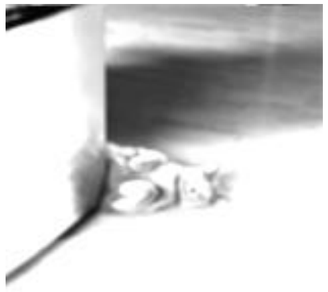
| | | |
|---|---|---|
| 1/16 | 1/8 | 1/16 |
| 1/8 | 1/4 | 1/8 |
| 1/16 | 1/8 | 1/16 |

원본

3x3 마스크
σ = 3

3x3 마스크
σ = 5

5x5 마스크
σ = 0.5



5x5 마스크
σ = 4

## 4. 샤프닝 :블러링과는 반대되는 효과, 고주파에 해당하는 상세한 부분을 더욱 강조하여 대비 효과를 증가

원본

3x3 마스크

5x5 마스크

mask2

| | | |
|---|---|---|
| 0 | -1 | 0 |
| -1 | 5 | -1 |
| 0 | -1 | 0 |

mask3

| | | |
|---|---|---|
| -1 | -1 | -1 |
| -1 | 9 | -1 |
| -1 | -1 | -1 |

```
// 샤프닝 마스크 생성
if (i == center && j == center)
    mask[i][j] = (double)size * size;
else
    mask[i][j] = -1.0;
```

# 영역 변환 :해당 입력 화소뿐만 아니라 그 주위의 화소 값도 함께 고려하는 공간 영역 연산

- **영역 변환에서의 마스크 회선 연산**

```
// 임시 이미지 할당(실수) : 패딩을 위한
double** tmpInImage = mallocDoubleMemory(inH + (size - 1), inW + (size - 1)); // 필터 크기에 따라 패딩 변화
double** tmpOutImage = mallocDoubleMemory(outH, outW);
```

↓

```
tmpInImage[i + 1][j + 1] = inImage[i][j];
```

↓

```
// 컨볼루션 연산
double sum;
for (int i = 0; i < inH; i++) {
    for (int j = 0; j < inW; j++) {
        sum = 0;
        for (int k = 0; k < size; k++) {
            for (int q = 0; q < size; q++) {
                sum += tmpInImage[i + k][j + q] * mask[k][q];
            }
        }
        tmpOutImage[i][j] = sum;
    }
```

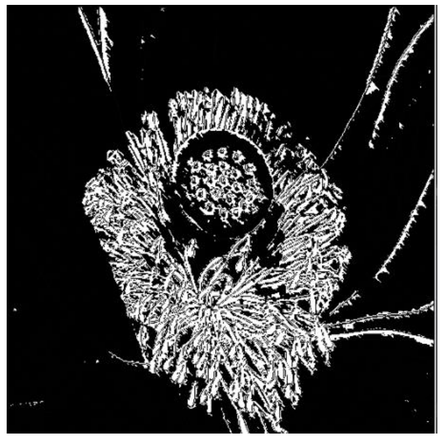
→

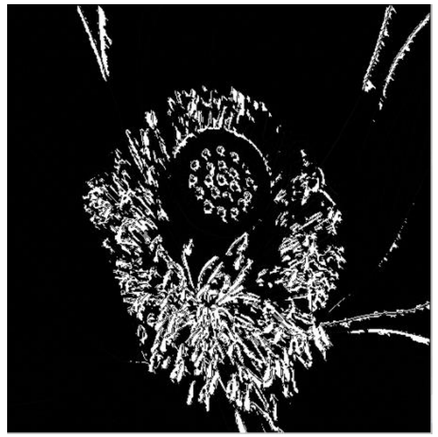
```
//임시 출력 영상 -> 출력 영상
if (tmpOutImage[i][j] < 0.0)
    outImage[i][j] = 0;
else if (tmpOutImage[i][j] > 255.0)
    outImage[i][j] = 255;
else
    outImage[i][j] = (unsigned char)tmpOutImage[i][j];
```

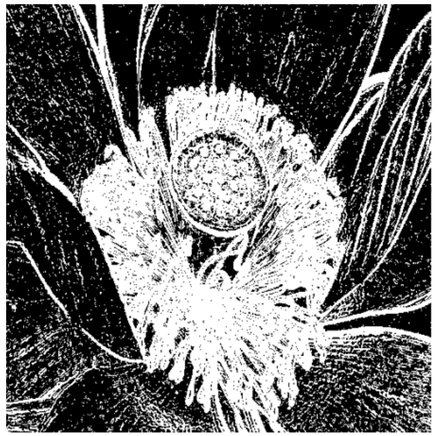
# 영역 변환 :해당 입력 화소뿐만 아니라 그 주위의 화소 값도 함께 고려하는 공간 영역 연산

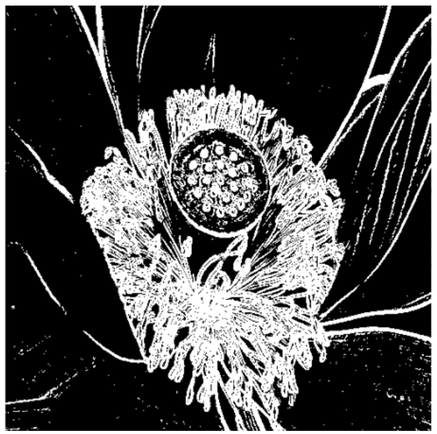
## 5. 에지 검출(LOG) :잡음에 민감한 라플라시안 마스크의 문제점 해결을 위해, 가우시안 스무딩 후 라플라시안 사용


원본


t = 10 (1)


t = 20 (1)


t = 10 (2)


t = 20 (2)

### 1) LOG 마스크 이용

| | | | | |
|---|---|---|---|---|
| 0 | 0 | −1 | 0 | 0 |
| 0 | −1 | −2 | −1 | 0 |
| −1 | −2 | 16 | −2 | −1 |
| 0 | −1 | −2 | −1 | 0 |
| 0 | 0 | −1 | 0 | 0 |

### 2) 직접 계산

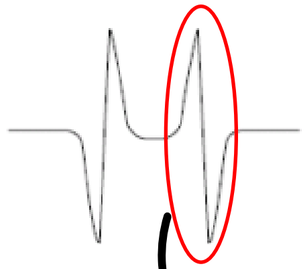
(1) →

| | | |
|---|---|---|
| 1/16 | 1/8 | 1/16 |
| 1/8 | 1/4 | 1/8 |
| 1/16 | 1/8 | 1/16 |

가우시안 마스크 →

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

라플라시안 마스크 →



**Zero-Crossing**
라플라시안 연산을 통해 얻은 영상에서 픽셀 값의 부호 변화를 검출하여 경계(에지)를 찾는 과정

(2) →

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

소벨 행 마스크

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

소벨 열 마스크 →

$$\sqrt{f_x^{\,2} + f_y^{\,2}}$$

에지 강도 계산 → 이진화

# 영역 변환 :해당 입력 화소뿐만 아니라 그 주위의 화소 값도 함께 고려하는 공간 영역 연산

## 5. 에지 검출(DOG) :계산량이 많은 LOG의 단점 극복을 위해 서로 다른 가우시안 마스크의 차를 이용

원본

DOG 마스크 이용

### 1) LOG 마스크 이용

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | −1 | −1 | −1 | 0 | 0 | 0 |
| 0 | −2 | −3 | −3 | −3 | −3 | −3 | −2 | 0 |
| 0 | −3 | −2 | −1 | −1 | −1 | −2 | −3 | 0 |
| −1 | −3 | −1 | 9 | 9 | 9 | −1 | −3 | −1 |
| −1 | −3 | −1 | 9 | 19 | 9 | −1 | −3 | −1 |
| −1 | −3 | −1 | 9 | 9 | 9 | −1 | −3 | −1 |
| 0 | −3 | −2 | −1 | −1 | −1 | −2 | −3 | 0 |
| 0 | −2 | −3 | −3 | −3 | −3 | −3 | −2 | 0 |
| 0 | 0 | 0 | −1 | −1 | −1 | 0 | 0 | 0 |

### 2) 직접 계산

| | | |
|---|---|---|
| 1/16 | 1/8 | 1/16 |
| 1/8 | 1/4 | 1/8 |
| 1/16 | 1/8 | 1/16 |

| | | |
|---|---|---|
| 1/16 | 1/8 | 1/16 |
| 1/8 | 1/4 | 1/8 |
| 1/16 | 1/8 | 1/16 |

$$DoG(x,y) = \frac{e^{\frac{-(x^2+y^2)}{2\sigma_1^2}}}{2\pi\sigma_1^2} - \frac{e^{\frac{-(x^2+y^2)}{2\sigma_2^2}}}{2\pi\sigma_2^2}$$

## 느낀점

처음으로 c로 영상처리를 구현했는데, 이 과정에서 어려움을 겪었지만 성공적으로 마무리하여 만족스러웠습니다.

또한, c언어와 메모리 관리에 대한 이해가 높아진 것을 느꼈습니다.

## 한계점

코드의 가독성을 높이고 계산량을 최적화하기 위해 변수명과 주석을 명확하게 작성하는 작업이 필요합니다.

## 향후 발전 방향

프로젝트에 포함되지 않은 알고리즘을 추가하여 영상처리 기능을 확장하고, 사용자 편의를 높이기 위한 기능을 추가합니다.

# 감사합니다

Github : https://github.com/suhwanjo