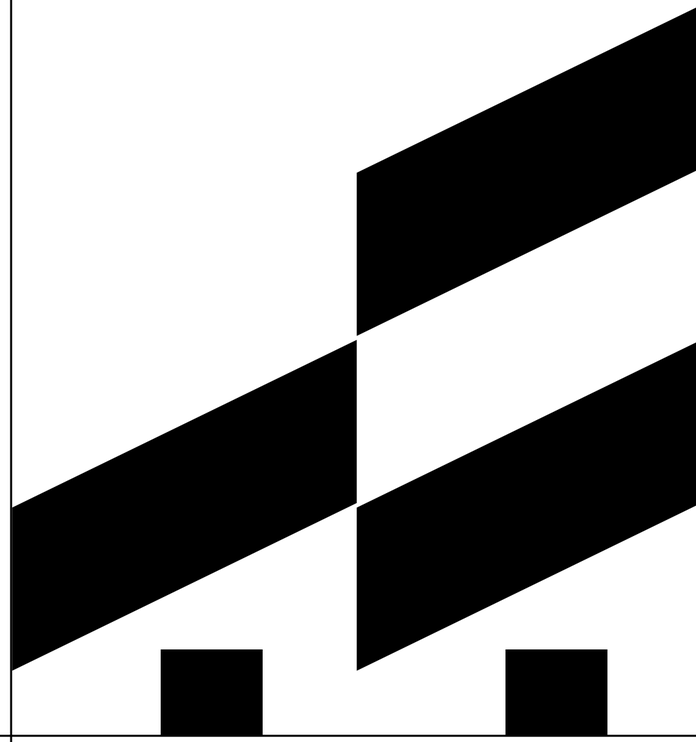
C언어 기반

GrayScale Image Processing



컴퓨터공학과 박유진

2024.07.25

목차

개요

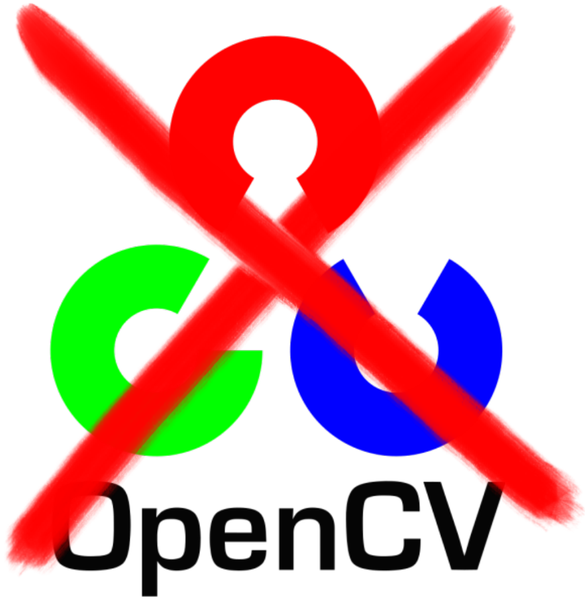
오픈 opencv를 사용하지 않고 직접 코드를 작성,
RAW이미지 파일 로드하며,
영상처리 알고리즘을 이용한 이미지 처리 프로그램

화소점 처리, 화소 영역 처리, 기하학적 처리,
히스토그램 처리 총 4가지 알고리즘 적용

환경

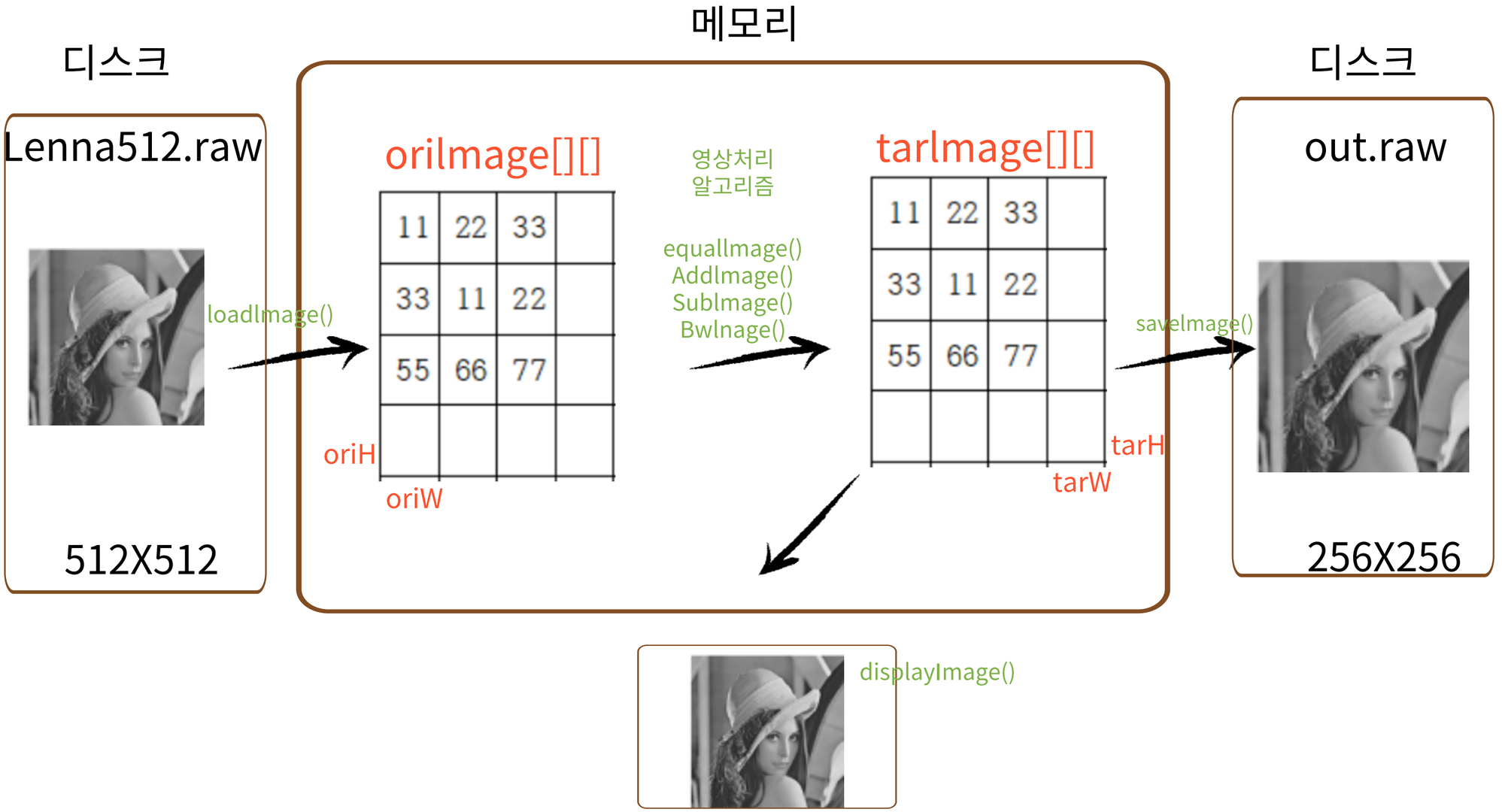
사용 언어 : C언어

사용 틀 : Visual Studio



OpenCV 사용 안함
직접 코드를 작성함

프로그램 구조도


메모리
디스크
Lenna512.raw
512X512
loadImage()
orilmage[][]
11 22 33
33 11 22
55 66 77
oriH
oriW
영상처리
알고리즘
equallmage()
Addlmage()
Sublmage()
Bwlnage()
tarlmage[][]
11 22 33
33 11 22
55 66 77
tarH
tarW
savelmage()
디스크
out.raw
256X256
displayImage()

구현 기능

밝기 조절

원본

+100

-100

```
printf("더할 값(-255~+255) :");
scanf("%d", &value);
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
int pixel = orilmage[i][k] + value;
if (pixel > 255)
pixel = 255;
if (pixel < 0)
pixel = 0;
tarImage[i][k] = (unsigned char)pixel;
}
}
```

코드 및 설명

+,-를 이용하여 이미지 밝기 조절

구현 기능

흑백 반전

원본

반전

```
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
tarImage[i][k] = 255 - oriImage[i][k];
}
}
```

코드 및 설명

최대값 - 원래화소값

구현 기능

축소

원본

2배

```
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
tarImage[i / scale][k / scale] =
oriImage[i][k];
}
}
```

구현 기능

확대(포워딩)

원본

2배

```
// *** 진짜 영상처리 알고리즘 ****
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
tarImage[i * scale][k * scale] =
oriImage[i][k];
}
}
```

구현 기능

확대(백워딩)

원본

2배

```
for (int i = 0; i < tarH; i++) {
for (int k = 0; k < tarW; k++) {
tarImage[i][k] = oriImage[i / scale][k /
scale];
}
}
```

구현 기능

회전

원본

45도

```
for (int i = 0; i < tarH; i++) {
for (int k = 0; k < tarW; k++) {
tarX = i;
tarY = k;
oriX = cos(radian) * (tarX - Cx) +
sin(radian) * (tarY - Cy) + Cx;
oriY = -sin(radian) * (tarX - Cx) +
cos(radian) * (tarY - Cy) + Cy;
if ((0 <= oriX && oriX < oriH) && (0 <=
oriY && oriY < oriW))
tarImage[tarX][tarY] = oriImage[oriX]
[oriY];}}
```

구현 기능

히스토그램 스트래

원본

After

```
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
tarImage[i][k]  =  ((double)oriImage[i]
[k] - low) / (high - low) * 255.0;
}
}
```

코드 및 설명

스트레치 :
이미지의 화소 분포를 고르게함

구현 기능

엔드-인

원본

After

```
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
double newVal = ((double)oriImage[i]
[k] - low) / (high - low) * 255;
if (newVal < 0.0)
newVal = 0;
if (newVal > 255.0)
newVal = 255;
tarImage[i][k] = newVal;
}
}
```

코드 및 설명

엔드-인 :
이미지 화소분포 고르게함
최소값과 최대값에 일정갑 + -
스트레치하는 방법

구현 기능

평활화

원본

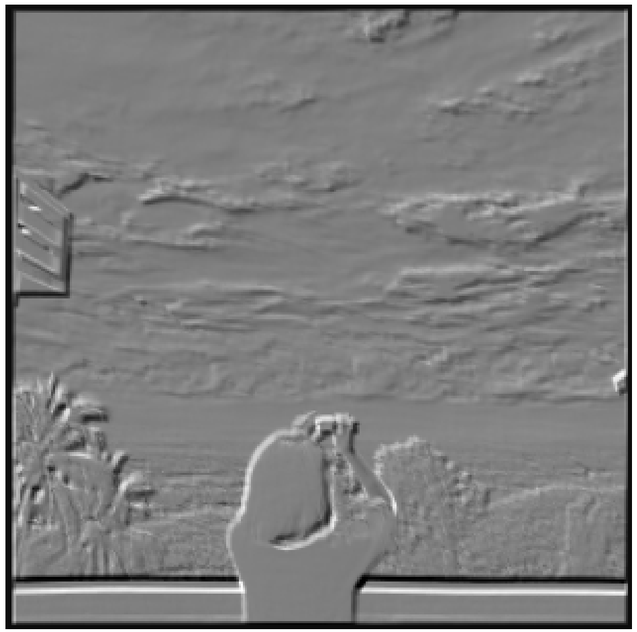
After

```
// 1단계 : 빈도수 히스토그램 생성
int hist[256] = { 0, };
for (int i = 0; i < oriH; i++)
for (int k = 0; k < oriW; k++)
hist[orilmage[i][k]]++;
// 2단계 : 누적 히스토그램 생성
int sumHist[256] = { 0, };
sumHist[0] = hist[0];
for (int i = 1; i < 256; i++)
sumHist[i] = sumHist[i - 1] + hist[i];
// 3단계 : 정규화된 히스토그램 생성
double normalHist[256] = { 0.0, };
for (int i = 0; i < 256; i++)
normalHist[i] = sumHist[i] * (1.0 /
(oriH * oriW)) * 255.0;
// 4단계 : 원래값을 정규화된 값으로 치환
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
tarImage[i][k] = (unsigned
char)normalHist[orilmage[i][k]];}}
```

구현 기능

엠보싱

원본



After

```
double mask[MSIZE][MSIZE] = {
{ 1.0, 0.0, 0.0 },
{ 0.0, 0.0, 0.0 },
{ 0.0, 0.0, -1.0 } };

// 회선 연산 → 마스크로 긁어가면서 계산하기
double S = 0; // 각점에 대한 마스크 연산 합계
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
S = 0.0; // 누적 초기화
for (int m = 0; m < MSIZE; m++)
for (int n = 0; n < MSIZE; n++)
S += mask[m][n] * tmpOriImage[i + m][k + n];
tmpTarImage[i][k] = S;}}
```

구현 기능

블러링

원본

After

```
double mask[MSIZE][MSIZE] = {
{ 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 } };
// 회선 연산 → 마스크로 긁어가면서 계산하기
double S = 0; // 각점에 대한 마스크 연산 합계
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
S = 0.0; // 누적 초기화
for (int m = 0; m < MSIZE; m++)
for (int n = 0; n < MSIZE; n++)
S += mask[m][n] * tmpOriImage[i + m][k + n];
tmpTarImage[i][k] = S;}}
```

구현 기능

수직에지

원본

After

```
double mask[MSIZE][MSIZE] = {
{ 0.0, 0.0, 0.0 },
{ -1.0, 1.0, 0.0 },
{ 0.0, 0.0, 0.0 } };
// 회선 연산 → 마스크로 긁어가면서 계산하기
double S = 0; // 각점에 대한 마스크 연산 합계
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
S = 0.0; // 누적 초기화
for (int m = 0; m < MSIZE; m++)
for (int n = 0; n < MSIZE; n++)
S += mask[m][n] * tmpOriImage[i + m][k + n];
tmpTarImage[i][k] = S;}}
```

구현 기능

수평에지

원본

After

```
double mask[MSIZE][MSIZE] = {
{ 0.0, -1.0, 0.0 },
{ 0.0, 1.0, 0.0 },
{ 0.0, 0.0, 0.0 } };
// 회선 연산 → 마스크로 긁어가면서 계산하기
double S = 0; // 각점에 대한 마스크 연산 합계
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
S = 0.0; // 누적 초기화
for (int m = 0; m < MSIZE; m++)
for (int n = 0; n < MSIZE; n++)
S += mask[m][n] * tmpOriImage[i + m][k + n];
tmpTarImage[i][k] = S;}}
```

구현 기능

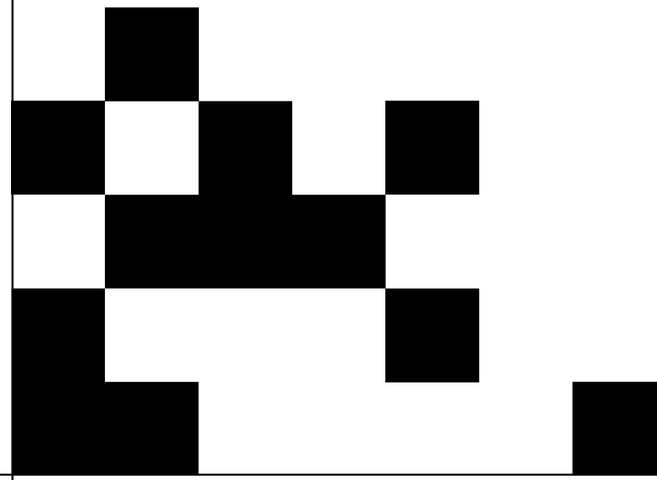
수직 + 수평 (임계값)

원본

임계값
= 20

```
double mask[MSIZE][MSIZE] = {
{ 0.0, -1.0, 0.0 },
{ -1.0, 2.0, 0.0 },
{ 0.0, 0.0, 0.0 } };
// 회선 연산 → 마스크로 긁어가면서 계산하기
double S = 0; // 각점에 대한 마스크 연산 합계
for (int i = 0; i < oriH; i++) {
for (int k = 0; k < oriW; k++) {
S = 0.0; // 누적 초기화
for (int m = 0; m < MSIZE; m++)
for (int n = 0; n < MSIZE; n++)
S += mask[m][n] * tmpOriImage[i + m][k + n];
tmpTarImage[i][k] = S;}}
```

감사합니다.



진행 기간 : 24.07.15 ~ 24.07.24