영상처리 HW3

16010811 김범수

Image_Filtering 함수

```
void Image_Filtering(Img_Buf* img)
{
    int Mask_size = 3;
    int Add_size = Mask_size - 1;
    UChar* Padding_buf;
    Padding_buf = (UChar*)calloc(Mask_size * Mask_size, sizeof(UChar));
    UChar* Focus_TMP;
    UChar* TMP;
    int Blur_count = 3;

    img->Embossing = (UChar*)calloc(img->info.width * img->info.height, sizeof(UChar));
    img->Median = (UChar*)calloc(img->info.width * img->info.height, sizeof(UChar));
    img->Homogeneity = (UChar*)calloc(img->info.width * img->info.height, sizeof(UChar));
    img->Blurring = (UChar*)calloc(img->info.width * img->info.height, sizeof(UChar));
    img->Derivative_1 = (UChar*)calloc(img->info.width * img->info.height, sizeof(UChar));
    img->Derivative_2 = (UChar*)calloc(img->info.width * img->info.height, sizeof(UChar));
    img->info.Ori_stride = img->info.width;
    img->info.Padding_stride = img->info.width + Add_size;

    Image_Padding(img, img->Ori_img, img->info.width, img->info.height,  Mask_size);
    //Image_Padding_0(img, img->Ori_img, img->info.width, img->info.height, Mask_size);
```

- Padding_buf와 다른 함수들의 수정을 통해 Array에서 Pointer 형식으로 변환하여 Mask_size값의 변화를 통해 모든 함수들을 조절 가능하게 코딩하였습니다.
- Add_size가 Mask_size / 2 + 1이었는데 Mask_size -1로 수정하였습니다.
- Blur_count는 Blurring 횟수를 나타내는 변수입니다.
- 대부분의 과정은 포인터를 이용하여 진행하였습니다.

Image_Filtering 함수

```
for (int i = 0; i < img->info.height; i++)
{
    for (int j = 0; j < img->info.width; j++)
    {
        Focus_TMP = img->padding + (i * img->info.Padding_stride + j);
        for (int k = 0; k < Mask_size; k++)
            for (int l = 0; l < Mask_size; l++)
            {
                TMP = Focus_TMP + (k * img->info.Padding_stride + l);
                *(Padding_buf + k * Mask_size + l) = TMP[0];
            }
        img->Embossing[i * img->info.Ori_stride + j] = Embossing(Padding_buf, Mask_size);
        img->Median[i * img->info.Ori_stride + j] = Median(Padding_buf, Mask_size);
        img->Homogeneity[i * img->info.Ori_stride + j] = Homogeneity(Padding_buf, Mask_size);
        img->Derivative_1[i * img->info.Ori_stride + j] = Derivative_1(Padding_buf, Mask_size);
        img->Derivative_2[i * img->info.Ori_stride + j] = Derivative_2(Padding_buf, Mask_size);
    }
}

for (int q = 0; q < Blur_count; q++) {
    for (int i = 0; i < img->info.height; i++)
    {
        for (int j = 0; j < img->info.width; j++)
        {
            Focus_TMP = img->padding + (i * img->info.Padding_stride + j);
            for (int k = 0; k < Mask_size; k++)
                for (int l = 0; l < Mask_size; l++)
                {
                    TMP = Focus_TMP + (k * img->info.Padding_stride + l);
                    *(Padding_buf + k * Mask_size + l) = TMP[0];
                }
            img->Blurring[i * img->info.Ori_stride + j] = Blurring(Padding_buf, Mask_size);
        }
    }
```

- Focus_TMP = 좌측 상단 좌표
- Padding_buf에 포인터를 이용하여 화소값을 넣어줍니다

- Blurring과정에서 Blurring과 Padding 과정의 반복을 Blur_count만큼 진행합니다.

Embossing

```
UChar Embossing(UChar* buf, int Mask_size)
{
    int Mask_Coeff[] = { -1, 0, 0, 0, 0, 0, 0, 0, 1 };
    int Convolution_All_coeff = 0;

    for (int i = 0; i < Mask_size * Mask_size; i++) Convolution_All_coeff += (Mask_Coeff[i] * (int)*(buf + i));

    Convolution_All_coeff = Convolution_All_coeff < 0 ? Min_Pix : Convolution_All_coeff;

    return (UChar)Convolution_All_coeff = Convolution_All_coeff + 128 > Max_Pix ? Max_Pix : Convolution_All_coeff + 128;
}
```

- Mask를 이용하여 컨볼루션이 진행되고 컨볼루션 진행 후 음수에 대해서 Clipping을 진행합니다

- 이후 128을 더한 후 Max_Pix에 대해서 Clipping을 진행 후 Unsigned Char로 형변환 후 값을 return 합니다.

Blurring

```
UChar Blurring(UChar* buf, int Mask_size)
{
    double Mask_Coeff[] = { 1.0 / 9.0, 1.0 / 9.0, 1.0 / 9.0, 1.0 / 9.0, 1.0 / 9.0, 1.0 / 9.0, 1.0 / 9.0, 1.0 / 9.0, 1.0 / 9.0 };
    double Convolution_All_coeff = 0;

    for (int i = 0; i < Mask_size * Mask_size; i++)
        Convolution_All_coeff += (Mask_Coeff[i] * (double)*(buf + i));

    return Convolution_All_coeff = Convolution_All_coeff > Max_Pix ? Max_Pix : Convolution_All_coeff < Min_Pix ? Min_Pix : Convolution_All_coeff;
}
```

- 컨볼루션 진행 시 Mask의 가중치들이 double형이므로 버퍼도 double으로 형변환 후 진행합니다.

- Max_pix, Min_pix에 대한 Clipping이후 Unsigned char형으로 return합니다.

Median Filtering

```
UChar Median(UChar* buf, int Mask_size)
{
    UChar* Asc_buf;
    Asc_buf = (UChar*)calloc(Mask_size * Mask_size, sizeof(UChar));
    int i, j = 0;
    UChar temp;

    for (i = 0; i < Mask_size * Mask_size; i++) {
        *(Asc_buf + i) = *(buf + i);
    }

    for (i = 0; i < Mask_size * Mask_size - 1; i++) {
        for (j = 0; j < Mask_size * Mask_size - 1 - i; j++) {
            if (*(Asc_buf + j) > *(Asc_buf + j + 1)) {
                temp = *(Asc_buf + j);
                *(Asc_buf + j) = *(Asc_buf + j + 1);
                *(Asc_buf + j + 1) = temp;
            }
        }
    }
    return *(Asc_buf + Mask_size * Mask_size / 2);
}
```

- 오름차순으로 정렬할 버퍼인 Asc_buf에 메모리 할당을 진행합니다.
- Asc_buf에 버퍼 값을 할당합니다.
- 픽셀 값들의 비교를 통하여 오름차순으로 정렬합니다
- 이후 중앙의 위치하는 픽셀 값을 return 합니다.

유사 연산자

```
UChar Homogeneity(UChar* buf, int Mask_size)
{
    UChar* temp;
    UChar max;
    temp = (UChar*)calloc(Mask_size * Mask_size, sizeof(UChar));

    for (int i = 0; i < Mask_size * Mask_size; i++) {
        *(temp + i) = abs(*(buf + i) - *(buf + Mask_size * Mask_size / 2 - 1));

    }
    max = *(temp);
    for (int i = 0; i < Mask_size * Mask_size; i++) {
        if (*(temp + i) > max) {
            max = *(temp + i);
        }
    }

    return (UChar)max = max + 60 > Max_Pix ? Max_Pix : max + 60;
}
```

- Mask_size x Mask_size의 화소들의 가운데 화소를 빼고 절댓값을 취합니다.

- 이후 최대값을 찾습니다.

- 최댓값에 60을 더한 후 Max_pix값에 대한 Clipping 후 Unsigned char형으로 return합니다

1차 미분 연산자

```
UChar Derivative_1(UChar* buf, int Mask_size)
{
    int Mask_Sobel_Row[] = { 1, 0, -1, 2, 0, -2, 1, 0, -1};
    int Mask_Sobel_Col[] = { -1, -2, -1, 0, 0, 0, 1, 2, 1};
    int Convolution_Row_coeff = 0;
    int Convolution_Col_coeff = 0;
    int Convolution_All_coeff = 0;

    for (int i = 0; i < Mask_size * Mask_size; i++) {
        Convolution_Row_coeff += (Mask_Sobel_Row[i] * (int) *(buf + i));
        Convolution_Col_coeff += (Mask_Sobel_Col[i] * (int) *(buf + i));
    }

    Convolution_Row_coeff = Convolution_Row_coeff > Max_Pix ? Max_Pix : Convolution_Row_coeff < Min_Pix ? Min_Pix : Convolution_Row_coeff;
    Convolution_Col_coeff = Convolution_Col_coeff > Max_Pix ? Max_Pix : Convolution_Col_coeff < Min_Pix ? Min_Pix : Convolution_Col_coeff;
    Convolution_All_coeff = Convolution_Row_coeff + Convolution_Col_coeff;

    return (UChar)Convolution_All_coeff = Convolution_All_coeff > Max_Pix ? Max_Pix : Convolution_All_coeff;
}
```

- Row 마스크 x 원본, Col 마스크 x 원본을 진행합니다.
- 이후 Max_pix, Min_pix에 대해 Clipping 진행 후 합을 구합니다.
- 구해진 값에 Max_pix 값에 Clipping 진행 후 Unsigned char형으로 return합니다.

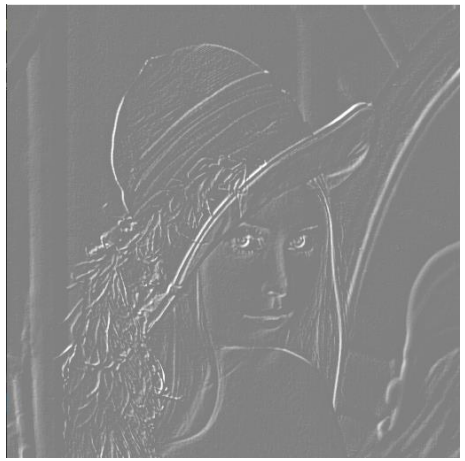
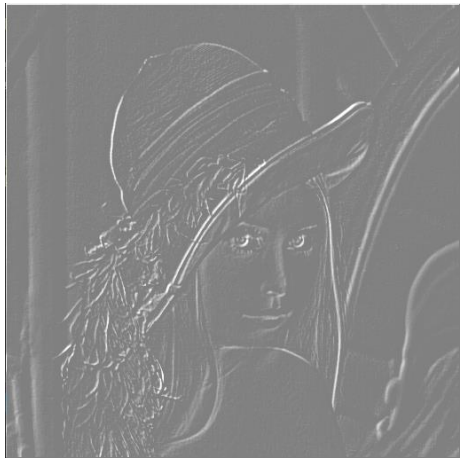
2차 미분 연산자

```
UChar Derivative_2(UChar* buf, int Mask_size)
{
    int Mask_DoG[] = { 0, 0, -1, -1, -1, 0, 0,
                           0, -2, -3, -3, -3, -2, 0,
                          -1, -3, 5, 5, 5, -3, -1,
                          -1, -3, 5, 16, 5, -3, -1,
                          -1, -3, 5, 5, 5, -3, -1,
                          0, -2, -3, -3, -3, -2, 0,
                          0, 0, -1, -1, -1, 0, 0 };
    int Convolution_All_coeff = 0;

    for (int i = 0; i < Mask_size * Mask_size; i++) {
        Convolution_All_coeff += (Mask_DoG[i] * (int)*(buf + i));
    }

    return (UChar)Convolution_All_coeff = Convolution_All_coeff > Max_Pix ? Max_Pix : Convolution_All_coeff < Min_Pix ? Min_Pix : Convolution_All_coeff;
}
```

- Image_filtering 함수에서 Mask_size를 7로 변경해야합니다.

- Max_pix, Min_pix에 대한 Clipping이후 Unsigned char형으로 return합니다.

결과

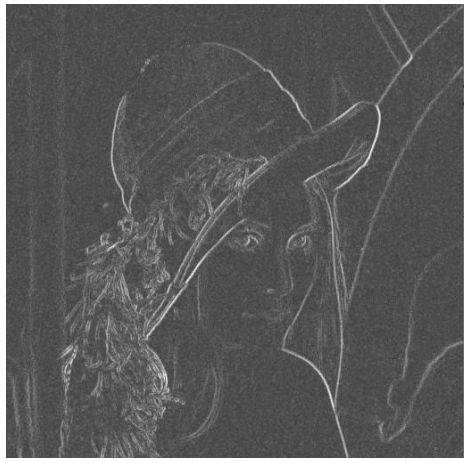
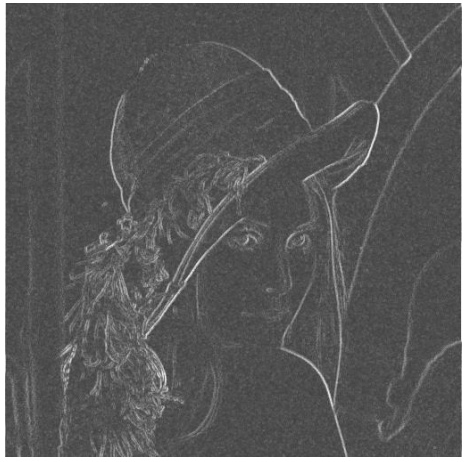
	Embossing	Blurring(10) Scratch	Blurring(3) Gaussian
copy			
zero			

- Embossing 을 통해 회색판에 Edge가 나타난 듯한 출력을 확인할 수 있었습니다.

- Blurring을 많이 진행할수록 이미지가 더 Smooth해짐을 확인할 수 있었습니다.

- Blurring쪽에서 zero padding의 Boundary는 검정, Embossing쪽에서는 흰색으로 확인되었습니다.(Embossing에서는 변화 큰 부분이 흰색(높은값)으로 표현되기 때문)

결과

	유사 연산	1차 미분 연산	2차 미분 연산
copy			
zero			

- 유사 연산에서는 1차, 2차 미분 연산에 비해 부드럽게 Edge가 검출되었습니다.
- 1차 미분은 점차적으로 변화되는 영역에도 반응하여 부드러운 Edge 검출되었습니다.
- 2차 미분은 점차적으로 변화 영역에는 반응하지 않고 1차 미분에 비해 날카로운 느낌으로 검출되었습니다.
- 변화가 큰 부분은 흰색으로 표현되는데 zero paddin의 경우 Boundary가 흰색으로 표현되는 부분이 확인되었습니다.