과제 3 보고서

지능기전공학부 스마트기기공학전공

18011789

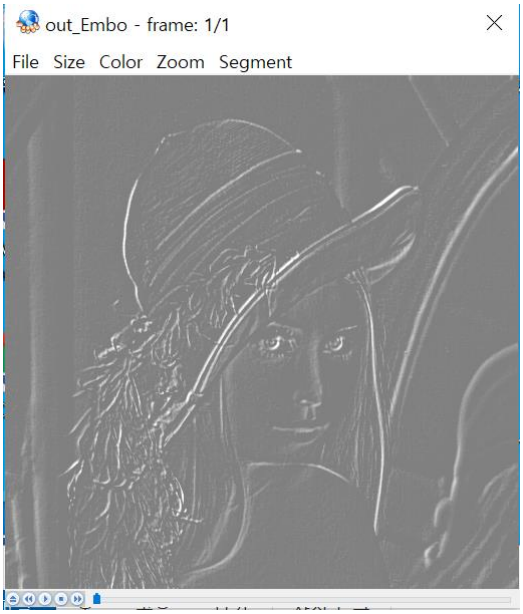
조혜수

목차

1. 결과 영상

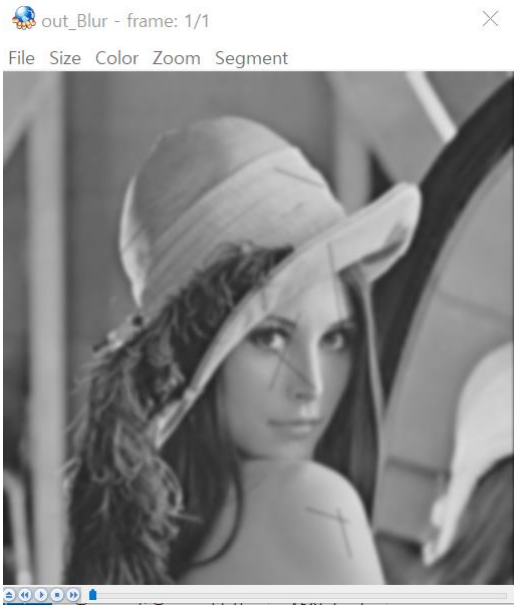
- Lena Original, Lena Scratch-Noise

- Embossing (확인용)



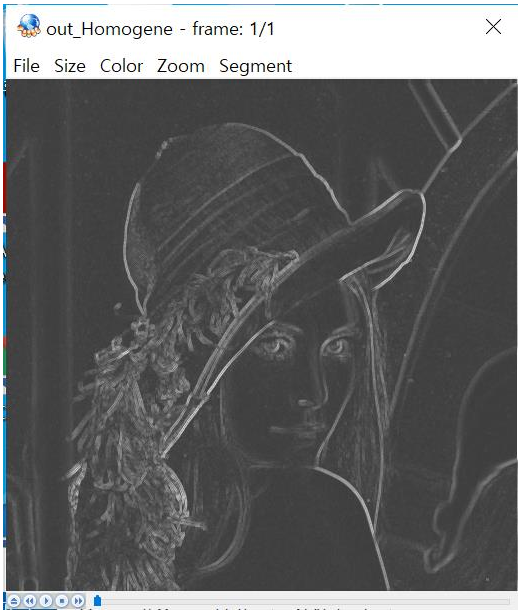
- Sharpning (확인용)

- Blurring

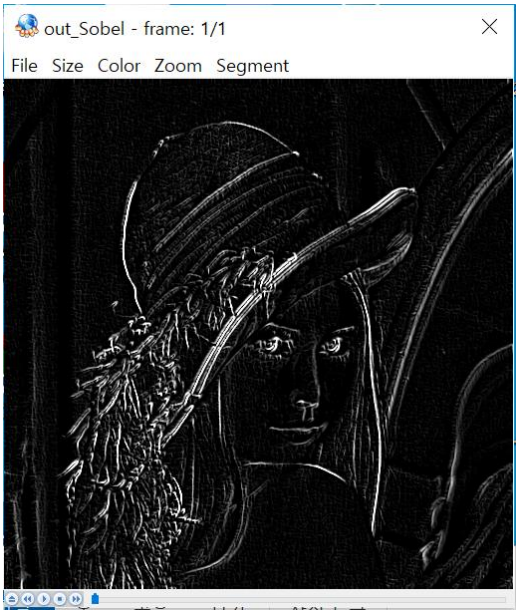


- Median

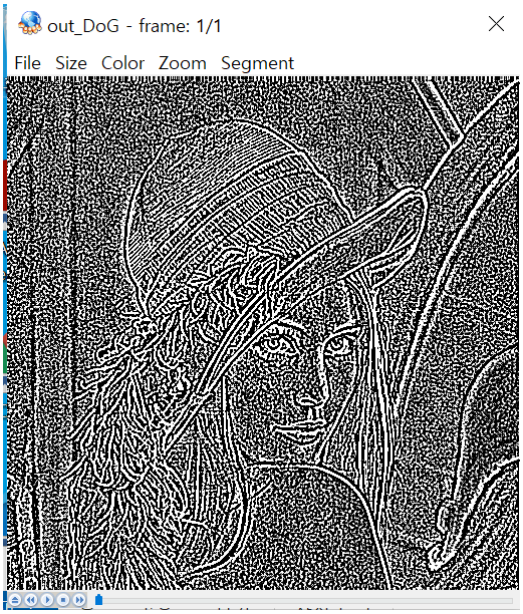
- Homogeneity



- Sobel


out_Sobel - frame: 1/1
File Size Color Zoom Segment

- DoG


out_DoG - frame: 1/1
File Size Color Zoom Segment

- LoG

2. 코드 분석

- Main.h

(분석 생략)

- Filter.h

```
#include "main.h"

#define CLIP(x) (x < minVal) ? minVal : x > maxVal ? maxVal : x   // 클리핑 과정
#define CLIPPIC_HOR(x) (x < 0) ? 0 : x >= WIDTH   ? WIDTH   - 1 : x
#define CLIPPIC_VER(x) (x < 0) ? 0 : x >= HEIGHT ? HEIGHT - 1 : x

#define MAX(x, y) x > y ? x : y // 큰 값 비교
#define SWAP(x, y) {int tmp; tmp = x; x = y; y = tmp;} // 두 값 바꾸기

typedef struct _Image_Buffer
{
        UChar* padding; // 패딩 영상 저장 버퍼

        UChar* Embossing;
        UChar* Sharpening;
        UChar* DoG;
        UChar* Median;
        UChar* Sobel;
        UChar* Blur;
        UChar* homogeneity;
        UChar* LoG; // LoG 추가

}Img_Buf;

void ImageFiltering(UChar* Data, Img_Buf* img);
void ImageFilteringBlur(UChar* Data, Img_Buf* img);
void ImagePadding(Img_Buf* img, UChar* Buf, int width, int height, int maskSize);
void ImageOutput(UChar* Data, Int wid, Int hei, Char String[]);

void OutputBlock(Img_Buf* img, UChar *paddingBlock, int maskSize, int i, int j, int PStride); // 각 함수들 선언 부분
```

- main.c

```
#include "Filter.h"

void main()
{
	FILE *fp;
	UChar *ori; //원본 영상 화소값들을 저장하기 위한 버퍼
	Img_Buf image;


	Int wid = WIDTH; Int hei = HEIGHT; // 영상 크기 기본 값 정의
	Int min = minVal; Int max = maxVal; // 화소 기본 값 정의

	Int picSize = wid * hei; //영상 사이즈


	////////////////////////////////////////////////////////////////////////////////////////////////////
	// Embossing, Sharpening, DoG, Sobel, Homogeneity
	fopen_s(&fp, "lena_512x512.raw", "rb"); //원본 영상 열기
	ori = (UChar*)malloc(sizeof(UChar) * picSize); //원본 영상 크기만큼 공간 선언
	memset(ori, 0, sizeof(UChar) * picSize); //0으로 초기화
	fread(ori, sizeof(UChar), picSize, fp); // 원본 영상 읽기(원본 영상의 픽셀 값을 배열 변수에
저장)

	ImageFiltering(ori, &image); // 원본영상( 스크래치 넣지 않은 영상 )을 인자로 가진 필터링
함수 호출

	free(ori);
	fclose(fp); //메모리 해제
	////////////////////////////////////////////////////////////////////////////////////////////////////

	////////////////////////////////////////////////////////////////////////////////////////////////////
	// Median, Blur
	fopen_s(&fp, "Lena-scratch-noise.raw", "rb"); //원본 영상 열기
	ori = (UChar*)malloc(sizeof(UChar) * picSize); //원본 영상 크기만큼 공간 선언
	memset(ori, 0, sizeof(UChar) * picSize); //0으로 초기화
	fread(ori, sizeof(UChar), picSize, fp); // 원본 영상 읽기(원본 영상의 픽셀 값을 배열 변수에
```

```
저장)

        ImageFilteringBlur(ori, &image); //스크래치 영상을 인자로 갖는 필터링 블러 함수 호출

        free(ori);
        fclose(fp);// 메모리 해제
        ////////////////////////////////////////////////////////////////////////////////////////////////////
}
```

- Filter.c

```
#include "Filter.h"

UChar EmbossingFunc(UChar* Data, int maskSize) // 엠보싱 계산 함수
{
        int Conv = 0; // 결과 값
        int Mask_Coeff[] =
        {   -1, 0, 0,
                0, 0, 0,
                0, 0, 1 }; // 마스크

        for (int i = 0; i < maskSize * maskSize; i++) // 픽셀과 마스크 값들을 곱해서 더함
                Conv += (Mask_Coeff[i] * (int)Data[i]);
        Conv = CLIP(Conv); // 클립핑

        return (UChar)CLIP(Conv + 128); // 음수값이 있으므로 +128을 해주고 클립핑 다시 해줘서
해당 픽셀의 최종 결과 값 return
}

UChar SharpeningFunc(UChar* Data, int maskSize) // 샤프닝 계산 함수
{
        int Conv = 0; //결과 값
        int Mask_Coeff[] =
        { -1, -1, -1,
        -1, 9, -1,
        -1, -1, -1 }; //마스크

        /* */
        for (int i = 0; i < maskSize * maskSize; i++) // 픽셀과 마스크 값들을 곱해서 더함
```

```
                Conv += (Mask_Coeff[i] * (int)Data[i]);

        return (UChar)CLIP(Conv); //결과 값을 클리핑해서 최종결과 반환
}

UChar DoGFunc(UChar* Data, int maskSize) //DoG 계산 함수
{
        int Conv = 0;//결과 값
        int Mask_Coeff[] =
        { 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 };// 7*7 마스크

        for (int i = 0; i < maskSize * maskSize; i++)// 픽셀과 마스크 값들을 곱해서 더함
                Conv += (Mask_Coeff[i] * (int)Data[i]);


        Conv = CLIP(Conv);//결과 값을 클리핑해서 최종결과 반환
        return (UChar)Conv;
}

UChar SobelFunc(UChar* Data, int maskSize) //Sobel 계산 함수
{
        int ConvVer = 0;
        int ConvHor = 0;
        int Mask_Coeff_Ver[] =
        { 1, 0, -1,
        2, 0, -2,
        1, 0, -1 }; // x축 방향 마스크

        int Mask_Coeff_Hor[] =
        { -1, -2, -1,
          0, 0, 0,
        1, 2, 1 }; //y축 방향 마스크

        /* */
```

```
	for (int i = 0; i < maskSize * maskSize; i++)
		ConvVer += (Mask_Coeff_Ver[i] * (int)Data[i]);

	for (int i = 0; i < maskSize * maskSize; i++)
		ConvHor += (Mask_Coeff_Hor[i] * (int)Data[i]);//수직축, 수평 축 따로 계산 함


	ConvVer = CLIP(ConvVer);
	ConvHor = CLIP(ConvHor);// 각각 클리핑 후

	return (UChar)CLIP(ConvVer + ConvHor); // 두 값 더해서 반환
}


UChar HomogeneityFunc(UChar* Data, int maskSize) //Homogeneity 계산 함수
{
	/* */
	int max = -1; // 최종 결과 값 ( 이후 max 값 모두 양수 이므로 -1로 초기화)
	int tmp = 0; // 각각 뺀 값들 비교할 때 쓸 변수

	int center = (int)Data[4]; // 센터 값 (항상 빼는 값)

	for (int i = 0; i < maskSize * maskSize; i++)
	{
		if (i == 4) continue; // 자기 자신과의 뺄셈은 건너 뜀
		tmp = (int)Data[i] - center; // 각 픽셀들 센터와 뺄셈 연산
		if (tmp < 0) tmp *= -1; // 절댓값
		if (max < tmp) max = tmp; // max 값 보다 크면 max 갱신
	}
	max = CLIP(max); // 클리핑

	return (UChar)CLIP(max + 60); //더 눈에 띄게 하기 위해 +60 해준 후 클리핑 해서 최종 결과
반환
}

UChar LoGFunc(UChar* Data, int maskSize) //LoG 계산 함수
{
	int Conv = 0;
	int Mask_Coeff[] =
	{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}; //5*5 마스크

		for (int i = 0; i < maskSize * maskSize; i++)
			Conv += (Mask_Coeff[i] * (int)Data[i]);// 픽셀과 마스크 값들을 곱해서 더함

		Conv = CLIP(Conv);
		return (UChar)Conv;// 클리핑 후 반환
}

void OutputBlock(Img_Buf* img, UChar *paddingBlock, int maskSize, int i, int j, int PStride) // input 영상에서 블록 떼기
{
		for (int k = 0; k < maskSize; k++) // 마스크 사이즈 만큼 떼어서 paddingBlock 에 저장
			for (int l = 0; l < maskSize; l++)
				paddingBlock[k * maskSize + l] = img->padding[(i + k) * PStride + (j + l)];
}

void ImageFiltering(UChar* Data, Img_Buf* img) //메인에서 호출하는 함수 : 필터링 호출 함수
{
		Int wid = WIDTH; Int hei = HEIGHT;
		Int min = minVal; Int max = maxVal;

		Int picSize = wid * hei; //영상 사이즈

		Int maskSize = 3;		//MxM size
		Int Add_size = (maskSize / 2) * 2;
		UChar *paddingBlock;

		Int Stride  = wid;
		Int PStride = wid + Add_size;

		Char String[8][10] = { "Embo", "Sharp", "Sobel", "Homogene", "Median", "DoG", "Blur","LoG"};

		img->Embossing	= (UChar*)calloc(picSize, sizeof(UChar));
		img->Sharpening	= (UChar*)calloc(picSize, sizeof(UChar));
		img->homogeneity	= (UChar*)calloc(picSize, sizeof(UChar)); //메모리 공간 초기화

		paddingBlock = (UChar*)calloc(maskSize * maskSize, sizeof(UChar)); // 영상에서 마스크 크기만큼 떼어오기 위한 공간
```

```
	ImagePadding(img, Data, wid, hei, maskSize); // 원본영상을 패딩 (Copy Padding)
	for (int i = 0; i < hei; i++)
		for (int j = 0; j < wid; j++)
		{
			OutputBlock(img, paddingBlock, maskSize, i, j, PStride); // 패딩된 영상에서 마스크 사이즈(3*3) 만큼 블록 가져오기
			img->Embossing   [i * wid + j] = EmbossingFunc   (paddingBlock, maskSize); // 가져온 블록(paddingBlock)에 대해 각각 연산
			img->Sharpening [i * wid + j] = SharpeningFunc (paddingBlock, maskSize);
			img->homogeneity[i * wid + j] = HomogeneityFunc(paddingBlock, maskSize);
		}
	free(img->padding);
	free(paddingBlock);


	////////////////////////////////////////////////////////////////////////////////////////////////////
	// Sobel 영상 만드는 부분, 입력영상은 Embossing이 적용된 영상

	img->Sobel =   (UChar*)calloc(picSize, sizeof(UChar)); //메모리 할당

	paddingBlock = (UChar*)calloc(maskSize * maskSize, sizeof(UChar)); // 영상에서 마스크 크기만큼 떼어오기 위한 공간
	ImagePadding(img, img->Embossing, wid, hei, maskSize); // Embossing 처리 된 영상을 패딩 (Copy Padding)

	for (int i = 0; i < hei; i++)
		for (int j = 0; j < wid; j++)
		{
			OutputBlock(img, paddingBlock, maskSize, i, j, PStride); // 패딩된 영상에서 마스크 사이즈(3*3)만큼 블록 가져오기
			img->Sobel[i * wid + j] = SobelFunc(paddingBlock, maskSize); // 가져온 블록(paddingBlock)에 대해 SobelFunc 연산함수 호출

		}
	free(img->padding);
	free(paddingBlock);


	////////////////////////////////////////////////////////////////////////////////////////////////////

	////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
// DoG 영상 만드는 부분, 입력영상은 Sharpening이 적용된 영상, Mask 크기는 7x7

img->DoG = (UChar*)calloc(picSize, sizeof(UChar)); // 메모리 할당

maskSize = 7; //DoG필터이므로 마스크 사이즈 3에서 7로 바꿔줌
Add_size = (maskSize / 2) * 2; // 폭에 더해주는 변수도 새로 갱신
PStride = wid + Add_size; //폭 갱신
paddingBlock = (UChar*)calloc(maskSize * maskSize, sizeof(UChar));// 영상에서 마스크
크기만큼 떼어오기 위한 공간
ImagePadding(img, img->Sharpening, wid, hei, maskSize);// Sharpening 처리 된 영상을 패딩
(Copy Padding)

for (int i = 0; i < hei; i++)
    for (int j = 0; j < wid; j++)
    {
        OutputBlock(img, paddingBlock, maskSize, i, j, PStride);   // 패딩된 영상에서
마스크 사이즈(7*7)만큼 블록 가져오기
        img->DoG[i * wid + j] = DoGFunc(paddingBlock, maskSize);// 가져온
블록(paddingBlock)에 대해 DoGFunc 연산함수

    }
free(img->padding);
free(paddingBlock);




//////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////


// LoG //
img->LoG = (UChar*)calloc(picSize, sizeof(UChar));

maskSize = 5; // 마스크 사이즈 5로 갱신
Add_size = (maskSize / 2) * 2;
PStride = wid + Add_size; //마스크 사이즈와 관련된 변수값 바꿔주기

paddingBlock = (UChar*)calloc(maskSize * maskSize, sizeof(UChar)); // 영상에서 마스크
크기만큼 떼어오기 위한 공간
ImagePadding(img, Data, wid, hei, maskSize); // 원본 영상 패딩
```

```
	for (int i = 0; i < hei; i++)
		for (int j = 0; j < wid; j++)
		{
			OutputBlock(img, paddingBlock, maskSize, i, j, PStride);// 패딩된 영상에서 마스크 사이즈(5*5)만큼 블록 가져오기
			img->LoG[i * wid + j] = LoGFunc(paddingBlock, maskSize); // 가져온 블록(paddingBlock)에 대해 LoGFunc 연산함수

		}
	free(img->padding);
	free(paddingBlock);


	ImageOutput(img->Embossing,     wid, hei, String[0]);//확인용
	ImageOutput(img->Sharpening, wid, hei, String[1]);//확인용
	ImageOutput(img->Sobel,          wid, hei, String[2]);
	ImageOutput(img->homogeneity, wid, hei, String[3]);
	ImageOutput(img->DoG,            wid, hei, String[5]);
	ImageOutput(img->LoG,                 wid, hei, String[7]);

	free(img->Embossing);
	free(img->Sharpening);
	free(img->Sobel);
	free(img->homogeneity);
	free(img->DoG);
	free(img->LoG);//공간 해제
}
```

- Blurring.c

```
#include "Filter.h"

UChar BlurFunc(UChar* Data, int maskSize) // 블러링 연산 함수
{
	double Conv = 0; //화소 결과 값
	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 };//마스크
```

```
	for (int i = 0; i < maskSize * maskSize; i++)
		Conv += (Mask_Coeff[i] * (double)Data[i]); // 각각 곱해서 더하기(평균내주기)

	return (UChar)CLIP(Conv);//클리핑해서 반환
}

UChar MedianFunc(UChar* Data, int maskSize) // 메디안 연산 함수
{

	int i, j;
	for (i = 0; i < maskSize * maskSize; i++)
	{
		for (j = i; j < maskSize * maskSize; j++)
		{

			if ((int)Data[i] > (int)Data[j]) // 앞의 수가 뒤의 수보다 더 크면
			{
				SWAP(Data[i], Data[j]);//두 값 바꾸기
			}
		}
	}// 오름차순


	return (int)Data[4];   // 가운데 값 반환
}

void ImageFilteringBlur(UChar* Data, Img_Buf* img) //메인에서 호출 : 스크래치 된 영상 필터링 하는 함수
{
	Int wid = WIDTH; Int hei = HEIGHT;
	Int min = minVal; Int max = maxVal;

	Int picSize = wid * hei; //영상 사이즈

	Int maskSize = 3;		//MxM size
	Int Add_size = (maskSize / 2) * 2;;
	UChar *paddingBlock;

	Int Stride = wid;
	Int PStride = wid + Add_size;
```

```
	Char String[8][10] = { "Embo", "Sharp", "Sobel", "Homogene", "Median", "DoG", "Blur","LoG" };

	img->Blur = (UChar*)calloc(picSize, sizeof(UChar)); //공간할당
	for (int cnt = 0; cnt < 6; cnt++) // 블러 6번 시행
	{
		paddingBlock = (UChar*)calloc(maskSize * maskSize, sizeof(UChar)); // 영상에서 마스크 사이즈만큼 가져올 공간

		if (cnt == 0) // 첫 시행에는 원본 영상 패딩
			ImagePadding(img, Data, wid, hei, maskSize);
		else // 그 이후에는 그 전 결과 영상 패딩
			ImagePadding(img, img->Blur, wid, hei, maskSize);

		for (int i = 0; i < hei; i++)
			for (int j = 0; j < wid; j++)
			{
				OutputBlock(img, paddingBlock, maskSize, i, j, PStride); // input 에서 3*3 블록 가져오기
				img->Blur[i * wid + j] = BlurFunc(paddingBlock, maskSize); //그 블록에 대해 블러링 계산해서 갱신
			}
		free(img->padding);
		free(paddingBlock);
	}


	////////////////////////////////////////////////////////////////////////////////////////////////////////////////
	// Median 영상 만드는 부분(10회 적용), 처음 입력영상은 원본 스크래치 레나, 2회부터는 Median이 적용된 영상


	img->Median = (UChar*)calloc(picSize, sizeof(UChar)); //공간 할당
	for (int cnt = 0; cnt < 10; cnt++) // 10번 필터링
	{
		paddingBlock = (UChar*)calloc(maskSize * maskSize, sizeof(UChar));  // 영상에서 마스크 사이즈만큼 가져올 공간

		if (cnt == 0) // 첫 시행 : 원본 패딩
			ImagePadding(img, Data, wid, hei, maskSize);
		else // 이 후 시행 : 그 전 결과 영상 패딩
```

```
                ImagePadding(img, img->Median, wid, hei, maskSize);

        for (int i = 0; i < hei; i++)
                for (int j = 0; j < wid; j++)
                {
                        OutputBlock(img, paddingBlock, maskSize, i, j, PStride); // input 에서
3*3 블록 가져오기
                        img->Median[i * wid + j] = MedianFunc(paddingBlock, maskSize); //
가져온 거에서 중앙 값 찾아서 갱신
                }
        free(img->padding);
        free(paddingBlock);
    }



    ////////////////////////////////////////////////////////////////////////////////////////////////////

    ImageOutput(img->Blur, wid, hei, String[6]);
    ImageOutput(img->Median, wid, hei, String[4]);

    free(img->Blur);
    free(img->Median);
}
```

- Padding.c

```
#include "Filter.h"

void ImagePadding(Img_Buf* img, UChar* Buf, int width, int height, int maskSize) // Copy Padding 함수
{
    int line, i, j;
    int Add_size = (maskSize / 2) * 2; // 상하, 좌우에 더해줄 값
    int Stride = width; // 원래 폭
    int PStride = width + Add_size; // 패딩 후 폭
    img->padding = (UChar*)calloc((width + Add_size) * (height + Add_size), sizeof(UChar)); //메모리
할당
```

```
	for (line = 0; line < (maskSize / 2); line++) //마스크 크기에 따라 패딩 크기( 반복문 횟수 ) 달라짐
	{
		//상하단 패딩
		for (i = 0; i < width; i++)
		{
			img->padding[line * PStride + (maskSize / 2 + i)] = Buf[i]; // 상단 패딩 : 첫 행 값 넣어주기
			img->padding[((height + (maskSize / 2)) + line) * PStride + (maskSize / 2 + i)] = Buf[(height - 1) * Stride + i]; // 하단 패딩 : 마지막 행 값 넣어주기
		}

		//좌우측 패딩
		for (i = 0; i < height; i++)
		{
			img->padding[(maskSize / 2 + i) * PStride + line] = Buf[i * Stride]; //좌측 패딩 : 첫 열 값 넣어주기
			img->padding[(maskSize / 2 + i) * PStride + width + line] = Buf[i * Stride + (width - 1)]; // 우측 패딩 : 마지막 열 값 넣어주기
		}
	}

	for (i = 0; i < (maskSize / 2); i++) //모서리 패딩해주기
	{
		for (j = 0; j < (maskSize / 2); j++)
		{
			/*** 좌상단 패딩 ***/
			img->padding[i * PStride + j] = Buf[0];

			/*** 우상단 패딩 ***/
			img->padding[i * PStride + ((maskSize / 2) + width + j)] = Buf[width - 1];

			/*** 좌하단 패딩 ***/
			img->padding[(height + (maskSize / 2) + i) * PStride + j] = Buf[(height - 1) * Stride];

			/*** 우하단 패딩 ***/
			img->padding[(height + (maskSize / 2) + i) * PStride + ((maskSize / 2) + width + j)] = Buf[(height - 1) * Stride + (width - 1)];
		}
	}
```

```
	/*** 원본 버퍼 불러오기 ***/
	for (i = 0; i < height; i++) // 패딩 부분 외의 원본 부분 넣어주기
	{
		for (j = 0; j < width; j++)
		{
			img->padding[(i + (maskSize / 2)) * PStride + (j + (maskSize / 2))] = Buf[i *
Stride + j];
		}
	}
}
```

- Imageout.c

```
#include "Filter.h"

void ImageOutput(UChar* Data, Int wid, Int hei, Char String[]) //결과 이미지 내보내기
{
	char Name_Hist[50] = "out_";
	char Name_extension[10] = ".raw";   //결과 이미지 파일이름 기본 설정

	FILE *fp;

	strcat_s(Name_Hist, 20, String); //해당하는 이름 붙이기
	strcat_s(Name_Hist, 20, Name_extension);


	fopen_s(&fp, Name_Hist, "wb"); //원본 영상 열기
	fwrite(Data, sizeof(UChar), wid * hei, fp); //필터 결과
	fclose(fp);
}
```