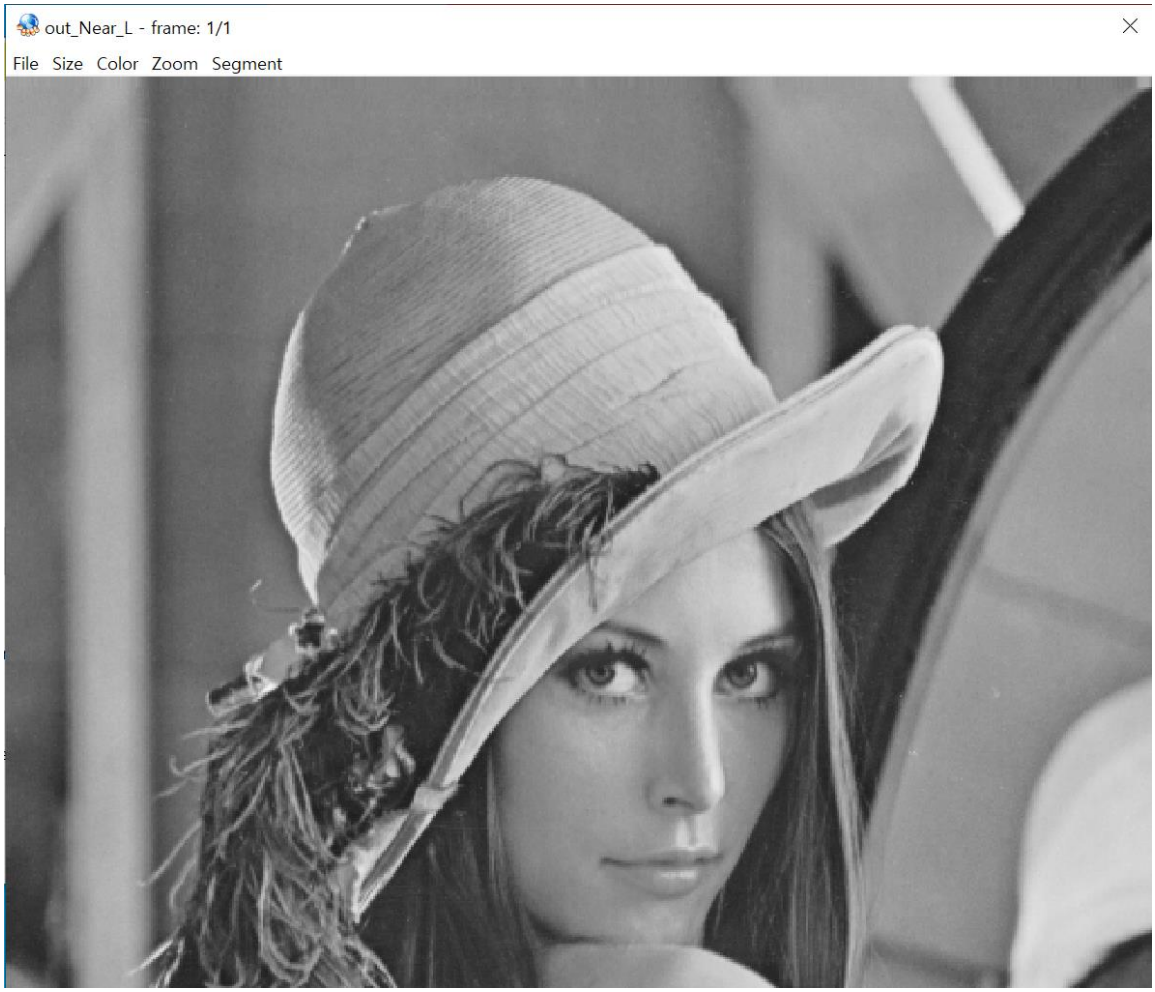
과제2 결과 보고서

18011789 조혜수

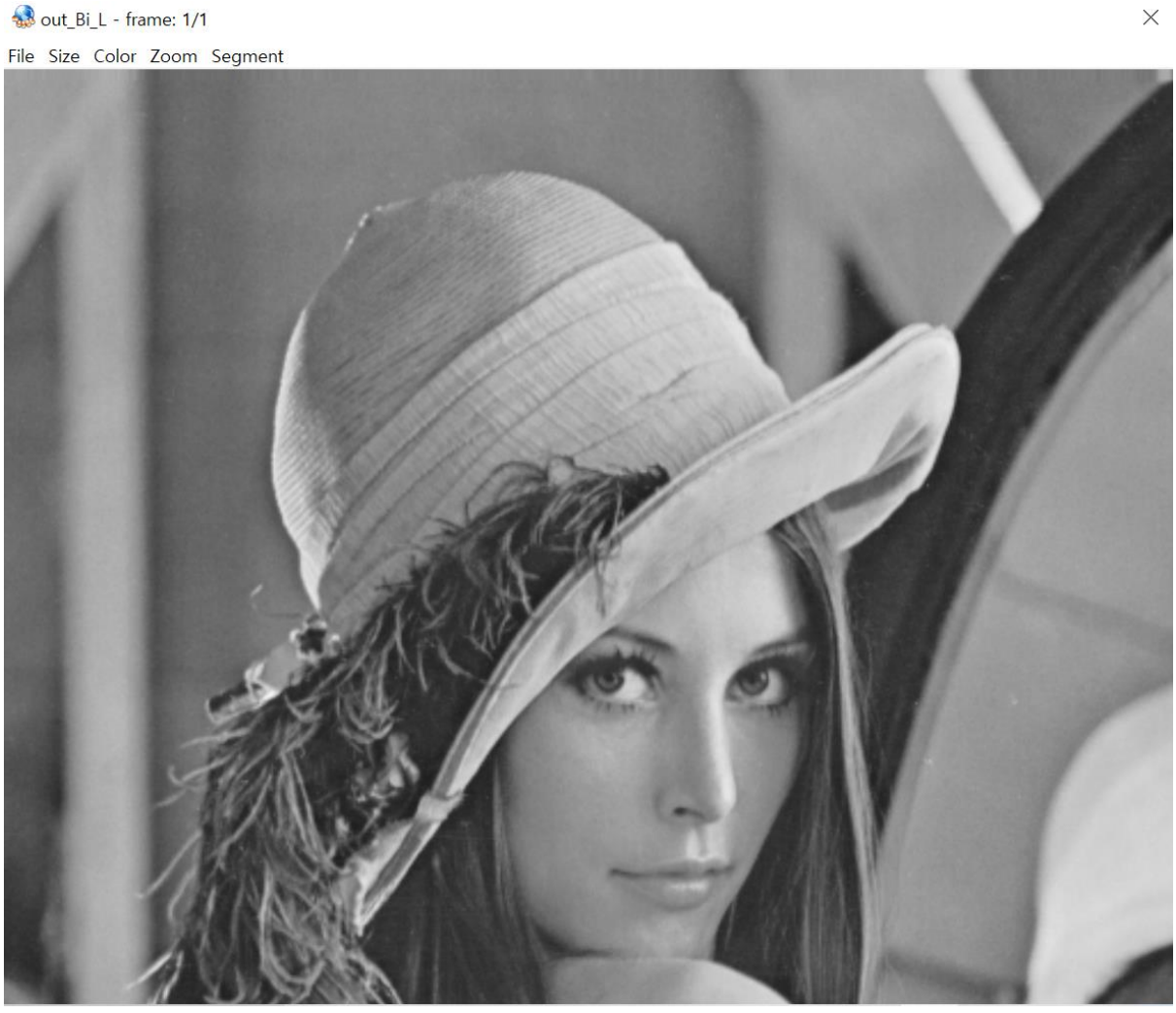
목차

1. 결과 영상

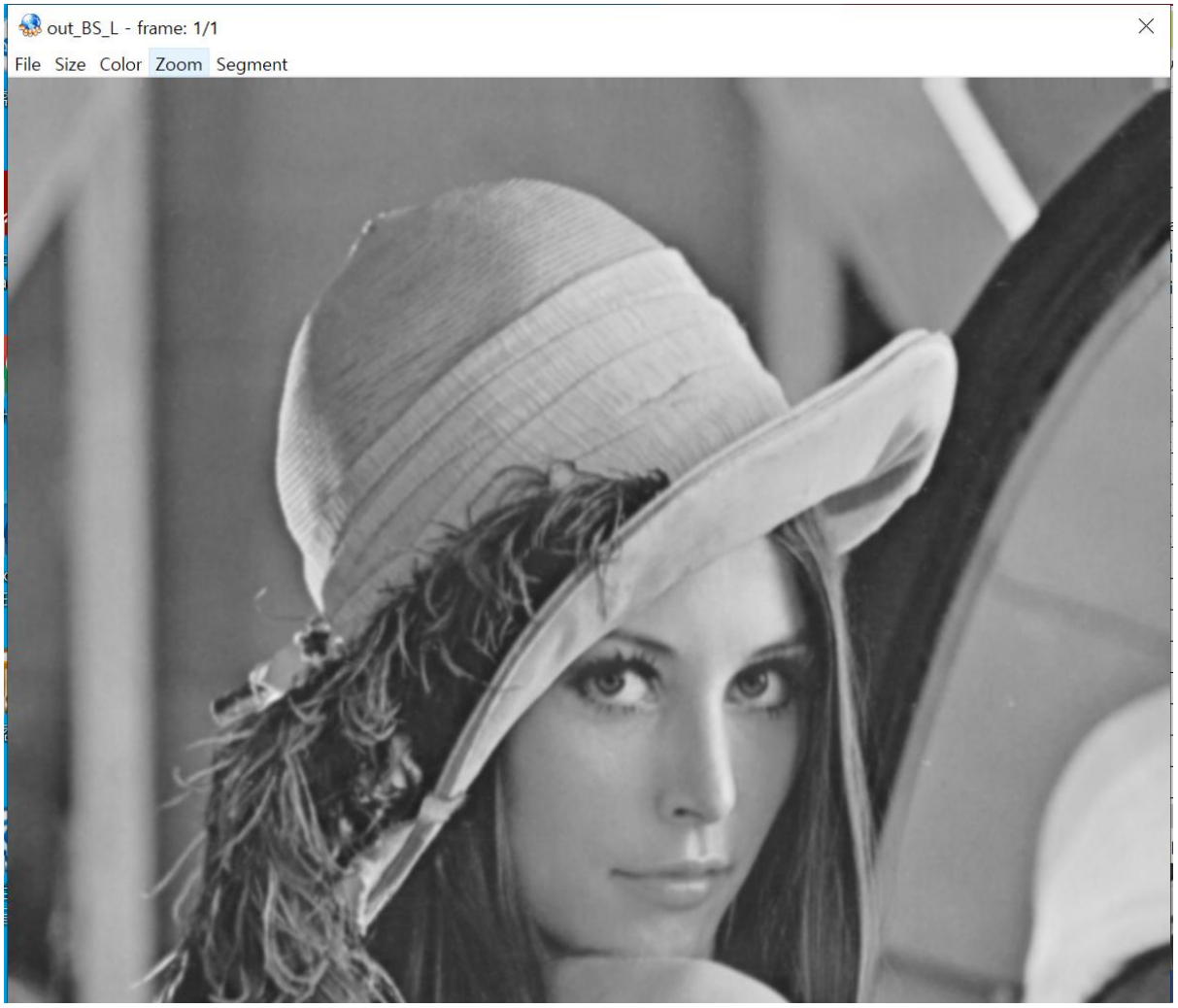
Near Neighbor Interpolation 축소 / 확대



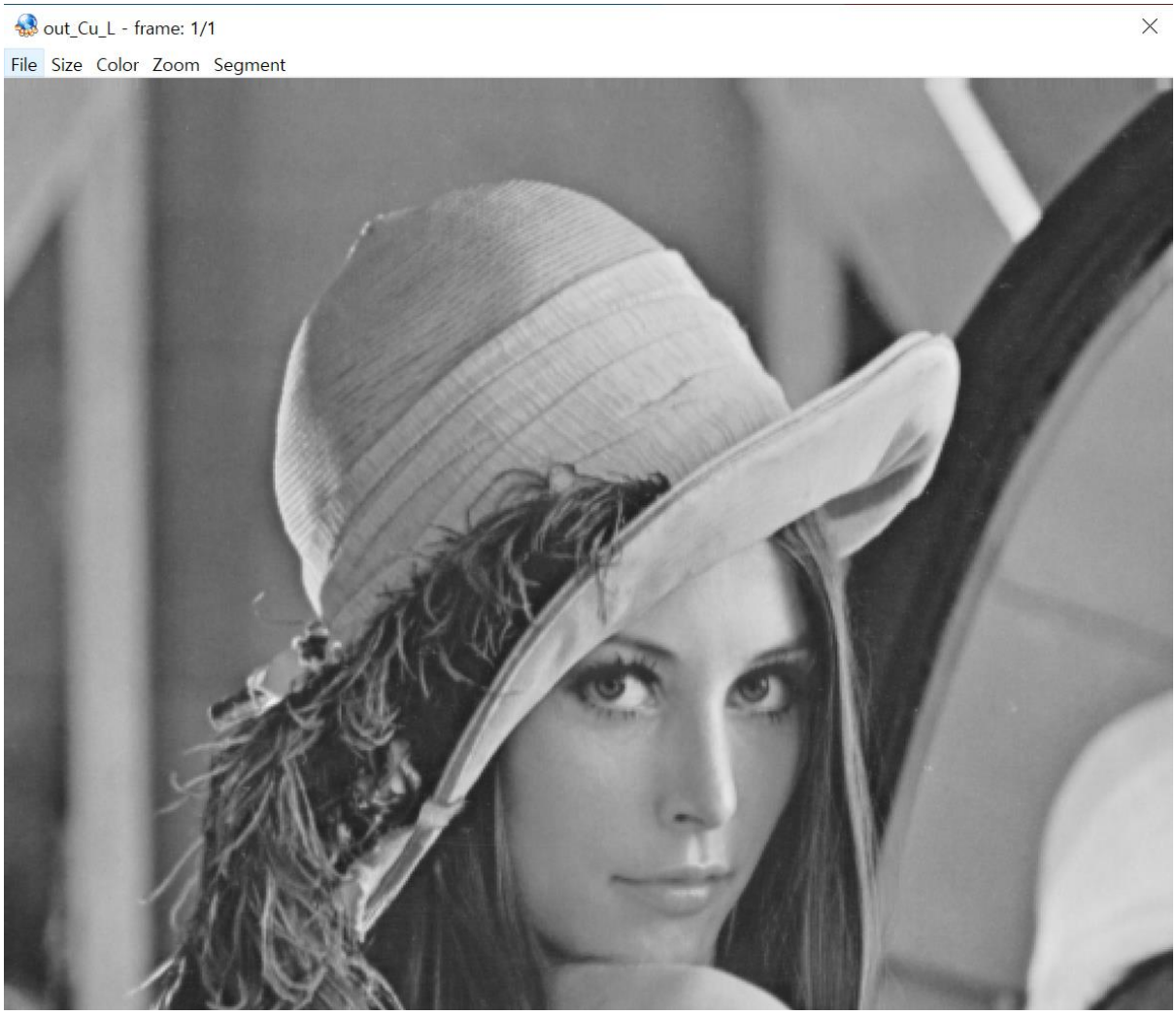
Bilinear Interpolation 축소 / 확대



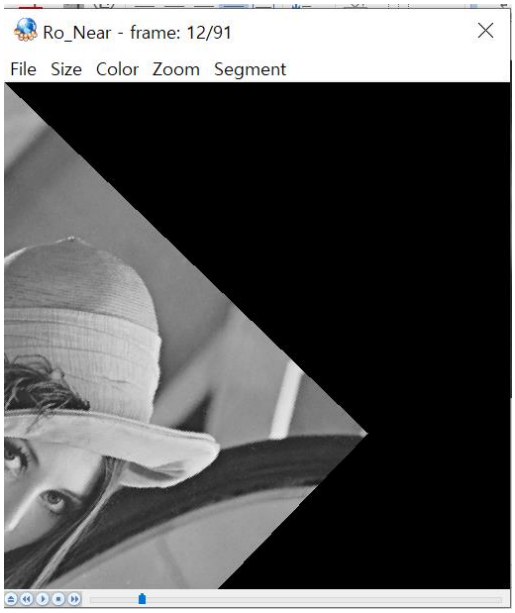
B-spline Interpolation 축소 / 확대

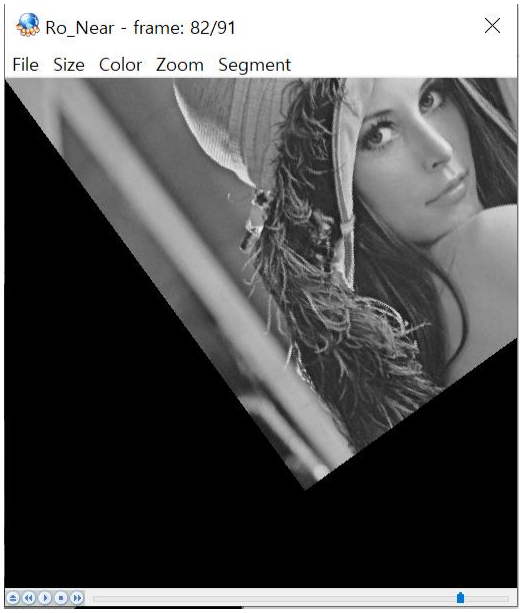


Cubic convolution Interpolation 축소 / 확대



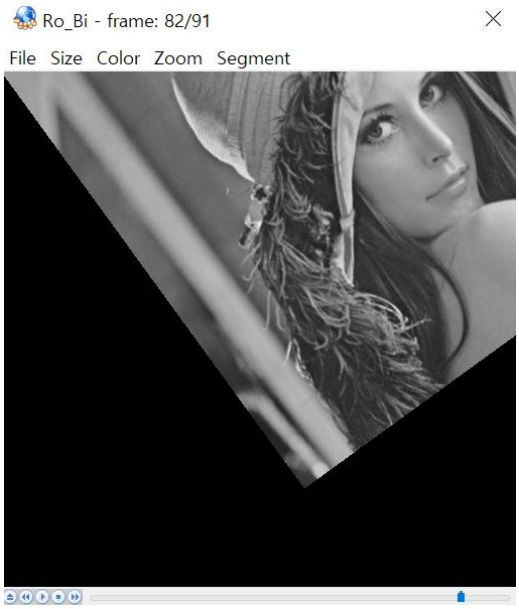
회전 – 중심점 : 원점 - Near Neighbor Interpolation



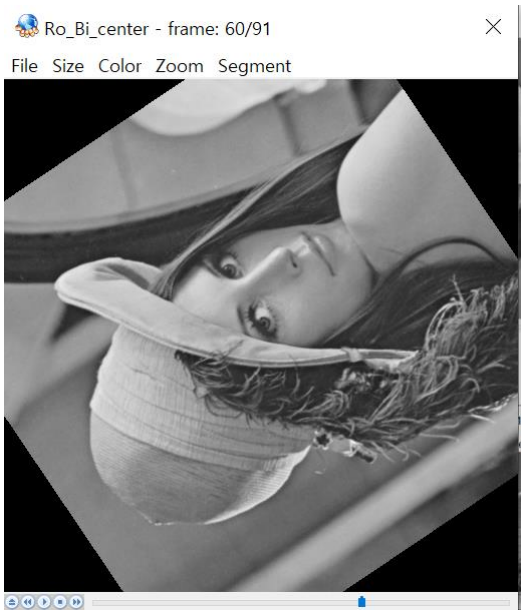


회전 - 중심점 : C(WIDTH / 2, HEIGHT / 2) - Near Neighbor Interpolation

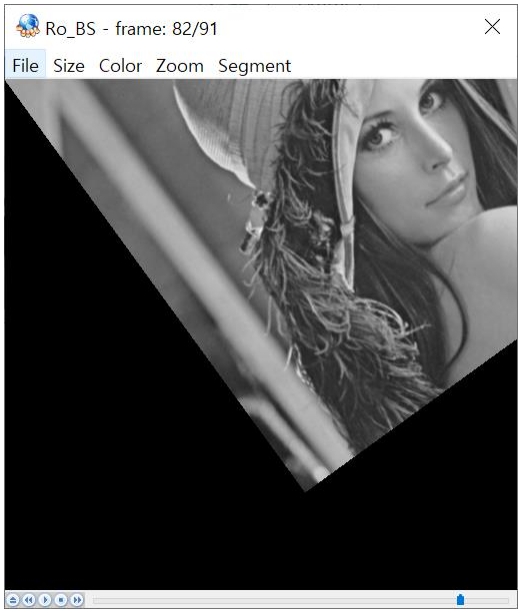
회전 - 중심점 : 원점- Bilinear Interpolation



회전 - 중심점 : C(WIDTH / 2, HEIGHT / 2) - Bilinear Interpolation



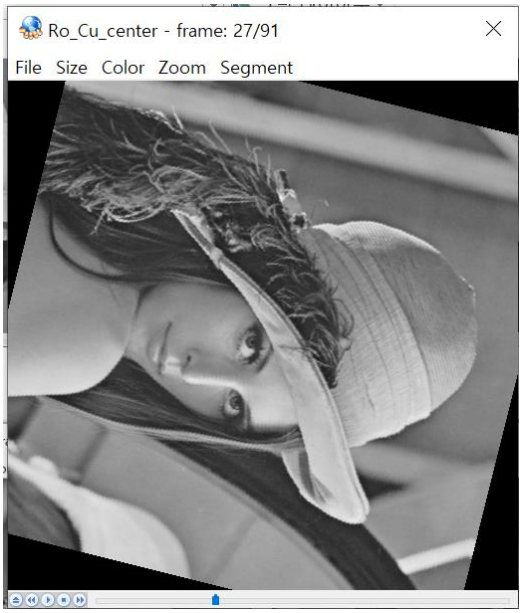
회전 - 중심점 : 원점 – B-spline Interpolation



회전 - 중심점 : C(WIDTH / 2, HEIGHT / 2) - B-spline Interpolation

회전 - 중심점 : 원점- Cubic convolution Interpolation

회전 - 중심점 : C(WIDTH / 2, HEIGHT / 2) - Cubic convolution Interpolation



2. 코드 분석

- main.h

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <memory.h>
#include <math.h>
#define _CRT_SECURE_NO_WARNINGS


#define WIDTH          512                  // 영상의 가로 크기
#define HEIGHT         512                  // 영상의 세로 크기

//#define scaleConstant    0.47              // 스케일링 상수 (축소)
#define scaleConstant    2.32               // 스케일링 상수 (확대)

#define maxVal       255
#define minVal       0

typedef unsigned char UChar;
typedef char          Char;
typedef double                Double;
typedef int           Int;
```

메인 헤더파일을 통해 여러 변수들을 정의 해준다.

축소1번 확대1번 총 두번 실행시켜 스케일링을 두번 해주었으며 축소시엔 scaleConstant = 0.47으로, 확대시엔 scaleConstant = 2.32 로 해주었다.

- GEO.h

```
#include "main.h"

#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 PI                                  3.14159265358979323846264338327 9

typedef struct _Scale_Buffer
{
        UChar* Near;
        UChar* Bi;
        UChar* BS;
        UChar* Cu;

}SCALE;

typedef struct _Rotation_Buffer
{
        UChar* Near;
        UChar* Bi;
        UChar* BS;
        UChar* Cu;

}ROTATION;

typedef struct _Image_Buffer
{

        UChar* padding; // 패딩 영상 저장 버퍼
        UChar* Result_Blurring;          // 축소 영상을 위한 블러링 결과 저장

}Img_Buf;

void Geometric_Transformation(UChar* Data, Img_Buf* img);
void ImageOutput(UChar* Data, Int wid, Int hei, Char String[]);

UChar NearesetNeighbor(UChar* Data, Double srcX, Double srcY, Int Stride);
UChar Bilinear(UChar* Data, Double srcX, Double srcY, Int Stride);
UChar B_Spline(UChar* Data, Double srcX, Double srcY, Int Stride);
```

```
UChar Cubic(UChar* Data, Double srcX, Double srcY, Int Stride);
```

여러 구조체들에 Cubic Interpolation 을 위한 변수를 추가 해주었다.

Geometric 관련 함수 등 여러 함수들의 선언이 들어가 있다.

- main.c

```
#include "GEO.h"

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


	Int wid = WIDTH; Int hei = HEIGHT;
	Int min = minVal; Int max = maxVal;

	fopen_s(&fp, "lena_512x512.raw", "rb"); //원본 영상 열기

	ori = (UChar*)malloc(sizeof(UChar) * (wid * hei)); //원본 영상 크기만큼 공간 선언
	memset(ori, 0, sizeof(UChar) * (wid * hei)); //0으로 초기화
	fread(ori, sizeof(UChar), (wid * hei), fp); // 원본 영상 읽기(원본 영상의 픽셀 값을 배열 변수에 저장)

	Geometric_Transformation(ori, &image); //함수 호출

	free(ori);
	fclose(fp);
}
```

원본 영상을 열어주고, 미리 정의한 영상 크기 변수들을 통해 원본 영상을 저장할 공간을 초기화 해준다. 그 후 원본 영상을 Geometric_Transformation 함수에 넣어 호출해준다.

함수 호출,실행이 끝나면 할당 받은 공간을 해제해준다.

- GEO.c

메인함수에서 호출한 Geometric_Transformation 함수가 정의 되어있다.

Geometric_Transformation 함수에서 스케일링, 회전을 위한 Scaling 함수와, Rotation 함수도 정의 되어 있다.

그 외에도 Image_Padding, Blurring, Image_Filtering 함수가 정의 되어 있다.

```
#include "GEO.h"

void Scaling(UChar* Data, Int dstWid, Int dstHei, Double scaleVal) // 스케일링 함수 ( 확대/축소 )
{//원본 데이터와 계산된 output 영상의 크기와 몇 배로 스케일링할 지에대한 변수를 매개변수로 받음
	SCALE scale;
	double srcX, srcY; //역 추적한 원본 화소 위치 (정수 위치가 아닐 수 있음)

	Char String[4][10] = { "Near_L", "Bi_L" , "BS_L" ,"Cu_L" }; //파일 명으로 출력될 문자열들.
축소시엔 ("Near") 확대시엔 ("Near_L")

	scale.Near = (UChar*)calloc(dstWid * dstHei, sizeof(UChar));// 4가지 보간 법으로 각각
축소/확대를 진행 함
	scale.Bi = (UChar*)calloc(dstWid * dstHei, sizeof(UChar)); // 각각 output 영상을 담을 공간
할당
	scale.BS = (UChar*)calloc(dstWid * dstHei, sizeof(UChar));
	scale.Cu = (UChar*)calloc(dstWid * dstHei, sizeof(UChar));

	for (int i = 0; i < dstHei; i++)
	{
		for (int j = 0; j < dstWid; j++)
		{
			srcX = (double)j / scaleVal;
			srcY = (double)i / scaleVal;

			scale.Near[i * dstWid + j] = NearesetNeighbor(Data, srcX, srcY, WIDTH);
			scale.Bi[i * dstWid + j] = Bilinear(Data, srcX, srcY, WIDTH);
			scale.BS[i * dstWid + j] = B_Spline(Data, srcX, srcY, WIDTH);
			scale.Cu[i * dstWid + j] = Cubic(Data, srcX, srcY, WIDTH); // 각 보간법에
대한 함수를 호출
		}
	}

	ImageOutput(scale.Near, dstWid, dstHei, String[0]);
```

```
	ImageOutput(scale.Bi, dstWid, dstHei, String[1]);
	ImageOutput(scale.BS, dstWid, dstHei, String[2]);
	ImageOutput(scale.Cu, dstWid, dstHei, String[3]); // 이미지 출력

	free(scale.Near);
	free(scale.Bi);
	free(scale.BS);
	free(scale.Cu); // 공간 해제
}

void Rotation (UChar* Data) // 회전 함수
{// 원본영상을 매개변수로 받음
	ROTATION rot;
	FILE* up1, * up2, * up3, * up4;

	double Angle;
	double srcX, srcY; // Source 위치

	int New_X, New_Y;
	int Center_X = WIDTH / 2, Center_Y = HEIGHT / 2;


	//회전 중심 : 원점일 경우

	fopen_s(&up1, "Ro_Near.raw", "wb");
	fopen_s(&up2, "Ro_Bi.raw", "wb");
	fopen_s(&up3, "Ro_BS.raw", "wb");
	fopen_s(&up4, "Ro_Cu.raw", "wb");

	rot.Near = (UChar*)calloc(WIDTH * HEIGHT, sizeof(UChar)); //output 영상을 위한 공간 할당
	rot.Bi = (UChar*)calloc(WIDTH * HEIGHT, sizeof(UChar));
	rot.BS = (UChar*)calloc(WIDTH * HEIGHT, sizeof(UChar));
	rot.Cu = (UChar*)calloc(WIDTH * HEIGHT, sizeof(UChar));

	for (Angle = 0; Angle <= 360; Angle += 4) //4도씩 돌려 한바퀴(360도) 돌림
	{
		double Seta = PI / 180.0 * Angle; // 세타 구해줌

		for (int i = 0; i < HEIGHT; i++)
		{
			for (int j = 0; j < WIDTH; j++)
```

```
                        {
                                srcX = cos(Seta) * j + sin(Seta) * i; //inverse 추적.
                                srcY = -sin(Seta) * j + cos(Seta) * i;

                                New_X = (int)srcX;
                                New_Y = (int)srcY;//원시 화소가 영상 경계 밖에 있는지를
확인하기 위한 변수

                                if (!(New_X < 0 || New_X >= WIDTH - 1 || New_Y < 0 || New_Y >=
HEIGHT - 1)) // 원시 화소가 영상 경계 밖에 위치
                                {
                                        rot.Near[i * WIDTH + j] = NearesetNeighbor(Data, srcX,
srcY, WIDTH);
                                        rot.Bi[i * WIDTH + j] = Bilinear(Data, srcX, srcY, WIDTH);
                                        rot.BS[i * WIDTH + j] = B_Spline(Data, srcX, srcY, WIDTH);
                                        rot.Cu[i * WIDTH + j] = Cubic(Data, srcX, srcY,
WIDTH);//원시 화소가 영상 경계 밖에 위치하지 않을 경우 보간 함수를 호출
                                }
                                else
                                {
                                        rot.Near[i * WIDTH + j] = 0;
                                        rot.Bi[i * WIDTH + j] = 0;
                                        rot.BS[i * WIDTH + j] = 0;
                                        rot.Cu[i * WIDTH + j] = 0;//원시 화소가 영상 경계 밖에
위치할 경우 화소값 0
                                }
                        }
                }

                fwrite(rot.Near, sizeof(UChar), (WIDTH * HEIGHT), up1);
                fwrite(rot.Bi, sizeof(UChar), (WIDTH * HEIGHT), up2);
                fwrite(rot.BS, sizeof(UChar), (WIDTH * HEIGHT), up3);
                fwrite(rot.Cu, sizeof(UChar), (WIDTH * HEIGHT), up4);
        }

        free(rot.Near);
        free(rot.Bi);
        free(rot.BS);
        free(rot.Cu);

        fclose(up1);
```

```
		fclose(up2);
		fclose(up3);
		fclose(up4);


		//회전 중심 :  C ( WIDTH / 2,  HEIGHT / 2 ) 일 경우;

		fopen_s(&up1, "Ro_Near_center.raw", "wb");
		fopen_s(&up2, "Ro_Bi_center.raw", "wb");
		fopen_s(&up3, "Ro_BS_center.raw", "wb");
		fopen_s(&up4, "Ro_Cu_center.raw", "wb"); // 파일이름을 바궈서 다시 오픈

		rot.Near = (UChar*)calloc(WIDTH * HEIGHT, sizeof(UChar));
		rot.Bi = (UChar*)calloc(WIDTH * HEIGHT, sizeof(UChar));
		rot.BS = (UChar*)calloc(WIDTH * HEIGHT, sizeof(UChar));
		rot.Cu = (UChar*)calloc(WIDTH * HEIGHT, sizeof(UChar)); // 다시 메모리 할당

		for (Angle = 0; Angle <= 360; Angle += 4)
		{// 원점일 때와 거의 동일 하나 srcX, srcY 을 구하는 식만 달라짐
			double Seta = PI / 180.0 * Angle;

			for (int i = 0; i < HEIGHT; i++)
			{
				for (int j = 0; j < WIDTH; j++)
				{
					srcX = Center_X - Center_X * cos(Seta) + cos(Seta) * j - Center_Y *
sin(Seta) + i * sin(Seta);
					srcY = Center_Y - Center_Y * cos(Seta) + i * cos(Seta) + Center_X *
sin(Seta) - j * sin(Seta);

					New_X = (int)srcX;
					New_Y = (int)srcY;

					if (!(New_X < 0 || New_X >= WIDTH - 1 || New_Y < 0 || New_Y >=
HEIGHT - 1)) // 원시 화소가 영상 경계 밖에 위치
					{
						rot.Near[i * WIDTH + j] = NearesetNeighbor(Data, srcX,
srcY, WIDTH);
						rot.Bi[i * WIDTH + j] = Bilinear(Data, srcX, srcY, WIDTH);
						rot.BS[i * WIDTH + j] = B_Spline(Data, srcX, srcY, WIDTH);
						rot.Cu[i * WIDTH + j] = Cubic(Data, srcX, srcY, WIDTH);
```

```
                                        }
                                        else
                                        {
                                                rot.Near[i * WIDTH + j] = 0;
                                                rot.Bi[i * WIDTH + j] = 0;
                                                rot.BS[i * WIDTH + j] = 0;
                                                rot.Cu[i * WIDTH + j] = 0;
                                        }
                                }
                        }

                        fwrite(rot.Near, sizeof(UChar), (WIDTH * HEIGHT), up1);
                        fwrite(rot.Bi, sizeof(UChar), (WIDTH * HEIGHT), up2);
                        fwrite(rot.BS, sizeof(UChar), (WIDTH * HEIGHT), up3);
                        fwrite(rot.Cu, sizeof(UChar), (WIDTH * HEIGHT), up4);
                }

                free(rot.Near);
                free(rot.Bi);
                free(rot.BS);
                free(rot.Cu);


                fclose(up1);
                fclose(up2);
                fclose(up3);
                fclose(up4);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////
void Image_Padding(Img_Buf* img, UChar* Buf, int width, int height, int Mask_size) //output 영상의
과도 축소를 막기 위한 패딩 함수
{
        int line, i, j;

        img->padding = (UChar*)calloc((width + Mask_size - 1) * (height + Mask_size - 1),
sizeof(UChar));
        for (line = 0; line < (Mask_size / 2); line++)
        {
                //상하단 패딩
                for (i = 0; i < width; i++)
```

```
		{
			img->padding[(width + Mask_size - 1) * line + Mask_size / 2 + i] = Buf[i];
			img->padding[(width + Mask_size - 1) * (height + Mask_size - 2 - line) +
Mask_size / 2 + i] = Buf[i + (width * (height - 1))];
		}

		//좌우측 패딩
		for (i = 0; i < height; i++)
		{
			img->padding[(width + Mask_size - 1) * (Mask_size / 2 + i) + line] = Buf[i *
width];
			img->padding[(width + Mask_size - 1) * (Mask_size / 2 + 1 + i) - 1 - line] =
Buf[i * width + (width - 1)];
		}
	}

	for (line = 0; line < 4; line++)
	{
		for (i = 0; i < (Mask_size / 2); i++)
		{
			for (j = 0; j < (Mask_size / 2); j++)
			{
				/*** 좌상단 패딩 ***/
				if (line == 0)
				{
					img->padding[(width + Mask_size - 1) * i + j] = Buf[0];
				}
				/*** 우상단 패딩 ***/
				else if (line == 1)
				{
					img->padding[(width + Mask_size - 1) * i + Mask_size /
2 + width + j] = Buf[width - 1];
				}
				/*** 좌하단 패딩 ***/
				else if (line == 2)
				{
					img->padding[(width + Mask_size - 1) * (height +
Mask_size - 2 - i) + j] = Buf[width * (height - 1)];
				}
				/*** 우하단 패딩 ***/
				else
```

```
                                        {
                                                img->padding[(width + Mask_size - 1) * (height +
Mask_size - 2 - i) + Mask_size / 2 + width + j] = Buf[width * height - 1];
                                        }
                                }
                        }
                }

                /*** 원본 버퍼 불러오기 ***/
                for (i = 0; i < height; i++)
                {
                        for (j = 0; j < width; j++)
                        {
                                img->padding[(width + Mask_size - 1) * (Mask_size / 2 + i) + Mask_size / 2
+ j] = Buf[i * width + j];
                        }
                }
}

UChar Blurring(UChar* buf, int Mask_size) // aliasing 을 피하기 위한 블러링 함수
{
        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 > maxVal ? maxVal :
Convolution_All_coeff < minVal ? minVal : Convolution_All_coeff;
}

void Image_Filtering(UChar* Data, Img_Buf* img)
{
        int Mask_size = 3;              //MxM size
        int Add_size = Mask_size / 2 + 1;
        UChar Padding_buf[9] = { 0 };

        Image_Padding(img, Data, WIDTH, HEIGHT, 3);

        img->Result_Blurring = (UChar*)calloc(WIDTH * HEIGHT, sizeof(UChar));
```

```
	for (int i = 0; i < HEIGHT; i++)
	{
		for (int j = 0; j < WIDTH; j++)
		{
			for (int k = 0; k < Mask_size; k++)
				for (int l = 0; l < Mask_size; l++)
					Padding_buf[k * Mask_size + l] = img->padding[(i + k) *
(WIDTH + Add_size) + (j + l)];

			img->Result_Blurring[i * WIDTH + j] = Blurring(&Padding_buf, Mask_size);
		}
	}
	free(img->padding);
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
//////////////////////////

void Geometric_Transformation(UChar* Data, Img_Buf* img) // main 함수에서 이미지 변형을 위해
호출한 함수
{

	Int wid = WIDTH; Int hei = HEIGHT;
	Int min = minVal; Int max = maxVal;

	Double scaleVal = scaleConstant; // 스케일링 크기 변수 ( 0.47 / 2.32 )

	Int dstWid; //스케일링 적용된 영상의 가로 길이
	Int dstHei; ////스케일링 적용된 영상의 세로 길이

	dstWid = wid * scaleVal + 0.5;
	dstHei = hei * scaleVal + 0.5; // 스케일링 될 영상의 크기 ( 반올림 포함 )

	Rotation(Data); //회전 함수 호출

	if (scaleVal < 1) // 축소시 원본 영상 블러링 적용
	{
		Image_Filtering(Data, img);
		memcpy(Data, img->Result_Blurring, sizeof(UChar) * wid * hei);
		free(img->Result_Blurring);
	}
```

```
        Scaling(Data, dstWid, dstHei, scaleVal); //스케일링 함수 호출
}
```

- INTERPOLATION.c

```
#include "GEO.h"

int Round(double x) //반올림 해주는 함수.
{
        x += 0.5;
        x = (int)x;
        return x;
}

int Min(int a, int b) { // 두 인자 중 더 작은 값을 찾아주는 함수
        if (a > b) return b;
        else return a;
}

int Max(int a, int b) { // 두 인자 중 더 큰 값을 찾아주는 함수
        if (a > b) return a;
        else return b;
}

UChar NearesetNeighbor(UChar* Data, Double srcX, Double srcY, Int Stride) // Near Neighbor Interpolation 함수
{//srcX와 srcY을 역 추적해서 주변 점 4개중 가장 가까운 화소 값을 찾아줌 ( 반올림을 통해 찾을 수 있음 )
        return Data[((int)(srcY + 0.5) * Stride + (int)(srcX + 0.5))];
}

UChar Bilinear(UChar* Data, Double srcX, Double srcY, Int Stride) //Bilinear Interpolation 함수
{ //3번의 linear Interpolation을 통해 화소 값을 찾아줌
        int SrcX_Plus1, SrcY_Plus1;
        double Hor_Wei, Ver_Wei; //Horizontal Weight, Vertical Weight
        int TL, TR, BL, BR; //각 화소 위치
```

```
		SrcX_Plus1 = CLIPPIC_HOR((int)srcX +1);// src 화소의 왼쪽 위 화소는 src(실수)에서 버림을 통해 얻을 수 있음
		SrcY_Plus1 = CLIPPIC_VER((int)srcY + 1);// 따라서 src를 이용해서 X축, Y축으로 +1 된 점 위치를 찾아줌


		Hor_Wei = srcX - (int)srcX;
		Ver_Wei = srcY - (int)srcY;  //가중치 값 구하기 (거리 구하기)

		TL = (int)srcY * Stride + (int)srcX; //위에서 찾아준 Src_Plus1 값을 통해 주변 4개 화소 위치들을 찾아줌
		TR = (int)srcY * Stride + SrcX_Plus1;
		BL = SrcY_Plus1 * Stride + (int)srcX;
		BR = SrcY_Plus1 * Stride + SrcX_Plus1;

		UChar TMP =  // 수평 방향(x축 방향)으로 먼저 2번 linear Interpolation 해주고, 결과 값들로 수직 방향 (y축 방향)으로 linear Interpolation
			(1 - Ver_Wei) * (((1 - Hor_Wei) * Data[TL]) + (Hor_Wei * Data[TR])) +

			Ver_Wei * (((1 - Hor_Wei) * Data[BL]) + (Hor_Wei * Data[BR]));

		TMP = Min(Max(Round(TMP), 0), 255); // 반올림, 클리핑 과정
		return TMP;
}

double BSpline_function(Double x) { //화소에 weight를 주는 커널 함수 정의 (BSpline 보간법에 해당하는)
		// 매개변수 x는 각각의 주변화소와 src 화소와의 거리

		double result;

		if (x < 0) x *= -1; // X의 절댓값으로 계산 하기 위한 과정

		if (fabs(x) >= 0 && fabs(x) < 1) {
			result = (0.5) * x * x * x - x * x + (0.66666);
		}
		else if (fabs(x) >= 1 && fabs(x) < 2) {
			result = (-0.166666) * x * x * x + x * x - 2 * x + (1.3333);
		}
		else
```

```
                result = 0;

        return result;
}

UChar B_Spline(UChar* Data, Double srcX, Double srcY, Int Stride)// B_Spline Interpolation 함수
//주변 16개 점을 5번의 1차원 Interpolation을 통해 화소 값을 찾아줌 ( 4번 수평방향으로 보간, 그 결과를 갖고 수직방향으로 1번 보간)
{
        int Src_X_Minus_1, Src_X_Plus_1, Src_X_Plus_2;
        int Src_Y_Minus_1, Src_Y_Plus_1, Src_Y_Plus_2;
        double Hor_Wei, Ver_Wei; //Horizontal Weight, Vertical Weight
        double TMP_Hor[4] = { 0,0,0,0 };
        double TMP = 0;

        Src_X_Plus_1 = CLIPPIC_HOR((int)srcX + 1);
        Src_X_Plus_2 = CLIPPIC_HOR((int)srcX + 2);
        Src_Y_Plus_1 = CLIPPIC_VER((int)srcY + 1);
        Src_Y_Plus_2 = CLIPPIC_VER((int)srcY + 2);

        Src_X_Minus_1 = CLIPPIC_HOR((int)srcX - 1);
        Src_Y_Minus_1 = CLIPPIC_VER((int)srcY - 1); //위와 같은 방법으로 주변 화소 위치 찾기 위한 과정

        Hor_Wei = srcX - (int)srcX;
        Ver_Wei = srcY - (int)srcY;

        int X_Pix[] = { Src_X_Minus_1, (int)srcX, Src_X_Plus_1, Src_X_Plus_2 };
        int Y_Pix[] = { Src_Y_Minus_1, (int)srcY, Src_Y_Plus_1, Src_Y_Plus_2 };
        double Distance_Hor[] = { Hor_Wei + 1, Hor_Wei, 1 - Hor_Wei, (1 - Hor_Wei) + 1 };
        double Distance_Ver[] = { Ver_Wei + 1, Ver_Wei, 1 - Ver_Wei, (1 - Ver_Wei) + 1 };

        for (int i = 0; i < 4; i++)
        {
                for (int j = 0; j < 4; j++)
                {
                        TMP_Hor[i] += BSpline_function(Distance_Hor[j]) * Data[Y_Pix[i] * Stride + X_Pix[j]];
                }
        }
```

```
	for (int i = 0; i < 4; i++)
	{
		TMP += BSpline_function(Distance_Ver[i]) * TMP_Hor[i];
	}

	TMP = Min(Max(Round(TMP), 0), 255); // 클리핑과 반올림

	return (UChar)TMP;
}



double Cubic_function(Double x){ //화소에 weight를 주는 커널 함수 정의 (Cubic 보간법에 해당하는)
// 매개변수 x는 각각의 주변화소와 src 화소와의 거리

	double result;
	double a;
	a = 0.5; // 커널 파라미터 b,c가 b=0,c=1/2 가 되는 것이 a=1/2 되는 것임

	if (x < 0) x *= -1;  // X의 절댓값으로 계산 하기 위한 과정

	if (fabs(x) >= 0 && fabs(x) < 1) {
		result = (a + 2) * x * x * x - (a + 3) * x * x + 1;
	}
	else if (fabs(x) >= 1 && fabs(x) < 2) {
		result = a * x * x * x - 5 * a * x * x + 8 * a * x - 4 * a;
	}
	else
		result = 0;

	return result;
}

UChar Cubic(UChar* Data, Double srcX, Double srcY, Int Stride)// Cubic Interpolation 함수
//주변 16개 점을 5번의 1차원 Interpolation을 통해 화소 값을 찾아줌 ( 4번 수평방향으로 보간, 그
결과를 갖고 수직방향으로 1번 보간)
//B-spline 보간 함수와 동일
{
	int Src_X_Minus_1, Src_X_Plus_1, Src_X_Plus_2;
	int Src_Y_Minus_1, Src_Y_Plus_1, Src_Y_Plus_2;
	double Hor_Wei, Ver_Wei; //Horizontal Weight, Vertical Weight
```

```
	double TMP_Hor[4] = { 0,0,0,0 };
	double TMP = 0;

	Src_X_Plus_1 = CLIPPIC_HOR((int)srcX + 1);
	Src_X_Plus_2 = CLIPPIC_HOR((int)srcX + 2);
	Src_Y_Plus_1 = CLIPPIC_VER((int)srcY + 1);
	Src_Y_Plus_2 = CLIPPIC_VER((int)srcY + 2);

	Src_X_Minus_1 = CLIPPIC_HOR((int)srcX - 1);
	Src_Y_Minus_1 = CLIPPIC_VER((int)srcY - 1);

	Hor_Wei = srcX - (int)srcX;
	Ver_Wei = srcY - (int)srcY;

	int X_Pix[] = { Src_X_Minus_1, (int)srcX, Src_X_Plus_1, Src_X_Plus_2 };
	int Y_Pix[] = { Src_Y_Minus_1, (int)srcY, Src_Y_Plus_1, Src_Y_Plus_2 };
	double Distance_Hor[] = { Hor_Wei + 1, Hor_Wei, 1 - Hor_Wei, (1 - Hor_Wei) + 1 };
	double Distance_Ver[] = { Ver_Wei + 1, Ver_Wei, 1 - Ver_Wei, (1 - Ver_Wei) + 1 };

	for (int i = 0; i < 4; i++)
	{
		for (int j = 0; j < 4; j++)
		{
			TMP_Hor[i] += Cubic_function(Distance_Hor[j]) * Data[Y_Pix[i] * Stride +
X_Pix[j]];
		}
	}

	for (int i = 0; i < 4; i++)
	{
		TMP += Cubic_function(Distance_Ver[i]) * TMP_Hor[i];
	}

	TMP = Min(Max(Round(TMP), 0), 255); // 클리핑과 반올림 필요

	return (UChar)TMP;
}
```

- Imageout.c

```
#include "GEO.h"

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

	FILE *fp;



	strcat_s(Name_Hist, 20, String);
	strcat_s(Name_Hist, 20, Name_extension);


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

3. Blurring 사용 이유

Blurring 이란 흐리게 만든다는 뜻이고 이는 대비를 줄인다는 것이다. 따라서 frequency 높은 영역을 없애는 작업이다.
이는 Downsampling 할 때에 Aliasing 를 해결하기 위해 사용된다.

Downsampling 시 Aliasing 문제가 발생할 수 있다.
예를 들어 4 배로 축소 한다고 했을 때 원본 영상에서 화소 4 개중 한개만 사용하게 된다.
그럼 나머지 3 개의 화소는 사용하지 않게 되어 output 영상에서 일부는 표현이 안되는 것이다.

좀 더 자세하게는
A bandwidth 를 갖는 영상이 있을 때 Downsampling 을 하게 되면 sampling frequency 도 줄어들게 되어, 원본영상의 frequency 와 sampling 된 영상의 frequency 가 겹치게 된다.

이를 해결 하기 위해 Blurring(= Low pass filtering) 을 사용한다.
frequency 가 겹치지 않도록 원본 영상의 bandwidth 를 줄이는 것이다.