영상처리 과제 2

16010811 김범수

이미지 및 Scale_factor 설정

```
void INPUT_FILE(Img_Buf* img)
{
    File_Info file;

    fopen_s(&file.Input_Ori_img, "lena_512x512.raw ", "rb");                    //lena 오픈
    //fopen_s(&file.Input_Ori_img, "city_1280x720.raw ", "rb");                 //city 오픈

    IMAGE_INITIALIZATION(img, &file);

    fclose(file.Input_Ori_img);
}
```

- Image.c -> INPUT_FILE 함수 내에서 Lena or City
- 어떤 이미지 사용할지 설정

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

//#define WIDTH 1280                //City 영상의 가로 크기
//#define HEIGHT 720                //City 영상의 세로 크기
```

- main.h 내에서 이미지 크기 설정

```
#define Scale_factor  0.8
//#define Scale_factor  2.32
//#define Scale_factor  0.47
```

- main.h 내에서 Scale_factor 설정

Gemetric_Tranfomation

```
void Geometric_Trancsformation(Img_Buf* img)
{
    double Scaling_Val = Scale_factor;

    int New_Wid = (double)img->info.width * Scaling_Val + 0.5;                    //새로 바뀐 가로세로
    int New_Hei = (double)img->info.height * Scaling_Val + 0.5;

    if (Scaling_Val < 1) // 영상 축소를 위한 블러링
    {
        Image_Filtering(img);  ////////////// 순서 3-2번 (영상 축소시 사용) ---> Filtering.c 코드
        memcpy(img->Ori_img, img->Result_Blurring, sizeof(UChar) * img->info.width * img->info.height);
        free(img->Result_Blurring);
    }


    Scaling (img, New_Wid, New_Hei, Scaling_Val);
    //Rotation(img);
    Rotation_23_Zero(img);
    Rotation_23_Cen(img);


    FILE* wp;
    fopen_s(&wp, "Img_Near_Scale.raw", "wb");
    fwrite(img->Near_Scale, sizeof(UChar), (New_Wid * New_Hei), wp);
    free(img->Near_Scale);

    fopen_s(&wp, "Img_Bi_Scale.raw", "wb");
    fwrite(img->Bi_Scale, sizeof(UChar), (New_Wid * New_Hei), wp);
    free(img->Bi_Scale);

    fopen_s(&wp, "Img_Cubic_Scale.raw", "wb");
    fwrite(img->Cubic_Scale, sizeof(UChar), (New_Wid * New_Hei), wp);
    free(img->Cubic_Scale);

    fopen_s(&wp, "Img_Bspline_Scale.raw", "wb");
    fwrite(img->Bspline_Scale, sizeof(UChar), (New_Wid * New_Hei), wp);
    free(img->Bspline_Scale);
    fclose(wp);
}
```

- Scaling을 위한 New_Wid, New_Hei 설정
- 영상 축소 시 Artifact를 줄이기 위한 Blurring 작업

- Scaling, Rotation, 원점 기준 23도 Rotation, 중심 기준 23도 Rotation 진행

- Scaling 기록

Scaling

```
void Scaling(Img_Buf* img, int New_Wid, int New_Hei, double Scaling_Val)
{
    double Src_x, Src_y;                                                    //원본 화소 위치

    img->Near_Scale = (UChar*)calloc(New_Wid * New_Hei, sizeof(UChar));
    img->Bi_Scale   = (UChar*)calloc(New_Wid * New_Hei, sizeof(UChar));
    img->Cubic_Scale = (UChar*)calloc(New_Wid * New_Hei, sizeof(UChar));
    img->Bspline_Scale = (UChar*)calloc(New_Wid * New_Hei, sizeof(UChar));

    for (int i = 0; i < New_Hei; i++)
    {
        for (int j = 0; j < New_Wid; j++)
        {
            Src_x = (double)j / Scaling_Val;
            Src_y = (double)i / Scaling_Val;

            img->Near_Scale[i * New_Wid + j] = NearesetNeighbor(img->Ori_img, Src_x, Src_y, img->info.width);
            img->Bi_Scale  [i * New_Wid + j] = Bilinear        (img->Ori_img, Src_x, Src_y, img->info.width);
            img->Cubic_Scale[i * New_Wid + j] = Cubic          (img->Ori_img, Src_x, Src_y, img->info.width);
            img->Bspline_Scale[i * New_Wid + j] = Bspline      (img->Ori_img, Src_x, Src_y, img->info.width);
        }
    }
}
```

- Nearest, Bilinear, Cubic, B-spline Scaling image 동적 할당
- 실수 좌표 Src_x, Src_y 생성
- Interpolation을 이용한 Scaling 실행

Nearest & Bilinear Interpolation

```
UChar NearesetNeighbor(UChar *Data, double Src_X, double Src_Y, int Stride)
{
    return Data[((int)(Src_Y + 0.5) * Stride + (int)(Src_X + 0.5))];
}
```

```
UChar Bilinear(UChar* Data, double Src_X, double Src_Y, int Stride)
{
    int Src_X_Plus_1, Src_Y_Plus_1;
    double Hor_Wei, Ver_Wei; //Horizontal Weight, Vertical Weight
    int TL, TR, BL, BR; //각 화소 위치

    Src_X_Plus_1 = CLIP_HOR((int)Src_X + 1);
    Src_Y_Plus_1 = CLIP_VER((int)Src_Y + 1);

    Hor_Wei = Src_X - (int)Src_X;
    Ver_Wei = Src_Y - (int)Src_Y;

    TL = (int)Src_Y   * Stride + (int)Src_X;        //top left 좌표
    TR = (int)Src_Y   * Stride + Src_X_Plus_1;
    BL = Src_Y_Plus_1 * Stride + (int)Src_X;
    BR = Src_Y_Plus_1 * Stride + Src_X_Plus_1;

    UChar TMP =
    (1 - Ver_Wei)*  (((1 - Hor_Wei) * Data[TL]) + (Hor_Wei * Data[TR])) +

         Ver_Wei *  (((1 - Hor_Wei) * Data[BL]) + (Hor_Wei * Data[BR])) + 0.5;

    return TMP;
}
```

- 강제 형변환을 이용한 Nearest Neighbor Interpolation
- Clipping 과정을 포함한 Src_x + 1, Src_y + 1 좌표 생성
- Intrepolation의 변수가 될 거리 가중치 Hor_Wei, Ver_Wei 생성
- 근처 좌표 4개를 이용한 Interpolation 수행, 반올림 과정이 필요하다고 생각하여 추가 후 강제 형변환

Cubic Convolution Interpolation

```
UChar Cubic(UChar* Data, double Src_X, double Src_Y, int Stride)
{
    int Src_X_Minus_1, Src_X_Plus_1, Src_X_Plus_2, Src_Y_Minus_1, Src_Y_Plus_1, Src_Y_Plus_2;
    double Hor_Wei, Ver_Wei; //Horizontal Weight, Vertical Weight
    double a = 0.5;
    Src_X_Minus_1 = CLIP_MINUS((int)Src_X - 1);                                    //좌표
    Src_X_Plus_1 = CLIP_HOR((int)Src_X + 1);
    Src_X_Plus_2 = CLIP_HOR((int)Src_X + 2);
    Src_Y_Minus_1 = CLIP_MINUS((int)Src_Y - 1);
    Src_Y_Plus_1 = CLIP_VER((int)Src_Y + 1);
    Src_Y_Plus_2 = CLIP_VER((int)Src_Y + 2);

    Hor_Wei = Src_X - (int)Src_X;
    Ver_Wei = Src_Y - (int)Src_Y;


    double TMP_Cubic_0 =
        (a * pow(Hor_Wei + 1, 3) - 5 * a * pow(Hor_Wei + 1, 2) + 8 * a * (Hor_Wei + 1) - 4 * a) * Data[(int)Src_Y_Minus_1 * Stride + (int)Src_X_Minus_1]
        + ((a + 2) * pow(Hor_Wei, 3) - (a + 3) * pow(Hor_Wei, 2) + 1) * Data[(int)Src_Y_Minus_1 * Stride + (int)Src_X]
        + ((a + 2) * pow(1 - Hor_Wei, 3) - (a + 3) * pow(1 - Hor_Wei, 2) + 1) * Data[(int)Src_Y_Minus_1 * Stride + (int)Src_X_Plus_1]
        + (a * pow(2 - Hor_Wei, 3) - 5 * a * pow(2 - Hor_Wei, 2) + 8 * a * (2 - Hor_Wei) - 4 * a) * Data[(int)Src_Y_Minus_1 * Stride + (int)Src_X_Plus_2];

    double TMP_Cubic_1 =
        (a * pow(Hor_Wei + 1, 3) - 5 * a * pow(Hor_Wei + 1, 2) + 8 * a * (Hor_Wei + 1) - 4 * a) * Data[(int)Src_Y * Stride + (int)Src_X_Minus_1]
        + ((a + 2) * pow(Hor_Wei, 3) - (a + 3) * pow(Hor_Wei, 2) + 1) * Data[(int)Src_Y * Stride + (int)Src_X]
        + ((a + 2) * pow(1 - Hor_Wei, 3) - (a + 3) * pow(1 - Hor_Wei, 2) + 1) * Data[(int)Src_Y * Stride + (int)Src_X_Plus_1]
        + (a * pow(2 - Hor_Wei, 3) - 5 * a * pow(2 - Hor_Wei, 2) + 8 * a * (2 - Hor_Wei) - 4 * a) * Data[(int)Src_Y * Stride + (int)Src_X_Plus_2];

    double TMP_Cubic_2 =
        (a * pow(Hor_Wei + 1, 3) - 5 * a * pow(Hor_Wei + 1, 2) + 8 * a * (Hor_Wei + 1) - 4 * a) * Data[(int)Src_Y_Plus_1 * Stride + (int)Src_X_Minus_1]
        + ((a + 2) * pow(Hor_Wei, 3) - (a + 3) * pow(Hor_Wei, 2) + 1) * Data[(int)Src_Y_Plus_1 * Stride + (int)Src_X]
        + ((a + 2) * pow(1 - Hor_Wei, 3) - (a + 3) * pow(1 - Hor_Wei, 2) + 1) * Data[(int)Src_Y_Plus_1 * Stride + (int)Src_X_Plus_1]
        + (a * pow(2 - Hor_Wei, 3) - 5 * a * pow(2 - Hor_Wei, 2) + 8 * a * (2 - Hor_Wei) - 4 * a) * Data[(int)Src_Y_Plus_1 * Stride + (int)Src_X_Plus_2];

    double TMP_Cubic_3 =
        (a * pow(Hor_Wei + 1, 3) - 5 * a * pow(Hor_Wei + 1, 2) + 8 * a * (Hor_Wei + 1) - 4 * a) * Data[(int)Src_Y_Plus_2 * Stride + (int)Src_X_Minus_1]
        + ((a + 2) * pow(Hor_Wei, 3) - (a + 3) * pow(Hor_Wei, 2) + 1) * Data[(int)Src_Y_Plus_2 * Stride + (int)Src_X]
        + ((a + 2) * pow(1 - Hor_Wei, 3) - (a + 3) * pow(1 - Hor_Wei, 2) + 1) * Data[(int)Src_Y_Plus_2 * Stride + (int)Src_X_Plus_1]
        + (a * pow(2 - Hor_Wei, 3) - 5 * a * pow(2 - Hor_Wei, 2) + 8 * a * (2 - Hor_Wei) - 4 * a) * Data[(int)Src_Y_Plus_2 * Stride + (int)Src_X_Plus_2];

    UChar TMP =
        (a * pow(Ver_Wei + 1, 3) - 5 * a * pow(Ver_Wei + 1, 2) + 8 * a * (Ver_Wei + 1) - 4 * a) * TMP_Cubic_0
        + ((a + 2) * pow(Ver_Wei, 3) - (a + 3) * pow(Ver_Wei, 2) + 1) * TMP_Cubic_1
        + ((a + 2) * pow(1 - Ver_Wei, 3) - (a + 3) * pow(1 - Ver_Wei, 2) + 1) * TMP_Cubic_2
        + (a * pow(2 - Ver_Wei, 3) - 5 * a * pow(2 - Ver_Wei, 2) + 8 * a * (2 - Ver_Wei) - 4 * a) * TMP_Cubic_3
          + 0.5;
    UChar TMP_Cubic = CLIP(TMP);
    return TMP_Cubic;
}
```

- 0 미만, Hei, Wid 이상에 대한 Clipping 수행

  (Boundary mirroring)

- 거리 가중치 Hor_wei, Ver_wei 생성
- 16개 좌표를 이용한 가로 Interpolation 4번 수행

- 위에서 구한 4개 값을 이용한 세로 Interpolation 수행 후 반올림과 강제 형 변환
- Cubic Interpolation은 Clipping 작업이 필요하므로 수행

B-spline Interpolation

```
UChar Bspline(UChar* Data, double Src_X, double Src_Y, int Stride)
{
    int Src_X_Minus_1, Src_X_Plus_1, Src_X_Plus_2, Src_Y_Minus_1, Src_Y_Plus_1, Src_Y_Plus_2;
    double Hor_Wei, Ver_Wei; //Horizontal Weight, Vertical Weight
    Src_X_Minus_1 = CLIP_MINUS((int)Src_X - 1);
    Src_X_Plus_1 = CLIP_HOR((int)Src_X + 1);
    Src_X_Plus_2 = CLIP_HOR((int)Src_X + 2);
    Src_Y_Minus_1 = CLIP_MINUS((int)Src_Y - 1);
    Src_Y_Plus_1 = CLIP_VER((int)Src_Y + 1);
    Src_Y_Plus_2 = CLIP_VER((int)Src_Y + 2);
    Hor_Wei = Src_X - (int)Src_X;
    Ver_Wei = Src_Y - (int)Src_Y;
    double TMP_Bspline_0 =
        ((-1.0 / 6.0) * pow(Hor_Wei + 1, 3) + pow(Hor_Wei + 1, 2) - 2 * (Hor_Wei + 1) + (4.0 / 3.0)) * Data[(int)Src_Y_Minus_1 * Stride + (int)Src_X_Minus_1]
        + ((1.0 / 2.0) * pow(Hor_Wei, 3) - pow(Hor_Wei, 2) + (2.0 / 3.0)) * Data[(int)Src_Y_Minus_1 * Stride + (int)Src_X]
        + ((1.0 / 2.0) * pow(1 - Hor_Wei, 3) - pow(1 - Hor_Wei, 2) + (2.0 / 3.0)) * Data[(int)Src_Y_Minus_1 * Stride + (int)Src_X_Plus_1]
        + ((-1.0 / 6.0) * pow(2 - Hor_Wei, 3) + pow(2 - Hor_Wei, 2) - 2 * (2 - Hor_Wei) + (4.0 / 3.0)) * Data[(int)Src_Y_Minus_1 * Stride + (int)Src_X_Plus_2];

    double TMP_Bspline_1 =
        ((-1.0 / 6.0) * pow(Hor_Wei + 1, 3) + pow(Hor_Wei + 1, 2) - 2 * (Hor_Wei + 1) + (4.0 / 3.0)) * Data[(int)Src_Y * Stride + (int)Src_X_Minus_1]
        + ((1.0 / 2.0) * pow(Hor_Wei, 3) - pow(Hor_Wei, 2) + (2.0 / 3.0)) * Data[(int)Src_Y * Stride + (int)Src_X]
        + ((1.0 / 2.0) * pow(1 - Hor_Wei, 3) - pow(1 - Hor_Wei, 2) + (2.0 / 3.0)) * Data[(int)Src_Y * Stride + (int)Src_X_Plus_1]
        + ((-1.0 / 6.0) * pow(2 - Hor_Wei, 3) + pow(2 - Hor_Wei, 2) - 2 * (2 - Hor_Wei) + (4.0 / 3.0)) * Data[(int)Src_Y * Stride + (int)Src_X_Plus_2];

    double TMP_Bspline_2 =
        ((-1.0 / 6.0) * pow(Hor_Wei + 1, 3) + pow(Hor_Wei + 1, 2) - 2 * (Hor_Wei + 1) + (4.0 / 3.0)) * Data[(int)Src_Y_Plus_1 * Stride + (int)Src_X_Minus_1]
        + ((1.0 / 2.0) * pow(Hor_Wei, 3) - pow(Hor_Wei, 2) + (2.0 / 3.0)) * Data[(int)Src_Y_Plus_1 * Stride + (int)Src_X]
        + ((1.0 / 2.0) * pow(1 - Hor_Wei, 3) - pow(1 - Hor_Wei, 2) + (2.0 / 3.0)) * Data[(int)Src_Y_Plus_1 * Stride + (int)Src_X_Plus_1]
        + ((-1.0 / 6.0) * pow(2 - Hor_Wei, 3) + pow(2 - Hor_Wei, 2) - 2 * (2 - Hor_Wei) + (4.0 / 3.0)) * Data[(int)Src_Y_Plus_1 * Stride + (int)Src_X_Plus_2];

    double TMP_Bspline_3 =
        ((-1.0 / 6.0) * pow(Hor_Wei + 1, 3) + pow(Hor_Wei + 1, 2) - 2 * (Hor_Wei + 1) + (4.0 / 3.0)) * Data[(int)Src_Y_Plus_2 * Stride + (int)Src_X_Minus_1]
        + ((1.0 / 2.0) * pow(Hor_Wei, 3) - pow(Hor_Wei, 2) + (2.0 / 3.0)) * Data[(int)Src_Y_Plus_2 * Stride + (int)Src_X]
        + ((1.0 / 2.0) * pow(1 - Hor_Wei, 3) - pow(1 - Hor_Wei, 2) + (2.0 / 3.0)) * Data[(int)Src_Y_Plus_2 * Stride + (int)Src_X_Plus_1]
        + ((-1.0 / 6.0) * pow(2 - Hor_Wei, 3) + pow(2 - Hor_Wei, 2) - 2 * (2 - Hor_Wei) + (4.0 / 3.0)) * Data[(int)Src_Y_Plus_2 * Stride + (int)Src_X_Plus_2];

    UChar TMP_Bspline =
        ((-1.0 / 6.0) * pow(Ver_Wei + 1, 3) + pow(Ver_Wei + 1, 2) - 2 * (Ver_Wei + 1) + (4.0 / 3.0)) * TMP_Bspline_0
        + ((1.0 / 2.0) * pow(Ver_Wei, 3) - pow(Ver_Wei, 2) + (2.0 / 3.0)) * TMP_Bspline_1
        + ((1.0 / 2.0) * pow(1 - Ver_Wei, 3) - pow(1 - Ver_Wei, 2) + (2.0 / 3.0)) * TMP_Bspline_2
        + ((-1.0 / 6.0) * pow(2 - Ver_Wei, 3) + pow(2 - Ver_Wei, 2) - 2 * (2 - Ver_Wei) + (4.0 / 3.0)) * TMP_Bspline_3
        + 0.5;
    return TMP_Bspline;
}
```

- Cubic과 같은 구조, 다른 수식으로 Interpolation 수행

- B-spline Interpolation은 가중치의 합이 1이므로 Clipping X

Rotation

```
for (Angle = 0; Angle <= 360; Angle += 4)
{
    double Seta = PI / 180.0 * Angle;

    for (int i = 0; i < img->info.height; i++)
    {
        for (int j = 0; j < img->info.width; j++)
        {
            Basis_X = (j - Center_X) * cos(Seta) + (i - Center_Y) * sin(Seta) + Center_X;
            Basis_Y = (i - Center_Y) * cos(Seta) - (j - Center_X) * sin(Seta) + Center_Y;

            New_X = (int)Basis_X;
            New_Y = (int)Basis_Y;

            if (!(New_X < 0 || New_X >= img->info.width - 1 || New_Y < 0 || New_Y >= img->info.height - 1)) // 원시 화소가
            {
                img->Near_Ro[i * img->info.width + j] = NearesetNeighbor(img->Ori_img, Basis_X, Basis_Y, img->info.width);
                img->Bi_Ro[i * img->info.width + j] = Bilinear(img->Ori_img, Basis_X, Basis_Y, img->info.width);
                img->Cubic_Ro[i * img->info.width + j] = Cubic(img->Ori_img, Basis_X, Basis_Y, img->info.width);
                img->Bspline_Ro[i * img->info.width + j] = Bspline(img->Ori_img, Basis_X, Basis_Y, img->info.width);
            }
            else
            {
                img->Near_Ro[i * img->info.width + j] = 0;
                img->Bi_Ro[i * img->info.width + j] = 0;
                img->Cubic_Ro[i * img->info.width + j] = 0;
                img->Bspline_Ro[i * img->info.width + j] = 0;
            }
        }
    }
```

- 4도 씩 늘어나며 회전 하는 이미지 생성
- 중심점 기준 회전 식으로 변환 하였음 (Inverse mapping)
- 회전 후 픽셀이 원본 영상 크기 내이면 Interpolation 이용하여 구하기
- 밖이면 0 할당

Rotation 23도 (0,0)

```
int Center_X = 0, Center_Y = 0;                                                    //원점 기준 회전
double Angle = 23.0;

double Seta = PI / 180.0 * Angle;

for (int i = 0; i < img->info.height; i++)
{
    for (int j = 0; j < img->info.width; j++)
    {
        Basis_X = (j - Center_X) * cos(Seta) + (i - Center_Y) * sin(Seta) + Center_X;
        Basis_Y = (i - Center_Y) * cos(Seta) - (j - Center_X) * sin(Seta) + Center_Y;

        New_X = (int)Basis_X;
        New_Y = (int)Basis_Y;

        if (!(New_X < 0 || New_X >= img->info.width - 1 || New_Y < 0 || New_Y >= img->info.height - 1)) // 원시 화소가 영
        {
            img->Near_Ro[i * img->info.width + j] = NearesetNeighbor(img->Ori_img, Basis_X, Basis_Y, img->info.width);
            img->Bi_Ro[i * img->info.width + j] = Bilinear(img->Ori_img, Basis_X, Basis_Y, img->info.width);
            img->Cubic_Ro[i * img->info.width + j] = Cubic(img->Ori_img, Basis_X, Basis_Y, img->info.width);
            img->Bspline_Ro[i * img->info.width + j] = Bspline(img->Ori_img, Basis_X, Basis_Y, img->info.width);
        }
        else
        {
            img->Near_Ro[i * img->info.width + j] = 0;
            img->Bi_Ro[i * img->info.width + j] = 0;
            img->Cubic_Ro[i * img->info.width + j] = 0;
            img->Bspline_Ro[i * img->info.width + j] = 0;
        }
    }
}
```

- 원점 기준 회전이므로 좌표 설정
- 각도 23도로 고정

- 23도로 회전된 이미지 한 장만 출력하기 위해서 Rotation 함수에서 for문 수정함

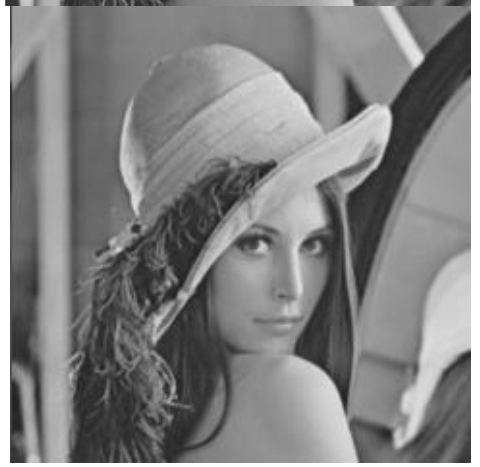
Rotation 23도 중심점

```
double Angle = 23.0;
int Center_X = img->info.width / 2, Center_Y = img->info.height / 2;                    //중심점 기준 회
double Seta = PI / 180.0 * Angle;

for (int i = 0; i < img->info.height; i++)
{
    for (int j = 0; j < img->info.width; j++)
    {
        Basis_X = (j - Center_X) * cos(Seta) + (i - Center_Y) * sin(Seta) + Center_X;
        Basis_Y = (i - Center_Y) * cos(Seta) - (j - Center_X) * sin(Seta) + Center_Y;

        New_X = (int)Basis_X;
        New_Y = (int)Basis_Y;

        if (!(New_X < 0 || New_X >= img->info.width - 1 || New_Y < 0 || New_Y >= img->info.height - 1)) // 원시 화소가 영
        {
            img->Near_Ro[i * img->info.width + j] = NearesetNeighbor(img->Ori_img, Basis_X, Basis_Y, img->info.width);
            img->Bi_Ro[i * img->info.width + j] = Bilinear(img->Ori_img, Basis_X, Basis_Y, img->info.width);
            img->Cubic_Ro[i * img->info.width + j] = Cubic(img->Ori_img, Basis_X, Basis_Y, img->info.width);
            img->Bspline_Ro[i * img->info.width + j] = Bspline(img->Ori_img, Basis_X, Basis_Y, img->info.width);
        }
        else
        {
            img->Near_Ro[i * img->info.width + j] = 0;
            img->Bi_Ro[i * img->info.width + j] = 0;
            img->Cubic_Ro[i * img->info.width + j] = 0;
            img->Bspline_Ro[i * img->info.width + j] = 0;
        }
    }
}
```

- 각도 23도로 고정
- 중심점 기준 회전이므로 좌표 설정

- 23도로 회전된 이미지 한 장만 출력하기 위해서 Rotation 함수에서 for문 수정함

Lena Scaling 결과 이미지

	Nearest	Bilinear	Cubic	B-spline
x2.32				
x0.47				

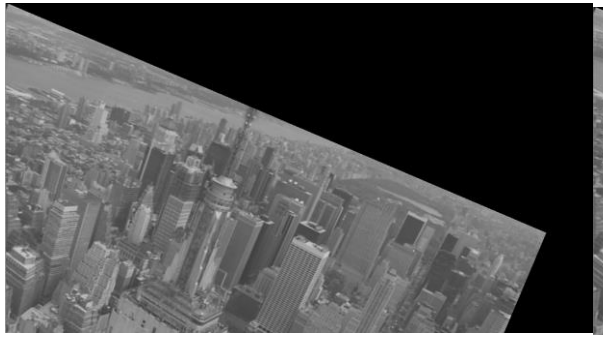
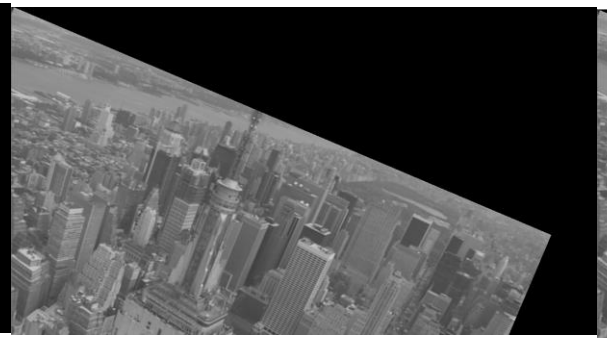
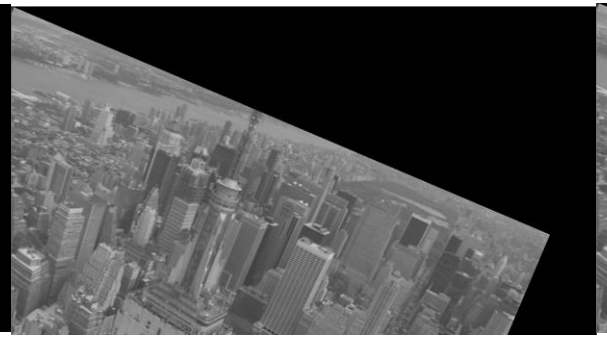
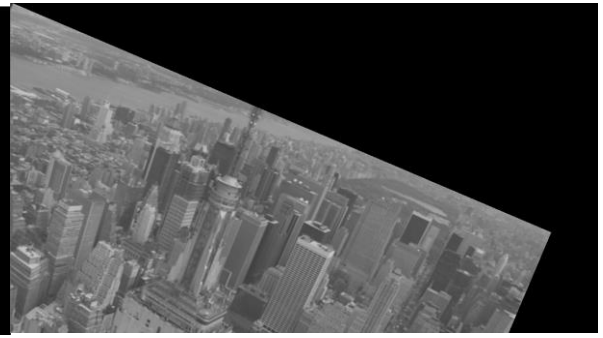
Lena Rotation 결과 이미지

	Nearest	Bilinear	Cubic	B-spline
(0,0)				
Center				

Lena Scaling 결과 이미지

	Nearest	Bilinear	Cubic	B-spline
x2.32				
x0.47				

Lena Rotation 결과 이미지

	Nearest	Bilinear	Cubic	B-spline
(0,0)				
Center