영상처리 HW4 15012974 최태호

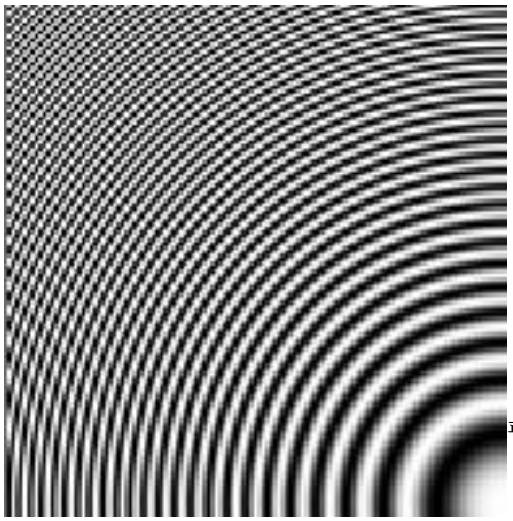
Lab06에서 제공된 code들을 활용하여 기하학적 처리기법에서 다룬 bilinear interpolation, cubic convolution interpolation을 구현하라.

원본 - bilinear - cubic 순서

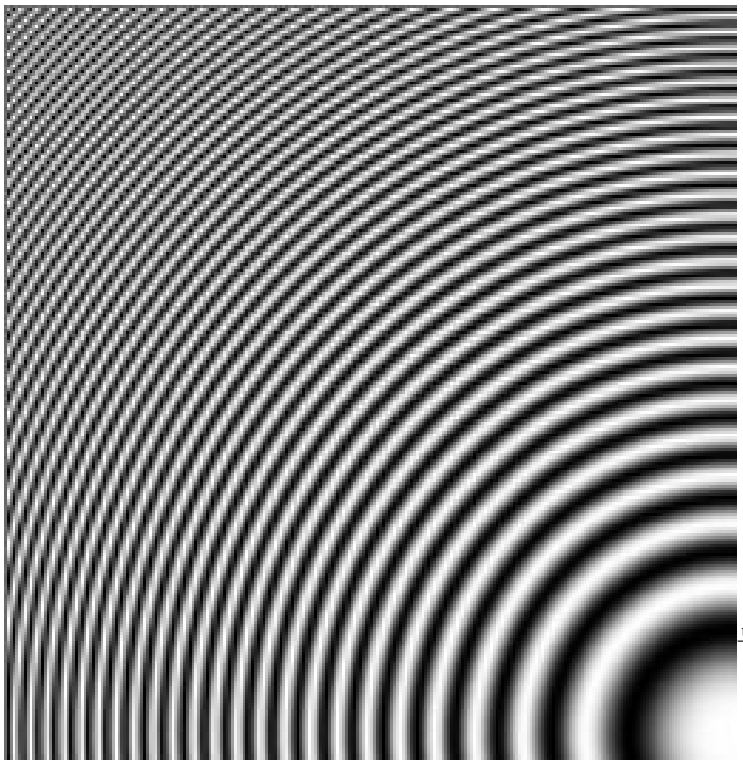
정확한 비교를 위해 lena영상을 3배로 키운후에 보간법을 적용하였습니다.

뒤로 갈수록 픽셀 경계선이 눈에띄게 사라지는 것을 볼 수 있습니다.

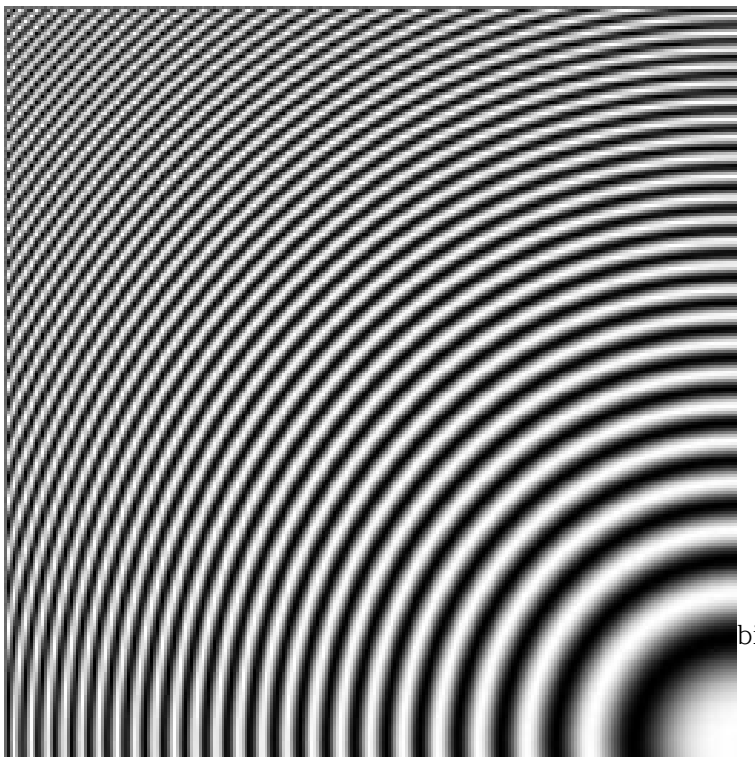
원본 - bilinear - cubic 순서



픽셀간의 경계가 눈에띄게 많다.



두배로 키웠음에도 원본보다 픽셀 경계선이 매우 줄어듦.



bilinear보다 매우 부드러워짐

bilinear_interpolation 함수

```
double bilinear_interpolation(image_ptr buffer,int x,int y,unsigned long X_Source, unsigned long Y_Source, int cols,
float x_scale, float y_scale)
{
    float NW, NE, SW, SE, EWweight, NSweight, EWbottom, EWtop;
    unsigned char dest_pixel;
    NW = buffer[Y_Source * cols + X_Source];
    NE = buffer[Y_Source * cols + X_Source + 1];
    SW = buffer[(Y_Source + 1) * cols + X_Source];
    SE = buffer[(Y_Source + 1) * cols + X_Source + 1];

    EWweight = (x / x_scale) - (float)X_Source;
    NSweight = (y / y_scale) - (float)Y_Source;

    //printf("%f %f\n", EWweight, NSweight);

    EWtop = NW + EWweight * (NE - NW);
    EWbottom = SW + EWweight * (SE - SW);

    dest_pixel = (char)EWtop + NSweight * (EWbottom - EWtop);
    return dest_pixel;
}
```

cubic_interpolation 함수

가중치 함수에대한 계수 a는 임의로 -0.5로 정하고 식을 전개하여 거리 d에대한 다항식으로 구현

```
double cubic_interpolation(double v1, double v2, double v3, double v4, double d)
{
    double v, p1, p2, p3, p4;

    p1 = 2 * v2;
    p2 = -v1 + v3;
    p3 = 2 * v1 - 5 * v2 + 4 * v3 - v4;
    p4 = -v1 + 3 * v2 - 3 * v3 + v4;

    v = (p1 + d * (p2 + d * (p3 + d * p4))) / 2.;
    return v;
}
```

$$v=\left\{-\frac{1}{2}(1+d)^3+\frac{5}{2}(1+d)^2-4(1+d)+2\right\}v_1$$

$$f(x)=\begin{cases}\frac{3}{2}|x|^3-\frac{5}{2}|x|^2+1 & 0\le|x|<1 \\ -\frac{1}{2}|x|^3+\frac{5}{2}|x|^2-4|x|+2 & 1\le|x|<2 \\ 0 & 2\le|x|\end{cases}$$

$$+\left\{\frac{3}{2}d^3-\frac{5}{2}d^2+1\right\}v_2$$

$$+\left\{\frac{3}{2}(1-d)^3-\frac{5}{2}(1-d)^2+1\right\}v_3$$

$$+\left\{-\frac{1}{2}(2-d)^3+\frac{5}{2}(2-d)^2-4(2-d)+2\right\}v_4$$

거리 d에 대한 다항식

$$v=\frac{1}{2}\left[(-v_1+3v_2-3v_3+v_4)d^3+(2v_1-5v_2+4v_3-v_4)d^2+(-v_1+v_3)d+2v_2\right]$$

```
for (y = 0; y < new_rows; y++)
    {
        index = 0;
        for (x = 0; x < new_cols; x++)
        {
            X_Source = (double)(cols-1)*x / (new_cols-1);
            Y_Source = (double)(rows-1)*y / (new_rows-1);
            source_index = Y_Source * cols + X_Source;


            //1행
            a2 = (int)X_Source;
            a1 = a2 - 1;
            if (a1 < 0) a1 = 0;
            a3 = a2 + 1;
            if (a3 >= cols) a3 = cols - 1;
            a4 = a2 + 2;
            if (a4 >= cols) a4 = cols - 1;
            p = (x /x_scale) - (float)X_Source;

            //2행
            b2 = (int)Y_Source;
            b1 = b2 - 1;
            if (b1 < 0) b1 = 0;
            b3 = b2+ 1;
            if (b3 >= rows) b3 = rows - 1;
            b4 = b2+ 2;
            if (b4 >= rows) b4 = rows - 1;
            q = (y /y_scale) - (float)Y_Source;

            //printf("%f\n", p);
            v1 = cubic_interpolation(buffer[b1 * cols + a1], buffer[b1 * cols + a2], buffer[b1 * cols + a3], buffer[b1
* cols + a4], p);
            v2 = cubic_interpolation(buffer[b2 * cols + a1], buffer[b2 * cols + a2], buffer[b2 * cols + a3], buffer[b2
* cols + a4], p);
            v3 = cubic_interpolation(buffer[b3 * cols + a1], buffer[b3 * cols + a2], buffer[b3 * cols + a3], buffer[b3
* cols + a4], p);
            v4 = cubic_interpolation(buffer[b4 * cols + a1], buffer[b4 * cols + a2], buffer[b4 * cols + a3], buffer[b4
* cols + a4], p);
            dest_pixel= cubic_interpolation(v1, v2, v3, v4, q);

            //dest_pixel = bilinear_interpolation(buffer, x, y, X_Source, Y_Source, cols, x_scale, y_scale);



            if (type == 5) {      /* PGM */
                //line_buff[index++] = buffer[source_index]; //Nearest
                line_buff[index++] = dest_pixel; //bilinear, cubic
```