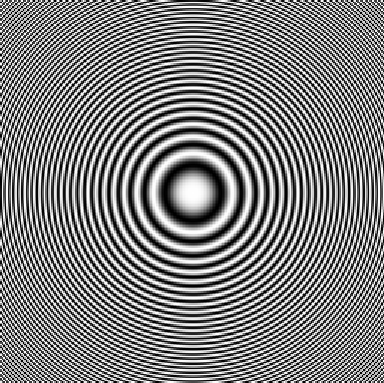
영상처리 HW 4

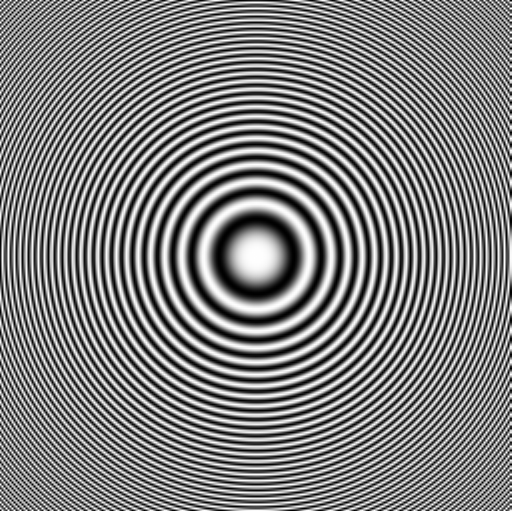
16010980 이우석

# 결과

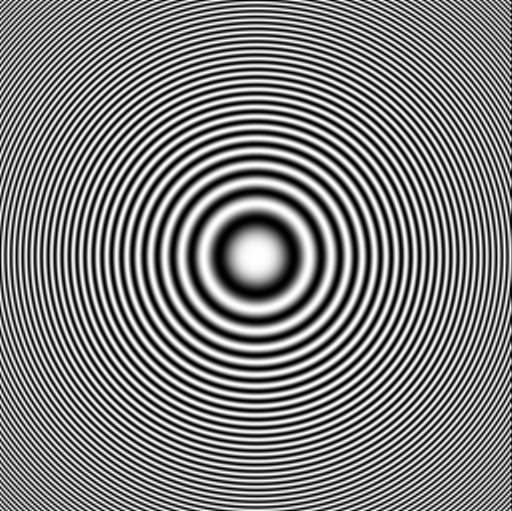
< 원본 이미지 - czp.pgm >



< 2.0 scale Bilinear – czp\_2.0scaleBili.pgm >



< 2.0 scale Cubic convolution – czp\_2.0scaleCubic.pgm >



< 원본 이미지 - lena.pgm >

사람, 여자, 의류, 모자이(가) 표시된 사진

자동 생성된 설명

< 2.0 scale Bilinear – lena\_2.0scaleBili.pgm >

사람, 여자, 실외, 머리장식이(가) 표시된 사진

자동 생성된 설명

< 2.0 scale Cubic convolution – lena\_2.0scaleCubic.pgm >

사람, 여자, 실외, 머리장식이(가) 표시된 사진

자동 생성된 설명

# 코드

< bilinear() >

void bilinear(image\_ptr buffer, char\* fileout, int rows, int cols,

float x\_scale, float y\_scale, int type)

{

unsigned long x, y; /\* loop indices for columns and rows \*/

unsigned long index; /\* index into line buffer \*/

unsigned long source\_index; /\* address of source pixel in image buffer \*/

unsigned char\* line\_buff; /\* output line buffer \*/

int new\_rows, new\_cols; /\* values of rows and columns for new image \*/

unsigned line; /\* number of pixels in one scan line \*/

FILE\* fp; /\* output file pointer \*/

unsigned long X\_Source, Y\_Source; /\* x and y address of source pixel \*/

pixel\_ptr color\_buff; /\* pointer to a color image in memory \*/

int NW, NE, SW, SE, result;

float EWweight, NSweight, EWtop, EWbottom;

/\* open new output file \*/

if ((fp = fopen(fileout, "wb")) == NULL)

{

printf("Unable to open %s for output\n", fileout);

exit(1);

}

new\_cols = cols \* x\_scale;

new\_rows = rows \* y\_scale;

/\* print out the portable bitmap header \*/

fprintf(fp, "P%d\n%d %d\n255\n", type, new\_cols, new\_rows);

if (type == 5) /\* PGM file \*/

line = new\_cols;

else /\* PPM file \*/

{

line = new\_cols \* 3;

color\_buff = (pixel\_ptr)buffer;

}

line\_buff = (unsigned char\*)malloc(line);

for (y = 0; y < new\_rows; y++)

{

index = 0;

for (x = 0; x < new\_cols; x++)

{

X\_Source = (unsigned long)(x / x\_scale);

Y\_Source = (unsigned long)(y / y\_scale);

if (type == 5) /\* PGM \*/

{

/\* bilinear interpolation \*/

// duplicate the border edges

NW = buffer[Y\_Source \* cols + X\_Source];

if (X\_Source + 1 >= cols && Y\_Source + 1 >= rows)

{

NE = NW;

SW = NW;

SE = NW;

}

else if (X\_Source + 1 >= cols)

{

NE = NW;

SW = buffer[(Y\_Source + 1) \* cols + X\_Source];

SE = SW;

}

else if (Y\_Source + 1 >= rows)

{

NE = buffer[Y\_Source \* cols + X\_Source + 1];

SW = NW;

SE = NE;

}

else

{

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;

EWtop = NW + EWweight \* (NE - NW);

EWbottom = SW + EWweight \* (SE - SW);

result = EWtop + NSweight \* (EWbottom - EWtop);

line\_buff[index++] = result;

}

else /\* PPM \*/

{

line\_buff[index++] = color\_buff[source\_index].r;

line\_buff[index++] = color\_buff[source\_index].g;

line\_buff[index++] = color\_buff[source\_index].b;

}

}

fwrite(line\_buff, 1, line, fp);

}

fclose(fp);

}

< cubic\_convolution() >

void cubic\_convolution(image\_ptr buffer, char\* fileout, int rows, int cols,

float x\_scale, float y\_scale, int type)

{

unsigned long x, y; /\* loop indices for columns and rows \*/

unsigned long index; /\* index into line buffer \*/

unsigned long source\_index; /\* address of source pixel in image buffer \*/

unsigned char\* line\_buff; /\* output line buffer \*/

int new\_rows, new\_cols; /\* values of rows and columns for new image \*/

unsigned line; /\* number of pixels in one scan line \*/

FILE\* fp; /\* output file pointer \*/

unsigned long X\_Source, Y\_Source; /\* x and y address of source pixel \*/

pixel\_ptr color\_buff; /\* pointer to a color image in memory \*/

int cubic\_arr[4][4], i, j;

int arr\_x, arr\_y;

float cubic\_EWweight[4], cubic\_NSweight[4], cubic\_intermResult[4], cubic\_result, dif\_tmp, a = -0.5f; // a (계수) 의 값 : -0.5f

/\* open new output file \*/

if ((fp = fopen(fileout, "wb")) == NULL)

{

printf("Unable to open %s for output\n", fileout);

exit(1);

}

new\_cols = cols \* x\_scale;

new\_rows = rows \* y\_scale;

/\* print out the portable bitmap header \*/

fprintf(fp, "P%d\n%d %d\n255\n", type, new\_cols, new\_rows);

if (type == 5) /\* PGM file \*/

line = new\_cols;

else /\* PPM file \*/

{

line = new\_cols \* 3;

color\_buff = (pixel\_ptr)buffer;

}

line\_buff = (unsigned char\*)malloc(line);

for (y = 0; y < new\_rows; y++)

{

index = 0;

for (x = 0; x < new\_cols; x++)

{

X\_Source = (unsigned long)(x / x\_scale);

Y\_Source = (unsigned long)(y / y\_scale);

if (type == 5) /\* PGM \*/

{

/\* cubic convolution interpolation \*/

// 배열에 픽셀 값을 배치.

arr\_y = Y\_Source - 1;

for (i = 0; i < 4; i++)

{

arr\_x = X\_Source - 1;

for (j = 0; j < 4; j++)

{

// zero padding

if (arr\_y < 0 || arr\_x < 0 || arr\_y >= rows || arr\_x >= cols)

cubic\_arr[i][j] = 0;

else

cubic\_arr[i][j] = buffer[arr\_y \* cols + arr\_x];

arr\_x++;

}

arr\_y++;

}

// 가로 가중치 계산.

arr\_x = X\_Source - 1;

for (i = 0; i < 4; i++)

{

dif\_tmp = fabsf((x / x\_scale) - arr\_x);

if (0 <= dif\_tmp && dif\_tmp < 1)

cubic\_EWweight[i] = (a + 2) \* pow(dif\_tmp, 3) - (a + 3) \* pow(dif\_tmp, 2) + 1;

else if (1 <= dif\_tmp && dif\_tmp < 2)

cubic\_EWweight[i] = a \* pow(dif\_tmp, 3) - 5 \* a \* pow(dif\_tmp, 2) + 8 \* a \* dif\_tmp - 4 \* a;

else

cubic\_EWweight[i] = 0;

arr\_x++;

}

// 세로 가중치 계산.

arr\_y = Y\_Source - 1;

for (i = 0; i < 4; i++)

{

dif\_tmp = fabsf((y / y\_scale - arr\_y));

if (0 <= dif\_tmp && dif\_tmp < 1)

cubic\_NSweight[i] = (a + 2) \* pow(dif\_tmp, 3) - (a + 3) \* pow(dif\_tmp, 2) + 1;

else if (1 <= dif\_tmp && dif\_tmp < 2)

cubic\_NSweight[i] = a \* pow(dif\_tmp, 3) - 5 \* a \* pow(dif\_tmp, 2) + 8 \* a \* dif\_tmp - 4 \* a;

else

cubic\_NSweight[i] = 0;

arr\_y++;

}

// 가로 가중치를 이용하여 중간 결과값 계산.

for (i = 0; i < 4; i++)

{

cubic\_intermResult[i] = 0;

for (j = 0; j < 4; j++)

{

cubic\_intermResult[i] += cubic\_EWweight[j] \* cubic\_arr[i][j];

}

}

// 세로 가중치를 이용하여 최종 결과값 계산.

cubic\_result = 0;

for (i = 0; i < 4; i++)

{

cubic\_result += cubic\_NSweight[i] \* cubic\_intermResult[i];

}

// float 형식의 특성 상 계산 상의 약간의 오차가 발생하는데 이 때문에 오버플로우, 언더플로우가 발생.

// 이를 CLIP 함수를 이용하여 조정.

CLIP(cubic\_result, 0, 255);

line\_buff[index++] = cubic\_result;

}

else /\* PPM \*/

{

line\_buff[index++] = color\_buff[source\_index].r;

line\_buff[index++] = color\_buff[source\_index].g;

line\_buff[index++] = color\_buff[source\_index].b;

}

}

fwrite(line\_buff, 1, line, fp);

}

fclose(fp);

}