수치해석

HW02-Interpolation

신윤석

2019202053

컴퓨터정보공학부

Introduction

이번 과제에서는 interpolation을 해서 사진의 해상도를 늘려본다. 사용하는 interpolation 방법은 Nearest Neighbor Interpolation, Bilinear Interpolation, Bicubic Interpolation, Six-tab Linear Interpolation이다.

Interpolation Method

Nearest Neighbor Interpolation은 주변의 픽셀을 그대로 가져다 붙이는 알고리즘이다. 이렇게 하면 픽셀 하나하나가 그대로 커진 느낌이라 마치 모자이크를 보는 느낌을 준다. 이번 과제에 적용하는 방식은 128*128 픽셀의 사진을 512*512 픽셀로 키우는 것이기 때문에 512*512픽셀의 사진을 4*4짜리 칸으로 나눠서 각 칸에 맞게 128*128사진의 픽셀 값으로 채우는 것이다. 반복문 4개를 중첩해서 해결할 수 있다.

```
//Nearest Neighbor Interpolation
for (int i = 0; i < 4; i++)
        for (int j = 0; j < 4; j++)
                for (int row = 0; row < 128; row++)
                        for (int col = 0; col < 128; col++)
                                procImg[(row << 2) + i][(col << 2) + j] = oriImg[row][col];
```

다음으로는 Bilinear Interpolation이다. linear라는 이름 답게 선형 함수로 interpolation을 진행한다. 각 축(row축과 column축)으로 1차원 interpolation을 하기 때문에 Bi-라는 접두사가 붙어 bilinear Interpolation이라고 한다. 이번 과제에서는 128*128이었던 사진을 4배 확대하여 512*512 사이즈로 확대하는 것이기 때문에 0번 칸과 4번 칸에 128*128사이즈 사진의 데이터가 있다고 한다면, 2번 칸에는 0번 칸의 데이터와 4번 칸 데이터의 평균값이 필요하고, 1번 칸에는 0번 칸의 데이터를 3배, 4번 칸의 데이터를 1배로 한 가중평균이 필요하며 3번 칸에는 반대로 0번 칸의 데이터를 1배, 4번 칸의 데이터를 3배로 한 가중평균이 필요하다. 이런 방법을 사용한다면 선형함수를 근사할 필요 없이 바로 linear interpolation이 가능해진다.

Bicubic Interpolation은 역시 각 축마다 interpolation을 하기 때문에 bi-라는 접두사가 붙은 형태이고, cubic이라는 이름 그대로 3차함수로 interpolation을 진행하는 것이다. 3차 함수를 근사해야 하기 때문에 점 4개가 필요하다. 예를 들어 4번 칸과 8번 칸 사이의 데이터를 채우기 위해 0번 칸과 4번 칸과 8번 칸과 12번 칸의 데이터를 사용해야 하는 것이다. 0,4,8,12번 칸의 data를 이용하여 Lagrange 다항식을 이용하여 3차 함수로 만들고 다시 원하는 칸의 index를 넣어서 원하는 칸의 data를 추출하는 방법을 사용한다.

Six-tap interpolation은 신호처리 관점에서 사진을 interpolation한다. 점 6개짜리 filter와 사진의 1d 데이터를 convolution하여 픽셀 하나의 데이터를 얻는다. 이때, filter가 sinc 함수를 샘플링한 점 6개로 구성이 되어 있다면, 점 6개의 중점의 데이터를 구할 수 있는 것이다. 이 알고리즘은 6

개 점 중 중점의 값을 취하는 방법이기 때문에 조금 복잡한 알고리즘이 필요하다. 다음과 같은 방법을 통해 interpolation을 진행할 수 있다.

```
Integer row, col
Mat Original, Processed
Set Processed with original data at correct position.
(Processed[4n+p][4m+q] = Original[n][m], p and q are phase)
// interpolate 4n row
For row from 0+p to 512+p increasing 4 each step:
   For col from 0+q to 512+q increasing 4 each step:
      Processed[row][col+11] = inner product(six-tap filter, Original[row/4][col/4:(col+20)/4]
// interpolate 2m column
For col from 0+q to 512+q increasing 4 each step:
   For row from 0+p to 512+p increasing 4 each step:
      Processed[row+11][col] = inner product(six-tap filter, Original[row/4:(row+20)/4][col]
// interpolate 4n+2 row
For row from 2+p to 512+p increasing 4 each step:
   For col from 0+q to 512+q increasing 4 each step:
      Processed[row][col+11] = inner product(six-tap filter, Original[row/4][col/4:(col+20)/4]
Bilinear interpolate left pixels.
```

위 방법대로 한다면 2n위치에 존재하는 pixel은 six-tap filter에 의해 interpolation되나, 나머지 픽셀은 Bilinear interpolate가 되기 때문에 그냥 양 옆에 존재하는 data의 평균을 넣으면 된다. 1d 백터의 임의의 점(n)에 대한 평균을 생각해 본다면,

$$P_n = \frac{P_{n-1} + P_{n+1}}{2}$$

$$P_{n-1} = \frac{P_{n-11} - 5P_{n-7} + 20P_{n-3} + 20P_{n+1} - 5P_{n+5} + P_{n+9}}{32}$$

$$P_{n+1} = \frac{P_{n+11} - 5P_{n+7} + 20P_{n+3} + 20P_{n-1} - 5P_{n-5} + P_{n-9}}{32}$$

$$64P_n = P_{n-11} + P_{n-9} - 5P_{n-7} - 5P_{n-5} + 20P_{n-3} + 20P_{n-1} + 20P_{n+1} + 20P_{n+3} - 5P_{n+5} - 5P_{n+7} + P_{n+9} + P_{n+11}$$

$$filter = \left\{\frac{1}{64}, \frac{1}{64}, \frac{-5}{64}, \frac{-5}{64}, \frac{20}{64}, \frac{20}{64}, \frac{20}{64}, \frac{20}{64}, \frac{-5}{64}, \frac{-5}{64}, \frac{1}{64}, \frac{1}{64}\right\}$$

위와 같은 결과를 얻을 수 있다. 사실 신호처리 관점에서 보면 sinc함수가 아니어 보이긴 하는데, 두 개씩 때서 보면 어차피 곱하고 더한다고 생각할 때, 다음과 같이 생각해 볼 수 있다.

$$P_n = \frac{1}{2}\left(\frac{P_{n-11}}{64} + \frac{P_{n-9}}{64}\right) - 5 \cdot \frac{1}{2}\left(\frac{P_{n-7}}{64} + \frac{P_{n-5}}{64}\right) + 20 \cdot \frac{1}{2}\left(\frac{P_{n-3}}{64} + \frac{P_{n-1}}{64}\right) + 20 \cdot \frac{1}{2}\left(\frac{P_{n+1}}{64} + \frac{P_{n+3}}{64}\right) - 5 \cdot \frac{1}{2}\left(\frac{P_{n+5}}{64} + \frac{P_{n+7}}{64}\right) + \frac{1}{2}\left(\frac{P_{n-11}}{64} + \frac{P_{n-9}}{64}\right)$$

결론적으로, 단순 Bilinear interpolation을 취하면 각 점에 대해서 Bilinear interpolation을 취해 six-tap convolution을 한 결과와 동일하게 처리됨을 알 수 있다.

Experiments

Nearest Neighbor Interpolation은 간단하게 인근 pixel을 대표 값으로 도배하면 된다. Phase로 생각한다면, 반복문을 이용해서 각 Phase에 해당하는 pixel에 대표 값을 넣는 것을 반복하면 쉽게 구현이 가능하다.

Bilinear, Bicubic interpolation을 해결하기 위해서는 두 점 혹은 네 점을 지나는 다항식을 구해야 한다. 사실 임의의 미지수로 계수를 상정하고 방정식에 점의 좌표를 넣어서 연립방정식을 해결하여 계수를 구하는 방법도 있기는 하지만 컴퓨터한테 이런 방법을 시키기엔 경우의 수가 너무 많아지기 때문에 조금 더 체계적인 방법으로 다항식을 구하는 방법을 사용한다.

Lagrange polynomial, 라그랑주 다항식은 특정 점을 지나는 다항식을 만들기에 적합한 다항식이다. 점 n개를 지나는 k번째 라그랑주 다항식은 다음과 같다:

$$L_{n,k} = \frac{\overbrace{(x-x_1)(x-x_2)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_{n-1})(x-x_n)}^{(x-x_k)\text{항을 제외한 } n-1\text{개 항의 곱}}}{\underbrace{(x_k-x_1)(x_k-x_2)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_{n-1})(x_k-x_n)}_{(x_k-x_k)\text{항을 제외한 } n-1\text{개 항의 곱}}}$$

이 다항식을 가지고 n개 점 $(x_k, f(x_k))$를 지나는 다항식은 다음과 같이 나타낼 수 있다:

$$f(x) = \sum_{k=1}^{n} L_{n,k} \cdot f(x_k)$$

이 함수를 이용하여 Bilinear interpolation과 Bicubic interpolation을 진행한다.

사실 Bilinear interpolation은 직관적으로 보면 선형함수로 interpolation하는 것이기 때문에 위의 Interpolation Method 항목에서도 설명했듯이 중점위치는 평균을, 1,3번위치는 가중평균을 이용해서 구한다. 이를 Lagrange polynomial을 이용한 수식으로 나타내면 다음과 같다. 우선 0번 위치와 4번 위치에 sampling된 값이 있다고 하고 1,2,3번 위치의 값을 구하는 상황이라고 해 보자.

$$\begin{aligned}
f(x) &= \sum_{k=1}^{2} L_{2,k} \cdot f(x_k) \\
&= \frac{x-4}{0-4} f(0) + \frac{x}{4} f(4) \\
f(2) &= \frac{2-4}{0-4} f(0) + \frac{2}{4} f(4) = \frac{1}{2} f(0) + \frac{1}{2} f(4) \\
f(1) &= \frac{1-4}{0-4} f(0) + \frac{1}{4} f(4) = \frac{3}{4} f(0) + \frac{1}{4} f(4) \\
f(3) &= \frac{3-4}{0-4} f(0) + \frac{3}{4} f(4) = \frac{1}{4} f(0) + \frac{3}{4} f(4)
\end{aligned}$$

라그랑주 다항식을 이용해서 해당 위치의 값을 구하려 하면 가중치가 직관과 동일하게 나오게 된다.

Bicubic의 경우는 계산이 복잡해진다. 상황은 위와 같이 0,4,8,12위치에 sampled data가 있다고 가정하고 주어진 점 4개를 지나는 방정식만 구해보면 다음과 같다:

$$f(x)=\sum_{k=1}^{4} L_{4,k}\cdot f(x_k)$$
$$=\frac{(x-4)(x-8)(x-12)}{(0-4)(0-8)(0-12)}f(0)+\frac{x(x-8)(x-12)}{4(4-8)(4-12)}f(4)+\frac{x(x-4)(x-12)}{8(8-4)(8-12)}f(8)$$
$$+\frac{x(x-4)(x-8)}{12(12-4)(12-8)}f(12)$$
$$=-\frac{x^3-24x^2+176x-384}{384}f(0)+\frac{x^3-20\,x^2+96\,x}{128}f(4)-\frac{x^3-16\,x^2+48\,x}{128}f(8)$$
$$+\frac{x^3-12\,x^2+32\,x}{384}f(12)$$
$$=\left(-\frac{x^3}{384}+\frac{x^2}{16}-\frac{11x}{24}+1\right)f(0)+\left(\frac{x^3}{128}-\frac{5\,x^2}{32}+\frac{3\,x}{4}\right)f(4)+\left(-\frac{x^3}{128}+\frac{x^2}{8}-\frac{3\,x}{8}\right)f(8)$$
$$+\left(\frac{x^3}{384}-\frac{x^2}{32}+\frac{x}{12}\right)f(12)$$

위 식에서 공통되는 연산을 묶어서 행렬로 표현하면 다음과 같아진다:

$$=\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}\begin{bmatrix}-\frac{x^3}{384} & \frac{x^3}{128} & -\frac{x^3}{128} & \frac{x^3}{384}\\ \frac{x^2}{16} & -\frac{5\,x^2}{32} & \frac{x^2}{8} & -\frac{x^2}{32}\\ -\frac{11x}{24} & \frac{3\,x}{4} & -\frac{3\,x}{8} & \frac{x}{12}\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}f(0)\\ f(4)\\ f(8)\\ f(12)\end{bmatrix}$$

$$=\begin{bmatrix}1 & 1 & 1 & 1\end{bmatrix}\begin{bmatrix}x^3 & 0 & 0 & 0\\ 0 & x^2 & 0 & 0\\ 0 & 0 & x & 0\\ 0 & 0 & 0 & 1\end{bmatrix}\begin{bmatrix}-\frac{1}{384} & \frac{1}{128} & -\frac{1}{128} & \frac{1}{384}\\ \frac{1}{16} & -\frac{5}{32} & \frac{1}{8} & -\frac{1}{32}\\ -\frac{11}{24} & \frac{3}{4} & -\frac{3}{8} & \frac{1}{12}\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}f(0)\\ f(4)\\ f(8)\\ f(12)\end{bmatrix}$$

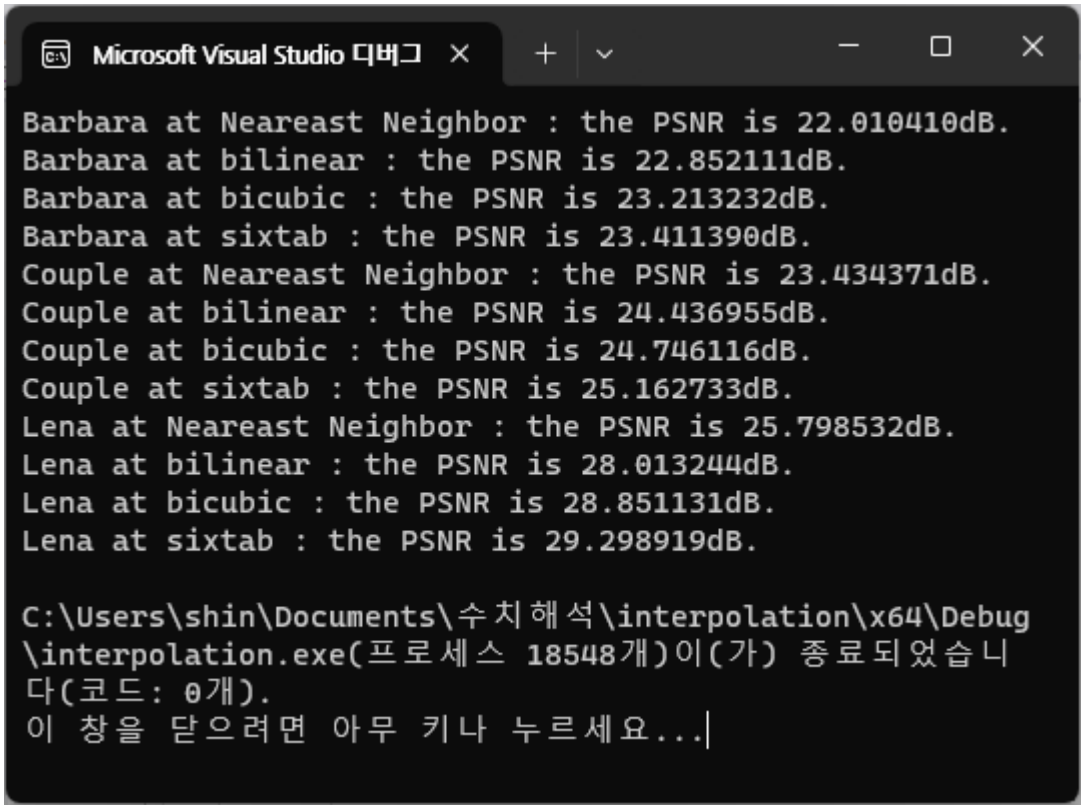
$$=\begin{bmatrix}x^3 & x^2 & x & 1\end{bmatrix}\begin{bmatrix}-\frac{1}{384} & \frac{1}{128} & -\frac{1}{128} & \frac{1}{384}\\ \frac{1}{16} & -\frac{5}{32} & \frac{1}{8} & -\frac{1}{32}\\ -\frac{11}{24} & \frac{3}{4} & -\frac{3}{8} & \frac{1}{12}\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}f(0)\\ f(4)\\ f(8)\\ f(12)\end{bmatrix}$$

이때 여러 점에 대한 값을 한 번에 찾고 싶으면,

$$\begin{bmatrix}x_1^3 & x_1^2 & x_1 & 1\\ x_2^3 & x_2^2 & x_2 & 1\\ \vdots & \vdots & \vdots & \vdots\\ x_n^3 & x_n^2 & x_n & 1\end{bmatrix}\begin{bmatrix}-\frac{1}{384} & \frac{1}{128} & -\frac{1}{128} & \frac{1}{384}\\ \frac{1}{16} & -\frac{5}{32} & \frac{1}{8} & -\frac{1}{32}\\ -\frac{11}{24} & \frac{3}{4} & -\frac{3}{8} & \frac{1}{12}\\ 1 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}f(0)\\ f(4)\\ f(8)\\ f(12)\end{bmatrix}$$

위 식을 이용한다면 한 함수에 대해 중복된 계산 없이 최소한의 계산으로 각 x에 맞는 값을 가진 열행렬을 얻을 수 있다.

마지막으로 six-tab Interpolation은 신호처리 관점에서 바라본 interpolation이다. Sinc 함수와 원 신호를 convolution하여 더욱 촘촘하게 sampling된 집합을 찾는 방법이다.
다음은 프로그램 실행 결과이다.



다음은 연산 결과 사진이다. Nearest Neighbor, Bilinear, Bicubic, Six-Tab interpolation을 진행한 순서대로 영상을 나열한다. 이미지가 크기가 커서 한 장에 하나씩만 나온다.

- Barbara

Nearest Neighbor

Bilinear

Bicubic

Six-Tab

- Couple

Nearest Neighbor

Bilinear

Bicubic

Six-tab

- Lena

Nearest Neighbor

Bilinear

Bicubic

Six-tab

Conclusion

이번 과제를 하면서 고려한 사항이 두가지 있다. Padding에 관한 부분과 연산 정확성에 대한 부분이다.

우선 Padding에 관한 이야기부터 하자면 Nearest Neighbor hood는 Padding을 신경 쓸 필요가 없다. Bilinear의 경우에는 Padding 선택지가 세가지 존재한다. 0으로 padding하거나, 가장자리와 같은 값으로 Padding하거나, 맨 끝의 data에 바로 옆에 인접한 data의 차를 뺀 값을 넣어주거나. 맨 마지막 선택지는 직관적으로 설명하자면 맨 끝 점과 바로 붙어있는 점 두개를 이용해서 직선을 그어서 Padding하겠다는 것이다. 결과부터 이야기하자면, 마지막 선택지가 가장 PSNR이 높았고, 두번째 선택지가 근소하지만 약간 낮았고(PSNR 소수점 2째짜리까지 같았다.) 첫번째 선택지가

눈에 띄게 낮았다. 우선 차이가 근소하기에 이번 과제 제출로는 두번째 선택지로 골랐다. 그 이유는 연산 정확성을 보장하기 위해 세번째 옵션은 더 많은 연산을 해야 하기 때문이다. 그리고 0 padding을 하게 되면 가장자리가 어두워지는 문제가 있다.

Bicubic와 Six-tab interpolation은 선택지가 두가지 있다. 거울에 반사시킨 것처럼 대칭되게 padding을 하는 것과 같은 값으로 padding하는 것이다. 이 역시도 근소하게 대칭되게 padding을 한 쪽이 PSNR이 더 높았다. 따라서 대칭 padding을 선택하여 과제로 제출한다. 이렇게 되는 원인에 대해 생각해 보자면 511번째 원소를 얻는 과정을 생각해 보면 된다. 511번째 원소를 얻기 위해서는 bicubic 관점에서 바라보면 대칭이면 509번이 양 옆에 있고 509값 너머로 505값이 양쪽에 있는 것과 양 옆에 509의 값과 그 너머로 한쪽은 505번값, 한쪽은 509번 값이 있다고 하면 전자는 안정적으로 511번 값이 의미가 있다고 할 수 있는데, 후자는 505에서 509,509,509 값이 나오니까 함수가 의미없이 요동치는 와중에 값을 구한 것이 되는 셈이다. Six-tab의 관점에서는 전자의 경우에 필터도 대칭, 입력 값도 대칭이므로 안정적으로 값을 잘 구할 것 같지만 후자의 경우에는 입력 값이 비대칭이고 심지어 한쪽은 DC성분만 남는 문제가 있다.

결론적으로 Padding의 문제는 없는 정보를 예측해서 만들어 내야 하기 때문에 벌어지는 문제이다. 이를 해결하는 근본적인 방법은 대충 0 padding해서 일단 interpolation을 한 후 원하는 사이즈로 다시 가장자리를 잘라내는 방법이 가장 정확한 interpolation 방법일 것이다. 심리적으로 봤을 때도 사진이나 영상을 찍을 때, 보여주고 싶은 것을 가운데에 놓지 가장자리에 잘 놓지는 않기 때문에 의미 있는 방법일 것이다.

또 한가지 고려해야 할 사항은 연산의 정확성이다. 연산 과정에서 1이하의 소수를 곱해야 하기도 하고 여러 개의 값을 더하기도 한다. 두가지 문제가 생긴다. 첫번째는 소수를 다루기 때문에 소수점에 대한 정확도 문제가 있고 두번째는 여러 수를 더하기 때문에 연산 결과가 0~255를 초과해버리는 문제가 있다. 연산 범위 문제는 경우를 따져서 범위를 초과할 시 0이나 255로 맞춰주는 clipping을 시행하면 된다.

Clipping이 필요한 경우는 bicubic interpolation이다. Bilinear 보간법의 경우에는 padding 방법에 따라 여부가 달라지는데, 0 padding이나 같은 값으로 padding을 하게 되면 선형함수 특성상 bound를 보장받기에 범위를 초과하는 일이 없다. 하지만 맨 끝 값에 맨 끝 값에서 바로 옆 값을 뺀 값을 뺀 padding을 하게 되면 이야기가 달라진다. 예를 들어 맨 끝 값이 0이고 인접한 값이 255였다면, padding하는 값은 -255가 되며, 이 경우 interpolation을 하게 되면 무조건 음수가 나온다. 범위를 초과한다. 마찬가지로 bicubic도 다항함수가 범위를 보장해 주지 않는다. 511번 원소의 값을 구하려고 하는데, 509번 원소의 값이 0이고 505번 원소의 값이 255라면, padding을 대칭으로 하면 이 역시도 무조건 함수의 값이 음수가 나오게 되고 같은 값으로 padding을 했다 쳐도 역시 무조건적으로 함수 값이 음수가 나오게 된다. 범위를 초과한다. 따라서 bilinear의 경우에는 정확성이 다소 떨어지더라도 계산의 편의와 계산 속도를 위해 같은 값을 padding하는 것으로 선택을 했고, bicubic의 경우에는 clipping을 계산 결과에 적용한다. Nearest Neighbor 보간법의 경우에는 data를 변형할 필요가 없고, phase만 잘 맞추면 padding을 할 필요가 없다. Six-tab 보간법의

경우에도 filter의 총 합이 1이므로 convolution구조 상 입력이 0~255의 bound가 지켜지면 출력도 0~255의 bound가 지켜지므로 clipping이 필요가 없다. 물론 계산 과정 중간에는 bound를 넘길 가능성이 있긴 하므로 자료형을 char보다 큰 것을 선택할 필요가 있다.

Nearest Neighbor 보간법을 제외한 세 보간법은 특성상 분수 가중치가 붙기 때문에 소수 연산이 불가피 해진다. Six-tab, bilinear 보간의 경우에는 나누는 수가 2,4,32 등 2의 제곱수라 그냥 정수로 연산하고 bit shift로 해결했다. 어차피 필요한 결과값은 정수이기 때문이다. 문제는 bicubic이다. 연산 행렬을 보면 알 수 있겠지만 384를 나눠야 한다. 이게 128*3이라서 나누기가 불가피해진다. 그래서 정확성을 생각해 볼까 한다. Double형 변수는 1개의 sign bit와 11개 bit의 지수와 52개 bit의 fraction을 갖는다. 유효 숫자가 2진수로 53개라는 의미이다. 그런데 bicubic interpolation을 하기 위해 만들어진 다항식을 생각해 보면 x = 0,4,8,12에서 범위 0~255가 보장받는다. 보간 함수 또한 그 범위 근처에 있을 가능성이 높다. 보간 함수가 가장 큰 값을 갖기 위해서는 (0,0),(4,255),(8,255),(12,0)을 지나는 다항함수를 생각해 볼 수 있다. 이는 이차함수로, 6에서 최대값을 가질 것이다. 최대값은 2295/8로 286.875이다. 대략적으로 정수부를 표현하기 위해 9비트만 있으면 표현이 가능하다. 가장 큰 값이 나올 때도 52bit fraction도 8bit가 정수부를 위해 쓰이고 나머지 44bit는 소수부 표현을 위해 쓰인다. 아주 확률이 없지는 않겠지만, 소수부 round error로 1단위까지 에러가 나는 경우는 거의 없다고 볼 수 있다. 그래서 bicubic은 그냥 double형 변수를 사용하였다. 그리고 0~255라는 표현 자체도 연속인 광량을 양자화 한 것이기 때문에 1정도의 오차가 있어도 눈으로 보는 데는 크게 이상이 없을 것이라고 생각한다.

반올림에 대해서도 생각해 볼 것이 있다. 강의자료를 보면 six-tab interpolation에서 filter와 data를 곱한 값에 32를 더라하고 되어 있는데, 32를 더하면 실제로 PSNR이 올라간다. 그래서 32말고 여러가지 값을 해 보았는데, 비등비등하지만 47과 31도 PSNR이 높게 잡힌다. 2학년 어셈블리 시간에 float를 다룰 일이 있어서 2진수의 반올림을 배운 적이 있는데, 반올림했을 때, 반올림했을 때 1의 자리가 되는 bit를 L, 그 바로 하위 비트를 G, G의 하위 비트를 모두 OR 한 것을 S라 할 때, (L&G)|(G&S)를 하면 그것이 반올림한 1의 자리라고 한다. Six tab에서 이를 구현하고자 한다면, (G&S)는 단순히 15를 더하면 된다. S가 1이라면 carry out되어서 G에 들어갈거고 G가 1이었다면 또 Carry out되어서 1에 자리에 1이 들어가게 되어서 맞는 반올림이 된다. (L&G)가 문제였다. 이걸 구현하려면 bit연산을 여러 번 해서 할 수는 있지만 (L&G)|(G&S)까지 구현하면 그냥 31이나 47을 더한 것만 못한 PSNR이 나왔다. 그래서 결론은 그냥 31을 더해서 LSR5를 하였다. 그리고 원래 Six-tab연산은 bound를 보장받는데, 그 범위가 0~255를 초과할 수 있기 때문에 clipping을 해줬다.

이상으로 과제를 마치겠습니다.