3SK3 Project 1

Bicubic Interpolation

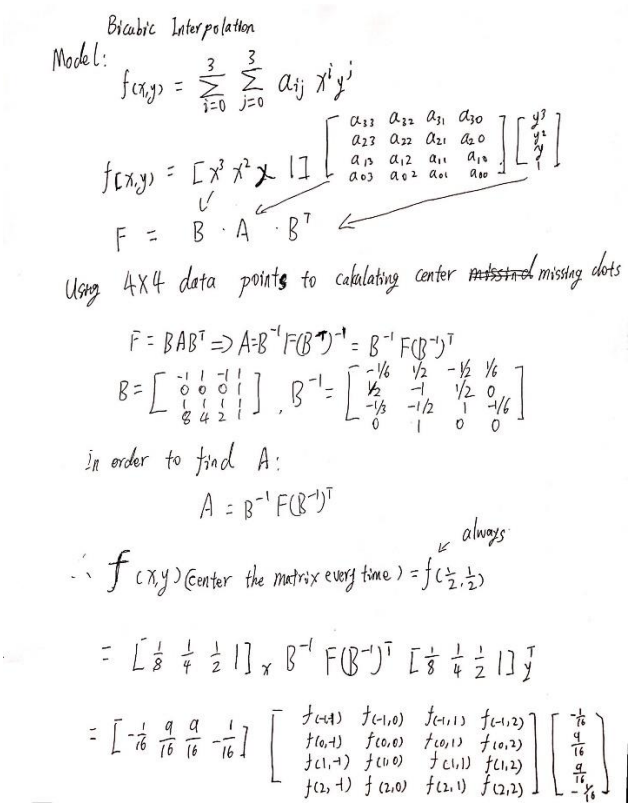
Jiaxian Wang

400113480

(a)

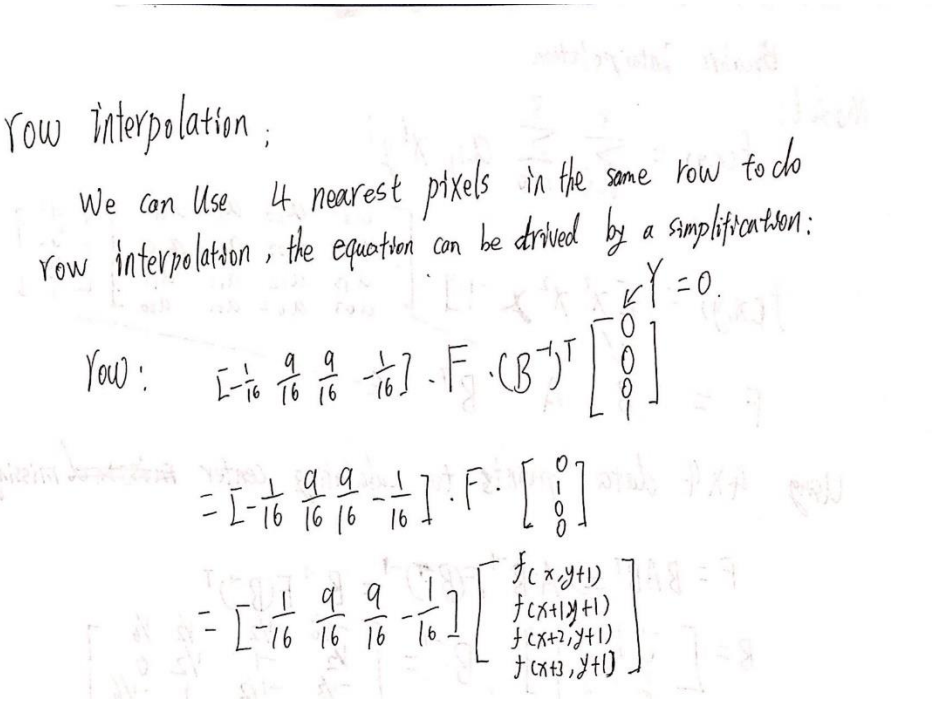
In the interpolation algorithm, the center missing pixels (shaded white dots) been interpolated first and then comes to the other missing pixels (empty white dots) and finally the four edges, which are caused by the algorithm used and it will be explained later.

1. The center missing pixels: using 4*4 nearest original pixels given

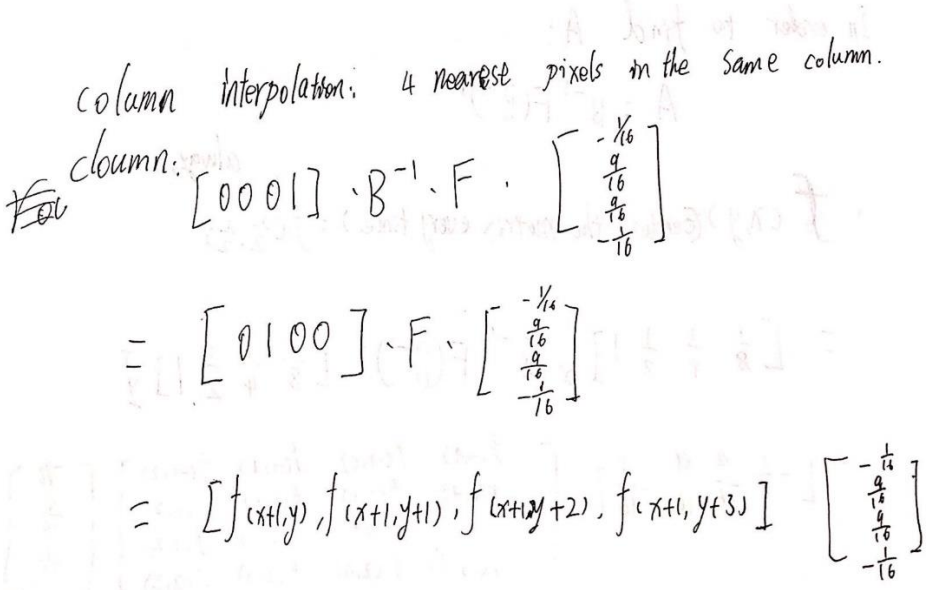


2. The other missing pixels: 4 in the same row or in the same column

Row interpolation

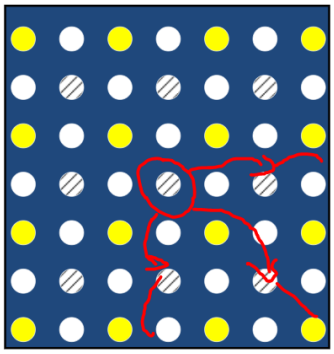


Column interpolation



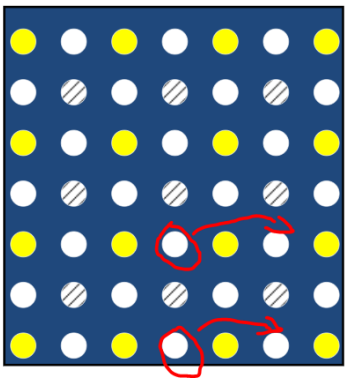
3. The edges

In the algorithm, the first center starts at the (3,3) and end at (596,796) since the center missing pixels out of this range has not enough 4*4 dots to estimate their value.

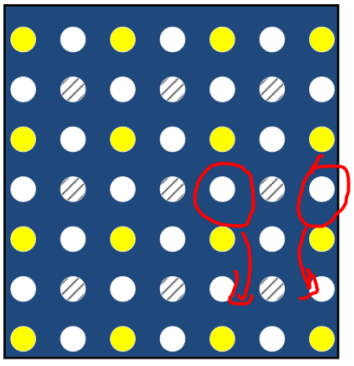


Therefore, there is a 3*3 edge and we need to interpolate these values using given pixels (yellow dots)

The row interpolation firstly starts at (4,3) and end at (596,795) since the center missing pixels out of this range has not enough 4 dots to estimate their value.

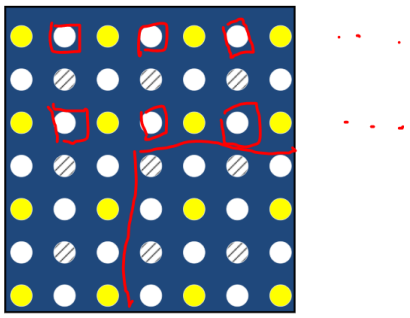


The column interpolation firstly starts at (3,4) and ends at (595,796) since the center missing pixels out of this range has not enough 4 dots to estimate their value.

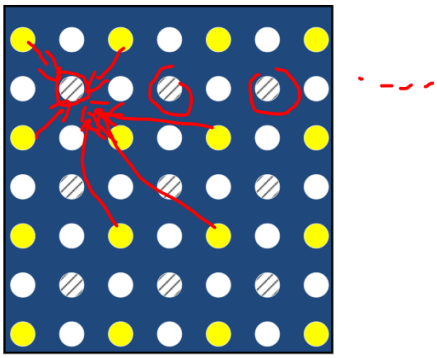


And the edge needs to be considered in using these three interpolation method accordingly.

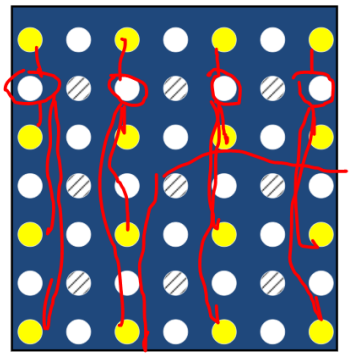
1. Dealing with row 0 and 2 using row interpolation



2. Dealing with shaded dots in row 1 using bicubic interpolation, if there is not enough to do the calculation, the information comes from the nearest available dot multiple times.



3. Dealing with the other missing dots in row 1 by doing column interpolation



By now, the missing pixels in an edge has been interpolated. The same order and method need to be used on the rest of 3 edges (row 597 and 598, column 0 to 2, column 797 and 798)

There is no need to interpolate row 599 and column 799 (last row and last column) since there are no effective information pixels near them. We can crop the outcome image by one row and on column as the requirement desired.

(b)

The difference between interpolate center missing pixels and other missing pixels is that only 4 pixels need to be used in the same column or same row (only one method for other missing pixels) after the simplification of the bicubic interpolation, and the 4*4 data points must be used to do the bicubic interpolation once it is approachable (center missing dots). The algorithm as been showed in the answer of question (a).

(c)

The run time program may take longer, but every single pixel interpolation information comes from the original pixel, in that case it ensures the accuracy.

Input image (original size):

300*400 pixels

Output image (original size):

599*799 pixels

Input image (original size):

192*256 pixels

Output image (original size):

383*511 pixel

Given example comparison (original size):

(d)

For center missing pixels:

```
for i in range(3, 2 * H - 3, 2):
    for j in range(3, 2 * W - 3, 2):
        for c in range(C):
            matF = np.matrix([[img_new[int(i - 3), int(j - 3), c], img_new[int(i - 3), int(j - 1), c],
                               img_new[int(i - 3), int(j + 1), c], img_new[int(i - 3), int(j + 3), c]],
                              [img_new[int(i - 1), int(j - 3), c], img_new[int(i - 1), int(j - 1), c],
                               img_new[int(i - 1), int(j + 1), c], img_new[int(i - 1), int(j + 3), c]],
                              [img_new[int(i + 1), int(j - 3), c], img_new[int(i + 1), int(j - 1), c],
                               img_new[int(i + 1), int(j + 1), c], img_new[int(i + 1), int(j + 3), c]],
                              [img_new[int(i + 3), int(j - 3), c], img_new[int(i + 3), int(j - 1), c],
                               img_new[int(i + 3), int(j + 1), c], img_new[int(i + 3), int(j + 3), c]]])
            coef_h = np.matrix([-1 / 16, 9 / 16, 9 / 16, -1 / 16])
            coef_l = np.matrix([[-1 / 16], [9 / 16], [9 / 16], [-1 / 16]])
            img_new[i, j, c] = np.dot(coef_h, np.dot(matF, coef_l))
```

Big_theta(n^2*3)

For row or column interpolation:

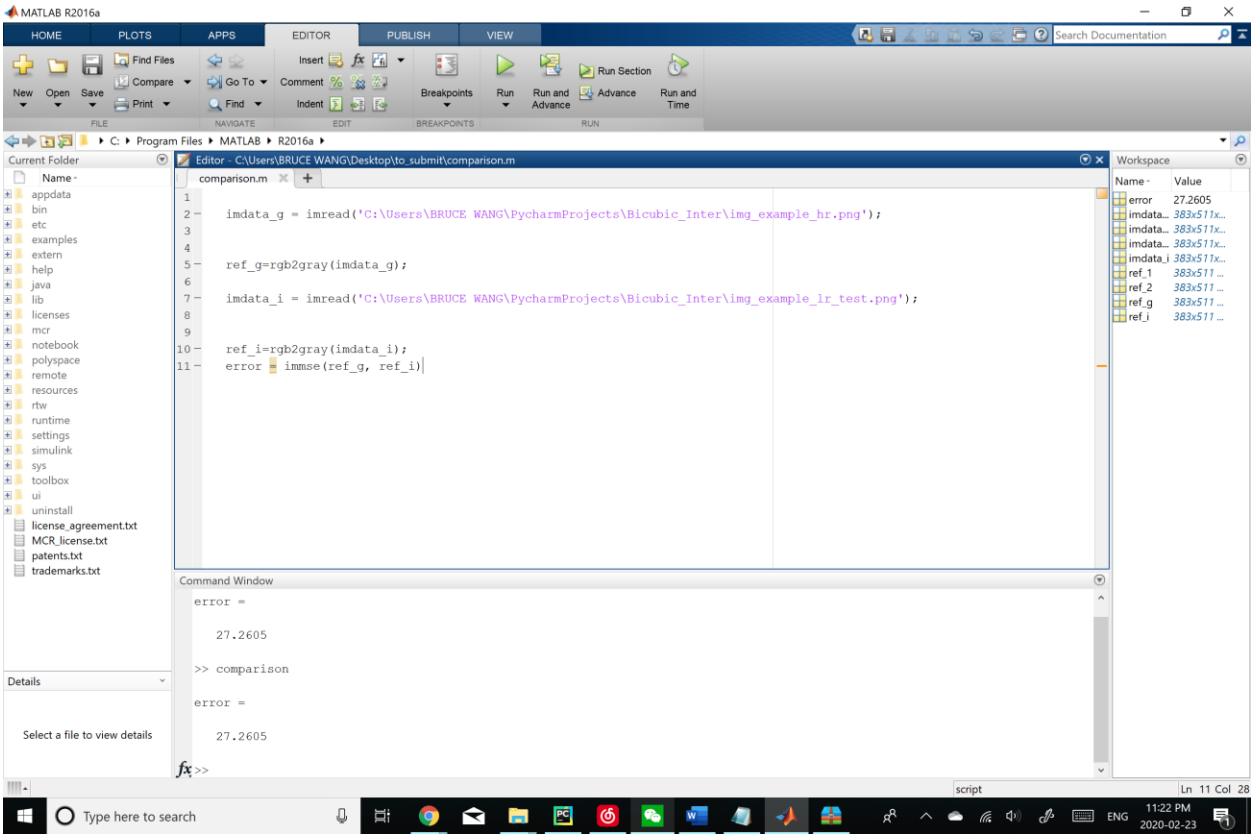
```
# row & column interpolation for missing dots
# do column interpolation first
for j in range(4, 2 * W - 3, 2):
    for i in range(3, 2 * H - 3, 2):
        for c in range(C):
            col_F = np.matrix(
                [img_new[int(i - 3), int(j), c], img_new[int(i - 1), int(j), c], img_new[int(i + 1), int(j), c],
                 img_new[int(i + 3), int(j), c]])
            coef_F = np.matrix([[-1 / 16], [9 / 16], [9 / 16], [-1 / 16]])

            img_new[i, j, c] = np.dot(col_F, coef_F)

# do row interpolation for missing dots
for i in range(4, 2 * H - 3, 2):
    for j in range(3, 2 * W - 3, 2):
        for c in range(C):
            row_F = np.matrix(
                [[img_new[int(i), int(j - 3), c]], [img_new[int(i), int(j - 1), c]], [img_new[int(i), int(j + 1), c]],
                 [img_new[int(i), int(j + 3), c]]])
            coef_R = np.matrix([-1 / 16, 9 / 16, 9 / 16, -1 / 16])

            img_new[i, j, c] = np.dot(coef_R, row_F)
```

Big_theta(n^2*3)

So, the average number of operations for a missing pixel is big_theta(n^2)

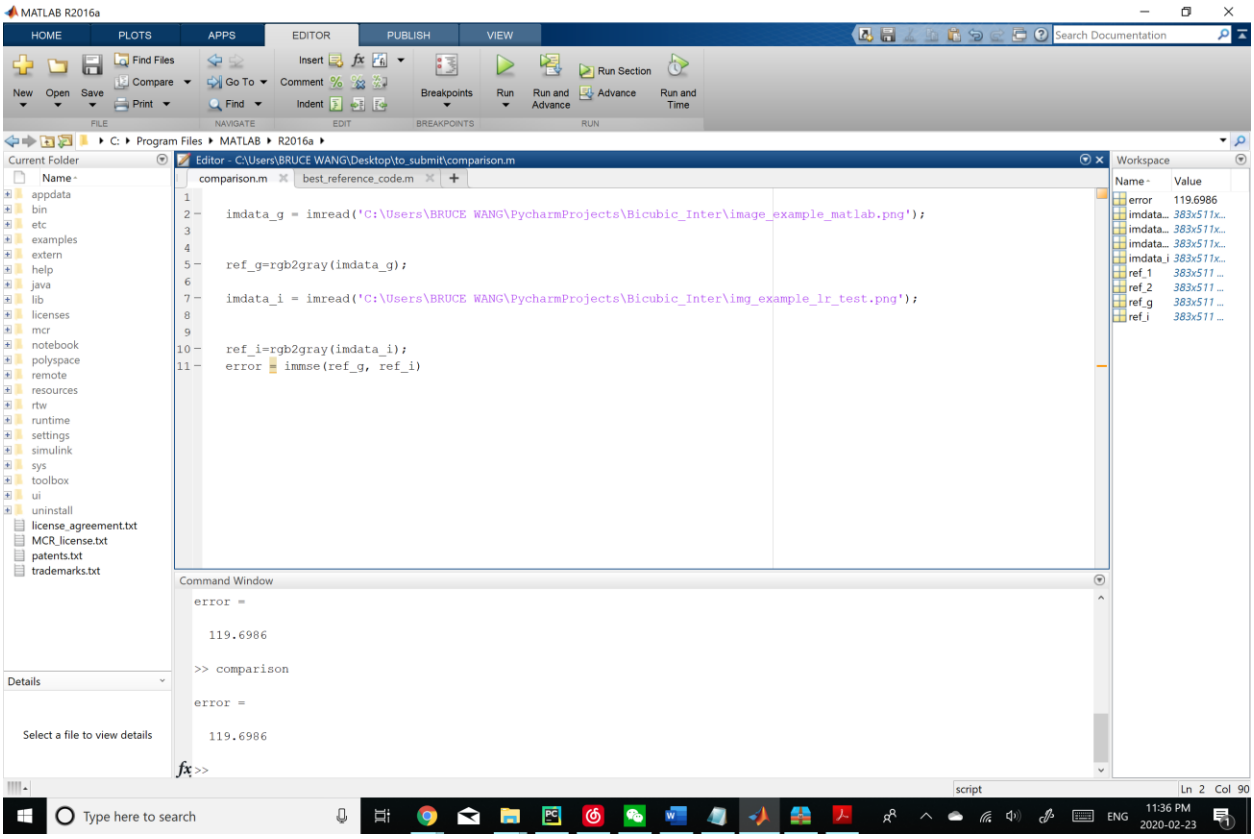
(e)

The mean square error using this algorithm compare with the given example_hr



Using the interpolation in found on internet (matlab based):

Code attached



It runs much more than my own program but the accuracy is not as high as the algorithm introduced.