

Notes for Lab 2 (Image Scaling Algorithm Description)

Algorithm for Image Rescaling:

1. Load the original image into a 2D array $o_img[x][y]$ by using the provided image IO functions. And record the width and height of the original image in variables: o_width and o_height .

Note: Most of the image load functions load a digital image into a 1D array by lining up the pixels in the image. Assume the 1D array is named as: $o_img_1D[t]$, where t is from 0 to $(o_width \times o_height - 1)$, it is easy to convert the 1D array to a corresponding 2D array $o_img[x][y]$ representing the original image, by using the following algorithm:

```
FOR x = 0 TO o_width-1 DO
    FOR y = 0 TO o_height-1 DO
        o_img[x][y] = o_img_1D[y * o_width + x];
    END FOR
END FOR
```

2. Decide the size (width and height) of the target image (scaled image) by setting its width: $t_width = o_width \times q_w$, where q_w is the width (horizontal) changing ratio provided by users. And its height is set to: $t_height = o_height \times q_h$, where q_h is the height (vertical) changing ratio provided by users.
3. Initialize a 2D array $t_img[x][y]$ for storing the pixel grey levels of the target image, where the size of the array is corresponding to the size of the target image, which is $t_width \times t_height$.
4. Starting to draw the target image (filling out the array $t_img[x][y]$) by using the following algorithm:

```
FOR x = 0 TO t_width -1 DO
    FOR y = 0 TO t_height -1 DO
        float o_x = x * (o_width - 1) / (t_width - 1)
        float o_y = y * (o_height - 1) / (t_height - 1)
        IF both o_x and o_y are actually integers, which means their
           fraction parts are zero
        THEN
            t_img[x][y] = o_img[(int)o_x][(int)o_y];
        ELSE
```

```

        t_img[x][y] = Nearest_neighbor_inter(o_img, o_x, o_y);
        //OR = bi_linear_inter(o_img, o_x, o_y, o_width, o_height);
        //OR = bi_cubic_inter(o_img, o_x, o_y, o_width, o_height);
    END IF
END FOR
END FOR

```

5. Output the target image (t_img) by using the image IO functions. Because most of the image write functions requires 1D array, so you need to change the 2D array $t_img[x][y]$ into a 1D array $t_img_1D[t]$. The size of the array t_img_1D is $t_width \times t_height$, which equals to the size of the target image. You can use the following algorithm to do so:

```

FOR x = 0 TO t_width-1 DO
    FOR y = 0 TO t_height-1 DO
        t = y × o_width + x;
        t_img_1D[t] = t_img[x][y];
    END FOR
END FOR

```

6. Algorithm End

Pseudo-code for interpolation functions:

```

FUNCTION Nearest_neighbor_inter
INPUT:    o_img // The 2D array of the original image
          o_x, o_y // the float position
OUTPUT:   The value of interpolation at position (o_x, o_y)

```

```

BEGIN
    SET nearest_x = ground(o_x + 0.5); // See document of lab 1
    SET nearest_y = ground(o_y + 0.5);
    RETURN (int)o_img[nearest_x][nearest_y];
END

```

```

FUNCTION bi_linear_inter
INPUT:    o_img // The 2D array of the original image
          o_x, o_y // the float position
          o_width, o_height // width and height of the original image
OUTPUT:   The value of interpolation at position (o_x, o_y)

```

```

BEGIN
    SET i_x as the integer part of o_x;
    SET f_x as the fraction part of o_x;
    SET i_y as the integer part of o_y;
    SET f_y as the fraction part of o_y; // Refer to the document of lab 1.

```

```

SET g1 = (1-f_x) * o_img[i_x][i_y] + f_x * o_img[i_x+1][i_y];
SET g2 = (1-f_x) * o_img[i_x][i_y+1] + f_x * o_img[i_x+1][i_y+1];
RETURN (int) ((1-f_y) * g1 + f_y * g2);

/* NOTE before retrieving the values of o_img[i_x][i_y], o_img[i_x][i_y+1],
o_img[i_x+1][i_y], o_img[i_x+1][i_y+1], you need to be sure the array indexes
are in the range of the original image: 0..o_width -1 and 0..o_height-1.
Otherwise you need to extend the original image in order to get valid values. */

END

FUNCTION bi_cubic_inter
INPUT:      o_img // The 2D array of the original image
           o_x, o_y // the float position
           o_width, o_height // width and height of the original image
OUTPUT:     The value of interpolation at position (o_x, o_y)

BEGIN

SET i_x as the integer part of o_x;
SET f_x as the fraction part of o_x;
SET i_y as the integer part of o_y;
SET f_y as the fraction part of o_y; // Refer to the document of lab 1.

Calculate R1, R2, R3, R4, S1, S2, S3, S4 // See the document of lab 1.
SET g1 = (1/6)*(S1*o_img[i_x-1][ i_y-1] + S2* o_img[i_x][ i_y-1] +
           S3*o_img[i_x+1][ i_y-1] + S4* o_img[i_x+2][ i_y-1]);
SET g2 = (1/6)*(S1*o_img[i_x-1][ i_y] + S2* o_img[i_x][ i_y] +
           S3*o_img[i_x+1][ i_y] + S4* o_img[i_x+2][ i_y]);
SET g3 = (1/6)*(S1*o_img[i_x-1][ i_y+1] + S2* o_img[i_x][ i_y+1] +
           S3*o_img[i_x+1][ i_y+1] + S4* o_img[i_x+2][ i_y+1]);
SET g4 = (1/6)*(S1*o_img[i_x-1][ i_y+2] + S2* o_img[i_x][ i_y+2] +
           S3*o_img[i_x+1][ i_y+2] + S4* o_img[i_x+2][ i_y+2]);

RETURN (int)( (1/6)*(R1*g1 + R2*g2 + R3*g3 + R4* g4) );

/* Note you also need to extend the original image when reaching outside of the
image, for example i_x-1 may be negative and therefore the value of o_img[i_x-
1][i_y] is not defined. */

```