CMPEN/EE455: Digital Image Processing I

Computer Project # 1:

Lab Introduction and Effects of Resolution Changes

Group members: Jiachen Chen; Yijie Tong; Lyuzhou Zhuang

Date: 09/10/2015

Objectives:

- Understanding down sampling technique and use it to shrink the size of a image; use nearest neighbor interpolation to zoom the image
- Learning interpolation methods (bilinear in this case) to up-scale a image; see its advantage over near neighboring interpolation
- Knowing how to change grey-level resolution by mapping the original value within a range to the new desired value

Methods:

Problem 1:

In this problem, the first step is to change the original images (256 x 256 pixels) to different sizes: 128 x 128, 64 x 64 and 32 x 32. Then, saving these images as 256 x 256 images.
To realize the first step, a function names *convert* is used. In this function, the original image (256x256 pixels) can be converted specific version image through selecting particular pixels from original image. In order to know the times between the original and result, a variable *time* added in to show the answer of equation: 256/(result's pixels in row/column). So we choose a pixel right and down from the last pixel, with the distance equals to the *time*.
For example, if we need a new image with 128x128pixels, the variable *time* equals to 256/128=2, so we select the coordinate (1,1),(1,3)…,(3,1),(3,3)…,(255,253), (255,255). Then these coordinates make the new image in order.
Second, to save these images as 256x256 pixel versions, we need to zoom them in suitable ways. Function save256 realizes this feature. We use the variable *time* to express the times between the 256 and the original size: time = 256/(origin pixel size). Then, each pixel in origin image is zoom to a square, with the size amount is time and all the elements in the square are the same as the origin pixel value.

For example, if we need to save a 128x128 image to 256x256image, the variable *time* = 256/128 = 2. So for any coordinate (x, y), in original image, it is zoomed to be a 2x2 square with four same values f(x, y). Then, these new pixels make up the new image.
After defining these two functions, we can use these two functions to complete the requirements.
We get in an image (‘boat.gif’) into the matlab program and display it. Then we use *convert* to convert the image to 128x128pixels image. Then, use *save256* function to save the result a new image—256x256pixels. Following are the repeat of these two steps, complete changing to 64x64 and 32x32pixels image and all are saved as 256x256pixels images. We finished this problem.

Problem 2:

This part aims at scaling a small 64x64 image up to 256x256 by using bilinear interpolation technique. The flow chart of such procedure is shown as Figure 1.1.

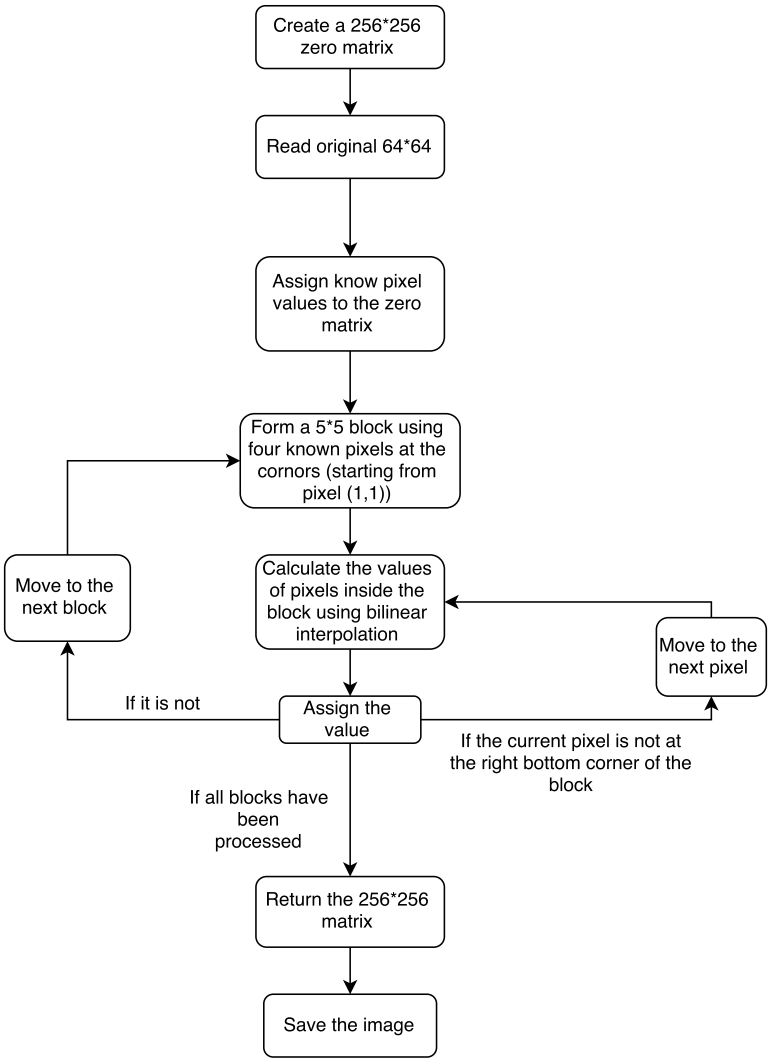

Create a 256*256 zero matrix
Read original 64*64
Assign know pixel values to the zero matrix
Form a 5*5 block using four known pixels at the cornors (starting from pixel (1,1))
Calculate the values of pixels inside the block using bilinear interpolation
Move to the next block
Move to the next pixel
If it is not
Assign the value
If the current pixel is not at the right bottom corner of the block
If all blocks have been processed
Return the 256*256 matrix
Save the image

Figure 1.1 Flowchart of problem 2

A function called *bilint* is used to implement this transformation.

The function first creates a zero-valued matrix **upcon** with the desired size, which in this case is 256x256. It then reads the original 64x64 image from the main function and assigns these values to **upcon.** Note that there are three unknown pixels between each assigned positions (column and row-wise).

Next, starting from the pixel (1,1), four neighboring pixels are chosen and they form a 5x5 block. For example, the first set would be (1,1), (1,5), (5,1) and (5,5). Given the equation 1-3 (from Proj1-Interpolation.pdf) of bilinear interpolation, every pixel value inside the block is calculated and assigned.

$$f_1 = \frac{y_2 - y}{y_2 - y_1} f(x_1, y_1) + \frac{y - y_1}{y_2 - y_1} f(x_1, y_2) \qquad eq.1$$

$$f_2 = \frac{y_2 - y}{y_2 - y_1} f(x_2, y_1) + \frac{y - y_1}{y_2 - y_1} f(x_2, y_2) \qquad eq.2$$

$$f(x, y) = \frac{x_2 - x}{x_2 - x_1} f_1 + \frac{x - x_1}{x_2 - x_1} f_2 \qquad eq.3$$

This operation sweeps from the top left corner to the bottom right corner and is repeated for 25 times, even with the four known pixels.

Upon finishing the last pixel in the current block, the next four points are chosen to form another block. The calculating process needs to be repeated for all blocks inside the matrix **upcon**.

As shown in FIgure 1.2, note that two consecutive blocks will have a common row or column shown as orange areas in Figure x and the values on these edges will be rewritten (may be different) during calculation for the next block.

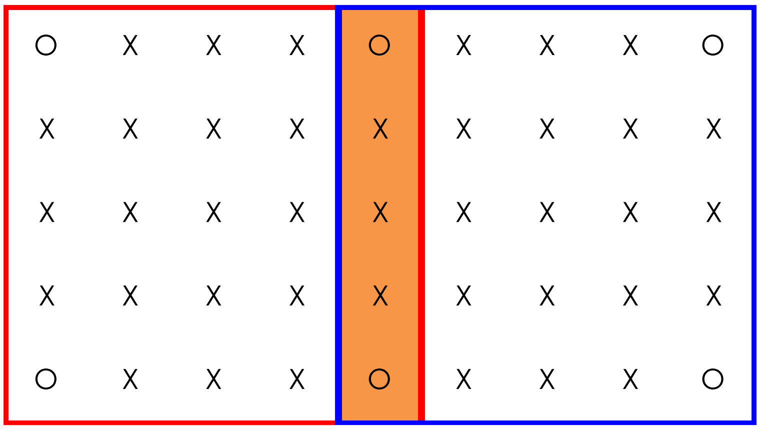


Figure 1.2 Two 5x5 blocks with a common column marked as orange (o known values read from the original 64*64 image)

In this particular case, since no border handling method is specified, all blocks that is partially out of the **upcon** matrix are discarded and remains to be zero (except for the common edges with other completed blocks)

Problem 3:

In this problem, we need to write a program to change the gray-level quantization of a certain image by reducing the number of bits per pixel from 8 to 6, 4, 2 and 1 bits/pixel. For example, for the 2 bits/pixel case (2^2 grey levels), the image pixels should use the gray levels 32, 96, 160, or 224, etc.

To accomplish the task, we wrote a function ”greyconvert.m” to scan each pixel of a given image, calculate a new proper grey value for this pixel and save the new grey value to the corresponding pixel in a new image. In the 2 bits/pixel case(4 grey levels), all pixels whose grey values are between 0 and 256/4 will be assigned a new grey value 32 (comes from$\frac{0*\frac{256}{4}+1*\frac{256}{4}}{2}$), those between 256/4 and 2*256/4 will be assigned 96 (comes from $\frac{1*\frac{256}{4}+2*\frac{256}{4}}{2}$), and so on. In the 6 bits/pixel case(2^6 grey levels), all pixels whose grey values are between 0 and 256/64 will be assigned a new grey value 32 (comes from$\frac{0*\frac{256}{64}+1*\frac{256}{64}}{2}$), those between 256/64 and 2*256/64 will be assigned 96 (comes from $\frac{1*\frac{256}{64}+2*\frac{256}{64}}{2}$), and so on. All other cases are similar.

Lastly, print out the new image.

Results:

Problem 1:

The original image is showed in Figure2.1

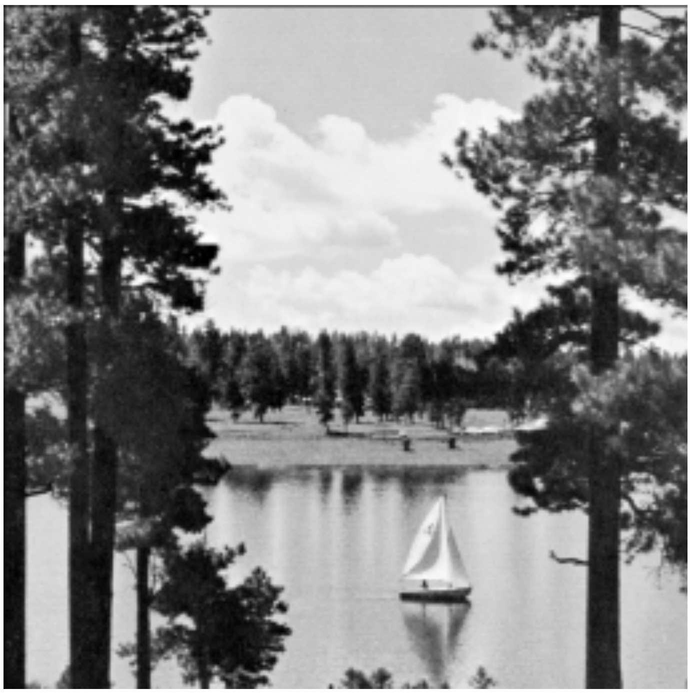


Figure2.1 The original image: boat

The Figure2.2 shows images with 128x128 pixels, 64x64 pixels and 32x32 pixels.

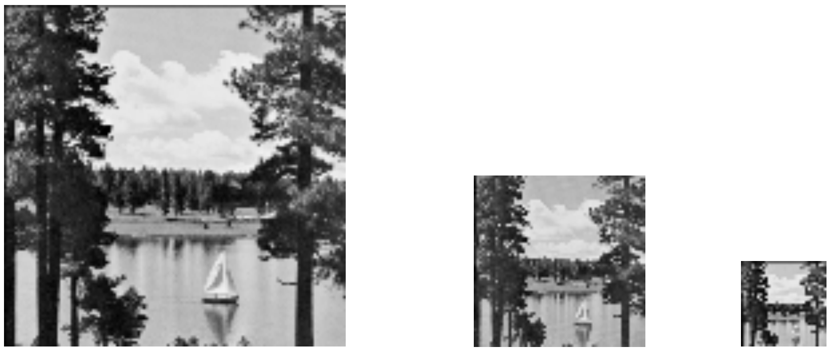
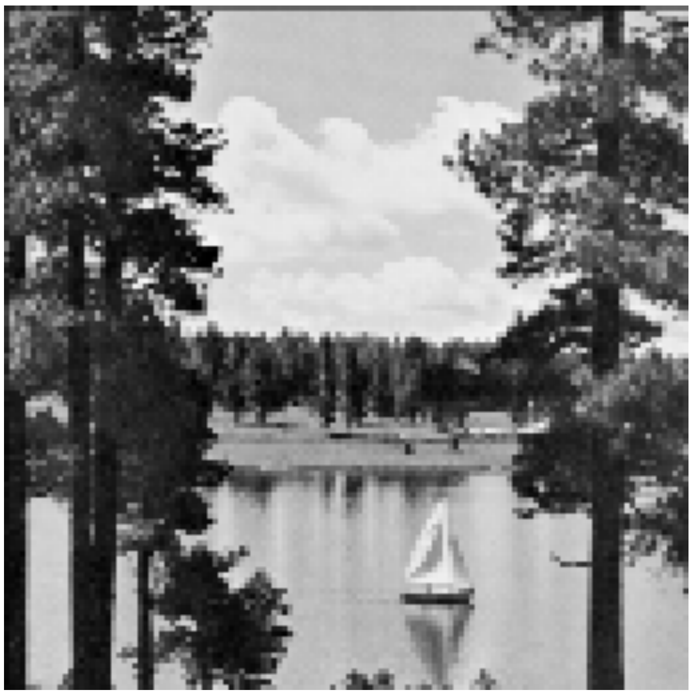
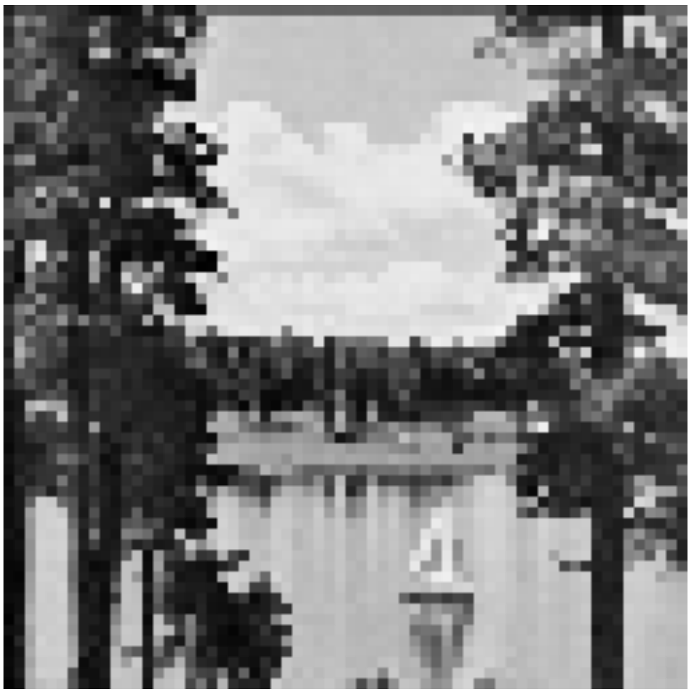

(a) 128x128 (b) 64x64 (c) 32x32

Figure2.2 images changed from boat.gif

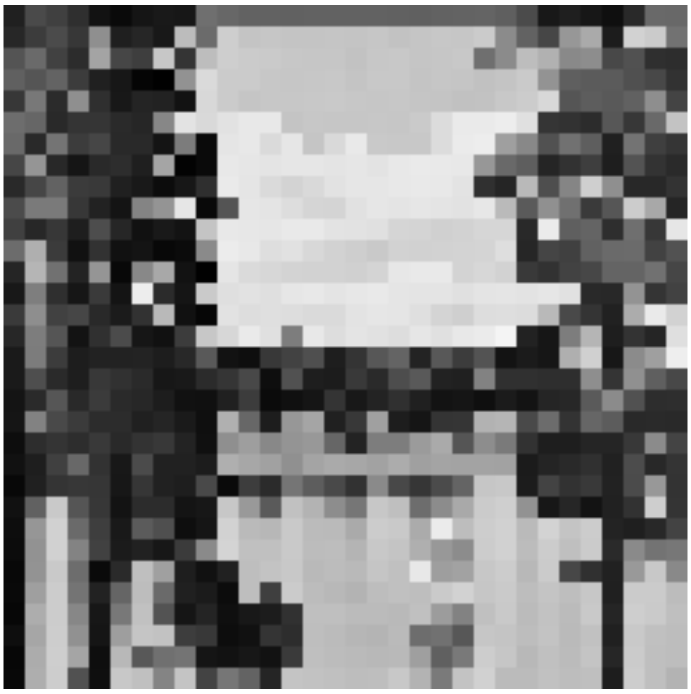
We can see that with fewer pixels in the image, the image is smaller and smaller, but we can see the outline even in 32x32pixel image.

Then the following Figure2.3 shows the saving version with 256x256 pixels.


(a)the origin is 128x128pixels



(b)the origin is 64x64

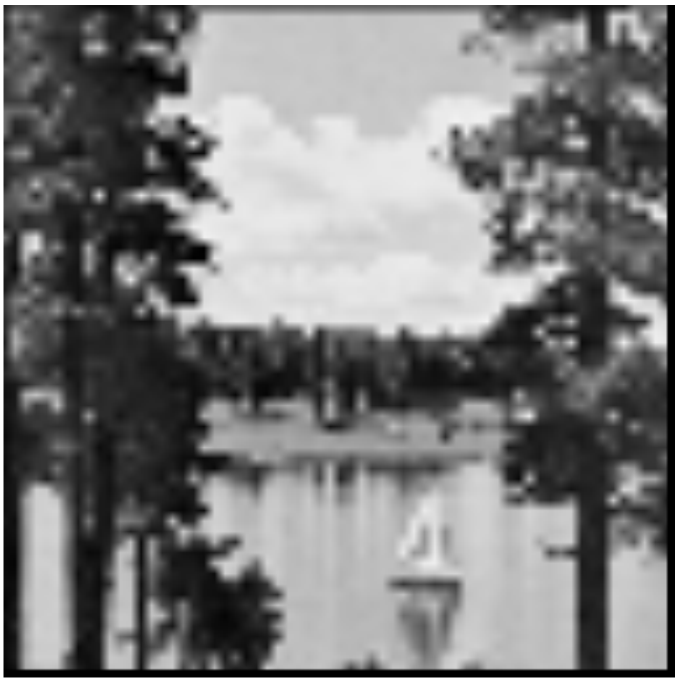


(c) the origin is 32x32

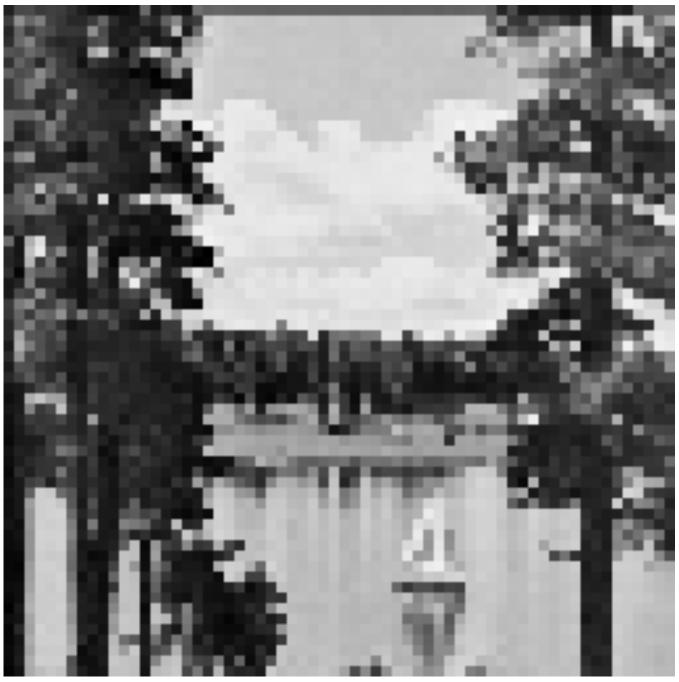
Figure2.3 Three new images saved as 256x256 pixels images

We can see that although we zoom them to 256x256pixels images, fewer pixels in original image contribute to more blurry results. We almost cannot identify the last two figures.

Problem 2:



(a)



(b)

Figure 2.4 Image scaled by (a) bilinear interpolation (b) nearest neighbor interpolation

Figure 2.4 (a) shows the result of using bilinear interpolation to scale a 64x64 image to 256x256. Compare it with the image using nearest neighbor interpolation in question 1 shown as Figure 2.4 (b), image (a) contains more detail but blurred.
There are two black edges, which are caused by border issues mentioned above, in Figure 2.4 (a).

Problem 3:

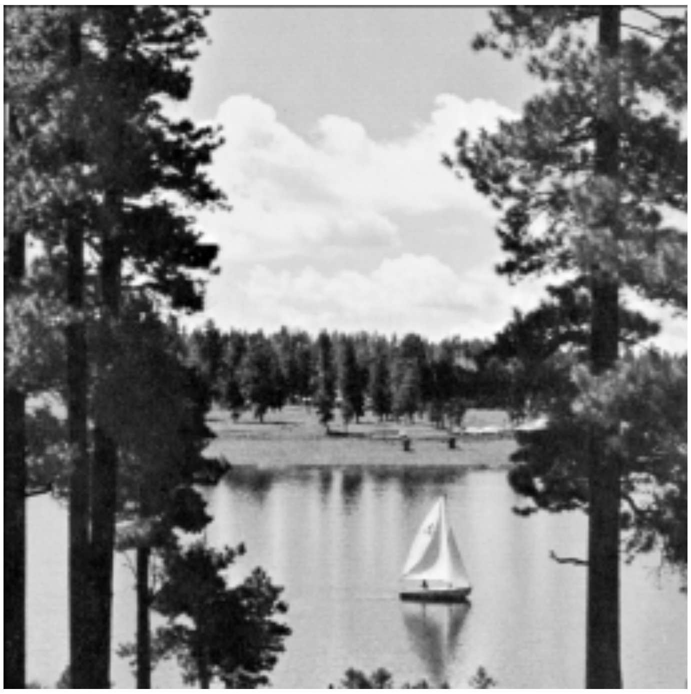


Figure2.5 The original image: boat.gif

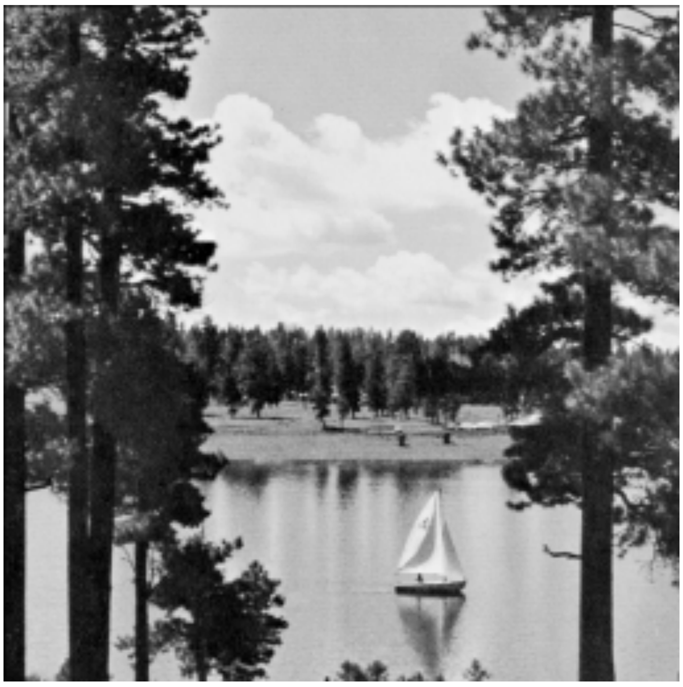


Fig 2.6 the image with 256 grey levels

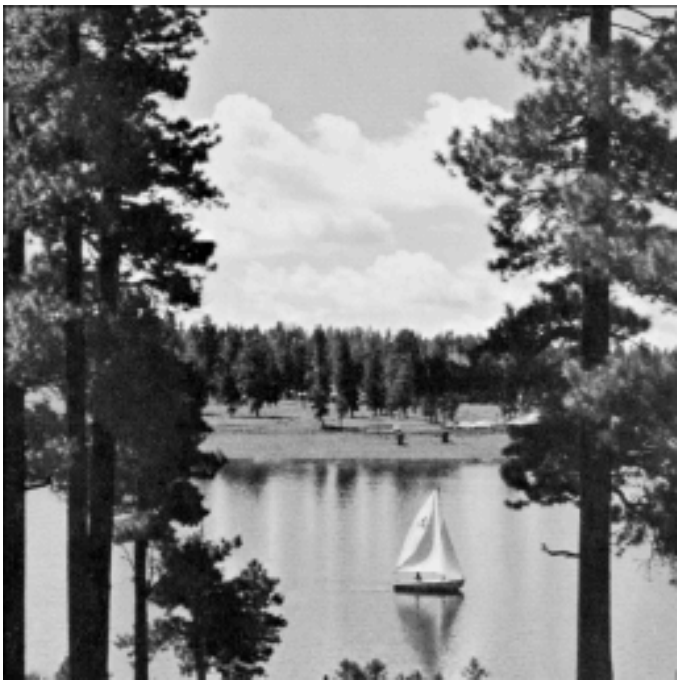


Fig 2.7 the image with 64 grey levels

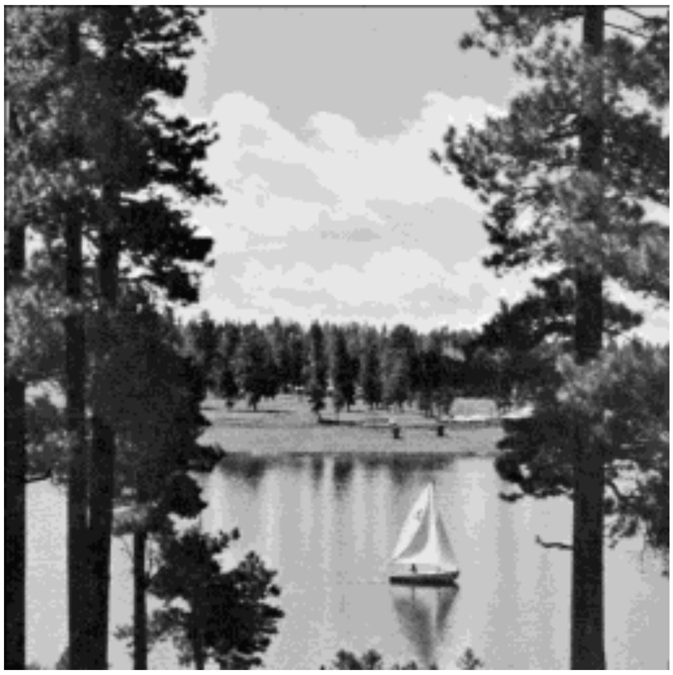


Fig 2.8 the image with 16 grey levels

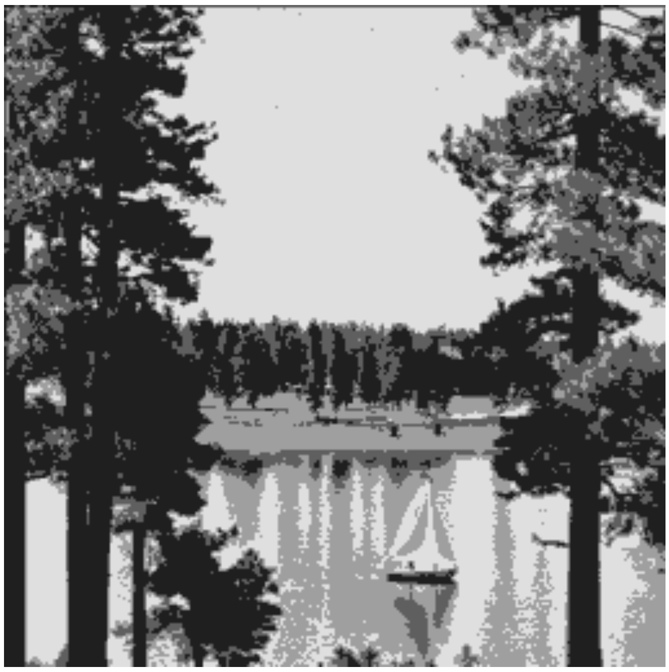


Fig 2.9 the image with 4 grey levels

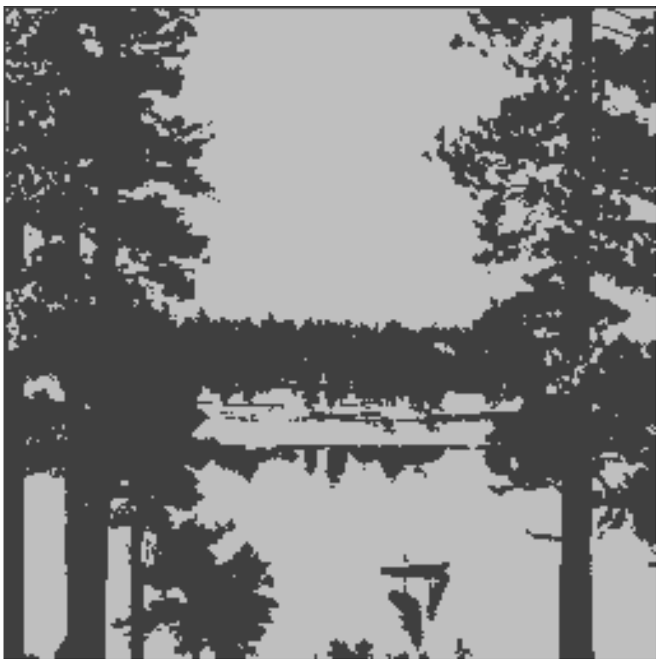


Fig 2.10 the image with 2 grey levels

We can tell from the results that as the quantity of the grey levels of an image decreases, the quality of the image decreases, too.

Conclusion:

In conclusion, we successfully demonstrate the method of down sampling and zooming using nearest neighbor interpolation. The smaller the original image is, the poorer the quality of zoomed image is. Bilinear interpolation is implemented and in comparison with the former method, the image contains richer details. Grey-scale resolution is also changed and the image starts to look distorted only after it drops below 64 grey levels, which means human eyes cannot resolve 256 grey levels.