CMPEN/EE455: Digital Image Processing I

Computer Project # 1

Introduction and Digital Image Quantization

Ya Mei, Ziheng Fu, Songmeng Wang

Date: 09/13/2019

Objectives

This project is primarily designed to introduce students to MATLAB's capabilities for digital image processing. Through the project, students can get a better understanding of the effects of spatial and gray-scale resolution changes on digital images. Nearest - neighbor interpolation and bilinear interpolation are applied for algorithms.

Methods

Part 1:

In order to down sample the original 512×512 image to 256×256, 128×128 and 32×32 images, we need to resample the image with different incremental values, that is, the increment is 2 for 256×256 image, 4 for 128×128 image and 16 for 32×32 image.

After the image is downsampled, we have to scale it back to a 512×512 image. The method we used is nearest-neighbor interpolation. We will first create our desired 512×512 matrix, and then the values are assigned into the pixels. We get our 512×512 image by dividing the coordinates of our 512×512 matrix by different relationship factor (2 for 256×256. 4 for 128×128 and 16 for 32×32), the larger integer of the outcome is taken by using the ceil function. To illustrate this, we assume we want to upsample the 256×256 image to 512×512; the relationship factor is 2. The pixels are replicated to its next coordinates in both vertical and horizontal axis. The same process applies but just for different relationship factors. Using this procedure and algorithm, we are able to downsample the image and upsample it back.

Part 2:

Bilinear Interpolation is applied to create a 512×512 image from the 32×32 image of part 1. The idea about Bilinear Interpolation is to perform linear interpolation in both directions while they are mutually orthogonal.

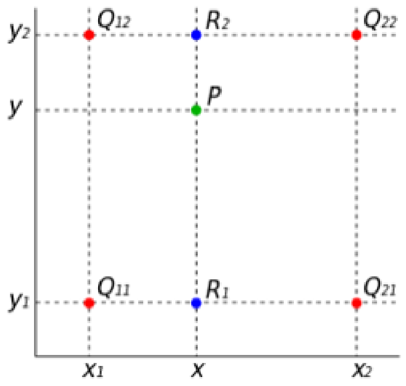


Figure 1: Introduction to Bilinear Interpolation.

As indicated in figure 1, P is an unknown pixel of the image while Q_{12}, Q_{11}, Q_{21} and Q_{22} are known pixels (the gray values of them are known). To compute the value of P, the linear interpolation f_1 along $Q_{12}$and Q_{11}, and f along $Q_{22}$and Q_{21} are firstly computed,

$$f_1 = \frac{y_2 - y}{y_2 - y_1} f(Q_{12}) + \frac{y - y_1}{y_2 - y_1} f(Q_{11})$$

$$f_2 = \frac{y_2 - y}{y_2 - y_1} f(Q_{22}) + \frac{y - y_1}{y_2 - y_1} f(Q_{21})$$

then the result of P can be easily worked out by taking the linear interpolation along y_1 and y_2.

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

By performing the same algorithm to all pixels in the 32×32 image, the 512×512 image can be generated accordingly.

Practically, a 512×512 zero image (the image whose pixels are set to '0') is generated to form a platform for operating the algorithm. Specifically, all pixels in the 32×32 image are attributed to the upper left pixel of every 16 pixels along horizontal and vertical direction in the 512×512 image, then the proper values of other zero pixels could be computed.

Part 3:

f_8 is defined for the original image and m, n for size for further usage. Size function is used in order to get m and n value. The flow chart is shown below.

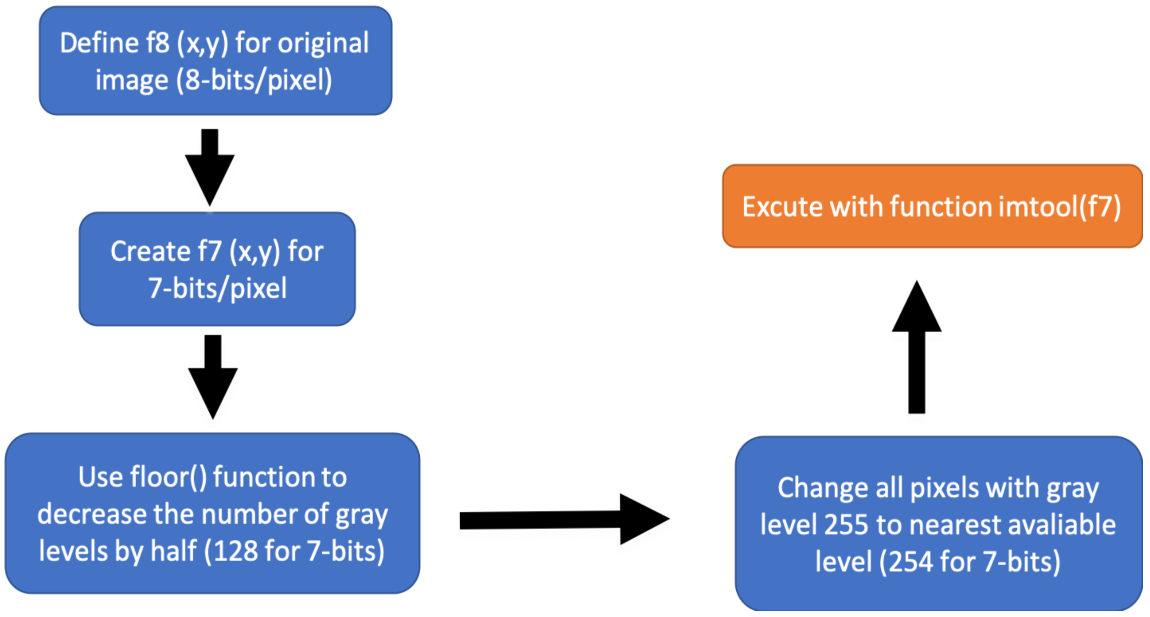

Define f8 (x,y) for original image (8-bits/pixel)
Create f7 (x,y) for 7-bits/pixel
Use floor() function to decrease the number of gray levels by half (128 for 7-bits)
Change all pixels with gray level 255 to nearest avaliable level (254 for 7-bits)
Excute with function imtool(f7)

Figure 1.1: Flow chart for part (3)

After defining f_8, we create functions based on the number of bits per pixel. Floor() function is then used to decrease the bits by decreasing the number of gray levels. For each n-bits per pixel, we would have (2 to the power of n) gray levels totally. Floor() function could help us restrict the number of gray levels as expected. All gray values are changed as expected except 255. To replace this exception, we add a function to change the unexpected to the nearest available gray level. Imtool() is used at last to check the results of the operations.

To execute the entire program in part 3, we could simply type in p1_3 to get all the images required.

Part 4:

Part 4 requires the algorithms from part 1 and part 3.

Results

Part 1:

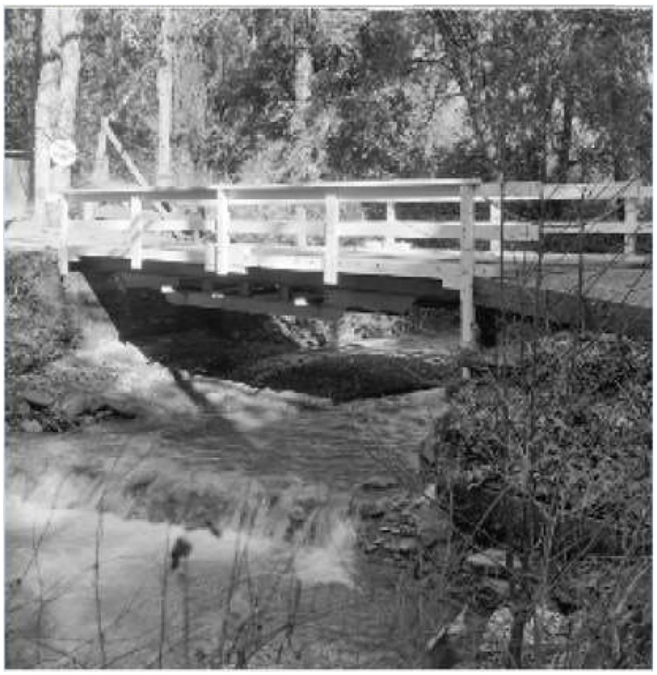


Figure 2: Result after up sampling 256x256 image to 512x512

This 512×512 image is obtained from up sampling 256×256 image. We can see that there are no significant differences on the overall appearance. However, if we zoom in to view its details, we can tell that the 256×256 image is less clear and detailed than the original 512×512 image.

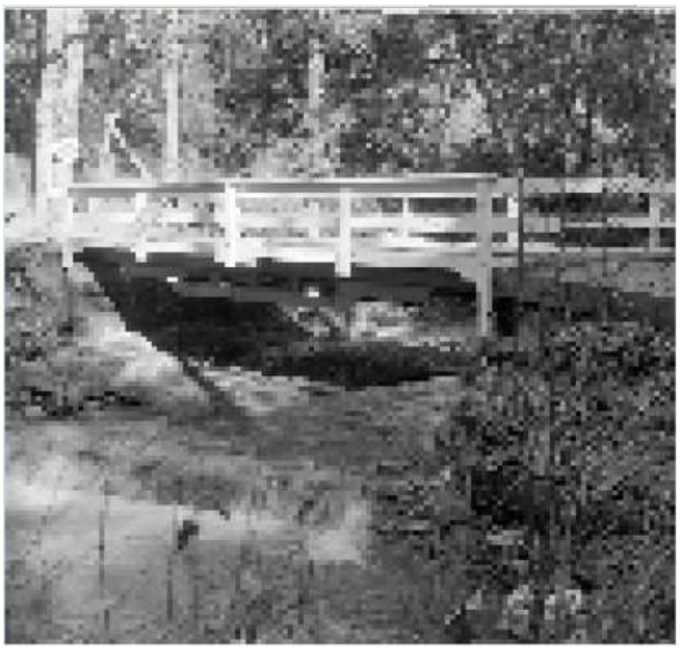


Figure 3: Result after up sampling 128x128 image to 512x512

This image is obtained from upsampling the 128×128 image to a 512×512 image. We can see this 128×128 image is much less clear and detailed than the original 512×512 image even though we are still able to identify the objects in the image.

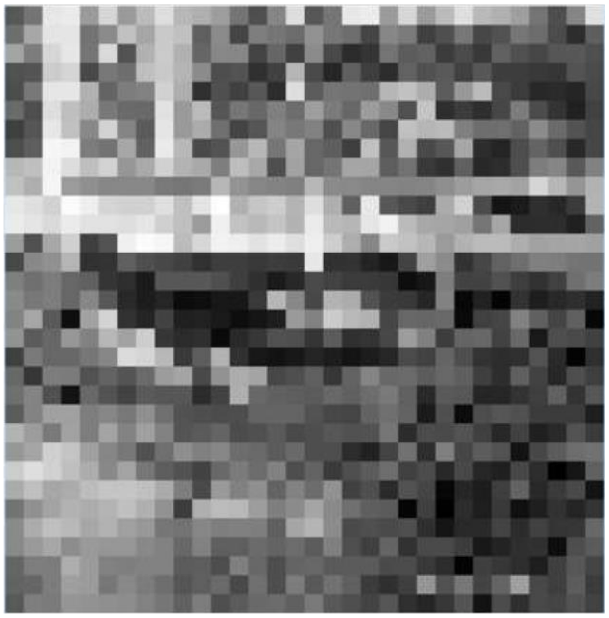
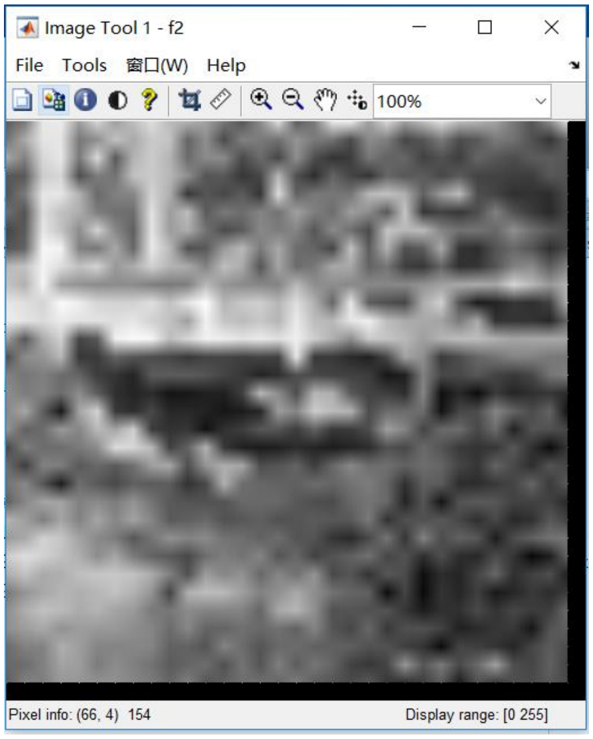


Figure 4: Result after up sampling 32x32 image to 512x512

This image is obtained from upsampling the 32×32 image to a 512×512 image. We cannot even identify the objects in the image anymore. All we can see are the disordered pixels laid out.

Part 2:

The result of bilinear interpolation is shown by figure 5.



Pixel info: (66, 4) 154

Display range: [0 255]

Figure 5: The result after performing Bilinear Interpolation.

The result returns a 512×512 image which is blurry and murky compared with the image which was performed by nearest neighbor interpolation. However, the shape of the bridge can still be visualized.

It can't be ignored that there exists two straight black line at the right and bottom edges due to the restriction of bilinear interpolation. This is understandable because the pixels at the edge of the 512×512 image are zero and only linear interpolation can be applied on the pixels at the edges.

Although the pixels at the edges are default to be 0, bilinear interpolation can still be applied on those pixels given the valid pixels are limited. Figure 3.3 shows the result of applying bilinear interpolation on the pixels at the edges.

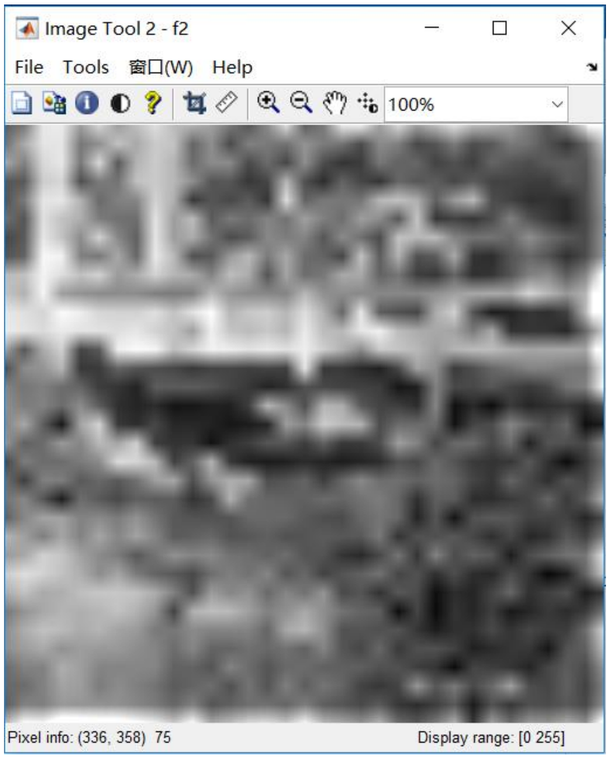


Figure 6: Result of Interpolating the edge pixels.

From figure 6, the left and bottom edges are even blurrier than other pixels. Subjectively, this is not a successful attempt.

Part 3:

Eight images with different bits per pixel (8-bits / pixel to 1-bit/pixel) are derived in part 3. We could see the change when decreasing the number of bits per pixel. Images are shown below:

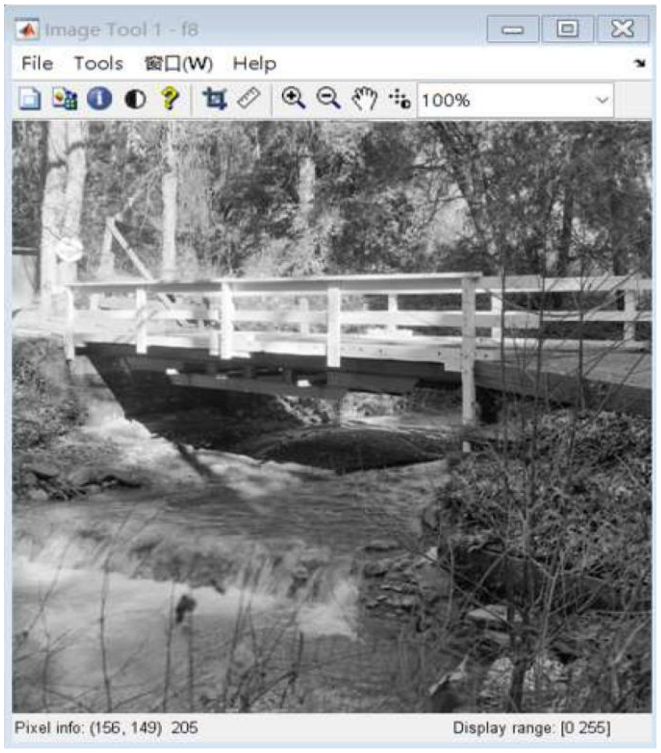


Figure 7: 8-bits/pixel image (256 gray levels: 0~255)

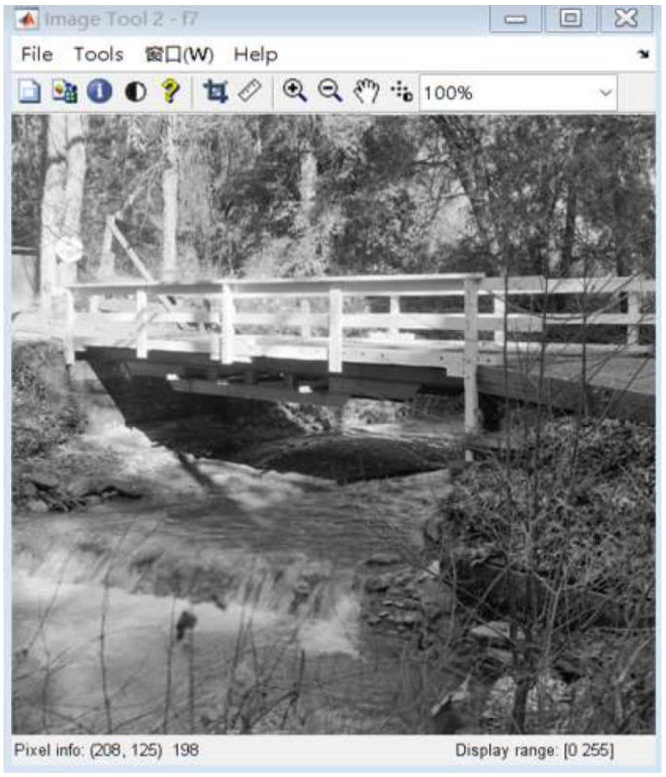


Figure 8: 7 bits/pixel image (128 gray levels: 0, 2, 4, 6 … 254)

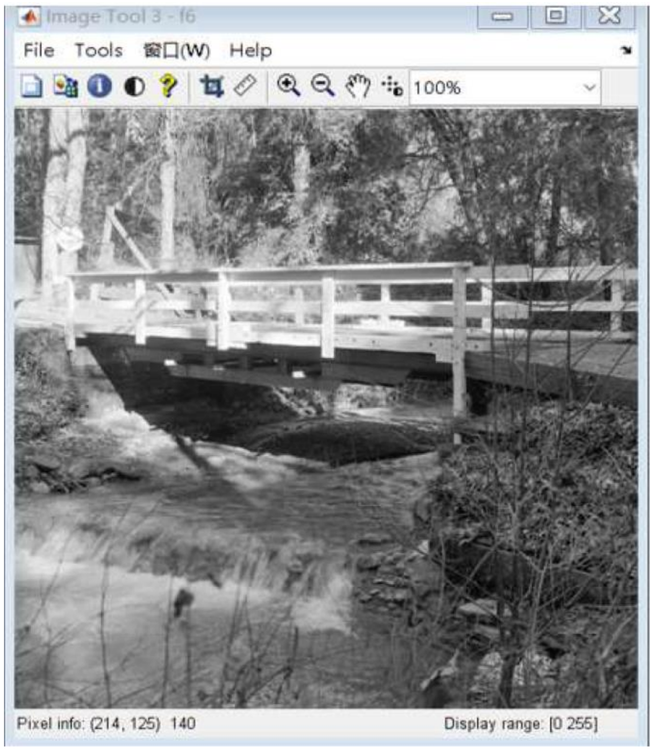


Figure 9: 6 bits/pixel image (64 gray levels: 0, 4, 8, 12 … 252)

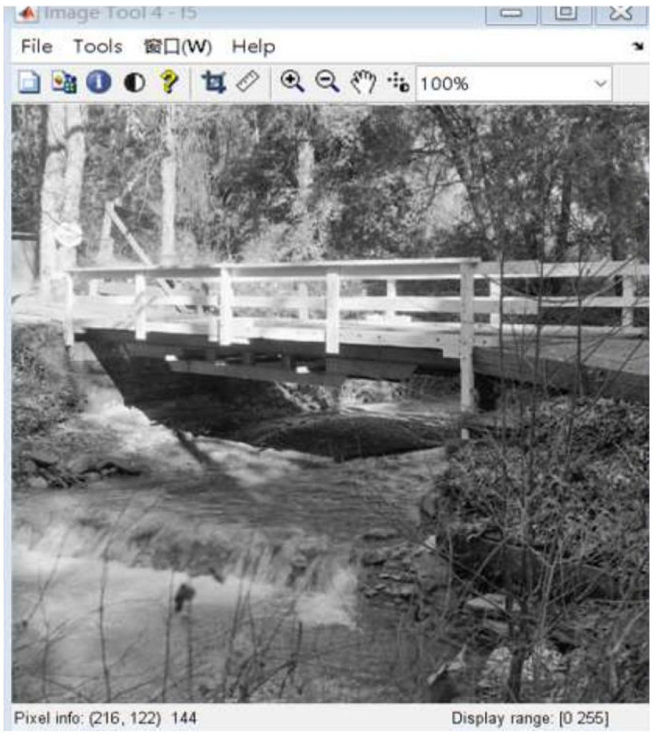


Figure 10: 5 bits/pixel image (32 gray levels: 0, 8, 16, 24 … 248)

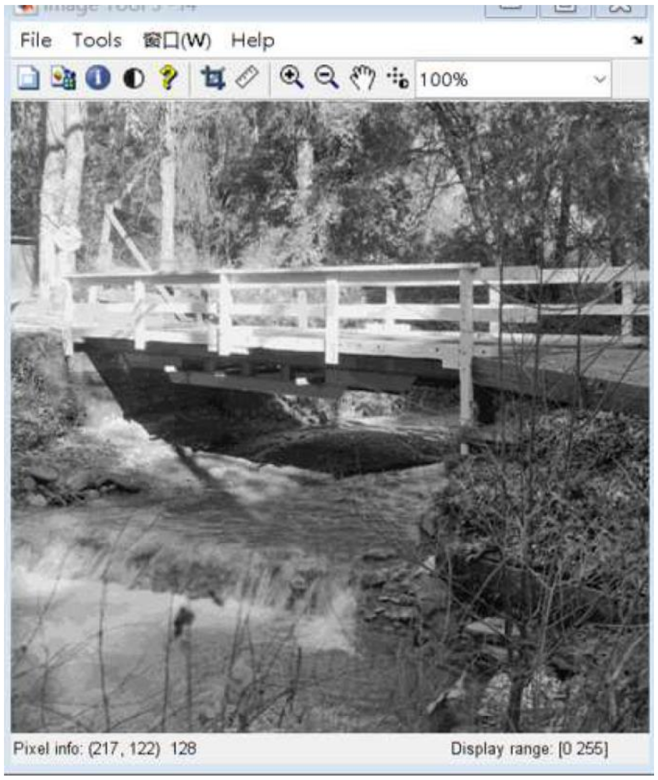


Figure 11: 4 bits/pixel image (16 gray levels: 0, 16, 32, 48 … 240)

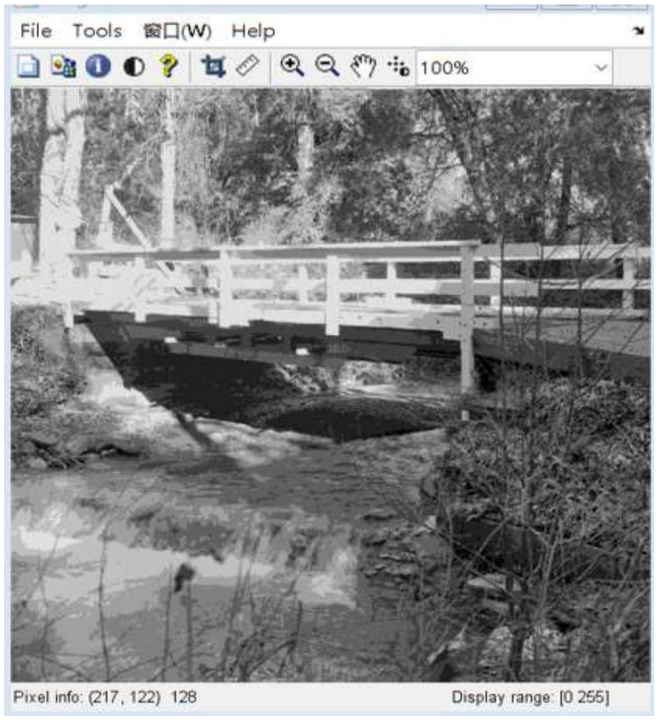
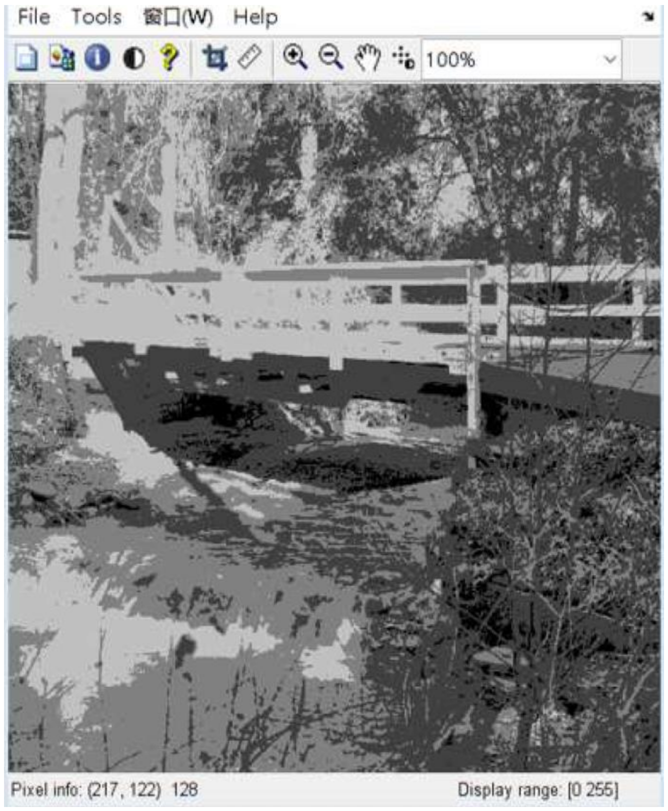


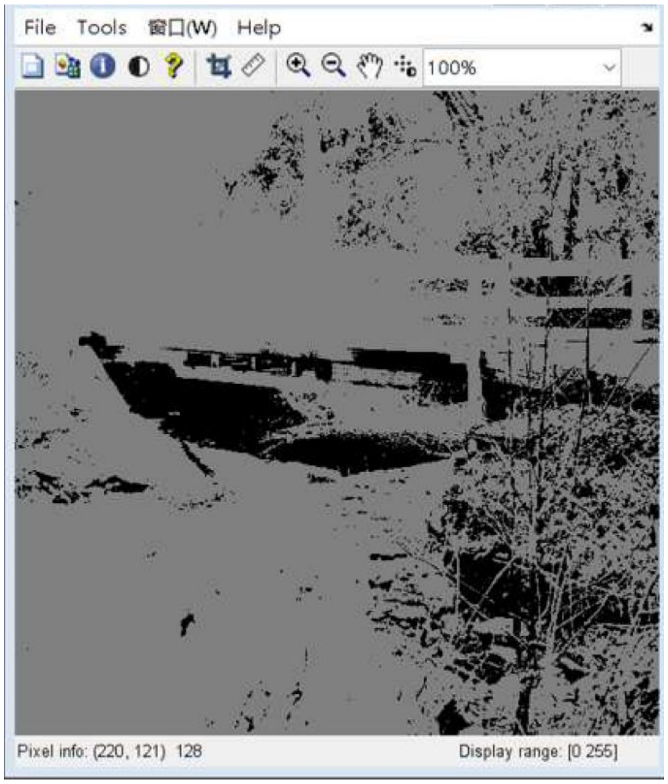
Figure 12: 3 bits/pixel image (8 gray levels: 0, 32, 64, 96 … 224)



Pixel info: (217, 122) 128

Display range: [0 255]

Figure 13: 2-bits/pixel image (4 gray levels: 0, 64, 128, 192)



Pixel info: (220, 121) 128

Display range: [0 255]

Figure 14: 1-bit/pixel image (2 gray levels: 0, 128)

As we can see from all the images above, there are not many differences between the first five images (from 8 bits/pixel to 4 bits/pixel). Change from 4 bits/pixel to 3 bits/pixel is more visible, but we could still easily figure out what the image is about with 8 gray levels. Images become harder to recognize with less or equal to 4 gray levels (2-bits/pixel and 1-bit/pixel). We have only 4 gray level with 2-bits/pixel. The whole image looks like a sketch instead of a photo. Finally, we get only 2 gray levels left with 1-bit per pixel. We could hardly tell what the image is about. All the lighter parts of the image become grey with gray value 128 and all the darker regions become black with gray value 2. Images could not be comprehended with only 2 gray levels (1 bit).

Part 4:

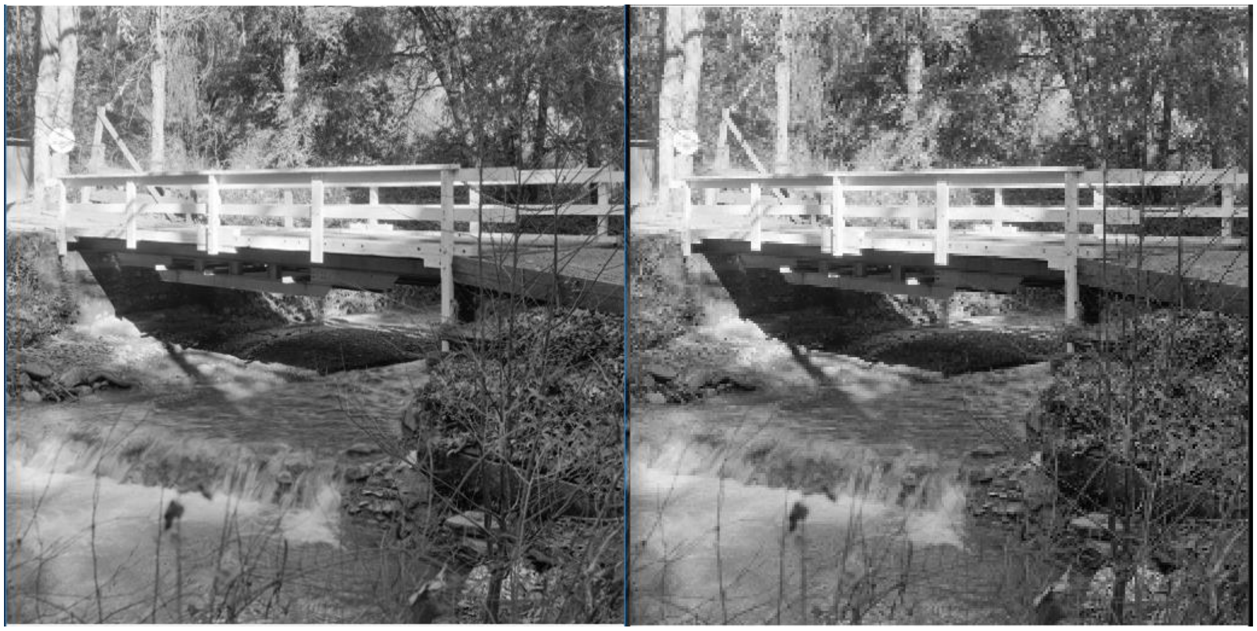


Figure 15: The left image is 512x512 pixels and 8 bits/pixel; the right image is 256x256 pixels (but the image size is 512x512) and 6 bits/pixel

Comparing to the original high-resolution image, the new obtained image does display some minor artifacts. Although we cannot tell there is any difference of those two images from gray-scale perspective, we can tell there are some minor differences from spatial resolution perspective. For example, the branches of the trees on the right side are more detailly and clearly depicted in the original high-resolution image than those depicted in the new obtained image.

Conclusions

In this project, the spatial resolution and gray-scale resolution of a 512×512 image is changed into different resolutions. Decreasing the spatial resolution returns an image which has less visual quality. This sort of change of visual quality is more obvious than changing the gray-scale resolution given the similar scale of decrement. Additionally, the apparent contrast is increased as the gray-scale resolution decreased (recall the discussion in Sec 2.4 in the textbook G&W about the iso-preference curves).

The effect of bilinear interpolation and nearest neighbor interpolation can be also concluded to have quite a lot of differences. The contrast between the pixels in the image resulted from nearest neighbor interpolation is more obvious than of the bilinear interpolation while bilinear interpolation generates a blurrier result than that of nearest neighbor interpolation.