HW1 Report

13331270 吴家荣 嵌软方向

1 Exercises

Answer the following questions in your report.

1.1 Storage (3 ∗ 3 = 9 Points)

If we consider an N-bit gray image as being composed of N 1-bit planes, with plane 1 containing the lowest-order bit of all pixels in the image and plane N all the highest-order bits, then given a 2048 × 2048, 256-level gray-scale image:

1. How many bit planes are there for this image?

8比特

2. Which panel is the most visually significant one?

plane 8 是最重要的

3. How many bytes are required for storing this image? (Don't consider image headers and compression.)

2048 * 2048 * 8 / 8 = 4 M

1.2 Adjacency (3 ↤ 3 = 9 Points)

Figure 1 is a 5 ↦ 5 image. Let V = {1, 2, 3} be the set of pixels used to define adjacency. Please report the lengths of the shortest 4, 8, and m-path between p and q. If a particular path does not exist, explain why.

3	4	1	2	0
0	1	0	4	2(*q*)
2	2	3	1	4
(*p*)2	0	4	2	1
1	2	0	3	4

Figure 1: Adjacency.

p, q的4连通没有路径，因为在$N_4(q)$都不在V中

p, q的8连通最短路径为：4

p, q的m连通最短路径为：5

1.3 Logical Operations (3 ⇤ 3 = 9 Points)

Figure 2 are three di↵erent results of applying logical operations on sets A,B and C. For each of the result, please write down one logical expression for generating the shaded area. That is, you need to write down three expressions in total.

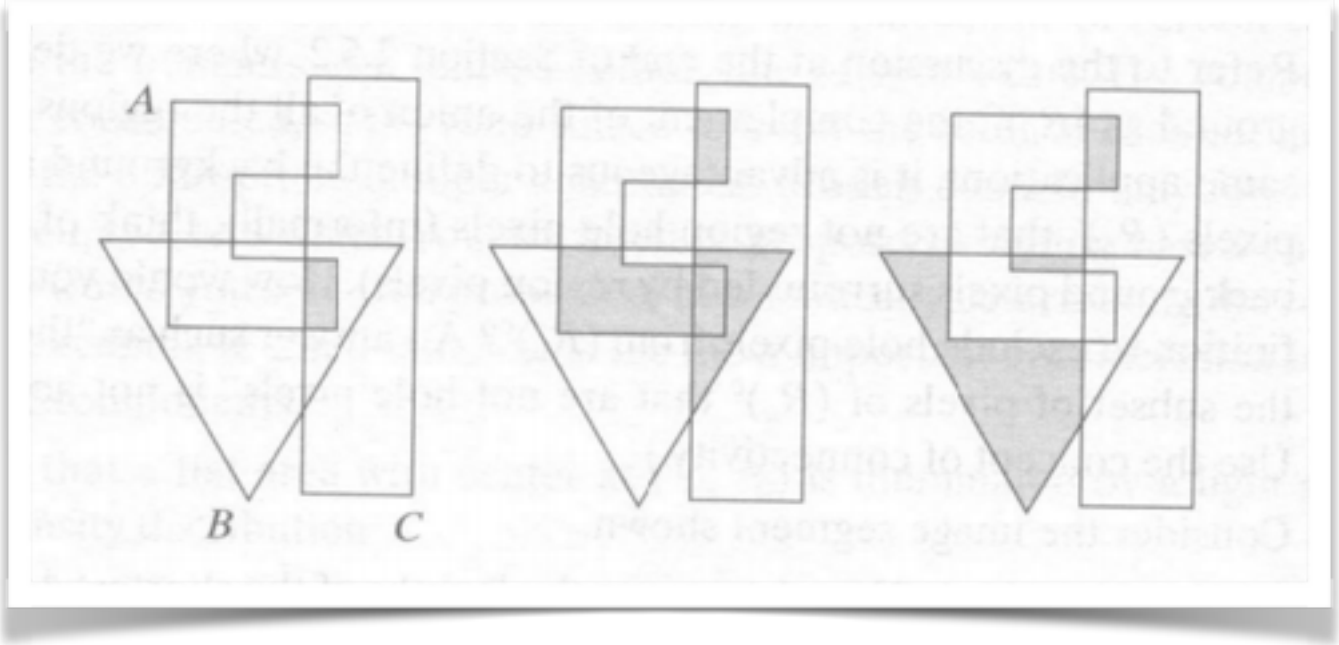


Figure 2: Logical Operations.

1. **A ∩ B ∩ C**
2. **(A ∩ C) ∪ (A ∩ B) ∪ (B ∩ C)**
3. **B ∪ (A ∩ B) - C - A**

2 Programming Tasks

Write programs to finish the following two tasks, and answer questions in your report. Don't forget to submit all relevant codes.

2.1 Pre-requirement

Input Please download the archive "hw1.zip", unzip it and choose the image corresponding to the last two digits of your student ID. This image is the initial input of your programming tasks in HW1. For example, if your student ID is "12110563", then you should take "63.png" as your input. You can convert the image format (to BMP, JPEG, ...) via Photoshop if necessary.

Make sure that you have selected the correct image. Misusing images may result in zero scores.

Language Any language is allowed, but you can only use element-wise indexing and scalar operations. Advanced indexing (including but not limited to row-wise/col-wise/range indexing) and vectorization are forbidden. As an example, given a matrix A, if you want to divide all its elements by 2, you should use loops over matrix elements, sequentially performing "A[i][j] /= 2", instead of directly writing "A /= 2" in python with numpy. This "annoying" requirement will be removed in future homework.

Others There remain some issues that you should pay attention to:

1. You can use third-party packages for loading/saving images.

2. Good UX (User Experience) is encouraged, but will only bring you negligible bonuses. Please don't spend too much time on it, since this is not an HCI course.

3. Keep your codes clean and well-documented. Bad coding styles will result in 20% penalty at most.

2.2 Scaling (45 Points)

Write a function that takes a gray image and a target size as input, and generates the scaled image as output. The function prototype is “scale(input img, size) ! output img”, where “input img” and “output img” are two-dimensional matrices storing images, and “size” is a tuple of (width, height) defining the spatial resolution of output. You can modify the prototype if necessary.

For the report, please load your input image and use your “scale” function to:

我的学号为13331270，因此选择图片70.png。这个程序已经打包成.jar给出，可以利用它压缩任意一张灰度图到任意的空间分辨率，同时对于2.3，这个程序可以处理任何图片的灰度量化变换。

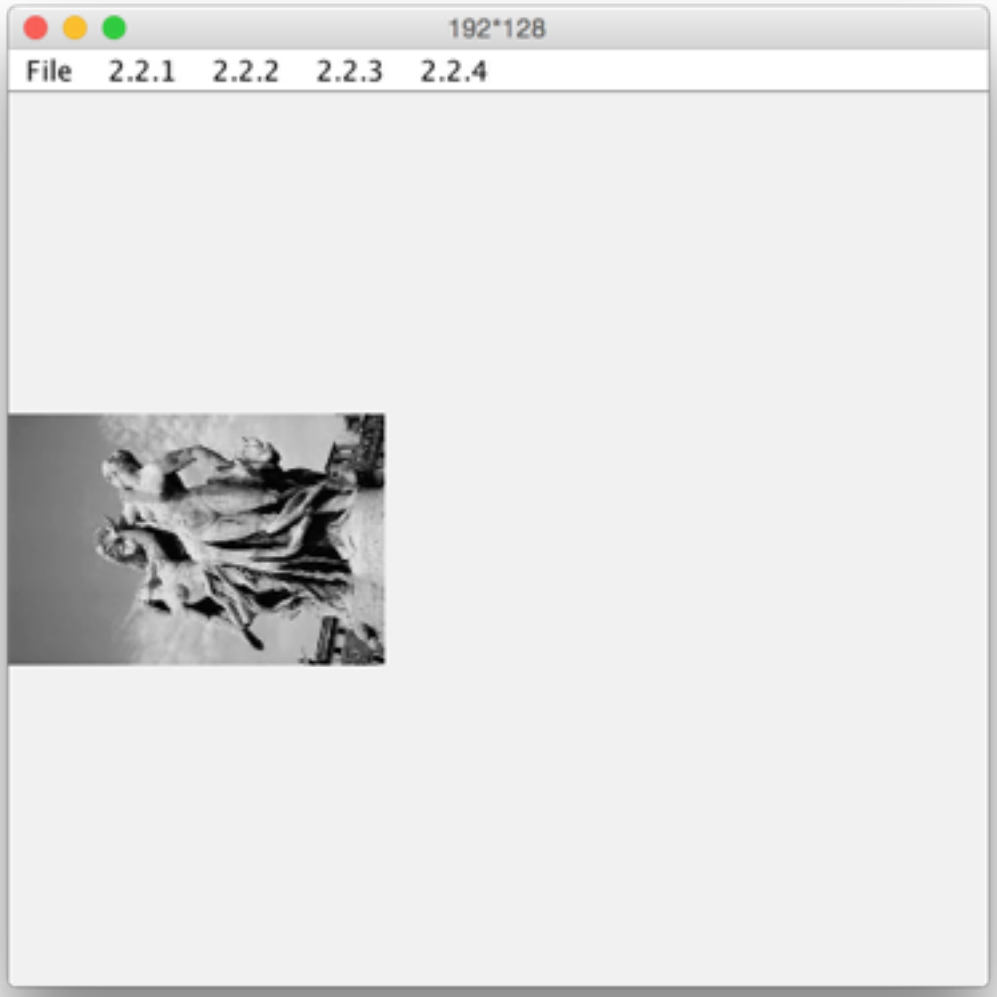
有关第二部分程序的分析，我放在了我的github上。

https://github.com/wujr5/digital-image-processing

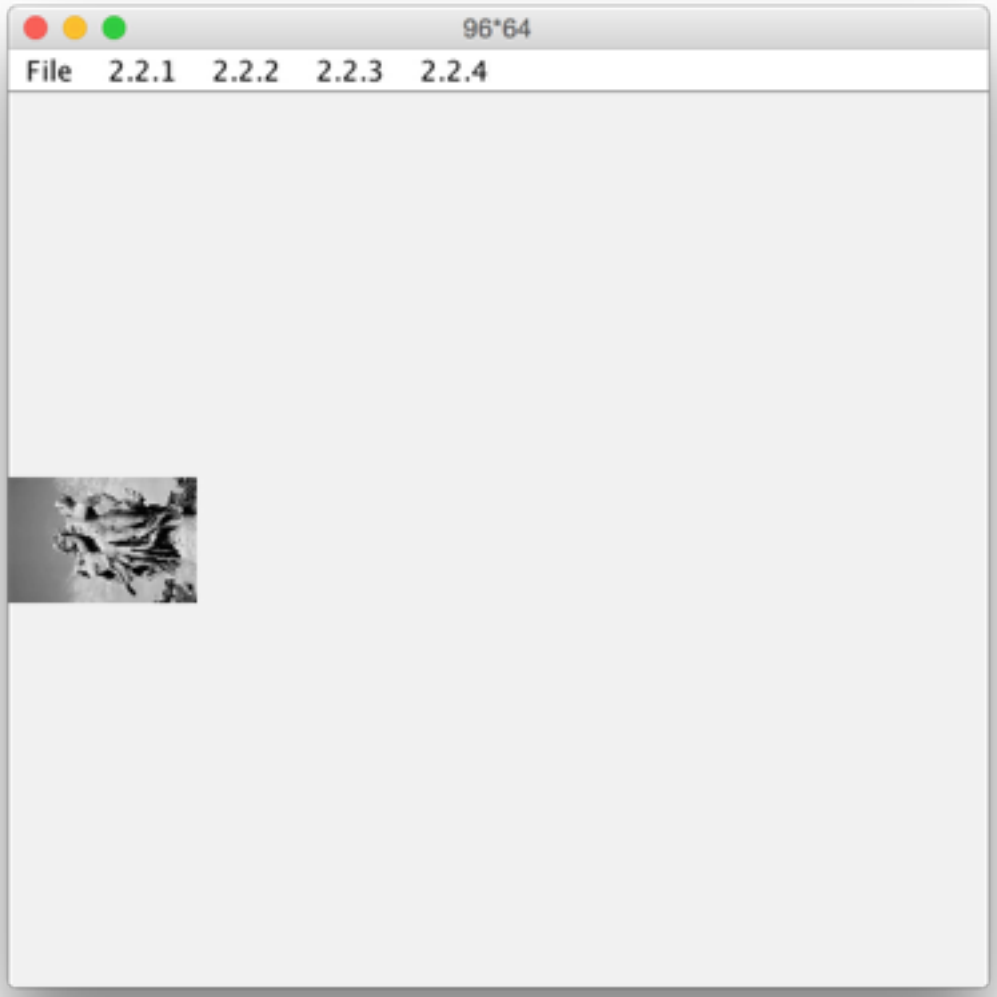
用户界面程序如下：

1. Down-scale to 192 ↦ 128 (width: 192, height: 128), 96 ↦ 64, 48 ↦ 32, 24 ↦ 16 and 12 ↦ 8, then manually plot your results in the report. (10 Points)

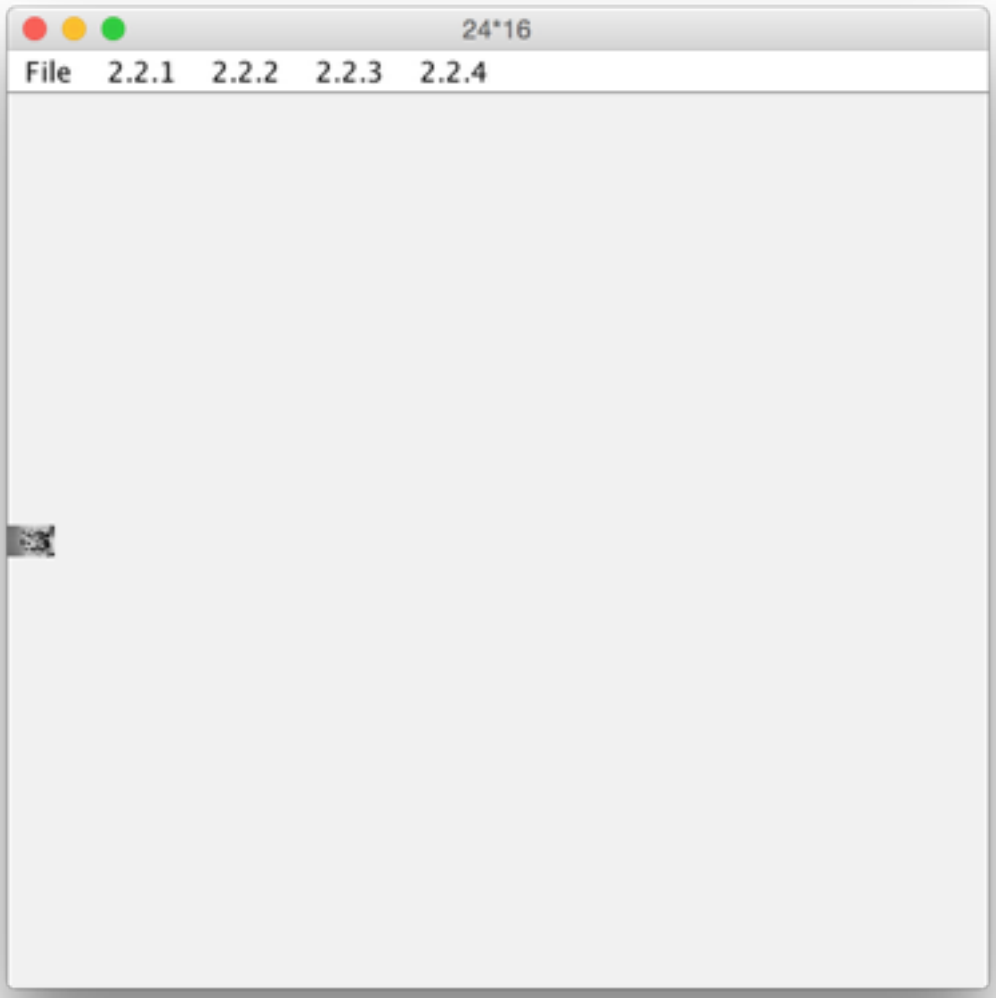
192*128:



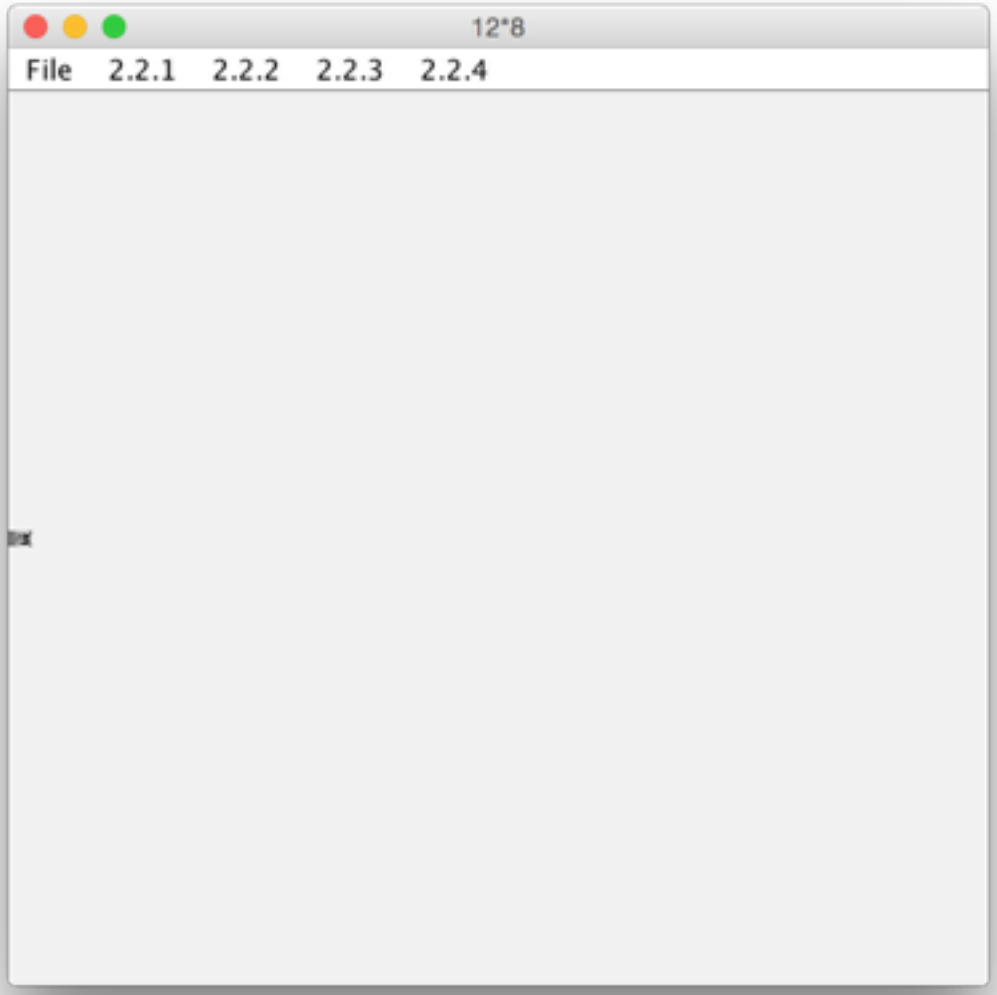
96*64：



24*16



12*8



2. Down-scale to 300 ↦ 200, then plot your result. (5 Points)

300*200

3. Up-scale to 450 ↠ 300, then plot your result. (5 Points)

450*300

4. Scale to 500 ↦ 200, then plot your result. (5 Points)

500*200

5. Detailedly discuss how you implement the scaling operation, i.e., the "scale" function, in less than 2 pages. Please focus on the algorithm part. If you have found interesting phenomenons in your scaling results, analyses and discussions on them are strongly welcomed and may bring you bonuses. But please don't widely copy/paste your codes in the report, since your codes are also submitted. (20 Points)

Attention: you should not rescale your scaling results in your report, unless they exceed the page height/width.

程序解析：

使用Java写的图像处理程序好像要比用Matlab写的复杂一点。原因是图像使用Java的imageIO还有BufferedImage解析出来的对象，存储图像RGB的方式是，每一个像素用4bytes的大小存储。可以利用int类型进行读写，但是解析的时候是根据每一个byte来解析的。比如：最前面的4个byte。第一个byte表示R的值，第二个是G，第三个是B，第四个是A。

因此，处理灰度值的时候就要综合考虑这几个值。比起Matlab直接操作灰度值要复杂一点。

主函数如截图：

```
public int[] imgScale(int[] inPixelsData, int srcW, int srcH, int destW, int destH) {
  double[][][] input3DData = processOneToThreeDemension(inPixelsData, srcH, srcW);

  int[][][] outputThreeDemensionData = new int[destH][destW][4];
  float rowRatio = ((float)srcH) / ((float)destH);
  float colRatio = ((float)srcW) / ((float)destW);

  for(int row = 0; row < destH; row++) {

    double srcRow = ((float)row) * rowRatio;
    double j = Math.floor(srcRow);
    double t = srcRow - j;

    for(int col = 0; col < destW; col++) {

      double srcCol = ((float)col) * colRatio;
      double k = Math.floor(srcCol);
      double u = srcCol - k;

      double coffiecent1 = (1.0d - t) * (1.0d - u);
      double coffiecent2 = (t) * (1.0d - u);
      double coffiecent3 = t * u;
      double coffiecent4 = (1.0d - t) * u;

      for (int i = 0; i < 4; i++) {
        outputThreeDemensionData[row][col][i] = (int)(
          coffiecent1 * input3DData[getClip((int)j, srcH - 1, 0)][getClip((int)k, srcW - 1, 0)][i] +
          coffiecent2 * input3DData[getClip((int)(j + 1), srcH - 1, 0)][getClip((int)k, srcW - 1, 0)][i] +
          coffiecent3 * input3DData[getClip((int)(j + 1), srcH - 1, 0)][getClip((int)(k + 1),srcW - 1, 0)][i] +
          coffiecent4 * input3DData[getClip((int)j, srcH - 1, 0)][getClip((int)(k + 1), srcW - 1, 0)][i]
        );
      }
    }
  }

  return convertToOneDim(outputThreeDemensionData, destW, destH);
}
```

其中，输入为：存储所有像素值得一维数组，图像原宽度，图像原高度，压缩后的宽度，压缩后的高度。

程序运行逻辑如下：

1. 读入以为像素数组，求出原图像和压缩后的图像的宽高比率
2. 根据求取的比率，来选择目的像素对应的最接近的原图像的像素
3. 利用双线性内插法求出目的像素的RGBA的值
4. 返回最终得到的目的图像所有像素值得一维数组

整个程序的关键之处在于双线性内插法的理解。

双线性内插法，是利用目的像素坐标，找到原图像浮点像素坐标。比如(j + c, k + u)，其中，j，k分别为浮点像素坐标的整数部分，c，u分别为对应的小数部分。

第一次线性内插：也就是在x方向上的内插值：

Q11 = f(j, k) * (1 - c) + f(j + 1, k) * c

Q22 = f(j + 1, k) * (1 - c) + f(j + 1, k + 1) * c

第二次线性内插：也就是在y方向上的内插值：

D(x, y) = Q11 * (1 - u) + Q22 * u

展开得到四个权重系数：

q1 = (1 - c) * (1 - u)

q2 = c * (1 - u)

q3 = (1 - c) * u

q4 = c * u

从而求出目的像素双线性内插得到的灰度值。

2.3 Quantization (28 Points)

Write a function that takes a gray image and a target number of gray levels as input, and generates the quantized image as output. The function prototype is "quantize(input img, level) ! output img", where "level" is an integer in [0, 255] defining the number of gray levels of output. You can modify the prototype if necessary.

For the report, please load your input image and use your "quantize" function to:

1. Reduce gray level resolution to 128, 32, 8, 4 and 2 levels, then plot your results respectively. Note that, in practice computers always represent "white" via the pixel value of 255, so you should also follow this rule. For example, when the gray level resolution is reduced to 4 levels, the resulting image should contain pixels of 0, 85, 170, 255, instead of 0, 1, 2, 3. (8 Points)

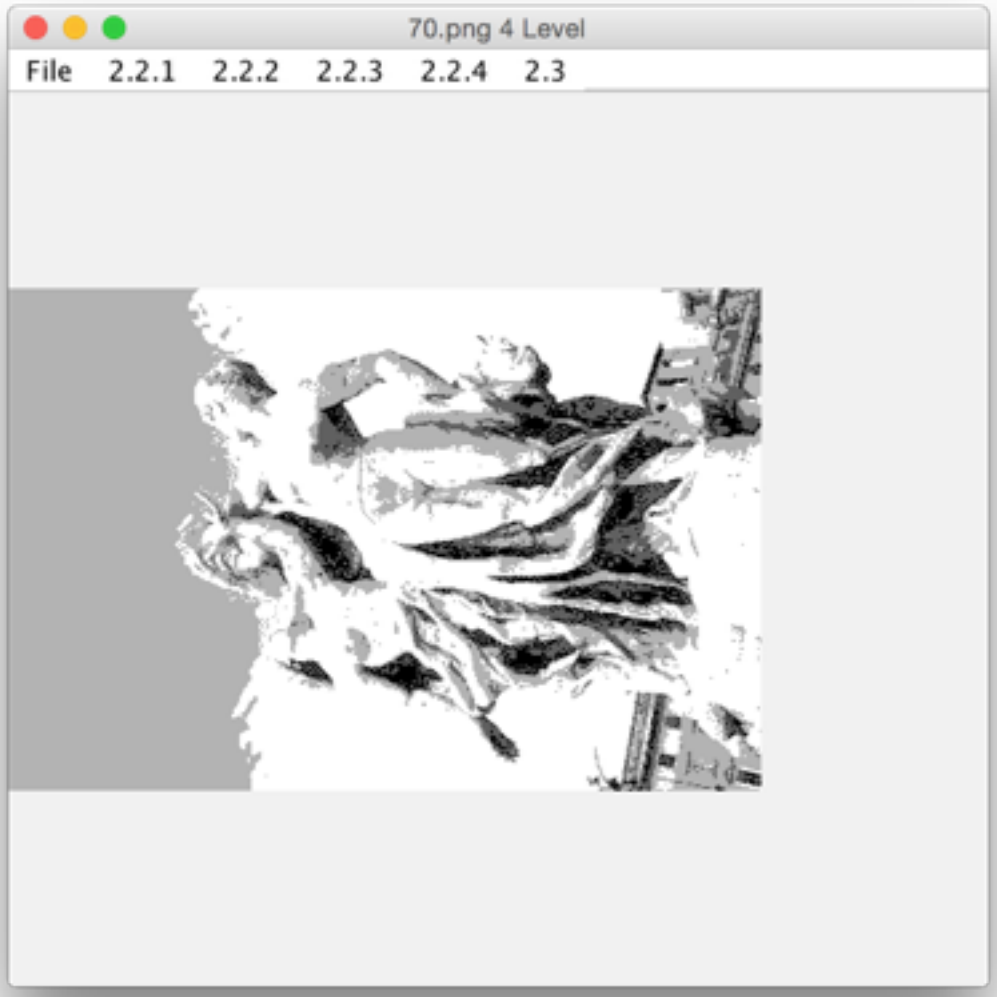
继续处理70.png

128 level:

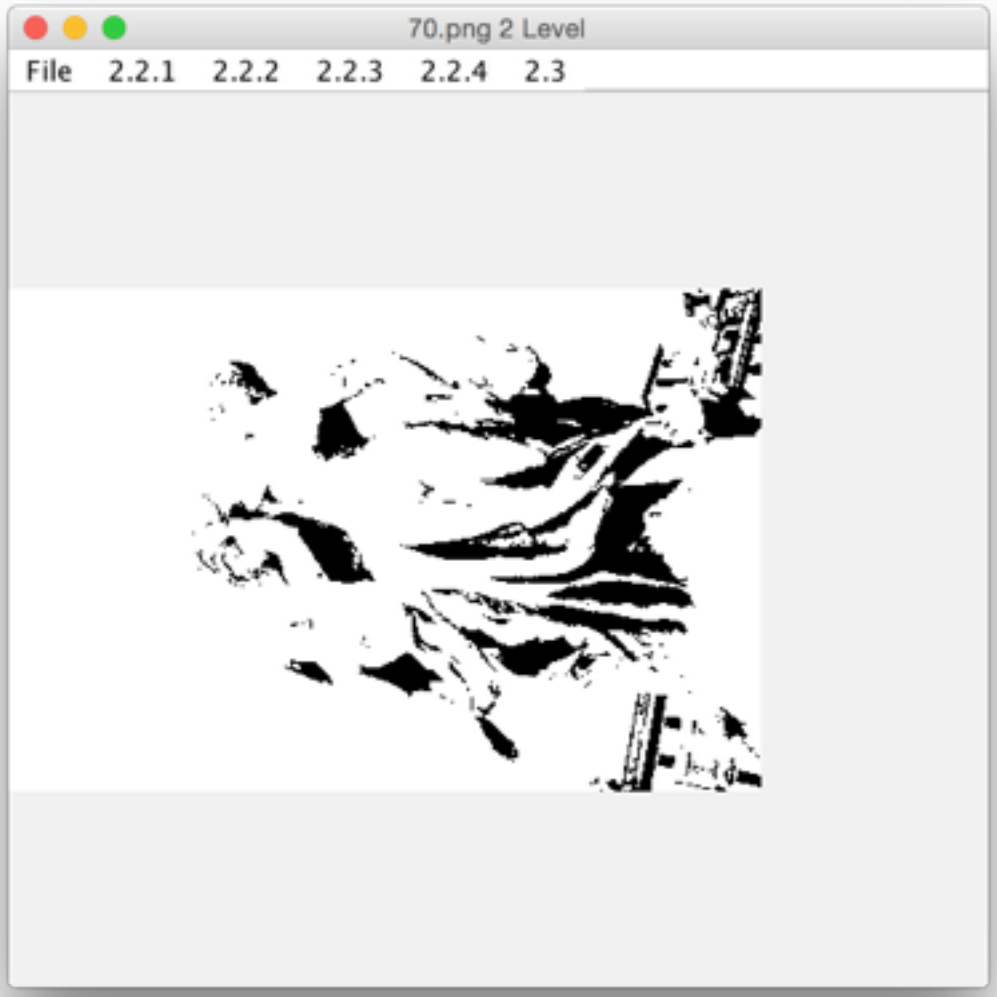
32 level:

8 level :

4 level :



2 level :



2. Detailedly discuss how you implement the quantization operation, i.e., the "quantize" function, in less than 2 pages. Again, please focus on the algorithm part. Analyzing and discussing interesting experiment results are also welcomed, but please don't widely copy/paste your codes in the report (20 Points).

主程序如下:

```
public int changeAGRBByLevel(int gray, int gap) {
  int A = (gray >> 24) & 0xFF;
  int temp = gray & 0xFF;

  double ratio = ((double)temp - (int)(temp / gap) * gap) / gap;
  if (ratio > 0.4) {
    temp = (temp / gap + 1) * gap;
    if (temp > 255) temp = 255;
  } else {
    temp = (temp / gap) * gap;
  }

  int grayARGB = ((A << 24) & 0xFF000000)
      | ((temp << 16) & 0x00FF0000)
      | ((temp << 8) & 0x0000FF00)
      | (temp & 0x000000FF);

  return grayARGB;
}

public BufferedImage quantize(BufferedImage img, int level) {
  int gap = (int) (256 / (level - 1));
  int width = img.getWidth();
  int height = img.getHeight();

  for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
      int grayARGB = img.getRGB(j, i);
      img.setRGB(j, i, changeAGRBByLevel(grayARGB, gap));
    }
  }
  return img;
}
```

主要逻辑是：

1. 将图片转化为灰度图
2. 取得传进来的level的值，计算得到gap
3. 对于每一个像素的灰度值，通过gap估算该像素点得灰度值更接近哪一个level的值
4. 根据估算重新给像素的灰度值赋值