HW1: Fundamentals

DIP homework

罗剑杰 15331229

1 Exercises

1.1 Storage

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 1024 × 2048, 128-level gray-scale image:

1. **How many bit planes are there for this image?**
   $128 = 2^7$, there are 7 bit planes for the image.
2. **Which panel is the most visually significant one?**
   The panel N is the most visually significant one. 因为 panel N 的灰度值最大，会使得图片的颜色最深。
3. **How many bytes are required for storing this image? (Don't consider image headers and compression.)**

$$1024 * 2028 = 2^{30}(个像素) = 2^{30} * 7bits = 7 * 2^{27} bytes$$

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.

Figure 1:

$$\begin{array}{ccccc} 3 & 4 & 1 & 2 & 0 \\ 0 & 1 & 0 & 4 & 2(q) \\ 2 & 2 & 3 & 1 & 4 \\ (p)2 & 0 & 4 & 2 & 1 \\ 1 & 2 & 0 & 3 & 4 \end{array}$$

1. 4-path

不存在对应的4-path。原因是q点的上下左右的元素都不在集合V中，没有办法连通出去。

2. 8-path

The shortest length is 4. One path is shown below:
p (3,0):2 -> (2,1):2 -> (2,2):3 -> (2,3):1 -> (1, 4):2 q

3. m-path

The shortest length is 5. One path is shown below:
p (3,0):2 -> (2,0):2 -> (2,1):2 -> (2,2):3 -> (2,3):1 -> (1,4):2 q

1.3 Logical Operations (3 * 3 = 9 Points)

Figure 2 are three different 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.

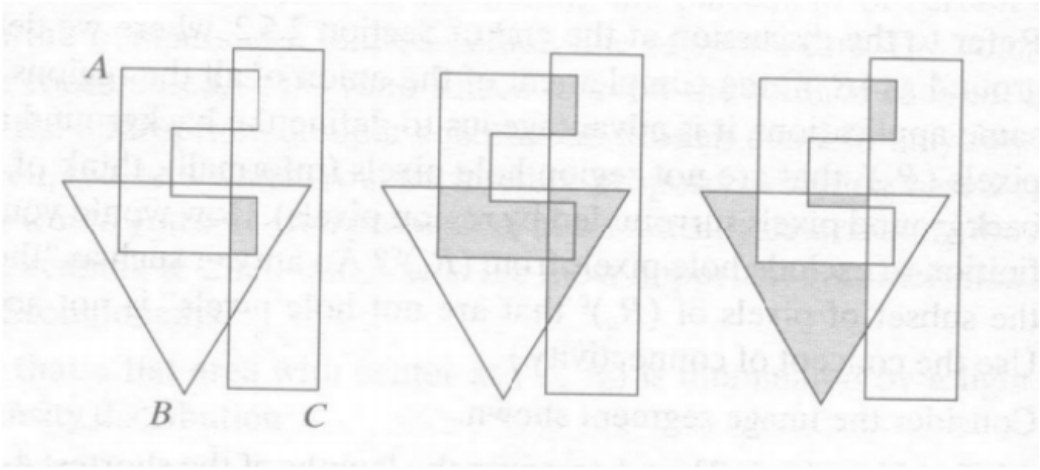


Figure 2: *Logical Operations.*

1. $A \cap B \cap C$
2. $(A \cap B) \cup (B \cap C) \cup (A \cap C)$
3. $(A \cap C - A \cap B \cap C) \cup (B - (A \cup C) \cap B)$

2 Programming Tasks

> Notice: The program is written in python 3.5.2, if TA can't run the program, please check your python version and `pip install -r requirements.txt`, thx.

2.2 Scaling

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

192 × 128

96 x 64

48 x 32

24 x 16

12 x 8

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

300 x 200

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

450 x 300

4. Scale to 500 × 200, then paste your result. (5 Points)

500 x200

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)

参考资料

算法实现：

> 注意，选择的参考坐标系原点在左上角

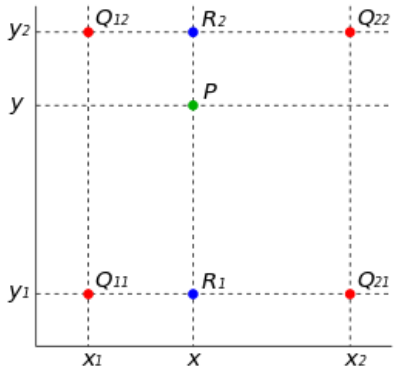
1. 创建一个目标大小的新图像 `newIm`。
2. 计算得到原图与结果图的宽和高各自的比例：

$$twidth = srcimg.width/desimg.width$$

$$theight = srcimg.width/desimg,width$$

3. 由新图像的某个像素$(h，w)$映射到原始图像$(h'，w')$处。通过 `get_src_point_info` 方法得到$(h'，w')$两个坐标值的整数部分(i,j)以及小数部分(u,v)，在下图显示中(i,j)即是点Q_{12}的坐标，$i = y_2, j = x_1, u = y - y_2, v = x - x_1$。

> 注意，选择的参考坐标系原点在左上角，与参考图片有点出入。



4. 根据i, j, u, v的值，利用双线性插值得到像素点(x，y)的值并写回新图像。双线性插值的一般表达公式是：

$$f(i+u,j+v)=(1-u)(1-v)f(i,j)+(1-u)vf(i,j+1)+u(1-v)f(i+1,j)+uvf(i+1,j+1)$$

但是其中为了避免出现数组越界的情况，在实际的编程应用中需要针对u, v的值使用不同的处理方式。

eg.

原图：

$$\begin{pmatrix} (0,0) & (0,1) & \cdots & (0,382) & (0,383) \\ (1,0) & (1,1) & \cdots & (1,382) & (1,383) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ (254,0) & (254,1) & \cdots & (254,382) & (254,383) \\ (255,0) & (255,1) & \cdots & (255,382) & (255,383) \end{pmatrix}$$

缩小至 192 x 128：

$$\begin{pmatrix} (0,0) & (0,1) & \cdots & (0,190) & (0,191) \\ (1,0) & (1,1) & \cdots & (1,190) & (1,191) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ (126,0) & (126,1) & \cdots & (126,190) & (126,191) \\ (127,0) & (127,1) & \cdots & (127,190) & (127,191) \end{pmatrix}$$

放大至 768 x 512：

$$\begin{pmatrix} (0,0) & (0,1) & \cdots & (0,766) & (0,767) \\ (1,0) & (1,1) & \cdots & (1,766) & (1,767) \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ (510,0) & (510,1) & \cdots & (510,766) & (510,767) \\ (511,0) & (511,1) & \cdots & (511,766) & (511,767) \end{pmatrix}$$

由于在python中下标是从0开始，为了使目标图的右上角对应原图的右上角，我们不能简单地使用$h' = h * theight$的转换公式，因为下标不同，转换后并不能正好对应到角落点坐标。针对这个问题作出修改，采用$h' = (h+1) * theight - 1$的公式后，角落的点可以正确映射。另外，对于所有的点，我们不能单纯地使用

$$f(i+u,j+v)=(1-u)(1-v)f(i,j)+(1-u)vf(i,j+1)+u(1-v)f(i+1,j)+uvf(i+1,j+1)$$

这个变换公式，因为映射到矩阵的最右端或者最下端如果没有判断而直接调用上述公式的话就会出现数组越界的情况。针对所有可能的情况根据u， v的值采用下面的变换公式，它们在本质上都是上述双线性插值公式的变形，但是可以有效避免越界问题。

u	v	处理
= 0	= 0	直接取原图上的点(i, j)的值
= 0	!= 0	在横向上进行一次插值(就是把一般式u=0的项给去掉)
!= 0	= 0	在竖向上进行一次插值(就是把一般式v=0的项给去掉)
!= 0	!= 0	调用一般的双线性插值公式

5. 重复步骤4直到新图像的所有像素写完。

2.3 Quantization (28 Points)

1. Reduce gray level resolution to 128, 32, 8, 4 and 2 levels, then paste 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)

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).

算法实现：

1. 确定好每个新的灰度级里共有多少个旧的灰度级的灰度值，该值为 `size_per_level` 。
2. 对于排在最后的灰度值因个数问题没能组成一个新的灰度级的那些灰度值，归并到它们前一个灰度级里，也就是说最后一个灰度级的灰度值个数可能会比 `size_per_level` 要大。
3. 对于第一个和最后一个新的灰度级，分别统一令新的灰度值为0和255，对于中间的新的灰度值，采用该灰度级内的所有旧的灰度值的平均数。构建出一个以 `（新的灰度级：计算机中对应的灰度值）` 为元素的字典 `levelgray` 。
4. 创建一个与原图一样大小的新图像 `newIm` 。
5. 通过for循环遍历所有像素点，将像素点通过 `int(input_img[h][w] / size_per_level)` 映射到新的灰度级，然后到 `levelgray` 里面去查找它所属于的新的灰度值是多少，写回 `newIm` 中。