Report of HW1: Fundamentals

15331314 吴林蔓

1.1 Storage

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

128-level gray-scale image is composed of 7 bits. Thus, there are 7 bit planes for this image.

2. Which panel is the most visually significant one?

Bit plane 7.

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

Because there are 7 bit planes, and the size of each plane is 1024 x 2048. Thus, 1024*2048*7 = 14680064 bits = 14680064/8 bytes = 1835008 bytes.

1.2 Adjacency

1. There is no 4-path between p and q. Because the 4-neighbors of q don't include any one of values from V = {1,2,3}.

2. The shortest 8-path between p and q is 4 (p-2-3-1-q).

3. The shortest m-path between p and q is 5 (p-2-2-3-1-q).

1.3 Logical Operations

1. $A \cap B \cap C$

2. (A ∩ B) ∪ (B ∩ C) ∪ (A ∩ C)

3. (B-A-C) ∪ [(A ∪ C)-B]

2.2 Scaling

1. (a)192x128

(b)96x64

(c)48x32

(d) 24x16

(e) 12x8

2. 300x200

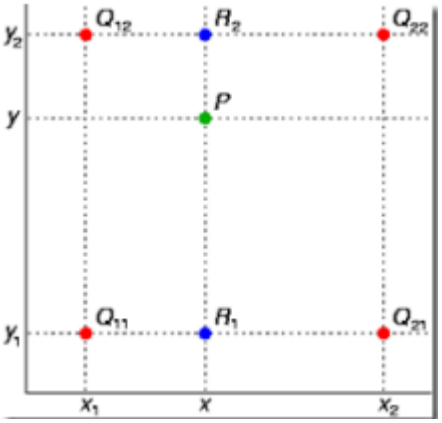
3. 450x300

4. 500x200

5. 这次作业的所有编程题都是在 matlab 平台上实现的，不经意间又感受到了 matlab 的强大，的确是一个能帮我们处理图片的强有力的工具。而这次的 scale 函数，用到了双线性插值法，核心思想是在两个方向分别进行一次线性插值。

即用 4 个最近邻去估计给定位置的灰度。目标图像的每个像素点，都可通过目标图像和源图像的边长比，映射到源图像上去。但映射到源图像的坐标却不一定是整数。然而在实际源图像上，每个像素点的坐标都是整数，因此若映射得到的对应源图像坐标为非整数，这个像素点其实是不存在的。而双线性插值法就是利用这个坐标值，寻找其 4 个最近邻（真实存在的像素点），通过推算的公式估计出待求像素点的值。

假设未知点为 P(x,y)，并已知其四个最近邻为$Q_{11}(x_{1,}y_1)$、$Q_{12}(x_{1,}y_2)$、$Q_{21}(x_{2,}y_1)$、$Q_{22}(x_{2,}y_2)$。且 x1,x2,y1,y2 均为整数。如下图所示。



根据维基百科上对双线性插值的推导，可得到公式：

$$f(x,y) \approx \frac{f(Q_{11})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y_2-y)+\frac{f(Q_{21})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y_2-y)$$
$$+\frac{f(Q_{12})}{(x_2-x_1)(y_2-y_1)}(x_2-x)(y-y_1)+\frac{f(Q_{22})}{(x_2-x_1)(y_2-y_1)}(x-x_1)(y-y_1).$$

易知（x2-x1）(y2-y1)=1。(x-x1)实际为 P 的横坐标 x 的小数部分，设为 u。(y-y1)实际为 P 的纵坐标 y 的小数部分，设为 v。进一步得(x2-x)=(1-u), (y2-y)=(1-v)。进一步简化上述公式得到：

f(x,y) ≈ (1-u)*(1-v)*f(Q_{11}) + u*(1-v)*f(Q_{21}) + (1-u)*v*f(Q_{12}) + u*v*f(Q_{22})

这个经过简化的公式也是我实现双线性插值法的核心内容。

通过遍历目标图像的每一个像素点，求出其映射在原图像的未知点 P(x,y)，最后通过上述公式来计算未知点 P 处对应的像素值 f(x,y)，即为目标图像中某像素点的像素值。

在这个实现过程中，matlab 中的 floor 和 ceil 函数帮助我很快找出 P 的 4 个最近邻的横纵坐标，并规避了边界容易出现的问题。除此之外，我发现 matlab 许多函数的第一个参数都是高，第二个才是宽。一开始我弄混了高与宽的顺序，导致变换出来的图像宽高颠倒了。其次，matlab 的语法规定矩阵的索引从 1 开始。因此在 up-scale 的时候，通过计算可能会访问到源图像矩阵中索引为 0 的的数据，此时就会出错。所以需要对这个地方进行一个小处理，处理代码如下：

```
if x==0 %Up-scale对边界处理，避免因访问矩阵索引0而出错
    x=1;
end

if y==0
    y=1;
end
```

（此处的 x,y 分别是未知点 P 横纵坐标的整数部分）

有关 scale 的函数的代码详细写在了 scale.m 。

2.3 Quantization

1. (a) 128levels

(b) 32levels

(c) 8levels

(d) 4levels

(e) 2levels

2. 对图像的量化我采用的是均匀量化（uniform quantization），即把输入图像的取值域等间隔分割的量化。我的算法就是先将灰度的取值域按照所需的灰度级来均匀划分， 例如量化到 4levels 就将灰度取值域[0, 255]均匀划分成 4 个区间 ,每个区间里的值都是离散的 ,因为灰度值为整数。因此划分就得到[0, 63]、[64, 127]、[128, 191]、[192, 255]这 4 个区间，即对源图像的像素点按其灰度值进行分组。通过 256/(level)即可求出区间长度 Length，再利用源图像某点的灰度值 f(i,j)除以区间长度再取整，就能很快算出该像素属于第几个区间。

除此之外，还需将每个区间的灰度值都映射到某个值上。这里量化的 level 的指的大概是量化后的图像不同灰度值的个数。而比较明确的是，0 和 255 都是必不可少的灰度值，无论是将图像量化到 128、32、8、4 还是 2levels，目标图像中的灰度值都应该有 0 和 255。就比如 4levels 时，目标图像中需要有 4 个灰度值，除了 0 和 255，还需要寻找 2 个灰度值。因此在 0 和 255 之间按比例划

分，找出剩余的 2 个值，即为 85 和 170。因此源图像和目标图像之间就成功构造了一个映射，源图像中灰度值为[0, 63]、[64, 127]、[128, 191]、[192, 255]的像素点的灰度值分别映射到目标图像的 0、85、170、255。其他 level 的量化也是类似这样，通过 255/（level-1）就能求出 0 之后的第一个目标灰度值。再乘上之前通过 f(i,j)/Length 算出的第几个区间 ,就能求出该像素点在目标图像中的灰度。即 output_img(i,j) = floor(f(i,j)/Length) * (255/(level-1))，通过这个小公式就能很快求出目标图像各个像素的灰度值。

对量化的实现的详细代码都在 quantize.m。其中计算区间的取整是通过 floor 函数实现的.

这次是第一次数图作业，虽然不是很难，但一开始还不知道如何下手。不过通过上网查找大量资料，总算是学会了缩放和量化图像。虽然过程中也遇到了一些小 bug（一般都是关于 matlab 规则的小问题），不过都很快解决了。希望下次能继续加油。