**《数字媒体技术基础》Hw1**

16340041 陈亚楠

**第一题**

**一、题目描述:**

Now suppose we wish to create a video transition such that the second video appears under the first video through a moving radius (like a clock hand). Write a formula to use the correct pixels from the two videos to achieve this special effect for the red channel.

**二、算法设计：**

题目的整体思想是：在一个图片中，有一个半径逐渐增大的圆，圆内显示lena.jpg同一个位置的像素，而圆外显示诺贝尔.jpg同一个位置的像素，随着该圆半径的增大，实现类似于相机光圈的动态覆盖效果。

给出算法伪代码如下：

// xMax, yMax 表示图片坐标轴上的最大范围

// rMax 表示圆的最大半径

// factor 用来控制圆半径逐渐增大

rMax = sqrt( xMax/2 ^ 2 + yMax/2 ^ 2 )

factor初始化为0

for (x, y)

// 当前控制的圆的半径

radius = rMax \* factor / 10

//计算该点到圆心的距离

dis = sqrt((x - xMax/2)^2 + (y - yMax/2)^2)

if dis < radius

img(x, y) = lena(x, y)

else

img(x, y) = nobel(x, y)

// 控制factor逐渐增大

factor ++

**三、程序实现：**

该实现程序由python3.7语言编写，使用了第三方库PIL，程序关键部分代码如下:

1. nobelImg = Image.open("诺贝尔.jpg")
2. lenaImg = Image.open("lena.jpg")
3. xMax = nobelImg.width
4. yMax = nobelImg.height
5. rMax = math.sqrt(math.pow(xMax/2, 2) + math.pow(yMax/2, 2))
6. factor = 0
8. **while** factor <= 10:
9. **for** x **in** range(xMax):
10. **for** y **in** range(yMax):
11. # 计算图片切换半径
12. radius = math.sqrt(math.pow(x - xMax/2, 2) + math.pow(y - yMax/2, 2))
13. **if** radius < rMax \* factor / 10:
14. nobelImg.putpixel((x,y), lenaImg.getpixel((x, y)))
15. outName = str(factor) + ".jpg"
16. r, g, b = nobelImg.split()
17. # 保存为图片序列
18. r.save(outName, "JPEG")
19. factor += 1

源代码详见Problem1.py。

**四、实现效果：**

题目实现的图像序列如下：



**第二题**

**一、题目描述：**

For the color LUT problem, try out the median-cut algorithm on a sample image. Explain briefly why it is that this algorithm, carried out on an image of red apples, puts more color gradation in the resulting 24-bit color image where it is needed, among the reds.

**二、算法设计：**

该题目实际是实现中值切分算法，得到一个256行的颜色查找表，算法具体可以描述为以下几步：

① 新建一个colorTable，包含源图片的所有颜色；

② 根据colorTable中R或G或B值将该表进行排序；

③ 找到排序后的colorTable中R或G或B的中值，依据该值将colorTable一分为二；

④ 重复②、③步，划分8次后，将初始颜色空间划分为256个部分；本实验中，排序依据依次为：R、G、B、R、G、B、R、G；

⑤ 计算每个部分的R、G、B三值的平均值，得到一个新的RGB值，并将其添加到新建的目标颜色查找表LUT中；

⑥ 对于源图片中的每个像素，在新的颜色查找表中找到具有最短欧式距离的RGB值，并将其分配给该像素的RGB值。

**三、程序实现：**

该程序由python3.7语言编写，使用了第三方库PIL。

该程序总体设计分为4个功能函数和一个main函数，四个主要函数分别为：getColorTable(image)，用来获取源图像的所有颜色；medianCut(slice, times)，中值切分算法的实现部分，使用了递归的思想进行颜色空间的划分；getNewColor(cube)，用来在目标颜色查找表中找到具有最短欧式距离的颜色值；toNewImage(image)，用来生成目标8位彩色图像。

各个功能函数实现如下：

**（1）getColorTable(image):**

1. # 获取原图像的所有颜色
2. **def** getColorTable(image):
3. width = image.width
4. height = image.height
5. **for** x **in** range(width):
6. **for** y **in** range(height):
7. r, g, b = image.getpixel((x, y))
8. cube = (r,g,b)
9. colorTable.append(cube)

**（2）medianCut(slice, times)：**

1. # 中位切分算法
2. **def** medianCut(slice, times):
3. length = len(slice)
4. **if** times >= 8:
5. rSum = 0
6. gSum = 0
7. bSum = 0
8. **for** i **in** range(length):
9. rSum += slice[i][0]
10. gSum += slice[i][1]
11. bSum += slice[i][2]
12. newCube = (rSum / length, gSum / length, bSum / length)
13. colorLUT.append(newCube)
14. **return**
15. **if** times % 3 == 0:
16. slice.sort(key=sortByR)
17. **elif** times % 3 == 1:
18. slice.sort(key=sortByG)
19. **else**:
20. slice.sort(key=sortByB)
21. medianCut(slice[:length//2], times+1)
22. medianCut(slice[length//2:], times+1)

**（3）getNewColor(cube)：**

1. # 获取最短欧式距离的颜色
2. **def** getNewColor(cube):
3. length = len(colorLUT)
4. index = 0
5. dis = getDis(cube, colorLUT[0])
6. **for** i **in** range(length):
7. newDis = getDis(cube, colorLUT[i])
8. **if** newDis < dis:
9. index = i
10. dis = newDis
11. **return** colorLUT[index]

**（4）toNewImage(image)：**

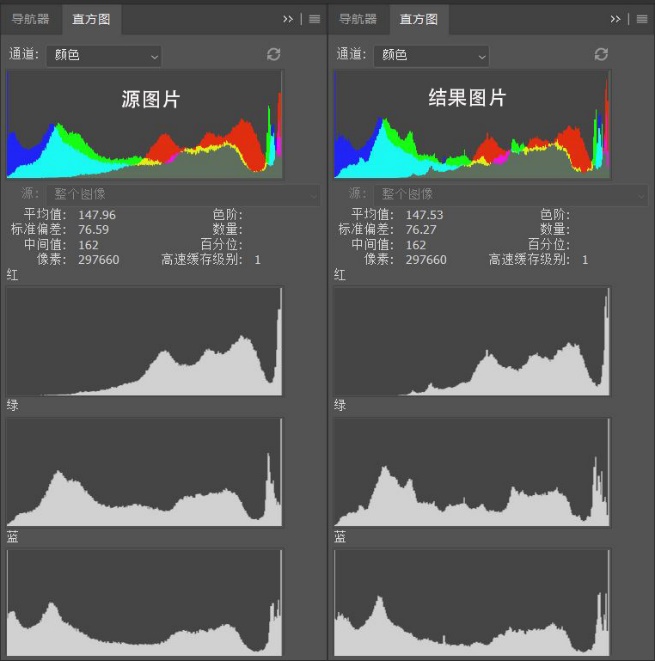
1. # 生成新的8位彩色图像
2. **def** toNewImage(image):
3. width = image.width
4. height = image.height
5. img = Image.new("RGB", (width, height))
6. **for** x **in** range(width):
7. **for** y **in** range(height):
8. r, g, b = image.getpixel((x, y))
9. cube = (r,g,b)
10. colorCube = getNewColor(cube)
11. img.putpixel((x, y), (int(colorCube[0]), int(colorCube[1]), int(colorCube[2])))
12. **return** img

源代码详见Problem2.py。

**四、实现效果：**



目标图片result.jpg

源图片与结果图片对比 直方图数据对比

将结果进行对比发现，生成的8位彩色图像与原图像在直方图数据上存在差异，但这种差异很小，并不影响人眼的直观视觉感受。