《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))
```

```
factor = 0

while factor <= 10:
    for x in range(xMax):
        for y in range(yMax):
            # 计算图片切换半径
            radius = math.sqrt(math.pow(x - xMax/2, 2) + math.pow(y - yMax/2, 2))
            if radius < rMax * factor / 10:
                nobelImg.putpixel((x,y), lenaImg.getpixel((x, y)))
    outName = str(factor) + ".jpg"
    r, g, b = nobelImg.split()
    # 保存为图片序列
    r.save(outName, "JPEG")
    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):

```
# 获取原图像的所有颜色
def getColorTable(image):
    width = image.width
    height = image.height
    for x in range(width):
        for y in range(height):
            r, g, b = image.getpixel((x, y))
            cube = (r,g,b)
            colorTable.append(cube)
```

（2）medianCut(slice, times)：

```
# 中位切分算法
def medianCut(slice, times):
    length = len(slice)
    if times >= 8:
        rSum = 0
        gSum = 0
```

```
        bSum = 0
        for i in range(length):
            rSum += slice[i][0]
            gSum += slice[i][1]
            bSum += slice[i][2]
        newCube = (rSum / length, gSum / length, bSum / length)
        colorLUT.append(newCube)
        return
    if times % 3 == 0:
        slice.sort(key=sortByR)
    elif times % 3 == 1:
        slice.sort(key=sortByG)
    else:
        slice.sort(key=sortByB)
    medianCut(slice[:length//2], times+1)
    medianCut(slice[length//2:], times+1)
```

（3）getNewColor(cube)：

```
# 获取最短欧式距离的颜色
def getNewColor(cube):
    length = len(colorLUT)
    index = 0
    dis = getDis(cube, colorLUT[0])
    for i in range(length):
        newDis = getDis(cube, colorLUT[i])
        if newDis < dis:
            index = i
            dis = newDis
    return colorLUT[index]
```

（4）toNewImage(image)：

```
# 生成新的 8 位彩色图像
def toNewImage(image):
    width = image.width
    height = image.height
    img = Image.new("RGB", (width, height))
    for x in range(width):
        for y in range(height):
            r, g, b = image.getpixel((x, y))
            cube = (r,g,b)
            colorCube = getNewColor(cube)
            img.putpixel((x, y), (int(colorCube[0]), int(colorCube[1]), int(colorCube[2])))
```

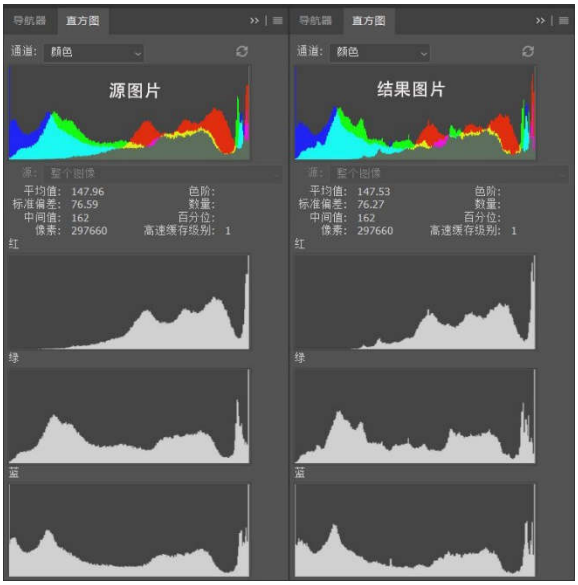
```
12.     return img
```

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。