数字媒体技术基础实验报告

姓名：陈梓峰 学号：16340050 实验名称：hw1

一、 实验内容

1. 实现前后两张给定图片的 Iris wipe 过渡效果(要求使用 R 通道)
2. 使用 Median-Cut algorithm 对给定的 24 位彩色图进行处理，得到一张 8 位彩色图

二、 算法描述

1. Iris wipe 过渡效果部分：Iris Wipe 过渡效果，实际上是在过渡期间，每一帧图像的四周部分由过渡前一张图构成，中心圆的部分由过渡后一张图构成。于是我们以下算法。
   a) 我们创建 25 张中间帧图像，每张中间帧图像持续 0.04 秒，整个过渡过程共 1s。
   b) 对于每张中间帧，有一个值r_i，遍历图像中的每一个像素点$\mathrm{p(x,y)}$，若它与中心像素点的欧几里得距离 d 有$\mathrm{d} \geq r_i$，则该点的值与过渡前的图像在该位置的像素值相同,$\mathrm{p(x,y)} = previous\_img(x,y)$，若$\mathrm{d} < r_i$则与过度后的图像在该位置的像素值相同, $\mathrm{p(x,y)} = next\_img(x,y)$。$r_i = \frac{i}{25} * R$, 其中 R 为图像对角线长度的一半$\mathrm{R} = \frac{\sqrt{\mathrm{width}^2 + height^2}}{2}$
   c) 使用 25 张中间帧生成 transition.gif
2. Median-Cut Algorithm 部分：

a) 将图像的所有像素点加入到一个区域

b) 对于每个区域，执行以下操作

   i. 计算出该区域内所有像素的 RGB 通道的极差

   ii. 选出极差最大的通道

   iii. 根据该通道上值的大小对区域内的像素点进行排序

   iv. 将排序后前一半的像素和后一半的像素分给到两个不同的区域

c) 重复步骤 b，直到最后所有的像素点被分到 256 给不同的区域当中

d) 将每个区域的像素 RGB 值平均，得到 256 个 RGB 24 位颜色，组成一个颜色查找表（LUT）

e) 对原图所有的像素点执行以下操作

   i. 计算该像素点 RGB 值与颜色查找表 LUT 中所有颜色的欧几里得距离

   ii. 找到颜色查找表 LUT 中距离最近的颜色，将 8 位图（使用 LUT 作为调色板）中该位置的值改为距离最近的颜色的下标。

f) 经过 e 后得到一种 8 位彩色图，这就是我们所求的图。

三、 实验代码

1. Iris Wipe 部分，使用 PIL 库来进行图像的处理，使用 imageio 库生成 gif。

   a) 读入图片并提取 R 通道

```
# open images
imgA = Image.open("assets/诺贝尔.jpg")
imgB = Image.open("assets/lena.jpg")

# get the red channel
Ar, Ag, Ab = imgA.split()
Br, Bg, Bb = imgB.split()
pre_image = Ar
nxt_image = Br
```

b) 根据上述算法生成 25 张中间帧

```
# produce 25 frames
for i in range(25):
    cur_image = pre_image.copy()
    # set pixel of the current frame image
    for x in range(0, width):
        for y in range(0, height):
            vector = numpy.array([x, y])
            # calculate the distance between current position and center position
            if numpy.linalg.norm(center - vector) < i / 25 * radius:
                # set pixel
                cur_image.putpixel((x, y), (nxt_image.getpixel((x, y))))
    # save the frames to the disk
    cur_image.save("frames/{0}.jpg".format(i))
    # store the frames
    frames.append(numpy.array(cur_image))
```

c) 生成最终 gif

```
# produce a gif using the frames
output_file_name = "transition.gif"
imageio.mimsave(output_file_name, frames, 'GIF', duration=0.04)
```

2. Median_Cut 部分：

a) 根据上述算法的 median-cut algorithm 的实现

```python
def median_cut(current_lis, depth):
    if depth == 8:
        # store all colors(RGB) of this part into all_color
        all_color = []
        for coordinate in current_lis:
            x = coordinate[0]
            y = coordinate[1]
            all_color.append(img.getpixel((x, y)))

        # get the average RGB value
        new_color = np.array(all_color).mean(axis=0)

        # add the average RGB value into LUT
        LUT.append(new_color)
        LUT_linear.append(int(new_color[0]))
        LUT_linear.append(int(new_color[1]))
        LUT_linear.append(int(new_color[2]))
        return

    rgb_max = [0, 0, 0]
    rgb_min = [256, 256, 256]

    # get the max and min rgb in current part
    for coordinate in current_lis:
        x = coordinate[0]
        y = coordinate[1]
        rgb_pixel = img.getpixel((x, y))

        for i in range(3):
            rgb_max[i] = max(rgb_max[i], rgb_pixel[i])
            rgb_min[i] = min(rgb_min[i], rgb_pixel[i])

    # get the rgb range in current part
    rgb_range = rgb_max.copy()
    for i in range(3):
        rgb_range[i] -= rgb_min[i]

    # sort the pixel in the part according to the channel that vary most
    for i in range(3):
        if rgb_range[i] == max(rgb_range):
            current_lis = sorted(current_lis, key=lambda value: img.getpixel((value[0], value[1]))[i])
            break

    # spilt current part into 2 parts, and do the same thing in each new part
    median_cut(current_lis[0: int(len(current_lis) / 2)], depth + 1)
    median_cut(current_lis[int(len(current_lis) / 2):], depth + 1)
```

b) 根据欧几里得距离和 median-cut 所得 LUT 进行八位图转换

```python
# convert the 24-bit image to a 8-bit image using the LUT from median-cut algorithm
def to8bit():
    global img_8bit
    global LUT
    global LUT_linear
    # create a 8-bit image using the LUT
    img_8bit = img_8bit.convert(mode='P')
    img_8bit.putpalette(LUT_linear)

    LUT = np.array(LUT)
    # set the value of each pixel according to the LUT
    for i in range(img_8bit.size[0]):
        for j in range(img_8bit.size[1]):
            # calculate the distance between LUT and current pixel
            dis = distance(LUT, np.array(img.getpixel((i, j))))
            # find the nearest color in LUT
            nearest = 1000000
            mark = -1
            for idx, dist in enumerate(dis):
                if dist < nearest:
                    nearest = dist
                    mark = idx
            # set pixel
            img_8bit.putpixel((i, j), mark)

    # save the 8-bit image
    img_8bit.save("my_palette_manual.bmp")


# calculate the distance between LUT and current pixel
def distance(LUT, pixel):
    return np.sqrt(np.sum((LUT - pixel) ** 2, axis=1))
```

3. 完整代码见 transitio.py 和 median_cut.py

四、 实验效果

1. Iris Wipe 部分：

   a) 中间帧生成如下（部分）：

b) 最终生成的 gif 见 transition.gif

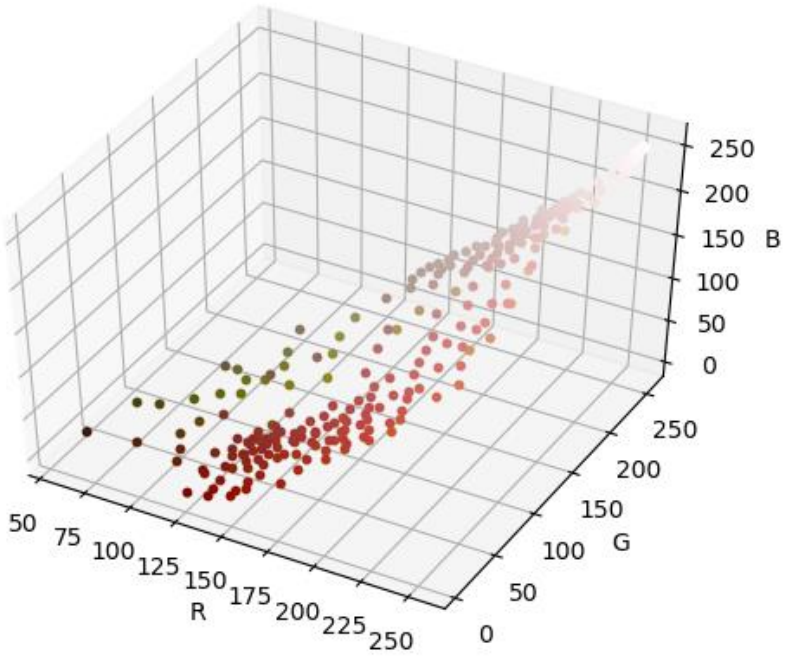
2. Median-Cut 部分：

a) 原图(见 redapple.jpg)

b) 生成的 8 位图(见 redapple_8bit.bmp)

五、 实验分析

1. Iris Wipe 部分：代码运行速度在普通 PC 上大约需要 40s 左右，程序约一半的时间花了像素点与中心点距离的计算这一句代码上 `if numpy.linalg.norm(center - vector) < i / 24 * radius:` ，进一步优化的话，应该从这里入手，考虑更高效的计算或者改用其他逻辑来避免距离的计算
2. Median-cut 部分：在计算欧几里得距离部分，如果使用朴素的循环求解，耗时大概需要 60s，针对这部分进行优化，使用了 numpy 当中的矩阵广播机制进行加速，可以较为快速地求出单个 pixel 与 LUT 表中每个颜色的距离，优化后耗时可以跑进 30s 以内。
3. Median-cut 算法生成的 256 色 LUT 中，颜色的分布如下图所示



其中红色色系和和白色色系居多，有少量的绿色和黄色，这个原图的颜色分布情况一致，说明了这是一个合理的算法。(原图的颜色大约有 90000 种，鉴于有限的计算机资源，没有画出原图的颜色分布)

4. pillow 库的 Image 对象提供一个 convert 接口，使用 mode='P'参数时，可以使用 pillow 提供的默认方法将图像转成 8bit 图像，下面是 pillow 方法生成的图（上, default_palette.bmp）和使用 median-cut + 欧几里得距离得到的图（下, redapple_8bit.bmp）的对比

放大进行局部对比，上边为 pillow 方法生成，下边为 median-cut+欧几里得距离生成。可以看到，pillow 方法生成的图有抖动效果，而 median-cut+欧几里得距离生成的图则没有抖动效果，查看 pillow 源码后发现 pillow 默认生成八位图的方法是采用了抖动处理。两种方法的视觉效果差异不大。