《多媒体技术实验报告》

（作业 1）

学院名称：　数据科学与计算机学院

专业方向：　计算机应用

学生姓名：　欧穗新

学号：　　16340173

时间：　　2018 年 10 月 16 日

第一题：

题目描述：

Suppose we wish to create a video transition such that the second video appears under the first video through an opening circle (like a camera iris opening), as in Fig. 2.22. Write a formula to use the correct pixels from the two videos to achieve this special effect. Just write your answer for the red channel.

思路分析&问题回答：

题目要求实现 nobel.jpg 到 lena.jpg 的切换，切换方式：
lena.jpg 从中间向四周扩大，覆盖范围是以图片中心为中心，随时间线性增大的 r=r(t)为半径的圆，直到 lena.jpg 完全覆盖 nobel.jpg。所以我们首先写出半径 r 关于 t 的函数：

$$r=r(t)=\frac{t}{t_{max}}*\sqrt{((\frac{height}{2})^2+(\frac{width}{2})^2)}$$

然后，得出图片的（x，y）处的像素点的像素值：

$$R(x,y)=\begin{cases} R_{lena}(x,y), \sqrt{(x-\frac{width}{2})^2+(y-\frac{height}{2})^2}<r(t) \\ R_{nobel}(x,y), \sqrt{(x-\frac{width}{2})^2+(y-\frac{height}{2})^2}>r(t) \end{cases}$$

其中 R 是最终的结果图片，为黑白图片，颜色值是一维的，R_{lena}是 lena.jpg 的红色通道颜色矩阵，R_{nobel}是 nobel.jpg 颜色矩阵（由于是黑白图片所以颜色值是一维的）；

算法描述：

1. 读入 lena.jpg 和 nobel.jpg，它们大小相同，将大小记为（宽，高）=（width，height）；
2. 填充每一帧图片的每一个像素值：
   For t 从 1 到 t_{max}，遍历时间直到切换完成
   For x 从 1 到 width，遍历被切换图片横坐标
   For y 从 1 到 height，遍历被切换图片纵坐标
   If $\sqrt{(x-\frac{width}{2})^2+(y-\frac{height}{2})^2}<\frac{t}{t_{max}}*\sqrt{((\frac{height}{2})^2+(\frac{width}{2})^2)}$
   R(x, y)= $R_{lena}(x,y)$; (像素点距离图像中心在当前切换半径范围之内时，该像素来自R_{lena}对应位置，即切换上去的图片)
   Else
   R(x, y)= $R_{nobel}(x,y)$; (像素点距离图像中心在当前切换半径范围之外时，该像素来自 nobel 图片对应位置，即被切换的图片)
3. 显示图片 R，即 imshow(R)，显示之后会随着 t 的变化不断刷新帧，从而实现图片切换

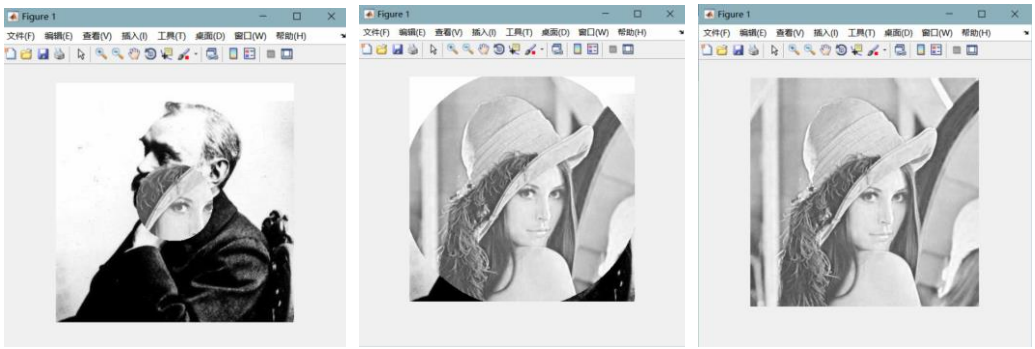
程序实现：（Task1.m）

本次实验自己采用 matlab 进行实现，以下是 Task1.m 代码：

```
figure(1);
img = imread('lena.jpg');
background = imread('Åµ±´¶û.jpg');
[h,w,c] = size(background);

img_r = zeros(h,w,3);
img_r(:,:,1) = img(:,:,1);
img_r = uint8(img_r);

center = [h/2, w/2];

for i = 1 : 5 : center(1)* (2^0.5)
    for j = center(1) - i : center(1) + i
        for k = center(2) - i : center(2) + i
            if ( (j-center(1))^2 + (k-center(2))^2 < i^2 && (j>0&&j<=h) && (k>0&&k<=w) )
                background(j,k,1) = img_r(j,k,1);
                background(j,k,2:3) = [0,0];
            end
        end
    end
   imshow(background);
end
```

实现效果：

下面给出过程的一些截图，详细的演示视频在 Task1.mp4 中



从截图/视频中的实现效果看来，我们完成了题目要求的切换，即 lena.jpg 从中间向四周扩大，覆盖范围是以图片中心为中心，随时间线性增大的 r=r(t)为半径的圆， 直到 lena.jpg 完全覆盖 nobel.jpg。

第二题：

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.
（必须使用本作业提供的图片**(redapple.jpg)**，其他图片无效）

思路分析&问题回答：

题目要求使用中值区分算法将给出的 24 位彩色图 redapple.jpg 转化为 8 位彩色图
所以我们按照书本给出的算法在 Task2.m 中写出具体的代码实现，将 redapple.jpg 转化为 8 位彩色图并输出即可
对 redapple.jpg 进行处理时，该算法在红颜色上进行更详细的梯度划分是因为：
redapple.jpg 这张图片的主体部分红苹果的主要颜色是红色，有大量的像素颜色都是红色的，所以使用中值区分算法之后，生成的 256 个区域中有大量的区域都是红色区域（因为是排序划分的区域，所以每个区域颜色都相近），所以最后的 256 种颜色中红色颜色占多数，即就是看起来该算法在红颜色上进行了更详细的梯度划分。
这种做法使用尽可能多的颜色来描述图像的主体部分（红苹果），在最需要的地方赋予最大的识别能力，从而使得主体尽可能少的损失。

算法描述：（自己使用的是书本上精简版本的中值区分算法）

1. 找出最小的颜色方形区，它包含图像中的所有颜色；
2. 先将所有不同的颜色按照红色字节排序，并且找到红色中值，红色字节比中值小的颜色其下标值的第一位标记为 0，红色字节比中值大的颜色其下标值的第一位标记为 1；
3. 然后在不同的分组中（目前有两个分组）将不同颜色各自按照绿色字节排序，并且找到绿色中值，比绿色字节比中值小的颜色其下标值的第二位标记为 0，绿色字节比中值大的颜色其下标值的第二位标记为 1；
4. 然后在不同的分组中（目前有四个分组）将所有不同的颜色按照蓝色字节排序，并且找到蓝色中值，蓝色字节比中值小的颜色其下标值的第三位标记为 0，蓝色字节比中值大的颜色其下标值的第三位标记为 1，此后分组数目变为 8 个；
5. 重复 2、3、4 步骤一次之后，再重复 2，3 步骤一次，这样就能够得到 256 个不同分组，而且每个像素也有了 8 位下标（下标的值就对应组号，一共 0-255 这 256 个下标值对应着 256 个分组），相应的 24 位颜色值是该像素颜色所处的最终小颜色立方体中心（该中心值取立方体中所有颜色的均值）；
   其中第二次经过 2 步骤之后，分组变为 16 个，第二次经过 3 步骤之后，分组变为 32 个，第二次经过 4 步骤之后，分组变为 64 个；并且此后得到了 5 位颜色下标值；
   其中第三次经过 2 步骤之后，分组变为 128 个，第三次经过 3 步骤之后，分组变为 256 个；

至此得到了最终的 8 位颜色下标值。

6. 根据第五步得到的 256 个分组建立颜色表，每个分组的组号就是颜色表中的下标值，该分组所有颜色的均值就是该下标值对应的 24 位颜色值（组号从 0-255 对应于下标 0-255，从而完成颜色表建立）；
7. 遍历图片中的所有像素点，根据每个像素的 8 位颜色下标值，在颜色表中找到对应的 24 位颜色，替代原来该像素点的颜色值，最后就得到 8 位彩色图。

程序实现：(Task2.m)

本次实验自己采用 matlab 进行实现，以下是 Task2.m 代码：

```
figure(1);
img = imread('redapple.jpg');
imshow(img);
[r,c,z] = size(img);
pixel_num = r*c;

% 将三维的img变为二维表示，其中行表示像素位置（(row-1) *width+col），列表示通道（r/g/b）
img_temp = zeros(r*c,4);
for i = 1:r
    for j = 1:c
        img_temp((i-1)*c+j,1:3) = img(i,j,1:3);
        img_temp((i-1)*c+j,4) = (i-1)*c+j;
    end
end

% 遍历八次，每次将分组变多两倍，最后得到256个分组
part = 1;
for i = 1:8
    % 每次循环part会变多一倍，表示下次划分时的组数，初始化为1
    if (i==1||i==4||i==7)
        cmp = 1;
    elseif (i==2||i==5||i==8)
        cmp = 2;
    elseif (i==3||i==6)
        cmp = 3;
    end
     for j = 1:part
         % 在这里面的每次循环需要将每个分组划分成两个分组，如果i==1||i==4||i==7，划分标准是第一列，i==2||i==5||i==8，划分标准是第二列，如果i==3||i==6，划分标准是第三列
        if (i == 1)
            img_temp(floor((j-1)/part*pixel_num+1):floor(j/part*pixel_num),1:4) =
sortrows(img_temp(floor((j-1)/part*pixel_num+1):floor(j/part*pixel_num),1:4));
        else
            img_temp(floor((j-1)/part*pixel_num+1):floor(j/part*pixel_num),1:4) =
sortrows(img_temp(floor((j-1)/part*pixel_num+1):floor(j/part*pixel_num),1:4), cmp);
        end
    end
    part = part*2;

    % 在第一次循环之后去重
    if (i==1)
        k = 1;
        img_unique = zeros(pixel_num, 4);
        img_unique(1,1:4) = img_temp(1,1:4);
        k = k+1;
        for j = 2:pixel_num
            if (img_temp(j-1,1) ~= img_temp(j,1) || img_temp(j-1,2) ~= img_temp(j,2) || img_temp(j-1,3) ~=
img_temp(j,3))
                img_unique(k,1:4) = img_temp(j,1:4);
                k = k+1;
            end
        end
        img_ori = img_temp;
        img_temp = img_unique(1:k-1,1:4);
        [pixel_num,~] = size(img_temp);

         % 将img_temp的下表更换为img_ori的下标
        start = 1;
        for j = 1:pixel_num
            img_temp(j,4) = start;
            while (start<=r*c
&&img_temp(j,1)==img_ori(start,1)&&img_temp(j,2)==img_ori(start,2)&&img_temp(j,3)==img_ori(start,3))
                start = start+1;
            end
        end
    end
end

% 分组完毕之后，一共256个分组，将每个组的r、g、b平均值作为代表颜色，并使用这三个均值替换原图中这个组的像素的rgb值
img = zeros(r,c,z); % 这里将img置为纯黑图片，足以知道新图片是重新由color_table中的颜色值生成的，因为代码中给img像素颜色赋值的只有color_table这个大小为256*3的矩阵中的行（在75行）
color_table = zeros(256,3);
for i = 1:256
    color_table(i,1:3) = mean(img_temp(floor((i-1)/256*pixel_num+1):floor(i/256*pixel_num),1:3));
end
```

```
for i = 1:pixel_num
    index_ori = img_temp(i,4);
    cur_color = img_ori(index_ori, 1:3);
    while (index_ori<=r*c&&cur_color(1) == img_ori(index_ori, 1)&&cur_color(2) == img_ori(index_ori,
2)&&cur_color(3) == img_ori(index_ori, 3))
        index = img_ori(index_ori, 4);
        img(floor((index-1)/c+1),mod(index-1,c)+1,1:3) = color_table(floor((i-1)/(pixel_num/256)+1),1:3);
        index_ori = index_ori+1;
    end
end
img = uint8(img);
figure(2);
imshow(img);
```

实现效果:

题目所给原图

使用中值取分算法处理得到的图片：（该图片中一共只有 256 种颜色）

从两张图片的结果对比来看，还是能够看出后面一张图片的颜色有所丢失：
红苹果、绿叶、花朵部分颜色丢失的很少，对比之下基本上看不到太多差别；
但是背景的白色部分发生变色，这一点比较明显，这是因为自己在代码中进行了颜色去重，

保证用于排序的颜色互不相同，这样能够使得最终得到的颜色表中 256 种颜色各不相同，从而尽可能多的保存原图中的颜色，但是这样做导致原来的白色值在和其它颜色值取平均值之后不再是完全的白色，而是发生了一点点变化，变成了 rgb=（254，248，245），导致背景的白色部分泛红。

所以说自己采用的书本上的中值区分算法能够将 24 位彩色图转化为 8 位彩色图，从而节省存储空间，同时带来图片质量的损失，但是这个损失并不是很大，特别由于中值区分算法对于颜色密集区会将梯度划分的更细致，在最需要的地方赋予最大的识别能力，使得图像的主体部分得到照顾，尽量少的发生质量损失。