多媒体技术第一次作业

第一题

题目描述

> 完成第一张图片(诺贝尔.jpg)到第二张图片(lena.jpg)的切换

编程工具和语言

Matlab

算法描述

先读取第一张图片，从图片的中心点p开始，半径从零开始，每过一定时间（自己设定）增加半径的长度，然后遍历像素矩阵，如果该像素到中心点的欧氏距离小于半径，则将该像素的RGB设置为第二张图片的RGB值。

程序实现

```
function transition()

img1 = imread('诺贝尔.jpg');
size1 = size(img1);
len = size1(1,1);
hei = size1(1,2);
img2 = imread('lena.jpg');
radius = 0;
point = zeros(1,2);
point(1,1) = len / 2;
point(1,2) = hei / 2;
max = sqrt((len / 2)^2 + (hei/2)^2);
disp(max);
while(radius <= max)
    for i = 1 : len
        for j = 1 : hei
            if((i-point(1,1))^2 + (j-point(1,2))^2 <= radius * radius)
                img1(i,j,:) = img2(i,j,:);
            end
        end
    end
    imshow(img1);
    pause(0.05);
    radius = radius + 1;
end
```

实现效果

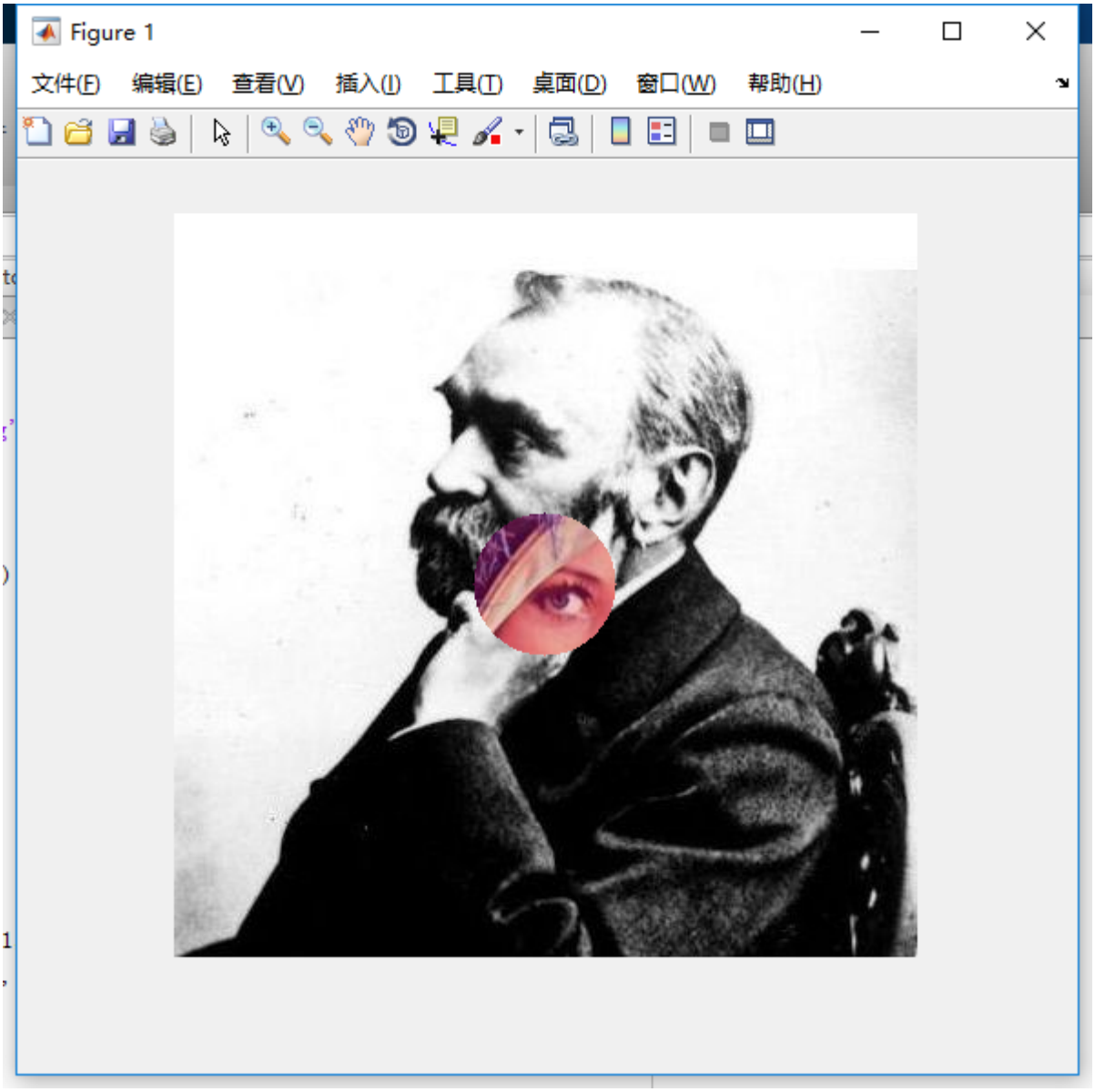
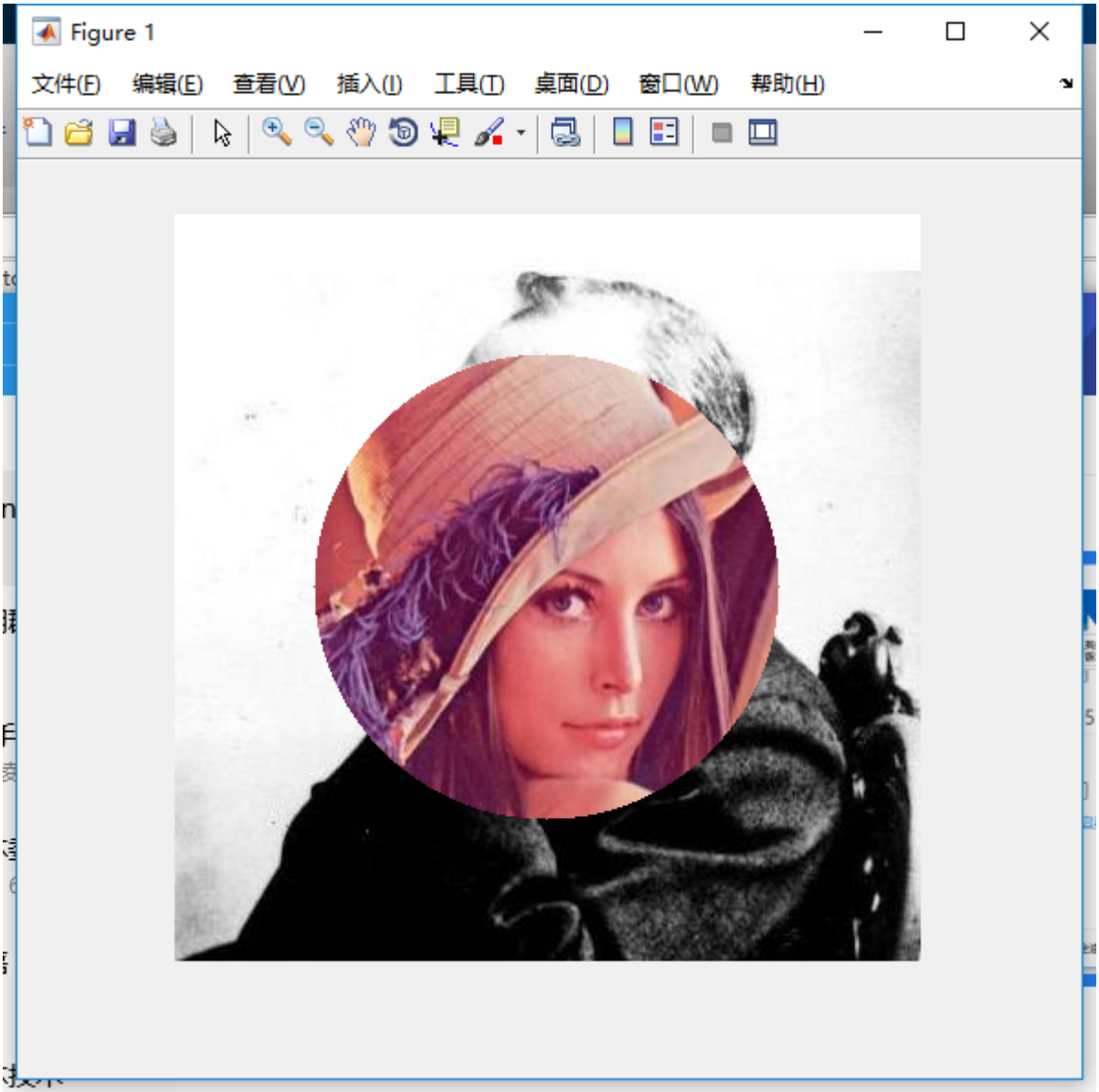


Figure 1
文件(F) 编辑(E) 查看(V) 插入(I) 工具(T) 桌面(D) 窗口(W) 帮助(H)



实验分析

实验结果为从图片一中心点开始，图片二慢慢显示，符合题目要求

第二题

题目描述

> 实现中值区分法（median-cut），解释该算法如何解决LUT问题

编程工具和语言

goland IDE 和 Golang语言

算法描述

分为以下几个步骤：

- 获取颜色

- 首先将所有颜色按照红色排序，取中值，将小于中值的划分为区间0，大于中值的划分为区间1，然后在区间0按照绿色排序划分。最后通过红、绿、蓝、红、绿、蓝、红、绿的顺序将所有颜色划分为256个区间
- 计算每个区间的R，G，B的平均值，并把该颜色添加到颜色查找表中
- 遍历每个像素，计算出像素与查找表中的颜色的欧氏距离，取距离最小的颜色作为像素的颜色，总共可以得到256种颜色，这样就可以使用八位数字表示整张图片。

程序实现

```
package main

import (
    "image"
    "image/color"
    "image/jpeg"
    "imgo"
    "math"
    "os"
    "sort"
)

/* variable */
var(
    colorTable []RGBcolor
)

/* type */
type RGBcolor struct {
    R int
    G int
    B int
}

type RGBcolors []RGBcolor

type sortR struct {
    RGBcolors
}

type sortG struct {
     RGBcolors
}

type sortB struct {
     RGBcolors
}
/* sort */
func (rgblist RGBcolors) Len() int {
    return len(rgblist)
}

func (rgblist RGBcolors) Swap(i, j int) {
    rgblist[i], rgblist[j] = rgblist[j], rgblist[i]
}
```

```
func (data sortR) Less(i, j int) bool {
    return data.RGBcolors[i].R > data.RGBcolors[j].R
}

func (data sortG) Less(i, j int) bool {
    return data.RGBcolors[i].G > data.RGBcolors[j].G
}

func (data sortB) Less(i, j int) bool {
    return data.RGBcolors[i].B > data.RGBcolors[j].B
}

func main() {
    img := imgo.MustRead("redapple.jpg")
    y := len(img)
    x := len(img[0])
    colorlist := make(RGBcolors, x*y)
    /* 获取像素RGB值 */
    for i := 0; i < x; i++ {
        for j := 0; j < y; j++ {
            r, g, b := img[j][i][0], img[j][i][1], img[j][i][2]
            colorlist = append(colorlist, RGBcolor{R:int(r),G:int(g),B:int(b)})
        }
    }

    /* 递归处理 */
    medianCut(colorlist[:], 0)
    newImg := image.NewRGBA(image.Rect(0,0,x,y))
    /* 计算欧式距离 */
    for i := 0; i < x; i++ {
        for j := 0; j < y; j++ {
            var min float64
            var cor RGBcolor
            r, g, b := img[j][i][0], img[j][i][1], img[j][i][2]
            for index, v := range colorTable {
                sum := math.Pow(float64(int(r)-v.R),2)
                sum += math.Pow(float64(int(g)-v.G),2)
                sum += math.Pow(float64(int(b)-v.B),2)
                dis := math.Sqrt(sum)
                if index == 0 || dis < min{
                    min = dis
                    cor = v
                }
            }
            newImg.SetRGBA(i, j, color.RGBA{R:uint8(cor.R), G:uint8(cor.G),
B:uint8(cor.B)})
        }
    }
    outputfile, _ := os.Create("newapple.jpg")
    jpeg.Encode(outputfile, newImg, &jpeg.Options{100})
}
```

```
func medianCut(data RGBcolors, colortype int){
    if(colortype == 9){
        // 已分成256个区间,计算平均值
        sumR, sumG, sumB := 0, 0, 0
        for _, v := range data {
            sumR = sumR + v.R
            sumG = sumG + v.G
            sumB = sumB + v.B
        }
        colorTable = append(colorTable, RGBcolor{
            R: sumR/data.Len(),
            G: sumG/data.Len(),
            B: sumB/data.Len() })
        return
    }
    switch colortype % 3 {
    case 0:
        sort.Sort(sortR{data})
    case 1:
        sort.Sort(sortG{data})
    case 2:
        sort.Sort(sortB{data})
    }
    length := data.Len()/2
    medianCut(data[:length], colortype+1)
    medianCut(data[length:], colortype+1)
}
```

实现效果

原图redapple.jpg

新生成图newapple.jpg

实验分析

中值区分法（median-cut）解决LUT问题的思想是，将颜色进行排序之后取中值，最后生成256个区间，这样相似的颜色就会集中到一个区间，并且颜色出现比较集中的区域也会更多的被包含到颜色查找表中，此外因为人眼对红色和绿色比较敏感，通过多对红色和绿色进行一次排序，可以使得相邻两个颜色查找表中的颜色跨度不那么大，就能有效的解决LUT问题。

通过比较可以发现，红色白色绿色出现的比较多，因此图上相应的位置上的压缩效果也比较好，但是苹果中间的颜色就出现了失真，这是因为黄色出现的比较少，因此在查找表上黄色部分的相邻两个颜色差别非常的大，因此就出现了以上的失真。