多媒体技术作业一

学号：16340261 姓名：徐伟元 专业：软件工程

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). Write a formula to use the correct pixels from the two videos to achieve this special effect.

算法描述与作答

从图片中心，以半径扩大的方式切换第二张图片。也就是说，对于任意时刻t，在圆内的像素点，都是切换的第二张图片的像素点，在圆外的则是第一张图片的像素点。那么，可以考虑将切换的半径视作时间t的函数，建立函数r = f(t)。其中，在t = 0时刻，r = 0；t = t_max时刻，r = r_max。那么，对于任意时刻t，它对应的半径r_t，可以利用分式推导出函数f(t)的表达式，即

$$\frac{r\_t}{r\_max} = \frac{t}{t\_max} \Rightarrow \text{r\_t} = r\_max \times \frac{t}{t\_max}$$

接下来，我们需要确定在圆内的像素点。我们将图片看作二维矩阵，其中每一项是图片的像素点。从左下角建立x − y坐标系，设图片宽为x_max，长为y_max，则变换圆心为$(\frac{x_max}{2}, \frac{y_max}{2})$，最大变换半径$r_max = \sqrt{(\frac{x_max}{2})^2 + (\frac{y_max}{2})^2}$。所以，在时刻t，变换半径$r_t = \text{r_max} \times \frac{t}{t_max} = \sqrt{(\frac{x_max}{2})^2 + (\frac{y_max}{2})^2} \times \frac{t}{t_max}$。

定义好了坐标系和半径-时间函数，算法也就呼之欲出了(*将 pixel 变为 R 通道，则是题目答案*)：

```
for t in (0, t_max) :
    if (x, y) in circle which center at () with radius r_t :
        show_pixel(x,y) = second_pixel(x,y)
    else :
        show_pixel(x,y) = first_pixel(x,y)
```

但是，考量一下算法，可以发现，判断圆外的写入操作是多余的。因为，如果图片的像素点开始就是第一幅图的像素点，那么这个写入操作就重复了，我们可以稍做改进：

```
show_pixel = first_pixel
for t in (0, t_max) :
    if (x, y) in circle which center at () with radius r_t :
        show_pixel(x,y) = second_pixel(x,y)
```

程序代码

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 29 2018
@author: xwy
@environment: python2.7
@dependency: opencv, numpy
"""

import numpy as np
import cv2 as cv
import math

noble = cv.imread('noble.jpg', cv.IMREAD_COLOR)
lena = cv.imread('lena.jpg', cv.IMREAD_COLOR)
result = cv.imread('noble.jpg', cv.IMREAD_COLOR)

if noble.shape[0] != lena.shape[0] or noble.shape[1] != lena.shape[1]:
    print("Different Size")
else:
    x_max = noble.shape[0]
    y_max = noble.shape[1]
    r_max = math.sqrt((x_max/2) * (x_max/2) + (y_max/2) * (y_max/2))
    t_max = 25
    for t in range(0, t_max + 1):
        r_t = math.floor(r_max * t / t_max)
        for x in range(0, x_max):
            for y in range(0, y_max):
                r = math.sqrt((x_max/2 - x) * (x_max/2 - x) + (y_max/2 - y) * (y_max/2 - y))
                if r < r_t:
                    result.itemset((x, y, 0), lena.item(x, y, 0))
                    result.itemset((x, y, 1), lena.item(x, y, 1))
                    result.itemset((x, y, 2), lena.item(x, y, 2))
        cv.imshow('transition', result)
        cv.waitKey(1)
    cv.waitKey(0)
    cv.destroyAllWindows()
```

实现效果

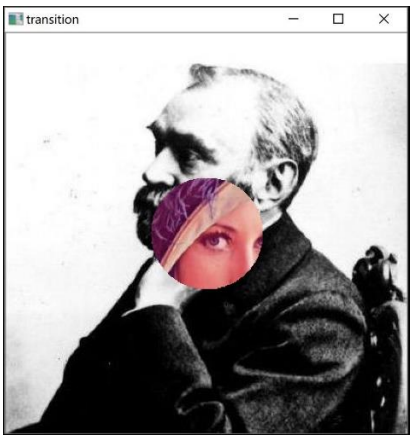

Figure 1 One

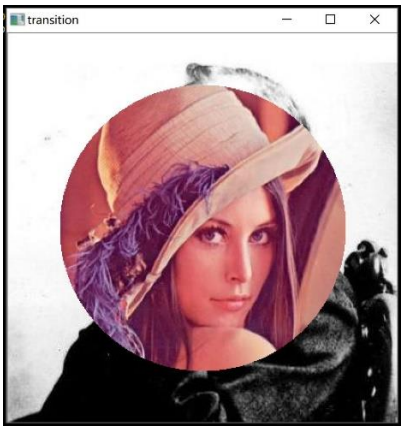

Figure 2 Two

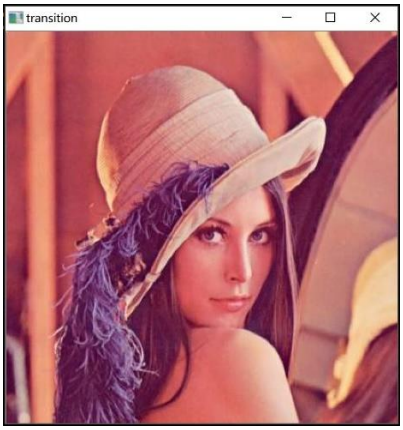

Figure 3 Three

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.

算法描述与作答

在 RGB 彩色空间中，以 R，G，B 为基轴构造空间坐标系。对于一张多彩色图，将其 RGB 构成彩色空间中的一个立方体，其中所有 RGB 三元组都对应于立方体上的一个点。

中值切分算法(Median-Cut Algorithm)是将图片的彩色立方体切成 256 个(或更少，但为 2 的整数次幂个)小立方体。小立方体的中心 RGB 坐标则由落在其中的像素点的 RGB 坐标的平均值决定，同时，这些像素点的新 RGB 由这个平均值决定。切割的思想为，图片像素点在上一次切割的基础上，交替在 RGB 通道上进行排序，并选择通道中值进行分割。

算法步骤如下:

1. 读入图片像素点，构造像素点链表
2. 根据 R/G/B 通道排序像素点链表
3. 寻找中值切分链表，小于中值则标记为 0，大于中值则标记为 1
4. 最大标记值小于 256，重复 2，3 步骤
5. 计算每个切割链表(小立方体)的中心 RGB 值，建立 LUT
6. 根据 LUT，对图片像素点重新分配 RGB 值

下面我们来看看细节。

1) 像素链表。图片载入后，使用像素矩阵存储。为了排序，我们需要将其构造成链表。同时，链表应该能储存原始像素的位置信息和新 LUT 的检索下标。所以，**构造存储元素为[[i，j]，[R，G，B]，index]的链表**，其中[i，j]为原始矩阵下标，[R，G，B]为对应 RGB 值，index 是新的 LUT 的检索下标;
2) 新 LUT 的检索下标。我们算法采用循环连接标记 0 和 1 来构造检索下标，但存储一个二进制数组是不必要的。我们考虑位运算，每次加入 0 或 1 等价于**原有的数乘以二，再加上 0 或 1**。所以，采用数字运算可以得到十进制的检索下标;
3) 根据前面的描述，我们的映射关系为：像素矩阵 <-> 像素链表 <-> LUT;
4) 256 个小立方体。可以发现，每次循环，我们的立方体个数翻一倍，也就是说，算法的最终形成的立方体个数是 2 的指数次幂的。所以，进行 8 次循环，可以构造出所需的 256 个小立方体。不必进行小立方体个数的判断。

对于题目的问题，因为红苹果图中，红色为主要颜色，采用中值区分算法，红色会占据大多数的小立方体，由此，更多的颜色等级被分配到红色中。

程序代码

```
# -*- coding: utf-8 -*-
"""
Created on Sat Aug 29 2018
@author: xwy
@environment: python2.7
@dependency: opencv, numpy
"""

from enum import Enum, unique
import numpy as np
import cv2 as cv
import math

class CV_RGBChannel(Enum):

    '''
    RBG Channel Value in OpenCV
    ATTENTION: BGR mode in OpenCV
    '''

    BLUE  = 0
    GREEN = 1
    RED = 2

def takeR(elem):

    '''
    select red channel
    '''

    return elem[1][1][CV_RGBChannel.RED.value]


def takeG(elem):

    '''
    select green channel
    '''

    return elem[1][1][CV_RGBChannel.GREEN.value]
```

```
def takeB(elem):

    '''
    select blue channel
    '''

    return elem[1][1][CV_RGBChannel.BLUE.value]


def sortColorSpace(list, channel):

    '''
    sort the list by color channel values\n
    :param {list}list  the list to be sorted\n
    :param {CV_RGBChannel/integer}channel  the color channel used as the base to sort color
space
    '''

    if channel == CV_RGBChannel.RED  or\
        channel == CV_RGBChannel.RED.value:
        list.sort(key=takeR)
    elif channel == CV_RGBChannel.GREEN  or\
        channel == CV_RGBChannel.GREEN.value:
        list.sort(key=takeG)
    elif channel == CV_RGBChannel.BLUE  or\
        channel == CV_RGBChannel.BLUE.value:
        list.sort(key=takeB)


def splitColorSpace(list,  channel):

    '''
    split sorted list by median value of color channel values\n
    :param {list}list    the color space to be split\n
    :param {CV_RGBChannel}channel    the color channel used to split color space
    '''

    if len(list) > 1:
        colorChannels = []
        for pixel in list:
            colorChannels.append(pixel[1][1][channel.value])

        half = len(colorChannels) // 2
        m = len(colorChannels) % 2
        median = 0
```

```
        if m > 0:
            median = colorChannels[half]
        else:
            median = (colorChannels[half] + colorChannels[half - 1]) / 2

        for pixel in list:
            if pixel[1][1][channel.value] >= median:
                pixel[1][2] = pixel[1][2] * 2 + 1
            else:
                pixel[1][2] = pixel[1][2] * 2 + 0



apple = cv.imread("redapple.jpg", cv.IMREAD_COLOR)
row, col, channel = apple.shape

# Initial color space
colorSpace = []
for i in range(0, row):
    for j in range(0, col):
        colorSpace.append([[i, j], apple[i, j], 0])

# Median-Cut Algorithm
for i in range(0, 8):
    temp = []
    for j in range(0, int(math.pow(2, i))):
        temp2 = []
        for index, pixel in enumerate(colorSpace):
            if pixel[2] == j:
                temp2.append([index, pixel])
        if i % 3 == 0:
            sortColorSpace(temp2, CV_RGBChannel.RED)
            splitColorSpace(temp2, CV_RGBChannel.RED)
        elif i % 3 == 1:
            sortColorSpace(temp2, CV_RGBChannel.GREEN)
            splitColorSpace(temp2, CV_RGBChannel.GREEN)
        elif i % 3 == 2:
            sortColorSpace(temp2, CV_RGBChannel.BLUE)
            splitColorSpace(temp2, CV_RGBChannel.BLUE)

        temp = temp + temp2

    for t in temp:
        colorSpace[t[0]][2] = t[1][2]
```

```
# Calculate LUT
temp = []
for i in range(0, int(math.pow(2, 8))):
    sum = [0, 0, 0]
    total = 0
    for pixel in colorSpace:
        if pixel[2] == i:
            sum += pixel[1]
            total = total + 1
    if total > 0:
        sum =    [sum[0]/total, sum[1]/total, sum[2]/total]
        temp.append([i, sum])

# Map origin image RGB with LUT
for t in temp:
    for pixel in colorSpace:
        if pixel[2] == t[0]:
            pixel[1] = t[1]

for i in range(0, len(colorSpace)):
        apple[colorSpace[i][0][0], colorSpace[i][0][1]] = colorSpace[i][1]

cv.imshow('median-cut', apple)
cv.waitKey(0)
cv.destroyAllWindows()
```

实现效果

Figure 4 RedApple_Origin

Figure 5 RedApple_Median-cut