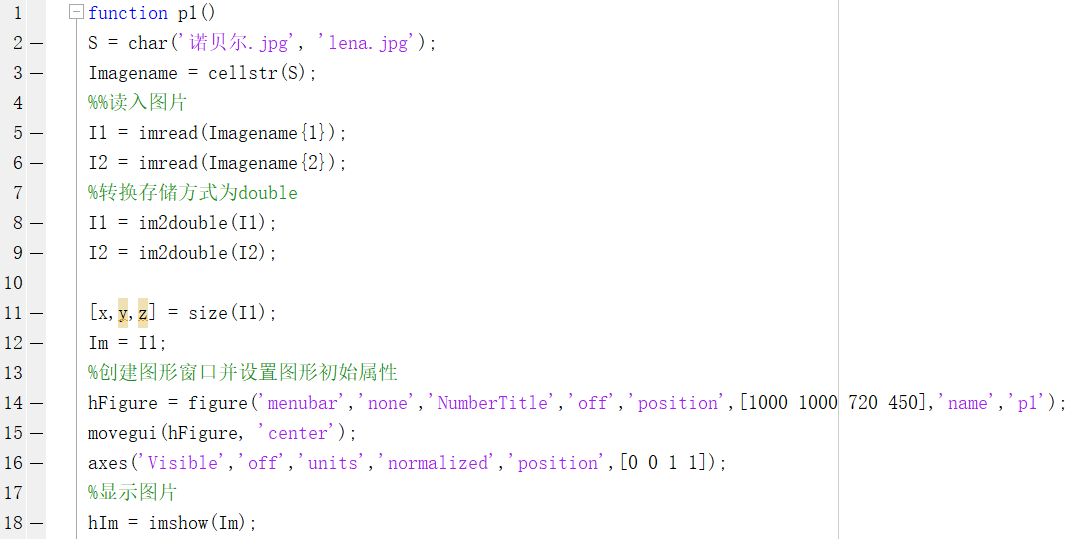
**多媒体第一次作业**

1. 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.

这道题需要实现的效果就是一个照片通过一个开放的圆圈慢慢显现覆盖另一个照片，这里我们需要做的就是实现这个效果。

这里我选择通过MATLAB来实现这个效果。在MATLAB中，用于数据可视和界面制作的基本绘图要素称为句柄图形对象，每个图形对象有相应的属性值。可以改变图形对象的属性值，重绘图形对象，从而创建程序动画。其基本思路是：首先新建一个图形窗口，再循环内逐渐改变图形对象的相应属性值，并使用drawnow函数更新当前图形，整个循环就会表现出变化的动画效果。

首先我们需要新建一个图形窗口并为窗口中对象设置一个初始属性：

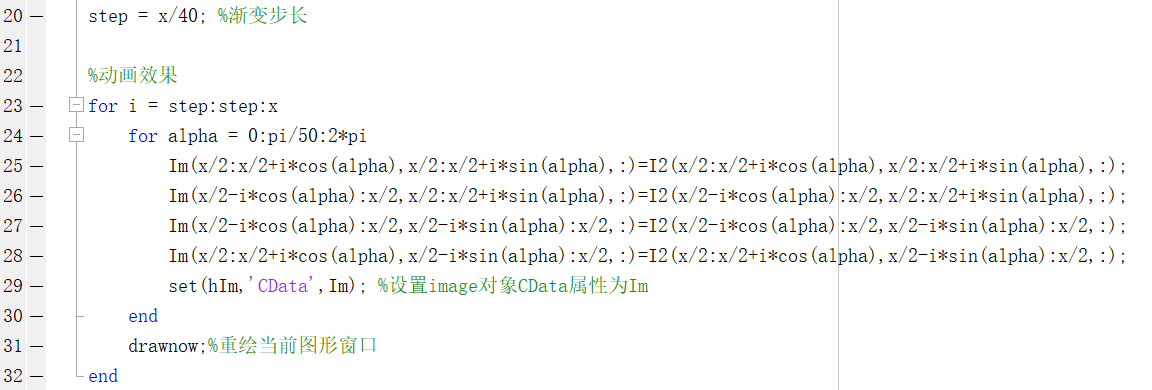


1~9行主要是将两张图片读入并赋予到I1和I2上。

第14行的figure函数用于Figure图形窗口的创建。menubar','none'表示禁用菜单栏；‘NumberTitle’’off’表示图形标题中不显示图形编号。Position设置图形窗口位置和大小，name设置标题。

movegui(hFigure,'center')将图片的位置居中。Axes函数用于坐标轴的创建，参数与Figure类似，就不累述。

Imshow(Im)将诺贝尔.jpg先显示在布局中作为背景。

以上主要是为后续的动画制作提供一个良好的环境，接下来就是实现的部分：  


要实现一个动画效果，需要将图片在不同的时间段的每一帧结合起来，时间间隔越短，那么帧数越大，动画效果就相对平滑。

第20行是设置一个赋值的范围跨度，值越大，动画被切割的份数就越多，那么圆圈扩散的过程就显得更加平滑。

然后就是for循环。alpha是角度，范围是0到2pi。在每一个step中，将诺贝尔,jpg中的一部分转换为lena.jpg。相当于将以诺贝尔.jpg的中心为圆心，以i为半径，将这个圆的部分显现出lena.jpg的一部分，随着每个step的进行，lena的部分逐渐覆盖掉诺贝尔的部分。

Im(x/2:x/2+i\*cos(alpha),x/2:x/2+i\*sin(alpha),:)=I2(x/2:x/2+i\*cos(alpha),x/2:x/2+i\*si(alpha),:);现在来解释下这条语句：

每个逗号将括号内分为三个部分，分别为第一维，二维和三维。第一维和二维分别设置横坐标和纵坐标，第三维可认为是对色彩的设定(“：”表示取索引全部范围，可以认为是保留全部色彩)，第一维的范围从x/2到x/2+i\*cos(alpha)，随着i的增大，越来越多的区域被I2(lena.jpg)中x/2到/2+i\*cos(alpha)范围的I2覆盖。从纵坐标的索引x/2到x/2+i\*sin(alpha)(其实应该是y/2到y/2+i\*sin(alpha)，但是由于图片本身是正方形，所以没有差别)可以看出，这条语句是对右上角的四分之一圆部分实现动画效果。同理，其他的三条语句实现其他位置的动画效果。

第29行将Image对象的CData属性赋给Im，确保显示新的图像。drawnow函数重绘图形窗口，显示新的一帧。

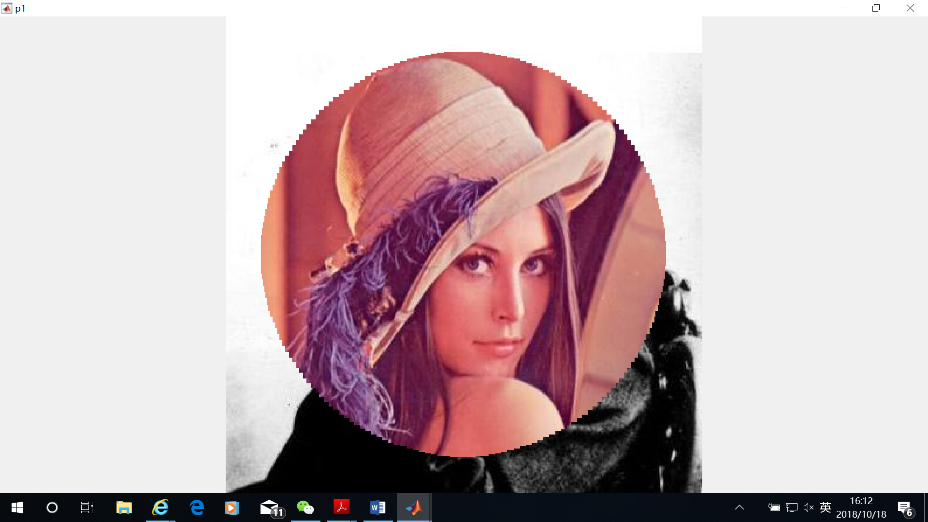
实现的大概效果如图所示（word传的gif不会动，所以只好一帧一帧截图，截了三帧）：





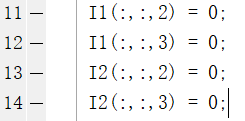


仔细看我们会发现，这个圆形的效果其实并不太好，外边缘还有许多锯齿状。这是因为在alpha循环里切割的份数不够。第24行如果我们将每一次循环的step改成pi/100，得到的效果如下：

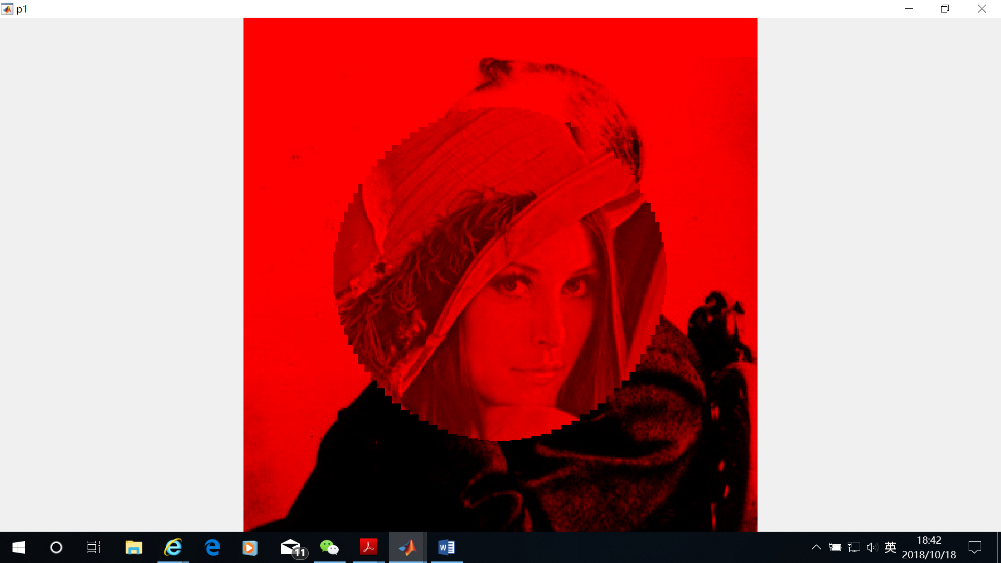
  
可以看出来效果要好不少。同理，如果改成pi/200, pi/400……效果会越来越好。

但是这样改也需要付出相应的代价。随着切割份数的增加，for循环进行的次数会越来越多，这会导致整个动画从一帧到另一帧所需要的时间越来越长，动画就会显得非常不连贯。所以我反复测试了之后调到了一个折中的数值。

获取红色通道更新的话，添加几行代码即可：



得到的效果如下（这个没有保存gif）：



总体实现效果一般，但是还没找到优化的方法，目前就做到这样了。

1. 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.

在进行这次作业之前，我们需要先知道什么是中位切分算法（median-cut algorithm）。

中位切分算法的原理很简单直接，将图像颜色看作是色彩空间中的长方体（VBox），从初始整个图像作为一个长方体开始，将RGB中最长的一边从颜色统计的中位数一切为二，使得到的两个长方体所包含的像素数量相同，重复上述步骤，直到最终切分得到长方体的数量等于主题颜色数量为止。

具体步骤如下：

1) 找出最小的方形区，它包含图像中的所有颜色。

2) 沿方形区的长边排序它所包含的颜色。

3) 在排序链表的中间处把该方区划分成两个区域。

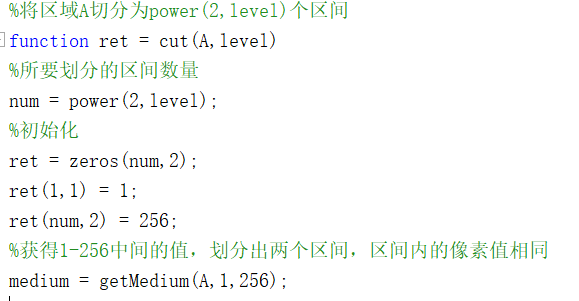
4) 重复上面步骤2、3，直到初始的颜色空间被分割成256个区域为止。

5) 对每个方形区，把该方形区中RGB的平均值作为其代表(中心)颜色。

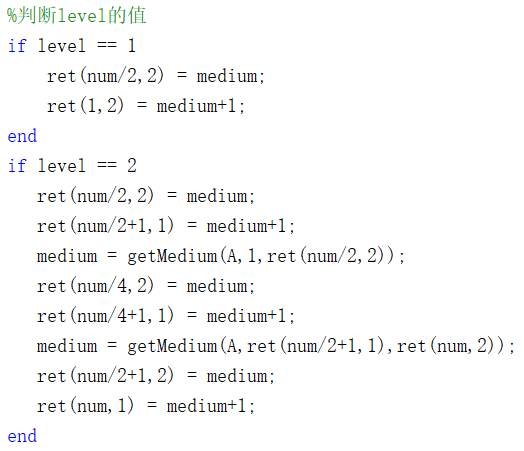
6) 根据一个像素的RGB值与每个方形区的中心值的欧氏距离，给每一个像素分配一个代表颜色。在指向代表颜色的查找表中用编号代替像素。

实现起来的过程挺麻烦的（可能是方式有点问题），就不一一解释代码了，基本的写在注释里。

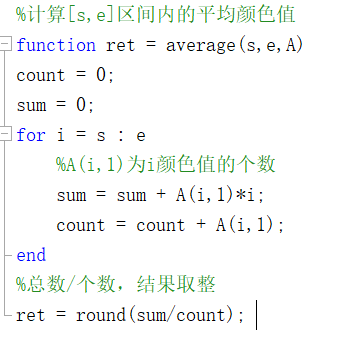
重点在划分区域的实现，首先是准备过程：

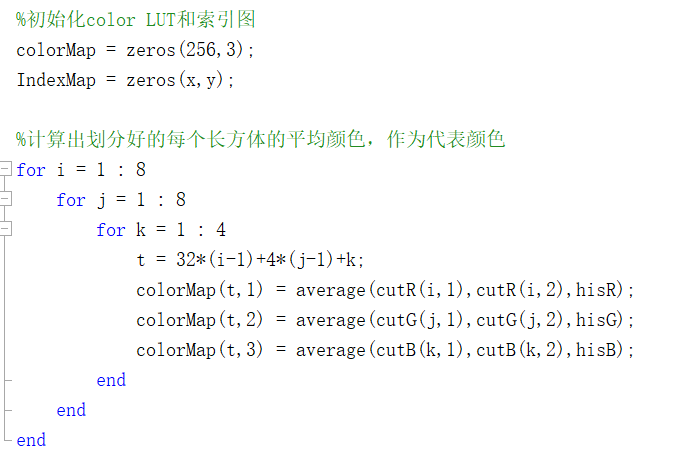


然后根据不同level的值进行不同的划分，这里贴出level为1和2的划分

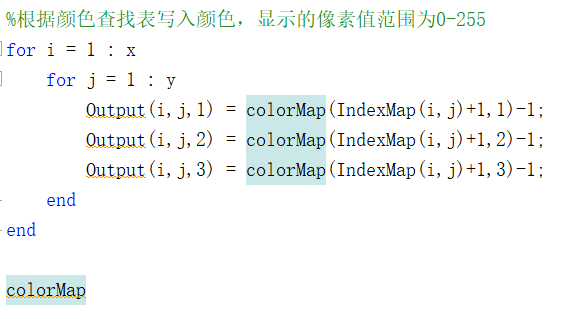


然后创建了一个新的函数来求区间的平均颜色值：

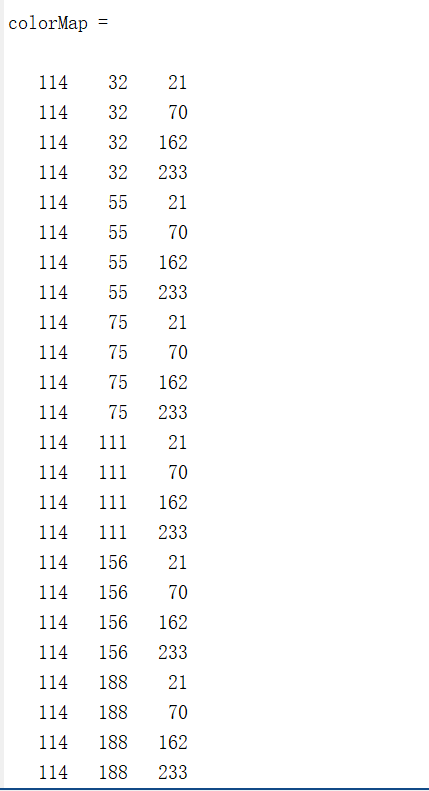




然后根据color LUT写入颜色：

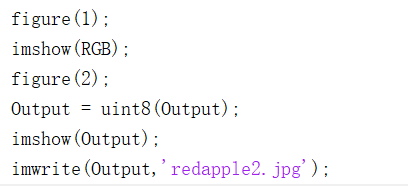


单独的一个colorMap输出颜色查找表（在command window可以看到，图没有完全截完）：



这样颜色查找表就算是构建出来了，问题也算解决了。

至于为什么要把颜色等级放在24位彩色图像的地方，我们可以与8位的图像比较一下：





显而易见，24位真彩色图像相比于低位级图像，有较好的色彩过滤，更加细腻，携带的色彩信息可以更加丰富。24位RGB，一个R通道里有2的24次方个灰度级，而8位RGB只有2的8次方个灰度级。所以24位的RGB会有更多的颜色等级。