实验题目名称：:图片缩放的实验

目录

（一）实验内容

设原始图片的宽度和高度分别为 W 和 H .横向、纵向缩放的比例为 rx ， ry .设缩放后的图片宽度和高度分别为 Ws 和 Hs .则基本有如下结果：

$$W_s = r_x W 或者 W_s \approx r_x W$$
$$H_s = r_v H 或者 H_s \approx r_v H$$

采用下列参数进行实验：

$$r_x = r_v = 2$$
$$r_x = r_v = 0.5$$

实现对于图像的缩放。

（二）实验目的

（1） 熟悉 matlab 软件命令编写调试的基本方法
（2） 利用 matlab 编程，实现对于图像的缩放
（3） 彩色图像和灰度图像分别进行缩放处理

（三）实验过程

1、输入图片利用 imread 命令读取其像素矩阵获取原图的高度和宽度
2、根据函数输入参数获得缩放后新高度和宽度
3、根据原图与处理后图的映射关系实现处理图像的像素填充
4、初始化矩阵，用双线性差值获得对应位置的像素值
5、显示元素和进行缩放处理后的图像
6、根据像素矩阵特征分别进行灰度图像和彩色图像处理

1.1. 问题分析

图像是由一个一个的像素点组成的，在进行图像的缩放时，本质上是在增加或者减少像素点的个数，因此我们只需要为缩放后图像对应位置上填充相应的像素值并进行图像显示后即可以得到缩放后的图像。

通过缩放比例，可以求得新图像 B 在(i,j)处对应原图像的点为(x,y)，但由于缩放比例的不同，导致映射至原图像中的像素点坐标可能为浮点数，而原图像的像素点坐标值均为整数，所以这个点在原图像中对应的可能是“虚”点。

所以，我们需要借助于双插值法通过“虚点”周围的像素点来实现虚点处的像素填充。

1.2. 模型假设

模型的建立在本实验中主要设计两部分：
（1） 图像数学模型建立：
利用 matlab 自带的 imread 命令可以将图像转换为三阶的数学矩阵形式，在本处需要注意，由于本实验中涉及两种图像处理：灰度图像和彩色图像。通过 imread 命令可以将图像转换为三阶矩阵，而三阶矩阵可以视作三个二阶矩阵的叠加，唯一的区别在于彩色图像三个二阶矩互不相同，而灰度图像三个二阶矩阵完全相同。
（2） 双插值法模型建立：
双插值法模型用于求解原图像中“虚点”处的像素值，为解决像素求解问题建立数学模型，利用虚点周围四个整点的像素值来近似计算虚点处的像素值。问题可以简化为三个相似三角形结合求解像素值。

1.3. 变量与符号说明

h：原图高度
w：原图宽度
i:像素矩阵的行
j:像素矩阵的列
x:缩放后图片高度
y:缩放后图片宽度
u:x 的整数部分及行相邻
v:y 的整数部分及列相邻
uint8:像素值范围 0-255，所以为 8 比特

1.4. 模型建立与求解(实验过程)

通过缩放比例,可以求得新图像 B 在(i,j)处对应原图像的点为(x,y)(其中 x= i*h/scale_h, y= j*w/scale_w)。

但是这两个值可能为浮点数，而像素中的位置是整数，所以这个点在原图像中对应的可能是“虚”点。所以，我们需要根据找该“虚”点周围的四个点来进行双线性插值得到新图像的灰度值。

我们应该取的是(x,y)邻近的四个像素：

$(x_0,y_0),(x_1,y_0),(x_0,y_1),(x_1,y_1)$它们对应的灰度值$f(x_0,y_0),f(x_1,y_0),f(x_0,y_1),f(x_1,y_1)$

对其中某一个方向进行插值（以 y 方向为例）：

$$z_1 = \frac{f(x_0,y_1)-f(x_0,y_0)}{y_1-y_0} * v + f(x_0,y_0)$$

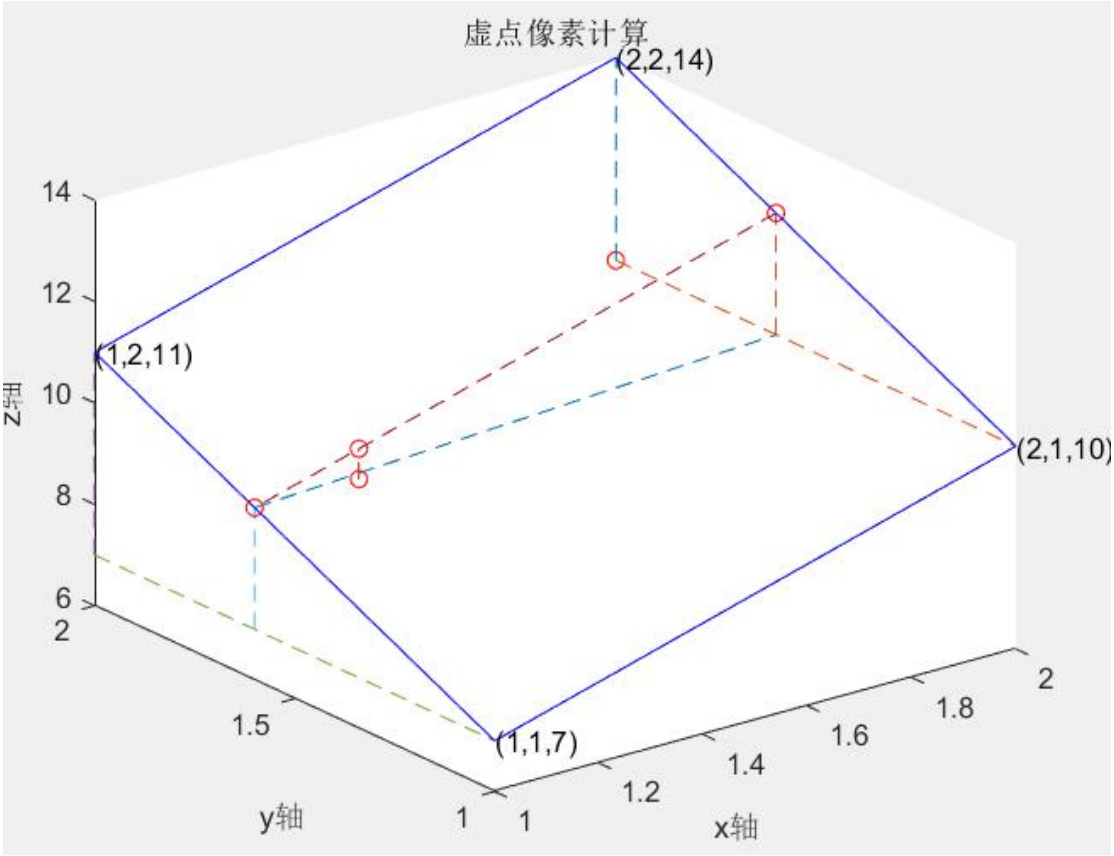
$$z_1 = \frac{f(x_1,y_1)-f(x_1,y_0)}{y_1-y_0} * v + f(x_1,y_0)$$

$$z = \frac{z_2-z_1}{x_1-x_0} * u + z_1$$

最终推导出像素点之间的对应关系：

$$z = v*[u*f(x_1,y_1)+(1-u)*f(x_0,y_1)]+(1-v)*[u*f(x_1,y_0)+(1-u)*f(x_0,y_0)]$$

双线性插值法模型建立：



以特殊点为例讲解算法模型：

在本实验中以（1.2，1.6）点为例讲解利用双向插值法求解“虚点”像素值的原理与方法。

所谓双向插值法，就是在两个方向上进行插值求解虚点处的像素值，以（1.2，1.6）点为例，首先便是在 x = 1 上进行插值，利用相似三角形进行像素值求解：(1.6-1) /1 = 像素值 1 / (11-7) 可以计算出 x = 1 方向上的像素值。另外在 x = 2 方向上进行插值求解虚点处的像素值，利用相似三角形进行像素值求解：(1.6-1) /1 = 像素值 2/ (14-10) 可以计算出 x = 2 方向上的像素值。最终在 x 方向上进行第三个相似三角形求解像素值：(1.2-1) /1 = 像素值 3/(像素值 2 - 像素值 1)
通过上述三个相似三角形求解，进行双向插值法建模即可以计算出虚点处的像素值。拓展至其他点位即可以计算出像素矩阵的像素值，展示缩放后的图像。

1.5. 主要代码解析

以彩色图像缩放为例解析代码：

```
function test
scale('test.jpg',[540,960])

function output_img = scale(input_img, scale_size)

%Input - input_img is a two-dimensional matrices storing image
%      - scale_size is a tuple of [width, height] defining the spatial resolution
of output
%Output - output_img is the same as input_img

img = imread(input_img); %读取输入图片的数据
[h,w] = size(img(:,:,1)) %获取行和列，即原图的高度和宽度

scale_h = scale_size(1); %根据输入获得缩放后的新宽度
scale_w = scale_size(2); %根据输入获得缩放后的新高度
output_img = zeros(scale_h, scale_w,3); %初始化
for m = 1:3
    for i = 1 : scale_h         %缩放后的图像的（i,j）位置对应原图的（x,y）
        for j = 1 : scale_w
            x = i * h / scale_h;
            y = j * w / scale_w;
            u = x - floor(x);
            v = y - floor(y); %取小数部分

            if x < 1           %边界处理
                x = 1;
            end

            if y < 1
                y = 1;
            end
```

```
        %用原图的四个真实像素点来双线性插值获得“虚”像素的像素值
            output_img(i, j,m) = img(floor(x), floor(y),m) * (1-u) * (1-v) + ...
                                 img(floor(x), ceil(y),m) * (1-u) * v + ...
                                 img(ceil(x), floor(y),m) * u * (1-v) + ...
                                 img(ceil(x), ceil(y),m) * u * v;
        end
    end
end

imwrite(uint8(output_img), '../output_img.png'); %保存处理后的图像
imshow(input_img); %显示原图
figure,imshow(uint8(output_img)) %显示处理后的图像
figure,imshow(imresize(img,0.5))
```

Tips:

（1）function output_img = scale(input_img, scale_size)两个输入参数，input_img 为待处理图像，以文件形式呈现，scale_size 为缩放矩阵，其中存放缩放后图片的预定尺寸。

（2）img = imread(input_img);imread 命令对于图像进行建模处理，将待处理图像转换为数学矩阵，注意 imread 命令读取为三阶矩阵，需要将其转换为二阶矩阵。

（3）output_img(i, j,m) = img(floor(x), floor(y),m) * (1-u) * (1-v) + ...
img(floor(x), ceil(y),m) * (1-u) * v + ...
img(ceil(x), floor(y),m) * u * (1-v) + ...
img(ceil(x), ceil(y),m) * u * v;

根据公式进行缩放后的图像像素值填充,注意两个取整命令,floor 朝负无穷方向进行取整,ceil 朝正无穷方向进行取整

（4）figure,imshow(uint8(output_img)) %显示处理后的图像,显示缩放处理后的图像时,uint8 命令表示 8 位无符号整数（取值范围 0-255）保证数值处于灰度图像像素值范围内（0-255）

（5）figure,imshow(imresize(img,0.5))matlab 系统自带的图像缩放函数,满足实验要求,与个人编写函数进行对比。

（四）实验结果分析

0.5 倍灰度图像缩放实验结果展示：

原图像：

像素矩阵：（256*384）

缩放后的图像：

像素矩阵：（128*192）

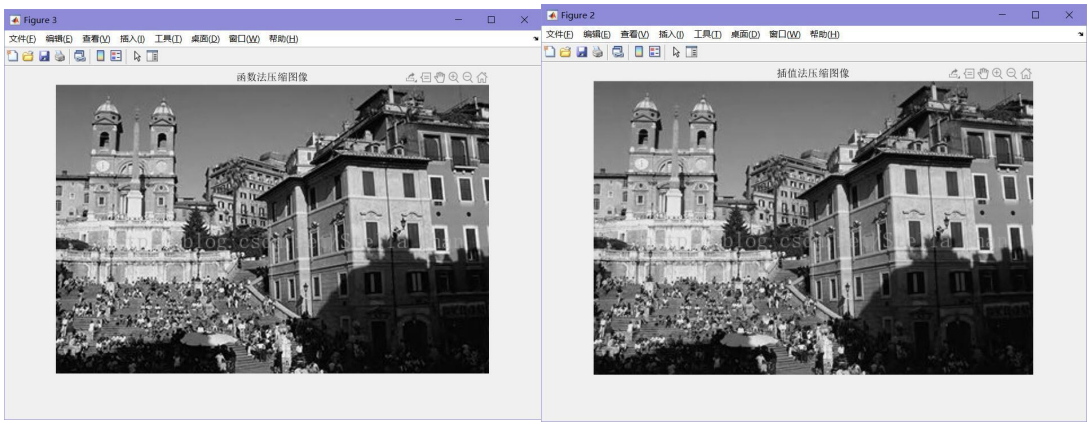
由上述图像对比可以见的：函数法与插值法处理后的图像基本一致

2 倍灰度图像缩放实验展示：

原图像：

像素矩阵：（256*384）

放大后的图像：

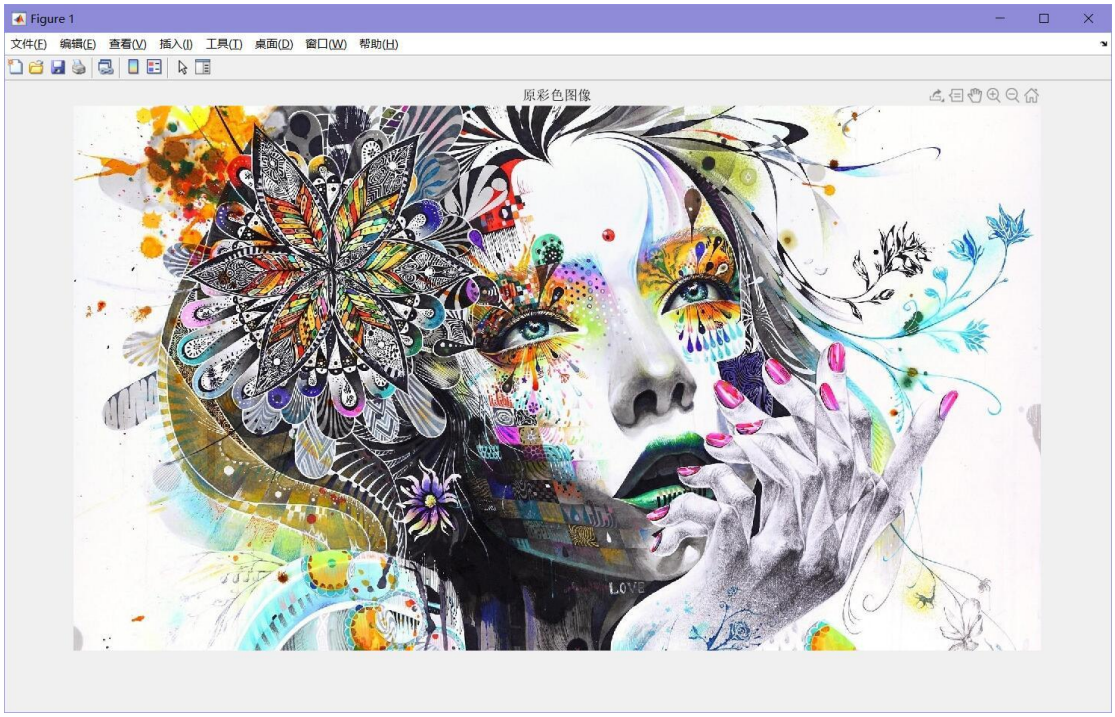


像素矩阵：（512*768）

由上述图像对比可以见的：函数法与插值法处理后的图像基本一致

0.5 倍彩色图像缩放实验展示：

原图像：



像素矩阵：（1080*1920）

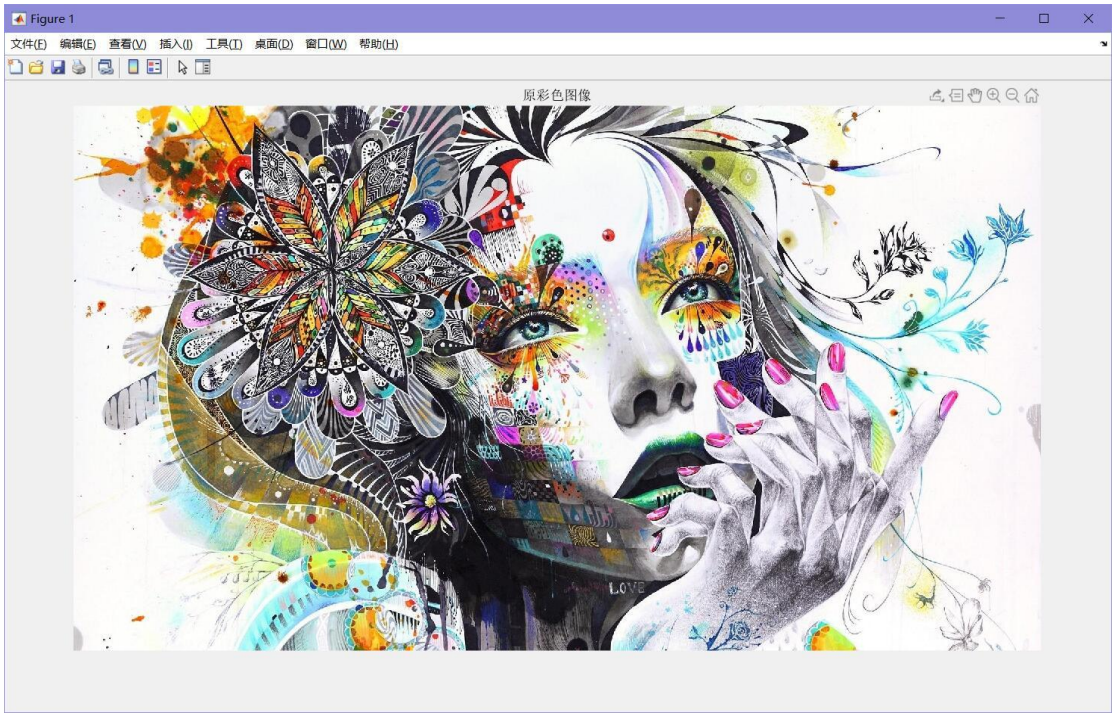
缩放处理后图像：

像素矩阵：（540*960）

由上述图像对比可以见的：函数法与插值法处理后的图像基本一致

2 倍彩色图像缩放实验展示：

原图像：



像素矩阵：（1080*1920）

放大后的图像：

像素矩阵：（2160*3840）

由上述图像对比可以见的：函数法与插值法处理后的图像基本一致

（五）优缺点及改进方向

本实验采用双线性插值法实现图像的缩放处理，程序简单易于理解，同时实验结果接近 matlab 自带 imresize 函数进行图像缩放处理的结果。同时针对于两种不同的图像类型分别进行处理：灰度图像和彩色图像，均得到较为理想的实验结果。同时本实验提供的图像缩放的方法相较于 imresize 而言，可以直接设置缩放后图像的高度和宽度，比较方便。

但同时实验设计存在部分细节待改进，在本此的图像缩放实验中，对于图像边缘化的处理比较粗糙，在映射后坐标值小于 1 时，直接采用重新赋值为 1 的处理，相当于直接采用周边点位的像素值进行像素填充，可以采取更好的方式进行边缘处理。

针对于实验的改进方向，现行存在有诸多双线性插值法的改进模型，三向插值法，基于局部均值的图像缩放方法，在后续的改进中可以增加类似算法实现对于本实验的改进。

（六）心得体会与总结

通过本次实验设计，了解了图像的基本组成原理，同时掌握如何利用双线性插值法借助于matlab软件进行图像的缩放，通过本次实验实训，对于matlab编程语言有进一步的掌握， 收获非常大。

在本次实验过程中，深刻理解数学建模的思想的在问题解决中的应用，在未来的学习与生活中，应该着重培养自己将实际问题转换为数学问题的能力。

（七）附件

（1）彩色图像处理：

```
function test
scale('test.jpg',[540,960])

function output_img = scale(input_img, scale_size)
%Input - input_img is a two-dimensional matrices storing image
%      - scale_size is a tuple of [width, height] defining the spatial resolution
of output
%Output - output_img is the same as input_img

img = imread(input_img); %读取输入图片的数据
[h,w] = size(img(:,:,1)) %获取行和列，即原图的高度和宽度

scale_h = scale_size(1); %根据输入获得缩放后的新宽度
scale_w = scale_size(2); %根据输入获得缩放后的新高度
output_img = zeros(scale_h, scale_w,3); %初始化
for m = 1:3
    for i = 1 : scale_h         %缩放后的图像的（i,j）位置对应原图的（x,y）
        for j = 1 : scale_w
            x = i * h / scale_h;
            y = j * w / scale_w;
            u = x - floor(x);
            v = y - floor(y); %取小数部分

            if x < 1           %边界处理
                x = 1;
            end

            if y < 1
                y = 1;
            end
```

```
        %用原图的四个真实像素点来双线性插值获得“虚”像素的像素值
            output_img(i, j,m) = img(floor(x), floor(y),m) * (1-u) * (1-v) + ...
                                 img(floor(x), ceil(y),m) * (1-u) * v + ...
                                 img(ceil(x), floor(y),m) * u * (1-v) + ...
                                 img(ceil(x), ceil(y),m) * u * v;
        end
    end
end

imwrite(uint8(output_img), '../output_img.png'); %保存处理后的图像
imshow(input_img); %显示原图
figure,imshow(uint8(output_img)) %显示处理后的图像
figure,imshow(imresize(img,0.5))
```

（2）灰度图像处理：

```
function test
scale('web.png',[128,192])

function output_img = scale(input_img,scale_size)
A = imread(input_img);
img = A(:,:,3);
[h,w] = size(img);
% h = 256 w = 384
scale_h = scale_size(1);
scale_w = scale_size(2);
output_img = zeros(scale_h,scale_w);

for i = 1:scale_h
    for j = 1:scale_w
        x = i.*h / scale_h; %x:高度
        y = j.*w / scale_w; %y:宽度
        u = x - floor(x);
        v = y - floor(y);

        if x < 1
            x = 1;
        end

        if y < 1
            y = 1;
        end
             output_img(i, j,m) = img(floor(x), floor(y),m) * (1-u) * (1-v) + ...
```

```
                                    img(floor(x), ceil(y),m) * (1-u) * v + ...
                                    img(ceil(x), floor(y),m) * u * (1-v) + ...
                                    img(ceil(x), ceil(y),m) * u * v;
    end
end

% imwrite(uint8(output_img),'../output_img.png');
imshow(input_img)
figure,imshow(uint8(output_img))
figure,imshow(imresize(img,0.5))
```