遥感数字图像处理实验课

常用图像的几何变换

李荣昊

2021年11月

实验内容

- 将给定的图像进行平移、缩放、旋转以及镜像的变换
- 了解常见的插值方法，并通过实验体会不同插值方法的结果

几何变换

◆几何变换内容

- 空间位置变换

即用来描述每个像素空间位置的变换，建立原图与校正图像之间像素坐标的映射关系，根据映射关系对图像各个像素坐标进行变换求解。

- 像元灰度插值

确定变换图像各像素的灰度值，变换前后图像的灰度是不发生变化的。但其像素分布可能是不规则的，会出现挤压疏密不均的情况，需要通过灰度插值生成规则的栅格图像。

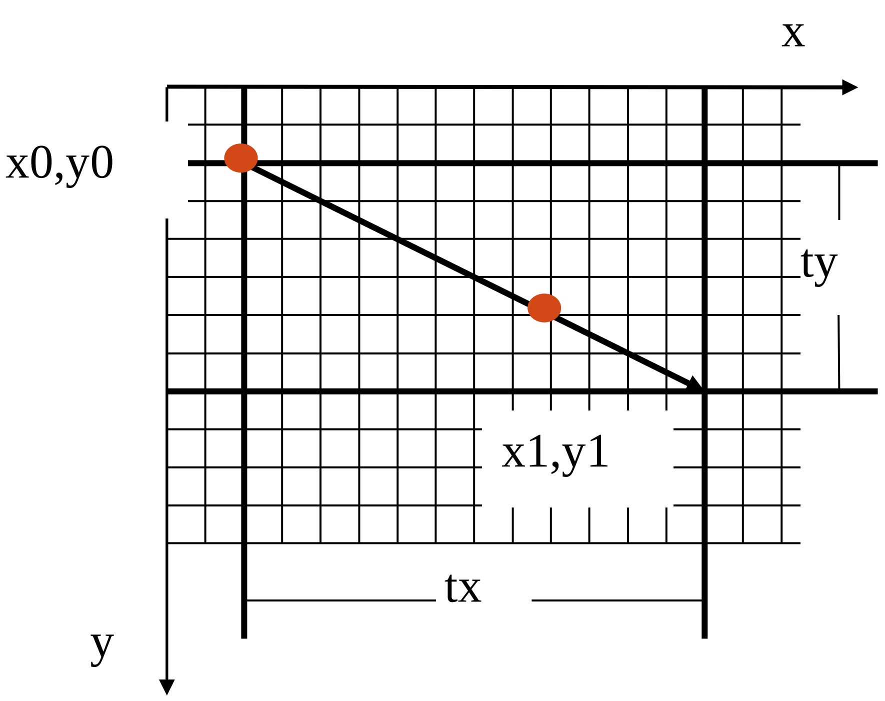
几何变换

◆ 数字图像常用的几何变换

- ★ 平移
- ★ 缩放
- ★ 旋转
- ★ 镜像
- ★ 复合

几何变换——平移

◆原理



现设点P0(x0, y0) 进行平移后，移到P(x1, y1) .
其中x方向的平移量为tx, y方向的平移量为ty。
那么，点P(x1, y1) 的坐标为

$$\begin{cases} x1 = x0 + tx \\ y1 = y0 + ty \end{cases}$$

几何变换——平移

◆齐次坐标

$$\begin{cases} x0 = x1 - tx \\ y0 = y1 - ty \end{cases} \qquad \begin{cases} x1 = x0 + tx \\ y1 = y0 + ty \end{cases}$$

- 上述过程可以用矩阵表述为：

$$\begin{bmatrix} x1 \\ y1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x0 \\ y0 \end{bmatrix} + \begin{bmatrix} tx \\ ty \end{bmatrix}$$

几何变换——平移

◆齐次坐标

而平面上点的变换矩阵$T = \begin{bmatrix} \mathrm{a} & b \\ c & d \end{bmatrix}$中没有引入平移常量,无论abcd取什么值,都不能实现上述的平移变换。

因此，需要使用2 ×3 阶变换矩阵，取其形式为

$$T = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & \mathrm{t}y \end{bmatrix}$$

第三列为平移向量。

但是，图像的点集矩阵是2 ∗ n阶的，但是构建的T是2*3阶的，无法直接相乘。

几何变换——平移

◆齐次坐标

所以，对于点集矩阵[x y]^{T}扩展为[x y 1]^T，变换结果可以写作：

$$P1 = T \cdot P0 = \begin{bmatrix} 1 & 0 & tx \\ 0 & 1 & ty \end{bmatrix} \begin{bmatrix} x0 \\ y0 \\ 1 \end{bmatrix} = \begin{bmatrix} x0 + tx \\ y0 + ty \end{bmatrix} = \begin{bmatrix} x1 \\ y1 \end{bmatrix}$$

T 则为平移变换矩阵。

通常将2*3的句子扩充为3*3，以拓宽功能：

$$T = \begin{bmatrix} 1 & 0 & \mathrm{tx} \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix}$$

几何变换——平移

◆齐次坐标

表达式变为：

$$P1 = T \cdot P0 = \begin{bmatrix} 1 & 0 & \mathrm{tx} \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x0 \\ y0 \\ 1 \end{bmatrix} = \begin{bmatrix} x0 + tx \\ y0 + ty \\ 1 \end{bmatrix} = \begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix}$$

T 为平移变换矩阵。

更一般的，点(x, y)的齐次坐标可以表示为(Hx, Hy, H)，其中H为非零实数，H=1时成为规范齐次坐标。

几何变换——平移

◆齐次坐标

齐次坐标在2D 图像几何变换中的另一个应用是：如某点 S(60 000，40000) 在字长为16位的计算机上表示则大于32767的最大坐标值，需要进行复杂的操作。但如果把S 的坐标形式变成(Hx,Hy,H) 形式的齐次坐标，在齐次坐标系中 ，设H = 1/2， 则 （ 60 000，40000) 的齐次坐标为(1/2x，1/2y，1/2) ， 那么所要表示的点变为(30000，20000, 1/2)，此点显然在 16 位计算机上二进制数所能表示的范围之内。

几何变换——平移

◆原理

- 平移后的每个点与原图一一对应，无需插值
- 不在原图中的点

  有损：统一赋值0或255

  无损：新图像的宽度扩大|tx|，高度扩大|ty|

几何变换——平移

◆具体算法实现

```
[36]: def geo_translation(image,trans_mat,interpolation_method:"插值方法"="b"):
          h,w,c = image.shape
          img_trans = np.zeros(image.shape,dtype = "uint8")
          for i in range(h):
              for j in range(w):
                  pos = np.array([i,j,1],dtype = np.float)
                  pos_ori = np.dot(trans_mat,pos)
                  if interpolation_method == 'b':
                      img_trans[i,j] = bilinear_interpolation(image,pos_ori[:2])
                  elif interpolation_method == "n":
                      img_trans[i,j] = nearest_interpolation(image,pos_ori[:2])
          return img_trans
```

平移

```
[62]: trans_mat = np.array([[1,0,100],
                            [0,1,100],
                            [0,0,1]])
      trans_mat_inv = np.linalg.inv(trans_mat)
```

```
[65]: img2 = geo_translation(img_np,trans_mat_inv,"b")
```

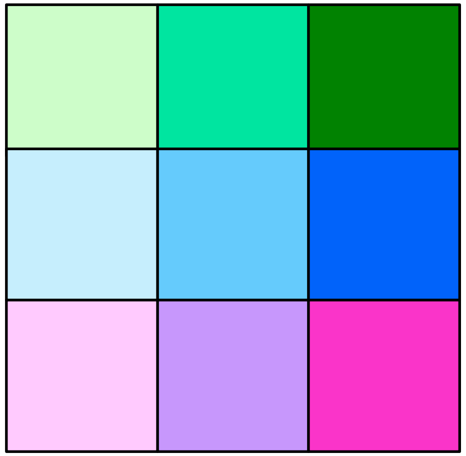
几何变换——镜像

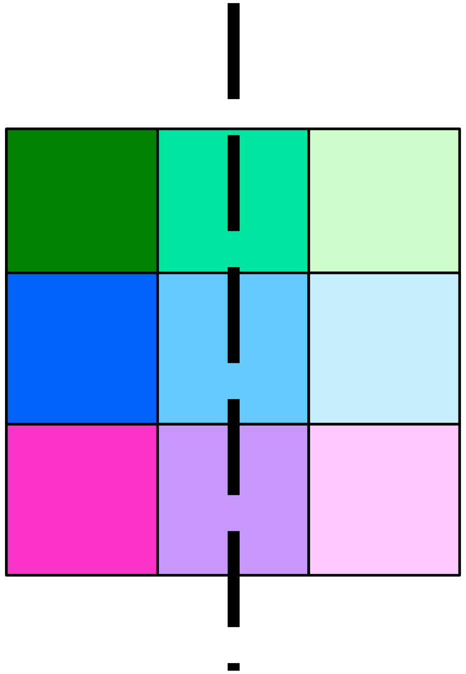
◆原理

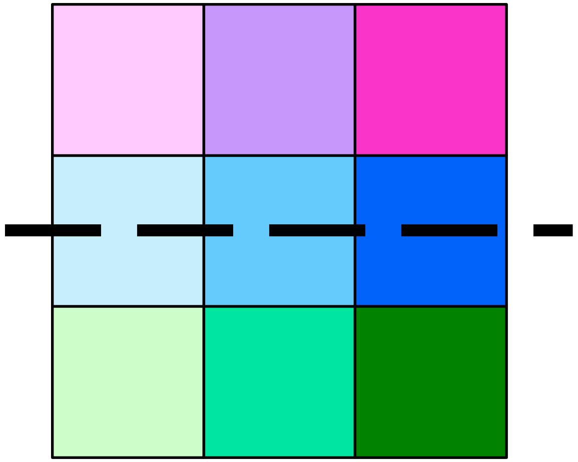
图像的镜像不改变图像形状。主要分为两种：

1. 水平镜像，将图像的左半部分和右半部分以图像垂直中轴线为中心进行镜像对换。

2. 垂直镜像，将图像上半部分和下半部分以图像水平中轴线为中心进行镜像对换。







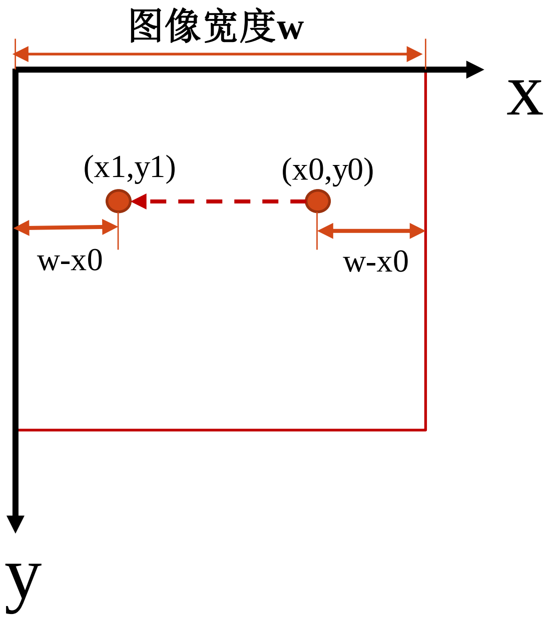
几何变换——镜像

◆原理

图像的镜像变换分为水平镜像和垂直镜像，无论是水平镜像还是垂直镜像，镜像后高度和宽度都不变。

- 水平镜像

以原图像的垂直中轴线为中心，将图像分为左右两部分进行对称变换。



$$\begin{cases} x1 = -x0 + \mathrm{w} \\ y1 = y0 \end{cases}$$

$$\Downarrow$$

$$\begin{cases} x0 = -x1 + \mathrm{w} \\ y0 = y1 \end{cases}$$

几何变换——镜像

◆原理

- 水平镜像

矩阵形式：

$$\begin{cases} x1 = -x0 + \mathrm{w} \\ y1 = y0 \end{cases} \qquad \begin{cases} x0 = -x1 + \mathrm{w} \\ y0 = y1 \end{cases}$$

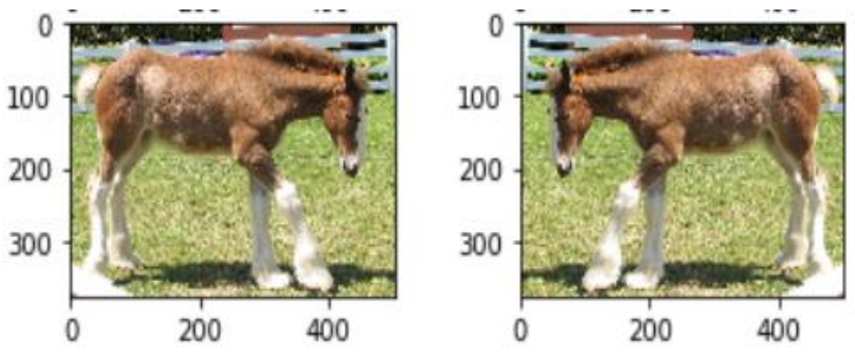
$$\begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & \mathrm{w} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x0 \\ y0 \\ 1 \end{bmatrix} \xrightarrow{\text{逆变换}} \begin{bmatrix} x0 \\ y0 \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & \mathrm{w} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix}$$

几何变换——镜像

◆具体算法实现

- 水平镜像

```
def mirror_translation(image, trans_mat):
    img_ori = np.array(image)
    trans_mat = np.array(trans_mat, dtype=float)#将旋转矩阵（list）转换为数组，浏
    height, width, bands =img_ori.shape
    img_trans = np.zeros(shape = img_ori.shape, dtype=np.uint8)#构建零数组，存放
    for i in range(height):
        for j in range(width):
            trans_pos_np = np.array([j,i,1])#逐像元进行转换前坐标反算
            ori_pos = np.dot(trans_mat,trans_pos_np)#求解变换后坐标对应的原图坐标
            i_ori=int(ori_pos[1])
            j_ori=int(ori_pos[0])
            img_trans[i][j][:] = img_ori[i_ori][j_ori]
    return img_trans
```

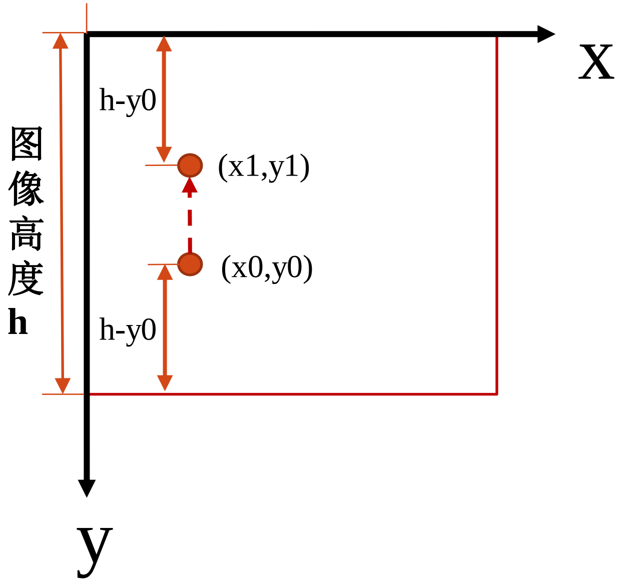


几何变换——镜像

◆原理

- 垂直镜像

以原图像的水平中轴线为中心，将图像分为上下两部分进行对称变换。



$$\begin{cases} x1 = x0 \\ y1 = -y0 + \mathrm{h} \end{cases}$$

$$\begin{cases} x0 = x1 \\ y0 = -y1 + \mathrm{h} \end{cases}$$

几何变换——镜像

◆原理

- 垂直镜像

矩阵形式：

$$\begin{cases} x1 = x0 \\ y1 = -y0 + \mathrm{h} \end{cases}$$

$$\begin{cases} x0 = x1 \\ y0 = -y1 + \mathrm{h} \end{cases}$$

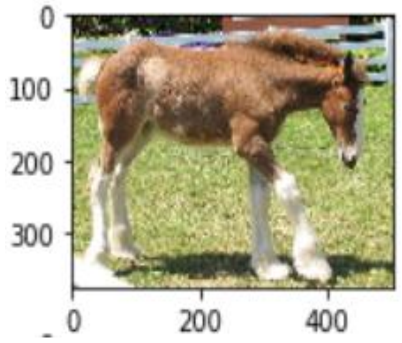
$$\begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & \mathrm{h} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x0 \\ y0 \\ 1 \end{bmatrix} \xrightarrow{\text{逆变换}} \begin{bmatrix} x0 \\ y0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & \mathrm{h} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix}$$

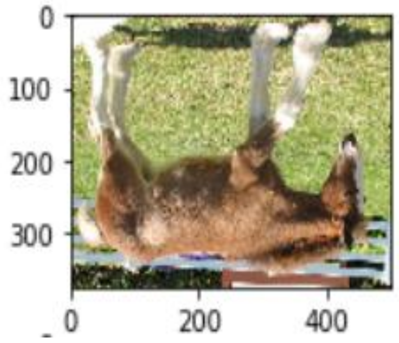
几何变换——镜像

◆具体算法实现

- 垂直镜像

```
def mirror_translation(image,trans_mat):
    img_ori = np.array(image)
    trans_mat = np.array(trans_mat, dtype=float)#将旋转矩阵（list）转换为数组，
    height,width,bands =img_ori.shape
    img_trans = np.zeros(shape = img_ori.shape, dtype=np.uint8)#构建零数组，存放
    for i in range(height):
        for j in range(width):
            trans_pos_np = np.array([j,i,1])#逐像元进行转换前坐标反算
            ori_pos = np.dot(trans_mat,trans_pos_np)#求解变换后坐标对应的原图坐
            i_ori=int(ori_pos[1])
            j_ori=int(ori_pos[0])
            img_trans[i][j][:] = img_ori[i_ori][j_ori]
    return img_trans
```





几何变换——缩放

◆原理

- 不再是1:1的变换，新图像中的像素在原图中可能找不到相应像素点
- 近似处理方法

  插值算法——效果好，运算量大

  直接赋予最接近的像素的值（插值特例：最邻近插值法）

几何变换——缩放

◆原理

- 不再是1:1的变换，新图像中的像素在原图中可能找不到相应像素点
- 近似处理方法

  插值算法——效果好，运算量大

  直接赋予最接近的像素的值（插值特例：最邻近插值法）

几何变换——缩放

◆原理

- x轴方向缩放比率是fx，y轴方向缩放比率是fy，则：

$$\begin{cases} x1 = x0 * fx \\ y1 = y0 * fy \end{cases} \Rightarrow \begin{cases} x0 = x1 / fx \\ y0 = y1 / fy \end{cases}$$

- 例如，当fx=fy=0.5时，图像被缩到一半大小
- 这里默认缩放原点为（0，0）

几何变换——缩放

◆原理

- 矩阵形式

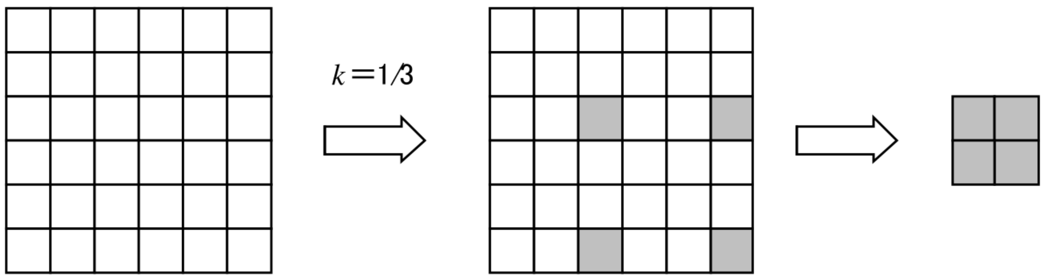
$$\begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & 0 \\ 0 & fy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x0 \\ y0 \\ 1 \end{bmatrix} \xrightarrow{\text{逆变换}} \begin{bmatrix} x0 \\ y0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1/fx & 0 & 0 \\ 0 & 1/fy & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix}$$

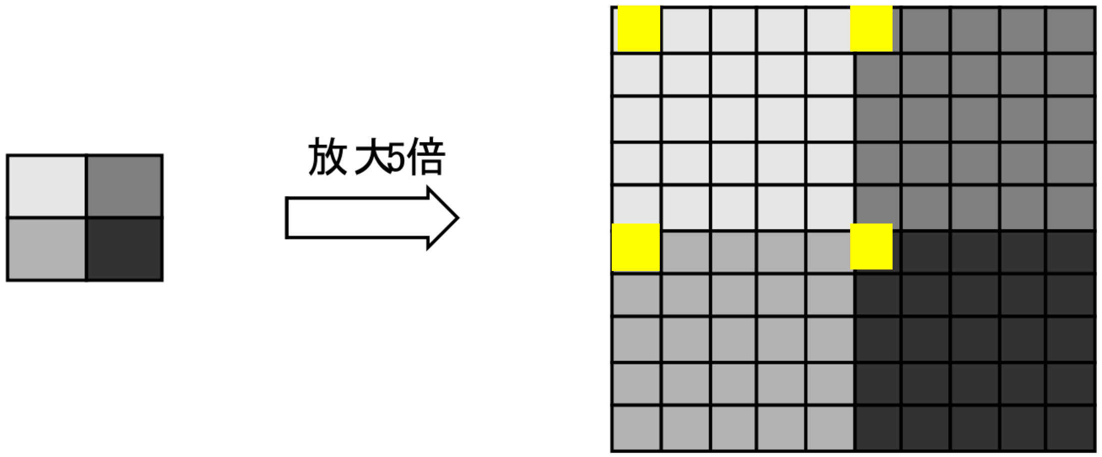
- 像素值填充

  —最邻近插值法
  —双线性内插法

几何变换——缩放

◆原理





图像在缩小操作中，是在现有的信息里如何挑选所需要的有用信息。

而在图像的放大操作中，则需要对尺寸放大后所多出来的空格填入适当的 像素值，这是信息的估计问题，所以较图像的缩小要难一些。

几何变换——缩放

◆原理

- 比例缩放产生的图像像素可能在源图像中找不到对应的像素点，这样就必须进行插值处理。
- 插值算法分两种:1. 直接赋值为最近像素；2. 通过一些插值算法计算相对应的像素值。
- 前者：计算简单，但是有马赛克
- 后者：计算复杂，但是效果好

- 像素值填充

  —最邻近插值法
  —双线性内插法

几何变换——缩放

◆具体算法实现

- 最邻近插值法

```
[34]: def geo_translation(image,trans_mat,interpolation_method="b"):
          '''
          该方法为反算方法，即从转换后图像反算转换前图像的坐标并插值
          image:输入图像，格式为numpy数组
          trans_mat: 图像变换矩阵，是正变换的逆矩阵
          interpolation_method:"插值方法"
          '''
          h,w,c = image.shape
          img_trans = np.zeros(image.shape,dtype = "uint8")
          for i in range(h):
              for j in range(w):
                  pos = np.array([i,j,1],dtype = np.float) #逐像素反算
                  pos_ori = np.dot(trans_mat,pos)#求解原图坐标
                  if interpolation_method == 'b':
                      img_trans[i,j] = bilinear_interpolation(image,pos_ori[:2])
                  elif interpolation_method == "n":
                      img_trans[i,j] = nearest_interpolation(image,pos_ori[:2])
          return img_trans
```

```
[10]: trans_mat = np.array([[0.5,0,0],
                            [0,0.5,0],
                            [0,0,1]])
      trans_mat_inv = np.linalg.inv(trans_mat)
```

```
[11]: img3 = geo_translation(img_np,trans_mat_inv,"n")
```

```
[12]: Image.fromarray(img3)
```

几何变换——缩放

◆具体算法实现

- 双线性内插法

```
[34]: def geo_translation(image,trans_mat,interpolation_method="b"):
        '''
        该方法为反算方法，即从转换后图像反算转换前图像的坐标并插值
        image:输入图像，格式为numpy数组
        trans_mat: 图像变换矩阵，是正变换的逆矩阵
        interpolation_method:"插值方法"
        '''
        h,w,c = image.shape
        img_trans = np.zeros(image.shape,dtype = "uint8")
        for i in range(h):
            for j in range(w):
                pos = np.array([i,j,1],dtype = np.float) #逐像素反算
                pos_ori = np.dot(trans_mat,pos)#求解原图坐标
                if interpolation_method == 'b':
                    img_trans[i,j] = bilinear_interpolation(image,pos_ori[:2])
                elif interpolation_method == "n":
                    img_trans[i,j] = nearest_interpolation(image,pos_ori[:2])
        return img_trans
```

```
[35]: trans_mat = np.array([[0.5,0,0],
                            [0,0.5,0],
                            [0,0,1]])
      trans_mat_inv = np.linalg.inv(trans_mat)
```

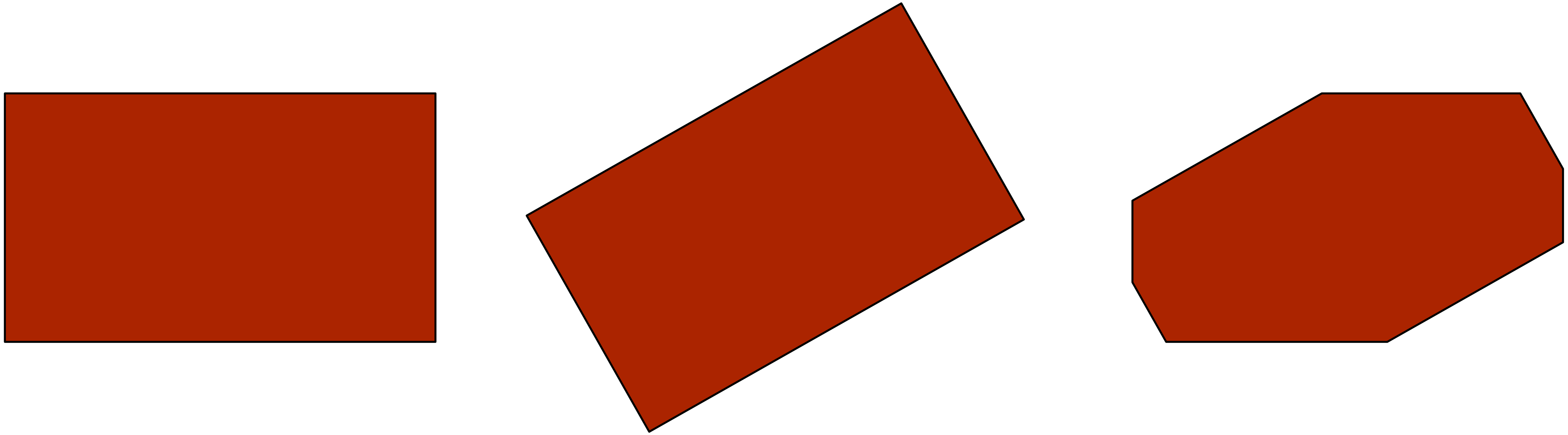
```
[*]: img3 = geo_translation(img_np,trans_mat_inv,"b")
```

```
[*]: Image.fromarray(img3)
```

几何变换——旋转

◆原理

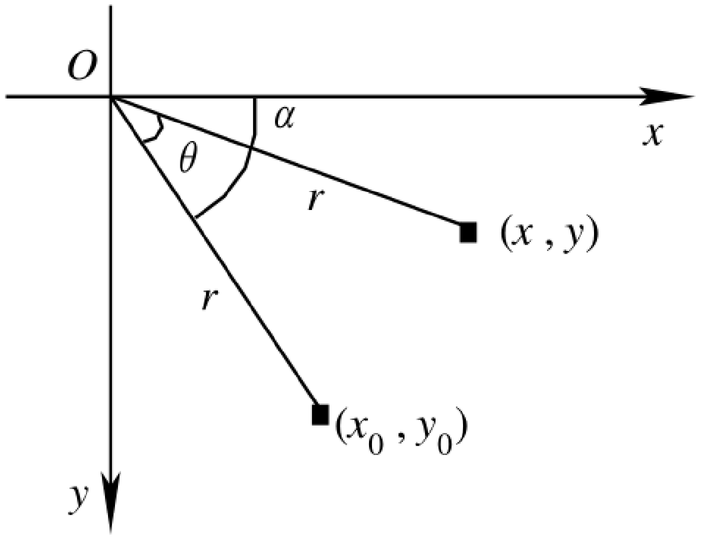
- 一般图像的旋转是以图像的中心为原点 ，将图像上的所有像素都旋转一个相同的角度，图像的旋转变换是图像的位置变换，但旋转后，图像的大小一般会改变。
- 和图像平移一样，在图像旋转变换中既可以把转出显示区域的图像截去，也可以扩大图像范围以显示所有的图像。



几何变换——旋转

◆原理

- 以原点为旋转中心
- 以图像远点(0,0)为圆心旋转，设P0(x0,y0)旋转θ后对应点为P(x,y)。那么，有如下关系：



$$\begin{cases} x_0 = r\cos\alpha \\ y_0 = r\sin\alpha \end{cases}$$

$$\begin{cases} x = r\cos(\alpha-\theta) = r\cos\alpha\ \cos\theta + r\sin\alpha\sin\theta = x_0\cos\theta + y_0\sin\theta \\ y = r\sin(\alpha-\theta) = r\sin\alpha\ \cos\theta - r\cos\alpha\sin\theta = -x_0\sin\theta + y_0\cos\theta \end{cases}$$

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

几何变换——旋转

◆原理

- 以原点为旋转中心

  矩阵形式：

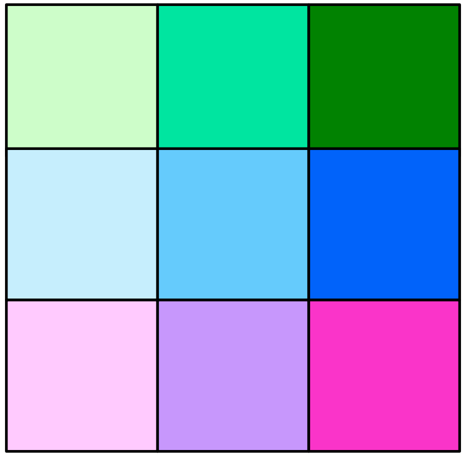
$$\begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos a & \sin a & 0 \\ -\sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x0 \\ y0 \\ 1 \end{bmatrix} \xrightarrow{\text{逆变换}} \begin{bmatrix} x0 \\ y0 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos a & -\sin a & 0 \\ \sin a & \cos a & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix}$$

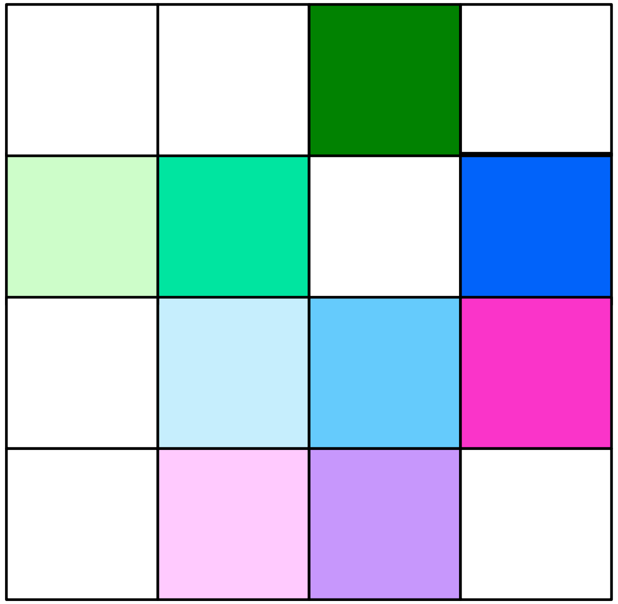
- 像素值填充

  —最邻近插值法
  —双线性内插法

几何变换——旋转

◆原理





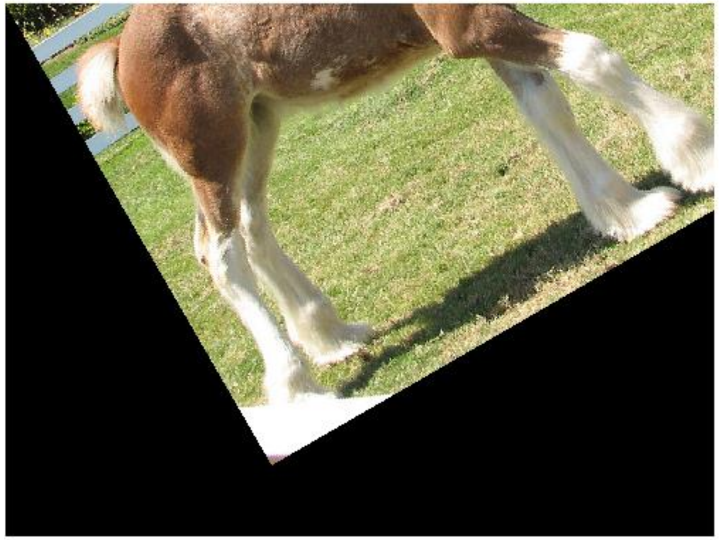
几何变换——旋转

◆具体算法实现

- 最邻近插值法

```
[34]: def geo_translation(image,trans_mat,interpolation_method="b"):
    '''
    该方法为反算方法，即从转换后图像反算转换前图像的坐标并插值
    image:输入图像，格式为numpy数组
    trans_mat: 图像变换矩阵，是正变换的逆矩阵
    interpolation_method:"插值方法"
    '''
    h,w,c = image.shape
    img_trans = np.zeros(image.shape,dtype = "uint8")
    for i in range(h):
        for j in range(w):
            pos = np.array([i,j,1],dtype = np.float) #逐像素反算
            pos_ori = np.dot(trans_mat,pos)#求解原图坐标
            if interpolation_method == 'b':
                img_trans[i,j] = bilinear_interpolation(image,pos_ori[:2])
            elif interpolation_method == "n":
                img_trans[i,j] = nearest_interpolation(image,pos_ori[:2])
    return img_trans
```

```
theta = math.pi*-30/180
trans_mat = np.array([[math.cos(theta),math.sin(theta),0],
                      [-math.sin(theta),math.cos(theta),0],
                      [0,0,1]])
trans_mat_inv = np.linalg.inv(trans_mat)
```



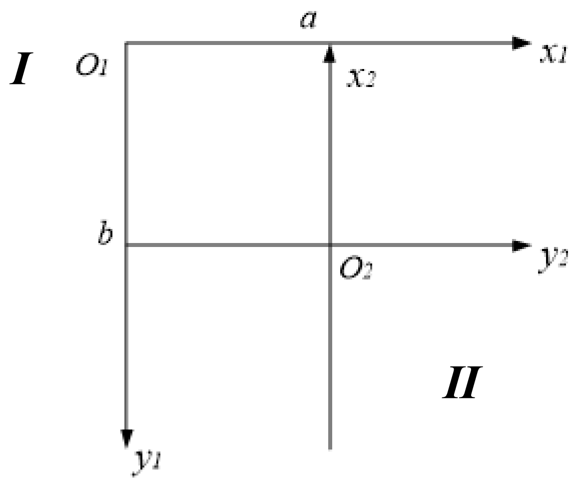
几何变换——旋转

◆原理

- 以任意点为旋转中心

绕一个指定点(a,b)旋转，则先要将坐标系平移到该点，再进行旋转，然后平移回新的坐标原点

假设绕中心点(a,b)旋转，坐标系*I*是图像的坐标系，坐标系*II*是旋转坐标系，坐标系*II*的原点在坐标系中为（*a*，*b*），如下图所示。



两种坐标系之间的转换为：

$$\begin{bmatrix} x_{\mathrm{II}} \\ y_{\mathrm{II}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -a \\ 0 & -1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{I}} \\ y_{\mathrm{I}} \\ 1 \end{bmatrix}$$

几何变换——旋转

◆原理

- 以中心点为旋转中心

设原图像某像元点的坐标为(x_0，y_0)，图像绕中心点(a,b)旋转，旋转后在目标图像的坐标为(x_1，y_1)，则变换分为三步：

S1:将坐标系I变成II，假设图像的宽度为w，高度为 h,则两个坐标的关系为：

$$x_1 = x_0 - w/2;\quad y_1 = -y_0 + h/2;$$

矩阵形式：

$$\begin{bmatrix} x_{\mathrm{II}} \\ y_{\mathrm{II}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -a \\ 0 & -1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{I}} \\ y_{\mathrm{I}} \\ 1 \end{bmatrix}$$

S2:旋转θ（逆时针为正，顺时针为负），变换矩阵为：

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

几何变换——旋转

◆原理

- 以中心点为旋转中心

设原图像某像元点的坐标为(x_0，y_0)，图像绕中心点(a,b)旋转，旋转后在目标图像的坐标为(x_1，y_1)，则变换分为三步：

S3:将坐标系II变换回I，这样就得到了总的变换矩阵，变换矩阵为：

$$\begin{bmatrix} x_{\mathrm{I}} \\ y_{\mathrm{I}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & -1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{\mathrm{II}} \\ y_{\mathrm{II}} \\ 1 \end{bmatrix}$$

几何变换——旋转

◆原理

- 以中心点为旋转中心

设原图像某像元点的坐标为(x_0, y_0)，旋转后在目标图像的坐标为(x_1, y_1)，则旋转变换的矩阵表达式为：

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \boxed{\begin{bmatrix} 1 & 0 & c \\ 0 & -1 & d \\ 0 & 0 & 1 \end{bmatrix}} \boxed{\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}} \boxed{\begin{bmatrix} 1 & 0 & -a \\ 0 & -1 & b \\ 0 & 0 & 1 \end{bmatrix}} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & \sin\theta & -a\cos\theta - b\sin\theta + c \\ -\sin\theta & \cos\theta & a\sin\theta - b\cos\theta + d \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix}$$

逆变换

$$\begin{bmatrix} x_0 \\ y_0 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & a \\ 0 & -1 & b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -c \\ 0 & -1 & d \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\theta & -\sin\theta & -c\cos\theta + d\sin\theta + a \\ \sin\theta & \cos\theta & -c\sin\theta - d\cos\theta + b \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix}$$

几何变换——旋转

◆原理

- 以中心点为旋转中心

最近邻插值

双线性内插

几何变换——旋转

◆原理

- 图像的复合变换是指对给定的图像连续施行若干次如前所述的平移、镜像、比例、旋转等基本变换后所完成的变换，图像的复合变换又叫级联变换。
- 利用齐次坐标对给定的图像依次按一定顺序连续施行若干次基本变换，其变换的矩阵仍然可以用3×3阶的矩阵表示，而且从数学上可以证明，复合变换的矩阵等于基本变换的矩阵按顺序依次相乘得到的组合矩阵。
- 设对给定的图像依次进行了基本变换F1，F2，…，FN，它们的变换矩阵分别为T1，T2，…，TN，图像复合变换的矩阵T可以表示为：T=TN*TN-1*…T1

几何变换——插值

前述讨论中可以看出，在进行图像的比例缩放、 图像的旋转变换时，整个变换过程由两部分组成，即需要两个独立的算法。

1. 需要一个算法来完成几何变换本身，用它描述每个像素如何从其初始位置移动到终止位置；

2. 还需要一个用于灰度级插值的算法。这是因为，在一般情况下，原始（输入）图像的位置坐标(x, y)为整数，而变换后（输出）图像的位置坐标为非整数，即产生“空穴”， 反过来也是如此。

◆ 数字图像常用的插值方法

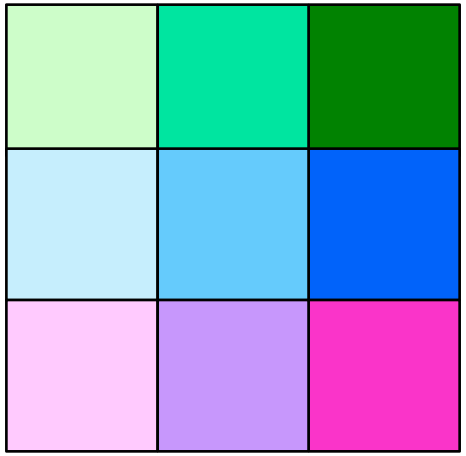
★ 最近邻插值

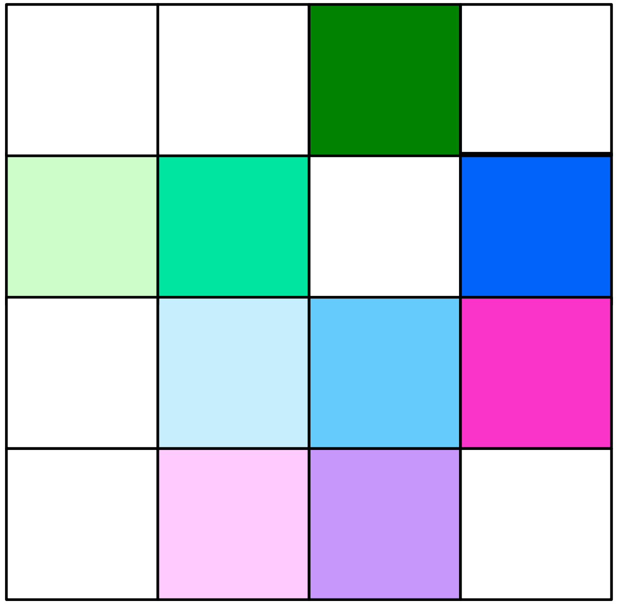
★ 双线性内插

★ 其他

几何变换——旋转

◆原理



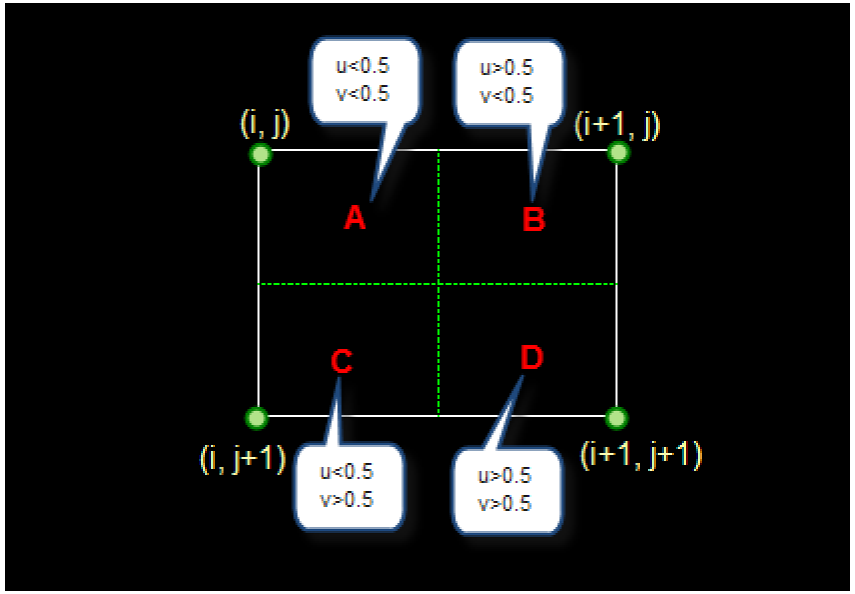


几何变换——最近邻插值

◆原理

在待求像元的四邻像元中，将距离待求像元最近的邻像元灰度赋给待求像元。

设（i+u,j+v）(i, j为正整数， u, v为大于零小于1的小数，下同)为待求像元坐标，则待求像元灰度的值 f(i+u,j+v)如下图所示：



几何变换——最近邻插值

◆算法实现

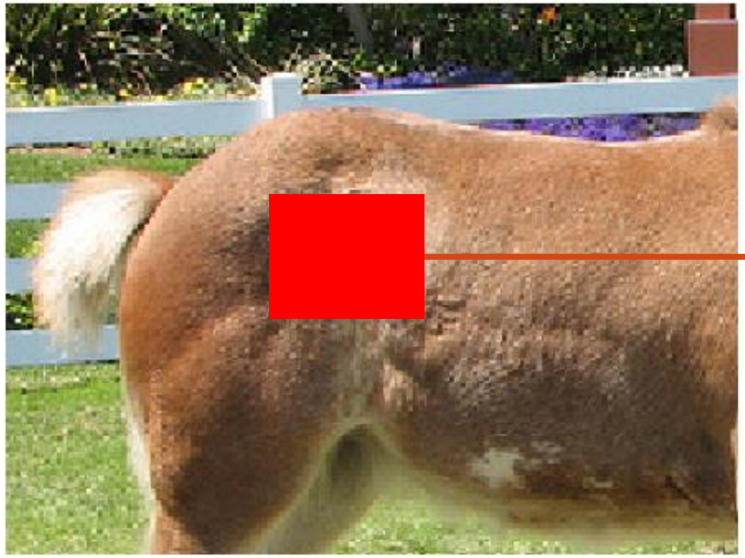
[42]:
```
def nearest_interpolation(img_ori:'原始图像',pos_ori:'坐标'):
    h,w,c = img_ori.shape
    center_x,center_y = pos_ori[0],pos_ori[1]
    x,y = round(center_x),round(center_y)
    if x in range(h) and y in range(w):
        return img_ori[x,y]
    else:
        return np.zeros((3,),dtype = "uint8")
```

几何变换——最近邻插值

◆特点

——计算量较小，速度快

——可能会造成灰度上的不连续，在灰度变化的地方可能出现明显的锯齿状

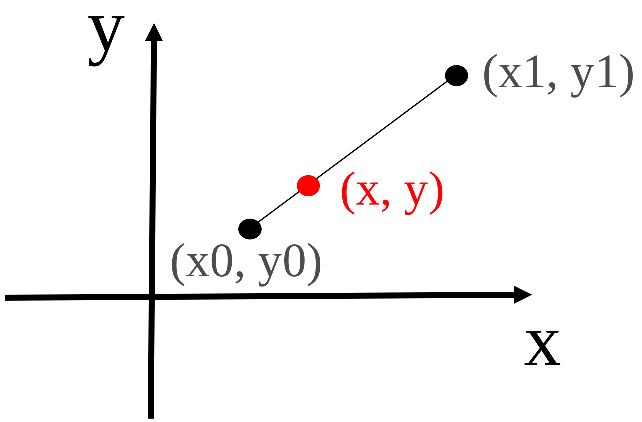


几何变换——双线性内插

◆原理

- 线性内插

　　已知数据 (x0, y0) 与 (x1, y1)，要计算 [x0, x1] 区间内某一位置 x 在直线上的y值：



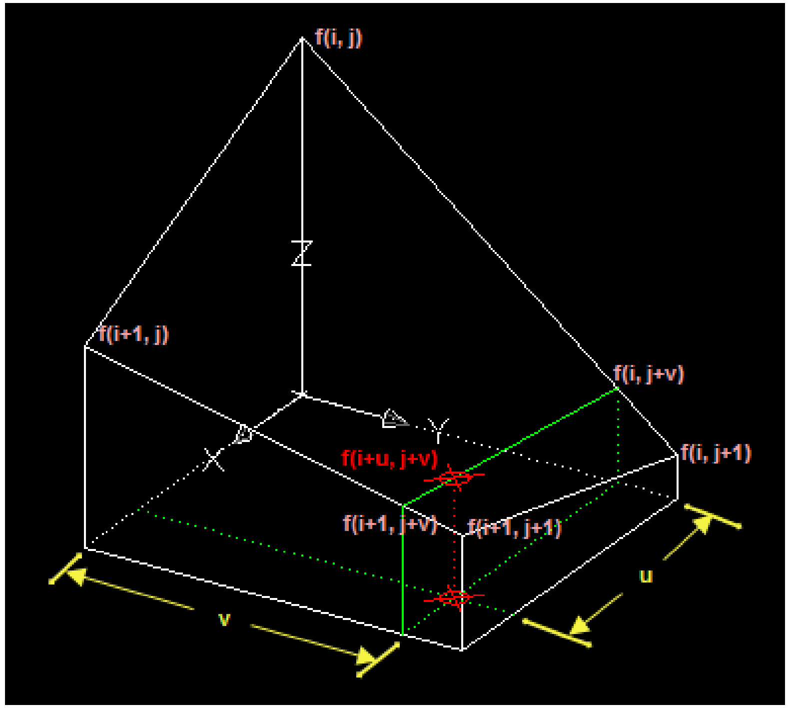
$$\frac{y-y_0}{x-x_0}=\frac{y_1-y_0}{x_1-x_0}$$

$$y=\frac{x_1-x}{x_1-x_0}y_0+\frac{x-x_0}{x_1-x_0}y_1$$

几何变换——双线性内插

◆基本思想

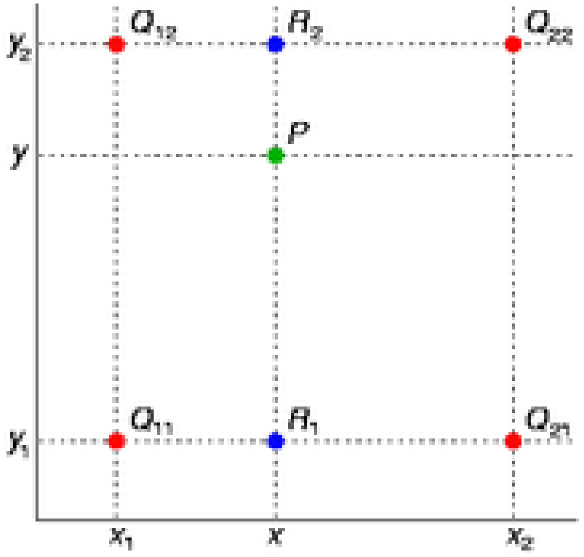
　　它认为某一点处的取值可以由它周围四个点的取值决定，且距离越近，决定作用越大，因此赋予一个更大的权重，这样我们利用周围四个点的值加权求和就能计算出待求像素点的值了。



几何变换——双线性内插

◆算法原理

已知数据$Q_{11}(x_1,y_1)$, $Q_{12}(x_1,y_2)$，$Q_{21}(x_2,y_1)Q_{22}(x_2,y_2)$, 计算$P(x,y)$点的灰度值，计算过程如下：



$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \quad where\ R_1 = (x, y_1)$$

$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \quad where\ R_2 = (x, y_2)$$

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2) \quad where\ \mathrm{P} = (x, y)$$

几何变换——双线性内插

◆算法实现

[59]:
```
def bilinear_interpolation(img_ori:'原始图像',pos_ori:'坐标'):
    h,w,c = img_ori.shape
    center_x,center_y = pos_ori
    x0 = int(center_x)
    x1 = x0 + 1
    y0 = int(center_y)
    y1 = y0 + 1
    if x1 in range(h) and y1 in range(w):
        temp1 = (center_y-y0)*img_ori[x1,y1]+(y1-center_y)*img_ori[x1,y0]
        temp2 = (center_y-y0)*img_ori[x0,y1]+(y1-center_y)*img_ori[x0,y0]
        return (center_x-x0)*temp1 +(x1-center_x)*temp2
    else:
        return np.zeros((3,),dtype = "uint8")
```

几何变换——双线性内插

◆特点

——计算量较大

——不会出现灰度的不连续，插值结果基本满意

——具有低通滤波的性质，使高频分量受损

几何变换——双线性内插

◆最近邻插值与双线性内插

最近邻插值

双线性内插

几何变换——其他

◆双立方（三次）卷积插值(4x4像素邻域)

https://dailc.github.io/2017/11/01/imageprocess_bicubicinterpolation.html

◆Lanczos插值(8x8像素邻域)

https://www.jianshu.com/p/8ae52a88ca61

几何变换——作业

◆课后作业

- 以图像中心(a, b)为旋转中心，旋转角度θ，计算图像上任一点(x1, y1)旋转之后的坐标(x2, y2)，请写出推导过程。
- 学习python自带(opencv ，PIL均可)的相关函数，实现图像的缩放(resize)，旋转(rotate)和置换(transpos)变换；请自行查找相关文档并了解函数参数含义。
- 打开一张图片，实现图像绕任意点旋转的函数rotate(img,theta,(x,y)) ,输入为图像、角度和旋转中心，插值方法任选，旋转后的图像信息完整即可。

Tips1：注意循环的边界条件，不要超出图像
Tips2：注意数组的下标大小
Tips3：将不再原图上的像素做不显示处理

- 完成实验报告（如果可以的话希望是PDF谢谢）