Problem 6

Digital Image Processing

肖逸群 116033910112

Geometric transform

问题描述

> Develop a geometric transform program that will rotate, translate, and scale an image by specified amounts, using the nearest neighbor and bilinear interpolation methods, respectively.

算法思想

在数字图像处理领域中，图像的几何变换包含了两种基本操作：

1. **旋转变换**：旋转变换对图片中的像素点进行重新整理排列
2. **灰度值篡改**：对图片中的像素点的灰度值进行修改

图像的几何变换，通常包括图像的平移、图像的镜像变换、图像的转置、图像的缩放和图像的旋转等。在本题中，我们需要对图像进行旋转变换、平移变换、缩放变换。

以下是几种变换的算法思想：

1. **旋转变换**
   一般图像的旋转是以图像的中心为原点，旋转一定的角度。旋转后，图像的大小一般会改变。和图像平移一样，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。
   在我的算法中，我直接把转出显示区域的图像截去。
2. **平移变换**
   图像平移就是将图像中所有的点都按照指定的平移量水平、垂直移动。设 (x0, y0) 为原图像上的一点，图像水平平移量为 tx，垂直平移量为 ty，则平移后点 (x0, y0) 坐标将变为 (x1, y1)。同样，若有点不在原图中，也就说明原图中有点被移出显示区域。如果不想丢失被移出的部分图像，可以将新生成的图像宽度扩大 |tx|，高度扩大 |ty|。
   在我的算法中，我直接把移除显示区域的部分给丢弃了，这样更能呈现变换后的结果。并且我对 x, y 的平移量设为二者相同的 delta。
3. **缩放变换**
   图像的缩放操作将会改变图像的大小，产生的图像中的像素可能在原图中找不到相应的像素点，这样就必须进行近似处理。一般的方法是直接赋值为和它最相近的像素值，也可以通过一些插值算法来计算。

源码分析

- 实验过程如下：
  a. 读取原始图片并将其转为灰度图像
  b. 对原始图片进行两种算法的旋转变换，在实验中我把角度设为 45 度
  c. 对原始图片进行两种算法的平移变换，并设置不同的平移值，观看不同 delta 下的实验效果
  d. 对原始图片进行两种算法的缩放变换，并设置不同的缩放量

```
1 def main():
```

```
    originalImage = Image.open('./resource/ray_trace_bottle.tif')

    plt.subplot(1, 3, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    rotateImage1 = geometryTransform.rotateOperate(originalImage, 45, 'nearest')
    plt.subplot(1, 3, 2)
    plt.imshow(rotateImage1, cmap = plt.get_cmap('gray'))
    plt.title('Rotate By Nearest Method')

    rotateImage2 = geometryTransform.rotateOperate(originalImage, 45, 'bilinear')
    plt.subplot(1, 3, 3)
    plt.imshow(rotateImage2, cmap = plt.get_cmap('gray'))
    plt.title('Rotate By Bilinear Method')

    plt.show()

    plt.subplot(1, 3, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    delta = 100
    translateImage1 = geometryTransform.translateOperate(originalImage, delta, 'nearest'
)
    plt.subplot(1, 3, 2)
    plt.imshow(translateImage1, cmap = plt.get_cmap('gray'))
    plt.title('Translate By Nearest Method(delta = %s)' % delta)

    translateImage2 = geometryTransform.translateOperate(originalImage, delta, 'bilinear
')
    plt.subplot(1, 3, 3)
    plt.imshow(translateImage2, cmap = plt.get_cmap('gray'))
    plt.title('Translate By Bilinear Method(delta = %s)' % delta)

    plt.show()

    scalingX = 0.9
    scalingY = 1.1
    plt.subplot(1, 3, 1)
    plt.imshow(originalImage, cmap = plt.get_cmap('gray'))
    plt.title('Original')

    scaleImage1 = geometryTransform.scaleOperate(originalImage, scalingX, scalingY, 'nea
rest')
    plt.subplot(1, 3, 2)
    plt.imshow(scaleImage1, cmap = plt.get_cmap('gray'))
    plt.title('Scale By Nearest Method(scalingX = %s, scalingY = %s)' % (scalingX, scali
ngY))

    scaleImage2 = geometryTransform.scaleOperate(originalImage, scalingX, scalingY, 'bil
inear')
    plt.subplot(1, 3, 3)
    plt.imshow(scaleImage2, cmap = plt.get_cmap('gray'))
    plt.title('Scale By Bilinear Method(scalingX = %s, scalingY = %s)' % (scalingX, scal
ingY))

    plt.show()
```

- nearest 与 bilinear 算法函数

```python
def interpolateTransform(originalImage, x, y, method):
    m, n = originalImage.size
    pix = originalImage.load()

    a = np.floor(x)
    b = np.floor(y)

    if method == 'nearest':
        px = a if x - a < 0.5 else a + 1
        py = b if y - b < 0.5 else b + 1

        if px < 0 or px >= m or py < 0 or py >= n:
            return 0
        else:
            return pix[px, py]
    elif method == 'bilinear':
        res = [0, 0, 0, 0]
        if a >= 0 and a < m:
            if b >= 0 and b < n:
                res[0] = pix[a, b]
            if b + 1 >= 0 and b + 1 < n:
                res[1] = pix[a, b+1]

        if a + 1 >= 0 and a + 1 < m:
            if b >= 0 and b < n:
                res[2] = pix[a+1, b]
            if b + 1 >= 0 and b + 1 < n:
                res[3] = pix[a+1, b+1]

        res1 = res[0] + (res[1] - res[0]) * (y - b)
        res2 = res[2] + (res[3] - res[2]) * (y - b)

        return res1 + (res2 - res1) * (x - a)
```

- 旋转变换函数

```python
def rotateOperate(originalImage, angle, method):
    m, n = originalImage.size
    ox = m / 2
    oy = n / 2
    rotateImage = originalImage.copy()
    dest = rotateImage.load()

    for i in range(m):
        for j in range(n):
            x = (i - ox) * np.cos(np.radians(-angle)) - (j - oy) * np.sin(np.radians(-angle)) + ox
            y = (i - ox) * np.sin(np.radians(-angle)) + (j - oy) * np.cos(np.radians(-angle)) + oy
            dest[i, j] = int(interpolateTransform(originalImage, x, y, method) )

    return rotateImage
```

- 平移变换函数

```python
def translateOperate(originalImage, delta, method):
    m, n = originalImage.size
    translateImage = originalImage.copy()
    dest = translateImage.load()

    for i in range(m):
```

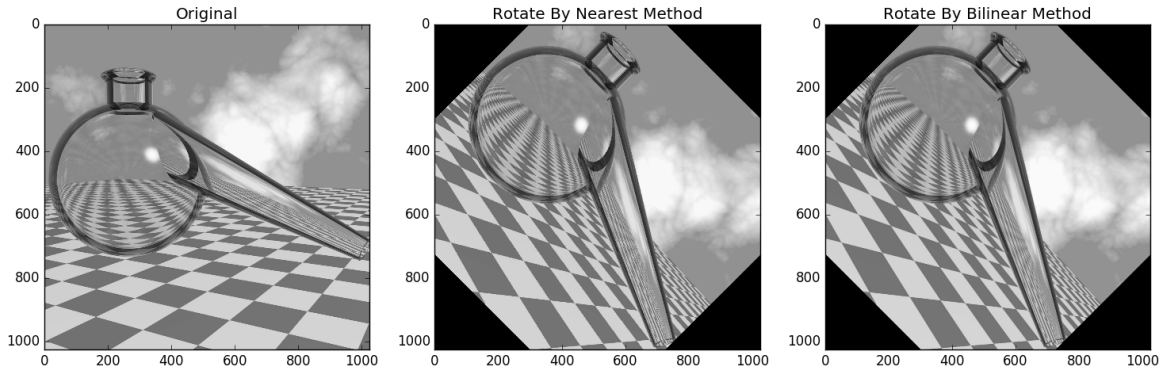
```
        for j in range(n):
            x = i - delta
            y = j - delta
            dest[i, j] = int(interpolateTransform(originalImage, x, y, method) )

    return translateImage
```

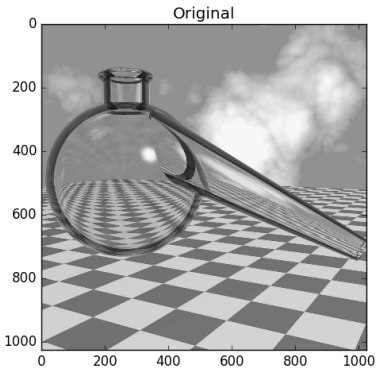
- 缩放变换函数

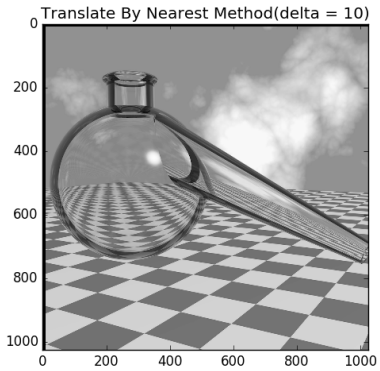
```
def scaleOperate(originalImage, scalingX, scalingY, method):
    m, n = originalImage.size
    scaleImage = originalImage.copy()
    dest = scaleImage.load()
    ox = m / 2
    oy = n / 2

    for i in range(m):
        for j in range(n):
            x = (i - ox) / scalingX + ox
            y = (j - oy) / scalingY + oy
            dest[i, j] = int(interpolateTransform(originalImage, x, y, method) )

    return scaleImage
```

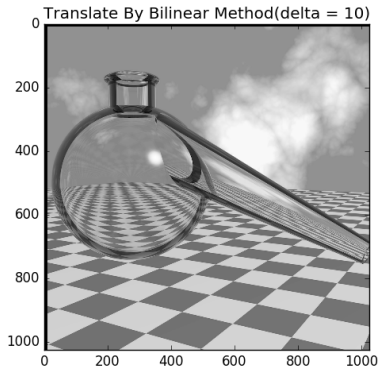
实验结果

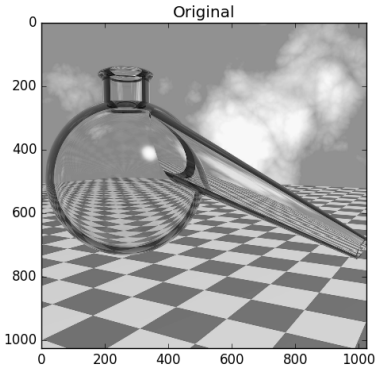
下列图片为不同算法和变量下三种变换操作的结果：

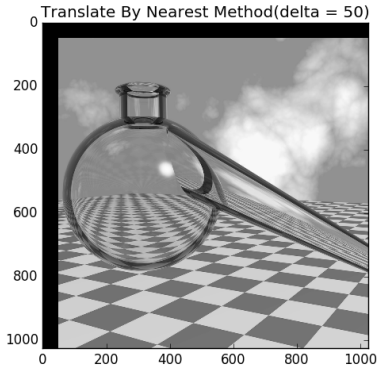


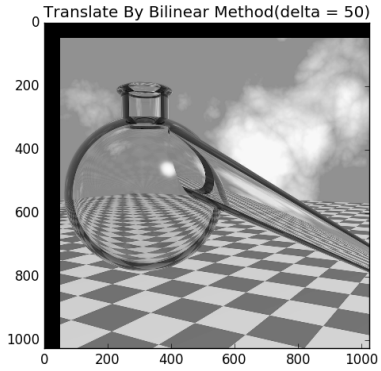

Original
0
200
400
600
800
1000
0
200
400
600
800
1000

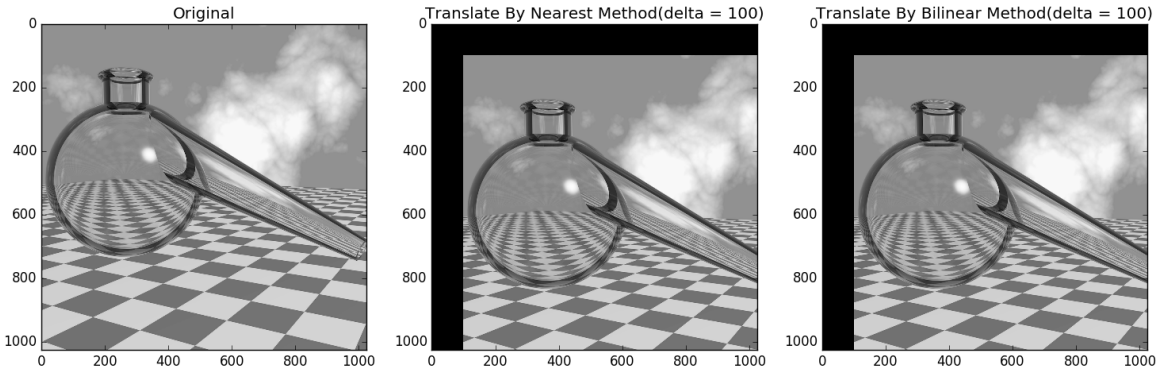

Translate By Nearest Method(delta = 10)
0
200
400
600
800
1000
0
200
400
600
800
1000

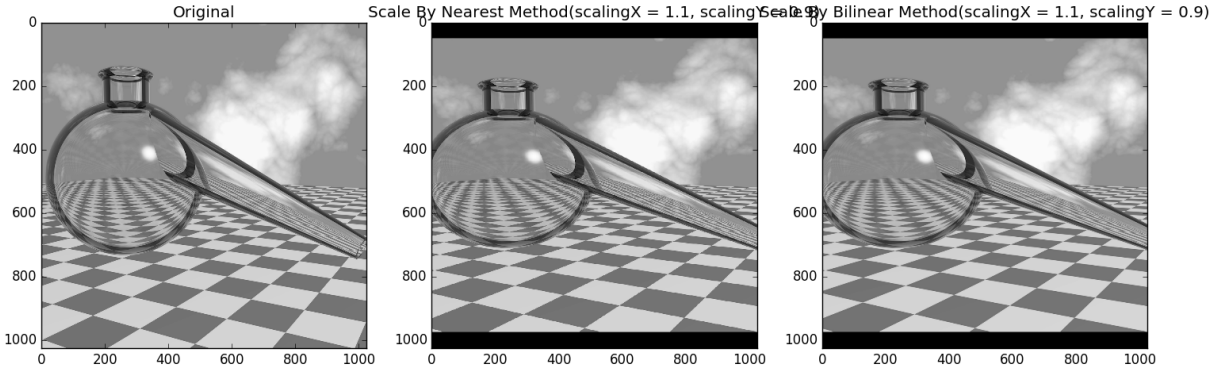

Translate By Bilinear Method(delta = 10)
0
200
400
600
800
1000
0
200
400
600
800
1000

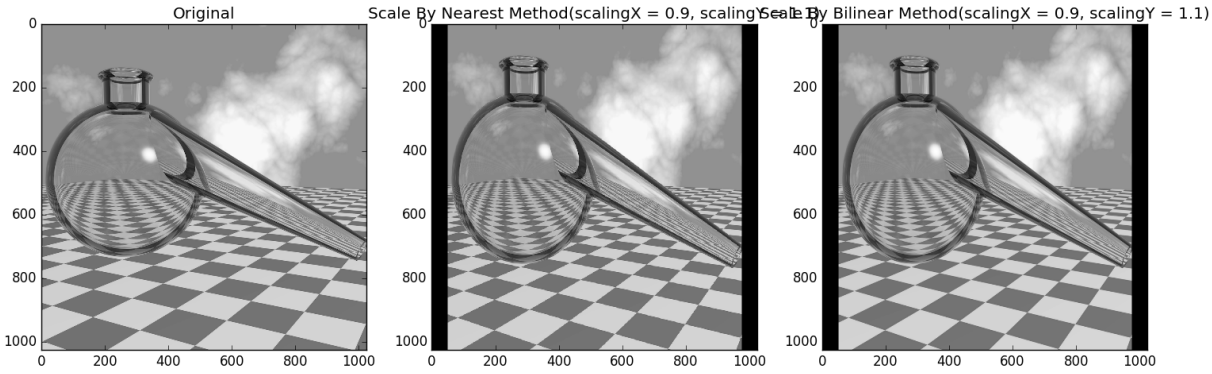

Original
0
200
400
600
800
1000
0
200
400
600
800
1000


Translate By Nearest Method(delta = 50)
0
200
400
600
800
1000
0
200
400
600
800
1000


Translate By Bilinear Method(delta = 50)
0
200
400
600
800
1000
0
200
400
600
800
1000


Original
Translate By Nearest Method(delta = 100)
Translate By Bilinear Method(delta = 100)


Original
Scale By Nearest Method(scalingX = 1.1, scalingY = 0.9)
Scale By Bilinear Method(scalingX = 1.1, scalingY = 0.9)


Original
Scale By Nearest Method(scalingX = 0.9, scalingY = 1.1)
Scale By Bilinear Method(scalingX = 0.9, scalingY = 1.1)
0
200
400
600
800
1000