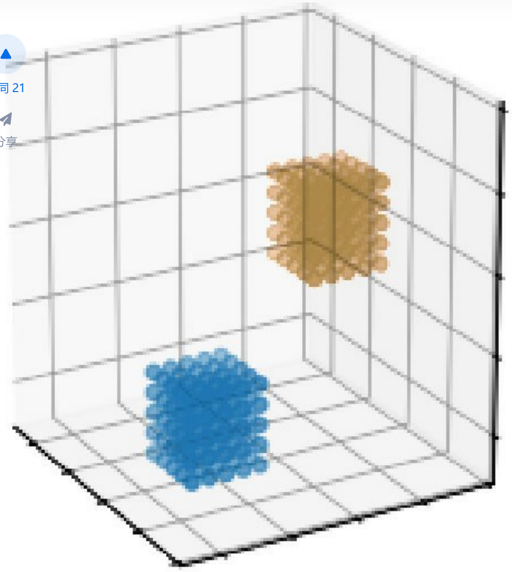


【深度好文】3D坐标系下的点的转换矩阵（平移、缩放、旋转、错切）

卓不凡

上一节中往我们介绍了2D坐标系下点的转换矩阵，本节将2D坐标系扩展到3D坐标系，来研究对应的转换矩阵。

1. 平移 （Translation）

在3D空间中，假设我们需要将一个点平移到另一个位置。假设空间中的一点P，其用坐标表示为（x,y,z）；将其向 x方向平移 tx，向y方向平移ty, 向z方向平移tz, 设平移后点的坐标为（x′，y′，z′）,则上述点的平移操作可以归纳为如下公式：

$$
\begin{aligned}
x' &= x + t_x \\
y' &= y + t_y \\
z' &= z + t_z
\end{aligned}
$$

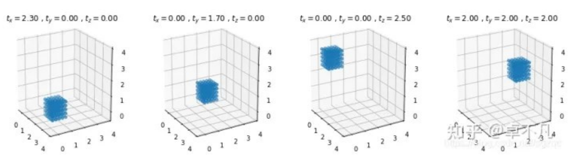
使用齐次矩阵表示如下：

$$
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
$$

将上述过程用代码实现如下：

```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
X, Y, Z = np.mgrid[0:1:5j, 0:1:5j, 0:1:5j]
x, y, z = X.ravel(), Y.ravel(), Z.ravel()
def trans_translate(x, y, z, tx, ty, tz):
    T = [[1, 0, 0, tx],
         [0, 1, 0, ty],
         [0, 0, 1, tz],
         [0, 0, 0, 1 ]]
    T = np.array(T)
    P = np.array([x, y, z, [1]*x.size])
    return np.dot(T, P)

fig, ax = plt.subplots(1, 4, subplot_kw={'projection': '3d'})

T_ = [[2.3, 0, 0], [0, 1.7, 0], [0, 0, 2.5], [2, 2, 2]]

for i in range(4):
    tx, ty, tz = T_[i]
    x_, y_, z_, _ = trans_translate(x, y, z, tx, ty, tz)
    ax[i].view_init(20, -30)
    ax[i].scatter(x_, y_, z_)
    ax[i].set_title(r'$t_x={0:.2f}$ , $t_y={1:.2f}$ , $t_z={2:.2f}$'.format(tx, ty, t

    ax[i].set_xlim([-0.5, 4])
    ax[i].set_ylim([-0.5, 4])
    ax[i].set_zlim([-0.5, 4])

plt.show()
```

效果如下：



动态效果如下：

2. 缩放 （Scaling）

在3D空间中，对点（x,y,z）常用的另一种操作为相对于另一点(px,py,pz)进行缩放操作，我们不妨x方向的缩放因子为sx,y方向的缩放因子为sy,z方向的缩放因子为sz, 则上述点（x,y,z）相对于点（px,py,pz）的缩放操作可以归纳为如下公式:

$$
\begin{aligned}
x' &= s_x(x-p_x)+p_x = s_x x + p_x(1-s_x)\\
y' &= s_y(y-p_y)+p_y = s_y y + p_y(1-s_y)\\
z' &= s_z(z-p_z)+p_z = s_z z + p_z(1-s_z)
\end{aligned}
$$

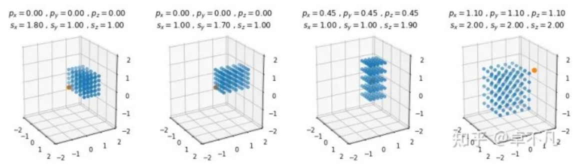
使用齐次矩阵表示如下:

$$
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 & p_x(1-s_x) \\ 0 & s_y & 0 & p_y(1-s_y) \\ 0 & 0 & s_z & p_z(1-s_z) \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
$$

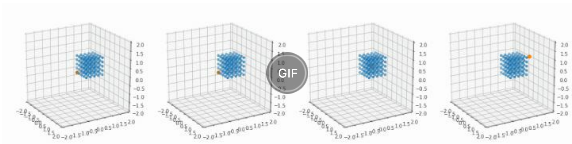
将上述过程用代码实现如下:

```
def trans_scale(x, y, z,
                px, py, pz,
                sx, sy, sz):
    T = [[sx, 0 , 0 , px*(1 - sx)],
         [0 , sy, 0 , py*(1 - sy)],
         [0 , 0 , sz, pz*(1 - sz)],
         [0 , 0 , 0 , 1          ]]
    T = np.array(T)
    P = np.array([x, y, z, [1]*x.size])
    return np.dot(T, P)

fig, ax = plt.subplots(1, 4, subplot_kw={'projection': '3d'})

S_ = [[1.8, 1, 1], [1, 1.7, 1], [1, 1, 1.9], [2, 2, 2]]
P_ = [[0, 0, 0], [0, 0, 0], [0.45, 0.45, 0.45], [1.1, 1.1, 1.1]]

for i in range(4):
    sx, sy, sz = S_[i]; px, py, pz = P_[i]
    x_, y_, z_, _ = trans_scale(x, y, z, px, py, pz, sx, sy, sz)
    ax[i].view_init(20, -30)
    ax[i].scatter(x_, y_, z_)
    ax[i].scatter(px, py, pz, s=50)
    ax[i].set_title(
        r'$p_x={0:.2f}$ , $p_y={1:.2f}$ , $p_z={2:.2f}$'.format(px, py, pz) + '\n'
        r'$s_x={0:.2f}$ , $s_y={1:.2f}$ , $s_z={2:.2f}$'.format(sx, sy, sz)
    )

    ax[i].set_xlim([-2, 2])
    ax[i].set_ylim([-2, 2])
    ax[i].set_zlim([-2, 2])

plt.show()
```

效果如下:



动态效果如下:



3. 旋转（Rotation）

在3D空间中，对点（x,y,z）常用的另一种操作为相对于另一点(px,py,pz)进行旋转操作，我们依旧采用右手坐标系，即旋转角的正方向为逆时针方向。旋转我们可分为绕x轴、y轴、z轴旋转。假设绕x轴旋转角度为alpha,绕y轴旋转角度为beta,绕z轴旋转的角度为gamma。则相应的变换如下:

1）绕x轴旋转

公式如下:

$$
\begin{aligned}
y' &= (y-p_y)\cos\alpha - (z-p_z)\sin\alpha + p_y = y\cos\alpha - z\sin\alpha + p_y(1-\cos\alpha) + p_z\sin\alpha\\
z' &= (y-p_y)\sin\alpha + (z-p_z)\cos\alpha + p_z = y\sin\alpha + z\cos\alpha + p_z(1-\cos\alpha) - p_y\sin\alpha
\end{aligned}
$$

使用齐次矩阵表示如下:

$$
\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha & p_y(1-\cos\alpha)+p_z\sin\alpha \\ 0 & \sin\alpha & \cos\alpha & p_z(1-\cos\alpha)-p_y\sin\alpha \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}
$$

2）绕y轴旋转

公式如下:

$$
\begin{aligned}
x' &= (x-p_x)\cos\beta + (z-p_z)\sin\beta + p_x = x\cos\beta + z\sin\beta + p_x(1-\cos\beta) - p_z\sin\beta\\
z' &= -(y-p_y)\sin\beta + (z-p_z)\cos\beta + p_z = -x\sin\beta + z\cos\beta + p_z(1-\cos\beta) + p_x\sin\beta
\end{aligned}
$$

使用齐次矩阵表示如下：

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\beta & 0 & \sin\beta & p_x(1-\cos\beta) - p_z\sin\beta \\ 0 & 1 & 0 & 0 \\ -\sin\beta & 0 & \cos\beta & p_z(1-\cos\beta) + p_x\sin\beta \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

3）绕 z 轴旋转

公式如下：

$$x' = (x-p_x)\cos\gamma - (y-p_y)\sin\gamma + p_x = x\cos\gamma - y\sin\gamma + p_x(1-\cos\gamma) + p_y\sin\gamma$$
$$y' = (x-p_x)\sin\gamma + (y-p_y)\cos\gamma + p_y = x\sin\gamma + y\cos\gamma + p_y(1-\cos\gamma) - p_x\sin\gamma$$

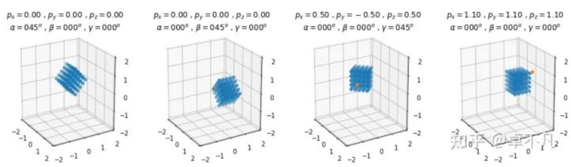
使用齐次矩阵表示如下：

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 & p_x(1-\cos\gamma) + p_y\sin\gamma \\ \sin\gamma & \cos\gamma & 0 & p_y(1-\cos\gamma) - p_x\sin\gamma \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

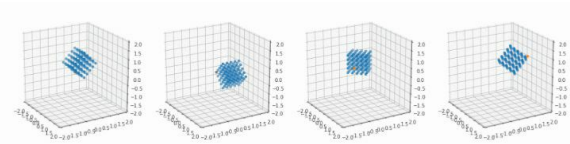
将上述过程用代码实现如下：

```
def trans_rotate(x, y, z, px, py, pz, alpha, beta, gamma):
    alpha, beta, gamma = np.deg2rad(alpha), np.deg2rad(beta), np.deg2rad(gamma)
    Rx = [[1, 0             ,  0             , 0
          [0, np.cos(alpha), -np.sin(alpha), py*(1 - np.cos(alpha)) + pz*np.sin(alph
          [0, np.sin(alpha),  np.cos(alpha), pz*(1 - np.cos(alpha)) - py*np.sin(alph
          [0, 0             ,  0             , 1
    Ry = [[ np.cos(beta), 0, np.sin(beta), px*(1 - np.cos(beta)) - pz*np.sin(beta)],
          [ 0           , 1, 0           , 0                                       ],
          [-np.sin(beta), 0, np.cos(beta), pz*(1 - np.cos(beta)) + px*np.sin(beta)],
          [ 0           , 0, 0           , 1                                       ]]
    Rz = [[np.cos(gamma), -np.sin(gamma), 0, px*(1 - np.cos(gamma)) + py*np.sin(gamm
          [np.sin(gamma),  np.cos(gamma), 0, py*(1 - np.cos(gamma)) - px*np.sin(gamm
          [0            ,  0            , 1, 0
          [0            ,  0            , 0, 1

    Rx = np.array(Rx); Ry = np.array(Ry); Rz = np.array(Rz)
    P = np.array([x, y, z, [1]*x.size])
    return np.dot(np.dot(np.dot(Rx, Ry), Rz), P)

fig, ax = plt.subplots(1, 4, subplot_kw={'projection': '3d'})

R_ = [[45, 0, 0],
      [0, 45, 0],
      [0, 0, 45],
      [0, 0, 0]]
P_ = [[0, 0, 0],
      [0, 0, 0],
      [0.5, -0.5, 0.5],
      [1.1, 1.1, 1.1]]

for i in range(4):
    alpha, beta, gamma = R_[i]; px, py, pz = P_[i]
    x_, y_, z_, _ = trans_rotate(x, y, z, px, py, pz, alpha, beta, gamma)
    ax[i].view_init(20, -30)
    ax[i].scatter(x_, y_, z_)
    ax[i].scatter(px, py, pz)
    ax[i].set_title(
        r'$p_x={0:.2f}$ , $p_y={1:.2f}$ , $p_z={2:.2f}$'.format(px, py, pz) + '\n'
        r'$\alpha={0:03d}^o$ , $\beta={1:03d}^o$ , $\gamma={2:03d}^o$'.format(alpha,
    )

    ax[i].set_xlim([-2, 2])
    ax[i].set_ylim([-2, 2])
    ax[i].set_zlim([-2, 2])

plt.show()
```

效果如下：



动态效果如下：



4. 错切（Shearing）

在3D空间中，对点（x,y,z）常用的另一种操作为相对于另一点(px,py,pz)进行错切操作。不妨假设在yz平面的投影相对于y轴的错切参数为lambaxy,相对于z轴的错切参数为lambaxz;在xz平面的投影相对于x轴的错切参数为lambayx,相对于z轴的错切参数为lambayz;在xy平面的投影相对于x轴的错切参数为lambazx,相对于y轴的错切参数为lambazy。则上述点（x,y,z）相对于点（px,py,pz）的错切操作可以归纳为如下公式：

$$x' = x + \lambda_x^y(y-p_y) + \lambda_x^z(z-p_z) = x + \lambda_x^y y + \lambda_x^z z - (\lambda_x^y + \lambda_x^z)p_x$$
$$y' = y + \lambda_y^x(x-p_x) + \lambda_y^z(z-p_z) = y + \lambda_y^x x + \lambda_y^z z - (\lambda_y^x + \lambda_y^z)p_y$$
$$z' = z + \lambda_z^x(x-p_x) + \lambda_z^y(y-p_y) = z + \lambda_z^x x + \lambda_z^y y - (\lambda_z^x + \lambda_z^y)p_z$$

使用齐次矩阵表示如下：

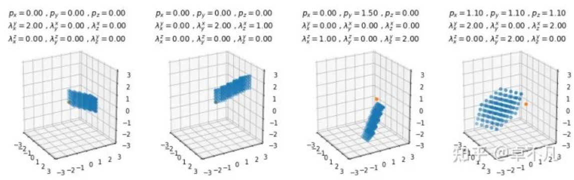
$$\begin{bmatrix} x' \\ \vdots \end{bmatrix} = \begin{bmatrix} 1 & \lambda_x^y & \lambda_x^z & -(\lambda_x^y + \lambda_x^z)p_x \\ \vdots \end{bmatrix} \begin{bmatrix} x \\ \vdots \end{bmatrix}$$

$$
\begin{bmatrix} y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \lambda_y^x & 1 & \lambda_y^z & -(\lambda_y^x+\lambda_y^z)p_y \\ \lambda_z^x & \lambda_z^y & 1 & -(\lambda_z^x+\lambda_z^y)p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} y \\ z \\ 1 \end{bmatrix}
$$

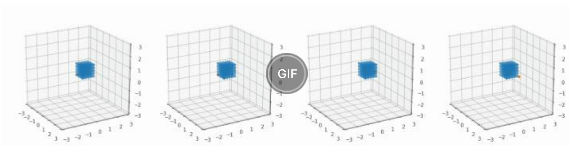
将上述过程用代码实现如下：

```
def trans_shear(x, y, z, px, py, pz,
                lambdaxy, lambdaxz,
                lambdayx, lambdayz,
                lambdazx, lambdazy):
    T = [[1       , lambdaxy, lambdaxz, -(lambdaxy+lambdaxz)*px],
         [lambdayx, 1       , lambdayz, -(lambdayx+lambdayz)*py],
         [lambdazx, lambdazy, 1       , -(lambdazx+lambdazy)*py],
         [0       , 0       , 0       , 1                      ]]
    T = np.array(T)
    P = np.array([x, y, z, [1]*x.size])
    return np.dot(T, P)

fig, ax = plt.subplots(1, 4, subplot_kw={'projection': '3d'})

L_ = [[[2, 0], [0, 0], [0, 0]],
      [[0, 0], [2, 0], [1, 0]],
      [[0, 1], [0, 0], [0, 2]],
      [[2, 0], [0, 2], [2, 0]]]
P_ = [[0, 0, 0], [0, 0, 0], [0, 1.5, 0], [1.1, 1.1, 1.1]]

for i in range(4):
    lambdax, lambday, lambdaz = L_[i]; px, py, pz = P_[i]
    x_, y_, z_, _ = trans_shear(x, y, z, px, py, pz,
                                *lambdax, *lambday, *lambdaz)
    ax[i].view_init(20, -30)
    ax[i].scatter(x_, y_, z_)
    ax[i].scatter(px, py)
    ax[i].set_title(
        r'$p_x={0:.2f}$ , $p_y={1:.2f}$ , $p_z={2:.2f}$'.format(px, py, pz) + '\n'
        r'$\lambda_x^y={0:.2f}$ , $\lambda_y^x={1:.2f}$ , $\lambda_z^x={2:.2f}$'.form
        r'$\lambda_x^z={0:.2f}$ , $\lambda_y^z={1:.2f}$ , $\lambda_z^y={2:.2f}$'.form
    )

    ax[i].set_xlim([-3, 3])
    ax[i].set_ylim([-3, 3])
    ax[i].set_zlim([-3, 3])

plt.show()
```

效果如下：



动态效果如下：



5. 总结

有了以上平移、旋转、缩放和错切矩阵后，我们就可以通过矩阵乘法求得3D空间下点P任意变化后坐标。

编辑于 2021-07-09 15:50

笛卡尔坐标系 坐标 矩阵

写下你的评论...

还没有评论，发表第一个评论吧

文章被以下专栏收录

图像处理
图像处理基础知识