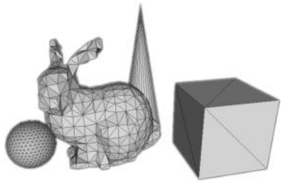


Introduction to Computer Graphics

Geometry Objects and Transformations

Ming Zeng

Software School

Xiamen University

Contact Information:

zengming@xmu.edu.cn

第四章 几何对象与变换
Geometry Objects and Transformations

- 第一次课：几何及其表示 (更注重数学概念)
  - 与坐标无关的几何：点、标量、向量
    - 向量空间、仿射空间、欧氏空间
  - 几何的坐标表示
  - 几何在OpenGL中是如何表示的（OpenGL中各种不同标架）
- 第二次课：变换及OpenGL实现（更注重实际算法运作）
  - 几何变换的数学表示
  - OpenGL中的几何变换
  - 专题：旋转的表示及其平滑操作

变换及OpenGL实现

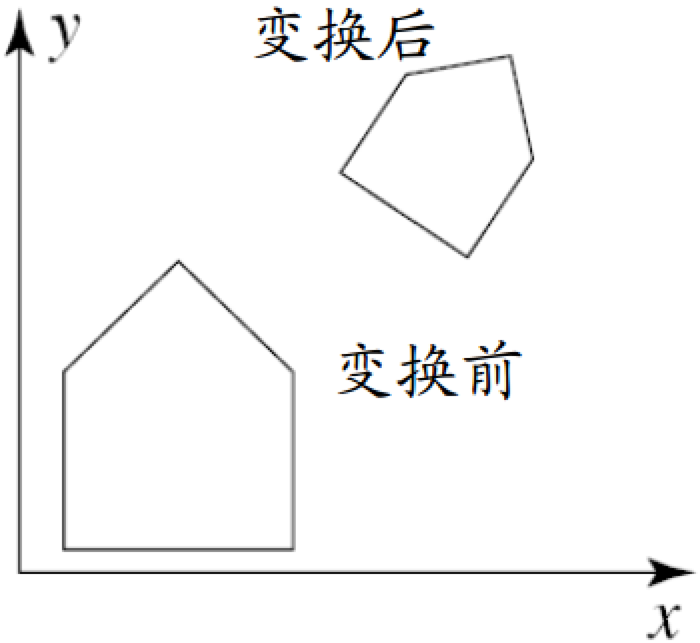
- 几何变换的数学表示
- OpenGL中的几何变换
- 专题：虚拟轨迹球

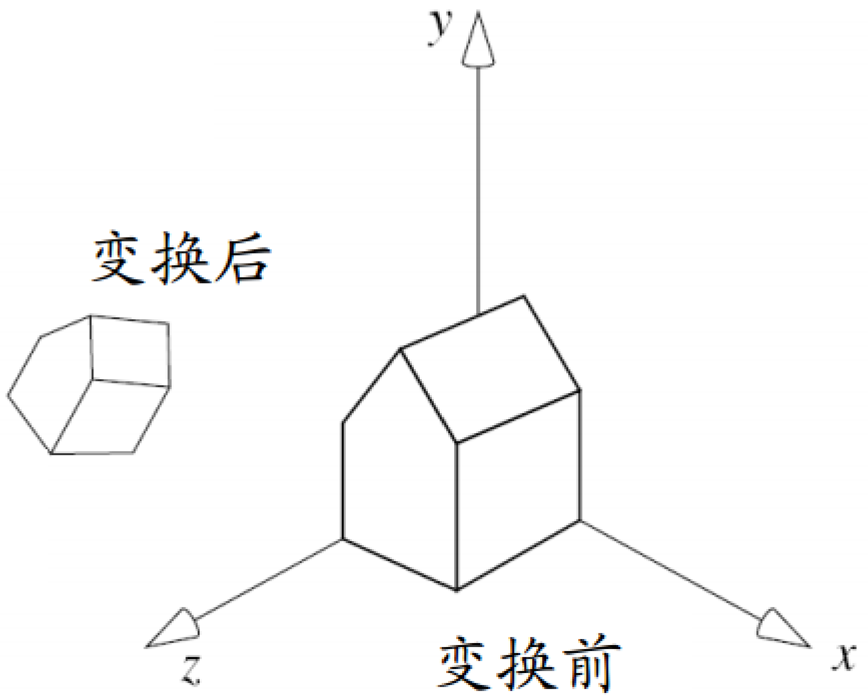
几何变换的数学表示

- 变换的作用及绘制流水线中变换的位置
- 基本变换
  - 平移 Translation
  - 旋转 Rotation
  - 缩放 Scaling
- 复杂变换
  - 特殊旋转
  - 变换的复合

变换的作用及绘制流水线中变换的位置

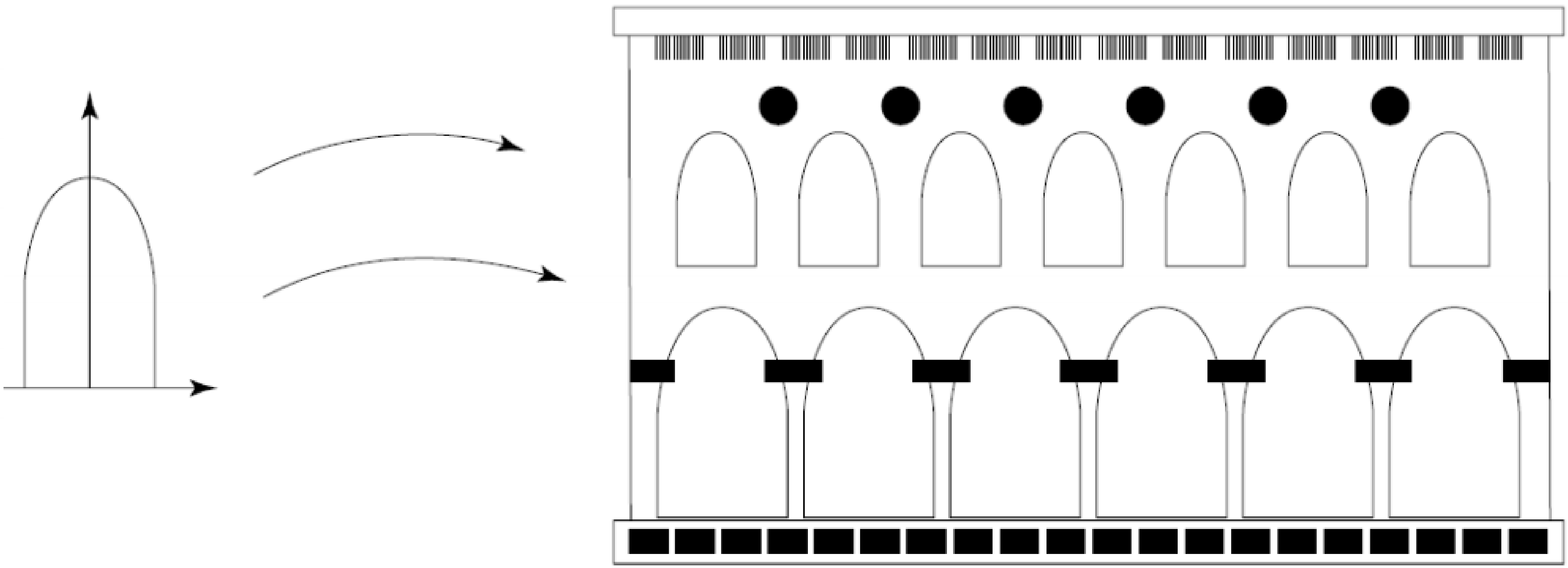
- 作用之一：改变物体的位置





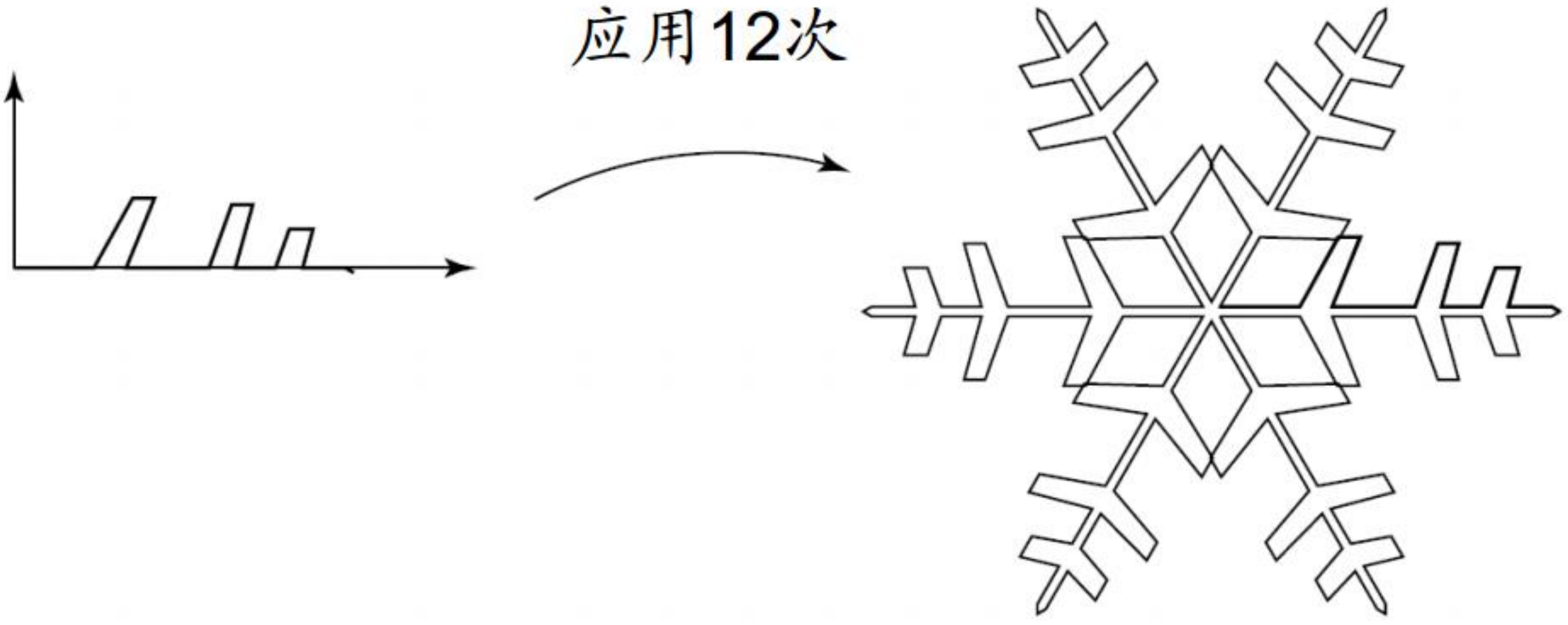
变换的作用及绘制流水线中变换的位置

- 例如：组合场景



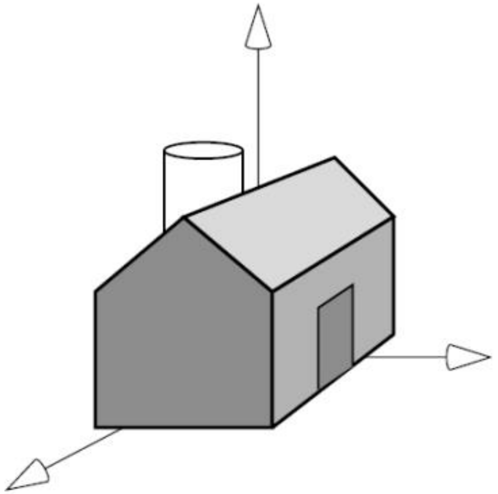
变换的作用及绘制流水线中变换的位置

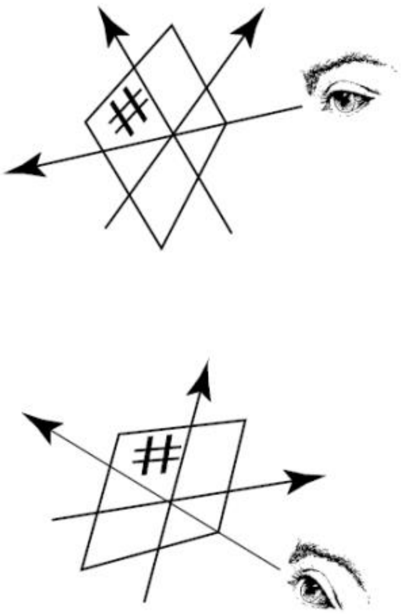
- 例如：构造雪花



变换的作用及绘制流水线中变换的位置

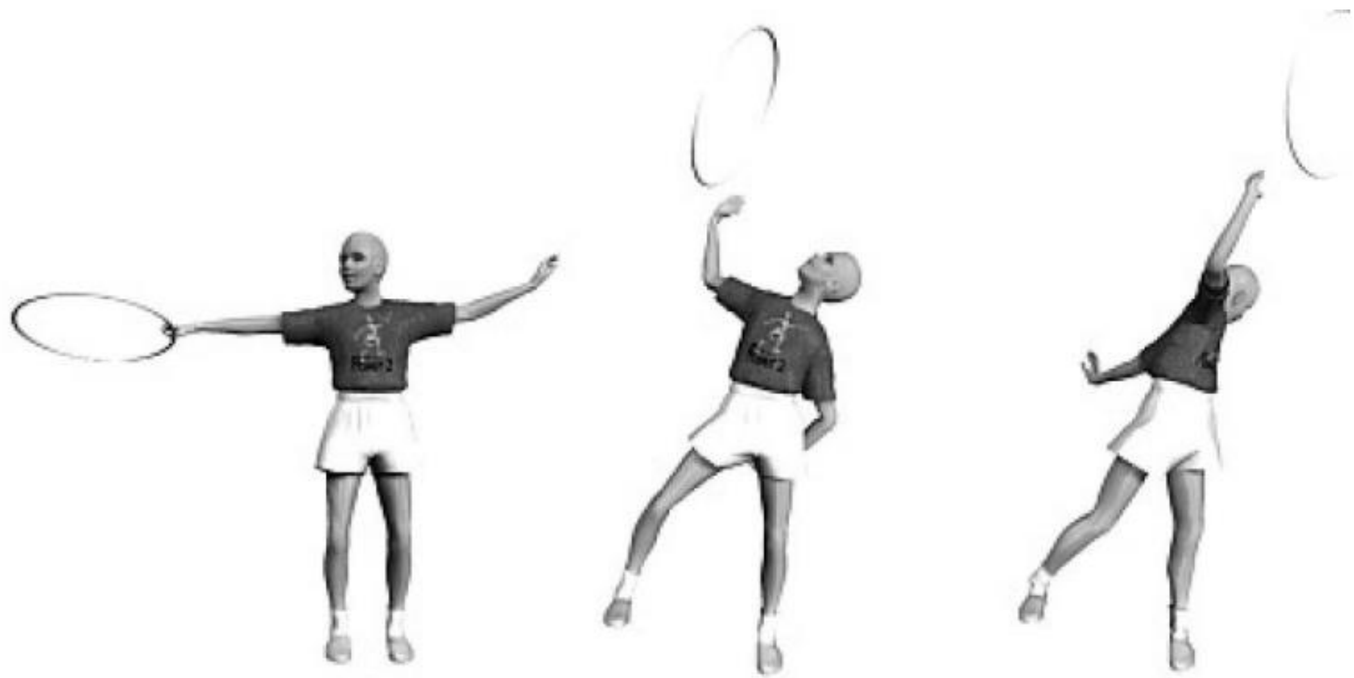
- 作用之二：改变观察视角
  - 设计者可能需要从不同的角度查看同一个场景，此时自然的方法是对象不变，而变换照相机的位置





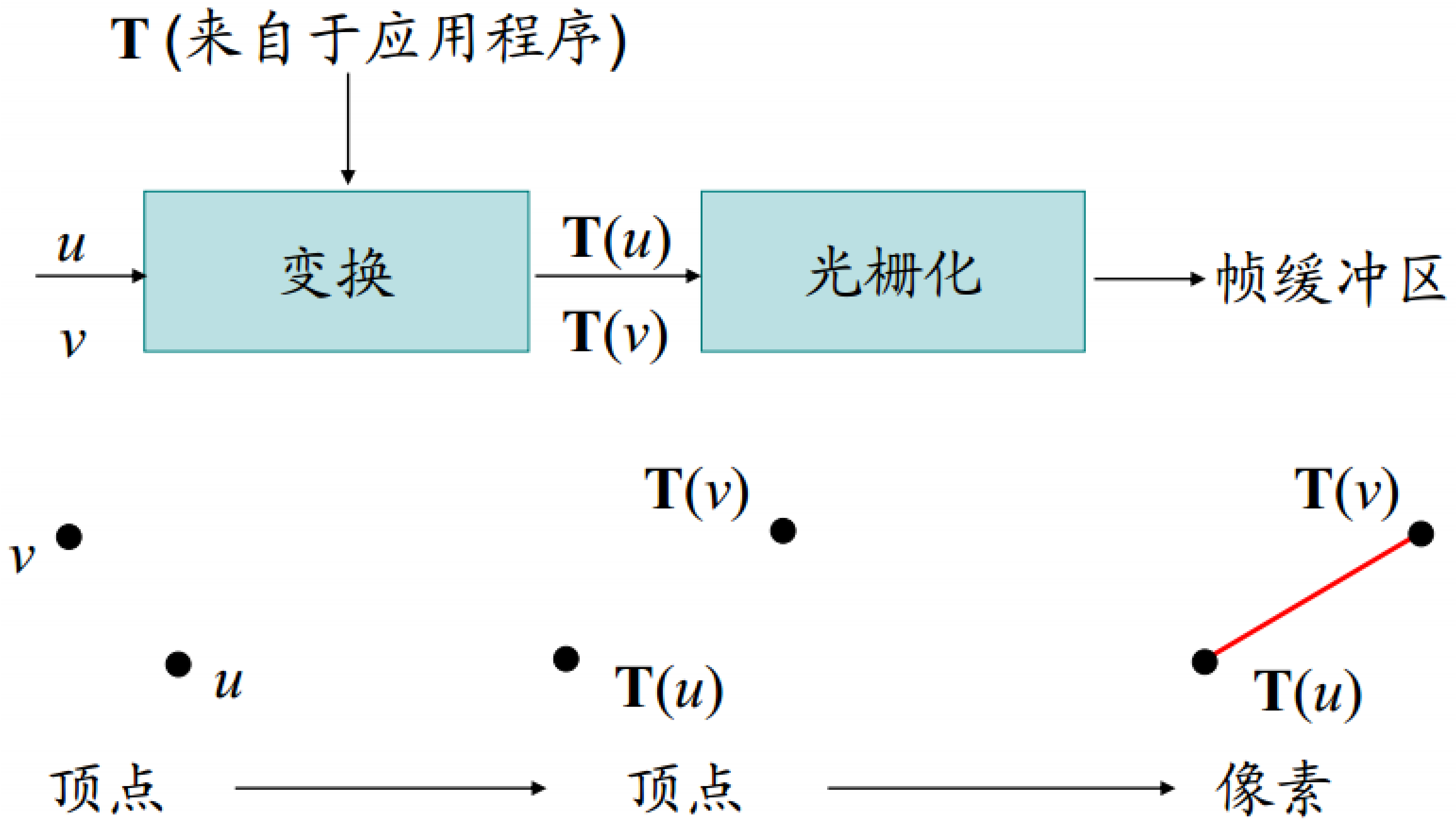
变换的作用及绘制流水线中变换的位置

- 作用之三：动画
  - 在计算机动画中，在相邻帧图像中，几个对象相对彼此的位置进行移动。这可以通过平移和旋转局部坐标系实现



变换的作用及绘制流水线中变换的位置

- 图形流水线流水线实现

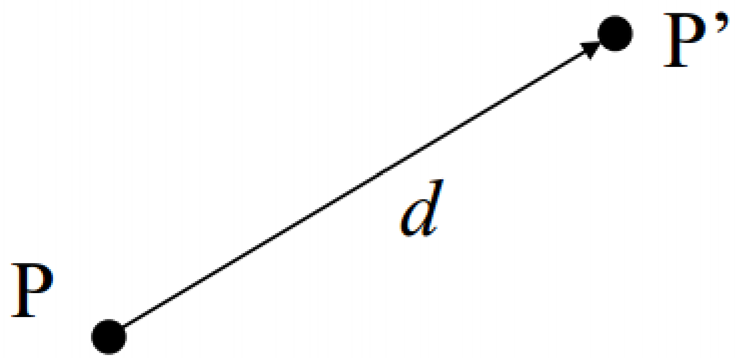


几何变换的数学表示

- 变换的作用及绘制流水线中变换的位置
- 基本变换
  - 平移 Translation
  - 旋转 Rotation
  - 缩放 Scaling
  - 错切 Shear
- 复杂变换
  - 逆变换
  - 变换的复合
  - 特殊旋转

平移（translation)

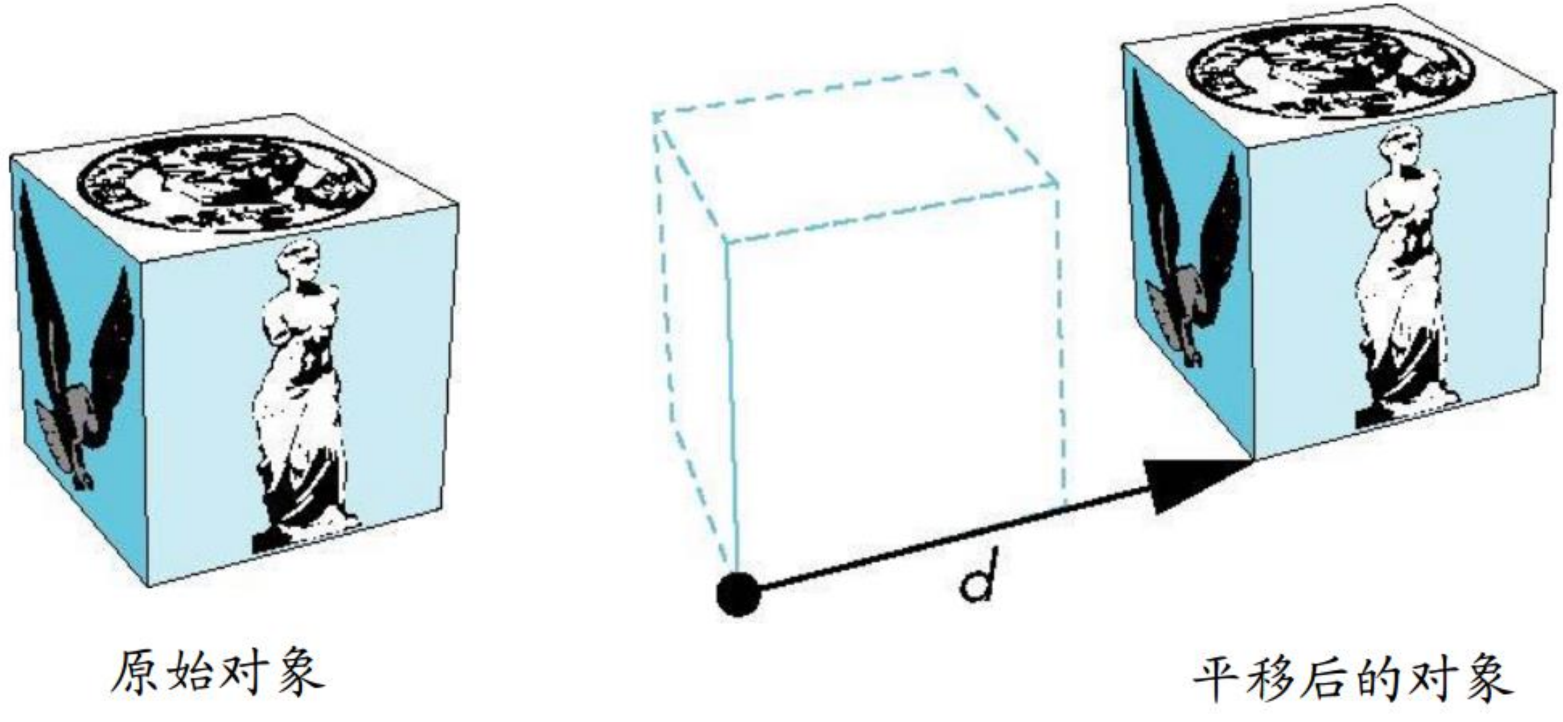
- 改变物体的位置



- 平移由平移向量d确定
  - 三个自由度 (degrees of freedom, DOF)
  - $P' = P + d$

对象的平移

- 把一个对象上的所有点沿同一方向移动相同距离



平移的表示

- 应用在某个标架中的齐次坐标表示

$\mathbf{p}=[\ x\ y\ z\ 1]^T$

$\mathbf{p}'=[x'\ y'\ z'\ 1]^T$

$\mathbf{d}=[d_x\ d_y\ d_z\ 0]^T$

注意：这里是四维的齐次坐标，用的是点=点+向量

第四维度为1表示点，为0表示方向（向量）

因此 $\mathbf{p}' = \mathbf{p} + \mathbf{d}$ 或者

$x'=x+d_x$

$y'=y+d_y$

$z'=z+d_z$

平移矩阵

- 可以用在齐次坐标中一个4x4的矩阵**T**表示平移：**p'=Tp** 其中

$$\mathbf{T} = \mathbf{T}(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

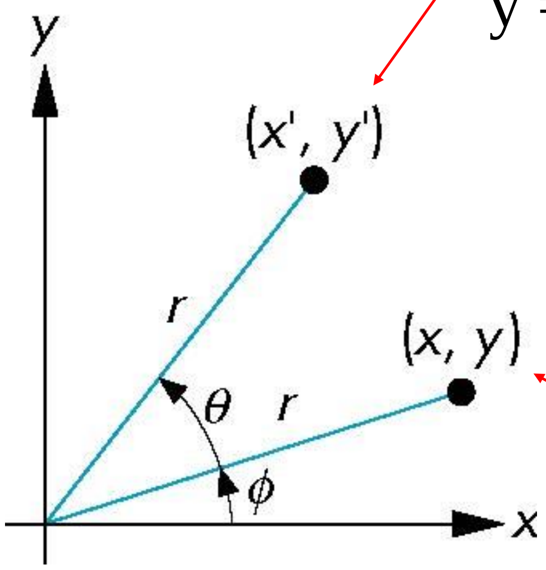
请自行动手乘一下。

图形学中大量采用这种形式，这种形式更容易实现，因为所有的仿射变换（旋转和平移）都可以用这种形式统一表示，矩阵乘法可以复合在一起

二维旋转（2-dimensional Rotation)

- 考虑绕原点旋转θ度
  - 半径保持不变，角度增加了θ

$x = r\cos(\phi + \theta)$
$y = r\sin(\phi + \theta)$

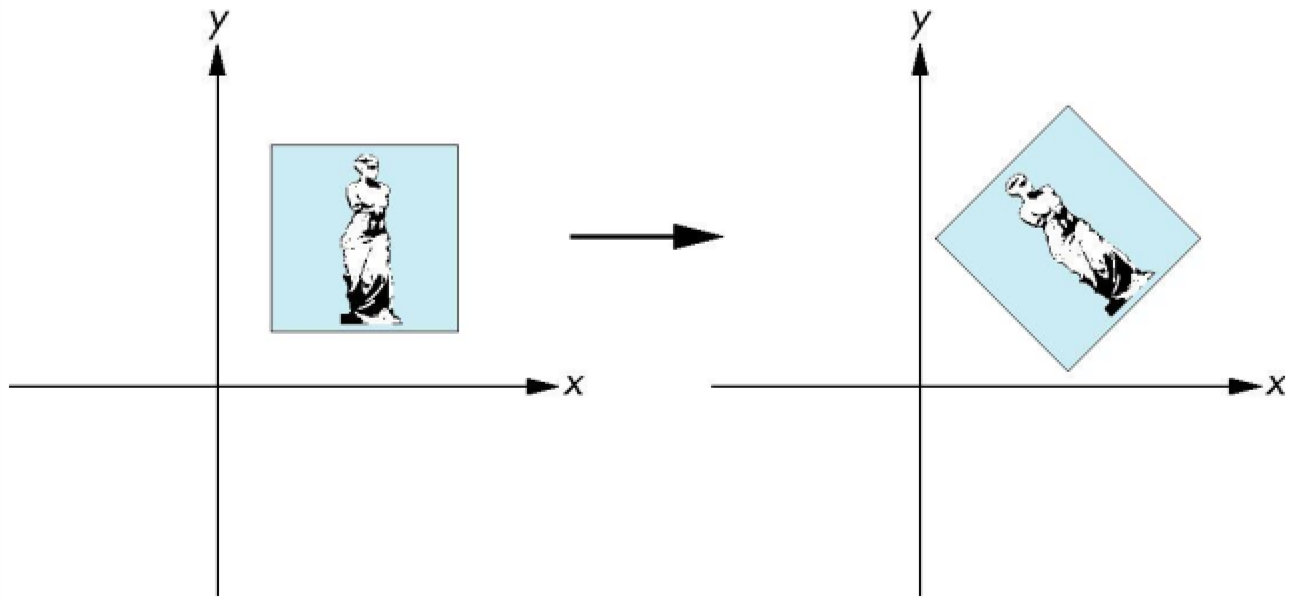


$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$

$x = r\cos\phi$
$y = r\sin\phi$

二维旋转

- 旋转轴：等效于三维空间绕z轴旋转
- 旋转角：（从z轴正方向看）逆时针方向为正向



绕z轴的旋转

- 在三维空间中绕z轴旋转，点的z坐标不变
  - 等价于在z=常数的平面上进行二维旋转

$$x' = x\cos\theta - y\sin\theta$$
$$y' = x\sin\theta + y\cos\theta$$
$$z' = z$$

  - 其齐次坐标表示为

$$\mathbf{p}' = \mathbf{R}_{\mathbf{z}}(\theta)\mathbf{p}$$

绕z轴的旋转矩阵

$$x' = x \cos\theta - y \sin\theta$$
$$y' = x \sin\theta + y \cos\theta$$
$$z' = z$$

$$\mathbf{R} = \mathbf{R}_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

绕X轴和绕Y轴的旋转矩阵

- 与绕z轴的旋转完全类似
  - 对于绕x轴的旋转，x坐标不变

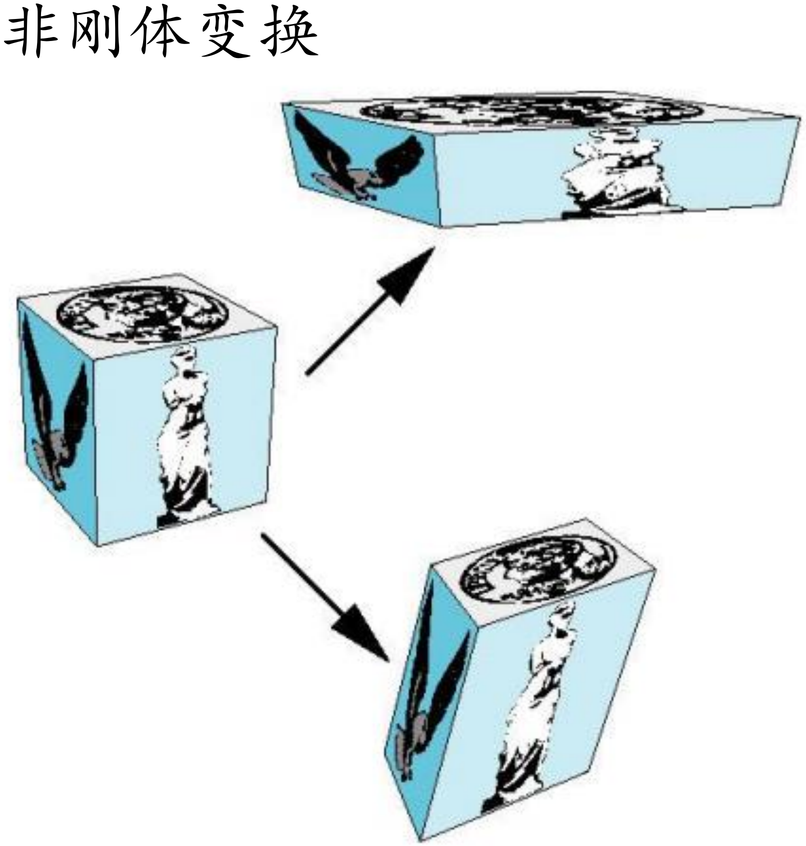
$$\mathbf{R} = \mathbf{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - 对于绕y轴的旋转，y坐标不变

$$\mathbf{R} = \mathbf{R}_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

刚体变换（Rigid Transformation)

- 旋转与平移是两种刚体变换
  - 这两种变换的复合只能改变对象的位置与定向
  - 保角度和长度
- 其它的仿射变换会改变对象的形状和体积



缩放（scaling）

- 沿每个坐标轴伸展或收缩(原点为不动点)

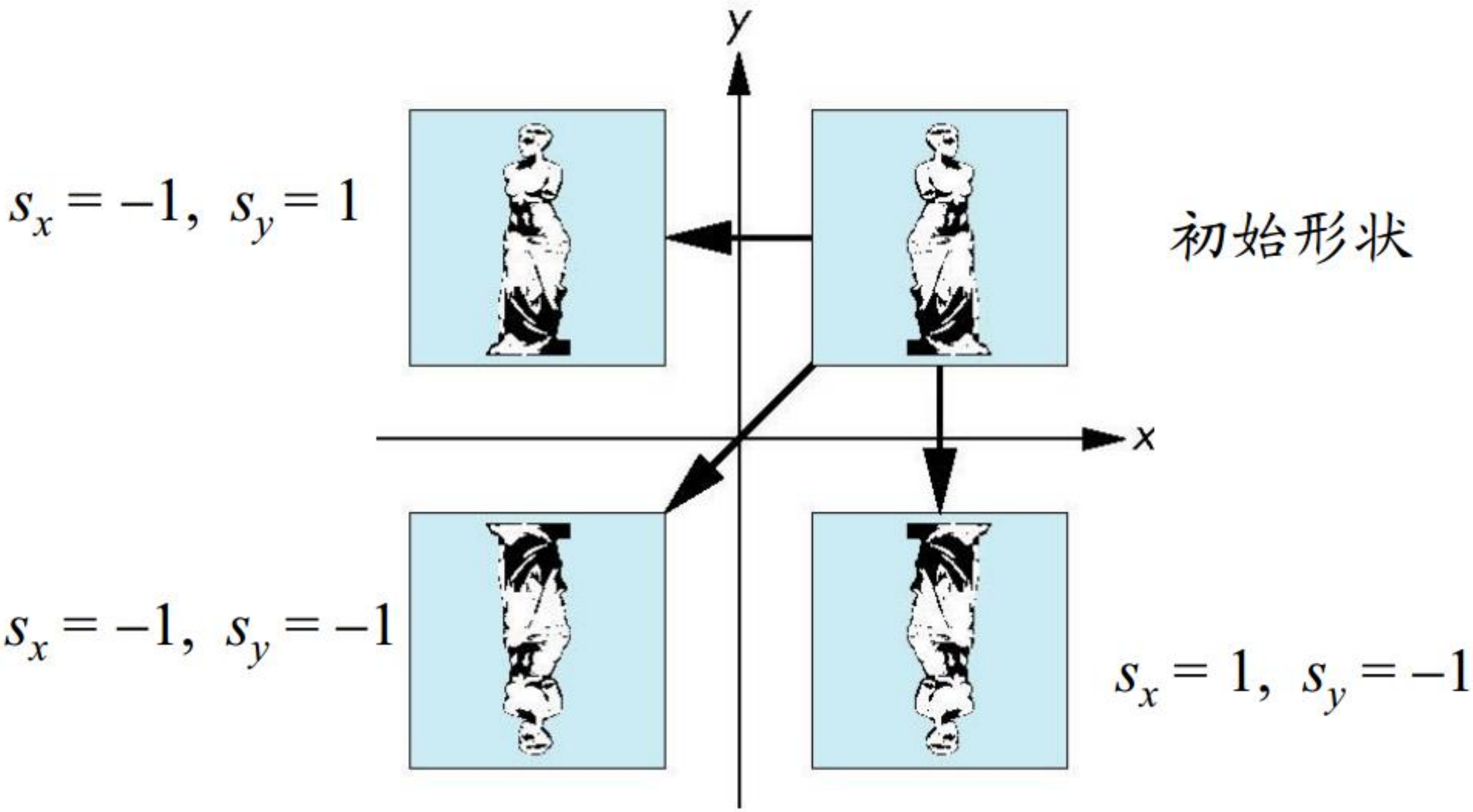
$$x' = s_x x$$
$$y' = s_y x$$
$$z' = s_z x$$

$$\mathbf{p}' = \mathbf{S}\mathbf{p}$$

$$\mathbf{S} = \mathbf{S}(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

反射（reflection)

- 特殊的缩放
  - 缩放系数为负数



几何变换的数学表示

- 变换的作用及绘制流水线中变换的位置
- 基本变换
  - 平移 Translation
  - 旋转 Rotation
  - 缩放 Scaling
- 复杂变换
  - 逆变换
  - 变换的复合
  - 特殊旋转

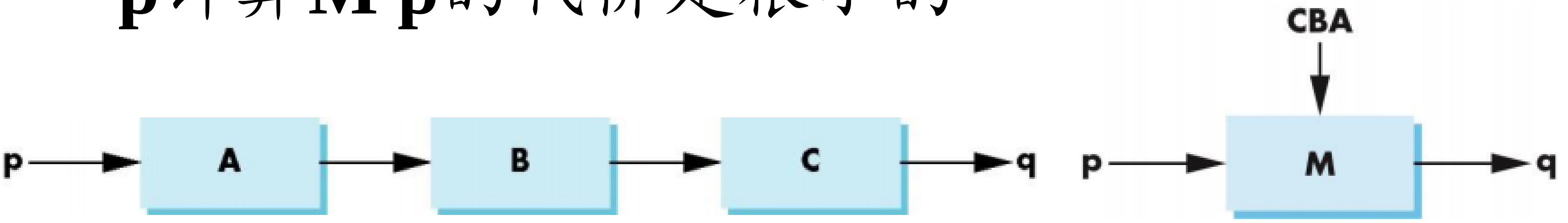
逆变换 (Inverse Transformation)

- 虽然可以直接计算矩阵的逆，但根据几何意义可以给出各种变换的逆
  - 平移: $\mathbf{T}^{-1}(d_x, d_y, d_z) = \mathbf{T}(-d_x, -d_y, -d_z)$
  - 旋转: $\mathbf{R}^{-1}(q) = \mathbf{R}(-q)$
    - 对所有旋转矩阵成立
    - 注意 $\cos(-q) = \cos(q)$, $\sin(-q)=-\sin(q)$

      $$\mathbf{R}^{-1}(q) = \mathbf{R}^{T}(q)$$
  - 缩放: $\mathbf{S}^{-1}(s_x, s_y, s_z) = \mathbf{S}(1/s_x, 1/s_y, 1/s_z)$

变换的复合 (Transformation Concatenation)

- 可以通过把旋转、平移与缩放矩阵相乘从而形成任意的仿射变换
- 由于对许多顶点应用同样的变换，因此构造矩阵**M = C B A**的代价相比于对许多顶点**p**计算**M p**的代价是很小的



- 难点在于如何根据应用程序的要求构造出满足要求的变换矩阵

变换的顺序(Order of Transformations)

- 注意在右边的矩阵是首先被应用的矩阵

- 从数学的角度来说，下述表示是等价的

  p’ = ABCp = A(B(Cp))

- 变换的顺序是不可交换的
  - 矩阵乘法不满足交换律

绕原点的一般旋转

- 绕过原点任一轴旋转θ角可以分解为绕*x*, *y*, z轴旋转的复合

$$\mathbf{R}(\theta) = \mathbf{R}_z(\theta_z)\,\mathbf{R}_y(\theta_y)\,\mathbf{R}_x(\theta_x)$$

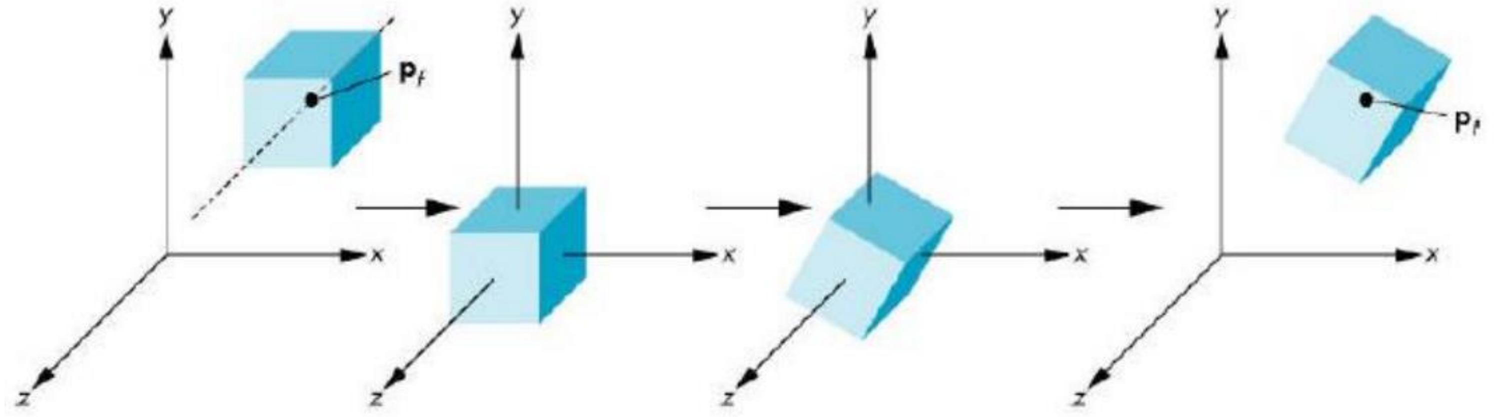
$\theta_x\ \theta_y\ \theta_z$ 被称为欧拉角（Euler angles）

- 注意：因为矩阵乘法不具有交换性，因此调换z,y,x的顺序将导致不同的旋转效果。调换顺序之后，如果为了得到原来的旋转效果，则旋转角度应相应改变。

不动点为$\mathbf{p}_f$的旋转

- 把不动点移到原点 $\mathbf{T}(-\mathrm{p}_f)$
- 旋转 $\mathbf{R}(\theta)$
- 把不动点移回到原来位置 $\mathbf{T}(\mathbf{p}_f)$

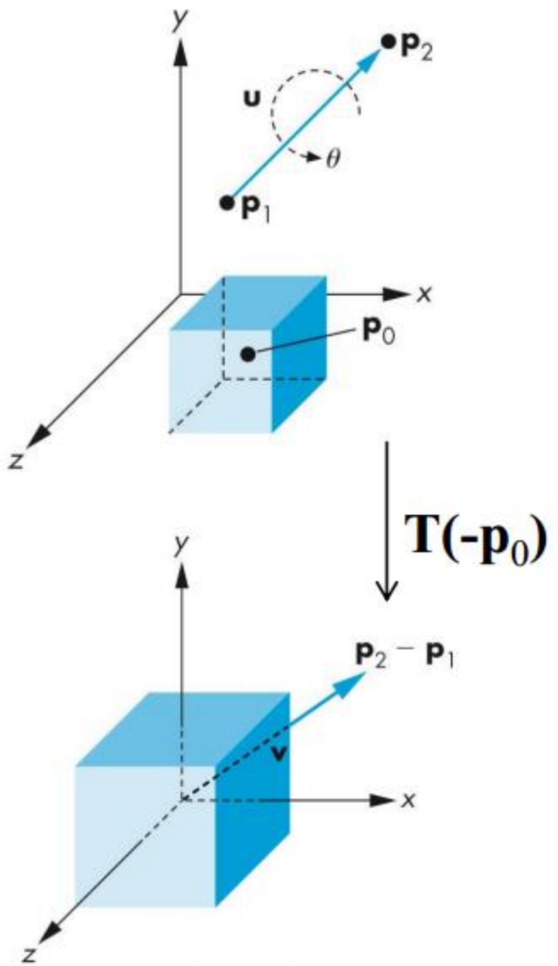
$$\mathbf{M} = \mathbf{T}(\mathbf{p}_f)\,\mathbf{R}(\theta)\,\mathbf{T}(-\mathbf{p}_f)$$



绕任意轴的旋转 (处理旋转中心)

- 不动点：立方体中心$\mathbf{p}_0$
- 旋转轴方向向量：

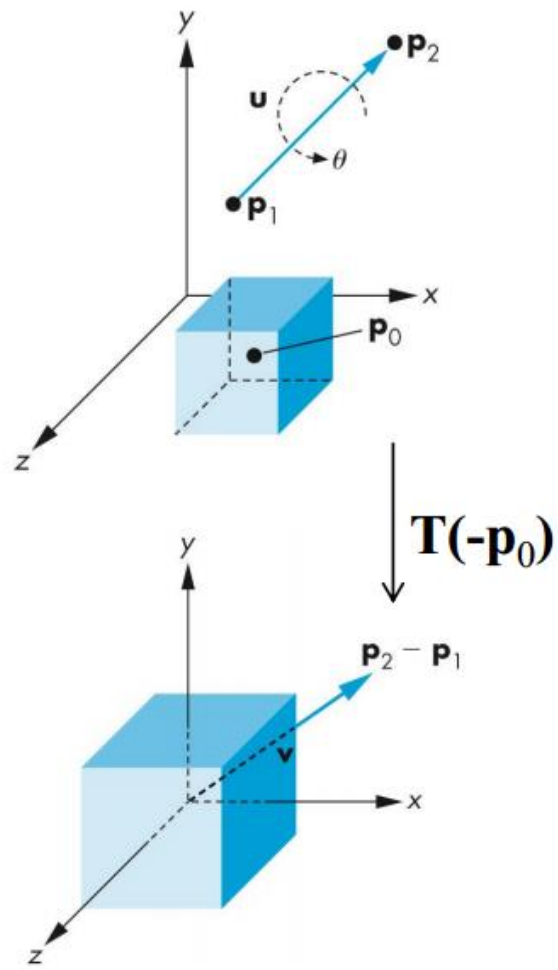
  $\mathbf{u} = \mathbf{p}2 - \mathbf{p}1$
- $\mathbf{T}(-\mathbf{p}_0)$平移不动点到原点



绕任意轴的旋转
(处理旋转)

- 策略：先经过两次旋转使旋转轴v与z轴对齐，然后绕z轴旋转角度θ

$$\mathbf{R} = \mathbf{R}_x(-\theta_x)\mathbf{R}_y(-\theta_y)\mathbf{R}_z(\theta)\,\mathbf{R}_y(\theta_y)\mathbf{R}_x(\theta_x)$$



所有矩阵串乘

$$\mathbf{M} = \mathbf{T}(\mathbf{p}_0)\, \mathbf{R}_x(-\theta_x)\, \mathbf{R}_y(-\theta_y)\, \mathbf{R}_z(\theta)\, \mathbf{R}_y(\theta_y)\, \mathbf{R}_x(\theta_x)\, \mathbf{T}(-\mathbf{p}_0)$$

- 请自行推导 (课本4.9.4)
  - 如何求θ_x和θ_y

本部分需要掌握

- 齐次坐标乘法
- 平移矩阵
- 缩放矩阵、反射矩阵
- 旋转中心在原点，分别绕x,y,z轴的旋转矩阵
- 旋转中心在原点，任意旋转的旋转矩阵
- 旋转中心不在原点，旋转轴任意的旋转矩阵（希望自己能推导）

变换及OpenGL实现

- 几何变换的数学表示
- OpenGL中的几何变换
- 专题：虚拟轨迹球

OpenGL中的几何变换

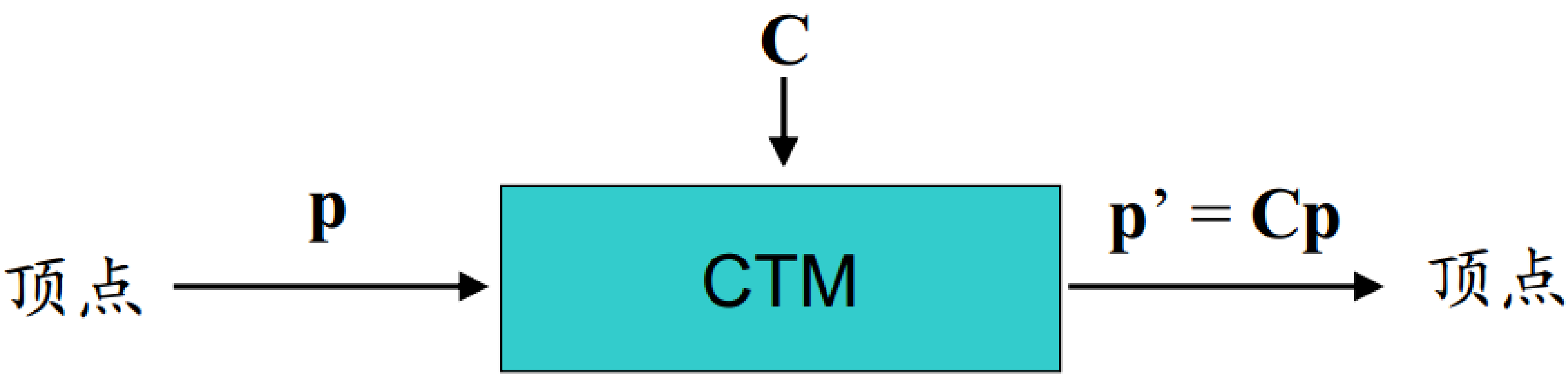
- OpenGL中的矩阵
- 平移
- 旋转
- 缩放

OpenGL中的矩阵

- 在OpenGL中，矩阵是状态的一部分
- 有多种类型
  - 模型视图 (GL_MODELVIEW)
  - 投影 (GL_PROJECTION)
  - 纹理 (GL_TEXTURE) (不讲）
  - 颜色 (GL_COLOR) (不讲）
- 矩阵由一组共同的函数来设置或修改
- 选择所操作的矩阵
  - glMatrixMode(GL_MODELVIEW);
  - glMatrixMode(GL_PROJECTION);

当前变换矩阵 (Current Transformation Matrix)

- 从概念上说， 当前变换矩阵(CTM)就是一个4x4阶的齐次坐标矩阵，它是状态的一部分，被应用到经过流水线中的所有顶
- CTM是在应用程序中定义的，并被加载到变换单元中



CTM操作

- CTM可以被改变，改变的方法是加载一个新的CTM或者右乘一个矩阵
  - 加载单位阵： $C \leftarrow I$
  - 加载任意矩阵： $C \leftarrow M$
  - 加载一个平移矩阵： $C \leftarrow T$
  - 加载一个旋转矩阵： $C \leftarrow R$
  - 加载一个缩放矩阵： $C \leftarrow S$
  - 右乘任意矩阵： $C \leftarrow CM$
  - 右乘一个平移矩阵： $C \leftarrow CT$
  - 右乘一个旋转矩阵： $C \leftarrow CR$
  - 右乘一个缩放矩阵： $C \leftarrow CS$

绕固定点的旋转

- 从单位阵开始： $\mathbf{C} \leftarrow \mathbf{I}$
  把固定点移到原点： $\mathbf{C} \leftarrow \mathbf{CT}$
  旋转： $\mathbf{C} \leftarrow \mathbf{CR}$
  把固定点移回到原处： $\mathbf{C} \leftarrow \mathbf{CT}^{-1}$
  结果： $\mathbf{C} = \mathbf{TR\,T}^{-1}$ 是逆向的

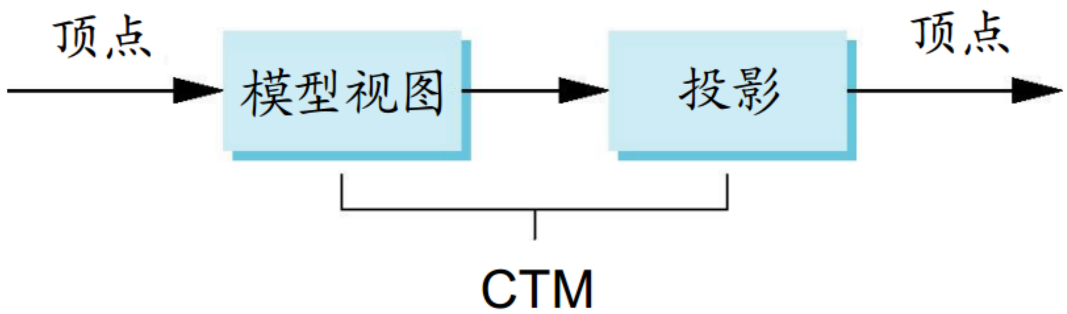
- 这是做矩阵右乘的结果
- 注意：在程序中最后指定的变换最先被应用

倒转次序

- 我们想要得到 $\mathbf{C} = \mathbf{T}^{-1}\,\mathbf{R}\,\mathbf{T}$
  所以必须按下面的次序进行运算
  $\mathbf{C} \leftarrow \mathbf{I}$
  $\mathbf{C} \leftarrow \mathbf{C}\,\mathbf{T}^{-1}$
  $\mathbf{C} \leftarrow \mathbf{CR}$
  $\mathbf{C} \leftarrow \mathbf{CT}$
- 每个运算对应于程序中的一个函数调用
- 注意： 在程序中最后指定的变换最先被应用

OpenGL中的CTM

- 在OpenGL的流水线中有一个模型视图矩阵和一个投影矩阵，这两个矩阵复合在一起构成CTM



- 可以通过首先设置正确的矩阵模式处理每个矩阵
  - 选择所操作的矩阵
    - glMatrixMode(GL_MODELVIEW);
    - glMatrixMode(GL_PROJECTION);

旋转、平移和缩放

- 加载单位阵：
  - glLoadIdentity()
- 右乘变换矩阵：
  - glRotatef(theta, vx, vy, vz)
    - theta以角度为单位，(vx,vy,vz)定义旋转轴
  - glTranslatef(dx, dy, dz)
  - glScalef(sx, sy, sz)
    - 每个函数的参数还可以是d(double)类型

示例

- 固定点为(1.0, 2.0, 3.0), 绕z轴旋转30度

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
glTranslatef(1.0, 2.0, 3.0);
glRotatef(30.0, 0.0, 0.0, 1.0);
glTranslatef(-1.0, -2.0, -3.0);
```

- 记住在程序中最后指定的矩阵最先被应用

任意矩阵

- 可以加载应用程序中定义的矩阵，或者使之与CTM相乘
  glLoadMatrixf(m)
  glMultMatrixf(m)
- 矩阵m是有16个元素的一维数组，按列优先排列定义了4x4矩阵
- 在glMultMatrixf(m)中，m右乘当前矩阵，OpenGL没提供左乘

任意矩阵

- 16维向量m=(m_0 , m_1 ,…,m_{15})指定的列主序矩阵为

$$\mathbf{M} = \begin{bmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{bmatrix}$$

- C语言定义的矩阵m[4][4]是行主序的，元素m[i][j]相当于OpenGL变换矩阵的i列j行元素。为避免行列混淆，声明矩阵为m[16]。

矩阵堆栈

- 许多情况中需要保存变换矩阵，待稍后再用
- OpenGL为每种类型的矩阵维持一个堆栈
- 应用下述函数处理矩阵堆栈（也是由glMatrixMode设置矩阵类型）

  glPushMatrix()

  glPopMatrix()

获取当前矩阵

- 可以利用查询函数读入矩阵（以及其它部分的状态）
  - glGetIntegerv, glGetFloatv, glGetBooleanv, glGetDoublev

- 例如，对于模型视图矩阵：

```
double m[16];
glGetDoublev(GL_MODELVIEW_MATRIX, m);
```

变换的应用

- 例如：应用空闲函数旋转立方体，鼠标函数改变旋转的方向
- 立方体旋转程序
  - 立方体中心在原点
  - 各方向与坐标轴平行

main函数

- int main(int argc, char **argv)
```
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB
| GLUT_DEPTH);
glutInitWindowSize(500, 500);
glutCreateWindow("colorcube");
glutReshapeFunc(reshape);
glutDisplayFunc(display);
glutIdleFunc(spinCube);
glutMouseFunc(mouse);
glEnable(GL_DEPTH_TEST);
glutMainLoop();
}
```

空闲和鼠标回调函数

```
void spinCube()
{
theta[axis] += 2.0;
if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
glutPostRedisplay();
}
void mouse(int btn, int state, int x, int y)
{
if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN)
axis = 0;
if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)
axis = 1;
if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN)
axis = 2;
}
```

显示回调函数

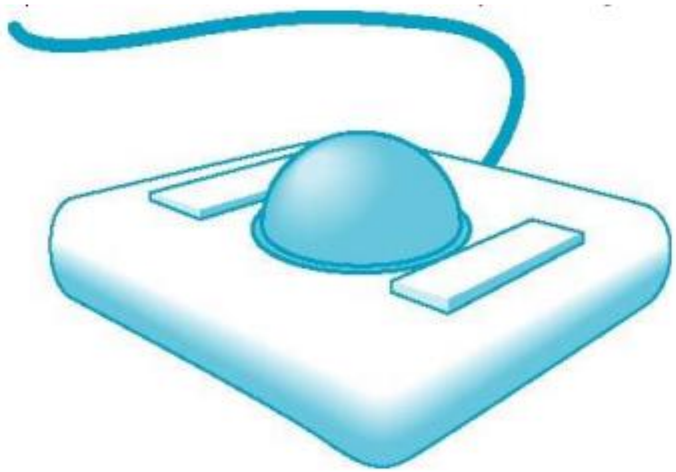
```
void display()
{
    glClear(GL_COLOR_BUFFER_BIT |
    GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
    colorcube();
    glutSwapBuffers();
}
```

变换及OpenGL实现

- 几何变换的数学表示
- OpenGL中的几何变换
- 专题：虚拟轨迹球

物理轨迹球

- 轨迹球是一个底朝上的鼠标



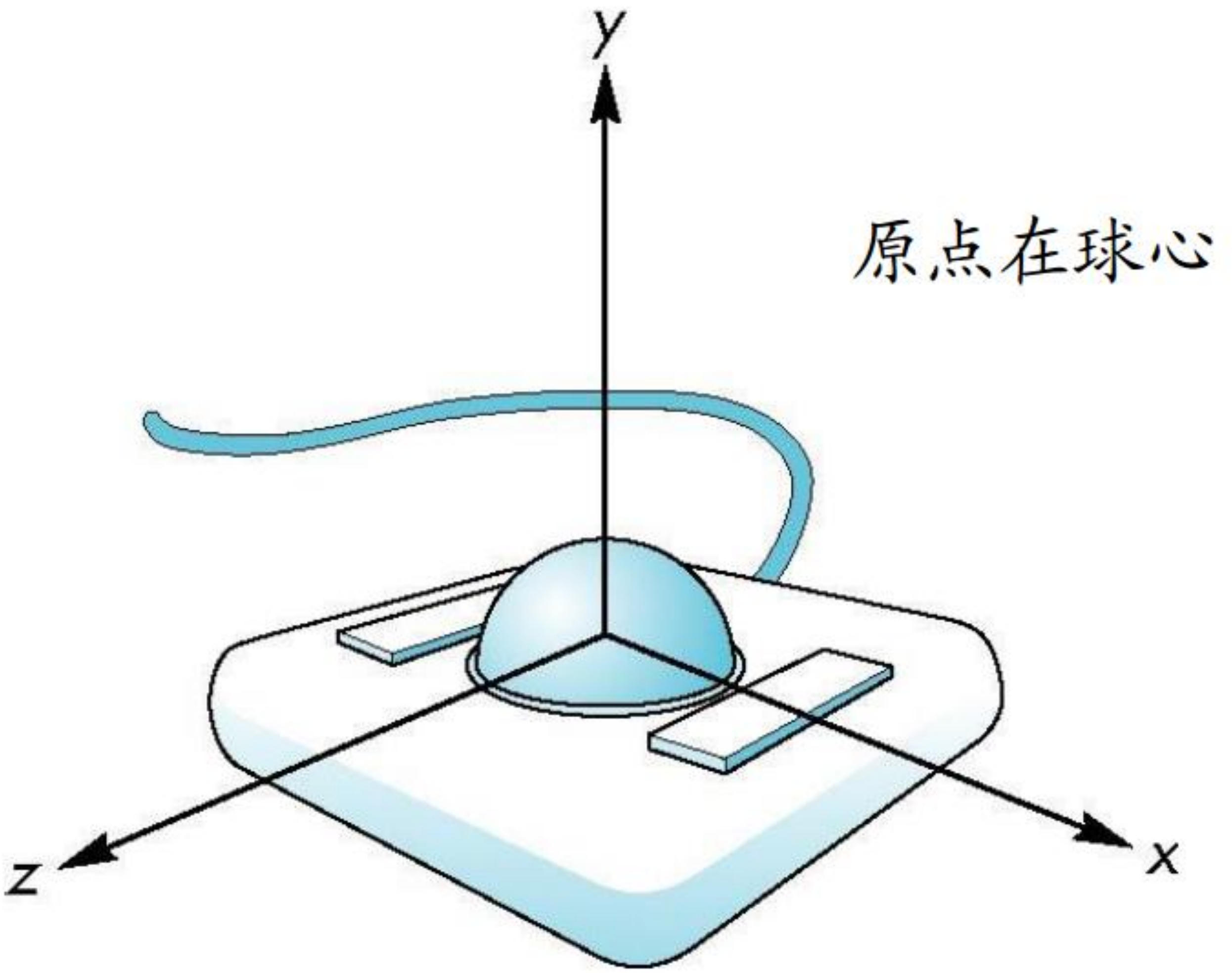
- 如果球和滚轴之间有小的摩擦力，我们可以推动球使其保持滚动产生连续的变化
- 通过滚动球，控制物体旋转

鼠标模拟虚拟轨迹球

- 希望利用鼠标模拟轨迹球，鼠标左键按下拖动实现物体旋转

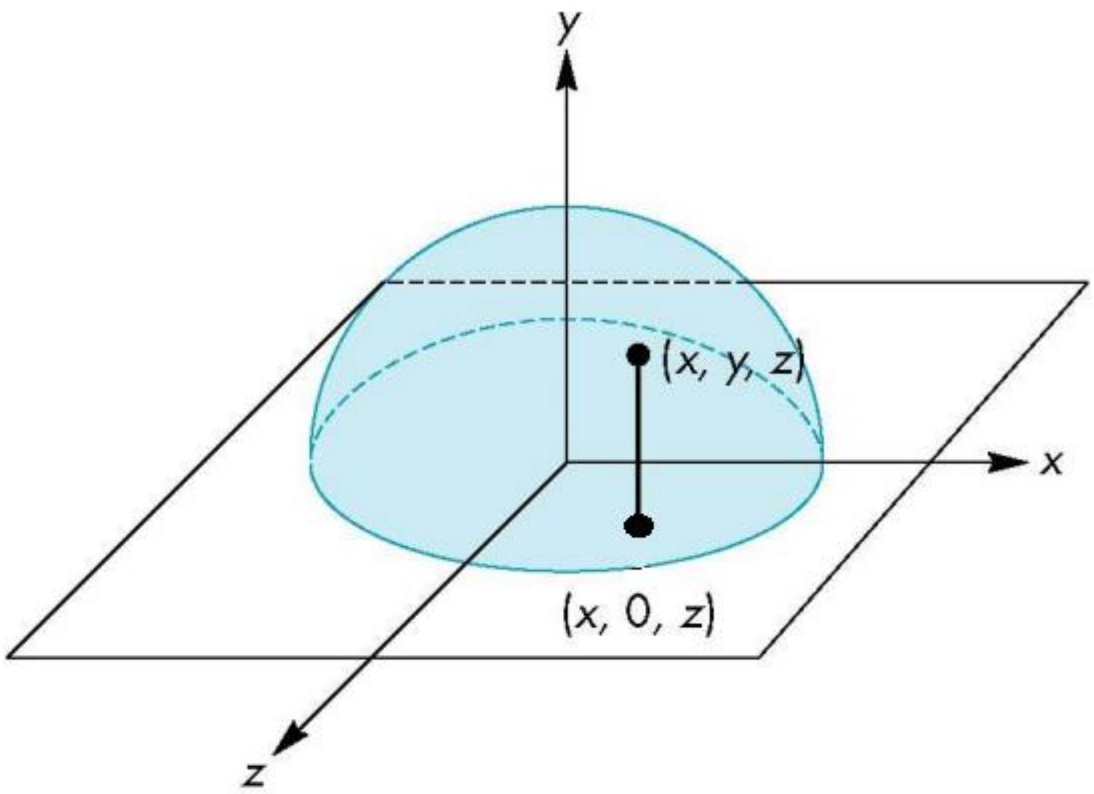
- 分两步解决：
  - 把跟踪球位置映射到鼠标位置
  - 根据鼠标位置计算出旋转角度

轨迹球标架



投影轨迹球位置

- 把轨迹球面上一点正交投影到y=0平面（鼠标垫）



逆投影

- 因为投影平面和上半球面都是二维曲面，我们可以逆向投影

- y=0平面上一点(x,0,z) 对应到半球面上的点(x,y,z)，这里

$$y = \sqrt{r^2 - x^2 - z^2} \qquad \text{if } r \geq |x| \geq 0,\ r \geq |z| \geq 0$$

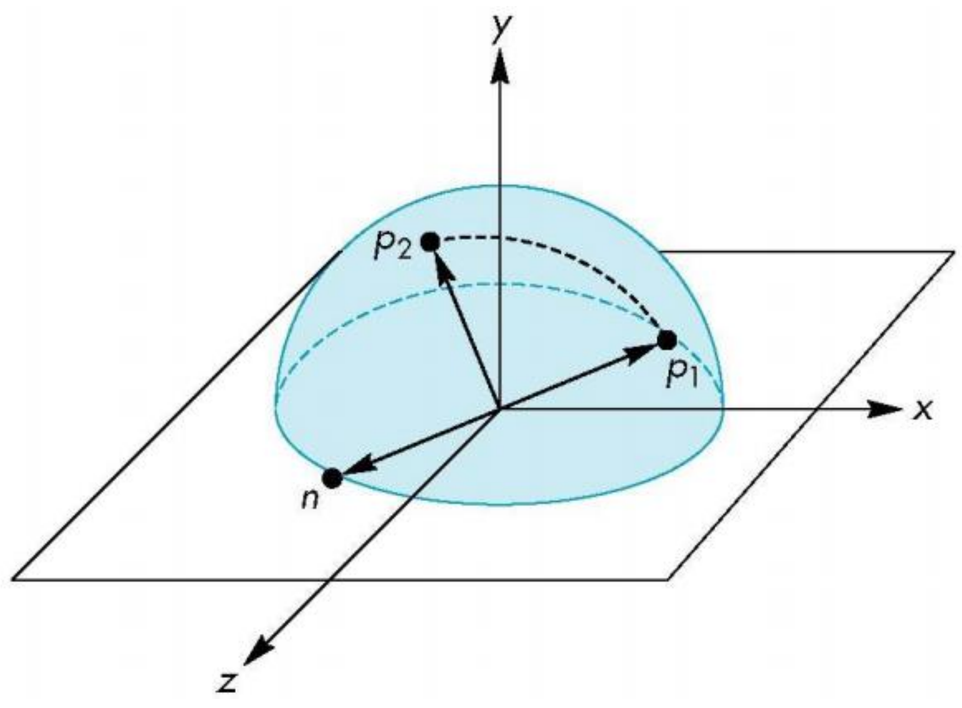
计算旋转

- 假设我们从鼠标得到两个点
- 可以把它们投影到半球面上的点**p**1和**p**2
- 这两个点确定了球面上的一个大圆
- 找到恰当的旋转轴和旋转角度就可以把**p**1旋转到**p**2

利用叉积确定旋转轴

- 旋转轴就是原点和**p**1， **p**2这两个点所确定平面的法向

  n = **p**1 x **p**2



旋转角

- $\mathbf{p}1$和 $\mathbf{p}2$ 的夹角为

$$\sin\theta = \frac{|\mathbf{n}|}{|\mathbf{p}_1||\mathbf{p}_2|}$$

- 如果我们缓慢移动鼠标或者频繁采样鼠标位置，那么 θ 将很小，我们可用近似计算公式

$$\sin\theta \approx \theta$$

问题

- 如果鼠标点的点在球之外呢？