CPT205W03_变换管线和几何变换

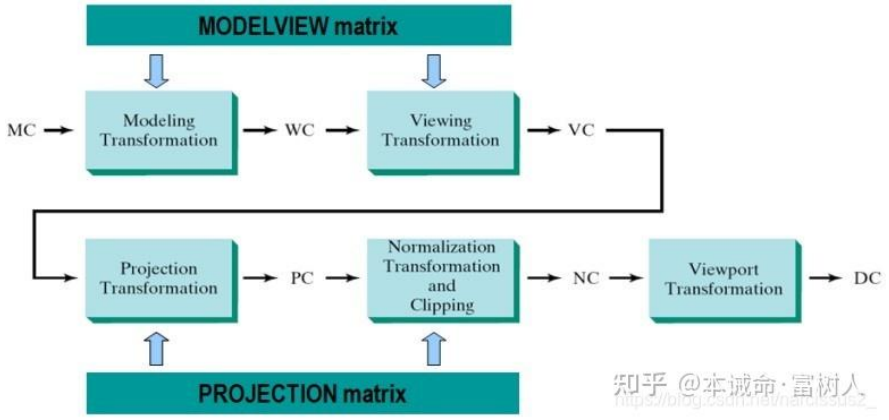
Transformation Pipeline

Pipeline：sequence of steps

变换管线是必须先应用于对象的一系列变换/转换（the series of transformations(alterations)），然后才能在屏幕上正确地显示对象。

变换可认为由数个处理阶段组成，省略其中某些阶段会导致显示的不正确，如若跳过projection投影阶段，物体将失去透视，看起来没有深度。

一旦对象通过管线，就可以显示为线框（wire-frame item）或实体（solid item）。



MC：模型坐标系（model coordinate）

- **Modelling transformation：模型变换**

  将对象放入虚拟世界中设定的位置

WC：世界坐标系

- **Viewing Transformation：视图变换**

  我们只关心摄像机可见的视野；从虚拟世界中的不同位置（摄像机位置）查看对象，得到物体与摄像机的相对位置

VC：视界坐标系

- **Projection Transformation:投影变换**

  查看对象的深度；在视图变换之后，我们得到了所有可视范围内的物体相对摄像机的相对位置坐标(x,y,z)，根据需求进行正交投影或透视投影，投影至标准二维平面([-1,1]^2)上（这里的深度z并没有丢掉，为了之后的遮挡关系检测）

PC：投影坐标系

- **Normalization Transformation and Clipping: 正则化变换和剪裁**

  （裁剪坐标系-》透视分割-》规范化设备坐标系（NDC space））

  临时将“感兴趣窗口”定义的体积加上前后剪切平面映射到一个单位立方体中。在这种情况下，某些其他操作更容易执行。

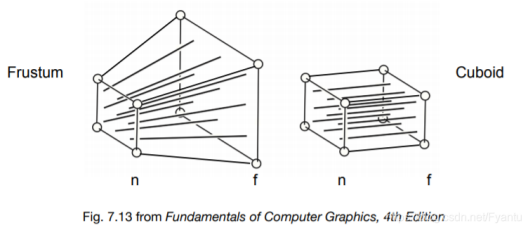


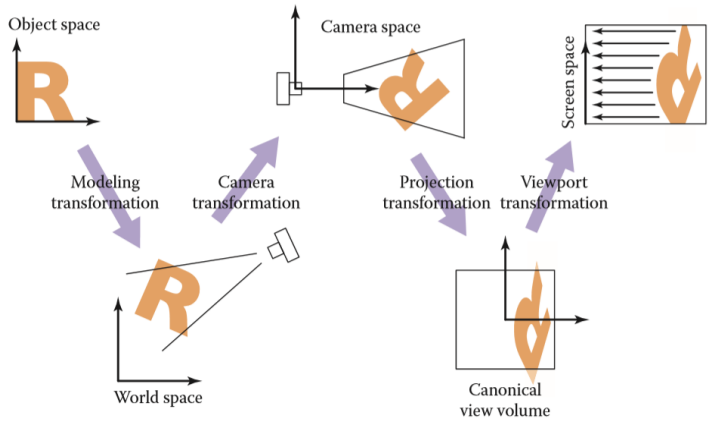
Fig. 7.13 from *Fundamentals of Computer Graphics, 4th Edition*

NC:正则化坐标

- **Viewport Transformation: 视口转换**

  将处于标准平面映射到屏幕分辨率范围之内，即[-1,1]^2[0,width]*[0,height]，其中width和height指屏幕分辨率大小

DC: 设备坐标

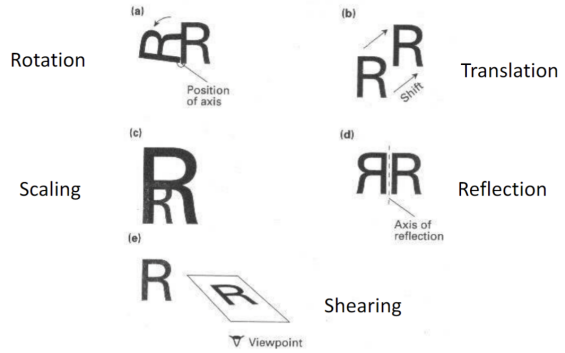


计算机图形学二：视图变换(坐标系转化，正交投影，透视投影，视口变换) - 孙小磊的文章 - 知乎
https://zhuanlan.zhihu.com/p/144329075

当对象从文件中加载并准备好进行处理时开始。当对象准备好显示在计算机屏幕上时完成。

几何变换的类型

Types of geometric transformation



https://blog.csdn.net/qq_40595787/article/details/121555358

2D平移 translation

若$P(x,y)$是原位置，$T<t_x,t_y>$是平移的方向，$P'(x',y')$是变换后的位置：

$$P'=P+T\rightarrow\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}t_x\\t_y\end{bmatrix}$$

等效于使用齐次矩阵（homogeneous co-ordinates）：

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

（这样所有的几何变换都可用矩阵相乘表示，而不是相加或相乘）

2D旋转 rotation

- 绕原点旋转：

  $$P'=R\cdot P\rightarrow\begin{bmatrix}x'\\y'\end{bmatrix}=\begin{bmatrix}cos\theta&-sin\theta\\sin\theta&cos\theta\end{bmatrix}\cdot\begin{bmatrix}x\\y\end{bmatrix}$$

  类似之前PCA旋转的想法。

  等效于使用齐次矩阵：

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

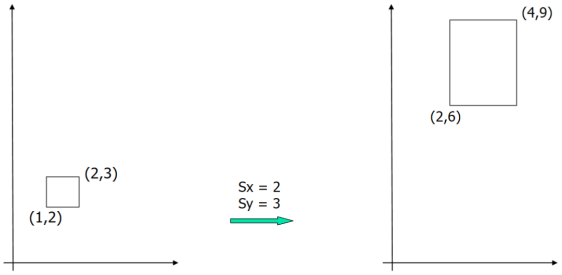
- 绕定点旋转：

  将定点（与对象）移至原点，旋转对象，将定点移回原位。

  $$M=T(p_f)R(\theta)T(-p_f)$$

2D缩放 scaling

$$P' = S \cdot P \rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$



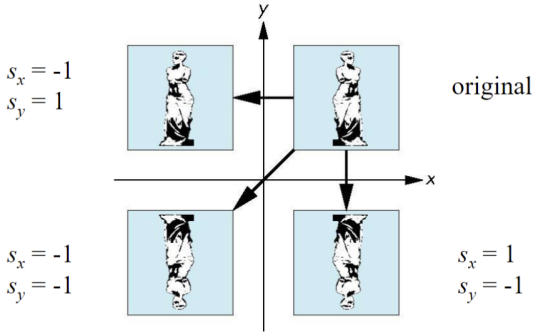
注意，这种方式的缩放，对象的大小和位置都产生变化（按原点缩放）

等效于齐次矩阵：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

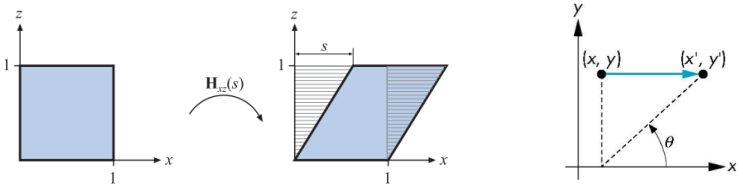
2D反射 reflection

一种缩放的特殊情况，缩放因子（scale factors）为负。



2D切变 shearing

可以用于游戏中扭曲整个场景，以创建迷幻效果或以其他方式扭曲模型的外观。



$$\begin{cases} x' = x + ycot\theta \\ y' = y \\ z' = z \end{cases} \rightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 1 & cot\theta \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix}$$

等效于齐次矩阵：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & cot\theta & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

2D复合变换

使用齐次矩阵（homogeneous co-ordinates）：

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} rs_{xx} & rs_{xy} & trs_x \\ rs_{yx} & rs_{yy} & trs_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

其中rs是旋转-缩放项，包括旋转角度和缩放因子；trs是平移项，包含平移距离，pivot-point，fixed-point co-ordinates，旋转角度，缩放参数。

3D变换

平移&缩放

平移和缩放操作可直接从2D方法扩展

旋转

2D的旋转是关于任意点旋转，但3D的旋转是关于任意轴旋转。这并不简单，我们将其拆解为分别关于xyz轴做3次旋转：

$R(q) = R_z(q_z)R_y(q_y)R_x(q_x)$其中每个$q_n$都称为欧拉角（Eular angle）

*注意，交换旋转运算的顺序会导致不同的旋转结果

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ (About z-axis)

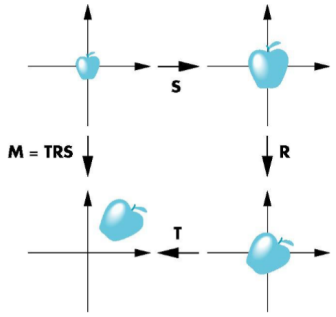
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ (About y-axis)

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$ (About x-axis)

> 平移、缩放、旋转、反射和切变都是仿射变换，因为变换后的点 P'（x'，y'，z'） 是原始点 P（x，y，z） 的线性组合。

实例化 Instancing

在建模中，我们将实例转换应用于顶点来缩放（scale），转向（orient），移动（locate）。



M=TRS

OpenGL矩阵

在 OpenGL 中，矩阵是**状态(state)**的一部分。

https://docs.gl/gl3/glFrustum 见**Matrix State栏目**

当前变换矩阵（CTM）

从概念上讲，有一个4x4齐次坐标矩阵，即当前变换矩阵（CTM），它是状态的一部分，应用于沿管线传递的所有顶点。
CTM在用户程序中定义并加载到转换单元（transformation unit）中。
CTM可以通过加载新的CTM（例如identity matrix）或右乘（postmultiplication）来改变。
OpenGL在管道中有一个模型视图和一个投影矩阵，它们连接在一起形成CTM。
CTM可以通过首先设置正确的矩阵模式来操纵每个模式。

CTM操作

- 切换矩阵模式 glMatrixMode — specify which matrix is the current matrix

```
1 void glMatrixMode(GLenummode);
```

指定几何查看、投影、纹理或颜色转换的当前矩阵

mode 可以采用以下四个值之一：

- `GL_MODELVIEW` Applies subsequent matrix operations to the modelview matrix stack.

将后续矩阵运算应用于 modelview 矩阵堆栈。

- `GL_PROJECTION` Applies subsequent matrix operations to the projection matrix stack.

  将后续矩阵运算应用于投影矩阵堆栈。

- `GL_TEXTURE` Applies subsequent matrix operations to the texture matrix stack.

  将后续矩阵操作应用于纹理矩阵堆栈。

- `GL_COLOR` Applies subsequent matrix operations to the color matrix stack.

  将后续矩阵操作应用于颜色矩阵堆栈。

- 加载4x4单位矩阵 glLoadIdentity — replace the current matrix with the identity matrix

```
void glLoadIdentity(void);
```

- 将当前矩阵后乘以指定的矩阵 glMultMatrix — multiply the current matrix with the specified matrix

```
void glMultMatrixf(const GLfloat *m);
```

c=c·m

- 为栅格操作（raster operations）指定2D缩放参数 glPixelZoom — specify the pixel zoom factors

```
void glPixelZoom(GLfloat xfactor,
                 GLfloat yfactor);
```

xfactor, yfactor
Specify the x and y zoom factors for pixel write operations.

任意矩阵 Arbitrary Matrices

法1：Load函数加载

void glLoadMatrixf(const GLfloat *m);

> 注：通常有单精度f和双精度d版函数(glLoadMatrixd)，笔记中默认仅写单精度版。

指定指向 16 个连续值的指针，这些值用作 4×4 列主序(column-major)矩阵的元素。

CTM不是一个矩阵，而是一个矩阵堆栈（matrix stack），current在栈顶。

- 当我们想保存转换矩阵以备后用时：

  遍历层次数据结构。

  避免在执行显示列表时更改状态

3.1版本之前的OpenGL为每种类型的矩阵维护堆栈（glPushMatrix()，glPopMatrix()），但现在只有一个CTM。

法2：使用查询函数（query function）

- glGetIntegerv
- glGetFloatv
- glGetBooleanv
- glGetDoublev
- glIsEnabled

```
// 切换矩阵
double m[16];
glGetFloatv(GL_MODELVIEW, m);
```

OpenGL的变换函数

- 平移 glTranslate — multiply the current matrix by a translation matrix

```
void glTranslatef(  GLfloat x,
                    GLfloat y,
                    GLfloat z);
```

x, y, z

Specify the x, y, and z coordinates of a translation vector.

- 旋转 glRotate — multiply the current matrix by a rotation matrix

```
void glRotatef( GLfloat angle,
                GLfloat x,
                GLfloat y,
                GLfloat z);
```

angle
Specifies the angle of rotation, in degrees.
x, y, z
Specify the x, y, and z coordinates of a vector, respectively.

- 缩放 glScale — multiply the current matrix by a general scaling matrix

```
void glScalef(  GLfloat x,
                GLfloat y,
                GLfloat z);
```

x, y, z
Specify scale factors along the x, y, and z axes, respectively.

示例：对点(1,2,3)按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);
```