三维变换中的矩阵

知乎用户7P1Bw5
赞同 19
19 人赞同了该文章

扩充三维世界

为了给诸多变换的实现扫清障碍，首先扩充三维世界的第四个分量，通常合并表示为*xyzw*

Point

$$(v_x \quad v_y \quad v_z \quad 1)^T$$

Vector

$$(v_x \quad v_y \quad v_z \quad 0)^T$$

点和向量的*w*分量

我们知道点可以被平移而向量的平移并不改变其数学含义，*w*分量为0、1时正体现了这样的区别。

在后文中，我们还将看到*w*分量的其他意义。

矩阵变换

矩阵变换大致有平移、缩放、旋转、投影几种。

要理解这些矩阵，首先我要告诉你的是：这一系列矩阵的根本来源是构造，先有需求，再有结果。请你暂时摒弃证明数学定理时的思维，我们不是要证明这些矩阵可以通过某种方式推导出来即其正确性，而是要构造出一种矩阵来解决需求。

平移矩阵

$$\mathbf{T}(\mathbf{t}) = \mathbf{T}(t_x, t_y, t_z) = \begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

顾名思义，平移矩阵的功能是平移一个点。平移简言之即给原坐标的各分量分别加一个偏移量，暂且用

$$P = (x \quad y \quad z \quad 1)^T$$

表示待变换的点，后文中若无说明也使用这个点。

需求已经明确了：将P点平移到

$$(p_x + t_x \quad p_y + t_y \quad p_z + t_z \quad 1)^T$$

即有以下等式：

$$\begin{aligned} T_{(0,0)} \cdot P_x + T_{(0,1)} \cdot P_y + T_{(0,2)} \cdot P_z + T_{(0,3)} &= P_x + t_x \\ T_{(1,0)} \cdot P_x + T_{(1,1)} \cdot P_y + T_{(1,2)} \cdot P_z + T_{(1,3)} &= P_y + t_y \\ T_{(2,0)} \cdot P_x + T_{(2,1)} \cdot P_y + T_{(2,2)} \cdot P_z + T_{(2,3)} &= P_z + t_z \\ T_{(3,0)} \cdot P_x + T_{(3,1)} \cdot P_y + T_{(3,2)} \cdot P_z + T_{(3,3)} &= 1 \end{aligned}$$

其中T(0, 1)表示平移矩阵第0行第1列的值。

每一个方程有4个未知量，显然解不唯一，但是我们可以找出最简单的那个，所以就有了最初列出的平移矩阵。

我们来看平移过程的数学表示

$$\begin{pmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_x \\ p_y \\ p_z \\ 1 \end{pmatrix}$$

根据矩阵运算法则，很轻易地可以得到预期结果。

平移矩阵逆矩阵的意义是反向平移，这个可以通过计算其逆矩阵得出：

$$T^{-1}(t) = T(-t) = \begin{pmatrix} 1 & 0 & 0 & -t_x \\ 0 & 1 & 0 & -t_y \\ 0 & 0 & 1 & -t_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

缩放矩阵

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

有了先前的推导过程，缩放矩阵的来历更为直观，此处不再赘述。

通过计算得到缩放矩阵的逆矩阵

$$S^{-1}(s) = S(1/s_x, 1/s_y, 1/s_z) = \begin{pmatrix} 1/s_x & 0 & 0 & 0 \\ 0 & 1/s_y & 0 & 0 \\ 0 & 0 & 1/s_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

旋转矩阵

旋转矩阵比上述两种要复杂一些，最终目的地是三维旋转矩阵，但是出发点是二维旋转矩阵。

二维旋转

P点的参数方程

$$\begin{cases} x = r\cos\theta \\ y = r\sin\theta \end{cases}$$

将P点按逆时针旋转Φ后P点坐标表示为

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

将上述过程表示以矩阵，有如下等式

$$\begin{pmatrix} ? & ? \\ ? & ? \end{pmatrix}\begin{pmatrix} rcos\theta \\ rsin\theta \end{pmatrix} = \begin{pmatrix} r(cos\theta cos\emptyset - sin\theta sin\emptyset) \\ r(sin\theta cos\emptyset + cos\theta sin\emptyset) \end{pmatrix}$$

从此得出二维的旋转矩阵

$$\begin{pmatrix} cos\emptyset & -sin\emptyset \\ sin\emptyset & cos\emptyset \end{pmatrix}$$

三维旋转

首先把三维旋转与先前推导的二维旋转联系起来，想两个问题：二维旋转是怎样推导的？现在还能不能这样推导？

三维旋转会保留坐标的其中一个分量不变。例如绕y轴旋转，则y坐标不变。（使用右手系）

看到这里，我们挑个好欺负的先来两下：在平面直角坐标系中将一个点绕z轴旋转Φ。

相比于二维旋转，x轴还是x轴，y轴也仍然是y轴，只是需要扩充一些数据来满足新的需求即不改变其z值。所以构造出以下矩阵

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

这个矩阵的推导可以说是水到渠成，那么能不能用类似的思维来推导绕另外两轴旋转的矩阵呢？

以绕x轴旋转为例。根据右手定则，绕x轴旋转时三维坐标系中的y轴等价于二维坐标系的x轴，而三维坐标系中的z轴等价于二维坐标系中的y轴，所以有以下结果

$$R_x(\emptyset) = \begin{pmatrix} \square & \square & \square & \square \\ \square & cos\emptyset & -sin\emptyset & \square \\ \square & sin\emptyset & cos\emptyset & \square \\ \square & \square & \square & \square \end{pmatrix}$$

再结合不改变x坐标等要求有

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

同理有最后一个旋转矩阵

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

虽然旋转矩阵的推导和形式比平移矩阵或缩放矩阵更为复杂，但其逆矩阵比先前两者都要简单。

由于这三个旋转矩阵都是正交矩阵，所以其逆矩阵等于其转置矩阵，这在计算上节省了不少性能。旋转矩阵逆矩阵的几何意义也非常明确：反向旋转。

$$R_i{}^{-1}(\emptyset) = R_i(-\emptyset) = R_i{}^T(\emptyset)$$

不论是二维或三维，上述矩阵的应用前提是目标单位处于坐标系原点，对于非原点坐标，需要先采用其他变换如平移将其变换至原点，旋转之后再复原。

绕任意轴的旋转矩阵

旋转轴是用三维向量来表示的，我们要做的是以给出的旋转轴为基础建立一个新的三维坐标系，旋转轴即其轴之一。

现在已经有一轴，还需要再找出一轴，第三轴可由叉乘得出。

第二轴的寻找是有数学规律在其中的：将旋转轴绝对值最小的分量置为0，再交换剩下两分量，并将两分量中的第一个置负。

$$
\begin{aligned}
\bar{\mathbf{s}} &= \begin{cases} (0, -r_z, r_y), & \text{if } |r_x| < |r_y| \text{ and } |r_x| < |r_z|, \\ (-r_z, 0, r_x), & \text{if } |r_y| < |r_x| \text{ and } |r_y| < |r_z|, \\ (-r_y, r_x, 0), & \text{if } |r_z| < |r_x| \text{ and } |r_z| < |r_y|, \end{cases} \\
\mathbf{s} &= \bar{\mathbf{s}} / ||\bar{\mathbf{s}}||, \\
\mathbf{t} &= \mathbf{r} \times \mathbf{s}.
\end{aligned}
$$

r是给定的任意旋转轴。

应用这个矩阵之后，在新的坐标系中，给定的旋转轴就是原*x*轴，计算的第二轴是原*y*轴，叉乘得出的为原*z*轴。所以接下来的旋转用绕*x*轴旋转的旋转矩阵即可，最后不要忘记反变换回去。

$$
\mathbf{M} = \begin{pmatrix} \mathbf{r}^T \\ \mathbf{s}^T \\ \mathbf{t}^T \end{pmatrix}. \tag{4.25}
$$

This matrix transforms the vector $\mathbf{r}$ into the x-axis ($\mathbf{e}_x$), $\mathbf{s}$ into the y-axis, and $\mathbf{t}$ into the z-axis. So the final transform for rotating α radians around the normalized vector $\mathbf{r}$ is then

$$
\mathbf{X} = \mathbf{M}^T \mathbf{R}_x(\alpha) \mathbf{M}. \tag{4.26}
$$

万向节锁

这种旋转也被称为欧拉变换，如果将绕*x(roll)，y(head)，z(pitch)*轴旋转的矩阵联结，我们会发现一个无法规避的问题：万向节锁（Gimbal Lock）

当 `cosp = 0` 时，矩阵中的[2, 0]、[2, 2]两个值始终为0。显然其不该如此，因为这两个值是受*h*、*p*共同影响的。这就是万向节锁的根本原因。

如果你从未了解过万向节锁，可以去查阅一些相关资料，只是仅从动画层面来解释万向节锁多少有些难以理解。

顺便一提，在Unity中测试时发现不论模型的朝向如何，绕y轴转时始终是绕世界坐标系的y轴旋转，而非本地坐标系。

投影矩阵

投影是将一定范围内的对象映射到某一平面的过程。

投影矩阵并未某一特定矩阵，根据不同的需要可以有不同的投影策略。在图形学或游戏中，最常用的两种分别是正交投影和透视投影。

正交投影

正交投影前后不改变两点之间的距离，平行线投影后仍然是平行线。所以，哪怕粗暴地将某个图元的某个坐标改为投影平面位置也称得上是一种正交投影。

正交投影矩阵通常用 `(l, r, b, t, n, f)` 6个标量来定义，分别代表*left, right, bottom, top, near, far*六个平面。在此*AABB Box*范围内的一切会被投影到某一平面，进入下一阶段的处理且有机会被渲染在显示器上，不过这是渲染管线的事儿了。

*l, r, b ,t, n, f*定义的*box*接下来会被按照一定策略变换，不同的图形库有不同的策略，OpenGL最终将其变换成一个最小点为 `(-1, -1, -1)` 最大点为 `(1, 1, 1)` 的远平面，DirectX则是从 `(-1, -1, 0)` 到 `(1, 1, 1)` 的长方体。变换的方式是平移和缩放。

这是OpenGL中的正交投影矩阵雏形，平移矩阵的结果是将定义的*box*中心平移到原点，缩放矩阵则是将*box*的长宽高都变换为2（-1~1）。在几何上，将 `(l, b, n)` 变换到 `(-1, -1, 1)`，将 `(r, t, f)` 变换到 `(1, 1, -1)`，其内部的坐标也变换到了相应的位置。

注意这并不是最终应用的结果，由于OpenGL使用的右手系是面向负z轴的，所以从数字上来说，近平面的值要大于远平面，为了便于用户理解，还会进行一次对称变换，将其改为左手系使其迎合近平面值小远平面值大的习惯。

下面是手性变换时用到的矩阵，并不难理解：

如之前所述，DirectX的策略并不相同（需要将z值映射到[0, 1]之间），但核心思想与OpenGL是一致的。

透视投影

与正交投影相比，透视投影后的平行线会在无穷远处相交，更复杂也更符合现实世界，我们的视觉就是透视投影。

透视投影范围的定义与正交投影有稍许不同，但同样是六个变量。可以想象现实场景，我们的可视范围不会是一个方体，而是一个四棱锥，最终目标仍然是要将这个可视体（平截头体）变换为位于原点的正方体（或长方体），如下图所示

现在要先将平截头体变换为一个方体，从而可以利用正交投影中的矩阵。

这个矩阵的来源同样是构造。

根据相似三角形

平截头体内的一点会以如下方式变换：

而近平面上的点变换前后是一致的：

结合以上要求，最终得到的压缩矩阵是：

将平截头体变换为方体之后，我们就回到了正交投影的轨道上，联结正交投影矩阵得到透视投影矩阵：

同样要注意，这不是最终矩阵。

经过透视投影后图元的分布情况：

显然，当 `near = 1` 时靠近近平面一侧的z坐标分布非常疏，大部分坐标集中在远平面（相对熟悉OpenGL的读者别忘记手性转换）。

大部分时候数学库是通过 `(fov, aspect, far, near)` 四个参数来定义平截头体的，通过数学手段计算所需值代入即可。

法线变换

上述变换都是针对点而言的，如果直接应用在法线（向量）上，结果不如预期：

解决方法是计算变换矩阵的逆矩阵的转置。

> The transpose of the inverse can be used to transform normals.

参考资料

Real-Time Rendering 4th

发布于 2020-06-10 16:07

矩阵 线性代数 向量

写下你的评论...

1 条评论
默认
最新

moonlight
你这个也没讲多最后一个数字是什么应用啊，为什么把3*3扩展成4*4啊
2022-05-20
1

推荐阅读

图形学基础 - 变换 - 矩阵变换基础
杨鼎超 发表于图形学基础

【解题方法】矩阵初等变换的应用
济云 发表于数学僧的青...

图形学之矩阵变换的深度理解
papalqi

【矩阵论】线性变换知识汇总
ohanl... 发表于Math