你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的<u>仿射空间</u>中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维 世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 $\$v_{0}$, 也就是点 $\$(x_{0},y_{0},z_{0})$ 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的仿射空间。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$和下所示。

\$\$

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就不一样了,需要大一个维度,也就是说,3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

\$\$

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 \\\\ 0.8 & 0.9 & 0 \\\\

```
0 & 0 & 1 \end {array} \right] $$
```

三维立体中图片放大90%就是:

\$\$

$$\begin{split} & S=\left(\left[\left(\frac{1}{2} \right) \right] \right) \\ & S=\left(\frac{1}{2} \right) \\ & S=\left(\frac{$$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘 $\$\sqrt{frac}\{1\}\{2\}$ \$,创建一个 $\$\sqrt{frac}\{1\}\{4\}$ \$的页边留白,\$x\$方向就要乘 $\$\sqrt{frac}\{3\}\{4\}$ \$,这样得到的缩放矩阵就是:

$\mathcal{P}\mathcal{P}$

S=\left[\begin{array} {III} \\frac {3} {4} & 0 & 0 \\\\ 0 & \frac {1} {2} & 0 \\\\ 0 & 0 & 1 \\end {array} \right] \\\$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样乘: \$vST\$。注意:它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从2×2就变成了3×3矩阵\$R\$。

\$\$

Q=\left[\begin{array} {cc} \\cos \theta & -\sin \theta \\\\\sin \theta & \\cos \theta \\\right[array} \right] \$\$

\$\$

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着,旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

\$\$

 $\begin{array}{l} v T_{00} R T_{45} = \left[\left[\left(\frac{3}{2} \right) \right] \\ x \& y \& 1 \\ end_{array} \right] \left[\left(\frac{3}{2} \right) \\ 1 \& 0 \& 0 \\ 0 & 1 \& 0 \\ -4 \& -5 \& 1 \\ \end{array} \right]$

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{x} = \left[\left(\frac{3}{2} \right) \right] \\ R_{x} = \left(\frac{3}{2} \right) \\ R_{x} = \left(\frac{3}{$

2.围绕\$y\$轴方向旋转:

\$\$

3.围绕\$z\$轴方向旋转:

\$\$

 $R_{z}=\left[\begin{array}{l} R_{z}=\left[R_{z}=\left[\begin{array}{l} R_{z}=\left[R_{z}=\left[\begin{array}{l} R_{z}=\left[\begin{array}{l} R_{z}=\left[R_{z}=\left[R_{z}=\left[\begin{array}{l} R_{z}=\left[R_{z}=\left$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$v\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: $\$n^{T}v=0\$$ 。

而投影到平面的投影矩阵是: \$I-nm^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量\$n\$投影后成为了0向量,而平面向量\$v\$投影后还是其自身。

```
(I-n n^{T}) n=n-n(n^{T}) n=0
$$
$$
(I-n n^{T}) v=v-n(n^{T} v)=v
$$
接下来,我们在齐次坐标中来看一下4×4的投影矩阵:
```

 $P=\left[\left\lceil \left(array \right) \right]$ & & & 0 \\\ & I-n n^{T} & & 0 \\\ & & & 0 \\\

0 & 0 & 0 & 1

\end{array}\right]

假设现在有一个不过原点的平面,\$v {0}\$是这个平面上的一个点,现在要把\$v {0}\$投影到这个平面,则需要经历三个步骤, 和刚才介绍的围绕点\$(4,5)\$, 让平面旋转\$\theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v {0}\$。

整个过程通过数学公式来表达就是:

 $T_{-v_{0}} P T_{+v_{0}} = \left[\frac{1}{c} \right]$ I & 0 \\\ -v {0} & 1 \end{array}\right]\left[\begin{array} {cc} I-n $n^{T} \& 0 \parallel$ 0 & 1 \end{array}\right]\left[\begin{array} {ll} I & 0 \\\ v {0} & 1 \end{array}\right] \$\$

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要 经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换 到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里 没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学 知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley,这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算: 平移、缩放、旋转和投影的基本概 念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了 这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的<u>仿射空间</u>中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放:
- 旋转:
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 (x_{0},y_{0}) 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$加下所示。

\$\$

T=\left[\begin{array} {IIII}
1 & 0 & 0 & 0 \\\
0 & 1 & 0 & 0 \\\\
0 & 0 & 1 & 0 \\\\\
x_{0} & y_{0} & z_{0} & 1
\end{array}\right]

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就不一样了,需要大一个维度,也就是说,3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

\$\$

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 \\\ 0 & 0.9 & 0 \\\ 0 & 0 & 0 \\\ end{array} \right]

三维立体中图片放大90%就是:

\$\$

S=\left[\begin{array} {cccc} 0.9 & 0 & 0 & 0 \\\ 0 & 0.9 & 0 & 0 & 0 \\\\ 0 & 0 & 0.9 & 0 \\\\ 0 & 0 & 0 & 0 & 1 \\end{array}\right] \$\$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘 $\$ \text{frac} \{1\} \{2\} \$$,创建一个 $\$ \text{frac} \{1\} \{4\} \$$ 的页边留白,\$x\$方向就要乘 $\$ \text{frac} \{3\} \{4\} \$$,这样得到的缩放矩阵就是:

\$\$

 $S=\left\{ \begin{array}{l} S=\left(\frac{3}{4} \& 0 \& 0 \right) \\ \frac{3}{4} \& 0 \& 0 \end{array} \right. \\ 0 \& \left(\frac{1}{2} \& 0 \right) \\ 0 \& 0 \& 1 \\ \left(\frac{4rray}{right} \right) \\ \end{array}$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样乘: \$vST\$。注意:它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从2×2就变成了3×3矩阵\$R\$。

\$\$

Q=\leff[\begin{array} {cc} \\cos \theta & -\sin \theta \\\\\\\\sin \theta & \\cos \theta \\\\\right[{array} \right] \$\$

\$\$

R=\left[\begin{array} {ccc} \\cos \theta & -\sin \theta & 0 \\\\\ \sin \theta & \cos \theta & 0 \\\\\ 0 & 0 & 1 \\end{array} \right] \$\$

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着,旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

22

R_{x}=\left[\begin{array} {cccc} 1 & 0 & 0 & 0 \\\ 0 & \cos \theta & -\sin \theta & 0 \\\ 0 & \sin \theta & \cos \theta & 0 \\\ 0 & 0 & 0 & 1 \end{array}\right] \$\$

2.围绕\$y\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{y} = \left(\frac{1}{2} \cdot \frac{2}{2} \right) & ccc \\ \cos \theta & ccc \\ \cos \theta & ccc \\ 0 & 1 & 0 & ccc \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & ccc \\ 0 & 0 & 0 & 1 \\ - ccc & ccc \\ 0 & 0 & 0 & 1 \\ - ccc & ccc \\ 0 & 0 & 0 & 1 \\ - ccc & ccc \\ 0 & 0 0 &$

ΦΦ

3.围绕\$z\$轴方向旋转:

\$\$

R_{z}=\leff[\begin{array} {cccc} \\cos \theta & -\sin \theta & 0 & 0 \\\\\sin \theta & \cos \theta & 0 & 0 \\\\ 0 & 0 & 1 & 0 \\\\\ 0 & 0 & 0 & 1 \\\\\\\\$ind{array}\right] \$\$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: \$n^{T}v=0\$。

而投影到平面的投影矩阵是: \$I-m^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量**\$n\$**投影后成为了**0**向量,而平面向量**\$v\$**投影后还是其自身。

\$\$ (I-n n^{T}) n=n-n(n^{T} n)=0 \$\$ \$\$ (I-n n^{T}) v=v-n(n^{T} v)=v \$\$

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$
P=\left[\begin{array} {III}
& & & 0 \\\
& I-n n^{T} & & 0 \\\
& & & 0 \\\
& & & 0 \\\
o & 0 & 0 & 1
\end {array} \right]
\$\$

假设现在有一个不过原点的平面, v_{0} \$是这个平面上的一个点,现在要把 v_{0} \$投影到这个平面,则需要经历三个步骤,和刚才介绍的围绕点(4,5)\$,让平面旋转 \hbar 0}\$从theta\$角度经历的三个步骤类似:

- 1. 把\$v_{0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v {0}\$。

整个过程通过数学公式来表达就是:

\$\$ \$T_{-v_{0}} P T_{+v_{0}}=\left[\left\lceil \frac{ array}{cc} I \& 0 \right] \$\$ I \& 0 \right] \$\$ -v_{0} \& 1 \$\$ end {array} \right]\left[\left\lceil \frac{ array}{cc} I-n n^{T} \& 0 \right] \$\$ 0 \& 1 \$\$ end {array} \right]\left[\left\lceil \frac{ array}{I} I \& 0 \right] \$\$ 1 \& 0 \right] \$\$ 1 \$\$ end {array} right] \$\$ \$\$ 1 \$\$ end {array} right] \$\$

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算:平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的<u>仿射空间</u>中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 (x_{0},y_{0},z_{0}) 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是 $\mathbf{sf(a+b)}$ \$不等于 $\mathbf{sf(a)+f(b)}$ \$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以, $\mathbf{3} \times \mathbf{3}$ 矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$加下所示。

\$\$

T=\left[\begin{array} {IIII}
1 & 0 & 0 & 0 \\\
0 & 1 & 0 & 0 \\\
0 & 0 & 1 & 0 \\\\
x_{0} & y_{0} & z_{0} & 1
\end{array}\right]

平移的整个过程是这样的: 假设要把原来的某个点x,y,z*** R\$\text{0}\$, 我们需要切换到齐次坐标x,y,z**, 然后,x,y,z**, 就能得到每个原来的向量x*** R\$\text{0}\$\$ R\$\tex

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位 矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就 不一样了, 需要大一个维度, 也就是说, 3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

\$\$

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 $0 \& 0.9 \& 0 \parallel$ 0 & 0 & 1 \end{array}\right]

三维立体中图片放大90%就是:

\$\$

S=\left[\begin{array} {cccc} 0.9 & 0 & 0 & 00 & 0.9 & 0 & 0 $0 \& 0 \& 0.9 \& 0 \parallel$ 0 & 0 & 0 & 1 \end{array}\right] \$\$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘\$\frac{1}{2}\$,创建一 个\$\frac{1}{4}\$的页边留白, \$x\$方向就要乘\$\frac{3}{4}\$, 这样得到的缩放矩阵就是:

\$\$

 $S=\left\{ \frac{11}{s} \right\}$ \frac{3}{4} & 0 & 0 \\\ $0 \& \frac{1}{2} \& 0$ 0 & 0 & 1 \end{array}\right] \$\$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样 乘: \$vST\$。注意: 它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从 2×2就变成了3×3矩阵\$R\$。

Q=\left[\begin{array} {cc} \cos \theta & -\sin \theta \\\ \sin \theta & \cos \theta \end{array}\right] \$\$

\$\$

R=\left[\begin{array} {ccc} $\cos \theta \& -\sin \theta \& 0$ $\sin \theta \& \cos \theta \ 0$ 0 & 0 & 1 \end{array}\right]

\$\$

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转, 而是先平移再旋转,比如我们要围绕点\$(4,5)\$, 让平面旋转\$\theta\$角度的话:

1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;

- 2. 接着, 旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

22

 $v\ T_{\{00\}}\ R\ T_{\{45\}} = \left[\left\lceil \left\lceil \frac{1}{2} \right\rceil \right\rceil \right]$

x & y & 1

\end \array\\right]\left[\begin\array\ \ccc\

 $1 & 0 & 0 \$

0 & 1 & 0 \\\

-4 & -5 & 1

\end{array}\right]\left[\begin{array} {ccc}

 $\cos \theta \& -\sin \theta \& 0$

 $\sin \theta \& \cos \theta \ 0$

0 & 0 & 1

\end{array}\right]\left[\begin{array} {ccc}

1 & 0 & 0 \\\

0 & 1 & 0 \\\

4 & 5 & 1

\end{array}\right]

\$\$

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\$\$

1&0&0&0\\\

 $0 \& \cos \theta \& -\sin \theta \& 0$

 $0 \& \sin \theta \& \cos \theta \& 0$

0 & 0 & 0 & 1

\end{array}\right]

22

2.围绕\$y\$轴方向旋转:

\$\$

 $R \{y\} = \left(\frac{y}{\sqrt{2\pi}} \right)$

\cos \theta & 0 & \sin \theta & 0 \\\

0 & 1 & 0 & 0 \\\

-\sin \theta & 0 & \cos \theta & 0 \\\

0 & 0 & 0 & 1

\end{array}\right]

\$\$

3.围绕\$z\$轴方向旋转:

\$\$

 $R_{z}=\left[\left[\left[\left(x\right) \right] \right] \right]$

\cos \theta & -\sin \theta & 0 & 0 \\\

 $\sin \theta \& \cos \theta \$

0 & 0 & 1 & 0 \\\

0 & 0 & 0 & 1

\end{array}\right]

\$\$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: $$n^{T}v=0$$ 。

而投影到平面的投影矩阵是: \$I-nm^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量**\$n\$**投影后成为了**0**向量,而平面向量**\$v\$**投影后还是其自身。

\$\$ (I-n n^{T}) n=n-n(n^{T} n)=0 \$\$ \$\$

 $(I-n n^{T}) v=v-n(n^{T}) v=v$

ΨΨ

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$
P=\left[\begin{array} {\!!!}
& & & 0 \\\
& I-n n^{T} & & 0 \\\
& & & 0 \\\\
& & & 0 \\\\
& & & 0 \\\\
| 0 & 0 & 0 & 1 \\
| \end {array} \right]

\$\$

假设现在有一个不过原点的平面, v_{0} \$是这个平面上的一个点,现在要把 v_{0} \$投影到这个平面,则需要经历三个步骤,和刚才介绍的围绕点(4,5)\$,让平面旋转 \hbar 0}\$从theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v {0}\$。

整个过程通过数学公式来表达就是:

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算:平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的仿射空间中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 $\$v_{0}$,也就是点 $\$(x_{0},y_{0},z_{0})$ 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$加下所示。

\$\$

这里很重要的一点是, 计算机图形图像是基于行向量计算的。也就是说, 计算方法是行乘矩阵, 而不是矩阵乘列, 比如:

 $\left[\left(x \in \mathbb{Z} \right) \right] = 1 - 1$

平移的整个过程是这样的:假设要把原来的某个点\$(x,y,z)\$平移 $$v_{0}$ \$,我们需要切换到齐次坐标\$(x,y,z,1)\$,然后,\$(x,y,z,1)\$再乘\$T\$,就能得到每个原来的向量\$v\$平移到 $$v+v_{0}$ \$的最终结果: $$\left[\text{begin} {\text{array} {\text{IIII}} x \& y \& z \& 1 \end{\text{array} right} T=\left[\text{begin} {\text{array} {\text{IIII}} x + x_{0} \& y + y_{0} \& z + z_{0} \& 1 \end{\text{array} right} \right]} $$ 。

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就不一样了,需要大一个维度,也就是说,3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

\$\$

$$\begin{split} & S=\left[\left[\left[\left(\frac{3}{2} \right) \right] \right] \\ & 0.9 \& 0 \& 0 \right] \\ & 0.9 \& 0 \right] \\ & 0.9 \& 0 \right] \\ & 0.9 \& 0 \\ & 0.9 \& 0 \right] \\ & \left[\left(\frac{3}{2} \right) \right] \\ & \left[\left(\frac{3$$

三维立体中图片放大90%就是:

\$\$

S=\leff[\begin{array} {cccc} 0.9 & 0 & 0 & 0 \\\ 0 & 0.9 & 0 & 0 & 0 \\\\ 0 & 0.9 & 0 & 0 \\\\ 0 & 0 & 0 & 0 & 1 \\end{array}\right]

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘 $\$fac{1}{2}\$$,创建一个 $\$fac{1}{4}\$$ 的页边留白,\$x\$方向就要乘 $\$fac{3}{4}\$$,这样得到的缩放矩阵就是:

\$\$

 $S=\left\{ \begin{array}{l} S=\left(\frac{3}{4} \& 0 \& 0 \right) \\ 0 \& \left(\frac{1}{2} \& 0 \right) \\ 0 \& 0 \& 1 \\ \end{array} \right.$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样乘: \$vST\$。注意:它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从2×2就变成了3×3矩阵\$R\$。

\$\$

Q=\leff[\begin{array} {cc} \\cos \theta & -\sin \theta \\\\\\\sin \theta & \\cos \theta \\\\right[{array} \right] \$\$

\$\$

0 & 0 & 1 \end {array} \right] \$\$

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着,旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

\$\$

 $\begin{array}{l} v T_{00} R T_{45} = \left[\left[\left(\frac{2}{5} \right) \right] \\ x \& y \& 1 \\ \end{array} \right] \\ v d_{array} \right] \left[\left(\frac{2}{5} \right) \right] \\ v d_{array} \left[\left(\frac{2}{5} \right) \right] \\ v d_{array} \right] \left[\left(\frac{2}{5} \right) \right] \\ v d_{array} \left[\left(\frac{2}{5} \right) \right] \\ v d_{ar$

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\end{array}\right]

\$\$

 $\begin{array}{l} R_{x} = \left[\left[\left(\frac{3}{4} \right) \right] \\ R_{x} = \left(\frac{3}{4} \right) \\ R_{x} = \left(\frac{$

2.围绕\$y\$轴方向旋转:

22

 $\begin{array}{l} R_{y}=\left| \begin{array}{c} R_{y}=\left| \begin{array}{c} CCC \end{array} \right| \\ Cos \right| \\ Cos \right| \\ Cos \left| \begin{array}{c} CCC \end{array} \right| \\ COS \right| \\ COS \left| \begin{array}{c} CCC \end{array} \right| \\ COS \left| CCC \right| \\ COS \left| CCC$

3.围绕\$z\$轴方向旋转:

\$\$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: $$n^{T}v=0$$ 。

而投影到平面的投影矩阵是: \$I-nm^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量**\$n\$**投影后成为了**0**向量,而平面向量**\$v\$**投影后还是其自身。

```
$$
(I-n n^{T}) n=n-n(n^{T} n)=0
$$
$$
(I-n n^{T}) v=v-n(n^{T} v)=v
$$
```

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$
P=\left[\begin\{array\} \{\!\}
& & & 0 \\\
& I-n n^{T} & & 0 \\\
& & & 0 \\\\
& & & 0 \\\\
0 & 0 & 0 & 1 \\end\{array\}\right]
\$\$

假设现在有一个不过原点的平面, v_{0} \$是这个平面上的一个点,现在要把 v_{0} \$投影到这个平面,则需要经历三个步骤,和刚才介绍的围绕点(4,5)\$,让平面旋转 \hbar 0}\$从theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v {0}\$。

整个过程通过数学公式来表达就是:

\$\$ T_{-v_{0}} P T_{+v_{0}}=\left[\left\lceil \frac{ \arctan y}{ cc} I \& 0 \right] \\ I \& 0 \right] \\ -v_{0} \& 1 \\ -v_{0} \& 1 \\ -n n^{T} \& 0 \right] \\ 0 \& 1 \\ -n d_{array} \right] \left[\left\lceil \frac{ \arctan y}{ array} { ll} I \& 0 \right] \\ 0 \& 1 \\ -n d_{array} \right] \left[\left\lceil \frac{ \arctan y}{ array} { ll} I \& 0 \right] \\ I \& 0 \right] \\ -v_{0} \& 1 \\ -v_{0

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换

到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里 没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算: 平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的<u>仿射空间</u>中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移;
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 (x_{0},y_{0},z_{0}) 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$

```
1 \& 0 \& 0 \& 0 \%
0 & 1 & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
x \{0\} \& y \{0\} \& z \{0\} \& 1
\end{array}\right]
$$
```

这里很重要的一点是, 计算机图形图像是基于行向量计算的。也就是说, 计算方法是行乘矩阵, 而不是矩阵乘列, 比如: $\left[\left(x < 0 \ & z < 0 \ & 1 \right) \ & 0 \ & 1 \right] T = \left[\left(x < 0 \ & z < 0 \right) \ & z < 0 \right] \ & 1 \ & 1 \ & 2 \ & 3$

平移的整个过程是这样的: 假设要把原来的某个点\$(x,y,z)\$平移\$v {0}\$, 我们需要切换到齐次坐标\$(x,y,z,1)\$, 然 后, \$(x,y,z,1)\$再乘\$T\$, 就能得到每个原来的向量\$v\$平移到\$v+v {0}\$的最终结果: \$\left\begin{array} {III} x & y & z & $1\end{array} T=\left[\left[\frac{3x+x}{0} & y+y \\ 0 \right] & z+z \\ 0 \ & 1\end{array}\right].$

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位 矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就 不一样了,需要大一个维度,也就是说,3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 \\\ $0 \& 0.9 \& 0 \parallel$ 0 & 0 & 1 \end{array}\right] \$\$

三维立体中图片放大90%就是:

\$\$

 $S=\left[\left(\frac{1}{2} \right) \right]$ 0.9 & 0 & 0 & 0 %0 & 0.9 & 0 & 0 \\\ 0 & 0 & 0.9 & 0\\\ 0 & 0 & 0 & 1 \end{array}\right] \$\$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘\$\frac{1}{2}\$,创建一 个 ${\frac{1}{4}}$ 的页边留白,x方向就要乘 ${\frac{3}{4}}$,这样得到的缩放矩阵就是:

\$\$

S=\left[\begin{array} {lll} \frac{3}{4} & 0 & 0 \\\ 0 & \frac{1}{2} & 0 \\\ 0 & 0 & 1 \end{array}\right]

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘:\$vTS\$,如果我们要先缩放再平移,那应该这样 乘: \$vST\$。注意: 它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从 2×2就变成了3×3矩阵\$R\$。

Q=\left[\begin{array} {cc} \cos \theta & -\sin \theta \\\

0 & 0 & 1 \end {array}\right]

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着, 旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

```
$$
```

v T_{00} R T_{45}=\left[\left[\left[\left(\frac{27}{45}\right)\right] \right] x & y & 1 \\ = \left(\frac{27}{45}\right] \left[\left(\frac{27}{45}\right)\right] \\ = \left(\frac{27}{45}\right) \\ = \left(\frac{27}{

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\$\$

R_{x}=\leff[\begin{array} {cccc} 1 & 0 & 0 & 0 \\\ 0 & \cos \theta & -\sin \theta & 0 \\\\ 0 & \sin \theta & \cos \theta & 0 \\\\ 0 & 0 & 0 & 1 \\end{array} \right] \$\$

2.围绕\$y\$轴方向旋转:

\$\$

R_{y}=\left[\begin{array} {cccc} \\cos \theta & 0 & \sin \theta & 0 \\\\ 0 & 1 & 0 & 0 & \\\-\sin \theta & 0 & \\\ 0 & 0 & 0 & 1 \\\end{array} \right] \$\$

3.围绕\$z\$轴方向旋转:

\$\$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$v\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: $\$n^{T}v=0\$$ 。

而投影到平面的投影矩阵是: \$I-nm^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量**\$n\$**投影后成为了**0**向量,而平面向量**\$v\$**投影后还是其自身。

\$\$ (I-n n^{T}) n=n-n(n^{T} n)=0 \$\$ \$\$ (I-n n^{T}) v=v-n(n^{T} v)=v

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$

P=\leff[\begin{array} {\!!!} & & & 0 \\\\ & I-n n^{T} & & 0 \\\\ & & & 0 \\\\\ & & & 0 \\\\\ & & & 1 \\end{array}\right] \\
\$\$

假设现在有一个不过原点的平面, $$v_{0}$ \$是这个平面上的一个点,现在要把 $$v_{0}$ \$投影到这个平面,则需要经历三个步骤,和刚才介绍的围绕点\$(4,5)\$,让平面旋转 $$\theta$ 的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v_{0}\$。

整个过程通过数学公式来表达就是:

\$\$

 $T_{-v_{0}} P T_{+v_{0}} = \left[\left\lceil \frac{\alpha ray}{cc} I \& 0 \right] \\ -v_{0} \& 1 \\ -v_{0} \& 1 \\ -r_{0} \& 1 \\ -r_{1} \& 0 \\ -r_{1} \& 0 \\ -r_{1} & r_{1} & r_{2} \\ -r_{1} & r_{2} \\ -r_{1} & r_{2} \\ -r_{2} & r_{$

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里 没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算: 平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的仿射空间中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维 世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放:
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 x_{0} y_{0} , y_{0

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量\$v {0}\$平移整个三维空间,把原点平移到了\$(x {0},y {0},z {0})\$,这也就意味着三维空间的每个点都加上了点 \$(x {0}, y {0}, z {0})\$。使用齐次坐标,把整个空间平移了\$v {0}\$的4×4矩阵\$T\$如下所示。

\$\$

T=\left[\begin{array} {llll} 1 & 0 & 0 & 0 \\\ 0 & 1 & 0 & 0 \\\ 0 & 0 & 1 & 0 \\\ $x_{0} & y_{0} & z_{0} & 1$ $\end{array}\rightarrow$

这里很重要的一点是, 计算机图形图像是基于行向量计算的。也就是说, 计算方法是行乘矩阵, 而不是矩阵乘列, 比如: $\label{thm:condition} $\left(x_{0} & y_{0} & z_{0} & 1 \right) T = \left(x_{0} & y_{0} & z_{0} & 1 \right) . $$$

平移的整个过程是这样的: 假设要把原来的某个点\$(x,y,z)\$平移\$v {0}\$, 我们需要切换到齐次坐标\$(x,y,z,1)\$, 然 后, \$(x,y,z,1)\$再乘\$T\$, 就能得到每个原来的向量\$v\$平移到\$v+v {0}\$的最终结果: \$\leff\begin{array} {III} x & y & z & $1\end{array} $$ T=\left[\left[\frac{3 & y+y_{0} & z+z_{0} & 1\end{array}\right]}.$

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如: 把图片整体放大90%, 那么在线性代数中就是0.9乘单位 矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就 不一样了, 需要大一个维度, 也就是说, 3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 \\\ 0 & 0.9 & 00 & 0 & 1 \end{array}\right]

\$\$

三维立体中图片放大90%就是:

\$\$

S=\left[\begin{array} {cccc} 0.9 & 0 & 0 & 0 %0 & 0.9 & 0 & 0 $0 \& 0 \& 0.9 \& 0 \parallel$ 0 & 0 & 0 & 1 \end{array}\right]

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘\$\frac{1}{2}\$,创建一 个\$\frac {1} {4}\$的页边留白, \$x\$方向就要乘\$\frac {3} {4}\$, 这样得到的缩放矩阵就是:

\$\$

S=\left[\begin{array} {lll} \frac{3}{4} & 0 & 0 \\\ $0 \& \frac{1}{2} \& 0$ 0 & 0 & 1 \end{array}\right] \$\$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样 乘: \$vST\$。注意: 它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从 2×2 就变成了 3×3 矩阵\$R\$。

\$\$

\$\$

R=\left[\begin\{array\} \{ccc\} \\cos \theta & -\sin \theta & 0 \\\\\sin \theta & \cos \theta & 0 \\\\ 0 & 0 & 1 \\end\{array\}\right]

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着,旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

\$\$

 $\label{eq:cc} $$v T_{00} R T_{45}=\left[\left[\left(\frac{25}{1 \& 0 \& 0 }\right)\right] \left(\frac{25}{1 \& 0 \& 0 }\right) \right] \left(\frac{25}{1 \& 0 \& 0 }\right) \left(\frac{25}{1 \& 0 \& 0 }\right)$

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\$\$

R_{x}=\left[\begin{array} {cccc} 1 & 0 & 0 & 0 \\\ 0 & \cos \theta & -\sin \theta & 0 \\\ 0 & \sin \theta & \cos \theta & 0 \\\ 0 & 0 & 0 & 1 \\end{array} \right] \$\$

2.围绕\$y\$轴方向旋转:

\$\$

 $R_{y}=\left[\left\langle array\right\rangle \left\langle ccc\right\rangle \right]$

```
0 \& 1 \& 0 \& 0 \
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
```

3.围绕\$z\$轴方向旋转:

\$\$

 $R \{z\} = \left(\frac{x}{z} \right)$ $\cos \theta \& -\sin \theta \& 0 \& 0$ $\sin \theta \& \cos \theta \ 0 \& 0 \$ 0 & 0 & 1 & 00 & 0 & 0 & 1 \end{array}\right] \$\$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则 不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下第九篇的内 容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等 式: \$n^{T}v=0\$。

而投影到平面的投影矩阵是: \$I-m^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量\$n\$投影后成 为了0向量,而平面向量\$v\$投影后还是其自身。

 $(I-n n^{T}) n=n-n(n^{T}) n=0$

\$\$

\$\$

 $(I-n n^{T}) v=v-n(n^{T} v)=v$

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

P=\left[\begin{array} {lll} & & & 0 \\\ & I-n n^{T} & & 0 \\\ & & & 0 \\\ 0 & 0 & 0 & 1 \end{array}\right]

\$\$

假设现在有一个不过原点的平面,\$v {0}\$是这个平面上的一个点,现在要把\$v {0}\$投影到这个平面,则需要经历三个步骤, 和刚才介绍的围绕点\$(4,5)\$, 让平面旋转\$\theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v_{0}\$。

整个过程通过数学公式来表达就是:

 $T_{-v_{0}} P T_{+v_{0}} = \left[\frac{1}{c} \right]$ $I \& 0 \, \text{n}$ -v {0} & 1 \end{array}\right]\left[\begin{array} {cc}

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算:平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的<u>仿射空间</u>中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道, 计算机图形图像处理的是图片, 且计算机屏幕是二维的。那你有没有想过, 我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢? 这个就要涉及到三维到二维的投影技术了, 这类技术都离不开矩阵, 而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移;
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 (x_{0},y_{0},z_{0}) 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点

之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的 描述,就需要使用第九篇中说的<u>仿射空间</u>。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了 4×4 矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如: 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 $v \{0\}$ \$平移整个三维空间,把原点平移到了 $x \{0\}$, $v \{0\}$, $z \{0\}$,s, 这也就意味着三维空间的每个点都加上了点 \$(x {0},y {0},z {0})\$。使用齐次坐标,把整个空间平移了\$v {0}\$的4×4矩阵\$T\$如下所示。

T=\left[\begin{array} {llll} 1 & 0 & 0 & 0 \\\ $0 \& 1 \& 0 \& 0 \$ $0 \& 0 \& 1 \& 0 \$ $x \{0\} \& y \{0\} \& z \{0\} \& 1$ \end{array}\right]

这里很重要的一点是, 计算机图形图像是基于行向量计算的。也就是说, 计算方法是行乘矩阵, 而不是矩阵乘列, 比如: $\left[\frac{3}{2} \right] 0 & 0 & 0 & 1 \end{array}$ 1=\left[\begin{\array}{\lll} x \ \ 0 \ & y \ \ 0 \ & z \ \ 0 \ & 1 \end{\array}\right]\\$.

平移的整个过程是这样的: 假设要把原来的某个点\$(x,y,z)\$平移\$v {0}\$, 我们需要切换到齐次坐标\$(x,y,z,1)\$, 然 后, \$(x,y,z,1)\$再乘\$T\$, 就能得到每个原来的向量\$v\$平移到\$v+v {0}\$的最终结果: \$\left\begin{array} {Ш} x & y & z & $1\end{array} \ T=\left[\left[\frac{3 & y+y_{0} & z+z_{0} & 1\end{array}\right]}{2} & 1\end{array}\right].$

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位 矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就 不一样了, 需要大一个维度, 也就是说, 3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

\$\$

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 \\\ 0 & 0.9 & 0 \\\ 0 & 0 & 1 \end{array}\right]

三维立体中图片放大90%就是:

S=\left[\begin{array} {cccc} 0.9 & 0 & 0 & 0 %0 & 0.9 & 0 & 0 $0 \& 0 \& 0.9 \& 0 \parallel$ 0&0&0&1 \end{array}\right] \$\$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘\$\frac{1}{2}\$,创建一 个\$\frac{1}{4}\$的页边留白,\$x\$方向就要乘\$\frac{3}{4}\$,这样得到的缩放矩阵就是:

 $S=\left[\frac{array}{111} \right]$ \frac{3}{4} & 0 & 0 \\\ $0 \& \frac{1}{2} \& 0$ 0 & 0 & 1 \end{array}\right] \$\$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样乘: \$vST\$。注意:它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从 2×2 就变成了 3×3 矩阵\$R\$。

22

\$\$

R=\left[\begin{array} {ccc} \\ \cos \theta & -\sin \theta & 0 \\\ \\ \sin \theta & \cos \theta & 0 \\\ 0 & 0 & 1 \\ \end {array} \right] \$\$

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着, 旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

\$\$

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\$\$

R_{x}=\left[\begin{array} {cccc}
1 & 0 & 0 & 0 \\\
0 & \cos \theta & -\sin \theta & 0 \\\
0 & \sin \theta & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end {array} \right]
\$\$

2.围绕\$y\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{y}=\left(\frac{1}{2} \cdot \frac{2}{2} \cdot$

3.围绕\$z\$轴方向旋转:

\$\$

 $R_{z} = \left[\left[\left[\left(\frac{2}{2} \right) \right] \right] \right]$ \(\cos \text{ theta & -\sin \text{ theta & 0 & 0 \\\ \sin \text{ theta & 0 & 0 \\\ 0 & 0 & 1 & 0 \\\ 0 & 0 & 0 & 1 \\ \end{array} \right]

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式:\$n^{T}v=0\$。

而投影到平面的投影矩阵是: \$I-nm^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量**\$n\$**投影后成为了**0**向量,而平面向量**\$v\$**投影后还是其自身。

\$\$

 $(I-n n^{\uparrow}\{T\}) n=n-n(n^{\uparrow}\{T\} n)=0$ \$\$

\$\$

 $(I-n n^{(T)}) v=v-n(n^{(T)} v)=v$

\$\$

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$

假设现在有一个不过原点的平面, v_{0} \$是这个平面上的一个点,现在要把 v_{0} \$投影到这个平面,则需要经历三个步骤,和刚才介绍的围绕点(4,5)\$,让平面旋转 \hbar 0}\$从heta\$角度经历的三个步骤类似:

- 1. 把\$v_{0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v_{0}\$。

整个过程通过数学公式来表达就是:

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley,这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算: 平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的<u>仿射空间</u>中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维 世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放;
- 旋转;

投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 x_{0},y_{0} 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$和下所示。

22

 $T=\left\{\left[\left(\frac{3}{2}\right) \left(\frac{3}{2}\right)\right] \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ x_{0} & y_{0} & z_{0} & 1 \\ case & 1 & 0 \\ case & 1$

平移的整个过程是这样的: 假设要把原来的某个点x,y,z*平移 x_{0} \$,我们需要切换到齐次坐标x,y,z*1)\$,然后,x,y,z*1)\$再乘\$T\$,就能得到每个原来的向量x****P移到 $x+v_{0}$ \$的最终结果: \$\left{\begin{array}{\lll}x & y & z & 1\end{array}\right] T=\left{\begin{array}{\lll}x+x {0} & y+y {0} & z+z {0} & 1\end{array}\right}\$.

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就不一样了,需要大一个维度,也就是说,3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

\$\$

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 \\\ 0 & 0.9 & 0 \\\ 0 & 0 & 0 \\\ 1 \end{array} \right]

三维立体中图片放大90%就是:

\$\$

S=\leff[\begin{array} {cccc} 0.9 & 0 & 0 & 0 \\\ 0 & 0.9 & 0 & 0 & 0 \\\\ 0 & 0.9 & 0 & 0 \\\\ 0 & 0 & 0 & 0 & 1 \\end{array} \right] \$\$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘 $\$\sqrt{frac}\{1\}\{2\}$ \$,创建一个 $\$\sqrt{frac}\{1\}\{4\}$ \$的页边留白,\$x\$方向就要乘 $\$\sqrt{frac}\{3\}\{4\}$ \$,这样得到的缩放矩阵就是:

 $S=\left[\left[begin\{array\} \{III\} \right] \\ frac \{3\} \{4\} \& 0 \& 0 \right] \\ 0 \& frac \{1\} \{2\} \& 0 \right] \\ 0 \& 0 \& 1 \\ end \{array\} \right] \\ \$$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样乘: \$vST\$。注意:它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从 2×2 就变成了 3×3 矩阵\$R\$。

\$\$

\$\$

 $R=\left[\left[\left(\frac{2\pi }{2\pi } \right) \right] \\ \cos \theta & -\sin \theta & 0 \right] \\ \sin \theta & \cos \theta & 0 \right] \\ 0 & 0 & 1 \\ end & (2\pi)^{right} \\ $$

ФФ

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着, 旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

\$\$

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

```
 \begin{array}{l} R_{x} = \left[ \left( \frac{3}{4} \right) \right] & C \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \\ -\sin \theta & \cos \theta & 0 \\ \end{array}
```

2.围绕\$y\$轴方向旋转:

\$\$ R_{y}=\left[\begin{array} {cccc} \\cos \theta & 0 & \sin \theta & 0 \\\ 0 & 1 & 0 & 0 & \\\-\sin \theta & 0 & \\\ 0 & 0 & 0 & 1 \\\ 0 & 0 & 0 & 1 \\\\ end {array} \right] \$\$

3.围绕\$z\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{z} = \left[\left[\left(\frac{3}{2} \right) \right] \\ \cos \left(\frac{4}{2} \right) \\ \cos \left(\frac{$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: $\$n^{T}v=0\$$ 。

而投影到平面的投影矩阵是: \$I-mr^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量**\$n\$**投影后成为了**0**向量,而平面向量**\$v\$**投影后还是其自身。

\$\$ (I-n n^{T}) n=n-n(n^{T} n)=0 \$\$ \$\$ (I-n n^{T}) v=v-n(n^{T} v)=v \$\$

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$ P=\left[\begin{array} {III} & & & 0 \\\ & I-n n^{T} & & 0 \\\ & & & 0 \\\ & & & 1 \end{array} \right] \$\$

假设现在有一个不过原点的平面,\$v_{0}\$是这个平面上的一个点,现在要把\$v_{0}\$投影到这个平面,则需要经历三个步骤,

和刚才介绍的围绕点\$(4,5)\$, 让平面旋转\$\theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v {0}\$。

整个过程通过数学公式来表达就是:

 $T_{-v_{0}} P T_{+v_{0}} = \left[\left(\frac{3}{2} \right) \right]$ I & 0 & 0 & 1 $-v_{0} & 1$ $\left(\frac{3}{2} \right) \left(\frac{3}{2} \right)$ $\left(\frac{3}{2} \right)$ $\left(\frac{3}{2} \right)$ $\left(\frac{3}{2} \right)$ $\left(\frac{3}{2} \right)$

v_{0} & 1

\end{array}\right]

\$\$

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算: 平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的<u>仿射空间</u>中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维

世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移;
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 x_{0} y_{0} , y_{0} .

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点 之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的 描述,就需要使用第九篇中说的仿射空间。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$加下所示。

\$\$

 $T=\left\{ \left[\left(\frac{3 \times 0 & 0 \times 0}{1 & 0 & 0 & 0} \right) \right] \\ 1 & 0 & 0 & 0 & 0 \times 0 \\ 0 & 0 & 0 & 0 & 0 \times 0 \\ x_{\{0\}} & y_{\{0\}} & z_{\{0\}} & 1 \\ end_{array}\right] \\ ss$

平移的整个过程是这样的: 假设要把原来的某个点x,y,z平移 x_{0} 、我们需要切换到齐次坐标x,y,z1)、然后,x,y,z1)、再乘x1、就能得到每个原来的向量x2、数目\end{array}\right] T=\left[\begin{array} {\lll|} x & y & z & 1\end{array}\right] T=\left[\begin{array} {\lll|} x & y & z & 1\end{array}\right] & 1\end{array}\right] & 1\end{array}\right] & 1\end{array}\right] & 1\end{array}\right] \lll|

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就不一样了,需要大一个维度,也就是说,3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

\$\$

 $S=\left[\left[\left[\left(\frac{3 - 1}{2} \right) \right] \right] \\ S=\left[\left(\frac{3 - 1}{2} \right) \right] \\ S=\left[\left(\frac{3 - 1}{2}$

三维立体中图片放大90%就是:

22

S=\left[\begin{array} {cccc} 0.9 & 0 & 0 & 0 \\\ 0 & 0.9 & 0 & 0 & 0 \\\\ 0 & 0.9 & 0 & 0 \\\\ 0 & 0 & 0.9 & 0 \\\\

```
0 & 0 & 0 & 1 \end {array} \right] $$
```

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘 $\$\sqrt{a}$ {2}\$,创建一个 $\$\sqrt{a}$ {4}\$的页边留白,\$x\$方向就要乘 $\$\sqrt{a}$ {4}\$,这样得到的缩放矩阵就是:

22

$$\begin{split} & S \!\!=\!\! \text{leff[egin{array} \{III\} \\ frac{3}{4} \& 0 \& 0 \text{ \ldots } \\ 0 \& frac{1}{2} \& 0 \text{ \ldots } \\ 0 \& 0 \& 1 \\ \text{lend{array} \\ right]} } \end{split}$$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样乘: \$vST\$。注意:它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从2×2就变成了3×3矩阵\$R\$。

\$\$

Q=\left[\begin{array} {cc} \\cos \theta & -\sin \theta \\\\\sin \theta & \\cos \theta \\end {array} \right] \$\$

\$\$

 $R = \left[\left[\left(\frac{2 \times 0}{1 + 2 \times 0} \right) \right] \\ \cos \left(\frac{4 \times 0}{1 + 2 \times 0} \right) \\ \sin \left(\frac{4 \times 0}{1 + 2 \times 0} \right) \\ \cos \left(\frac{4 \times 0}{1 + 2 \times$

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着, 旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

22

\$\$

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是, 围绕\$\(\lambda=1\)\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

 $R_{x}=\left[\left\langle x\right\rangle \right]$ $1 \& 0 \& 0 \& 0 \$ $0 \& \cos \theta \& -\sin \theta \& 0 \le 0 \le 0$ $0 \& \sin \theta \& \cos \theta \& 0 \le$ 0 & 0 & 0 & 1 \end{array}\right] \$\$

2.围绕\$y\$轴方向旋转:

\$\$

 $R \{y\} = \left[\left(\frac{1}{y} \right) \right]$ $\cos \theta \& 0 \& \sin \theta \& 0$ $0 \& 1 \& 0 \& 0 \$ -\sin \theta & 0 & \cos \theta & 0 \\\ 0 & 0 & 0 & 1 \end{array}\right] \$\$

3.围绕\$z\$轴方向旋转:

\$\$

 $R \ \{z\} = \left\{ \left\{ ccc \right\} \right\}$ \cos \theta & -\sin \theta & 0 & 0 \\\ $\sin \theta \& \cos \theta \& 0 \& 0$ 0 & 0 & 1 & 0 \\\ 0 & 0 & 0 & 1 \end{array}\right] \$\$

你看出来哪里不同了吗? 其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则 不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内 容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等 式: \$n^{T}v=0\$。

而投影到平面的投影矩阵是: $I-m^{T}$ 。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量\$n\$投影后成 为了0向量,而平面向量\$v\$投影后还是其自身。

\$\$

```
(I-n n^{T}) n=n-n(n^{T}) n=0
$$
```

\$\$

```
(I-n n^{T}) v=v-n(n^{T} v)=v
```

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$

 $P=\left[\left[\left(array \right) \right] \right]$

& & & 0 \\\
& I-n n^{T} & & 0 \\\
& & & 0 \\\
0 & 0 & 0 & 1 \end{array}\right]
\$\$

假设现在有一个不过原点的平面, v_{0} 是这个平面上的一个点,现在要把 v_{0} \$投影到这个平面,则需要经历三个步骤,和刚才介绍的围绕点(4,5)\$,让平面旋转 θ \$\theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v {0}\$。

整个过程通过数学公式来表达就是:

\$\$

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里 没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算: 平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的仿射空间中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 (x_{0},y_{0},z_{0}) 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$和下所示。

\$\$

$$\begin{split} & T \!\!=\!\! \text{leff[\begin{array} { IIII} } \\ & 1 \& 0 \& 0 \& 0 \& 0 \, \text{\ldots} \\ & 0 \& 1 \& 0 \& 0 \, \text{\ldots} \\ & 0 \& 0 \& 1 \& 0 \, \text{\ldots} \\ & x_{0} \& y_{0} \& z_{0} \& 1 \\ & \text{end {array} right]} \\ \end{split}$$

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如: 把图片整体放大90%, 那么在线性代数中就是0.9乘单位矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放, 在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中, 就不一样了, 需要大一个维度, 也就是说, 3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

22

S=\leff[\begin{array} {ccc} 0.9 & 0 & 0 \\\ 0 & 0.9 & 0 \\\ 0 & 0 & 1 \\end{array} \right] \$\$

三维立体中图片放大90%就是:

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘 $\$fac{1}{2}\$$,创建一个 $\$fac{1}{4}\$$ 的页边留白,\$x\$方向就要乘 $\$fac{3}{4}\$$,这样得到的缩放矩阵就是:

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样乘: \$vST\$。注意:它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从2×2就变成了3×3矩阵\$R\$。

\$\$
Q=\left[\begin{array} {cc}
\cos \theta & -\sin \theta \\\\\\sin \theta & \cos \theta
\end {array} \right]
\$\$

\$\$

R=\left[\begin{array} {ccc} \\ \cos \theta & -\sin \theta & 0 \\\ \\ \sin \theta & \cos \theta & 0 \\\ 0 & 0 & 1 \\ \end {array} \right] \$\$

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着, 旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

0 & 1 & 0 \\\
4 & 5 & 1 \\end{array}\right]

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{x}=\left\{\begin{array}{l} R_{x}=\left(\begin{array}{l} R_{x}=\left(\begin{array}{l} R_{x}=A_{x}\right) \\ 1 \& 0 \& 0 \& 0 \end{array}\right) \\ 0 \& \cos \theta \& -\sin \theta \& 0 \end{array}\right\} \\ 0 \& \sin \theta \& \cos \theta \& 0 \end{array} \\ 0 \& 0 \& 0 \& 1 \\ 0 \& array \right\} \\ \end{array}$

2.围绕\$y\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{y}=\left| \log \left(\operatorname{array} \left(\operatorname{ccc} \right) \right. \\ \cos \left(\operatorname{a.0.8} \right) \right| \\ 0 \& 1 \& 0 \& 0 \right| \\ -\sin \left(\operatorname{a.0.8} \right) \\ 0 \& 0 \& 0 \& 1 \\ -\operatorname{array} \right| \\ \left[\operatorname{array} \right] \\ \end{array}$

3.围绕\$z\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{z} = \left[\left[\left(\frac{3}{2} \right) \right] \\ \cos \theta & -\sin \theta \\ \cos \theta & 0 \end{array} \right] \\ \sin \theta & \cos \theta & 0 \end{array} \\ \begin{array}{l} 0 & 0 & 0 & 0 \end{array} \\ \begin{array}{l} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ \end{array} \\ \begin{array}{l} \left[\left(\frac{3}{2} \right) \right] & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3}{2} \right) & \cos \theta & \cos \theta \\ \end{array} \\ \begin{array}{l} \left(\frac{3$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

\$\$

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: $\$n^{T}v=0\$$ 。

而投影到平面的投影矩阵是: \$I-m^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量**\$n\$**投影后成为了**0**向量,而平面向量**\$v\$**投影后还是其自身。

```
$$ (I-n n^{T}) n=n-n(n^{T} n)=0 $$ $$ (I-n n^{T}) v=v-n(n^{T} v)=v
```

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$
P=\left[\begin\{array\} \{\!\}
& & & 0 \\\
& I-n n^\{T\} & & 0 \\\
& & & 0 \\\\
0 & 0 & 0 & 1 \\end\{array\}\right]
\$\$

假设现在有一个不过原点的平面, v_{0} \$是这个平面上的一个点,现在要把 v_{0} \$投影到这个平面,则需要经历三个步骤,和刚才介绍的围绕点(4,5)\$,让平面旋转 \hbar 0}\$从theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v {0}\$。

整个过程通过数学公式来表达就是:

\$\$

 $T_{-v_{0}} P T_{+v_{0}} = \left[\left\lceil \frac{\alpha r ay}{cc} I \& 0 \right] \\ -v_{0} \& 1 \\ -v_{0} \& 1 \\ -r_{T} \& 0 \\ -r_{T} & 0 \\$

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里 没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算: 平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的<u>仿射空间</u>中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 $\$v_{0}$, 也就是点 $\$(x_{0},y_{0},z_{0})$ 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的仿射空间。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$和下所示。

\$\$

$$\begin{split} & T \!\!=\!\! \text{leff[begin\{array\} \{IIII\} \\ & 1 \& 0 \& 0 \& 0 \text{ \lambda} \text{\lambda} \\ & 0 \& 1 \& 0 \& 0 \text{ \lambda} \text{\lambda} \\ & 0 \& 0 \& 1 \& 0 \text{ \lambda} \text{\lambda} \\ & x \!\!=\!\! \{0\} \& y \!\!=\!\! \{0\} \& z \!\!=\!\! \{0\} \& 1 \text{ \lend \{array\} \right]} \end{split}$$

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就不一样了,需要大一个维度,也就是说,3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

\$\$

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 \\\\ 0 & 0.9 & 0 \\\\\

```
0 & 0 & 1 \end {array} \right] $$
```

三维立体中图片放大90%就是:

\$\$

$$\begin{split} & S=\left(\left[\left(\frac{1}{2} \right) \right] \right) \\ & S=\left(\frac{1}{2} \right) \\ & S=\left(\frac{$$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘 $\$\sqrt{frac}\{1\}\{2\}$ \$,创建一个 $\$\sqrt{frac}\{1\}\{4\}$ \$的页边留白,\$x\$方向就要乘 $\$\sqrt{frac}\{3\}\{4\}$ \$,这样得到的缩放矩阵就是:

$\mathcal{P}\mathcal{P}$

S=\left[\begin{array} {III} \\frac {3} {4} & 0 & 0 \\\\ 0 & \frac {1} {2} & 0 \\\\ 0 & 0 & 1 \\end {array} \right] \\\$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样乘: \$vST\$。注意:它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从2×2就变成了3×3矩阵\$R\$。

\$\$

Q=\left[\begin{array} {cc} \\cos \theta & -\sin \theta \\\\\sin \theta & \\cos \theta \\\right[array} \right] \$\$

\$\$

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着,旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

\$\$

 $\begin{array}{l} v T_{00} R T_{45} = \left[\left[\left(\frac{3}{2} \right) \right] \\ x \& y \& 1 \\ end_{array} \right] \left[\left(\frac{3}{2} \right) \\ 1 \& 0 \& 0 \\ 0 \\ -4 \& -5 \& 1 \\ \end{array} \right]$

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\$\$

$$\begin{split} R_{x} &= \left[\left(\frac{3}{2} \right) \right] \\ R_{x} &= \frac{3}{2} \\ R_{x} &= \frac{3}{2$$

2.围绕\$y\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{y}=\left| \log \left(\frac{2 }{c \cos } \right) \right| & c \cos \left(\frac{4 }{c \cos } \right) \\ \cos \left(\frac{4 }{c \cos } \right) & c \cos \left(\frac{4 }{c \cos } \right) \\ & c \sin \left(\frac{4 }{c \cos } \right) \\ & c \cos \left(\frac$

3.围绕\$z\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{z}=\left|\left| \left(\frac{2}{\epsilon} \right) \right| \\ R_{z}=\left| \left(\frac{2}{\epsilon} \right) \right| \\ R_{z}=\left(\frac{2}{\epsilon} \right) \\ R_{z$

\$\$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: $$n^{T}v=0$$ 。

而投影到平面的投影矩阵是: \$I-nm^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量\$n\$投影后成为了0向量,而平面向量\$v\$投影后还是其自身。

```
(I-n n^{T}) n=n-n(n^{T}) n=0
$$
$$
(I-n n^{T}) v=v-n(n^{T} v)=v
$$
接下来,我们在齐次坐标中来看一下4×4的投影矩阵:
```

 $P=\left[\left[\left(array \right) \right] \right]$ & & & 0 \\\ & I-n n^{T} & & 0 \\\ & & & 0 \\\ 0 & 0 & 0 & 1 \end{array}\right]

假设现在有一个不过原点的平面,\$v {0}\$是这个平面上的一个点,现在要把\$v {0}\$投影到这个平面,则需要经历三个步骤, 和刚才介绍的围绕点\$(4,5)\$, 让平面旋转\$\theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v {0}\$。

整个过程通过数学公式来表达就是:

 $T_{-v_{0}} P T_{+v_{0}} = \left[\frac{1}{c} \frac{1}{c} \right]$ I & 0 \\\ -v {0} & 1 \end{array}\right]\left[\begin{array} {cc} I-n $n^{T} \& 0 \parallel$ 0 & 1 \end{array}\right]\left[\begin{array} {ll} I & 0 \\\ v {0} & 1 \end{array}\right] \$\$

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要 经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换 到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里 没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学 知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley,这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算: 平移、缩放、旋转和投影的基本概 念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了 这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的<u>仿射空间</u>中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维 世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放:
- 旋转:
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 (x_{0},y_{0}) 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$加下所示。

\$\$

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就不一样了,需要大一个维度,也就是说,3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

\$\$

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 \\\ $0 \& 0.9 \& 0 \parallel \$ 0 & 0 & 1 \end{array}\right]

三维立体中图片放大90%就是:

S=\left[\begin{array} {cccc} 0.9 & 0 & 0 & 0 %0 & 0.9 & 0 & 0 $0 \& 0 \& 0.9 \& 0 \parallel$ 0 & 0 & 0 & 1 \end{array}\right]

\$\$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配, \$y\$方向就要乘\$\frac{1}{2}\$,创建一 个\$\frac{1}{4}\$的页边留白,\$x\$方向就要乘\$\frac{3}{4}\$,这样得到的缩放矩阵就是:

S=\left[\begin{array} {lll} $\frac{3}{4} \& 0 \& 0$ $0 \& \frac{1}{2} \& 0$ 0 & 0 & 1 \end{array}\right]

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘:\$vTS\$,如果我们要先缩放再平移,那应该这样 乘: \$vST\$。注意: 它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从 2×2就变成了3×3矩阵\$R\$。

\$\$

Q=\left[\begin{array} {cc} \cos \theta & -\sin \theta \\\ \sin \theta & \cos \theta \end{array}\right] \$\$

\$\$

R=\left[\begin{array} {ccc} $\cos \theta \& -\sin \theta \& 0$ $\sin \theta & \cos \theta & 0$ 0 & 0 & 1 \end{array}\right] \$\$

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转, 而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着,旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$v\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

22

 $\begin{array}{l} R_{x} = \left[\left[\left(\frac{3}{2} \right) \right] & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 & 0 \\ 0 & \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -\cos \theta & \cos \theta & 0 & 0 \\ \end{array} \right]$

2.围绕\$y\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{y}=\left| \begin{array}{l} R_{y}=\left| \begin{array}{l} R_{y} \end{array} \right| \\ \cos \left| \begin{array}{l} C \end{array} \right| \\ \cos \left| \begin{array}{l} C \end{array} \right| \\ 0 \& 1 \& 0 \& 0 \end{array} \right| \\ 0 \& 1 \& 0 \& 0 \end{array} \right| \\ -\left| \begin{array}{l} C \end{array} \right| \\ 0 \& 0 \& 0 \& 1 \\ \left| \begin{array}{l} C \end{array} \right| \\ \left| C \end{array} \right| \\ \left| \begin{array}{l} C \end{array} \right| \\ \left| C \end{array} \right| \\ \left| \begin{array}{l} C \end{array} \right| \\ \left| C \end{array} \right| \\ \left|$

3.围绕\$z\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{z} = \left[\left| \frac{2}{\epsilon} \right| \\ \cos \left| \frac{2}{\epsilon} \right| \\ \cos \left| \frac{2}{\epsilon} \right| \\ \cos \left| \frac{2}{\epsilon} \right| \\ \sin \left| \frac{2}{\epsilon} \right| \\ \cos \left| \frac{2}{\epsilon}$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: \$n^{T}v=0\$。

而投影到平面的投影矩阵是: \$I-m^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量**\$n\$**投影后成为了**0**向量,而平面向量**\$v\$**投影后还是其自身。

\$\$ (I-n n^{T}) n=n-n(n^{T} n)=0 \$\$ \$\$ (I-n n^{T}) v=v-n(n^{T} v)=v \$\$

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$
P=\left[\begin{array} {III}
& & & 0 \\\
& I-n n^{T} & & 0 \\\
& & & 0 \\\
& & & 0 \\\
o & 0 & 0 & 1
\end{array}\right]
\$\$

假设现在有一个不过原点的平面, v_{0} \$是这个平面上的一个点,现在要把 v_{0} \$投影到这个平面,则需要经历三个步骤,和刚才介绍的围绕点(4,5)\$,让平面旋转 \hbar 0}\$从theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v {0}\$。

整个过程通过数学公式来表达就是:

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算:平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像 处理中的线性变换,比如物体的拉伸和旋转。在第九篇的<u>仿射空间</u>中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 (x_{0},y_{0},z_{0}) 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是 $\mathbf{sf}(\mathbf{a}+\mathbf{b})\mathbf{s}$ 不等于 $\mathbf{sf}(\mathbf{a})+\mathbf{f}(\mathbf{b})\mathbf{s}$ 。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以, $\mathbf{s}\times\mathbf{s}$ 3年降是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$加下所示。

22

T=\left[\begin{array} {IIII}
1 & 0 & 0 & 0 \\\
0 & 1 & 0 & 0 \\\
0 & 0 & 1 & 0 \\\\
x_{0} & y_{0} & z_{0} & 1
\end{array}\right]

平移的整个过程是这样的: 假设要把原来的某个点x,y,z*平移 x_{0} \$,我们需要切换到齐次坐标x,y,z*,然后,x,y,z*,就能得到每个原来的向量x**平移到 $x+v_{0}$ \$的最终结果: x^* \$\left_\text{begin} \text{array} \text{IIII} x & y & z & 1\reft_\text{begin} \text{array} \text{IIII} x+x \ \{0\} & y+y \ \{0\} & z+z \ \{0\} & 1\reft_\text{array} \text{right}\$.

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位 矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就 不一样了, 需要大一个维度, 也就是说, 3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

\$\$

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 $0 \& 0.9 \& 0 \parallel$ 0 & 0 & 1 \end{array}\right]

三维立体中图片放大90%就是:

\$\$

S=\left[\begin{array} {cccc} 0.9 & 0 & 0 & 00 & 0.9 & 0 & 0 $0 \& 0 \& 0.9 \& 0 \parallel$ 0 & 0 & 0 & 1 \end{array}\right]

\$\$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘\$\frac{1}{2}\$,创建一 个\$\frac{1}{4}\$的页边留白, \$x\$方向就要乘\$\frac{3}{4}\$, 这样得到的缩放矩阵就是:

\$\$

 $S=\left\{ \frac{11}{s} \right\}$ \frac{3}{4} & 0 & 0 \\\ $0 \& \frac{1}{2} \& 0$ 0 & 0 & 1 \end{array}\right] \$\$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样 乘: \$vST\$。注意: 它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从 2×2就变成了3×3矩阵\$R\$。

Q=\left[\begin{array} {cc} \cos \theta & -\sin \theta \\\ \sin \theta & \cos \theta \end{array}\right] \$\$

\$\$

R=\left[\begin{array} {ccc} $\cos \theta \& -\sin \theta \& 0$ $\sin \theta \& \cos \theta \ 0$ 0 & 0 & 1 \end{array}\right] \$\$

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转, 而是先平移再旋转,比如我们要围绕点\$(4,5)\$, 让平面旋转\$\theta\$角度的话:

1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;

- 2. 接着, 旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

```
22
```

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\end{array}\right]

ФФ

 $\begin{array}{l} R_{x}=\left[\begin{array}{cccc} R_{x}=\left[\begin{array}{cccc} 1 \& 0 \& 0 \& 0 \end{array}\right] \\ 1 \& 0 \& 0 \& 0 \end{array}\right] \\ 1 \& 0 \& 0 \& 0 \end{array} \\ \begin{array}{l} 0 \& \cos \theta \& 0 \end{array} \\ 0 \& \cos \theta \& 0 \end{array} \\ \begin{array}{l} 0 \& \sin \theta \& \cos \theta \& 0 \end{array} \\ \begin{array}{l} 0 \& 0 \& 0 \& 1 \\ 0 \& 0 \& 0 \& 1 \end{array} \\ \begin{array}{l} 0 \& 0 \& 0 \& 1 \\ 0 \& 0 \& 0 \& 1 \\ \end{array} \\ \begin{array}{l} 0 \& 0 \& 0 \& 0 \& 1 \\ \end{array} \\ \begin{array}{l} 0 \& 0 \& 0 \& 0 \& 1 \\ \end{array}$

2.围绕\$y\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{y} = \left(\frac{3 - 4}{2} \right) & C \leq \\ \cos \theta & C \leq \frac{3 - 4}{2} \\ \cos \theta & C \leq \frac{3 - 4}{2} \\ 0 & 1 & 0 & 0 \\ -\sin \theta & C \leq \frac{3 - 4}{2} \\ 0 & 0 & 0 & 1 \\ -\cos \theta & C \leq \frac{3 - 4}{2} \\ \end{array}$

3.围绕\$z\$轴方向旋转:

\$\$

$$\begin{split} R_{z} = & \left[\left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \right] \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\ & \left(\frac{2}{\epsilon} \right) \left(\frac{2}{\epsilon} \right) \\$$

\$\$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$v\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: $$n^{T}v=0$$ 。

而投影到平面的投影矩阵是: \$I-nm^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量**\$n\$**投影后成为了**0**向量,而平面向量**\$v\$**投影后还是其自身。

\$\$ (I-n n^{T}) n=n-n(n^{T} n)=0 \$\$ \$\$ (I-n n^{T}) v=v-n(n^{T} v)=v

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$
P=\left[\begin{array} {\!!!}
& & & 0 \\\
& I-n n^{T} & & 0 \\\
& & & 0 \\\\
& & & 0 \\\\
& & & 0 \\\\
\tag{array}\right]
\$\$

假设现在有一个不过原点的平面, v_{0} \$是这个平面上的一个点,现在要把 v_{0} \$投影到这个平面,则需要经历三个步骤,和刚才介绍的围绕点(4,5)\$,让平面旋转 \hbar 0}\$从theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v {0}\$。

整个过程通过数学公式来表达就是:

 $\label{eq:continuous} $$ T_{-v_{0}} P T_{+v_{0}}=\left[\left\lceil \frac{2v_{0}}{R}\right] \ Cc} I \& 0 \ Cv_{0} \& 1 \ \left[\frac{2v_{0}}{R}\right] \ Cc} I-n n^{T} \& 0 \ Co} I-n n^{T} \& 0 \ Co} I \ \left[\frac{2v_{0}}{R}\right] \ I \& 0 \ Co} I \ \left[\frac{2v_{0}}{R}\right] \ I \& 0 \ Co} I \ \left[\frac{2v_{0}}{R}\right] \ I \& 0 \ Co} I \ Color \ Colo$

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算: 平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的仿射空间中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维 世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 x_{0},y_{0} 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$和下所示。

\$\$

\$\$

这里很重要的一点是, 计算机图形图像是基于行向量计算的。也就是说, 计算方法是行乘矩阵, 而不是矩阵乘列, 比如:

 $\left[\left(x \in \mathbb{Z} \right) \right] = 1 - 1$

平移的整个过程是这样的: 假设要把原来的某个点\$(x,y,z)\$平移 $$v_{0}$ \$,我们需要切换到齐次坐标\$(x,y,z,1)\$,然后,\$(x,y,z,1)\$再乘\$T\$,就能得到每个原来的向量\$v\$平移到 $$v+v_{0}$ \$的最终结果: $$\left[\text{begin}_{array} \right] T = \left[\text{begin}_{array} \right]$

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就不一样了,需要大一个维度,也就是说,3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

\$\$

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 \\\ 0 & 0.9 & 0 \\\ 0 & 0 & 0 \\\ 1 \end{array} \right] \$\$

三维立体中图片放大90%就是:

\$\$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘 $\$\sqrt{fac}\{1\}\{2\}$ \$,创建一个 $\$\sqrt{fac}\{1\}\{4\}$ \$的页边留白,\$x\$方向就要乘 $\$\sqrt{fac}\{3\}\{4\}$ \$,这样得到的缩放矩阵就是:

\$\$

S=\left[\begin{array} {III} \\frac{3}{4} & 0 & 0 \\\\ 0 & \\frac{1}{2} & 0 \\\\ 0 & 0 & 1 \\end{array} \right] \\\$\$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样乘: \$vST\$。注意:它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从2×2就变成了3×3矩阵\$R\$。

\$\$

Q=\leff[\begin{array} {cc} \\cos \theta & -\sin \theta \\\\\\\\sin \theta & \\cos \theta \\\\\right[{array} \right] \$\$

\$\$

0 & 0 & 1 \end {array} \right] \$\$

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着, 旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

\$\$

 $\begin{array}{l} v T_{00} R T_{45} = \left[\left[\left(\frac{2}{5} \right) \right] \\ x \& y \& 1 \\ \end{array} \right] \\ v d_{array} \right] \left[\left(\frac{2}{5} \right) \right] \\ v d_{array} \left[\left(\frac{2}{5} \right) \right] \\ v d_{array} \right] \left[\left(\frac{2}{5} \right) \right] \\ v d_{array} \left[\left(\frac{2}{5} \right) \right] \\ v d_{ar$

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\end{array}\right]

\$\$

 $\begin{array}{l} R_{x} = \left[\left[\left(\frac{3}{4} \right) \right] \\ R_{x} = \left(\frac{3}{4} \right) \\ R_{x} = \left(\frac{$

2.围绕\$y\$轴方向旋转:

22

 $\begin{array}{l} R_{y}=\left| \begin{array}{c} R_{y}=\left| R_{y}=\left| \begin{array}{c} R_{y}=\left| \begin{array}{c} R_{y}=\left| R_{$

3.围绕\$z\$轴方向旋转:

\$\$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: $$n^{T}v=0$$ 。

而投影到平面的投影矩阵是: \$I-nm^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量**\$n\$**投影后成为了**0**向量,而平面向量**\$v\$**投影后还是其自身。

```
$$
(I-n n^{T}) n=n-n(n^{T} n)=0
$$
$$
(I-n n^{T}) v=v-n(n^{T} v)=v
$$
```

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$
P=\left[\begin\{array\} \{\!\}
& & & 0 \\\
& I-n n^{T} & & 0 \\\
& & & 0 \\\\
& & & 0 \\\\
0 & 0 & 0 & 1 \\end\{array\}\right]
\$\$

假设现在有一个不过原点的平面, v_{0} \$是这个平面上的一个点,现在要把 v_{0} \$投影到这个平面,则需要经历三个步骤,和刚才介绍的围绕点(4,5)\$,让平面旋转 \hbar 0}\$从theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v {0}\$。

整个过程通过数学公式来表达就是:

\$\$ T_{-v_{0}} P T_{+v_{0}}=\left[\left\lceil \frac{ \arctan y}{ cc} I \& 0 \right] \\ I \& 0 \right] \\ -v_{0} \& 1 \\ -v_{0} \& 1 \\ -n n^{T} \& 0 \right] \\ 0 \& 1 \\ -n d_{array} \right] \left[\left\lceil \frac{ \arctan y}{ array} { ll} I \& 0 \right] \\ 0 \& 1 \\ -n d_{array} \right] \left[\left\lceil \frac{ \arctan y}{ array} { ll} I \& 0 \right] \\ I \& 0 \right] \\ -v_{0} \& 1 \\ -v_{0

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换

到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里 没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算:平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的<u>仿射空间</u>中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移;
- 缩放;
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 (x_{0},y_{0},z_{0}) 。

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$

```
1 & 0 & 0 & 0 \\\
0 & 1 & 0 & 0 \\\
0 & 0 & 1 & 0 \\\
x \{0\} \& y \{0\} \& z \{0\} \& 1
\end{array}\right]
$$
```

这里很重要的一点是, 计算机图形图像是基于行向量计算的。也就是说, 计算方法是行乘矩阵, 而不是矩阵乘列, 比如: $\left[\left(x < 0 \ & y < 0 \ & 1 \right) \ & 0 \ & 1 \right] T = \left[\left(x < 0 \right) \ & y < 0 \ & z < 0 \ & 1 \right] S$

平移的整个过程是这样的: 假设要把原来的某个点\$(x,y,z)\$平移\$v {0}\$, 我们需要切换到齐次坐标\$(x,y,z,1)\$, 然 后, \$(x,y,z,1)\$再乘\$T\$, 就能得到每个原来的向量\$v\$平移到\$v+v {0}\$的最终结果: \$\left\begin{array} {III} x & y & z & $1\end{array} T=\left[\left[\frac{3x+x}{0} & y+y \\ 0 \right] & z+z \\ 0 \ & 1\end{array}\right].$

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位 矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就 不一样了,需要大一个维度,也就是说,3×3矩阵变成了4×4矩阵。

比如,二维平面中图片放大90%就是:

S=\left[\begin{array} {ccc} 0.9 & 0 & 0 \\\ $0 \& 0.9 \& 0 \parallel$ 0 & 0 & 1 \end{array}\right] \$\$

三维立体中图片放大90%就是:

\$\$

 $S=\left[\left(\frac{1}{2} \right) \right]$ 0.9 & 0 & 0 & 0 %0 & 0.9 & 0 & 0 \\\ 0 & 0 & 0.9 & 0\\\ 0 & 0 & 0 & 1 \end{array}\right] \$\$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘\$\frac{1}{2}\$,创建一 个 ${\frac{1}{4}}$ 的页边留白,x方向就要乘 ${\frac{3}{4}}$,这样得到的缩放矩阵就是:

\$\$

S=\left[\begin{array} {lll} \frac{3}{4} & 0 & 0 \\\ 0 & \frac{1}{2} & 0 \\\ 0 & 0 & 1 \end{array}\right]

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘:\$vTS\$,如果我们要先缩放再平移,那应该这样 乘: \$vST\$。注意: 它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从 2×2就变成了3×3矩阵\$R\$。

Q=\left[\begin{array} {cc} \cos \theta & -\sin \theta \\\

0 & 0 & 1 \end {array}\right]

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着, 旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

```
$$
```

v T_{00} R T_{45}=\left[\left[\left[\left(\frac{27}{45}\right)\right] \right] x & y & 1 \\ = \left(\frac{27}{45}\right] \left[\left(\frac{27}{45}\right)\right] \\ = \left(\frac{27}{45}\right) \\ = \left(\frac{27}{

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\$\$

R_{x}=\leff[\begin{array} {cccc} 1 & 0 & 0 & 0 \\\ 0 & \cos \theta & -\sin \theta & 0 \\\\ 0 & \sin \theta & \cos \theta & 0 \\\\ 0 & 0 & 0 & 1 \\end{array} \right] \$\$

2.围绕\$y\$轴方向旋转:

\$\$

R_{y}=\left[\begin{array} {cccc} \\cos \theta & 0 & \sin \theta & 0 \\\\ 0 & 1 & 0 & 0 & \\\\ 0 & 0 & 0 & 1 \\\\ end {array} \right] \$\$

3.围绕\$z\$轴方向旋转:

\$\$

 $\begin{array}{l} R_{z} = \left[\left[\left(\frac{3}{2} \right) \right] \\ \cos \left(\frac{4}{2} \right) \\ \cos \left(\frac{4}{2} \right) \\ \cos \left(\frac{4}{2} \right) \\ \sin \left(\frac{4}{2} \right) \\ \cos \left(\frac{$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$v\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下<u>第九篇</u>的内容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等式: $$n^{T}v=0$$ 。

而投影到平面的投影矩阵是: \$I-nm^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量**\$n\$**投影后成为了**0**向量,而平面向量**\$v\$**投影后还是其自身。

\$\$ (I-n n^{T}) n=n-n(n^{T} n)=0 \$\$ \$\$ (I-n n^{T}) v=v-n(n^{T} v)=v

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

\$\$

假设现在有一个不过原点的平面, v_{0} \$是这个平面上的一个点,现在要把 v_{0} \$投影到这个平面,则需要经历三个步骤,和刚才介绍的围绕点 v_{0} \$,让平面旋转 v_{0} \$他在\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v_{0}\$。

整个过程通过数学公式来表达就是:

\$\$

 $T_{-v_{0}} P T_{+v_{0}} = \left[\left\lceil \frac{\alpha ray}{cc} I \& 0 \right] \\ I \& 0 \right] \\ -v_{0} \& 1 \\ -v_{0} \& 1 \\ -r_{1} \& 0 \\ I-n r^{T} \& 0 \\ \\ I-n r^{T} \& 0 \\ \\ 0 \& 1 \\ -r_{1} \& 0 \\ \\ 1 \& 0 \\ \\ v_{0} \& 1 \\ -r_{0} \& 1 \\$

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里 没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley, 这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算: 平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。

你好,我是朱维刚。欢迎你继续跟我学习线性代数,今天我要讲的内容是"如何通过矩阵转换让3D图形显示到二维屏幕上"。

在第八篇的<u>线性映射</u>中,我从二维直角坐标系的角度,讲解了线性映射和变换矩阵。其中,我特别讲到了,二维平面图形图像处理中的线性变换,比如物体的拉伸和旋转。在第九篇的仿射空间中,更是提到了3D的平移矩阵、缩放矩阵和旋转矩阵。

而这一篇则有些不一样,我会从更实践的角度,让你了解到二维平面和三维空间的变换,以及3D图形是如何显示到二维屏幕上的。矩阵在这里扮演的角色可以说是功不可没,接下来我们一起来看下矩阵到底是怎么做到的。

三维空间变换

我们都知道,计算机图形图像处理的是图片,且计算机屏幕是二维的。那你有没有想过,我们在屏幕上看到的静态和动态三维世界到底是怎么回事呢?这个就要涉及到三维到二维的投影技术了,这类技术都离不开矩阵,而且是超大规模矩阵运算。

三维空间的变换依赖于4×4矩阵,可能你会想,为什么不是3×3呢?这是因为四个关键运算中有一个无法用3×3矩阵来完成,其他三个运算为了统一也就都采用4×4矩阵了,这四个关键运算是:

- 平移:
- 缩放:
- 旋转;
- 投影。

平移就是那个无法用 3×3 矩阵来完成的特殊运算,也是看起来最简单的运算,只是每个点都加上向量 v_{0} ,也就是点 x_{0} y_{0} , y_{0

但是,你别被这个假象欺骗了,平移这个运算是非线性的。这一点只需要看平移前各点与原点的连线,以及平移后各点与原点之间的连线就知道了。或者,你也可以从公式的角度理解,就是\$f(a+b)\$不等于\$f(a)+f(b)\$。而为了表示平移,以及现实世界的描述,就需要使用第九篇中说的<u>仿射空间</u>。所以,3×3矩阵是无法平移原点的。

但是,如果我们把原点坐标变成\$(0,0,0,1)\$,那就能解决平移的问题了。点\$(x,y,z)\$的齐次坐标就是\$(x,y,z,1)\$,这就变成了4×4矩阵。接下来,我分别介绍这四个关键运算,它们是3D图形显示在屏幕上的第一步,也就是坐标系变换要做的事情,比如:将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。

平移

我们沿着向量 v_{0} \$平移整个三维空间,把原点平移到了 x_{0},y_{0},z_{0} \$,这也就意味着三维空间的每个点都加上了点 x_{0},y_{0},z_{0} \$。使用齐次坐标,把整个空间平移了 x_{0} \$的4×4矩阵 x_{0} \$加下所示。

\$\$

 $T=\left\{ \begin{array}{l} T=\left\{ \left(\frac{3}{2} \right) \right\} \\ 1 \& 0 \& 0 \& 0 \end{array} \right\} \\ 1 \& 0 \& 0 \& 0 \end{array} \\ 0 \& 1 \& 0 \& 0 \Biggr \\ 0 \& 0 \& 1 \& 0 \Biggr \\ x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& z_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right] \\ \left[\begin{array}{l} x_{\{0\}} \& y_{\{0\}} \& y_{\{0\}} \& 1 \Biggr \\ \end{array} \right]$

这里你需要注意:一个行向量乘T的结果还是一个行向量。

缩放

在前端开发中,我们经常会调整图片宽度和高度来适配页面,比如:把图片整体放大90%,那么在线性代数中就是0.9乘单位矩阵。在二维平面中,我们通常用2×2矩阵来表达缩放,在三维立体中则是3×3矩阵。而在计算机图形图像的齐次坐标中,就不一样了,需要大一个维度,也就是说,3×3矩阵变成了4×4矩阵。

比如, 二维平面中图片放大90%就是:

\$\$

\$\$

三维立体中图片放大90%就是:

\$\$

$$\begin{split} & S=\left[\left[\left(\frac{\operatorname{array}}{\operatorname{cccc}}\right).9 \& 0 \& 0 \& 0 \right]\right] \\ & 0.9 \& 0 \& 0 \& 0 \right]\\ & 0 \& 0.9 \& 0 \& 0 \right]\\ & 0 \& 0 \& 0.9 \& 0 \right]\\ & 0 \& 0 \& 0 \& 1 \\ & \operatorname{array}\left[\operatorname{array}\right] \\ & S \end{split}$$

缩放还可以在不同的方向上进行,比如:一个二维平面图片从整页适配调整到半页适配,\$y\$方向就要乘 $\$fac{1}{2}\$$,创建一个 $\$frac{1}{4}\$$ 的页边留白,\$x\$方向就要乘 $\$frac{3}{4}\$$,这样得到的缩放矩阵就是:

\$\$

S=\left[\begin{array} {III} \\frac{3}{4} & 0 & 0 \\\ 0 & \frac{1}{2} & 0 \\\ 0 & 0 & 1 \\end{array} \right] \\$\$

平移和缩放组合情况会怎样呢?如果我们要先平移再缩放,那应该这样乘: \$vTS\$,如果我们要先缩放再平移,那应该这样乘: \$vST\$。注意:它们乘的顺序是不同的,哪个运算先做就先乘,因为矩阵的左乘和右乘的结果是不同的。

在第九篇的仿射空间中提到了平移和缩放矩阵,你也可以回过头再去看看。

旋转

二维和三维空间的旋转由正交矩阵\$Q\$来完成,它的行列式是+1。同样我们使用齐次坐标,一个平面旋转的正交矩阵\$Q\$就从 2×2 就变成了 3×3 矩阵\$R\$。

\$\$

\$\$

R=\left[\begin\{array\} \{ccc\} \\cos \theta & -\sin \theta & 0 \\\\\sin \theta & \cos \theta & 0 \\\\ 0 & 0 & 1 \\end\{array\}\right]

这个矩阵是围绕原点旋转了平面,那如果矩阵旋转时围绕的不是原点,而是其他点呢?这个就稍微复杂一些,不是直接旋转,而是先平移再旋转,比如我们要围绕点\$(4,5)\$,让平面旋转\$\theta\$角度的话:

- 1. 首先,要把\$(4,5)\$平移到\$(0,0)\$;
- 2. 接着,旋转\$\theta\$角度;
- 3. 最后,再把\$(0,0)\$平移回\$(4,5)\$。

整个过程通过数学公式来表达就是:

\$\$

 $\label{eq:cc} $$v T_{00} R T_{45}=\left[\left[\left(\frac{25}{1 \& 0 \& 0 }\right)\right] \left(\frac{25}{1 \& 0 \& 0 }\right) \right] \left(\frac{25}{1 \& 0 \& 0 }\right) \left(\frac{25}{1 \& 0 \& 0 }\right)$

说完二维我们再来说三维。不过在三维空间中,旋转就有些不一样了,因为它是围绕一个轴"翻转"的。更"数学"的说法就是,围绕\$λ=1\$的特征向量的一条线翻转。

现在,我们来看看分别围绕\$x\$、\$y\$和\$z\$轴方向旋转的矩阵\$R\$有什么不同?

1.围绕\$x\$轴方向旋转:

\$\$

R_{x}=\left[\begin{array} {cccc} 1 & 0 & 0 & 0 \\\ 0 & \cos \theta & -\sin \theta & 0 \\\ 0 & \sin \theta & \cos \theta & 0 \\\ 0 & 0 & 0 & 1 \\end{array} \right] \$\$

2.围绕\$y\$轴方向旋转:

\$\$

 $R_{y}=\left[\left\langle array\right\rangle \left\langle ccc\right\rangle \right]$

```
0 \& 1 \& 0 \& 0 \
-\sin \theta & 0 & \cos \theta & 0 \\\
0 & 0 & 0 & 1
\end{array}\right]
```

3.围绕\$z\$轴方向旋转:

\$\$

 $R \{z\} = \left(\frac{x}{z} \right)$ $\cos \theta \& -\sin \theta \& 0 \& 0$ $\sin \theta \& \cos \theta \ 0 \& 0 \$ 0 & 0 & 1 & 00 & 0 & 0 & 1 \end{array}\right] \$\$

你看出来哪里不同了吗?其实主要就是1的位置不同,以及\$y\$轴方向旋转的\$sin\$互换了。

投影

现在,我们想把3D图形显示到二维屏幕上,该怎么做呢?

从数学角度理解就是把三维向量投影到平面上。在线性代数中,我们看到的大部分的平面都是通过原点的,但在现实生活中则 不是。一个通过原点的平面是一个向量空间,而其他的平面则是仿射空间,具体仿射空间的定义你可以回顾一下第九篇的内 容。

我们先来看看平面通过原点的情况。假设一个通过原点的平面,它的单位法向量是\$n\$,那么平面中的向量\$v\$,满足这个等 式: \$n^{T}v=0\$。

而投影到平面的投影矩阵是: \$I-m^{T}\$。

如果把原来的向量和这个投影矩阵相乘,就能投影这个向量。我们可以用这个投影矩阵来验证一下:单位法向量\$n\$投影后成 为了0向量,而平面向量\$v\$投影后还是其自身。

 $(I-n n^{T}) n=n-n(n^{T}) n=0$

\$\$

\$\$

 $(I-n n^{T}) v=v-n(n^{T} v)=v$

接下来,我们在齐次坐标中来看一下4×4的投影矩阵:

P=\left[\begin{array} {lll} & & & 0 \\\ & I-n n^{T} & & 0 \\\ & & & 0 \\\ 0 & 0 & 0 & 1 \end{array}\right]

\$\$

假设现在有一个不过原点的平面,\$v {0}\$是这个平面上的一个点,现在要把\$v {0}\$投影到这个平面,则需要经历三个步骤, 和刚才介绍的围绕点\$(4,5)\$, 让平面旋转\$\theta\$角度经历的三个步骤类似:

- 1. 把\$v {0}\$平移到原点;
- 2. 沿着\$n\$方向投影;
- 3. 再平移回\$v_{0}\$。

整个过程通过数学公式来表达就是:

 $T_{-v_{0}} P T_{+v_{0}} = \left[\frac{1}{c} \frac{1}{c} \right]$ $I \& 0 \, \text{n}$ -v {0} & 1 \end{array}\right]\left[\begin{array} {cc}

计算机3D图形介绍

有了数学知识的铺垫,我们再来看计算机3D图形显示到二维屏幕上的过程。在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换(又称为空间变换),才能生成二维图像显示在输出设备上。

将一个3D物体显示出来需要经历三个步骤,其中,第一步,也是最重要的一步就是坐标系变换,将局部坐标系表示的点变换到世界坐标系中,然后再变换到视图坐标系(或叫摄像机坐标系),接着继续变换到裁剪坐标系(投影坐标系)。

- 将一个点从局部坐标系变换到世界坐标系是通过平移、缩放及旋转矩阵进行的。
- 如果将世界坐标系中的一个点变换到视图坐标系(摄像机坐标系),则可以使用视图矩阵进行操作。视图矩阵我们这里 没有详细说明,它有个相对复杂的推导过程的,感兴趣的同学可以参考我后面推荐的两本书。
- 如果将视图坐标系(摄像机坐标系)中的一个点变换到裁剪坐标系(投影坐标系),则可以使用投影矩阵进行操作。

最后,我推荐两本非常好的书作为你继续研究计算机3D图形的参考。

《TypeScript图形渲染实战:基于WebGL的3D架构与实现》,作者:步磊峰,这本书描述了3D图形处理的基本数学知识的同时,更注重WebGL框架下的图形渲染实战。

《Computer Graphics: Principles and Practice (3rd Edition)》,作者: Hughes, Van Dam, McGuire, Skylar, Foley, Feiner, Akeley,这本书虽然也有实践,但更偏重计算机图形理论一些。

本节小结

今天的整篇内容都是围绕三维空间的变换展开的,你需要掌握三维空间中的四个关键运算:平移、缩放、旋转和投影的基本概念,以及对应的平移、缩放、旋转和投影矩阵,这些都是继续深入学习计算机3D图形处理的数学基础。

因为在3D环境中,三维物体从取景到屏幕显示,需要经历一系列的坐标变换,才能生成二维图像显示在输出设备上。了解了这些之后,你就能掌握计算机3D图形处理的本质,也许还能在将来的实践中优化图形渲染效率。

线性代数练习场

今天我要给你一道开放题:如果把正方形投影到一个平面上,你会得到一个什么形状的图形?

欢迎在留言区晒出你的结果和思考,我会及时回复。同时,也欢迎你把这篇文章分享给你的朋友,一起讨论、学习。