个人资料

Twinsen

访问： 247752次
积分： 2862
等级： BLOG 5
排名： 第8224名

原创： 19篇 转载： 0篇
译文： 10篇 评论： 376条

weibo

微博

GameDevelope

加关注

GPU programming里面类似float3, half4的数据类型，属于packed array，不同于general-purpose语言的array（比如C，Java），packed array对元素的运行时随机存储低效，甚至不被一些GPU支持。当然general-purpose语言通过SIMD, SSE等指令集

TA 的粉丝 (479) 全部»

Xiaotian

skandhas

Anansi王

芥末师太

最新评论

深入探索透视投影变换(续)
独饮月色的猫: 求教CVV和NDC的区别
深入探索透视投影变换

【专家问答】韦玮：Python基础编程实战专题 【知识库】Swift资源大集合 【公告】博客新皮肤上线啦 CSDN福利第二期

# 深入探索透视投影变换

标签： 图形 算法 编程 游戏 c

2007-09-23 17:18 48659人阅读 评论(115) 收藏 举报

分类： 3D图形固定管线（7）

版权声明：本文为博主原创文章，未经博主允许不得转载。

**深入探索透视投影变换**

**最近更新：2013年11月22日**

-Twinsen编写

-本人水平有限，疏忽错误在所难免，还请各位数学高手、编程高手不吝赐教

-email: popyy@netease.com

透视投影是3D固定流水线的重要组成部分，是将相机空间中的点从视锥体(frustum)变换到规则观察体(Canonical View Volume)中，待裁剪完毕后进行透视除法的行为。在**算法**中它是通过透视矩阵乘法和透视除法两步完成的。

透视投影变换是令很多刚刚进入3D图形领域的开发人员感到迷惑乃至神秘的一个图形技术。其中的理解困难在于步骤繁琐，对一些基础知识过分依赖，一旦对它们中的任何地方感到陌生，立刻导致理解停止不前。

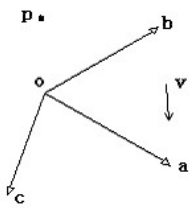
没错，主流的3D APIs如OpenGL、D3D的确把具体的透视投影细节封装起来，比如gluPerspective(…)就可以根据输入生成一个透视投影矩阵。而且在大多数情况下不需要了解具体的内幕算法也可以完成任务。但是你不觉得，如果想要成为一个职业的图形程序员或游戏开发者，就应该真正降伏透视投影这个家伙么？我们先从必需的基础知识着手，一步一步深入下去（这些知识在很多地方可以单独找到，但我从来没有在同一个地方全部找到，但是你现在找到了☺）。

我们首先介绍两个必须掌握的知识。有了它们，我们才不至于在理解透视投影变换的过程中迷失方向（这里会使用到向量几何、矩阵的部分知识，如果你对此不是很熟悉，可以参考《向量几何在游戏编程中的使用》系列文章）。

齐次坐标表示

透视投影变换是在齐次坐标下进行的，而齐次坐标本身就是一个令人迷惑的概念，这里我们先把它理解清楚。

根据《向量几何在游戏编程中的使用6》中关于基的概念。对于一个向量v以及基oabc，




可以找到一组坐标(v1,v2,v3)，使得

v = v1 a + v2 b + v3 c （1）

Twinsen: @mikewolf2007:受教：）
深入探索透视投影变换
mikewolf2007: @popy007:我最近在看clip的文档，所以才关注的，其实硬件就是在范围裁剪的(opengl),...
深入探索透视投影变换
Twinsen: @mikewolf2007:其实我的意思是说是在4d空间中进行裁剪，因为保留了w。但在裁剪算法的实施...
深入探索透视投影变换
mikewolf2007: 硬件进行clip操作是在透视除法之前做的，此时的视锥体xyz范围是，clip之后可能会产生 很多新的...
推导相机变换矩阵
Twinsen: @wanlongwu:左右手系不同。
推导相机变换矩阵
wanlongwu: 楼主：N(相机的Z轴)应该是N = eye-looat,楼主应该写反了。
深入探索透视投影变换(续)
vae4716: 大神你好，拜读了你的文章，受益匪浅，有个问题请教，一幅正常图像中的目标识别和这幅图像的非一致性变换后...
向量几何在游戏编程中的使用4
Twinsen: @u013448456:因为v1'和v2'是共线方向公式，而后面的计算是普遍的不共线方式，我们必须先...
深入探索透视纹理映射（下）
Twinsen: @iwantnon:1）是原始z。2）应该是7楼所说的。这一点我原文需要在修正一下。

**文章搜索**

**文章存档**

2013年05月 (1)
2013年03月 (1)
2013年02月 (1)
2013年01月 (9)
2012年12月 (4)
展开

**文章分类**

2D及3D向量几何图形学 (6)
3D图形固定管线 (8)
后期渲染技术 (0)
设计模式在游戏开发中的使用 (3)
C/C++ (1)
游戏AI (0)
PHP (0)
函数式编程 (6)
内存管理 (0)
多线程 (0)
GPU (4)
ios游戏开发 (1)

**阅读排行**

深入探索透视投影变换 (48644)
深入探索透视投影变换(续 (21259)
推导正交投影变换

而对于一个点p，则可以找到一组坐标（p1,p2,p3），使得

p – o = p1 a + p2 b + p3 c （2）

从上面对向量和点的表达，我们可以看出为了在坐标系中表示一个点（如p），我们把点的位置看作是对这个基的原点o所进行的一个位移，即一个向量——p – o（有的书中把这样的向量叫做位置向量——起始于坐标原点的特殊向量），我们在表达这个向量的同时用等价的方式表达出了点p:

p = o + p1 a + p2 b + p3 c (3)

(1)(3)是坐标系下表达一个向量和点的不同表达方式。这里可以看出，虽然都是用代数分量的形式表达向量和点，但表达一个点比一个向量需要额外的信息。如果我写出一个代数分量表达(1, 4, 7)，谁知道它是个向量还是个点！

我们现在把（1）（3）写成矩阵的形式：

$$\mathbf{v} = (a, b, c, o) \times \begin{pmatrix} v1 \\ v2 \\ v3 \\ 0 \end{pmatrix}$$

$$\mathbf{p} = (a, b, c, o) \times \begin{pmatrix} p1 \\ p2 \\ p3 \\ 1 \end{pmatrix}$$

这里(a,b,c,o)是坐标基矩阵，右边的列向量分别是向量v和点p在基下的坐标。这样，向量和点在同一个基下就有了不同的表达：3D向量的第4个代数分量是0，而3D点的第4个代数分量是1。像这种这种用4个代数分量表示3D几何概念的方式是一种齐次坐标表示。

“齐次坐标表示是计算机图形学的重要手段之一，它既能够用来明确区分向量和点，同时也更易用于进行仿射（线性）几何变换。”—— F.S. Hill, JR

这样，上面的(1, 4, 7)如果写成（1,4,7,0），它就是个向量；如果是(1,4,7,1)，它就是个点。
下面是如何在普通坐标(Ordinary Coordinate)和齐次坐标(Homogeneous Coordinate)之间进行转换：

从普通坐标转换成齐次坐标时，
如果(x,y,z)是个点，则变为(x,y,z,1);
如果(x,y,z)是个向量，则变为(x,y,z,0)

从齐次坐标转换成普通坐标时，
如果是(x,y,z,1)，则知道它是个点，变成(x,y,z);
如果是(x,y,z,0)，则知道它是个向量，仍然变成(x,y,z)

以上是通过齐次坐标来区分向量和点的方式。从中可以思考得知，对于平移T、旋转R、缩放S这3个最常见的仿射变换，平移变换只对于点才有意义，因为普通向量没有位置概念，只有大小和方向，这可以通过下面的式子清楚地看出：

$$\begin{pmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x+Tx \\ y+Ty \\ z+Tz \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 & Tx \\ 0 & 1 & 0 & Ty \\ 0 & 0 & 1 & Tz \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ 0 \end{pmatrix}$$

而旋转和缩放对于向量和点都有意义，你可以用类似上面齐次表示来检测。从中可以看出，齐次坐标用于仿射变换非常方便。

此外，对于一个普通坐标的点P=(Px, Py, Pz)，有对应的一族齐次坐标(wPx, wPy, wPz, w)，其中w不等于零。比如，P(1, 4, 7)的齐次坐标有(1, 4, 7, 1)、（2, 8, 14, 2）、（-0.1, -0.4, -0.7, -0.1）等等。因此，如果把一个点从普通坐标变成齐次坐标，给x,y,z乘上同一个非零数w，然后增加第4个分量w；如果把一个齐次坐标转换成普通坐标，把前三个坐标同时除以第4个坐标，然后去掉第4个分量。

由于齐次坐标使用了4个分量来表达3D概念，使得平移变换可以使用矩阵进行，从而如F.S. Hill, JR所说，仿射（线性）变换的进行更加方便。由于图形硬件已经普遍地支持齐次坐标与矩阵乘法，因此更加促进了齐次坐标使用，使得它似乎成为图形学中的一个标准。

推导相机变换矩阵 (16528)
向量几何在游戏编程中的 (13327)
向量几何在游戏编程中的 (11896)
深入探索透视纹理映射（ (11287)
深入探索3D拾取技术 (11245)
深入探索透视纹理映射（ (11086)
一个基于observer模式的 (10312)
(10037)

评论排行

深入探索透视投影变换 (115)
推导相机变换矩阵 (49)
深入探索透视纹理映射（ (33)
深入探索透视投影变换(续 (31)
一个基于observer模式的 (21)
深入探索3D拾取技术 (19)
向量几何在游戏编程中的 (17)
向量几何在游戏编程中的 (17)
一个多态性的游戏状态机 (16)
向量几何在游戏编程中的 (11)

推荐文章

*Android官方开发文档Training系列课程中文版：网络操作之XML解析
*Delta - 轻量级JavaWeb框架使用文档
*Nginx正反向代理、负载均衡等功能实现配置
* 浅析ZeroMQ工作原理及其特点
*android源码解析（十九）-->Dialog加载绘制流程
*Spring Boot 实践折腾记（三）：三板斧，Spring Boot下使用Mybatis

博客推荐

赖勇浩的编程私伙局

简单的线性插值

这是在图形学中普遍使用的基本技巧，我们在很多地方都会用到，比如2D位图的放大、缩小，Tweening变换，以及我们即将看到的透视投影变换等等。基本思想是：给一个x属于[a, b]，找到y属于[c, d]，使得x与a的距离比上ab长度所得到的比例，等于y与c的距离比上cd长度所得到的比例，用数学表达式描述很容易理解：

$$\frac{x-a}{b-a}=\frac{y-c}{d-c}$$

这样，从a到b的每一个点都与c到d上的唯一一个点对应。有一个x，就可以求得一个y。

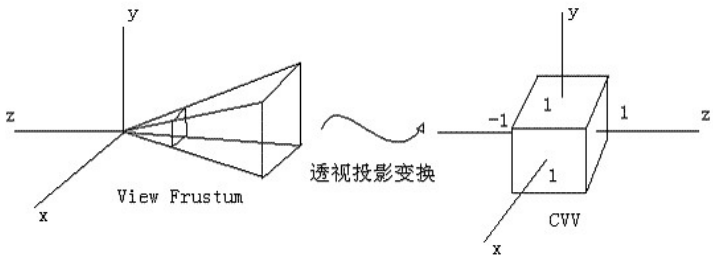
此外，如果x不在[a, b]内，比如x＜a或者x＞b，则得到的y也是符合y＜c或者y＞d，比例仍然不变，插值同样适用。

透视投影变换

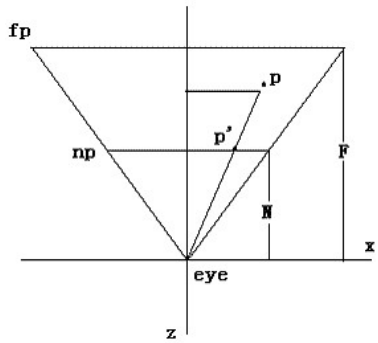
好，有了上面两个理论知识，我们开始分析这次的主角——透视投影变换。这里我们选择OpenGL的透视投影变换进行分析，其他的APIs会存在一些差异，但主体思想是相似的，可以类似地推导。经过相机矩阵的变换，顶点被变换到了相机空间。这个时候的多边形也许会被视锥体裁剪，但在这个不规则的体中进行裁剪并非那么容易的事情，所以经过图形学前辈们的精心分析，裁剪被安排到规则观察体(Canonical View Volume, CVV)中进行，CVV是一个正方体，x, y, z的范围都是[-1，1]，多边形裁剪就是用这个规则体完成的。所以，事实上是透视投影变换由两步组成：

1） 用透视变换矩阵把顶点从视锥体中变换到裁剪空间的CVV中。

2） CVV裁剪完成后进行透视除法（一会进行解释）。




我们一步一步来，我们先从一个方向考察投影关系。




上图是右手坐标系中顶点在相机空间中的情形。设P(x,z)是经过相机变换之后的点，视锥体由eye——眼睛位置，np——近裁剪平面，fp——远裁剪平面组成。N是眼睛到近裁剪平面的距离，F是眼睛到远裁剪平面的距离。投影面可以选择任何平行于近裁剪平面的平面，这里我们选择近裁剪平面作为投影平面。设P'(x',z')是投影之后的点，则有z' = -N。通过相似三角形性质，我们有关系：

$$\frac{x}{x'}=\frac{z}{z'}=\frac{z}{-N}$$

$$x'=-N\frac{x}{z}$$

同理，有

$$y'=-N\frac{y}{z}$$

这样，我们便得到了P投影后的点P'

$$p'=\left(-N\frac{x}{z}\quad -N\frac{y}{z}\quad -N\right)$$

从上面可以看出，投影的结果z'始终等于-N，在投影面上。实际上，z'对于投影后的P'已经没有意义了，这个信息点已经没用了。但对于3D图形管线来说，为了便于进行后面的片元操作，例如z缓冲消隐算法，有必要把投影之前的z保存下来，方便后面使用。因此，我们利用这个没用的信息点存储z，处理成：

$$p'=\left(-N\frac{x}{z}\quad -N\frac{y}{z}\quad z\right)$$

这个形式最大化地使用了3个信息点，达到了最原始的投影变换的目的，但是它太直白了，有一点蛮干的意味，我感觉我们最终的结果不应该是它，你说呢？我们开始结合CVV进行思考，把它写得在数学上更优雅一致，更易于程序处理。假入能够把上面写成这个形式：

$$p' = \left( -N\frac{x}{z} \quad -N\frac{y}{z} \quad -\frac{az+b}{z} \right)$$

那么我们就可以非常方便的用矩阵以及齐次坐标理论来表达投影变换：

$$\begin{pmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} Nx \\ Ny \\ az+b \\ -z \end{pmatrix} \quad \text{齐次坐标变普通坐标} \quad \begin{pmatrix} -Nx/z \\ -Ny/z \\ -(az+b)/z \\ 1 \end{pmatrix}$$

其中

$$P = \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \qquad P' = \begin{pmatrix} -Nx/z \\ -Ny/z \\ -(az+b)/z \\ 1 \end{pmatrix}$$

哈，看到了齐次坐标的使用，这对于你来说已经不陌生了吧？这个新的形式不仅达到了上面原始投影变换的目的，而且使用了齐次坐标理论，使得处理更加规范化。注意在把 $\begin{pmatrix} Nx \\ Ny \\ az+b \\ -z \end{pmatrix}$ 变成 $\begin{pmatrix} -Nx/z \\ -Ny/z \\ -(az+b)/z \\ 1 \end{pmatrix}$ 的一步我们是使用齐次坐标变普通坐标的规则完成的。这一步在透视投影过程中称为透视除法（Perspective Division），这是透视投影变换的第2步，经过这一步，就丢弃了原始的z值（得到了CVV中对应的z值，后面解释），顶点才算完成了投影。而在这两步之间的就是CVV裁剪过程，所以裁剪空间使用的是齐次坐标 $\begin{pmatrix} Nx \\ Ny \\ az+b \\ -z \end{pmatrix}$，主要原因在于透视除法会损失一些必要的信息（如原始z，第4个-z保留的）从而使裁剪变得更加难以处理，这里我们不讨论CVV裁剪的细节，只关注透视投影变换的两步。

矩阵

$$\begin{pmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

就是我们投影矩阵的第一个版本。你一定会问为什么要把z写成

$$-\frac{az+b}{z}$$

有三个原因：

0）后面投影之后的光栅化阶段，要通过x'和y'对z进行线性插值，以求出三角形内部片元的z，进行z缓冲深度测试。在数学上，投影后的x'和y'，与z不是线性关系，与1/z才是线性关系。而 $-\frac{az+b}{z}$ 正是1/z的线性关系，即-a+b/z。用这个1/z的线性组合值和x'、y'进行插值才是正确的。（2013年11月补充条目。对此感到迷惑的读者可以参考《深入探索透视纹理映射》，里面从细节上说明了这个问题。）

1） P'的3个代数分量统一地除以分母-z，易于使用齐次坐标变为普通坐标来完成，使得处理更加一致、高效。

2） 后面的CVV是一个x,y,z的范围都为[-1，1]的规则体，便于进行多边形裁剪。而我们可以适当的选择系数a和b，使得 $-\frac{az+b}{z}$ 这个式子在z = -N的时候值为-1，而在z = -F的时候值为1，从而在z方向上构建CVV。

接下来我们就求出a和b：

$$-\frac{az+b}{z} = \begin{cases} -1, & \text{当} z = -N \\ 1, & \text{当} z = -F \end{cases}$$

$$a = -\frac{F+N}{F-N}$$

$$b = \frac{-2FN}{F-N}$$

这样我们就得到了透视投影矩阵的第一个版本：

$$\begin{pmatrix} N & 0 & 0 & 0 \\ 0 & N & 0 & 0 \\ 0 & 0 & a & b \\ 0 & 0 & -1 & 0 \end{pmatrix} \quad a = -\frac{F+N}{F-N} \quad b = \frac{-2FN}{F-N}$$ 投影矩阵ver1

使用这个版本的透视投影矩阵可以从z方向上构建CVV，但是x和y方向仍然没有限制在[-1,1]中，我们的透视投影矩阵的下一个版本就要解决这个问题。

为了能在x和y方向把顶点从Frustum情形变成CVV情形，我们开始对x和y进行处理。先来观察我们目前得到的最终变换结果：

$$\begin{pmatrix} -Nx/z \\ -Ny/z \\ -(az+b)/z \\ 1 \end{pmatrix}$$

我们知道-Nx / z的有效范围是投影平面的左边界值（记为left）和右边界值（记为right），即[left, right]，-Ny / z则为[bottom, top]。而现在我们想把-Nx / z属于[left, right]映射到x属于[-1, 1]中，-Ny / z属于[bottom, top]映射到y属于[-1, 1]中。你想到了什么？哈，就是我们简单的线性插值，你都已经掌握了！我们解决掉它：

$$\begin{cases} \dfrac{\frac{-Nx}{z} - left}{right - left} = \dfrac{x-(-1)}{1-(-1)} \\ \dfrac{\frac{-Ny}{z} - bottom}{top - bottom} = \dfrac{y-(-1)}{1-(-1)} \end{cases}$$

$$\begin{cases} x = \dfrac{2Nx/-z}{right-left} - \dfrac{right+left}{right-left} \\ y = \dfrac{2Ny/-z}{top-bottom} - \dfrac{top+bottom}{top-bottom} \end{cases}$$

则我们得到了最终的投影点：

$$P' = \begin{pmatrix} \dfrac{2Nx/-z}{right-left} - \dfrac{right+left}{right-left} \\ \dfrac{2Ny/-z}{top-bottom} - \dfrac{top+bottom}{top-bottom} \\ -(az+b)/z \\ 1 \end{pmatrix}$$

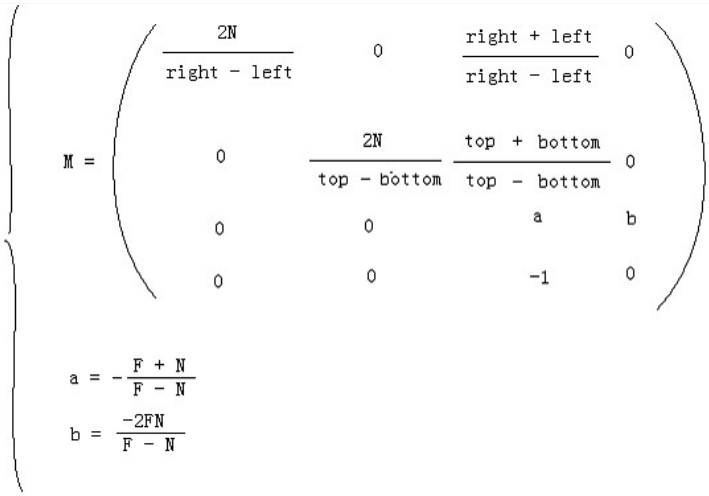
下面要做的就是从这个新形式出发反推出下一个版本的透视投影矩阵。注意到 $\begin{pmatrix} -Nx/z \\ -Ny/z \\ -(az+b)/z \\ 1 \end{pmatrix}$ 是 $\begin{pmatrix} Nx \\ Ny \\ az+b \\ -z \end{pmatrix}$ 经过透视除法的形式，而P'只变化了x和y分量的形式，az+b和-z是不变的，则我们做透视除法的逆处理——给P'每个分量乘上-z，得到

$$\begin{pmatrix} \dfrac{2Nx}{right-left} + \dfrac{right+left}{right-left}z \\ \dfrac{2Ny}{top-bottom} + \dfrac{top+bottom}{top-bottom}z \\ az+b \\ -z \end{pmatrix}$$

而这个结果又是这么来的：

$$M \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \dfrac{2Nx}{right-left} + \dfrac{right+left}{right-left}z \\ \dfrac{2Ny}{top-bottom} + \dfrac{top+bottom}{top-bottom}z \\ az+b \\ -z \end{pmatrix}$$

则我们最终得到：



$$a = -\frac{F + N}{F - N}$$

$$b = \frac{-2FN}{F - N}$$

M就是最终的透视变换矩阵。相机空间中的顶点，如果在视锥体中，则变换后就在CVV中。如果在视锥体外，变换后就在CVV外。而CVV本身的规则性对于多边形的裁剪很有利。OpenGL在构建透视投影矩阵的时候就使用了M的形式。注意到M的最后一行不是(0 0 0 1)而是(0 0 -1 0)，因此可以看出透视变换不是一种仿射变换，它是非线性的。另外一点你可能已经想到，对于投影面来说，它的宽和高大多数情况下不同，即宽高比不为1，比如640/480。而CVV的宽高是相同的，即宽高比永远是1。这就造成了多边形的失真现象，比如一个投影面上的正方形在CVV的面上可能变成了一个长方形。解决这个问题的方法就是在对多变形进行透视变换、裁剪、透视除法之后，在归一化的设备坐标(Normalized Device Coordinates)上进行的视口(viewport)变换中进行校正，它会把归一化的顶点之间按照和投影面上相同的比例变换到视口中，从而解除透视投影变换带来的失真现象。进行校正前提就是要使投影平面的宽高比和视口的宽高比相同。

便利的投影矩阵生成函数

3D APIs都提供了诸如gluPerspective(fov, aspect, near, far)或者D3DXMatrixPerspectiveFovLH(pOut, fovY, Aspect, zn, zf)这样的函数为用户提供快捷的透视矩阵生成方法。我们还是用OpenGL的相应方法来分析它是如何运作的。

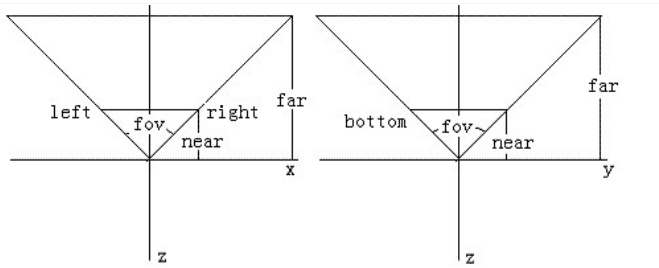
gluPerspective(fov, aspect, near, far)

fov即视野，是视锥体在xz平面或者yz平面的开角角度，具体哪个平面都可以。OpenGL和D3D都使用yz平面。

aspect即投影平面的宽高比。

near是近裁剪平面的距离

far是远裁剪平面的距离。




$$aspect = \frac{right}{top}$$

```
right = near x tan(fov / 2)          top =  near   x   tan(fov/2)
left = -right                        bottom = -top
top = right / aspect                 right = top x aspect
bottom = -top                        left = -right
```

上图中左边是在xz平面计算视锥体，右边是在yz平面计算视锥体。可以看到左边的第3步top = right / aspect使用了除法（图形程序员讨厌的东西），而右边第3步right = top x aspect使用了乘法，这也许就是为什么图形APIs采用yz平面的原因吧！

到目前为止已经完成了对透视投影变换的阐述，我想如果你一直跟着我的思路下来，应该能够对透视投影变换有一个细节层次上的认识。当然，很有可能你已经是一个透视投影变换专家，如果是这样的话，一定给我写信，指出我认识上的不足，我会非常感激☺。Bye!

**深入探索透视投影变换**

顶 10　　踩 0

上一篇　向量几何在游戏编程中的使用6
下一篇　深入探索透视投影变换(续)

## 我的同类文章

**3D图形固定管线（7）**


- 深入探索3D拾取技术 2013-01-07 阅读 11086
- 关于投影平面变换到viewpor… 2012-12-21 阅读 3276
- 深入探索透视纹理映射（下） 2010-05-08 阅读 10312
- 深入探索透视纹理映射（上） 2010-05-04 阅读 11245
- 推导相机变换矩阵 2010-01-02 阅读 13327
- 推导正交投影变换 2009-04-26 阅读 16528
- 深入探索透视投影变换(续) 2009-04-19 阅读 21259


## 参考知识库

**算法与数据结构知识库**

3549 关注 | 4535 收录

猜你在找

| | |
|---|---|
| iOS8开发技术（Swift版）：iOS基础知识 | 深入探索透视投影变换 |
| iOS8-Swift开发教程 | 深入探索透视投影变换 |
| 使用Cocos2d-x 开发3D游戏 | 深入探索透视投影变换 |
| Part 1：基础语言-Cocos2d-x手机游戏开发必备C++语言 | 深入探索透视投影变换 |
| 软件测试基础 | 深入探索透视投影变换 |

别在有线电视上花冤枉钱了

iTalkBB中文电视,热门频道/影视随你点 正版中港台频道，无设备费
www.italkbb.com/TV

**查看评论**

80楼 mikewolf2007 2016-01-23 09:18发表

硬件进行clip操作是在透视除法之前做的，此时的视锥体xyz范围是[-w,w]，clip之后可能会产生 很多新的三角形，之后再进行透视除法，转化到[-1,1]范围，你的文章中一直说裁剪在透视除法之前，又说是在[-1,1]范围裁剪，这个是矛盾的。

Re: Twinsen 2016-02-01 10:25发表

回复mikewolf2007：其实我的意思是说是在4d空间中进行裁剪，因为保留了w。但在裁剪算法的实施过程中，当进行到cvv面和三角形边的相交计算时，会通过暂时的4d变3d过程得到cvv中的值，这个只是暂时的，只是为了得到3d空间中的值计算用。最后完成后进行的透视除法，或者是说在裁剪中最后进行的透视除法，才是最终的透视除法。这个地方我说的确实有些不太清楚，见谅。

Re: mikewolf2007 2016-02-01 14:40发表

回复popy007：我最近在看clip的文档，所以才关注的，其实硬件就是在[-w,w]范围裁剪的(opengl), d3d的 z裁剪范围是在(0,w)范围裁剪。

当然这儿我假设guard band为1,实际上，gpu在内部会给一个guardband值，x，y裁剪方向都要乘以这个系数，从而扩大范围，减少裁剪的可能性，因为裁剪非常耗时。
所以x，y方向裁剪范围是 [- gb*w, gb*w],z方位仍是[-w,w], 当然这会增加扫描转化的时间，但对硬件来说，这是一个tradeoff

Re: Twinsen 2016-02-01 16:00发表

回复mikewolf2007：受教：）

79楼 dancing11 2015-07-09 17:17发表

z-buffer中的值怎么计算出来的，可以讲讲吗？

Re: Twinsen 2015-07-10 10:22发表

回复dancing11：zbuffer中的值就是深度缓存中存储的z值，也就是-（za+b）/z。

78楼 dancing11 2015-07-09 17:09发表

初学者，受益匪浅，感谢楼主的倾心奉献。想问下深度缓存中存储的Z值是-（za+b）/z吗？

77楼 nshen121 2014-10-21 12:28发表

小弟把Flash的Stage3D投影矩阵全推导了一遍，各位flasher请参考： http://www.nshen.net/article/2014-10-16/stage3d-projection-matrix/

76楼 yangfan_Lxy 2014-09-16 19:45发表

仔细看了一遍，很有收获，博主搞计算机图形学很多，能不能推荐一些国外的经典3D计算机图形学的书籍，看书+看博客效率很高

Re: Twinsen 2014-09-17 09:27发表

回复yangfan_Lxy：《计算机图形学opengl版》，作者是Hill。强烈推荐。

75楼 rentongdu 2014-07-04 10:59发表

所以投影变换就是把3D以某种方式(透视,正投影)投影到2D平面上(第3个分量没有用,但是仍然利用它来保存一些信息)?

74楼 z1050334 2014-05-22 14:36发表

http://www.web-tinker.com/article/20157.html
以前看过一篇文章，对投影矩阵有些了解，得益于博主，今天又有了更进一步的理解。thanks.

73楼 xhhusthdq 2014-04-13 22:37发表

经过这样的变换，原来的坐标的Z信息就没有了，如果我想根据屏幕上面的坐标反求对应的点的原始坐标，那该如何呢？我觉得z-buffer里面保存的z信息应该就是视图坐标系下的z，这样就可以了吧？

Re: Twinsen 2014-04-14 09:26发表

回复xhhusthdq：反求是不能的，perspective division已经改变了原始的z，并且第四个分量已经变为1了，原始信息都没了。division之后的那个z是cvv中的z，它在闭区间[-1,1]中，不是相机空间的z。

72楼 netbaixc 2014-03-23 15:28发表

感谢楼主先！
楼主，“裁剪被安排到规则观察体(Canonical View Volume, CVV)中进行”。这话本身容易引起误解。
在经过透视变换后得到的4D齐次坐标，已经完全可以进行裁剪了。无非就是怎么裁剪的算法问题。因为本来这个4D齐次坐标就是在后面透视除法后一定会变成CVV的NDC，那么只要基于这个关系设计算法就行了。但是这个裁剪算法本身一定会用透视除法甚至会用除法吗？没有必要。
反过来讲，当初设计透视变换矩阵时就揉进了CVV的概念，得到的4D齐次坐标就有CVV的影子，所以完全可以说“裁剪算法是基于满足了CVV要求的透视转换而设计的”。

Re: Twinsen 2014-03-24 17:29发表

回复netbaixc：你理解、表达得很到位。很多朋友都在这个地方有理解偏颇，应是我表达的问题。

71楼 multiapple 2014-03-04 11:32发表

楼主大大，我还想问几个渲染管线的问题0.0
1.裁剪被安排到规则观察体(Canonical View Volume, CVV)中进行，CVV是一个正方体，x, y, z的范围都是[-1，1]，多边形裁剪就是用这个规则体完成的。（当完成透视矩阵的乘法后，cvv的坐标范围好像不是【-1,1】吧，所以我之前认为裁剪是发生在透视除法之后，因为按照这段话的理解此时cvv的坐标已经是【-1,1】了）
2.裁剪是api完成的还是需要人工干预（我只了解过固定管线，对可编程管线不了解），裁剪之前图元已经装配（索引缓存的使用）了吗？
3.光栅化是由硬件完成的，程序员没法干预吗？
4.视口，帧缓存，窗口大小的关系是什么？
初学者，还请楼主多多关照。

Re: Twinsen 2014-03-08 17:29发表

回复multiapple：楼主大大，我还想问几个渲染管线的问题0.0
1.这个地方很多读者都比较迷惑。裁剪是在4D的齐次坐标下进行的，还没有进行透视除法。但在裁剪的过程中可以用除法得到cvv中的值以利用。裁剪完毕后，才进行真正的除法变成3D形式。
2.裁剪是自动完成的，不能干预，可编程管线也不行！裁剪之前图元已经装配了，否则哪来的三角形拓扑信息？没有这些信息，就无法确定三角形数据。
3.光栅化是由硬件完成的，程序员没法干预。pixel shader得到的是光栅化过程中的每个fragment。
4.视口使用窗口坐标系，就是用户能看到的窗口中指定的区域。帧缓存就是存储片元数据的多种缓存的集合：比如颜色缓存，depth/stencil缓存，等等。一个屏幕窗口可以包含一到多个视口。

70楼 multiapple 2014-03-03 20:11发表

请教文主几个问题：
1.裁剪是发生在原始投影之后，cvv之前的吗？如果是这样，那cvv的作用就没了啊;如果不是，那么cvv之后的原始z信息丢失后怎么完成裁剪啊？
2.透视投影完了之后，顶点数据是不是就是2D的了？然后根据索引缓存把一个个的三角形图元送进光栅单元？
3.经过透视投影后，顶点的z信息不是丢失了吗（透视除法）？
那么后期的z——buff测试怎么进行？
希望文主指教 TKS

Re: Twinsen 2014-03-03 22:06发表

回复multiapple：1.裁剪发生在透视变换之后，透视除法之前。这个时候顶点的坐标是4D的齐次形式，用透视除法可以得到CVV中的NDC坐标。而不用除法的话，第四个分量保存了一部分原始z的信息。
2.透视投影完了之后，顶点数据还是3D的，但是已经是单位话的设备坐标了，也就是NDC。三个分量的范围都在[-1,1]（opengl），也就是CVV中。这个时候进行viewport(VP)变换就可以进入VP中了，变换之后在VP中进行光栅化。
3.经过透视投影后，顶点的原始z信息丢了，但得到了[-1,1]中的z，顶点之间的z顺序不会有变，不会影响通过NDC的z光栅化得到的fragment的z进行depth test。

69楼 i_dovelemon 2013-09-19 00:58发表

看了Introduction to 3D Game Programming with DirectX 9.0 : A shade Approach ,Real-Time rendering 还有3D 游戏编程大师技巧，都没有搞得太明白这个问题，楼主神人啊。。。通俗易懂，终于搞清楚一点，原来Normalize Device Coordinate坐标就是所谓的CVV啊。我就是不明白Luna为什么要转化到[-1,1]这个坐标来，原来是为了以后裁剪方便啊！！！大大的解惑。非常感谢楼主的文章！！！希望楼主能够继续推出这样的文章。

对了，还想问下：如果要熟练的对这些矩阵变换掌握，能够到达自己编写3D引擎的程度，应该看下什么方面的数学知识？？？

Re: Twinsen 2013-09-23 10:06发表

回复i_dovelemon：如果想自己写3D引擎，以下几个应该要掌握：
1. 向量（矢量）
2. 矩阵
3. SGI经典流水线的每个细节。
我的这些文章基本上都涵盖了：）

68楼 superleg_2008 2013-09-15 09:24发表

很不错的文章，受益匪浅，谢谢作者的分享！
我有两个问题。
1. 投影面跟近裁剪面是同一个面吗？

2. 一般流程是在得到3d物体的投影坐标后，再转换到cvv坐标，最后再转成屏幕坐标。但是整个推导只是得到了将投影坐标转成cvv坐标的矩阵，最后将cvv坐标转成屏幕坐标需要做哪些工作？

Re: Twinsen 2013-09-18 15:17发表

回复superleg_2008：1. 投影平面可以是平行于前裁剪平面的任何平面。只要fov固定，投影得到的结果都是一样的。
2. 将cvv转换到视口坐标，就是把[-1,1]转换到[0, viewportWidth]，做个简单的线性插值就可以了。

Re: superleg_2008 2013-09-19 14:02发表

回复popy007：谢谢Twinsen的解释，受教了！

67楼 wowo5050 2013-08-29 22:35发表

受教了，继续研究！

66楼 xiaoxia19920920 2013-03-18 23:01发表

mark

65楼 lookupheaven 2013-03-18 16:27发表

看到这篇文章的心情，只能说是狂喜

Re: Twinsen 2013-03-18 16:41发表

回复Razor87：呵呵。过奖了。

64楼 oLiZhi123456789 2013-01-26 14:47发表

我用线性插值写了一个软件模拟zbuffer的。在画三角形时，对屏幕坐标进行插值，同时对z进行插值。得到的z不是很正确，用公式也能看出不是很正确。但貌似这种方法又是一个比较快速的方式，虽然不正确，但还勉强可以接受。
http://wonderfl.net/c/emla

Re: Twinsen 2013-01-26 15:07发表

回复oLiZhi123456789：计算出的数据是错误的，但用来做z排序"看起来正确"。因为z都错了，但前后关系基本保持正确。这就是它比仿射纹理映射更能被接受的原因，因为纹理可以看出来，而z"看不出来"，只做排序用。

63楼 Eran 2013-01-26 13:38发表

首先感谢Twinsen，帮了我很多，谢谢。
http://www.cnblogs.com/eran/articles/Stage3D_3.html
我自己也写了一篇小文章，阐述了一下我对透视投影矩阵的理解，期间也多次引用到了Twinsen的文章，如果有人看了Twinsen的文章不是很明白，可以看一下我的。也许会有些收获。

Re: Twinsen 2013-01-26 14:08发表

回复iamzealotwang：很棒！已经在我的weibo里面进行了分享：）http://weibo.com/panhong101

62楼 dlmult 2013-01-11 18:12发表

留个记号，向楼主学习。

61楼 Eran 2012-12-18 13:38发表

Hi Twinsen 有个问题想问一下你,我目前正在看Andre LaMothe的3D游戏编程大师,在书中讲到的投影矩阵和你最后推导出来的矩阵相差很大
(P311页,其矩阵为
(d,0,0,0),
(0,d,0,0),
(0,0,1,1),
(0,0,0,0)),
我个人认为存在差别的原因有几个:
1`书中的矩阵乘法和你推导的是相反的(你推导的是矩阵x坐标点,书中的是坐标点x矩阵)。
2`关于x,y的投影,书中采用的是特殊形式(两个角度均为90度,d值为1,也就是你用到的-N值[那块也不太理解,为何为-N?]),而你推导的是一般形式
3`书中没有涉及到z值的投影,这块就我后续的一些了解,需要将z转换为1/z,因为1/z才是成线性的,用于后续的剪裁,z缓冲,纹理映射(目前不太懂这块)
我想问的是
1·关于前两点我说的是否正确？
2·第三点如果为了转换为1/z,为何为(az+b)/z=a+(1/z)b的形式？,我看你后来给silent2088的解释为如果为(z+b)/z=1+(1/z)*b将没有唯一的解,这块不是很理解,另外我还是不太理解为何不能直接存储1/z？
我需要先看看哪方面的知识才能理解么？
谢谢

Re: Twinsen 2012-12-18 14:00发表

回复iamzealotwang：恩，那本书的推导我很久之前也研究过。在这本书中，作者去掉了cvv裁剪的部分，因此得到了更简单一些的投影矩阵。

1 你的前两点说的没错。还有一点就是他好像把视口变幻也揉进了这个矩阵。另外，作者去掉了cvv裁剪这个步骤，使用了更简单的提出方式。
2 其实用1/z已经可以了，但这也是去掉了cvv裁剪的情况。至于我为什么说是(az+b)/z=a+(1/z)b这样的形式，是因为：z和z'也非线性关系，和透视纹理映射一样，是z'=a(1/z)+b的关系，也就是说z'和1/z是线性关系，也就是z'=(az+b)/z的关系（线性关系的一般形式是y=ax+b，其中x和y是变量,a和b是系数）。这是我在研究透视纹理映射的时候学到的。具体的推论可以参考我的透视纹理映射的文章；）

60楼 jxtgddlt 2012-11-16 12:51发表

豁然开朗 谢谢了！

59楼 through_life 2012-09-04 17:12发表

写的真棒，学到很多！
不过在技术a,b值的时候，我有些不同意见。
我觉得 (-az+b) / z 这个公式，在 z = -N是，值为1，而不是-1
在 z = -F时，为-1，所以我计算出来的结果是：
a = (N+F) / (F-N), b = (-2NF)/(F-N)。所以下面的矩阵亦如此。
不知我的想法是否正确？

Re: through_life 2012-09-05 11:29发表

回复through_life：不好意思，又算了一遍，发现我的结果：a = (N+F) / (F-N), b = (-2NF)/(F-N)
是错的，您的结果：
a = -(N+F) / (F-N), b = (-2NF)/(F-N)
是对的。

58楼 flyflyking 2012-08-05 16:28发表

谢谢，慢慢看，顶了再看

57楼 blogyuantg 2011-08-19 10:12发表

谢谢楼主知识分享的精神谢谢了！

56楼 sandwich007 2011-05-31 16:12发表

多谢讲解，

55楼 sunningsphere 2011-04-20 14:41发表

很感谢大哥的精彩讲解！

54楼 tdl1001 2011-04-19 09:43发表

最容易理解的讲法，很强大，终于了解了

53楼 mayingzhen 2011-04-18 18:45发表

[e01]
终于明白透视变换了，谢谢分享啊。。。

52楼 赵勇文 2011-03-16 16:00发表

[e01]

51楼 hpain 2011-01-26 11:25发表

看了几大本书，都对透视投影一知半解，这下彻底明白了。it行业绝对需要像楼主这样愿意分享知识的人，严重鄙视那些自己懂却不透露的人[e03]

50楼 weihuai1 2010-11-18 21:36发表

[e03]写的很好,对我帮助很大!

49楼 c1a9o6h6o4n0g 2010-11-10 20:25发表

高人哪，以后多多请指教

48楼 zatman 2010-08-21 21:20发表

[e03]谢谢，恍然大悟。

47楼 Twinsen 2010-03-11 13:17发表

silent2088，我已经加你了！

46楼 silent2088 2010-03-11 11:39发表

我应向你多多指教!
我怎么加不了你为好友啊！

45楼 Twinsen 2010-03-09 11:25发表

在shader下的几何阶段，来自用户的矩阵基本上包括：模型变换矩阵，观察矩阵，透视矩阵这三个（前两个可以合并）。其中只有最后一个矩阵和裁剪有关。裁剪程序会使用CVV对透视变换之后的图元进行裁剪，用户只能够通过设置透视矩阵来改变变成CVV之前的视锥体，对CVV本身没有插手的余地，CVV是固定的。最后程序会用CVV来对顶点组成的图元进行裁剪操作。裁剪信息只有两类：裁剪体CVV（固定的东西）以及图元顶点们。

44楼 CxxlMan 2010-03-09 11:14发表

我的意思是用戶在 shader 下使用自訂的矩陣做頂點轉換,但卻不須要告訴 d3d 或 OpenGL,顯然裁剪的訊息不須直接來自用戶自訂的矩陣,因此我猜測 d3d 和 OpenGL 是利用轉換後的頂點資料拼湊出裁剪資訊,因這些頂點是唯一能得到訊息的依據,你看法如何

43楼 Twinsen 2010-03-09 09:47发表

如果在shader下没有指定透视矩阵的相关参数（fov, aspect, near, far），则d3d或者ogl会使用默认的透视矩阵进行操作。具体的默认值可以参考d3d或者ogl的相关手册。

42楼 CxxlMan 2010-03-09 03:02发表

現在的情況是在 shader 下用戶自定的矩陣並未告知 D3D 和 OpenGL,而裁剪須要有足夠的訊息,這些訊息要從哪取得呢

41楼 Twinsen 2010-03-06 16:01发表

CxxlMan，-(za+b)/z=-1的时候，z是等于-N的（OpenGL环境）。非shader时，D3D会通过用户提供的参数生成透视投影矩阵。shader下需要用户自己定制这个矩阵，具体的矩阵形式可以参考《深入探索透视投影变换（续）》中关于d3d相应矩阵的推导，如果对推导过程不感兴趣，可以直接使用结论。

40楼 CxxlMan 2010-03-06 08:25发表

要由 -(za+b)/z 得到近裁剪面的 z 值，須設 -(za+b)/z = -1，經轉化後 z = -(b / (a-1))，但這是已知矩陣的情況下，若是使用 Shader 的方式來編寫 D3D 的程式，D3D 應不知矩陣的內容吧？只知一系列轉換前的頂點(x,x,z,1)和一系列轉換後的頂點(x',x',z',w)，所以須由兩個轉換後的頂點中的 z' 值(za+b)求 a 和 b 的解，D3D 是不是這樣做的呢？

39楼 Twinsen 2010-03-02 21:45发表

对，-(za+b)/z就是原始z在CVV空间中的对应物，范围是[-1, 1]。

38楼 silent2088 2010-03-02 18:55发表

-(za+b)/z，不是投影之后的z，投影之后的z已经在-N平面上了，这个z是原始z在CVV空间中的对应物。
上面是引用你的话，我想说：“这个z是原始z在CVV空间中的对应物。”在这一句中的z是指-(za+b)/z这个嘛？

37楼 Twinsen 2010-03-02 15:19发表

这样得到的z会存在精度问题。越靠近-N的原始z在CVV中的z的精度范围越高。提高整体精度的方法就增大N。这一点可以通过考察方程组进行分析，CVV的z范围曲线在很多图形学文献中都有展现。

36楼 Twinsen 2010-03-02 15:14发表

silent2088，你说的没错！-(za+b)/z，不是投影之后的z，投影之后的z已经在-N平面上了，这个z是原始z在CVV空间中的对应物，它的范围会在[-1, 1]，用来在光栅化阶段进行片元的z缓冲测试。请注意，虽然原始z在范围[-N, -F]对应新z在范围[-1， 1]，这个变化却不是线性的。因为新z的计算并不是对原始z进行简单线性插值得到的，而是通过透视矩阵ver1上面的非线性方程组得到的。

35楼 Twinsen 2010-03-02 09:35发表

silent2088，za+b是一种一元一次的线性形式，可以用矩阵非常容易的处理，你可以看透视矩阵版本一的a和b。如果写成二次或者以上的形式就无法用矩阵简单表示，方程组解起来也很困难。如果写成za，会和下面的z抵销掉。如果写成z+b，关于z的方程组无唯一解。因此使用上述形式。

Re: silent2088 2010-03-02 14:16发表

回复 popy007：你刚开始说投影的结果z'始终等于-N，在投影面上。实际上，z'对于投影后的P'已经没有意义了，这个信息点已经没用了，利用这个没用的信息点存储z，那你为什么突然变到(za+b)/z？
我想是不是如下这样的(你帮忙看下对不对)
最后得到的z'你说是在归一化的坐标系中的cvv的值，然后通过比较最后变化后的z'值来进行z深度测试的吗？

Re: Twinsen 2010-05-04 14:26发表

回复 silent2088：

D3DXMatrixPerspectiveLH()调用的时候并没有进行实际的透视投影，只是设置了当前的透视投影矩阵。透视投影是在用API渲染多边形的时候在流水线内部自动进行的，也包括透视除法。就算用shader，有几个地方也是无法插手的：裁剪、透视除法、视口变换以及光栅化。

34楼 silent2088 2010-03-01 17:35发表

我弄不懂你那个为什么要把z写成(za+b)/z,你说了两个原因，第一个我了解了，第二个，你说为了方便进行多边形剪裁，怎么方便啦！ 为什么是za+b呢？你为什么不写成其它的形式呢？

Re: zjghs2007 2011-12-07 16:57发表

回复silent2088：回复silent2088：
已知： z（点p在z轴上的值）
目标： z'（深度信息 -1~+1）
原始问题就转化为，怎么在已知点p的z坐标的情况下，求得用在深度测试中的z'
透视投影中z'与z之间为非线性关系 —— z'z = az +b
（PS：正交投影中z'与z为线性关系—— z' = az + b）
那你肯定要问： (1) 为什么正交投影中z'和z为线性关系 (2)为什么透视投影不能和正交投影一样都采用线性关系
如果理解了这两个问题，那么就解决你的疑问了~~~
关于(1): 正交投影的实质就是 将Frustum的中心先平移到eye坐标系原点，然后再缩放至CVV。x，y，z三轴均是如此，而对z平移和缩放不就是 z' = az + b吗？为线性关系 ^_^解决
关于(2): 透视投影没有这么简单粗暴，x'和y'不是简单的平移和缩放就能搞定，x' = -Nx/z; y'= -Ny/z，即 x'和x，y'和y不是线性关系，均与z有关。但是注意，此时z'和z仍可以写成线性关系z' = az+b，因为透视投影跟正交投影不同的地方只是影响x'，y'的计算，那为什么z'也要采取z'z = az+b的非线性关系呢？因为，要少数服从多数啊~~~ ^_^

如果现在你还没有清楚，那是正常的，因为还有一个关键的东西需要理解！
重要： 矩阵*向量 只能模拟线性关系的处理 @###@
所以对于 x' = -Nx/z; y'= -Ny/z 必须要先用 矩阵*向量 得到线性关系 x' = -Nx; y'= -Ny，然后同时除以z才能得到最终结果 x' = -Nx/z; y'= -Ny/z。
记住x，y，z三个轴地位等价，所以处理过程肯定是相同的，对z而言，也需要先用 矩阵*向量 得到线性关系 z' = az+b （公式1），然后再除以z得到最终结果 z' = a + b/z， 即得到z'和z的非线性关系式 z'z = az + b
现在又有问题了，凭什么 公式1 要这样写，线性关系 z'= az 或者z' = b为什么不行？ 当然啦。要保证方程组 z=N时，z'=-1；z=F时，z'=1有解就必须要 z' = a + b/z。 z'= az 和z' = b均是无解！ ^_^ 解决

不用谢我哈~~~

Re: multiapple 2014-03-06 09:17发表

回复zjghs2007：不好意思 打扰下 你说的
【矩阵*向量 只能模拟线性关系的处理】
那透视矩阵为什么能把平头椎（视见体）转换成立方体（齐次裁剪空间）呢？这应该不是线性关系吧》？。。。求解释

Re: Twinsen 2011-12-08 09:53发表

回复zjghs2007：恩。 zjghs2007解释的非常详细，很好！其实除了这些方面，还有一个重要的点，也是我后来研究透视纹理映射才得到的结论。那就是，z和z'也非线性关系，和透视纹理映射一样，是z'=a(1/z)+b的关系，也就是说z'和1/z是线性关系。因此就是z'=(az+b)/z的关系。具体的推论可以参考我的透视纹理映射的文章；）

33楼 Twinsen 2010-02-28 10:12发表

这个也是目前图形API标准流水线发展的结果。dos年代图形程序员使用的透视投影矩阵大都没有这么复杂，都是很原始、直接的投影目的，就像上面的透视矩阵版本一，甚至都没有将z分量进行规范化。大多数的流水线还是自己定制的，因为没有硬件加速，所以使用软件渲染。

32楼 CxxlMan 2010-02-27 23:55发表

因為一般討論轉換矩陣和裁剪的關係著墨實在太少了，直覺上認為轉換矩陣的功能就只在空間變換和投影上，我也不是沒想過在透視除法前做裁剪，但以為要另外做處理，須另外提供裁剪邊界相關訊息，沒想到裁剪平面的訊息就隱藏在矩陣中

Re: tjwhs 2012-08-20 14:29发表

回复CxxlMan：个人认为为什么在透视除法前做裁剪：
1:透视除法前能做裁剪：做法：
1.1：判定阶段：如果-w<x<w,-w<y<w,-w<z<w。那么这一点就在CVV中。
1.2：计算阶段：如果线段AB需呀裁剪，计算新点：
t=-（Xa+Wa）/((Xb+Wb)-(Xa+Wa))(对于-w裁剪)
Xnew=Xa+t(Xb-Xa).
所以在除法前完全能做。
2：为什么在除法前做：因为除法是很费时的，在除法前剔除一些顶点，对于除法来说能减少不少时间。
综上所述：既然裁剪在除法之前能做，又能节约时间，那为什么不在除法之前做呢？？？对吧。。

31楼 Twinsen 2010-02-26 16:52发表

但是，实际上的裁剪使用的形式是透视除法之前的形式，也就是(x, y, z, w)的形式，使用这样一个形式的好处是，我们可以在裁剪的时候使用w分量进行某些工作，而且也可以在裁剪过程中进行临时的透视除法得到CVV中的坐标，一举两得。最后经过了裁剪，就进行真正的透视除法，得到了CVV中的物体坐标。这样讲是不是清楚一些了？

Re: silent2088 2010-05-04 13:54发表

回复 popy007：你好！ 我实现了阴影之后，对这个东西还有一定的理解了， 我还想问下楼主，你上面所说到"其中z/w得到的就是NDC中的深度值，用来做z缓冲深度测试" 我用的是DX，我想问的是它不是有函数吗？进行透视投影的DX中是D3DXMatrixPerspectiveLH ,它是在这个函数执行的时候就进行透视除法吧！ 我看到的有些关于阴影的例子，它们怎么要自己去执行z/w啊？

Re: silent2088 2010-05-04 14:24发表

回复 silent2088： 噢！ 我差不多想明白了！[e04]

Re: Twinsen 2010-05-04 14:27发表

回复 silent2088：
自己作透视除法的情况基本上都是希望自己控制这几个shader不能编程的阶段。或者是要把一个顶点不通过流水线而是自己变换到屏幕上。比如一些特殊UI，或者做2D的拾取等等。

Re: Twinsen 2010-05-04 14:27发表

回复 silent2088：

D3DXMatrixPerspectiveLH()调用的时候并没有进行实际的透视投影，只是设置了当前的透视投影矩阵。透视投影是在用API渲染多边形的时候在流水线内部自动进行的，也包括透视除法。就算用shader，有几个地方也是无法插手的：裁剪、透视除法、视口变换以及光栅化。

Re: silent2088 2010-05-04 15:06发表

回复 popy007：嗯！明白了！将相机坐标系中的(x,y,z)实际上是转换到cvv中的坐标吧！像z值应该是离相机的距离值，乘完透视矩阵后-（za+b）/z实际就是在cvv中的值,如果近平面和远平面的值分别是1和1000，那么乘完后的值就是在这1~1000之间的值，就是这样进行z缓冲消隐的吧！

Re: Twinsen 2010-05-04 15:39发表

回复 silent2088：
在相机空间中的近远裁剪平面不管是多少，变换到CVV中之后都是[-1, 1]（OpenGL）。

Re: silent2088 2010-05-04 15:57发表

回复 popy007： 噢！ 乘出来以后它都是在-1~1之间的啊！

Re: Twinsen 2010-05-04 16:03发表

回复 silent2088：
对，没错！

30楼 Twinsen 2010-02-26 16:52发表

CxxlMan，我大概知道你在哪个地方的理解和我不一样了。你可能觉得，在透视除法之前，还没有得到CVV，怎么可能进行CVV裁剪呢？对吧？其实，顶点经过了透视变换之后，变成了齐次形式(x, y, z, w)，虽然还没有进行透视除法，但是已经具备了透视除法的条件了，已经可以做透视除法了。而且一旦进行了透视除法(x/w, y/w, z/w, w/w)，得到的(x', y', z')就在CVV所在空间中了，可以针对CVV对图元进行裁剪了，这个是比较合理的思路。

29楼 CxxlMan 2010-02-26 16:30发表

依你本文的推論，的確可以在透視除法前先為 近 和 遠先做裁剪，但上下左右還是得做完透視除法再裁剪是吧，所以 D3D 是分兩個階段做裁剪是嗎

28楼 Twinsen 2010-02-26 13:16发表

：）对于裁剪来说，就是CVV阿，xyz都在[-1, 1]的范围。这个是透视变换的目的之一阿。

27楼 CxxlMan 2010-02-26 09:06发表

哈哈! 我懂了, 只要將 -(az+b)/z 指定等於 -1,就可以倒推出近裁剪平面的 z 位置,腦筋一時沒轉過來 = =" 很感謝你的指導 ^^

26楼 CxxlMan 2010-02-26 02:18发表

你提到：「透視變換就是用透視矩陣去乘頂點(x, y, z, 1)，乘完之后呢，我們得到了頂點的齊次形式(x', y', z', w')，用這個齊次形式來做裁剪，得到新的點(x'', y'', z'', w'')」這一點我覺得行不通，就以能避免 w = 0 近裁剪來說。(1) D3D 和 OpenGL 是否真的有在這階段做裁剪。(2) 若不做透視除法，D3D 和 OpenGL 要怎麼決定近裁剪平面在哪。

25楼 CxxlMan 2010-02-25 19:59发表

本來透視投影和裁剪是沒直接關西,我是對 w = 0 的情況有疑問,可是你說在透視除法前會先裁剪掉造成 w = 0 的頂點,若真是如此就有依存關西了,若 D3D 和 OpenGL 真的在透視除法前會先裁剪掉就太好了,不用擔心 w 會等於 0 的問題,可是真的有這麼做嗎?而且現在流行 Shader 的方式,把各個轉換矩陣都先乘好,直接用來處理頂點,也沒事先做裁剪,完全信任 D3D 和 OpenGL 會在 CVV 做好裁剪,這對 w = 0 的情況要怎麼解決

24楼 Twinsen 2010-02-25 18:15发表

如果把 透视变换+投影 应用到视锥体上，得到的就是CVV。实际上透视变换 + 投影 的目的就是把视锥体变成CVV，而把它应用于顶点的则可以看作是一种“统一的形变”或者“空间的变换”——经过这个变换，视锥体变成了CVV，而其他的顶点也按照这个模子进行变换。

23楼 Twinsen 2010-02-25 18:11发表

另外，这么说你可能就容易理解了。透视投影可以分成透视变换+投影两个步骤。透视变换就是用透视矩阵去乘顶点(x, y, z, 1)，乘完之后呢，我们得到了顶点的齐次形式(x', y', z', w')，用这个齐次形式来做裁剪，得到新的点(x'', y'', z'', w'')，然后是透视除法——也就是投影！(x''/w'', y''/w'', z''/w'')=(x''', y''', z''', 1)，最后这个坐标就是在归一化的设备坐标中的点。

22楼 Twinsen 2010-02-25 18:03发表

你所说的应该是（2），这个裁剪是在相机空间中进行的，是没有经过透视变换的。是通过在原点的相机系统的视锥体进行裁剪的，这个裁剪就是用视锥体的6个平面（近、远、左、右、上、下）对进入相机空间的多边形进行的。请注意，无论在什么地方进行裁剪，裁剪和投影都是没有直接关系的，如果没有裁剪，投影仍然可以进行。

21楼 Twinsen 2010-02-25 18:03发表

在固定流水线中，低级的观察体裁剪可以在（1）世界坐标系中通过视锥体裁剪；（2）相机坐标系中通过视锥体裁剪；（3）裁剪坐标系中通过CVV裁剪；我们在这篇文章中所说的裁剪就是（3）。这个可以通过图形API（比如opengl以及d3d）在硬件中完成。

20楼 CxxlMan 2010-02-25 14:22发表

要裁剪得要知道裁剪平面在哪，雖然建立透視投影矩陣時有提供近裁剪平面，但已被轉換進計算式中了，因此在透視除法前，不管從頂點或是矩陣中都沒法得到裁剪平面的原始訊息，若真要在這一階段做裁剪，得把你文中那張透視圖往下拉，讓 np 貼齊到 Z=0 位置，自然就有一個 Z=0 的 xy 平面來作裁剪平面，不過這也要 D3D 和 OpenGL 有支援這樣的裁剪才行，我看過的範例程式大多依據類似你的透視投影圖設計，所以不認為 D3D 和 OpenGL 有支援。

19楼 Twinsen 2010-02-24 09:52发表

未作透视除法的顶点具有最大的信息量，因为它包含了w，具有原始的z信息在里面，裁剪的时候是使用齐次形式(x, y, z, w)，而透视除法（投影）之后得到的是普通形式(x/w, y/w, z/w)这个就是NDC中的坐标，它是一个损失了信息量的结果，用来进行光栅化操作，其中z/w得到的就是NDC中的深度值，用来做z缓冲深度测试。请先理解透视投影的这个思路和z-buffer的使用情况，再考虑w buffer以及其他一些技术。

18楼 Twinsen 2010-02-24 09:52发表

最原始的透视投影的目的不是为了裁剪，而是为了把3D物体投影到2D平面上在屏幕上显示出来。CVV不是必需的，而是为了方便裁剪而生成的产物，如果没有CVV，也可以直接用视锥体进行裁剪，但复杂度很大，但透视变换矩阵会简单一些。

17楼 CxxlMan 2010-02-22 18:08发表

透視除法前做裁剪我覺得有矛盾，整個透視投影變換不就是為了能用 CVV 做裁剪嗎，未做透視除法前的頂點應不足以做裁剪吧，一個三角面或線段的各頂點在這階段還未處於 CVV 的座標間中，若能做裁剪還要透視除法做什麼。
另外那個透視投影變換矩陣通常還會和 Z Buffer 或 W Buffer 結合，讓我覺得更頭痛，是不是能補充這部份呢。

16楼 Twinsen 2010-02-18 12:11发表

因为这篇文章主要是推导透视投影变换矩阵,而裁剪部分的内容一来比较多,二来完全可以和透视部分分开来理解.因此不会在这篇文章中介绍.如果希望了解裁剪的细节,有很多图形学的材料可以参考.

15楼 CxxlMan 2010-02-16 16:04发表

你文中沒提到在透視除法前做裁剪的細節,這要怎麼做呢

14楼 Twinsen 2010-02-11 09:46发表

CxxlMan，z会被放入w，但是进行透视除法之前会进行裁剪，会把z=0的部分剔除掉从而保证透视除法的时候不会存在z=0的顶点。

13楼 CxxlMan 2010-02-11 02:25发表

那個 z 不是會被放入 w (應說是放入 -z) 嗎,若 z 是 0 ，進行透視除法時沒問題嗎

Re: zjghs2007 2011-12-06 20:50发表

回复popy007：回复CxxlMan：我认为，z为0的点是肯定没有意义的，绝对可以剔除。因为近平面不可能为 z=0，所以z轴坐标为0的点p不可能存在于投影视椎体里

12楼 Twinsen 2010-02-10 09:46发表

CxxlMan，我们推导矩阵的时候需要除以z，但实际上矩阵推导完成之后，会在透视除法之前进行裁剪。所以z=0的部分都会被剔除掉，不会出现除零的情况。这也看出裁剪的必要性。

11楼 CxxlMan 2010-02-09 20:16发表

我有一個疑問
那個 z 的值若正好為 0 不會有問題嗎?

10楼 Twinsen 2009-09-27 12:45发表

lovewhatilove

恩，你说的没错，这样简化计算不错。实际上投影平面可以是任意的，但大多数的情况则简化为前裁剪平面。这里的推导肯定是以通用的前裁剪平面来看待，为什么呢？就是因为现在的图形API基本上都用这个平面。

9楼 Twinsen 2009-09-27 12:40发表

diyer2002

opengl在进行裁剪的时候使用的是未经透视除法的齐次坐标形式，在这个形式的基础上进行裁剪比较方便（主要是可以留住原始的z值，并且在CVV裁剪的时候随时可以用x，y，z除以w得到CVV中的值）。而裁剪完毕之后呢，才会真正的使用透视除法把齐次形式变成普通形式。在外部看来，实际上可以理解为裁剪和透视除法是一步完成的，但具体来看就是我上面所说那样的。

8楼 diyer2002 2009-09-20 19:41发表

"是将相机空间中的点从视锥体(frustum)变换到规则观察体(Canonical View Volume)中，待裁剪完毕后进行透视除法的行为"

请问如果没有进行透视除法，如何能得到CVV，既然裁剪是在CVV中进行的，为啥在裁剪完毕，才进行透视除法（没有进行透视除法，如何得到CVV？）？

7楼 ruf 2009-07-03 13:01发表

在那个X,Y的求值那块和3D图形中的求法不一样，我一般就以Z=1为参考面，对X,Y单独来考虑，投影到Z=1面后，然后进行除tan(x/2.0)，进行拉伸到1.0即可。
这样得到的和DX和GL的是一样的结果。。。
如果像你这样选择Z=ZNEAR为计算面，未免把他搞太复杂了点。。。
个人的一点小小意见，具体可以参考计算机图形的透视截头体到平行标准截头体的转换那块。。。

6楼 huangguo 2008-12-24 21:58发表

还是看得一头雾水

5楼 huangguo 2008-12-24 21:58发表

还是看得一头雾水

4楼 伟慧 2008-07-17 23:13发表

写得很好额，终于懂了一点，不过我还得继续研究

3楼 suhonghit 2008-06-23 15:52发表

恩，写的很不错，再接再厉。

2楼 勤奋happyfire 2008-02-09 00:05发表

很不错，最好再补上视口变换部分

1楼 alexandercer 2007-09-23 23:29发表

赞，研究中。。。

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场

**核心技术类目**

全部主题 Hadoop AWS 移动游戏 Java Android iOS Swift 智能硬件 Docker OpenStack
VPN Spark ERP IE10 Eclipse CRM JavaScript 数据库 Ubuntu NFC WAP jQuery
BI HTML5 Spring Apache .NET API HTML SDK IIS Fedora XML LBS Unity
Splashtop UML components Windows Mobile Rails QEMU KDE Cassandra CloudStack FTC
coremail OPhone CouchBase 云计算 iOS6 Rackspace Web App SpringSide Maemo
Compuware 大数据 aptech Perl Tornado Ruby Hibernate ThinkPHP HBase Pure Solr
Angular Cloud Foundry Redis Scala Django Bootstrap