# 第3章　透视投影

透视投影是投影几何里面的概念，是一种将三维物体投影在二维平面的方法。

整个图形系统3D的概念，以及从3D到2D的投影，都是基于透视投影和正交投影的(光线追踪除外)。从开发者的角度看，用户最终获得的图像，和创建3D场景时输入的顶点坐标、模型视图变换矩阵、投影矩阵紧密相关。所以理解投影背后的设计原理，有助于开发者理解给定输入顶点坐标和投影矩阵，将会获得怎样的输出。理解输入和输出，对3D应用的开发者而言，是非常必要的。另外，如果要进行GPU设计，或者软件模拟GPU的行为，也需要深入理解投影背后的设计。本章介绍3D编程中最常用的透视投影，后续章节会介绍正交投影。

虽然本章都是公式推导，但是数学难度并不高。对读者数学上的要求有两点：①高中的立体几何知识；②矩阵运算的基本知识(甚至都不需要了解矩阵的逆)。本章使用的矩阵运算，一定程度上可以映射到高中的多元一次函数方程组的求解问题。从这个角度看，理解本章的内容，有高中的数学基础就可以了。

前面提到，3D流水线抽象出了多种坐标空间。透视投影实现的是从眼睛坐标到NDC坐标的变换，而眼睛坐标依赖于输入的物体坐标、模型视图矩阵等。所以要分析透视投影，还要考虑模型视图等的影响。模型视图矩阵让问题的分析变得复杂，测试起来也很不方便。针对本章分析透视投影矩阵，一种简化方法是，将模型视图变换设置为单位矩阵，这样的话，物体坐标、世界坐标、眼睛坐标就重合了。但是透视投影的几何模型不受这个简化的影响。

本章讨论的透视投影模型，就是在没有模型变换和视图变换的前提下，如何将眼睛坐标系的点，变换到归一化的NDC坐标系。没有模型变换，物体坐标系就是世界坐标系，这一点比较直接。没有视图变换的意思是，使用系统默认的视图变换。视图代表了用户的眼睛(或摄像头)观察3D场景的位置和方向。在GL或者Vulkan里面，眼睛默认在世界坐标系的原点。但是GL/Vulkan都可以通过观察函数(一般都叫lookAt)来修改眼睛的位置和观察方向。使用默认的眼睛位置和观察方向(有些应用程序，例如Chromium的合成器，Android的SurfaceFlinger都没有额外设置观察函数)，带来的一个好处是，视图变换就是没有位移旋转的变换，也就是单位矩阵。因而视图变换后形成的眼睛坐标系和世界坐标重合了。

这个从眼睛坐标系变换到NDC坐标系的变换，需要具有将3D内容以合适的方式呈现在2D平面上的能力。所谓合适的方式，是指：

(1) 保持立体感，立体感的本质就是远的物体显得较小，近的物体显得较大。

(2) 和人眼距离一样的物体，投影后物体间的相对位置不变。

透视投影，就正好满足了这两个条件。

那么，如何求解得到这个透视投影的变换矩阵？

求解一个矩阵，要先理解这个矩阵的输入和输出是什么。透视投影的输入是眼睛坐标（如前述，因为眼睛坐标和物体坐标重合，所以也就是物体坐标）。输出呢？有两种比较直接的选择，一种是输出窗口（本书讨论窗口和视口重合的情况）坐标，这样的坐标是非归一化的。这个选择的缺点是，投影矩阵和窗口系统耦合到了一起。也就是窗口变换了（放大、缩小甚至移动），投影矩阵要重新计算。如果选择输出归一化的坐标呢？那就需要一次额外的窗口变换来将归一化的坐标映射到窗口坐标。GL 和 Vulkan 都选择了归一化的窗口坐标方式。这个归一化的窗口坐标系叫作 NDC 坐标系（normalized device coordinates）。

现在问题变成了：给出眼睛坐标，如何求解出相应的 NDC 坐标？这里的眼睛坐标位于眼睛坐标系（等同于物体坐标系，世界坐标系），NDC 坐标位于 NDC 坐标系（归一化的坐标系）。

但是通过分析发现，透视投影，也就是从眼睛坐标系到 NDC 坐标系的变换，不是一次矩阵运算就能解决的（所谓一次矩阵运算，里面涉及的矩阵可以是多个矩阵的乘积），其中有一个过程甚至引入了非线性部分。这个非线性部分，让整个透视投影变得没那么直观，让 3D 流水线变得更加复杂。

另外，公开的文献在分析透视投影的时候，或者直接给出结论，或者将几何模型和数学算法上的优化混合到了一起，这也让透视投影变得难以理解。

针对透视投影理解上的难点，本章将透视投影的几何模型和数学算法上的优化分开，因而得到了两个模型：透视投影的几何模型、透视投影的透视除法模型。前一个几何模型完全使用初等几何知识推导，重在理解物体和空间的几何关系，因此比较直观。后一个透视除法模型则侧重于数学算法优化。这两个模型降低了学习曲线，能够帮助读者理解透视投影的本质。

本章内容安排如下。

（1）左右手坐标系。介绍左右手坐标系的区别。

（2）3D 坐标和坐标系。介绍了 3D 坐标的特点及所在的坐标系、坐标系之间的变换。

（3）3D 流水线的基本概念。

（4）小孔成像。介绍了小孔成像的原理，并根据小孔成像推导出透视投影的模型。

（5）透视投影的几何模型。

（6）透视投影的透视除法模型。

本章要讨论非常多的坐标，对这些坐标约定如下。

物体坐标：$x_o, y_o, z_o$。齐次坐标形式：$x_o, y_o, z_o, 1$。

世界坐标：$x_{world}, y_{world}, z_{world}$。齐次坐标形式：$x_{world}, y_{world}, z_{world}, 1$。

眼睛坐标：$x_e, y_e, z_e$。齐次坐标形式：$x_e, y_e, z_e, 1$。

裁剪坐标：$x_c, y_c, z_c$。齐次坐标形式：$x_c, y_c, z_c, w_c$。

归一化的 NDC 坐标：$x_n, y_n, z_n$。齐次坐标形式：$x_n, y_n, z_n, 1$。

窗口（视口）坐标：$x_w$，$y_w$。

## 3.1　左右手坐标系

3D 坐标系分为左手坐标系、右手坐标系，如图 3-1 所示。

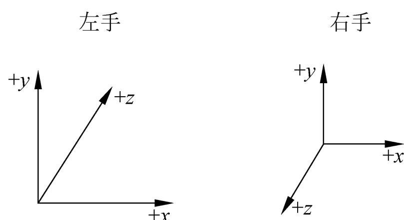


图 3-1　左手坐标系和右手坐标系

3D 接口的主要功能是提供渲染的接口，通常并不强制要求用户使用何种坐标系[①]。就像在一张白纸上书写，可以选择从左到右书写，也可以从上到下书写。所以对于一个 3D 程序而言，如果仅仅是为了测试的用途，或者程序完全由个人独立完成，那么使用左手右手坐标系都是可以的。但是考虑多人协作的情况，以及要和第三方程序兼容的情况，就有必要在程序设计之初就约定好该使用何种坐标系。

GL 和 Vulkan 使用的坐标系可能是不同的。本书对坐标系的约定如下。

(1) GL：物体坐标、世界坐标、眼睛坐标都使用右手坐标系；NDC 坐标和窗口坐标使用左手坐标系。

(2) Vulkan：物体坐标、世界坐标、眼睛坐标、NDC 坐标和窗口坐标都使用右手坐标系。

就 NDC 而言，GL 使用左手，Vulkan 使用右手，两者 $y$ 坐标相反。此外，GL 的 $z$ 坐标位于[－1.0，1.0]，Vulkan 的 $z$ 坐标位于[0.0，1.0]。

## 3.2　3D 坐标和坐标系

用户在代码里面指定的顶点坐标，经过模型、视图、透视投影等变换后，最终显示在 2D 平面(屏幕或者像面)上。这些模型视图透视投影变换，需要用到下面的坐标及坐标系。

(1) 物体坐标和物体坐标系。创建 3D 场景时传入的顶点坐标，就是物体坐标，用 $(x_o, y_o, z_o)$ 表示。虽然用户传入的坐标不包含 $w$ 分量，但是在实际的实现里面，会自动加上 $w=1$，因此使用的还是齐次坐标 $(x_o, y_o, z_o, 1)$。一个物体所有坐标点所在的坐标系，就是物体坐标系。

---

① OpenGL 标准在 Appendix B 部分提到，OpenGL 并不强制要求程序一定使用某个坐标系。OpenGL 主要是作为一种渲染的接口，它本身不和某个具体的坐标系统绑定。具体使用哪一种坐标系，是一种约定。具体标准 https://www.khronos.org/registry/OpenGL/specs/gl/glspec46.core.pdf。

(2) 世界坐标和世界坐标系。物体坐标乘以模型变换矩阵,就得到世界坐标。世界坐标的齐次形式是($x_{world}$,$y_{world}$,$z_{world}$,1)。世界坐标的本质是,位于不同物体坐标系的多个物体,最终都要显示在同一个空间的坐标系。这个坐标系,就是世界坐标系。

(3) 眼睛坐标和眼睛坐标系。世界坐标系乘以视图矩阵,得到眼睛坐标。其齐次坐标的 $w$ 分量是1,用($x_e$,$y_e$,$z_e$,1)来表示眼睛坐标的点。世界坐标到眼睛坐标的变换即视图变换是线性的,可以用通常的位移、旋转、缩放来表达。本章的物体坐标系、世界坐标系、眼睛坐标系三者重合,所以后面都用眼睛坐标系来描述。

(4) 投影坐标。位于眼睛坐标系,是眼睛坐标系的点投影到视景体近平面上产生的。近平面位于眼睛坐标系的 $-n$ 处,所以投影点的 $z$ 等于 $-n$。通常对于投影坐标只讨论 $x$、$y$ 两个分量,$z$ 分量是常数,没有使用。

(5) 裁剪坐标和裁剪坐标系。裁剪坐标是数学意义上的中间坐标。没必要将这个坐标系和具体的坐标空间联系起来。这里定义的坐标,携带了NDC空间坐标计算所需要的信息,但是它本身不是实际的有物理意义的点。之所以把这个坐标叫裁剪坐标,是因为根据裁剪坐标,可以把视景体之外的点裁剪掉。所以相对裁剪坐标的裁剪其实发生在投影之后,透视除法之前。齐次坐标形式是($x_c$,$y_c$,$z_c$,$w_c$)。后文会分析 $w_c$ 分量的特殊用途,所以并不是1。

(6) NDC坐标和NDC坐标系。裁剪坐标经过透视除法,得到NDC坐标。这个坐标是归一化的。其和眼睛坐标的一种可能的关系是,$x$:[$l$, $r$]到[$-1.0$, $1.0$];$y$:[$b$, $t$]到[$-1.0$, $1.0$];$z$:[$-n$, $-f$]到[$-1.0$, $1.0$]或[$0.0$, $1.0$]。NDC坐标并不要求gl_Position输出的值除以 $w$ 之后必须落在NDC之内。只是说,落在NDC之外的点将会被裁剪掉,所以无法参与后面的光栅化以及显示。齐次坐标形式是:($x_n$,$y_n$,$z_n$,1)。

(7) 窗口坐标和窗口坐标系。将NDC坐标映射到和具体窗口尺寸关联的坐标。

为了用方便的方法来表达同一个场景的多个物体(每个物体有自己独特的运动轨迹),抽象出了物体坐标、世界坐标以及实现两个坐标之间变换的模型变换。用户输入的顶点坐标,定义在物体坐标系,所有具有相同模型变换的点组成一个物体空间。这些点,经过同一个模型变换矩阵,变换到世界坐标空间。举个例子,一个场景里面有两个物体,物体1在移动,物体2在旋转。这个时候,我们的场景就位于世界坐标。每个物体会定义自己的物体坐标。对于物体1,在运动好了之后,用它的模型变换矩阵(平移),乘以相应的物体坐标位置,就得到物体在世界坐标的位置。同样,可以用另一个模型变换矩阵(旋转),计算出物体2旋转后的世界坐标。如果用GL/Vulkan来实现这个过程,可以在每一帧图像通过两次绘图来实现:第一次的输入是物体1,以及物体1的模型变换矩阵,调用绘图函数。第二次绘图输入的则是物体2及其模型变换矩阵,然后调用绘图函数。再看一个更加实际的例子,Chromium的合成器GLRenderer。当系统里面有多个不同的物体,并且它们的模型矩阵各不相同的时候,系统会触发多次绘图,但是每次绘图,都只涉及一个模型矩阵。最后这些绘图合成显示到一个最终的目标上去。所以,对于合成器程序而言,它的顶点有很多组,每组表示一个具体的物体,每组有一个自己的模型变换矩阵。

如果用户没有设置模型变换,物体坐标系就和世界坐标系重合。如果没有设置视图矩阵,世界坐标系和眼睛坐标系是重合的。在没有设置眼睛位置(调用lookAt函数)的时

候，默认眼睛位于世界坐标系的原点，朝向坐标系的$-z$方向。

图 3-2 列举了透视投影相关的 3D 变换。

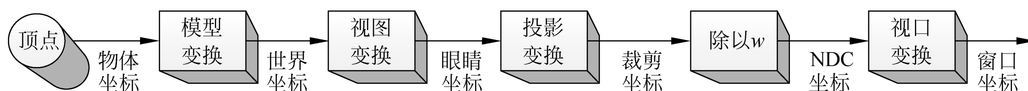


图 3-2　透视投影相关的 3D 变换

## 3.3　3D 流水线

GL 和 Vulkan 的流水线是类似的，如图 3-3 所示。

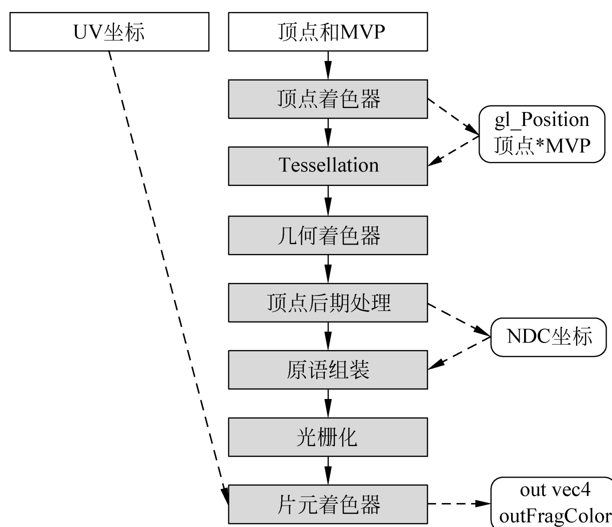


图 3-3　3D 流水线和顶点坐标、MVP、纹理坐标等的关系

用户输入的顶点、MVP 矩阵、纹理以及纹理坐标，和流水线关联的部分主要如下。

(1) 顶点着色器(vertex shader)：根据用户输入的顶点和 MVP 信息，生成裁剪坐标 gl_Position。

(2) 顶点后期处理(vertex post processing)：对顶点着色器输出的 gl_Position 进行透视除法，获得 NDC 坐标，如公式 3-1 所示。然后对 NDC 坐标进行视口变换，获得窗口坐标。

$$\begin{pmatrix} x_n \\ y_n \\ z_n \end{pmatrix} = \begin{pmatrix} \dfrac{\text{gl_Position}.x}{\text{gl_Position}.w} \\ \dfrac{\text{gl_Position}.y}{\text{gl_Position}.w} \\ \dfrac{\text{gl_Position}.z}{\text{gl_Position}.w} \end{pmatrix}$$

**公式 3-1　透视除法**

在 GPU 里面,透视投影是在顶点着色器和顶点后期处理两个阶段分两步完成的。第一步在顶点着色器里面,将顶点坐标和透视投影矩阵相乘,得到裁剪坐标;第二步,在顶点后期处理阶段,通过固定管线的透视除法得到 NDC 坐标。

(3) 光栅化:获得了图形的窗口顶点坐标后,要根据图形的窗口顶点生成相应的图形,譬如三角形,这个过程就是计算机图形学里面的扫描线算法。这个算法将三角形变成一条条的线,线上的点都是根据顶点插值生成的。每次插值生成线上的一个点,相应生成一个 *uv* 坐标,这个 *uv* 坐标对应一个片元,然后将这个 *uv* 坐标传递给片元着色器(fragment shader),片元着色器通过 *uv* 坐标对纹理进行采样,并对这个点进行着色。通常也用光栅化来代指基于透视投影和正交投影的 3D 编程模型,以和基于光线传播实现的光线追踪 3D 编程模型区分开。

(4) 片元着色器(fragment shader):对扫描线生成的片元进行着色。颜色可以根据用户在输入顶点时指定的颜色插值生成,也可以来自用户提供的图片相应 *uv* 坐标位置的像素颜色。

通常用户代码会将 *uv* 坐标和顶点坐标以及顶点的颜色一起输入,如程序清单 3-1 所示。

**程序清单 3-1　*uv* 坐标的一种输入方式(Vulkan)**

```
std::vector<Vertex> vertices =
{
 { { 1.0f, 1.0f, 0.0f }, { 1.0f, 1.0f },{ 0.0f, 0.0f, 1.0f } },
 { { -1.0f, 1.0f, 0.0f }, { 0.0f, 1.0f },{ 0.0f, 0.0f, 1.0f } },
 { { -1.0f, -1.0f, 0.0f }, { 0.0f, 0.0f },{ 0.0f, 0.0f, 1.0f } },
 { { 1.0f, -1.0f, 0.0f }, { 1.0f, 0.0f },{ 0.0f, 0.0f, 1.0f } }
};
```

上面的结构包含四个顶点信息,每个顶点信息包含:顶点坐标、*uv* 坐标、顶点颜色。中间的 *uv* 坐标会和其他的顶点信息一起传递给顶点着色器和片元着色器。这里 *uv* 坐标就是{1.0f,1.0f},{0.0f,1.0f},{0.0f,0.0f},{1.0f,0.0f}。

相应的片元着色器如程序清单 3-2 所示。

**程序清单 3-2　片元着色器使用插值后的 *uv* 坐标**

```
layout (location = 0) in vec2 inUV;
```

但是不能将 inUV 和用户顶点里面的四个 uv 坐标直接等同起来。inUV 就是这四个顶点提供的 *uv* 坐标插值生成的点的集合。请注意,顶点着色器操作的是用户输入的顶点,而片元着色器操作的是通过顶点插值生成的所有点。

gl_Position 和用户指定的顶点(vertex)是一一对应的(前提是顶点没有落在视景体外面)。或者说,物体空间的每一个顶点,在裁剪坐标空间都有一个对应的 gl_Position。gl_Position 经过透视除法后,得到 NDC 空间坐标。NDC 空间的点,经过视口变换,得到窗口坐标。总的来说,每个顶点(vertex)都有一个对应的 gl_Position,一个 NDC 空间的点。渲染(rasterization)阶段,对顶点坐标组装成的三角形应用扫描线(scanline)算法,插

值得到顶点之外的填充区域的坐标点，并用片元着色器（fragment shader）对这些点进行着色（shading）。

所以顶点着色器是以顶点为单位进行的，一个三角形就是三个顶点，相应的顶点着色器会被调用三次。而片元着色器，是以片元（可以暂时理解为像素）为单位的，也就是最终显示在窗口上面的每个像素，都会执行一次片元着色器（没有多重采样的时候）。两者之间通过 GPU 的扫描线关联起来。

## 3.4　小孔成像

透视投影和小孔成像的原理是一样的。小孔成像的原理如图 3-4 所示。

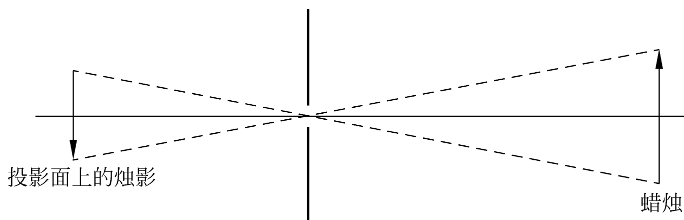


图 3-4　小孔成像

小孔成像的特点如下。

（1）小孔比较小的时候，蜡烛同一个位置（物点）发出的光线将汇聚在投影面很小的一个范围（近似为一个像点），蜡烛不同位置发出的光线会到达投影面的不同位置，因而不会在屏幕上相互重叠，所以屏幕上的像比较清晰。

（2）当孔比较大的时候，蜡烛同一个位置发出的光线会分散在投影面的一个区域，蜡烛不同部分发出的光线有可能在投影面的同一个位置上重叠，投影面上的像就不清晰了。如图 3-5 所示，如果孔很大，那么蜡烛两个不同位置发出的光线，可能投影到了投影面的同一个位置。

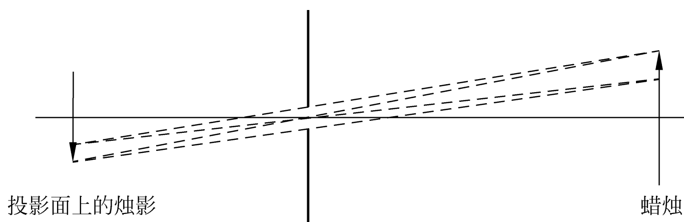


图 3-5　大孔成模糊的像

由于小孔成像画质受孔大小的影响，所以透视投影对小孔成像做了一点假设：透视投影的小孔无限小，小到物体的每个点，只能有一根光线通过小孔。这样的优点是，无论物体到小孔的距离是多少，物体上的每个点都只有一条光线通过小孔，这也就保证了两个点的两条光线，通过小孔后不会发生重叠，因而保证了成像一定是最清晰的。

除了这点假设之外，透视投影的近平面(也就是小孔成像里面的投影面)放在了小孔和物体的同侧。这带来了计算上的便利，但是投影和物体之间的三角形关系并没有改变。唯一改变的是，小孔成像成的是倒立的影像。透视投影，则把这个成像再次倒立了，也就是成为正立的影像。

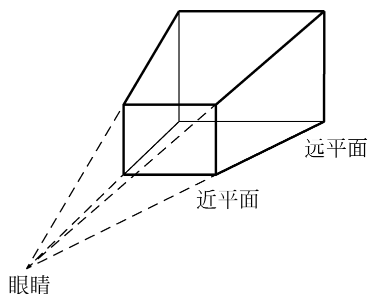

图 3-6　透视投影模型

所以，我们得到的透视投影模型是如图 3-6 所示的：小孔成了眼睛(或者摄像头)；投影面成了近平面。蜡烛就是位于近平面和远平面之间的物体，这意味着用户定义的 3D 场景，也需要位于近平面与远平面之间，否则不可见。

## 3.5　模型变换和世界变换的意义

物体坐标经过模型矩阵变换之后，变成世界坐标。在世界坐标系中，来自不同物体坐标系的物体，都是相对于同一个世界坐标系原点的。

为什么物体坐标之外，还需要引入一个世界坐标？如果整个场景里面只有一个物体，那么物体坐标和世界坐标是可以重合的。如果场景里面有多个物体，每个物体都有自己的物体坐标，系统可以很方便地对每个物体进行调整，因为局部坐标总是很方便使用的。例如如果以人为中心，要向左旋转自己的胳膊 90°，这是很容易做到的事情。但是，如果要求以地球为中心，旋转胳膊 90°，那需要坐飞机绕着地球飞行 1/4 圈。

不过还可以从性能优化的角度，来理解世界变换的意义。

考虑一个情况，也就是用户视角变化，这是常见的场景。

如果没有世界坐标。假设 $\boldsymbol{M}_{obj->view}$ 直接将物体坐标点变换到视图空间。因为视角变了，所以每个物体的 $\boldsymbol{M}_{obj->view}$ 都要重新计算，再重新计算物体到视图空间的变换。变换的过程如下。

### 1. 重新计算每个物体的 $M_{obj->view}$($n$ 次)

这个计算是针对所有物体的，如伪代码 3-1 所示。

**伪代码 3-1**

```
循环遍历(物体: 所有的物体){
 针对每个物体重新计算 M obj->view;
}
```

### 2. 重新计算每个点在视图空间的坐标 $P_{view}$

针对物体的每个点，计算其在新的视图空间的坐标，如伪代码 3-2 所示。

**伪代码 3-2**

```
M_vp = M_projection × M_obj->view;
循环遍历 (点 P: 物体所有的点) {
 P_view = M_vp × P;
}
```

如果有世界坐标，变换过程如下（视图变换的时候，物体坐标到世界坐标的变换 $\boldsymbol{M}_{obj->world}$ 保持不变）。

(1) 重新计算世界坐标到视图空间的变换 $\boldsymbol{M}_{view}$（1 次）；

(2) 重新计算每个点的视图空间坐标 $\boldsymbol{P}_{view}$，如伪代码 3-3 所示。

**伪代码 3-3**

```
M_mvp = M_projection × M_view × M_obj->world;
循环遍历(点 P: 物体所有的点) {
P_view = M_mvp × P;
}
```

综上，在场景里面有多个不同物体的时候，世界坐标可以将循环里面的矩阵运算优化到循环外面来，因而还可以带来性能上的提升。

## 3.6　透视投影的几何模型

我们将透视投影定义为通过眼睛坐标生成 NDC 坐标的过程。眼睛坐标用于描述 3D 空间的物体，NDC 坐标则可以很容易地线性映射到 2D 平面上。这和 GPU 实现的透视投影行为是一致的。

本章介绍透视投影理论上的几何模型。这个模型没有经过优化，理论上可以工作，也能实现眼睛坐标到 NDC 坐标的转换，但是并没有实际应用到 GPU 流水线。这个模型的价值在于，它得到的一些结论，是后文推导优化后实际应用于 GPU 流水线的基础。

在 3.4 节，根据小孔成像得到了透视投影的基本模型。实际上，为了减少对输出窗口系统的依赖，透视投影得到的结果并不是窗口坐标，而是中间坐标 NDC 坐标。透视投影的视景体通常用 left($l$)、right($r$)、bottom($b$)、top($t$)、near($n$)、far($f$)等参数来表示，如图 3-7(a)和图 3-8(b)所示。

虽然只是一个理论模型，但是也要有相应的坐标系统。所以本章会使用 GL 和 Vulkan 的坐标系统。应注意的是，仅仅是使用 GL 和 Vulkan 的坐标系统，而不是说 GL 和 Vulkan 的实现，使用了本节推导的几何理论模型。

如果使用 GL 坐标系，透视投影和 NDC 坐标系的关系如图 3-7 所示。图示视景体里面的点($l$, $t$, $-n$)对应 NDC 坐标系的点(−1, 1, −1)，以此类推。此外，由于远平面的映射关系和近平面是类似的，为了排版简洁，没有标注视景体远平面的点，相应的 NDC 坐标标注了远平面顶部的两个点。

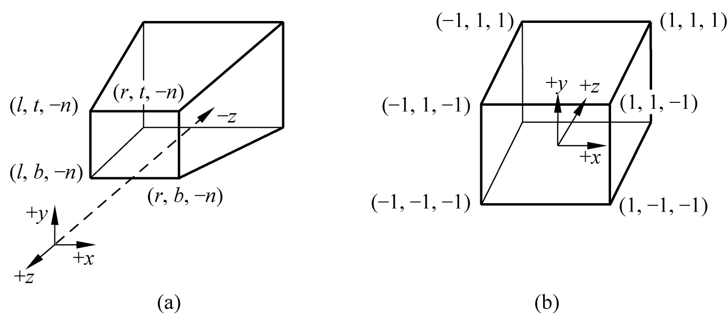


图 3-7　视景体和 NDC 坐标系(GL)

GL 视景体和 NDC 坐标系的映射关系如表 3-1 所示。

**表 3-1　视景体和 NDC 坐标系的关系(GL)**

| | 视景体 | NDC |
|---|---|---|
| $x$ | $[l, r]$ | $[-1, 1]$ |
| $y$ | $[b, t]$ | $[-1, 1]$ |
| $z$ | $[-n, -f]$ | $[-1, 1]$ |

如果使用 Vulkan 坐标系,视景体和 NDC 坐标系的关系如图 3-8 所示。

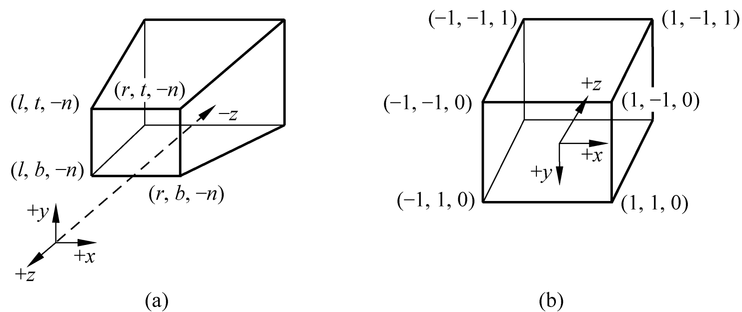


图 3-8　视景体和 NDC 坐标系(Vulkan)

Vulkan 的映射关系如表 3-2 所示。

**表 3-2　视景体和 NDC 坐标系的关系(Vulkan)**

| | 视景体 | NDC |
|---|---|---|
| $x$ | $[l, r]$ | $[-1, 1]$ |
| $y$ | $[b, t]$ | $[1, -1]$ |
| $z$ | $[-n, -f]$ | $[0, 1]$ |

## 3.6.1　眼睛坐标到投影坐标

如图 3-9 所示,在物体点 $P$ 和眼睛之间连接一条直线,直线和投影面(近平面)的交