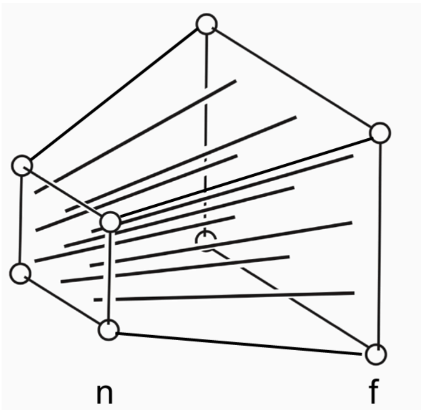
Games101

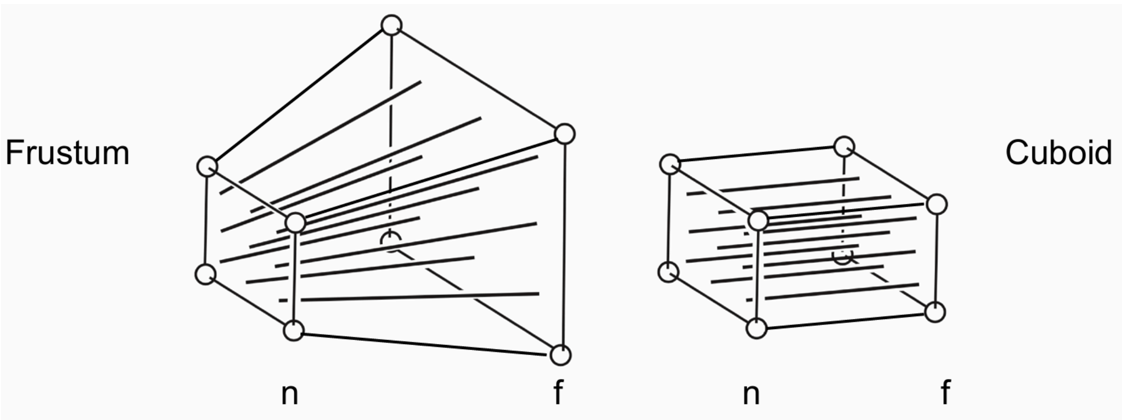
Chapter 4

如图所示，目的是将 f 面 投影至 n 面



- 这里将投影的过程分解为两个步骤

  1. 将 frustum “挤压” 为一个 “cuboid”



  2. 然后再将 f 面经正交投影， 投影至 n 面

  这里只讨论第一个步骤

挤压

在挤压的过程中，以下 3 个结论是显然的，或者说是我们挤压的一种前置要求

n 面所有点坐标都不会变

f 面所有点的 z 坐标都不会变

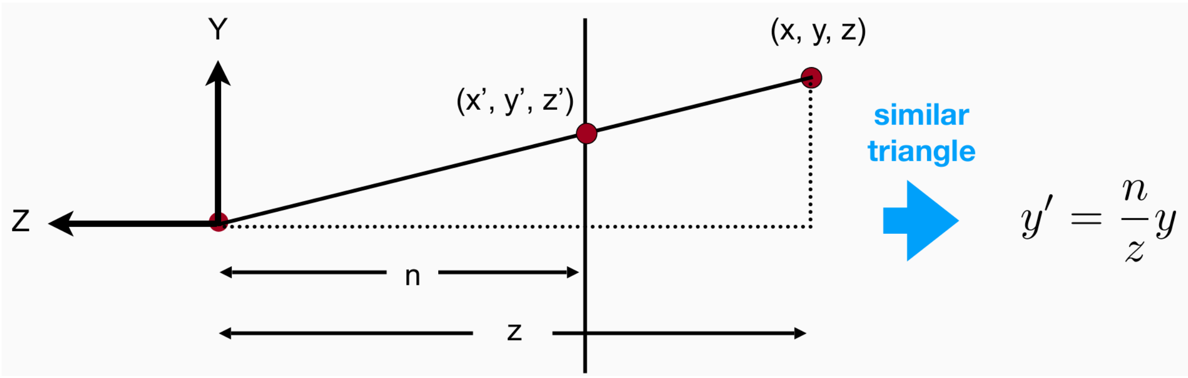
f 面中心点的坐标不会变

m 面([f, n] 之间的面)任意点的

设 (x_m, y_m, z_m)是 m 面上一点, 挤压后为 (x'_m, y'_m, z'_m)

挤压后的 m 面与 n 面的尺寸是完全相等的, 因此任意点的(x'_m, y'_m) 与 n 面的对应点的 (x_n, y_n) 是相等的, 所以这里不妨通过计算 n 面的 (x_n, y_n) ,间接的计算 m 面的 (x'_m, y'_m)

如图所示,



由相似三角形可得

$$x'_m = x_n = \frac{n}{z_m} x_m$$
$$y'_m = y_n = \frac{n}{z_m} y_m$$

而 z'_m 目前无法求得, 可暂时保持未知

综上述条件可得(这里将变换后的向量扩大了 z_m 倍, 后面还会提到这件事)

$$M^{4\times4}_{persp\to ortho}\begin{pmatrix} x_m \\ y_m \\ z_m \\ 1 \end{pmatrix} = \begin{pmatrix} x'_m \\ y'_m \\ z'_m \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{n}{z_m} x_m \\ \frac{n}{z_m} y_m \\ ? \\ 1 \end{pmatrix} = \begin{pmatrix} n x_m \\ n y_m \\ ? \\ z_m \end{pmatrix}$$

"挤压"变换矩阵的形式如下

$$M^{4\times4}_{persp\to ortho} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

则

$$
\begin{aligned}
&(1) \quad a_{11}x_m + a_{12}y_m + a_{13}z_m + a_{14} = nx_m \\
&(2) \quad a_{21}x_m + a_{22}y_m + a_{23}z_m + a_{24} = ny_m \\
&(3)? \\
&(4) \quad a_{41}x_m + a_{42}y_m + a_{43}z_m + a_{44} = z_m
\end{aligned}
$$

由

$$
\begin{aligned}
&(1) \Rightarrow nx_m\text{只与}x_m\text{相关且无常数项},\ \therefore a_{12} = a_{13} = a_{14} = 0,\ a_{11} = n \\
&(2) \Rightarrow ny_m\text{只与}y_m\text{相关且无常数项},\ \therefore a_{21} = a_{23} = a_{24} = 0,\ a_{21} = n \\
&(4) \Rightarrow z_m\text{只与}z_m\text{相关且无常数项},\ \therefore a_{41} = a_{42} = a_{44} = 0,\ a_{43} = 1
\end{aligned}
$$

$$
\therefore \quad M^{4\times4}_{persp\to ortho} = \begin{pmatrix} n & 0 & 0 & 0 \\ 0 & n & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 1 & 0 \end{pmatrix}
$$

n 面任意点

设 (x_n, y_n, z_n)是 n 面上一点, 挤压后为 (x'_n, y'_n, z'_n)

显然

$$
\begin{aligned}
x'_n &= x_n \\
y'_n &= y_n \\
z'_n &= z_n = n
\end{aligned}
$$

下面变换后的点的齐次坐标扩大了n倍, 本质也是扩大了z_m倍, 因为此时$z_m = n$

则

$$
M^{4\times4}_{persp\to ortho}\begin{pmatrix} x_n \\ y_n \\ n \\ 1 \end{pmatrix} = \begin{pmatrix} x'_n \\ y'_n \\ z'_n \\ 1 \end{pmatrix} = \begin{pmatrix} x_n \\ y_n \\ n \\ 1 \end{pmatrix} = \begin{pmatrix} nx_n \\ ny_n \\ n^2 \\ n \end{pmatrix}
$$

$$
\therefore \quad a_{31}x_n + a_{32}y_n + a_{33}n + a_{34} = n^2
$$

$$
\therefore \quad a_{31} = a_{32} = 0 \quad \text{且}\ a_{33}n + a_{34} = n^2
$$

f 面中心点

设 $(0, 0, z_f)$是 f 面中心点, 挤压后为 $(0, 0, z'_f)$

显然 $z'_f = z_f = f$

下面变换后的点的齐次坐标扩大了f倍, 本质也是扩大了z_m倍, 因为此时$z_m = f$

则

$$M_{persp\to ortho}^{4\times4}\begin{pmatrix}0\\0\\f\\1\end{pmatrix}=\begin{pmatrix}0\\0\\z'_f\\1\end{pmatrix}=\begin{pmatrix}0\\0\\f\\1\end{pmatrix}=\begin{pmatrix}0\\0\\f^2\\f\end{pmatrix}$$

$\therefore\ a_{33}f+a_{34}=f^2$

由

$$a_{33}n+a_{34}=n^2$$
$$a_{33}f+a_{34}=f^2$$

可计算得

$$a_{33}=n+f$$
$$a_{34}=-nf$$

综上所述

$$M_{persp\to ortho}^{4\times4}=\begin{pmatrix}n&0&0&0\\0&n&0&0\\0&0&n+f&-nf\\0&0&1&0\end{pmatrix}$$

最后

假设模型上任一点的坐标为$(x,y,z,1)^T$,该点经过 $M_{persp\to ortho}^{4\times4}$ 变换的结果为 $(nx,ny,(n+f)\cdot z-nf\cdot z,z)^T$，显然这是一个 `enlarged` 的齐次坐标, 再进行正交投影变换后, 其依然是非标准形式的齐次坐标, 反映在模型上就是模型并不是在 `标准设备坐标系` 下, 我们需要将其标准化, 标准化的结果为 $(nx/z,ny/z,(n+f)-nf,1)^T$

进一步探讨, 假设变换后(未标准化前) 的坐标为 $(x',y',z',z)^T$, 标准化后为 $(x'/z,y'/z,z'/z,1)^T$, 显然 `z` 值 越大, `标准化` 时被缩小的程度就越高, 换句话说, 某个点离视点越远, 看起来就越小, 透视效果越强, 所以说 (x',y',z',w) 中, w 分量可以表征点的透视程度

Chapter 9 - homework 3

`gl_NormalMatrix` 存在于许多顶点着色器中, 本节将介绍这个矩阵是什么以及它的用途, 本节的灵感来自 Eric Lengyel 所著的优秀书籍《3D 游戏编程和计算机图形数学》

许多计算都是在观察空间进行的, 这与照明通常在观察空间)中进行这一事实有关，否则与观察位置相关的效果（如镜面反射光）将更难实现.

因此，我们需要一种将法线转换到观察空间的方法, 将顶点转换到观察空间的方法如下:

$$vertexEyeSpace=gl\_ModelViewMatrix*gl\_Vertex$$

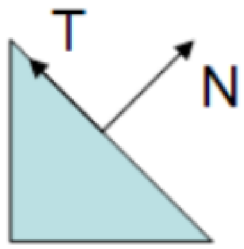
那么为什么我们不能对法线向量做同样的处理呢?
首先, 法线是一个包含 `3` 个浮点的向量，而 `Modelview` 矩阵是 `4×4`
其次，由于法线是一个矢量，我们只想变换它的方向, 模型视图变换矩阵中包含方向的区域是左上角的 `3×3` 子矩阵, 那么为什么不将法线乘以这个子矩阵呢?

用下面的代码就可以轻松实现:

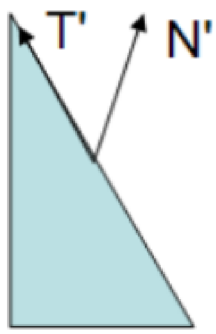
$$normalEyeSpace = vec3(gl\_ModelViewMatrix * vec4(gl\_Normal, 0.0));$$

那么，`` `gl_NormalMatrix `` 只是简单对代码进行了简化或优化吗? 不，并非如此, 上述代码在某些情况下可以工作，但并非所有情况都可以.

让我们来看看一个潜在的问题:



在上图中，我们看到一个三角形，它有一个法向量和一个切向量, 下图显示了当模型视图矩阵包含非均匀比例时发生的情况



注意：如果比例是统一的，那么法线的方向就会被保留，而长度则会受到影响，但这很容易通过归一化来解决.

在上图中，`Modelview` 矩阵应用于所有顶点和法线，结果显然是错误的：变换后的法线不再垂直于曲面.

我们知道，矢量可以表示为两点之间的差值, 考虑到切线向量，它可以计算为三角形边上两个顶点之间的差值, 如果 P_1和P_2 是定义三角形边的顶点，我们可以知道

$$T = P_2 - P_1$$

考虑到矢量可以写成最后一个分量为零的四分量元组，我们可以将相等的两边与模型视图矩阵相乘

$T * Modelview = (P_2 - P_1) * Modelview$

结果是:

$T * Modelview = P_2 * Modelview - P_1 * Modelview$

$T' = P'_2 - P'_1$

P'_1和P'_2 是变换后三角形的顶点， `T'` 仍然与三角形的边缘相切, 因此， `Modelview` 保留了切线, 但没有保留法线.

考虑到与向量 `T` 相同的方法，我们可以找到两个点 Q_1和Q_2，使得

$N = Q_2 - Q_1$

主要问题在于，如上图所示，通过变换点定义的向量 $Q'_2 - Q'_1$并不一定保持法线, 法向量不像切线向量那样定义为两点之间的差值，它被定义为垂直于曲面的向量,

因此，我们现在知道，不能在所有情况下都使用 `Modelview` 来变换法向量, 那么问题来了，我们应该应用什么矩阵呢?

考虑一个 `3x3` 矩阵 `G`，让我们看看如何计算这个矩阵来正确变换法向量.

我们知道，在矩阵变换之前，$T \cdot N$ = 0，因为根据定义，向量是垂直的, 我们还知道，变换后$T' \cdot N'$必须保持等于零，因为它们必须保持相互垂直, `T` 可以安全地与模型视图左上方的 `3x3` 子矩阵相乘（`T` 是一个向量，因此 `w` 分量为零），我们称这个子矩阵为 `M`,

假设矩阵 `G` 是转换法向量 T 的正确矩阵, 因此方程如下

$N' \cdot T' = (G * N) \cdot (M * T) = 0$

点积可以转化为矢量的积，因此

$N' \cdot T' = (G * N)^T * (M * T) = 0$

请注意，必须考虑第一个向量的转置，因为这是将向量(`准确来说应该是矩阵`)相乘的必要条件, 我们还知道，乘法的转置是转置的乘法，因此

$N' \cdot T' = N^T * G^T * (M * T) = N^T * (G^T * M) * T = 0$

因为 `N` 与 `T` 是正交的, 因此 `N` 与 `T` 的点积为零，因此, 如果

$(G^T * M) = I$

则

$$N^T * (G^T * M) * T = N^T * I * T = 0$$

这正是我们想要的, 因此，我们可以根据 `M` 计算 `G`, 显然需要:

$$G^T = M^{-1}$$

则

$$(G^T)^T = G = (M^{-1})^T$$

因此，转换法线的正确矩阵是 `M` 矩阵的逆的转置, `OpenGL` 会在 `gl_NormalMatrix` 中为我们计算这个矩阵.

本节开头提到，在某些情况下使用 `Modelview` 矩阵是可行的, 只要模型视图的 `3x3` 左上方子矩阵是正交的，因为正交矩阵的逆 = 正交矩阵的转置, 因此我们可以得到:

$$G^T = M^{-1} = M^T$$

则

$$G = M$$

什么是正交矩阵呢? (这里我没有使用原文, 因为正交矩阵的性质很明确, 简单概括一下)

若A 为正交矩阵, 则

- A 的 行向量 与 列向量为单位正交向量组
- $|A| = \pm 1$

那么我们什么时候才能确定 M 是正交的呢? 由于我们的几何操作仅限于旋转和平移时，即在 `OpenGL` 应用程序中只使用 `glRotate` 和 `gl_Translate`，而不使用 `glScale`, 这些操作可确保 `M` 是正交的.
注: `gluLookAt` 也会创建一个正交矩阵!

```
首先，Translate变换矩阵矩阵并不是正交矩阵，这部分我认为原作者是错误的
而 Translate变换 也不是正交变换，严格而言，应该属于仿射变换
但是 Translate 变换是否会破坏正交性? 这部分的内容是值得商榷的，如下
- 直觉上，假设向量 N 与 T 是正交，那么二者经过同样的平移后，必然仍保持正交
- 线性代数层面，平移变换矩阵矩阵并不是正交矩阵，假如我们对 N 与 T 左乘同一个正交矩阵，将导致变换后的 N' 与 T' 的内积不再为0，意味着他俩不再垂直
- 所以，问题出在哪里呢? 百思不得其解!
```

Chapter 13 - homework 5

这里主要总结 `Ray Tracing` 中 `屏幕, NDC 和 投影平面` 之间的坐标关系, 假设

- 屏幕的宽度为 `scene.width` , 长度为 `scene.length` , 宽高比为 `imageAspectRatio`
- `NDC` 中, `xy` 的范围均为 `[-1,1]` , 其 `z` 坐标为0
- 相机的位置为 `(0, 0, 0)` , 竖向可视角度为 `fov`

NDC 中每个像素的中心坐标

- `屏幕坐标系` 的原点为左下角, 坐标系中点的坐标为

$$(i, j), \text{ 其中 } i \in [0, scene.width - 1],\ j \in [0, , scene.length - 1]$$

- 将 `屏幕坐标系` 中的点转换到以左下角为原点, $x, y \in [0, 2]$ 范围的坐标系

$$x'_{ndc} = \frac{2 \times i}{scene.width}$$
$$y'_{ndc} = \frac{2 \times j}{scene.height}$$

为了确保计算的是像素中心而不是边缘, 需要调整为：

$$x'_{ndc} = \frac{2 \times (i + 0.5)}{scene.width}$$
$$y'_{ndc} = \frac{2 \times (j + 0.5)}{scene.height}$$

- `NDC` 的 `xy` 坐标的范围为 `[-1, 1]` , 因此上述坐标还需要继续进行转换

$$x_{ndc} = \frac{2 \times i}{scene.width} - 1$$
$$y_{ndc} = \frac{2 \times j}{scene.height} - 1$$

由于 NDC 的原点为左上角, 因此, y 坐标的起始位置为左上角而不是右下角, 因此, 坐标需要调整为：

$$x_{ndc} = \frac{2 \times (i + 0.5)}{scene.width} - 1$$
$$y_{ndc} = 1 - \frac{2 \times (j + 0.5)}{scene.height}$$

由 NDC 转换到投影平面

- 投影平面坐标系的特点
    - 保持宽高比与屏幕一致
    - `z` 坐标为 `-1` (习惯性做法, 简化计算和保持一致性, 这同时意味着 NDC 和 投影平面的间距为1), `y` 坐标的范围与相机的可视范围一致(根据宽高比可计算得 `x` 坐标的范围)
    - 坐标原点为投影平面的中心, 且 `y` 坐标向下为正增长方向
- 假设 `scale = tan(fov / 2)`

$$y_{proj} = y_{ndc} * scale * 1$$
$$x_{proj} = y_{proj} * imageAspectRatio = y_{ndc} * scale * imageAspectRatio$$

$primary\ ray$ 的方向向量为：

$$dir = normalize(x_{proj}, y_{proj}, -1)$$

概念说明

光栅化与射线追踪的区别

光栅化的流程：

1. **从世界坐标到标准设备坐标 (NDC)**：
    - 光栅化过程会先将三维模型的顶点从世界坐标系经过**投影变换**（例如透视投影矩阵），转化到标准设备坐标系（NDC，范围是 `[-1, 1]`）。
2. **从标准设备坐标到屏幕坐标**：
    - 将 NDC 映射到屏幕坐标（像素空间），这一步通常是简单的线性缩放。
3. **光栅化生成像素**：
    - 在屏幕坐标中，光栅化将几何形状划分为像素，生成像素的深度、颜色等信息。

射线追踪的流程：

1. 射线追踪并不直接进行从世界坐标到 NDC 的变换。
2. 相反，它是从屏幕空间（像素空间）出发，生成每个像素的射线，并通过这些射线追踪场景中的物体。

这种区别是因为两者的核心目标不同：

- 光栅化需要快速将三维几何投影到二维平面。
- 射线追踪需要通过屏幕像素生成光线，去计算它与三维物体的交点。

术语解释

屏幕坐标

- **定义**：
  - 通常指屏幕上的像素坐标 `(i, j)`，范围是 `[0, width-1]` 和 `[0, height-1]`。
  - 这是实际屏幕或图像帧缓存的分辨率空间中像素的离散索引。
- **在光栅化中的地位**：
  - 屏幕坐标在光栅化中是最终的结果，它是从三维空间投影得到的二维空间坐标。
- **在光线追踪中的地位**：
  - 在光线追踪中，屏幕坐标是**光线生成的起点**。我们需要根据这些像素位置计算出每条射线的方向，因此屏幕坐标是生成射线的输入。

投影平面坐标

- **定义**：
  - 指的是屏幕像素在虚拟的投影平面（通常位于 `z = -1`）上的对应位置。这是一个连续的二维坐标空间，通常以**标准化设备坐标 (NDC)** 为基础，并根据视场角 (FOV) 和宽高比 (aspect ratio) 进行缩放。
- **在光栅化中的地位**：
  - 投影平面坐标隐含在投影变换矩阵的结果中，通常在标准设备坐标 (NDC) 中直接使用。
- **在光线追踪中的地位**：
  - 光线追踪中，投影平面坐标是生成射线的中间结果，用于确定每条射线的方向向量。它通过从屏幕坐标映射到 NDC，再转化到投影平面上的点来生成。

光线追踪中的标准术语

在光线追踪的技术文档或教材中，相关的标准术语包括：

1. **标准设备坐标 (Normalized Device Coordinates, NDC)**：
    - 标准化到 `[-1, 1]` 范围的二维坐标，是一个比较常用的术语。
2. **射线方向 (Ray Direction)**：
    - 从摄像机起点指向场景中像素对应的投影平面点的向量，通常使用“direction”描述。
3. **相机空间 (Camera Space)** 或 **视图空间 (View Space)**：
    - 表示从摄像机出发的坐标系。

"屏幕坐标"和"投影平面坐标"的非正式性

1. **屏幕坐标**：
    - 并不严格特指像素空间，有时也可泛指“图像平面”上的某个点。
2. **投影平面坐标**：
    - 这种描述只是为了方便理解光线追踪的生成逻辑，因为投影平面本身并不是真实存在的物理平面。

尽管如此，这两个术语在光线追踪的教程和代码实现中非常常见，特别是在解释如何从屏幕像素生成光线时，它们是直观而有效的辅助术语。

Chapter 16 - homework 7

关于半球均匀采样

假设 θ 是极角, 如果直接对 $\theta \in [0, \pi/2]$ 采样, 由于$sin\theta$ 相对 θ 是非线性的, 因而, 虽然 θ 是均匀的, 但是实际反应到 $sin\theta$ 却是非均匀的

未完...