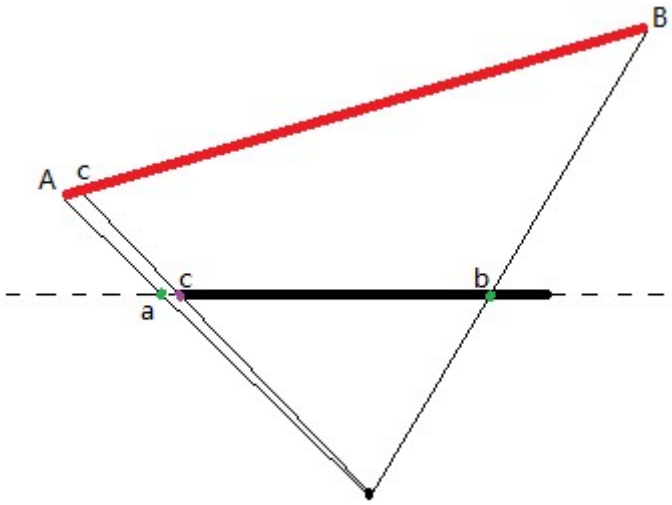
六、CVV 和 Clip Space

我们可以发现前一章的屏幕左边原点是左下角，同时对应到三维空间之后，仍然将屏幕左下角作为原点，这样使用起来是有一点不方便的。通常(其实我所知道的就是 Opengl 和 Direct3D 了)会将屏幕的中心作为原点，X,Y 取值范围[-1,1],同时往垂直平面的方向延伸出去作为 z 轴，取值范围同样也是[-1,1]。不过 Direct3D 是以屏幕往里方向为 Z 轴正方向，而 Opengl 是以屏幕向外方向为 Z 轴正方向，（在前面几个小节里面我们将忽略 Z 轴，因为我们暂时只关心二维屏幕）。这个(x,y,z) ∈[-1,1]的空间叫做 Canonical View Volume，简称 CVV。

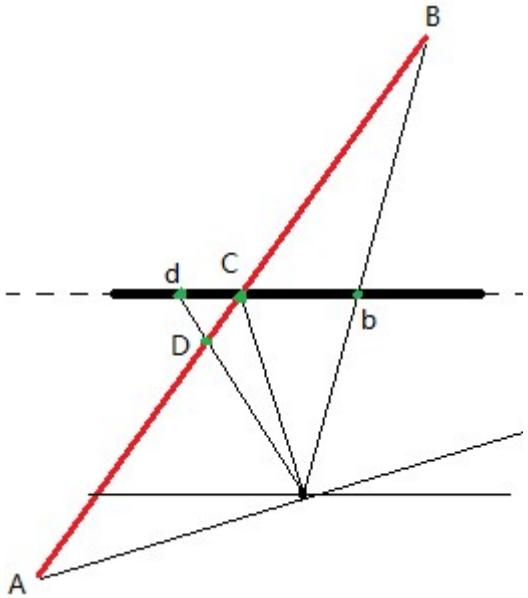
1. 裁剪的必要性

第一种情况：



如图所示，这种设线段 AB 在屏幕上的投影为 ab，并且屏幕实际显示范围为加粗的这一段。可以看到点 A 的投影点 a 已经超出了屏幕的最大显示范围，所以点 a 实际上是无法显示的。实际上在屏幕左侧能显示 AB 线段的极限为 C 点，所以我们需要把线段 AB 裁剪成线段 CB。

第二种情况：



如图所示，如果有一顶点的深度值小于 near 时，是不应该显示出来的，实际上像这种情况我们同样需要把 AB 线段裁剪掉。注意：这里只要一个顶点的深度值小于 near

则该点就不应该投影了，有的人可能会觉得仅仅是像点 A 这样深度值小于等于 0 才无法投影，实际上像 D 点是进入了相机内部了，这个 D 点是不应该被投影到屏幕上的，也就是我们要求的投影方式必须是从 P->near->相机原点这样的路径才是正确的投影。

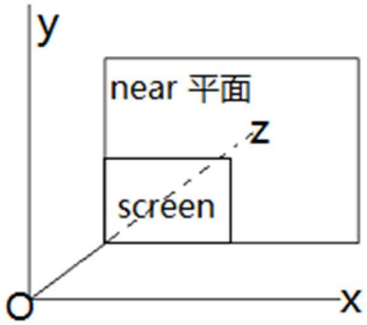
以上两种情况已经说明了裁剪的必要性。

2. 上一章投影计算的不完整

   在上一章中，我们投影使用的公式是:

$$\begin{cases} x' = \dfrac{nx}{z} \\ y' = \dfrac{ny}{z} \end{cases}$$

   是在如下的坐标系下进行投影计算:



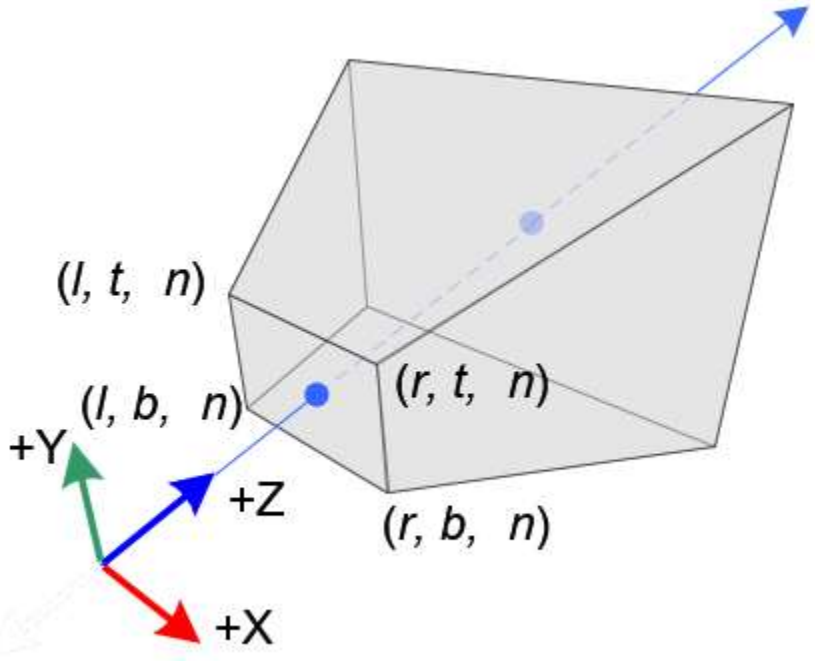
   在这里我们让屏幕和 near 平面重合，则多边形在 near 平面的投影如果出现在屏幕的区域就将其绘制出来，如果不在屏幕区域则不绘制，所以上一章我们缺少了裁剪这一步骤。完整的绘制流程应该是先将三角形投影在 near 平面，然后使用屏幕作为一个裁剪窗口对投影之后的三角形进行裁剪。但是后面我们将会介绍在 CVV 空间中的裁剪，这个裁剪不仅仅包含了在屏幕的上下左右(x,y 轴方向)的裁剪，还包含了前后(z 轴方向)的裁剪，这样可以使得裁剪功能更为完善。

3. 屏幕坐标的统一

   我们把屏幕的宽高统一一下，使用[-1,1]表示屏幕的最左到最右和最下到最上。也就是原来屏幕的 x 取值范围是[0,Width-1]，y 取值范围是[-1,1]。这样表示的话可以使得我们不必关系一个点的具体像素位置，而仅仅使用[-1,1]的一个值表示相对位置，这样的话一个多边形在同等宽高比的不同分辨率屏幕上面可以得到相似的多边形。在绘制过程中需要将[-1,1]表示的坐标转换成屏幕像素坐标。

4. 视锥体(Frustum)

   我们在 near 平面选择一个矩形区域作为屏幕映射区域，需要给出该区域的 Left、Right、Bottom、Top，这组参数限制了被投影区域的面积，同时额外给出一组 z 轴范围限制参数 near 和 far，这个六个参数可以围成一个六面体，**其中 near>0，并且有 left<right，bottom<top**，如下图所示:



上图中与 z 轴垂直的靠近原点的较小的面为 near 平面，另一较大的面则为 far 平面，并且(l,t,n), (l,b,n) (r,b,n) (r,t,n)围成的这个矩形为屏幕在这个坐标系中的映射。所以从现在开始我们想要正确的在屏幕上面显示的话需要两个步骤：1、先执行投影(和上一章一样的方法)。2、将投影结果转换成使用([-1,1])表示的坐标。3、进行裁剪。4、绘制([-1,1]转换成屏幕的像素坐标)。实际上做完第一步之后，后面的几个步骤顺序是可以交换的，甚至第 2、4 步都是不必要的，可以直接将投影结果转换成屏幕的像素坐标，但是为了流程的规范化，我们先按照这个步骤来做。

1) xy 坐标的变换

现在我们讨论一下将投影结果的坐标转换成[-1,1]这种坐标，假设现在已知一个三维空间的点被投影成(x',y')，我们要对其进行转换，这时候只需要一个很简单的线性插值就行了(参考第四章)。我们现在先处理 x，已知 x=left 时转换后的值为-1，x=r 时,转换后的值为 1，所以我们可以直接写出 x 和 y 的转换公式:

$$\begin{cases} x'' = \dfrac{2x' - (r+l)}{r-l} & ① \\ y'' = \dfrac{2y' - (t+b)}{t-b} & ② \end{cases}$$

现在我们就完整的推导一次 x,y 的转换:

空间中一点(x,y,z)经过投影之后变为(nx, ny, z)(齐次坐标)，这时候我们把$\frac{nx}{z} = x'$以及$\frac{ny}{z} = y'$分别带入上面两个公式①②得到$x'' = \frac{2nx}{(r-l)z} - \frac{r+l}{r-l}$，$y'' = \frac{2ny}{(t-b)z} - \frac{t+b}{t-b}$，即坐标$(\frac{2nx}{(r-l)z} - \frac{r+l}{r-l}, \frac{2ny}{(t-b)z} - \frac{t+b}{t-b})$就是最终在屏幕上的坐标。这只是一个普通坐标，他的一种齐次坐标表示方法为$(\frac{2nx}{(r-l)z} - \frac{r+l}{r-l}, \frac{2ny}{(t-b)z} - \frac{t+b}{t-b}, 1)$，现在把每一项乘

以 z 可以得到同一个坐标的另一个表达方法$\left(\frac{2n}{r-l}-z\frac{r+l}{r-l},\frac{2ny}{t-b}-z\frac{t+b}{t-b},z\right)$,这里的最后一个分量ω取的是点的原始 Z 值很重要，后面的裁剪和透视校正插值都依靠这个齐次坐标进行。(一定要习惯齐次坐标的表示方法，这个坐标和前面的坐标表示的是同一个点，用齐次坐标会有其他方面的便捷，所以我才坚持使用齐次坐标)。

这次的代码依然没有加入裁剪，所以设置点的时候千万不要超出边界，因为 FillTriangle 函数只能正确绘制在屏幕范围之内的三角形，超出边界将会导致程序崩溃。

代码在/chaper6/CVV1/main.cpp 中

```
//投影函数,分别传递一个点和near平面，计算出新投影点并将结果保存至一个齐次坐标中
void Projection(Point3& p,double l,double r,double b,double t,double n, Point3& result)
{
	result.X = (2 * n * p.X)/(r-l)-p.Z*(r+l)/(r-l);
	result.Y = (2*n * p.Y)/(t-b)-p.Z*(t+b)/(t-b);
	result.Z = p.Z;
}
```

上述代码完成投影计算，得到的坐标是用$\left(\frac{2nx}{r-l}-z\frac{r+l}{r-l},\frac{2ny}{t-b}-z\frac{t+b}{t-b},z\right)$表达的一个齐次坐标，在得到计算结果后：139 行至 143 行代码调用 Projection 函数进行投影，并将投影之后得到的齐次坐标转换成正常的坐标(用[-1,1]区间表示的坐标)。

```
for (int i = 0; i < 3; i++)
{
	Projection(ps[i],-320,320,-240,240, 25,result[i]);
	Polygon[i].X = result[i].X / result[i].Z;//将齐次坐标转换为普通的坐标
	Polygon[i].Y = result[i].Y / result[i].Z;
 }
```

然后在第 32 行中将[-1,1]区间表达的坐标还原成屏幕像素坐标，使用的同样是线性插值(将 X、Y 的[-1,1]表示插值成[0,WIDTH]和[0, HEIGHT]表示)。

2) 深度的变换

前一小节我们只讨论了顶点 x、y 的变换，对顶点深度没有进行处理。现在我们对深度进行处理，这次我们同样使用线性插值将深度值从[n,f]插值成[-1,1]。一旦说了使用线性插值，那么可以直接写出公式了$z'=\frac{2z-(f+n)}{f-n}$,然后我们将他和上一小节的坐标合并，得到:

$$(x,y,z)\overset{投影}{\Longrightarrow}\left(\frac{2nx}{(r-l)z}-\frac{r+l}{r-l},\frac{2ny}{(t-b)z}-\frac{t+b}{t-b},\frac{2z}{f-n}-\frac{f+n}{f-n},1\right)$$

对于这个齐次坐标，给各个分量同时乘以 z 得到用四元齐次坐标表示的三维点:

$$\left(\frac{2nx}{r-l}-z\frac{r+l}{r-l},\frac{2ny}{t-b}-z\frac{t+b}{t-b},\frac{2z^2}{f-n}-z\frac{f+n}{f-n},z\right)$$

这个坐标的前两个分量可以表示出顶点被投影之后的坐标，第三个分量表示原始的 Z 值(也叫做深度值)，第四个分量ω取点的原始 Z 值。深度值有什么作用呢？可以用来消隐，关于消隐我们后面会讲。这个公式虽然能够实现正确的消隐，但是对于裁剪来说却并不是那么方便，因为 z 分量和ω分量不是线性关

系，我们裁剪的时候如果是对线性关系进行裁剪的话会很方便。
适用的深度值的特点为：
1.是关于 Z 值的单调递增函数。
2.分别当 z=n 和 z=f 时,变换结果应该分别等于-1 和 1。
3.Z''分量应该和顶点的原始 Z 值成线性关系。

我们可以证明$\frac{2z-(f+n)}{f-n}$是关于 z 的单调递增函数(令 g(x)= $\frac{2z_2-(f+n)}{f-n}$因为 f>n，设$Z_2>Z_1$，则$g(Z_2)-g(Z_1)=\frac{2z_2-(f+n)}{f-n}-\frac{2z_1-(f+n)}{f-n}$>0，证明出 g(z)关于 z 单调递增)，但是这种形式并不是很友好，因为这个坐标变成齐次表达之后，有了一个项包含了Z^2，而且变量之间不再是线性关系。所以我们换个式子使其满足上述三个条件:使用 z''表示齐次坐标中的 z 分量，z'表示$\frac{z''}{z}$，z 表示原始顶点的 z 值。

$$z''=Ax+By+Cz+D$$

应该关于 x,y,z 都成线性关系，这样可以方便的写成矩阵表达形式(如果读者不熟悉矩阵的话，可以不用管，我们现在的目的就是为了使式子计算起来更加方便一点点，毕竟计算平方得多乘一次)。通过上面的分析我们可以知道 z''是和 x,y 无关的，所以 AB 应该都等于 0。即公式为:

$$z''=Cz+D$$

我们只需要确定出 C 和 D 即可，怎么得到 CD 的值呢？回到之前说的点，我们知道 z'是关于 z 的递增函数，并且当 z=n 时,z'=-1,当 z=f 时,z'=1。

$$z'=C+\frac{D}{z}$$

把 z=n 和 z=f 分别带入上式得:

$$\begin{cases}C+\frac{D}{n}=-1\\ C+\frac{D}{f}=1\end{cases}$$

联立上述两式可以解出:

$$\begin{cases}C=\frac{f+n}{f-n}\\ D=\frac{2fn}{n-f}\end{cases}$$

$$z'=\frac{f+n}{f-n}+\frac{2fn}{(n-f)z}$$
$$z''=\frac{f+n}{f-n}z+\frac{2fn}{n-f}$$

并且我们不难发现 z'是关于 z 单调递增的(因为$\frac{2fn}{(n-f)}<0$，$\frac{1}{z}$是一个单调递减函数，一个单调递减函数乘以一个负数变成单调递增函数)，当然 z≠0(后面介绍裁剪的时候会将这种点裁剪掉)，并且当 z=n 时 z'=-1，z=f 时 z.=1,符合我们对深度表达的两点要求，这个深度值叫做**伪深度**，因为他和原来的深度值已经不再是线性关系，产生了变形(注意：虽然$z'=\frac{z''}{z}$和 z 不是线性关系，但是$z''=$

$\frac{f+n}{f-n}z+\frac{2fn}{n-f}$和 z 仍然是线性关系)。

经过上述变化，**我们只需要绘制出 x,y,z 都在[-1,1]区间的顶点，任何不在这个范围之内的顶点都将会被裁剪。**

5. 裁剪空间

(很硬的两个小节(4、5 小节)，在计《算机图形学(OpenGL 版)》中没有讲述这节知识，我推导的这个结论是被直接使用的，这本书的作者是大佬，默认读者懂了这些东西，可能对于初学者或者数学基础不太好的人来说搞不明白书上到底为什么要这么写)

接下来就要介绍一下裁剪了，裁剪可以使得空间中的一个三角形最终只保留需要绘制的部分，超出 Frustum 的部分将会被裁剪掉。因为 Frustum 是个凸多面体，在这里我们会使用类似于 Sutherland-Hodgman 的思想，使用 Frustum 的六个面依次对多边形进行裁剪。表面看起来 Frustum 是个多面体，实际上我们在裁剪中最关心的是裁剪时的边界条件。我们先使用 Frustum 的一个平面作为边界对三角形进行裁剪，如果你还记得第三章的内容的话，应该知道，我们会先把三角形的三个顶点依次记录，然后判断每一条边的裁剪情况(平凡拒绝、平凡接受、裁剪)。

我们先定义一个裁剪空间，裁剪空间是一个抽象的空间，因为我们用裁剪的齐次坐标有四个分量，可以认为裁剪空间是一个四维空间(四维空间很难想象出来，所以)，我们在裁剪空间定义一些边界条件用于做裁剪工作。

首先要证明的是相机坐标系中两点 AB 连接成的直线在裁剪空间中也是直线(我们需要证明齐次坐标在四维空间中各个分量仍然是线性关系):

设在相机坐标系中有两点$A(x_a,y_a,z_a)$和$B(x_b,y_b,z_b)$,则我们可以用一个点向式方程表示出这条直线:

$$\frac{x-x_a}{x_b-x_a}=\frac{y-y_a}{y_b-y_a}=\frac{z-z_a}{z_b-z_a}$$

假设令$\frac{x-x_a}{x_b-x_a}=\frac{y-y_a}{y_b-y_a}=\frac{z-z_a}{z_b-z_a}=t$，则可以得到一个参数方程

$$\begin{cases}x=x_a+(x_b-x_a)t\\y=y_a+(y_b-y_a)t\\z=z_a+(z_b-z_a)t\end{cases}$$

因为$t=\frac{x-x_a}{x_b-x_a}$，所以这里面的 t 表示点$P(x_p,y_p,z_p)$到 A 和点 B 到 A 的距离比值。如果你还记得第四章的内容，那么你对 t 值一定很熟悉，这里 t 同时也表示了 B 点的权值。

现在假设在相机坐标系中上述的三个点 A、B、P 经过变换后得到在四维空间中的点(齐次坐标)分别为

$$\begin{cases}A'=(\frac{2nx_a}{r-l}-z_a\frac{r+l}{r-l},\frac{2ny_a}{t-b}-z_a\frac{t+b}{t-b},\frac{f+n}{f-n}z_a+\frac{2fn}{n-f},z_a)\\B'=(\frac{2nx_b}{r-l}-z_b\frac{r+l}{r-l},\frac{2ny_b}{t-b}-z_b\frac{t+b}{t-b},\frac{f+n}{f-n}z_b+\frac{2fn}{n-f},z_b)\\P'=(\frac{2nx_p}{r-l}-z_p\frac{r+l}{r-l},\frac{2ny_p}{t-b}-z_p\frac{t+b}{t-b},\frac{f+n}{f-n}z_p+\frac{2fn}{n-f},z_p)\end{cases}$$

若四维点 P'满足于:

$$\frac{x_{P\prime}-x_{A\prime}}{x_{B\prime}-x_{A\prime}}=\frac{y_{P\prime}-y_{A\prime}}{y_{B\prime}-y_{A\prime}}=\frac{z_{P\prime}-z_{B\prime}}{z_{B\prime}-z_{A\prime}}=\frac{\omega_{P\prime}-\omega_{B\prime}}{\omega_{B\prime}-\omega_{A\prime}}$$

则说明 P'在四维空间中在四维直线 A'B'上面，这就证明了相机坐标系中两点 AB 连接成的直线在裁剪空间中也是直线。我们把

$$x_{A'}=\frac{2nx_a}{r-l}-z_a\frac{r+l}{r-l}$$

…(省略)…

$$x_{P'}=\frac{2nx_p}{r-l}-z_p\frac{r+l}{r-l}$$

其中$x_p=x_a+(x_b-x_a)t, y_p=y_a+(y_b-y_a)t,\ z_p=z_a+(z_b-z_a)t$

等一系列值带入$\frac{x_{P\prime}-x_{A\prime}}{x_{B\prime}-x_{A\prime}}=\frac{y_{P\prime}-y_{A\prime}}{y_{B\prime}-y_{A\prime}}=\frac{z_{P\prime}-z_{B\prime}}{z_{B\prime}-z_{A\prime}}=\frac{\omega_{P\prime}-\omega_{B\prime}}{\omega_{B\prime}-\omega_{A\prime}}$中，得到：

$$\frac{x_{P'}-x_{A'}}{x_{B'}-x_{A'}}$$

$$=\frac{\left[\frac{2n[x_a+[x_b-x_a]t]}{r-l}-[z_a+[z_b-z_a]t]\frac{r+l}{r-l}\right]-\left[\frac{2nx_a}{r-l}-z_a\frac{r+l}{r-l}\right]}{\left[\frac{2nx_b}{r-l}-z_b\frac{r+l}{r-l}\right]-\left[\frac{2nx_a}{r-l}-z_a\frac{r+l}{r-l}\right]}$$

$$=\frac{\frac{2nx_a+2ntx_b-2ntx_a}{r-l}-\frac{rz_a+lz_a}{r-l}-\frac{rtz_b+ltz_b}{r-l}+\frac{rtz_a+ltz_a}{r-l}-\frac{2nx_a}{r-l}+\frac{rz_a+lz_a}{r-l}}{\frac{2nx_b}{r-l}-\frac{rz_b+lz_b}{r-l}-\frac{2nx_a}{r-l}+\frac{rz_a+lz_a}{r-l}}$$

$$=\frac{2nx_a+2ntx_b-2ntx_a-2nx_a-rz_a-rtz_b+rtz_a+rz_a-lz_a-ltz_b+ltz_a+lz_a}{2nx_b-2nx_a+rz_a-rz_b+lz_a-lz_b}$$

$$=t=\frac{y_{P\prime}-y_{A\prime}}{y_{B\prime}-y_{A\prime}}=\frac{z_{P\prime}-z_{B\prime}}{z_{B\prime}-z_{A\prime}}=\frac{\omega_{P\prime}-\omega_{B\prime}}{\omega_{B\prime}-\omega_{A\prime}}$$

我们可以发现，原来在三维空间中的直线经过我们的变换之后在齐次空间中仍然是直线，并且保留了原来的比例关系 t，即齐次向量和原来的三维向量仍然保持着如下的关系：

$$\frac{\overrightarrow{A'P'}}{\overrightarrow{A'B'}}=\frac{\overrightarrow{AP}}{\overrightarrow{AB}}$$

这个性质使得我们可以在齐次空间中进行裁剪，并且因为第四章第五小节中说的那样，裁剪之后要计算裁剪点的插值结果，因为存在上述的比例关系，使得我们在齐次空间中裁剪可以得到和三维相机坐标系一样的裁剪插值结果(如果你能完全明白第四章的内容，**可以发现这仍然是一维数轴的推广，只要坐标的(x,y,z,ω)中的各个分量保持线性关系即可，甚至我们可以推广到任意维度的直线线性插值**)。

6. 在裁剪空间中对直线进行裁剪

只要知道 t 值，我们就可以在裁剪空间中对直线进行插值，假设现在在裁剪空间中有线段 AB 需要插值，设插值点为 P，则：

$$x=x_a+(x_b-x_a)t$$
$$y=y_a+(y_b-y_a)t$$
$$z=z_a+(z_b-z_a)t$$
$$\omega=\omega_a+(\omega_b-\omega_a)t$$

假设现在我们使用$\frac{x}{\omega}=-1$来选择插值点，则有：

$$\frac{x_a+(x_b-x_a)t}{\omega_a+(\omega_b-\omega_a)t}=-1①$$

那么我们很容易计算出 t 值，知道 t 值之后，可以很容易的为 p 点插值。

现在我们假设用$\frac{x}{\omega}\geq -1$作为裁剪边界，则有:

$$\frac{x}{\omega}\geq -1$$

$$\begin{cases}x+\omega\geq 0(\omega>0)②\\ x+\omega\leq 0(\omega<0)③\\ \omega=0\text{ 时无法计算}④\end{cases}$$

并且当ω=0 时，无法计算，考虑如下的两点 A(-2,y,z,3)和 B(4,y,z,-3)，如果用①②③作为裁剪边界判断条件，则点 A 被保留,点 B 被裁剪，在计算 t 的过程中发现 t 无法被计算出来，即$\frac{-2+(4-(-2))t}{3+(-3-3)t}=\frac{-2+6t}{3-6t}=-1$无解，所以要避免出现无解的情况。由①可知，$t=\frac{x_a+\omega_a}{x_a+\omega_a-(x_b+\omega_b)}$，当分母为 0 时，可以认为这个线段和边界没有交点(裁剪边界不一定是一个平面，可以是一个方程)，这时候我们可以做一个小小的处理，令分母永远不等于 0，对应于边界条件②③④，我们把接受条件修改为:

$$x+\omega\geq 0$$

删除掉边界条件的③④，只保留②。

这样当需要计算边界交点时，永远不会出现$x_a+\omega_a-(x_b+\omega_b)=0$的情况，因为只有一种情况需要计算交点：一点被接受，另一点被拒绝，被接受的点 x+ω≥0,被拒绝的点 x+ω<0，即只有$x_a+\omega_a-(x_b+\omega_b)>0$(A 点被接受,B 点被拒绝)或者$x_a+\omega_a-(x_b+\omega_b)<0$(B 点被接受,A 点被拒绝)，对于这样的边界条件，细心的同学肯定会问：这个条件只对ω>0 能做出正确的判断，当ω<0 时，就会把原本该拒绝的点接受，而原本该接受的点拒绝了。比如上面的 B(4,y,z,-3)按照计算应该是被拒绝的，但是使用$x+\omega>0$作为接受条件的话，B 点被接受了，而对于某一点

P(2,y,z,-3)

来说，$\frac{x}{\omega}=\frac{2}{-3}\geq -1$，应该被接受的，但是使用$x+\omega\geq 0$作为边界，该点被拒绝了，那么这个修改过的边界条件还能用吗？答案是：**能。因为我们使用相机坐标系的 z 值作为齐次坐标的ω，ω小于等于 0 的本来就该拒绝(因为我们的 near 平面必须大于 0)，即对于上述的点 B 和 P 都应该被拒绝**，当我们使用$x+\omega>0$作为边界时，原本就该被拒绝的某些点被接受了，比如(2,y,z,-3)这个点。在后续判断齐次坐标的 z 分量时,我们会把这些被错误接受的点也拒绝掉。对于裁剪边界$\frac{x}{\omega}\leq 1,\frac{y}{\omega}\geq -1,\frac{y}{\omega}\leq 1$，$\frac{z}{\omega}\leq 1$也是一样的处理方法。

接下来我们只单独再需要讨论一下$\frac{z}{\omega}\geq -1$作为裁剪边界的情况就能把六个裁剪边界全部搞定，前面几步和上面类似:

$\frac{z}{\omega}\geq -1$可以推出下面的判断条件

$$\begin{cases} z+\omega \geq 0(\omega > 0) \\ z+\omega \leq 0(\omega < 0) \\ \omega = 0 \text{ 时无法计算} \end{cases}$$

我们同样修改边界条件为令$\omega \leq 0$的直接被拒绝，所以上面的边界接受条件可以被修改为:

$$z+\omega \geq 0 \text{ 且} \omega > 0$$

即拒绝条件为:

$$\begin{cases} z+\omega < 0(\omega > 0) \\ \omega \leq 0 \end{cases}$$

我们同样要判断下面使用 Z''作为其次坐标的 Z 分量标记，用 Z 作为点在相机坐标系中的原始 Z 值,

我们知道

$$z'' = \frac{f+n}{f-n}z + \frac{2fn}{n-f}$$

其中$\frac{f+n}{f-n} > 0, \frac{2fn}{n-f} < 0$，可以发现，当$z \leq -\frac{2fn}{n-f}$时,$z'' \leq 0$，即当$\omega \leq 0$时，必然有$z+\omega \leq 0$。所以我们可以得到唯一的边界条件为:

$$z+\omega \geq 0$$

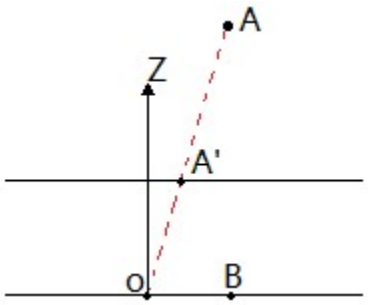
这样不会出现和其他五个边界条件类似的当ω≤0 时出现的错误接受某些点的情况了。在确定了直线需要裁剪之后(一个点被拒绝，一个点被接受)，我们就可以使用$\frac{z_a+(z_b-z_a)t}{\omega_a+(\omega_b-\omega_a)t} = -1$计算出 t 值，进行顶点属性插值。

实际上在齐次空间中裁剪并不是唯一的裁剪方法，我们完全可以在把三角形先在相机坐标系中的 Frustum 中裁剪，使用一个视锥体作为裁剪窗口。这里本质上是一样的，只是为了和业界采用一样的方法，我才介绍这种裁剪方法，实际上在齐次裁剪空间中裁剪和在相机坐标系中使用视锥体直接裁剪是得到一样的结果。

上面介绍的这个齐次裁剪是在四维空间中进行裁剪的，这个四维的空间我们也叫做**裁剪空间(Clip Space)。**

注意,需要大家区分清楚的是 Canonical View Volume 和 Clip Space(简称 CS)是不同的概念，Clip Spcae 是个四维的空间，我们在里面进行裁剪，CVV 是在 CS 执行了$x = \frac{x''}{\omega}$、$y = \frac{y''}{\omega}$和$z = \frac{z''}{\omega}$之后的三维空间。

其中，把相机坐标系中的点变换到次坐标的过程叫做**透视乘法**，而把齐次坐标变换到 CVV 空间中的过程叫做**透视除法**。我们可以发现，裁剪必须在透视除法之前进行。考虑下面的情况，假设在相机坐标系中的点为 p(1,1,0),我们设 l=f=b=-1，f=r=t=1，f=10，则该点经过透视乘法之后得到 P'为(1,1,-2.2222,0)，此时，我们将无法进行透视除法，因为ω分量为 0，此时无法将 CS 中的 p'变换到 CVV 中。这种情况产生的原因在下图中可以很清楚的看出来:



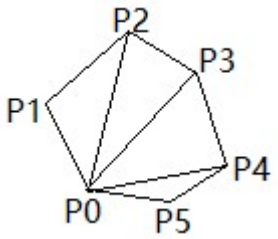
我们知道，求一个点在 near 平面的投影实际上就行将该点和原点 O 连接成一条直线，然后求该直线和 near 平面的交点。而当该点的 Z 值为 0 时，如上图的 B 点，如果直线 OB 和 near 平面平行，则直线 OB 和 near 平面无交点。所以我们在绘制的时候必须进行裁剪(直线使用Frustum裁剪或者在齐次四维空间中裁剪都行)，将 B 点拒绝。

为了方便查看，我把六个边界的裁剪条件做成一个表格

边界	接受条件	接受条件	拒绝条件	t 值方程	t 值
$\frac{x}{\omega} \geq -1$	$x + \omega \geq 0$	$\omega + x \geq 0$	$\omega + x < 0$	$\frac{x_a + (x_b - x_a)t}{\omega_a + (\omega_b - \omega_a)t} = -1$	$\frac{\omega_a + x_a}{(x_a + \omega_a) - (x_b + \omega_b)}$
$\frac{x}{\omega} \leq 1$	$x - \omega \leq 0$	$\omega - x \geq 0$	$\omega - x < 0$	$\frac{x_a + (x_b - x_a)t}{\omega_a + (\omega_b - \omega_a)t} = 1$	$\frac{\omega_a - x_a}{(\omega_a - x_a) - (\omega_b - x_b)}$
$\frac{y}{\omega} \geq -1$	$y + \omega \geq 0$	$\omega + y \geq 0$	$\omega + y < 0$	$\frac{y_a + (y_b - y_a)t}{\omega_a + (\omega_b - \omega_a)t} = -1$	$\frac{\omega_a + y_a}{(y_a + \omega_a) - (y_b + \omega_b)}$
$\frac{y}{\omega} \leq 1$	$y - \omega \leq 0$	$\omega - y \geq 0$	$\omega - y < 0$	$\frac{y_a + (y_b - y_a)t}{\omega_a + (\omega_b - \omega_a)t} = 1$	$\frac{\omega_a - y_a}{(\omega_a - y_a) - (\omega_b - y_b)}$
$\frac{z}{\omega} \geq -1$	$z + \omega \geq 0$	$\omega + z \geq 0$	$\omega + z < 0$	$\frac{z_a + (z_b - z_a)t}{\omega_a + (\omega_b - \omega_a)t} = -1$	$\frac{\omega_a + z_a}{(z_a + \omega_a) - (z_b + \omega_b)}$
$\frac{z}{\omega} \leq 1$	$z - \omega \leq 0$	$\omega - z \geq 0$	$\omega - z < 0$	$\frac{z_a + (z_b - z_a)t}{\omega_a + (\omega_b - \omega_a)t} = 1$	$\frac{\omega_a - z_a}{(\omega_a - z_a) - (\omega_b - z_b)}$

7. 多边形的细分

对于一个 n 边的凸多边形，如果被一个线性边界裁剪，则可能被裁剪成 n 边形或者 n+1 边形。假设现在我们把三角形经过 6 个边界裁剪之后得到一个凸(因为裁剪窗口也是凸多边形)n 边形(9≥n≥3)，则我们可以把这个 n 边形分解成 n-2 个三角形，如下图：



我们可以从起点开始，依次得到△$P_0P_1P_2$、△$P_0P_2P_3$、△$P_0P_3P_4$、△$P_0P_4P_5$。

8. 相关代码

下面我将会放上关于 CS 的相关代码，并作出简单解释。代码在/chapter6/ClipSpace/main.cpp 中。

首先对 Projection 函数稍作修改，参数新增 f，结果新增一个 Z 分量。

```
//投影函数,分别传递一个点和near平面，计算出新投影点并将结果保存至一个齐次坐标中
void Projection(Point3& p,double l,double r,double b,double t,double n,double f, Point4&
result)
{
   result.X = (2 * n * p.X)/(r-l)-p.Z*(r+l)/(r-l);
   result.Y = (2*n * p.Y)/(t-b)-p.Z*(t+b)/(t-b);
   result.Z = (f+n)/(f-n)*p.Z+2*f*n/(n-f);
   result.W = p.Z;
}
```

新增函数 ClipLine 用于完成对线段的裁剪:

```
//对直线进行裁剪,返回-1表示平凡拒绝，返回0表示平凡接受，返回1表示对A点进行裁剪，返回2表示
对B点进行裁剪，最后两个参数result和t表示裁剪点坐标和t值
//其中参数flag从0到5分别表示边界为left、right、bottom、top、near、far
int ClipLine(int flag, Point4& A, Point4& B, Point4& result, double& t)
{
   double aClipCondition;
   double bClipCondition;
   switch (flag)
   {
   case 0:
        aClipCondition = A.W + A.X;
        bClipCondition = B.W + B.X;
        break;
   case 1:
        aClipCondition = A.W - A.X;
        bClipCondition = B.W - B.X;
        break;
   case 2:
```

```
            aClipCondition = A.W + A.Y;
            bClipCondition = B.W + B.Y;
            break;
        case 3:
            aClipCondition = A.W - A.Y;
            bClipCondition = B.W - B.Y;
            break;
        case 4:
            aClipCondition = A.W + A.Z;
            bClipCondition = B.W + B.Z;
            break;
        case 5:
            aClipCondition = A.W - A.Z;
            bClipCondition = B.W - B.Z;
            break;
        default:
            return 0;
        }
        if (aClipCondition < 0 && bClipCondition < 0)//平凡拒绝
        {
            return -1;
        }
        if (aClipCondition >= 0 && bClipCondition >= 0)//平凡接受
        {
            return 0;
        }
        t = aClipCondition / (aClipCondition - bClipCondition);//计算交点的t值
        result.X = A.X + (B.X - A.X) * t;//计算交点
        result.Y = A.Y + (B.Y - A.Y) * t;
        result.Z = A.Z + (B.Z - A.Z) * t;
        result.W = A.W + (B.W - A.W) * t;
        if (aClipCondition < 0)
        {
            return 1;
        }
        else
        {
            return 2;
        }
    }
```

在函数 ClipTriangleAndDraw 中使用类似第三章中的算法对三角形进行裁剪，裁剪完成之后把裁剪剩余的多边形分解成多个三角形，然后把这些三角形一一绘制。程序运行结果如下：