**二维裁剪算法**

1）点的裁剪

2）线段的裁剪（直线段）

3）区域的裁剪（多边形）

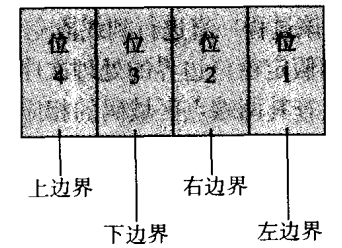
4）曲线的裁剪

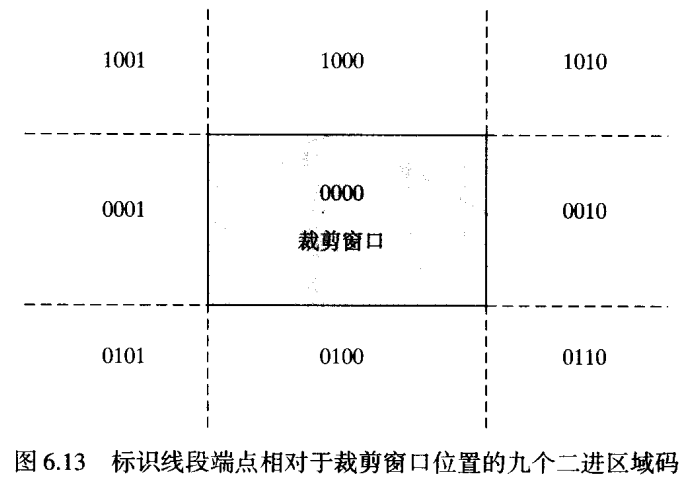
5）文字的裁剪

除非特别声明，我们都假设裁剪区域是一个正则矩形，这些边界与从0到1或从-1到1的规范化正方形的边界对应

线段裁剪方法：

（1）Cohen-Sutherland线段方法





一旦给所有的线段端点建立了区域码，就可以快速判断哪条线段完全在裁剪窗口之内，哪条线段完全在窗口之外。完全在窗口边界内的线段，其两个端点的区域码均为0000，因此保留了这些线段。两个端点的区域码中，有一对相同位置都是1的线段则完全落在裁剪矩形之外，因此丢弃这些线段

如果两个端点的区域码进行逻辑或的结果为0000，则线段完全位于裁剪区域之内

如果两个端点的区域码进行逻辑与的结果不为0000，则线段完全位于裁剪区域之外

对于不能判断为完全在窗口外或窗口内的线段，则要测试其与窗口边界的交点。

**裁剪算法**

裁剪最多应用于观察流水线，目的是为了从场景（二维或三维）中指定部分显示在输出设备上。裁剪也用于对象边界的反走样、实体建模法构造对象，管理多窗口环境及在绘画程序中将图的一部分移动、复制或擦除

对裁剪窗口的规范化边界应用裁剪算法是实现观察流水线裁剪的高效方法，由于可以在裁剪前合并所有的几何和观察变换矩阵并应用于场景描述，因此大大减少了计算量。裁剪后的场景送到屏幕坐标系进行最后的处理。假设裁剪区域为XWmin,XWmax,YWmin,YWmax

图元的裁剪包括：点的裁剪、线段的裁剪、区域裁剪、曲线裁剪、文字裁剪

**（1）点的裁剪**

满足条件：XWmin <= x <= XWmax && YWmin <= y <= YWmax

**（2）线段的裁剪**

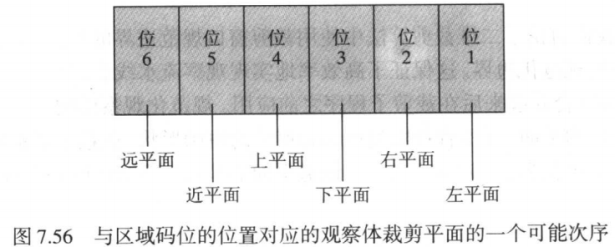
线段的裁剪算法通过一系列的测试和求交计算来判断是否整条线段或其中的某部分可以保存下来。线段与窗口边界的交点计算是线段裁剪函数的耗时部分，因此，减少交点计算是任一线段裁剪算法的主要目标。为此，我们可以先进行测试，确定线段是否完整地在裁剪窗口内部或完整地位于外部。确定一线段是否完整地在裁剪窗口内部是很容易的，但要确认整条线段都在窗口外部就比较困难。如果不能确定一条线段是否完整地在裁剪窗口的内部或者外部，则必须通过进行焦点计算来确定是否该线段有一部分落在窗口内部。目前已经有一些快速的裁剪算法，有些算法是针对二维图形的，有些算法可以很容易地移植到三维应用中。

1）Cohen-Sutherland裁剪算法

这是一个最早开发的快速线段裁剪算法，已经得到了广泛的使用

**三维裁剪：**

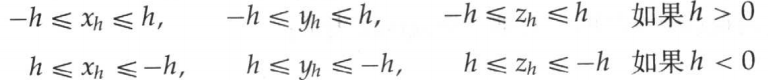
1）点和线段

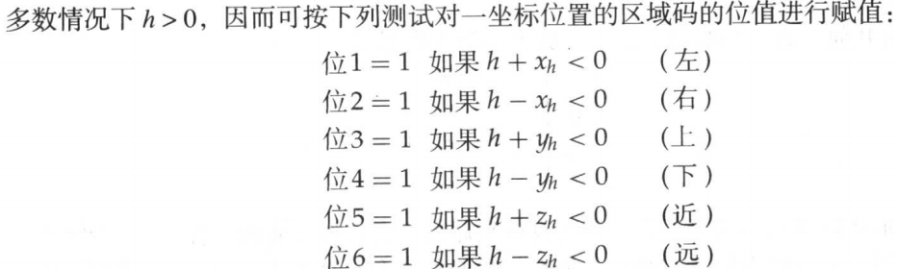


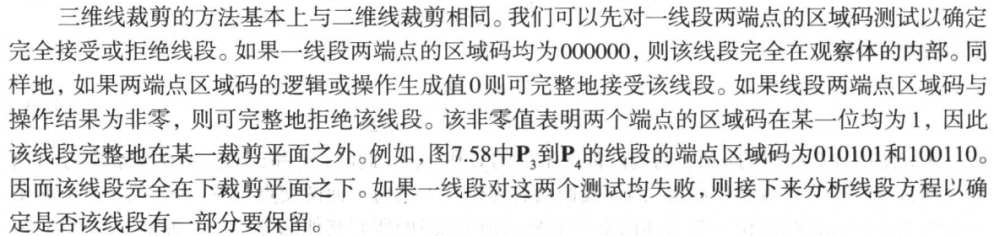
上述方法与之前的二维裁剪一样，只是加上了用于近和远裁剪平面的两个附加条件。然而，对于一个三维场景，我们需对变换到规范化空间的投影坐标使用裁剪子程序。在投影变换后，场景每一个四元素表示。假设用规范化对称立方体的边界进行裁剪，则当一点的投影坐标满足如下六个不等式时该店位于规范化观察体的内部



该齐次参数h不会为0，除非出现错误，但是我们可以再执行区域码过程之前先检查齐次参数为0或一个太小的数的可能性。齐次参数也可以为正或负，因此，假设h!=0，可以将前面的不等式写成下列形式







2）多边形