计算机图形学
Computer Graphics

陈中贵
chenzhonggui@xmu.edu.cn
http://graphics.xmu.edu.cn/~zgchen

Graphics@XMU

第七章

从顶点到片断

第七章 第一节

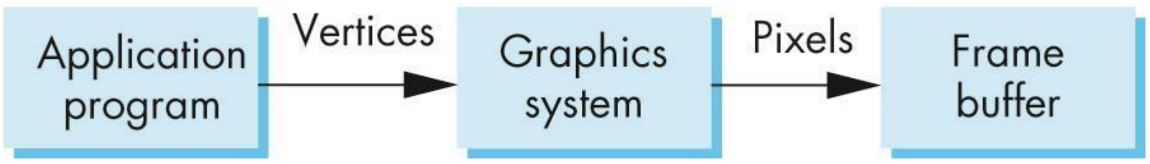
框架与裁剪

基本内容

- 基本实现策略
- 线段裁剪
- 多边形裁剪

图形处理过程

- 计算机图形系统可看作一个黑盒子
  - 输入：程序中定义的顶点和状态量，即几何对象、属性和照相机设置
  - 输出：帧缓冲区里的彩色像素阵列
  - 内部：几何变换、裁剪、明暗处理、隐藏面消除和图元光栅化
    - 至少有两个循环
      - 对每个几何对象进行处理
      - 给颜色缓冲区的每个像素赋颜色值

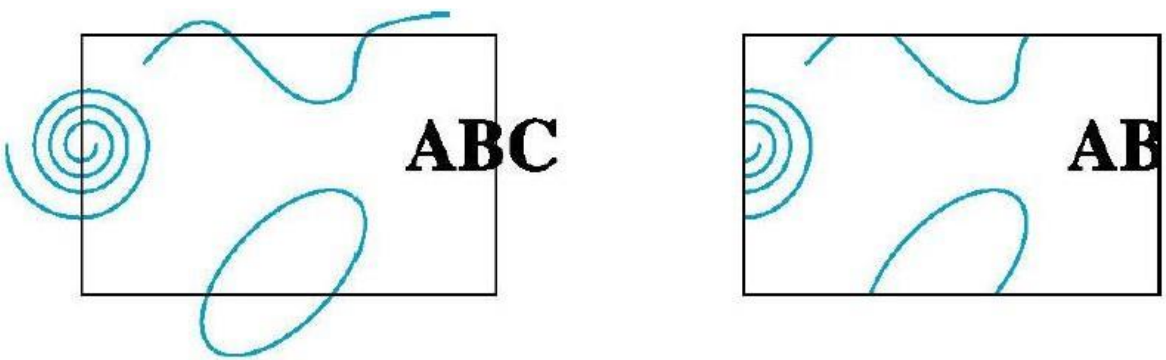


裁剪

- 裁剪器确定哪些图元或图元的哪些部分位于裁剪体（或视景体）内部，即可能会显示在屏幕上
- 三种情形：
  - 接受
  - 拒绝或剔除
  - 裁剪
- 裁剪处理可以发生在渲染流水线的一个或多个地方
  - OpenGL中，在光栅化之前使用一个三维视景体对图元进行裁剪

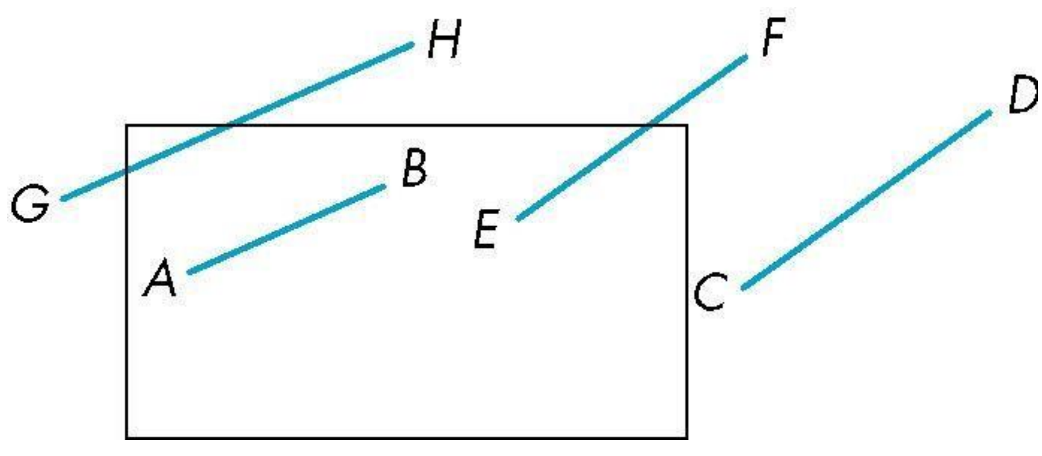
裁剪

- 二维相对于裁剪窗，三维相对于裁剪体
- 对线段和多边形很容易进行，对于曲线、曲面和文本很难进行
  - 首先转化为线段和多边形



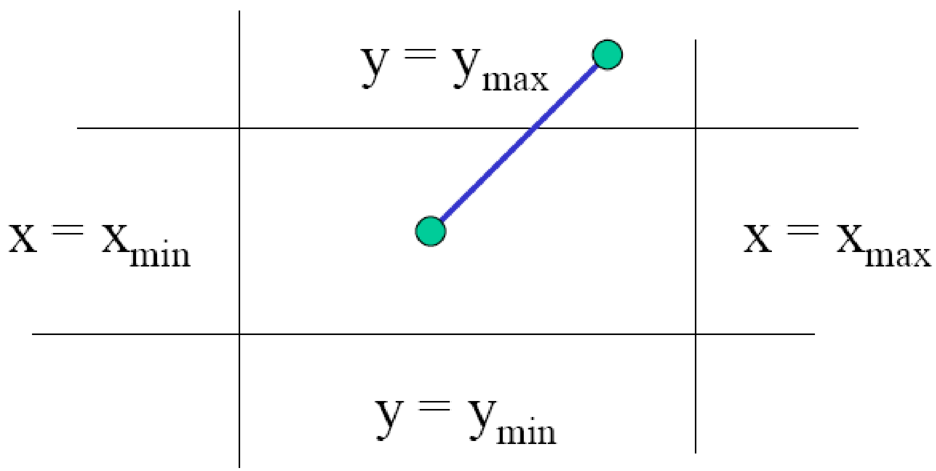
二维线段的裁剪

- 直观方法：计算线段与裁剪窗口各条边界的交点
  - 低效：每次求交需要一次浮点除法



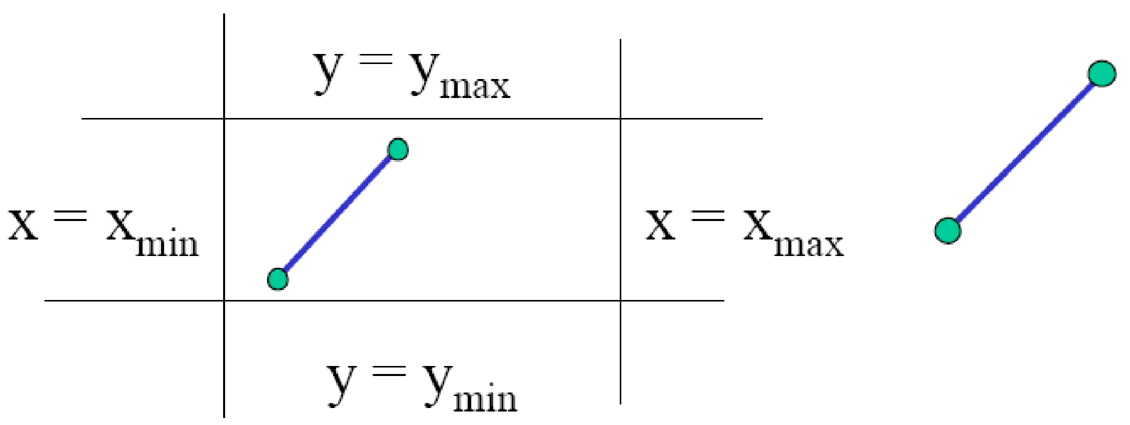
Cohen-Sutherland算法

- 思想：尽可能不经过求交就排除许多情形
- 从确定裁剪窗口边界的四条直线开始



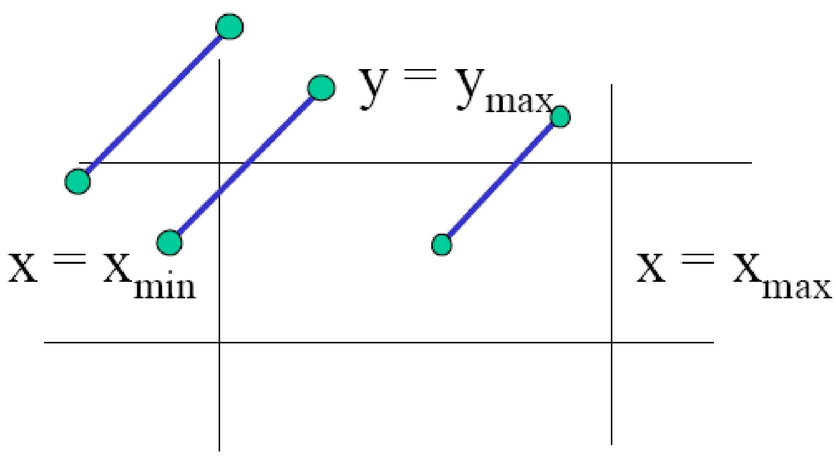
各种情形

- Case 1: 线段的两个端点都在裁剪窗口内
  - 原样绘制直线，即接受
- Case 2: 两个端点都在窗口外，且在同一条直线的外侧
  - 丢弃这条直线，即拒绝



各种情形

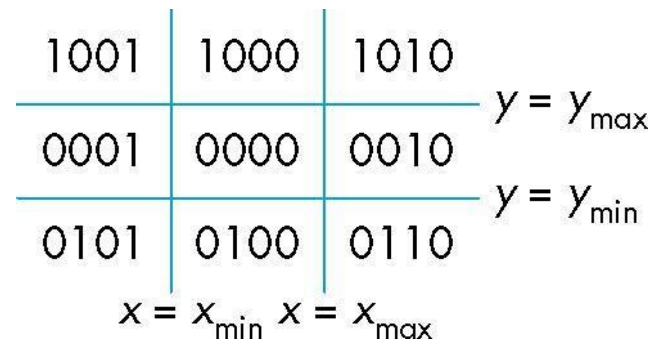
- Case 3: 一个端点在内部，一个端点在外部
  - 必须进行至少一次求交
- Case 4: 都在外部
  - 仍可能部分在内部
  - 必须进行至少一次求交



定义编码

- 对于每个端点，定义一个编码$b_0b_1b_2b_3$
  - 如果$y>y_{max}$, b_0 =1，否则=0
  - 如果$y<y_{min}$, b_1 =1，否则=0
  - 如果$x>x_{max}$, b_2 =1，否则=0
  - 如果$x<x_{min}$, b_3 =1，否则=0

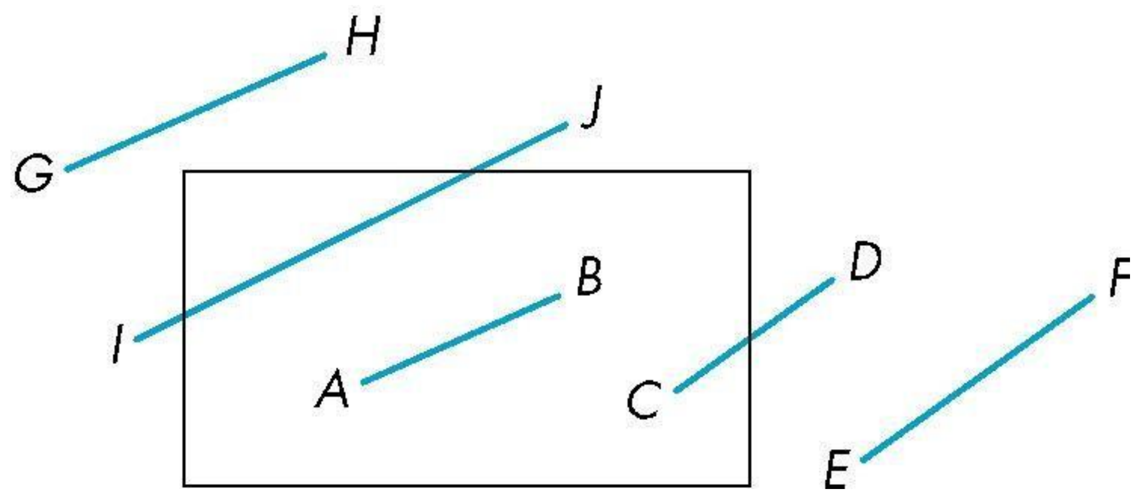
上下右左



- 编码把空间分成九个区域
- 计算编码最多需要四次减法

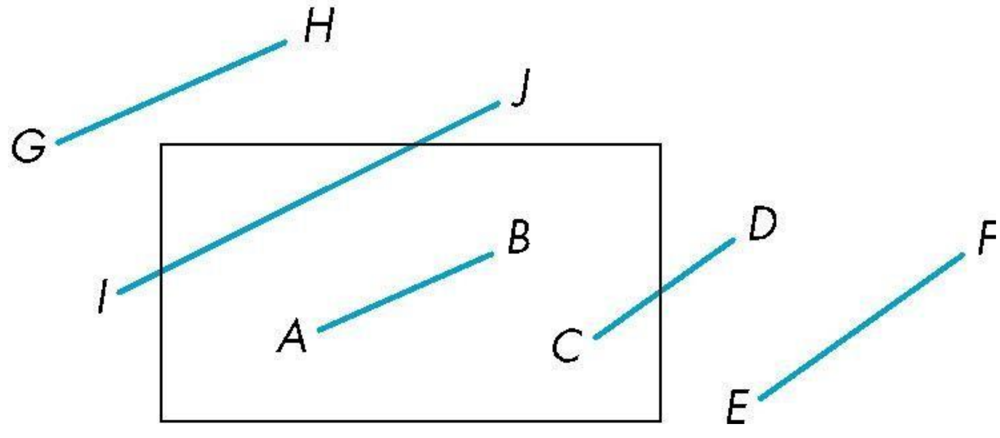
应用编码

- 考虑图中所示的五种情形
- AB: 编码(A) = 编码(B) = 0000
  - AB为可接受线段



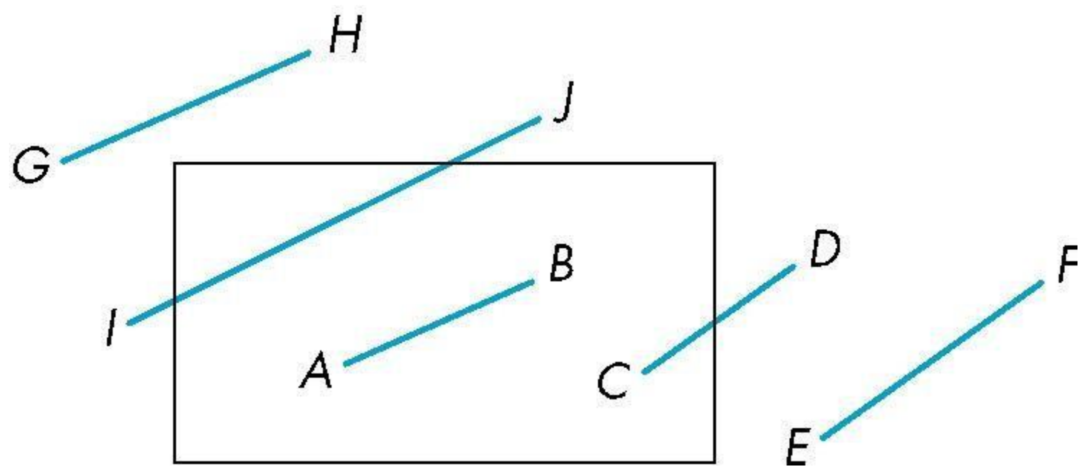
应用编码

- CD: 编码(C)=0000，编码(D)=0010≠0000
  - 计算交点
  - 在编码(D)中的1确定线段与哪条边相交
  - 如果有一条从点A出发的线段，另一端点的编码中有两个1，那么可能需要进行两次求交



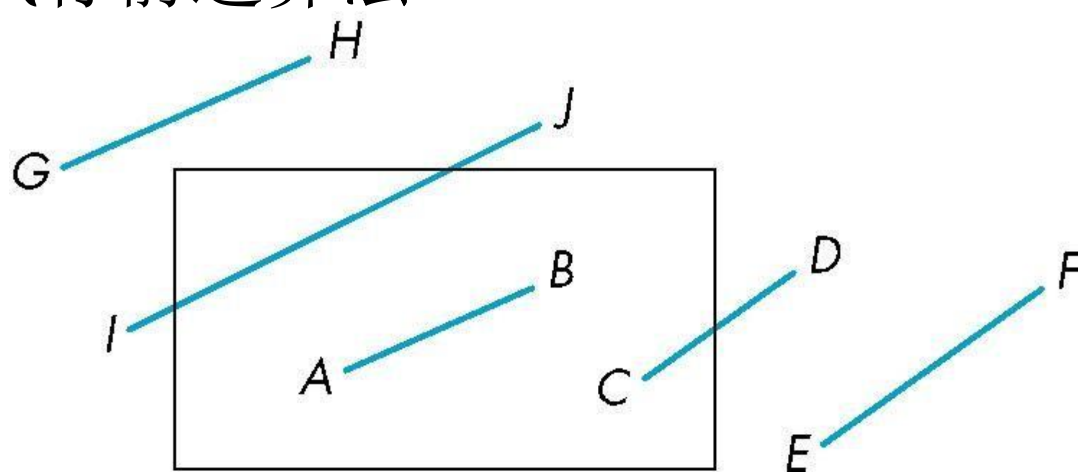
应用编码

- EF:编码(E)与编码(F)的逻辑与=0010≠0000
  - 即两个编码中有某一位同时等于1
  - 线段在裁剪窗口的对应边界的外侧
  - 拒绝



应用编码

- GH与IJ: 相似的编码，不全是零，但逻辑与为0000
  - 通过与窗口一条边求交缩短线段
  - 计算新端点的编码
  - 重新执行前述算法

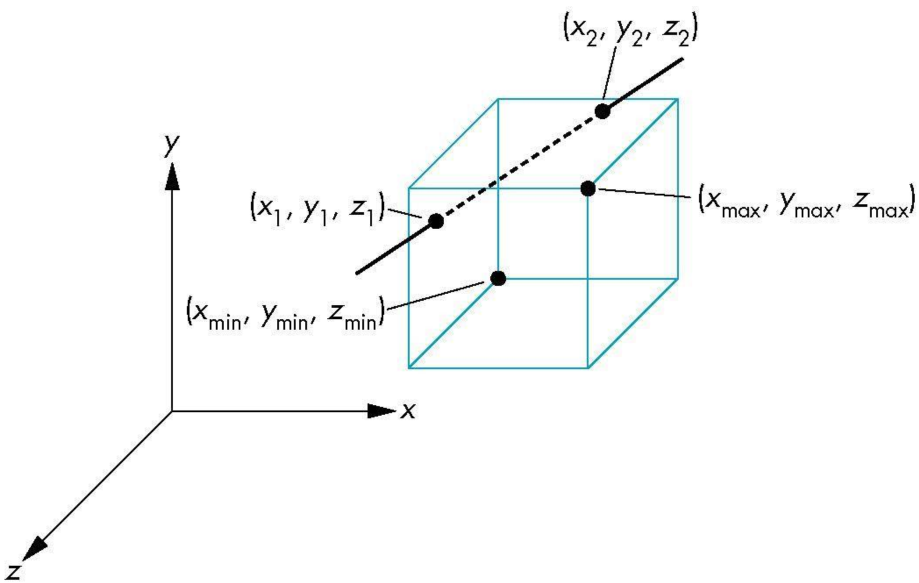


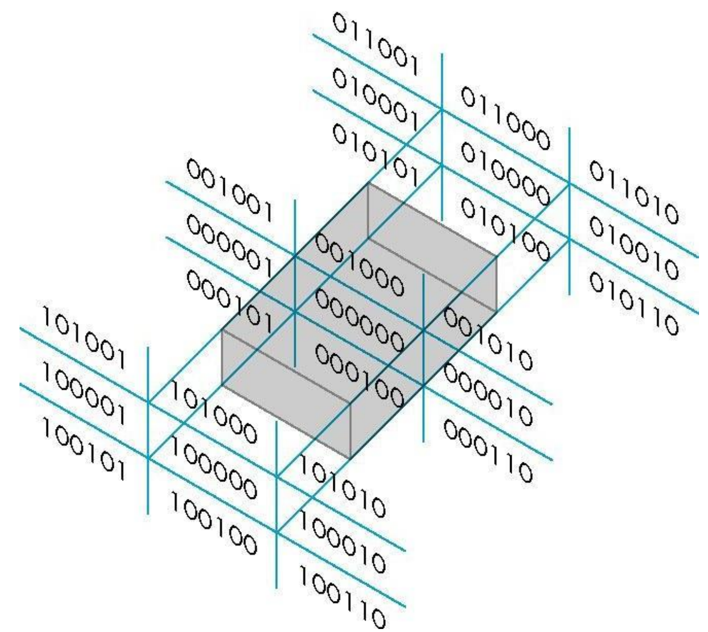
效率

- 在绝大多数应用中，裁剪窗口相对于整个对象数据库而言是比较小的
  - 大多数线段是在窗口的一条边或多条边外面，从而可以基于编码把它们丢弃
- 当线段需要用多步进行缩短时，代码要被重复执行，这时效率不高
- 求交可采用直线的斜截式表示：y=mx+h
  - 斜截式不能表示垂直的线段

三维的Cohen-Sutherland算法

- 利用6位进行编码
- 必要时，相对于平面裁剪线段

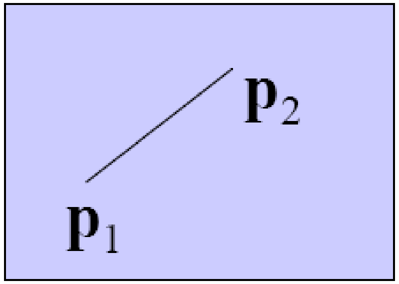




Liang-Barsky裁剪算法

- 考虑线段的参数化表示

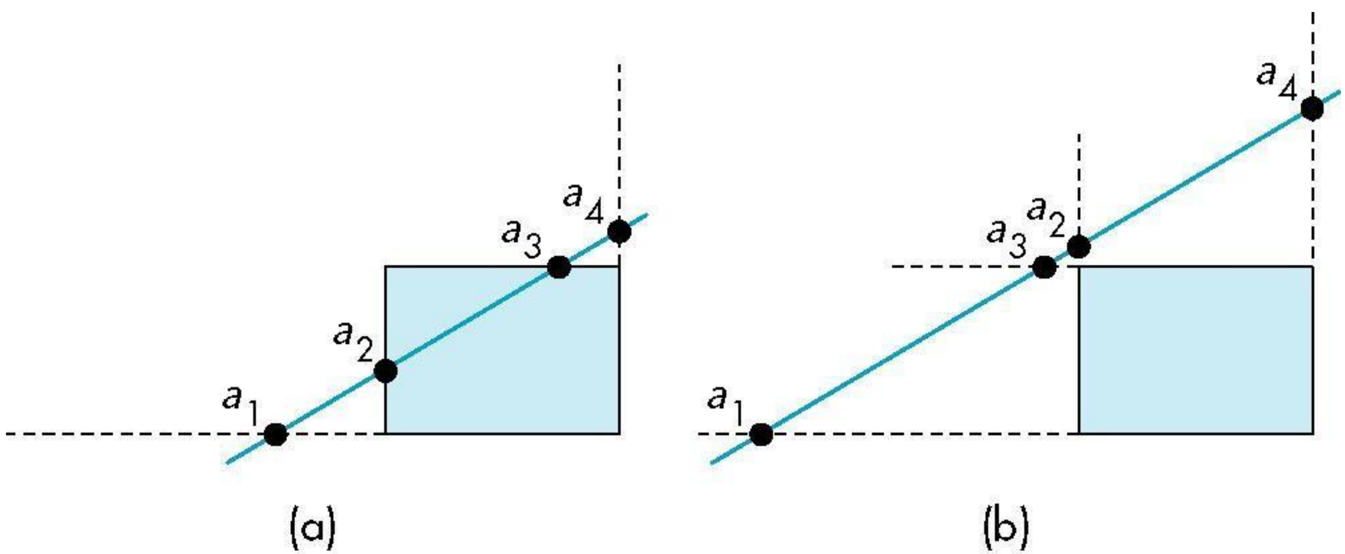
$$\mathbf{p}(\alpha) = (1 - \alpha)\, \mathbf{p}_1 + \alpha\, \mathbf{p}_2,\ 0 \le \alpha \le 1$$



- 在计算出线段所在直线与窗口各边交点对应的α值后，我们可以通过这些参数值的顺序区分出各种情形

其中两种情形

- 情形(a): $1 > \alpha_4 > \alpha_3 > \alpha_2 > \alpha_1 > 0$
  - 交点依次在右、顶、左、底：缩短
- 情形(b): $1 > \alpha_4 > \alpha_2 > \alpha_3 > \alpha_1 > 0$
  - 交点依次在右、左、顶、底：丢弃



效率的考虑

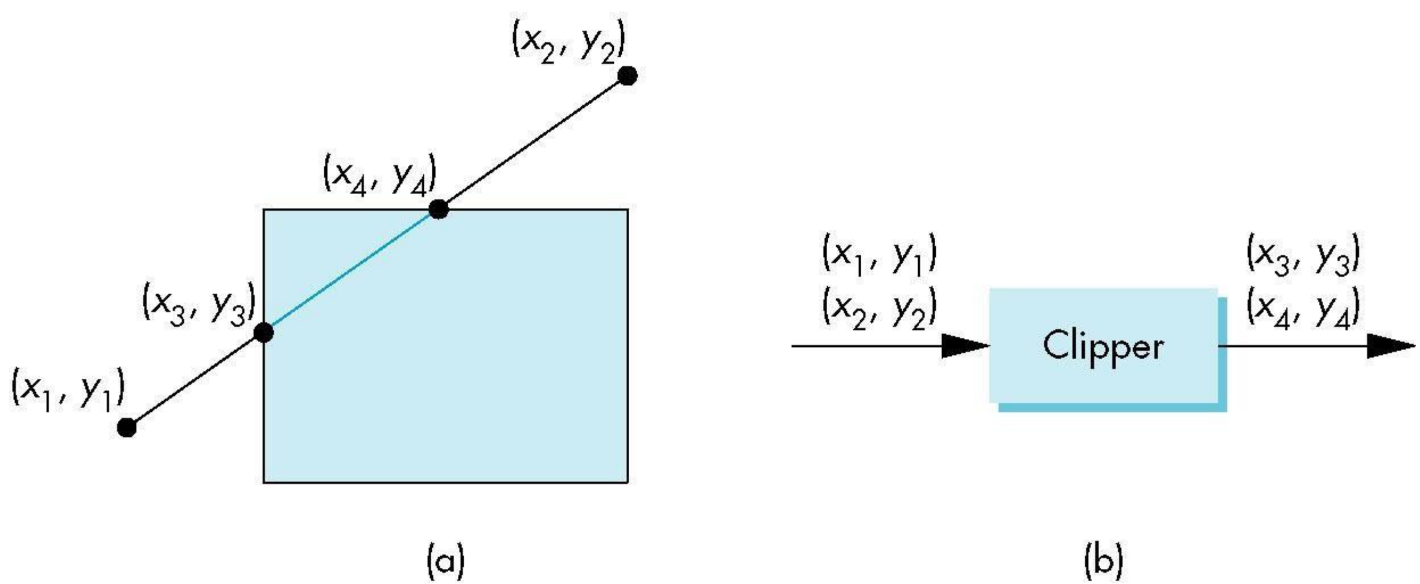
- 顶边交点的表示 $\alpha = (y_{max} - y_1)/(y_2 - y_1)$
  - 需要浮点除法
- 交点方程的重写：
  - $\alpha\ (y_2 - y_1) = \alpha\ \Delta y = (y_{max} - y_1) = \Delta y_{max}$
  - 所需要的测试可以对Δy_{max}和Δy以及其它类似项进行，不需要浮点除法运算
  - 只有当需要对直线进行缩短时才计算交点

优势

- 与Cohen-Sutherland算法一样很简单地接受或拒绝
- 应用α值使得不必要像Cohen-Sutherland算法那样重复应用算法进行多次裁剪
- 也可以推广到三维的情形

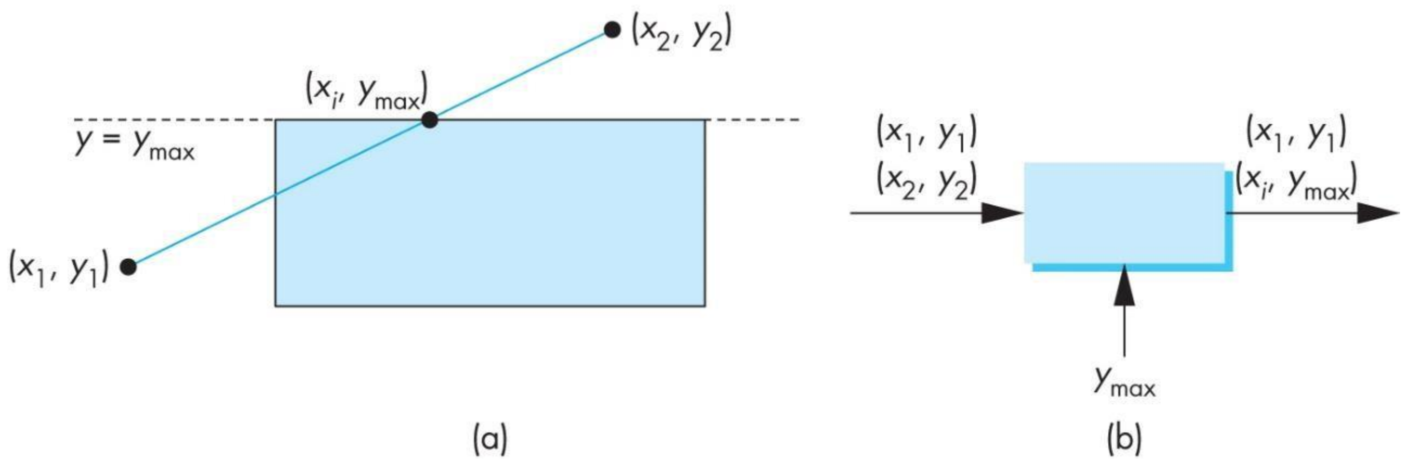
Sutherland-Hodgeman算法

- 裁剪器是一个黑盒
  - 可以认为线段的裁剪就是从两个顶点出发，得到的结果为：没有顶点或者裁剪后线段的顶点



Sutherland-Hodgeman算法

- 使用裁剪窗口的顶边来进行裁剪
  - 输入和输出都是一对顶点坐标，把y_{max}作为裁剪器已知的输入参数



顶裁剪器

– 利用相似三角形，如果存在交点，为

$$x_3 = x_1 + (y_{max} - y_1)(x_2 - x_1)/(y_2 - y_1)$$

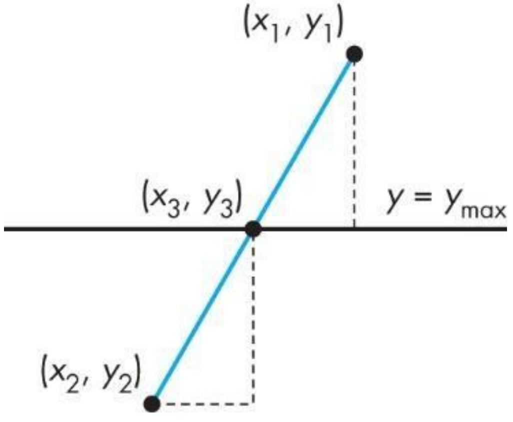
$$y_3 = y_{max}$$

– 裁剪器返回下面三对顶点中的某一对：

$\{(x_1,y_1), (x_2,y_2)\}$

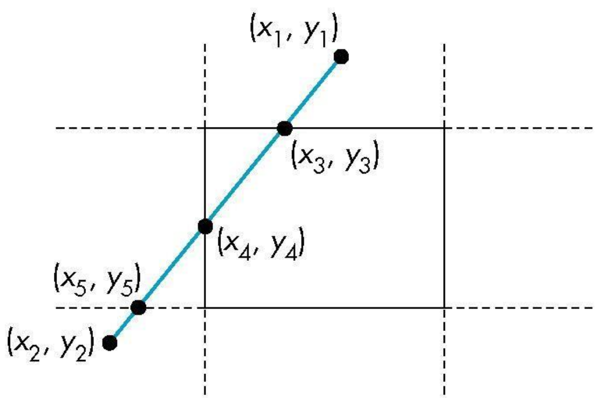
$\{(x_1,y_1), (x_i, y_{max})\}$

$\{(x_i, y_{max}), (x_2,y_2)\}$

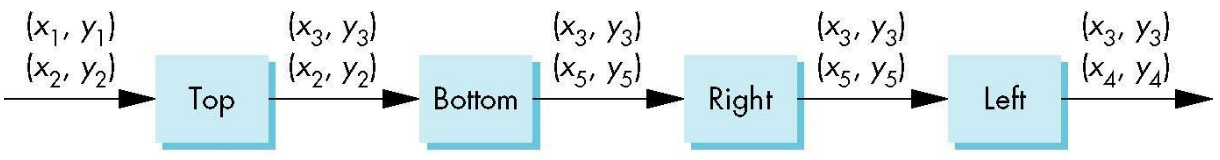


线段裁剪的流水线体系

- 用窗口一边进行裁剪时，与其它边无关
  - 在流水线中要用到四个独立的裁剪器



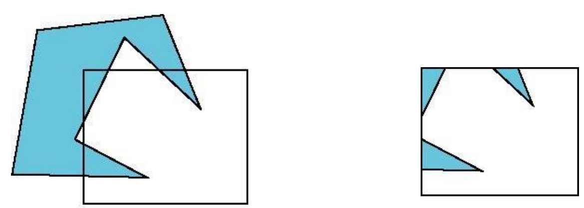
(a)



(b)

多边形的裁剪

- 并不像线段裁剪那样简单
  - 裁剪一条线段最多得到一条线段
  - 裁剪一个多边形可以得到多个多边形

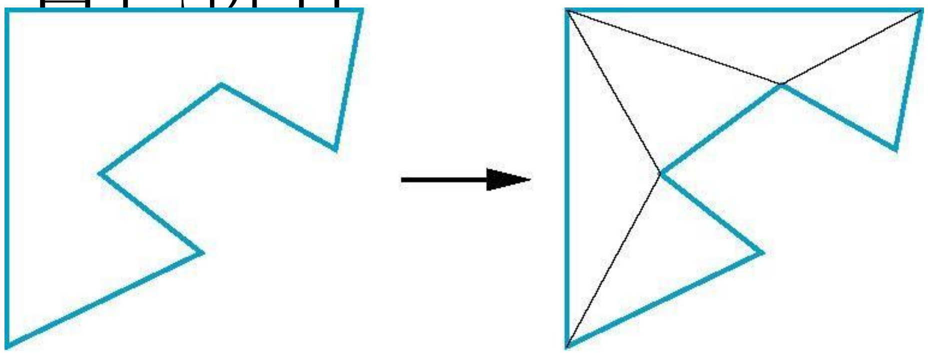


(a) (b)

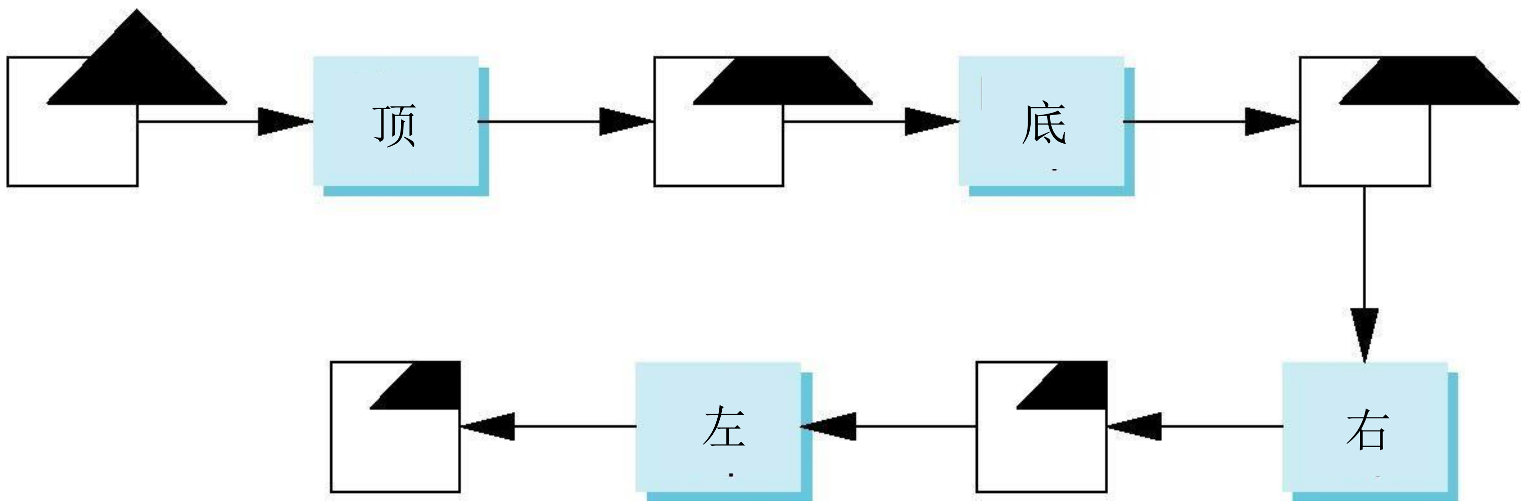
- 然而裁剪凸多边形最多得到一个凸多边形

剖分与凸性

- 一种方法就是把非凸(凹)多边形用一组三角 形代替，这个过程称为剖分(tessellation)
- 这同样也使得填充变得简单
- 在GLU库中有剖分代码，但最好的方法就是由用户自己进行



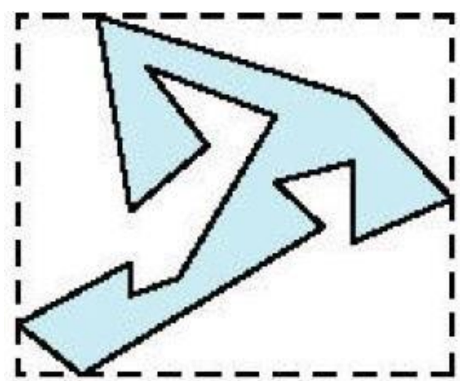
多边形裁剪的流水线体系



- 三维：增加前与后裁剪器
- SGI Geometry Engine中采用了这种策略
- 在等待时间方面有很小的增长

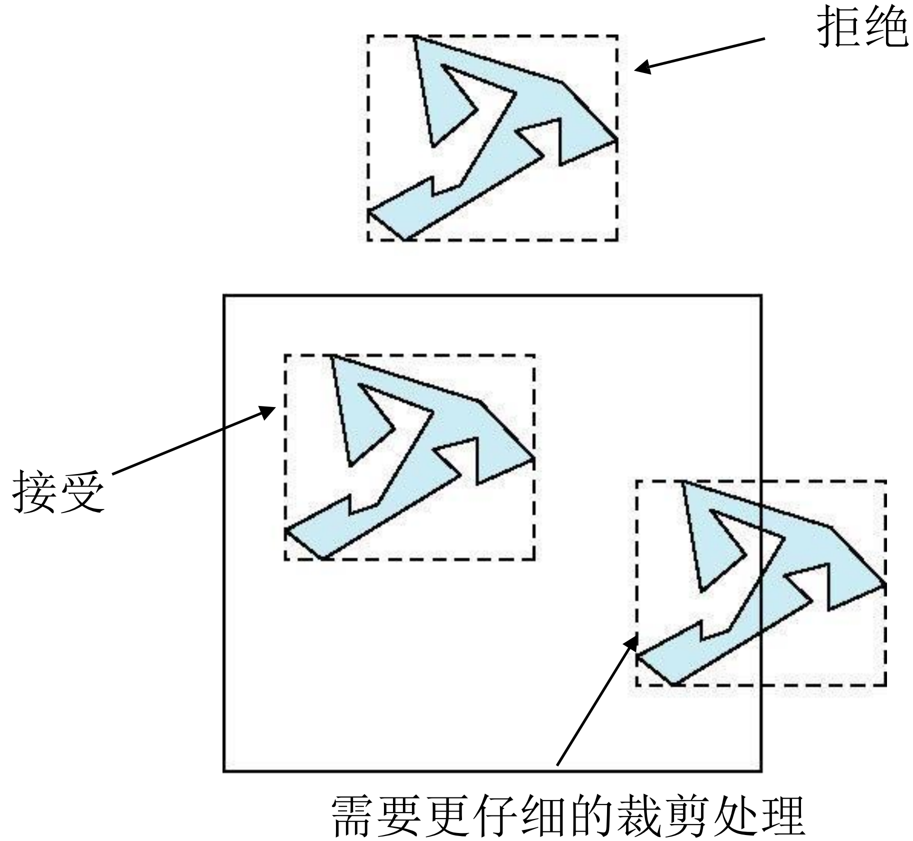
包围盒

- 不直接对复杂多边形进行裁剪，而是使用一个轴对齐包围盒(axis-aligned bounding box, AABB)
  - 与裁剪窗口的边(或坐标轴)对齐且包含该多边形的最小矩形
  - 易计算：求出多边形顶点的最大/小x和y坐标



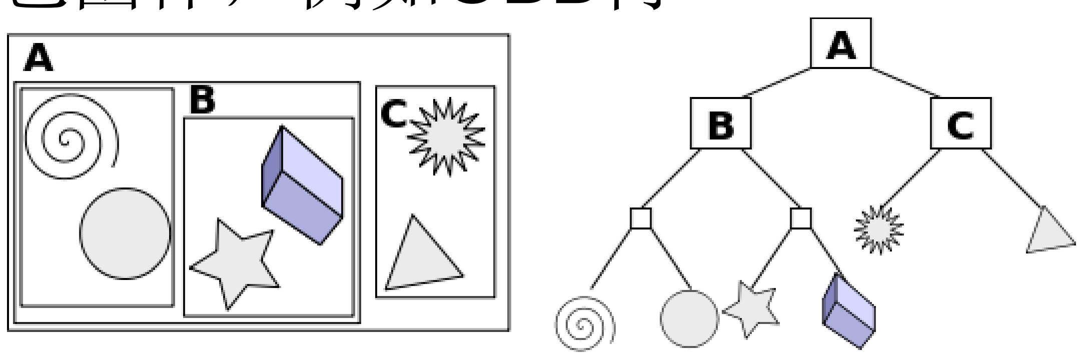
应用包围盒

- 直接基于包围盒确定多边形的接受与拒绝



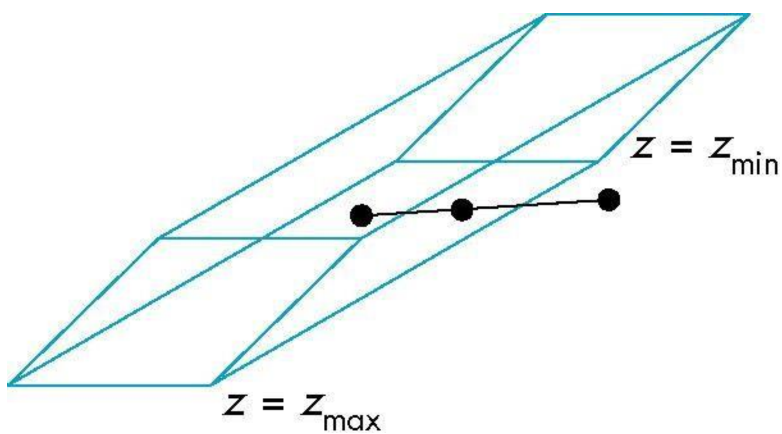
应用包围体

- 轴对齐包围盒在二维和三维情形下都适用
- 其他形式的包围体
  - 包围球(bounding sphere)
  - 方向包围盒(oriented bounding box, OBB)
- 除了裁剪，包围体还常应用在碰撞检测(collision detection)和光线跟踪中
  - 层次包围体，例如OBB树



裁剪与规范化

- 在三维空间中一般的裁剪需要计算线段与任意平面的交点
- 例：斜投影视图



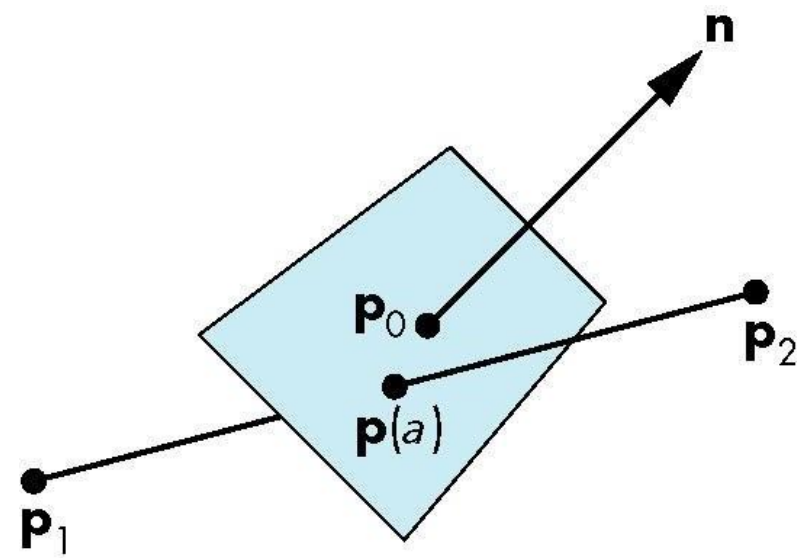
平面与直线的交点

直线：$\mathbf{p}(\alpha) = (1 - \alpha)\,\mathbf{p}_1 + \alpha\,\mathbf{p}_2$

平面：$\mathbf{n}\cdot(\,\mathbf{p}(\alpha) - \mathbf{p}_0) = 0$

交点的参数值：

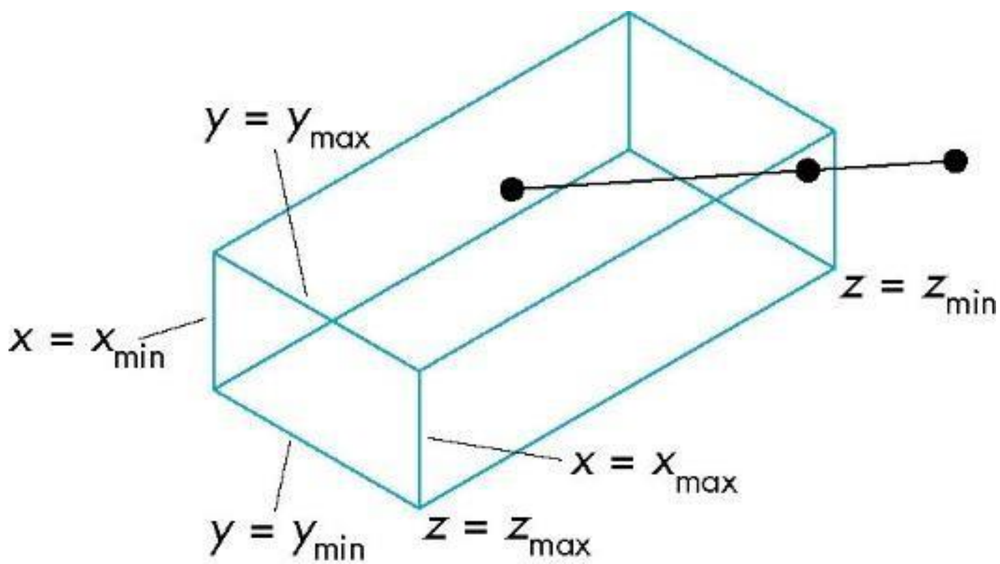
$$\alpha = \frac{\mathbf{n}\cdot(\mathbf{p}_0 - \mathbf{p}_1)}{\mathbf{n}\cdot(\mathbf{p}_2 - \mathbf{p}_1)}$$



求一个交点需要6次乘法运算和1次除法运算

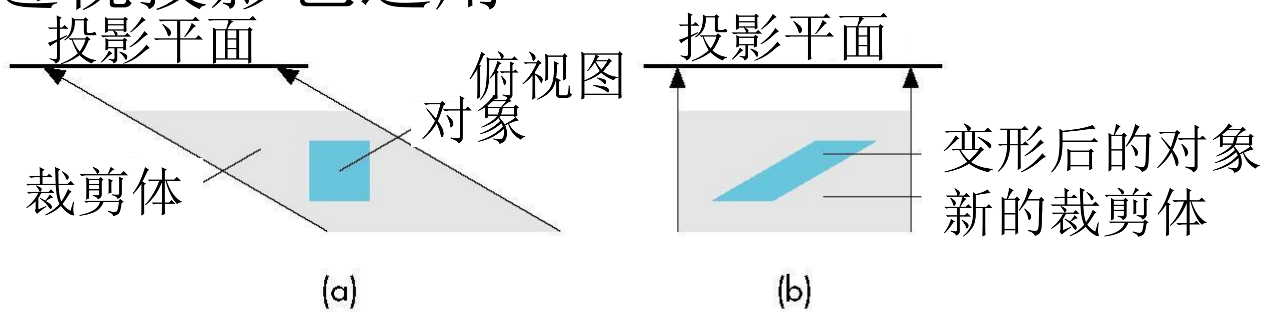
正交投影的裁剪

- 对于正交投影，视景体是一个长方体
  - 求交运算可以简化为单次浮点除法运算



规范化

- 规范化过程是视图生成的一部分(在裁剪前 进行)。在规范化后，相对于正平行六面体
  （长方体）进行裁剪
  - 这时典型的求交计算只需要一次浮点除法
  - 对透视投影也适用



规范化前 规范化后

流水线中的裁剪

- 定义顶点的对象坐标(4D)经由模型-视图矩阵变换为视点坐标(4D)
- 视点坐标由投影矩阵变换(规范化)为裁剪坐标(4D)
  - 裁剪空间(clip space)：

$$-w \le x \le w$$

$$-w \le y \le w$$

$$-w \le z \le w$$

- 在裁剪标架中进行裁剪处理后，执行透视除法把顶点的裁剪坐标变换为规范化设备坐标(3D)
  - 在透视除法之前完成裁剪处理，可避免对裁剪体之外的图元顶点进行透视除法运算