2018-2019年度第二学期 00106501

计算机图形学

童 伟 华　管理科研楼1205室

E-mail: tongwh@ustc.edu.cn

中国科学技术大学　数学科学学院

http://math.ustc.edu.cn/

第七章 从顶点到片元

第一节　框架与剪裁

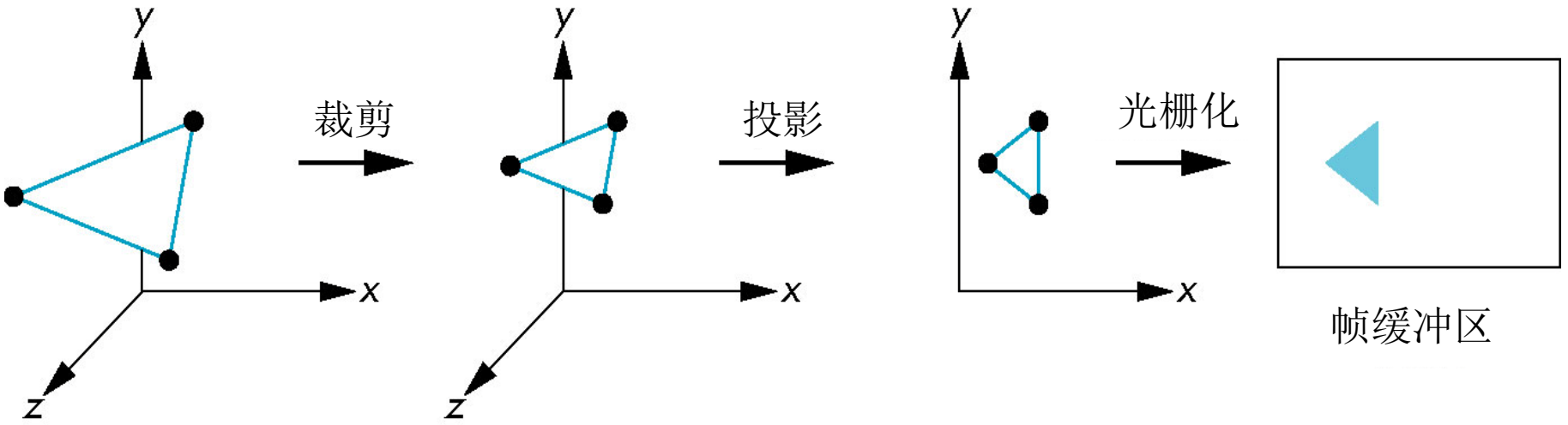
基本算法

- 考虑输出由不透明对象构成的场景的如下两种方法
  - 对于每个像素，确定投影到这个像素的离观察者最近的那个对象，从而基于该对象计算像素的亮度
    - 光线跟踪框架
  - 对于每个对象，确定它所覆盖的像素，并用对象的状态确定像素的亮度
    - 流水线过程
    - 必须跟踪深度
- 隐藏面消除（Hidden-surface removal）或可见面确定（Visible-surface determination）

算法类型

- 前述两种过程分别称为面向图像（image-oriented）的过程和面向对象（object-oriented）的过程
  - 也分别称为先排序（sort-first）与后排序（sort-last）过程
    - 基于隐藏面消除发生的地方

基于对象的过程

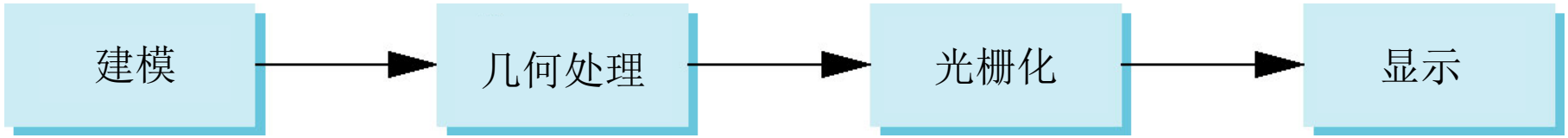


缺点与优点

- **缺点**
  - 需要占用较多的内存
  - 每个对象都要单独处理
  - 无法处理大部分的全局计算
    - 隐藏面消除例外
- **优点**
  - 对每个图元执行相同的操作
    - 可并行化
    - 便于硬件实现
  - 算法相对简单

基本任务

- 裁剪
- 光栅化或者称为扫描转化
- 隐藏面消除
- 反走样



建模

- 在后续课程中将介绍图形学中其它的建模技术
- 在OpenGL中，模型通常用点、线段、多边形等基本图元来表示，可描述为：顶点数据＋装配信息
- 建模程序可以认为是一个黑盒子

几何处理

- 即确定哪些几何对象要被显示，以及显示出来的颜色或亮度是多少
- 相关联的过程为
  - 规范化
  - 裁剪
  - 明暗处理
  - 隐藏面消除
- 基于浮点数进行运算，处理的是顶点

规范化

- 把几何体从用户坐标转换为照相机坐标或者屏幕坐标
- 通常创建一个标准的视景体
  - 把所有的投影转化为等价的正交投影

裁剪

- 几何对象经过一系列变换后可能会变形或者改变了表示
  - 只有在视景体内的对象被显示出来
  - 不能把所有的对象都光栅化，希望硬件处理完全在视景体的对象

明暗处理

- 也需要几何信息
  - 法向
  - 其它特殊方法

光栅化

- 在进行了扫描后，那么得到的就是二维对象
  - 虽然可能已用屏幕坐标表示了对象，但是所拥有的只是用顶点给出的表示
- 基于顶点给出表示对象的一组像素就称为光栅化（rasterization）或者扫描转化（scan conversion）

隐藏面消除

- hidden-surface removal，也称为可见面判定（visible-surface determination）
  - 基于三维的位置关系

显示

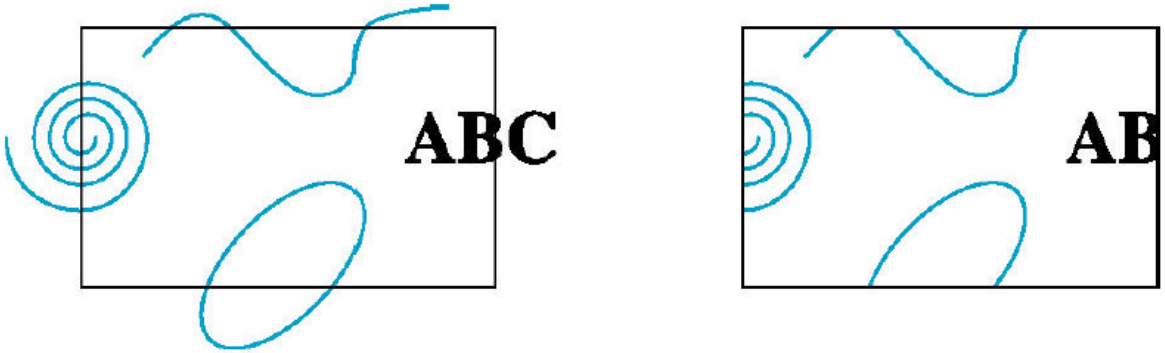
- 在大多数显示设备上，从帧缓冲区中获取信息，并自动显示出来
  - 应用程序一般并不关心这一部分
- 但有许多值得关注的问题
  - 走样

裁剪的目的

- 在几何流水线体系的最后，顶点被集成为基本几何形状
- 需要把在视景体外面的形状删除
  - 算法要基于用一列顶点表示的几何形状
- 需要找到被每个几何形状影响的像素
  - 片段生成
  - 光栅化或扫描转化

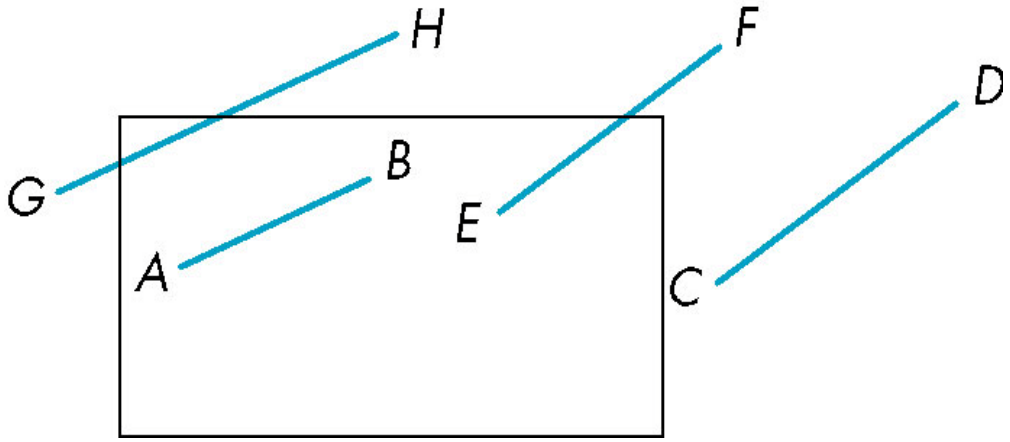
裁剪

- 二维裁剪
- 三维裁剪
- 对线段和多边形很容易进行
- 对于曲线和文本很难进行
  - 首先转化为线段和多边形



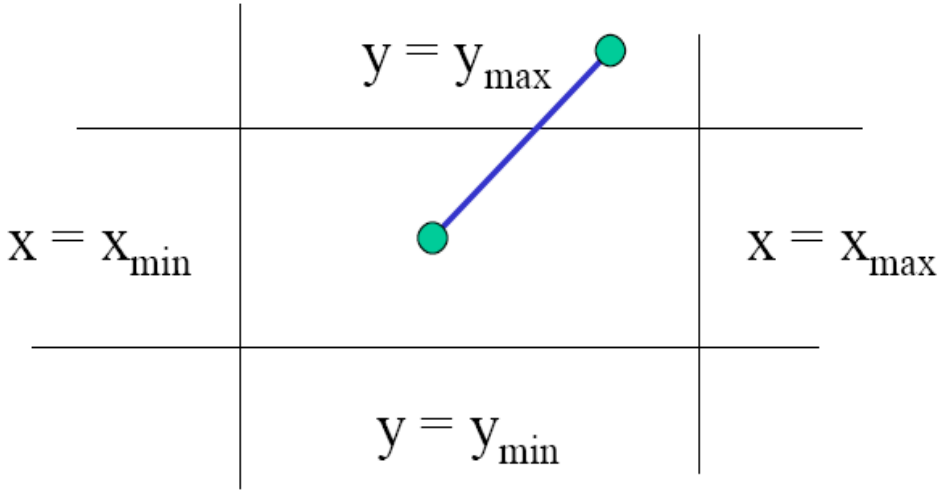
二维线段的裁剪

- 直观方法
  - 计算线段与裁剪窗口边界的交点
    - 低效：每次求交需要一次除法



Cohen-Sutherland算法

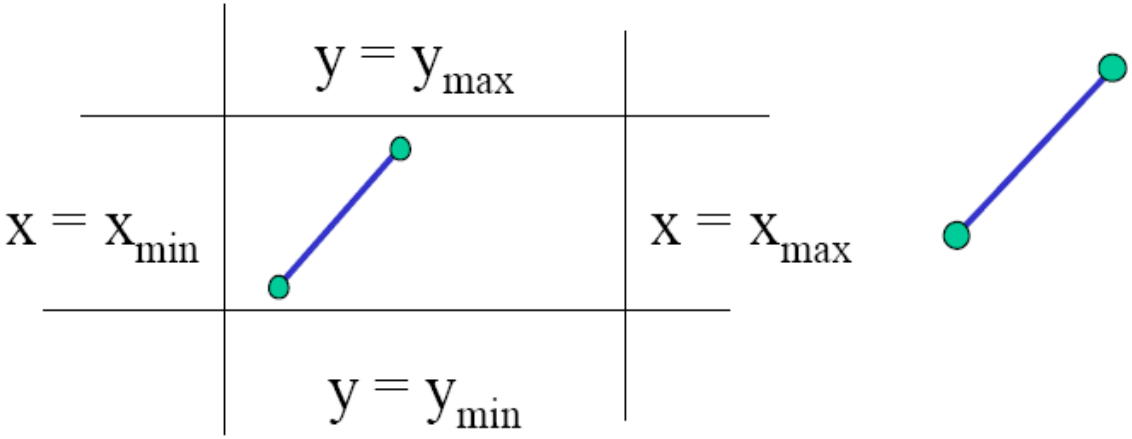
- 想法：尽可能不经过求交就消除许多情形
- 从确定裁剪窗口边界的四条直线开始



各种情形

- Case 1: 线段的两个端点都在四条直线内
  - 原样绘制直线

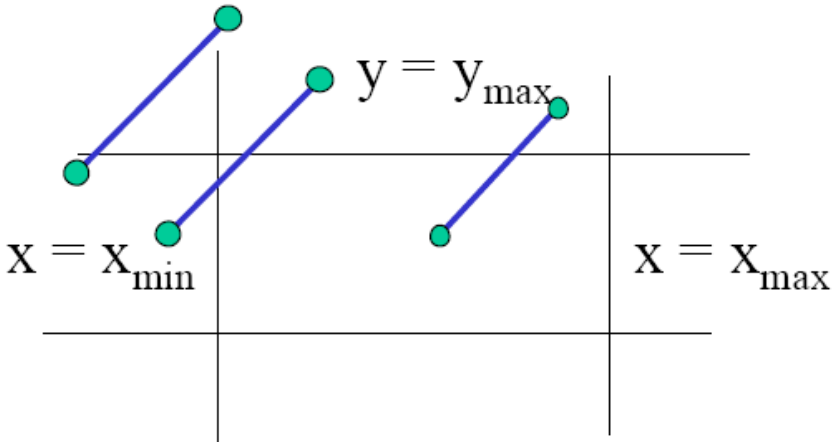
- Case 2: 两个端点都在直线外，而且在一条直线的同侧
  - 抛弃这条直线



各种情形

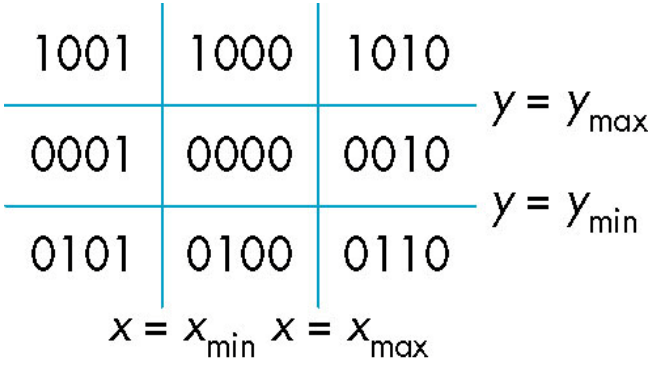
- Case 3: 一个端点在内部，一个端点在外部
  - 必须进行至少一次求交

- Case 4: 都在外面
  - 仍可能部分在内部
  - 必须进行至少一次求交



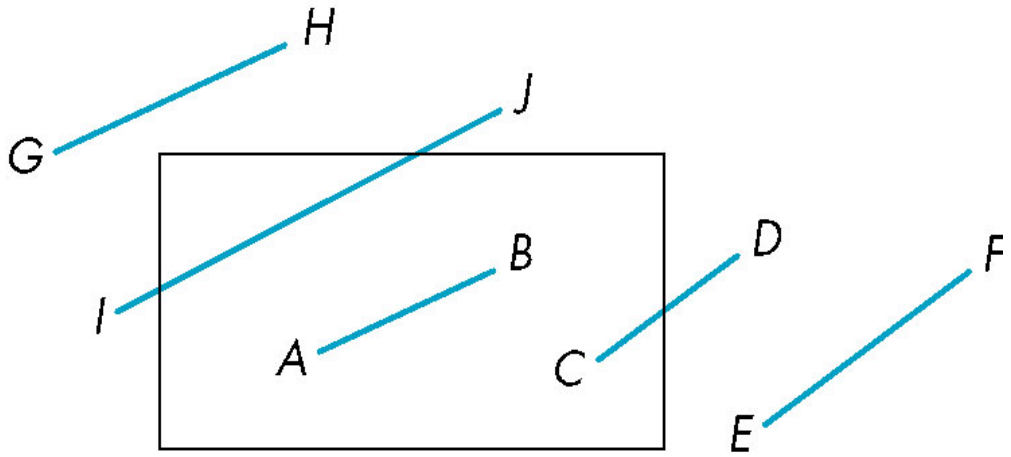
定义编码

- 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$=0$
- 编码把空间分成九个区域
- 计算编码最多需要四次减法



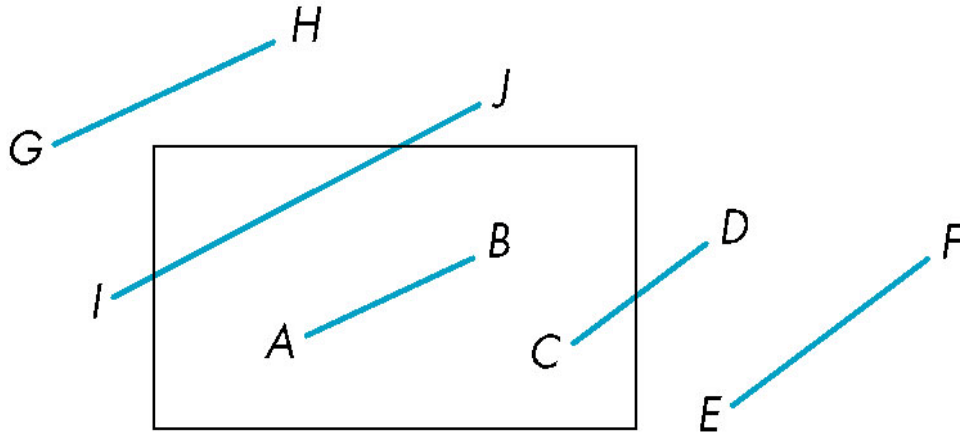
应用编码

- 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



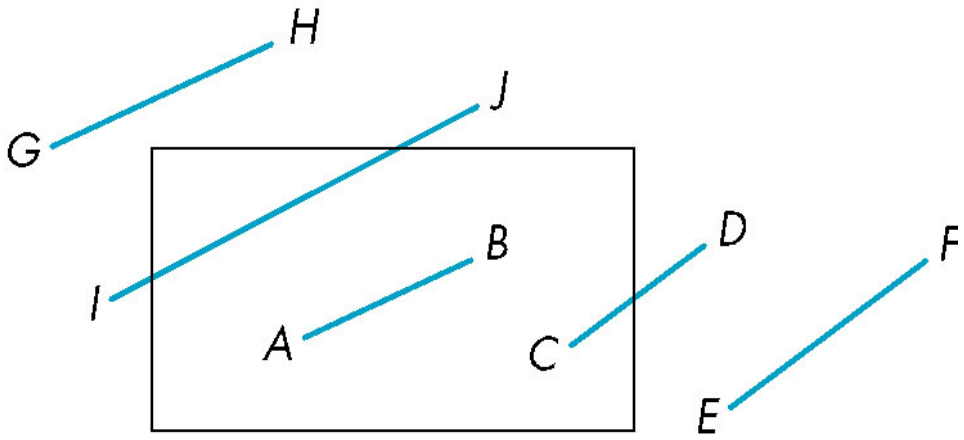
应用编码

- 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



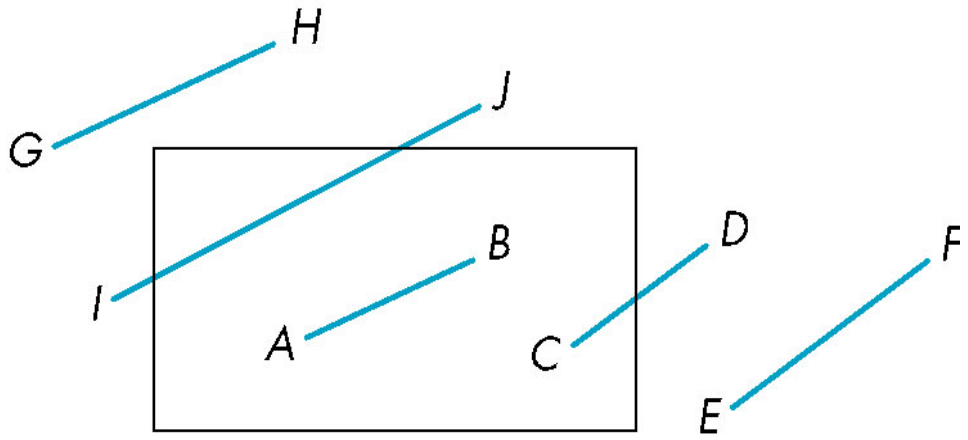
应用编码

- EF: 编码(E) 与编码(F) 的逻辑和 = 0010 ≠ 0
  - 即两个编码中有某一位同时等于1
  - 线段在相应边的外侧
  - 应当抛弃



应用编码

- GH与IJ: 编码相同，也不全是零，但逻辑和为0
  - 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

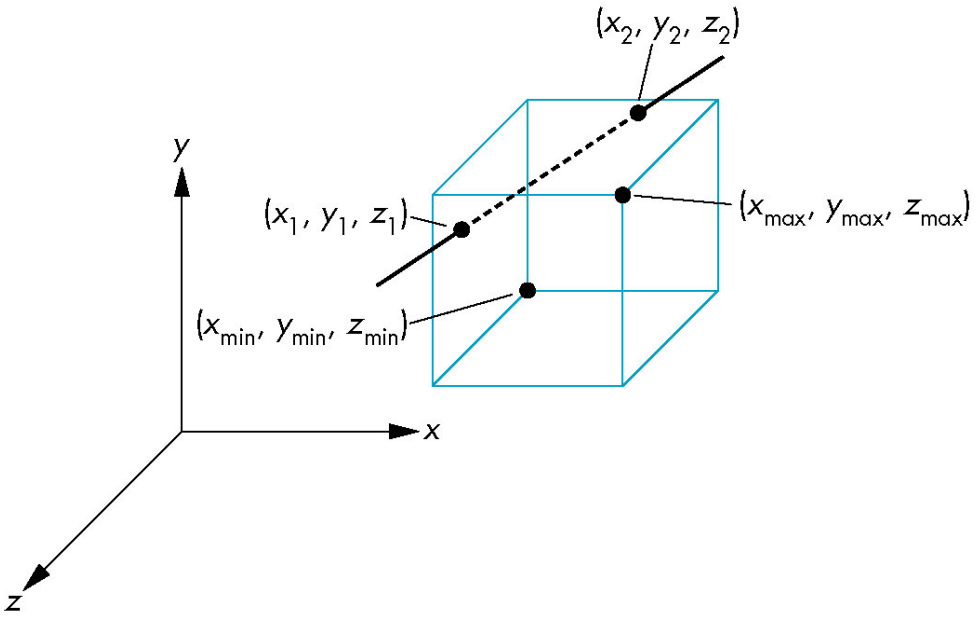


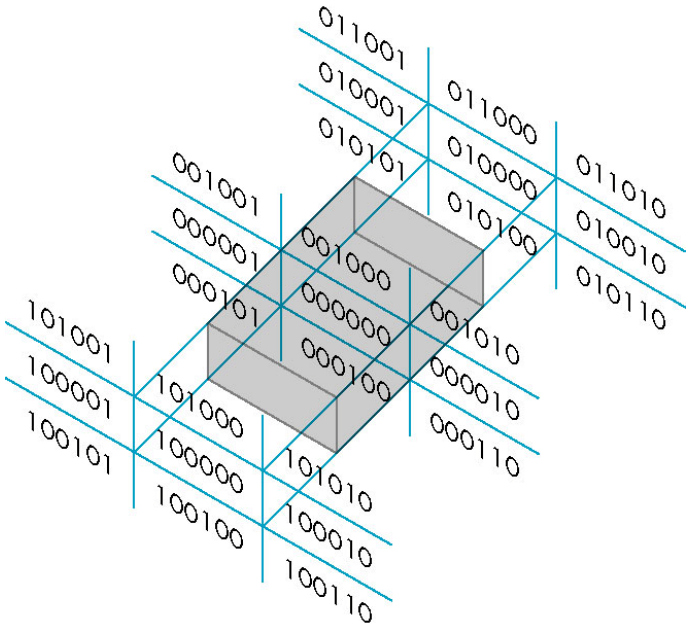
效率

- 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抛弃
- 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

三维空间中的Cohen-Sutherland算法

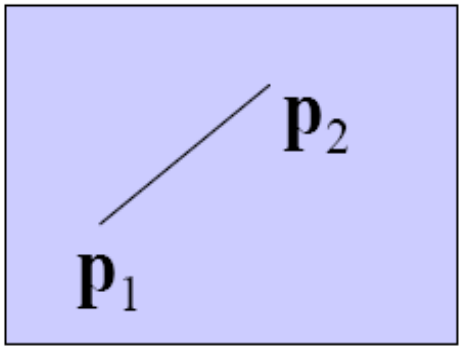
- 利用6位进行编码
- 必要时，相对于平面裁剪线段





Liang-Barsky裁剪算法

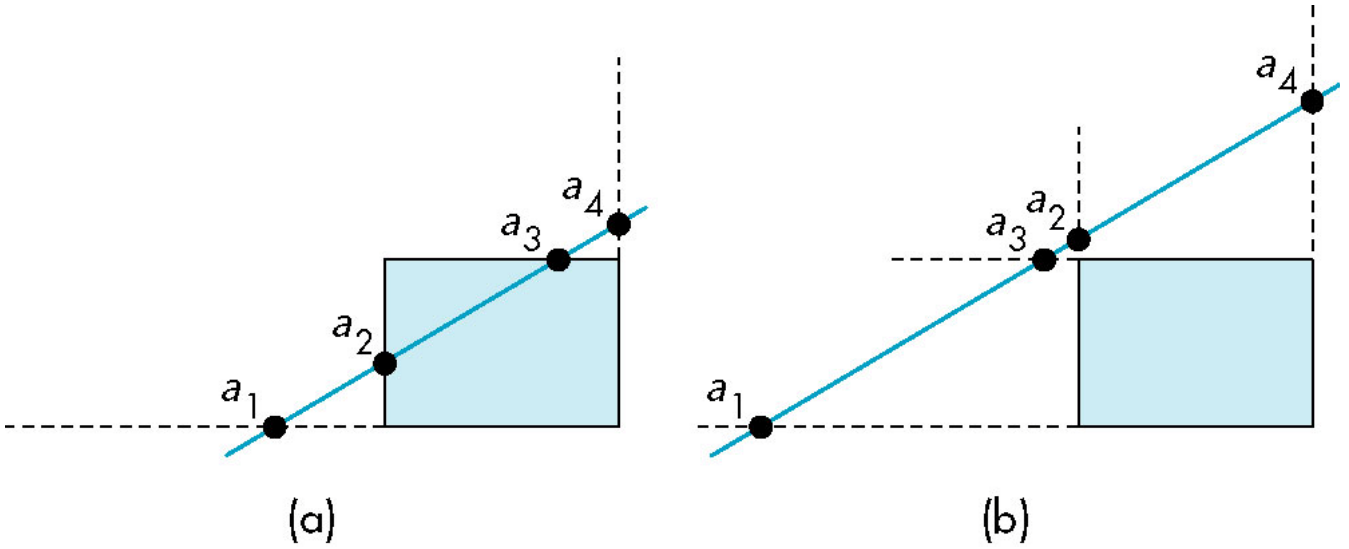
- 考虑线段的参数化表示
  - $p(\alpha) = (1-\alpha)p_1 + \alpha p_2, 0 \le \alpha \le 1$



- 在计算出来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

各种情形

- 情形(a): $\alpha_1 < \alpha_2 < \alpha_3 < \alpha_4$
  - 交点依次在右、顶、左、底边：缩短
- 情形(b): $\alpha_1 < \alpha_3 < \alpha_2 < \alpha_4$
  - 交点依次在右、左、顶、底：抛弃



效率的提高

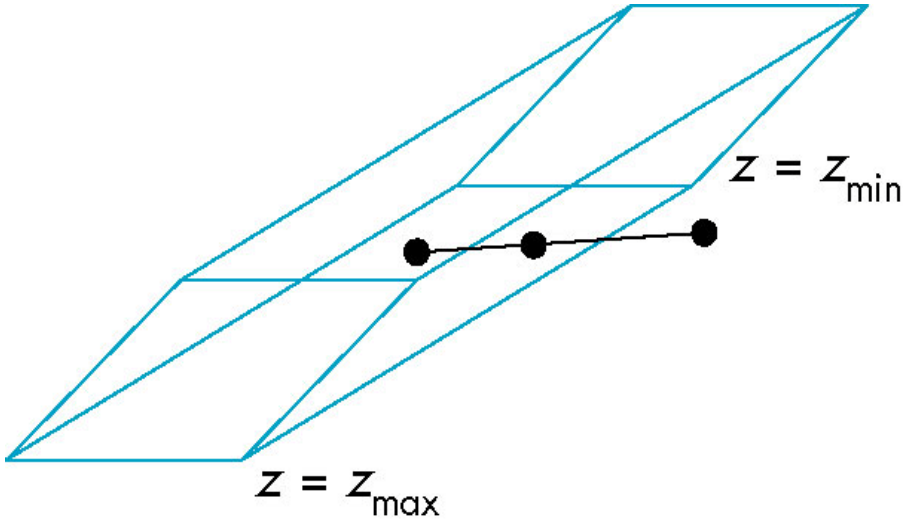
- 交点的表示 $\alpha = (y_{max} - y_1)/(y_2 - y_1)$
  - 需要浮点除法
  - 尽可能地避免交点计算
- 交点方程的重写：$\alpha (y_2 - y_1) = (y_{max} - y_1)$
  - 所需要的测试可以对Δy_{max}和Δy以及其它类似项进行
  - 只有当需要对直线进行缩短时才需要把交点计算出来

优势

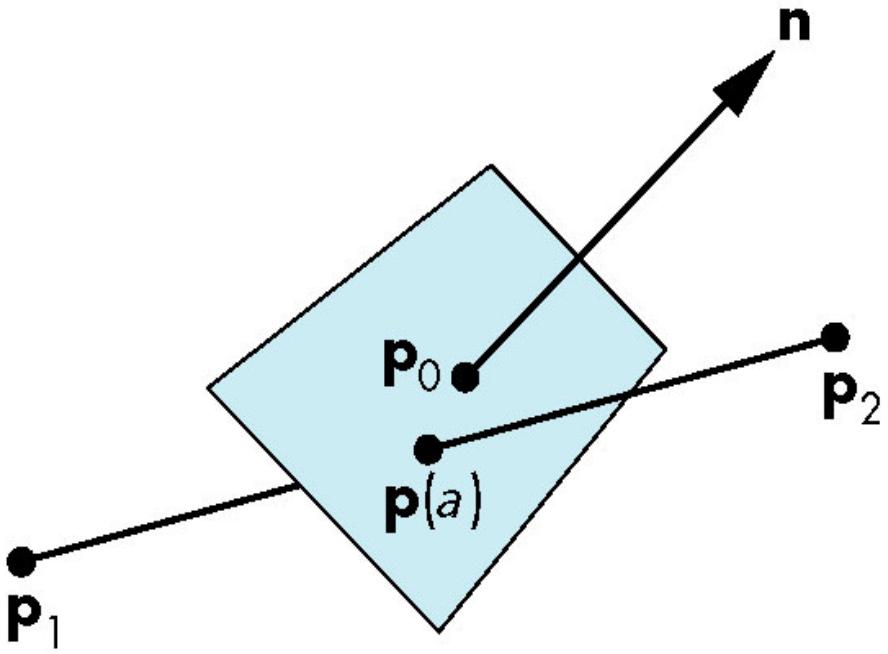
- 与Cohen-Sutherland算法一样很简单地接受或抛弃直线
- 应用α值使得不必要像Cohen-Sutherland算法那样重复应用代码
- 可以推广到三维的情形

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：倾斜投影



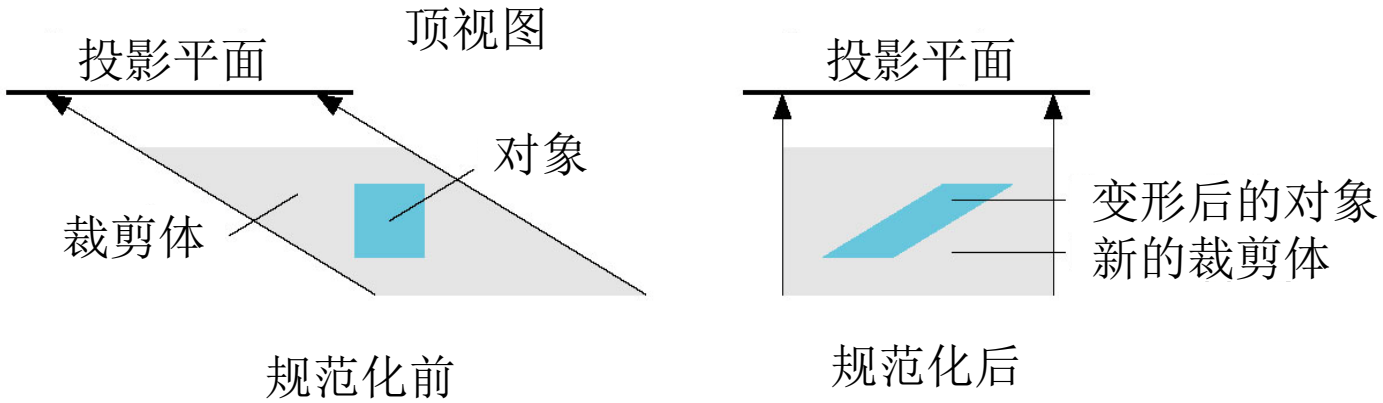
平面与直线的交点



$$\alpha = \frac{n \cdot (p_0 - p_1)}{n \cdot (p_2 - p_1)}$$

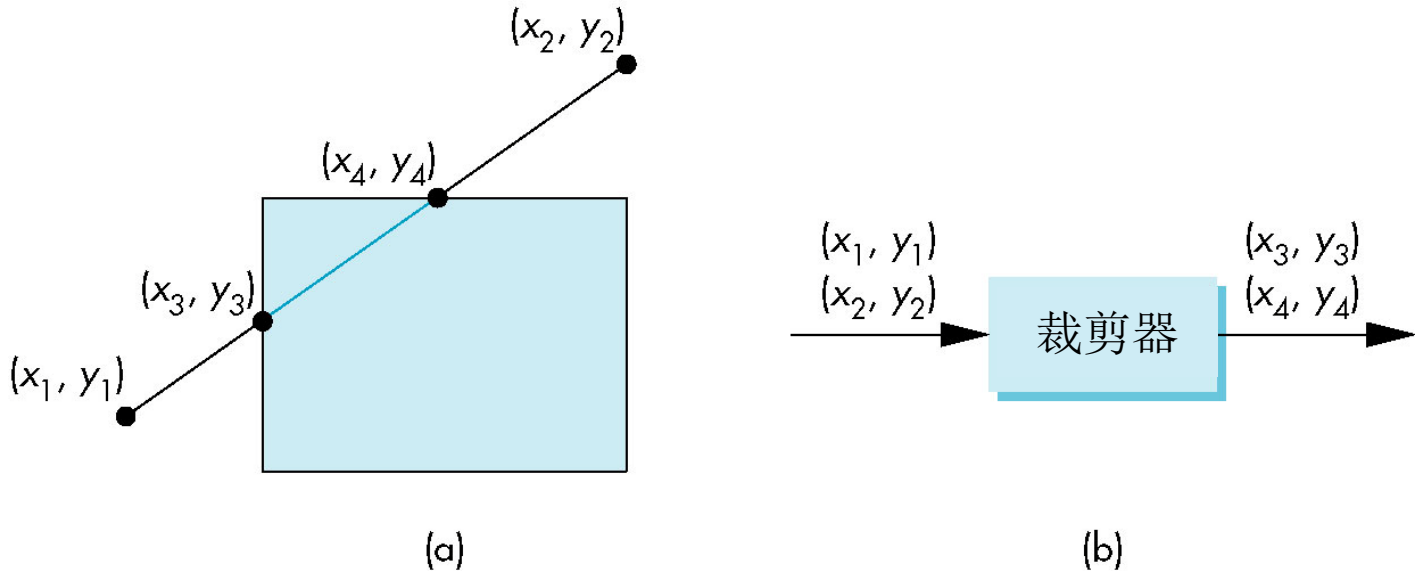
规范形式

- 规范化过程是视图生成的一部分（在裁剪前进行）。但在规范化后，相对于长方体进行裁剪
- 这时典型的求交计算只需要一次浮点减法，例如：x>xmax?



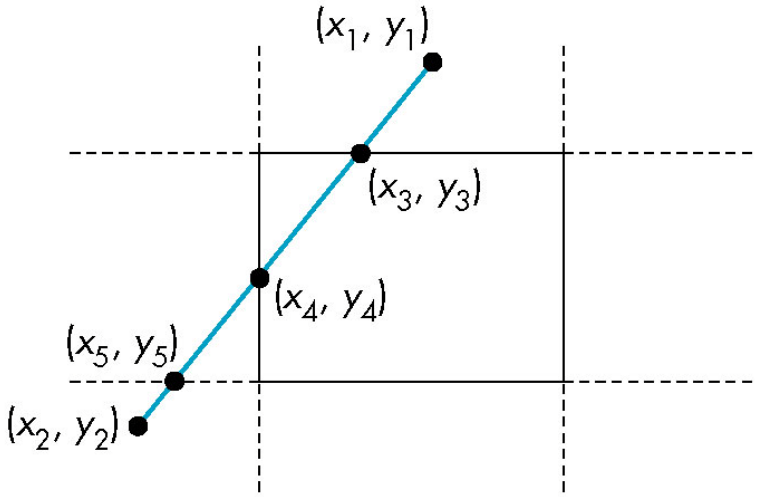
裁剪是一个黑盒子

- 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

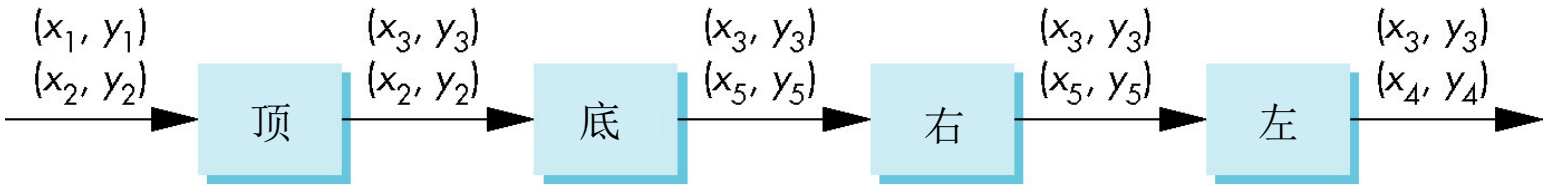


线段裁剪的流水线体系

- 对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



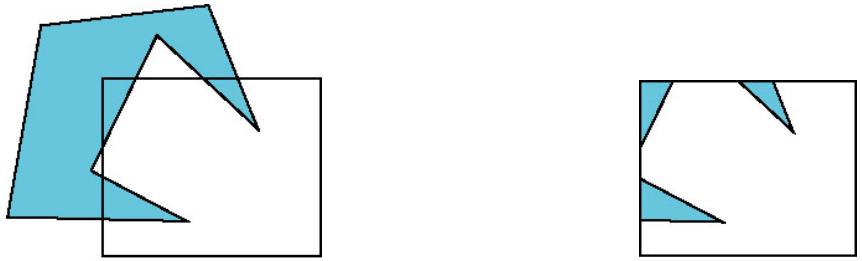
(a)



(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

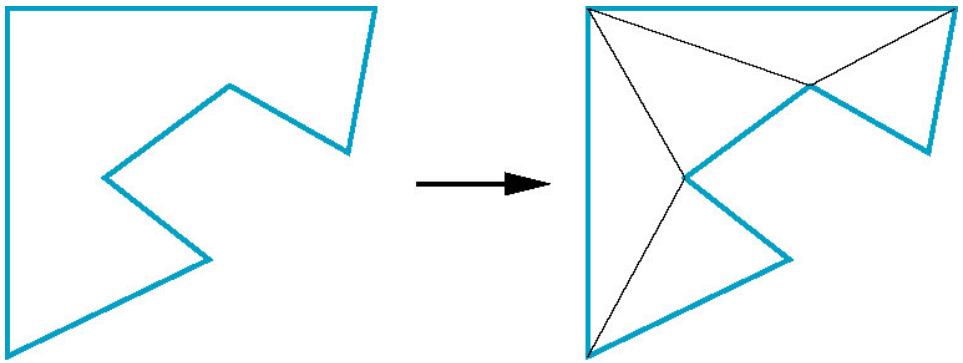


(a)　　(b)

- 然而，裁剪凸多边形最多得到一个多边形

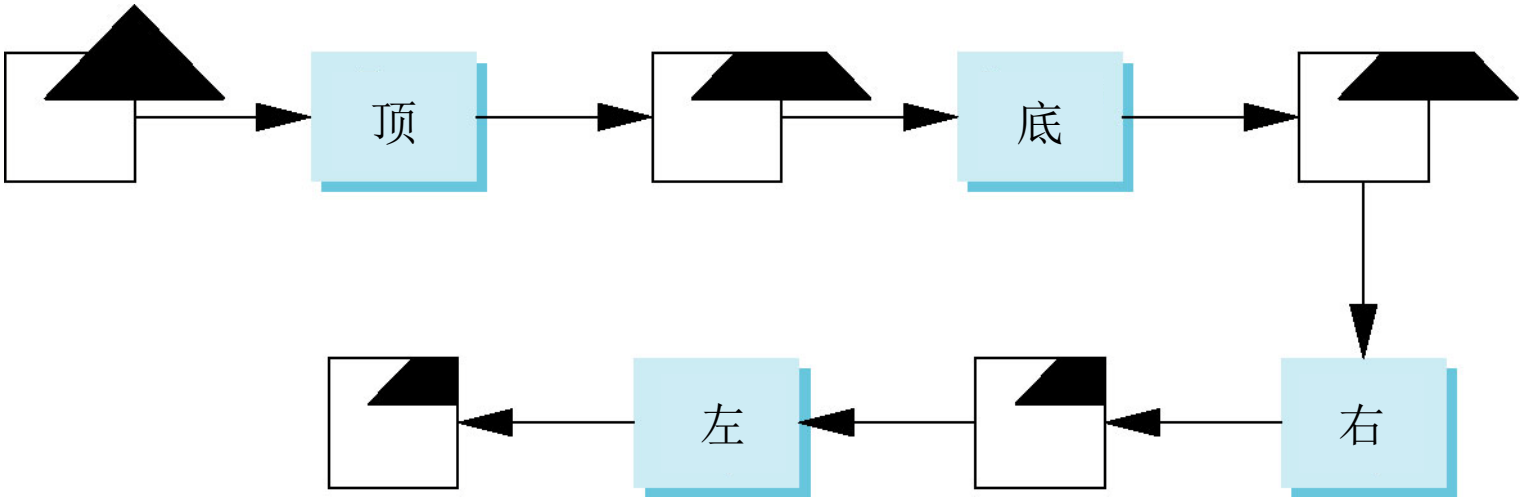
划分与凸性

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



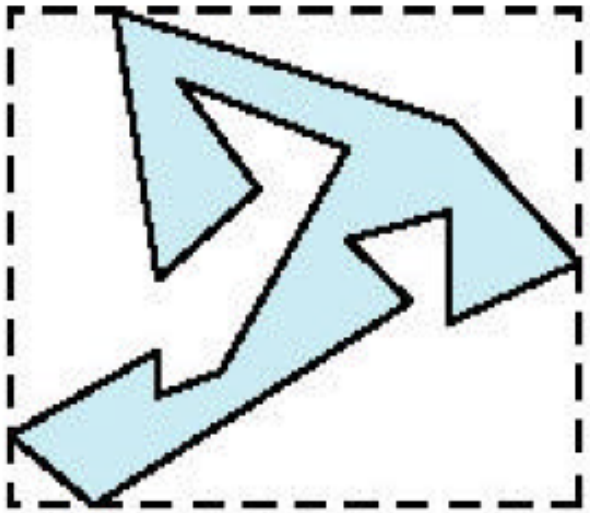
多边形裁剪的流水线体系

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



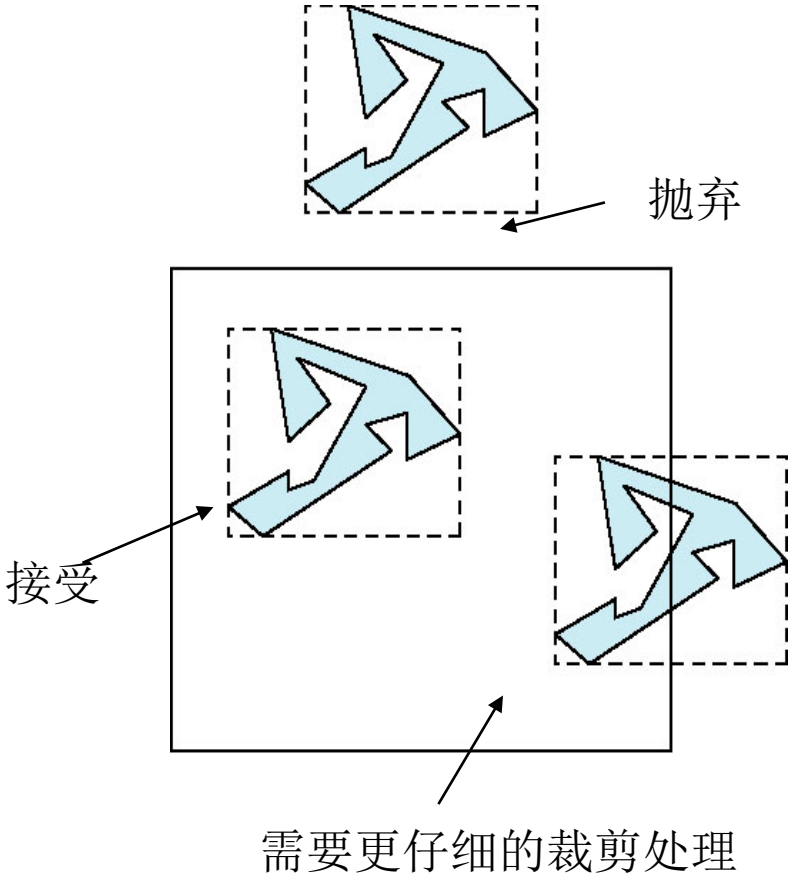
包围盒（bounding box）

- 不是直接对复杂多边形进行裁剪，而是先用一个方向与坐标轴平行的立方体或其它形状包围多边形
  - 包围盒应尽可能得小
  - 容易计算出坐标的最大值与最小值



应用包围盒

- 通过直接基于包围盒确定多边形的接受与抛弃



裁剪与可见性

- 在隐藏面消除中经常用到裁剪
- 实际上，对于裁剪与隐藏面消除，都是希望把看不到的对象从视野中去掉
- 通过可以在处理过程中提早应用可见性或者遮挡检测，从而在进入流水线体系之间消去尽可能多的多边形

Thanks for your attention!