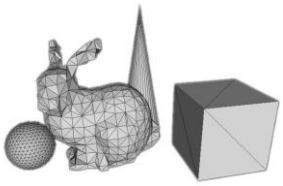


Introduction to Computer Graphics

Implement Your Pipeline 1

Ming Zeng

Software School

Xiamen University

Contact Information:

zengming@xmu.edu.cn

第六章 明暗着色(Shading)

本章主要介绍光源作用在物体上呈现出不同明暗效果的物理过程。

- 基本概念
  - 增强真实感的几种方式
  - 为什么需要明暗着色
  - 光源的类型、材料的类型
- Phong光照明模型
  - 给定光源与材质、视点，如何计算光照效果
  - Phong模型的改进版本——Blinn模型
- 多边形明暗处理
- OpenGL明暗处理

第六章 明暗着色(Shading)

本章主要介绍光源作用在物体上呈现出不同明暗效果的物理过程。

- 多边形明暗处理
  - Flat Shading
  - Gauroud Shading [要求实现]
  - Phong Shading
- OpenGL明暗处理
  - 光照模式、光源、材质属性的设置
  - 如何设置光源位置（例如：光源随视点一起运动，视点运动光源固定，物体运动光源固定等）

第七章 流水线实现
——从顶点到片段

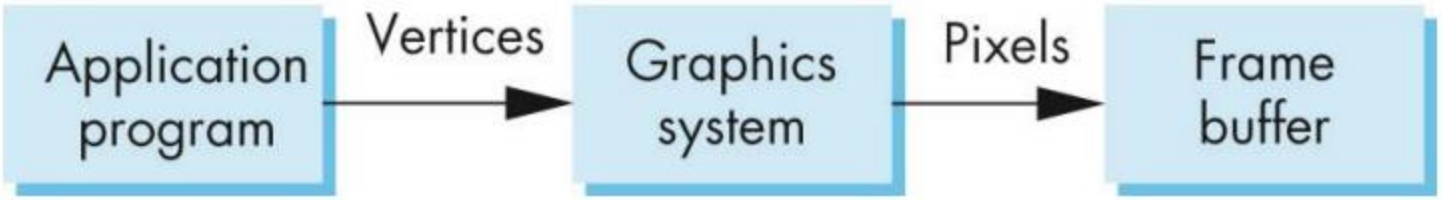
- 主要内容
  - 框架与裁剪
    - 基本实现策略
    - 线段裁剪
    - 多边形裁剪
  - 光栅化
  - 隐藏面消除

第一部分 框架与裁剪

- 框架与裁剪
  - 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



基本算法

- 绘制由不透明对象构成场景的两种方法
- 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明暗值
  - 光线跟踪框架

```
for (each_pixel)
    assign_a_color(pixel)
```

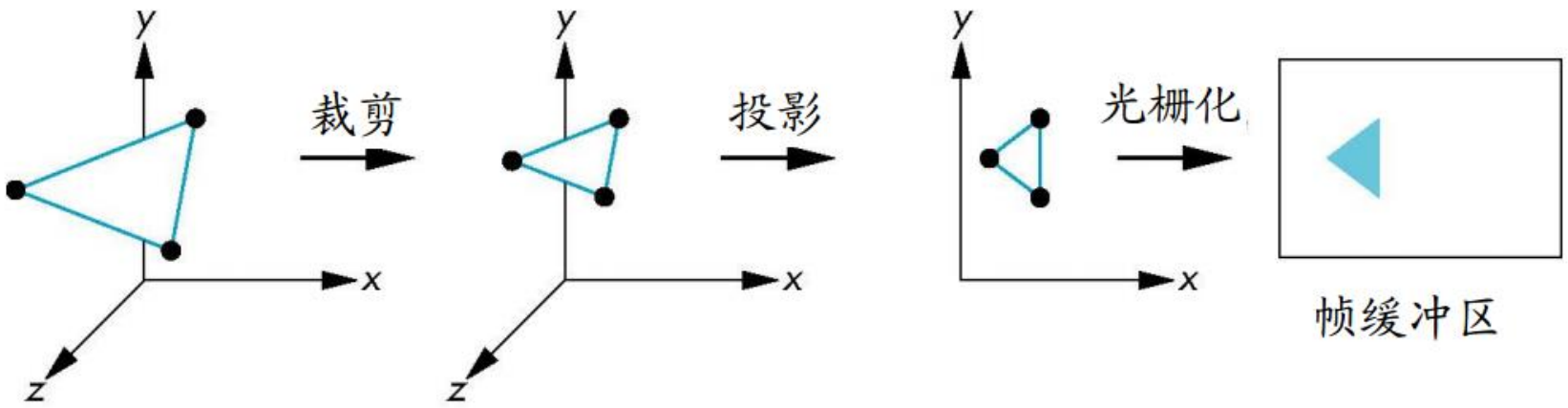
- 对于每个对象，确定它所覆盖的像素，并用对象的状态确定像素的明暗值
  - 流水线方法
  - 必须跟踪深度值

```
for (each_object)
    render(object)
```

算法类型

- 前述两种方法分别称为面向图像(image-oriented)的方法和面向对象(object-oriented)的方法
  - 也分别称为先排序(sort-first)与后排序(sort-last)方法
    - 基于隐藏面消除发生的地方

面向对象方法



面向对象方法的优缺点

- 优点：处理速度快，运行成本低
- 缺点：
  - 过去需要占用大量内存，每个对象单独处理的代价较大。由于低价大容量内存，以及专用硬件芯片的出现，已不再是问题
  - 不能得到大多数全局效果，例如涉及多个对象的复杂光照
    - 隐藏面消除例外

面向图像方法的优缺点

- 优点：
  - 任何时刻只需有限的显示内存，且可以按显示器的刷新速度和次序生成像素
  - 利用相邻像素间的连贯性可设计增量式算法
  - 适合处理全局效果
- 缺点：
  - 需要复杂的几何对象的数据结构
    - 必须确定哪些图元影响哪些像素
    - 绘制过程中需要随时访问所有的几何数据

四个主要任务

- 建模
- 几何处理
  - 投影、图元装配、裁剪、明暗处理
- 光栅化
- 片段处理
  - 纹理映射、融合、隐藏面消除、反走样

几何处理

- 几何处理作用于顶点数据，目标是确定可以显示在屏幕上的几何对象，并确定这些对象顶点的明暗值或颜色值
  - 投影
  - 图元装配
  - 裁剪
  - 明暗处理 (只是计算顶点处光照)
- 这些处理统称为前端处理(front-end processing)
  - 都用到浮点运算，且是逐顶点处理

投影

- 第一步：利用模型-视图变换把几何对象从对象坐标系变换到照相机坐标系或视点坐标系
- 第二步：利用投影变换把顶点变换到规范视景体
  - 顶点表示为裁剪坐标
  - 规范化处理把透视投影和正交投影都变换为视景体为立方体的简单正投影，简化了后期的裁剪处理

图元装配

- 变换是作用在顶点上的，而进行后续操作（裁剪或光栅化）需要把顶点组装成几何对象，称为图元装配(primitive assembly)
  - 裁剪是针对图元进行的
  - 光栅器不能对顶点单独进行处理

裁剪器

- 裁剪器(clipper)确定哪些图元或图元的哪些部分会被传送到光栅器，最终可能会显示在屏幕上
  - 只有在视景体内的对象经过光栅化后能被显示在屏幕上
  - 部分在视景体内的图元，裁剪后生成新的图元
  - 没被裁剪掉的顶点仍表示为四维齐次坐标，通过透视除法转换为三维的规范化设备坐标

明暗处理

- 明暗处理为每个顶点赋颜色值
  - 颜色要么由图形绘制系统的当前颜色决定
  - 当开启光照后，则由改进的Phong光照模型计算得到

光栅化

- 光栅化(rasterization)或扫描转换(scan conversion)：从裁剪后的几何对象生成片段(准像素)
  - 对于线段，确定哪些像素可用来近似表示顶点间的线段
  - 对于多边形，确定哪些像素位于多边形顶点定义的二维区域的内部
  - 复杂的曲线和曲面，用多条线段和多个多边形来近似表示
  - 片段颜色由顶点的明暗值插值得到

片段处理

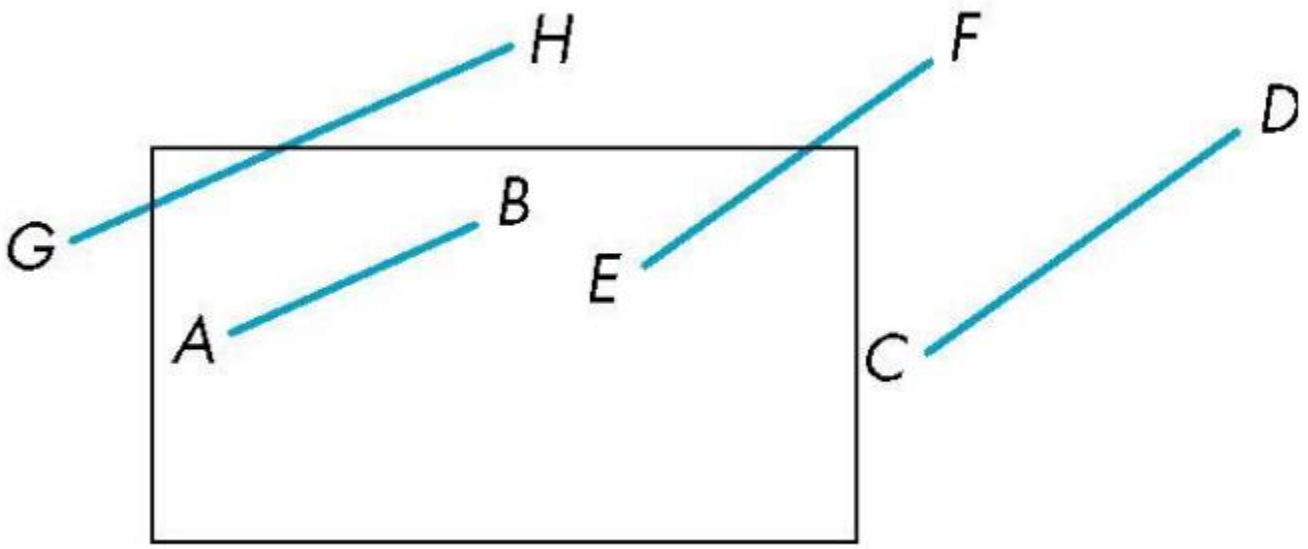
- 在最简单的情形下，片段颜色由光栅器赋值，并且该值就是片段位置对应的帧缓冲区中像素的颜色值
- 隐藏面消除：确定可见对象的片段值
  - 可见对象指位于视景体内，且没有被其他离视点更近的不透明对象遮挡的对象
  - 利用几何处理后仍保留的顶点深度信息
- 纹理贴图
- 融合：半透明效果，反走样

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
- OpenGL中，在光栅化之前使用一个三维视景体对图元进行裁剪
- 二维相对于裁剪窗，三维相对于裁剪体

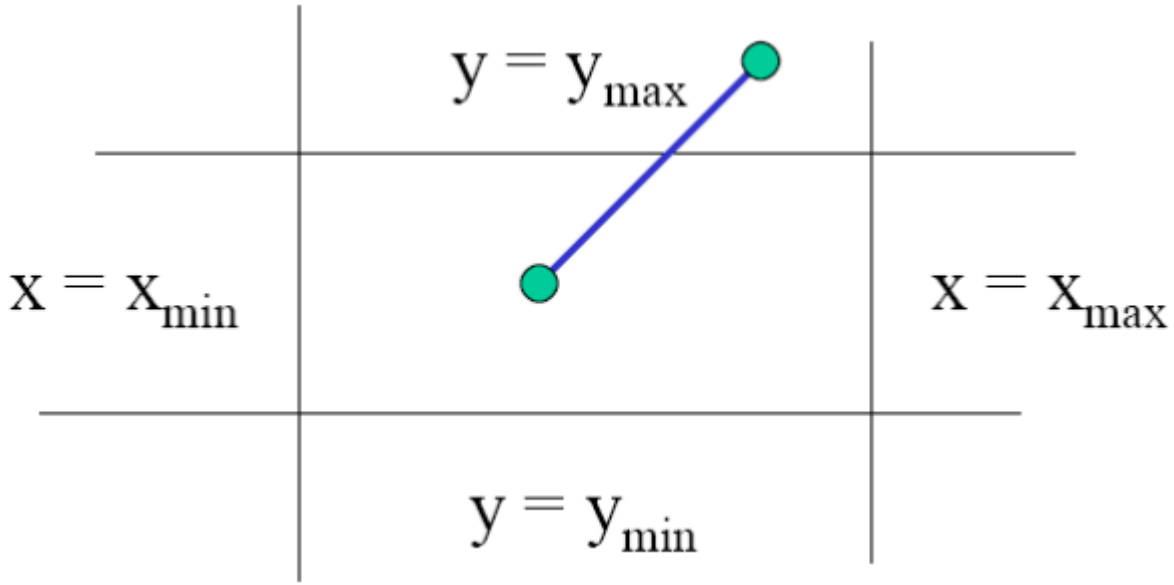
二维线段的裁剪

- 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



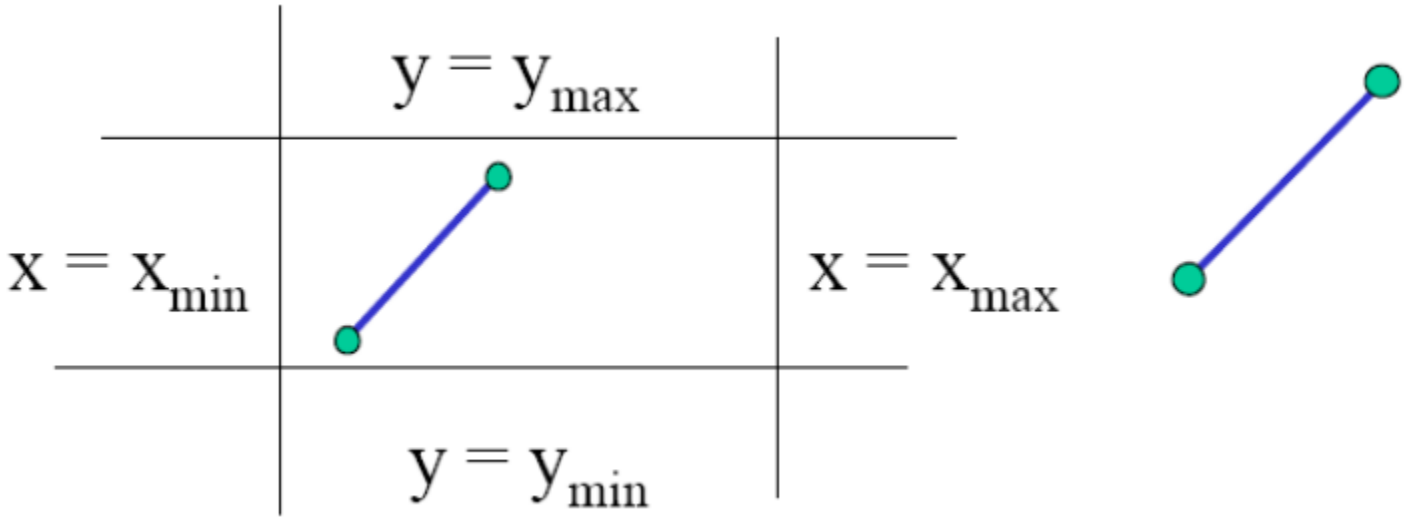
Cohen-Sutherland算法

- 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



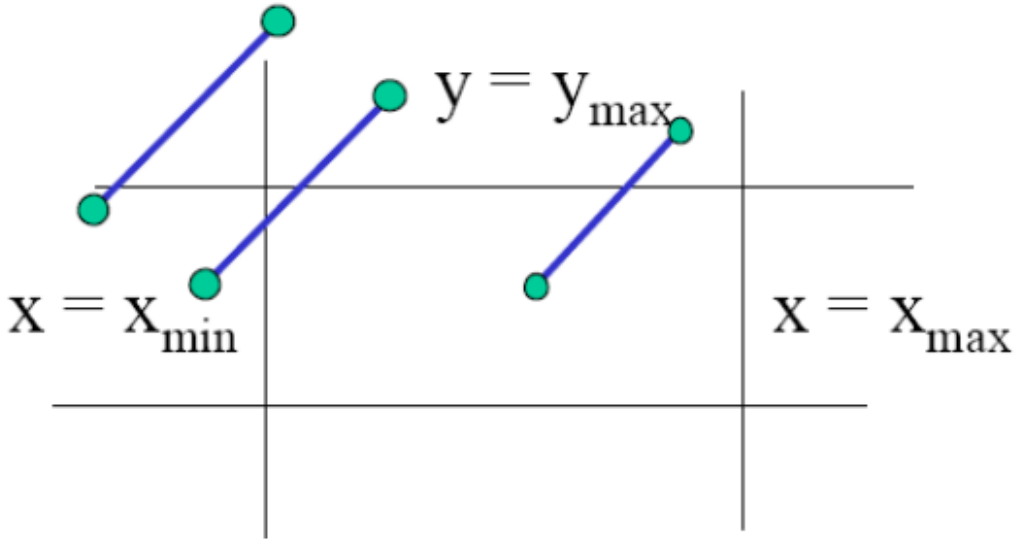
各种情形

- 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



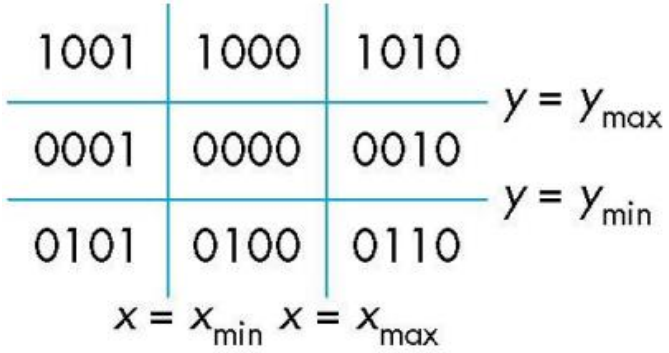
各种情形

- 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 [何时需要两次]
- Case 4: 都在外部
  - 仍可能部分在内部
  - 必须进行至少一次求交 [何时需要两次？]



定义编码

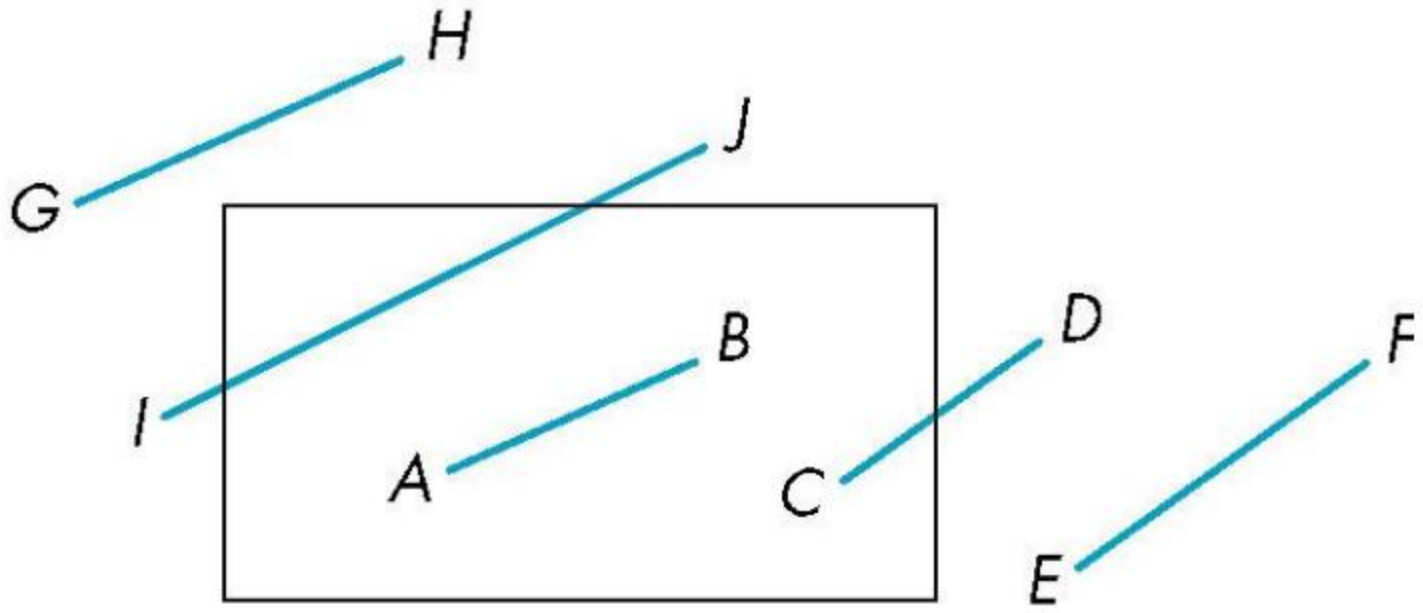
- 对于每个端点，定义一个编码$b_0b_1b_2b_3$
  - 如果$y>y_{max}$, b_0 =1，否则=0
  - 如果$y<y_{min}$, b_1 =1，否则=0
  - 如果$x>x_{max}$, b_2 =1，否则=0
  - 如果$x<x_{min}$, b_3 =1，否则=0



- 编码把空间分成九个区域
- 计算编码最多需要四次比较

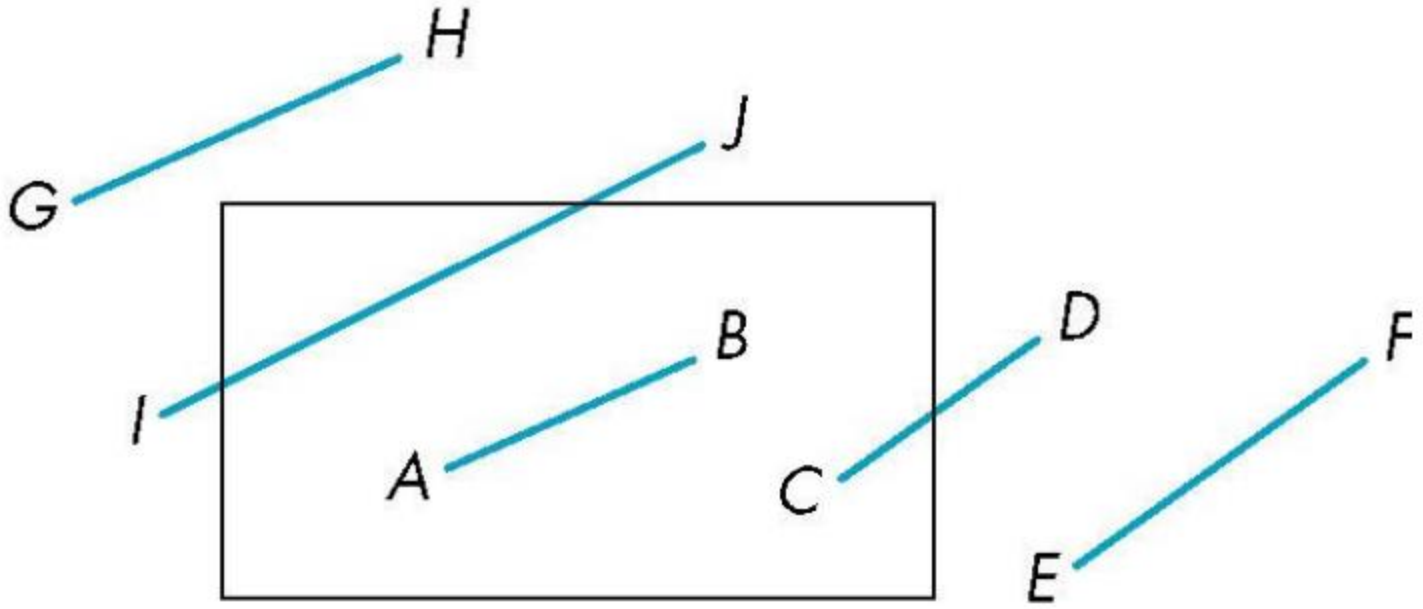
应用编码

- 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



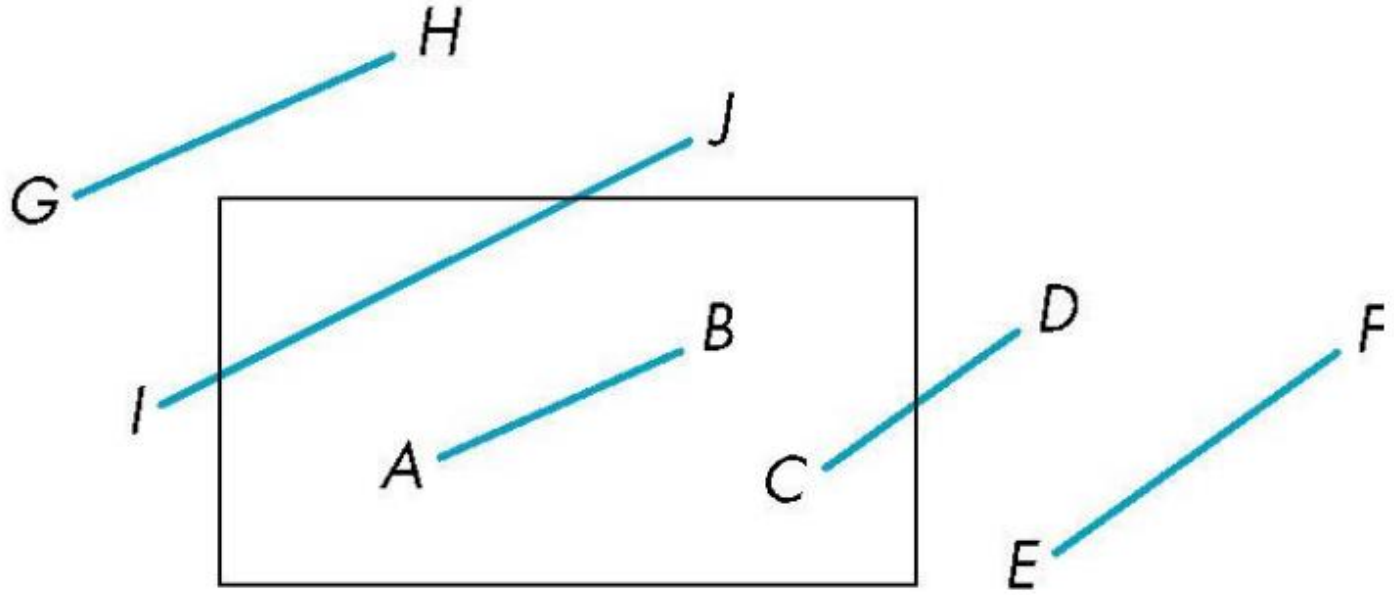
应用编码

- 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



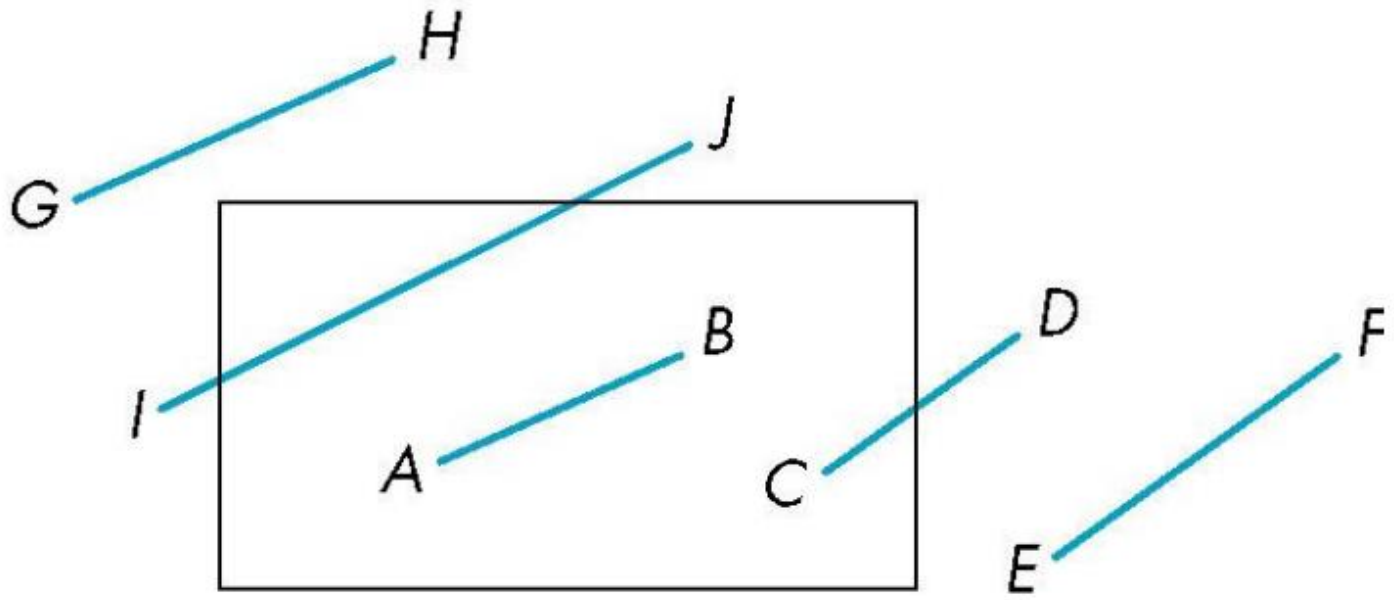
应用编码

- 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



应用编码

- 请说明EF的情况

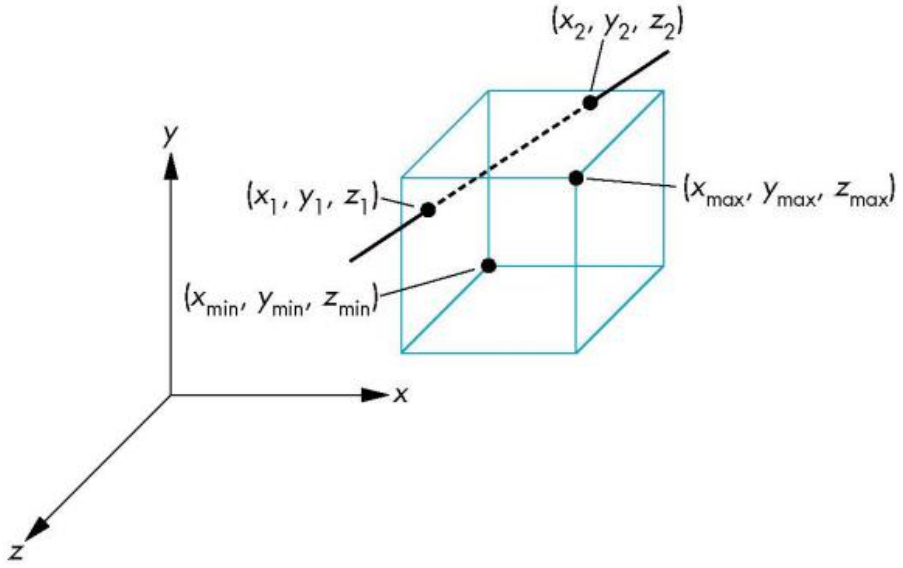


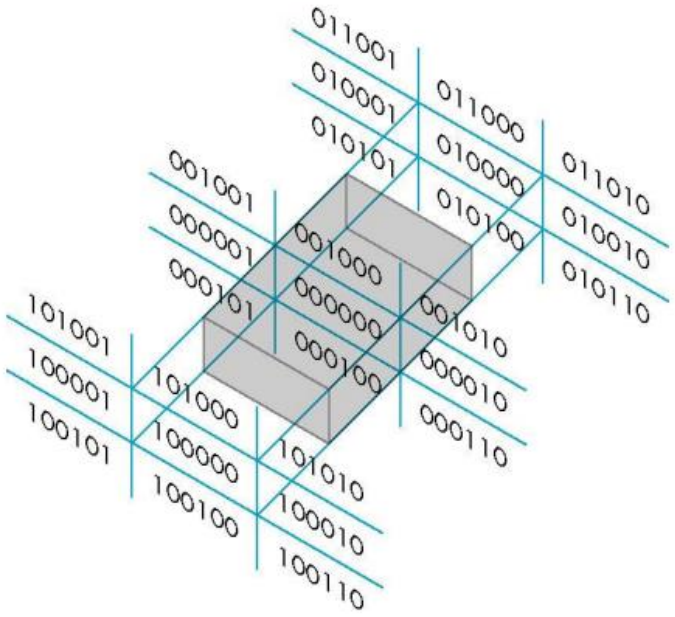
效率

- 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

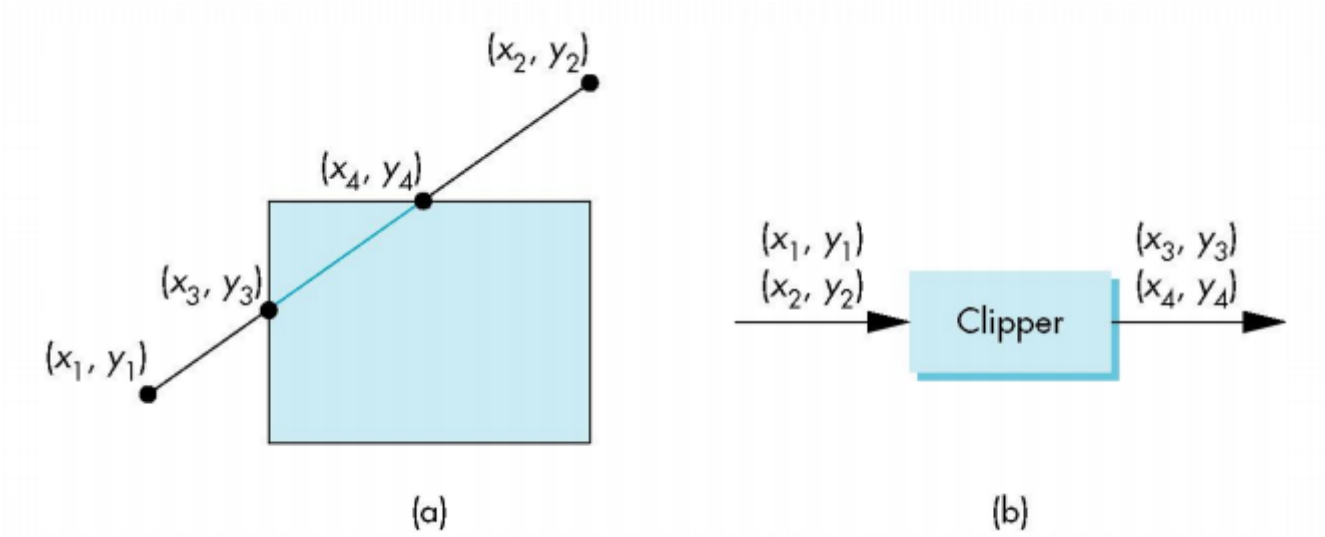
- 利用6位进行编码
- 必要时，相对于平面裁剪线段





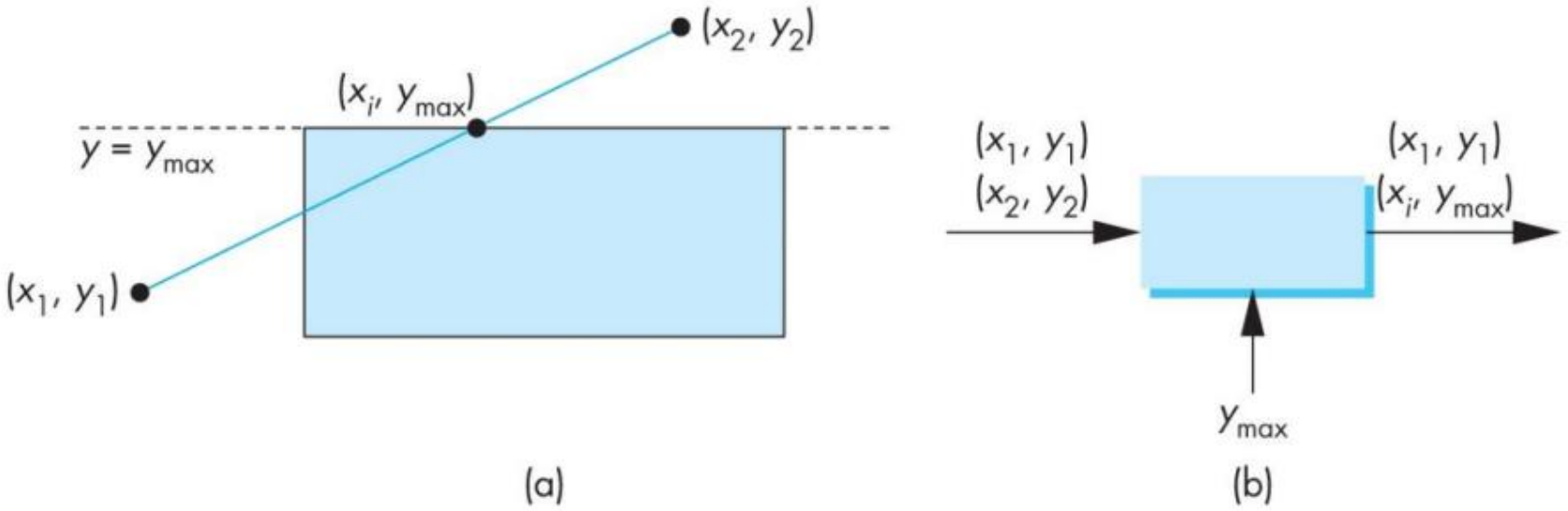
Sutherland-Hodgeman算法

- 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

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

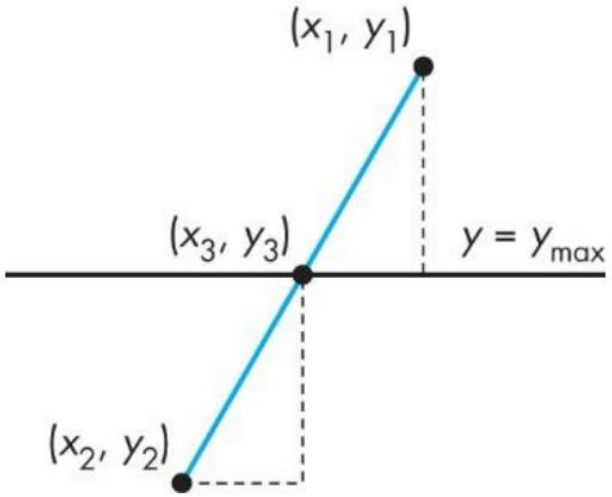
$y_3 = y_{max}$

- 裁剪器返回下面三对顶点中的某一对：

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

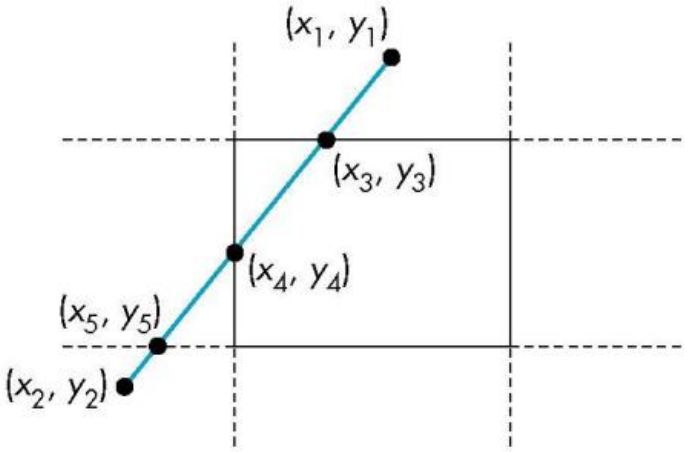
$\{(x_1,y_1), (x_i, y_{max})\}$

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

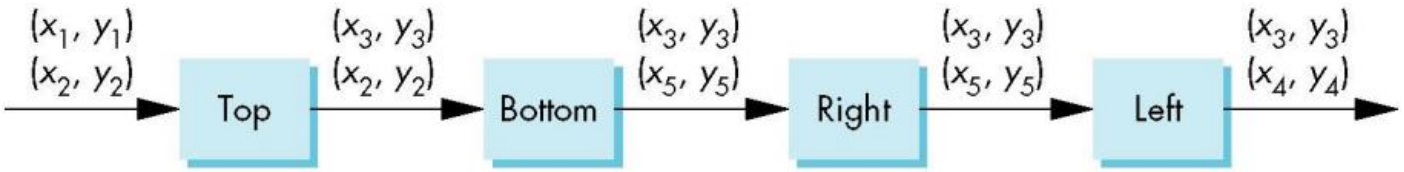


线段裁剪的流水线体系

- 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)

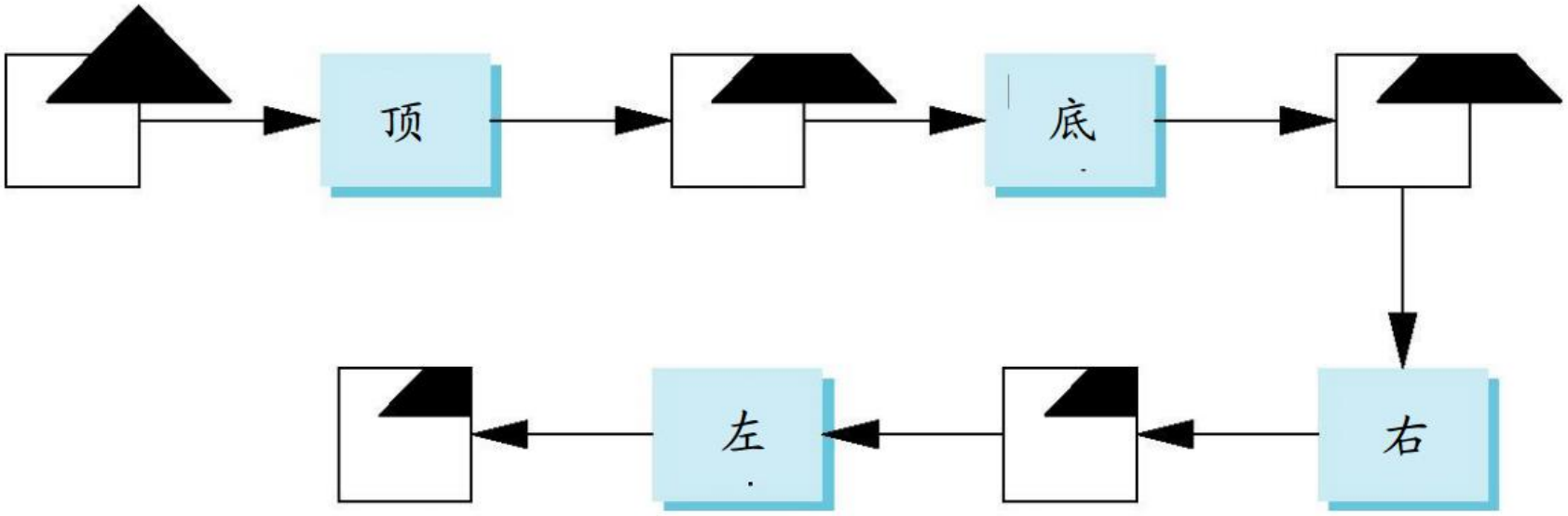
多边形的裁剪

- 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

- 然而裁剪凸多边形最多得到一个凸多边形

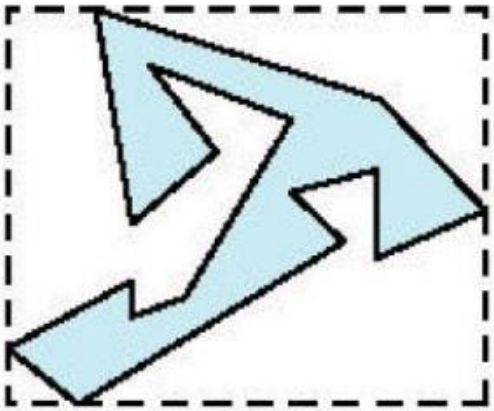
多边形裁剪的流水线体系

- 三维：增加前与后裁剪器



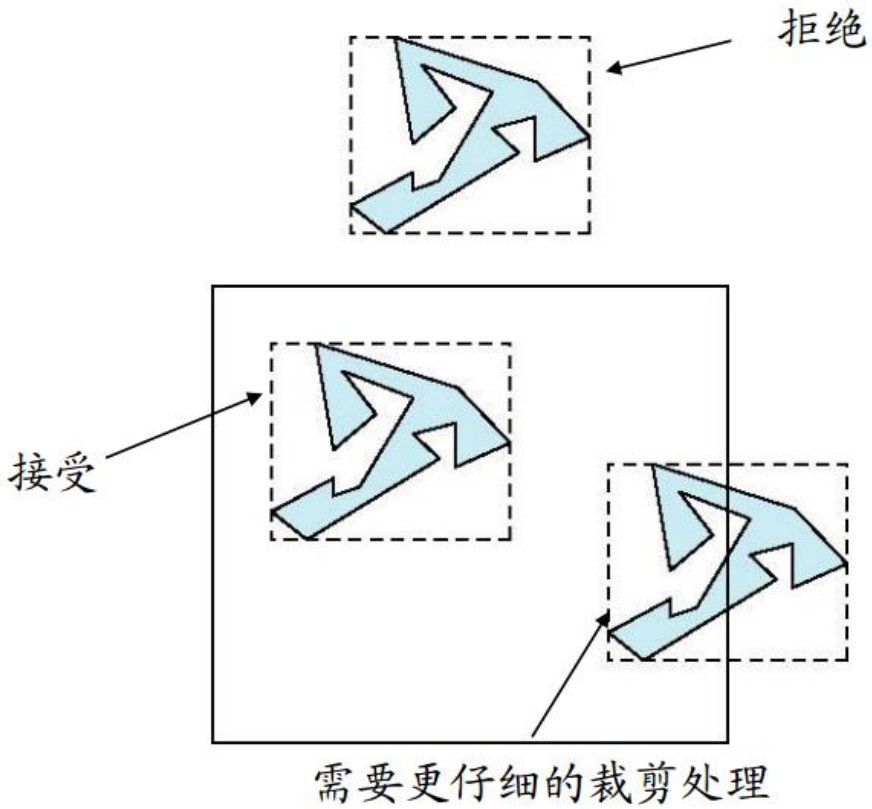
应用包围盒辅助裁剪

- 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



应用包围盒辅助裁剪

- 直接基于包围盒确定多边形的接受与拒绝
- 轴对齐包围盒在二维和三维情形下都适用



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

本节小结

- 从顶点到片元的整个过程
- 裁剪
  - 线段裁剪： Cohen-Sutherland算法，Sutherland-Hodgeman算法，每个算法都包括二维和三维两种情形
  - 多边形裁剪算法： Sutherland-Hodgeman算法，Weiler-Atherton算法 （课外参考）