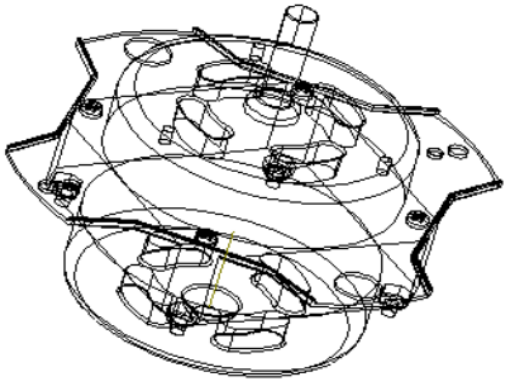
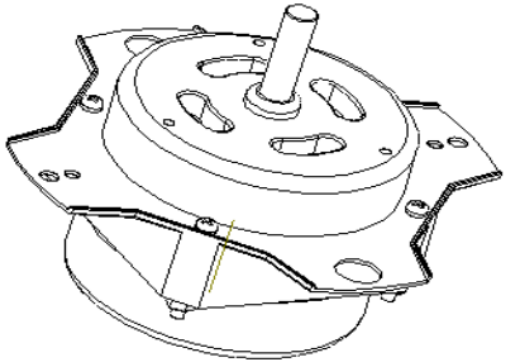
第5讲：消隐

吴文明
计算机与信息学院

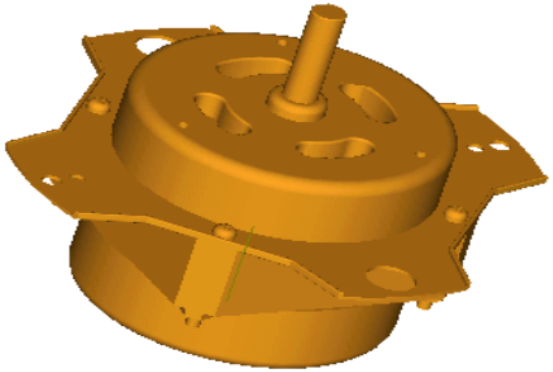
导论



线框图

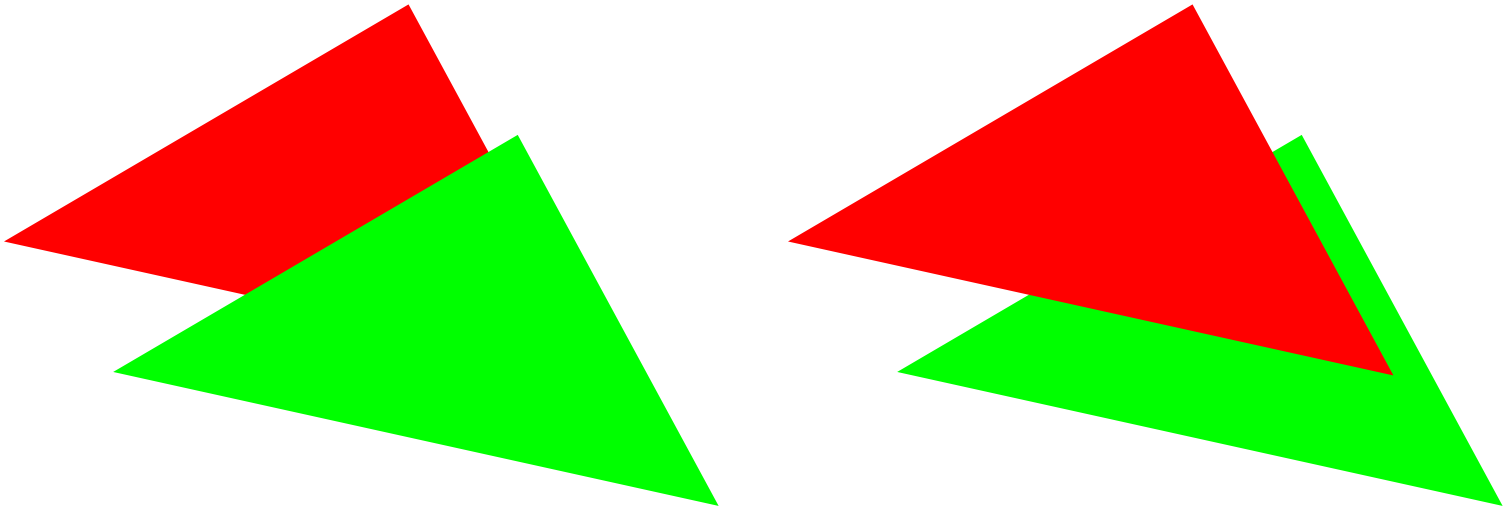


消隐图

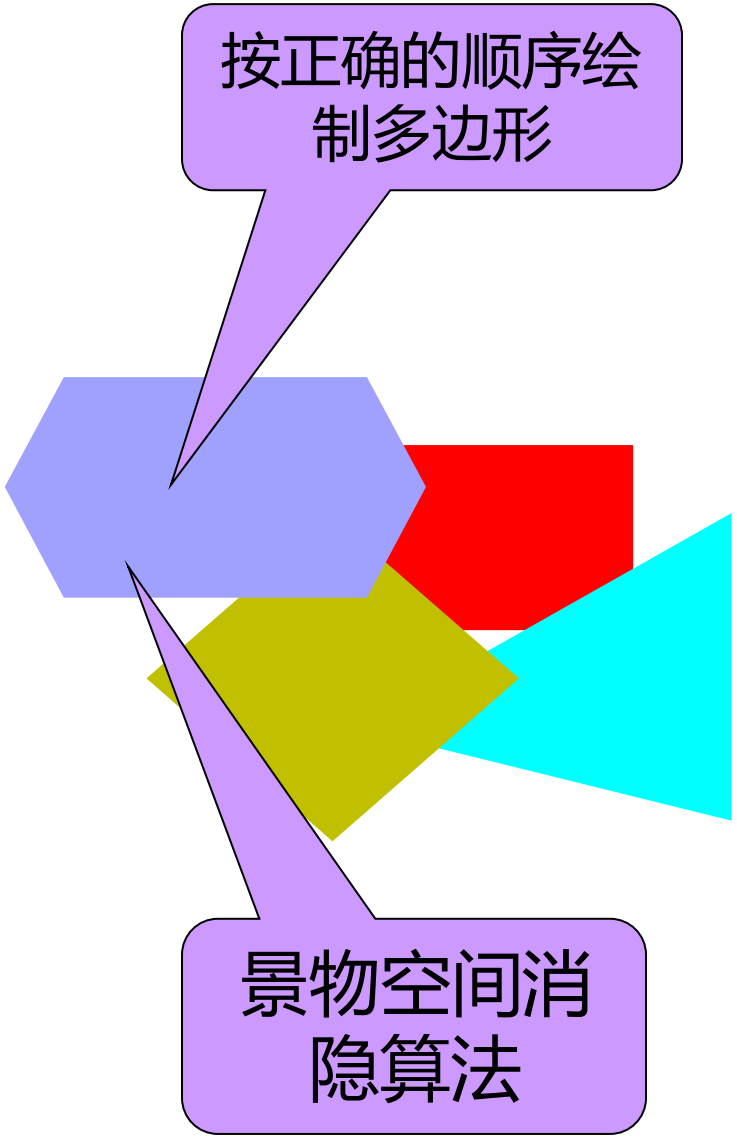


真实感图形

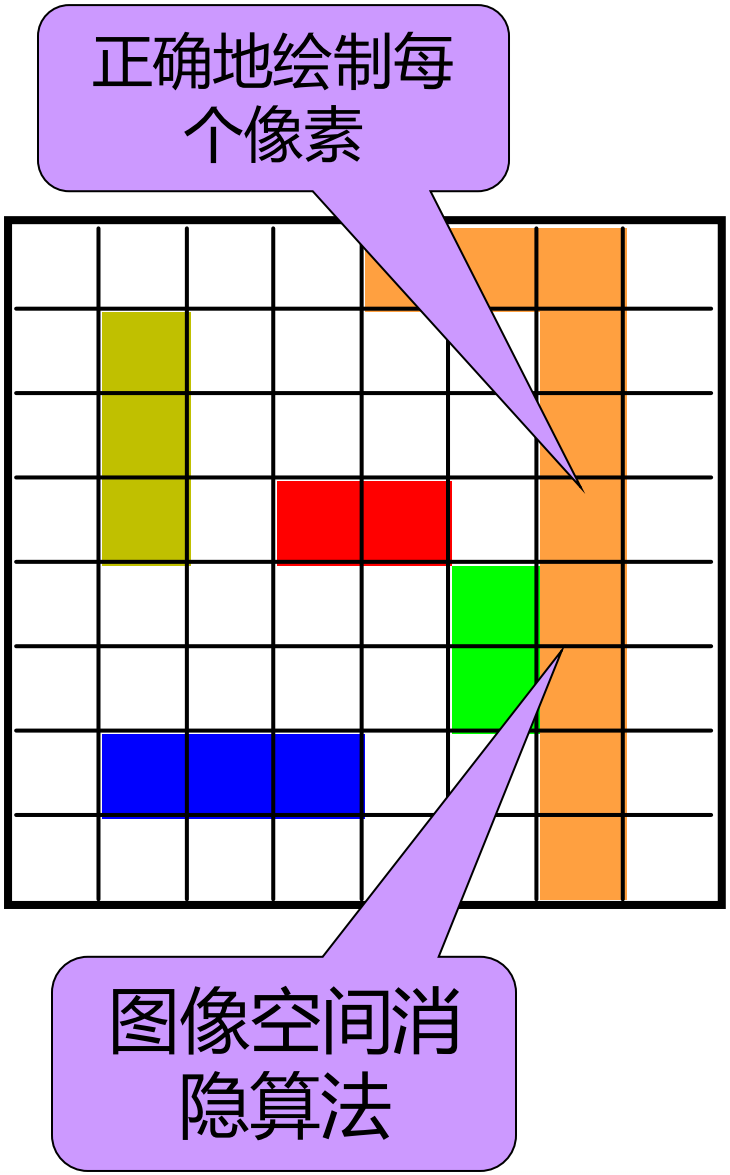
导论



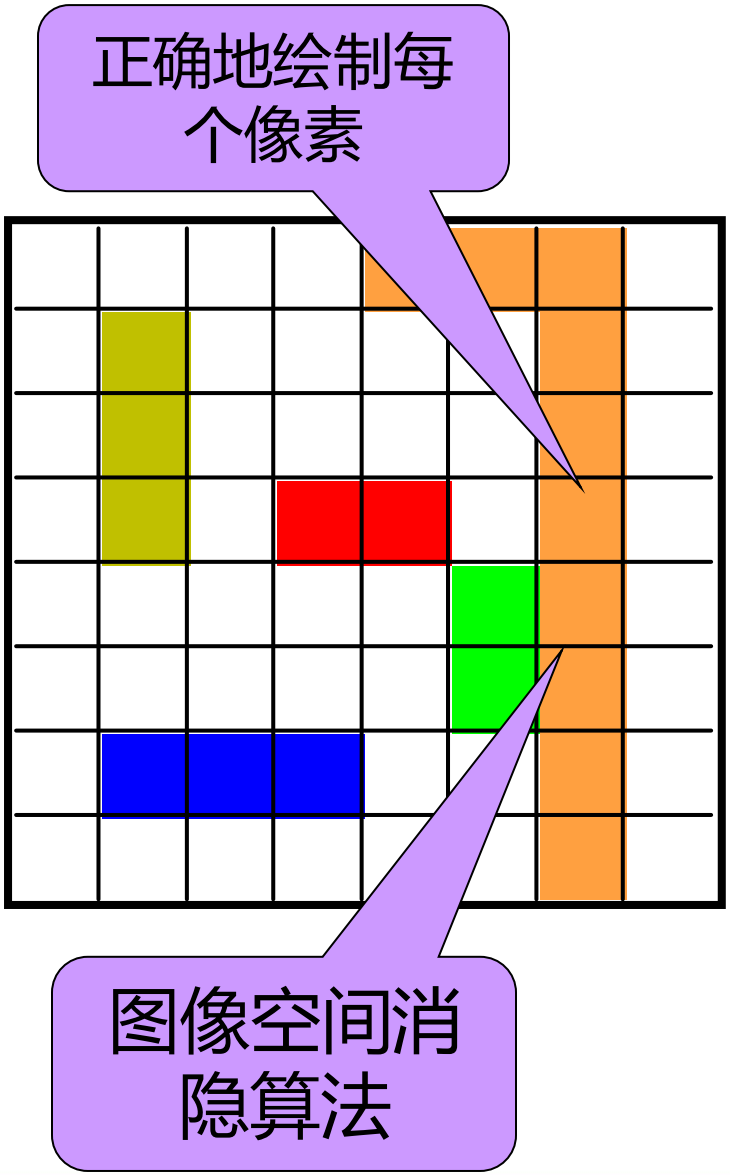
导论



导论



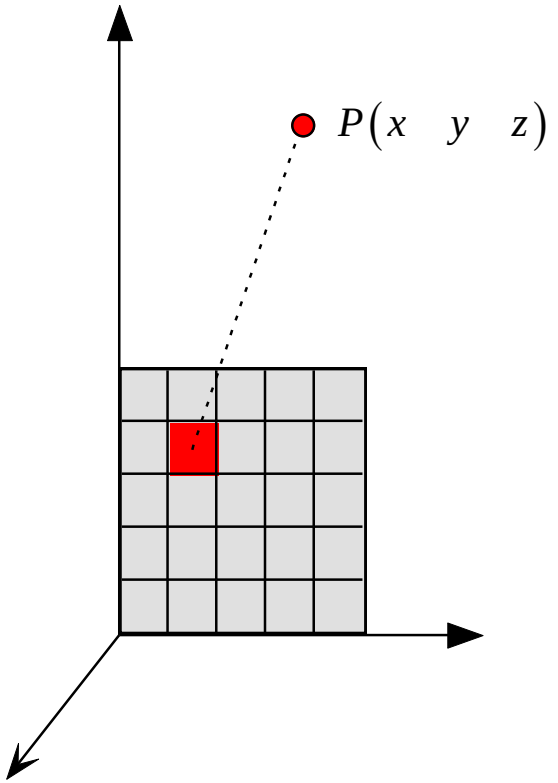
导论



消隐算法

算法	描述
深度缓存器算法	图像空间
区间扫描线算法	图像空间
深度排序算法	图像空间、景物空间之间
区域细分算法	图像空间、景物空间之间
光线投射算法	图像空间、景物空间之间
BSP算法	景物空间
多边形区域排序算法	景物空间

深度缓存器算法（Z-buffer）



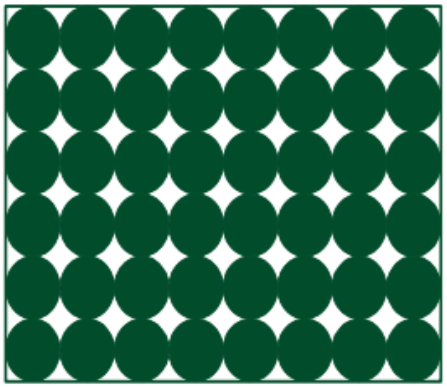
基本过程：

- 将点P（x，y，z）投影到像素（x，y）
- 根据P的属性设置像素（x，y）的颜色

问题：如果多个点P1(1, 2, 3)，P2(1，2，5）等投影到同一个像素，怎么办？

深度缓存器算法

屏幕



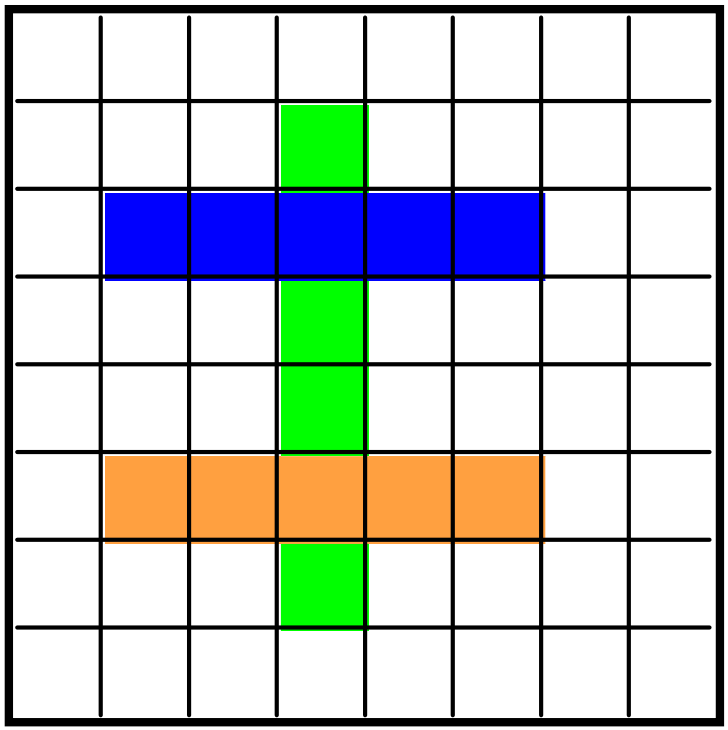
帧缓冲器

每个单元存放对应
象素的颜色值

Z缓冲器

每个单元存放对应
象素的深度值

深度缓存器算法



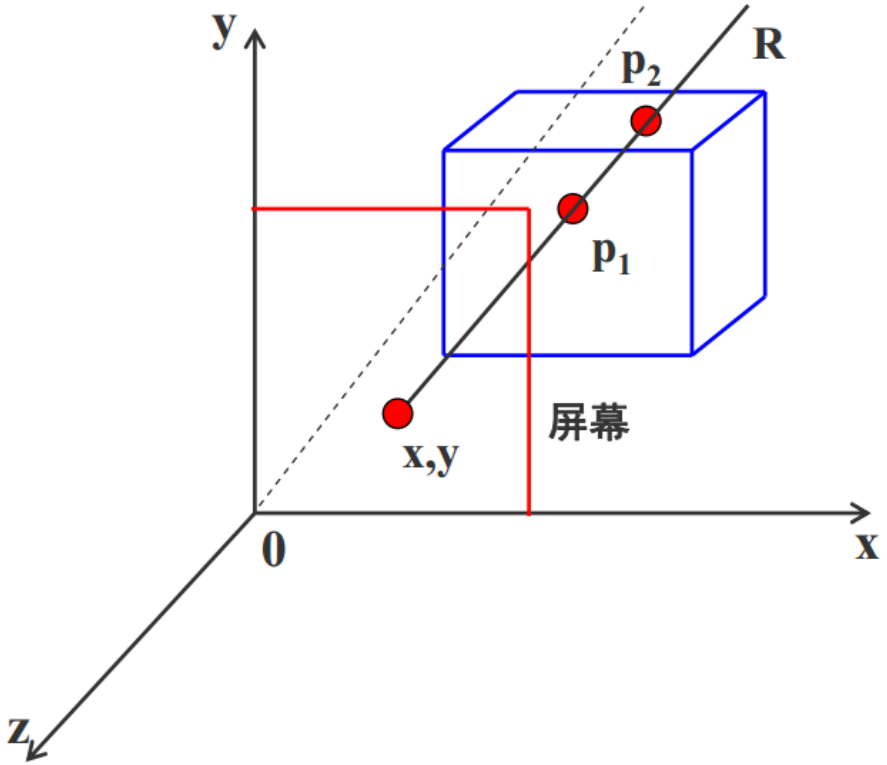
-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	0.0	-1	-1	-1	-1
-1	0.1	0.1	0.1	0.1	0.1	-1	-1
-1	-1	-1	0.0	-1	-1	-1	-1
-1	-1	-1	0.0	-1	-1	-1	-1
-1	0.2	0.2	0.2	0.2	0.2	-1	-1
-1	-1	-1	0.0	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1

深度缓存器算法

假定xoy面为投影面，z轴为观察方向

过屏幕上任意像素点（x, y）作平行于z轴的射线R，与物体表面相交于p_1和p_2点

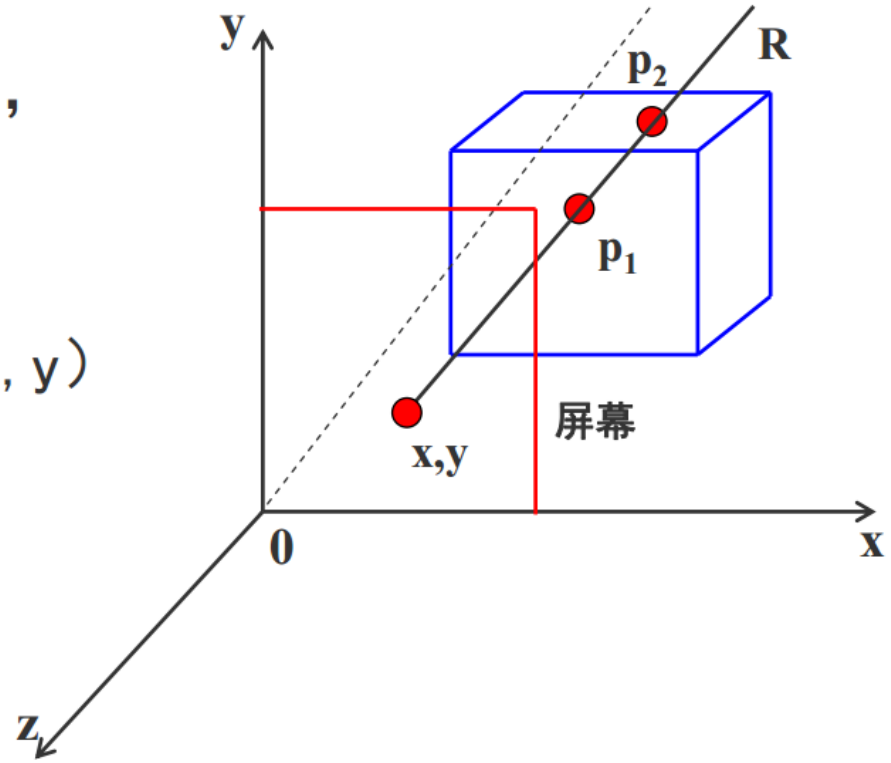
p_1和p_2点的z值称为该点的深度值



深度缓存器算法

z-buffer算法比较p_1和p_2的z值，
将最大的z值存入z缓冲器中

显然，p_1在p_2前面，屏幕上（x, y）
这一点将显示p_1点的颜色



深度缓存器算法

- 初始化：把Z缓存中各(x, y)单元置为z的最小值，而帧缓存各(x, y)单元置为背景色。

- 在把物体表面相应的多边形扫描转换成帧缓存中的信息时，对于多边形内的每一采样点(x, y)进行处理：
  - 计算采样点(x, y)的深度z(x, y)；
  - 如z(x, y)大于Z缓存中在(x, y)处的值，则把z(x, y)存入Z缓存中的(x, y)处，再把多边形在z(x, y)处的颜色值存入帧缓存的(x, y)中。

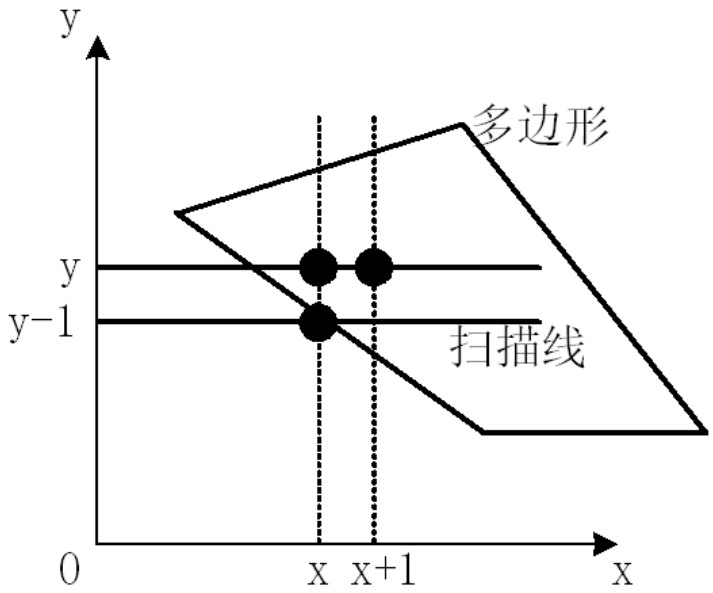
深度缓存器算法

问题：**计算采样点(x, y)的深度z(x, y)**。

假定多边形的平面方程为：Ax+By+Cz+D=0。

$$z(x,y)=\frac{-Ax-By-D}{C}$$

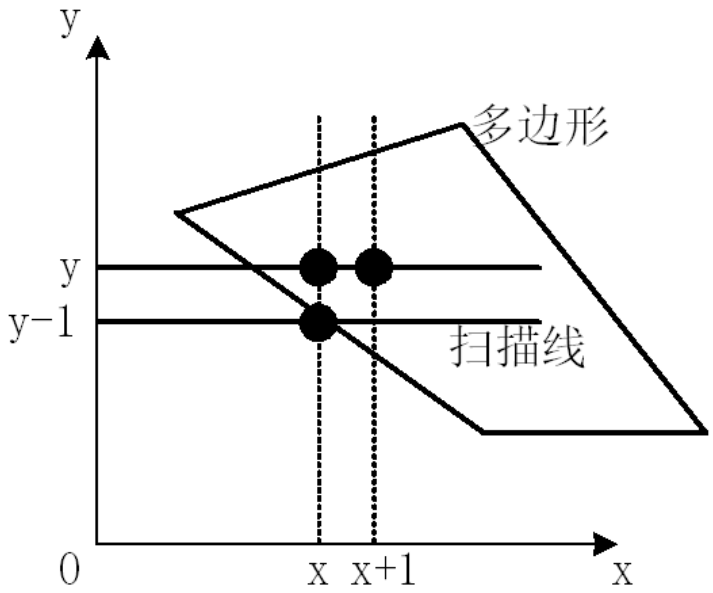
深度缓存器算法



扫描线上所有后继点的深度值：

$$z(x+1,y)=\frac{-A(x+1)-By-D}{C}=z(x,y)-\frac{A}{C}$$

深度缓存器算法



- 当处理下一条扫描线y=y−1时，该扫描线上与多边形相交的最左边（x最小）交点的x值可以利用上一条扫描线上的最左边的x值计算：

$$x\big|_{y-1,\min} = x\big|_{y,\min} - \frac{1}{k}$$

深度缓存器算法

$$z(x|_{y-1,\min}, y-1) = \frac{-Ax|_{y-1,\min} - B(y-1) - D}{C}$$

$$= \frac{-A(x|_{y,\min} - \frac{1}{k}) - B(y-1) - D}{C}$$

$$= z(x|_{y,\min}, y) + \frac{\frac{A}{k} + B}{C}$$

深度缓存器算法

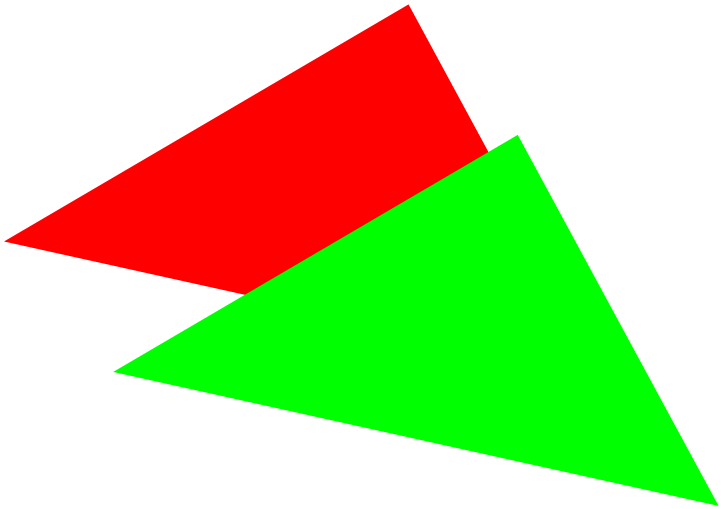
优点：

- 简单
- 便于硬件实现

缺点：

- 占用太多内存
- 在实现反走样、透明、半透明效果困难

区间扫描线算法



- 避免对被遮挡区域的采样———以提高扫描线算法的计算效率

区间扫描线算法

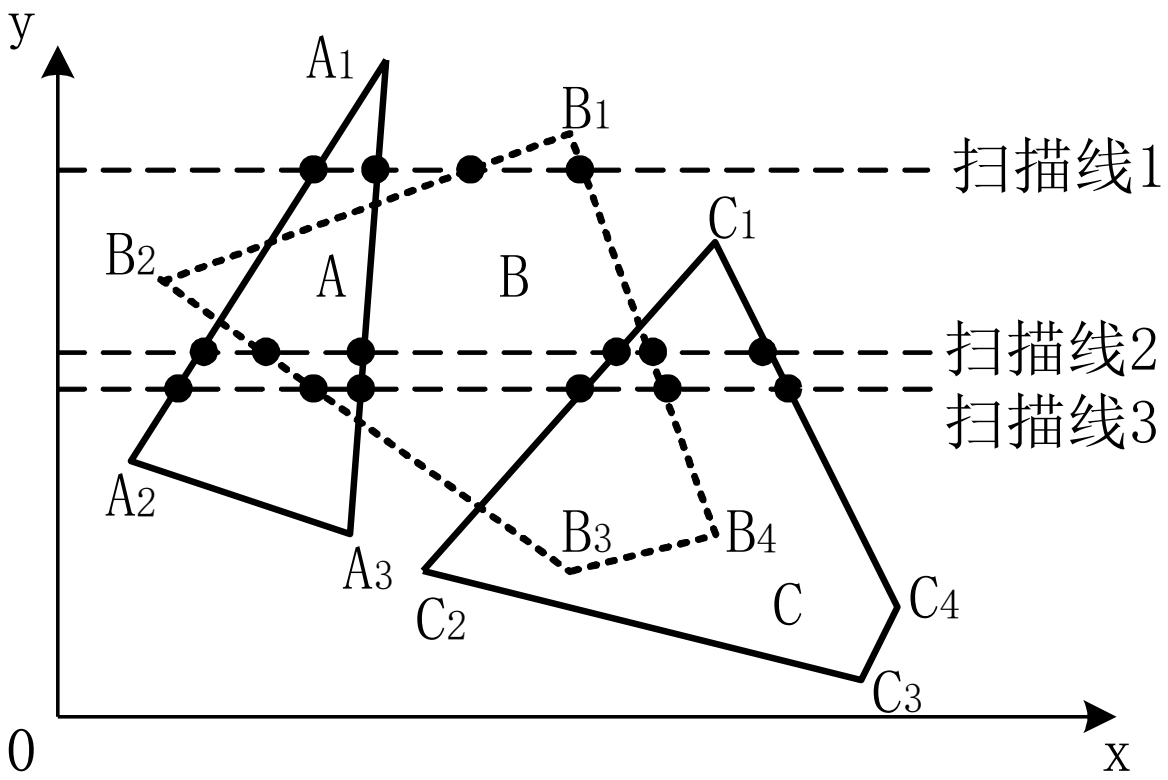
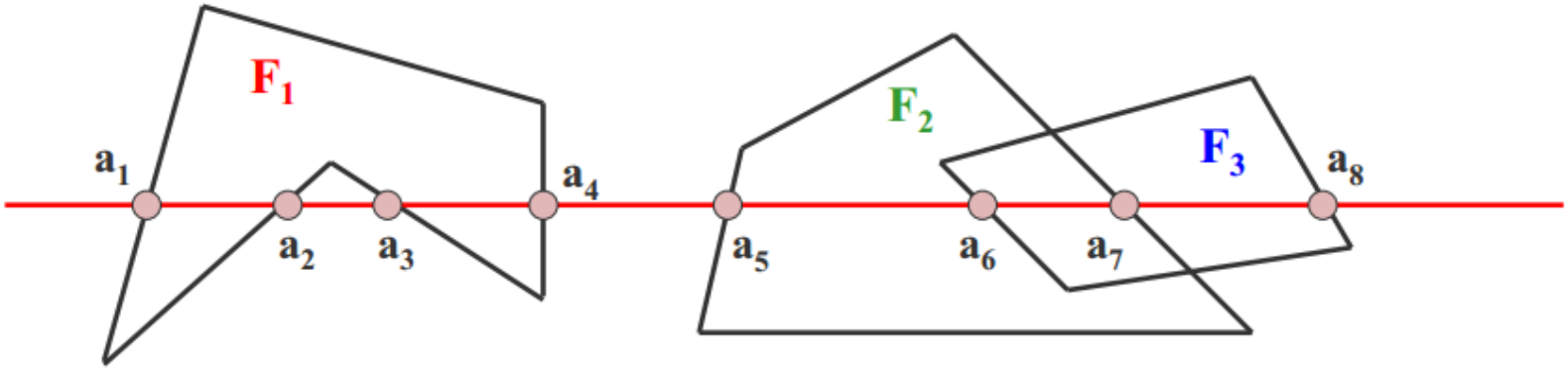


图9-3　区间扫描线算法原理

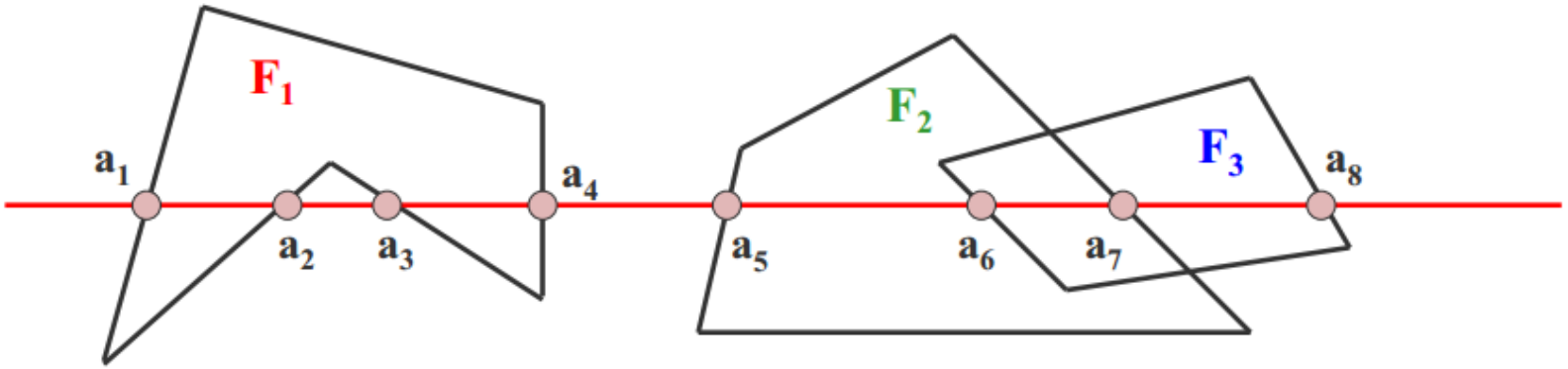
区间扫描线算法



扫描线的交点把这条扫描线分成了若干个区间，每个区间上必然是同样一种颜色

对于有重合的区间，如a6a7这个区间，要么显示F2的颜色，要么显示F3的颜色，不会出现颜色的跳跃

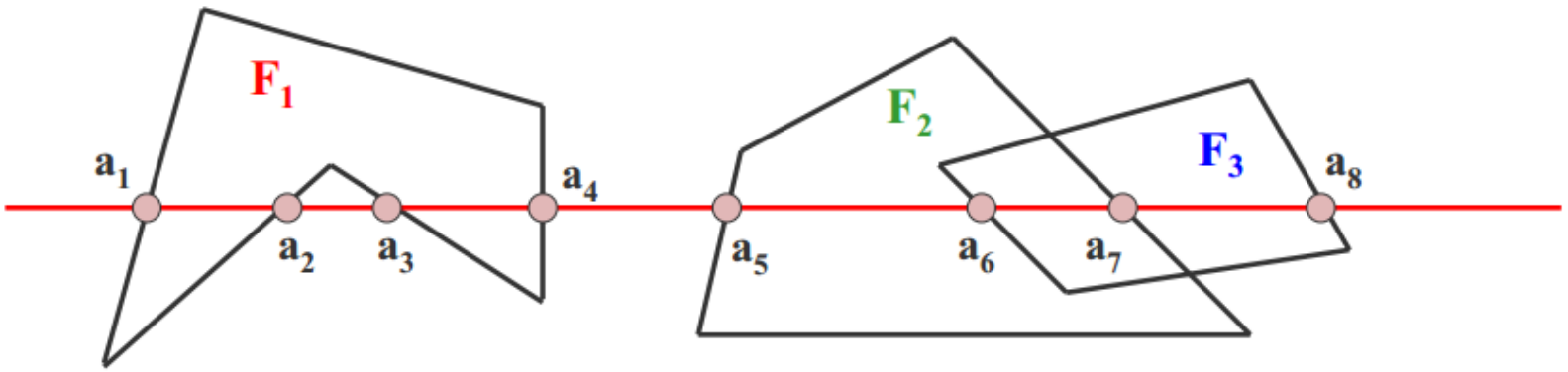
区间扫描线算法



如果把扫描线和多边形的这些交点都求出来，对每个区间，只要判断一个像素的要画什么颜色，那么整个区间的颜色都解决了，这就是区间扫描线算法的主要思想

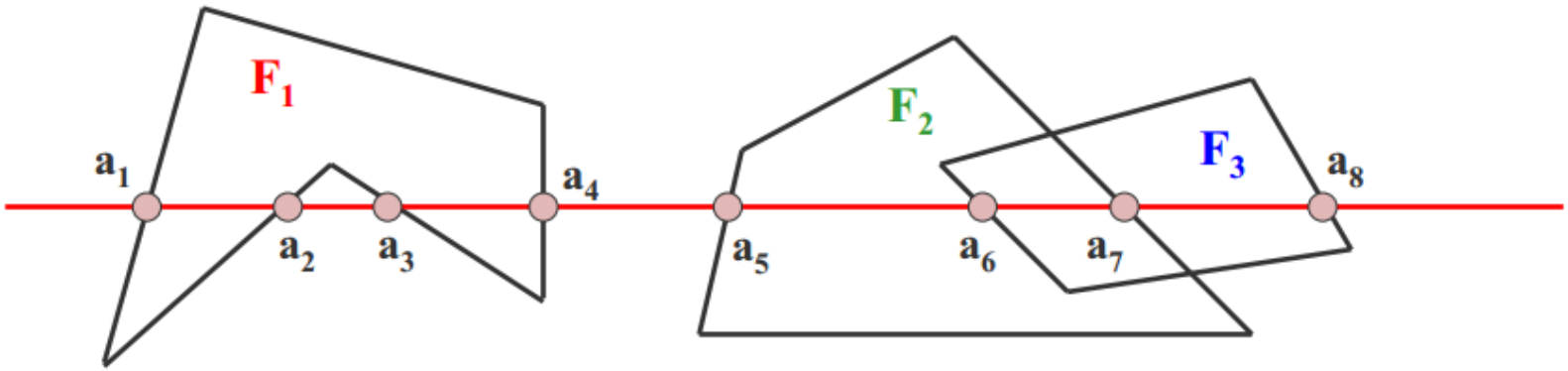
算法的优点：将象素计算改为逐段计算，效率大大提高！

区间扫描线算法



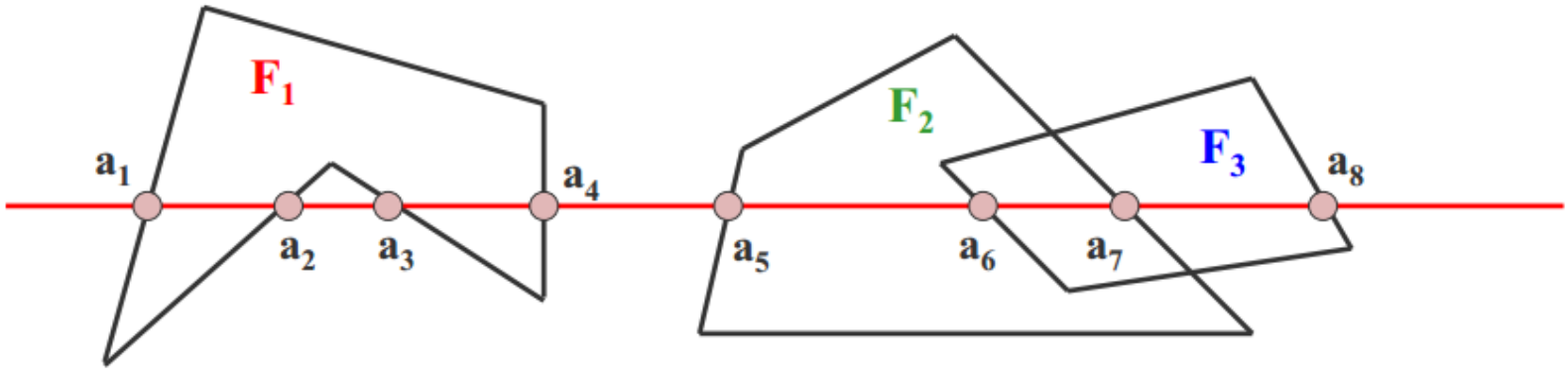
首先要有投影多边形，然后求交点，然后交点进行排序排序

区间扫描线算法



（1）小区间上没有任何多边形，如[a4, a5]，用背景色显示

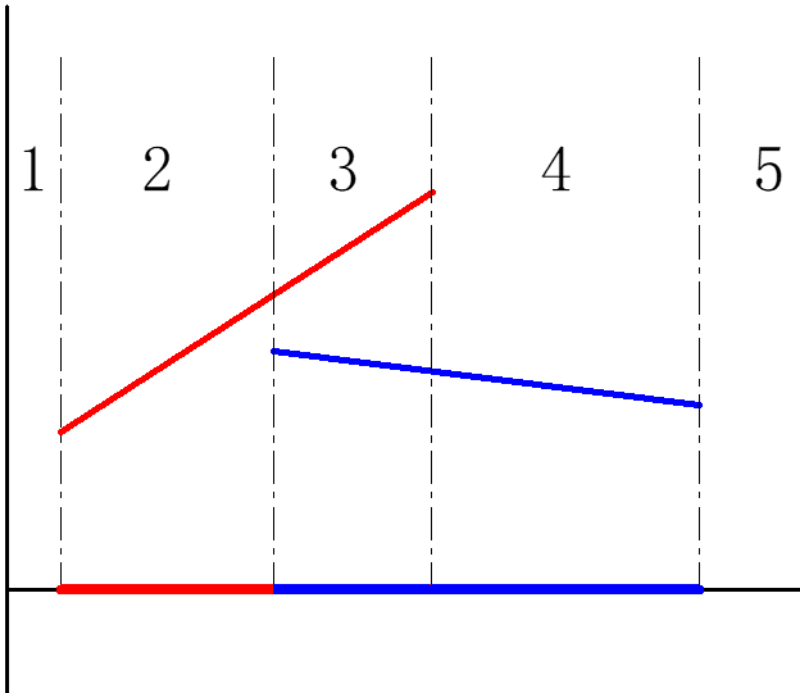
区间扫描线算法



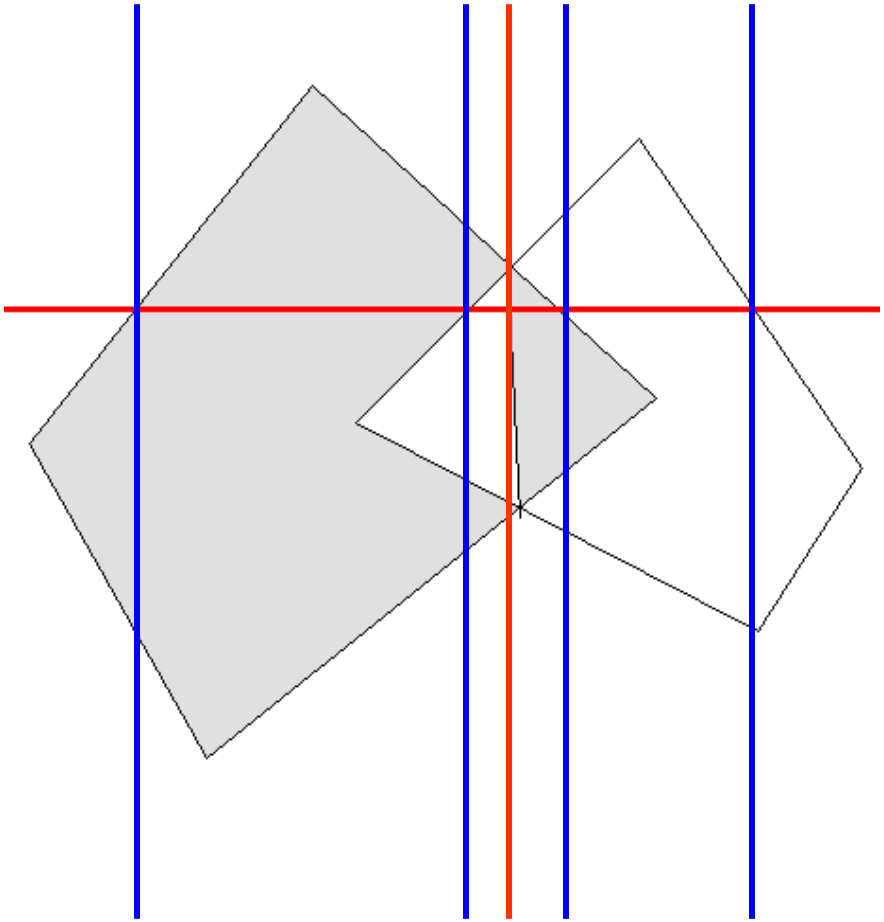
（2）小区间只有一个多边形，如[a1, a2]，显示该多边形的颜色

（3）小区间上存在两个或两个以上的多边, 比如[a6, a7]， 必须通过深度测试判断哪个多边形可见

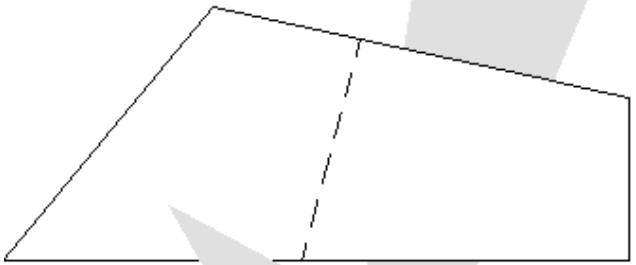
区间扫描线算法



贯穿情形



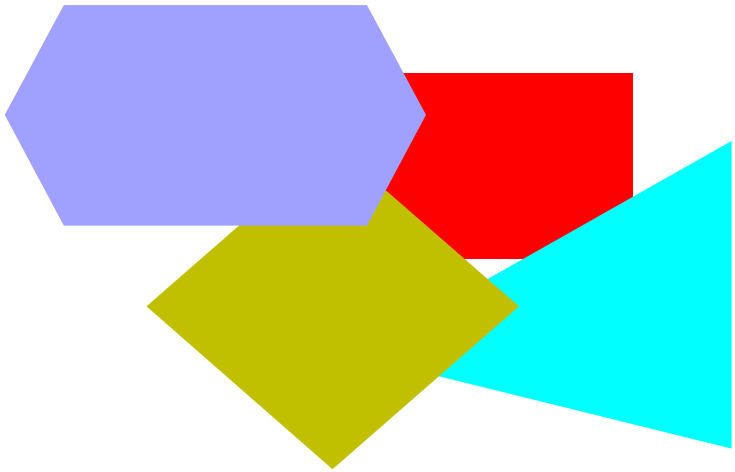
循环遮拦



消隐算法

算法	描述
深度缓存器算法	图像空间
区间扫描线算法	图像空间
深度排序算法	图像空间、景物空间之间
区域细分算法	图像空间、景物空间之间
光线投射算法	图像空间、景物空间之间
BSP算法	景物空间
多边形区域排序算法	景物空间

深度排序算法



和视点相关

多边形之间不能：

- 贯穿
- 循环遮拦

深度排序算法

- 将多边形按深度进行排序：距视点近的优先级高，距视点远的优先级低
- 由优先级低的多边形开始逐个对多边形进行扫描转换

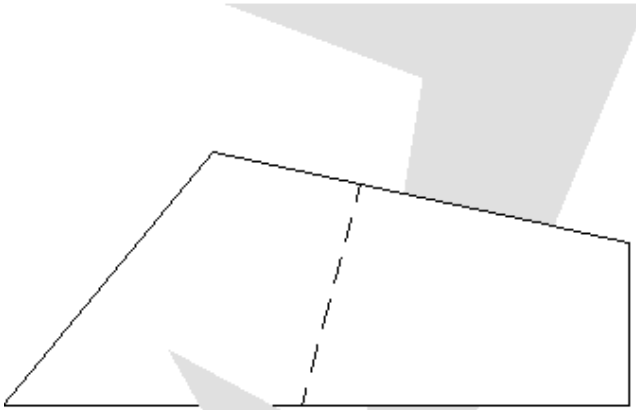
画家算法

深度排序算法

- 介于景物空间消隐算法和图像空间消隐算法之间的算法：
  - 排序：景物空间
  - 消隐：图像空间

- 特点：**适合用于透明、半透明处理！！**

循环遮挡



区域细分算法

考察投影平面上的一块区域：

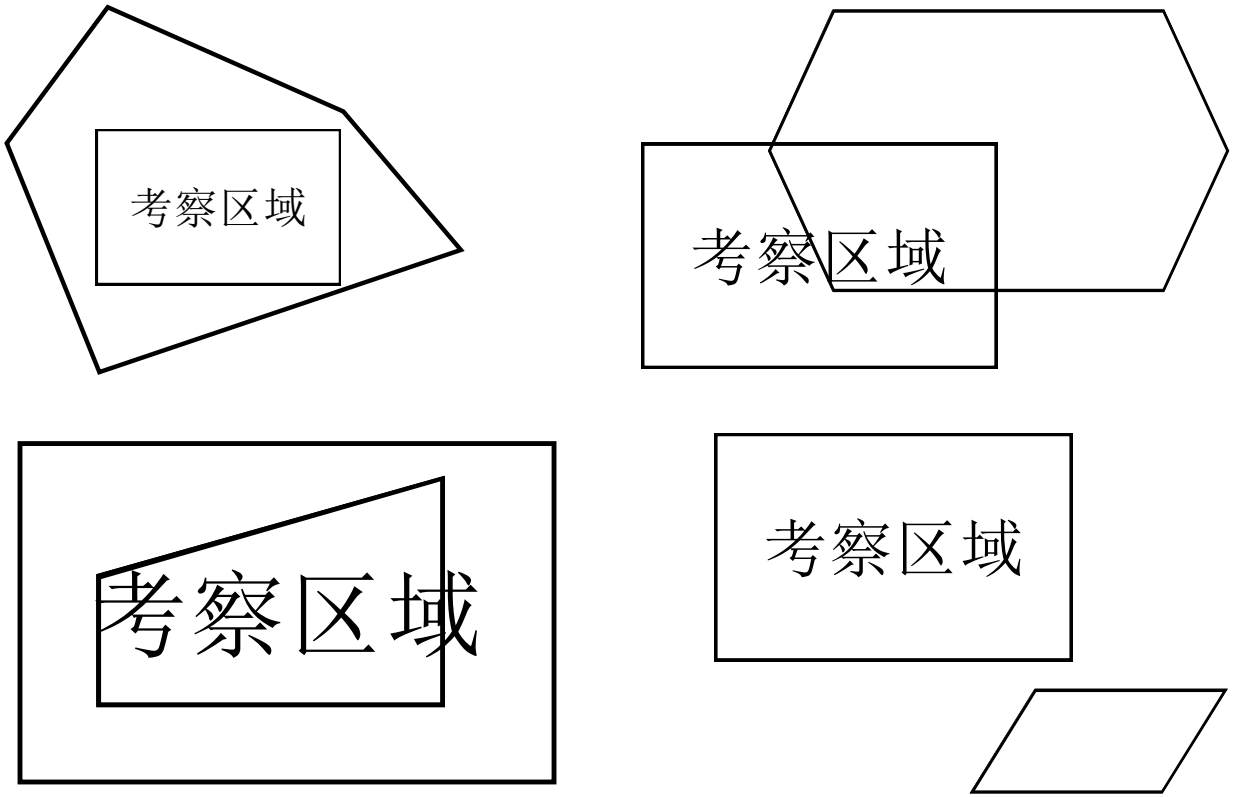
- 如果可以很“容易”地判断覆盖该区域中的哪个或哪些多边形可见，则根据这些多边形处理该区域
- 否则将该区域分成几个更小的区域，然后处理小的区域

基本原理：越小的区域，越“容易”处理！

关键点：

- 如何划分更小的区域
- 如何进行“容易”判断？

多边形与考察区域之间的关系

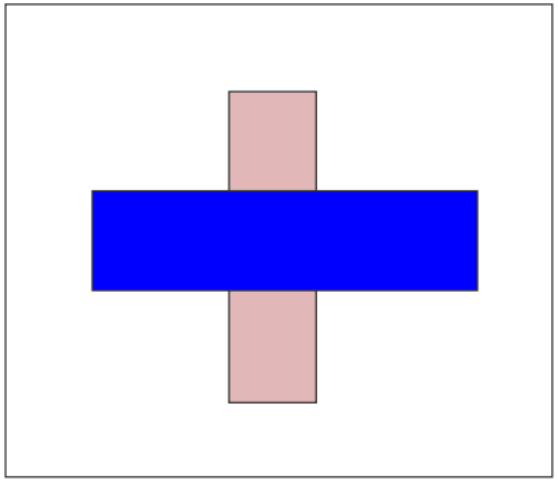


区域细分算法

算法步骤：

（1）如果窗口内没有物体则按背景色显示

（2）若窗口内只有一个面，则把该面显示出来

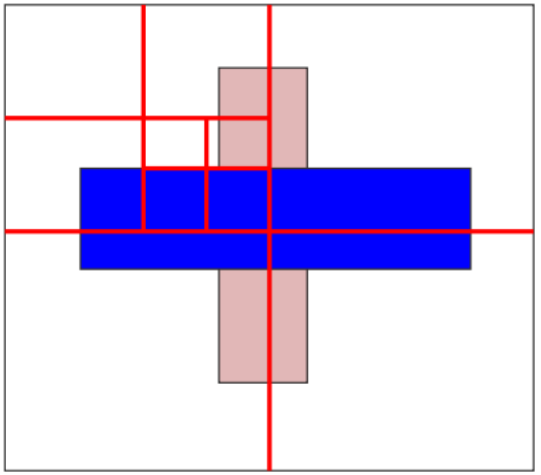


（3）否则，窗口内含有两个以上的面，则把窗口等分成四个子窗口。对每个小窗口再做上述同样的处理。这样反复地进行下去

区域细分算法

（3）窗口内含有两个以上的面，则把窗口等分成四个子窗口。对每个小窗口再做上述同样的处理。这样反复地进行下去

把四个子窗口压在一个堆栈里（后进先出）。



光线投射法

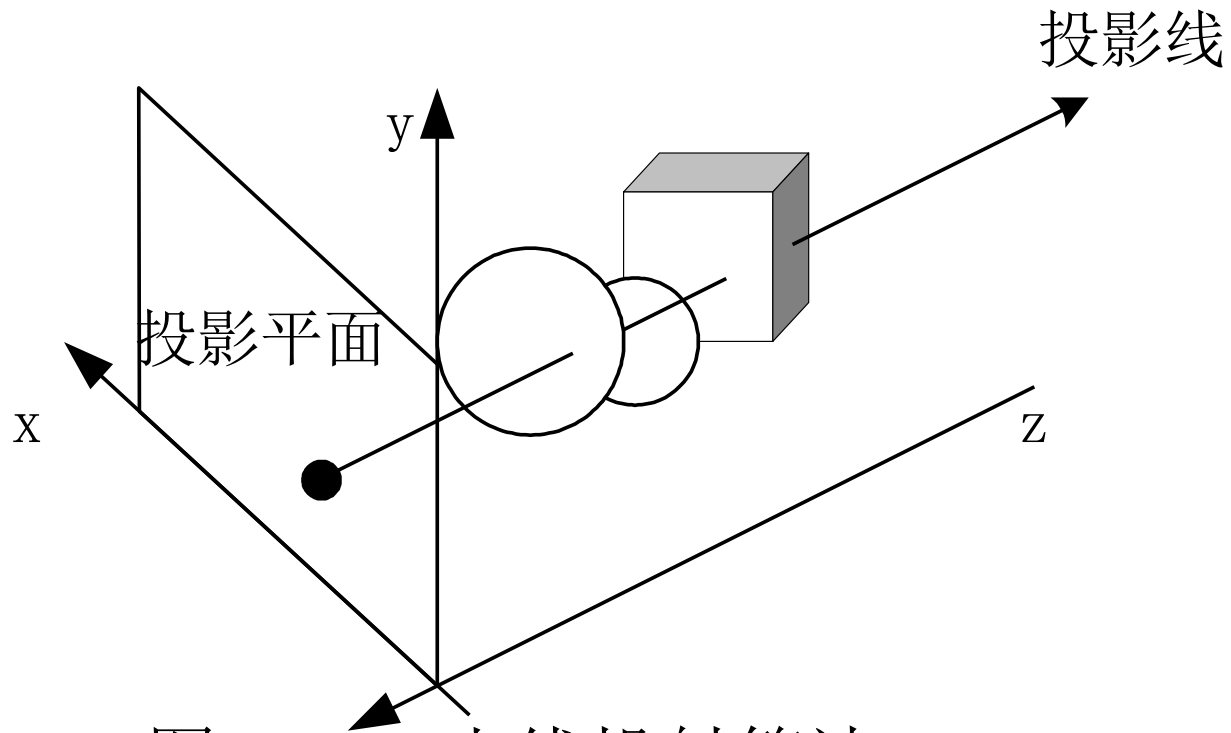


图9-13　光线投射算法

光线投射法

- 通过视点和投影平面（显示屏幕）上的所有象素点作一入射线，形成投影线。
- 将任一投影线与场景中的所有多边形求交。
- 若有交点，则将所有交点按z值的大小进行排序，取出最近交点所属多边形的颜色；若没有交点，则取出背景的颜色。
- 将该射线穿过的象素点置为取出的颜色。

光线投射法

- 关键问题：求交点
  - 利用连贯性
  - 外接矩形
  - 空间分割技术
  - ……
- 对包含曲面的场景计算效率高

消隐算法

算法	描述
深度缓存器算法	图像空间
区间扫描线算法	图像空间
深度排序算法	图像空间、景物空间之间
区域细分算法	图像空间、景物空间之间
光线投射算法	图像空间、景物空间之间
BSP算法	景物空间
多边形区域排序算法	景物空间

BSP树消隐算法

基本原理：

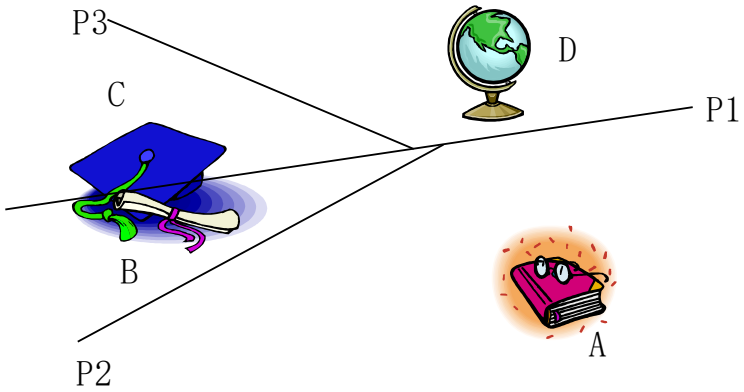
- 与画家算法类似，优先绘制后面的多边形

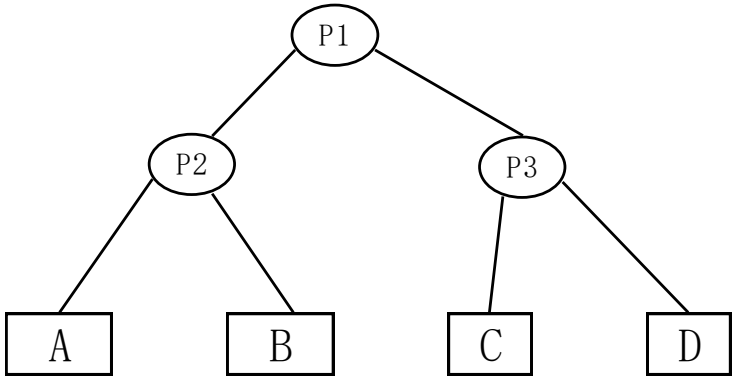
使用BSP树对场景中的对象进行排序

BSP树消隐算法

用递归的方法不断地用平面划分空间

- 这就定义了一个Binary Space Partitioning树
- 需要将被平面穿过的物体进行分割





BSP树消隐算法

- 构建场景的BSP树
- 依据当前视点所在位置，对场景中的每个分割面所生成的两个子空间进行分类
  - 前面（包含视点）的标为Front，后面的标为Back
- 遍历BSP树，优先绘制标为Back的区域；

BSP树消隐算法

- BSP树可以用于按次序绘制场景
  1. 从根平面开始
  2. 找出视点所在的一侧
  3. 将所对侧设为反向
  4. 递归的完成上述过程

- BSP树可以用于任意位置的视点，是一种视点无关的数据结构

BSP树消隐算法

- 适合：
  - 场景不变
  - 视点变化
- 已有硬件支持

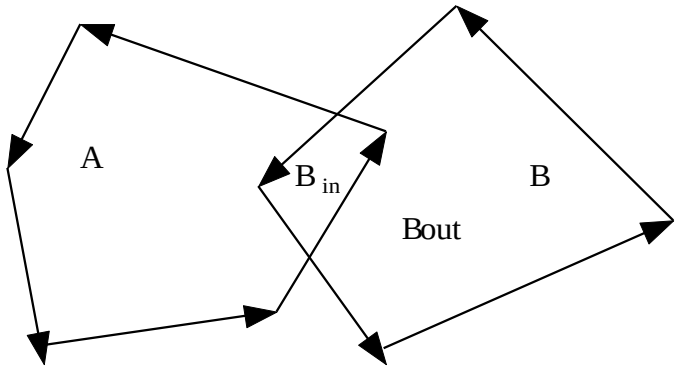
多边形区域排序算法

假设：

- 如果场景中多边形P1、P2…Pn满足：
  Pi完全可见或完全不可见；

如何得到这样的多边形？

- 裁减
- 循环遮拦



多边形区域排序算法

- 将多边形按深度值由小到大排序
- 用前面的可见多边形去切割位于其后的多边形
- 使得最终每一个多边形要么是完全可见的，要么是完全不可见的

和深度排序的区别！！

消隐算法

算法	描述
深度缓存器算法	图像空间
区间扫描线算法	图像空间
深度排序算法	图像空间、景物空间之间
区域细分算法	图像空间、景物空间之间
光线投射算法	图像空间、景物空间之间
BSP算法	景物空间
多边形区域排序算法	景物空间

谢 谢