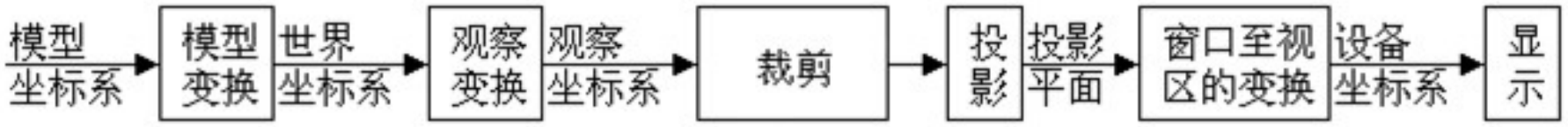
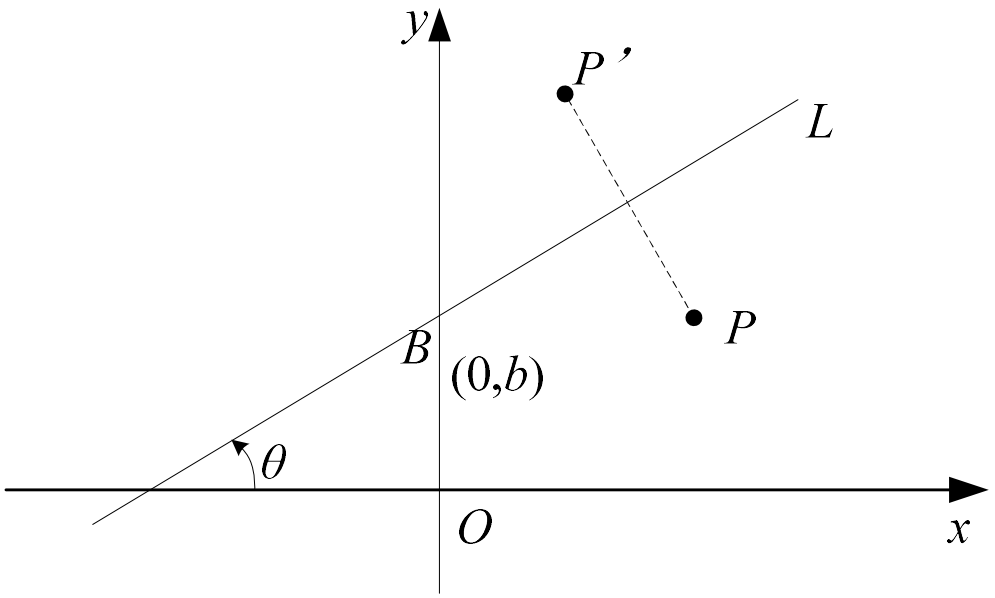
上节要点

- 反走样
  - 提高分辨率
  - 区域采样
  - 加权区域取样
- 图形变换
  - 图形变换的数学基础：向量和矩阵
  - 齐次坐标→统一的变换表示
  - 二维/三维图形的基本几何变换：平移、旋转、比例、错切、对称变换

- 复合变换
- 二维图形的显示流程
  - 世界坐标系和屏幕坐标系
  - 窗口和视口
- 投影变换
  - 平行投影和透视投影
  - 观察坐标系
  - 投影空间和裁剪空间
- 三维图形的显示流程



- 求下图中以直线***L***为对称轴的对称变换矩阵**M**，其中***L***与***y***轴交于**B(0,*b*)**点，与***x***轴夹角为θ。



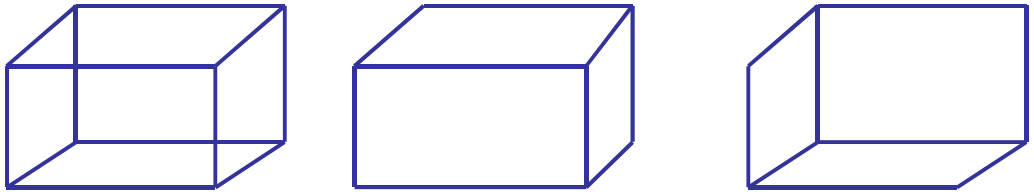
第六章　消隐

清华大学软件学院　张慧　2019-9-25

本章内容

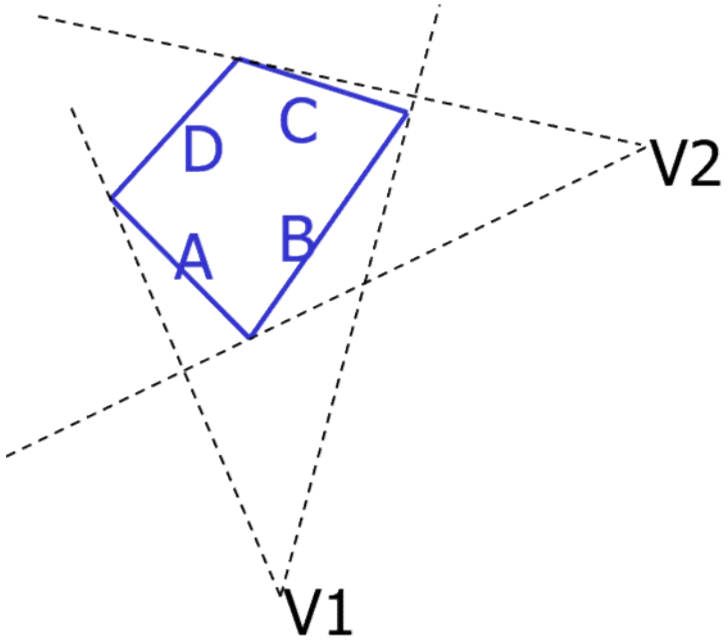
- 准备知识
- 消隐的分类
- 提高消隐算法效率的常用方法
- 消除隐藏线(**Hidden-Line Elimination**)
- 消除隐藏面(**Hidden-Surface Elimination**)

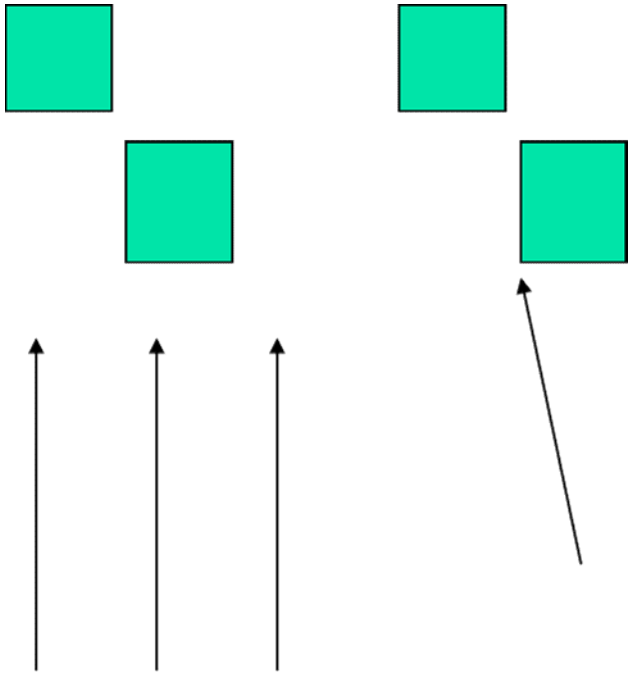
- 消隐：为了真实地反映不同物体之间或物体的不同部分之间由于遮挡而造成的某些物体或部分的不可见现象，须把不可见部分从图中消除，称为消除隐藏线和消除隐藏面，简称消隐，也称为可见面检测。
- 投影变换失去了深度信息，往往导致图形的二义性。



- 消隐的时机

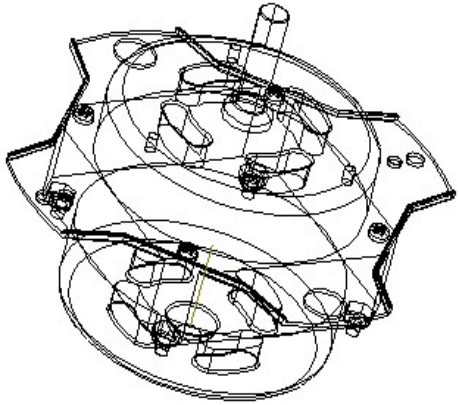
- 消隐结果与被观察物体有关，也与投影方式等有关。



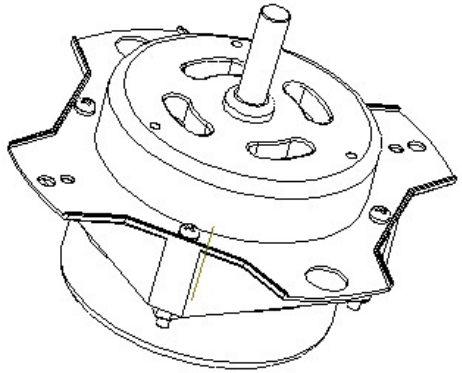


- 取**xoy**平面为投影平面，视线方向取**Z**轴的负方向。

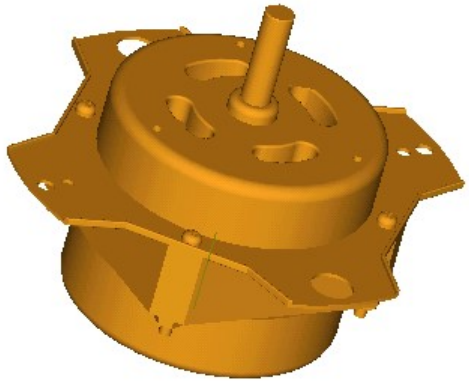
- 消隐的对象是三维物体。



线框图



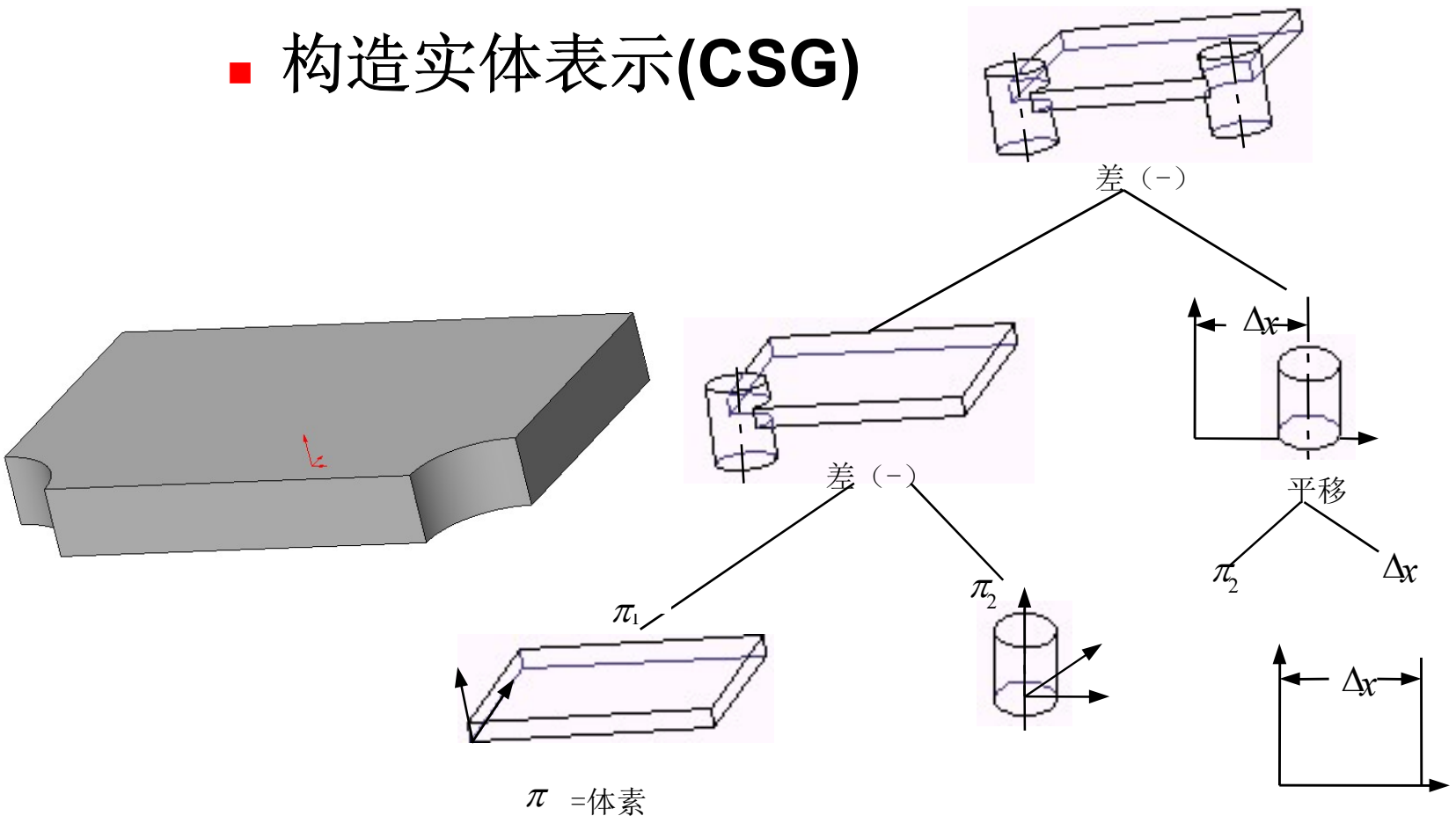
消隐图



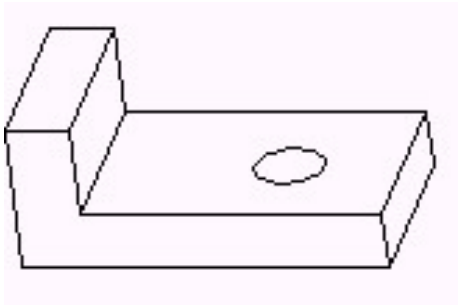
真实感图形

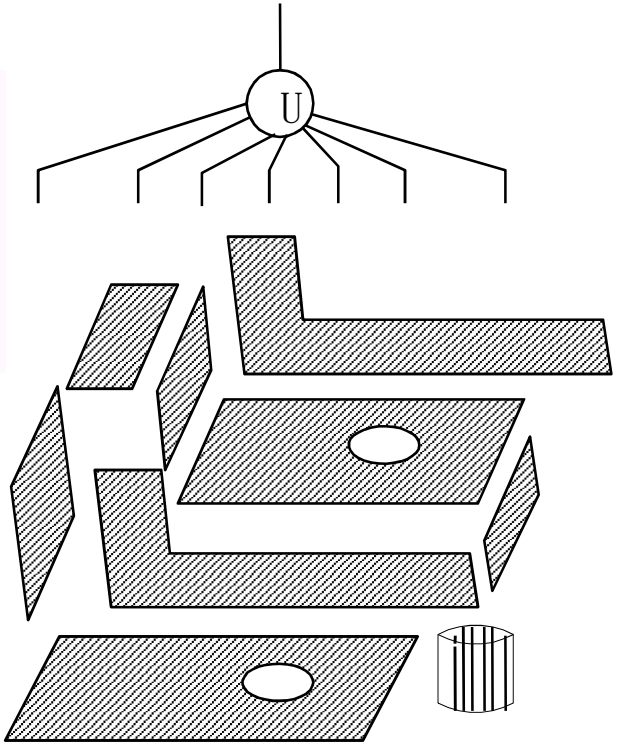
- 实体模型的表示方法主要有边界表示和**CSG**表示等。

- 三维物体的表示
  - 构造实体表示(CSG)



- 边界表示(B-rep)
  - 体、面、环、边、点





拓扑信息

几何信息

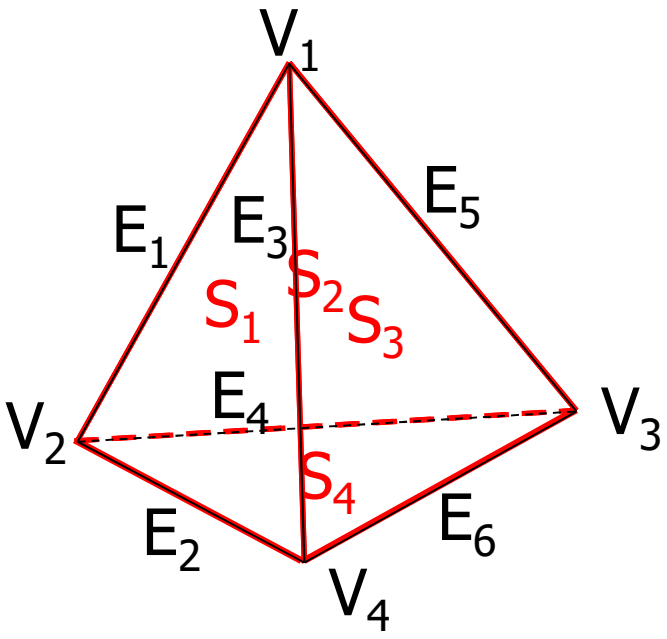
多边形网格模型

- 边界表示的一种，被普遍应用
  - 多面体：用多边形表示精确定义物体的表面特征
  - 其它物体：用多边形网格逼近物体表面
- 信息组织
  - 几何信息：多边形顶点和用来标识多边形表面空间方向的参数
  - 属性信息：物体透明度及表面反射度的参数和纹理特征

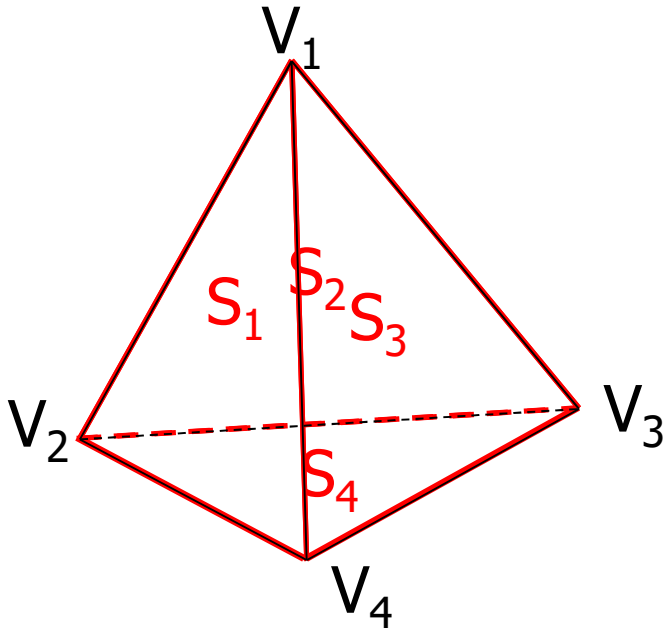
- 数据结构
  - 顶点表
  - 边表
  - 多边形表

顶点表
V_1: x_1,y_1,z_1
V_2: x_2,y_2,z_2
V_3: x_3,y_3,z_3
V_4: x_4,y_4,z_4

边表
E_1: V_1,V_2
E_2: V_2,V_4
E_3: V_1,V_4
E_4: V_2,V_3
E_5: V_1,V_3
E_6: V_3,V_4

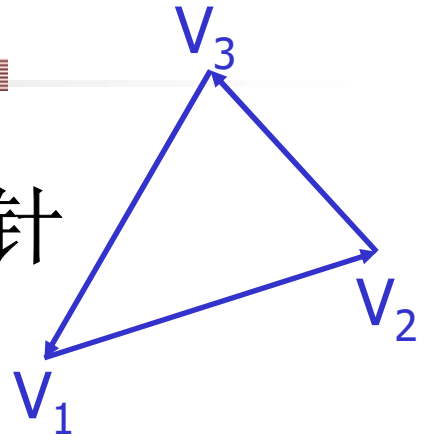


多边形面表
S_1: E_1,E_2,E_3
S_2: E_1,E_5,E_4
S_3: E_3,E_6,E_5
S_4: E_2,E_4,E_6



顶点表
V_1: x_1,y_1,z_1
V_2: x_2,y_2,z_2
V_3: x_3,y_3,z_3
V_4: x_4,y_4,z_4

多边形面表
S_1: V_1,V_2,V_4
S_2: V_1,V_3,V_2
S_3: V_1,V_4,V_3
S_4: V_2,V_3,V_4



- 多边形顶点排列顺序：逆时针
- 多边形所在的平面方程：

$$Ax+By+Cz+D=0$$

(x,y,z)为平面中任意的点

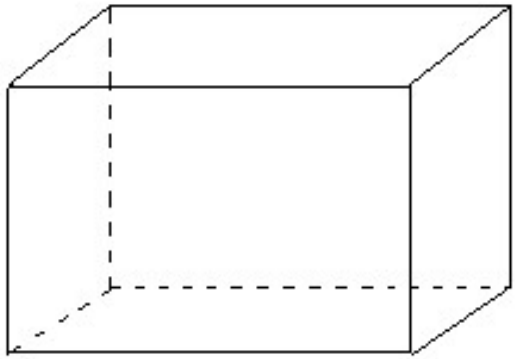
由平面上三个不共线的点可确定A,B,C,D

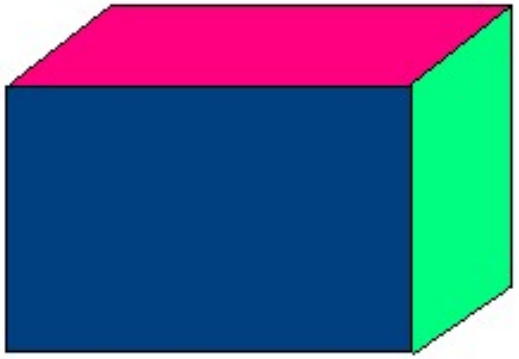
- 平面法向量$N=(V_2-V_1)\times(V_3-V_1)$，可确定平面系数$A=N_x$, $B=N_y$, $C=N_z$，再用平面上任意一点可确定系数D。

- 所有多边形的法向都指向体外；
- 对于空间任意一个不在平面上点(x,y,z)
  - $Ax+By+Cz+D>0$　　点在表面的外部
  - $Ax+By+Cz+D<0$　　点在表面的内部

消隐的分类

- 按消隐对象分类
  - 线消隐：消隐对象是物体上的边，消除的是物体上不可见的边。
  - 面消隐：消隐对象是物体上的面，消除的是物体上不可见的面。





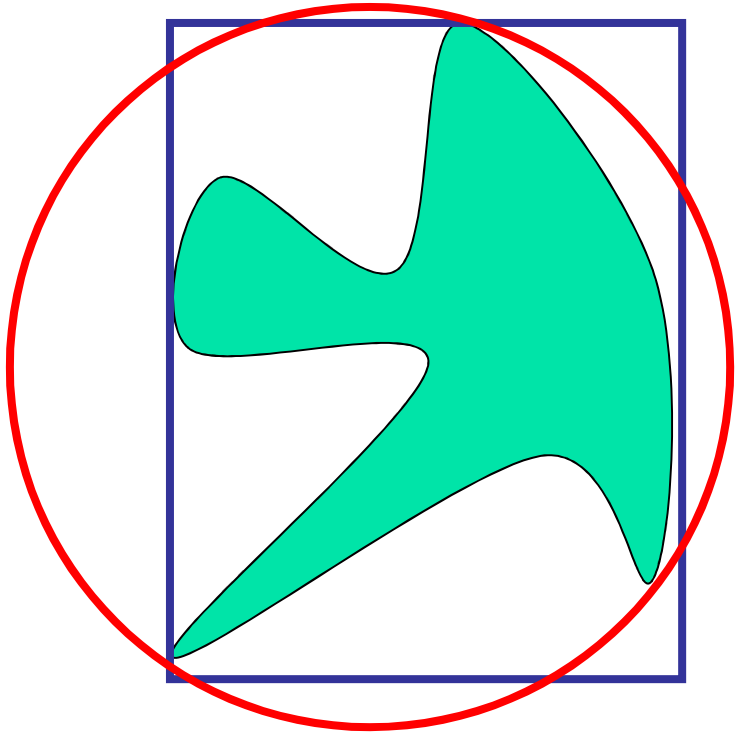
- **Southerland按消隐空间分类**
  - 物体空间的消隐算法 (光线投射、**Roberts)**
    - 将场景中每一个面与其他每个面比较，求出所有点、边、面遮挡关系。
  - 图像空间的消隐算法 **(Z-Buffer**、扫描线、**Warnock)**
    - 对屏幕上每个象素进行判断，决定哪个多边形在该象素可见。
  - 物体空间和图像空间的消隐算法 (画家算法)
    - 在物体空间中预先计算面的可见性优先级，再在图像空间中生成消隐图。

提高消隐算法效率的常用方法

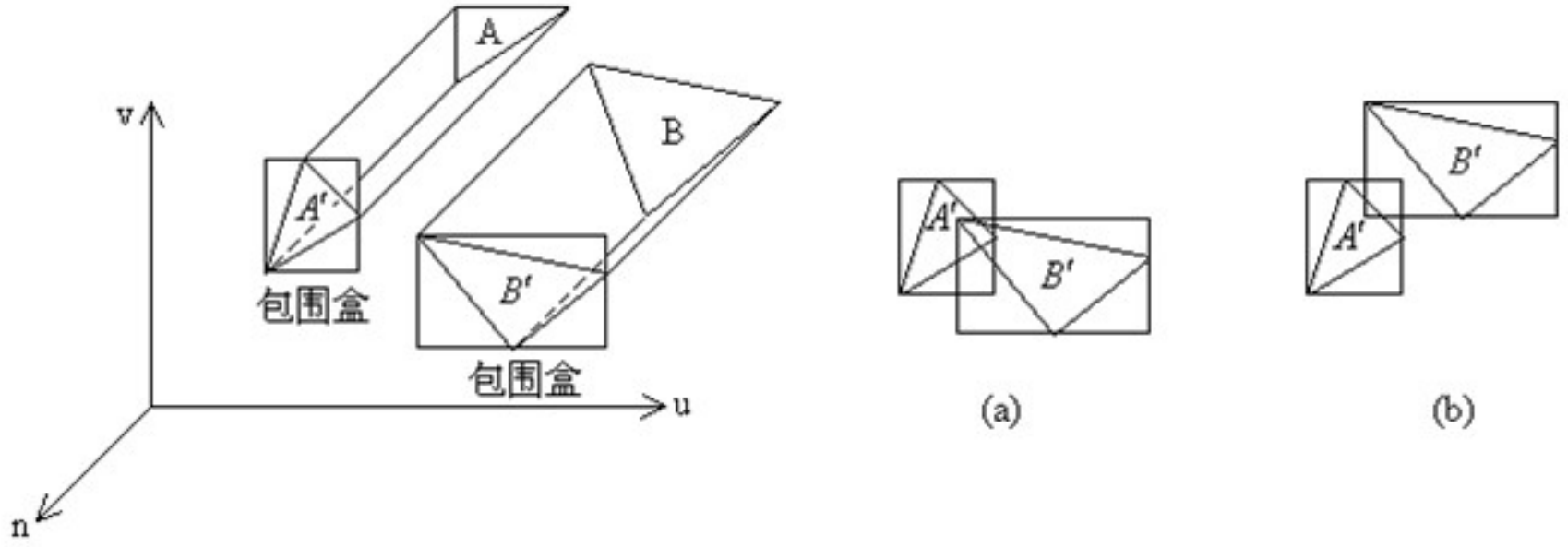
- 排序
  - 为了确定消隐对象之间的遮挡关系，通常在**X，Y，Z**三个方向上都要进行排序。
  - 消隐算法的效率在很大程度上取决于排序的效率。
    - 整体排序：画家算法
    - 点排序：**Z-Buffer**算法、光线投射算法
    - 区间排序：扫描线算法
    - 区域排序：区域子分算法

- 利用连贯性
  - 所观察的物体或视口内的图象局部保持不变或变化缓慢；
  - 物体的连贯性
  - 面的连贯性(深度连贯性)
  - 区域的连贯性
  - 扫描线的连贯性

- 包围盒技术
  - 包围盒——包围目标的简单形体
    - 紧密包围
    - 简单
    - 常用的包围盒
      - **2D**：矩形、圆
      - **3D**：长方体、球

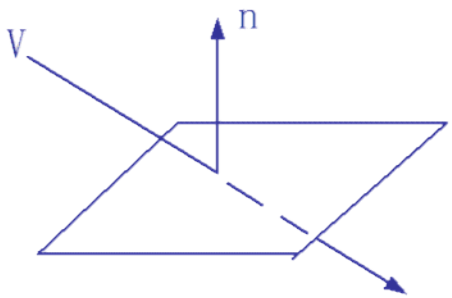


- 应用—避免盲目求交

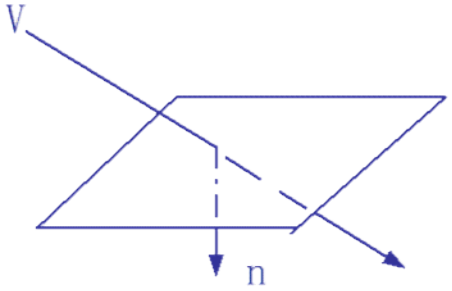


AABB (Axis Aligned Bounding Box)

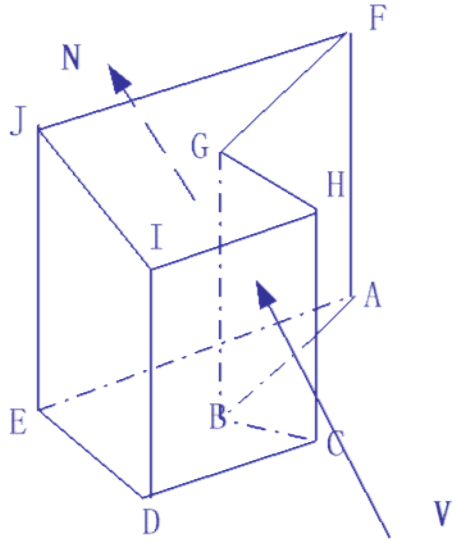
- 后向面剔除
  - 前向面与后向面
    - 后向面：若**V· N>0**；
    - 前向面：若**V· N<0**；
  - 后向面是不可见的，不会由于后向面的遮挡，而使别的棱成为不可见的。因此计算时，可以把这些后向面全部去掉，这并不影响消隐结果。



前向面



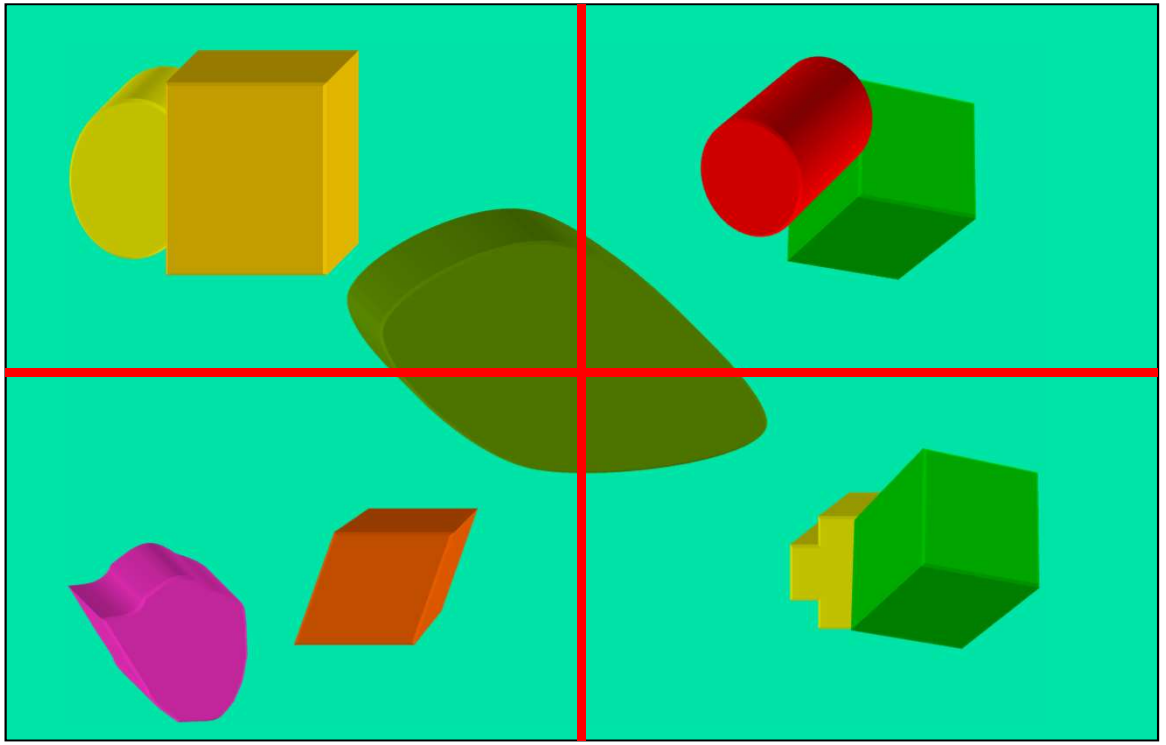
后向面



多面体的隐藏线消除

后向面：**JEAF、HCBG和DEABC**

- 空间分割技术



消除隐藏线

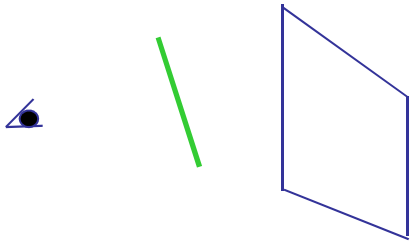
- 在线框显示模型中
- 基本数据结构

  面表(存放参与消隐的面) +

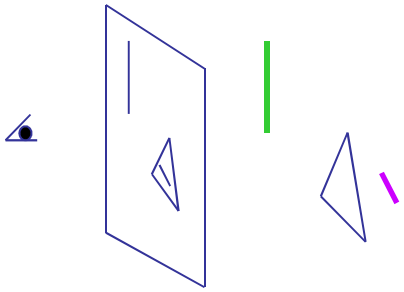
  线表(存放待显示的线)
- 算法：假设**E**为面**F**的一条边，需判别**F**以外每一个面与**E**的遮挡关系。

```
HiddenLineRemove()
{   for(对每一个面Fj 的每一条边Ei) 将二元组< Ei ,j>压入堆栈
    While(栈不空)
    {     < Ei ,j0> = 栈顶;
          for(j != j0 的每一个面Fj)
          {   if( Ei 被Fj 全部遮挡)
              { 将Ei 清空;   break; }
              if( Ei 被Fj 部分遮挡)
              {  从Ei 中将被遮挡的部分裁掉;
                 if(Ei 被分成若干段)
                 { 取其中的一段作为当前段;
                   将其它段及相应的j 压栈;
                 }
              }
          }
          if(Ei 段不为空)          显示Ei ;
    }
}
```

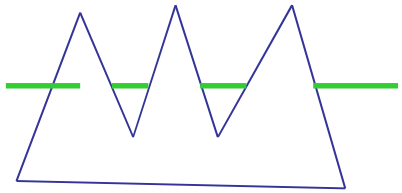
多边形面对直线段的遮挡判断算法



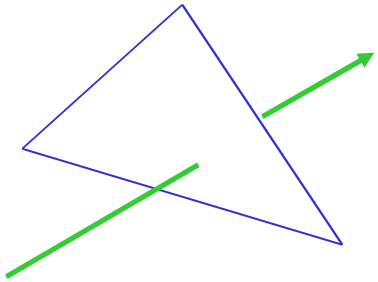
视点与线段同侧



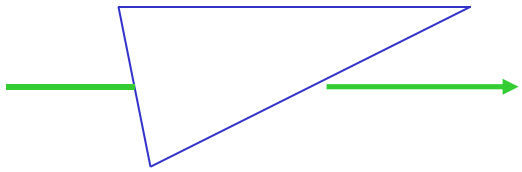
投影无交



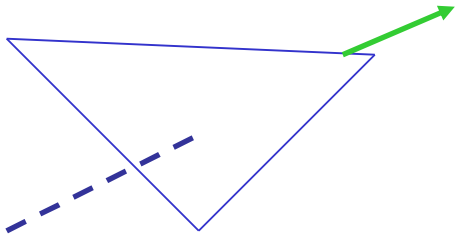
分段交替取值



线面相交



线面平行，线在面后



线面交于线段外

多边形对直线段的遮挡判断算法

1. 若线段的两端点及视点在给定多边形面的同侧，线段不被给定面遮挡，转**7**
2. 若线段的投影与多边形的投影无交，判断线段的投影是否完全在多边形的投影外，是则线段不被遮挡，转**7**
3. 求直线与相应无穷平面的交。若无交点，转**4**。否则，交点在线段内部或外部。若交点在线段内部，交点将线段分成两段，与视点同侧的一段不被遮挡，另一段在视点异侧，转**4**再判；若交点在线段外部，转**4**。

4. 求所剩线段的投影与多边形投影的所有交点，并根据交点在原直线参数方程中的参数值求出Z值(即深度)。若无交点，转5。
5. 以上所求得的各交点将线段的投影分成若干段，求出第一段中点。
6. 若第一段中点在多边形的投影内，则相应的段被遮挡，否则不被遮挡；其他段的遮挡关系可依次交替取值进行判断。
7. 结束。

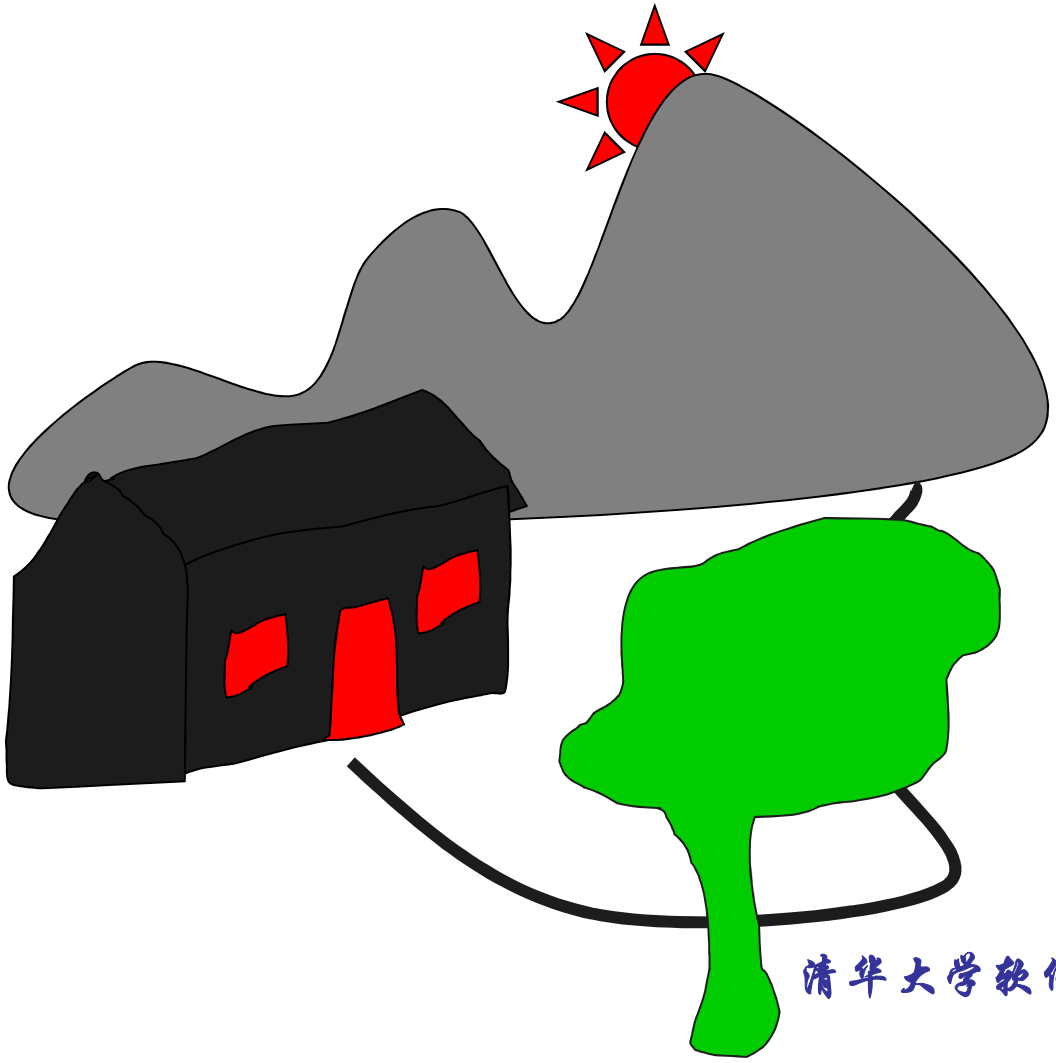
- 判断面对线的遮挡关系涉及到的运算
  - 反复的线线、线面之间的求交运算
  - 线段与多边形的位置关系

- 加速方法
  - 剔除后向面
  - 包围盒测试
  - 空间分割

消除隐藏面

- 画家算法
- **Z缓冲器(Z-Buffer)算法**
- 扫描线**Z-buffer**算法
- 区域子分割算法
- 光线投射算法

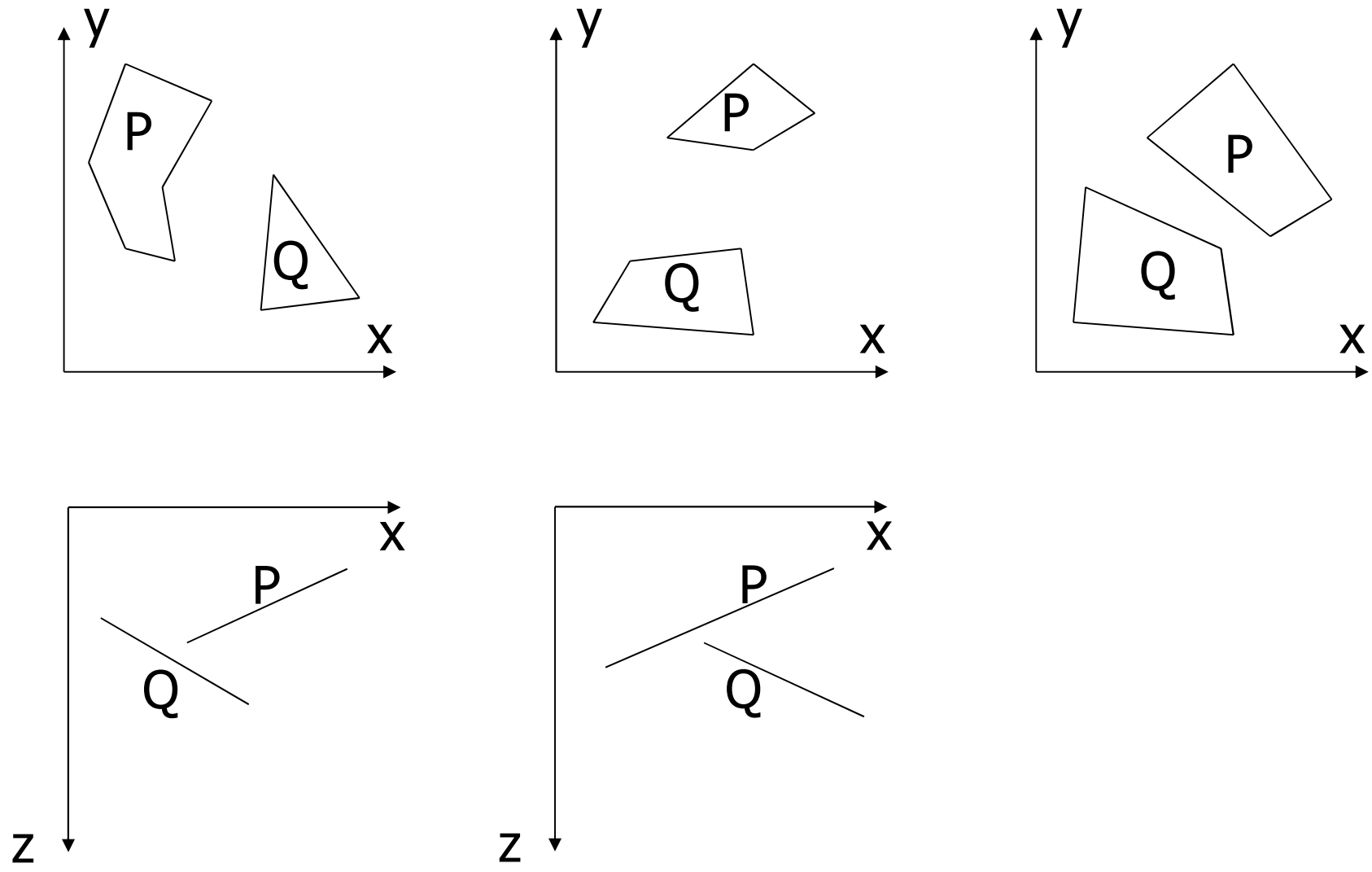
画家算法(列表优先算法)



- 基本思想
  - 先把屏幕置成背景色；
  - 再把物体的各个面按其离视点的远近进行排序，排序结果存在一张深度优先级表中。
  - 然后按照从远到近的顺序逐个绘制各个面。

- 关键：如何对场景中的物体按深度排序

- 对场景中的物体按深度排序
  - 深度重叠测试
    - 根据各多边形顶点**Z**坐标的极小值对多边形进行初步排序；
    - **Zmin(P)<Zmin(Q),若Zmax(P)<Zmin(Q)**，则**P**肯定不能遮挡**Q**。
  - 投影重叠判断
    - **P**和**Q**在**oxy**平面上投影的包围盒在**x**方向上不相交
    - **P**和**Q**在**oxy**平面上投影的包围盒在**y**方向上不相交

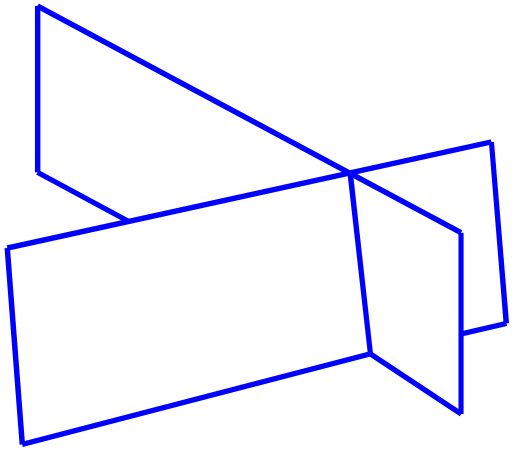


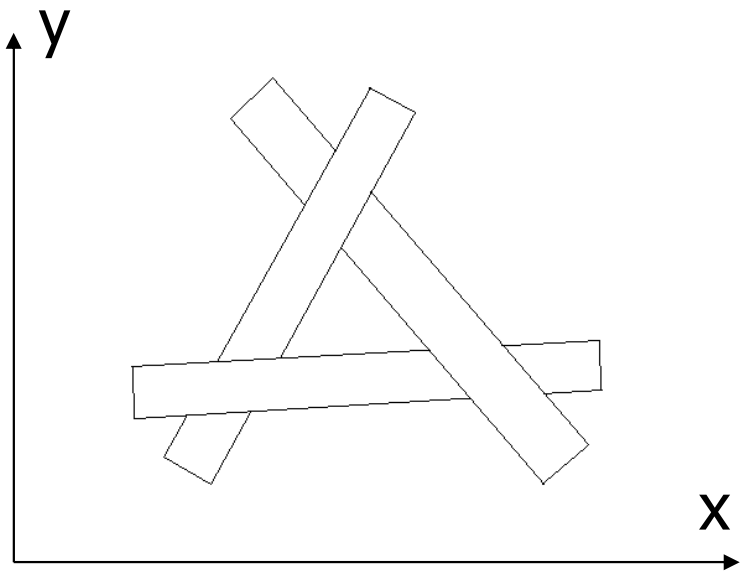
P不遮挡**Q**的各种情况

- P在Q之后：P的各顶点均在Q的远离视点的一侧
- Q在P之前：Q的各顶点均在P的靠近视点的一侧
- P和Q在oxy平面上的投影不相交

- 精确的重叠测试
  - 计算投影的交点，在交点处进行深度比较，若P的深度小，P不遮挡Q；否则，交换P和Q在列表中的位置，用原来的Q作为最低优先级的多边形，与其他多边形进行投影重叠测试；

- 多边形互相穿透或循环重叠
  - 必须在相交处分割多边形，然后进行判断。

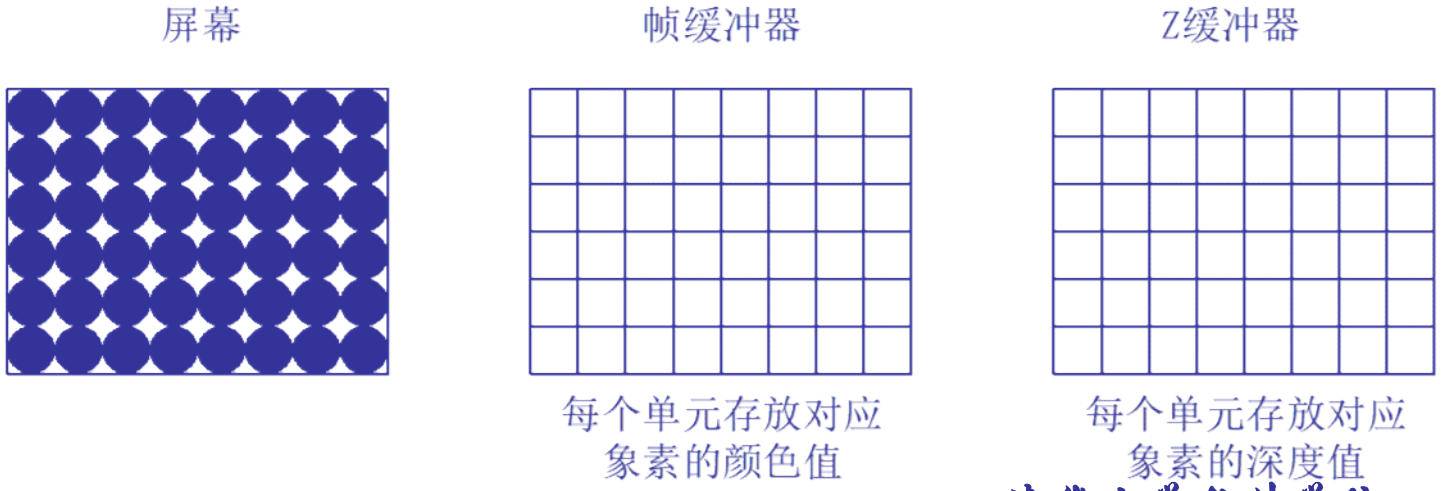




- 优缺点

Z缓冲器(Z-Buffer)算法

- 帧缓存来存放每个象素的颜色值
  - 初值可放对应背景颜色的值
- 深度缓存来存放每个象素的深度值。
  - 初值取成z的极小值。



- 算法过程
  - 在把显示对象的每个面上每一点的属性（颜色或灰度）值填入帧缓冲器相应单元前，要把这点的z坐标值和z缓冲器中相应单元的值进行比较。只有前者大于后者时才改变帧缓冲器的那一单元的值，同时z缓冲器中相应单元的值也要改成这点的z坐标值。
  - 如果这点的z坐标值小于z缓冲器中的值，则说明对应象素已经显示了对象上一个点的属性，该点要比考虑的点更接近观察点。
  - 对显示对象的每个面上的每个点都做了上述处理后，便可得到消除了隐藏面的图。

Z-Buffer算法()

```
{   帧缓存全置为背景色
    深度缓存全置为最小Z值
    for(每一个多边形)
    {
       for(该多边形在投影平面上投影内的每个象素(x,y) )
      {   计算该多边形在该象素的深度值Z(x,y);
          if(Z(x,y)大于Z缓存在(x,y)的值)
          {   把Z(x,y)存入Z缓存中(x,y)处;
              把多边形在(x,y)处的颜色值存入帧缓存的(x,y)处;
          }
       }
    }
}
```

- 优点
  - **Z-Buffer**算法在象素级上以近物取代远物，实现起来远比总体排序灵活简单；
  - 消隐顺序与形体的遮挡关系无关；
  - 有利于硬件实现；
- 缺点
  - 占用空间大；
  - 对每个多边形覆盖的象素都要计算深度值，没有利用图形的相关性与连续性；

- 解决空间占用大的问题
  - 将整个区域分成若干子区域，一区一区显示，Z缓冲器的单元数即为子区域的象素个数；
  - 以屏幕一行为区域单位处理显示——扫描线Z-Buffer算法。
  - 只用一个深度缓存变量Z-Buffer的改进算法。

一个深度缓存变量Z-Buffer改进算法

```
{   帧缓存全置为背景色
    for(屏幕上的每个象素(i,j))
     {     深度缓存变量zb置最小值MinValue
           for(多面体上的每个多边形Pk)
           {
                if(象素点(i,j)在Pk的投影多边形之内)
                     {
                       计算Pk在(i,j)处的深度值depth;
                        if(depth大于zb)
                       {       zb = depth;
                               indexp = k;
                       }
                     }
           }
            if(zb != MinValue)
                     计算多边形Pindexp在象素(i,j) 处的颜色并显示
    }
}
```