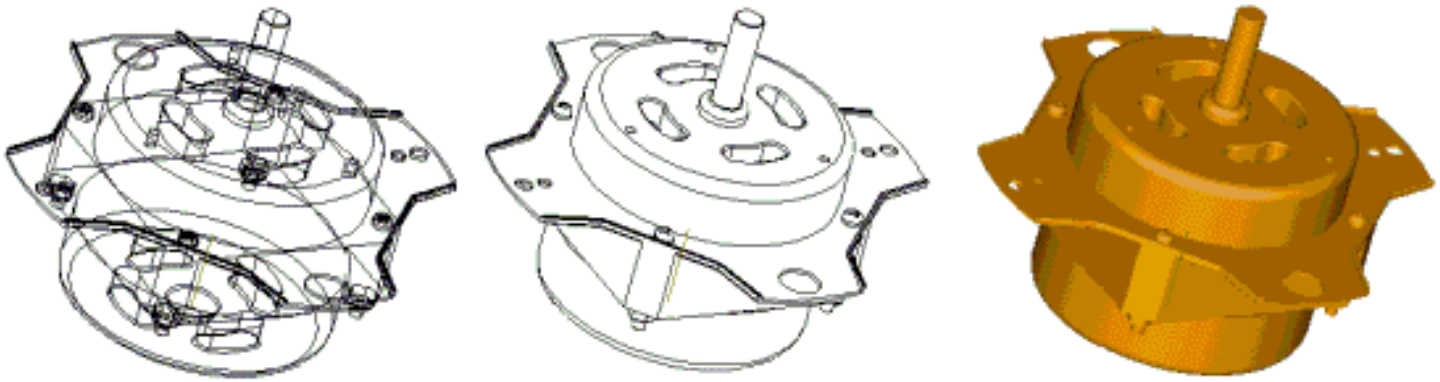
第十一章 消隐

基本概念(消隐)

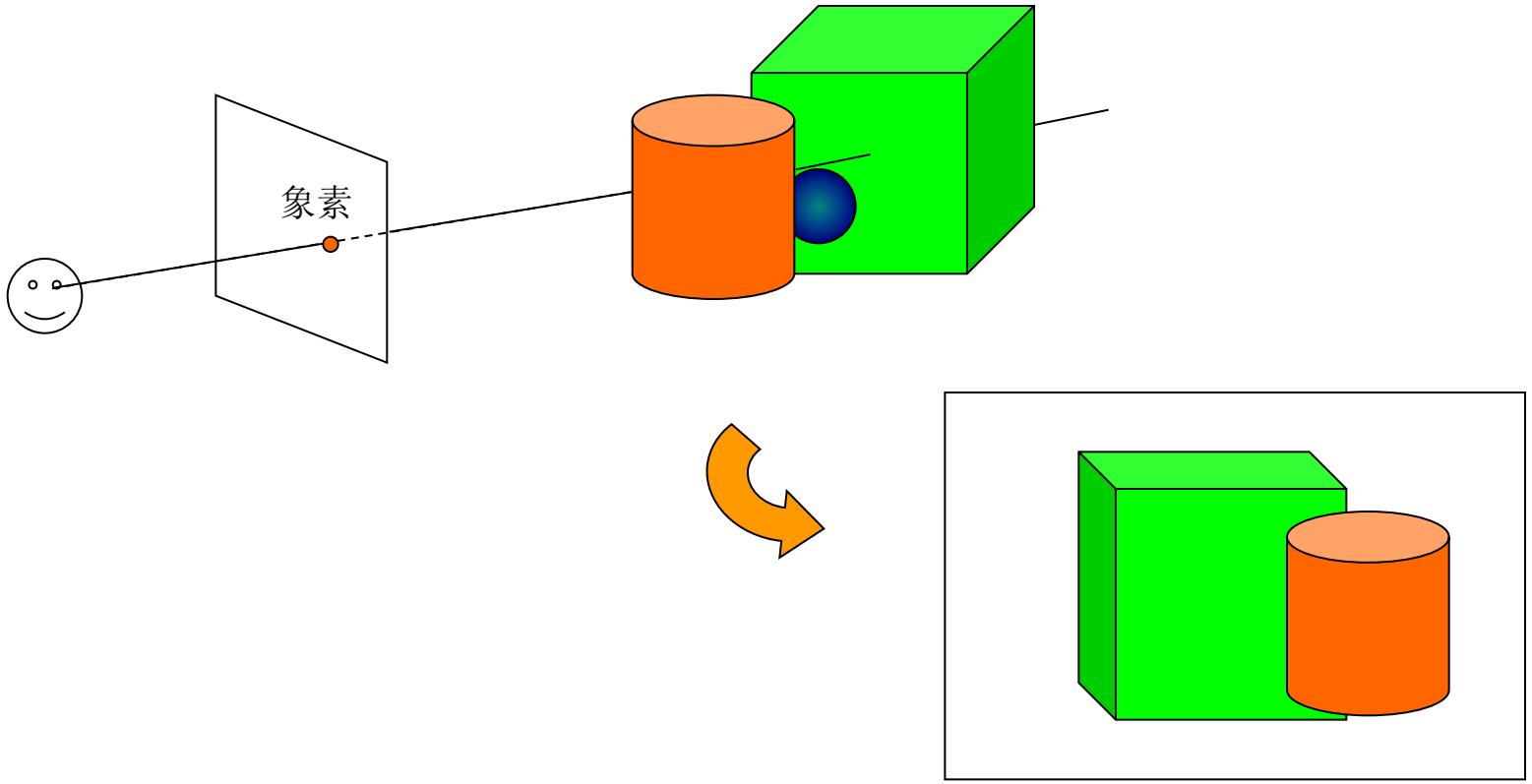
- 在用显示设备描述物体的图形时，必须把三维信息经过某种投影变换，在二维的显示表面上绘制出来。由于投影变换失去了深度信息，往往导致图形的二义性。

基本概念(消隐)

- 要消除二义性，就必须在绘制时消除被遮挡的不可见的线或面，习惯上称作消除隐藏线和隐藏面，或简称为消隐。



消隐的说明



11.1 消隐的分类

- 消隐的对象是三维物体。
- 三维物体的表示主要有边界表示和CSG表示等。最简单的表示方式是用表面上的平面多边形表示。如物体的表面是曲面，则将曲面用多个平面多边形近似。
- 消隐结果与被观察物体有关，也与视点有关。
- 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。

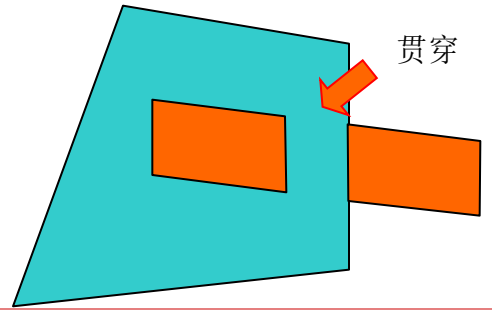
11.2 消除隐藏线(线消隐，补充)

- 对造型的要求
  - 线框模型用边界线表示有界平面，所以待显示的所有实体均为线。但线不可能对线有遮挡关系，只有面或体才有可能对线形成遮挡。
  - 故消隐算法要求造型系统中有面的信息，最好有体的信息。
  - 正则形体的消隐可利用其面的法向量，这样比一般情况快的多。

- 坐标变换
  - 为运算方便，一般通过平移、旋转、透视等各种坐标变换，将视点变换到Z轴的正无穷大处，视线方向变为Z轴的负方向。
  - 变换后，坐标Z值反映了相应点到视点的距离，可以作为判断遮挡的依据。另外，对视锥以外的物体应先行滤掉，以减少不必要的运算。

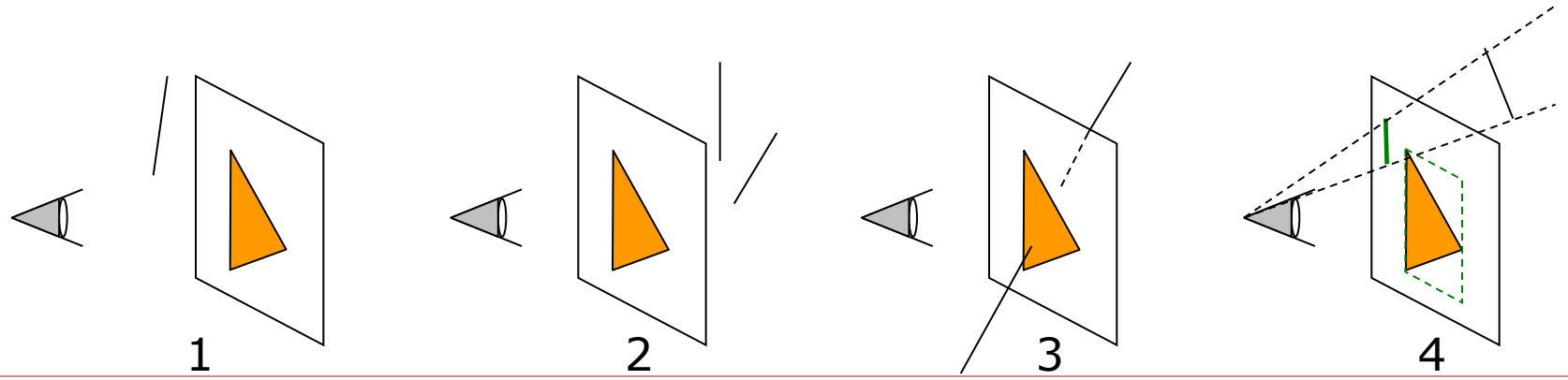
基本运算

- 线消隐中，最基本的运算为：判断面对线的遮挡关系。体也要分解为面，再判断面与线的遮挡关系。
- 在遮挡判断中，要反复地进行线线、线面之间的求交运算。
- 本章主要讨论正则形体的消隐问题。若无特殊说明，均假设面和面之间不能相互贯穿。

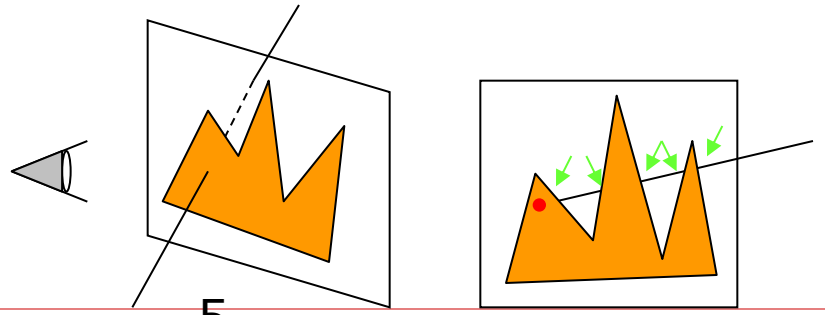


平面对直线段的遮挡判断算法

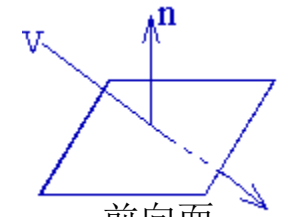
1. 若线段的两端点与视点在给定平面的同一侧，则线段位于给定平面的前面，是可见的，转(9)。
2. 求线段与相应的无限平面的交。若无交点，转(4)；否则，转(3)。
3. 交点将该线段分成两段，与视点同侧的一段是可见的，它没有被遮挡；另一段的可见性还不能确定(多边形可能很复杂)，转(4)，继续测试。
4. 若线段的投影与给定平面的投影的包围盒不相交，则线段可见，它不被该平面遮挡，转(9)。



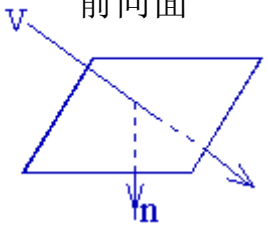
5. (处理复杂多边形情况)求所剩线段的投影与平面边界投影的所有交点。
6. 将线段的投影在交点处依次分成若干段，根据交点在原直线参数方程中的参数值的大小对线段进行排序。
7. 求出第一投影线段的中点，若第一段的中点在多边形的投影内，则相应的线段被遮挡，否则不被遮挡。
8. 其它各段的遮挡关系可依次按“遮挡/可见”交替地取值。例如：如果第一段被遮挡，则后面线段的可见性依次为“可见，遮挡，可见，…”；若第一段可见，则后面的线段依次为“遮挡，可见，遮挡，…”。
9. 结束。



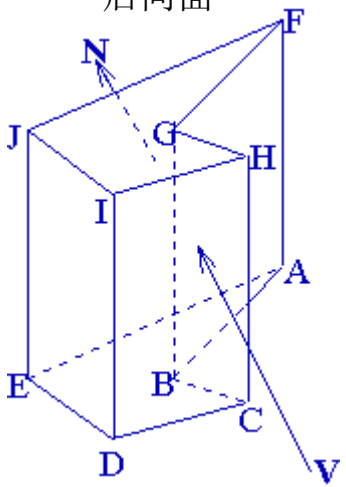
- 如果消隐对象有N条棱，当N很大时，用两两求交的方法工作量是很大的。$O(N^2)$。
- 设V为由视点出发的观察向量，N为某多边形面的法向量。若V·N>0，称该多边形为后向面。若V·N<0，称该多边形为前向面。如下图中的JEAF、HCBG和DEABC所在的面均为后向面。
- 后向面总是看不见的。因此可以把这些后向面全部去掉，这并不影响消隐结果。



前向面



后向面



面消隐算法的分类

- Sutherland根据消隐空间的不同，将面消隐算法分为三类：
  - 物体空间的消隐算法 (光线投射、Roberts算法)
    1. 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。
    2. for (场景中的每一个物体) {

       将其与场景中的其它物体比较，确定其表面的可见部分；

       显示该物体表面的可见部分；

       }

- 图像空间的消隐算法 (Z-buffer、扫描线Z-buffer、区域子分算法)
  1. 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。
  2. for (窗口内的每一个像素) {

     确定距视点最近的物体;

     以该物体表面的颜色来显示像素

     }
- 物体空间和图像空间的消隐算法 (画家算法)
  1. 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。

算法复杂度分析

- 一般来说，对于物体空间方法，一个物体必须和其它所有物体进行比较，才能确定其可见性。如果场景中含有 n 个物体，则比较的计算量为n^2次。
- 对于图像空间方法，每个物体都分解为像素，然后要在各个像素之间进行比较，才能决定哪个物体的像素是可见的。如果每个物体投影后含有m个像素，则比较计算量为m×n次。m虽然很大，但像素之间的比较极其简单，而且可以利用空间连贯性(Coherence)简化计算，因而有时效率反而比较高。
- 目前实用的消隐算法经常将物体空间方法和图像空间方法结合起来使用：首先使用物体空间方法删去物体中一部分肯定不可见的面，然后对其余面再用图像空间方法进行分析。

11.3 消除隐藏面

- 在使用光栅图形显示器绘制物体的真实图形时，必须解决消除隐藏面的问题。
- 这方面已有许多实用算法：
  - 画家算法
  - Z缓冲区(Z-Buffer)算法
  - 扫描线Z-Buffer算法
  - 区间扫描线算法
  - 区域子分割算法 (Warnack算法)
  - 光线投射算法

画家算法

- 画家的作画顺序(先画远景，再画中景，后画近景)暗示出所画物体之间的相互遮挡关系。
- 算法基本思路：
  - 把屏幕置成背景色，
  - 将场景中的物体按其距观察点的远近进行排序，结果放在一张线性表中 (距观察点远的称优先级低，放在表头；距观察点近的称优先级高，放在表尾。该表称为深度优先级表)；
  - 按照从表头到表尾的顺序逐个绘制物体。
- 关键：如何建立深度优先级表?

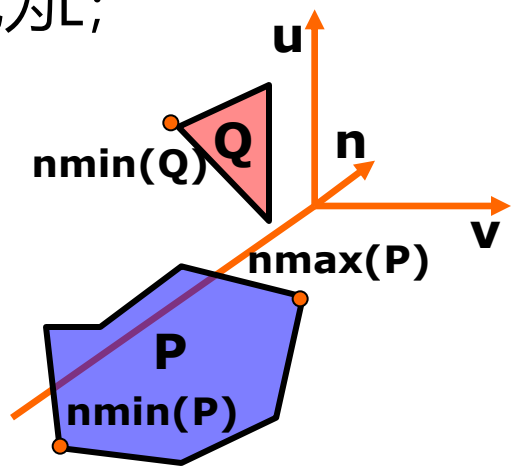
- 一种针对多边形的排序算法如下：
  - 假定在规范化投影坐标系uvn中，投影方向是n轴的负向，因而物体n坐标越大，距观察者越近。
  - 记nmin(P), nmax(P)分别为多边形P的各个顶点n坐标的最小值和最大值，

1. 将场景中所有多边形存入一个线性表(链表或数组)，记为L；
2. 如果L中仅有一个多边形，算法结束；

   否则根据每个多边形的nmin对它们预排序。

   不妨假定多边形P落在表首，即nmin(P)为最小。

   再记Q为L–{P}(表中其余多边形)中任意一个；

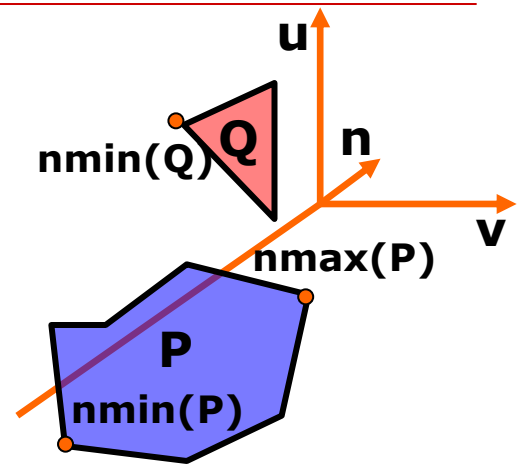


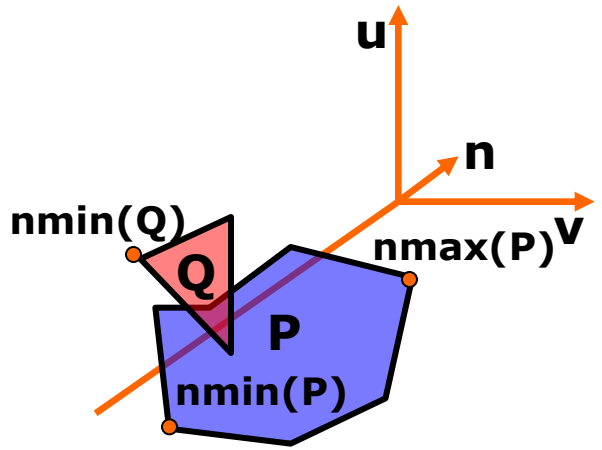
3. 判别P, Q之间的关系，有如下二种：

① 对所有的Q，有nmax(P)<nmin(Q)，则多边形的确距观察点最远，它不可能遮挡别的多边形。

令L=L–{P}(即从L中删去P)，返回step 2;

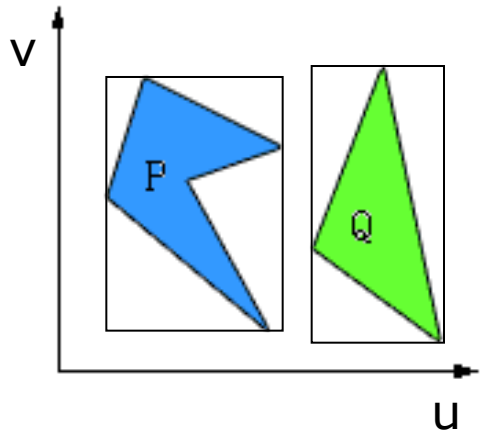
② 存在某一个多边形Q，使nmax(P) > nmin (Q),需进一步判别：

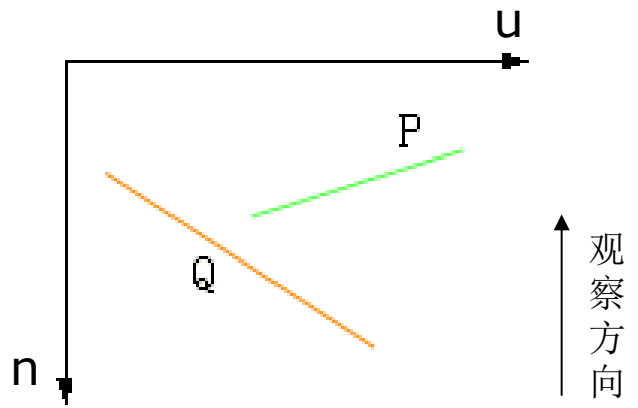




a) 若P,Q在投影平面上的投影P’,Q’的包围盒不相交，则P,Q不相互遮挡，在表中的次序不重要，令L=L–{P}, 返回step 2;否则进行下一步。

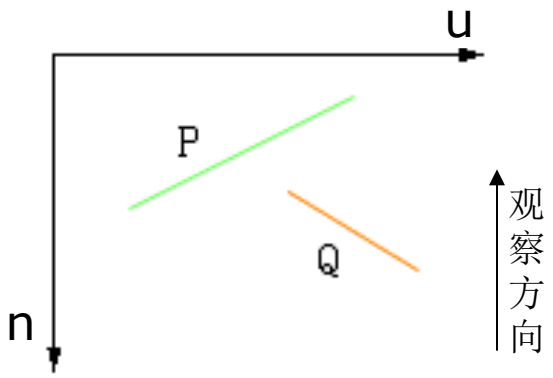
b) 若P的所有顶点位于Q所在平面的不可见的一侧，则P,Q关系正确，令L=L–{P}, 返回step 2;否则进行下一步。

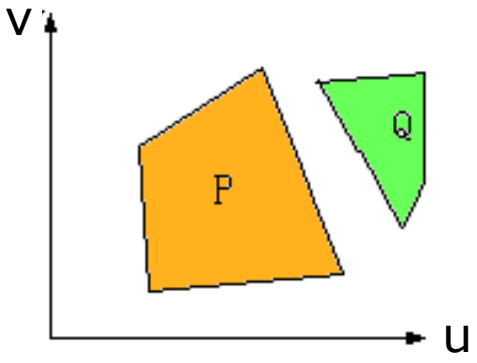


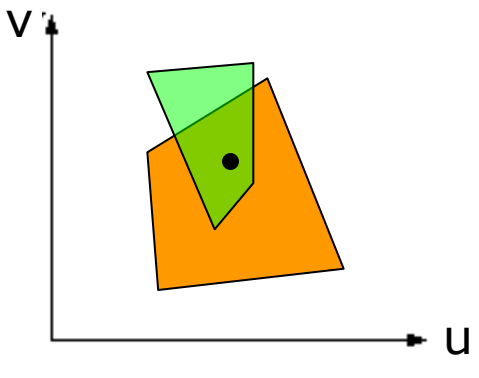


c) 若Q 的所有顶点位于P所在平面的可见的一侧，则P,Q关系正确，令L=L–{P}, 返回step 2;否则进行下一步。

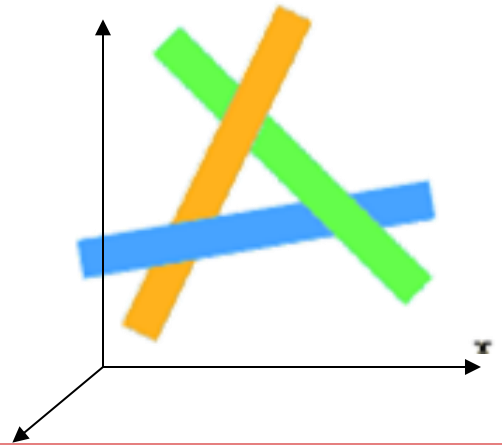
d) 对P,Q投影P’,Q’求交，若P’,Q’不相交，则P,Q在表中的次序不重要，令L=L–{P}, 返回step 2;否则在它们所相交的区域中任取一点，计算P,Q在该点的深度值，如果P的深度小，则P,Q关系正确，令L=L–{P}, 返回step 2;否则交换P,Q,返回step 3.

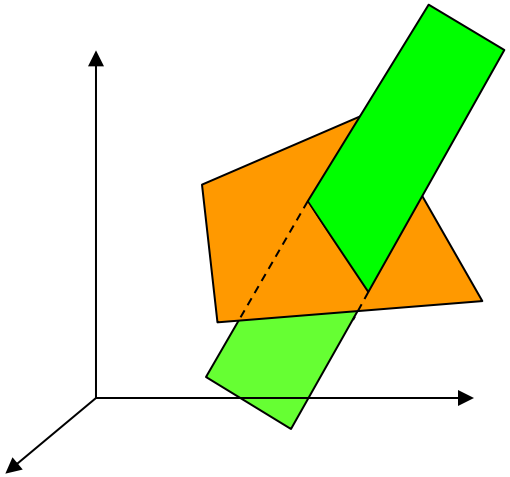






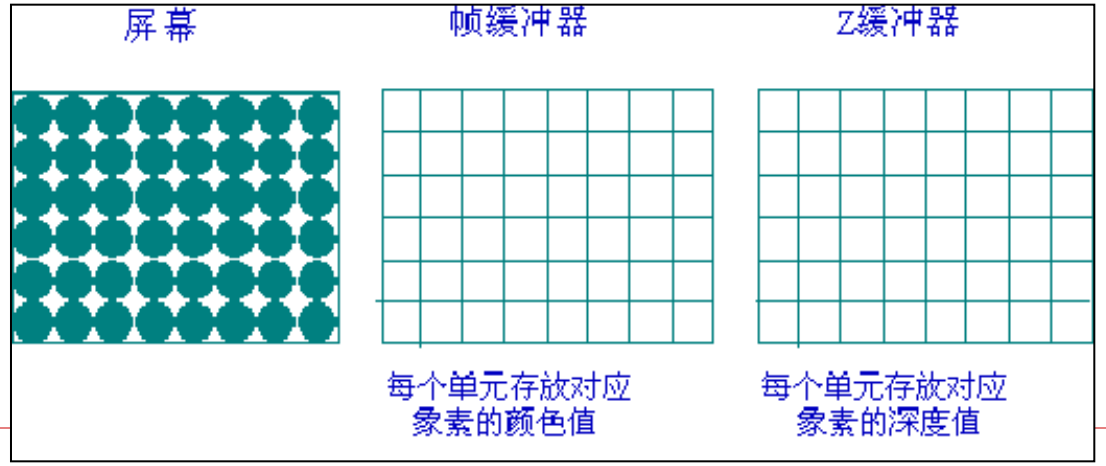
- 画家算法不能处理的情况：
  - 多边形循环遮挡
  - 多边形相互穿透
  - 解决办法：分割成两个





Z缓冲器算法

- 画家算法中，深度排序计算量大，而且排序后，还需再检查相邻的面，以确保在深度优先级表中前者在前，后者在后。若遇到多边形相交或循环重叠的情形，还必须分割多边形。
- Z缓冲区(Z-Buffer)算法避免了这些复杂的运算。在这个算法里，不仅需要有帧缓存来存放每个象素的颜色值，还需要一个深度缓存(Z-Buffer)来存放每个象素的深度值。



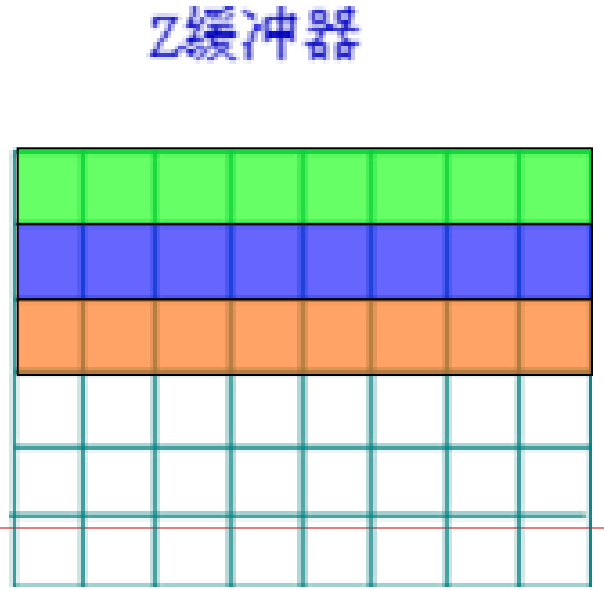
- 算法描述：

```
for(v=0; v<vmax; v++)
for(u=0; u<umax; u++) {
    将帧缓冲器的第(u,v)单元置为背景色;
    将Z缓冲器的第(u,v)单元置为-1 (可见的最小深度值)
    for(每个多边形)
    for(多边形在投影平面上的投影区域内的每个像素(u,v) ) {
        计算多边形在当前像素(u,v)处的深度值d;
        if (d > Z缓冲器的第(u,v)单元的值) {
            置帧缓冲器的第(u,v)单元值为当前多边形颜色;
            置Z缓冲器的第(u,v)单元值为d;
        }
    }
}
```

- 优点：简单稳定，利于硬件实现
- 缺点：
  - 需要一个额外的Z缓冲器
  - 在每个多边形占据的每个像素处都要计算深度值，计算量大

扫描线Z缓冲器算法

- Z缓冲器算法中所需要的Z缓冲器容量较大，为克服这个缺点：
  - 将整个绘图区域分割成若干个小区域，然后一个区域一个区域地显示，这样Z缓冲器的单元数只要等于一个区域内像素的个数就可以了。
  - 如果将小区域取成屏幕上的扫描线，就得到扫描线Z缓冲器算法。



算法描述

```
for ( v= 0;v<vmax;v++)  {
        for (u= 0; u<umax; u++)    { //对绘图窗的每一条扫描线初始化
            将帧缓冲器的第(u,v)单元置为背景色;
          将Z缓冲器的第u单元置为-1(可见的最小深度值 )
      }
     for (每个多边形) { //求出多边形在投影平面上的投影与当前扫描线的相交区间
     //实际上是判断一个给定的点是在一个多边形内、在多边形外，还是在多边形边界上。可采用射线法或弧长法
         for  (该区间内的每个像素(u,v) )  {
             计算多边形在该像素处的深度值d;
             if (d > Z缓冲器的第u单元的值)  {
                 置帧缓冲器的第(u,v)单元值为当前多边形颜色;
                 置Z缓冲器的第u单元值为d;
             }
           }
     }
}
```

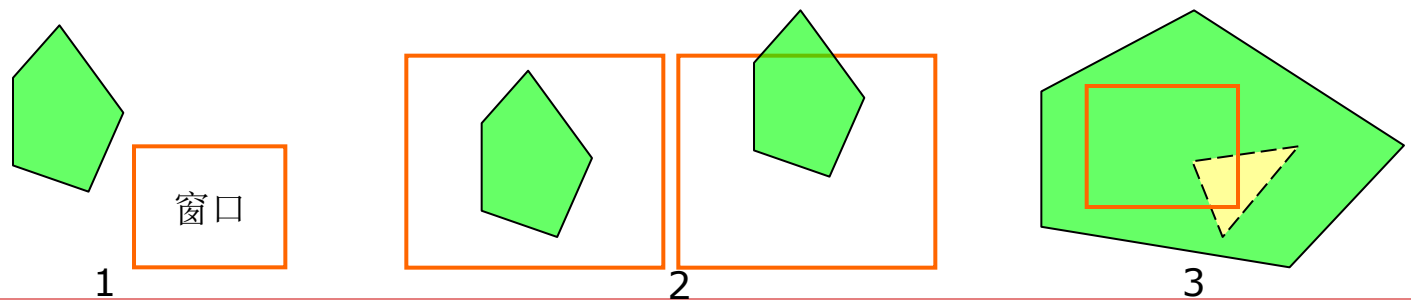
区域子分算法

- Z缓冲器算法与扫描线Z缓冲器算法中都是将像素孤立来考虑，未利用相邻像素之间存在的属性的连贯性，即区域的连贯性，所以算法效率不高。
- 实际上，可见多边形至少覆盖了绘图窗内的一块区域。如果能将这类区域找出来，则避免了在每个像素处计算深度值，消隐问题也就解决了。

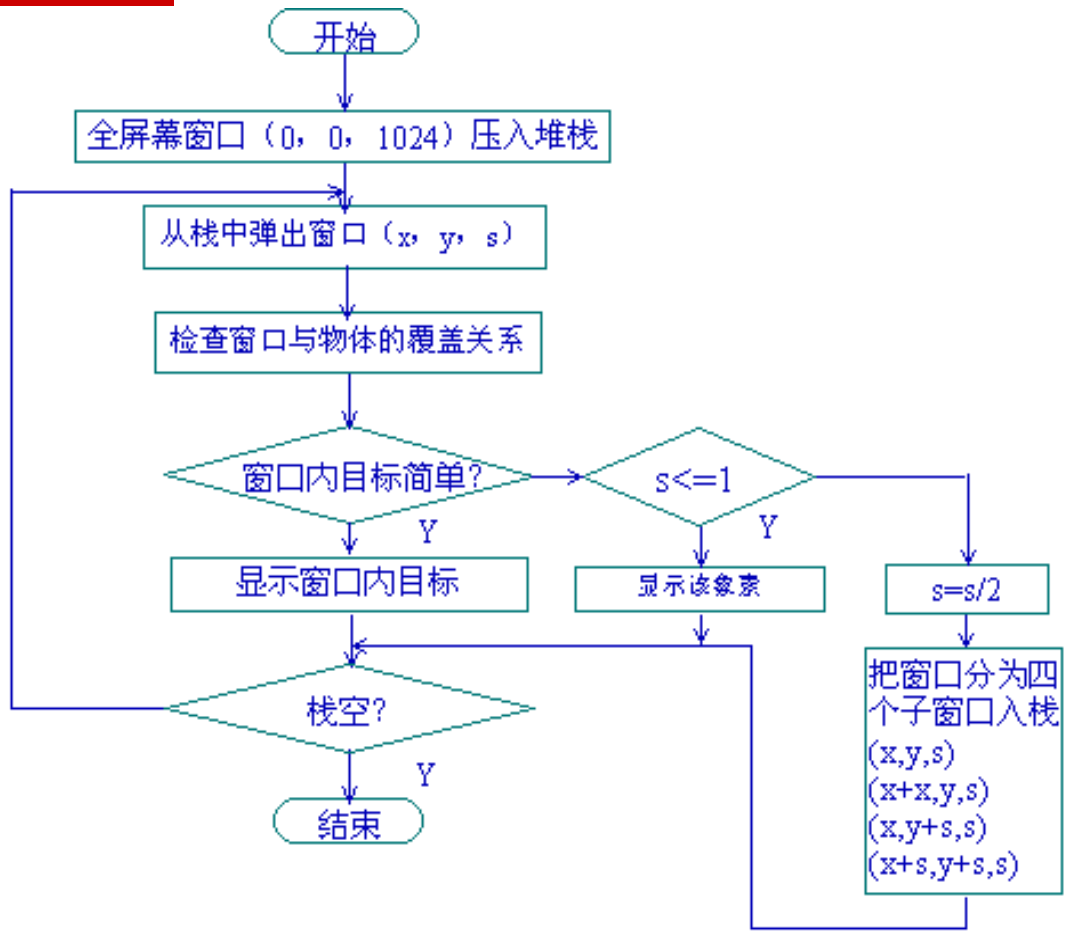
- 算法基本思路：
  - 首先将场景中的多边形投影到绘图窗口内(假设它为边长为k的正方形)，判断窗口是否足够简单，若是则算法结束；
  - 否则将窗口分为四块，对此四个小窗口重复上述过程，直到窗口仅为一个像素大小。此时可能有多个多边形覆盖了该像素，计算它们的深度值，以最近的颜色显示该像素即可。

□ 窗口足够简单的定义(存在下列情况之一)

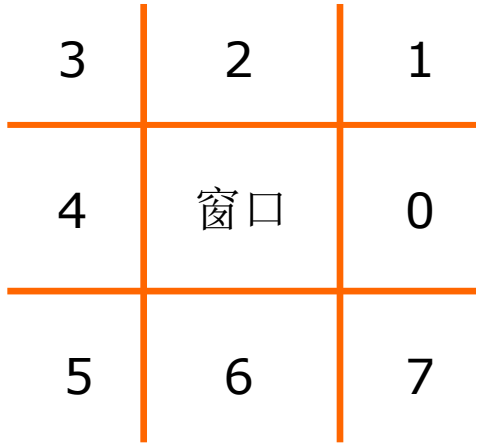
1. 窗口为空，即多边形与窗口的关系是分离的；
2. 窗口内仅含一个多边形，即有一个多边形与窗口的关系是包含或相交。此时先对多边形投影进行裁剪，再对裁剪结果进行填充；
3. 有一个多边形的投影包围了窗口，并且它是最靠近观察点的，以该多边形颜色填充窗口。



- 右图为使用栈结构实现的区域子分割算法流图。
  - 假设全屏幕窗口分辨率为1024×1024。
  - 窗口以左下角点(x，y)和边宽s定义。



- 如何判别多边形与窗口的分离与包围关系?
  1. 区域编码
     - 如右
  2. 多边形顶点编码
     - 多边形顶点的投影落在哪个区域，区域编码就是顶点编码

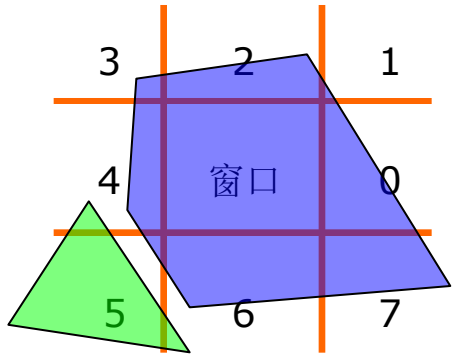


3. 多边形边的编码

- 边v_iv_{i+1}的编码为$\Delta_i=I_{i+1}-I_i$
- $\Delta_i>4$时$\Delta_i=\Delta_i-8$； $\Delta_i<-4$时$\Delta_i=\Delta_i+8$
- $\Delta_i=\pm4$时，按该边与窗口边的延长线将其分为两段，对两段分别编码，再令Δ_i为二者之和。

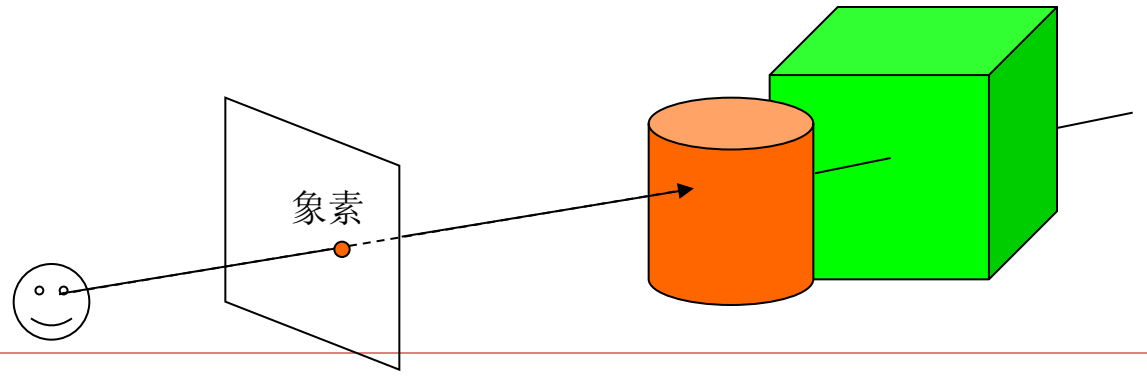
4. 多边形的编码

- 多边形编码为其边的编码之和 $\sum_{i=0}^{n}\Delta_i$。
- $\sum_{i=0}^{n}\Delta_i=0$ 窗口与多边形分离。
- $\sum_{i=0}^{n}\Delta_i=\pm8$ 多边形包围窗口。



光线投射算法

- 算法思路：
  - 考察由视点出发穿过观察屏幕的一象素而射入场景的一条射线，则可确定出场景中与该射线相交的物体。
    - 在计算出光线与物体表面的交点之后，离象素最近的交点的所在面片的颜色为该象素的颜色；
    - 如果没有交点，说明没有多边形的投影覆盖此像素，用背景色显示即可。



算法描述：

```
for (v= 0; v<vmax; v++)
for (u= 0; u<umax; u++)  {
    形成通过像素(u,v)的投影线;
    for (场景中每一个多边形)  {
        将投影线与多边形求交;
        if  (有交点)
            以最近交点所属多边形的颜色显示像素(u,v);
        else  以背景色显示像素(u,v);
    }
}
```

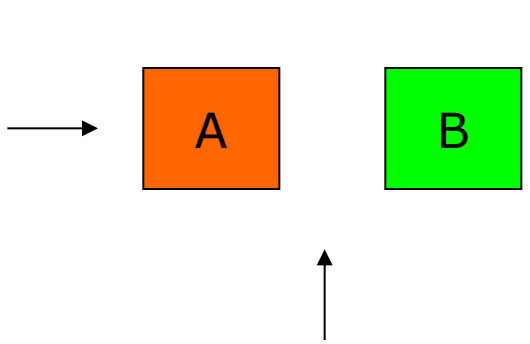
- 光线投射算法与Z缓冲器算法相比，它们仅仅是内外循环颠倒了一下顺序，所以它们的算法复杂度类似。
- 区别在于光线投射算法不需要Z缓冲器。
- 为了提高本算法的效率可以使用包围盒技术，空间分割技术以及物体的层次表示方法等来加速。

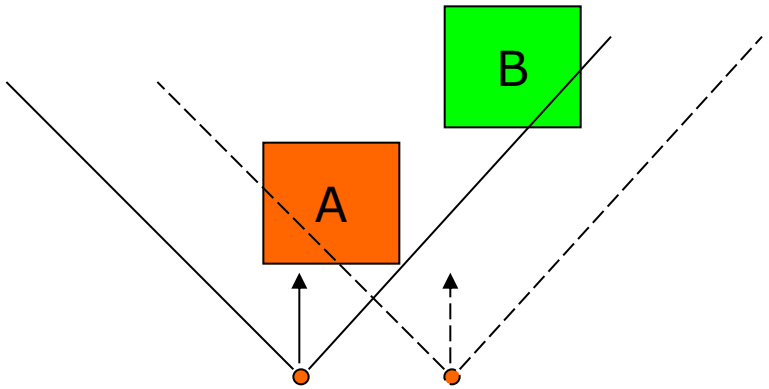
11.4 提高消隐算法效率的常见方法

- 消隐是一个非常费时的工作，人们研究出了各种计算提高算法效率。
  - 利用连贯性
  - 将透视投影转换成平行投影
  - 包围盒技术
  - 背面剔除
  - 空间分割技术
  - 物体分层表示

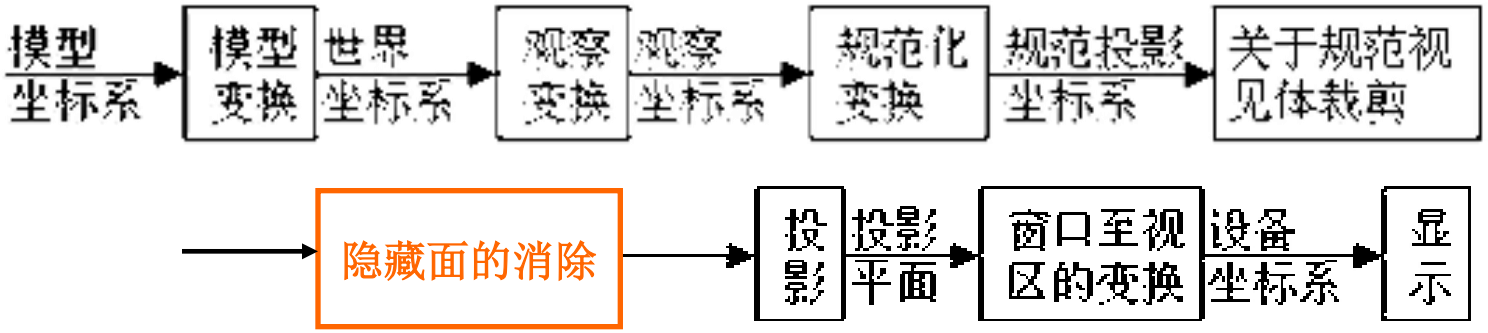
- 利用连贯性(相邻对象的属性值相似)
  - 物体连贯性
  - 面的连贯性
  - 区域连贯性
  - 扫描线的连贯性
  - 深度连贯性

- 将透视投影转换成平行投影
  - 消隐与透视关系密切，体现有：
    - 消隐必须在投影之前完成；
    - 物体之间的遮挡关系与投影中心(视点)的选取有关；
    - 物体之间的遮挡关系与投影方式有关





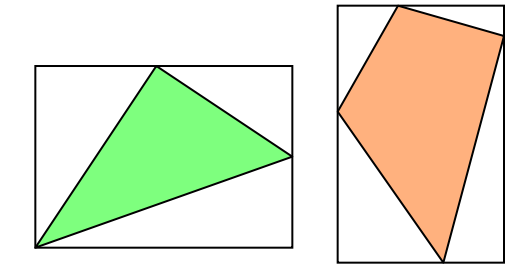
- 根据三维图形显示流程图，消隐工作应该在规范化变换后的规范投影坐标系中进行。如下图。



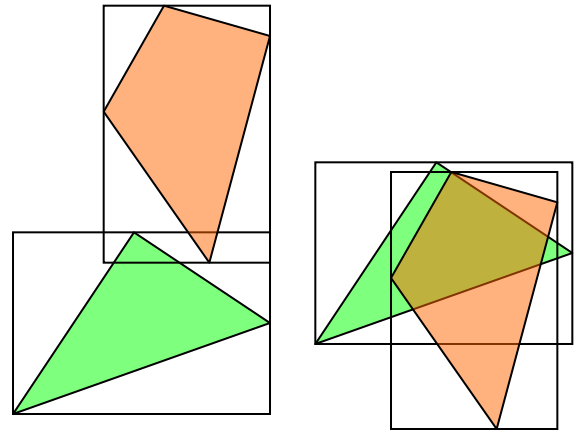
- 在规范投影坐标系中，平行投影和透视投影的投影方向都平行于n轴。但是平行投影时，遮挡关系容易求。
- 平行投影和透视投影可以互相转化，因此可以考虑把透视投影转化为平行投影。此时的三维物体显示流程图：

包围盒技术

- 一个形体的包围盒指的是包围它的简单形体。
- 该技术常用于避免盲目的求交测试；各种物体间的比较等。
- 一个好的包围盒要具有两个条件：
  - 包围盒充分紧密包围着形体；
  - 对其的测试比较简单。



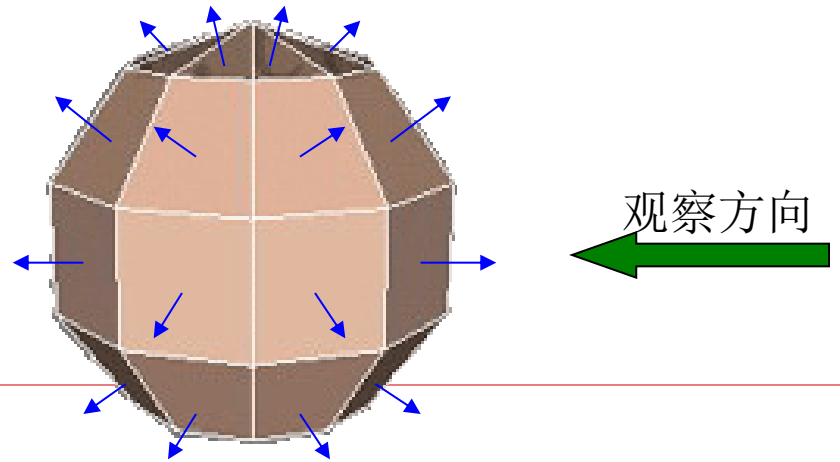
包围盒不相交，物体必然不相交



包围盒相交，物体可能相交

背面剔除

- 外法向：多边形指向物体外部的法向
- 多边形外法向与投影方向(观察方向)的夹角是钝角时，为前向面(正面)，否则为后向面(背面)。
- 消隐前先剔除后向面，可大大减少计算量。

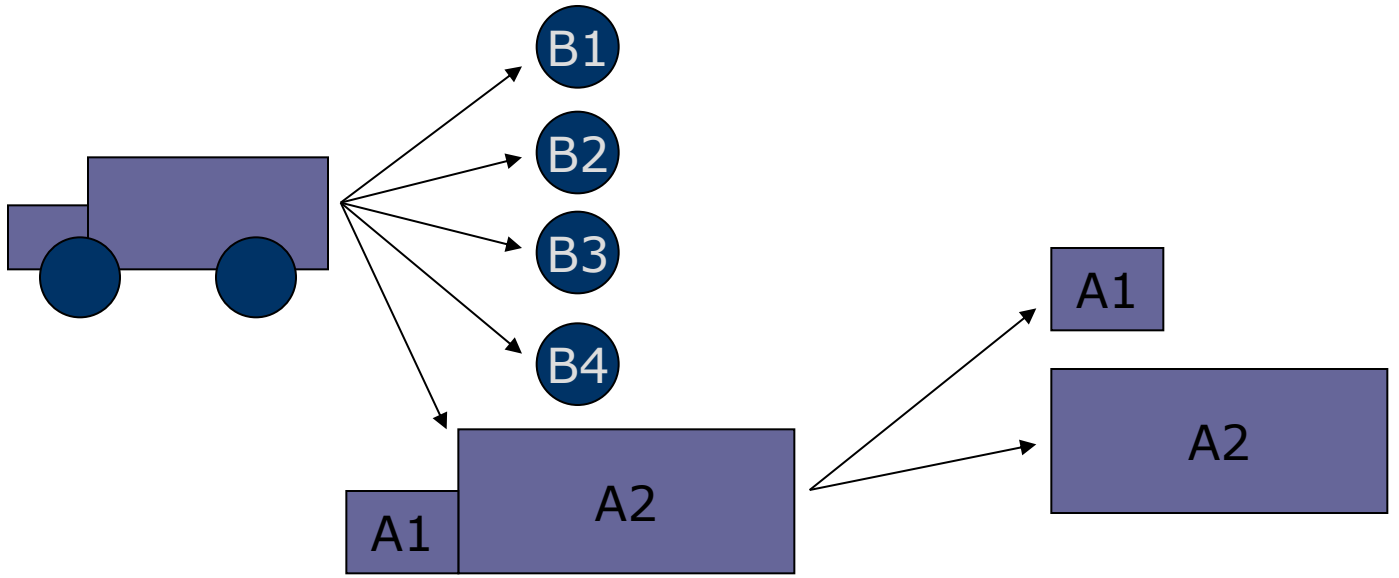


空间分割技术

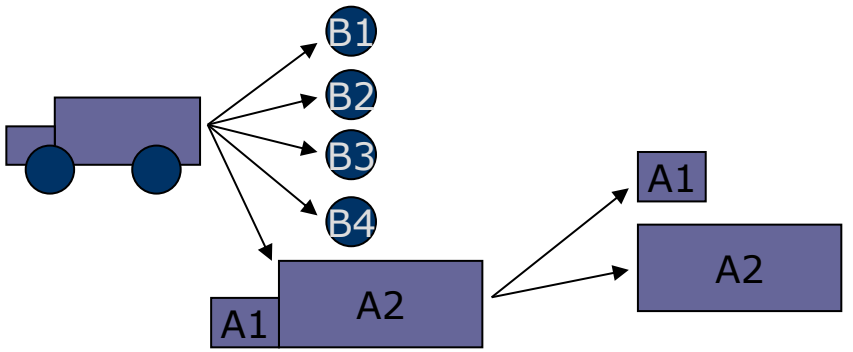
- 依据：场景中的物体，它们的投影在投影平面上是否有重叠部分？(是否存在相互遮挡的可能？)对于根本不存在相互遮挡关系的物体，应避免这种不必要的测试。
- 方法：将投影平面上的窗口分成若干小区域；为每个小区域建立相关物体表，表中物体的投影于该区域有相交部分；则在小区域中判断那个物体可见时，只要对该区域的相关物体表中的物体进行比较即可。

物体分层表示

- 利用模型变换将物体表示为树形。从而减少场景中物体的个数，降低算法复杂度。



- 方法：将父节点所代表的物体看成子节点所代表物体的包围盒，当两个父节点之间不存在遮挡关系时，就没有必要对两者的子节点做进一步测试。
- 父节点之间的遮挡关系可以用它们之间的包围盒进行预测试。



END