第七章 消除隐藏线和隐藏面的算法

- 第一节 线面比较法消除隐藏线
- 第二节 浮动水平线算法
- 第三节 深度排序算法
- 第四节 **z-**缓冲算法
- 第五节 扫描线算法
- 第六节 区域分割算法
- 第七节 **BSP**树算法
- 第八节 光线投影算法

消隐

面消隐

线消隐

三维形体表示为多边形表面形集合

投影约定为沿着z轴正向的投影

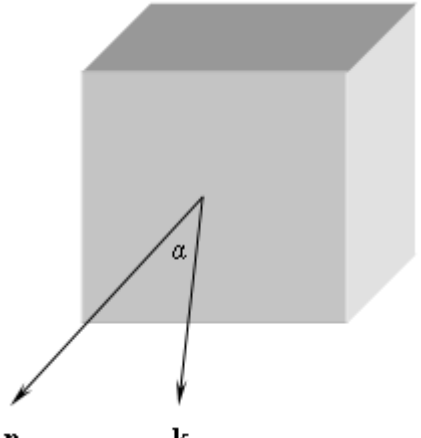
消除隐藏面算法：

图像空间算法

客体空间算法

- 图像空间(屏幕坐标系)算法对显示设备上每一个可分辨像素进行判断，看组成物体的多个多边形表面中哪一个在该像素上可见，即要对每一像素检查所有的表面。
- 客体空间(观察坐标系)算法把注意力集中在分析要显示形体各部分之间的关系上，这种算法对每一个组成形体的表面，都要与其它各表面进行比较，以便消去不可见的面或面的不可见部分。

第一节 线面比较法消除隐藏线



- 多面体的面可见性

  对象：凸多面体

  可见面：朝向观察位置的面

  观察方向：由指向观察位置的一个方向向量**k**给出，

  考查面的外法向量是**n**，则这两个向量的夹角α满足

  $0 \le \alpha < \pi/2$时，所考查面是可见的，否则就是不可见的

把$\mathbf{n}$和$\mathbf{k}$记作$\mathbf{n}=(n_x,n_y,n_z),\mathbf{k}=(k_x,k_y,k_z)$,

则

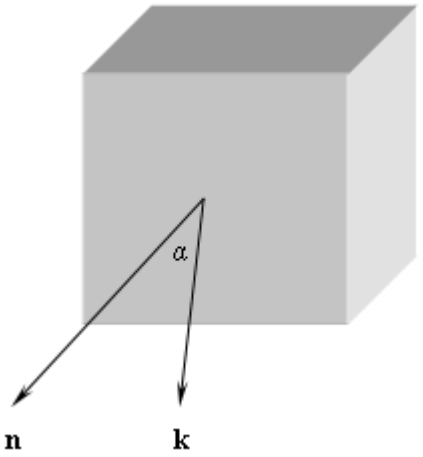
$$\cos\alpha=\frac{\mathbf{n}\cdot\mathbf{k}}{|\mathbf{n}||\mathbf{k}|}=\frac{n_xk_x+n_yk_y+n_zk_z}{\sqrt{n_x^2+n_y^2+n_z^2}\cdot\sqrt{k_x^2+k_y^2+k_z^2}}$$

$$\mathbf{n}\cdot\mathbf{k}=n_xk_x+n_yk_y+n_zk_z$$

$\mathbf{n}\cdot\mathbf{k}>0 \quad 0\le\alpha<\frac{\pi}{2}$ 面为可见

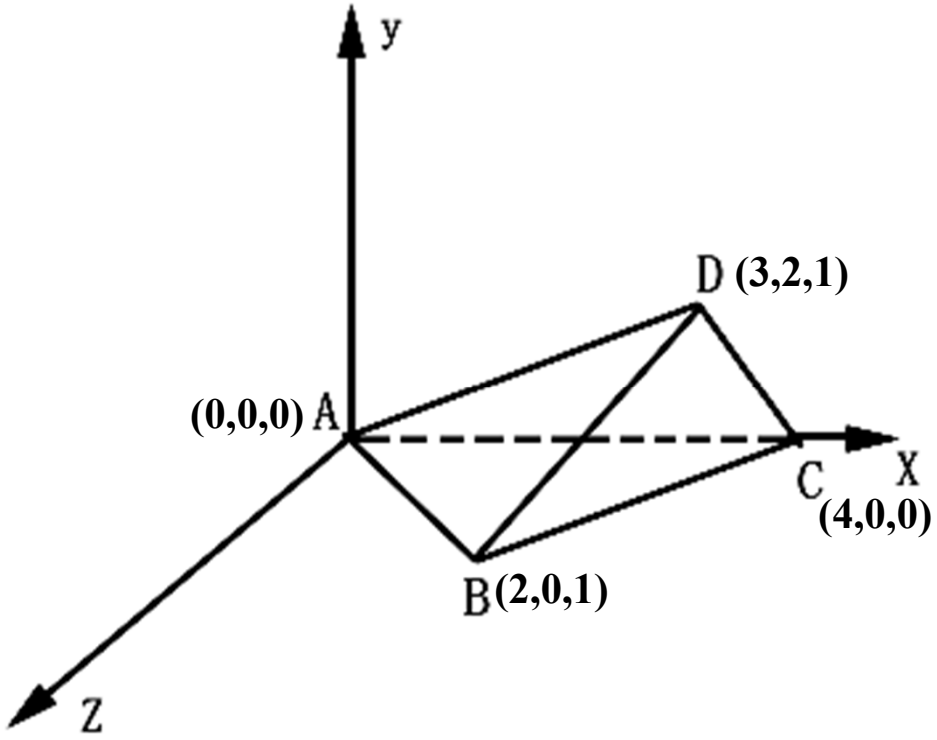
$\mathbf{n}\cdot\mathbf{k}<0 \quad \frac{\pi}{2}<\alpha\le\pi$ 面不可见

$\mathbf{n}\cdot\mathbf{k}=0 \quad \alpha=\frac{\pi}{2}$ 面退化为线。



设空间有一个四面体，顶点A，B，C，D的坐标依次是(0,0,0) ,(2,0,1) ,(4,0,0),(3,2,1)从z轴正向无穷远处观察,求各面的可见性

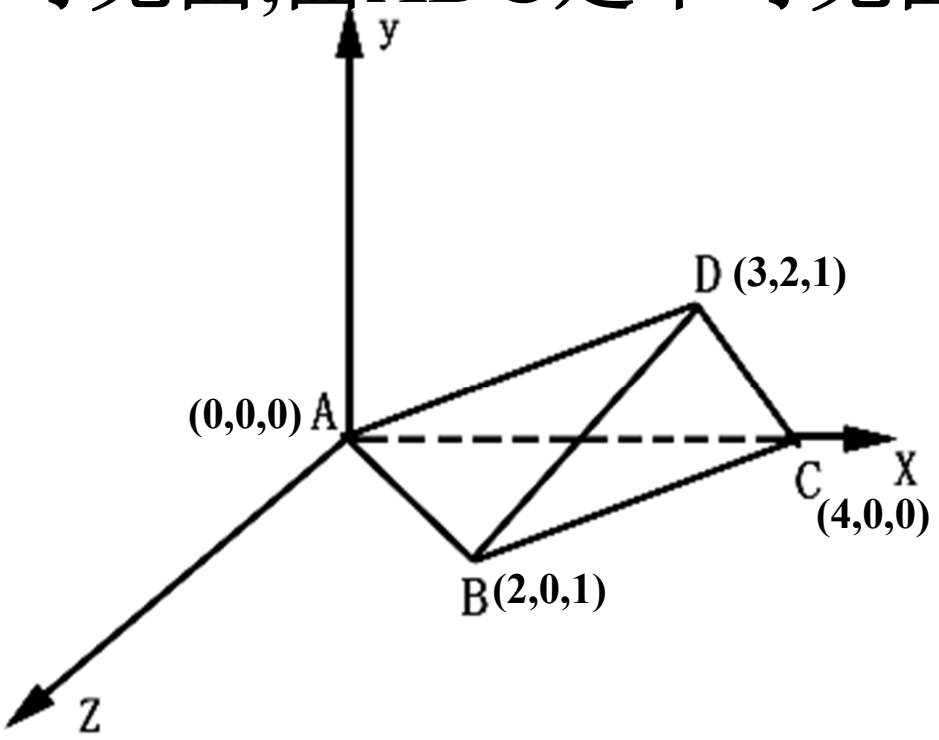
观察方向向量是k=(0,0,1)= (0,0,1)-(0,0,0),观察位置(0,0,1),A(0,0,0)是面DAB, ACB上的点。



三角面DAB的法向量是:

$$\mathbf{n} = \overrightarrow{DA} \times \overrightarrow{AB} = (-3,-2,-1) \times (2,0,1) = \begin{vmatrix} i & j & k \\ -3 & -2 & -1 \\ 2 & 0 & 1 \end{vmatrix} = (-2,1,4)$$

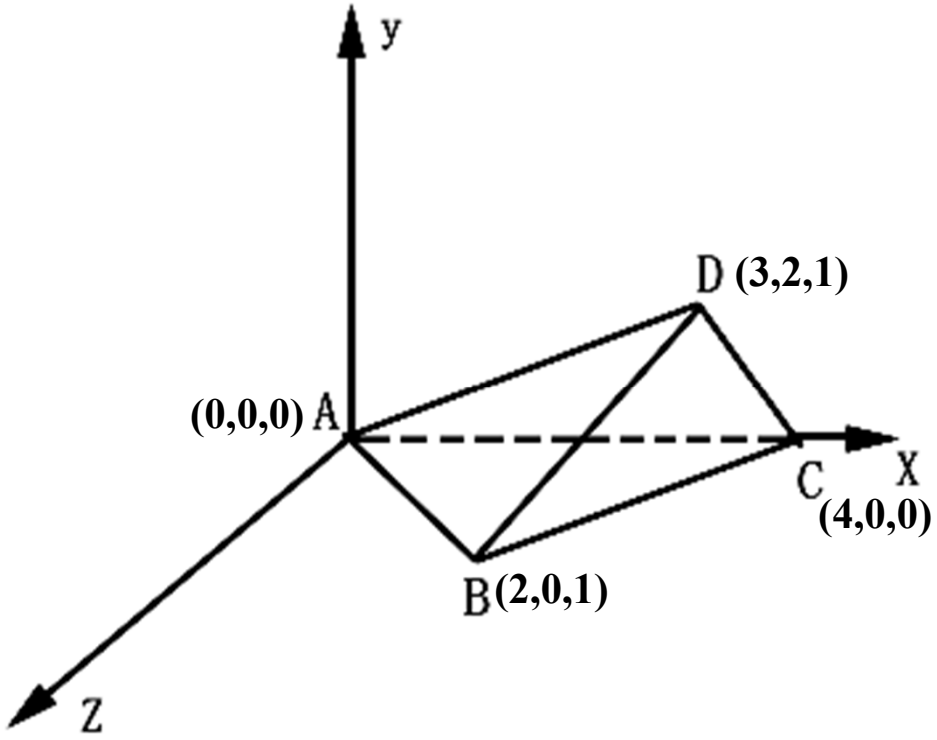
因此,n·k=4>0，面DAB为可见面.类似计算可知，面DBC是可见面,面ADC是不可见面。

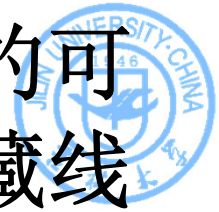


三角面ACB的法向量是:

$$\mathbf{n} = \overrightarrow{AC} \times \overrightarrow{CB} = (4,0,0) \times (-2,0,1) = \begin{vmatrix} i & j & k \\ 4 & 0 & 0 \\ -2 & 0 & 1 \end{vmatrix} = (0,-4,0)$$

因此,n·k=0，面ACB退化为线。





利用外法线就可以判断凸多面体上各表面的可见性，由此就能解决对单个凸多面体的隐藏线和隐藏面的消除问题。

单个凸多面体——可见面上的线是可见线，

多个凸多面体或非凸多面体——用上面的方法预处理，排除以后不必考虑的不可见面，剩下可能可见面。

可能可见的棱线:在可见面上或与可见面有公共边

线面比较法：(观察位置位于Z轴负方向无穷远处)

0.预处理，用外法线法判断出所有可能可见面

思想：求出所有可能可见面，可能可见面上的线段是可能可见线。

每一条可能可见线和每一个可能可见面比较，

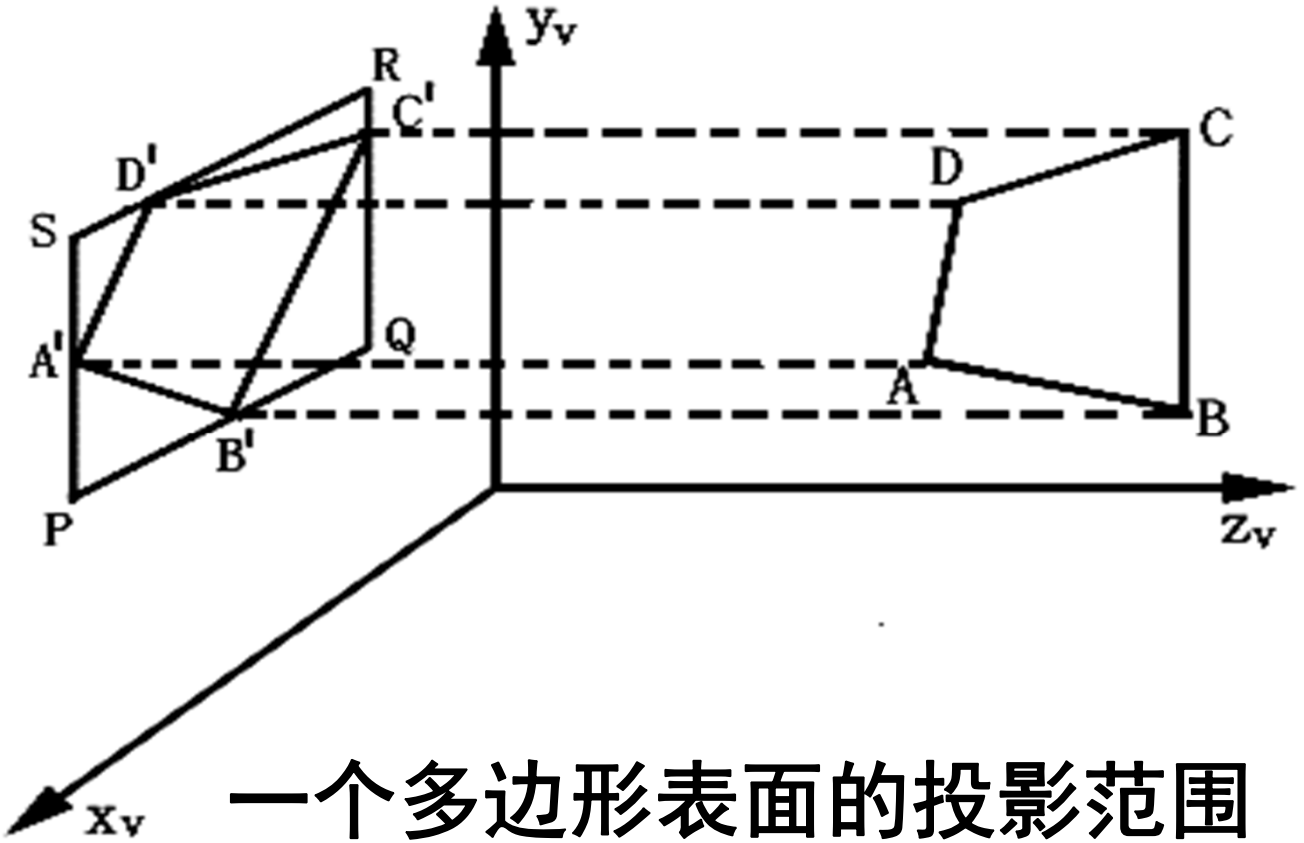
↓ ↓

线段 **多边形表面**

从而确定出可见线、隐藏线及可见线上的隐藏部分。

- 可能可见线和可能可见面

空间任一线段，只有其投影与多边形表面的投影范围发生交叠时，才可能与多边形表面有遮档关系



一个多边形表面的投影范围

1.范围检查：最大最小检验，即通过比较有关的最大或最小值来判定范围的交叠情形。（xv,yv方向）

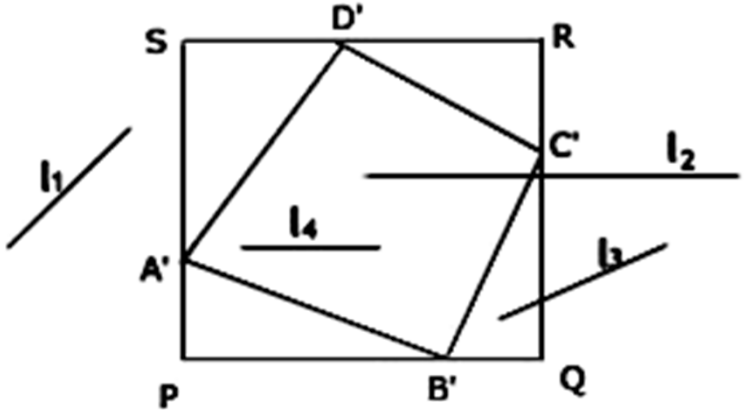
①求出线段的投影

x_{min1},x_{max1},y_{min1},y_{max1}

② 求出多边形表面的投影范围

（zv平面上包含多边形投影的最小矩形）

x_{min2},x_{max2},y_{min2},y_{max2}



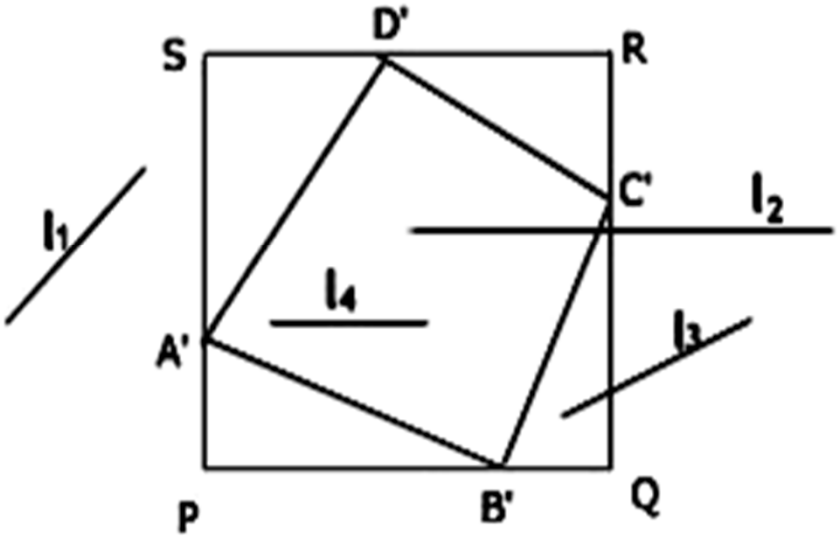
③比较投影范围：

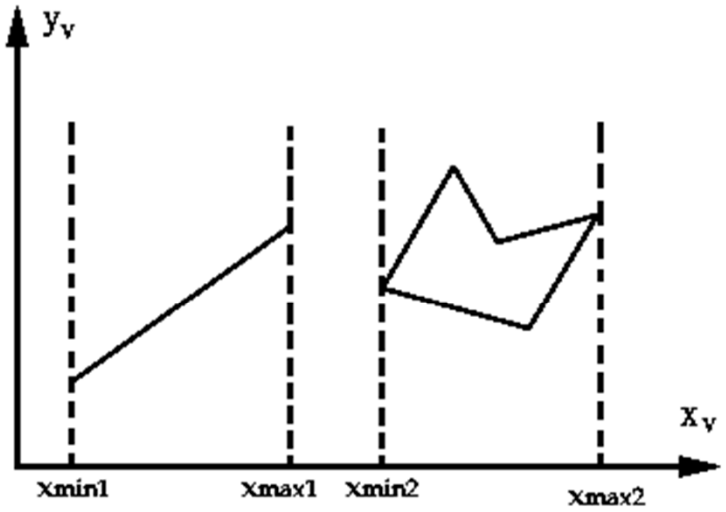
在xv方向:

若$x_{max1}<=x_{min2}$或$x_{max2}<=x_{min1}$,则无遮挡关系

在yv方向：

若$y_{max1}<=y_{min2}$或$y_{max2}<=y_{min1}$,则无遮挡关系

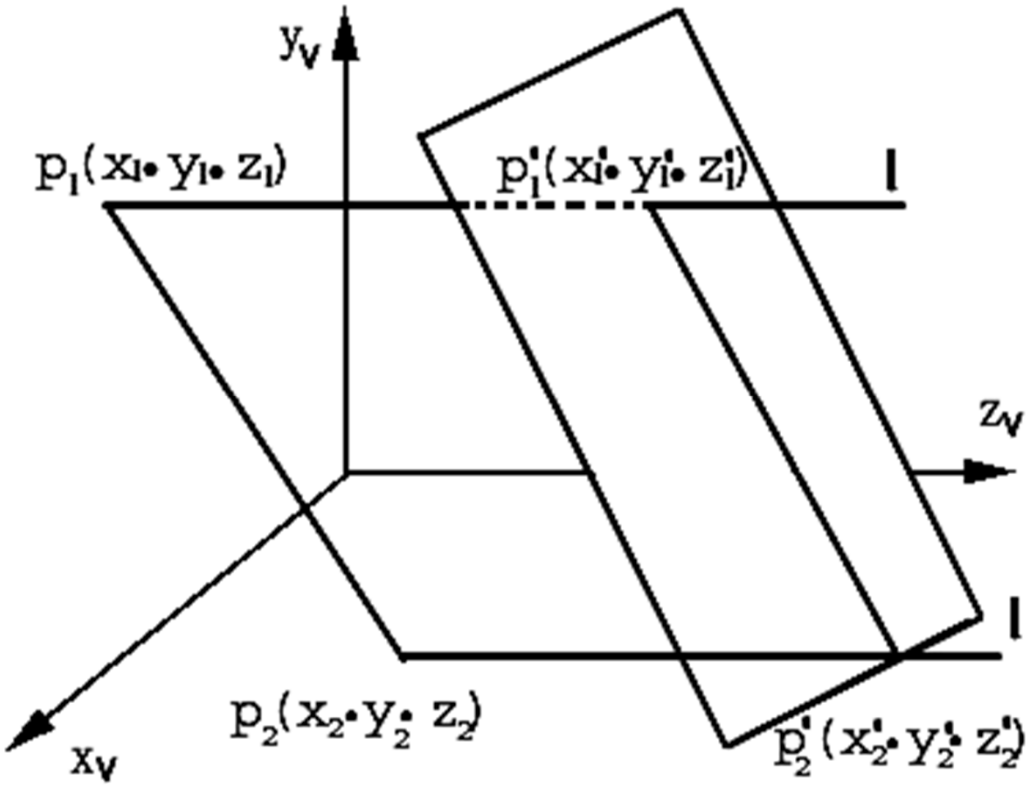




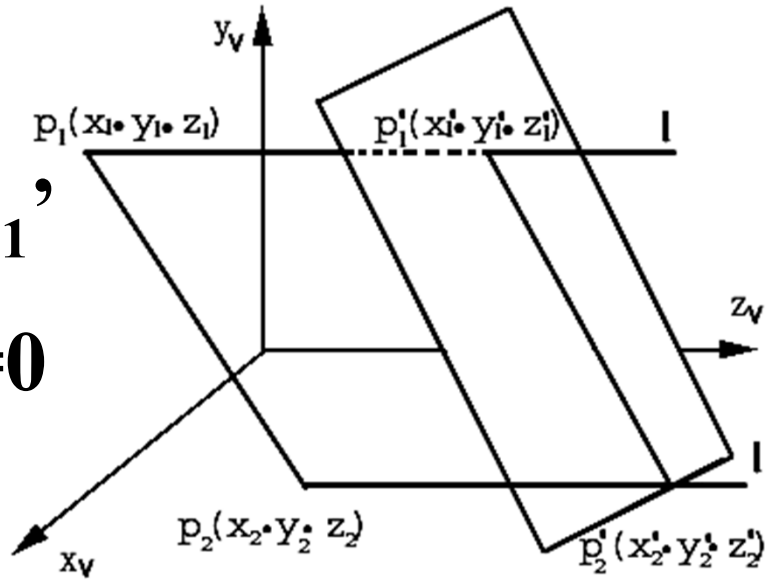
2. z_v方向粗略的深度检查。

若线段投影的最大z坐标z_{max1}小于多边形表面投影范围最小的z坐标z_{min2}，则线段完全在表面前面，不发生遮挡现象，可以不必再往下做精确的深度检验。

$z_{max1} <= z_{min2}$



3.精确的深度检验



已知z_1,求z_1 在平面的投影点z_1’

设平面方程为Ax+By+Cz+D=0

直线L_1的参数方程

$X=x_1$， $Y=y_1$， $Z=z_1+t$,

代入平面方程得： $Ax_1+By_1+C(z_1+t)+D=0$

解得 $$t=-\frac{Ax_1+By_1+Cz_1+D}{C}$$

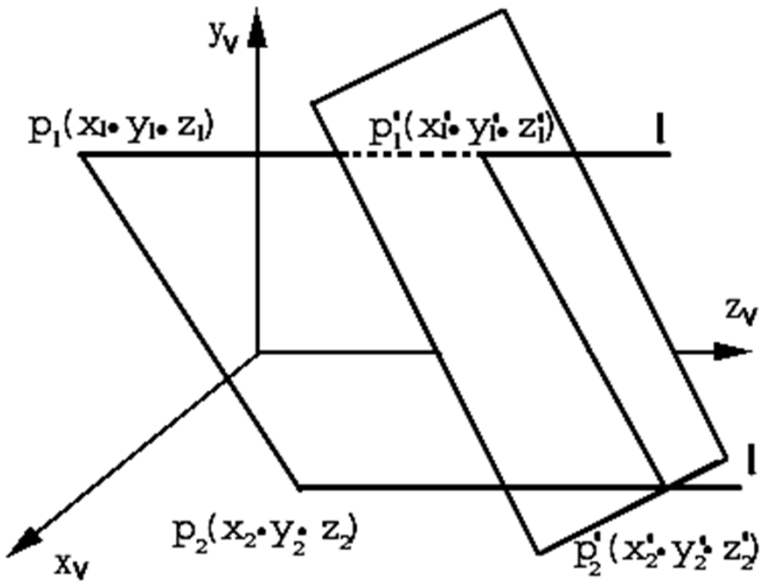
若$t\geq 0$,则$z_1\leq z_1$’， 靠近视点 可见
若$t<0$,则$z_1>z_1$’， 远离视点

线段P_1P_2

若$z_1 \leq z_1'$且$z_2 \leq z_2'$，则线段不被遮挡

若$z_1 \geqslant z_1'$且$z_2 \geq z_2'$，有可能遮挡需要进一步检查

若非以上两种情况，必然相交，求出交点，交点将原线段分成两段，分别属于上面两种情况

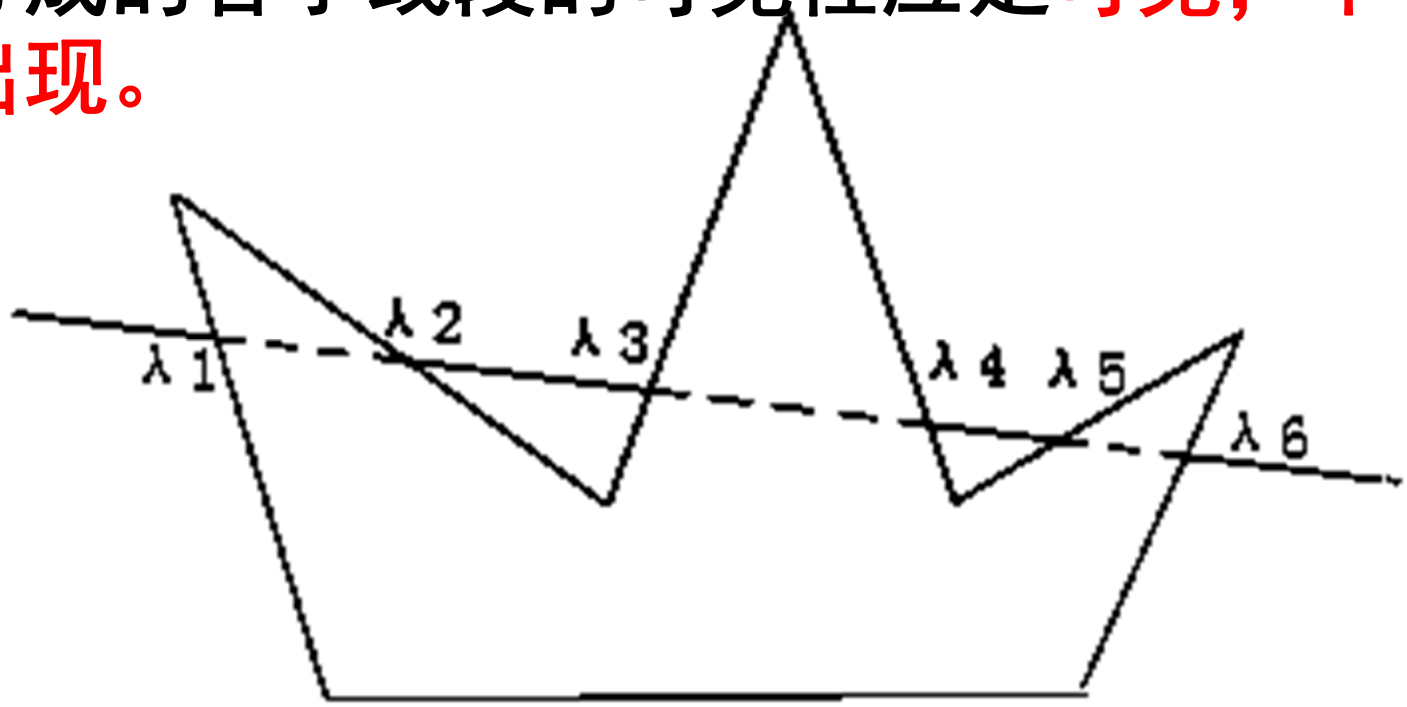


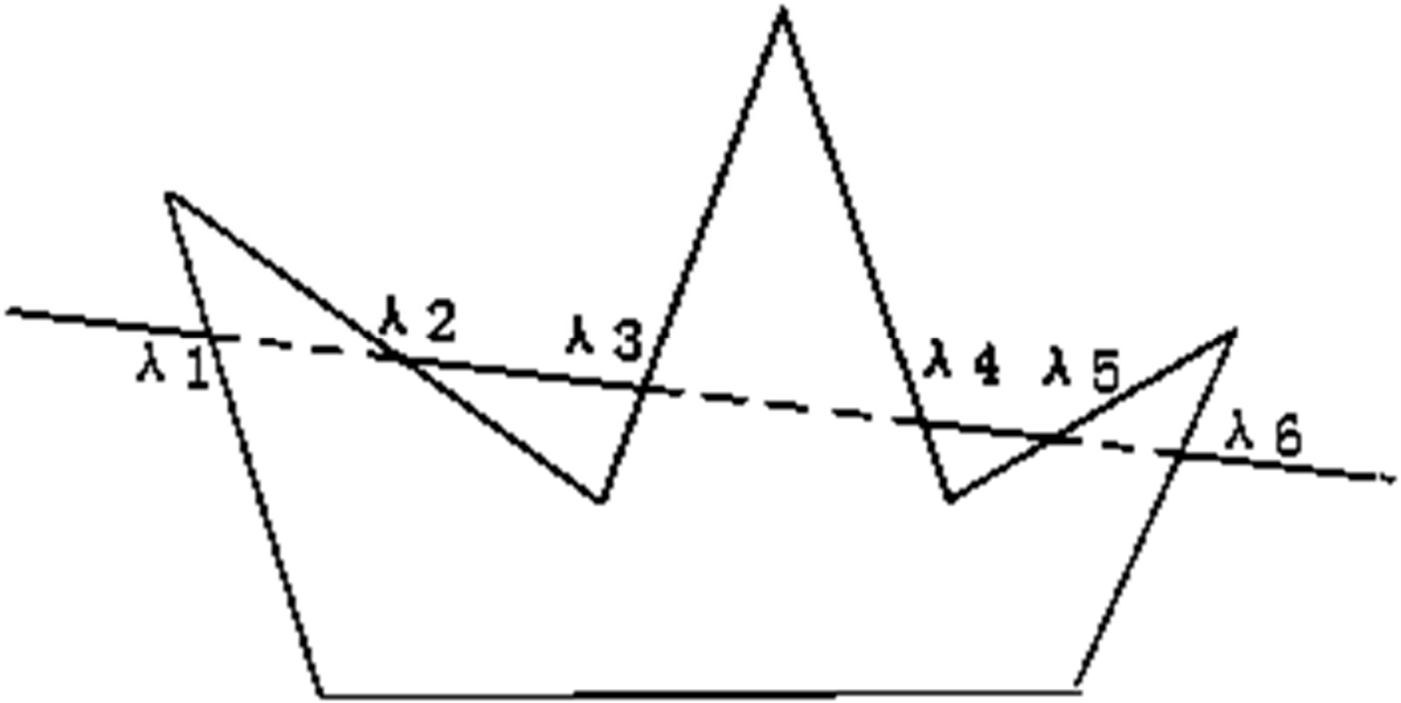
4.进一步检查

对平面遮挡了线段的哪些部分做精确计算

求线段投影与多边形边框投影的交点

设交点已经求出，设其对应的参数λ，按从小到大依次排序后是λ_1，λ_2，…，则这些交点将投影线段分成的各子线段的可见性应是可见，不可见交替出现。





判断子线段的可见性：

取子线段上任意一点，若这点在多边形各边投影所形成的封闭多边形内，该子线段就不可见，否则就可见。

线面遮挡判断算法如下：

1）视点和线段在给定平面的同侧，线段不被给定平面遮挡。

2）线段的投影和平面的投影的包围盒不相交，线段不被平面遮挡。

3）计算直线与平面是否相交。若无相交转4）；否则交点在线段内部或外部。若交点在线段的内部，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）；若交点在线段外部，转4）步。

4）求所剩线段的投影和平面边界的投影的所有交点，根据交点在原直线参数方程中的参数值求出Z值。若无交点，转7）步。

5）所求的各交点将线段的投影分成若干段，求出第一段中点。

6）若第一段中点在平面的投影内，则相应的段被遮挡，否则不被遮挡，其它段依次交替取值进行判断。

7）算法结束。

第二节 浮动水平线算法

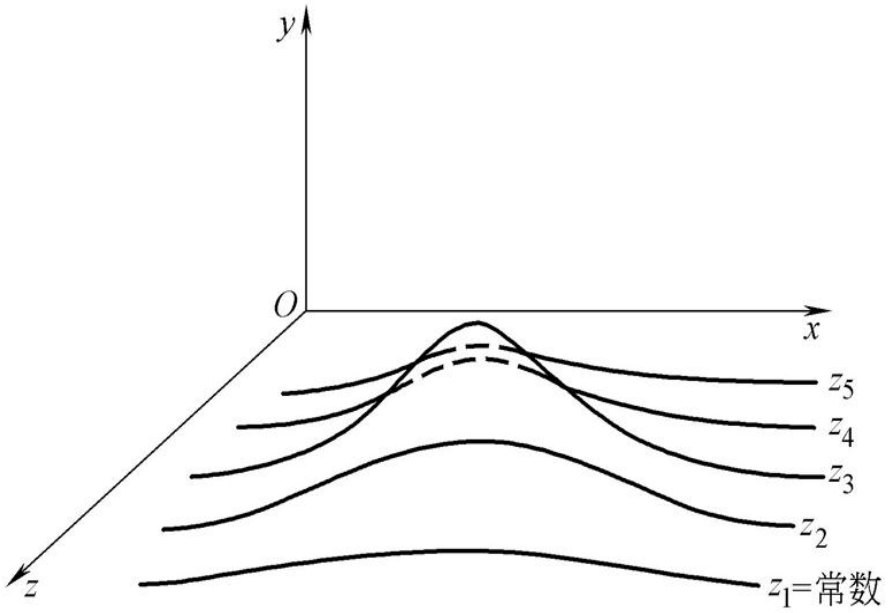
浮动水平线算法又称峰值线算法,它适用于绘制双变量函数表示的空间曲面。这类函数的形式为

F(x,y,z)=0或y=f(x,z)或x=g(y,z)

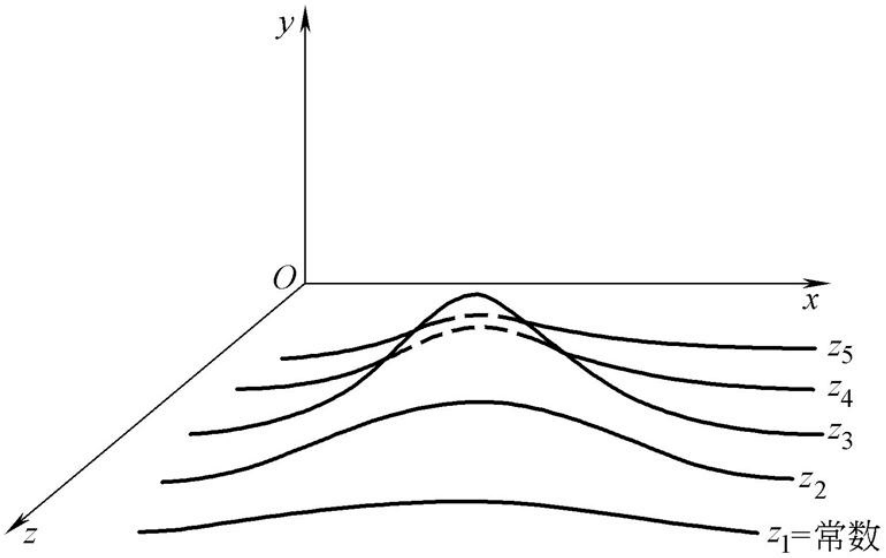
如,函数$y^2=x^2+z^2$表示顶点在原点的圆锥面;

又如函数$y^2=x^2+z^2$表示顶点在原点的旋转抛物面。但对于函数$y=-8e^{-x^2-z^2}(x+y)$,直观上就难于描述曲面的形状。

- 由于曲面方程的这种特定的表示形式,它的消隐绘图常采用浮动水平线算法,并且在图像空间中实现。算法的基本思想是,用一系列与坐标平面平行的平面(x,y或z为定值)去切割曲面,得到一系列平面曲线,用这些平面曲线表示曲面,把三维问题转化为二维问题。

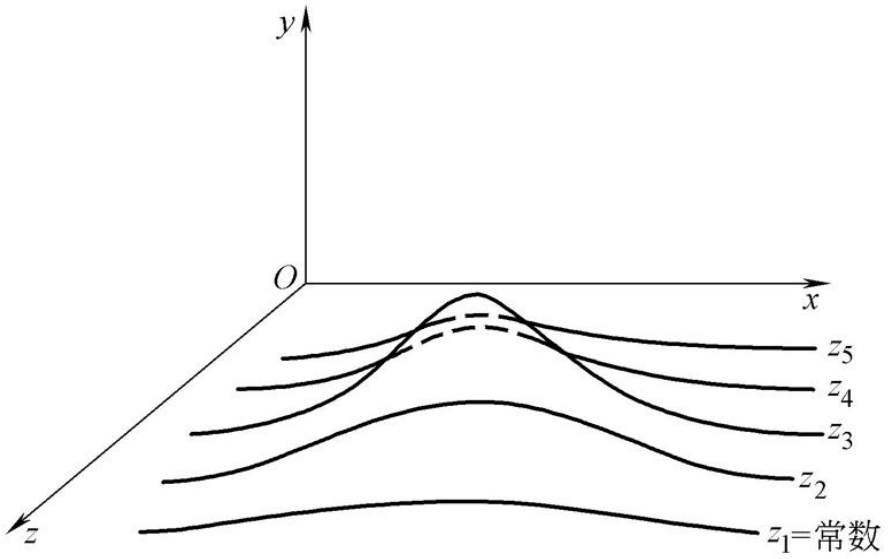


- 考虑$\begin{cases} y = f(x,z) \\ z = z_1,z_2,\cdots \end{cases}$，$z=z_i(i=1, 2, \cdots)$是一个平行于$xOy$平面的平面，它与曲面$y=f(x, z)$的交线$y=f(x, z_i)$是一条平面曲线。



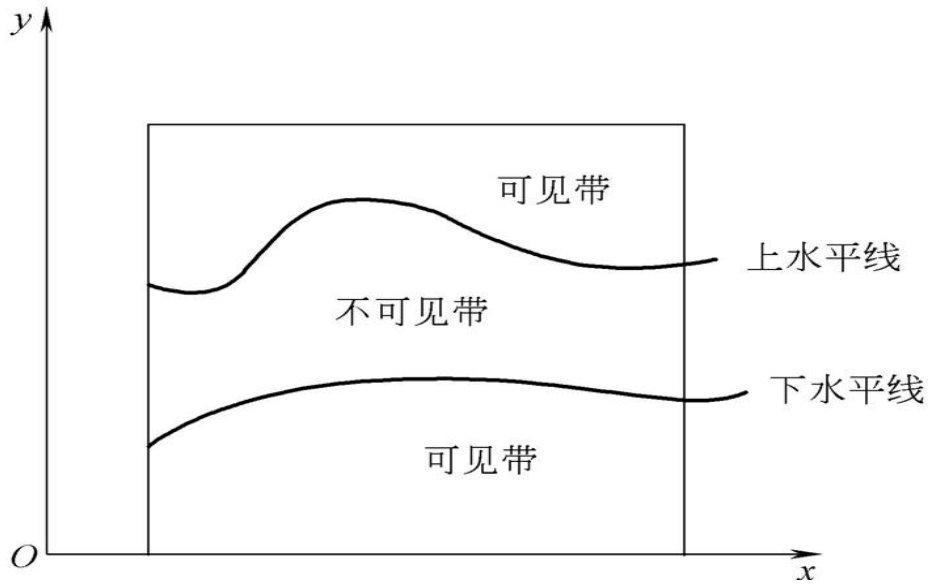
$$\begin{cases} y = f(x,z) \\ z = z_1, z_2, \ldots \end{cases}$$

假设它是关于单自变量x的单值函数,把这些曲线一一投影到z=0平面上,立即可得到一个消隐算法。首先将$z=z_i$的平行平面按其离观察点距离的递增顺序排序,然后从离观察点最近的平面开始,求出这一平行平面族交于曲面的剖面曲线,即对于图像空间中的每一x值,求出其对应的y值。在相邻两个x值之间,用直线段表示曲线。最后由近及远,逐条画出已消去不可见部分的曲线。



$$\begin{cases} y = f(x,z) \\ z = z_1, z_2, \ldots \end{cases}$$

- 算法的实现相当简单,只要设置两个长度等于图像空间在x方向分辨率的一组数组即可。一个为上水平线数组,用于存储每一x值处平面曲线所取到的最大y值;一个为下水平线数组,用于存储每一x值处平面曲线所取到的最小y值。



- 显然,两个数组中的值记录了当前的水平线位置,每画出一条新的曲线,就要相应修改这两个数组。这两条浮动水平线在z=0平面上形成了三个带

- 设在任意时刻所画最后一点为x_n,当前要画的点是x_{n+k},根据x_n和x_{n+k}在三个带中的位置,算法可表述为:
- 对于当前平面曲线上的任一给定x值,若其对应的y值均大于先前曲线上同一x对应的最大y值或小于对应的最少y值,则曲线可见;否则不可见。
- 若从上一x值(x_n)至当前x值(x_{n+k})之间的曲线段变成可见或不可见,则求交点(x_i)。
- 若从x_n至x_{n+k}之间的曲线段全部可见,则画出整个曲线段;若该曲线段由可见变成不可见,则画出从x_n至x_i之间的曲线段;若该曲线段由不可见变成可见,则画出从x_i至x_{n+k}之间的曲线段。
- 最后,修改上、下水平线数组。

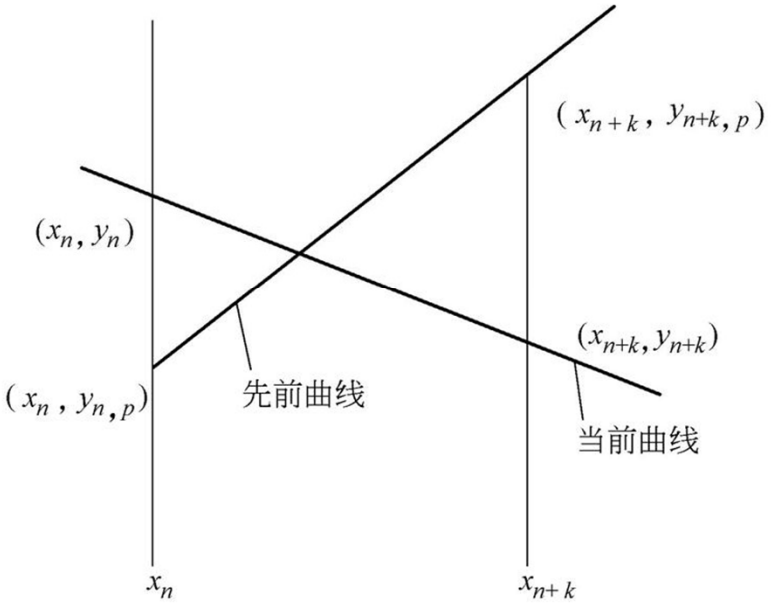
位于x_n, y_n和x_{n+k}, y_{n+k}的当前曲线段与先前曲线段的交点求解问题：

当前曲线为

$$\frac{x-x_n}{x_{n+k}-x_n}=\frac{y-y_n}{y_{n+k}-y_n} \quad (7\text{-}8)$$

先前曲线为

$$\frac{x-x_n}{x_{n+k}-x_n}=\frac{y-y_{n,p}}{y_{n+k,p}-y_{n,p}} \quad (7\text{-}9)$$



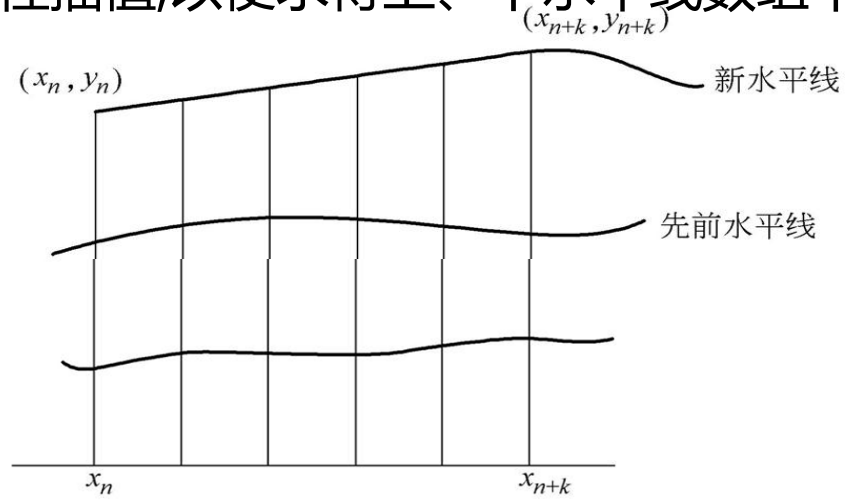
解得

$$x=\frac{y_n x_{n+k}-y_n+kx_n-y_{n,p}x_{n+k}+y_{n+k,p}x_n}{y_{n+k,p}-y_{n,p}-y_{n+k}+y_n} \quad (7\text{-}10)$$

$$y=\frac{y_n-y_{n+k,p}-y_{n+k}y_{n,p}}{y_{n+k,p}-y_{n,p}-y_{n+k}+y_n} \quad (7\text{-}11)$$

这里p表示先前曲线。

如果对于某些x值,相对应的y值不存在,那么上、下水平线数组的值将无法确定。这时,需对已知点做线性插值,以便求得上、下水平线数组中相应元素的值。



设

$$y=ax+b \quad (7-12)$$

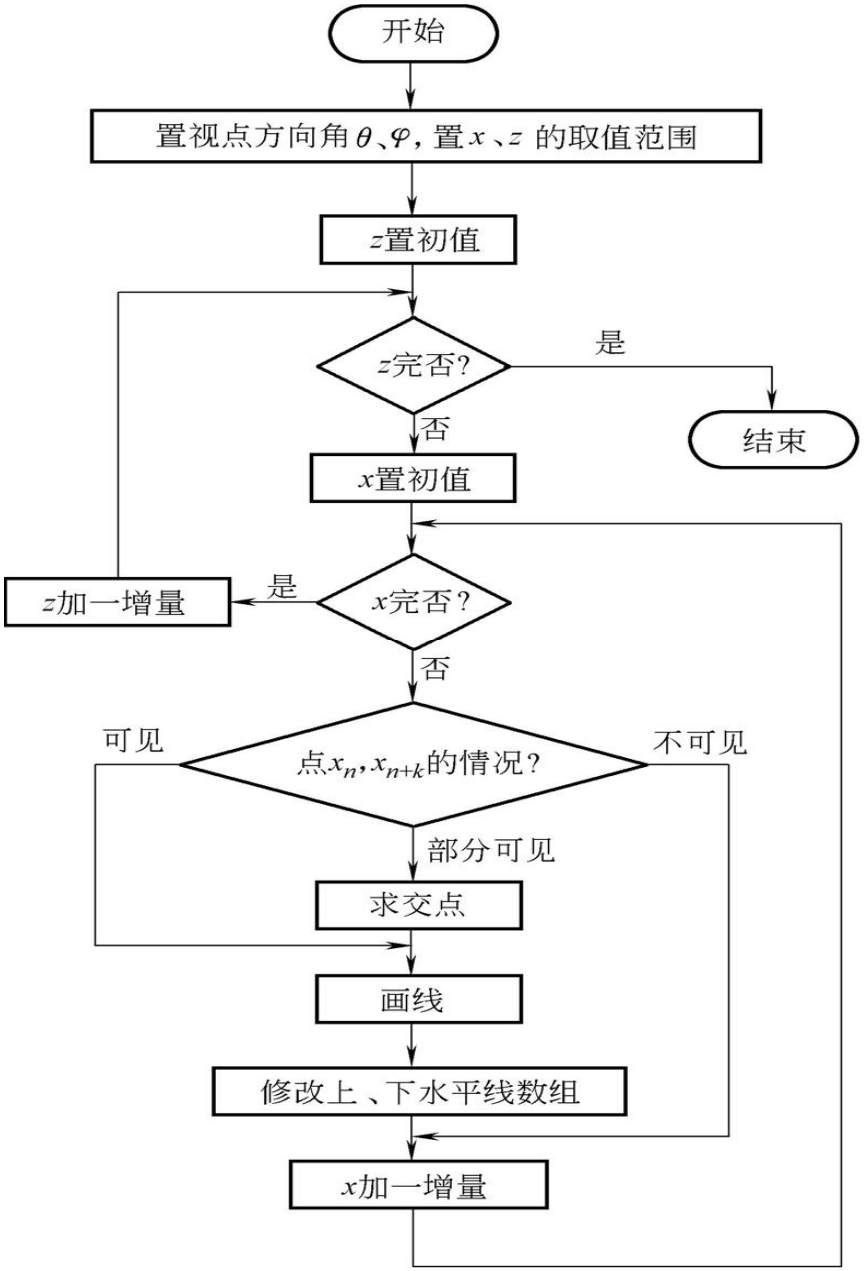
在点(x_n, y_n)处有$y_n=ax_n+b$, 在点(x_{n+k}, y_{n+k})处有$y_{n+k}=ax_{n+k}+b$,

解得

$$a=\frac{y_{n+k}-y_n}{x_{n+k}-x_n},\ b=\frac{y_n x_{n+k}-y_{n+k}x_n}{x_{n+k}-x_n} \quad (7-13)$$

把式(7-13)代入直线方程(7-12), 这样, 可以用循环变量I从x_n+1到x_{n+k}, 步长为1求得插值点的y坐标。

- 算法允许观察者在任何角度下对曲面进行观察,并且可以有透视投影和平行投影两种选择。为了保证图形总是自左至右出现在屏幕上,若0°<θ<180°,z按递减顺序变化,同一平面内,x按递增顺序变化;若180°<θ<360°,z按递增顺序变化,同一平面内,x按递减顺序变化。


开始
置视点方向角θ、φ，置x、z的取值范围
z置初值
z完否?
是
结束
否
x置初值
z加一增量
是
x完否?
否
可见
点x_n,x_{n+k}的情况?
不可见
部分可见
求交点
画线
修改上、下水平线数组
x加一增量

第三节 深度排序算法

深度排序算法：先画最远的，再画最近的

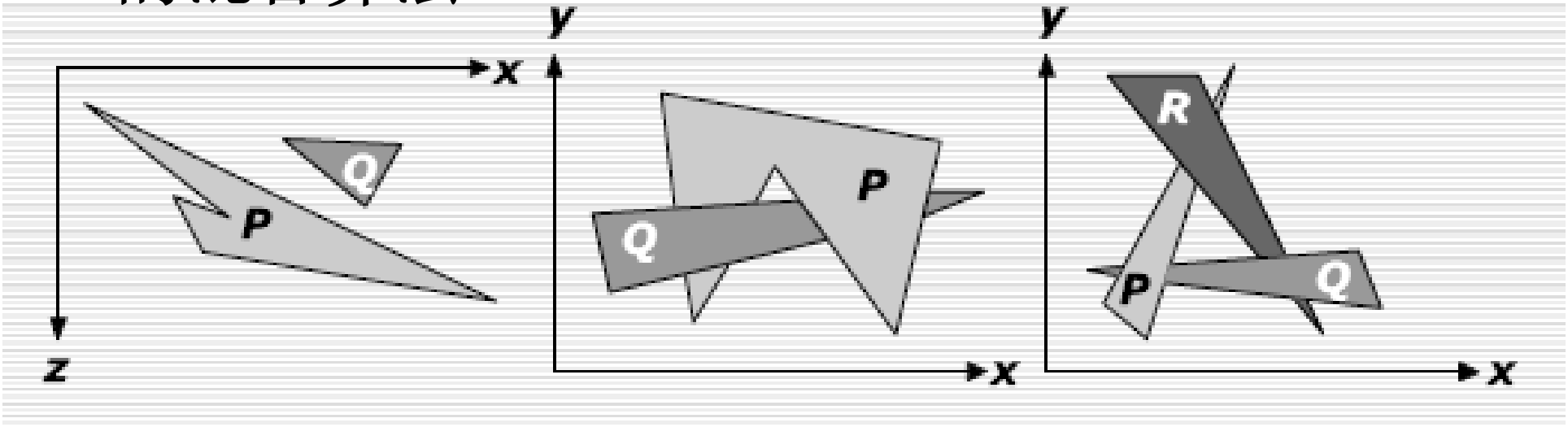
（优先级算法，画家算法）

深度排序算法的主要步骤：

1. 把所有的多边形按顶点最大z坐标值进行排序。（客体空间）(观察坐标系)
2. 解决当多边形z范围发生交叠时出现的不明确问题。
3. 按最大z坐标值逐渐减小的次序，对每个多边形进行扫描转换。（图像空间）(屏幕坐标系)

算法的基本思想是按多边形离开观察位置的距离进行排序，然后按照距离减少的次序，把每个多边形内部点应有的像素值送入帧缓存存贮器中。

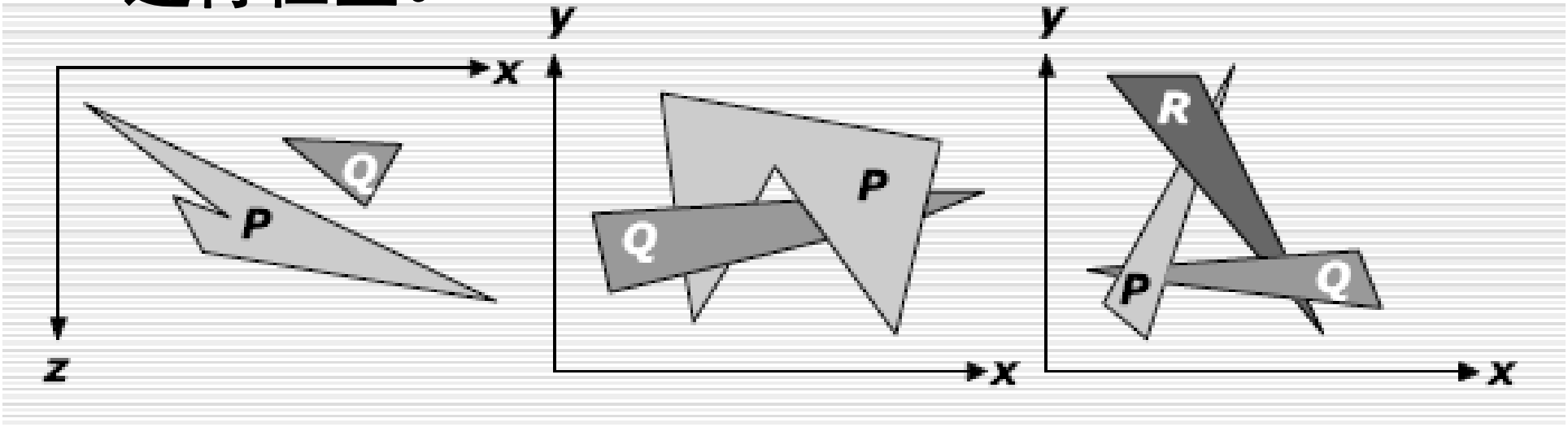
算法考查多边形的深度次序是在客体空间中进行，图形显示时覆盖步骤是在图像空间中实现，所以可以说是一个客体空间和图像空间的混合算法。



不明确问题检验方法

所有多边形按顶点最大z坐标值排序后得到一个排序表，设P是排在表中最后的那个多边形。

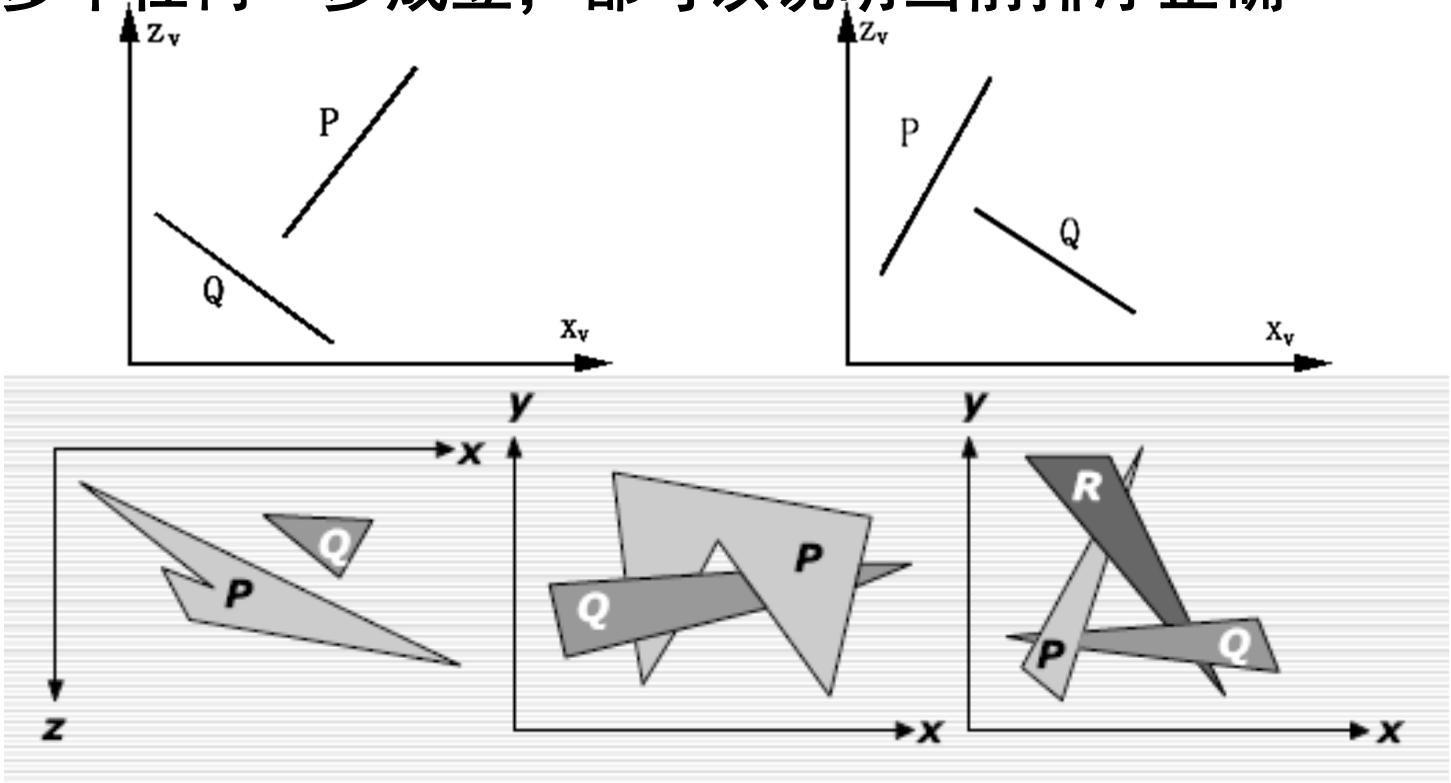
设Q是排在P前面并且z坐标范围与其发生交叠的一个多边形，对Q与P的次序关系进行检查。



z坐标范围发生交叠的QP次序检查，可按下列五步进行，每个步骤判断一种情况。

1. 多边形的x坐标范围不相交叠，所以多边形不相交叠。
2. 多边形的y坐标范围不相交叠，所以多边形不相交叠。
3. P整个在Q远离观察点的一侧。
4. Q整个在P的靠近观察点的一侧。
5. 多边形在z=0平面上的投影本身不相交叠。

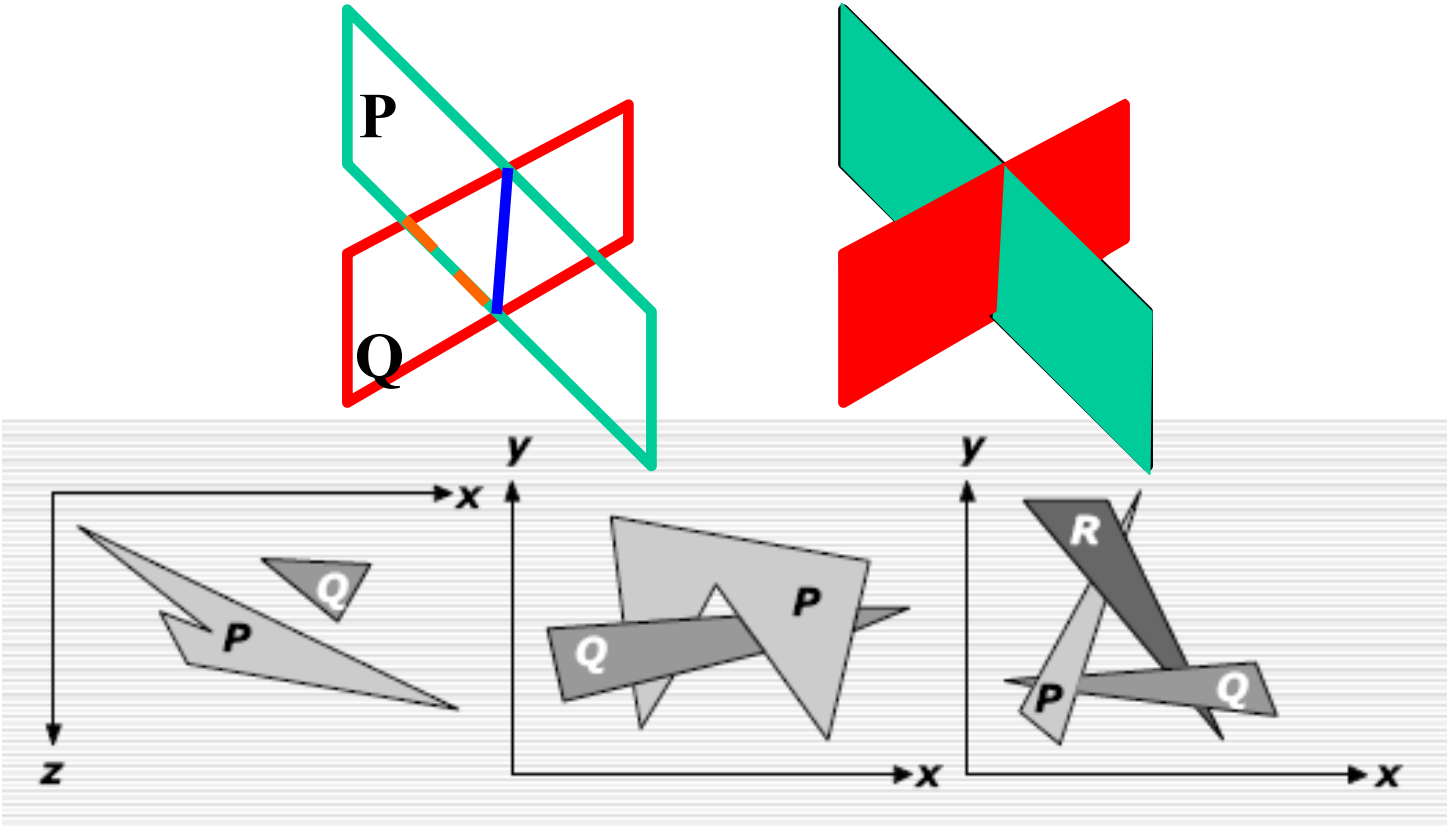
以上五步中任何一步成立，都可以说明当前排序正确



如果五步检查都为假，就假定P是遮挡了Q，交换P和Q在排序表中的位置。(QP→PQ)

如果仍做交换，算法会永远循环下去而没有结果。

为了避免循环，可以做一个限制。当做过首次五步检查后，发生某个多边形被移到排序表的末尾时，就立即加上一个标记，以后就不能再做移动。出现再次应该移动时，用一个多边形所在的平面，把另一个多边形裁剪分为两个。



第四节 z–缓冲算法

z–缓冲算法:(深度缓冲算法)是一种最简单的图像空间算法。

帧缓冲存储器:存储各点的像素值，初始化为背景值。

z–缓冲存储器:存储相应的z值。初始化最大z值。

对每一个多边形，不必进行深度排序算法要求的初始排序，立即就可以逐个进行扫描转换。

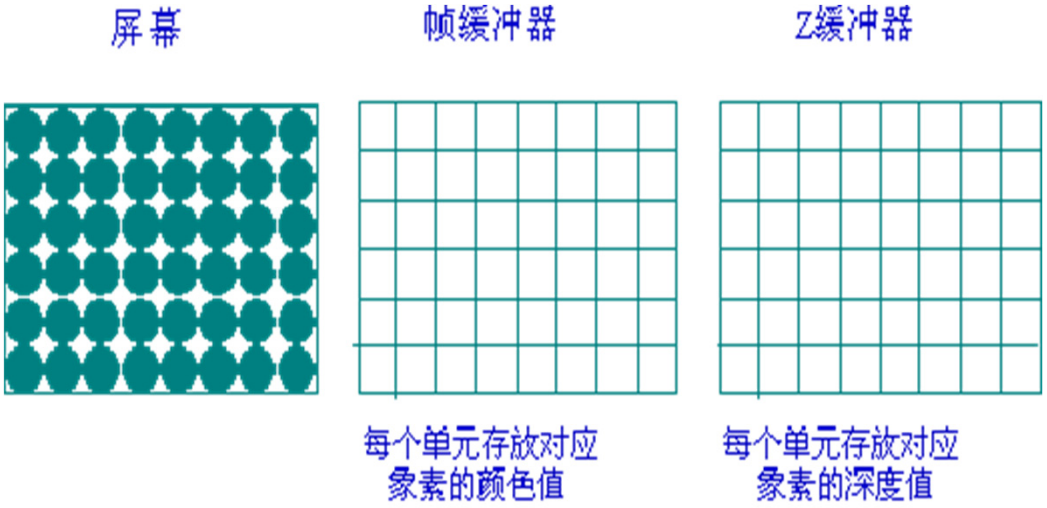
扫描转换时，对每个多边形内部的任意点(x,y),实施如下步骤：

1.计算在点(x,y)处多边形的深度值z(x,y) 。

2.如果计算所得的z(x,y)值，小于在z–缓冲存储器中点(x,y)处记录的深度值，那么就做：

（1）把值z(x, y)送入z–缓冲存储器的点(x,y)处。

（2）把多边形在深度z(x,y)处应有的像素值，送入帧缓冲存储器的点(x, y)处。



算法中深度计算，可通过多边形的顶点坐标求出所在平面的方程，然后再使用平面方程，对每个点(x,y)，解出相应的z。

对面方程 $Ax + By + Cz + D = 0$

解出z：

$$z = \frac{-D - Ax - By}{C}$$

设在点(x,y)处的深度值是z_1：$\frac{-D - Ax - By}{C} = z_1$

则在点$(x+\Delta x,y)$ 处的深度值就是

$$\frac{-D - A(x + \Delta x) - By}{C} = \frac{-D - Ax - By}{C} - \frac{A}{C}\Delta x$$

$$= z_1 - \frac{A}{C}\Delta x$$

z–缓冲算法的工作流程：

帧缓冲区置成背景色；

z–缓冲区置成最大z值；

```
for (各个多边形){
        扫描转换该多边形；
        for(计算多边形所覆盖的每个像素(x,y)){
                计算多边形在该像素的深度值Z(x,y)；
                if(Z(x,y)小于Z缓冲区中的(x,y)处的值) {
                        把Z(x,y)存入Z缓冲区中的(x,y)处；
                        把多边形在(x,y)处的亮度值存入帧缓
存区的(x,y)处；
                }
        }
}
```

第五节 扫描线算法

消除隐藏面的扫描线算法：图像空间算法

图像的建立是通过每次处理一条扫描线来完成的。

第二章 第三节 多边形扫描转换算法→推广

多边形填充算法：针对一个多边形做扫描转换

扫描线消隐算法：同时对多个多边形做扫描转换。

ET表（边表） 中各登记项按边的较小的y坐标递增排列；每一登记项下的“吊桶”，按所记x坐标递增排列。 ET中吊桶的内容：

1.与较小的y坐标对应的端点的x坐标xmin。

2.边的另一端点的较大的y坐标ymax。

3.x的增量Δx，它实际上是边的斜率的倒 数，是从一条扫描线走到下一条扫描线 时，按x方向递增的步长。

4. 边所属多边形的标记。

5. 指针指向下一条边 。

AET表（活跃边表）：存储与当前扫描线相交 的各边的信息。

PT表（Polygon Table，多边形表）：记录多边形信息，一个记录对应一个多边形。

一个记录的内容：

1．多边形所在平面方程的系数A,B,C,D
在需要比较深度时，要通过对所在(x,y)，根据平面方程解出深度z。

2．每个多边形的亮度或颜色值。实际做扫描转换时应用。

3．一个“进入\退出”标志，初值为“假(退出)”。
在扫描转换处理时，用以标记扫描线对该多边形是“进入”，还是“退出”。

APT表（活跃多边形表）：与当前扫描线相交的所有多边形，按**深度递增**排序。

扫描线y=α:

AET表中有AB,AC,

到AB时，$Flag_{ABC}$=1，AB到AC一段用ABC的颜色填充

到AC时，$Flag_{ABC}$=0，

扫描线y=β:

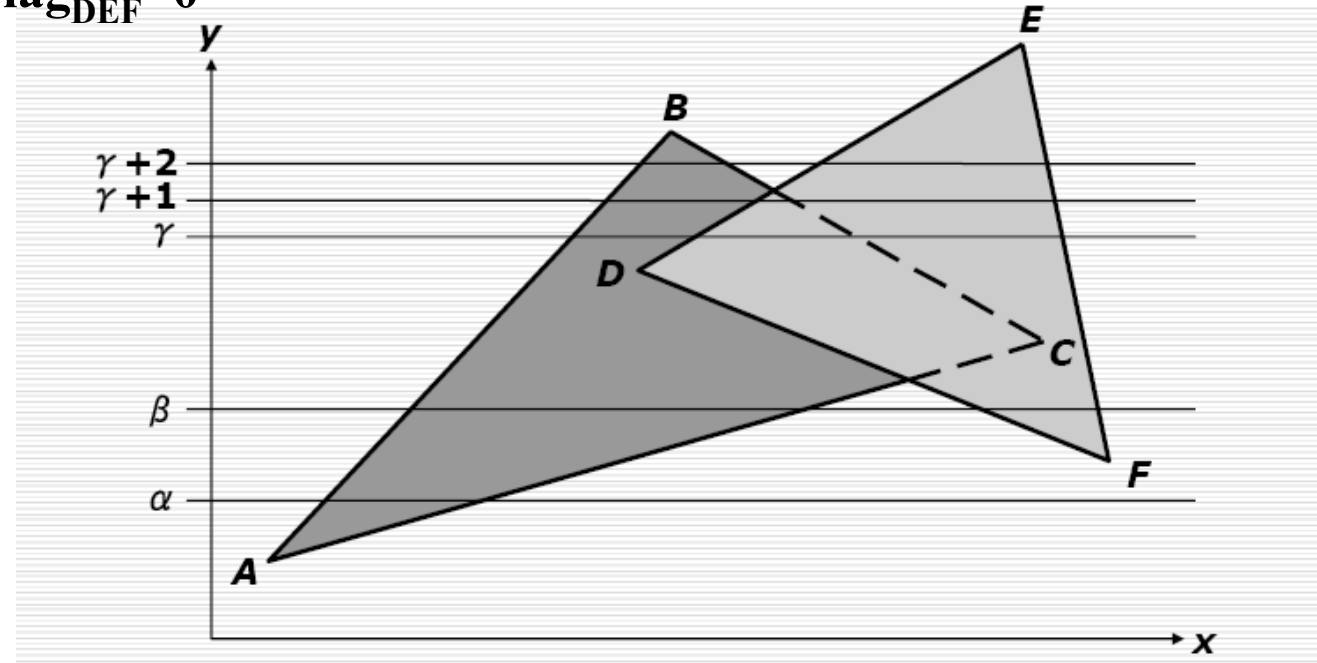
AET表中有四项AB,AC,FD,FEF,

到AB时，$Flag_{ABC}$=1，AB到AC一段用ABC的颜色填充

到AC时，$Flag_{ABC}$=0，AC到FD之间填充背景色

到FD时，$Flag_{DEF}$=1，FD到FE之间填充DEF颜色

到FE时，$Flag_{DEF}$=0



扫描线y=γ：

AET表中有四项AB,DE,CB,FE

到AB时， $Flag_{ABC}$=1 ，AB到DE之间填ABC的颜色

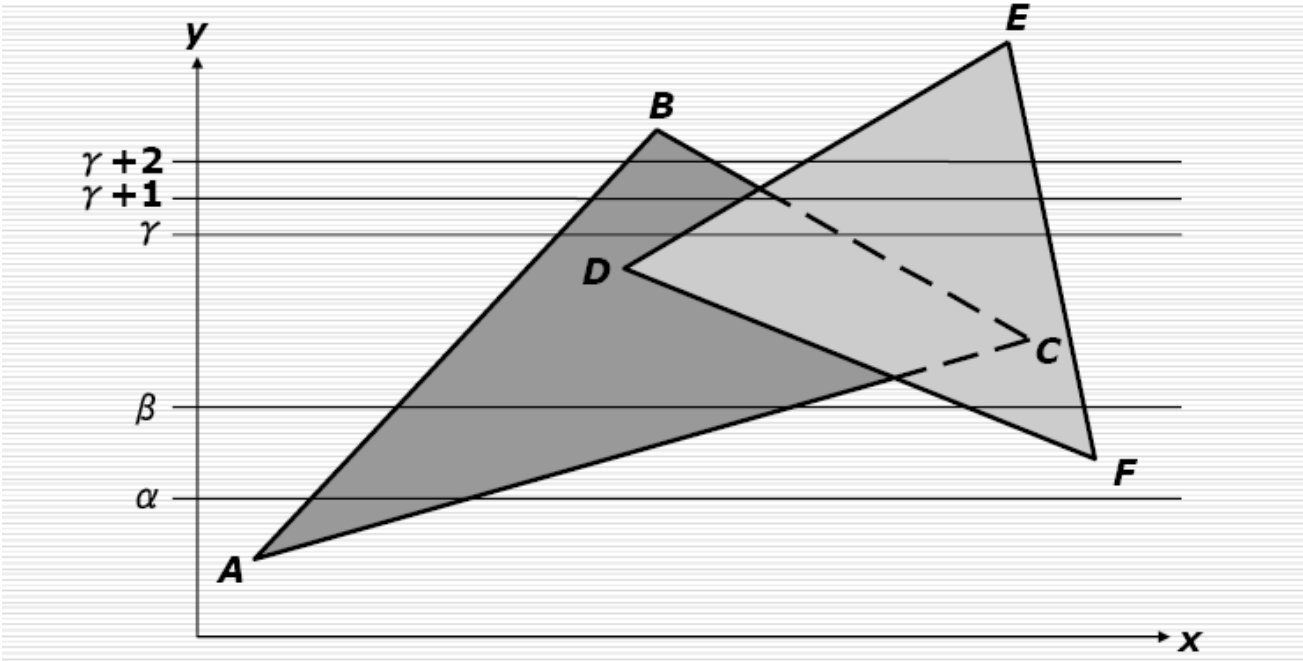
到DE时， $Flag_{DEF}$=1，做深度比较，求（x’, γ）的深度

$$z_{ABC}=\frac{-A_1x'-B_1\gamma-D_1}{C_1},z_{DEF}=\frac{-A_2x'-B_2\gamma-D_2}{C_2}$$

若Z_{DEF}<Z_{ABC} ，则DE到BC之间填DEF的颜色

到BC时， $Flag_{ABC}$=0，BC到EF之间填DEF颜色

到EF时， $Flag_{DEF}$=0



Polygonfill(EdgeTableET,COLORREFcolor)

{

1.y=置y为边表ET中各登记项对应的y 坐标中最小的值;

2.活跃边表AET初始化为空表;

3.若AET非空或ET非空，则做下列步骤，否则算法结束

{

3.1.将ET中登记项y对应的各“吊桶”合并到表AET中，将AET中各吊桶按x坐标递增排序；

3.2.在扫描线y上，按照AET表提供的x坐标对, 用 color 实施填充；

3.3.将AET表中有$y=y_{max}$的各项清除出表；

3.4.对AET中留下的各项，分别将x换为x+1/m，求出AET中各边与下一条扫描线交点的x坐标；

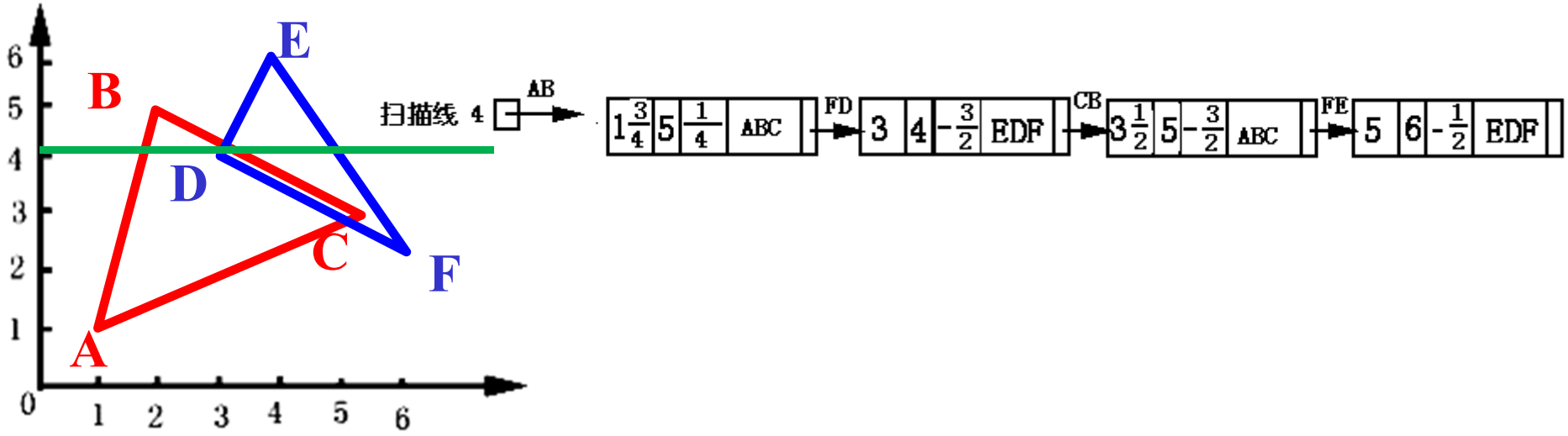
3.5.y=y＋1，返回步骤3处理下一条扫描线。

}}

通过以上的讨论，可以写出整个扫描线算法实施的步骤。

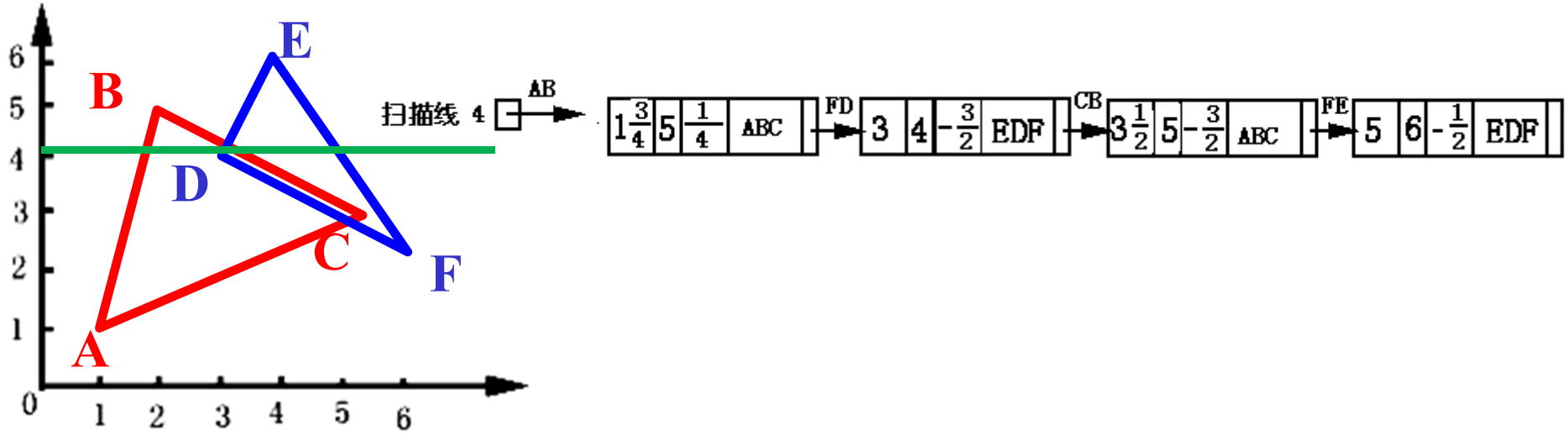
首先正确形成边表ET和多边形表PT之后，实施步骤就与第二章叙述的多边形填充的扫描线算法的实施步骤基本相同，只是需要把那里的步骤:在扫描线y上，按照AET表提供的x坐标对，用color实施填充修改加细如下：

1. 将实施扫描转换时遍查AET表中各“吊桶”的指针i初始置为1，扫描线正在多少个多边形内的累计数值s初始置为零，将活动多边形表，即扫描线正在通过的多边形按深度递增次序排列而形成的表，记为p，初始置为空。



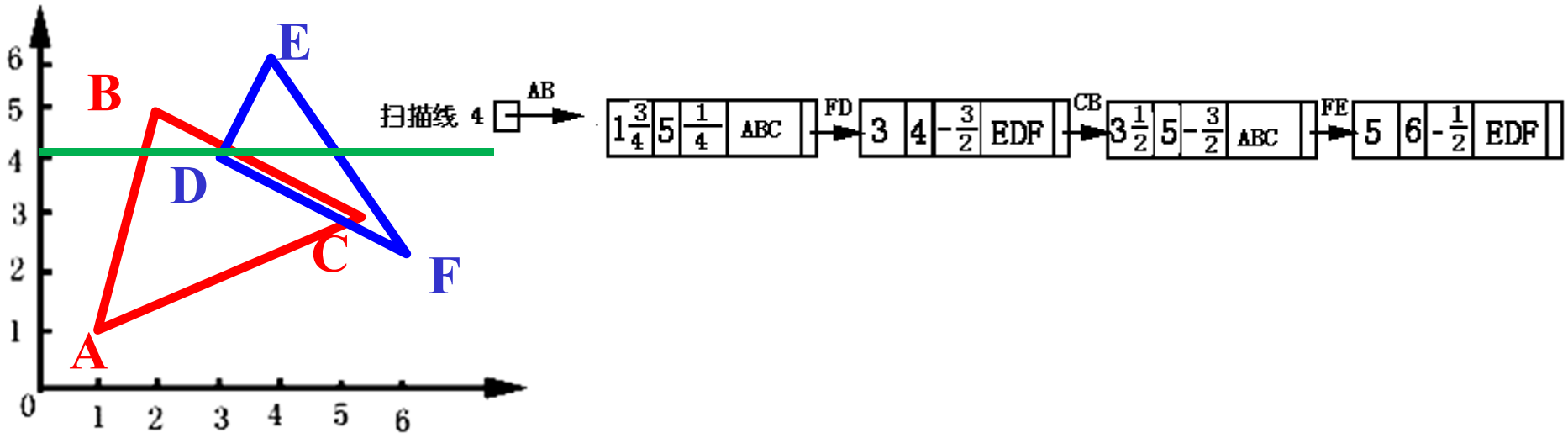
2.设第i个“吊桶”记录的相应多边形是A。若A的“进入/退出”标记F_A为“假”，则改F_A为“真”，将A加到表P的前面，s增加1。否则，F_A即为“真”，则改F_A为“假”，将P中的A去掉，s减少1。

3.若s=0，则到5。（这时扫描线不在任何多边形内，正通过背景，不必做扫描转换。）若s=1，则到4（这时扫描线只在一个多边形内，不必做深度比较，去做扫描转换。）若前面两个判断都为“假”，扫描线至少在两个多边形内，应做深度比较。对表P前面两个多边形做深度比较，比较后放回应保证P表中的多边形按深度递增的次序。

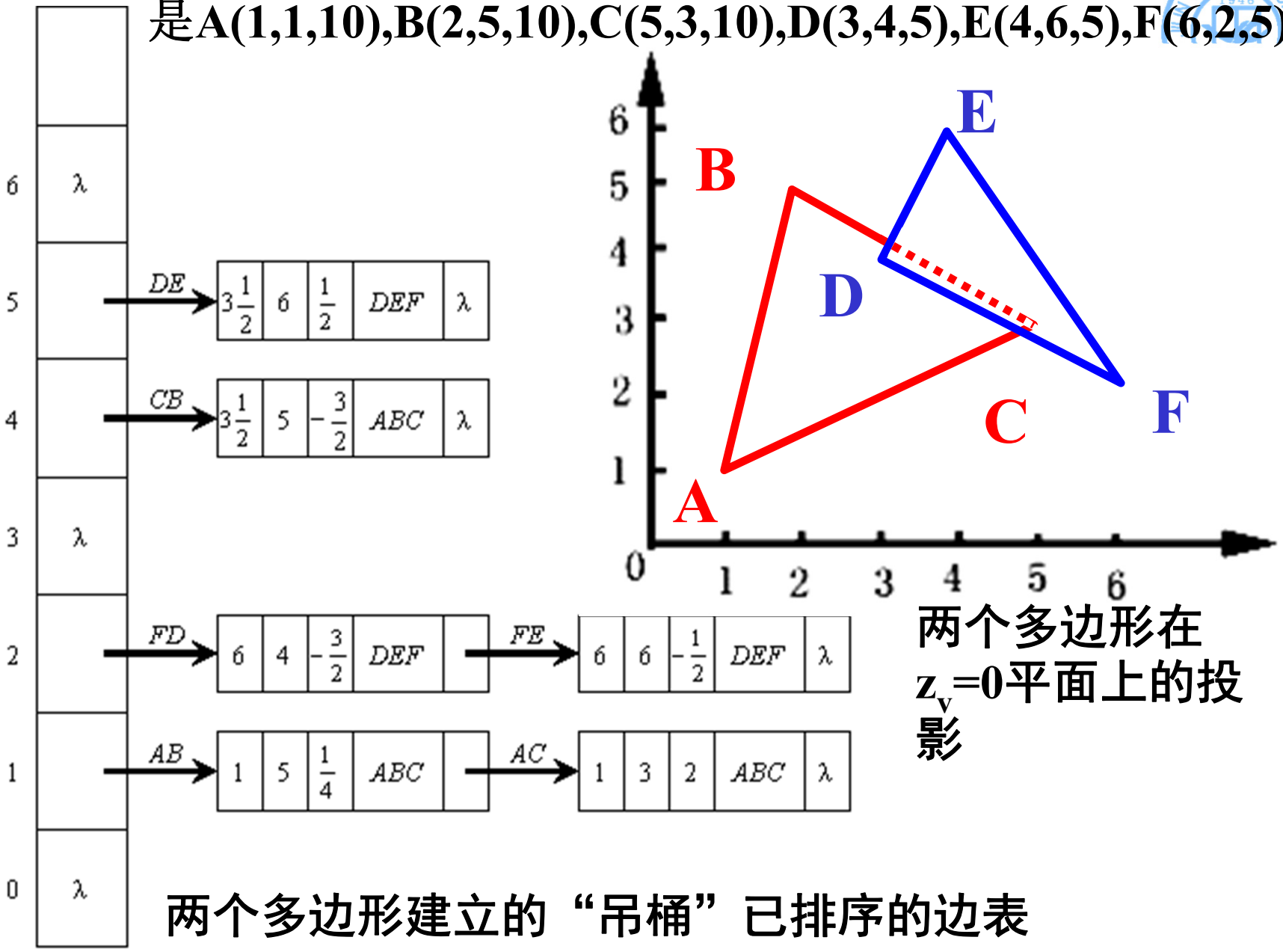


4. 对第i个和第i+1个“吊桶”存有的x坐标指示的扫描线上的一段，按照P表最前面多边形指示的亮度或颜色，实施扫描转换。

5. i增加1，若i所指已无“吊桶”，步骤结束，去下一步骤（删除y=ymax的边）。否则，回到步骤2。

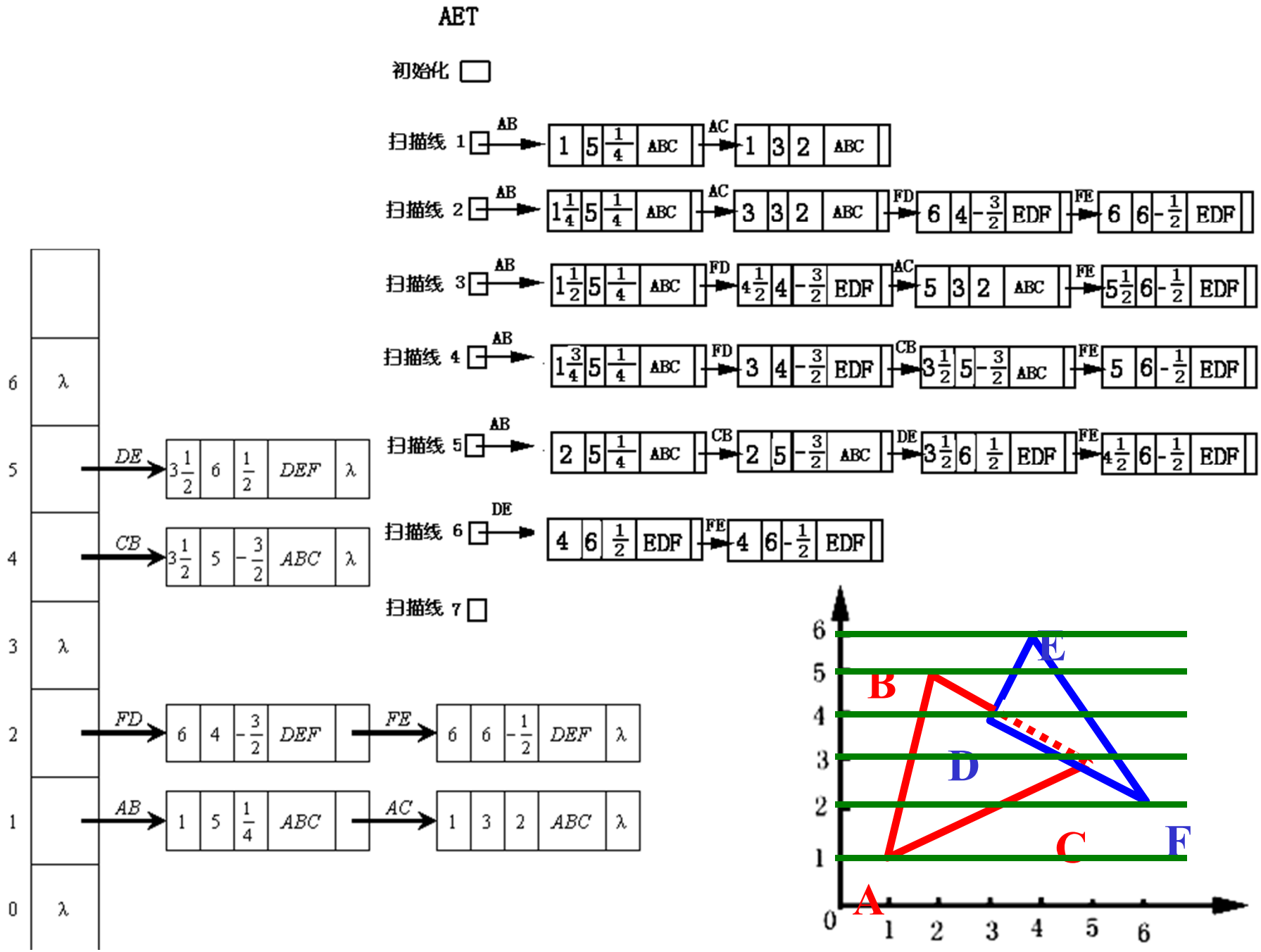


设有两个空间的三角形ABC、DEF,各顶点的坐标依次是A(1,1,10),B(2,5,10),C(5,3,10),D(3,4,5),E(4,6,5),F(6,2,5)。

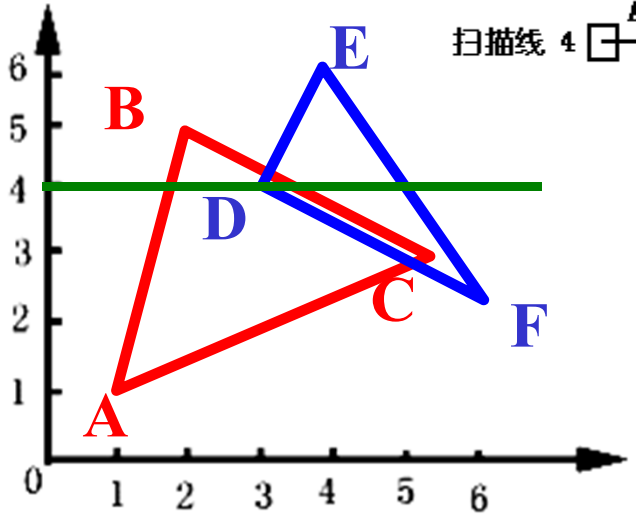


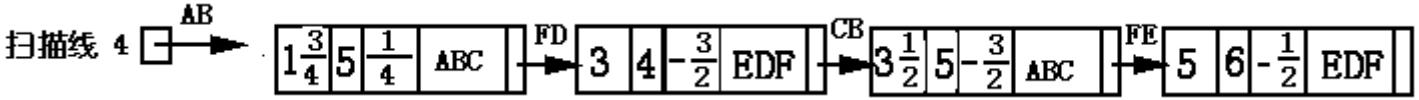
两个多边形在z_v=0平面上的投影

两个多边形建立的“吊桶”已排序的边表

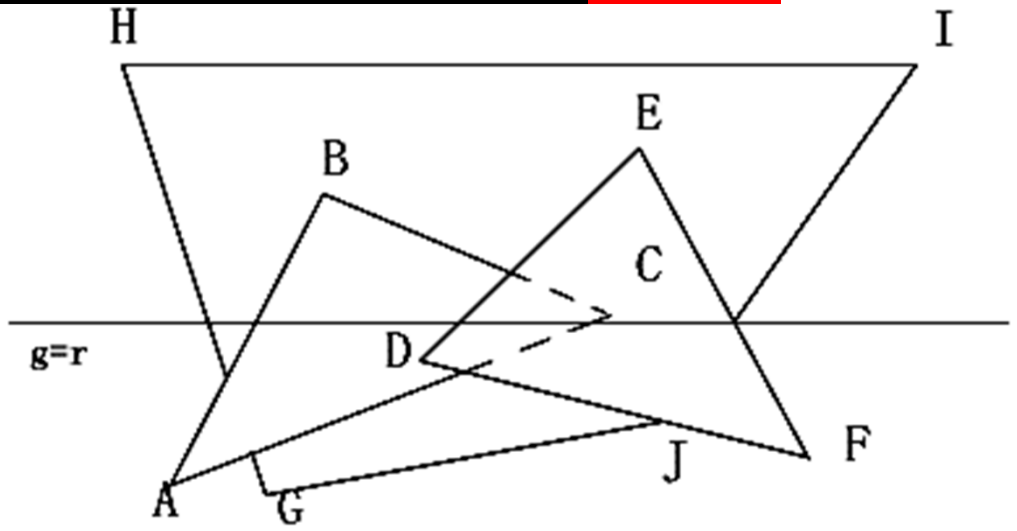

AET
初始化
扫描线 1
扫描线 2
扫描线 3
扫描线 4
扫描线 5
扫描线 6
扫描线 7
AB
AC
FD
FE
CB
DE
ABC
EDF
DEF
λ
6
5
4
3
2
1
0
A
B
C
D
E
F

i	s	P	PT表		说明
			F_{ABC}	F_{DEF}	
1	1	(ABC)	true		1¾ 到3填ABC的亮度或颜色值
2	2	(DEF, ABC)		true	在步骤3.2.3发生深度比较，比较结果DEF更靠近观察者，它仍在P表前面，3到3½ 填DEF的亮度或颜色值
3	1	(DEF)	false		3½到5填DEF的亮度或颜色值
4	0	( )		false	

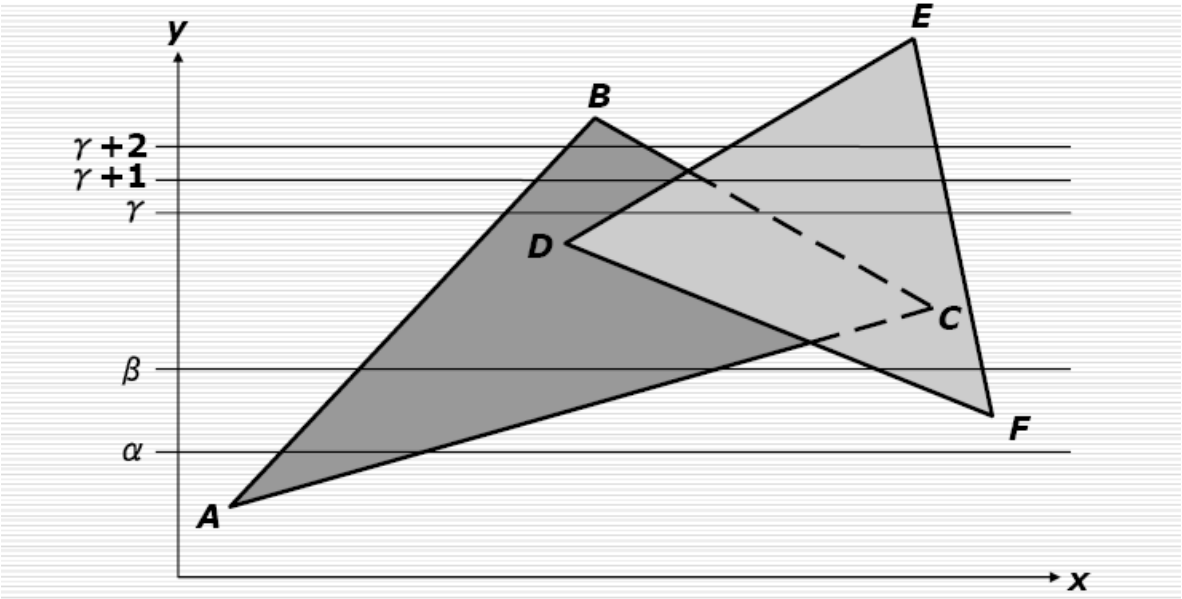


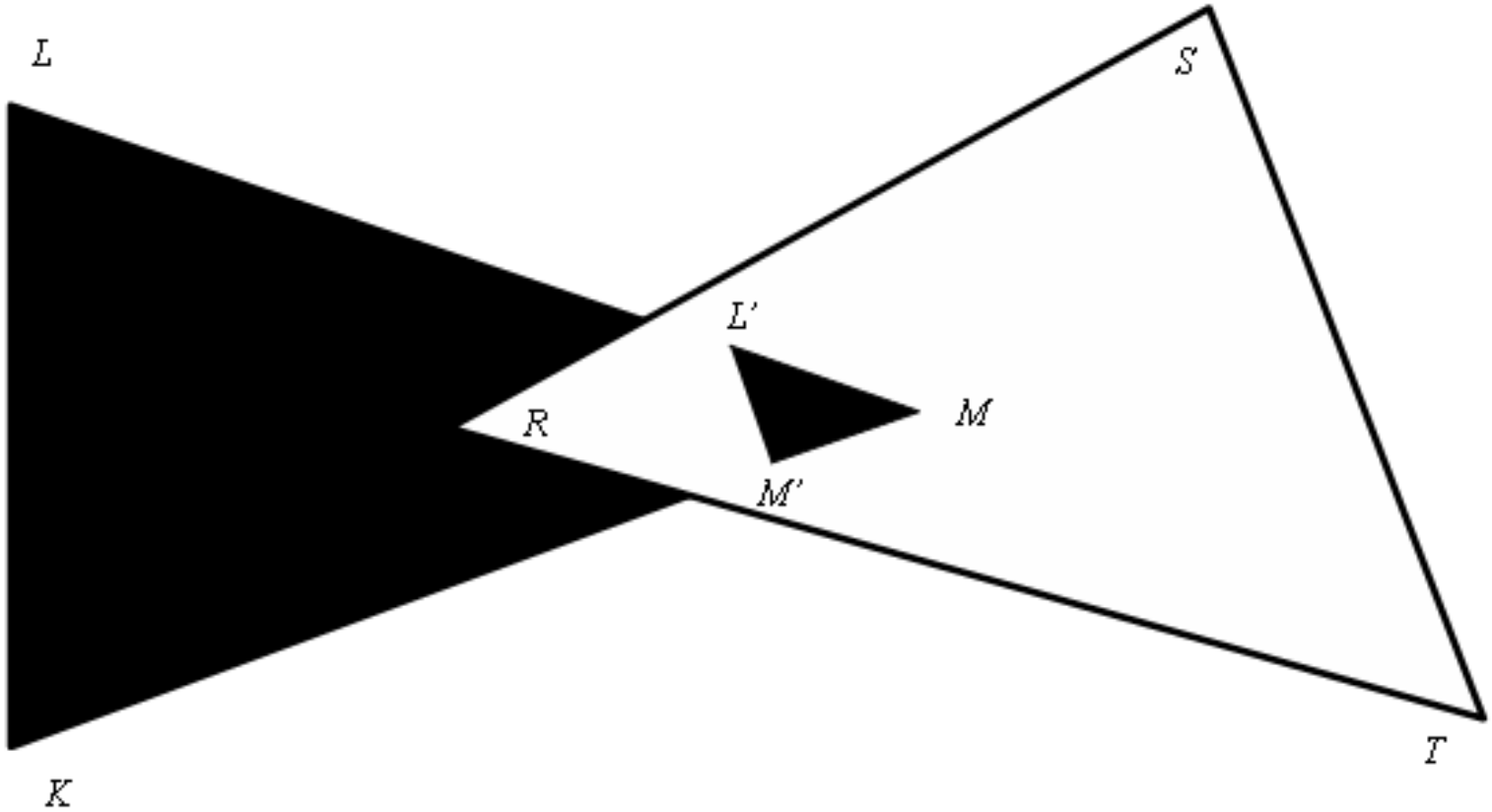


如果按照步骤3的条件决定是否做深度比较，会有许多深度比较是不必要的。例如，假定有一个大的多边形GHIJ在两个三角形ABC和DEF的后面。在扫描线y=γ离开边CB时，按步骤3的条件判断，扫描线还同时在DEF和GHIJ中，应该做深度比较，但是在大多数可以假定没有多边形穿透另一个多边形的情况下，DEF和GHIJ的深度关系并没有变化，深度比较是不必要的。如果扫描线从一个被遮挡的多边形中走出，深度比较将是不必要的；扫描线从一个遮挡了其它多边形的多边形中走出，深度比较才可能必要。



事实上前面给出的算法基本步骤没有很好地利用深度的相关性。深度相关性是指多个多边形之间的深度关系，常常对于一组相邻的扫描线来说是不变化的。如果在某条扫描线上，在AET表中保存的边及次序关系，与在前面一条扫描线上时完全相同，那么深度关系就不会变化，深度比较也就并不必要了。例如，图7.21中的扫描线γ到γ+1就是这种情形，在两条扫描线上AET表中的边及次序关系都是AB，DE，CB，FE。但从γ+1到γ+2，DE，CB的次序要变化为CB，DE，次序关系就变化了。利用深度的相关性可以对算法作出改进。





上图所示是一个多边形穿透了另一个多边形。为使算法能处理这种情况，应该把其中的多边形KLM分成两个多边形KLL'M'和L'MM'，加进了一条没有的边M'L'。

关于背景的处理，

1、把更新缓冲存储器整个初始化为某个合适的值，于是算法就只需处理扫描线在多边形内的情形。

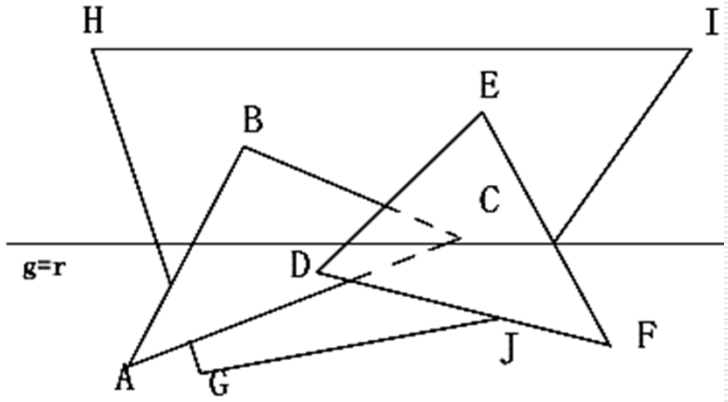
2、定义一个大的矩形，让它包含了客体中所有的多边形，位于比其它多边形都更远离观察者的平行于投影平面的一个平面上，并具有某个合适的亮度或颜色值。

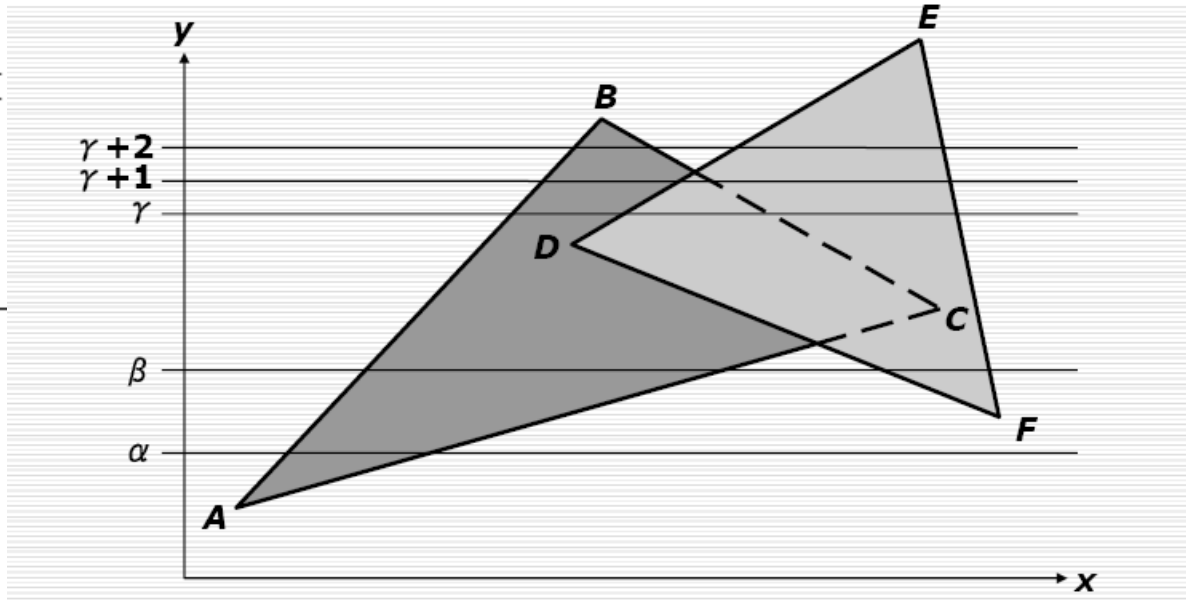
3、修改算法，使得扫描线不在任何多边形内时，就往帧缓冲存储器中送入背景的像素值。

3.2.3深度比较条件太宽松，实际很多不必要的的比较

改进1：假定没有多边形穿透另一个多边形的情况下，如果扫描线从一个被遮挡的多边形中走出，深度比较将是不必要的；扫描线从一个遮挡了其它多边形的多边形中走出，深度比较才可能必要。

改进2：利用深度相关性，对于一组相邻的扫描线来说，次序关系不发生变化，多边形之间的深度关系不发生变化的。





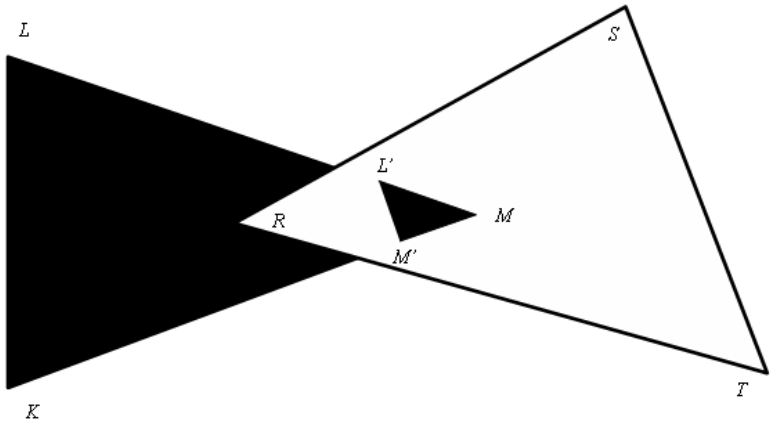
改进3：对于穿透情况的处理，将一个多边形分成两个子多边形

改进4：背景颜色处理

（1）帧缓存初始为某个特定的值

（2）定义一个包含了客体中所有的多边形，位于比其它多边形都更远离观察者的平行于投影平面的一个平面上，并具有某个合适的亮度或颜色值。

（3）修改算法。当扫描线不在任何多边形内时，就往帧缓冲存储器中送入背景的像素值。



第六节 区域分割算法

区域分割算法：

将投影平面分割成区域，考察区域内的图像。

1.如果容易决定在这个区域内某些多边形是可见的，显示那些可见的多边形，完成对这一区域的显示任务。否则，将区域再分割成小的区域，对小的区域递归地进行判断。

2.由于区域逐渐变小，在每个区域内的多边形逐渐变小，最终总可以判定哪些多边形是可见的。

该算法利用了区域的相关性，即位于适当大小的区域内的所有像素，表示的其实是同一个表面。

区域分割算法：图像空间算法

区域的相关性:是指位于适当大小的区域内的所有像素，表示的其实是同一个表面。

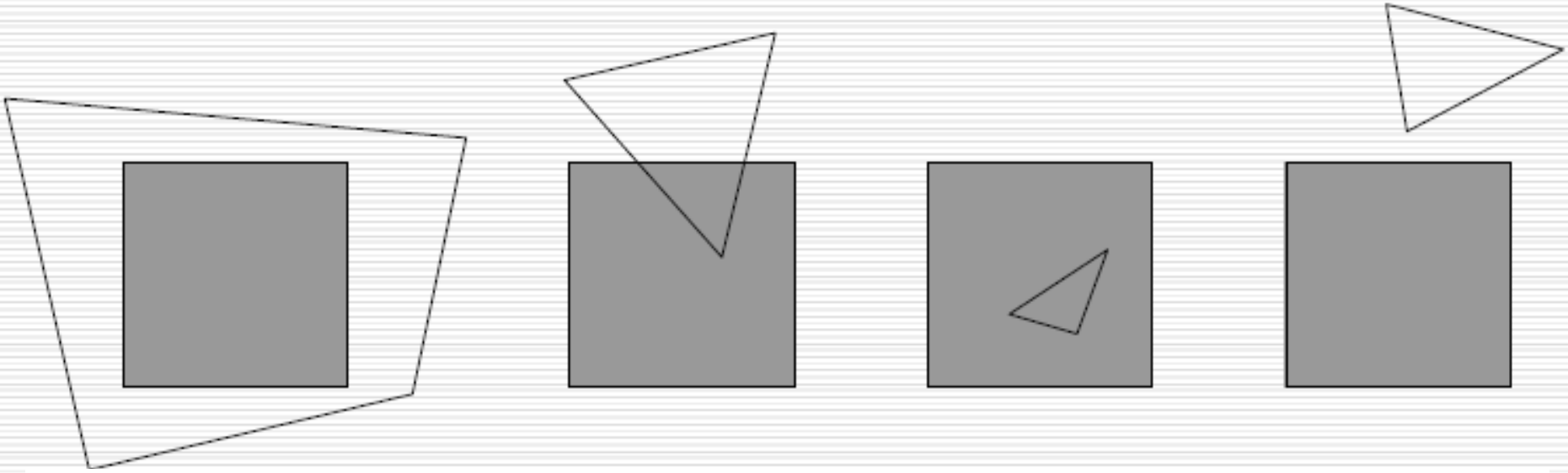
在递归分割的每一步，要显示客体中每个多边形的投影多边形与所考察区域之间的关系，必然是下列四种之一：

1.包围的多边形，即多边形包围了所考察的区域。

2.相交的多边形，即多边形所考察的区域相交。

3.被包含的多边形，即多边形全部在所考察的区域之内。

4.分离的多边形，即多边形与所考察的区域完全分离。



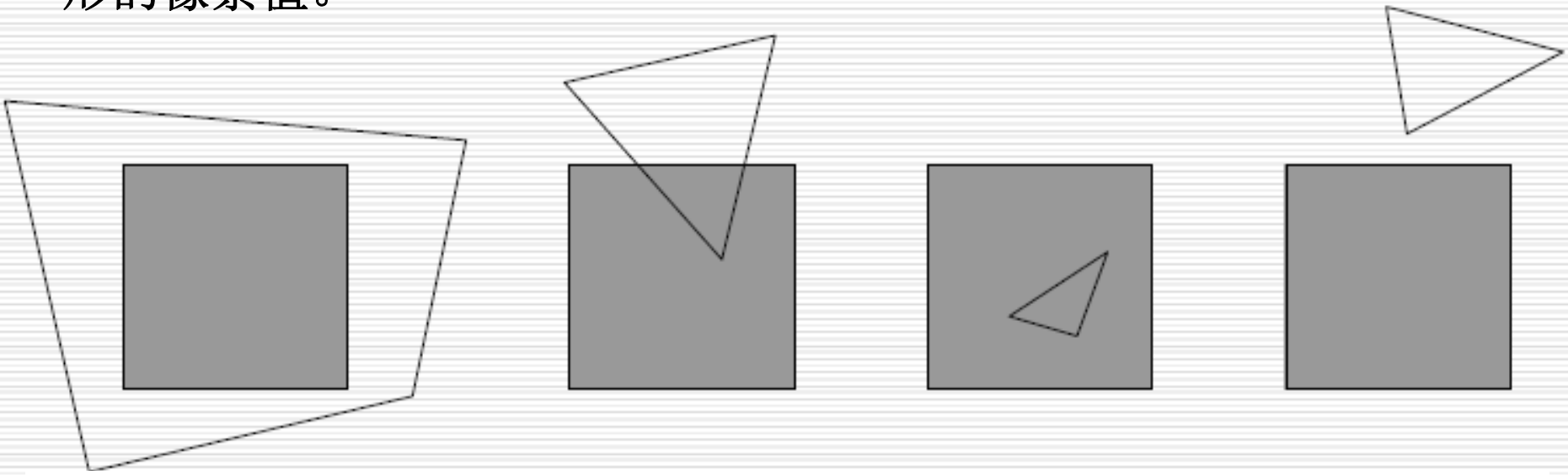
包围的　　相交的　　被包含的　　分离的

若区域与多边形为下面的四种情况，不必再做进一步的分割，可直接绘制。

1．所有的多边形与区域分离，所以在区域内只需显示背景值。

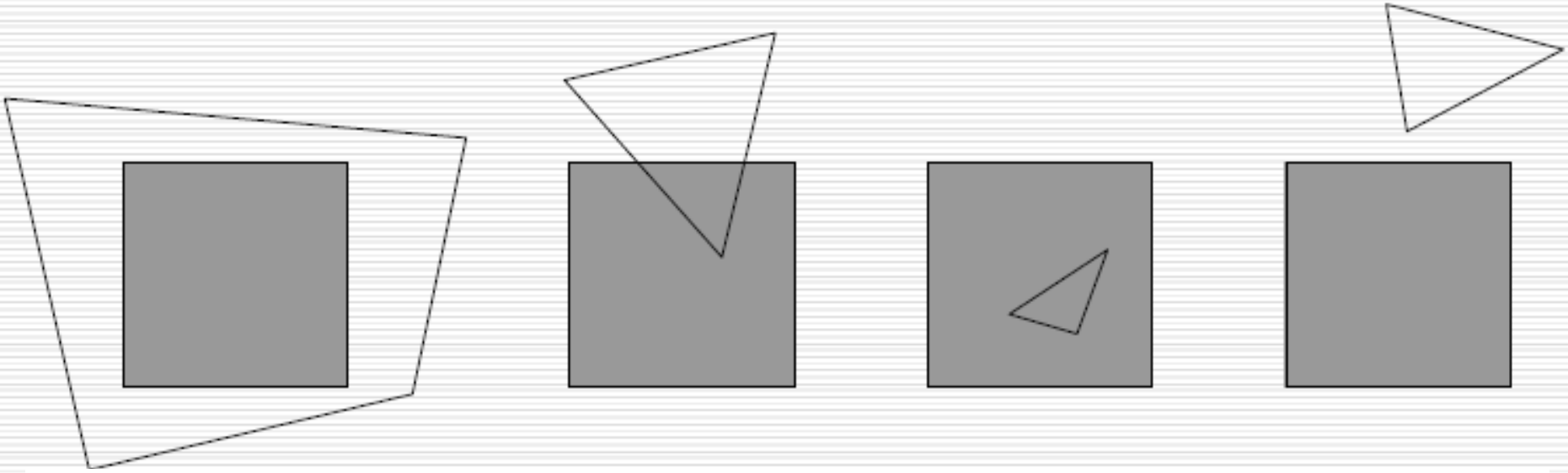
2．只有一个相交的多边形，或者只有一个被包含的多边形。这时可以对区域首先填充背景值，然后对多边形进行扫描转换。在某些显示设备上，将整个帧缓冲存储器都初始化为背景值，可能更为方便。对于相交的多边形，只是被包含的部分被扫描转换。

3．只有一个包围的多边形，无其它的多边形。整个区域填充该多边形的像素值。



包围的　　相交的　　被包含的　　分离的

4．有多于一个的包围的、相交的或被包围的多边形，且至少有一个包围的多边形。检查是否能有一个包围的多边形，它位于所有其它多边形的前面。如果有，就可以让整个区域都填充为这个多边形的像素值。具体的检查方法是，对所有的多边形，计算其所在平面在区域的四个角点的应有深度，即相应的z坐标，如果有一个包围的多边形对应的四个z坐标，都小于其它多边形的对应z坐标，那么这个包围的多边形就位于所有其它多边形的前面。



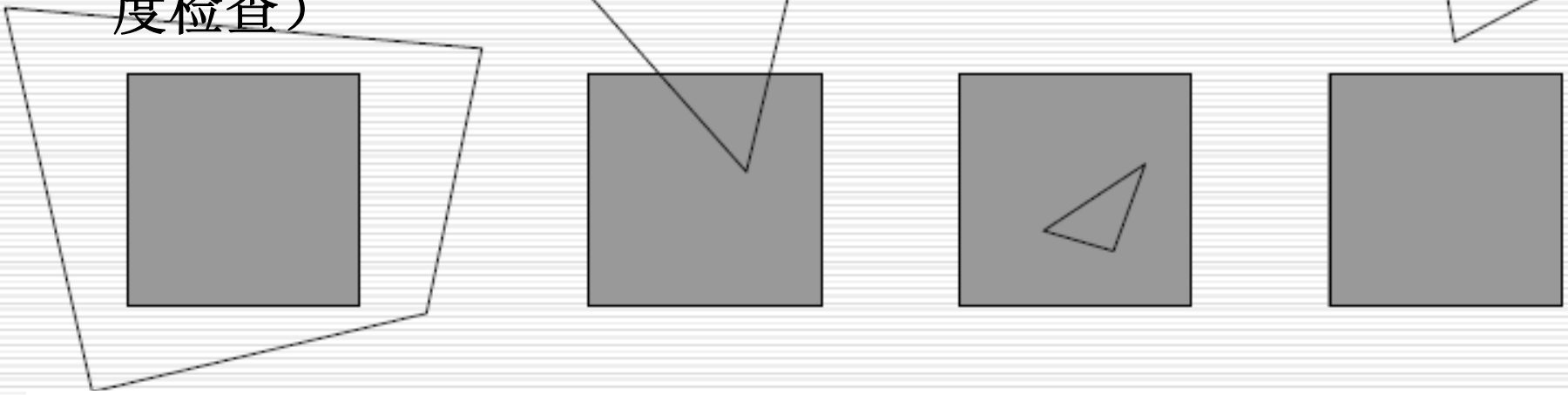
包围的　　相交的　　被包含的　　分离的

若区域与多边形为下面的四种情况，不必再做进一步的分割，可直接绘制：

1. 所有的多边形与区域分离，所以在区域内只需显示背景值。
2. 只有一个相交的多边形，或者只有一个被包含的多边形。这时可以对区域首先填充背景值，然后对多边形进行扫描转换。（相交的多边形，只对被包含的部分做扫描转换）
3. 只有一个包围的多边形，无其它的多边形。整个区域填充该多边形的像素值。

（1,2,3用范围检查）

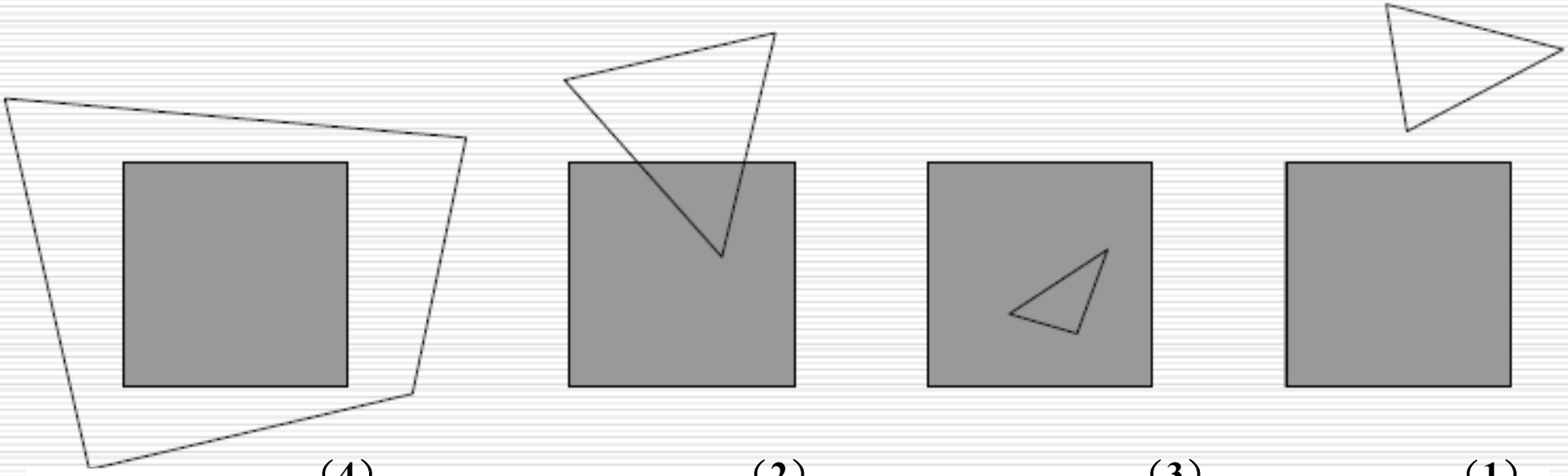
4. 多个多边形与之相交、包围、或被包含，且有一个包围的多边形位于其它多边形最前面，区域填充为该多边形的颜色。（深度检查）



包围的　　相交的　　被包含的　　分离的

区域经过分割变小以后，只需要考虑被包含的多边形和相交的多边形的变化。

因为分离的或包围的多边形，对变小的区域，仍然保持是分离的或包围的。分割达到显示设备的分辨能力之后就可以停止，即最小的区域可以是显示表面上的一个像素单位。如果在做了最大数目的分割之后，仍然不能确定该如何填充，那么，就计算所有相关的多边形在这个不可再分区域对应点的范围的中心z坐标值，取z坐标最小的多边形像素值填充这



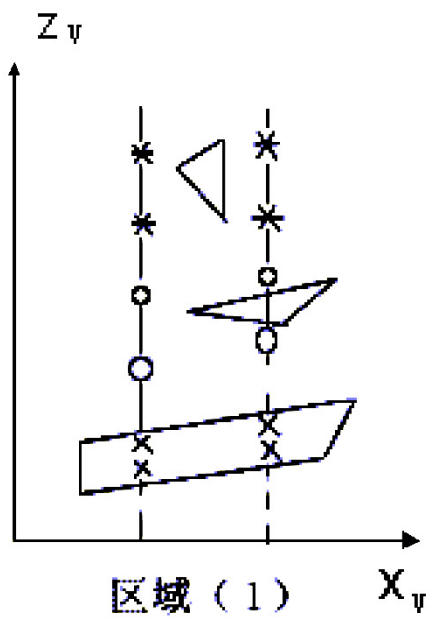
（4） （2） （3） （1）

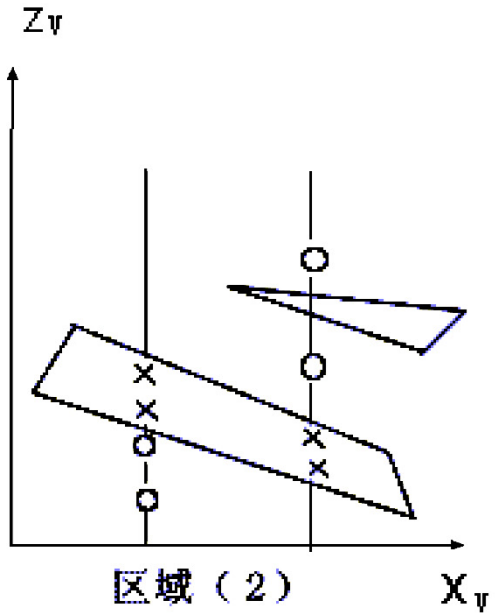
包围的 相交的 被包含的 分离的

如何判断包围的多边形位于其它多边形最前面：

1. 对所有的多边形，计算其所在平面在区域的四个角点的应有深度，即相应的z坐标，如果有一个包围的多边形的对应四个z坐标，都小于其它多边形的对应z坐标，那么这个包围的多边形就位于所有其它多边形的前面。

2. 判别相交的多边形是否在整个包围多边形的后面，将相交多边形的顶点坐标代入包围多边形的平面方程，判别符号。





X--表示包围的多边形平面交点

*—表示被包围的多边形平面交点

0--表示相交的多边形平面交点

区域分割成子区域：

1.子区域需考察的情况：

只需考虑父区域被包含的或相交的多边形。

对于分离的或包围的多边形，进行区域分割之后仍保持分离包围的关系。

2.分割中止的条件：一个像素单位或最大数目的分割

对于仍不能判断的，则找出中心点，计算中心点对应的所有多边形的z值，取z值最小的多边形像素值来填充该区域。

3.分割方法

（1）等分

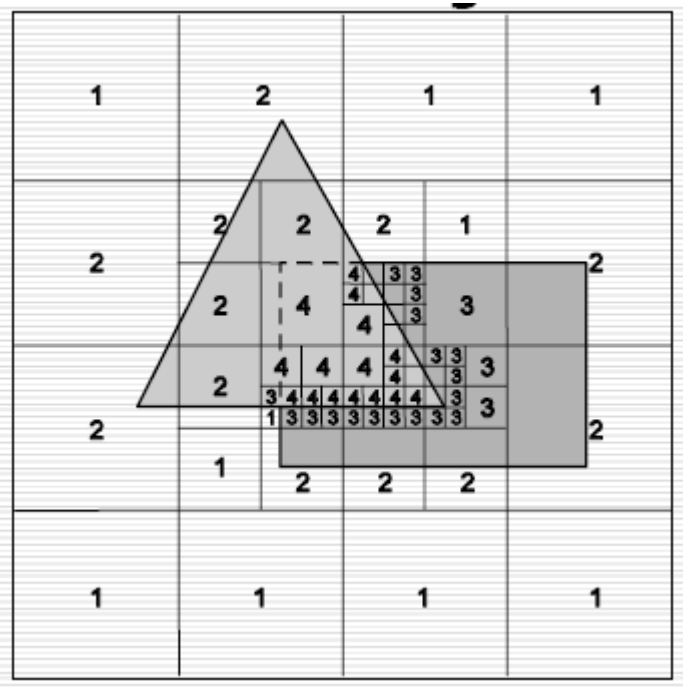
（2）按多边形顶点做分割

（3）按多边形投影进行分割

Wanock首先提出的最初的区域分割算法是每次把区域分成四个正方形。

下图做了**5**次区域分割的情形，其中区域内标出的数字，表示前面所说的四种情形中的哪一种，可以作出决定。没有标出数字的区域是还不能做出决定。

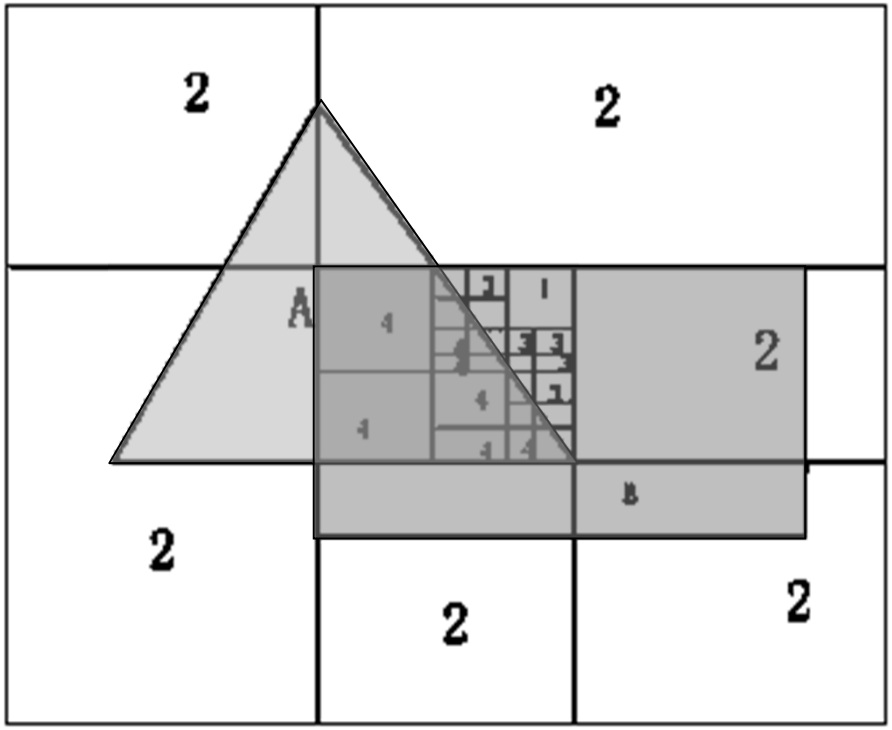
（1）等分：分割区域为正方形



1. 所有的多边形与区域分离。
2. 只有一个相交的多边形，或者只有一个被包含的多边形。
3. 只有一个包围的多边形，无其它的多边形。
4. 多个多边形与之相交、包围、或被包含，且有一个包围的多边形位于其它多边形最前面

（2）按照顶点位置来做分割(先A后B)

区域分割不一定总是等分，当区域内有多边形的顶点时，可以按照顶点位置来做分割，这样可以少做一些分割。



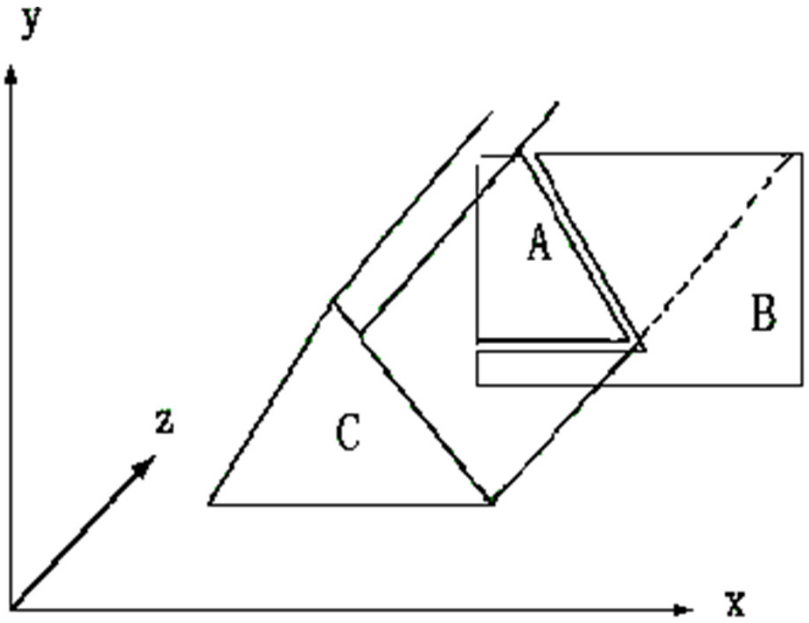
围绕多边形顶点分割（先是A，后是B）

1. 所有的多边形与区域分离。
2. 只有一个相交的多边形，或者只有一个被包含的多边形。
3. 只有一个包围的多边形，无其它的多边形。
4. 多个多边形与之相交、包围、或被包含，且有一个包围的多边形位于其它多边形最前面

（3）按照多边形投影选择区域做分割

区域分割还可以按照客体中多边形投影的范围进行，这可以少做很多分割。

Weiler和Atherton提出的算法，直接就用多边形的投影做为分割的区域。选择用做分割的区域时，可以按照多边形各顶点坐标最小值的递增次序



按照多边形投影选择区域做分割

（3）按照多边形投影选择区域做分割

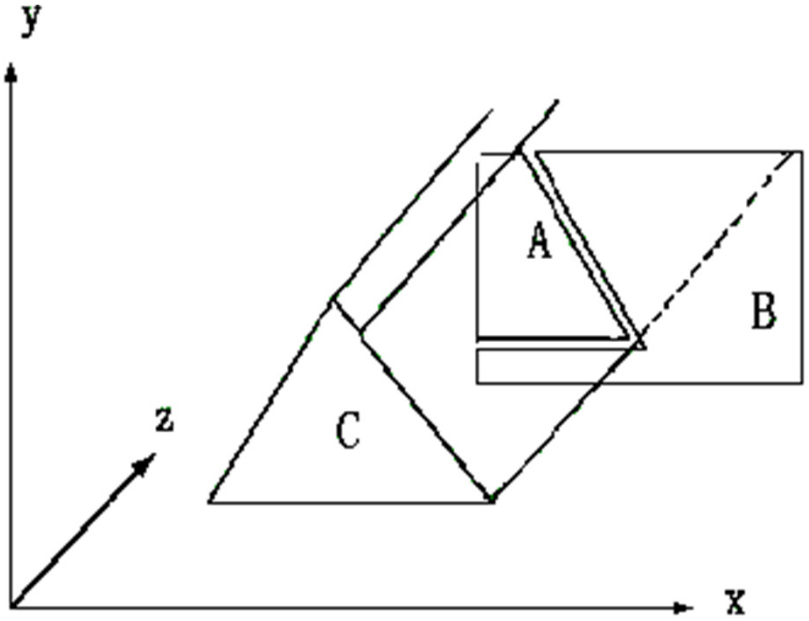
按深度进行预排序。

用距视点最近的多边形对所有多边形进行裁剪和区域分类。

删去位于离视点最近的多边形之后的其他多边形。

必要时递归地进行分割。

最后用深度排序消除所有不确定性。

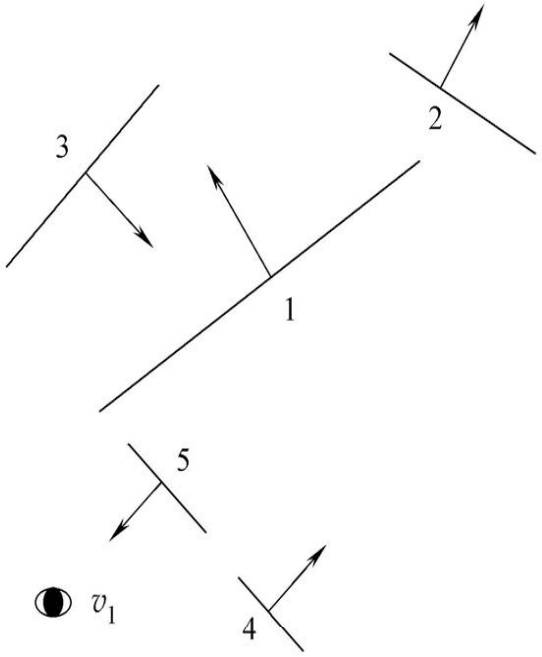


按照多边形投影选择区域做分割

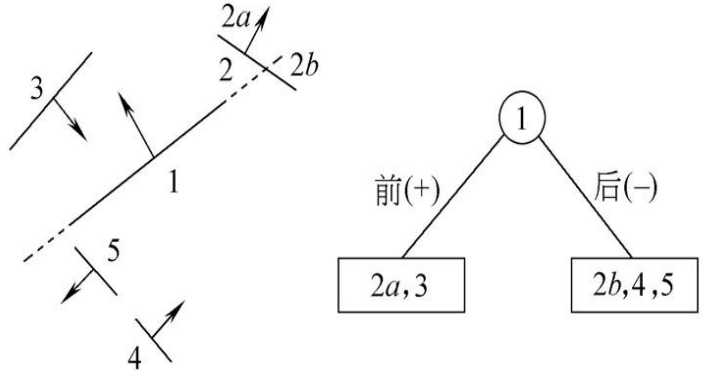
第七节 BSP树算法

- 二叉空间剖分(Binary Space-Partitioning,BSP)树算法是一种判别物体可见性的有效算法。它类似于深度排序算法,将表面由后往前地在屏幕上绘出。该算法特别适用于场景中位置固定不变,仅视点移动的情况。

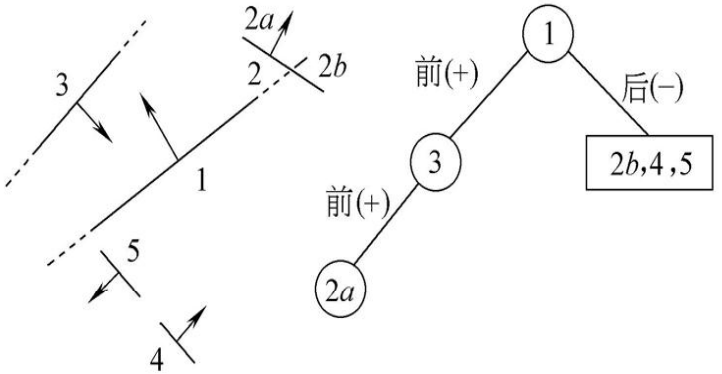
- 一、构造BSP树
- BSP树取场景中的一个多边形作为分割面,递归地将空间分为两个子空间。场景中的其他多边形中,完全位于剖分平面某一侧的多边形被归入相应的子空间,与剖分面相交的多边形沿剖分面被分割成两部分,分别放入相应的子空间。这两个子空间再分别选一个多边形作分割面递归地子分下去,直到每个子空间只剩一个多边形为止。上述划分过程可用二叉树方便地表示出来,二叉树的根为最初被选作分割面的多边形。BSP树的构造在景物空间内完成。


3
2
1
5
v_1
4

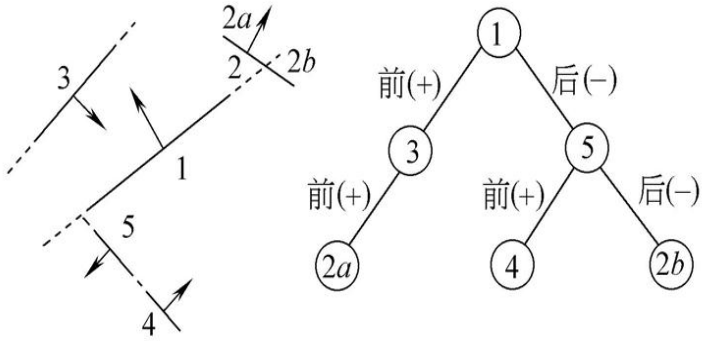
a)


2a
2
2b
3
1
5
4
1
前(+)
后(−)
2a,3
2b,4,5

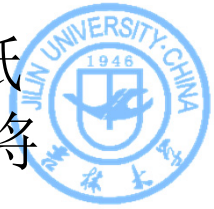
b)


2a
2
2b
3
1
5
4
1
前(+)
后(−)
3
前(+)
2a
2b,4,5

c)

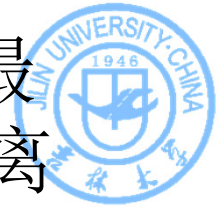

2a
2
2b
3
1
5
4
1
前(+)
后(−)
3
5
前(+)
前(+)
后(−)
2a
4
2b

d)



- 为简单起见,这里假设每个多边形和分割面都垂直于纸面。先取多边形1作为分割面,分割面与多边形2相交,将其分为多边形2*a*和2*b*,多边形2*a*和3在分割面的前面,多边形2*b*、4和5在分割面的后面,见图7-27中BSP树生成的中间过程。然后从“树”枝中选取多边形3作为分割面,其“前”枝子空间包含多边形2*a*,“后”枝子空间为空。至此,树的这一分枝分割完毕,返回到根,处理树的“后”枝。取多边形5作为分割面,分割后,多边形4归入其前枝,多边形2*b*归入其后枝。因为每一子空间都仅包含了一个多边形,故这个分枝也分割完毕。实际上,多边形4和多边形5是共面的,在这种情况下,可放入任一分枝。
- 因为初始分割面和后面每一子空间的分割面的选取是任意的,表示同一场景的BSP树并不唯一。

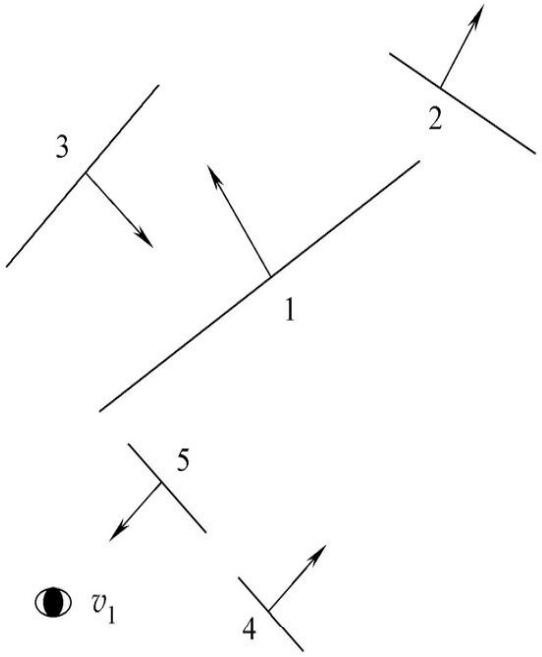
- 二、BSP树遍历
- BSP树算法的一个明显优势在于BSP树不依赖于视点的位置,其显示算法在图像空间执行。依据BSP树确定可见面时,仅需知道视点与树的根结点多边形的相对位置关系即可。对树的中序遍历算法稍加修改就可确定可见面,即先遍历一子树,访问根结点、再遍历其另一子树。



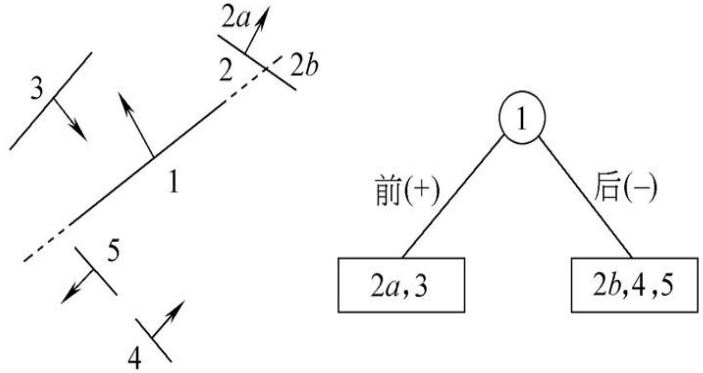
- BSP树可见面算法的基本思想是,先将离视点最远的多边形写入帧缓冲或进行显示,然后再将离视点较近的多边形写入帧缓冲,即按从后到前的顺序对多边形绘制。这有两种情况,如果视点在根结点多边形的前面,则BSP树可按下面顺序遍历:
- 1)后枝(所有位于根结点的后半空间的多边形)。
- 2)根(结点)多边形。
- 3)前枝(所有位于根结点的前半空间的多边形)。

- 如果视点在根结点多边形的后面,则按下面顺序遍历:
- 1)前枝(所有位于根结点的前半空间的多边形)。
- 2)根(结点)多边形。
- 3)后枝(所有位于根结点的后半空间的多边形)。

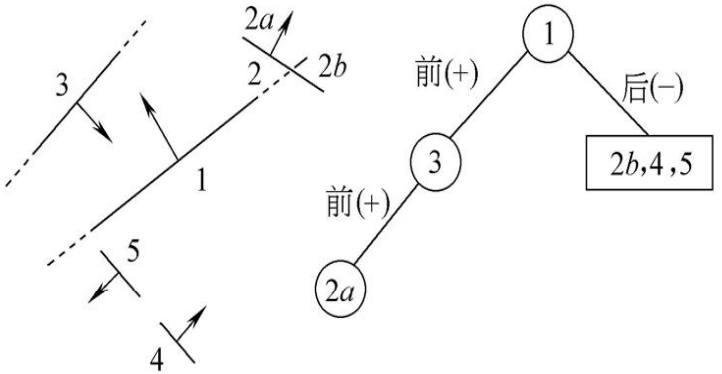
- 如果把不包含视点的根结点半空间称为远半空间,包含视点的半空间称为近半空间,则这两种遍历情况可统一表示为:
- 1)远分枝。
- 2)根(结点)多边形。
- 3)近分枝。


3
2
1
5
v_1
4

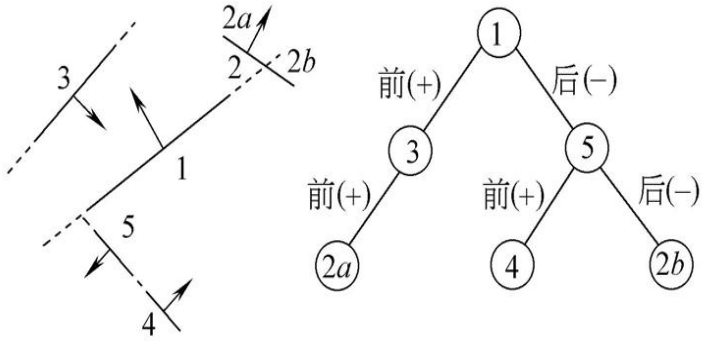
a)


2a
2
2b
3
1
5
4
1
前(+)
后(−)
2a,3
2b,4,5

b)


2a
2
2b
3
1
5
4
1
前(+)
后(−)
3
前(+)
2a
2b,4,5

c)


2a
2
2b
3
1
5
4
1
前(+)
后(−)
3
5
前(+)
前(+)
后(−)
2a
4
2b

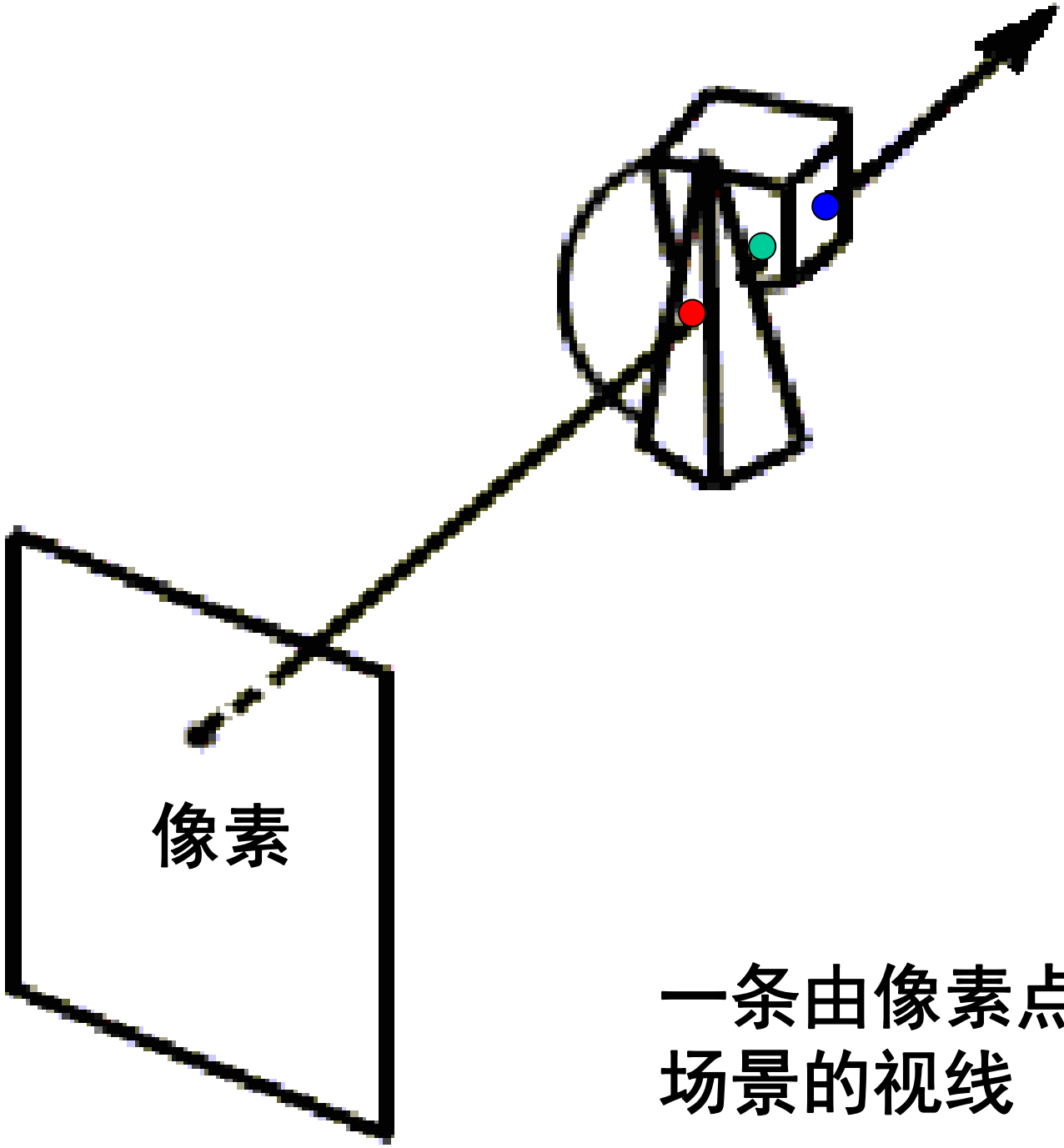
d)

- 设视点v_1在根结点多边形后侧,因此BSP树遍历顺序为前枝→根→后枝。在多边形3处递归地应用该算法,这时,v_1在多边形3前侧,该分枝的遍历顺序为后枝→根→前枝,由于该结点无后枝,故直接显示根多边形3。再遍历前枝,它只包含叶结点多边形2*a*,直接显示2*a*。至此,前枝显示完毕,返回到根,显示根多边形1。
- 遍历根结点1的后枝,子树的根为多边形5,视点位于其前侧,因此按后枝→根→前枝的顺序遍历该子树。多边形2*b*是叶结点多边形,可直接显示。接着显示子树的根多边形5,最后是子树的前枝。多边形4是叶结点多边形,可直接显示。至此,场景全部显示完毕。最终的显示顺序依次为3、2*a*、1、2*b*、5、4。

第八节 光线投射算法

考察由视点出发穿过观察平面上一像素而射入场景的一条射线，则可以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，离像素最近的交点所在的表面即为可见面。这种可见性判别模式应用了光线投射算法。

光线投射（**ray casting**）建立于几何光学的基础之上，它沿光线的路径追踪可见面，是一种有效的可见性判别手段。由于场景中有无限多条光线，而我们仅对穿过像素的光线感兴趣，因此可考虑从像素出发，逆向跟踪射入场景的光线路径。光线投射算法对于包含曲面，特别是球面的场景有很高的效率 。



一条由像素点射入场景的视线

光线投射算法可以看做是深度缓冲器算法的一种变形。在深度缓冲器算法，每次处理一个表面并对表面上的每个投影点计算深度值。计算出来的值与以前保存的深度进行比较，从而确定每个像素所对应的可见表面。在光线投射算法中，每次处理一个像素，并沿光线的投射路径计算出该像素所对应的所有表面的深度值。

光线跟踪技术通过追踪多条光线在场景中的路径，以得到多个物体表面所产生的反射和折射效果。而在光线投射中，跟踪的光线仅从每个像素到最近的物体为止。

在视图平面上选择投影中心和窗口

```
for(图像中的每条扫描线){
    for(扫描线上的每个象素){
        确定从投影中心穿过象素的线;
        for(场景中的每个对象){
            if(对象被交并且最近)
                记录交线和对象名;
        }
    }
}
```