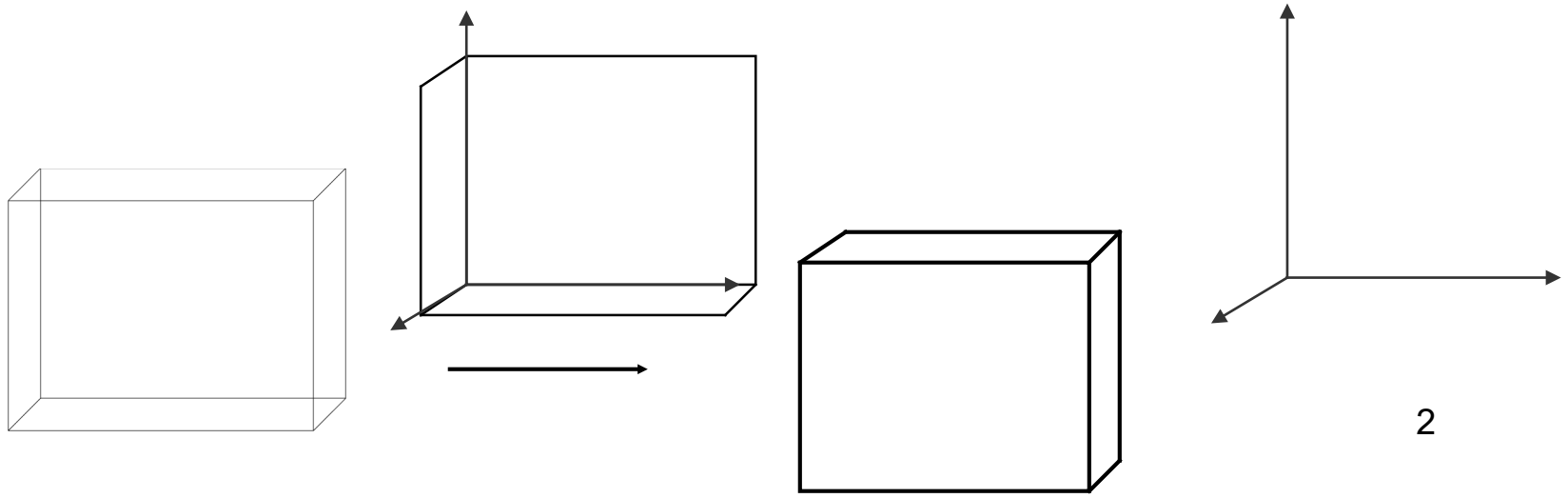
消　　隐

隐藏线消除



可见面确定/隐藏面消除

- 给定一组 3D 对象和观察参数，确定沿投影方向观察时可见的对象部分，或者，等效地，消除隐藏部分（隐藏的线和表面）。
- 可见部分将以适当的颜色和阴影绘制/显示

消隐方法

两大类

- 物体空间法
- 图像空间法

物体空间法

for （场景中的每个物体) {

确定物体的哪些部分未被其他物体遮挡;

将这些部分画上合适的颜色;

}

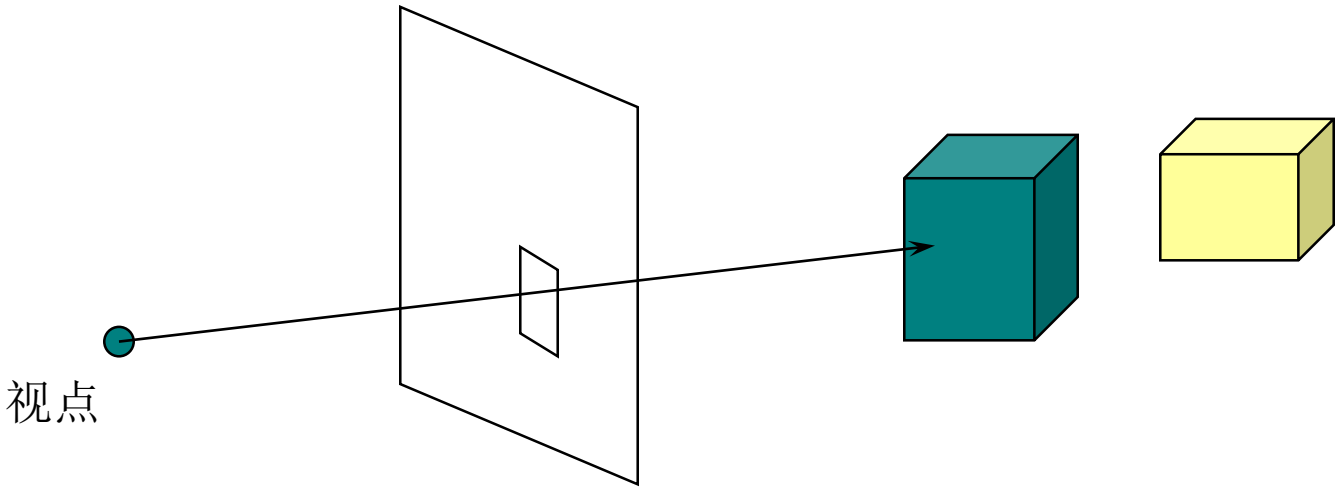
图像空间法

for (图像的每个像素) {

确定从视点穿过这个像素的射线与场景中的哪个物体最先相交;

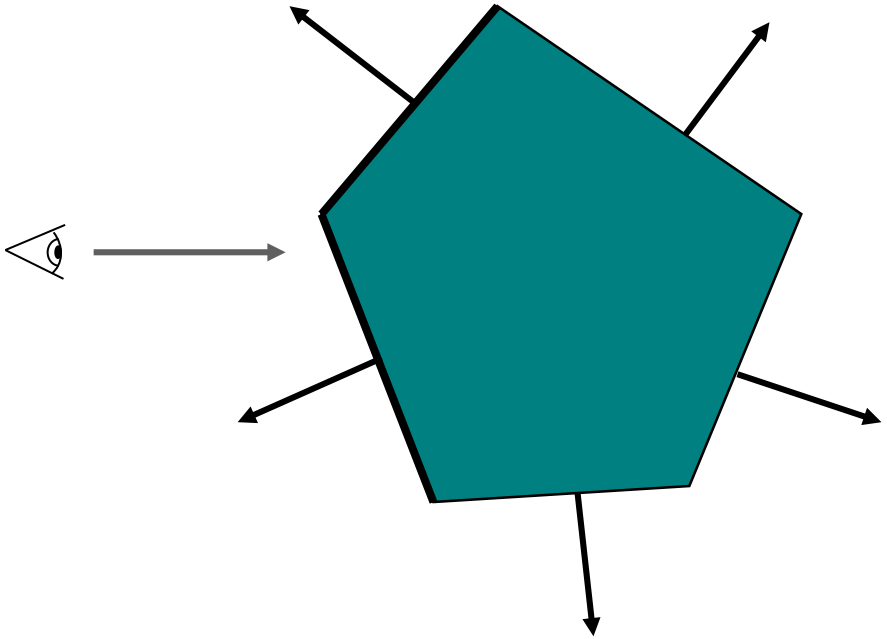
将这个像素填上合适的颜色;

}



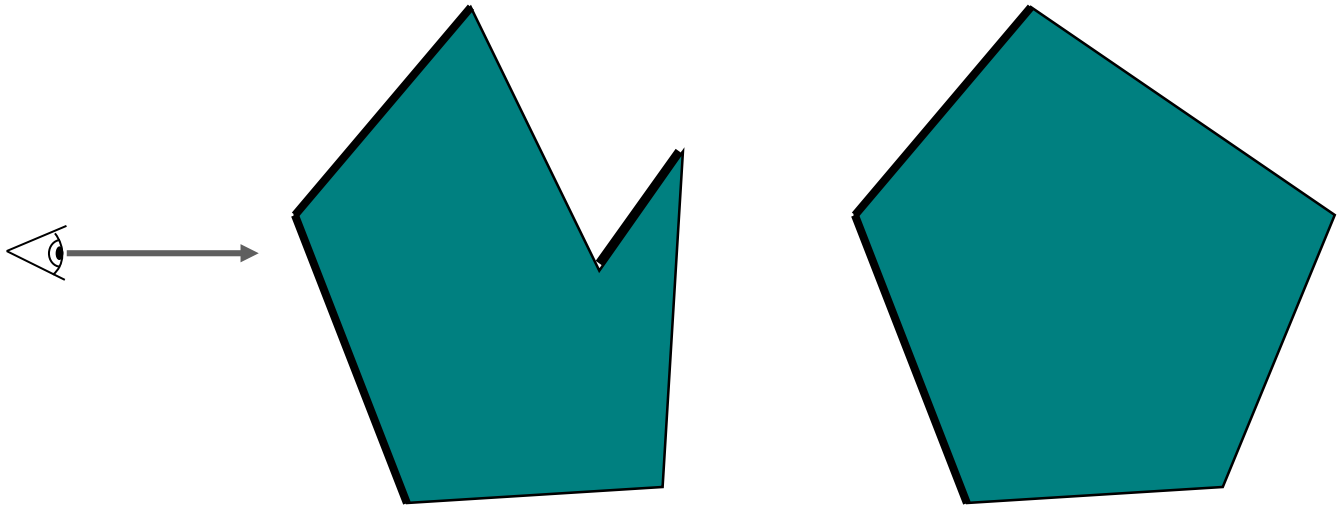
背面剔除

- 外法线方向与视线方向夹角小于90度的面不可见，称为背面，可先行剔除。

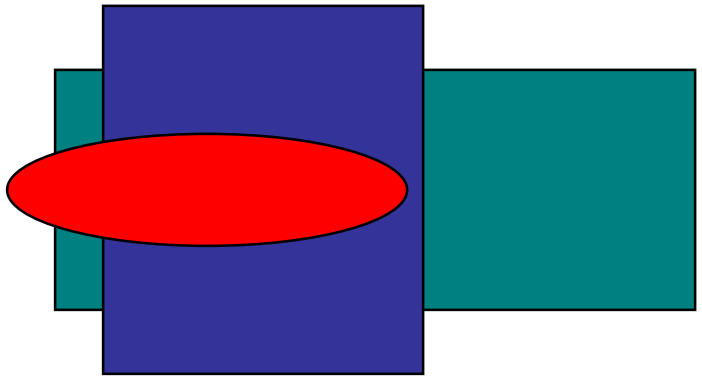


背面剔除

- 背面剔除只能消除部分不可见面.



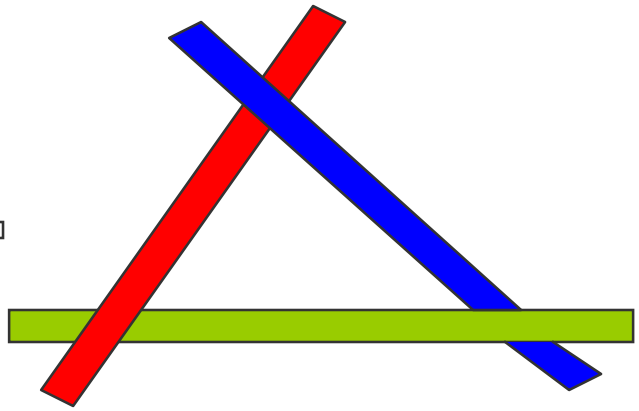
隐藏面消除



画家算法：

从后往前画

??

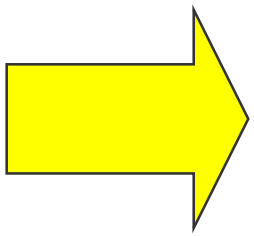


无法排序的情况

裁剪

区域排序

裁剪及区域排序
计算量极大



Z-buffer算法

扫描线算法

Warnock算法：

Warnock 算法
(图像空间法)

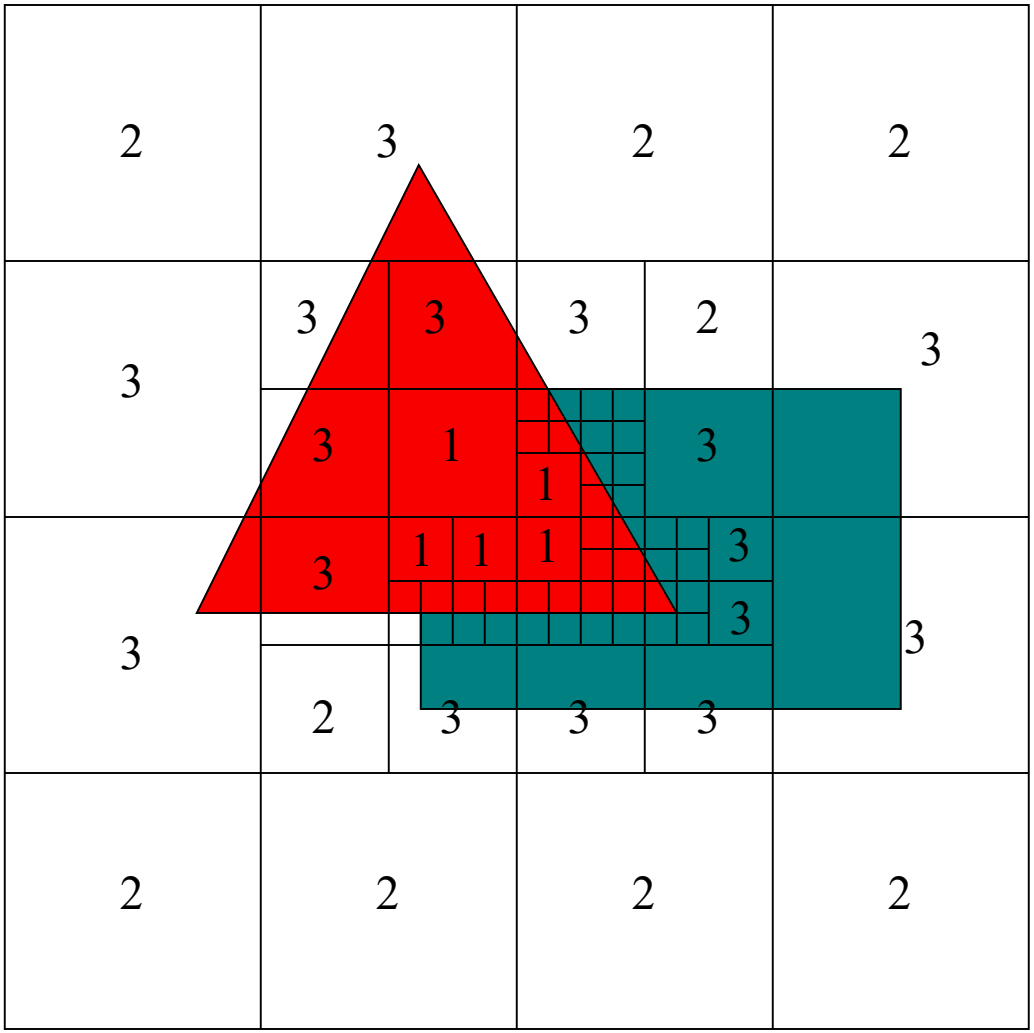
从整个窗口开始

•如果窗口内的多边形可以排序，则排序后从后往前画。

•否则，将区域细分为 4 个窗口并递归前述过程

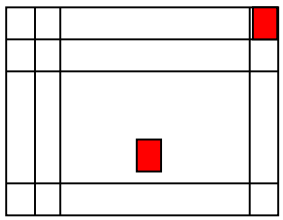
问题： 测试窗口内的多边形是否可以排序比较困难：简单快速的方法准确性不高，引起过度细分；准确性好的方法相当复杂和缓慢。

Warnock's 算法

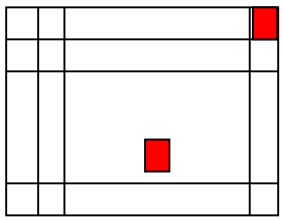


Z-Buffer 算法

- 图像空间算法
- 除了存放颜色的frame buffer外，还需要一个同样大小z-buffer，用于存放z 值

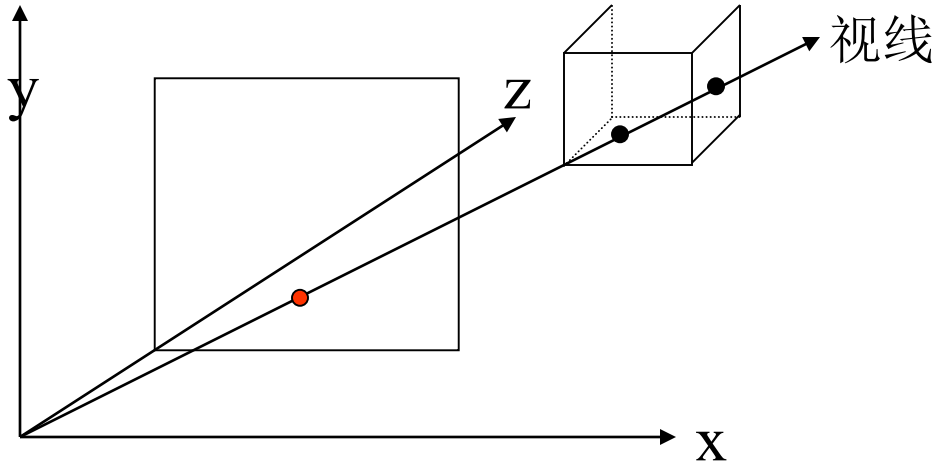


F — Buffer

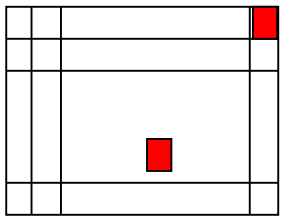


Z—Buffer

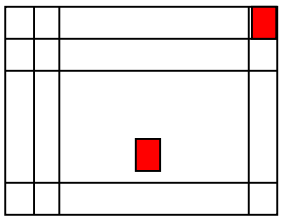
Z-Buffer



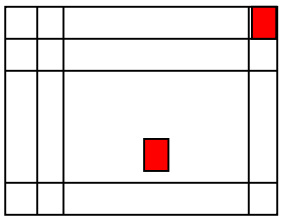
视点



屏幕

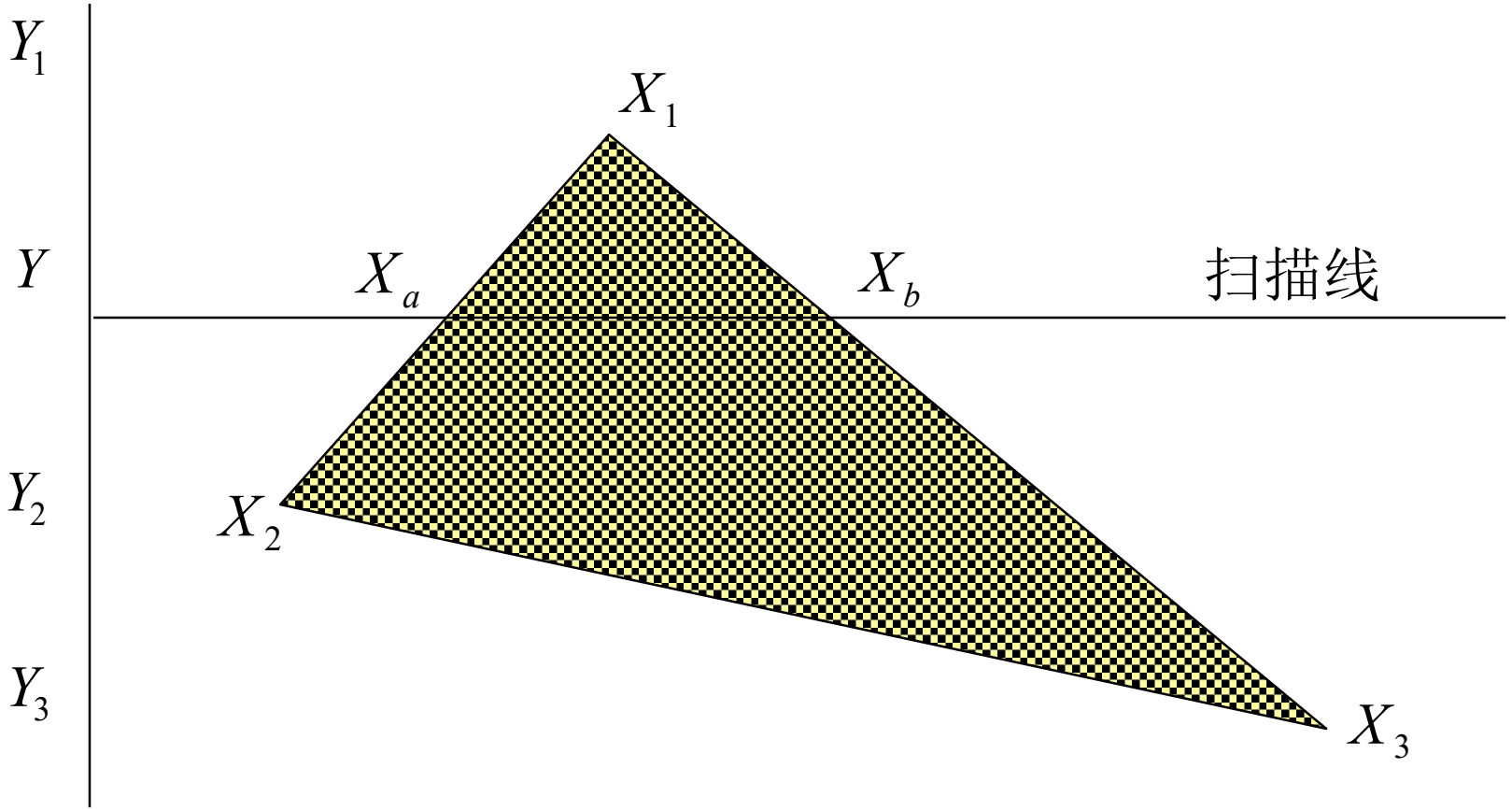


F — Buffer



Z—Buffer

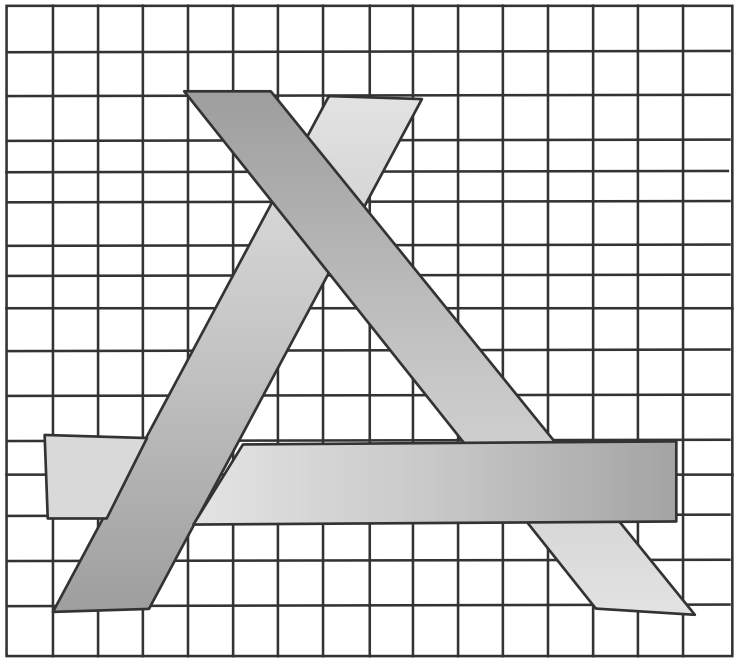
多边形扫描转换

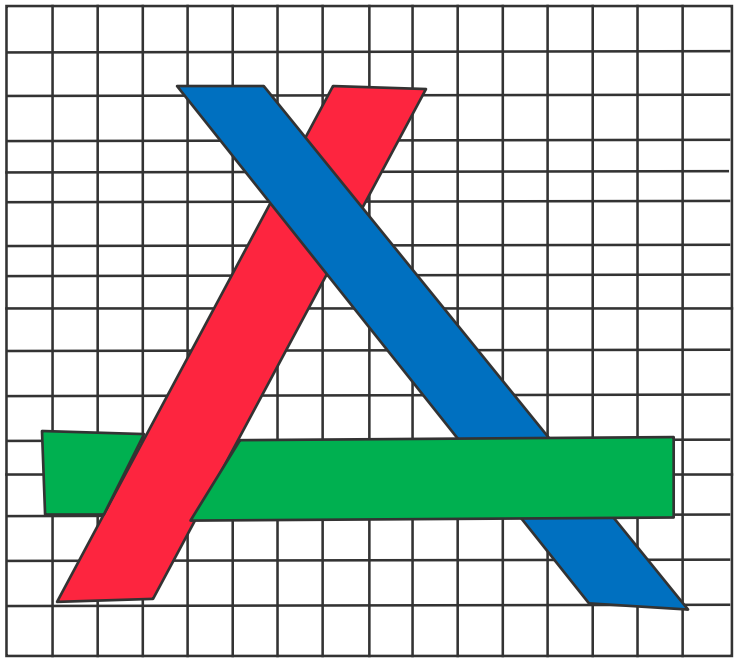


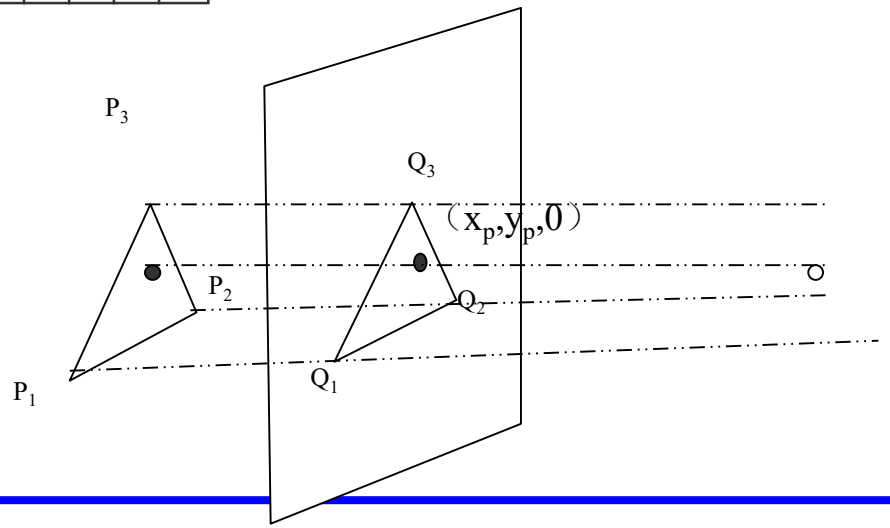
Z-Buffer 伪代码

```
for ( j=0; j<SCREEN_HEIGHT; j++ )
    for ( i=0; i<SCREEN_WIDTH; i++ ) {
        WriteToFrameBuffer(i,j, BackgroundColor);
        WriteToZBuffer(i,j, MAX);   }
for ( each polygon )
    for ( each pixel in polygon's projection ) {
        z = polygon's z value at (i,j) ;
        if ( z < ReadFromZBuffer(i,j) ) {
            WriteToFrameBuffer(i,j, polygon's color at (i,j));
            WriteToZBuffer(i,j, z);  }}
```

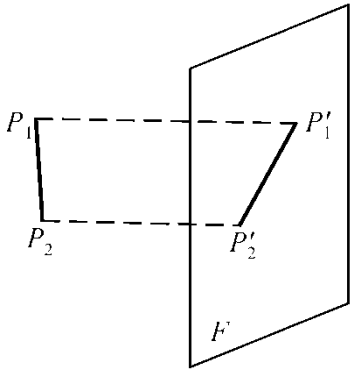
Z-buffer：



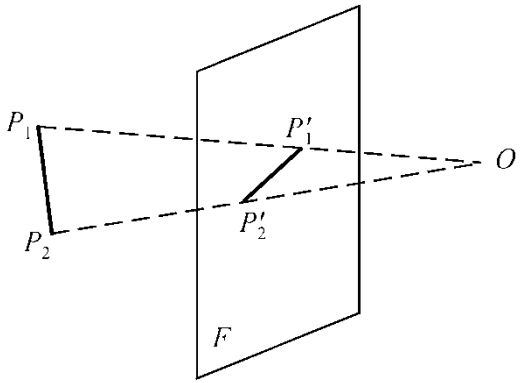




Project：



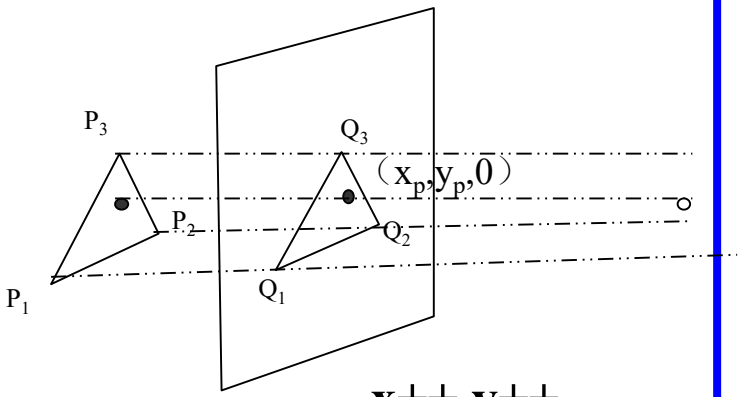
Orthographic



Perspective

计算Z值。

Ax+By+Cz+D=0



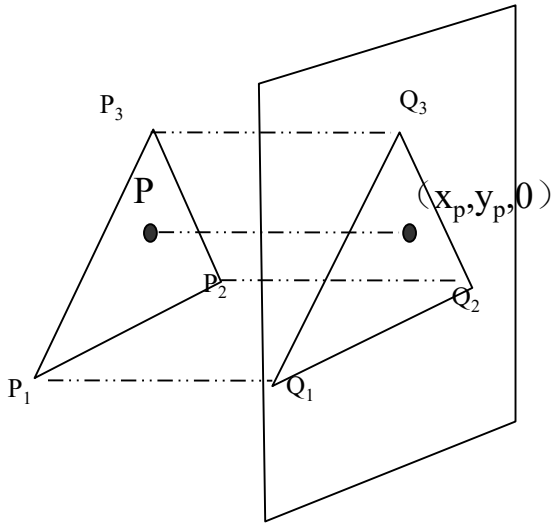
x++,y++

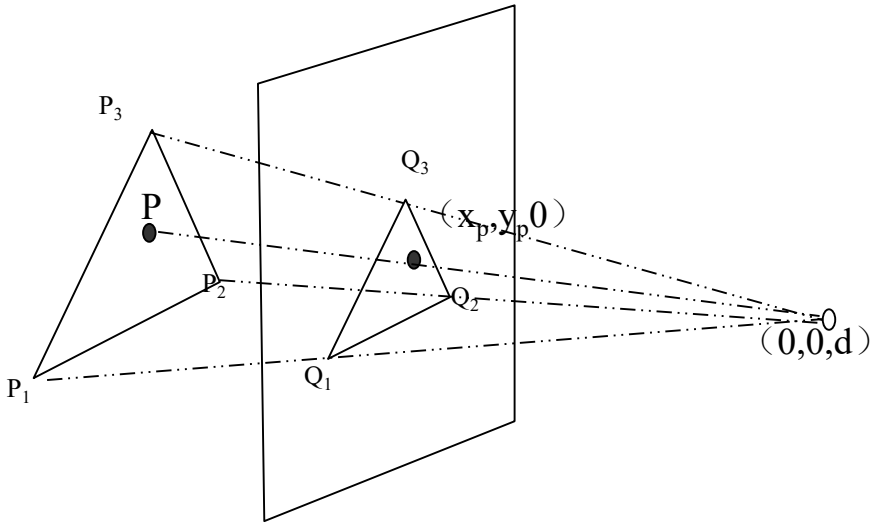
DDA

Z+??

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

问题：如何加速





Ax+By+Cz+D=0

$(x,y,z) \rightarrow (x,y,0)$

$(x,y,z) \rightarrow (x_p,y_p,0)$

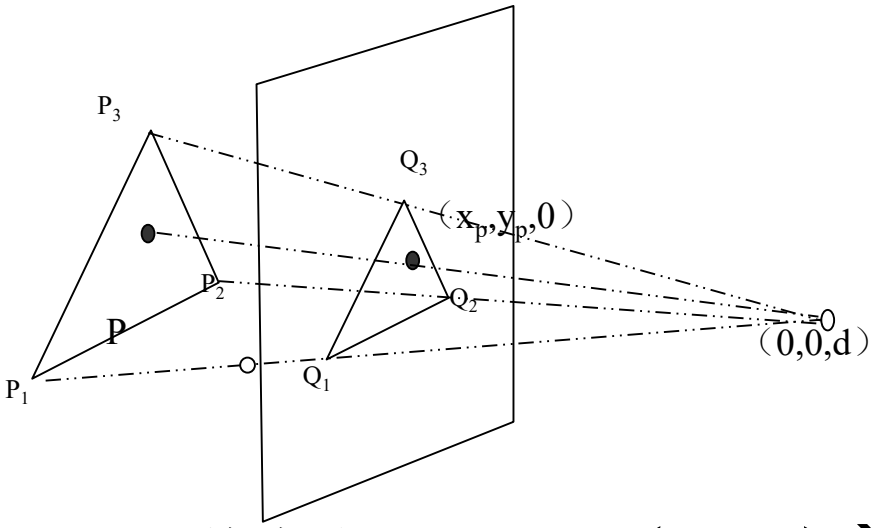
$(x,y,z) \rightarrow (x_p,y_p,z)$

正交投影 →

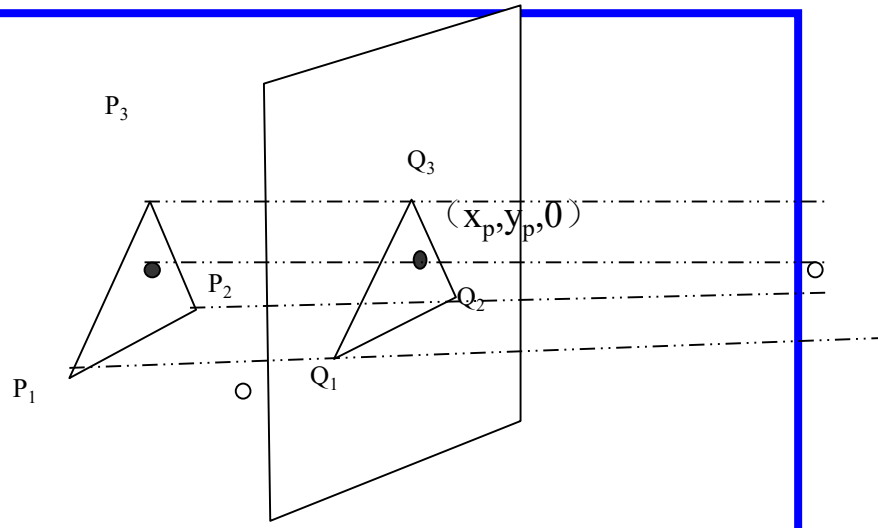
$(x_p,y_p,0)$

$$\frac{x}{x_p} = \frac{y}{y_p} = \frac{d-z}{d}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix}$$



透视投影



透视变换

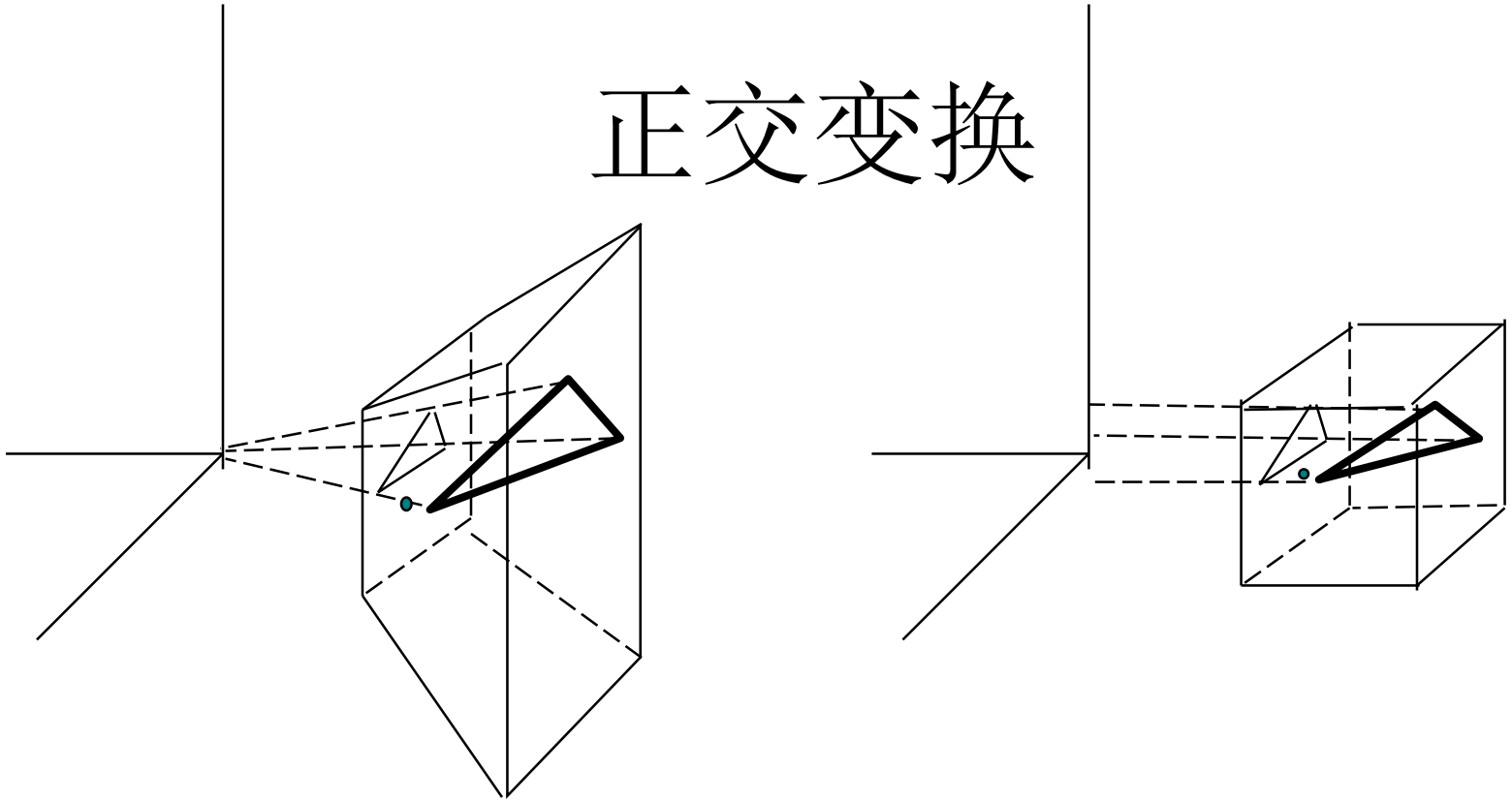
$(x,y,z) \rightarrow (x_p,y_p,z)$

正交投影 → $(x_p,y_p,0)$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1/d & 1 \end{bmatrix}$$

将透视投影拆分为两步：
透视变化+正投影

正交变换



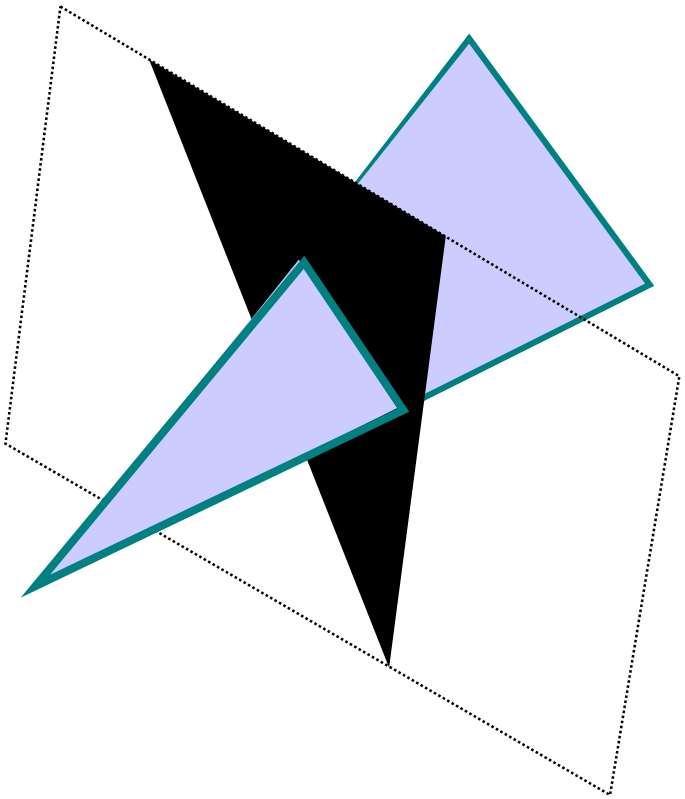
- 先施加透视变换，使得透视投影的棱台变换为正投影的长方体，再进行正投影。

BSP 树
Binary Space Partitioning Trees

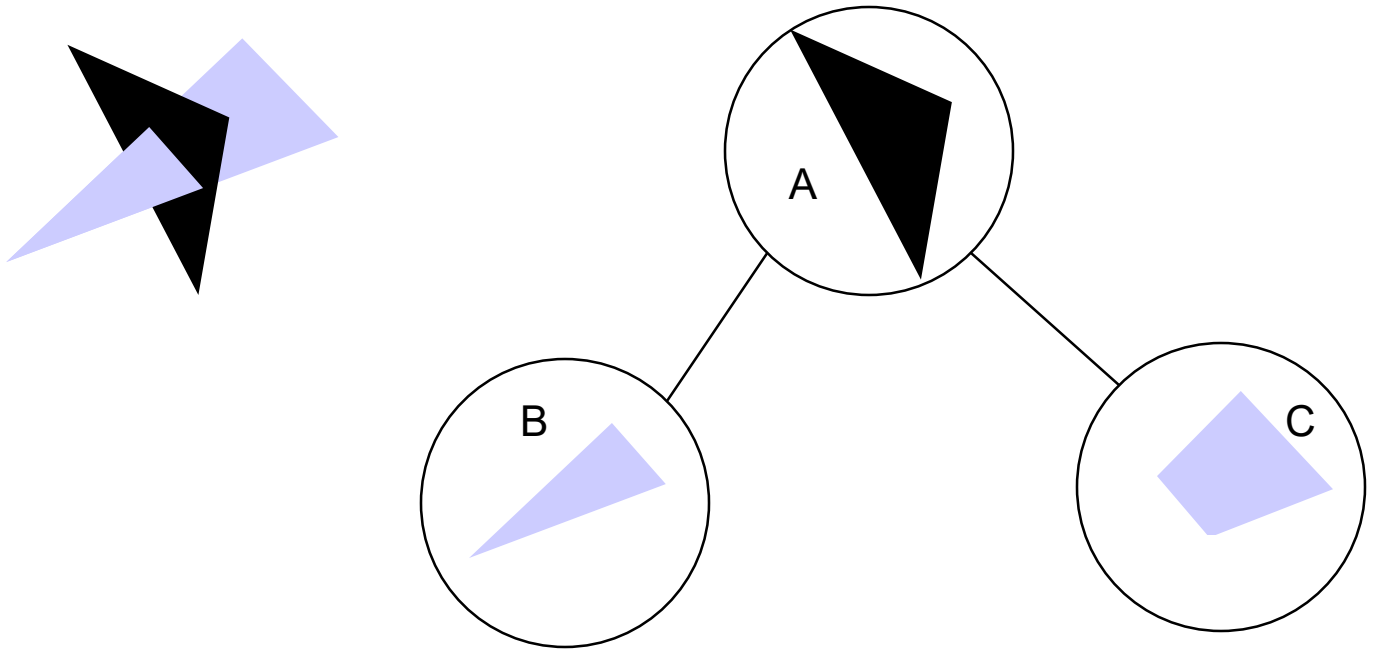
- Very efficient for a static group of 3D polygons as seen from an arbitrary viewpoint
- Correct order for Painter's algorithm is determined by a suitable traversal of the binary tree of polygons: BSP Tree

BSP 树

- 对场景中的面片构造一个二叉树，能够根据视点位置给出合适的二叉树遍历方法对场景中的面片按前后关系排序。



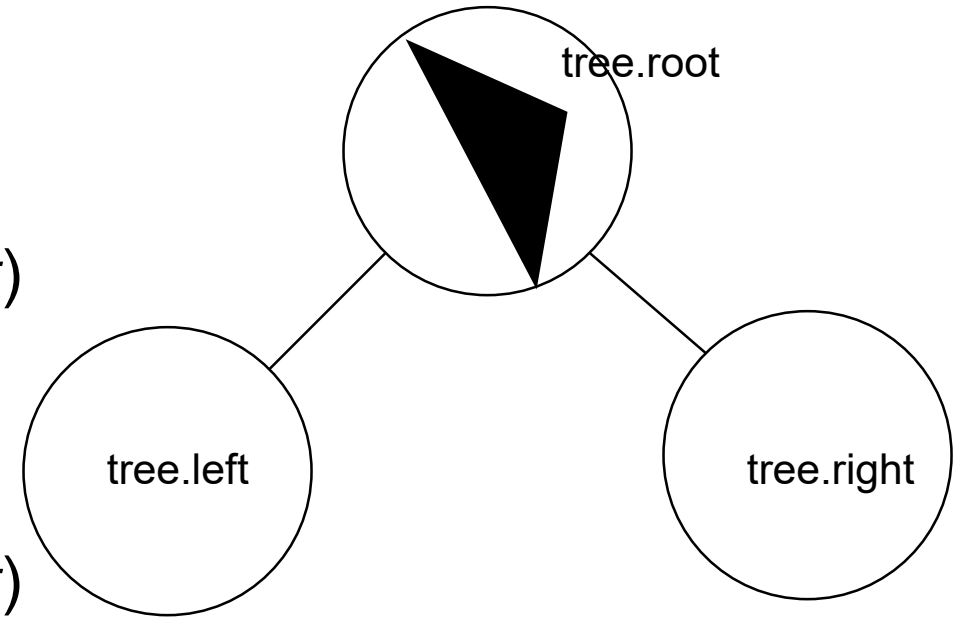
BSP Tree



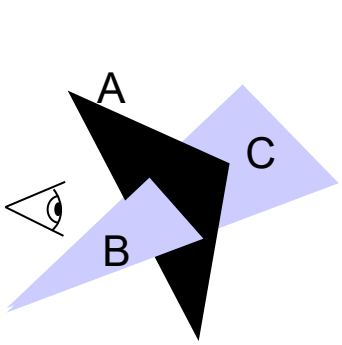
BSP 树

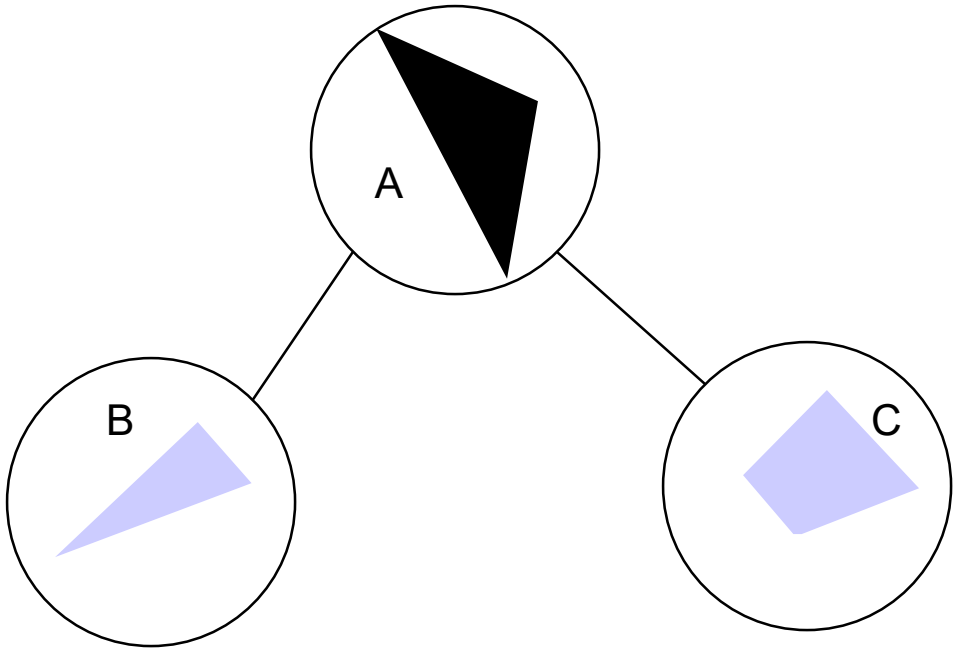
BSP Tree 的显示

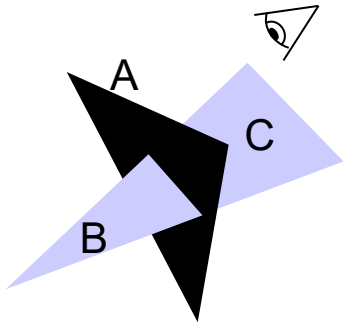
function *draw*(bsptree *tree*, point *eye*)
if *tree*.*empty* then
    return
if $f_{tree.root}(eye) < 0$
    draw (*tree.right*)
    rasterize(*tree.root*)
    draw(*tree.left*)
else
    draw (*tree.left*)
    rasterize(*tree.root*)
    draw(*tree.right*)



BSP Tree







rasterize(C)
rasterize(A)
rasterize(B)

rasterize(B)
rasterize(A)
rasterize(C)

BSP 树的构建

- 构建方法
  - 选择一个多边形，先将与此多边形相交的多边形都用按此多边形分割成两个，然后将集合中多边形分成两组：一组位于给定多边形平面前面，构成左子树；另一组位于给定多边形后面构成，右子树。
  - 分别对左右子树重复上述过程直至子树中只有一个多边形。

谢　谢