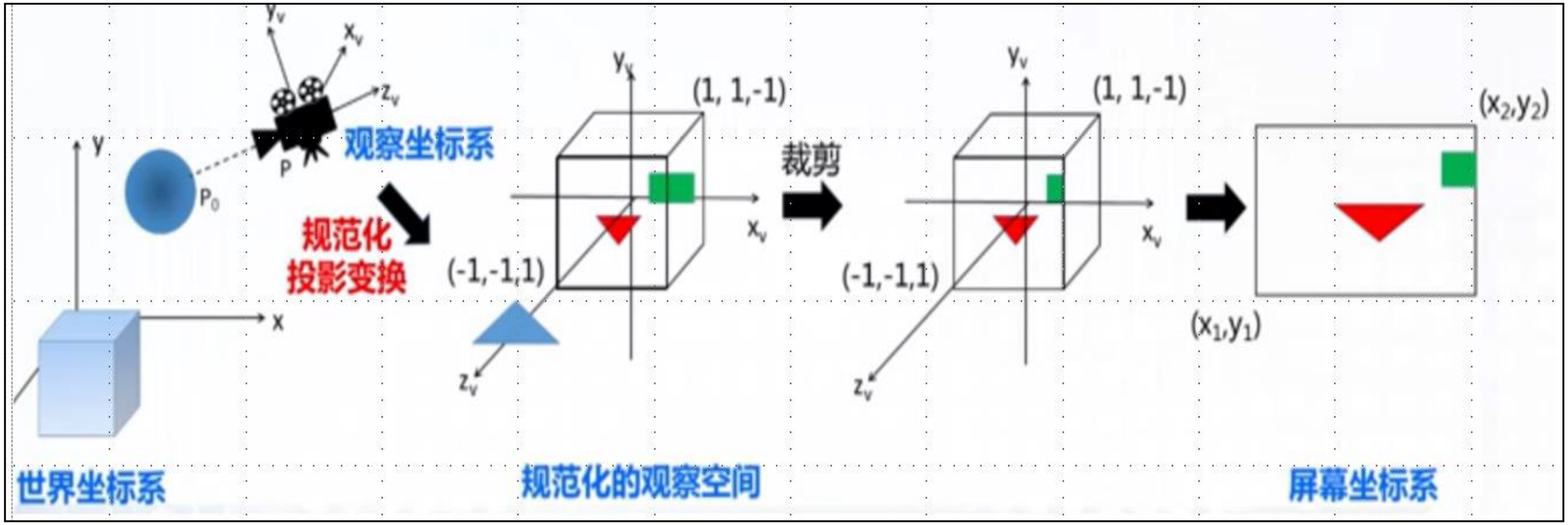
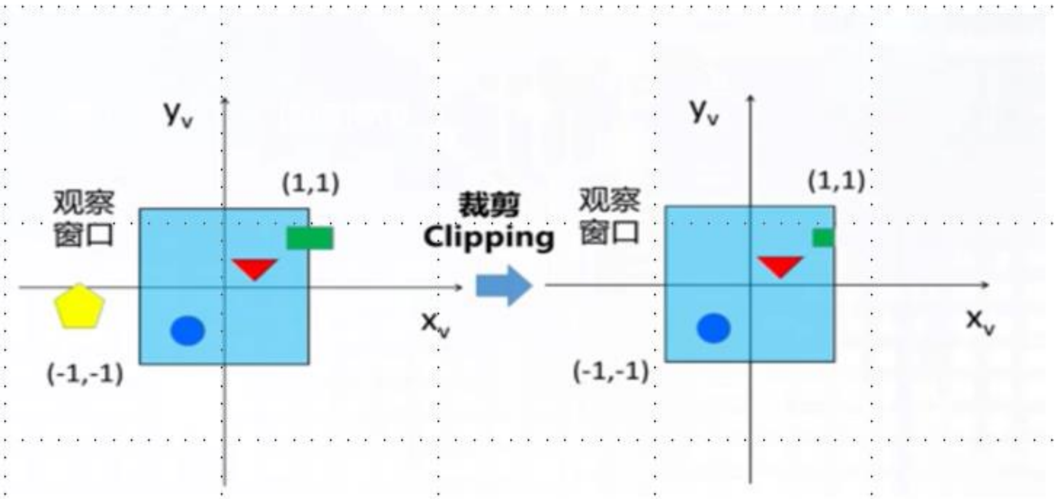
Recap



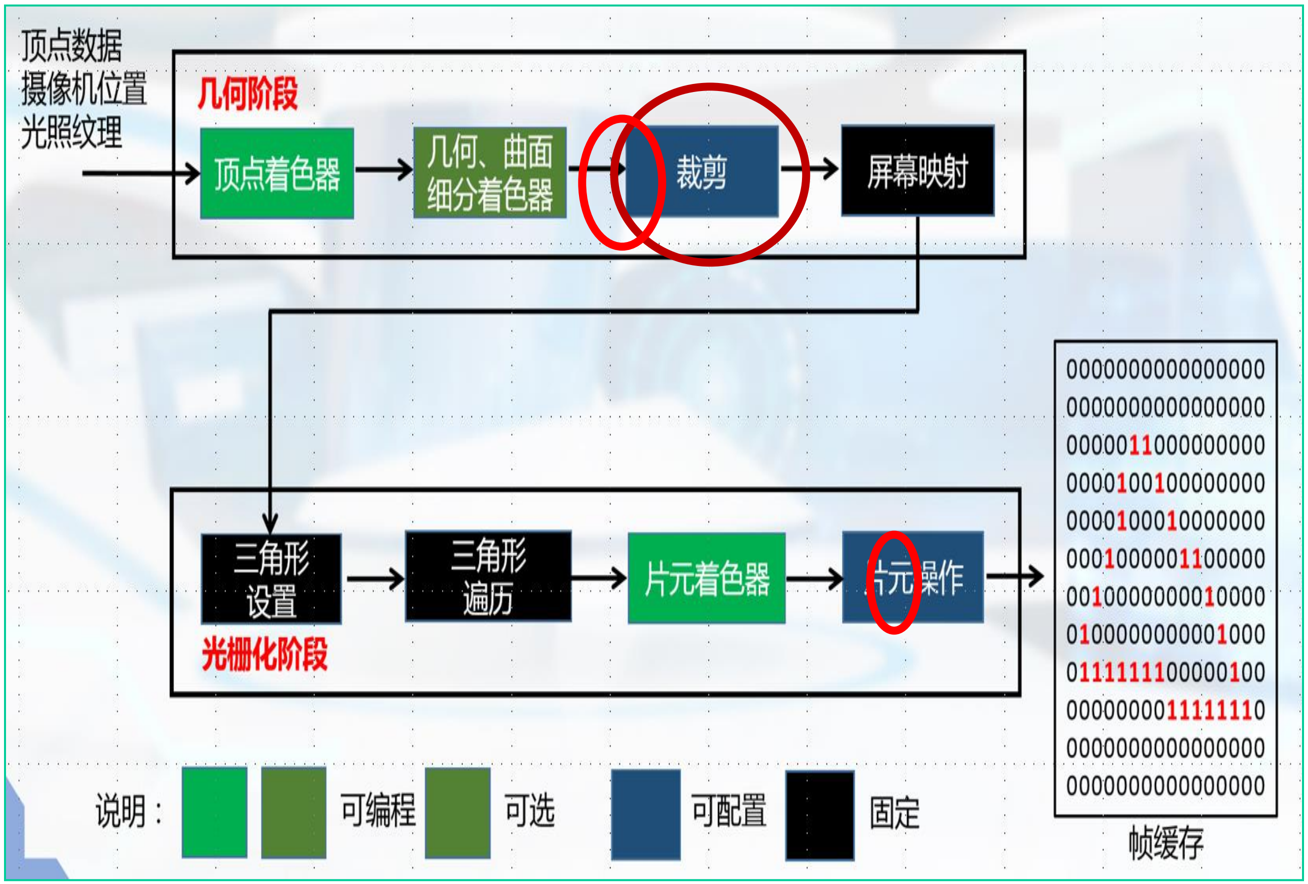
观察坐标系
规范化
投影变换
(1, 1,-1)
(-1,-1,1)
裁剪
(1, 1,-1)
(-1,-1,1)
(x_2,y_2)
(x_1,y_1)
世界坐标系
规范化的观察空间
屏幕坐标系



观察
窗口
(1,1)
(-1,-1)
裁剪
Clipping
观察
窗口
(1,1)
(-1,-1)

Recap(cont.)

- 目前学习位置:裁剪和消隐

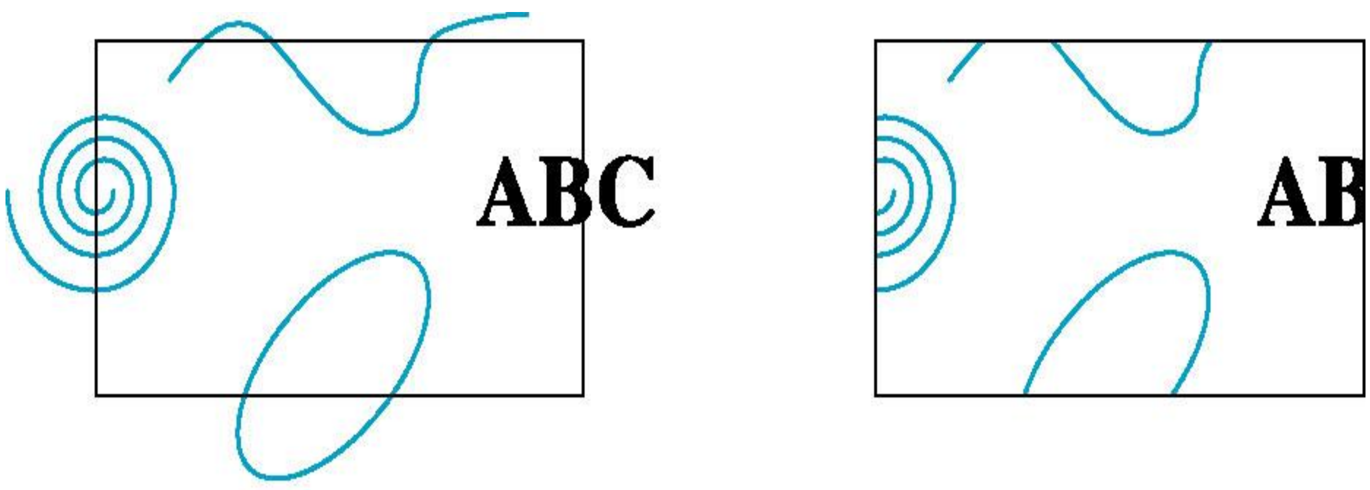


Outline

- **Clipping Algorithm**
  - Clipping Line Segments
    - Cohen-Sutherland Line Clipping编码裁剪算法
    - Liang-Barsky Line Clipping梁永栋裁剪算法
  - Clipping Polygons
    - Sutherland-Hodgeman Polygon Clipping 逐边裁剪算法
- **Hidden Surface Removal Algorithm**
  - Object Space Approach
    - Back-face culling后向面剔除（着色前就消隐掉）
  - Image Space Approach
    - Depth-buffer 深度缓存算法（光栅化渲染中的消隐算法）
    - Ray-casting光线投射（光线跟踪渲染中的消隐算法）

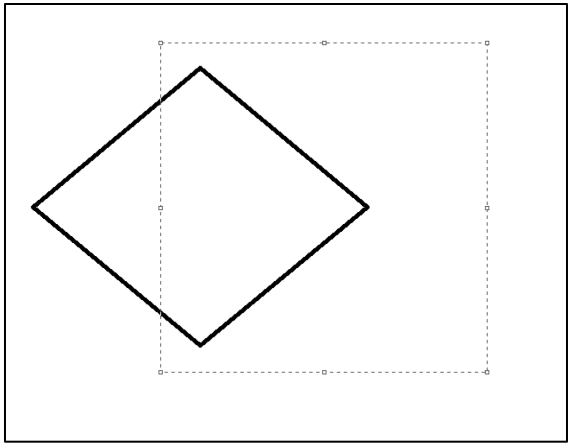
Clipping Algorithm

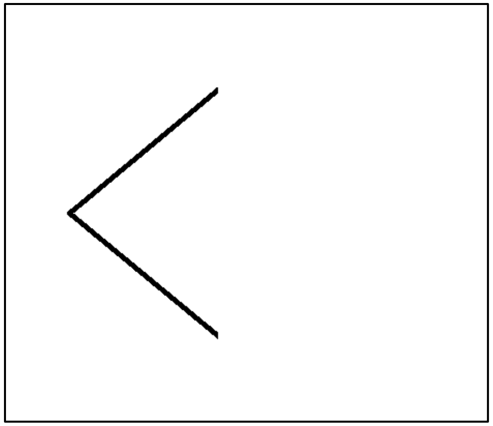
- Clipper 裁剪器:
  - 2D against **clipping window**,3D against **clipping volume**
  - 一般2D裁剪边选择主轴, 3D裁剪面选择主面
- Clipped Primitives 被裁剪图元:
  - Easy for **point, line segments, polygons**
  - Hard for curves and text: Convert to lines and polygons first



Clipping Algorithm(cont.)

- 裁剪算法使用广泛
  - 裁剪算法在渲染管线中使用
  - 裁剪算法可在交互软件中使用，如“绘图”软件中





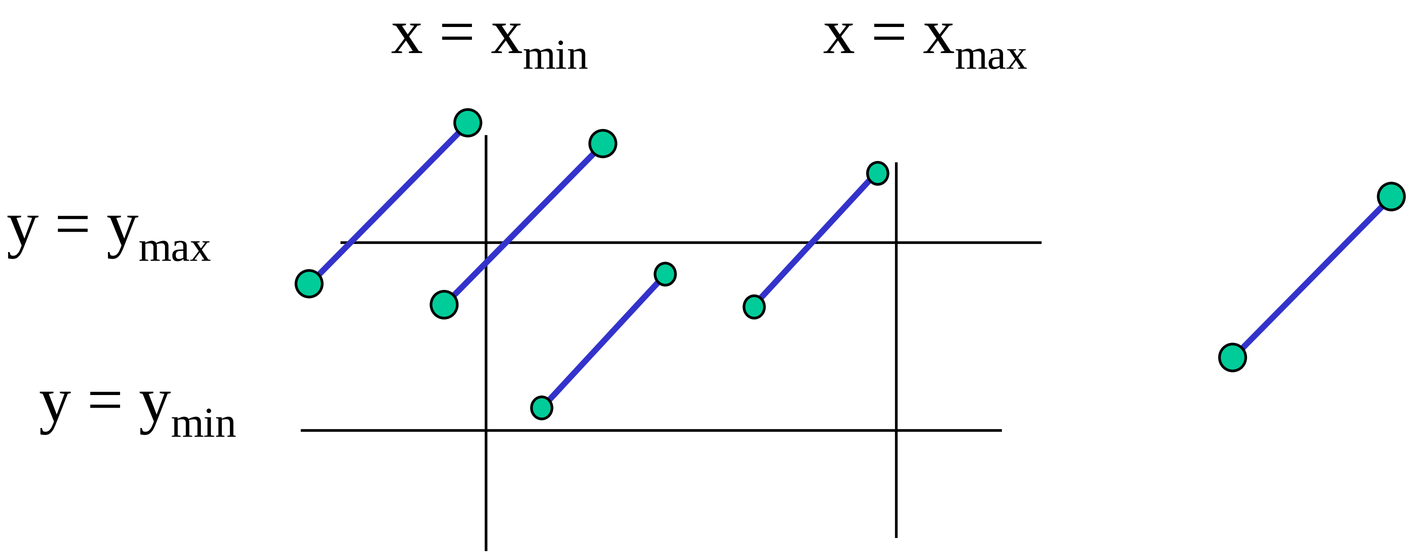
The University of New Mexico

Outline

- **Clipping Algorithm**
  - **Clipping Line Segments**
    - Cohen-Sutherland Line Clipping编码裁剪算法
    - Liang-Barsky Line Clipping梁永栋裁剪算法
  - Clipping Polygons
    - Sutherland-Hodgeman Polygon Clipping 逐边裁剪算法
- **Hidden Surface Removal Algorithm**
  - Object Space Approach
    - Back-face culling后向面剔除（着色前就消隐掉）
  - Image Space Approach
    - Depth-buffer 深度缓存算法（光栅化渲染中的消隐算法）
    - Ray-casting光线投射（光线跟踪渲染中的消隐算法）

Clipping Line Segments

- **Brute force approach**
  - Start with each segment, compute intersections with all sides of clipping window（对每条线段，用裁剪窗口得四条裁剪边去求交，每次求交就分割一次线段（one division per intersection））
  - 缺点 : Inefficient（低效）



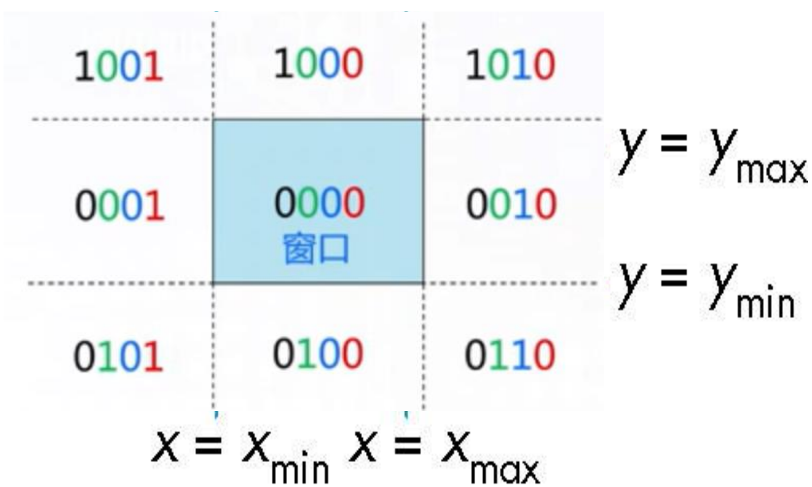
Outline

- Clipping Algorithm
  - Clipping Line Segments
    - **Cohen-Sutherland Line Clipping编码裁剪算法**
    - Liang-Barsky Line Clipping梁永栋裁剪算法
  - Clipping Polygons
    - Sutherland-Hodgeman Polygon Clipping 逐边裁剪算法
- **Hidden Surface Removal Algorithm**
  - Object Space Approach
    - Back-face culling后向面剔除（着色前就消隐掉）
  - Image Space Approach
    - Depth-buffer 深度缓存算法（光栅化渲染中的消隐算法）
    - Ray-casting光线投射（光线跟踪渲染中的消隐算法）

The University of New Mexico

Cohen-Sutherland Line Clipping

- IDEA: eliminate as many cases as possible “without computing intersections”（消除尽可能多的“不用计算交点”的情况）
  - Start with four lines that determine the sides of the clipping window：Xmin, Xmax, Ymin, Ymax.
  - Defining Outcodes $\mathbf{b_0b_1b_2b_3}$ ：For each endpoint, define an outcode , Outcodes divide space into 9 regions



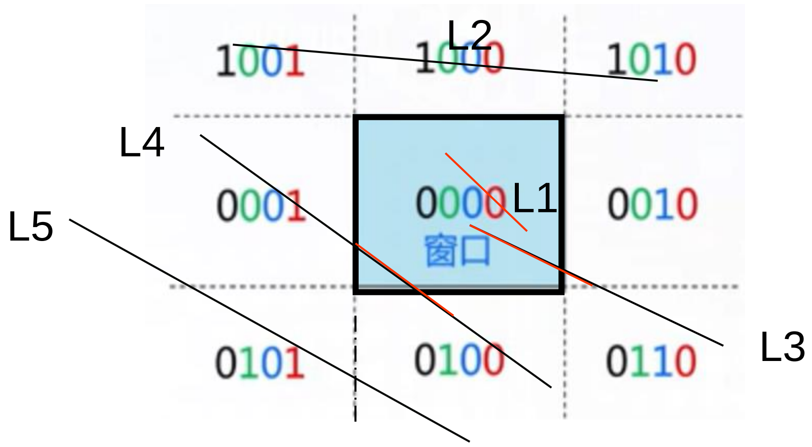
outcode: $b_0b_1b_2b_3$

$b_0 = 1$ if $y > y_{max}$, 0 otherwise
$b_1 = 1$ if $y < y_{min}$, 0 otherwise
$b_2 = 1$ if $x > x_{max}$, 0 otherwise
$b_3 = 1$ if $x < x_{min}$, 0 otherwise

Cohen-Sutherland Line Clipping(cont.)

The University of New Mexico

- **根据线段两端点的outcodes, 可以判定线段和窗口的关系如下**
  - Case 1: both endpoints of line segment inside all four lines
    - Draw (accept) line segment as is
  - Case 2: both endpoints outside all lines and on same side of a line
    - Discard (reject) the line segment
  - Case 3: One endpoint inside, one outside
    - Must do at least one intersection
  - Case 4: Both outside
    - May have part inside, Must do at least one intersection



L1(0000 0000) visiblee: accept
L2(1010 1001) invisible： reject
L3(0000 0110) part visible
L4(0001 0100) part visible
L5(0001 0100) invisible

Cohen-Sutherland Line Clipping(cont.)

The University of New Mexico

- 算法(演示)
- Algorithm Description

(1)求出线段的两个端点P_1、P_2的区域码为code1,code2

(2)While(done){

1. 若code1|code2=0,即两端点P_1、P_2的区域码位"或"操作结果为0000，则完全可见(两端点在裁剪窗口内)，结束并输出线段done =true(简取)
2. 若code1&code2!=0，即两端点P_1、P_2的区域码位"与"操作结果不为0000，则完全不可见（两端点同一外侧)，结束done =true(简弃)
3. 否则，线段有可能是部分可见或完全不可见。这时用四条裁剪边逐个测是否与线段有交，若有交则作如下操作：

   a、先判断P_1区域码code1，如果P1在窗口内，则交换P_1、P_2, (保证P1在窗口外)。

   b、再用线段P_1P_2与该裁剪边的有效交点代替P_1,并求出新P_1的区域码code1后，返回(2)进入下一轮循环。

}

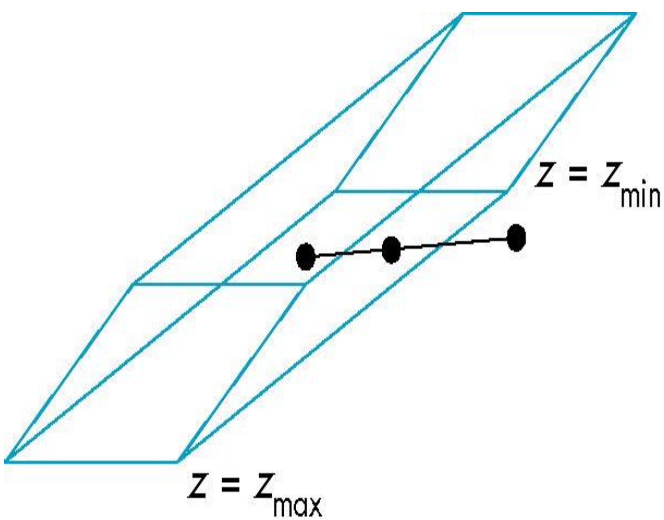
Cohen-Sutherland Line Clipping(cont.)

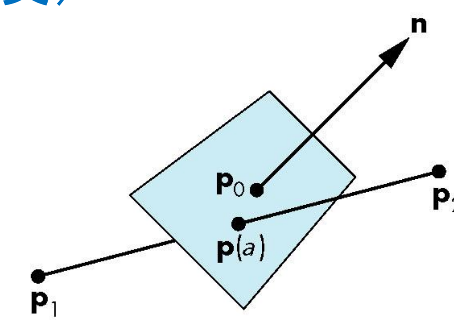
- Efficiency
  - In many applications, the size of clipping window sides is small relative to the size of the entire data base（一般裁剪窗口边比被裁剪图元相对少得多）
    - Most line segments are outside one or more side of the window and can be eliminated based on their outcodes
- Inefficiency
  - when code has to be reexecuted for line segments that must be shortened in more than one step(有些线段不只一次被裁剪)

Cohen-Sutherland Line Clipping(cont.)

The University of New Mexico

- Clipping 3D Line Segments
  - General clipping in 3D requires intersection of line segments against arbitrary plane
  - Example: oblique view(斜的观察)
    - Need Plane-Line Intersections(线面求交)





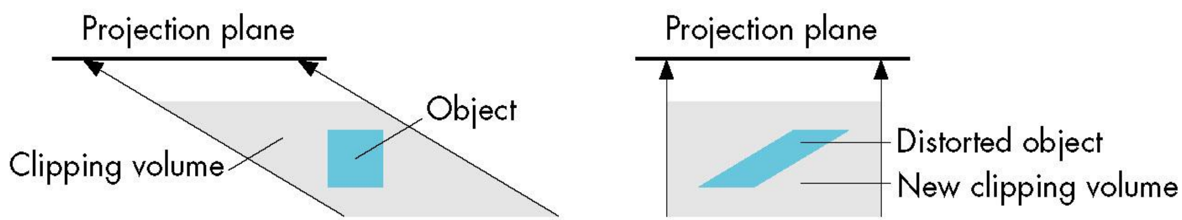
$$a = \frac{n \bullet (p_o - p_1)}{n \bullet (p_2 - p_1)}$$

Cohen-Sutherland Line Clipping(cont.)

➢Clipping 3D Line Segments（cont.)

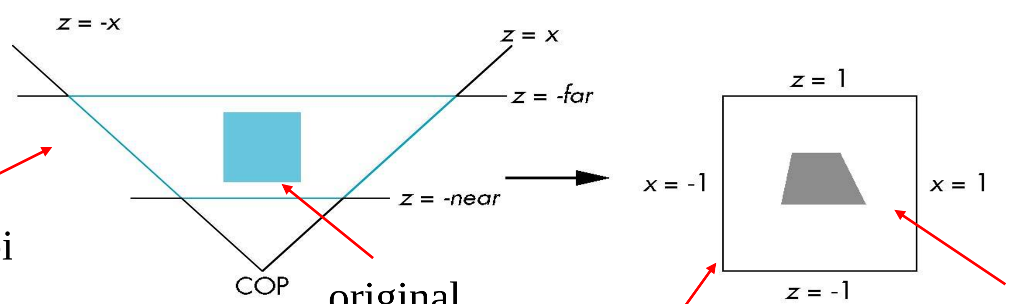
- after normalization Projection, we clip against sides of right parallelepiped,

top view



before normalization

after normalization



original clippi volume

original object

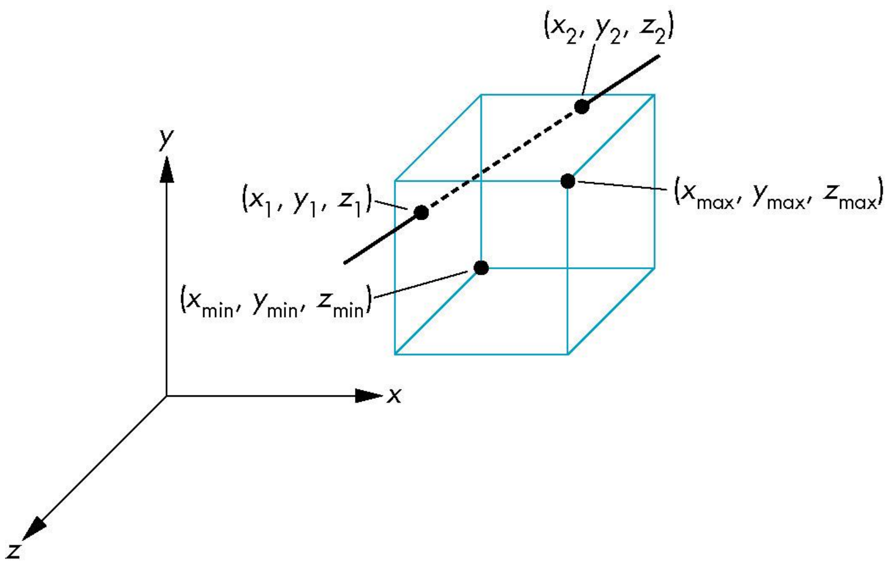
new clipping volume

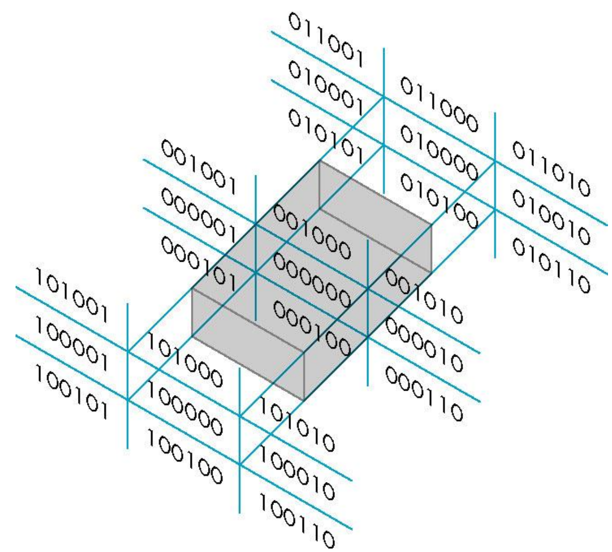
distorted object projects correctly

Clipping 3D Line Segments（cont.)

➢Clipping 3D Line Segments（cont.)

- Use 6-bit outcodes区域码, 27 个区域
- “clip line segment裁剪边” against “clip planes裁剪面”



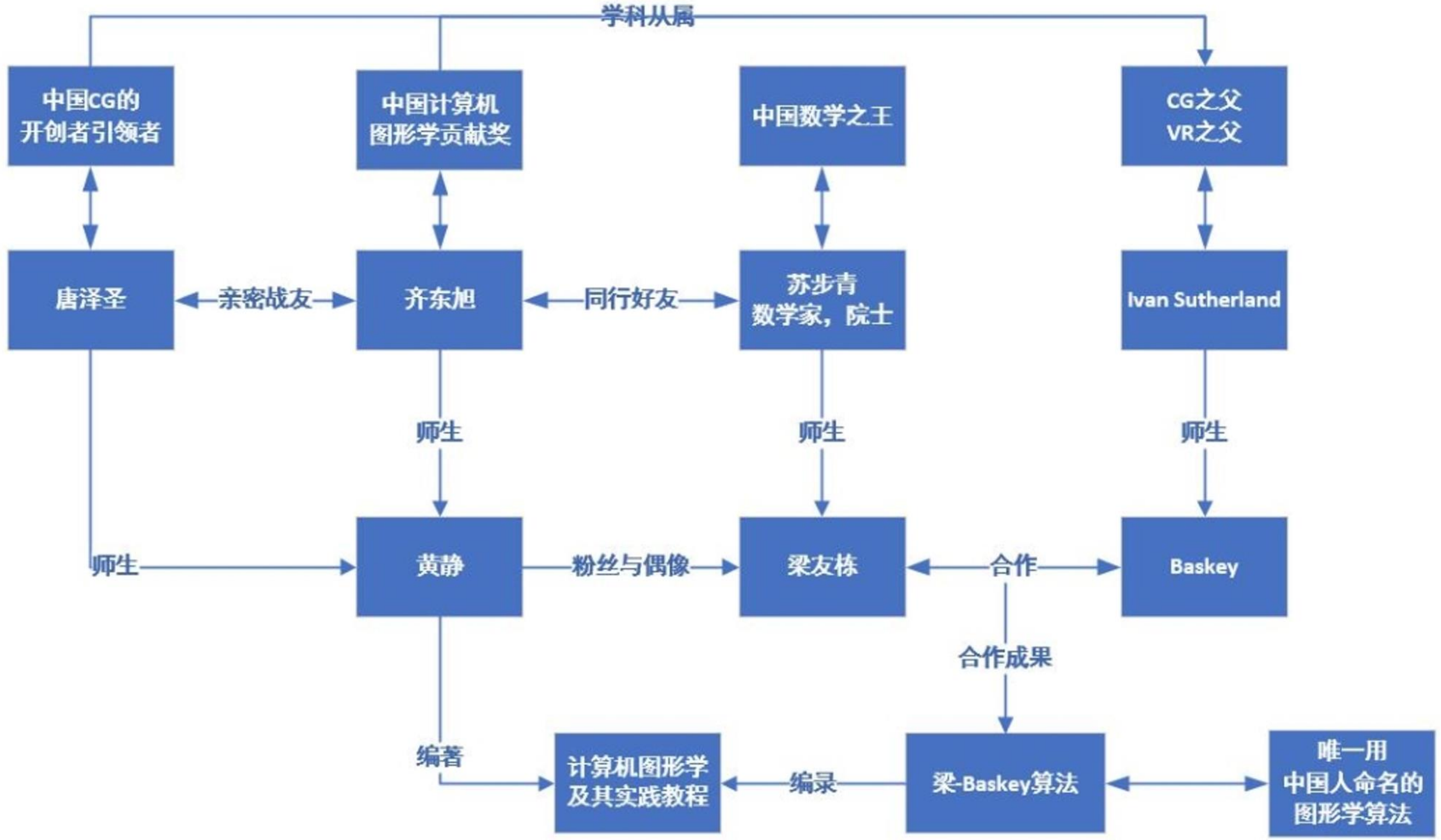


Outline

- **Clipping Algorithm**
  - **Clipping Line Segments**
    - Cohen-Sutherland Line Clipping编码裁剪算法
    - **Liang-Barsky Line Clipping梁永栋裁剪算法**
  - Clipping Polygons
    - Sutherland-Hodgeman Polygon Clipping 逐边裁剪算法
- **Hidden Surface Removal Algorithm**
  - Object Space Approach
    - Back-face culling后向面剔除（着色前就消隐掉）
  - Image Space Approach
    - Depth-buffer 深度缓存算法（光栅化渲染中的消隐算法）
    - Ray-casting光线投射（光线跟踪渲染中的消隐算法）

Liang-Barsky Line Clipping

- 梁永栋-Barsky线段裁剪算法

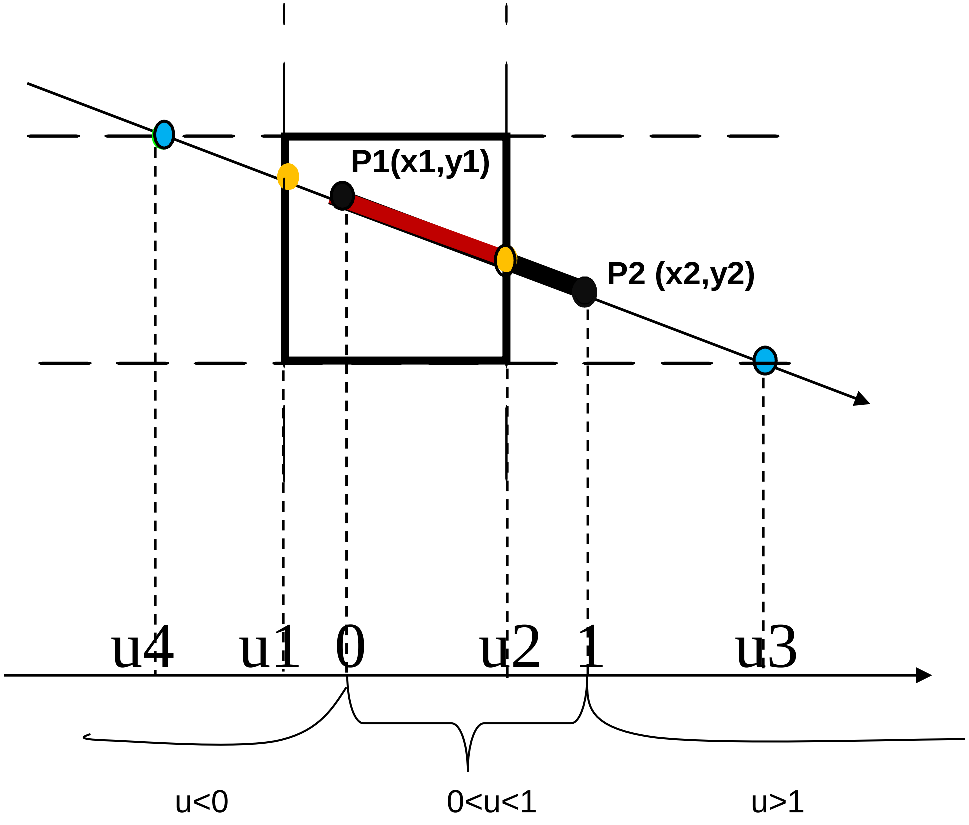


Liang-Barsky Line Clipping(cont.)

- **Idea:**
  - **采用直线的参数方程表示线段 P(u)=P1+u(P2-P1)　0≤u≤1;**
  - **把找被裁剪后的线段端点问题转换为找端点对应的参数Umin,Umax**

如下图中： Umin =MAX(0,u1,u4)=0, Umax = MIN (1,u2,u3)=u2，

则裁剪后线段为P1'P2'： P1' =P1+ Umin *(P2-P1)=P1;　P2' =P1+ Umax *(P2-P1)



Liang-Barsky Line Clipping(cont.)

The University of New Mexico

➢既在裁剪窗口中又在线段上的点的参数，需要满足三个不等式：

1）on the line segment p1p2：P(u)=P1+u(P2-P1)

- 0≤u≤1

2）in the clipping windows :

- 根据：wmin≤x ≤Xwmax，Ywmin≤y≤Ywmax
- 以及：X= x_1+u(x2-x1)=x1+u△x 和y= y_1+u(y2-y1)=y1+u△y
- 得到两个不等式：

■Xwmin≤x1+u△x≤Xwmax

■Ywmin≤y1+u△y≤Ywmax

Liang-Barsky Line Clipping(cont.)

The University of New Mexico

➢线段有裁剪结果的情况判定：

- ➢If **Umin<Umax**， have clipped line segment in the window

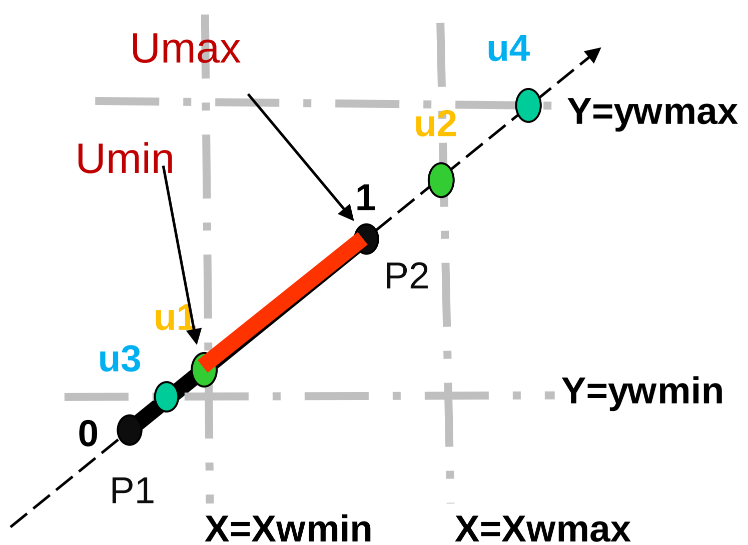
- Bring Umin , Umax in the following formula, get the clipped line segment endpoints:
- 例如下图中情况满足三个不等式

0≤ u ≤1， **u1≤u≤ u2,** **u3 ≤u≤ u4**

则：**Umin=max(0,u1,u3)=u1, Umax=min(1,u2,u4)=1,**

因： **Umin< u <Umax** ，有裁剪结果为**P(Umin)P(Umax)**，端点坐标计算如下：

- $x= x_1+Umin(x2-x1)=x1+Umin\Delta x$, $y = y_1+Umin(y2-y1)=y1+Umin\Delta y$
- $x= x_1+Umax(x2-x1)=x1+Umax \Delta x$, $y= y_1+ Umax(y2-y1)=y1+ Umax \Delta y$



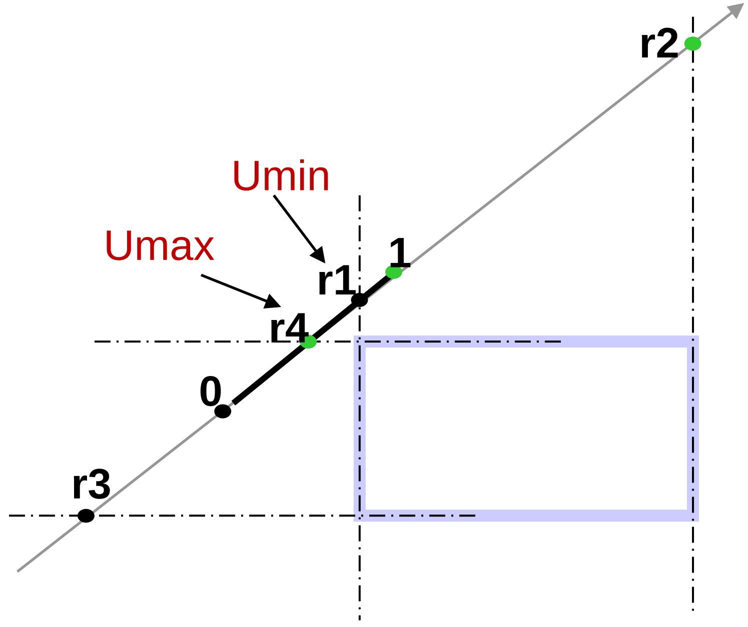
Liang-Barsky Line Clipping(cont.)

- 线段完全在窗口外的判定：
  - If Umin>Umax   this line segment is outside the window

例如下图情况：

- **Umin =MAX(0, r1, r3)   // here, r1**
- **Umax=MIN(1, r2,   r4) ;   // here, r4**

因为Umin>Umax,所以没有满足三个条件不等式的参数u存在，即没有点



Liang-Barsky Line Clipping(cont.)

➢算法大致思路

根据：$0 \le u \le 1$， $xw_{min} - x_1 \le u\triangle x \le xw_{max} - x_1$， $yw_{min} - y_1 \le u\triangle y \le yw_{max} - y_1$

后两个不等式转换为4个单边不等式

$u(-\triangle x) \le x1 - xwmin$， $u(\triangle x) \le xwmax - x1$, $u(-\triangle y) \le y1 - ywmin$, $u(\triangle y) \le ywmax - y1$

为便于讨论，四个不等式统一形式为：$u^* pk \le qk$（k=1,2,3,4），

- If pk=0, *there is no corresponding rk*
- If pk!=0, compute rk=qk/pk（k=1,2,3,4)

$$p_1 = -\triangle x \quad q_1 = x_1 - xw_{min} \quad r_1 = q_1/p_1$$
$$p_2 = \triangle x \quad q_2 = xw_{max} - x_1 \quad r_2 = q_2/p_2$$
$$p_3 = -\triangle y \quad q_3 = y_1 - yw_{min} \quad r_3 = q_3/p_3$$
$$p_4 = \triangle y \quad q_4 = yw_{max} - y_1 \quad r_4 = q_4/p_4$$

  - Compute Umin,Umax, Group by the sign of pk

    初始：Umin=0,Umax=1

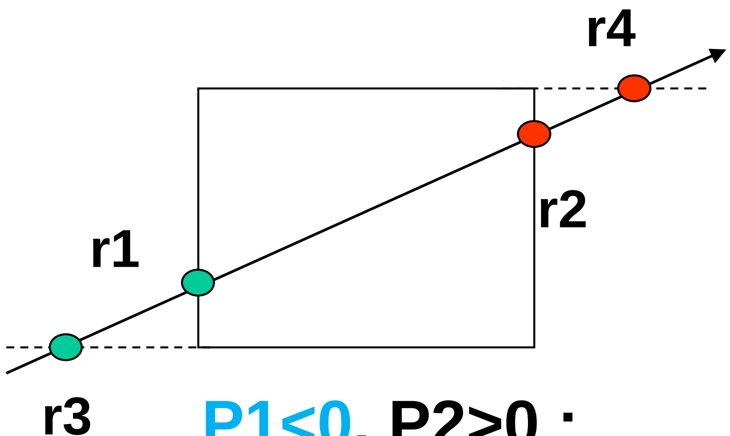
    $p_k<0$：Umin=max(Umin, r_k) ，$p_k>0$：Umax=min(Umax, r_k)

- Compute result line segment endpoints using Umin and Umax
  - If Umin>=Umax: this line segment is outside the clipping window.
  - If Umin<Umax: compute (X(umin),Y(umin)), (X(uman), Y(umin))

The University of New Mexico

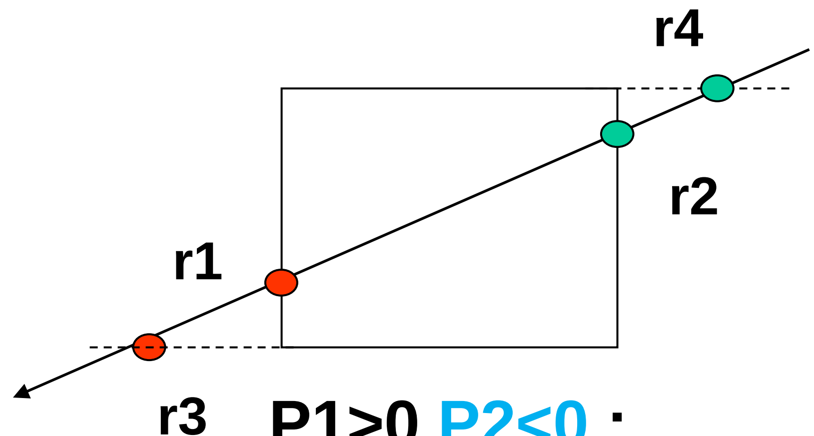
Liang-Barsky Line Clipping(cont.)

- If $P_k != 0$，有 $r_k = q_k / p_k$,



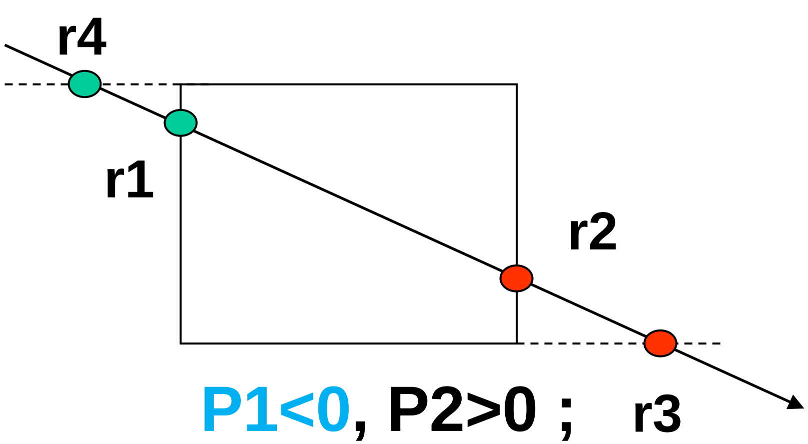
P1<0, P2>0 ;

P3<0, P4>0



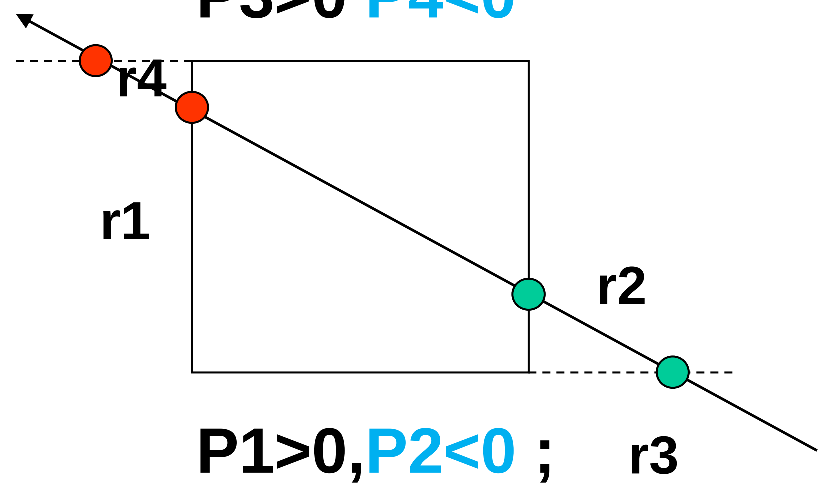
P1>0 P2<0 ;

P3>0 P4<0



P1<0, P2>0 ;

P3>0, P4<0



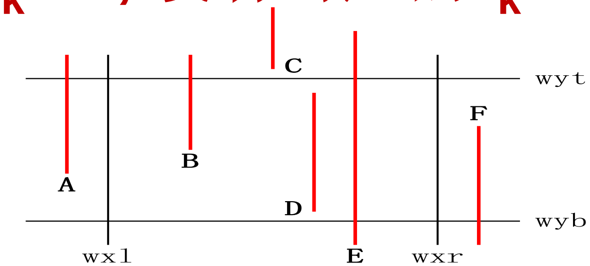
P1>0,P2<0 ;

P3<0,P4>0

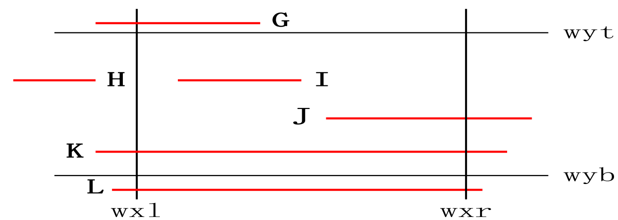
The University of New Mexico

Liang-Barsky Line Clipping(cont.)

- **If P_k =0, 没有对应的r_k**

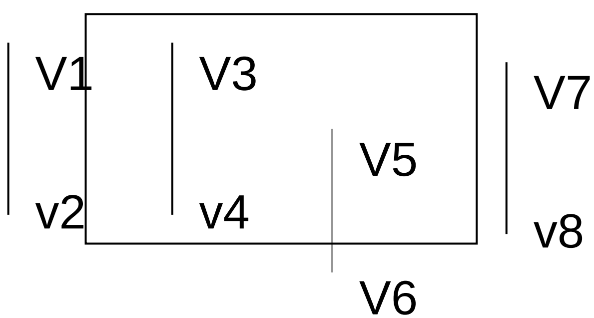


(a) 直线段与窗口边界 wxl和wxr平行的情况



(b) 直线段与窗口边界 wyb和wyt平行的情况

- □当p1 =-△x =0，必有p2= △x= 0，线与Y平行;
- □当p3=-△y =0，必有p4=△y=0，线与X平行



pk=0时,如果对应的某个q_k<0，则该线段完全不可见，退出裁剪

- 对于线段V_1V_2，$p_1=p_2=0$，(k=1,2)，线与Y平行
  此时$q_1=x_1- xw_{min}<0$，$q_2 =xw_{max} - x_1>0$，因此V_1V_2在窗口外。
- 对于线段V_7V_8，p1=p2=0，(k=1,2)，线与Y平行
  此时$q_1=x_1- xw_{min}>0$，$q_2 =xw_{max} - x_1<0$，因此V_7V_8在窗口外。

pk=0时,如果其对应的两个q_k>0，则必有窗口内线段，继续裁剪。

- 对于直线V_3V_4，$p_1=p_2=0$，(k=1,2)，线与Y平行
  此时，$q_1>0$，$q_2>0$，则V_3V_4在窗口X方向的两裁剪边内.
- 对于直线V_5V_6，$p_1=p_2=0$，(k=1,2)，线与Y平行
  此时，$q_1>0$，$q_2>0$，则V_5V_6在窗口X方向的两裁剪边内.

Liang-Barsky Line Clipping(cont.)

The University of New Mexico

➢总结:完整算法描述

(1)输入直线段的两端点坐标(x_1,y_1)和(x_2,y_2)，以及窗口的四条边界坐标:wyt、wyb、wxl和wxr。 令Umin=0; Umax=1; Δx=x2-x1; Δy=y2-y1;

(2)若Δx=0，则$p_1=p_2=0$。若$q_1<0$或$q_2<0$，则该直线段不在窗口内，算法转(7)。否则$q_1>0$且$q_2>0$必有裁剪结果，算法转(4) 。

(3)若Δy=0，则$p_3=p_4=0$。若$q_3<0$或$q_4<0$，则该直线段不在窗口内，算法转(7)。否则$q_3>0$且$q_4>0$必有裁剪结果，算法转(4)。

(4) 若Δx=0，则有pk≠0(k=3,4),计算r3,r4,

若Δy=0， pk≠0(k=1,2),，计算r1,r2。

若pk≠0（k=1,2,3,4)，计算r1,r2,r3,r4。

将pk<0的rk与 Umin**比较后取大值赋予**Umin **，将pk>0的rk与Umax比较后取小值赋予Umax。**

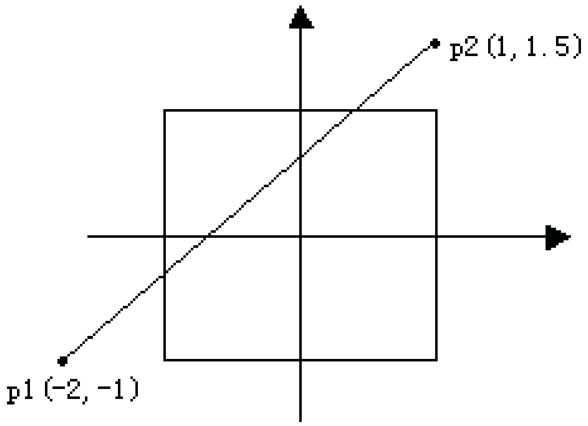
(5) 若Umin>Umax，则直线段在窗口外，算法转(7)。

(6)若Umin<Umax，利用直线的参数方程求得直线段在窗口内的两端点坐标，调用画线程序绘制裁剪后线段，结束。

(7) 线段在窗口外，无裁剪结果，结束。

Liang-Barsky Line Clipping(cont.)

- **Practice:**
  - Clipping window: (-1,-1),(1,1)
  - Line segment: p1p2(-2,-1)(2,1.5)



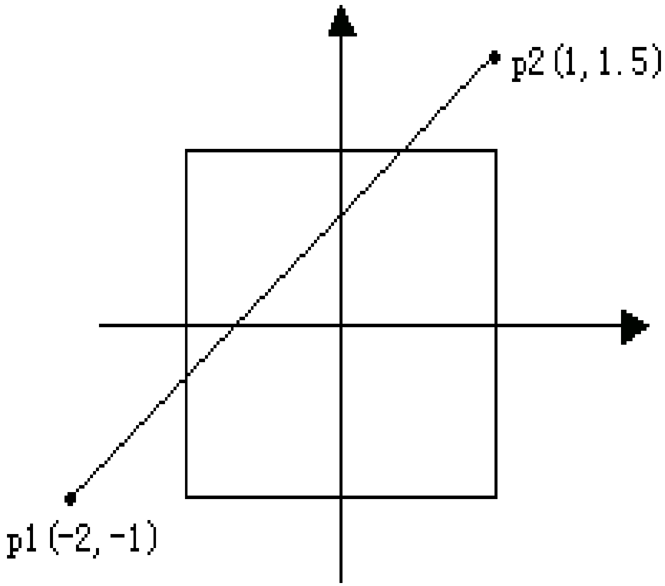
The University of New Mexico

$p_1=-\triangle x \quad q_1= x_1- xw_{min} \quad r_1=q_1/p_1$

$p_2=\triangle x \quad q_2= xw_{max}- x_1 \quad r_2=q_2/p_2$

$p_3=-\triangle y \quad q_3= y_1- yw_{min} \quad r_3=q_3/p_3$

$p_4=\triangle y \quad q_4= yw_{max}- y_1 \quad r_4=q_4/p_4$



- $\triangle x=3$, $\triangle y=2.5$
- $p_1=-3 \quad q_1=-1$;
- $p_2=3 \quad q_2= 3$
- $p_3=-2.5 \quad q_3= 0$;
- $p_4=2.5 \quad q_4= 2$
- 对于p<0，计算小端u
  - r1=1/3, r3=0，(X1,Y1)对应的u=0,
  - 取三个数值中的最大值u_1=max(1/3,0,0)=1/3。
- 对于p>0，计算大端u
  - r2=1，r4=4/5，(X2,Y2)对应的u=1,
  - 取三个数值中的最小值u_2=min(1,4/5,1)=4/5
- 因为$u_1<u_2$，则可见线段的端点坐标可计算方法出：
  - $x=x_1$+u1*$\triangle x$=-1,$y=y_1$+u1* 2.5 =-1/6即(-1,-1/6)
  - $x=x_1$+u2*$\triangle x$=2/5,$y=y_1$+u2*2.5 =1 即(2/5,1)

The University of New Mexico

Liang-Barsky Line Clipping(cont.)

➤Liang-Barsky Line Segment Clipping in 3D

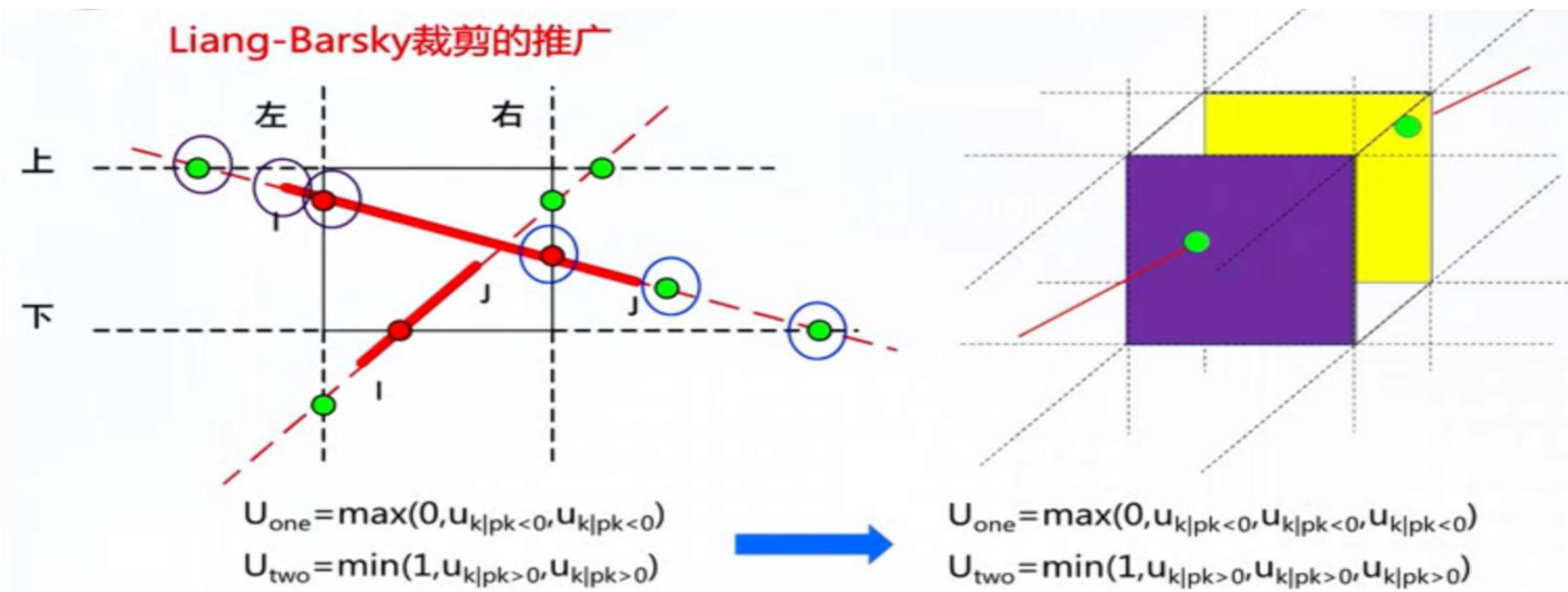
1）on the line segment: $0 \le u \le 1$

2）in the clipping view volume

$Xwmin \le X1+u\Delta X \le Xwmax$, $\Delta x=x2-x1$

$Ywmin \le Y1+u\Delta Y \le Ywmax$, $\Delta y=y2-y1$

$Zwmin \le Z1+u\Delta Z \le Zwmax$, $\Delta z=z2-z1$

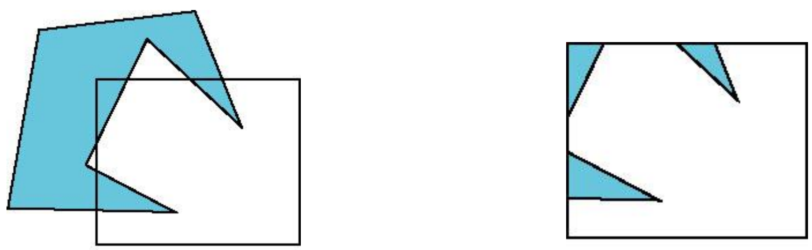


Outline

- Clipping Algorithm
  - Clipping Line Segments
    - Cohen-Sutherland Line Clipping编码裁剪算法
    - Liang-Barsky Line Clipping梁永栋裁剪算法
  - Clipping Polygons
    - Sutherland-Hodgeman Polygon Clipping 逐边裁剪算法
- Hidden Surface Removal Algorithm
  - Object Space Approach
    - Back-face culling后向面剔除（着色前就消隐掉）
  - Image Space Approach
    - Depth-buffer 深度缓存算法（光栅化渲染中的消隐算法）
    - Ray-casting光线投射（光线跟踪渲染中的消隐算法）

Polygon Clipping

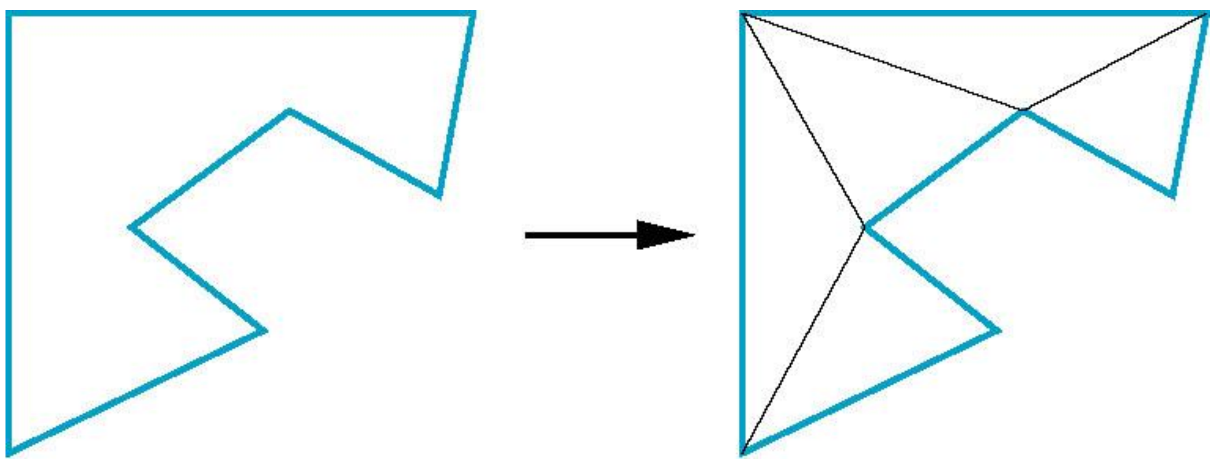
- Not as simple as line segment clipping
  - Clipping a line segment yields at most one line segment（裁剪多边形可能产生“多”条线段）
  - Clipping a polygon can yield multiple polygons （裁剪凹多边形可能产生“多”个多边形）



  - However, clipping a convex(凸) polygon can yield at most one other polygon（但是裁剪凸多边形只产生“一”个多边形）

Polygon Clipping(cont.)

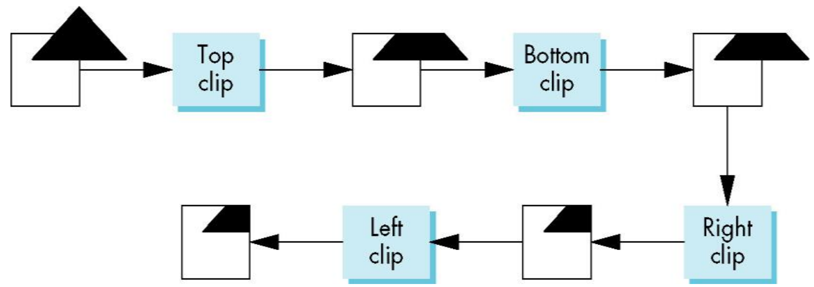
- Tessellation and Convexity(细分和凸性)
  - One strategy is to replace nonconvex (*concave*凹) polygons with a set of triangular polygons (a *tessellation*),（将非凸多边形替换为一组三角多边形）
  - Also makes fill easier(使得后续填充变得容易)
  - Tessellation through “tesselllation shaders”(网格细分可通过“细分着色器”完成)



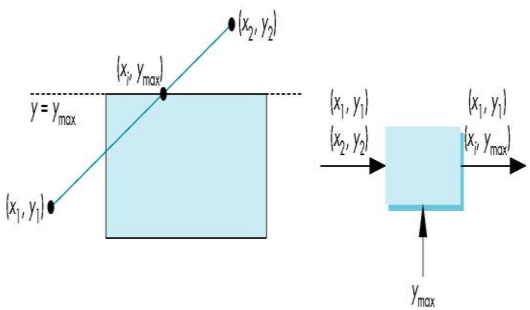
The University of New Mexico

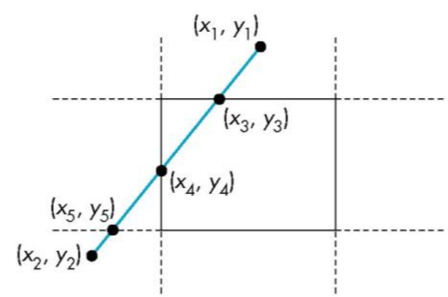
Polygon Clipping(cont.)

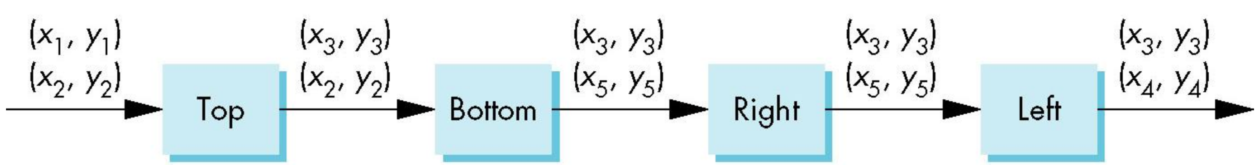
- Sutherland-Hodgeman Polygon Clipping
  - Clipping against each side of window is independent of other sides



  - Can use four independent clippers in a pipeline



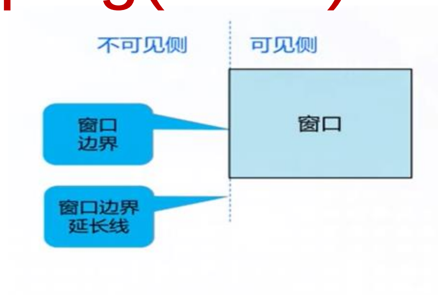




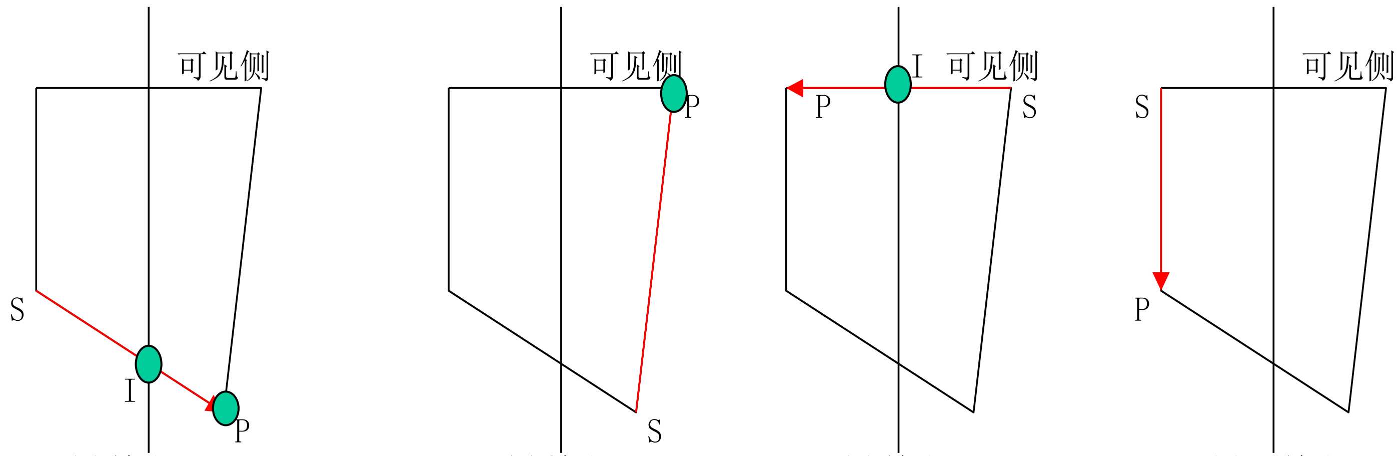
Polygon Clipping(cont.)

Sutherland-Hodgeman Polygon Clipping(cont.)

How to output intersection and vertex?



Notice: 不可重复输出线段的可见端点!

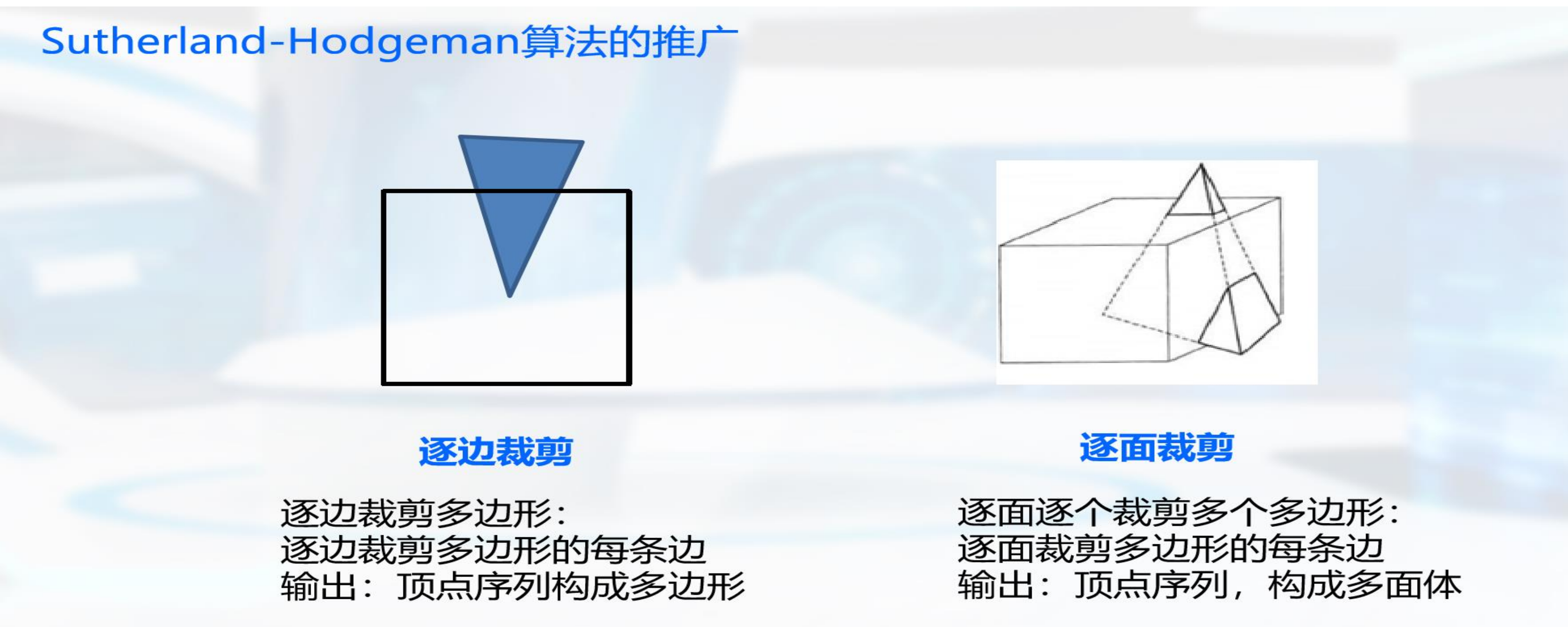


(a)输出I、P　(b)输出P　(c)输出I　(d)不输出

此图中，输出“线段与裁剪边的交点”和“可见边的结束顶点”

Polygon Clipping(cont.)

- Sutherland-Hodgeman Polygon Clipping(cont.)
  - Three Dimensions: add front and back clippers
    - Strategy used in SGI Geometry Engine



Outline

- **Clipping Algorithm**
  - **Clipping Line Segments**
    - Cohen-Sutherland Line Clipping编码裁剪算法
    - **Liang-Barsky Line Clipping梁永栋裁剪算法**
  - Clipping Polygons
    - Sutherland-Hodgeman Polygon Clipping 逐边裁剪算法
- **Hidden Surface Removal Algorithm**
  - **Object Space Approach**
    - Back-face culling后向面剔除（着色前就消隐掉）
  - **Image Space Approach**
    - Depth-buffer 深度缓存算法（光栅化渲染中的消隐算法）
    - Ray-casting光线投射（光线跟踪渲染中的消隐算法）

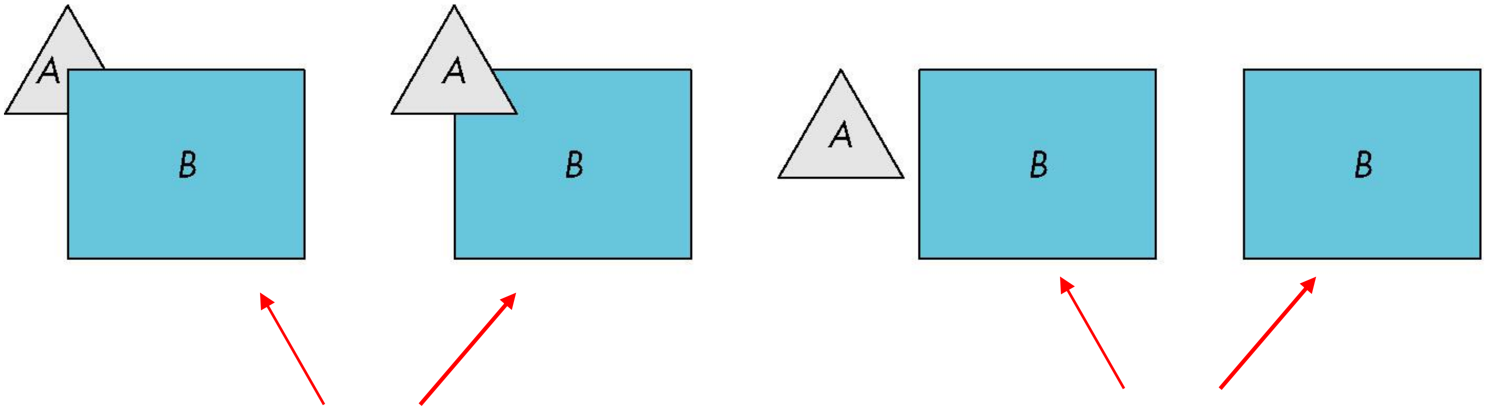
Hidden Surface Removal

- hidden-surface removal（隐藏面消除）has much in common with Clipping（裁剪），In both cases, we are trying to remove objects that are not visible to the camera
  （消隐和裁剪都是去除相机不可见的对象）

- Often we can use“visibility可见性”or “occlusion testing阻塞测试” early in the process to eliminate as many polygons as possible before going through the entire pipeline
  （通常，越早使用使用“可见性”测试，，可在通过整个管道之，　尽可能消除不可见的多边形）

Hidden Surface Removal(cont.)

- Object-space approach对象空间算法
  - use pairwise testing between polygons (objects)
  - Worst case complexity O(n^2) for n polygons



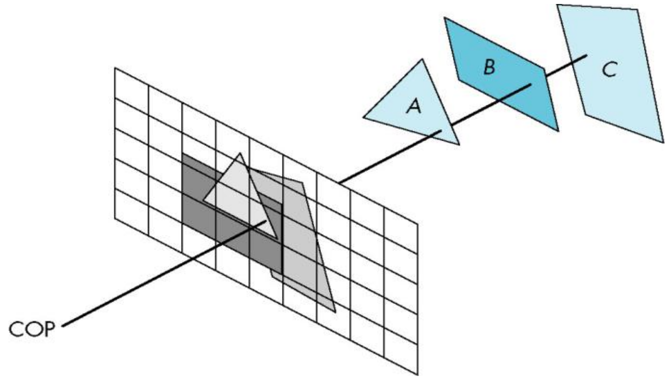
partially obscuring掩盖 can draw independently

Image Space Approach

➢Image Space Approach：

Idea：Look at each projector (nm for an n x m frame buffer) and find closest of k polygons，Complexity O(nmk)

从投影中心点（COP即视点）出发，与m*n分辨率的每个像素发出投影线(projector)，与场景中的k个多边形求交，再判定每个像素的颜色。复杂度O(nmk)



✓在像素级别上确定“可见性”

✓Algorithms：**Depth Buffer，Ray-casting**

Outline

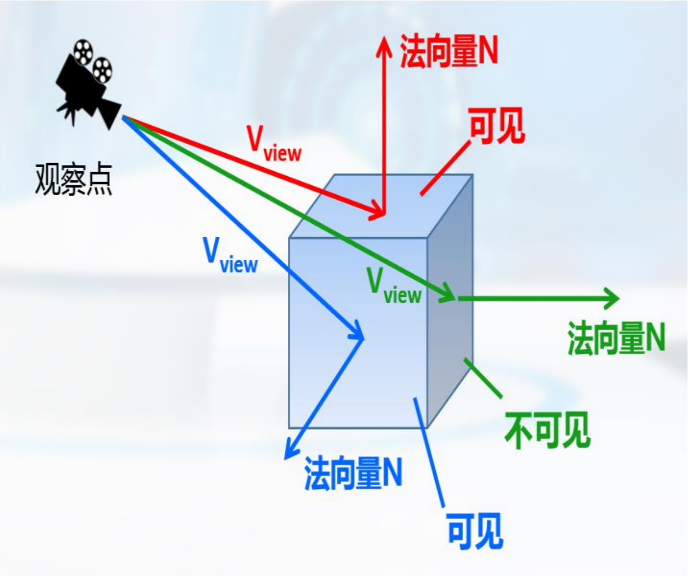
- **Clipping Algorithm**
  - **Clipping Line Segments**
    - Cohen-Sutherland Line Clipping编码裁剪算法
    - **Liang-Barsky Line Clipping梁永栋裁剪算法**
  - Clipping Polygons
    - Sutherland-Hodgeman Polygon Clipping 逐边裁剪算法
- **Hidden Surface Removal Algorithm**
  - **Object Space Approach**
    - Back-face culling后向面剔除（裁剪前就消掉）
  - Image Space Approach
    - Depth-buffer 深度缓存算法（光栅化渲染中的消隐算法）
    - Ray-casting光线投射（光线跟踪渲染中的消隐算法）

Back-Face Culling

The University of New Mexico

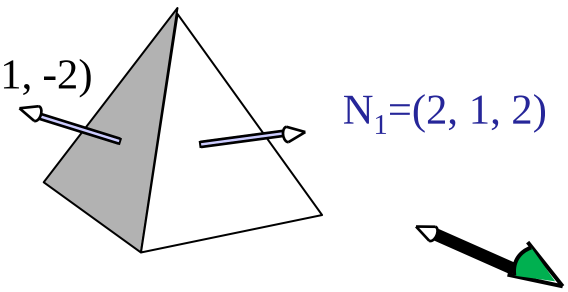
➢Back-Face Removal (Culling)后向面消除（剔除）

- 对每个凸多边形面，其外法向量n和观察方向v的夹角若小于90度，则是后向面back-Face，可判定为“不可见”，直接剔除culling
- Back face判定的计算方法：
  - face is invisible iff $90 \geq \theta \geq -90$,
  - face is invisible $\cos\theta \geq 0$ or $\mathbf{v} \cdot \mathbf{n} \geq 0$ equivalently



Back-Face Culling (cont.)

➢ ***Example***



✓**solution :**

$N_1 \bullet V = (2, 1, 2) \bullet (0, 0, -5) = -10$, so $N_1 \bullet V < 0$,
→ N_1 is not back-face polygon, it is visible

$N_2 \bullet V = (-3, 1, -2) \bullet (0, 0, -5) = 10$, so $N_2 \bullet V > 0$,
→ N_2 is Back-face polygon, it is invisible, need culling!

Back-Face Culling (cont.)

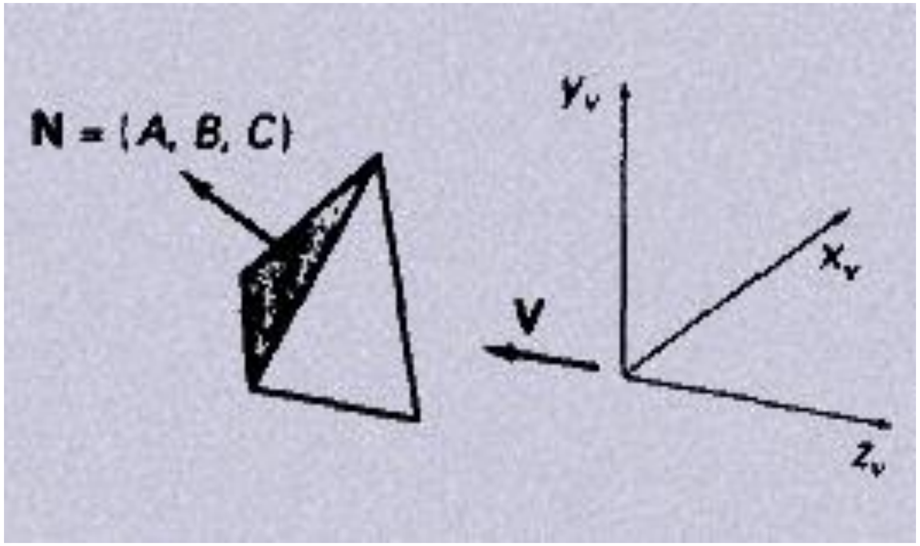
- Special case：

  若：V=(0, 0, Vz) , N=(Nx, Ny, Nz)=(A, B, C)

  则：**v•n**=VxNx+VyNy+VzNz=0*Nx+0*Ny+Vz*Nz=Vz • C

  If Vz • C >0, then this face is Back-face.

➢ **So , if Vz<0,need C<0 , this face is back face.**

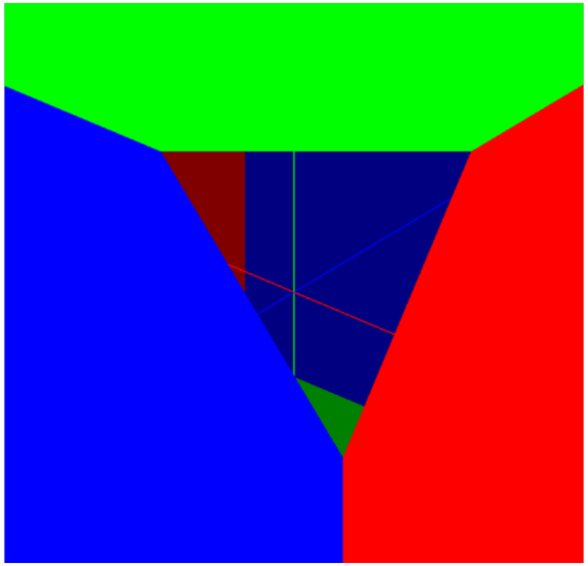


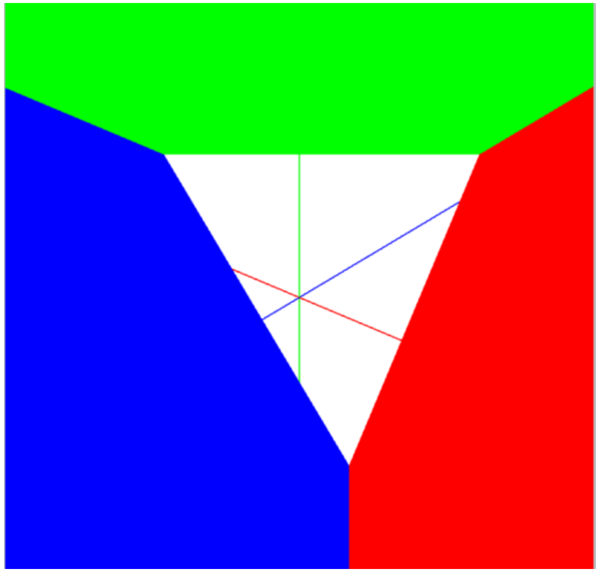
Back-Face Culling (cont.)

- In OpenGL we can simply enable culling
- **Open/Close culling function 开启/关闭剔除功能**
  - gl.enable (gl.CULL_FACE); //开启后向面剔除
  - gl.disable(gl.CULL_FACE); //关闭后向面剔除，Default
- **Choose which face to be culled 选择哪个面被剔除**
  - gl.cullFace(face);//face: gl.BACK or gl.FRONT;
    // default is gl.BACK

Back-Face Culling (cont.)

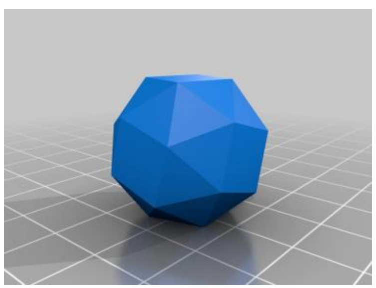
- 后向面剔除实例:
  - 左边图:没有开启后向面剔除，默认情况
    - gl.disable(gl.CULL_FACE); // Default
  - 右边图:开启了后向面剔除，可加快渲染速度。
    - gl.enable (gl.CULL_FACE);





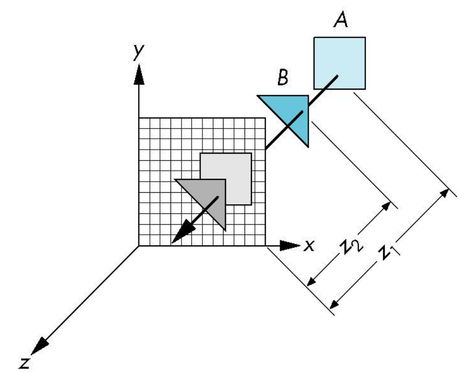
Back-Face Culling(cont.)

- 优点：
  - 在光栅化之前，可剔除掉物体的不会显示的后向面
- 局限：
  - 凹多面体的可见面或不同物体上的可见面的遮挡，不能进行判定消隐。



凸多面体

凹多面体



不能判定“可见面”之间的遮挡

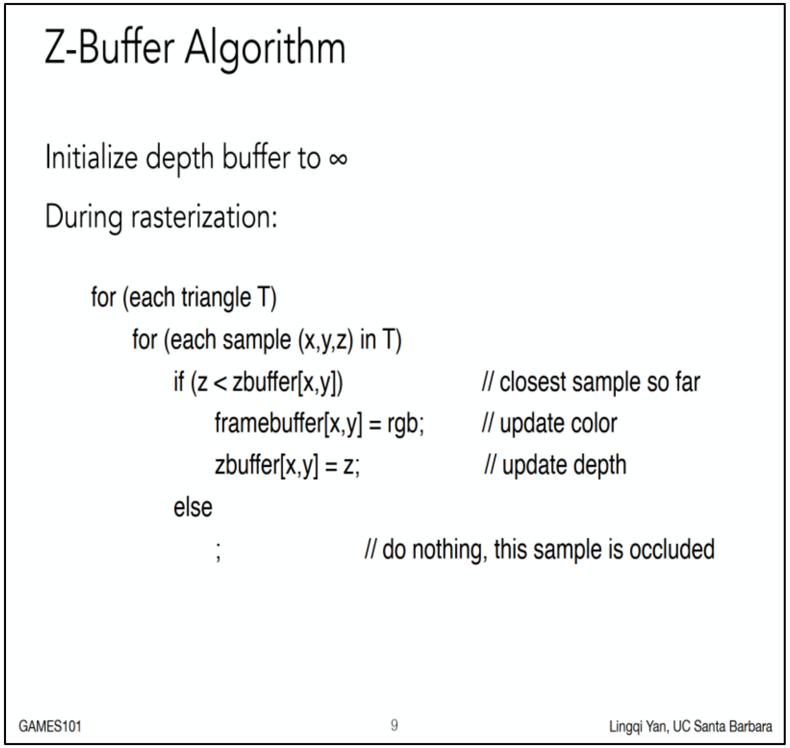
Outline

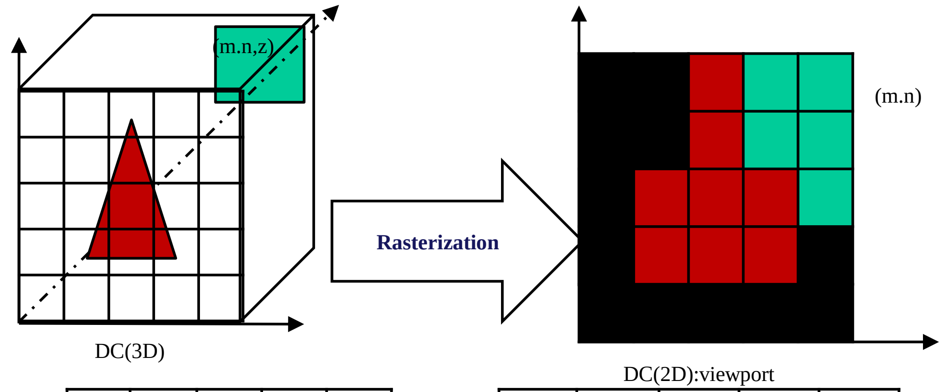
- **Clipping Algorithm**
  - **Clipping Line Segments**
    - Cohen-Sutherland Line Clipping编码裁剪算法
    - **Liang-Barsky Line Clipping梁永栋裁剪算法**
  - Clipping Polygons
    - Sutherland-Hodgeman Polygon Clipping 逐边裁剪算法
- **Hidden Surface Removal Algorithm**
  - Object Space Approach
    - Back-face culling后向面剔除(着色前就消隐掉)
  - **Image Space Approach**
    - **Depth-buffer 深度缓存算法(光栅化渲染中的消隐算法)**
    - Ray-casting光线投射(光线跟踪渲染中的消隐算法)

The University of New Mexico

Z-Buffer Algorithm

- Ref: GAME101 Lecture7（注：下例左手坐标系Z>0）
  - Use a buffer called the “z-buffer” or “depth buffer” to store the depth of the closest object at each pixel found so far
  - As we render each polygon, compare the depth of each pixel to depth in z buffer , If less, place shade of pixel in color buffer and update z buffer





99	99	3	5	5
99	99	3	5	5
99	3	3	3	5
99	3	3	3	99
99	99	99	99	99

z_buffer

000	000	100	010	010
000	000	100	010	010
000	100	100	100	010
000	100	100	100	000
000	000	000	000	000

color_buffer

Z-Buffer Algorithm

The University of New Mexico

例如下图:对场景中三角形A,B进行绘制填充颜色(即写帧缓存)(注意:本例中采用右手坐标系:Z<0)(Z1,Z2表示为深度)

1)若先光栅化填充A时,

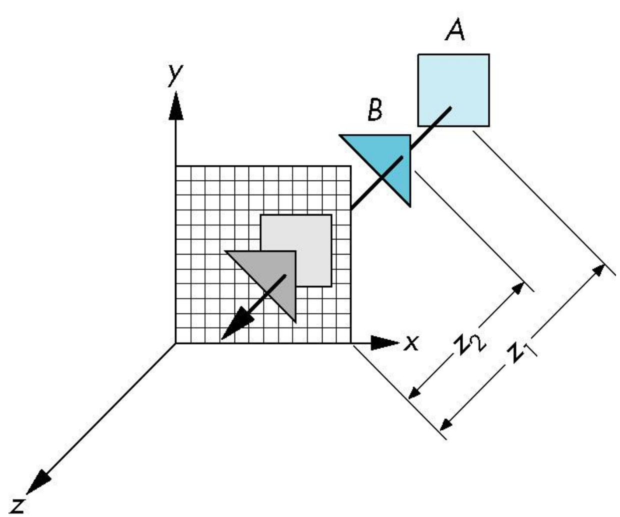
对像素pixel(x,y): 有z(x,y)=z1 < Dbuffer(x,y) = 初始化为最大深度

则: Dbuffer(x,y)=z1,ColorBuffer(x,y)="浅蓝"

2) 再光栅化填充B时,

对像素pixel(x,y): 有z(x,y)=z2 < Zbuffer(x,y) =z1

则: Dbuffer(x,y)=z2,ColorBuffer(x,y)="深蓝"



Z-Buffer Algorithm（cont.）

- WEBGL中实现深度检测算法：
  - 设置Z-BUFFER初值(无穷大（左手坐标），无穷小（右手坐标）

    gl.clear(gl.DEPTH_BUFFER_BIT);

    //也常和颜色缓存一起进行初始化：设置背景颜色和深度值

    //gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  - 开启Z-BUFFER算法

    gl.enable(gl.DEPTH_TEST);

  - 关闭Z-BUFFER算法

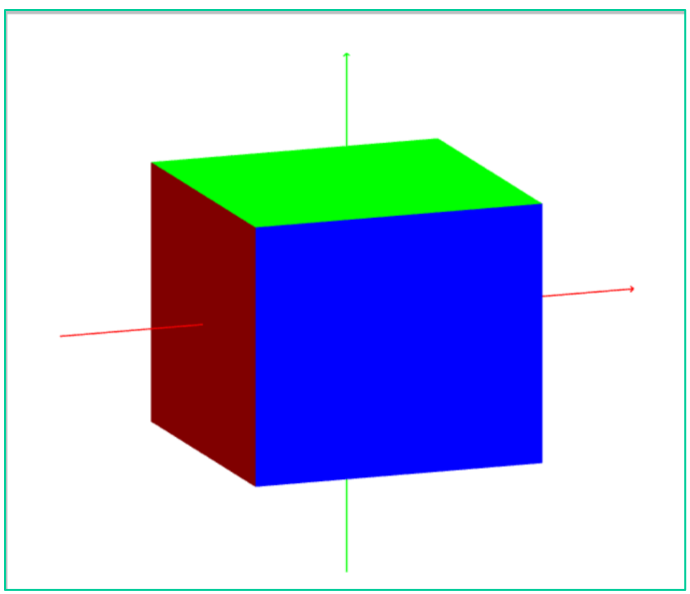
    gl.disable(gl.DEPTH_TEST);

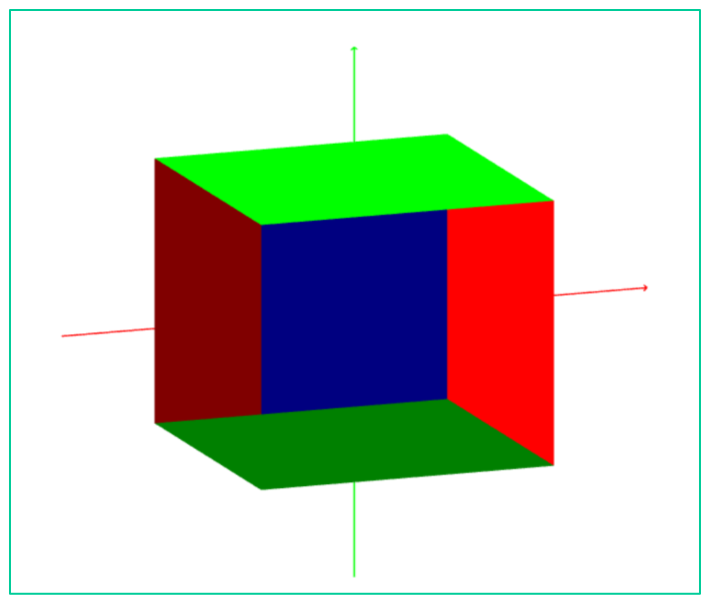
The University of New Mexico

Z-Buffer Algorithm（cont.）

➢深度缓存的实例

- ➢左边:开启了深度缓存消隐算法
  - ➢渲染结果与图元绘制的先后顺序“”无关“”，正确显示
- ➢右边:没有开启深度缓存算法(默认)
  - ➢渲染结果与图元绘制的先后顺序 “相关”，错误显示





```
function generateCube()
{
    quad( 1, 0, 3, 2 ); //Z正-前
    quad( 4, 5, 6, 7 ); //Z负-后

    quad( 2, 3, 7, 6 ); //X正-右
    quad( 5, 4, 0, 1 ); //X负-左

    quad( 6, 5, 1, 2 ); //Y正-上
    quad( 3, 0, 4, 7 ); //Y负-下
}
```

Z-Buffer Algorithm（cont.）

优点：

- 不需要考虑整个场景的多边形面之间关系
- 简单稳定，利于硬件实现

缺点：

- 需要一个分辨率大小的额外的Z缓存空间。
- 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，但可以优化。

Outline

- **Clipping Algorithm**
  - **Clipping Line Segments**
    - Cohen-Sutherland Line Clipping编码裁剪算法
    - **Liang-Barsky Line Clipping梁永栋裁剪算法**
  - Clipping Polygons
    - Sutherland-Hodgeman Polygon Clipping 逐边裁剪算法
- **Hidden Surface Removal Algorithm**
  - Object Space Approach
    - Back-face culling后向面剔除（着色前就消隐掉）
  - **Image Space Approach**
    - **Depth-buffer 深度缓存算法（光栅化渲染中的消隐算法）**
    - Ray-casting光线投射（光线跟踪渲染中的消隐算法）

Ray-casting Algorithm

光线投射:

模拟人的视觉效果，沿视线的路径跟踪场景的可见面，计算颜色并写入帧缓存

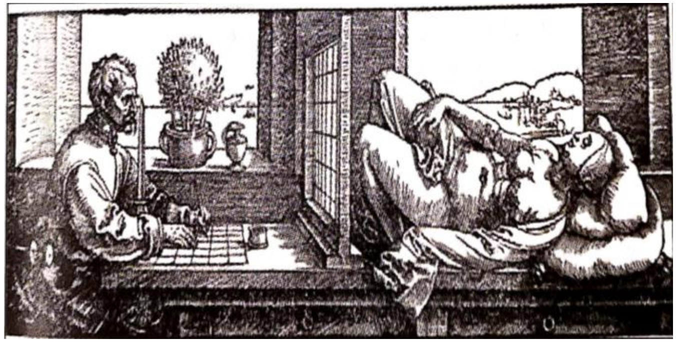
（光线投射可以直接实现可见性判定而实现了消隐）

算法步骤：

1. 通过视点相机和投影平面上的象素点作一入射线。“发射投影线或发射光线”

2. 将任一投影线与场景中的所有多边形求交。“求交”（关键步骤)

3. 若有交点，则将所有交点按z值的大小进行排序，取最近交点（离视点/相机最近）所属多边形的颜色作为像素颜色；若没有交点，则取背景的颜色。“输出最近交点颜色”



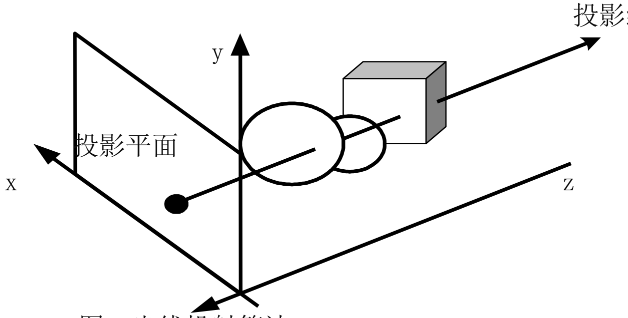


图 光线投射算法

Ray-casting Algorithm（cont.)

- **“求交”举例： “光射线和平面求交”**

1、假设射线起点为（x0,y0,z0）,方向向量为（d1,d2,d3）,则射线为

x=d1*t+x0

y=d2*t+y0

z=d3*t+z0

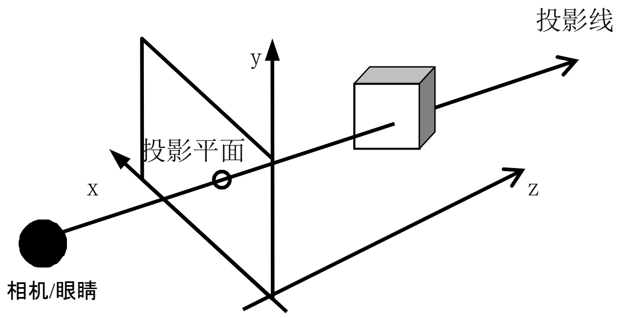


图　光线投射算法

2、假设物体表面是“平面”，设平面方程为

ax+by+cz+d=0.

3、将光线“射线”代入物体表面“平面方程”得

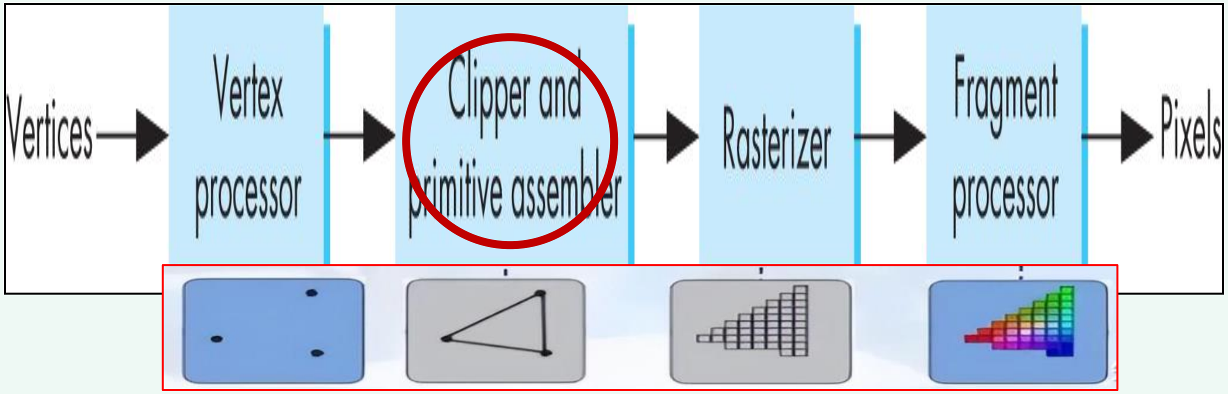
a(d1*t+x0)+b(d2*t+y0)+c(d3*t+z0)+d=0

即：(a*d1+b*d2+c*d3)*t+(a*x0+b*y0+c*z0+d)=0 //关于t的一次方程

- 若a*d1+b*d2+c*d3=0时，射线与平面平行，无交点。
- 否则解出t=-(a*x0+b*y0+c*z0+d)/(a*d1+b*d2+c*d3)
  - 当t<0时，交点在射线的反向延长线上，视线看不到不求交。
  - 当t>=0时，交点在射线的正向上，视线看得见，算出交点。

Summary

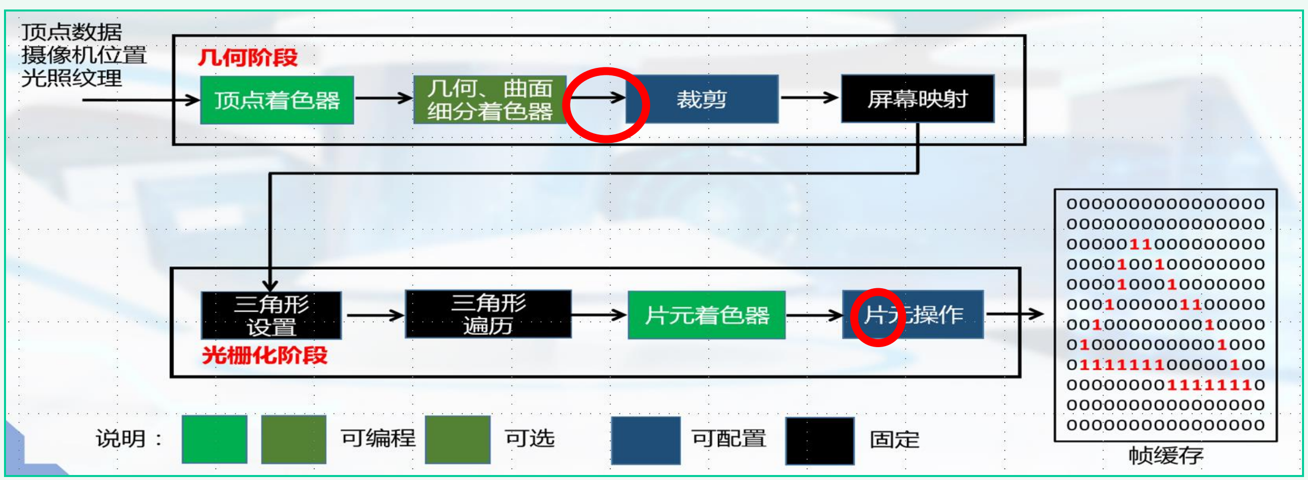
- 裁剪和图元组装：视见体外对象被剔除
  - 线段的裁剪算法
    - Cohen-Sutherland Line Clipping编码裁剪算法
    - Liang-Barsky Line Clipping梁永栋裁剪算法
  - 简单多边形的裁剪算法
    - Sutherland-Hodgeman Polygon Clipping 逐边裁剪算法



Summary(cont.)

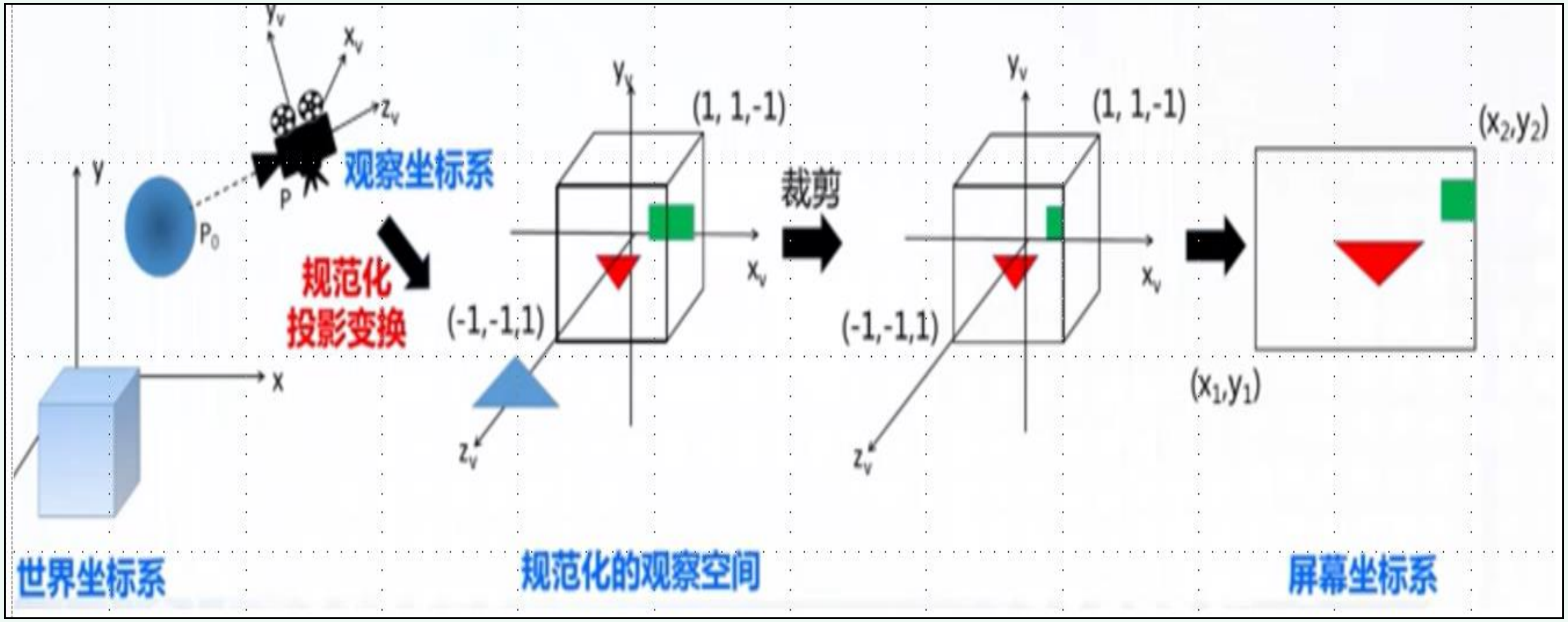
➢ 消隐算法：剔除后限免和消除隐藏的对象

- 对象空间中的消隐算法
  - Back-face culling后向面剔除（着色前就消隐掉）
- 图像空间中的消隐算法
  - Depth-buffer 深度缓存算法（光栅化渲染中的消隐算法）
  - Ray-casting光线投射（光线跟踪渲染中的消隐算法）



Summary（cont.)

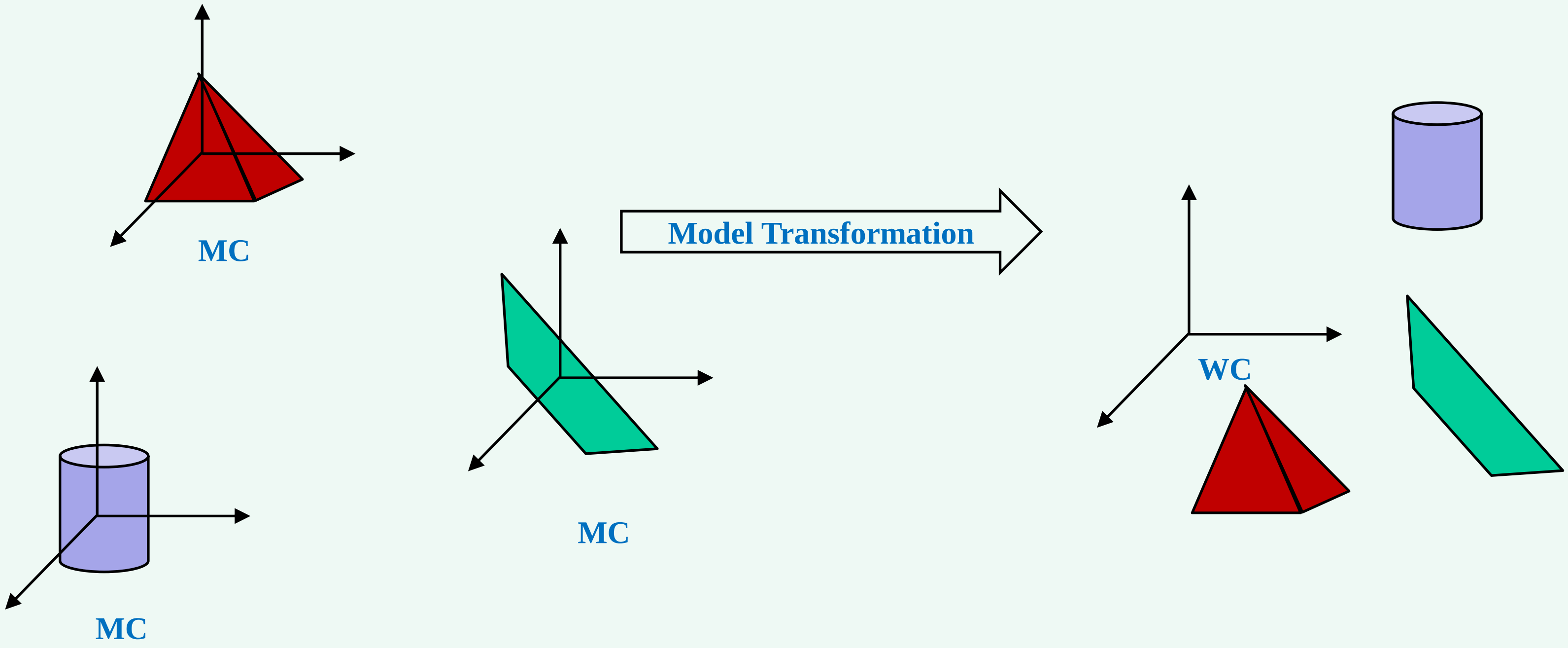
- ➢View Process Transformations(观察流程中的变换）
- ➢Clipping and Primitive Assembling（裁剪和图元组装）
- ➢Hidden Removal（隐藏（对象）的消除）



Summary(cont.)

1.from MC to WC

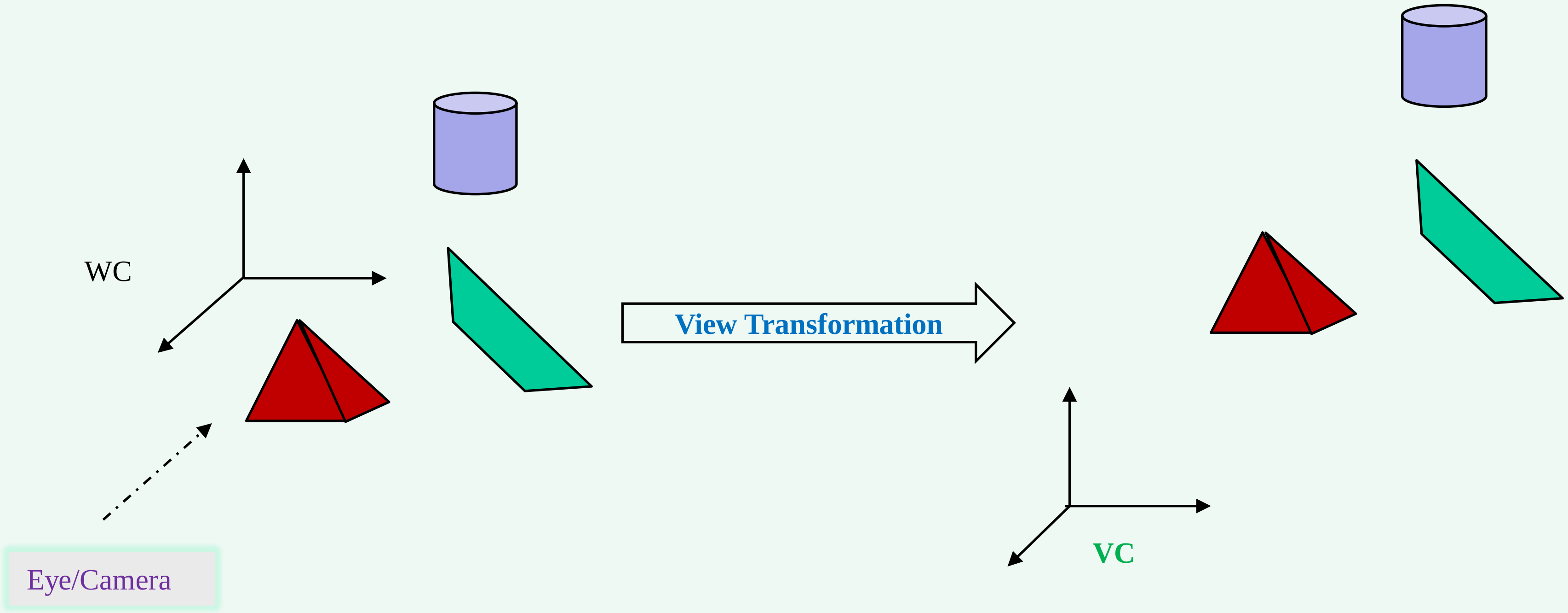
- Model Transformation (Instance Transformation)
  - **Common : TRS**



Summary(cont.)

2. from WC to VC

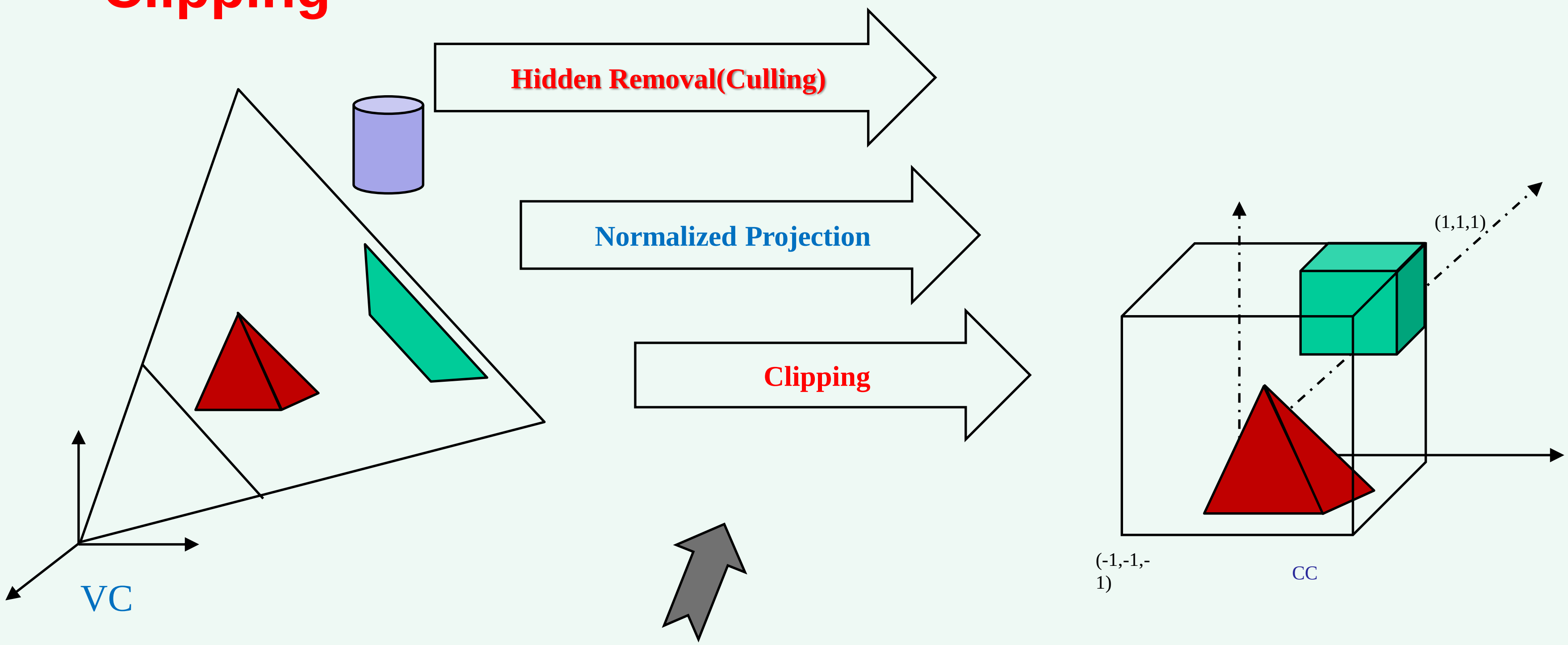
- View Transformation
  - *Common Lookat() =* ***R(u,v,n)T(-eye)***



Summary(cont.)

3. from VC to CC

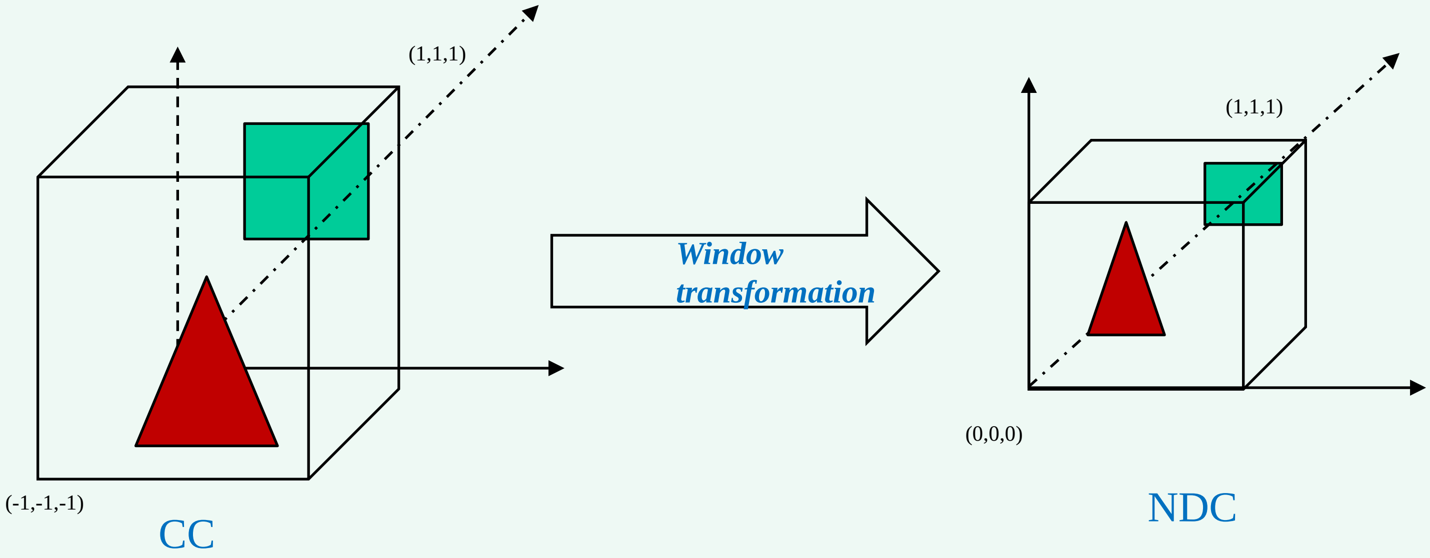
- **Hidden Removal(Culling剔除不可见面)**
- View Normalization Projection Transformation
- **Clipping**



Summary(cont.)

4. from CC to NDC

- *Window Viewport Transformation*



Summary(cont.)

5.from NDC to DC

- ***Window Viewport Transformation***
- *Perspective Division, Orthogonal Projection,* ***Rasterization***
- ***Hidden Removal Algorithm (Z-buffer algorithm)***