華東理工大學

计算机图形学

2023年10月

奉贤校区

06

二维变换和二维观察

2D Transformation and View

绘制流水线

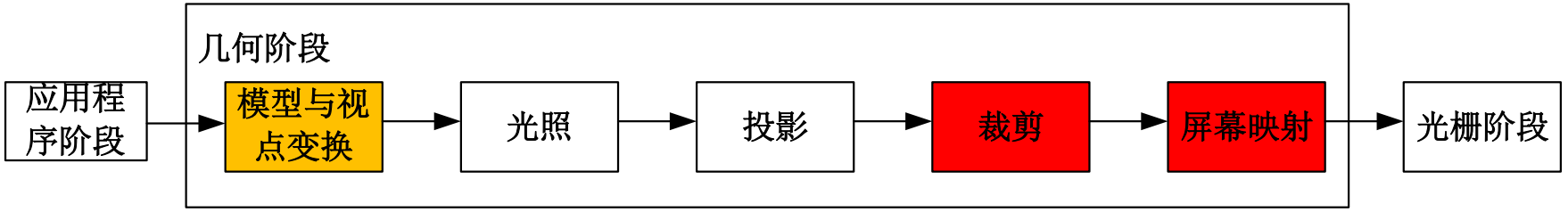
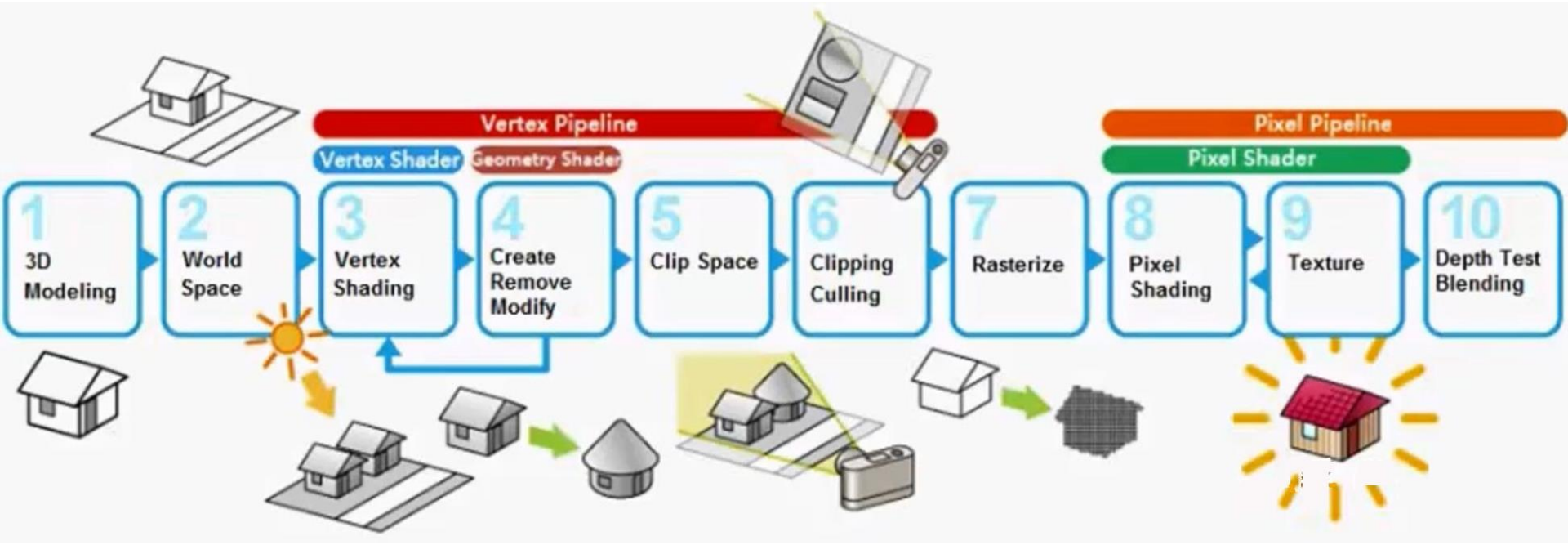


图2.22 绘制流水线的结构

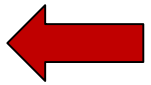

Vertex Pipeline
Vertex Shader
Geometry Shader
Pixel Pipeline
Pixel Shader
1
3D Modeling
2
World Space
3
Vertex Shading
4
Create Remove Modify
5
Clip Space
6
Clipping Culling
7
Rasterize
8
Pixel Shading
9
Texture
10
Depth Test Blending

问题

- 为啥用齐次坐标？
- 基本几何变换的主要特点？
- 矩阵乘法用结合律有没有问题？
- 复合变换的解决思路是算法中的什么思想？

二维变换

1. 基本几何变换
2. 二维图形几何变换的计算
3. 复合变换
4. 变换的性质



变换的性质

二维仿射变换是具有如下形式的二维坐标变换：

$$\begin{cases} x' = ax + by + m \\ y' = cx + dy + n \end{cases}$$

- 平移、比例、旋转、错切和反射等变换均是二维仿射变换的特例
- 任何常用的二维仿射变换总可以表示为这五种变换的复合。
- 本质上**一次线性变换**并接上**一个平移**，变换为另一个向量空间

变换的性质

- 仅包含旋转、平移和反射的仿射变换维持角度和长度的不变性；
- 比例变换可改变图形的大小和形状；
- 错切变换引起图形角度关系的改变，甚至导致图形发生畸变。

欧氏几何(初等几何)

研究图形在“搬动”之下保持不变的性质和数量
(统称不变性，如距离、角度、面积、体积等)

搬动 —— 正交变换 —— 对图形作有限次的平移、旋转、轴反射的结果

欧氏几何 —— 研究图形在正交变换下不变性质的科学

仿射几何

平行射影 —— 透视仿射对应

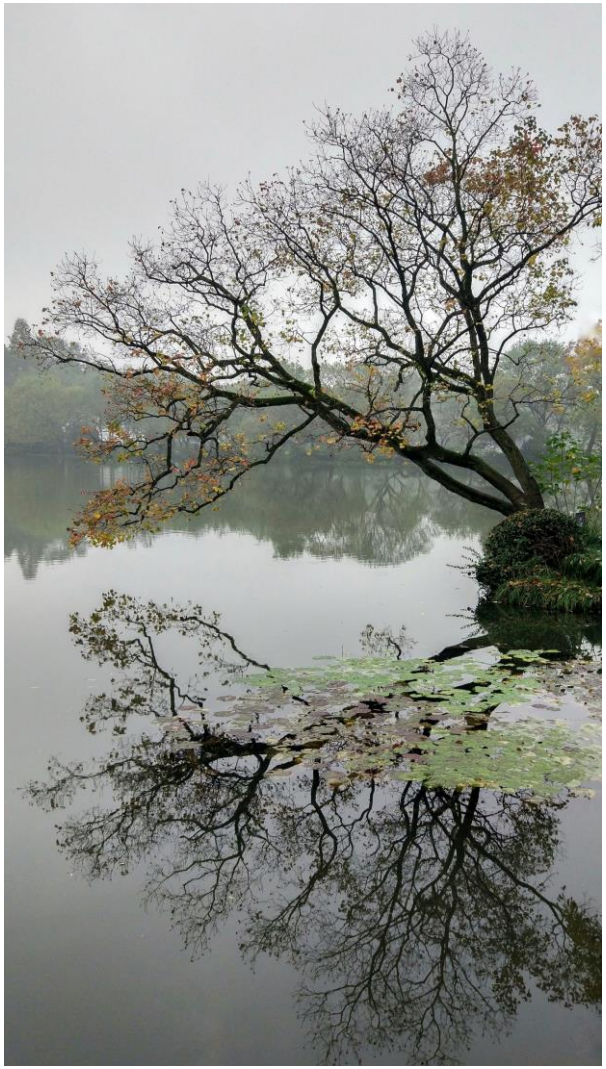
有限次平行射影的结果 —— 仿射变换

仿射几何 —— 研究图形在仿射变换下不变性质的科学

仿射不变性 —— 比如——平行性、两平行线段的比等等

二维观察

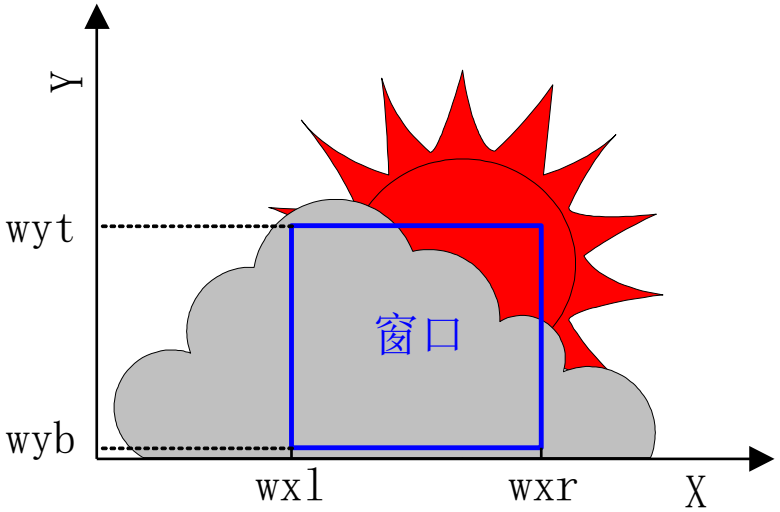
1. 基本概念
2. 二维观察变换
3. 二维裁剪算法



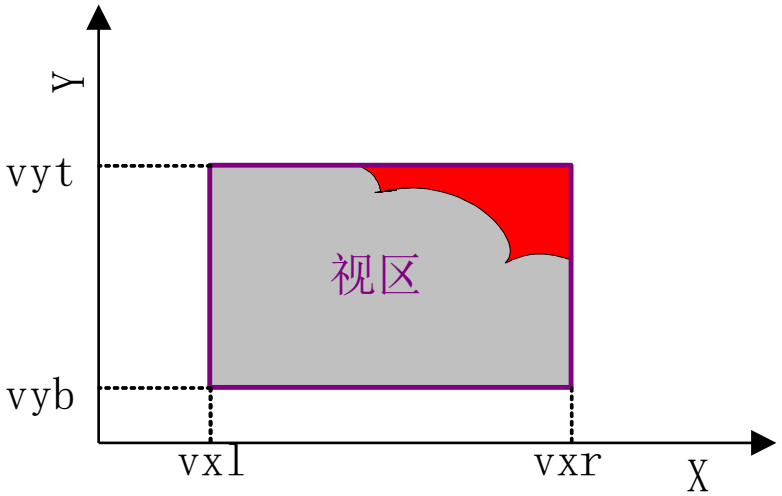
二维观察——基本概念

- 在计算机图形学中，将在用户坐标系中需要进行观察和处理的一个坐标区域称为**窗口（Window）**。
- 将窗口映射到显示设备上的坐标区域称为**视区（Viewport）**。

二维观察——基本概念



(a)用户坐标系中的窗口



(b)屏幕坐标系中的视区

- 要将窗口内的图形在视区中显示出来，必须经过将窗口到视区的变换（Window-Viewport Transformation）处理，这种变换就是观察变换（Viewing Transformation）。

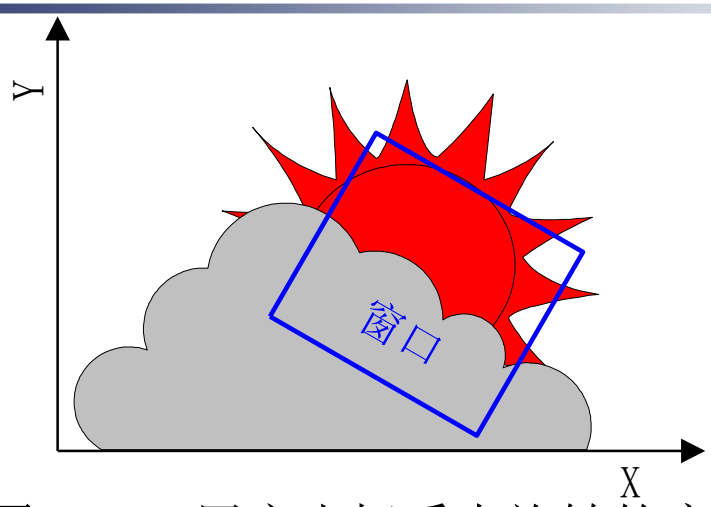
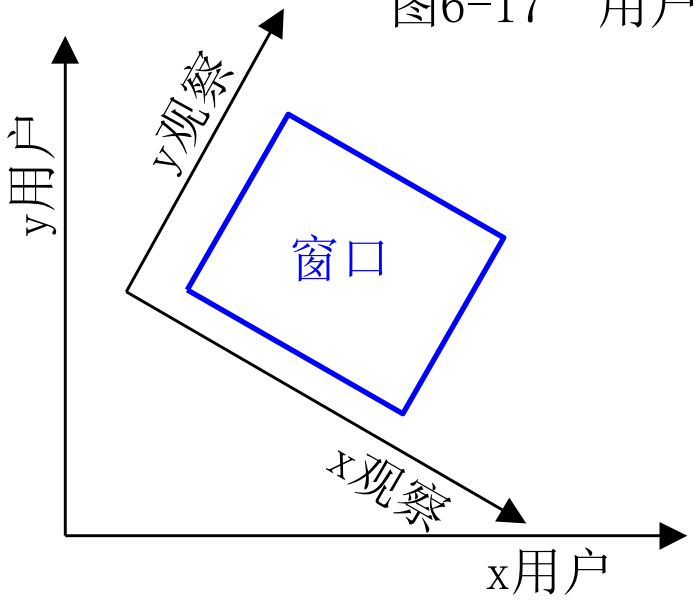
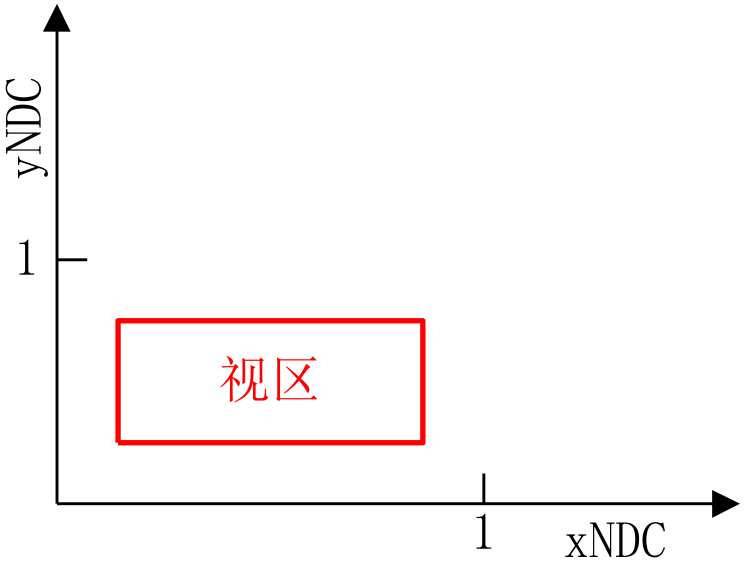


图6-17　用户坐标系中旋转的窗口



(a) 观察坐标系



(b) 规格化设备坐标系

二维观察——基本概念

- 观察坐标系(View Coordinate)是依据窗口的方向和形状在用户坐标平面中定义的直角坐标系。
- 规格化设备坐标系(Normalized Device Coordinate)也是直角坐标系，它是将二维的设备坐标系规格化到（0.0，0.0）到（1.0，1.0）的坐标范围内形成的。

二维观察——基本概念

- 引入了观察坐标系和规格化设备坐标系后，观察变换分为如下图所示的几个步骤，通常称为二维观察流程。

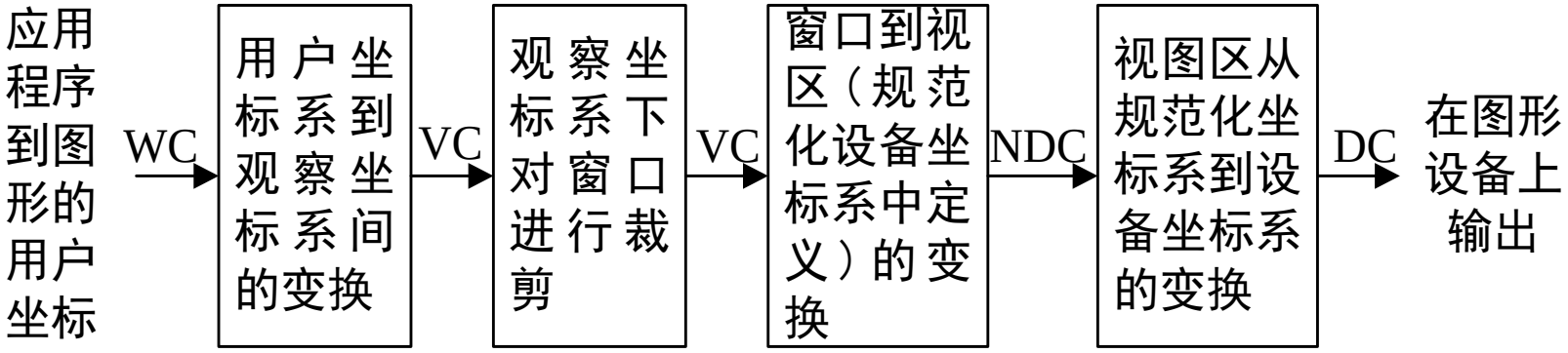
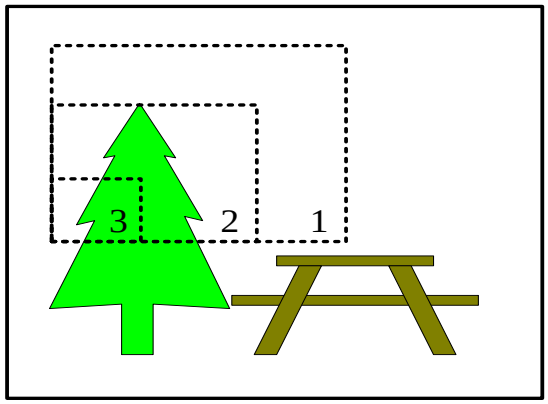
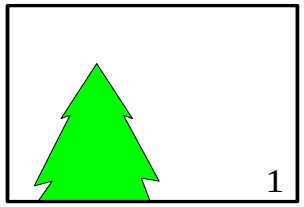


图6.18 二维观察流程

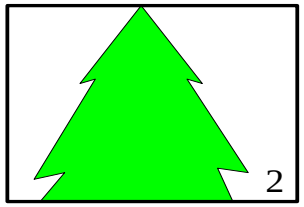
变焦距效果



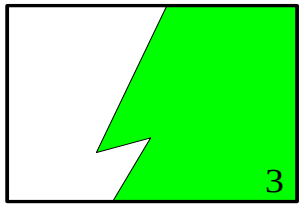
(a) 原图及变化的窗口



(b)与窗口对应的视区1



(c)与窗口对应的视区2



(d)与窗口对应的视区3

图6.19 变焦距效果（窗口变、视区不变）

整体放缩效果

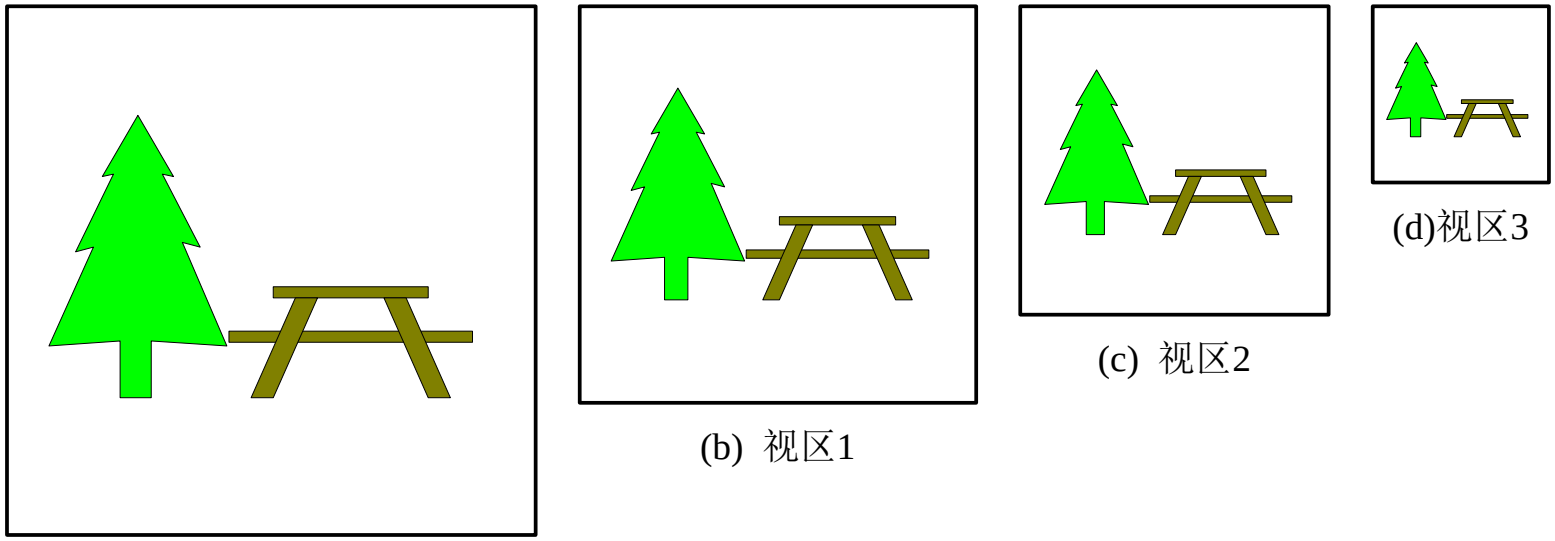


图6.20 整体放缩效果（窗口不变、视区变）

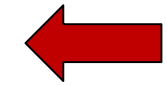
- 漫游效果

二维观察

1 基本概念

2 二维观察变换

3 二维裁剪算法

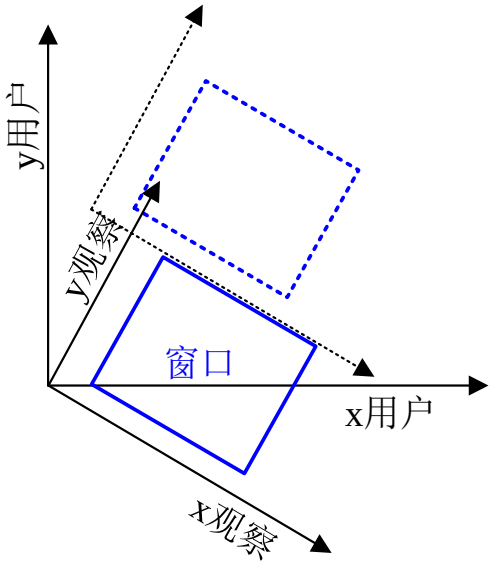


二维观察变换

- 用户坐标系到观察坐标系变换
- 二维裁剪
- 窗口到视区变换
- 视区从规格化设备坐标系到设备坐标系变换

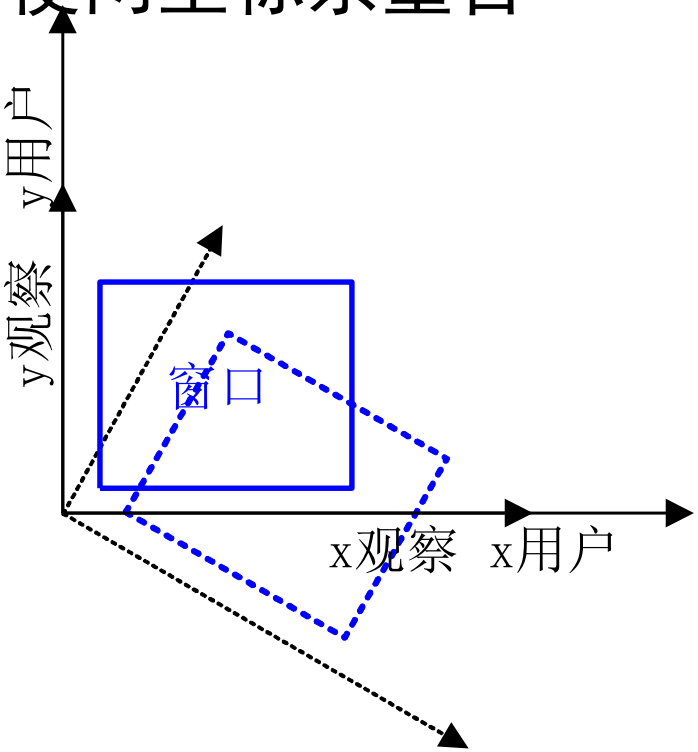
用户坐标系到观察坐标系的变换

- 用户坐标系到观察坐标系的变换分由两个变换步骤合成：
  - ◆将观察坐标系原点移动到用户坐标系原点；



用户坐标系到观察坐标系的变换

◆绕原点旋转使两坐标系重合



(b)　旋转变换

窗口到视区的变换

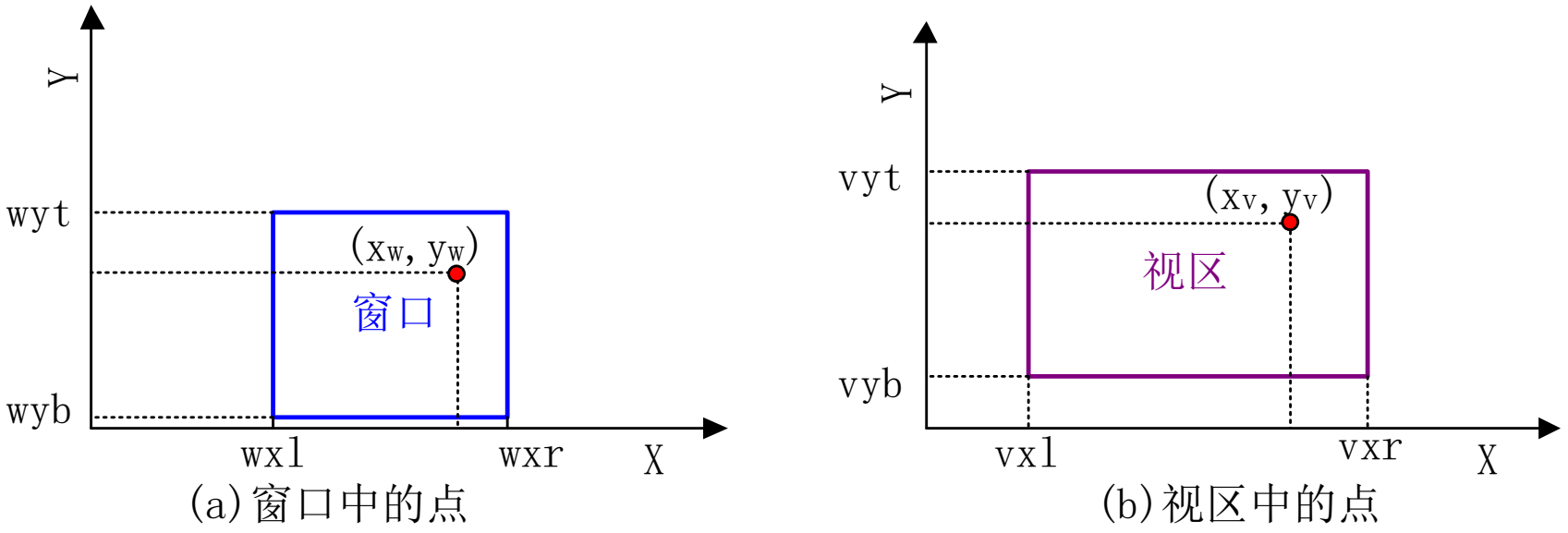


图6.21 窗口到视区的变换

窗口到视区的变换

- 要将窗口内的点（x_w,y_w）映射到相对应的视区内的点（x_v,y_v）需进行以下步骤：

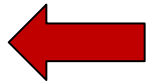
(1) 将窗口左下角点移至用户系统系的坐标原点；

(2) 针对原点进行比例变换；

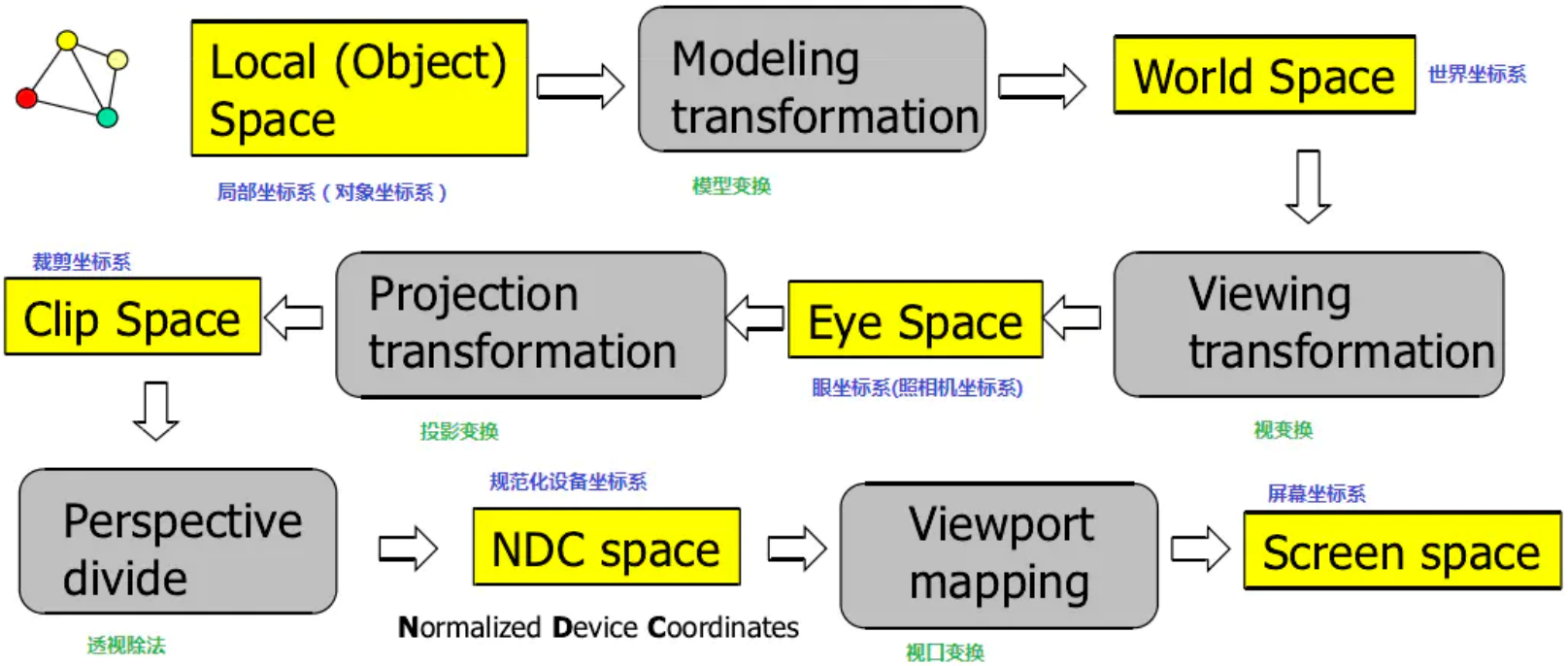
(3) 进行反平移。

OpenGL中的变换

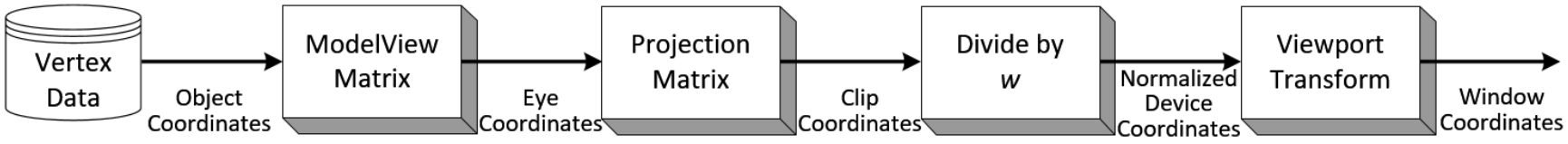
1. 变换种类
2. 模型视图矩阵
3. 矩阵操作
4. 投影变换



建模过程和OpenGL



建模过程



OpenGL坐标

原始矩阵计算

- Glm库

当前 V

```
glMatrixMode ( GL_MODELVIEW );
glLoadIdentity();
glMultMatrixf ( M ); //glScale*();
glMultMatrixf ( N ); //glRotate*()
glMultMatrixf ( L ); //glTranslate*();
glBegin ( GL_POINTS );
glVertex3f(V);
glEnd();
```

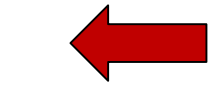
M×N×L×V， 始终是右乘

变换种类

- 视图变换：指定观察者或摄影机的位置；
- 模型变换：在场景中移动对象；
- **模型视图变换：描述视图变换与模型变换的对偶性；**
- **投影变换：对视见空间进行修剪和改变大小；**
- 视见区变换：对窗口的最终输出进行缩放；

OpenGL中的变换

1. 变换种类
2. 模型视图矩阵
3. 矩阵操作
4. 投影变换



模型视图矩阵

- 平移

  void glTranslated(f)(GLdouble x，GLdouble y，GLdouble z)；

- 旋转

  void glRotated(f)(GLdouble angle，GLdouble x，GLdouble y，GLdouble z)；

- 比例

  void glScaled(f)(GLdouble x，GLdouble y，GLdouble z)；

模型视图矩阵

- 视图变换函数（定义观察坐标系）

void gluLookAt（GLdouble eyex，GLdouble eyey，GLdouble eyez，GLdouble centerx，GLdouble centery，GLdouble centerz，GLdouble upx，GLdouble upy，GLdouble upz）；

模型视图矩阵

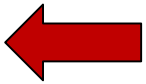
- OpenGL中使用列向量矩阵

$$p'=\begin{bmatrix}x'\\y'\\z'\\1\end{bmatrix}=T_{3D}\cdot p=\begin{bmatrix}a&d&h&l\\b&e&h&m\\c&f&j&n\\p&q&r&s\end{bmatrix}\cdot\begin{bmatrix}x\\y\\z\\1\end{bmatrix}$$

$$P'=T\cdot P=(T_n\cdot T_{n-1}\cdot T_{n-2}\cdots T_1)\cdot P\qquad(n>1)$$

OpenGL中的变换

1. 变换种类
2. 模型视图矩阵
3. 矩阵操作
4. 投影变换



矩阵操作

glMatrixMode(GLenum mode)；

参数mode用于确定将哪个矩阵堆栈用于矩阵操作。

GL_MODELVIEW：模型视图矩阵堆栈

GL_PROJECTION：投影矩阵堆栈

GL_TEXTURE：纹理矩阵堆栈

矩阵操作——单位矩阵

```
glTranslatef(10.0f, 0.0f, 0.0f);
    glutSolidSphere(1.0f, 15, 15);
glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0f, 10.0f, 0.0f);
    glutSolidSphere(1.0f , 15, 15);
```

矩阵堆栈

- OpenGL为模型视图矩阵和投影矩阵各维护着一个“矩阵堆栈”，可以把当前矩阵压到堆栈中保存它，然后对当前矩阵进行修改。把矩阵弹出堆栈即可恢复。使用的函数如下：

```
void glPushMatrix(void);
void glPopMatrix(void);
```

投影变换

- OpenGL中只提供了两种投影方式，一种是平行投影（正射投影），另一种是透视投影。在投影变换之前必须指定当前处理的是投影变换矩阵：

```
glMAtrixMode(GL_PROJECTION);
glLoadIdentity();
```

投影变换

- 平行投影：视景体是一个矩形的平行管道，也就是一个长方体，其特点是无论物体距离相机多远，投影后的物体大小尺寸不变。

  void glOrtho (GLdouble left，GLdouble right，GLdouble bottom，GLdouble top，GLdouble near，GLdouble far)；

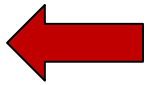
投影变换

void gluOrtho2D (GLdouble left，GLdouble right，GLdouble bottom，GLdouble top)；

- 一个特殊的正射投影函数，主要用于二维图像到二维屏幕上的投影。其near和far缺省值分别为-1.0和1.0，所有二维物体的Z坐标都为0.0。因此它的裁剪面是一个左下角点为（left，bottom）、右上角点为（right，top）的矩形。

二维观察

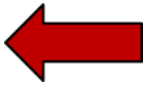
1. 基本概念
2. 二维观察变换
3. 二维裁剪算法



二维裁剪

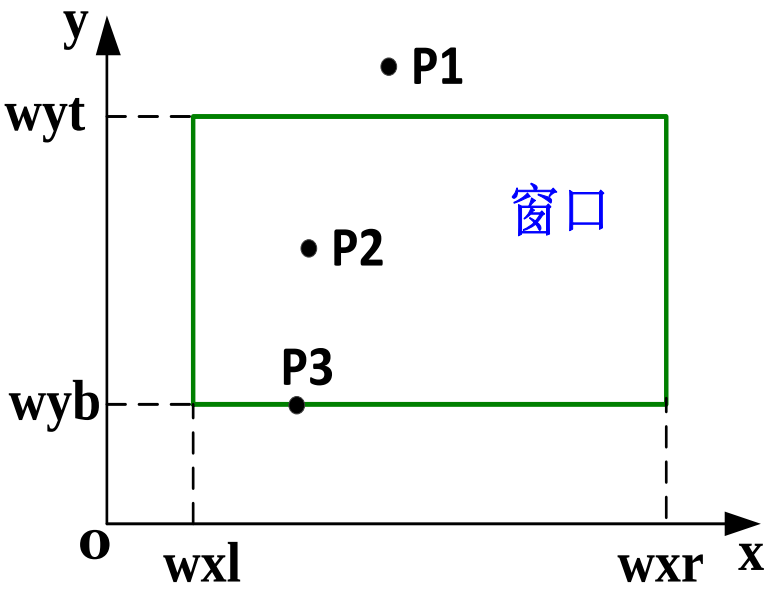
- 在二维观察中，需要在观察坐标系下对窗口进行裁剪，即只保留窗口内的那部分图形，去掉窗口外的图形。
- 假设窗口是标准矩形，即边与坐标轴平行的矩形，由上（y=wyt）、下（y=wyb）、左（x=wxl）、右（x=wxr）四条边描述。

二维裁剪



- 点的裁剪
- 直线段的裁剪
- 多边形的裁剪

二维点的裁剪

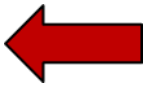


$$wxl \le x \le wxr,$$

且$$wyb \le y \le wyt$$

二维裁剪

- 点的裁剪
- 直线段的裁剪
- 多边形的裁剪

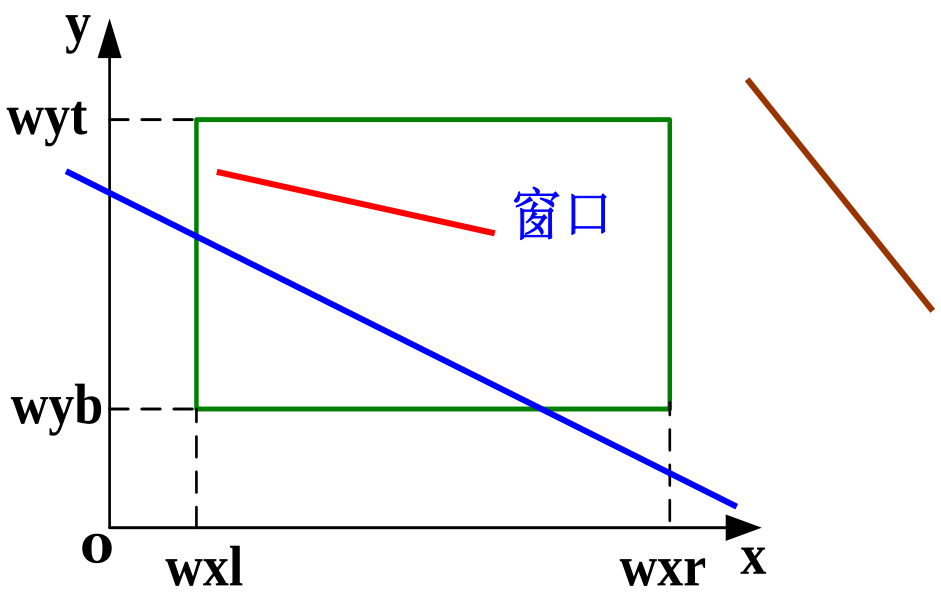


二维直线段的裁剪

已知条件：

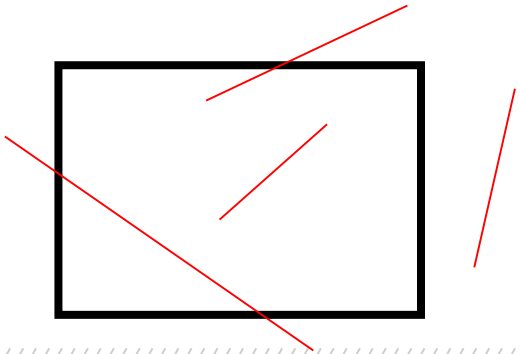
（1）窗口边界wxl，wxr，wyb，wyt的坐标值；

（2）直线段端点p_1p_2的坐标值x_1,y_1,x_2,y_2。



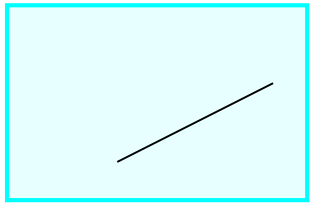
二维直线段的裁剪

- 对于矩形窗口
  - 任何直线至多只有一段处于该窗口之内
  - 即在此窗口范围内永远不会产生一条直线的两条或更多的可见部分线段
- 基本思想：
  - 判断直线与窗口的位置关系，确定该直线是完全可见、部分可见或完全不可见
  - 输出处于窗口内线段的端点，并显示此线段

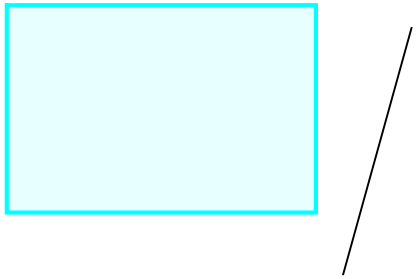


二维直线段的裁剪

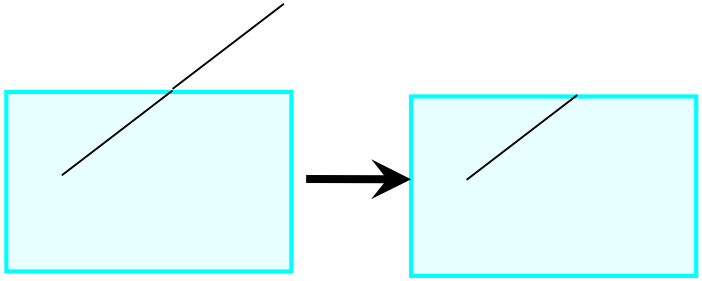
- 整条线在窗口之内
  - 不需剪裁，显示整条线段



- 整条线在窗口之外
  - 不需剪裁，不显示整条线段

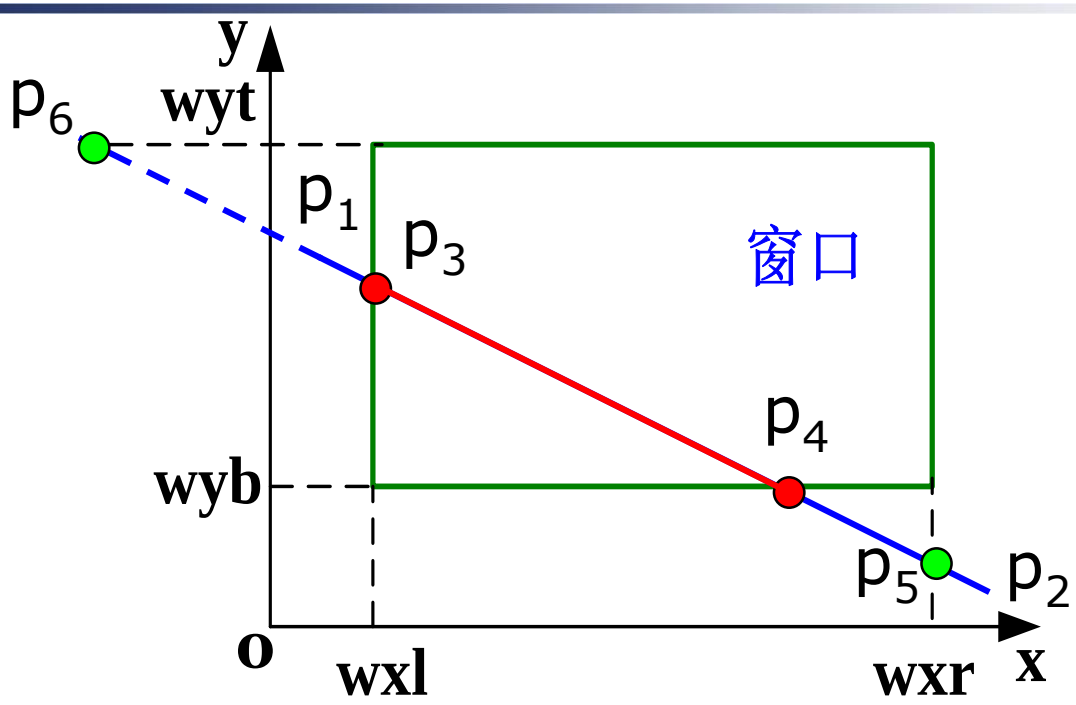


- 部分线在窗口之内
  - 求出线段与窗口边界的交点
  - 将窗口外的线段部分剪裁掉
  - 显示窗口内的部分



直线裁剪的效率策略

- 快速排除
  - 完全在窗口内
  - 完全在窗口外的直线
- 若是部分在窗口内的情况
  - 减少直线与窗口边界的求交次数和每次的求交计算量



- **实交点**：直线段与窗口矩形边界的交点；
- **虚交点**：处于直线段延长线或窗口边界延长线上的交点。

Cohen-Sutherland算法

编码：对于任一端点(x,y)，根据其坐标所在的区域，赋予一个4位的二进制码$D_3D_2D_1D_0$。

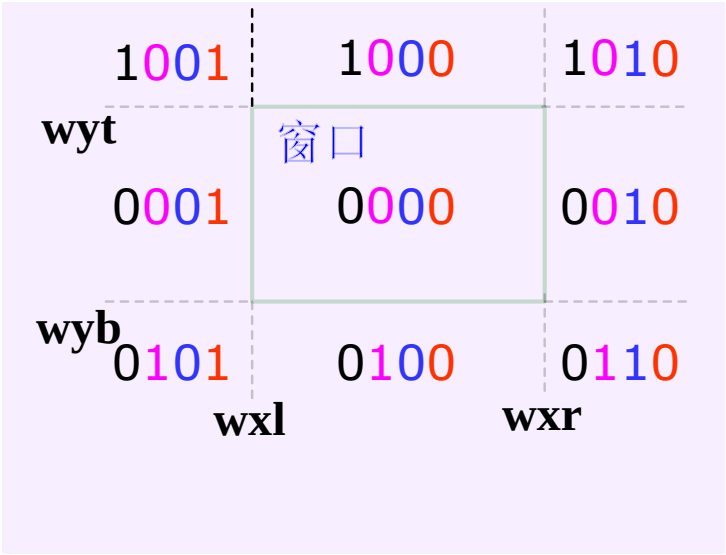
编码规则如下：

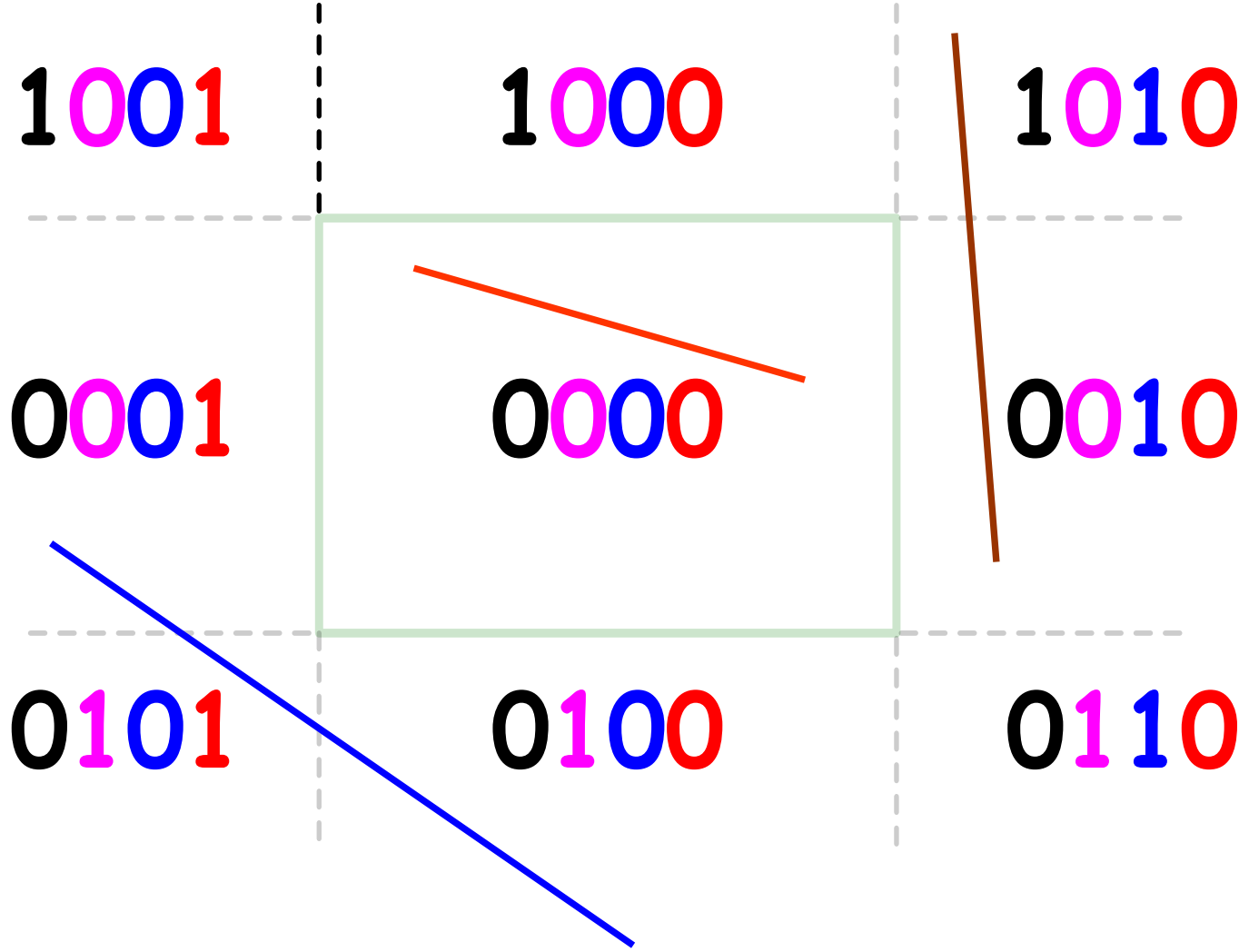
(1) 若x<wxl，D_0=1，否则D_0=0；

(2) 若x>wxr，D_1=1，否则D_1=0；

(3) 若y<wyb，D_2=1，否则D_2=0；

(4) 若y>wyt，D_3=1，否则D_3=0。




1001
1000
1010
0001
0000
0010
0101
0100
0110

Cohen-Sutherland算法

（1）判断

裁剪一条线段时，先求出直线段端点p_1和p_2的编码code1和code2，然后：

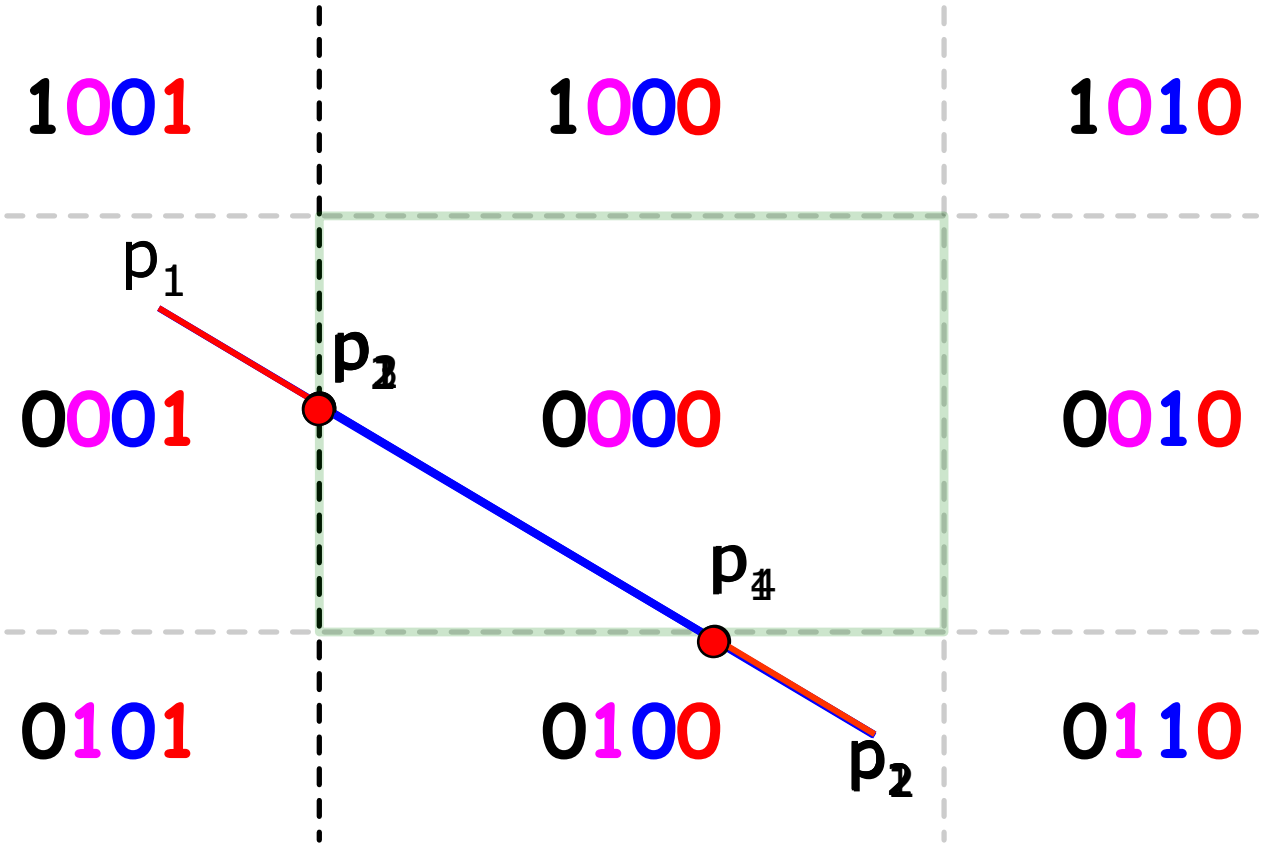
a.若code1|code2=0，对直线段简取之；

b.若code1&code2≠0，对直线段简弃之；

（2）求交

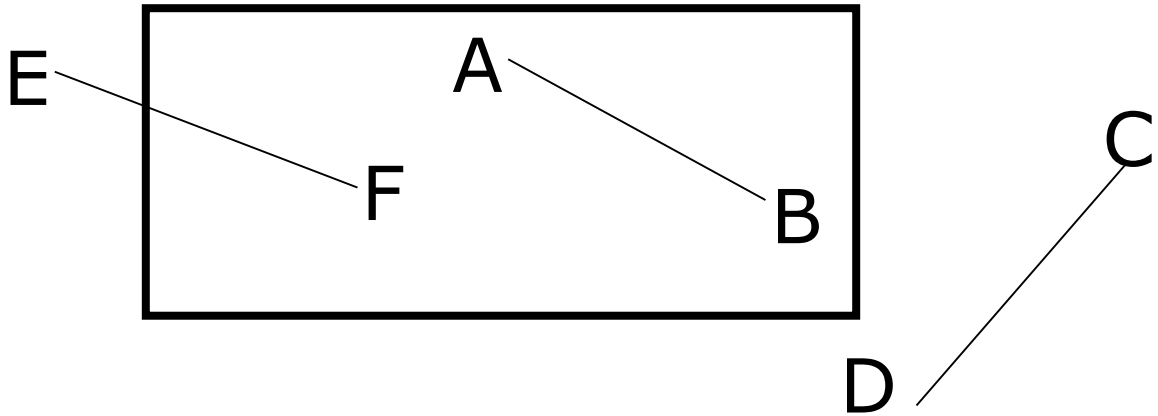
若上述判断条件不成立，则需求出直线段与窗口边界的交点。

Cohen-Sutherland算法



Cohen-Sutherland算法

例：如图，裁剪AB、CD、EF直线段。



解：根据编码算法，各直线段的端点四位二进制编码如下：

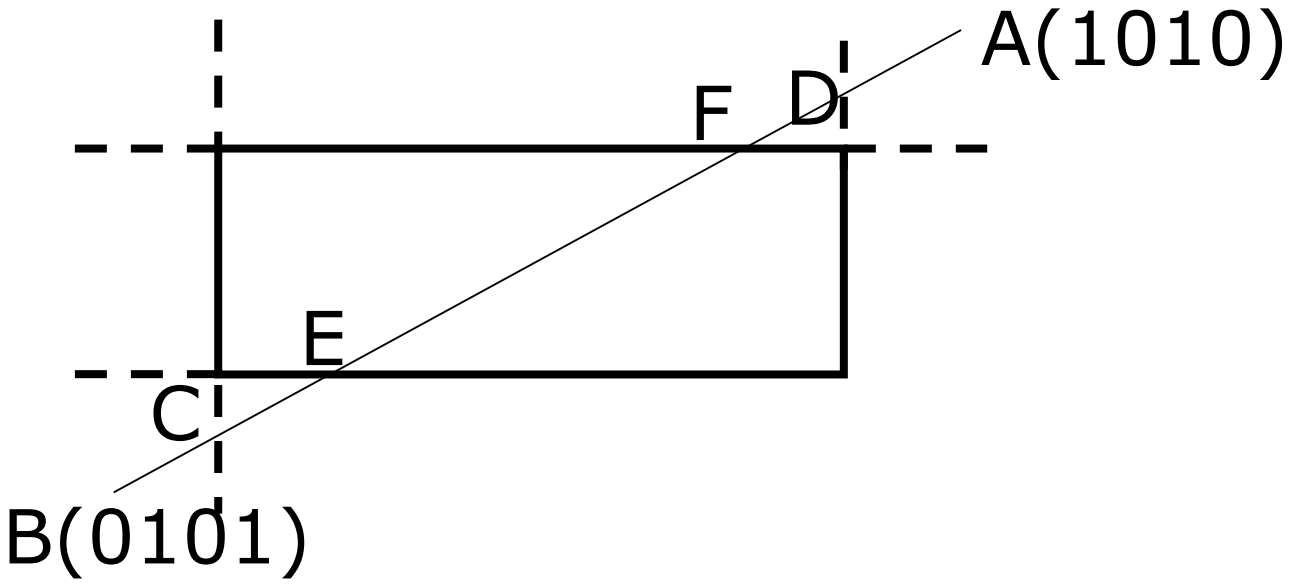
A:0000　B:0000　A,B编码全为0，AB直接保留

C:0010　D:0110　C,D编码进行逻辑与运算，结果非0，CD直线段直接舍弃

E:0001　F:0000　E,F编码进行逻辑与运算，结果为0，此时应求EF与窗口边界的交点

Cohen-Sutherland算法

- 求交
  - 将两个端点的编码$C_tC_bC_rC_l$进行逻辑或操作，
  - 根据其结果中1的位置来确定可能相交的窗口边
  - 求交按照固定的顺序来进行（左右下上或上下右左）
  - 一条线段与窗口最多求交4次



Cohen-Sutherland算法

a. 左、右边界交点的计算：$y = y_1 + k(x - x_1)$；

此时x取左边界或者右边界的值

b. 上、下边界交点的计算：$x = x_1 + (y-y_1)/k$。

其中，$k = (y_2-y_1) / (x_2-x_1)$。

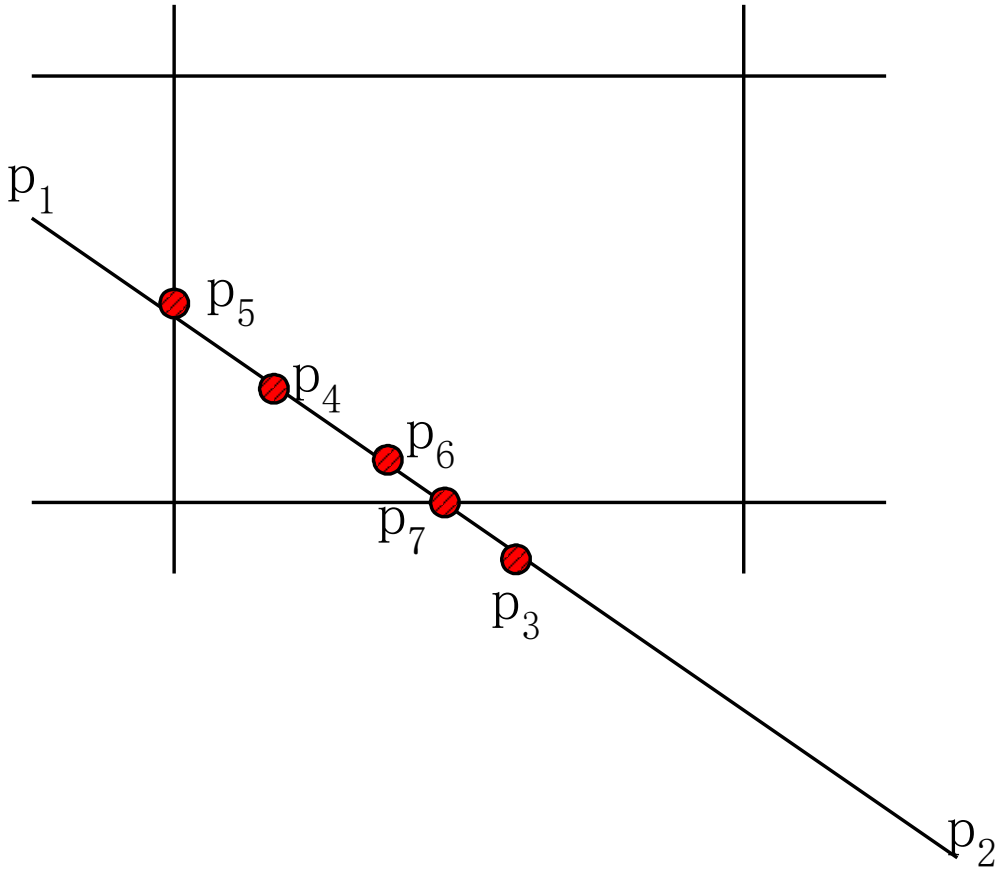
此时y取上边界或者下边界的值

Cohen-Sutherland算法

- 用编码方法实现了对完全可见和不可见直线段的快速接受和拒绝；
- 求交过程复杂，有冗余计算，并且包含浮点运算，不利于硬件实现。

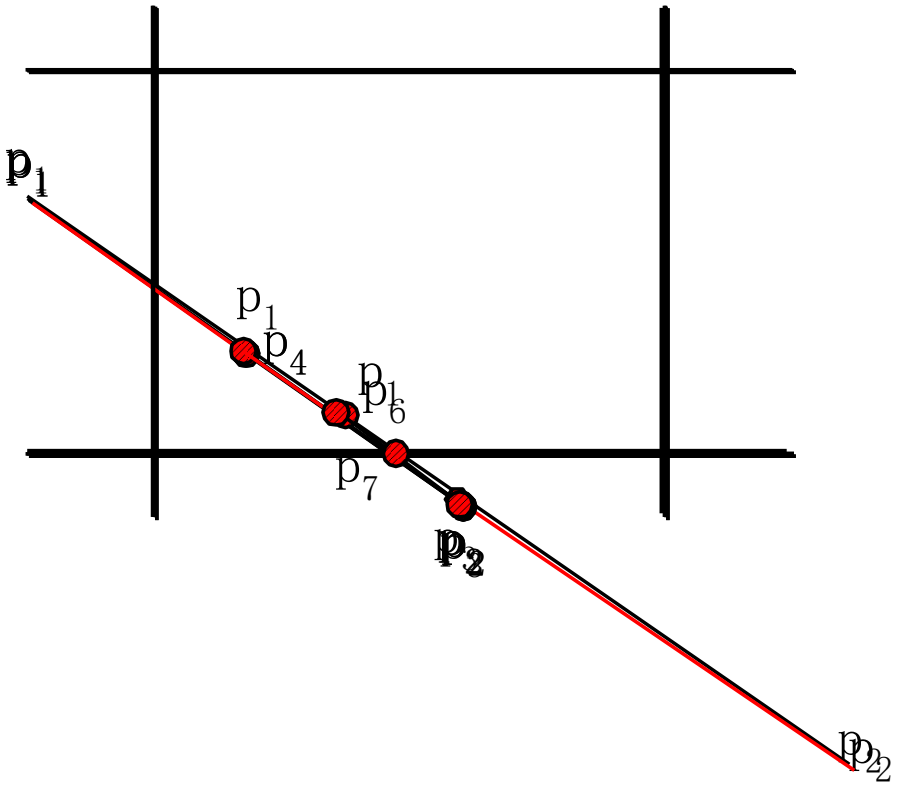
中点分割算法

- 中点分割算法的核心思想是通过二分逼近来确定直线段与窗口的交点。



中点分割算法

- 特点：主要计算过程只用到加法或位移运算，易于硬件实现，同时适合于并行计算。



梁友栋–Barsky裁剪算法

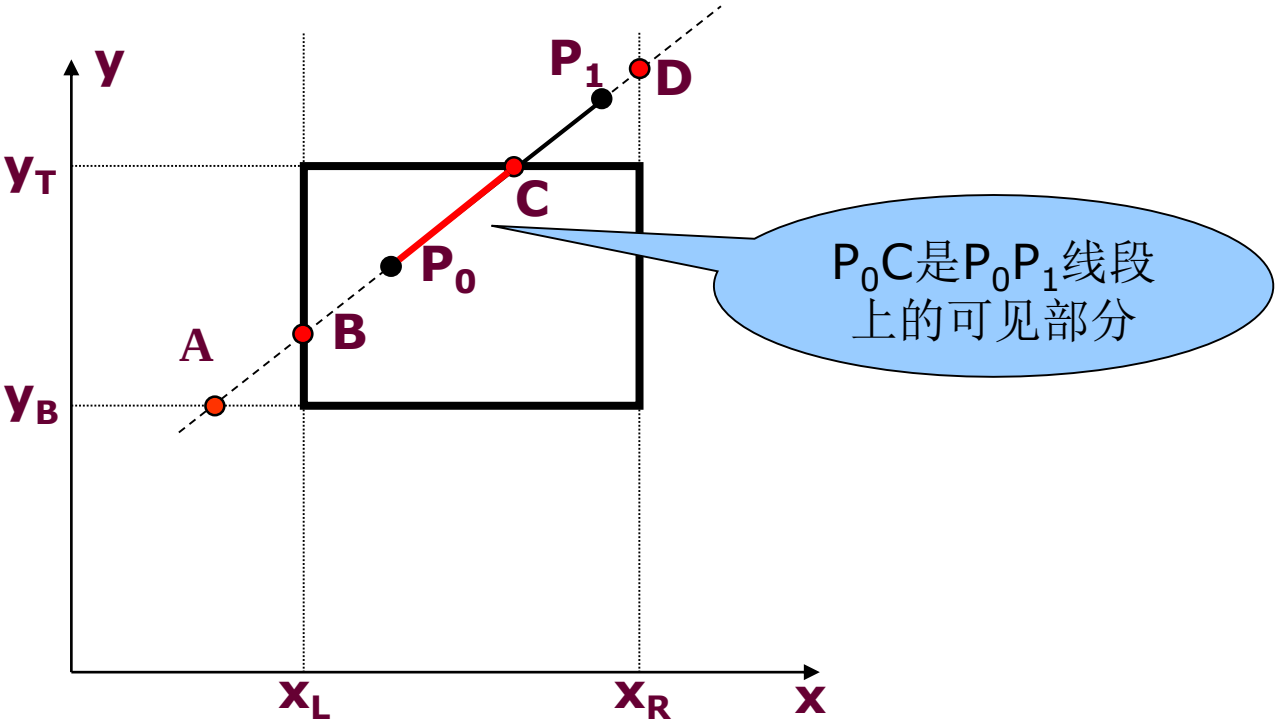
梁友栋

- 生于1935年7月，福建福州人
- 1956-1960年：复旦大学研究生，师从苏步青先生学习几何理论
- 1960年研究生毕业后任教于浙江大学数学系
- 1984-1990年任浙大数学系主任
- Liang, Y.D., and Barsky, B., "A New Concept and Method for Line Clipping", *ACM Transactions on Graphics*, 3(1):1-22, January 1984.

- Liang-Barsky算法

梁友栋-Barsky裁剪算法

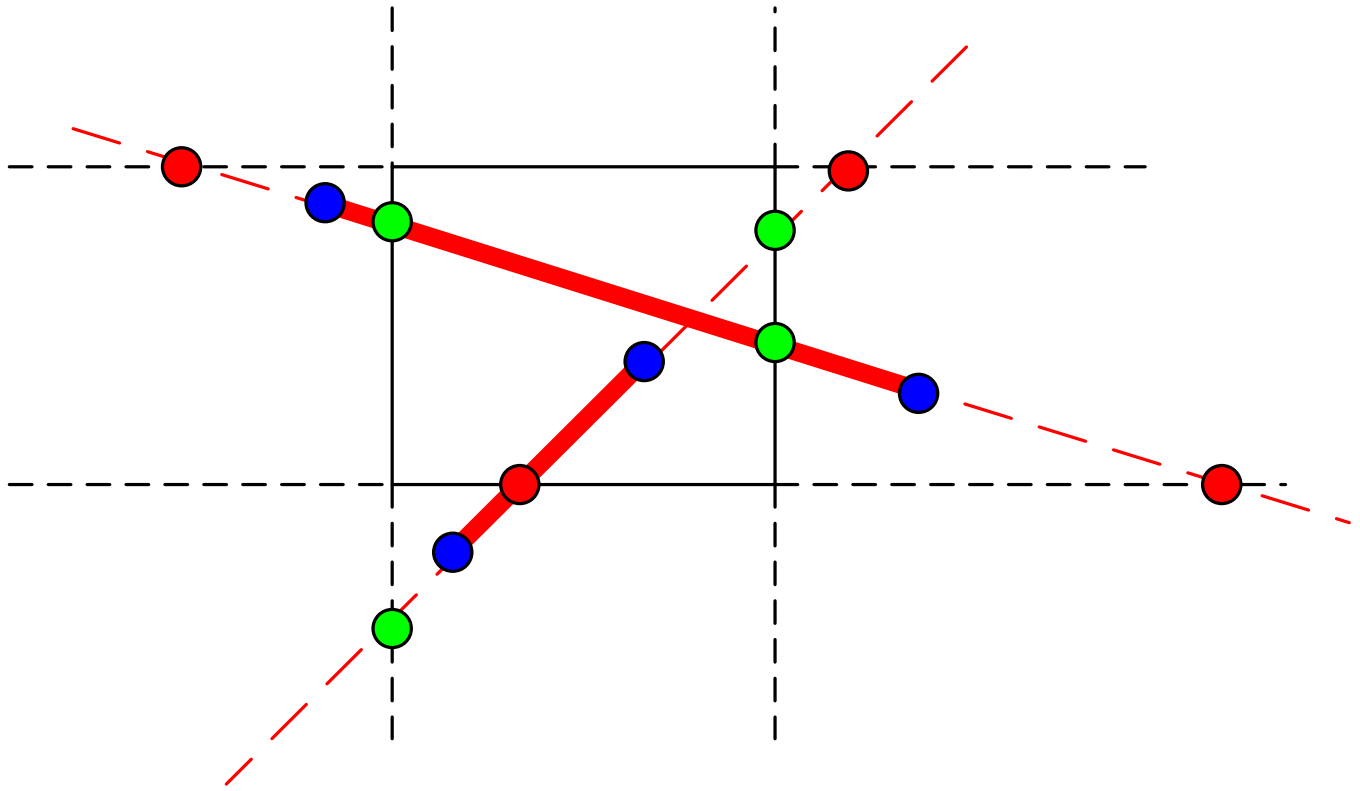
- 算法的基本思想：当要裁剪的线段是P_0P_1， P_0P_1和窗口边界交于A, B, C, D四点。从A, B和P_0三点中找出最靠近的P_1的点，从C, D和P_1中找出最靠近P_0的点。



把二维裁剪的问题化成二次一维裁剪问题,而把裁剪问题转化为解一组不等式的问题;

Liang-Barsky算法

分析



Liang-Barsky算法

直线的参数方程

$$x = x_1 + u \cdot (x_2 - x_1)$$
$$y = y_1 + u \cdot (y_2 - y_1) \qquad 0 \le u \le 1$$

对于直线上一点（x,y），若它在窗口内则有

$$wxl \le x_1 + u \cdot (x_2 - x_1) \le wxr$$
$$wxb \le y_1 + u \cdot (y_2 - y_1) \le wyt$$

$$u \cdot (x_1 - x_2) \le x_1 - wxl$$

$$u \cdot (x_2 - x_1) \le wxr - x_1$$

$$u \cdot (y_1 - y_2) \le y_1 - wyb$$

$$u \cdot (y_2 - y_1) \le wyt - y_1$$

令

$$p_1 = -(x_2 - x_1) \qquad q_1 = x_1 - wxl$$

$$p_2 = x_2 - x_1 \qquad q_2 = wxr - x_1$$

$$p_3 = -(y_2 - y_1) \qquad q_3 = y_1 - wyb$$

$$p_4 = y_2 - y_1 \qquad q_4 = wyt - y_1$$

则有 $u \cdot p_k \le q_k$

Liang-Barsky算法

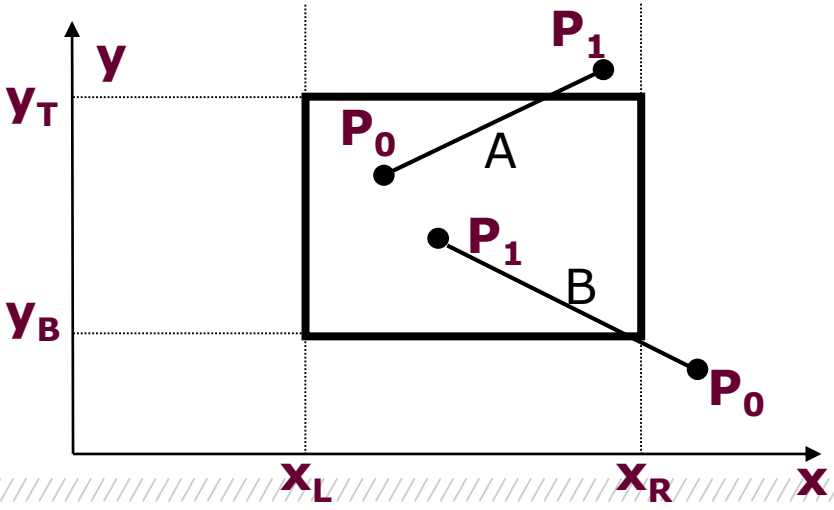
由 $u \cdot p_k \leq q_k$ 线段与裁剪边界的交点有

$$up_{\mathrm{k}} = q_{\mathrm{k}},$$

$$u_{\mathrm{k}} = \frac{q_{\mathrm{k}}}{p_{\mathrm{k}}}, \qquad p_{\mathrm{k}} \neq 0.$$

Liang-Barsky算法

- 窗口边界的四条边分成两类，一类称为始边（直线从该边界进入区域），另一类称为终边（直线从该边界离开区域）
  - 当Δx≥0时（$p_1 \leq 0$，$p_2 \geq 0$），称$x=x_L$为始边，$x=x_R$为终边
  - 当Δx<0时（$p_1 \geq 0$，$p_2 \leq 0$），称$x=x_L$为终边，$x=x_R$为始边
  - 当Δy≥0时（$p_3 \leq 0$，$p_4 \geq 0$），称$y=y_B$为始边，$y=y_T$为终边
  - 当Δy<0时（$p_3 \geq 0$，$p_4 \leq 0$），称$y=y_B$为终边，$y=y_T$为始边



Liang-Barsky算法

可知边界交点的参数：

$$up_{\mathrm{k}} = q_{\mathrm{k}},$$

$$u_{\mathrm{k}} = \frac{q_{\mathrm{k}}}{p_{\mathrm{k}}}, \qquad p_{\mathrm{k}} \neq 0.$$

- 当p_k<0时，
  - 求得的u_k必是P_0P_1和**始边**的交点的参数。**（下限组）**
- 当p_k>0时，
  - u_k必是P_0P_1和**终边**的交点的参数。 **（上限组）**

特殊处理：

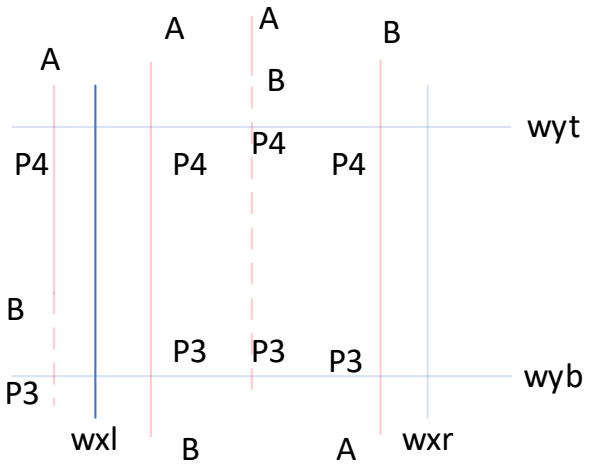
$p_3 = -(y_2 - y_1)$　　$q_3 = y_1 - wyb$

$p_4 = y_2 - y_1$　　$q_4 = wyt - y_1$

求出参数值：

$u3=q_3/p_3$，　$u4=q_4/p_4$

$u_A=0$，　$u_B=1$，



(a)直线段与窗口边界
wxl和wxr平行的情况

$$u_{\max} = \max(0, u_k \mid p_k < 0)$$　（下限组中，取最大）

$$u_{\min} = \min(u_k \mid p_k > 0, 1)$$　（上限组中，取最小）

（可见条件，下限u小于等于上限u）

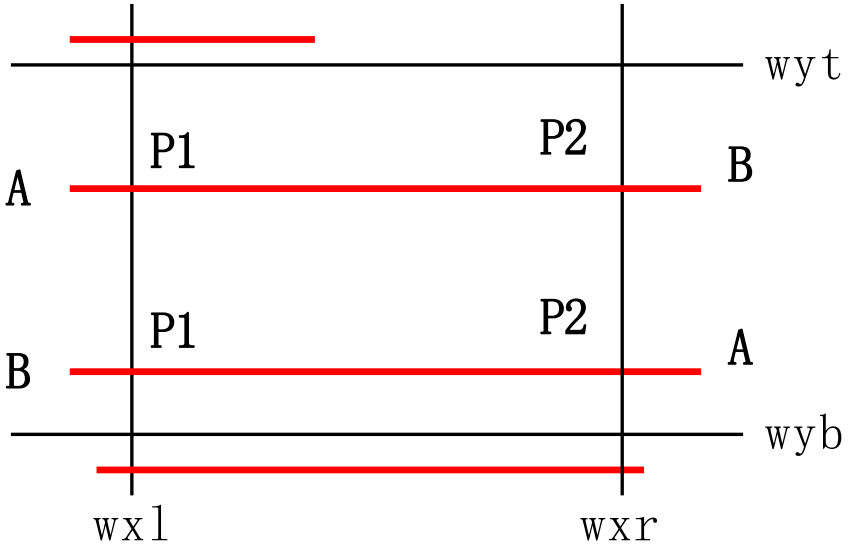
$$p_1 = -(x_2 - x_1) \qquad q_1 = x_1 - wxl$$

$$p_2 = x_2 - x_1 \qquad q_2 = wxr - x_1$$

求出参数值：

$u_1 = q_1/p_1$，　$u_2 = q_2/p_2$

$u_A = 0$，　$u_B = 1$，



(b) 直线段与窗口边界wyb和wyt平行的情况

$$u_{\max} = \max(0, u_k \mid p_k < 0)$$

（下限组中，取最大）

$$u_{\min} = \min(u_k \mid p_k > 0, 1)$$

（上限组中，取最小）

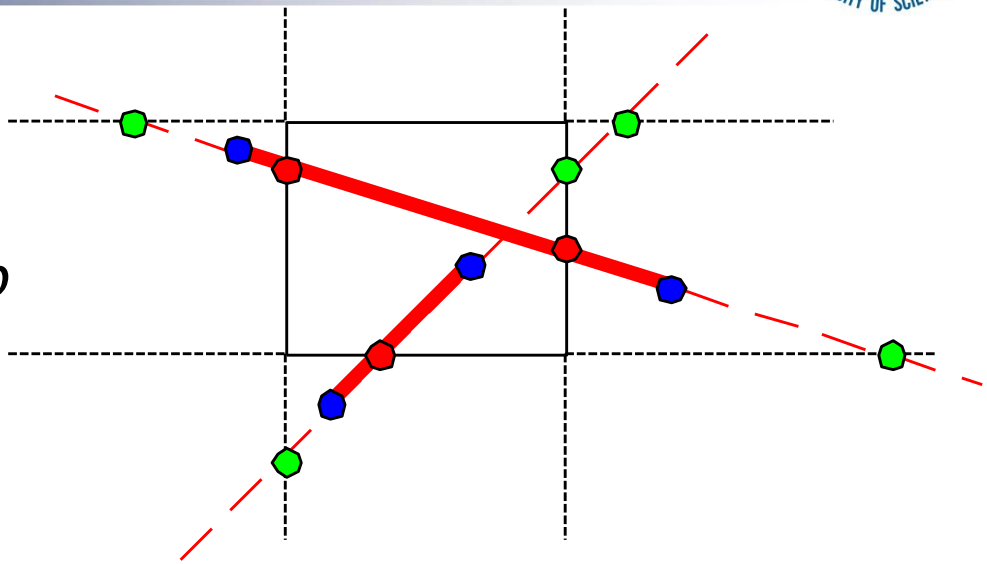
（可见条件，下限u小于等于上限u）

$$p_1 = -(x_2 - x_1) \qquad q_1 = x_1 - wxl$$

$$p_2 = x_2 - x_1 \qquad q_2 = wxr - x_1$$

$$p_3 = -(y_2 - y_1) \qquad q_3 = y_1 - wxb$$

$$p_4 = y_2 - y_1 \qquad q_4 = wyt - y_1$$



一般情况：

$$u_{\max} = \max(0, u_k | p_k < 0, u_k | p_k < 0)$$ （下限组中，取最大）

$$u_{min} = \min(u_k | p_k > 0, u_k | p_k > 0, 1)$$ （上限组中，取最小）

（可见条件，下限u小于等于上限u）

算法步骤：

(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。

(2)若$\Delta x=0$，则$p_1=p_2=0$。此时进一步判断是否满足$q_1<0$或$q_2<0$，若满足，则该直线段不在窗口内，算法转(7)。否则，满足$q_1>0$且$q_2>0$，则进一步计算u_1和u_2。算法转(5)。

(3)若$\Delta y=0$，则$p_3=p_4=0$。此时进一步判断是否满足$q_3<0$或$q_4<0$，若满足，则该直线段不在窗口内，算法转(7)。否则，满足$q_1>0$且$q_2>0$，则进一步计算u_1和u_2。算法转(5)。

(4)若上述两条均不满足，则有$p_k\neq0$（k=1,2,3,4）。此时计算u_1和u_2。

(5)求得u_1和u_2后，进行判断：若$u_1>u_2$，则直线段在窗口外，算法转(7)。若$u_1<u_2$，利用直线的参数方程求得直线段在窗口内的两端点坐标。

(6)利用直线的扫描转换算法绘制在窗口内的直线段。算法结束。

二维裁剪

- 点的裁剪
- 直线段的裁剪
- 多边形的裁剪

作业

- 6. 4
- 6. 11
- 6. 14