

第六章 二维变换及二维观察

- 如何对二维图形进行方向、尺寸和形状方面的变换。
- 如何进行二维观察。

二维变换及二维观察

- 基本几何变换与基本概念
- 二维图形几何变换的计算
- 复合变换
- 变换的性质

6.2 基本几何变换

- **图形的几何变换**是指对图形的几何信息经过平移、比例、旋转等变换后产生新的图形，是图形在方向、尺寸和形状方面的变换。(位置、尺寸-形状、方向)
- **基本几何变换**都是相对于坐标原点和坐标轴进行的几何变换。

基本几何变换——平移变换

□ 平移是指将 p 点沿直线路径从一个坐标位置移到另一个坐标位置的重定位过程。

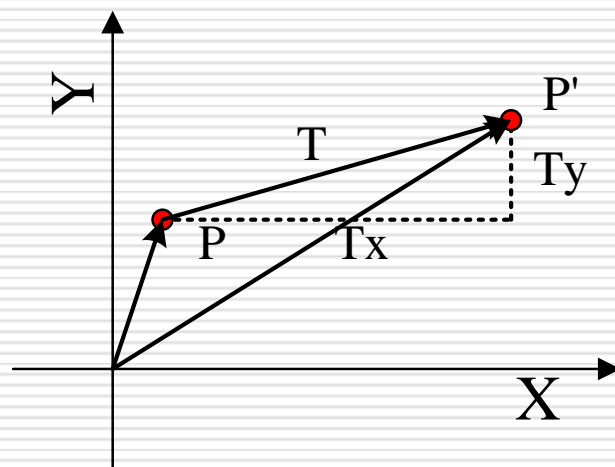


图6-1 平移变换

基本几何变换——平移变换

推导:

$$x' = x + T_x$$

$$y' = y + T_y$$

矩阵形式:

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} + \begin{bmatrix} T_x & T_y \end{bmatrix}$$

T_x , T_y 称为平移矢量。

基本几何变换——比例变换

推导:

$$x' = S_x \cdot x$$

$$y' = S_y \cdot y$$

矩阵形式:

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$

基本几何变换——比例变换

□ 比例变换是指对 p 点相对于坐标原点沿 x 方向放缩 S_x 倍，沿 y 方向放缩 S_y 倍。其中 S_x 和 S_y 称为比例系数。

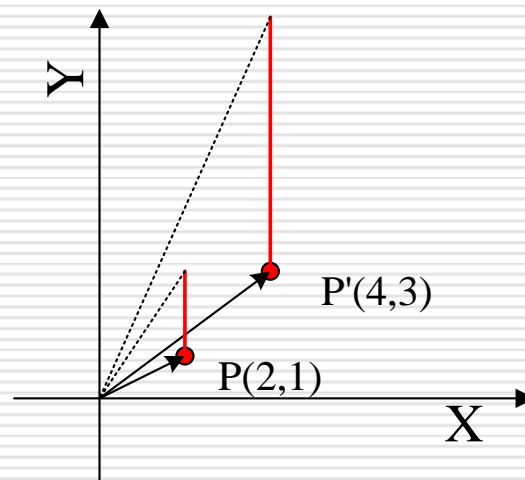
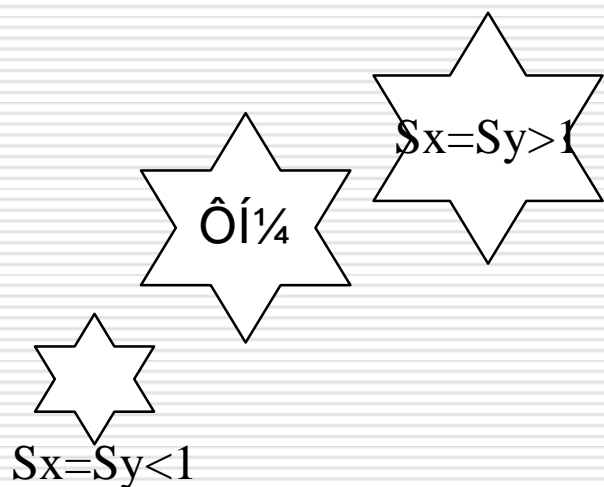
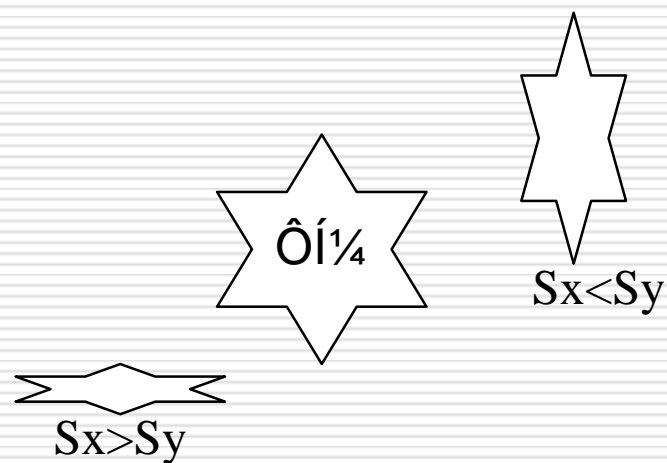


图6-2 比例变换($S_x=2, S_y=3$)

基本几何变换——比例变换



(a) $S_x = S_y$ 比例



(b) $S_x \neq S_y$ 比例

图6-3 比例变换

基本几何变换——旋转变换

- 二维旋转是指将 p 点绕坐标原点转动某个角度（逆时针为正，顺时针为负）得到新的点 p' 的重定位过程。

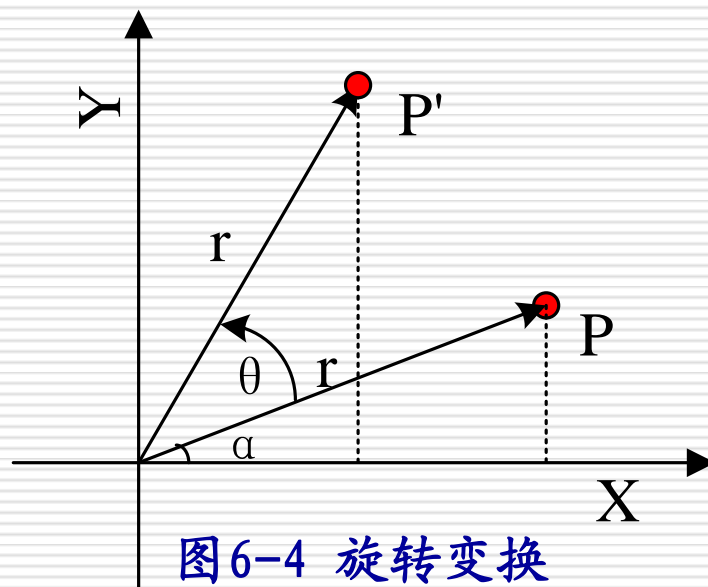


图6-4 旋转变换

基本几何变换——旋转变换

□ 推导：（极坐标）

$$x = r \cos \alpha \quad y = r \sin \alpha$$

$$x' = r \cos(\alpha + \theta) = x \cos \theta - y \sin \theta$$

$$y' = r \sin(\alpha + \theta) = x \sin \theta + y \cos \theta$$

□ 矩阵：逆时针旋转 θ 角

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$$

基本几何变换

□ 平移、缩放、旋转变换的矩阵表示:

$$P' = P + T$$

$$P' = P \cdot S \quad \longrightarrow \quad P' = P \cdot T_1 + T_2$$

$$P' = P \cdot R$$

□ 图形通常要进行一系列基本几何变换，希望能够把二维变换统一表示为矩阵的乘法。

基本几何变换——规范化齐次坐标

□ 齐次坐标表示就是用 $n+1$ 维向量表示一个 n 维向量。

$$(x, y) \Leftarrow (xh, yh, h) \quad h \neq 0$$

□ 规范化齐次坐标表示就是 $h=1$ 的齐次坐标表示。

$$(x, y) \Leftarrow (x, y, 1)$$

基本几何变换

平移: $[x' \quad y'] = [x \quad y] + [T_x \quad T_y]$



$$[x' \quad y' \quad 1] = [x \quad y \quad 1] \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix}$$

基本几何变换

比例:

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} S_x & 0 \\ 0 & S_y \end{bmatrix}$$



$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

基本几何变换

整体比例变换:

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & S \end{bmatrix}$$

基本几何变换

旋转变换:

$$\begin{bmatrix} x' & y' \end{bmatrix} = \begin{bmatrix} x & y \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & \sin \theta \\ -\sin \theta & \cos \theta \end{bmatrix}$$

\Downarrow

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

基本几何变换——二维变换矩阵

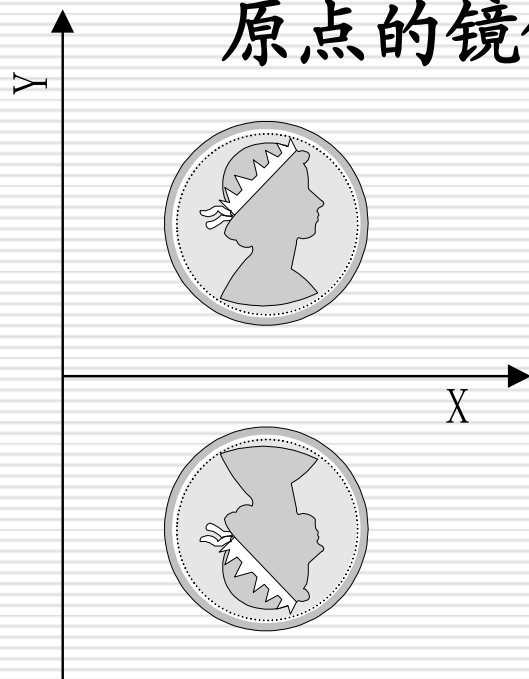
$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot T_{2D} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} a & b & p \\ c & d & q \\ l & m & s \end{bmatrix}$$

$$x' = \frac{ax + cy + l}{px + qy + s}$$

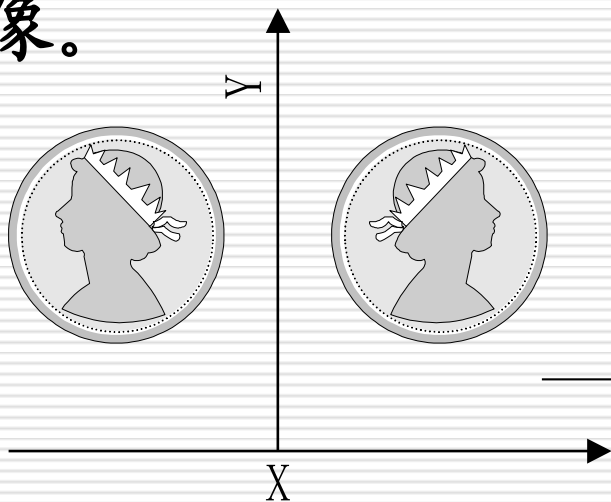
$$y' = \frac{bx + dy + m}{px + qy + s}$$

基本几何变换——对称变换

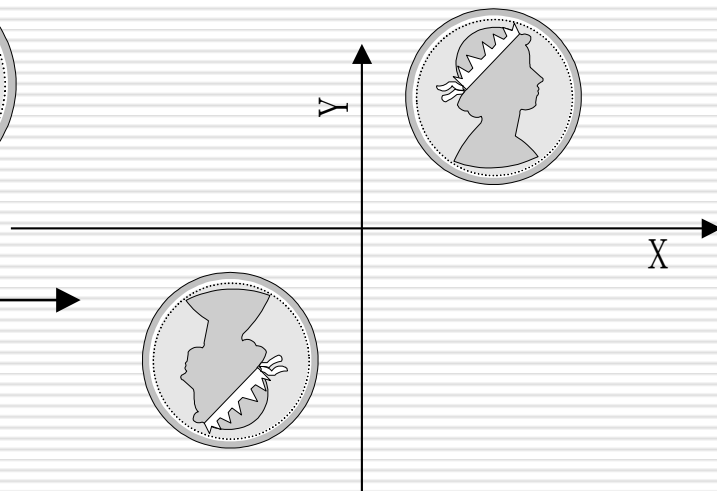
□ 对称变换后的图形是原图形关于某一轴线或原点的镜像。



(a) 关于x轴对称

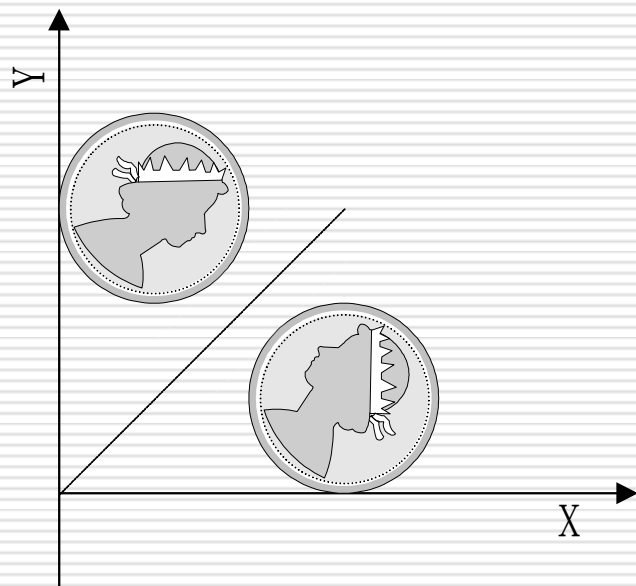


(b) 关于y轴对称

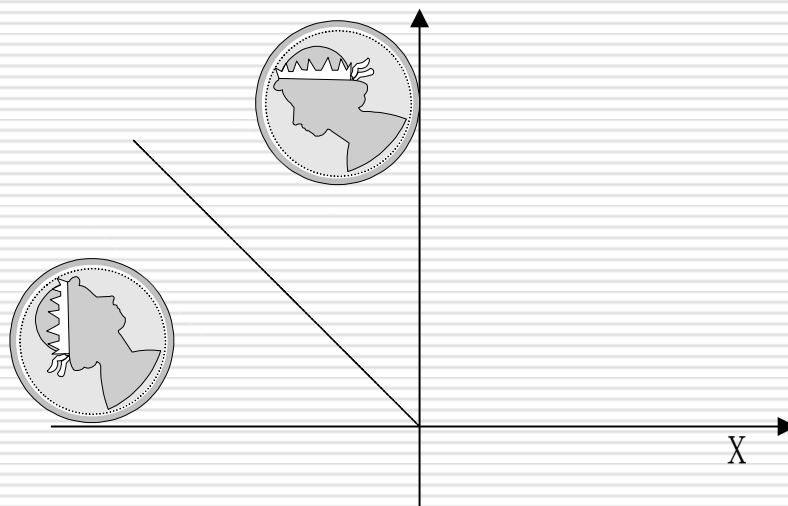


(c) 关于原点对称

基本几何变换——对称变换



(d) 关于 $x=y$ 对称



(e) 关于 $x=-y$ 对称

基本几何变换——对称变换

(1)关于x轴对称

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

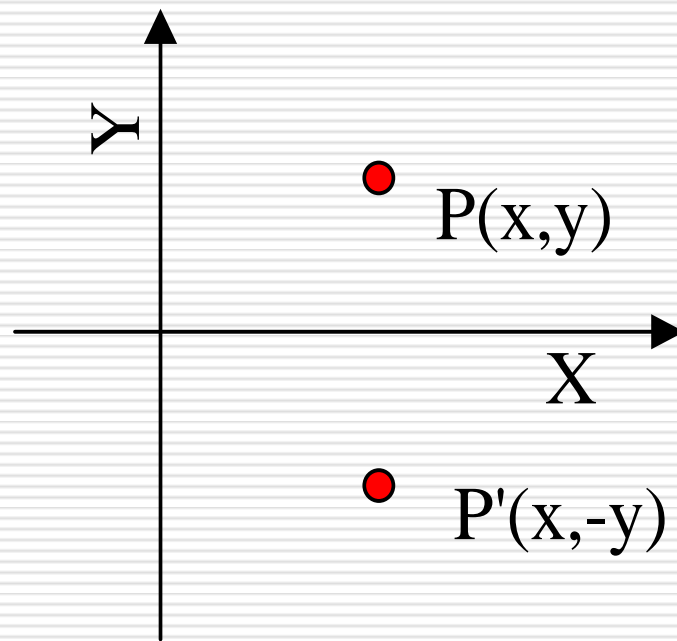


图6-5 关于x轴对称

基本几何变换——对称变换

(2)关于y轴对称

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

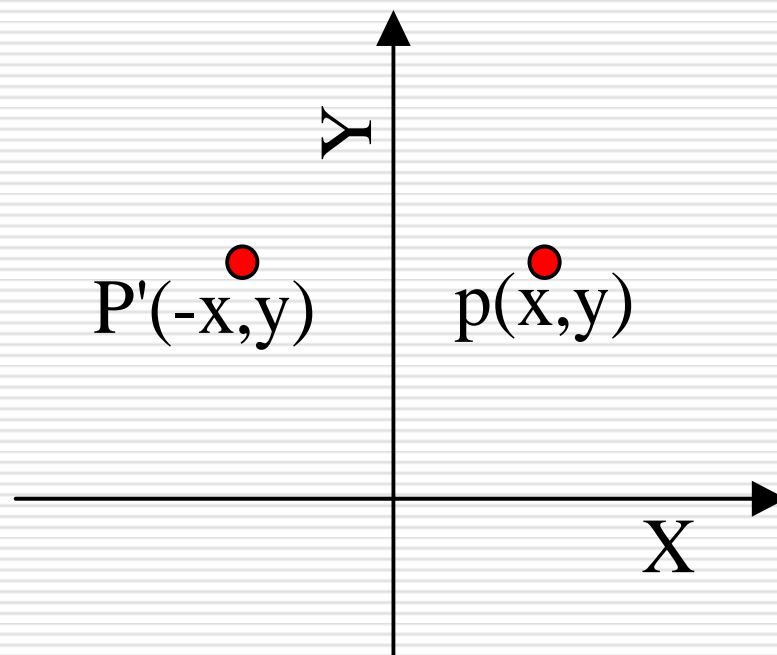


图6-6 关于y轴对称

基本几何变换——对称变换

(3)关于原点对称

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

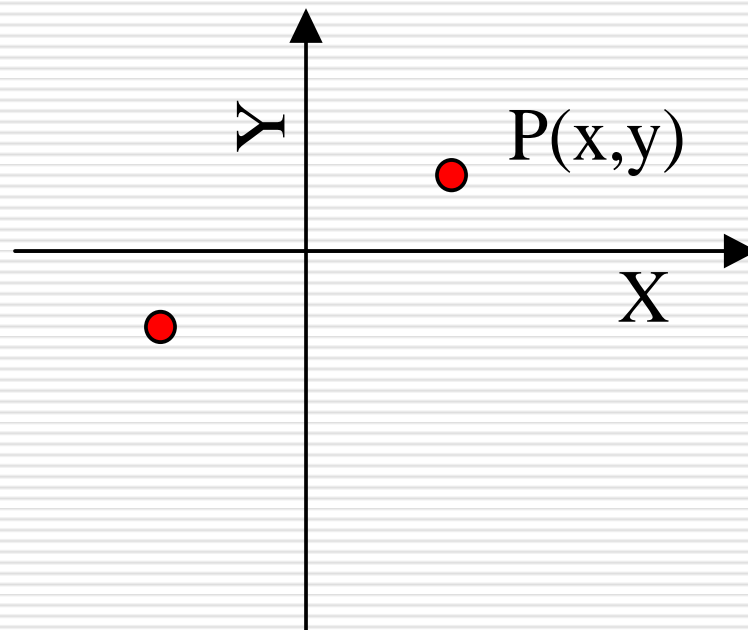


图6-7 关于原点对称

基本几何变换——对称变换

(4)关于 $y=x$ 轴对称

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

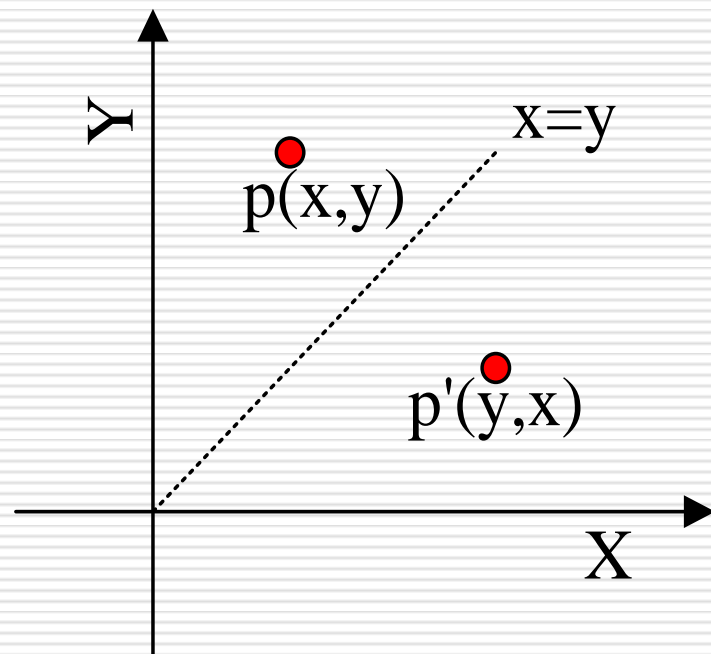


图6-8 关于 $x=y$ 对称

基本几何变换——对称变换

(5)关于 $y=-x$ 轴对称

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

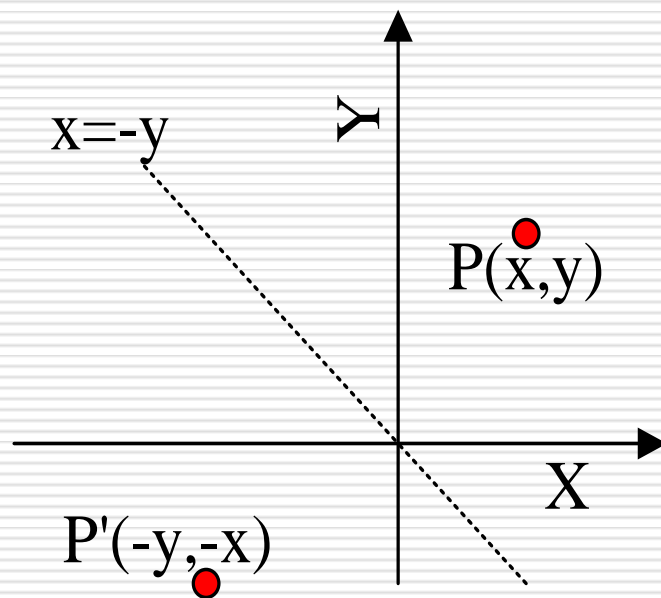


图6-9 关于 $x = -y$ 对称

基本几何变换——错切变换

□ 错切变换，也称为剪切、错位变换，用于产生弹性物体的变形处理。

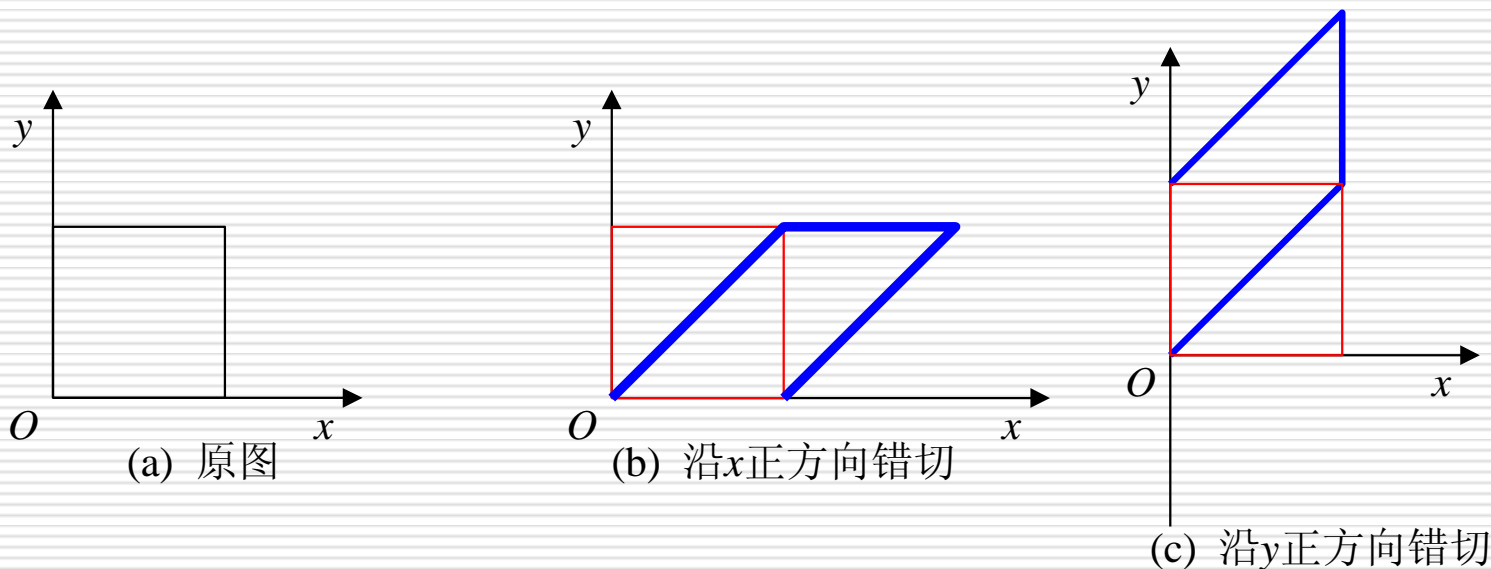


图6-10 错切变换

基本几何变换——错切变换

其变换矩阵为：

$$\begin{bmatrix} 1 & b & 0 \\ c & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(1)沿x方向错切

(2)沿y方向错切

(3)两个方向错切

二维图形几何变换的计算

几何变换均可表示成 $P'=P*T$ 的形式。

1. 点的变换

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \cdot \begin{bmatrix} a & b & p \\ c & d & q \\ l & m & r \end{bmatrix}$$

二维图形几何变换的计算

2. 直线的变换

$$\begin{bmatrix} x_1' & y_1' & 1 \\ x_2' & y_2' & 1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{bmatrix} \cdot \begin{bmatrix} a & b & p \\ c & d & q \\ l & m & r \end{bmatrix}$$

二维图形几何变换的计算

3. 多边形的变换

$$\begin{bmatrix} x'_1 & y'_1 & 1 \\ x'_2 & y'_2 & 1 \\ x'_3 & y'_3 & 1 \\ \dots & \dots & \dots \\ x'_n & y'_n & 1 \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ \dots & \dots & \dots \\ x_n & y_n & 1 \end{bmatrix} \cdot \begin{bmatrix} a & b & p \\ c & d & q \\ l & m & r \end{bmatrix}$$

复合变换

- 图形作一次以上的几何变换，变换结果是每次变换矩阵的乘积。
- 任何一复杂的几何变换都可以看作基本几何变换的组合形式。
- 复合变换具有形式：

$$\begin{aligned} P' &= P \cdot T = P \cdot (T_1 \cdot T_2 \cdot T_3 \cdots T_n) \\ &= P \cdot T_1 \cdot T_2 \cdot T_3 \cdots T_n \quad (n > 1) \end{aligned}$$

复合变换——二维复合平移

$$\begin{aligned} T_t &= T_{t1} \cdot T_{t2} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x1} & T_{y1} & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x2} & T_{y2} & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_{x1} + T_{x2} & T_{y1} + T_{y2} & 1 \end{bmatrix} \end{aligned}$$

复合变换——二维复合比例

$$\begin{aligned} T_s = T_{s1} \cdot T_{s2} &= \begin{bmatrix} S_{x1} & 0 & 0 \\ 0 & S_{y1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_{x2} & 0 & 0 \\ 0 & S_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} S_{x1} \cdot S_{x2} & 0 & 0 \\ 0 & S_{y1} \cdot S_{y2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

复合变换——二维复合旋转

$$\begin{aligned} T_r = T_{r1} \cdot T_{r2} &= \begin{bmatrix} \cos \theta_1 & \sin \theta_1 & 0 \\ -\sin \theta_1 & \cos \theta_1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta_2 & \sin \theta_2 & 0 \\ -\sin \theta_2 & \cos \theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos(\theta_1 + \theta_2) & \sin(\theta_1 + \theta_2) & 0 \\ -\sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

$$R = R_{(\theta_1)} \bullet R_{(\theta_2)} = R(\theta_1 + \theta_2)$$

复合变换

$$\begin{aligned} R &= \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & 0 \\ 0 & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & \operatorname{tg} \theta & 0 \\ -\operatorname{tg} \theta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & \operatorname{tg} \theta & 0 \\ -\operatorname{tg} \theta & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & 0 & 0 \\ 0 & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

相对任一参考点的二维几何变换

□ 相对某个参考点 (x_F, y_F) 作二维几何变换，其变换过程为：

(1) 平移；

(2) 针对原点进行二维几何变换；

(3) 反平移。

相对任一参考点的二维几何变换

例1. 相对点 (x_F, y_F) 的旋转变换

$$\begin{aligned} T_{RF} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_F & -y_F & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ x_F & y_F & 1 \end{bmatrix} \\ &= \begin{bmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ x_F - x_F \cos \theta + y_F \sin \theta & y_F - y_F \cos \theta - x_F \sin \theta & 1 \end{bmatrix} \end{aligned}$$

相对任意方向的二维几何变换

□ 相对任意方向作二维几何变换，其变换的过程是：

(1) 旋转变换；

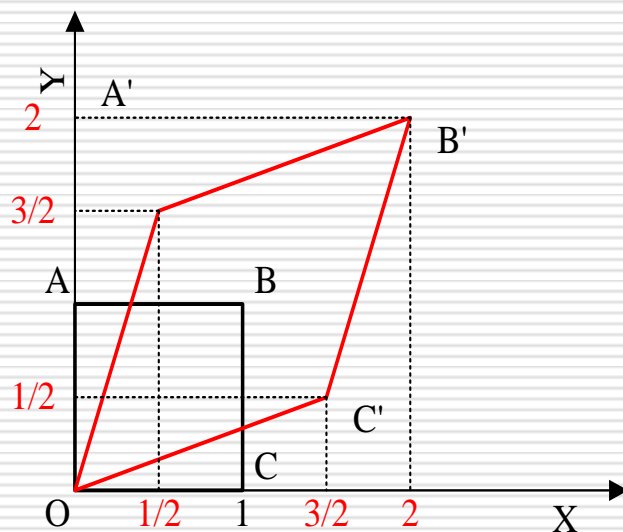
(2) 针对坐标轴进行二维几何变换；

(3) 反向旋转。

□ 例2. 相对直线 $y=x$ 的反射变换

复合变换

例3. 将正方形 $ABCO$ 各点沿下图所示的 $(0,0) \rightarrow (1,1)$ 方向进行拉伸，结果为如图所示的，写出其变换矩阵和变换过程。



可能发生的变换：沿 $(0, 0)$
到 $(1, 1)$ 的比例变换

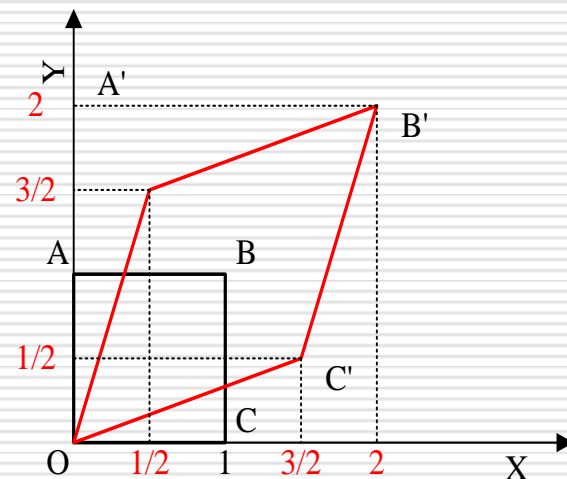


图6-11 沿固定方向拉伸

$$T = \begin{bmatrix} \cos(-45^\circ) & \sin(-45^\circ) & 0 \\ -\sin(-45^\circ) & \cos(-45^\circ) & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos 45^\circ & \sin 45^\circ & 0 \\ -\sin 45^\circ & \cos 45^\circ & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$P' = P \cdot T$$

坐标系之间的变换

问题:

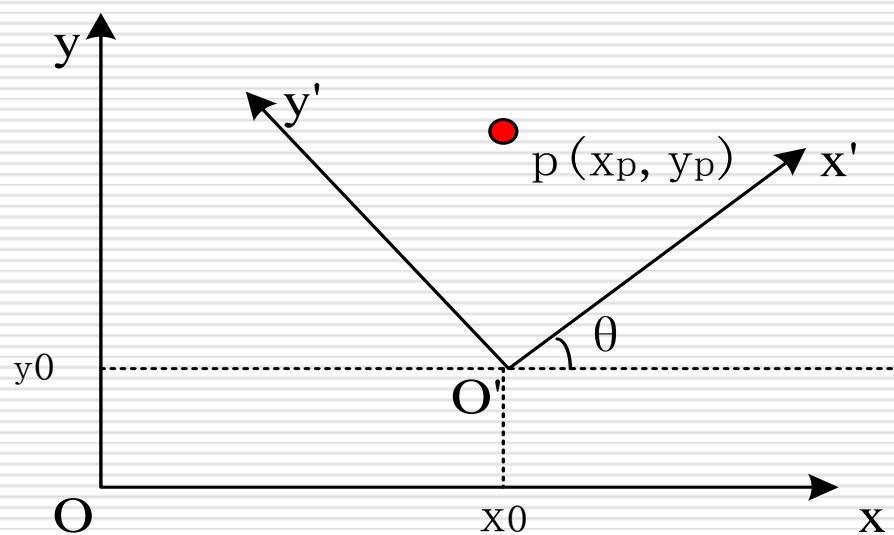


图6-12 坐标系间的变换

坐标系之间的变换

分析:

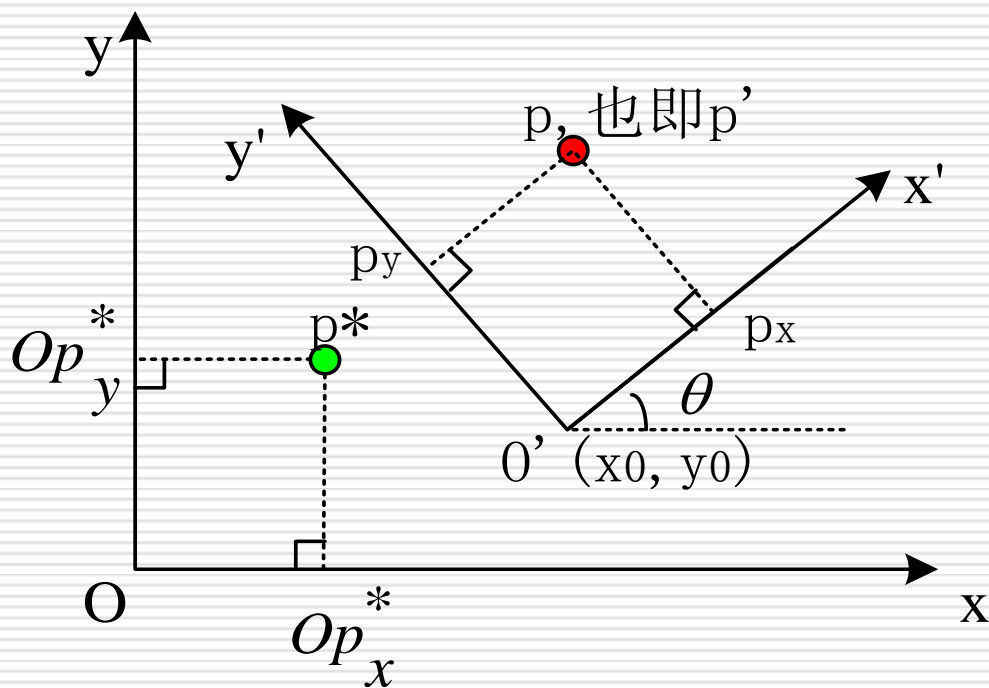


图6-13 坐标系间的变换的原理

可以分两步进行:

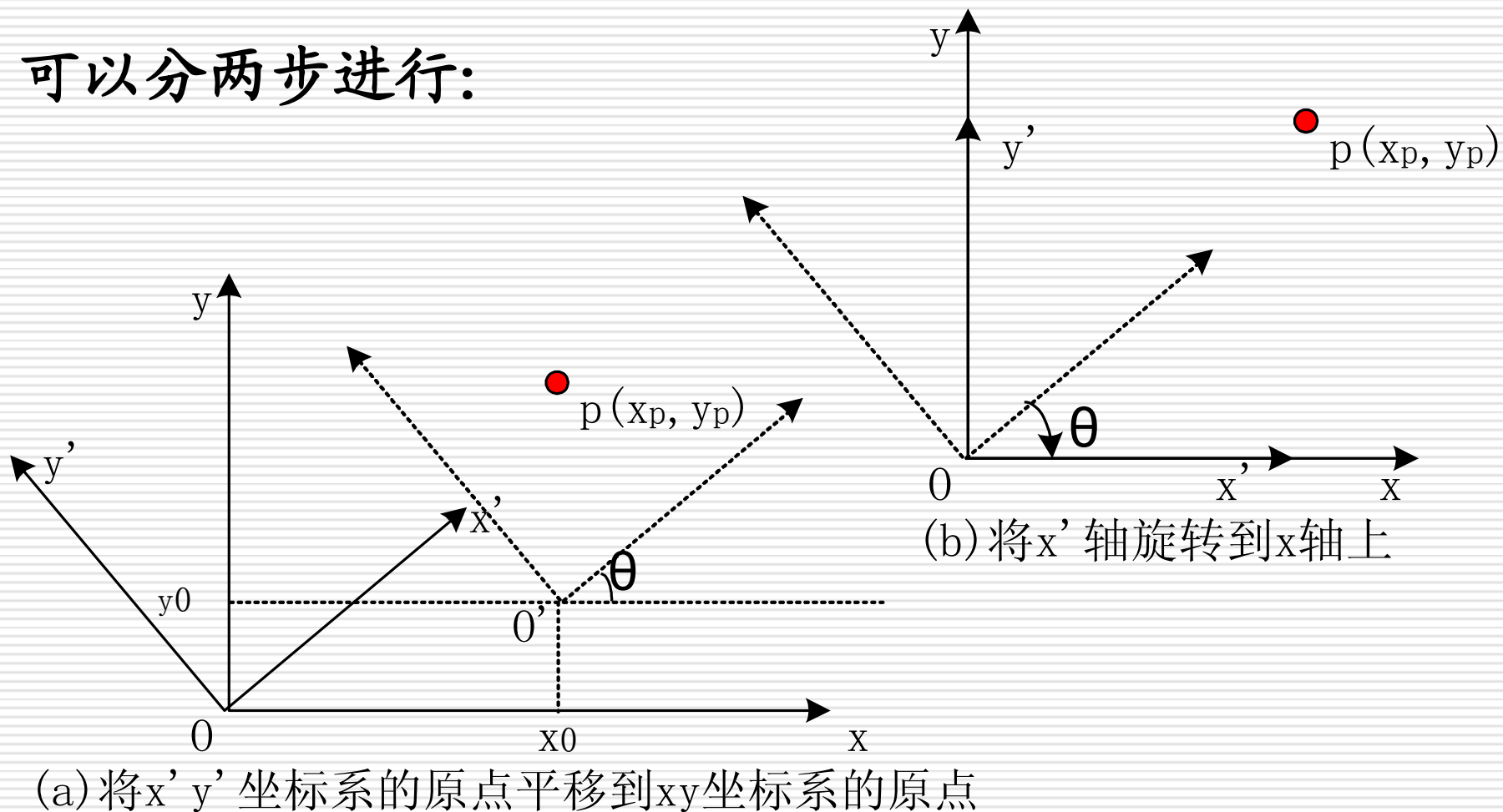


图6-14 坐标系间的变换的步骤

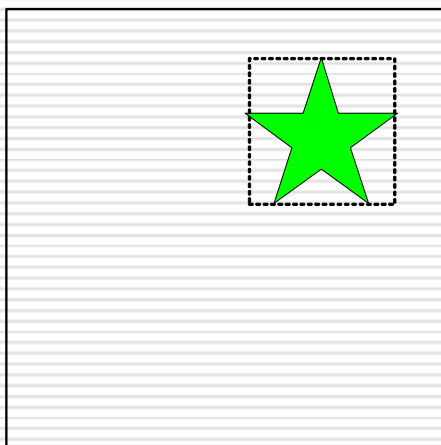
于是:

$$\begin{aligned} p' &= \begin{bmatrix} x'_p & y'_p & 1 \end{bmatrix} = \begin{bmatrix} x_p & y_p & 1 \end{bmatrix} \cdot T \\ &= p \cdot T = p \cdot T_t \cdot T_R \end{aligned}$$

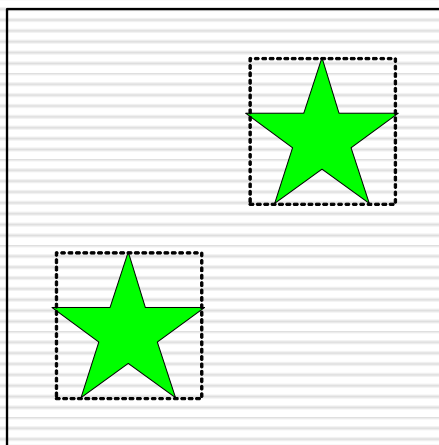
$$T = T_t \cdot T_r = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -x_0 & -y_0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

光栅变换

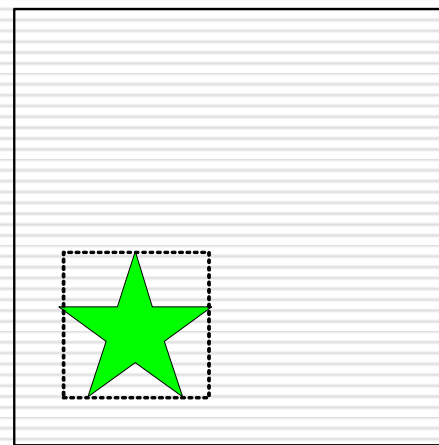
- ❑ 直接对帧缓存中象素点进行操作的操作称为光栅变换。
- ❑ 光栅平移变换:



(a) 读出象素块的内容



(b) 复制象素块的内容



(c) 擦除原象素块的内容

光栅变换

□ 90° 、 180° 和 270° 的光栅旋转变换:

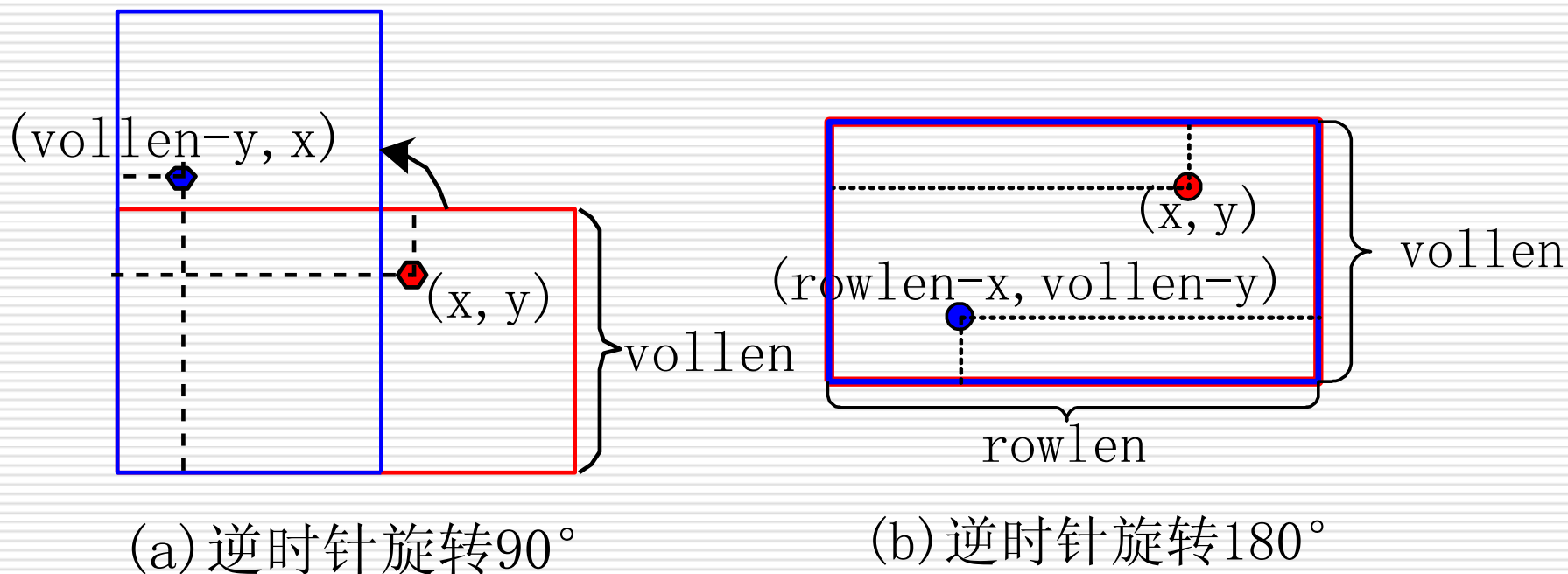


图6-15 光栅旋转变换

光栅变换

□ 任意角度的光栅旋转变换:

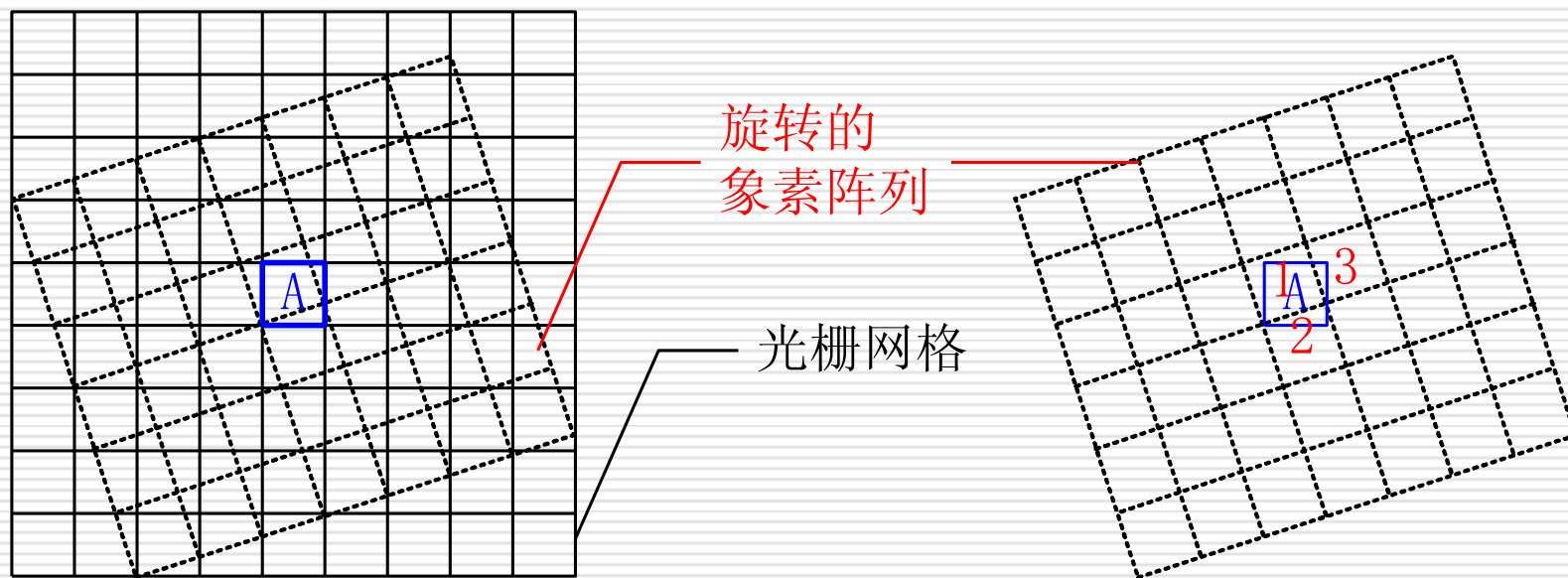


图6-16 任意角度的光栅旋转变换

光栅变换

□ 光栅比例变换：进行区域的映射处理。

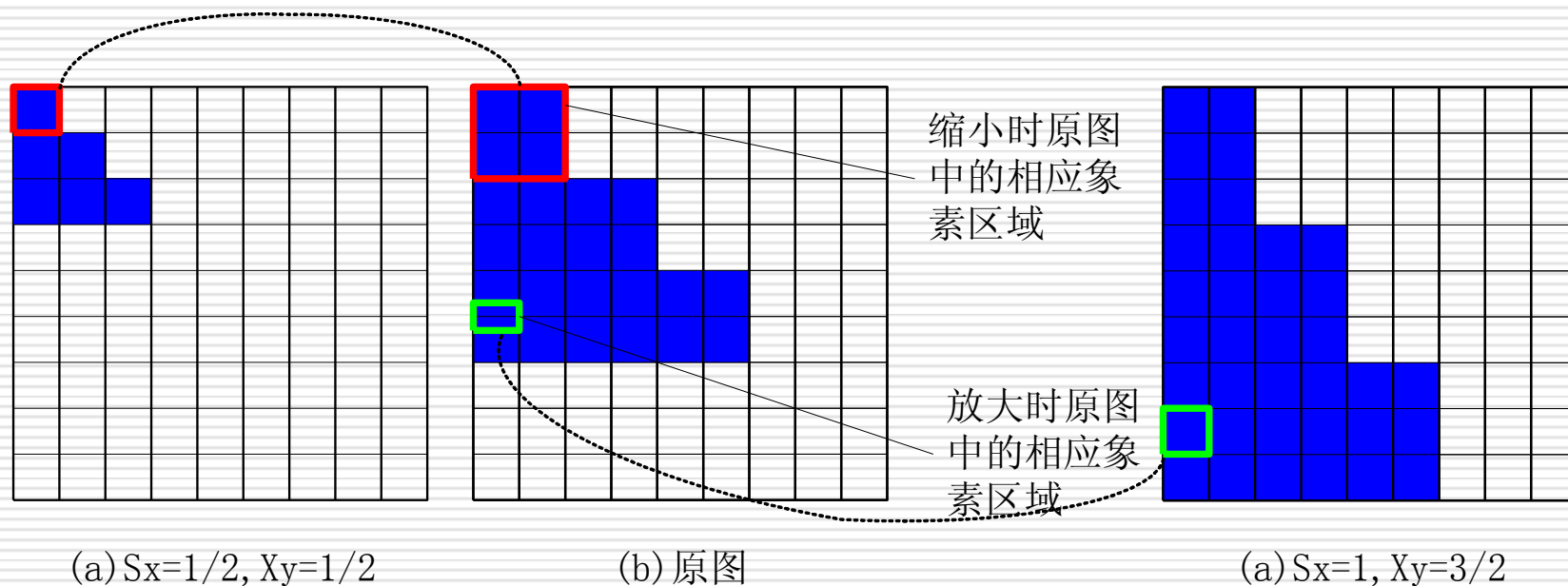


图6-16 光栅比例变换

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，反过来，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。

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。

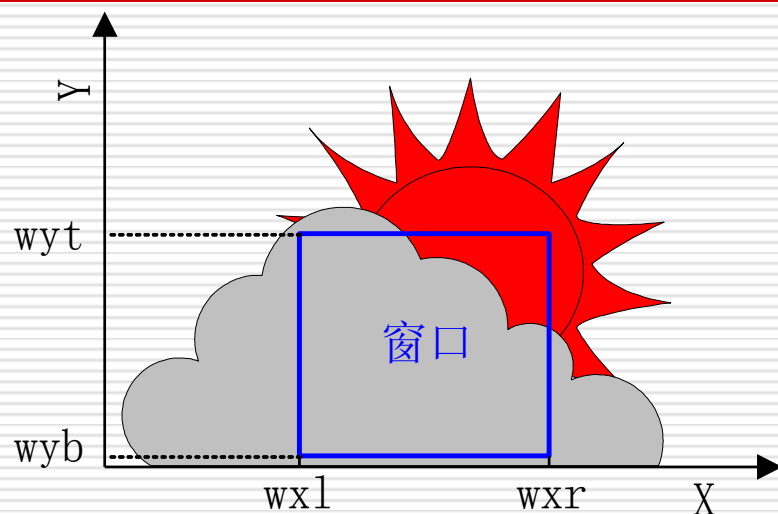
二维观察

- 基本概念
- 二维观察变换
- 二维裁剪
- **OpenGL**中的二维观察

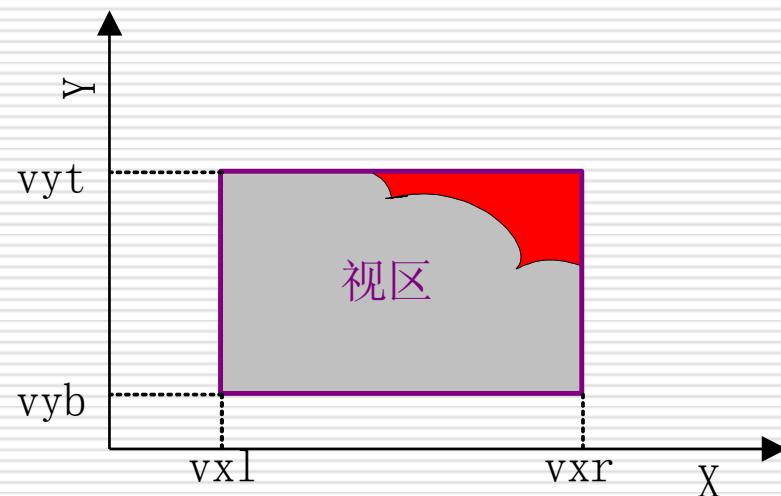
二维观察——基本概念

- 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**）。

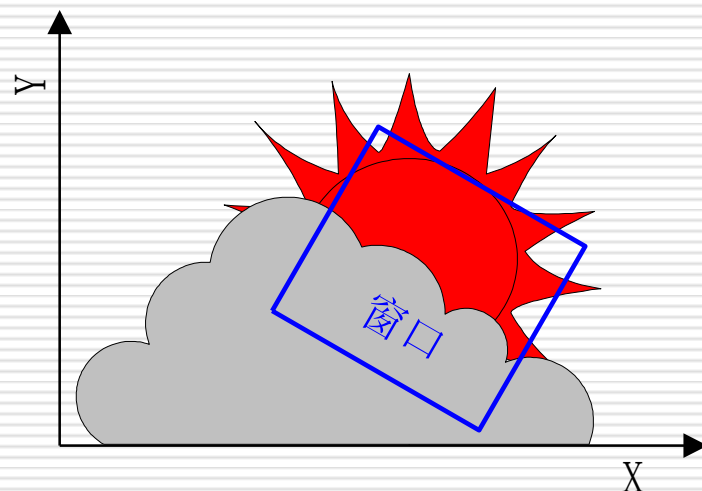
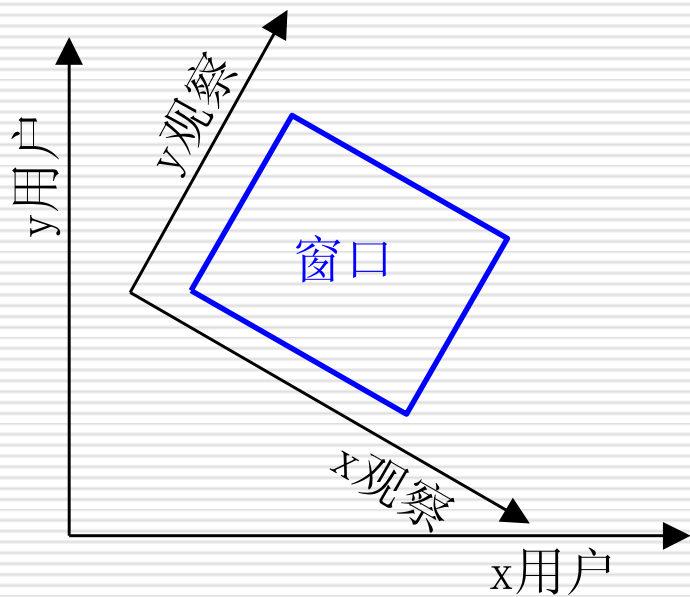
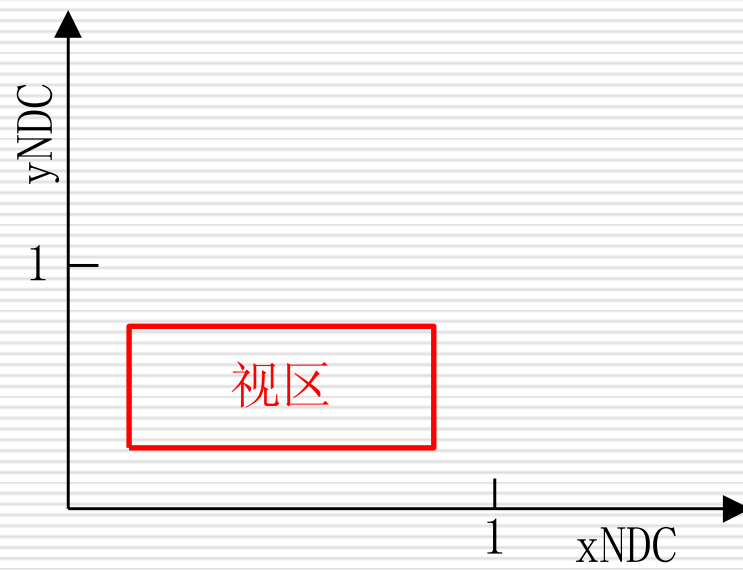


图6-17 用户坐标系中旋转的窗口



(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6-17 二维观察流程

二维观察——基本概念

□ 变焦距效果

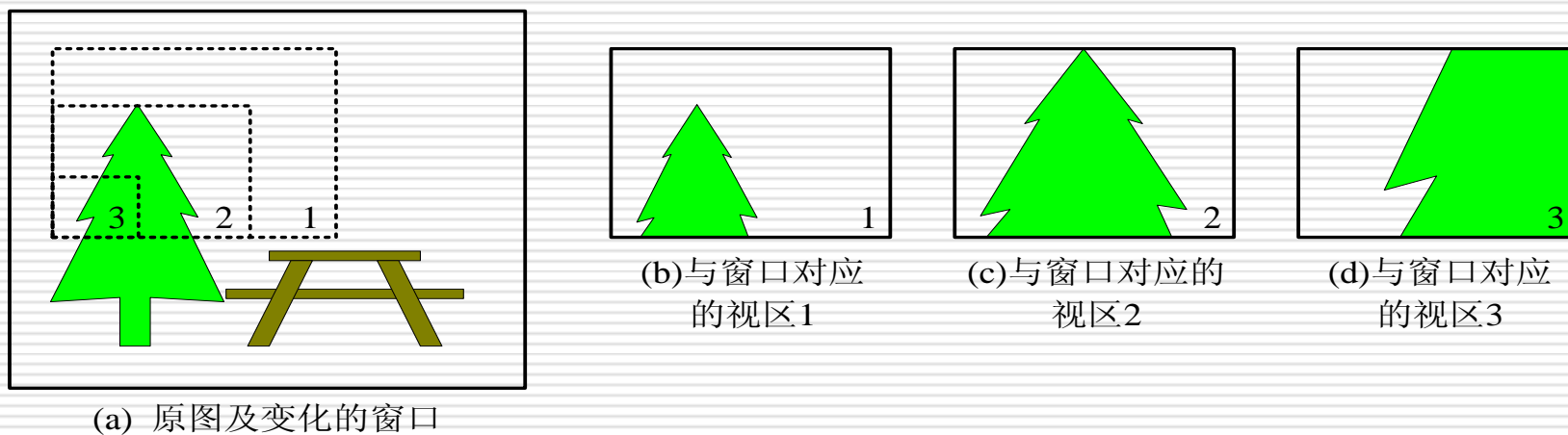


图6-18 变焦距效果（窗口变、视区不变）

二维观察——基本概念

□ 整体放缩效果

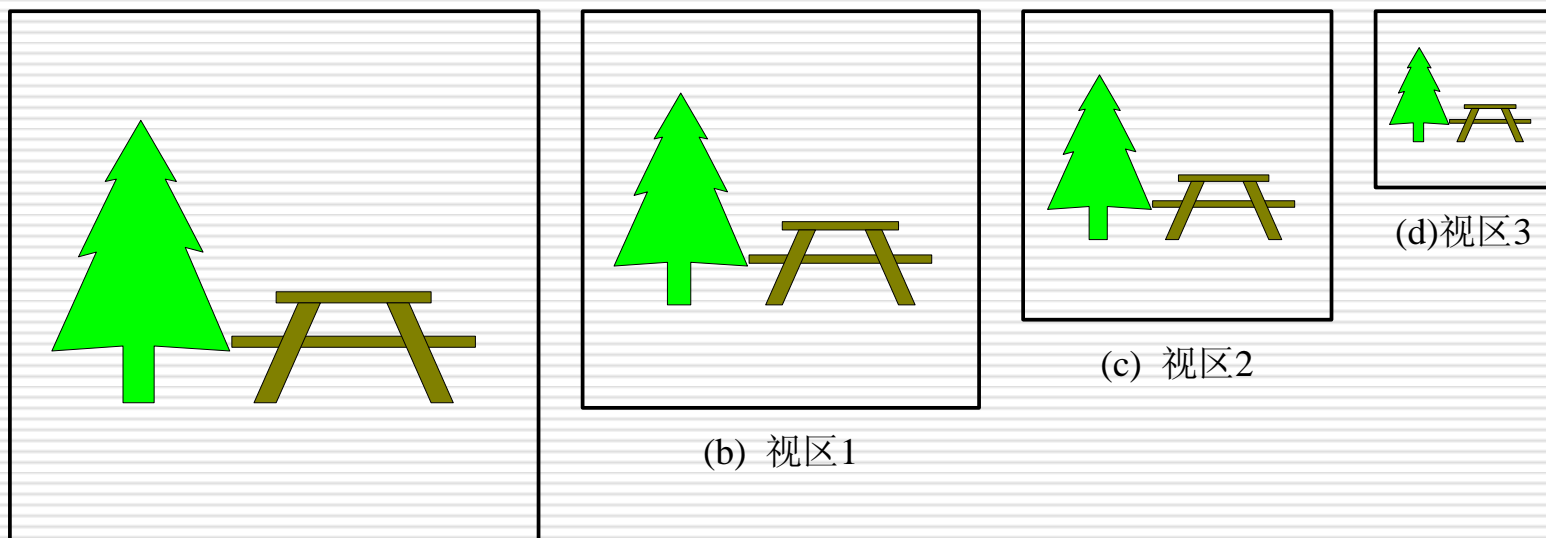
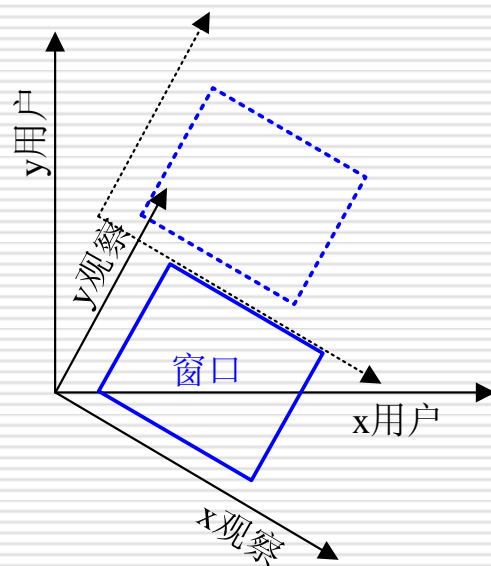


图6-17 整体放缩效果（窗口不变、视区变）

□ 漫游效果

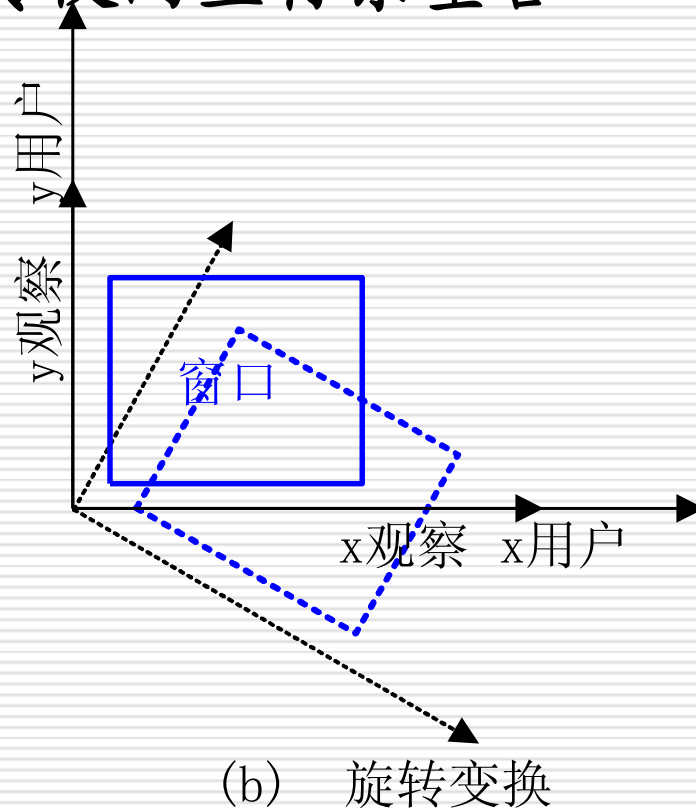
用户坐标系到观察坐标系的变换

- 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；

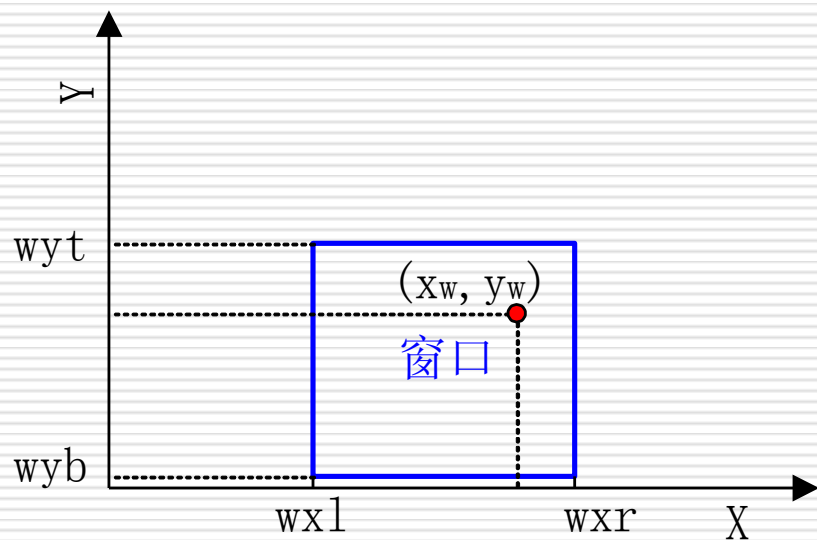


用户坐标系到观察坐标系的变换

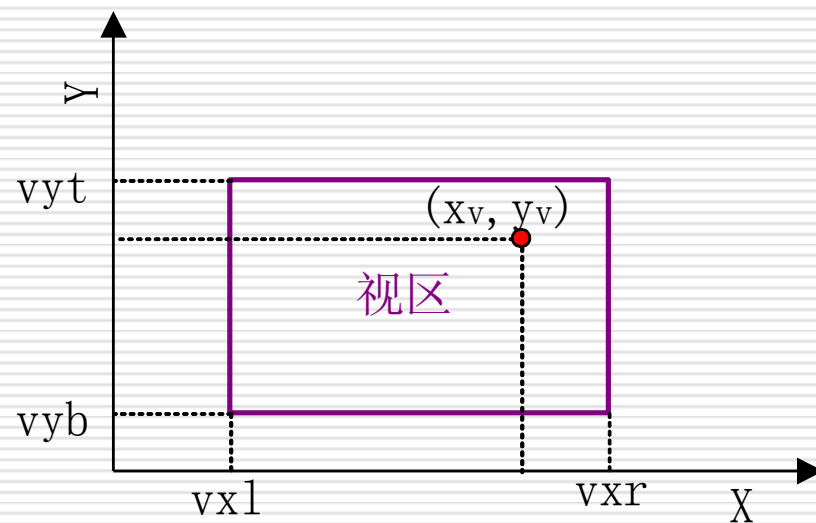
◆ 绕原点旋转使两坐标系重合



窗口到视区的变换



(a) 窗口中的点



(b) 视区中的点

图6-23 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。

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。

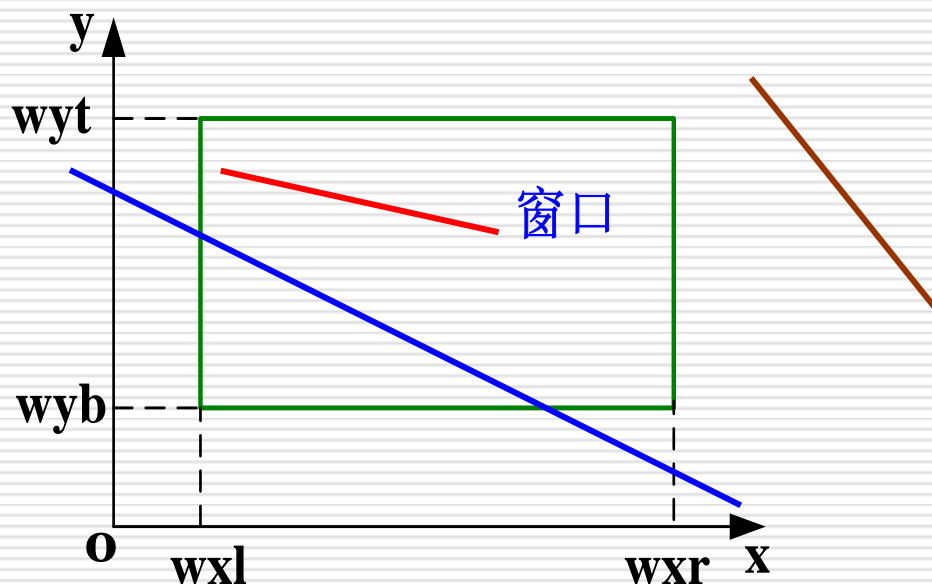
裁剪——点的裁剪

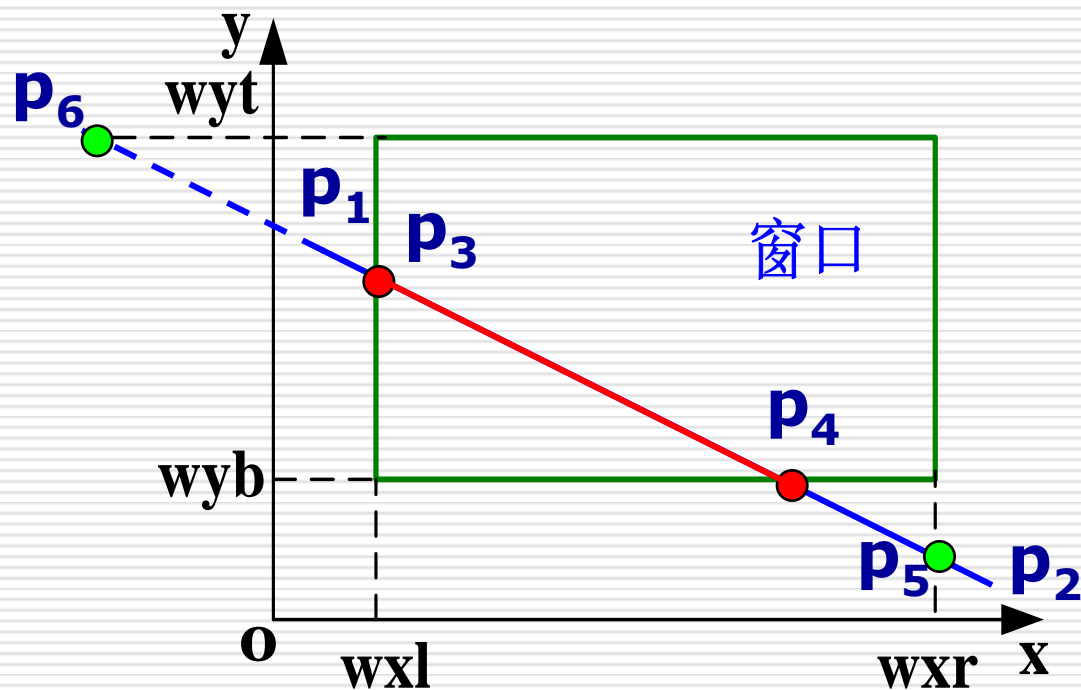
$$wxl \leq x \leq wxr,$$
$$\text{且 } wyb \leq y \leq wyt$$

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 。





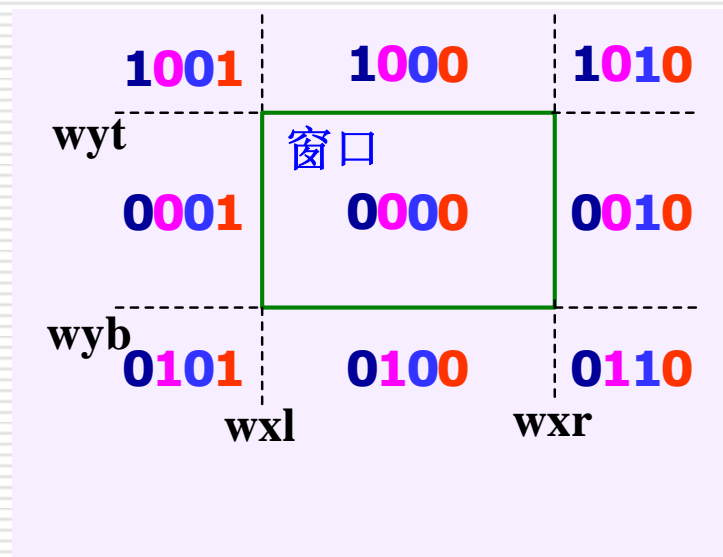
- 实交点：直线段与窗口矩形边界的交点；
- 虚交点：处于直线段延长线或窗口边界延长线上的交点。

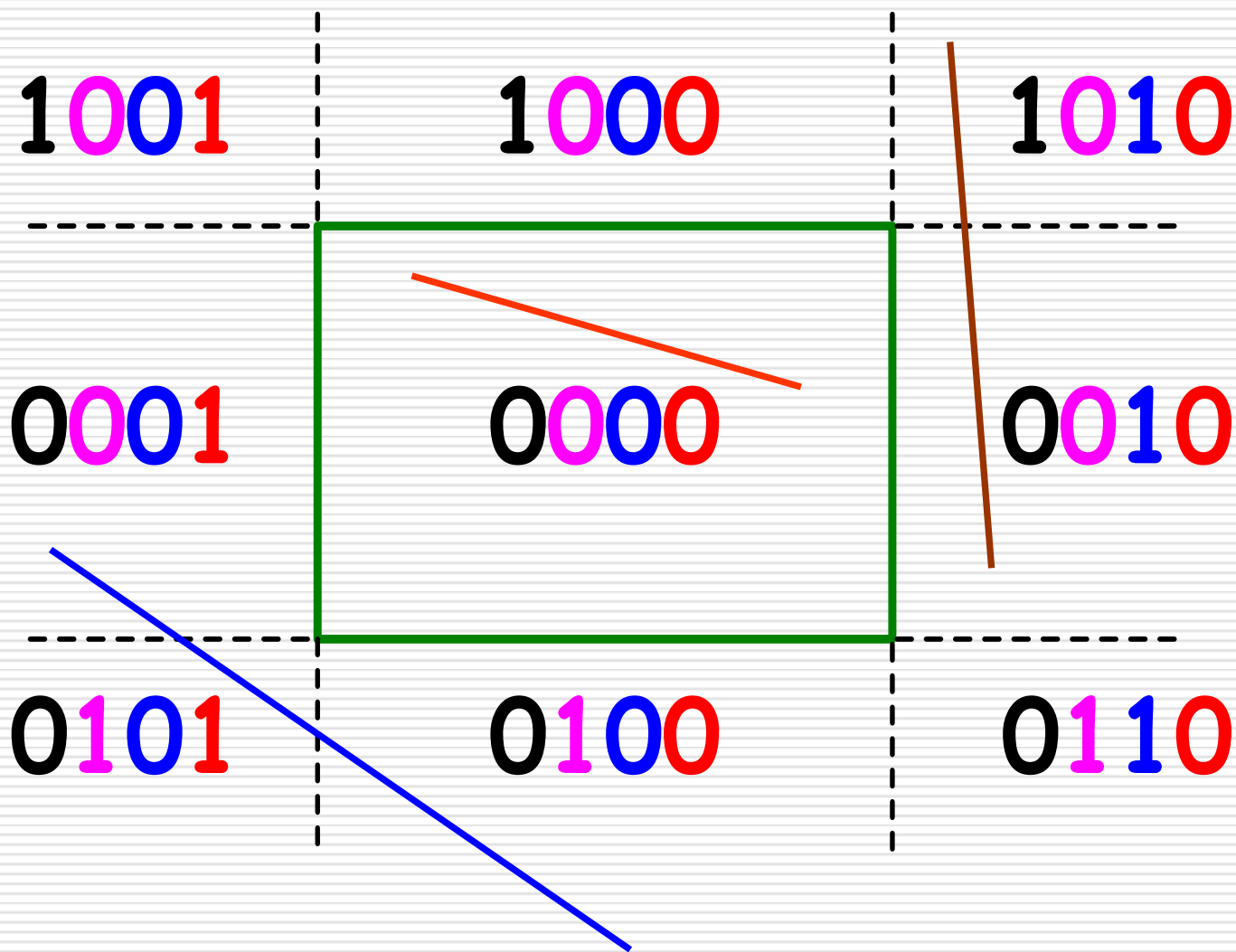
Cohen-Sutherland算法

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 $D_3=0$ 。





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 $code_1$ 和 $code_2$ ，然后：

a. 若 $code_1 | code_2 = 0$ ，对直线段简取之；

b. 若 $code_1 \& code_2 \neq 0$ ，对直线段简弃之；

Cohen-Sutherland算法

(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。

a. 左、右边界交点的计算: $y = y_1 + k(x - x_1)$;

b. 上、下边界交点的计算: $x = x_1 + (y - y_1)/k$ 。

其中, $k = (y_2 - y_1) / (x_2 - x_1)$ 。

Cohen-Sutherland算法

```
#define LEFT    1
#define RIGHT   2
#define BOTTOM   4
#define TOP     8
```

```
void dda_line(float x1, float y1, float x2, float y2);
```

```
typedef struct ClipWindow
{
    float XL, XR, YB, YT;
} *TPtrClipWindow;
```

Cohen-Sutherland算法

```
void encode(float x, float y, int *code, TPtrClipWindow win)
{
    int c;
    c=0;
    if (x<win->XL) c=c|LEFT;
    else if (x>win->XR) c=c|RIGHT;

    if (y<win->YB) c=c|BOTTOM;
    else if (y>win->YT) c=c|TOP;

    *code=c;
    return;
}
```

```

void C_S_LineClip(float x1, float y1, float x2, float
    y2, TPtrClipWindow win)
{
    int code , code1, code2;
    float x, y;
    encode(x1, y1, &code1, win);
    encode(x2, y2, &code2, win);
    while (code1!=0 || code2!=0) {
        if ((code1 & code2) != 0) return;
        code=code1;
        if (code==0) code=code2; //让code为窗口外的端点的编码
        if ( (code & LEFT) != 0) //线段与左边界相交
        {
            x=win->XL;
            y=y1+(y2-y1)*(win->XL-x1)/(x2-x1);
        }
        else if ( (code & RIGHT) !=0) //线段与右边界相交
        {
            x=win->XR;
            y=y1+(y2-y1)*(win->XR-x1)/(x2-x1);
        }
    }
}

```



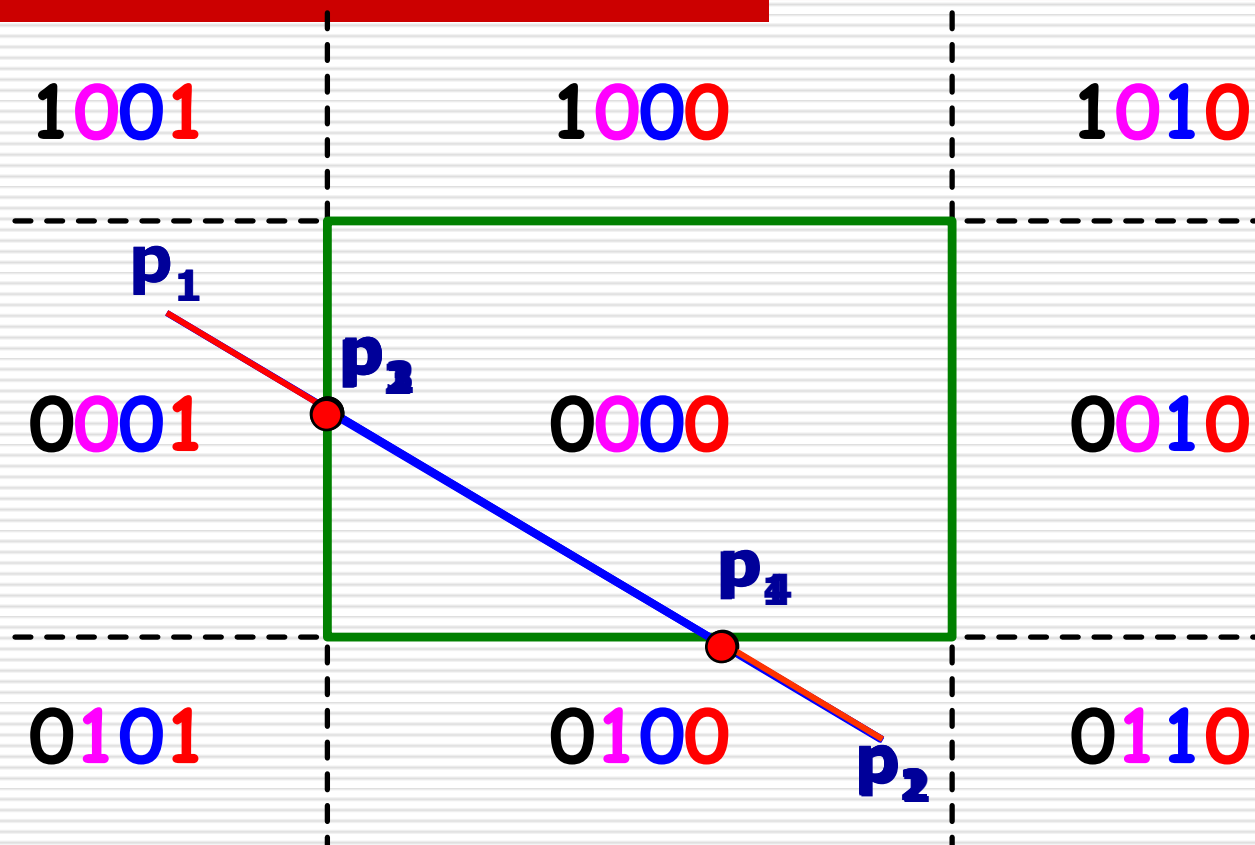
```

else if ((code & BOTTOM) != 0) //线段与下边界相交
{
    y=win->YB;
    x=x1+(x2-x1)*(win->YB-y1)/(y2-y1);
}
else if ((code & TOP) != 0) //线段与上边界相交
{
    y=win->YT;
    x=x1+(x2-x1)*(win->YT-y1)/(y2-y1);
}
if (code == code1) {
    x1=x; y1=y;    encode(x, y, &code1, win);
}
else
{
    x2=x; y2=y;    encode(x, y, &code2, win);
}
}

//显示直线的可见部分
dda_line(x1, y1, x2, y2);
return;
}

```

Cohen-Sutherland 算法

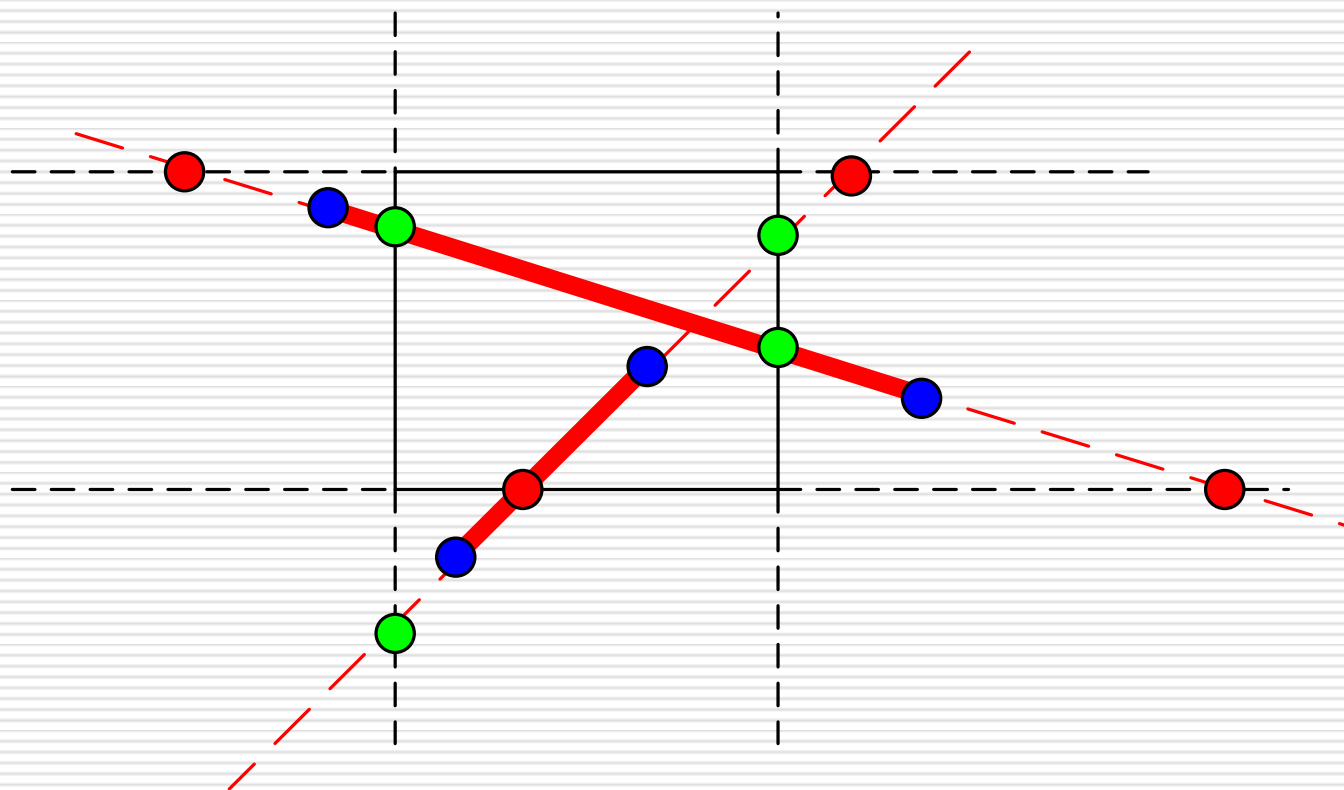


Cohen-Sutherland算法

- 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。

Liang-Barsky算法

分析



Liang-Barsky算法

直线的参数方程

$$\begin{aligned}x &= x_1 + u \cdot (x_2 - x_1) \\ y &= y_1 + u \cdot (y_2 - y_1)\end{aligned} \quad 0 \leq u \leq 1$$

对于直线上一点 (x, y) , 若它在窗口内则有

$$wxl \leq x_1 + u \cdot (x_2 - x_1) \leq wxr$$

$$wxb \leq y_1 + u \cdot (y_2 - y_1) \leq wyt$$

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

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

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

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

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

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

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

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

则有 $u \cdot p_k \leq q_k$

- 任何平行于剪切边界之一的直线 $p_k=0$,其中 k 对应于该剪切边界 ($k=1,2,3,4$ 对应于左、右、下、上边界)。如果还满足 $q_k<0$,则线段完全在边界之外,因此舍弃该线段。如果 $q_k\geq 0$,则线段位于边界之内。
- 当 $p_k<0$,线段从剪切边界延长线的外部延长到内部。当 $p_k>0$,线段从剪切边界延长线的内部延长到外部。当 $p_k\neq 0$,可以计算出线段与边界 k 的延长线的交点的 u 值:

$$u = \frac{q_k}{p_k}$$

Liang-Barsky 算法

$$\text{由 } u \cdot p_k \leq q_k$$

$$\left\{ \begin{array}{l} \frac{q_k}{p_k} (p_k < 0) \leq u \leq \frac{q_k}{p_k} (p_k > 0) \quad k = 1, 2 \\ \frac{q_k}{p_k} (p_k < 0) \leq u \leq \frac{q_k}{p_k} (p_k > 0) \quad k = 3, 4 \\ 0 \leq u \leq 1 \end{array} \right.$$

特殊处理:

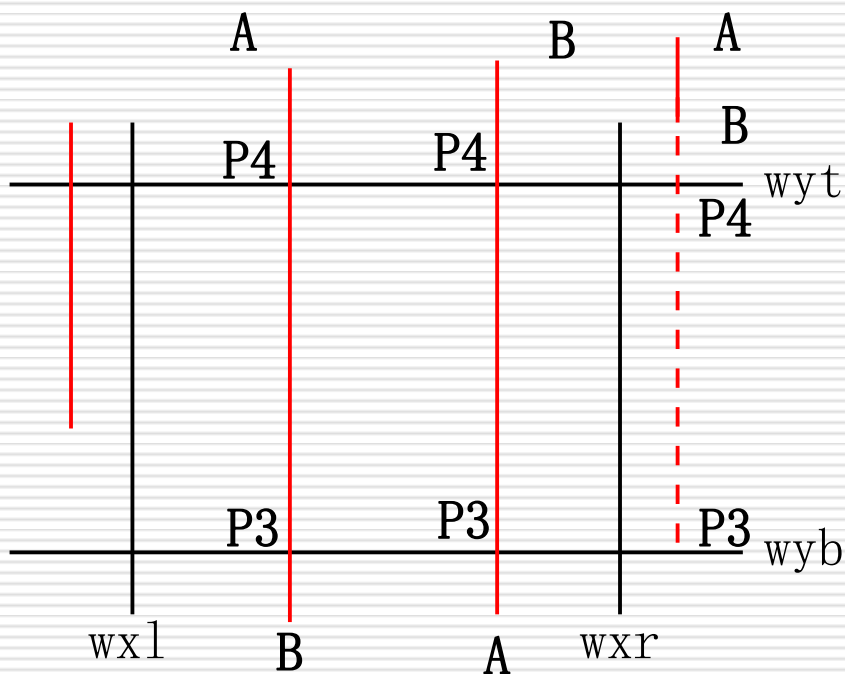
$$p_3 = -(y_2 - y_1) \quad q_3 = y_1 - wyb$$

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

求出参数值:

$$u_3 = q_3/p_3, \quad u_4 = q_4/p_4$$

$$u_A = 0, \quad u_B = 1,$$



(a) 直线段与窗口边界
 wxl 和 wxr 平行的情况

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

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

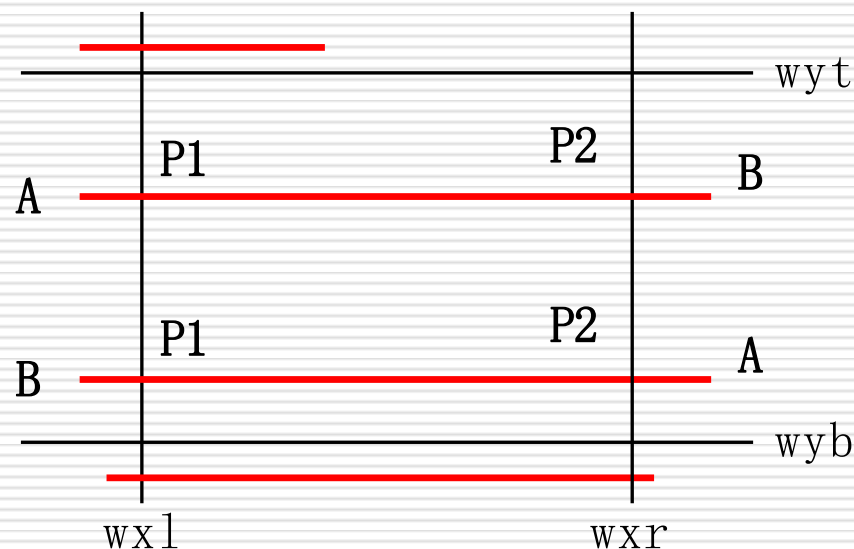
$$p_1 = -(x_2 - x_1) \quad q_1 = x_1 - wxl$$

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

求出参数值:

$$u_1 = q_1/p_1, \quad u_2 = q_2/p_2$$

$$u_A = 0, \quad u_B = 1,$$



(b) 直线段与窗口边界
 wyb 和 wyt 平行的情况

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

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

$$p_1 = -(x_2 - x_1) \quad q_1 = x_1 - wxl.$$

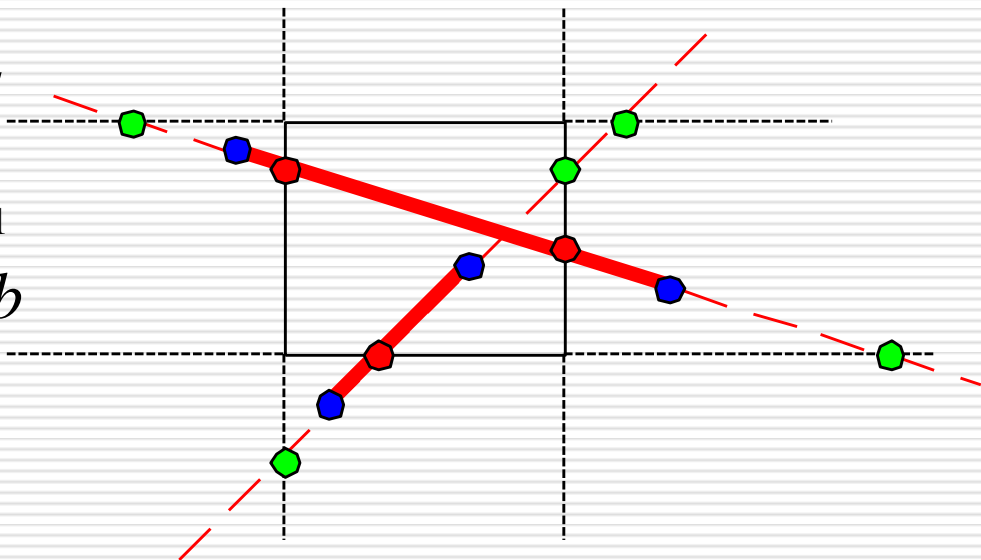
$$p_2 = x_2 - x_1$$

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

$$q_3 = y_1 - wxb$$

$$p_4 = y_2 - y_1$$

$$q_4 = wyt - y_1$$



一般情况:

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

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

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 $q_3 > 0$ 且 $q_4 > 0$, 则进一步计算 u_3 和 u_4 。算法转(5)。
- (4) 若上述两条均不满足, 则有 $p_k \neq 0$ ($k=1, 2, 3, 4$)。此时计算 u_1, u_2, u_3 和 u_4 。求出 (u_{max}, u_{min}) 赋值给 (u_1, 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。
- (7) 算法结束。

```

#define TRUE 1
#define FALSE 0
int clip(float d, float q, float *t1, float *t2)
{
    float t; int retVal=TRUE;

    if(d<0.0){
        t=q/d;
        if(t>*t2)
            retVal=FALSE;
        else
            if(t>*t1) //t1 中取最大的
            *****
                *t1=t;
    }else
        if(d>0.0){
            t=q/d;
            if(t<*t1)
                retVal=FALSE;
            else
                if(t<*t2) //t2 中取最小的
                *****
                    *t2=t;
        }else//此时d==0.0
            if(q<0.0) retVal=FALSE;
    return(retVal);
}

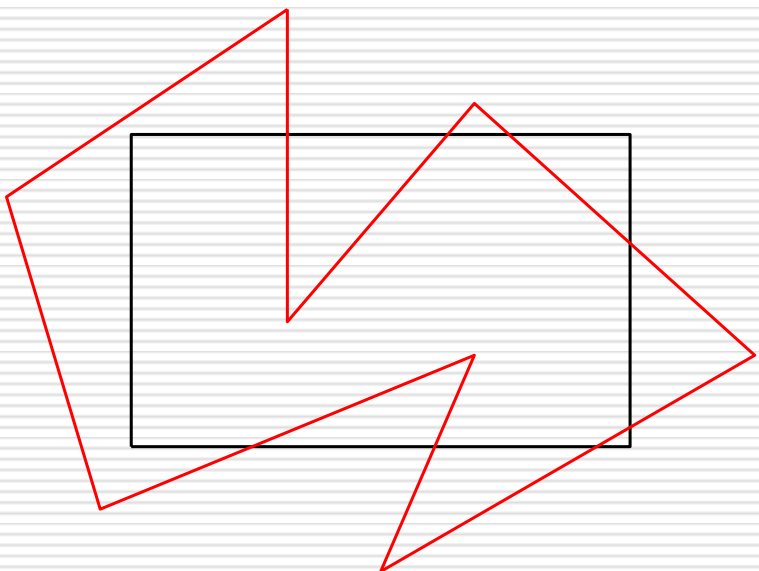
```

```
void LiangBarskyClipLine(int xwmin, int ywmin, int xwmax,
    int ywmax, float x1, float y1, float x2, float y2)
{
    float t1=0,t2=1,dx=x2-x1,dy;
    if(clip(-dx, x1-xwmin, &t1, &t2))
        if(clip(dx, xwmax-x1, &t1, &t2)){
            dy=y2-y1;
            if(clip(-dy, y1-ywmin, &t1, &t2))
                if(clip(dy, ywmax-y1, &t1, &t2)){

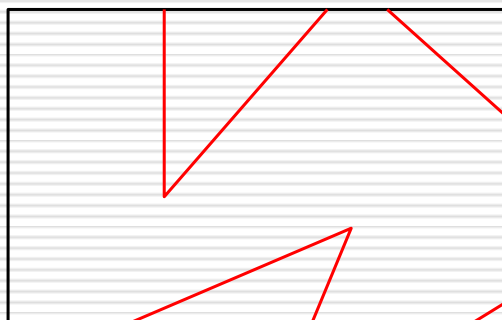
                    if(t2<1.0){
                        x2=x1+t2*dx;
                        y2=y1+t2*dy;
                    }
                    if(t1>0.0){
                        x1=x1+t1*dx;
                        y1=y1+t1*dy;
                    }
                    dda_line(x1, y1, x2, y2);
                }
            }
        }
}
```

多边形的裁剪

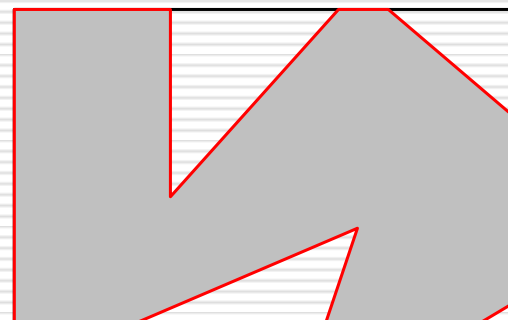
问题的提出:



(a) 裁剪前



(b) 直接采用直线段
裁剪的结果



(c) 正确的裁剪结果

Sutherland-Hodgeman 多边形裁剪

- 基本思想：将多边形的边界作为一个整体，每次用窗口的一条边界对要裁剪的多边形进行裁剪，体现分而治之的思想。

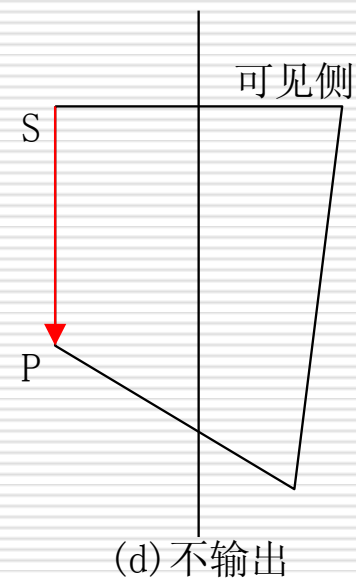
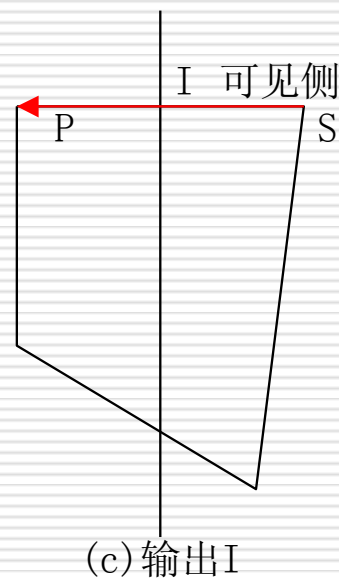
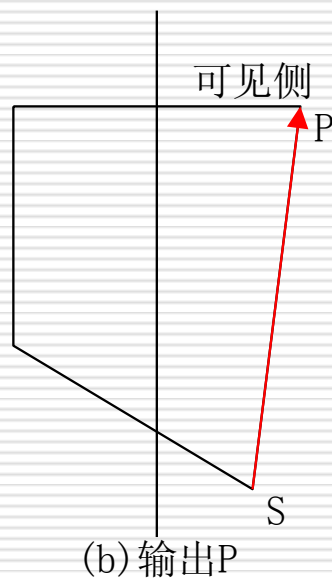
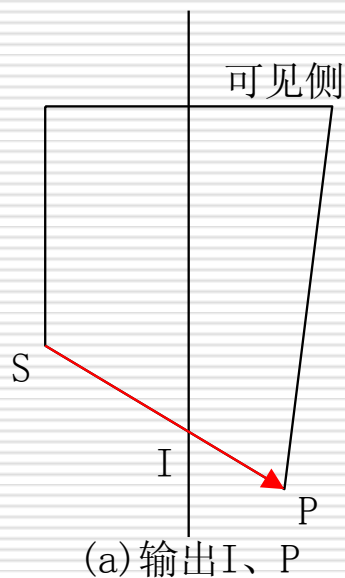
Sutherland-Hodgeman 多边形裁剪

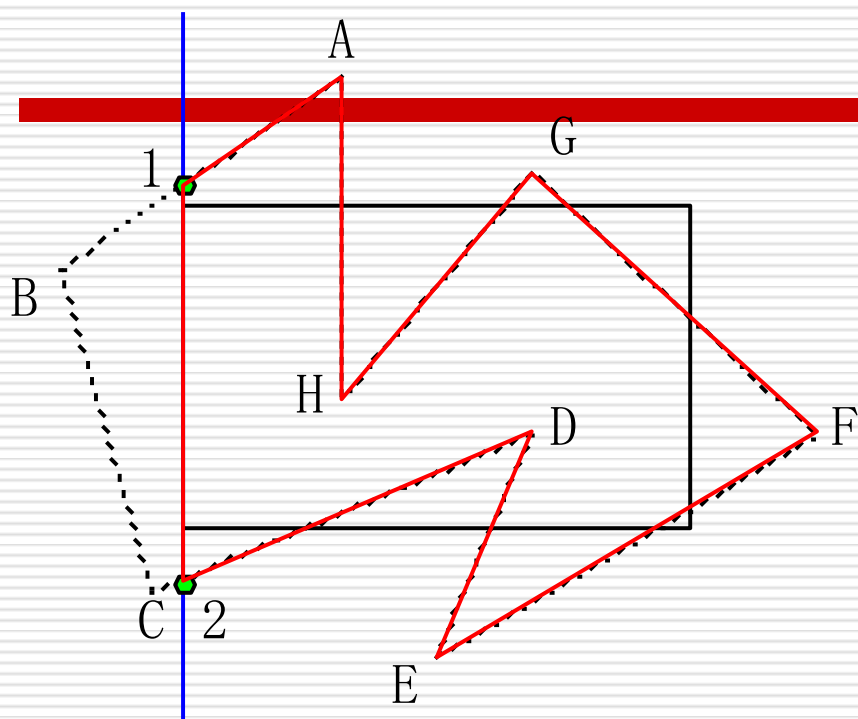
□ 算法实施策略:

- 为窗口各边界裁剪的多边形存储输入与输出顶点表。在窗口的一条裁剪边界处理完所有顶点后，其输出顶点表将用窗口的下一条边界继续裁剪。
- 窗口的一条边以及延长线构成的裁剪线把平面分为两个区域，包含窗口区域的区域称为可见侧；不包含窗口区域的域为不可见侧。

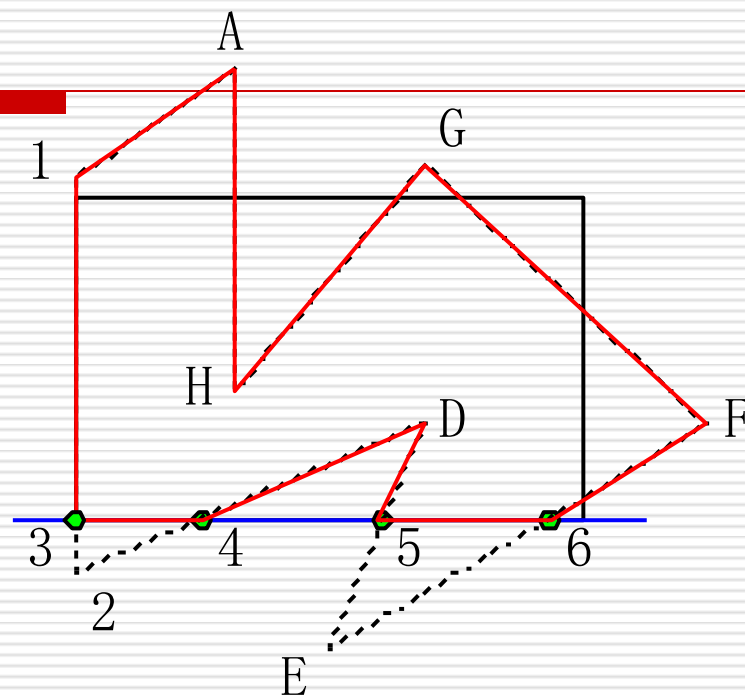
Sutherland-Hodgeman 多边形裁剪

■ 沿着多边形依次处理顶点会遇到四种情况：

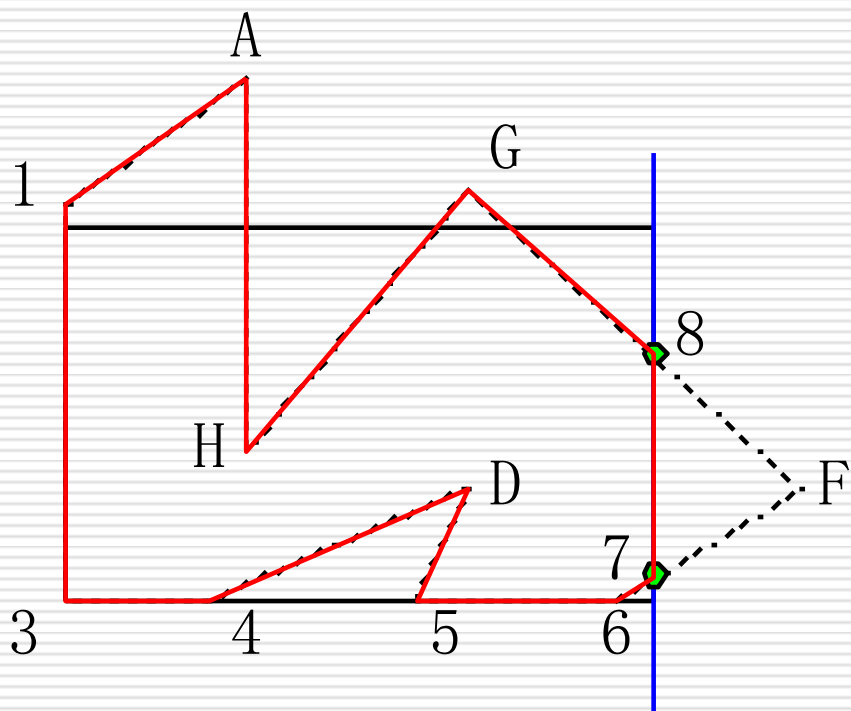




输入: ABCDEFGH
 输出: 12DEFGHA
 (a) 用左边界裁剪

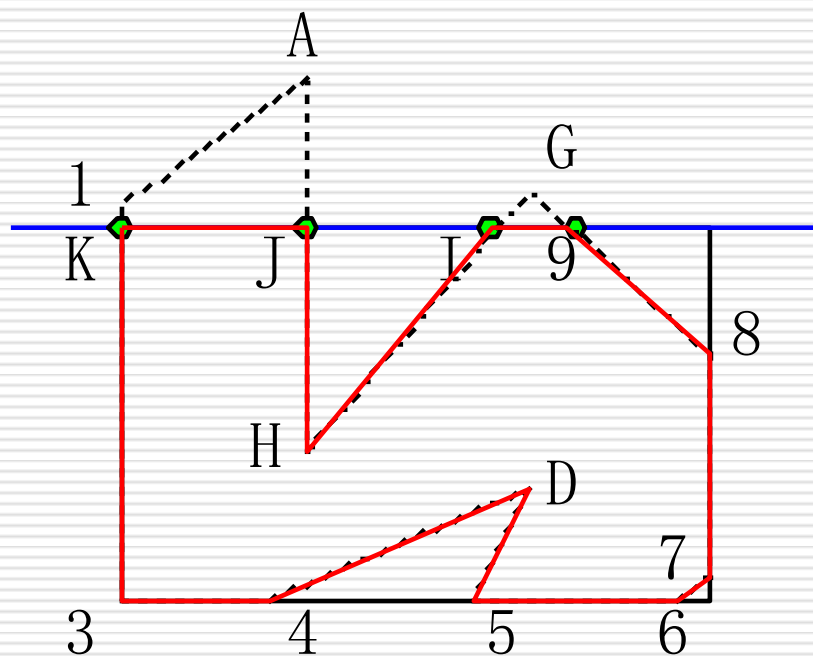


输入: 12DEFGHA
 输出: 34D56FGHA1
 (b) 用下边界裁剪



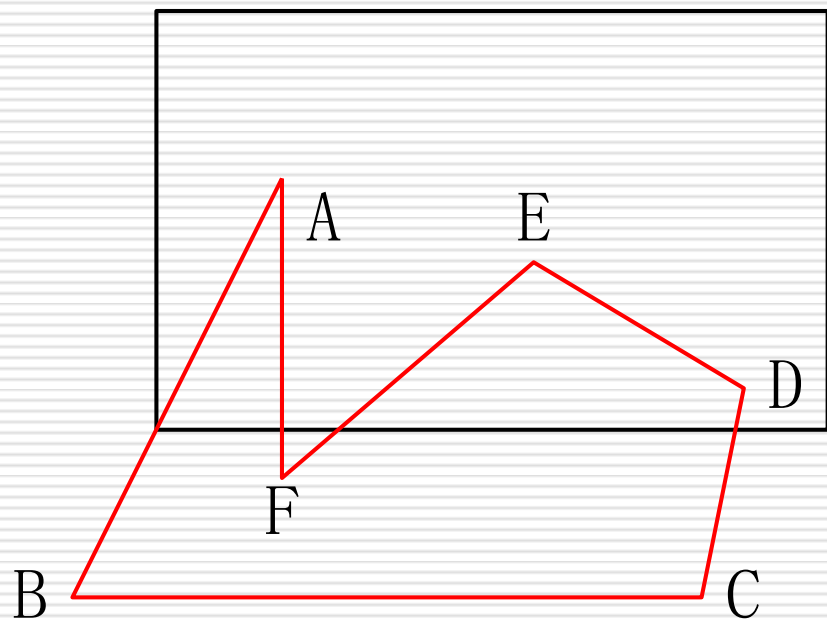
输入：34D56FGHA1

(c) 用右边界裁剪

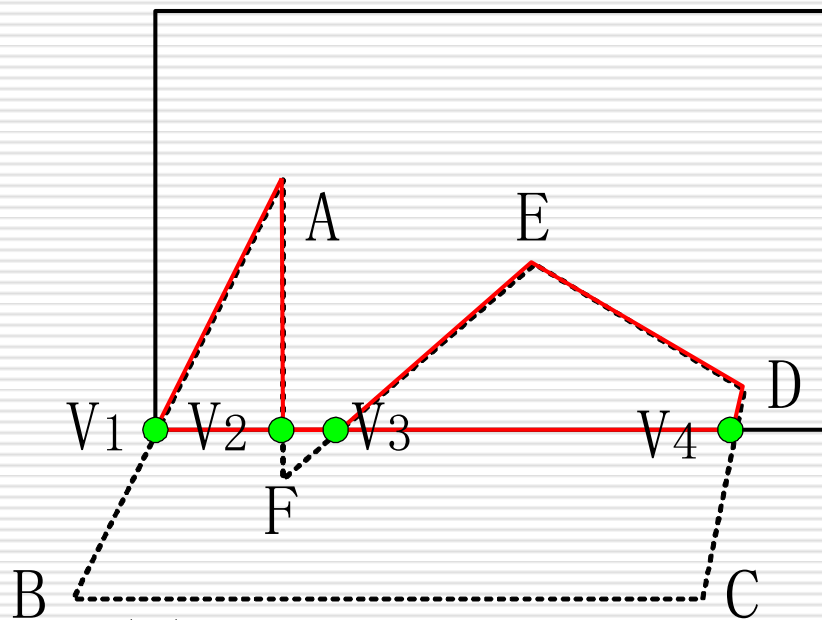


(d) 用上边界裁剪

特点



(a) 裁剪前



(b) Sutherland-Hodgeman
算法的裁剪结果

Weiler-Atherton 多边形裁剪

□ 假定按顺时针方向处理顶点，且将用户多边形定义为 P_s ，窗口矩形为 P_w 。算法从 P_s 的任一点出发，跟踪检测 P_s 的每一条边，当 P_s 与 P_w 相交时（实交点），按如下规则处理：

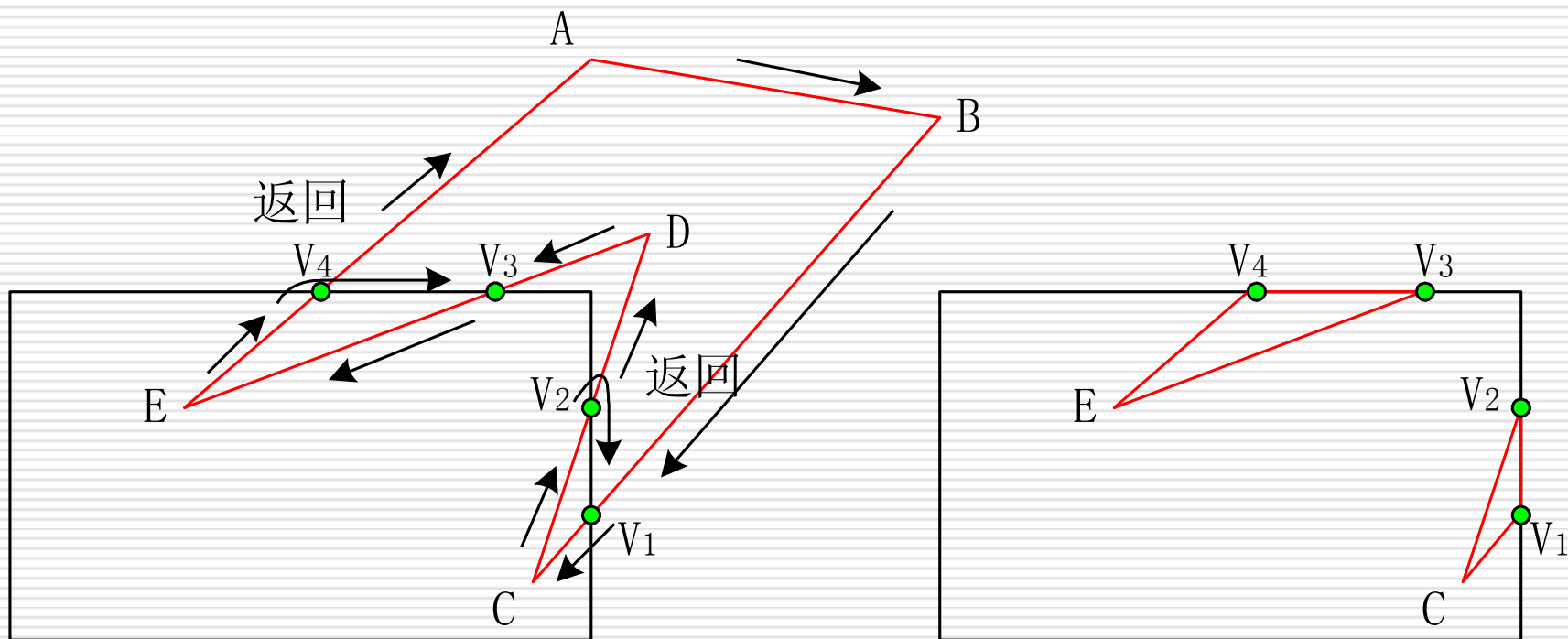
(1)若是由不可见侧进入可见侧，则输出可见直线段，转(3)；

Weiler-Atherton 多边形裁剪

(2)若是由可见侧进入不可见侧，则从当前交点开始，沿窗口边界顺时针检测 P_w 的边，即用窗口的有效边界去裁剪 P_s 的边，找到 P_s 与 P_w 最靠近当前交点的另一交点，输出可见直线段和由当前交点到另一交点之间窗口边界上的线段，然后返回处理的当前交点；

(3)沿着 P_s 处理各条边，直到处理完 P_s 的每一条边，回到起点为止。

□ 下图示了Weiler-Atherton算法裁剪凹多边形的过程和结果。



(a) 裁剪前

(b) Weiler-Atherton算法的裁剪结果

图6-34 Weiler-Atherton算法裁剪凹多边形

其他裁剪

2. 文字裁剪

文字裁剪的策略包括几种：

- 串精度裁剪
- 字符精度裁剪
- 笔划、象素精度裁剪

3. 外部裁剪

保留落在裁剪区域外的图形部分、去掉裁剪区域内的所有图形，这种裁剪过程称为外部裁剪，也称空白裁剪。

6.6 OpenGL中的二维观察

- 指定矩阵堆栈
- 指定裁剪窗口
- 指定视区

指定矩阵堆栈

- 指定当前操作的是投影矩阵堆栈

glMatrixMode (GL_PROJECTION)

- 初始化，即指定当前操作的矩阵堆栈的栈顶元素为单位矩阵。

glLoadIdentity();

指定裁剪窗口

□ 定义二维裁剪窗口

`gluOrtho2D(xwmin, xwmax, ywmin, ywmax);`

□ 其中，双精度浮点数 **xwmin**, **xwmax**, **ywmin**, **ywmax** 分别对应裁剪窗口的左、右、下、上四条边界。

□ 默认的裁剪窗口，四条边界分别为 **wxl=-1.0**, **wxr=1.0**, **wyb=-1.0**, **wyt=1.0**。

指定裁剪窗口

□ 指定视区

`glViewport (xvmin, yvmin, vpWidth, vpHeight) ;`

□ **xvmin**和**yvmin**指定了对应于屏幕上显示窗口中的矩形视区的左下角坐标，单位为像素。

□ 整型值**vpWidth**和**vpHeight**则指定了视区的宽度和高度。

□ 默认的视区大小和位置与显示窗口保持一致。