裁剪

冯结青

浙江大学 CAD&CG国家重点实验室

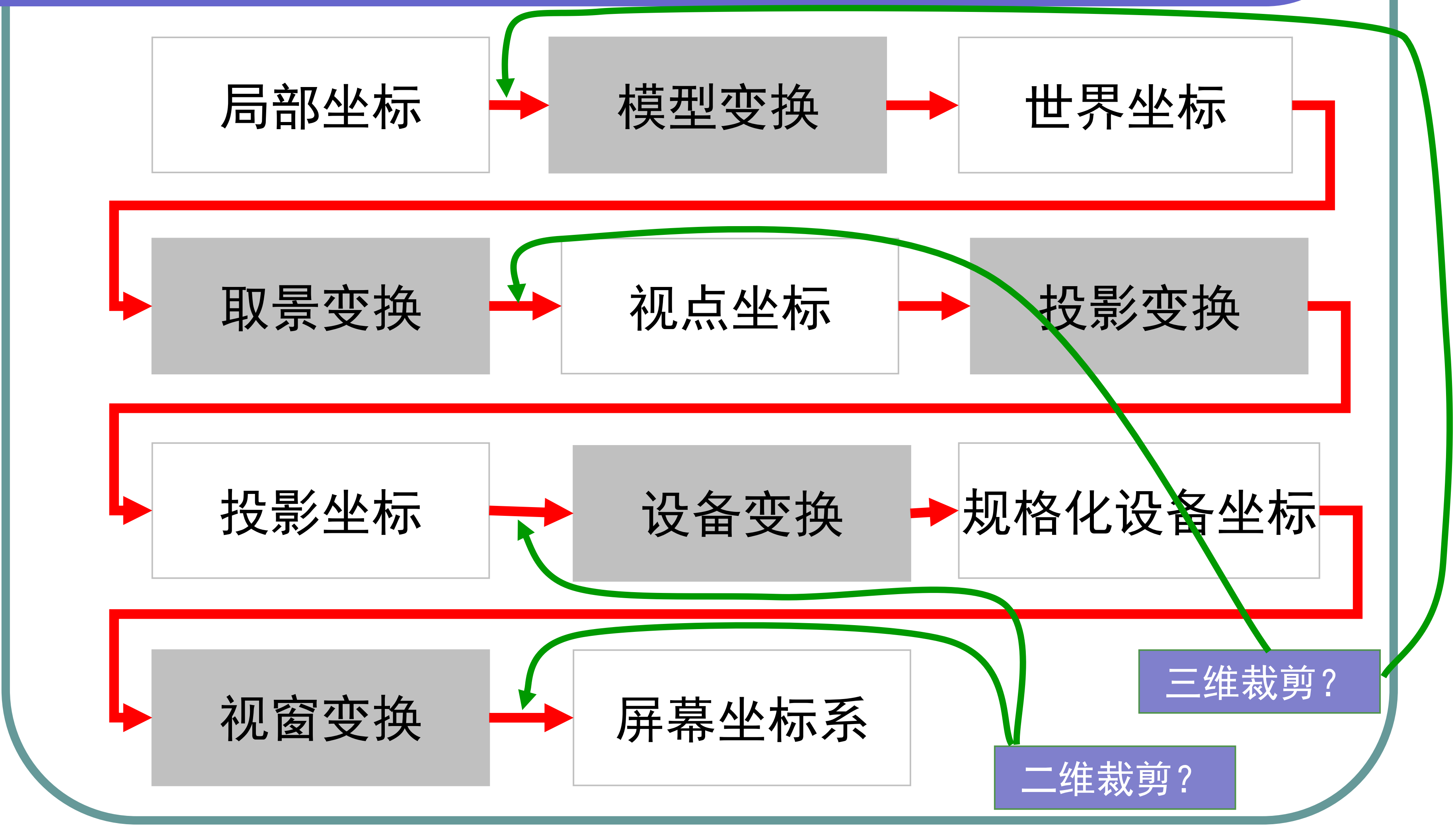
裁剪

- 裁剪
- 二维线裁剪
  - Cohen-Sutherland 裁剪算法
  - 中点分割算法
  - 梁友栋-Barsky 裁剪算法
- 二维多边形裁剪
- 文本裁剪
- 三维裁剪
- 关于三维变换与裁剪

裁剪

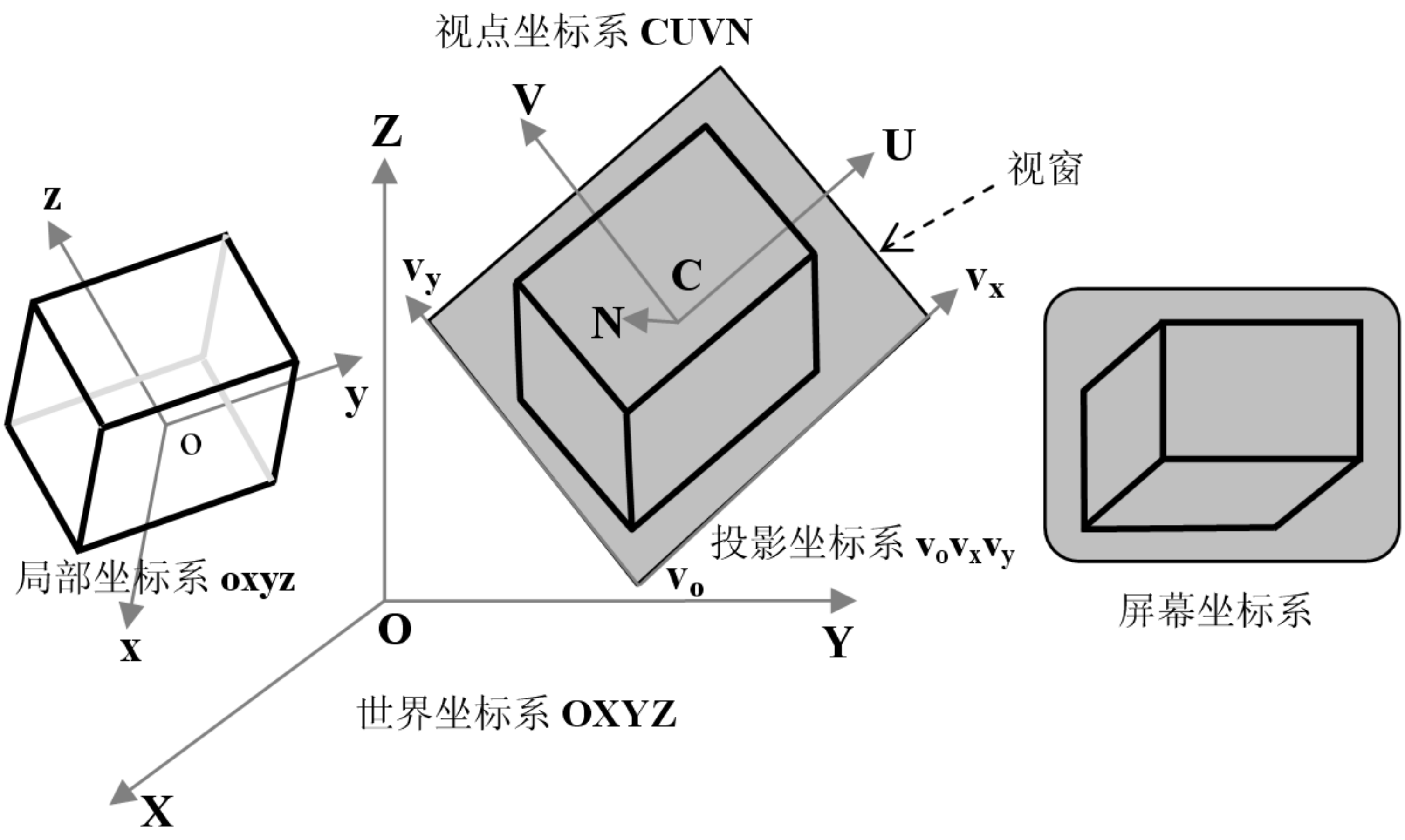
- 裁剪
- 二维线裁剪
- 二维多边形裁剪
- 文本裁剪
- 三维裁剪
- 关于三维变换与裁剪

三维变换流程图



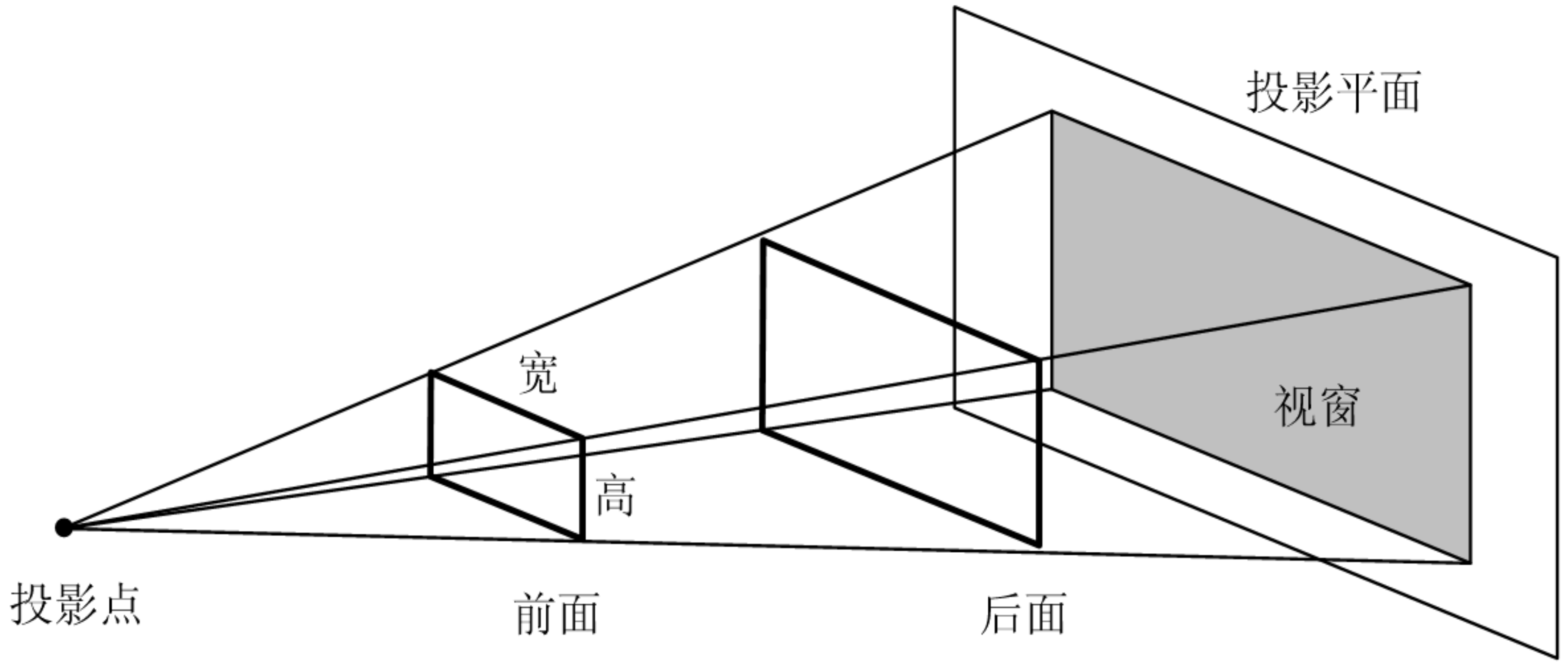
局部坐标
模型变换
世界坐标
取景变换
视点坐标
投影变换
投影坐标
设备变换
规格化设备坐标
视窗变换
屏幕坐标系
三维裁剪？
二维裁剪？

三维变换中的各种坐标系



三维变换中的各种坐标系

视域四棱锥裁剪



透视投影中的视域四棱

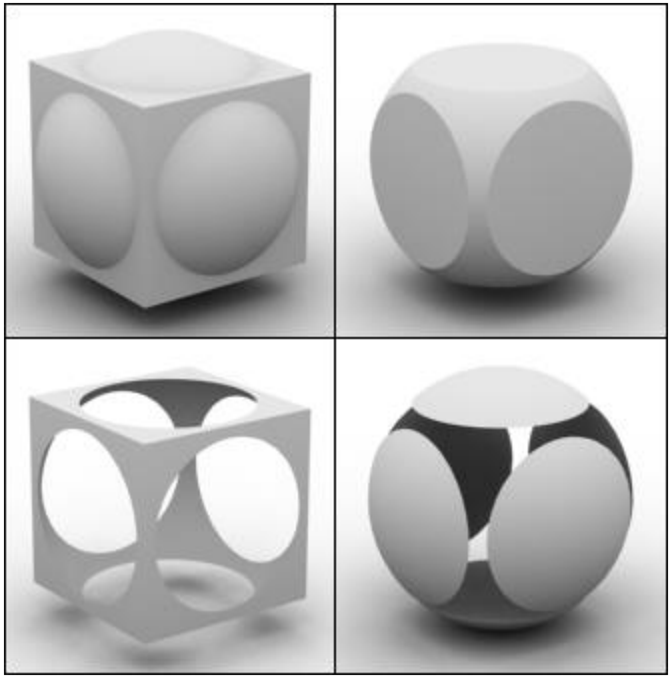
屏幕坐标系和设备变换

- 在投影平面上，有一个矩形区域称为视窗(窗口)，见上图坐标系$v_o v_x v_y$中的矩形和“视域四棱锥”图中的矩形
- 二维变换
  - 设备变换：将定义投影平面中的视窗变换到标准设备坐标内的规范化视区
  - 视窗变换：将规范化视区转换到以像素为单位的屏幕坐标
  - 扫描转换：将连续的几何物体转换为离散的光栅表示

裁剪(Clipping)

- 裁剪是确定场景或画面中位于**给定区域**(2D或3D裁剪窗口)之内的部分
- 裁剪还可用于图形反走样、线消隐、面消隐、阴影、纹理等算法中
- 裁剪算法的推广应用：
  - 多面体对多面体的裁剪，实体造型中的布尔运算
  - 在窗口系统中复制、移动或删除画面中某一部分(Cut-Copy-Paste)

裁剪的应用举例



实体造型中的布尔操作

Copy & Paste操作

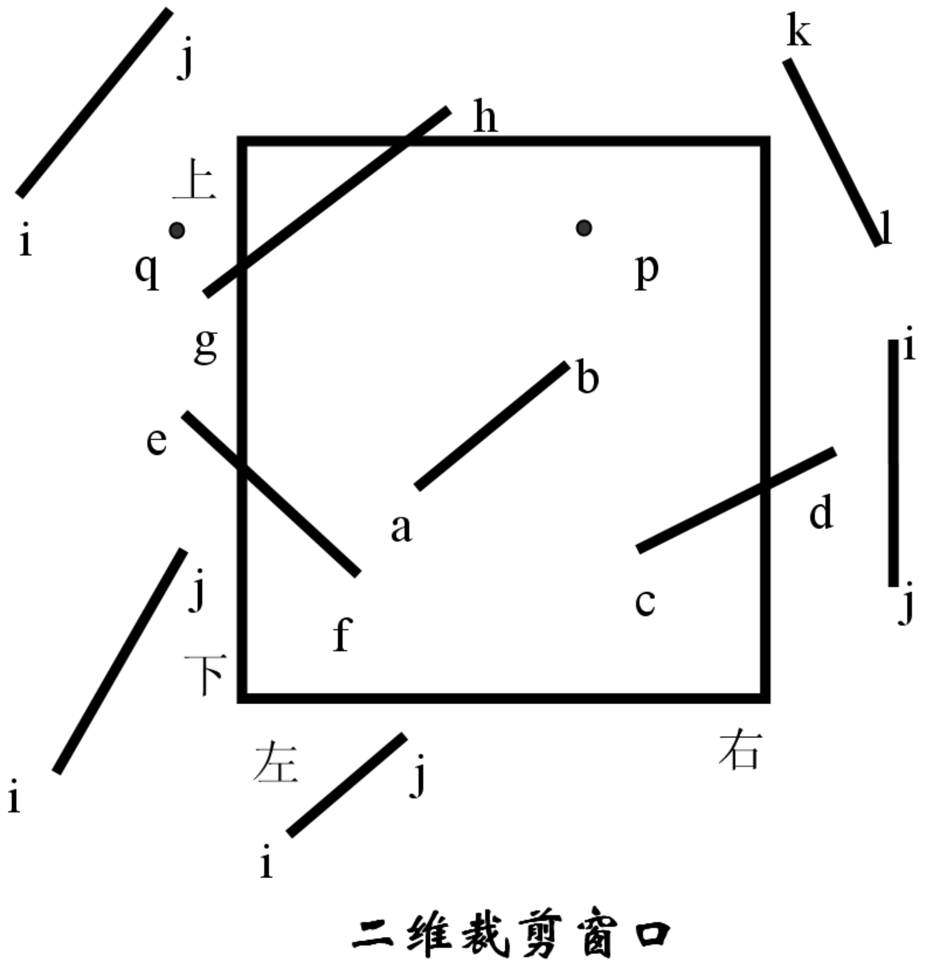
裁剪

- 裁剪算法分类：
  - 裁剪窗口的维数：二维、三维
  - 裁剪窗口：规则(矩形、六面体)和不规则的(任意多边形和多面体)
  - 对象维数：点、线、多边形、多面体
  - 实现方式：软件和硬件实现

裁剪

- 裁剪
- 二维线裁剪
  - Cohen-Sutherland 裁剪算法
  - 中点分割算法
  - 梁友栋-Barsky 裁剪算法
- 二维多边形裁剪
- 文本裁剪
- 三维裁剪
- 关于三维变换与裁剪

二维线裁剪



二维线裁剪

- 二维线裁剪
  - 判断并计算位于裁剪窗口内线段或部分线段
  - 位于窗口内线段或部分线段被保留用于显示，而其它部分则被抛弃
- 裁剪算法的效率十分重要
  - 需要对场景之中大量的线段进行裁剪
  - 核心：快速拒绝和接受

裁剪

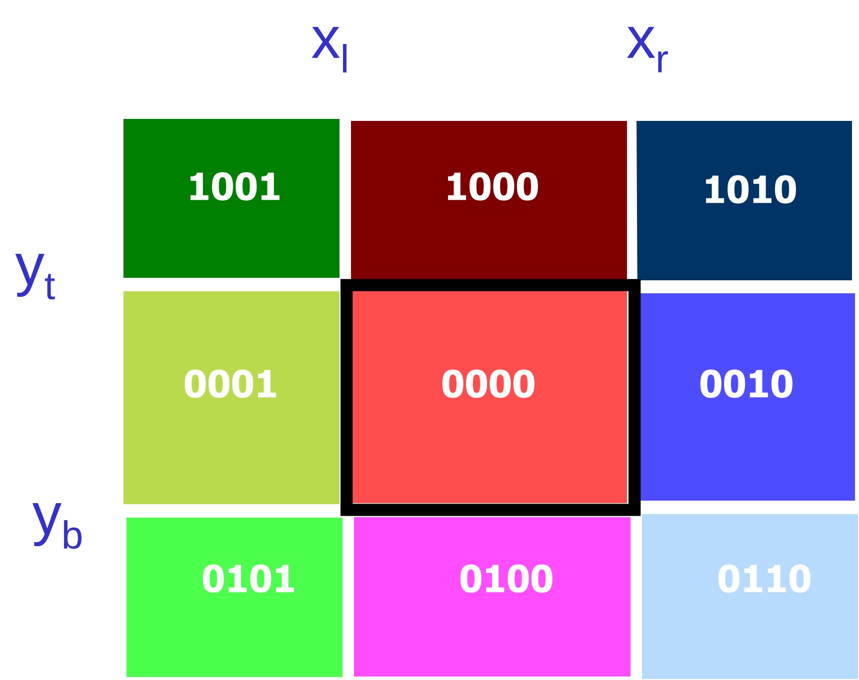
- 裁剪
- 二维线裁剪
  - Cohen-Sutherland 裁剪算法
  - 中点分割算法
  - 梁友栋-Barsky 裁剪算法
- 二维多边形裁剪
- 文本裁剪
- 三维裁剪
- 关于三维变换与裁剪

Cohn-Sutherland裁剪

- 平面矩形裁剪窗口
- 算法思想
  - 直线段端点的编码
  - 快速拒绝/接受判断：相对于裁剪窗口
    - 完全可见 √
    - 完全不可见 √
    - 部分可见……
  - 部分可见线段的求交

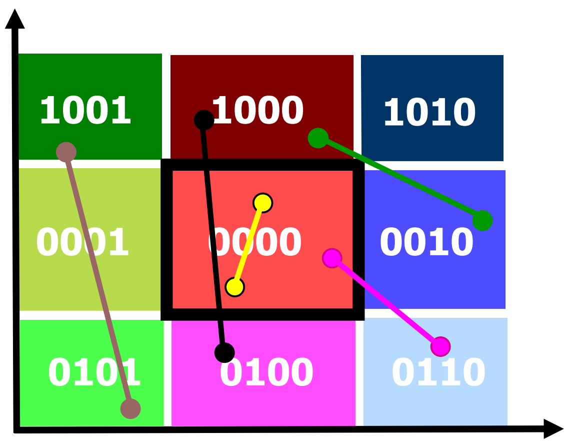
直线段端点的4bit编码

- 裁剪窗口的边将平面分成九个区域，对于每一个区域进行4bit编码$C_tC_bC_rC_l$
  - $C_l=1$ if $x < x_l$
  - $C_r=1$ if $x > x_r$
  - $C_b=1$ if $y < y_b$
  - $C_t=1$ if $y > y_t$



直线段端点的4bit编码

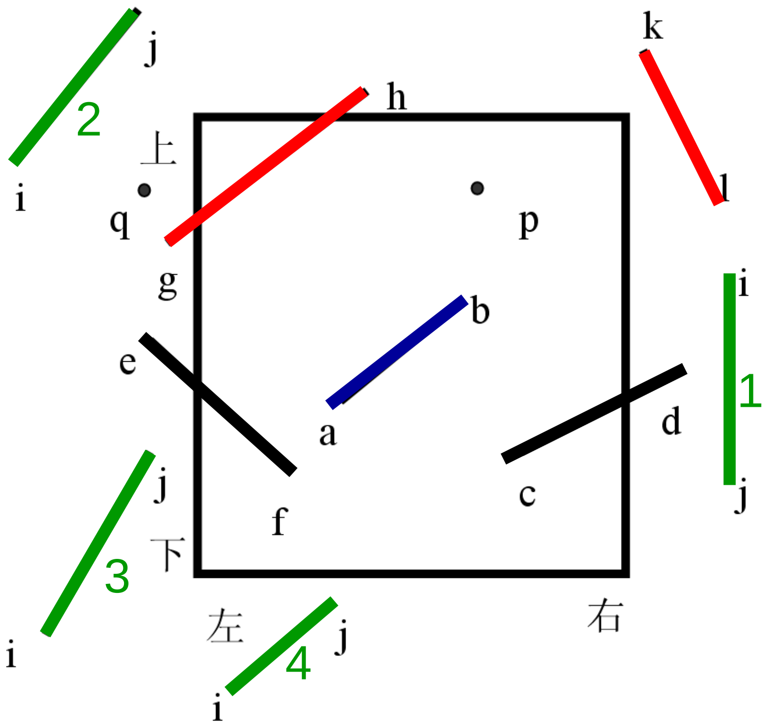
- 线段两端点的编码均为零，即两端点均在窗口之内，则线段可见
- 将线段两端点的编码逐位取逻辑“与”，若结果非零，则该线段必为完全不可见线，因而可立即抛弃
- 其它情形为部分可见和不可见，此时需要求交



编码举例

1001	1000	1010
0001	0000	0010
0101	0100	0110

线 段 端 点 的 区 域 编 码



二维裁剪窗口

线段	端点编码	逻辑与	注释
ab	0000 0000	0000	完全可见
*ij*1	0010 0010	0010	完全不可见
*ij*2	0001 1001	0001	完全不可见
*ij*3	0101 0001	0001	完全不可见
*ij*4	0100 0100	0100	完全不可见
cd	0000 0010	0000	部分可见
ef	0001 0000	0000	部分可见
gh	0001 1000	0000	进一步判断
kl	1000 0010	0000	进一步判断

直线段与窗口求交

- 窗口：　$(x_{Left}, x_{Right}, y_{Top}, y_{Bottom})$
- 直线段：$\mathbf{P}_1(x_1,y_1)$和$\mathbf{P}_2(x_2,y_2)$
- 直线的显式方程：

$$y = m(x - x_1) + y_1 \quad \text{或} \quad y = m(x - x_2) + y_2$$

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$ 为直线段斜率

直线段与窗口求交

- 它与窗口诸边的交点

左：x_L， $y = m(x_L - x_1) + y_1, \quad m \neq \infty$

右：x_R， $y = m(x_R - x_1) + y_1, \quad m \neq \infty$

上：y_T， $x = x_1 + (1/m)(y_T - y_1), \quad m \neq 0$

下：y_B， $x = x_1 + (1/m)(y_B - y_1), \quad m \neq 0$

直线段与窗口求交

- 特殊情形的考虑
  - 若直线的斜率为无穷大，则直线平行于窗口的左边和右边，仅需检查直线与上、下两边的交点
  - 若直线斜率为零，则它平行于窗口的上、下两边，仅需检查直线与左、右两边的交点

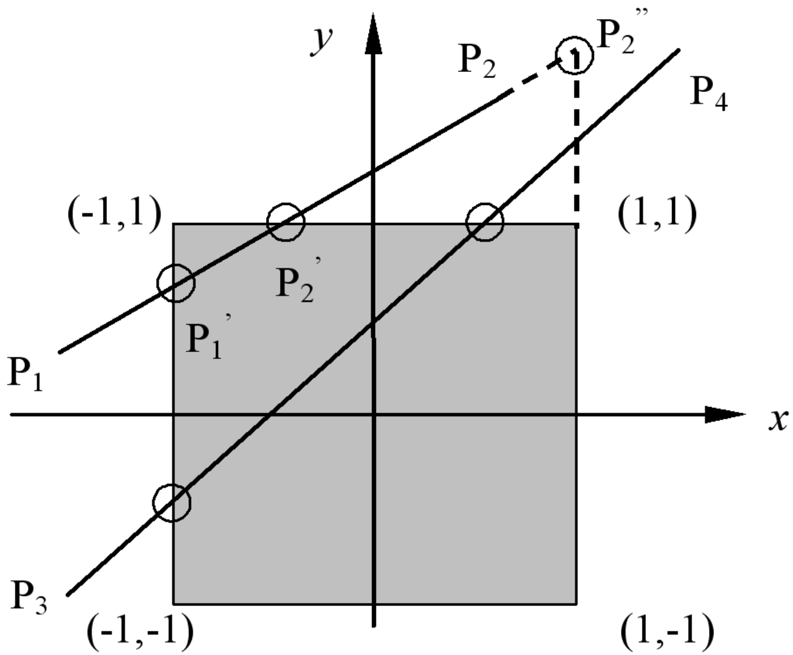
部分可见线段的处理

- Cohen-Sutherland算法的关键：
  - 得知位于窗口之外的一个端点
  - 此端点至(线段与窗口)交点之间的区段必为不可见，故可抛弃。
- 然后，算法继续处理被裁剪后的剩余线段，此时取交点来代替被裁剪线段的一个端点。

Cohen-Sutherland算法的描述

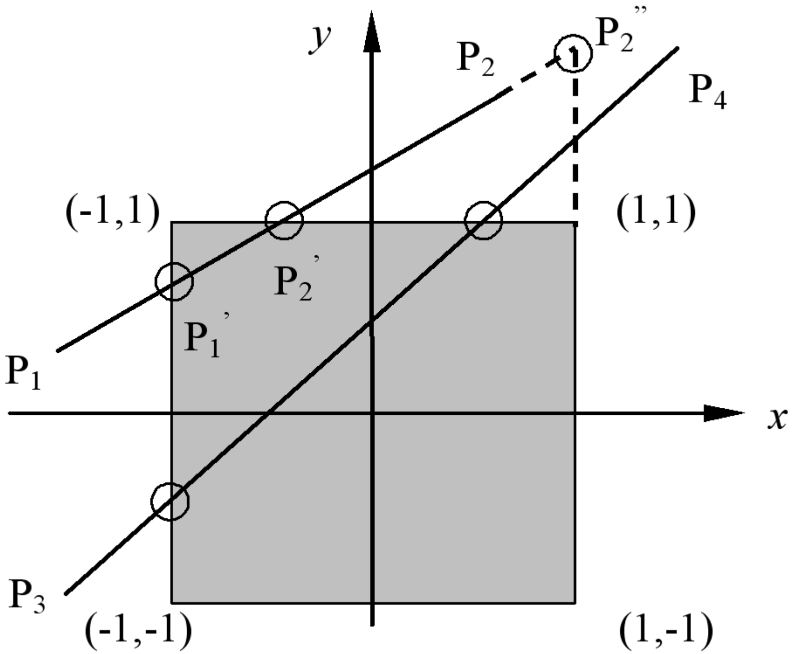
- 对于每个窗口边
  - 检查线段P_1P_2是否为完全可见段或可以抛弃的显然不可见线段
    - 完全可见：保留用于绘制
    - 完全不可见：抛弃
    - 部分可见：如下规则继续判断
  - 若P_1在窗口外，继续执行算法；否则交换
  - 用P_1P_2和窗口边的交点取代点P_1

Cohen-Sutherland算法实例



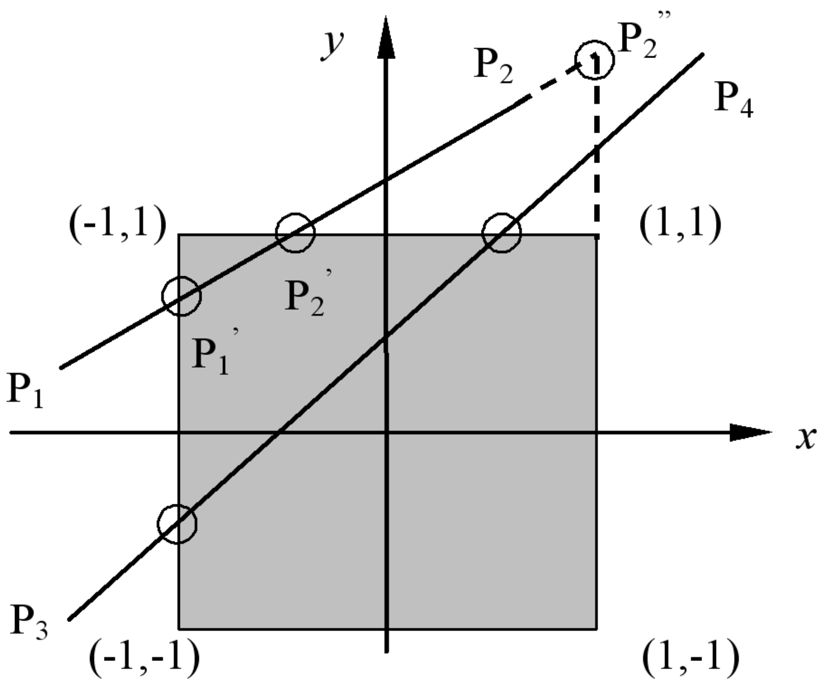
线段端点$P_1(-3/2,1/6)$和$P_2(1/2,3/2)$的编码分别为(0001)和(1000)。两个端点编码不全为0，逻辑与结果为0。因此该线段既非完全可见，也不是显然不可见。比较两端点编码的第一位可以发现该线段跨越窗口的左边界，并且端点P_1位于窗外。

Cohen-Sutherland算法实例



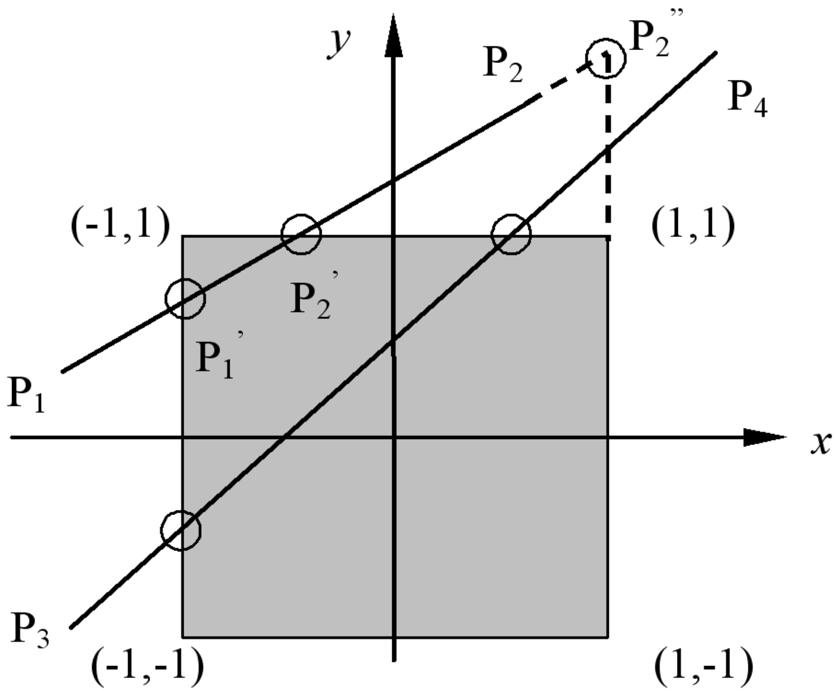
线段与窗口左边(x=-1)的交点为$P_1'(-1,1/2)$。用P_1'取代P_1得到新线段$P_1'(-1,1/2)$ $P_2(1/2,3/2)$

Cohen-Sutherland算法实例



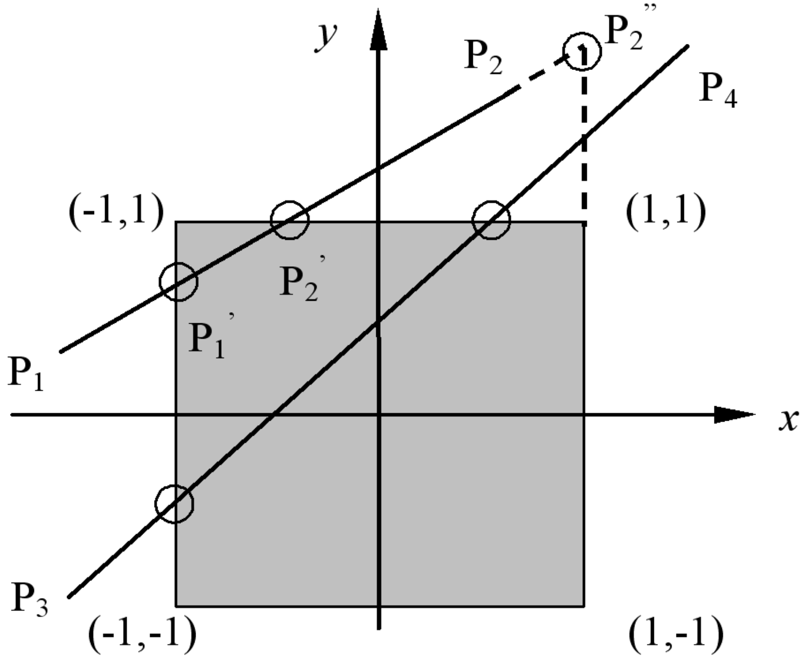
现在端点P_1、P_2的编码分别为(0000)和(1000)。新线段仍非完全可见或显然不可见。比较端点编码的第二位，发现线段并不跨越窗口右边界，转而考察窗口底部边界。

Cohen-Sutherland算法实例



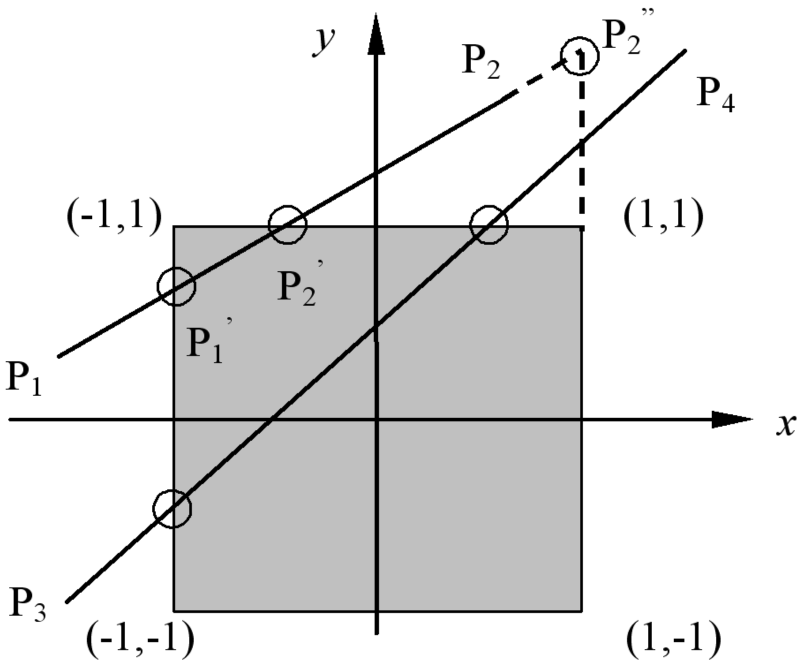
端点$\mathbf{P}_1$、$\mathbf{P}_2$的编码仍然为(0000)和(1000)，此线段既非完全可见，也不是完全不可见。比较端点编码的第三位，发现线段并不跨越窗口底部边界，转而考察窗口顶部边界。

Cohen-Sutherland算法实例



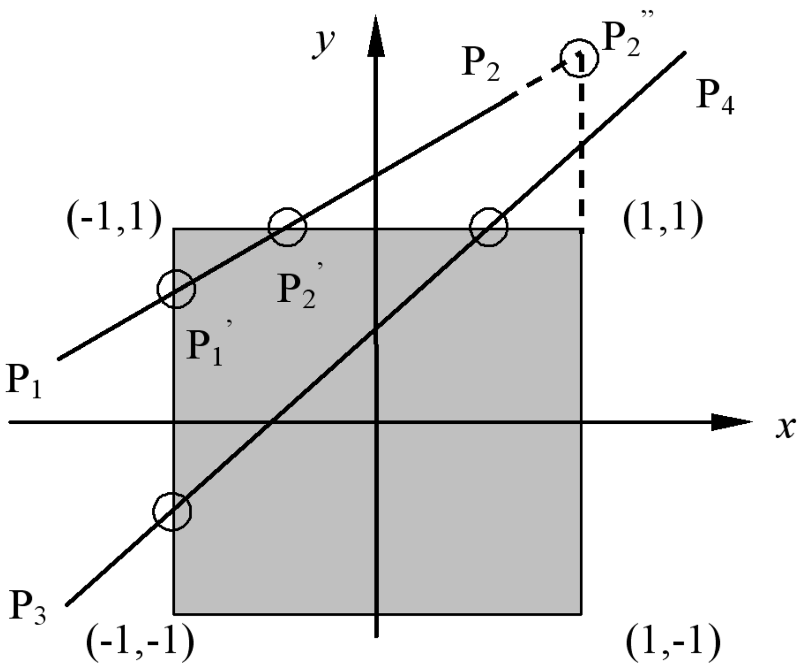
端点$\mathbf{P}_1$、$\mathbf{P}_2$的编码仍然为(0000)和(1000)，此线段既非完全可见，也不是完全不可见。比较端点编码的第四位，发现该线段跨越窗口顶部边界，$\mathbf{P}_1$不在窗外，交换$\mathbf{P}_1$、$\mathbf{P}_2$得到一新线段$\mathbf{P}_1(1/2,3/2)\mathbf{P}_2(-1,1/2)$

Cohen-Sutherland算法实例



线段同窗口顶边界(y=1)的交点是$\mathbf{P}_1'(-1/4,1)$。用$\mathbf{P}_1'$取代$\mathbf{P}_1$得到新线段$\mathbf{P}_1(-1/4,1)\mathbf{P}_2(-1,1/2)$

Cohen-Sutherland算法实例



端点P_1、P_2的编码分别为(0000)和(0000)，该线段完全可见。裁剪过程结束。

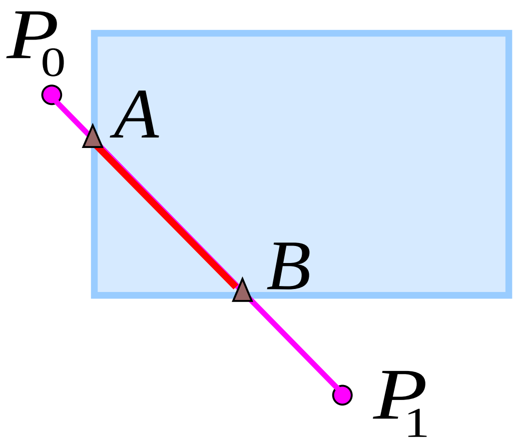
思考：如何准确地设定直线段与窗口交点的编码？

裁剪

- 裁剪
- 二维线裁剪
  - Cohen-Sutherland 裁剪算法
  - 中点分割算法
  - 梁友栋-Barsky 裁剪算法
- 二维多边形裁剪
- 文本裁剪
- 三维裁剪
- 关于三维变换与裁剪

中点分割算法

基本思想



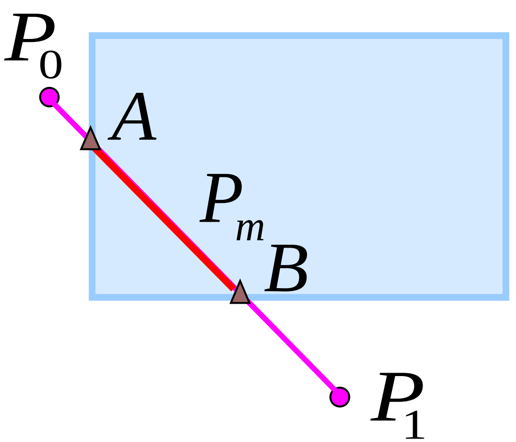
从P_0出发找出离P_0最近的可见点A

从P_1出发找出离P_1最近的可见点B

则线段AB即为原线段在窗口内的部分

中点分割算法

基本思想



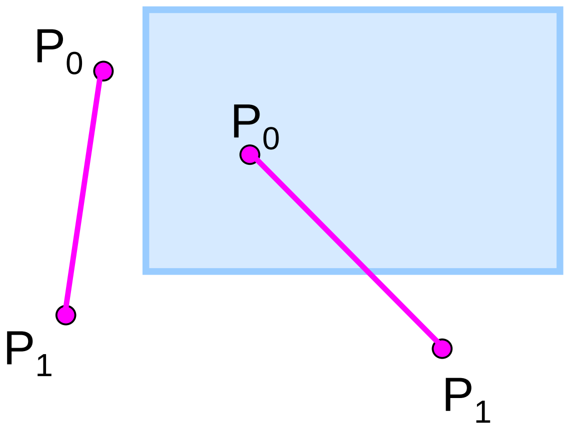
求最近的可见点：

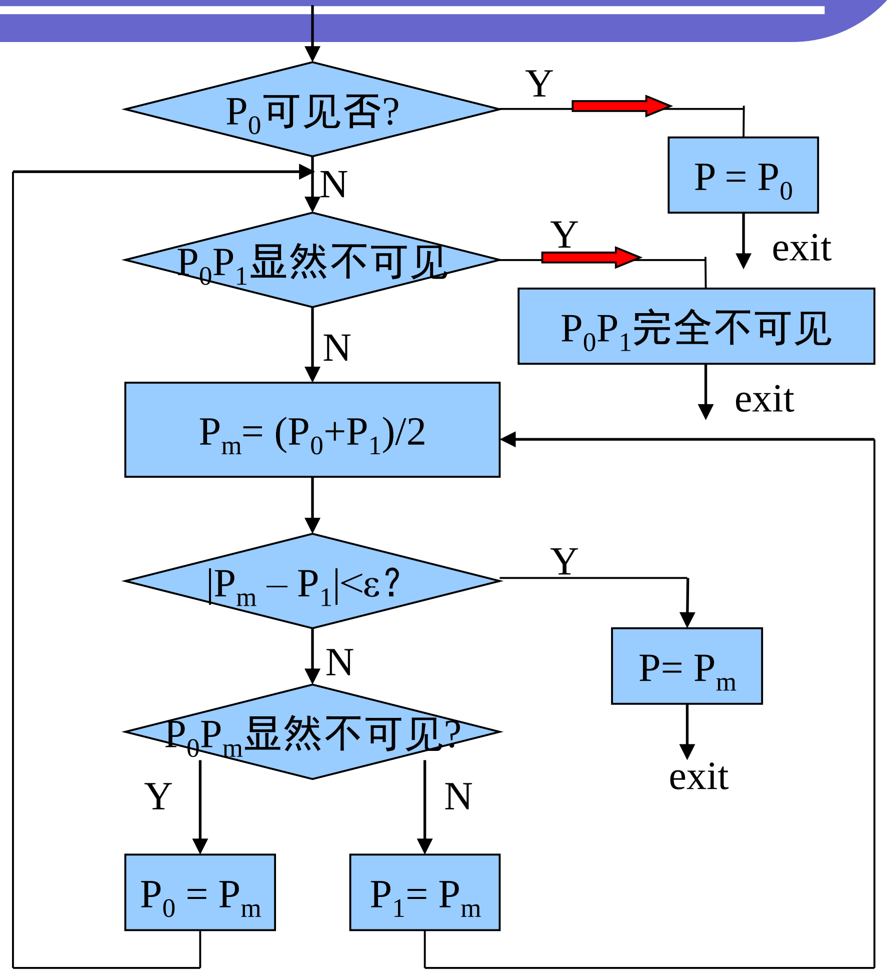
取线段中点

$$P_m = \frac{(P_0 + P_1)}{2}$$

中点分割算法

从P_0出发找出离P_0最近的可见点P

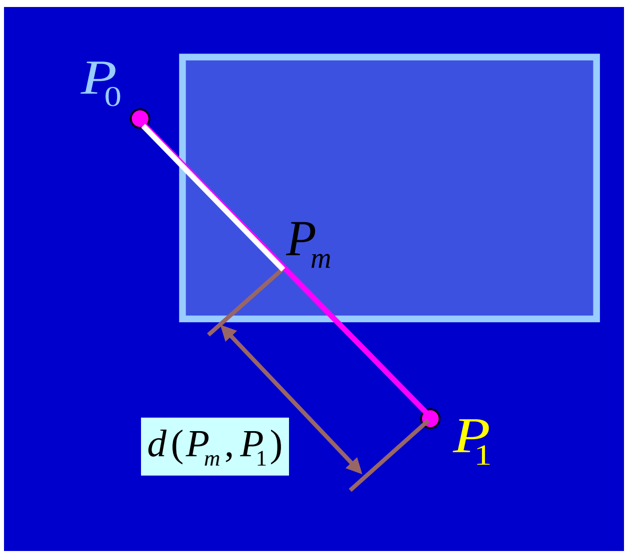


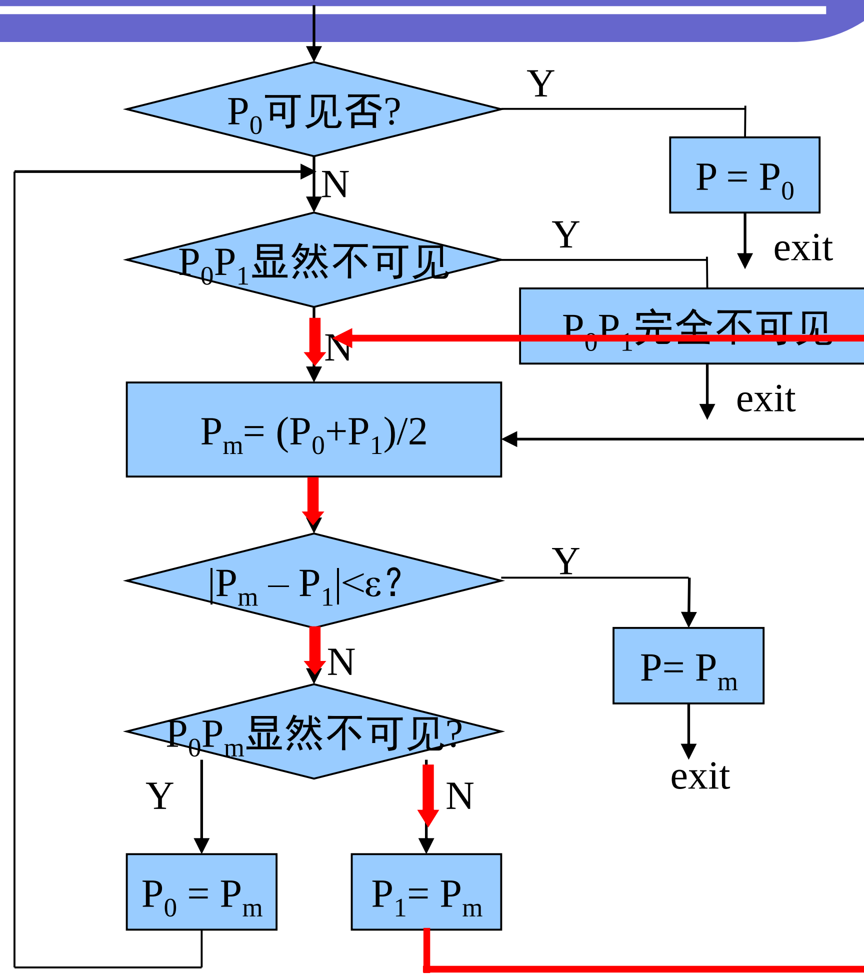


中点分割算法框图

中点分割算法

从P_0出发找出离P_0最近的可见点P

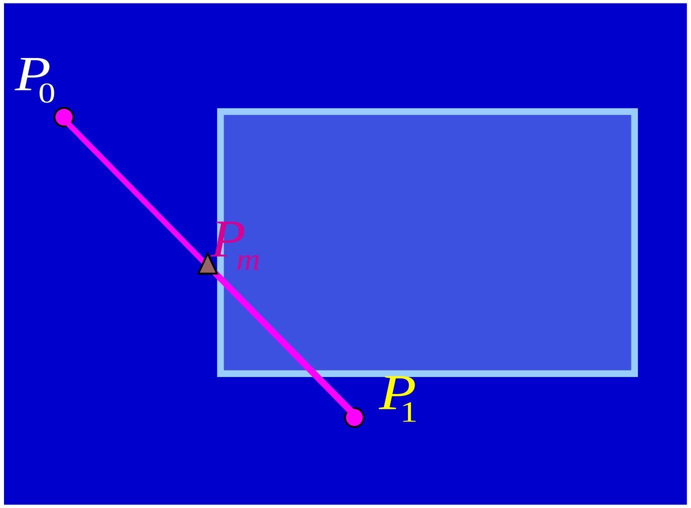


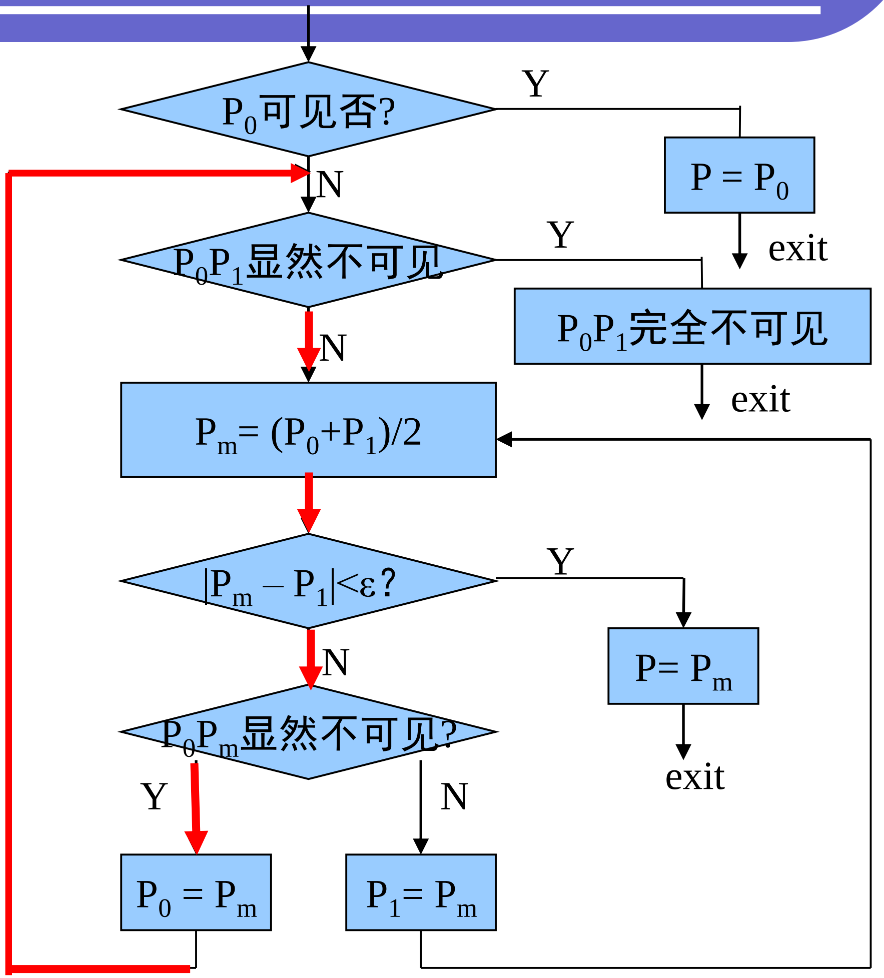


中点分割算法框图

中点分割算法

从P_0出发找出离P_0最近的可见点P

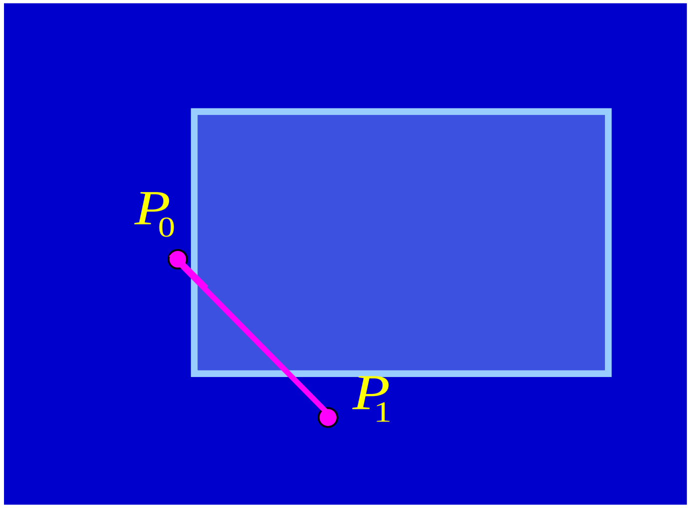


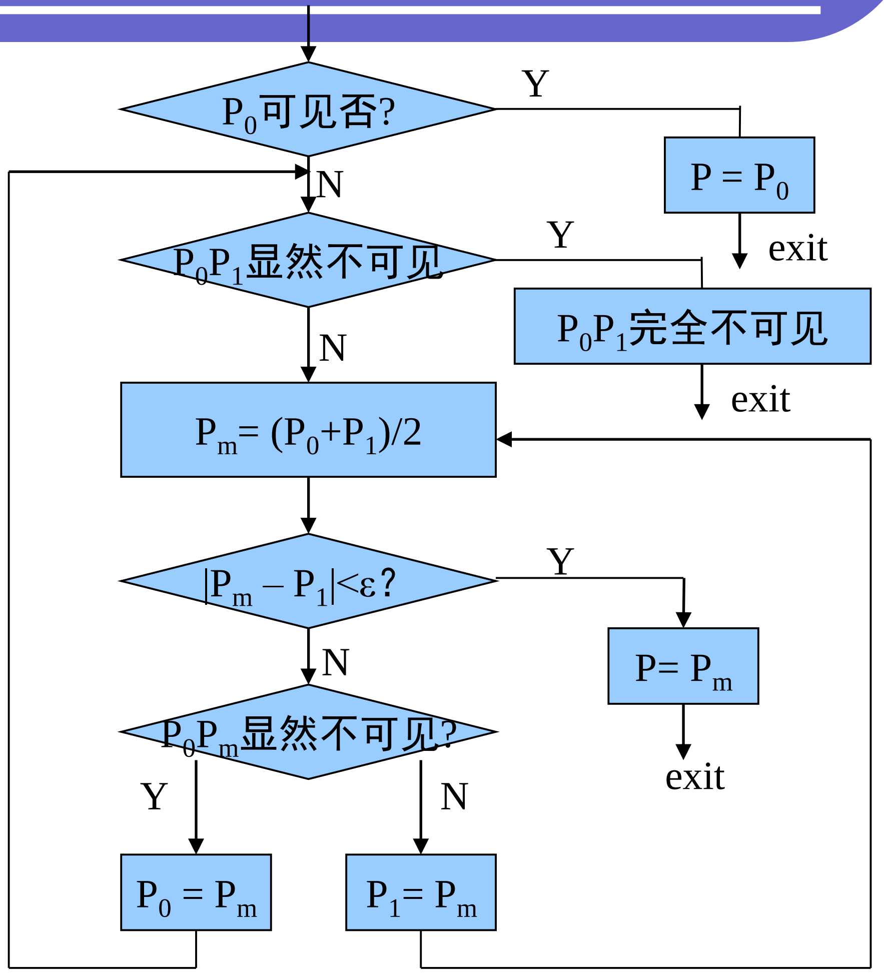


中点分割算法框图

中点分割算法

从P_0出发找出离P_0最近的可见点P

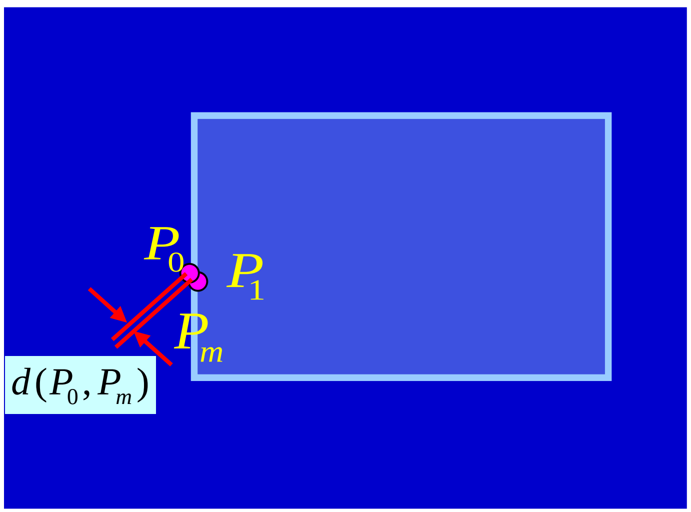


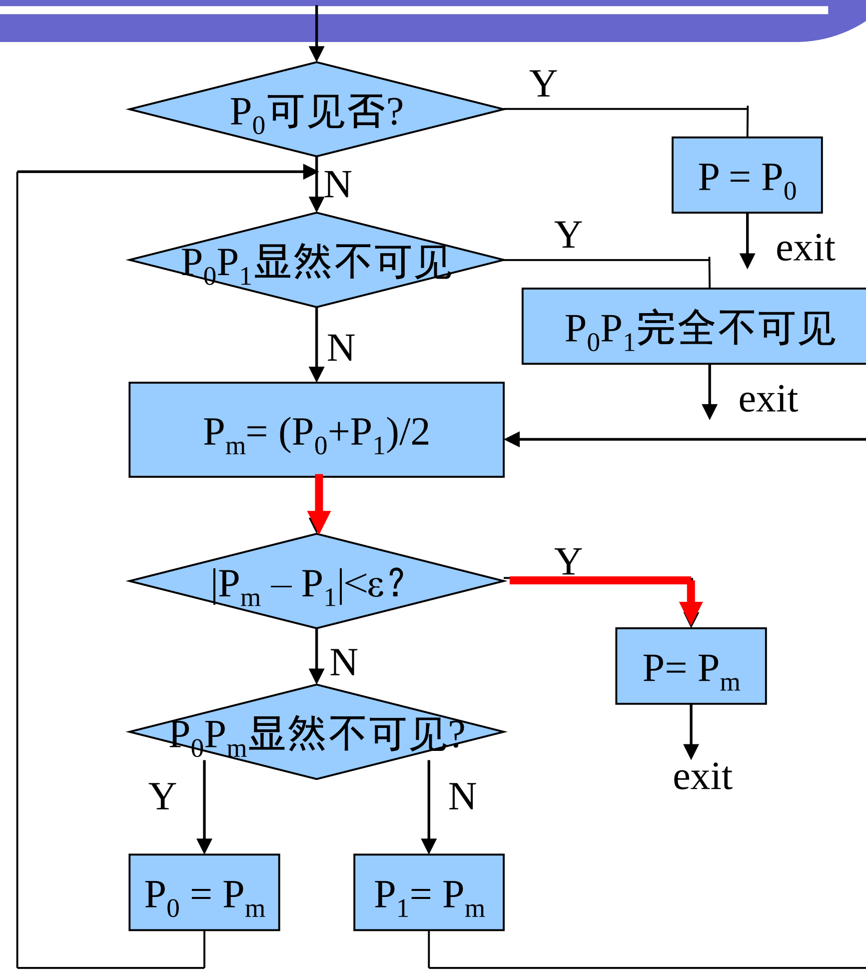


中点分割算法框图

中点分割算法

从P_0出发找出离P_0最近的可见点P

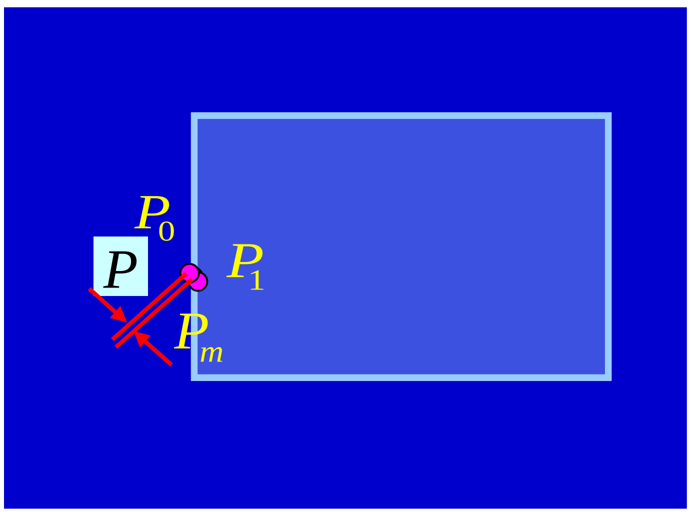


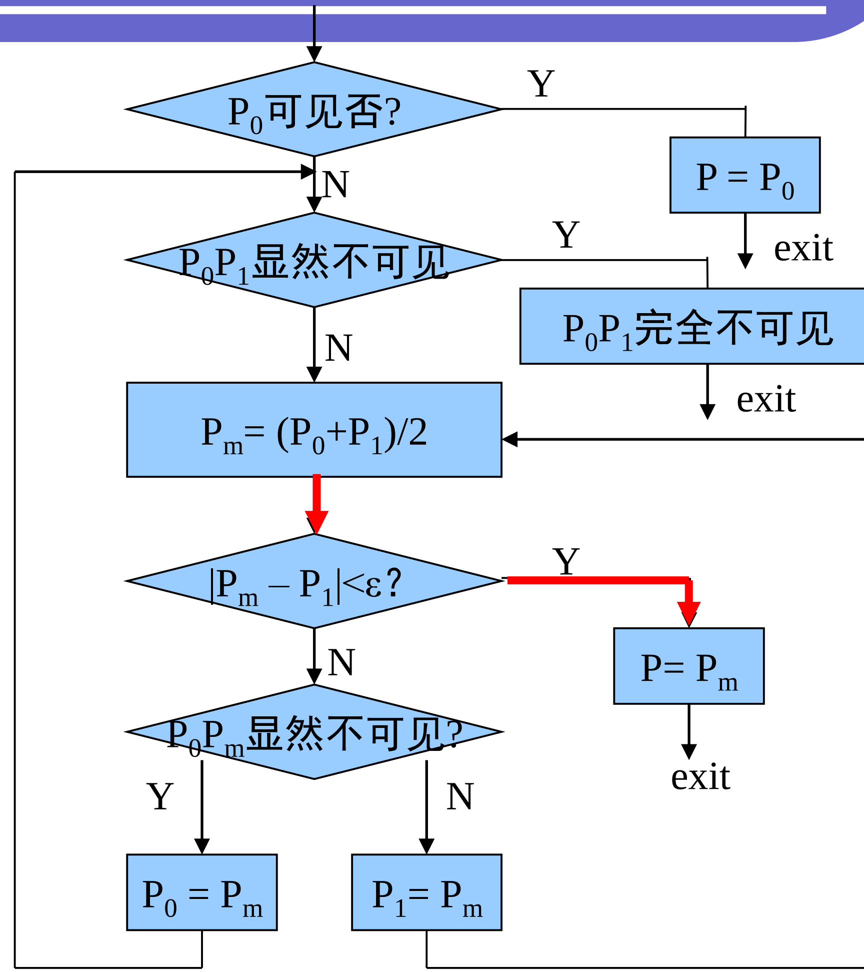


中点分割算法框图

中点分割算法

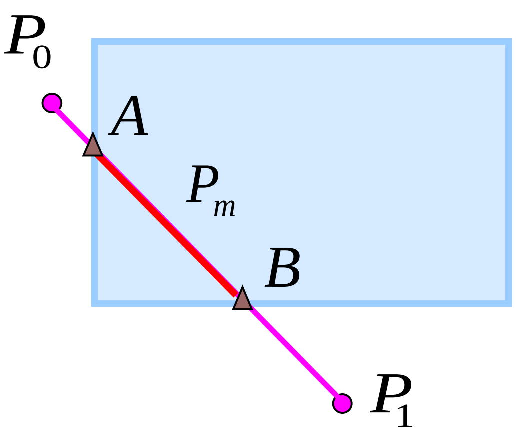
从P_0出发找出离P_0最近的可见点P





中点分割算法框图

中点分割算法

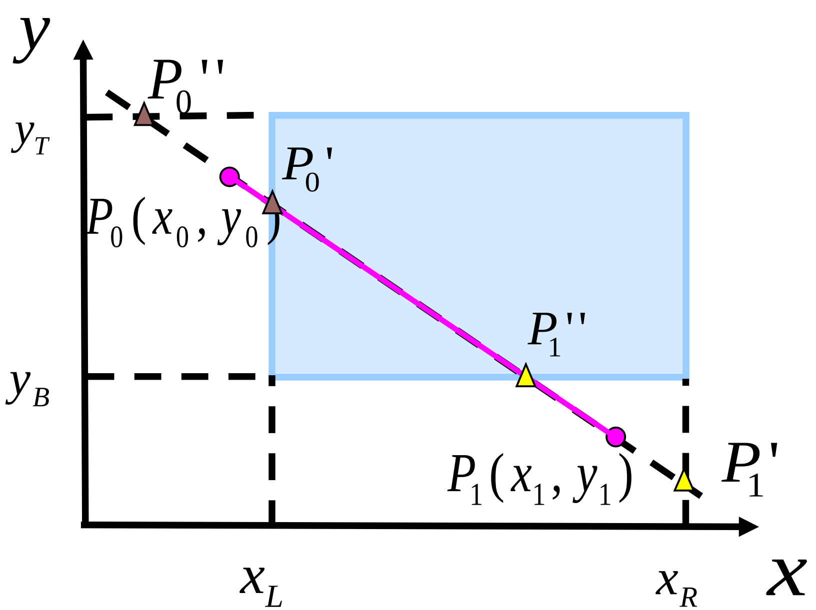


- 阈值 ε取为一个象素的宽度
  - 分辨率为$2^n \times 2^n$ 的显示器，本算法的二分过程最多为n次
- 主要过程只用到加法和除2运算，适合硬件实现
- 课后练习：结合端点编码，写出中点分割算法的伪代码

裁剪

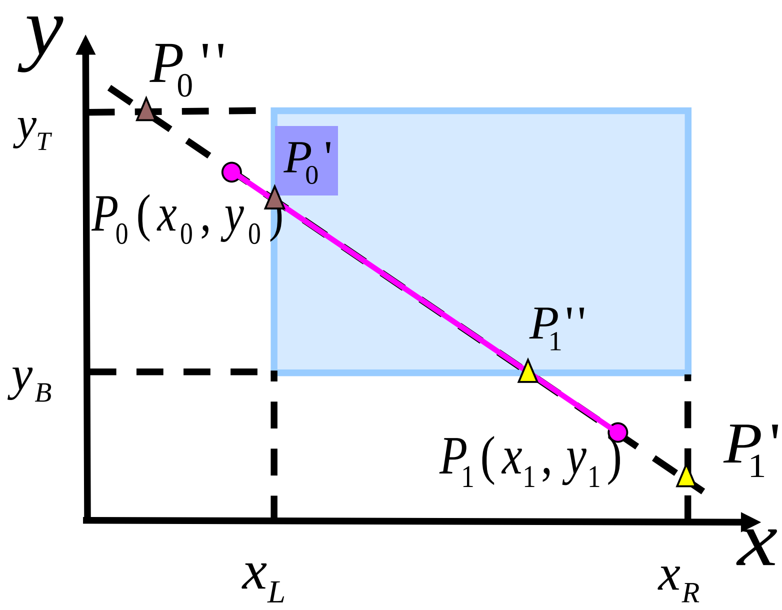
- 裁剪
- 二维线裁剪
  - Cohen-Sutherland 裁剪算法
  - 中点分割算法
  - 梁友栋-Barsky 裁剪算法
- 二维多边形裁剪
- 文本裁剪
- 三维裁剪
- 关于三维变换与裁剪

梁友栋-Barsky裁剪算法

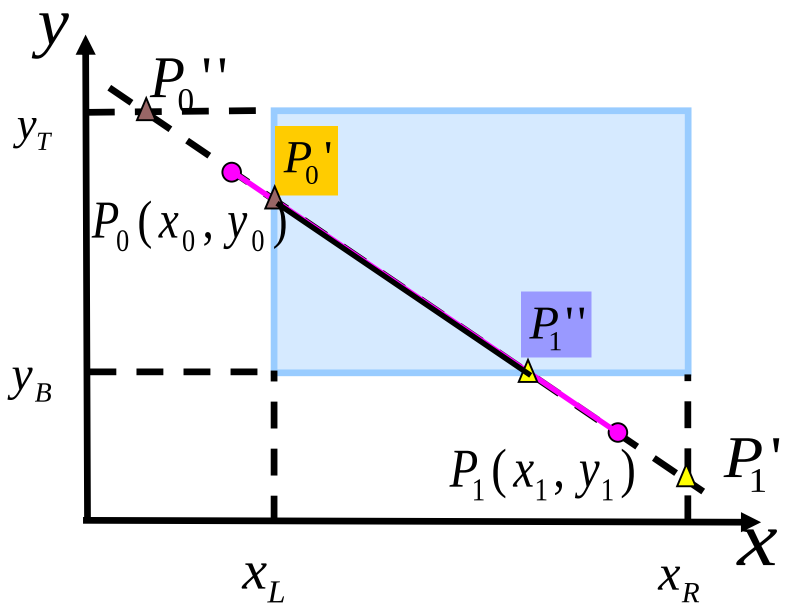


- Cyrus-Beck参数化裁剪算法：计算4个参数值，比较取舍
- Liang-Barsky裁剪：改进的参数化裁剪
  - 快速地拒绝与窗口不相交的线段
  - 每计算一个参数值，都会快速裁剪掉一部分线段

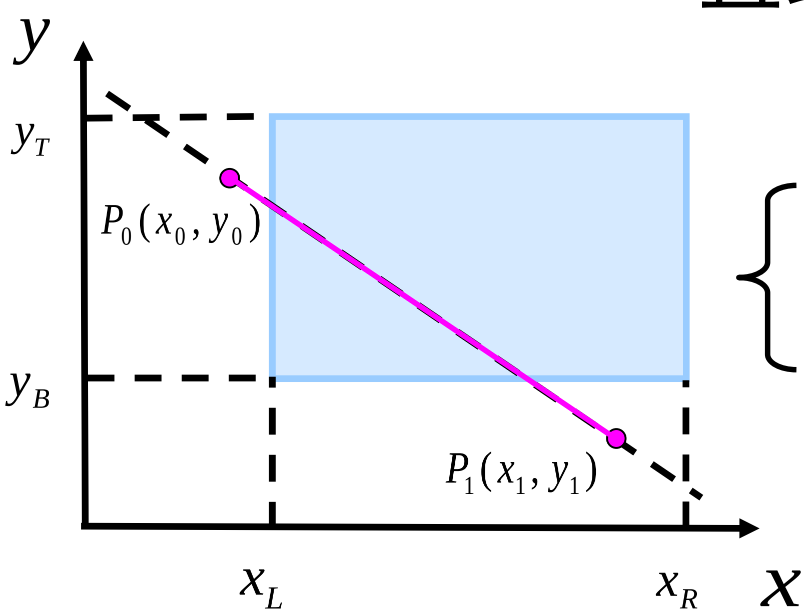
梁友栋-Barsky裁剪算法



梁友栋-Barsky裁剪算法



梁友栋-Barsky裁剪算法

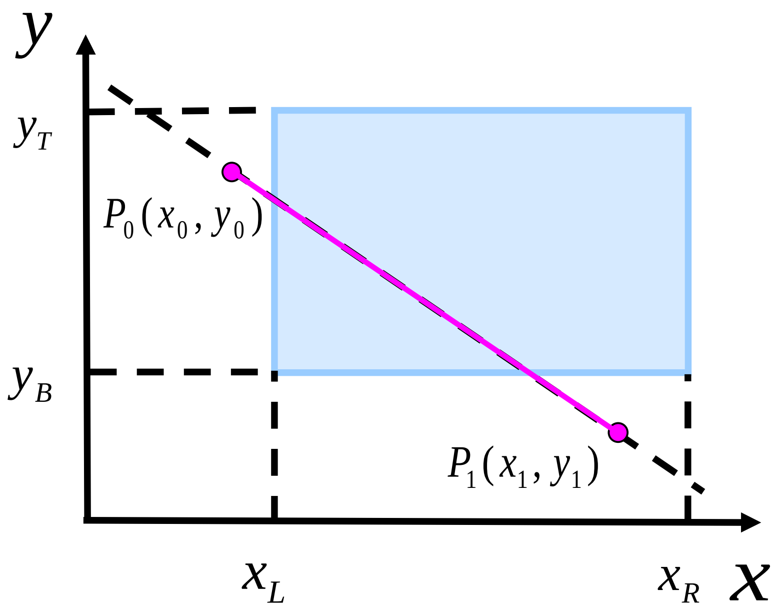


直线线段的参数化方程表示

$$\begin{cases} x = x_0 + t \cdot \Delta x & \Delta x = x_1 - x_0 \\ y = y_0 + t \cdot \Delta y & \Delta y = y_1 - y_0 \end{cases}$$

$$t \in [0,1]$$

梁友栋-Barsky裁剪算法

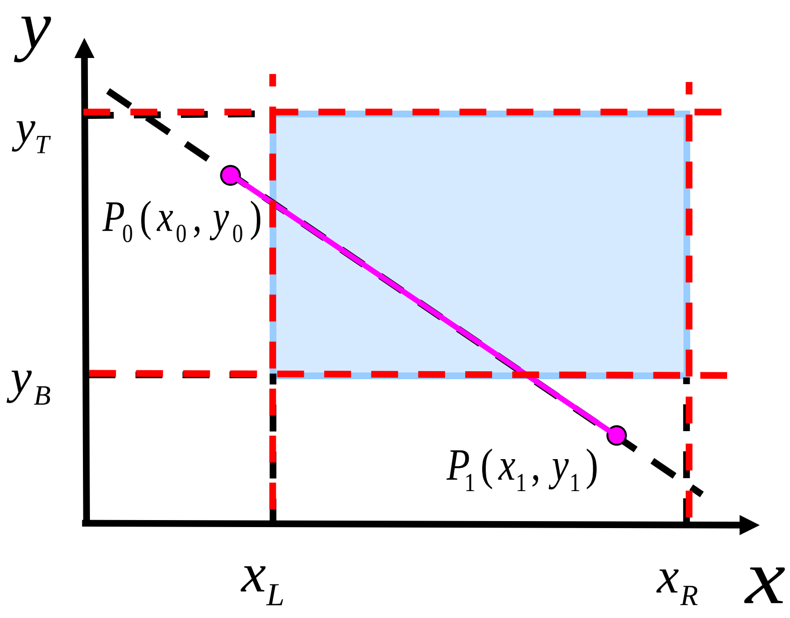


点裁剪条件的参数化形式

$$\begin{cases} x_L \le x_0 + t\Delta x \le x_R \\ y_B \le y_0 + t\Delta y \le y_T \end{cases}$$

$$\Rightarrow \quad tp_k \le q_k \quad k = L, R, B, T$$

梁友栋-Barsky裁剪算法

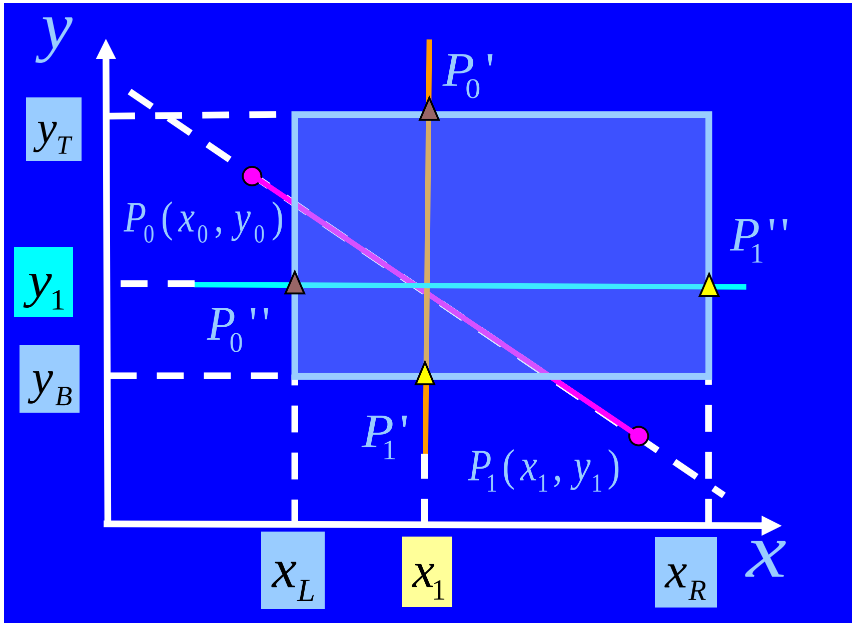


四个不等式表示的点裁剪条件

$$\boldsymbol{tp_k} \le \boldsymbol{q_k} \quad k = L, R, B, T$$

$$\begin{cases} -\Delta x t \le x_0 - x_L \\ \Delta x t \le x_R - x_0 \\ -\Delta y t \le y_0 - y_B \\ \Delta y t \le y_T - y_0 \end{cases}$$

梁友栋-Barsky裁剪算法



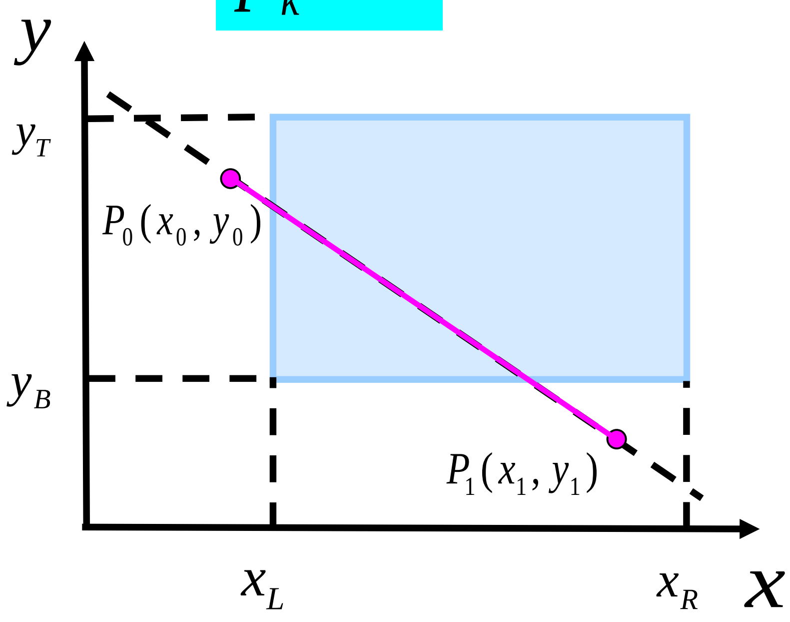
四个不等式表示的点裁剪条件

$$tp_k \le q_k \quad k = L, R, B, T$$

$$p_k = 0 \quad \begin{cases} q_k < 0 & \text{完全不可见} \\ q_k \ge 0 & \text{求交点} \end{cases}$$

梁友栋-Barsky裁剪算法

$$p_k \neq 0$$

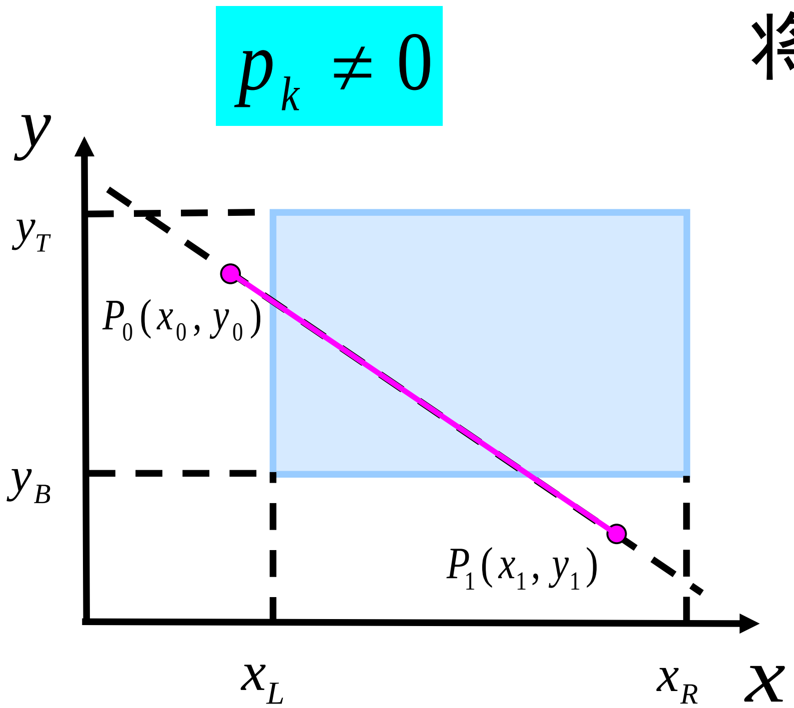


四个不等式表示的点裁剪条件

$$\boldsymbol{tp_k \le q_k} \quad k = L, R, B, T$$

$$\begin{cases} -\Delta x t \le x_0 - x_L \\ \Delta x t \le x_R - x_0 \\ -\Delta y t \le y_0 - y_B \\ \Delta y t \le y_T - y_0 \end{cases}$$

梁友栋-Barsky裁剪算法



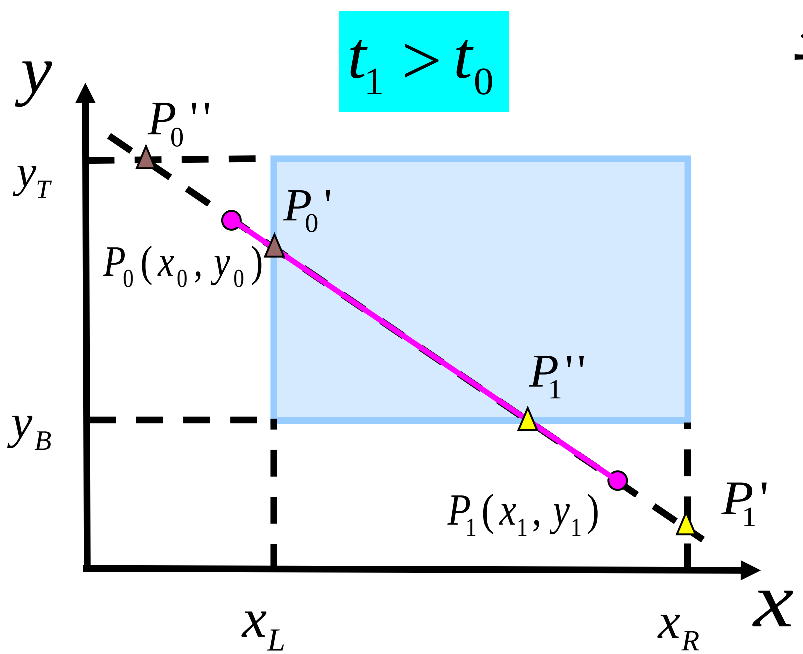
基于直线段参数方程的方法将窗口的边界分为

始边：线段从裁剪边界延长线的外部延伸到内部

$$
\begin{cases}
\text{当} \quad \Delta x \geq 0 \quad (\Delta y \geq 0) \\
\qquad x = x_L \quad (y = y_B) \\
\text{当} \quad \Delta x < 0 \quad (\Delta y < 0) \\
\qquad x = x_R \quad (y = y_T)
\end{cases}
$$

终边：其它边

梁友栋-Barsky裁剪算法



计算 $t_k = q_k / p_k$

$k = L, R, B, T$

$t_0 = \max(t_0', t_0'', 0)$ 0与两个始边交点的t值

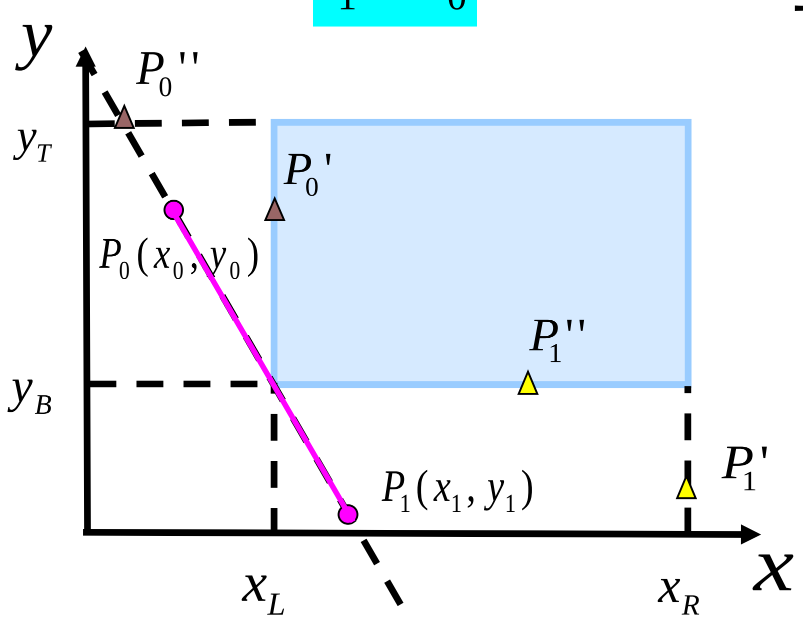
$t_1 = \min(t_1', t_1'', 1)$ 1与两个终边交点的t值

如果 $t_1 > t_0$ $t \in [t_0, t_1]$ 部分可见

否则 完全不可见

梁友栋-Barsky裁剪算法

$t_1 = t_0$



计算 $t_k = q_k / p_k$

$k = L, R, B, T$

$t_0 = \max(t_0', t_0'', 0)$ 0与两个始边交点的t值

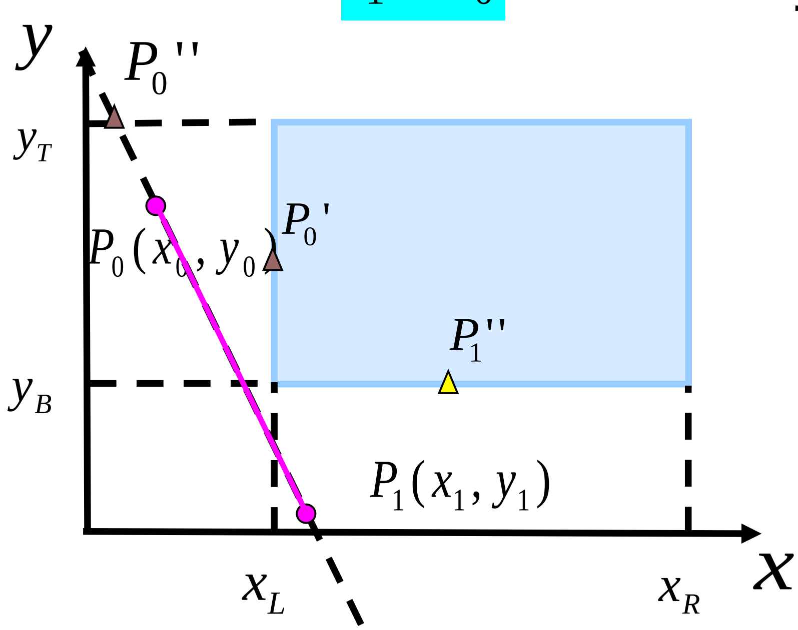
$t_1 = \min(t_1', t_1'', 1)$ 1与两个终边交点的t值

如果 $t_1 > t_0$ $t \in [t_0, t_1]$ 部分可见

否则 完全不可见

梁友栋-Barsky裁剪算法

$t_1 < t_0$



计算 $t_k = q_k / p_k$

$k = L, R, B, T$

$t_0 = \max(t_0', t_0'', 0)$ 0与两个始边交点的t值

$t_1 = \min(t_1', t_1'', 1)$ 1与两个终边交点的t值

如果 $t_1 > t_0$ $t \in [t_0, t_1]$ 部分可见

否则 完全不可见

梁友栋-Barsky裁剪算法

- 课后阅读： Cyrus-Beck参数化裁剪算法，体会梁友栋-Basky裁剪算法的改进之处在哪里？

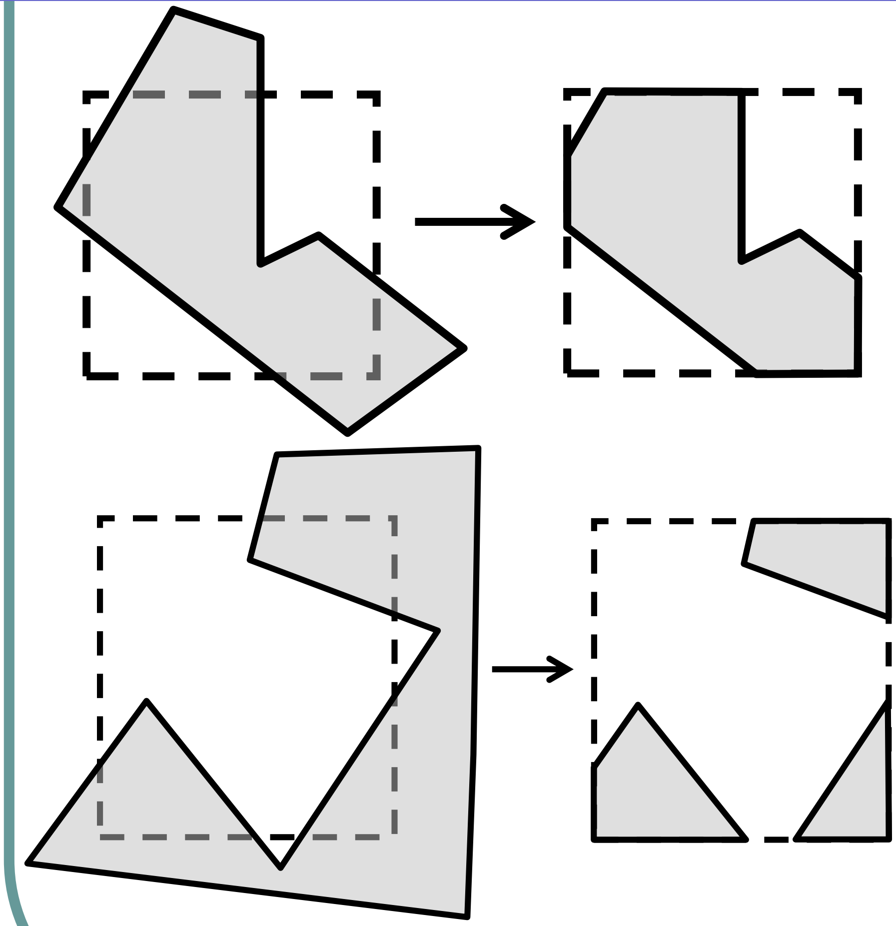
裁剪

- 裁剪
- 二维线裁剪
- 二维多边形裁剪
- 文本裁剪
- 三维裁剪
- 关于三维变换与裁剪

二维多边形裁剪

- 简单的处理方法：对多边形的每条线段采用线裁剪算法
  - 适用于线框图显示
  - 不适用于多边形的着色显示
- 正确的处理方法：裁剪后的多边形仍为封闭多边形
  - 可能会并入一部分窗口作为多边形边界
  - 也可能是多个不相连的多边形

二维多边形裁剪实例



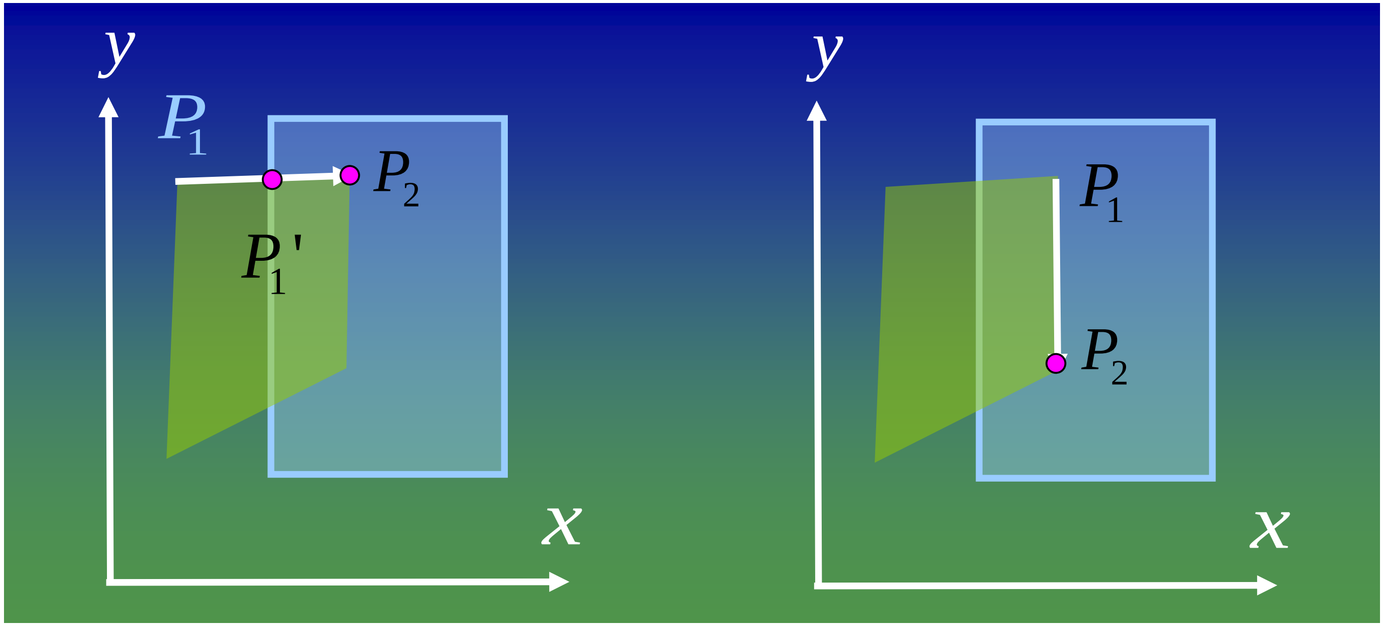
矩形窗口多边形裁剪实例

- 多边形裁剪后的输出应该是定义裁剪后的多边形边界的顶点序列
  - 如何保证裁剪后区域的封闭性
  - 如何确定裁剪后区域的边界

Sutherland-Hodgman算法

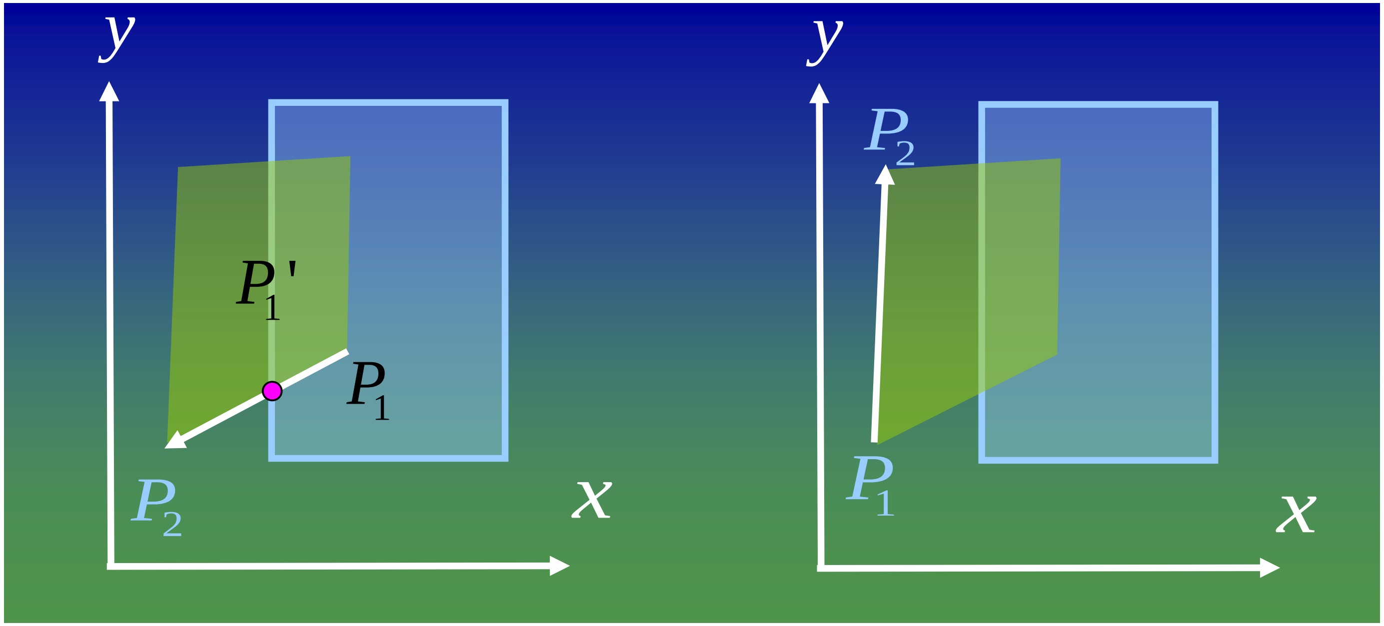
- 基本思想
  - 对多边形用窗口的四条边依次裁剪便得到裁剪后的多边形
  - 用窗口的一条边界处理完多边形的所有顶点后，其输出顶点表将用窗口的下一条边界继续裁剪。

Sutherland-Hodgman算法



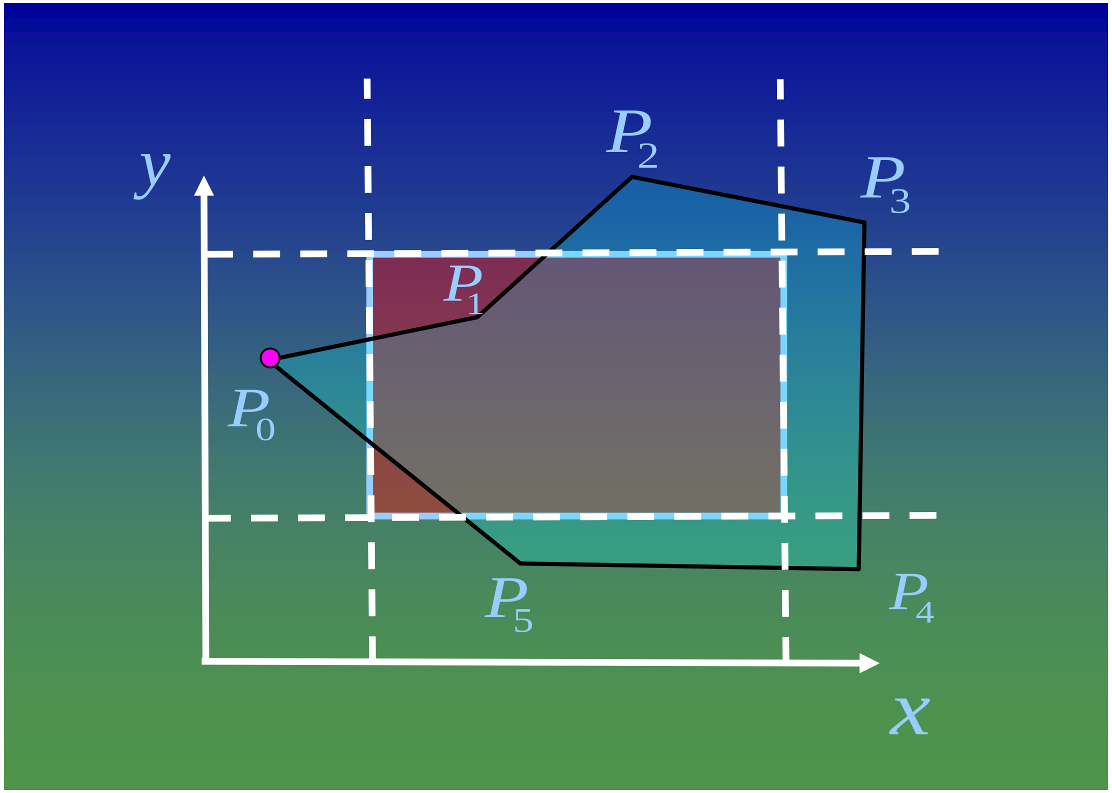
由外到内，保存P_1'、P_2　　　　由内到内，保存P_2

Sutherland-Hodgman算法

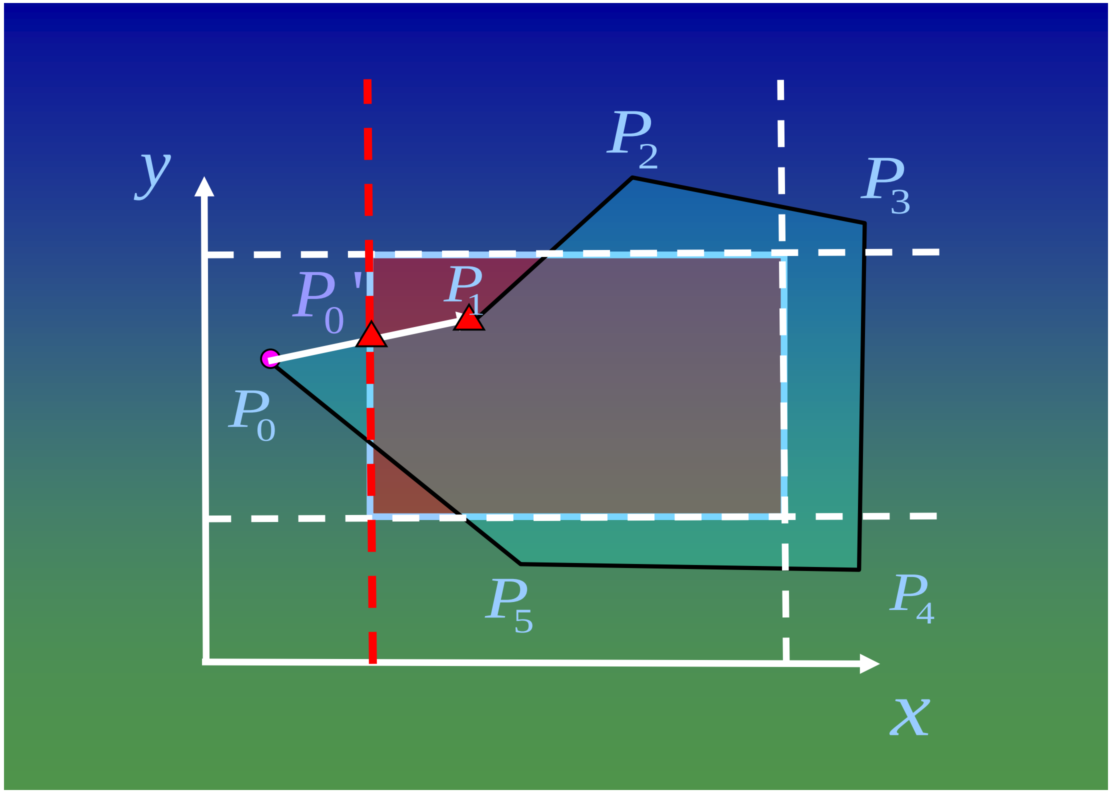


由内到外，保存P_1'　　　　由外到内，不保存

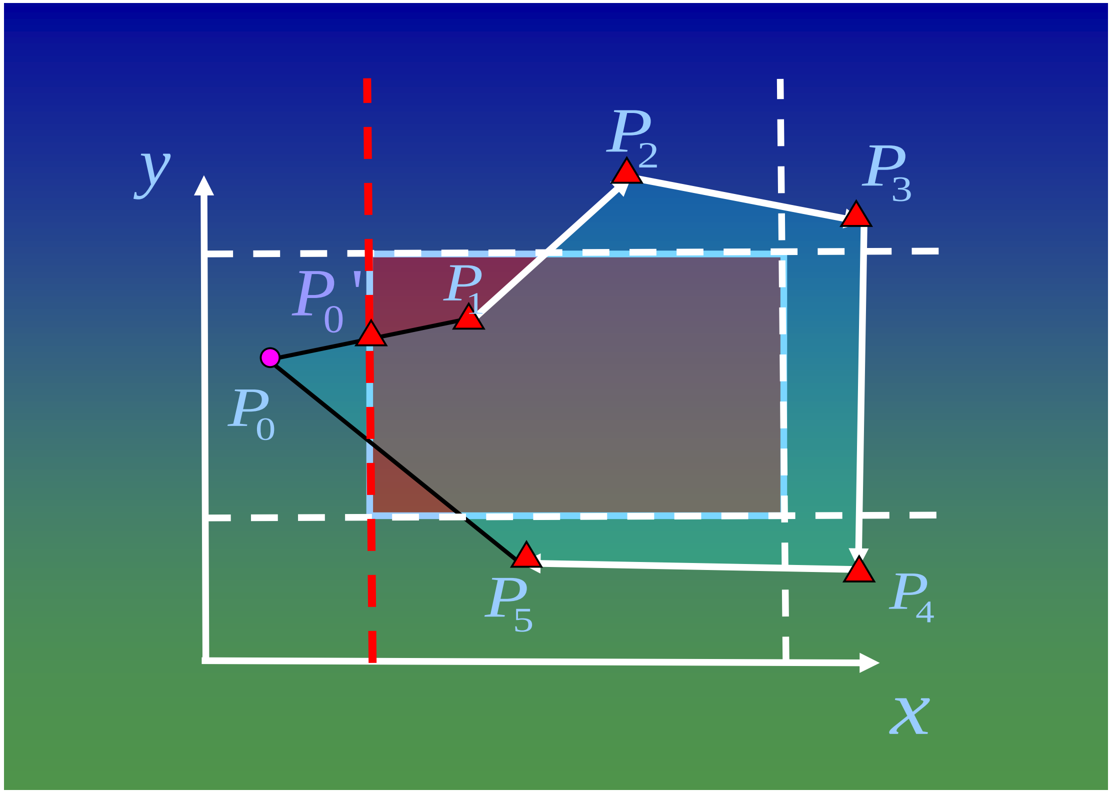
Sutherland-Hodgman算法



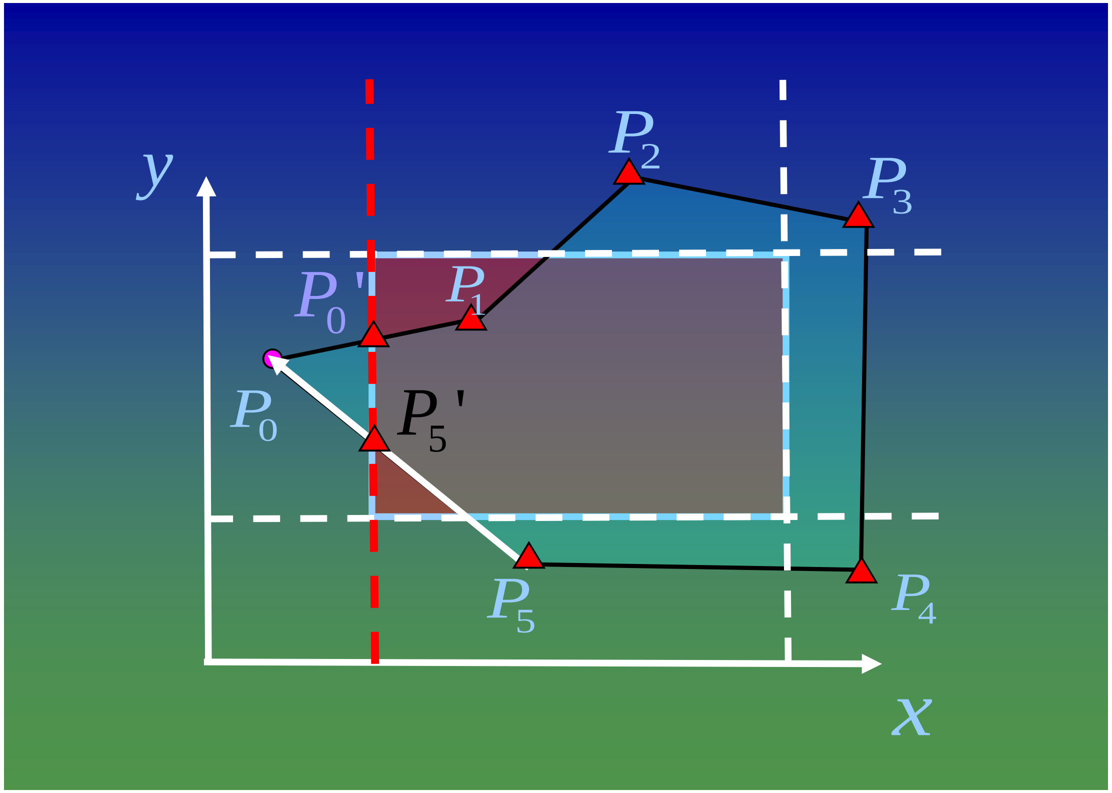
Sutherland-Hodgman算法



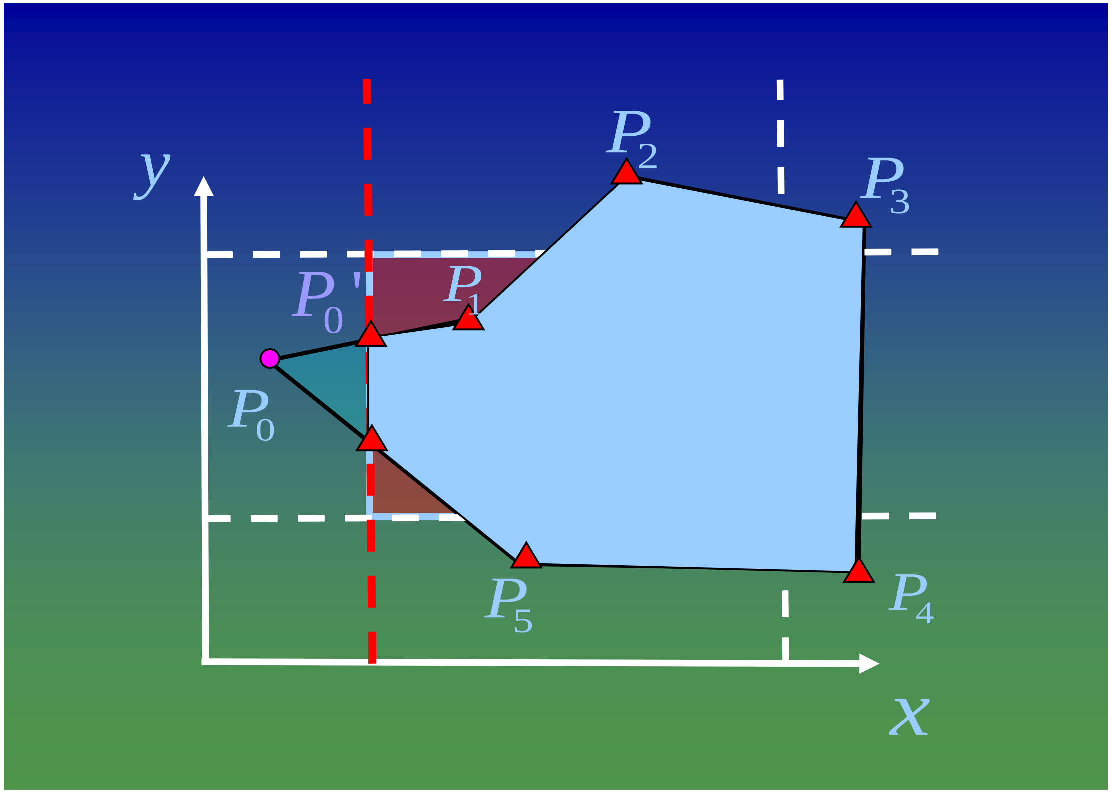
Sutherland-Hodgman算法



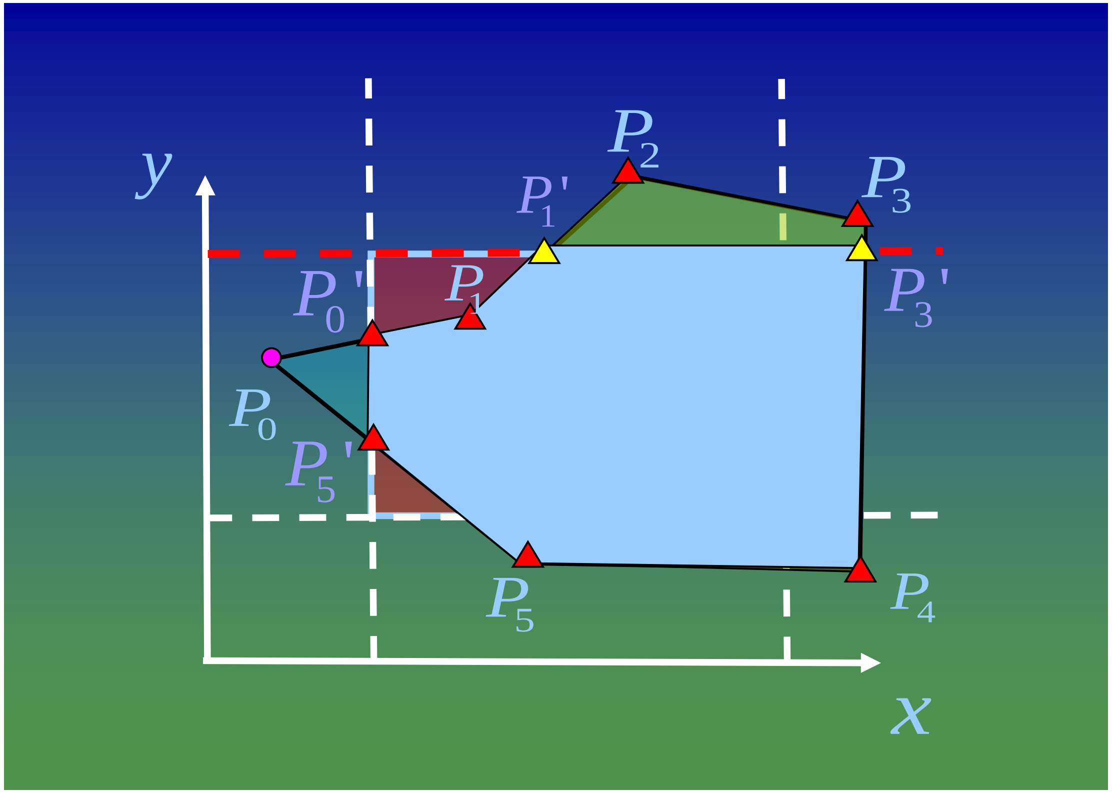
Sutherland-Hodgman算法



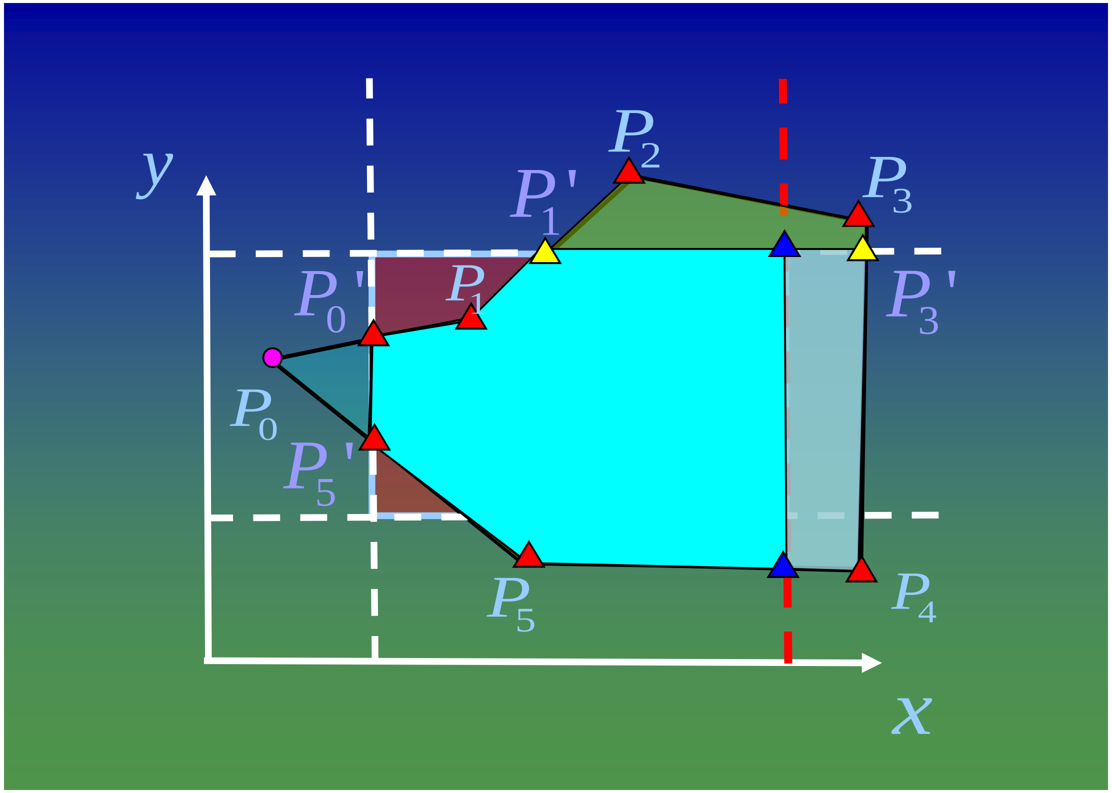
Sutherland-Hodgman算法



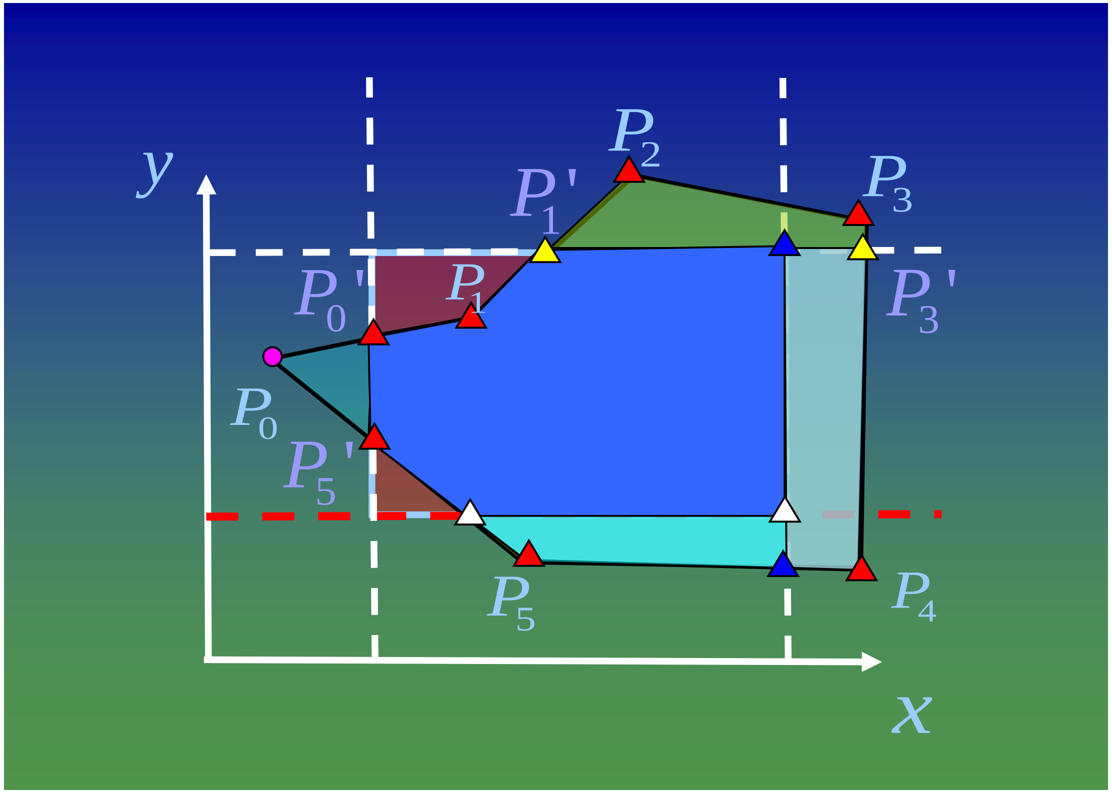
Sutherland-Hodgman算法



Sutherland-Hodgman算法

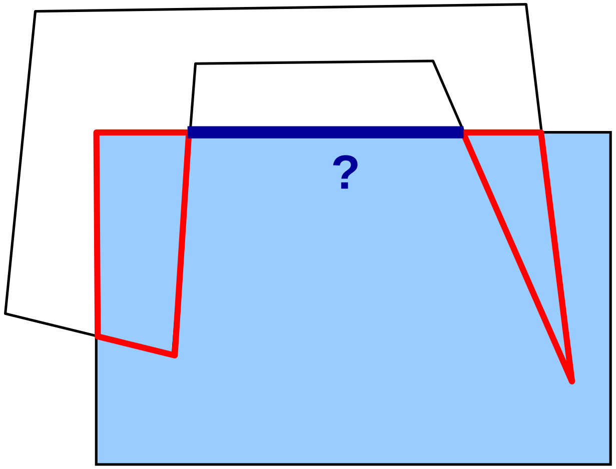


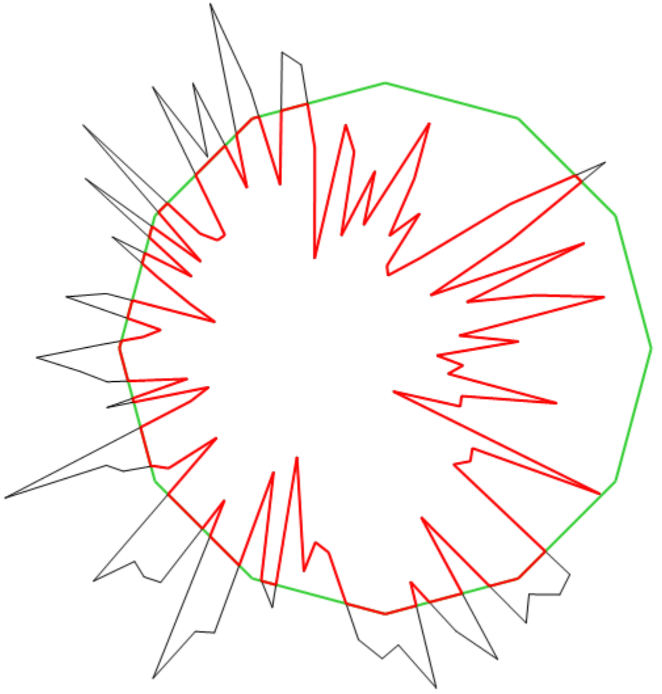
Sutherland-Hodgman算法



Sutherland-Hodgman算法

- 课后阅读：Sutherland-Hodgman算法
  - 完整算法
  - 如何消除多余的边?
  - 裁剪窗口为任意多边形?





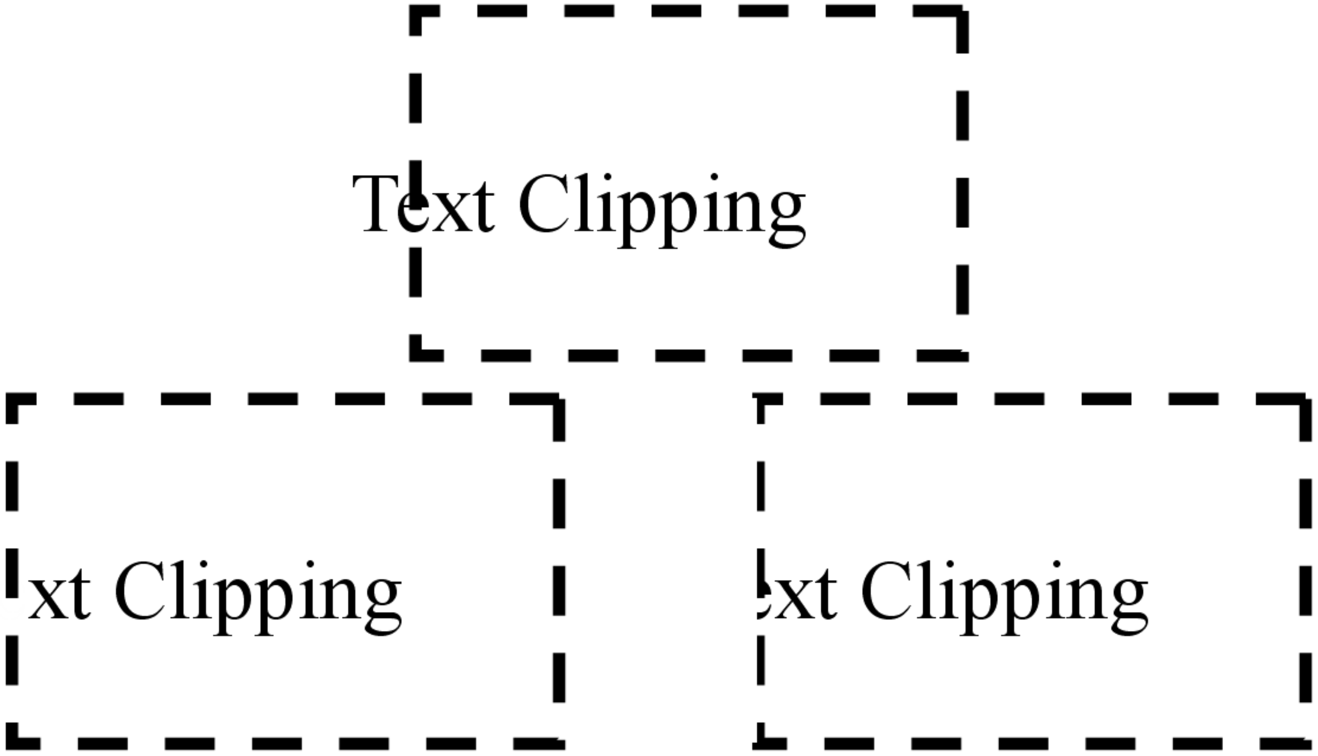
裁剪

- 裁剪
- 二维线裁剪
- 二维多边形裁剪
- 文本裁剪
- 三维裁剪
- 关于三维变换与裁剪

文本裁剪

- 矢量文本裁剪：采用前面的多边形裁剪算法实现文本的裁剪
- 点阵文本裁剪：
  - 如果点阵是由软件生成的，点阵式文本的裁剪可以归结为点的裁剪问题；
  - 如果点阵式文本是由硬件生成的，裁剪就会变得比较复杂，一个简单的处理方法是：如果字符完全位于裁剪窗口内才会显示

文本裁剪



文本裁剪

裁剪

- 裁剪
- 二维线裁剪
- 二维多边形裁剪
- 文本裁剪
- 三维裁剪
- 关于三维变换与裁剪

三维裁剪

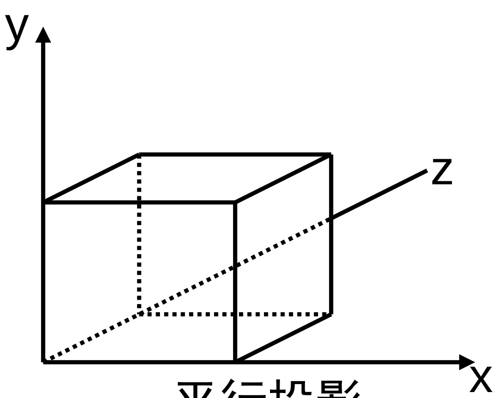
- 三维裁剪
  - 裁剪对象：线裁剪、面裁剪
  - 裁剪窗口：规范的立方体、视域四棱锥
- Cohen-Sutherland、梁友栋-Basky裁剪、Sutherland-Hodgman算法都可以推广到三维情形

三维裁剪窗口的规范化

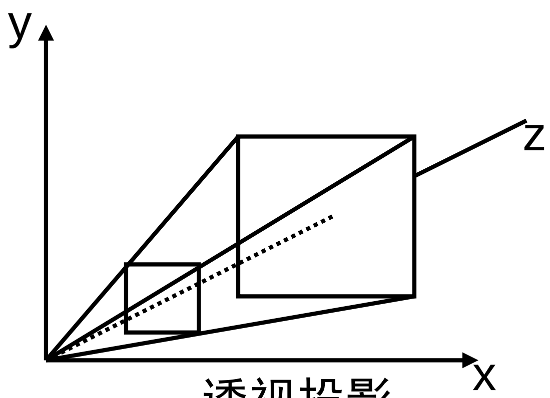
- 为什么引入规范视域体
  - 简化投影
  - 简化裁剪
- 规范化变换
  - 将任意视域体变换成规范视域体的变换

三维裁剪窗口的规范化

- 平行投影：x=0, x=1, y=0, y=1, z=0, z=1 or
  x=-1, x=1, y=-1, y=1, z=0, z=1
- 透视投影：x=z, x=-z, y=z, y=-z, z=z_{min}, z=1



平行投影



透视投影

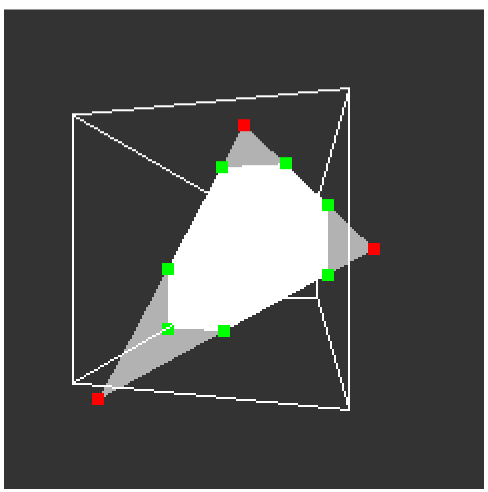
两种规范化的视域体

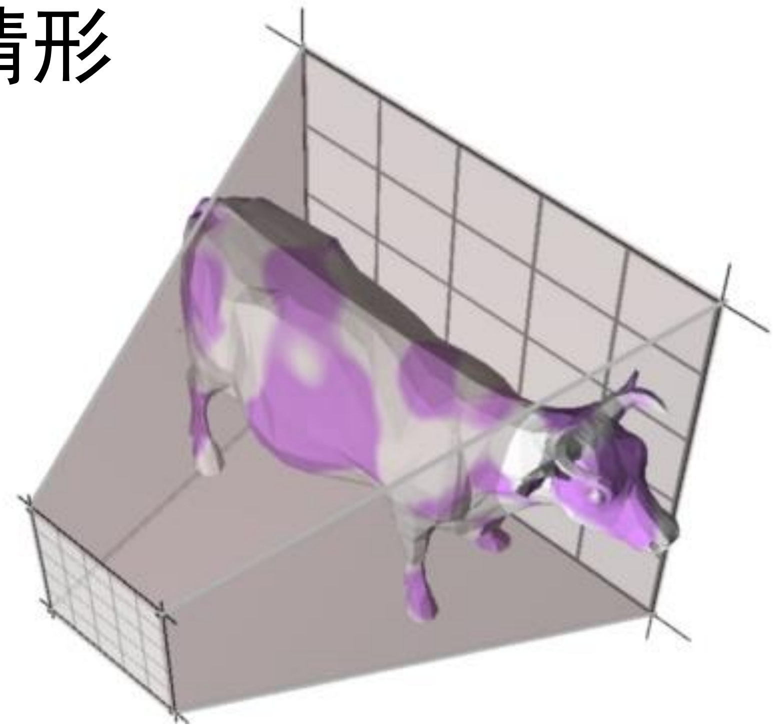
四维齐次坐标空间裁剪

- 优点：
  - 不需要将齐次坐标转换为三维坐标
  - 有理曲线曲面可能直接用齐次坐标来表示，对它们的裁剪只能在齐次坐标空间中进行
- 缺点：四维裁剪更复杂

三维裁剪

- 课后阅读：Cohen-Sutherland、梁友栋-Basky裁剪、 Sutherland-Hodgman算法都可以推广到三维情形





裁剪

- 裁剪
- 二维线裁剪
- 二维多边形裁剪
- 文本裁剪
- 三维裁剪
- 关于三维变换与裁剪

关于三维变换与裁剪

- 何时裁剪?
  - 投影之前裁剪——三维裁剪
    - 优点：只对可见的物体进行投影，提高消隐效率
    - 缺点：三维裁剪相对复杂
  - 投影之后裁剪——二维裁剪
    - 优点：二维裁剪相对容易
    - 缺点：需要对所有的物体进行投影变换

更为一般的三维显示过程

- 课后阅读：《计算机图形学教程》(修订版)唐荣锡等， 科学出版社，2000年。pp74-82