山东大学 计算机科学与技术 学院

计算机网络 课程实验报告

学号:201700130011	姓名：刘建东	班级：17 级菁英班

实验题目：Liang-Barsky 裁剪算法

实验内容：

1. 实现 Liang-Barsky 裁剪算法

实验过程：

一、画线程序：

由于 Liang-Barsky 算法是矩形窗口对于直线段的裁剪，考虑到鼠标的交互性，需要添加画线程序。

因此调用了 STL 中的 vector 来保存所有直线，采用了第一次实验的 Bresenham 算法来画出直线。

```
struct BASE{
    int x0,y0,x1,y1;
};
vector<BASE> Vline,VRectangle,Vline1;
//Vline —— 保存已经画完的直线，Vline1 —— 保存当前画的直线，VRectangle —— 保存窗口坐标
```

二、画矩形程序：

因为此裁剪算法是用矩形窗口对直线进行裁剪，因此矩形的构造不可避免。此处采用了两点确定一个矩形的原则，用矩形四个角上的两个端点来确定这个矩形。

这里会出现一个小问题，就是四个角上的两个端点不一定是左下角和右下角，因此我们需要先对 x、y 坐标求一个最大和最小值，以此确定左下角和右下角，便于绘制这个图形。然后再调用 Bresenham 算法画线即可。

```
if(VRectangle.size()){
    int x0 = VRectangle[0].x0, y0 = VRectangle[0].y0, x1 = VRectangle[0].x1, y1 = VRectangle[0].y1;
    int xlow = min(x0,x1), ylow = min(y0,y1), xh = max(x0,x1), yh = max(y0,y1);
    DrawLine(xlow, ylow, xlow, yh);
    DrawLine(xlow, ylow, xh, ylow);
    DrawLine(xlow, yh, xh, yh);
    DrawLine(xh, ylow, xh, yh);
```

三、裁剪算法：

裁剪算法即实现了 Liang-Barsky 的算法原理。基本步骤如下。

(1)输入 (x1,y1)、(x2,y2)、wxl、wxr、wyb、wyt。

(2)若△x = 0，则 p1=p2=0，此时进一步判断 q1 < 0 || q2 < 0，则直线段不在窗口内，转⑦结束。否则，满足 q1 >= 0 && q2 >= 0，进一步计算 u_max 与 u_min。

(3)若△y=0，则 p3=p4=0，此时进一步判断是否满足 q3 < 0 || q4 < 0，若满足则该直线段不在窗口内，转⑦结束。否则，q3 >= 0 && q4 >= 0，进一步计算 u_max 与 u_min。

(4)若上述两条均不满足，则 pk != 0 (k = 1,2,3,4)，则计算 u_max 和 u_min。

(5) 求得 u_max 和 u_min 后，进行判断：若 u_max > u_min，则直线段在窗口外，转⑦。若 u_max <= u_min，则得到 x、y 坐标。

(6) 画出直线段。

(7) 算法结束。

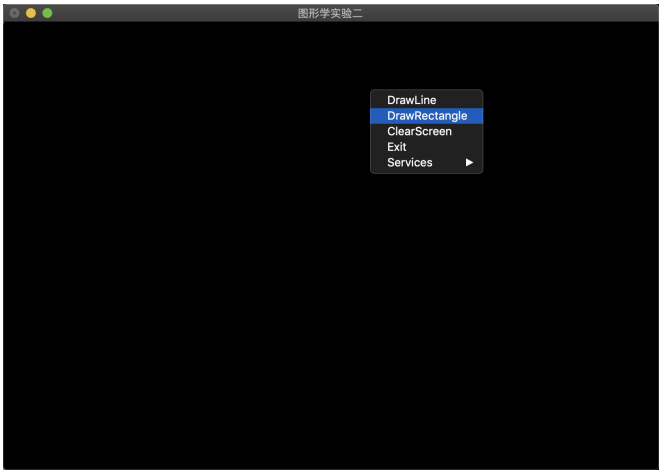
代码实现过程中，该函数的参数为直线的两点坐标，窗口的左右上下边界，然后在函数中直接判断此直线在窗外还是窗内，然后直接画出直线。

```
void LiangBarsky(int x1, int y1, int x2, int y2, int xleft, int xright, int ybottom, int ytop){
    int p[5] = {0,x1-x2,x2-x1,y1-y2,y2-y1},
        q[5] = {0,x1-xleft,xright-x1,y1-ybottom,ytop-y1}, L = 1, R = 4;
    db u[5], umin = 1, umax = 0;
    if(p[1] == 0 && p[3] == 0) return;
    if(p[1] == 0){
        if(q[1] < 0 || q[2] < 0) return;
        else {L = 3; R = 4;}
    }
    else if(p[3] == 0){
        if(q[3] < 0 || q[4] < 0) return;
        else {L = 1; R = 2;}
    }
    for(int i = L; i <= R; i++){
        u[i] = (db)q[i]/(db)p[i];
        if(p[i] < 0) umax = max(umax,u[i]);
        else umin = min(umin,u[i]);
    }
    if(sign(umax-umin) == 1) return;
    db xans1 = (db)x1+umin*(db)(x2-x1), yans1 = (db)y1+umin*(db)(y2-y1), xans2 = (db)x1+umax*(db)(x2-x1), yans2 = (db)y1+umax*(db)(y2-y1)
    DrawLine((int)xans1, (int)yans1, (int)xans2, (int)yans2);
}
```

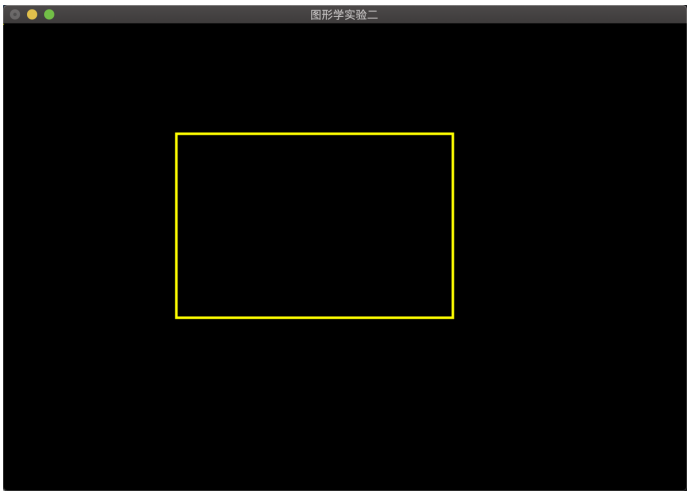
四、功能演示：

(1) 确定窗口：

选择画矩形按钮确定窗口。如果直接画线会默认无窗口。

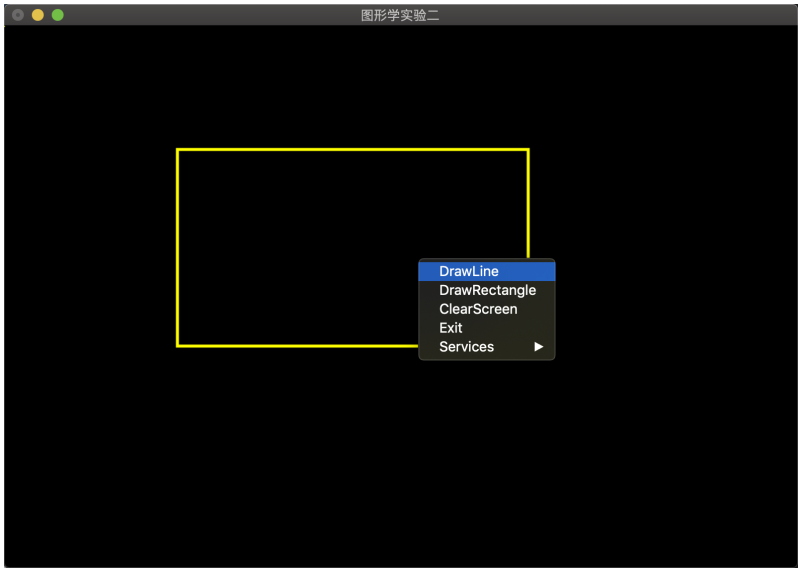


鼠标拖拽即可确定窗口矩形。

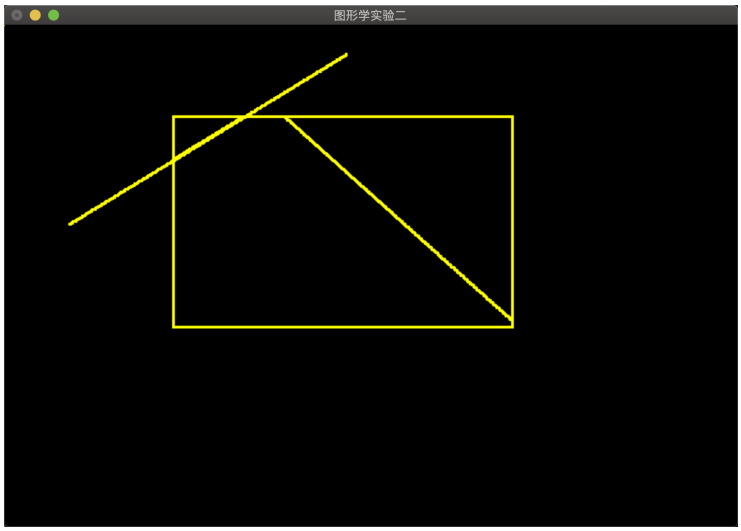


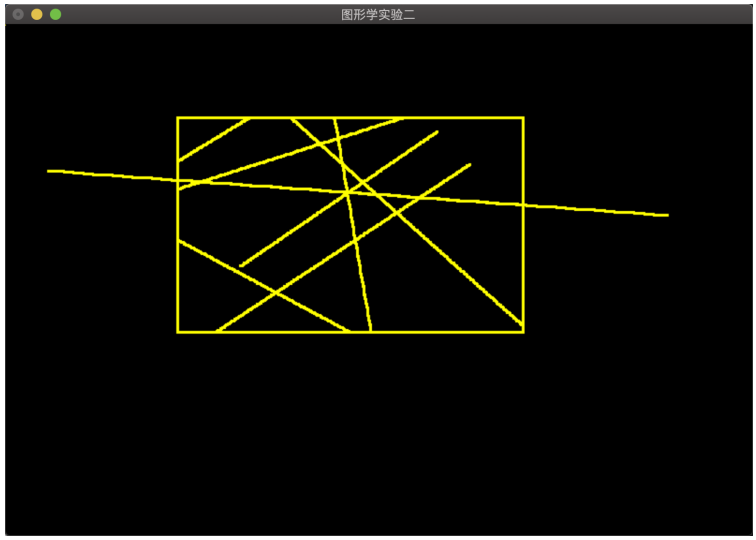
(2) 画线：

选择画线按钮进行画线裁剪。



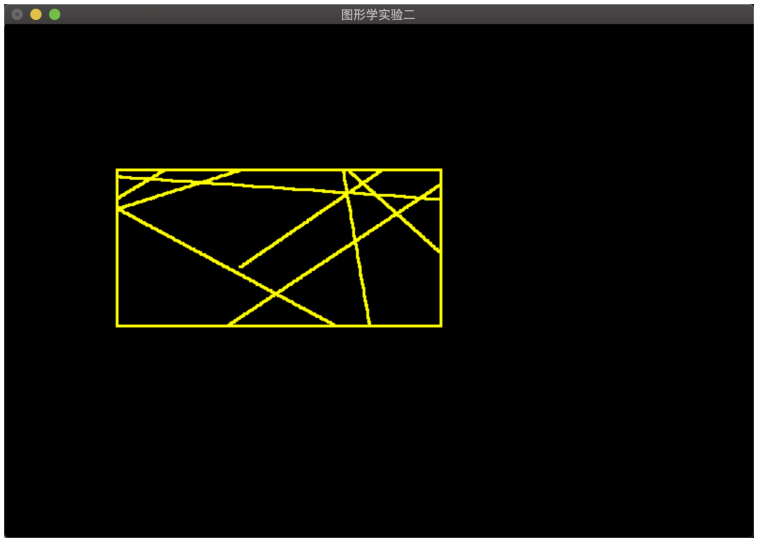
鼠标拖拽即可画线，刚画好的直线不会直接裁剪，当松开鼠标画下一条直线时才会认为上一条直线已经画完，才会开始裁剪。

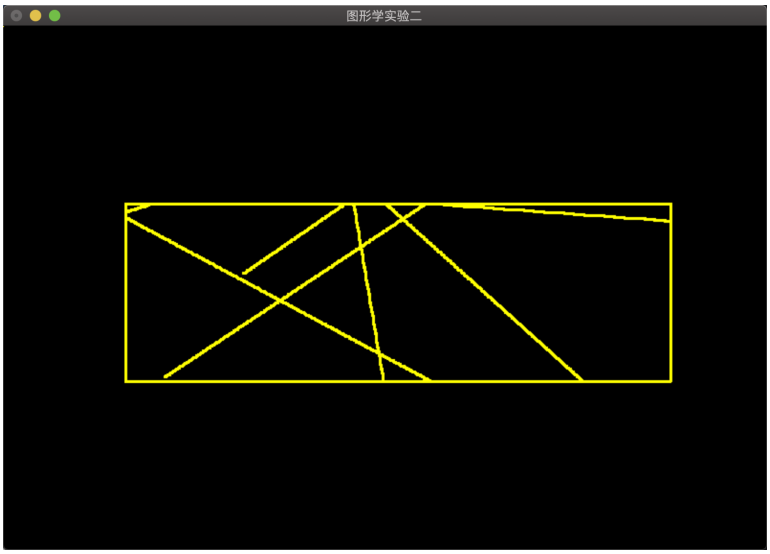


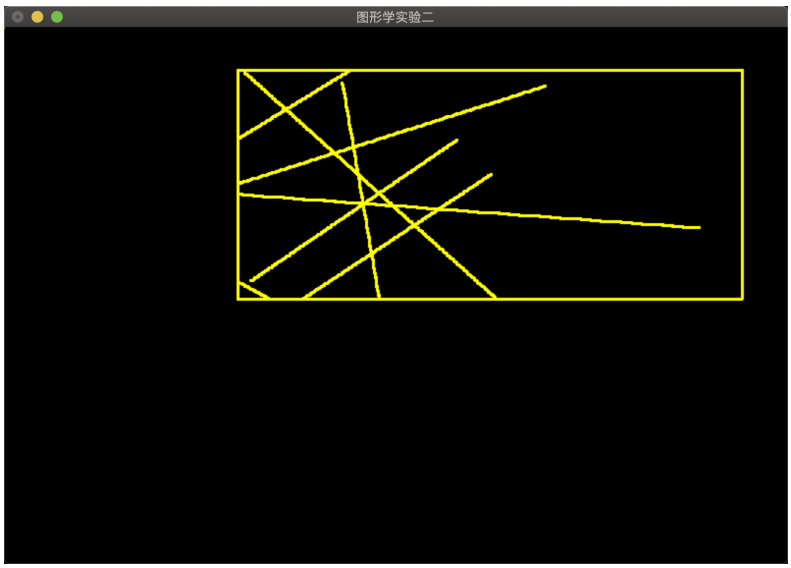


(3)在画矩形部分重新确定窗口：

由于画出的直线都会被保存，因此可以重新确定窗口，每条直线都会被再次裁剪后输出。拖拽鼠标画出窗口即可看到之前每条直线被裁剪之后显示的图形。







实验总结：

1. 实验中为了将所有直线都保存下来，使用了一个 vector 来存储所有的直线，以此实现拖拽矩形窗口时可以看到直线的裁剪状态。

```
struct BASE{
    int x0,y0,x1,y1;
};
vector<BASE> Vline,VRectangle,Vline1;
//Vline —— 保存已经画完的直线，Vline1 —— 保存当前画的直线，VRectangle —— 保存窗口坐标
```

2. 回调函数中每次刷新屏幕，每次重新画线、画矩形。在矩形窗口存在的时候才会画线，矩形窗口不存在的时候不会显示直线。

3. 由于 Liang-Barsky 算法中涉及到了浮点数比较，因此手写了比较函数，将精度误差限制在了 1e-7。

```
typedef double db;
const db EPS = 1e-7;
inline int sign(db a) {return a < -EPS ? -1 : a > EPS; } //返回-1表示a < 0, 1表示a > 0, 0表示a = 0
```