软件设计文档

课程：图形学

任课老师：孙正兴

姓名：金鑫

学号：121220307

CONTENT

1.1 编写目的

对软件进行模块级别的详细设计说明，以便编写代码人员参考，也方便维护人员在软件维护过程中的维护工作。

本文档的阅读对象为软件的开发和使用人员。

1.2 背景

项目名称：Drawing Board

项目的提出：孙正兴老师

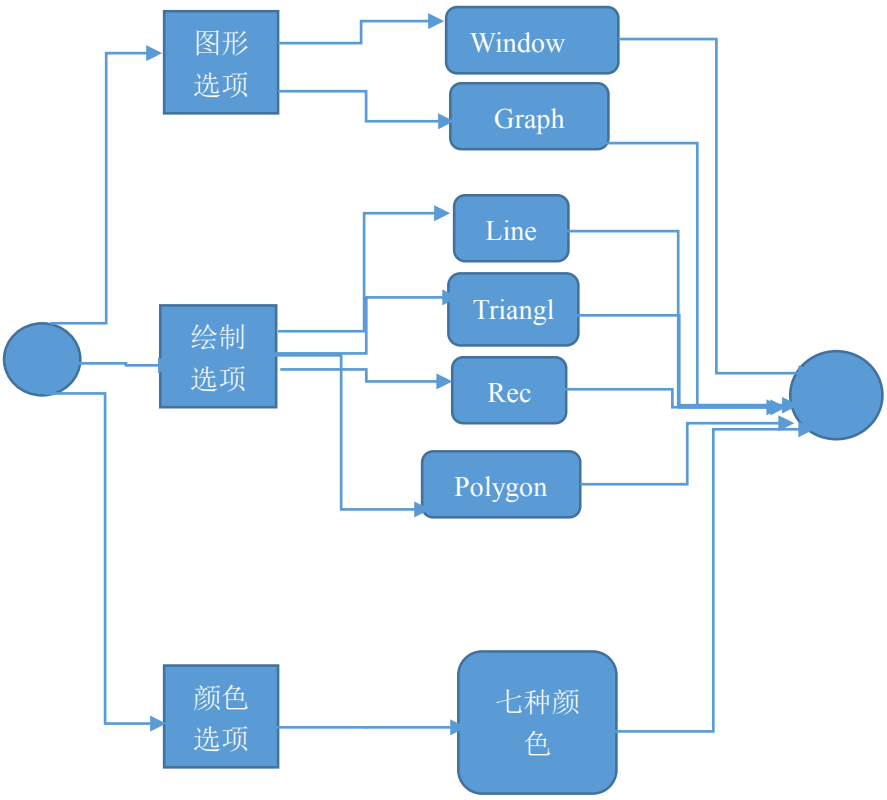
项目的开发者：金鑫

1.3 定义

A. 系统：Drawing Board

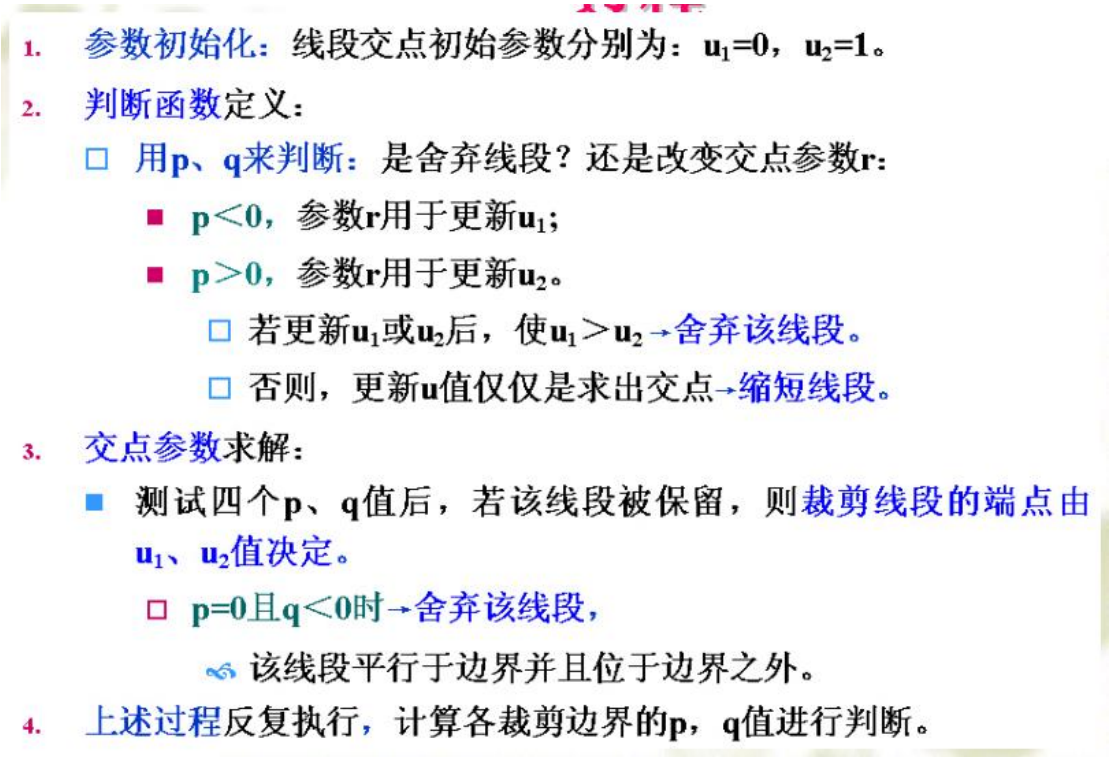
B. 用户：使用该系统的用户

2. 程序系统的结构



3. 算法描述

3.1 Liang-Barsky 裁剪算法



3.2 Sutherland-Hodgman 多边形裁剪算法

算法介绍[1]：

Sutherland-Hodgman[suth74b]算法的基本思想是，多边形对一条边或一个面的裁剪容易实现，故用窗口的边，一条一条地对原多边形和中间结果多边形进行裁剪。图3-31显示了一个矩形窗口裁剪多边形的过程。原多边形由一系列顶点$P_1,\cdots,P_n$所定义，它的边为$P_1P_2,P_2P_3,\cdots,P_{n-1}P_n,P_nP_1$。

这些边首先被窗口左边裁剪，生成一个中间多边形，如图3-31所示。然后重新调用裁剪程序，将中间多边形对窗口顶边进行裁剪，生成第二个中间多边形。继续这一过程，直至多边形被窗口的所有边都裁剪过为止。裁剪的每一步均显示在图3-31中。注意在最终结果多边形中，角点Q_8的加入并非难事。这一算法可以在任何凸多边形窗口内对任何凸或凹的、平面或非平面的多边形进行裁剪。窗口各边以什么顺序裁剪多边形是无关紧要的。

算法的输出结果是一个多边形的顶点列表，均位于裁剪面的可见一侧。由于多边形的每一条边都是单独地与裁剪面进行比较，因此只需考虑单条边和单个裁剪面之间的位置关系。取多边形顶点列表中的一点P(第一点除外)作为一边的终点，列表中位于P前面的一点S作为该边的起点，则边SP同裁剪面之间只有四种可能的关系，如图3-32所示。

每次将多边形的边与裁剪面比较之后，向裁剪后的多边形顶点列表输出一个到两个顶点，或者不输出顶点。若边完全可见，则输出P点。注意不必再输出边的起点S。因为顶点表中的点是依序处理的，S作为前一条边的终点已经被输出。若边完全不可见，则没有输出。

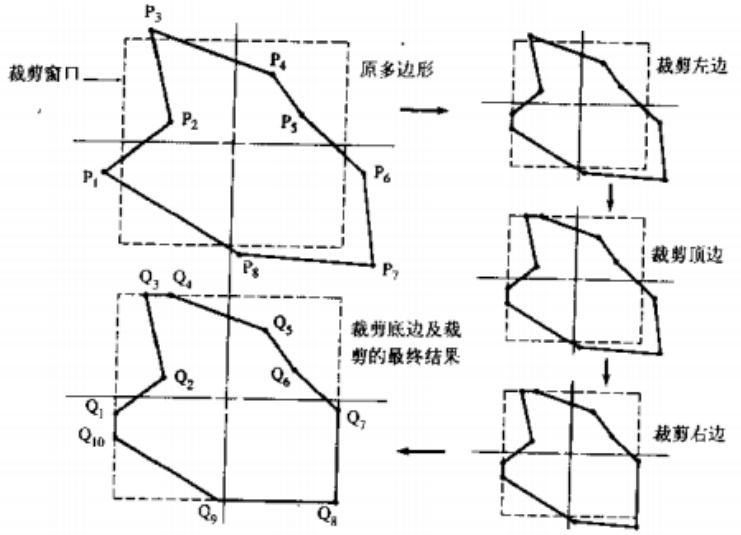


图3-31 逐次多边形裁剪

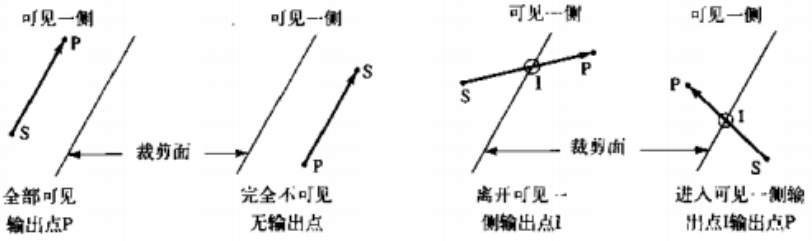


图3-32 边与裁剪面之间的关系

若边部分可见，则边或者进入、或者离开裁剪面的可见侧。如果边离开可见侧，则必须计算并输出多边形边与裁剪面的交点。如果边进入可见侧，则同样需计算并输出多边形与裁剪面的交点。由于此时边的终点P可见，故也应输出。

对于多边形的第一个顶点，只需判断它是否可见。若可见，则输出并作为起点S。若不可见，则不输出，但仍需作为起点S保存。

最后一边P_nP_1需分开处理。做法是将顶点表中的第一个顶点保存为F。这样最后一边变为P_nF，然后可与其他边一样统一处理。

算法流程图[1]：

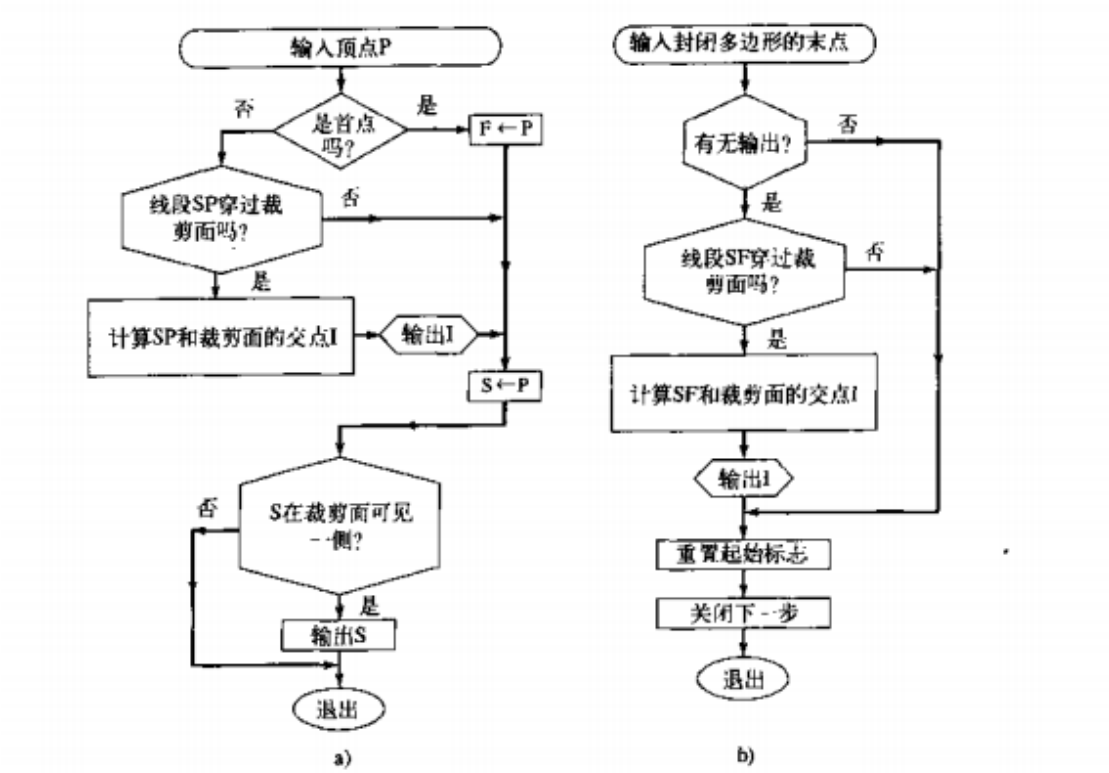


图3-35 Sutherland-Hodgman 逐次多边形裁剪框图

4. 程序实现

4.1Sutherland-Hodgman 多边形裁剪实现

```
//裁剪算法
    public static List<Point> Sutherland_Hodgeman(List<Point>
points,List<Vector> vectors){
        List<Point> result = new ArrayList<Point>();
        List<Point> cur = new ArrayList<Point>();

        int vectorsSize = vectors.size();
        int pointSize = points.size();

        Point S = points.get(pointSize-1);
        //初始化操作的集合
```

```
        for(int i=0;i<pointSize;i++){
            result.add(points.get(i));
        }

        boolean flag;
        for(int j=0;j<vectorsSize;j++){
            //flag = false表示在内侧，flag = true表示在外侧
            if(isInside(S,vectors.get(j)))
                flag = false;
            else
                flag = true;
            int resultSize = result.size();
            for(int i=0;i<resultSize;i++){
                //证明其在当前vector的内
                if(isInside(result.get(i),vectors.get(j))){
                    //如果前一个点在vector的外侧，那么将他们的交点加入到结果集中
                    if(flag){
                        flag = false;
                        cur.add(Intersection(S, result.get(i),
vectors.get(j).start, vectors.get(j).end));
                    }
                    //并将当前节点加入到结果集中
                    cur.add(result.get(i));
                }
                else{
                    //前一个点在外侧吗？
                    if(!flag){
                        flag = true;
                        //如果前一个点在vector的内侧，那么将他们的交点加入到结
果集中
                        cur.add(Intersection(S, result.get(i),
vectors.get(j).start, vectors.get(j).end));
                    }
                }
                //更新首次比较的节点
                S = result.get(i);
            }
            //将本次结果拷贝出来，作为下次对比的样本，并将本次结果进行清空
            int resultLen = cur.size();
            result.clear();
            for(int i=0;i<resultLen;i++){
                result.add(cur.get(i));
            }
            cur.clear();
```

```
        }
        return result;
    }

    //求一个点是否在一条边的内侧，在点序为逆时针的时候（如果点在线上，也算在内侧）
    public static boolean isInside(Point p,Vector v){
        return Multi(p,v.start,v.end)>=0?true:false;
    }


    //求叉积p0->p1与p0->p2的叉积
    public static double Multi(Point p0,Point p1,Point p2){
        return (p1.x-p0.x)*(p2.y-p0.y)-(p2.x-p0.x)*(p1.y-p0.y);
    }

    public static Point Intersection(Point start0,Point end0,Point
start1,Point end1){
        //由正弦定理推出
        double pX = (Multi(start0, end1, start1)*end0.x - Multi(end0, end1,
start1)*start0.x)/
                (Multi(start0, end1, start1) - Multi(end0, end1, start1));
        double pY = (Multi(start0, end1, start1)*end0.y - Multi(end0, end1,
start1)*start0.y)/
                (Multi(start0, end1, start1) - Multi(end0, end1, start1));
        return new Point(pX,pY);
    }
```

5. 操作介绍

5.1 三角形窗口裁剪三角形

Step1：选择 Window
Step2：选择 Triangle
Step3：在屏幕上点击三点即可
Step4：选择 Graph
Step5： 选择 Triangle
Step6： 在屏幕上画三个点，每画三个点就多一个三角形。同时显示被剪裁区域为黑色。

5.2 三角形窗口剪裁矩形

Step1：选择 Window
Step2：选择 Triangle
Step3：在屏幕上点击三点即可
Step4：选择 Graph
Step5： 选择 Rectangle
Step6： 在屏幕上画两个点，每画两个点就多一个矩形。同时显示被剪裁区域为黑色。

5.3 三角形窗口剪裁多边形

Step1：选择 Window
Step2：选择 Triangle
Step3：在屏幕上点击三点即可
Step4：选择 Graph
Step5： 选择 Polygon
Step6： 在屏幕上点击鼠标左键画任意多个点，双击左键结束。同时显示被剪裁区域为黑色。

5.4 矩形窗口剪裁三角形

Step1：选择 Window
Step2：选择 Rectangle
Step3：在屏幕上点击两点即可
Step4：选择 Graph
Step5： 选择 Triangle

Step6：在屏幕上画三个点，每画三个点就多一个三角形。同时显示被剪裁区域为黑色。

5.5 矩形窗口剪裁矩形

Step1：选择 Window
Step2：选择 Rectangle
Step3：在屏幕上点击两点即可
Step4：选择 Graph
Step5： 选择 Rectangle
Step6：在屏幕上画两个点，每画两个点就多一个矩形。同时显示被剪裁区域为黑色。

5.6 矩形窗口剪裁任意多边形

Step1：选择 Window
Step2：选择 Rectangle
Step3：在屏幕上点击两点即可
Step4：选择 Graph
Step5： 选择 Polygon
Step6： 在屏幕上点击鼠标左键画任意多个点，双击左键结束。同时显示被剪裁区域为黑色。

5.7 多边形窗口剪裁三角形

Step1：选择 Window
Step2：选择 Polygon
Step3：在屏幕上点击鼠标左键画任意多个点，双击左键结束
Step4：选择 Graph
Step5： 选择 Triangle
Step6：在屏幕上画三个点，每画三个点就多一个三角形。同时显示被剪裁区域为黑色。

5.8 多边形窗口剪裁矩形

Step1：选择 Window
Step2：选择 Polygon
Step3：在屏幕上点击鼠标左键画任意多个点，双击左键结束
Step4：选择 Graph
Step5： 选择 Rectangle
Step6：在屏幕上画两个点，每画两个点就多一个矩形。同时显示被剪裁区域为黑

色

5.9 多边形窗口剪裁多边形

Step1：选择 Window
Step2：选择 Polygon
Step3：在屏幕上点击鼠标左键画任意多个点，双击左键结束
Step4：选择 Graph
Step5： 选择 Polygon
Step6：在屏幕上点击鼠标左键画任意多个点，双击左键结束
。同时显示被剪裁区域为黑色

6. 类介绍

- 类表

所属包	名称	标识符	数据项	操作	层次关系
Graphics	主界面	Home	ContentPane Panel comboBox comboBox_1 comboBox_2	drawimage	继承
graphics	点类	Poing	x, y	getX getY setX setY	
graphics	多边形	Vector	Start end	Vector	