

# 《计算机图形学》 实验报告

## 线画图元生成算法实验 & 裁剪算法实验

姓名\_\_\_\_\_王昊恩\_\_\_\_\_

学号\_\_\_\_\_3018216021\_\_\_\_\_

专业\_\_\_\_\_计算机科学与技术\_\_\_\_\_

班级\_\_\_\_\_1 班\_\_\_\_\_

天津大学智能与计算学部

2020 年 10 月 19 日

## 一、目的

通过实验，深入理解并进一步学习计算机图形学基础算法，在课堂学习算法的思路后利用 MFC 实现这些算法。

本次两个实验均使用 C/C++ 语言完成，利用 MFC 应用，应用程序类型为 单个文档，Microsoft Visual Studio Professional 2019 版本号为 16.7.5。

本次实验报告及源代码已上传至 Github，

仓库地址为：[https://github.com/FireSSang/20\\_21\\_1\\_Computer\\_Graphics](https://github.com/FireSSang/20_21_1_Computer_Graphics)

## 二、实验内容

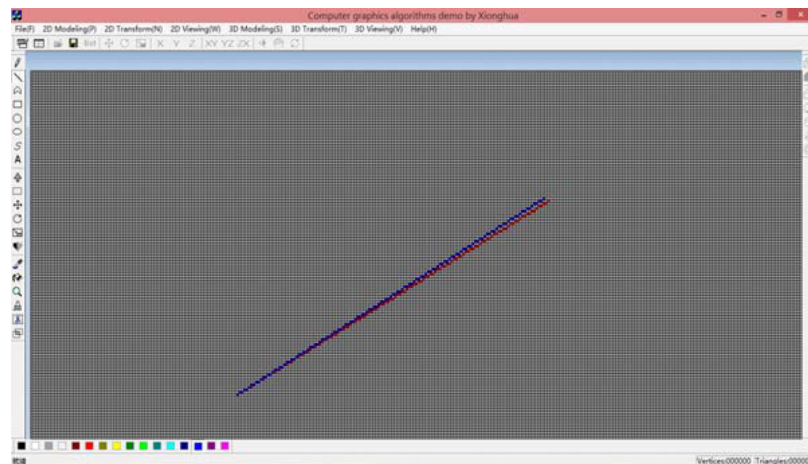
### • (1) 线画图元生成算法

内容：

1. 实现 DDA 直线生成算法
2. 实现 Bresenham 直线生成算法

要求：

1. 自定义直线段起点和终点坐标；
2. 采用不同的彩色显示两种算法生成的直线结果；
3. 为突出显示效果，采用网格表示像素，实现如下绘制效果。



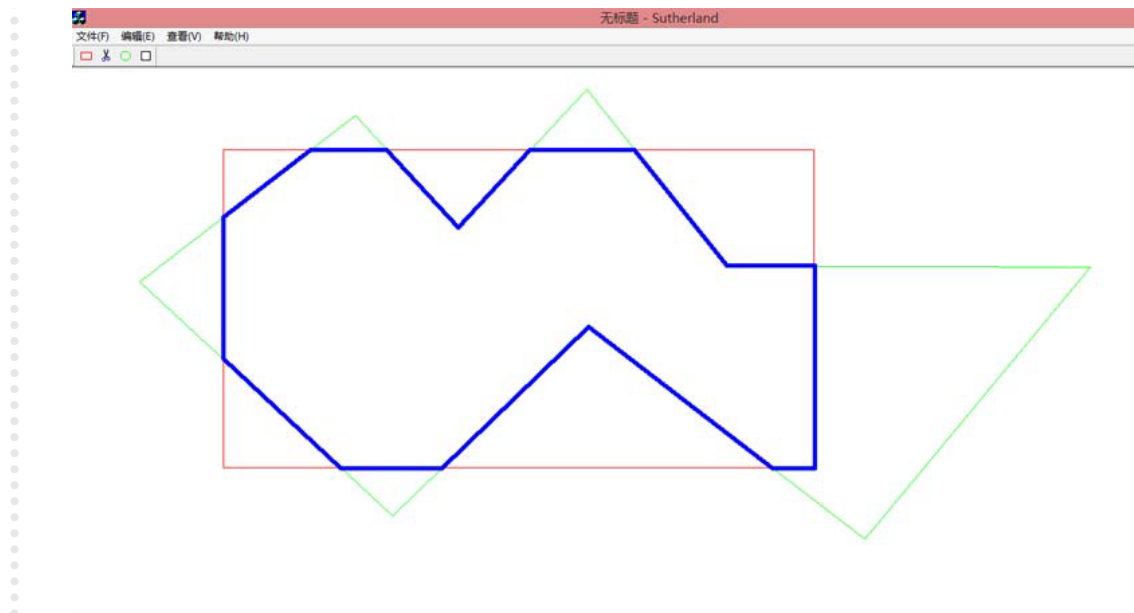
### • (2) 裁剪算法实验

内容：

1. 实现 Cohen-Sutherland 直线裁剪算法（选做）
2. 实现 Sutherland-Hodgman 多边形裁剪算法

要求：

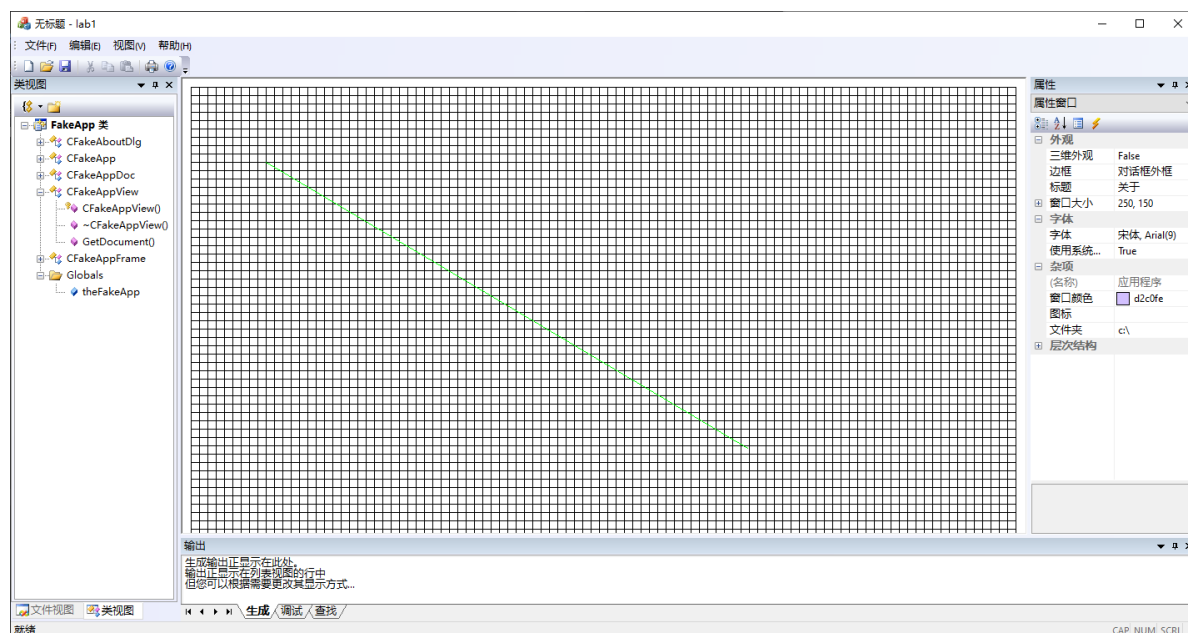
1. 自定义裁剪窗口和待裁剪直线段（或多边形）；
2. 采用不同颜色突出显示裁剪结果，如下图所示。



### 三、结果

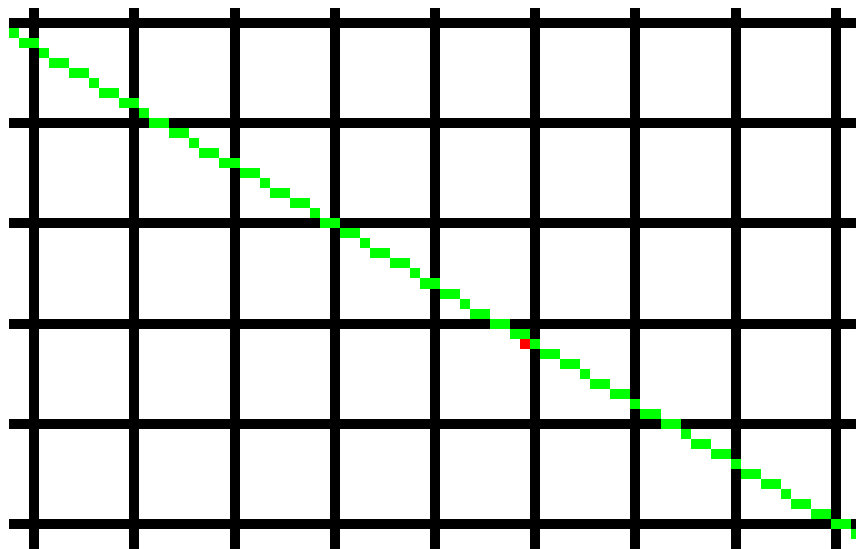
#### • (1) 线画图元生成算法

实验结果如下：



其中，DDA 算法所画直线使用红色 (255,0,0)，Bresenham 算法所画直线使用绿色 (0,255,0)，先使用 DDA 算法，后使用 Bresenham 算法，故实验结果所画直线主要为绿色 (0,255,0)。

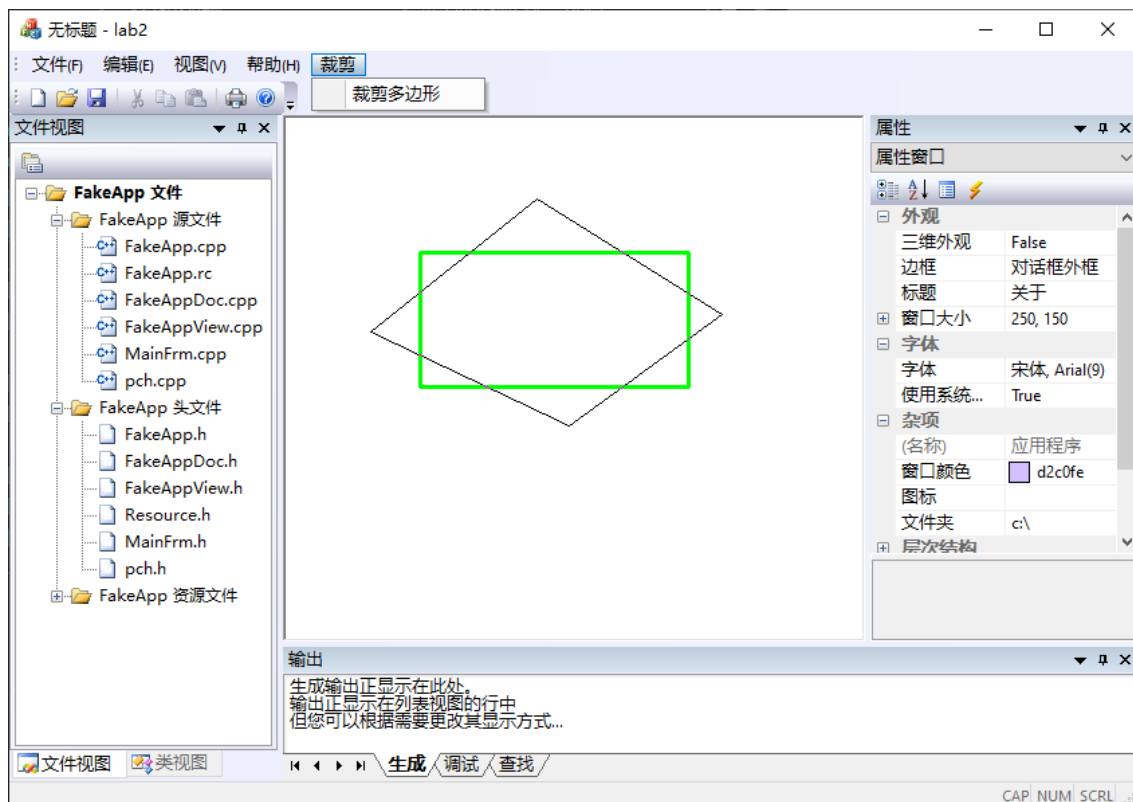
通过仔细观察，发现两算法仅在一处出现差异，以下为放大后的局部图片：



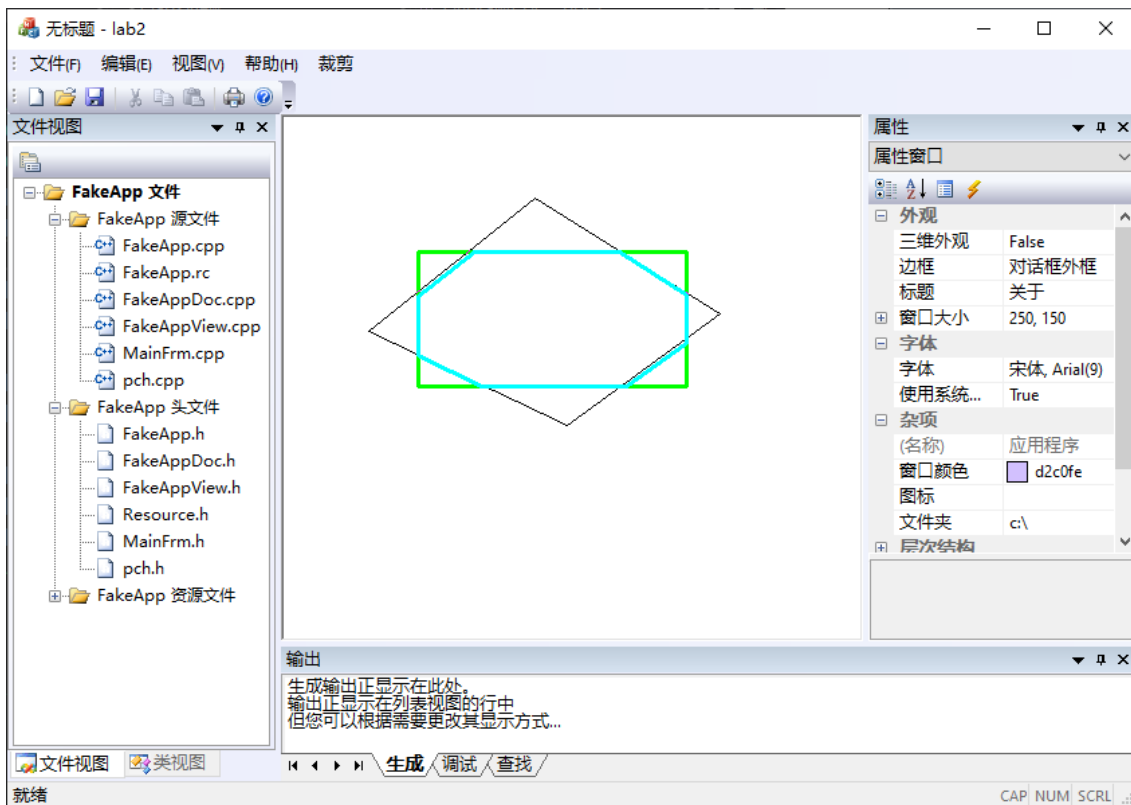
## • (2) 裁剪算法实验

为简化操作，我固定了裁剪区域，并以绿色 (0,255,0) 表示，使用橡皮筋技术绘制待裁剪多边形，以黑色表示 (0,0,0)。

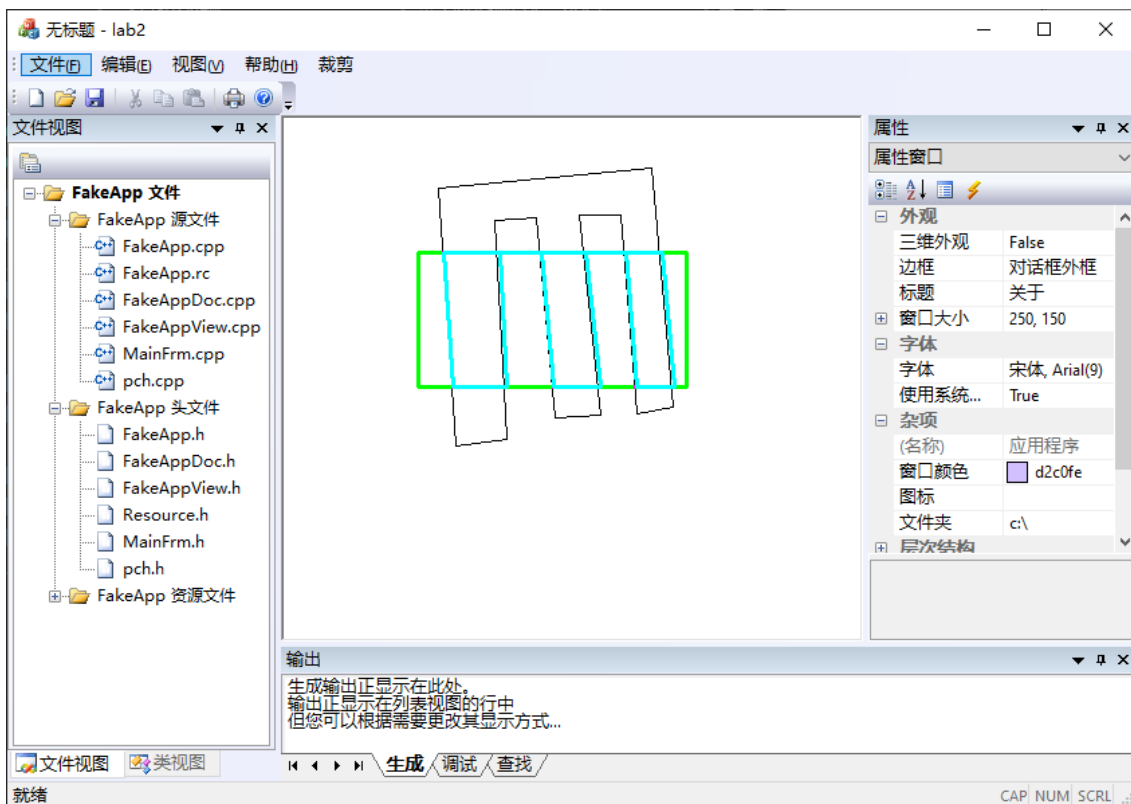
绘制待裁剪多边形后，裁剪前：

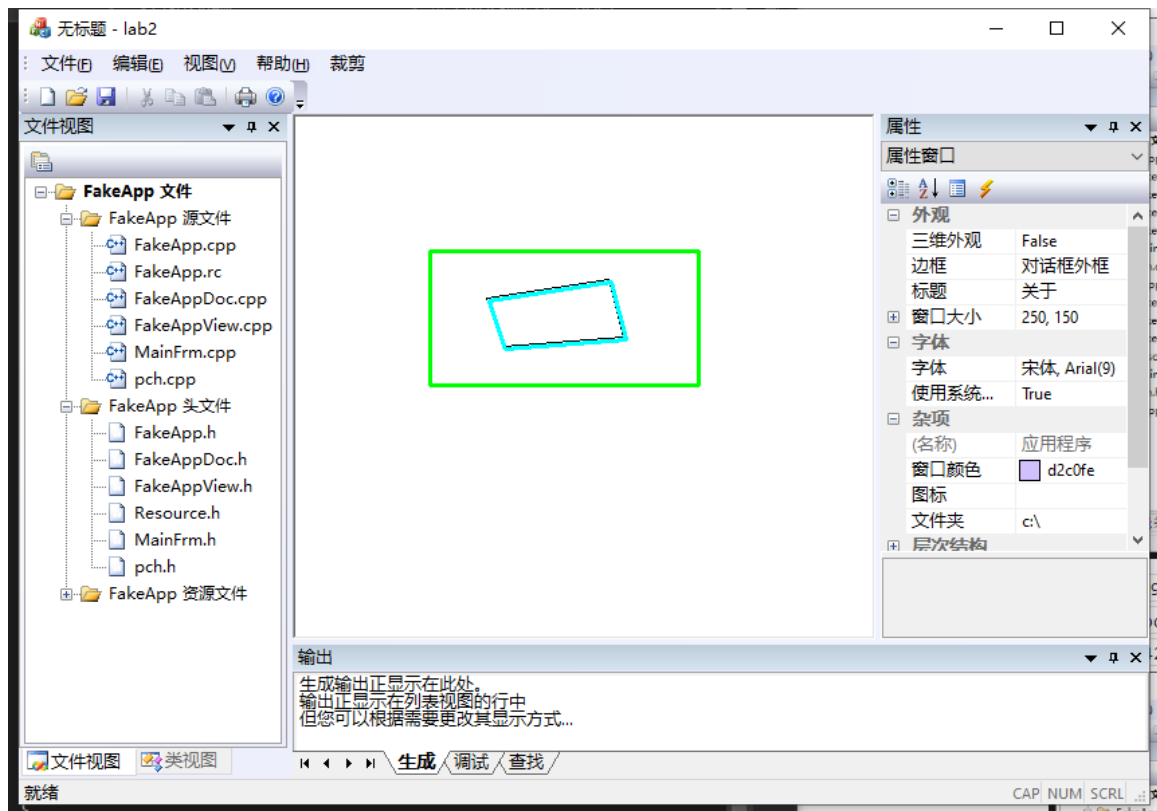
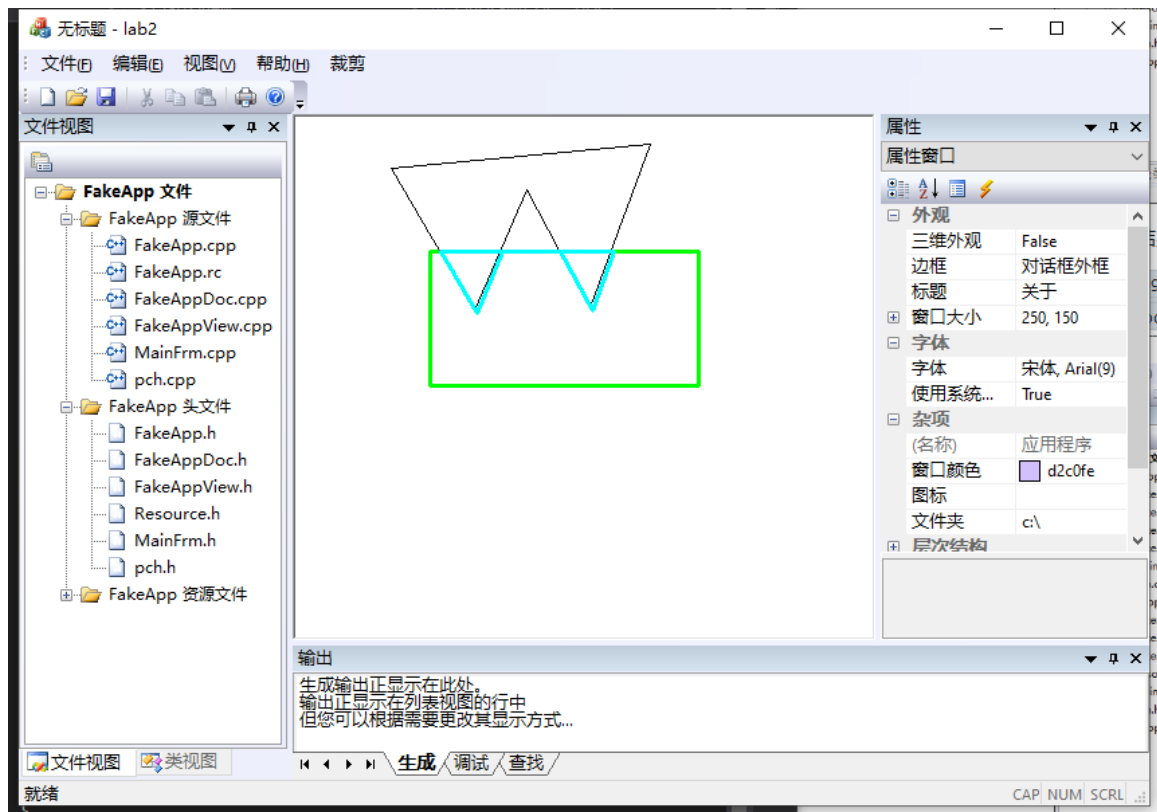


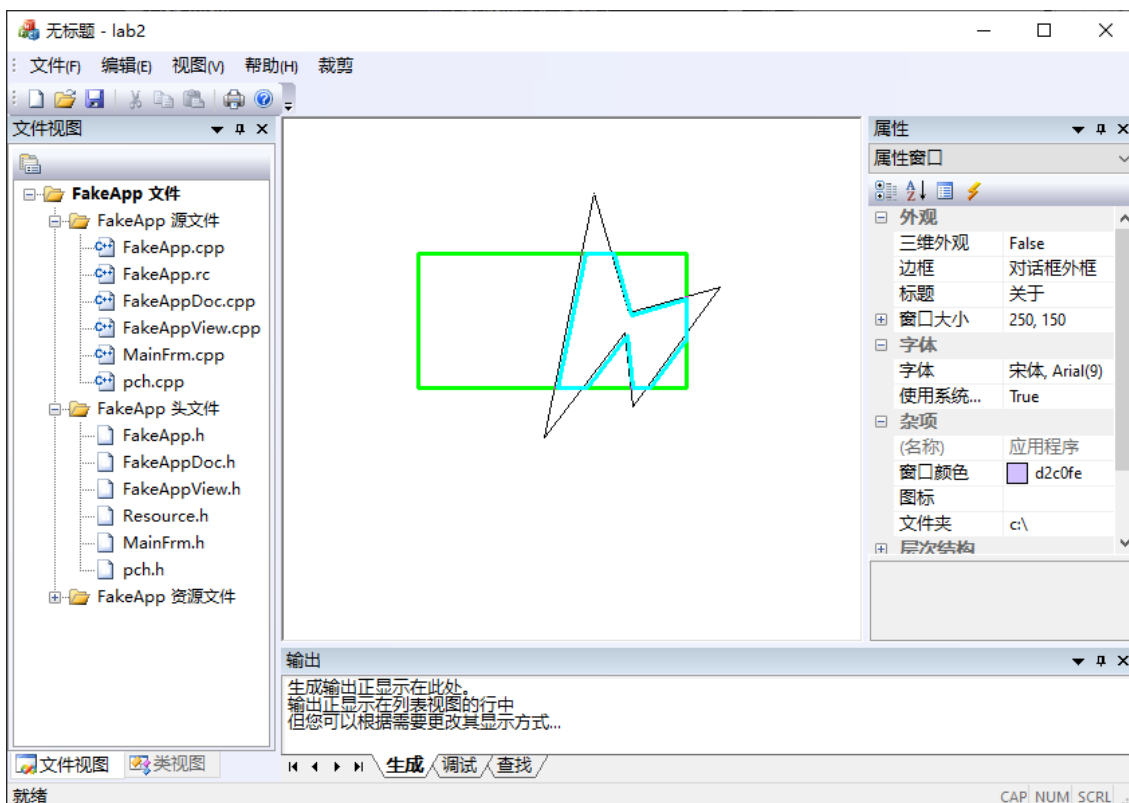
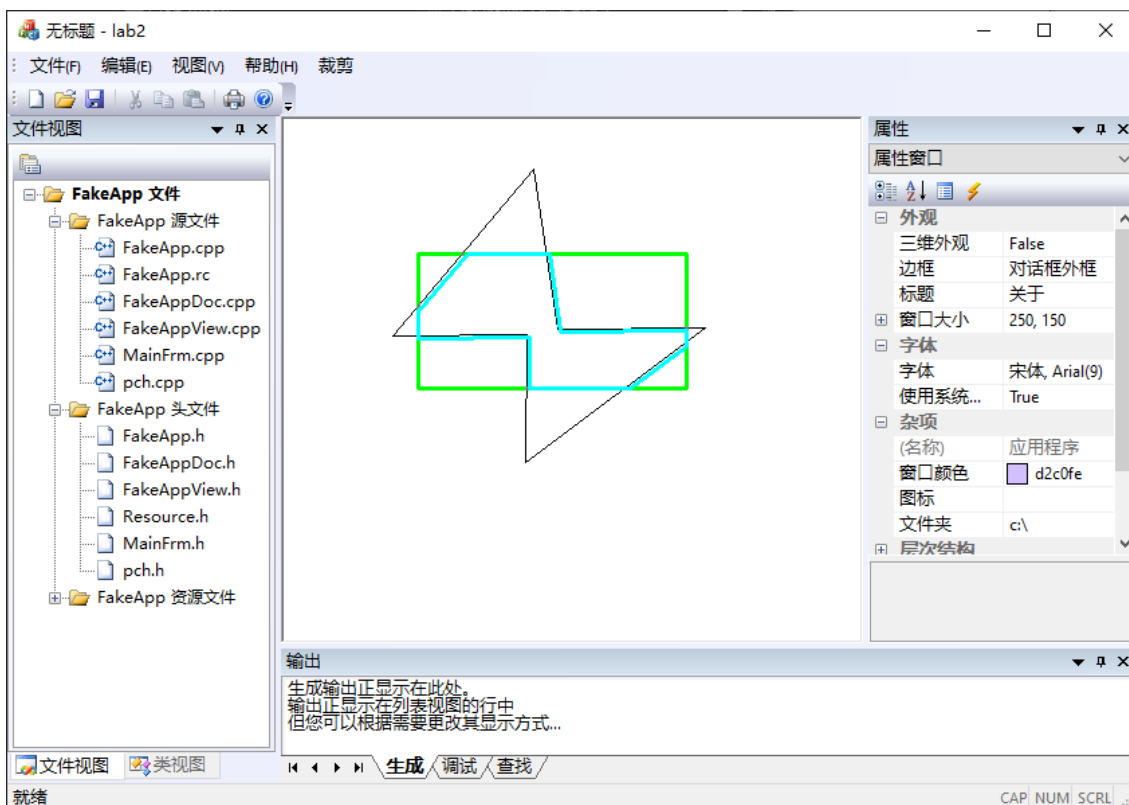
使用设定好的时间处理程序进行裁剪，裁剪后区域以青色绘制 (0,255,255)：



以下是其他多边形的裁剪结果展示：







## 四、分析和总结

### • (1) 线画图元生成算法

本实验主要难点在于两算法处理各种斜率的直线时需要分类讨论，总体实现难度不大。

通过本实验初步了解了 [MFC应用](#)，能够实现简单的计算机图形学算法。

## • (2) 裁剪算法实验

本实验我使用 橡皮筋技术 绘制待裁剪多边形，网络上关于 MFC应用 的教程过少并且描述太过简略，在 消息处理程序 和 事件处理程序 的添加、修改中遇到了很大的困难，耗费了很多时间和精力，好在通过摸索最终完成了本次实验。经过这次实验，我深入了解了 MFC应用 的实现原理，熟悉了 MFC应用 的使用方法，通过自己摸索留下的印象或许会更加深刻。

在 Sutherland-Hodgman 算法的编程中，由于编程水平不足，我的裁剪算法写的过于冗杂，好在结果正确，这暴露出我对该算法的理解不够深入、编程水平不足两个问题，在日后的学习研究工作中我需要继续提升编程水平，避免此类问题再次发生。

## 五、源代码

### • (1) 线画图元生成算法

lab1View.h 中添加以下函数定义：

```
1 void swap(int a, int b);
2 void DDA_line(CDC* pDC, int x0, int y0, int x1, int y1, COLORREF color);
3 void Bre_line(CDC* pDC, int x1, int y1, int x2, int y2, COLORREF color);
4 void grid(CDC* pDC);
```

lab1View.cpp 中添加及修改以下函数：

```
1 // Clab1View swap函数
2 void Clab1View::swap(int a, int b)
3 {
4     int temp = a;
5     a = b;
6     b = temp;
7 }
8
9 // Clab1View DDA算法
10
11 void Clab1View::DDA_line(CDC* pDC, int x0, int y0, int x1, int y1, COLORREF color)
12 {
13     double k; //斜率
14     if (x1 != x0 && fabs(k = (y1 - y0) * 1.0 / (x1 - x0)) <= 1) //斜率<=1时
15     {
16         //交换
17         if (x0 > x1)
18         {
19             swap(x0, x1);
20             swap(y0, y1);
21         }
22
23         double y = y0;
24         for (int x = x0; x <= x1; x++)
25         {
26             pDC->SetPixel(x, int(y + 0.5), color); //绘制像素点
27             y += k;
28         }
29     }
30     else //斜率>1时
31     {
32         k = (x1 - x0) * 1.0 / (y1 - y0);
```



```

33         //交换
34         if (y0 > y1)
35         {
36             swap(x0, x1);
37             swap(y0, y1);
38         }
39
40         double x = x0;
41         for (int y = y0; y <= y1; y++)
42         {
43             pDC->SetPixel(int(x + 0.5), y, color);
44             x += k;
45         }
46     }
47 }
48
49 // Clab1View Bresenham算法
50
51 void Clab1View::Bre_line(CDC* pDC, int x1, int y1, int x2, int y2, COLORREF color)
52 {
53     int dx, dy; // 横轴纵轴差量
54     int e;
55     int x, y;
56     dx = abs(x2 - x1);
57     dy = abs(y2 - y1);
58     y = y1;
59     x = x1;
60     int cx, cy; // 表明x、y方向的增量
61     if (x1 > x2)
62         cx = -1; // x递减方向
63     else
64         cx = 1; // x递增方向
65     if (y1 > y2)
66         cy = -1; // y递减方向
67     else
68         cy = 1; // y递增方向
69     if (dx == 0 && dy == 0)
70     {
71         printf("The input is not a line. It is just a point. Please input two
different points.!\n");
72     }
73     else if (dy == 0)
74     {
75         for (x = x1; (cx == 1 ? x <= x2 : x >= x2); x += cx)
76         {
77             pDC->SetPixel(x, y, color);
78         }
79     }
80     else if (dx == 0)
81     {
82         for (y = y1; (cy == 1 ? y <= y2 : y >= y2); y += cy)
83         {
84             pDC->SetPixel(x, y, color);
85         }
86     }
87     else if (dx >= dy) {
88         e = -dx;
89         for (x = x1; (cx == 1 ? x <= x2 : x >= x2); x += cx)
90         {
91             pDC->SetPixel(x, y, color);

```

```

93         e += dy << 1;
94         if (e > 0)
95         {
96             y += cy;
97             e -= dx << 1;
98         }
99     }
100 }
101 else {
102     e = -dy;
103     for (y = y1; (cy >= 0 ? y <= y2 : y >= y2); y += cy)
104     {
105         pDC->SetPixel(x, y, color);
106         e += dx << 1;
107         if (e > 0)
108         {
109             x += cx;
110             e -= dy << 1;
111         }
112     }
113 }
114 }
115
116 // Clab1grid 画个网格
117 void Clab1View::grid(CDC* pDC)
118 {
119     for (int i = 10; i <= 1000; i += 10)
120     {
121         for (int j = 10; j <= 1000; j++)
122         {
123             pDC->SetPixel(i, j, RGB(0, 0, 0));
124             pDC->SetPixel(j, i, RGB(0, 0, 0));
125         }
126     }
127 }
128
129
130 // Clab1View 绘图
131
132 void Clab1View::OnDraw(CDC* pDC)
133 {
134     Clab1Doc* pDoc = GetDocument();
135     ASSERT_VALID(pDoc);
136     if (!pDoc)
137         return;
138
139     // TODO: 在此处为本机数据添加绘制代码
140     grid(pDC);
141     DDA_line(pDC, 100, 100, 678, 443, RGB(255, 0, 0));
142     Bre_line(pDC, 100, 100, 678, 443, RGB(0, 255, 0));
143 }

```

## • (2) 裁剪算法实验

选择 类视图，右键单击 Clab2View，选择 类向导，选择 消息，添加以下两个消息处理程序（自动在 Clab2View.h 中添加定义，在 Clab2View.cpp 中创建函数定义），利用橡皮筋技术实现绘制待裁剪多边形。

OnLButtonDown：

```

1 void Clab2View::OnLButtonDown(UINT nFlags, CPoint point)
2 {
3     // TODO: 在此添加消息处理程序代码和/或调用默认值
4     if (if_Press == 1 || Number == 0)
5     {
6         if_Press = 1; //说明已经点过了一个点
7         start_point = point;
8         end_point = start_point;
9         Point[Number] = start_point; //把这个点存入数组里
10        if (Number == 0) //表示第一次左击的点为第一个点,
11            first = start_point; //把第一个点保存下来
12        Number++; //多边形的点数加一
13    }
14    CView::OnLButtonDown(nFlags, point);
15 }

```

OnMouseMove :

```

1 void Clab2View::OnMouseMove(UINT nFlags, CPoint point)
2 {
3     // TODO: 在此添加消息处理程序代码和/或调用默认值
4     CDC* pDC = this->GetWindowDC();
5     if (if_Press)
6     {
7         pDC->SetROP2(R2_NOT); //添加橡皮绳（只要没左击，鼠标在桌面上移动，线条就会跟到
        哪去）
8         pDC->MoveTo(start_point); //将点坐标移到上一个点击的点的位置
9         pDC->LineTo(end_point); //没有画出线，此时的start_point = end_point
10        if ((point.x - first.x) < 0.1 && (point.y - first.y) < 0.1) //当鼠标指着的点离
        第一个点的距离小于一个值的时候，两点重合（这里两个点会自动重合，不用再次点击左键）
11        {
12            point = first;
13            if_Press = 0;
14        }
15        pDC->MoveTo(start_point); //画出直线
16        pDC->LineTo(point);
17        end_point = point;
18    }
19    CView::OnMouseMove(nFlags, point);
20 }

```

在 Clab2View.h 中添加如下定义，包括待裁剪多边形的存储、裁剪结果的存储以及裁剪区域的边界值：

```

1     CPoint start_point; //起始点
2     CPoint end_point; //终止点
3     CPoint first; //第一个点
4     CPoint Point[100];
5     int if_Press = 0;
6     int Number = 0; //多边形的点数
7
8     CPoint New_Point[100];
9
10    int XL = 100, XR = 300, YB = 100, YT = 200; //分别是左边的x值，右边的x值，下边的y
    值，上边的y值

```

在 Clab2View.cpp 中修改 OnDraw 函数，设定裁剪区域，并以绿色 (0,255,0) 着色：

```

1 void Clab2View::OnDraw(CDC* pDC)
2 {
3     Clab2Doc* pDoc = GetDocument();
4     ASSERT_VALID(pDoc);
5     if (!pDoc)
6         return;
7
8     // TODO: 在此处为本机数据添加绘制代码
9     CPen newpen(PS_SOLID, 3, RGB(0, 255, 0));
10    CPen* old = pDC->SelectObject(&newpen);
11    pDC->MoveTo(100, 100);
12    pDC->LineTo(100, 200);
13
14    pDC->MoveTo(100, 200);
15    pDC->LineTo(300, 200);
16
17    pDC->MoveTo(300, 200);
18    pDC->LineTo(300, 100);
19
20    pDC->MoveTo(300, 100);
21    pDC->LineTo(100, 100);
22    pDC->SelectObject(old);
23    //ReleaseDC(pDC);
24 }

```

选择 资源视图，展开 lab2，展开 lab2.rc，展开 Menu，双击 IDR\_MAINFRAME，在 帮助 后添加一个 裁剪 控件，下拉菜单添加一个选项并命名为 裁剪多边形，右键 裁剪多边形 控件，选择 添加事件处理程序，类列表中选择 Clab2View，消息类型选择 COMMAND，函数名命名为 OnSutherHodgman，在 Clab2View.cpp 中完成该函数的定义：

```

1 void Clab2View::OnSutherHodgman()
2 {
3     // TODO: 在此添加命令处理程序代码
4     {
5         // TODO: 在此添加命令处理程序代码
6         int temp_num; //这个临时的变量存储的就是多边形的点个数，也就是前面画多边形时的
Number
7         int new_num = 0; //这个变量是用来存储通过算法裁剪后的点的个数
8         CPoint temp_point; //定义一个临时点，在后面是用来求交点用的
9         int i;
10        int j;
11        temp_num = Number;
12
13        //先说明下，画线的时候是从起始点到终止点，在后面的解释中我会把起始点说成头点，终
止点说成尾点，也就是从头到尾，还有一点，如果这个点在可视区外，在后面就直接简称为外了
14        ///////////////
15        //对矩形框上边处理
16        for (i = 0; i < temp_num; i++)//temp_num 为多边形的边数，执行次数也就是边数
17        {
18            if (Point[i % temp_num].y > YT) //如果头点在上边之上，即头点在不可视区
19            { //这样要注意下Point数组选取元素是加的%号的含义，因为头号是i，尾号是
i+1，可是到了最后一个点的时候，头号还是i，可是尾却不是i+1了，此时的头要和最初的第一个点相
连，才能形成一个闭合的多边形，所以整除（点的个数），刚好能让最后一个的值变成0
20                //本来%只是最后一个点才用得着，这里为了不乱掉，索性全加上，反正对前面的点也没
什么影响。。。
21                if (Point[(i + 1) % temp_num].y > YT) //尾点也在上边之上，即两点都在
不可视区，均不进入新数组，这种情况就是（外到外），所以什么都不用写。
22                {
23

```

```

24     }
25     else //尾点在上边之下，即尾点在可视区，尾点和与上边的交点进入新数组（外
到内）
26     {
27
28         temp_point.x = Point[i % temp_num].x - (Point[i % temp_num].y
- YT) * (Point[i % temp_num].x - Point[(i + 1) % temp_num].x)
29         / (Point[i % temp_num].y - Point[(i + 1) % temp_num].y);
        //这一大串求交点的x坐标值是通过相似的原理来表示的，想深究的朋友可以画个实例图研究，接下来的
很多求交点都是通过这个原理
30         temp_point.y = YT; //求交点的y坐标，因为交点在上边，所以上边的y值
就是交点的y值
31         New_Point[new_num] = temp_point; //把这个临时的交点存到New_point
数组里，同时点的个数加一，接下来同样每送到新数组一个点，new_num（新多边形的点个数）就加一
32         new_num += 1;
33         /*New_Point[new_num] = Point[(i + 1) % temp_num]; //这里要注意
下，按理来说，画一条从外到内的线应该要把与边的交点和内点都要进入新数组里，但是内点却不在这
段代码中加到新数组里面，而是到下一个循环，这个内点从尾点变成头点的时候加到数组里
34         new_num += 1;*/ //如果在这里
加到数组里会导致这样一种情况，这个点作为尾点加到数组，到下一个循环，这个点变成了头点，难道
又要再加一遍？这会导致同一个点在数组内存在了两次
35         //凡是都有例外，其中有一种情况是要在一段代码中添加两个点的，接下来会
解释到
36
37     }
38 }
39 else //头点在上边之下，即在可视区
40 {
41     if (Point[(i + 1) % temp_num].y <= YT) //尾点也在可视区，头尾两点均
入新数组（内到内）
42     {
43         New_Point[new_num] = Point[i % temp_num];
44         new_num += 1;
45         /*New_Point[new_num] = Point[(i + 1) % temp_num];
46         new_num += 1;*/
47     }
48     else //尾点在不可视区，所以头点和交点进入新数组（内到外），这种情况就是
上面说到的要添加两个点，大家可以拿笔画个草图看看，内到外，肯定先把内点加到新数组里，如果说
把那个交点留着下一个循环来处理会导致什么样的情况，下一次肯定是从外到内
49     { //或者从外到外，不管哪种情况，能确定的是和这个交点已经没什么关系
了，画图很容易看出来，所以如果不在这种情况下把交点加到新数组里，就会导致这个交点漏掉，即使
在下面的循环也没法把这个交点加到数组了
50         New_Point[new_num] = Point[i % temp_num];
51         new_num += 1;
52         //计算交点坐标
53         temp_point.x = Point[i % temp_num].x + (YT - Point[i %
temp_num].y) * (Point[(i + 1) % temp_num].x - Point[i % temp_num].x)
54         / (Point[(i + 1) % temp_num].y - Point[i % temp_num].y);
55         temp_point.y = YT;
56         New_Point[new_num] = temp_point;
57         new_num += 1;
58     }
59 }
60 }
61 }
62 //////////////////////////////////////////////////
63
64 //重新整理处理上边后的图形
65
66
67 for (j = 0; j < new_num; j++)

```

```

68     {
69         Point[j] = New_Point[j]; //将新数组的点调回Point, 此时的新数组已经保留了裁
        剪上边后的新的点, 这时, 把这个经过上边处理后的多边形调回Point, 可以这么说, 把这个新的多边形
        重新作为待处理的多边形, 进行下边处理
70     }
71     Number = new_num; //此时的多边形点数为上边裁剪后的新多边形的点数
72     new_num = 0; //归0, 进行下边处理
73     temp_num = Number;
74     //对矩形框下边处理, 接下来的对下边, 左边, 右边和上边处理类似, 只是在一些细节方面
        做了一些改变, 例如在求交点的相似处理上, 有所改变, 而且上下两边和左右两边的处理也稍有不同,
        不过大同小异, 把上边处理看懂后, 下面的都差不多
75     for (i = 0; i < temp_num; i++) //temp_num 为多边形的边数, 执行次数也就是边数
76     {
77         if (Point[i % temp_num].y > YB) // 头点在下边上, 即头点在可视区内
78         {
79             if (Point[(i + 1) % temp_num].y > YB) // 尾点也在可视区内, 两点均在
                可视区内 (内到内)
80             {
81                 New_Point[new_num] = Point[i % temp_num];
82                 new_num += 1;
83                 /*New_Point[new_num] = Point[(i + 1) % temp_num];
84                 new_num += 1;*/
85             }
86             else //尾点在可视区外, 头点和交点进入新数组 (内到外)
87             {
88                 New_Point[new_num] = Point[i % temp_num];
89                 new_num += 1;
90                 //求交点
91                 temp_point.x = Point[i % temp_num].x - (Point[i % temp_num].y
- YB) * (Point[i % temp_num].x - Point[(i + 1) % temp_num].x)
92                 / (Point[i % temp_num].y - Point[(i + 1) % temp_num].y);
93                 temp_point.y = YB;
94                 New_Point[new_num] = temp_point;
95                 new_num += 1;
96             }
97         }
98     }
99     else //头在可视区外
100     {
101         if (Point[(i + 1) % temp_num].y >= YB) //尾在可视区内 (外到内)
102         {
103             //计算交点坐标
104             temp_point.x = Point[i % temp_num].x + (YB - Point[i %
temp_num].y) * (Point[(i + 1) % temp_num].x - Point[i % temp_num].x)
105             / (Point[(i + 1) % temp_num].y - Point[i % temp_num].y);
106             temp_point.y = YB;
107             New_Point[new_num] = temp_point;
108             new_num += 1;
109             /*New_Point[new_num] = Point[(i + 1) % temp_num];
110             new_num += 1;*/
111         }
112         else //头尾两点均在可视区外 (外到外)
113         {
114
115
116         }
117     }
118 }
119 //重新整理处理下边后的图形
120 for (j = 0; j < new_num; j++)
121 {

```

```

122         Point[j] = New_Point[j]; //将新数组的点调回Point
123     }
124     Number = new_num; //此时的点数为改变上边后的图形的点数
125     new_num = 0; //归0, 进行下边处理
126     temp_num = Number;
127     //对矩形框左边处理
128     for (i = 0; i < temp_num; i++) //temp_num 为多边形的边数, 执行次数也就是边数
129     {
130         if (Point[i % temp_num].x >= XL) // 头点在可视区内
131         {
132             if (Point[(i + 1) % temp_num].x >= XL) // 尾点也在可视区内, 两点均在
可视区内(内到内)
133             {
134                 New_Point[new_num] = Point[i % temp_num];
135                 new_num += 1;
136                 /*New_Point[new_num] = Point[(i + 1) % temp_num];
137                 new_num += 1;*/
138             }
139             else //尾点在可视区外, 头点和交点进入新数组(内到外)
140             {
141                 New_Point[new_num] = Point[i % temp_num];
142                 new_num += 1;
143                 //求交点
144                 temp_point.y = Point[i % temp_num].y - (Point[i % temp_num].x
- XL) * (Point[i % temp_num].y - Point[(i + 1) % temp_num].y)
145                 / (Point[i % temp_num].x - Point[(i + 1) % temp_num].x);
146                 temp_point.x = XL;
147                 New_Point[new_num] = temp_point;
148                 new_num += 1;
149             }
150         }
151     }
152     else //头在可视区外
153     {
154         if (Point[(i + 1) % temp_num].x >= XL) //尾在可视区内(外到内)
155         {
156             //计算交点坐标
157             temp_point.y = Point[i % temp_num].y + (XL - Point[i %
temp_num].x) * (Point[(i + 1) % temp_num].y - Point[i % temp_num].y)
158             / (Point[(i + 1) % temp_num].x - Point[i % temp_num].x);
159             temp_point.x = XL;
160             New_Point[new_num] = temp_point;
161             new_num += 1;
162             /*New_Point[new_num] = Point[(i + 1) % temp_num];
163             new_num += 1;*/
164         }
165         else //头尾两点均在可视区外(外到外)
166         {
167
168
169         }
170     }
171 }
172 //重新整理处理左边后的图形
173 for (j = 0; j < new_num; j++)
174 {
175     Point[j] = New_Point[j]; //将新数组的点调回Point
176 }
177 Number = new_num; //此时的点数为改变上边后的图形的点数
178 new_num = 0; //归0, 进行下边处理
179 temp_num = Number;

```

```

180 //对矩形框右边处理
181 for (i = 0; i < temp_num; i++)//temp_num 为多边形的边数，执行次数也就是边数
182 {
183     if (Point[i % temp_num].x <= XR) // 头点在可视区内
184     {
185         if (Point[(i + 1) % temp_num].x <= XR) // 尾点也在可视区内，两点均在
可视区内(内到内)
186         {
187             New_Point[new_num] = Point[i % temp_num];
188             new_num += 1;
189             /*New_Point[new_num] = Point[(i + 1) % temp_num];
190             new_num += 1;*/
191         }
192         else //尾点在可视区外，头点和交点进入新数组(内到外)
193         {
194             New_Point[new_num] = Point[i % temp_num];
195             new_num += 1;
196             //求交点
197             temp_point.y = Point[i % temp_num].y + (XR - Point[i %
temp_num].x) * (Point[(i + 1) % temp_num].y - Point[i % temp_num].y)
198             / (Point[(i + 1) % temp_num].x - Point[i % temp_num].x);
199             temp_point.x = XR;
200             New_Point[new_num] = temp_point;
201             new_num += 1;
202         }
203     }
204     else //头在可视区外
205     {
206         if (Point[(i + 1) % temp_num].x <= XR) //尾在可视区内(外到内)
207         {
208             //计算交点坐标
209             temp_point.y = Point[i % temp_num].y - (Point[i % temp_num].x
- XR) * (Point[i % temp_num].y - Point[(i + 1) % temp_num].y)
210             / (Point[i % temp_num].x - Point[(i + 1) % temp_num].x);
211             temp_point.x = XR;
212             New_Point[new_num] = temp_point;
213             new_num += 1;
214             /*New_Point[new_num] = Point[(i + 1) % temp_num];
215             new_num += 1;*/
216         }
217         else //头尾两点均在可视区外(外到外)
218         {
219
220
221
222         }
223     }
224 }
225 //这里是最后一步，经过上下左右四条边的处理，此时的多边形已经裁剪完成，并且储存在
New_Point在，下面的工作就是用循环一条边一条边得画出来
226 CDC* pDC = this->GetDC();
227 CPen newpen(PS_SOLID, 3, RGB(0, 255, 255));
228 CPen* old = pDC->SelectObject(&newpen);
229 for (j = 0; j < new_num; j++)
230 {
231     pDC->MoveTo(New_Point[j % new_num].x, New_Point[j % new_num].y);
232     pDC->LineTo(New_Point[(j + 1) % new_num].x, New_Point[(j + 1) %
new_num].y);
233 }
234 pDC->MoveTo(New_Point[0].x, New_Point[0].y);
235 pDC->LineTo(New_Point[1].x, New_Point[1].y);

```



```
236     pDC->SelectObject(old);
237     ReleaseDC(pDC);
238
239 }
240 }
```