华东师范大学计算机科学与技术实验报告

实验课程：计算机图形学	年级：2018级	实验成绩：
实验名称：裁剪算法	姓名：董辰尧	实验日期：2021-4-20
实验编号：7	学号：10185102144	实验时间：13:00-14:30
指导教师：王长波、李洋	组号：	

一、实验目的

利用操作系统API实现基本裁剪算法。

二、实验环境

- 实现Sutherland-Cohen编码算法
- 实现直线剪裁算法
- 实现多边形剪裁算法

三、实验环境

Visual studio 2017

四、实验过程与分析

- 首先是下载代码并且导入，由于这个步骤每次实验都会做，是最基本的，就不放图了。
- 然后是实现Sutherland-Cohen编码算法。这个算法比较简单，ppt上面已经给出了完整的图片，甚至还有相对完整的伪代码。不过需要注意的是我们用的图像的坐标原点一直都是在左上角，所以y_min反而在上面。具体代码如下：

```
OutCode calc_OutCode(const Vector2 &P, const Vector2 &clip_min,
                     const Vector2 &clip_max) {
  OutCode code = INSIDE;
  // Write you code here
  //int x = P.x, y = P.y;
  if (P.x() < clip_min.x()) code |= LEFT;
  if (P.x() > clip_max.x()) code |= RIGHT;
  if (P.y() < clip_min.y()) code |= TOP;
  if (P.y() > clip_max.y()) code |= BOTTOM;
  return code;
}
```

- 接下来是实现实线剪裁算法。这个算法比较难的点在于前两种情况（$code1|code2=0$、$code1\&code2\neq 0$)都不符合的时候。就需要求出交点，然后用一个焦点的 坐标取代原来的一个不在可见范围内点的坐标。这里我的做法是挨个判断$code_out$在框框哪里，比如在下面就求出两点确定的直线和窗口底线所在直线的交点。核心代码如下：

```
while (true)
  {
      if ((code0 | code1) == 0) {
```

```
            accept = true;
            break;
        }
        if ((code0 & code1) != 0){
            break;
        }
        OutCode code_out = code1 > code0 ? code1 : code0;
        double x, y;
        if ((code_out & LEFT) != 0) //在左边
        {
            x = clip_min.x();
            if (x1 == x0)  y = y0;
            else y = (int)(y0 + (y1 - y0)*(min_x - x0) / (x1 - x0));
        }
        else if ((code_out & RIGHT) != 0) //在右边
        {
            x = clip_max.x();
            if (x1 == x0)  y = y0;
            else y = (int)(y0 + (y1 - y0)*(max_x - x0) / (x1 - x0));
        }
        else if ((code_out & BOTTOM) != 0) //在下边
        {
            y = clip_max.y();
            if (y1 == y0)  x = x0;
            else x = (int)(x0 + (x1 - x0)*(max_y - y0) / (y1 - y0));
        }
        else if ((code_out & TOP) != 0) //在上边
        {
            y = clip_min.y();
            if (y1 == y0)  x = x0;
            else x = (int)(x0 + (x1 - x0)*(min_y - y0) / (y1 - y0));
        }
        if (code_out == code0)
        {
            x0 = x;
            y0 = y;
            p0 = { x0, y0 };
        }
        else
        {
            x1 = x;
            y1 = y;
            p1 = { x1, y1 };
        }
        code0 = calc_OutCode(p0, clip_min, clip_max);
        code1 = calc_OutCode(p1, clip_min, clip_max);
    }
```

- 接下来就是最难的多边形裁剪算法，其核心思想是每次都只用窗口的某一条边切割多边形，这样4次切割之后我们的结果就出来了。ppt里用的是双循环，但是我不知道最外层循环k代表4个边的时候应该怎么表示现在是在用哪个边在裁剪，所以我比较笨的写了4遍内层循环，每次循环都要判断一下多边形边和窗口的关系，然后选择输出哪个点。其中的一次循环代码如下：

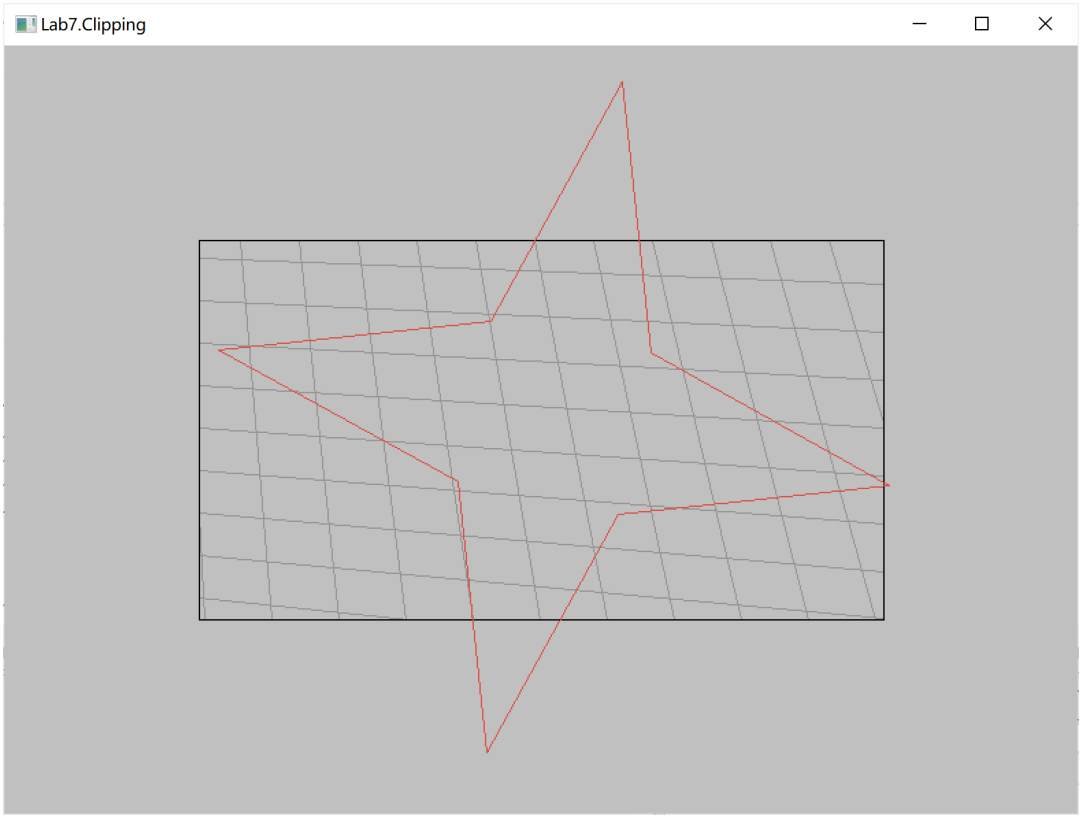
```
std::vector<Vector2> input_poly; //用左边分割
  for (int i = 0; i < output_poly.size(); i++)
  {     // 每一个顶点
        //下一个顶点为j = i + 1 or 0 if i == size()
```

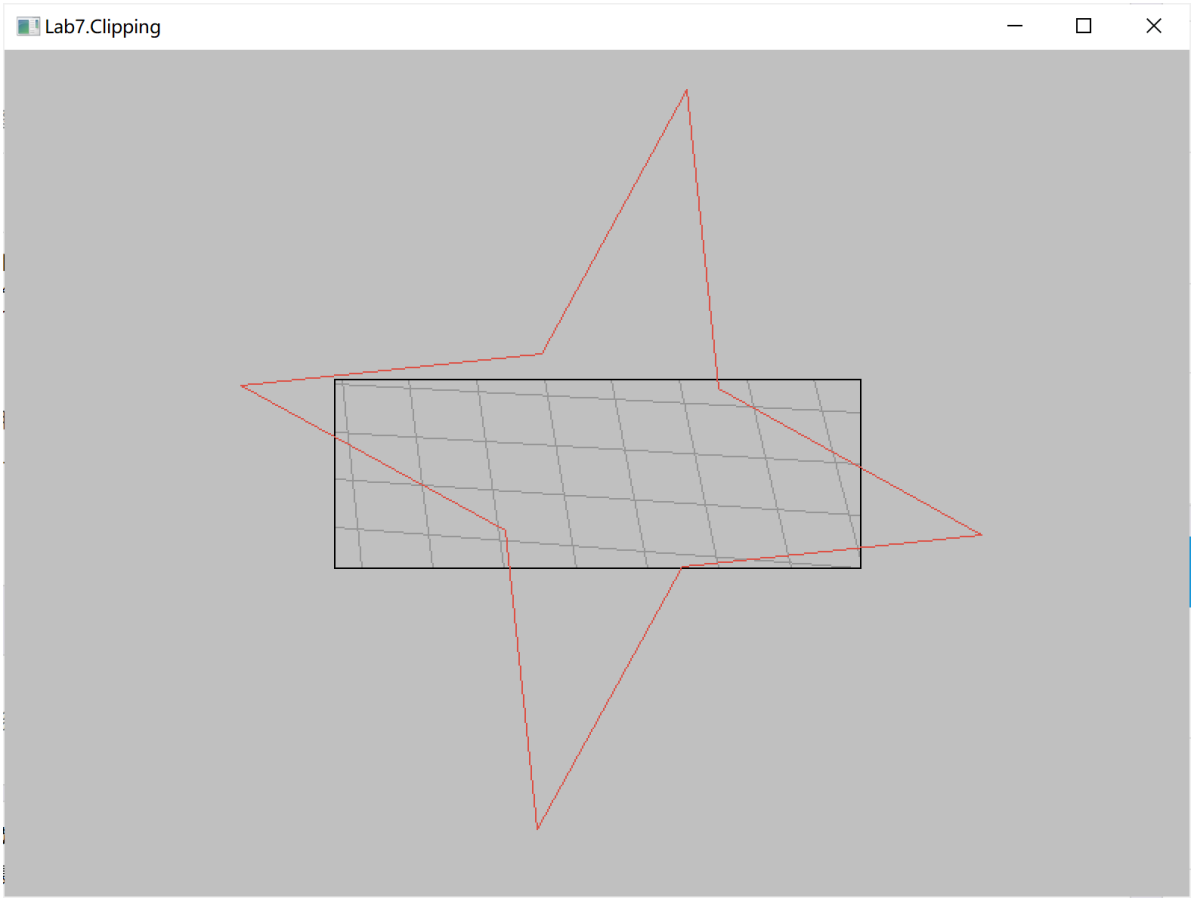
```
        //判断四种情况，并输出响应点到output
      int j;
      if (i == (output_poly.size()-1)) j = 0;
      else j = i + 1;
      Vector2 p0, p1;
      p0 = output_poly[i];
      p1 = output_poly[j];
      OutCode code0 = calc_OutCode(p0, clip_min, clip_max);
      OutCode code1 = calc_OutCode(p1, clip_min, clip_max);
      auto x0 = p0.x(), y0 = p0.y();
      auto x1 = p1.x(), y1 = p1.y();
      auto min_x = clip_min.x(), min_y = clip_min.y();
      auto max_x = clip_max.x(), max_y = clip_max.y();
      if (((code0 & LEFT) == 0) && ((code1 & LEFT) == 0)) //都在里面
      {
          input_poly.push_back(p1);
      }
      else if (((code0 & LEFT) != 0) && ((code1 & LEFT) == 0)) //→
      {
          double x, y;
          x = clip_min.x();
          y = (int)(y0 + (y1 - y0)*(min_x - x0) / (x1 - x0));
          Vector2 p;
          p = {x,y};
          input_poly.push_back(p);
          input_poly.push_back(p1);
      }
      else if (((code0 & LEFT) == 0) && ((code1 & LEFT) != 0)) //←
      {
          double x, y;
          x = clip_min.x();
          y = (int)(y0 + (y1 - y0)*(min_x - x0) / (x1 - x0));
          Vector2 p;
          p = {x,y};
          input_poly.push_back(p);
      }

  }
  output_poly = input_poly;
```

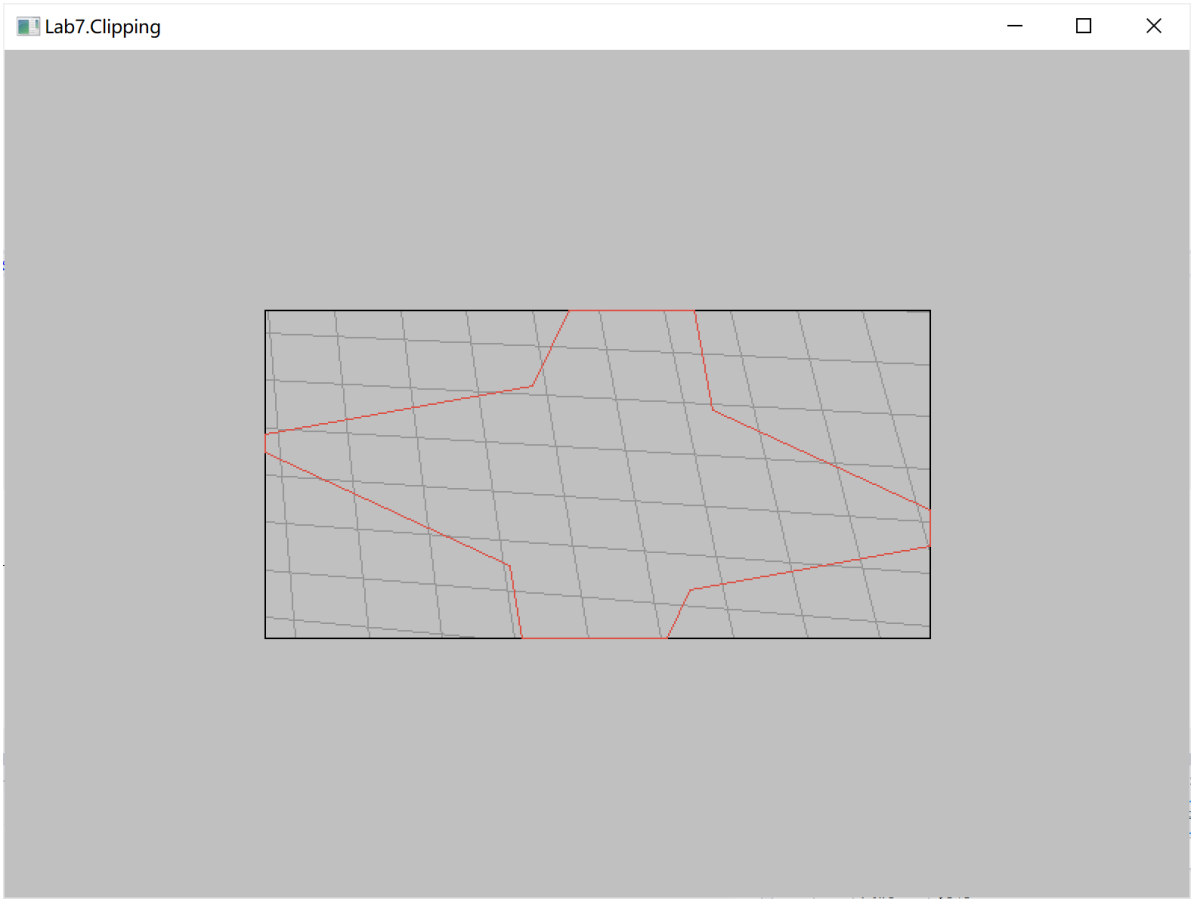
五、实验过程总结

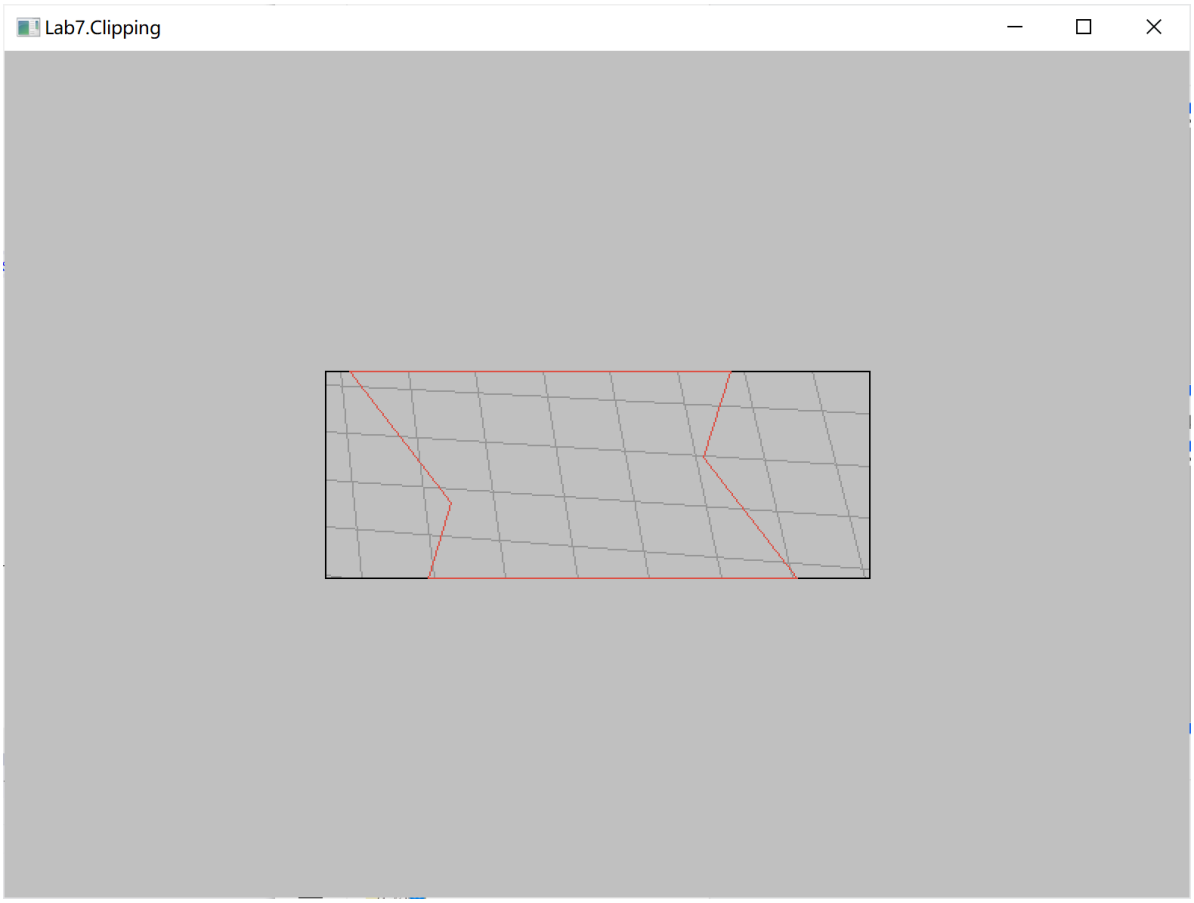
- 实现裁剪算法实现后的运行结果：





- 下面是多边形裁剪算法实现之后的运行结果：





- 总结：本次实验需要非常细心才能完成，我在写完代码之后遇到了很多bug，最终幸运的一一解决。我认为还有一个难点是熟悉助教已经编好的框架，中间有一些用法用错了导致出现错误也比较常见。