《计算机图形学》实验报告

《裁剪算法实验》

姓名 王红阳

学号 3019244233

专业 计算机科学与技术

班级 3班

天津大学智能与计算学部

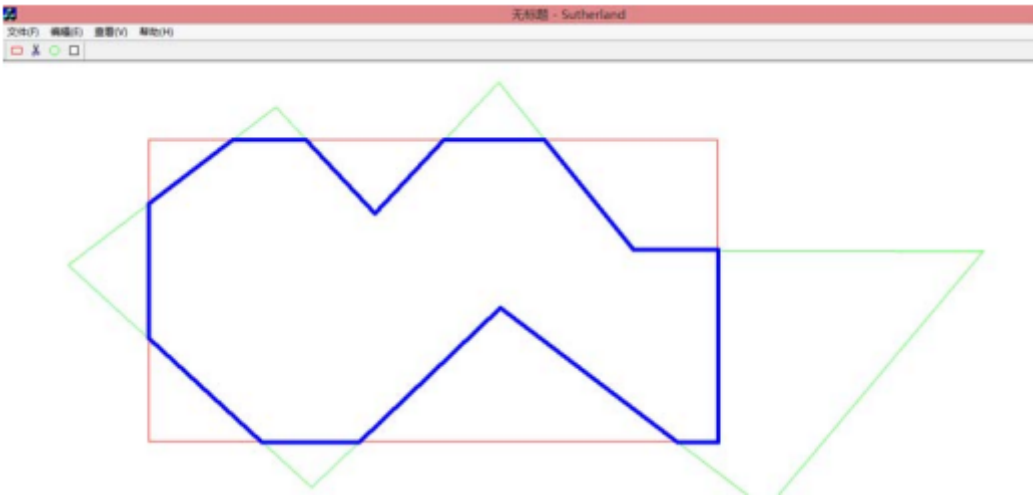
2021年 09月27 日

一、实验目的

- 用编程语言实现如何裁剪直线和裁剪多边形

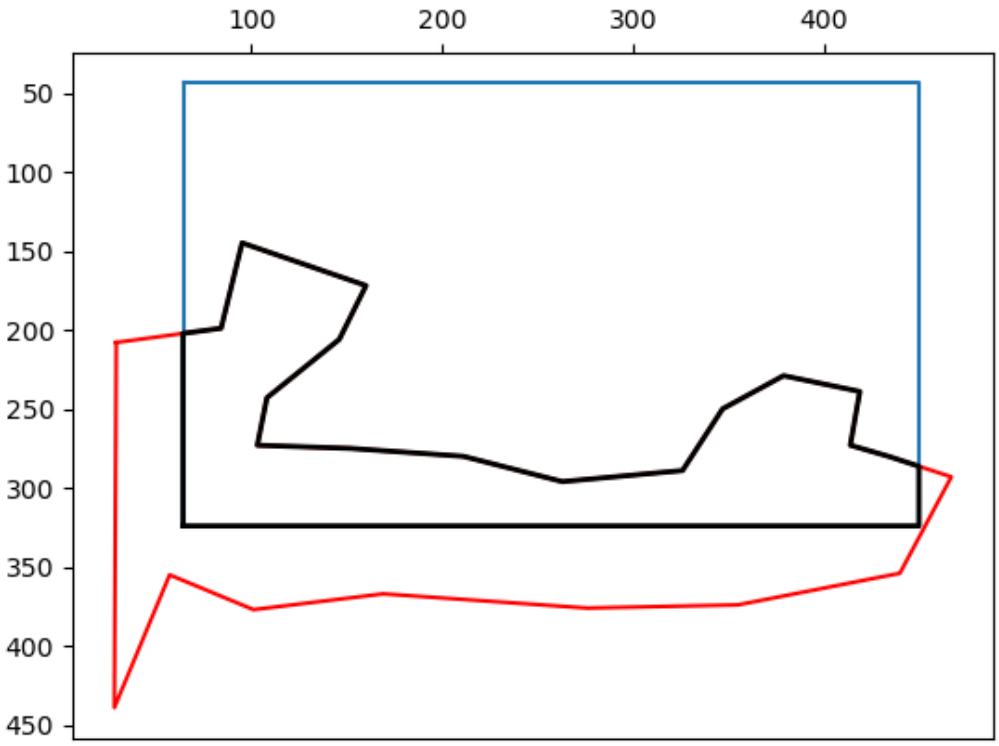
二、实验内容

- 内容:
  - 实现 Cohen-Sutherland 直线裁剪算法（选做）
  - 实现 Sutherland-Hodgman 多边形裁剪算法
- 要求：
  - 自定义裁剪窗口和待裁剪直线段(或多边形)；
  - 采用不同颜色突出显示裁剪结果，如下图所示

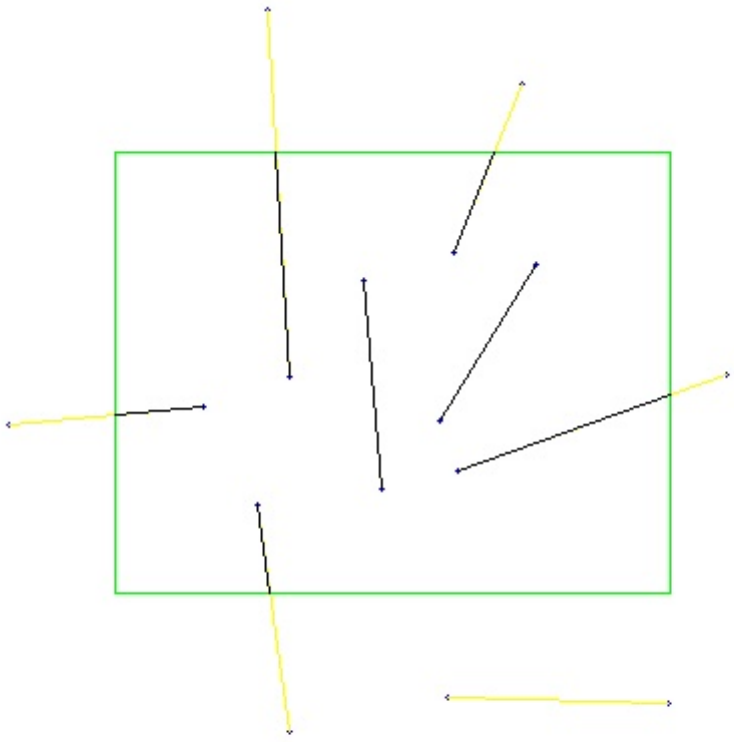


三、实验结果

使用Sutherland_Hodgman算法对多边形进行裁剪：



使用 Cohen-Sutherland 算法对多边形进行裁剪：



四、实验分析和总结

上次实验，我使用python中的matplotlib实现了画线，但是matplotlib并不能很好地自定义地画出多边形和多条直线。经过考察，我决定使用python中的opencv模块进行实现本次实验。

关于 Cohen-Sutherland 直线裁剪算法的实现

使用流程：首先运行代码，辉县创建一个白色画布，使用鼠标左键画矩形裁剪框（颜色为绿色），鼠标中键标注线段起点和终点，然后就会实时显示裁剪后的结果了（起点和终点为蓝色，原线段为黄色，裁剪后的线段为为黑色），之后按ESC退出，然后就会自动保存刚才的画布了

主要的算法实现部分在Cohen-Sutherland函数内，这个函数先对输入的线段起点和终点进行编码，再送入while循环内，直至将该线段裁剪好，然后调用opencv里的函数，进行画线

关于 Sutherland-Hodgman 多边形裁剪算法的实现

使用流程：使用鼠标左键画矩形裁剪框，鼠标中键标注多边形顶点，鼠标右键勾画多边形，之后按ESC退出，然后就会出现裁剪后的图形了

代码流程：使用opencv工具自定义矩形裁剪框和多边形待裁剪图形，然后将每一线段的一对顶点送入一组裁剪器(左、右、下、上）一个裁剪器完成一对顶点的处理后，该边裁剪后留下的坐标值送给下一个裁剪器。最终将裁剪完成的图形，使用matplotlib进行显示。

总结

通过本次实验，我更加深入地了解了如何使用python画出图形，如何使用opencv模块进行鼠标事件的监听和使用matplotlib模块画图，为后续进一步深入实现计算机图形学领域的经典算法奠定了基础。

五、源代码

Cohen-Sutherland.py:

```
import matplotlib.pyplot as plt
import numpy as np
import cv2

# 定义不同区域的编码
LEFT = 1
RIGHT = 2
BOTTOM = 4
TOP = 8

# 窗口的边界值 (暂未初始化)
x_left_window = -1
x_right_window = -1
y_bottom_window = -1
y_top_window = -1

# 矩形的边界
ix, iy = -1, -1  # 左下角
```

```
px, py = -1, -1  # 右上角

# 线段的起点，终点
x_end, y_end = 0, 0
x_start, y_start = 0, 0

# 当前线段是否画完
is_line_drawn = 1

drawing = False  # 鼠标按下为真
notdone = True
img = []


def draw(event, x, y, flags, param):
    """响应鼠标事件，用于画矩形，画线段的起点和终点，并进行线段的裁剪
    """

    # 定义全局变量
    global ix, iy, px, py
    global x_end, x_start, y_start, y_end
    global is_line_drawn, notdone, drawing
    global x_left_window, x_right_window, y_bottom_window, y_top_window

    # 鼠标左键画矩形
    if event == cv2.EVENT_LBUTTONDOWN and notdone == True:
        drawing = True
        ix, iy = x, y
    # 给矩形涂色
    elif event == cv2.EVENT_MOUSEMOVE and notdone == True:
        if drawing == True:
            # cv2.rectangle(img, (ix, iy), (px, py), (0, 0, 0), 0)  # 将刚刚拖拽的矩形涂黑
            # cv2.rectangle(img, (ix, iy), (x, y), (0, 255, 0), 0)
            px, py = x, y

    # 结束画矩形，以开始画线
    elif event == cv2.EVENT_LBUTTONUP and notdone == True:
        drawing = False
        # 矩形颜色为绿色
        cv2.rectangle(img, (ix, iy), (x, y), (0, 255, 0), 0)
        px, py = x, y
        notdone = False

        # 获取窗口大小
        x_left_window, y_bottom_window = ix, iy
        x_right_window, y_top_window = x, y

    # 画线
    elif event == cv2.EVENT_MBUTTONDOWN:
        # 画线段端点颜色为蓝色
        cv2.circle(img, (x, y), 1, (255, 0, 0))
```

```
        if is_line_drawn % 2 == 1:
            x_start = x
            y_start = y
            is_line_drawn += 1
        else:
            x_end = x
            y_end = y
            is_line_drawn += 1
            print("需裁剪的线段：  ", (x_start, y_start), (x_end, y_end))
            # 需裁减的线段颜色为黄色
            cv2.line(img, (x_start, y_start), (x_end, y_end), (0, 255,
255))
            # 窗口裁剪直线，并显示
            Cohen-Sutherland(x_start, y_start, x_end, y_end)


def encode(x, y):
    """给点(x,y) 进行编码
    """
    c = 0
    if x < x_left_window:
        c = c | LEFT
    if x > x_right_window:
        c = c | RIGHT
    if y < y_bottom_window:
        c = c | BOTTOM
    if y > y_top_window:
        c = c | TOP
    return c


def Cohen-Sutherland(x1, y1, x2, y2):
    """裁剪线段
    Args:
        (x1, y1) 线段起点
        (x2, y2) 线段终点
    """
    code1 = encode(x1, y1)
    code2 = encode(x2, y2)
    outcode = code1  # outcode是总在窗口外的那个端点
    x, y = 0, 0
    area = False  # 设置一个是否满足条件的区分标志
    while True:
        if (code2 | code1) == 0:
            area = True
            break
        if (code1 & code2) != 0:  # 简弃之
            break
        if code1 == 0:  # 开始求交点
            outcode = code2
        if (LEFT & outcode) != 0:  # 与窗口左边界相交
```

```
            x = x_left_window
            y = y1 + (y2 - y1) * (x_left_window - x1) / (x2 - x1)
        elif (RIGHT & outcode) != 0:
            x = x_right_window
            y = y1 + (y2 - y1) * (x_right_window - x1) / (x2 - x1)
        elif (BOTTOM & outcode) != 0:
            y = y_bottom_window
            x = x1 + (x2 - x1) * (y_bottom_window - y1) / (y2 - y1)
        elif (TOP & outcode) != 0:
            y = y_top_window
            x = x1 + (x2 - x1) * (y_top_window - y1) / (y2 - y1)
        x = int(x)  # 转换为整型
        y = int(y)
        if outcode == code1:
            x1 = x
            y1 = y
            code1 = encode(x, y)
        else:
            x2 = x
            y2 = y
            code2 = encode(x, y)
    if area == True:  # 若满足条件即可划线
        print("裁剪后的边是：  ", (x1, y1), (x2, y2))
        # 裁剪后的边的颜色是黑色
        cv2.line(img, (x1, y1), (x2, y2), (0, 0, 0))
    return


def main():
    # 定义全局变量
    global img

    # 生成一张图片，相当于创建画布，颜色为白色
    img = np.full((512,512,3),255, np.uint8)

    # 直接显示窗口
    cv2.namedWindow('Cohen-Sutherland', cv2.WINDOW_NORMAL)

    # 处理事件，用于画矩形，画线段的起点和终点，并进行线段的裁剪
    cv2.setMouseCallback('Cohen-Sutherland', draw)

    while (1):
        # 显示图像
        cv2.imshow('Cohen-Sutherland', img)

        # 接收键盘按键，如果按‘Esc'键，则退出
        k = cv2.waitKey(1) & 0xFF
        if k == ord('q'):
            break
        elif k == 27:
            break
```

```
    # 保存最后的图片
    cv2.imwrite('./clip_line/Cohen-Sutherland/out.jpg', img)

    # 清除窗口
    cv2.destroyAllWindows()


if __name__ == '__main__':
    main()
```

Sutherland_Hodgman.py:

```
import matplotlib.pyplot as plt
import numpy as np
import cv2

drawing = False  # 鼠标按下为真
notdone = True
ix, iy = -1, -1  # 左下角
px, py = -1, -1  # 右上角
l = []  # 多边形顶点的列表


def pointInRec(p):
    """判断点P是否在区域内
    """
    if ix <= p[0] <= px and iy <= p[1] <= py:
        return True
    return False


def draw_rectangle(event, x, y, flags, param):
    """响应鼠标事件，画矩形
    """
    global ix, iy, drawing, px, py, l, notdone

    if event == cv2.EVENT_LBUTTONDOWN and notdone == True:  # 鼠标左键画矩形
        drawing = True
        ix, iy = x, y
    elif event == cv2.EVENT_MOUSEMOVE and notdone == True:
        if drawing == True:
            cv2.rectangle(img, (ix, iy), (px, py), (0, 0, 0), 0)  # 将刚刚
拖拽的矩形涂黑
            cv2.rectangle(img, (ix, iy), (x, y), (0, 255, 0), 0)
            px, py = x, y
    elif event == cv2.EVENT_LBUTTONUP and notdone == True:
        drawing = False
        cv2.rectangle(img, (ix, iy), (x, y), (0, 255, 0), 0)
        px, py = x, y
```

```
            notdone = False
        elif event == cv2.EVENT_MBUTTONDOWN:  # 鼠标中键标记多边形顶点
            print((x, y))
            l.append([x, y])
            cv2.circle(img, (x, y), 1, (255, 255, 255))
        elif event == cv2.EVENT_RBUTTONDOWN:  # 鼠标右键生成多边形
            pts = np.array(l, np.int32)
            pts = pts.reshape((-1, 1, 2))
            cv2.polylines(img, [pts], True, (255, 255, 255))


def line_intersection(line1, line2):
    """计算两条线的交点
    """
    xdiff = (line1[0][0] - line1[1][0], line2[0][0] - line2[1][0])
    ydiff = (line1[0][1] - line1[1][1], line2[0][1] - line2[1][1])

    def det(a, b):
        return a[0] * b[1] - a[1] * b[0]

    div = det(xdiff, ydiff)
    if div == 0:
        return 99999, 99999

    d = (det(*line1), det(*line2))
    x = det(d, xdiff) / div
    y = det(d, ydiff) / div
    return x, y


def fun(p1, p2):
    """求出两个点的delta y,delta  x, 叉积
    """
    x1 = p1[0]
    y1 = p1[1]
    x2 = p2[0]
    y2 = p2[1]
    a = y2 - y1
    b = x1 - x2
    c = x2 * y1 - x1 * y2
    return a, b, c


def clip_left(pointList):
    global ix, iy, px, py
    newList = []
    for i in range(len(pointList)):
        p1 = pointList[i - 1]
        p2 = pointList[i]
        if p1[0] < ix and p2[0] > ix:  # 由外到内
            a, b, c = fun(p1, p2)
            y = (-c - a * ix) / b
```

```
            intersection = [ix, y]
            newList.append(intersection)
            newList.append(p2)
        elif p1[0] > ix and p2[0] > ix:  #由内到内
            newList.append(p2)
        elif p1[0] > ix and p2[0] < ix:  #由内到外
            a, b, c = fun(p1, p2)
            y = (-c - a * ix) / b
            intersection = [ix, y]
            newList.append(intersection)
    return newList


def clip_bottom(pointList):
    pointList = clip_left(pointList)
    global ix, iy, px, py
    newList = []
    for i in range(len(pointList)):
        p1 = pointList[i - 1]
        p2 = pointList[i]
        if p1[1] < iy and p2[1] > iy:  # 由外到内
            a, b, c = fun(p1, p2)
            x = (-c - b * iy) / a
            intersection = [x, iy]
            newList.append(intersection)
            newList.append(p2)
        elif p1[1] > iy and p2[1] > iy:  #由内到内
            newList.append(p2)
        elif p1[1] > iy and p2[1] < iy:  #由内到外
            a, b, c = fun(p1, p2)
            x = (-c - b * iy) / a
            intersection = [x, iy]
            newList.append(intersection)
    return newList


def clip_right(pointList):
    pointList = clip_bottom(pointList)
    global ix, iy, px, py
    newList = []
    for i in range(len(pointList)):
        p1 = pointList[i - 1]
        p2 = pointList[i]
        if p1[0] > px and p2[0] < px:  # 由外到内
            a, b, c = fun(p1, p2)
            y = (-c - a * px) / b
            intersection = [px, y]
            newList.append(intersection)
            newList.append(p2)
        elif p1[0] < px and p2[0] < px:  #由内到内
            newList.append(p2)
        elif p1[0] < px and p2[0] > px:  #由内到外
```

```
            a, b, c = fun(p1, p2)
            y = (-c - a * px) / b
            intersection = [px, y]
            newList.append(intersection)
    return newList


def clip_top(pointList):
    pointList = clip_right(pointList)
    global ix, iy, px, py
    newList = []
    for i in range(len(pointList)):
        p1 = pointList[i - 1]
        p2 = pointList[i]
        if p1[1] > py and p2[1] < py:  # 由外到内
            a, b, c = fun(p1, p2)
            x = (-c - b * py) / a
            intersection = [x, py]
            newList.append(intersection)
            newList.append(p2)
        elif p1[1] < py and p2[1] < py:  #由内到内
            newList.append(p2)
        elif p1[1] < py and p2[1] > py:  #由内到外
            a, b, c = fun(p1, p2)
            x = (-c - b * py) / a
            intersection = [x, py]
            newList.append(intersection)
    return newList


if __name__ == '__main__':
    img = np.zeros((512, 512, 3), np.uint8)
    cv2.namedWindow('Sutherland-hodgman')
    cv2.setMouseCallback('Sutherland-hodgman', draw_rectangle)
    while (1):
        cv2.imshow('Sutherland-hodgman', img)
        k = cv2.waitKey(1) & 0xFF
        if k == ord('q'):
            break
        elif k == 27:
            break
    cv2.destroyAllWindows()

    recList = [[ix, iy], [px, iy], [px, py], [ix, py], [ix, iy]]  # 裁剪窗
口
    flag = []
    pointList = l
    pointList.append(l[0])
    newList = clip_top(pointList)
    newList.append(newList[0])
    x = [x[0] for x in recList]
    y = [x[1] for x in recList]
```

```
    x1 = [x[0] for x in pointList]
    y1 = [x[1] for x in pointList]
    x2 = [x[0] for x in newList]
    y2 = [x[1] for x in newList]

    # 把坐标系原点设置为左上角，使得plt与cv2保持一致
    ax = plt.gca()  # 获取到当前坐标轴信息
    ax.xaxis.set_ticks_position('top')  # 将X坐标轴移到上面
    ax.invert_yaxis()  # 反转Y坐标轴
    plt.plot(x, y)
    plt.plot(x1, y1, color='red')
    plt.plot(x2, y2, color='black', linewidth='2')
    plt.show()
```