

河海大学



计算机图形学课程报告

专 业 计算机科学与技术

学 号 1961010516

姓 名 秦晓

2022 年 4 月

目录

一、实验目的	3
二、实验内容	3
1. 系统总体算法流程图	
2. 数据结构定义	
3. 多边形的扫描填充算法实现（能处理自交情况）	
4. 点和多边形的撤销和重置	
5. 旋转和平移功能	
6. 颜色设置功能	
7. 存储多边形文件功能	
8. 读取多边形文件功能	
9. AET表打印控制功能	
三、算法实现	4
1. 菜单交互	
2. 填充多边形绘制	
3. 绘制的撤销和重置	
4. 旋转	
5. 平移	
6. 若干多边形的文件读写	
四、实验结果	10
五、总结与体会	13
六、关键代码附录	14

一、实验目的

- 理解多边形的扫描转换原理、方法
- 掌握橡皮筋算法和扫描线填充算法。
- 理解多边形填充算法的意义。
- 进一步掌握在 VC 集成环境中实现图形算法的方法与过程。

二、实验内容

必要功能：

- 要 OpenGL 实现
- 通过橡皮筋交互输入不同颜色、大小的多边形
- 清屏重置多边形
- 多边形扫描算法中的顶点处理以每条边减去一个像素方法处理
- 要类（或模版类）来表示数据结构
- 多文档组织，至少要用头（.h）文件表示数据结构

选做功能：

- 填充模式自由设定，如等间隔、斜扫描线填充
- 通过菜单交互
- 自相交多边形、多个多边形的扫描填充
- 通过文件存储和读出已经交互输入的多边形
- 绘制操作的撤销和全部删除

三、算法实现

1. 系统总体算法流程图

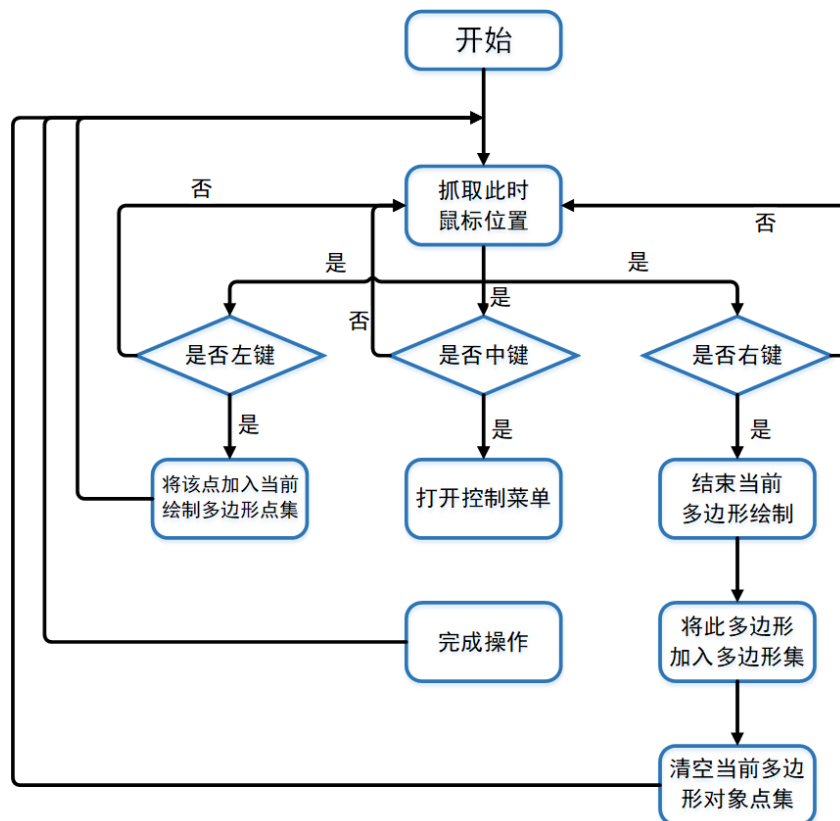


图 1

- ① 初始化多边形点集 `vector<point> p`; 多边形类向量 `vector<polygon> s`;
- ② 设置鼠标监控事件 `myMouse`，当鼠标左键点击时，生成一个点类变量并将此时的坐标存入点类中，再存入多边形点集向量 `p` 中；若鼠标点击右键，则是将正在绘制的多边形封闭，将 `select` 选项置 `true`；若鼠标点击中键，展开选项菜单。
- ③ 填充阶段时，先判断多边形类向量是否为空，不为空则对多边形向量循环，向量中的每个元素即为此前绘制的一个个多边形，再对多边形中的每个点进行循环，开始画多边形，首先确定前一个点和后一个点，对两点进行连线，遍历循环，将整一个多边形绘制出，最后将所有多边形都画出来。
- ④ 判断右键是否被点下，若点集不为空，就将当前绘制的多边形的最后一点和第一个点连接起来，当前所有点顺次连接完毕，再将封闭的多边形保存到多边形类向量集中。
- ⑤ 用 `glutAttachMenu` 监听由鼠标中键发出的菜单命令，完成指定操作。

2. 数据结构定义

(1) 头文件 edge.h 中 Edge 类

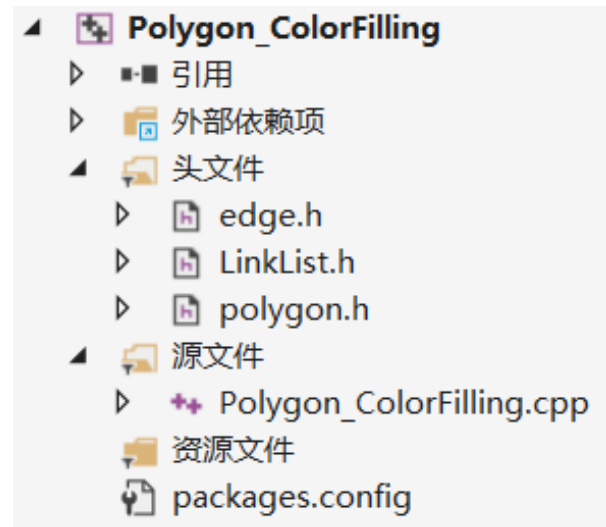
```
class Edge
{
public:
    int ymax;
    float x;
    float dx;
    Edge* next;
};
```

ymax 为边最高点的纵坐标

x 为边最低点的横坐标

dx 为边斜率的倒数

Edge* next 指向下一条边



(2) 头文件 polygon.h 中 point 类和 polygon 类

```
class point //点类，存储了一个点的两坐标值
{
public:
    int x;
    int y;
};
```

```
class polygon //多边形类，存了一个多边形
{
public:
    vector<point> p; //多边形的顶点
    int color;
};
```

point 类存储鼠标选取的多边形点位置；

polygon 类存储一个完整的多边形和其颜色, 多边形信息由点集给出。

3. 多边形的扫描填充算法（能处理自相交情况）

①算法流程图如下（图 2）：

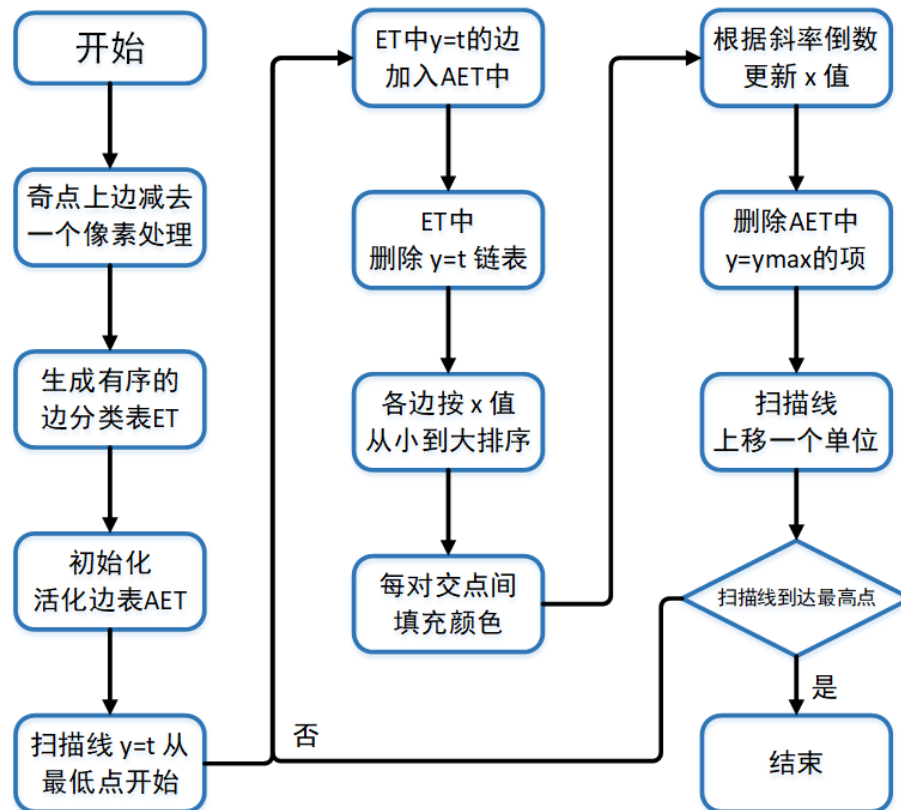


图 2

②以自己的实验为例写出活性边表的具体变化过程：

为了打印 AET 表的全部信息，并不占用过大篇幅，实验绘制较小的自相交多边形，见下图（图 3），为 27×27 像素的截图后适当放大的效果。



图 3

对于图 3 的自相交多边形，AET 打印如下：

结构：扫描线 $y=t$ $dx \mid x \mid ymax \longrightarrow dx \mid x \mid ymax \longrightarrow \dots\dots\dots$

$y=335$	0.33333	633	347	\longrightarrow	1.75	633	339
$y=336$	0.333333	633.333	347	\longrightarrow	1.75	634.75	339
$y=337$	0.333333	633.667	347	\longrightarrow	1.75	636.5	339
$y=338$	0.333333	634	347	\longrightarrow	1.75	638.25	339
$y=339$	0.333333	634.333	347	\longrightarrow	1.75	640	339
$y=340$	0.333333	634.667	347	\longrightarrow	-1.6	638.4	344
$y=341$	0.333333	635	347	\longrightarrow	-1.6	636.8	344
$y=342$	0.333333	635.333	347	\longrightarrow	-1.6	635.2	344
$y=343$	-1.6	633.6	344	\longrightarrow	0.333333	635.667	347
$y=344$	-1.6	632	344	\longrightarrow	0.333333	636	347
$y=345$	1.66667	633.667	347	\longrightarrow	0.333333	636.333	347
$y=346$	1.66667	635.333	347	\longrightarrow	0.333333	636.666	347

易见，在 $y=342$ 处，两条边的 x 值分别是 635.333 和 635.2，在物理层面光栅无法达到 0.133 的精度，在显示器上表现为多边形自相交，此时，算法更新了 AET 表，重新排序使得多边形填充效果无误，否则，错误的填充效果如下图（图 4）相似。

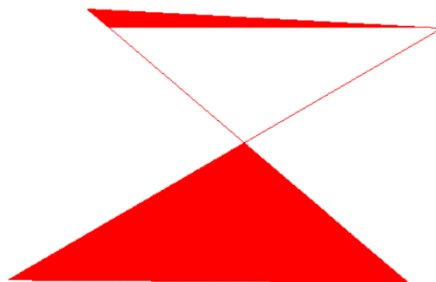


图 4

4. 点和多边形的撤销和全部删除功能

系统提供撤销上一个绘制顶点、重置当前正在绘制的多边形、撤销上一个绘制的多边形、清空白板删除所有多边形

算法流程图如图 5 所示：

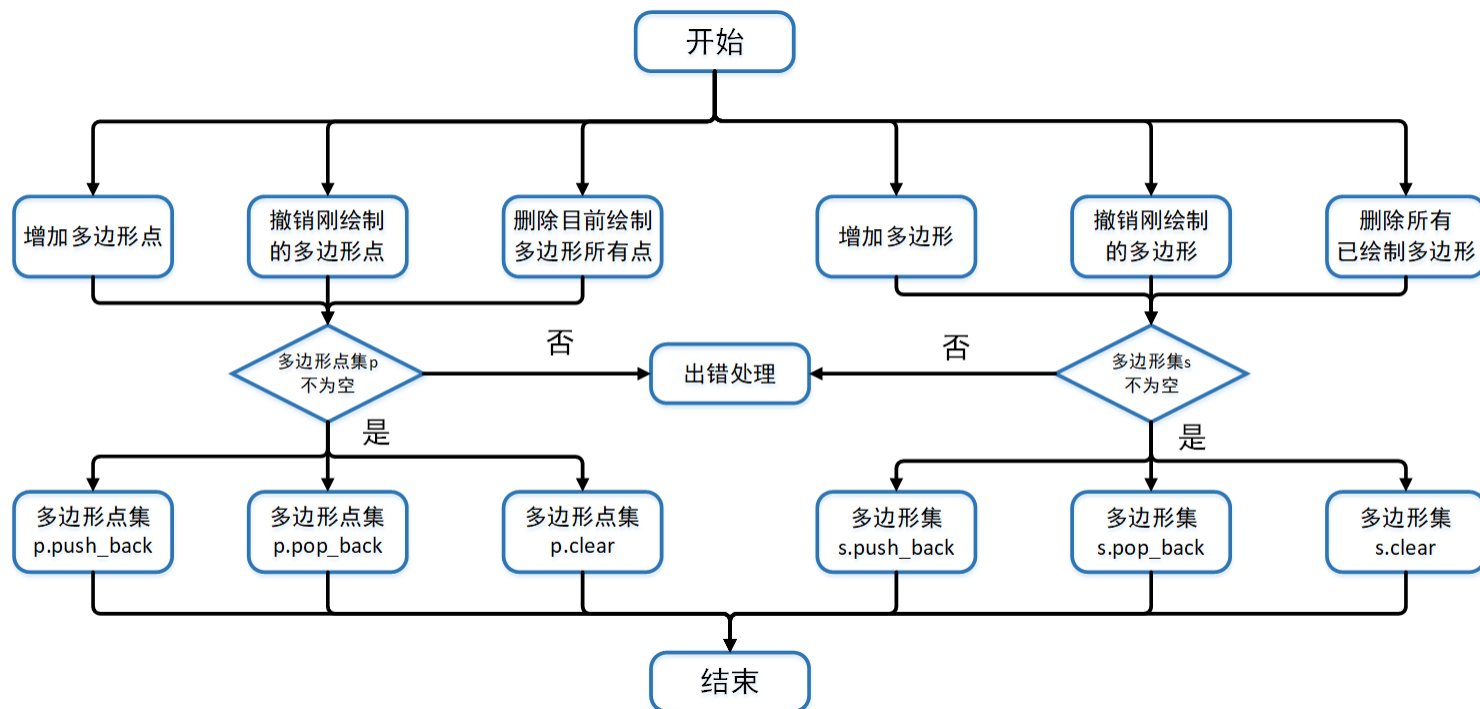


图 5

5. 旋转和平移功能

使用glut库提供的glRotatef()；glTranslated()函数进行旋转和平移操作，通过菜单选项，用bool变量来控制是否平移，用旋转角来控制是否旋转和旋转方向。

6. 颜色设置功能

通过菜单交互选择颜色模式，在菜单中点击颜色更换，使用户的颜色变量值加一，对颜色变量取模4，得到4种颜色之一，用switch语句选择glColor3f参数设置颜色。

7. 存储多边形文件功能

先读取多边形集的size，保存到文件第一行，然后将多边形集中的元素，按照“多边形颜色模式-多边形顶点的个数-多边形点坐标”顺序依次写入文件。

8. 读取多边形文件功能

先读取文件第一行，作为外层循环变量控制读取多边形个数，然后读取下一个多边形颜色模式，接着读取多边形顶点的个数作为内层循环变量，在循环体内读取多边形点集和颜色模式赋值给多边形对象，然后将多边形对象加入到多边形集合，直到循环结束。

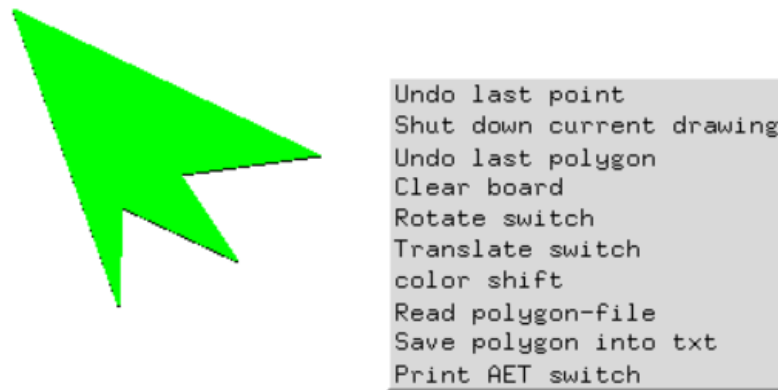
9. AET表打印控制功能

该功能负责控制系统是否打印AET表，因为打印AET表的操作很耗时间，在调试时打印比较方便修复BUG，此功能仅需菜单修改一个bool变量控制if语句。

四、实验结果

1. 菜单交互

UI 界面提供由鼠标中键触发的菜单选项



2. 填充的多边形绘制

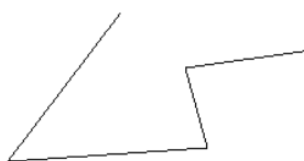


3. 绘制的撤销和重置（左边为原图，右边为操作后图）

① 撤销多边形顶点（Undo Last Point）

多边形扫描填充算法

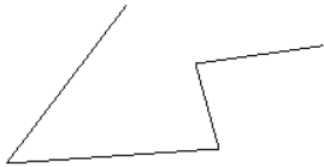
多边形扫描填充算法



②重置当前绘制的多边形 (Shut down Current Drawing)

多边形扫描填充算法

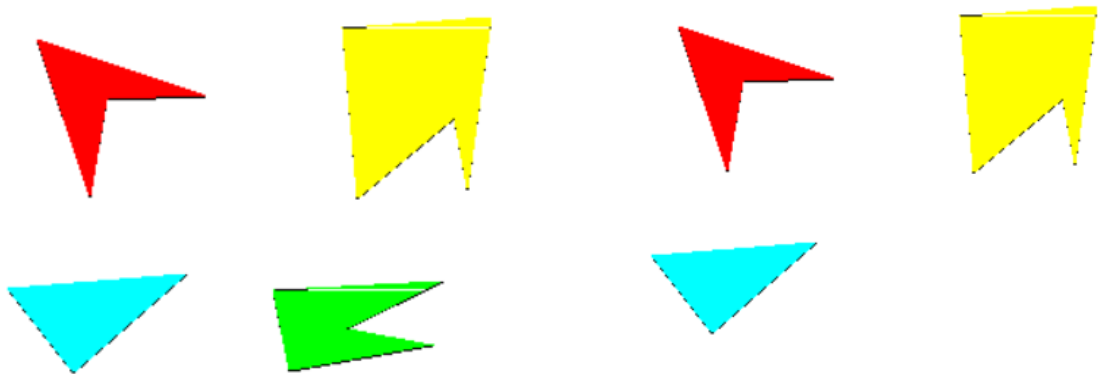
多边形扫描填充算法



③撤销上一个绘制的多边形 (Undo Last Polygon)

多边形扫描填充算法

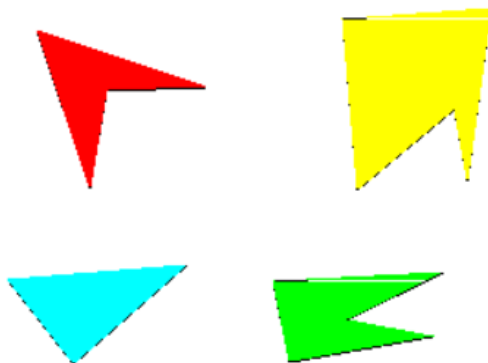
多边形扫描填充算法



④ 删除所有多边形，清空画布 (Clear Board)

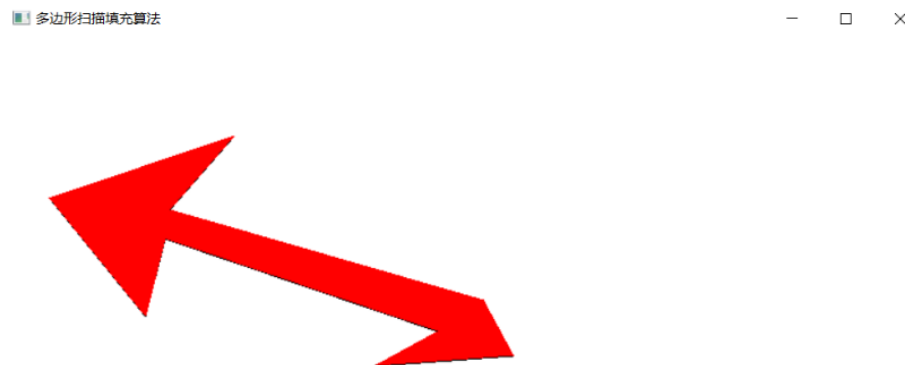
多边形扫描填充算法

多边形扫描填充算法

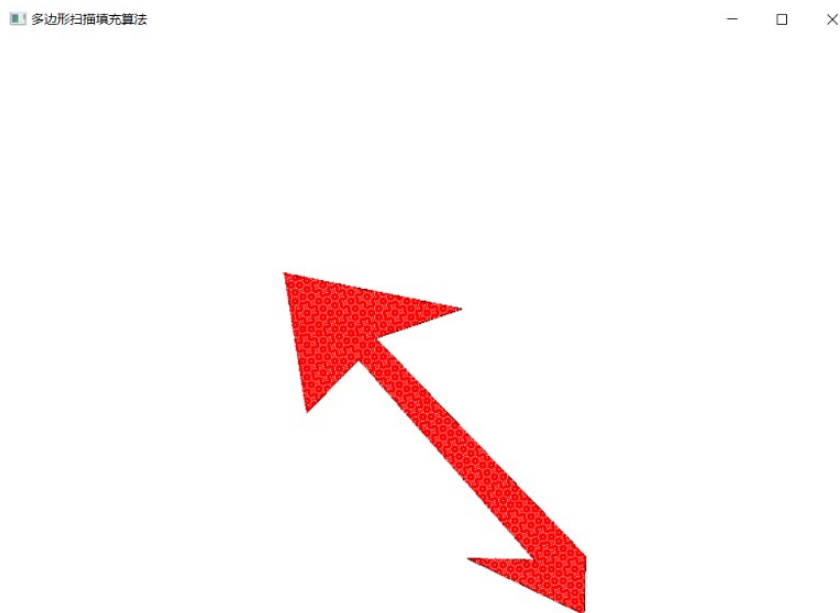


4. 旋转，每次右键绕 (0, 0) 旋转 15° （此处旋转了 45° ）

旋转前：



旋转后：

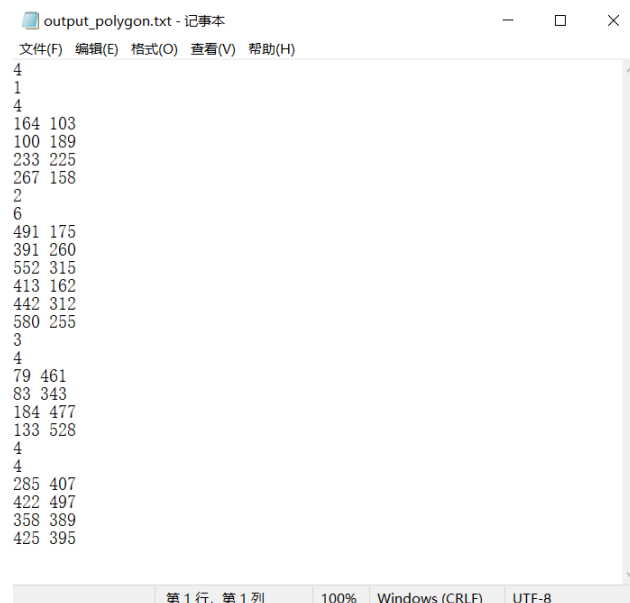


5. 平移，每次右键沿 (40, 40) 的向量平移（此处平移一次）



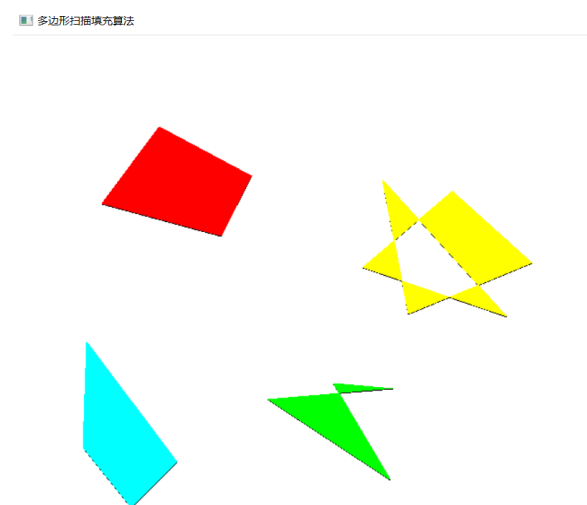
6. 若干多边形的文件读写

写入后的多边形 txt 文件：



```
4
1
4
164 103
100 189
233 225
267 158
2
6
491 175
391 260
552 315
413 162
442 312
580 255
3
4
79 461
83 343
184 477
133 528
4
4
285 407
422 497
358 389
425 395
```

读取多边形文件后的画板：



五、总结与体会

通过本次课程实践，我对图形学有了更深刻具体的认识，尤其对橡皮筋算法和扫描填充算法有了更加深入的理解，能够熟练使用 OpenGL 的部分库函数。在项目中也遇到一些问题，比如对 C++ 不够熟悉边编程边查语法，对奇点的判断遇到困难等。

在初学课程时，有些算法没有理解透彻，本次实验促使我将书上的知识应用到实践中来，将一些算法编成程序执行，在实践中又体会到乐趣，将书本上的知识融会贯通。

项目源码

```
#ifndef DATA_H_1
#define DATA_H_1
#include<vector>
using namespace std;
```

```
class Edge
{
public:
    int ymax;
    float x;
    float dx;
    Edge* next;
};
#endif
```

```
#ifndef DATA_H_
#define DATA_H_
#include<vector>
using namespace std;
class point //点类，存储了一个点的两坐标值
{
public:
    int x;
    int y;
};

class polygon //多边形类，存了一个多边形
{
public:
    vector<point> p; //多边形的顶点
    int color;
};

#endif
```

```

#define _CRT_SECURE_NO_WARNINGS
// PSFA.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。
// Polygon scanning and filling algorithm
// 多边形扫描填充算法

#include <iostream>
#include<gl/glut.h>
#include"polygon.h"
#include <string>
#include <fstream>
#include"edge.h"

static const int screenwidth = 800; //自定义窗口宽度
static const int screenheight = 600; //自定义窗口高度

int move_x, move_y; //鼠标当前坐标值，在鼠标移动动态画线时使用
bool select = false; //多边形封闭状态判断变量，当为 true 时即按下鼠标右键，将
多边形封闭
bool flash = false;
bool Transform = false;
int angle = 0;
bool translate = false;
bool readtxt = false;
bool writetxt = false;
bool AET_printer = false;
int now_color = 1;
bool ALLOW_printAET = false;

vector<point> p; //多边形点集向量
vector<polygon> s; //多边形类向量，用来保存已经画完的多边形

//边表
Edge* ET[screenheight];
//活性边表
Edge* AET;

void init()
{
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, screenwidth, 0.0, screenheight);
}

void sortAET(Edge *q)

```

```

{
    while (q->next)
    {
        Edge* p = q;
        while (p->next && p->next->next)
        {
            if (p->next->x > p->next->next->x)
            {
                int nowymax;
                float nowx;
                float nowdx;
                nowymax = p->next->ymax;
                nowx = p->next->x;
                nowdx = p->next->dx;
                p->next->ymax = p->next->next->ymax;
                p->next->x = p->next->next->x;
                p->next->dx = p->next->next->dx;
                p->next->next->ymax = nowymax;
                p->next->next->x = nowx;
                p->next->next->dx = nowdx;
            }
            else
            {
                p = p->next;
            }
        }
        q = q->next;
    }
}

```

// 多边形填充算法

```

void mypaint(vector<point> mypoints, int color)
{
    int max_Y = 0;
    for (int i = 0; i < mypoints.size(); i++)
    {
        if (mypoints[i].y > max_Y)
        {
            max_Y = mypoints[i].y;
        }
    }
    // 初始化 ET
    Edge* ET[screenheight];
    for (int i = 0; i < max_Y; i++)

```



```

{
    ET[i] = new Edge();
    ET[i]->next = nullptr;
}
// 初始化 AET
AET = new Edge();
AET->next = nullptr;

//glClear(GL_COLOR_BUFFER_BIT);
glColor3f(0.0, 0.0, 0.0);

glBegin(GL_POINTS);

// 建立边表 ET 存储处理过非极值点的边
for (int i = 0; i < mypoints.size(); i++)
{
    // 当前边的前一条边的起点
    int x0 = mypoints[(i - 1 + mypoints.size()) %
mypoints.size()].x;
    int y0 = mypoints[(i - 1 + mypoints.size()) %
mypoints.size()].y;

    // 当前边的起点 前一条边的终点
    int x1 = mypoints[i].x;
    int y1 = mypoints[i].y;

    // 当前边的终点 后一条边的起点
    int x2 = mypoints[(i + 1) % mypoints.size()].x;
    int y2 = mypoints[(i + 1) % mypoints.size()].y;

    // 当前边的后一条边的终点
    int x3 = mypoints[(i + 2) % mypoints.size()].x;
    int y3 = mypoints[(i + 2) % mypoints.size()].y;

    // 当前边是水平线，不用填充，进入下一条边
    if (y1 == y2)
        continue;

    //分别计算下端点 y 坐标、上端点 y 坐标、下端点 x 坐标和斜率倒数
    int ymin = y1 < y2 ? y1 : y2;
    int ymax = y1 > y2 ? y1 : y2;
    float x = y1 < y2 ? x1 : x2;
    float dx = (x1 - x2) * 1.0f / (y1 - y2);

```

```

int nowy;
if (y1 < y2)
{
    nowy = y1;
    ymax = y2;
}
else
{
    nowy = y2;
    ymax = y1;
}

```

//奇点特殊处理，若点 2->1->0 的 y 坐标单调递减则 y1 为奇点，若点 1->2->3 的 y 坐标单调递减则 y2 为奇点

```

if (((y0 < y1) && (y1 < y2)) || ((y2 < y1) && (y3 < y2)))
{
    nowy++;
    x += dx;
}

```

```

// 将该边插入边表 ET
Edge* p = new Edge();
p->ymax = ymax;
p->x = x;
p->dx = dx;
p->next = ET[nowy]->next;
ET[nowy]->next = p;
}

```

//扫描线从下往上扫描，每轮扫描 y 加 1

```

for (int i = 0; i < max_Y; i++)
{

```

//取出 ET 中当前扫描行的所有边并按 x 的递增顺序（若 x 相等则按 dx 的递增顺序）插入 AET

```

while (ET[i]->next)
{
    //取出 ET 中当前扫描行表头位置的边
    Edge* pInsert = ET[i]->next;
    Edge* p = AET;
    //在 AET 中搜索合适的插入位置
    while (p->next)
    {

```

```

        if (pInsert->x > p->next->x)
        {
            p = p->next;
            continue;
        }
        if (pInsert->x == p->next->x && pInsert->dx >
p->next->dx) // 等于的时候按斜率排
        {
            p = p->next;
            continue;
        }
        //找到位置
        break;
    }
    //将pInsert 从 ET 中删除, 并插入 AET 的当前位置
    ET[i]->next = pInsert->next;
    pInsert->next = p->next;
    p->next = pInsert;
}

// AET 中的边两两配对并填色
Edge* p = AET;
while (p->next && p->next->next)
{
    for (int x = p->next->x; x < p->next->next->x; x++)
    {
        switch (color)
        {
            case 1:
                glColor3f(1.0, 0.0, 0.0);
                break;
            case 2:
                glColor3f(1.0, 1.0, 0.0);
                break;
            case 3:
                glColor3f(0.0, 1.0, 1.0);
                break;
            case 4:
                glColor3f(0.0, 1.0, 0.0);
                break;
            case 5:
                glColor3f(0.0, 0.0, 1.0);
                break;
        }
    }
}

```

```

        //随机闪烁颜色
        if (flash)
            glColor3f(rand() % 2, rand() % 2, rand() % 2);
        glVertex2i(x, i);
    }
    // 打印活性边表信息
    if (ALLOW_printAET)
        if (AET_printer)
        {
            cout << i << endl;
            cout << p->next->dx << " | " << p->next->x << " | " << p->next->ymin << endl;
            cout << p->next->ymax << endl;
            cout << p->next->next->dx << " | " << p->next->next->x << " | " << p->next->next->ymin << endl;
            cout << p->next->next->ymax << endl;
        }
        p = p->next->next; // 跳到间隔一个的交点
    }

    //删除 AET 中满足 y=ymin 的边
    p = AET;
    while (p->next)
    {
        if (p->next->ymin == i)
        {
            Edge* pDelete = p->next;
            p->next = pDelete->next;
            pDelete->next = nullptr;
            delete pDelete;
        }
        else
        {
            p = p->next;
        }
    }

    // 更新 AET 中边的 x 值
    p = AET;
    while (p->next)
    {
        p->next->x += p->next->dx;
        p = p->next;
    }

    // 重新排序 AET 中的边, 按从小到大

```

```

        Edge* q = AET;
        sortAET(q);
    }

    glEnd();
}

void savepolygon()
{
    ofstream fout("C:\\Users\\52954\\Desktop\\output_polygon.txt");
    //创建一个 data.txt 的文件
    fout << s.size() << endl; // 多边形个数
    for (int i = 0; i < s.size(); i++) {
        fout << s[i].color << endl; //多边形颜色
        fout << s[i].p.size() << endl;
        for (int j = 0; j < s[i].p.size(); j++) {
            fout << s[i].p[j].x << " " << screenheight - s[i].p[j].y <<
endl; //多边形点
        }
    }
    fout.close(); //关闭文件
    writetxt = !writetxt;
}
// C:\\Users\\52954\\Desktop\\output_polygon.txt
void my_readtxt()
{
    int pcnt = 0;
    glutPostRedisplay(); //重绘窗口
    FILE* fp; //文件指针
    //char filename[100] =
"C:\\Users\\52954\\Desktop\\my_polygon.txt";
    char filename[100] =
"C:\\Users\\52954\\Desktop\\output_polygon.txt";
    if ((fopen(filename, "r")) == NULL)
    {
        /*二进制只读打开文件*/
        printf("fail to open file\n");
        exit(1);
    }
    fp = fopen(filename, "r");
    fscanf_s(fp, "%d", &(pcnt));
    for (int j = 0; j < pcnt; j++)
    {
        int lines;
        fscanf_s(fp, "%d", &(now_color));
    }
}

```

```

fscanf_s(fp, "%d", &(lines));
for (int i = 0; i < lines; i++)
{
    int xx, yy;
    fscanf_s(fp, "%d", &(xx)); //输出数据到数组
    fscanf_s(fp, "%d", &(yy));
    point fv;
    fv.x = xx;
    fv.y = screenheight - yy;
    p.push_back(fv); //将点信息存入多边形点集向量 p 中
    glutPostRedisplay(); //重绘窗口
}
polygon sq;
int i;
//将封闭了的多边形保存到多边形类中
for (i = 0; i < p.size(); i++)
    sq.p.push_back(p[i]);
sq.color = now_color;
s.push_back(sq); //将绘成的多边形存入多边形类向量中
mypaint(sq.p, sq.color); //给当前画完的多边形填色
p.clear();
}
fclose(fp); /*关闭文件*/
//s.clear();
readtxt = !readtxt;
}

void lineSegment()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0); //设定颜色，既是线段颜色也是填充色
    int i, j; //两个循环控制变量，在下面的向量循环和数组循环中将被多次调用。
    if (!s.empty()) //看多边形类向量是否为空，即判断除了当前正在画的多边形是否还有曾经已经画好的多边形
    {
        for (i = 0; i < s.size(); i++) //对多边形类向量循环，该向量中的每个元素代表一个多边形
        {
            int k = s[i].p.size(); //将一个多边形的点的个数，后面划线会用到
            // s[i].line(); //生成多边形的边
            for (j = 0; j < s[i].p.size(); j++) //画多边形
            {

```

```

        glBegin(GL_LINES); //将当前的点与后一个点连线
        glVertex2i(s[i].p[j].x, s[i].p[j].y); //确定前一个点
        glVertex2i(s[i].p[(j + 1) % k].x, s[i].p[(j + 1) %
k].y); //确定后一个点, 通过取模操作来避免越界问题, 该思路来源于循环队列.
        glEnd();
    }
    if (Transform)
    {
        // 旋转
        glRotatef(angle, 0, 0, -1);
        if (translate)
            glTranslated(60, 60, 0);
        // printf("%d", rotate0);
        Transform = !Transform;
    }
    mypaint(s[i].p, s[i].color); //为当前的多边形填充颜色
//    s[i].edge.clear(); //清空当前多边形的边向量
    glColor3f(0.0, 0.0, 0.0);
}
}
i = 0;
j = p.size() - 1;
while (i < j) //循环画图, 画当前正在画的多边形
{
    glBegin(GL_LINES); //将已经确定的点连接起来
    glVertex2i(p[i].x, p[i].y);
    glVertex2i(p[i + 1].x, p[i + 1].y);
    glEnd();
    i++;
}
if (!p.empty()) //画当前点到鼠标点的线段(变化)
{
    // int i = p.size() - 1; //将确定的最后一个点与当前鼠标所在位置连
    线, 即动态画线
    glBegin(GL_LINES);
    glVertex2i(p[j].x, p[j].y);
    glVertex2i(move_x, move_y);
    glEnd();
}

// readtxt = true;
// 读多边形文件
if (readtxt)
    my_readtxt();

```

```

// 保存多边形
if (writetxt)
    savepolygon();
if (select) //判断右键是否被点下
{
    select = false; //将状态值置为假
    if (!p.empty())
    {
        glBegin(GL_LINES); //自动将当前点和第一点连接起来封闭多边形
        glVertex2i(p[j].x, p[j].y);
        glVertex2i(p[0].x, p[0].y);
        glEnd();
        polygon sq;
        //将封闭了的多边形保存到多边形类中
        for (i = 0; i < p.size(); i++)
            sq.p.push_back(p[i]);
        sq.color = now_color;
        s.push_back(sq); //将绘成的多边形存入多边形类向量中
        mypaint(sq.p, sq.color); //给当前画完的多边形填色
    }
    p.clear();
}
glFlush();
}

void myMouse(int button, int state, int x, int y) //鼠标点击事件响应函数
{
    if (state == GLUT_DOWN && button == GLUT_LEFT_BUTTON) //当鼠标左键被
    点击
    {
        point v; //申请一个点类变量，点类为自定义类，在 z1.h 中定义
        v.x = x; //将点击处的点坐标，即 x 和 y 的值存入 v 中
        v.y = screenheight - y;
        cout << "该点坐标  x:" << v.x << ",  y:" << v.y << '\n';
        p.push_back(v); //将点信息存入多边形点集向量 p 中
        glutPostRedisplay(); //重绘窗口
    }

    if (state == GLUT_DOWN && button == GLUT_RIGHT_BUTTON) //当鼠标右键
    被点击
    {
        select = true;
        AET_printer = true;
        Transform = true;
    }
}

```



```

        glutPostRedisplay();
    }
}

void myPassiveMotion(int x, int y) //鼠标移动事件响应函数
{
    move_x = x; //保存当前鼠标所在的坐标的值
    move_y = screenheight - y;
    AET_printer = false;
    glutPostRedisplay();
}

void mymenuOption(GLint option)
{
    switch (option)
    {
        case 1:// 撤销上一步
            if (!p.empty())
                p.pop_back();
            break;
        case 2:// 重画目前在绘制的多边形
            p.clear();
            break;
        case 3:// 删除上一个已经绘制好的多边形
            if (!s.empty())
                s.pop_back();
            break;
        case 4:// 清空画布
            p.clear();
            s.clear();
            glutPostRedisplay();
            break;
        case 5:
            if (angle == 0)
                angle = 10;
            else
                angle = 0;
            break;
        case 6:
            translate = !translate;
            break;
        case 7:
            readtxt = !readtxt;
            break;
    }
}

```

```

    case 8:
        now_color = (now_color + 1) % 5;
        break;
    case 9:
        writetxt = !writetxt;
        break;
    case 10:
        ALLOW_printAET = !ALLOW_printAET;
        break;
}
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(50, 100);
    glutInitWindowSize(screenwidth, screenheight);
    glutCreateWindow("多边形扫描填充算法");
    init();
    glutMouseFunc(myMouse); //鼠标点击消息监控, 即监控鼠标是否被点击, 若被点
击就调用 myMouse 函数
    glutDisplayFunc(lineSegment);
    glutCreateMenu(mymenuOption);
    glutAddMenuEntry("Undo last point", 1);
    glutAddMenuEntry("Shut down current drawing", 2);
    glutAddMenuEntry("Undo last polygon", 3);
    glutAddMenuEntry("Clear board", 4);
    //glutAddMenuEntry("polygon flash", 5);
    //glutAddMenuEntry("Rotate Clockwise or Counterclockwise
Exchange", 6);
    glutAddMenuEntry("Rotate switch", 5);
    glutAddMenuEntry("Translate switch", 6);
    glutAddMenuEntry("color shift", 8);
    glutAddMenuEntry("Read polygon-file", 7);
    glutAddMenuEntry("Save polygon into txt", 9);
    glutAddMenuEntry("Print AET switch", 10);
    glutAttachMenu(GLUT_MIDDLE_BUTTON);
    glutPassiveMotionFunc(myPassiveMotion); //鼠标移动消息监控, 即监控鼠标
是否移动, 若移动就调用 myPassiveMotion 函数
    glutMainLoop();

    return 0;
}

```