实验 2 多边形扫描转换与二维图形变换

一、实验目的与要求：

题目：课本 p142-T5.11：对如图的多边形采用扫描转换算法（改进的有效边表算法）进行填充，试写出该多边形的 ET 表和当扫描线 y=4 时的有效边表 AET。[可选：尝试编写有效边表法程序予以验证（为了便于 OpenGL 显示，建议坐标放大 100 倍）]

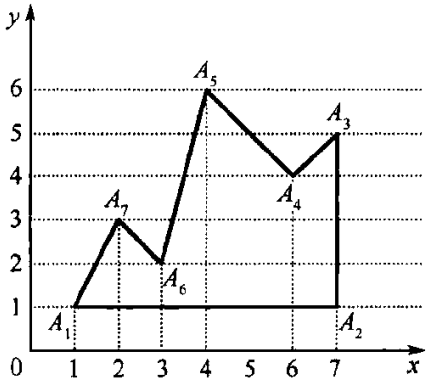


图 5-67　　题 5.11 图

二、实验原理：

有效边表算法——算法步骤：

(1)初始化：构造边表，AET 表置空；

(2)将第一个不空的 ET 表中的边与 AET 表合并；

(3)由 AET 表中取出交点对进行填充。填充之后删除 y=ymax 的边；

(4)y_{i+1}=yi+1,根据 x_{i+1}=xi+1/k 计算并修改 AET 表，同时合并 ET 表中 y=yi+1 桶中的边，按次序插入到 AET 表中，形成新的 AET 表；

(5)AET 表不为空则转(3)，否则结束。

		x|ymin	ymax	1/k		next	x|ymin	ymax	1/k	next
y=1	->	1	3	1/2	->	7	5	0		
y=2	->	3	3	-1	->	3	6	1/4		
y=3										
y=4	->	6	6	-1	->	6	5	1		
y=5										
y=6										

(1)边表

y=4	->	3.5	6	1/4	->	6	6	-1	--	
	-->	6	5	1	->	7	5	0		

(2) y=4 时的有效边表

二、实验步骤及源码：

1. 创建边链表

```
class CEdgeTable { // 边链表
public:
    EDGENODE* m_pHeadEdges;
public:
    CEdgeTable() {
        m_pHeadEdges = nullptr;
    }

    ~CEdgeTable() {
        clear();
    }

    void insertEdge(EDGENODE* pEdge) {// 插入边节点
        if (isEmpty()) {
            m_pHeadEdges = pEdge;
            m_pHeadEdges->next = nullptr;
        }
        else {
            EDGENODE* pE1 = nullptr, * pE2 = m_pHeadEdges;    // 将插入到pE1和pE2之间
            while (pE2) {
                if (pEdge->x < pE2->x) break;
                else if (pEdge->x < pE2->x + 0.001) {// x坐标相等, 则比较斜率倒数
                    if (pEdge->delta > pE2->delta) {// 如该边斜率倒数大于pE2的，则插入到pE2的后面；否则插入到pE2前面
                        pE1 = pE2;
                        pE2 = pE1->next;
                    }
                    break;
                }
                pE1 = pE2;
                pE2 = pE1->next;
            }
            if (pE1) {  // 插入该边节点
```

```
                    pE1->next = pEdge;
                    pEdge->next = pE2;
                }
                else {      // 如果是插入到首节点之前
                    m_pHeadEdges = pEdge;
                    pEdge->next = pE2;
                }
            }
        }

        void insertEdgeTable(CEdgeTable* pEdgeTable) {// 插入边链表，将把pEdgeTable中的所有节点移入到当前边链表中
            EDGENODE* pE1 = pEdgeTable->m_pHeadEdges, * pE2 = nullptr;
            while (pE1)
            {
                pE2 = pE1->next;
                insertEdge(pE1);
                pE1 = pE2;
            }
            pEdgeTable->m_pHeadEdges = nullptr;
        }

        int move2NextLine(int yCur, OUT vector<int>& xInters) {// 返回当前扫描线的交点X坐标，并移动到下一条扫描线
            if (isEmpty()) return 0;

            EDGENODE* pE1 = nullptr, * pE2 = m_pHeadEdges, * q = nullptr;

            while (pE2) {
                xInters.push_back(Roundf(pE2->x));
                pE2->x += pE2->delta;
                if (pE2->ymax == yCur)
                {
                    q = pE2->next;
                    if (pE1) {
                        pE1->next = q;
                    }
                    else {      // 如果首节点要被删除
                        m_pHeadEdges = q;
                    }
                    delete pE2;
                    pE2 = q;
                }
                else
```

```
                {
                    pE1 = pE2;
                    pE2 = pE2->next;
                }
            }
            return xInters.size();
        }

        void clear() {   // 清空边链表
            while (m_pHeadEdges)
            {
                EDGENODE* pNext = m_pHeadEdges->next;
                delete m_pHeadEdges;
                m_pHeadEdges = pNext;
            }
        }

        bool isEmpty() {     // 边链表是否为空
            return (m_pHeadEdges == nullptr);
        }
};
```

2. 创建边表

```
void CreateEageTable(IN vector<VERTEX>& polygon, OUT int& ymin, OUT int& ymax, OUT
vector<CEdgeTable*>& vaET) {
    int i, j, k, nVertex = polygon.size();
    float delta = 0.0f, x = 0.0f;
    int yi = 0, ya = 0;
    // 获取扫描线Y范围
    ymin = ymax = polygon[0].y;
    for (i = 1; i < nVertex; i++) {
        if (polygon[i].y < ymin) ymin = polygon[i].y;
        if (polygon[i].y > ymax) ymax = polygon[i].y;
    }
    int nRows = ymax - ymin + 1;     // 水平扫描线总数
    vaET.resize(nRows);
    for (i = 0; i < nRows; i++)     vaET[i] = nullptr;       // 初始为空桶

    // 按边放入对应的桶
    for (i = 0; i < nVertex; i++)
    {
        j = (i < nVertex - 1) ? (i + 1) : 0;
        VERTEX p1 = polygon[i], p2 = polygon[j];
```

```
            if (p1.y == p2.y) continue;   // 忽略水平边
            if (p1.y < p2.y) {
                x = p1.x;
                yi = p1.y;
                ya = p2.y;
                {
                    k = (j < nVertex - 1) ? (j + 1) : 0;
                    VERTEX p3 = polygon[k];
                    if (p3.y > p2.y) ya--;
                }
            }
            else {
                x = p2.x;
                yi = p2.y;
                ya = p1.y;
                {
                    k = (i == 0) ? (nVertex - 1) : (i - 1);
                    VERTEX p3 = polygon[k];
                    if (p3.y > p1.y) ya--;
                }
            }
            delta = (double)(p2.x - p1.x) / (p2.y - p1.y);

            // 获取yi桶
            CEdgeTable* p = vaET[yi - ymin];
            if (!p)
            {
                p = new CEdgeTable;
                vaET[yi - ymin] = p;
            }
            EDGENODE* pEdge = new EDGENODE(x, ya, delta);
            p->insertEdge(pEdge);
        }
    }
```

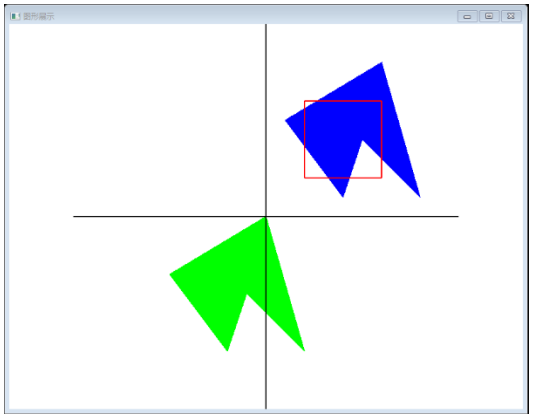
3. AET 法实现多边形扫描转换

```
void ActiveEageTableX(vector<VERTEX>& polygon, COLORREF color) {
	int i, j, ymin, ymax, nRows = 0, nVertex = polygon.size();
	vector<CEdgeTable*> vaET;
	CreateEageTable(polygon, ymin, ymax, vaET);	// 创建ET表，并获取最小和最大扫描线
	nRows = ymax - ymin + 1;	// 水平扫描线总数

	BYTE r = GetRValue(color), g = GetGValue(color), b = GetBValue(color);
	glColor3ub(r, g, b);
	glBegin(GL_POINTS);

	// 从边表里提取AET并逐行填充
	CEdgeTable* pAET = new CEdgeTable;
	for (i = ymin; i <= ymax; i++)
	{
		// ...
		if (vaET[i - ymin]!=nullptr)
		{
			pAET->insertEdgeTable(vaET[i - ymin]);
		}
		vector<int> xInters;
		pAET->move2NextLine(i, xInters);
		for (j = 0; j < xInters.size(); j += 2)
		{
			for (int x = xInters[j]; x <= xInters[j+1]; x++)
				glVertex2i(x, i);
		}

	}
	glEnd();

	// 释放AET和ET
	delete pAET;
	for (i = 0; i < nRows; i++) {
		if (vaET[i]) delete vaET[i];
	}
}
```

三、实验结果及分析

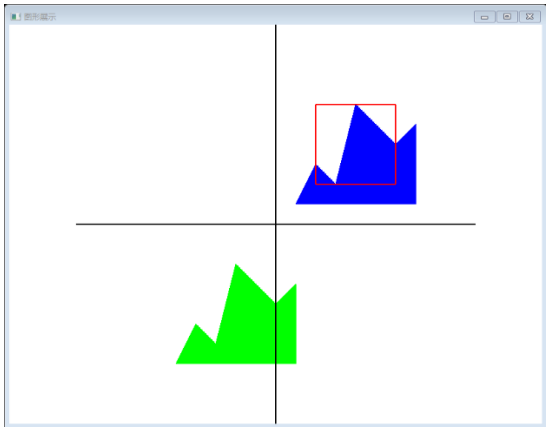
点集：VERTEX p[5] = { {400, 100}, {500, 400}, {800, 100}, {600, 800}, {100, 500} };

运行结果：



点集：VERTEX p[7] = { {100, 100}, {700, 100}, {700, 500}, {600, 400}, {400, 600}, {300,200}, {200,300}};

运行结果：



分析：

1. 了解了算法的基本原理和实现方式：改进有效边表算法是一种用于绘制多边形的算法，通过动态维护一个边表，可以高效地进行三角化和光栅化，从而实现多边形的渲染。
2. 感受到算法的优越性：相比于传统的扫描线算法和逐点法，改进有效边表算法具有更高的效率和更好的性能，可以处理更加复杂的多边形和图形。
3. 熟悉了 OpenGL 的基本使用和编程技巧：学习改进有效边表算法需要熟悉 OpenGL 的基本使用和编程技巧，包括顶点数据的定义、边链表和边表的创建等等。
4. 遇到的问题：在 ActiveEageTableX 函数中，从边表中提取 AET 时，没考虑到空边表导致程序运行中断出错，加上 if 条件判断后即可。