实验二：实域填充算法

综述：使用有序边表实域填充算法和种子实域填充算法进行交互和图形填充

框架：使用MFC：  
cgPoltLine.h包括其他项目特定的标头（包括 Resource.h），并声明 Ccg2022XJPolyLineApp 应用程序类。

cgPolyLineDoc.h编辑文件以添加特殊文档数据并实现文件保存和加载。

CgPolyLineView.h用于查看 cgPolyLineDoc 对象，声明主要的绘制函数

cgControlSelect.h继承MainFrm类，用于一些控件的使用。

算法原理：

1. 有序边表算法：

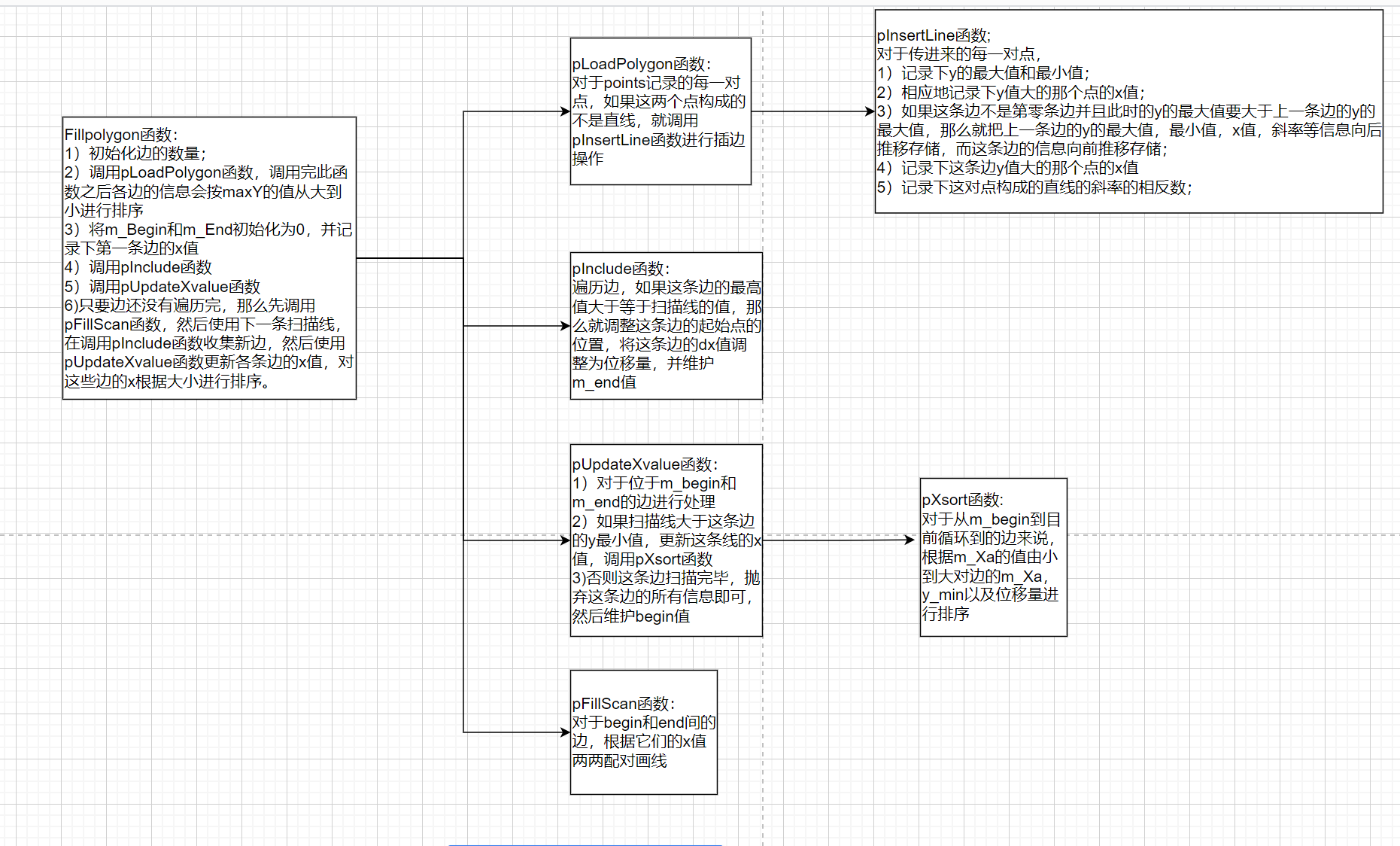
用水平扫描线从上到下（或从下到上）扫描由多条首尾相连的线段构成的多边形，每根扫描线与多边形的某些 边产生一系列交点。

1. 种子算法：

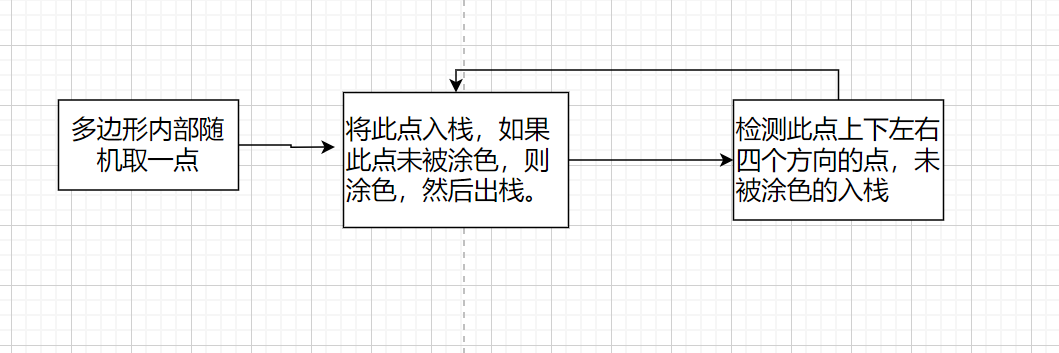
**从多边形区域的一个内点开始，由内向外用给定的颜色画点直到边界为止**。

处理流程：

1. 有序边表算法

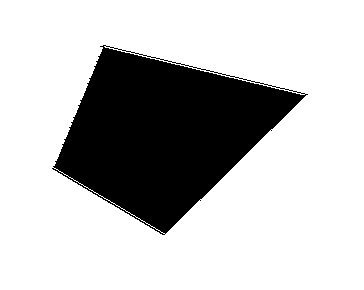
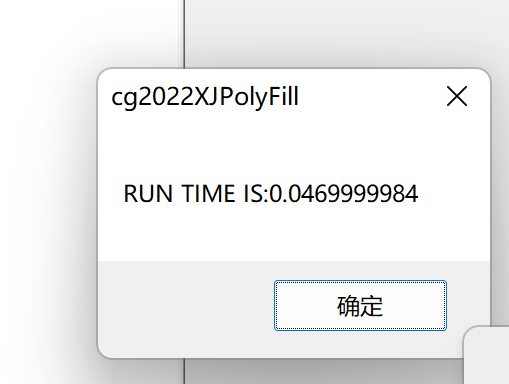


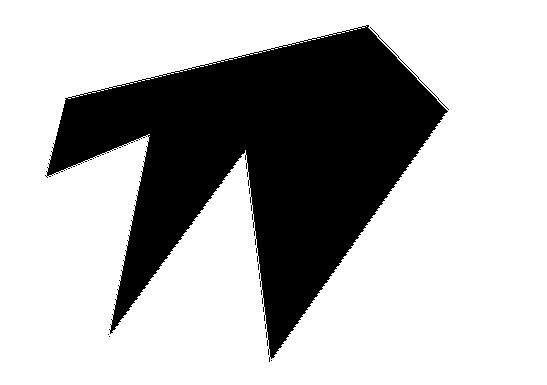
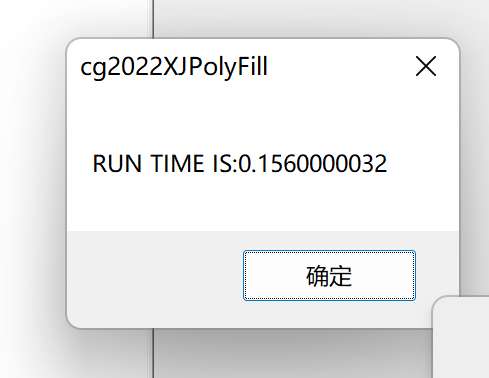
1. 种子算法：



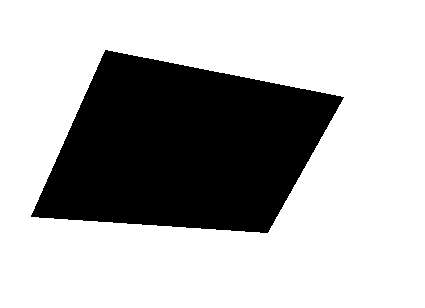
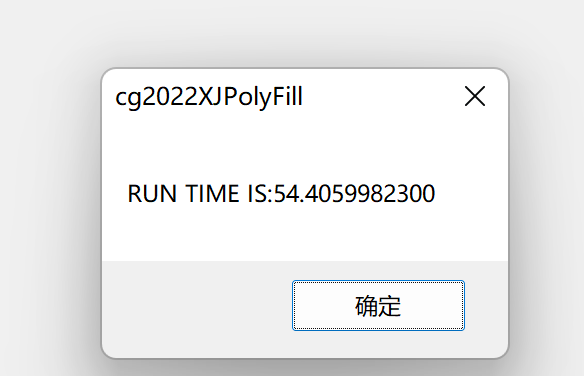
算法分析说明：

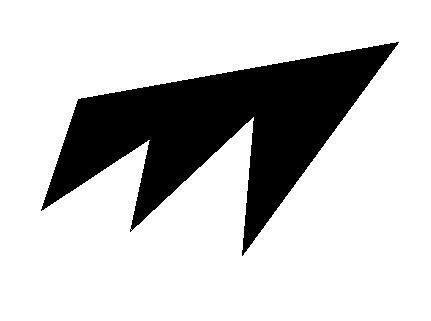
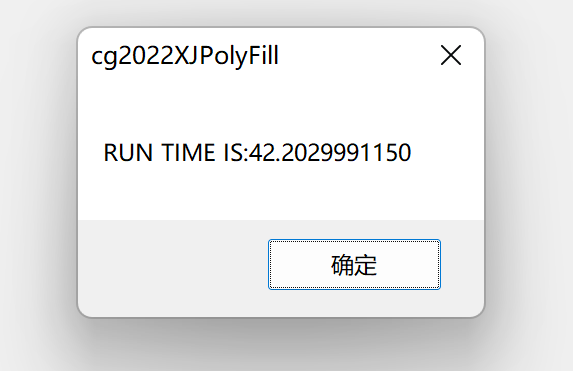
1. 有序边表算法：

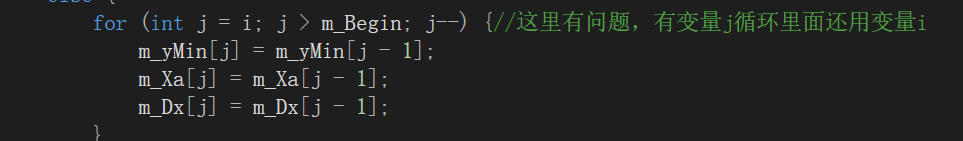
1. 种子算法：

可以看到种子算法的效率比较低。

有序边表算法的错误



将变量j写成了变量i.

源代码：

有序边表算法：

void Ccg2022XJPolyFillView::Fillpolygon(int pNumbers, CPoint \*points, CDC \*pDC)

{

m\_edgeNumbers = 0;//边的数量

pLoadPolygon(pNumbers, points);

m\_Begin = m\_End = 0;

m\_Scan = (int)m\_yMax[0];

pInclude();

pUpdateXvalue();

while (m\_Begin != m\_End) {

pFillScan(pDC);

m\_Scan--;

pInclude();

pUpdateXvalue();

}

}

void Ccg2022XJPolyFillView::pLoadPolygon(int pNumbers, CPoint \*points)//加载多边形

{

float x1, y1, x2, y2;

x1 = points[0].x; y1 = points[0].y + 0.5;

for (int i = 1; i < pNumbers; i++) {

x2 = points[i].x; y2 = points[i].y + 0.5;

if (y1 - y2) pInsertLine(x1, y1, x2, y2);

x1 = x2; y1 = y2;

}

}

void Ccg2022XJPolyFillView::pInsertLine(float x1, float y1, float x2, float y2)//插边

{

int i;

float Ymax, Ymin;

Ymax = (y2 > y1) ? y2 : y1;

Ymin = (y2 < y1) ? y2 : y1;

i = m\_edgeNumbers;//一开始边的数量肯定是0

while (i > 0 && Ymax > m\_yMax[i - 1]) {

m\_yMax[i] = m\_yMax[i - 1]; m\_yMin[i] = m\_yMin[i - 1];

m\_Xa[i] = m\_Xa[i - 1]; m\_Dx[i] = m\_Dx[i - 1];

i--;

}

m\_yMax[i] = Ymax; m\_yMin[i] = Ymin;

//是要记录每条边的终点吗

if (y2 > y1) m\_Xa[i] = x2;

else m\_Xa[i] = x1;

//斜率的相反数

m\_Dx[i] = (x2 - x1) / (y2 - y1); m\_edgeNumbers++;

}

void Ccg2022XJPolyFillView::pInclude()

{

while (m\_End < m\_edgeNumbers && m\_yMax[m\_End] > m\_Scan) {

m\_Xa[m\_End] = m\_Xa[m\_End] - 0.5\*m\_Dx[m\_End];

m\_Dx[m\_End] = -m\_Dx[m\_End];

m\_End++;

}

}

void Ccg2022XJPolyFillView::pUpdateXvalue()

{

int i, start = m\_Begin;

for (i = start; i < m\_End; i++) {

if (m\_Scan > m\_yMin[i]) {

m\_Xa[i] += m\_Dx[i];

pXsort(m\_Begin, i);

}

else {

for (int j = i; j > m\_Begin; j--) {//这里有问题，有变量j循环里面还用变量i

m\_yMin[j] = m\_yMin[j - 1];

m\_Xa[j] = m\_Xa[j - 1];

m\_Dx[j] = m\_Dx[j - 1];

}

m\_Begin++;

}

}

}

void Ccg2022XJPolyFillView::pXsort(int Begin, int i)

{

float temp;

while (i > Begin && m\_Xa[i] < m\_Xa[i - 1]) {

temp = m\_Xa[i]; m\_Xa[i] = m\_Xa[i - 1]; m\_Xa[i - 1] = temp;

temp = m\_Dx[i]; m\_Dx[i] = m\_Dx[i - 1]; m\_Dx[i - 1] = temp;

temp = m\_yMin[i]; m\_yMin[i] = m\_yMin[i - 1]; m\_yMin[i - 1] = temp;

i--;

}

}

void Ccg2022XJPolyFillView::pFillScan(CDC\* pDC)

{

int x, y;

// pDC->SetROP2(10);

for (int i = m\_Begin; i < m\_End; i += 2) {

pDC->MoveTo(m\_Xa[i], m\_Scan);

pDC->LineTo(m\_Xa[i + 1], m\_Scan);

/\* y = m\_Scan;

for (int x = m\_Xa[i]; x < m\_Xa[i + 1]; x++)

if (m\_patternData[y % 7][x % 8])

pDC->SetPixel(x, y, RGB(255, 0, 0));

\*/

}

}

种子算法：

void Ccg2022XJPolyFillView::SeedFill(CDC \*pDC) {

stack<CPoint> p;

COLORREF newColor = RGB(0, 0, 0);

COLORREF oldColor = pDC->GetPixel(m\_seedPoint.x, m\_seedPoint.y);

int dx[4] = { 0,1,0,-1 }, dy[4] = { -1,0,1,0 };

p.push(CPoint(m\_seedPoint.x, m\_seedPoint.y));

while (!p.empty()) {

CPoint temp = p.top();

p.pop();

if (pDC->GetPixel(temp.x, temp.y) != newColor) {

pDC->SetPixel(temp.x, temp.y, newColor);

int x, y;

for (int i = 0; i < 4; i++) {

x = temp.x, y = temp.y;

x += dx[i], y += dy[i];

if (pDC->GetPixel(x, y) == oldColor)

p.push(CPoint(x, y));

}

}

}