# 实验二 实域填充算法 实验报告

姓名：王刚
学号：16020031075

## 一．实验目的：

了解实域填充的基本原理，掌握逐点、有序边表和种子三种实域填充算法原理与实现。

## 二．实验内容：

1. 编程实现逐点实域填充算法。
2. 编程实现有序边表实域填充算法。
3. 编程实现种子实域填充算法。
4. 同实验一添加交互控制按钮或其它控件。
5. 扩展：编程实现多边形三角剖分。

## 三．综述：

1. **Pointwise 按钮**：实现逐点实域填充算法（下简称 **PW** 算法），并实现在 EditControl 控件（ID 为 IDC_SI）上显示执行时间。注：可多次执行，累计显示，并会更新数据为最新执行所获得的数据，其 `MDisplayMode=0`。
2. **SortedEdgeTable 按钮**：实现有序边表实域填充算法（下简称 **SET** 算法），并实现在 IDC_SI 上显示执行时间。注：可多次执行，累计显示，并会更新数据为最新执行所获得数据，其 `MDisplayMode=1`。
3. **Seed 按钮**：实现种子实域填充算法（下简称为 **SEED** 算法），并实现在 IDC_SI 上显示执行时间和执行次数。注：可多次执行，累计显示，并会更新数据为最新执行所获得的数据，其 `MDisplayMode=2`。
4. **Comparison 按钮**：实现 PW 算法、SET 算法和 SEED 算法的比较，比较方面为计算效率。对于计算效率，比较值为算法的运行时间。并在屏幕上再次执行 PW 算法、SET 算法和 SEED 算法。点击按钮以后会自动更新数据，其 `MDisplayMode=3`。
5. **PolygonTriangulation 按钮**： 实现多边形的三角剖分算法（下简称 **PT** 算法）。注意：需要逆时针画点，其 `MDisplayMode=4`。

## 四．算法描述：

**注：算法过程在代码注释。**

**1. 变量定义以及初始化：**

`cgWGFillPolyView.h` 中的变量定义以及含义如下：

```
public:
    CcgWGFillPolyDoc* GetDocument() const;/// 获取Doc指针函数
    int pNumbers;/// 多边形顶点的个数
    CPoint MPolygon[N];/// 多边形存储
    CPoint mousePoint;/// 鼠标位置
    int edgeNumbers;/// 多边形边的条数
    double yMax[N], yMin[N], Xa[N], Dx[N];/// 有序边表发数据结构
    int edgeEnd, edgeBegin;/// 有序边表法边起开始和边结束
    int yScan;/// 有序边表法y扫描线
    int dx[4];/// 种子法的四连通x坐标表示
    int dy[4];/// 种子法的四连通y坐标biaoshi
    COLORREF judgeRgb;/// 种子法初始位置的像素值
    COLORREF targetRgb;/// 种子法需要填充的像素值

// 操作
public:
    void Define_OriginalPoint(CDC *pDC);/// 重新定义坐标系函数
    void SortedEdgeTableFillPolygon(int pNumbers, CPoint MPolygon[], CDC* pDC);/// 有序边表法
    void LoadPolygon(int pNumbers, CPoint MPolygon[]);/// 载入多边形的边
    void InsertPolygon(double x1, double y1, double x2, double y2);/// 插入多边形的边
    void Include();/// 求交函数
    void UpdataXvalue();/// 更新操作变集合函数
    void SortX(int edgeBedin, int i);/// 排序函数
    void FillScan(CDC* pDC);/// 填充函数
    void PointwiseFillPolygon(int pNumbers, CPoint MPolygon[], CDC* pDC);/// 逐点法
    bool JudgeIn(int x, int y, double tempx[], double tempy[], int pNumbers);/// 判断一个点是否在多边形内
    void SeedFillPolygon(int pNumbers, CPoint MPolygon[], CDC* pDC);/// 种子填充法
    void SeedRecursion(int x, int y, CDC* pDC);/// 终止填充法递归函数
    bool PT(int pNumbers, CPoint MPolygon[], CDC* pDC);/// 多边形的三角剖分函数
```

其中，`#define N 1000`

cgWGFillPolyView.cpp 中各变量的初始化如下：

```
CcgWGFillPolyView::CcgWGFillPolyView()
{
    // TODO: 在此处添加构造代码
    pNumbers = 0;/// 初始化pNumbers
    /// 初始化dx和dy数组
    dx[0] = 0; dx[1] = 1; dx[2] = 0; dx[3] = -1;
    dy[0] = -1, dy[1] = 0, dy[2] = 1, dy[3] = 0;
}
```

cgWGFillPolyDoc.h 中的公有函数定义：

```
public:
    int MHeight;/// 视窗高度
    int MWidth;/// 视窗宽度
    int MDisplayMode;/// 函数模式

    double pointwiseRunTime;/// 逐点法运行时间
    CString pointwiseInformation;/// 逐点法显示信息
    double sortedEdgeTableTime;/// 有序边表法运行时间
    CString sortedEdgeTableInformation;/// 有序边表法显示信息
    double seedRunTime;/// 种子法运行时间
    CString seedInformation;/// 种子法显示信息
    CString comparisonInforamtion;/// 比较显示信息
    double polygonalTriangualtionRuntime;/// 多边形的三角剖分时间
    CString polygonalTriangulationInformation;/// 多边形的三角剖分信息
```

cgWGFillPolyDoc.cpp 中的初始化：

```
30  CcgWGFillPolyDoc::CcgWGFillPolyDoc()
31  {
32      // TODO: 在此添加一次性构造代码
33      MHeight = 0;
34      MWidth = 0;
35
36      MDisplayMode = -1;
37
38      sortedEdgeTableInformation = "Here is the SortedEdgeTable information:\r\n";
39      pointwiseInformation = "Here is the Pointwise information:\r\n";
40      seedInformation = "Here is The Seed information:\r\n";
41      comparisonInforamtion = "Here is The Comparison information:\r\n";
42      polygonalTriangulationInformation = "Here is The PolygonTriangulation information:\r\n";
43      sortedEdgeTableTime = 0;
44      pointwiseRunTime = 0;
45      seedRunTime = 0;
46      polygonalTriangualtionRuntime = 0;
47  }
```

## 2. 逐点实域填充算法之 PointwiseFillPolygon 函数：

算法原型：`void PointwiseFillPolygon(int pNumbers, CPoint MPolygon[], CDC* pDC)`

函数代码：

```
136 void CcgWGFillPolyView::PointwiseFillPolygon(int pNumbers, CPoint MPolygon[], CDC* pDC)
137 {
138     DWORD dwStart = GetTickCount();/// 获取开始时间
139
140     const int INF = 0x3f3f3f3f;/// 定义最大值常量
141     int maxX = -INF, minX = INF, maxY = -INF, minY = INF;/// 变量赋值
142     /// tempx和tempy是MPolygon的上移半格存储数组
143     double tempx[N];/// 定义tempx
144     double tempy[N];/// 定义tempy
145     for (int i = 0; i < pNumbers; i++) {/// 更新tempx和tempy数组
146         tempx[i] = MPolygon[i].x;
147         tempy[i] = MPolygon[i].y;
148         if (tempx[i] > maxX) {
149             maxX = (int)tempx[i];
150         }
151         if (tempx[i] < minX) {
152             minX = (int)tempx[i];
153         }
154         if (tempy[i] > maxY) {
155             maxY = (int)tempy[i];
156         }
157         if (tempy[i] < minY) {
158             minY = (int)tempy[i];
159         }
160         tempy[i] += 0.5;/// 上移半格
161     }
162     for (int i = minX; i <= maxX; i++) {/// 循环遍历矩形区域
163         for (int j = minY; j <= maxY; j++) {
164             if (JudgeIn(i, j, tempx, tempy, pNumbers)) {/// 判定该点是不是在多边形内
165                 pDC->SetPixel(i, j, RGB(255, 0, 0));
166             }

167         }
168     }
169
170     DWORD dwStop = GetTickCount();/// 获得结束时间
171     DWORD dwInterval = dwStop - dwStart;/// 获得运行时间
172     CcgWGFillPolyDoc* pDoc = (CcgWGFillPolyDoc*)GetDocument();/// 获得Doc指针
173     pDoc->pointwiseRunTime = (double)dwInterval;/// 将数据写入pointwiseRunTime
174     CString str;/// 将数据写入pointwiseInformation，并将其格式化
175     str.Format(_T("%.2lf"), pDoc->pointwiseRunTime);
176     pDoc->pointwiseInformation += _T("Runtime Consuming : ") + str + _T("ms\r\n");
177
178     pDoc->UpdateAllViews(this);/// 更新视图
179 }
180
```

## 3. 逐点实域填充算法之 JudgeIn 函数：

算法原型：`bool JudgeIn(int x, int y, double tempx[], double tempy[], int pNumbers)`

函数代码：

```
bool CcgWGFillPolyView::JudgeIn(int x, int y, double tempx[], double tempy[], int pNumbers)
{/// 判定一个点(x, y)是不是在多边形(tempx和tempy表示)内，pNumbers是多边形点的个数
    int i, j;/// 定义循环变量
    int ans = 0;/// ans表示当前点向右做射线和多边形的交点个数
    for (i = 1, j = 0; i < pNumbers; j = i++) {/// 遍历
        bool cond1 = (y < tempy[i] && y > tempy[j]);/// 条件一
        bool cond2 = (y < tempy[j] && y > tempy[i]);/// 条件二
        bool cond3;/// 条件三
        if (tempy[i] - tempy[j] > 0)
            cond3 = ((y*(tempx[i] - tempx[j]) - tempy[j] * (tempx[i] - tempx[j]) + (tempy[i] - tempy[j])*tempx[j])
                >= x * (tempy[i] - tempy[j]));
        else
            cond3 = ((y*(tempx[i] - tempx[j]) - tempy[j] * (tempx[i] - tempx[j]) + (tempy[i] - tempy[j])*tempx[j])
                <= x * (tempy[i] - tempy[j]));
        if ((cond1 || cond2) && cond3) {
            ans++;/// 更新ans
        }
    }
    return (ans % 2 != 0);/// ans为偶数返回FALSE，奇数返回TRUE
}
```

## 4. 有序边表实域填充算法之 SortedEdgeTableFillPolygon 函数：

函数原型：`void SortedEdgeTableFillPolygon(int pNumbers, CPoint MPolygon[], CDC* pDC)`

函数代码：

```
void CcgWGFillPolyView::SortedEdgeTableFillPolygon(int pNumbers, CPoint MPolygon[], CDC* pDC)
{
    DWORD dwStart = GetTickCount();/// 获取开始时间
    const int runCount = 100;/// 定义运行次数
    int time = runCount;

    while (time--) {/// 循环画图
        edgeNumbers = 0;/// 定义边个数
        LoadPolygon(pNumbers, MPolygon);/// 读入MPolygon数组
        edgeBegin = edgeEnd = 0;/// 定义边开始和边结束变量
        yScan = (int)yMax[0];/// 初始化y扫描线
        Include();/// 判定是否有交点
        UpdataXvalue();/// 更新开始边和结束边，亦即当前操作的边的集合
        while (edgeBegin < edgeEnd) {/// 当开始边和结束边不重合时
            FillScan(pDC);/// 画直线
            yScan--;/// y扫描线迭代更新
            Include();/// 判定是否交点，亦即是否有新的边加入
            UpdataXvalue();/// 更新当前操作的边的集合
        }
    }

    DWORD dwStop = GetTickCount();/// 获取结束时间
    DWORD dwInterval = dwStop - dwStart;/// 获取运行时间
    CcgWGFillPolyDoc* pDoc = (CcgWGFillPolyDoc*)GetDocument();/// 获取Doc指针
    pDoc->sortedEdgeTableTime = (double)dwInterval / runCount;/// 将运行时间写入sortedEdgeTableTime
    CString str;/// 将数据写入sortedEdgeTableInformation中
    str.Format(_T("%.2lf"), pDoc->sortedEdgeTableTime);
    pDoc->sortedEdgeTableInformation += _T("Runtime Consuming : ") + str + _T("ms\r\n");

    pDoc->UpdateAllViews(this);/// 更新视图
}
```

注：有序边表法运行效率非常快，因此我让有序边表法填充实域 runCount(=100)次，然后再将运行时间除以 runCount 得到填充一次的平均时间。

## 5. 有序边表实域填充算法之 LoadPolygon 函数：

算法原型：`void LoadPolygon(int pNumbers, CPoint MPolygon[])`

函数代码：

```
230  void CcgWGFillPolyView::LoadPolygon(int pNumbers, CPoint MPolygon[])
231  {
232      double x1, y1, x2, y2;/// 定义变量
233
234      x1 = MPolygon[0].x;
235      y1 = MPolygon[0].y + 0.5;/// 上移半格
236      for (int i = 1; i < pNumbers; i++) {
237          x2 = MPolygon[i].x;
238          y2 = MPolygon[i].y + 0.5;
239          if (y1 != y2) {/// 把斜率不为0的直线才执行写入
240              InsertPolygon(x1, y1, x2, y2);
241          }
242          x1 = x2;/// 更新x1
243          y1 = y2;/// 更新x2
244      }
245  }
246
```

### 6. 有序边表实域填充算法之 InsertPolygon 函数：

函数原型：`void InsertPolygon(double x1, double y1, double x2, double y2)`

函数代码：

```
247  void CcgWGFillPolyView::InsertPolygon(double x1, double y1, double x2, double y2)
248  {
249      int i;/// 定义循环变量
250      double Ymax, Ymin;/// 定义纵坐标的最大值Ymax和最小值Ymin
251      Ymax = (y2 > y1) ? y2 : y1;/// 更新Ymax
252      Ymin = (y2 > y1) ? y1 : y2;/// 更新Ymin
253      i = edgeNumbers;/// 更新i值为边个数
254      /// 冒泡排序，按照纵坐标从小到大排序
255      while (i > 0 && Ymax > yMax[i - 1]) {///当边不空，且当前Ymax大于上一条边的纵坐标
256          yMax[i] = yMax[i - 1];/// 更新数值
257          yMin[i] = yMin[i - 1];
258          Xa[i] = Xa[i - 1];
259          Dx[i] = Dx[i - 1];
260          i--;/// 迭代i，i同时又代表Ymax和Ymin需要填入的位置
261      }
262      yMax[i] = Ymax;/// 赋值
263      yMin[i] = Ymin;/// 赋值
264      if (y2 > y1) /// 更新Xa[i]的数值为Ymax对应的横坐标值;
265          Xa[i] = x2;
266      else
267          Xa[i] = x1;
268      Dx[i] = (x1 - x2) / (y1 - y2);/// 跟新Dx[i]为当前直线的斜率的倒数
269      edgeNumbers++;/// 边的个数加一
270  }
271
```

### 7. 有序边表实域填充算法之 Include 函数：

算法原型：`void Include()`

函数代码：

```
272  void CcgWGFillPolyView::Include()
273  {
274      while (edgeEnd < edgeNumbers && yScan < yMax[edgeEnd]) {
275          /// 当边结束小于边条数且扫描线小于边结束边的y坐标时
276          Xa[edgeEnd] = Xa[edgeEnd] - 0.5 * Dx[edgeEnd];///获得真正的x坐标
277          Dx[edgeEnd] = -Dx[edgeEnd];/// 去Dx的倒数，方便后续计算
278          edgeEnd++;/// 这证明加入了一条新边
279      }
280  }
281
```

### 8. 有序边表实域填充算法之 UpdataXvalue 函数：

函数原型：`void UpdataXvalue()`

函数代码：

```
282  void CcgWGFillPolyView::UpdataXvalue()
283  {
284      int start = edgeBegin;/// 定义start边变量
285
286      for (int i = start; i < edgeEnd; i++) {///从start边到结束边遍历
287          if (yScan > yMin[i]) {/// 当前i边的yMin小于y扫描线
288              Xa[i] = Xa[i] + Dx[i];/// 更新Xa[i]的数值，从上一次的Xa[i]获得
289              SortX(edgeBegin, i);/// Xa数组从开始变到当前边i从小到大排序
290          }
291          else {/// 当前边需要被去掉
292              for (int j = i; j > edgeBegin; j--) {
293                  /// 从当前边i到开始边从后往前更新
294                  yMin[j] = yMin[j - 1];/// 后一条边的数据更新为前一条边的数据
295                  Xa[j] = Xa[j - 1];
296                  Dx[j] = Dx[j - 1];
297                  ///yMax[j] = yMax[j - 1];可以加上，但是用不到
298              }
299              edgeBegin++;/// 更新开始边
300          }
301      }
302  }
303
```

### 9. 有序边表实域填充算法之 SortX 函数：

函数原型：`void SortX(int edgeBegin, int i)`

函数代码：

```
304  void CcgWGFillPolyView::SortX(int edgeBegin, int i)
305  {
306      double temp;/// 定义临时变量
307      while (i > edgeBegin && Xa[i] < Xa[i - 1]) {/// 冒泡排序
308          temp = Xa[i]; Xa[i] = Xa[i - 1]; Xa[i - 1] = temp;
309          temp = Dx[i]; Dx[i] = Dx[i - 1]; Dx[i - 1] = temp;
310          temp = yMin[i]; yMin[i] = yMin[i - 1]; yMin[i - 1] = temp;
311          ///temp = yMax[i]; yMax[i] = yMax[i - 1]; yMax[i - 1] = temp;可以加上
312          i--;/// 迭代
313      }
314  }
315
```

### 10. 有序边表实域填充算法之 FillScan 函数：

函数原型：`void FillScan(CDC* pDC)`

函数代码：

```
316  void CcgWGFillPolyView::FillScan(CDC* pDC)
317  {
318      for (int i = edgeBegin; i < edgeEnd; i += 2) {///点点之间画点
319          pDC->MoveTo(Xa[i], yScan);
320          pDC->LineTo(Xa[i + 1], yScan);
321      }
322  }
323
```

### 11. 种子实域填充算法之 SeedFillPolygon 函数：

函数原型：`void SeedFillPolygon(int pNumbers, CPoint MPolygon[], CDC* pDC)`

函数代码：

```cpp
void CcgWGFillPolyView::SeedFillPolygon(int pNumbers, CPoint MPolygon[], CDC* pDC)
{
    DWORD dwStart = GetTickCount();/// 获得开始时间
    const int INF = 0x3f3f3f3f;/// 定义最大值常量
    int maxX = -INF, minX = INF, maxY = -INF, minY = INF;/// 赋值
    /// tempx和tempy是MPolygon的上移半格存储数组
    double tempx[N];/// 定义tempx
    double tempy[N];/// 定义tempy
    for (int i = 0; i < pNumbers; i++) {/// 更新tempx和tempy数组
        tempx[i] = MPolygon[i].x;
        tempy[i] = MPolygon[i].y;
        if (tempx[i] > maxX) {
            maxX = (int)tempx[i];
        }
        if (tempx[i] < minX) {
            minX = (int)tempx[i];
        }
        if (tempy[i] > maxY) {
            maxY = (int)tempy[i];
        }
        if (tempy[i] < minY) {
            minY = (int)tempy[i];
        }
        tempy[i] += 0.5;/// 上移半格
    }
    srand(unsigned(time(NULL)));/// 随机种子
    int randX, randY;/// 定义随机变量坐标(randX, randY)
    while (TRUE) {/// 直到找到在多边形内部的一个点为止
        /// 在矩形区域(minX,minY)~(maxX, maxY)内找点

        randX = rand() % (maxX - minX) + minX;
        randY = rand() % (maxY - minY) + minY;
        if (JudgeIn(randX, randY, tempx, tempy, pNumbers))
            break;/// 判断当前点是不是在多边形内
    }

    judgeRgb = pDC->GetPixel(randX, randY);/// 设置judgeRgb
    targetRgb = RGB(0, 255, 0);/// 设置需要填充的颜色

    SeedRecursion(randX, randY, pDC);/// 执行递归函数

    DWORD dwStop = GetTickCount();/// 获得结束时间
    DWORD dwInterval = dwStop - dwStart;/// 运行时间
    CcgWGFillPolyDoc* pDoc = (CcgWGFillPolyDoc*)GetDocument();/// 获取Doc指针
    pDoc->seedRunTime = (double)dwInterval;/// 把运行时间写入seedRunTime中
    CString str;/// 把数据写入seedInformation,并格式化
    str.Format(_T("%.2lf"), pDoc->seedRunTime);
    pDoc->seedInformation += _T("Runtime Consuming : ") + str + _T("ms\r\n");

    pDoc->UpdateAllViews(this);/// 更新视图
}
```

## 12. 种子实域填充算法之 SeedRecursion 函数：

函数原型：`void SeedRecursion(int x, int y, CDC* pDC)`

函数代码：

```cpp
void CcgWGFillPolyView::SeedRecursion(int x, int y, CDC* pDC)
{/// 图像不能画太大qwq
    pDC->SetPixel(x, y, targetRgb);/// 画当前点
    for (int i = 0; i < 4; i++) {/// 循环遍历dx和dy数组,以实现对区域的四连通
        COLORREF now = pDC->GetPixel(x + dx[i], y + dy[i]);/// 得到新的当前点的像素值
        if (now == judgeRgb)/// 如果新的当前点的像素值是judgeRbg的话
            SeedRecursion(x + dx[i], y + dy[i], pDC);///递归调用SeedRecursion
    }
}
```

## 13. 多边形三角剖分函数 PT：

函数原型：`bool PT(int pNumbers, CPoint MPOlygon[], CDC* pDC)`

函数代码：

```cpp
bool CcgWGFillPolyView::PT(int pNumbers, CPoint MPOlygon[], CDC* pDC)
{
    if (pNumbers <= 4)/// 如果当前多边形的边数小于3那么判定该多边形非法
        return FALSE;

    DWORD dwStart = GetTickCount();/// 获得开始运行时间
    CPen pNewPen;///定义画笔颜色为红色
    pNewPen.CreatePen(PS_SOLID, 1, RGB(255, 0, 0));
    CPen* poldPen = pDC->SelectObject(&pNewPen);
    /// 定义循环变量i,j,k
    int i = 0;
    int j = 1;
    int k = 2;
    while (pNumbers > 4) {/// 当多边形的边数大于3时进行
        bool cond1;/// 条件一，向量ij的和向量ik的夹角必须小于π
        double a1 = MPOlygon[j].x - MPOlygon[i].x;
        double a2 = MPOlygon[j].y - MPOlygon[i].y;
        double b1 = MPOlygon[k].x - MPOlygon[i].x;
        double b2 = MPOlygon[k].y - MPOlygon[i].y;
        cond1 = (a1*b2 <= a2*b1);
        bool cond2 = TRUE;/// 条件二，当前多边形中不能有其他点
        double tempx[4];/// 当前多边形上移半格
        double tempy[4];
        tempx[0] = MPOlygon[i].x;
        tempy[0] = MPOlygon[i].y+0.5;
        tempx[1] = MPOlygon[j].x;
        tempy[1] = MPOlygon[j].y+0.5;
        tempx[2] = MPOlygon[k].x;
        tempy[2] = MPOlygon[k].y+0.5;
        tempx[3] = tempx[0];
        tempy[3] = tempy[0];

        for (int p = 0; p < pNumbers; p++) {
            if (p != i && p != j && p != k) {/// 判断其他点是否在当前三角形内
                if (JudgeIn(MPOlygon[p].x, MPOlygon[p].y, tempx, tempy, 3 + 1)) {
                    cond2 = FALSE;
                    break;
                }
            }
        }
        if (cond1 && cond2) {/// 如果同时满足条件一和条件二
            pDC->MoveTo(MPOlygon[i]);/// 从i点到k点画线
            pDC->LineTo(MPOlygon[k]);
            for (int t = j; t < pNumbers - 1; t++) {/// 更新当前多边形
                MPOlygon[t] = MPOlygon[t + 1];
            }
            pNumbers--;/// 更新多边形的边数
            i = 0;/// 重新赋值i，j，k
            j = 1;
            k = 2;
        }
        else {/// 否则
            i ++;/// i下移一点
            j = i + 1;
            k = j + 1;
        }
    }

    pDC->SelectObject(poldPen);/// 跟新pDC

    DWORD dwStop = GetTickCount();/// 获得结束时间
    DWORD dwInterval = dwStop - dwStart;/// 获得运行时间

    CcgWGFillPolyDoc* pDoc = (CcgWGFillPolyDoc*)GetDocument();/// 获得Doc指针
    pDoc->polygonalTriangualtionRuntime = (double)dwInterval;/// 将数据写入Doc类中
    CString str;/// 将数据写入polygonalTriangulationInformation，并将其格式化
    str.Format(_T("%.2lf"), pDoc->polygonalTriangualtionRuntime);
    pDoc->polygonalTriangulationInformation += _T("Runtime Consuming : ") + str + _T("ms\r\n");

    pDoc->UpdateAllViews(this);/// 更新视图
    return TRUE;
}
```

注：对于多边形的三角剖分，我定义输入的多边形是以逆时针的顺序画出的，因此需要逆时针画点。当然，可以加一些重复代码进行拓展成任意时针，但是考虑到代码长度和实验考察重点，这里就不再做拓展了。请在输入时注意 MessageBox 和逆时针输入即可。

**14. 鼠标交互之 OnLButtonDown 函数：**

**函数原型：**`void OnLButtonDown(UINT nFlags, CPoint point)`

**函数代码：**

```
466  void CcgWGFillPolyView::OnLButtonDown(UINT nFlags, CPoint point)
467  {
468      // TODO: 在此添加消息处理程序代码和/或调用默认值
469      if (pNumbers < N) {/// 防止栈溢出
470          MPolygon[pNumbers] = point;/// 更新MPolygon数组
471          pNumbers++;
472          mousePoint = point;
473      }
474      CView::OnLButtonDown(nFlags, point);
475  }
476
```

**15. 鼠标交互之 OnMouseMove 函数：**

函数原型：`void OnMouseMove(UINT nFlags, CPoint point)`

函数代码：

```
477  void CcgWGFillPolyView::OnMouseMove(UINT nFlags, CPoint point)
478  {
479      // TODO: 在此添加消息处理程序代码和/或调用默认值
480      CDC* pDC = GetDC();
481      if (pNumbers > 0) {
482          pDC->SetROP2(2);///取背景色的反，抹除上一条线
483          pDC->MoveTo(MPolygon[pNumbers - 1]);
484          pDC->LineTo(mousePoint);
485
486          mousePoint = point;
487          pDC->MoveTo(MPolygon[pNumbers - 1]);
488          pDC->LineTo(mousePoint);
489      }
490
491      CView::OnMouseMove(nFlags, point);
492  }
493
```

### 16. 鼠标交互之 OnLButtonDblClk 函数：

函数原型：`void OnLButtonDblClk(UINT nFlags, CPoint point)`

函数代码：

```
494  void CcgWGFillPolyView::OnLButtonDblClk(UINT nFlags, CPoint point)
495  {
496      // TODO: 在此添加消息处理程序代码和/或调用默认值
497      CcgWGFillPolyDoc* pDoc = (CcgWGFillPolyDoc*)GetDocument();/// 获取Doc指针
498      CDC* pDC = GetDC();/// 获取CDC指针
499      pDC->MoveTo(MPolygon[pNumbers-1]);
500      pDC->LineTo(MPolygon[0]);/// 话最后一条线
501      MPolygon[pNumbers] = MPolygon[0];/// 再次添加一个点为第一个点
502      pNumbers++;
503
504      if (pDoc->MDisplayMode == 0) {/// 点击按钮Pointwise后执行
505          PointwiseFillPolygon(pNumbers, MPolygon, pDC);
506      }
507      if (pDoc->MDisplayMode == 1) {/// 点击SortedEdgeTable按钮后执行
508          SortedEdgeTableFillPolygon(pNumbers, MPolygon, pDC);
509      }
510      if (pDoc->MDisplayMode == 2) {/// 点击Seed按钮后执行
511          SeedFillPolygon(pNumbers, MPolygon, pDC);
512      }
513      if (pDoc->MDisplayMode == 3) {/// 点击Comparison按钮后执行
514          /// 更新comparisonInformation，并将其格式化
515          pDoc->comparisonInforamtion = "Here is The Comparison information:\r\n";
516          pDoc->comparisonInforamtion += _T("\tPointWise\t\tSorted\t\tSeed\r\nTime:\t");
517          CString str;/// 以下为将再次执行上述三个函数的结果写入comparisonInformation
518          PointwiseFillPolygon(pNumbers, MPolygon, pDC);
519          str.Format(_T("%.2lf"), pDoc->pointwiseRunTime);
520          pDoc->comparisonInforamtion += str + _T("\t\t");
521          SortedEdgeTableFillPolygon(pNumbers, MPolygon, pDC);
522          str.Format(_T("%.2lf"), pDoc->sortedEdgeTableTime);
523          pDoc->comparisonInforamtion += str + _T("\t\t");
524          SeedFillPolygon(pNumbers, MPolygon, pDC);
525          str.Format(_T("%.2lf"), pDoc->seedRunTime);
526          pDoc->comparisonInforamtion += str + _T("\r\n");
527
528          pDoc->UpdateAllViews(this);/// 更新视图
529      }
530      if (pDoc->MDisplayMode == 4) {/// 点击PolygonTriangulation按钮后执行
531          if (!PT(pNumbers, MPolygon, pDC))/// 要求点必须逆时针画qwq
532              MessageBox(_T("The Polygon is illegal!"));
533      }
534      pNumbers = 0;/// 更新多边形的边数
535
536      CView::OnLButtonDblClk(nFlags, point);
537  }
```

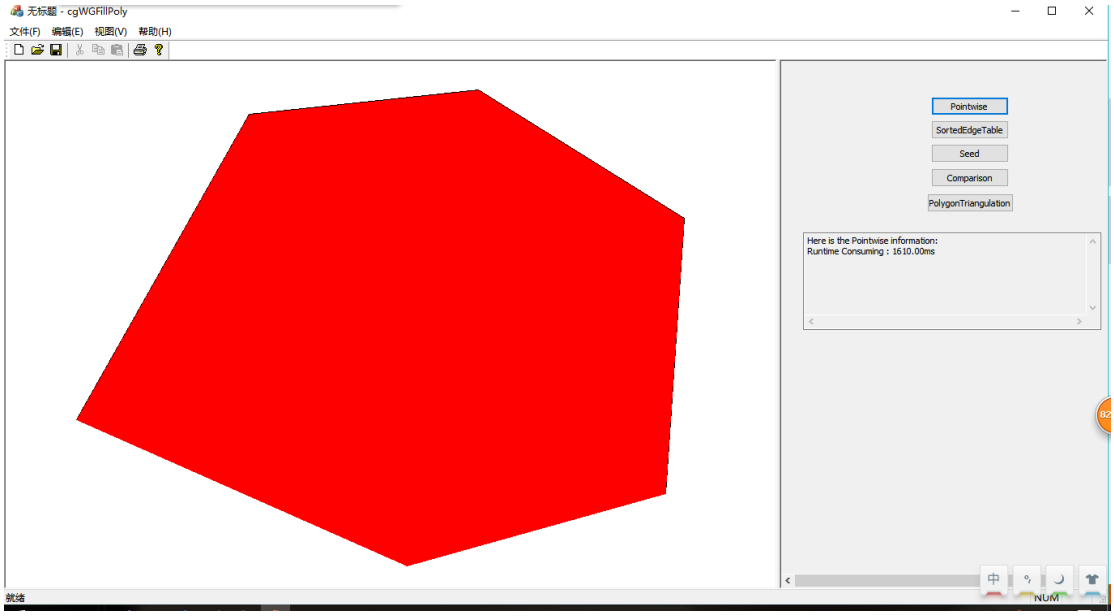
注：当画图形时，最后一条线不需要画出，只需要在最后一个点双击鼠标左键，程序会自动画出最后一条线，且对应算法会自动执行。

## 五．处理流程说明：

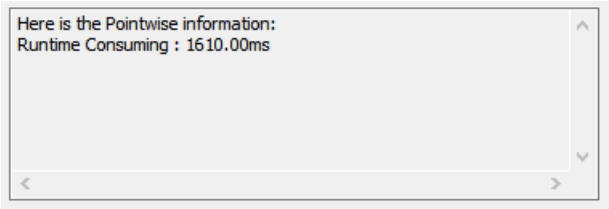
### 1. 程序运行界面如下：


**2. 点击 Pointwise 按钮:**

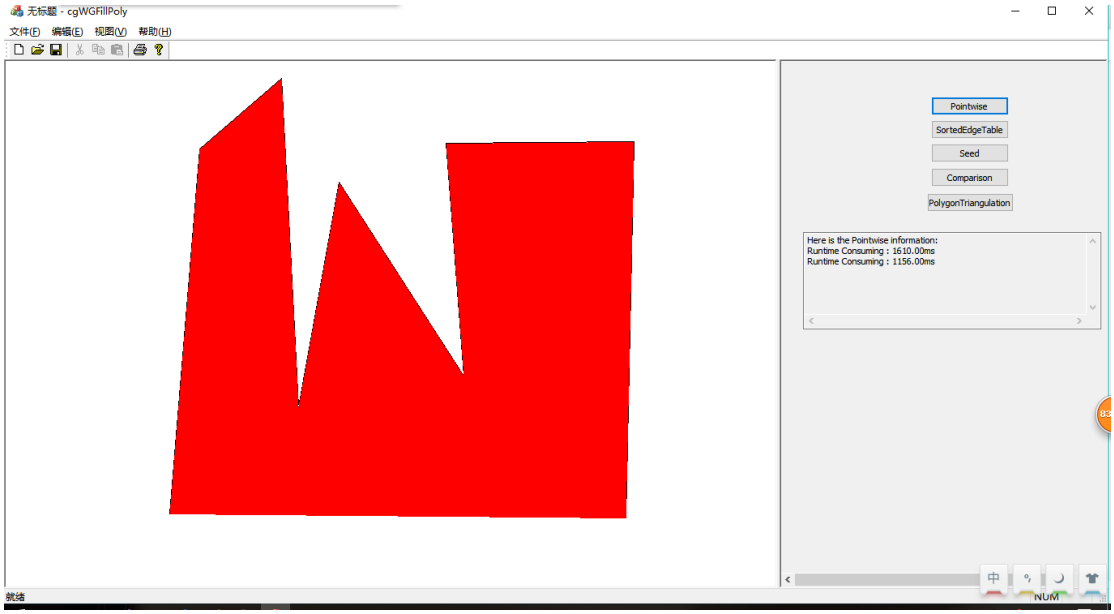
**画凸多边形：**



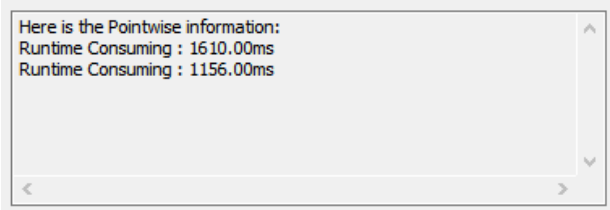

EditControl 控件 IDC_SI 放大：




**画凹多边形：**

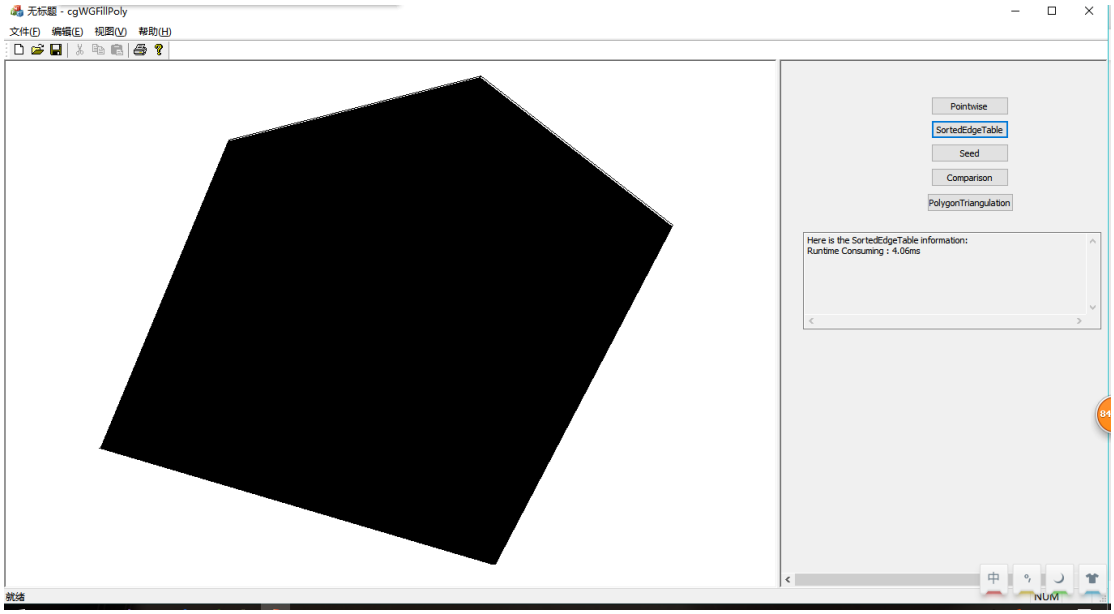



EditControl 控件 IDC_SI 放大：

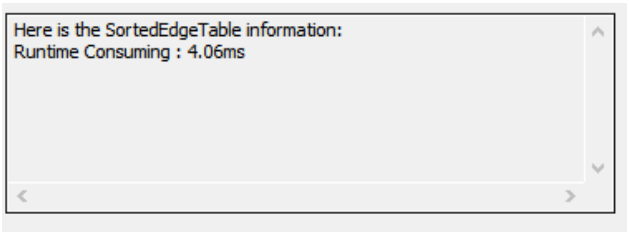



### 3. 点击 SortedEdgeTable 按钮:

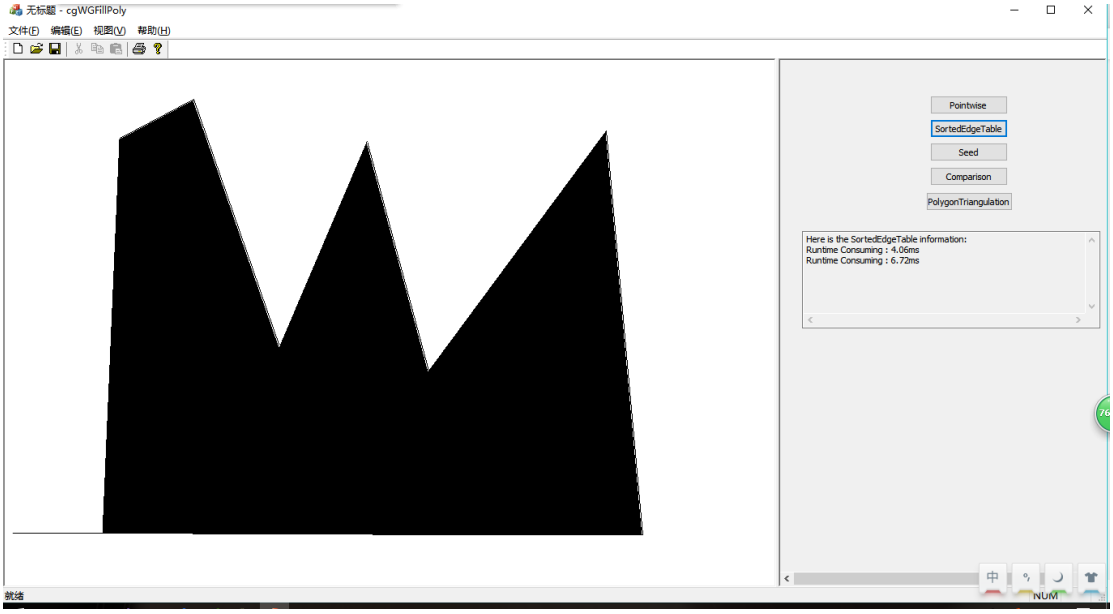
**画凸多边形：**



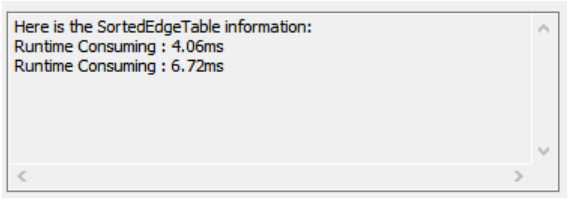

EditControl 控件 IDC_SI 放大：




**画凹多边形：**

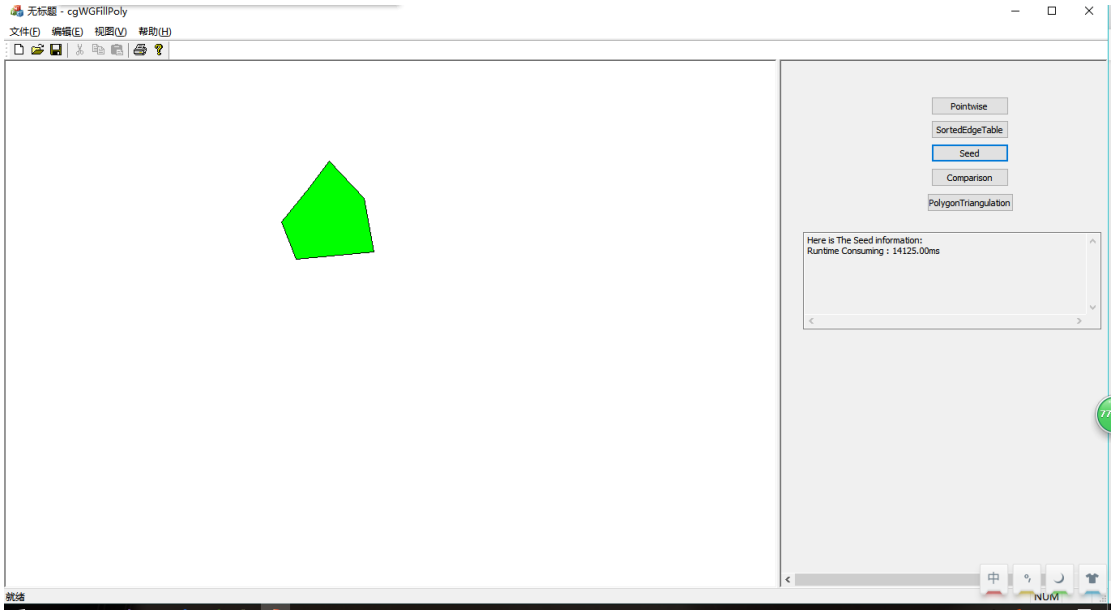



EditControl 控件 IDC_SI 放大：

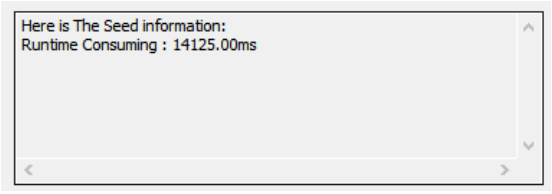



**4. 点击 Seed 按钮:**

**画凸多边形：**

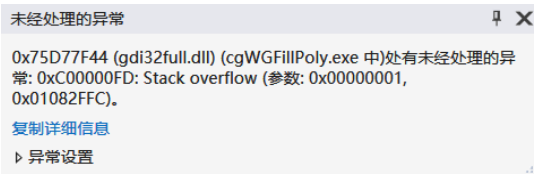



EditControl 控件 IDC_SI 放大：

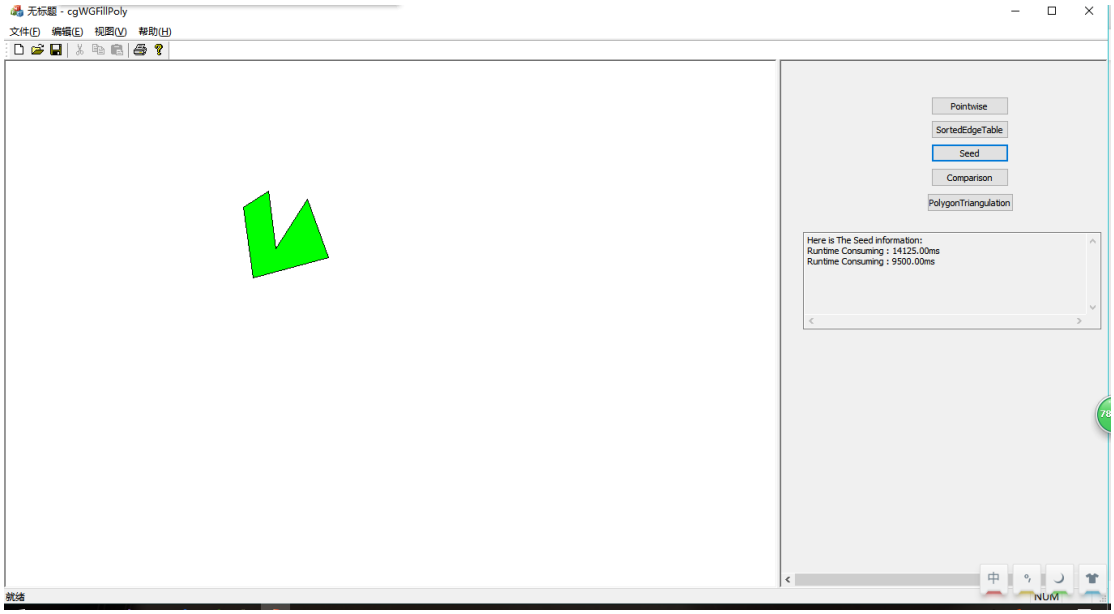



注：

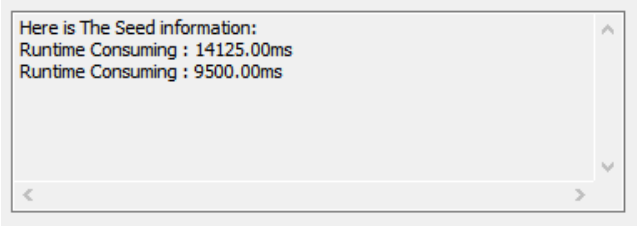
1. 可以看出，当画得区域很小时，运行时间就已经是 14125ms 了，这一点说明，种子实域填充法运行效率是非常的慢！
2. 种子实域填充算法使用递归，且需要判断点的像素值。因此，画图时不能画得太大，画得太大会导致栈溢出（Stack overflow），如下图。经过我的反复调试，所画的多边形最大面积为 9cm^2（不同的环境可能不同）。




**画凹多边形：**

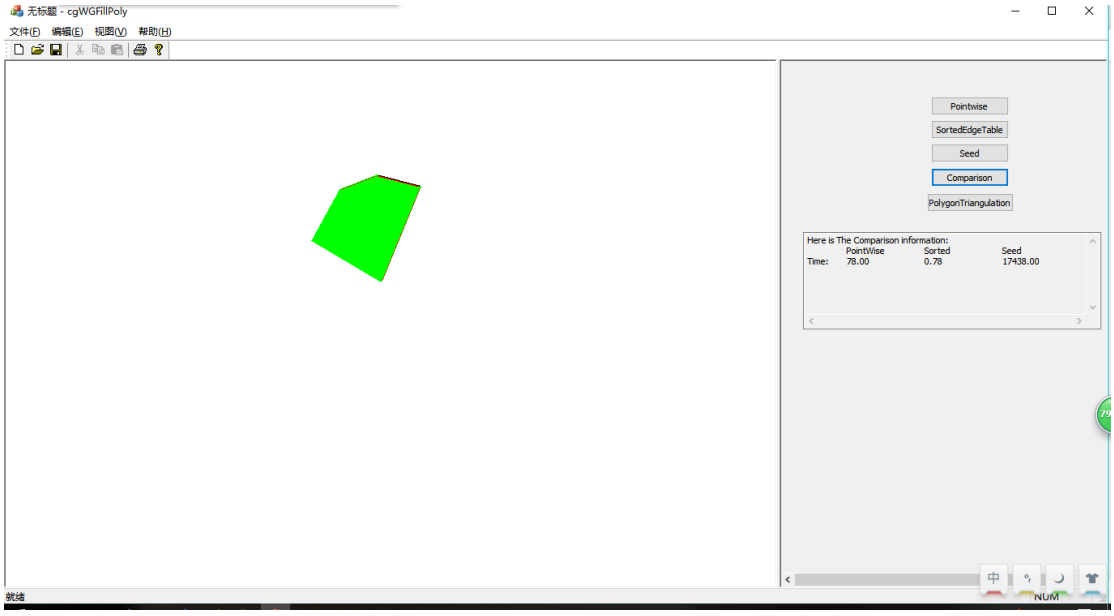



EditControl 控件 IDC_SI 放大：

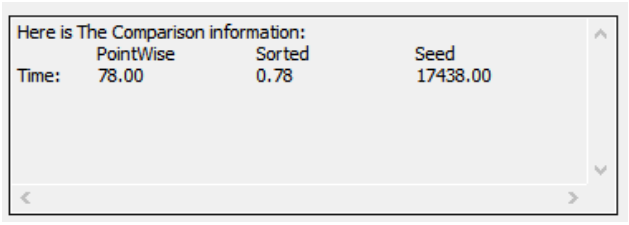



**5. 点击 Comparison 按钮：**

**画凸多边形：**



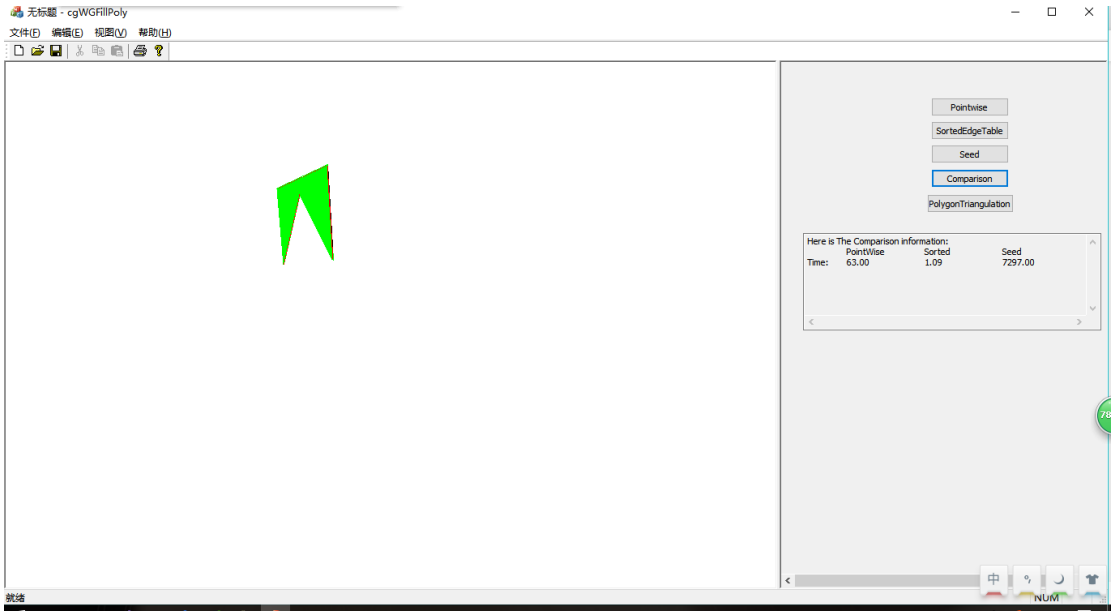

EditControl 控件 IDC_SI 放大（单位 ms）：



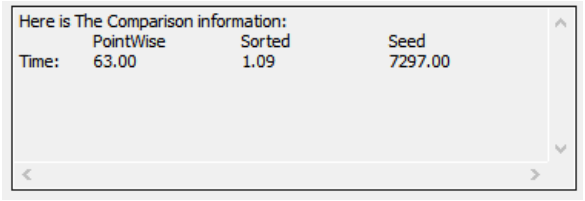

注：上述结果说明运行效率：SEED 算法(Seed) < PW 算法(PointWise) < SET 算法(Sorted)。显然，其中，SET 算法的效率比 SEED 算法快了不止一个数量级。

同时，由于 SEED 算法会可能导致栈溢出，因此，画的图形不能太大。这一点上述有提到。

**画凹多边形：**

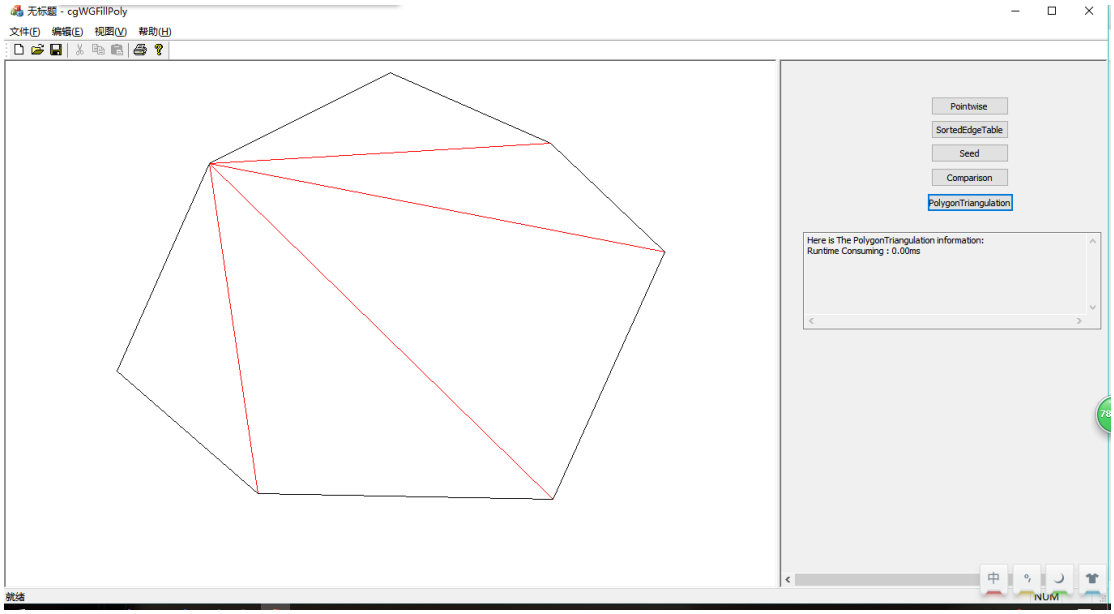



EditControl 控件 IDC_SI 放大：

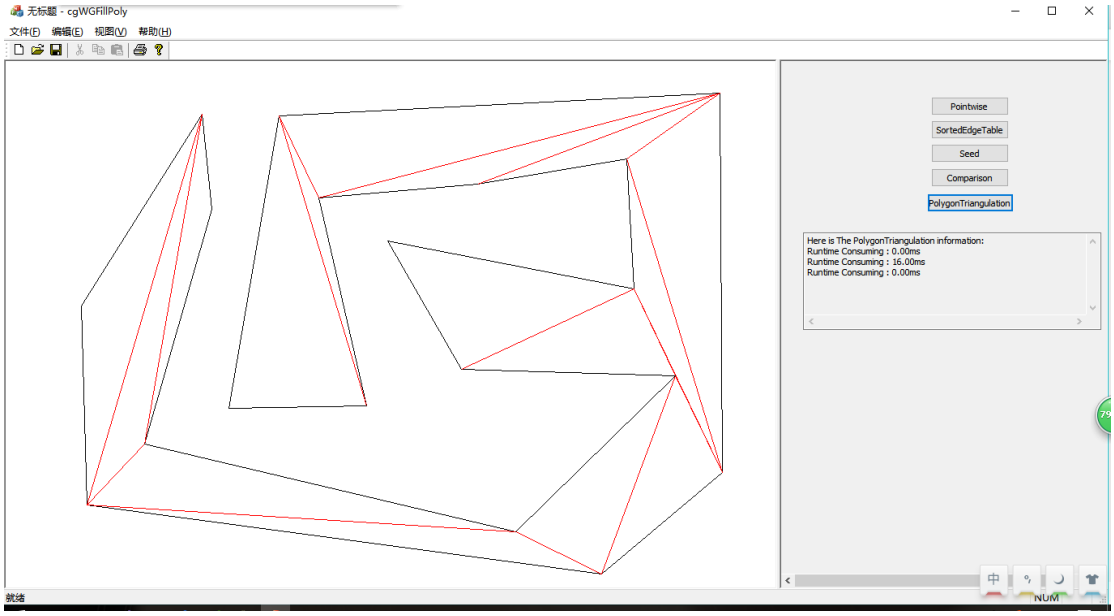



**6. 点击 PolygonTriangulation 按钮：**

**凸多边形的三角剖分：**




**画凹多边形的三角剖分（无论多复杂的都可以成功实现三角剖分）：**




注：从 EditControl 控件 IDC_SI 显示的 0.00ms（这里显示精度不够）可以看出，算法的运行效率也是非常快的。

这里我写的代码的时间复杂度是 O(n^3)，可以优化成时间复杂度为 O(n^2)，改动方法是：判定某个点是不是在当前三角形中时，我调用了 JudgeIn 函数（时间复杂度为 O(n)），而显然可以在 O(1)的时间复杂都内判定一个点是不是在三角形的内部。考虑到画的点太少，因此，在代码中，我直接使用了已经封装好的 JudgeIn 函数。

## 六．实验心得

“问渠那得清如许，为有源头活水来。”本实验成功地印证了这句古诗，这“源头”便是通过码代码发现自己的不足，并不断地去学习新的知识，改进自己的代码与程序。在写本实验程序的初步阶段， 程序中总是出现各种各样的 bug，比如，明明已经将图像填充好了，如果你把鼠标放到图像上静止两秒，那么鼠标所在的地方就会画出一条水平的白线。当我加上

if 条件句以后这种奇特无比的现象便会消失，究其主要原因是对 MFC 图像显示机制的理解不够深刻。对于程序的加分项也就是多边形的三角剖分，对于凸多边形来讲很简单，但是对于凹多边形来说就没有那么容易实现了。经过思考和上网搜索有关知识以后，写出了可以完美实现多边形三角剖分的算法，这也是本实验中，我最为引以为豪的地方了。

希望在以后的实验中，自己能够再接再厉，一直**认真**下去！

2018 年 11 月 7 日星期三