计算几何课程实验报告

题目：基于均匀平面划分的三角剖分算法实现与分析

Title：A fast polygon triangulation algorithm based on uniform plane subdivision

指导教师：邓俊辉

小组成员：曹媛媛　2004310410

柳佳　　2004310427

张荷花　2004310445

清华大学计算机系

2005 年 1 月 3 日

一．背景

多边形是计算机图形学和计算几何中得几何形状，实际应用中经常要求对多边形进行简单形状的划分，其中三角剖分最常用到。

多边形的三角剖分算法主要分为三类：基于对角线插入的方法、Delaunay 方法和使用 Seiner point 的方法，他们在计算时间和实现复杂度上各有千秋。在过去的十多年时间里，关于三角剖分领域有很多工作，meister 在 1975 年提出了第一个实现算法，他不断得搜索多边形的 ear,然后划分，他的算法复杂度为 O(n3)，在 1990 年 Elgindy 等改进了他的算法，使时间复杂度减少到 O(n2)，Garey 在 1978 年提出了一种分而治之的算法，首次将算法复杂度减少到 O(nlogn)。Kirkpatrik 等的算法继续将复杂度减少到了 O(loglogn)，并且数据结构也很简单。在 1990 年初期 Bernard 证明出任何简单的三角剖分都可以在 O(n)内完成。在实际应用中，不仅希望算法的复杂度较低，还希望它容易理解和实现，本实验实现的 Marko Lamot 算法就是一种有效且实现简单的算法。

二．算法描述

1．算法大体思想：

本算法结合了 EAR-CUTTING 算法和 DIAGONAL 算法,利用平面子区域划分提供性能:算法将多边形的顶点分为凹点和凸点两类，将凹点排列到对应的平面子区域中，将凸点存放到一链表中。算法依次从凸点链表中取点，判断应该使用 EAR-CUTTING 方法还是 DIAGONAL 方法，直到凸点链表为空。

2．算法步骤描述：

（1）首先进行初始化，包括一下几步：

A） 从界面获取多边形的顶点坐标以及连接关系，存入到我们的多边形类 MyPolygon 的实例中；

B） 对多边形进行调整，处理相应的退化情况（各退化情况在后面的部分中将提到），考察多边形中每个顶点的 Y 坐标，选择 Y 坐标最大的顶点，记做 P，作为该多边形的第一个顶点,然后,判断与 P 坐标相邻的两个顶点中，哪个是 P

顺时针方向的下一个顶点,从而对多边形的所有顶点按照顺时针排序.

C） 对多边形的顶点按照凸点和凹点进行严格的分类.组织凹点和凸点的数据结构。

D） 按照多边形的顶点位置,对多边形所在的区域作子区域划分,子区域划分是为了减少对凹点是否在三角形内的检查次数,该做法能提高性能,尤其适合于凹点在区域内分布比较均匀的情况.在我们的算法中,子区域的大小采用下面的公式来计算:

$$size = \frac{4}{3}\sqrt{\frac{(x_{max} - x_{min})(y_{max} - y_{min})}{n}},$$

其中 $x_{min}, x_{max}, y_{min}, y_{max}$ 表示 X 坐标的最小值、最大值、Y 坐标的最小值、最大值。

子区域大小的选择在当 n 较小的时候,体现不出性能的优劣,size 的大小依赖于顶点的规模已经凹点的分布.

按照上述公式选取子区域大小后,可以得到子区域的数目:

$$x_{res} = \left\lfloor \frac{x_{max} - x_{min}}{size} + 1 \right\rfloor,$$
$$y_{res} = \left\lfloor \frac{y_{max} - y_{min}}{size} + 1 \right\rfloor.$$

其中 x_{res} 和 y_{res} 分别表示多边形被分为的列和行的数目.

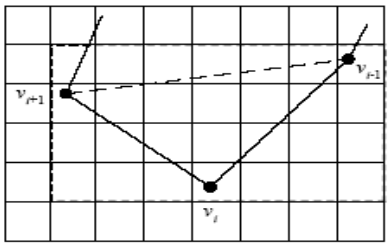
E） 将凹顶点通过下面的公式,确定到对应的子区域中:

$$i_{cell} = \left\lfloor \frac{x_i - x_{min}}{size} \right\rfloor,$$
$$j_{cell} = \left\lfloor \frac{y_i - y_{min}}{size} \right\rfloor.$$

x_i, y_i 表示凹点对应的坐标。

（2） 将多边形的凸点构造成一条链，从链尾中取出一个元素，记为 v_i，作为主顶点。

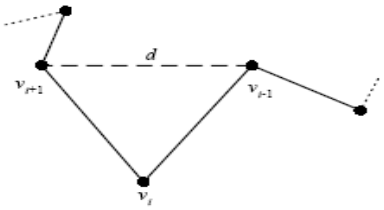
（3）找到 v_i 顶点的前驱顶点 v_{i-1} 和后继顶点 v_{i+1}，检查由这三个顶点构成的三角形 $t = \triangle v_{i-1} v_i v_{i+1}$. 中是否存在其他多边形顶点。



在该步骤中，我们实际不需要检查每一个多边形顶点，而只需要检查多边形的凹点，利用子区域平面划分，不需要检查多边形的任何一个凹点，而只需要检查三角形 t 对应的子区域内是否存在凹点，这样，根据步骤 1 中 D）即可得到。根据多边形中是否存在凹点，我们可以分为两种情况进行讨论：

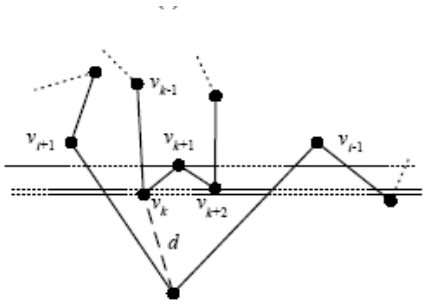
如果 t 中无凹点，按照 EAR-CUTTING 进行处理，为下面的步骤 4；如果存在一个或多个凹点，按照 DIAGONAL 方法进行处理，为下面的步骤 5。

(4) t 中无凹点，也表示 t 中无其他多边形顶点（如果存在顶点在 t 中，则一定包含一个凹点）。此时，t 可以看作是一个 ear，连接 v_{i-1} 和 v_{i+1}，将三角形 t 从多边形中切除，如图：



此时，应该从多边形中去掉顶点 v_i，判断顶点 v_{i-1} 和 v_{i+1} 的凹凸性，如果其中某个顶点由于三角形 t 从多边形中切除而从一个凹点变成了凸点，则将这个凹点从对应的数据结构中删除，并将该顶点加入凸链的尾端。接着执行第六步。

(5) 如果 t 中存在凹点，我们找到离线段 v_{i-1} v_{i+1} 最远的凹点，记为 v_k，连接 v_i 和 v_k 为一条对角线，这条对角线将多边形分为两个子多边形，初始化一个多边形实例进行存储，同时，将对角线根据不同的多边形分为两条，分别为 $v_{il}v_{kl}$ 与 $v_{ir}v_{kr}$，其中 v_{il} 和 v_{ir} 一定是凸点，而 v_{kl} 和 v_{kr} 中某一个可能是凹点，分类进行处理：



- 如果v_{kl}和v_{kr}都是凸点，则将v_{kl}和v_{kr}插入到凸点链表的前端，将v_{il}和v_{ir}插入到凸点链表的尾端。
- 如果v_{kl}和v_{kr}中v_{kl}为凹点，依次将v_{ir}和v_{il}插入到凸点链表的尾端，将v_{kr}插入凸点链表的前端，如果v_{kr}为凹点，依次将v_{il}和v_{ir}插入到凸点链表的尾端，将v_{kl}插入凸点链表的前端。

将原来存在于凸点链表中的v_i从凸点链表中删除。

(6) 判断凸点链表是否为空，若不空，执行第二步，若空，算法结束。

3．处理的退化情况包括：

(1) 输入多边形中，有多个顶点可以看作近似的重合，即两个顶点之间的线段退化为一个点。处理：在存入多边形实例时，将重合的多个点看作一个多边形顶点。

(2) 输入多边形中，三个或更多的顶点共线的情形。处理：存入多边形实例时，只考虑共线的多个顶点中处于线段两端的顶点，线段上的其他顶点忽略。

(3) 在检查三角形中是否存在其他顶点的情况下，当存在多个凹点，并且多个凹点到三角形的边v_{i-1} v_{i+1}距离相等，都是最远的情形。处理：计算这些点与顶点v_i之间的距离，选取与顶点v_i之间的距离最小的点。

三．系统设计和数据结构：

系统结构分为三个大模块：输入、数据处理、输出。

1．输入：

数据的输入包括三种方法：

- 用户自己描绘顶点
- 利用随机生成多边形算法
- 文件导入

我们通过存储多边形的所有顶点记录用户通过上述任一方式给出的多边形。多边形的所有顶点分别存储在数组 m_PolyGon 和链表 m_PointSet 中。其中，使用 m_PolyGon 检查多边形是否自交、在屏幕上显示多边形；使用 m_PointSet 作为输入和数据处理的接口。

（1）用户描绘顶点

通过单击左键确定一个顶点、移动鼠标画线、单击右键结束多边形的绘制。

(2) 随机生成多边形

这里又分为两种生成多边形方法。

- 用户输入多边形的顶点数，使用随机算法生成 x 单调的多边形。可以处理的多边形的最大规模为 10000 个顶点。
- 用户输入递归层数，初始形状设置为三角形，使用 Couch 分形算法生成星形多边形。允许的最大层数为 6，得到略小于 10000 个顶点的多边形。

(3) 文件导入

选择一个已有的多边形文件，利用 Serialize 函数导入多边形信息并显示多边形。

2．数据处理：

按照前面的算法描述，数据处理主要包括两大步骤：

- 根据输入得到的数据初始化各数据结构
- 根据凸点对应的数据结构，依次取点，循环执行剖分

这部分用到的主要数据结构包括：

1) 多边形 CMyPolygon：

我们定义了一个类为多边形，其中用了一个数组来存储所有多边形顶点的坐标，数组中相邻的序号对应的两点之间有一条边。存储顶点的数组的第一个元素具有最

大的 Y 坐标，其他的元素是按照顺时针的顺序依次存储的。在我们的算法中，构造多边形有两处，一处是初始化时，根据从界面传入的多边形顶点的坐标和连接关系；另外一处是在 DIAGONAL 方法中，连接一条 DIAGONAL，将一个多边形分解为两个，因此，我们根据这两种不同的情况，设计了两个对应的构造函数。另外，表示多边形的类中还有许多对多边形的操作：

- 删除多边形的一个顶点；
- 获取多边形顶点的个数；
- 给定顶点序号，判断某个顶点是否为凸点；
- 给定坐标值，判断是否为该多边形的一个顶点；
- 获取多边形所处区域，主要包括求对应 X 坐标的最大、最小，Y 坐标的最大、最小。
- 给定序号，求前驱顶点；
- 给定序号，求后继顶点；
- 处理三点共线的退化处理函数；
- 处理顶点重合的退化处理函数；

2) 对角线链表 CDiagonalList

该链表将 EAR-CUTTING 法和 DIAGONAL 法处理得到的顶点间的连线进行记录，这些连线即为对多边形进行剖分的线，将在输出部分进行显示。每条连线，即对角线链表的每个节点，都是一个结构 CDiagonal：

```
struct CDiagonal {
      POINT nFrom;
      POINT nTo;
      int iFrom;
      int iTo;
  };
```

记录的信息包括连线对应的两个多边形顶点的坐标和在多边形中的序号。

3) 凸点链表 ConcaveList

该链表在初始化后顺序的记载多边形中的所有凸点，随着剖分的进行，链表包含的节点数目将不断调整，算法完毕时链表为空。该链表的每个节点是一个表示凸点的结构：

```
struct Convex {
        int polyIndex;
        POINT convexValue;
        CMyPolygon  *relatePolygon;
              };
```

记录的信息包括对应在多边形中的序号，对应的坐标值和一指向相关多边形的指针。

4) 凹点链表 ConcaveList

在均有平面子区域划分时，每个子区域中可能包含多个凹点，我们将这些凹点组织为一个 ConcaveList。链表的每个节点是一个表示凹点的结构：

```
struct Concave {
        int subIndex;
        int polyIndex;
        POINT concaveValue;
        };
```

记录的信息包括对应的子区域序号、在多边形中的序号和该凹点对应的坐标。

5) 子区域链表 GridList

这个链表对所有包含了凹点的子区域进行记录，在初始化后，最初为凹点的所有点都将被记录到对应的子区域中，随着剖分的进行，凹点的数目不断减小， GridList 的长度也将不断减小，算法结束的时候，该链表的长度为 0。链表的每个节点是一个表示子区域的结构：

```
struct Grid{
        int label;
        ConcaveList *InconcaveList;
        };
```

记录的信息包括子区域的序号和一个指向该子区域中包含的所有凹点构成的链表的指针。

3．输出：

该模块对记录三角剖分信息的数据结构进行读取，进行中间过程及最终结果的显示。

主要的数据结构包括：

(1) 多边形的平面划分链表 m_PlaneGrid

这个链表记录输入的多边形对应的均匀平面划分，每个链表的结点类型如下：

```
typedef struct gridType
{   int number;
    int minX;
    int minY;
    int size;
} GridType;
```

每个节点记录了每个方格的坐标、方格编号以及方格大小。根据数据处理部分给出的方格编号访问此链表，即可得到方格的全部信息以便显示平面划分和活跃方格集。

(2) 单步信息存储数组 m_showInfoArray

为了单步方式显示三角剖分的中间过程，在数据处理部分进行三角剖分时，每执行一步的同时将当前处理的三角形、当前的动作类型、当前的活跃方格集、考虑的凹点、得到的对角线等信息存储在数组 m_showInfoArray 中，从而在输出阶段，只需访问此数组即可完成对中间过程的显示。数组成员的结构定义如下：

```
typedef struct stepShowType
{
    POINT v1;
    POINT v2;
    POINT v3;                       //当前三角形
    int flag;                       //flag = 0 表示切耳,flag=1 表示插入对角线
    int m_activegrid[MAXVERTICE];   //当前活跃方格集
    int m_activegrid_num;           //当前活跃方格集的数目
    POINT concavePoint;             //要考虑的凹点
    CDiagonal currentDiagonal;     //得到的对角线
} StepShowType;
```

(3) 利用数据处理阶段得到的对角线链表 allDiagonal，进行三角剖分最终结果的显示。

四．工作量分析

1．使用现成的源代码的部分：

- 检查多边形是否自交；
- 生成多边形用到的 Couch 分形算法；

2．借鉴已有代码，并进行改进的部分：

- 多边形的导入、导出；
- 判断多边形多个顶点是否共线的退化情况

3．独立实现的部分：（除 1 和 2 之外的所有部分）

主要包括：整体的逻辑；

数据结构的设计；

和平面划分相关所有处理；

输入、输出、中间过程的显示；

……

五．实现环境

程序设计语言：C++

编译器：　　Microsoft Visual Studio C++ 6.0

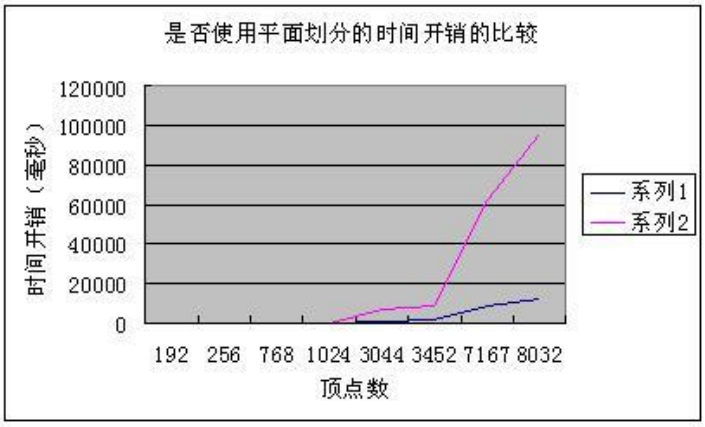
测试平台：　　Windows XP

机器配置：　　CPU P4 1.5G ， 内存 256M

六．测试和比较：

1． 时间开销随顶点数的变化

测试对象是利用 Couch 分形算法得到的星形多边形（图中红色表示不使用平面划分，蓝色表示使用平面划分）



由图可知，使用文章中给出的基于平面划分的三角剖分与不使用平面划分相比，时间性能有了很大提高。

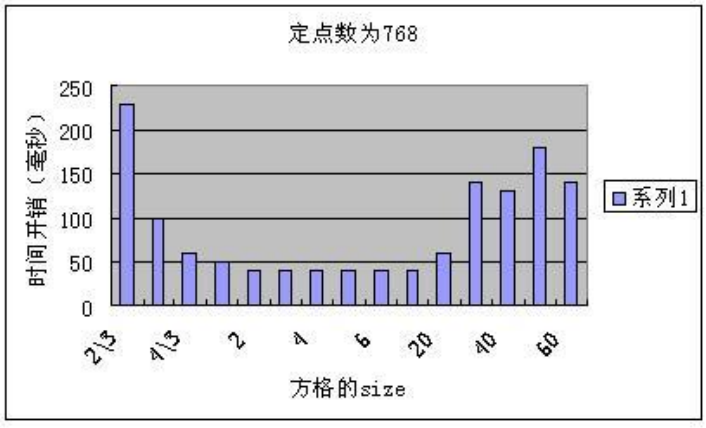
2 时间开销随平面划分粒度的变化曲线：

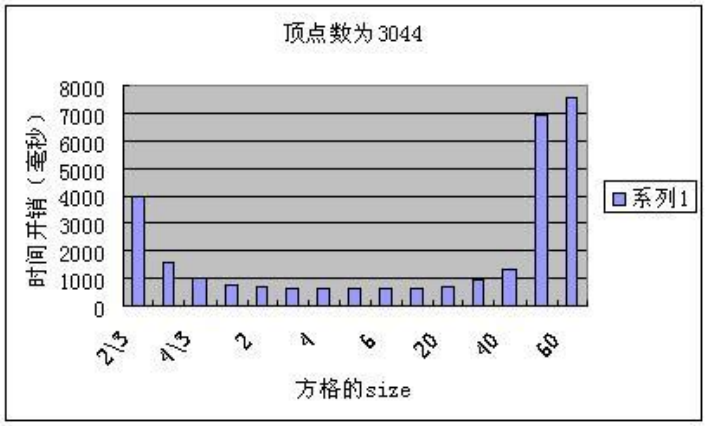
平面划分的作用是为了尽量减少需要检查的凹点，从而加快三角剖分的速度。然而当粒度太小或者太大时，三角剖分的时间开销反而很大。因为存在一个最佳的平面划分粒度。文章中进行均匀平面划分的方格大小计算公式为：

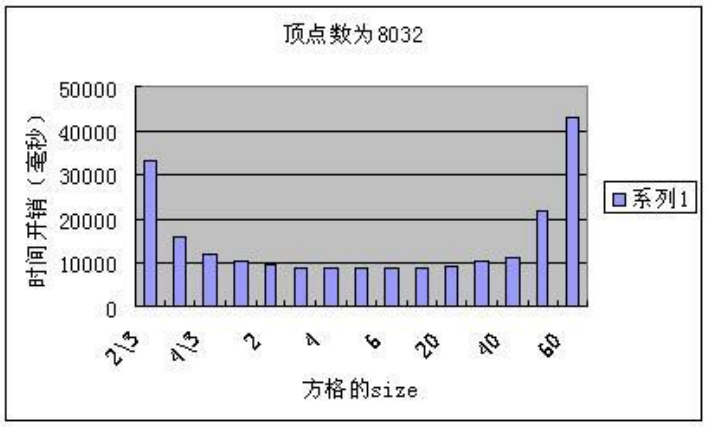
$$size = \frac{4}{3}\sqrt{\frac{(x_{max} - x_{min})(y_{max} - y_{min})}{n}},$$

这是作者经过实践得到的时间性能最好的平面划分粒度。针对我们实现时利用 Couch 分形算法得到的星形多边形，我们通过改变公式中的系数，来测试平面划分的粒度对时间性能的影响，如下图所示：

批注 [Jordan1]:
关键所在







由图可知：对于我们的实现而言，最佳粒度并不是来自系数取 4/3 时得到的平面划分，而是当系数取介于 4 到 6 之间的数时，三角剖分的性能可以达到最好。

批注 [Jordan2]:
实践总结的经验

七．需要改进的地方

1．时间效率

实验数据表明，我们的实现基本验证了利用均匀平面划分，确实可以提高三角剖分的速度，但是实现的时间效率，与文章作者的实现相比，还有一定差距。改进数据结构，并使用更快的查找算法（如 Hash 等）可以进一步改进时间性能。

2．随机算法

仍然没有找到一个非常好的随机算法，可以随机生成任意类型的多边形，因此实验结果不能全面的反映此算法的优劣。选取根据实际应用抽取出来的多边形或者使用 GIS 数据库中的多边形进行测试，更能说明一般情形。

3．测试用例的规模

目前我们的实现可以对顶点数在 10000 以内的任意多边形进行处理，虽然已经达到了文章作者使用的测试用例规模(1000~10000)，但仍然可以继续改进，使得算法对多于 10000 个顶点的多边形仍然可以使用。

参考文献

[1] Marko Lamot, Borut Zalik, A fast polygon triangulation algorithm based on uniform plane subdivision, Computers & Graphics 27 (2003) 239－253

[2] Zalik B, Clapworthy GJ. A universal trapezoidation algorithm for planar polygons. Computers & Graphics 1999;23(3):353－63.

[3] Seidel R. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. Computational Geometry:Theory and Applications 1991;1(1):51－64.

致谢