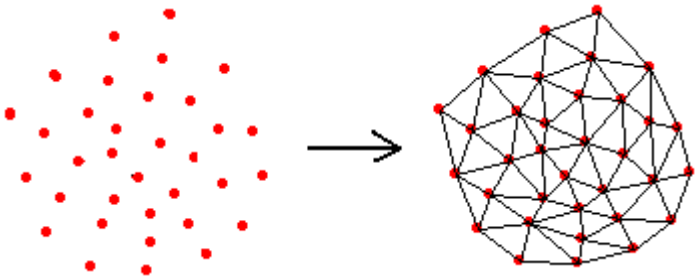
Delaunay 三角剖分

杨超群 周璟勇

1 理论背景

在图形学，材料计算，有限元结构分析，地球信息系统等领域，经常需要把采样得到的二维空间的散点，进行三角剖分。这里，三角剖分主要是作为一种预处理技术出现的。在所有的三角剖分中，Delaunay 三角剖分因为其最优性（最大程度避免了狭长三角形的出现）而得到广泛的重视。如下图所示：



关于更多 Delaunay 三角剖分的细节，由于课堂上讲述比较多，这里不再赘述。

2 算法介绍

在我们的程序里，为了提高算法的效率，使用了两种方法进行 Delaunay 三角剖分。

1、 若有初始点集（比如文件读入，随机生成），则使用 CGAA 上的随机增量算法【1】。但是，对于如何生成新的 Delaunay 三角剖分，则把随机增量算法的边翻转 flip 操作，进行替换，改用 Boywer-Waston 算法中的相关部分【2】。这样能使结构更加简洁。同时使用的数据结构也能得到化简。

2、 在 1 的基础上（这时候，如果没有 1，初始点集视为空集），如果要手动加入点的话，由于已经不能使用课堂上讲述的 bucketing/rebucketing 技巧。因此，只能进行全局搜索，但势必降低算法的效率。我们使用文献【3】提出的折线逼近方法，能迅速的找到点所在的三角形。遗憾的是，这种方法在高维空间中的推广效果并不好。在这个基础上，再次使用 Boywer-waston 算法。

下面我们进行详述。随机增量算法 CGAA 已经详述，这里不再赘述。下面，我们把新加入点在外接圆内的三角形称为目标三角形。

1、 Boywer-waston 算法

a) 概述

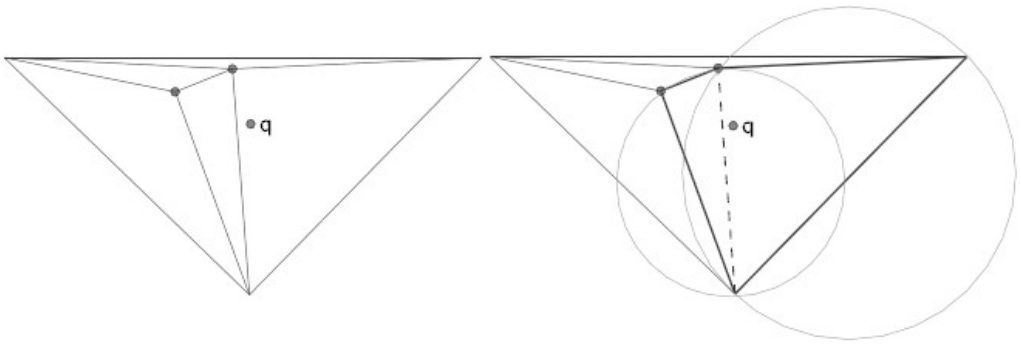
当第一个目标三角形找到的时候（在算法第一步，可以用 bucketing/rebucketing 技巧来实现，在第二步，可以用折线逼近方法实现），我们可以利用这个三角形作为出发点寻找该新加点在外接圆内的其他三角形。当获得所有的目标三角形时候，将新插入的点连接这些目标三角形的外边界（凸包）的各点，同时删除原来的目标三角形。

b) 具体步骤

Bowyer-Waston 算法仍然属于随机增量算法的一种。

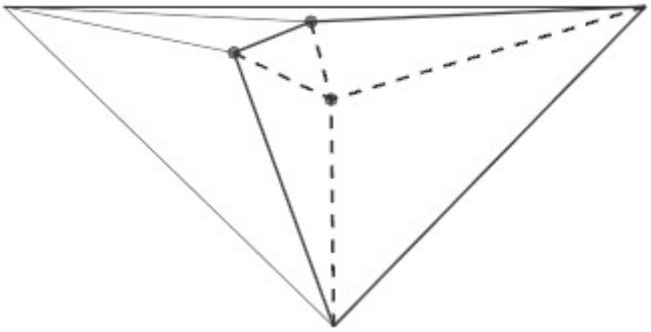
Bowyer-Waston 算法
● 构造一个超级三角形，包含所有待剖分的点集 V ● While（!V.isEmpty） 1、 将一个新点 p 插入到目前的剖分中 2、 找到外接圆包含 p 的三角形 3、 删除这些三角形，只留下这些三角形集合的外边界多边形 4、 简单的将 p 与多边形的各顶点连接即可 ● 删除与超级三角形顶点相关联的三角形

c) 图示



Bowyer/Watson算法，当把q插入一个三角剖分的时候，我们找到目标三角形集合，并且标记出插入多边形（图中黑线所示）

然后，连接q与插入多边形的各个顶点，如下图所示：



2、 折线逼近寻找目标三角形

Bucketing/rebucketing技巧，若能拥有节点位置的先验知识，那么就会快速的提高算法效率。但是，若没有这些先验知识，则就没法使用Bucketing/rebucketing技巧。因此，这时候的关键问题如何查询第一个目标三角形，也就是插入点q所在的三角形。方法有三种，依次如下列：

A) Brute-Force

规模较小时，此法较合适。

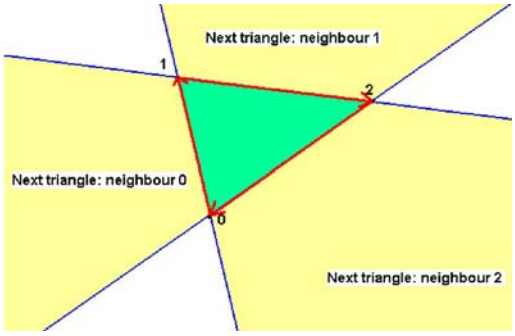
B) Brute-Force变种

由于各个剖分三角形在程序中是相邻的三角形有指针连接，因此，如果第一个查询的三角形靠近q所在的三角形的话，查询的速度就会加快。方法就是观察实际

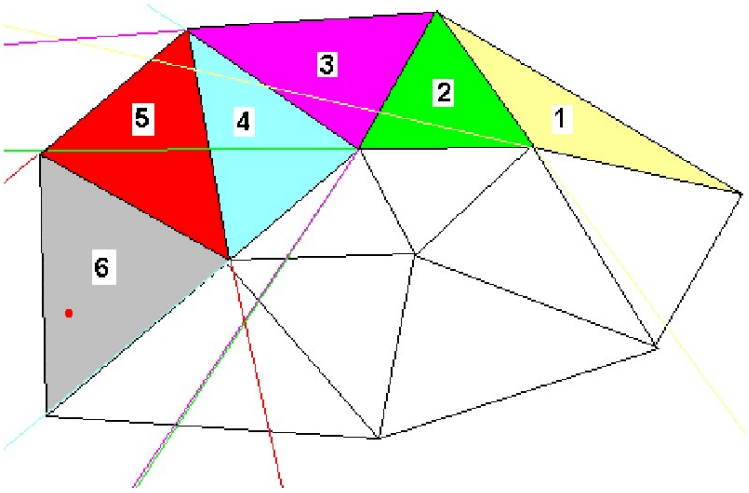
应用中，如果不是随机插入的话，基本上是相邻近的点依次插入的，因此只要找到上一个刚插入的三角形，以此为出发点，进行搜索就会加快速度。

C) B方法还是较慢，因为搜寻的方向是盲目的。因此，在B上继续改进，那就是增加搜寻的方向性。在二维空间中，这种方法是非常好的。高维空间进行推广效果就不太好了。如下图所示。

首先观察假设绿色区域不含点q，那么能含q的三角形一定在周围的6个区域，因此，我们要计算出q在哪个区域，然后才进行搜索。每次都这样迭代。



迭代的图示如下图所示：



假设三角形 1 为 start triangle，那么依次逼近，最后达到目标三角形 6，这就是新插入点 q（红点）所在的三角形。

3、 程序中使用的数据结构

本程序使用JAVA语言开发。开发环境为Eclipse IDE for Java Developers，Version: Helios Service Release 1，Build id: 20100917-0705

由于Delaunay三角剖分皆为三角形，因此没有使用更为一般化的DCEL数据结构，转为用三角图结构来代替。

a) 点

```
package utility;
public class Point2D
{
    private double x;
    private double y;
```

b） 三角形

```
public class Triangle extends ArraySet<Point2D>
{
    private Point2D circumcenter;
    private int id;
    private int idGen;

    @Override
    public int hashCode()
    {
        return (int)(id^(id>>>32));//Effective Java Programming Language Guide
    }
    @Override
    public boolean equals(Object o)
    {
        return (this==o);
    }
```

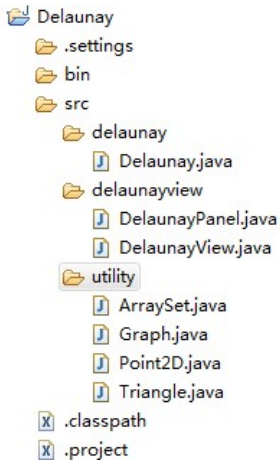
c） 三角图

```
public class Delaunay extends AbstractSet<Triangle>
{
    private Triangle mostRecentTriangle;
    private Graph<Triangle> delaunayTriangleSet;
    private ArraySet<Point2D> delaunayPoint;

    public  Map<Point2D, Triangle> bucket =
        new HashMap<Point2D,Triangle>();
    public Map<Triangle,ArraySet<Point2D>> bucket2 =
        new HashMap<Triangle,ArraySet<Point2D>>();
```

其中 Graph 类的结构如下：

```
public class Graph<T> {

    private Map<T, Set<T>> theNeighbors =    // Node -> adjacent nodes
        new HashMap<T, Set<T>>();
    private Set<T> theNodeSet =              // Set view of all nodes
        Collections.unmodifiableSet(theNeighbors.keySet());
```

程序结构分为三大部分，如下图所示。



delaunay 包进行核心算法计算。utility 包则提供基础的功能模块。delaunayview 包则是进行图形界面的显示工作。

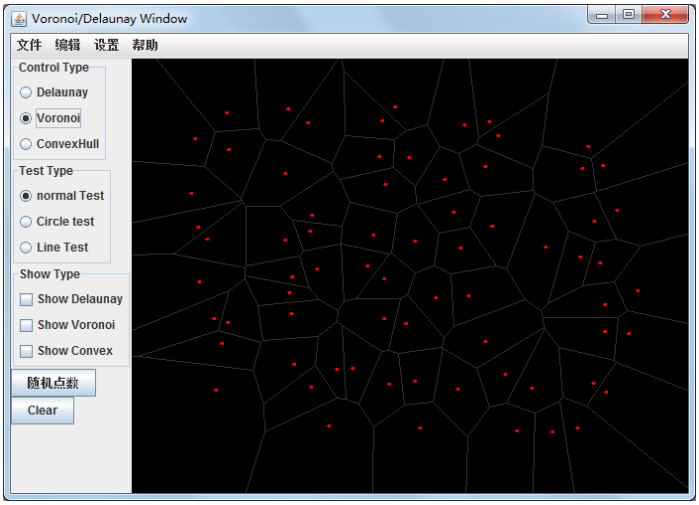
上述各个数据结构，欲详明了知，可以参看源代码。

3 运行效果

程序的主功能是 Delaunay 三角剖分，附属功能有依据对偶原理做出 Voronoi 图，凸包等。同时能支持进行圆测试，穿刺查询等功能。

剖分数据可以手动添加、随机生成和文件读取三种方式。

1、 程序界面



Control Type

1、Delaunay 2、Voronoi 3、ConvexHull

此栏目表示控制主界面显示类型

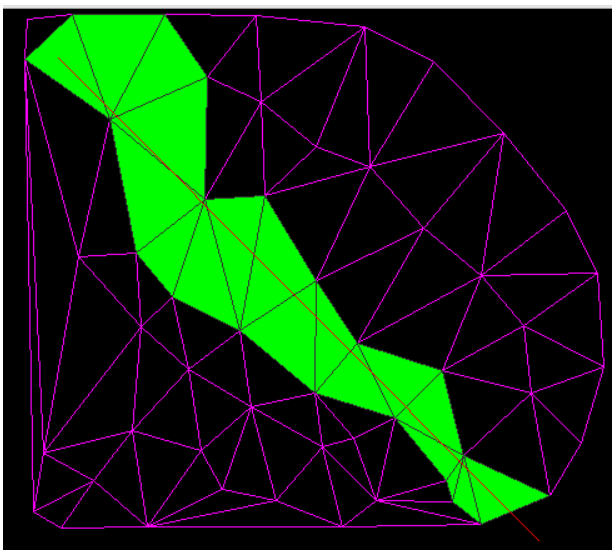
Test Type

1、 Normal Test :加点测试

2、 Circle Test：圆测试

3、 Line Test：穿刺查询

效果图



Show Type

1、 Show Delaunay :高亮 Delaunay 图

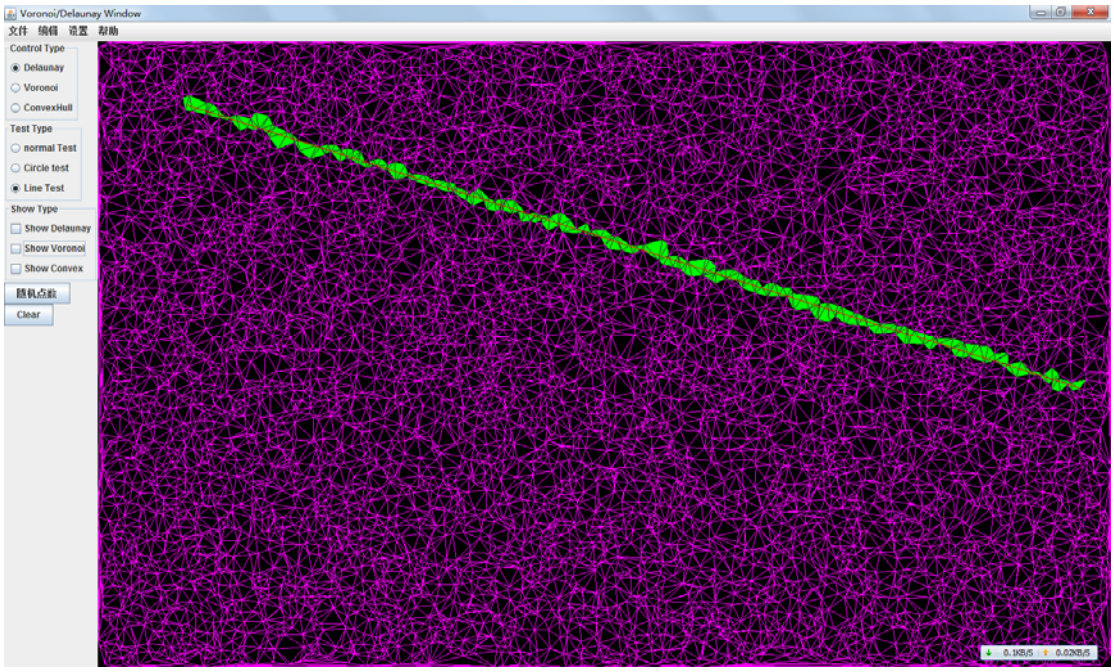
2、 Show Voronoi：高亮 Voronoi 图

3、 Show Convex ： 高亮 ConvexHull 图

随机点数

输入整数 N，则随机生成 N 个点。

当输入数位 10000 点的时候，如下图所示。



Clear

清除当前界面

4 工作总结

1、 代码写到最后的时候，由于本科时在电子系没有受到良好的开发训练，开始阶段比较结构化的结构，到后面开始杂乱无章。造成最后的调试经常出现 bug。调试的很辛苦。所以下一步仍然需要进一步的调整结构。

2、 令我们不解的是，就算输入 10000 点的时候，显示速度还是挺慢的（一分钟左右），这个时候我们采用的是随机增量算法。Bucketing/rebuckting 技巧并没有很好的帮我们降低时间。我们花了很多时间来探讨，却仍然没有得出结论。是不是与点的随机性有关？不得而知，这是需要下一步进一步完善的。

3、 在开发的过程中，深切体会到了视图与内核之间应该尽量的解耦合。同时，我们对 JAVA 语言的图形界面开发都不太熟悉，花了很多功夫进行再学习。

4、 虽然有成功的喜悦，但更多还有很多感觉需要提高的地方。一个深刻的经验就是：由于基础比较薄弱，造成从编程语言底层到抽象高层出现的问题层出不穷。这样，精力很难得到集中。希望以后的时间内，我们都能提高各个阶段的能力。

5、 在实现的算法和界面设计的过程中，参考了【4】的代码，以及其他前几届师兄们若干实现代码和界面。就不一一详举了。

参 考 文 献

【1】 CGAA

【2】 The Bowyer-Watson algorithm : C.A.Arens

【3】 Voronoi and Delaunay Techniques: Henrik Zimmer

【4】 http://www.cs.cornell.edu/home/chew/Delaunay.html