计算几何课程设计报告

基于分治法的 3D Delaunay Triangulation

Computer-. Aided Design 1998

2006310840　吴中

2006310839　董悦

引论

三角划分是计算几何的一个重要研究问题，并且在众多领域具有实际的应用价值。在计算机图形学、计算机视觉、图像合成等领域都有具体的应用。给定一组点的集合 P，Delaunay Triangulation 即基于这个点的集合构造一组三角剖分，这组三角剖分满足 Empty Circum-Circle 属性，即对于每个划分的 cell，其外接圆内部均不含有集合 P 中的点。

目前已经有很多算法可以解决二维、三维以及 n 维空间上的 Delaunay Triangulation 问题，本学期的《计算几何》课上介绍的 2D 情况下的 Delaunay Triangulation 的 Incremental 的算法便是其中一种。

在 3 维空间中的 Delaunay Triangulation 是计算 3D 流形网格上微分导数所必需的，在 3 维空间中的 Delaunay Triangulation 在计算机图形学，尤其是 Mesh 网格，曲面造型领域有相当广泛的应用。

P.Cignoni，C.Montani，R.Scopigno 在《DeWall: A Fast Divide & Conquer Delaunay Triangulation Algorithm in E^d》一文中提出了一种基于分治法的 Delaunay Triangulation 算法。该算法用一种 Divide & Conquer 的方法一般性的解决了 d 维空间中的 Delaunay Triangulation 问题，并可以在其中使用计算机图形学的一些加速技巧进行算法加速。

本学期我们计划实现这篇文章里提到的 Delaunay Triangulation 算法，并将其应用在 3D 的情况下，用可视化的方法显示出来，作为本学期的课程大作业。

算法简介

传统的分治算法如下：递归地将点集合拆分成互不相交的两个点集，分别在这两个点

集上进行局部的三角剖分，随后在合并阶段将三角剖分的结果接合起来。

当前的分治法的方法可以在 2 维空间中达到增量插入算法的复杂度，而在 d 维空间（d>2）上，分治法却难以实现。其主要的原因在于：在二维空间中，一个顶点上的边有明显的序关系，因此在二维空间上的合并十分简单；而在高维空间中的边没有这样的序关系，从而使得分治后的合并过程变得十分困难。

该文章提出的方法避开了这样的问题，因此可以很好的推广到 d 维空间上。该算法调换了分组和合并的次序，基于一个更复杂分组过程，参考增量构造三角剖分的方法，将输入点集合 P 分为子集 P1、P2 及一个可以直接构建 Delaunay Triangulation 的点集 E。（如图 1，图中 S^{α} 为点集 E，S^{-}、S^{+} 分别为点集 P1、P2）。在递归地对子集 P1 和 P2 进行 Delaunay Triangulation 后，用 E 将 P1、P2 的剖分结果边界结合起来，这样就避免了上面提到的高维情况下合并阶段的问题。该分组算法可以递归的在 P1 和 P2 中继续进行，并且在每次分组中都考虑到分组后局部 Delaunay Triangulation 结果的边界。

在该算法中，合并过程实际上是在分组之前就已经解决了。这个算法结构简单，容易实现，并且可以很容易的加入一些实现时的优化技巧。

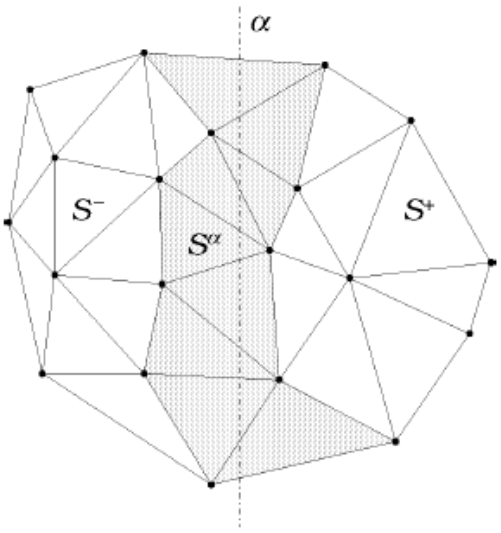

图 1

算法的详细介绍见[1]，在此不再赘述。

时间复杂度

虽然该算法在最坏情况下可能退化成普通的增量构造法，即在最坏情况下时间复杂度可能是$O(n^{\lceil \frac{d}{2} \rceil+1})$。但 P.Cignoni，C.Montani，R.Scopigno 的实验表明，平均意义下，在多数实际 3D Delaunay Triangulation 情况下，其实际时间复杂度约为$O(n^2)$。

目标

利用 C++在 3 维空间中实现《DeWall: A Fast Divide & Conquer Delaunay Triangulation Algorithm in E^d》一文中提出的基于分治法的 Delaunay Triangulation 算法。并通过 3D 图示化的形式将整个算法的过程进行演示。通过方便的用户交互，可以使得用户通过交互体会 Delaunay Triangulation 的相关性质，体会不同输入分布情况对 Delaunay Triangulation 结果的影响，输入变化时 Delaunay Triangulation 结果的对应变化等。通过交互式的展示算法的计算过程，使得算法的思路更加容易理解。

通过存储、载入特定数据集，可以方便的实现教学实例的展示。并可以实现在大数据集输入情况下的算法测试及演示。

具体实现与算法分析

1、基本结构

由于算法最后返回的是 simplex 的集合，并不记录它们内在的空间位置关系。因此，实现中均用点来表示与 simplex 相关的几何结构。4 个点表示一个 3D 的 simplex，3 个点表示 simplex 上的一个面，同时用记录 simplex 表面的外方向。为了方便求取点与平面的距离，算法中需要用到的分割平面α用其参数表示，即： ax + by + cz = d 。实现中还定义了 Vector3d 类用于进行向量的表示以及叉乘、内积等计算，用于通过向量运算判断点相对于面的空间位置；定义了 Record 类，记录每一层递归过程中所生成的 simplex 以及其分割平面α，用于对算法过程的演示。实现中所定义的基本结构如下：

```
class Vector3d
{
```

```
public:
    float x;
    float y;
    float z;
    //outer product
    Vector3d operator *(Vector3d &v2) {}
    //inner product
    float operator ^(Vector3d &v2) {}
    //opposite direction
    Vector3d operator -() {}
};
class Point3d
{
public:
    float x;
    float y;
    float z;
    bool operator ==(Point3d &point){}
    //minus of point
    Vector3d operator -(Point3d &point) {}
};

class Face3d
{
public:
    //use the equation : ax+by+cz=d  to specify a face in E(3)
    float a;    //coefficence of x
    float b;    //coefficence of y
    float c;    //coefficence of z
    float d;    //the constrast
    bool operator ==(Face3d &face) {}
};

class Simplex3
{
public:
    Point3d p1;
    Point3d p2;
    Point3d p3;
    Point3d p4;
    bool operator ==(Simplex3 &simplex) {}
    void Sort() {}
};
```

```
class SimplexFace
{
public:
    //use 3 points to denote a face of 3-simplex
    Point3d p1;
    Point3d p2;
    Point3d p3;
    //outer norm
    Vector3d outNorm;
    bool operator ==(SimplexFace &face) {}
        void Sort(){}
};

//record the procedure of DeWall algorithm
class Record
{
public:
        int iLevel;
        std::vector<Simplex3> simplexList;
        Face3d splitPlane;
};
```

该 DeWall 算法基于 Delaunay 的 Incremental Construction 算法，该算法在实现中需要维护一个 Active Face List（AFL），用于存放待处理的 simplex 面，并提供对 AFL 的 Insert, Extract, Delete, Member 操作。因此，[1] 中提出的 DeWall 方法同样需要维护这样的结构，且根据与分割平面 α 相交与否分别存在 AFL_1、AFL_α、AFL_2 三个结构中。对 AFL 结构的操作效率将影响算法的效率，可以采用 Hash 表等结构，使对 AFL 的操作效率基本保持为常数。但因为本实验初始目的是作一个 Demo 程序，演示用分治法进行 3D Delaunay 剖分的过程，且限于时间和精力，这次实验只给出另一个用线性结构实现 AFL 的程序。该实现在 AFL 中元素数量较多时，将使得算法效率降低，仍有可以改进之处。

2、主要算法实现

（1）、DeWall 算法

根据 Deluanay Triangulation 的性质，对于每一个 simplex，满足其外接球的空球性（即 simplex 的外接球内部不含其他的节点，且每一个 simplex 的表面最多只关联两个 simplex。该算法基本思想类似于 Incremental Construction 算法，从一个初始的 simplex 开始，依次遍

历其外表面，利用 Deluanay Triangulation 的空球性构造与该表面相关联的另一 simplex。每次生成一个 simplex，均将其未处理过的 simplex 表面作为 Active Face 推入 AFL 结构中，留待下一次循环处理。而 DeWall 算法与传统 Incremental Construction 算法不同之处在于，在每次递归中，DeWall 只处理与分割面相交的 Active Face，而将与分割面不相交的 Active Face 留待下一层递归进行。具体做法是将与分割面相交的 Active Face 推入 AFL_α，每次循环只从 AFL_α 中提取 Active Face；而与分割面不相交的 Active Face 则推入 AFL_1、AFL_2，作为参数传入调用下层递归。

DeWall 的算法伪代码详见文[1]， 在此不再赘述。

具体实现的函数定义如下：

```
vector<Simplex3> DeWall( IN vector<Point3d> pointSet,
                         IN vector<SimplexFace> ActiveFL,
                         IN vector<SimplexFace> &pastFaceList,
                         IN int iLevel, OUT vector<Record> &record);
```

（2）、第一个 Simplex 的构造

与 Incremental Construction 算法相同，作为预处理，DeWall 算法同样需要面对如何构造第一个 simplex 的问题。而相比与 Incremental Construction 算法，DeWall 算法在构造第一个 simplex 时还需要满足该 simplex 必须与分割平面相交。为此，采取构造方法如下：

<1> 选取离分割平面最近的点，作为 p1。

<2> 在分割平面的另一侧，选取离 p1 距离最近的点，为 p2。

<3> 遍历除 p1、p2 的其他点集，选取与 p1、p2 组成外接球半径最小的两点，为 p3、p4。

如此生成的四面体，可以保证是一个 simplex，同时与分割平面相交。说明如下：

由于 p1、p2 点分居分割平面的两端，因此 simplex 必然与分割平面相交。由于 Delaunay Triangulation 具有 Closest Pairs 性质，即每对最短距离的点必构成一条 DT 的边，而 p1 作为离平面最近的点，必有从 p1 出发的一条 DT 边穿过分割面到达另一端。因此，在分割面对侧寻找与 p1 距离最近的点 p2，则 p1p2 必是一个 DT 边。在此基础上求得的最小外接球满足空球性，组成一个 simplex。

该过程只需进行一次，可以视为预处理。时间复杂度为 $O(n^2)$，因此不会在数量级上

增加整体的时间复杂度。

程序中，函数定义如下：

```
//make the frist simplex
Simplex3 MakeFirstSimplex( IN vector<Point3d> pointSet, IN Face3d &alphaFace );
```

（3）、邻接 Simplex 的生成

程序中用 MakeSimplex() 函数生成与某一面相邻接的 simplex，文[1] 中定义了一种 simplex 表面与任一点的 Delaunay distance，如下：

$$dd(f,p)=\begin{cases} r, & c \in Halfspace(f,p) \\ -r & otherwise \end{cases}$$

其中，f 为 simplex 表面，p 为点，c 为 f 与 p 的外接球球心。

对于 AFL 中的 Active Face：f，遍历其外半平面上的点，求得与 f 的 Delaunay distance 最小的点 p，则 f 与 p 构成的四面体为 DT 中的一个 simplex。这样，可以保证每次生成的四面体均是 DT 集合中的一个 simplex。同时，该节点遍历过程同样可以用 Union grid 进行加速。

详细的论述见文[1]， 在此不再赘述。

相关函数如下：

```
//construction of the generic simplex
Simplex3 MakeSimplex( IN vector<Point3d> pointSet, IN SimplexFace face );
//calculate the Delaunay distance between a face and a point
float DelaunayDistance( IN Point3d point, IN SimplexFace face );
float DelaunayDistance( IN Point3d point, IN Point3d p1, IN Point3d p2 );
```

（4）、分割面的选择

每次递归中都需要选定一个分割面，具体实现中，返照 k－d tree 的方式，依次选取与各个坐标轴相垂直的分割面。分割面的位置由点集在对应坐标上的平均值决定。

相关函数定义如下：

```
//partition the point set into two sets, p1 and p2
void PointPartition(IN vector<Point3d> P, OUT Face3d &face,
                    OUT vector<Point3d> &P1, OUT vector<Point3d> &P2, IN int iLevel);
```

（5）、其他主要函数

程序中实现的其他主要函数及其用途列表如下：

```
//calculate the delta of n*n matrix
float Delta( IN float *pArray, IN int n);
//calculate the radius of circum circle of p1,p2,p3
float CircumCircleRadius( IN Point3d p1, IN Point3d p2, IN Point3d p3);
float CircumCircleRadius( IN Point3d p1, IN Point3d p2, IN Point3d p3, OUT Point3d &center);
//calculate the radius of circum sphere of p1,p2,p3,p4
float CircumSphereRadius( IN Point3d p1, IN Point3d p2, IN Point3d p3, IN Point3d p4);
float CircumSphereRadius( IN Point3d p1, IN Point3d p2, IN Point3d p3, IN Point3d p4, OUT
Point3d &center);
//extract simplex face from a simplex
vector<SimplexFace> GetFaces(IN Simplex3 t);
//judge if a simplex intersected with a face alpha
bool IsIntersected(IN SimplexFace simplex, IN Face3d face);
//judge whether a point is in list or not
bool IsInPointList(IN Point3d point, IN vector<Point3d> pointList);
//judge whether a point set is in list or not
bool IsInPointList(IN vector<Point3d> pointSet, IN vector<Point3d> pointList);
//return the union of simplex and simplexList
vector<Simplex3> UnionSimplex(IN vector<Simplex3> simplexList, IN Simplex3 simplex);
//return the union of simplex list and simplexList
vector<Simplex3> UnionSimplex(IN vector<Simplex3> list1, IN vector<Simplex3> list2);
//union a simplex face with a simplex face list
void UnionSimplexFace(IN SimplexFace face, IN OUT vector<SimplexFace> &faceList);
//judge if a face in face list
bool IsInFaceList(IN vector<SimplexFace> faceList, IN SimplexFace face);
```

交互式界面的开发

为了能够有效的展示 3D 空间中的 Delaunay Triangulation，我们使用了基于 OpenGL 的机制实现结果的实时渲染。同时，我们还采用 The OpenGL Utility Toolkit (GLUT)实现交互界面。

结果分析

程序运行界面如图：

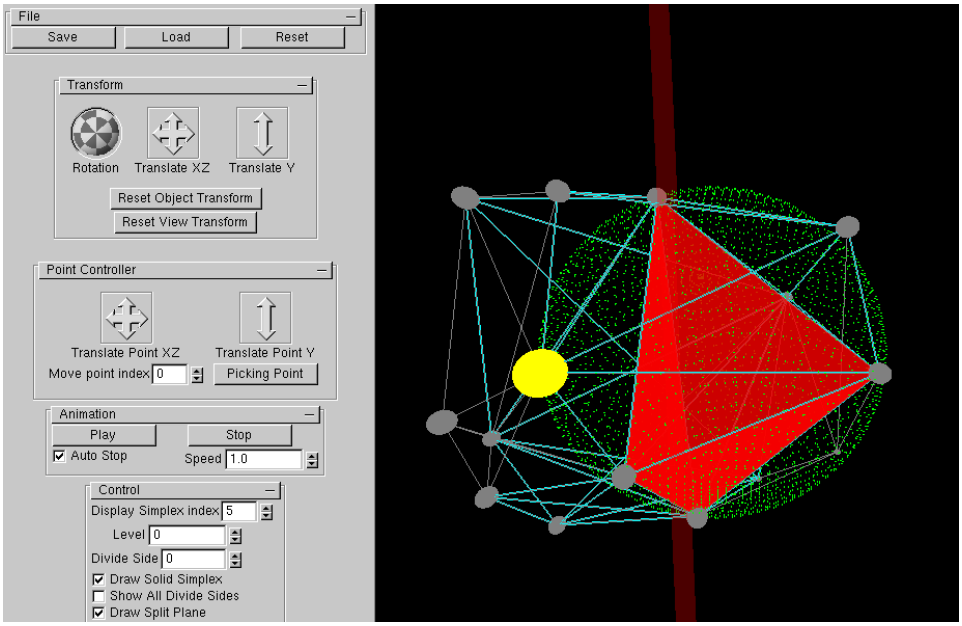

图 1　15 个点

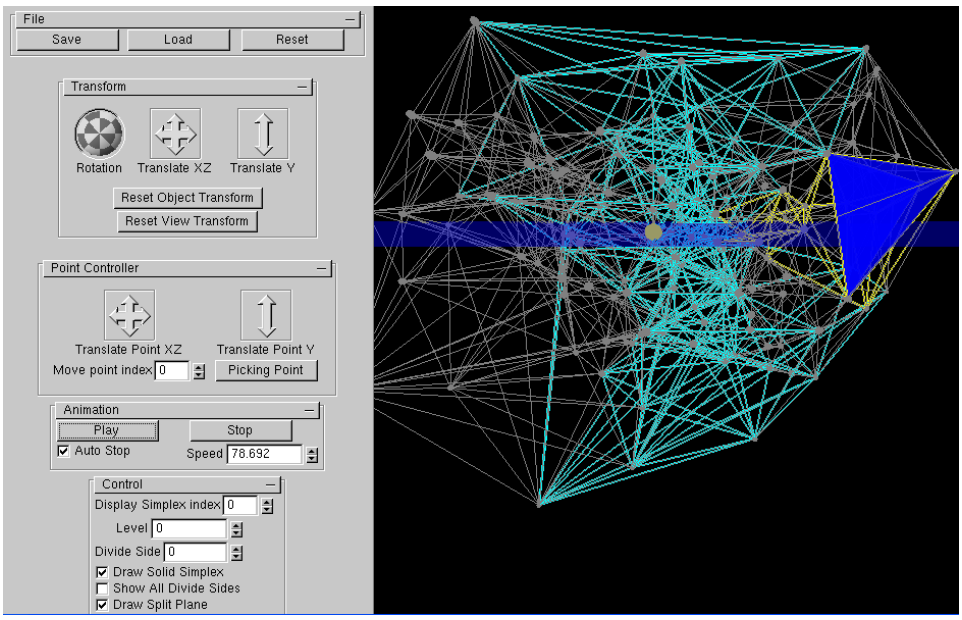

图 2　150 个随机点

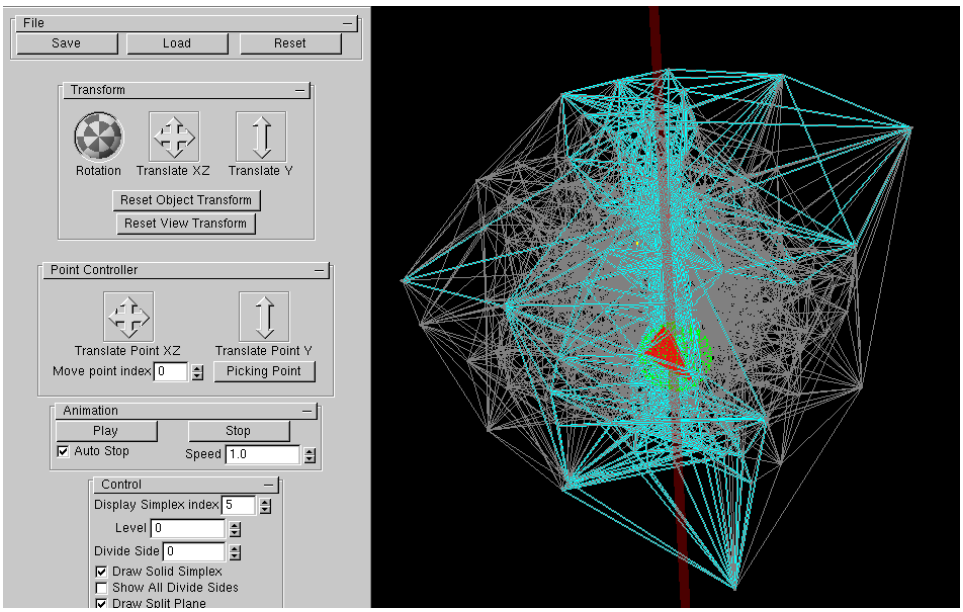

图 3　1000 个随机点

图中的半透明平面表示该递归层的分割曲面，红色的四面体为当前显示的 simplex，而绿色的虚线球则表示当前 simplex 的外接球，可以通过该外接球来验证 Delaunay Triangulation 的空球性质。

经验证程序能进行正确的 3D Delaunay Triangulation，在点数较多的情况下可以较清楚的演示 DeWall 算法的剖分过程。

算法效率上，由于程序实现时候有一些优化算法未能实现，同时由于 3D 情况下 Delaunay Triangulation 具有较高的时间复杂度，程序在计算 1000 个随机点以上的情况时已经比较吃力。可以通过一些改进来解决该问题，具体可采用的做法将在下文叙述。

编译安装说明

工程文件可以直接在 Microsoft Visual Studio 2005 中打开并直接编译。GLUT 相关的头文件(glut.h)已经包含在工程中。GLUT 编译需要的库文件(glut32.lib)也已经包含在工程中。GLUT 动态链接库 glut32.dll 包含在工程文件中，运行时需要手工拷贝到程序运行目录。GLUT 的相关说明详见 http://www.opengl.org/resources/libraries/

批注 [J1]: 未找到
随便下载了一个，似乎还能使用，尽管不知道是否有版本问题

程序使用说明

1、选取 3D 空间中的点

用户可以通过点击 Picking Point 按钮开始在 3D 空间中拾取点。此时 Picking Point 按钮变成 Stop Picking Point，即进入选取点状态，用户可以在右侧屏幕上选取点。选取完成，点击 Stop Picking Point 结束选取点。

2、浏览 3D 空间

用户可以用鼠标左键拖拽来实现视点平移，右键拖拽实现视点旋转，中键拖拽实现视点缩放。同时，用户还可以使用 Translate 页面中的三个调整按钮来实现物体的平移和旋转。

3、移动输入的点

用户可以通过 Point Controller 页面中 Translate Point XZ,以及 Translate Point Y 两个拖动按钮调整 3D 空间中的点的位置。用户当前可调整的点用黄色显示，用户可以通过 Move Point index 来改变当前选择并进行编辑的点。

4、计算 Delaunay Triangulation

每当新的点输入或有点被移动时，系统会自动计算 Delaunay Triangulation，并实时的进行显示。

5、显示的 Delaunay Triangulation 结果

在右侧 3D 空间中，显示了 Delaunay Triangulation 的结果。其中小的圆球代表了用户输入的点集合。其间的连线即 Delaunay Triangulation 结果。绿色虚线构成的球为 Simplex 的外接球，用这个球可以观察并检验 Delaunay Triangulation 应具有的空球性质。

6、显示某一 Simplex

在右侧以红色实心显示的 Simplex 是当前显示的 Simplex。用户可以通过调整 Control 页面中 Display simplex index 以及 Level、Divide Side 来选择显示某一个 Simplex。其中 Level 代表 Divide and conquer 迭代的深度，Divide Side 表示每次迭代时的不同分支，

simplex index 代表在这个迭代分支中被刨分出的第 n 个 simplex。

同时，当前刨分所利用到的平面也会以半透明的形式显示。图 1 中，红色的平面即当前迭代时所采用的平面。

不同的 Level 将会采用不同颜色绘制平面和对应的 simplex,同时，当前 Level 生成的 simplex 所构成的边会用粗线来表示。

7、动画显示 Delaunay Triangulation 过程

用户可以通过点击 Play 键来开始动画显示 Delaunay Triangulation 过程，点击 Pause 可以暂停，并继续播放。用户可以实时的通过改变 Speed 来改变动画播放的速度。当 Auto Stop 选中时，播放完成程序会自动停止，否则，程序将停在最后一步迭代的最后一个 Simplex 处，直到用户点 Stop 停止为止。

下来的工作

本次实验中限于我们的时间和精力，对算法中的一些结构和过程并未作太多的优化，因此，还有不少可以改进之处，可提高算法效率。

1、 用 Hash table 组织 AFL。目前的线性结构在 AFL 的 Member 操作中耗费了额外的时间，通过用 Hash table 组织 AFL，可以期望将该操作时间降为常数量级。

2、 用类似于 ray tracing 中所用的 union grid 方法划分空间点集，则在生成 simplex 的时候，可以只考虑空间中半平面中的点，避免对全空间点集的遍历。

通过上述改进，应该可以使得该算法的计算效率得到不少提升。

同时，在 UI 上，我们尝试加上 shading 提升显示效果，但实际效果并不是很理想，仍可以继续考虑如何使得显示过程更为清晰。

主要参考文献

[1] P.Cignoni, C.Montani, R.Scopigno. DeWall:: A Fast Divide & Conquer Delaunay Triangulation Algorithm in Ed. Computer-. Aided Design 1998

[2] P. Su and R. L. Scot Drysdale. A comparison of sequential delaunay triangulation algorithms. In 11th ACM Computational Geometry Conf. Proc (Vancouver, Canada), pages 61-70. ACM Press, 1995.

[3] Junhui Deng.《计算几何》课件, 2006