Delaunay Triangulation in OpenCascade

eryar@163.com

摘要：本文简要介绍了 Delaunay 三角剖分的基础理论，并使用 OpenCascade 的三角剖分算法将边界 BRep 表示的几何体进行三角离散化后在 OpenSceneGraph 中显示。

关键字：Delaunay Triangulation、OpenCascade、OpenSceneGraph

一、概述

三角剖分是平面剖分中的一个重要课题，在数字图像处理、计算机三维曲面造型、有限元计算、逆向工程等领域有着广泛应用。由于三角形是平面域中的单纯形，与其他平面图形相比，其有描述方便、处理简单等特性，很适合于对复杂区域进行简化处理。因此，无论在计算几何、计算机图形处理、模式识别、曲面逼近，还有有限元网格生成方面有广泛的应用。

虽然曲线、曲面等有精确的方程来表示，但是在在计算机中，只能用离散的方式来逼近。如曲线可用直线段来逼近，而曲面可用多边形或三角形来表示。用多边形网格表示曲面是设计中经常使用的形式，可以根据应用要求选择网格的密度。利用三角形面片表示的曲面在计算机图形学中也称为三角形网格。用三角形网格表示曲面需要解决几个问题：三角形的产生、描述、遍历、简化和压缩等，这些问题都是计算几何研究的范畴，相关问题都可以从中找到答案。下图所示的圆柱和立方体是由 OpenCascade 生成，使用 OpenCascade 的算法离散成三角网格后在 OpenSceneGraph 中显示的效果。

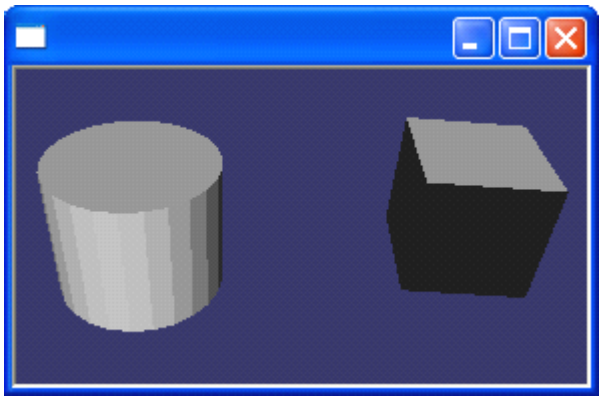

Figure 1.1 Shaded Cylinder and Box

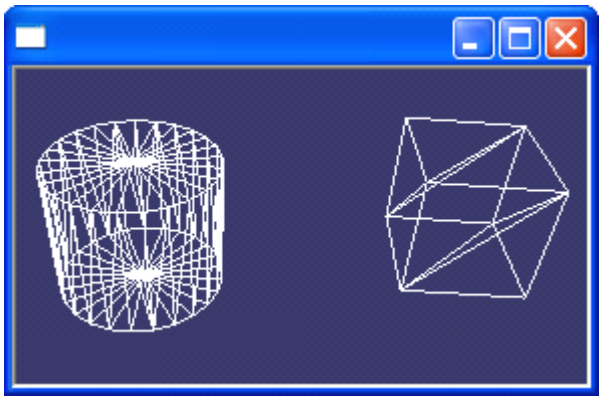

Figure 1.2 Mesh generated by OpenCascade

从图中可以看出，平面的三角形网格效果还不错，曲面的三角形网格表示只能是近似表示，可以通过提高网格的密度来增加真实性，但相应渲染的数据量就大了。有人说 OpenCascade 的显示模块做得不是很好，上述方法则可以只使用 OpenCascade 的造型模块，再结合 OpenSceneGraph 来对图形进行显示。

三维数据交换 STL 格式文件中保存的都是三角面片的数据，STL 文件格式是由美国 3D System 公司开发，已被工业界认为是目前快速自动成型领域的准标准零件描述文件格式。它对三维实体描述的解释具有惟一性。几乎所有的几何造型系统都提供 STL 文件数据交换接口。OpenCascade 中的数据交换模块也提供对 STL 格式的支持，由此可见三角网格在几何造型系统中的重要性。

Voronoi 图和 Delaunay 三角剖分的应用领域十分广泛：几何建模——用来寻找三维曲面“好的”三角剖分；有限元分析——用来生成“好的”有限元网格；地理信息系统——用来进行空间领域分析；结晶学——用来确定合金的结构；人类学和考古学——用来确定氏族部落、首领权威、居住中心或堡垒等的影响范围；天文学——用来确定恒星和星系的分布；生物学生态学和林学——用来确定动植物的竞争；动物学——分析动物的领地；统计学和数据分析——用来分析统计聚合；机器人学——用来进行运动轨迹规划（在存在障碍物的情况下）；模式识别——作为寻找物体骨架点的工具；生理学——用来分析毛细作用的领域；气象学——用来估计区域平均降雨量；市场学——用来建立城市的市场辐射范围；以及在遥感图像处理、化学、地理学、地质学、冶金学、数学等学科的应用等。

本文只对 OpenCascade 中的三角剖分进行简要介绍，希望对三角剖分在三维几何造型方面有兴趣的朋友可以对其深入研究。水平很有限，文中不当之处欢迎批评指正、指导，联系邮箱：eryar@163.com。

二、Voronoi 图

Dirichlet 于 1850 年研究了平面点的邻域问题，Voronoi 于 1908 年将其结果扩展到高维空间。半空间定义 Voronoi 图：给定平面上 n 个点集 S，S={p1, p2, …, pn}。定义：

$$V(p_i) = \mathop{\mathrm{I}}_{i \neq j} H(p_i, p_j)$$

PiPj 连线的垂直平分面将空间分为两半，V(Pi)表示比其他点更接近 Pi 的点的轨迹是 n-1 个半平面的交，它是一个不多于 n-1 条边的凸多边形域，称为关联于 Pi 的 Voronoi 多边形或关联于 Pi 的 Voronoi 域。如下图所示为关联于 P1 的 Voronoi 多边形，它是一个四边形，而 n=6.

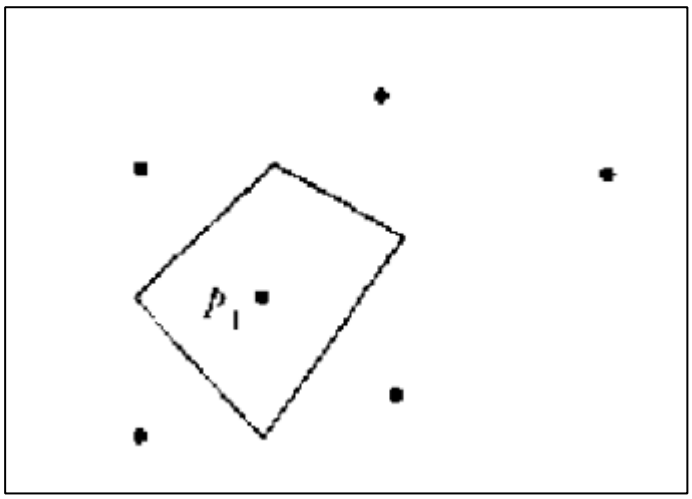

Figure 2.1 n=6 时的一种 V(p1)

对于点集 S 中的每个点都可以做一个 Voronoi 多边形，这样的 n 个 Voronoi 多边形组成的图称为 Voronoi 图，记为 Vor(S)。如下图所示：

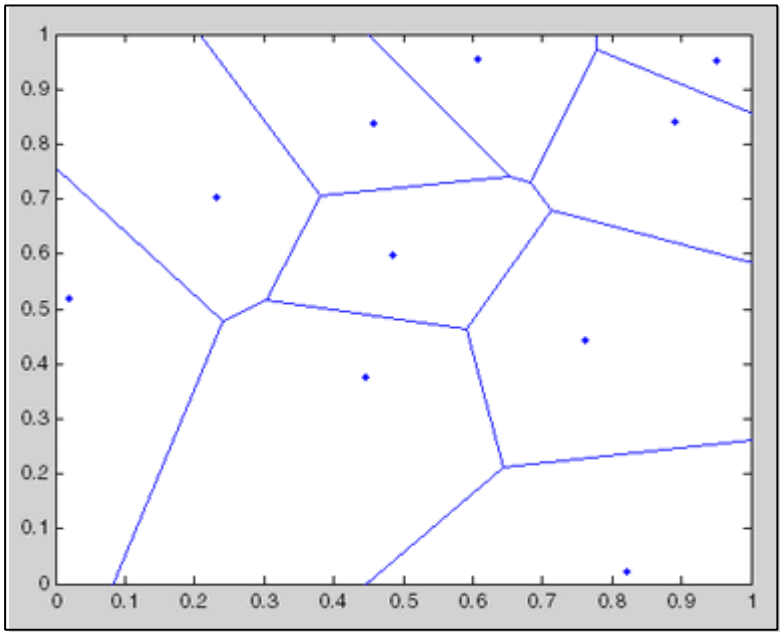

Figure 2.2 Voronoi diagram for 10 randomly points (Generated by MATLAB)

图中的顶点和边分别称为 Voronoi 顶点和 Voronoi 边。显然，|S|=n 时，Vor(S)划分平面成 n 个多边形域，每个多边形域 V(Pi)包含 S 中的一个点而且只包含 S 中的一个点，Vor(S)的边是 S 中某点对的垂直平分线上的一条线段或半直线，从而为该点对所在的两个多边形域所

共有。Vor(S)中有的多边形域是无界的。

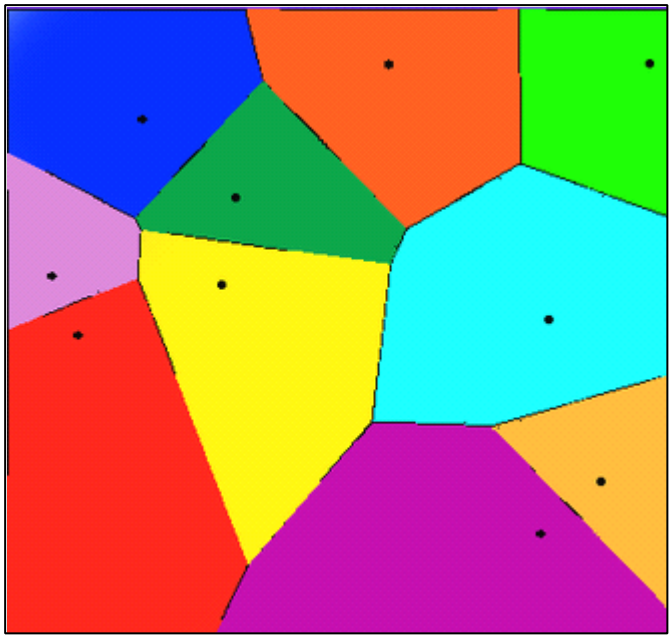

Figure 2.3 Ten shops in a flat city and their Voronoi cells
(http://en.wikipedia.org/wiki/Voronoi_diagram)

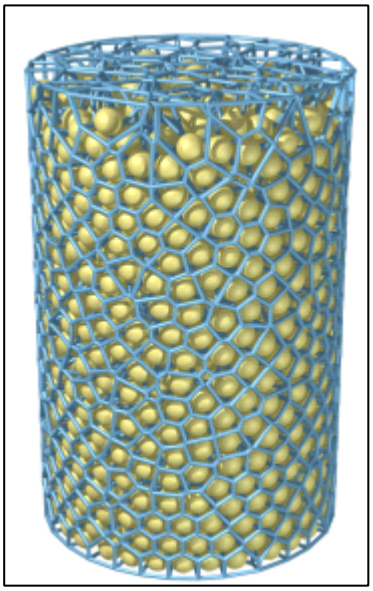

Figure 2.4 Voronoi tessellation in a cylinder (Voro++ library: http://math.lbl.gov/voro++/)

Voronoi 图有如下性质：

- n 个点的点集 S 的 Voronoi 图至多有 2n-5 个顶点和 3n-6 条边；
- 每个 Voronoi 点恰好是三条 Voronoi 边的交点；
- 设 v 是 Vor(S)的顶点，则圆 C(v)内不含 S 的其他点；
- 点集 S 中点 Pi 的每一个最近邻近点确定 V(Pi)的一条边；
- Voronoi 图的直线对偶图是 S 的一个三角剖分；
- 如果 Pi,Pj 属于 S，并且通过 Pi,Pj 有一个不包含 S 中其他点的圆，那么线段 PiPj 是点集 S 三角剖分的一条边，反之亦成立。

三、Delaunay 三角剖分

1. 二维实数域上的三角剖分

假设 V 是二维实数域上的有限点集，边 e 是由点集中的点作端点构成的封闭线段，E 为 e 的集合，那么该点集 V 的一个三角剖分 T＝（V，E）是一个平面图：

- 除了端点，平面图中的边不包含点集中的任何点；
- 没有相交边；
- 平面图中所有的面都是三角面，且所有三角面的合集是点集 V 的凸包。

2. Delaunay 边

假设 E 中的一条边（两端点 a,b），e 满足下列条件，则称为 Delaunay 边：存在一个圆经过 a，b 两点，圆内不包含点集 V 中的任何的点。这一特性又称为空圆特性。

3. Delaunay 三角剖分

如果点集 V 的一个三角剖分 T 中只包含 Delaunay 边，那么该三角剖分称为 Delaunay 剖分。

最近点意义下的 Voronoi 图的对偶图实际上是点集的一种三角剖分，该三角剖分就是 Delaunay 剖分（表示为 DT(S)），其中每个三角形的外接圆不包含点集中的其他任何点。因此，在构造点集的 Voronoi 图之后，再作其对偶图，即对每条 Voronoi 边作通过点集中某两点的垂直平分线，即得到 Delaunay 三角剖分。

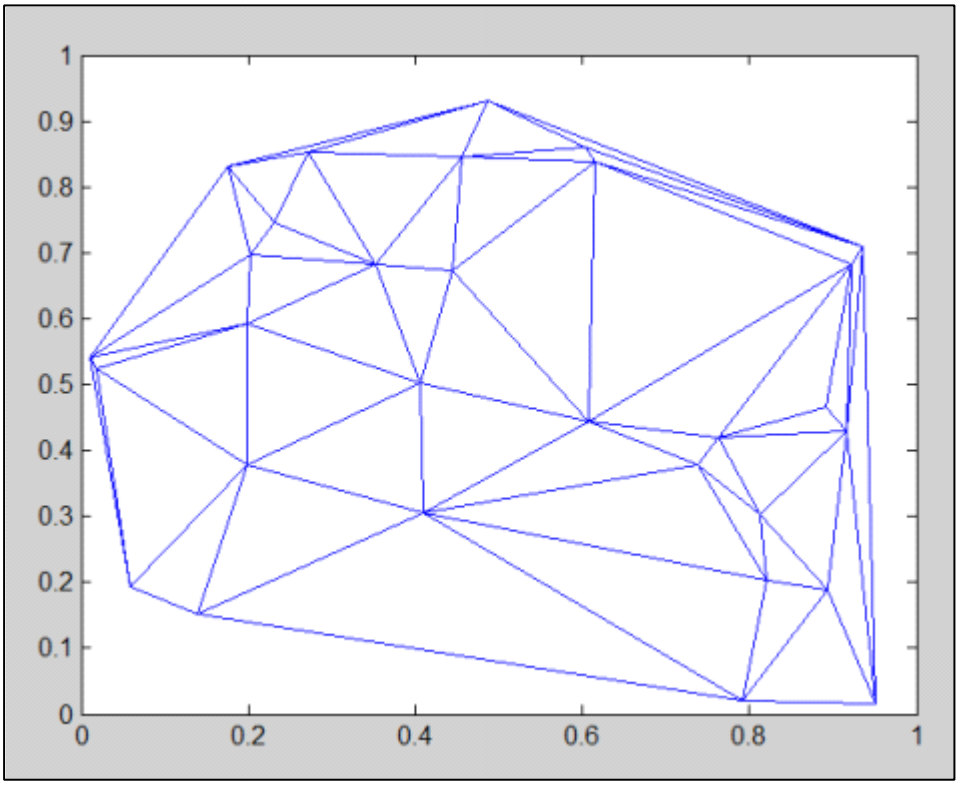

Figure 3.1 Delaunay Triangulation (Generated by MATLAB)

再看几个图片，加深对 Delaunay 三角剖分的理解：

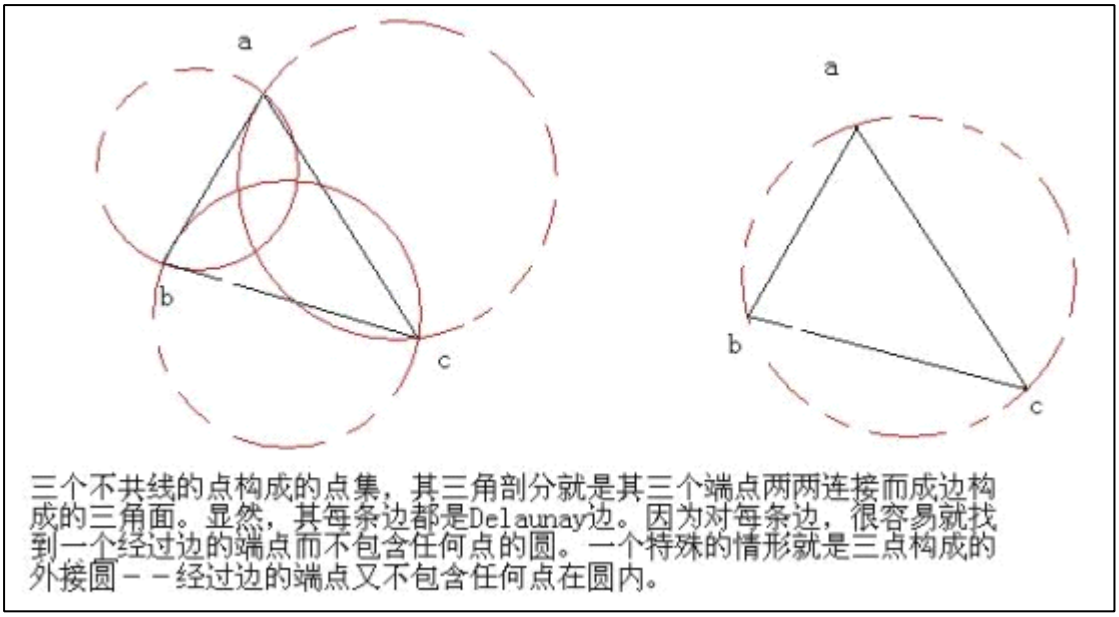

Figure 3.2 Delaunay Edge

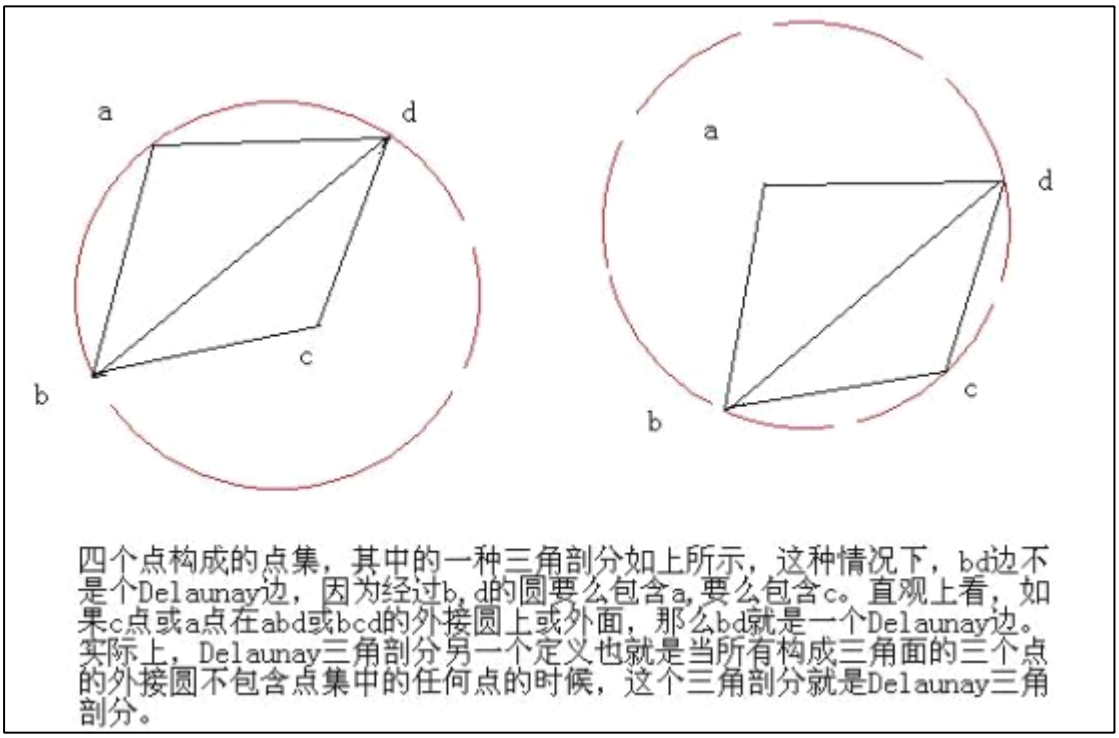

Figure 3.3 Illustrate Delaunay Edge

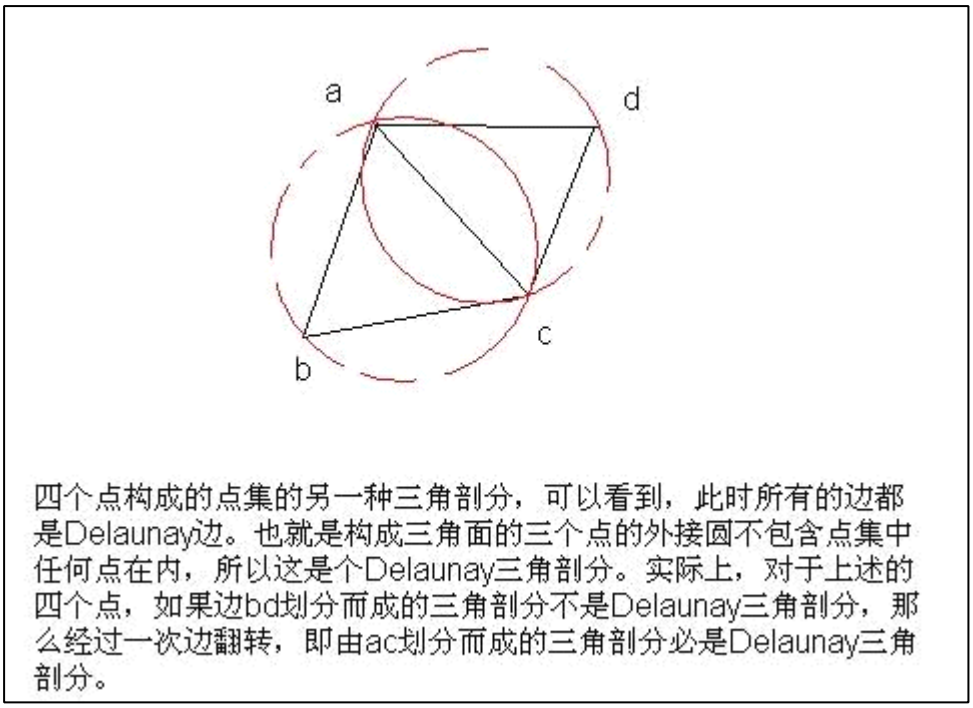

Figure 3.4 Delaunay Edge

4. Delaunay 三角剖分的特性

- 1978 年 Sibson 证明了在二维的情况下，在点集的所有三角剖分中，Delaunay 三角剖分使得生成的三角形的最小角达到最大（max-min angle）。因为这一特性，对于给定点集的 Delaunay 三角剖分总是尽可能避免“瘦长”三角形，自动向等边三角形逼近；
- 局部优化与整体优化（locally optimal and globally optimal）；
- Delaunay 空洞（cavity）与局部重连（local reconnection）；

5. 经典的 Delaunay 三角剖分算法

目前常用的算法分为几种，有扫描线法（Sweepline）、随机增量法（Incremental）、分治法（Divide and Conquer）等。

经典的 Delaunay 三角剖分算法主要有两类：Bowyer/Watson 算法和局部变换法。

- Bowyer/Watson 算法又称为 Delaunay 空洞算法或加点法，以 Bowyer 和 Watson 算法为代表。从一个三角形开始，每次加一个点，保证每一步得到的当前三角形是局部优化的。以英国 Bath 大学数学分校 Bowyer，Green，Sibson 为代表的计算 Dirichlet 图的方法属于加点法，是较早成名的算法之一；以澳大利亚悉尼大学地学系 Watson 为代表的空外接球法也属于加点法。加点法算法简明，是目前应用最多的算法，该方法利用了 Delaunay 空洞性质。Bowyer/Watson 算法的优点是与空间的维数无关，并且算法在实现上比局部变换算法简单。该算法在新点加入到 Delaunay 网格时，部分外接球包含新点的三角形单元不再符合 Delaunay 属性，则这些三角形单元被删除，形成 Delaunay 空洞，然后算法将新点与组成空洞的每一个顶点相连生成一个新边，根据空球属性可以证明这些新边都是局部 Delaunay 的，因此新生成的三角网格仍是 Delaunay 的。

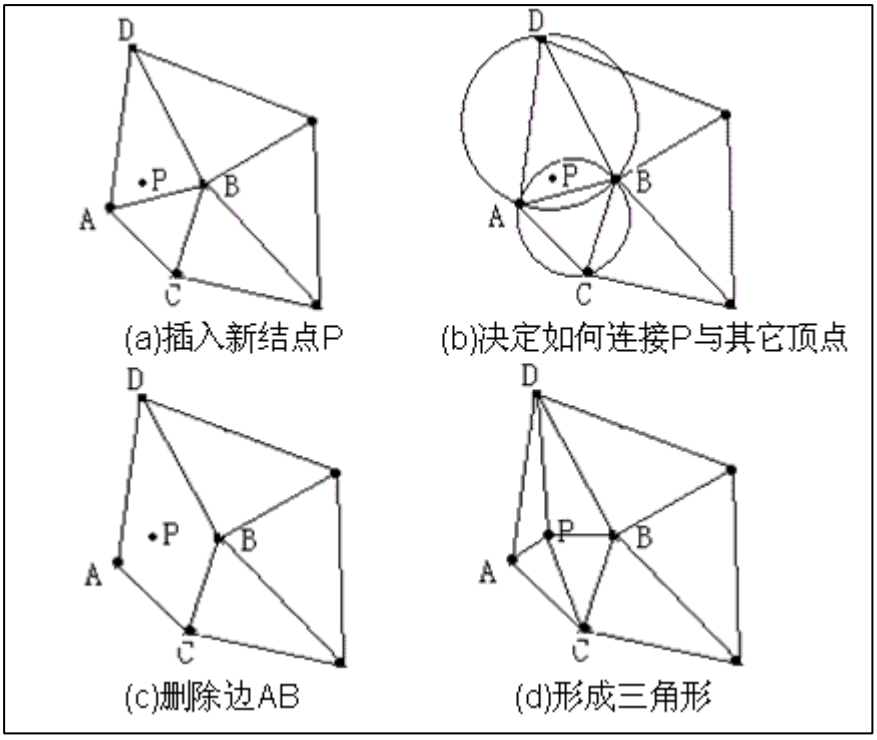

Figure 3.5 Illustration of 2D Bowyer/Watson algorithm for Delaunay Triangulation

- 局部变换法又称为换边、换面法。当利用局部变换法实现增量式点集的 Delaunay 三角剖分时，首先定位新加入点所在的三角形，然后在网格中加入三个新的连接该三角形顶点与新顶点的边，若该新点位于某条边上，则该边被删除，四条连接该新点的边被加入。最后，在通过换边方法对该新点的局部区域内的边进行检测和变换，重新维护网格的 Delaunay 性质。局部变换法的另一个优点是其可以对已存在的三角网格进行优化，使其变换成为 Delaunay 三角网格，该方法的缺点则是当算法扩展到高维空间时变得较为复杂。

四、Delaunay 三角剖分在 OpenCascade 的应用

OpenCascade 中网格剖分的包主要有 BRepMesh、MeshAlgo、MeshVS，其中，类 MeshAlgo_Delaunay 使用算法 Watson 来进行 Delaunay 三角剖分。从类 StlTransfer 中的注释 The triangulation is computed with the Delaunay algorithm implemented in package BRepMesh. 可以看出包 BRepMesh 就是 Delaunay 三角剖分的具体实现。使用方法如下：

```
BRepMesh::Mesh (aShape, Deflection);
```

这个函数主要是用来对拓扑形状进行三角剖分。以下通过将一个圆柱三角剖分为例说明如何将一个拓扑形状进行三角剖分并将结果进行可视化。

```
/**
*    Copyright (c) 2013 eryar All Rights Reserved.
*
*       File    : Main.cpp
*       Author  : eryar@163.com
*       Date    : 2013-05-26
*       Version : 0.1
*
*    Description : Use BRepMesh_Delaun class to learn
*                  Delaunay's triangulation algorithm.
*
*/
// Open Cascade library.
#include <gp_Pnt.hxx>
#include <gp_Pln.hxx>
#include <BRep_Tool.hxx>
#include <TopoDS.hxx>
#include <TopoDS_Edge.hxx>
#include <TopoDS_Wire.hxx>
#include <TopoDS_Face.hxx>
#include <BRepBuilderAPI_MakeEdge.hxx>
#include <BRepBuilderAPI_MakeWire.hxx>
#include <BRepBuilderAPI_MakeFace.hxx>

#include <BRepPrimAPI_MakeBox.hxx>
#include <BRepPrimAPI_MakeCone.hxx>
#include <BRepPrimAPI_MakeCylinder.hxx>
#include <BRepPrimApI_MakeSphere.hxx>

#include <BRepMesh.hxx>
#include <TopExp_Explorer.hxx>
#include <Poly_Triangulation.hxx>
#include <TShort_Array1OfShortReal.hxx>

#pragma comment(lib, "TKernel.lib")
#pragma comment(lib, "TKMath.lib")
```

```
#pragma comment(lib, "TKBRep.lib")
#pragma comment(lib, "TKPrim.lib")
#pragma comment(lib, "TKMesh.lib")
#pragma comment(lib, "TKTopAlgo.lib")

// OpenSceneGraph library.
#include <osgDB/ReadFile>
#include <osgViewer/Viewer>
#include <osgViewer/ViewerEventHandlers>
#include <osgGA/StateSetManipulator>

#pragma comment(lib, "osgd.lib")
#pragma comment(lib, "osgDbd.lib")
#pragma comment(lib, "osgGAd.lib")
#pragma comment(lib, "osgViewerd.lib")

osg::Node* BuildShapeMesh(const TopoDS_Shape& aShape)
{
    osg::ref_ptr<osg::Group> root = new osg::Group();
    osg::ref_ptr<osg::Geode> geode = new osg::Geode();
    osg::ref_ptr<osg::Geometry> triGeom = new osg::Geometry();
    osg::ref_ptr<osg::Vec3Array> vertices = new osg::Vec3Array();
    osg::ref_ptr<osg::Vec3Array> normals = new osg::Vec3Array();

    BRepMesh::Mesh(aShape, 1);

    TopExp_Explorer faceExplorer;

    for (faceExplorer.Init(aShape, TopAbs_FACE); faceExplorer.More();
faceExplorer.Next())
    {
        TopLoc_Location loc;
        TopoDS_Face aFace = TopoDS::Face(faceExplorer.Current());

        Handle_Poly_Triangulation triFace =
BRep_Tool::Triangulation(aFace, loc);
        Standard_Integer nTriangles = triFace->NbTriangles();

        gp_Pnt vertex1;
        gp_Pnt vertex2;
        gp_Pnt vertex3;

        Standard_Integer nVertexIndex1 = 0;
        Standard_Integer nVertexIndex2 = 0;
```

```
        Standard_Integer nVertexIndex3 = 0;

        TColgp_Array1OfPnt nodes(1, triFace->NbNodes());
        Poly_Array1OfTriangle triangles(1, triFace->NbTriangles());

        nodes = triFace->Nodes();
        triangles = triFace->Triangles();

        for (Standard_Integer i = 1; i <= nTriangles; i++)
        {
            Poly_Triangle aTriangle = triangles.Value(i);

            aTriangle.Get(nVertexIndex1, nVertexIndex2, nVertexIndex3);

            vertex1 = nodes.Value(nVertexIndex1);
            vertex2 = nodes.Value(nVertexIndex2);
            vertex3 = nodes.Value(nVertexIndex3);

            gp_XYZ vector12(vertex2.XYZ() - vertex1.XYZ());
            gp_XYZ vector13(vertex3.XYZ() - vertex1.XYZ());
            gp_XYZ normal = vector12.Crossed(vector13);
            Standard_Real rModulus = normal.Modulus();

            if (rModulus > gp::Resolution())
            {
                normal.Normalize();
            }
            else
            {
                normal.SetCoord(0., 0., 0.);
            }

            vertices->push_back(osg::Vec3(vertex1.X(), vertex1.Y(),
vertex1.Z()));
            vertices->push_back(osg::Vec3(vertex2.X(), vertex2.Y(),
vertex2.Z()));
            vertices->push_back(osg::Vec3(vertex3.X(), vertex3.Y(),
vertex3.Z()));

            normals->push_back(osg::Vec3(normal.X(), normal.Y(),
normal.Z()));
        }
    }
```

```
    triGeom->setVertexArray(vertices.get());
    triGeom->addPrimitiveSet(new
osg::DrawArrays(osg::PrimitiveSet::TRIANGLES, 0, vertices->size()));

    triGeom->setNormalArray(normals);
    triGeom->setNormalBinding(osg::Geometry::BIND_PER_PRIMITIVE);

    geode->addDrawable(triGeom);

    root->addChild(geode);

    return root.release();
}

int main(int argc, char* argv[])
{
    osgViewer::Viewer myViewer;
    osg::ref_ptr<osg::Group> root = new osg::Group();

    root->addChild(BuildShapeMesh(BRepPrimAPI_MakeCylinder(.6, 1)));

    myViewer.setSceneData(root);

    myViewer.addEventHandler(new
osgGA::StateSetManipulator(myViewer.getCamera()->getOrCreateStateSet()
));
    myViewer.addEventHandler(new osgViewer::StatsHandler);
    myViewer.addEventHandler(new osgViewer::WindowSizeHandler);

    return myViewer.run();
}
```

结果如下图所示：

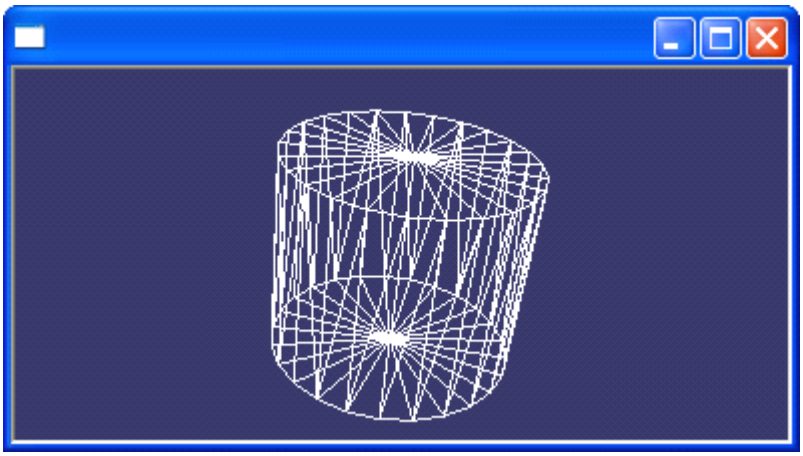

Figure 4.1 Cylinder mesh generated by BRepMesh::Mesh

BRepMesh::Mesh 是经过封装的，便于对拓扑形状进行三角剖分。以下通过一个简单的例子来说明直接使用 BRepMesh_Delaun 的方法：

```
/**
*    Copyright (c) 2013 eryar All Rights Reserved.
*
*       File    : Main.cpp
*       Author  : eryar@163.com
*       Date    : 2013-05-26
*       Version : 0.1
*
*    Description : Use BRepMesh_Delaun class to learn
*                  Delaunay's triangulation algorithm.
*
*/

#include <BRepMesh_Edge.hxx>
#include <BRepMesh_Delaun.hxx>
#include <BRepMesh_Array1OfVertexOfDelaun.hxx>
#include <TColStd_MapIteratorOfMapOfInteger.hxx>

#pragma comment(lib, "TKernel.lib")
#pragma comment(lib, "TKMesh.lib")

int main(int argc, char* argv[])
{
    BRepMesh_Array1OfVertexOfDelaun vertices(1, 4);

    vertices.SetValue(1, BRepMesh_Vertex(0, 0, MeshDS_Free));
    vertices.SetValue(2, BRepMesh_Vertex(1, 0, MeshDS_Free));
    vertices.SetValue(3, BRepMesh_Vertex(1, 1, MeshDS_Free));
    vertices.SetValue(4, BRepMesh_Vertex(0, 1, MeshDS_Free));

    BRepMesh_Delaun triangulation(vertices);
    //triangulation.AddVertex(BRepMesh_Vertex(0.5, 0.5,
MeshDS_OnSurface));
    Handle_BRepMesh_DataStructureOfDelaun meshData =
triangulation.Result();

    std::cout<<"Iterate Mesh Triangles:"<<std::endl;

    MeshDS_MapOfInteger::Iterator triDom;
    for (triDom.Initialize(meshData->ElemOfDomain()); triDom.More();
triDom.Next())
    {
```

```
        Standard_Integer triId = triDom.Key();
        const BRepMesh_Triangle& curTri = meshData->GetElement(triId);

        Standard_Integer vertexIndex1 = 0;
        Standard_Integer vertexIndex2 = 0;
        Standard_Integer vertexIndex3 = 0;

        Standard_Integer edgeIndex1 = 0;
        Standard_Integer edgeIndex2 = 0;
        Standard_Integer edgeIndex3 = 0;

        Standard_Boolean o1 = Standard_False;
        Standard_Boolean o2 = Standard_False;
        Standard_Boolean o3 = Standard_False;

        curTri.Edges(edgeIndex1, edgeIndex2, edgeIndex3, o1, o2, o3);

        const BRepMesh_Edge& edge1 = meshData->GetLink(edgeIndex1);
        const BRepMesh_Edge& edge2 = meshData->GetLink(edgeIndex2);
        const BRepMesh_Edge& edge3 = meshData->GetLink(edgeIndex3);

        vertexIndex1 = (o1? edge1.FirstNode(): edge1.LastNode());
        vertexIndex2 = (o1? edge1.LastNode() : edge1.FirstNode());
        vertexIndex3 = (o2? edge2.LastNode() : edge2.FirstNode());

        const BRepMesh_Vertex& vertex1 =
meshData->GetNode(vertexIndex1);
        const BRepMesh_Vertex& vertex2 =
meshData->GetNode(vertexIndex2);
        const BRepMesh_Vertex& vertex3 =
meshData->GetNode(vertexIndex3);

        const gp_XY& p1 = vertex1.Coord();
        const gp_XY& p2 = vertex2.Coord();
        const gp_XY& p3 = vertex3.Coord();

        std::cout<<"--------"<<std::endl;
        std::cout<<p1.X()<<" , "<<p1.Y()<<std::endl;
        std::cout<<p2.X()<<" , "<<p2.Y()<<std::endl;
        std::cout<<p3.X()<<" , "<<p3.Y()<<std::endl;
        std::cout<<"========"<<std::endl;
    }

    return 0;
```

```
}
```

上述程序是以一个正方形为例，使用 BRepMesh_Delaun 三角剖分的结果为两个三角形，如下所示：

```
Iterate Mesh Triangles:
--------
1 , 1
0 , 0
1 , 0
=========
--------
1 , 1
0 , 1
0 , 0
=========
```

以上结果都是二维空间上的，三维空间中的使用方法可以参考类：BRepMesh_FastDiscretFace。这个类说明了如何将一个面进行网格划分。

五、结论

Delaunay 三角剖分理论在三维几何造型中还是比较重要的，通过对形状的三角剖分，不仅可以对其进行可视化，还便于对形状做进一步的处理，如消隐、光照处理等。通过对 OpenCascade 中三角剖分算法的使用，以进一步了解三角剖分理论应用及其算法实现。

六、参考资料

1. 周培德. 计算几何—算法设计与分析. 清华大学出版社, 2011
2. 李海生. Delaunay 三角剖分理论及可视化应用研究. 哈尔滨工业大学出版社, 2010
3. 何援军. 计算机图形学. 机械工业出版社, 2010
4. 周元峰, 孙峰, 王文平, 汪嘉业, 张彩明. 基于局部修复的移动数据点 Delaunay 三角化快速更新方法. 计算机辅助设计与图形学学报, 2011, 12: 2006-1012
5. http://en.wikipedia.org/wiki/Voronoi_diagram