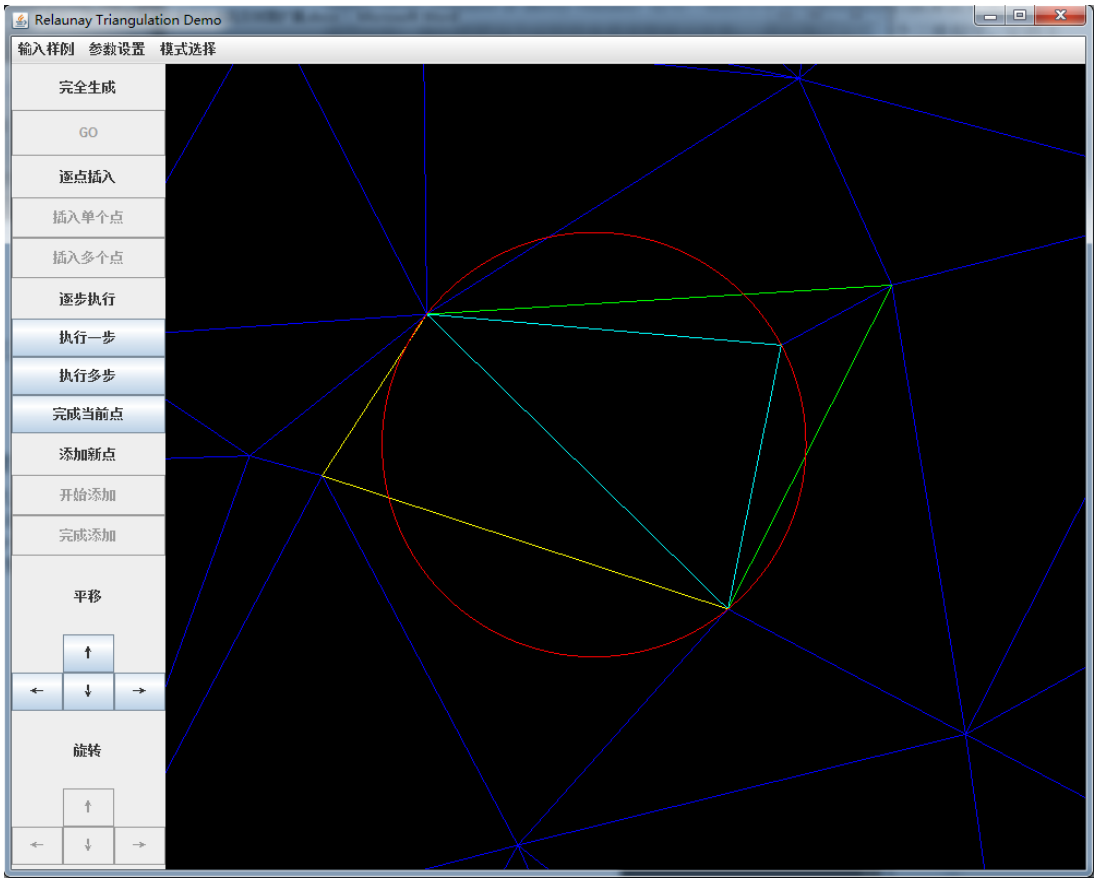
Delaunay 三角剖分随机增量算法的演示性实现及球面扩展

2010310573 袁文

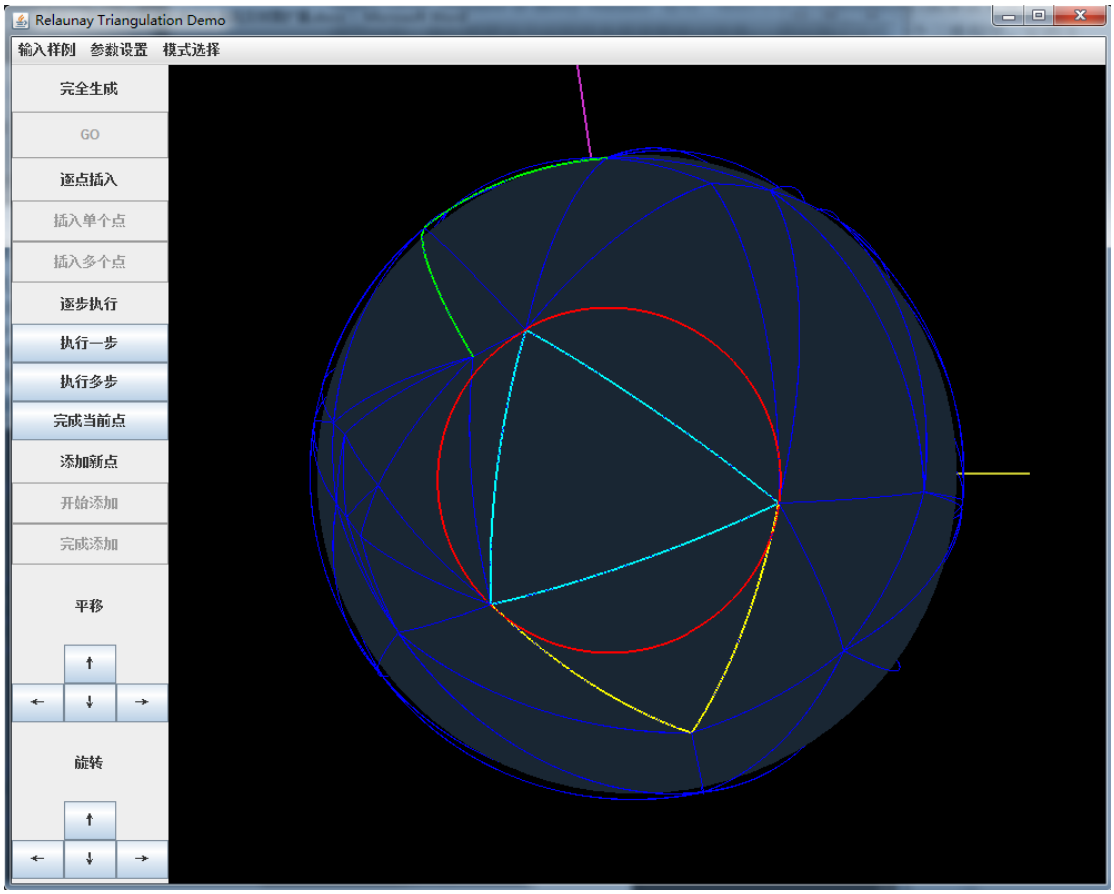
一、 问题背景

在计算几何课上，我们学习了 Delaunay 三角剖分算法，并且详细学习了一个期望时间复杂度为$O(n \log n)$的随机增量算法。本次大实验，我的主要目标就是实现该算法，并且分步实现算法中的每一步骤，使算法可以逐点、逐步执行，达到较好的演示效果。并进一步将该二维平面算法扩展到三维球面上，达到类似的演示效果。

- 程序的二维平面演示效果如下：



- 程序的三维球面演示效果如下：



二、 算法及原理

Delaunay 三角剖分算法参照课程讲义上的随机增量算法来实现。

二维平面算法中具体实现了“无穷远点”概念，在“toLeft”测试以及“inCircle”测试上加入了关于无穷远点的判断。

在初始化三角形的选择上，即无穷远点的选择上，使用了如下三个无穷远点：

$$\mathrm{InfiA}(0,+\infty),\mathrm{InfiB}(-\infty,-\infty),\mathrm{InfiC}(+\infty,-\infty)$$

以及无穷远三角形$\mathrm{InfiA}-\mathrm{InfiB}-\mathrm{InfiC}$。

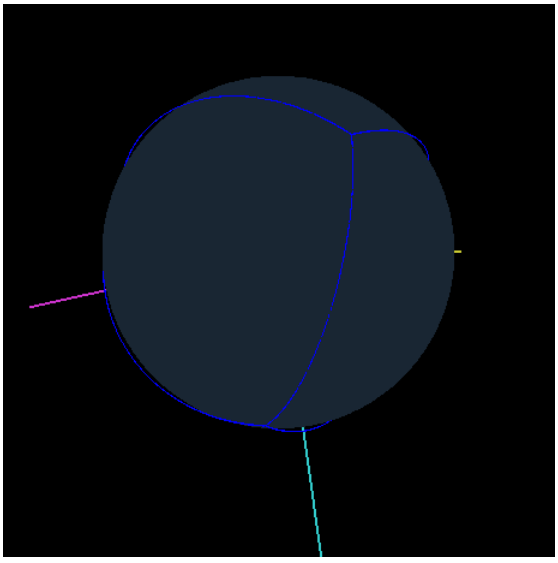
三维球面算法中，首先明确了球面顶点、球面边、球面三角形、“toLeft”测试以及“inCircle”测试的概念。

球面顶点的概念较简单，就是在三维直角坐标系中任意一个到原点距离为 1 的坐标点。

对于任意两个不重合的球面顶点 A、B，若它们加上坐标原点 O 三点不共线，则这三点确定了一个唯一的平面 AOB。这个平面与单位球 R（原点 O 为球心，1 为半径）相交成一个以原点 O 为圆心，1 为半径的圆 c。圆 c 上以 A、B 为端点的劣弧$\widehat{AB}$即为球面边 AB。当 A、B 与原点 O 三点共线时，球面边 AB 没有定义。同时，圆 C 将单位球 R 分为了两个部分R_{c1}和R_{c2}。规定向量$\vec{n} = \overrightarrow{OA} \times \overrightarrow{OB}$的方向为球面边 AB 的法向量方向。定义球面上球面边 AB 的左侧为R_{c1}和R_{c2}中法向量所在的那一侧。即，在定点 V 上，球面边 AB 的 toLeft 测试返回结果为是否满足$\overrightarrow{OV} \cdot \vec{n} \geq 0$。

球面上的任意三个顶点 A、B、C，若球面边 AB、BC、CA 都有定义，且 A、B、C 三点不在同一单位圆上，则由球面边 AB、BC、CA 首尾相连组成的封闭图形定义为球面三角形。

与二维平面上的情形相比较，三维球面的情形较为复杂：任意给定两个顶点 A、B，不一定存在合理的球面边 AB（A、B 可能与原点 O 共线）。同样的，任意给定三个顶点 A、B、C，也不一定存在球面三角形 ABC。但可以通过在初始化时，在球体上插入一个“正四面体”作为初始三角形集合，来避免这些情况的出现。初始化正四面体如下图所示：



在三维空间中，过任意不共线的三点有且仅有一个圆。当这三个点是球面三角形 ABC 的三个顶点时，这个圆 ABC 也在球面上。圆 ABC 将球面划分为两个部分。定义面积较小的那个区域为圆 ABC 的“内部”。同时，定义 inCircle 测试 $\mathrm{inCircle}(p, a, b, x)$为，判断 x 是否在圆 pab 的内部。

不难发现，当初始化三角形集合均为合理的三角形时，此后生成的三角形也都将是合理的。通过对算法中插入新点时添加新边以及“flip”操作的分析可以得出以上结论。同时，这四个球面三角形覆盖了整个球面，因此也是一个合理的初始化三角形集合。另外，这四个三角形均满足：球面三角形 ABC 中，C 在 AB 的左侧（也可推断出 A 在 BC 左侧，B 在 CA 左侧）。并且，当初始化三角形集合满足上述条件时，后续生成的新三角形也满足上述条件。这一结论也可以通过分析算法的详细步骤得出。

三、 系统设计

实验中程序采用 Java 和 OpenGL 编写。采用 eclipse + jdk 1.6.0 + jogl 1.1.1。

程序中代码分为两部分：

- 算法部分

  包括数据结构、算法实现等等，在 datastructure 包中。

- 界面显示部分

  负责前端显示、控制等，在 demo 包中，仅含有 Demo.java 一个文件。

两部分具体设计及功能的介绍如下：

A. 算法部分

算法部分包括了 DCEL 数据结构的实现。具体有二维平面中使用到的 Vertex 类、HalfEdge 类以及 Triangle 类；以及三维球面中使用到的 SphericalVertex 类、SphericalHalfEdge 类以及 SphericalTriangle 类。另外，在

处理二维平面的无穷远点时，还用到了 VertexInfi、HalfEdgeInfiS、HalfEdgeInfiT、HalfEdgeInfiST 等辅助类。

在具体使用 Delaunay 三角剖分的随机增量算法时，还用到了 Bucket 类以及 SphericalBucket 类，用来实现算法中的“Bucket”概念。

算法的对外接口在 Graph 类（二维平面）和 SphericalGraph 类（三维球面）中实现。

Graph 类和 SphericalGraph 类中，主要有如下对外接口：

- addVertex(double x, double y)

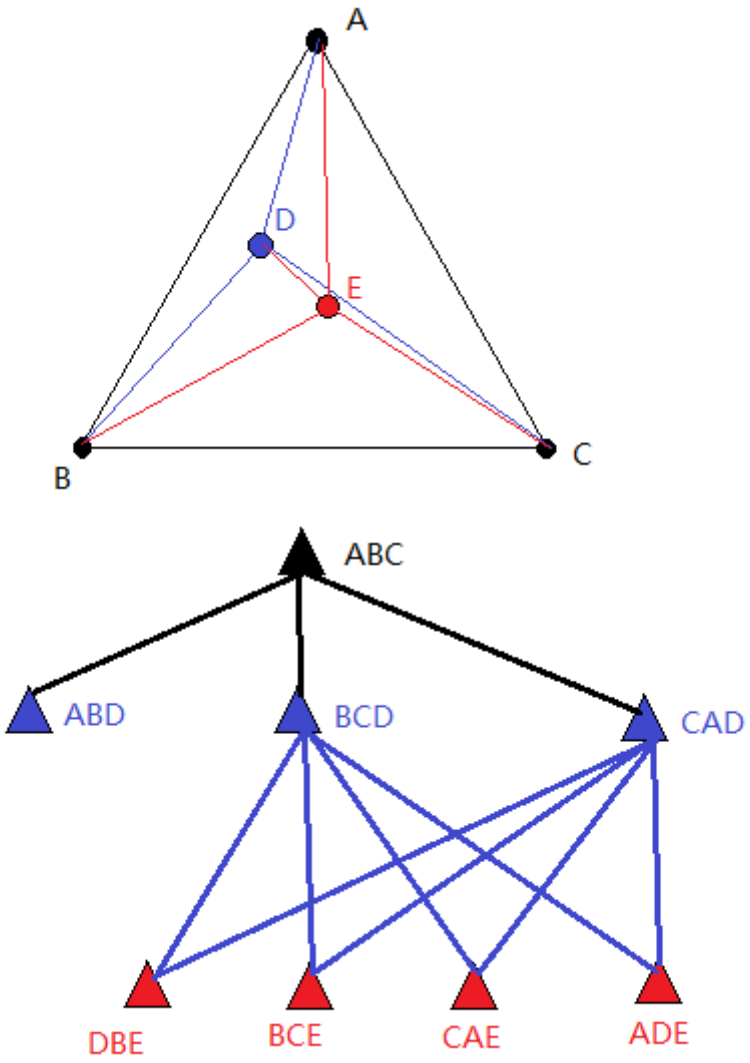
  此接口的作用是动态的向当前平面图（或球面图）中添加一个新顶点。

  由于采用的算法是增量式的，所以对于动态增加顶点天然的具有很好的支持性。实际中遇到的问题是，添加一个新顶点之后，需要确定该顶点所处的区域。即，找到当前三角剖分状态下，包含该顶点的三角形。

  在算法中，所有的顶点都在添加初始化三角形时被分配到相应的三角形区域中；此后每处理一个顶点，都会为涉及到的顶点重新分配三角形区域。算法通过这种 rebucketing 机制来确保预先添加的顶点可以快速找到正确的三角形区域。

  而对于动态添加的顶点，处理方法也较为简单：在算法运行过程中，保存所有生成过的三角形；并让每个被移除的三角形，都保存指向有可能替代它的新三角形的“指针”结构（Java 中没有指针的概念，实际上是引用）。具体来说，算法每处理一个顶点，都会移除一些已有的“旧三角形”，并添加一些“新三角形”。在移除这些“旧三角形”时，让它们中的每一个都指向“新三角形”的集合。实际上，这些“指针”和算法中生成的三角形构成了一个多叉链路结构，同时也在一定程度上反映了算法的执行过程。

以上链路结构的一个可能模型如下：



有了这种数据结构之后，我们可以在它上面进行类似于多叉树上的搜索操作，从初始化三角形集合（无穷远三角形或是正四面体的四个三角面）开始，逐层搜索到当前的三角剖分状态，找到任意顶点所处的三角形区域。

而且，这种搜索的复杂度也不会很高，期望是$O(\log n)$。这是因为，假设这一新添加的顶点一开始就在平面图（球面图）中，则对它进行 rebucketing 操作的次数期望为$O(\log n)$（见课程讲义），而这一数目和在以上链路结构中搜索到这一顶点所处三角形区域所需的操作次数一样。

在程序的实际运行过程中，这种搜索区域的方法效果也很好。添加新顶点均可以立刻完成，感觉不到延时。

- go()

此接口的作用是完成当前二维平面图或三维球面图上的 Delaunay 三角剖分。

该接口就是讲义上随机增量算法的完整实现。在三维球面图中，除了接口内部调用的 toLeft 测试以及 inCircle 测试有所变化之外，并没有太大的变化。

如果要对算法的效率进行测试，则应该调用此接口，并记录运行时间等参数。但由于本次试验的主要目标是用作逐步演示，所以没有运行时间测试等相关内容。

在程序自带的测试样例中，点数最多的样例“方格整点（二维平面图）”中有 25600 个顶点。直观感觉上，此接口可在一两秒之内完成“方格整点”样例上的 Delaunay 三角剖分。

- step()

此接口的作用是增量地在当前二维平面图或三维球面图中处理下一个随机选取的顶点。

该接口实现了逐点执行的功能。

在演示程序中，通过调用该接口，可以清楚地观察到 Delaunay 三角剖分的增量生成过程。

- moveon()

此接口的作用是执行算法的下一步骤。

该接口实现了逐步执行的功能。

这一接口的工作方式类似于一个状态机。在 Graph 类对象和 SphericalGraph 类对象中，保存了一个指示当前算法执行状况的状态

state。在调用这一接口时，它会根据算法执行状态 state 和其他一些运行过程中保存下来的变量执行算法的下一步骤。

state 的可能状态有：Ready，NewVertexSelected，TriangleFound，VertexInserted，SwapReady，InCircleTested，EdgeFilpped，SwapDone，RebucketReady，VertexRebucketed，RebucketDone，AllDone。

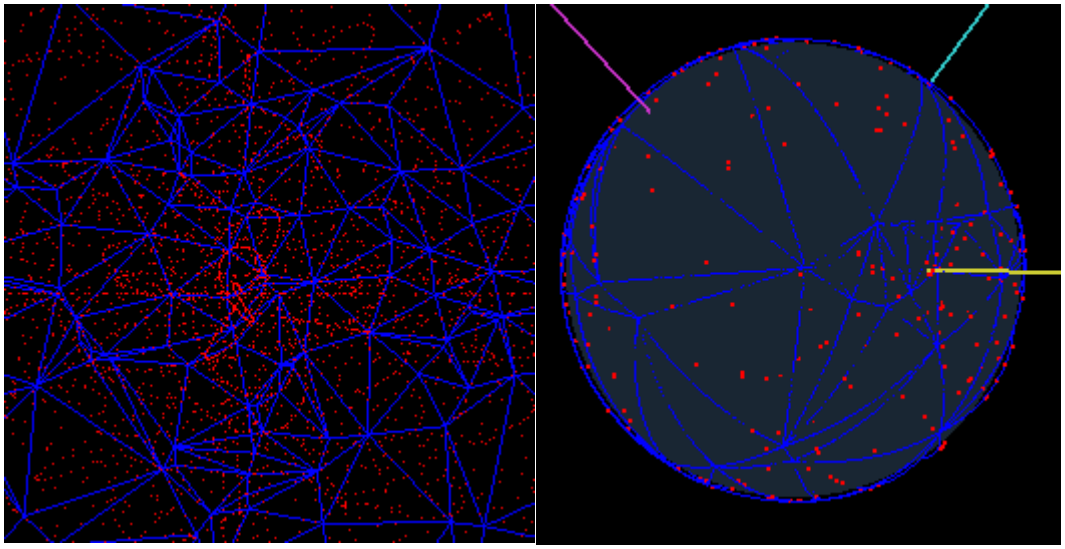
它们的意义分别为：

✧ Ready

表示算法正在准备处理下一顶点。这意味着：1、当前图中的三角剖分（在已处理的顶点上）是合理的 Delaunay 三角剖分；2、当前图中仍有顶点尚未处理。

这一状态可以认为是“初始状态”。算法运行过程中，每处理完一个顶点，都会回到这一状态。

这一状态下，二维平面图与三维球面图状态分别如下：

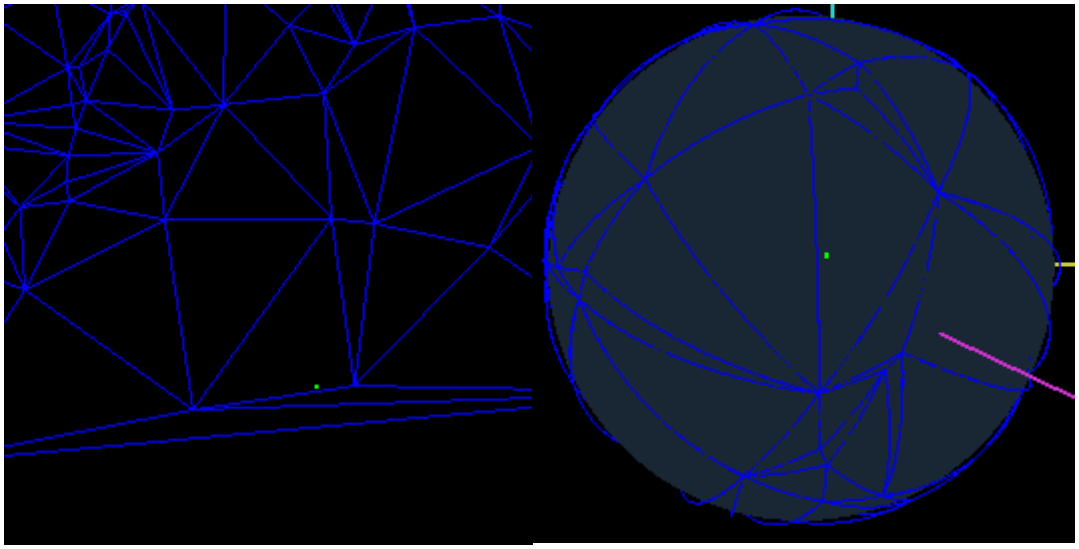


其中，红色点为尚未处理的顶点，蓝色边为 Delaunay 三角剖分的边。（右图中的黄、青、紫三条轴线表示坐标轴。）

✧ NewVertexSelected

表示算法已随机地选定下一顶点，准备处理。

这一状态下，二维平面图与三维球面图状态分别如下：

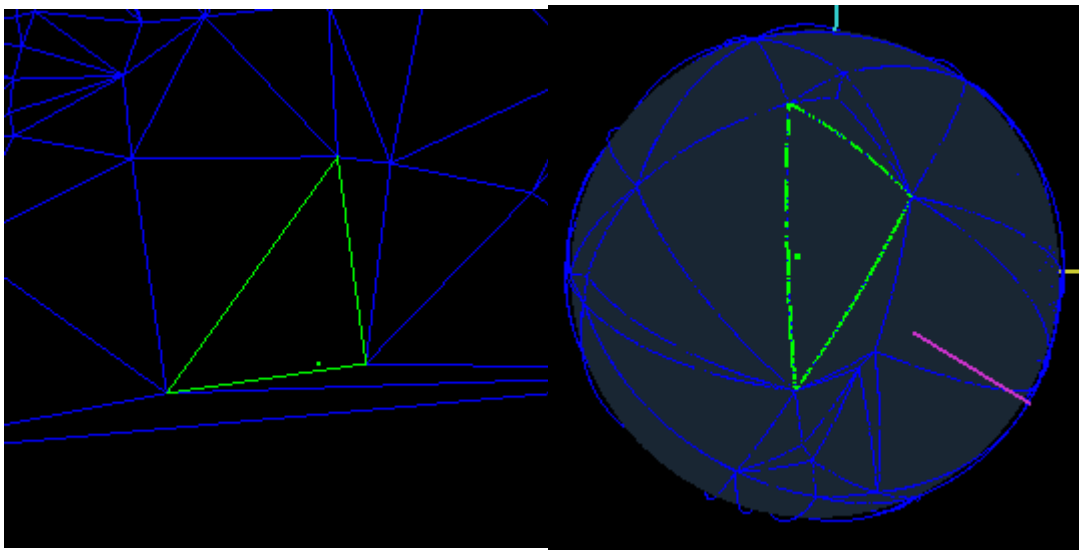


其中，绿色点为新选定的顶点。所有尚未处理的顶点（前一例中的红色点）此时不显示。

- TriangleFound

表示算法已找到该顶点所处的三角形区域。

这一状态下，二维平面图与三维球面图状态分别如下：

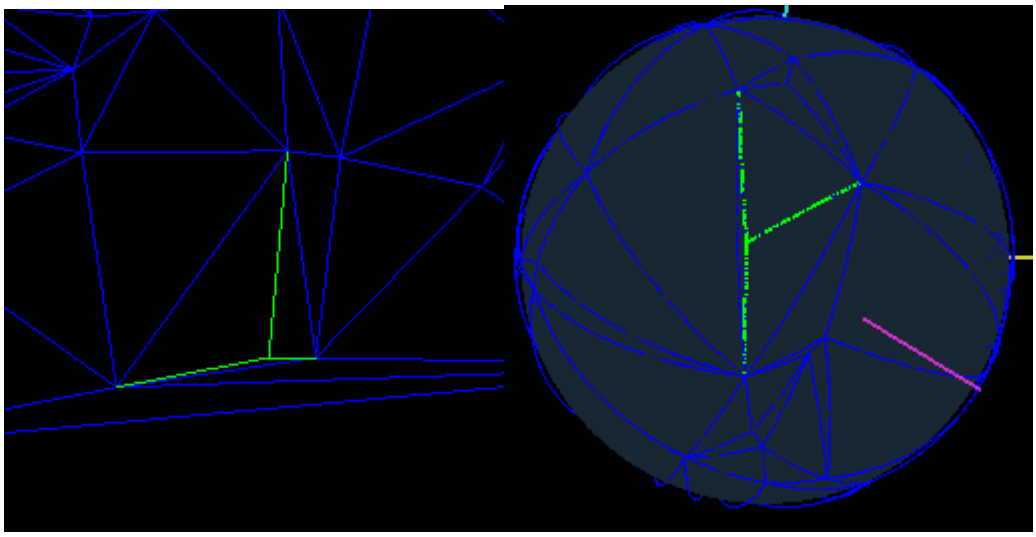


其中，绿色三角形表示该顶点所处的三角形。

- VertexInserted

表示算法已将该顶点插入已有的三角剖分中，并已完成新生成边（三角形）的添加。

这一状态下，二维平面图与三维球面图状态分别如下：

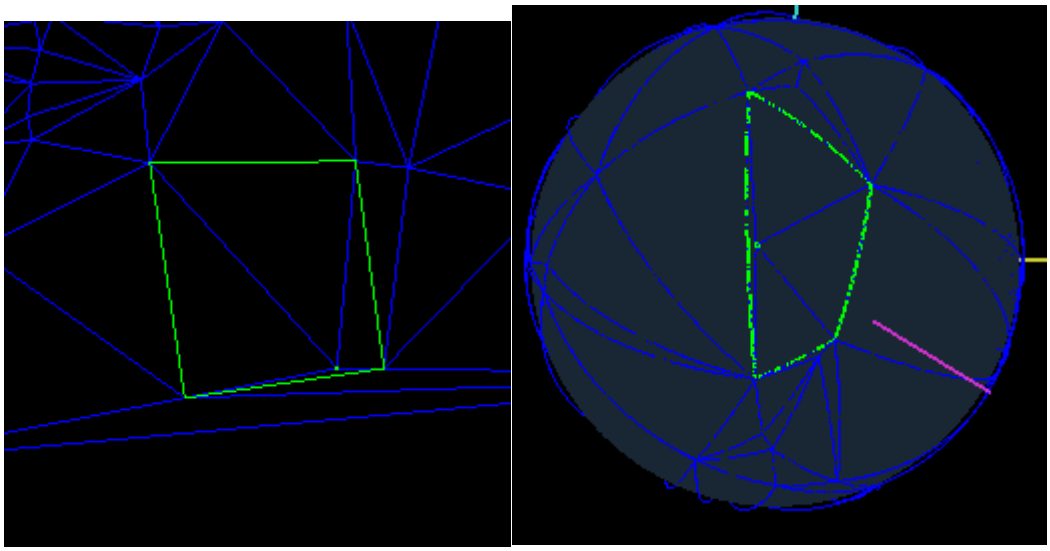


其中，绿色边表示新添加的边。

✧ SwapReady

表示算法准备进行下一步 swapTest。在处理一个顶点 p 的过程中，由于需要对多条边 ab（以及相对应的顶点 x）进行 swapTest，算法也会多次到达此状态。

这一状态下，二维平面图与三维球面图状态分别如下：

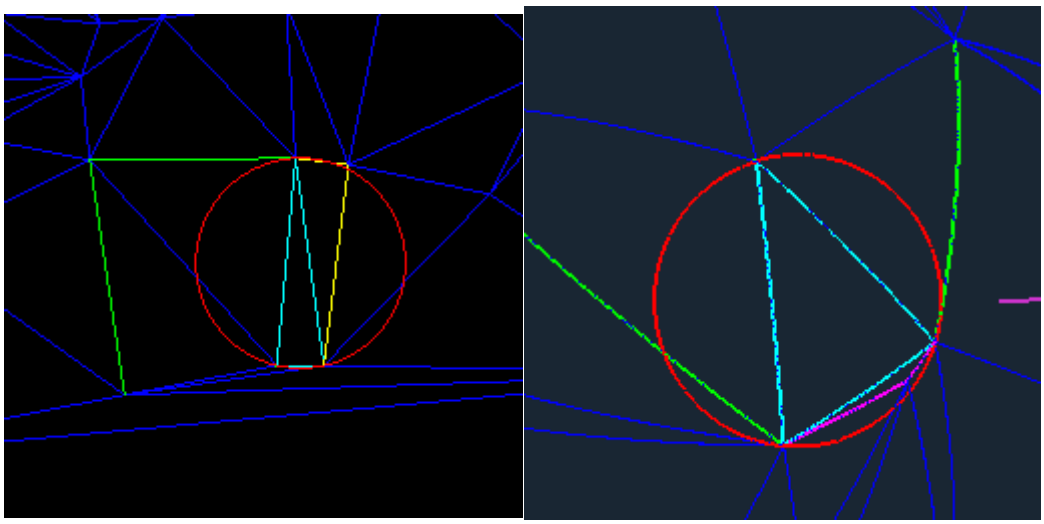


其中，绿色顶点是正在处理的顶点，而绿色的边是当前待进行 swapTest 的边。

✧ InCircleTested

表示算法完成了对某条边进行的 inCircle 测试。

这一状态下，二维平面图与三维球面图状态分别如下：

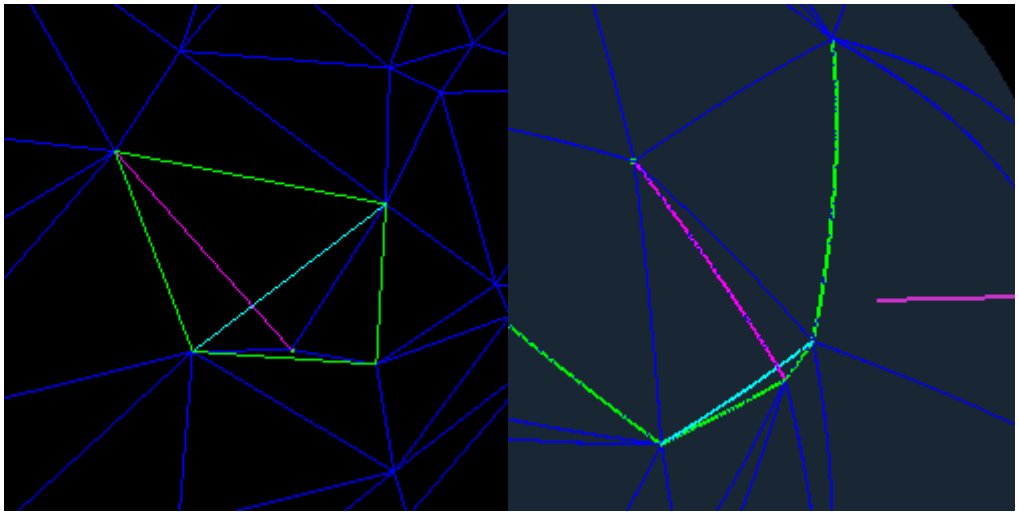


其中，青色边表示 inCircle 测试中的 pa、pb 以及 ab 三条边，红色圆是三角形 pab 的外接圆（算法运行中不需要计算此圆，画出此圆只是为了演示更为直观）。黄色和紫色边均表示 xa、xb 两条边。黄色说明 x 不在 pab 的外接圆中，而紫色说明 x 在 pab 的外接圆中。

✧ EdgeFilpped

表示算法完成了 filp 操作。只有 inCircle 测试返回结果为“真”时，才会进行 flip 操作。

这一状态下，二维平面图与三维球面图状态分别如下：

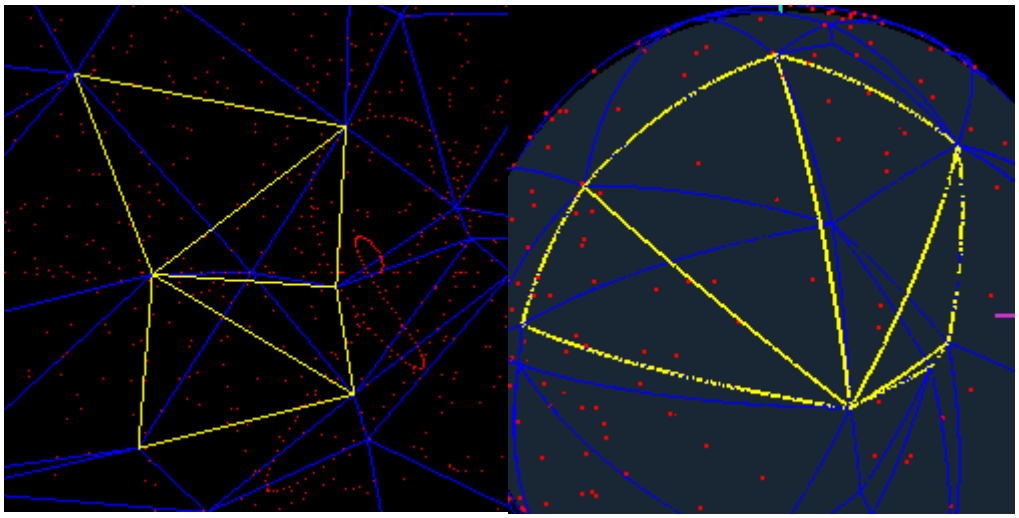


其中，紫色边表示新加边 px，青色边表示待移除边 ab。

✧ SwapDone

表示所有的 swapTest 均已完成。

这一状态下，二维平面图与三维球面图状态分别如下：

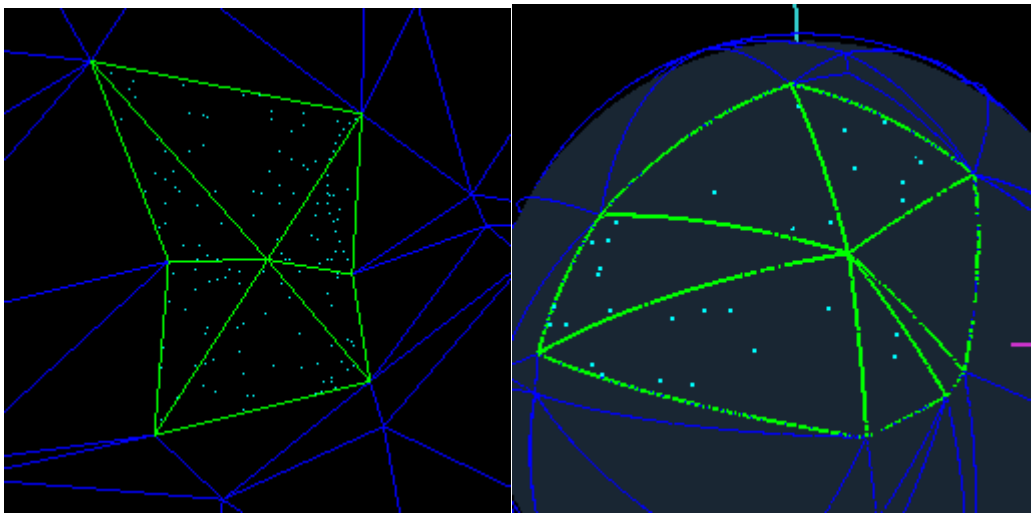


其中，黄色边表示处理当前顶点的过程中需要移除的边（三角形）。

✧ RebucketReady

表示算法准备进行 rebucketing 操作。

这一状态下，二维平面图与三维球面图状态分别如下：



其中，绿色边表示处理当前顶点的过程中将要添加的边（三角形）。青色点表示所有需要 rebucket 的顶点。

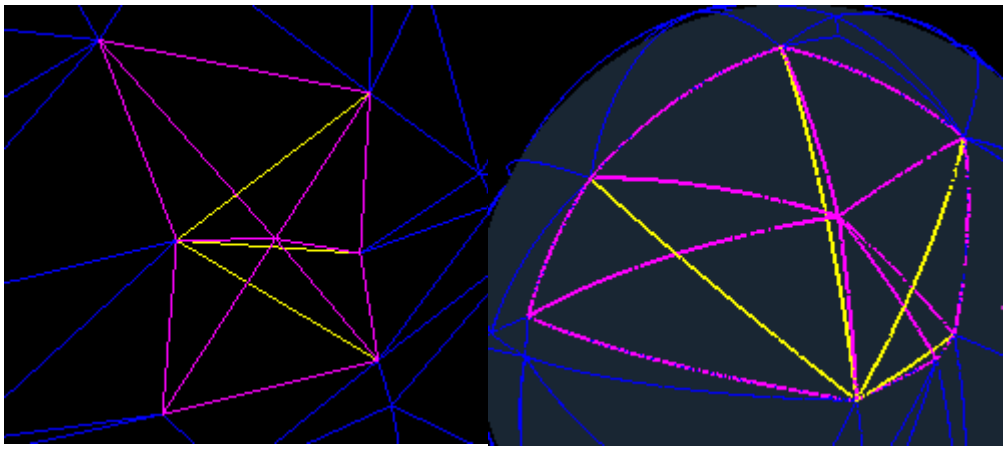
✧ VertexRebucketed

在原先的想法中，我希望把每个顶点的 rebucket 操作都展示出来。即把每一个顶点原先所处三角形以及之后所处三角形的对比展示出来。但由于样例中顶点数一般都较多，逐个顶点展示显得很繁琐，会消耗很多时间，所以后来不再采用这一方案。在现在的程序中，这一状态将不再会用到。

✧ RebucketDone

表示算法已经完成了所有涉及到的顶点的 rebucket 操作。

这一状态下，二维平面图与三维球面图状态分别如下：

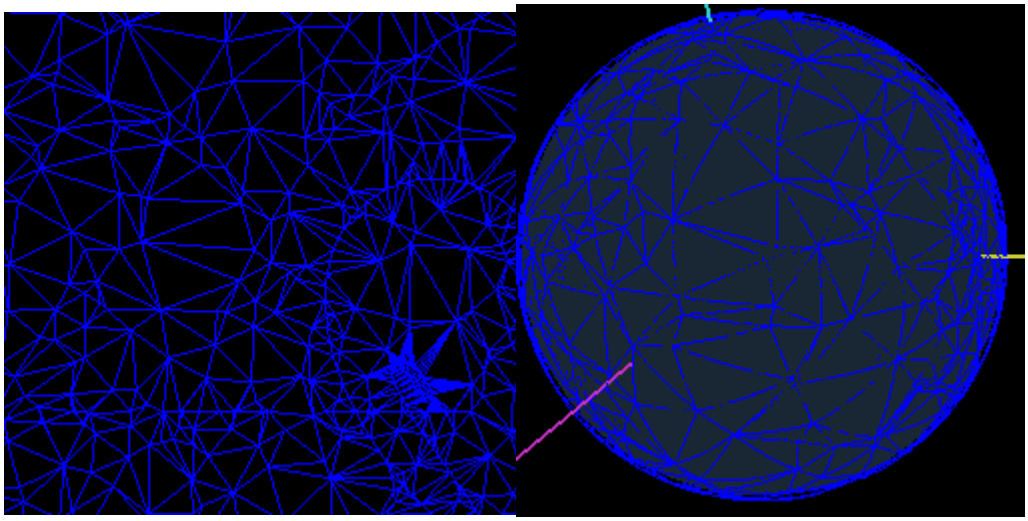


其中，紫色边表示所有将添加的新边（三角形），黄色边表示所有将移除的旧边（三角形，与紫色共用最外围的一圈边）。

✧ AllDone

表示算法已经处理完所有顶点。

这一状态下，二维平面图与三维球面图状态分别如下：



当继续向图中添加新顶点时，算法状态会从 AllDone 状态转移到 Ready 状态。

算法部分的主要内容就是以上这些。

B. 界面显示部分

在界面及显示上，可分为算法控制和显示控制两大部分，下面将分别说明。

a) 算法控制

算法控制主要包括：模式选择、参数设置、输入样例、完全生成、逐点插入、逐步执行以及添加新点这几部分。其中，前三项在菜单栏内，

后几项在程序界面的左侧控制面板内。

模式选择用于切换二维平面模式和三维球面模式。

参数设置用于设置逐点插入以及逐步执行时的连续执行步数和每两步之间的延时（以毫秒 ms 为单位）。

输入样例中，对于二维平面模式和三维球面模式，程序都自带三种不同的样例：随机取点、方格整点和三角函数样例。另外，也可以通过读取文本文件来使用新样例。程序也附带了两个文本样例 PlaneExample.txt 和 SphericalExample.txt，分别用于二维平面模式和三维球面模式。两个样例混用也不会出现 bug，但是演示效果不好。而且 PlaneExample.txt 样例中点数要多一点，在球面上展示时，由于绘制球面边（圆弧）的函数没有太多优化，显示时会有较明显的延时。

完全生成功能是完成当前图上的 Delaunay 三角剖分，处理完所有顶点。实际上是在调用 go()接口函数。

逐点插入是让算法逐点处理，增量地生成 Delaunay 三角剖分。实际上是在调用 step()接口函数。

逐步执行是让算法一步一步处理，展示出所有中间过程。实际上是在调用 moveon()接口函数。逐步执行也是演示效果相对来说最好的执行方式。在逐步执行过程中，由于算法运行状态会改变，所以完全生成、逐点插入以及添加新点这几项功能暂时会禁用。当算法完成当前顶点的处理之后，这些功能才可以再次启用。可以通过“完成当前点”按钮迅速完成当前顶点的处理。

添加新点是向当前图中动态的插入新顶点。点击“开始添加”按钮之后，在显示区域内连续点击鼠标可以连续添加多个顶点。点击“完成添加”按钮可以完成添加过程，继续进行算法执行上的控制。在二维平面模式中，程序会在鼠标点击的位置添加新顶点；而在三维球面模式中，由于有坐标旋转等功能，判断鼠标点击点与球面坐标之间的关系更为复杂，未实现此功能，代替它的是在球面上随机地添加一点。

b) 显示控制

二维平面模式与三维球面模式均支持显示上的放缩、平移，三维球面模式还支持坐标旋转。

放缩功能没有相应的按钮实现。可在显示区域通过滚动鼠标滚轮来实现放缩。

平移功能和旋转功能在程序的控制面板上都有对应的四个类似于方向键的按钮。另外，在显示区域拖动鼠标也可以实现平移功能，按住 Alt 键并拖动鼠标可以实现旋转功能。使用鼠标操作时，平移和旋转的连贯性都要优于使用按钮进行操作。

在进行旋转操作时，水平方向上的按钮和鼠标拖动都会让图形沿水平面旋转，而垂直方向上的按钮和鼠标拖动会让图形绕显示图中的黄色轴转动。这也是程序中仍为解决的问题之一：垂直方向上的旋转操作不够直观。

四、 待解决的问题

目前程序中仍存在的问题有：

- 采用 jogl 的 OpenGL 显示面板之后，面板有时会出现重绘不及时的情况。考虑过的解决方法是新建一个线程，定时重绘面板。但考虑到三维球面显示的效率不是很高，这种解决方法的效率也值得考虑。目前，当出现面板重绘不及时的情况时，可通过在面板区域点击鼠标来让它重绘，仍然没有很好的解决这一问题。
- 三维球面模式中，显示效率较低。OpenGL 中没有专门的画弧函数，所有球面边均是通过很多小段的直线段联接而成。因此，在显示效率与显示质量上需要做出权衡。目前，当顶点数较多（两、三千个以上）时，可以比较明显地感觉到显示时的延时。
- 三维球面模式中，采用添加“初始化正四面体”的方式来解决一系

列与球面三角形有关的特例问题。如果能更进一步地分析，处理好这些特例，甚至在球面边中加入“优弧球面边”的情况，三维球面上的三角剖分将会更加有趣。

- 精度问题。在二维平面以及三维球面的 toLeft 测试、inCircle 测试中，都只用到了加减、乘法以及比较这些精度和效率都相对较高的运算，但算法仍对顶点集合的数据有所要求。在某些情况下，由于精度问题，还是会出现在细节上的 Delaunay 三角剖分不合理，或是程序陷入死循环这类问题。在程序中，通过加入对“重复顶点”的判断，忽略掉了距离过近的顶点，以及放宽 toLeft 测试的条件、加强 inCircle 测试的条件这些手段，使得程序的鲁棒性有一定的提升，但仍未彻底解决这类问题。