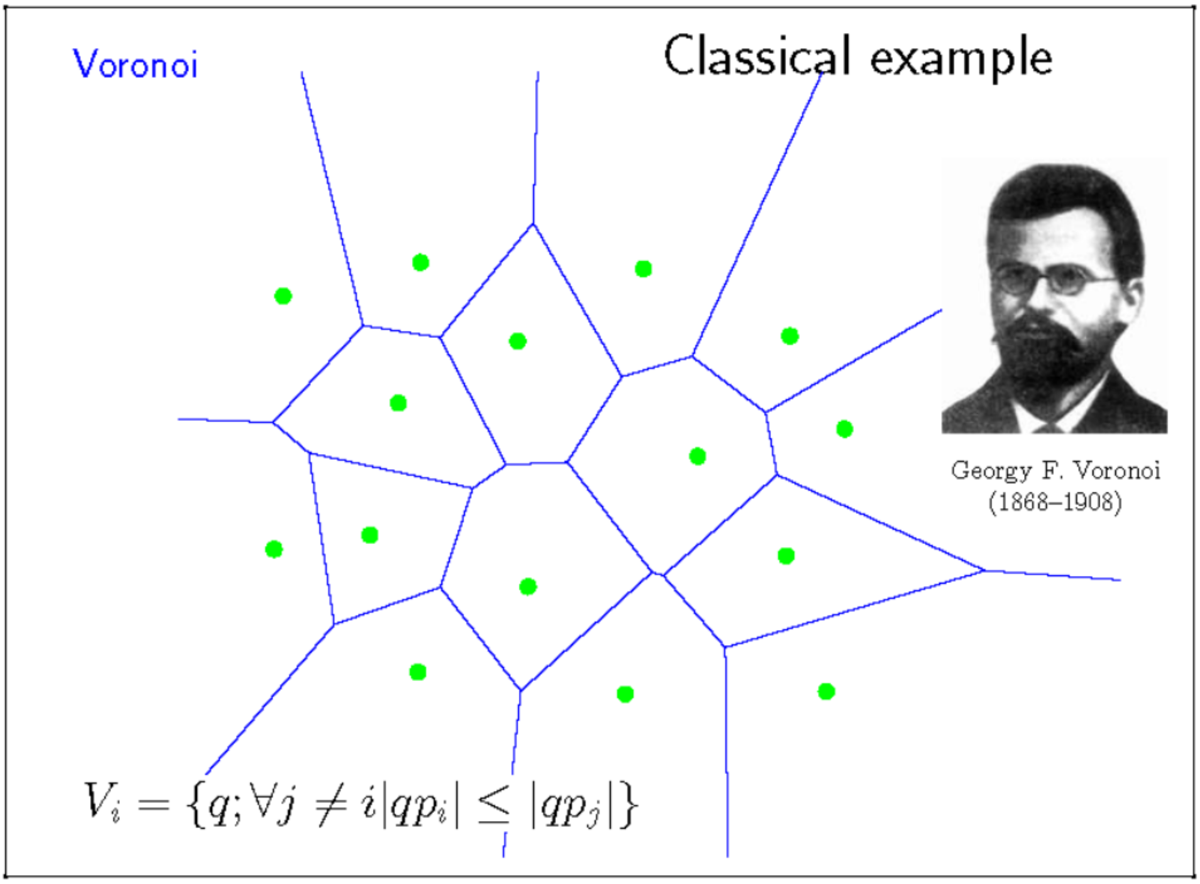
homework 8

Voronoi图与Delaunay三角化

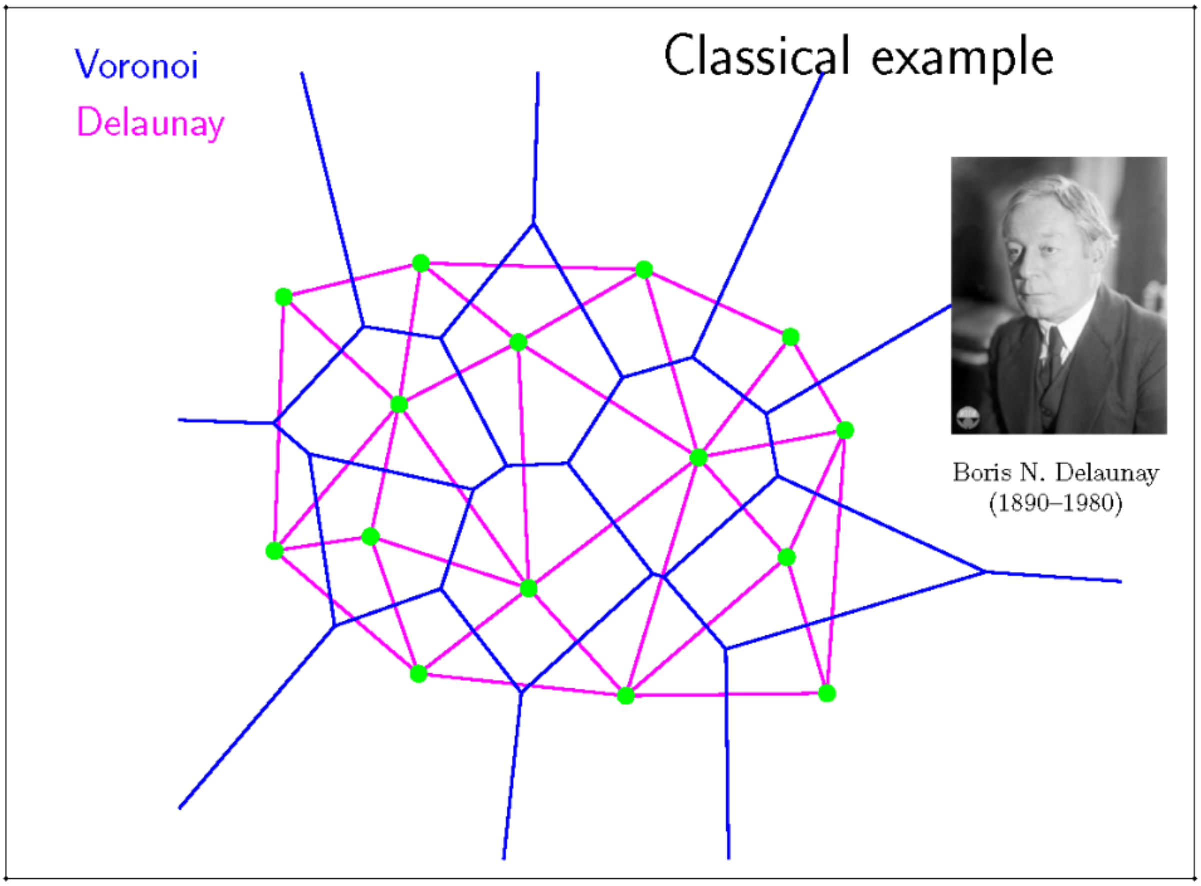
Voronoi Diagram

Voronoi图指的是给定一组点集，为每个点分配一个区域，属于这片区域的任何点到该点的距离比其他任何点集中的点都小。

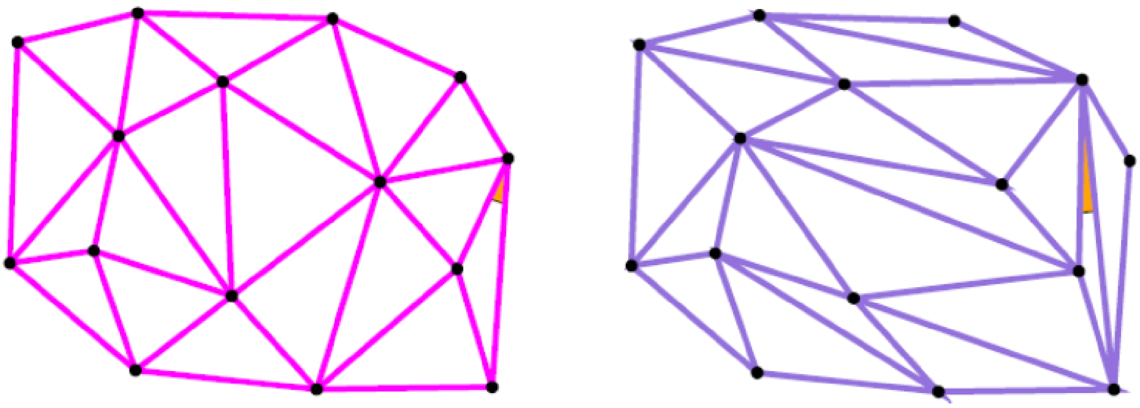


Delaunay triangulation

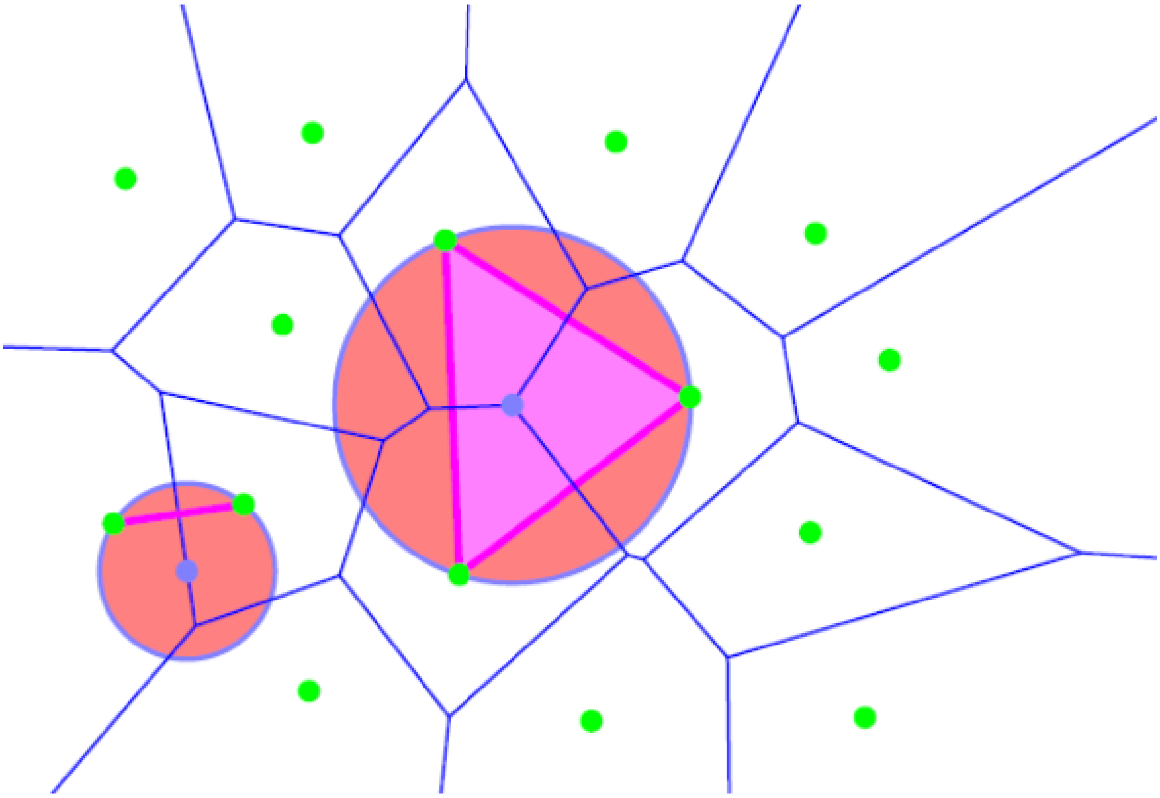
Delaunay三角化，会利用Voronoi图来对一个点集进行三角化。



Delaunay能保证在给定点集情况下，生成最优的三角网格（最大化最小角度），下图中左侧为Delaunay三角化，而右侧为一般算法：



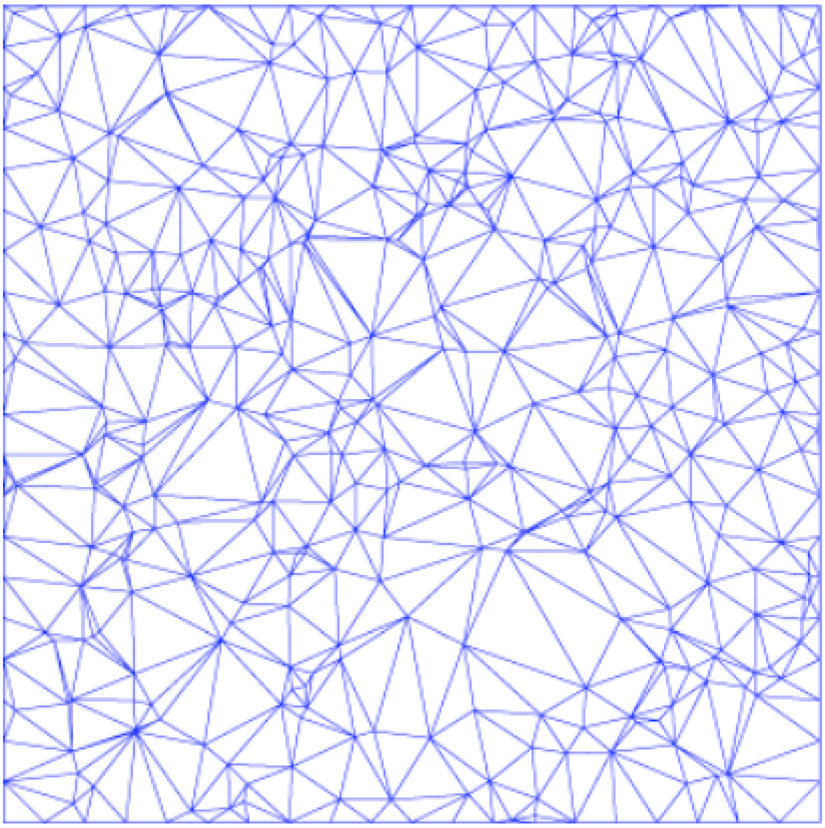
且任何一个三角形的外接圆内部，不会包含其他顶点：



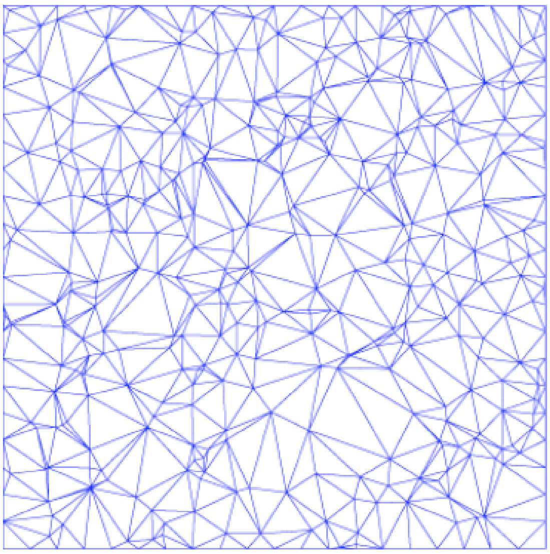
生成的三角形的最外层是该点集合的一个convex hull。

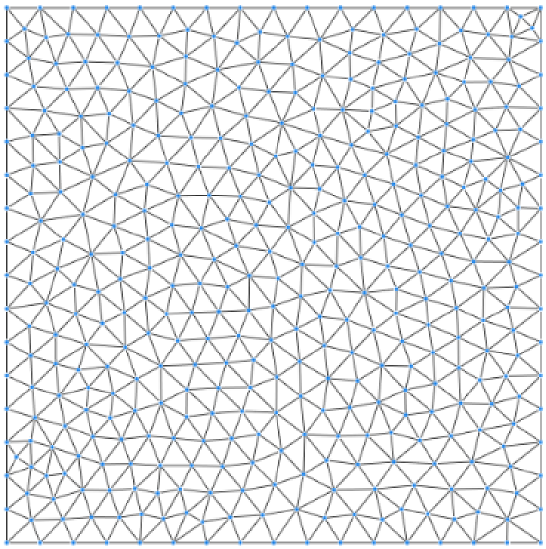
Lloyd Algorithm

但是DT算法只会改变点的连接关系，而不会移动点的位置。因此用DT算法生成的网格不一定是最好的，如下图：



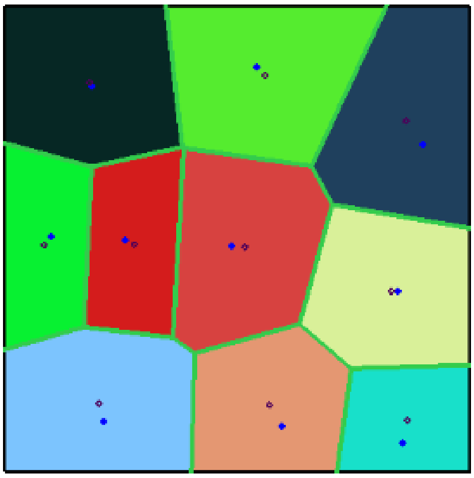
更多情况下，点的分布比连接关系更重要。如何让下图中左侧的网格变成右侧的样子：



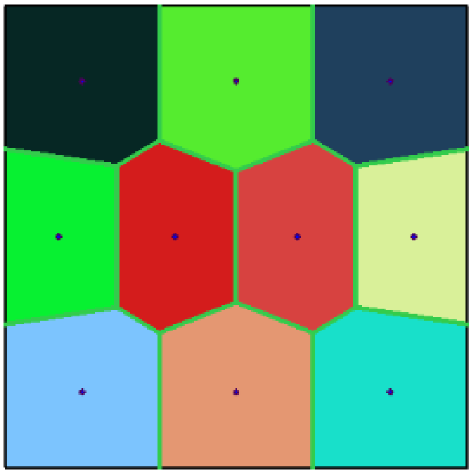


Centroidal Voronoi Tessellation

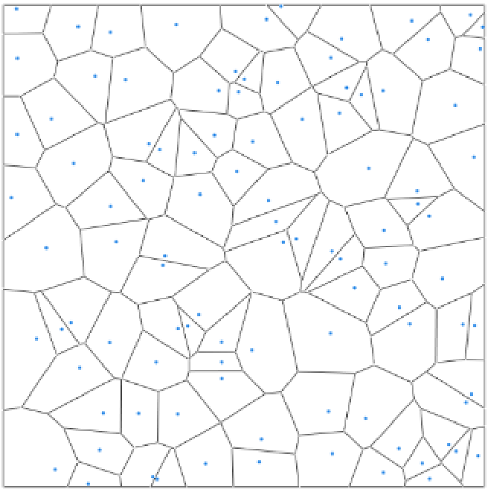
如果一个Voronoi图的每个面的中心就在这个面的点上，那么这个图被称为CVT。



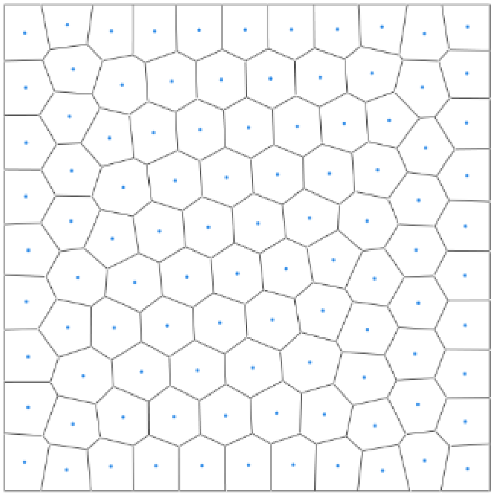
Generic VT



CVT

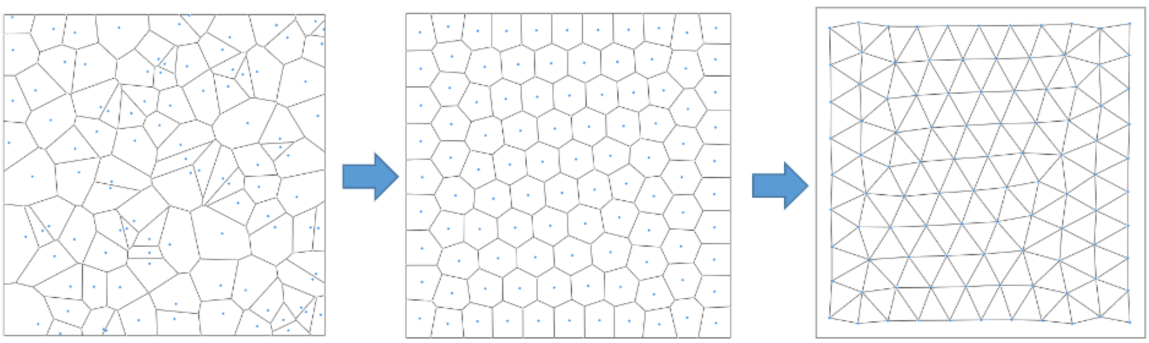


Generic VT



CVT

很显然的是，CVT生成的三角形质量更高：



而如何从VT到CVT？这个算法很直观，就是不断调整这些点的位置，往重心靠拢，不断迭代。这个算法就是Llyod算法。

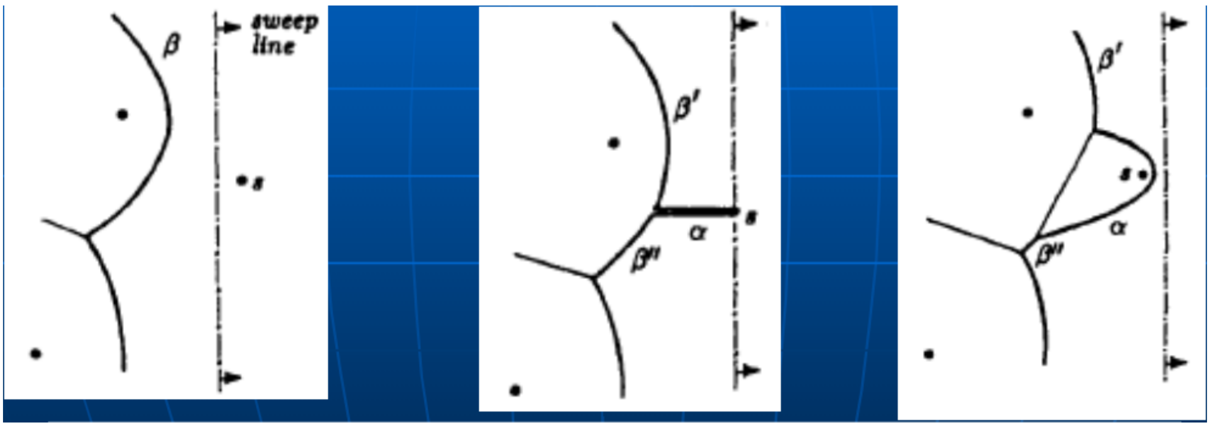
- 在给定的正方形区域内随机生成若干采样点
- 生成这些 点的Voronoi剖分
- 计算每个剖分的重心, 将采样点的位置更新到该重心
- 迭代步骤2和3

这个算法又被称为Voronoi图的relaxation。

扫描线算法

Voronoi图生成有一个我看来非常有意思的算法，那就是扫描线算法，同时扫描线算法也具有最差情况下$O(n\lg n)$的复杂度，平均复杂度为$O(n)$。我想要自己实现一下这个算法，具体的细节如下：

扫描线算法生成Voronoi图的平均时间复杂度为$\mathrm{O(n)}$，最坏情况下为$\mathrm{O(n\log n)}$。在二维平面上有n个点，成为点集P，想象一个直线在对这个平面进行扫描。这个线被称为扫描线。此外我们还需要维护一个beach line，可以成为波浪线。位于波浪线之后的区域的任意一点，与已经扫描过的点中某点的最小距离小于到扫描线的距离，而位于beach line上的点，到扫描线的距离与最小距离相等。



如上图中，最开始sweep line扫到某个点p时，这个点对应的beach line是一条直线，这个也容易理解，因为这时候这条线上的点到p的距离就等于到扫描线的距离，而其他位置的点到扫描线的距离都会大于到点p的点，对于已经扫描的其他点集就更不用说了，因为它们不在波浪线与扫描线之间。随着扫描线向前运动，这个线段会逐渐变成一个抛物线，从窄变宽。显然，对于每一个点都有这样一个抛物线，当抛物线有交点时，该点就在新的voronoi图的边界上。

接下来的难点就是如何实现这个算法了，目前为止我们应该意识到很明显的是我们不可能随时维护着这个SweepLine与BeachLine，除非是为了做动态演示；以及如何求得抛物线交点等等。下面就是实现细节。

在beachline中，每两个弧相交的点我们成为break point，这个点一定属于voronoi图的某个边。

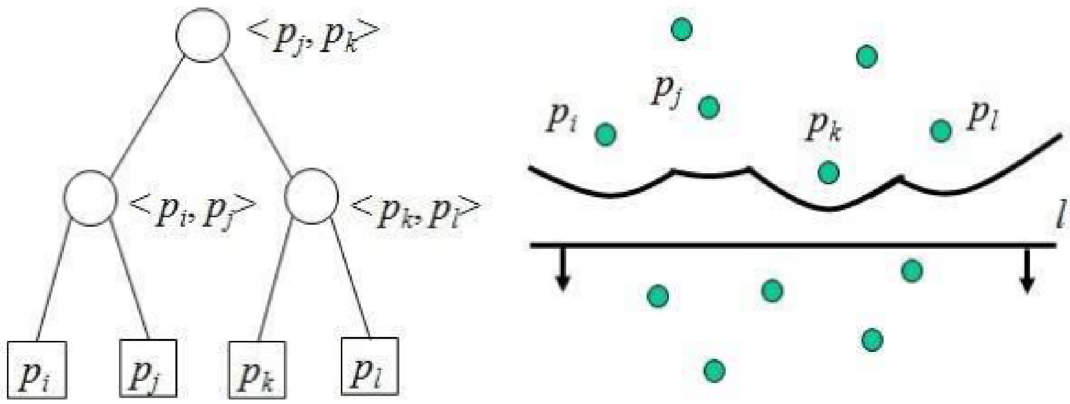
在想象中扫描线是连续前进的，但是实际上实现中，是离散前进的，这就是算法运行时候需要维护的两个队列，用来处理不同种类的事件。第一类事件，site事件，指的是扫描线触碰到了新的点，第二个是circle事件，指的是当有至少三个点位于同一个圆上，当这个圆与扫描线相切时，会触发circle事件，这时候的circle的圆心就是一个新的voronoi图的顶点。Voronoi图中每个顶点都是垂直平分线的交点，对应三个点也就是外接圆的圆心，这也就是为什么我们要处理circle事件。如果观察上面的动图，我们就会发现，一个circle事件发生是因为一个弧消失了，而这个弧两侧的弧碰撞了，也就产生了新的边。

Site Event

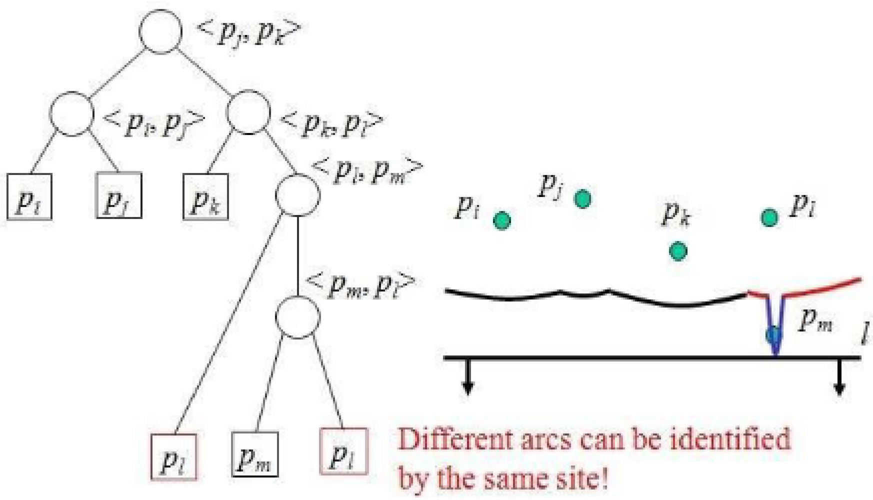
当一个site event发生时，会产生一个新的弧。新的弧往往是诞生在某个弧上的：

一个比较好的存储弧长以及breakpoint的方法是使用二叉树，因为：

1. 需要保留弧与弧之间的左右位置关系
2. 查找弧的位置时候，可以根据site点在弧的左右侧来查找需要插入的位置。当弧的个数比较多时，查找会更加复杂，而使用二叉平衡树可以让这个复杂度从$O(n)$到$O(\lg n)$。



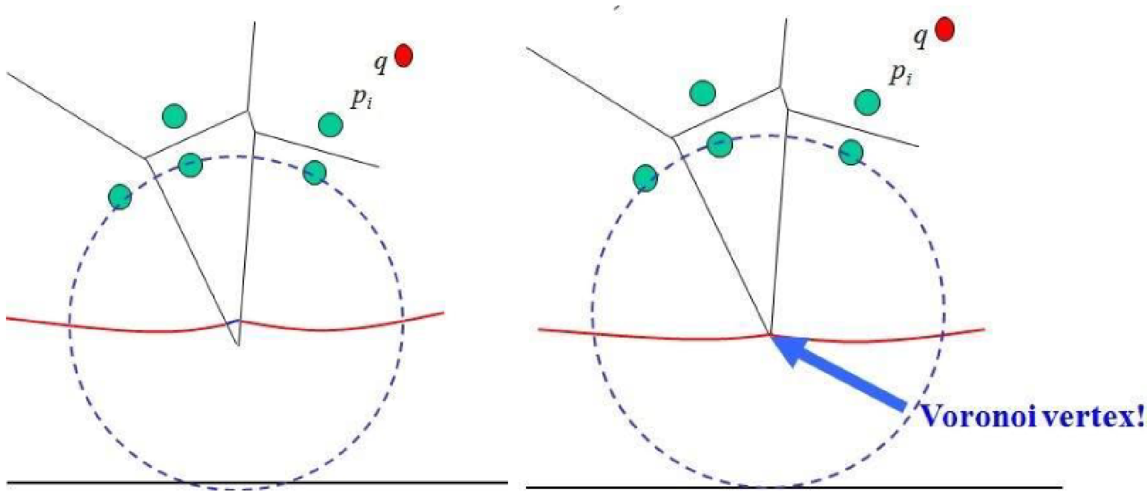
上图中，二叉树的叶子结点指某个弧，而其他结点为break point。需要注意的是某个弧可能出现多次，因为这个弧长是被其他弧割裂了，最简单的就是site event发生时：



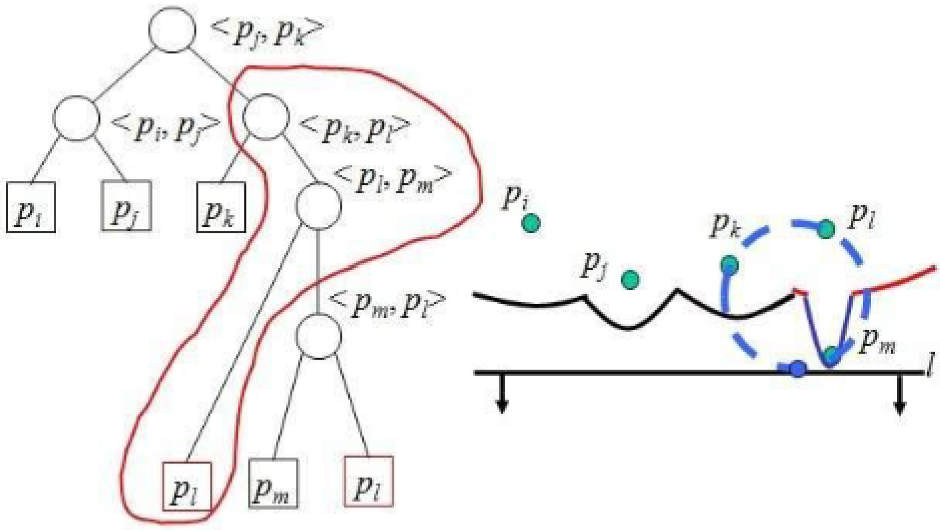
因此Site event发生时，需要做的是在二叉树中，对对应的弧进行分裂，然后进行Circle Event的预测检测，检查出是否有符合条件的circle，加入circle event中。

Circle Event

当发生一个circle event时候，会确定一个voronoi图的顶点。



可以看到的是这之后一定会有一个对应的弧消失了，因此我们需要维护二叉树中这部分，删除掉丢失的弧，并重新恢复好二叉树：



上图中，一部分弧长被删掉了，该弧长为p_l的一部分。这时候二叉树中要删除的部分已经用红色圈出来了。

用来存放Voronoi图的数据结构为DCEL，又叫HalfEdge。每次site event发生时，我们根据break point往halfedge中添加edge，当circle事件发生时，我们往halfedge中添加顶点，并添加一条新的edge。

具体实现

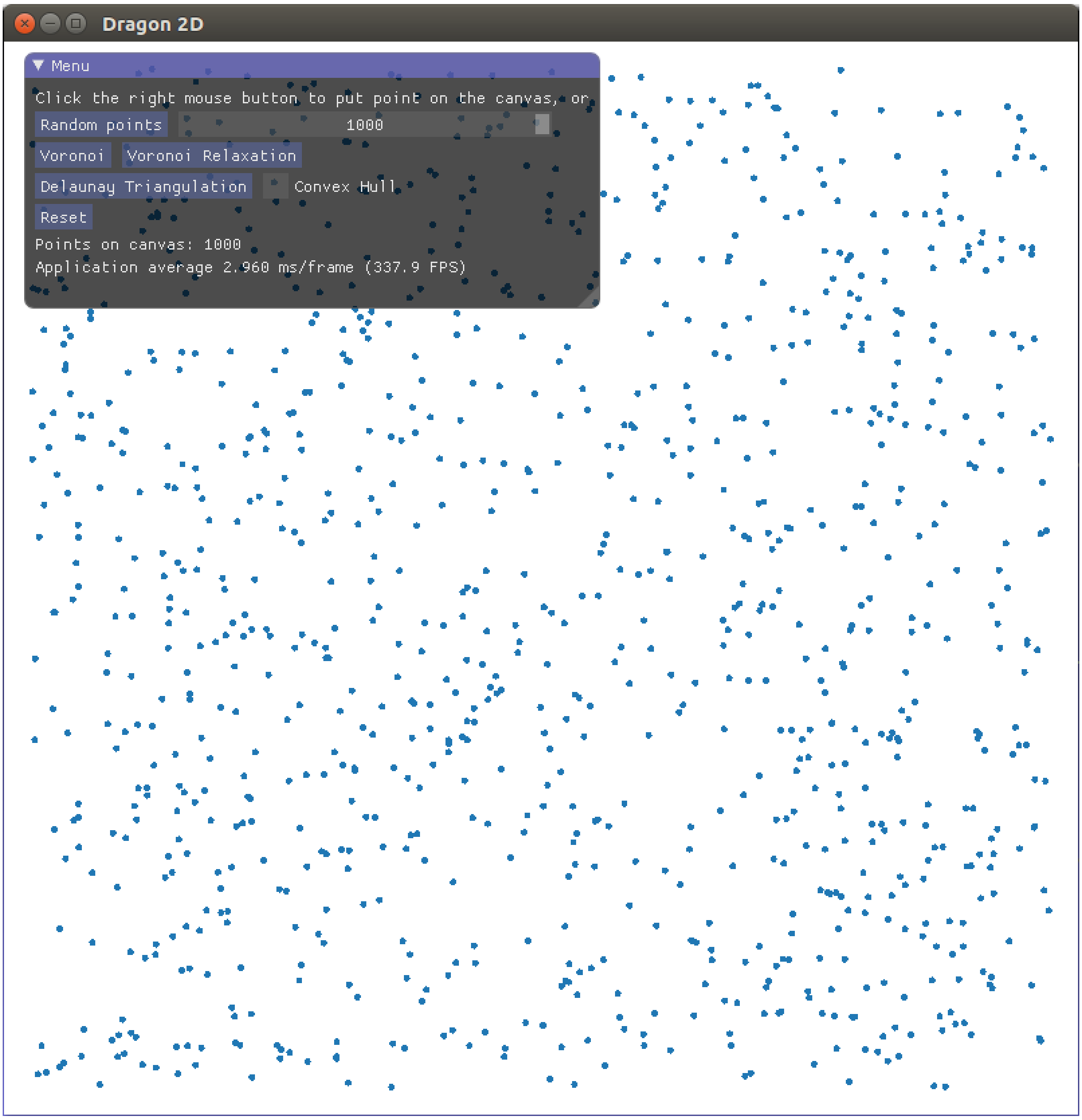
具体实现中做了权衡，也遇到了不少问题。

- 二叉平衡树的实现比较复杂，而且只有在样本量非常大的时候有优势，而普通的二叉树复杂度复杂度依然为$O(n)$，且维护起来较为复杂，因此我使用了最简单的双向链表，便于维护。
- 数值精度问题：判断是否是circle事件时候，当一些corner case发生时，可能会出现circle事件与触发其的site事件坐标一致的情况。但是这两个事件的坐标是分别计算的，因此在数值上可能导致circle事件坐标在在site事件之前。这个时候加上一个微小偏移来判断可以解决，或者使用更高精度。
- 使用包围盒截断问题：直接使用voronoi生成的图有一些cell是边是没有边界的，也有一些边界点会远远超出屏幕的范围，这导致relaxation时候会让site point也移动到屏幕范围之外，因此一般需要使用包围盒来截断。截断过程中要维护每个cell的edge的连接关系，维护顺时针或者逆时针的次序，也有一些边和点会舍弃掉。

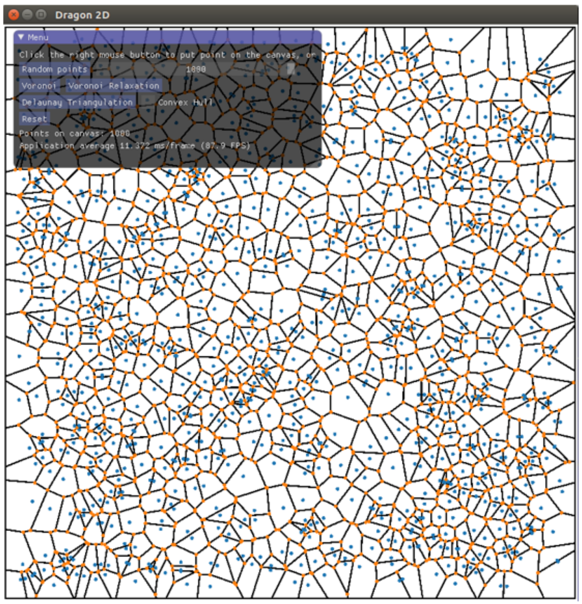
当voronoi图实现后，生成对偶的三角剖分就是Delaunay剖分。这个很简单，只要对生成的HalfEdge中的定点相连接的三个面对应的site point按照一定次序连接即可。另外一个算法就是Floyd算法，有了voronoi图后这个算法实现也是非常容易了。

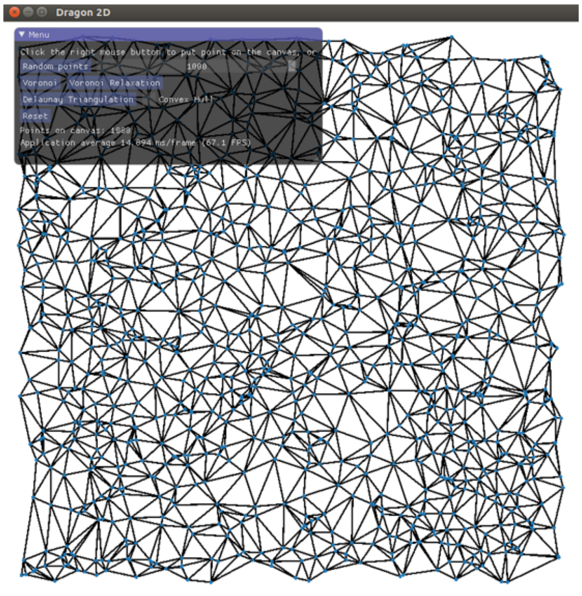
结果

随机生成1000个点

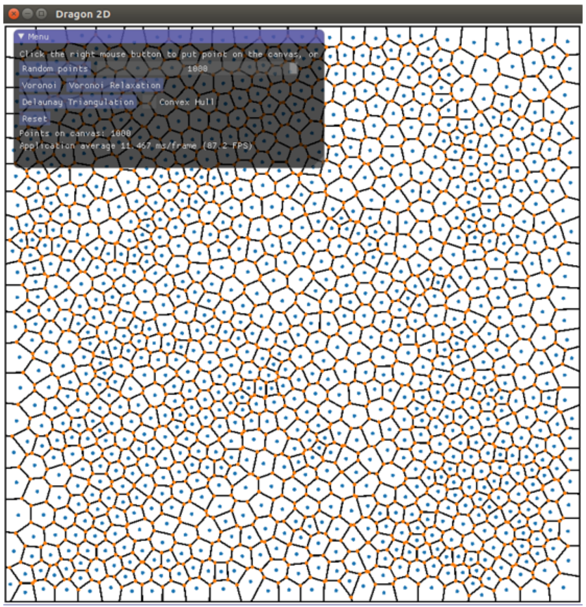


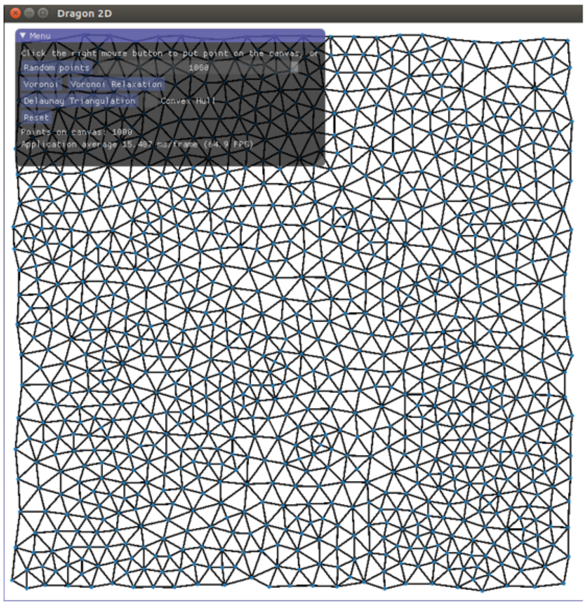
生成的voronoi图以及对应的三角剖分：



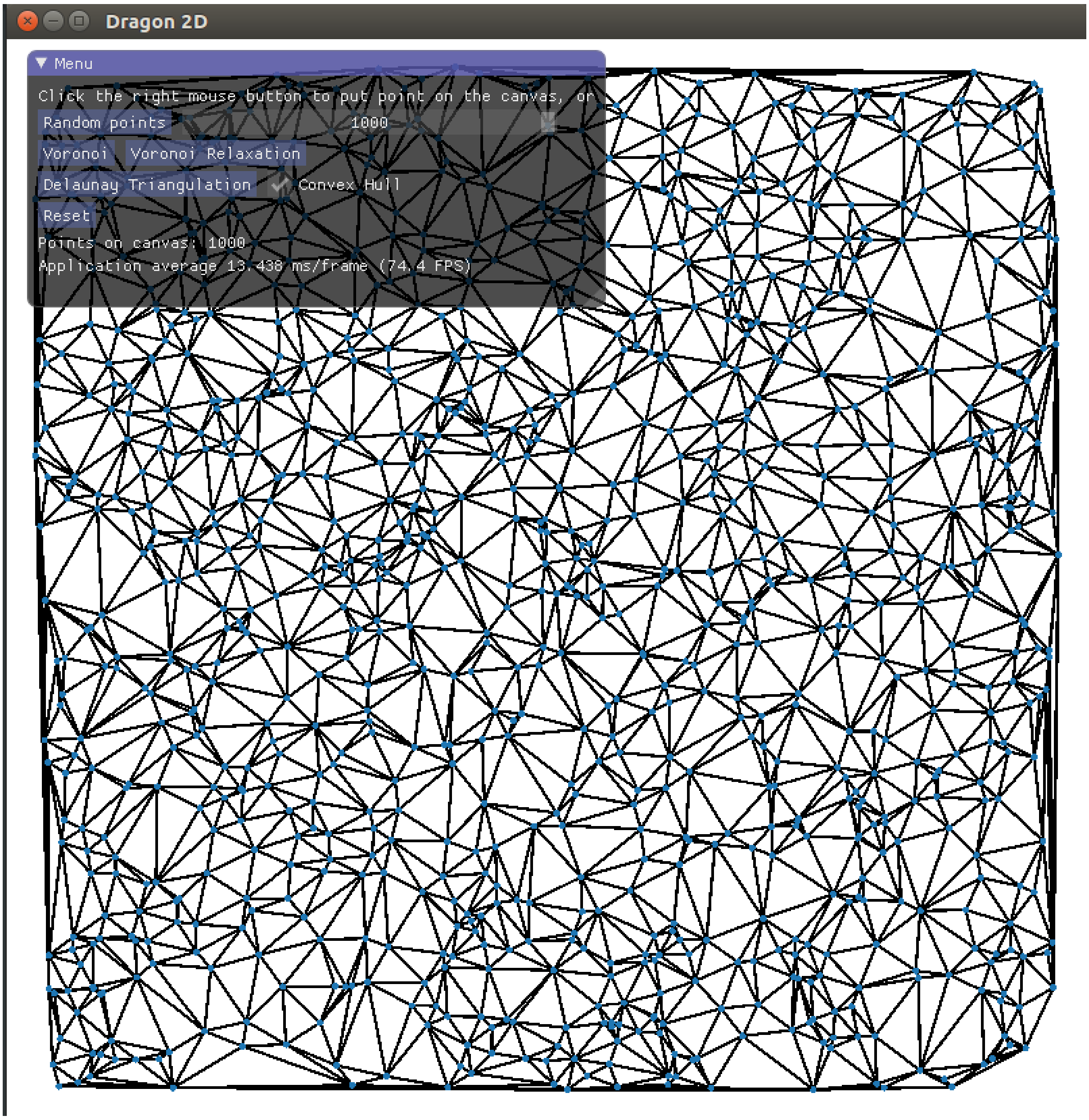


经过Relaxation之后的voronoi图于对应的三角剖分：





这里需要注意一点是，严格意义上来说，voronoi的对偶三角剖分最外层边会保留凸包性质的，这里为了美观将落在屏幕外的HalfEdge的顶点去掉了，因此不是一个凸包，一个完全的Delaunay剖分（保留了convex hull）如下：



代码发布在https://github.com/MyEvolution/Dragon， 更多内容可以查看视频。

以上是本次作业的内容，谢谢老师与助教！