基于 divide and conquer 的平面 Voronoi 图算法

介绍

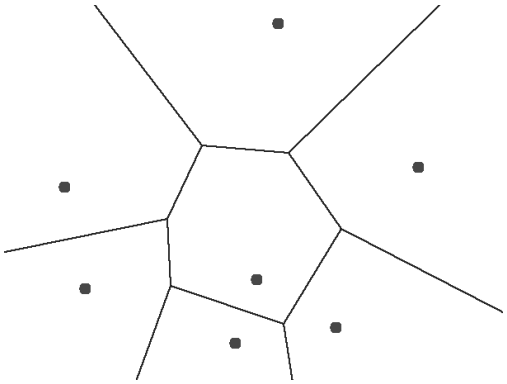
Voronoi 图的计算是计算几何中的一个基本问题。实验采用 divide and conquer 的策略，计算平面点集的 Voronoi 图，并通过动画形式，演示算法的计算方法。

Voronoi 图的定义

E^d中给定一组点$S = \{p_1, \dots, p_n\}$，则 Voronoi cell 定义为：

$$Cell(p_i) = \{q \in E^d : ||q, p_i|| < ||q, p_j||, \forall j \neq i\}$$

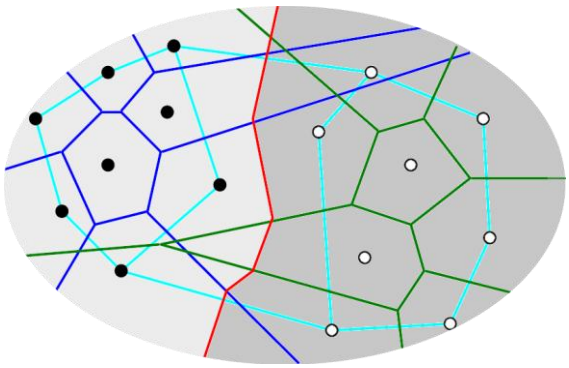
即$Cell(p_i)$点距p_i的距离小于其到任意其他点的距离
给出 Voronoi cell 划分的图为 Voronoi 图。



图：平面 Voronoi 图示例（摘自课件）

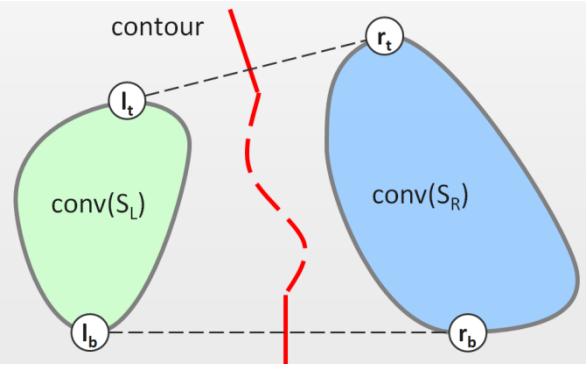
基于 divide and conquer 的 Voronoi 图算法

每次将点集分为左右两部分，分别计算 Voronoi 图后，将其合并。



图：算法示意（摘自课件）

算法主要工作在于合并左右两部分的 Voronoi 图。



图：算法示意（摘自课件）

理论已经证明，可以从左右凸包的上切线的中垂线开始，每次比较左右两边工作点的中垂线的关系，得到一条新的分割线。从而合并左右两张 Voronoi 图。合并时间复杂度为 $O(n+m)$，其中 n、m 为左右点集大小。

合并算法描述：

1. $l = l_t, r = r_t$
2. *b is bisector of* $Edge(l,r)$
3. *compute* $v_1 = b \cap Edge(Cell(l)), v_2 = b \cap Edge(Cell(r))$
4. *compare* v_1, v_2*, find the upper one*
5. *flip the first one to the next cell*
6. *update* l *or* r
7. *if* $l == l_b$ *&&* $r == r_b$*, add bisector of* $Edge(l,r)$
   else goto 2

其中$b \cap Edge(Cell(p_i))$可以利用凸性加速。

时间复杂度

每次合并为$O(n+m)$

$$\therefore T(n) = 2T\left(\frac{n}{2}\right) + O(n)$$

$$\therefore T(n) = O(n \log n)$$

数据结构

程序中用类 MyVoronoi 来计算和存储 Voronoi 图，存储采用 DCEL 结构。
具体说明如下

```
Class MyVoronoi   {
      DcelPoint *dcelp;            //Voronoi 图的点
      DcelEdge *dcele;             // Voronoi 图中的边
      DcelFace *dcelf;             // Voronoi 图的面
      void BuildVoronoiGraph();    //用 Divide&conquer 的方法构造 Voronoi 图
};
Class DcelPoint {
      double p.x & p.y;            //点的坐标
      Int incEdge;                 //指向以该点为端点的一条边
};
```

```
Class DcelEdge {
        Int twinEdge;              //指向 halfedge
        Int oriVertex  ;           //指向该边对应的点
        Int incFace;               //指向它左边的边
        Int prevEdge;              //指向逆时针方向上一条边
        Int nextEdge  ;            //指向逆时针方向下一条边
};
Class DcelFace {
        Int incEdge;               //指向任意的一条围成改变的边
};
```

演示的实现方法

考虑到演示中图的点和边可能比较多，直接绘图速度可能达不到动画的需要，因此采用增量法传递信息。

由于实际构造过程中，每一步都只会从上一步已经得到的图中添加或者删除常数条边，因此实际传输的信息是图像每一步的变化情况，这样可以很快的通过上一帧的图像来绘制下一帧，使动画效果流畅。

具体信息分为 3 种：

MESSAGE#1:绘制线

MESSAGE#2:删除线

MESSAGE#3:绘制交点

演示的全部信息都保存在 MyMessageBox mmb 中，其中的 type 变量就对应信息的种类。

程序说明

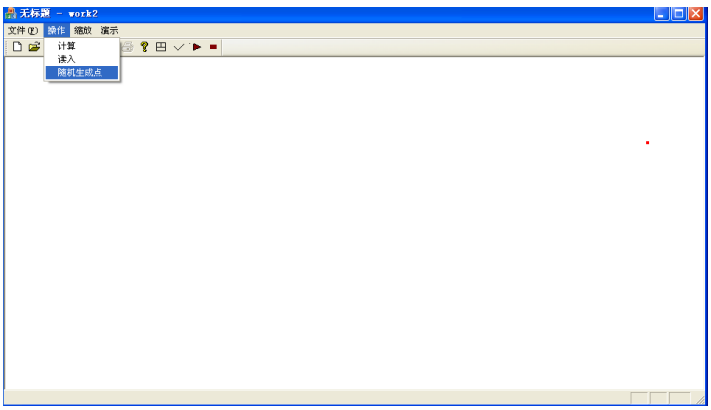
1. 读入数据

读入数据可以采用 3 种模式

手工绘点：默认模式。

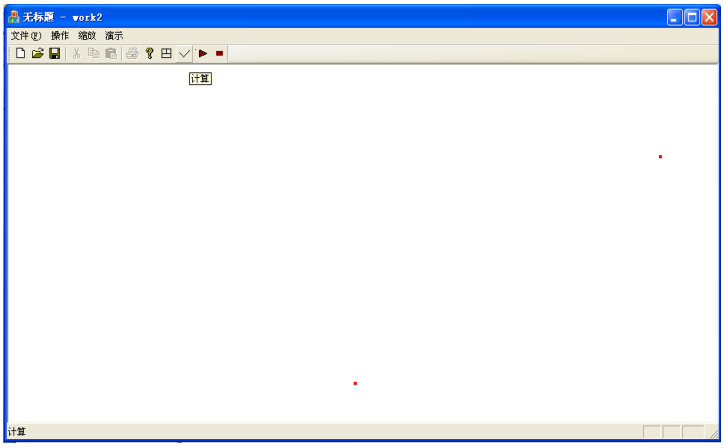
随机生成点：使用菜单操作->随机生成点来实现。

从文件读入：使用菜单操作->读入来实现。



2. 计算

通过菜单操作->计算来实现，也可以通过 TOOLBAR 上的对应按钮来计算。



3. 自动演示

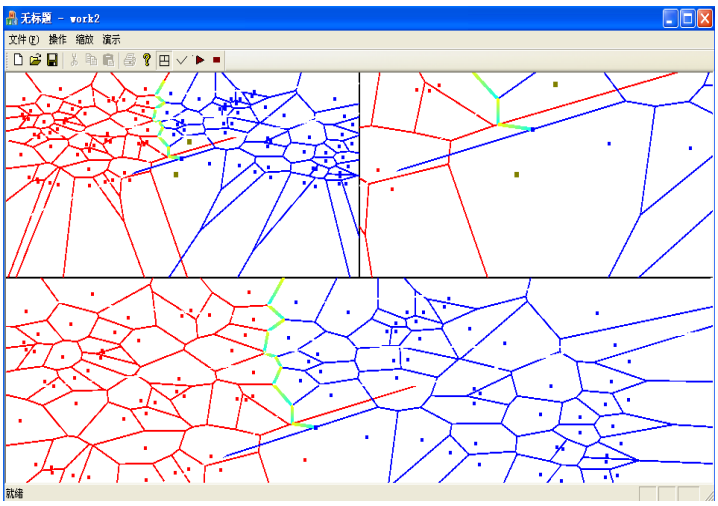
注意，如果点数较大时，该功能对机器性能要求较高，因此不推荐使用。通常情况下，1000 个点可以非常流畅的运行起来。

运行时需要设置两个参数，分别是延时和层数。

延时：决定演示的速度，点多时推荐 20，点少时可酌情增加。

层数：决定演示最后多少层，N 比较大时推荐演示层数不要超过 3，否则因为点太多观看效果不佳。

特别的，当点很多时，可以切换视图，切换后会出现两个新视图，分别按当前两点距离的 1.1 倍和 5 倍缩放，方便清晰的观察计算过程。

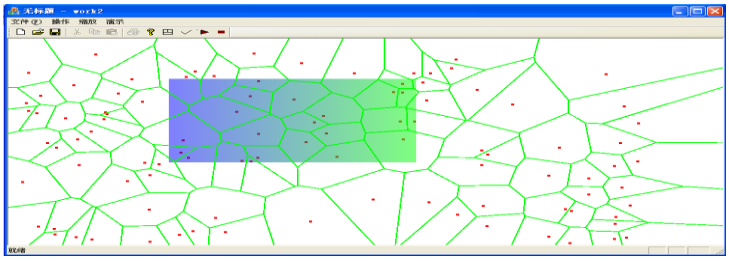


彩色线是 Bisector,红的的点是左边的点，蓝色的点是右边的点，土黄色的点是当前处理的 CELL 的点，灰色的点是与本次操作无关的点。

4. 拖拽和缩放

拖拽：直接按住鼠标左键拖拽即可。

缩放：通过菜单缩放->选择区域来实现。



实验心得

通过本次实验，我们掌握了基于 divide and conquer 构造 Voronoi 图的算法，也熟悉了 MFC 和 OPENGL 编程的一些技巧。我们队对实验进行很好的分工，制定了比较详细的接口方案，这些团队合作的经验都是非常宝贵的。

我还想谈一下我个人心得。通过本次实验，我觉得计算几何问题在具体实现的时候绝对不能轻视一些边角情况，例如三点共线或者存在垂直线段等等。起初由于忽视了对这些特殊情况的处理，我们程序一直不能正常运行，浪费了相当长的时间，我觉得以后应该更加注视这些不起眼的因素，做出完美的程序来。