Voronoi 图的 Beachline 扫描算法的三维图形演示

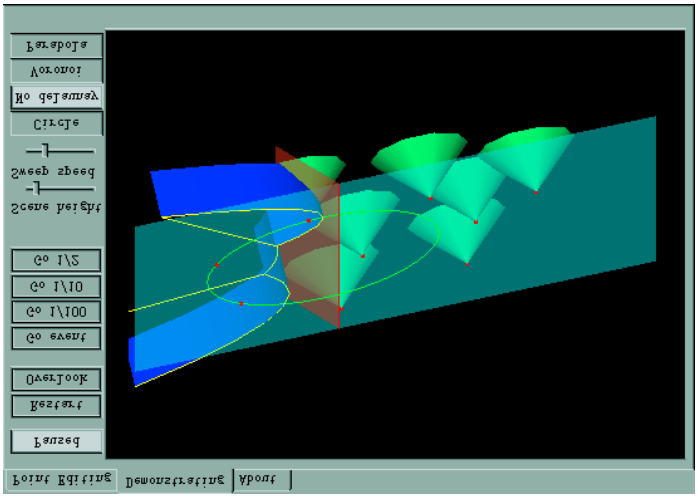
王益，王垠，潘达

"I found the animation very entertaining"
- Steven Fortune

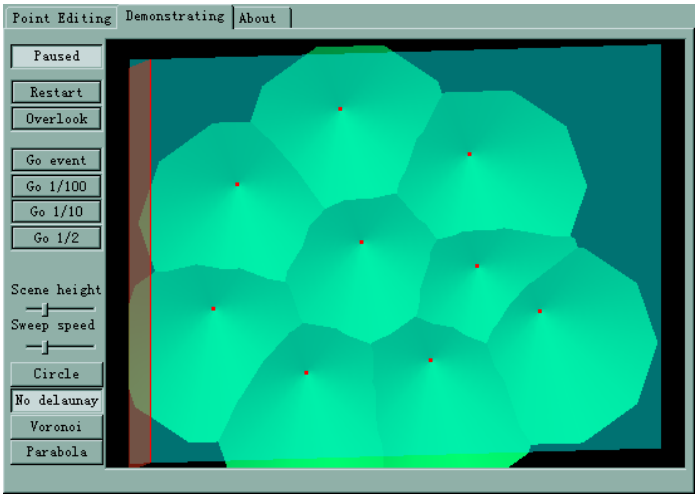
1. 问题背景

本次大实验我们用三维图形程序来演示 Voronoi 图的 Beachline 算法。我们的主要目的是用直观的方式来演示 Beachline 扫描算法以辅助计算几何的教学。以前我们见到的 Java applet (*Allan Odgaard & Benny Nielsen)* 是用二维的方式演示 Beachline 扫描算法生成 Voronoi 图的过程。我们想把这个过程的本质思想演示出来， 所以做了这样一个三维程序。它的主要功能是：

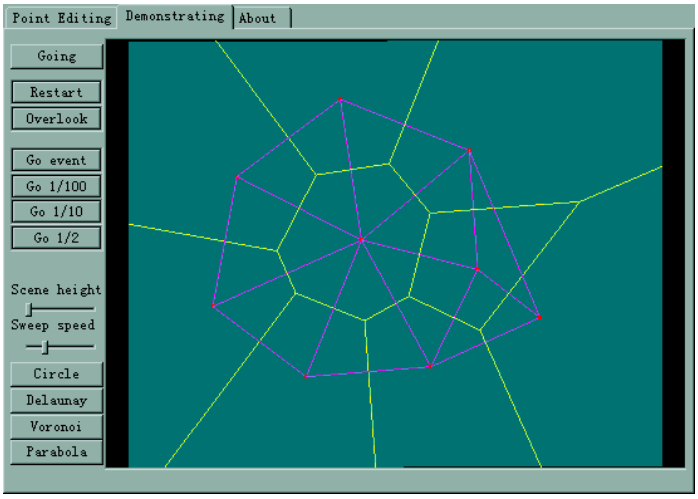
- 把二位空间中的点表示成三维空间中的无穷大的圆锥，然后用一个斜平面扫描这些圆锥，当平面切过圆锥时，产生的截面投影到上方平面就是扫描线的 Beachline.



- 即使不扫描，从上方看这些圆锥的交线，实际上就是一个 Voronoi 图。



- 同时还能描绘 Delaunay Triangulation.



- 还有其它方便的功能。

2. 算法及原理

我们演示的算法是 Steven Fortune 的 Beachline 扫描算法。具体内容请参考 Fortune 的论文 *A Sweepline Algorithm for Voronoi Diagrams.* 在我们的工作过程中，Fortune 先生提供这篇论文和热情的帮助。

3. 系统设计

我们的程序代码分为前端和后端。

前端程序负责 GUI 界面，点的输入，保存和读取，三维图形（圆锥，切平面，Voronoi 边，Delaunay 边，圆事件）的绘制以及旋转等功能。

后端主要是实现一个完整的 Beachline 算法。我们将他封装再一个 lib 中。前端程序通过一个 C header file 调用这个库中的 API 函数。这个库既可以用于演示，也可以用于 Voronoi 的工程计算。而以往的 Voronoi Diagram 算法通常不能兼顾二者。

前端程序

前端程序使用 MFC 和 OpenGL 实现。

后端程序

后端程序使用标准 C 语言实现。其中加入了一些修改后的 Steven Fortune 的 C 语言代码。

数据结构：沿用了 Fortune 代码的存储描绘 Voronoi 图的一种类似 DCEL 的结构，这种结构更容易用于实现扫描算法。 因为这里双向链接的不是边而是半边(Halfedge). 边 (Edge) 是挂在半边结构的一个指针上的，而 Halfedge 形成一个双向链表，而事件产生时我们需要搜索的就是这种生长的 Halfedge 结构。 一条边最多可以挂在两个 Halfedge 上。 关于 Halfedge 的细节请参考 Fortune 的论文或者代码。Halfedge 和圆事件都是放在分桶的链表里的，这样可以加速搜索。Fortune 原始程序中 Edge 记录了这条边的直线方程 (ax+by = c)，它的两个端点，以及相邻的 Site. 为了演示方便，避免每次前端调用接口时都去计算 Halfedge 与抛物线的交点，所以为 Edge 结构加入了两个成员 fep[2] 来表示这条边当前生长到的位置。

程序实现：Fortune 的程序是不断的取出事件点，最后构造出整个 Voronoi 图。而我们需要演示扫描线在任何中间位置的图形，所以必须加入对 Halfedge 当前生长到的位置，以及抛物线控制点，抛物线范围的确定。其中有很多特殊情况需要处理，比如防止尖角的抛物线产生，Halfedge 与抛物线求交过程中的问题，退化的抛物线的处理，防止垂直边导致浮点数

NAN，或者出现不该出现的抛物线等问题。

前端程序与后端程序的交互：

后端程序向前端程序提供几种接口：

1.在开始演示时，前端程序调用接口

```
void InitVoronoiConstruction(void);
```

告诉后端程序初始化数据结构。

2.与之对应，前端程序调用接口

```
void UninitVoronoiConstruction(void);
```

告诉后端程序释放所有数据结构和分配的内存。

3.前端程序提供方便的点的输入，删除，存取盘操作。在接受用户输入点后，把这些点做成一个后端程序能识别的结构数组，交给后端程序处理。这是由前端程序调用接口

```
void RestartVoronoiConstruction(Site *, int)
```

实现的。

4.演示的时候，前端程序负责接受用户的操作并且使用接口

```
int NotifyLineChanged(float );
float/*newy*/ NotifyMoveToNextEvent(void);
```

来告知后端程序扫描线的移动或者跳到下一个事件产生的位置。

5.前端程序移动了扫描线，或者要求移动到下个事件点后，需要重新绘制图形时，调用

```
void GetCurrentContent(struct RCircle **, int *,
                       struct RParabola **, int *,
                       struct REdge **, int *,
                       struct REdge **, int *);
```

来得到需要绘制的图形。

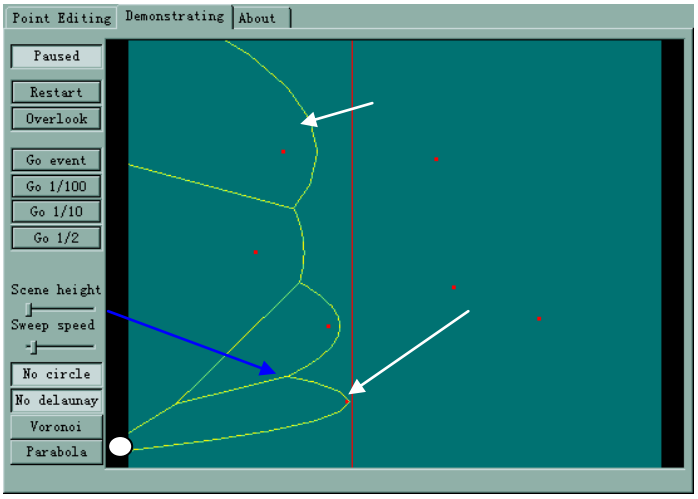
4. 实现过程中遇到的问题及解决对策

实现过程中有很多简单的细节问题，这里就不赘述了。这里说明一下几何算法演示程序经常需要注意的问题。

1. 尖角抛物线及其连带的问题。实现这个演示程序之前，我们参考了 *Allan Odgaard & Benny Nielsen* 的 Java 源程序，他们的抛物线使用直线段来逼近的，这种方法适合二位的曲线绘制，但是为了充分利用三维图形库的功能，需要给出某一时刻的所有存在的抛物线的 3 个控制点的位置，以及抛物线的绘制范围。由此引入了很多问题。

这里我们假设扫描线是由下往上扫的。

比如最左和最右边抛物线的控制点的合理取值。如图中的两个白色箭头指向的部分就是最两头的抛物线，它们是无穷大的。比如，最右边的抛物线，如果我们取右边边框处的抛物线上的点作为右边的控制点，那么当扫描线离交点很近时，这个点在很下面，它的 Y 坐标是很大一个负数，那里的斜率很大，我们由此得到很高的一个上面的控制点。这样画出来的抛物线很尖，不像一个抛物线，而且与左边的抛物线的右边沿不能缝合（蓝色箭头所示）。可惜这里是正确绘制的图形，我们没有想到抓下错误图形的样子。

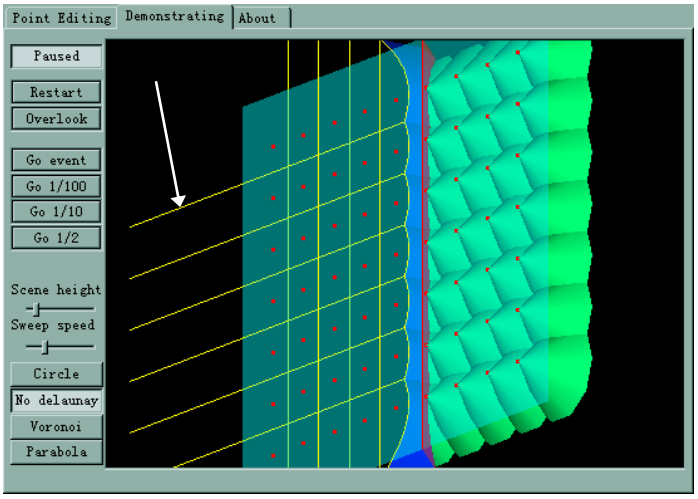


我们的解决办法是如果抛物线的最右边在我们确定的框以下，那么我们就取这个抛物线的右边与框下边沿的交点为控制点（白点所示）。

但是由此又引出一个问题，那就是：因为抛物线是根据生长的半边产生的，它的左右绘制范围是又半边与抛物线求交得到的，如果半边与抛物线交与焦点右边，而且交点的纵坐标低于下边框，而这条抛物线正好又是最右边那条抛物线，而且它的最右边也落在下边框下。本来我们应该画出的是一条抛物线在焦点右边的一段，但是按照刚才的思路，我们应该取当 Y＝0 时的点作为控制点，我们就该画出抛物线在下边框上方的所有部分：一个对称的抛物线。这是错误的。为了解决这个问题，而不引入新的问题，我们决定：需要绘制的抛物线段，如果两个端点都在焦点的一边，而且落在边框外，那么这条抛物线不画，而这并不影响演示效果。

2. 多点重合：如果多点坐标相同，那么算法会给出错误的结果，因为我们是假设没有两点的坐标完全相同。这个问题可以又后端，也可以有前端解决，最后我们商量后，决定由前端程序负责阻止用户输入相同坐标的点。

3. 垂直边如果两个 site 的 Y 坐标相同，那么就会生成垂直的 Voronoi 边。如图中白色箭头所示。因为抛物线总是夹在两个 Halfedge 之间，我们绘制抛物线时是通过遍历 Halfedge 列表来画出所有 Halfedge 两边的抛物线来实现的。我们通过 Halfedge 和相邻的抛物线的交点来判断抛物线的绘制范围。所以垂直向下生长的 Halfedge 引入一个问题：它左边的抛物线怎么画？答案是：它左边根本就没有抛物线。所以这种垂直向下的 Halfedge 产生后就应该立即从半边列表里去掉，以免画出不应该出现的抛物线。后端程序通过为垂直向下的 Halfedge 对应的 Edge 的下端点指定一个很大的负数来实现绘图时的端点确定。工程实际中不需要这样做。



5. 测试、对比结果

根据测试，这个程序能处理很多点的情况，以及特殊情况：比如 3 个以上的点共圆，点事件和圆事件重合，纵坐标相同的点，横坐标相同的点。

而且这个程序提供了方便的输入点的坐标，以及存取盘的功能。这样可以精确的演示算法的各种特殊情况的处理。

6. 没有解决的问题

这个演示程序使用了 Fortune 代码里的分桶链表结构，这种结构的查找效率不是最高的，所以实际上这个后端算法程序不能达到 Beachline 算法的 O(nlogn)的时间复杂度。如果换用平衡树结构，会得到 O(nlogn)的结果。但是平衡树结构很复杂，所以为了简单，我们还是使用了原来的分桶链表。

后端为前端提供的接口是专门用于演示画图的，如果要用于工程应用，可以去掉最后的 final_stretch（）函数， 因为为了提供给绘图的前端一个清晰的接口，这个函数会把所有最后还在长的 Halfedge 都延伸到绘图的边框处。否则如果由前端处理射线，就需要提供给前端直线的方程等参数，加重前端的复杂性。

所以为了工程应用，需要

- 使用平衡数结构
- 去掉多余的代码
- 返回射线的方程等数据

7. 有关文献及代码来源

参考文献主要有：

1) A Sweepline Algorithm for Voronoi Diagrams, Steven Fortune.
2) Voronoi Diagrams and Delaunay Triangulations, Steven Fortune.
3) Computational Geometry, Algorithms and Applications.

代码来源：借用了一些 Steven Fortune 的 C 代码。见附件。