基于 Kirkpatrick 结构的点定位算法实现

1 问题背景

点定位（Point Location）问题在日常有广泛的应用。医疗方面，点定位主要应用在射频手术中，手术需要把导管通过静脉穿刺深入到病人的心脏疾病所在的地方，再通过激光的手段进行治疗；导管头部的精确定位能够让导管不触及心脏其他部位，降低了手术的危险性，同时能够在最关键的部位进行手术，使手术得到良好的效果。故障处理方面，故障发生之后救援工作的进行需要对发生故障的位置进行区域的划分，这是进行快捷而有效的救援的基础，只需要根据故障的破坏性程度安排在故障点周围的救援人员前去排出故障即可。导航方面，导航索要解决的是把绝对的位置转化成所处的相对区域的问题，从而判断出继续前进的方向。

点定位所要解决的是从总体上来看把一个或几个定位点划分到具体的小区域的过程。首先，要对全局进行预处理；其次在预处理的基础上，把定位点归入到某个划分小区域中。它的算法复杂度包括预处理复杂度和查询复杂度两个方面。

2 点定位算法介绍

2.1 Slabs

依据多边形划分的每个顶点，作竖直的分界线，从而获得不同的 slab。对于任意一个 slab S，它与多边形划分不同边的相交的部分，把其再划分成不同的部分，如 Figure.1 所示。这些子划分可以通过两条在竖直方向上连续的边唯一确定。在对定位点确定了它属于哪个 slab 的那个子划分之后，就可以依据子划分与原始的多边形划分的关系得到点定位。

这一算法在进行预处理的时候要对多边形划分的所有顶点进行排序，根据排序结果获得 slab 的排序，且此要对多边形划分的每一条边都与每一个 slab 求交，

并对求交的结果再进行排序，最后将每个子划分与原始的多边形划分建立起所属的联系，所以预处理的复杂度为 $o(n^2)$。在查询的时候，需要在 slab 排好序的基础上确定定位点所处的 slab，之后在排好序的子划分中查找确切的子划分，并通过映射得到定位结果，所以查询的复杂度为 $o(\log(n))$。

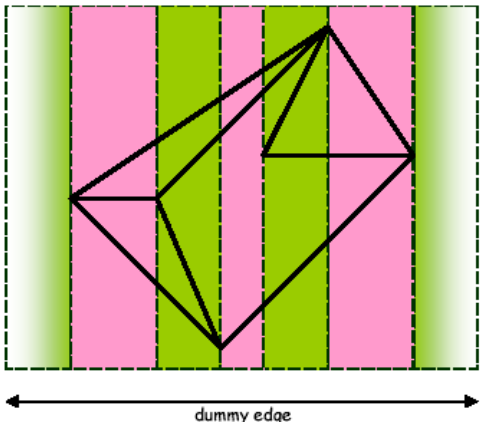

Figure.1 Slabs 分解

2.2 Path/Node-Copying

Path/Node-Copying 是在 Slabs 方法的基础上进行的优化。主要思想是每一个 slab 内的子划分用树的形式表示，在对某个 slab 建树的时候，可以应用到前面一个 slab 的树的结果，也就是说要在前一个树中找到当前 slab 与前一个 slab 所共有的分界顶点的位置，两个树所不同的部分正是从这个顶点开始的。Path/Node-Copying 的算法采用这种思想，但是并不是简单的对前一个 slab 的树的节点进行变换，而是把前一棵树的从根节点到分界的基点的所有 $n\log(n)$ 个节点全部拷贝到当前树中，然后再进行必要的修改。可以证明，采用这种表示虽然查询复杂度不会变小，但是存储的复杂度会减小到 $o(n\log(n))$。

这两种的方法的复杂度分别为：

	存储复杂度	查询复杂度	预处理复杂度
Slabs	$o(n^2)$	$o(\log(n))$	$o(\text{nlog(n)})$
Path/Node-Copying	$o(n\log(n))$	$o(\log(n))$	$o(\text{nlog(n)})$

Table.1 两种算法的复杂度

2.3 Kirkpatrick Structure

Kirkpatrick 算法是基于层级结构的一种进行平面点定位的最优算法。它的预处理复杂度可以达到$o(\log(n))$，查询的空间和时间复杂度都可以降到$o(n)$。层级结构中每一层表示的是一种独立的三角形划分，而层级结构之间则用三角形划分的交叠关系联系起来。

但是对于一般的情况，多边形划分并不是理想的三角形划分。所以建立层级结构之前，需要根据以下几个步骤得到三角形划分：

（1）求得多边形划分的凸包，凸包的外接三角形可以看作三角形划分树中的根节点。

（2）对多边形划分凸包内的任意多边形划分进行三角化。

（3）对凸包以外、外接三角形以内的部分进行三角化，从而在外接三角形内都是三角形划分。

经过上面三个步骤，多边形划分转变为三角形划分。为了查找每个三角形划分都需要加一个标示，标明他原始的多边形划分。普通的三角形划分并不一定能够像理想的三角形划分那样用树结构来表示。经过三角化以后，只得到了它的根节点（外接三角形划分）和叶子节点（每个不可分三角形划分），如 Figure.2 所示。

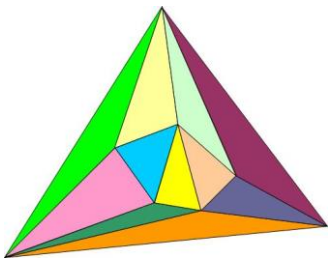

Figure.2 对多边形划分及外接三角形三角化

根据 Kirkpatrick 算法，从底层的叶子节点开始，每次取得新的三角形划分，新的三角形划分中三角形的个数会少于当前的三角形划分中的个数。在生成新的三角形划分的时候，并不是每个三角形划分都记入数据结构中，而只对两层之间不一致的部分作记录，并对不一致的部分内的三角形的交叠关系作记录；对于两层之间一致的部分，可以直接从原始层中引用。不断的生成新的三角形划分，直到最后的三角形划分中只包含一个三角形，这就是层极结构的根节点。具体算法

在 3.1 节中介绍。

以上是生成层级结构的预处理部分，它的算法复杂度为 $o(n\log(n))$。

查询需要根据层级结构进行。从根节点开始，确定定位点所在的三角形，根据不同层级结构三角形划分之间的交叠关系，在下一层中被交叠的一组三角形中进行查询，得到定位点所在的三角形。如此往复进行，当查询进行到最后一层叶子节点的时候结束，根据叶子节点本身的标识（在预处理阶段第一步得到），可以得到定位点的三角形划分的归属。

Kirkpatrick 算法空间上的复杂度为 $o(n)$，查询时间上的复杂度为 $o(n)$，预处理的复杂度为 $o(n\log(n))$。理论上讲，它是点定位的最优算法，但实际上各项复杂所包含的常数因子会很大，所以实际的性能并不是最好的。

3 Kirkpatrick 算法的具体实现

在第 2 节中简单介绍了一下 Kirkpatrick 算法的内容。下面给出具体的实现：

3.1 预处理的整体流程

预处理就是建立层级结构的过程。输入的多边形划分首先要作增强，即求外围的三角形，然后对增强后的多边形划分三角化。三角化结果作为层级结构的最底层，同时需要将结果与原始图作从属标记。从最底层节点开始，每次求得更高一层的结构，直到求得顶层的根节点，只剩下 3 个顶点，即只包括 1 个三角形。

从当前层结构建立更高一层的结构，首先要找到当前顶点集合中的一组独立顶点，详见 3.2 节；其次从当前顶点集合中删除这些独立顶点，删除之后当前的划分会出现一些新的多边形面片（空洞）；最后对新的多边形面片重新三角化，并求新得到的三角形与原始三角形的交叠关系。见 Figure.3。

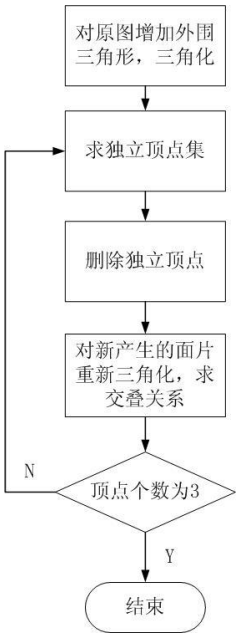

Figure.3 初始化流程

Figure.4 所示的是生成层级结构的迭代过程。T_0 是对原图进行增强和三角化之后的结果，T_4 是作为根节点的最大的三角形。

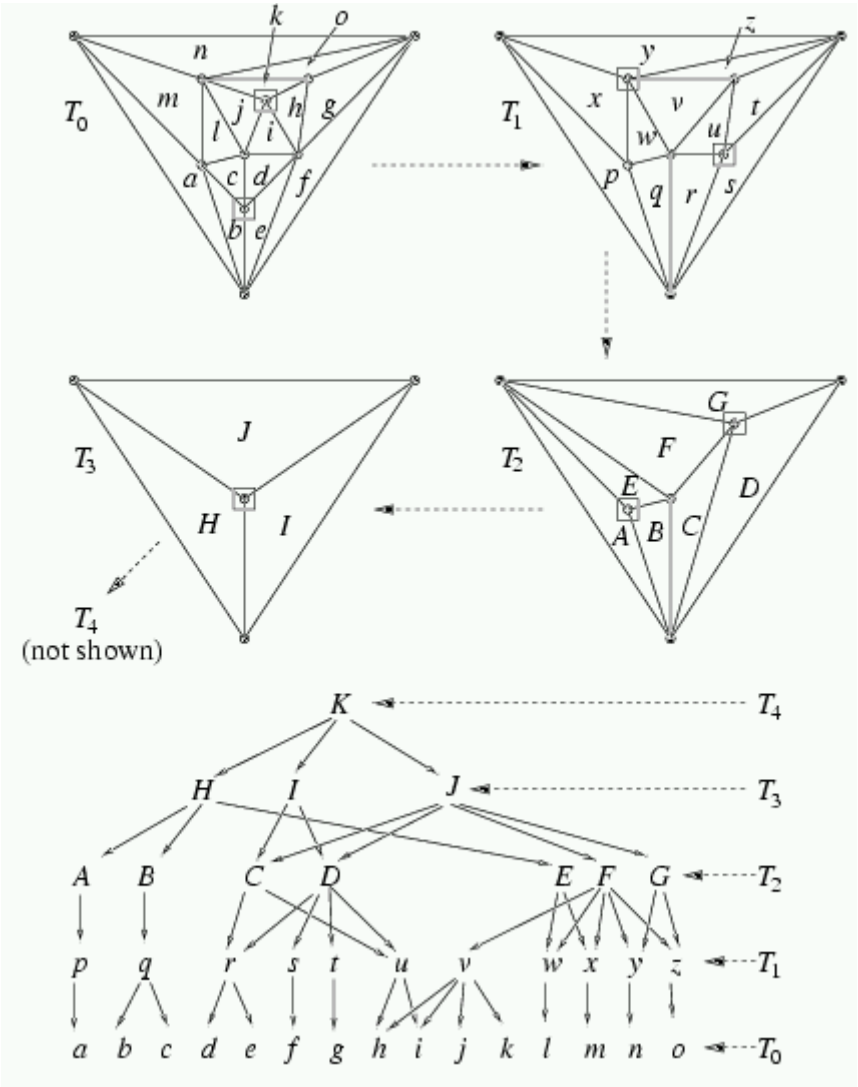

Figure.4 层级结构的生成过程

3.2 求独立顶点集

前提 1：为了保证层级结构的深度小于 $o(\log(n))$，每层独立顶点集的顶点数需要足够大。应当至少与它的全集的顶点数有常数比例的关系。

前提 2：存储和查询时间的复杂度与三角形划分之间的交叠成正比，因此为了节省空间和时间复杂度，新生成的三角形划分与原始的三角形的划分之间的交叠应当尽量的少，至多为常数个，也就是独立顶点集内的每个顶点的度应当为常

数。

根据 Kirkpatrick 的结论，对包括 n 个顶点的三角化结果，存在至少包含 $\frac{n}{18}$ 个顶点的独立顶点集，满足独立顶点集中的每个顶点的度≤8。前提 1 和前提 2 同时得到了满足。

```
Algorithm Construct-Independent-Set(Graph G=(V, E))
{
//    Initialize a candidate set of vertices whose degrees are no more than 11
      Set Candidate =φ;
      for (each vertex v ∈ V)
          if (v.degree ≤ 11) Candidate = Candidate ∪ {v};
//    Assertion: as mentioned earlier, at this moment, card(Candidate) ≥ card(V)/2
//    Initialize an empty independent set
      Set Independent =φ;
//    Find independent vertices incrementally
      while (Candidate ≠φ) {
          Choose a vertex v ∈ Candidate;
          Independent = Independent ∪ {v};
          Candidate = Candidate \ {v};
          for (each neighbor u of v) //no more than 11 neighbors
          Candidate = Candidate \ {u}; //will be removed
       }
       return(Independent);
}
```

Figure.5 求独立顶点的算法

3.3 三角化

实验中三角化采用的是 Graham Scan 算法。该算法的时间算法复杂度是 $o(kn)$，其中，$k-1$ 是多边形内角大于 180° 的顶点的个数。这种算法表面上看起来并不是很理想，但是注意到 k 的含义，在本实验的应用中较少出现内角大于 180° 的情况，所以 k 可以认为是个很小的常数，甚至可能是 0，这样 Graham Scan 的算法的时间复杂度就会变为 $o(n)$，对于点定位应用来说，该算法具有较高的效率。

3.4 三角形划分的交叠关系

由 3.2 中所述，独立顶点集中的每个顶点的度≤8，所以新生成的三角形划分最多与 8 个初始的三角形划分存在交叠关系。

如 Figure.4 所示，去掉某个独立顶点后形成的多边形空洞，它所覆盖的所有初始三角形以去掉的顶点为公共顶点。因此可以根据去掉得顶点v与新生成的三角形划分T的位置关系分以下三种情况来讨论：

A）v在T内部的时候，T与所有以v为顶点的三角形划分都有交叠关系，即覆盖以v为中心的一组初始三角形。如 Figure.6 所示。

B）v在T外部的时候，因为T的三个顶点都是多边形划分的顶点，而 v对每个多边形的顶点都是可见的，所以v也对T的所有顶点都是可见的。因为去掉v顶点后形成的多边形划分的每一条边与v组成的三角型都是初始的三角形划分之一。所以v通过T可见的所有初始三角形划分都被T交叠。以T的三条边中距离v最近的边作为分界面，多边形的边中所有与T同侧的边与v构成的三角形划分都被T交叠。如 Figure.7 所示。

C）v在T边上的时候，可以看作距离v最近的边就是它所在的边。从而仍然可以按照 B）中所述的方法求解。

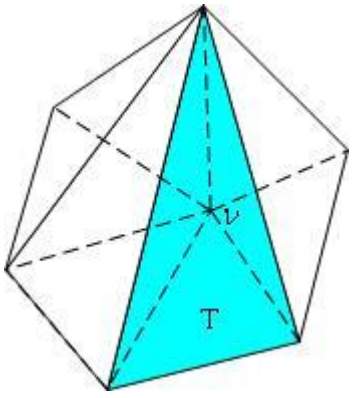

Figure.6 v在T内部

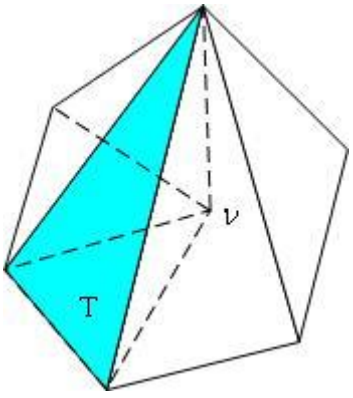

Figure.7 v在T外部

3.4 查询

查询时从根节点出发，根据层级之间的连接关系，可以从根节点三角形划分查询到底层的叶子节点，查询过程如 Figure.8 所示：

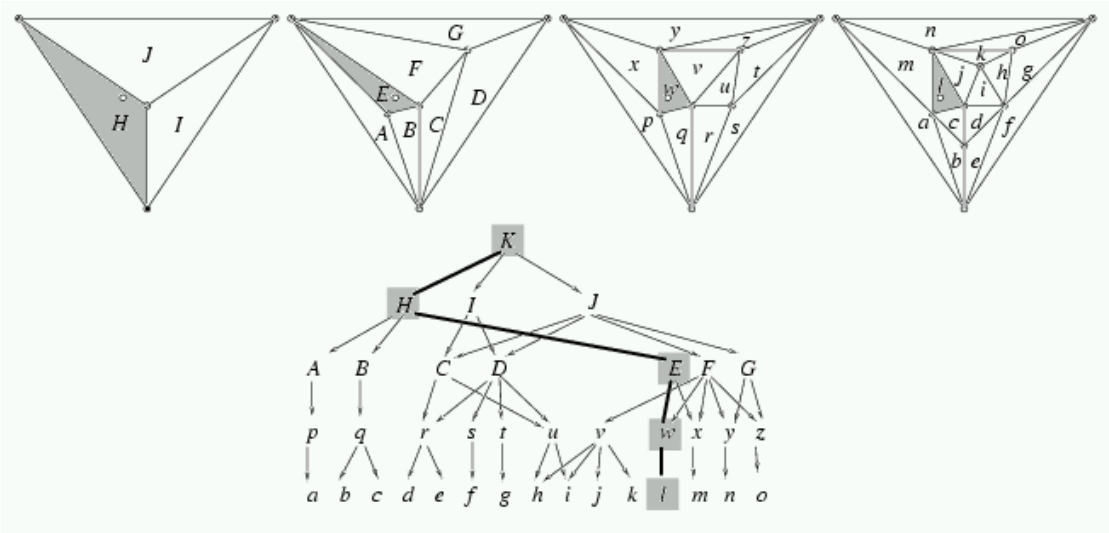

Figure.8 层级结构查询过程

4 Kirkpatrick 算法分析

Kirkpatrick 算法能够使空间的复杂度降为 $o(n)$，查询时间上的复杂度为 $o(n)$，预处理的复杂度为 $o(n\log(n))$，理论上分析是最优的算法，但是实际应该用中有以下几点缺陷：

A 算法复杂度中的常系数很大，存储部分的复杂度常系数是 18，预处理部分得复杂度常系数是 11，查询部分得复杂度常系数于层级之间的交叠关系有关。这样对于很简单只有几个多边形划分组成的图，很大的常系数就成为了重点，效果反而不好。

B 在对原始图增强和三角化的过程中，需要加入冗余的边。对于很难进行三角化的多边形划分，加入的冗余的边的数量会很多，本算法对这些多边形划分的处理的效率会变得很低。

C 这个算法是以多边形划分为基础的，即每个区域的边界都是规整的多边形，这与实际应用中大多数的边界都非纯直线的不吻合，因此很难在实际应用中得到推广。

5 系统设计与数据结构

我们编程实现了 Kirkpatrick 算法。实验环境为 WinXP, Visual Studio 6.0，全

部程序使用 C++语言实现。程序主要包括三个模块：输入图结构模块，创建 Kirkpatrick 结构模块和点查询模块。图结构的创建封装在 CDCEL 类中，Kirkpatrick 结构的创建和点查询封装在 CKirkHierStruct 类中，多边形的三角化装在 CTriangulation 类中。创建 Kirkpatrick 结构和点查询过程能够同时进行二维和三维显示。其中，CDrawForm 类用来绘制二维的图形，用 MFC 实现，CKirkAppView 类用来绘制三维的图形，用 OpenGL 实现。

算法实现过程中涉及到的主要数据结构有两个：描述图结构的 DCEL 和描述层级结构 DAG。

DCEL 描述了点、线、面的数据结构，如 Figure.9 所示：

```
// Definition of VERTEX type
//in DCEL

typedef float _coord;
//coordinate type

typedef struct {

int id; //ID#

_coord x, y; //coordinates

_dEdge* inc_edge; //incident
//(directed) edges

} _vertex; //vertex
```

```
// Definition of EDGE type in
//DCEL

typedef struct {

int id; //ID#

_dEdge* twin; //twin edge
_vertex* org; //origin vertex.
_face* left; //left face
_dEdge* prev; //previous
edge counterclockwisely
along the left incident face
_dEdge* next; //next edge
counterclockwisely along
the left incident face   }
_dEdge; //directed half-edge
```

```
// Definition of FACE
type in DCEL

typedef struct {

int id; //ID number

_dEdge* inc_edge;

//a half-edge taking this
// face as its incident
face

} _face; //face
```

Figure.9 DCEL 定义点、线、面的数据结构

DAG 描述了层级结构。其中，DAGNode 描述了层级结构中最基本的节点信息，包括节点对应的三角形面、它所在的层号以及它所交叠的下一层的三角形；DAGStruct 描述整个层级结构的信息，包括层级结构的根节点和所有的节点。如 Figure.10 所示：

```
typedef struct {
      face* tri; //from DCEL
      int level_id;
      DAGNodeList OverLapNodesList;
}DAGNode;    / DAG Node
```

```
typedef struct {
      DAGNode* top;
      DAGNodeList AllNodesList;
}DAGStruct; // whole DAG Struct
```

Figure.10 DAG 定义单个节点及层级结构

6 实验中遇到的问题及解决方法

实验中遇到了对点查询退化情况的处理，即查询点在多边形划分的点或边上。我们解决的办法是先查找出退化定位点所属于的多边形划分，这里给出的是首先查找到的一个多边形划分，其次判断定位点是否在这个多边形的顶点或边上，如果在顶点或边上给出其所在的顶点或着边，否则给出其所在的多边形划分。这种做法，对非退化的定位点做了冗余的查询，但是考虑到只对一个多边形划分进行上述的冗余查询，增加的复杂度为常数，对整体的查询复杂度没有影响。

7 总结和进一步的工作

我们实现了基于 Kirkpatrick 结构的点定位算法，并能进行相关的二维和三维显示，便于查看和理解 Kirkpatrick 结构和查询过程。创建结构和查询过程中涉及到的面和边都用特殊的颜色显示，非常直观的演示了创建结构和查询的过程。

以下一些方面还需要进一步的改进：

程序输入图结构模块用户界面不太友好。目前为了便于构造 DCEL 结构，对输入图结构定义了两条规则：一，输入多边形时，各个顶点必须按照逆时针顺序输入；二，在已有多边形的基础上加入新的多边形，新多边形的起始点和第二个点必须在已有多边形的边上（或顶点）。进一步的工作应当考虑提供更加友好的用户输入界面。

通过 OpenGL 三维显示层级结构的时候，不同层三角形划分之间的交叠关系用等粗的线连接进行表示，由于交叠关系比较多，这样容易造成混淆，不便于用户快速查看。下一步应当考虑用由粗变细的圆柱表示，类似树干到树叶的过程，这样可以看出不同层之间的差别。同时在查询的时候也可以考虑用特殊的颜色标记查询经过的路径，使得查询的过程更加直观。

本实验中的三角化方法采用的是 Graham Scan 方法，在 3.3 节中分析它的复杂度不会影响整体预处理的复杂度，但是未将它与其他三角化方法的结果进行比较，进一步可以考虑采用其他的三角化方法，并对不同的实验结果进行比较。

8 参考文献

[1] D. Kirkpatrick. **Optimal search in planar subdivisions**. SIAM Journal on Computing, 12(1), pp. 28 - 35, 1983.

[2] John Iacono. **A 3-D visualization of kirkpatrick's planar point location algorithm**. Proceedings of ASCG03, pp.377 - 377, 2003.

[3] Junhui.Deng. **Computational Geometry Course 2003, Lecture Notes**. Chapter 4, Chapter17.