梯形图

1. 问题背景

平面点定位是计算几何中的一类经典问题，其研究的目标是通过对平面子区域的预处理，使我们能够快速获得平面上特定点所处的位置。在许多不同的场合中，都会遇到类似点定位查询的问题。例如，我们有一张电子格式的航海地图，希望利用计算机来帮助确定方位。这样，无论何时何地，计算机都能够动态地显示出现在所处位置的水流情况。在这种情况下，我们可能会拥有一套相当详细的专题图，我们希望能够频繁地进行点定位查询，从而在航行的过程中，不断地更新显示当前信息。这就意味着，我们希望对航海图进行与处理，然后将有关信息组织为某种数据结构，从而使得点定位查询可以很快完成。

点定位问题可以出现在不同的领域。假设我们想要实现一个通过屏幕显示地图的交互式地理信息系统。用户只要用鼠标点击某个国家，就可以查阅该国的信息。随着鼠标在屏幕的不同位置之间移动，该系统应该能够始终提供鼠标所在国家的名称。显然，对于屏幕上所显示的那张地图而言，这就是一个点定位问题；在这里，鼠标的位置就相当于查询点。这类查询进行的频率很高，毕竟，我们希望能够实时地更新屏幕上的信息。因此，这种查询必须很快得到回答。这样，我们就再次需要借助某种数据结构，来支持这种快速的点定位查询[1]。

为了解决该问题，人们从子区域的每个顶点出发引入一条垂线，引入的垂线、原子区域的边以及设定的矩形区域外边共同组成了该区域的梯形图（如图 1 所示），梯形图组成的任意子区域均为原区域的一个细分。通过对梯形图的数据结构存储（如图 2 所示），实现对平面点的快速精确定位。在本次试验中，我们将实现梯形图的构造，通过查询结构实现点的定位，并对实现的过程进行实验分析。

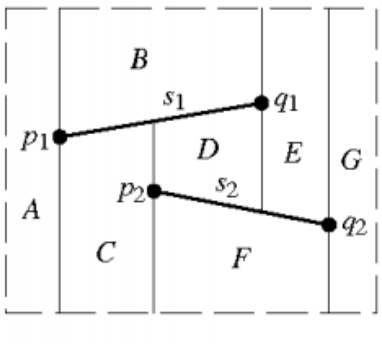

图 1　梯形图

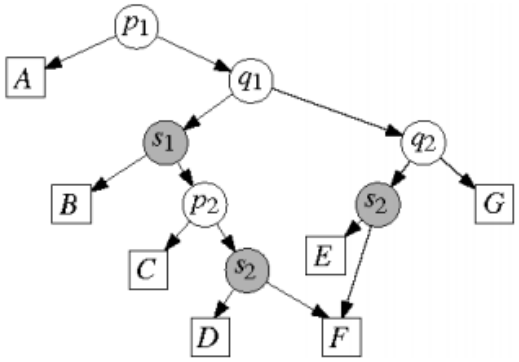

图 2 梯形图查找结构

2. 系统设计

2.1 开发环境

Windows XP + Visual studio 2005。

2.2 编译与链接

工程中使用了 MFC+OpenGL 进行开发。

2.3 总体框架

系统主要分为三大模块：

① 系统框架模块

这部分是整个程序框架，负责数据的输入及输出，与其余两大模块进行连接发，完成所有控件操作。

② 核心算法模块

这部分主要实现核心算法，对输入线段集构建梯形图和有向无环图的查找结构，并将结果返回给系统。

③ 图形绘制模块

这部分主要完成图形界面的绘制，实现对算法结果的图形演示。

3. 数据结构

3.1 基本数据结构

```
//顶点信息
typedef struct point
{
    int x;
    int y;
}Point;

//边集信息
typedef struct edge
{
    Point start;
    Point end;
    int edgenum;
}Edge;
```

3.2 DCEL 数据结构

```
//顶点信息
class Vertex
{
public:
    //坐标信息
    double x;
    double y;

    //该边指向从当前顶点出发的任意一条半边
    HalfEdge* pEdge;

    //构造器和析构器
    Vertex(double newX, double newY);
    Vertex(double newX, double newY, void * newPoint);
    Vertex(void);
    ~Vertex(void);

    //比较操作符，便于排序，
    //原则：先比较横坐标，再比较纵坐标，坐标小者返回true
    bool operator<(Vertex& v);
    bool operator<(Vertex* pV);

    //计算到另一点的距离(平方)
    double distance(Vertex& v);
    double distanceSquare(Vertex& v);

    void * point;
};


//半边信息
class HalfEdge
{
public:
    //半边起点在vector里的编号
    //Vertex* ori;
    int oriNum;
    //孪生边
    HalfEdge* twin;
    //沿着顺时针方向遇到的第一条同起点半边
    HalfEdge* pre;

    //沿着逆时针方向遇到的第一条同起点半边
    HalfEdge* next;
private:
    HalfEdge(void);
public:
    //构造一个半边而不构造其对应的孪生边，其pre和next都被初始化为NULL
    HalfEdge(int pOriNum,HalfEdge* pre,HalfEdge* next,vector<Vertex>& vertexSet);

    //在构造了当前半边的同时也构造了其孪生边,当前半边及其孪生边的pre和next都被初始化为NULL
    HalfEdge(int pOriNum,int pDestNum,vector<Vertex>& vertexSet);

    //析构器
    ~HalfEdge(void);

    bool operator!=(const HalfEdge& edge);
};
```

3.3 梯形图数据结构

```
typedef struct Trap_Face
{
    int facenum;  //梯形编号
    edge* Edge_top; //梯形上边
    edge* Edge_bottom;//梯形下边
    D_Point*  leftp;//梯形左点
    D_Point*  rightp;//梯形右点
    Trap_Face* upleft;//左上梯形
    Trap_Face* lowleft;//左下梯形
    Trap_Face* upright;//右上梯形
    Trap_Face* Trap_Face *Trap_Face::upright
    D_Edge *HEdge;//梯形的上DCEL半边
    D_Edge *LEdge;//梯形的下DCEL半边
    SearchStruct* SNode;//对应的查找结点
    bool used;
}T_Face;
```

3.4 查找结点数据结构

```
typedef struct SearchStruct
{
 int mode;//0:结点 1:线段 2:叶子
 SearchStruct *left;//左孩子
 SearchStruct *right;//右孩子
 D_Point* point;//点结点信息
 Edge* edge;//边结点信息
 T_Face* face;//面结点信息
 SearchStruct* prev;//父亲结点
 bool leftchild;//左孩子标志
 P_L *parentlink;//面结点父亲结点链表
 int level;//树层
  }S_Struct;

 .
//面结点父结点链表
typedef struct ParentLink
{
SearchStruct *value;//父结点值
bool left;//父结点的左孩子
ParentLink * next;//下一父结点
}P_L;
```

4. 算法介绍

本实验采用了随机增量式算法实现梯形图。该算法逐一加入各条线段，每增加一条线段都相应的更新梯形图，并构造出所需的数据查找结构。同时，为了克服算法中因线段引入次序导致的算法性能不均，在线段次序选择上采用了随机化方法。

4.1 随机增量式算法

对任意一组线段 S，通过逐一引入各条线段，为任意一组线段 S,构造出对应的梯形图 T(S)，同时构造出点定位所需的数据查找结构 D。通过随机增量式算法构造出的数据查找结构 D，为一有向无环图（directed acyclic graph-DAG），其中有唯一的根结点，同时对应于线段集 S 的梯形图中的每个梯形，有且仅有一片叶子。每个内部节点的出度都是 2。所有内部节点分为两类：x 节点和 y 节点。每个 x 节点都被标记为 S 中某条线段的一个端点；而每个 y 节点都被标记为某条线段。

在对点 q 进行查询时，要从根节点出发，沿着某条有向路径到达某匹叶子。最终到达的那匹叶子，就对应于 T(S)中包含 q 的那个梯形。

算法描述：

输入：一组 n 条互不相交的线段

输出：梯形图 T(S)，以及与之对应的、限制于一个包围框之内的查找结构 D

1、构造一个包围框 R，大小足以容纳全部线段 S。初始化梯形图结构 T 为包围框 R 本身，构造对应的查找结构 D 为对应于 R 的一片叶子。

2、将 S 中的所有线段随机打乱，得到一个随机序列:s_1，s_2,…，s_n。

3、for i=1 to n

4、 do 在当前的梯形图 T 中，找到与 s_i 相交的所有梯形，$\triangle_0$，$\triangle_1$，…$\triangle_k$（算法 4.2）

5、 将$\triangle_0$，$\triangle_1$，…$\triangle_k$从 T 中删除，并替换为由于 s_i的引入而新生出的若干梯形（算法 4.3）

6、 将与$\triangle_0$，$\triangle_1$，…$\triangle_k$对应的叶子从 D 中删除，对应于每个新生成的梯形，生成一片新的叶子，将新生叶子与已有的内部节点相联接。

算法流程如图 3 所示

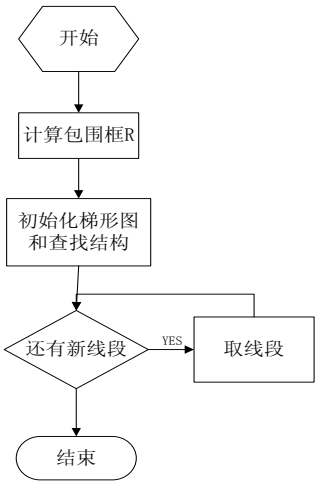

图 3 随机增量算法流程图

4.2 梯形查找算法

输入： 梯形图 T，与 T 相对应的查找结构 D，以及新引入的一条线段 s_i

输出： 由所有与 s_i 真相交的梯形（自左向右）组成的一个序列：$\triangle_0$，…$\triangle_k$

1、 分别令 p 和 q 为 s_i 的左右端点

2、 在查找结构 D 中对 p 进行查找，最终找到梯形$\triangle_0$

3、 j←0

4、 ***while***（q 位于 rightp($\triangle_j$)的右侧

5、 ***do if*** （rightp($\triangle_j$)位于 s_i 的上方

6、 ***then*** 令$\triangle_{j+1}$为$\triangle_j$的右下方邻居

7、 ***else*** 令$\triangle_{j+1}$为$\triangle_j$的右上方邻居

8、 j←j+1

9、 ***return***($\triangle_0$，$\triangle_1$，…$\triangle_j$)

10、

算法流程如图 4 所示：

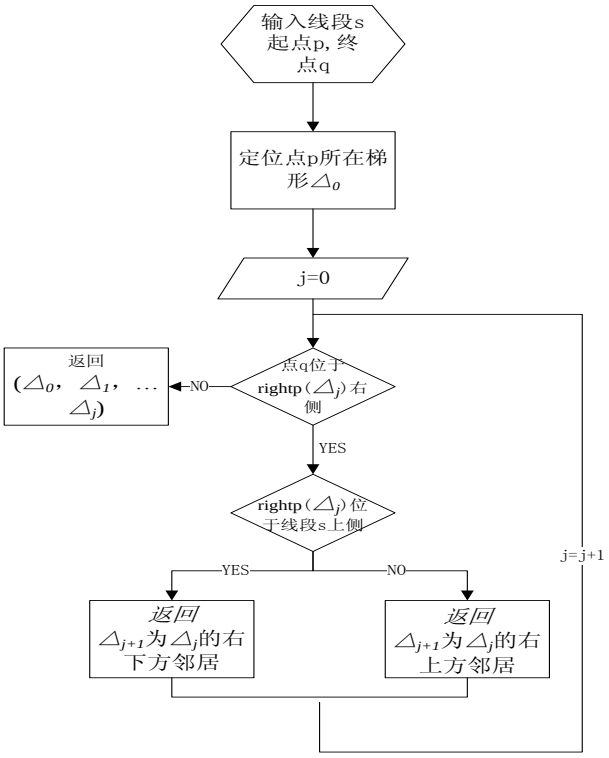

图 4 梯形查找算法流程图

4.3 新生梯形算法

输入：新引入的一条线段 s_i，以及所有与 s_i 真相交的梯形（自左向右）组成的序列：$\triangle_0$，…$\triangle_k$

输出：新生的梯形序列$\triangle_0$，…$\triangle_j$

1、***if*** s_i 完全落入某个梯形$\triangle_0$中，

2、 ***if*** s_i 的端点不是$\triangle_0$ 的端点，

3、 从 s_i 的端点分别发垂直延长线，将$\triangle_0$ 分为 2-4 个新生梯形

4、***if*** s_i 穿越多个梯形

5、 *if* *si* 的端点不是Δ_0与Δ_j的端点，从 s_i 的两个端点分别发垂直延长线，将 Δ_0 和Δ_j一分为三。

6、考查每个梯形Δ_j的右端点

7、 ***if*** 右端点在 s_i 之上，

8、 ***if*** 上方无临时梯形的，上方产生新梯形

9、 ***else*** 补充上方临时梯形的右端点，构成梯形

10、 ***if*** 下方无临时梯形的，生成一临时梯形（梯形右点待定）

11、 ***else***

12、 ***if*** 下方无临时梯形，下方产生新梯形

13、 ***else*** 补充下方临时梯开的右点，构成梯形

14、 ***if*** 上方无临时梯形，生成一临时梯形(梯形右点待定)

线段穿越梯形的情形如图 5~图 7 所示

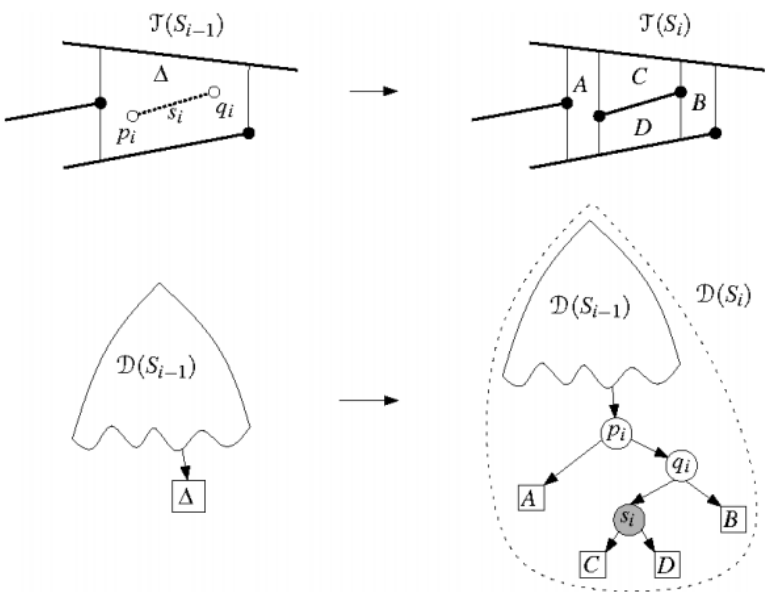

图 5　线段落入唯一梯形

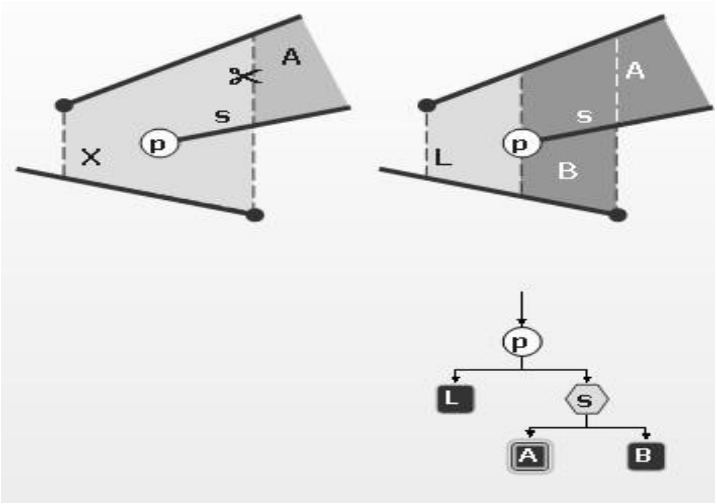

图 6 仅一个端点落入梯形中

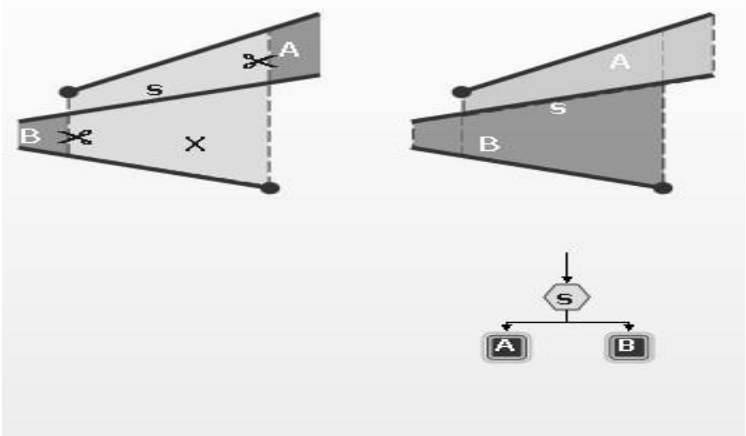

图 7 线段穿越梯形

4.4 不相交线段的随机生成算法

在屏幕上随机生成若干条不相交的线段是一件比较困难的事情。在本实验中，我们借助三角剖分完成这件事情。其基本思想是，借助三角剖分中线段的不相交性，在 O(nlogn)的时间内获得所需数量的不相交线段。

我们知道，对于任意 n 的点的三角剖分，其生成的边的数量一定是大于或等于 n 的。这样，从理论上来说，如果需要随机生成 n 条不相交的线段，我们就可以首先随机生成 n 个点，然后求得这 n 个点的三角剖分，之后在这个三角剖分中随机选取 n 条边。

但是，这样做的效果并不理想，因为这样可能生成许多共端点的边。在实际应用中，我们在 5n 的点的三角剖分中随机选取 n 条边，并以一定的概率抑制共同点的边的出现。通过试验，取得了比较好的效果。下图是随机生成 20 条边和 1000 条边的效果图：

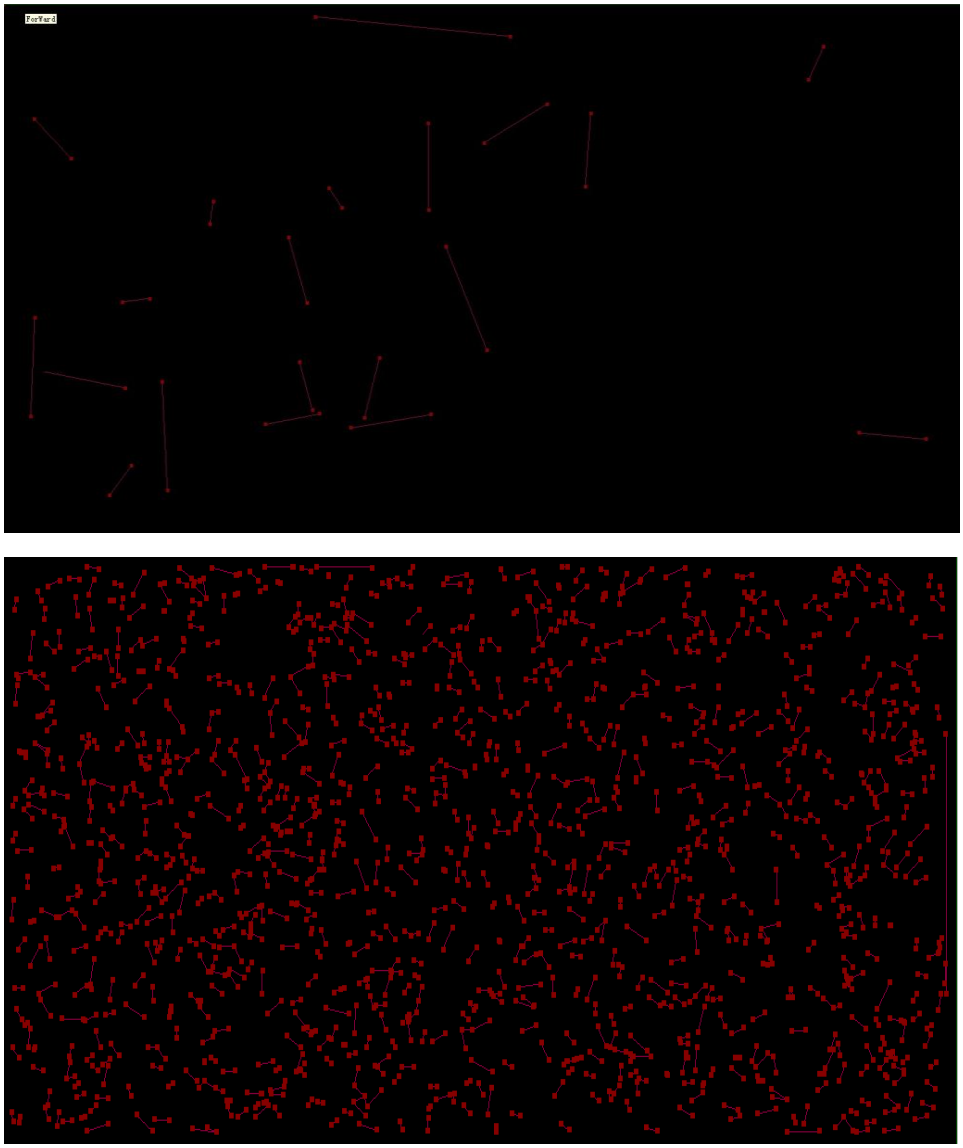

图 8 不相交线段效果图

5 退化情况处理 - 输入线段为竖直垂线

我们采用的方法是，将垂线做为一般性线段处理。由于垂线仅穿越一个梯形，处理方法可与仅落入一个梯形的一般性线段保持一致，即垂线仍然将梯形分解为四个梯形，其中左梯形的参数为（e1,e2,p1,p3）、右梯形的参数为（e1,e2,p4,p2）、上梯形的参数为(e1,e3,p3,p4)、下梯形的参数为(e3,e2,p3,p4)。

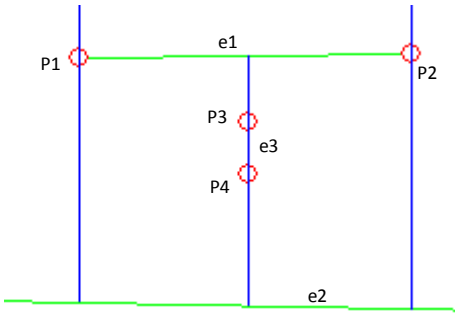

图 8 线段为垂线情形

6 项目遇到的问题及解决方案

1. 生成随机线段

随机生成不相交线段是比较困难的事情。项目之初我们试图使用 Brute-Force 的办法，每次随机生成一条线段，然后与当前全部线段进行相交性检验，若生成的线段与所有线段均不相交，则将其保留下来，之后进行下一条线段的生成。经过实验，这种方法效果并不理想，尤其在线段数目较多的时候会非常慢。

经过几天的思考与尝试，我们想到了利用三角剖分生成随机线段的方法。这种方法可以既可以保证线段之间彼此不相交，又可以保证较低的时间复杂度(O(nlogn))。通过实际工程的检验，这种方法取得了很好的效果。

2. OpenGL 与 MFC 的整合

虽然经过前两个项目的锻炼，小组成员已经基本熟悉了 OpenGL 与 MFC 的使用，但是，由于本身均非图形学方向，基础比较弱，因此在尝试做出漂亮的界面和复杂的动画效果方面遇到了较大困难，花费了大量时间。本来算法已经写好，单纯使用 MFC 验证也没有错误，但是加入了 OpenGL 之后同样的场景就产生了各种各样的 BUG，直至最后也没有全部解决，这是我们非常遗憾的事情。

3. 系统崩溃

在项目截止的最后一天，我们在调试的过程中不小心覆盖了代码，导致已经临近完成的代码受到重大损失，不得不匆忙重写大量代码。时间所限，为了及时提交项目，在提交时还有若干 BUG 没有解决，非常遗憾。

这次教训使我们深刻意识到程序备份的重要性。在以后的工作中，我们一定会做好程序的备份工作，保证类似事件不再发生。

7 总结

本次项目的提交，标志着为期一个学期的计算几何课程已经结束。

在这一个学期的时间里，我们本着对计算几何的兴趣，尽自己的最大努力进行学习和项目的开发。在这期间，曾经遇到许多许多的困难，也曾因为方向不同被自己的老板训斥。但是，我们义无反顾，坚持到了最后。在课堂上，邓俊辉老师交给了我们许多以前不知道也不会想到的知识，极大地扩展了我们的视野；在项目中，我们学会了开发图形界面的基本方法；在互评中，我们对自己的不足有了更清醒的认识，同时从其他同学那里学到了许多宝贵的知识和经验。

虽然这个过程是艰辛的，但收获是巨大的。在这一个学期的时间里，我们极大地提高了自己的能力，更重要的是，我们从邓俊辉老师与其他同学那里获得了一笔宝贵的精神财富，我们相信这会使我们受益终生。

对于每一个项目，尽管做了最大努力，但是，我们仍有许多不尽人意的地方，留下了许多遗憾。虽然课程已经结束，但我们对计算几何的兴趣不会消减，对自己的不足也不会忘却。在今后的学习与工作中，我们会努力将课程所学与本专业结合，将知识转化为解决实际问题的钥匙，同时，弥补自己在项目中的不足，争取取得更大的进步。

参考文献

[1] Mark de Berg，Otfried Cheong，Marc van Kreveld，Mark Overmars 著，邓俊辉译，计算几何。北京：清华大学出版社，2009.8。