组员：戴斌华，陈瑞明

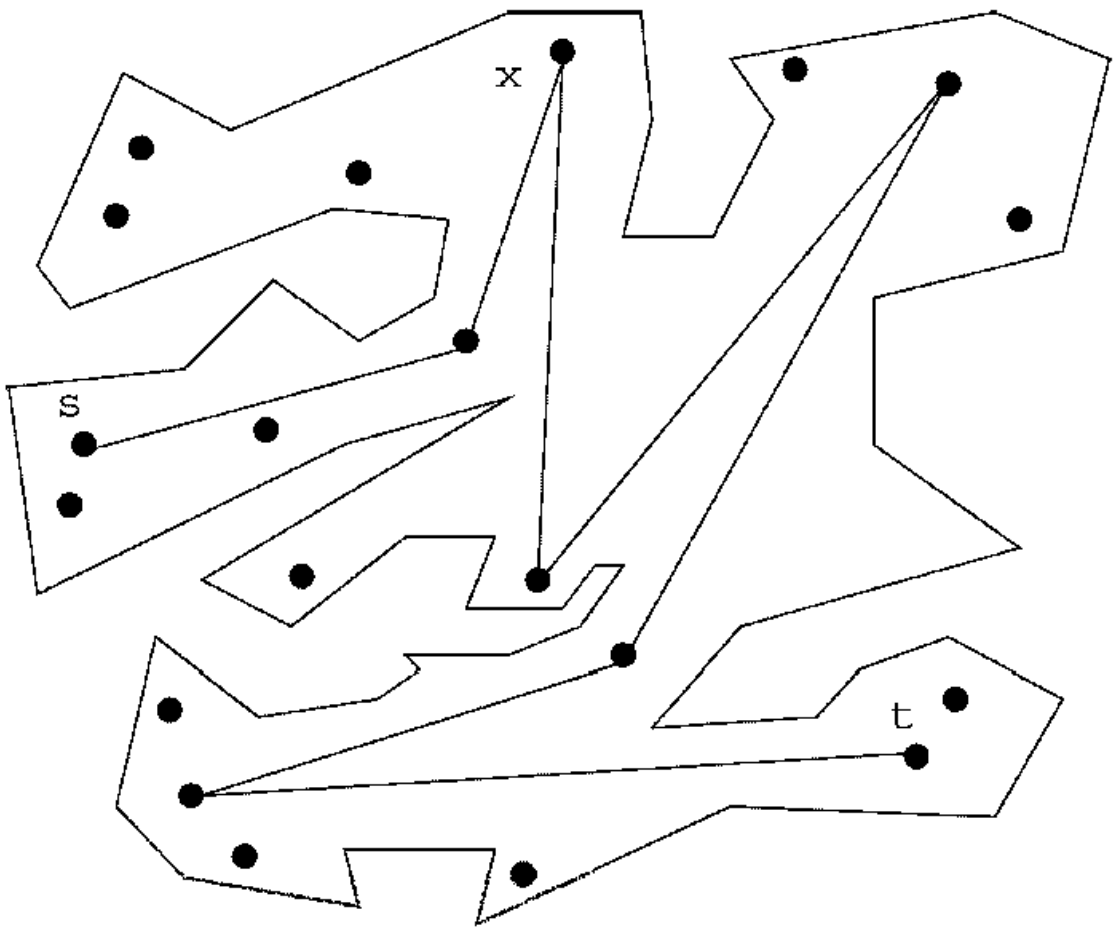
计算几何大作业实验报告

—在障碍物中寻找简单路径

● 论文研究动机

在障碍物中寻找一条简单路径的研究是由生成 polygon 的问题引出的。假设在平面上给出一个点集 X，我们想生成所有以这些点为顶点的简单的 polygon。一个简单直接的办法就是从点集 X 中任意选取一个点 x, 然后不停的继续从点集 X 选取点来延伸这条路径 L。假使当前的路径 L 的尾端的端点为 y,并且让 Y 代表所有不再路径上的点集 X 的点。我们就可以把路径延伸到所有满足下面两个条件的点。一，该点属于并集 Y∪{x}；二，线段 yz 不与 L 相交。在一些情况下，当某点 y 被选为路径上的点以后，以后无论怎么选取点都不能使新加入的线段不与 L 相交。这样的选取对构造简单的 polygon 是不可行的。为了避免这种困难和减少回朔，在选取点 y 的任一步，假定当前的路径 L 已是可行的（可以延伸至构成一简单的 polygon）,我们要验证那一个点被选取后路径仍是可行的。我们可以把这个问题看作是在找一条顶点已给出的避免障碍物的简单路径。

我们的任务就是：给定一个有限的点集 X,其中两个特定的点,源点 s 和目的点 t.。一些由线段组成的障碍物集Φ。如果存在一条简单路径能够避免所有障碍物集Φ中的障碍，我们希望找到它。否则报告不存在



这样一条路径。

图 1：一个简单路径

和这个问题相关的一个问题是：计算任意一条路径避免障碍物集Φ，这条路径不要求一定是简单的，也就是说，路径本身可以相交。这个问题可以用 $O(n^2)$ 的复杂度可用视图来解决。要求路径是简单的问题要难一些。如上图所示的是一个简单多边形的例子。注意，如果我们去掉点 x,将不会有任何简单的存在。但是仍然存在一条自身相交的路径。

如果障碍物集是由任意的线段组成的，我们可以证明是严格完全 NP hard。如果障碍物集是一个简单 polygon,那么我们可以给出一个多项式时间的算法，在这个简单多边形内找出简单路径。该算法的复杂度是 $O(m^2n^2)$,其中 n 是点集 X 的元素个数，m 是障碍物集 P 的定点数。

● 主要算法

1. 主要概念和引理

- **切割**

  如下图的线段 pq 就定义了一个切割

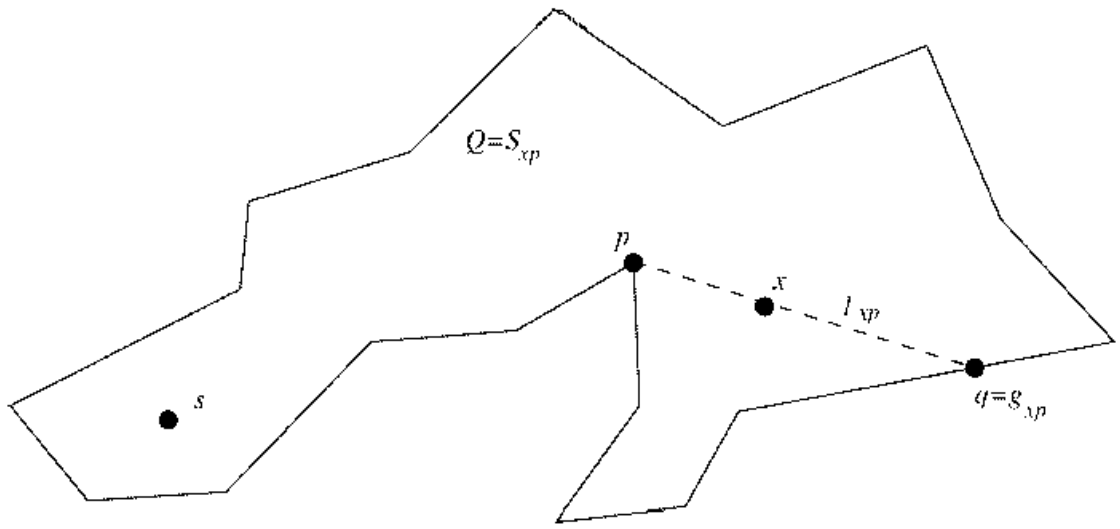


图 2： 切割的图例

- S_{xp}

  一个切割会把简单 polygon 的内部分成两部分，我们把包含点集起点 s 的那一部分定义为 S_{xp}

- **引理 1**

  如果在简单 polygon 内存在一条简单的路径，也存在一条不存在捷径的这样的简单路径。

- **引理 2**

如果终点 t 跟点集的起点 s 不可见，而且一个简单 polygon 的顶点 r 与 s 可见，那么任何不存在捷径的简单路径不和线段（s, g_{sr}）相交。

2. 算法流程

- **起点在边界的情形**

我们把起点 s 看作三个点：s, s', s".把 s 当作是点集 X 中的点，s'，s"当作是多边形边界的顶点。我们引入切割 $l_{ss'}$,该切割和 polygon 边界的交点为 s"。

a) 初始化数据结构。
b) 生成点集 X 的路径点间的可视图。
c) 生成路径点与多边形顶点的可视图。
d) 用 Rayshooting 算法计算出所有切割与多边形边界的交点 g_{xp}。
e) 根据上面产生的可视图生成所有可能的切割的集合，最多可有 mn 个切割。
f) 从 s 到 s"顺时针遍历多边形的边，按点的出现次序给边界交点以及多边形的顶点从 s 到 s"按统一编号.
g) 判断切割间是否有弧，然后生成切割集合的有向图。任意两个切割 l_{xp} 和 l_{yq} 间有弧的标准是：
（1） x 和 y 是可见的
（2） q 和 g_{yq} 的编号在 p 和 g_{xp} 的编号之间，即，$\min(p,g_{xp})<q,g_{yq}<\max(p,g_{xp})$。
h) 用 dijkstra 算法在切割有向图中寻找路径。如果找不到路径，则报告无路径。否则根据切割集合间的路径构造出实际的从起点到终点的路径。构造方法如下：设 K 是切割有向图中找到的路径，我们这样构造实际的从起点 s 到终点 t 的简单路径。如果切割 lxp 在切割路径 K 上，那么 x 在路径 P 上。且如果 lyq 实在路径 K 上紧随 lxp 的点，那么 y 在路径 L 上紧随 x。

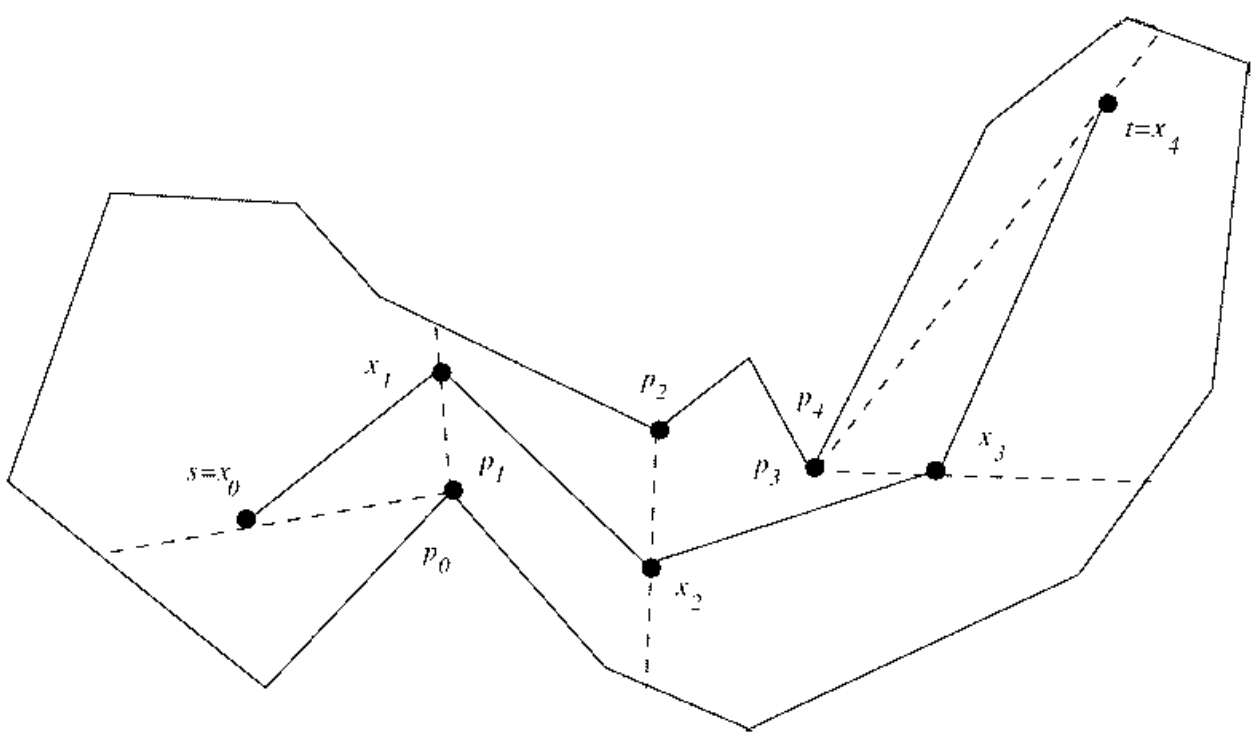


图 3：一个简单路径和与它对应的切割序列

- **起点在多边形内部的情形**

当简单多边形 P 内部的起点 s 的位置是任意时，我们可以构造这样一个简单多边形 P'，它的一个顶点就是 s。假设多边形顶点 r 与起点 s 可见，让[u,v]代表包含 g_{sr} 的多边形的边，引入两个点 u',v'∈[u, v]，使得

(1) 这些点在线段[u, v]上的出现的顺序是 u,u',g_{sr},v',v。

(2) u',v'离 g_{sr} 足够近，使得三角形 u',gsr,v'不包含任何点集 X 中的点。

将简单多边形 P 的边[u,v]用四条边代替[u,u'],[u',s],[s,v'],[v',v]代替就得到构造出的 P'。

3. 算法正确性分析

- 起点在边界的情形

设[x,y]是路径 L 上的一条边，那么 l_{xp}→l_{yq} 是切割有向图上的一个弧。只就意味着 x 和 y 在简单多边形 P 内是可见的。因此，路径 P 和简单多边形的边界不相交。我们还知道 S_{xp} 属于 S_{yq}，暗含(x,y)不和 L 上从起点 s 到 x 的路径相交。因此，路径 L 是简单的。

- 起点在多边形内部的情形

由引理 2，我们知道简单多边形 P 有简单路径 L 的充分必要条件是 P'有这样一条简单路径。

4. 算法复杂度分析

设 n 为点集 X 的元素个数，m 是简单多边形的顶点数。点集 X 的点的可视图可以在 $O((n+m)^2)$的时间内计算出来。我们有最多 n^2m^2 对切割 l_{xp},l_{yq}。对其中的每一对我们都要判断是否存在一条弧。我们以下说明我们可以在 O(1)的复杂度下完成判断。首先，用 Rayshooting 算法计算出每一个切割和多边形边界的交点。由于没能找到论文中提到得 O(log(n))的 Rayshooting 算法，我用我的 O(n)的算法实现了这一功能，但不影响总的算法复杂度。从 s 到 s"顺时针遍历多边形的边，按点的出现次序给边界交点以及多边形的顶点从 s 到 s"按统一编号. 我们可以在 O(1)的时间判断是否有弧：只需检查（1） x 和 y 是可见的，（2）q 和 g_{yq} 的编号在 p 和 g_{xp} 的编号之间，即，$\min(p,g_{xp})<q,g_{yq}<\max(p,g_{xp})$。

综上所述，总的算法复杂度是 $O(m^2n^2)$。

● 实验中遇到的问题

（1） 该算法由于要用到 Rayshooting 算法求出切割与多边形边界的交点，而论文中提到的效率高的 Ray shooting in polygons using geodesic triangulations 算法在图书馆和网上都找不到，只好采用自己写的较为低效的算法代替。程序运行速度有所降低。

（2） 在调试中测试找出的路径是否是 Simple 时，发现错误，找出的路径竟然是自身相交的。仔细检查后发现自己把检查两个切割是否有弧的一个条件理解错了，本来条件要求 q 和 g_{yq} 两点的编号都在 p 和 g_{xp} 的编号之间，即 $\min(p,g_{xp})<q<\max(p,g_{xp})$ 并且 $\min(p,g_{xp})<g_{yq}<\max(p,g_{xp})$。我理解成了 $\min(p,g_{xp})<q$，且 $g_{yq}<\max(p,g_{xp})$。改正后，来类似图一的例子测试

结果正确。

（3） 在调试中还发现程序的内存没有释放干净，对代码进行修改释放了一部分本该释放的内存，但有些内存释放的困难较大，比如很那确定是不会马上又被访问，一些内存还是没有被释放。由于本算法的复杂度是 O(m2n2),导致运行时需要较多的内存，因此，未被释放的内存也会很多。

（4） 在找切割集的有向图是否存在一条路径时，我开始采用的是 floyd 算法。调试中发现程序的绝大多数时间都用在寻找路径上，有时，这一过程要耗时数分钟。后来发现没有必要求出每一对节点间的最短路径，改用求一与源点到其他点最短路径的 dijkstra 算法。程序运行时间大大提高，主要运行时间不是用来找路径，而是用来构造切割集的有向图。这同上面算法复杂度的分析是一致的。

（5） 值得注意的是，虽然程序中我使用的是找最短路径的算法来寻找切割集有向图的路径。但最终求出的路径不一定是最短路径。因为找出的是以切割为节点的有向图的最短路径。这和构造出来的实际的起点 s 到终点 t 的是否为最短路径没有直接关系，因此，这条路径可能存在捷径，但是由引理 1，必然存在已跳无捷径的最短路径。

● 心得体会

这次大实验使我们感觉收获很大。通过编程的实践，不仅将我们课堂上所学的计算几何知识加深了认识，而且也接触到一些在课堂上没有讲到的计算几何知识。比如可视图的概念和 Rayshooting 算法。

我们还认识到要做好一个项目，提前收集好相关资料是必不可少的。如果我们早一些开始收集资料，一定能找到现在还没找到的 Ray shooting in polygons using geodesic triangulations 算法，使程序运行更加高效。

这次大实验还让我运用了很多其他科目的知识，比如数据结构上的 dijkstra 算法和 floyd 算法。并且让我深刻意识到当问题维度很大时稀疏矩阵存贮的好处。

除此以外，我们还深刻认识到将计算几何的理论和实际应用相联系的重要性。一个好的理论必须要有好的应用前景。在研究理论的时候不要和实际脱离开来。