

障碍最短路径算法研究

小组成员

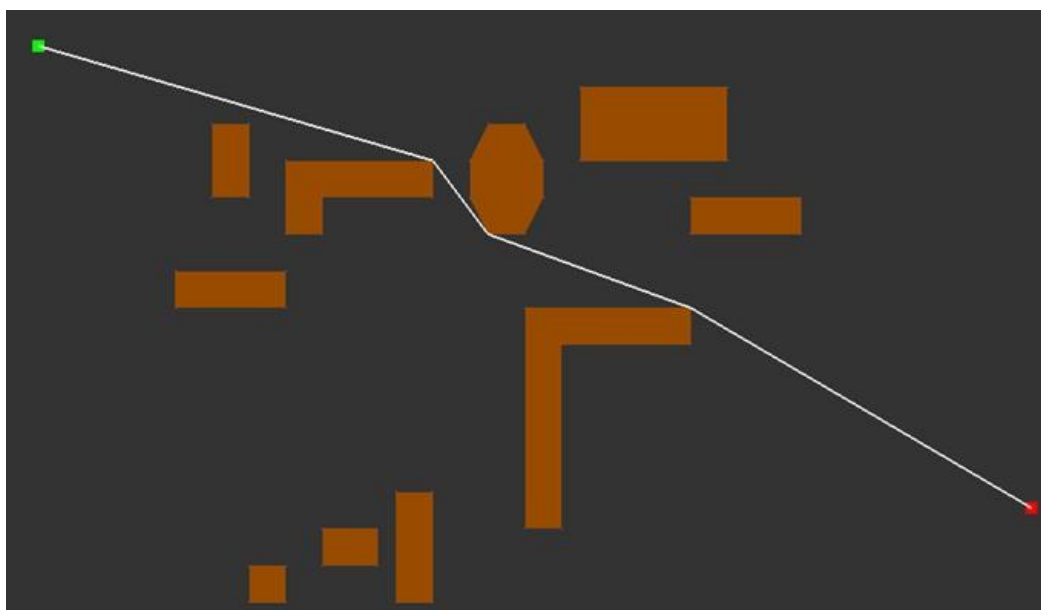
陈 康 (2012311316) 交叉研 2 班

冯立新 (2012210914) 计研 123 班

应用背景

障碍最短路径问题是图论中最短路径问题的一个自然的推广，考虑的是在有障碍情况下的最短路径问题。如工厂中机器人在工作间内如何规划运动的线路、GIS 系统里地图上两点之间的最短距离、游戏设计中自动寻路、网络路由策略选择、电路板中线路设计等诸多问题都可归结为有障碍下的最短路径问题。解决这个问题可以带来非常明显经济效益和社会效益。同时这个问题是计算几何领域的古老而经典问题之一，而且与它相关的子问题，如单起始点多终止点障碍最短路径、最短链接、可见图等问题也受到了持续的关注与研究。

问题定义



给定在平面中指定起始点 $pStart$ 、终止点 $pEnd$ 和 N 个互不相交的多边形障碍物 P_1, P_2, \dots, P_n , 计算起始点 $pStart$ 和终止点 $pEnd$ 之间的最短路径。

相关工作

障碍最短路径问题在计算几何领域曾经得到了广泛的研究，但是由于一直缺乏简单高效的算法，目前这个方向也逐渐冷却。在实际应用中，人们往往更愿意使用简单高效的非精确算法（如 A*等），但我们这里只讨论精确算法。

目前求解障碍最短路径问题有两种基本思路：一种是直接构造全局可见性图，把问题转化成无障碍图的最短路径求解问题；另外一种是通过区域分割，不断排除不可能区域，最终构造出最短路径地图。构造全局可见性图原理比较简单，但由于需要为大量不可能经过的点构造可见性图，所以计算代价比较高。最短路径地图的原理比较复杂，但算法效率显著提高。当前理论上的最优算法就是基于这种思路，利用波浪线的方法对几何区域进行快速分割实现的。

通过构造可见性图求解最短路径问题直观上很容易理解，一旦构造出全局的可见性图，我们就从原问题中抽象出了一幅完全图，每一条边都代表了一个可行的路径，接下来要做的就是使用图论中经典的 Dijkstra 算法求解最短路径。事实上构造可见性图本身就是一个相当基本的问题，包括 Art Gallery Problem 在内的一系列经典问题都依赖于该问题，所以单纯对可见性图构造算法有着大量的研究[3]。

构造可见性图的一种经典的方法是旋转扫描线算法[1]，其复杂度为 $O(n^2 \log n)$ ，其中 n 为障碍物顶点的数目。我们实现的也是这个算法，具体内容我们会在下文中仔细分析。Ghosh 和 Mount 对该算法进行了改进[2]，使得复杂度降至 $O(n \log n + E)$ （ n 是顶点数， E 是可见性图的边数）。它的思路是以平面扫描三角剖分形式给出了有障碍情况下可见图的构建。主要的思想是 Funnel Sequence（FNL）漏斗的分割和合并。漏斗可以看作覆盖所有障碍的多边形边 E 的两个顶点，按照顺时针或者逆时针顺序的所有可见顶点的序列，相连顶点形成的边是最靠近 E 的左或者右的线段。通过平面扫描的方式，新加入的多边形的边对应的新漏斗是从原多边形内链上边的漏斗中分割得到。最后得到的就是要求的可见图。

同样在三维空间中也存在可见性图，但由于 2 维可见性图的概念并不能简单地直接推广到 3 维，维数升高带来的代数难度和组合难度导致三维空间的可见性图复杂度更高。Joseph [6]回顾了 3 维可见图的发展历程，指出了 3 维可见图构造的 NP-hard 性质。

障碍最短路径的另外一种求解思路是直接几何域上进行运算。Mitchell[4]的算法考虑了移动物体的尺寸，将整幅地图划分成大小相等的小区域，计算哪些

小区域是可以通过的，最终将每一个小区域抽象成一个节点，形成一个联通图，在此基础上同样应用图论最短路径算法。算法的主要代价在于决定哪些小区域是可以通过的，该算法的时间复杂度为 $O(kn)$ ， k 为障碍的个数， n 为障碍多边形的边的个数。目前理论上的最优的算法在此基础上由 Hershberger 和 Suri 改进 [5] 提出，时间复杂度 $O(n \log n)$ 。他们采用的是 continuous Dijkstra 方法，也就是形象的波浪线传播方法对最短路径地图进行计算，算法通过对平面采用四叉树划分加速波浪的事件传递，对非近邻的波浪则采取近似处理从而得到最优的构造算法。但与大多数理论最优算法类似，该算法在实现上过于复杂，实用性不高。

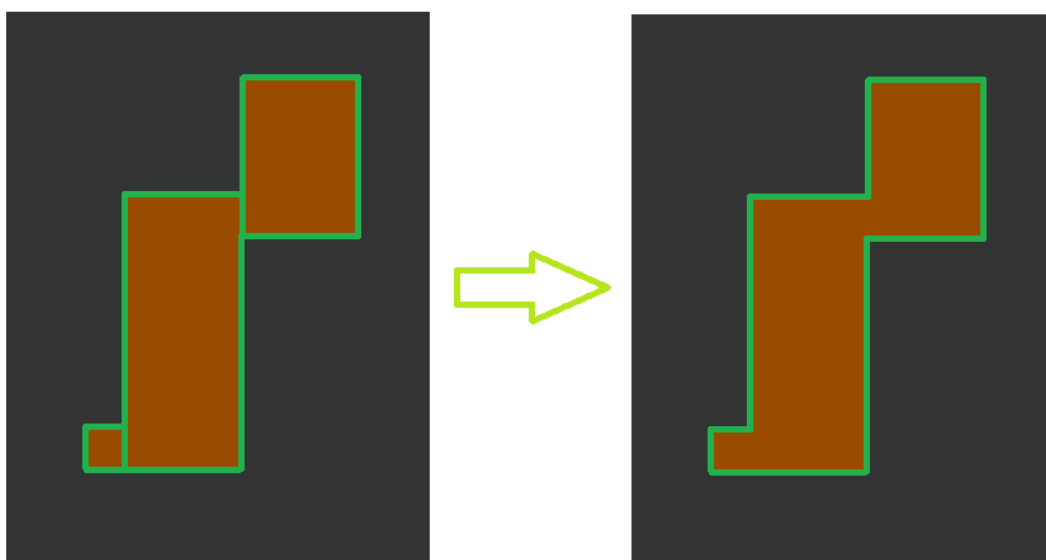
输入限定和衰退处理

计算几何的很多算法之所以停留在理论阶段，除了算法本身比较复杂以外，还有一个重要的原因就是算法的通用性比较差，对各种各样的奇异情况失效。障碍最短路径的算法也是如此，虽然理论上存在 $O(n \log n)$ 的算法，但是在需要精确解时，比较常见的还是 Lee 的旋转扫描线算法。事实上，即使是旋转扫描线算法同样面临着各种各样衰退情况的困扰。在应用中求取障碍最短路径往往是使用一些近似的启发式算法（如 A^* ）。为了算法能够更高效的处理大多数正常情况，我们对输入进行了严格的限定，主要包含以下几点：

- a) 所有障碍物必须为简单多边形
- b) 障碍物之间禁止重合
- c) 源端点和目标端点不能出现在障碍物内部

d) 障碍物边界可以重叠，但只能有顶点重叠，边之间禁止重叠

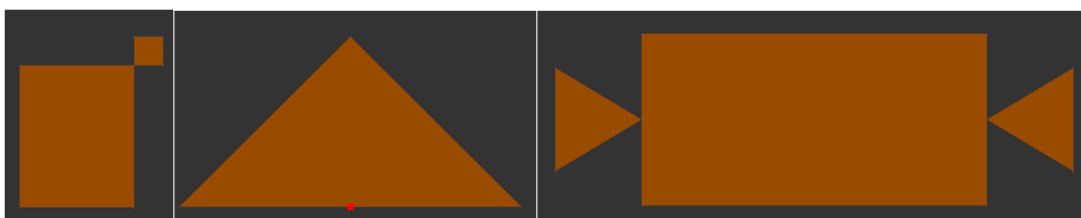
前三个限定直观上是比较可以理解的，这里重点说明下第四个限定。这个限定并不是算法必须的，也可以作为一种特殊情况进行处理，但是为了处理这个情况，我们不得不在一个基础的底层操作上加上很多判断，导致算法整体性能下降。权衡之后，我们决定将这种情况交给用户来处理，处理方法其实比较简单，比如



上图中左边的三个障碍物存在边相交的情况，那么可以直接将这三个障碍物合并成一个。

在这样的限定下，只可能出现三种衰退：

- a) 障碍物交与定点处（下图左）
- b) 源端点或目的端点于障碍物边界相交（下图中）
- c) 一个障碍物的端点与另一个障碍物交与非断点处（下图右）



对于前两种情况,我们在算法中进行了判断。第三种情况我们同样交给用户,如果用户发现存在这种情况,那么我们会在预处理中检测到交点,然后为相关障碍物插入一些顶点(如上面的情况,我们会为中间的矩形障碍物两边各插入一个顶点)。这样处理并不影响障碍物的形状,但是我们就可以在一个统一的框架下处理所有的输入。

算法实现

主体思路

我们实现的是旋转式平面扫描算法,其核心思路就是对一组输入(源端点、目的端点和一堆不相交的障碍物)首先构造可见性图,然后在可见性图的基础上应用 Dijkstra 最短路径算法求解最短路径。

这算法主要基于一个事实:只要源端点和目的端点之间不直接可见,那么最短路径一定沿着障碍物的边缘前进。对这个事实有一个比较直观的解释,假设源端点和目的端点之间有一条路径,我们沿着这条路径放一根拉伸的橡皮筋,橡皮筋一定会收缩向障碍物的边缘,得到一条更短的路径。因此只需确定最短路径经过哪些障碍物的顶点,问题也就可以做下面的抽象:

给定一幅图 G_{vis} , 其定顶点是源端点, 目的端点以及障碍物的所有端点, 顶点 V_a, V_b 如果直接可见, 那么 V_a, V_b 之间就添加一条边, 其权重是 V_a, V_b 的欧式距离。这幅图就是可见性图, 源端点跟目的端点在这幅图上的最短路径就是其在原问题下的最短路径。

Algorithm SHORTESTPATH(S, p_{start}, p_{goal})

Input:

A set S of disjoint polygonal obstacles, and two points p_{start} and p_{goal} in the free space.

Output:

The shortest collision-free path connecting p_{start} and p_{goal} .

1. $G_{vis} \leftarrow \text{VISIBILITYGRAPH}(S \cup \{p_{start}, p_{goal}\})$
2. Assign each arc (v,w) in G_{vis} a weight, which is the Euclidean length of the segment vw .
3. Use Dijkstra's algorithm to compute a shortest path between p_{start} and p_{goal} in G_{vis} .

可见性图

可见性图的构造是这个算法的核心部分，也是算法整体复杂度的主要来源。

构造方式简单的说就是对每一个顶点，确定所有跟其可见的顶点，决定可见性图上的边。暴力方法，在确定两个顶点是否可见时，需要遍历所有障碍物的边，判断是否有遮挡，这样构造一个顶点的可见性图就需要 $O(n^2)$ 的复杂度，整体复杂度将达到 $O(n^3)$ 。但采用了旋转扫描算法后，构造一个顶点的可见性图，复杂性可以降到 $O(n \log n)$ ，整体复杂性为 $O(n^2 \log n)$ 。

Algorithm VISIBILITYGRAPH(S)

Input:

A set S of disjoint polygonal obstacles.

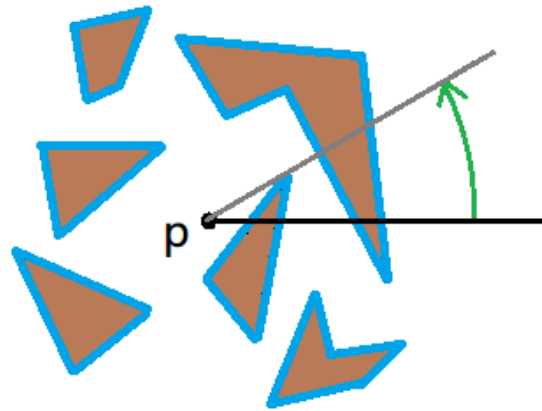
Output:

The visibility graph $G_{vis}(S)$.

1. Initialize a graph $G = (V, E)$ where V is the set of all vertices of the polygons in S and $E = \Phi$.
2. for all vertices $v \in V$
3. do $W \leftarrow \text{VISIBLEVERTICES}(v, S)$
4. For every vertex $w \in W$, add the arc (v, w) to E .
5. return G .

旋转扫描

这是整个算法的关键，其思想与平面的扫描线算法相当类似，都是按一定顺序处理输入数据，扫过的点是已经处理结束的部分，扫描线上的点是当前正在处理的部分，未进入扫描曲线的点是未处理的部分。与常见的



水平扫描线算法不同，这里的扫描线算法是以当前点为中心的 360 度旋转扫描。该算法含义上也比较直观，相当于在原地转了一圈，从而确定周围所有的可见点。算法流程如下：

Algorithm **VISIBLEVERTICES**(p,S)

Input:

A set S of polygonal obstacles and a point p that does not lie in the interior of any obstacle.

Output:

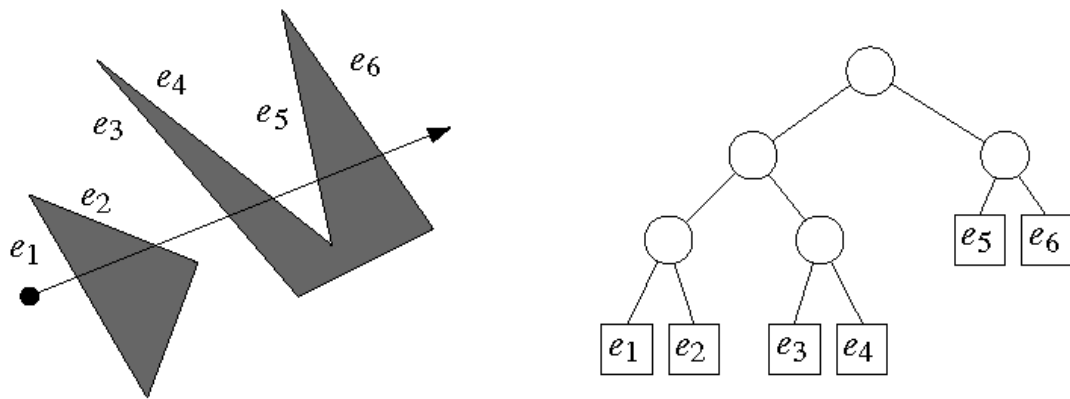
The set of all obstacle vertices visible from p.

1. Sort the obstacle vertices according to the counter-clockwise angle that the halfline from p to each vertex makes with the positive x-axis. In case of ties, vertices closer to p should come before vertices farther from p. Let w_1, \dots, w_n be the sorted list.
2. Let p be the half-line parallel to the positive x-axis starting at p. Find the obstacle edges that are properly intersected by p , and store them in a balanced search tree T in the order in which they are intersected by p .
3. $W \leftarrow \Phi$
4. for $i \leftarrow 1$ to n
5. do if **VISIBLE**(w_i) then Add w_i to W.
6. Insert into T the obstacle edges incident to w_i that lie on the counter-clockwise side of the half-line from p to w_i .
7. Delete from T the obstacle edges incident to w_i that lie on the clockwise side of the half-line from p to w_i .
8. **return** W.

a. 固定需要计算可见性图的点 p 将所有其他点按到 p 点的极角逆时针排序，如果两个点极角相等，则距离 p 近的排在前面。这样就固定了一个扫描顺序，按这样的顺序处理所有顶点就可以充分利用前面处理过的顶点来加速当前处理过程。

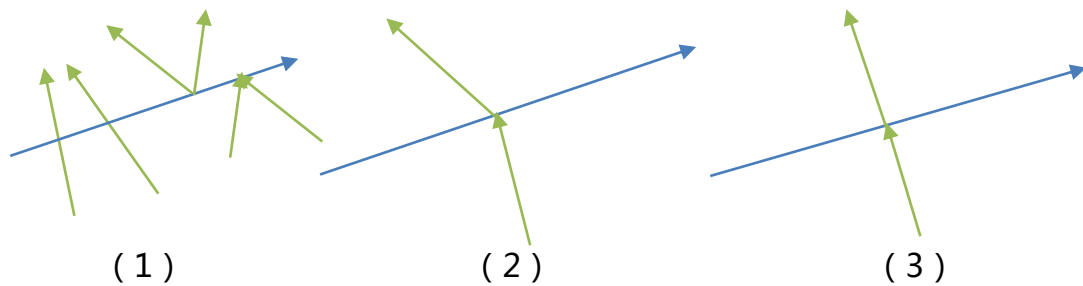
b. 初始化扫描线，从 p 点出发，沿着 X 轴正方向发出一条射线，遍历所有障碍物的边，找出与这条射线相交的所有障碍物的边，插入一颗二叉树 T 中。

c. 按第一步得到的顺序遍历所有顶点，如果可见就将其加入 p 的可见顶点列表中，判断顶点可见性的方法会在下一节阐述。每当遍历完一个顶点就取出与该顶点相关的两条边，分别判断与当前扫描线的关系：如果在扫描线的左边，说明在扫描线下一次移动后，这条边可能与扫描线相交，从而形成遮挡关系，因此将其插入二叉树 T 中；如果在扫描线的右边，那么在扫描线下一次移动后，这条边就会离开扫描线，不可能对后面的点形成遮挡，因此将其从二叉树 T 中删除。



上面的介绍中多次提到了一棵二叉树 T ，在这棵树中保存了所有可能会与当前扫描线存在遮挡关系的边，按于扫描线相交的先后顺序排列。之所以要建立成一棵树的形式，主要是保证快速的查找，插入和删除。在实现中，我们充分利用了 STL 底层的红黑树，将所有的边存储在了一个 STL 的 set 中，并且用 `ToLeft`

测试自定义了一个比较器，当然这里需要处理衰退情况。



上面三个例子中 (1) 是正常情况，toLeft 可以很明确地确定顺序。(2) 和 (3) 都是衰退情况，(2) 中两个线段都在对方左边，(3) 中两条线共线。但是我们注意到 (2) (3) 中的边都只可能相交在原有障碍物的端点处（前面对输入的限定和衰退预处理保证了这一点），因此这两条边总是一条被插入一条被删除，他们的顺序其实无关紧要。

可见性判断

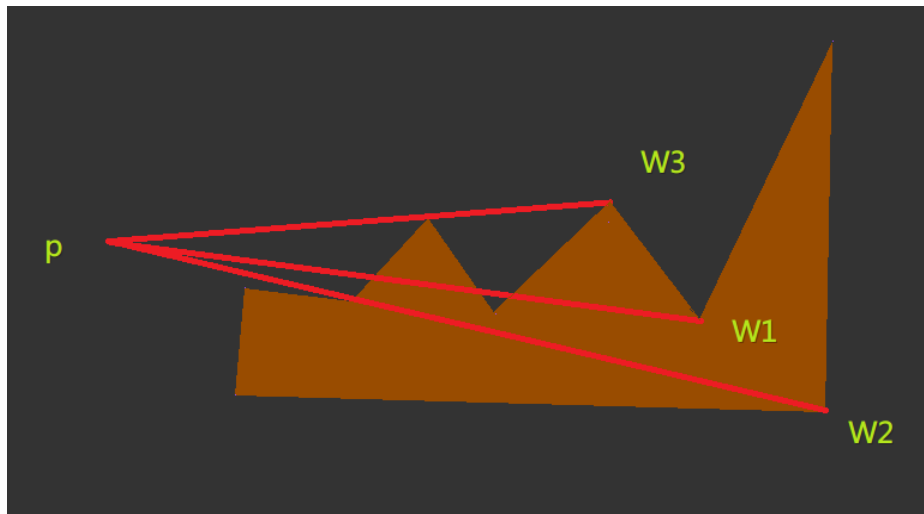
VISIBLE(w_i)

1. **if** pw_i intersects the interior of the obstacle of which w_i is a vertex, locally at w_i
2. **then return false**
3. **else if** $i = 1$ or w_{i-1} is not on the segment pw_i
4. **then** Search in T for the edge e in the leftmost leaf.
5. **if** e exists and pw_i intersects e
6. **then return false**
7. **else return true**
8. **else if** w_{i-1} is not visible
9. **then return false**
10. **else** Search in T for an edge e that intersects $w_{i-1}w_i$.
11. **if** e exists
12. **then return false**
13. **else return true**

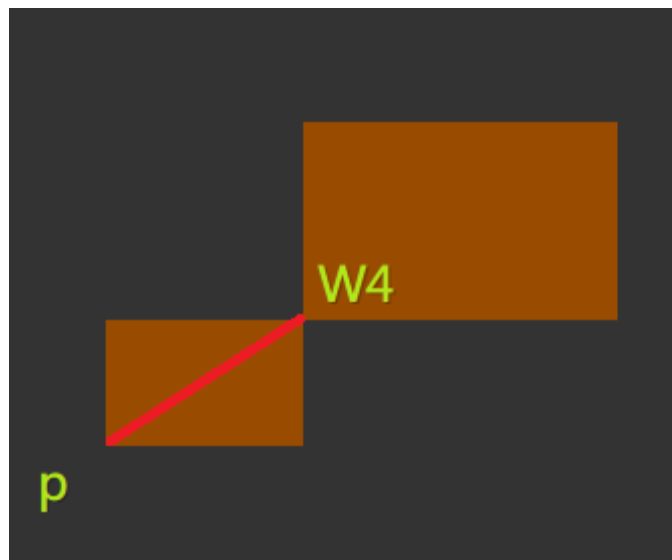
这是整个算法最繁琐的一步，需要对各种情况进行判断。主要分为三个方面：

- a. 首先如果 pw_i 跟 w_i 所在的障碍物交与内部，那么 w_i 一定不可见。为了判

断 pw_i 跟 w_i 所在的障碍物是否交与内部，我们做了如下处理：



- 遍历 w_i 所在的障碍物的所有边，如果发现跟真 pw_i 相交，那么一定交于内部。这排除了上图中 W1 的情况。
- 遍历 w_i 所在的障碍物的所有顶点，如果该点相邻的两个点分布在两 pw_i 侧，且该点到 p 的距离小于 w_i 到 p 的距离，则相交于内部。这解决了上图中 W2 和 W3 的情况。



- 在非衰退情况下，上面的判断已经足够了，但是实际上一旦出现衰退，就会出现新的情况，比如 $pW4$ 虽然不跟 W4 所在的障碍物交与内部，但是 W4 本身可能跟其他障碍物的顶点重合，并且当前处理的 p 的

也是这个障碍物的顶点，这导致 pW_4 穿越其他障碍物的内部，并且不与任何边界真相交。这一点我们在实现初期没有考虑到，后来在测试中发现了这个情况，就补充了一步预处理，我们为所有的顶点对维护了一张表 `commonObstacle`，记录了他们是否同属于某一个障碍物。前两步处理结束后，已经保证了 pw_i 一定不会跨越任何障碍物的边界，因此我们通过 `commonObstacle` 找到 p 和 w_i 共同所属的障碍物（没有则直接跳过），然后判断 pw_i 的中点跟该障碍物的关系，如果 w_i 在内部则同样不可见。

- b. 第二步如果 $i=1$ 或者 p 、 w_i 、 w_{i-1} 不共线，则直接判断 pw_i 与 T 中第一条边的关系，相交则 w_i 不可见，不相交 w_i 就一定可见。
- c. 函数到了这一步，只有一种情况，那就是 w_i 、 w_{i-1} 到 p 的极角相等，且 w_{i-1} 距离 p 更近。此时如果 w_{i-1} 不可见，那 w_i 一定也不可见，如果 w_{i-1} 可见，那就判断 T 中 w_i 、 w_{i-1} 之间的边是否会遮挡 w_i 。

开发环境

开发平台：Windows 7(64 bit) + Visual Studio 2010









第三方库：Qt 4.8，OpenGL, CGAL, BOOST

测试硬件：Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz 3.30GHz,
8GB ram

图形界面



工具栏上我们提供了下面一些快捷按钮：

- ✧ ：新建一幅障碍地图
- ✧ ：打开已有的障碍地图
- ✧ ：保存当前的障碍对图
- ✧ ：绘制新的障碍（鼠标左键拖动增加边，鼠标右键完成绘制）
- ✧ ：移动选中的障碍
- ✧ ：删除选中的障碍
- ✧ ：对起始点进行设置
- ✧ ：对终止点进行设置

另外，鼠标滚轮可以控制视角的远近，鼠标右键可以整体拖动场景，ctrl 键加鼠标左键可以选择障碍物。

菜单栏 Run 下面包含了下面几个栏目：

- ✧ Visibility Graph: 计算并显示可见性图
- ✧ Shortest Path-> with Visibility Graph: 显示最短路
- ✧ Shortest Path-> with Visibility Graph: 显示可见性图和最短路
- ✧ Standardize Input: 规范化输入，主要用于处理衰退情况
- ✧ Run without Check: 不对输入合法性进行检查，直接运行算法

文件格式

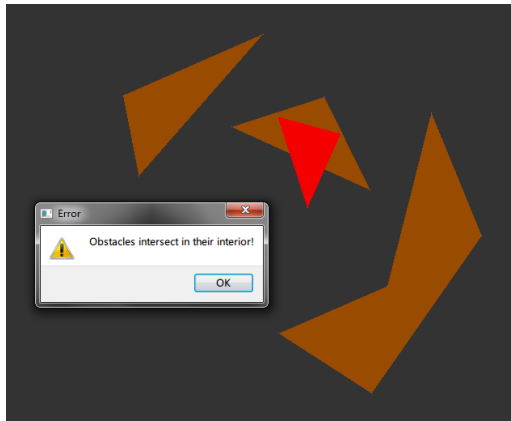
对于障碍物地图，我们自己定义了一个简单的文件格式，既方便了障碍物地图的读取和存储，也方便用户精确控制障碍物的位置，格式如下：

```
#OBSTACLEMAP
2
14 1 -4 6 -4 6 1 4 1 4 0 5 0 5 -3 2 -3 2 3 5 3 5 2 6 2 6 4 1 4
8 -1 3 -1 4 -6 4 -6 -4 -1 -4 -1 -3 -5 -3 -5 3
```

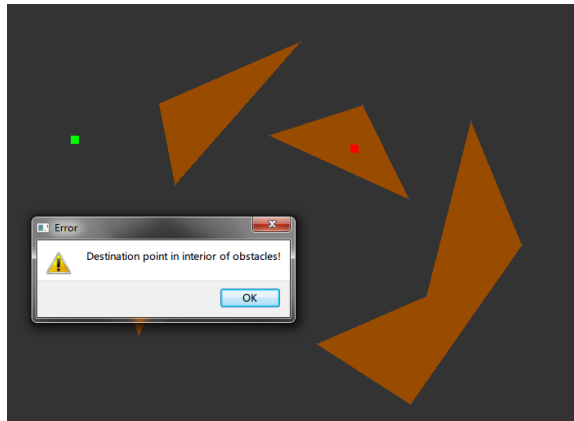
第一行定义了一个标志，用以标志文件格式。第二行是障碍物总数。接着每一行是一个障碍物，每行第一个数表示该障碍物的顶点数量，接着是按顺序存数的定点列表 $X_1, Y_1, X_2, Y_2 \dots$

结果展示

合法性检查

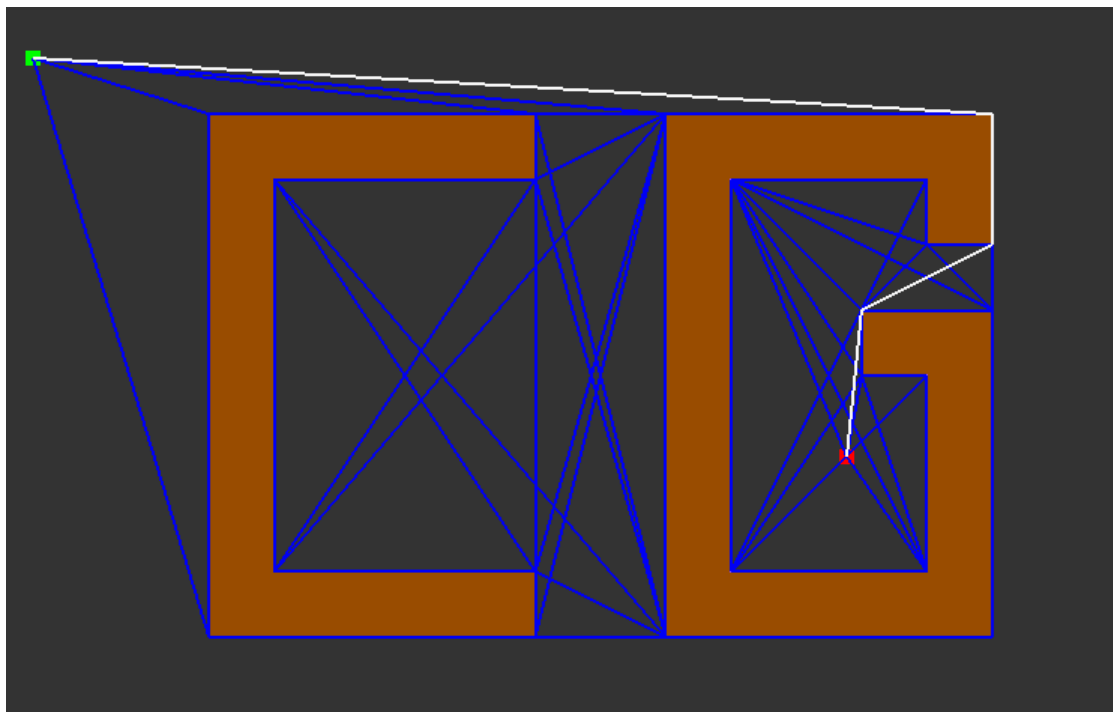


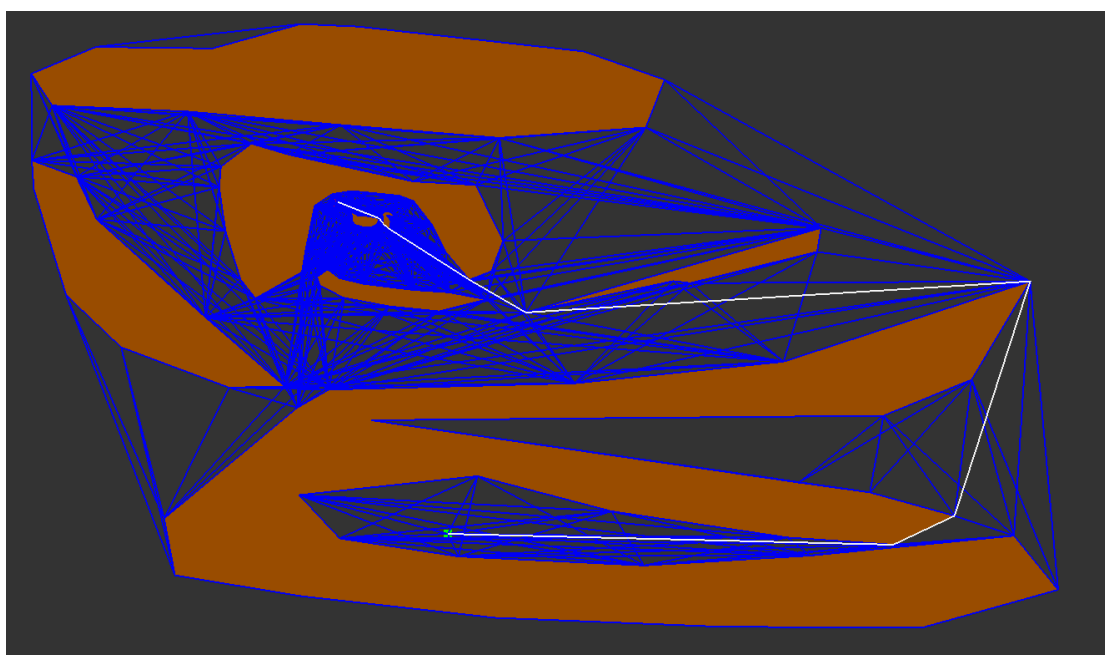
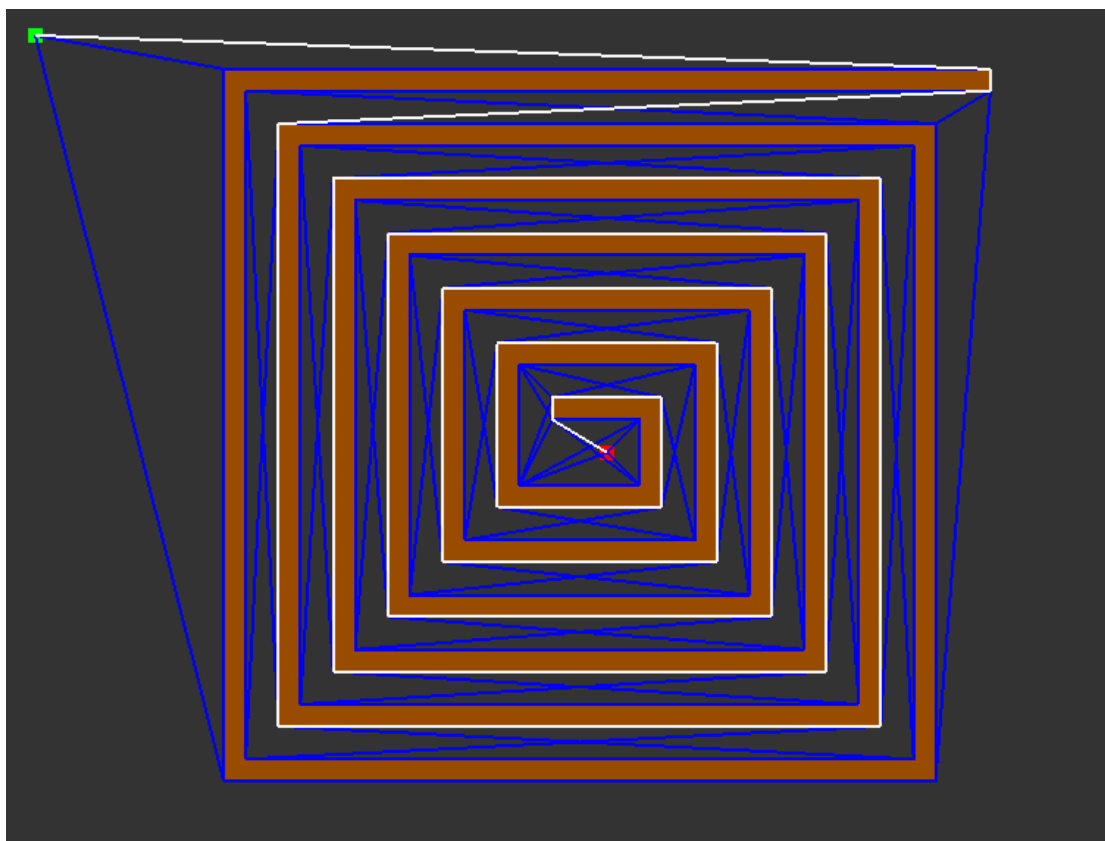
障碍物重叠

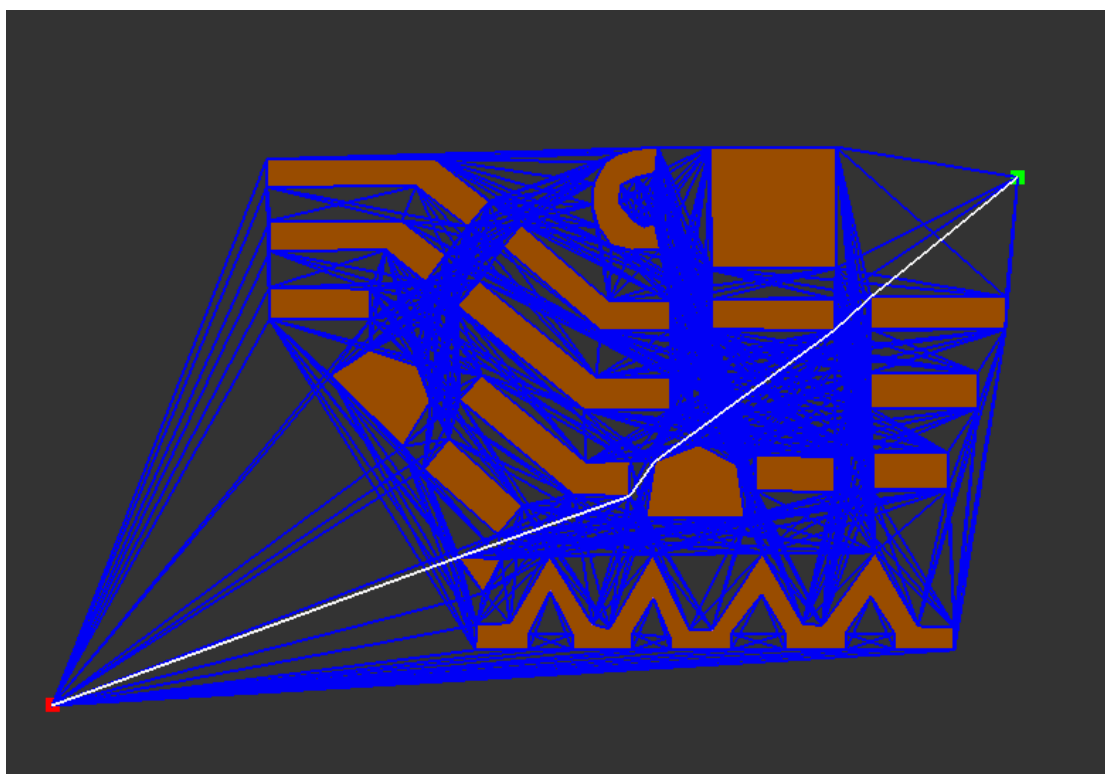


目的端点在障碍物内部

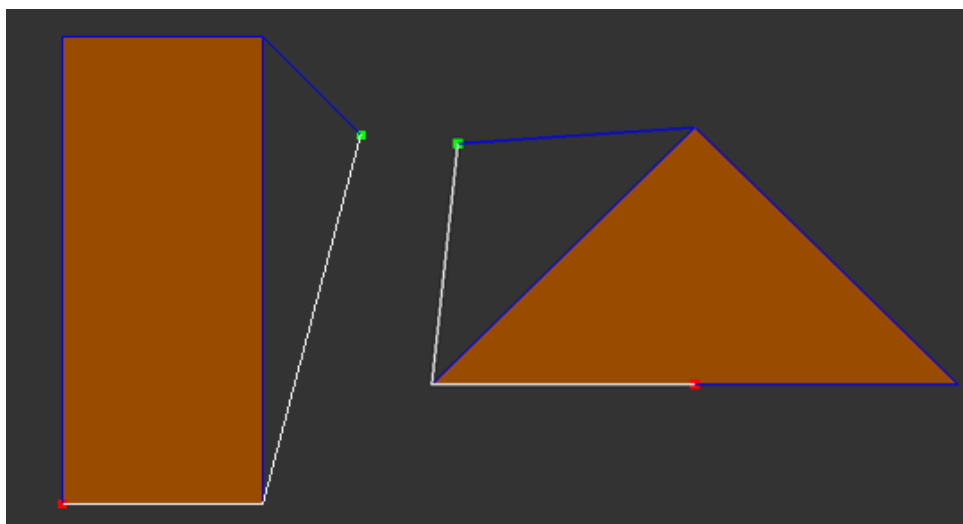
非衰退输入



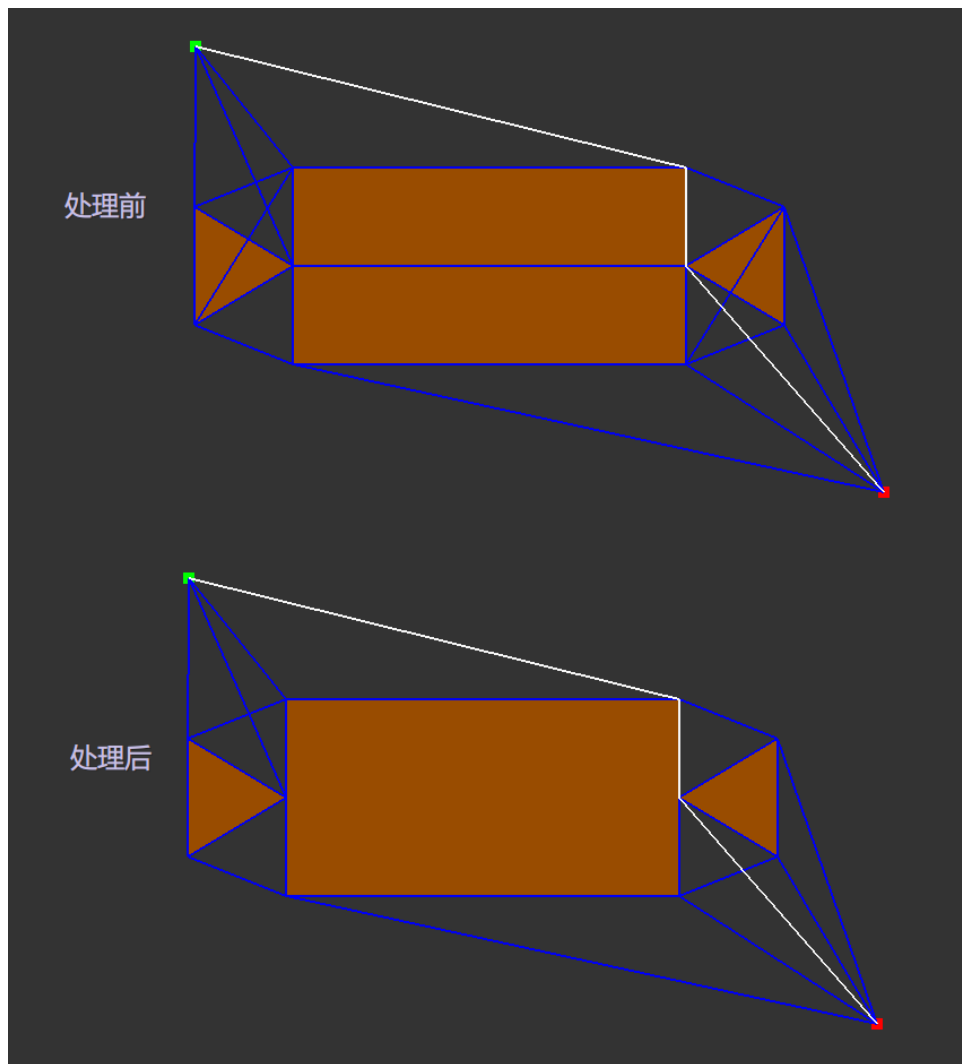




衰退输入



这两种衰退我们在算法中已经做了判断，可以直接处理。

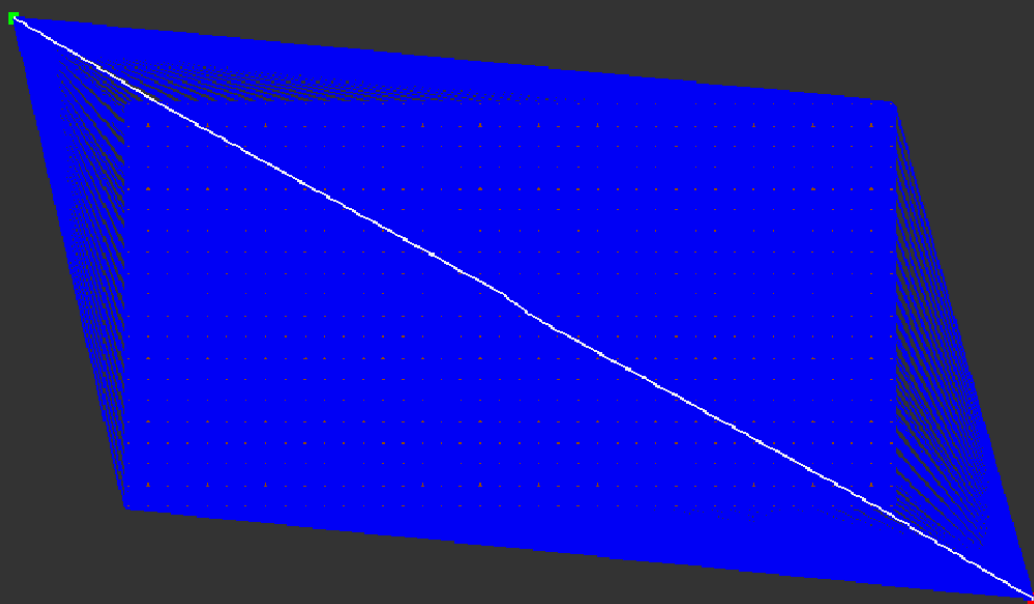


这种衰退算法中默认是不处理的，如果用户发现存在这种情况，需要选择 Standardize Input 进行预处理。

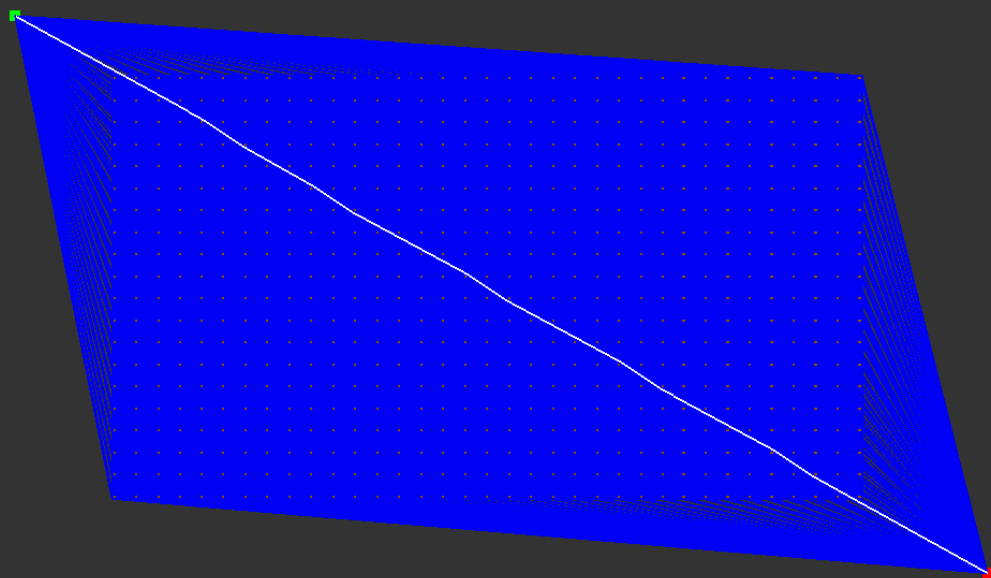
压力测试

我们的算法在比较大规模的数据下的表现仍然是比较鲁棒的，下面是我们的压力测试的结果。

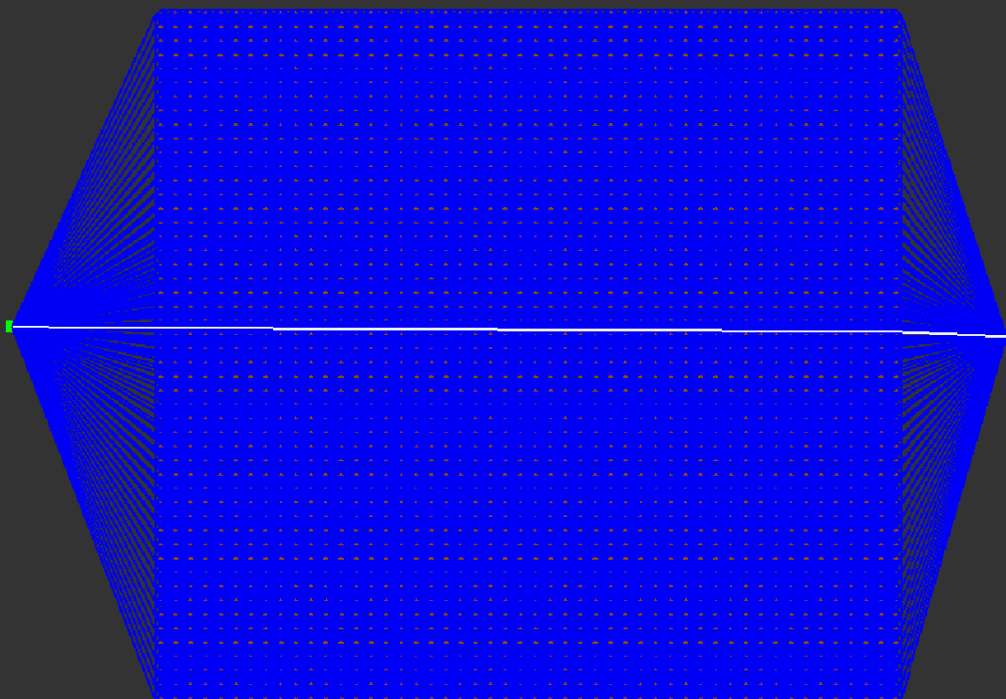
800个三角形，2400个顶点，耗时约5秒



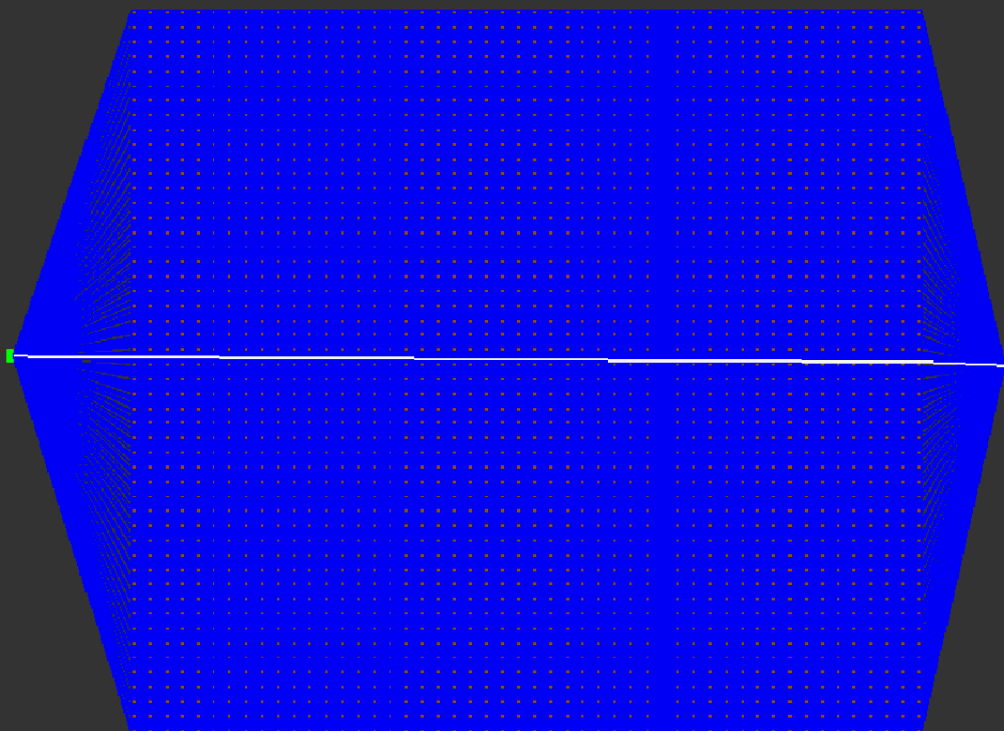
700个正方形，2800个顶点，耗时约7秒



2500个三角形，7500个顶点，耗时约62秒



2500个正方形，10000个顶点，耗时约118秒



参考文献

- [1] Der-Tsai Lee. 1978. *Proximity and Reachability in the Plane*. Ph.D. Dissertation. University of Illinois at Urbana-Champaign, Champaign, IL, USA. AAI7913526.
- [2] Ghosh, Subir Kumar; Mount, D.M., "*An output sensitive algorithm for computing visibility graphs*," Foundations of Computer Science, 1987., 28th Annual Symposium on , vol., no., pp.11,19, 12-14 Oct. 1987.
- [3] Boaz Ben-Moshe, Olaf Hall-Holt, Matthew J. Katz, and Joseph S. B. Mitchell. 2004. *Computing the visibility graph of points within a polygon*. In Proceedings of the twentieth annual symposium on Computational geometry (SCG '04). ACM, New York, NY, USA, 27-35.
- [4] Joseph S. B. Mitchell. 1993. *Shortest paths among obstacles in the plane*. In Proceedings of the ninth annual symposium on Computational geometry (SCG '93). ACM, New York, NY, USA, 308-317.
- [5] John Hershberger, Subhash Suri, *An Optimal Algorithm for Euclidean Shortest Paths in the Plane*, SIAM J. Comput, 1997, 28, 2215—2256.
- [6] Joseph S. B. Mitchell, *New Results on Shortest Paths in Three Dimensions*, Proc. 20th Annual ACM Symposium on Computational Geometry, 2004, 124—133.
- [7] Mark de Berg, Marc van Kreveld, Mark Overmars, Otfried Schwarzkopf, 邓俊辉 (译). *计算几何——算法与应用*. 第三版