## 二维空间多边形障碍环境中最短路径的规划

倪学明，计算机与信息学院

摘 要：

二维空间多边形障碍环境下的最短路径规划问题是图论、计算几何、网络设计、等领域的基本理论问题。对该问题的探索与研究有着重要的实际意义与经济价值。

路径规划问题的难点主要在于对障碍环境的表达与抽象、最短路径的搜索。本文通过实现一种被称作旋转扫描线的算法，构建多边形障碍环境的可见性图，将环境信息抽象成一张关于环境中多边形障碍物顶点间关系的带权图。该算法可以保证在O(n2logn)时间内构造出障碍环境的可见性图。对于路径搜索算法，选择实现单源最短路径的Dijkstra算法，算法可在O(n2)时间内搜索出指定起点到终点的最短路径，通过使用最小优先级队列可将算法的时间复杂度降至O(e + nlogn)。

关键词：障碍环境；路径规划；Dijkstra算法；最短路径；线段求交；可见性图；Bresenham直线算法；区域填充；

**Shortest Path Planning in Two-Dimensional Space Polygon Obstacle Environment**

NiXueming,School of Computer and Infomation

**Abstract:**

The shortest path planning problem in the two-dimensional polygonal obstacle environment is the basic theoretical problem in the fields of graph theory, computational geometry, network design, and so on. The exploration and research of this issue has important practical significance and economic value.

The difficulty of the path planning problem lies mainly in the expression of the obstacle environment and the search of abstract and shortest paths. In this paper, by implementing an algorithm called rotating scan line, the visibility map of the polygon obstacle environment is constructed, and the environment information is abstracted into a weighted graph about the relationship between the vertices of the polygon obstacles in the environment. The algorithm can guarantee the visibility map of the obstacle environment in O(n2logn) time. For the path search algorithm, choose the Dijkstra algorithm that implements the single source shortest path. The algorithm can search for the shortest path from the specified start point to the end point in O(n2) time. By using the minimum priority queue, the time complexity of the algorithm can be reduced to 0. (e + nlogn).

**Key words:** Obstacle environment; path planning; Dijkstra algorithm; shortest path; line segment intersection; visibility map; Bresenham line algorithm

目录

[第1章 绪论](#_Toc29958_WPSOffice_Level1) [1](#_Toc29958_WPSOffice_Level1)

[1.1背景概述](#_Toc18491_WPSOffice_Level2) [1](#_Toc18491_WPSOffice_Level2)

[1.2问题概述](#_Toc28195_WPSOffice_Level2) [1](#_Toc28195_WPSOffice_Level2)

[1.3本文内容简介](#_Toc367_WPSOffice_Level2) [1](#_Toc367_WPSOffice_Level2)

[第2章 经典的路径搜索算法](#_Toc18491_WPSOffice_Level1) [3](#_Toc18491_WPSOffice_Level1)

[2.1 Dijkstra算法](#_Toc23918_WPSOffice_Level2) [3](#_Toc23918_WPSOffice_Level2)

[2.2 Floyd算法](#_Toc22779_WPSOffice_Level2) [3](#_Toc22779_WPSOffice_Level2)

[2.3 A\*算法](#_Toc1065_WPSOffice_Level2) [4](#_Toc1065_WPSOffice_Level2)

[第3章 障碍环境的编辑与数据组织](#_Toc28195_WPSOffice_Level1) [5](#_Toc28195_WPSOffice_Level1)

[3.1多边形障碍环境的图形编辑器](#_Toc12626_WPSOffice_Level2) [5](#_Toc12626_WPSOffice_Level2)

[3.2环境信息的合法性检验并保存](#_Toc7019_WPSOffice_Level2) [5](#_Toc7019_WPSOffice_Level2)

[3.3最短路径规划测试](#_Toc14810_WPSOffice_Level2) [6](#_Toc14810_WPSOffice_Level2)

[3.4数据组织](#_Toc21003_WPSOffice_Level2) [6](#_Toc21003_WPSOffice_Level2)

[第4章 构建可见性图](#_Toc367_WPSOffice_Level1) [8](#_Toc367_WPSOffice_Level1)

[4.1 可见性图的定义](#_Toc11446_WPSOffice_Level2) [8](#_Toc11446_WPSOffice_Level2)

[4.2构造可见性图的算法说明](#_Toc2626_WPSOffice_Level2) [8](#_Toc2626_WPSOffice_Level2)

[4.3 子方法VisibilityVertexes的实现](#_Toc31620_WPSOffice_Level2) [9](#_Toc31620_WPSOffice_Level2)

[4.3.1 相交边的判定与二分查找树动态维护](#_Toc972_WPSOffice_Level3) [9](#_Toc972_WPSOffice_Level3)

[4.3.2可见性判断](#_Toc2199_WPSOffice_Level2) [9](#_Toc2199_WPSOffice_Level2)

[4.4实际工程中的可见性图构造](#_Toc12626_WPSOffice_Level3) [10](#_Toc12626_WPSOffice_Level3)

[第5章 Dijkstra算法的实现](#_Toc23918_WPSOffice_Level1) [12](#_Toc23918_WPSOffice_Level1)

[5.1 关键数据结构](#_Toc3849_WPSOffice_Level2) [12](#_Toc3849_WPSOffice_Level2)

[5.1.1 邻接矩阵](#_Toc21413_WPSOffice_Level3) [12](#_Toc21413_WPSOffice_Level3)

[5.1.2 顶点路径](#_Toc7019_WPSOffice_Level3) [12](#_Toc7019_WPSOffice_Level3)

[5.2算法主要逻辑实现](#_Toc27015_WPSOffice_Level3) [13](#_Toc27015_WPSOffice_Level3)

[5.3测试运行案例的结果展示](#_Toc14810_WPSOffice_Level3) [13](#_Toc14810_WPSOffice_Level3)

[第6章 画直线算法和区域填充算法](#_Toc22779_WPSOffice_Level1) [15](#_Toc22779_WPSOffice_Level1)

[6.1 Bresenham直线算法](#_Toc1922_WPSOffice_Level2) [15](#_Toc1922_WPSOffice_Level2)

[6.2 扫描线多边形填充算法](#_Toc22557_WPSOffice_Level3) [15](#_Toc22557_WPSOffice_Level3)

[第7章 总结](#_Toc1065_WPSOffice_Level1) [16](#_Toc1065_WPSOffice_Level1)

[参考文献 1](#_Toc1065_WPSOffice_Level1)7

[致谢 1](#_Toc1065_WPSOffice_Level1)8

**第1章 绪论**

**1.1背景概述**

障碍环境下的路径规划问题，可以描述为：在有障碍物的环境中，规划一条从起始点到目标点且与任何环境障碍物都无碰撞的路径。该问题是机器人运动规划、游戏AI寻路、地理信息系统、VLSI设计、导航系统等众多领域的基本问题。例如在机器人执行任务时，采用规划良好的路径可以节省大量的机器人作业时间提高机器人的工作效率并减少不必要的耗能。

而二维空间下的多边形障碍环境最短路径规划，是对广义的路径规划问题的限定和抽象。进行这样的简化和抽象是因为真实世界下三维空间中的障碍物形体复杂多样，信息处理、数据计算量大，而且三维空间下的避障路径规划问题是NP难的，目前没有解决该问题的成熟算法。

二维空间多边形障碍环境最短路径规划，首先是把三维的真实空间简化为了二维的平面空间，即在二维的平面上讨论路径规划。将路径规划问题限制在二维，看起来是条件苛刻，实际上往往进行二维空间的路径规划就可以达到现实中进行路径规划的目标——例如对于一个在厂房作工的机器人来说，只要提供包含墙壁、机床等信息的平面建筑图，就足够进行运动规划了；又如在VLSI设计中，在电路板上如何合理的安排元件，设计线路，这实质上也是避障路径规划问题，此时电路板就相当于一个二维平面。因此讨论二维空间的避障路径规划有着重要意义和实际价值。

**1.2问题概述**

二维空间多边形障碍环境最短路径规划问题将三维空间的障碍物，都抽象为了二维平面的多边形，即用二维平面的多边形表示现实的立体障碍物。如之前所说，往往进行二维空间的路径规划就可以达到在现实中进行路径规划的目标。那么如何在二维平面表示三维空间的障碍物，以达到既可以正确合理的描述障碍物，又能方便处理呢？常见的做法是在二维平面上用障碍物实体的包围盒（一个包围原障碍物俯视图边界的最小凸多边形）代替原本的障碍物。根据求解问题的实际情况，有时还会对障碍物的包围盒进行膨胀处理，以避免有体积的实际运动物体（如机器人）与障碍物边界发生碰撞。

经过抽象和简化后的二维空间的避障路径规划问题，更具普遍性，同时方便了机器对障碍环境信息的表示和简化求解问题的运算过程。

对于路径规划问题，主要难点就是环境表达，和路径搜索策略。其中环境表达是研究怎样无误的描述和记录环境信息，并进行化简和抽象以方便进行路径搜索处理。而路径搜索策略则是指规划求解路径的算法技术。在这方面已经有很多成熟的算法，例如单源最短路径的Dijkstra算法、任意两点间最短路径的Floyd动态规划算法、A\*搜索算法。在这几种算法中，环境信息被抽象成图论中定义的有向或无向图并利用图的节点邻接矩阵记录节点间的关联信息。在进行图的遍历以搜索最短路径时，以邻接矩阵为基础不断进行目标值的最小性判断，直到获得最后的最优路径。

**1.3本文内容简介**

本文首先在第二章介绍了几种典型的路径搜索算法——单源最短路径的Dijkstra算法、任意两点间最短路径的Floyd动态规划算法、和A\*搜索算法。

1. 概括了本文实现的测试工具主要功能和使用方法，以及实际工程中表示和存储多边形障碍环境信息的数据结构。
2. 详细介绍了构造障碍环境信息的可见性图算法的实现细节。该算法是一种被称为旋转扫描线的算法，它对多边形障碍环境进行处理，将多边形障碍环境抽象成一张可见性图（保存了环境中多边形障碍物顶点间关联关系的矩阵）。

第五章 讲述了测试工具中实现的Dijkstra算法，对一些编程细节、数据结构进行了详述。并且在最后展示了测试工具进行路径搜索测试的完整流程，和测试结果。

本文最后还给出了在工具中用到的一些经典的图形算法（画线算法，区域填充算法）。

1. **经典的路径搜索算法**

本章介绍几种经典的路径搜索算法，包括求单源最短路径的Dijkstra算法、任意两点间最短路径的Floyd算法、和A\*搜索算法。

**2.1 Dijkstra算法**

Dijkstra算法运用的是贪心法，其基本策略是在一个带权图中，选择一个起点，依次构造出从起点到其余各顶点的最短路径，每次构造出一条新的最短路径，就会根据其信息调整更新起点到其它顶点的路径，再从中选取新的最短路径，直到起点到终点的最短路径被选出。

算法会维护两个关于顶点的集合T和S。集合S中保存所有已经确定最短路径的顶点，而集合T则暂存其它所有还未确定是最短路径的顶点。集合S初始状态为空，而后算法的每一次循环都会确定起点到一个顶点的最短路径，并将该顶点从T移到S。

设dis[v]表示从起点到顶点v的路径距离，而且这条路径所经过的节点必须是集合S中的顶点。

初始时，若一个顶点v与起点之间有边，则把dis[v]设为起点与顶点v之间的距离，否则把dis[v]设为无穷大。

如果一个顶点u的dis[u]的值是集合dis中最小值时，就意味着不能通过别的顶点来减小从起点到顶点u的距离了，所以dis[u]所表示的这条路径即是从起点到顶点u的最短路径。这时就要将顶点u从T中删除并添加到S中去。

当一个顶点u从T中转移到了S中，此时需要对集合T中所有顶点的路径进行更新（这被称为边的扩展）——如果存在一条从u到v的边(u,v)，通过将边(u,v)添加到从s到u的路径尾部，可以缩短从s到v的路径长度。那么就对顶点v的路径进行更新，将它的dis[v]的值更新为dis[u]加上(u,v)边的距离的和。

Dijkstra算法最简单的实现方法是用一个链表或者数组来存储所有顶点的集合T,所以搜索T中最小元素的运算只需要线性搜索T中的所有元素。这样的话算法总得的运行时间是O(n2)。而对于边数少于n2稀疏图来说，我们可以用邻接表来更有效的实现Dijkstra算法。同时需要将一个二叉堆或者斐波纳契堆用作优先队列来寻找最小的顶点。当用到斐波纳契堆的时候，算法所需的时间可以减少为O(e+nlogn)。

**2.2 Floyd算法**

Floyd算法是一种多源最短路径算法，通过动态规划的思想来寻找给定加权图中多源点间的最短路径。其核心思想是：基于加权图的权值矩阵，对于图中的每一对顶点(u,v)间的最短路径，分成对每一个顶点k的子测试——若借助当前测试的顶点，可以缩小顶点u到v的权值，则更新(u,v)的权值为新的权值，当所有顶点都被测试过后，得到的矩阵便是各顶点对之间最小权值矩阵。

需要准确理解Floyd算法关键点就在于解读它的递推公式：设Di,k,j为从i到j的只以集合（1...k）中的节点为中间节点的最短路径。那么Di,n,j所表示的就是i到j的最短路径。

假设已知di,k-1,j ，求di,k,j（即已知i到j只经过集和(1...k-1)中的节点的最短路径，求i到j只经过集和(1...k)中的节点的最短路径）；我们需要测试的便是，经过k顶点会不会缩短i到j的路径。

如果最短路径经过点k，那么di,k,j = di,k-1,k + dk,k-1,j ；否则di,k,j = di,k-1,j 。可以推出算法进行迭代更新的递推关系：di,k,j = min( di,k-1,j , di,k-1,k + dk,k-1,j )。

算法要求测试的顶点是按次序进行的，即在求di,k,j时必须已经求得了所有顶点只经过k-1顶点的最短路径。算法主体是一个三层循环，时间复杂度为O(n3)，算法伪代码如下：

1.将图的邻接矩阵复制到矩阵d[n][n]

2.for( k ← 1 to n)

3. for( i ← 1 to n)

4. for( j ← 1 to n)

5. If(d[i][j] > d[i][k] + d[k][j])

6. d[i][j] = d[i][k] + d[k][j]

**2.3 A\*算法**

A\*搜索算法是是一种启发式搜索算法，其核心在于对估值函数的设计。A\*算法不像上述两种算法一样有固定的实现结构，根据不同的实际情况设计不同的估值函数，是A\*算法的灵活性所在。正是因为具有这种灵活性，A\*搜索算法常用于游戏中NPC的寻路。不同的游戏对于NPC寻路的需求各有不同，此时路径搜索不再是单纯的进行最短路径的搜索，更需要考虑游戏特有的因素，因此需要设计符合游戏需求的估值函数，以实现A\*算法，可以实现更符合游戏NPC的寻路算法。

A\*算法综合了最良优先搜索和Dijkstra算法的优点，既通过进行启发式搜索来提高算法效率，又保证可以找到一条最优路径。

设A\*算法的估值函数为：f(n) = g(n) + h(n)。其中g(n)表示起点到任意顶点n的实际距离，h(n)是启发函数，可以表示为任意顶点到目标顶点的估算距离。当g(n)为0时，即只对顶点进行估值，进行启发式搜索，此时算法便是贪心策略的最良优先搜索，速度最快，但是可能得不到最优解；当h(n)为0时，即不进行估值，此时就等价于单源最短路径问题，和Dijkstra算法相同，此时需要计算的顶点数最多。

所以一个好的A\*算法在于设计一个优良的估值函数，该函数在保证可以得到最优解的同时，需要尽可能的减少计算的顶点数。常见的估值函数有切比雪夫距离（二个点之间的距离定义为其各座标数值差的最大值），曼哈顿距离（两个点之间的距离定义为在标准坐标系上的绝对轴距之和）。

**第3章 障碍环境的编辑与数据组织**

要进行二维空间多边形障碍环境的最短路径规划，首先就是要处理障碍环境信息的输入和环境信息的数据组织问题。

为了便于输入多边形障碍环境信息，本人制作的测试工具中，使用unity制作了一个图形化的多边形障碍环境编辑器。测试工具在保存环境信息之前，还会对环境中的出现的相交多边形、自交多边形、非多边形等无效非法的图形进行自动识别和删除。

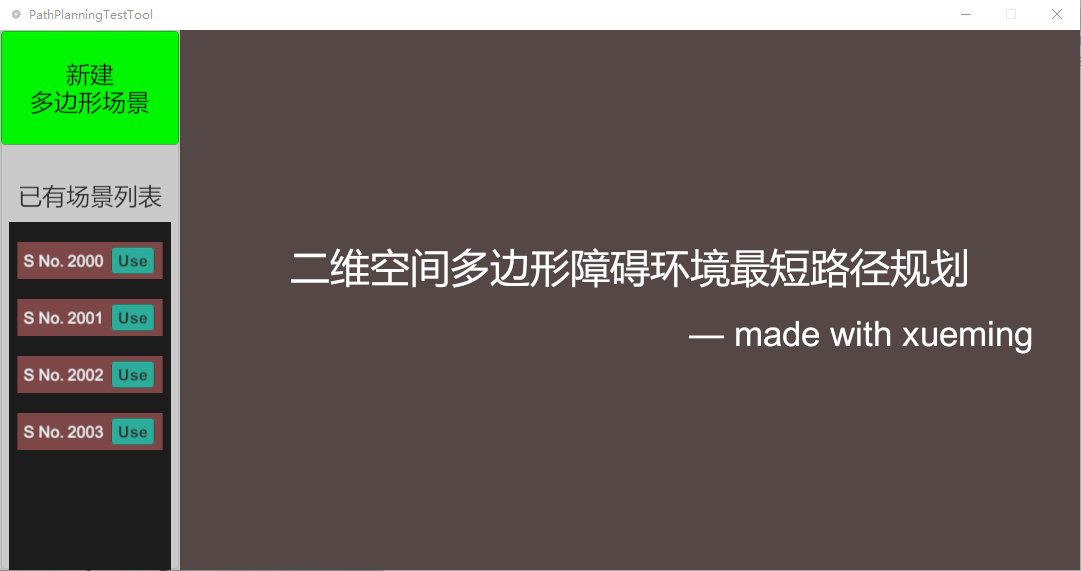


图3-1 测试工具主界面

**3.1多边形障碍环境的图形编辑器**

进入工具后点击主界面的新建多边形场景，即进入多边形障碍环境的编辑界面，如下图所示

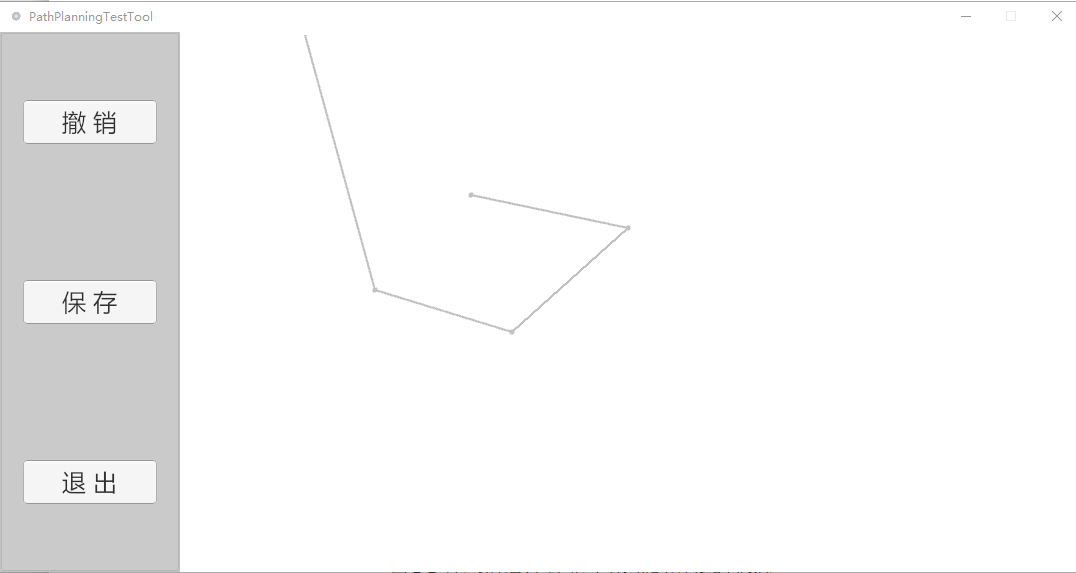


图3-2 测试工具环境编辑界面

用户通过鼠标在白色画板上进行多边形的编辑。编辑器允许用户在一个画板上编辑生成多个多边形，用户每次点击鼠标会在画板上确定一个多边形的顶点（并从该点发出一条连接鼠标的线段，当下一顶点确定，就自动成为连接这两个顶点的边）。程序会自动记录输入的多边形的第一个顶点为该多边形的起点，当后续用户指定的顶点在起点附近，程序默认用户要结束当前多边形的编辑，并将多边形的终点落在起点同一位置处，以形成一个封闭的多边形。

**3.2环境信息的合法性检验并保存**

当用户想要结束当前多边形障碍环境的编辑时，点击界面右边的保存按钮即可。程序将自动进行环境信息的合法性检查（多边形判交检查，未形成封闭空间的多边形检查）。检查不通过时，程序会自动把非法的多边形删除，并弹窗提升继续完善环境信息。



图3-3 测试工具错误提示界面

检查通过后环境信息将会被成功保存，并赋予一个环境编号，归到环境列表中，并保存到本地文件。

**3.3最短路径规划测试**

点击主界面的已有的场景列表上的use按钮，即可进入最短路径规划的测试运行界面。

在进行测试前，用户需要在场景上指定起点和终点，程序将会屏蔽将起点或者终点设在多边形内部的操作。同个环境可测试多次，进行多次不同的起点与终点的指定。

当用户确定好起点与终点后，即可点击测试按钮，进行路径规划测试。程序会将算法的运行过程分两个阶段：

1.构造可见性图，这个阶段会生成环境信息的可见性图，并在环境中用绿色的细线连接两个可见的顶点。

2.最短路径搜索，这个阶段会运行Dijkstra算法，进行最短路径的搜索。程序会按算法生成的每条最短路径的顺序，在环境中用紫色的线代替原先的绿色线。

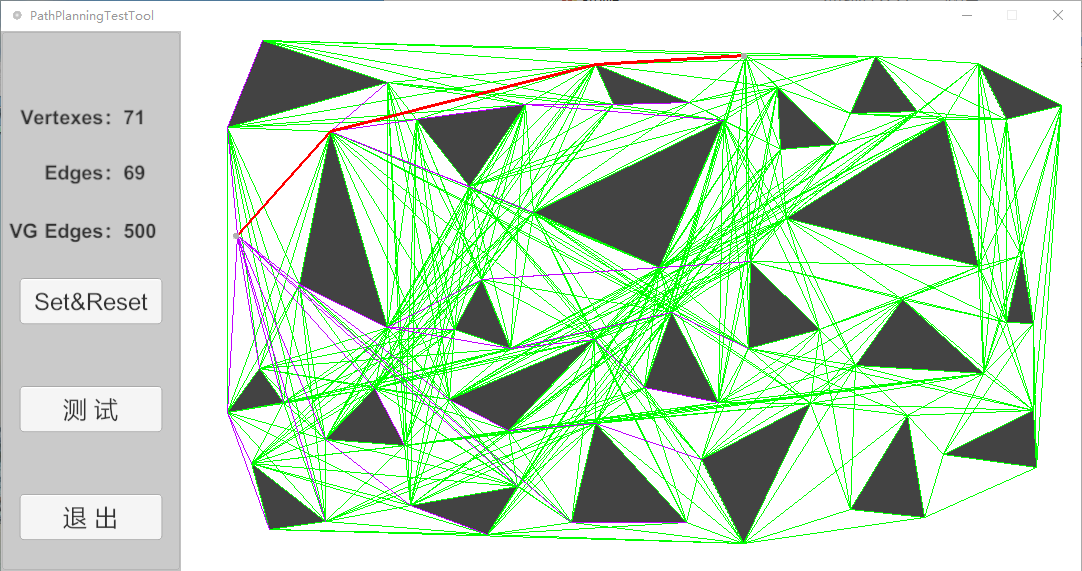


图3-4 测试工具运行结果界面

当搜索到起点到终点的最短路径（因为前面的环境构造已经屏蔽了不存在最短路径的情况，所以这里必定会搜索到一条最短路径），算法便运行结束。程序会用红色线段标识最终的最短路径，并会显示场景中的顶点数可见图的边数等信息。

**3.4数据组织**

在测试工具中涉及对环境的编辑（即编辑多边形障碍物），修改和保存。所以在组织多边形障碍环境信息的数据结构时，应该方便进行工具中涉及的操作。下面是实际工程中多边形障碍环境信息的数据结构，代码截图皆来自真实的工程代码。



图3-5 进行存储的多边形场景数据结构

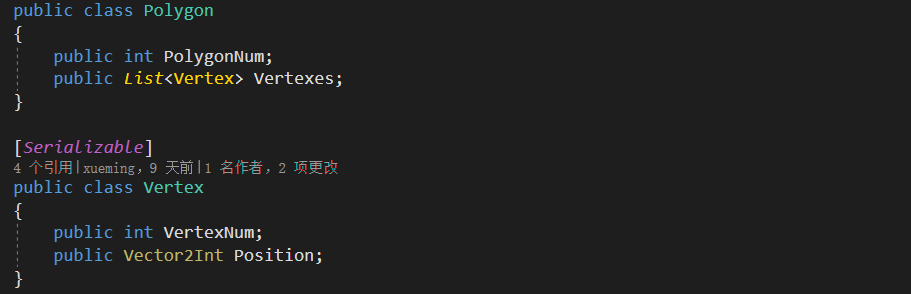


图3-6 进行存储的多边形数据结构

以上是用于保存的多边形障碍环境信息的数据结构截图，是一个嵌套结构，程序会对多边形场景进行唯一编号（从2000开始按序编号），然后对场景中的多边形也进行唯一编号（从1000开始按序编号），而多边形包含的顶点也会有一个从0开始的编号。这些编号起着对场景、多边形、顶点的索引作用，方便程序查找。

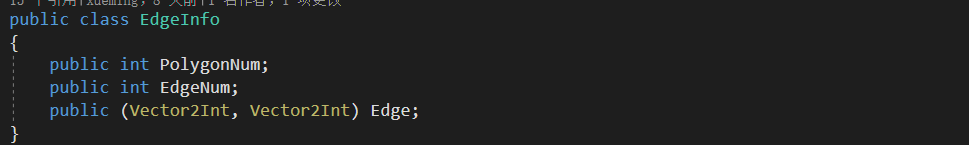


图3-7 多边形边的数据结构



图3-8 多边形顶点的数据结构

以上两张截图是组织多边形障碍环境中的顶点和边的信息的数据结构。我们需要通过边来查找到所属的多边形，通过顶点查找到对应的多边形和起入边、出边的编号。保存的这些额外信息，在之后的程序中都会用到——比如进行环境信息的合法性检验，当检测到相交边时，需要索引到对应多边形进行删除；生成可见性图时，需要判断两个顶点是否同属于一个多边形和一条边，以进行特殊情况的处理。

**第4章 构建可见性图**

本章介绍可见性图的构造，包括在测试工具中程序实现的一些细节。

**4.1 可见性图的定义**

首先回归到本文所讨论的问题二维空间多边形障碍环境最短路径规划，问题的输入，显然是一组多边形，而在程序中表示多边形只需要记录多边形的顶点位置即可。然而，进行路径搜索的Dijkstra算法，需要的是一张带权图的顶点间的距离关系。所以我们需要把离散的多边形顶点信息抽象成一张带权图，从而得到Dijkstra算法的输入，再通过Dijkstra算法进行路径搜索，得到最短路径。

我们可以将起点S到终点E的最短路径想象成一根有弹性的橡皮筋：其两端被固定在起点S和终点E处，由于不能穿过多边形，其形状被强制与路径一致。最终这根橡皮筋上的转折点一定是场景中多边形障碍物的顶点。

引理：穿行于一组互不相交的多边形障碍物之间、从起点S到终点E的任何一条最短路径，都是一条多边形路径，其上所有的内部顶点都是场景中多边形障碍物的顶点**[3]**。

在确定最短路径上述的特性后，我们可以知道最短路径上的顶点一定是多边形障碍物的顶点。那么我们可以根据所有多边形障碍物的顶点和起点、终点，构造出一张线路图，该图中的边连接两个顶点，并且不与任何多边形相交。这张线路图就是所谓的可见性图。

**4.2构造可见性图的算法说明**

为了构造可见性图，必须找出所有相互可见的顶点对，直观上求解，便是对所有顶点对之间的连线进行测试，看其是否与某个多边形相交，这样做的时间复杂度将是O(n3)。显然我们应该考虑寻找更有效的算法，降低时间复杂度。

有一种被称为旋转扫描线的算法，该算法保证任意给定一组互不相交的多边形障碍物，其可见性图可以在O(n2logn)时间内构造出来，其中n为障碍物中顶点总数。

该算法的核心思想是，对于可见性图中的每一个顶点p，按照围绕p的环形次序来判断其余顶点与其的可见性，并动态地维护某些信息，以帮组对下一待处理的顶点做出判断。

算法在遍历生成一个顶点p的所有可见顶点时，先按照从p发出的指向其它顶点的射线的极角顺序，同极角则按顶点之间的距离将其他顶点进行排序，并且使用一个二分查找树存储当前状态下于射线相交的边用于快速检查。

该算法实际上降低的时间复杂度在于：判断两个顶点间的线段与多边形是否相交这一步，这个问题实质上是判断两个顶点间的线段与所有多边形的边是否相交，是计算几何中典型的线段求交问题。算法通过判断多边形的边相对于顶点p发出的射线位置关系，放入一个二分查找树中进行动态维护从而减少了这部分的搜索时间。

下面是CreateVisibilityGraph算法的伪代码，其输入是多边形障碍物环境中多边形的顶点信息以及起点和终点，输出是多边形障碍物环境的可见性图VisGraph。

1.对图VisGraph进行初始化，使其包含所有多边形的顶点和起终点，并建立顶点和它的入边出边之间的关联信息

2.for(对于每一个顶点i ← 0 to n)

3. 初始化保存可见点的列表w为空

4. do w ← VisibilityVertexes(i, VisGraph)

5. 对于w中的每个顶点，在VisGraph中设置顶点i和顶点w之间的距离

6.return

**4.3 子方法VisibilityVertexes的实现**

CreateVisibilityGraph算法中，子方法VisibilityVertexes将找出所有对顶点i可见的顶点，并返回它们编号的一个列表。

**4.3.1 相交边的判定与二分查找树动态维护**

令pw是从顶点p出发的一条射线，指向要检测的顶点w。算法需要动态维护一个边的二分查找树T，该二分查找树中保存的边是所有与射线pw相交的边。当我们判断一个点是否与顶点p可见，只需要对T进行查找，看是否有相交边在w前面。

初始化时，令射线r平行于x轴，指向x轴正方向。找出所有与r真相交的多边形障碍物边，按照与射线r相交的次序，存放到查找树T中。

随后对每个顶点w进行检查时，将w相关联的所有多边形障碍物边落在射线pw顺时针一侧的插入到查找树T中，落在逆时针一侧的从T中删除。

这里涉及的一个几何问题——线段与射线的位置关系。通过分析线段与射线的三种位置关系，可以得到，当线段与射线相交时，线段的两个端点a、b与顶点p的连线和射线r的顺时针夹角必定是一个小鱼180度一个大于180度，且pa和pb的顺时针夹角小于180度。

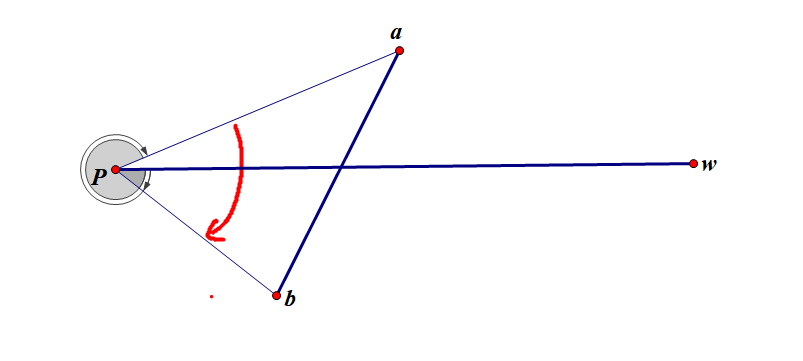


图4-1 线段与射线的位置关系示意图

所以我们只需对线段两端点a、b和顶点p的连线与射线之间的夹角关系进行比较判断，便可以得到线段ab与射线r的位置关系。下图是工程代码中关于射线对边的划分方法：

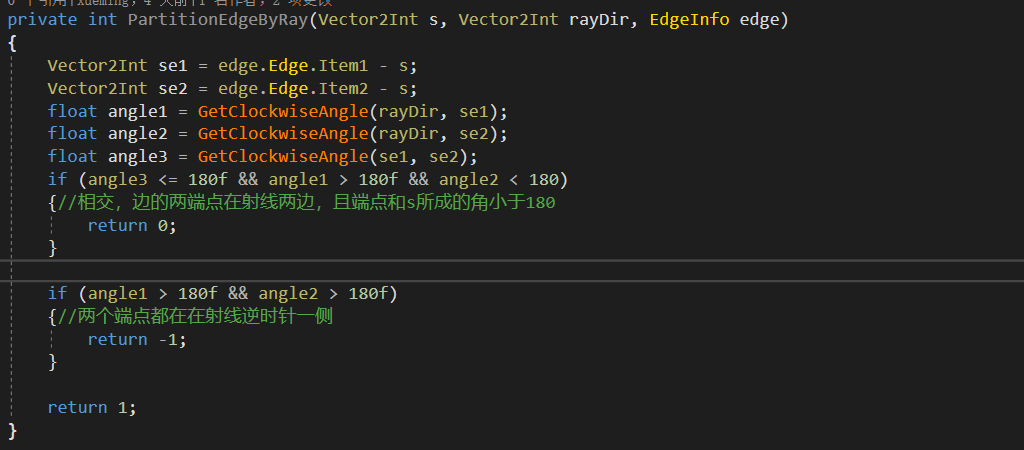


图4-2 射线对边的划分方法

**4.3.2可见性判断**

对于点w与p之间的可见性判断，一般情况下，只需要对相交边的查找树T进行查找，若是有相交边落在w之前，则可判定w相对于p是不可见的。但是对于一些特殊情况，仅仅对T进行查找判定是不够的，这里需要额外小心。

如果pw上包含其它顶点的话，设在w前的一个顶点为wi-1。若wi-1对p不可见，那么w自然也不可见。若wi-1对p可见，w的可见性有以下几种情况**[3]**：

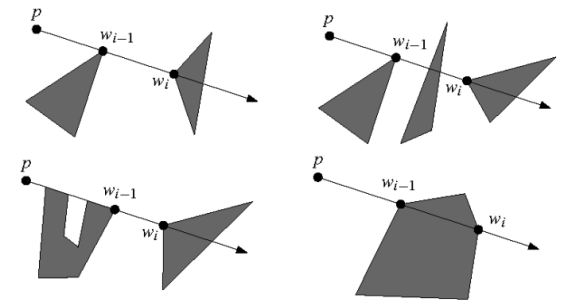


图4-3 点w与点p可见性四种情况

上图左侧两种情况wi是可见的，右侧的两种情况wi则是不可见的。在这两种情况中，要么wi-1和wi都属于同一障碍物，而且线段wi-1wi完全落在该障碍物的内部；要么线段 wi-1wi与T中的某条边相交（若是后一种情况，则这条边夹在wi-1 与wi之间，因此它必然与wi-1wi真相交）。

根据以上分析，可以得到一个处理点w对点p的可见性的子方法Visible:

1. if ( pwi与 wi所属的障碍物相交于内部)

2. return false

3. else if (i = 1) or (wi-1不在线段 —pwi上)

4. 找出T中最左侧的叶子，取出其中存放的边e

5. if (e存在，而且pwi与e相交)

6. return false

7. else

8. return true

9. else if (wi-1不可见)

10. return false

11. else

12. 在T中查找与wi-1wi相交的一条边e

13. if (e 存在)

14. return false

15. else

16. return true

**4.4实际工程中的可见性图构造**

上述构建可见性图的旋转扫描线算法时间复杂度为O(n2logn)，其中n2来自对n个顶点与其它每个顶点进行可见性判断，logn来自对所有边的相交测试。进行路径搜索的Dijkstra算法的时间复杂度为O(n2)。可以看出，构造可见性图是解决整个最短路径规划问题最花时间的一个环节。

同时由于特殊情况和程序进行几何运算时很容易出差错，在实际的测试工具的制作中我并没有采用该算法。而是用直观的方式，在进行边的相交测试时，直接对所有边遍历。通过在进行真正的判交计算时进行一些特殊条件快速筛选判定，减少运算量，并减小误判漏判以及运算误差的概率。

这种直观的做法，相对来说更容易把控和跟踪查找错误。

下面是实际工程中的构造可见性图的方法代码：



图4-4 创建可见性图的方法代码

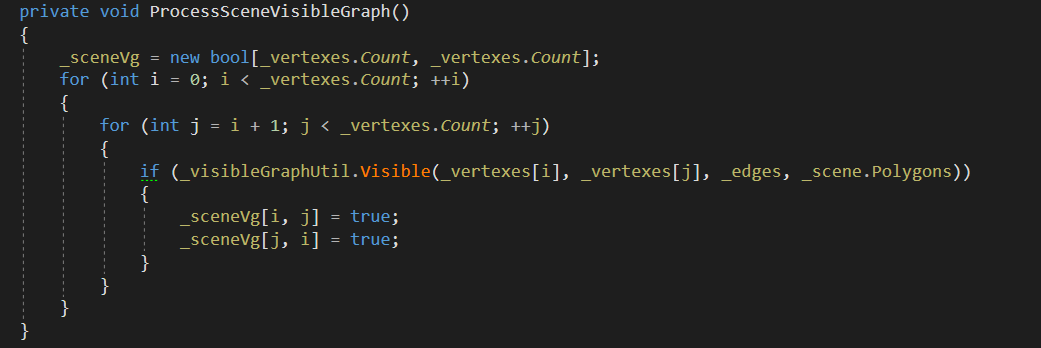


图4-5 生成多边形场景的可见性图

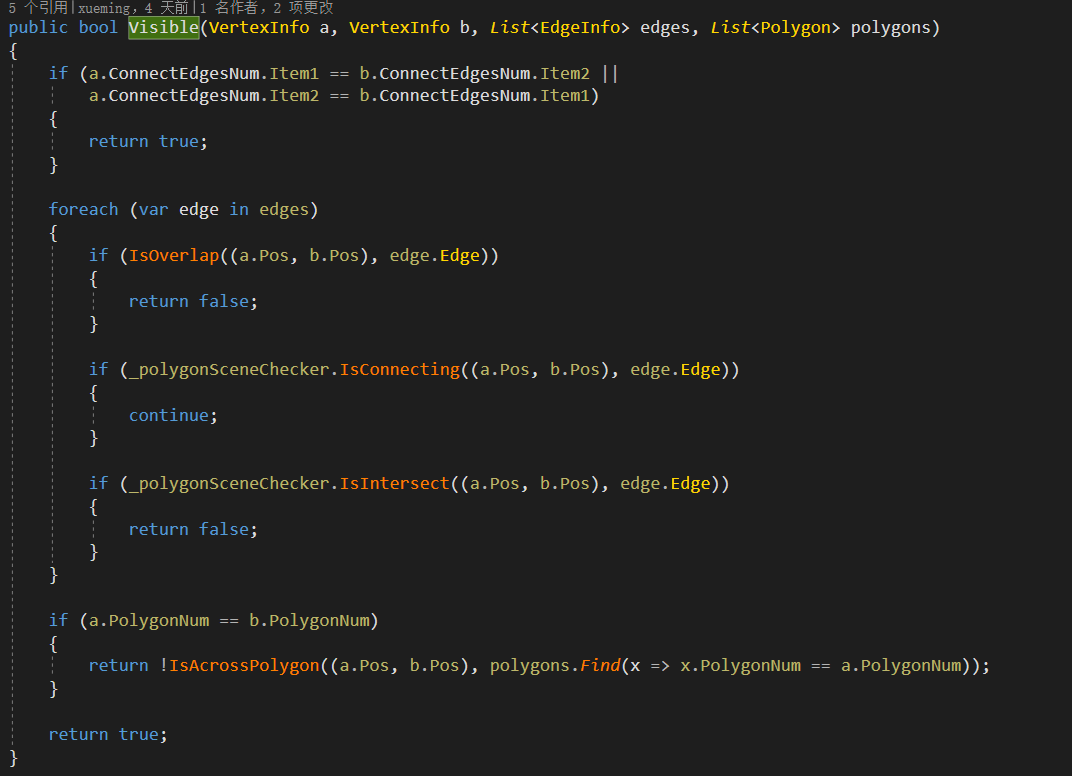


图4-6 两点间的可见性判断

在测试工具中，由于会对同一个多边形障碍环境进行多次测试，所以将可见性图的构造分为SceneVisibleGraph（环境中多边形障碍物之间的可见性图）和AppendVisibleGraph（起点、终点相对于多边形顶点的可见性图）两部分的构造。对于相同的环境就只需构造一次环境信息的可见性图，进行测试时再追加上起点和终点的可见性图。

**第5章 Dijkstra算法的实现**

经过上一章的处理，多边形障碍环境信息被抽象成一张可见性图，作为Dijkstra算法的输入。而在测试工具实际的程序中，构造的可见性图信息只是作为一个布尔值的二维数组传入到Dijkstra算法。

**5.1 关键数据结构**

设计合适的数据结构，对算法的实现起着至关重要的作用，本小节介绍Dijkstra算法中涉及的两个关键数据结构，并且给出了其定义与初始化代码截图。

**5.1.1 邻接矩阵**

邻接矩阵是常见的用于保存带权图的顶点间相邻关系的矩阵，常见的Dijkstra算法的输入便是一个保存了带权图的顶点间边的距离的邻接矩阵。

上一章中，构建出的可见性图，标识了两个顶点间是否有边，实际上构造出的可见性图被保存为一个布尔值的二维矩阵。在进入Dijkstra算法时，先借助它对邻接矩阵进行初始化，再将记录顶点间距离关系的邻接矩阵作为Dijkstra算法的输入。

下面是邻接矩阵的初始化代码截图：

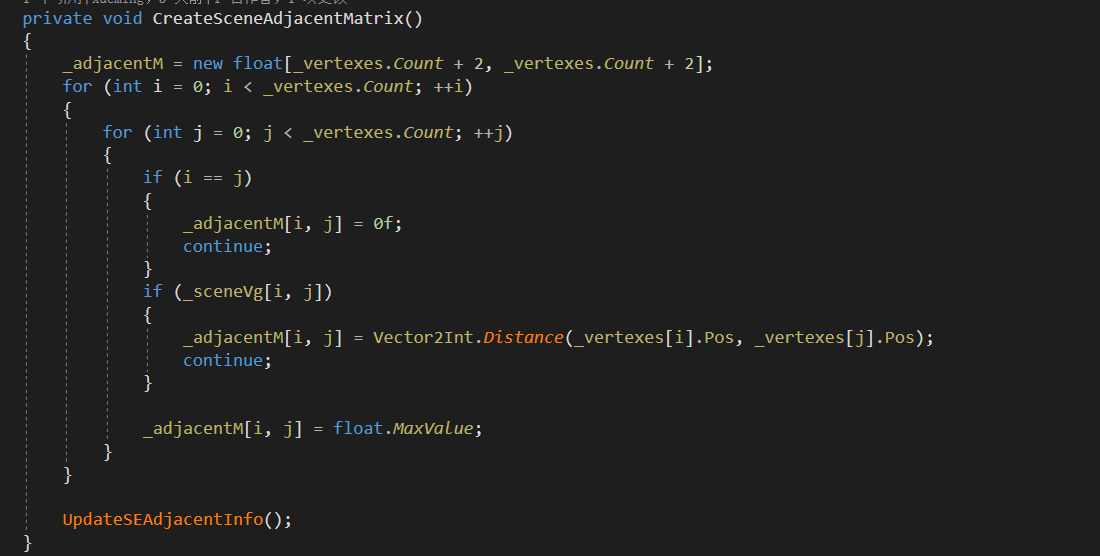


图5-1 邻接矩阵的初始化

**5.1.2 顶点路径**

根据第二章所述，我们可以这样定义某一顶点路径：从起点s到顶点v的一条路径，路径上经过的任一顶点u，从起点s到u的最短路径必须已经被确定。

算法的核心循环便是维护和更新起点到其它各顶点间路径的“最短”距离。每确定从起点到一个顶点的一条最短路径，就要借助它去更新其余还未确定是最短路径的其余路径。

首先我们需要描述从起点出发到另一顶点的路径的数据结构，这个数据结构必须记录的是路径距离，路径末顶点编号，该数据结构如下：

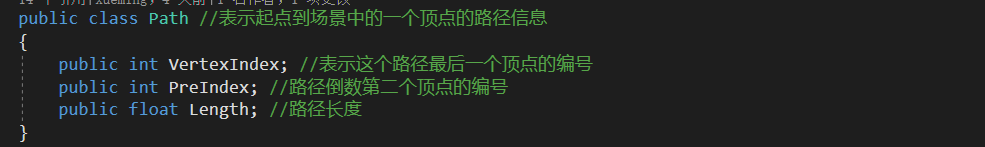


图5-2 描述路径的数据结构

其中还额外附加了一个表示路径末顶点前置顶点编号，这项信息相当于一个链指针，方便最后构造一条完整的路径。

我们需要维护两个路径集合，一个保存暂未确定是从最短的顶点路径，一个是已经确定是最短的路径，这两个集合的定义与初始化如下：



图5-3 Dijkstra算法中两个路径集合的定义

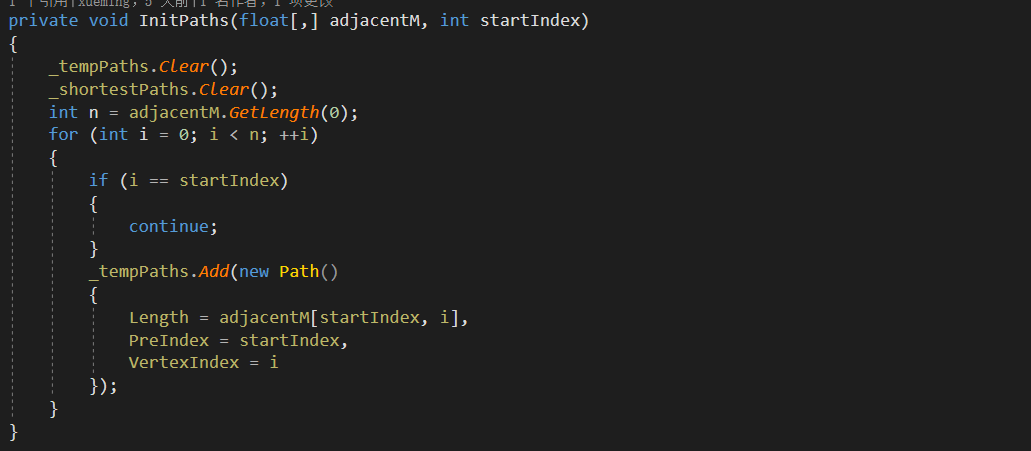


图5-4 路径集合的初始化

**5.2算法主要逻辑实现**

算法运行过程中，每次从\_tempPaths集合中选出一个路径距离最小的路径，它不可能再通过其它顶点来缩短路径距离，所以可以确定它已经是起点到该顶点的最短路径，因此将它移到\_shortestPaths集合中。再借助该路径的顶点对\_tempPaths集合中的路径一一测试，更新它们的路径距离。下图是算法的核心循环：

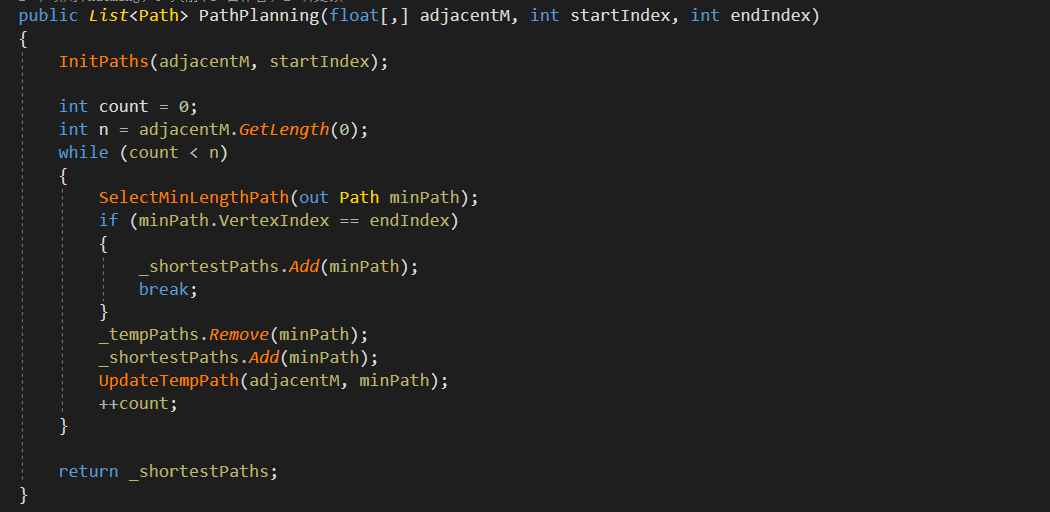


图5-5 Dijkstra算法的主循环

循环当从\_tempPaths集合中选出的最小路径的顶点编号和终点编号一样时，表明起点到终点的最短路径已经搜索到了，即退出循环。

**5.3测试运行案例的结果展示**

到目前为止，我们已经完整的实现了二维空间多边形最短路径规划问题涉及的主要算法。下面的图5-6到5-8是使用测试工具对一些测试用例的实验结果的截图，其中障碍环境中的多边形数从少到多，从简单的凸多边形，到包含有复杂凹多边形，测试工具都能够搜索到最短路径。

可以看到随着用例中障碍环境的多边形增加，可见性图的边数也在成倍的增加。原本在测试工具中打算记录进行最短路径规划所需的时间，但由于可编辑的多边形数量无法达到一定的量级，测试用例的结果都是在按下测试按钮后立马得到的，运行时间几乎可以不计。

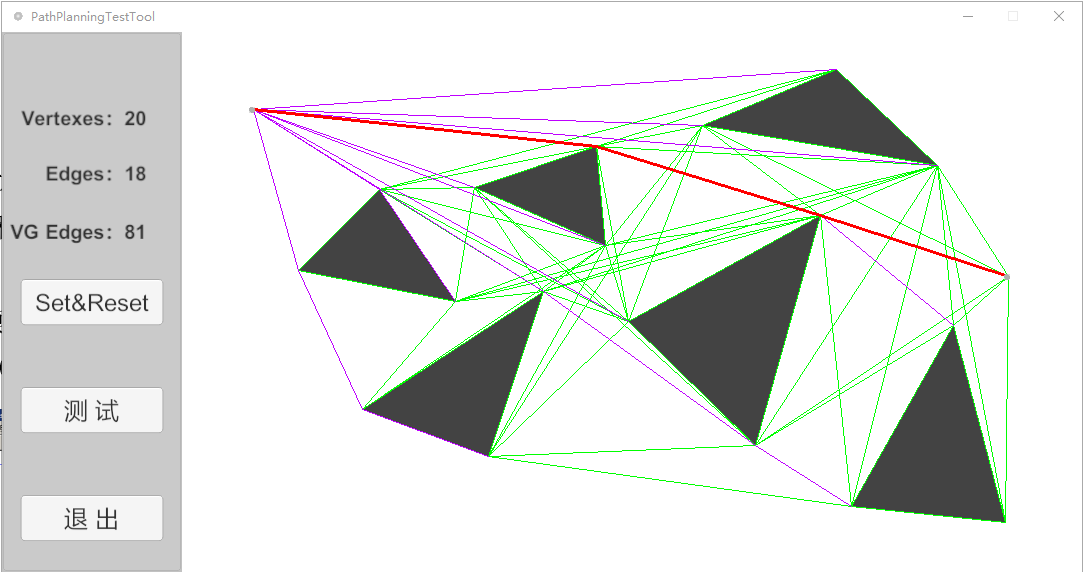


图5-6 简单用例的测试结果

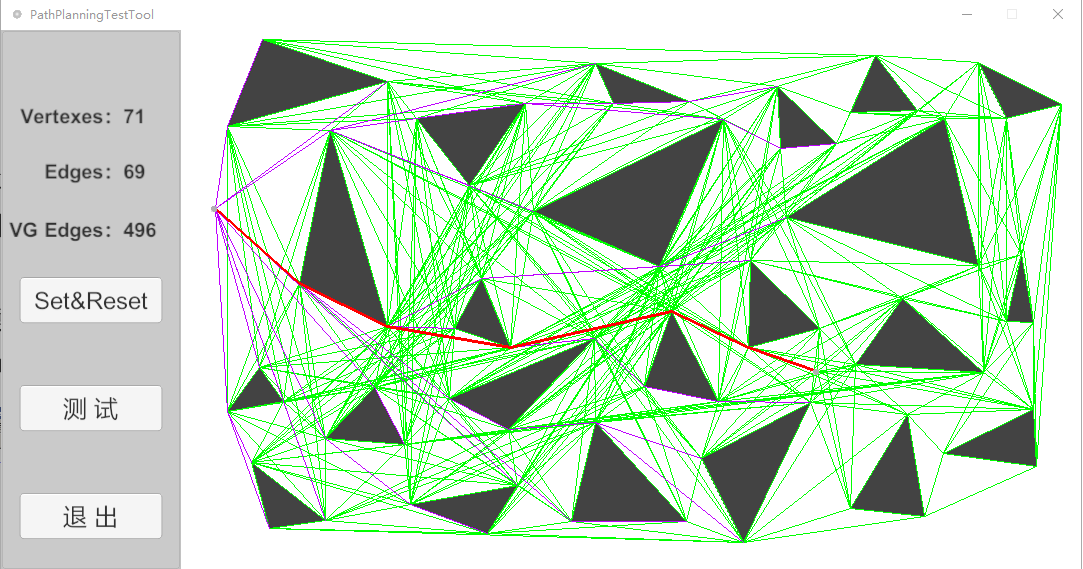


图5-7 中等规模用例的测试结果

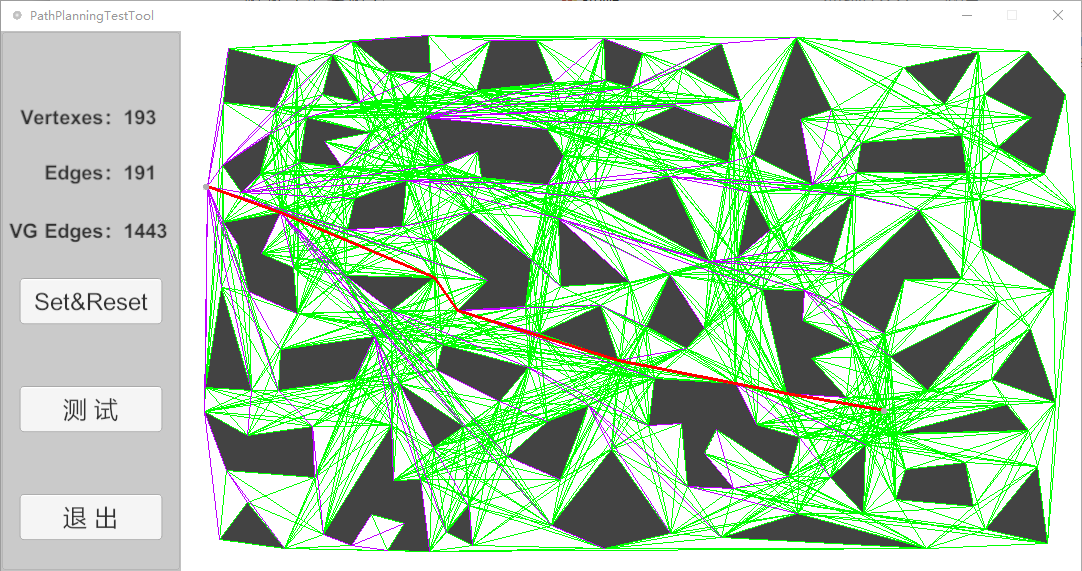


图5-8 复杂测试用例的测试结果

**第6章 画直线算法和区域填充算法**

由于unity并没有开放在图片纹理上的画图接口，只留出一个最基础画点接口。所以测试工具的程序中，需要自己实现划线算法和区域填充算法。

**6.1 Bresenham直线算法**

工程中采用的划线算法是布雷森汉姆Bresenham直线算法，该算法是一种精确而有效的光栅线生成算法，它仅仅使用了增量整数计算。

下图是画任意方向的的Bresenham直线算法的代码：

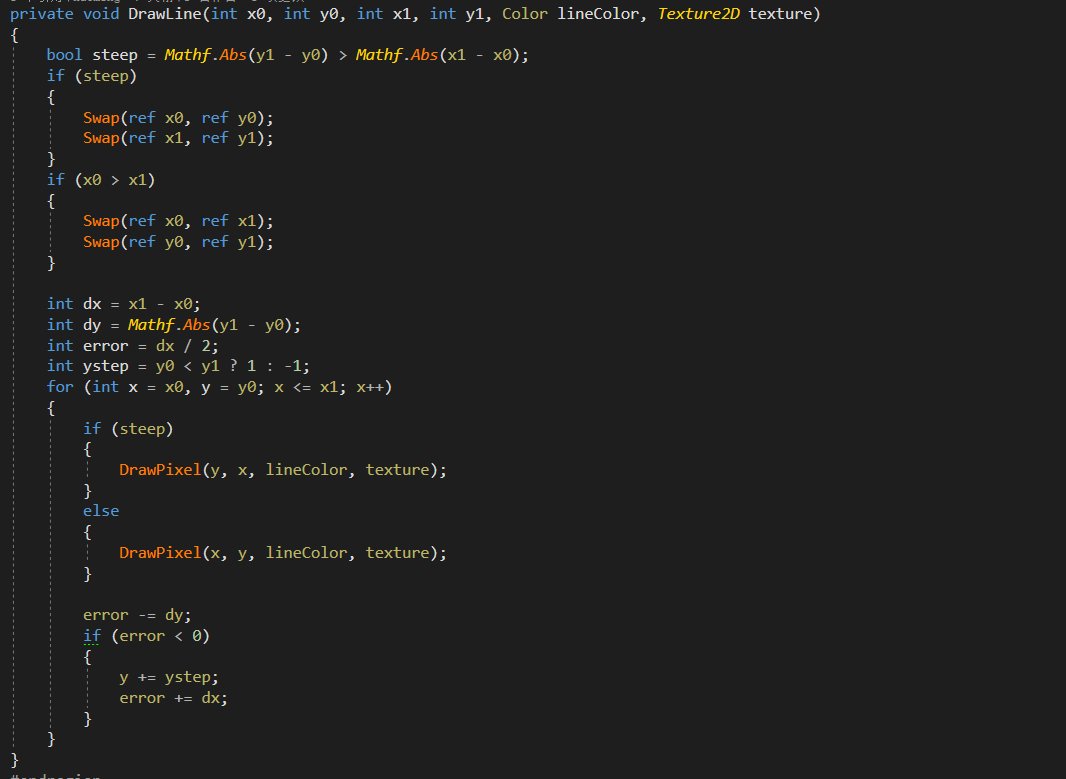


图6-1 Bresenham划线算法

**6.2 扫描线多边形填充算法**

另外测试工具中对多边形障碍物进行了颜色填充，经过颜色填充后，可以借助判断当前像素点的颜色来达到屏蔽用户将起点与终点的放置在多边形内。

工程中采用的是典型的扫描线多边形填充算法，其大致过程是：使用一条水平的扫描线，从多边形最下端的顶点一直扫描到最上端顶点，水平的扫描线会与多边形交于一系列点，将这些交点按照x轴坐标排序，两两配对，然后依次填充交点间的线段。

下图是扫描线多边形填充算法的实现代码：

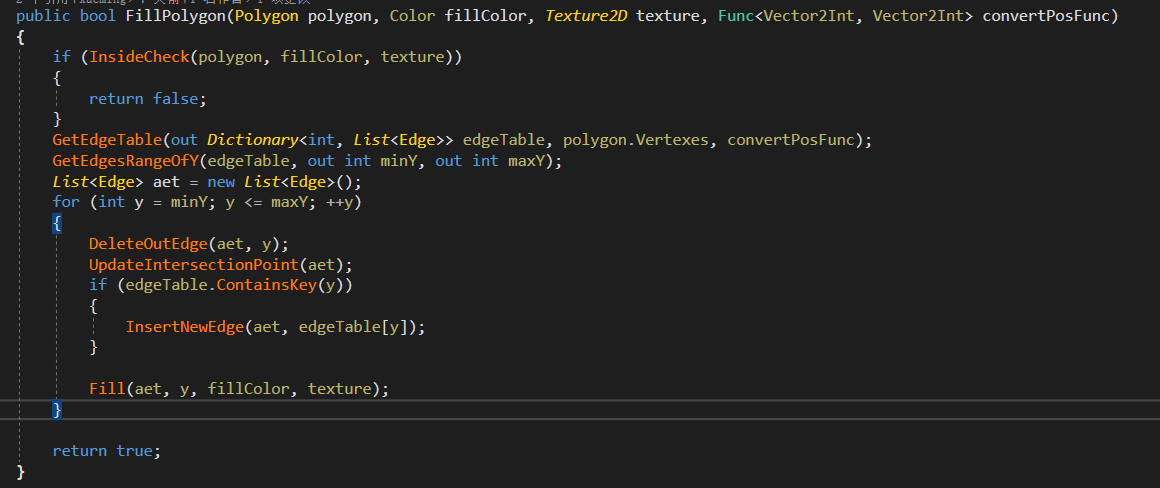


图6-2 扫描线多边形填充算法

**第7章 总结**

本文主要讨论二维空间的多边形障碍环境下的最短路径规划，概述了几种典型的路径搜索算法。并选择其中Dijkstra算法进行了编程实现，同时用unity制作了一个图形界面的测试工具。

虽然在写论文前期查阅了大量与论题相关的资料与书籍，对于问题的求解有了很好的理解，但在实现测试工具时，仍然碰到了许多问题和难点——例如进行环境信息的检验处理时，就涉及如何高效的实现点定位的问题。

通过实践，使我对求解二维空间的多边形障碍环境下的最短路径规划问题方面的理论知识有了更好和更深的理解与掌握。

**参考文献：**

[1] 杨素琼，林碧琴，何伟.基于A\*算法的地图路径搜索的实现[J].铁路计算机应用，2000,9(4)：8-11

[2] 唐文武，施晓东，朱大奎.GIS中使用改进的Dijkstra算法实现最短路径的计算[J].中国图象图形学报，2000,5(A)(12)：1019-1023

[3] Mark de Berg，Otfried Cheong，Marc van Kreveld，Mark Overmars.计算几何——算法与应用（第三版）[M].北京：清华大学出版社，2008.

[4] 周培德.计算几何[M].北京：清华大学出版社，2000.

[5] 卢开澄，卢华明.图论及其应用（第二版）[M].北京：清华大学出版社，1997.

[6] 李颖等.Open GL图形程序设计指南[M].北京：中国水利水电出版社，2001.

[7] Donald Hearn，M.Pauline Baker，Warren R.Carithers.计算机图形学（第四版）[M].北京：电子工业出版社，2014.

[8] 朱大奇，颜明重.移动机器人路径规划技术综述[J].控制与决策，2010(07)

[9] 王树西，吴政学.改进的Dijkstra最短路径算法及其应用研究[J].计算机科学，2012，39(5)：223-228

[10] 李卓君.**Dijkstra和矩阵迭代两种算法的对比研究**[J].[电脑与信息技术,2012,20(6):16-20](http://www.cnki.com.cn/Article/CJFDTotal-dnxj201204007.htm" \t "http://journal.just.edu.cn/jweb_zkb/article/2017/_blank)

[11] 赵卫绩，巩占宇，王雯，樊守芳.[几种经典的最短路径算法比较分析](http://kns.cnki.net/kcms/detail/detail.aspx?filename=CFXB201812019&dbcode=CJFQ&dbname=CJFDTEMP&v=" \t "http://kns.cnki.net/kcms/detail/frame/kcmstarget)[J].赤峰学院学报(自然科学版), 2018(12)

**致谢**

这篇论文的结束，也意味着大学生活临近了末尾。在此感谢大学三年中，曾教导过我的老师，从他们那里我学到的不仅是知识还有求索问题的方法。

感谢杜安红老师对我的论文的耐心指导，老师虽然平时严苛，其实对学生有着宽容和关怀的心，希望我们学以致力，苛己向上。

也感谢现在实习工作中，指导我的两位主程，他们给了我一些良好的建议，以及更多的时间去编写论文。