《二维路径规划问题研究》总结报告

小组成员：

杨晟　　2014310588
张惜今　2014310589
余旻婧　2014310625

背景介绍：

路径规划(motion-planning)问题一直是计算几何领域着重研究的问题，随着计算机科学的发展和人们对经济利润问题的日益关注，如何得到在有障碍情况下的最短路径问题逐渐成为研究热点。现实生活中的很多应用，如电路布线，管道铺设，道路规划，路线设计及机器人运动等都需要考虑在有障碍情况下的最短路径问题。因此，障碍最短路径问题不仅具有理论意义还有一定的应用价值。

在《计算几何》课程中，参考往年师兄师姐的选题可看出，有相当数量的小组围绕此问题展开了分析和实现工作。区别在于移动的对象和障碍物种类不同，有针对点的路径规划也有针对凸多边形的路径规划，并且考虑的奇异（退化）情况各有不同。结合课程要求、时间精力以及创新能力，我们将预期解决的问题定义如下：

问题定义与限定：

问题：在平面中指定起点p_s与终点p_d和M个简单多边形障碍物$P_1, P_2, P_3, \dots, P_m$，$N$个圆形障碍物$R_1, R_2, R_3, \dots, R_n$，计算$p_s$与$p_d$之间的最短路径。

限定：

① 输入的多边形障碍物不含有线段和类线段型（无宽度）结构。、

② 若多边形产生相交，相交部分的面积应大于 0，即多边形不可相交于若干条线或点。

③ 多边形和圆之间不相交。

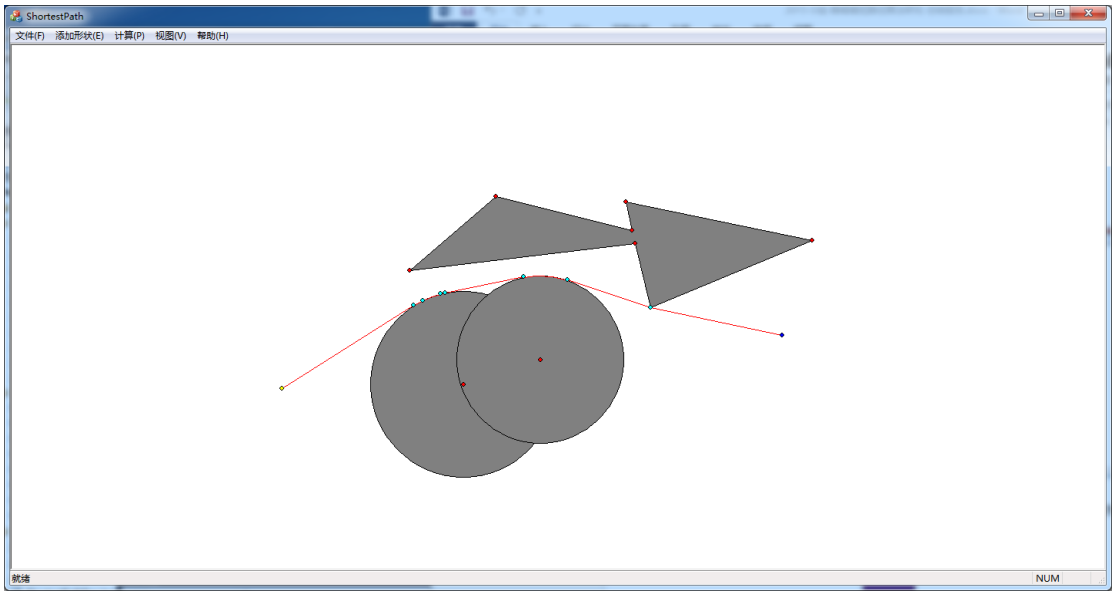

图 1 问题描述

给出如上限定的原因是：当出现类线段型结构时，构造一个可见性图需要引入额外的特殊情形讨论，如对于图 2：

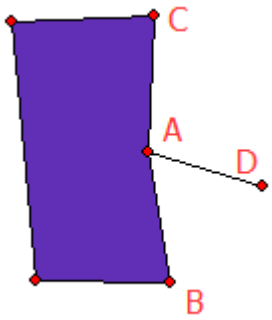

图 2 含类线段型结构的多边形

对于路径 B->A->?，该路径的下一个拐点应该为角 DAB 范围内的某一点，而路径 C->A->?的下一个拐点应该为角 DAC 范围内的某一点。即：经过 A 的路径可达的下一拐点将由其上一拐点所决定。而在实际问题中，完全可以在 A 的ε邻域增加一个多边形顶点（ε足够小），得到问题的近似精确解。所以我们统一将这种带类线段型结构的多边形“归为”非简单多边形，不处理该情况。

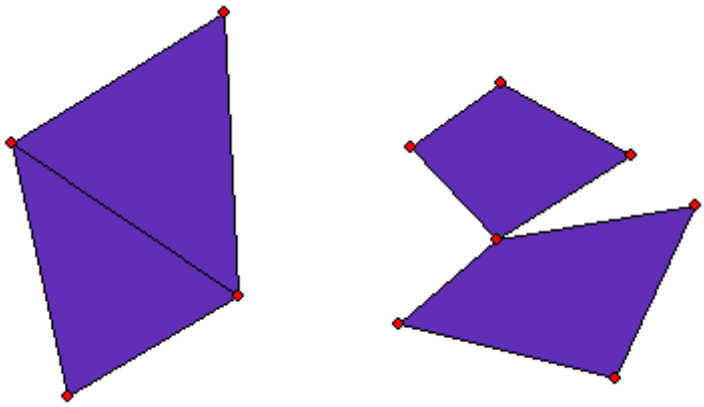

图 3 多边形相交面积为 0 的情形

此外，如图 3，若多个多边形相交面积为 0，也会繁琐化代码逻辑结构并降低性能。在实际问题中，若规定重叠边（点）不可通过，可额外在预处理时加入判断，将两个多边形合并为一个；若规定重叠边（点）可通过，可通过微调点的坐标形成一条宽ε的“通道”（ε足够小），得到问题的近似解，在本程序中，我们将忽略对该情况的处理，并且对用户输入加以限定防止该情况的出现。

相关工作：

障碍最短路径曾是计算几何领域中一个相对热门的问题，但是由于精确算法与简单、高效的应用需求相悖（多想路不如早走路）。所以在系统与游戏应用中人们更多地采用了非精确的启发式算法以期快速获得近似最优解。

启发式算法如 A*，公式表示为$f(\mathrm{n}) = g(\mathrm{n}) + h(\mathrm{n})$，$f$为n从初始点到目标点的估价函数，$g$为从初始点到n的实际代价，$h$为n到目标点的估计代价，故其核心在于设计一个估价函数$h(p_a, p_b)$，估计由p_a到p_b的最短距离，所设计的h函数应在满足估计值小于等于真实值的前提下尽可能地接近于真实值。在设计好了估价函数h以后，程序将起点加入一个处理列表，并每次取出处理列表中f值最小的节点，计算与它直接相邻的点的f值并加入或更新处理列表（列表中不存在则插入，若相邻点已在处理列表中且新计算的f值更小，表示这是一条比之前结果更好的路径，此时将更新该点的f和g值，并重新标记该点的父亲节点），直至将终点加入处理列表（直接输出结果）或处理列表为空（查找失败）。

对于仅包含多边形的最短路径问题，精确算法有两种基本思路：一种是通过区域分割，不断排除不可能区域并最终构造出最短路径地图；另一种是构造可见性图，该图满足：图中两点拥有一条边当且仅当这两点对应的障碍地图中的点可直达，通过将 2D 地图模型抽象成一个可见性图，我们可以直接使用图论中的最短路径问题求得精确解。

构造纯多边形障碍物的可见性图的算法有很多种，我们实现了其中一种经典的旋转扫描线算法解决纯多边形障碍物的路径问题。在纯多边形障碍物下，两点之间直线最短，如果存在障碍物，则最短路径应“贴合”障碍物（可以直观想象一个橡皮筋抱着障碍物直至障碍物的某个顶点与端点直达），暴力遍历两点的可见性可以达到$O(n^3)$的时间复杂度，但没有利用地图中顶点与障碍边的密切关系。所以旋转扫描算法进行加速的核心思想为，对于可见性图中的每一个顶点，遍历生成可见性图的边时先按照极角顺序，同极角则由近及远将其他点排序，并且使用一个 BST 存储当前状态下的遮挡边用于快速检查。该算法的复杂度为$O(n^2 \log n)$，n为障碍物顶点的数目。该算法在之后有很多改进，如使用 Funnel Sequence 漏斗的分割和合并将复杂度降至$O(n \log n + E)$，E为可见性图的边数。

对于仅包含圆障碍物的最短路径问题，基本思路是判断点之间是否可见：若不可见，则应该绕行。根据 Craggs 定理：若可行区域的边界是光滑曲面，则最短路径必有下列弧构成——它们或者是空间中的自然最短曲线，或者是可行区域的边界弧，而且组成最短路径的各段弧在连接点处必定相切。我们可以知道，在本问题中，可能组成最优绕行路线的弧包括各离散点到各圆的切线，各圆之间的公切线，以及这些切点之间的弧（优先劣弧），且所有的这些线段和弧都不能与任何圆相交。

所以对应于该类仅含圆障碍物的解决思路是：向点集之中添加原有的离散点到各个圆的切点和圆之间的公切线切点，将所有的这些点作为图的顶点建立图模型。若一对点均不是圆上的点，或者一对点之间的连线是圆的切线，则判断这对点之间的可见性（若连线不与任何圆相交则可见）。如果可见，向图中此两顶点之间添加一条边，边权为连线长度（平面欧氏距离）；若两顶点是同一个圆上，则需判断它们之间的劣弧是否与任何圆相交，若不相交，则向图中此两顶点之间添加一条边，边权为劣弧长度；若相交，则同理判断优弧是否与任何圆相交并视结果添加边。若有n个离散点，m个圆，切线部分的建图时间复杂度为$O(nm^2 + m^3)$；最坏情况下，弧线部分的建图时间复杂度为$O((n+m)^2 \times m^2)$。为了判断每个弧与圆是否相交，只需要判断它与其所在圆相交的圆是否相交。在实际中，圆障碍物之间大量相交的情况并不会出现，与每个圆相交的圆是常数级别,所以可以认为实践中实践复杂度为$O((n+m)^2 \times m)$。

根据上述现有算法，我们可以合并出一个能同时处理存在两种障碍物的算法。大体上来说，同时包含两种障碍物的可见性图是仅含圆障碍物的可见性图与仅含多边形障碍物的可见性图的交。

预处理算法说明：

预处理包含两个关键组成部分：简单多边形判断和多边形合并。

1.1.简单多边形判断：

```
isSimplePolygon()
// judge whether the polygon is Simple
If the number of vertices in polygon = 2
    return false
End if
For each pair of two edges ei and ej in the polygon
    If ei and ej same or truly coincide or truly intersect
        return false
    End if
End For
return true
```

在这里，我们定义：线段“真相交”和“真重合”，真相交：指两个线段相交于线段上一点而非线段端点，即呈“X”型而不是“T”或“┌”型。同理，真重合：指两个线段存在无数个重合点，即一段公共的线段。这两个基本判断可以通过 to-left 测试和点相等判断实现。

1.2.多边形合并：

```
MergeSimplePolygon()
// merge the overlapped polygons, return false when error occurred
Sort vertices of every polygon counter-clockwise
For each pair of two polygons pi and pj in dataset
    If pi and pj intersect
        Merge pi and pj
    Else If a vertex of pi in pj        // j include i
        remove pi
    End If
End For
```

当两个多边形的边出现相交时两个多边形即相交，此时记录产生的交点。获取所有可能出现的交点后，在两个多边形的顶点中选取一个交点并沿其中一个所在的多边形行进，收集顶点。每当遇到交点时，便按另一个多边形继续行进，收集顶点，直至最后回到起点位置。

当两个多边形的边没有相交时，还需要判断是否为重合关系，此时只需要取出一多边形的一个顶点，判断其是否在另一多边形内。如果在，则二者为重合关系；反之不是。

可见性图算法说明：

本算法的总体思想就是建立起点，障碍物上的点与终点之间的可见性图，然后用 Dijkstra 算法求解最短路径。其中可见性图中的点包括起点、终点以及所有的多边形顶点和这些点到所有圆的切点，以及圆之间的公切点。而可见性图中的边包括多边形顶点之间的连线、顶点到圆的切线、圆之间的公切线和同一个圆上点之间的弧。且它们不与任何多边形或者圆相交，边权为点之间的欧几里得距离或者弧长。

本算法将首选将各类切点求出来加入点集，然后分别针对多边形和圆求一个候选可见性图，最后把两种可见性图求交。保留在两种可见性图中都可见的边，最后用 Dijkstra 算法求解。

2.1. 可见性图顶点的生成

可见性图的点来源于起点、终点、多边形顶点、切点和公切点五类。

```
getAllPoints(data_polygons, data_circles, data_src, data_dst)
// get all the points of Visibility Graph from input data
data_scatteredpoints ← ∅
For each pair of point and circle (pi, circlei) in [data_polygons, data_src, data_dst]
   and data_circles, respectively.
      add tangent points of (pi, circlei) to data_scatteredpoints
End For
For each pair of two circles (pi, pj) in data_circles
      Add tangent points on common tangent of (pi, pj) to data_scatteredpoints
End for
Add data_src to data_scatteredpoints
Add data_dst to data_scatteredpoints
return (data_polygons, data_scatteredpoints)
```

一般来说，圆外一点引对圆的切线会得到两个切点，两圆公切线数量有 0 到 4 条不等，每条对于每个圆均产生一个切点。所有的切点均由平面几何定理通过向量旋转在O(1)时间内求出。

2.2. 多边形可见性图的构建

本部分借鉴了旋转扫描线算法的实现。不同的是，由于障碍物由圆和多边形构成，所以在构造多边形相关的可见性图时，需要将顶点分为两部分，一部分为多边形上的点，在旋转扫描碰到这些点时，将引起边障碍物的变化；另一部分为孤立点，即起点、终点和圆上的点，扫描线经过这些点时不会引起边障碍物的变化。该部分几个关键算法如下：

```
PolyVisibilityGraph(data_polygons, data_scatteredpoints)
```

```
// get the visibility Graph of polygon block relation
Initialize G = (V, E) where V is the set of all vertices of polygons in data_polygons and
data_scatteredpoints. Let E ← ∅
For each vi in V
    Edgelist ← VisibleVertices(vi, data_polygons, data_scatteredpoints)
    Add all arcs in Edgelist to E
End For
Return G
```

在构造可见性图的过程中，需要逐个确定每个点的可见点，以形成可见图的边。

```
VisibleVertices(vi, data_polygons, data_scatteredpoints)
// get all the visible vertices of vi under the block of polygons
Sort the vertices according to the counter-clockwise angle that the halfline from vi to to each
vertex makes with the positive x-axis. Vertices closer to vi should come before vertices farther
from vi. form the sorted list W
Let ρ be the half-line parallel to the positive x-axis starting at vi. Find the obstacle edges that
are properly intersected by ρ, and store them in a BST T in the order in which they are
intersected by ρ.
Edgelist ← ∅
For each wi in W
    If i > 1 and Visible(vi, wi-1, wi, T) or i = 1 and Visible(vi, -1, wi, T)
        Add arc(vi, wi) into E
    End If
    If vi is a vertex in polygons
        Insert into T the obstacle edges incident to wi that lie on the counter-clockwise side
        of the half-line from vi to wi.
        Delete from T the obstacle edges incident to wi that lie on the clockwise side of the
        half-line from vi to wi.
    End If
End for
return Edgelist
```

按照极角逆时针排序后并初始化遮挡边后，以该扫描顺序遍历点，每遇到一个多边形上的点时更新遮挡边表。当遮挡边以 BST 结构存储时，可保证数据的快速查找、插入和删除，这就是旋转扫描算法将单点可见性复杂度从$O(n^2)$降到$O(n \log n)$的关键。

```
Visible(vi, wi-1, wi, T)
// check vi and wi is visible under the block of T
If segment viwi intersects the interior of the obstacle of which wi is a vertex, locally at wi
    return false
Else if wi-1 = -1 or wi-1 is not on the segment viwi
    e ← leftmost leaf in T
    if e exists and viwi intersects e
        return false
    else
```

```
            return true
        end if
    Else if wi-1 is not visible
        return false
    Else if wi-1 and wi in same polygon and wi-1wi in the polygon
        return false
    Else
        Search in T for an edge e that intersects wi-1wi
        If e exists
            return false
        Else
            return true
        End if
    End if
```

整个 Visible 判断中一共分为 3 步：

第一步是判断v_iw_i是否为从w_i所在多边形内部穿出交于w_i的线段，这个判断条件较为复杂但不可或缺，包含三个子判断：

①w_i所在多边形的边是否与v_iw_i为“真相交”关系，一旦有这样一个真相交关系，表示v_iw_i途中有一条边挡住了两者。

②当v_i和w_i在同一多边形中时，判断v_iw_i是否在多边形内部。

③v_iw_i上是否有多边形的一个顶点，且这个顶点满足与其相邻的两个顶点连成的两个边分居于v_iw_i异侧，挡住去路。

在第一步判断之后，在本限定范围中已经消去了所有的奇异情况。对于一个新的极角值上的第一个点，就只需要判断 BST 中的第一条边是否与v_iw_i为“真相交”了，而对于同一极角值上存在的较远的点，则需要取出 BST 中所有的边，判断其是否与$w_{i-1}w_i$ “真相交”，当w_{i-1}, w_i属于同一多边形时，也同样还需要判断$w_{i-1}w_i$是否在多边形内部。

同之前的讨论，该算法的时间复杂度为$O(n^2 \log n)$。但n为输入的总顶点个数（多边形顶点+之前生成的切点总数）。

2.3. 圆可见性图的构建与合并

实现时可以借助之前多边形可见性图的结果简化边的判断数量。

```
VisibilityGraph(GPoly, data_circles)
// get the visibility Graph
G ← GPoly
For each arc (pi, pj) in G
    If pi or pj on circle but not be tangent point of pipj of the circle
        remove (pi, pj) from G
```

```
        Else if pipj intersects a circle
            remove (pi, pj) from G
        End If
    Add all circle arc to G
    return G
```

对于多边形可见性图产生的结果，因为多边形顶点连向一个圆的非切点是没有意义的（除非该点是起点/终点，不然会有更短的路径），故对于含圆上点的多边形可见性图中的边，仅保留不与任何圆相交的切点边。

上一过程去掉了同圆上的弧，需要重新加入可见性图。接下来，对每个圆上的所有切点，枚举每对切点之间所形成的弧，判断其与跟其所在圆相交的所有圆是否相交。此处判断使用三分查找圆心角寻找相交位置的方法，在要求精度固定的情况下，每个弧与一个圆的一次判断可以认为是常数O(1)时间。将所有不与其他圆相交的弧（劣弧优先于优弧）直接加入最终的可见性图。对于总共有n个多边形点，m个圆的输入，该步处理的时间复杂度为$O(nm^2 + mn^2 + m^3)$

2.4. 算法分析与改进点

提出的算法作为一种尝试，其代价主要来源于将大量的离散点输入了多边形可见性图，导致多边形可见性图处理了大量的切点。而这些切点又在合并时被大量的删除，因为它们并不是该多边形对应于该圆的切点。

但是，如果将对应的多边形——圆点对送入多边形旋转扫描算法，则无法有效的利用扫描算法的特性进行加速，最后表现的n的输入规模仍然与之前相同，时间复杂度仍然为$O(n^2 \log n)$。所以本算法适用于多边形较多，圆较少的输入。当输入的圆较多时，可以使用暴力算法而非旋转扫描直接遍历。

开发与运行环境：

开发环境：Windows 7 (64bit) + Visual Studio 2010

依赖的第三方库：(MSVC)

数据格式：

为了调试、展示以及编辑测例，我们定义了障碍物的文件格式，障碍物文件按文本文件存储，由于 MFC 界面的关系，输入的点坐标为整形，实际上算法本身可以处理浮点数据。文本格式如下（数据中不包含注释）：

```
1213 183                    // 起点坐标
1309 204                    // 终点坐标
2                           // 障碍物个数
1 5                         // 1-多边形；5 个顶点
711 204
876 291
596 521
275 396
270 327
2                           // 2-圆；圆心和圆上一点（不需要指定顶点个数）
493 131
541 129
```

运行情况说明：

我们使用了一些测例来测试我们的算法，比如：

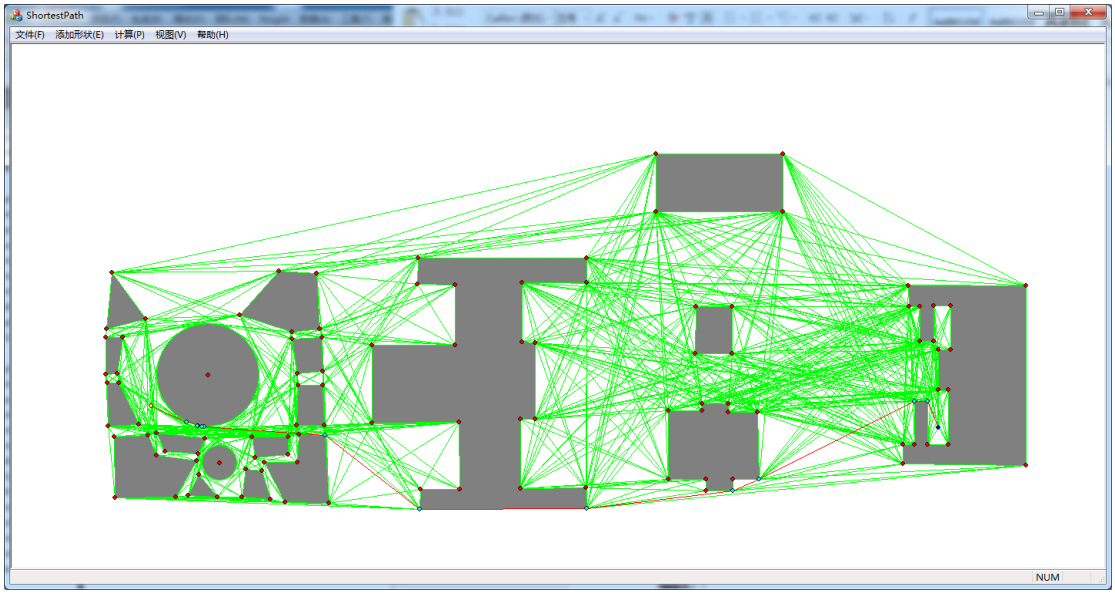

图 4 2 个圆，17 个多边形，111 个多边形顶点，耗时 0.375 秒

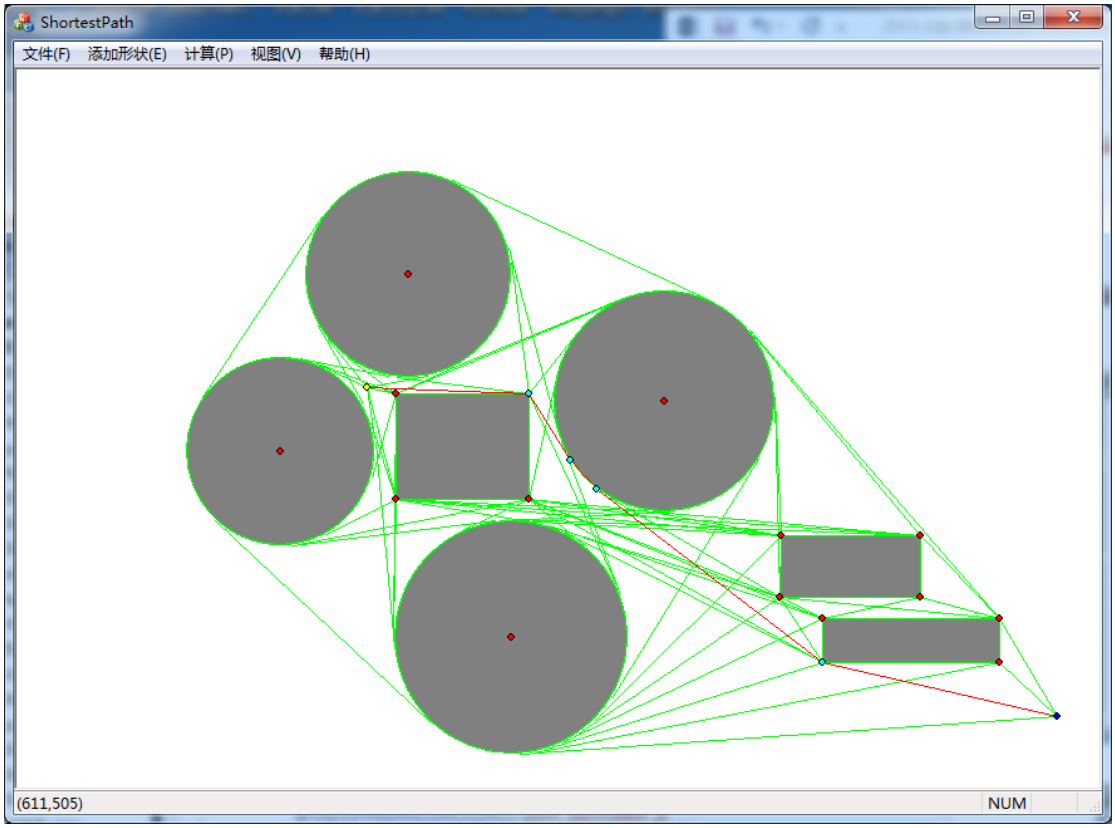

图 5 4 个圆，3 个多边形，12 个多边形顶点，耗时 0.019 秒

（附）界面使用指南：

界面包含菜单栏，主界面和状态栏。状态栏会实时显示当前的程序状态。

编辑类：

***CTRL+Q*：插入多边形**　　***CTRL+W*：插入圆**

***CTRL+E*：撤销上一个点**　　***CTRL+R*：撤销上一个形状**

***CTRL+T*：设置起点**　　***CTRL+Y*：设置终点**

在插入多边形时，每左击一下将会定义一个多边形顶点，右击一下将会完成多边形（将最后一个顶点与第一个在界面上连接）

在插入圆时，左击第一下将会定义圆心，左击或右击第二下定义圆上一点后将会完成圆形障碍物的输入。

文档类：

***CTRL+N*：清空输入**　　***CTRL+O*：打开数据集**

***CTRL+S*：保存数据集**

计算类：

检查形状：将会挨个分别检查输入的形状是否符合要求，主要用于检查某个形状是不是简单多边形且不含有类线段结构。出错时将会告知哪一个形状不符合要求。

合并形状：首先完成检查形状动作，如果无异常将合并有交的多边形，如果相交多边形

存在面积为 0 的相交部分（线相交和点相交），将会报错。

显示可见性图：自动完成上述两步动作并求可见性图和最短路径后，显示可见性图。再次点击将隐藏。

显示最短路径：自动完成上述所有动作后，显示最短路径，再次点击将隐藏。

（附）代码结构说明：

MFC 自动生成的文件（负责人-杨晟）：

MFC 自动生成的类不做介绍，CMFCDoc 类的序列化未被使用，文件的读写，界面展示与 UI 整合在了 CMFCView 类中。

GraphBuilder（负责人-张惜今）：

该类的主要作用是接受输入，生成可见性图并使用 dijkstra 求最短路径。依次调用 load()，addCirclePoints()，buildGraph()和 shortestPath()即可获取结果。其中 nodes 变量为可见性图的顶点，edges 为图中存在的边，自定义结构 PID 中存储的是该边另一顶点和该边长度，vert 中的 originID 即该顶点属于原输入的第几个多边形/圆。

Preprocess（负责人-余旻婧）：

该类的主要作用是完成简单多边形输入检测和多边形合并，isSimplePolygon 对类下的第 Pos 个多边形/圆进行判断，preProcess 为合并输入，结果存放在类下 data_points 和 data_polytype 变量中。

VisiGraph（负责人-杨晟）：

该类修改了现有的旋转扫描算法使其适用于本题含圆的处理，依次调用 Init()，buildVisGraph()后可从 visGraph 中读取在多边形遮挡限定下存在的边。其中 Init 输入一组多边形和一些零散的点。“图邻接矩阵”的编号将按照输入各点的顺序生成。