《计算几何》课程实验总结报告

组员：涂煜　2013213499　　贾晨　2014213501

1. 简介

1.1 选题题目

基于精确的点模式识别和基于 Turning Function 的几何形状相似性判定

1.2 问题简述

几何图样和形状的匹配和分析在各种应用领域中都起着重要的作用，特别是在计算机视觉和模式识别领域中。在一般情况下，我们被给定两个对象 A 和 B，想要知道的是它们之间在多大程度上相似。通常其中的一个对象需要经历一定的转换，例如旋转、缩放等，来尽可能地和另一个对象匹配[1]。

本文通过输入多边形 A 和 B 的顶点信息，判断这两个多边形是否相似，即多边形 A 是否能经过一系列仿射变换（平移、旋转、翻转、错切）后和多边形 B 相重合。几何形状相似性判定的方法有很多，本文主要介绍两种：基于精确的点模式识别的方法和基于 Turning Function 的方法，分别从顶点和边这两个角度来判断两个多边形的相似性。

2. 功能描述

用户可以选择使用何种方法来判断两个多边形是否相似，两种判定方法分别为：基于精确的点模式识别的方法和基于 Turning Function 的方法。

所用的两种方法均首先让用户选择两个存储多边形信息的文本文件作为输入，并在显示界面中央绘制出这两个多边形。系统判定完成后会在界面的左上角输出判定结果：若两多边形相匹配，输出是否需要翻转、需要旋转的角度和缩放的比例；若两多边形不匹配，只提示两多边形不匹配。

同时，在使用基于精确的点模式识别的方法判定多边形相似性时，通过添加顶点闪烁的效果动态显示匹配结果，使两个匹配多边形之间的对应关系更加清晰明了。在使用基于 Turning Function 的方法的时候，用户可以自己设定两多边形匹配判定的阈值，实现不同程度上的相似性判定。

此外，在界面的友好性上，用户可以选择是否显示多边形顶点，来看到不同的展示效果。

3. 数据结构

系统主要设计了如下几种数据结构，方便数据的存储和计算等。

- **存储点的坐标**

```
//直角坐标表示下的点
struct BASEPOINT
{
    double x, y;        //分别为点的 x 坐标和 y 坐标值
};

//极坐标表示下的点
struct CPPoint
{
    double  ρ;          //极边
    double angle;       //极角
    int No;             //点的编号
};
```

- **存储读入的简单多边形**

```
struct CPolygon
{
public:
    CPolygon():bOrientation(true),pPts(NULL),nNum(0){}
    BASEPOINT* pPts;  //按序存储的顶点坐标集
    int nNum;               //多边形顶点个数
    bool bOrientation;      //顶点遍历方向：true(顺) false(逆)
};
```

- **存储分割出的多边形轮廓线段**

```
struct PolySegment
{
    BASEPOINT *pts;                 //线段中包含的顶点集合
    int num;                        //线段中包含的顶点数目
    double endCornerAngle;          //该线段结束角的逆时针转角
    double segLength;               //该线段长度
    double *lengths;                //轮廓点间距离
    double *angles;                 //轮廓点的转角信息
    double *curvatures;             //轮廓点对应的曲率
```

```
    int fpNum;                                  //特征点个数
    int *featurePoints;                         //特征点下标数组
    PolyFeaturePointType *featureEncode;//特征点编码
    PolySegments*   poly;                       //该轮廓线所属的多边形

    //构造函数和析构函数由于过于复杂此处省略，只展示上述变量
};
struct PolySegments
{
    PolySegment **segments; //分割的多边形轮廓线段集合
    double perimeter;           //多边形周长
    int num;                    //分割的多边形轮廓线段条数
    int firstIndex;             //0 号轮廓线 0 号点对应的原点中的下标
    double disFac;              //计算曲率时，多边形中点间距的缩放比例

    //构造函数和析构函数由于过于复杂此处省略，只展示上述变量
};
```

- **存储多边形的转角折线图**

```
struct TFL2Map
{
    vector<double> gra;  // 每个折线累计的转角大小
    vector<double> dis;  //每个折线对应的相对长度
    bool isFlip;         //是否进行了翻转
    TFL2Map()            //构造函数
    {
        isFlip = false;
    }
};
```

- **存储匹配结果**

```
struct PolysAdjustParams
{
    bool isNeedFlip;     //是否需要翻转
    double angle;        //旋转角度
    int matchedPos1;     //两多边形匹配时分别对应的顶点的编号
    int matchedPos2;
};
```

4. 算法

4.1 基于精确的点模式识别的方法

当我们需要对两个多边形进行严格的匹配时，可以用基于精确的点

模式识别的方法，该方法要求两多边形顶点个数及拓扑位置严格相同。

在二维空间中，精确的点模式匹配能够很容易地被简化规约为字符串匹配问题[2]，很多论文中都对该方法有所描述。本文在这些方法的基础上做了少许修改，以更适合解决我们的选题。详细算法描述如下：

1. 若两多边形顶点个数不同，无法使用该方法判定，结束该算法；否则继续执行步骤 2~6。
2. 分别计算出多边形 A 和 B 的重心 c_A, c_B。
3. 以点 c_A 为原点，计算出 A 中所有顶点的极坐标，然后对这些极坐标按照字典序排序（首先按极角排序，极角相同的点按极边长度排序），得到序列(Φ_1, r_1),…, (Φ_n, r_n)。
4. 令 u 表示序列(Ψ_1, r_1),…, (Ψ_n, r_n)，其中 $\Psi_i = \Phi_i - \Phi_{(i+1) \bmod n}$。
5. 对多边形 B 用同样的方法，按照步骤 3~4 计算出相应的序列 v。
6. 判断序列 v 是否是序列 u 的一个循环移位，此处使用 KMP（快速匹配）算法判断 v 是否是 uu 的一个子序列。

可以看出，如果算法给出一个肯定的答复，则说明多边形 A 和 B 是相似的。算法中第 3 步的排序过程耗时 O(n log$n$)，其它所有操作均为线性时间复杂度，故该算法的整体时间复杂度为 O(n log$n$)。

4.2 基于 Turning Function 的方法

基于 Turning Function 的方法是一种将多边形按照其内角和边长转化为 Turning Function 曲线，并以 L_2 范式作为衡量标准，来计算两个多边形的相似程度的方法。

4.2.1 多边形轮廓线分割

基于 Turning Function 的方法考虑的匹配单元为相对平滑的轮廓线段，故需要多边形轮廓线进行区间分割，并构建以平滑线段为基本元素的数据集合。本文认为满足以下两个条件的轮廓线段是合法的轮廓线段：

a) 分割出轮廓线段要么是直线段要么是单弧曲线（凹弧或凸弧）要么是凹弧凸弧交替构成的 S 型曲线；

b) 若相邻的三个轮廓线段为“单弧曲线直线单弧曲线”或“直线单弧

曲线直线”时，且直线段长度小于相邻单弧曲线中弧长最短的那个时，且将相邻的轮廓线段合并成一条后累计转角小于某一阈值（本文取 66 度）时，那么最后的分割结果应为合并后的轮廓线段；

对多边形顶点进行分割时第一个需要考虑的因素为转角，如图 4.1 所示：

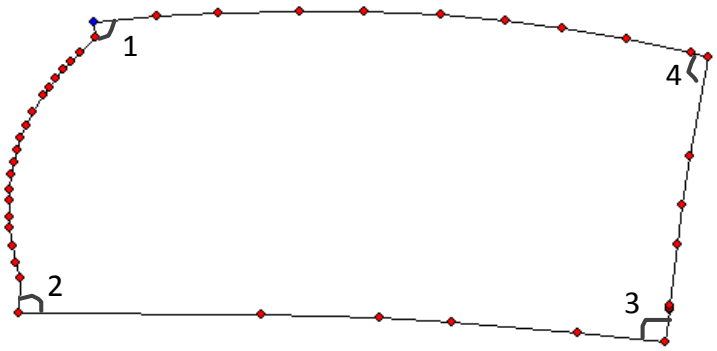
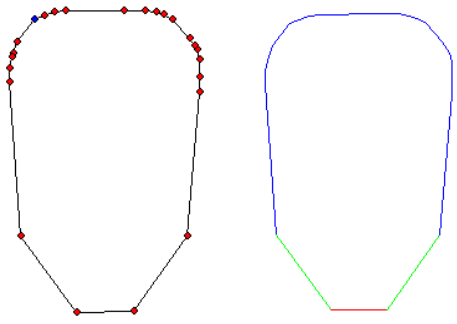


图 4.1 多边形中大转角位置示意图

图 1 中标注的 4 个转角均为多边形中较大的转角，同时这四个转角位置的轮廓点也是四条轮廓线段的端点。本文设定相邻两个向量转角超过 30° 的顶点为轮廓线段分割点，记由此分割出的轮廓线段集合为 Seg_{big}。

单靠转角并不能完美的分割出轮廓线中的曲线段和直线段部分，且当多边形转角过渡较为连续时，通过转角分割出的轮廓线段会出现既包含直线段也包含曲线段的情况，如图 4.2 所示，由于多边形中靠上部分的点集间的向量转角过度比较平滑，故仅通过转角提取的轮廓线段会出现非法情况，既分割后的轮廓线段同时包含直线段和曲线段：



(a)多边形点集数据　　(b)只通过转角分割

图 4.2　转角分割示意图

轮廓线分割第二个要考虑的因素为曲率，使用近似曲率对通过角度分割得到的轮廓线段进行二次细分，用以提取其中的直线段和曲线段。近似曲率的计算方法如下：

$$p_i\text{点处近似曲率} = \frac{p_i\text{点位置处的转角}}{p_i\text{点所在两条线段的长度和}}$$

当两多边形拓扑相同而大小差异较大时，由于拓扑相同故相对转角相同，但由于大小不同故顶点间距离会出现差异，所以相同拓扑位置的两个顶点处的近似曲率值也会出现差异。为了消除多边形大小对近似曲率计算的影响，在对多边形进行轮廓线分割之前，先将多边形缩放至一个半径为 10 的外接圆内，这样使得多边形的点间距不会出现过大幅度的变化，也保证了近似曲率的有效性。本文取曲线段近似曲率阈值 T_{Cur}为3，直线段近似曲率阈值T_{Line}为1.5，当点p_i位置的曲率$c_i > T_{Cur}$时判定点p_i属于曲线；当$c_i < T_{Line}$时判定点p_i属于直线；否则判定点p_i归属于距离当前已判定出的直线段或曲线段中最近的那条。

二次细分虽然能够提取Seg_{big}中直线和曲线部分，但往往会分割得过于细致，如图 4.3 所示，该多边形右侧凹弧在匹配过程中应被当作一条完整的曲线，但这里却分为 2 段曲线和 3 段直线。一般出现此类问题时往往是由于曲线整体曲率较小且相邻的直线长度过短，将直线归入相邻曲线后整体曲线的累积转角也同样较小。

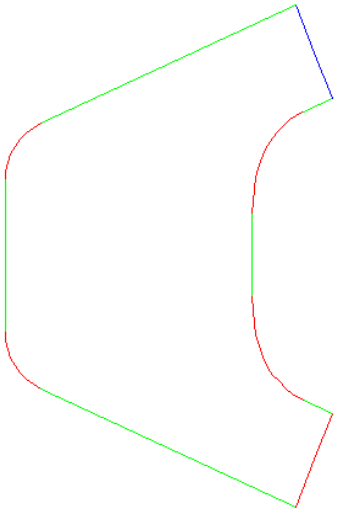


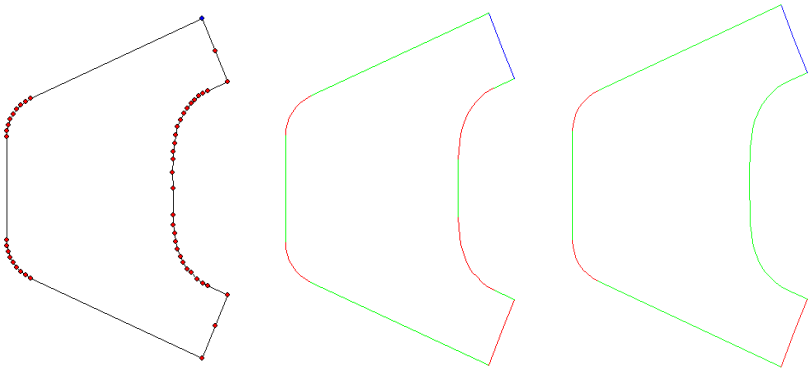
图 4.3　二次细分过度示意图

因此，轮廓线分割的最后一步也就是对二次细分得到的直线段和曲线段进行合并。本文设定合并后累积转角阈值为$T_{mergedCur} = 66°$，当合并后累积转角小于该阈值时认为需要将细分出的相邻直线和曲线进行合并。

至此，我们可以得到多边形轮廓线分割的完整的算法流程如下所述：

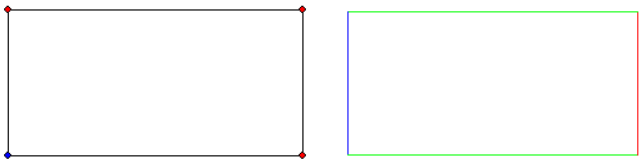
1. 计算出多边形的所有边长、顶点处的转角及近似曲率；
2. 根据转角值提取多边形的大区间线段集合Seg_{big}；
3. 根据近似曲率对集合Seg_{big}进行二次细分，得到集合Seg_{small}；
4. 遍历集合Seg_{small}中的所有线段，判断是否出现了“曲线直线曲线”或“直线曲线直线”类的状态，并根据阈值尝试合并。

上述步骤 4 执行结束后，得到的合并后的线段集合即为将多边形合理分割后所得的轮廓线段集合，根据该集合即可进行接下来的相似性判定操作。下列各图展示的是一些绘制出顶点的多边形及对它们进行轮廓线分割后的结果：



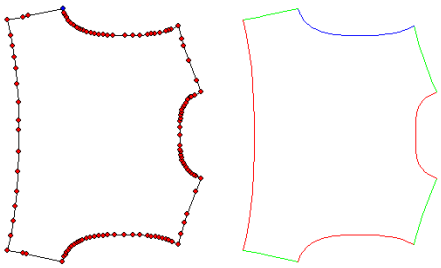
(a)多边形点集数据　(b)二次过度分割结果　(c)合并处理后结果

a. 多边形二次细分及合并后结果



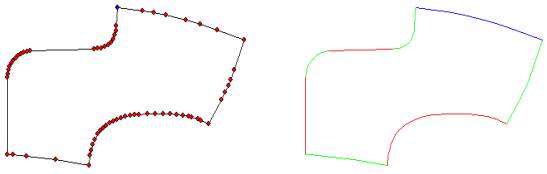
(a)轮廓点数据　(b)分割结果

b. 正常类测试用例



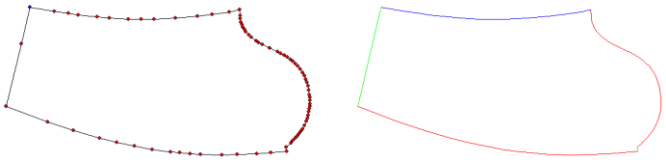
(a)轮廓点数据　(b)分割结果

c. 多凹弧类测试用例



(a)轮廓点数据　(b)分割结果

d. 异形类测试用例



(a)轮廓点数据　　(b)分割结果

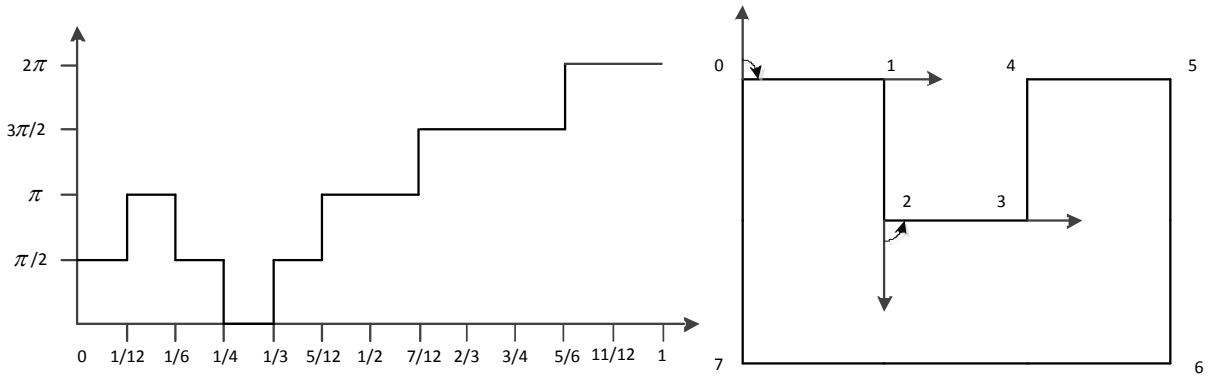
e. S 弧类测试用例

图 4.4 各类多边形及其轮廓线分割结果示意图

4.2.2 多边形相似性判定

基于 Turning Function 的多边形相似性判定方法是一种基于转角折线图和L_2范式距离的多边形匹配算法，其核心思想是将多边形间的匹配转化为转角折线图的最小L_2值匹配[3,4]。

Turning function 曲线是一种将多边形按照其内角和边长转化而来的曲线。图 4.5 所示为多边形按照顺时针顺序生产的曲线，转角折线图中横轴为多边形相对边长，对应到[0,1]区间；纵轴为多边形累计转角，对应到[0,2π]区间。由于将多边形周长统一到了[0,1]区间，故曲线对缩放无关。转角折线图的生成同两多边形的遍历起点有关，同旋转状态无关，故枚举两多边形的遍历起点既可消除旋转对匹配过程的影响；多边形进行一次水平翻转后其遍历方向会反向，但翻转操作并不会改变轮廓点间的转角和长度信息，故逆方向遍历得到的转角折线图即为多边形翻转后的转角折线图。



(a)转角折线图　　(b)原图形

图 4.5　转角折线图生成示例

使用两转角曲线的L_2距离来衡量两多边形的相似度。记$\theta_A(x,i), x \in [0,1]$表示多边形 A 以p_i为顺时针遍历起始点形成的转角曲线，则多边形 A 以p_i为

起始点同多边形 B 以p_j为起始点的L_2距离$d_{A,B}$的计算公式如式（1）所示：

$$\mathrm{d_{A,B}} = \left(\int_0^1 \left(\theta_{\mathrm{A}}(\mathrm{x,i}) - \theta_{\mathrm{B}}(\mathrm{x,j})\right)^2 \mathrm{dx}\right)^{\frac{1}{2}} \quad (1)$$

$d_{A,B} \in [0, +\infty)$, $d_{A,B}$越小则表明多边形 A 和 B 几何拓扑越相似，通过判定$d_{A,B}$是否小于阈值$T_{similar}$即可判定两多边形是否相似，用户可以通过交互界面自行调节阈值，实现不同容忍程度上的相似性判定。

记输入的两个多边形为 A、B，根据 4.2.1 中分割得到的轮廓线段集合，可以得到基于 Turning Function 的多边形相似性判定算法如下所述：

1. 两层循环遍历多边形 A 和 B 的所有轮廓线段的端点，分别计算以每个顶点对为遍历起点得到的两多边形的转角折线图，并计算它们间的L_2距离$d_{A,B}$；
2. 对 B 进行翻转，然后再执行步骤 1 操作；
3. 获取步骤 1~2 中计算出的所有$d_{A,B}$中的最小值，若$d_{A,B} < T_{similar}$，两多边形相似，否则，两多边形不相似。

记多边形顶点个数为 n，分割所得轮廓线条数为 k。由于多边形每个顶点只遍历一次便可生成转角折线图，故计算转角折线图的时间复杂度为$O(n)$。上述算法中，双层循环执行次数为 k^2 次，计算L_2距离耗时 O(1)，故算法整体时间复杂度为$O(n * k^2)$。

5. 实验结果

由于篇幅限制，此处只列举了部分测试用例结果，用于向读者直观展示系统功能及显示界面。详细展示见演示视频。

5.1 基于精确的点模式识别的方法

下列各图分别展示了使用精确的点模式识别的方法判定多边形相似性时，各个测试用例的的输出及显示。

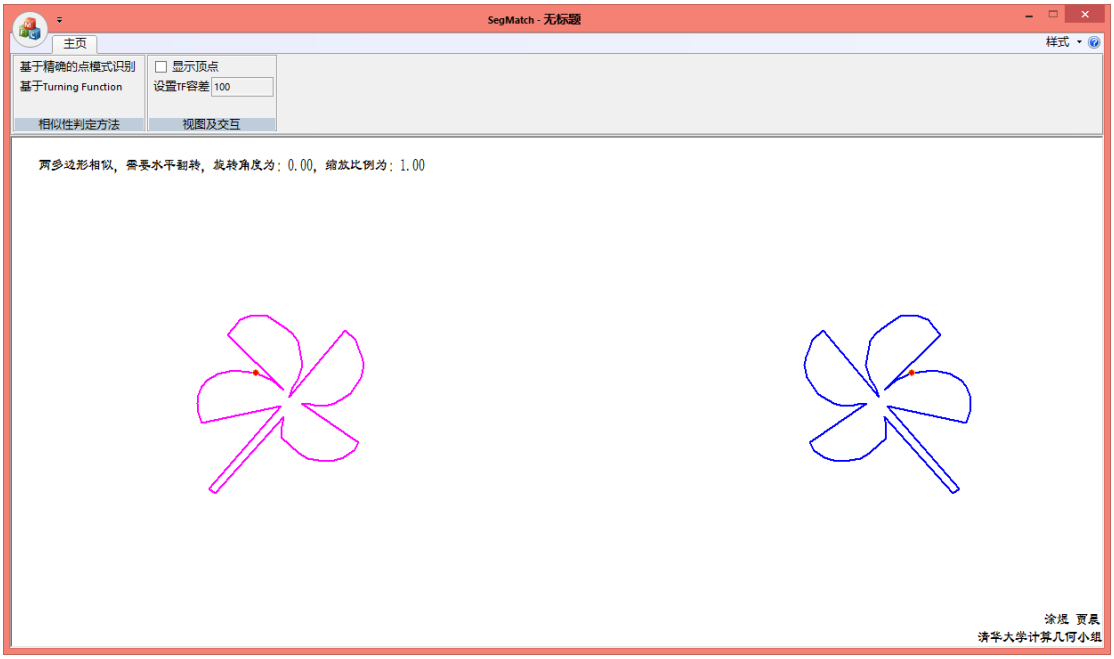


图 5.1 水平翻转后匹配（不显示顶点）

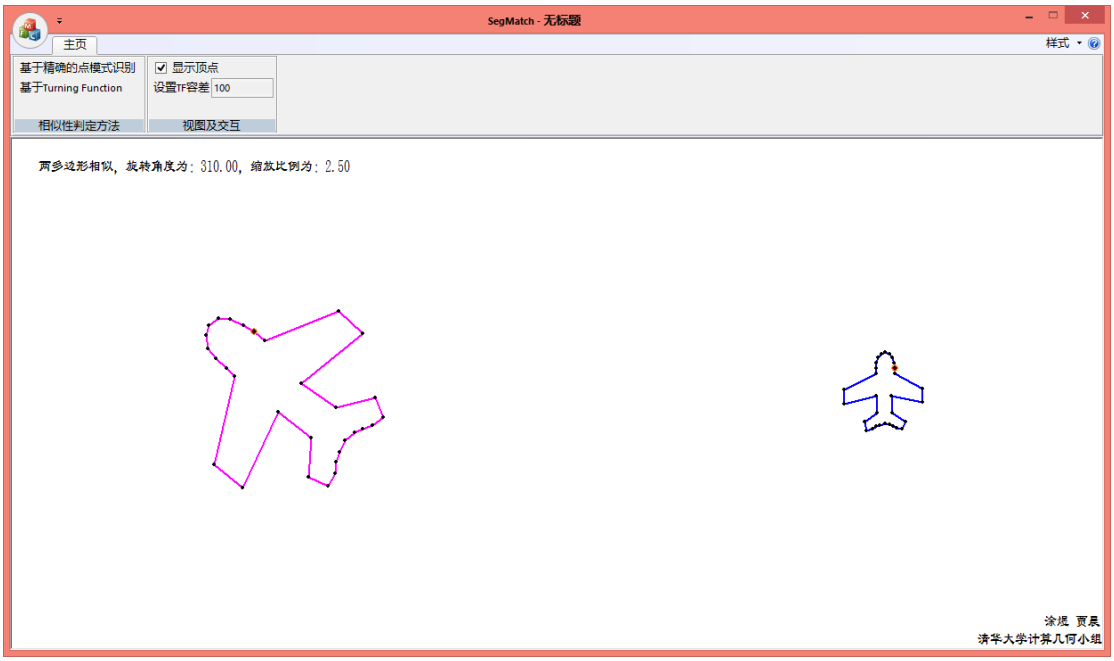


图 5.2 旋转并缩放后匹配（显示顶点）

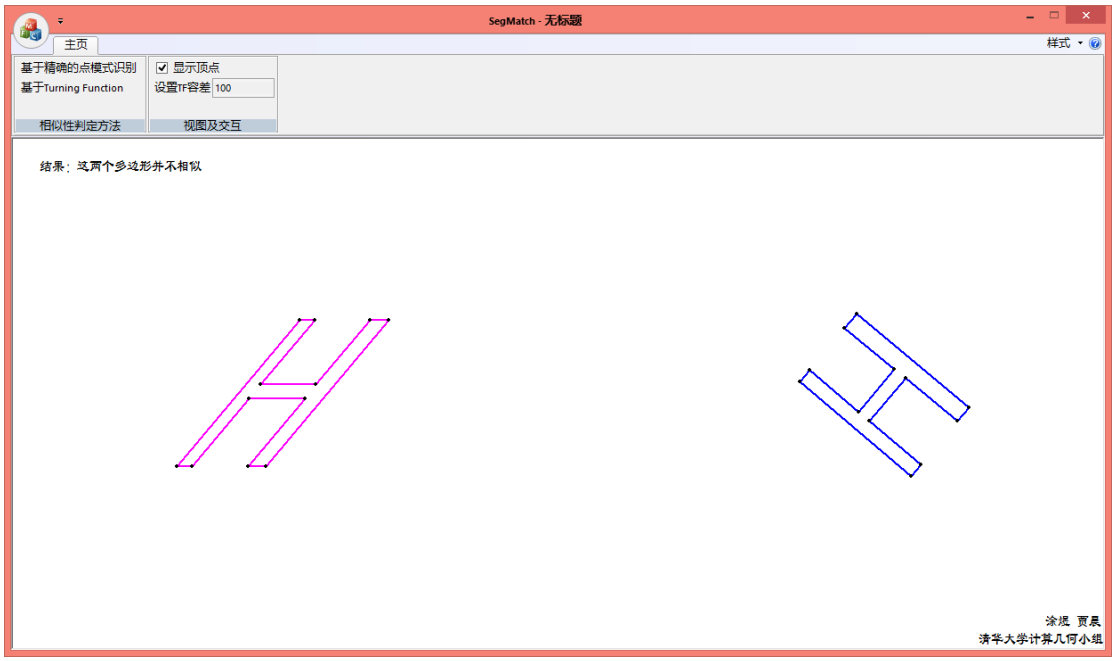


图 5.3 两多边形不相似

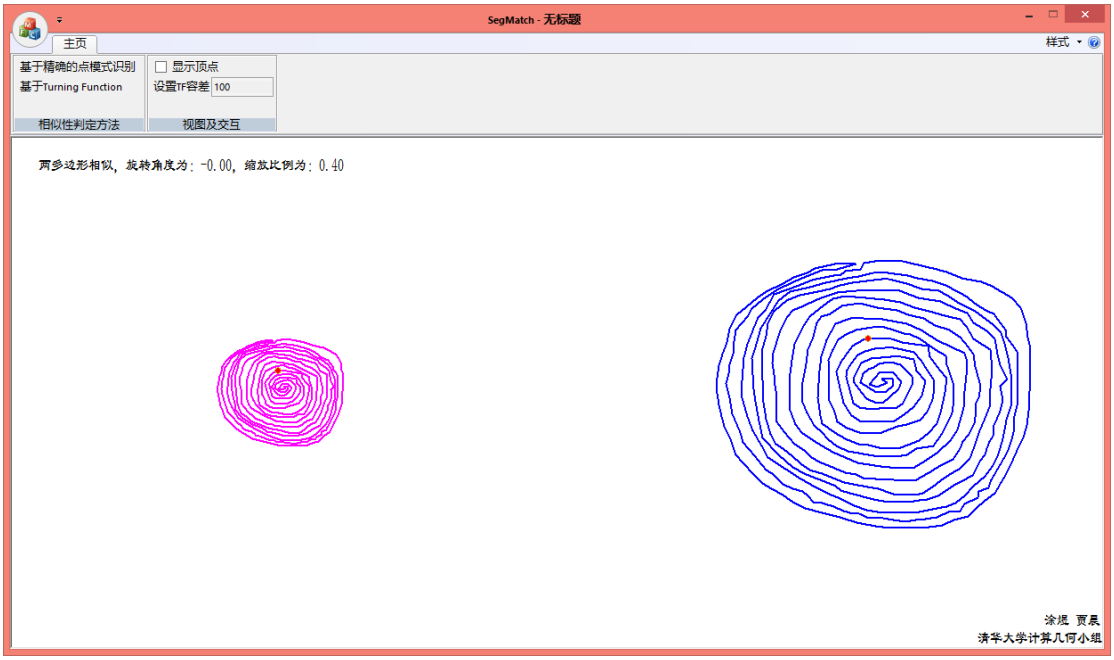


图 5.4 数据量较大时匹配的情况

5.2 基于 Turning Function 的方法

下列各图分别展示了使用精确的点模式识别的方法判定多边形相似性时，各个测试用例的的输出及显示。

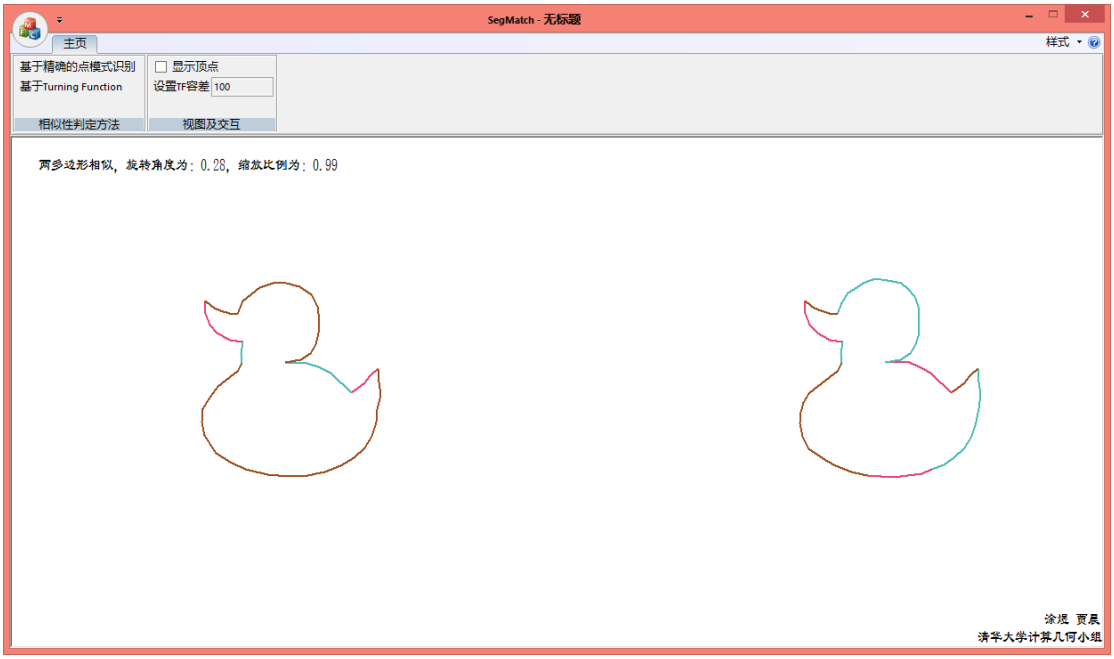


图 5.5 匹配的情况（不显示顶点）

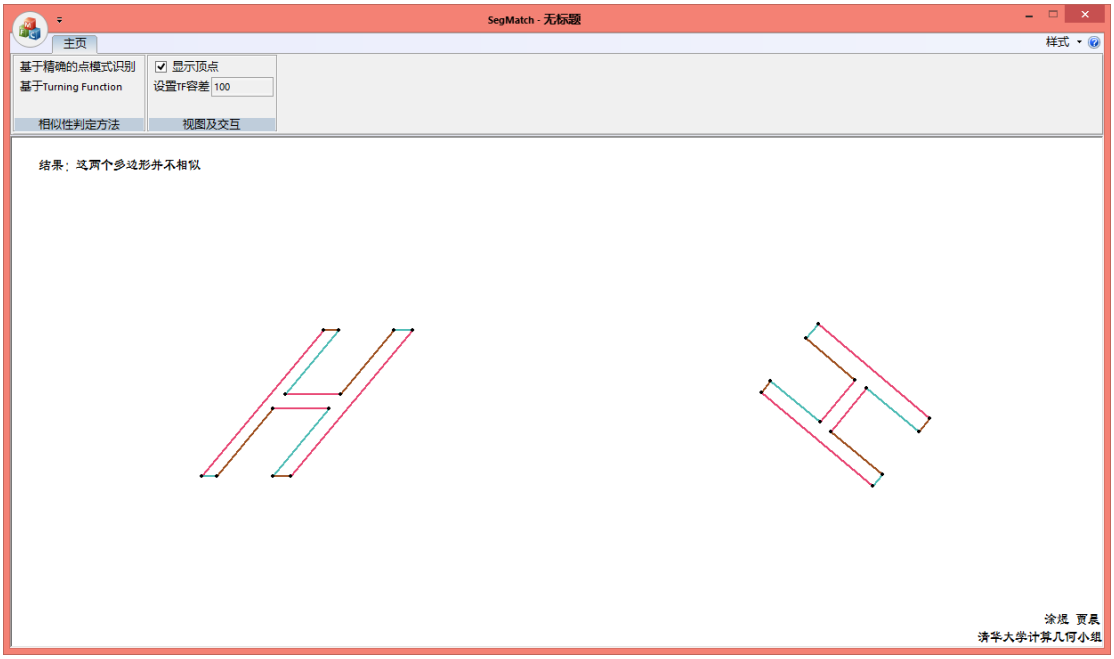


图 5.6 不匹配的情况（显示顶点）

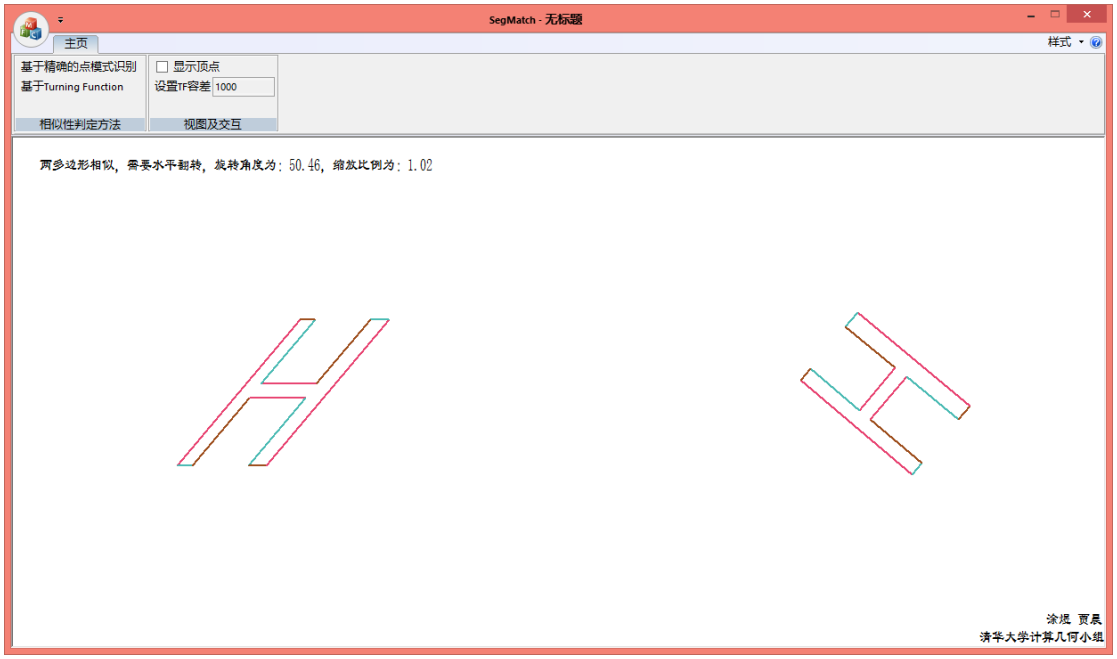


图 5.7 用户调整 TF 容差后匹配的情况

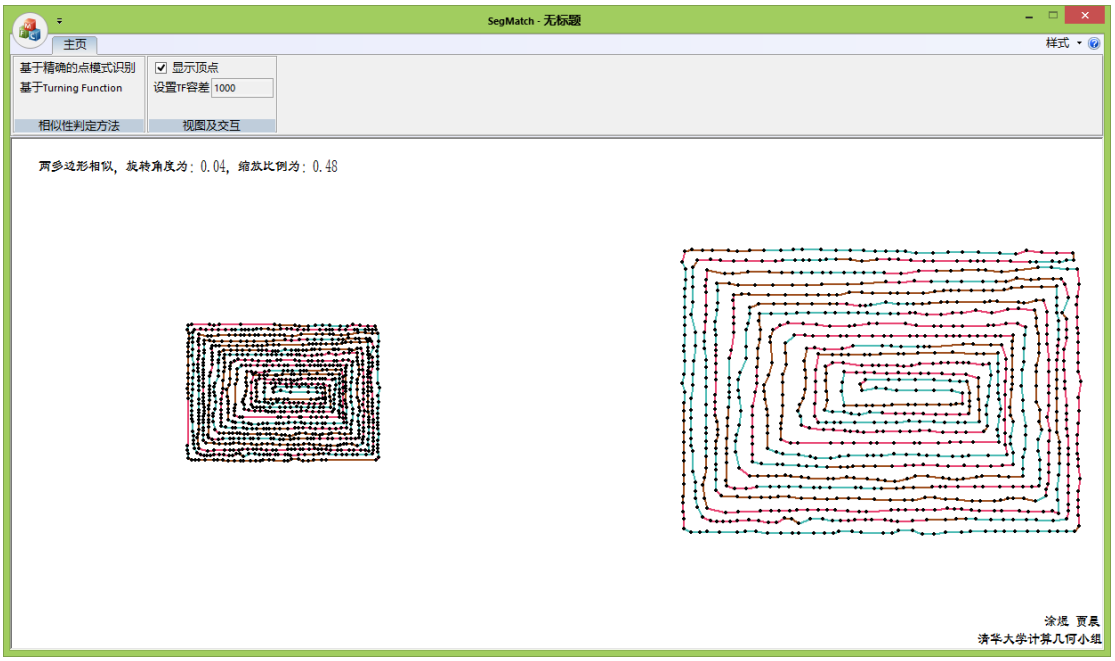


图 5.8 数据量较大时匹配的情况

6. 总结

从确定选题并通过调研后确定算法，然后搭建 SVN 服务器，开始编写系统框架，到编写和完善系统的算法流程，系统可以运行之后的调试、优化，再到最后编写文档，总共经历了一个月的时间，最终顺利完成了计算几何的大实验。

系统从使用多边形顶点匹配和使用多边形轮廓线匹配两个不同的角度出发，

实现了两种判定多边形相似性的算法：基于精确的点模式识别的算法和基于Turning Function 的算法。系统可以与用户交互，让用户能够自行调节使用 Turning Function 方法时的判定阈值，并且使用了动态闪烁点的效果，以及交叉使用不同颜色对轮廓线段着色的方法，将判定结果友好地展示给用户，使得匹配结果更加清晰明了。

目前的系统具有较高的鲁棒性，不会因为异常情况而崩溃。经过优化之后，系统具有较高的吞吐量和运行效率，能够快速处理由上千规模的顶点组成的多边形间的相似性判定。当然，系统所用算法也存在不足之处，在基于 Turning Function 的方法中，可以再分割完多边形轮廓线后进一步提取特征点并进行编码，然后使用 LCS 匹配（最长公共子串）来对 k^2 的循环遍历过程加速。

总之，通过本次课程实验，我们深入了解学习了多边形匹配的算法和思想，并将其应用到了实践中。我们两位组员在实验过程中通力合作，系统编写过程中我们分工明确、各司其职，共同克服和解决了一些困难和问题，最终完成了大实验。

最后，感谢老师在课堂上耐心细致的讲解，以及同学们在课堂上及课下的相互帮助。我们在课程中学到了很多，在实践中应用了很多，收获了很多知识。

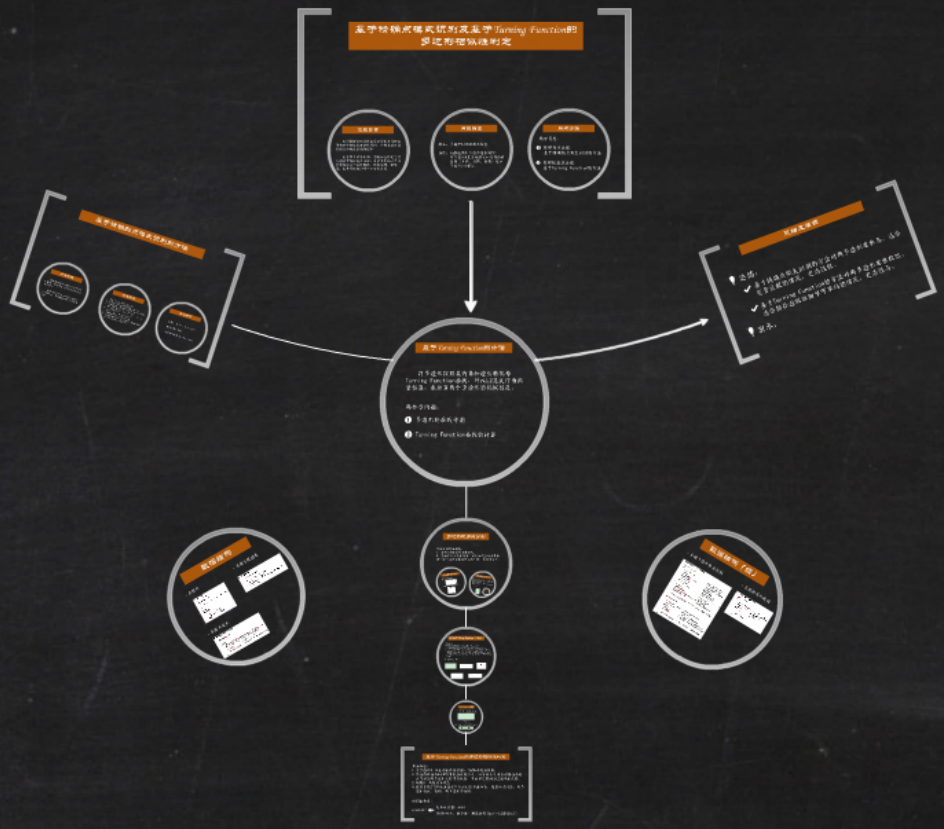
7. 引用

[1] J-R.Sack, J.Urrutia. Handbook of Computational Geometry[M]. Holland:ELSEVIER, 1999:123.

[2] M.D. Atkinson, An optimal algorithm for geometrical congruence, J. Algorithms 8 (1987), 159-172.

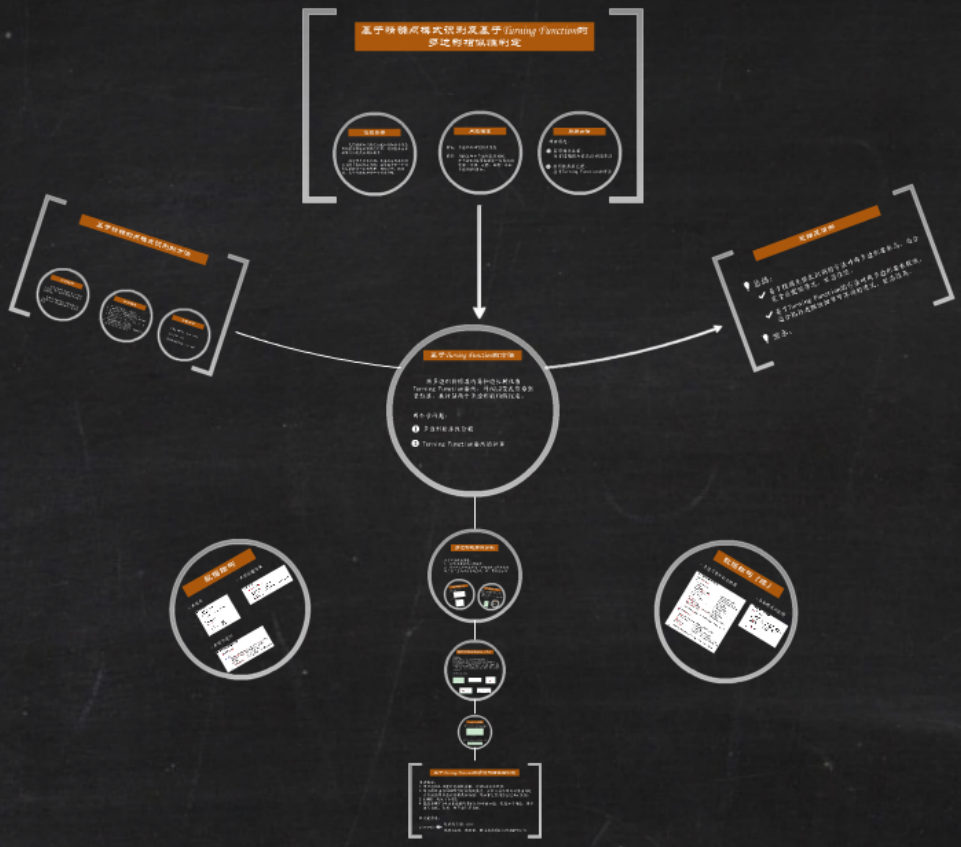
[3] Arkin E M, Chew L P, Huttenlocher D P, et al. An efficiently computable metric for comparing polygonal shapes[C]. Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, 1990: 129-137.

[4] Philip M. Polygon Matching:Measuring similarity between polygons[M/OL]. 2010 [2010-7-19]. http://www.student.cs.uwaterloo.ca/~cs763/Projects/phil.pdf.



计算几何课程实验展示

软件学院：涂煜　贾晨



计算几何课程实验展示

软件学院：涂煜　贾晨

基于精确点模式识别及基于*Turning Function*的多边形相似性判定

选题背景

几何图样和形状的匹配和分析在各种应用领域中都起着重要的作用，特别是在计算机视觉和模式识别领域中。

给定两个对象A和B，想要知道的是它们之间在多大程度上相似。通常其中的一个对象需要经历一定的转换，例如旋转、缩放等，来尽可能地和另一个对象匹配。

问题描述

输入：多边形A和B的顶点信息

操作：判断这两个多边形是否相似，即多边形A是否能经过一系列仿射变换（平移、旋转、翻转）后和多边形B相重合。

所用方法

两种角度：

1. 按照顶点匹配：
   基于精确的点模式识别的方法
2. 按照轮廓线匹配：
   基于Turning Function的方法

选题背景

几何图样和形状的匹配和分析在各种应用领域中都起着重要的作用，特别是在计算机视觉和模式识别领域中。

给定两个对象A和B，想要知道的是它们之间在多大程度上相似。通常其中的一个对象需要经历一定的转换，例如旋转、缩放等，来尽可能地和另一个对象匹配。

问题描述

输入：多边形A和B的顶点信息

操作：判断这两个多边形是否相似，即多边形A是否能经过一系列仿射变换（平移、旋转、翻转）后和多边形B相重合。

所用方法

两种角度：

1. 按照顶点匹配：
   基于精确的点模式识别的方法

2. 按照轮廓线匹配：
   基于Turning Function的方法

基于精确的点模式识别的方法

方法说明

当需要对两个多边形进行严格的匹配时，可以用基于精确的点模式识别的方法。

要求两多边形顶点个数及拓扑位置严格相同。可被规约为字符串匹配问题。

算法描述

1. 若A、B顶点数不同，结束算法。
2. 分别计算出多边形A和B的重心Ca，Cb。
3. 以Ca为原点，计算A中顶点极坐标，按字典序排序得序列(Φ1,r1)…(Φn,rn)。
4. 令u表示序列(Ψ1，r1)，…，(Ψn，rn)，其中Ψi =Φi - Φ(i+1) mod n。
5. 对B用同样的方法计算出序列v。
6. 判断v是否是u的一个循环移位：用KMP算法判断v是否是uu的一个子序列。

性能分析

步骤3（排序）：O(n logn)

其它步骤：O(n)

整体时间复杂度：O(n logn)

方法说明

当需要对两个多边形进行严格的匹配时，可以用基于精确的点模式识别的方法。

要求两多边形顶点个数及拓扑位置严格相同。可被规约为字符串匹配问题。

算法描述

1. 若A、B顶点数不同，结束算法。
2. 分别计算出多边形A和B的重心Ca，Cb。
3. 以Ca为原点，计算A中顶点极坐标，按字典序排序得序列(Φ1,r1)…(Φn,rn)。
4. 令u表示序列(Ψ1，r1),…，(Ψn，rn)，其中Ψi =Φi - Φ(i+1) mod n。
5. 对B用同样的方法计算出序列v。
6. 判断v是否是u的一个循环移位：用KMP算法判断v是否是uu的一个子序列。

性能分析

步骤3（排序）：O(n logn)

其它步骤：O(n)

整体时间复杂度：O(n logn)

基于*Turning Function*的方法

将多边形按照其内角和边长转化为Turning Function曲线，并以L2范式作为衡量标准，来计算两个多边形的相似程度。

两个子问题：

1. 多边形轮廓线分割
2. Turning Function曲线的计算

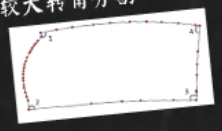
多边形轮廓线分割

合法轮廓线应满足：

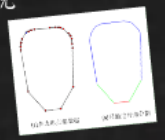
1. 直线/单弧曲线/S型曲线。
2. 若相邻三个轮廓线为“单弧曲线直线单弧曲线”或“直线单弧曲线直线”时，需考虑合并。

通过转角粗分割

- 通过较大转角分割



- 不当情况

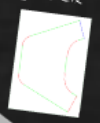


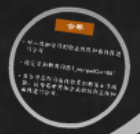
通过曲率二次细分

p点曲率=p点转角/p点所在两线段长度和

- 消除多边形大小对曲率影响：将多边形缩放至半径为10的外接圆内
- 曲线段近似曲率阈值T_Cur=3，直线段近似曲率阈值T_Line=1.5

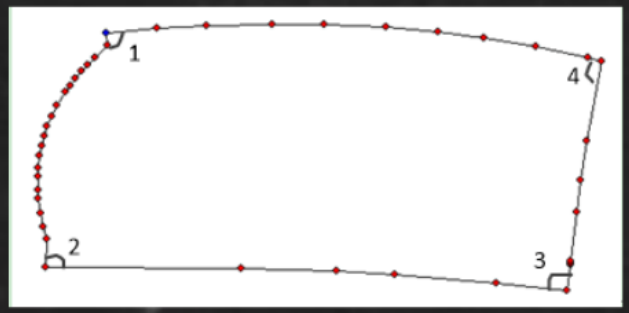
⚠ 细分过度



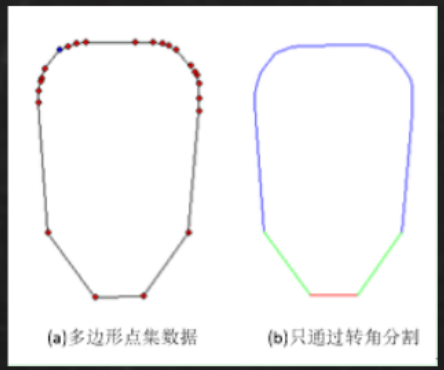


通过转角粗分割

- 通过较大转角分割



- 不当情况

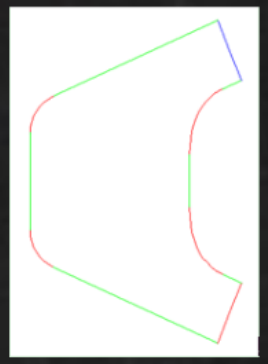


通过曲率二次细分

p点曲率=p点转角/p点所在两线段长度和

- 消除多边形大小对曲率影响：将多边形缩放至半径为10的外接圆内
- 曲线段近似曲率阈值T_Cur=3，直线段近似曲率阈值T_Line=1.5

⚠ 细分过度



合并

- 对二次细分得到的直线段和曲线段进行合并
- 设定累积转角阈值T_mergedCur=66°
- 当合并后所得曲线的累积转角小于阈值，认为需要将细分出的相邻直线和曲线进行合并。

合并

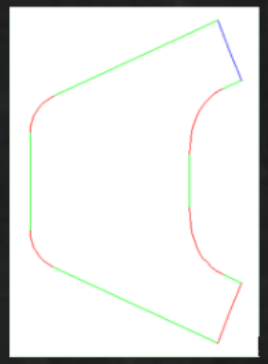
- 对二次细分得到的直线段和曲线段进行合并
- 设定累积转角阈值T_mergedCur=66°
- 当合并后所得曲线的累积转角小于阈值，认为需要将细分出的相邻直线和曲线进行合并。

通过曲率二次细分

p点曲率=p点转角/p点所在两线段长度和

- 消除多边形大小对曲率影响：将多边形缩放至半径为10的外接圆内
- 曲线段近似曲率阈值T_Cur=3，直线段近似曲率阈值T_Line=1.5

⚠ 细分过度



合并

- 对二次细分得到的直线段和曲线段进行合并
- 设定累积转角阈值T_mergedCur=66°
- 当合并后所得曲线的累积转角小于阈值，认为需要将细分出的相邻直线和曲线进行合并。

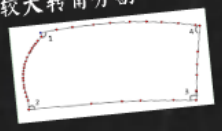
多边形轮廓线分割

合法轮廓线应满足：

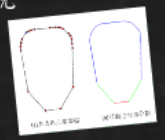
1. 直线/单弧曲线/S型曲线。
2. 若相邻三个轮廓线为“单弧曲线直线单弧曲线”或“直线单弧曲线直线”时，需考虑合并。

通过转角粗分割

- 通过较大转角分割



- 不当情况

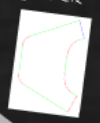


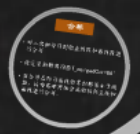
通过曲率二次细分

p点曲率=p点转角/p点所在两线段长度和

- 消除多边形大小对曲率影响：将多边形缩放至半径为10的外接圆内
- 曲线段近似曲率阈值T_Cur=3，直线段近似曲率阈值T_Line=1.5

⚠ 细分过度



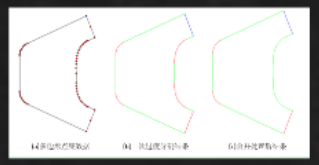


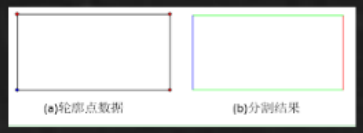
多边形轮廓线分割（续）

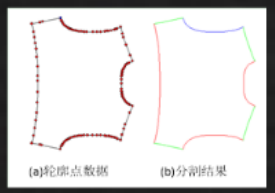
算法流程：

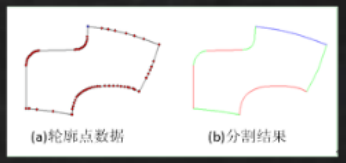
1. 计算多边形边长、顶点处转角及近似曲率；
2. 根据转角值提取多边形的大区间线段集合Seg_big；
3. 根据近似曲率对Seg_big二次细分，得到集合Seg_small；
4. 遍历集合Seg_small中所有线段，判断是否出现了“曲线直线曲线”或“直线曲线直线”类的状态，并根据阈值尝试合并。

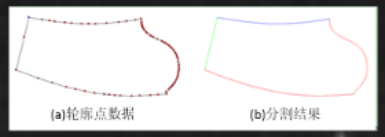
时间复杂度：O(n)

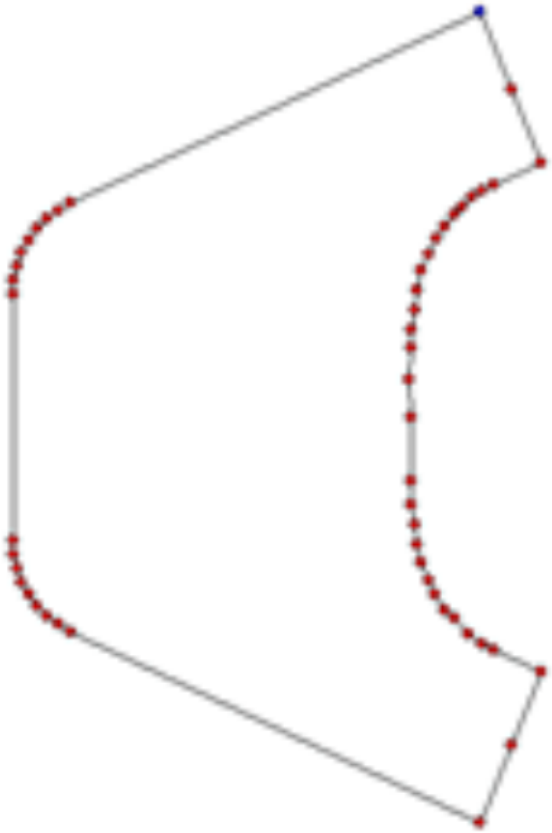












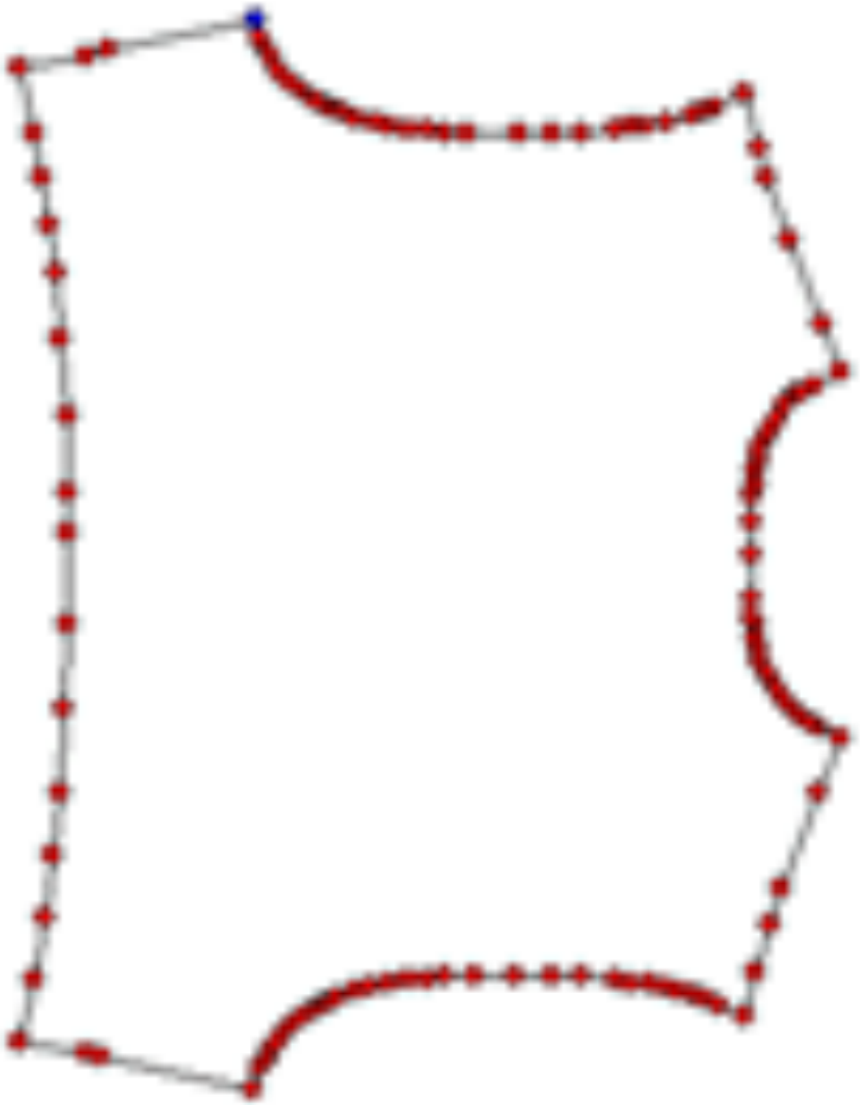
(a)多边形点集数据

(b)二次过度分割结果

(c)合并处理后结果

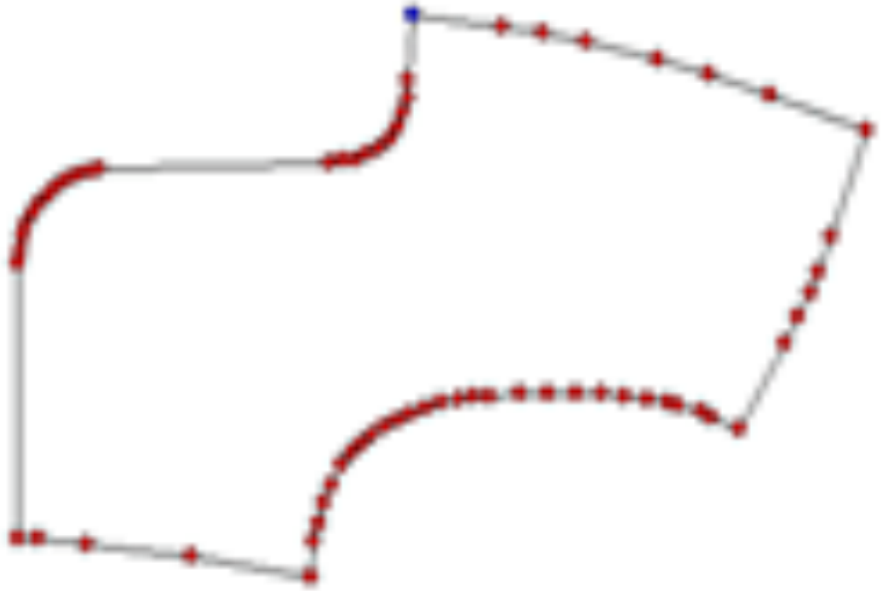
(a)轮廓点数据

(b)分割结果



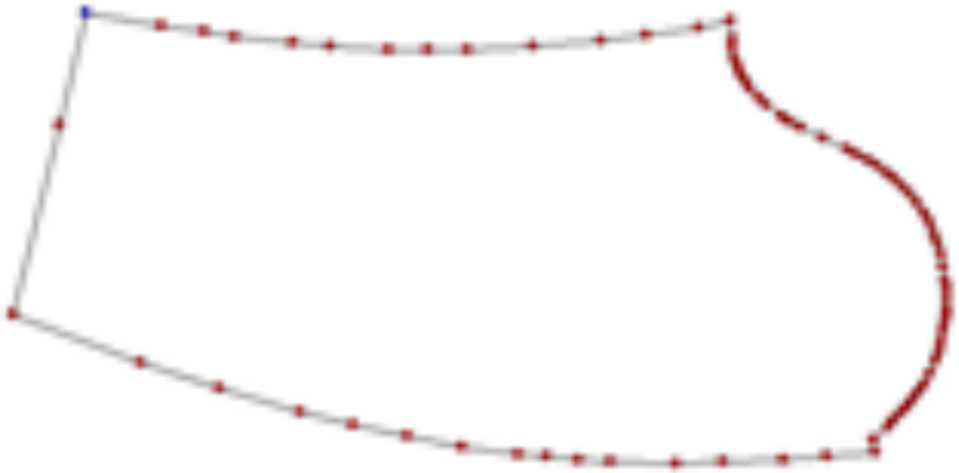
(a)轮廓点数据

(b)分割结果

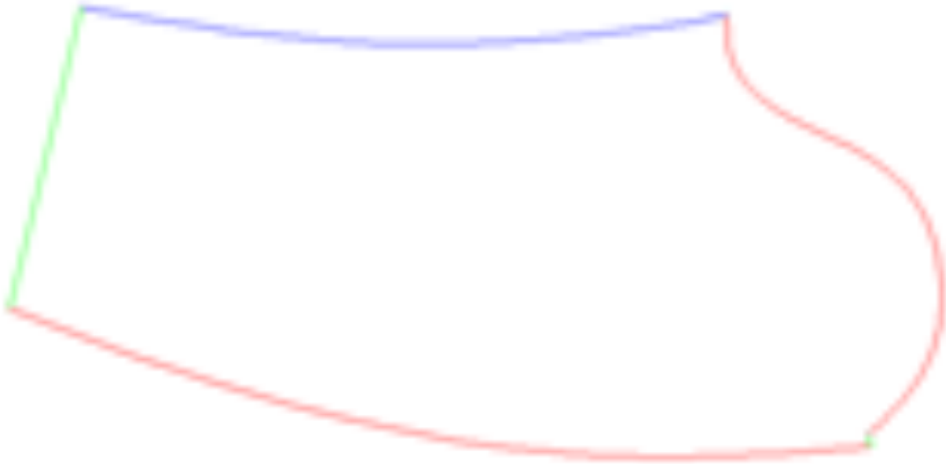


(a)轮廓点数据

(b)分割结果



(a)轮廓点数据



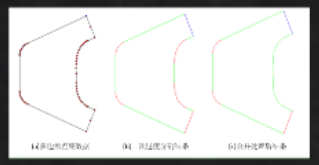
(b)分割结果

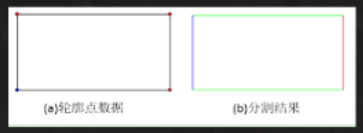
多边形轮廓线分割（续）

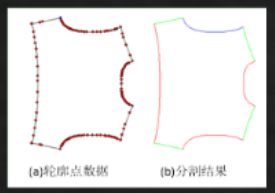
算法流程：

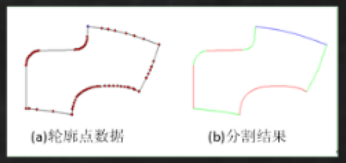
1. 计算多边形边长、顶点处转角及近似曲率；
2. 根据转角值提取多边形的大区间线段集合Seg_big；
3. 根据近似曲率对Seg_big二次细分，得到集合Seg_small；
4. 遍历集合Seg_small中所有线段，判断是否出现了“曲线直线曲线”或“直线曲线直线”类的状态，并根据阈值尝试合并。

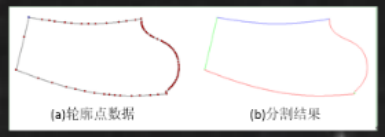
时间复杂度：O(n)





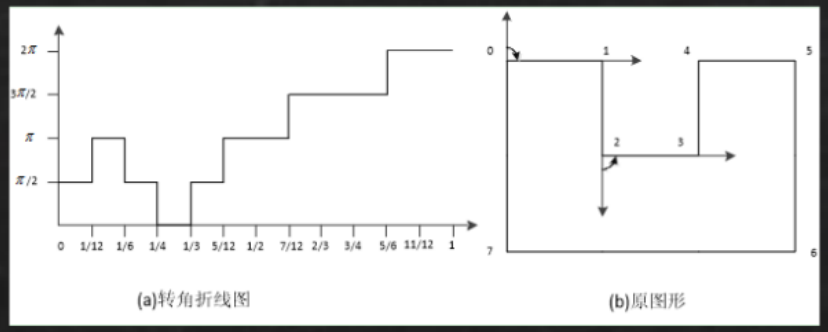






Turning Function曲线

Turning function曲线：一种将多边形按照其内角和边长转化而来的曲线。

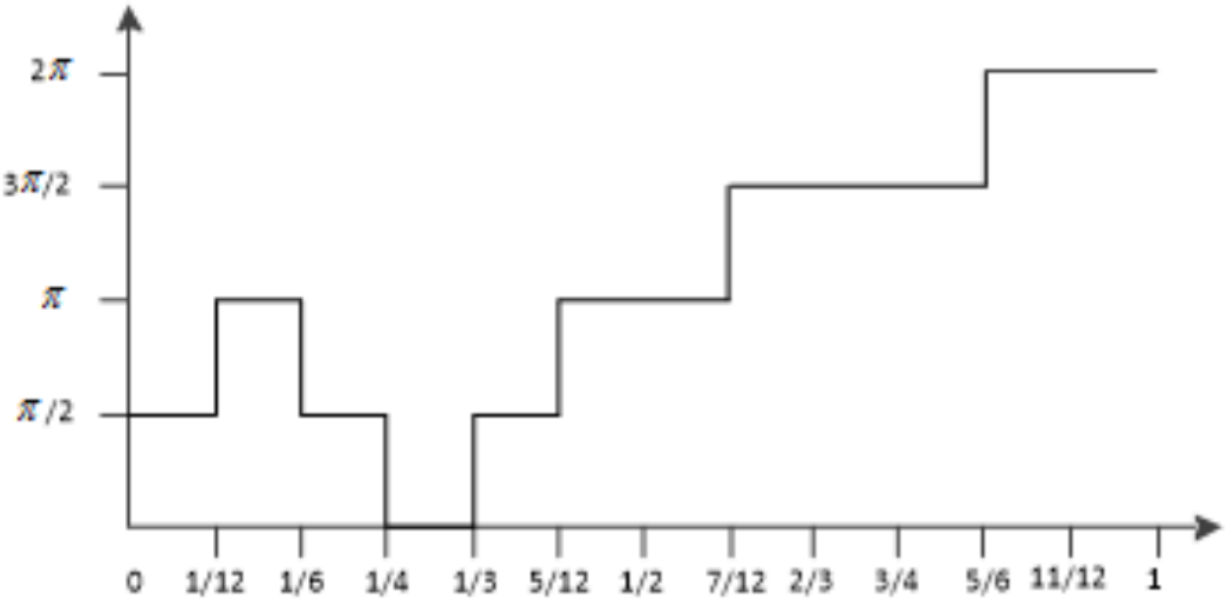


时间复杂度：O(n)

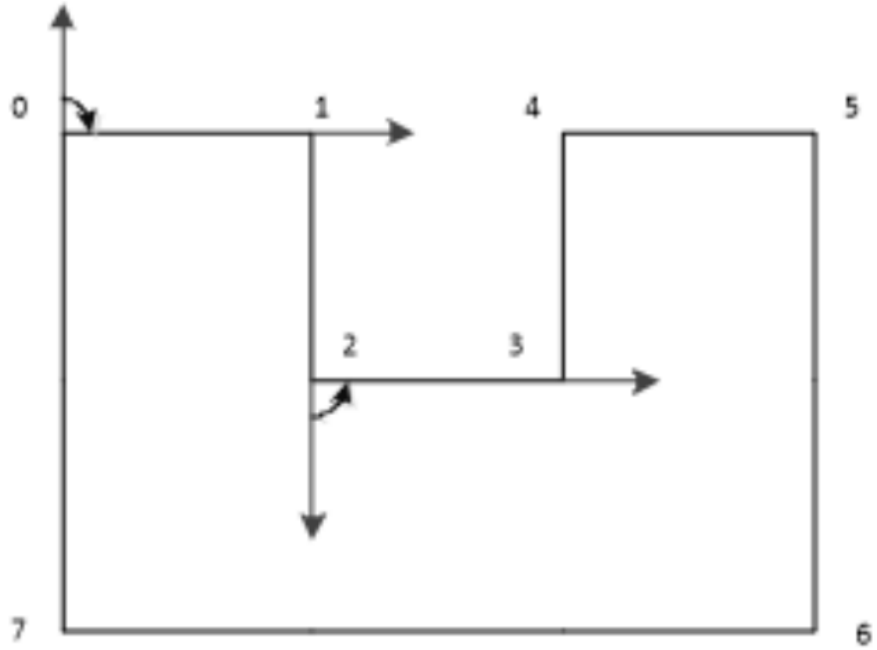
衡量两多边形相似度：使用两转角曲线的L2距离

$$d_{A,B} = \left(\int_0^1 \left(\theta_A(x,i) - \theta_B(x,j) \right)^2 dx \right)^{\frac{1}{2}}$$

function曲线：一种将多边形按照
和边长转化而来的



(a)转角折线图



(b)原图形

时间复杂度：O(n)

基于Turning Function的多边形相似性判定

算法描述：

1. 将多边形A、B进行轮廓线分割，得到k条轮廓线段；
2. 两层循环遍历A和B所有轮廓线段端点，计算以每个顶点对为遍历起点得到的两多边形的转角折线图，并计算它们间的L2距离d(A, B)；
3. 翻转B，再执行步骤2；
4. 获取步骤2~3中计算出的所有d(A, B)中最小值，若其小于阈值，两多边形相似，否则，两多边形不相似。

时间复杂度：

O(n*k*k) ←

轮廓线分割：O(n)

循环k*k次，循环体：转角折线图O(n)+L2距离O(1)

数据结构

- 存储点

存储点的坐标

```
//直角坐标表示下的点
struct BASEPOINT
{
    double x, y;        //分别为点的 x 坐标和 y 坐标值
};

//极坐标表示下的点
struct CPPoint
{
    double  ρ;          //极边
    double angle;       //极角
    int No;             //点的编号
};
```

- 存储匹配结果

存储匹配结果

```
struct PolysAdjustParams
{
    bool isNeedFlip;        //是否需要翻转
    double angle;           //旋转角度
    int matchedPos1;        //两多边形匹配时分别对应的顶点的编号
    int matchedPos2;
};
```

- 存储多边形

存储读入的简单多边形

```
struct CPolygon
{
public:
    CPolygon():bOrientation(true),pPts(NULL),nNum(0){}
    BASEPOINT* pPts;  //按序存储的顶点坐标集
    int nNum;         //多边形顶点个数
    bool bOrientation;    //顶点遍历方向：true(顺) false(逆)
};
```

- **存储点的坐标**

```
//直角坐标表示下的点
struct BASEPOINT
{
    double x, y;        //分别为点的 x 坐标和 y 坐标值
};

//极坐标表示下的点
struct CPPoint
{
    double  ρ;          //极边
    double angle;       //极角
    int No;             //点的编号
};
```

存储多边形

- 存储读入的简单多边形

```
struct CPolygon
{
public:
    CPolygon():bOrientation(true),pPts(NULL),nNum(0){}
    BASEPOINT* pPts;  //按序存储的顶点坐标集
    int nNum;         //多边形顶点个数
    bool bOrientation;  //顶点遍历方向：true(顺) false(逆)
};
```

- 存储匹配结果

- 存储匹配结果

```
struct PolysAdjustParams
{
    bool isNeedFlip;        //是否需要翻转
    double angle;           //旋转角度
    int matchedPos1;        //两多边形匹配时分别对应的顶点的编号
    int matchedPos2;
};
```

数据结构（续）

- 存储多边形轮廓线段

- 存储分割出的多边形轮廓线段

```
struct PolySegment
{
    BASEPOINT *pts;                 //线段中包含的顶点集合
    int num;                        //线段中包含的顶点数目
    double endCornerAngle;          //该线段结束角的逆时针转角
    double segLength;               //该线段长度
    double *lengths;                //轮廓点间距离
    double *angles;                 //轮廓点的转角信息
    double *curvatures;             //轮廓点对应的曲率

    int fpNum;                      //特征点个数
    int *featurePoints;             //特征点下标数组
    PolyFeaturePointType *featureEncode;//特征点编码
    PolySegments*  poly;            //该轮廓线所属的多边形

    //构造函数和析构函数由于过于复杂此处省略，只展示上述变量
};
struct PolySegments
{
    PolySegment **segments; //分割的多边形轮廓线段集合
    double perimeter;       //多边形周长
    int num;                //分割的多边形轮廓线段条数
    int firstIndex;         //0 号轮廓线 0 号点对应的原点中的下标
    double disFac;          //计算曲率时，多边形中点间距的缩放比例

    //构造函数和析构函数由于过于复杂此处省略，只展示上述变量
};
```

- 存储转角折线图

- 存储多边形的转角折线图

```
struct TFL2Map
{
    vector<double> gra;  // 每个折线累计的转角大小
    vector<double> dis;  //每个折线对应的相对长度
    bool isFlip;         //是否进行了翻转
    TFL2Map()            //构造函数
    {
        isFlip = false;
    }
};
```

- 存储分割出的多边形轮廓线段

```
struct PolySegment
{
    BASEPOINT *pts;                 //线段中包含的顶点集合
    int num;                        //线段中包含的顶点数目
    double endCornerAngle;          //该线段结束角的逆时针转角
    double segLength;               //该线段长度
    double *lengths;                //轮廓点间距离
    double *angles;                 //轮廓点的转角信息
    double *curvatures;             //轮廓点对应的曲率

    int fpNum;                      //特征点个数
    int *featurePoints;             //特征点下标数组
    PolyFeaturePointType *featureEncode;//特征点编码
    PolySegments*   poly;           //该轮廓线所属的多边形

     //构造函数和析构函数由于过于复杂此处省略，只展示上述变量
};
struct PolySegments
{
    PolySegment **segments; //分割的多边形轮廓线段集合
    double perimeter;       //多边形周长
    int num;                //分割的多边形轮廓线段条数
    int firstIndex;         //0 号轮廓线 0 号点对应的原点中的下标
    double disFac;          //计算曲率时，多边形中点间距的缩放比例

     //构造函数和析构函数由于过于复杂此处省略，只展示上述变量
};
```

- 存储

```
struc
{
};
```

- 存储多边形的转角折线图

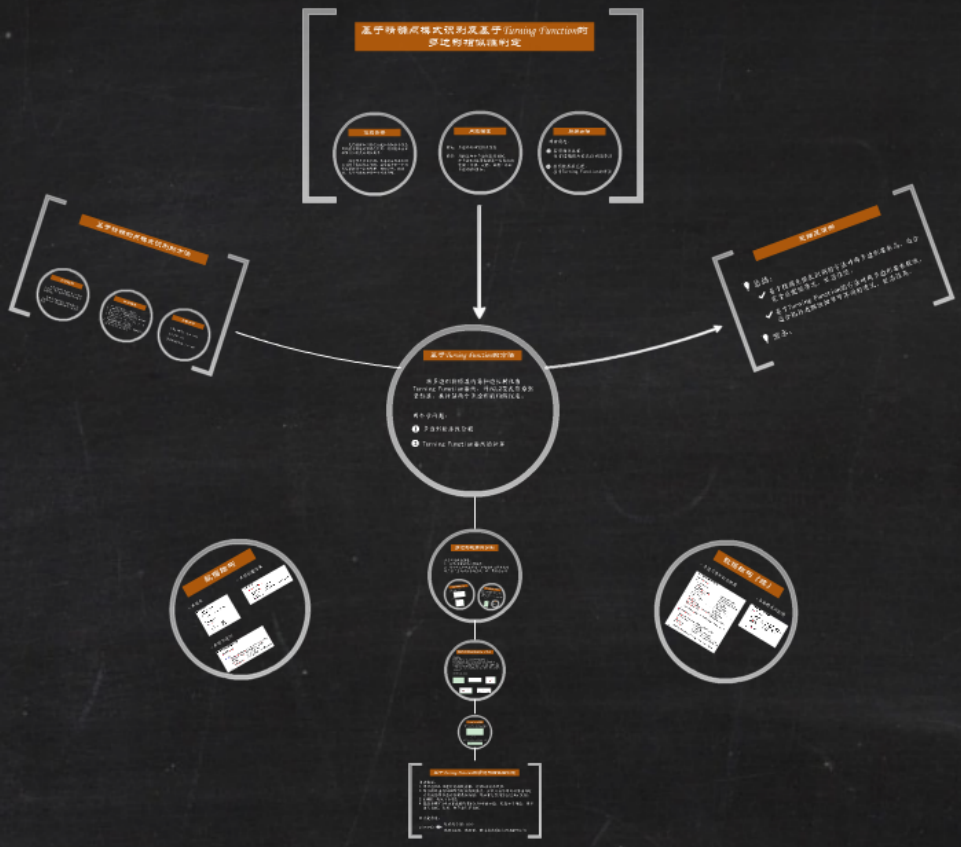
```
struct TFL2Map
{
    vector<double> gra;  // 每个折线累计的转角大小
    vector<double> dis;  //每个折线对应的相对长度
    bool isFlip;         //是否进行了翻转
    TFL2Map()            //构造函数
    {
        isFlip = false;
    }
};
```

总结及演示

总结：

- 基于精确点模式识别的方法对两多边形要求高，适合完全匹配的情况，灵活性低。
- 基于Turning Function的方法对两多边形要求较低，适合拓扑近似但细节可不同的情况，灵活性高。

演示：

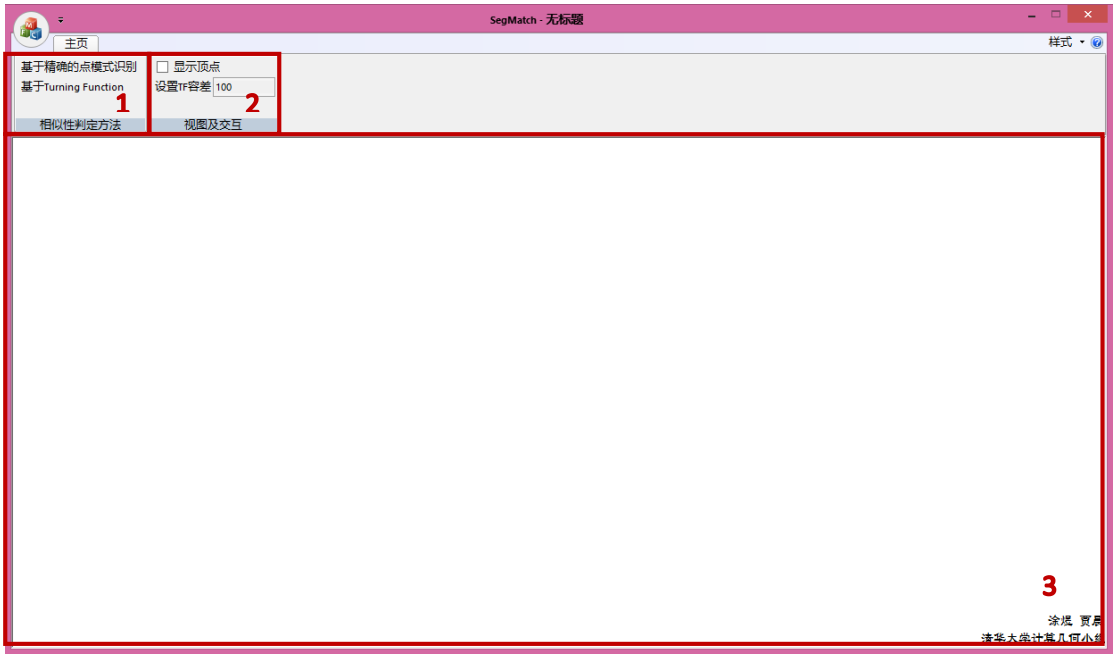


计算几何课程实验展示

软件学院：涂煜　贾晨

SegMatch 使用手册

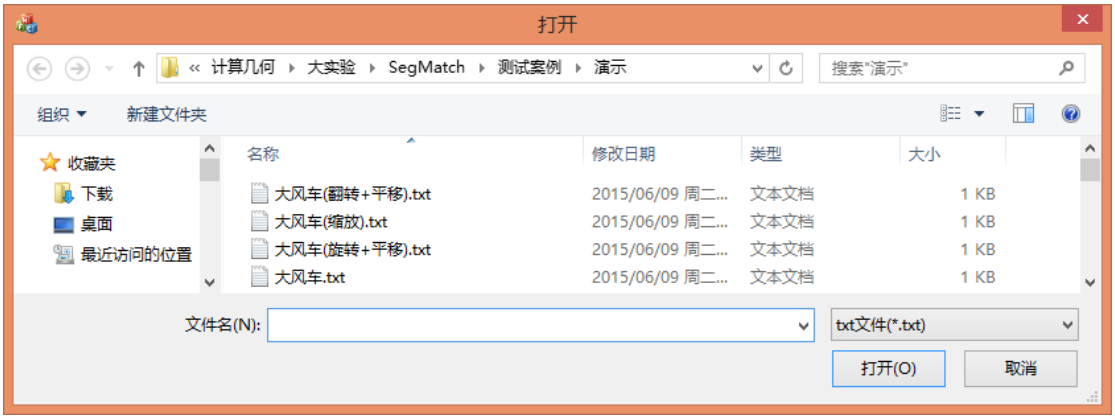
1. 程序主界面



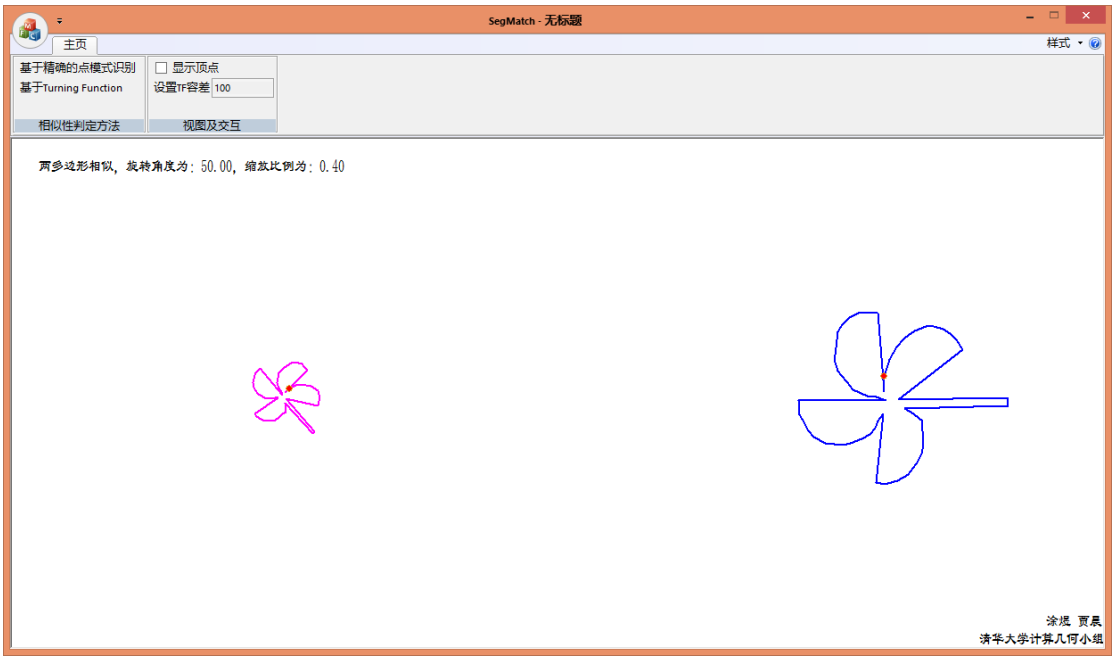
上图所示为程序主界面，共分为三个区：1 区为算法功能区，用户可以选择要使用的相似性判定方法；2 区为交互区，用户可以选择是否显示顶点，及使用“基于 Turning Function 的相似性判定方法”时的判定容差；3 区为结果显示区，用于向用户展示读入的多边形及程序运行结果。

2. 基于精确的点模式识别

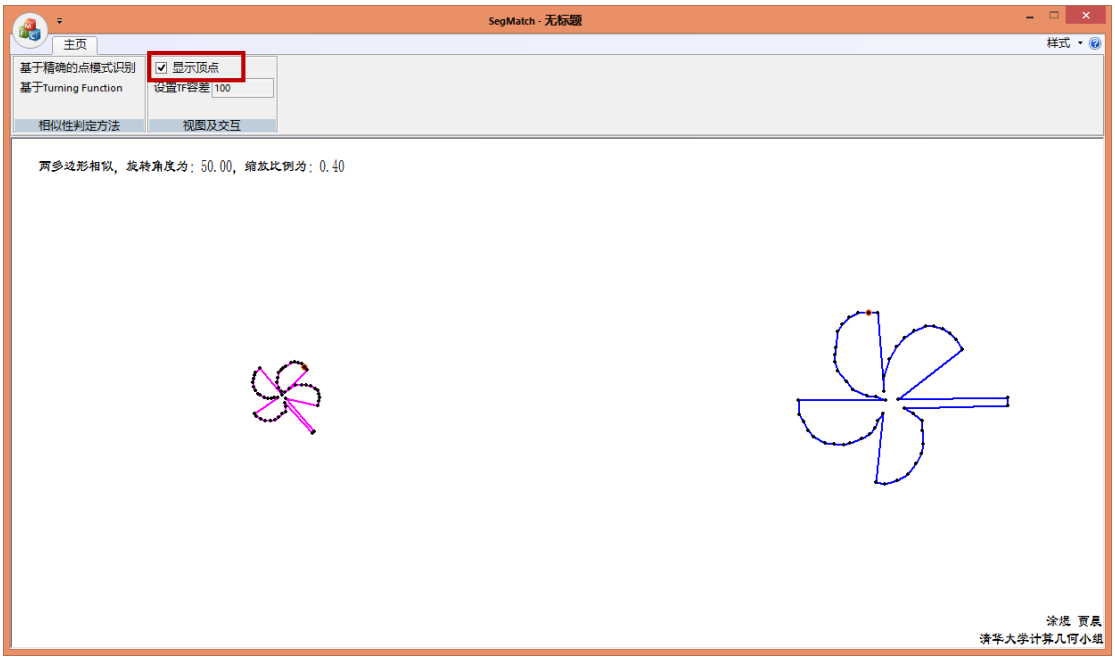
点击 1 区中的“基于精确的点模式识别”按钮，弹出如下显示框，用户选择两个多边形 txt 文件作为输入：



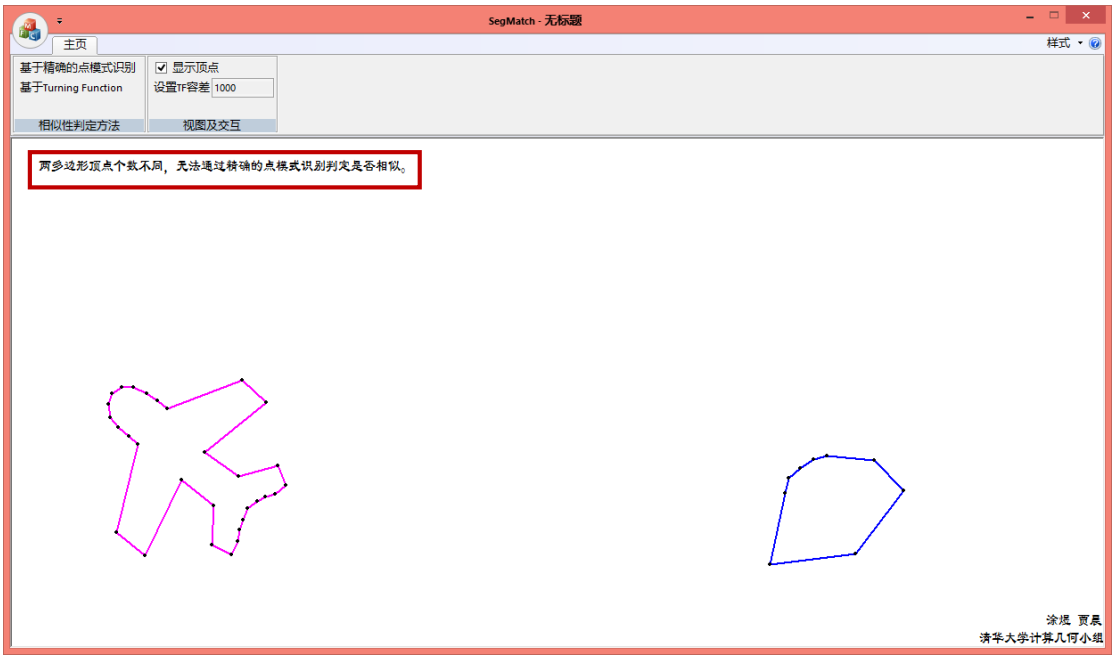
选择完成后程序使用基于精确的点模式识别的方法对这两个多边形进行相似性判定，并将结果展示在 3 区。如下图所示，左上角输出相似性判定结果，中央绘制两个多边形，红色点为动态闪烁点，用于向用户直观展示两个多边形的匹配关系，即哪一对点是相互对应的：



用户可以通过勾选“显示顶点”框在使程序在绘制多边形的同时绘制出多边形顶点，如下图所示：

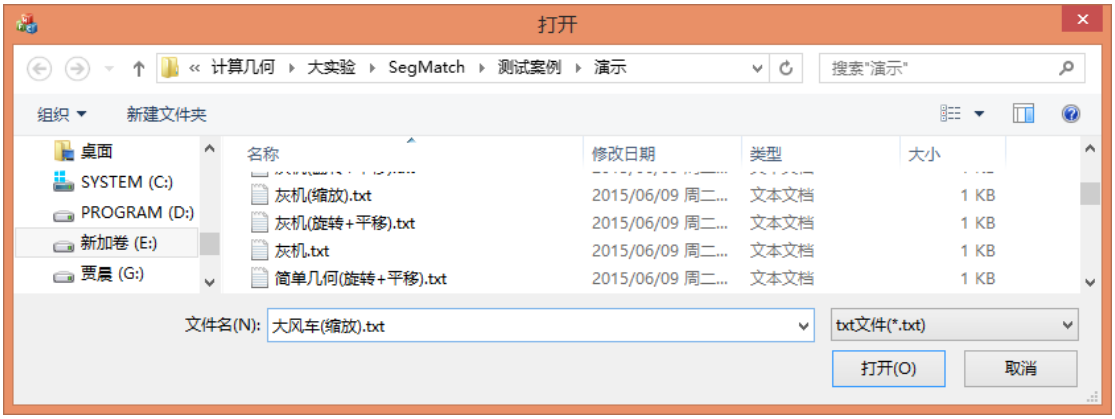


若输入的两个多边形的顶点个数不同，则不能使用基于精确的点模式识别的方法来判定相似性，如下图所示：

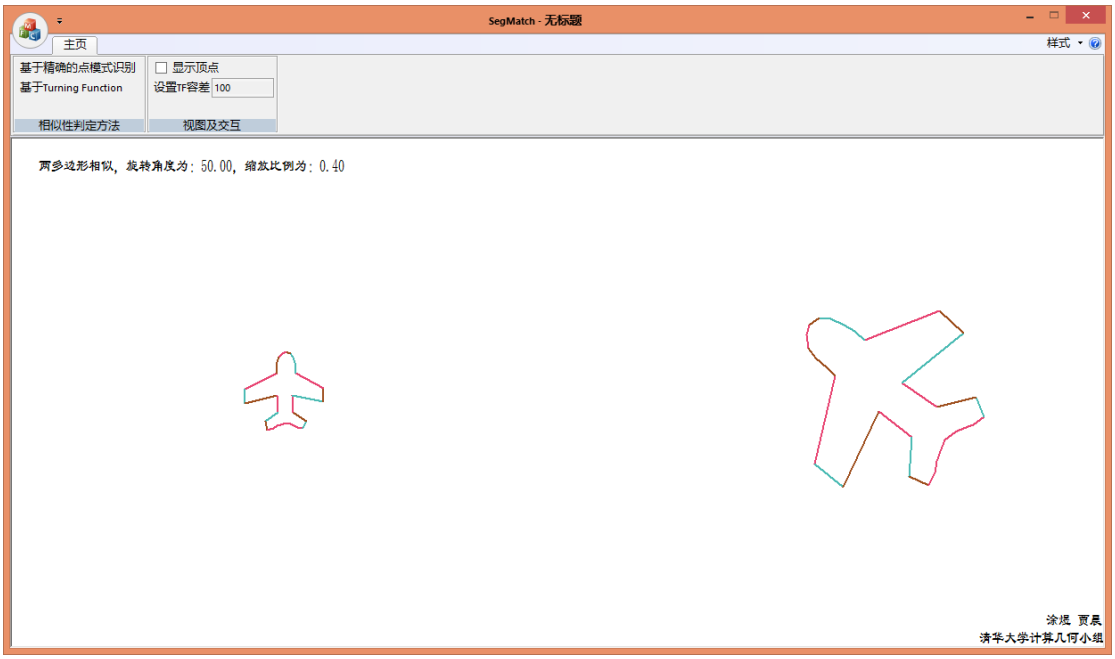


3. 基于精确的点模式识别

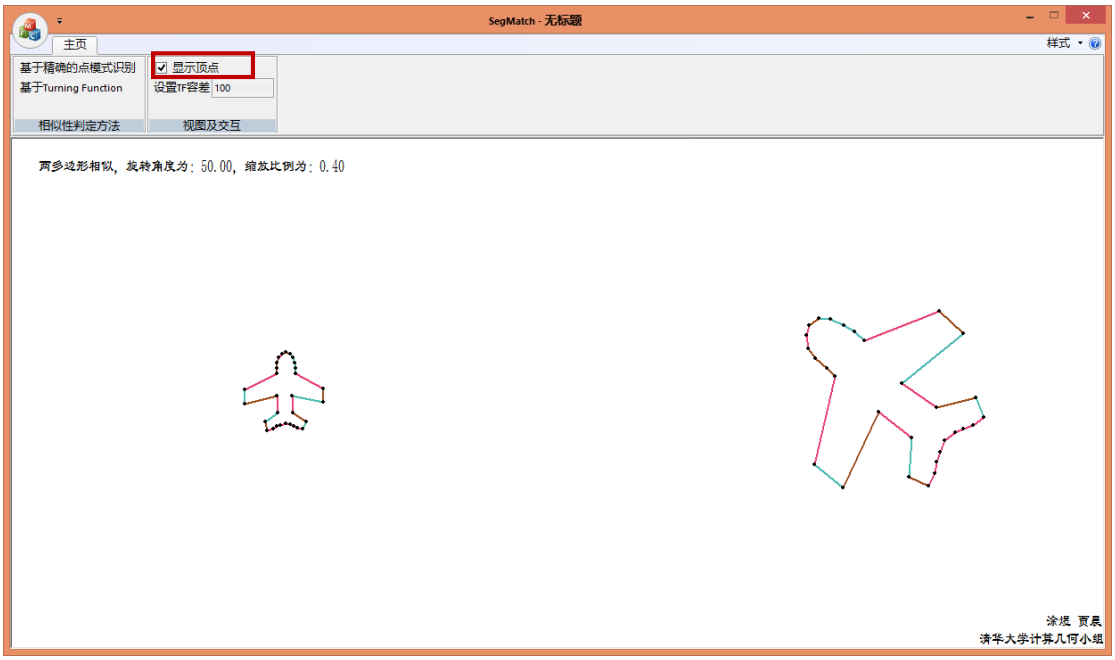
点击 1 区中的“基于 Turning Function”按钮，弹出如下显示框，用户选择两个多边形 txt 文件作为输入：



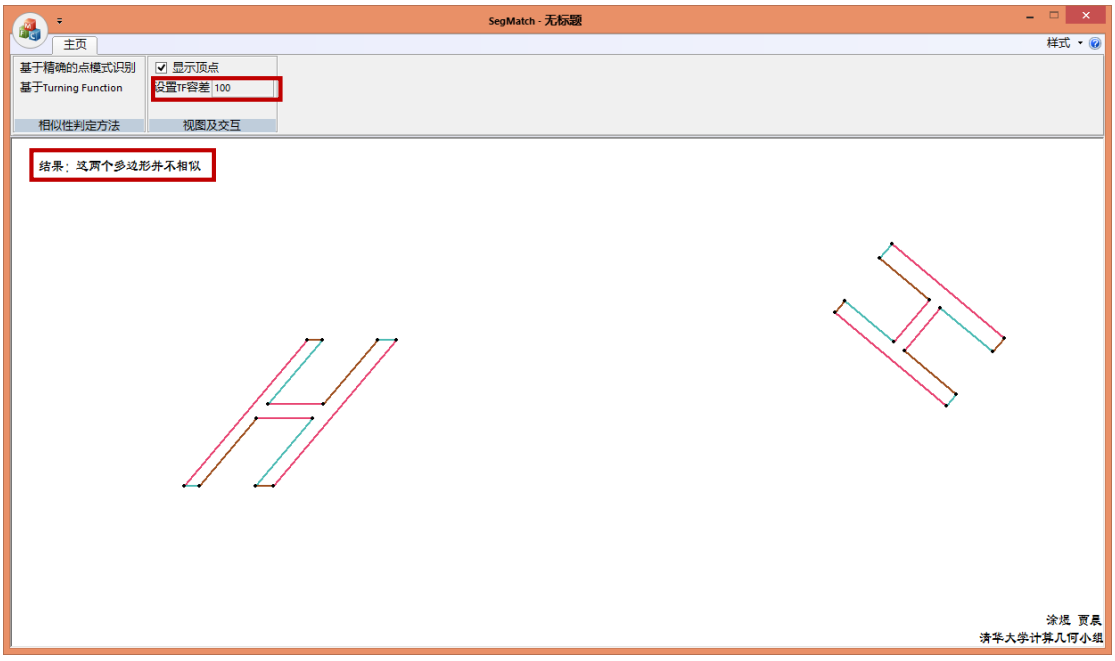
选择完成后程序使用基于 Turning Function 的方法对这两个多边形进行相似性判定，并将结果展示在 3 区。如下图所示，左上角输出相似性判定结果，中央绘制两个多边形，程序使用不同颜色标示分割出的各个轮廓线段。用户可以通过线段的颜色获取多边形轮廓线的分割结果。



用户可以通过勾选“显示顶点”框在使程序在绘制多边形的同时绘制出多边形顶点，如下图所示：



用户可以通过在“设置 TF 容差”输入框中输入不同的容差值调节使用 Turning Function 的方法判定多边形相似性时的阈值，使多边形相似的标准变得更严苛或更宽松。如下图所示，在此容差下两多边形不相似：



调节容差值，使其变为 1000，再使用基于 Turning Function 的方法即可得到两多边形相似的结果，如下图所示：