

**滁州学院3**

**本 科 生 毕 业 论 文**

**（申请学士学位）**

**论文题目** 从等高线图上产生地貌特征点的方法

**作者姓名**  王宝红

**所学专业名称** 地理信息科学

**指导教师** 费立凡

**2015年4月15日**

**学 生： （签字）**

**学 号：**

**论文答辩日期： 年 月 日**

**指 导 教 师 ： （签字）**

目 录

[摘要： 2](#_Toc417551929)

[关键词 2](#_Toc417551930)

[Abstract 2](#_Toc417551931)

[Key words 2](#_Toc417551932)

[1 引言 3](#_Toc417551933)

[1.1 选题背景与意义 3](#_Toc417551934)

[1.2 国内外研究进展 3](#_Toc417551935)

[1.3 论文组织 4](#_Toc417551936)

[2 研究基础与方法 4](#_Toc417551937)

[2.1 研究基本概念介绍 4](#_Toc417551938)

[2.2 研究内容与技术流程图 4](#_Toc417551939)

[2.3 数据前期准备 5](#_Toc417551940)

[3算法的设计与实现 6](#_Toc417551941)

[3.1等高线上正负地貌特征点分类识别与计算方法基本介绍 6](#_Toc417551942)

[3.1.1等高线光滑——三点求导三次多项式算法 6](#_Toc417551943)

[3.1.2地貌特征点提取——转角法 10](#_Toc417551944)

[3.1.3正负地貌特征点的自动判断与分类 11](#_Toc417551945)

[3.2、山顶点、洼底点、鞍部点位置与高程的推算 14](#_Toc417551946)

[3.2.1山顶、洼底点、鞍部点位置的确定 14](#_Toc417551947)

[3.2.2山顶点、洼底点、鞍部点高程的推算 17](#_Toc417551948)

[3.3 小结 18](#_Toc417551949)

[4 系统测试 18](#_Toc417551950)

[4.1 实验样区概况 18](#_Toc417551951)

[4.2 实验结果分析 18](#_Toc417551952)

[4.2.1等高线光滑结果 18](#_Toc417551953)

[4.2.2统一了等高线数字化走向 19](#_Toc417551954)

[4.2.3正负地貌特征点自动提取 20](#_Toc417551955)

[4.3 小结 22](#_Toc417551956)

[5 结论与讨论 23](#_Toc417551957)

[5.1 主要成果 23](#_Toc417551958)

[5.2 研究展望 23](#_Toc417551959)

[参考文献 24](#_Toc417551960)

[致 谢 25](#_Toc417551961)

[附 录：关键程序代码 26](#_Toc417551962)

从等高线图上产生地貌特征点的方法

摘要：地貌特征点包括山顶点、洼底点、鞍部点、等高线上的山脊点、山谷点等要素，本论文进行相关理论和实验研究，试图由矢量数据等高线，较为有效地解决地形特征点的自动提取问题。利用等高线近似曲率判别法，提取山脊线和谷底线的候选点，同时根据等高线数据，自动寻找出山顶点、洼底点、鞍部点等特征点的位置，并推算出其高程，为地貌结构线的提取打下坚实基础，从而将为等高线的自动间接综合提供帮助。

关键词：地貌特征点；地貌结构线；山脊线；山谷线；山顶点；洼底点；鞍部点；自动提取

**Methods of Extracting Geomorphological Feature Points Based on Contour Line Maps**

Abstract: The geomorphological feature points include hilltops， bottoms of depressions， saddle points， mountain ridge and vally points etc. on the contours. This thesis introduces the study on the theories and experiments for effective extraction of geomorphological feature points based on the source contour lines. The feature points on ridge and valley lines are extracted by the approximate curvature calculation， and the hilltops， the depression bottoms and the saddle points， including their elevation， are all calculated based on the original data of the contour lines. The extraction of these feature points has laid down a solid foundation for the extraction of the geomorphological structure lines and will be helpful for the automated indirect generalization of contour lines.

Key words：geomorphological feature points， geomorphological structure lines， ridge lines， valley lines， hilltops， depression bottoms， saddle points， automated extraction

# 1 引言

## 1.1 选题背景与意义

地形特征线又叫地性线，包括山脊线和山谷线。它描述了地形的骨架结构，揭示了地貌形态的本质[1]。

山脊线和山谷线是主要的地形结构线，描述形态复杂、起伏不定的地形的骨架结构，揭示地貌形态的本质，为数字高程模型(DEM)生成、流域分析、地貌综合等相关研究提供重要信息[2]。

由于地形结构线将地形划分为基本斜坡，是特征相同单元曲面的边界线，考虑地形结构线可以减少DEM内插面跨越山谷或山脊而引起跨越结构的误差。又因为山脊线、谷底线具有分水、合水的物理意义，地形结构线的提取是流域分析、水文分析、河系自动生成等的重要内容[2]。在对等高线的综合中，无论是用直接综合，还是间接综合法，在等高线图上隐含表示的地貌结构线都是不可或缺的信息。在作战模拟和指挥自动化中，地形特征数据是确定进攻、防御部署重要区域的依据，军事测绘为这种更高层次的地形分析应用提供有关信息[3]。

鉴于地形结构线在军事上、国民经济中的重要作用和人工输入工作繁琐、量大、人为因素影响大，从地形的数字化表示一DEM中自动提取山脊线和山谷线一直是地学研究的热点问题，也是地貌水系综合的基础问题之一，吸引了大量的研究工作者[2]。而识别和提取等高线图上的特征点如山顶点、洼底点、鞍部点、等高线上的山脊点、谷底点等，是跟踪山脊线、谷底线等地形特征线的前提。

## 1.2 国内外研究进展

目前，出现的各种地形结构线提取算法均源于两种原理:基于几何形态分析和基于地形表面流水分析.属于前者的算法有基于图像处理的特征提取算法、基于TIN的特征提取、地形断面极值法、等高线曲率判别法和等高线骨架化法等，属于后者的有地形表面流水物理模拟法与等高线垂线跟踪法等，也有将两种原理结合使用的算法[黄培之2001][4]。

许多作者的地貌结构线提取主要是指在规则格网DEM上自动提取山脊线和山谷线数据[5,6,7,8,9]。由于规则格网DEM的误差、特别是DEM粗差的存在将会引起伪极值点和遗漏极值点的情况出现，伪极值点又会错误引导结构线的连接，而遗漏极值点则又会阻断结构线的连接，而DEM误差是无法避免的，并且粗差也难以剔除，这是此类算法效果不佳、徘徊不前的主要原因[2]。

陈晓勇(1990)主要利用数学形态学有关算法来提取等高线图像的骨架，以此得到地貌特征点和特征线[10]； 吴艳兰（2004）运用地图代数相关的算法也进行了基于等高线骨架法的结构线提取研究。等高线骨架化法是指将山脊线和山谷线两侧的地形视为对称变化，然而这与多数地形的实际变化不相符，因此，用该方法得到的山脊线和山谷线走向很生硬，有一定程度的近似性。这是从宏观图形的角度上理解等高线骨架法的概念，将地貌结构线理解为等高线平面图形的骨架线，由于等高线族是相互协调、致密的空间曲线族，这种概念下的骨架化存在各种内在的矛盾[2]。

冯桂(2001)在研究现有等高线提取地貌特征线的几种方法基础上，提出了等高线弦线判断法，这是一种根据等高线的弦线来判断地形特征的，主要根据等高线上离散分布点之间所连弦线的垂线与相邻两条等高线之间关系来判别山谷、山脊、山顶、斜坡等七个地形特征。文中给出了特征分类的方法以及算法流程，但是并没有指明如何进行等高线上离散分布点的取样 [11]。

前人研究的方法各有优缺点，但还都不够完善，本论文尝试利用等高线近似曲率判别法，提取山脊线和谷底线的候选点。同时根据等高线数据，自动寻找出山顶点、洼底点、鞍部点等特征点的位置，并计算出其高程，为地貌结构线的提取打下坚实基础。

曲率分析法的理论基础建立在 Attneave的著名结论之上：描述一条曲线形状的信息集中在具有高曲率的特征点上[12]。特征点一般在弯曲程度最大（曲率最大）的地方，根据 Plazanet 1995年[13]对弯曲的定义：弯曲是两个相邻拐点之间的几个顶点的集合[14] 。

## 1.3 论文组织

本文共有五个章节，  
　　第一章是总体概述，主要论述论文的选题背景与意义、国内外研究进展以及论文的组织情况。  
　　第二章主要介绍论文的研究内容与方法以及数据的前期准备

第三章主要介绍论文的算法设计与实现，此章节分为两部分：一是介绍脊、谷线处特征点识别与计算算法；二是山顶点、洼底点、鞍部点位置与高程的推算算法的介绍。

第四章主要介绍地貌特征点提取的结果、结果对比分析以及得出的结论。  
　　第五章则是对本论文的总结，并介绍论文的不足之处以及对未来研究方向的进一步展望。

# 2 研究基础与方法

## 2.1 研究基本概念介绍

地貌特征点包括山顶点、洼底点、鞍部点、等高线上的山脊点、山谷点等要素。

本论文主要实现提取等高线上的山脊点、山谷点，并将正负向地貌区别开来(正向地貌是山脊，负向地貌是山谷)，由于内容量太多，时间和精力都有限，故此对山脊山谷处特征点提取理论与实验都完成，而山顶点、洼底点、鞍部点只做了理论上的研究，具体实验还有待于进一步研究。

## 2.2 研究内容与技术流程图

本论文研究以地学、地理信息科学、地图制图学等学科的原理为理论基础，使用ArcGIS Engine开发包，Visual Studio2010开发环境，采用C#语言进行程序开发。编程实现地貌特征点自动提取。以下是技术流程及流程图2-1：

1、等高线光滑——三点求导三次多项式算法；

2、等高线等分——累加弦长等分法；

3、正负地貌特征点识别与曲率计算——判别等高线数字化方向以及用转角法计算近似曲率；

4、山顶点、洼底点、鞍部点位置与高程的推算；

5、得到所有地貌特征点。

|  |
| --- |
| 脊谷处特征点提取  得到所有地貌特征点  等高线数字化方向的判别与纠正  正负地貌特征点符号的计算  推算山顶、洼底、鞍部点的位置和高程  等高线数据  对原始等高线的光滑  以小段截等长弧段 |
| 图2-1 本文研究的技术流程图 |

## 2.3 数据前期准备

本论文用的是安徽部分地区等高线数据和黄土地貌部分地区等高线数据，根据图2-1 本文研究技术流程图可知，等高线数据是进行整个论文研究的基础，因此，这些信息的提取是论文前期工作的关键。

之所以选择安徽和黄土地貌等高线数据是因为它们的地貌特征有很大的区别，安徽的趋于平缓而黄土地貌的地形沟谷众多，地形复杂多变，两个数据同时使用使本论文更有研究意义和价值。

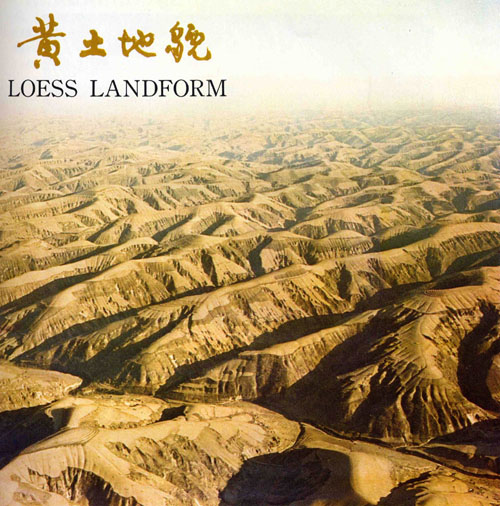
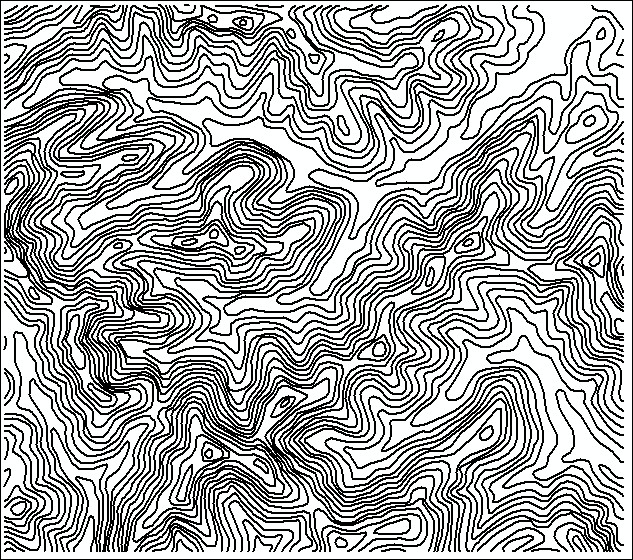


图2-2安徽部分地区图 2-3黄土地貌局部形态



图2-4黄土地貌部分地区等高线图

# 3算法的设计与实现

## 3.1等高线上正负地貌特征点分类识别与计算方法基本介绍

### 3.1.1等高线光滑——三点求导三次多项式算法

在 GIS 条件下，曲线以折线来逼近，一条折线上的每条直线段长短不一，整个被逼近的等高线又难于用一个函数表示，所以无法获取一点的真正曲率。本文先改善原始等高线的光滑性，然后按某一小值重新截取等高线，使得在等高线各截取点之间的累加弦长相等，最后计算在各截取点上的近似曲率值，经曲率阈值筛选出特征点。

**三点求导三次多项式光滑插值算法：**

三点求导分段三次多项式曲线光滑法是由五点求导分段三次多项式曲线光滑法改进而来，首先介绍：五点求导分段三次多项式曲线光滑法。

这里，对曲线进行光滑的要求是：

1、虽然每条等高线是由长短不一的线段首尾相接组成，但其视觉效果应该是光滑的。

2、线段本身所构成的函数要处处连续，且其一阶导数也要处处连续，即使在端点处也要做到左导数等于右导数（当然二阶导数、三阶导数至n阶导数都连续其光滑效果更加，但考虑到算法的实用性与效率性，一阶导数连续即可）。

3、曲线上距离端点相对较近的位置两边应该是对称的。

4、要尽量避免自相交。

算法: 如下是从一条等高线截取的一段由四条长短不一的线段通过五个点首尾相接构成，五个点分别是P1，P2，P3，P4，P5，作P1P2、P2P3延长线，作P3P4、P4P5反向延长线相交如图3-1，再过P3点作CD使满足关系式3-1，过P3点的切线由图3-2公式求出。

 （3-1）

 （3-2）

图3-1

为了方便，我们使用cosθ3和sinθ3表示t3 （即tgθ3 ），以及分子和t3被看作是一个直角三角形的两个边缘中的表达的分母公式（3-3）（参照图3-2）：

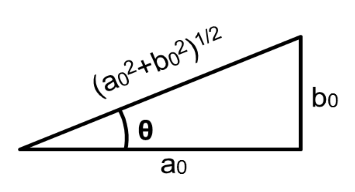
  （3-3）

图3-2

当 w2=w3=0， cos θ3 和 sin θ3 都不确定， 则使 w2=w3=1。

让两个连续点pi和pi+1之间的三阶多项式方程为(3-4)、(3-5)：

x=p0+p1z+p2z2+p3z3 (3-4)

y=q0+q1z+q2z2+q3z3 (3-5)

在这些公式中pi、qi(i=0，1，2，3)为常数，z是参数，当曲线从pi到pi+1， z值从0到1。此方程式需满足以下条件：

When z=0， x=xi， y=yi， dx/dz=rcosθi， dy/dz=rsinθi，

When z=1， x=xi+1， y=yi+1， dx/dz=rcosθi+1， dy/dz=rsinθi+1.

Where r=[(xi+1-xi)2+(yi+1-yi)2]1/2

在这种情况下pi 和qi (i=0，1，2，3)可以如下赋值：

p0=xi

p1=rcosθi

p2=3(xi+1-xi)-r(cosθi+1+2cosθi)

p3=-2(xi+1-xi)+r(cosθi+1+cosθi)

q0=yi

q1=rsinθi

q2=3(yi+1-yi)-r(sinθi+1+2sinθi)

q3=-2(yi+1-yi)+r(sinθi+1+sinθi)

如上需要考虑在等高线两端，即起点和终点各补2个点，补点方法：假设前三个给定的点（x3，y3），（x4，y4），（x5，y5），则需要被补的两个点（x2，y2）和（x1，y1）需满足如下公式(3)：

x=g0+g1z+g2z2 (3-6)

y=h0+h1z+h2z2 (3-7)

其中gK， hK（k = 0，1，2）为常数，z是参数，当z = j，x = xj ，y = yj（j= 1，2，3，4，5） ，则：

x2 = 3x3-3x4 + x5

x1 = 3x2-3x3 + x4

y2 = 3y3-3y4 + y5

y1 = 3y2-3y3 + y4

等高线最后2个点的补充与此类似。

五点求导分段三次多项式曲线光滑法的优点：

1. 计算相对简单；
2. 在数学上严格；
3. 所得的曲线可以通过每一个给定的点正好同时有连续一阶导数处处连续；
4. 当给定的点相对密集，所得到的曲线将很满意。

但在某些条件下其有较大缺点：

1. 曲线上突然急转弯的地方(不理想的曲线)；
2. 对于连续的曲折蜿蜒，曲线有时相交于本身；如下图3-3

图3-3

故本文使用三点求导分段三次多项式曲线光滑法去弥补五点求导分段三次多项式曲线光滑法的缺陷。这种方法的基本原理与上述五点求导三次多项式光滑插值类似，其主要的区别在于它是三点求导，每一个给定点的切线方向是由前一点、此点本身和下一点的相对位置来确定，因此，三个连续点被用来计算过中间点的曲线切向。

假设有三个点Pi-1， Pi， Pi+1，它们的坐标分别是(xi-1， yi-1)， (xi， yi)， (xi+1， yi+1)，如图3-4

图3-4

线段PiM的斜率是：

KPiM = (ym-yi) / (xm-xi)

因此，关于点Pi的切线的斜率为：

KEF = -1 / KPiM

设EF的方向是曲线的移动方向，由此判断象限，则角度α在0 - 2π由EF和X轴唯一确定。tgα是曲线上的点Pi的一阶导数，以此类推，求曲线上每个tgα。三点求导分段三阶多项式方程与五点求导分段三阶多项式方程相同，但在曲线两端进行补点时，第二个点与第三个点确定一个圆心并形成一圆弧和第一个点与第二点形成的圆弧弧长相等，由此类推第一个点的位置，最后一个点获得与此类似。与第三个点。其中r’值表示曲线的松紧度可随时调整，r对应五点求导公式中的r，满足如下公式(3-8)：

r’ = r · |(1-z)sinβi+zsinβi+1| (3-8)

曲线如图3-5。

图3-5

用三点求导分段三次多项式曲线光滑法可以很好的解决图3-3出现的问题，利用同一套分布非常稀疏的原始数据点，光滑后的效果如图3-6。

图3-6

与五点求导分段三次多项式光滑法相比三点求导分段三次多项式光滑法有如下优点：

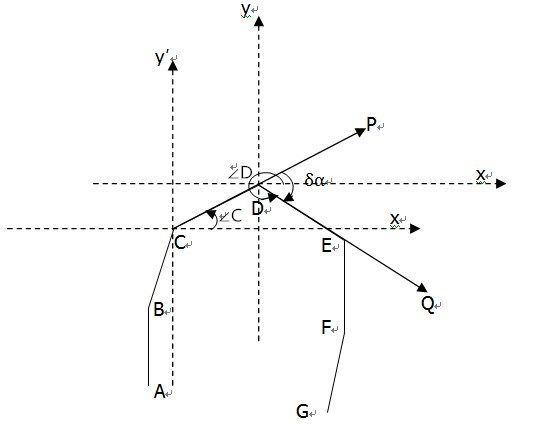
1. 即使在给定点的分布相对稀疏的情况下也能取得令人满意的结果；
2. 该曲线通过所有指定点；
3. 一阶导数是处处连续的；
4. 节点附近的每一个点的图形对称；
5. 数学原理简单，易于编写程序；
6. 有很强的避免自相交的能力。

### 3.1.2地貌特征点提取——转角法

近似曲率计算用以下公式（3-8）：

(3-8)

先对光滑后的等高线进行累加弦长等分（公式里两点之间的距离，在试验中要根据具体等高线及探索结果来定），直接调用AE接口就可以实现，等分的话容易造成部分原始节点不被保留，但是在允许的误差范围内的，不然难以实现真正的累加弦长等分。等分的目的是相等，计算时就可以只计算转角代替曲率即近似曲率，提高了效率。



如图3-7所示：是从一条等高线上截取的一段，该段由节点A、B、C、D、E、F、G构成，∠A、（后面依此类推地改。）∠B∠C∠D∠E∠F∠G是对应节点向下一节点的方向角（如∠C是CD段的方向角，∠D是DE段的方向角），且方向角的取值范围是[0゜，360゜)；

代表曲线上拐点D处的转角即“∠D∠C”，且转角的取值范围是[-180゜，+180゜)；表示CD直线距离。

图3-7

由3.1.1光滑及等分后产生的新的等高线，取代上述公式直接计算提取特征点，转角的公式：

=i+1 (新方向角）i （老方向角） 如图3-7：=∠d∠c

规定：∈[0゜，360゜)，∈[-180゜，+180゜]

1. 若| |>180゜且>0゜则=-360゜；
2. 若| |>180゜且<0゜则=+360゜；
3. 方向角的计算要先排除特殊角即+90゜和-90゜，

If（（fabs（xi+1-xi）<0.0000001）and （yi+1> yi））then i=+90゜；

If（（fabs（xi+1-xi）<0.0000001）and （yi+1< yi））then i=—-90゜。

### 3.1.3正负地貌特征点的自动判断与分类

在制图综合中，规定数字化方向必须严格遵循左高右低，即数字化时等高线的走向必须严格遵循左边地势比当前的高，右边的地势比当前的低，如果数据是规范的，我们提取特征点的算法能判断出转角为正的最大值顶点是正向地貌特征点，转角为负的绝对值最大点是负向地貌特征点，然而在现实的数字生产中，不见得制图员都按照左高右低规范跟踪数字化，所以我们要对原始数据的数字化方向是否符合规范进行自动判断和修正——将数字化方向统一成左高右低，以便能达到无误地自动判别正负向地貌特征点的要求。

若等高线走向是严格按照规范化的左高右低且转角计算结果为正的最大值，则可断定其是正向地貌特征点，反之，转角计算结果为负值且绝对值最大则可断定其是负向地貌特征点（如下图3-8中对一根选中的等高线用红色“̶”表示负向地貌特征点，红色“+”表示正向地貌）。

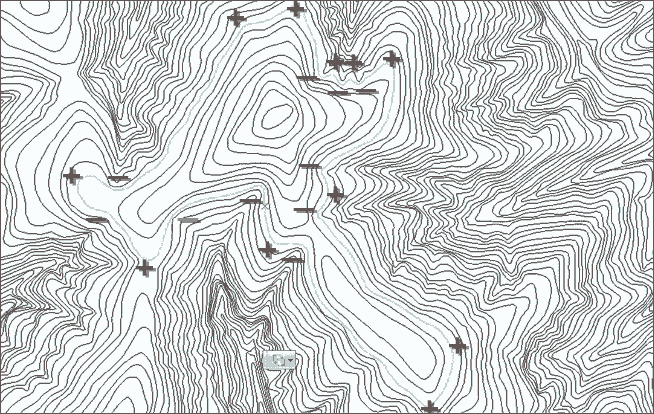


图3-8

等高线走向是否符合规范要分闭合与不闭合等高线两种情况判断：

对于正向地貌（等高线处于山地）来说，若是闭合的等高线，且等高线走向为逆时针，则符合左高右低数字化原则，此时通过计算多边形的面积的正负来判断等高线的整体走向，例如用三角形或梯形累加法计算得到的面积为正值则可以断定其整体走向为顺时针，因而其数字化方向需反之。

对于负向地貌（等高线处于洼地）来说，若是闭合的等高线，且等高线走向为顺时针，则符合左高右低数字化原则，此时通过计算多边形的面积的正负来判断等高线的整体走向，例如用三角形或梯形累加法计算得到的面积为负值则可以断定其整体走向为逆时针，因而其数字化方向需反之。

计算多边形面积的方法有两种：分别是三角形累加法和梯形累加法。如图3-9、图3-10：

* 1. 求多边形面积的三角形累加法

点序：顺时针

（P6 = P1）



图3-9

②求多边形面积的梯形累加法

点序：顺时针

（P6 = P1）

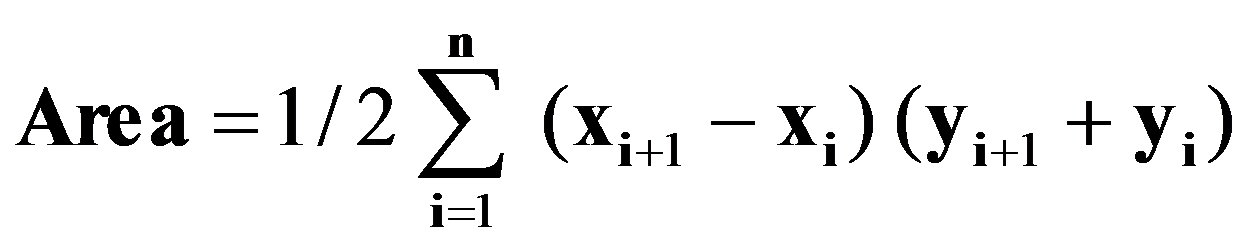


图3-10

如果地形图上等高线不是闭合的，则将其与图框相交处进行连接形成闭合的多边形，判断是否符合规范——左高右低。

整体算法：

1、将整幅地图的闭合等高线按面积从小到大排队。

2、将上一步中已经排队好的等高线一一取出如图3-11，选取其（起始闭合等高线）起点直线向右探测如图3-12（a、b、c、d）蓝色箭头指向，遇到等高线1高程值小于其高程值（起始闭合等高线的高程值为origin，origin是一个变量），则说明是山顶，判断其是否为逆时针，若不是则反向；遇到等高线1高程值大于其高程值，则说明是洼地，判断其是否为顺时针，若不是则反向。

3、判断等高线1是否为闭合，不闭合则加图框点使其形成闭合区域如图3-12c及图3-12d。

4、如下情况：

a、等高线1的高程值小于origin：包含起始等高线如图3-12a、图3-12b（图3-12b表示的是等高线2，这里是说接边框包含情况类似于3-12b等高线2，下c、d同此），则等高线1方向须同origin方向相同（计算经过接边框后闭合的等高线1的面积值符号要与origin面积值符号相同则可以断定origin与等高线1方向相同），若不是（等高线的面积符号与origin的相反），则反向；等高线1不包含起始等高线如图3-12c、图3-12d，则等高线1须与origin方向相异，若不是，则反向。完成后标记这根等高线。

b、等高线1的高程值大于origin：包含起始等高线，则等高线1须与origin方向相异，如不是，则反方向；等高线1不包含起始等高线，则等高线1须与origin方向相同，如不是，则反方向。完成后标记这根等高线。

5、将探测到等高线的高程值赋给origin。

6、接着从起始等高线向右探测第二条等高线（等高线2），规则如上，直至探测到图框最右，如果探测到图框的最右边还没有探测到等高线，则用上述算法向左，向下，向上探测，如果还没探测到则换下一个闭合等高线探测。

7、以此类推，直至完成所有等高线的标记。

然后用此统一方向后的等高线进行3.2中的转角法提取特征点，根据角度值的负正即可相应地得到特征点所在的地貌的正负向。

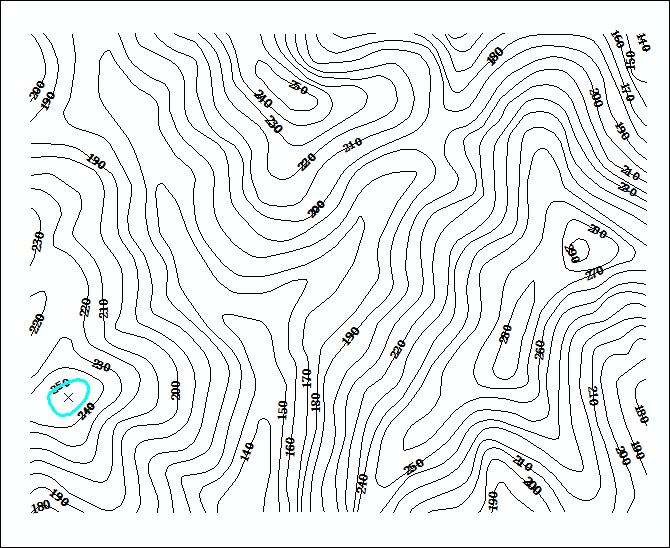


图3-11

|  |  |
| --- | --- |
|  |  |
| a | b |

|  |  |
| --- | --- |
|  |  |
| c | d |

图3-12

## 3.2、山顶点、洼底点、鞍部点位置与高程的推算

### 3.2.1山顶、洼底点、鞍部点位置的确定

核心算法：

1. 任意取一条等高线，若不是闭曲线，则放弃。
2. 若是闭曲线，于是对其进行填充探测（通过直接调用AE接口可以实现面域的填充栅格化，在这里使用栅格填充探测不会有遗漏），若在探测时发现“非空”，则放弃对这一条闭曲线的兴趣，而另取一条闭曲线进行填充式探测，若发现此闭曲线内部是“空的”，则说明这是个小山包或坑底，用此方法直到该原始数据里的所有闭曲线全部探测完为止。获取的山包形态各样，大致可以分为三类如图3-13（A、B、C）所示。
3. 有两种方法可以推算出山顶点及鞍部点的位置

方法一：用矢量计算的方法，获知在该小山包中相距最远的两个点MN如图3-13（A1、B1、C1）。

方法二：求出各种形态的小山包的最小外接矩形，如图3-13（A2、B2、C2）中的虚线表示的最小外接矩形，然后在矩形的最长的两个边找到对应的中点M、N连接MN。

接着用此两个点连线方向的垂线方向以在图上0.2mm的间隙扫描该山包，并随时记录其每根扫描线的中心及宽度（此时用矢量计算更为方便），将所有中点顺序连接起来就是它的轴线（一般为曲线），根据其宽度的变化就可以确定有几个“花生米”，每个“花生米”需要确定一个中心（取“花生米”里最长的线段的中点）即山顶点的位置。

|  |  |  |
| --- | --- | --- |
| 找到距离最大的两点MN，将MN连接，以垂直于MN的方向等距离扫描该小山包（图中黑实线），根据扫描宽度的变化（从M到ab逐渐增大接着逐渐减小至N），可知有一个起伏，在ab距离最大，取ab中点为山顶点。  A1 | 同样连接MN，此种情况下仅找到一个起伏，且最长为ab，山顶点为ab中点。    B1 | 根据扫描宽度的变化，可知有三个起伏（三个“花生米”），即有三个山顶点（分别是ab中点、ef中点、ij中点），在两个“最细处”（扫描线宽度值最小）取一个扫描线的中心即是它的鞍部点（分别是cd中点、gh中点）    C1 |
| A2 | B2 | C2 |
| A | B | C |

图3-13

鞍部点一般的寻找方法，除了以上隐含在一个封闭曲线里的鞍部点(如图3-13C)以外，实际地形复杂多变如图3-14，这类鞍部点如何确定？接下来有详细介绍：

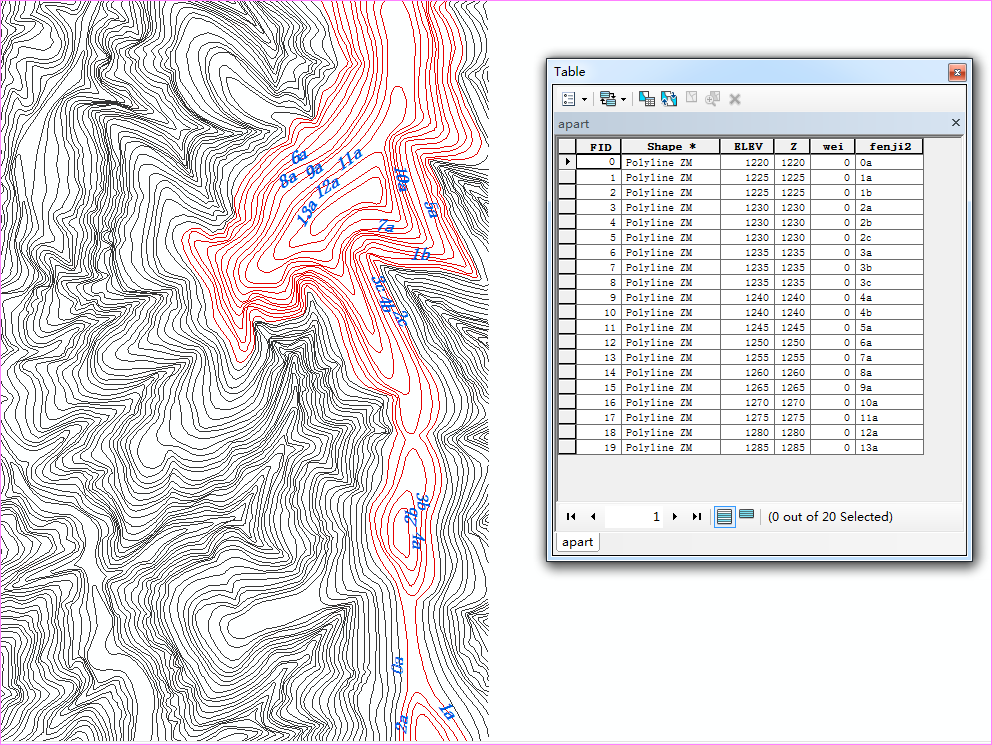
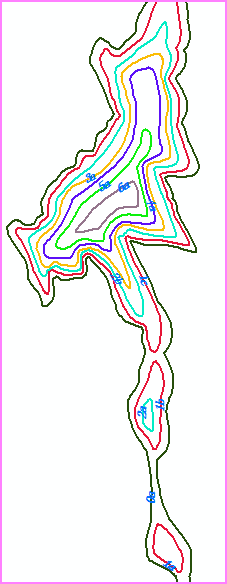


图3-14

作为一幅带有图框的等高线地图，可以从等高线的高程由低到高组织成父结点与子结点，子结点又有其子结点的一对多的树状结构。如下图3-15是将原始等高线数据按高程进行分类分级渲染，高程相等的作为兄弟等高线设置同一颜色，“亲兄弟”等高线之间必有鞍部点，“堂兄弟”等高线间便没有鞍部点，3-16是根据父子关系建立的数据结构。



4a

6a

0a

1b

1c

1a

2a

2b

3a

5a

图3-15是等高线图，图3-16是其对应的父子数据结构图 ，“亲兄弟”等高线之间必有鞍部点（如1a和1b、1b和1c之间必各有一个鞍部点），叶子结点中必有一个山顶点。再根据以上核心算法求出鞍部点、山顶点的具体位置即可。

图3-15 图3-16

### 3.2.2山顶点、洼底点、鞍部点高程的推算

山顶点、洼底点、鞍部点高程的推算可以用八方向扫描或者用十六方向扫描，为了算法的精度和效率本论文用八方向扫描法（在此用矢量扫面线）如下图3-17所示：

|  |  |
| --- | --- |
| A地形1 | D地形2 |
| B ④⑧方向为A的纵切面 | E ④⑧方向为D的纵切面 |
| C ②⑥方向为A的纵切 | F ②⑥方向为D的纵切面 |

图3-17

图3-17A地形1按照八方向扫描，从八个方向的纵坡面中找到距离最大和最小的对立方向构建纵坡面，得到④⑧方向距离最小交等高线于A、B、C、A′、B′、C′，且其平面坐标就是其与等高线对应交点的坐标，图3-17B是其横剖面图，②⑥方向距离最大交等高线于DEF D′E′F′且坐标亦知，坡面图如图3-17C。在图3-17B和图3-17C中分别借助三点求导三次多项式光滑插值算法思想可得山顶点O的不同的两个高程，再求加权平均值即可得到O点的最终三维坐标值。

鞍部点也同山顶点按照八方向扫描最终获得鞍部点的三维坐标值，如图3-17（D、E、F）。

## 3.3 小结

本章主要介绍的是算法设计与研究内容。分为两大块：一块介绍脊、谷线处特征点识别与计算算法，一块介绍山顶点、洼底点、鞍部点位置与高程的推算，做到图文并茂，部分算法还有待于进一步研究与细化。

# 4 系统测试

## 4.1 实验样区概况

论文以安徽和黄土地貌部分区域为例，其基本概况如下：

安徽地形类型多种多样，平原丘陵山地都有分布；地势东南部分地势较高，东部为黄淮平原，地势低平，大致为南高北低。

黄土地貌是在黄土堆积过程中遭受强烈侵蚀的产物。黄土地貌的特征是：① 沟谷众多、地面破碎。 ② 侵蚀方式独特、过程迅速。 ③ 沟道流域内有多级地形面。

## 4.2 实验结果分析

本论文理论与实践相结合，使用ArcGIS Engine开发包，Visual Studio2010开发环境，采用C#语言进行程序开发，编程实现了：

1. 等高线光滑结果——三点求导三次多项式算法
2. 统一了等高线数字化走向
3. 正负地貌特征点自动提取。

而这些正是本文的主要创新点。

### 4.2.1等高线光滑结果

图4-1中同一块局部数据（局部1）的a是光滑前等高线，b是光滑后等高线，c、d是另外的同一块数据（局部2）的光滑前、后等高线。e是局部1光滑前后的等高线（a、b）叠加对比，f是局部2（c、d）叠加对比。从图中可以看到此光滑算法得到的等高线数据，没有改变原始数据点的情况下还能有比较漂亮的弧度，比ArcGIS里的光滑工具先进很多（此光滑算法没有出现偏移，每一个原始数据都没有挪动或删除）。

|  |  |
| --- | --- |
|  |  |
| a光滑前等高线（局部1） | b 光滑后等高线（局部1） |
|  |  |
| c光滑前等高线（局部2） | d光滑后等高线（局部2） |

|  |  |
| --- | --- |
|  |  |
| e光滑前后叠加对比（黑色是光滑前，红色是光滑后） | e光滑前后叠加对比（黑色是光滑前，红色是光滑后） |

图4-1光滑前后等高线对比

### 4.2.2统一了等高线数字化走向

图4-2中的a是没有统一数字化走向之前的一根不符合左高右低的等高线（绿色点为起点，红色点为终点，图b同样），图b是统一好数字化方向的符合左高右低的规则的，这里只拿一根做对比，所有的等高线最终处理后都是符合左高右低要求的。

|  |  |
| --- | --- |
|  |  |
| a统一方向前 | b 统一方向后 |

图4-2统一等高线数字化走向前后对比对比

### 4.2.3正负地貌特征点自动提取

图4-3a是原始安徽部分等高线数据，b是最终提取的正负地貌特征点数据，其中蓝色是负向地貌，红色是正向地貌。

|  |  |
| --- | --- |
|  |  |
| a安徽等高线 | b 安徽正负地貌特征点 |

图4-3安徽等高线及其正负地貌特征点

图4-4中的a是原始黄土地貌部分数据，图b是最终得到的正负地貌特征点数据，蓝色是负向地貌特征点，红色是正向地貌特征点。

|  |
| --- |
|  |
| a黄土地貌等高线 |
|  |
| b 黄土地貌正负地貌特征点 |

图4-4黄土地貌等高线及其正负地貌特征点

## 4.3 小结

本章主要介绍了两方面内容：实验样区概况和实验结果分析。实验样区主要以安徽等高线和黄土地貌等高线数据，本论文对正负地貌特征点自动化提取的理论加实验研究最终得到比较良好的实验结果，在达到最终目的之前进行了的三点求导三次多项式算法光滑的等高线要比ArcGIS本身的光滑工具光滑效果更好，这是一个创新之处，对于统一数字化方向是ArcGIS本身没有的，从而也是一个比较大的创新点，本文最主要也是最大的创新点是利用等高线数据自动提取地貌特征点，从而为下一步自动提取地貌结构线打下了坚实的基础（（ArcGIS里没有自动提取地貌结构线的功能）。

# 5 结论与讨论

## 5.1 主要成果

本论文通过理论依据—>实验验证—>理论分析—>优化实验……最终得到比较令人满意的结果

1、三点求导三次多项式算法光滑的等高线要比ArcGIS本身的光滑工具光滑效果更好，因为此方法光滑的等高线是不改动原始数据上的所有原始点数据，而ArcGIS中的光滑工具则会导致原始数据偏移、失真。

2、数字化规范是左高右低，而在实际生产过程不能保证数字制图员都能按规范办事，ArcGIS又没有相应的软件来统一数字化方向，所以本论文实现的统一方向则弥补了这一点疏忽，从而使得软件对于正负向地貌特征点的识别可以更为正确、更为简单。

3. 本文利用等高线数据，自动提取地貌特征点，从而为下一步自动提取地貌结构线打下了坚实的基础。

## 5.2 研究展望

本论文虽然取得了上述成果，但还不尽完美，有一些问题尚待进一步研究与实验：

由于时间和精力有限，本论文暂且只对山脊、山谷处特征点提取做了理论和实验的研究，对于山顶点、洼底点、鞍部点的提取，目前只有理论研究，还有待于进一步实验。

# 参考文献

1. 张尧，樊红，李玉娥.一种基于等高线的地形特征线提取方法.测绘学报.2013
2. 吴艳兰，胡鹏.地貌综合的地图代数模型和方法研究.2004. [博士论文].武汉：武汉大学.
3. 曲均浩，程久龙，崔先国.垂直剖面法自动提取山脊线和山谷线. 测绘学报.2007
4. 黄培之.提取山脊线和山谷线的一种新方法.武汉:武汉大学学报(信息科学版).Vol.26(3).2001.PP.247~252
5. 倪星航.2008.基于规则格网DEM提取地形特征线的方法研究.[硕士论文]. 西南交通大学
6. 陈婷， 周汝良， 朱大运，黄钰.2011基于DEM的2种提取地形特征线算法对比研究. 林业调查规划
7. 周毅， 汤国安， 张婷， 王春.2007基于格网DEM线状分析窗口的地形特征线快速提取方法. 测绘通报
8. 汝绪伟.2004.基于等高线数据的地形特征线提取及地形建模研究. [硕士论文].山东科技大学.
9. 查正军.2007.基于数字高程模型（DEM）的地形特征提取. [硕士论文].同济大学.
10. 陈晓勇.数学形态学理论和模型的若干扩展及在ccD扫描等高线图的DEM自动建立中的应用.1990.[博士论文].武汉:武汉测绘科技大学
11. 冯桂.2001.数字高程模型内插中等高线保形方法的研究. [博士论文].武汉:武汉大学
12. F Attneave. Some Informational Aspects of Visual Percep- tion[J].Psycholo.Rev.，1954，61( 3) : 183-193.
13. Plazanet C. Measurement Characterization and Classifica- tion for Automated Line Feature Generalization［C］/ / Pro- ceedings of AutoCarto，1995，12: 59-68.
14. 常会，柴华彬，邹友峰 . 基于搜素圆法的等值线追踪技术［J］． 测绘科学，2009，34( 2) : 119-120.

# 致 谢

毕业论文就这样完成了，四年的读书生活在这个季节即将划上一个句号，而对于我的人生却只是一个逗号。四年的求学生涯在师长、亲人朋友的陪伴下，走得太快太匆忙。离别之际，心中有太多的不舍，那么多可爱的人值得我去感谢。

首先，我要感谢我的指导老师费立凡老师，他是我唯一一个接触时间最长的老师，在科研上他是我师亦是我友，在生活上他既像是我母亲又像是我父亲，在他面前从来不会感觉到任何压力，每一次的科研探讨都被他称作“脑风暴”，他说在他面前讨论科研不分辈分，面对问题大家都有同样的发言权，教学相长，费老师经常邀请我们到他家吃饭，每次都会摆上丰盛的一桌饭菜，最佩服的是他那科研精神，饭前没讨论完的科研任务吃饭时也不罢休，直到达到毫无疑问的境界为止，而且从来没有不耐烦的时候！他渊博的专业知识，严谨的治学态度，诲人不倦的高尚师德，平易近人的人格魅力对我影响深远，不仅使我树立了远大的学术目标、掌握了基本的研究方法，还使我明白了许多待人接物与为人处事的道理。论文从选题到完成，占用了老师不少的工作和休息时间，在此谨向费老师表示崇高的敬意和衷心的感谢！

其次，也要感谢班主任赵来老师，感谢他对我们日常生活和学习无微不至的关怀，四年里为学生为班级付出的心血我们不会忘记！感谢刘民士老师、刘东良老师、邓凯老师、王本林老师、蒋文明老师……，感谢他们在我的开发路上指点迷津，一路陪伴着，让我在开发的路上不至于太迷茫，感谢地理信息科学系其他老师，虽然他们没有直接参与我的论文指导，但在开题时也给我提供了不少的意见，提出了一系列可行性建议，最后非常感激陈来顺同学，平时与我深入探讨加速了论文的进展，在我生病身体不适无力继续写程序的时候帮助我完善程序，在此向他们表示深深的感谢！

最后，感谢陪我走过大学四年的各位同学，在我遇到困难的时候，是你们无私的帮助让我渡过难关，我祝愿你们前程似锦！还要感谢我的父母、家人，他们对我始终如一的关爱和支持是我在人生道路上不断前行的强大动力，我祝愿他们永远健康快乐！

凌乱的记忆变作凌乱的文字，要感谢的人还有很多很多，在这里，我只能对你们说，谢谢！

祝你们幸福快乐、事业有成！

# 附 录：关键程序代码

1、UnifyDirection类

using System；

using System.Collections.Generic；

using System.Linq；

using System.Text；

using System.Windows.Forms；

using ESRI.ArcGIS.Carto；

using ESRI.ArcGIS.Geodatabase；

using ESRI.ArcGIS.Geometry；

using ESRI.ArcGIS.DataSourcesRaster；

using ESRI.ArcGIS.DataSourcesGDB；

using ESRI.ArcGIS.esriSystem；

using ESRI.ArcGIS.ArcMapUI；

namespace FeaturePointDataModel

{

public class UnifyDirection

{

private IFeatureClass feaCla；

public IFeatureClass FeaCla

{

get { return feaCla； }

set { feaCla = value； }

}

public UnifyDirection(IFeatureClass cfeaCla)

{

feaCla = cfeaCla；

}

/// <summary>

/// 获取闭合等高线中面积由小到大的feature的FID

/// </summary>

/// <param name="feacla"></param>

/// <returns></returns>

public List<int> getClosed()

{

List<int> oidList = new List<int>()；

SortedDictionary<double， int> sortFidDic = new SortedDictionary<double， int>()；

AddField(feaCla， "Mark")；//添加标记字段

IFeatureCursor pfcurse = feaCla.Search(null， false)；

IFeature pFeature = pfcurse.NextFeature()；

while (pFeature != null)

{

IPolyline pLine = pFeature.Shape as IPolyline；

if (pLine.IsClosed)

{

pFeature.set\_Value(pFeature.Fields.FindField("Mark")， 1)；//将此标记为1已遍历

pFeature.Store()；//保存

IPointCollection pc = pLine as IPointCollection；

IPolygon pgn = MapHelp.polylineToPolygon(pc)；

IArea area = pgn as IArea；

double d\_area = area.Area；

sortFidDic.Add(d\_area， pFeature.OID)；

}

pFeature = pfcurse.NextFeature()；

}

foreach (var item in sortFidDic)//将面积由大到小的feature的FID加入到集合中

{

oidList.Add(item.Value)；

}

MapHelp.maxValue = oidList.Count；

return oidList；

}

/// <summary>

/// 调整等高线方向

/// </summary>

/// <param name="feaCla"></param>

/// <param name="oidList"></param>

public void Reverse(List<int> oidList， bool leftRight)

{

List<int> isLookList = new List<int>()；

/\*图框4个边角点\*/

IPoint p1 = Opt(feaCla).ElementAt(0)；

IPoint p2 = Opt(feaCla).ElementAt(1)；

IPoint p3 = Opt(feaCla).ElementAt(2)；

IPoint p4 = Opt(feaCla).ElementAt(3)；

try

{

for (int i = 0； i < oidList.Count； i++)

{

MapHelp.value = i + 1；

IFeature fea = feaCla.GetFeature(oidList.ElementAt(i))；

IPolyline pl = fea.Shape as IPolyline；

IPointCollection pointCollection = pl as IPointCollection；

IPoint fp = pointCollection.get\_Point(0)；

IPoint tp = new PointClass()；

if (leftRight)//向右探测线

{

fp.X = fp.X + 0.1；//探测线起点x值，y值不变

tp.Y = fp.Y；//探测线终点的y值

tp.X = p2.X；//探测线终点的x值

}

else//向下探测线

{

fp.Y = fp.Y - 0.1；

tp.Y = p3.Y；

tp.X = fp.X；

}

IPolyline line = new PolylineClass()；//探测线

line.FromPoint = fp；

line.ToPoint = tp；

Dictionary<int， IPoint> dic = new Dictionary<int， IPoint>()；//探测线与等高线相交于某条等高线的某个点

SortedDictionary<double， int> dictanceSort = new SortedDictionary<double， int>()；//根据距离排序相交等高线的FID

dic = getPtCollection(feaCla， line)；

foreach (var item in dic)

{

IPolyline p\_line = new PolylineClass()；

p\_line.FromPoint = fp；

p\_line.ToPoint = item.Value；

double dictance = p\_line.Length；

dictanceSort.Add(dictance， item.Key)；

}

double origin = fp.Z；

for (int j = 0； j < dictanceSort.Count； j++)

{

if (getZ(feaCla.GetFeature(dictanceSort.ElementAt(j).Value)) < origin)//山顶圆圈线

{

IFeature p\_fea = feaCla.GetFeature(dictanceSort.ElementAt(j).Value)；

foreach (var item in isLookList)

{

if (item == p\_fea.OID)

{

continue；

}

}

p\_fea.set\_Value(p\_fea.Fields.FindField("Mark")， 1)；//遍历后就标记等高线

unify(fp， p1， p2， p3， p4， p\_fea， pl， true)；

isLookList.Add(p\_fea.OID)；

p\_fea.Store()；

origin = getZ(feaCla.GetFeature(dictanceSort.ElementAt(j).Value))；

}

else if (getZ(feaCla.GetFeature(dictanceSort.ElementAt(j).Value)) > origin)//则为洼地圆圈

{

IFeature p\_fea = feaCla.GetFeature(dictanceSort.ElementAt(j).Value)；

foreach (var item in isLookList)

{

if (item == p\_fea.OID)

{

continue；

}

}

unify(fp， p1， p2， p3， p4， p\_fea， pl， true)；

p\_fea.set\_Value(p\_fea.Fields.FindField("Mark")， 1)；

isLookList.Add(p\_fea.OID)；

p\_fea.Store()；

origin = getZ(feaCla.GetFeature(dictanceSort.ElementAt(j).Value))；

}

else if (getZ(feaCla.GetFeature(dictanceSort.ElementAt(j).Value)) == getZ(fea))

{

continue；

}

else

{

break；

}

}

}

}

catch (Exception err)

{

MessageBox.Show(err.Message)；

}

}

/// <summary>

/// 所有等高线与探测线的交点集合

/// </summary>

/// <param name="feaCla"></param>

/// <param name="line"></param>

/// <returns></returns>

private Dictionary<int， IPoint> getPtCollection(IFeatureClass feaCla， IPolyline line)

{

IFeatureCursor pfcurse = feaCla.Search(null， false)；

IFeature pFeature = pfcurse.NextFeature()；

Dictionary<int， IPoint> dic = new Dictionary<int， IPoint>()；

while (pFeature != null)

{

IPolyline pl = pFeature.Shape as IPolyline；

IPointCollection pc = pl as IPointCollection；

IPoint pt = getMouPt(pl， line)；

if (pt != null)

{

dic.Add(pFeature.OID， pt)；

pt.Z = pc.get\_Point(0).Z；

}

pFeature = pfcurse.NextFeature()；

}

return dic；

}

/// <summary>

/// 遍历等高线时与探测线的第一个交点

/// </summary>

/// <param name="pl"></param>

/// <param name="line"></param>

/// <returns></returns>

private IPoint getMouPt(IPolyline pl， IPolyline line)

{

ITopologicalOperator topoOperator = pl as ITopologicalOperator；

IGeometry geo = topoOperator.Intersect(line， esriGeometryDimension.esriGeometry0Dimension)；

IPoint pt = new PointClass()；

if (!geo.IsEmpty)

{

IPointCollection pc = geo as IPointCollection；

pt = pc.get\_Point(0)；

return pt；

}

return null；

}

/// <summary>

///

/// </summary>

/// <param name="fea"></param>

/// <returns></returns>

private double getZ(IFeature fea)

{

IPolyline pl = fea.Shape as IPolyline；

IPointCollection pc = pl as IPointCollection；

IPoint fpt = pc.get\_Point(0)；

return fpt.Z；

}

/// <summary>

/// fp为山顶（洼地）圆圈线的起点，即栅格探测线的起点

/// extent为整个地图图幅的区域

/// pl为山顶（洼地）圆圈线

/// i为圆圈线中的第几条索引

/// </summary>

/// <param name="fp"></param>

/// <param name="fea"></param>

/// <param name="extent"></param>

/// <param name="pl"></param>

/// <param name="i"></param>

private void unify(IPoint fp， IPoint p1， IPoint p2， IPoint p3， IPoint p4， IFeature fea， IPolyline pl， bool isTop)

{

IFeature p\_fea = fea；

IPolyline p\_pl = p\_fea.Shape as IPolyline；//探测到的等高线

IPointCollection pc = p\_pl as IPointCollection；

IPoint fpt = pc.get\_Point(0)；

IPoint tpt = pc.get\_Point(pc.PointCount - 1)；

List<IPoint> vertexPointList = new List<IPoint>()；

List<IPoint> addPointList = new List<IPoint>()；

addPointList = MapHelp.addPoint(fpt， tpt， p1， p2， p3， p4)；

object missing = Type.Missing；

for (int f = 0； f < addPointList.Count； f++)

{

pc.AddPoint(addPointList.ElementAt(f)， ref missing， ref missing)；

}

pc.AddPoint(pc.get\_Point(0))；

if (isTop)//isTop为真，则是山顶

{

if (MapHelp.isClockWise(pl as IPointCollection) == false)//逆时针

{

isContain(p\_pl， pc， fp， addPointList， true)；

}

else

{

pl.ReverseOrientation()；//因为已经确定为山顶，所以必须是逆时针

isContain(p\_pl， pc， fp， addPointList， true)；

}

}

else//isTop为假，则是山洼地

{

if (MapHelp.isClockWise(pl as IPointCollection))//顺时针

{

isContain(p\_pl， pc， fp， addPointList， false)；

}

else

{

pl.ReverseOrientation()；//因为已经确定为山洼地，所以必须是顺时针

isContain(p\_pl， pc， fp， addPointList， false)；

}

}

}

/// <summary>

/// pl为探测线

/// pc为探测到的线的点集合

/// fp为山顶（洼地）圆圈线的起点，即栅格探测线的起点

/// addPointList为探测到线上添加的点，以便形成一个闭合区域，进行包含判断

/// </summary>

/// <param name="pl"></param>

/// <param name="pc"></param>

/// <param name="fp"></param>

/// <param name="addPointList"></param>

private void isContain(IPolyline pl， IPointCollection pc， IPoint fp， List<IPoint> addPointList， bool isTop)

{

IPolygon polygon = MapHelp.polylineToPolygon(pc)；

IRelationalOperator pRelOpr = polygon as IRelationalOperator；

if (pRelOpr.Contains(fp))//包含山顶或山洼地圆圈

{

if (isTop)//包含最里面山顶圆圈，则探测到的等高线就该是逆时针

{

if (MapHelp.isClockWise(pc))//顺时针，则逆置

{

//删除添加的点（添加点是为了构成包围区域进行包围判断，包围判断已有，则要删除添加的多余点）

pc.RemovePoints(pc.PointCount - addPointList.Count - 1， addPointList.Count + 1)；

pl.ReverseOrientation()；

}

else

{

pc.RemovePoints(pc.PointCount - addPointList.Count - 1， addPointList.Count + 1)；

}

}

else//包含最里面山洼地圆圈，则探测到的等高线就该是顺时针

{

if (MapHelp.isClockWise(pc) == false)//逆时针，则逆置

{

pc.RemovePoints(pc.PointCount - addPointList.Count - 1， addPointList.Count + 1)；

pl.ReverseOrientation()；

}

else

{

pc.RemovePoints(pc.PointCount - addPointList.Count - 1， addPointList.Count + 1)；

}

}

}

else//若不包

{

if (isTop)//若不包含山顶圆圈，则探测到的等高线就该是顺时针

{

if (MapHelp.isClockWise(pc))//顺时针则是对的

{

pc.RemovePoints(pc.PointCount - addPointList.Count - 1， addPointList.Count + 1)；

}

else//反之逆置

{

pc.RemovePoints(pc.PointCount - addPointList.Count - 1， addPointList.Count + 1)；

pl.ReverseOrientation()；

}

}

else//若不包含山洼地圆圈，则探测到的等高线就该是逆时针

{

if (MapHelp.isClockWise(pc) == false)//逆时针则是对的

{

pc.RemovePoints(pc.PointCount - addPointList.Count - 1， addPointList.Count + 1)；

}

else//反之逆置

{

pc.RemovePoints(pc.PointCount - addPointList.Count - 1， addPointList.Count + 1)；

pl.ReverseOrientation()；

}

}

}

}

/// <summary>

/// 返回图框的的四个顶点

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private List<IPoint> Opt(IFeatureClass fea)

{

IPoint p1 = new PointClass()；

IPoint p2 = new PointClass()；

IPoint p3 = new PointClass()；

IPoint p4 = new PointClass()；

IFeatureClass pFeatureClass = fea；

IFeatureCursor pfcurse = pFeatureClass.Search(null， false)；

IFeature pFeature = pfcurse.NextFeature()；

int i = 0；

double \_xMax = 0；

double \_xMin = 0；

double \_yMax = 0；

double \_yMin = 0；

while (pFeature != null)

{

IPolyline pl = pFeature.Shape as IPolyline；

double xMax = pl.Envelope.XMax；

double xMin = pl.Envelope.XMin；

double yMax = pl.Envelope.YMax；

double yMin = pl.Envelope.YMin；

if (i == 0)

{

\_xMin = xMin；

\_yMin = yMin；

\_xMax = xMax；

\_yMax = yMax；

}

i++；

if (yMin < \_yMin)

{

\_yMin = yMin；

}

if (xMin < \_xMin)

{

\_xMin = xMin；

}

if (xMax > \_xMax)

{

\_xMax = xMax；

}

if (yMax > \_yMax)

{

\_yMax = yMax；

}

pFeature = pfcurse.NextFeature()；

}

List<IPoint> ptList = new List<IPoint>()；

p1.PutCoords(\_xMin， \_yMax)；

p2.PutCoords(\_xMax， \_yMax)；

p3.PutCoords(\_xMax， \_yMin)；

p4.PutCoords(\_xMin， \_yMin)；

ptList.Add(p1)；

ptList.Add(p2)；

ptList.Add(p3)；

ptList.Add(p4)；

return ptList；

}

/// <summary>

/// 添加字段

/// </summary>

/// <param name="pFeatureClass"></param>

/// <param name="name"></param>

private void AddField(IFeatureClass pFeatureClass， string name)

{

//若存在，则不需添加

if (pFeatureClass.Fields.FindField(name) > -1) return；

IField pField = new FieldClass()；

IFieldEdit pFieldEdit = pField as IFieldEdit；

pFieldEdit.AliasName\_2 = name；

pFieldEdit.Name\_2 = name；

pFieldEdit.Type\_2 = esriFieldType.esriFieldTypeInteger；

IClass pClass = pFeatureClass as IClass；

pClass.AddField(pField)；

}

}

}

2、FeaturePoint类

using System；

using System.Collections.Generic；

using System.Linq；

using System.Text；

using System.Windows.Forms；

using ESRI.ArcGIS.Carto；

using ESRI.ArcGIS.Geodatabase；

using ESRI.ArcGIS.Geometry；

using ESRI.ArcGIS.DataSourcesRaster；

using ESRI.ArcGIS.DataSourcesGDB；

using ESRI.ArcGIS.esriSystem；

using ESRI.ArcGIS.ArcMapUI；

namespace FeaturePointDataModel

{

public class FeaturePoint

{

private IMxDocument document；

private IFeatureClass feaCla；

private ISpatialReference spatialRef；

public IMxDocument Document

{

get { return document； }

set { document = value； }

}

public IFeatureClass FeaCla

{

get { return feaCla； }

set { feaCla = value； }

}

public ISpatialReference SpatialRef

{

get { return spatialRef； }

set { spatialRef = value； }

}

public FeaturePoint(IFeatureClass cfeaCla， ISpatialReference cspatialRef， IMxDocument cdocument)

{

feaCla = cfeaCla；

spatialRef = cspatialRef；

document = cdocument；

}

/// <summary>

/// 获取特征点集合

/// </summary>

/// <param name="feaCla"></param>

/// <returns></returns>

public void getPtShp(int tempAngle)

{

IFeatureCursor pfcurse = feaCla.Search(null， false)；

IFeature pFeature = pfcurse.NextFeature()；

List<IPoint> ptList = new List<IPoint>()；

IFeatureClass p\_feaPtCla = MapHelp.getFeaCla("testFeaPt"， MapHelp.OpenInMemoryWorkspace() as IFeatureWorkspace， spatialRef)；//特征点featureclass(数据集)

IFeatureClass inflection\_feaPtCla = MapHelp.getFeaCla("testInfle"， MapHelp.OpenInMemoryWorkspace() as IFeatureWorkspace， spatialRef)；//特征点featureclass(数据集)

while (pFeature != null)

{

Dictionary<int， IPoint> inflectionPtIndex = new Dictionary<int， IPoint>()；

Dictionary<int， double> dicList = new Dictionary<int， double>()；

IPolyline pl = pFeature.Shape as IPolyline；

IPointCollection pc = pl as IPointCollection；

List<double> differList = new List<double>()；//拐点的转角值

int count = 0；

for (int i = 0； i < pc.PointCount - 3； i++)

{

double differ1 = Angle(pc.get\_Point(i)， pc.get\_Point(i + 1)， pc.get\_Point(i + 2))；

double differ2 = Angle(pc.get\_Point(i + 1)， pc.get\_Point(i + 2)， pc.get\_Point(i + 3))；

if (differ1 \* differ2 <= 0)

{

differList.Add(differ2)；

inflectionPtIndex.Add(i + 2， pc.get\_Point(i + 2))；

}

else

{

count++；

}

}

IFeature p\_Feature = null；

IFeature infle\_Feature = null；

if (count == pc.PointCount - 3)//生成特征点features，添加（闭合而且始终是一个方向的特征点）的字段

{

int mAngleIndex = getMaxAngleIndex(pc)；//如果闭合而且始终是一个方向，则找最大的转角点

p\_Feature = p\_feaPtCla.CreateFeature()；//生成特征点feature

p\_Feature.Shape = pc.get\_Point(mAngleIndex)；

p\_Feature.set\_Value(1， pFeature.OID)；

p\_Feature.set\_Value(2， mAngleIndex)；

p\_Feature.set\_Value(3， Convert.ToInt32(pc.get\_Point(mAngleIndex).Z))；

p\_Feature.set\_Value(5， Angle(pc.get\_Point(mAngleIndex - 1)， pc.get\_Point(mAngleIndex)， pc.get\_Point(mAngleIndex + 1)) \* 180 / Math.PI)；

if (Angle(pc.get\_Point(mAngleIndex - 1)， pc.get\_Point(mAngleIndex)， pc.get\_Point(mAngleIndex + 1)) < 0)

{

p\_Feature.set\_Value(4， -1)；

}

else

{

p\_Feature.set\_Value(4， 1)；

}

p\_Feature.Store()；

}

for (int n = 0； n < inflectionPtIndex.Count - 1； n++)//每条线中 在每两个拐点找转角最大值

{

infle\_Feature = inflection\_feaPtCla.CreateFeature()；//生成拐点feature

infle\_Feature.Shape = pc.get\_Point(inflectionPtIndex.ElementAt(n).Key)；//生成features，给拐点featureclass添加字段

infle\_Feature.set\_Value(1， pFeature.OID)；

infle\_Feature.set\_Value(2， inflectionPtIndex.ElementAt(n).Key)；

infle\_Feature.set\_Value(3， pc.get\_Point(inflectionPtIndex.ElementAt(n).Key).Z)；

infle\_Feature.set\_Value(5， differList.ElementAt(n) \* 180 / Math.PI)；

if (differList.ElementAt(n + 1) \* 180 / Math.PI < 0)

{

infle\_Feature.set\_Value(4， -1)；

}

else

{

infle\_Feature.set\_Value(4， 1)；

}

infle\_Feature.Store()；

if (n == inflectionPtIndex.Count - 2)//生成最后一个拐点

{

infle\_Feature = inflection\_feaPtCla.CreateFeature()；

infle\_Feature.Shape = pc.get\_Point(inflectionPtIndex.ElementAt(n + 1).Key)；

infle\_Feature.set\_Value(1， pFeature.OID)；

infle\_Feature.set\_Value(2， inflectionPtIndex.ElementAt(n + 1).Key)；

infle\_Feature.set\_Value(3， pc.get\_Point(inflectionPtIndex.ElementAt(n + 1).Key).Z)；

infle\_Feature.set\_Value(5， Convert.ToInt32(differList.ElementAt(n + 1) \* 180 / Math.PI))；

if (differList.ElementAt(n + 1) \* 180 / Math.PI < 0)

{

infle\_Feature.set\_Value(4， -1)；

}

else

{

infle\_Feature.set\_Value(4， 1)；

}

infle\_Feature.Store()；

}

double negativeMinAgle = 0， positivemaxAgle = 0；

int negativePoIndex = 0；

bool negativeIsTure = false， positveIsTrue = false；

int frm = inflectionPtIndex.ElementAt(n).Key；

int to = inflectionPtIndex.ElementAt(n + 1).Key；

for (int m = frm； m < to - 2； m++)

{

double agle = Angle(pc.get\_Point(m)， pc.get\_Point(m + 1)， pc.get\_Point(m + 2))；

if (agle < negativeMinAgle)

{

negativeMinAgle = agle；

negativePoIndex = m + 1；

negativeIsTure = true；

}

else if (agle > positivemaxAgle)

{

positivemaxAgle = agle；

negativePoIndex = m + 1；

positveIsTrue = true；

}

}

if (negativeIsTure)

{

if (Math.Abs(negativeMinAgle \* 180 / Math.PI) >= tempAngle)

{

p\_Feature = p\_feaPtCla.CreateFeature()；//生成特征点feature

p\_Feature.Shape = pc.get\_Point(negativePoIndex)；

p\_Feature.set\_Value(1， pFeature.OID)；

p\_Feature.set\_Value(2， negativePoIndex)；

p\_Feature.set\_Value(3， Convert.ToInt32(pc.get\_Point(negativePoIndex).Z))；

p\_Feature.set\_Value(5， negativeMinAgle \* 180 / Math.PI)；

p\_Feature.set\_Value(4， -1)；

p\_Feature.Store()；

}

}

else if (positveIsTrue)

{

if (Math.Abs(positivemaxAgle \* 180 / Math.PI) >= tempAngle)

{

p\_Feature = p\_feaPtCla.CreateFeature()；//生成特征点feature

p\_Feature.Shape = pc.get\_Point(negativePoIndex)；

p\_Feature.set\_Value(1， pFeature.OID)；

p\_Feature.set\_Value(2， negativePoIndex)；

p\_Feature.set\_Value(3， pc.get\_Point(negativePoIndex).Z)；

p\_Feature.set\_Value(5， Convert.ToInt32(positivemaxAgle \* 180 / Math.PI))；

p\_Feature.set\_Value(4， 1)；

p\_Feature.Store()；

}

}

}

pFeature = pfcurse.NextFeature()；

}

//for (int f = 0； f < ptList.Count； f++)

//{

// IFeature p\_Feature = p\_feaPtCla.CreateFeature()；

// p\_Feature.Shape = ptList.ElementAt(f)；

// //p\_Feature.set\_Value(pFeature.Fields.FindField("LineId")， pFeature.OID)；

// //p\_Feature.set\_Value(pFeature.Fields.FindField("Number")， i + 1)；

// //p\_Feature.set\_Value(pFeature.Fields.FindField("Elev")， pc.get\_Point(i + 1).Z)；

// //p\_Feature.set\_Value(pFeature.Fields.FindField("CancaveType")， 1)；

// p\_Feature.Store()；

//}

MapHelp.AddFeatureClass2Map(p\_feaPtCla， document.FocusMap， "feaPt")；

MapHelp.AddFeatureClass2Map(inflection\_feaPtCla， document.FocusMap， "inflection")；

}

/// <summary>

/// 求转角angle

/// </summary>

/// <param name="p1"></param>

/// <param name="p2"></param>

/// <param name="p3"></param>

/// <returns></returns>

private double Angle(IPoint p1， IPoint p2， IPoint p3)

{

ILine line1 = new LineClass()；

ILine line2 = new LineClass()；

line1.PutCoords(p1， p2)；

line2.PutCoords(p2， p3)；

double angle1 = line1.Angle；

double angle2 = line2.Angle；

if (line1.Angle < 0)

{

angle1 = 2 \* Math.PI + angle1；

}

if (line2.Angle < 0)

{

angle2 = 2 \* Math.PI + angle2；

}

double differ = angle2 - angle1；

if (Math.Abs(differ) > Math.PI)

{

if (differ > 0)

{

differ = (differ - 2 \* Math.PI)；

}

else

{

differ = (differ + 2 \* Math.PI)；

}

}

return differ；

}

/// <summary>

/// 求最大转角的索引

/// </summary>

/// <param name="pc"></param>

/// <returns></returns>

private int getMaxAngleIndex(IPointCollection pc)

{

double maxAngle = 0；

int index = 0；

for (int i = 0； i < pc.PointCount - 2； i++)

{

double angle = Angle(pc.get\_Point(i)， pc.get\_Point(i + 1)， pc.get\_Point(i + 2))；

if (Math.Abs(angle) > Math.Abs(maxAngle))

{

maxAngle = angle；

index = i；

}

}

return index；

}

/// <summary>

/// reverse等高线方向时，判断等高线是否已经遍历

/// </summary>

/// <param name="table"></param>

/// <param name="oid"></param>

/// <returns></returns>

private bool isLook(ITable table， int oid， int fieldIndex)

{

IRow row = table.GetRow(oid)；

object mark = row.get\_Value(fieldIndex)；

if (Convert.ToInt16(mark) == 1)

{

return true；

}

else

{

return false；

}

}

}

}

3、MapHelp类

using System；

using System.Collections.Generic；

using System.Linq；

using System.Text；

using ESRI.ArcGIS.Carto；

using ESRI.ArcGIS.Geodatabase；

using ESRI.ArcGIS.Geometry；

using ESRI.ArcGIS.DataSourcesRaster；

using System.IO；

using ESRI.ArcGIS.DataSourcesFile；

using ESRI.ArcGIS.DataSourcesGDB；

using ESRI.ArcGIS.esriSystem；

namespace FeaturePointDataModel

{

public static class MapHelp

{

public static int value = 0；

public static int maxValue = 0；

/// <summary>

/// 获取某行某列的栅格值

/// </summary>

/// <param name="rasterLayer"></param>

/// <param name="row"></param>

/// <param name="col"></param>

/// <returns></returns>

public static object rasterValue(IRasterLayer rasterLayer， int row， int col)

{

IRaster clipRaster = rasterLayer.Raster；

IRasterProps rasterProps = (IRasterProps)clipRaster；

int dHeight = rasterProps.Height；//当前栅格数据集的行数

int dWidth = rasterProps.Width； //当前栅格数据集的列数

double dX = rasterProps.MeanCellSize().X； //栅格的宽度

double dY = rasterProps.MeanCellSize().Y； //栅格的高度

IEnvelope extent = rasterProps.Extent； //当前栅格数据集的范围

rstPixelType pixelType = rasterProps.PixelType； //当前栅格像素类型

IPnt pntSize = new PntClass()；

pntSize.SetCoords(dX， dY)；//扫面像素块的大小

IPixelBlock pixelBlock = clipRaster.CreatePixelBlock(pntSize)；//创建pixelBlock像素块

IPnt pnt = new PntClass()；

pnt.SetCoords(row， col)；//查找的位置即某行某列

clipRaster.Read(pnt， pixelBlock)；

if (pixelBlock != null)

{

object obj = pixelBlock.GetVal(0， 0， 0)；

if (obj != null)

{

return obj；

}

else

{

return null；

}

}

return null；

}

/// <summary>

/// polyline转化为polygon

/// </summary>

/// <param name="pc"></param>

/// <returns></returns>

public static IPolygon polylineToPolygon(IPointCollection pc)

{

Ring ring = new RingClass()；

object missing = Type.Missing；

for (int i = 0； i < pc.PointCount； i++)

{

ring.AddPoint(pc.get\_Point(i)， ref missing， ref missing)；

}

IGeometryCollection pointPolygon = new PolygonClass()；

pointPolygon.AddGeometry(ring as IGeometry， ref missing， ref missing)；

IPolygon polyGonGeo = pointPolygon as IPolygon；

return polyGonGeo；

}

/// <summary>

/// 链状结构形成包围关系

/// </summary>

/// <param name="fpt"></param>

/// <param name="tpt"></param>

/// <param name="p1"></param>

/// <param name="p2"></param>

/// <param name="p3"></param>

/// <param name="p4"></param>

/// <returns></returns>

public static List<IPoint> addPoint(IPoint fpt， IPoint tpt， IPoint p1， IPoint p2， IPoint p3， IPoint p4)

{

List<IPoint> ptList = new List<IPoint>()；

ptList.Add(p1)；

ptList.Add(p2)；

ptList.Add(p3)；

ptList.Add(p4)；

ptList.Add(p1)；

//起点入集合

if ((fpt.X >= p1.X && fpt.X < p2.X))

{

ptList.Insert(1， fpt)；

}

else if ((fpt.Y >= p3.Y && fpt.Y < p2.Y))

{

ptList.Insert(2， fpt)；

}

else if ((fpt.X >= p4.X && fpt.X < p3.X))

{

ptList.Insert(3， fpt)；

}

else if ((fpt.Y >= p4.Y && fpt.Y < p1.Y))

{

ptList.Insert(4， fpt)；

}

//末点入集合

if ((tpt.X >= p1.X && tpt.X < p2.X))

{

ptList.Insert(1， tpt)；

}

else if ((tpt.Y >= p3.Y && tpt.Y < p2.Y))

{

ptList.Insert(2， tpt)；

}

else if ((tpt.X >= p4.X && tpt.X < p3.X))

{

ptList.Insert(3， tpt)；

}

else if ((tpt.Y >= p4.Y && tpt.Y < p1.Y))

{

ptList.Insert(4， tpt)；

}

List<IPoint> outList = new List<IPoint>()；

int index = ptList.IndexOf(tpt)；//找到终点所在索引

index++；

for (； index < ptList.Count； index++)

{

if (ptList.ElementAt(index) == p1)//遇到p1进入下一个循环

{

index = 0；

outList.Add(ptList.ElementAt(0))；

}

else if (ptList.ElementAt(index) != fpt)//没有遇到起点就接着遍历

{

outList.Add(ptList.ElementAt(index))；

}

else

{

break；

}

}

return outList；

}

/// <summary>

/// 判断包围的区域面积是顺时针

/// </summary>

/// <param name="pc"></param>

/// <returns></returns>

public static bool isClockWise(IPointCollection pc)

{

IPointCollection ring = new RingClass() as IPointCollection；

IRing rng = ring as IRing；

for (int i = 0； i < pc.PointCount； i++)

{

ring.AddPoint(pc.get\_Point(i))；

}

if (rng.IsExterior)

{

return true；

}

else

{

return false；

}

}

/// <summary>

/// 建立数据集featuresclass

/// </summary>

/// <param name="name"></param>

/// <param name="pFWS"></param>

/// <param name="SpatialRef"></param>

/// <returns></returns>

public static IFeatureClass getFeaCla(string name， IFeatureWorkspace pFWS， ISpatialReference SpatialRef)

{

//string strShapeFolder = "C:/"；

//string strShapeFile = "test.shp"；

//string shapeFileFullName = strShapeFolder + strShapeFile；

//IWorkspaceFactory pWorkspaceFactory = new ShapefileWorkspaceFactory()；

//IFeatureWorkspace pFeatureWorkspace = (IFeatureWorkspace)pWorkspaceFactory.OpenFromFile(strShapeFolder， 0)；

IFeatureClass pFeatureClass；

//if (File.Exists(shapeFileFullName))

//{

// pFeatureClass = pFeatureWorkspace.OpenFeatureClass(strShapeFile)；

// IDataset pDataset = (IDataset)pFeatureClass；

// pDataset.Delete()；

//}

IFields pFields = new FieldsClass()；

IFieldsEdit pFieldsEdit = (IFieldsEdit)pFields；

IField pField = new FieldClass()；

IFieldEdit pFieldEdit = (IFieldEdit)pField；

pFieldEdit.Name\_2 = "SHAPE"；

pFieldEdit.Type\_2 = esriFieldType.esriFieldTypeGeometry；

IGeometryDefEdit pGeoDef = new GeometryDefClass()；

IGeometryDefEdit pGeoDefEdit = (IGeometryDefEdit)pGeoDef；

pGeoDefEdit.GeometryType\_2 = esriGeometryType.esriGeometryPoint；

pGeoDefEdit.SpatialReference\_2 = SpatialRef； //new UnknownCoordinateSystemClass()；

pGeoDefEdit.HasM\_2 = false；

pGeoDefEdit.HasZ\_2 = true；

pFieldEdit.GeometryDef\_2 = pGeoDef；

pFieldsEdit.AddField(pField)；

pField = new FieldClass()；

pFieldEdit = (IFieldEdit)pField；

pFieldEdit.Name\_2 = "LineId"；

pFieldEdit.Type\_2 = esriFieldType.esriFieldTypeInteger；

pFieldsEdit.AddField(pField)；

pField = new FieldClass()；

pFieldEdit = (IFieldEdit)pField；

pFieldEdit.Name\_2 = "Number"；

pFieldEdit.Type\_2 = esriFieldType.esriFieldTypeInteger；

pFieldsEdit.AddField(pField)；

pField = new FieldClass()；

pFieldEdit = (IFieldEdit)pField；

pFieldEdit.Name\_2 = "Elev"；

pFieldEdit.Type\_2 = esriFieldType.esriFieldTypeInteger；

pFieldsEdit.AddField(pField)；

pField = new FieldClass()；

pFieldEdit = (IFieldEdit)pField；

pFieldEdit.Name\_2 = "CancaveType"；

pFieldEdit.Type\_2 = esriFieldType.esriFieldTypeInteger；

pFieldsEdit.AddField(pField)；

pField = new FieldClass()；

pFieldEdit = (IFieldEdit)pField；

pFieldEdit.Name\_2 = "Angle"；

pFieldEdit.Type\_2 = esriFieldType.esriFieldTypeSingle；

pFieldsEdit.AddField(pField)；

pFeatureClass = pFWS.CreateFeatureClass(name， pFields， null， null， esriFeatureType.esriFTSimple， "SHAPE"， "")；

//for (int i = 0； i < ptList.Count； i++)

//{

// IFeature pFeature = pFeatureClass.CreateFeature()；

// pFeature.Shape = ptList.ElementAt(i)；

// //pFeature.set\_Value(pFeature.Fields.FindField("ID")， "D-1")；

//pFeature.set\_Value(pFeature.Fields.FindField("Pixels")， 1)；

// pFeature.Store()；

//}

return pFeatureClass；

//IFeatureLayer pFeaturelayer = new FeatureLayerClass()；

//pFeaturelayer.FeatureClass = pFeatureClass；

//pFeaturelayer.Name = "layer"；

//axMapControl1.AddLayer(pFeaturelayer)；

}

/// <summary>

/// 将数据集添加到地图上

/// </summary>

/// <param name="fc"></param>

/// <param name="map"></param>

public static void AddFeatureClass2Map(IFeatureClass fc， IMap map， string layerName)

{

IFeatureLayer layer = new FeatureLayerClass()；

layer.FeatureClass = fc；

layer.Name = layerName；

map.AddLayer(layer)；

}

/// <summary>

/// //创建临时工作空间，用于存放临时数据

/// </summary>

/// <returns></returns>

public static IWorkspace OpenInMemoryWorkspace()

{

IWorkspaceFactory workspaceFactory = new InMemoryWorkspaceFactoryClass()；

// Create a new in-memory workspace. This returns a name object.

IWorkspaceName workspaceName = workspaceFactory.Create(null， "MyWorkspace"， null， 0)；

IName name = (IName)workspaceName；

// Open the workspace through the name object.

IWorkspace workspace = (IWorkspace)name.Open()；

return workspace；

}

}

}