|  |
| --- |
|  |
| 高级地理信息系统 |
| **编程作业：**  **主要源程序代码及简要说明** |

姓 名：孙奇

学 号：1701210185

院 系：地球与空间科学学院

专 业：地图学与地理信息系统

二〇一八年七月

主要源程序代码目录

目录

[ContourPolyline.cs 1](#_Toc518489630)

[ContourPolylineSet.cs 3](#_Toc518489631)

[DataPoint.cs 4](#_Toc518489632)

[Edge.cs 6](#_Toc518489633)

[EdgeSet.cs 9](#_Toc518489634)

[MBR.cs 13](#_Toc518489635)

[PointSet.cs 14](#_Toc518489636)

[TopoPoint.cs 17](#_Toc518489637)

[TopoPointSet.cs 20](#_Toc518489638)

[TopoPolygon.cs 26](#_Toc518489639)

[TopoPolygonSet.cs 29](#_Toc518489640)

[TopoPolyline.cs 30](#_Toc518489641)

[TopoPolylineSet.cs 33](#_Toc518489642)

[Triangle.cs 35](#_Toc518489643)

[TriangleSet.cs 37](#_Toc518489644)

[Vector2D.cs 38](#_Toc518489645)

[CreateTIN.cs 39](#_Toc518489646)

[GridCreateContourLine.cs 49](#_Toc518489647)

[GridInterpolation.cs 53](#_Toc518489648)

[AgisControl 55](#_Toc518489649)

[MainForm.cs 61](#_Toc518489650)

[END 82](#_Toc518489651)

主要源程序代码说明

本程序基于.NET Framework 4.5.2架构，采用C#语言进行开发。考虑到要进行一定的用户交互，程序基于Windows窗体应用开发。

程序主要分为四个部分，DataStruct，Method，AgisControl，Forms。本文只列举了其中最重要的部分，且文中也不包含IDE自动生成的代码。

DataStruct，即数据结构。该部分规定了数据在内存中的组织结构。相关的CS文件从第1页至第38页。数据点相关的类DataPoint，PointSet，MBR为较基础的类，从文件中读取数据并有效组织。Edge，ContourPolyline为其衍生的类，用于储存计算过程中产生的边或等值折线。Triangle类为TIN模型中的基础类，用以储存三角网。TopoPoint，TopoPolyline，TopoPolygon分别对应拓扑关系中的点线面。Vector2D用来进行一些向量计算。

Method，即计算方法，该部分对一些计算方法进行了一定的封装。相关的CS文件从第39页至54页。CreateTIN是对TIN自动生成的封装，GridCreateContourLine是对格网等值线自动追踪的封装，GridInterpolation则是对两种不同的格网插值算法的封装。

AgisControl，是为显示数据和与用户交互专门开发的控件。主要实现显示图像的缩放、漫游、转换等操作，并针对不同的操作模式下的用户操作进行相应的反馈。

Forms，是与用户进行交互的窗体，最主要的MainForm，即主窗体。它是用户主要操作的界面，搭载了AgisControl控件，使得用户通过鼠标操作或菜单栏进行相应的操作。其他Form如输入读取文件路径，输入等值线间隔等窗体限于篇幅并没有列出。

以上四部分构成了该程序的主要框架，该项目可维护性指数为71，圈复杂度886，类耦合度104，代码度量值行数为2834，项目的在线地址是<https://github.com/Qi-Sun/AGIS-Task> 。后文是主要部分相应文件的源代码。

# ContourPolyline.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. //等值线
10. **public** **class** ContourPolyline
11. {
12. **public** **int** PID { **get**; **private** **set**; }   //唯一标识码
13. **private** **static** **int** \_pid = 777777;
14. **public** List<DataPoint> PointList = **new** List<DataPoint>();//点序列
15. **public** ContourPolyline() { **this**.PID = \_pid++; }
16. **public** ContourPolyline(DataPoint[] points)
17. {
18. **this**.PointList.AddRange(points);
19. **this**.PID = \_pid++;
20. }
21. //获取折线与边的交点
22. **public** **static** Object[] IntersectResult(ContourPolyline pl1, Edge edge)
23. {
24. List<ContourPolyline> sublineFromPL1 = **new** List<ContourPolyline>();
25. List<Edge> suEdgeFromEdge = **new** List<Edge>();
26. //对边上点排序
27. List<DataPoint> subEdgePoint = **new** List<DataPoint>();
28. subEdgePoint.Add(edge.StartPoint);
29. subEdgePoint.Add(edge.EndPoint);
30. edge.StartPoint.RelativeLoc = 0;
31. edge.EndPoint.RelativeLoc = 1;
32. //对折线上点排序
33. List<DataPoint> subLinePoint = **new** List<DataPoint>();
34. subLinePoint.Add(pl1.PointList[0]);
35. **for** (**int** i = 0; i < pl1.PointList.Count - 1; i++)
36. {
37. Edge pl1OneEdge = **new** Edge(pl1.PointList[i], pl1.PointList[i + 1]);
38. DataPoint intersectP = Edge.IntersectPoint(pl1OneEdge, edge);
39. **double** relativeLocOnLine = Edge.IntersectPointRelativeLoc(pl1OneEdge, edge);
40. **double** relativeLocOnEdge = Edge.IntersectPointRelativeLoc(edge, pl1OneEdge);
41. **if** (intersectP != **null**)
42. {
43. **if** (relativeLocOnEdge < 1 && relativeLocOnEdge > 0)
44. { intersectP.RelativeLoc = relativeLocOnEdge; subEdgePoint.Add(intersectP); }
45. **if** (relativeLocOnLine <= 1 && relativeLocOnLine > 0)
46. {
47. subLinePoint.Add(intersectP);
48. sublineFromPL1.Add(**new** ContourPolyline(subLinePoint.ToArray()));
49. subLinePoint = **new** List<DataPoint>();
50. subLinePoint.Add(intersectP);
51. }
52. }
53. subLinePoint.Add(pl1.PointList[i + 1]);
54. }
55. sublineFromPL1.Add(**new** ContourPolyline(subLinePoint.ToArray()));
56. subEdgePoint.Sort((x, y) => x.RelativeLoc.CompareTo(y.RelativeLoc));
57. **for** (**int** i = 0; i < subEdgePoint.Count - 1; i++)
58. { suEdgeFromEdge.Add(**new** Edge(subEdgePoint[i], subEdgePoint[i + 1])); }
59. **return** **new** Object[2] { sublineFromPL1, suEdgeFromEdge };
60. }
61. //获取多条等值线与线段你的交点
62. **public** **static** Object[] IntersectResult(ContourPolyline[] plineList, Edge edge)
63. {
64. List<ContourPolyline> sublineFromPLs = **new** List<ContourPolyline>();
65. List<Edge> suEdgeFromEdge = **new** List<Edge>();
66. //对边上点排序
67. List<DataPoint> subEdgePoint = **new** List<DataPoint>();
68. subEdgePoint.Add(edge.StartPoint);
69. subEdgePoint.Add(edge.EndPoint);
70. edge.StartPoint.RelativeLoc = 0;
71. edge.EndPoint.RelativeLoc = 1;
72. **for** (**int** k = 0; k < plineList.Length; k++)
73. {
74. //对折线上点排序
75. List<DataPoint> subLinePoint = **new** List<DataPoint>();
76. ContourPolyline curCpl = plineList[k];
77. subLinePoint.Add(curCpl.PointList[0]);
78. //选取一个等值线
79. **for** (**int** i = 0; i < curCpl.PointList.Count - 1; i++)
80. {
81. Edge pl1OneEdge = **new** Edge(curCpl.PointList[i], curCpl.PointList[i + 1]);
82. DataPoint intersectP = Edge.IntersectPoint(pl1OneEdge, edge);
83. **double** relativeLocOnLine = Edge.IntersectPointRelativeLoc(pl1OneEdge, edge);
84. **double** relativeLocOnEdge = Edge.IntersectPointRelativeLoc(edge, pl1OneEdge);
85. **if** (intersectP != **null**)
86. {
87. **if** (relativeLocOnEdge < 1 && relativeLocOnEdge > 0)
88. { intersectP.RelativeLoc = relativeLocOnEdge; subEdgePoint.Add(intersectP); }
89. **if** (relativeLocOnLine <= 1 && relativeLocOnLine > 0)
90. {
91. **if** (subLinePoint.Count == 1 && subLinePoint[0].OID == intersectP.OID) { }
92. **else**
93. {
94. subLinePoint.Add(intersectP);
95. sublineFromPLs.Add(**new** ContourPolyline(subLinePoint.ToArray()));
96. subLinePoint = **new** List<DataPoint>();
97. subLinePoint.Add(intersectP);
98. }
99. }
100. }
101. subLinePoint.Add(curCpl.PointList[i + 1]);
102. }
103. **if** (subLinePoint.Count >= 2 && !(subLinePoint.Count == 2 && subLinePoint[0].OID == subLinePoint[1].OID))
104. sublineFromPLs.Add(**new** ContourPolyline(subLinePoint.ToArray()));
105. }
106. subEdgePoint.Sort((x, y) => x.RelativeLoc.CompareTo(y.RelativeLoc));
107. **for** (**int** i = 0; i < subEdgePoint.Count - 1; i++)
108. {
109. **if** (subEdgePoint[i].RelativeLoc != subEdgePoint[i + 1].RelativeLoc)
110. suEdgeFromEdge.Add(**new** Edge(subEdgePoint[i], subEdgePoint[i + 1]));
111. }
112. **return** **new** Object[2] { sublineFromPLs, suEdgeFromEdge };
113. }
115. **public** **override** **string** ToString()
116. { **return** **string**.Format("CLid:{0},PtsCount:{1}", **this**.PID, **this**.PointList.Count); }
117. }
118. }

# ContourPolylineSet.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. /// <summary>
10. /// 等值线集合
11. /// </summary>
12. **public** **class** ContourPolylineSet
13. {
14. **public** List<ContourPolyline> ContourPolylineList = **new** List<ContourPolyline>();
15. **public** ContourPolylineSet() { }
16. **public** ContourPolylineSet(ContourPolyline[] polylines) { ContourPolylineList.AddRange(polylines); }
17. }  }

# DataPoint.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. /// <summary>
10. /// 数据点类
11. /// </summary>
12. **public** **class** DataPoint
13. {
14. **public** **int** ID { **get**; **private** **set**; }
15. **public** **string** Name { **get**; **private** **set**; }
16. **public** **double** X { **get**; **private** **set**; }
17. **public** **double** Y { **get**; **private** **set**; }
18. **public** **double** Value { **get**; **private** **set**; }
19. **public** MinBoundRect MBR { **get**; **private** **set**; }
20. **public** **int** OID { **get**; **private** **set**; }
21. **private** **static** **int** \_oid = 1000000;
22. **public** **double** RelativeLoc { **get**; **set**; }
24. **public** DataPoint(**int** id, **string** name, **double** x, **double** y, **double** value,**int** oid)
25. {
26. **this**.ID = id;
27. **this**.Name = name;
28. **this**.X = x;
29. **this**.Y = y;
30. **this**.Value = value;
31. **this**.MBR = **new** MinBoundRect(x, y, x, y);
32. **this**.OID = oid;
33. }
35. **public** DataPoint(**int** id, **string** name, **double** x, **double** y, **double** value)
36. {
37. **this**.ID = id;
38. **this**.Name = name;
39. **this**.X = x;
40. **this**.Y = y;
41. **this**.Value = value;
42. **this**.MBR = **new** MinBoundRect(x, y, x, y);
43. **this**.OID = \_oid++;
44. }
46. **public** **override** **string** ToString()
47. {**return** **string**.Format("ID:{0} Name:{1}\r\n Point({2},{3})\r\nValue:{4}",
48. ID, Name, X, Y, Value);}
50. //获取与另一点得距离
51. **public** **double** GetDistance(DataPoint other)
52. {**return** Math.Sqrt(Math.Pow(**this**.X - other.X, 2) + Math.Pow(**this**.Y - other.Y, 2));}
54. **public** **double** GetDistance(**double** x, **double** y)
55. {**return** Math.Sqrt(Math.Pow(**this**.X - x, 2) + Math.Pow(**this**.Y - y, 2));}
57. **public** **double** GetDistanceP2(**double** x, **double** y)
58. {**return** (Math.Pow(**this**.X - x, 2) + Math.Pow(**this**.Y - y, 2));}
60. //获取在另一点的方位角(角度)
61. **public** **double** GetPosition(**double** x,**double** y)
62. {
63. **double** deltaX = **this**.X - x;
64. **double** deltaY = **this**.Y - y;
65. **if** (deltaX \* deltaY == 0)
66. {
67. **if** (deltaX == 0)
68. {**if** (deltaY > 0)**return** 90;**else** **if** (deltaY < 0)**return** 270;
69. **else** **throw** **new** Exception("DataPoint.GetPosition:两点重合"); }
70. **else**{**if** (deltaX > 0)**return** 0;**else** **return** 180;}
71. }
72. **else**
73. {
74. **double** alpha = Math.Atan(Math.Abs(deltaY / deltaX));
75. **if** (deltaX > 0) {**if** (deltaY > 0) **return** alpha;**else** **return** 360 - alpha; }
76. **else**  {**if** (deltaY > 0) **return** 180 - alpha;**else** **return** 180 + alpha;}
77. }
79. }
81. **public** **static** Vector2D **operator** - (DataPoint p1 ,DataPoint p2)
82. {**return** **new** Vector2D(p1.X - p2.X, p1.Y - p2.Y);}
84. //获取三点构成的角度
85. **public** **static** **double** Angle(DataPoint c, DataPoint a, DataPoint b)
86. {
87. **double** ang;
88. **double** l1 = Math.Sqrt((b.X - c.X) \* (b.X - c.X) + (b.Y - c.Y) \* (b.Y - c.Y));
89. **double** l2 = Math.Sqrt((a.X - c.X) \* (a.X - c.X) + (a.Y - c.Y) \* (a.Y - c.Y));
90. **double** l3 = Math.Sqrt((b.X - a.X) \* (b.X - a.X) + (b.Y - a.Y) \* (b.Y - a.Y));
91. ang = Math.Acos((l1 \* l1 + l2 \* l2 - l3 \* l3) / (2 \* l1 \* l2));
92. **return** ang;
93. }
94. **public** **static** **int** LeftOrRight(DataPoint c, DataPoint a, DataPoint b)
95. {
96. **int** youbian;
97. **double** S= (a.X - c.X) \* (b.Y - c.Y) - (a.Y - c.Y) \* (b.X - c.X);
98. **if** (S > 0){youbian = 1;}
99. **else** **if** (S < 0) {youbian = -1;}
100. **else**{youbian = 0;}
101. **return** youbian;
102. }
103. }
104. }

# Edge.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. /// <summary>
10. /// 线段类
11. /// </summary>
12. **public** **class** Edge
13. {
14. **public** **int** EID { **get**; **private** **set**; }
15. **public** DataPoint StartPoint { **get**; **private** **set**; }
16. **public** DataPoint EndPoint { **get**; **private** **set**; }
17. **public** **int** StartOID
18. { **get** { **return** StartPoint.OID; } }
19. **public** **int** EndOID
20. { **get** { **return** EndPoint.OID; } }
21. **public** Triangle OwnerTriangle { **get**; **set**; }
22. **public** **object** Tag { **get**; **set**; }
23. **private** **static** **int** \_eid = -777777;
25. **public** Edge(DataPoint startP, DataPoint endP)
26. {
27. **this**.StartPoint = startP;
28. **this**.EndPoint = endP;
29. **this**.EID = \_eid--;
30. }
31. **public** **double** MaxValue()
32. { **return** Math.Max(StartPoint.Value, EndPoint.Value); }
34. **public** **double** MinValue()
35. { **return** Math.Min(StartPoint.Value, EndPoint.Value); }
37. **public** **double** GetRelativeCoordinate(**double** value)
38. { **return** (value - StartPoint.Value) / (EndPoint.Value - StartPoint.Value); }
40. **public** **double** GetValue(**double** ralativeCoordinate)
41. { **return** ralativeCoordinate \* (EndPoint.Value - StartPoint.Value) + StartPoint.Value; }
43. **public** **bool** IsEqulesEdge(**int** oid1, **int** oid2)
44. {
45. **return** ((StartPoint.OID == oid1) && (EndPoint.OID == oid2) ||
46. (StartPoint.OID == oid2) && (EndPoint.OID == oid1));
47. }
49. //获取两边交点
50. **public** **static** DataPoint IntersectPoint(Edge e1, Edge e2)
51. {
52. **double** IntersectX =
53. ((e1.EndPoint.X - e1.StartPoint.X) \* (e2.StartPoint.X - e2.EndPoint.X) \* (e2.StartPoint.Y - e1.StartPoint.Y) -
54. e2.StartPoint.X \* (e1.EndPoint.X - e1.StartPoint.X) \* (e2.StartPoint.Y - e2.EndPoint.Y) +
55. e1.StartPoint.X \* (e1.EndPoint.Y - e1.StartPoint.Y) \* (e2.StartPoint.X - e2.EndPoint.X)) /
56. ((e1.EndPoint.Y - e1.StartPoint.Y) \* (e2.StartPoint.X - e2.EndPoint.X) -
57. (e1.EndPoint.X - e1.StartPoint.X) \* (e2.StartPoint.Y - e2.EndPoint.Y));
58. **double** IntersectY =
59. ((e1.EndPoint.Y - e1.StartPoint.Y) \* (e2.StartPoint.Y - e2.EndPoint.Y) \* (e2.StartPoint.X - e1.StartPoint.X) -
60. e2.StartPoint.Y \* (e1.EndPoint.Y - e1.StartPoint.Y) \* (e2.StartPoint.X - e2.EndPoint.X) +
61. e1.StartPoint.Y \* (e1.EndPoint.X - e1.StartPoint.X) \* (e2.StartPoint.Y - e2.EndPoint.Y)) /
62. ((e1.EndPoint.X - e1.StartPoint.X) \* (e2.StartPoint.Y - e2.EndPoint.Y) -
63. (e1.EndPoint.Y - e1.StartPoint.Y) \* (e2.StartPoint.X - e2.EndPoint.X));
64. **double** relativeE1 = 0;
65. **if** ((e1.EndPoint.X - e1.StartPoint.X) != 0)
66. relativeE1 = (IntersectX - e1.StartPoint.X) / (e1.EndPoint.X - e1.StartPoint.X);
67. **else** relativeE1 = (IntersectY - e1.StartPoint.Y) / (e1.EndPoint.Y - e1.StartPoint.Y);
68. **double** relativeE2 = 0;
69. **if** ((e2.EndPoint.X - e2.StartPoint.X) != 0)
70. relativeE2 = (IntersectX - e2.StartPoint.X) / (e2.EndPoint.X - e2.StartPoint.X);
71. **else** relativeE2 = (IntersectY - e2.StartPoint.Y) / (e2.EndPoint.Y - e2.StartPoint.Y);
72. **int** tempID = Math.Abs(e1.StartOID) + Math.Abs(e1.EndOID) + Math.Abs(e2.StartOID) + Math.Abs(e2.EndOID);
73. **if** (Math.Abs(relativeE1) < 1E-5)
74. **return** e1.StartPoint;
75. **else** **if** (Math.Abs(relativeE1 - 1) < 1E-5)
76. **return** e1.EndPoint;
77. **else** **if** (Math.Abs(relativeE2) < 1E-5)
78. **return** e2.StartPoint;
79. **else** **if** (Math.Abs(relativeE2 - 1) < 1E-5)
80. **return** e2.EndPoint;
81. **else** **if** (relativeE1 < 1 && relativeE1 > 0 && relativeE2 > 0 && relativeE2 < 1)
82. **return** **new** DataPoint(tempID, tempID.ToString(), IntersectX, IntersectY,
83. e1.StartPoint.Value + relativeE1 \* (e1.EndPoint.Value - e1.StartPoint.Value));
84. **else** **return** **null**;
85. }
87. **public** **static** **double** IntersectPointRelativeLoc(Edge e1, Edge e2)
88. {
89. **double** IntersectX =
90. ((e1.EndPoint.X - e1.StartPoint.X) \* (e2.StartPoint.X - e2.EndPoint.X) \* (e2.StartPoint.Y - e1.StartPoint.Y) -
91. e2.StartPoint.X \* (e1.EndPoint.X - e1.StartPoint.X) \* (e2.StartPoint.Y - e2.EndPoint.Y) +
92. e1.StartPoint.X \* (e1.EndPoint.Y - e1.StartPoint.Y) \* (e2.StartPoint.X - e2.EndPoint.X)) /
93. ((e1.EndPoint.Y - e1.StartPoint.Y) \* (e2.StartPoint.X - e2.EndPoint.X) -
94. (e1.EndPoint.X - e1.StartPoint.X) \* (e2.StartPoint.Y - e2.EndPoint.Y));
95. **double** IntersectY =
96. ((e1.EndPoint.Y - e1.StartPoint.Y) \* (e2.StartPoint.Y - e2.EndPoint.Y) \* (e2.StartPoint.X - e1.StartPoint.X) -
97. e2.StartPoint.Y \* (e1.EndPoint.Y - e1.StartPoint.Y) \* (e2.StartPoint.X - e2.EndPoint.X) +
98. e1.StartPoint.Y \* (e1.EndPoint.X - e1.StartPoint.X) \* (e2.StartPoint.Y - e2.EndPoint.Y)) /
99. ((e1.EndPoint.X - e1.StartPoint.X) \* (e2.StartPoint.Y - e2.EndPoint.Y) -
100. (e1.EndPoint.Y - e1.StartPoint.Y) \* (e2.StartPoint.X - e2.EndPoint.X));
101. **double** relativeE1 = 0;
102. **if** ((e1.EndPoint.X - e1.StartPoint.X) != 0)
103. relativeE1 = (IntersectX - e1.StartPoint.X) / (e1.EndPoint.X - e1.StartPoint.X);
104. **else** relativeE1 = (IntersectY - e1.StartPoint.Y) / (e1.EndPoint.Y - e1.StartPoint.Y);
105. **if** (Math.Abs(relativeE1) < 1E-5)
106. **return** 0;
107. **else** **if** (Math.Abs(relativeE1 - 1) < 1E-5)
108. **return** 1;
109. **else** **return** relativeE1;
110. }
112. /// <summary>
113. /// 判断边2两点是否在边1两侧
114. /// </summary>
115. /// <param name="e1"></param>
116. /// <param name="e2"></param>
117. /// <returns></returns>
118. **public** **static** **bool** CheckCross(Edge e1, Edge e2)
119. {
120. Vector2D v1 = e1.StartPoint - e1.EndPoint;
121. Vector2D v2 = e2.StartPoint - e1.EndPoint;
122. Vector2D v3 = e2.EndPoint - e2.EndPoint;
123. **return** v1.CrossProduct(v2) \* v1.CrossProduct(v3) < 0;
124. }
126. **public** **override** **string** ToString()
127. { **return** **string**.Format("EdgeID:{0},StaID:{1},EndID:{2}", **this**.EID, **this**.StartOID, **this**.EndOID); }
128. }
129. }

# EdgeSet.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. //线段集合
10. **public** **class** EdgeSet
11. {
12. **public** List<Edge> EdgeList = **new** List<Edge>();
13. **public** EdgeSet() { }
14. **public** EdgeSet(Edge[] edges) { EdgeList.AddRange(edges); }
16. /// <summary>
17. /// 移除指定EID的边
18. /// </summary>
19. /// <param name="eid"></param>
20. **public** **void** RemoveEdgeByEID(**int** eid)
21. {
22. **int** index = 0;
23. **foreach** (var edge **in** EdgeList)
24. { **if** (edge.EID == eid) **break**; index++; }
25. EdgeList.RemoveAt(index);
26. }
28. **public** **void** AddEdge(Edge e)
29. { EdgeList.Add(e); }
31. **public** Edge GetEdgeByOID(**int** oid1, **int** oid2)
32. {
33. **foreach** (var edge **in** EdgeList)
34. {
35. **if** (edge.IsEqulesEdge(oid1, oid2))
36. **return** edge;
37. }
38. **return** **null**;
39. }
41. **public** Edge GetEdgeByOID(**int** oid)
42. {
43. **foreach** (var edge **in** EdgeList)
44. {
45. **if** (edge.StartOID == oid || edge.EndOID == oid)
46. **return** edge;
47. }
48. **return** **null**;
49. }
51. //由线段和点集生成三角形集合
52. **public** **static** TriangleSet TopologyGenerateTriangleSet(Edge[] Edges, PointSet PointSet)
53. {
54. TriangleSet triangleSet = **new** TriangleSet();
55. List<**int**> pointOID = **new** List<**int**>();
56. EdgeSet edgeSet = **new** EdgeSet(Edges);
57. **for** (**int** i = 0; i < Edges.Length; i++)
58. {
59. **if** (pointOID.Contains(Edges[i].StartOID) != **true**)
60. pointOID.Add(Edges[i].StartOID);
61. **if** (pointOID.Contains(Edges[i].EndOID) != **true**)
62. pointOID.Add(Edges[i].EndOID);
63. }
64. **for** (**int** i = 0; i < Edges.Length; i++)
65. {
66. **int** soid = Edges[i].StartOID;
67. **int** eoid = Edges[i].EndOID;
68. **for** (**int** j = 0; j < pointOID.Count; j++)
69. {
70. Edge edge1 = edgeSet.GetEdgeByOID(soid, pointOID[j]);
71. Edge edge2 = edgeSet.GetEdgeByOID(eoid, pointOID[j]);
72. **if** (edge1 != **null** && edge2 != **null**)
73. {
74. Triangle tri = **new** Triangle(PointSet.GetPointByOID(soid),
75. PointSet.GetPointByOID(eoid),
76. PointSet.GetPointByOID(pointOID[j]), i);
77. **if** (triangleSet.IsTriAlreadyExists(soid, eoid, pointOID[j]) == **false**)
78. triangleSet.AddTriangle(tri);
79. }
80. }
82. }
83. **return** triangleSet;
84. }
86. //由线段集合生成等值线集合
87. **public** **static** ContourPolylineSet TopologyGenerateContourPolylineSet(Edge[] Edges)
88. {
89. List<ContourPolyline> ContourPolylineList = **new** List<ContourPolyline>();
90. PointSet ContourPointSet = **new** PointSet();
91. EdgeSet ContourEdgeSet = **new** EdgeSet(Edges);
92. EdgeSet ContourEdgeSetCopy = **new** EdgeSet(Edges);
93. **for** (**int** i = 0; i < Edges.Length; i++)
94. {
95. ContourPointSet.AddPoint(Edges[i].StartPoint);
96. ContourPointSet.AddPoint(Edges[i].EndPoint);
97. }
98. List<**int**> PointOID = ContourPointSet.GetPointOIDList();
99. **while** (PointOID.Count > 0)
100. {
101. //选取一个等值线上的点
102. List<**int**> tempPointsOID = **new** List<**int**>();
103. tempPointsOID.Add(PointOID[0]);
104. PointOID.Remove(PointOID[0]);
105. **for** (**int** i = 0; i < tempPointsOID.Count; i++)
106. {
107. Edge tempEdge = ContourEdgeSetCopy.GetEdgeByOID(tempPointsOID[i]);
108. **if** (tempEdge != **null**)
109. {
110. **int** tempoid = (tempEdge.StartOID == tempPointsOID[i]) ? tempEdge.EndOID : tempEdge.StartOID;
111. tempPointsOID.Add(tempoid);
112. ContourEdgeSetCopy.EdgeList.Remove(tempEdge);
113. PointOID.Remove(tempoid);
114. i = -1;
115. }
116. }
117. //找到了这条线上的全部点
118. List<DataPoint> tempPointList = **new** List<DataPoint>();
119. tempPointList.Add(ContourPointSet.GetPointByOID(tempPointsOID[0]));
120. Edge firstEdge = ContourEdgeSet.GetEdgeByOID(tempPointsOID[0]);
121. **int** secondOID = (firstEdge.StartOID == tempPointsOID[0]) ? firstEdge.EndOID : firstEdge.StartOID;
122. tempPointList.Add(ContourPointSet.GetPointByOID(secondOID));
123. ContourEdgeSet.EdgeList.Remove(firstEdge);
124. **while** (**true**)
125. {
126. **int** endPointOID = tempPointList.Last().OID;
127. Edge endEdge = ContourEdgeSet.GetEdgeByOID(endPointOID);
128. **if** (endEdge != **null**)
129. {
130. **int** endOID = (endEdge.StartOID == endPointOID) ? endEdge.EndOID : endEdge.StartOID;
131. tempPointList.Add(ContourPointSet.GetPointByOID(endOID));
132. ContourEdgeSet.EdgeList.Remove(endEdge);
133. }
134. **else** **break**;
135. }
136. **while** (**true**)
137. {
138. **int** startPointOID = tempPointList.First().OID;
139. Edge startEdge = ContourEdgeSet.GetEdgeByOID(startPointOID);
140. **if** (startEdge != **null**)
141. {
142. **int** startOID = (startEdge.StartOID == startPointOID) ? startEdge.EndOID : startEdge.StartOID;
143. tempPointList.Insert(0, ContourPointSet.GetPointByOID(startOID));
144. ContourEdgeSet.EdgeList.Remove(startEdge);
145. }
146. **else** **break**;
147. }
148. ContourPolyline tempPolyline = **new** ContourPolyline(tempPointList.ToArray());
149. ContourPolylineList.Add(tempPolyline);
150. }
151. **return** **new** ContourPolylineSet(ContourPolylineList.ToArray());
152. }
153. }
154. }

# MBR.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. /// <summary>
10. /// 最小外包矩形类
11. /// </summary>
12. **public** **class** MinBoundRect
13. {
14. **public** **double** MinX { **get**; **private** **set**; }
15. **public** **double** MinY { **get**; **private** **set**; }
16. **public** **double** MaxX { **get**; **private** **set**; }
17. **public** **double** MaxY { **get**; **private** **set**; }
19. **public** MinBoundRect()
20. {
21. **this**.MinX = **double**.MaxValue;
22. **this**.MinY = **double**.MaxValue;
23. **this**.MaxX = **double**.MinValue;
24. **this**.MaxY = **double**.MinValue;
25. }
26. **public** MinBoundRect(**double** minX,**double** minY,**double** maxX,**double** maxY)
27. {
28. **this**.MinX = minX;
29. **this**.MinY = minY;
30. **this**.MaxX = maxX;
31. **this**.MaxY = maxY;
32. }
34. **public** **void** UpdateRect(**double** x,**double** y)
35. {
36. **this**.MinX = Math.Min(**this**.MinX, x);
37. **this**.MinY = Math.Min(**this**.MinY, y);
38. **this**.MaxX = Math.Max(**this**.MaxX, x);
39. **this**.MaxY = Math.Max(**this**.MaxY, y);
40. **return**;
41. }
43. **public** **void** UpdateRect(MinBoundRect mbr)
44. {
45. **this**.MinX = Math.Min(**this**.MinX, mbr.MinX);
46. **this**.MinY = Math.Min(**this**.MinY, mbr.MinY);
47. **this**.MaxX = Math.Max(**this**.MaxX, mbr.MaxX);
48. **this**.MaxY = Math.Max(**this**.MaxY, mbr.MaxY);
49. **return**;
50. }
52. **public** **void** PanningVector(**double** deltaX,**double** deltaY)
53. {
54. **this**.MinX += deltaX;
55. **this**.MinY += deltaY;
56. **this**.MaxX += deltaX;
57. **this**.MaxY += deltaY;
58. }
60. **public** **void** ZoomPointAndRatio(**double** x,**double** y,**double** ratio)
61. {
62. **this**.MinX = x - (x - **this**.MinX) \* ratio;
63. **this**.MinY = y - (y - **this**.MinY) \* ratio;
64. **this**.MaxX = (**this**.MaxX - x) \* ratio + x;
65. **this**.MaxY = (**this**.MaxY - y) \* ratio + y;
66. }
67. }
68. }

# PointSet.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
6. **using** System.IO;
8. **namespace** AGIS\_work.DataStructure
9. {
10. /// <summary>
11. /// 数据点集合
12. /// </summary>
13. **public** **class** PointSet
14. {
15. **public** **string** SetName { **get**; **private** **set**; }
16. **public** **string** FileName { **get**; **private** **set**; }
18. **public** List<DataPoint> PointList { **get**; **private** **set**; }
19. **public** MinBoundRect MBR { **get**; **private** **set**; }
21. **public** PointSet() { MBR = **new** MinBoundRect(-1, -1, 1, 1); PointList = **new** List<DataPoint>(); }
22. **public** PointSet(**string** setname, **string** filename, DataPoint[] points)
23. {
24. **this**.SetName = setname;
25. **this**.FileName = filename;
26. **this**.PointList = **new** List<DataPoint>(points);
27. //最小外接矩形
28. MBR = **new** MinBoundRect();
29. **foreach** (DataPoint point **in** points)
30. MBR.UpdateRect(point.X, point.Y);
31. }
33. /// <summary>
34. /// 从CSV文件中读取点集
35. /// </summary>
36. /// <param name="filename"></param>
37. /// <returns></returns>
38. **public** **static** PointSet ReadFromCSV(**string** filename)
39. {
40. PointSet pointSet = **null**;
41. StreamReader sr = **new** StreamReader(filename);
42. List<DataPoint> dataPoints = **new** List<DataPoint>();
43. **try**
44. {
45. **string** setName = sr.ReadLine();
46. **int** oid = 0;
47. **while** (!sr.EndOfStream)
48. {
49. **string** onePoint = sr.ReadLine();
50. **string**[] pointInfo = onePoint.Split(',');
51. dataPoints.Add(**new** DataPoint(**int**.Parse(pointInfo[0]), pointInfo[1], **double**.Parse(pointInfo[2]),
52. **double**.Parse(pointInfo[3]), **double**.Parse(pointInfo[4])));
53. oid++;
54. }
55. pointSet = **new** PointSet(setName, filename, dataPoints.ToArray());
56. }
57. **catch** (Exception err) { **throw** err; }
58. sr.Close();
59. **return** pointSet;
60. }
62. /// <summary>
63. /// 将点集写入CSV文件
64. /// </summary>
65. /// <param name="filename"></param>
66. **public** **void** WriteToCSV(**string** filename = **null**)
67. {
68. **string** filePath = filename == **null** ? **this**.FileName : filename;
69. StreamWriter sw = **new** StreamWriter(filePath);
70. **try**
71. {
72. sw.WriteLine(**this**.SetName);
73. **foreach** (DataPoint point **in** **this**.PointList)
74. {
75. sw.WriteLine(**string**.Format("{0},{1},{2],{3},{4}", point.ID, point.Name, point.X, point.Y, point.Value));
76. }
78. }
79. **catch** (Exception err)
80. {
81. **throw** err;
82. }
83. sw.Close();
84. **return**;
85. }
87. /// <summary>
88. /// 根据OID返回数据点
89. /// </summary>
90. /// <param name="oid"></param>
91. /// <returns></returns>
92. **public** DataPoint GetPointByOID(**int** oid)
93. {
94. **foreach** (var point **in** PointList)
95. { **if** (point.OID == oid) **return** point; }
96. **return** **null**;
97. }
99. /// <summary>
100. /// 添加数据点（OID不重复）
101. /// </summary>
102. /// <param name="point"></param>
103. /// <returns>是否添加成功</returns>
104. **public** **bool** AddPoint(DataPoint point)
105. {
106. **if** (GetPointByOID(point.OID) == **null**)
107. { PointList.Add(point); **return** **true**; }
108. **else** **return** **false**;
109. }
111. /// <summary>
112. /// 返回全部数据点的OID
113. /// </summary>
114. /// <returns></returns>
115. **public** List<**int**> GetPointOIDList()
116. {
117. List<**int**> OIDList = **new** List<**int**>();
118. **foreach** (var point **in** PointList)
119. OIDList.Add(point.OID);
120. **return** OIDList;
121. }
122. }
123. }

# TopoPoint.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. //拓扑点
10. **public** **class** TopoPoint
11. {
12. **private** **static** **int** \_pointID = 0;
13. **public** **int** Innerid = 0; //内部码
14. **public** **int** PointID { **get**; **private** **set**; }    //唯一标识
15. **public** **bool** IsNode { **get**; **private** **set**; }    //是否是结点
16. **public** **double** X { **get**; **private** **set**; }
17. **public** **double** Y { **get**; **private** **set**; }
18. **public** **double** Z { **get**; **private** **set**; }
19. **public** List<TopoPolyline> TopologyArcs { **get**; **private** **set**; }//相关弧段
21. **public** TopoPoint(DataPoint dpoint, **bool** isNode)
22. {
23. **this**.PointID = dpoint.OID;
24. **this**.IsNode = isNode;
25. **this**.X = dpoint.X;
26. **this**.Y = dpoint.Y;
27. **this**.Z = dpoint.Value;
28. **this**.TopologyArcs = **new** List<TopoPolyline>();
29. Innerid = \_pointID++;
30. }
32. **public** TopoPoint(**double** x, **double** y, **double** z, **bool** isNode)
33. {
34. **this**.PointID = \_pointID++;
35. **this**.IsNode = isNode;
36. **this**.X = x;
37. **this**.Y = y;
38. **this**.Z = z;
39. **this**.TopologyArcs = **new** List<TopoPolyline>();
40. Innerid = \_pointID++;
41. }
42. /// <summary>
43. /// 获取另一点对于当前点的方位角(角度)
44. /// </summary>
45. /// <param name="x"></param>
46. /// <param name="y"></param>
47. /// <returns></returns>
48. **public** **double** GetPosition(**double** x, **double** y)
49. {
50. **double** deltaX = x - **this**.X;
51. **double** deltaY = y - **this**.Y;
52. **if** (deltaX \* deltaY == 0)
53. {
54. **if** (deltaX == 0)
55. {
56. **if** (deltaY > 0)
57. **return** 90;
58. **else** **if** (deltaY < 0)
59. **return** 270;
60. **else**
61. **throw** **new** Exception("Topology.GetPosition:两点重合");
62. }
63. **else**
64. {
65. **if** (deltaX > 0)
66. **return** 0;
67. **else** **return** 180;
68. }
69. }
70. **else**
71. {
72. **double** alpha = Math.Atan(Math.Abs(deltaY / deltaX));
73. **if** (deltaX > 0)
74. {
75. **if** (deltaY > 0) **return** alpha;
76. **else** **return** 360 - alpha;
77. }
78. **else**
79. {
80. **if** (deltaY > 0) **return** 180 - alpha;
81. **else** **return** 180 + alpha;
82. }
83. }
85. }
87. /// <summary>
88. /// 获取另一点对于当前点的方位角(角度)
89. /// </summary>
90. /// <param name="p"></param>
91. /// <returns></returns>
92. **public** **double** GetPositon(TopoPoint p)
93. {  **return** **this**.GetPosition(p.X, p.Y);   }
95. //获取与另一点距离
96. **public** **double** GetDistance(TopoPoint other)
97. { **return** Math.Sqrt(Math.Pow(**this**.X - other.X, 2) + Math.Pow(**this**.Y - other.Y, 2)); }
99. **public** **override** **string** ToString()
100. { **return** **string**.Format("PointID:{0},IsNode:{1}", **this**.PointID, **this**.IsNode.ToString()); }
101. }
102. }

# TopoPointSet.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.IO;
4. **using** System.Linq;
5. **using** System.Text;
6. **using** System.Threading.Tasks;
8. **namespace** AGIS\_work.DataStructure
9. {
10. //拓扑点点集
11. **public** **class** TopoPointSet
12. {
13. /// <summary>
14. /// 中间点表
15. /// </summary>
16. **public** List<TopoPoint> TopoPointList { **get**; **private** **set**; }
17. /// <summary>
18. /// 结点表
19. /// </summary>
20. **public** List<TopoPoint> TopoNodeList { **get**; **private** **set**; }
22. **public** TopoPointSet()
23. {
24. TopoPointList = **new** List<TopoPoint>();
25. TopoNodeList = **new** List<TopoPoint>();
26. }
28. **public** TopoPointSet(TopoPoint[] topoPoints)
29. {
30. TopoPointList = **new** List<TopoPoint>();
31. TopoNodeList = **new** List<TopoPoint>();
32. **for** (**int** i = 0; i < topoPoints.Length; i++)
33. {
34. **if** (topoPoints[i].IsNode == **false**)
35. TopoPointList.Add(topoPoints[i]);
36. **else**
37. TopoNodeList.Add(topoPoints[i]);
38. }
39. }
41. /// <summary>
42. /// 判断中间点是否已经存在
43. /// </summary>
44. /// <param name="oid"></param>
45. /// <returns></returns>
46. **public** **bool** IfPointExists(**int** oid)
47. {
48. **foreach** (var point **in** TopoPointList)
49. {
50. **if** (point.PointID == oid)
51. **return** **true**;
52. }
53. **return** **false**;
54. }
56. /// <summary>
57. /// 判断节点是否已经存在
58. /// </summary>
59. /// <param name="oid"></param>
60. /// <returns></returns>
61. **public** **bool** IfNodeExists(**int** oid)
62. {
63. **foreach** (var point **in** TopoNodeList)
64. {
65. **if** (point.PointID == oid)
66. **return** **true**;
67. }
68. **return** **false**;
69. }
71. **public** TopoPoint GetNodeByPointID(**int** poid)
72. {
73. **foreach** (var point **in** TopoNodeList)
74. { **if** (point.PointID == poid) **return** point; }
75. **return** **null**;
76. }
78. **public** TopoPointSet(TopoPolyline[] topoLines)
79. {
80. TopoPointList = **new** List<TopoPoint>();
81. TopoNodeList = **new** List<TopoPoint>();
82. **for** (**int** i = 0; i < topoLines.Length; i++)
83. {
84. **for** (**int** j = 0; j < topoLines[i].MiddlePoint.Count; j++)
85. {
86. **if** (**this**.IfPointExists(topoLines[i].MiddlePoint[j].PointID) == **false**)
87. TopoPointList.Add(topoLines[i].MiddlePoint[j]);
88. }
89. **if** (**this**.IfNodeExists(topoLines[i].BeginNode.PointID) == **false**)
90. TopoNodeList.Add(topoLines[i].BeginNode);
91. **else**
92. {
93. TopoPoint existPoint = **this**.GetNodeByPointID(topoLines[i].BeginNode.PointID);
94. **if** (existPoint.TopologyArcs.Contains(topoLines[i]) == **false**)
95. existPoint.TopologyArcs.Add(topoLines[i]);
96. }
97. **if** (**this**.IfNodeExists(topoLines[i].EndNode.PointID) == **false**)
98. TopoNodeList.Add(topoLines[i].EndNode);
99. **else**
100. {
101. TopoPoint existPoint = **this**.GetNodeByPointID(topoLines[i].EndNode.PointID);
102. **if** (existPoint.TopologyArcs.Contains(topoLines[i]) == **false**)
103. existPoint.TopologyArcs.Add(topoLines[i]);
104. }
105. }
106. }
108. /// <summary>
109. /// 根据点的拓扑关系，生成多边形的拓扑关系
110. /// </summary>
111. /// <returns></returns>
112. **public** TopoPolygonSet GenerateTopoPolygonSet()
113. {
114. List<TopoPolygon> polygonList = **new** List<TopoPolygon>();
115. **foreach** (var tpPoint **in** **this**.TopoNodeList)
116. {
117. List<Tuple<TopoPoint, **double**, **double**, **int**, TopoPolyline>> curPointStructList =
118. **new** List<Tuple<TopoPoint, **double**, **double**, **int**, TopoPolyline>>();
119. List<TopoPolyline> relaArcs = tpPoint.TopologyArcs;
120. **foreach** (var arc **in** relaArcs)
121. {
122. **int** direct = arc.IsNode(tpPoint);
123. **double** angle = -1;
124. **double** otherNodeAngle = -2;
125. **switch** (direct)
126. {
127. **case** 1:
128. angle = arc.GetBeginNodeAngle();
129. otherNodeAngle = arc.GetEndNodeAngle();
130. **break**;
131. **case** -1:
132. angle = arc.GetEndNodeAngle();
133. otherNodeAngle = arc.GetBeginNodeAngle();
134. **break**;
135. **default**:
136. **break**;
137. }
138. Tuple<TopoPoint, **double**, **double**, **int**, TopoPolyline> curPointStruct
139. = **new** Tuple<TopoPoint, **double**, **double**, **int**, TopoPolyline>(arc.GetAnotherNode(tpPoint), angle, otherNodeAngle, direct, arc);
140. curPointStructList.Add(curPointStruct);
141. }
142. List<TopoPolyline> curPolygon = **new** List<TopoPolyline>();
143. List<**int**> directList = **new** List<**int**>();
144. **foreach** (var pointStruct **in** curPointStructList)
145. {
146. var iterationTuple = pointStruct;
147. **while** (iterationTuple.Item1.PointID != tpPoint.PointID)
148. {
149. curPolygon.Add(iterationTuple.Item5);
150. directList.Add(iterationTuple.Item4);
151. iterationTuple = **this**.GetNextNode(iterationTuple);
152. }
153. curPolygon.Add(iterationTuple.Item5);
154. directList.Add(iterationTuple.Item4);
155. //追踪成功
156. **if** (curPolygon.Count > 0)
157. {
158. TopoPolygon tempPolygon = **new** TopoPolygon(curPolygon.ToArray());
159. **for** (**int** i = 0; i < directList.Count; i++)
160. {
161. **if** (directList[i] > 0)
162. curPolygon[i].RightPolygon = tempPolygon;
163. **else** **if** (directList[i] < 0)
164. curPolygon[i].LeftPolygon = tempPolygon;
165. }
166. polygonList.Add(tempPolygon);
167. curPolygon = **new** List<TopoPolyline>();
168. directList = **new** List<**int**>();
169. }
170. }
172. }
173. **return** **new** TopoPolygonSet(polygonList.ToArray());
174. }
175. //对搜索边进行排序
176. **public** **void** SortTheSearchLine(List<Tuple<TopoPoint, **double**, **double**, **int**, TopoPolyline>> lineToSort, **double** startAngle)
177. {
178. lineToSort.Sort((x, y) =>
179. {
180. **double** x2 = x.Item2 - startAngle;
181. **double** y2 = y.Item2 - startAngle;
182. **if** (x2 <= 0) x2 += 360;
183. **if** (y2 <= 0) y2 += 360;
184. **return** x2.CompareTo(y2);
185. });
186. **return**;
187. }
188. //根据点标识获取点
189. **public** TopoPoint GetTopoPointByID(**int** poid)
190. {
191. **foreach** (var point **in** **this**.TopoPointList)
192. {
193. **if** (point.PointID == poid)
194. **return** point;
195. }
196. **return** **null**;
197. }
199. /// <summary>
200. /// 循环搜索下一个节点
201. /// </summary>
202. /// <param name="curNode">TopoPoint当前搜索点，double前一步的排序条件，double当前的步的排序条件，int方向，Polyline当前弧段</param>
203. /// <returns></returns>
204. **public** Tuple<TopoPoint, **double**, **double**, **int**, TopoPolyline> GetNextNode(Tuple<TopoPoint, **double**, **double**, **int**, TopoPolyline> curNode)
205. {
206. List<TopoPolyline> curArcs = **this**.GetNodeByPointID(curNode.Item1.PointID).TopologyArcs;
207. List<Tuple<TopoPoint, **double**, **double**, **int**, TopoPolyline>> tempList = **new** List<Tuple<TopoPoint, **double**, **double**, **int**, TopoPolyline>>();
208. **foreach** (var arc **in** curArcs)
209. {
210. **int** direct = arc.IsNode(curNode.Item1);
211. **double** angle = -1;
212. **double** otherNodeAngle = -2;
213. **switch** (direct)
214. {
215. **case** 1:
216. angle = arc.GetBeginNodeAngle();
217. otherNodeAngle = arc.GetEndNodeAngle();
218. **break**;
219. **case** -1:
220. angle = arc.GetEndNodeAngle();
221. otherNodeAngle = arc.GetBeginNodeAngle();
222. **break**;
223. **default**:
224. **break**;
225. }
226. tempList.Add(**new** Tuple<TopoPoint, **double**, **double**, **int**, TopoPolyline>(
227. arc.GetAnotherNode(curNode.Item1), angle, otherNodeAngle, direct, arc));
228. }
229. SortTheSearchLine(tempList, curNode.Item3);
230. **return** tempList[0];
231. }
232. //导出结点关系表至文件
233. **public** **void** SaveNodeTableToFile(**string** filename)
234. {
235. StreamWriter sw = **new** StreamWriter(filename);
236. sw.WriteLine("ID\tNode\_ID\tX\tY\tZ\tArc\_Num\tArc\_IDs");
237. **foreach** (var node **in** TopoNodeList)
238. {
239. **string** arcIDs = "";
240. **foreach** (var arc **in** node.TopologyArcs)
241. arcIDs += arc.ArcID + ",";
242. sw.WriteLine(
243. **string**.Format("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}",
244. node.Innerid, node.PointID, node.X, node.Y, node.Z, node.TopologyArcs.Count, arcIDs.Remove(arcIDs.Length - 1)));
245. }
246. sw.Close();
247. }
248. //导出中间点关系表至文件
249. **public** **void** SavePointTableToFile(**string** filename)
250. {
251. StreamWriter sw = **new** StreamWriter(filename);
252. sw.WriteLine("ID\tNode\_ID\tX\tY\tZ\tArc\_Num\tArc\_IDs");
253. **foreach** (var point **in** TopoPointList)
254. {
255. **string** arcIDs = "";
256. **foreach** (var arc **in** point.TopologyArcs)
257. arcIDs += arc.ArcID + ",";
258. sw.WriteLine(
259. **string**.Format("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}",
260. point.Innerid, point.PointID, point.X, point.Y, point.Z, point.TopologyArcs.Count, arcIDs.Remove(arcIDs.Length - 1)));
261. }
262. sw.Close();
263. }
264. }
265. }

# TopoPolygon.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. //拓扑多边形
10. **public** **class** TopoPolygon
11. {
12. **private** **static** **int** \_polygonID = 0;
13. **public** **int** innerId = 0; //内部码
14. **public** **int** PID { **get**; **private** **set**; }//唯一标识码
15. **public** List<TopoPolyline> TopologyArcs { **get**; **set**; }//相关弧段
16. **public** TopoPolygon OuterPolygon { **get**; **set**; }//外多边形
17. **public** List<TopoPolygon> InnerPolygons { **get**; **set**; }//内多边形
18. **public** MinBoundRect MBR { **get**; **private** **set**; }
20. **public** TopoPolygon()
21. {
22. **this**.PID = \_polygonID++;
23. OuterPolygon = **null**;
24. TopologyArcs = **new** List<TopoPolyline>();
25. InnerPolygons = **new** List<TopoPolygon>();
26. MBR = **new** MinBoundRect();
27. innerId = **this**.PID;
28. }
30. **public** TopoPolygon(TopoPolyline[] lines)
31. {
32. OuterPolygon = **null**;
33. MBR = **new** MinBoundRect();
34. TopologyArcs = **new** List<TopoPolyline>();
35. InnerPolygons = **new** List<TopoPolygon>();
36. **this**.TopologyArcs.AddRange(lines);
37. List<**int**> ArcIDList = **new** List<**int**>();
38. **foreach** (var arc **in** lines)
39. { ArcIDList.Add(arc.ArcID); MBR.UpdateRect(arc.MBR); }
40. ArcIDList.Sort();
41. **int** hasgCode = 1;
42. **foreach** (var arcid **in** ArcIDList)
43. {
44. hasgCode \*= arcid;
45. }
46. **this**.PID = hasgCode.GetHashCode();
47. innerId = \_polygonID++;
48. }
50. **public** TopoPoint[] ConvertToPointArray()
51. {
52. List<TopoPoint> pointArray = **new** List<TopoPoint>();
53. TopoPoint b1 = TopologyArcs[0].BeginNode;
54. TopoPoint e1 = TopologyArcs[0].EndNode;
55. TopoPoint b2 = TopologyArcs.Last().BeginNode;
56. TopoPoint e2 = TopologyArcs.Last().EndNode;
57. TopoPoint beginPoint;
58. **if** (b1.PointID == b2.PointID)
59. beginPoint = b1;
60. **else** **if** (b1.PointID == e2.PointID)
61. beginPoint = b1;
62. **else** beginPoint = e1;
63. pointArray.Add(beginPoint);
64. **for** (**int** i = 0; i < TopologyArcs.Count; i++)
65. {
66. **int** direct = TopologyArcs[i].IsNode(beginPoint);
67. **if** (direct > 0)
68. pointArray.AddRange(TopologyArcs[i].MiddlePoint.ToArray());
69. **else**
70. {
71. List<TopoPoint> middlePoint = TopologyArcs[i].MiddlePoint;
72. **for** (**int** k = 0; k < middlePoint.Count; k++)
73. pointArray.Add(middlePoint[middlePoint.Count - 1 - k]);
74. }
75. beginPoint = TopologyArcs[i].GetAnotherNode(beginPoint);
76. pointArray.Add(beginPoint);
77. }
78. **return** pointArray.ToArray();
79. }
80. //计算面积
81. **public** **double** GetArea()
82. {
83. TopoPoint[] points = **this**.ConvertToPointArray();
84. var area = Math.Abs(points.Take(points.Length - 1)
85. .Select((p, i) => (points[i + 1].X - p.X) \* (points[i + 1].Y + p.Y))
86. .Sum() / 2);
87. **return** area;
88. }
89. //计算周长
90. **public** **double** GetPerimeter()
91. {
92. TopoPoint[] points = **this**.ConvertToPointArray();
93. **double** perimeter = 0;
94. **for** (**int** i = 0; i < points.Length - 1; i++)
95. { perimeter += points[i].GetDistance(points[i + 1]); }
96. **return** perimeter;
97. }
98. //判断点是否在区域内
99. **public** **bool** IfPointInRegion(TopoPoint todeterPoint)
100. {
101. TopoPoint[] points = **this**.ConvertToPointArray();
102. TopoPoint rayPoint = **new** TopoPoint(todeterPoint.X \* 2, todeterPoint.Y, todeterPoint.Z, **false**);
103. **int** intersectCount = 0;
104. **for** (**int** i = 0; i < points.Length - 1; i++)
105. { **if** (TopoPolygon.IntersectPoint(points[i], points[i + 1], todeterPoint, rayPoint) == **true**) intersectCount++; }
106. **return** (intersectCount / 2 != 0);
107. }
108. //判断点是否在区域内 法2
109. **public** **static** **bool** IntersectPoint(TopoPoint p1, TopoPoint p2, TopoPoint todeterPoint, TopoPoint rays)
110. {
111. **double** IntersectX =
112. ((p2.X - p1.X) \* (todeterPoint.X - rays.X) \* (todeterPoint.Y - p1.Y) -
113. todeterPoint.X \* (p2.X - p1.X) \* (todeterPoint.Y - rays.Y) +
114. p1.X \* (p2.Y - p1.Y) \* (todeterPoint.X - rays.X)) /
115. ((p2.Y - p1.Y) \* (todeterPoint.X - rays.X) -
116. (p2.X - p1.X) \* (todeterPoint.Y - rays.Y));
117. **double** IntersectY =
118. ((p2.Y - p1.Y) \* (todeterPoint.Y - rays.Y) \* (todeterPoint.X - p1.X) -
119. todeterPoint.Y \* (p2.Y - p1.Y) \* (todeterPoint.X - rays.X) +
120. p1.Y \* (p2.X - p1.X) \* (todeterPoint.Y - rays.Y)) /
121. ((p2.X - p1.X) \* (todeterPoint.Y - rays.Y) -
122. (p2.Y - p1.Y) \* (todeterPoint.X - rays.X));
123. **double** relativeE1 = 0;
124. **if** ((p2.X - p1.X) != 0)
125. relativeE1 = (IntersectX - p1.X) / (p2.X - p1.X);
126. **else** relativeE1 = (IntersectY - p1.Y) / (p2.Y - p1.Y);
127. **if** (0 <= relativeE1 && relativeE1 < 1)
128. **return** **true**;
129. **else** **return** **false**;
130. }
131. }
132. }

# TopoPolygonSet.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.IO;
4. **using** System.Linq;
5. **using** System.Text;
6. **using** System.Threading.Tasks;
8. **namespace** AGIS\_work.DataStructure
9. {
10. //拓扑多边形集合
11. **public** **class** TopoPolygonSet
12. {
13. **public** List<TopoPolygon> TopoPolygonList { **get**; **private** **set**; }
15. **public** TopoPolygonSet()
16. { **this**.TopoPolygonList = **new** List<TopoPolygon>(); }
17. **public** TopoPolygonSet(TopoPolygon[] gons)
18. {
19. **this**.TopoPolygonList = **new** List<TopoPolygon>();
20. **for** (**int** i = 0; i < gons.Length; i++)
21. {
22. **if** (**this**.IsPolygonExist(gons[i].PID) == **false**)
23. TopoPolygonList.Add(gons[i]);
24. }
25. }
26. //二次筛查
27. **public** **void** Recheck(**double** regionArea)
28. {
29. **double** area\_max = 0;
30. **int** pid\_max\_index = -1;
31. **for** (**int** i = 0; i < TopoPolygonList.Count; i++)
32. {
33. **double** area = TopoPolygonList[i].GetArea();
34. **if** (area > area\_max)
35. { area\_max = area; pid\_max\_index = i; }
36. }
37. **if** (pid\_max\_index != -1 && area\_max > 0.5 \* regionArea)
38. TopoPolygonList.RemoveAt(pid\_max\_index);
39. }
40. //判断是否存在对应id的多边形
41. **public** **bool** IsPolygonExist(**int** pid)
42. {
43. **foreach** (var polygon **in** TopoPolygonList)
44. { **if** (polygon.PID == pid) **return** **true**; }
45. **return** **false**;
46. }
47. //获取选中的多边形
48. **public** TopoPolygon GetClickPointInsidePolygon(TopoPoint clickPoint)
49. {
50. **foreach** (var polygon **in** **this**.TopoPolygonList)
51. { **if** (polygon.IfPointInRegion(clickPoint)) **return** polygon; }
52. **return** **null**;
53. }
54. //导出多边形拓扑关系表至文件
55. **public** **void** SavePolygonTableToFile(**string** filename)
56. {
57. StreamWriter sw = **new** StreamWriter(filename);
58. sw.WriteLine(**string**.Format("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}\t{9}",
59. "ID", "Pol\_ID", "Arc\_Num", "ArcIds", "OuterPol\_ID", "InnerPol\_ID", "LX", "LY", "RX", "RY"));
60. **foreach** (var polygon **in** TopoPolygonList)
61. {
62. **string** arcids = " ";
63. **foreach** (var arc **in** polygon.TopologyArcs)
64. { arcids += arc.ArcID + ","; }
65. sw.WriteLine(**string**.Format("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}\t{9}",
66. polygon.innerId, polygon.PID, polygon.TopologyArcs.Count, arcids.Remove(arcids.Length - 1),
67. (polygon.OuterPolygon == **null**) ? "NULL" : polygon.OuterPolygon.PID.ToString(),
68. (polygon.InnerPolygons.Count == 0) ? "NULL" : polygon.InnerPolygons[0].PID.ToString(),
69. polygon.MBR.MinX, polygon.MBR.MinY, polygon.MBR.MaxX, polygon.MBR.MaxY));
70. }
71. sw.Close();
72. }
73. }
74. }

# TopoPolyline.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. //拓扑边
10. **public** **class** TopoPolyline
11. {
12. **public** **int** ArcID { **get**; **private** **set**; }//唯一标识码
13. **private** **static** **int** \_ArcID = 0;
14. **public** **int** Innerid { **get**; **private** **set**; }//内部码
15. **public** TopoPoint BeginNode { **get**; **private** **set**; }//起始节点
16. **public** TopoPoint EndNode { **get**; **private** **set**; }//终止节点
17. **public** List<TopoPoint> MiddlePoint { **get**; **private** **set**; }//中间点序列
18. **public** TopoPolygon LeftPolygon { **get**; **set**; }//左多边形
19. **public** TopoPolygon RightPolygon { **get**; **set**; }//右多边形
20. **public** MinBoundRect MBR { **get**; **private** **set**; }
22. **public** TopoPolyline()
23. {
24. **this**.ArcID = \_ArcID++;
25. MiddlePoint = **new** List<TopoPoint>();
26. Innerid = **this**.ArcID;
27. MBR = **new** MinBoundRect();
28. }
30. **public** TopoPolyline(ContourPolyline polyline)
31. {
32. **this**.ArcID = polyline.PID;
33. Innerid = \_ArcID++;
34. MiddlePoint = **new** List<TopoPoint>();
35. MBR = **new** MinBoundRect();
36. **if** (polyline.PointList.Count >= 2)
37. {
38. TopoPoint startPoint = **new** TopoPoint(polyline.PointList.First(), **true**);
39. startPoint.TopologyArcs.Add(**this**);
40. **this**.BeginNode = startPoint;
41. TopoPoint endPoint = **new** TopoPoint(polyline.PointList.Last(), **true**);
42. endPoint.TopologyArcs.Add(**this**);
43. **this**.EndNode = endPoint;
44. MBR.UpdateRect(startPoint.X, startPoint.Y);
45. MBR.UpdateRect(endPoint.X, endPoint.Y);
46. **for** (**int** i = 1; i < polyline.PointList.Count - 1; i++)
47. {
48. **if** (polyline.PointList[i].OID != startPoint.PointID &&
49. polyline.PointList[i].OID != EndNode.PointID)
50. {
51. TopoPoint midPoint = **new** TopoPoint(polyline.PointList[i], **false**);
52. MiddlePoint.Add(midPoint);
53. midPoint.TopologyArcs.Add(**this**);
54. }
55. MBR.UpdateRect(polyline.PointList[i].X, polyline.PointList[i].Y);
56. }
57. }
58. }
60. **public** TopoPolyline(Edge edge)
61. {
62. **this**.ArcID = edge.EID;
63. MiddlePoint = **new** List<TopoPoint>();
64. MBR = **new** MinBoundRect();
65. TopoPoint startPoint = **new** TopoPoint(edge.StartPoint, **true**);
66. startPoint.TopologyArcs.Add(**this**);
67. **this**.BeginNode = startPoint;
68. TopoPoint endPoint = **new** TopoPoint(edge.EndPoint, **true**);
69. endPoint.TopologyArcs.Add(**this**);
70. **this**.EndNode = endPoint;
71. MBR.UpdateRect(startPoint.X, startPoint.Y);
72. MBR.UpdateRect(endPoint.X, endPoint.Y);
73. }
74. //获取另一结点
75. **public** TopoPoint GetAnotherNode(TopoPoint p)
76. {
77. **if** (**this**.BeginNode.PointID == p.PointID)
78. **return** **this**.EndNode;
79. **else** **if** (**this**.EndNode.PointID == p.PointID)
80. **return** **this**.BeginNode;
81. **else** **return** **null**;
82. }
83. //获取起始节点的角度
84. **public** **double** GetBeginNodeAngle()
85. {
86. **if** (MiddlePoint.Count < 1) **return** **this**.BeginNode.GetPositon(**this**.EndNode);
87. **else** **return** **this**.BeginNode.GetPositon(**this**.MiddlePoint[0]);
88. }
89. //获取终止节点的角度
90. **public** **double** GetEndNodeAngle()
91. {
92. **if** (MiddlePoint.Count < 1) **return** **this**.EndNode.GetPositon(**this**.BeginNode);
93. **else** **return** **this**.EndNode.GetPositon(**this**.MiddlePoint[0]);
94. }
95. //判断是否是结点
96. **public** **int** IsNode(TopoPoint p)
97. {
98. **if** (**this**.BeginNode.PointID == p.PointID) **return** 1;
99. **else** **if** (**this**.EndNode.PointID == p.PointID) **return** -1;
100. **else** **return** 0;
101. }
103. **public** **override** **string** ToString()
104. {
105. **return** **string**.Format("ArcID:{0},StartID:{1},EndID:{2},MPointCount:{3}",
106. **this**.ArcID, **this**.BeginNode.PointID, **this**.EndNode.PointID, **this**.MiddlePoint.Count);
107. }
108. }
109. }

# TopoPolylineSet.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.IO;
4. **using** System.Linq;
5. **using** System.Text;
6. **using** System.Threading.Tasks;
8. **namespace** AGIS\_work.DataStructure
9. {
10. //拓扑边集合
11. **public** **class** TopoPolylineSet
12. {
13. **public** List<TopoPolyline> TopoPolylineList { **get**; **private** **set**; }
15. **public** TopoPolylineSet()
16. {
17. **this**.TopoPolylineList = **new** List<TopoPolyline>();
18. }
19. **public** TopoPolylineSet(TopoPolyline[] lines)
20. {
21. **this**.TopoPolylineList = **new** List<TopoPolyline>();
22. **this**.TopoPolylineList.AddRange(lines);
23. }
24. **public** **override** **string** ToString()
25. { **return** **base**.ToString(); }
26. //保存边拓扑关系至文件
27. **public** **void** SavePolylineTableToFile(**string** filename)
28. {
29. StreamWriter sw = **new** StreamWriter(filename);
30. sw.WriteLine(**string**.Format("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}\t{9}\t{10}\t{11}",
31. "ID", "Arc\_ID", "BeginNode", "EndNode", "LeftPolygon", "RightPolygon", "MiddlePtsNum",
32. "MiddlePtsCooridinate", "LX", "LY", "RX", "RY"));
33. **foreach** (var line **in** TopoPolylineList)
34. {
35. **string** middleCoorinate = " ";
36. **foreach** (var mpoint **in** line.MiddlePoint)
37. middleCoorinate += **string**.Format("({0},{1},{2}),", mpoint.X, mpoint.Y, mpoint.Z);
38. sw.WriteLine(**string**.Format("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}\t{9}\t{10}\t{11}",
39. line.Innerid, line.ArcID, line.BeginNode.PointID, line.EndNode.PointID,
40. (line.LeftPolygon == **null**) ? "NULL" : line.LeftPolygon.PID.ToString(),
41. (line.RightPolygon == **null**) ? "NULL" : line.RightPolygon.PID.ToString(),
42. line.MiddlePoint.Count, middleCoorinate.Remove(middleCoorinate.Length - 1),
43. line.MBR.MinX, line.MBR.MinY, line.MBR.MaxX, line.MBR.MaxY));
44. }
45. sw.Close();
46. }
47. }
48. }

# Triangle.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. //三角形
10. **public** **class** Triangle
11. {
12. **public** **int** TID { **get**; **private** **set**; }//标识码
13. **public** DataPoint VertexA { **get**; **private** **set**; }//A顶点
14. **public** DataPoint VertexB { **get**; **private** **set**; }//B定点
15. **public** DataPoint VertexC { **get**; **private** **set**; }//C顶点
17. **public** Triangle(DataPoint v0, DataPoint v1, DataPoint v2, **int** tid)
18. {
19. **this**.VertexA = v0;
20. **this**.VertexB = v1;
21. **this**.VertexC = v2;
22. **this**.TID = tid;
23. }
24. //判断点是否在三角形内
25. **public** **bool** IsPointInTriangle(DataPoint p)
26. {
27. Vector2D vectorPA = VertexA - p;
28. Vector2D vectorPB = VertexB - p;
29. Vector2D vectorPC = VertexC - p;
30. **double** vPaPb = vectorPA.CrossProduct(vectorPB);
31. **double** vPbPc = vectorPB.CrossProduct(vectorPC);
32. **double** vPcPa = vectorPC.CrossProduct(vectorPA);
33. **return** (vPaPb > 0 && vPbPc > 0 && vPcPa > 0) || (vPaPb < 0 && vPbPc < 0 && vPcPa < 0);
34. }
35. //判断是否全等
36. **public** **bool** IsEqulesTri(**int** oid1, **int** oid2, **int** oid3)
37. {
38. List<**int**> TriOIDList = **new** List<**int**>();
39. TriOIDList.Add(VertexA.OID);
40. TriOIDList.Add(VertexB.OID);
41. TriOIDList.Add(VertexC.OID);
42. TriOIDList.Sort();
43. List<**int**> VirOIDList = **new** List<**int**>();
44. VirOIDList.Add(oid1);
45. VirOIDList.Add(oid2);
46. VirOIDList.Add(oid3);
47. VirOIDList.Sort();
48. **return** (TriOIDList[0] == VirOIDList[0]) &&
49. (TriOIDList[1] == VirOIDList[1]) &&
50. (TriOIDList[2] == VirOIDList[2]);
51. }
52. //获取三角形内等值线
53. **public** Edge GetContourLine(**double** elevation)
54. {
55. List<DataPoint> points = **new** List<DataPoint>();
56. **if** ((elevation - VertexA.Value) \* (elevation - VertexB.Value) \* (elevation - VertexC.Value) == 0)
57. elevation += 0.1;
58. **if** ((elevation - VertexA.Value) \* (elevation - VertexB.Value) < 0)
59. {
60. **double** EleX = VertexA.X + (VertexB.X - VertexA.X) \* (elevation - VertexA.Value) / (VertexB.Value - VertexA.Value);
61. **double** EleY = VertexA.Y + (VertexB.Y - VertexA.Y) \* (elevation - VertexA.Value) / (VertexB.Value - VertexA.Value);
62. DataPoint p1 = **new** DataPoint(VertexA.OID \* 1000 + VertexB.OID,
63. "ContourPoint\_" + VertexA.OID \* 1000 + VertexB.OID,
64. EleX, EleY, elevation);
65. points.Add(p1);
66. }
67. **if** ((elevation - VertexA.Value) \* (elevation - VertexC.Value) < 0)
68. {
69. **double** EleX = VertexA.X + (VertexC.X - VertexA.X) \* (elevation - VertexA.Value) / (VertexC.Value - VertexA.Value);
70. **double** EleY = VertexA.Y + (VertexC.Y - VertexA.Y) \* (elevation - VertexA.Value) / (VertexC.Value - VertexA.Value);
71. DataPoint p1 = **new** DataPoint(VertexA.OID \* 1000 + VertexC.OID,
72. "ContourPoint\_" + VertexA.OID \* 1000 + VertexC.OID,
73. EleX, EleY, elevation);
74. points.Add(p1);
75. }
76. **if** ((elevation - VertexC.Value) \* (elevation - VertexB.Value) < 0)
77. {
78. **double** EleX = VertexC.X + (VertexB.X - VertexC.X) \* (elevation - VertexC.Value) / (VertexB.Value - VertexC.Value);
79. **double** EleY = VertexC.Y + (VertexB.Y - VertexC.Y) \* (elevation - VertexC.Value) / (VertexB.Value - VertexC.Value);
80. DataPoint p1 = **new** DataPoint(VertexC.OID \* 1000 + VertexB.OID,
81. "ContourPoint\_" + VertexC.OID \* 1000 + VertexB.OID,
82. EleX, EleY, elevation);
83. points.Add(p1);
84. }
85. **if** (points.Count == 2)
86. { **return** **new** Edge(points[0], points[1]); }
87. **else** **return** **null**;
88. }
89. }
90. }

# TriangleSet.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. //三角形集合
10. **public** **class** TriangleSet
11. {
12. **public** List<Triangle> TriangleList = **new** List<Triangle>();
13. **public** TriangleSet() { }
15. /// <summary>
16. /// 移除指定TID的三角形
17. /// </summary>
18. /// <param name="tid"></param>
19. **public** **void** RemoveTriangleByTID(**int** tid)
20. {
21. **int** index = 0;
22. **foreach** (var tri **in** TriangleList)
23. { **if** (tri.TID == tid) **break**; index++; }
24. TriangleList.RemoveAt(index);
25. }
26. //添加一个三角形
27. **public** **void** AddTriangle(Triangle t)
28. { TriangleList.Add(t); }
29. //求点所在三角形
30. **public** Triangle GetPointInsidesTri(DataPoint p)
31. {
32. **foreach** (var tri **in** TriangleList)
33. { **if** (tri.IsPointInTriangle(p)) **return** tri; }
34. **return** **null**;
35. }
36. //判断是否已经存在
37. **public** **bool** IsTriAlreadyExists(**int** oid1, **int** oid2, **int** oid3)
38. {
39. **foreach** (var tri **in** TriangleList)
40. { **if** (tri.IsEqulesTri(oid1, oid2, oid3)) **return** **true**; }
41. **return** **false**;
42. }
43. }
44. }

# Vector2D.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
7. **namespace** AGIS\_work.DataStructure
8. {
9. //二维向量
10. **public** **class** Vector2D
11. {
12. **public** **double** X { **get**; **private** **set**; }
13. **public** **double** Y { **get**; **private** **set**; }
15. **public** Vector2D(**double** x, **double** y)
16. {
17. **this**.X = x;
18. **this**.Y = y;
19. }
20. //叉乘
21. **public** **double** CrossProduct(Vector2D v2)
22. { **return** **this**.X \* v2.Y - **this**.Y \* v2.X; }
23. //点乘
24. **public** **double** DotProduct(Vector2D v2)
25. { **return** **this**.X \* v2.X + **this**.Y \* v2.Y; }
26. //向量相加
27. **public** **static** Vector2D **operator** +(Vector2D v1, Vector2D v2)
28. { **return** **new** Vector2D(v1.X + v2.X, v1.Y + v2.Y); }
29. }
30. }

# CreateTIN.cs

1. **using** AGIS\_work.DataStructure;
2. **using** System;
3. **using** System.Collections;
4. **using** System.Collections.Generic;
5. **using** System.Drawing;
6. **using** System.Linq;
7. **using** System.Text;
8. **using** System.Threading.Tasks;
10. **namespace** AGIS\_work.Mehtod
11. {
12. //简化的三角形边
13. **class** TinLine
14. {
15. **public** DataPoint Begin { **get**; **internal** **set**; }
16. **public** DataPoint End { **get**; **internal** **set**; }
17. }
18. //构建TIN模型
19. **public** **class** CreateTIN
20. {
21. **public** PointSet mPointSet;//点集
22. **private** PointF[] arrDots;//点序列
23. **private** ArrayList arrEdges = **new** ArrayList();//边序列
24. **private** ArrayList arrTris = **new** ArrayList();//三角形序列
26. **public** CreateTIN(PointSet pointSet)
27. { **this**.mPointSet = pointSet; }
28. //逐点插入法2
29. **public** Edge[] PointByPointInsertion()
30. {
31. EdgeSet sEdgeSet = **new** EdgeSet();
32. TriangleSet sTriangleSet = **new** TriangleSet();
33. MinBoundRect sMBR = **this**.mPointSet.MBR;
34. **double** width = sMBR.MaxX - sMBR.MinX;
35. **double** height = sMBR.MaxY - sMBR.MinY;
36. **double** middlePointX = (sMBR.MaxX + sMBR.MinX) / 2;
37. **double** middlePointY = sMBR.MinY;
38. DataPoint P0 = **new** DataPoint(-1, "P0", middlePointX - width, middlePointY, 0);
39. DataPoint P1 = **new** DataPoint(-2, "P1", middlePointX + width, middlePointY, 0);
40. DataPoint P2 = **new** DataPoint(-3, "P2", middlePointX, middlePointY + 2 \* height, 0);
41. Triangle T0 = **new** Triangle(P0, P1, P2, -1);
42. sTriangleSet.AddTriangle(T0);
43. sEdgeSet.AddEdge(**new** Edge(P0, P1));
44. sEdgeSet.AddEdge(**new** Edge(P1, P2));
45. sEdgeSet.AddEdge(**new** Edge(P1, P0));
46. **foreach** (var point **in** mPointSet.PointList)
47. {
48. Triangle CurTri = sTriangleSet.GetPointInsidesTri(point);
49. **if** (CurTri != **null**) { }
50. }
51. **return** sEdgeSet.EdgeList.ToArray();
52. }
53. //逐点插入法2
54. **public** Edge[] PointByPointInsertion2()
55. {
56. **double** ang;
57. ArrayList tinlines = **new** ArrayList();
58. //定义与第一点最近的点
59. List<DataPoint> pointList = **this**.mPointSet.PointList;
60. **double** mindis = pointList[0].GetDistance(pointList[1]);
61. **double** dis;
62. **int** count = 1;
63. TinLine tl = **new** TinLine();
64. **for** (**int** i = 1; i < pointList.Count; i++)
65. {
66. dis = pointList[0].GetDistance(pointList[i]);
67. **if** (dis < mindis)
68. { mindis = dis; count = i; }
69. }
70. //将第一条边反向已进行三角形扩展
71. tl.Begin = (DataPoint)pointList[0];
72. tl.End = (DataPoint)pointList[count];
73. tinlines.Add(tl);
74. TinLine line = **new** TinLine();
75. DataPoint a = ((TinLine)tinlines[0]).Begin;
76. DataPoint b = ((TinLine)tinlines[0]).End;
77. line.Begin = b;
78. line.End = a;
79. tinlines.Add(line);
80. //对每一条边进行扩展
81. **for** (**int** j = 0; j < tinlines.Count; j++)
82. {
83. **double** minang = 0;
84. **bool** OK;
85. OK = **false**;
86. TinLine tling1 = **new** TinLine();
87. TinLine tling2 = **new** TinLine();
88. **for** (**int** i = 0; i < pointList.Count; i++)
89. {
90. **int** youbian;
91. //判断第三点与前两点的位置关系
92. youbian = DataPoint.LeftOrRight((DataPoint)pointList[i], ((TinLine)tinlines[j]).Begin, ((TinLine)tinlines[j]).End);
93. **if** (youbian == 1)
94. {
95. //获取角度最大点
96. ang = DataPoint.Angle((DataPoint)pointList[i], ((TinLine)tinlines[j]).Begin, ((TinLine)tinlines[j]).End);
97. **if** (ang > minang) { minang = ang; count = i; }
98. OK = **true**;
99. }
100. }
101. **if** (OK == **true**)
102. {
103. //将新生成两条边添加入集合中
104. **int** t1 = 0;
105. **int** t2 = 0;
106. tling1.Begin = ((TinLine)tinlines[j]).Begin;
107. tling1.End = (DataPoint)pointList[count];
108. tling2.Begin = (DataPoint)pointList[count];
109. tling2.End = ((TinLine)tinlines[j]).End;
110. tinlines.Add(tling1);
111. tinlines.Add(tling2);
112. **for** (**int** i = 0; i < tinlines.Count - 2; i++)
113. {
114. //判断新生成的两边是否与已生成的边重合
115. **if** ((tling2.Begin == ((TinLine)tinlines[i]).Begin && tling2.End == ((TinLine)tinlines[i]).End) ||
116. (tling2.Begin == ((TinLine)tinlines[i]).End && tling2.End == ((TinLine)tinlines[i]).Begin))
117. { t2 = 1; }
118. **if** ((tling1.Begin == ((TinLine)tinlines[i]).Begin && tling1.End == ((TinLine)tinlines[i]).End) ||
119. (tling1.Begin == ((TinLine)tinlines[i]).End && tling1.End == ((TinLine)tinlines[i]).Begin))
120. { t1 = 1; }
121. }
122. //两条边都重合
123. **if** (t2 == 1 && t1 == 1)
124. { **for** (**int** i = 0; i < 2; i++) { tinlines.Remove(tinlines[tinlines.Count - 1]); } }
125. //第二条边重合
126. **else** **if** (t2 == 1) { tinlines.Remove(tinlines[tinlines.Count - 1]); }
127. //第一条边重合
128. **else** **if** (t1 == 1) { tinlines.Remove(tinlines[tinlines.Count - 2]); }
129. }
130. }
131. tinlines.Remove(tinlines[0]);//将集合中的第一条边删除
132. List<Edge> ResultEdge = **new** List<Edge>();
133. **int** eid = 1;
134. **foreach** (var tinLine **in** tinlines)
135. { ResultEdge.Add(**new** Edge(((TinLine)tinLine).Begin, ((TinLine)tinLine).End)); eid++; }
136. **return** ResultEdge.ToArray();
137. }
138. //简化的边类
139. **public** **class** Edge2
140. {
141. **public** **int** Start;//边的起点
142. **public** **int** End;//边的终点
143. **public** **int** LeftTri = -1;//左三角形索引
144. **public** **int** RightTri = -1;//右三角形索引
145. }
146. //简化的三角形类
147. **public** **class** Tri
148. {
149. **public** **int** NodeA;
150. **public** **int** NodeB;
151. **public** **int** NodeC;
152. **public** **int** AdjTriA = -1;
153. **public** **int** AdjTriB = -1;
154. **public** **int** AdjTriC = -1;
155. }
156. //生成三角网TIN
157. **public** List<Edge> GeneTIN()
158. {
159. arrEdges.Clear();
160. arrTris.Clear();
161. arrDots = **new** PointF[mPointSet.PointList.Count];
162. **for** (**int** kk = 0; kk < mPointSet.PointList.Count; kk++)
163. { arrDots[kk] = **new** PointF((**float**)mPointSet.PointList[kk].X, (**float**)mPointSet.PointList[kk].Y); }
164. **int** i, idxStart = 0, endTemp, ptindex;
165. **bool** isExist;
166. **double** angMax, angMin, angTemp, angRcdMax, angRcdTmp, lenMin, lenCur, lenTmp1, lenTmp2;
167. Edge2 edge = **new** Edge2();
168. //找到边界---（删除不需要的点，从X最小的地方开始找，直至回到起始点）
169. PointF dirCur = **new** PointF();
170. PointF dirTmp1 = **new** PointF();
171. PointF dirTmp2 = **new** PointF();
172. PointF ptStart = **new** PointF();
173. **for** (i = 1; i < arrDots.Length; i++)
174. { **if** (arrDots[i].X < arrDots[idxStart].X) { idxStart = i; } }
175. endTemp = idxStart - 1;
176. ptStart.X = arrDots[idxStart].X;
177. ptStart.Y = arrDots[idxStart].Y;
178. edge.Start = idxStart;
179. angMin = Math.PI;
180. dirCur.X = 0;
181. dirCur.Y = 500;
182. **while** (endTemp != idxStart)
183. {
184. lenCur = Math.Sqrt(dirCur.X \* dirCur.X + dirCur.Y \* dirCur.Y);
185. lenMin = 1000;
186. **for** (i = 0; i < arrDots.Length; i++)//找边界
187. {
188. **if** (i != edge.Start)
189. {
190. dirTmp1.X = arrDots[i].X - ptStart.X;
191. dirTmp1.Y = arrDots[i].Y - ptStart.Y;
192. lenTmp1 = Math.Sqrt(dirTmp1.X \* dirTmp1.X + dirTmp1.Y \* dirTmp1.Y);
193. angTemp = Math.Acos((dirCur.X \* dirTmp1.X + dirCur.Y \* dirTmp1.Y) / (lenTmp1 \* lenCur));
194. **if** (angTemp < angMin)
195. { angMin = angTemp; edge.End = i; lenMin = lenTmp1; }
196. **else** **if** (angTemp == angMin && lenTmp1 < lenMin)
197. { edge.End = i; lenMin = lenTmp1; }
198. }
199. }
200. arrEdges.Add(edge);
201. endTemp = edge.End;
202. edge = **new** Edge2();
203. angMin = Math.PI;
204. dirCur.X = arrDots[endTemp].X - ptStart.X;
205. dirCur.Y = arrDots[endTemp].Y - ptStart.Y;
206. ptStart = arrDots[endTemp];
207. edge.Start = endTemp;
208. }
209. //以下为自动生成TIN
210. //从第一条边开始，按照先左后右的顺序寻找，找到则加入三角形数组和边数组，没有则继续下一边，直到边到达最后
211. //注意边可能有两种顺序存储。
212. **for** (i = 0; i < arrEdges.Count; i++)
213. {
214. //取出一条边
215. edge = **new** Edge2();
216. edge = (Edge2)arrEdges[i];
217. //先左后右计算扩展点-判断三角形是否存在过（若本边的左三角已存在，则计算右三角）？？
218. **if** (edge.LeftTri == -1)
219. {
220. ptindex = -1;//选中的点的index
221. dirCur.X = arrDots[edge.End].X - arrDots[edge.Start].X;
222. dirCur.Y = arrDots[edge.End].Y - arrDots[edge.Start].Y;
223. angRcdMax = 0;//与该边夹角最大值
224. angMax = 0;//最大圆内接角
225. **for** (**int** j = 0; j < arrDots.Length; j++)
226. {
227. **if** (j != edge.Start && j != edge.End)//排除边的端点
228. {
229. dirTmp1.X = arrDots[j].X - arrDots[edge.Start].X;
230. dirTmp1.Y = arrDots[j].Y - arrDots[edge.Start].Y;
231. **if** (dirCur.X \* dirTmp1.Y - dirCur.Y \* dirTmp1.X < 0)//如果该点在左边，则计算
232. {
233. //找角度最大的
234. lenCur = Math.Sqrt(dirCur.X \* dirCur.X + dirCur.Y \* dirCur.Y);//当前向量长度
235. lenTmp1 = Math.Sqrt(dirTmp1.X \* dirTmp1.X + dirTmp1.Y \* dirTmp1.Y);
236. dirTmp2.X = arrDots[j].X - arrDots[edge.End].X;
237. dirTmp2.Y = arrDots[j].Y - arrDots[edge.End].Y;
238. lenTmp2 = Math.Sqrt(dirTmp2.X \* dirTmp2.X + dirTmp2.Y \* dirTmp2.Y);
239. angRcdTmp = Math.Acos((dirCur.X \* dirTmp1.X + dirCur.Y \* dirTmp1.Y) / (lenTmp1 \* lenCur));
240. angTemp = Math.Acos((dirTmp2.X \* dirTmp1.X + dirTmp2.Y \* dirTmp1.Y) / (lenTmp1 \* lenTmp2));
241. **if** (angTemp > angMax)
242. { angMax = angTemp; angRcdMax = angRcdTmp; ptindex = j; }
243. **else** **if** (angTemp == angMax && angRcdMax < angRcdTmp)//相等取最左
244. { angRcdMax = angRcdTmp; ptindex = j; }
245. }
246. }
247. }
248. **if** (ptindex != -1)//选择有点
249. {
250. //记录三角形
251. Tri tri = **new** Tri();
252. tri.NodeA = edge.Start;
253. tri.NodeB = edge.End;
254. tri.NodeC = ptindex;
255. edge.LeftTri = arrTris.Count;
256. isExist = **false**;
257. //记录边1-需要检索是否存在过这条边-由于每条边都先有左三角形，如有三角形加入，必定为右三角形
258. **for** (**int** k = 0; k < arrEdges.Count; k++)
259. {
260. Edge2 e = (Edge2)arrEdges[k];
261. **if** (e.Start == edge.Start && e.End == ptindex)//如果存在过这条边，则记录其右三角形
262. {
263. e.RightTri = arrTris.Count;
264. tri.AdjTriB = e.LeftTri;
265. isExist = **true**;
266. **break**;
267. }
268. **else** **if** (e.Start == ptindex && e.End == edge.Start)
269. {
270. e.LeftTri = arrTris.Count;
271. tri.AdjTriB = e.RightTri;
272. isExist = **true**;
273. **break**;
274. }
275. }
276. **if** (isExist == **false**)//如果不存在这条边，则新建一条边
277. {
278. Edge2 edgeadd = **new** Edge2();
279. edgeadd.Start = ptindex;
280. edgeadd.End = edge.Start;
281. edgeadd.LeftTri = arrTris.Count;
282. arrEdges.Add(edgeadd);
283. }
284. isExist = **false**;
285. //记录边2
286. **for** (**int** k = 0; k < arrEdges.Count; k++)
287. {
288. Edge2 e = (Edge2)arrEdges[k];
289. **if** (e.Start == ptindex && e.End == edge.End)//如果存在过这条边，则记录其右三角形
290. {
291. e.RightTri = arrTris.Count;
292. tri.AdjTriA = e.LeftTri;
293. isExist = **true**;
294. **break**;
295. }
296. **else** **if** (e.Start == edge.End && e.End == ptindex)
297. {
298. e.LeftTri = arrTris.Count;
299. tri.AdjTriA = e.RightTri;
300. isExist = **true**;
301. **break**;
302. }
303. }
304. **if** (isExist == **false**)//如果不存在这条边，则新建一条边
305. {
306. Edge2 edgeadd = **new** Edge2();
307. edgeadd.Start = edge.End;
308. edgeadd.End = ptindex;
309. edgeadd.LeftTri = arrTris.Count;
310. arrEdges.Add(edgeadd);
311. }
312. tri.AdjTriC = edge.RightTri;//如果edge的右三角形不存在，由if进来可见左三角也不存在，这只能是边界，从而tri.AdjTriC=-1合理
313. arrTris.Add(tri);//add the tri to the arraylist
314. }
315. }
316. **else** **if** (edge.RightTri == -1)//由于最开始的那部分都是边界，只有一个三角形；以后的边都已存在一个三角形，也仅剩余一个，故可以else if
317. {
318. //仅在右边找
319. ptindex = -1;//选中的点的index
320. dirCur.X = arrDots[edge.End].X - arrDots[edge.Start].X;
321. dirCur.Y = arrDots[edge.End].Y - arrDots[edge.Start].Y;
322. angMax = 0;//最大角度
323. angRcdMax = 0;//与该边夹角最大值
324. **for** (**int** j = 0; j < arrDots.Length; j++)
325. {
326. **if** (j != edge.Start && j != edge.End)//排除边的端点
327. {
328. lenCur = Math.Sqrt(dirCur.X \* dirCur.X + dirCur.Y \* dirCur.Y);//当前向量长度
329. dirTmp1.X = arrDots[j].X - arrDots[edge.Start].X;
330. dirTmp1.Y = arrDots[j].Y - arrDots[edge.Start].Y;
331. **if** (dirCur.X \* dirTmp1.Y - dirCur.Y \* dirTmp1.X > 0)//如果该点在右边，则计算
332. {
333. //找角度最大的
334. lenTmp1 = Math.Sqrt(dirTmp1.X \* dirTmp1.X + dirTmp1.Y \* dirTmp1.Y);
336. dirTmp2.X = arrDots[j].X - arrDots[edge.End].X;
337. dirTmp2.Y = arrDots[j].Y - arrDots[edge.End].Y;
338. lenTmp2 = Math.Sqrt(dirTmp2.X \* dirTmp2.X + dirTmp2.Y \* dirTmp2.Y);
339. angRcdTmp = Math.Acos((dirCur.X \* dirTmp1.X + dirCur.Y \* dirTmp1.Y) / (lenTmp1 \* lenCur));
340. angTemp = Math.Acos((dirTmp2.X \* dirTmp1.X + dirTmp2.Y \* dirTmp1.Y) / (lenTmp1 \* lenTmp2));
341. **if** (angTemp > angMax)
342. { angMax = angTemp; angRcdMax = angRcdTmp; ptindex = j; }
343. **else** **if** (angTemp == angMax && angRcdTmp > angTemp)//相等取最左
344. { angRcdTmp = angTemp; ptindex = j; }
345. }
346. }
347. }
348. **if** (ptindex != -1)//选择有点
349. {
350. //记录三角形
351. //记录三角形
352. Tri tri = **new** Tri();
353. tri.NodeA = edge.Start;
354. tri.NodeB = edge.End;
355. tri.NodeC = ptindex;
356. edge.RightTri = arrTris.Count;
357. isExist = **false**;
358. //记录边1-需要检索是否存在过这条边-由于每条边都先有左三角形，如有三角形加入，必定为右三角形
359. **for** (**int** k = 0; k < arrEdges.Count; k++)
360. {
361. Edge2 e = (Edge2)arrEdges[k];
362. **if** (e.Start == ptindex && e.End == edge.Start)//如果存在过这条边，则记录其右三角形
363. {
364. e.RightTri = arrTris.Count;
365. tri.AdjTriB = e.LeftTri;
366. isExist = **true**;
367. **break**;
368. }
369. **else** **if** (e.Start == edge.Start && e.End == ptindex)
370. {
371. e.LeftTri = arrTris.Count;
372. tri.AdjTriB = e.RightTri;
373. isExist = **true**;
374. **break**;
375. }
376. }
377. **if** (isExist == **false**)//如果不存在这条边，则新建一条边
378. {
379. Edge2 edgeadd = **new** Edge2();
380. edgeadd.Start = edge.Start;
381. edgeadd.End = ptindex;
382. edgeadd.LeftTri = arrTris.Count;
383. arrEdges.Add(edgeadd);
384. }
385. isExist = **false**;
386. //记录边2
387. **for** (**int** k = 0; k < arrEdges.Count; k++)
388. {
389. Edge2 e = (Edge2)arrEdges[k];
390. **if** (e.Start == edge.End && e.End == ptindex)//如果存在过这条边，则记录其右三角形
391. {
392. e.RightTri = arrTris.Count;
393. tri.AdjTriA = e.LeftTri;
394. isExist = **true**;
395. **break**;
396. }
397. **else** **if** (e.Start == ptindex && e.End == edge.End)
398. {
399. e.LeftTri = arrTris.Count;
400. tri.AdjTriA = e.RightTri;
401. isExist = **true**;
402. **break**;
403. }
404. }
405. **if** (isExist == **false**)//如果不存在这条边，则新建一条边
406. {
407. Edge2 edgeadd = **new** Edge2();
408. edgeadd.Start = ptindex;
409. edgeadd.End = edge.End;
410. edgeadd.LeftTri = arrTris.Count;
411. arrEdges.Add(edgeadd);
412. }
413. tri.AdjTriC = edge.LeftTri;//如果edge的左三角形不存在，由if进来可见右三角也不存在，这只能是边界，从而tri.AdjTriC=-1合理
414. arrTris.Add(tri);//add the tri to the arraylist
415. }
416. }
417. }
418. List<Edge> EdgeList = **new** List<Edge>();
419. **for** (**int** gg = 0; gg < arrEdges.Count; gg++)
420. {
421. Edge2 eg = (Edge2)arrEdges[gg];
422. PointF pt1, pt2;
423. pt1 = arrDots[eg.Start];
424. pt2 = arrDots[eg.End];
425. EdgeList.Add(**new** Edge(
426. **new** DataPoint(gg, gg.ToString(), pt1.X, pt1.Y, 0),
427. **new** DataPoint(-gg, (-gg).ToString(), pt2.X, pt2.Y, 0)));
428. }
429. **return** EdgeList;
430. }
431. }
432. }

# GridCreateContourLine.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Text;
5. **using** System.Threading.Tasks;
6. **using** AGIS\_work.DataStructure;
8. **namespace** AGIS\_work.Mehtod
9. {
10. //格网生成等值线
11. **public** **class** GridCreateContourLine
12. {
13. **public** List<**double**> XAxis = **new** List<**double**>();//横线值序列
14. **public** List<**double**> YAxis = **new** List<**double**>();//竖线值序列
15. **public** **double**[,] HH = **null**; //横边追踪数组
16. **public** **double**[,] SS = **null**; //竖边追踪数组
17. **public** **int** XCount = 0;
18. **public** **int** YCount = 0;
19. **public** **double** Elevation = 0;//当前等值线值
21. **public** GridCreateContourLine(List<**double**> xAxis, List<**double**> yAxis,
22. **double**[,] hh, **double**[,] ss, **int** xCount, **int** yCount, **double** elev)
23. {
24. XAxis = xAxis;
25. YAxis = yAxis;
26. HH = hh;
27. SS = ss;
28. XCount = xCount;
29. YCount = yCount;
30. Elevation = elev;
32. }
34. //           2
35. //       \_\_\_\_\_\_\_\_\_\_\_
36. //      |           |
37. //      |           |
38. //    3 |           |  5
39. //      |           |
40. //      |\_\_\_\_\_\_\_\_\_\_\_|
41. //  (i,j)     7
42. //
43. //生成所有等值线
44. **public** List<ContourPolyline> CreateContourLines()
45. {
46. List<ContourPolyline> tempPolylineLsit = **new** List<ContourPolyline>();
47. **for** (**int** i = 0; i < XCount; i++)
48. {
49. **for** (**int** j = 0; j <= YCount; j++)
50. {
51. **if** (HH[i, j] < 2)
52. {
53. ContourPolyline tempPolyline = CreateContourLine(i, j, 2);
54. **if** (tempPolyline != **null**)
55. tempPolylineLsit.Add(tempPolyline);
56. tempPolyline = CreateContourLine(i, j, 7);
57. **if** (tempPolyline != **null**)
58. tempPolylineLsit.Add(tempPolyline);
59. }
60. }
61. }
62. **for** (**int** i = 0; i <= XCount; i++)
63. {
64. **for** (**int** j = 0; j < YCount; j++)
65. {
66. **if** (SS[i, j] < 2)
67. {
68. ContourPolyline tempPolyline = CreateContourLine(i, j, 3);
69. **if** (tempPolyline != **null**)
70. tempPolylineLsit.Add(tempPolyline);
71. tempPolyline = CreateContourLine(i, j, 5);
72. **if** (tempPolyline != **null**)
73. tempPolylineLsit.Add(tempPolyline);
74. }
75. }
76. }
77. **return** tempPolylineLsit;
78. }
79. //生成一条等值线
80. **private** ContourPolyline CreateContourLine(**int** ii, **int** jj, **int** direct)
81. {
82. List<DataPoint> tempDataPoints = **new** List<DataPoint>();
83. **int**[] res = **new** **int**[3] { ii, jj, direct };
84. **while** (res != **null**)
85. {
86. res = Track(res[0], res[1], res[2]);
87. **if** (res != **null**)
88. {
89. **switch** (res[2])
90. {
91. **case** 2:
92. tempDataPoints.Add(**new** DataPoint(-res[0] \* 100 - res[1] - 1, "等值点" + (-res[0] \* 100 - res[1] - 1).ToString(),
93. XAxis[res[0]] + HH[res[0], res[1] + 1] \* (XAxis[res[0] + 1] - XAxis[res[0]]),
94. YAxis[res[1] + 1], Elevation/\*, -res[0] \* 100 - res[1] - 1\*/));
95. HH[res[0], res[1] + 1] = 5;
96. **break**;
97. **case** 3:
98. tempDataPoints.Add(**new** DataPoint(res[0] \* 100 + res[1],
99. "等值点" + (res[0] \* 100 + res[1]).ToString(), XAxis[res[0]],
100. YAxis[res[1]] + SS[res[0], res[1]] \* (YAxis[res[1] + 1] - YAxis[res[1]]),
101. Elevation/\*, res[0] \* 100 + res[1]\*/));
102. SS[res[0], res[1]] = 5;
103. **break**;
104. **case** 5:
105. SS[res[0] + 1, res[1]] = 5;
106. tempDataPoints.Add(**new** DataPoint((res[0] + 1) \* 100 + res[1],
107. "等值点" + ((1 + res[0]) \* 100 + res[1]).ToString(), XAxis[res[0] + 1],
108. YAxis[res[1]] + SS[res[0] + 1, res[1]] \* (YAxis[res[1] + 1] - YAxis[res[1]]),
109. Elevation/\*, (res[0] + 1) \* 100 + res[1]\*/));
110. **break**;
111. **case** 7:
112. tempDataPoints.Add(**new** DataPoint(-res[0] \* 100 - res[1], "等值点" + (-res[0] \* 100 - res[1]).ToString(),
113. XAxis[res[0]] + HH[res[0], res[1]] \* (XAxis[res[0] + 1] - XAxis[res[0]]),
114. XAxis[res[1]], Elevation/\*, -res[0] \* 100 - res[1]\*/));
115. HH[res[0], res[1]] = 5;
116. **break**;
117. **default**:
118. **break**;
119. }
120. }
121. }
122. **if** (tempDataPoints.Count > 0) { **return** **new** ContourPolyline(tempDataPoints.ToArray()); }
123. **else** **return** **null**;
124. }
125. //追踪
126. **private** **int**[] Track(**int** i, **int** j, **int** inDirc)
127. {
128. **if** (i < 0 || j < 0 || i >= XCount - 1 || j >= YCount - 1)
129. { **return** **null**; }
130. **switch** (inDirc)
131. {
132. **case** 2:
133. **if** (SS[i, j] < 2)
134. **return** **new** **int**[3] { i - 1, j, 5 };
135. **else** **if** (SS[i, j + 1] < 2)
136. **return** **new** **int**[3] { i + 1, j, 3 };
137. **else** **if** (HH[i, j] < 2)
138. **return** **new** **int**[3] { i, j - 1, 2 };
139. **break**;
140. **case** 3:
141. **if** (HH[i, j] < 2)
142. **return** **new** **int**[3] { i, j - 1, 2 };
143. **else** **if** (HH[i, j + 1] < 2)
144. **return** **new** **int**[3] { i, j + 1, 7 };
145. **else** **if** (SS[i, j + 1] < 2)
146. **return** **new** **int**[3] { i + 1, j, 3 };
147. **break**;
148. **case** 5:
149. **if** (HH[i, j + 1] < 2)
150. **return** **new** **int**[3] { i, j + 1, 7 };
151. **else** **if** (HH[i, j] < 2)
152. **return** **new** **int**[3] { i, j - 1, 2 };
153. **else** **if** (SS[i, j] < 2)
154. **return** **new** **int**[3] { i - 1, j, 5 };
155. **break**;
156. **case** 7:
157. **if** (SS[i, j + 1] < 2)
158. **return** **new** **int**[3] { i + 1, j, 3 };
159. **else** **if** (SS[i, j] < 2)
160. **return** **new** **int**[3] { i - 1, j, 5 };
161. **else** **if** (HH[i, j + 1] < 2)
162. **return** **new** **int**[3] { i, j + 1, 7 };
163. **break**;
164. **default**:
165. **break**;
166. }
167. **return** **null**;
168. }
169. }
170. }

# GridInterpolation.cs

1. **using** AGIS\_work.DataStructure;
2. **using** System;
3. **using** System.Collections.Generic;
4. **using** System.Linq;
5. **using** System.Text;
6. **using** System.Threading.Tasks;
8. **namespace** AGIS\_work.Mehtod
9. {
10. //插值类型
11. **public** **enum** GridInterpolationMehtod
12. {
13. None = 0,
14. 距离平方倒数法 = 1,
15. 按方位加权平均法 = 2
16. }
17. //插值算法
18. **public** **class** GridInterpolation
19. {
20. **public** PointSet mPointSet { **get**; **private** **set**; }
21. **public** GridInterpolation(PointSet pointSet)
22. { **this**.mPointSet = pointSet; }
23. //距离平方倒数法
24. **public** **double** CalculateValueBy距离平方倒数法(**double** x, **double** y, **int** pts)
25. {
26. List<Tuple<DataPoint, **double**>> PointAndDistanceList = **new** List<Tuple<DataPoint, **double**>>();
27. **foreach** (var point **in** mPointSet.PointList)
28. { PointAndDistanceList.Add(**new** Tuple<DataPoint, **double**>(point, point.GetDistance(x, y))); }
29. PointAndDistanceList.Sort((t1, t2) => t1.Item2.CompareTo(t2.Item2));
30. **double** sDenominator = 0; //分母
31. **double** sNumerator = 0;  //分子
32. **for** (**int** i = 0; i < pts; i++)
33. {
34. sDenominator += 1 / PointAndDistanceList[i].Item2;
35. sNumerator += (PointAndDistanceList[i].Item1.Value) / PointAndDistanceList[i].Item2;
36. }
37. **return** sNumerator / sDenominator;
38. }
39. //按方位加权平均法
40. **public** **double** CalculateValueBy按方位加权平均法(**double** x, **double** y, **int** sectorNums)
41. {
42. List<Tuple<DataPoint, **double**>>[] PointPositionDistanceList
43. = **new** List<Tuple<DataPoint, **double**>>[sectorNums];
44. **for** (**int** i = 0; i < sectorNums; i++)
45. PointPositionDistanceList[i] = **new** List<Tuple<DataPoint, **double**>>();
46. **double** sectorArc = 360.0 / sectorNums;
47. **foreach** (var point **in** mPointSet.PointList)
48. {
49. **double** alpha = point.GetPosition(x, y);
50. PointPositionDistanceList[(**int**)(alpha / sectorArc)].Add(**new** Tuple<DataPoint, **double**>(point, point.GetDistanceP2(x, y)));
51. }
52. List<Tuple<DataPoint, **double**>> SelectedPointList = **new** List<Tuple<DataPoint, **double**>>();
53. **for** (**int** i = 0; i < sectorNums; i++)
54. {
55. **if** (PointPositionDistanceList[i].Count != 0)
56. {
57. PointPositionDistanceList[i].Sort((t1, t2) => t1.Item2.CompareTo(t2.Item2));
58. SelectedPointList.Add(PointPositionDistanceList[i][0]);
59. }
60. }
61. List<**double**> WeightList = **new** List<**double**>();
62. **int** SelectPointCount = SelectedPointList.Count;
63. **if** (SelectPointCount != 0)
64. {
65. **double** sProduct = 1; //总的乘积
66. **double** sDenominator = 0; //分母
67. **double** sResult = 0; //结果
68. **for** (**int** j = 0; j < SelectPointCount; j++)
69. sProduct \*= SelectedPointList[j].Item2;
70. **for** (**int** j = 0; j < SelectPointCount; j++)
71. {
72. WeightList.Add(sProduct / SelectedPointList[j].Item2);
73. sDenominator += sProduct / SelectedPointList[j].Item2;
74. }
75. **for** (**int** j = 0; j < SelectPointCount; j++)
76. sResult += WeightList[j] \* SelectedPointList[j].Item1.Value / sDenominator;
77. **return** sResult;
78. }
79. **else** **throw** **new** Exception("CalculateValueBy按方位加权平均法:没有选中的点");
80. }
81. }
82. }

# AgisControl

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.ComponentModel;
4. **using** System.Drawing;
5. **using** System.Data;
6. **using** System.Linq;
7. **using** System.Text;
8. **using** System.Threading.Tasks;
9. **using** System.Windows.Forms;
10. **using** AGIS\_work.DataStructure;
11. **using** System.Drawing.Drawing2D;
12. **using** AGIS\_work.Mehtod;
14. **namespace** AGIS\_work
15. {
16. //用户操作类型
17. **public** **enum** UserOperationType
18. {
19. None = 0,
20. DisplayThePointSet = 1,
21. DisplayInGrid = 2,
22. DisplayInTIN = 3,
23. GenerateContourByGrid = 4,
24. GenerateContourByTin = 5,
25. GenerateTopology
27. }
28. //用户操作控件
29. **public** partial **class** AgisControl : UserControl
30. {
31. **public** AgisControl()
32. {
33. InitializeComponent();
34. ZoomScale = 1;
35. CenterPoint = **new** PointF(0, 0);
36. **this**.MouseWheel += **this**.AgisControl\_MouseWheel;
37. }
39. **public** PointSet PointSet { **get**; **private** **set**; }
40. **public** UserOperationType UserOperation { **get**; **private** **set**; }
42. **public** MinBoundRect MBR\_Origin { **get**; **private** **set**; }
43. **public** PointF CenterPoint { **get**; **private** **set**; }
44. **public** **double** ZoomScale { **get**; **private**  **set**; }   //  Screen / RawData
45. **public** Brush PointBrush = **new** SolidBrush(Color.Indigo);
46. **public** **double** PointRadius = 3;
47. **public** **double** FrameScaling = 1.2;
48. **public** **double** Zoom = 1;
49. **public** PointF CurMouseLocation;
50. **public** **bool** IsPanning = **false**;
51. **public** **double** OffsetX = 0;
52. **public** **double** OffsetY = 0;
54. //格网差值
55. **public** **int** 距离平方倒数法NearPts = -1;
56. **public** **int** 按方位加权平均法SectorNum = -1;
57. **public** GridInterpolationMehtod GridIntMethod = GridInterpolationMehtod.None;
59. **public** **bool** LoadPointSet(PointSet pointset, **double** frameScaling = 1.2)
60. {
61. **try**
62. {
63. **this**.PointSet = pointset;
64. **this**.FrameScaling = frameScaling;
65. //重绘
66. MinBoundRect pointMBR = pointset.MBR;
67. CenterPoint = **new** PointF((**float**)(pointMBR.MaxX + pointMBR.MinX) / 2,
68. (**float**)(pointMBR.MaxY + pointMBR.MinY) / 2);
69. **double** pointSetWidth = pointMBR.MaxX - pointMBR.MinX;
70. **double** pointSetHeight = pointMBR.MaxY - pointMBR.MinY;
71. **this**.ZoomScale = Math.Min(**this**.Height / (pointSetHeight),
72. **this**.Width / (pointSetWidth)) / frameScaling;
73. MBR\_Origin = **new** MinBoundRect(CenterPoint.X - pointSetWidth \* frameScaling / 2,
74. CenterPoint.Y - pointSetHeight \* frameScaling / 2,
75. CenterPoint.X + pointSetWidth \* frameScaling / 2,
76. CenterPoint.Y + pointSetHeight \* frameScaling / 2);
77. OffsetX = MBR\_Origin.MinX;
78. OffsetY = MBR\_Origin.MinY;
79. Zoom = ZoomScale;
80. UserOperation = UserOperationType.DisplayThePointSet;
81. **return** **true**;
82. }
83. **catch** {**return** **false**;}
84. }
86. **private** **void** AgisControl\_Load(**object** sender, EventArgs e){}
88. **private** **void** AgisControl\_Paint(**object** sender, PaintEventArgs e){}
90. **public** **void** SetUserOperationToDisplayInGrid()
91. {**this**.UserOperation = UserOperationType.DisplayInGrid;}
93. /// <summary>
94. /// 获取实际坐标点在屏幕上的位置
95. /// </summary>
96. /// <param name="x"></param>
97. /// <param name="y"></param>
98. /// <returns></returns>
99. **public** PointF GetScreenLocation(**double** x,**double** y)
100. {**return** **new** PointF((**float**)((x - **this**.OffsetX) \* **this**.Zoom),
101. (**float**)(**this**.Height - ((y - **this**.OffsetY) \* **this**.Zoom)));}
103. /// <summary>
104. /// 单独获取实际坐标点X轴在屏幕上的位置
105. /// </summary>
106. /// <param name="x"></param>
107. /// <returns></returns>
108. **public** **double** GetScreenLocX(**double** x)
109. {**return** (x - **this**.OffsetX) \* **this**.Zoom;}
110. /// <summary>
111. /// 单独获取实际坐标点Y轴在屏幕上的位置
112. /// </summary>
113. /// <param name="y"></param>
114. /// <returns></returns>
115. **public** **double** GetScreenLocY(**double** y)
116. {**return** (**this**.Height - ((y - **this**.OffsetY) \* **this**.Zoom));}
118. /// <summary>
119. /// 获取实际边在屏幕上的投影
120. /// </summary>
121. /// <param name="edge"></param>
122. /// <returns></returns>
123. **public** PointF[] GetScreenEdge(Edge edge)
124. {
125. PointF startP = **new** PointF((**float**)GetScreenLocX(edge.StartPoint.X), (**float**)GetScreenLocY(edge.StartPoint.Y));
126. PointF endP = **new** PointF((**float**)GetScreenLocX(edge.EndPoint.X), (**float**)GetScreenLocY(edge.EndPoint.Y));
127. **return** **new** PointF[] { startP,endP };
128. }
130. /// <summary>
131. /// 获取实际折线在屏幕上的投影
132. /// </summary>
133. /// <param name="polyline"></param>
134. /// <returns></returns>
135. **public** PointF[] GetScreenEdge(ContourPolyline polyline)
136. {
137. List<PointF> tempPointList = **new** List<PointF>();
138. **foreach** (var point **in** polyline.PointList)
139. {tempPointList.Add(**new** PointF((**float**)GetScreenLocX(point.X), (**float**)GetScreenLocY(point.Y)));}
140. **return** tempPointList.ToArray();
141. }
143. **public** PointF GetScreenPoint(TopoPoint point)
144. {
145. **return** **new** PointF((**float**)((point.X - **this**.OffsetX) \* **this**.Zoom),
146. (**float**)(**this**.Height - ((point.Y - **this**.OffsetY) \* **this**.Zoom)));
147. }
149. **public** PointF[] GetScreenPoints(TopoPoint[] points)
150. {
151. List<PointF> result = **new** List<PointF>();
152. **foreach** (var point **in** points)
153. {result.Add(**new** PointF((**float**)((point.X - **this**.OffsetX) \* **this**.Zoom),
154. (**float**)(**this**.Height - ((point.Y - **this**.OffsetY) \* **this**.Zoom))));}
155. **return** result.ToArray();
156. }
158. **public** PointF[] GetScreenLine(TopoPolyline line)
159. {
160. List<PointF> tempPointList = **new** List<PointF>();
161. List<TopoPoint> pointList = **new** List<TopoPoint>();
162. pointList.Add(line.BeginNode);
163. pointList.AddRange(line.MiddlePoint);
164. pointList.Add(line.EndNode);
165. **foreach** (var point **in** pointList)
166. tempPointList.Add(**this**.GetScreenPoint(point));
167. **return** tempPointList.ToArray();
168. }
170. **public** **double** GetRegionArea()
171. { **return** (**this**.MBR\_Origin.MaxX - **this**.MBR\_Origin.MinX) \*
172. (**this**.MBR\_Origin.MaxY - **this**.MBR\_Origin.MinY);}
174. /// <summary>
175. /// 获取屏幕点的实际位置。
176. /// </summary>
177. /// <param name="x"></param>
178. /// <param name="y"></param>
179. /// <returns></returns>
180. **public** **double**[] GetRealWorldLocation(**float** x , **float** y)
181. { **return** **new** **double**[] { (x / **this**.Zoom + **this**.OffsetX),
182. ((**this**.Height - y) / **this**.Zoom + **this**.OffsetY) };}
184. **public** **double** GetRealWorldLocX(**float** x)
185. {**return** x / **this**.Zoom + **this**.OffsetX; }
187. **public** **double** GetRealWorldLocY(**float** y)
188. {**return** (**this**.Height - y) / **this**.Zoom + **this**.OffsetY;}
190. **private** **void** AgisControl\_Resize(**object** sender, EventArgs e)
191. {**this**.Refresh(); }
193. **private** **void** AgisControl\_MouseClick(**object** sender, MouseEventArgs e)
194. {
195. **if** (e.Button == MouseButtons.Middle)
196. {
197. OffsetX = MBR\_Origin.MinX;
198. OffsetY = MBR\_Origin.MinY;
199. **this**.ZoomScale = Math.Min(**this**.Height / (**this**.PointSet.MBR.MaxY - **this**.PointSet.MBR.MinY),
200. **this**.Width / (**this**.PointSet.MBR.MaxX - **this**.PointSet.MBR.MinX)) / **this**.FrameScaling;
201. **this**.Zoom = **this**.ZoomScale;
202. }
203. **this**.Refresh();
204. }
206. **private** **void** AgisControl\_MouseWheel(**object** sender, MouseEventArgs e)
207. {
208. PointF mouseLoc = e.Location;
209. **double**[] curLoc = **this**.GetRealWorldLocation(mouseLoc.X, mouseLoc.Y);
210. **if** (e.Delta > 0)
211. {
212. OffsetX = curLoc[0] - (curLoc[0] - OffsetX) \* 0.9;
213. OffsetY = curLoc[1] - (curLoc[1] - OffsetY) \* 0.9;
214. Zoom /= 0.9;
215. }
216. **else**
217. {
218. OffsetX = curLoc[0] - (curLoc[0] - OffsetX) / 0.9;
219. OffsetY = curLoc[1] - (curLoc[1] - OffsetY) / 0.9;
220. Zoom \*= 0.9;
221. }
222. **this**.Refresh();
223. }
225. **private** **void** AgisControl\_MouseMove(**object** sender, MouseEventArgs e)
226. {
227. **if** (IsPanning == **true**)
228. {
229. PointF mouseLoc = e.Location;
230. **double**[] curLoc = **this**.GetRealWorldLocation(mouseLoc.X, mouseLoc.Y);
231. **double**[] lastLoc = **this**.GetRealWorldLocation(**this**.CurMouseLocation.X, **this**.CurMouseLocation.Y);
232. **this**.OffsetX += -curLoc[0] + lastLoc[0];
233. **this**.OffsetY += -curLoc[1] + lastLoc[1];
234. **this**.Refresh();
235. **this**.CurMouseLocation = mouseLoc;
236. }
237. }
239. **private** **void** AgisControl\_MouseDown(**object** sender, MouseEventArgs e)
240. {**this**.IsPanning = **true**;
241. **this**.CurMouseLocation = e.Location;}
243. **private** **void** AgisControl\_MouseUp(**object** sender, MouseEventArgs e)
244. {**this**.IsPanning = **false**;
245. **this**.CurMouseLocation = e.Location; }
247. **public** **double** GetGridInterpolationValue(**double** x,**double** y)
248. {
249. GridInterpolation method = **new** GridInterpolation(**this**.PointSet);
250. **switch** (**this**.GridIntMethod)
251. {
252. **case** GridInterpolationMehtod.None:
253. **throw** **new** Exception("未选择插值方法");
254. **case** GridInterpolationMehtod.距离平方倒数法:
255. **return** method.CalculateValueBy距离平方倒数法(x, y, 距离平方倒数法NearPts);
256. **case** GridInterpolationMehtod.按方位加权平均法:
257. **return** method.CalculateValueBy按方位加权平均法(x, y, 按方位加权平均法SectorNum);
258. **default**:
259. **throw** **new** Exception("未选择插值方法");
260. }
261. }
262. }
263. }

# MainForm.cs

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.ComponentModel;
4. **using** System.Data;
5. **using** System.Drawing;
6. **using** System.Linq;
7. **using** System.Text;
8. **using** System.Threading.Tasks;
9. **using** System.Windows.Forms;
10. **using** AGIS\_work.Forms.File;
11. **using** AGIS\_work.DataStructure;
12. **using** AGIS\_work.Forms.Grid;
13. **using** AGIS\_work.Mehtod;
14. **using** AGIS\_work.Forms.ContourLine;
15. **using** AGIS\_work.Forms.Topology;
16. **using** System.Threading;
18. **namespace** AGIS\_work
19. {
20. **public** partial **class** MainForm : Form
21. {
22. **public** MainForm()
23. {
24. InitializeComponent();
25. **this**.agisControl.MouseWheel += **this**.agisControl\_MouseWheel;
26. }
27. **public** PointSet mPointSet;
29. **public** UserOperationType UserOperation;
31. // -- 数据点
32. **private** **float** PointHalfWidth = 5;
33. **public** Brush PointIconBrush = **new** SolidBrush(Color.Red);
35. // ----格网相关
36. **public** **bool** IsGridVisible = **false**;
37. **public** **int** GridDivisionCount\_X = 0;
38. **public** **int** GridDivisionCount\_Y = 0;
39. **public** **int** EachGridDivisionCount\_X = 1;
40. **public** **int** EachGridDivisionCount\_Y = 1;
41. **public** **float** GridLineWidth = 2.0f;
42. **public** **float** GridSubLineWidth = 1.0f;
43. **public** Pen GridLinePen = **new** Pen(Color.Black, 2.0f);
44. **public** Pen GridSubLinePen = **new** Pen(Color.Black, 1.0f);
45. **public** **bool** IsQueryIntersection = **false**;
46. **public** List<**double**> Grid\_AxisX = **new** List<**double**>();
47. **public** List<**double**> Grid\_AxisY = **new** List<**double**>();
48. **public** List<**double**> GridScreen\_AxisX = **new** List<**double**>();
49. **public** List<**double**> GridScreen\_AxisY = **new** List<**double**>();
51. // -- 格网选中交点
52. **public** **int** SelectPixelThreshold = 9;
53. **public** PointF MouseLocation;
54. **public** Pen GridSelectedPointPen = **new** Pen(Color.Cyan, 3.0f);
55. **public** **double** SelectPointX = -1;
56. **public** **double** SelectPointY = -1;
58. // -- 格网等高线
59. **public** Edge[] GridContourList = **null**;
60. **public** ContourPolyline[] GridContourPolylineList = **null**;
61. **public** Pen GridContourLinePen = **new** Pen(Color.Brown, 1.5f);
62. **public** **double**[,] GridValueMatrix = **null**;
63. **public** **double**[,] SS = **null**;
64. **public** **double**[,] HH = **null**;
65. **private** **bool** ContourLineUseSpline = **false**;
67. // -- Tin相关
68. **public** **bool** ShowTin = **false**;
69. **public** Edge[] TinEdges = **null**;
70. **public** Pen TinPen = **new** Pen(Color.Blue, 1.0f);
72. // -- Tin等高线相关
73. **public** **int** ContourLineType = 0; //0:不显示，1：根据格网，2：根据Tin
74. **public** **bool** ShowContourLine = **true**;
75. **public** Edge[] TinContourLineList = **null**;
76. **public** Pen TinContourLinePen = **new** Pen(Color.Gray, 1.5f);
78. // -- 拓扑关系相关
79. **private** List<ContourPolyline> mSubPolyline;
80. **private** List<Edge> mSubEdge;
81. **public** **bool** ShowTopology = **false**;
82. **public** Brush TopologyNodeBrush = **new** SolidBrush(Color.Blue);
83. **public** Brush TopologyPointBrush = **new** SolidBrush(Color.Green);
84. **public** **int** TopologyPixelHalfWidth = 3;
85. **private** Pen TopolopyLinePen = **new** Pen(Color.Green, 1.5f);
87. // -- 拓扑表
88. **private** TopoPolygonSet mTopoPolygonSet;
89. **private** TopoPolylineSet mTopoPolylineSet;
90. **private** TopoPointSet mTopoPointSet;
92. // -- 拓扑交互
93. **private** **bool** ShowTopoPoint = **false**;
94. **private** **bool** ShowTopoPolyline = **false**;
95. **private** **bool** ShowTopoPolygon = **false**;
96. **private** **bool** IsQueryTopoPolygon = **false**;
97. **private** TopoPolygon SelectedTopoPolygon;
99. **private** **void** MainForm\_Load(**object** sender, EventArgs e)
100. {
101. GridLinePen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
102. GridSubLinePen.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDotDot;
103. mTopoPolygonSet = **new** TopoPolygonSet();
104. mTopoPolylineSet = **new** TopoPolylineSet();
105. mTopoPointSet = **new** TopoPointSet();
106. }
108. **private** **void** 打开ToolStripMenuItem1\_Click(**object** sender, EventArgs e)
109. {
110. OpenFileForm openFile = **new** OpenFileForm();
111. **if** (openFile.ShowDialog() == DialogResult.OK)
112. {
113. mPointSet = PointSet.ReadFromCSV(openFile.PointSetFileName);
114. **this**.Width = 1000;
115. **this**.Height = 800;
116. **this**.UserOperation = UserOperationType.DisplayThePointSet;
117. agisControl.LoadPointSet(mPointSet, 1.2);
118. agisControl.Refresh();
119. }
120. **return**;
121. }
123. **private** **void** agisControl\_Resize(**object** sender, EventArgs e)
124. {agisControl.Refresh();}
126. **private** **void** agisControl\_MarginChanged(**object** sender, EventArgs e)
127. {agisControl.Refresh();}
129. **private** **void** agisControl\_Paint(**object** sender, PaintEventArgs e)
130. {
132. //画一些基础的图形
133. **if** (**this**.UserOperation != UserOperationType.None) { }
134. //绘制拓扑数据
135. **if** (**this**.ShowTopology == **true**)
136. {
137. // -- 绘制多边形
138. **if** (**this**.ShowTopoPolygon == **true**)
139. {
140. **foreach** (var polygon **in** **this**.mTopoPolygonSet.TopoPolygonList)
141. {
142. TopoPoint[] tempLines = polygon.ConvertToPointArray();
143. Graphics g = e.Graphics;
144. PointF[] pf = agisControl.GetScreenPoints(tempLines);
145. Brush randomBrush = **new** SolidBrush(**this**.GetRandomColor());
146. g.FillPolygon(randomBrush, pf);
147. //Thread.Sleep(1000);
148. }
149. }
150. // -- 绘制折线
151. **if** (**this**.ShowTopoPolyline == **true**)
152. {
153. **foreach** (var line **in** **this**.mTopoPolylineSet.TopoPolylineList)
154. {
155. Graphics g = e.Graphics;
156. PointF[] pf = agisControl.GetScreenLine(line);
157. g.DrawLines(**this**.TopolopyLinePen, pf);
158. }
159. **if** (SelectedTopoPolygon != **null** && **this**.IsQueryTopoPolygon == **true**)
160. {TopoPoint[] tempLines = SelectedTopoPolygon.ConvertToPointArray();
161. Graphics g = e.Graphics;
162. PointF[] pf = agisControl.GetScreenPoints(tempLines);
163. g.DrawLines(**this**.GridSelectedPointPen, pf);}
164. }
165. **if** (**this**.ShowTopoPoint == **true**)
166. {
167. // -- 绘制中间点
168. **foreach** (var point **in** **this**.mTopoPointSet.TopoPointList)
169. {Graphics g = e.Graphics;
170. PointF pf = agisControl.GetScreenPoint(point);
171. g.FillRectangle(TopologyPointBrush, pf.X - **this**.TopologyPixelHalfWidth, pf.Y - TopologyPixelHalfWidth,
172. TopologyPixelHalfWidth \* 2, TopologyPixelHalfWidth \* 2); }
173. // -- 绘制结点
174. **foreach** (var point **in** **this**.mTopoPointSet.TopoNodeList)
175. {Graphics g = e.Graphics;
176. PointF pf = agisControl.GetScreenPoint(point);
177. g.FillRectangle(TopologyNodeBrush, pf.X - **this**.TopologyPixelHalfWidth, pf.Y - TopologyPixelHalfWidth,
178. TopologyPixelHalfWidth \* 2, TopologyPixelHalfWidth \* 2);}
179. }
180. }
181. //在网格中
182. **if** (**this**.UserOperation == UserOperationType.DisplayInGrid)
183. {
184. //格网可见，且XY方向等分数不为0
185. **if** (IsGridVisible != **false** && GridDivisionCount\_X != 0 && GridDivisionCount\_Y != 0)
186. {
187. Graphics g = e.Graphics;
188. PointF MinPointXY = **this**.agisControl.GetScreenLocation(agisControl.MBR\_Origin.MinX, agisControl.MBR\_Origin.MinY);
189. PointF MaxPointXY = **this**.agisControl.GetScreenLocation(agisControl.MBR\_Origin.MaxX, agisControl.MBR\_Origin.MaxY);
190. **float** width = MaxPointXY.X - MinPointXY.X;
191. **float** height = MaxPointXY.Y - MinPointXY.Y;
192. //g.DrawLine(new Pen(Color.Green), MinPointXY, MaxPointXY);
193. **for** (**int** i = 0; i < GridDivisionCount\_X; i++)
194. {
195. g.DrawLine(**this**.GridLinePen, MinPointXY.X + i \* (width / GridDivisionCount\_X), MinPointXY.Y,
196. MinPointXY.X + i \* (width / GridDivisionCount\_X), MaxPointXY.Y);
197. **for** (**int** ii = 1; ii < EachGridDivisionCount\_X; ii++)
198. {g.DrawLine(**this**.GridSubLinePen, MinPointXY.X + (i + ii \* 1.0f / EachGridDivisionCount\_X) \* (width / GridDivisionCount\_X), MinPointXY.Y,
199. MinPointXY.X + (i + ii \* 1.0f / EachGridDivisionCount\_X) \* (width / GridDivisionCount\_X), MaxPointXY.Y);}
200. }
201. g.DrawLine(**this**.GridLinePen, MinPointXY.X + width, MinPointXY.Y, MinPointXY.X + width, MaxPointXY.Y);
202. **for** (**int** j = 0; j < GridDivisionCount\_Y; j++)
203. {
204. g.DrawLine(**this**.GridLinePen, MinPointXY.X, MinPointXY.Y + j \* (height / GridDivisionCount\_Y),
205. MaxPointXY.X, MinPointXY.Y + j \* (height / GridDivisionCount\_Y));
206. **for** (**int** jj = 0; jj < EachGridDivisionCount\_Y; jj++)
207. { g.DrawLine(**this**.GridSubLinePen, MinPointXY.X, MinPointXY.Y + (j + jj \* 1.0f / EachGridDivisionCount\_Y) \* (height / GridDivisionCount\_Y),
208. MaxPointXY.X, MinPointXY.Y + (j + jj \* 1.0f / EachGridDivisionCount\_Y) \* (height / GridDivisionCount\_Y));}
209. }
210. g.DrawLine(**this**.GridLinePen, MinPointXY.X, MinPointXY.Y + height, MaxPointXY.X, MinPointXY.Y + height);
211. **if** (**this**.IsQueryIntersection == **true** && SelectPointX != 0 && SelectPointY != 0)
212. {
213. **double** sScreenSelectPointX = **this**.agisControl.GetScreenLocX(SelectPointX);
214. **double** sScreenSelectPointY = **this**.agisControl.GetScreenLocY(SelectPointY);
215. g.DrawEllipse(**this**.GridSelectedPointPen, (**float**)sScreenSelectPointX - SelectPixelThreshold,
216. (**float**)sScreenSelectPointY - SelectPixelThreshold,
217. SelectPixelThreshold \* 2, SelectPixelThreshold \* 2);
218. }
219. }
220. //绘制等值线
221. **if** (ShowContourLine == **true** && GridContourList != **null**)
222. {
223. **for** (**int** i = 0; i < GridContourList.Length; i++)
224. { PointF[] screenLine = agisControl.GetScreenEdge(GridContourList[i]);
225. Graphics g = e.Graphics;
226. g.DrawLine(GridContourLinePen, screenLine[0], screenLine[1]);}
227. }
228. **if** (ShowContourLine == **true** && GridContourPolylineList != **null**)
229. {
230. **for** (**int** i = 0; i < GridContourPolylineList.Length; i++)
231. {
232. PointF[] screenLine = agisControl.GetScreenEdge(GridContourPolylineList[i]);
233. Graphics g = e.Graphics;
234. **float** tension = 0f;
235. **if** (ContourLineUseSpline)
236. tension = 0.25f /\* (float)(agisControl.ZoomScale / agisControl.Zoom)\*/;
237. **if** (screenLine.Length > 1)
238. g.DrawCurve(GridContourLinePen, screenLine, tension);
239. }
240. }
241. }
242. **if** (**this**.UserOperation == UserOperationType.DisplayInTIN)
243. {
244. //绘制三角网
245. **if** (ShowTin == **true** && TinEdges != **null**)
246. {**for** (**int** i = 0; i < TinEdges.Length; i++)
247. {PointF[] screenLine = agisControl.GetScreenEdge(TinEdges[i]);
248. Graphics g = e.Graphics;
249. g.DrawLine(TinPen, screenLine[0], screenLine[1]);} }
250. //绘制等高线
251. **if** (ShowContourLine == **true** && TinContourLineList != **null**)
252. {
253. **for** (**int** i = 0; i < TinContourLineList.Length; i++)
254. {PointF[] screenLine = agisControl.GetScreenEdge(TinContourLineList[i]);
255. Graphics g = e.Graphics;
256. g.DrawLine(TinContourLinePen, screenLine[0], screenLine[1]);}
257. }
258. }
260. //绘制数据点
261. **if** (mPointSet != **null**)
262. {
263. **foreach** (var point **in** mPointSet.PointList)
264. {Graphics g = e.Graphics;
265. g.FillEllipse(PointIconBrush, (**float**)agisControl.GetScreenLocX(point.X) - **this**.PointHalfWidth,
266. (**float**)agisControl.GetScreenLocY(point.Y) - PointHalfWidth, PointHalfWidth \* 2, PointHalfWidth \* 2);}
267. }
268. }
270. **private** **float** GetLineLength(PointF[] line)
271. {
272. **float** length = 0;
273. **for** (**int** i = 0; i < line.Length - 1; i++)
274. length += (**float**)Math.Sqrt(Math.Pow(line[0].X - line[1].X, 2) + Math.Pow(line[0].Y - line[1].Y, 2));
275. **return** length;
276. }
278. **public** Color GetRandomColor()
279. {
280. Random RandomNum\_First = **new** Random((**int**)DateTime.Now.Ticks);
281. System.Threading.Thread.Sleep(RandomNum\_First.Next(5));
282. Random RandomNum\_Sencond = **new** Random((**int**)DateTime.Now.Ticks);
283. //  为了在白色背景上显示，尽量生成深色
284. **int** int\_Red = RandomNum\_First.Next(256);
285. **int** int\_Green = RandomNum\_Sencond.Next(256);
286. **int** int\_Blue = (int\_Red + int\_Green > 400) ? 0 : 400 - int\_Red - int\_Green;
287. int\_Blue = (int\_Blue > 255) ? 255 : int\_Blue;
288. **return** Color.FromArgb(int\_Red, int\_Green, int\_Blue);
289. }
291. **public** Color GetRandomColor(**int** pid)
292. {
293. **int** int\_Red = Math.Abs(pid) % 256;
294. **int** int\_Green = Math.Abs(pid.GetHashCode()) % 256;
295. **int** int\_Blue = (int\_Red + int\_Green > 400) ? 0 : 400 - int\_Red - int\_Green;
296. int\_Blue = (int\_Blue > 255) ? 255 : int\_Blue;
297. **return** Color.FromArgb(int\_Red, int\_Green, int\_Blue);
298. }
300. **private** **void** agisControl\_MouseMove(**object** sender, MouseEventArgs e)
301. {
302. **switch** (**this**.UserOperation)
303. {
304. **case** UserOperationType.None:
305. **break**;
306. **default**:
307. PointF mouse = e.Location;
308. StatusLabelScreenX.Text = mouse.X.ToString("0.000");
309. StatusLabelScreenY.Text = mouse.Y.ToString("0.000");
310. **double**[] realLoc = agisControl.GetRealWorldLocation(mouse.X, mouse.Y);
311. StatusLabel\_X.Text = realLoc[0].ToString("0.000");
312. StatusLabel\_Y.Text = realLoc[1].ToString("0.000");
313. **break**;
314. }
315. **if** (**this**.UserOperation == UserOperationType.DisplayInGrid)
316. {**if** (**this**.IsGridVisible && **this**.IsQueryIntersection && **this**.agisControl.IsPanning)GridDivisionScreenRefresh();}
317. }
319. **private** **void** agisControl\_MouseWheel(**object** sender, MouseEventArgs e)
320. {
321. **if** (**this**.UserOperation == UserOperationType.DisplayInGrid)
322. {**if** (**this**.IsGridVisible && **this**.IsQueryIntersection)GridDivisionScreenRefresh();}
323. }
325. **private** **void** 距离平方倒数法ToolStripMenuItem\_Click(**object** sender, EventArgs e)
326. {
327. //this.agisControl.SetUserOperationToDisplayInGrid();
329. **if** (agisControl.PointSet == **null**) **return**;
330. **int** tempPara = agisControl.距离平方倒数法NearPts;
331. **if** (tempPara < 0) tempPara = Math.Max(agisControl.PointSet.PointList.Count / 4, 1);
332. GridIntParaForm form = **new** GridIntParaForm("取插值点邻域内最近的N个点", tempPara, 1, agisControl.PointSet.PointList.Count);
333. **if** (form.ShowDialog(**this**) == DialogResult.OK)
334. {
335. **this**.UserOperation = UserOperationType.DisplayInGrid;
336. **this**.agisControl.GridIntMethod = Mehtod.GridInterpolationMehtod.距离平方倒数法;
337. 按方位加权平均法ToolStripMenuItem.Checked = **false**;
338. 距离平方倒数法ToolStripMenuItem.Checked = **true**;
339. agisControl.距离平方倒数法NearPts = form.ParaValue;
340. MessageBox.Show("参数设置成功！", "提示");
341. }
342. }
344. **private** **void** 按方位加权平均法ToolStripMenuItem\_Click(**object** sender, EventArgs e)
345. {
346. //this.agisControl.SetUserOperationToDisplayInGrid();
347. **if** (agisControl.PointSet == **null**) **return**;
348. **int** tempPara = agisControl.按方位加权平均法SectorNum;
349. **if** (tempPara < 0)
350. tempPara = Math.Max(agisControl.PointSet.PointList.Count / 8, 1);
351. GridIntParaForm form = **new** GridIntParaForm("每个象限等分的no个扇区", tempPara, 1,
352. Math.Max(agisControl.PointSet.PointList.Count / 4, 1));
353. **if** (form.ShowDialog(**this**) == DialogResult.OK)
354. {
355. **this**.UserOperation = UserOperationType.DisplayInGrid;
356. **this**.agisControl.GridIntMethod = Mehtod.GridInterpolationMehtod.按方位加权平均法;
357. 按方位加权平均法ToolStripMenuItem.Checked = **true**;
358. 距离平方倒数法ToolStripMenuItem.Checked = **false**;
359. agisControl.按方位加权平均法SectorNum = form.ParaValue \* 4;
360. MessageBox.Show("参数设置成功！", "提示");
361. }
362. }
364. **private** **void** 加密网格toolStripMenuItem\_Click(**object** sender, EventArgs e)
365. {
366. **if** (**this**.UserOperation != UserOperationType.DisplayInGrid)
367. {MessageBox.Show("请先生成格网！", "提示");**return**;}
368. **if** (**this**.IsGridVisible == **false**)
369. {
370. **if** (MessageBox.Show(**this**, "当先设置为不显示格网，继续操作将显示格网，是否继续？", "提示", MessageBoxButtons.OKCancel)
371. != DialogResult.OK)
372. { **this**.IsGridVisible = **true**;
373. **this**.显示隐藏格网ToolStripMenuItem.Checked = **true**;}
374. **else**
375. **return**;
376. }
377. GenerateSubGridForm form = **new** GenerateSubGridForm(**this**.EachGridDivisionCount\_X, **this**.EachGridDivisionCount\_Y);
378. **if** (form.ShowDialog(**this**) == DialogResult.OK)
379. {**this**.EachGridDivisionCount\_X = form.Division\_X;
380. **this**.EachGridDivisionCount\_Y = form.Division\_Y;}
381. GridDivisionRefresh();
382. **this**.agisControl.Refresh();
383. }
385. //每次格网重新划分时进行调用
386. **private** **void** GridDivisionRefresh()
387. {
388. **int** TotalSegmentNum\_X = GridDivisionCount\_X \* EachGridDivisionCount\_X;
389. **int** TotalSegmentNum\_Y = GridDivisionCount\_Y \* EachGridDivisionCount\_Y;
390. **double** MbrMinX = agisControl.MBR\_Origin.MinX;
391. **double** MbrMaxX = agisControl.MBR\_Origin.MaxX;
392. **double** MbrMinY = agisControl.MBR\_Origin.MinY;
393. **double** MbrMaxY = agisControl.MBR\_Origin.MaxY;
394. **double** width = MbrMaxX - MbrMinX;
395. **double** height = MbrMaxY - MbrMinY;
396. Grid\_AxisX = **new** List<**double**>();
397. **for** (**int** i = 0; i <= TotalSegmentNum\_X; i++)
398. Grid\_AxisX.Add(MbrMinX + i \* width / TotalSegmentNum\_X);
399. Grid\_AxisY = **new** List<**double**>();
400. **for** (**int** i = 0; i <= TotalSegmentNum\_Y; i++)
401. Grid\_AxisY.Add(MbrMinY + i \* height / TotalSegmentNum\_Y);
402. **return**;
403. }
405. //格网重新划分或屏幕窗口平移或缩放时调用
406. **private** **void** GridDivisionScreenRefresh()
407. {
408. **int** TotalSegmentNum\_X = GridDivisionCount\_X \* EachGridDivisionCount\_X;
409. **int** TotalSegmentNum\_Y = GridDivisionCount\_Y \* EachGridDivisionCount\_Y;
410. GridScreen\_AxisX = **new** List<**double**>();
411. **for** (**int** i = 0; i <= TotalSegmentNum\_X; i++)
412. {**double** screenX = agisControl.GetScreenLocX(Grid\_AxisX[i]);
413. **if** (screenX >= 0 && screenX < agisControl.Width)
414. GridScreen\_AxisX.Add(screenX);}
415. GridScreen\_AxisY = **new** List<**double**>();
416. **for** (**int** i = 0; i <= TotalSegmentNum\_Y; i++)
417. {**double** screenY = agisControl.GetScreenLocY(Grid\_AxisY[i]);
418. **if** (screenY >= 0 && screenY < agisControl.Height)
419. GridScreen\_AxisY.Add(screenY);}
420. **return**;
421. }
423. **private** **void** 查询节点属性ToolStripMenuItem\_Click(**object** sender, EventArgs e)
424. {
425. **if** (**this**.agisControl.GridIntMethod == Mehtod.GridInterpolationMehtod.None)
426. {MessageBox.Show("尚未选择格网插值方法！\r\n请在“格网模型”中选择“距离平方倒数法”或“按方位加权平均法”！", "提示");
427. **return**;}
428. **this**.IsQueryIntersection = (**this**.IsQueryIntersection == **true**) ? **false** : **true**;
429. **this**.查询节点属性ToolStripMenuItem.Checked = **this**.IsQueryIntersection;
430. **if** (**this**.查询节点属性ToolStripMenuItem.Checked == **true**)
431. { MessageBox.Show(" ‘双击’ 进行选取格网交点", "提示");}
432. **return**;
433. }
435. **private** **void** 生成等值线ToolStripMenuItem\_Click(**object** sender, EventArgs e)
436. {
437. **if** (**this**.UserOperation != UserOperationType.DisplayInGrid)
438. {MessageBox.Show("当前并没有在格网下显示，请先生成网格！", "提示");**return**;}
439. **else** **if** (**this**.agisControl.GridIntMethod == Mehtod.GridInterpolationMehtod.None)
440. { MessageBox.Show("尚未选择格网插值方法！\r\n请在“格网模型”中选择“距离平方倒数法”或“按方位加权平均法”！", "提示");**return**;}
441. **else**
442. {
443. **this**.生成等值线ToolStripMenuItem.Checked = (**this**.生成等值线ToolStripMenuItem.Checked == **false**);
444. **this**.ShowContourLine = (**this**.生成等值线ToolStripMenuItem.Checked == **true**);
445. **if** (**this**.ShowContourLine == **false**) { **this**.agisControl.Refresh(); **return**; }
446. ContourLineSettingForm settingForm = **new** ContourLineSettingForm();
447. **if** (settingForm.ShowDialog(**this**) == DialogResult.OK)
448. {
449. //生成格网矩阵
450. List<Edge> tempGridContourLineList = **new** List<Edge>();
451. List<ContourPolyline> tempContourPolylineList = **new** List<ContourPolyline>();
452. ContourPolylineSet tempContourPolyline = **new** ContourPolylineSet();
453. //计算等值线条数
454. **int** lineCount = (**int**)((settingForm.MaxValue - settingForm.MinValue) / settingForm.IntervalValue);
455. **for** (**int** k = 0; k <= lineCount; k++)
456. {
457. **double** tempElevation = settingForm.MaxValue - k \* settingForm.IntervalValue;
458. **double**[,] GridRealLoc = GridPointPositionMatrix();
459. **double**[,] tempHH = 内插等值点\_HH(tempElevation);
460. **double**[,] tempSS = 内插等值点\_SS(tempElevation);
461. **int** Grid\_Count\_all\_X = **this**.EachGridDivisionCount\_X \* **this**.GridDivisionCount\_X;
462. **int** Grid\_Count\_all\_Y = **this**.EachGridDivisionCount\_Y \* **this**.GridDivisionCount\_Y;
463. **for** (**int** i = 0; i < Grid\_Count\_all\_X; i++)
464. {
465. **for** (**int** j = 0; j < Grid\_Count\_all\_Y; j++)
466. {
467. List<DataPoint> tempPointList = **new** List<DataPoint>();
468. //横边有等值点
469. **if** (tempHH[i, j] < 2)
470. {tempPointList.Add(**new** DataPoint(-i \* 1000 - j, "等值点" + (-i \* 1000 - j).ToString(),
471. Grid\_AxisX[i] + tempHH[i, j] \* (Grid\_AxisX[i + 1] - Grid\_AxisX[i]),
472. Grid\_AxisY[j], tempElevation, (-i \* 1000 - j) \* 1000 + (**int**)tempElevation));}
473. //竖边有等值点
474. **if** (tempSS[i, j] < 2)
475. {tempPointList.Add(**new** DataPoint(i \* 1000 + j, "等值点" + (i \* 1000 + j).ToString(),
476. Grid\_AxisX[i],
477. Grid\_AxisY[j] + tempSS[i, j] \* (Grid\_AxisY[j + 1] - Grid\_AxisY[j]),
478. tempElevation, (i \* 1000 + j) \* 1000 + (**int**)tempElevation)); }
479. //另一条横边有等值点
480. **if** (tempHH[i, j + 1] < 2)
481. {tempPointList.Add(**new** DataPoint(-i \* 1000 - j - 1, "等值点" + (-i \* 1000 - j - 1).ToString(),
482. Grid\_AxisX[i] + tempHH[i, j + 1] \* (Grid\_AxisX[i + 1] - Grid\_AxisX[i]),
483. Grid\_AxisY[j + 1], tempElevation, (-i \* 1000 - j - 1) \* 1000 + (**int**)tempElevation));}
484. //另一条竖边有等值点
485. **if** (tempSS[i + 1, j] < 2)
486. {tempPointList.Add(**new** DataPoint((i + 1) \* 1000 + j, "等值点" + ((1 + i) \* 1000 + j).ToString(),
487. Grid\_AxisX[i + 1],
488. Grid\_AxisY[j] + tempSS[i + 1, j] \* (Grid\_AxisY[j + 1] - Grid\_AxisY[j]),
489. tempElevation, ((i + 1) \* 1000 + j) \* 1000 + (**int**)tempElevation));}
490. **if** (tempPointList.Count < 2)//无等值线
491. **continue**;
492. **else** **if** (tempPointList.Count < 4)
493. {tempGridContourLineList.Add(**new** Edge(tempPointList[0], tempPointList[1]));}
494. **else**
495. {tempGridContourLineList.Add(**new** Edge(tempPointList[0], tempPointList[1]));
496. tempGridContourLineList.Add(**new** Edge(tempPointList[2], tempPointList[3]));}
497. }
498. }
499. tempContourPolyline = EdgeSet.TopologyGenerateContourPolylineSet(tempGridContourLineList.ToArray());
500. /\*另一种方法
501. GridCreateContourLine CreateContourLineClass = new GridCreateContourLine(this.Grid\_AxisX, this.Grid\_AxisY,
502. tempHH, tempSS, Grid\_Count\_all\_X, Grid\_Count\_all\_Y, tempElevation);
503. tempContourPolylineList = CreateContourLineClass.CreateContourLines();
504. \*/
505. }
506. //this.GridContourList = tempGridContourLineList.ToArray();
507. **this**.GridContourPolylineList = tempContourPolyline.ContourPolylineList.ToArray();
508. }
509. //GridContourLinePen.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDot;
510. agisControl.Refresh();
511. }
512. }
514. //生成格网点的真实坐标位置
515. **private** **double**[,] GridPointPositionMatrix()
516. {
517. List<**double**> tempGridAxisX = **new** List<**double**>();
518. List<**double**> tempGridAxisY = **new** List<**double**>();
519. tempGridAxisX.AddRange(Grid\_AxisX);
520. tempGridAxisY.AddRange(Grid\_AxisY);
521. **int** Grid\_Count\_all\_X = **this**.EachGridDivisionCount\_X \* **this**.GridDivisionCount\_X;
522. **int** Grid\_Count\_all\_Y = **this**.EachGridDivisionCount\_Y \* **this**.GridDivisionCount\_Y;
523. **double**[,] GridRealLoc = **new** **double**[Grid\_Count\_all\_X + 1, Grid\_Count\_all\_Y + 1];
524. **for** (**int** i = 0; i <= Grid\_Count\_all\_X; i++)
525. **for** (**int** j = 0; j <= Grid\_Count\_all\_Y; j++)
526. GridRealLoc[i, j] = agisControl.GetGridInterpolationValue(tempGridAxisX[i], tempGridAxisY[j]);
527. **this**.GridValueMatrix = GridRealLoc;
528. **return** GridRealLoc;
529. }
531. **private** **double**[,] 内插等值点\_HH(**double** elev)
532. {
533. **int** Grid\_Count\_all\_X = **this**.EachGridDivisionCount\_X \* **this**.GridDivisionCount\_X;
534. **int** Grid\_Count\_all\_Y = **this**.EachGridDivisionCount\_Y \* **this**.GridDivisionCount\_Y;
535. **double**[,] tempHH = **new** **double**[Grid\_Count\_all\_X, Grid\_Count\_all\_Y + 1];
536. **for** (**int** i = 0; i < Grid\_Count\_all\_X; i++)
537. {
538. **for** (**int** j = 0; j <= Grid\_Count\_all\_Y; j++)
539. {**double** r = (elev - GridValueMatrix[i, j]) / (GridValueMatrix[i + 1, j] - GridValueMatrix[i, j]);
540. tempHH[i, j] = (r <= 1 && r >= 0) ? r : 3;}
541. }
542. **this**.HH = tempHH;
543. **return** tempHH;
544. }
546. **private** **double**[,] 内插等值点\_SS(**double** elev)
547. {
548. **int** Grid\_Count\_all\_X = **this**.EachGridDivisionCount\_X \* **this**.GridDivisionCount\_X;
549. **int** Grid\_Count\_all\_Y = **this**.EachGridDivisionCount\_Y \* **this**.GridDivisionCount\_Y;
550. **double**[,] tempSS = **new** **double**[Grid\_Count\_all\_X + 1, Grid\_Count\_all\_Y];
551. **for** (**int** i = 0; i <= Grid\_Count\_all\_X; i++)
552. {
553. **for** (**int** j = 0; j < Grid\_Count\_all\_Y; j++)
554. {**double** r = (elev - GridValueMatrix[i, j]) / (GridValueMatrix[i, j + 1] - GridValueMatrix[i, j]);
555. tempSS[i, j] = (r <= 1 && r >= 0) ? r : 3;}
556. }
557. **this**.SS = tempSS;
558. **return** tempSS;
559. }
561. **private** **void** 设置ToolStripMenuItem\_Click(**object** sender, EventArgs e){}
563. **private** **void** 逐点插入法ToolStripMenuItem\_Click(**object** sender, EventArgs e)
564. {
565. //交互-格网与TIN
566. **if** (逐点插入法ToolStripMenuItem.Checked == **true**)
567. {
568. //修改显示
569. **this**.UserOperation = UserOperationType.DisplayInTIN;
570. **this**.ShowTin = **true**;
571. **this**.显示隐藏TINToolStripMenuItem.Checked = **true**;
572. CreateTIN createTin = **new** CreateTIN(**this**.mPointSet);
573. Edge[] tinEdges = createTin.PointByPointInsertion2();
574. Edge[] tinEdges2 = createTin.GeneTIN().ToArray();
575. TinEdges = tinEdges;
576. TriangleSet triSet = EdgeSet.TopologyGenerateTriangleSet(tinEdges, mPointSet);
577. Triangle[] triList = triSet.TriangleList.ToArray();
578. TinContourLinePen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
579. agisControl.Refresh();
580. }
581. **else**
582. {
583. //修改显示
584. **this**.UserOperation = UserOperationType.None;
585. **this**.ShowTin = **false**;
586. **this**.显示隐藏TINToolStripMenuItem.Checked = **false**;
587. }
589. }
591. **private** **void** 生成等值线ToolStripMenuItem1\_Click(**object** sender, EventArgs e)
592. {
593. **this**.ShowContourLine = (**this**.生成等值线ToolStripMenuItem1.Checked == **true**);
594. **if** (**this**.ShowContourLine == **false**) { **this**.agisControl.Refresh(); **return**; }
595. ContourLineSettingForm settingForm = **new** ContourLineSettingForm();
596. **if** (settingForm.ShowDialog(**this**) == DialogResult.OK)
597. {
598. //生成Tin
599. CreateTIN createTin = **new** CreateTIN(**this**.mPointSet);
600. Edge[] tinEdges = createTin.PointByPointInsertion2();
601. TinEdges = tinEdges;
602. TriangleSet triSet = EdgeSet.TopologyGenerateTriangleSet(tinEdges, mPointSet);
603. Triangle[] triList = triSet.TriangleList.ToArray();
604. List<Edge> contourLinesList = **new** List<Edge>();
605. //计算等值线条数
606. **int** lineCount = (**int**)((settingForm.MaxValue - settingForm.MinValue) / settingForm.IntervalValue);
607. **for** (**int** i = 0; i <= lineCount; i++)
608. {
609. **for** (**int** j = 0; j < triList.Length; j++)
610. {Edge contourLine = triList[j].GetContourLine(settingForm.MaxValue - i \* settingForm.IntervalValue);
611. **if** (contourLine != **null**)
612. contourLinesList.Add(contourLine);}
613. **this**.TinContourLineList = contourLinesList.ToArray();
614. }
615. }
616. TinContourLinePen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
617. agisControl.Refresh();
618. }
620. **private** **void** 设置ToolStripMenuItem1\_Click(**object** sender, EventArgs e){}
622. **private** **void** 生成拓扑关系ToolStripMenuItem\_Click(**object** sender, EventArgs e)
623. {
624. **if** (GridContourPolylineList == **null**) **return**;
625. **try**
626. {
627. **this**.GenerateTopologyRelatation(**this**.GridContourPolylineList);
628. **this**.ConvertLineEdgeToPolyline();
629. **this**.mTopoPointSet = **new** TopoPointSet(**this**.mTopoPolylineSet.TopoPolylineList.ToArray());
630. **this**.mTopoPolygonSet = **this**.mTopoPointSet.GenerateTopoPolygonSet();
631. **this**.mTopoPolygonSet.Recheck(**this**.agisControl.GetRegionArea());
632. MessageBox.Show("拓扑关系生成成功！", "生成拓扑关系");
633. }
634. **catch** (Exception err){MessageBox.Show(err.Message, "错误！");}
635. **return**;
636. }
638. **private** **void** 可视化ToolStripMenuItem\_Click(**object** sender, EventArgs e)
639. {
640. **this**.ShowTopology = (可视化ToolStripMenuItem.Checked == **true**);
641. **this**.拓扑点ToolStripMenuItem.Checked = **this**.ShowTopology;
642. **this**.拓扑边ToolStripMenuItem.Checked = **this**.ShowTopology;
643. **this**.拓扑多边形ToolStripMenuItem.Checked = **this**.ShowTopology;
644. **this**.agisControl.Refresh();
645. }
647. **private** **void** 查询ToolStripMenuItem\_Click(**object** sender, EventArgs e)
648. {
649. QueryPolygonInfoForm queryForm = **new** QueryPolygonInfoForm(**this**.mTopoPolygonSet);
650. **if** (queryForm.ShowDialog(**this**) == DialogResult.OK) {}
651. }
653. **private** **void** 导出拓扑关系表ToolStripMenuItem\_Click(**object** sender, EventArgs e)
654. {
655. SaveTopologyTableForm saveForm =
656. **new** SaveTopologyTableForm(**this**.mTopoPointSet, **this**.mTopoPolylineSet, **this**.mTopoPolygonSet);
657. **if** (saveForm.ShowDialog(**this**) == DialogResult.OK){}
658. }
660. **private** **void** 程序信息ToolStripMenuItem\_Click(**object** sender, EventArgs e)
661. {
662. MessageBox.Show(**this**, @"
663. (1)读取文件
664. “文件” —— “打开”：选取特定的文本文件，打开成功后会在界面显示数据点。
665. (2)基本操作
666. 漫游：鼠标左键拖动。
667. 放大/缩小：鼠标滚轮 上/下 滚动。
668. 全局：单击鼠标中键，缩放至原始范围。
669. (3)选择插值算法
670. “格网模型” —— “距离平方倒数法”/“按方位加权平均法”设定参数并选择该插值方法。
671. (4)生成格网模型
672. “格网模型” —— “生成格网”，选择X,Y方向分位数生成网格。
673. “格网模型” —— “加密格网”，在原有格网上加密,需要已有格网。
674. “格网模型” —— “查询格网属性”，开启/关闭查询，双击格网点，显示信息。
675. “格网模型” —— “设置” —— “显示/隐藏格网”，设置格网可见性。
676. “格网模型” —— “设置” —— “清除格网”，清除已建立的格网模型。
677. (5)TIN模型
678. “TIN模型” —— “逐点插入法”，生成TIN模型并显示。
679. “TIN模型” —— “设置” —— “显示/隐藏TIN”，设置TIN可见性。
680. “TIN模型” —— “设置” —— “清除TIN”，清除已建立的TIN模型。
681. (6)等值线
682. 等值线的最大值，最小值，间距由对话框设定。
683. “格网模型” —— “生成等值线”，根据格网模型生成等值线。
684. “格网模型” —— “生成等值线” —— “平滑”，是否平滑生成的等值线。
685. “TIN模型” —— “生成等值线”，根据TIN模型生成等值线。
686. (7)拓扑关系
687. “拓扑关系” —— “生成拓扑关系”，根据由网格生成的等值线，构建要求的拓扑关系
688. “拓扑关系” —— “可视化”，对生成的拓扑点线面进行可视化，可分别选择可视性
689. 点：结点为蓝色方格，中间点为绿色方格
690. 线：绿色线划（与等值线，格网重叠，效果不好可取消格网和等值线）
691. 面：随机颜色（每次刷新颜色不同，故刷新有延迟）
692. “拓扑关系” —— “查询”，按多边形ID，对多边形的周长和面积进行查询
693. “拓扑关系” —— “导出拓扑多边形关系表”，可分别选择要导出的数据表和路径。
694. (8)其他
695. 格网模型与TIN模型之间的切换还存在些问题，可能会在显示过程中出现奇怪的现象。
696. 如果出现问题，重启程序试试。
697. ", "程序信息", MessageBoxButtons.OK);
698. }
700. **private** **void** agisControl\_MouseHover(**object** sender, EventArgs e)
701. {}
703. **private** **void** 显示隐藏格网ToolStripMenuItem\_Click(**object** sender, EventArgs e)
704. {**this**.IsGridVisible = (显示隐藏格网ToolStripMenuItem.Checked == **true**);
705. agisControl.Refresh();}
707. **private** **void** 生成格网ToolStripMenuItem\_Click(**object** sender, EventArgs e)
708. {
709. **this**.IsGridVisible = **true**;
710. **this**.显示隐藏格网ToolStripMenuItem.Checked = **true**;
711. **this**.UserOperation = UserOperationType.DisplayInGrid;
712. GenerateGridForm form = **new** GenerateGridForm(**this**.GridDivisionCount\_X, **this**.GridDivisionCount\_Y);
713. **if** (form.ShowDialog(**this**) == DialogResult.OK)
714. {
715. **this**.GridDivisionCount\_X = form.DivisionX;
716. **this**.GridDivisionCount\_Y = form.DivisionY;
717. GridDivisionRefresh();
718. **this**.agisControl.Refresh();
719. }
720. }
722. **private** **void** agisControl\_MouseClick(**object** sender, MouseEventArgs e)
723. {MouseLocation = e.Location;}
725. **private** **void** agisControl\_MouseDown(**object** sender, MouseEventArgs e){}
727. **private** **void** agisControl\_MouseDoubleClick(**object** sender, MouseEventArgs e)
728. {
729. MouseLocation = e.Location;
730. GridDivisionScreenRefresh();
731. **if** (**this**.UserOperation != UserOperationType.DisplayInGrid
732. || GridDivisionCount\_X \* EachGridDivisionCount\_X < 1
733. || GridDivisionCount\_Y \* EachGridDivisionCount\_Y < 1
734. || **this**.IsGridVisible == **false**)
735. **return**;
736. **if** (e.Clicks == 2 && **this**.IsQueryIntersection == **true** && **this**.ShowTopology == **false** && **this**.IsGridVisible == **true**)
737. {
738. SelectPointX = SelectPointY = -1;
739. **int** gridScreen\_AxisX\_count = GridScreen\_AxisX.Count;
740. **for** (**int** i = 0; i < gridScreen\_AxisX\_count; i++)
741. {**if** (Math.Abs(GridScreen\_AxisX[i] - **this**.MouseLocation.X) < **this**.SelectPixelThreshold)
742. SelectPointX = **this**.agisControl.GetRealWorldLocX((**float**)GridScreen\_AxisX[i]);}
743. **int** gridScreen\_AxisY\_count = GridScreen\_AxisY.Count;
744. **for** (**int** i = 0; i < gridScreen\_AxisY\_count; i++)
745. { **if** (Math.Abs(GridScreen\_AxisY[i] - **this**.MouseLocation.Y) < **this**.SelectPixelThreshold)
746. SelectPointY = **this**.agisControl.GetRealWorldLocY((**float**)GridScreen\_AxisY[i]);}
747. //选中了格网点
748. **if** (SelectPointX != -1 && SelectPointY != -1 && agisControl.GridIntMethod != Mehtod.GridInterpolationMehtod.None)
749. {
750. **this**.agisControl.Refresh();
751. **string** MethodName = "";
752. **string** Para = "";
753. **if** (agisControl.GridIntMethod == Mehtod.GridInterpolationMehtod.按方位加权平均法)
754. {**if** (agisControl.按方位加权平均法SectorNum < 0)
755. { MessageBox.Show("按方位加权平均法 参数尚未设置", "错误"); **return**; }
756. MethodName = "按方位加权平均法";
757. Para = **string**.Format("{0}:{1}", "每个象限等分扇区数N0", agisControl.按方位加权平均法SectorNum / 4);}
758. **else** **if** (agisControl.GridIntMethod == Mehtod.GridInterpolationMehtod.距离平方倒数法)
759. {**if** (agisControl.距离平方倒数法NearPts < 0)
760. { MessageBox.Show("距离平方倒数法 参数尚未设置", "错误"); **return**; }
761. MethodName = "距离平方倒数法";
762. Para = **string**.Format("{0}:{1}", "选取距插值点最近的N个点", agisControl.距离平方倒数法NearPts);}
763. MessageBox.Show(**string**.Format("{0}\t\r\nX:{1}\t\nY:{2}\t\r\nValue:{3}\r\n\r\n{4}\r\n{5}",
764. "格网点属性信息：", SelectPointX.ToString("0.00"), SelectPointY.ToString("0.00"),
765. agisControl.GetGridInterpolationValue(SelectPointX, SelectPointY).ToString("0.000"),
766. "插值方法：" + MethodName, Para
767. ), "属性查询");
768. }
769. }
770. **if** (e.Clicks == 2 && **this**.IsQueryTopoPolygon == **true** && **this**.ShowTopology == **true** && **this**.ShowTopoPolygon == **true**)
771. {
772. TopoPoint clickLoc = **new** TopoPoint(agisControl.GetRealWorldLocX(e.X), agisControl.GetRealWorldLocX(e.Y), 0, **false**);
773. **this**.SelectedTopoPolygon = **this**.mTopoPolygonSet.GetClickPointInsidePolygon(clickLoc);
774. **this**.agisControl.Refresh();
775. **if** (SelectedTopoPolygon != **null**)
776. MessageBox.Show(**string**.Format("PID:{0}\r\n弧段数:{1}\r\n周长:{2}\r\n面积:{3}",
777. SelectedTopoPolygon.PID, SelectedTopoPolygon.TopologyArcs.Count,
778. SelectedTopoPolygon.GetPerimeter().ToString("0.00"),
779. SelectedTopoPolygon.GetArea().ToString("0.00")), "多边形信息");
780. }
781. }
783. **private** **void** agisControl\_Load(**object** sender, EventArgs e) { }
785. **private** **void** 显示隐藏TINToolStripMenuItem\_Click(**object** sender, EventArgs e)
786. {
787. **this**.ShowTin = (显示隐藏TINToolStripMenuItem.Checked == **true**);
788. agisControl.Refresh();
789. }
791. **private** **void** 生成等值线ToolStripMenuItem1\_CheckedChanged(**object** sender, EventArgs e)
792. { }
794. **private** **void** Set等值线可见性(**bool** isVisable)
795. {
796. **this**.ShowContourLine = isVisable;
797. 生成等值线ToolStripMenuItem1.Checked = isVisable;
798. 生成等值线ToolStripMenuItem.Checked = isVisable;
799. agisControl.Refresh();
800. }
802. **public** **void** GenerateTopologyRelatation(ContourPolyline[] contourLines)
803. {
804. **double** BottomY = agisControl.MBR\_Origin.MinY;
805. **double** TopY = agisControl.MBR\_Origin.MaxY;
806. **double** LeftX = agisControl.MBR\_Origin.MinX;
807. **double** RightX = agisControl.MBR\_Origin.MaxX;
808. **double** CenterX = (LeftX + RightX) / 2;
809. **double** CenterY = (BottomY + TopY) / 2;
810. DataPoint rectP0 = **new** DataPoint(-10000, "Rect0", CenterX, CenterY, **this**.agisControl.GetGridInterpolationValue(CenterX, CenterY));
811. DataPoint rectP1 = **new** DataPoint(-10001, "Rect1", CenterX, TopY, **this**.agisControl.GetGridInterpolationValue(CenterX, TopY));
812. DataPoint rectP2 = **new** DataPoint(-10002, "Rect2", RightX, TopY, **this**.agisControl.GetGridInterpolationValue(RightX, TopY));
813. DataPoint rectP3 = **new** DataPoint(-10003, "Rect3", RightX, CenterY, **this**.agisControl.GetGridInterpolationValue(RightX, CenterY));
814. DataPoint rectP4 = **new** DataPoint(-10004, "Rect4", RightX, BottomY, **this**.agisControl.GetGridInterpolationValue(RightX, BottomY));
815. DataPoint rectP5 = **new** DataPoint(-10005, "Rect5", CenterX, BottomY, **this**.agisControl.GetGridInterpolationValue(CenterX, BottomY));
816. DataPoint rectP6 = **new** DataPoint(-10006, "Rect6", LeftX, BottomY, **this**.agisControl.GetGridInterpolationValue(LeftX, BottomY));
817. DataPoint rectP7 = **new** DataPoint(-10007, "Rect7", LeftX, CenterY, **this**.agisControl.GetGridInterpolationValue(LeftX, CenterY));
818. DataPoint rectP8 = **new** DataPoint(-10008, "Rect8", LeftX, TopY, **this**.agisControl.GetGridInterpolationValue(LeftX, TopY));
819. //给定的边
820. List<Edge> GivenEdges = **new** List<Edge>();
821. //矩形边缘
822. GivenEdges.Add(**new** Edge(rectP1, rectP2));
823. GivenEdges.Add(**new** Edge(rectP2, rectP3));
824. GivenEdges.Add(**new** Edge(rectP3, rectP4));
825. GivenEdges.Add(**new** Edge(rectP4, rectP5));
826. GivenEdges.Add(**new** Edge(rectP5, rectP6));
827. GivenEdges.Add(**new** Edge(rectP6, rectP7));
828. GivenEdges.Add(**new** Edge(rectP7, rectP8));
829. GivenEdges.Add(**new** Edge(rectP8, rectP1));
830. //矩形中心
831. GivenEdges.Add(**new** Edge(rectP0, rectP1));
832. GivenEdges.Add(**new** Edge(rectP0, rectP2));
833. GivenEdges.Add(**new** Edge(rectP0, rectP3));
834. GivenEdges.Add(**new** Edge(rectP0, rectP4));
835. GivenEdges.Add(**new** Edge(rectP0, rectP5));
836. GivenEdges.Add(**new** Edge(rectP0, rectP6));
837. GivenEdges.Add(**new** Edge(rectP0, rectP7));
838. GivenEdges.Add(**new** Edge(rectP0, rectP8));
839. //产生的结果
840. List<ContourPolyline> resultPolylineList = **new** List<ContourPolyline>();
841. resultPolylineList.AddRange(contourLines);
842. List<Edge> resultEdgeList = **new** List<Edge>();
843. //resultEdgeList.AddRange(GivenEdges.ToArray());
844. **for** (**int** i = 0; i < GivenEdges.Count; i++)
845. {
846. Object[] resIntersect = ContourPolyline.IntersectResult(resultPolylineList.ToArray(), GivenEdges[i]);
847. List<ContourPolyline> subPolyline = (List<ContourPolyline>)resIntersect[0];
848. List<Edge> subEdge = (List<Edge>)resIntersect[1];
849. resultPolylineList = subPolyline;
850. resultEdgeList.AddRange(subEdge);
851. }
852. **this**.mSubPolyline = resultPolylineList;
853. **this**.mSubEdge = resultEdgeList;
854. **return**;
855. }
857. /// <summary>
858. /// 转化边至拓扑边，生成拓扑边集合
859. /// </summary>
860. **public** **void** ConvertLineEdgeToPolyline()
861. {
862. List<TopoPolyline> topoLineList = **new** List<TopoPolyline>();
863. **foreach** (var subline **in** mSubPolyline)
864. topoLineList.Add(**new** TopoPolyline(subline));
865. **foreach** (var subEdge **in** mSubEdge)
866. topoLineList.Add(**new** TopoPolyline(subEdge));
867. **this**.mTopoPolylineSet = **new** TopoPolylineSet(topoLineList.ToArray());
868. }
870. **private** **void** 拓扑点ToolStripMenuItem\_Click(**object** sender, EventArgs e)
871. { 拓扑点ToolStripMenuItem.Checked = (拓扑点ToolStripMenuItem.Checked == **false**); }
873. **private** **void** 拓扑边ToolStripMenuItem\_Click(**object** sender, EventArgs e)
874. { 拓扑边ToolStripMenuItem.Checked = (拓扑边ToolStripMenuItem.Checked == **false**); }
876. **private** **void** 拓扑多边形ToolStripMenuItem\_Click(**object** sender, EventArgs e)
877. { 拓扑多边形ToolStripMenuItem.Checked = (拓扑多边形ToolStripMenuItem.Checked == **false**); }
879. **private** **void** 拓扑点ToolStripMenuItem\_CheckedChanged(**object** sender, EventArgs e)
880. { **this**.ShowTopoPoint = 拓扑点ToolStripMenuItem.Checked; **this**.Refresh(); }
882. **private** **void** 拓扑边ToolStripMenuItem\_CheckedChanged(**object** sender, EventArgs e)
883. { **this**.ShowTopoPolyline = 拓扑边ToolStripMenuItem.Checked; **this**.Refresh(); }
885. **private** **void** 拓扑多边形ToolStripMenuItem\_CheckedChanged(**object** sender, EventArgs e)
886. { **this**.ShowTopoPolygon = 拓扑多边形ToolStripMenuItem.Checked; **this**.Refresh(); }
888. **private** **void** 查询ToolStripMenuItem\_CheckedChanged(**object** sender, EventArgs e)
889. { **this**.IsQueryTopoPolygon = 查询ToolStripMenuItem.Checked; }
891. **private** **void** 作者信息ToolStripMenuItem\_Click(**object** sender, EventArgs e)
892. {
893. MessageBox.Show(**this**, **string**.Format(
894. @"
895. 作者：     SunQi
896. 作者单位： 北京大学地空学院
897. 专业：     地图学与地理信息系统
898. 项目：     https://github.com/Qi-Sun/AGIS-Task
899. "
900. ), "作者信息", MessageBoxButtons.OK);
901. }
903. **private** **void** 清除格网ToolStripMenuItem\_Click(**object** sender, EventArgs e)
904. {
905. GridDivisionCount\_X = 0;
906. GridDivisionCount\_Y = 0;
907. EachGridDivisionCount\_X = 1;
908. EachGridDivisionCount\_Y = 1;
909. **this**.IsGridVisible = **false**;
910. **this**.显示隐藏格网ToolStripMenuItem.Checked = **false**;
911. }
913. **private** **void** 清楚TINToolStripMenuItem\_Click(**object** sender, EventArgs e)
914. { **this**.ShowTin = **false**; }
916. **private** **void** 平滑ToolStripMenuItem\_Click(**object** sender, EventArgs e)
917. { **this**.ContourLineUseSpline = (平滑ToolStripMenuItem.Checked == **true**); }
918. }
919. }

# END