北京大學

高级地理信息系统

编程作业：
主要源程序代码及说明

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

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;
6.
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.                 {
```

```
                    if (relativeLocOnEdge < 1 && relativeLocOnEdge > 0)
                    { intersectP.RelativeLoc = relativeLocOnEdge; subEdgePoint.Add(intersectP); }
                    if (relativeLocOnLine <= 1 && relativeLocOnLine > 0)
                    {
                        subLinePoint.Add(intersectP);
                        sublineFromPL1.Add(new ContourPolyline(subLinePoint.ToArray()));
                        subLinePoint = new List<DataPoint>();
                        subLinePoint.Add(intersectP);
                    }
                }
                subLinePoint.Add(pl1.PointList[i + 1]);
            }
            sublineFromPL1.Add(new ContourPolyline(subLinePoint.ToArray()));
            subEdgePoint.Sort((x, y) => x.RelativeLoc.CompareTo(y.RelativeLoc));
            for (int i = 0; i < subEdgePoint.Count - 1; i++)
            { suEdgeFromEdge.Add(new Edge(subEdgePoint[i], subEdgePoint[i + 1])); }
            return new Object[2] { sublineFromPL1, suEdgeFromEdge };
        }
        //获取多条等值线与线段你的交点
        public static Object[] IntersectResult(ContourPolyline[] plineList, Edge edge)
        {
            List<ContourPolyline> sublineFromPLs = new List<ContourPolyline>();
            List<Edge> suEdgeFromEdge = new List<Edge>();
            //对边上点排序
            List<DataPoint> subEdgePoint = new List<DataPoint>();
            subEdgePoint.Add(edge.StartPoint);
            subEdgePoint.Add(edge.EndPoint);
            edge.StartPoint.RelativeLoc = 0;
            edge.EndPoint.RelativeLoc = 1;
            for (int k = 0; k < plineList.Length; k++)
            {
                //对折线上点排序
                List<DataPoint> subLinePoint = new List<DataPoint>();
                ContourPolyline curCpl = plineList[k];
                subLinePoint.Add(curCpl.PointList[0]);
                //选取一个等值线
                for (int i = 0; i < curCpl.PointList.Count - 1; i++)
                {
                    Edge pl1OneEdge = new Edge(curCpl.PointList[i], curCpl.PointList[i + 1]);
                    DataPoint intersectP = Edge.IntersectPoint(pl1OneEdge, edge);
                    double relativeLocOnLine = Edge.IntersectPointRelativeLoc(pl1OneEdge, edge);
                    double relativeLocOnEdge = Edge.IntersectPointRelativeLoc(edge, pl1OneEdge);
                    if (intersectP != null)
                    {
                        if (relativeLocOnEdge < 1 && relativeLocOnEdge > 0)
                        { intersectP.RelativeLoc = relativeLocOnEdge; subEdgePoint.Add(intersectP)
    ; }
```

```
                        if (relativeLocOnLine <= 1 && relativeLocOnLine > 0)
                        {
                            if (subLinePoint.Count == 1 && subLinePoint[0].OID == intersectP.OID) { }
                            else
                            {
                                subLinePoint.Add(intersectP);
                                sublineFromPLs.Add(new ContourPolyline(subLinePoint.ToArray()));
                                subLinePoint = new List<DataPoint>();
                                subLinePoint.Add(intersectP);
                            }
                        }
                    }
                    subLinePoint.Add(curCpl.PointList[i + 1]);
                }
                if (subLinePoint.Count >= 2 && !(subLinePoint.Count == 2 && subLinePoint[0].OID == subLinePoint[1].OID))
                    sublineFromPLs.Add(new ContourPolyline(subLinePoint.ToArray()));
            }
            subEdgePoint.Sort((x, y) => x.RelativeLoc.CompareTo(y.RelativeLoc));
            for (int i = 0; i < subEdgePoint.Count - 1; i++)
            {
                if (subEdgePoint[i].RelativeLoc != subEdgePoint[i + 1].RelativeLoc)
                    suEdgeFromEdge.Add(new Edge(subEdgePoint[i], subEdgePoint[i + 1]));
            }
            return new Object[2] { sublineFromPLs, suEdgeFromEdge };
        }

        public override string ToString()
        { return string.Format("CLid:{0},PtsCount:{1}", this.PID, this.PointList.Count); }
    }
}
```

ContourPolylineSet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AGIS_work.DataStructure
{
    /// <summary>
    /// 等值线集合
```

```
    /// </summary>
    public class ContourPolylineSet
    {
        public List<ContourPolyline> ContourPolylineList = new List<ContourPolyline>();
        public ContourPolylineSet() { }
        public ContourPolylineSet(ContourPolyline[] polylines) { ContourPolylineList.AddRange(polylines); }
    }  }
```

DataPoint.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AGIS_work.DataStructure
{
    /// <summary>
    /// 数据点类
    /// </summary>
    public class DataPoint
    {
        public int ID { get; private set; }
        public string Name { get; private set; }
        public double X { get; private set; }
        public double Y { get; private set; }
        public double Value { get; private set; }
        public MinBoundRect MBR { get; private set; }
        public int OID { get; private set; }
        private static int _oid = 1000000;
        public double RelativeLoc { get; set; }

        public DataPoint(int id, string name, double x, double y, double value,int oid)
        {
            this.ID = id;
            this.Name = name;
            this.X = x;
            this.Y = y;
            this.Value = value;
            this.MBR = new MinBoundRect(x, y, x, y);
            this.OID = oid;
        }

```

```
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.         }
45.
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);}
49.
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));}
53.
54.         public double GetDistance(double x, double y)
55.         {return Math.Sqrt(Math.Pow(this.X - x, 2) + Math.Pow(this.Y - y, 2));}
56.
57.         public double GetDistanceP2(double x, double y)
58.         {return (Math.Pow(this.X - x, 2) + Math.Pow(this.Y - y, 2));}
59.
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.             }
78.
79.         }
80.
81.         public static Vector2D operator - (DataPoint p1 ,DataPoint p2)
```

```
        {return new Vector2D(p1.X - p2.X, p1.Y - p2.Y);}

        //获取三点构成的角度
        public static double Angle(DataPoint c, DataPoint a, DataPoint b)
        {
            double ang;
            double l1 = Math.Sqrt((b.X - c.X) * (b.X - c.X) + (b.Y - c.Y) * (b.Y - c.Y));
            double l2 = Math.Sqrt((a.X - c.X) * (a.X - c.X) + (a.Y - c.Y) * (a.Y - c.Y));
            double l3 = Math.Sqrt((b.X - a.X) * (b.X - a.X) + (b.Y - a.Y) * (b.Y - a.Y));
            ang = Math.Acos((l1 * l1 + l2 * l2 - l3 * l3) / (2 * l1 * l2));
            return ang;
        }
        public static int LeftOrRight(DataPoint c, DataPoint a, DataPoint b)
        {
            int youbian;
            double S= (a.X - c.X) * (b.Y - c.Y) - (a.Y - c.Y) * (b.X - c.X);
            if (S > 0){youbian = 1;}
            else if (S < 0) {youbian = -1;}
             else{youbian = 0;}
             return youbian;
         }
    }
}
```

Edge.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AGIS_work.DataStructure
{
    /// <summary>
    /// 线段类
    /// </summary>
    public class Edge
    {
        public int EID { get; private set; }
        public DataPoint StartPoint { get; private set; }
        public DataPoint EndPoint { get; private set; }
        public int StartOID
        { get { return StartPoint.OID; } }
        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;
24.
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); }
33.
34.         public double MinValue()
35.         { return Math.Min(StartPoint.Value, EndPoint.Value); }
36.
37.         public double GetRelativeCoordinate(double value)
38.         { return (value - StartPoint.Value) / (EndPoint.Value - StartPoint.Value); }
39.
40.         public double GetValue(double ralativeCoordinate)
41.         { return ralativeCoordinate * (EndPoint.Value - StartPoint.Value) + StartPoint.Value; }
42.
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.         }
48.
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.Start
    Point.Y - e1.StartPoint.Y) -
54.                 e2.StartPoint.X * (e1.EndPoint.X - e1.StartPoint.X) * (e2.StartPoint.Y - e2.EndPoi
    nt.Y) +
55.                 e1.StartPoint.X * (e1.EndPoint.Y - e1.StartPoint.Y) * (e2.StartPoint.X - e2.EndPoi
    nt.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.Start
    Point.X - e1.StartPoint.X) -
60.                 e2.StartPoint.Y * (e1.EndPoint.Y - e1.StartPoint.Y) * (e2.StartPoint.X - e2.EndPoi
    nt.X) +
```

```
61.                 e1.StartPoint.Y * (e1.EndPoint.X - e1.StartPoint.X) * (e2.StartPoint.Y - e2.EndPoi
    nt.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) + Mat
    h.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.         }
86.
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.StartP
    oint.Y - e1.StartPoint.Y) -
91.                 e2.StartPoint.X * (e1.EndPoint.X - e1.StartPoint.X) * (e2.StartPoint.Y - e2.EndPoin
    t.Y) +
92.                 e1.StartPoint.X * (e1.EndPoint.Y - e1.StartPoint.Y) * (e2.StartPoint.X - e2.EndPoin
    t.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.Start
    Point.X - e1.StartPoint.X) -
97.                 e2.StartPoint.Y * (e1.EndPoint.Y - e1.StartPoint.Y) * (e2.StartPoint.X - e2.EndPoi
    nt.X) +
```

```
                e1.StartPoint.Y * (e1.EndPoint.X - e1.StartPoint.X) * (e2.StartPoint.Y - e2.EndPoint.Y)) /
                ((e1.EndPoint.X - e1.StartPoint.X) * (e2.StartPoint.Y - e2.EndPoint.Y) -
                 (e1.EndPoint.Y - e1.StartPoint.Y) * (e2.StartPoint.X - e2.EndPoint.X));
            double relativeE1 = 0;
            if ((e1.EndPoint.X - e1.StartPoint.X) != 0)
                relativeE1 = (IntersectX - e1.StartPoint.X) / (e1.EndPoint.X - e1.StartPoint.X);

            else relativeE1 = (IntersectY - e1.StartPoint.Y) / (e1.EndPoint.Y - e1.StartPoint.Y);

            if (Math.Abs(relativeE1) < 1E-5)
                return 0;
            else if (Math.Abs(relativeE1 - 1) < 1E-5)
                return 1;
            else return relativeE1;
        }

        /// <summary>
        /// 判断边 2 两点是否在边 1 两侧
        /// </summary>
        /// <param name="e1"></param>
        /// <param name="e2"></param>
        /// <returns></returns>
        public static bool CheckCross(Edge e1, Edge e2)
        {
            Vector2D v1 = e1.StartPoint - e1.EndPoint;
            Vector2D v2 = e2.StartPoint - e1.EndPoint;
            Vector2D v3 = e2.EndPoint - e2.EndPoint;
            return v1.CrossProduct(v2) * v1.CrossProduct(v3) < 0;
        }

        public override string ToString()
        { return string.Format("EdgeID:{0},StaID:{1},EndID:{2}", this.EID, this.StartOID, this.EndOID); }
    }
}
```

EdgeSet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
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); }
15.
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.         }
27.
28.         public void AddEdge(Edge e)
29.         { EdgeList.Add(e); }
30.
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.         }
40.
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.         }
50.
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.                 }
81.
82.             }
83.             return triangleSet;
84.         }
85.
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));
```

```
                        ContourEdgeSet.EdgeList.Remove(startEdge);
                    }
                    else break;
                }
                ContourPolyline tempPolyline = new ContourPolyline(tempPointList.ToArray());
                ContourPolylineList.Add(tempPolyline);
            }
            return new ContourPolylineSet(ContourPolylineList.ToArray());
        }
    }
}
```

MBR.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AGIS_work.DataStructure
{
    /// <summary>
    /// 最小外包矩形类
    /// </summary>
    public class MinBoundRect
    {
        public double MinX { get; private set; }
        public double MinY { get; private set; }
        public double MaxX { get; private set; }
        public double MaxY { get; private set; }

        public MinBoundRect()
        {
            this.MinX = double.MaxValue;
            this.MinY = double.MaxValue;
            this.MaxX = double.MinValue;
            this.MaxY = double.MinValue;
        }
        public MinBoundRect(double minX,double minY,double maxX,double maxY)
        {
            this.MinX = minX;
            this.MinY = minY;
            this.MaxX = maxX;
            this.MaxY = maxY;
```

```
        }

        public void UpdateRect(double x,double y)
        {
            this.MinX = Math.Min(this.MinX, x);
            this.MinY = Math.Min(this.MinY, y);
            this.MaxX = Math.Max(this.MaxX, x);
            this.MaxY = Math.Max(this.MaxY, y);
            return;
        }

        public void UpdateRect(MinBoundRect mbr)
        {
            this.MinX = Math.Min(this.MinX, mbr.MinX);
            this.MinY = Math.Min(this.MinY, mbr.MinY);
            this.MaxX = Math.Max(this.MaxX, mbr.MaxX);
            this.MaxY = Math.Max(this.MaxY, mbr.MaxY);
            return;
        }

        public void PanningVector(double deltaX,double deltaY)
        {
            this.MinX += deltaX;
            this.MinY += deltaY;
            this.MaxX += deltaX;
            this.MaxY += deltaY;
        }

        public void ZoomPointAndRatio(double x,double y,double ratio)
        {
            this.MinX = x - (x - this.MinX) * ratio;
            this.MinY = y - (y - this.MinY) * ratio;
            this.MaxX = (this.MaxX - x) * ratio + x;
            this.MaxY = (this.MaxY - y) * ratio + y;
        }
    }
}
```

PointSet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
using System.IO;

namespace AGIS_work.DataStructure
{
    /// <summary>
    /// 数据点集合
    /// </summary>
    public class PointSet
    {
        public string SetName { get; private set; }
        public string FileName { get; private set; }

        public List<DataPoint> PointList { get; private set; }
        public MinBoundRect MBR { get; private set; }

        public PointSet() { MBR = new MinBoundRect(-1, -
1, 1, 1); PointList = new List<DataPoint>(); }
        public PointSet(string setname, string filename, DataPoint[] points)
        {
            this.SetName = setname;
            this.FileName = filename;
            this.PointList = new List<DataPoint>(points);
            //最小外接矩形
            MBR = new MinBoundRect();
            foreach (DataPoint point in points)
                MBR.UpdateRect(point.X, point.Y);
        }

        /// <summary>
        /// 从 CSV 文件中读取点集
        /// </summary>
        /// <param name="filename"></param>
        /// <returns></returns>
        public static PointSet ReadFromCSV(string filename)
        {
            PointSet pointSet = null;
            StreamReader sr = new StreamReader(filename);
            List<DataPoint> dataPoints = new List<DataPoint>();
            try
            {
                string setName = sr.ReadLine();
                int oid = 0;
                while (!sr.EndOfStream)
                {
                    string onePoint = sr.ReadLine();
                    string[] pointInfo = onePoint.Split(',');
```

```
                    dataPoints.Add(new DataPoint(int.Parse(pointInfo[0]), pointInfo[1], double.Parse(pointInfo[2]),
                        double.Parse(pointInfo[3]), double.Parse(pointInfo[4])));
                    oid++;
                }
                pointSet = new PointSet(setName, filename, dataPoints.ToArray());
            }
            catch (Exception err) { throw err; }
            sr.Close();
            return pointSet;
        }

        /// <summary>
        /// 将点集写入 CSV 文件
        /// </summary>
        /// <param name="filename"></param>
        public void WriteToCSV(string filename = null)
        {
            string filePath = filename == null ? this.FileName : filename;
            StreamWriter sw = new StreamWriter(filePath);
            try
            {
                sw.WriteLine(this.SetName);
                foreach (DataPoint point in this.PointList)
                {
                    sw.WriteLine(string.Format("{0},{1},{2},{3},{4}", point.ID, point.Name, point.X, point.Y, point.Value));
                }

            }
            catch (Exception err)
            {
                throw err;
            }
            sw.Close();
            return;
        }

        /// <summary>
        /// 根据 OID 返回数据点
        /// </summary>
        /// <param name="oid"></param>
        /// <returns></returns>
        public DataPoint GetPointByOID(int oid)
        {
            foreach (var point in PointList)
            { if (point.OID == oid) return point; }
```

```
96.             return null;
97.         }
98.
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.         }
110.
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;
6.
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
20.
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.         }
31.
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.             }
84.
85.         }
86.
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);   }
94.
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)); }
98.
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;
7.
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; }
21.
22.         public TopoPointSet()
23.         {
24.             TopoPointList = new List<TopoPoint>();
25.             TopoNodeList = new List<TopoPoint>();
26.         }
27.
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.         }
40.
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.         }
55.
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.         }
70.
71.         public TopoPoint GetNodeByPointID(int poid)
72.         {
73.             foreach (var point in TopoNodeList)
74.             { if (point.PointID == poid) return point; }
75.             return null;
76.         }
77.
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.         }
107.
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;
```

```
                    }
                    Tuple<TopoPoint, double, double, int, TopoPolyline> curPointStruct
                        = new Tuple<TopoPoint, double, double, int, TopoPolyline>(arc.GetAnotherN
ode(tpPoint), angle, otherNodeAngle, direct, arc);
                    curPointStructList.Add(curPointStruct);
                }
                List<TopoPolyline> curPolygon = new List<TopoPolyline>();
                List<int> directList = new List<int>();
                foreach (var pointStruct in curPointStructList)
                {
                    var iterationTuple = pointStruct;
                    while (iterationTuple.Item1.PointID != tpPoint.PointID)
                    {
                        curPolygon.Add(iterationTuple.Item5);
                        directList.Add(iterationTuple.Item4);
                        iterationTuple = this.GetNextNode(iterationTuple);
                    }
                    curPolygon.Add(iterationTuple.Item5);
                    directList.Add(iterationTuple.Item4);
                    //追踪成功
                    if (curPolygon.Count > 0)
                    {
                        TopoPolygon tempPolygon = new TopoPolygon(curPolygon.ToArray());
                        for (int i = 0; i < directList.Count; i++)
                        {
                            if (directList[i] > 0)
                                curPolygon[i].RightPolygon = tempPolygon;
                            else if (directList[i] < 0)
                                curPolygon[i].LeftPolygon = tempPolygon;
                        }
                        polygonList.Add(tempPolygon);
                        curPolygon = new List<TopoPolyline>();
                        directList = new List<int>();
                    }
                }

            }
            return new TopoPolygonSet(polygonList.ToArray());
        }
        //对搜索边进行排序
        public void SortTheSearchLine(List<Tuple<TopoPoint, double, double, int, TopoPolyline>> l
ineToSort, double startAngle)
        {
            lineToSort.Sort((x, y) =>
            {
                double x2 = x.Item2 - startAngle;
                double y2 = y.Item2 - startAngle;
```

```
                if (x2 <= 0) x2 += 360;
                if (y2 <= 0) y2 += 360;
                return x2.CompareTo(y2);
            });
            return;
        }
        //根据点标识获取点
        public TopoPoint GetTopoPointByID(int poid)
        {
            foreach (var point in this.TopoPointList)
            {
                if (point.PointID == poid)
                    return point;
            }
            return null;
        }

        /// <summary>
        /// 循环搜索下一个节点
        /// </summary>
        /// <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>
        /// <returns></returns>
        public Tuple<TopoPoint, double, double, int, TopoPolyline> GetNextNode(Tuple<TopoPoint, double, double, int, TopoPolyline> curNode)
        {
            List<TopoPolyline> curArcs = this.GetNodeByPointID(curNode.Item1.PointID).TopologyArcs;
            List<Tuple<TopoPoint, double, double, int, TopoPolyline>> tempList = new List<Tuple<TopoPoint, double, double, int, TopoPolyline>>();
            foreach (var arc in curArcs)
            {
                int direct = arc.IsNode(curNode.Item1);
                double angle = -1;
                double otherNodeAngle = -2;
                switch (direct)
                {
                    case 1:
                        angle = arc.GetBeginNodeAngle();
                        otherNodeAngle = arc.GetEndNodeAngle();
                        break;
                    case -1:
                        angle = arc.GetEndNodeAngle();
                        otherNodeAngle = arc.GetBeginNodeAngle();
                        break;
                    default:
                        break;
```

```
                }
                tempList.Add(new Tuple<TopoPoint, double, double, int, TopoPolyline>(
                    arc.GetAnotherNode(curNode.Item1), angle, otherNodeAngle, direct, arc));
            }
            SortTheSearchLine(tempList, curNode.Item3);
            return tempList[0];
        }
        //导出结点关系表至文件
        public void SaveNodeTableToFile(string filename)
        {
            StreamWriter sw = new StreamWriter(filename);
            sw.WriteLine("ID\tNode_ID\tX\tY\tZ\tArc_Num\tArc_IDs");
            foreach (var node in TopoNodeList)
            {
                string arcIDs = "";
                foreach (var arc in node.TopologyArcs)
                    arcIDs += arc.ArcID + ",";
                sw.WriteLine(
                    string.Format("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}",
                    node.Innerid, node.PointID, node.X, node.Y, node.Z, node.TopologyArcs.Count,
arcIDs.Remove(arcIDs.Length - 1)));
            }
            sw.Close();
        }
        //导出中间点关系表至文件
        public void SavePointTableToFile(string filename)
        {
            StreamWriter sw = new StreamWriter(filename);
            sw.WriteLine("ID\tNode_ID\tX\tY\tZ\tArc_Num\tArc_IDs");
            foreach (var point in TopoPointList)
            {
                string arcIDs = "";
                foreach (var arc in point.TopologyArcs)
                    arcIDs += arc.ArcID + ",";
                sw.WriteLine(
                    string.Format("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}",
                    point.Innerid, point.PointID, point.X, point.Y, point.Z, point.TopologyArcs.C
ount, arcIDs.Remove(arcIDs.Length - 1)));
            }
            sw.Close();
        }
    }
}
```

TopoPolygon.cs

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.
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; }
19.
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.         }
29.
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.         }
49.
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, Top
    oPoint 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;
7.
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; }
14.
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)
```

```
            { if (polygon.PID == pid) return true; }
            return false;
        }
        //获取选中的多边形
        public TopoPolygon GetClickPointInsidePolygon(TopoPoint clickPoint)
        {
            foreach (var polygon in this.TopoPolygonList)
            { if (polygon.IfPointInRegion(clickPoint)) return polygon; }
            return null;
        }
        //导出多边形拓扑关系表至文件
        public void SavePolygonTableToFile(string filename)
        {
            StreamWriter sw = new StreamWriter(filename);
            sw.WriteLine(string.Format("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}\t{9}",
                    "ID", "Pol_ID", "Arc_Num", "ArcIds", "OuterPol_ID", "InnerPol_ID", "LX", "LY",
 "RX", "RY"));
            foreach (var polygon in TopoPolygonList)
            {
                string arcids = " ";
                foreach (var arc in polygon.TopologyArcs)
                { arcids += arc.ArcID + ","; }
                sw.WriteLine(string.Format("{0}\t{1}\t{2}\t{3}\t{4}\t{5}\t{6}\t{7}\t{8}\t{9}",
                    polygon.innerId, polygon.PID, polygon.TopologyArcs.Count, arcids.Remove(arcids
.Length - 1),
                    (polygon.OuterPolygon == null) ? "NULL" : polygon.OuterPolygon.PID.ToString(),

                    (polygon.InnerPolygons.Count == 0) ? "NULL" : polygon.InnerPolygons[0].PID.ToS
tring(),
                    polygon.MBR.MinX, polygon.MBR.MinY, polygon.MBR.MaxX, polygon.MBR.MaxY));
            }
            sw.Close();
        }
    }
}
```

TopoPolyline.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

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; }
21.
22.         public TopoPolyline()
23.         {
24.             this.ArcID = _ArcID++;
25.             MiddlePoint = new List<TopoPoint>();
26.             Innerid = this.ArcID;
27.             MBR = new MinBoundRect();
28.         }
29.
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.         }
59.
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.         }
```

```
102.
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.Cou
    nt);
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;
7.
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; }
14.
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{1
    1}",
31.                 "ID", "Arc_ID", "BeginNode", "EndNode", "LeftPolygon", "RightPolygon", "MiddlePtsN
    um",
```

```
                "MiddlePtsCooridinate", "LX", "LY", "RX", "RY"));
            foreach (var line in TopoPolylineList)
            {
                string middleCoorinate = " ";
                foreach (var mpoint in line.MiddlePoint)
                    middleCoorinate += string.Format("({0},{1},{2}),", mpoint.X, mpoint.Y, mpoint.
Z);
                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}",
                    line.Innerid, line.ArcID, line.BeginNode.PointID, line.EndNode.PointID,
                    (line.LeftPolygon == null) ? "NULL" : line.LeftPolygon.PID.ToString(),
                    (line.RightPolygon == null) ? "NULL" : line.RightPolygon.PID.ToString(),
                    line.MiddlePoint.Count, middleCoorinate.Remove(middleCoorinate.Length - 1),
                    line.MBR.MinX, line.MBR.MinY, line.MBR.MaxX, line.MBR.MaxY));
            }
            sw.Close();
        }
    }
}
```

Triangle.cs

```
1.  using System;
2.  using System.Collections.Generic;
3.  using System.Linq;
4.  using System.Text;
5.  using System.Threading.Tasks;
6.
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
16.
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();
```

```
            List<int> VirOIDList = new List<int>();
            VirOIDList.Add(oid1);
            VirOIDList.Add(oid2);
            VirOIDList.Add(oid3);
            VirOIDList.Sort();
            return (TriOIDList[0] == VirOIDList[0]) &&
                (TriOIDList[1] == VirOIDList[1]) &&
                (TriOIDList[2] == VirOIDList[2]);
        }
        //获取三角形内等值线
        public Edge GetContourLine(double elevation)
        {
            List<DataPoint> points = new List<DataPoint>();
            if ((elevation - VertexA.Value) * (elevation - VertexB.Value) * (elevation - VertexC.V
alue) == 0)
                elevation += 0.1;
            if ((elevation - VertexA.Value) * (elevation - VertexB.Value) < 0)
            {
                double EleX = VertexA.X + (VertexB.X - VertexA.X) * (elevation - VertexA.Value) /
(VertexB.Value - VertexA.Value);
                double EleY = VertexA.Y + (VertexB.Y - VertexA.Y) * (elevation - VertexA.Value) /
(VertexB.Value - VertexA.Value);
                DataPoint p1 = new DataPoint(VertexA.OID * 1000 + VertexB.OID,
                    "ContourPoint_" + VertexA.OID * 1000 + VertexB.OID,
                    EleX, EleY, elevation);
                points.Add(p1);
            }
            if ((elevation - VertexA.Value) * (elevation - VertexC.Value) < 0)
            {
                double EleX = VertexA.X + (VertexC.X - VertexA.X) * (elevation - VertexA.Value) /
(VertexC.Value - VertexA.Value);
                double EleY = VertexA.Y + (VertexC.Y - VertexA.Y) * (elevation - VertexA.Value) /
(VertexC.Value - VertexA.Value);
                DataPoint p1 = new DataPoint(VertexA.OID * 1000 + VertexC.OID,
                    "ContourPoint_" + VertexA.OID * 1000 + VertexC.OID,
                    EleX, EleY, elevation);
                points.Add(p1);
            }
            if ((elevation - VertexC.Value) * (elevation - VertexB.Value) < 0)
            {
                double EleX = VertexC.X + (VertexB.X - VertexC.X) * (elevation - VertexC.Value) /
(VertexB.Value - VertexC.Value);
                double EleY = VertexC.Y + (VertexB.Y - VertexC.Y) * (elevation - VertexC.Value) /
(VertexB.Value - VertexC.Value);
                DataPoint p1 = new DataPoint(VertexC.OID * 1000 + VertexB.OID,
                    "ContourPoint_" + VertexC.OID * 1000 + VertexB.OID,
                    EleX, EleY, elevation);
```

```
                points.Add(p1);
            }
            if (points.Count == 2)
            { return new Edge(points[0], points[1]); }
            else return null;
        }
    }
}
```

TriangleSet.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AGIS_work.DataStructure
{
    //三角形集合
    public class TriangleSet
    {
        public List<Triangle> TriangleList = new List<Triangle>();
        public TriangleSet() { }

        /// <summary>
        /// 移除指定 TID 的三角形
        /// </summary>
        /// <param name="tid"></param>
        public void RemoveTriangleByTID(int tid)
        {
            int index = 0;
            foreach (var tri in TriangleList)
            { if (tri.TID == tid) break; index++; }
            TriangleList.RemoveAt(index);
        }
        //添加一个三角形
        public void AddTriangle(Triangle t)
        { TriangleList.Add(t); }
        //求点所在三角形
        public Triangle GetPointInsidesTri(DataPoint p)
        {
            foreach (var tri in TriangleList)
            { if (tri.IsPointInTriangle(p)) return tri; }
            return null;
```

```
        }
        //判断是否已经存在
        public bool IsTriAlreadyExists(int oid1, int oid2, int oid3)
        {
            foreach (var tri in TriangleList)
            { if (tri.IsEqulesTri(oid1, oid2, oid3)) return true; }
            return false;
        }
    }
}
```

Vector2D.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace AGIS_work.DataStructure
{
    //二维向量
    public class Vector2D
    {
        public double X { get; private set; }
        public double Y { get; private set; }

        public Vector2D(double x, double y)
        {
            this.X = x;
            this.Y = y;
        }
        //叉乘
        public double CrossProduct(Vector2D v2)
        { return this.X * v2.Y - this.Y * v2.X; }
        //点乘
        public double DotProduct(Vector2D v2)
        { return this.X * v2.X + this.Y * v2.Y; }
        //向量相加
        public static Vector2D operator +(Vector2D v1, Vector2D v2)
        { return new Vector2D(v1.X + v2.X, v1.Y + v2.Y); }
    }
}
```

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;
9.
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
25.
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;
```

```
                    //判断第三点与前两点的位置关系
                    youbian = DataPoint.LeftOrRight((DataPoint)pointList[i], ((TinLine)tinlines[j]).Begin, ((TinLine)tinlines[j]).End);
                    if (youbian == 1)
                    {
                        //获取角度最大点
                        ang = DataPoint.Angle((DataPoint)pointList[i], ((TinLine)tinlines[j]).Begin, ((TinLine)tinlines[j]).End);
                        if (ang > minang) { minang = ang; count = i; }
                        OK = true;
                    }
                }
                if (OK == true)
                {
                    //将新生成两条边添加入集合中
                    int t1 = 0;
                    int t2 = 0;
                    tling1.Begin = ((TinLine)tinlines[j]).Begin;
                    tling1.End = (DataPoint)pointList[count];
                    tling2.Begin = (DataPoint)pointList[count];
                    tling2.End = ((TinLine)tinlines[j]).End;
                    tinlines.Add(tling1);
                    tinlines.Add(tling2);
                    for (int i = 0; i < tinlines.Count - 2; i++)
                    {
                        //判断新生成的两边是否与已生成的边重合
                        if ((tling2.Begin == ((TinLine)tinlines[i]).Begin && tling2.End == ((TinLine)tinlines[i]).End) ||
                            (tling2.Begin == ((TinLine)tinlines[i]).End && tling2.End == ((TinLine)tinlines[i]).Begin))
                        { t2 = 1; }
                        if ((tling1.Begin == ((TinLine)tinlines[i]).Begin && tling1.End == ((TinLine)tinlines[i]).End) ||
                            (tling1.Begin == ((TinLine)tinlines[i]).End && tling1.End == ((TinLine)tinlines[i]).Begin))
                        { t1 = 1; }
                    }
                    //两条边都重合
                    if (t2 == 1 && t1 == 1)
                    { for (int i = 0; i < 2; i++) { tinlines.Remove(tinlines[tinlines.Count - 1]); } }
                    //第二条边重合
                    else if (t2 == 1) { tinlines.Remove(tinlines[tinlines.Count - 1]); }
                    //第一条边重合
                    else if (t1 == 1) { tinlines.Remove(tinlines[tinlines.Count - 2]); }
                }
            }
```

```
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.PointLi
    st[kk].Y); }
164.             int i, idxStart = 0, endTemp, ptindex;
165.             bool isExist;
166.             double angMax, angMin, angTemp, angRcdMax, angRcdTmp, lenMin, lenCur, lenTmp1, lenTmp
    2;
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; } }
```

```
            endTemp = idxStart - 1;
            ptStart.X = arrDots[idxStart].X;
            ptStart.Y = arrDots[idxStart].Y;
            edge.Start = idxStart;
            angMin = Math.PI;
            dirCur.X = 0;
            dirCur.Y = 500;
            while (endTemp != idxStart)
            {
                lenCur = Math.Sqrt(dirCur.X * dirCur.X + dirCur.Y * dirCur.Y);
                lenMin = 1000;
                for (i = 0; i < arrDots.Length; i++)//找边界
                {
                    if (i != edge.Start)
                    {
                        dirTmp1.X = arrDots[i].X - ptStart.X;
                        dirTmp1.Y = arrDots[i].Y - ptStart.Y;
                        lenTmp1 = Math.Sqrt(dirTmp1.X * dirTmp1.X + dirTmp1.Y * dirTmp1.Y);
                        angTemp = Math.Acos((dirCur.X * dirTmp1.X + dirCur.Y * dirTmp1.Y) / (lenT
mp1 * lenCur));
                        if (angTemp < angMin)
                        { angMin = angTemp; edge.End = i; lenMin = lenTmp1; }
                        else if (angTemp == angMin && lenTmp1 < lenMin)
                        { edge.End = i; lenMin = lenTmp1; }
                    }
                }
                arrEdges.Add(edge);
                endTemp = edge.End;
                edge = new Edge2();
                angMin = Math.PI;
                dirCur.X = arrDots[endTemp].X - ptStart.X;
                dirCur.Y = arrDots[endTemp].Y - ptStart.Y;
                ptStart = arrDots[endTemp];
                edge.Start = endTemp;
            }
            //以下为自动生成 TIN
            //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
            //注意边可能有两种顺序存储。
            for (i = 0; i < arrEdges.Count; i++)
            {
                //取出一条边
                edge = new Edge2();
                edge = (Edge2)arrEdges[i];
                //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）？？
                if (edge.LeftTri == -1)
                {
```

```
                    ptindex = -1;//选中的点的 index
                    dirCur.X = arrDots[edge.End].X - arrDots[edge.Start].X;
                    dirCur.Y = arrDots[edge.End].Y - arrDots[edge.Start].Y;
                    angRcdMax = 0;//与该边夹角最大值
                    angMax = 0;//最大圆内接角
                    for (int j = 0; j < arrDots.Length; j++)
                    {
                        if (j != edge.Start && j != edge.End)//排除边的端点
                        {
                            dirTmp1.X = arrDots[j].X - arrDots[edge.Start].X;
                            dirTmp1.Y = arrDots[j].Y - arrDots[edge.Start].Y;
                            if (dirCur.X * dirTmp1.Y - dirCur.Y * dirTmp1.X < 0)//如果该点在左边，则计算
                            {
                                //找角度最大的
                                lenCur = Math.Sqrt(dirCur.X * dirCur.X + dirCur.Y * dirCur.Y);//当前向量长度
                                lenTmp1 = Math.Sqrt(dirTmp1.X * dirTmp1.X + dirTmp1.Y * dirTmp1.Y);
                                dirTmp2.X = arrDots[j].X - arrDots[edge.End].X;
                                dirTmp2.Y = arrDots[j].Y - arrDots[edge.End].Y;
                                lenTmp2 = Math.Sqrt(dirTmp2.X * dirTmp2.X + dirTmp2.Y * dirTmp2.Y);
                                angRcdTmp = Math.Acos((dirCur.X * dirTmp1.X + dirCur.Y * dirTmp1.Y) / (lenTmp1 * lenCur));
                                angTemp = Math.Acos((dirTmp2.X * dirTmp1.X + dirTmp2.Y * dirTmp1.Y) / (lenTmp1 * lenTmp2));
                                if (angTemp > angMax)
                                { angMax = angTemp; angRcdMax = angRcdTmp; ptindex = j; }
                                else if (angTemp == angMax && angRcdMax < angRcdTmp)//相等取最左
                                { angRcdMax = angRcdTmp; ptindex = j; }
                            }
                        }
                    }
                    if (ptindex != -1)//选择有点
                    {
                        //记录三角形
                        Tri tri = new Tri();
                        tri.NodeA = edge.Start;
                        tri.NodeB = edge.End;
                        tri.NodeC = ptindex;
                        edge.LeftTri = arrTris.Count;
                        isExist = false;
                        //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
                        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
    );
335.
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++)
```

```
                        {
                            Edge2 e = (Edge2)arrEdges[k];
                            if (e.Start == edge.End && e.End == ptindex)//如果存在过这条边，则记录其右三角形
                            {
                                e.RightTri = arrTris.Count;
                                tri.AdjTriA = e.LeftTri;
                                isExist = true;
                                break;
                            }
                            else if (e.Start == ptindex && e.End == edge.End)
                            {
                                e.LeftTri = arrTris.Count;
                                tri.AdjTriA = e.RightTri;
                                isExist = true;
                                break;
                            }
                        }
                        if (isExist == false)//如果不存在这条边，则新建一条边
                        {
                            Edge2 edgeadd = new Edge2();
                            edgeadd.Start = ptindex;
                            edgeadd.End = edge.End;
                            edgeadd.LeftTri = arrTris.Count;
                            arrEdges.Add(edgeadd);
                        }
                        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
                        arrTris.Add(tri);//add the tri to the arraylist
                    }
                }
            }
            List<Edge> EdgeList = new List<Edge>();
            for (int gg = 0; gg < arrEdges.Count; gg++)
            {
                Edge2 eg = (Edge2)arrEdges[gg];
                PointF pt1, pt2;
                pt1 = arrDots[eg.Start];
                pt2 = arrDots[eg.End];
                EdgeList.Add(new Edge(
                    new DataPoint(gg, gg.ToString(), pt1.X, pt1.Y, 0),
                    new DataPoint(-gg, (-gg).ToString(), pt2.X, pt2.Y, 0)));
            }
            return EdgeList;
        }
    }
}
```

GridCreateContourLine.cs

```cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using AGIS_work.DataStructure;

namespace AGIS_work.Mehtod
{
    //格网生成等值线
    public class GridCreateContourLine
    {
        public List<double> XAxis = new List<double>();//横线值序列
        public List<double> YAxis = new List<double>();//竖线值序列
        public double[,] HH = null; //横边追踪数组
        public double[,] SS = null; //竖边追踪数组
        public int XCount = 0;
        public int YCount = 0;
        public double Elevation = 0;//当前等值线值

        public GridCreateContourLine(List<double> xAxis, List<double> yAxis,
            double[,] hh, double[,] ss, int xCount, int yCount, double elev)
        {
            XAxis = xAxis;
            YAxis = yAxis;
            HH = hh;
            SS = ss;
            XCount = xCount;
            YCount = yCount;
            Elevation = elev;

        }

        //           2
        //       ___________
        //      |           |
        //      |           |
        //    3 |           |  5
        //      |           |
        //      |___________|
        //  (i,j)     7
        //
        //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.                     {
```

```
                        case 2:
                            tempDataPoints.Add(new DataPoint(-res[0] * 100 - res[1] - 1, "等值点
" + (-res[0] * 100 - res[1] - 1).ToString(),
                                                XAxis[res[0]] + HH[res[0], res[1] + 1] * (XAxis[res[0] + 1
] - XAxis[res[0]]),
                                                YAxis[res[1] + 1], Elevation/*, -
res[0] * 100 - res[1] - 1*/));
                            HH[res[0], res[1] + 1] = 5;
                            break;
                        case 3:
                            tempDataPoints.Add(new DataPoint(res[0] * 100 + res[1],
                                "等值点" + (res[0] * 100 + res[1]).ToString(), XAxis[res[0]],
                                                 YAxis[res[1]] + SS[res[0], res[1]] * (YAxis[res[1] + 1] -
 YAxis[res[1]]),
                                                 Elevation/*, res[0] * 100 + res[1]*/));
                             SS[res[0], res[1]] = 5;
                            break;
                         case 5:
                            SS[res[0] + 1, res[1]] = 5;
                            tempDataPoints.Add(new DataPoint((res[0] + 1) * 100 + res[1],
                                "等值点
" + ((1 + res[0]) * 100 + res[1]).ToString(), XAxis[res[0] + 1],
                                                YAxis[res[1]] + SS[res[0] + 1, res[1]] * (YAxis[res[1] +
1] - YAxis[res[1]]),
                                                Elevation/*, (res[0] + 1) * 100 + res[1]*/));
                            break;
                        case 7:
                            tempDataPoints.Add(new DataPoint(-res[0] * 100 - res[1], "等值点
" + (-res[0] * 100 - res[1]).ToString(),
                                                XAxis[res[0]] + HH[res[0], res[1]] * (XAxis[res[0] + 1] -
 XAxis[res[0]]),
                                                XAxis[res[1]], Elevation/*, -res[0] * 100 - res[1]*/));
                            HH[res[0], res[1]] = 5;
                            break;
                        default:
                            break;
                    }
                }
            }
            if (tempDataPoints.Count > 0) { return new ContourPolyline(tempDataPoints.ToArray());
 }
            else return null;
        }
        //追踪
        private int[] Track(int i, int j, int inDirc)
        {
            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;
7.
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].Item
    2;
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)
```

```
        {
            List<Tuple<DataPoint, double>>[] PointPositionDistanceList
                = new List<Tuple<DataPoint, double>>[sectorNums];
            for (int i = 0; i < sectorNums; i++)
                PointPositionDistanceList[i] = new List<Tuple<DataPoint, double>>();
            double sectorArc = 360.0 / sectorNums;
            foreach (var point in mPointSet.PointList)
            {
                double alpha = point.GetPosition(x, y);
                PointPositionDistanceList[(int)(alpha / sectorArc)].Add(new Tuple<DataPoint, doubl
e>(point, point.GetDistanceP2(x, y)));
            }
            List<Tuple<DataPoint, double>> SelectedPointList = new List<Tuple<DataPoint, double>>(
);
            for (int i = 0; i < sectorNums; i++)
            {
                if (PointPositionDistanceList[i].Count != 0)
                {
                    PointPositionDistanceList[i].Sort((t1, t2) => t1.Item2.CompareTo(t2.Item2));
                    SelectedPointList.Add(PointPositionDistanceList[i][0]);
                }
            }
            List<double> WeightList = new List<double>();
            int SelectPointCount = SelectedPointList.Count;
            if (SelectPointCount != 0)
            {
                double sProduct = 1; //总的乘积
                double sDenominator = 0; //分母
                double sResult = 0; //结果
                for (int j = 0; j < SelectPointCount; j++)
                    sProduct *= SelectedPointList[j].Item2;
                for (int j = 0; j < SelectPointCount; j++)
                {
                    WeightList.Add(sProduct / SelectedPointList[j].Item2);
                    sDenominator += sProduct / SelectedPointList[j].Item2;
                }
                for (int j = 0; j < SelectPointCount; j++)
                    sResult += WeightList[j] * SelectedPointList[j].Item1.Value / sDenominator;
                return sResult;
            }
            else throw new Exception("CalculateValueBy 按方位加权平均法:没有选中的点");
        }
    }
}
```

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;
13.
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
26.
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.         }
38.
39.         public PointSet PointSet { get; private set; }
40.         public UserOperationType UserOperation { get; private set; }
41.
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;
53.
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;
58.
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.         }
85.
86.         private void AgisControl_Load(object sender, EventArgs e){}
87.
88.         private void AgisControl_Paint(object sender, PaintEventArgs e){}
89.
90.         public void SetUserOperationToDisplayInGrid()
```

```
91.         {this.UserOperation = UserOperationType.DisplayInGrid;}
92.
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)));}
102.
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));}
117.
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)GetScreenL
     ocY(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.          }
129.
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(poi
    nt.Y)));}
140.             return tempPointList.ToArray();
141.         }
142.
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.         }
148.
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.         }
157.
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.         }
169.
170.         public double GetRegionArea()
171.         { return (this.MBR_Origin.MaxX - this.MBR_Origin.MinX) *
172.                 (this.MBR_Origin.MaxY - this.MBR_Origin.MinY);}
173.
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) };}
183.
184.         public double GetRealWorldLocX(float x)
185.         {return x / this.Zoom + this.OffsetX; }
186.
187.         public double GetRealWorldLocY(float y)
188.         {return (this.Height - y) / this.Zoom + this.OffsetY;}
189.
190.         private void AgisControl_Resize(object sender, EventArgs e)
191.         {this.Refresh(); }
192.
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.M
    BR.MinY),
200.                     this.Width / (this.PointSet.MBR.MaxX - this.PointSet.MBR.MinX)) / this.FrameS
    caling;
201.                 this.Zoom = this.ZoomScale;
202.             }
203.             this.Refresh();
204.         }
205.
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.         }
224.
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.CurMou
    seLocation.Y);
232.                 this.OffsetX += -curLoc[0] + lastLoc[0];
233.                 this.OffsetY += -curLoc[1] + lastLoc[1];
234.                 this.Refresh();
235.                 this.CurMouseLocation = mouseLoc;
236.             }
237.         }
238.
239.         private void AgisControl_MouseDown(object sender, MouseEventArgs e)
240.         {this.IsPanning = true;
241.             this.CurMouseLocation = e.Location;}
242.
243.         private void AgisControl_MouseUp(object sender, MouseEventArgs e)
244.         {this.IsPanning = false;
245.             this.CurMouseLocation = e.Location; }
246.
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;
17.
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;
28.
29.         public UserOperationType UserOperation;
30.
31.         // -- 数据点
32.         private float PointHalfWidth = 5;
33.         public Brush PointIconBrush = new SolidBrush(Color.Red);
34.
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>();
50.
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;
57.
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;
66.
67.         // -- Tin 相关
68.         public bool ShowTin = false;
69.         public Edge[] TinEdges = null;
70.         public Pen TinPen = new Pen(Color.Blue, 1.0f);
71.
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);
77.
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);
86.
87.         // -- 拓扑表
88.         private TopoPolygonSet mTopoPolygonSet;
89.         private TopoPolylineSet mTopoPolylineSet;
90.         private TopoPointSet mTopoPointSet;
```

```
91.
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;
98.
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.         }
107.
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.         }
122.
123.         private void agisControl_Resize(object sender, EventArgs e)
124.         {agisControl.Refresh();}
125.
126.         private void agisControl_MarginChanged(object sender, EventArgs e)
127.         {agisControl.Refresh();}
128.
129.         private void agisControl_Paint(object sender, PaintEventArgs e)
130.         {
131.
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, p
    f.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 / GridDivisionCoun
    t_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 / EachGrid
    DivisionCount_X) * (width / GridDivisionCount_X), MinPointXY.Y,
199.                             MinPointXY.X + (i + ii * 1.0f / EachGridDivisionCount_X) * (width / Gr
    idDivisionCount_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 / G
    ridDivisionCount_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)
    ;
```

```
                        g.DrawEllipse(this.GridSelectedPointPen, (float)sScreenSelectPointX - SelectPixelThreshold,
                            (float)sScreenSelectPointY - SelectPixelThreshold,
                            SelectPixelThreshold * 2, SelectPixelThreshold * 2);
                    }
                }
                //绘制等值线
                if (ShowContourLine == true && GridContourList != null)
                {
                    for (int i = 0; i < GridContourList.Length; i++)
                    { PointF[] screenLine = agisControl.GetScreenEdge(GridContourList[i]);
                        Graphics g = e.Graphics;
                        g.DrawLine(GridContourLinePen, screenLine[0], screenLine[1]);}
                }
                if (ShowContourLine == true && GridContourPolylineList != null)
                {
                    for (int i = 0; i < GridContourPolylineList.Length; i++)
                    {
                        PointF[] screenLine = agisControl.GetScreenEdge(GridContourPolylineList[i]);
                        Graphics g = e.Graphics;
                        float tension = 0f;
                        if (ContourLineUseSpline)
                            tension = 0.25f /* (float)(agisControl.ZoomScale / agisControl.Zoom)*/;
                        if (screenLine.Length > 1)
                            g.DrawCurve(GridContourLinePen, screenLine, tension);
                    }
                }
            }
            if (this.UserOperation == UserOperationType.DisplayInTIN)
            {
                //绘制三角网
                if (ShowTin == true && TinEdges != null)
                {for (int i = 0; i < TinEdges.Length; i++)
                    {PointF[] screenLine = agisControl.GetScreenEdge(TinEdges[i]);
                        Graphics g = e.Graphics;
                        g.DrawLine(TinPen, screenLine[0], screenLine[1]);} }
                //绘制等高线
                if (ShowContourLine == true && TinContourLineList != null)
                {
                    for (int i = 0; i < TinContourLineList.Length; i++)
                    {PointF[] screenLine = agisControl.GetScreenEdge(TinContourLineList[i]);
                        Graphics g = e.Graphics;
                        g.DrawLine(TinContourLinePen, screenLine[0], screenLine[1]);}
                }
            }
```

```
259.
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.         }
269.
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.         }
277.
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.         }
290.
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.         }
299.
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)Gri
    dDivisionScreenRefresh();}
317.         }
318.
319.         private void agisControl_MouseWheel(object sender, MouseEventArgs e)
320.         {
321.             if (this.UserOperation == UserOperationType.DisplayInGrid)
322.             {if (this.IsGridVisible && this.IsQueryIntersection)GridDivisionScreenRefresh();}
323.         }
324.
325.         private void 距离平方倒数法ToolStripMenuItem_Click(object sender, EventArgs e)
326.         {
327.             //this.agisControl.SetUserOperationToDisplayInGrid();
328.
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.         }
343.
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;
```

```
            int tempPara = agisControl.按方位加权平均法 SectorNum;
            if (tempPara < 0)
                tempPara = Math.Max(agisControl.PointSet.PointList.Count / 8, 1);
            GridIntParaForm form = new GridIntParaForm("每个象限等分的 no 个扇区", tempPara, 1,
                Math.Max(agisControl.PointSet.PointList.Count / 4, 1));
            if (form.ShowDialog(this) == DialogResult.OK)
            {
                this.UserOperation = UserOperationType.DisplayInGrid;
                this.agisControl.GridIntMethod = Mehtod.GridInterpolationMehtod.按方位加权平均
    法;
                按方位加权平均法 ToolStripMenuItem.Checked = true;
                距离平方倒数法 ToolStripMenuItem.Checked = false;
                agisControl.按方位加权平均法 SectorNum = form.ParaValue * 4;
                MessageBox.Show("参数设置成功！", "提示");
            }
        }

        private void 加密网格 toolStripMenuItem_Click(object sender, EventArgs e)
        {
            if (this.UserOperation != UserOperationType.DisplayInGrid)
            {MessageBox.Show("请先生成格网！", "提示");return;}
            if (this.IsGridVisible == false)
            {
                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)
                    != DialogResult.OK)
                { this.IsGridVisible = true;
                    this.显示隐藏格网 ToolStripMenuItem.Checked = true;}
                else
                    return;
            }
            GenerateSubGridForm form = new GenerateSubGridForm(this.EachGridDivisionCount_X, this
    .EachGridDivisionCount_Y);
            if (form.ShowDialog(this) == DialogResult.OK)
            {this.EachGridDivisionCount_X = form.Division_X;
                this.EachGridDivisionCount_Y = form.Division_Y;}
            GridDivisionRefresh();
            this.agisControl.Refresh();
        }

        //每次格网重新划分时进行调用
        private void GridDivisionRefresh()
        {
            int TotalSegmentNum_X = GridDivisionCount_X * EachGridDivisionCount_X;
            int TotalSegmentNum_Y = GridDivisionCount_Y * EachGridDivisionCount_Y;
            double MbrMinX = agisControl.MBR_Origin.MinX;
            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.         }
404.
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.         }
422.
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.         }
434.
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] - G
    rid_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] - G
    rid_AxisY[j]),
489.                                         tempElevation, ((i + 1) * 1000 + j) * 1000 + (int)tempEle
    vation));}
490.                                 if (tempPointList.Count < 2)//无等值线
491.                                     continue;
492.                                 else if (tempPointList.Count < 4)
493.                                 {tempGridContourLineList.Add(new Edge(tempPointList[0], tempPoint
    List[1]));}
494.                                 else
495.                                 {tempGridContourLineList.Add(new Edge(tempPointList[0], tempPoint
    List[1]));
496.                                     tempGridContourLineList.Add(new Edge(tempPointList[2], tempPo
    intList[3]));}
497.                             }
498.                         }
499.                         tempContourPolyline = EdgeSet.TopologyGenerateContourPolylineSet(tempGrid
    ContourLineList.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.ToArra
    y();
508.                 }
509.                 //GridContourLinePen.DashStyle = System.Drawing.Drawing2D.DashStyle.DashDot;
510.                 agisControl.Refresh();
511.             }
512.         }
513.
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], tempG
    ridAxisY[j]);
527.             this.GridValueMatrix = GridRealLoc;
528.             return GridRealLoc;
529.         }
530.
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] - GridVal
    ueMatrix[i, j]);
540.                     tempHH[i, j] = (r <= 1 && r >= 0) ? r : 3;}
541.             }
542.             this.HH = tempHH;
543.             return tempHH;
544.         }
545.
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] - GridVal
    ueMatrix[i, j]);
555.                     tempSS[i, j] = (r <= 1 && r >= 0) ? r : 3;}
556.             }
557.             this.SS = tempSS;
558.             return tempSS;
559.         }
560.
561.         private void 设置ToolStripMenuItem_Click(object sender, EventArgs e){}
562.
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.             }
588.
589.         }
590.
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 * sett
    ingForm.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.         }
619.
620.         private void 设置 ToolStripMenuItem1_Click(object sender, EventArgs e){}
621.
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.ToAr
    ray());
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.         }
637.
638.         private void 可视化 ToolStripMenuItem_Click(object sender, EventArgs e)
639.         {
640.             this.ShowTopology = (可视化 ToolStripMenuItem.Checked == true);
```

```
            this.拓扑点 ToolStripMenuItem.Checked = this.ShowTopology;
            this.拓扑边 ToolStripMenuItem.Checked = this.ShowTopology;
            this.拓扑多边形 ToolStripMenuItem.Checked = this.ShowTopology;
            this.agisControl.Refresh();
        }

        private void 查询 ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            QueryPolygonInfoForm queryForm = new QueryPolygonInfoForm(this.mTopoPolygonSet);
            if (queryForm.ShowDialog(this) == DialogResult.OK) {}
        }

        private void 导出拓扑关系表 ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            SaveTopologyTableForm saveForm =
                new SaveTopologyTableForm(this.mTopoPointSet, this.mTopoPolylineSet, this.mTopoPo
lygonSet);
            if (saveForm.ShowDialog(this) == DialogResult.OK){}
        }

        private void 程序信息 ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show(this, @"
(1)读取文件
    “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。
(2)基本操作
    漫游：鼠标左键拖动。
    放大/缩小：鼠标滚轮 上/下 滚动。
    全局：单击鼠标中键，缩放至原始范围。
(3)选择插值算法
    “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。
(4)生成格网模型
    “格网模型” — “生成格网”，选择 X,Y 方向分位数生成网格。
    “格网模型” — “加密格网”，在原有格网上加密,需要已有格网。
    “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。
    “格网模型” — “设置” — “显示/隐藏格网”，设置格网可见性。
    “格网模型” — “设置” — “清除格网”，清除已建立的格网模型。
(5)TIN 模型
    “TIN 模型” — “逐点插入法”，生成 TIN 模型并显示。
    “TIN 模型” — “设置” — “显示/隐藏 TIN”，设置 TIN 可见性。
    “TIN 模型” — “设置” — “清除 TIN”，清除已建立的 TIN 模型。
(6)等值线
    等值线的最大值，最小值，间距由对话框设定。
    “格网模型” — “生成等值线”，根据格网模型生成等值线。
    “格网模型” — “生成等值线” — “平滑”，是否平滑生成的等值线。
    “TIN 模型” — “生成等值线”，根据 TIN 模型生成等值线。
(7)拓扑关系
```

```
    “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
    “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
        点：结点为蓝色方格，中间点为绿色方格
        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）
        面：随机颜色（每次刷新颜色不同，故刷新有延迟）
    “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
    “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。
(8)其他
    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。
    如果出现问题，重启程序试试。
", "程序信息", MessageBoxButtons.OK);
        }

        private void agisControl_MouseHover(object sender, EventArgs e)
        {}

        private void 显示隐藏格网 ToolStripMenuItem_Click(object sender, EventArgs e)
        {this.IsGridVisible = (显示隐藏格网 ToolStripMenuItem.Checked == true);
            agisControl.Refresh();}

        private void 生成格网 ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.IsGridVisible = true;
            this.显示隐藏格网 ToolStripMenuItem.Checked = true;
            this.UserOperation = UserOperationType.DisplayInGrid;
            GenerateGridForm form = new GenerateGridForm(this.GridDivisionCount_X, this.GridDivis
ionCount_Y);
            if (form.ShowDialog(this) == DialogResult.OK)
            {
                this.GridDivisionCount_X = form.DivisionX;
                this.GridDivisionCount_Y = form.DivisionY;
                GridDivisionRefresh();
                this.agisControl.Refresh();
            }
        }

        private void agisControl_MouseClick(object sender, MouseEventArgs e)
        {MouseLocation = e.Location;}

        private void agisControl_MouseDown(object sender, MouseEventArgs e){}

        private void agisControl_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            MouseLocation = e.Location;
            GridDivisionScreenRefresh();
            if (this.UserOperation != UserOperationType.DisplayInGrid
                || 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.SelectPixelThres
    hold)
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.SelectPixelThre
    shold)
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).ToStrin
    g("0.000"),
766.                         "插值方法: " + MethodName, Para
```

```
                        ), "属性查询");
                }
            }
            if (e.Clicks == 2 && this.IsQueryTopoPolygon == true && this.ShowTopology == true &&
this.ShowTopoPolygon == true)
            {
                TopoPoint clickLoc = new TopoPoint(agisControl.GetRealWorldLocX(e.X), agisControl
.GetRealWorldLocX(e.Y), 0, false);
                this.SelectedTopoPolygon = this.mTopoPolygonSet.GetClickPointInsidePolygon(clickL
oc);
                this.agisControl.Refresh();
                if (SelectedTopoPolygon != null)
                    MessageBox.Show(string.Format("PID:{0}\r\n 弧段数:{1}\r\n 周长:{2}\r\n 面
积:{3}",
                        SelectedTopoPolygon.PID, SelectedTopoPolygon.TopologyArcs.Count,
                        SelectedTopoPolygon.GetPerimeter().ToString("0.00"),
                        SelectedTopoPolygon.GetArea().ToString("0.00")), "多边形信息");
            }
        }

        private void agisControl_Load(object sender, EventArgs e) { }

        private void 显示隐藏TINToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.ShowTin = (显示隐藏TINToolStripMenuItem.Checked == true);
            agisControl.Refresh();
        }

        private void 生成等值线ToolStripMenuItem1_CheckedChanged(object sender, EventArgs e)
        { }

        private void Set等值线可见性(bool isVisable)
        {
            this.ShowContourLine = isVisable;
            生成等值线ToolStripMenuItem1.Checked = isVisable;
            生成等值线ToolStripMenuItem.Checked = isVisable;
            agisControl.Refresh();
        }

        public void GenerateTopologyRelatation(ContourPolyline[] contourLines)
        {
            double BottomY = agisControl.MBR_Origin.MinY;
            double TopY = agisControl.MBR_Origin.MaxY;
            double LeftX = agisControl.MBR_Origin.MinX;
            double RightX = agisControl.MBR_Origin.MaxX;
            double CenterX = (LeftX + RightX) / 2;
            double CenterY = (BottomY + TopY) / 2;
```

```
            DataPoint rectP0 = new DataPoint(-10000, "Rect0", CenterX, CenterY, this.agisControl.GetGridInterpolationValue(CenterX, CenterY));
            DataPoint rectP1 = new DataPoint(-10001, "Rect1", CenterX, TopY, this.agisControl.GetGridInterpolationValue(CenterX, TopY));
            DataPoint rectP2 = new DataPoint(-10002, "Rect2", RightX, TopY, this.agisControl.GetGridInterpolationValue(RightX, TopY));
            DataPoint rectP3 = new DataPoint(-10003, "Rect3", RightX, CenterY, this.agisControl.GetGridInterpolationValue(RightX, CenterY));
            DataPoint rectP4 = new DataPoint(-10004, "Rect4", RightX, BottomY, this.agisControl.GetGridInterpolationValue(RightX, BottomY));
            DataPoint rectP5 = new DataPoint(-10005, "Rect5", CenterX, BottomY, this.agisControl.GetGridInterpolationValue(CenterX, BottomY));
            DataPoint rectP6 = new DataPoint(-10006, "Rect6", LeftX, BottomY, this.agisControl.GetGridInterpolationValue(LeftX, BottomY));
            DataPoint rectP7 = new DataPoint(-10007, "Rect7", LeftX, CenterY, this.agisControl.GetGridInterpolationValue(LeftX, CenterY));
            DataPoint rectP8 = new DataPoint(-10008, "Rect8", LeftX, TopY, this.agisControl.GetGridInterpolationValue(LeftX, TopY));
            //给定的边
            List<Edge> GivenEdges = new List<Edge>();
            //矩形边缘
            GivenEdges.Add(new Edge(rectP1, rectP2));
            GivenEdges.Add(new Edge(rectP2, rectP3));
            GivenEdges.Add(new Edge(rectP3, rectP4));
            GivenEdges.Add(new Edge(rectP4, rectP5));
            GivenEdges.Add(new Edge(rectP5, rectP6));
            GivenEdges.Add(new Edge(rectP6, rectP7));
            GivenEdges.Add(new Edge(rectP7, rectP8));
            GivenEdges.Add(new Edge(rectP8, rectP1));
            //矩形中心
            GivenEdges.Add(new Edge(rectP0, rectP1));
            GivenEdges.Add(new Edge(rectP0, rectP2));
            GivenEdges.Add(new Edge(rectP0, rectP3));
            GivenEdges.Add(new Edge(rectP0, rectP4));
            GivenEdges.Add(new Edge(rectP0, rectP5));
            GivenEdges.Add(new Edge(rectP0, rectP6));
            GivenEdges.Add(new Edge(rectP0, rectP7));
            GivenEdges.Add(new Edge(rectP0, rectP8));
            //产生的结果
            List<ContourPolyline> resultPolylineList = new List<ContourPolyline>();
            resultPolylineList.AddRange(contourLines);
            List<Edge> resultEdgeList = new List<Edge>();
            //resultEdgeList.AddRange(GivenEdges.ToArray());
            for (int i = 0; i < GivenEdges.Count; i++)
            {
                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.         }
856.
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.         }
869.
870.         private void 拓扑点ToolStripMenuItem_Click(object sender, EventArgs e)
871.         { 拓扑点ToolStripMenuItem.Checked = (拓扑点ToolStripMenuItem.Checked == false); }
872.
873.         private void 拓扑边ToolStripMenuItem_Click(object sender, EventArgs e)
874.         { 拓扑边ToolStripMenuItem.Checked = (拓扑边ToolStripMenuItem.Checked == false); }
875.
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); }
878.
879.         private void 拓扑点ToolStripMenuItem_CheckedChanged(object sender, EventArgs e)
880.         { this.ShowTopoPoint = 拓扑点ToolStripMenuItem.Checked; this.Refresh(); }
881.
882.         private void 拓扑边ToolStripMenuItem_CheckedChanged(object sender, EventArgs e)
883.         { this.ShowTopoPolyline = 拓扑边ToolStripMenuItem.Checked; this.Refresh(); }
884.
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(); }
887.
888.         private void 查询ToolStripMenuItem_CheckedChanged(object sender, EventArgs e)
889.         { this.IsQueryTopoPolygon = 查询ToolStripMenuItem.Checked; }
890.
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.         }
902.
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.         }
912.
913.         private void 清楚 TINToolStripMenuItem_Click(object sender, EventArgs e)
914.         { this.ShowTin = false; }
915.
916.         private void 平滑 ToolStripMenuItem_Click(object sender, EventArgs e)
917.         { this.ContourLineUseSpline = (平滑 ToolStripMenuItem.Checked == true); }
918.     }
919. }
```

END