

高级地理信息系统程序作业

姓名：周恩波

学号：1501210204

2016年6月

目录

[1.作业要求和完成情况 3](#_Toc454463360)

[2.结果展示 3](#_Toc454463361)

[2.1打开数据 3](#_Toc454463362)

[2.2生成格网 4](#_Toc454463363)

[2.3加密格网 4](#_Toc454463364)

[2.4查询格网交点属性值 5](#_Toc454463365)

[2.5根据格网生成等值线 6](#_Toc454463366)

[2.6生成点集凸包 6](#_Toc454463367)

[2.7生成TIN 7](#_Toc454463368)

[2.8由TIN生成等值线 8](#_Toc454463369)

[2.9生成拓扑 8](#_Toc454463370)

[2.10查询多边形信息 9](#_Toc454463371)

[2.11查看并保存拓扑关系表 10](#_Toc454463372)

[3.代码展示 11](#_Toc454463373)

[3.1距离平方倒数法插值 11](#_Toc454463374)

[3.2按方位加权平均法插值 11](#_Toc454463375)

[3.3格网自动生成等值线 13](#_Toc454463376)

[3.4生成点集凸包 24](#_Toc454463377)

[3.5生成点集TIN模型 25](#_Toc454463378)

[3.6 TIN自动生成等值线 30](#_Toc454463379)

[3.7自动生成拓扑 35](#_Toc454463380)

[3.8计算多边形周长和面积 85](#_Toc454463381)

# 1.作业要求和完成情况

本次程序设计作业的具体要求和完成情况见表1-1，其中功能部分红色表示作业要求之外的功能。

**表1-1 作业要求及完成情况**

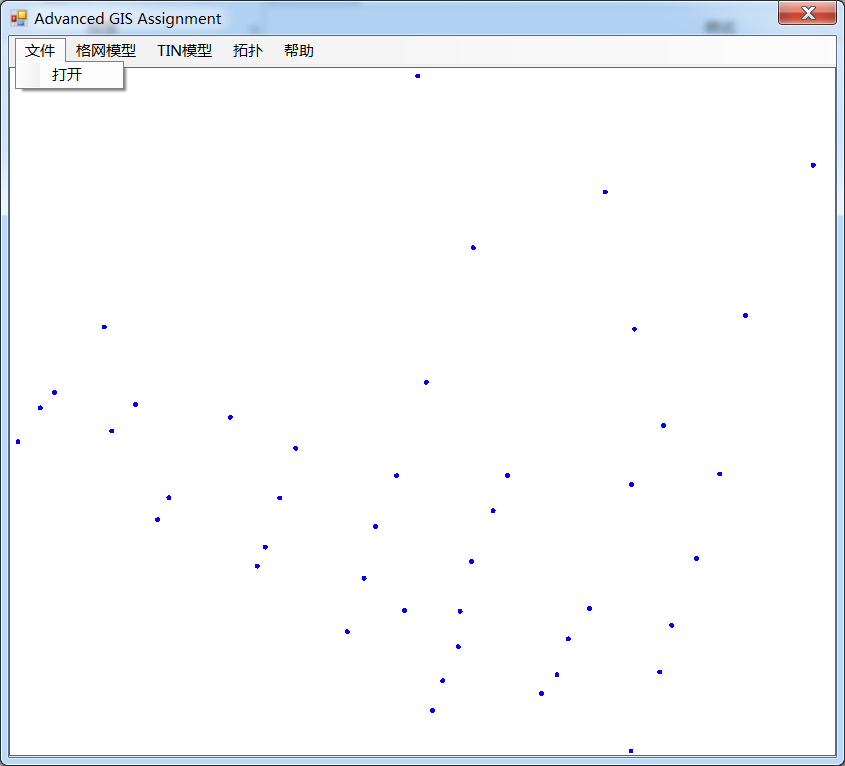
|  |  |  |
| --- | --- | --- |
| 项目 | 功能 | 完成情况 |
| 格网模型 | 格网生成（两种插值算法） | ✓ |
| 格网加密 | ✓ |
| 显示或者不显示格网 | ✓ |
| 查询交点属性值 | ✓ |
| 自动生成等值线（光滑和不光滑两种） | ✓ |
| TIN模型 | 生成点集凸包 | ✓ |
| 生成TIN | ✓ |
| 自动生成等值线（光滑和不光滑两种） | ✓ |
| 拓扑 | 自动生成 | ✓ |
| 多边形查询 | ✓ |
| 拓扑表保存 | ✓ |

本次作业采用C#编写，开发平台为Visual Studio 2013，运行环境为.Net Frame 3.5及以上，作业所要求的功能全部实现，经测试符合要求。

# 2.结果展示

## 2.1打开数据

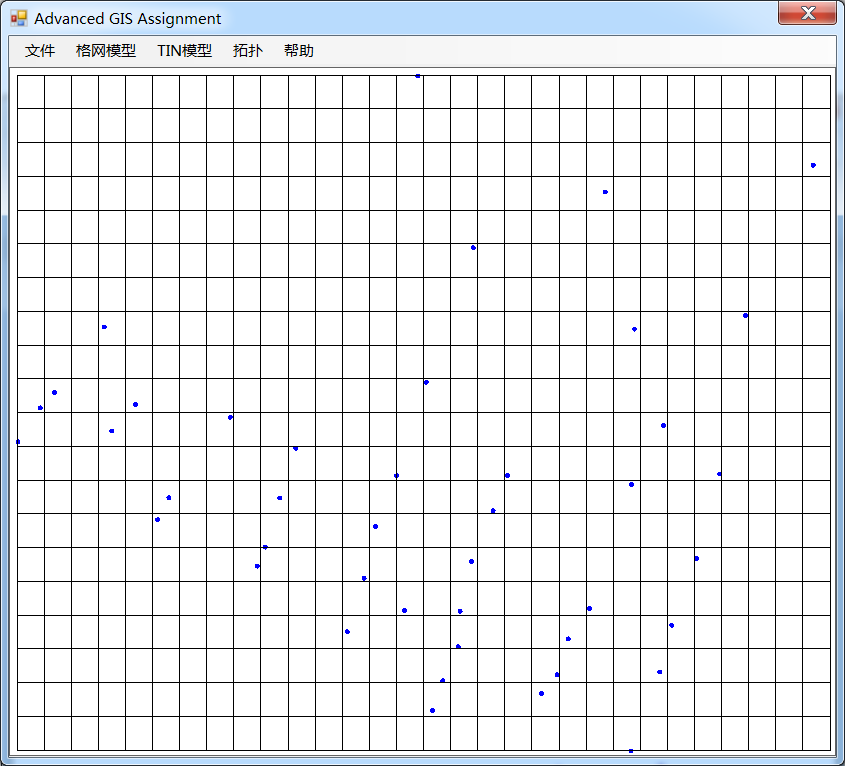
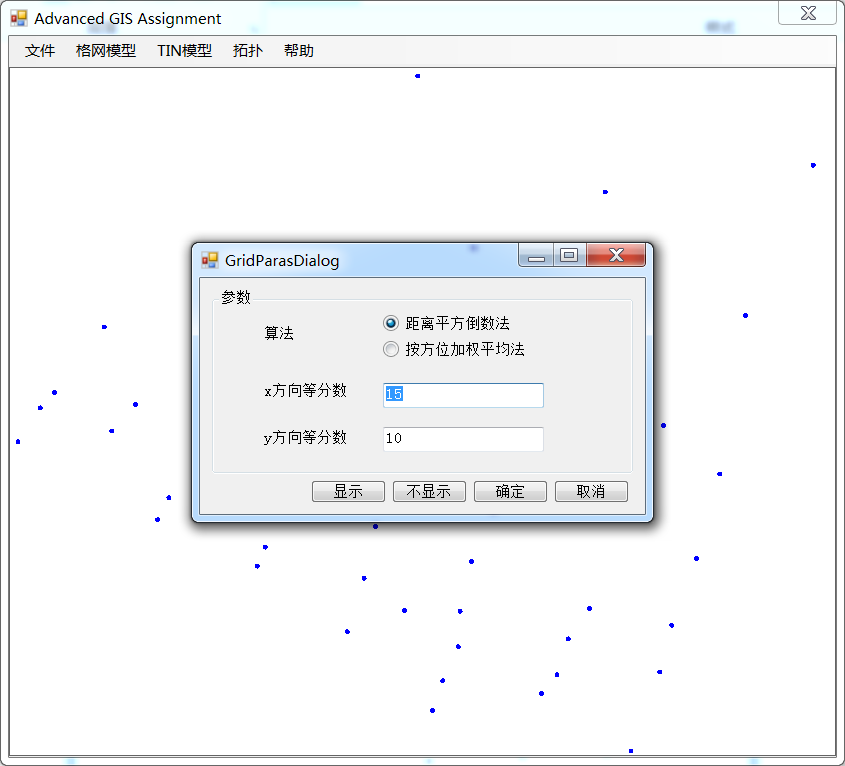
实验数据打开情况如图2.1所示。



**图2.1 实验数据**

## 2.2生成格网

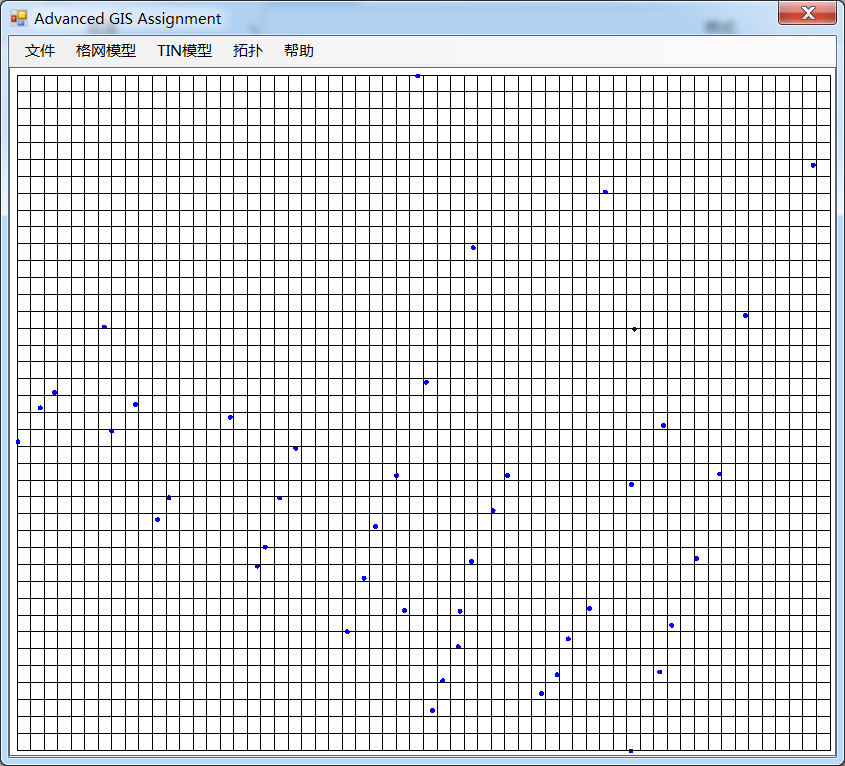
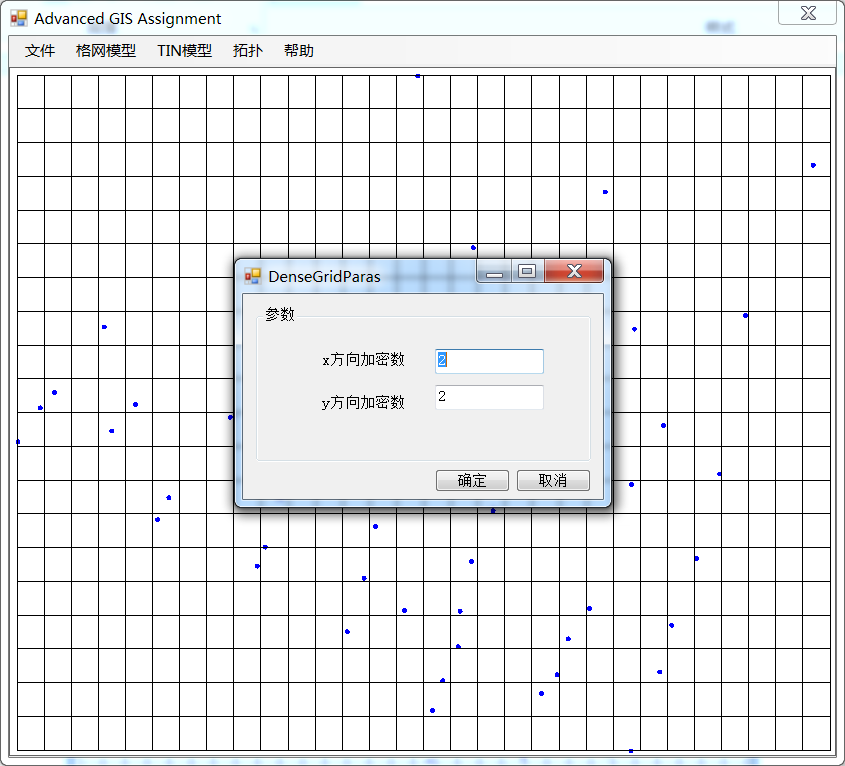
生成格网的界面和结果如图2.2所示。



**图2.2 生成格网**

## 2.3加密格网

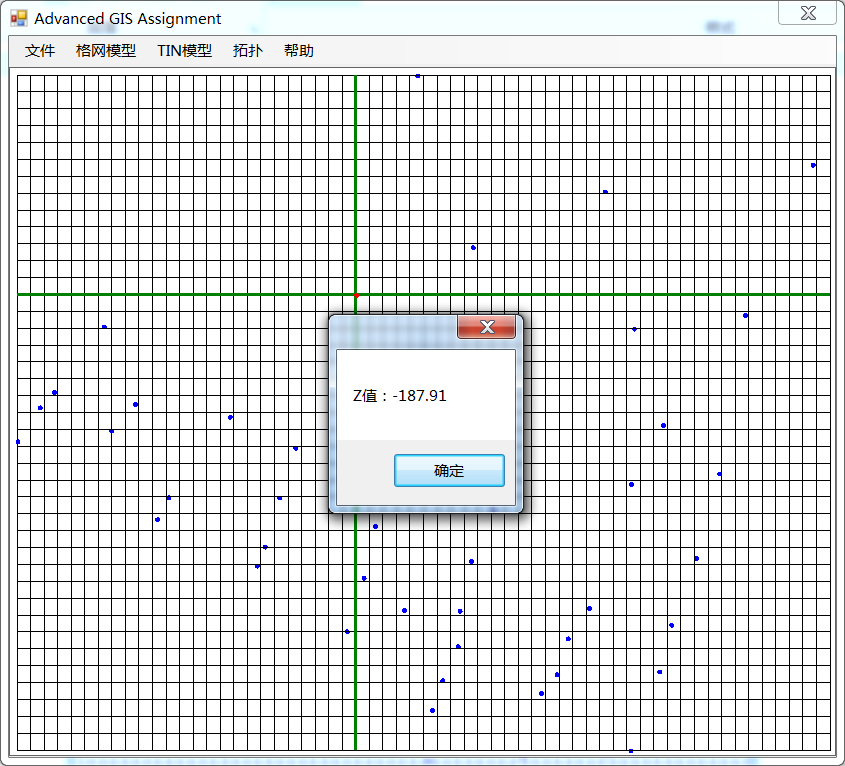
加密格网的界面和结果如图2.3所示。



**图2.3 加密格网**

## 2.4查询格网交点属性值

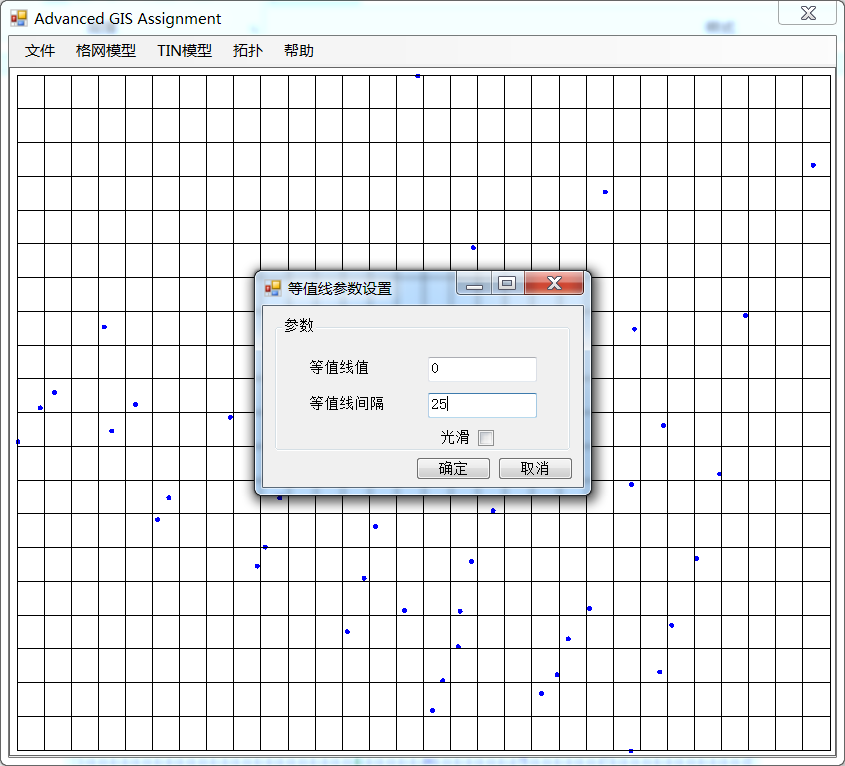
点击格网交点，查询格网交点的属性，结果如图2.4所示。



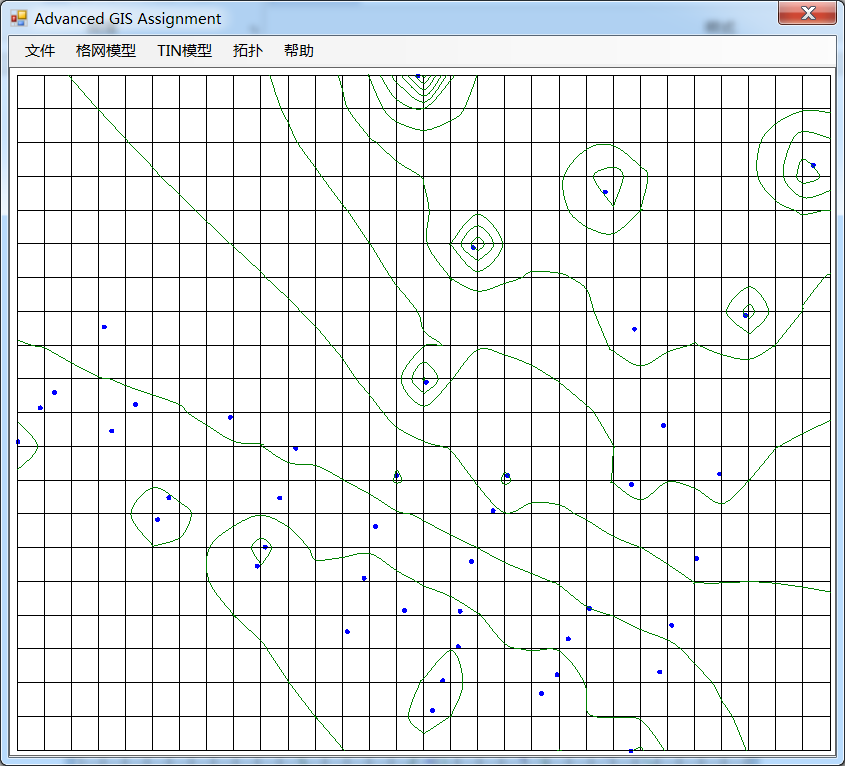
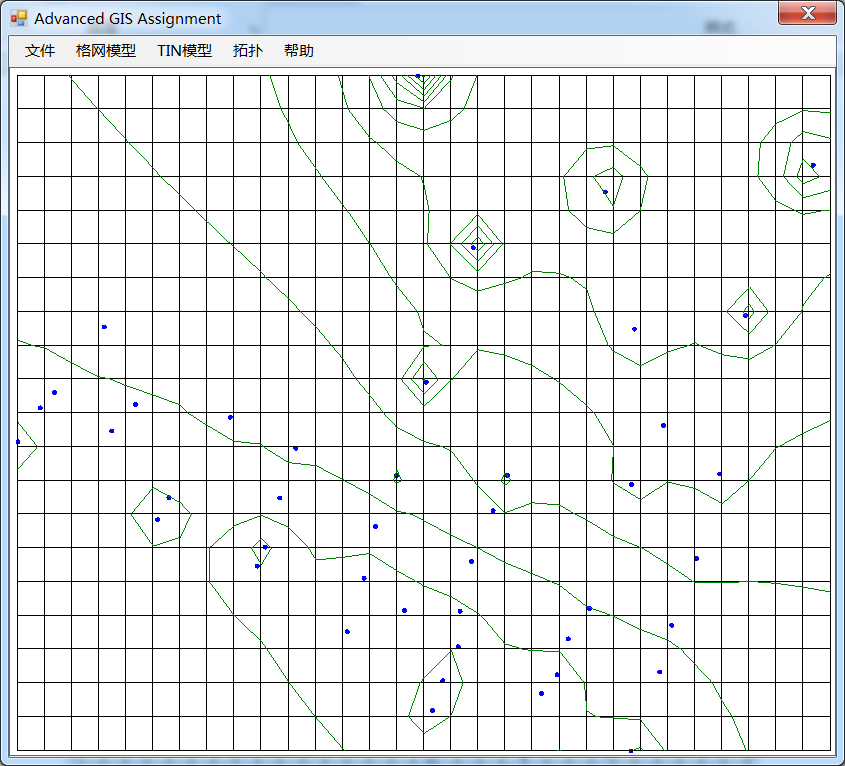
**图2.4 查询交点属性**

## 2.5根据格网生成等值线

根据格网生成等值线的参数设置和生成结果如图2.5和2.6所示。



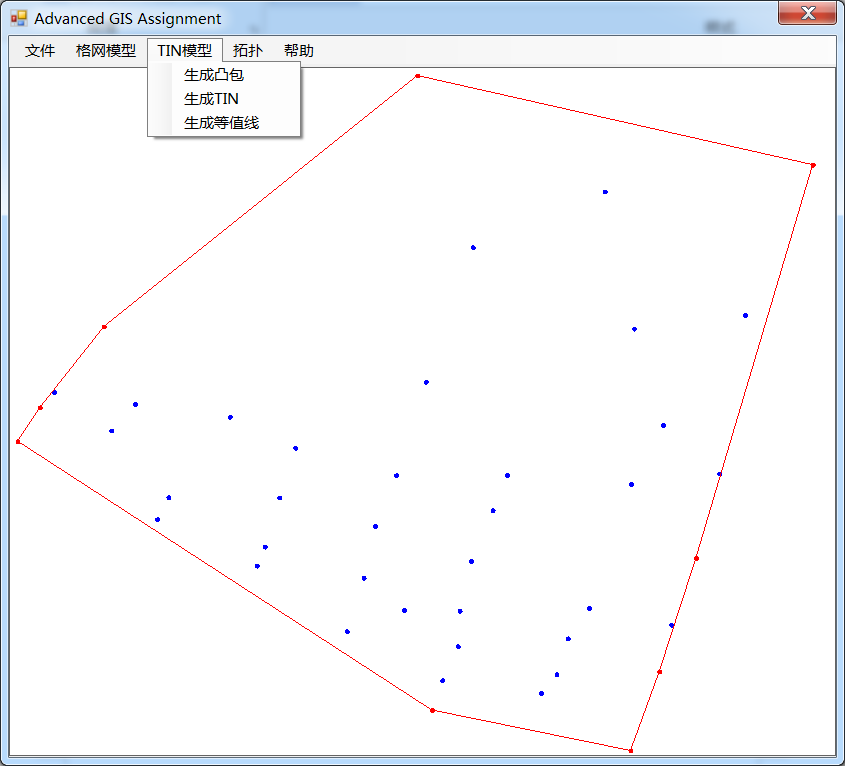
**2.5由格网生成等值线**



**2.6格网等值线生成结果（光滑和不光滑）**

## 2.6生成点集凸包

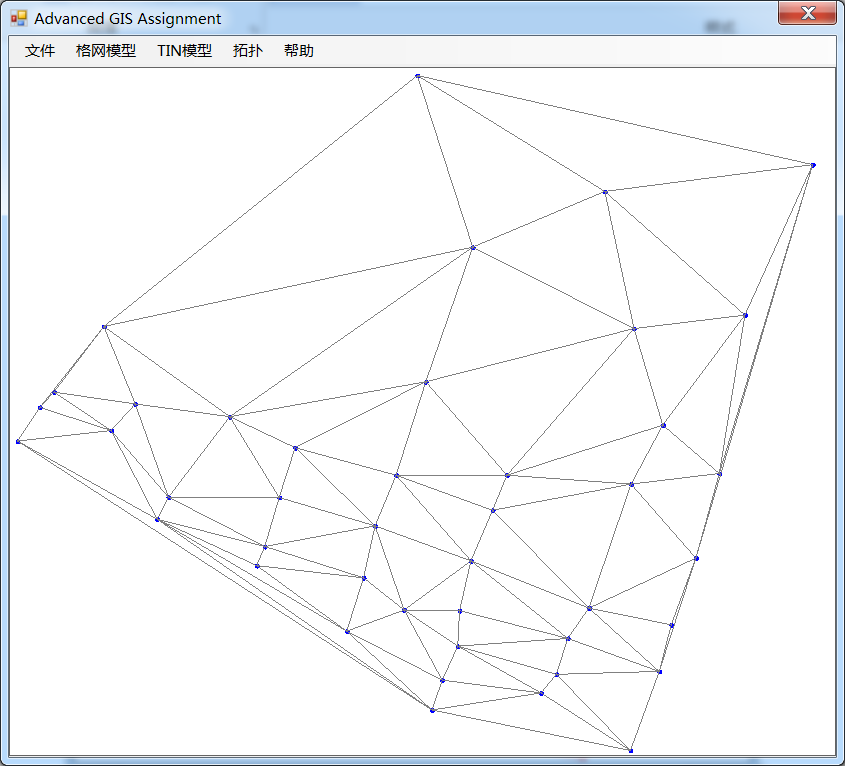
点集的凸包生成结果如图2.7所示。



**2.7 点集凸包生成结果**

## 2.7生成TIN

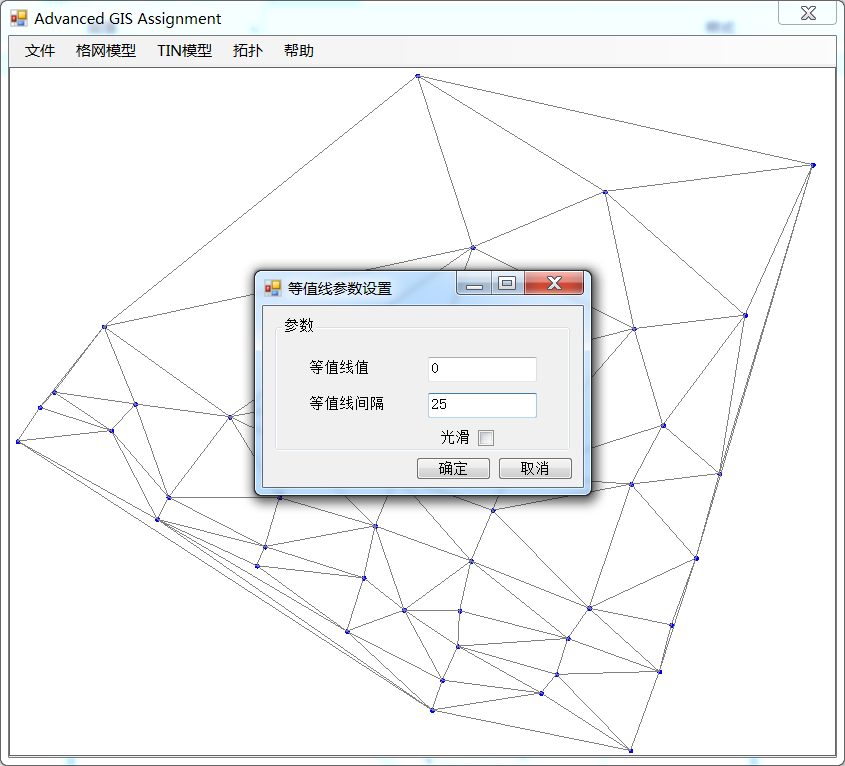
点集的TIN生成结果如图2.8所示。



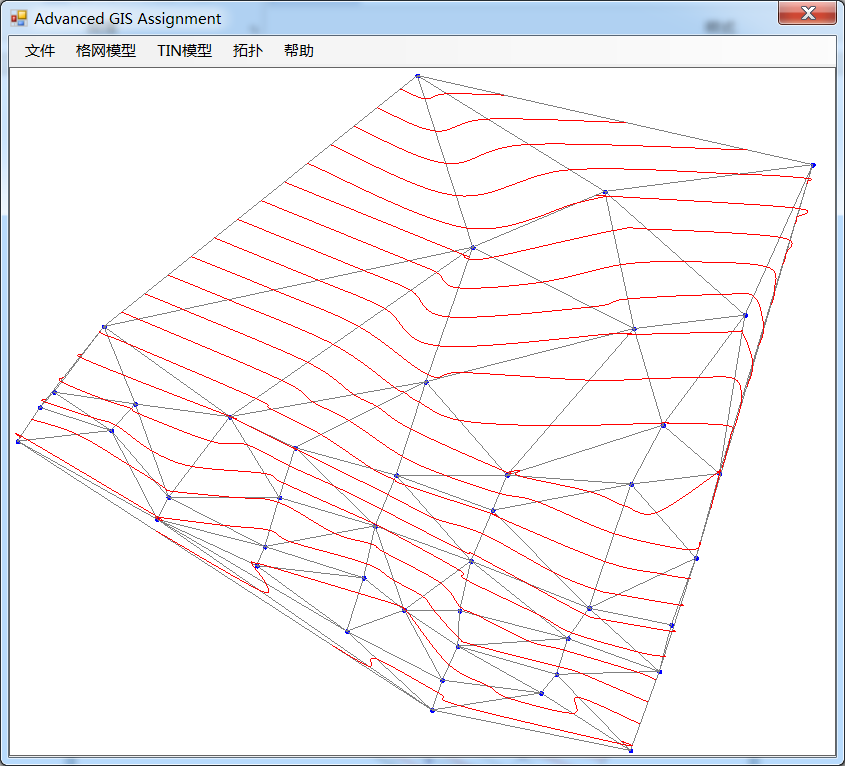
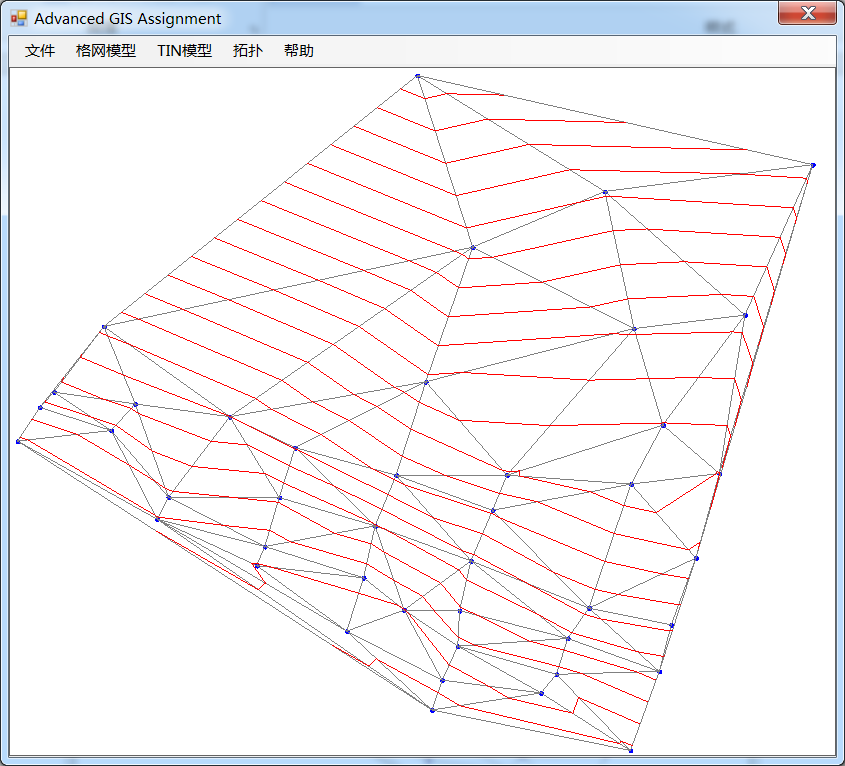
**2.8 TIN生成结果**

## 2.8由TIN生成等值线

根据TIN模型生成等值线的参数设置和生成结果如图2.9和2.10所示。



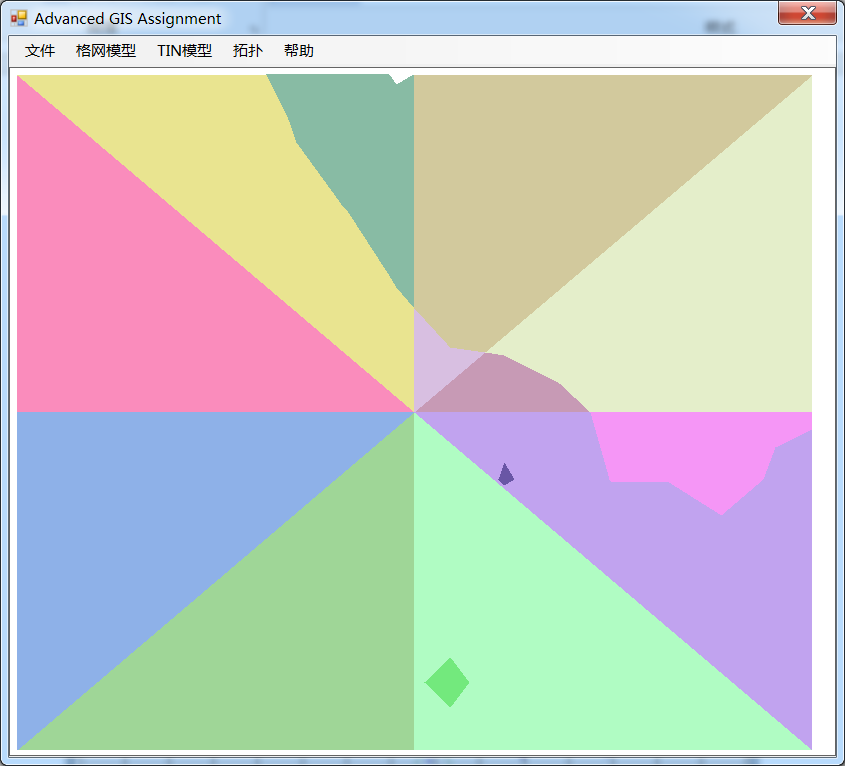
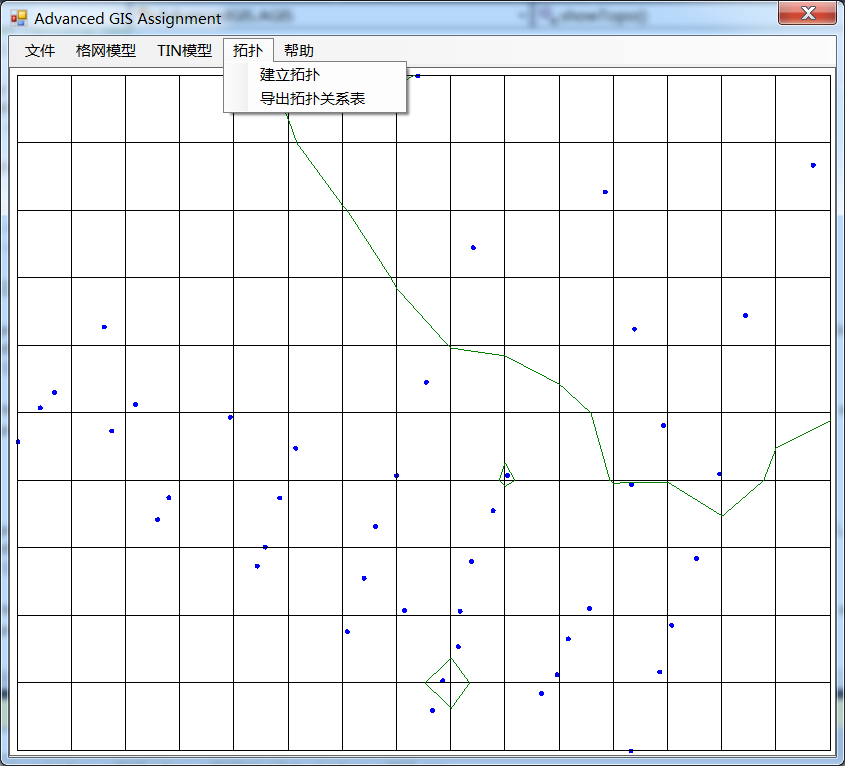
**2.9 由TIN生成等值线**



**2.10 生成TIN等值线结果（光滑和不光滑）**

## 2.9生成拓扑

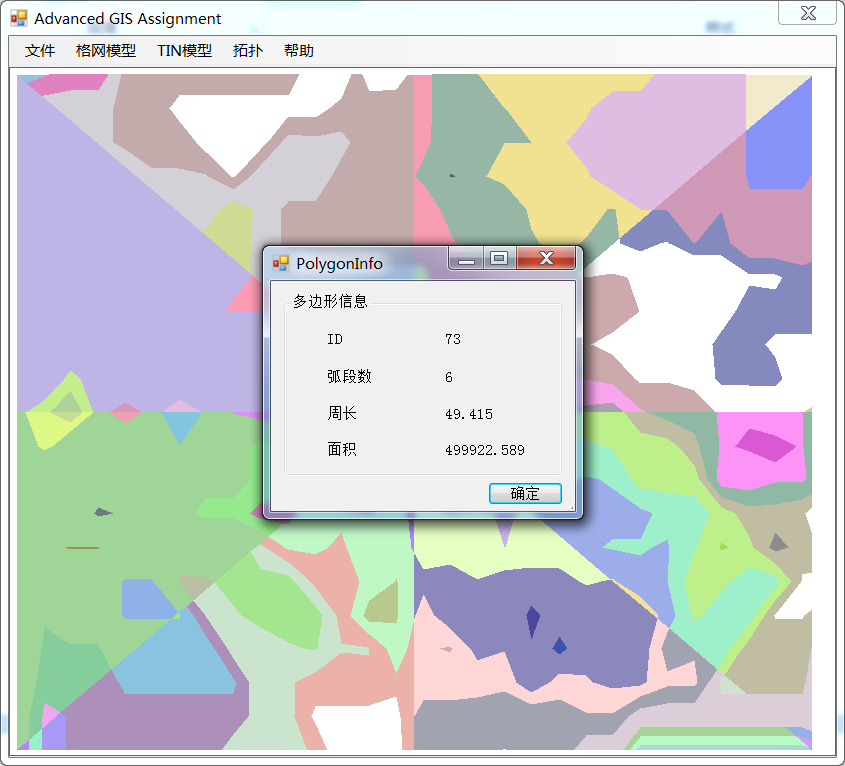
根据点集的外包矩形边界、外包矩形的对角线、各对边中点连线和格网生成的等值线自动生成拓扑如图2.11所示。



**图2.11 生成拓扑**

## 2.10查询多边形信息

根据生成的拓扑，查询生成的多边形的属性信息。用鼠标点击要查询的多边形，多边形的信息如图2.12所示。



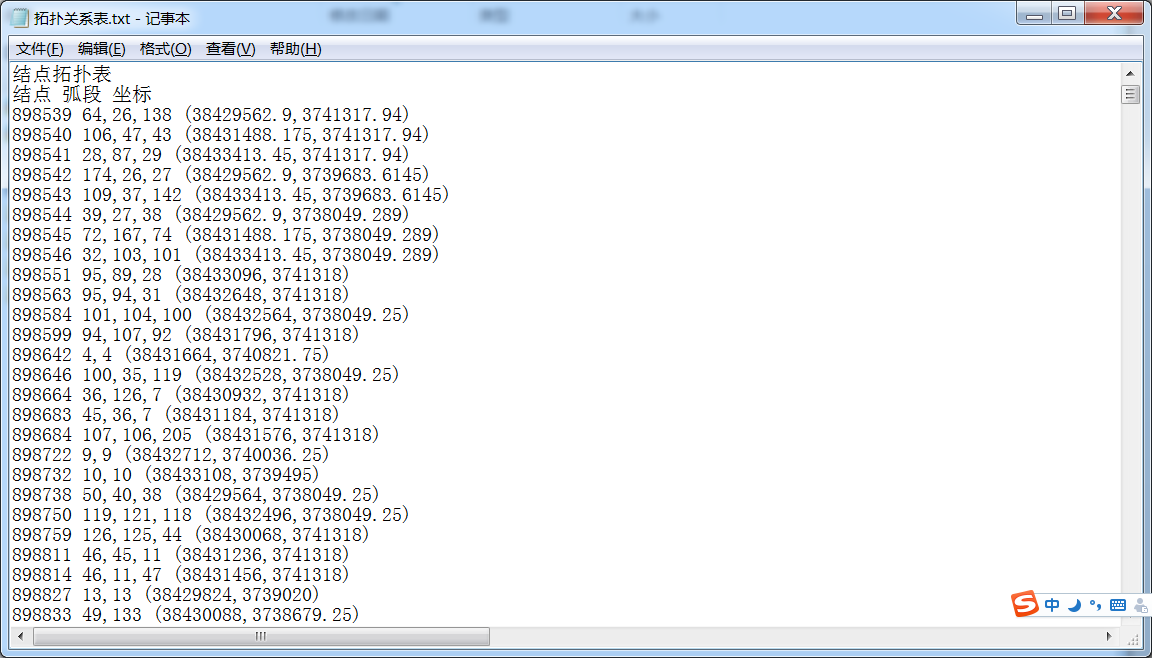
**图2.12 查询多边形信息**

## 2.11查看并保存拓扑关系表

查看生成的拓扑关系表并保存成文本文件，如图2.13和2.14所示。



**图2.13 拓扑关系表**



**图2.14 拓扑关系表保存成文本文件**

# 3.代码展示

## 3.1距离平方倒数法插值

距离平方倒数法插值的代码如下所示。

/// <summary>

/// 距离平方倒数法插值

/// </summary>

/// <param name="x">插值点屏幕横坐标</param>

/// <param name="y">插值点屏幕纵坐标</param>

/// <returns>插值点预测的属性值</returns>

private double GetAttributes0(double x,double y)

{

double wSum = 0,sum=0;

double xo = (x - margin) \* scale + minx;//待插值位置横坐标

double yo = (pictureBox1.Height-y - margin) \* scale + miny;//待插值位置纵坐标

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

{

double reciprocal = 1 / Math.Sqrt((data[i].getX() - xo) \* (data[i].getX() - xo) + (data[i].getY() - yo) \* (data[i].getY() - yo));

wSum += reciprocal;

sum += data[i].getZ() \* reciprocal;

}

double preVal = sum / wSum;

return preVal;

}

## 3.2按方位加权平均法插值

按方位加权平均法插值的代码如下所示。

/// <summary>

/// 按方位加权平均法插值

/// </summary>

/// <param name="x0">插值点屏幕横坐标</param>

/// <param name="y0">插值点屏幕纵坐标</param>

/// <returns>插值点属性预测值</returns>

private double GetAttributes1(double x0,double y0)

{

double x = (x0 - margin) \* scale + minx;//插值点位置横坐标

double y = (pictureBox1.Height - y0 - margin) \* scale + miny;//插值点位置纵坐标

double preValue = 0, wSum = 0;

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

List<double> minDis = new List<double>();

for (int i = 0; i < 8;i++ )//对八个方位，分别找离插值点最近的点

{

int tt1 = -1;

double tt2 = double.MaxValue;

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

{

double alpha=calAlpha(new MyPoint(x,y),new MyPoint(data[j].getX(),data[j].getY()));

if(alpha>=i\*45&&alpha<(i+1)\*45)

{

if(Math.Sqrt((x-data[j].getY())\*(x-data[j].getY())+(y-data[j].getY())\*(y-data[j].getY()))<tt2)

{

tt1 = j;

tt2 = Math.Sqrt((x - data[j].getY()) \* (x - data[j].getY()) + (y - data[j].getY()) \* (y - data[j].getY()));

}

}

}

if(tt1>=0)

{

index.Add(tt1);

minDis.Add(tt2);

}

}

for (int i = 0; i < index.Count; i++)//计算插值结果

{

double wTemp = 1;

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

{

if (j != i)

wTemp \*= minDis[j] \* minDis[j];

}

wSum += wTemp;

preValue += data[index[i]].getZ() \* wTemp;

}

preValue = preValue / wSum;

return preValue;

}

## 3.3格网自动生成等值线

网格自动生成等值线的代码如下所示。

/// <summary>

/// 网格自动生成等值线

/// </summary>

/// <param name="val">等值线的初始值</param>

/// <param name="IsoInterval">等值线间隔</param>

/// <param name="isSmooth">等值线是否光滑</param>

public void generateIsoLineGrid(double val,double IsoInterval,bool isSmooth)

{

resGrid.Clear();

//计算H矩阵

double[,] H = new double[xPaceC + 1, yPaceC + 1];

double xinterval = (pictureBox1.Width - 2 \* margin) / xPaceC;

double yinterval = (pictureBox1.Height - 2 \* margin) / yPaceC;

double HMax = double.MinValue, HMin = double.MaxValue;

for (int i = 0; i < xPaceC + 1; i++)

{

for (int j = 0; j < yPaceC + 1; j++)

{

if (methodC == 0)

H[i, j] = GetAttributes0(margin + xinterval \* i, pictureBox1.Height - (margin + yinterval \* j));

else

H[i, j] = GetAttributes1(margin + xinterval \* i, pictureBox1.Height - (margin + yinterval \* j));

if (HMax < H[i, j])

HMax = H[i, j];

if (HMin > H[i, j])

HMin = H[i, j];

}

}

//找到位于插值区域内的等值线最小值

if(val>HMax)

{

if ((int)((val - HMax) / IsoInterval) == (int)((val - HMin) / IsoInterval) && (int)((val - HMax) / IsoInterval) != (val - HMax) / IsoInterval)

return;

else

{

while(val>HMin)

{

val = val - IsoInterval;

}

val = val + IsoInterval;

}

}

else if(val<HMin)

{

if ((int)((HMin - val) / IsoInterval) == (int)((HMax - val) / IsoInterval) && (int)((HMin - val) / IsoInterval) != (HMin - val) / IsoInterval)

return;

else

{

while (val < HMin)

val = val + IsoInterval;

}

}

else

{

while(val>HMin)

val = val - IsoInterval;

val = val + IsoInterval;

}

for(double HZ=val;HZ<=HMax;HZ=HZ+IsoInterval)//对位于插值区域内的每个值进行插值

{

//对等值线的值HZ，计算HH

double[,] HH = new double[xPaceC, yPaceC+1];

for (int i = 0; i < xPaceC; i++)

{

for (int j = 0; j < yPaceC+1; j++)

{

HH[i, j] = (HZ - H[i, j]) / (H[i+1, j] - H[i, j]);

if (HH[i, j] < 0 || HH[i, j] > 1)

HH[i, j] = 2;

if (H[i, j] == 0)

H[i, j] = 0.0001;

if (H[i, j] == 1)

H[i, j] = 0.9999;

}

}

//对等值线的值HZ，计算SS

double[,] SS = new double[xPaceC+1, yPaceC];

for (int i = 0; i < xPaceC+1; i++)

{

for (int j = 0; j < yPaceC; j++)

{

SS[i, j] = (HZ - H[i, j]) / (H[i, j+1] - H[i, j]);

if (SS[i, j] < 0 || SS[i, j] > 1)

SS[i, j] = 2;

if (SS[i, j] == 0)

SS[i, j] = 0.0001;

if (SS[i, j] == 1)

SS[i, j] = 0.9999;

}

}

for (int i = 0; i < xPaceC; i++)//底边界找线头

{

if (HH[i, 0] >= 0 && HH[i, 0] <= 1)

{

PointF A = new PointF((float)(i + 0.1), -1);

PointF B = new PointF((float)(i + HH[i, 0]), 0);

resGrid.Add(GetOneIsoline(A, B, H, SS, HH));

}

}

for (int i = 0; i < yPaceC; i++)//左边界找线头

{

if(SS[0,i]>=0&&SS[0,i]<=1)

{

PointF A = new PointF(-1,(float)(i + 0.1));

PointF B = new PointF(0, (float)(i + SS[0, i]));

resGrid.Add(GetOneIsoline(A, B, H, SS, HH));

}

}

for (int i = 0; i < xPaceC; i++)//顶边界找线头

{

if (HH[i, yPaceC] >= 0 && HH[i, yPaceC] <= 1)

{

PointF A = new PointF((float)(i + 0.1), yPaceC + 1);

PointF B = new PointF((float)(i + HH[i, yPaceC]), yPaceC);

resGrid.Add(GetOneIsoline(A, B, H, SS, HH));

}

}

for (int i = 0; i < yPaceC; i++)//右边界找线头

{

if(SS[xPaceC,i]>=0&&SS[xPaceC,i]<=1)

{

PointF A = new PointF(xPaceC+1, (float)(i + 0.1));

PointF B = new PointF(xPaceC, (float)(i + SS[xPaceC, i]));

resGrid.Add(GetOneIsoline(A, B, H, SS, HH));

}

}

//找闭合等值线，纵边上找线头

for(int i=1;i<xPaceC;i++)

{

for(int j=0;j<yPaceC;j++)

{

if(SS[i,j]>=0&&SS[i,j]<=1)

{

PointF A = new PointF(i-1, (float)(j + 0.1));

PointF B = new PointF(i, (float)(j + SS[i,j]));

resGrid.Add(GetOneIsoline(A, B, H, SS, HH));

}

}

}

}

//绘制等值线

Graphics g = pictureBox1.CreateGraphics();

Pen myPen=new Pen(Color.Green);

foreach(List<PointF> lp in resGrid)

{

PointF[] temp = new PointF[lp.Count];

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

{

PointF tmp2 = new PointF((float)((lp[i].X - minx) / scale + margin), (float)(pictureBox1.Height - margin - (lp[i].Y - miny) / scale));

temp[i] = tmp2;

}

if (isSmooth)//画光滑等值线

g.DrawCurve(myPen, temp);

else//画不光滑等值线

g.DrawLines(myPen, temp);

}

myPen.Dispose();

g.Dispose();

return;

}

追踪一条等值线的代码如下所示。

/// <summary>

/// 根据初始线头在格网中追踪一条等值线

/// </summary>

/// <param name="AA">初始假想点，确定初始追踪方向</param>

/// <param name="BB">线头</param>

/// <param name="H">H矩阵</param>

/// <param name="SS">SS矩阵</param>

/// <param name="HH">HH矩阵</param>

/// <returns>一条等值线</returns>

public List<PointF> GetOneIsoline(PointF AA,PointF BB,double[,]H,double[,] SS,double[,]HH)

{

double xinterval = (pictureBox1.Width - 2 \* margin) / xPaceC;

double yinterval = (pictureBox1.Height - 2 \* margin) / yPaceC;

List<PointF> isol = new List<PointF>();

PointF A = AA;

PointF B = BB;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

//开始追踪

while (((int)A.Y < (int)B.Y && B.Y < yPaceC) || ((int)A.X < (int)B.X && B.X < xPaceC) || ((int)A.Y >= (int)B.Y && (int)A.X >= (int)B.X && (int)B.X < B.X && B.Y > 0) || ((int)A.Y >= (int)B.Y && (int)A.X >= (int)B.X && (int)B.Y < B.Y && B.X > 0))//追踪没有达到格网边界

{

if ((int)A.Y < (int)B.Y)//自下向上追踪

{

if (SS[(int)B.X, (int)B.Y] >= 0 && SS[(int)B.X, (int)B.Y] <= 1)//左边有点

{

if (SS[(int)B.X + 1, (int)B.Y] >= 0 && SS[(int)B.X + 1, (int)B.Y] <= 1)//三边都有点

{

double disL = Math.Sqrt(((B.X - (int)B.X) \* xinterval) \* ((B.X - (int)B.X) \* xinterval) + yinterval \* yinterval \* SS[(int)B.X, (int)B.Y] \* SS[(int)B.X, (int)B.Y]);

double disR = Math.Sqrt(((1 + (int)B.X - B.X) \* xinterval) \* ((1 + (int)B.X - B.X) \* xinterval) + SS[(int)B.X + 1, (int)B.Y] \* yinterval \* SS[(int)B.X + 1, (int)B.Y] \* yinterval);

if (disL > disR)//判断距离左右的距离

{

PointF tt = new PointF((int)B.X + 1, (float)(B.Y + SS[(int)B.X + 1, (int)B.Y]));

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

HH[(int)A.X, (int)A.Y] = 2;

continue;

}

else

{

PointF tt = new PointF((int)B.X, (float)(B.Y + SS[(int)B.X, (int)B.Y]));

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

HH[(int)A.X, (int)A.Y] = 2;

continue;

}

}

else//只有左边有点

{

PointF tt = new PointF((int)B.X, (float)(B.Y + SS[(int)B.X, (int)B.Y]));

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

HH[(int)A.X, (int)A.Y] = 2;

continue;

}

}

if (SS[(int)B.X + 1, (int)B.Y] >= 0 && SS[(int)B.X + 1, (int)B.Y] <= 1)//只有右边有点

{

PointF tt = new PointF((int)B.X + 1, (float)(B.Y + SS[(int)B.X + 1, (int)B.Y]));

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

HH[(int)A.X, (int)A.Y] = 2;

continue;

}

if (HH[(int)B.X, (int)B.Y + 1] >= 0 && HH[(int)B.X, (int)B.Y + 1] <= 1)//只有右边有点

{

PointF tt = new PointF((int)B.X + (float)HH[(int)B.X, (int)B.Y + 1], (int)B.Y + 1);

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

HH[(int)A.X, (int)A.Y] = 2;

continue;

}

}

if ((int)A.X < (int)B.X)//从左向右追踪

{

if (HH[(int)B.X, (int)B.Y + 1] >= 0 && HH[(int)B.X, (int)B.Y + 1]<=1)//上边有点

{

if(HH[(int)B.X,(int)B.Y]>=0&&HH[(int)B.X,(int)B.Y]<=1)//三边都有点

{

double disd = Math.Sqrt(((B.Y - (int)B.Y) \* yinterval) \* ((B.Y - (int)B.Y) \* yinterval) + xinterval \* xinterval \* HH[(int)B.X,(int)B.Y] \* HH[(int)B.X,(int)B.Y]);

double disu = Math.Sqrt(((1 + (int)B.Y - B.Y) \* yinterval) \* ((1 + (int)B.Y - B.Y) \* yinterval) + HH[(int)B.X, (int)B.Y + 1] \* xinterval \* HH[(int)B.X, (int)B.Y + 1] \* xinterval);

if (disd > disu)//判断上下边的距离

{

PointF tt = new PointF((float)(B.X + HH[(int)B.X, (int)B.Y + 1]), (int)B.Y + 1);

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

SS[(int)A.X, (int)A.Y] = 2;

continue;

}

else

{

PointF tt = new PointF((float)(B.X + HH[(int)B.X, (int)B.Y]), (int)B.Y);

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

SS[(int)A.X, (int)A.Y] = 2;

continue;

}

}

else//只有上边有点

{

PointF tt = new PointF((float)(B.X + HH[(int)B.X, (int)B.Y + 1]), (int)B.Y + 1);

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

SS[(int)A.X, (int)A.Y] = 2;

continue;

}

}

if(HH[(int)B.X,(int)B.Y]>=0&&HH[(int)B.X,(int)B.Y]<=1)//只有下边有点

{

PointF tt = new PointF((float)(B.X + HH[(int)B.X, (int)B.Y]), (int)B.Y);

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

SS[(int)A.X, (int)A.Y] = 2;

continue;

}

if(SS[(int)B.X+1,(int)B.Y]>=0&&SS[(int)B.X+1,(int)B.Y]<=1)//只有右边有点

{

PointF tt = new PointF((float)(B.X + 1), (int)B.Y + (float)SS[(int)B.X + 1, (int)B.Y]);

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

SS[(int)A.X, (int)A.Y] = 2;

continue;

}

}

if ((int)B.X < B.X)//自上向下追踪

{

if (SS[(int)B.X, (int)B.Y - 1] >= 0 && SS[(int)B.X, (int)B.Y - 1]<=1)//左边有点

{

if(SS[(int)B.X+1,(int)B.Y-1]>=0&&SS[(int)B.X+1,(int)B.Y-1]<=1)//三边都有点

{

double disL = Math.Sqrt(((B.X - (int)B.X) \* xinterval) \* ((B.X - (int)B.X) \* xinterval) + yinterval \* yinterval \* (1 - SS[(int)B.X, (int)B.Y - 1]) \* (1 - SS[(int)B.X, (int)B.Y - 1]));

double disR = Math.Sqrt(((1 + (int)B.X - B.X) \* xinterval) \* ((1 + (int)B.X - B.X) \* xinterval) + (1 - SS[(int)B.X + 1, (int)B.Y - 1]) \* yinterval \* (1 - SS[(int)B.X + 1, (int)B.Y - 1]) \* yinterval);

if (disL > disR)//判断连左边还是右边

{

PointF tt = new PointF((int)B.X + 1, (float)(B.Y-1 + SS[(int)B.X + 1, (int)B.Y-1]));

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

HH[(int)A.X, (int)A.Y] = 2;

continue;

}

else

{

PointF tt = new PointF((int)B.X, (float)(B.Y-1 + SS[(int)B.X, (int)B.Y-1]));

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

HH[(int)A.X, (int)A.Y] = 2;

continue;

}

}

else//只有左边有点

{

PointF tt = new PointF((int)B.X, (float)(B.Y - 1 + SS[(int)B.X, (int)B.Y - 1]));

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

HH[(int)A.X, (int)A.Y] = 2;

continue;

}

}

if (SS[(int)B.X + 1, (int)B.Y - 1] >= 0 && SS[(int)B.X + 1, (int)B.Y - 1] <= 1)//只有右边有点

{

PointF tt = new PointF((int)B.X + 1, (float)(B.Y - 1 + SS[(int)B.X + 1, (int)B.Y - 1]));

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

HH[(int)A.X, (int)A.Y] = 2;

continue;

}

if(HH[(int)B.X,(int)B.Y-1]>=0&&HH[(int)B.X,(int)B.Y-1]<=1)//只有下边有点

{

PointF tt = new PointF((int)B.X + (float)HH[(int)B.X, (int)B.Y - 1], (float)(B.Y - 1));

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

HH[(int)A.X, (int)A.Y] = 2;

continue;

}

}

if ((int)B.Y < B.Y)//自右向左追踪

{

if (HH[(int)B.X - 1, (int)B.Y + 1] >= 0 && HH[(int)B.X - 1, (int)B.Y + 1]<=1)//上边有点

{

if(HH[(int)B.X-1,(int)B.Y]>=0&&HH[(int)B.X-1,(int)B.Y]<=1)//三边都有点

{

double disd = Math.Sqrt(((B.Y - (int)B.Y) \* yinterval) \* ((B.Y - (int)B.Y) \* yinterval) + xinterval \* xinterval \* (1 - HH[(int)B.X - 1, (int)B.Y]) \* (1 - HH[(int)B.X - 1, (int)B.Y]));

double disu = Math.Sqrt(((1 + (int)B.Y - B.Y) \* yinterval) \* ((1 + (int)B.Y - B.Y) \* yinterval) + (1 - HH[(int)B.X - 1, (int)B.Y + 1]) \* xinterval \* (1 - HH[(int)B.X - 1, (int)B.Y + 1]) \* xinterval);

if (disd > disu)//判断该连上边还是下边

{

PointF tt = new PointF((float)(B.X-1 + HH[(int)B.X-1, (int)B.Y + 1]), (int)B.Y + 1);

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

SS[(int)A.X, (int)A.Y] = 2;

continue;

}

else

{

PointF tt = new PointF((float)(B.X - 1 + HH[(int)B.X - 1, (int)B.Y]), (int)B.Y);

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

SS[(int)A.X, (int)A.Y] = 2;

continue;

}

}

else//只有上边有点

{

PointF tt = new PointF((float)(B.X - 1 + HH[(int)B.X - 1, (int)B.Y + 1]), (int)B.Y + 1);

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

SS[(int)A.X, (int)A.Y] = 2;

continue;

}

}

if (HH[(int)B.X - 1, (int)B.Y] >= 0 && HH[(int)B.X - 1, (int)B.Y] <= 1)//只有下边有点

{

PointF tt = new PointF((float)(B.X - 1 + HH[(int)B.X - 1, (int)B.Y]), (int)B.Y);

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

SS[(int)A.X, (int)A.Y] = 2;

continue;

}

if (SS[(int)B.X - 1, (int)B.Y] >= 0 && SS[(int)B.X - 1, (int)B.Y]<=1)//只有左边有点

{

PointF tt = new PointF((float)(B.X - 1), (int)B.Y + (float)SS[(int)B.X - 1, (int)B.Y]);

A = B;

B = tt;

isol.Add(new PointF((float)(B.X \* xinterval \* scale + minx), (float)(B.Y \* yinterval \* scale + miny)));

SS[(int)A.X, (int)A.Y] = 2;

continue;

}

}

//如果四种情况都不是，说明是闭曲线，加入开始的起点，追踪完毕

isol.Add(new PointF(isol[0].X,isol[0].Y));

break;

}

//追踪的最后一段曲线的HH或者SS矩阵赋值为2

if(B.X-(int)B.X==0)

SS[(int)B.X,(int)B.Y]=2;

else

HH[(int)B.X, (int)B.Y] = 2;

return isol;

}

## 3.4生成点集凸包

生成点集凸包的代码如下所示。

/// <summary>

/// 根据点集生成凸包，Graham's Scan算法

/// </summary>

/// <param name="od">点集</param>

/// <returns>凸包点序列</returns>

public static List<int> generateConvex(List<Dt> od)

{

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

List<double> tmp1 = new List<double>();

List<int> tmp2 = new List<int>();

//找到最低点的序号

int minYIndex=0;

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

{

if (od[i].getY() < od[minYIndex].getY() || (od[i].getY() == od[minYIndex].getY() && od[i].getX() < od[minYIndex].getX()))

minYIndex = i;

}

tmp1.Add(-1);

tmp2.Add(minYIndex);

//将所有点按照和最低点构成的向量的角度从大到小排列

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

{

if(i!=minYIndex)

{

double cos = (od[i].getX() - od[minYIndex].getX()) / Math.Sqrt((od[i].getX() - od[minYIndex].getX()) \* (od[i].getX() - od[minYIndex].getX()) + (od[i].getY() - od[minYIndex].getY()) \* (od[i].getY() - od[minYIndex].getY()));

int j;

for(j=0;j<tmp1.Count;j++)

{

if(cos<tmp1[j])

{

tmp1.Insert(j,cos);

tmp2.Insert(j,i);

break;

}

}

if(j==tmp1.Count)

{

tmp1.Add(cos);

tmp2.Add(i);

}

}

}

//开始追踪凸包多边形

res.Add(tmp2[0]);

res.Add(tmp2[1]);

for (int i = 2; i < tmp2.Count;i++ )

{

//计算两个向量的叉积

double s = (od[res[res.Count - 1]].getX() - od[res[res.Count - 2]].getX()) \* (od[tmp2[i]].getY() - od[res[res.Count - 1]].getY()) - (od[res[res.Count - 1]].getY() - od[res[res.Count - 2]].getY()) \* (od[tmp2[i]].getX() - od[res[res.Count - 1]].getX());

while(s>0)//叉积大于零，需要删去之前的点

{

res.RemoveAt(res.Count - 1);

s = (od[res[res.Count - 1]].getX() - od[res[res.Count - 2]].getX()) \* (od[tmp2[i]].getY() - od[res[res.Count - 1]].getY()) - (od[res[res.Count - 1]].getY() - od[res[res.Count - 2]].getY()) \* (od[tmp2[i]].getX() - od[res[res.Count - 1]].getX());

}

res.Add(tmp2[i]);

}

return res;

}

## 3.5生成点集TIN模型

根据点集生成TIN模型的代码如下所示。

/// <summary>

/// 生长法生成TIN

/// </summary>

/// <param name="data">原始点集</param>

/// <param name="convex">点集凸包</param>

/// <param name="arcList">TIN的边的列表</param>

/// <param name="TINConvex">边描述的TIN凸包</param>

/// <param name="res">TIN三角网</param>

public static void generateTIN(List<Dt> data, List<int> convex, List<Arc> arcList, List<int> TINConvex, List<AdvancedGIS.dataStructure.Triangle> res)

{

res.Clear();

arcList.Clear();

TINConvex.Clear();

List<int> queue = new List<int>();//边的队列

Arc at = new dataStructure.Arc(0, convex[0] > convex[1] ? convex[1] : convex[0], convex[0] > convex[1] ? convex[0] : convex[1], 0);

at.po=true;

at.po = zhengfuqu(at,data,data[convex[convex.Count-1]]);

arcList.Add(at);

queue.Add(0);//加入初始生长边

while(queue.Count>0)//队列不空，继续生长

{

double tmpCos=double.MaxValue;

int ind=-1;

for(int i=0;i<data.Count;i++)//求位于当前生长边一侧的张角最大的点的序号

{

if (i != arcList[queue[0]].startPoint && i != arcList[queue[0]].endPoint && zhengfuqu(arcList[queue[0]],data,data[i]))

{

double tt=COS(data[arcList[queue[0]].startPoint], data[arcList[queue[0]].endPoint],data[i]);

if (tt < tmpCos)

{

tmpCos = tt;

ind = i;

}

}

}

if(ind<0)//在追踪边一侧未找到点，删除此边

{

arcList[queue[0]].useTimes++;

queue.RemoveAt(0);

continue;

}

//找到了点，则向队列里新加两边，得到一个三角形

arcList[queue[0]].useTimes++;

int sp=arcList[queue[0]].startPoint > ind ? ind : arcList[queue[0]].startPoint;

int ep=arcList[queue[0]].startPoint > ind ? arcList[queue[0]].startPoint:ind;

int k1;

for(k1=0;k1<arcList.Count;k1++)

{

if (arcList[k1].startPoint == sp && arcList[k1].endPoint == ep)

{

arcList[k1].useTimes++;

break;

}

}

if (k1 == arcList.Count)

{

at = new AdvancedGIS.dataStructure.Arc(arcList.Count, sp, ep, 1);

at.po=false;

at.po = zhengfuqu(at, data, data[arcList[queue[0]].endPoint]);

arcList.Add(at);

}

sp = arcList[queue[0]].endPoint > ind ? ind : arcList[queue[0]].endPoint;

ep = arcList[queue[0]].endPoint > ind ? arcList[queue[0]].endPoint : ind;

int k2;

for (k2 = 0; k2 < arcList.Count; k2++)

{

if (arcList[k2].startPoint == sp && arcList[k2].endPoint == ep)

{

arcList[k2].useTimes++;

break;

}

}

if (k2 == arcList.Count)

{

at = new AdvancedGIS.dataStructure.Arc(arcList.Count, sp, ep, 1);

at.po = false;

at.po = zhengfuqu(at,data,data[arcList[queue[0]].startPoint]);

arcList.Add(at);

}

int[] tmpID = new int[3];

tmpID[0]= queue[0];

tmpID[1] = k1;

tmpID[2] = k2;

for(int i=0;i<tmpID.Length-1;i++)

{

for(int j=i+1;j<tmpID.Length;j++)

{

if(tmpID[i]>tmpID[j])

{

int ttt = tmpID[i];

tmpID[i] = tmpID[j];

tmpID[j] = ttt;

}

}

}

Triangle tmpTri = new AdvancedGIS.dataStructure.Triangle(arcList[queue[0]].startPoint, arcList[queue[0]].endPoint, ind);

tmpTri.arcID1 = tmpID[0];

tmpTri.arcID2 = tmpID[1];

tmpTri.arcID3 = tmpID[2];

int kk;

for (kk = 0; kk < res.Count;kk++ )

{

if (res[kk].arcID1 == tmpTri.arcID1 && res[kk].arcID2 == tmpTri.arcID2 && res[kk].arcID3 == tmpTri.arcID3)

break;

}

if (kk == res.Count)

res.Add(tmpTri);

if (arcList[k1].useTimes < 2)

queue.Add(k1);

if (arcList[k2].useTimes < 2)

queue.Add(k2);

queue.RemoveAt(0);

}

//三角网生成结束，下面生成拓扑关系

//给边加入三角形信息

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

{

if (arcList[res[i].arcID1].tri[0] == -1)

arcList[res[i].arcID1].tri[0] = i;

else if (arcList[res[i].arcID1].tri[1] == -1)

arcList[res[i].arcID1].tri[1] = i;

if (arcList[res[i].arcID2].tri[0] == -1)

arcList[res[i].arcID2].tri[0] = i;

else if (arcList[res[i].arcID2].tri[1] == -1)

arcList[res[i].arcID2].tri[1] = i;

if (arcList[res[i].arcID3].tri[0] == -1)

arcList[res[i].arcID3].tri[0] = i;

else if (arcList[res[i].arcID3].tri[1] == -1)

arcList[res[i].arcID3].tri[1] = i;

}

//生成以边为基础的点集凸包

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

{

int minPP = convex[i] > convex[(i + 1) % convex.Count] ? convex[(i + 1) % convex.Count] : convex[i];

int maxPP = convex[i] <= convex[(i + 1) % convex.Count] ? convex[(i + 1) % convex.Count] : convex[i];

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

{

if (minPP == arcList[j].startPoint && maxPP == arcList[j].endPoint)

TINConvex.Add(j);

}

}

return;

}

正负区判断的代码如下所示。

public static bool zhengfuqu(Arc a,List<Dt> data,Dt p3)

{

if(data[a.startPoint].getX()==data[a.endPoint].getX())

{

if (p3.getX() > data[a.startPoint].getX())

return !a.po;

else

return a.po;

}

else

{

double tmp = (data[a.startPoint].getY() - data[a.endPoint].getY()) / (data[a.startPoint].getX() - data[a.endPoint].getX()) \* (p3.getX() - data[a.endPoint].getX()) + data[a.endPoint].getY();

return (tmp-p3.getY())> 0 ? (!a.po) : a.po;

}

}

## 3.6 TIN自动生成等值线

自动生成并绘制等值线的代码如下所示。

/// <summary>

/// TIN模型生成等值线

/// </summary>

/// <param name="val">等值线起始值</param>

/// <param name="IsoInterval">等值线间隔</param>

/// <param name="isSmooth">是否光滑</param>

public void generateIsoLineTIN(double val, double IsoInterval,bool isSmooth)

{

List<List<PointF>> res = new List<List<PointF>>();//存储结果

//计算区域内存在的值最小的等值线

double HMax = double.MinValue, HMin = double.MaxValue;

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

{

if (HMax < data[i].getZ())

HMax = data[i].getZ();

if (HMin > data[i].getZ())

HMin = data[i].getZ();

}

if (val > HMax)

{

if ((int)((val - HMax) / IsoInterval) == (int)((val - HMin) / IsoInterval) && (int)((val - HMax) / IsoInterval) != (val - HMax) / IsoInterval)

return;

else

{

while (val > HMin)

{

val = val - IsoInterval;

}

val = val + IsoInterval;

}

}

else if (val < HMin)

{

if ((int)((HMin - val) / IsoInterval) == (int)((HMax - val) / IsoInterval) && (int)((HMin - val) / IsoInterval) != (HMin - val) / IsoInterval)

return;

else

{

while (val < HMin)

{

val = val + IsoInterval;

}

}

}

else

{

while (val > HMin)

{

val = val - IsoInterval;

}

val = val + IsoInterval;

}

//对每个等值线的值追踪等值线

for (double HZ = val; HZ <= HMax; HZ = HZ + IsoInterval)

{

//计算HH矩阵

double[] HH = new double[triArc.Count];

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

{

HH[i]=(HZ-data[triArc[i].startPoint].getZ())/(data[triArc[i].endPoint].getZ()-data[triArc[i].startPoint].getZ());

if(HH[i]==1)

HH[i]=0.9999;

else if(HH[i]==0)

HH[i]=0.0001;

else if(HH[i]>1||HH[i]<0)

HH[i]=2;

}

//找开曲线线头

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

{

if(HH[TINConvex[i]]<=1&&HH[TINConvex[i]]>=0)

{

res.Add(GetOneIsolineTin(TINConvex[i], HH));

}

}

//找闭曲线线头

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

{

if(HH[i]>=0&&HH[i]<=1)

{

res.Add(GetOneIsolineTin(i, HH));

}

}

}

//绘制等值线

Graphics g = pictureBox1.CreateGraphics();

Pen myPen = new Pen(Color.Red);

foreach (List<PointF> lp in res)

{

PointF[] temp = new PointF[lp.Count];

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

{

PointF tmp2 = new PointF((float)((lp[i].X - minx) / scale + margin), (float)(pictureBox1.Height - margin - (lp[i].Y - miny) / scale));

temp[i] = tmp2;

}

if (isSmooth)//绘制光滑等值线

g.DrawCurve(myPen, temp);

else//不光滑等值线

g.DrawLines(myPen, temp);

}

myPen.Dispose();

g.Dispose();

return;

}

追踪一条等值线的代码如下所示。

/// <summary>

/// 根据线头追踪一条等值线

/// </summary>

/// <param name="arcStart">线头</param>

/// <param name="HH">HH矩阵</param>

/// <returns>追踪的等值线</returns>

public List<PointF> GetOneIsolineTin(int arcStart,double[]HH)

{

List<PointF> res = new List<PointF>();//存结果

List<int> queue=new List<int>();//追踪队列

queue.Add(arcStart);

res.Add(new PointF((float)(data[triArc[arcStart].startPoint].getX()+HH[arcStart]\*(data[triArc[arcStart].endPoint].getX()-data[triArc[arcStart].startPoint].getX())),(float)(data[triArc[arcStart].startPoint].getY()+HH[arcStart]\*(data[triArc[arcStart].endPoint].getY()-data[triArc[arcStart].startPoint].getY()))));

HH[arcStart]=2;

while(queue.Count>0)//追踪队列不空，追踪未结束

{

bool ischanged=false;

//找边所在的三角形

if(triArc[queue[0]].tri[0]!=-1)

{

Triangle tmpT=TINRes[triArc[queue[0]].tri[0]];

//分别找三角形三边是否存在当前点。若有则加入等值线，，所在边加入队列

if(HH[tmpT.arcID1]>=0&&HH[tmpT.arcID1]<=1)

{

queue.Add(tmpT.arcID1);

res.Add(new PointF((float)(data[triArc[tmpT.arcID1].startPoint].getX() + HH[tmpT.arcID1] \* (data[triArc[tmpT.arcID1].endPoint].getX() - data[triArc[tmpT.arcID1].startPoint].getX())), (float)(data[triArc[tmpT.arcID1].startPoint].getY() + HH[tmpT.arcID1] \* (data[triArc[tmpT.arcID1].endPoint].getY() - data[triArc[tmpT.arcID1].startPoint].getY()))));

HH[tmpT.arcID1] = 2;

ischanged=true;

}

else if (HH[tmpT.arcID2] >= 0 && HH[tmpT.arcID2] <= 1)

{

queue.Add(tmpT.arcID2);

res.Add(new PointF((float)(data[triArc[tmpT.arcID2].startPoint].getX() + HH[tmpT.arcID2] \* (data[triArc[tmpT.arcID2].endPoint].getX() - data[triArc[tmpT.arcID2].startPoint].getX())), (float)(data[triArc[tmpT.arcID2].startPoint].getY() + HH[tmpT.arcID2] \* (data[triArc[tmpT.arcID2].endPoint].getY() - data[triArc[tmpT.arcID2].startPoint].getY()))));

HH[tmpT.arcID2] = 2;

ischanged=true;

}

else if (HH[tmpT.arcID3] >= 0 && HH[tmpT.arcID3] <= 1)

{

queue.Add(tmpT.arcID3);

res.Add(new PointF((float)(data[triArc[tmpT.arcID3].startPoint].getX() + HH[tmpT.arcID3] \* (data[triArc[tmpT.arcID3].endPoint].getX() - data[triArc[tmpT.arcID3].startPoint].getX())), (float)(data[triArc[tmpT.arcID3].startPoint].getY() + HH[tmpT.arcID3] \* (data[triArc[tmpT.arcID3].endPoint].getY() - data[triArc[tmpT.arcID3].startPoint].getY()))));

HH[tmpT.arcID3] = 2;

ischanged=true;

}

}

if(triArc[queue[0]].tri[1]!=-1)

{

Triangle tmpT = TINRes[triArc[queue[0]].tri[1]];

//分别找三角形三边是否存在当前点。若有则加入等值线，，所在边加入队列

if (HH[tmpT.arcID1] >= 0 && HH[tmpT.arcID1] <= 1)

{

queue.Add(tmpT.arcID1);

res.Add(new PointF((float)(data[triArc[tmpT.arcID1].startPoint].getX() + HH[tmpT.arcID1] \* (data[triArc[tmpT.arcID1].endPoint].getX() - data[triArc[tmpT.arcID1].startPoint].getX())), (float)(data[triArc[tmpT.arcID1].startPoint].getY() + HH[tmpT.arcID1] \* (data[triArc[tmpT.arcID1].endPoint].getY() - data[triArc[tmpT.arcID1].startPoint].getY()))));

HH[tmpT.arcID1] = 2;

ischanged=true;

}

else if (HH[tmpT.arcID2] >= 0 && HH[tmpT.arcID2] <= 1)

{

queue.Add(tmpT.arcID2);

res.Add(new PointF((float)(data[triArc[tmpT.arcID2].startPoint].getX() + HH[tmpT.arcID2] \* (data[triArc[tmpT.arcID2].endPoint].getX() - data[triArc[tmpT.arcID2].startPoint].getX())), (float)(data[triArc[tmpT.arcID2].startPoint].getY() + HH[tmpT.arcID2] \* (data[triArc[tmpT.arcID2].endPoint].getY() - data[triArc[tmpT.arcID2].startPoint].getY()))));

HH[tmpT.arcID2] = 2;

ischanged=true;

}

else if (HH[tmpT.arcID3] >= 0 && HH[tmpT.arcID3] <= 1)

{

queue.Add(tmpT.arcID3);

res.Add(new PointF((float)(data[triArc[tmpT.arcID3].startPoint].getX() + HH[tmpT.arcID3] \* (data[triArc[tmpT.arcID3].endPoint].getX() - data[triArc[tmpT.arcID3].startPoint].getX())), (float)(data[triArc[tmpT.arcID3].startPoint].getY() + HH[tmpT.arcID3] \* (data[triArc[tmpT.arcID3].endPoint].getY() - data[triArc[tmpT.arcID3].startPoint].getY()))));

HH[tmpT.arcID3] = 2;

ischanged=true;

}

}

if (triArc[queue[0]].tri[1] != -1 && triArc[queue[0]].tri[0] != -1&&ischanged==false)//闭曲线

{

res.Add(new PointF(res[0].X,res[0].Y));

}

queue.RemoveAt(0);

}

return res;

}

## 3.7自动生成拓扑

拓扑生成的代码如下所示。

private void 建立拓扑ToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (gridIsoline == false)

MessageBox.Show("没有利用格网生成的等值线！");

gridFlag = false;

mp.Clear();

ma.Clear();

mpol.Clear();

initData();//初始化基础的点和线

breakLine();//线之间两两打断

for(int i = 0; i < ma.Count;i++)//给所有点生成拓扑关系

{

mp[ma[i].startPoint].lineNum++;

mp[ma[i].startPoint].relLine.Add(i);

mp[ma[i].endPoint].lineNum++;

mp[ma[i].endPoint].relLine.Add(-i-1);

for(int j=0;j<ma[i].midPointNum;j++)

{

mp[ma[i].midPoint[j]].lineNum++;

mp[ma[i].midPoint[j]].relLine.Add(i);

}

}

trackPolygon();//追踪多边形并生成拓扑

showTopo();//显示拓扑生成结果，用不同颜色表示不同多边形

topo = true;

}

/// <summary>

/// 加入原始点和线

/// </summary>

private void initData()

{

//加入点集外包矩形等的基础数据

double EnvMaxx = double.MinValue, EnvMinx = double.MaxValue, EnvMaxy = double.MinValue, EnvMiny = double.MaxValue;

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

{

if (EnvMaxx < data[i].getX())

EnvMaxx = data[i].getX();

if (EnvMinx > data[i].getX())

EnvMinx = data[i].getX();

if (EnvMaxy < data[i].getY())

EnvMaxy = data[i].getY();

if (EnvMiny > data[i].getY())

EnvMiny = data[i].getY();

}

mp.Add(new MyPoint(EnvMinx, EnvMaxy));

mp.Add(new MyPoint(0.5 \* (EnvMinx + EnvMaxx), EnvMaxy));

mp.Add(new MyPoint(EnvMaxx, EnvMaxy));

mp.Add(new MyPoint(EnvMinx, 0.5 \* (EnvMaxy + EnvMiny)));

mp.Add(new MyPoint(EnvMaxx, 0.5 \* (EnvMaxy + EnvMiny)));

mp.Add(new MyPoint(EnvMinx, EnvMiny));

mp.Add(new MyPoint(0.5 \* (EnvMinx + EnvMaxx), EnvMiny));

mp.Add(new MyPoint(EnvMaxx, EnvMiny));

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

arctmp.Add(0);

arctmp.Add(2);

ma.Add(new MyArc(arctmp));

arctmp.Clear();

arctmp.Add(0);

arctmp.Add(5);

ma.Add(new MyArc(arctmp));

arctmp.Clear();

arctmp.Add(2);

arctmp.Add(7);

ma.Add(new MyArc(arctmp));

arctmp.Clear();

arctmp.Add(5);

arctmp.Add(7);

ma.Add(new MyArc(arctmp));

arctmp.Clear();

arctmp.Add(0);

arctmp.Add(7);

ma.Add(new MyArc(arctmp));

arctmp.Clear();

arctmp.Add(2);

arctmp.Add(5);

ma.Add(new MyArc(arctmp));

arctmp.Clear();

arctmp.Add(3);

arctmp.Add(4);

ma.Add(new MyArc(arctmp));

arctmp.Clear();

arctmp.Add(1);

arctmp.Add(6);

ma.Add(new MyArc(arctmp));

//加入格网生成的等值线的基础数据

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

{

arctmp.Clear();

for (int j = 0; j < resGrid[i].Count - 1; j++)

{

arctmp.Add(addOnePointToMP(new MyPoint(resGrid[i][j].X, resGrid[i][j].Y)));

}

if (resGrid[i][resGrid[i].Count - 1].X == resGrid[i][0].X && resGrid[i][resGrid[i].Count - 1].Y == resGrid[i][0].Y)

{

arctmp.Add(arctmp[0]);

}

else

{

arctmp.Add(addOnePointToMP(new MyPoint(resGrid[i][resGrid[i].Count - 1].X, resGrid[i][resGrid[i].Count - 1].Y)));

}

ma.Add(new MyArc(arctmp));

}

}

/// <summary>

/// 线段之间两两打断

/// </summary>

private void breakLine()

{

for (int i = 0; i < ma.Count - 1; i++)

{

for (int j = i + 1; j < ma.Count; j++)

{

if (ma[i].midPointNum+2 == 2)

{

if (ma[j].midPointNum+2 == 2)

j=breakTwoTwo(i, j);//两点线段之间打断

else

j=breakTwoMore(i, j);//两点和多点

}

else

{

if (ma[j].midPointNum+2 == 2)

j = breakTwoMore(j, i);//多点和两点

else

j=breakMoreMore(i, j);//多点和多点

}

}

}

}

private int breakTwoTwo(int i,int j)

{

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

MyPoint p1 = mp[ma[i].startPoint], p2 = mp[ma[i].endPoint],p3 = mp[ma[j].startPoint],p4 = mp[ma[j].endPoint];

MyPoint tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].endPoint);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

}

else if (isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].endPoint);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j-1;

}

}

else

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(ma[i].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j - 1;

}

}

}

return j;

}

private int breakTwoMore(int i,int j)

{

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

MyPoint p1 = mp[ma[i].startPoint], p2 = mp[ma[i].endPoint], p3, p4;

p3 = mp[ma[j].startPoint];

p4 = mp[ma[j].midPoint[0]];

MyPoint tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)//两点对开头

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int k = 0; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

if (i < j)

{

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else

{

ma.RemoveAt(i);

ma.RemoveAt(j);

return j;

}

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

if (i < j)

return i;

else

return i - 1;

}

}

else if (isEqual(tres, p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(ma[i].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int k = 0; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

if (i < j)

return j - 1;

else

return j;

}

}

else

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].endPoint);

for (int k = 0; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

if (i < j)

return j - 1;

else

return j;

}

}

}

p3 = mp[ma[j].midPoint[ma[j].midPointNum - 1]];

p4 = mp[ma[j].endPoint];

tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)//两点与后面

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

if (i < j)

{

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else

{

ma.RemoveAt(i);

ma.RemoveAt(j);

return j;

}

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

if (i < j)

{

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else

{

ma.RemoveAt(i);

ma.RemoveAt(j);

return j;

}

}

else

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].endPoint);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

if (i < j)

return i;

else

return i - 1;

}

}

else if (isEqual(tres, p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[i].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

if (i < j)

return j - 1;

else

return j;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < ma[j].midPointNum; k++)

pArray.Add(ma[j].midPoint[k]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

if (i < j)

return j - 1;

else

return j;

}

}

else

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].endPoint);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

if (i < j)

return j - 1;

else

return j;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < ma[j].midPointNum; k++)

pArray.Add(ma[j].midPoint[k]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

if (i < j)

return j - 1;

else

return j;

}

}

}

//两个点和中间

for (int s = 0; s < ma[j].midPointNum - 1; s++)

{

p3 = mp[ma[j].midPoint[s]];

p4 = mp[ma[j].midPoint[s + 1]];

tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < s + 1; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int k = s + 1; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

if (i < j)

{

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else

{

ma.RemoveAt(i);

ma.RemoveAt(j);

return j;

}

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].midPoint[s]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[s]);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < s + 1; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int k = s; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

if (i < j)

{

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else

{

ma.RemoveAt(i);

ma.RemoveAt(j);

return j;

}

}

}

else if (isEqual(tres, p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < s + 1; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[i].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int k = s + 1; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

if (i < j)

return j - 1;

else

return j;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < s + 1; k++)

pArray.Add(ma[j].midPoint[k]);

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int k = s ; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

if (i < j)

return j - 1;

else

return j;

}

}

else

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < s + 1; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].endPoint);

for (int k = s + 1; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

if (i < j)

return j - 1;

else

return j;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int k = 0; k < s + 1; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int k = s; k < ma[j].midPointNum; k++)

{

pArray.Add(ma[j].midPoint[k]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

if (i < j)

return j - 1;

else

return j;

}

}

}

}

if(i<j)

return j;

else

return i;

}

//private int breakMoreTwo(int i,int j)

//{

// return j;

//}

private int breakMoreMore(int i,int j)

{

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

//开头

MyPoint p1 = mp[ma[i].startPoint], p2 = mp[ma[i].midPoint[0]], p3, p4;

p3 = mp[ma[j].startPoint];

p4 = mp[ma[j].midPoint[0]];

MyPoint tres = CalIntersect(p1, p2, p3, p4);

if(tres!=null)//开头和开头

{

if(!isEqual(tres,p1)&&!isEqual(tres,p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int s = 0; s < ma[i].midPointNum;s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int s = 0; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

}

else if(isEqual(tres,p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(ma[i].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j - 1;

}

}

}

p3 = mp[ma[j].midPoint[ma[j].midPointNum-1]];

p4 = mp[ma[j].endPoint];

tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)//开头和结尾

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int s = 0; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

for (int s = 0; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum-1]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].endPoint);

for (int s = 0; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

}

else if (isEqual(tres, p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

pArray.Add(ma[i].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j - 1;

}

else if(isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum-1]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j - 1;

}

}

}

for (int s = 0; s < ma[j].midPointNum - 1;s++)

{

p3 = mp[ma[j].midPoint[s]];

p4 = mp[ma[j].midPoint[s+1]];

tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)//开头和中间

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < s+1; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int ss = s+1; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if(isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

pArray.Add(ma[j].midPoint[s]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[s]);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < s + 1; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int ss = s + 1; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

}

else if(isEqual(tres, p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < s + 1; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[i].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = s + 1; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j-1;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < s + 1; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int ss = s + 1; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j - 1;

}

}

}

}

//尾部

p1 = mp[ma[i].midPoint[ma[i].midPointNum - 1]];

p2 = mp[ma[i].endPoint];

p3 = mp[ma[j].startPoint];

p4 = mp[ma[j].midPoint[0]];

tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)//尾部对开头

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int ss = 0; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if(isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

pArray.Add(ma[j].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

}

else if(isEqual(tres, p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ma[i].midPointNum-1]);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(ma[i].midPoint[ma[i].midPointNum - 1]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ma[i].midPointNum - 1]);

for (int ss = 0; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ma[i].midPointNum - 1]);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

}

else

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].endPoint);

for (int ss = 0; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j - 1;

}

}

}

p3 = mp[ma[j].midPoint[ma[j].midPointNum - 1]];

p4 = mp[ma[j].endPoint];

tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)//尾部和尾部

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if(isEqual(tres,p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

pArray.Add(ma[j].midPoint[ma[j].midPointNum-1]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].endPoint);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

}

else if(isEqual(tres,p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ma[i].midPointNum-1]);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[i].midPoint[ma[i].midPointNum - 1]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ma[i].midPointNum - 1]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ma[i].midPointNum - 1]);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ma[i].midPointNum - 1]);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

}

else

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].endPoint);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j-1;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum-1]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j - 1;

}

}

}

//后面对中间

for (int s = 0; s < ma[j].midPointNum - 1; s++)

{

p3 = mp[ma[j].midPoint[s]];

p4 = mp[ma[j].midPoint[s + 1]];

tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < s+1; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int ss = s+1; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

pArray.Add(ma[j].midPoint[s]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[s]);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < s + 1; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int ss = s; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

}

else if(isEqual(tres,p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ma[i].midPointNum-1]);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < s + 1; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[i].midPoint[ma[i].midPointNum - 1]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ma[i].midPointNum - 1]);

for (int ss = s + 1; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int ss = 0; ss < ma[i].midPointNum; ss++)

{

pArray.Add(ma[i].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ma[i].midPointNum - 1]);

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < s + 1; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int ss = s; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

}

else

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < s + 1; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].endPoint);

for (int ss = s + 1; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j - 1;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int ss = 0; ss < s + 1; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int ss = s; ss < ma[j].midPointNum; ss++)

{

pArray.Add(ma[j].midPoint[ss]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

return j - 1;

}

}

}

}

//还未检查

//中间

for (int ss = 0; ss < ma[i].midPointNum-1;ss++ )

{

p1 = mp[ma[i].midPoint[ss]];

p2 = mp[ma[i].midPoint[ss+1]];

p3 = mp[ma[j].startPoint];

p4 = mp[ma[j].midPoint[0]];

tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)//中间和开头

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int s = 0; s < ss+1; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int s = ss + 1; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if(isEqual(tres,p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int s = 0; s < ss + 1; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[j].startPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int s = ss + 1; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

}

else if(isEqual(tres,p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int s = 0; s < ss+1; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int s = ss + 1; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

pArray.Add(ma[i].midPoint[ss]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ss]);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if(isEqual(tres,p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int s = 0; s < ss + 1; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int s = ss + 1; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

}

}

p3 = mp[ma[j].midPoint[ma[j].midPointNum - 1]];

p4 = mp[ma[j].endPoint];

tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)//中间和结尾

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int s = 0; s < ss + 1; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int s = ss + 1; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p3))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int s = 0; s < ss + 1; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[j].midPoint[ma[j].midPointNum-1]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

for (int s = ss + 1; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[ma[j].midPointNum - 1]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int s = 0; s < ss + 1; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].endPoint);

for (int s = ss + 1; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

}

else if (isEqual(tres, p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int s = 0; s < ss + 1; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int s = ss + 1; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

pArray.Add(ma[i].midPoint[ss]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ss]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int s = 0; s < ss + 1; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int s = ss + 1; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int s = 0; s < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[s]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ma[i].midPointNum-1]);

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int s = 0; s < ss + 1; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int s = ss + 1; s < ma[i].midPointNum; s++)

{

pArray.Add(ma[i].midPoint[s]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(i);

return i;

}

}

}

for (int s = 0; s < ma[j].midPointNum - 1; s++)

{

p3 = mp[ma[j].midPoint[s]];

p4 = mp[ma[j].midPoint[s + 1]];

tres = CalIntersect(p1, p2, p3, p4);

if (tres != null)//中间对中间

{

if (!isEqual(tres, p1) && !isEqual(tres, p2))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

int indexMP = addOnePointToMP(tres);

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int sk = 0; sk < ss + 1; sk++)

{

pArray.Add(ma[i].midPoint[sk]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int sk = ss + 1; sk < ma[i].midPointNum; sk++)

{

pArray.Add(ma[i].midPoint[sk]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int sk = 0; sk < s+1; s++)

{

pArray.Add(ma[j].midPoint[sk]);

}

pArray.Add(indexMP);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(indexMP);

for (int sk = s+1; sk < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[sk]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int sk = 0; sk < ss + 1; sk++)

{

pArray.Add(ma[i].midPoint[sk]);

}

pArray.Add(ma[j].midPoint[s]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].midPoint[s]);

for (int sk = ss + 1; sk < ma[i].midPointNum; sk++)

{

pArray.Add(ma[i].midPoint[sk]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int sk = 0; sk < s + 1; s++)

{

pArray.Add(ma[j].midPoint[sk]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int sk = s; sk < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[sk]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

}

else if (isEqual(tres, p1))

{

if (!isEqual(tres, p3) && !isEqual(tres, p4))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int sk = 0; sk < ss + 1; sk++)

{

pArray.Add(ma[i].midPoint[sk]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int sk = ss; sk < ma[i].midPointNum; sk++)

{

pArray.Add(ma[i].midPoint[sk]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int sk = 0; sk < s + 1; s++)

{

pArray.Add(ma[j].midPoint[sk]);

}

pArray.Add(ma[i].midPoint[ss]);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[i].midPoint[ss]);

for (int sk = s+1; sk < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[sk]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

else if (isEqual(tres, p3))

{

pArray.Clear();

pArray.Add(ma[i].startPoint);

for (int sk = 0; sk < ss + 1; sk++)

{

pArray.Add(ma[i].midPoint[sk]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int sk = ss; sk < ma[i].midPointNum; sk++)

{

pArray.Add(ma[i].midPoint[sk]);

}

pArray.Add(ma[i].endPoint);

ma.Add(new MyArc(pArray));

pArray.Clear();

pArray.Add(ma[j].startPoint);

for (int sk = 0; sk < s + 1; s++)

{

pArray.Add(ma[j].midPoint[sk]);

}

ma.Add(new MyArc(pArray));

pArray.Clear();

for (int sk = s; sk < ma[j].midPointNum; s++)

{

pArray.Add(ma[j].midPoint[sk]);

}

pArray.Add(ma[j].endPoint);

ma.Add(new MyArc(pArray));

ma.RemoveAt(j);

ma.RemoveAt(i);

return i;

}

}

}

}

}

return j;

}

/// <summary>

/// 生成多边形

/// </summary>

private void trackPolygon()

{

double EnvMaxx = double.MinValue, EnvMinx = double.MaxValue, EnvMaxy = double.MinValue, EnvMiny = double.MaxValue;

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

{

if (EnvMaxx < data[i].getX())

EnvMaxx = data[i].getX();

if (EnvMinx > data[i].getX())

EnvMinx = data[i].getX();

if (EnvMaxy < data[i].getY())

EnvMaxy = data[i].getY();

if (EnvMiny > data[i].getY())

EnvMiny = data[i].getY();

}

int[] trackTimes = new int[ma.Count];//记录某弧段被用了几次，是顺时针还是逆时针

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

trackTimes[i] = 0;

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

{

if (mp[i].lineNum > 1&&isInEnv(mp[i],EnvMaxx,EnvMinx,EnvMaxy,EnvMiny))//是内部结点

{

//将结点拓扑关联的线段按照角度从小到大排列

for (int j = 0; j < mp[i].lineNum - 1; j++)

{

for (int k = j + 1; k < mp[i].lineNum; k++)

{

if (calAngle(i,j) > calAngle(i,k))

{

int tt = mp[i].relLine[j];

mp[i].relLine[j] = mp[i].relLine[k];

mp[i].relLine[k] = tt;

}

}

}

for (int j = 0; j < mp[i].lineNum; j++)

{

if (Math.Abs(trackTimes[convert(mp[i].relLine[j])]) != 2 && Math.Abs(trackTimes[convert(mp[i].relLine[(j + 1) % mp[i].lineNum])]) != 2)

{

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

int tempArc = mp[i].relLine[(j + 1) % mp[i].lineNum];

while ((tempArc >= 0 && trackTimes[convert(tempArc)] <= 0 && trackTimes[convert(tempArc)] > -2) || (tempArc < 0 && trackTimes[convert(tempArc)] >= 0 && trackTimes[convert(tempArc)]<2))

{

alist.Add(tempArc);

if (trackTimes[convert(tempArc)] > 0)

trackTimes[convert(tempArc)]++;

else if (trackTimes[convert(tempArc)] < 0)

trackTimes[convert(tempArc)]--;

else

trackTimes[convert(tempArc)] = tempArc >= 0 ? 1 : -1;

int nextP;

if (tempArc >= 0)//找末结点

{

nextP = ma[tempArc].endPoint;

ma[tempArc].rightPoly = mpol.Count;

}

else//找首节点

{

nextP = ma[convert(tempArc)].startPoint;

ma[convert(tempArc)].leftPoly = mpol.Count;

}

for (int l = 0; l < mp[nextP].lineNum - 1; l++)

{

for (int k = l + 1; k < mp[nextP].lineNum; k++)

{

if (calAngle(nextP,l) > calAngle(nextP,k))

{

int tt = mp[nextP].relLine[l];

mp[nextP].relLine[l] = mp[nextP].relLine[k];

mp[nextP].relLine[k] = tt;

}

}

}

for(int k=0;k<mp[nextP].lineNum;k++)

{

if(Math.Abs(convert(tempArc))==Math.Abs(convert(mp[nextP].relLine[k])))

{

tempArc = mp[nextP].relLine[(k + 1) % mp[nextP].lineNum];

break;

}

}

}

if (alist.Count != 0)

mpol.Add(new MyPolygon(alist));

}

}

}

}

//生成多边形的拓扑关系

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

{

for(int j=0;j<mpol[i].arcList.Count;j++)

{

if(ma[convert(mpol[i].arcList[j])].leftPoly==i&&ma[convert(mpol[i].arcList[j])].rightPoly!=-1)

{

mpol[i].adjPoly.Add(ma[convert(mpol[i].arcList[j])].rightPoly);

mpol[i].adjPolyNum++;

}

if (ma[convert(mpol[i].arcList[j])].rightPoly == i && ma[convert(mpol[i].arcList[j])].leftPoly != -1)

{

mpol[i].adjPoly.Add(ma[convert(mpol[i].arcList[j])].leftPoly);

mpol[i].adjPolyNum++;

}

}

}

}

/// <summary>

/// 绘制拓扑数据

/// </summary>

private void showTopo()

{

pictureBox1.Refresh();

Graphics g = pictureBox1.CreateGraphics();

SolidBrush myBrush = new SolidBrush(Color.Red);

// Pen myPen = new Pen(Color.Red);

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

{

myBrush.Color = Color.FromArgb(125,(i\*35)%256,(i\*76)%256,(i\*90)%256);

int pointNumPoly = 0;

for (int j = 0; j < mpol[i].arcList.Count; j++)

{

pointNumPoly += ma[convert(mpol[i].arcList[j])].midPointNum + 1;

}

Point[] tmp = new Point[pointNumPoly];

pointNumPoly = 0;

for (int j = 0; j < mpol[i].arcList.Count; j++)

{

if (mpol[i].arcList[j] >= 0)

{

tmp[pointNumPoly] = new Point((int)((mp[ma[convert(mpol[i].arcList[j])].startPoint].x - minx) / scale + margin), (int)(pictureBox1.Height - (mp[ma[convert(mpol[i].arcList[j])].startPoint].y - miny) / scale - margin));

pointNumPoly++;

for (int k = 0; k < ma[convert(mpol[i].arcList[j])].midPointNum; k++)

{

tmp[pointNumPoly] = new Point((int)((mp[ma[convert(mpol[i].arcList[j])].midPoint[k]].x - minx) / scale + margin), (int)(pictureBox1.Height - (mp[ma[convert(mpol[i].arcList[j])].midPoint[k]].y - miny) / scale - margin));

pointNumPoly++;

}

}

else

{

tmp[pointNumPoly] = new Point((int)((mp[ma[convert(mpol[i].arcList[j])].endPoint].x - minx) / scale + margin), (int)(pictureBox1.Height - (mp[ma[convert(mpol[i].arcList[j])].endPoint].y - miny) / scale - margin));

pointNumPoly++;

for (int k = ma[convert(mpol[i].arcList[j])].midPointNum-1; k >=0; k--)

{

tmp[pointNumPoly] = new Point((int)((mp[ma[convert(mpol[i].arcList[j])].midPoint[k]].x - minx) / scale + margin), (int)(pictureBox1.Height - (mp[ma[convert(mpol[i].arcList[j])].midPoint[k]].y - miny) / scale - margin));

pointNumPoly++;

}

}

}

g.FillPolygon(myBrush, tmp);

}

// myPen.Dispose();

myBrush.Dispose();

g.Dispose();

}

## 3.8计算多边形周长和面积

计算多边形周长和面积的代码如下。

//计算多边形周长

public double GetPerimeter(List<MyPoint> mp,List<MyArc> ma)

{

double res = 0;

//假设点线面都按照ID升序排列

for(int i=0;i<arcNum;i++)

{

res += ma[convert(arcList[i])].GetLength(mp);

}

return res;

}

//计算多边形的面积

public double GetArea(List<MyPoint> mp, List<MyArc> ma)

{

double res=0;

MyPoint tmp = mp[ma[convert(arcList[arcList.Count - 1])].startPoint];

foreach(int i in arcList)

{

res += TriArea(tmp, mp[ma[convert(i)].startPoint], mp[ma[convert(i)].endPoint]);

}

return res;

}

//计算三角形面积

public double TriArea(MyPoint p1,MyPoint p2,MyPoint p3)

{

return 0.5\*Math.Abs((p2.x-p1.x)\*(p3.y-p1.y)-(p2.y-p1.y)\*(p3.x-p1.x));

}