# 

# 空间分析算法

# 实习报告

**班级： 地理信息科学18-3**

**姓名： 李国荣**

**学号： 1801020815**

**提交日期： 2021年1月7日**

|  |  |  |  |
| --- | --- | --- | --- |
| 实习时间： | 17周一周 | 实习地点： | J14-404等 |
| 实习评语：指导教师签名：评阅时间： | | | |

# 实验任务和要求：

运用熟悉的编程语言（C#、C++、Python、Java、Matlab等）进行如下算法的设计与开发，要求程序运行稳定、运算结果可靠，并以图形化界面实现结果输出（实验报告中出现的内容这里不作要求，请删除）。

1. 空间实体量测算法设计

（1）请实现两个空间实体间的距离计算：点点距离、点线距离、点面距离、线线距离、线面距离、面面距离；

（2）请实现球面上两点间的距离计算；

（3）请实现面状实体的周长、面积、中心、重心、质心的计算，以及紧凑指数等形态参数的计算；

（4）请实现DEM表面的表面积以及指定高度的体积计算；

（5）请实现现状地物的曲率和弯曲度计算；

1. 线状数据的压缩算法

请运用道格拉斯普克算法进行线目标要素的数据压缩算法设计，以图形化方法实现压缩前、后的线状要素。

1. 不规则三角网（TIN）生成算法

基于离散点数据，运用Delaunay三角网生成准则，使用三角网生长法或逐点插入算法进行 TIN的生成。

1. 累计表面生成及应用算法设计

基于一幅50\*50的网格数据（每个格点值代表通过该网格的花费），随机设定一个网格作为起点，计算该起点到达其它网格的最小花费，进而生成累计表面矩阵。

1. 矢量数据叠加分析算法

（1）点到曲线最短距离计算；

（2）点在多边形内外的判定计算；（实验已做）

（3）曲线与曲线的求交运算；

（4）曲线与面的求交运算；

（5）面面的求交运算。

1. 栅格数据叠加分析算法

（1）进行栅格数据的局部变换、邻域变换、分带变换、全局变换等计算；

（2）基于地图代数，进行两幅栅格数据间的叠加分析计算。

1. 矢量数据的缓冲区分析计算

（1）进行单点、多点的缓冲区生成；（实验已做）

（2）分别基于角平分线法和凸角圆弧法进行线状要素的缓冲区生成；（实验已做）

（3）进行面状要素的缓冲区生成。

1. 基于DEM的地形特征提取算法

以某区域的DEM数据为基础，进行多种地形特征提取的算法设计，主要包括：地形因子计算（坡度、坡向、表面积、体积、坡度坡向变率、曲率、粗糙度、凸凹系数）、地形特征点的提取（山顶点、凹陷点、脊点、谷点、马鞍点、平地点）以及流长、汇水量、流域盆地等的计算网络分析中的最小代价生成树算法。

9. 网络分析中的最小代价生成树算法

基于一幅加权无向网络图，运用破回路算法，进行最小代价生成树的生成，并进行图形表达。

# 开发环境及系统运行环境

## 开发环境

Windows 10, Eclipse IDE for Java Developers, Java JDK 1.8

## 运行环境

Java运行环境（Java Runtime Environment）

# 算法原理及实现步骤

注：本次实习我只实现了部分实习要求的内容，但我加入了部分实验内容集成在一个系统中，主要包括8个内容： 点线面绘制（包括周长面积重心的计算）、道格拉斯-扑克算法、矢量缓冲区绘制、插值算法（DEM插值可视化）、地形特征点提取（坡度和坡向可视化）、不确定性分析（正态云生成云滴可视化和分形可视化）、K均值聚类（K-Means可视化）、TIN绘制。

## 空间实体量测算法设计

**3.1.1画图板的主界面设计（该代码包括以下内容的界面设计）**

以下内容只包括核心部分代码：

// 创建工具条

jtb = **new** JToolBar();

//创建菜单条

jmb = **new** JMenuBar();

menu1 = **new** JMenu("插值算法(F)");

//创建文本区域条

jta = **new** JTextArea();

// 将菜单添加到菜单栏上

menu1.add(fanjvli);

menu2.add(podu);

// 将菜单添加到菜单条上

jmb.add(menu1);

// 将菜单添加到窗体上

**this**.setJMenuBar(jmb);

**this**.setBackground(Color.***WHITE***);

setVisible(**true**);}

3.1.2创建监听器类，实现绘图与计算

要想实现对图形的绘制与计算，需要首先把图形坐标信息记录下来，因此需要在程序中建立链表类或数组类来记录绘制的图形的坐标信息。实现过程：通过鼠标的左键点击函数获取点击位置(ex,ey)，然后将该值记入对应的数组中，分别存储点线面坐标，并实现对应的方法。

//计算周长算法

**for** (**int** i = 0; i <= pol.size()-1; i++) {**if**(i<pol.size()-1) {

perimeter1 += Math.*sqrt*(Math.*pow*(pol.get(i + 1).x - pol.get(i).x, 2) + Math.*pow*(pol.get(i + 1).y - pol.get(i).y, 2));}

**if**(i==pol.size()-1) {

perimeter = perimeter1+Math.*sqrt*(Math.*pow*(pol.get(i).x - pol.get(0).x, 2) + Math.*pow*(pol.get(i).y - pol.get(0).y, 2));}}

//利用二维矩阵计算面积

**for** (**int** i = 0; i <= pol.size() - 1; i++) {

**if**(i<pol.size()-1) {

location[i][0]=pol.get(i).x;location[i][1]=pol.get(i).y;}

**if**(i==pol.size()-1){

location[pol.size()-1][0]=pol.get(pol.size()-1).x;

location[pol.size()-1][1]=pol.get(pol.size()-1).y;}}

location[pol.size()][0] = location[0][0];location[pol.size()][1] = location[0][1];**for** (**int** j = 0; j <=pol.size()-1 ; j++) {sum+=(location[j][0]\*location[j+1][1]-location[j+1][0]\*location[j][1]);}

area=sum/2;

//计算重心算法

**for** (**int** i = 0; i <= pol.size()-1; i++) {

**if**(i<pol.size()-1) {

gra1.x= pol.get(i + 1).x + pol.get(i).x;

gra1.y= pol.get(i + 1).y + pol.get(i).y;}

**if**(i==pol.size()-1) {gra2.x= pol.get(i).x + pol.get(0).x;

gra2.y= pol.get(i).y + pol.get(0).y;}}

gra.x=(gra2.x)/pol.size();gra.y=(gra2.y)/pol.size();}}}

3.1.3给按钮加上鼠标监听

// 添加面积按钮响应事件

getArea.addActionListener(**new** ActionListener() {})

// 添加周长按钮响应事件

getPerimeter.addActionListener(**new** ActionListener() {})

// 添加重心按钮响应事件

gravitycenter.addActionListener(**new** ActionListener() {})

## 线状数据的压缩算法

3.2.1道格拉斯扑克算法原理

对曲线的首末点虚连一条直线，求曲线上所有点与直线的距离，并找出最大距离值dmax，用dmax与事先给定的阈值D相比

1）若dmax<D，则将这条曲线上的中间点全部舍去；则该直线段作为曲线的近似，该段曲线处理完毕。

2）若dmax≥D，保留dmax对应的坐标点，并以该点为界，把曲线分为两部分，对这两部分重复使用该方法，即重复1），2）步，直到所有dmax均<D，即完成对曲线的抽稀。

显然，本算法的抽稀精度也与阈值相关，阈值越大，简化程度越大，点减少的越多，反之，化简程度越低，点保留的越多，形状也越趋于原曲线。

3.2.2添加按钮响应事件

dp.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

String o=JOptionPane.*showInputDialog*("请输入阈值：");

**int** epsilon=Integer.*parseInt*(o);

result = DouglasPeuckerUtil.*DouglasPeucker*(points, epsilon);

g = gf.getGraphics();

g.clearRect(0, 95, 650, 700);

g.setColor(Color.***BLUE***);

**for** (**int** i = 0; i < result.size() - 1; i++) {

Point b1 = result.get(i);

Point b2 = result.get(i + 1);

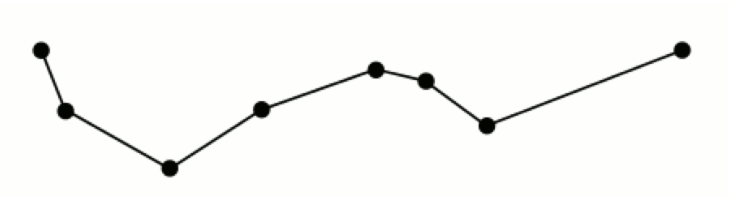
g.drawLine(b1.x+7, b1.y+88, b2.x+7, b2.y+88);

}}

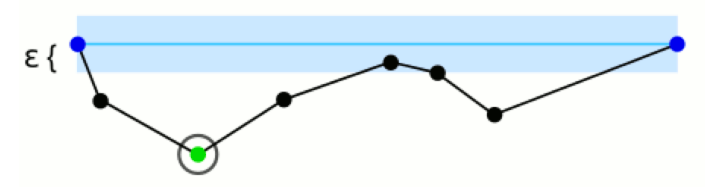
});

3.2.3道格拉斯扑克算法

假设当前我们有一条曲线，它由8各点来描述，如下图所示：



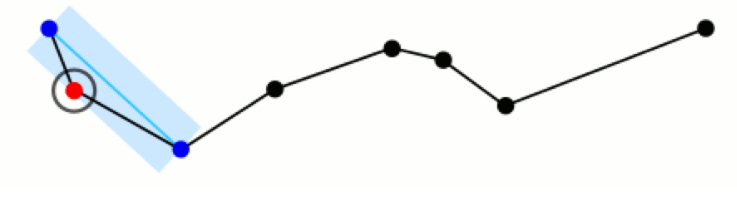
初始曲线是一系列有序的点集，我们需要设定一个距离（阈值）参数 ε > 0。最开始时，在曲线首尾两点A，B之间连接一条直线AB。算法自动将首尾两个点记下（也就是存入结果点集）。得到曲线上离该直线段（AB）距离最大的点C，计算其与AB的距离d，如下图所示:

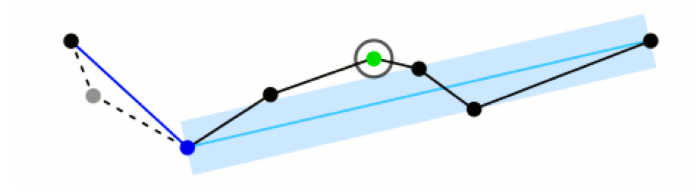


如果用直线段AB来作为原曲线的近似表示，那么点C显然是位于曲线上，离AB最远的点。现在比较距离 d 与预先给定的阈值 ε 的大小，如果小于 ε，则表明任何当前没有被记下的点都可以被丢弃了，因为用已经得到的直线段作为曲线的近似，不会比 ε 更早，即该段曲线处理完毕。

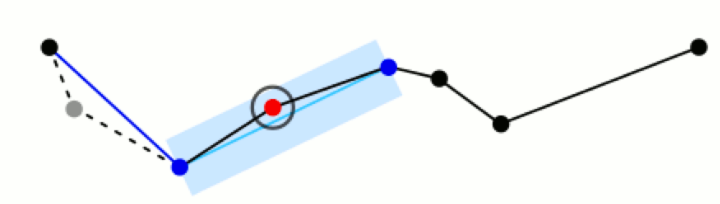
如果距离 d 大于阈值，则用C将曲线分为两段AC和BC，并将点C记下。

然后分别对已经得到两段曲线递归地进行上述处理。

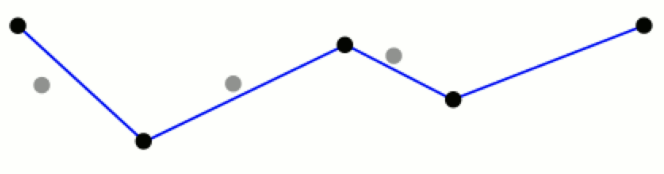




当所有曲线都处理完毕时，依次连接各个分割点形成的折线，即可以作为曲线的近似。就本例而言，接下来的处理步骤如下图所示：

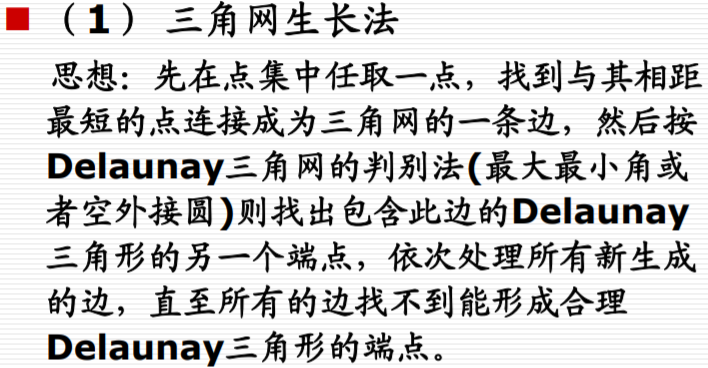


最终我们得到的（用更少的点表示的）近似曲线如下:



## 不规则三角网TIN的生成算法

3.3.1三角网生长算法原理



3.3.2绘图界面设计及画布

图形界面：

// 打开的DEM坐标文件的路径

**private** String filePath;

\* 绘图区域

**private** DrawCanvas drawArea;

\* 主窗口菜单项

**private** JMenuBar mainMenuBar;

//主窗口构造方法

//窗口初始化，在构造方法中被调用

\* 文件菜单项

JMenu fileMenu = **new** JMenu("文件");

JMenuItem importDateItem = **new** JMenuItem("导入DEM数据点");

importDateItem.addActionListener(**new** OpenFileListener(**this**));

fileMenu.add(importDateItem);

\* 绘图菜单项

JMenu drawMenu = **new** JMenu("绘图");

JMenuItem drawTINItem = **new** JMenuItem("绘制三角网");

drawTINItem.addActionListener(**new** DrawTINListener(**this**));

drawMenu.add(drawTINItem);

\* 工具菜单项

画布：

\* 单例模式

**synchronized** **public** **static** DrawCanvas newInstance(){

**if**(*mInstance* == **null**){

*mInstance* = **new** DrawCanvas();

}

**return** *mInstance* ;

}

\* 绘制坐标箭头

\* 绘制坐标点

\* 计算并绘制TIN

\* 计算TIN时，开启一个新的线程，避免UI线程被阻塞

\* 绘制矩形框

\* 展绘坐标点

\*绘制三角网

\* 得到DEM点的范围和中心点

3.3.3构造函数Point，Line，Triangle

点：

**public** **class** Point {

**public** **double** X = 0;

**public** **double** Y = 0;

**public** Point(){}

\* 构造方法

**public** Point(**double** X, **double** Y){

**this**.X = X;

**this**.Y = Y;}}

线：

**public** **class** Line{

\* 起点 **private** Point startPoint;

\* 终点 **private** Point endPoint;

\* 使用次数 **private** **int** useCount;

**public** Line(){}

**public** Line(Point startPoint , Point endPoint){

**this**.startPoint = startPoint;

**this**.endPoint = endPoint;

**this**.useCount = 0;}

三角形：

**public** **class** Triangle {

\* 三角形的三条边

**private** Line firstLine;

**private** Line secondLine;

**private** Line thirdLine;

\* 构造方法

**public** Triangle(Line firstLine, Line secondLine, Line thirdLine) {

**this**.firstLine = firstLine;

**this**.secondLine = secondLine;

**this**.thirdLine = thirdLine;}

}

3.3.4 DEM数据文件输入流

向数组中添加DEM点的坐标：

DEMPoints[j++] = **new** Point(Double.*parseDouble*(lines[0]),Double.*parseDouble*(lines[1]));

3.3.5 数据输入界面设计

\* 主窗口界面

**private** DrawFrame mDrawFrame;

**public** OpenFileListener(DrawFrame mDrawFrame){

**this**.mDrawFrame = mDrawFrame;}

3.3.6 绘制TIN

\* 计算并绘制三角网

**public** **void** draw(Graphics g, **double** xCenter, **double** yCenter , **double** scaleSize, **int** width, **int** height){

//点的颜色为绿色

g.setColor(**new** Color(0,0,255));

//绘制所有的边，即可形成三角网

**for**(**int** j = 0 ; j<lineList.size(); j++){

lineList.get(j).draw(g,xCenter,yCenter,scaleSize,width,height);}

}

\* 构TIN，在线程中被调用

\* 改变边的方向，即交换起始点

\* 判断当前点是否在当前直线的左侧

**public** **boolean** isOnLeft(Point currentPoint , Line currentLine){}

3.3.7 辅助部件：坐标箭头，进度条，图形放缩

坐标箭头：

//画线

g.drawLine(sx, sy, ex, ey);

//画箭头

**int**[] xPoints=**new** **int**[]{ex,x3,x4};

**int**[] yPoints=**new** **int**[]{ey,y3,y4};

g.drawPolygon(xPoints,yPoints,3);

进度条：

mJLabel = **new** JLabel("目前已完成进度：0%",JLabel.***CENTER***);

mJFrame.add(**new** JLabel("三角网生成进度："),BorderLayout.***NORTH***);

mJFrame.add(mJProgressBar,BorderLayout.***CENTER***);

mJFrame.add(mJLabel,BorderLayout.***SOUTH***);

mJFrame.setVisible(**true**);

\* 刷新进度条

图形放缩：

**放大两倍：**

**public** **void** actionPerformed(ActionEvent e) {

DrawCanvas drawArea = DrawCanvas.*newInstance*();

drawArea.setScaleSize(drawArea.getScaleSize()/2);

drawArea.repaint();}}

**缩小两倍：**

**public** **class** ScaleTwoTimesListener **implements** ActionListener {

**public** **void** actionPerformed(ActionEvent e) {

DrawCanvas drawArea = DrawCanvas.*newInstance*();

drawArea.setScaleSize(drawArea.getScaleSize()\*2);

drawArea.repaint();}}

3.3.8 向主界面中添加按钮响应事件

TIN.addActionListener(**new** ActionListener() {

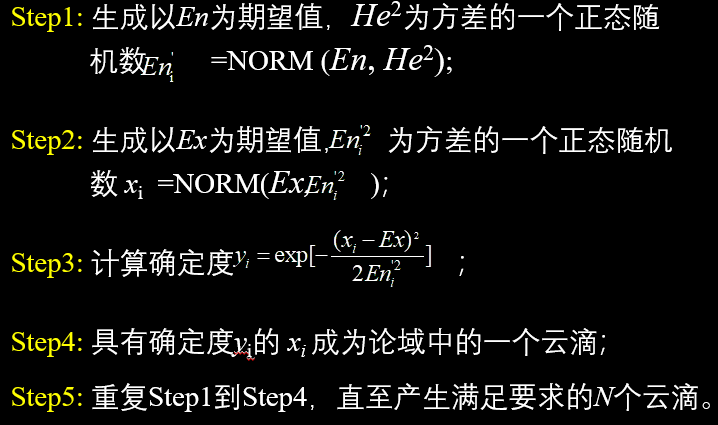
**public** **void** actionPerformed(ActionEvent e) {

**new** DrawFrame();

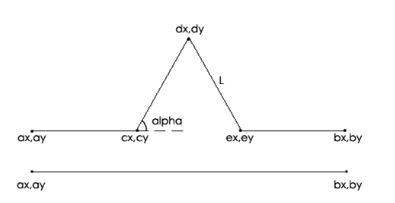
}});}

## 不确定性分析

3.4.1 正态云生成算法



3.4.2分形算法



代码核心：

**public** **synchronized** **void** move(**double** x, **double** y) {

**if** (penIsDown) {

pen.setColor(penColor);

**double** x2 = xpos - x;

**double** y2 = ypos - y;

**int** step;

**for** (**int** i = step - 1; i >= 0; --i) {

**double** xx = x + x2 \* i / step;

**double** yy = y + y2 \* i / step;

*pond*.setClip((**int**)xpos, (**int**)ypos, ***TURTLE\_SIZE*** + 2);

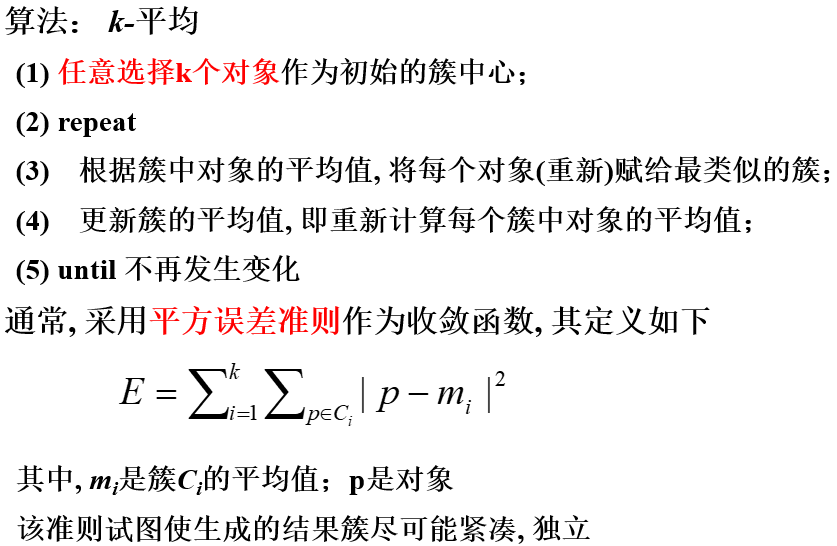
*pond*.setClip((**int**)xx, (**int**)yy, ***TURTLE\_SIZE*** + 2);

pen.drawLine((**int**)xpos, (**int**)ypos, (**int**)xx, (**int**)yy);

xpos = xx; ypos = yy;

repaint();}

## K-Means聚类分析算法



//根据质心，决定当前元组属于哪个簇

**public** **int** clusterOfTuple(Tuple means[],Tuple tuple){

**double** dist=getDistXY(means[0],tuple);

**double** tmp;

**int** label=0;

**for**(**int** i=1;i<*k*;i++){

tmp=getDistXY(means[i],tuple);

**if**(tmp<dist) {dist=tmp;label=i;}}

**return** label;}

//获得当前簇的质心

**public** Tuple getMeans(Vector<Tuple> cluster){

**float** meansX=0,meansY=0;

Tuple t = **new** Tuple();

**for** (**int** i=0;i<cluster.size();i++)

{

meansX+=cluster.get(i).getX();

meansY+=cluster.get(i).getY();}

t.setX(meansX/cluster.size());

t.setY(meansY/cluster.size());

**return** t;}

//根据默认的质心给簇赋值

**for**(i=0;i!=*tuples*.size();++i){

lable=clusterOfTuple(*means*,*tuples*.get(i));

*clusters*[lable].addElement(*tuples*.get(i));}

//根据新的质心获得新的簇

**for**(i=0;i!=*tuples*.size();++i){

lable=clusterOfTuple(*means*,*tuples*.get(i));

*clusters*[lable].add(*tuples*.get(i));}

## 矢量数据的缓冲区分析计算

3.6.1点缓冲区计算

pointBufferButton.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

**int** radius=50;

Graphics g = gf.getGraphics();

Graphics t = gf.getGraphics();

g.setColor(Color.***red***);

t.setColor(Color.***ORANGE***);

**for**(**int** i=0; i<po.size();i++) {

**if**(po.get(i).x!=0&po.get(i).y!=0){

g.fillOval(po.get(i).x+7,po.get(i).y+88, 2, 2);

t.drawOval(po.get(i).x+7-radius/2,po.get(i).y+88-radius/2,radius,radius);}}}});

3.6.2线缓冲区算法

使用凸角圆弧法对所绘制的折线添加线缓冲区：

基本思想：顾名思义，即是在转角外侧用圆弧来代替尖角，内侧仍然使用尖角的方法，生成缓冲区。

实施步骤：

1.        直线性判断，判断相邻三点是否在同一直线上；

2.        折点凸凹性判断，确定转角的地方哪侧使用直线求交，哪侧使用圆弧连接；

3.        凸点圆弧的嵌入，即将转角外侧形成的圆弧和两边的线段相连；

4.        边线关系的判别与处理，岛屿多边形参与缓冲区边界的构成，重叠多边形不参与缓冲区边界的构成；

5.        缓冲区边界的形成，具体是将重叠区域进行合并，绘制外围的边线，包括岛屿多边形的轮廓，形成最终的缓冲区边界。

## 基于DEM插值算法

反距离权重 (IDW) 插值显式假设：彼此距离较近的事物要比彼此距离较远的事物更相似。当为任何未测量的位置预测值时，反距离权重法会采用预测位置周围的测量值。与距离预测位置较远的测量值相比，距离预测位置最近的测量值对预测值的影响更大。反距离权重法假定每个测量点都有一种局部影响，而这种影响会随着距离的增大而减小。由于这种方法为距离预测位置最近的点分配的权重较大，而权重却作为距离的函数而减小，因此称之为反距离权重法。

//反距离加权插值核心算法

**for**(;r<100;r=r+5) {

**for**(**int** i=0;i<sum.size();i++) {

**if**(d[i] < r) {

w = 1/d[i]\*d[i];//距离权重

wh = w\*dh[i];//距离权重\*高程

num++;//num从0开始加，也就是说最多只进行五次加权运算

wz = wz + w;//权重的和

whz = whz + wh;//高程值的和 }

}

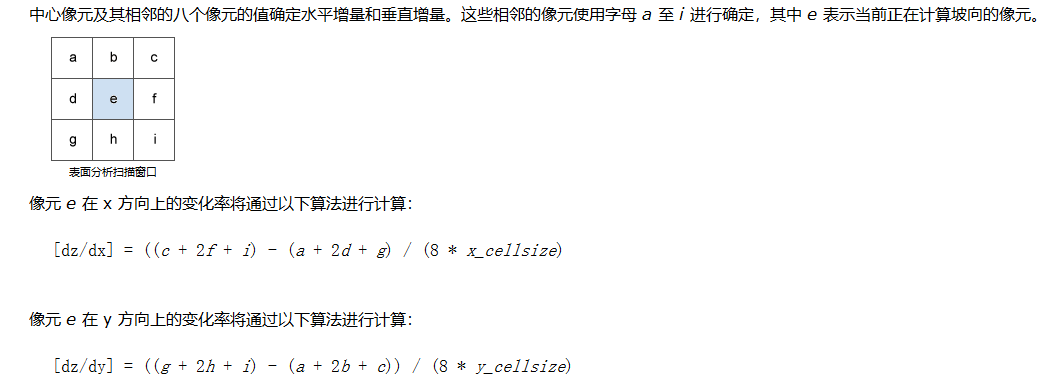
**if**(num>=5)**break**;}

**double** ph = Double.*parseDouble*(df.format(whz/wz));//格网点h

*dem*.add(**new** DEMPoint(px, py, ph));

## 3.8基于DEM地形特征提取算法

3.8.1坡度分析：



ex=((*dem*.get(j+s+c).getH()+2\**dem*.get(j+s+f).getH()+*dem*.get(j+s+i).getH())-(*dem*.get(j+s+a).getH()+2\**dem*.get(j+s+d).getH()+*dem*.get(j+s+g).getH()))/40;

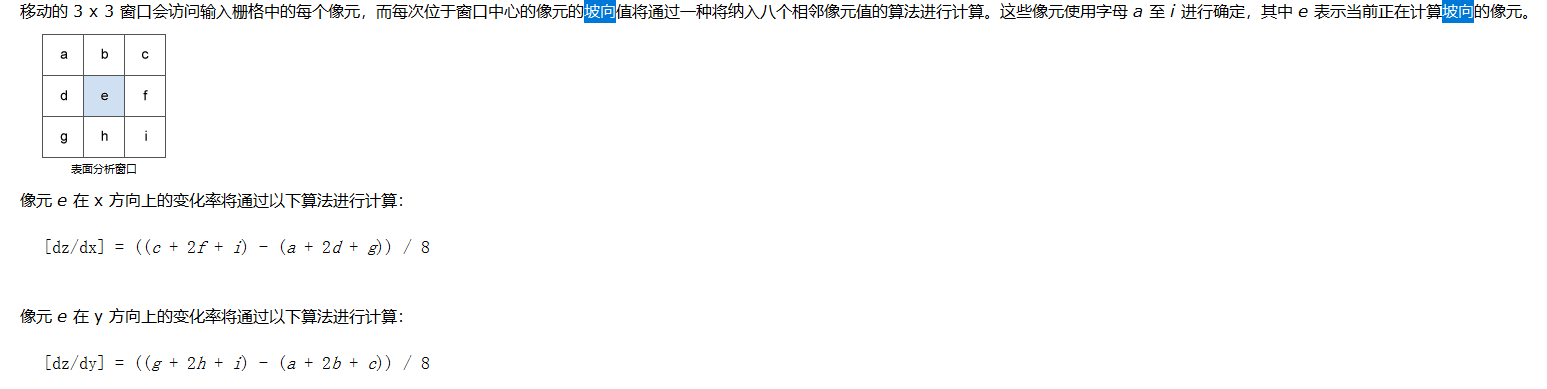
ey=((*dem*.get(j+s+g).getH()+2\**dem*.get(j+s+h).getH()+*dem*.get(j+s+i).getH())-(*dem*.get(j+s+a).getH()+2\**dem*.get(j+s+b).getH()+*dem*.get(j+s+c).getH()))/40;

**double** rise\_run=Math.*sqrt*(ex\*ex+ey\*ey);

**double** slope\_degree= Math.*atan*(rise\_run)\*57.29578;

*slope\_degrees*.add(**new** SlopePoint(slope\_degree));

3.8.2坡向分析：



ex=((*dem*.get(j+s+c).getH()+2\**dem*.get(j+s+f).getH()+*dem*.get(j+s+i).getH())-(*dem*.get(j+s+a).getH()+2\**dem*.get(j+s+d).getH()+*dem*.get(j+s+g).getH()))/8;

ey=((*dem*.get(j+s+g).getH()+2\**dem*.get(j+s+h).getH()+*dem*.get(j+s+i).getH())-(*dem*.get(j+s+a).getH()+2\**dem*.get(j+s+b).getH()+*dem*.get(j+s+c).getH()))/8;

**double** aspect=57.29578 \*Math.*atan2*(ey,ex);

**double** cell1=0;

**if**(aspect<0) {

cell1 = 90.0 - aspect;

}

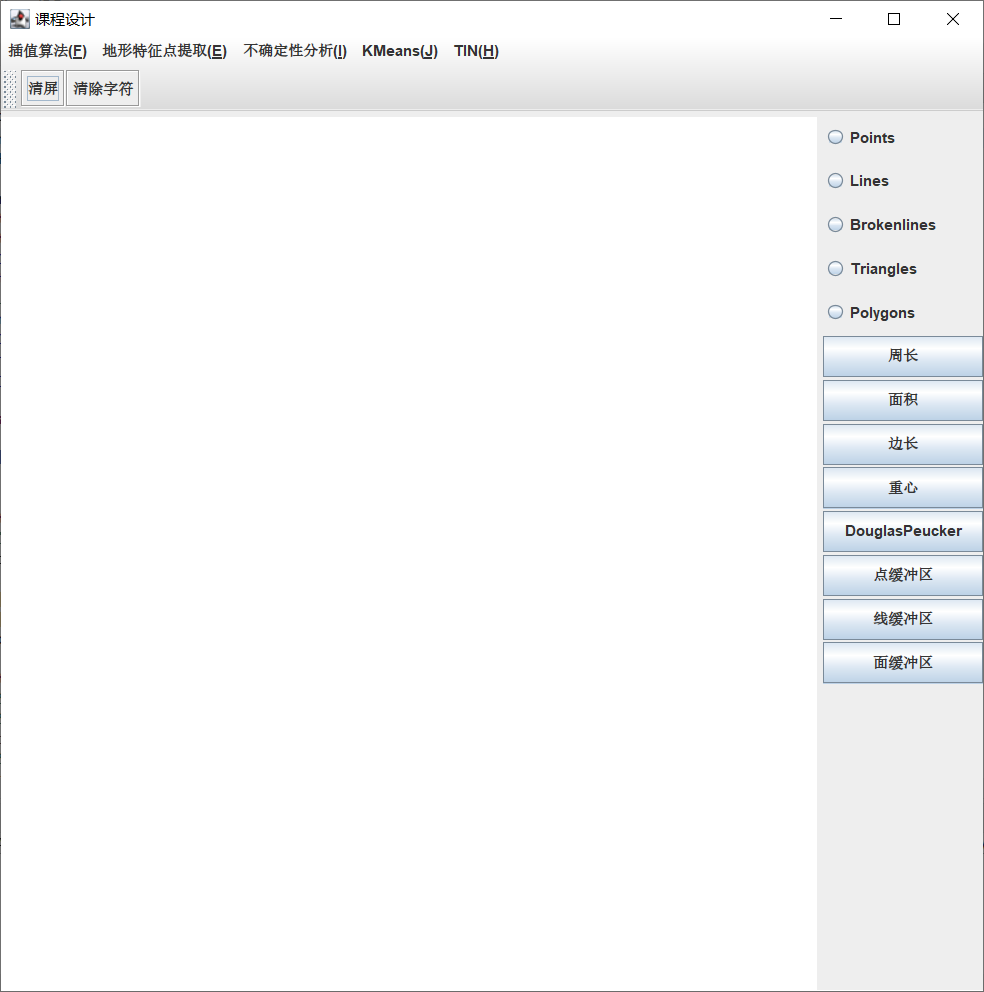
**else** **if**(aspect>=0) {

cell1=aspect;

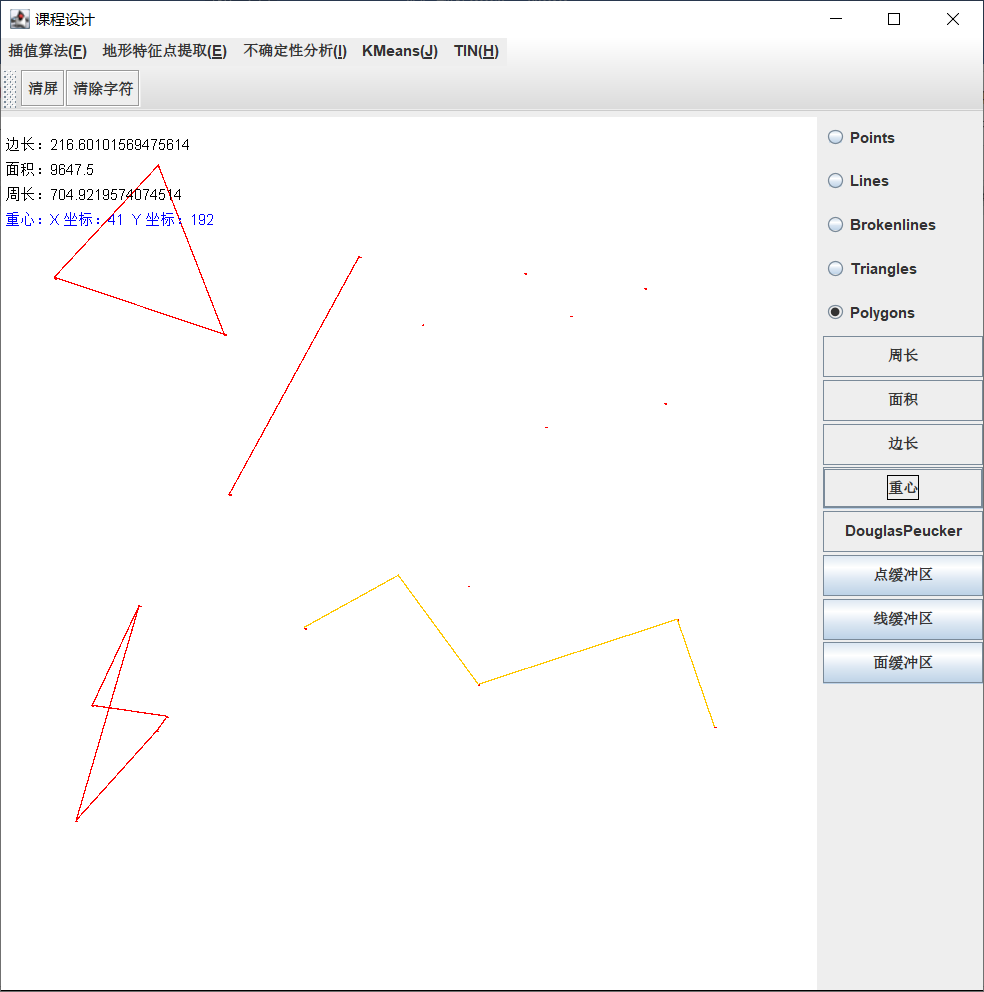
}

# 软件操作步骤及效果

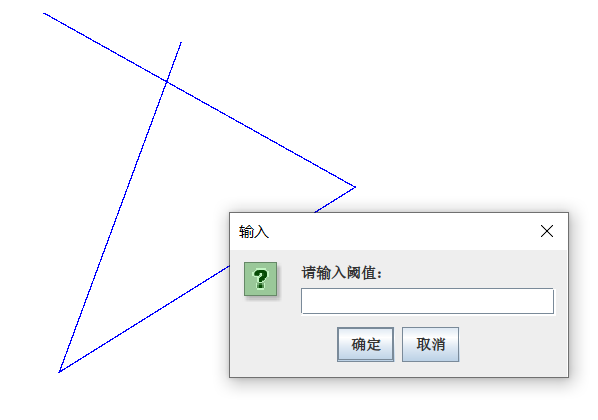
软件主界面：



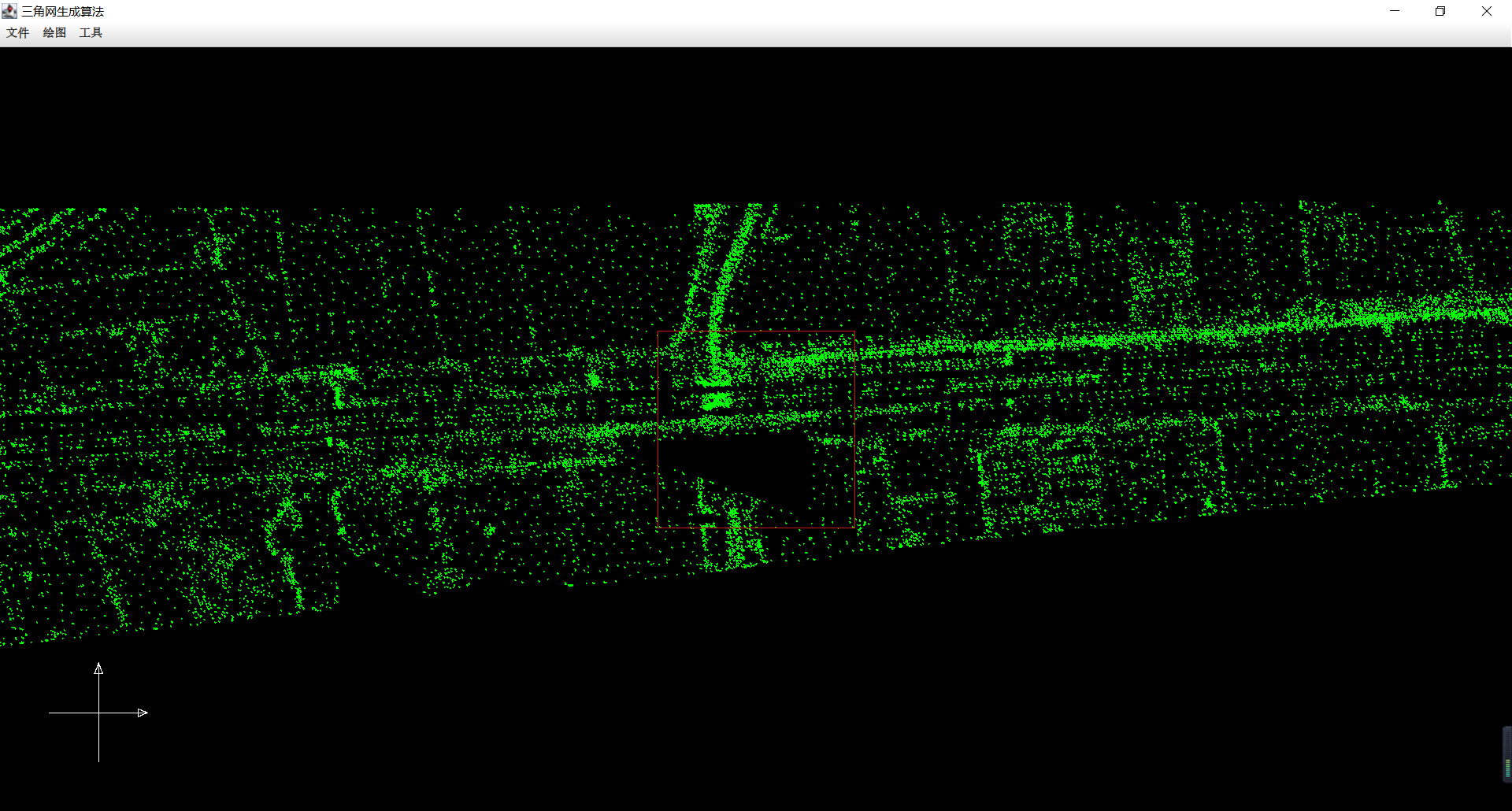
## 空间实体量测算法设计

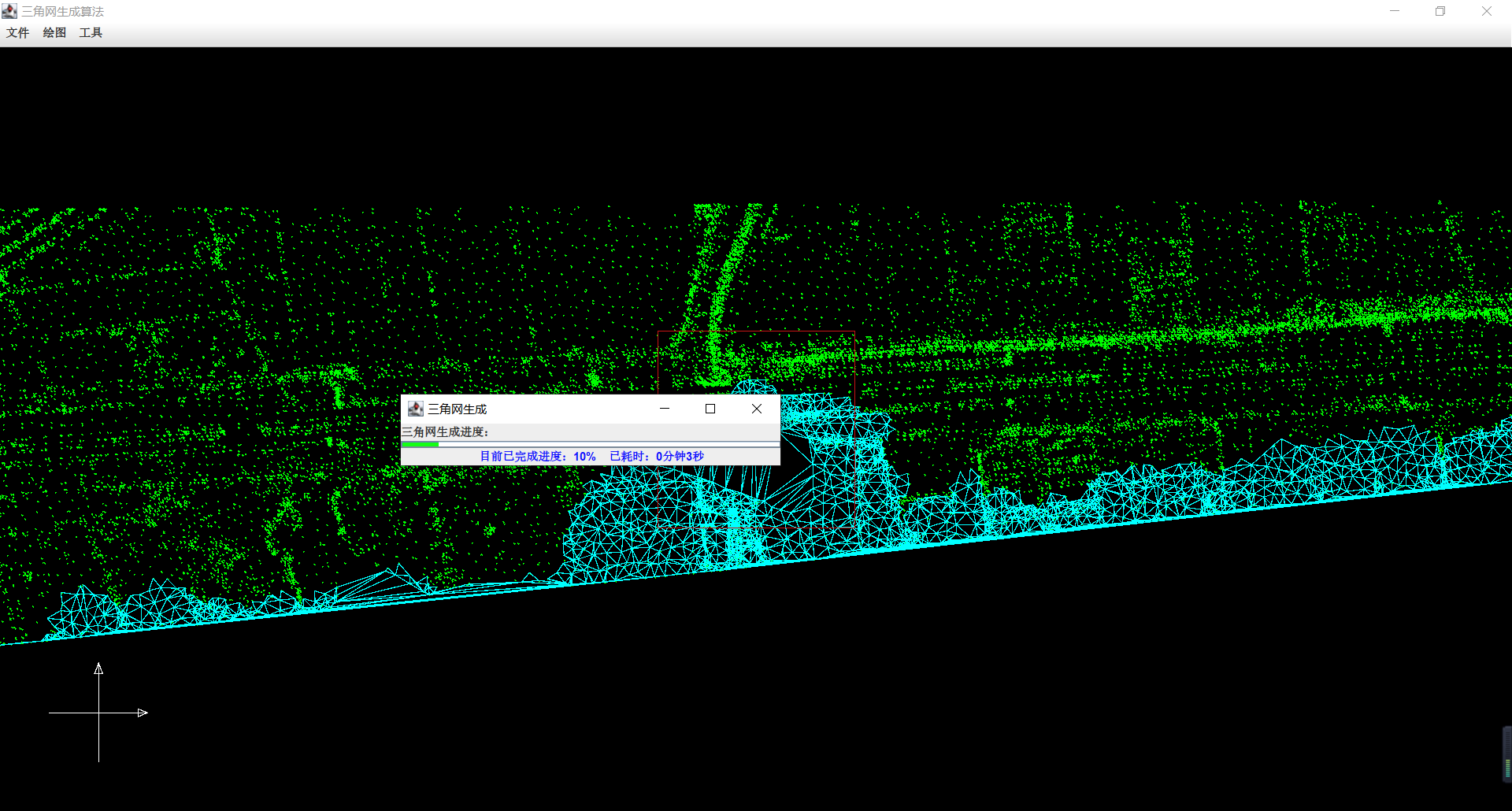


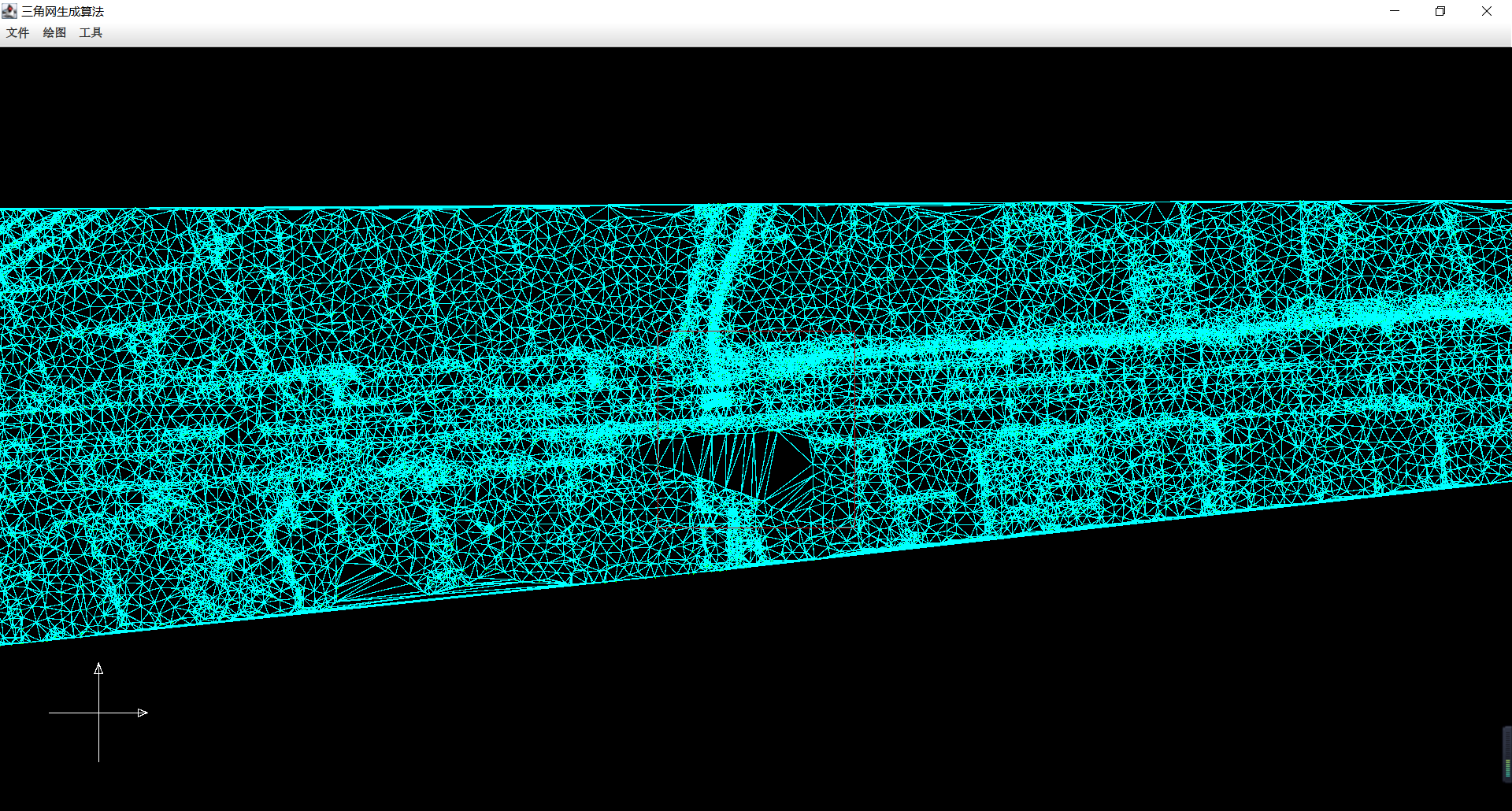
## 线状数据的压缩算法

## 不规则三角网（TIN）生成算法

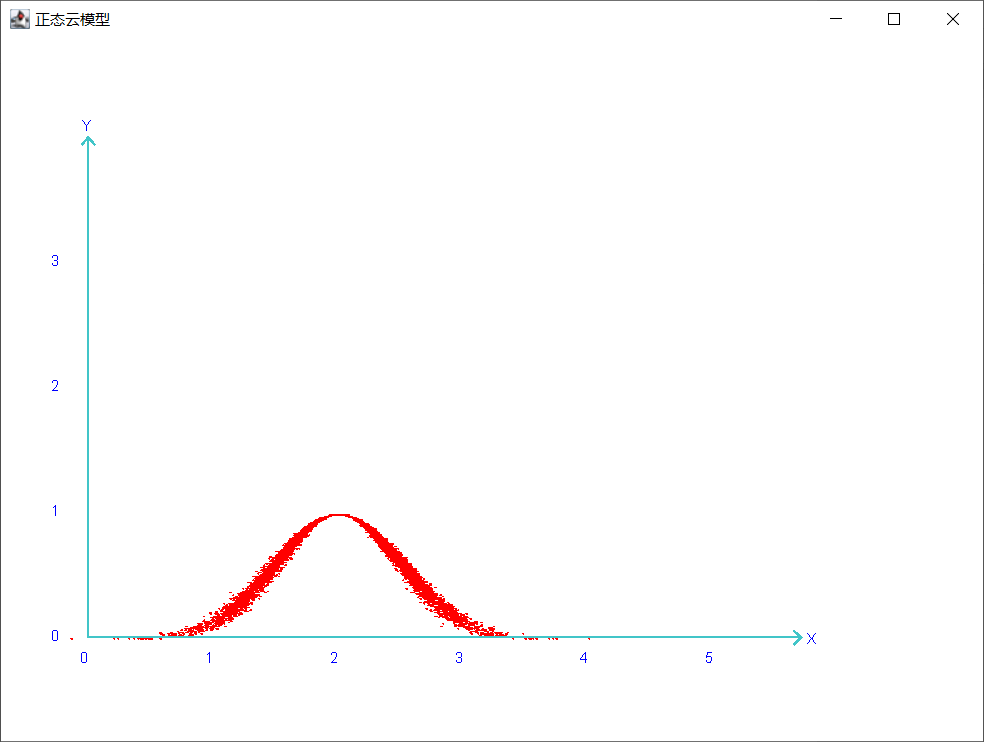




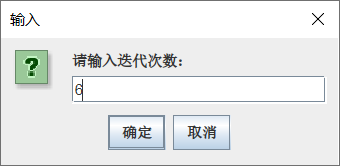
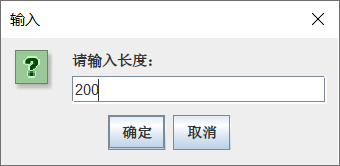


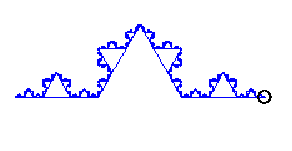
## 不确定性分析

4.4.1正态云生成

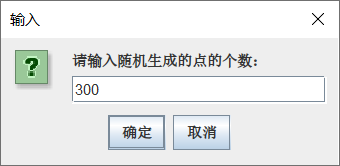


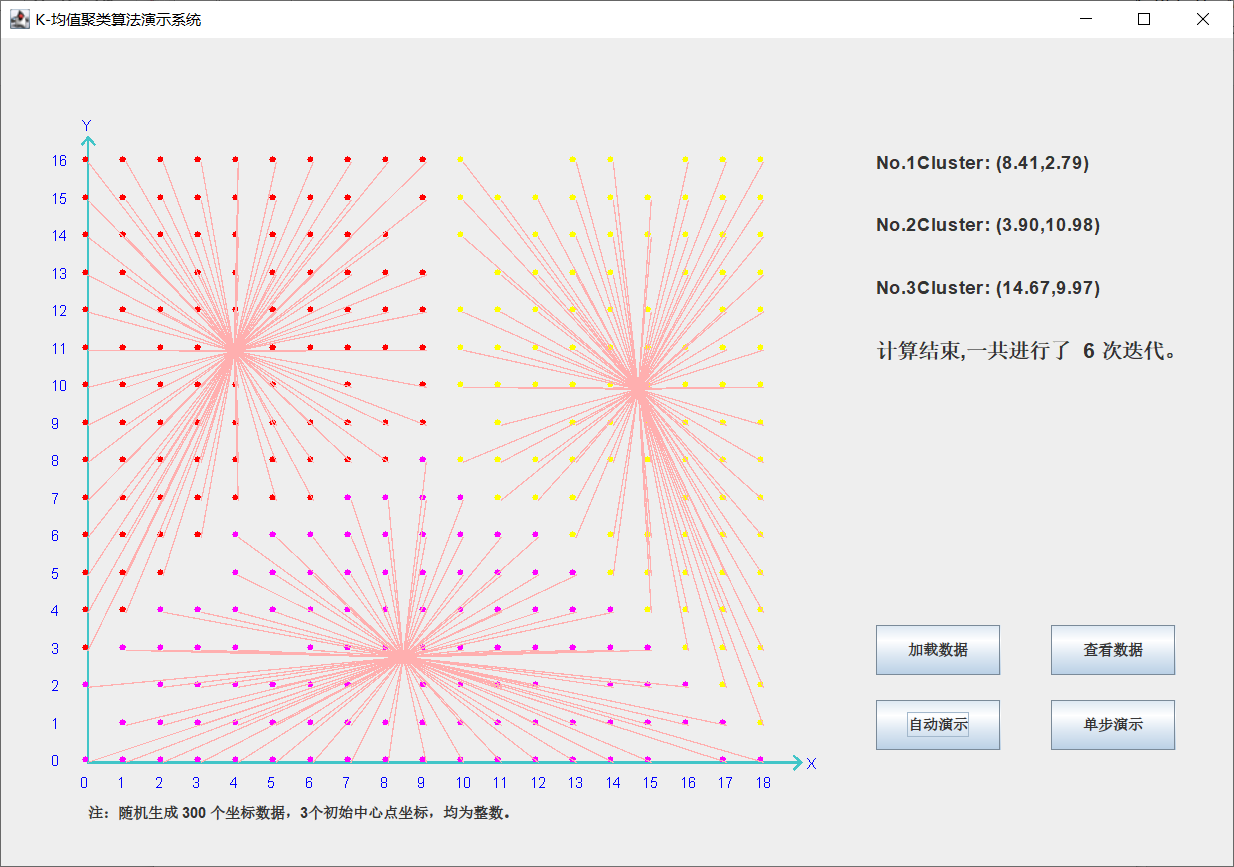
4.4.2分形算法

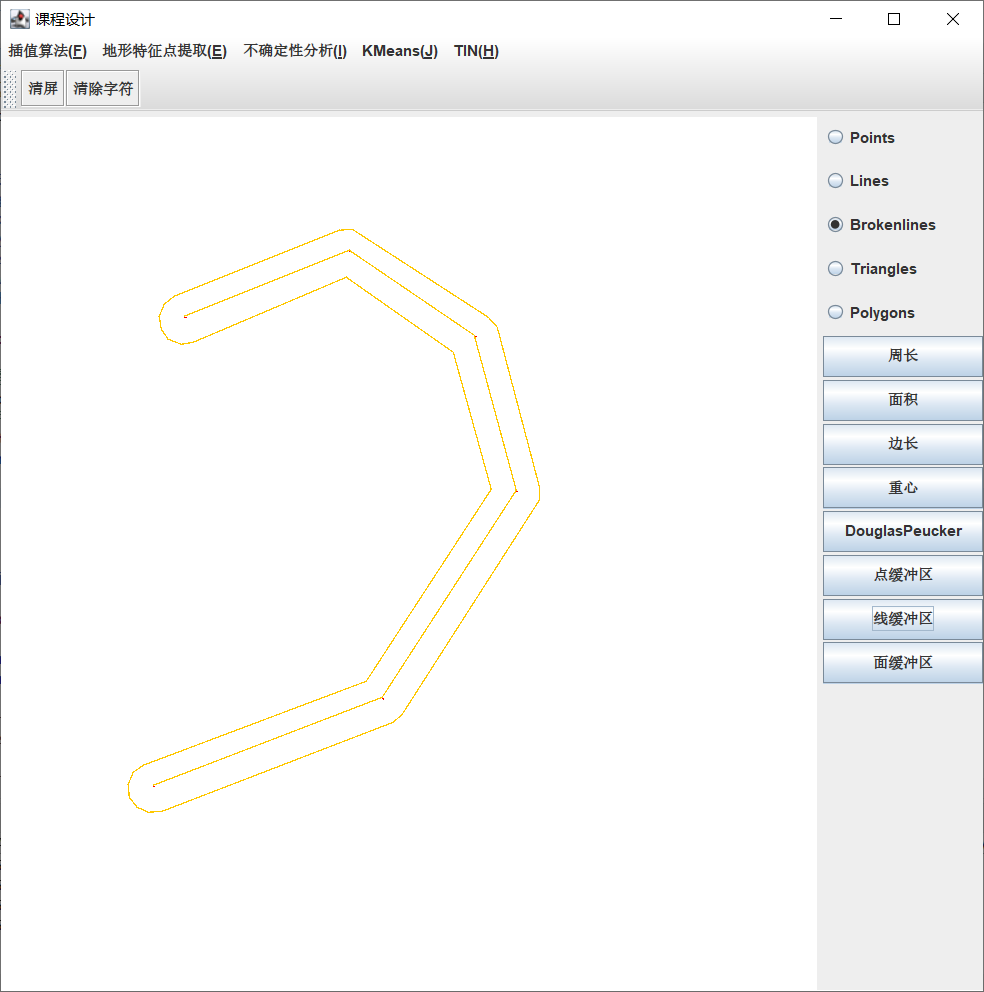


## K-Means聚类分析算法





## 矢量数据的缓冲区分析计算

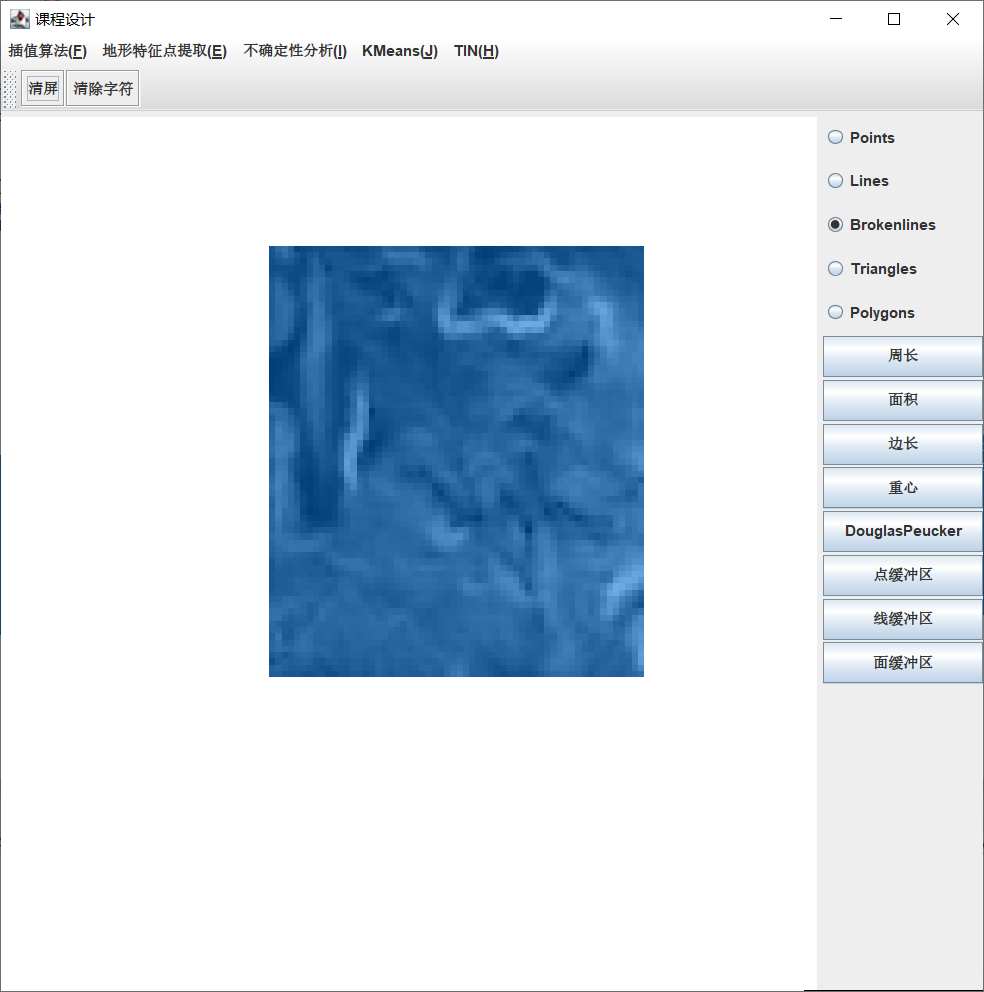


## 基于DEM插值算法

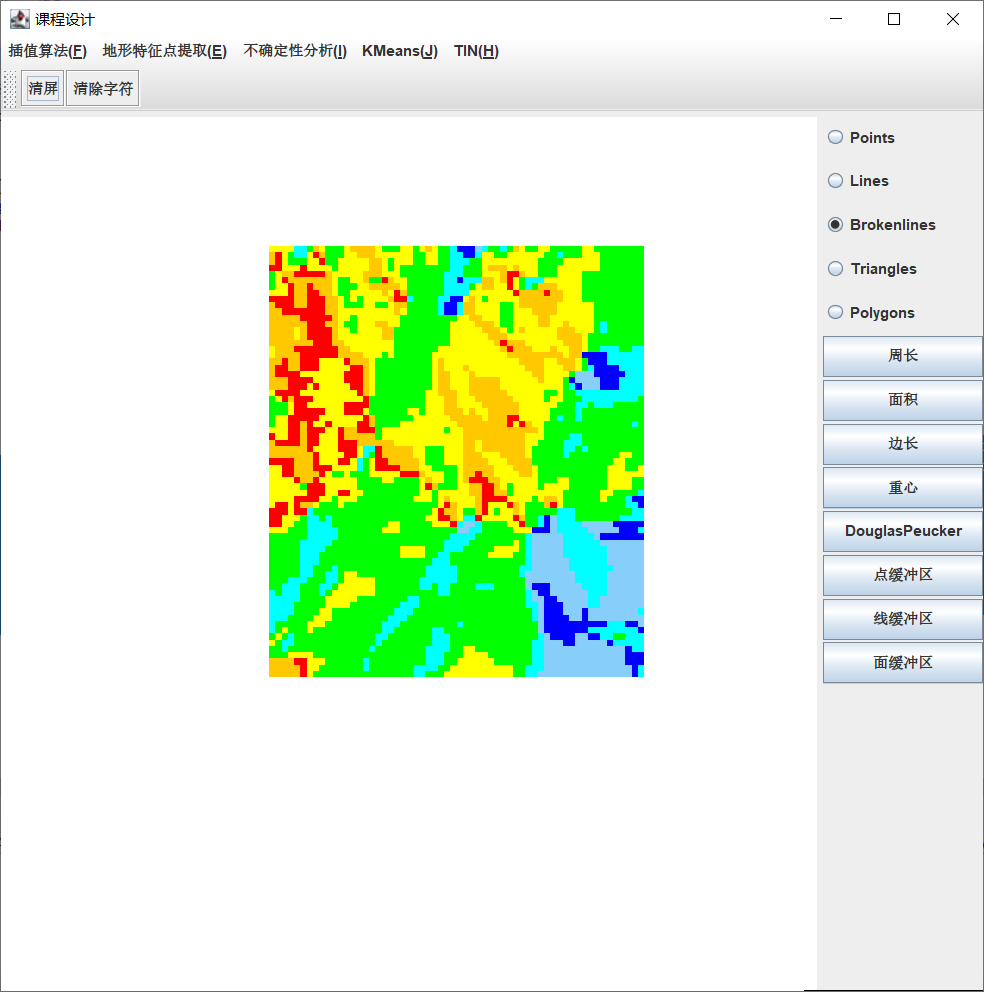


## 基于DEM地形特征点提取算法

4.8.1坡度分析



4.8.2坡向分析



# 实习总结

通过这次实习，我收获颇多。一方面是对空间分析的理论知识有了更深刻的理解，对算法的原理也有更充分的认识；另一方面，因为我是用JAVA实现的算法设计与可视化，因此我JAVA的编程能力有了很大的提高，同时也了解到JAVA作为一门编程语言的强大之处。

不仅如此，我解决问题的能力也有了很大的提升，遇到解决不了问题我会从网上查阅资料，与使用相同语言编程的同学讨论，或者询问老师解决方案。遇到问题不慌不乱，细心解决，平时躺在床上也可以想这个问题，有时思路就会灵光一现，马上打开电脑尝试，通过不断的试错来解决问题。

最后，我与一个使用相同语言的同学共享了一部分成果，相互讨论，一人提出一部分内容，将他们结合起来，并加以改进，使之变得更加完美，这对我合作能力的提升有很大的帮助。

当然，这次实习也有很多不足，因为各种原因，一部分内容没能实现，并且系统的用户体验也有一定的弊端，比如折线绘制完成清屏后，再次绘制连接的仍然是之前的折线点，这需要我对于JAVA线程的操作有更深刻的理解，需要日后对系统进行进一步的改进。