全国大学测绘学科创新创业智能大赛

测绘程序设计比赛

基于统计滤波的点云去噪

基于统计滤波的点云去噪

基于统计滤波的点云去噪

一、 程序优化性说明 1

1. 用户交互界面说明 1

2. 程序运行过程说明 2

3. 程序运行结果 5

二、 程序规范性说明 5

1. 程序功能与结构设计说明 5

2. 核心算法源码（给出主要算法的源码） 5

（1）关键函数汇总. 5

（2）关键源码展示. 6

三、 程序完整源代码 7

一、 程序优化性说明

1. 用户交互界面说明

在设计用户交互性页面的时候，本页面运用了菜单栏，按钮控件，Data 表格控件和状态栏等控件，一共有打开文件，计算，文件导出和帮助四个功能，点击不同的按钮控件即可跳转到不同的功能，程序的执行状态会在状态栏中进行展示，呈现的数据则通过 data 控件和 richttextbox 控件展示，具体如图 1 所示。

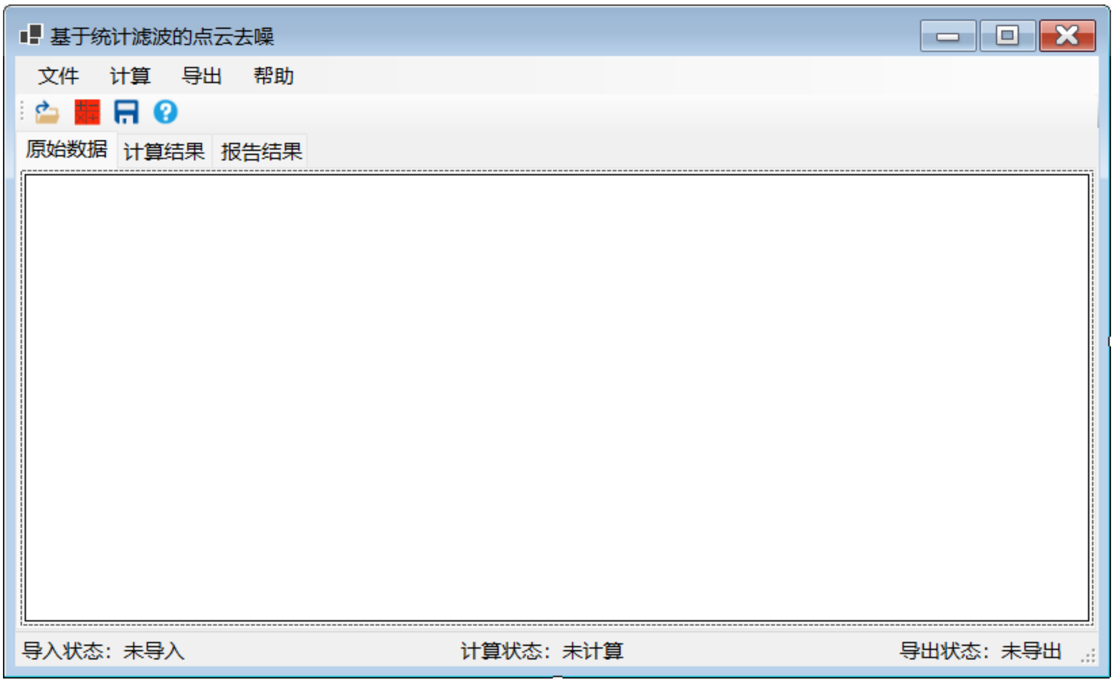


图 1 程序交互式界面说明

2. 程序运行过程说明

点击打开文件，即可打开文件对话框，选择输入的文件

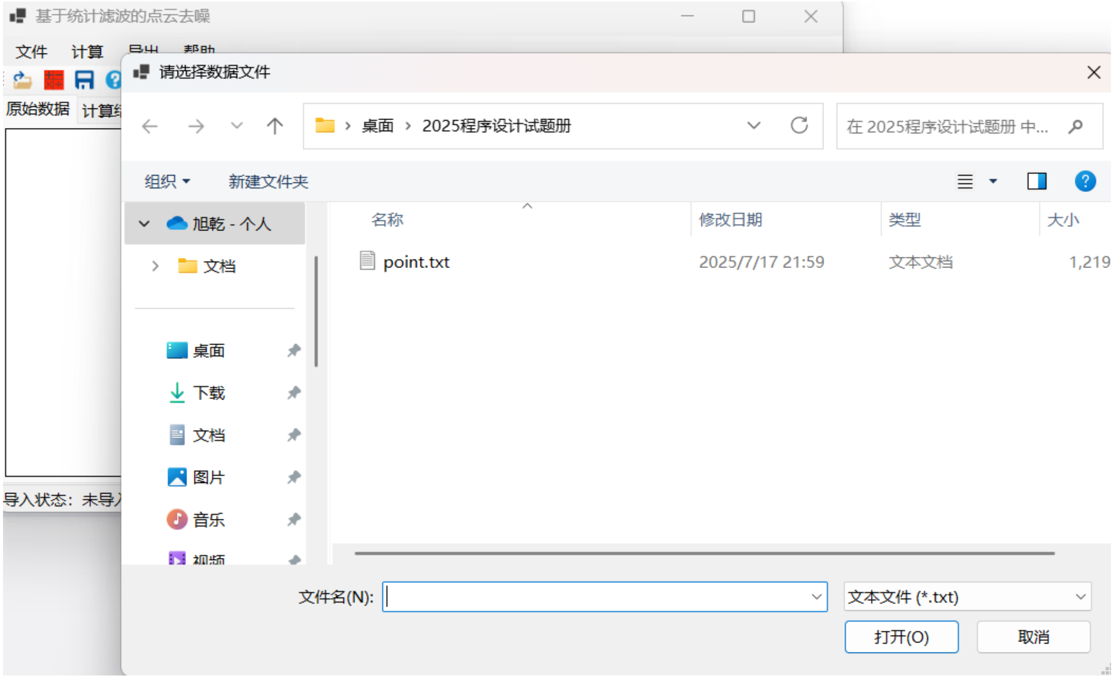
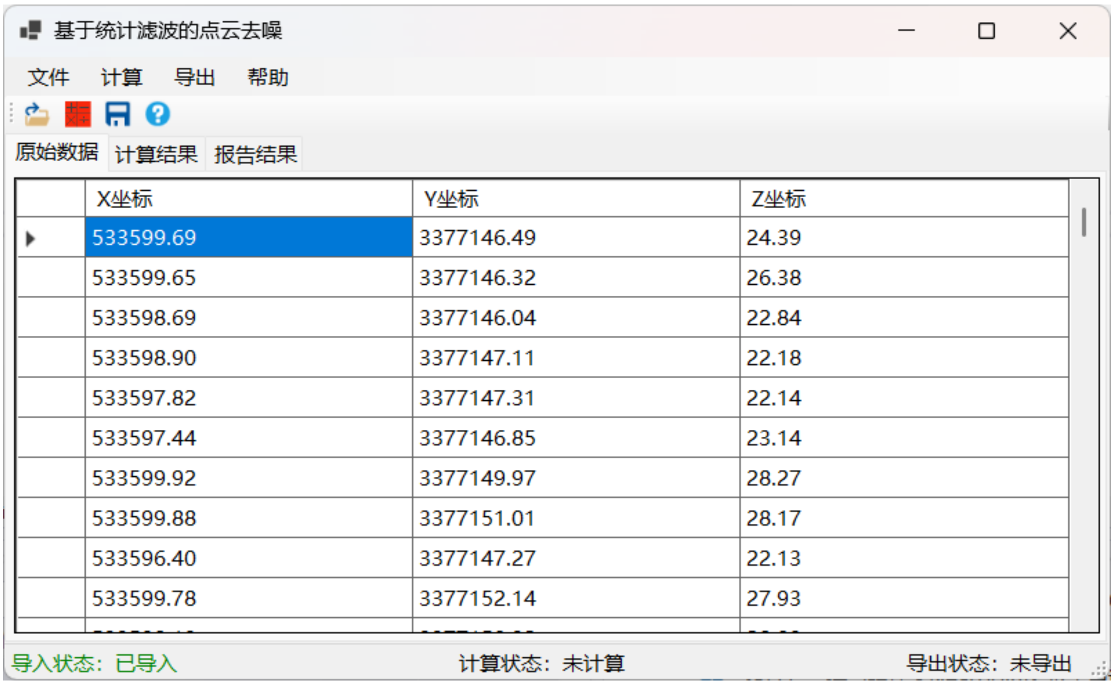


图 2 打开文件

待数据加载进来后，数据在主页面中显示，并更新状态栏。

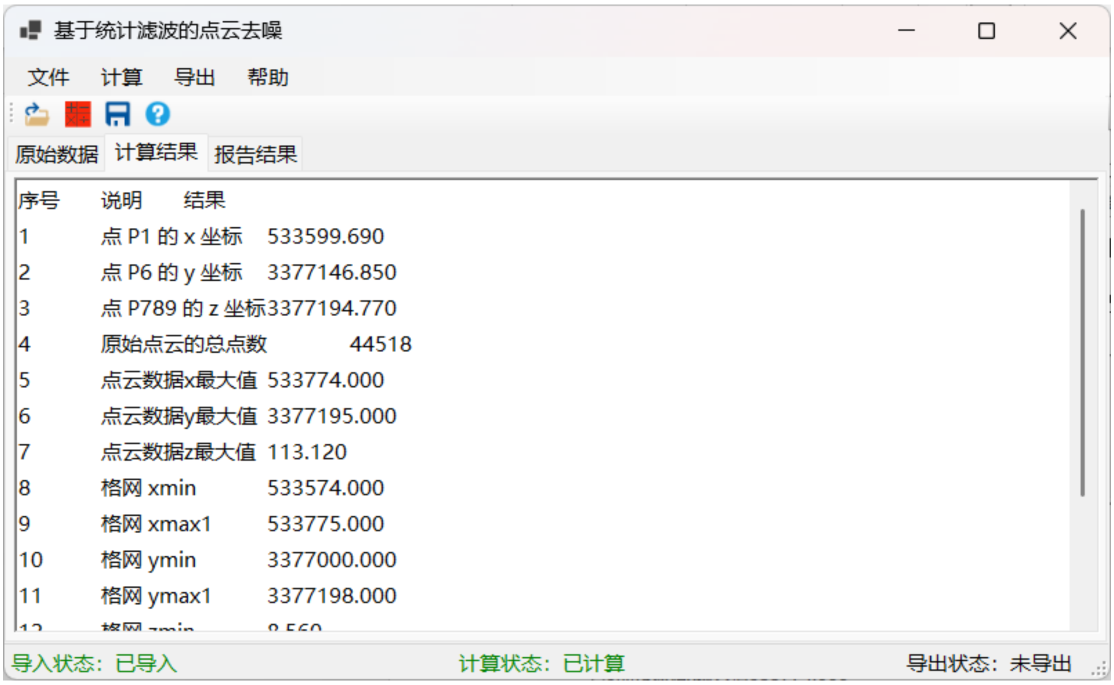


	X坐标	Y坐标	Z坐标
▶	533599.69	3377146.49	24.39
	533599.65	3377146.32	26.38
	533598.69	3377146.04	22.84
	533598.90	3377147.11	22.18
	533597.82	3377147.31	22.14
	533597.44	3377146.85	23.14
	533599.92	3377149.97	28.27
	533599.88	3377151.01	28.17
	533596.40	3377147.27	22.13
	533599.78	3377152.14	27.93

导入状态：已导入　　计算状态：未计算　　导出状态：未导出

图 3 文件数据展示

点击计算，即可调用具体的算法函数，计算出数据结果。

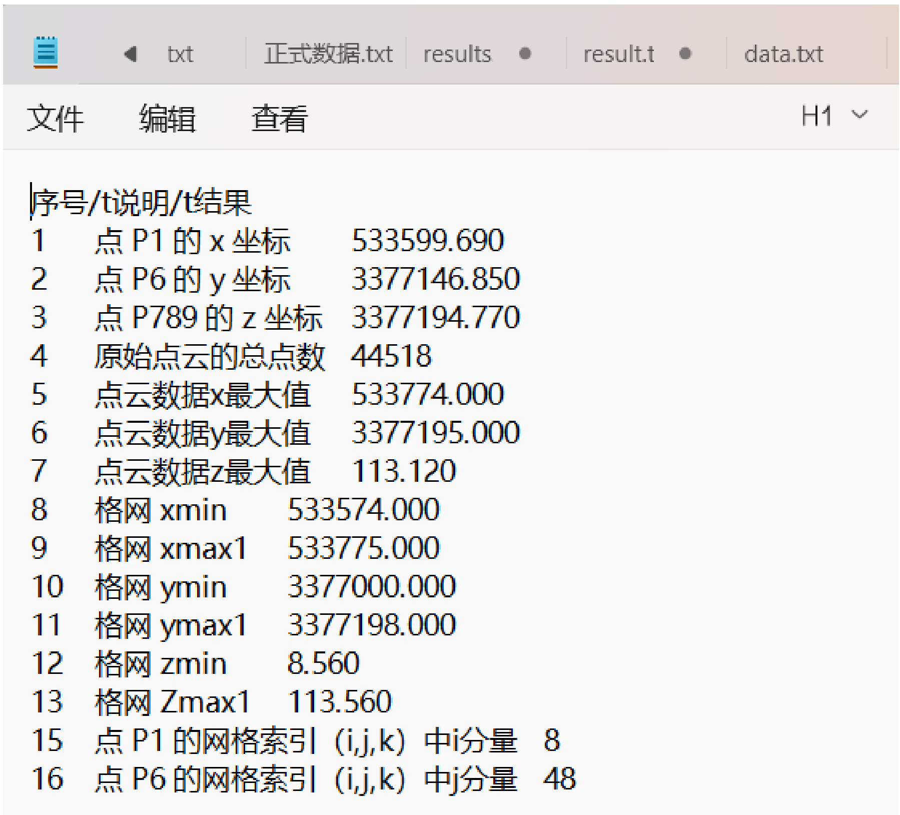


序号	说明	结果
1	点 P1 的 x 坐标	533599.690
2	点 P6 的 y 坐标	3377146.850
3	点 P789 的 z 坐标	3377194.770
4	原始点云的总点数	44518
5	点云数据x最大值	533774.000
6	点云数据y最大值	3377195.000
7	点云数据z最大值	113.120
8	格网 xmin	533574.000
9	格网 xmax1	533775.000
10	格网 ymin	3377000.000
11	格网 ymax1	3377198.000
12	格网 zmin	8.560

导入状态：已导入　　计算状态：已计算　　导出状态：未导出

图 4 文件计算结果

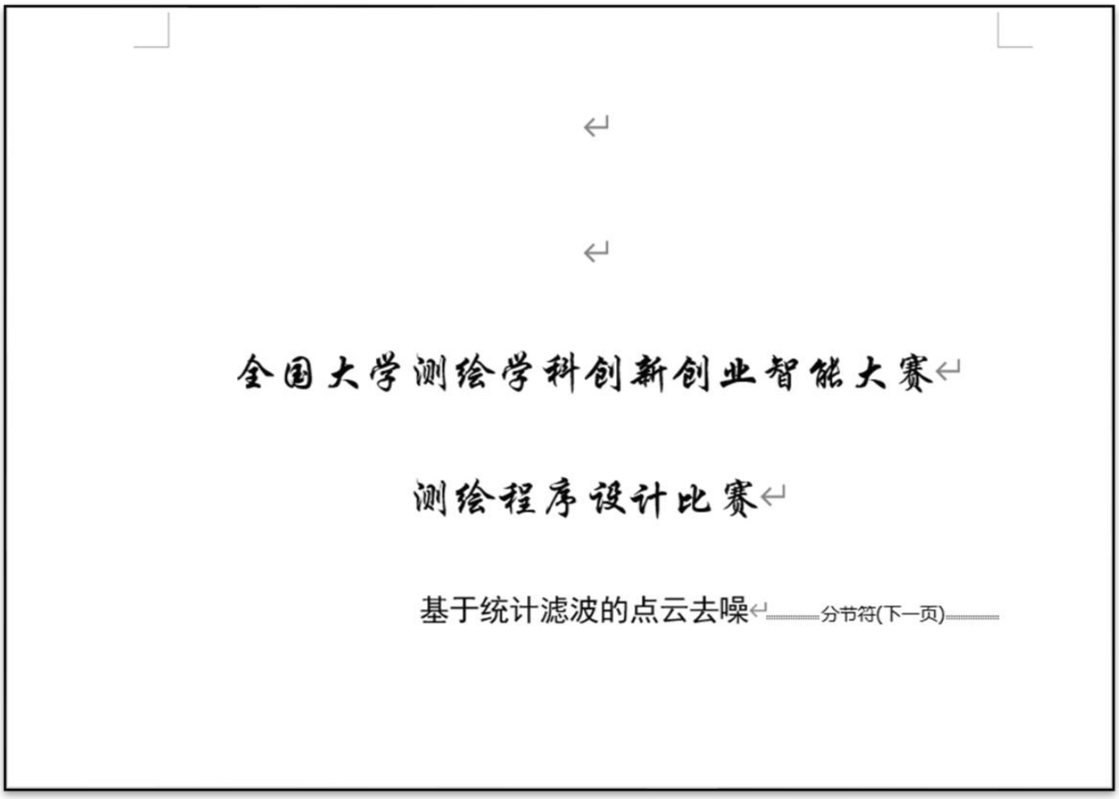
点击导出，即可将结果文件导出到程序的运行目录下。



```
序号/t说明/t结果
1    点 P1 的 x 坐标      533599.690
2    点 P6 的 y 坐标      3377146.850
3    点 P789 的 z 坐标    3377194.770
4    原始点云的总点数    44518
5    点云数据x最大值     533774.000
6    点云数据y最大值     3377195.000
7    点云数据z最大值     113.120
8    格网 xmin      533574.000
9    格网 xmax1     533775.000
10   格网 ymin      3377000.000
11   格网 ymax1     3377198.000
12   格网 zmin      8.560
13   格网 Zmax1     113.560
15   点 P1 的网格索引（i,j,k）中i分量   8
16   点 P6 的网格索引（i,j,k）中j分量   48
```

图 5 导出计算结果

点击帮助即可跳转到本报告文档。



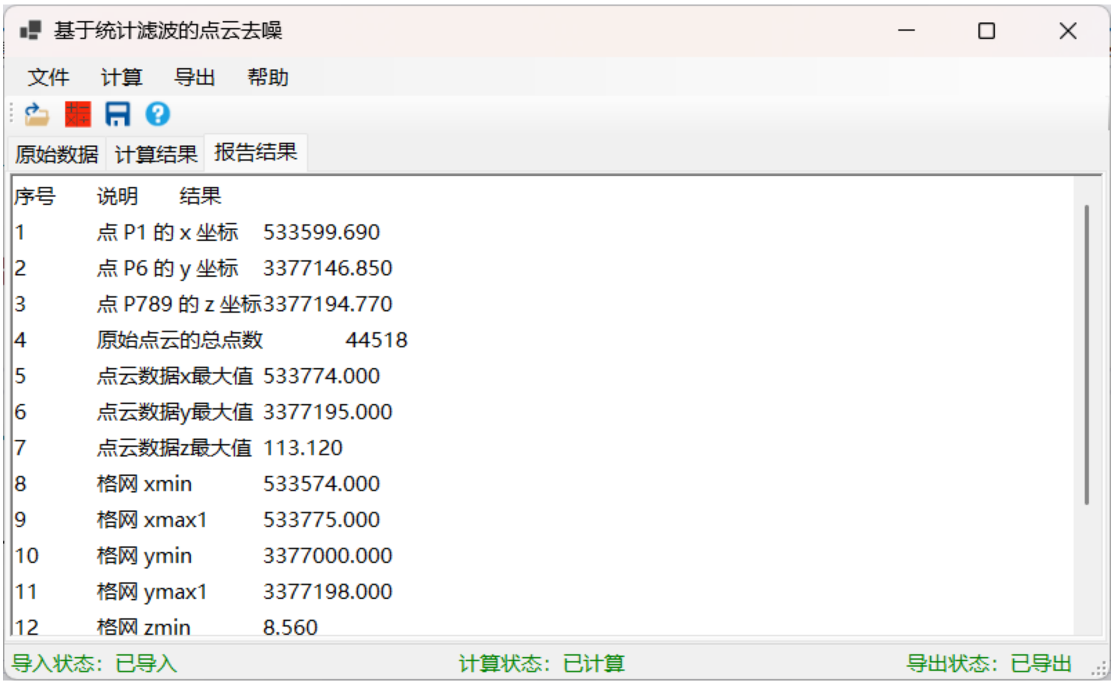
全国大学测绘学科创新创业智能大赛

测绘程序设计比赛

基于统计滤波的点云去噪

图 6 打开报告文档

3. 程序运行结果



```
序号    说明          结果
1       点 P1 的 x 坐标    533599.690
2       点 P6 的 y 坐标    3377146.850
3       点 P789 的 z 坐标3377194.770
4       原始点云的总点数          44518
5       点云数据x最大值 533774.000
6       点云数据y最大值 3377195.000
7       点云数据z最大值 113.120
8       格网 xmin          533574.000
9       格网 xmax1         533775.000
10      格网 ymin          3377000.000
11      格网 ymax1         3377198.000
12      格网 zmin          8.560
```

导入状态：已导入　　计算状态：已计算　　导出状态：已导出

图 6 程序运行结果

二、 程序规范性说明

1. 程序功能与结构设计说明

程序主要有一个算法类 Calculate 和三个实体类 pointCloud、Info 和 Grid，其中，Calculate 类用于数据计算，pointCloud 用于存储点云数据和邻域的点 List，Info 通过构建 results 类来存储算法执行结果，Grid 主要用于存储格网的索引与格网类的点。

在程序中主要有三个算法函数，分别进行格网的划分、K 最近的搜索于与分组和点云去噪，还有读文件与写文件函数用于读取文件。

2. 核心算法源码（给出主要算法的源码）

（1）关键函数汇总

序号	类名(若有)	函数名	输入参数	输出参数	主要功能描述
1	Calculate	step22CreateGrid	PointList	GridList	构建索引格网并将点加入格网中
2	Calculate	step23findnearpoint	无	无	k 邻近点搜索符合要求的点

3	Calculate	Step24	无	无	点云去噪
4	Calculate	CalculateDistance	Point1，point2	Distance	计算欧氏距离

（2）关键源码展示

```
public void step23findnearpoint() {
   for (int i = 0; i < pointCloulds.Count; i++)
   {
      List<grid> NearGrids = new List<grid>();//用于记录临近网格
      int Xindex = pointCloulds[i].gridIndex[0]; int Yindex = pointCloulds[i].gridIndex[1]; int Zindex =
pointCloulds[i].gridIndex[2];

      //搜索周围 3*3*3 的格网，并加入到临近网格的 List 中
      for (int x = Xindex - 1; x <= Xindex + 1; x++) {
         for (int y = Yindex - 1; y <= Yindex + 1; y++) {
            for (int z = Zindex - 1; z <= Zindex + 1; z++) {
               grid NearGrid = grids.Where(temp2 => (temp2.order[0] == x) && (temp2.order[1]) == y &&
(temp2.order[2]) == z).ToList()[0];
               if (NearGrid != null)
               {
                  NearGrids.Add(NearGrid);
               }
               else { continue; }

            }
         }
      }
      //遍历临近格网，计算 pi 和格网点之间的距离
      List<(point,double)> temp3 = new List<(point,double)> ();//记录点和 pi 之间距离的字典
      for (int j = 0; j < NearGrids.Count; j++) {
         for (int k = 0; k < NearGrids[j].innerpoints.Count; k++) {
            (point, double) temp4;
            temp4.Item1 = NearGrids[j].innerpoints[j];
            temp4.Item2 = CalculateDistance(NearGrids[j].innerpoints[j], pointCloulds[i]);//计算 pi 和 pk 之间的
欧氏距离
            temp3.Add(temp4);
         }
      }
      temp3.Sort((x, y) => x.Item2.CompareTo(y.Item2));//按照升序排列
      //从 1 开始赋值，跳过距离为 0 的点
      for (int j = 1; j < 7; j++) {
         (point, double) temp5 = temp3[j];
         pointCloulds[i].Nearpoints.Add(temp5);
      }
      //计算领域内点的平均值和标准差

      for (int j = 0; j < pointCloulds[i].Nearpoints.Count; j++) {
         pointCloulds[i].AverageDistance += pointCloulds[i].Nearpoints[j].Item2;
      }
      pointCloulds[i].AverageDistance /= pointCloulds[i].Nearpoints.Count;
      for (int j = 0; j < pointCloulds[i].Nearpoints.Count; j++)
      {
         pointCloulds[i].stardantdiff += Math.Pow((pointCloulds[i].Nearpoints[j].Item2 -
pointCloulds[i].AverageDistance), 2);
      }
      pointCloulds[i].stardantdiff = Math.Sqrt(pointCloulds[i].stardantdiff / pointCloulds[i].Nearpoints.Count);
   }

   //计算所有点的平均距离和标准差
```

```
    for (int i = 0; i < pointClouIds.Count; i++) {
        AverageDis += pointClouIds[i].AverageDistance;
        StardantDiff += pointClouIds[i].stardantdiff;
    }
    AverageDis /= pointClouIds.Count;
   StardantDiff /= pointClouIds.Count;
}
```

三、 程序完整源代码

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

namespace 基于统计滤波的点云去噪
{
    public class info{//每一行答案的类
        public int id;
        public string description;
        public string value;

        public info(int id, string description, string value)
        {//构造函数
            this.id = id;
            this.description = description;
            this.value = value;
        }
        public info()
        {

        }
    }
    public class point {//用于存储点的做坐标
        public int id;
        public double x;
        public double y;
        public double z;
        public int[] gridIndex = new int[3];//记录点云属于哪个格网
        public List<(point,double)> Nearpoints = new List<(point, double)>();//领域内的 6 个点
        public double AverageDistance = 0;//领域内的平均距离
        public double stardantdiff = 0;//领域内的标准差

        public point(int id, double x,double y,double z) {
            this.id = id;
            this.x = x;
            this.y = y;
            this.z = z;
        }
        public point(){}
    }

    public class grid() {
        public int[] order = new int[3];//分别记录格网的 x，y，z 索引
        public List<point> innerpoints = new List<point>();//记录格网内的点

    }
    public class Calculate
```

```
  {
    public List<point> pointCloulds = new List<point>();
    public List<info> results = new List<info>();
    public double edge_length = 3.0;//记录格网的边长
    public double xmin = 0; public double xmax = 0;
    public double ymin = 0; public double ymax = 0;
    public double zmin = 0; public double zmax = 0;//记录点范围的最小值
    public double Xmax1 = 0; public double Ymax1 = 0; public double Zmax1 = 0;//记录格网范围的最大值
和最小值
    public List<grid> grids = new List<grid>();
    public double AverageDis = 0; public double StardantDiff = 0;//所有点的平均距离和标准差

    //数据读取函数
    public void step21FileRead() {
      OpenFileDialog openFileDialog = new OpenFileDialog();
      openFileDialog.Title = "请选择数据文件";
      openFileDialog.Filter = "文本文件|*.txt";
      if (openFileDialog.ShowDialog() == DialogResult.OK) {
        StreamReader sr = new StreamReader(openFileDialog.FileName);
        int i = 0;
        while (!sr.EndOfStream)
        {
          string[] parts = sr.ReadLine().Split(' ');
          double x = double.Parse(parts[0]);
          double y = double.Parse(parts[1]);
          double z = double.Parse(parts[2]);
          pointCloulds.Add(new point(i++, x, y, z));
        }
        sr.Close();
        results.Add(new info(1, "点 P1 的 x 坐标", pointCloulds[0].x.ToString("F3")));
        results.Add(new info(2, "点 P6 的 y 坐标", pointCloulds[5].y.ToString("F3")));
        results.Add(new info(3, "点 P789 的 z 坐标", pointCloulds[788].y.ToString("F3")));
        results.Add(new info(4, "原始点云的总点数", pointCloulds.Count.ToString()));
      }
    }
    //构建索引格网并将点加入格网中
    public void step22CreateGrid() {
      //计算点云的范围
      xmin = pointCloulds.Min(x => x.x);
      xmax = pointCloulds.Max(x => x.x);
      ymin = pointCloulds.Min(y => y.y);
      ymax = pointCloulds.Max(y => y.y);
      zmin = pointCloulds.Min(z => z.z);
      zmax = pointCloulds.Max(z => z.z);
      //计算格网范围的最大值
      Xmax1 = (int)(((xmax - xmin) / edge_length) + 1) * edge_length + xmin;
      Ymax1 = (int)(((ymax - ymin) / edge_length) + 1) * edge_length + ymin;
      Zmax1 = (int)(((zmax - zmin) / edge_length) + 1) * edge_length + zmin;

      //计算点云所属的格网
      for (int i = 0; i < pointCloulds.Count; i++) {
        //计算每个点所在格网的索引
        int Xindex = (int)((pointCloulds[i].x - xmin) / edge_length);
        int Yindex = (int)((pointCloulds[i].y - ymin) / edge_length);
        int Zindex = (int)((pointCloulds[i].z - zmin) / edge_length);
        pointCloulds[i].gridIndex[0] = Xindex;
        pointCloulds[i].gridIndex[1] = Yindex;
        pointCloulds[i].gridIndex[2] = Zindex;

        //当格网数量为 0 时，直接将第一个点加入到格网中
        grid first = new grid();
        first.order[0] = pointCloulds[i].gridIndex[0]; first.order[1] = pointCloulds[i].gridIndex[1];
first.order[2] = pointCloulds[i].gridIndex[2];
        first.innerpoints.Add(pointCloulds[i]);
        grids.Add(first);
```

```
            //遍历格网列表，如果格网存在则将点加入，不存在则创建新格网
            bool exist = false;
            for (int j = 0; j < grids.Count; j++) {
                if ((pointCloulds[i].gridIndex[0] == grids[j].order[0]) && (pointCloulds[i].gridIndex[1] ==
grids[j].order[1]) && (pointCloulds[i].gridIndex[2] == grids[j].order[2])) {
                    exist = true; grids[j].innerpoints.Add(pointCloulds[i]);break;
                }
            }
            //如果找不到匹配的格网索引则创建格网
            if (!exist) {
                grid temp1 = new grid();
                temp1.order[0] = pointCloulds[i].gridIndex[0]; temp1.order[1] = pointCloulds[i].gridIndex[1];
temp1.order[2] = pointCloulds[i].gridIndex[2];
                temp1.innerpoints.Add(pointCloulds[i]);
                grids.Add(temp1);
            }
        }
        results.Add(new info(5, "点云数据 x 最大值", xmax.ToString("F3")));
        results.Add(new info(6, "点云数据 y 最大值", ymax.ToString("F3")));
        results.Add(new info(7, "点云数据 z 最大值", zmax.ToString("F3")));
        results.Add(new info(8, "格网 xmin", xmin.ToString("F3")));
        results.Add(new info(9, "格网 xmax1", Xmax1.ToString("F3")));
        results.Add(new info(10, "格网 ymin", ymin.ToString("F3")));
        results.Add(new info(11, "格网 ymax1", Ymax1.ToString("F3")));
        results.Add(new info(12, "格网 zmin", zmin.ToString("F3")));
        results.Add(new info(13, "格网 Zmax1", Zmax1.ToString("F3")));
        //grid temp = grids.Where(x => (x.order[0] == 0) && (x.order[1]) == 0 && (x.order[2]) ==
0).ToList()[0];//查询网格索引为 000 的网格
        //results.Add(new info(14, "网格 (0,0,0) 内的点个数", temp.innerpoints.Count.ToString()));
        results.Add(new info(15, "点 P1 的网格索引（i,j,k）中 i 分量",
pointCloulds[0].gridIndex[0].ToString()));
        results.Add(new info(16, "点 P6 的网格索引（i,j,k）中 j 分量",
pointCloulds[5].gridIndex[1].ToString()));

    }
    //k 邻近点搜索
    public void step23findnearpoint() {
        for (int i = 0; i < pointCloulds.Count; i++)
        {
            List<grid> NearGrids = new List<grid>();//用于记录临近网格
            int Xindex = pointCloulds[i].gridIndex[0]; int Yindex = pointCloulds[i].gridIndex[1]; int Zindex =
pointCloulds[i].gridIndex[2];

            //搜索周围 3*3*3 的格网，并加入到临近网格的 List 中
            for (int x = Xindex - 1; x <= Xindex + 1; x++) {
                for (int y = Yindex - 1; y <= Yindex + 1; y++) {
                    for (int z = Zindex - 1; z <= Zindex + 1; z++) {
                        grid NearGrid = grids.Where(temp2 => (temp2.order[0] == x) && (temp2.order[1]) == y &&
(temp2.order[2]) == z).ToList()[0];
                        if (NearGrid != null)
                        {
                            NearGrids.Add(NearGrid);
                        }
                        else { continue; }

                    }
                }
            }
            //遍历临近格网，计算 pi 和格网点之间的距离
            List<(point,double)> temp3 = new List<(point,double)> ();//记录点和 pi 之间距离的字典
            for (int j = 0; j < NearGrids.Count; j++) {
                for (int k = 0; k < NearGrids[j].innerpoints.Count; k++) {
                    (point, double) temp4;
                    temp4.Item1 = NearGrids[j].innerpoints[j];
                    temp4.Item2 = CalculateDistance(NearGrids[j].innerpoints[j], pointCloulds[i]);//计算 pi 和 pk 之间的欧氏距离
                    temp3.Add(temp4);
                }
            }
            temp3.Sort((x, y) => x.Item2.CompareTo(y.Item2));//按照升序排列
            //从 1 开始赋值，跳过距离为 0 的点
            for (int j = 1; j < 7; j++) {
                (point, double) temp5 = temp3[j];
                pointCloulds[i].Nearpoints.Add(temp5);
            }
            //计算领域内点的平均值和标准差

            for (int j = 0; j < pointCloulds[i].Nearpoints.Count; j++) {
                pointCloulds[i].AverageDistance += pointCloulds[i].Nearpoints[j].Item2;
            }
            pointCloulds[i].AverageDistance /= pointCloulds[i].Nearpoints.Count;
            for (int j = 0; j < pointCloulds[i].Nearpoints.Count; j++)
            {
                pointCloulds[i].stardantdiff += Math.Pow((pointCloulds[i].Nearpoints[j].Item2 - pointCloulds[i].AverageDistance), 2);
            }
            pointCloulds[i].stardantdiff = Math.Sqrt(pointCloulds[i].stardantdiff / pointCloulds[i].Nearpoints.Count);
        }

        //计算所有点的平均距离和标准差
        for (int i = 0; i < pointCloulds.Count; i++) {
            AverageDis += pointCloulds[i].AverageDistance;
            StardantDiff += pointCloulds[i].stardantdiff;
        }
        AverageDis /= pointCloulds.Count;
        StardantDiff /= pointCloulds.Count;

        results.Add(new info(17, "点 P1 的候选点总数", pointCloulds[0].Nearpoints.Count.ToString()));
        results.Add(new info(18, "点 P6 的候选点总数", pointCloulds[5].Nearpoints.Count.ToString()));
        int MAX1 = pointCloulds[0].Nearpoints.Max(x => x.Item1.id);
        results.Add(new info(19, "点 P1 的 6 个邻近点序号中最大值", MAX1.ToString()));
        int MAX6 = pointCloulds[5].Nearpoints.Max(x => x.Item1.id);
        results.Add(new info(20, "点 P6 的 6 个邻近点序号中最大值", MAX6.ToString()));
        results.Add(new info(21, "点 P1 的邻域平均距离 u1", pointCloulds[0].AverageDistance.ToString("F3")));
        results.Add(new info(22, "点 P1 的邻域距离标准差 σ1", pointCloulds[0].stardantdiff.ToString("F3")));
        results.Add(new info(23, "点 P6 的邻域平均距离 u1", pointCloulds[5].AverageDistance.ToString("F3")));
        results.Add(new info(24, "点 P6 的邻域距离标准差 σ1", pointCloulds[5].stardantdiff.ToString("F3")));
        results.Add(new info(25, "全局平均距离均值 μ1", AverageDis.ToString("F3")));
        results.Add(new info(26, "全局距离标准差 σ ", StardantDiff.ToString("F3")));
    }

    //噪声判断
    public void Step24() {
        List<point> pointscopy = new List<point>();
        for (int i = 0; i < pointscopy.Count; i++) {
            pointscopy.Add(pointscopy[i]);
        }
        for (int i = pointscopy.Count -1; i >= 0; i--) {
            double temp = AverageDis + 2 * StardantDiff;
            if (pointscopy[i].stardantdiff > temp) {
                pointscopy.RemoveAt(i);

            }
        }
        bool p1is = false; bool p6is = false;//记录是否是噪声 p1 和 p6
```

```
      point temp6 = pointscopy.Where(x => x.id == 0).ToList()[0];
      if (temp6 == null) {
        p1is = true;
      }
      point temp7 = pointscopy.Where(x => x.id == 5).ToList()[0];
      if (temp7 == null)
      {
        p6is = true;
      }
      if (p1is) {
        results.Add(new info(27, "点 P1 是否为噪声点（0、1 分别表示否、是）", "0"));
      }
      else { results.Add(new info(27, "点 P1 是否为噪声点（0、1 分别表示否、是）", "1")); }
      if (p6is)
      {
        results.Add(new info(28, "点 P6 是否为噪声点（0、1 分别表示否、是）", "0"));
      }
      else { results.Add(new info(28, "点 P6 是否为噪声点（0、1 分别表示否、是）", "1")); }
      results.Add(new info(29, "去噪后保留的点云总数", pointscopy.Count.ToString()));
    }
    public void FileWrite() {
      string filename = AppContext.BaseDirectory + "results.txt";
      StreamWriter sw = new StreamWriter(filename);
      sw.WriteLine("序号/t 说明/t 结果");
      foreach (var item in results)
      {
        sw.WriteLine(item.id.ToString() + "\t" + item.description + "\t" + item.value);
      }
      sw.Close();
    }

    //辅助函数，1.计算欧氏距离
    public double CalculateDistance(point p1,point p2) {
      double distance = 0;
      distance = Math.Sqrt(Math.Pow((p1.x - p2.x), 2) + Math.Pow((p1.y - p2.y), 2) + Math.Pow((p1.z - p2.z),
2));
      return distance;
    }

  }
}
一、
```