地矿测绘
Geological and Mineral Surveying and Mapping
第 3 卷◆第 1 期◆版本 1.0◆2020 年
文章类型：论文|刊号（ISSN）：2630-4732

基于 python 语言点云抽稀算法的研究及实现

李小勇　李召　张鹏
郑州市郑东新区土地规划勘测中心
DOI:10.32629/gmsm.v3i1.515

[摘　要] 随着测绘技术的迅猛发展,无人机和激光雷达广泛应用于测量生产中,可以获取到高精度和高密度的三维点云地形数据,而这些高密度的点云数据存在巨大的冗余,造成数据量的庞大,对后期的处理和成图、存储、显示带来困难,同时也造成计算资源和人力资源的浪费,因此对点云数据的抽稀处理尤为必要。本文主要探讨基于 python 语言点云抽希算法的优化和实现。
[关键词] 点云数据；点云抽稀；无人机；激光雷达；python；算法；阈值

引言

随着测量技术日新月异的发展,获取三维空间数据的手段发生了革命性的改变,特别是无人机航空摄影和激光雷达技术的应用,这种非接触式数据采集方式不但减轻了测绘工作的外业劳动强度,更重要的是大大提高了外业数据采集的效率。目前已广泛应用于众多测绘企业的外业数据采集中。然而,这种方式获取的点云数据的数据量十分庞大,冗余数据多,直接影响的数据存储速度、数据显示速度,在数据生产和数据交互的过程中也造成了很大的难度,同时也造成计算资源和人力资源浪费。因此对原始点云首先进行抽稀简化工作是十分必要的。

对于点云的抽稀从理论上讲,目的是希望既不损失原有数据的精度又能尽可能多的减少数据量,为数据的后处理节省更多的时间,其目的主要是用更少的点精确表述地面、地物的特征,在点云密度和数据精度之间达到平衡。本文着重从距离和高程两个因素考虑,运用python编程语言来实现点云抽希。

1 算法思路及实现

首先读取原始文件,设定一个距离阈值,从第一个点开始作为基点,计算其他点到基点的距离,遍历所有点,如果距离小于阈值就弃掉该点,如果距离大于阈值就保留,再在保留下来的点中选取第二个点作为基点,计算其后所有点到基点距离,重复上面的条件判断是否保留,直到循环处理所有的点,这样计算越到后面的点数据越少,计算速度就越快。

1.1程序代码及说明

先导入两个需要的模块：

```
import math
import time
```

为了测试程序的效率,在程序运行开始计时,并输入要处理的原始文件和成果输出文件路径：

```
stime=time.time()
a=r'E:\dsmdata\30.4.xyz'
b=r'E:\dsmdata\30.4-1.dat'
```

读出文件到一个空的列表中：

```
yxz=[]
with open(a,'r') as f1:
    for line in f1:
        lx = line.split()
        yxz.append([float(lx[0]),float(lx[1]),float(lx[2])])
print('原始数据点数：',len(yxz))
```

循环依次选取基点：

```
i=0
while i <len(yxz):
    y = yxz[i][0]
    x = yxz[i][1]
    j=i+1
```

循环计算其他点到基点的距离,并依据给定的阈值判断是否舍弃：

```
while j<len(yxz):
    y1 = yxz[j][0]
    x1 =yxz[j][1]
    if math.sqrt((x1 - x) ** 2 + (y1 - y) ** 2)<10:
        yxz.pop(j)
    else:
        j=+1
i+=1
```

把数据处理结果写入文件,并计算整个程序运行的时间：

```
print('处理后的点数：',len(yxz))
with open(b,'w') as f2:
    for j in range(0,len(yxz)):
f2.write(str(j)+',,'+str(yxz[j][0])+','+str(yxz[j][1])+','+str(yxz[j][2])+'\n')
end_time=time.time()
tim=end_time-start_time
print('用时:',tim)
```

1.2程序运行结果

程序运行结果如下：

```
F:\python\program\venv\Scripts\python.exe F:
原始数据点数：  843647
处理后点数：  1699
用时：  1170.7069606781006

Process finished with exit code 0
```

实例中文件大小为30.4M,共有843647个点,按10米的点间距抽稀最后结果是1699个点,用时1170.7秒。程序的思路虽然清晰,但是有很大的冗余计算量,我们设定的阈值为10米,10米内的点需要弃掉,而大多数点在10米以外,这些点是要保留的,把所有的这些点都计算一遍距离就产生了很大的冗余计算量,为了提高运行效率要尽可能多的减少计算量,因此可以做如下优化设计：(1)把原始数据读取到列表；(2)使用sorted方法对读取到的列表按照x坐标由小到大进行排序；(3)同样选取基点循环计算其它点到基点的距离,这时添加一个中断循环的条件,只要发现点的x坐标与基点x坐标差值大于阈值10米,那么就可以确定后面的点到基点的距离都大于10米,循环就可以中断,这样后面的点就省去了计算；(4)继续下一个基点进行同样的

计算,直到完成所有数据；(5)把得到结果写入文件。另外,为了更大程度的减少计算量,根据原始数据点云的密集程度,也可以先进行随机抽稀。

1.3程序优化后的代码

```
import math
import time
start_time=time.time()
a=r'E:\dsmdata\30.4.xyz'
b=r'E:\dsmdata\30.4-2.dat'
yxz=[]
with open(a,'r') as f1:
    for index, line in enumerate(f1):
        if index % 2 == 0:       #先进行随机抽希减少计算量
            lx = line.split()
            yxz.append([float(lx[0]), float(lx[1]), float(lx[2])])
yxz=sorted(yxz,key=lambda x:x[1])
print('原始数据点数：',index+1)
i=0
while i <len(yxz):
    y=yxz[i][0]
    x=yxz[i][1]
    j=i+1
    while j <len(yxz):
        y1 = yxz[j][0]
        x1 = yxz[j][1]
        if (x1-x)<=10：#设定循环条件
            if math.sqrt((x1 - x) ** 2 + (y1 - y) ** 2) < 10:
                yxz.pop(j)
            else:
                j+=1
        else:                          #超出条件循环中断
            break
    i+=1
print('处理后点数：',len(yxz))
with open(b,'w') as f2:
    for j in range(0,len(yxz)):
f2.write(str(j)+',,'+str(yxz[j][0])+','+str(yxz[j][1])+','+str(
yxz[j][2])+'\n')
end_time=time.time()
tim=end_time-start_time
prin('用时：',tim)
```

程序运行结果：

```
按距离抽希点云-优化后
F:\python\program\venv\Scripts\python.exe
原始数据点数： 843647
处理后点数： 2152
用时： 80.79011344909668

Process finished with exit code 0
```

1.4程序优化前后结果对比分析

同样大小的文件,程序优化后,运行效率明显提升,用时80.8秒,是优化前的用时1170.7秒的6.9%,而处理结果比优化前多出453个点,说明优化后的抽希更符合设定的阈值。把两次的结果展点,随机量测两点之间的距离对比(表格1)：

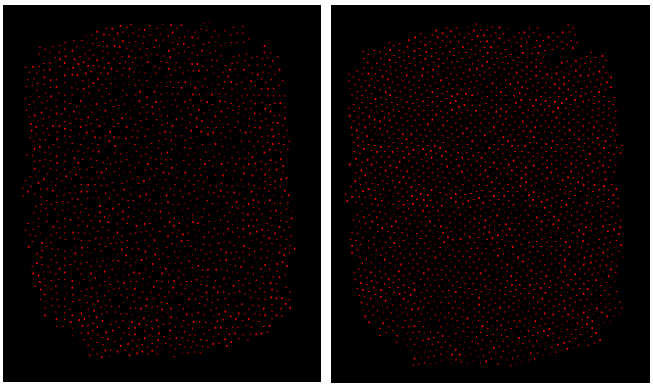

图1 为优化前数据处理结果点位图 图2为优化后数据处理结果点位图

图 1 图 2

表格 1

	指定点	邻近点	距离 Di	阈值 D	中误差
优化前	231	137	10.179	10	$\sqrt{\frac{\sum\Delta\Delta}{n}}$ =0.86
		678	10.420		
		392	10.625		
		414	11.710		
		318	10.497		
优化后	1140	1180	10.072		$\sqrt{\frac{\sum\Delta\Delta}{n}}$ =0.42
		1175	10.331		
		1106	10.001		
		1096	10.807		
		1137	10.328		

从图1和图2可以看出,优化后的数据分布更加均匀;根据表格1的随机抽样计算,得出优化后的中误差是优化前中误差的0.5倍,说明优化后任意两个点之间的距离更接近设定的阈值。

1.5增加高程因素

以上是根据给定阈值,只考虑点与点之间的距离间隔进行数据抽希,在实际的工作中,如果只考虑距离,会降低数据的地形特征,所以高程因素往往也是需要考虑的；算法与上面基本相同,只是在判断条件上再加入一个高差阈值就很容易实现,如果基点的高程和邻近点的高程小于高差阈值,则判定为冗余数据,据此弃掉冗余的数据点。值得注意的是给定的距离阈值与抽希的程度有关,给定的高差阈值与抽希的精度有关,这两个参数可以根据项目的具体要求来给定。

2 结束语

目前点云抽希算法主要分为三类,随机采样算法、基于距离高程算法、基于构建TIN(不规则三角网)算法。随机采样算法是根据一定的规则进行数据采样,以此抽希简化,运行效率最高,但同时严重损失数据精度,适用于对数据精度要求不高或者是只用来显示点云结果的项目；本文所述算法属于基于距离高程算法,这类算法使用较多,能更好的保留地形信息,适用于测图等工程测量；基于构建TIN算法对原始数据精度损失最低,但是由于构建TIN的复杂性导致计算量非常大处理速度较慢,适用于精度要求高、三维建模等项目。因此在工作中应该根据不同的项目要求和目标选择合适的算法。

[参考文献]

[1]钱金菊.LiDAR点云抽希算法研究述评[J].测绘通报,2017(SI):33-35.
[2]胡诚.精度约束下地表LiDAR点云抽希方法研究[D].成都:西南交通大学,2015.
[3]刘法军.无人机倾斜摄影测量在矿山测绘中的应用[J].世界有色金属,2019(19):29-30.

作者简介：

李小勇(1975—),男,河南南召人,汉族,大学本科,高级工程师,注册测绘师,一级建造师,主要从事城市测量、监督测量等工程测量工作。