文章编号：2095-2163(2019)04-0307-06 中图分类号：TP391 文献标志码：A

路面点云的并行简化研究

孙大林，唐好选
（哈尔滨工业大学 计算机科学与技术学院，哈尔滨 150001）

摘 要：为有效解决大规模路面激光点云简化过程中的时间延迟问题，在加速简化过程的同时准确保留特征点，研究了基于斜率差的扫描线点云简化算法及 2 种并行加速方式。首先从路面扫描线点云的分布特点出发，以相邻点间连线的斜率差作为识别特征点的基准，实现了串行简化算法。同时，在研究算法的流程并提取出可并行步骤的基础上，分别设计实现了利用多核 CPU 的并行简化算法和利用 GPU 的并行简化算法。前者依靠 OpenMP 技术，实现的是一种多线程并行；后者在 CUDA 框架下实现，属于 CPU 和 GPU 结合的异构并行计算。在实验阶段的实际路面点云上验证算法执行效果的同时，设计了 3 种算法在不同规模点云数据上的性能测试。通过绘制性能曲线，分析比较了 2 种并行算法的并行效果优劣。最终实现的利用 GPU 的并行简化算法与串行算法比较取得了 100 左右的加速比。

关键词：点云精简；GPU 并行计算；OpenMP

Research on parallel simplification of road surface point cloud

SUN Dalin, TANG Haoxuan
(School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)

【Abstract】 To effectively solve the problem of time delay in the simplification of large-scale road surface laser point cloud, accurately reserving the feature points while accelerating the simplification process, a simplification algorithm based on slope difference of the scanning line point cloud and two parallel acceleration methods are studied. Firstly, based on the distribution characteristics of point cloud of road surface scan line, the slope difference of the line between adjacent points is taken as the benchmark to identify feature points, and a serial simplification algorithm is implemented. At the same time, based on the study of the algorithm flow and the extraction of parallel steps, the parallel simplification algorithm using multi-core CPU and the parallel simplification algorithm using GPU are designed and implemented. The former relies on OpenMP technology to achieve a multi-threaded parallelism; the latter is implemented under the CUDA framework, which belongs to the heterogeneous parallel computing combining CPU and GPU. In the experimental stage, the effectiveness of the algorithm is verified on the actual road surface point cloud. Meanwhile, The performance tests of three algorithms on point cloud data of different scales are designed. By plotting the performance curve, the advantages and disadvantages of the two parallel algorithms are analyzed and compared. The final realization of the parallel simplification algorithm based on GPU has achieved a speedup about 100 compared to the serial algorithm.

【Key words】 point cloud simplification; GPU parallel computing; OpenMP

0 引 言

目前，中国公路网络已经十分发达，从过去的高速建设期步入了以管理养护为主的管养期。而对于里程数已居于全球第二的中国路网来说，单纯依靠人工进行路况检测和养护显然需要大量人力成本。同时还存在着效率欠佳、受天气状态影响等局限。因此引入装载高精度激光传感器的车载移动测量系统，来对路面信息进行建模及缺陷检测就成为了解决本领域诸多局限的关键技术。根据《公路技术情况评定标准》JTGH20-2007 中关于自动化路面状态检测部分的相关要求，车载移动测量系统在工作时需保证一定的速度。而在实际路面环境下，激光扫描设备采集到的路面点云规模庞大，对于建模而言将面临巨大的时间消耗和硬件成本，且由于路面作为被扫描对象的客观状态是总体平整、局部波动的，即很大一部分的点云对于建模工作而言是冗余的。为此将如何合理简化路面点云，一方面旨在减少需要处理的点云总数，另一方面力求突出更能体现局部状态的特征点，就显得尤为重要。

针对上述问题，提出使用并行计算技术。在研究点云简化算法的基础上，根据算法结构对可并行部分使用了并行处理，以提高算法执行效率。依此思路，分别设计实现了基于多核 CPU 的并行算法和

作者简介：孙大林(1994-)，男，硕士研究生，主要研究方向：虚拟现实与图形学；唐好选(1971-)，男，博士，副教授，主要研究方向：虚拟现实技术与应用、计算机图形学。
通讯作者：唐好选 Email：tanghx@ hit.edu.cn

收稿日期：2018-05-26

基于 GPU 的并行算法,并通过实验测试对算法的性能进行了分析。

1 相关工作

近年来,针对不同类型的点云简化已展开了一些研究。首先是对随机分布的点云进行简化的研究,本文虽然是聚焦于车载激光设备获取的路面扫描线点云,但对前者随机点云简化的诸多方法也有着研究上的积极融合与借鉴。比较典型的成果有:文献[1]将机器学习 K-means 聚类算法引入到点云的简化过程中,对于随机散乱点云,可以很好地提取到特征点,并通过删除冗余来维持均匀合适的点集规模。文献[2]和文献[3]都使用了 Kd-tree 分割点云数据空间。前者针对不均匀分布点云,使用了 Kd-tree 建立点间的拓扑关系,从而划定简化的边界,同时还结合局部法向量变化和已保留点作为阈值简化区域内的点云。算法有效减少了不均匀分布点云在简化率过高时产生的空洞。后者首先在空间域对点云进行全局聚类,构建 Kd-tree,并以部分初始节点作为初始化聚类中心,然后利用主成分分析法对其做出多次深入细分,最终把有特征点的聚类映射到高斯球获得的分类结果作为精简结果。另据研究可知,由于简化的关键在于识别和保留特征点,也有很多研究立足于此而陆续发表了一系列成果。较早的则当属 Moenning 等人设计的可以在调用阶段修改密度参数的简化算法。在简化过程中,结合色差和曲率变化决定每个点的特征权值,再依靠权值大小保留特征点,从而精简点云规模[4]。此外,还涌现了把主曲率这一点云的几何特征作为识别特征点的基准的方法,也就是通过对点云进行最小二乘抛物面拟合求出主曲率,并根据点的主曲率 Hausdorff 距离提取及保留特征点[5]。与此类似的是结合了边缘特征和法向量特征的方法,即通过区分边缘特征点和平滑区域特征点来分别进行简化处理,首先识别并保存前者,从而保证点云模型框架的设计完整性。然后依据邻近点的法向量变化在平滑区域保留一定的特征点[6]。上述算法的简化率和适用范围各有不同,但都能较好地保留原始点云的几何特征。

对于以逐扫描线形式存储的点云,文献[7]通过引入 RANSAC 直线估计算法估算每条扫描线的斜率,消除了各条扫描线上因路面颠簸导致的路面点云不规则起伏失真,由此将会得到有效的简化率。文献[8]则设计了基于不同尺度的点抽稀和线抽稀方法,通过分析扫描线上某点的高程和强度,识别并保留特征点,取得了不错的效果。考虑到与随机点云上根据特征进行简化的思路类似,扫描线点云也具备一些可以用来选取特征点的几何特性。从这一角度出发,文献[9]提出了扫描线点云的角度差简化算法和弦差简化算法,以及这 2 种特征结合的角度-弦差简化算法,取得了不错的效果。文献[10]也研发提出了一种基于扫描线点云的自适应角度限制简化算法。该方法通过限制扫描中心角度,移动简化窗口来达到简化目的。

结合本文致力研究的扫描线点云来自路面这一比较平整物体的基础背景,研究可知必然存在不同于其它扫描线点云的高冗余,即非特征点数量占绝大部分。此时就需要简化算法提供很高的简化率,并且在面向大规模点云的同时,简化算法的时间效率也将尤其显得重要。下面将重点探讨本文的简化算法,以及算法设计过程中借助不同并行计算技术实现的优化加速。

2 算法概述

车载激光测量设备获取到的路面点云规模庞大,在空间上分布密集。为明确需要执行简化算法的点云数据的各项特点,文中将给出实验环境下路面点云的相关参数,具体参数设置见表 1。并且,对于每条扫描线上的点数,还将满足如下公式:

$$D = \frac{n * FOV}{R} \tag{1}$$

其中,n 为激光传感器数量;FOV 为横向测量宽度;R 为横向采样的间隔。根据表 1 中的参数和公式(1),计算得到每条扫描线的点数 D 为 1 627。

在此基础上,研究推得设备每秒采集到的总点数 S 为:

$$S = \sum_{i=1}^{n} D_i \times f \tag{2}$$

其中,D_i 为每个激光传感器采集到的点数,f 为每秒钟扫描频率。

根据式(1)、(2)计算得到车载系统每秒采集到的总点数 S 为3 254 000 个。而如此可观规模的点云数据则意味着后期建模时的大量时间消耗。但经由分析可知在这些点中,有相当一部分点对于生成路面的三角网模型却是冗余的。

因此,为了提高系统的建模速度,必须对点云数据进行简化操作。在简化后的特征点云上建模不但可以节省时间,也能使得路面三角剖分模型中各个

三角形形态更趋标准，避免产生过小三角形或狭长三角形来导致模型精度走低。设计简化算法的预期目标是在保留点云特征点的同时，还能大幅度减少冗余点数量。

表 1　相关参数设置

Tab. 1　Related parameter settings

参数名	参数值	说明
Frame Rate	2 000 Hz	每秒钟扫描次数
X Field Of View	1 080 mm	横向测量最大宽度
Measurement Range	500 mm	纵向测量高度范围
Exposure	245 μs	每帧曝光时长
Resample	1.93 mm	X 方向采样间隔

2.1　串行算法流程

分析本文研究场景下点云分布特点，可得其直观表征是全部点云都是离散分布在逐条等距的扫描线上的，这样就使得对每条扫描线点云进行简化就可以得到全部点云的简化。而对于分布在一条扫描线上的点之间，利用相邻点间的斜率差关系可以衡量每个点对于描述路面特征的重要性，即与左、右点间线段斜率差大的可以判定为是特征点将其保留，斜率差小的点则视为连续的平整路面点进行简化。同时为了不把连续平整路面的点全部简化去掉，导致特征点云出现空洞，可设定固定步长值用来保证能在平整的扫描线片段上取到一定数量的点。这里，针对扫描线上相邻点间斜率差关系的示意则如图 1 所示。

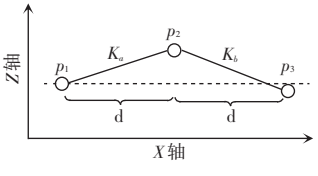


图 1　相邻点间斜率差

Fig. 1　Slope difference between adjacent points

根据图 1 中 $P_1(x_1,z_1)$、$P_2(x_2,z_2)$、$P_3(x_3,z_3)$ 三点位置关系，得到斜率 K_a、K_b 及斜率差 Δ 的数学公式可分别表示如下：

$$\begin{cases} K_a = \dfrac{z_2 - z_1}{x_2 - x_1} = \dfrac{z_2 - z_1}{d} \\ K_b = \dfrac{z_3 - z_2}{x_3 - x_2} = \dfrac{z_3 - z_2}{d} \end{cases} \tag{3}$$

$$\Delta = K_b - K_a = \frac{z_3 - 2z_2 + z_1}{d} \tag{4}$$

对于每组点间的斜率差 Δ 而言，发生变动的只有式(4)中的分子部分。基于斜率差的算法并不需要准确的斜率差值，而是不同组点斜率差变化的程度与阈值的关系，因此在算法实现时对其做出简化处理，以式(4)分子部分代替斜率差。

综上论述分析，研究中采用的基于斜率差的扫描线点云简化算法的设计步骤可分述如下：

输入：原始点云数据

输出：简化后的特征点云数据

Step1　设定斜率差阈值 η，确定需要保留点的步长 M。

Step2　从第一个点开始，指针 P_1，P_2，P_3 分别指向 3 个连续的点。

Step3　计算 $\Delta = z_3 - 2z_2 + z_1$，其中 z_1，z_2，z_3 分别是 P_1，P_2，P_3 的高程坐标，设定计数器 N 值为 0。

Step4　如果 $\Delta < \eta$，删除点 P_2，并把计数器 N 加一。

Step5　如果 $\Delta \geqslant \eta$，或者计算器 N 等于步长 M，则保留 P_2，并把计数器 N 清零。

Step6　判断所有点是否已处理完毕。若是，则当前扫描线已结束简化，可转入处理下一条扫描线数据；否则将 P_1，P_2，P_3 向后移动一个点的距离，再跳至 **Step3**。

2.2　利用多核 CPU 的点云简化算法并行加速

分析实际应用场景中点云的采集和存储模式，发现在 y 轴方向上看，点云是被等距分成逐条扫描线进行保存的。也就是说，同一条扫描线上的点云数据，其 y 坐标是一样的，x 坐标均匀变化布满整条扫描线长度。 同步处理不同扫描线点云并不会引起访存冲突。与此同时，再次考虑到前述简化算法是逐扫描线执行的，那么采用并行方法一次处理多条扫描线点云将会有效提高修补算法的运行速度。设计的并行算法借助 OpenMP 来研发实现，采用动态调度机制将计算任务迭代分配到各个线程，保证计算任务在各线程间做到均匀分布。而在各线程内部，对调度分配输送的扫描线点云数据依然执行 2.1 节中的串行简化算法。可直接在原始点云数据各条扫描线上进行简化，并不需要引入并行后的合并操作。因此采用并行方法一次简化多条扫描线点云，就可有效提高简化算法的执行速度。

以使用 4 核 CPU 的 4 线程执行为例，此时并行执行模式为同时做 4 条扫描线点云的简化，其并行简化算法的设计流程如图 2 所示。

2.3　利用 GPU 的点云简化算法并行加速

在 GPU 上进行算法的并行加速，主要是依托

CUDA 编程框架。CUDA 是一种利用 GPU 解决复杂计算问题的并行计算架构。经由分析探得,一个完整的 CUDA 程序是由一系列的运行于 GPU 上的 Kernel 函数和运行于主机 CPU 端的串行处理步骤共同构建而成,因此利用 GPU 并不只是单纯使用 GPU,而是一种 CPU 和 GPU 的异构计算模型。其中,CPU 端的串行代码的研究包括在 Kernel 启动前预筹的数据准备、设备初始化以及在 Kernel 之间定制的一部分串行化计算。适用于 CUDA 框架的问题需要具备重复性计算多、分支判断少的特点,而本课题面向的大规模扫描线点云简化显然能够符合这些要求。

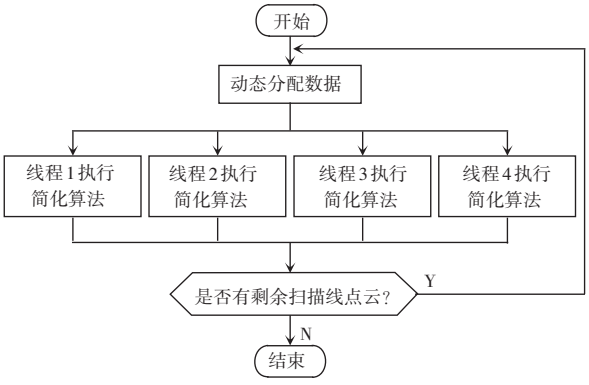


图 2 多核 CPU 并行简化算法流程

Fig. 2 Multi-core CPU parallel simplification algorithm flow

设计时,CUDA 程序需要包含 6 个步骤,设计内容详见如下。

(1)初始化 GPU 设备 Device。

(2)输入数据。

(3)复制数据到 Device 数据,需要提前分配好空间。

(4)执行 Kernel 核函数。

(5)复制得到的结果到 Host 内存。

(6)释放掉申请的 Device 内存。

这里由于各条扫描线点云作为输入数据是相互独立的,可以直接利用这一点进行计算任务的分解。具体是在每一个 GPU 运算单元上执行相同的 Kernel 核函数,而每个 Kernel 核函数的输入数据都是不同的扫描线数据。并且研究中,在 CUDA 框架里,CPU 端被称为主机 Host,GPU 端被称为设备 Device。在完整的 CUDA 程序中,涉及到 Host 和 Device 两边的不同操作,同时也要考虑到数据搬运产生的额外时间消耗。

根据上述对 CUDA 框架下研发算法的相关分析,基于本文简化算法设计的利用 GPU 的点云简化算法的框架流程将如图 3 所示。

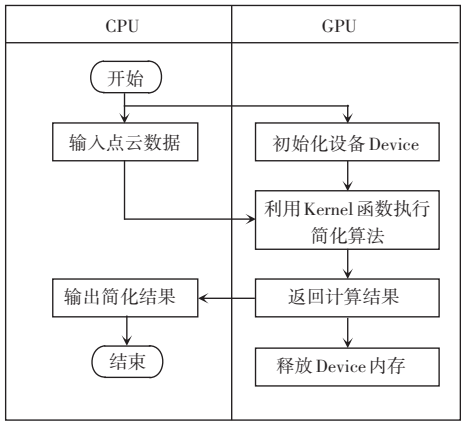


图 3 GPU 并行简化算法流程

Fig. 3 GPU parallel simplification algorithm flow

其中,Kernel 核函数即为执行一条扫描线点云简化算法的改写。由于 GPU 不支持动态的数据结构,就使得在初始化设备 Device 时,要根据输入数据的大小提前分配存储空间,再结合上述对本文实验场景下的扫描线点云分析,可知每条扫描线是定长的,即初始点数固定为 1 627 个,那么根据 CUDA 框架的 Grid×Block×Thread 三层线程编制模式,设计得到的 Device 端线程组织形式如图 4 所示。

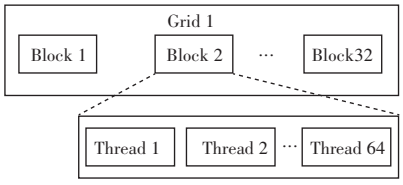


图 4 Device 端线程组织形式

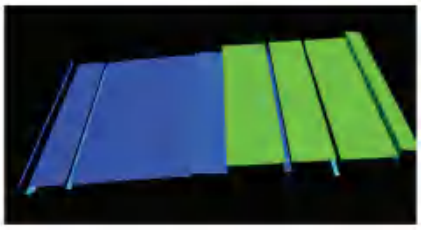
Fig. 4 Device threads organization

这里 CUDA 的线程组织使用的是二维形式,即图 4 中 Thread1 在编程被调用时的序号为(0,0)。另外,GPU 的 stream multiprocessor 在执行时,会把 32 个 thread 合成一个称作 wrap 线程数的单位进行统一调度,因此,设计的 block 内线程数为 wrap 大小的 2 倍,这样既满足了计算要求,也可使硬件并行效率达到了最小化。

3 实验结果与分析

实验的硬件环境为四核 Intel(R) Core i5-5200U 处理器,8 G 内存;操作系统为 Windows 7 64 位。使用的 GPU 为 Nvidia GeForce 940M。

在激光扫描仪采集到的实际点云数据上进行测试,结果如图 5 所示。图 5(a)是实验采集到的原始路面点云分布。图 5(b)为执行简化算法后的结果,由此可以看到在保留路面特征的基础上,简化了大部分冗余点云。



（a）原始路面点云分布

（a）Distribution of point clouds on original road surface

（b）简化后路面点云分布

（b）Distribution of point clouds after simplification

图 5　路面点云上的简化效果

Fig. 5　Simplification result on road surface point clouds

接下来，就将进行算法性能测试，选取从 1 万到 200 万规模的扫描线点云数据各 10 组，分别运行串行简化算法、利用多核 CPU 的并行简化算法和利用 GPU 的并行简化算法，记录平均时间消耗，运算结果数据参见表 2。

表 2　不同简化算法的时间消耗

Tab. 2　Time consumption of different simplification algorithms

点集规模/万	串行算法/ms	多核 CPU 并行算法/ms	GPU 并行算法/ms
1	0.31	1.52	0.17
10	9.37	4.43	0.45
50	61.72	30.08	0.71
100	145.63	73.50	1.42
200	308.55	154.41	2.94

由表 2 可知，其中的点集规模变化呈倍数增长，便于直观比较随此变化而带来的不同时间消耗。根据下文的运算公式（5）及表 2 实验数据绘制得到的算法性能折线则如图 6 所示。

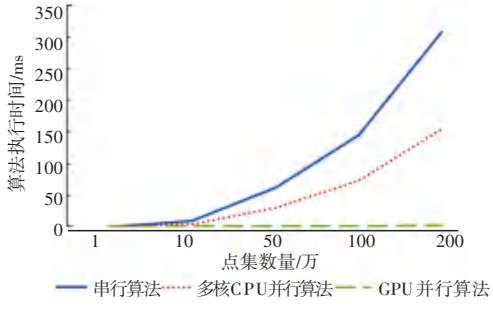


图 6　算法性能折线图

Fig. 6　Algorithm performance line chart

分析图 6 可知，在点云简化问题上利用多核 CPU 的并行加速也可获得有限的加速效果，而相比串行算法而言，在点云规模增大时则基本保持着一倍左右的时间效率提升。

另外，在点集规模很小的时候，使用多线程并行来加速并不能达到加速效果，反而因为增加了线程切换等开销而增加了时间消耗。只有当点集扩大到类似百万这样的规模时，才可真正体现出这样的加速效果。但使用 4 核也只能达到略大于 2 的加速比，与理想状态下的线性加速比 4 尚有一定距离。所以利用多核 CPU 的并行加速可以取得一定范围内的涨幅和加速，但难以达到很好的加速效果。

在同样的实验点云数据上，又测试了利用 GPU 的并行简化算法在各个规模点云上平均时间消耗，发现不仅是与串行算法相比取得了很好的加速效果，与利用多核 CPU 的并行简化算法之间比较也呈现出很高的速度提升。为了更直观比较 2 种并行算法加速效果，引入公式（5），可将其写作如下形式：

$$S = \frac{T_{serial}}{T_{parallel}} \tag{5}$$

其中，T_{serial} 为串行算法执行时间；$T_{parallel}$ 为并行算法执行时间；S 为并行算法达到的加速比。

根据公式（5）及表 2 中的实验数据，绘制 2 种并行算法的加速比随点云规模变化的柱形图可如图 7 所示。

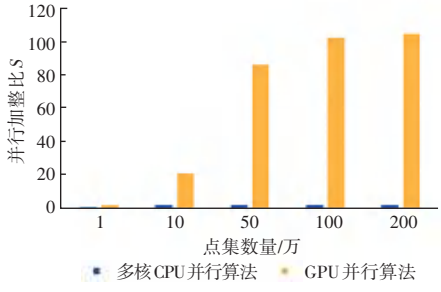


图 7　不同并行算法加速比

Fig. 7　Different parallel algorithm speedup chart

分析图 7 可知，除了在小规模点集上加速效果表现平平外，在扩大点集规模后，利用 GPU 的并行算法的加速比 S 有了显著提高。在模拟实验用到最大规模、即 200 万点的点集里，S 超过了 100。体现在时间消耗上就是相当于将耗时超过 0.3 s 的操作加速到耗时约 0.003 s。由于实际要处理的点集是每秒扫描得到 325 万多点，一段路面在文件形式上可能会是 TB 大小数量级的离散数据文件。这样的加速效果就给后续的路面三角网模型构建省下许多时间。

特别地,点集规模不需要特别大的情况下,并行加速取得的加速比 S 就已十分明显,且随着点集规模扩大,S 增加的幅度也在逐渐加大。究其原因在于前期设定的定长扫描线点数,使得点集规模扩大,加入计算的 GPU 运算单元数量就越多,同时再加上算法本身的特点,这些运算单元之间都是充分并行的,因此就取得了最终可观的加速比。在本文应用场景下,可以直接用激光传感器获得的每条线 1 627 点作为定长进行 CUDA 线程编制的设计(可见 2.3 节图 4)。而对于其它同类型的扫描线点云,则可以根据每条线上的点数设计实验环境的显卡最为适宜的线程组织形式。

4 结束语

本文在研究扫描线点云简化算法的基础上,根据算法结构对可并行部分辅以并行处理,分别使用基于多核 CPU 的并行和基于 GPU 的并行两种模式,提高了算法执行效率。同时又提供了实际路面点云上的实验分析,上述 2 种并行算法分别对比串行算法取得了一定的加速效果。其中,利用 GPU 的并行模式在大规模点云上得到了 100 左右的加速比。比较时间效率可知,对于扫描线点云简化这类重复计算量很大、但分支判断很少的问题,应用 GPU 进行并行优化可以取得相当高的加速比。综上研究可知,通过设计并行的简化算法,在正确保留足够多的特征点的基础上,大幅减少了时间消耗,而且也为后续路面三角网构建和缺陷识别等操作节省了时间。

参考文献

[1] SHI Baoquan, LIANG Jin, LIU Qing. Adaptive simplification of point cloud using k-means clustering[J]. Computer-Aided Design, 2011, 43(8):910-922.

[2] XIAO Zhaoxia, HUANG Wenming. Kd-tree based nonuniform simplification of 3D point cloud[C]// 2009 Third International Conference on Genetic and Evolutionary Computing. Guilin, China:IEEE, 2009:339-342.

[3] 袁小翠, 吴禄慎, 陈华伟. 特征保持点云数据精简[J]. 光学精密工程, 2015, 23(9):2666-2676.

[4] MOENNING C, DODGSON N A. A new point cloud simplification algorithm [C]// 3rd IASTED International Conference on Visualization, Imaging, and Image Processing. Spain: ACTA press, 2003:1027-1033.

[5] 朱煜, 康宝生, 李洪安,等. 一种改进的点云数据精简方法[J]. 计算机应用, 2012, 32(2):521-523,544.

[6] SONG Hao, FENG H Y. A progressive point cloud simplification algorithm with preserved sharp edge data[J]. The International Journal of Advanced Manufacturing Technology, 2009, 45(5-6): 583-592.

[7] 曹毓, 冯莹, 杨云涛,等. RANSAC 直线估计方法在路面三维点云优化中的应用[J]. 红外与激光工程, 2012, 41(11):3108-3112.

[8] 刘如飞, 王鹏. 保留路面特征的车载激光点云非均匀压缩方法[J]. 遥感信息, 2017, 32(1):100-103.

[9] WANG Gefang, LV Yanmei, HAN Ning, et al. Simplification method and application of 3D laser scan point cloud data[C]// Proceedings of the 1st International Conference on Mechanical Engineering and Material Science. Shanghai, China: Atlantis press, 2012:266-268.

[10] GUO J, LIU J Y, ZHANG Y L, et al. Filtering of ground point cloud based on scanning line and self-adaptive angle-limitation algorithm[J]. Journal of Computer Applications, 2011, 31(8): 2243-2245.

(上接第 306 页)

文件系统的数据存储结构原理,探讨了超级块、块组描述符、i-节点表、目录项等要素的含义,可以为专业数据恢复技术人员提供参考。

参考文献

[1] 刘伟. 数据恢复技术深度揭秘[M]. 北京:电子工业出版社, 2010.

[2] 徐国天. 基于 EXT3 文件系统的数据库文件恢复与检验软件的开发[J]. 信息网络安全,2011(10):44-46,70.

[3] 李巍. Ext-扩展文件系统的研究[J]. 信息系统工程,2010(8): 134-135.

[4] 涂健,孙辉. Linux2.6 内核下 Ext3 文件系统的数据结构及性能分析[J]. 南昌水专学报,2004,23(2):8-10,33.

[5] 黄步根. 数据恢复与计算机取证[J]. 计算机安全,2006(6):79-80.

[6] 夏煜,郎荣玲,戴冠中. Linux 操作系统的文件系统建立过程的研究[J]. 计算机工程与应用,2001(15):90-92.