第 43 卷 第 6A 期
2016 年 6 月

计 算 机 科 学
Computer Science

Vol. 43 No. 6A
June 2016

三维激光点云数据的可视化研究

徐旭东 李 泽
(北京工业大学计算机学院 北京 100124)

摘 要 大量的点云数据是通过三维激光扫描得到的,而点云数据的显示快慢受到了数据索引的直接影响,这是一个基础性问题。经过研究,八叉树与叶节点 KD 树相结合的混合空间索引结构以及 LOD 构建的层次细节模型是用来解决点云数据管理与可视化效率不高的问题的有效方法。在局部,通过在叶子节点中构建的 KD 树实现高效的查询和显示;在全局,为了实现快速检索与调度使用了八叉树模型。采用这种混合数据模型进行点云组织,建立空间索引,并对点云数据进行 LOD 构建,实现了点云数据的高效检索以及可视化。

关键词 八叉树,KD 树,点云数据,可视化
中图法分类号 TP391.41 文献标识码 A

Visualization Research of Point Cloud Data in 3D Laser Scanning

XU Xu-dong LI Ze
(College of Computer Science, Beijing University of Technology, Beijing 100124, China)

Abstract 3D laser scanning can obtain a lot of point cloud data, the display speed of which is directly affected by their structure. After the research, a spatial index structure mixing the octree and the leaf node of K-D tree, as well as a level detail (LOD) model is an efficient method to solve the problem of the low efficiency in managing and visualizing the point cloud data. Globally, quick indexing and management can be realized by the octree model. Locally, efficient query and display can be realized by the K-D tree constructed in memory. This mixed data model is adopted to organize point cloud, establish spatial index, and construct point cloud data by LOD, thereby realizing the index and visualization of point cloud data.

Keywords Octree, K-D tree, Point cloud data, Visualization

1 序言

三维激光扫描技术(3D Laser Scanning Technology)是一项先进的获取空间数据的技术,同传统的测量手段相比,三维激光扫描测量技术可以持续、主动、快速地获取大量的目标物表的数据,即点云数据,而且拥有许多特有的优势。它是一种新出现的全自动高精度主动立体扫描技术。因为点云数据量大,所以必须要进行合理的数据组织结构,建立索引,才能避开一次性把全部的点调入内存、系统运行迟缓的问题。

因三维激光扫描的目标物多为不规则形状,且取得的数据为三维目标物点云数据,而八叉树数据结构模型对于体状目标物数据的管理具有独到的优势,所以大多采用八叉树数据结构模型进行组织。根据以往对八叉树及 KD 树进行分析和实际的应用研究后发现,实现简单、自动化水平高是八叉树的特点,可是八叉树最小分割粒度(叶节点)的确定随着点云数据的增多,成为制约其效率的主要原因,如果最小分割粒度太小,随着八叉树的深度加深,后续的定位及查找效率降低,相反,效率也会降低;但是当点云分布不均匀,数据比较大时,这样叶子节点分配到的点的数量会很大,在其中对点进行检索的效率也大大降低。为了提高检索的效率,需要对叶子节点的点云进行二次重新组织。因此研究利用 KD 树在邻域检索方面具有的优势,实现了对叶子节点的点的二次重新组织。

在全局索引区域下层即八叉树叶子节点中加入 KD 树索引来管理点云数据,即在八叉树叶子节点中增加 KD 树索引来管理小区域点云数据,这样 KD 树在邻域查找方面的优势可以得到充分的发挥;在局部点云数据区域的上层用八叉树结构组织数据。最后利用 LOD 建立层次细节模型对不同层次点云可视化进行快速实现。

2 八叉树的组织构建

八叉树是一种用树状数据结构描述三维空间的模型。在八叉树中每个节点代表一个体积大小固定的正方体,每个正方体可以均分为 8 个相等的子正方体,父正方体的体积等于 8 个子正方体的体积之和。

在数据叶子节点的上层构建八叉树组织结构,步骤为:

1)设置最深递归深度;

2)建立第一个立方体,其范围是所有点云数据的最大范围;

本文受国家自然科学基金:基于多维点云与配准影像的结构特征自动探测(41371434)资助。
徐旭东(1960—),男,硕士,副教授,主要研究方向为软件理论、图像处理,E-mail:xuxudong@bjut.edu.cn;李 泽(1989—),男,硕士生,主要研究方向为软件理论、图像处理。

3)把点云元素依次存入没有子节点且在其范围中的立方体里;

4)如果没有达到最深递归深度,则对立方体进行 8 等分,再用 8 个子立方体分别存入各自范围内的父立方体所存的点云元素;

5)如果子立方体所存储的点云元素数量与父立方体是一样的且不为零,则应停止细分;

6)递归步骤 3)至 5),直到最深递归深度。

3 KD 树的组织

KD 树是每个节点都为 k 维点的二叉树。所有非叶子节点都可以视作用一个超平面把一空间分区成两个小空间。节点右边的子树代表在超平面右边的点,节点左边的子树代表在超平面左边的点。选择超平面的方法是:k 维中垂直于超平面的那一维用来划分每个节点的归属。因此,如果选择按照 z 轴来划分,右子树会包含所有 z 值大于指定值的节点,左子树会包含所有 z 值小于指定值的节点。这样,可以用 z 值来确定超平面,z 轴的单位矢量就是它的法矢。

如上所述,KD 树是把整个空间划分为几个特定的部分的空间划分树,并且与它相关的查找操作都在特定空间的内部进行。如果有一个二维或三维空间,KD 树按照一定的划分规则把这个二维或三维空间分成多个空间,如图 1 所示。

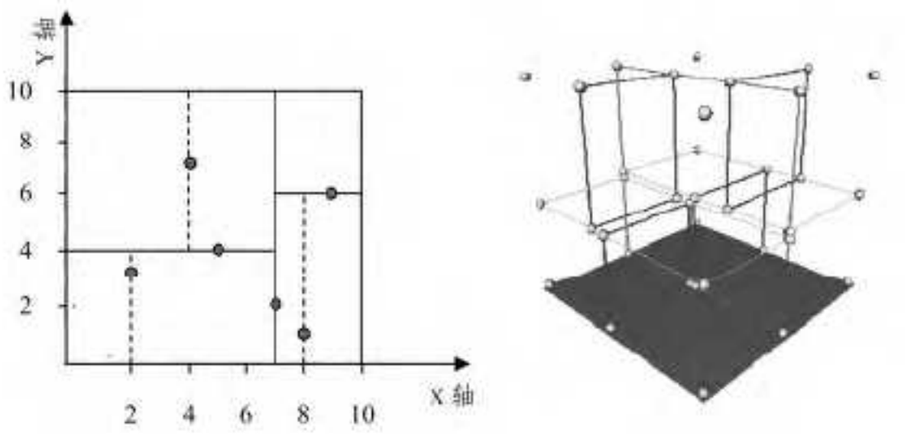


图 1 KD 树划分

在点云数据中,每个点云数据都没有明确给出其对应的几何拓扑信息,只是包含点云的三维坐标值,这样的点通常称为散乱数据点。因此一般点云对应的拓扑关系需要根据点云的邻域关系进行估算,从而估算点云对应的几何信息(如邻接关系、曲率大小、微切平面和数据点单位法向量等)。一种很直观的、但对于海量的点云数据十分耗时并且效率低下的计算某点的 k 个最近邻域的方法即求出其余 $n-1$ 个点与选定点的欧氏距离后,把所有距离值按非递减的顺序进行排序,选定点的 k 最近邻域就是排序前面的 k 个点。

通过对八叉树叶节点部分的点云数据建立 KD 树索引,从而利用 KD 树进行点云邻域搜索,具体如下:假设 KD 树的节点为 i,每一个节点都对应一个区域(根节点对应所有点区域),那么节点 i 所对应的区域为 $Reg(i)$;R 表示要查找的区域范围。范围查询的函数 $Search(i,R)$ 的算法描述如下。

1)首先 $i=$root,即表示从根节点开始查找;

2)如果 i 是叶子节点,那么返回该叶子节点中处于 R 中的点;

3)如果 R 包含 $Reg(i)$,那么返回 i 的所有子树节点;

4)否则,若 $Reg(left(i))$ 和 R 相交,则 $search(left(i),R)$;若 $Reg(right(i))$ 和 R 相交,则 $search(right(i),R)$。

KD 树把整个空间划分为几个特定的部分,然后在特定空间的部分内可以进行相关搜索操作。搜索的核心在于找到实例点的邻居,关于搜索主要有两种搜索模式[7]:范围搜索和最近邻搜索。

KD 树最邻近算法如下:

```
输入:Kd,//KD 树类型
      target//查询点云
输出:nearest,//最邻近点云
      dist//最邻近点云与查询点云间的距离
1. If Kd 为 NULL,则设 dist 为 infinite 并返回
2. //进行二分搜索,生成查找路径
  Kd-point=&Kd;//Kd-point 中保存 KD 树根节点地址
  nearest=Kd-point->Node-data;//初始化最近邻点云
  while(Kd-point)
  push(Kd-point)到 search-path 中;//search-path 是一个后进先出的栈结构,存储着查找路径中的节点指针
  /***
  If Dist(nearest,target) > Dist(Kd-point -> Node-data,target)
      nearest=Kd-point -> Node-data;//更新最近邻点点云
      Max-dist=Dist(Kd-point,target);//更新最邻近点云与查询点间的距离
  ***/
  s=Kd-point->split; //确定待分割的方向
  If target[s]<=Kd-point->Node-data[s]//进行二分查找
  Kd-point=Kd-point -> left;
  else
  Kd-point=Kd-point ->right;
  nearest=search-path 中最后一个叶子节点;
  Max-dist=Dist(nearest,target);//直接取最后叶子节点作为回溯前的初始最近邻点云
3. //回溯查找
  while(search-path != NULL)
  back-point=从 search-path 取出一个节点指针; //从 search-path 堆栈弹栈
  s=back-point->split; //确定分割方向
  If Dist(target[s],back-point -> Node-data[s]) < Max-dist //判断还需进入的子空间
      If target[s] <=back-point -> Node-data[s]
      Kd-point=back-point -> right;//如果 target 位于右子空间,就应进入右子空间
      else
          Kd-point=back-point -> left;//如果 target 位于左子空间,就应进入左子空间
  将 Kd-point 压入 search-path 堆栈;
  If Dist(nearest,target) > Dist(Kd-Point -> Node-data,target)
      nearest=Kd-point->Node-data;//更新最近邻点云
  Min-dist=Dist(Kd-point->Node-data,target);
  //更新最邻近点云与查询点云间的距离
```

对于在全局八叉树块边界上的点,进行 k 邻近搜索时,需要考虑更多的情况。如果只在该 KD 树块内进行查找,最终得到的 k 个点并不是全局最近邻的 k 个点。因此,需要确定一个靠近边界值的距离 d,当要搜索靠近边界距离 d 以内的点的 k 邻近时,就需要将其他靠近该边界的 KD 数块考虑进来,一起进行 KD 邻近搜索。这样我们确保最终 k 邻近是全局 k 邻近。

4 KD 树与八叉树的混合索引

算法的基本思想和步骤如下:

1)首先获取所有点云数据最小外包围盒,并且设置八叉

树的递归深度，依据递归深度再进一步确定八叉树的叶节点的大小。

2）建立对应的八叉树，并把所有的节点信息都存储到八叉树结构中，建立时的参数为上一步获取的递归深度值和叶节点的大小。

3）用 KD 树的数据结构（包含节点坐标信息和索引信息）组织方式把每个八叉树的叶节点中的点云数据进行组织，同时八叉树叶节点中还要保存 KD 树的首记录。

4）在进行邻域查找时，要特别注意的是，扩大查找范围的条件是当当前点云所在的八叉树叶子节点不能找到满足要求的点云数据时，要把查找范围扩大到邻近节点进行再次查找，直到得到满足要求的点云数据。然后在查询数据时，最终的查找是要在 KD 树点云数据集中按照查找的条件进行的，结束后返回满足要求的点云数据。而要查找哪个 KD 树需要先判断点云数据在哪个叶子节点中，进而在叶子节点中确定要查找的 KD 树。

点云的组合嵌套组织结构如图 2 所示。

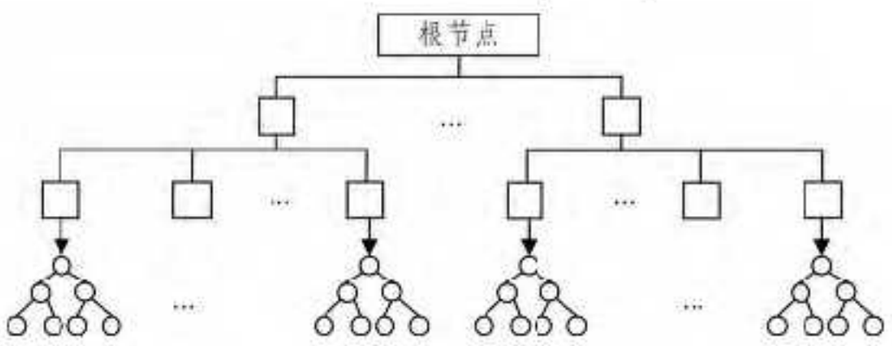


图 2　KD 树与改进八叉树混合索引结构

这样分别利用 KD 树、八叉树以及 KD 树、八叉树的混合索引对点云数据进行组织，并对相同的几个指定点进行查询。当前运行环境为宏碁 Intel(R) Core(TM) i5-4590 CPU @ 3.30GHz，内存为 8GB，其构建以及查询的统计如表 1 所列。从表中可以很直观地看出单一利用八叉树进行点云的组织，随着八叉树深度的加深，会导致查询的效率显著下降，而且用这种结构进行点云的组织查找时，时间消耗得最多。但是单一的 KD 树组织的点云可以明显提高查询的效率，不过如果点云数量增加，其效率也有比较明显的下降。为了大大改善查询点的效率进而取得有利于对点云数据的查询、可视化与交互效果，必须利用混合结构的索引方式。上述组合嵌套结构可以有效划分各个节点中的点云数量。

表 1　查找性能对比(ms)

结构	1044531(点数)	3113452(点数)	6265372(点数)
八叉树	6390	24325	65687
KD 树	825	2176	5443
八叉树+KD 树	257	801	1481

5　点云数据 LOD 构建

这一步骤应用 LOD 算法，即基于分块分层 KD 树的 LOD 算法，它是一种基于 TIN 的、动态和静态相结合的 LOD 算法。该算法分为预处理和运行时两部分，预处理主要负责点云数据的预处理和对所有点云进行一次 Delaunay 三角网的构建，运行时部分是在预处理数据的基础上，基于视点动态实时地进行三角网的更新。

在预处理阶段，需要对点云数据进行多分辨率分层处理。在运行时实时处理阶段，当视点改变时，先对内存中的 LOD 模型数据进行动态更新，最终由图形硬件绘制。

5.1　预处理阶段

预处理阶段，算法对所有点云进行分块、分层，并对它们进行一次 Delaunay 三角网的构建。对点云进行分块分层，使用上述的空间存储管理策略中的空间索引技术，即八叉树分块与 ID 树块内分层结合的空间索引技术；三角网的构建使用了 Isenburg 的流式 Delaunay 三角化（Streaming Delaunay Triangulation）。该方法的优点是速度快且仅需使用很少的内存空间，这是由于它利用了点云数据一般是区域连续存储这一常被人忽略的属性。

预处理部分有以下几个步骤：

1）进行流式 Delaunay 三角化；

2）记录每一分块内的点云包围圈以及沿着分块间边缘的三角带；

3）对每个分块除去点云包围圈外的点云进行 KD 树空间划分；

4）将每一分块的数据写入外存的缓冲数据文件；

5）进行下一个分块的流式 Delaunay 三角化，直到完成所有分块。

以上是整个 LOD 预处理流程。预处理除了完成基于分块分层 KD 树的空间索引和外存调度管理技术之外，最重要的是找出了块内的点云包围圈和块间的分割三角带。块间的分割三角带将各个分块的三角网独立开来。于是，各个块内的点云包围圈内的点云可以各自独立地进行 Delaunay 三角化。

5.2　运行时 LOD 实时更新

运行时阶段分为两个重要的部分：简化的 LOD 模型层次切换准则和动态更新 Delaunay 三角网。前者负责对 LOD 模型中多个层次的切换进行决策，后者负责实施模型层次切换的最关键的步骤，即三角网更新。

5.2.1　简化的 LOD 模型层次准则

视点改变时，可能需要改变 LOD 模型的分辨率，即 LOD 模型的层次。而决定是否需要改变和如何改变的是 LOD 模型层次切换准则。

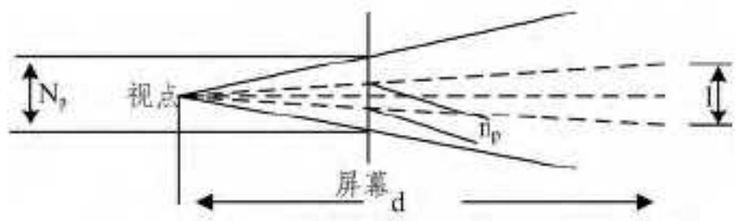


图 3　视点成像屏幕误差图

如图 3 所示，长度为 l 的物体在与它相距 d 的视点的观察下，被投影到屏幕上。其中 a 为视角，N_p 为屏幕在视点视野范围内的纵向切线上的像素数目，n_p 是该物体投影到屏幕上在该切线上占用的像素数目。各个量间有如下式关系：

$$n_p = \frac{l}{2d\tan\frac{a}{2}} N_p \tag{1}$$

当 $n_p = l$ 时，即此时物体投影到屏幕在切线上占用一个像素时，上式变为式(2)：

$$l = \frac{2d\tan\frac{a}{2}}{N_p} \tag{2}$$

此时长度小于或等于 l 的物体在屏幕上被压缩在一个像素内，这时的物体细节已经无法分辨了，所有在该物体上的点云只需用一个顶点表示即可。

式(2)就是 LOD 算法的模型 Cecil 切换的基本准则。但是如果用式(2)对每个点进行实时的判断，计算量将是惊人的。因此很有必要对它进行简化，将分块而不是点云作为判

断的最小单位。其基本的思路是:当分块内当前层点云的平均点距小于或等于 l 时,即认为模型层次切换条件满足,此时模型将被切换到更粗糙的层次,满足新的层次中分块内点云平均点距大于 l;相对地,如果存在更细节的层次中点云的平均点距也大于 l,则也进行切换,保证当前层次中点云的平均点距恰好大于 l。

由于屏幕的分辨率和视角对于所有的分块是一样的,式(2)可以进一步简化为式(3):

$$l=k_e d \tag{3}$$

在预处理时,预先将每一个分块中的各个层次点云的平均点距计算出来。当视点发生变化时,更准确地说是视点和各个分块的距离 d 改变时,根据式(3),就可以得到每个分块内所需要达到的最小点距 l。于是,查找预先计算出来的该块内各层次点云的平均点距,直到找到一个恰好大于 l 的层次,这就是该分块需要切换到的层次。在这个过程中,CPU 的计算量相当小。

5.2.2 动态更新 Delaunay 三角网

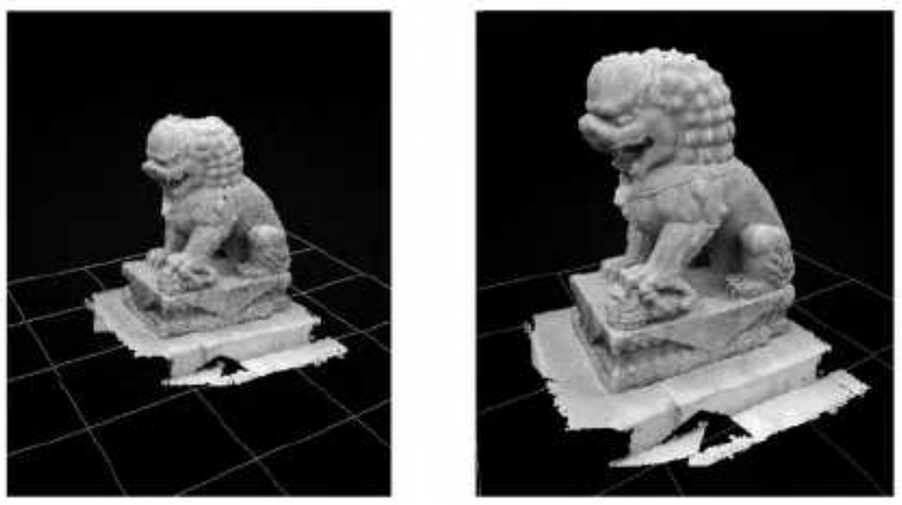
当视点改变时,各个分块内的模型层次需要改变,这样,要显示的点云和三角网都需要更新。对于点云,仅仅将需要的数据读入内存即可;而三角网需要动态地进行更新。

与以往经典的基于 TIN 的 LOD 算法经历避免实时地进行 Delaunay 三角网的重建不同,这个算法在视点改变时会通过实时地进行 Delaunay 三角网的重建来实现三角网的动态更新。

6 实验结果

在 Intel(R)Core(TM)i5-4590 CPU @3.30GHz,RAM 4.0GB Windows7 系统下利用 QT 作为前台的开发工具,利用 OpenGL 作为三维的开发引擎,利用 C++ 开发实现了对扫描取得的点云的预处理、可视化显示的操作。同时,利用 FARO 三维激光扫描仪获取的数据,测试了相关算法的可行性,同样实现了对点云的可视化显示。

基于上述的八叉树以及 KD 树混合索引和 LOD 构建以及以上分步实验,最终处理的效果如图 4、图 5 所示。



(a)远距离　　(b)中距离

图 4

图 5　近距离

图 4(a)、图 4(b)、图 5 分别为远、中、近距离时系统的 3 种不同的模型分辨率。这种差别在显示一个固定的局部细节时可以看出来。

实际上,这 3 种模型分辨率对应的全局点云数目和三角形数目如表 2 所列。

表 2　3 种不同分辨率对比

远近	全局点数	全局三角数	实际点数	实际三角数
近距离分辨率	919521	1723649	74153	120247
中距离分辨率	195437	418233	71864	117389
远距离分辨率	63023	72479	47926	73528

由表 2 可知,本文的 LOD 算法能很好地降低要绘制的点云和三角形的数目,从而大大提高显示速度。

结束语　本文算法可以实现大量三维点云数据在通用配置机器上的轻松浏览查看。首先在全局通过八叉树分块操作降低大型数据处理的复杂性,并在局部八叉树分块的内部使用分层的 KD 树组织点数据,动态控制各块数据对应的多分辨率数据的 Delaunay 三角网更新。通过视点区域的定义完成当前视点下的多分辨率 Delaunay 三角网绘制,并借助基于视点的调度策略实现对当前视点区域以及整个点云数据表面绘制的实时更新。实验结果验证了本文算法的有效性。

参 考 文 献

[1] 李春鑫,彭认灿. 基于三维 Delaunay 三角化的快速可视化方法[J]. 计算机科学,2015,42(6A):236-237,248

[2] 程效军,贾东峰,程小龙. 海量点云数据处理理论与技术[M]. 上海:同济大学出版社,2014:13

[3] 邱春丽,许宏丽. 一种散乱点云空间直接剖分算法[J]. 计算机科学,2014,41(2):157-160

[4] 游安清,韩晓言,李世平,等. 激光点云中输电线拟合与杆塔定位方法研究[J]. 计算机科学,2013,40(4):298-300

[5] 廖丽琼,白俊松,罗德安. 基于八叉树及 KD 树的混合点云数据存储结构[J]. 计算机系统应用,2012,21(3):88

[6] 徐鹏. 海量三维点云数据的组织与可视化研究[D]. 南京:南京师范大学,2013

[7] 姚定忠. 海量地形点云可视化与处理系统的研究[D]. 广州:华南理工大学,2013

[8] 王儒,胡萍,蒋俊豪. 基于 KD 树的点云索引技术研究[J]. 科技视界,2015(30):111

[9] 缪君,储珺,张桂梅,等. 基于稀疏点云的多平面场景稠密重建[J]. 自动化学报,2015,41(4):813-822

[10] Elseberg J, Borrmann D, Nüchter A. One billion points in the cloud-an octree for efficient processing of 3D laser scans[J]. ISPRS Journal of Photogrammetry and Remote Sensing, 2013, 76: 76-88

[11] Goswami P, Erol F, Mukhi R, et al. An efficient multi-resolution framework for high quality interactive rendering of massive point clouds using multi-way kd-trees[J]. The Visual Computer, 2012, 29(1): 69-83

[12] van Oosteroma P, Martinez-Rubib O, Ivanova M, et al. Massive point cloud data management: Design, implementation and execution of a point cloud benchmark [J]. Computers & Graphics, 2015, 49: 92-125