基于几何特征枝干点云骨架提取最短路径算法

杨杰[5,2]，温小荣[1,2*]，叶金盛[3]，汪求来[3]

（1. 南京林业大学南方现代林业协同创新中心，江苏 南京 210037；2.南京林业大学林学院 ，江苏 南京 210037；3.广东省林业调查规划院，广东 广州 510520）

摘 要：树木建模广泛应用于林业信息化等领域，点云各项优良特性使其也称为树木建模主要方法。基于几何特征的树木枝干点云骨架提取中以根节点距离相似归类的方法在枝条分叉处更加合理，而该方法的实际应用受制于传统使用的最短路径求解算法的 Dijkstra 算法因而较少。本研究主要针对树木枝干点云，将现有典型最短路径算法进行相应的改进以应用于基于几何特征的树木枝干点云骨架提取中。通过实际数据验证可知，利用邻接表能够大幅度降低内存需求，相较于以往采用的 Dijkstra 算法，SPFA 的执行速度是理想的，更加快速，能够对精细化点云树木建模提供帮助。

关键词：点云；树木建模；骨架提取；最短路径

Shortest Path Extraction Algorithm of Tree Branch Point Cloud Skeleton Based on Geometric Characteristics

Abstract: Nowadays, tree modeling is widely used in computer games, forestry informatization and other fields. The excellent characteristics of point cloud make it also known as the main method of tree modeling. In the field of extracting tree branch point cloud skeleton based on geometric characteristics, the method that uses the similarity of the shortest-path distance between each point and the root point to classify is more reasonable in trunk bifurcations, but this method has less practical application as it is subject to Dijkstra algorithm – a traditional algorithm of solving the shortest path. This research mainly aims at the tree branch point cloud, and improves the existing shortest path algorithm to be applied to extracting tree branch point cloud skeleton based on geometric characteristics. Through the verification of actual data, it can be seen that the use of adjacency table can greatly reduce the memory requirements. Compared with the previous adopted Dijkstra's algorithm, the execution speed of SPFA is ideal and faster, which can provide help for the fine point cloud tree modeling.

Key words: point cloud; tree modeling; skeleton extraction; shortest path

经过近 50 年的研究，树木建模已广泛应用于多个领域的活动中，如电脑游戏和林业信息化。不断地提高树木建模的精度，可以增加游戏场景的沉浸感；同时，又极大地促进林业信息化，实现精准林业（张玥焜等，2021；Pyataev，2018）。激光雷达技术（Light Detection and Ranging，LiDAR）正逐步成为林业调查的重要手段，其扫描得到的点云数据具有精度高、时效性好等特点，因而成为林业信息化中的研究热点（曹伟等，2021）。点云数据可把现实世界原子化，是将现实世界中以大量且密集的点的形式将被测量区域内物体的表面表达出来，因此通过高精度的点云亦可以还原现实世界，通过研究某一研究目标在点云中的信息来反演其现实世界的信息（HE *et al.*，2020），利用点云数据进行树木建模的意义也正是如此。通过点云数据进行测量与传统的人工测量相比，生物量、产量、叶面积指数等相关参数获得的

基金项目：广东省林业科技创新基金项目(2021KJCX001)；国家重点研发计划（2016YFC0502704）；江苏省高校优势学科建设工程自助项目（PAPD）

* 温小荣为通讯作者

更加准确（夏明鹏等，2015）。基于点云、基于图像、基于规则或过程和基于草图共同构成现在树木建模的四大主要方法。

单木建模由枝干建模和叶片建模两部分构成，其中，枝干模型表达的一种方法是骨架，此外比较常见的方法还有 Delaunay 三角网等。石银涛指出，现有的研究中点云树木枝干模型骨架的提取算法主要基于三类——体素空间（Voxel Space，VS）、点云收缩（Point Clouds Contract，PCC）、几何特征（Geometrical Characteristic，GC），其中，GC 算法仅在运算效率中未达到最优，次于 PCC 算法，而在此外的各个方面均达到最优或并列最优，包括处理细小枝、模型拓扑、整体效果、处理邻近分支和中心偏差（石银涛，2018）。以牺牲较小的时间代价换取更高的建模精度满足大部分研究者的研究需求，以中国知网为例，搜索关键词“点云树木建模”，最早一篇发表于 2011/12/1，截至 2021/10/24 共计 40 篇文章中，排除文献综述（4 篇）、未直接指明骨架提取方法（包括直接使用三维建模软件的 3 篇）、非骨架提取（17 篇）的文献以外，剩余 16 篇中，有 2 篇是 VS 算法，1 篇是 PCC 算法，其余共 13 篇均为 GC 算法，可见其研究与应用之多。

GC 算法的思路是将枝干点云按照一定规则分成若干 bins，再将每个 bins 通过聚类拆分成一个或多个 bin，对每一个 bin 求出中心点，按照真实关系将点连接即可得到枝干点云（XU *et al.*，2007），bin 有文献译作枝干段，但 bins 却无对应的翻译，根据实际意义可解释为“枝干段集”，bins、bin 与最后建立的骨架的示意图如图 1(a)至(c)所示。

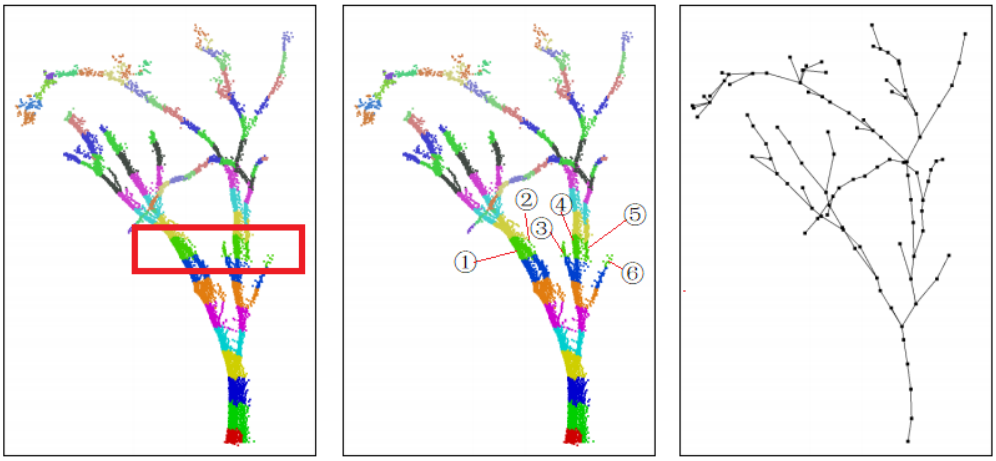

图 1　bins、bin 和枝干骨架

Fig.1　bins,bin and branch skeleton

目前 GC 算法采用的规则分为两种，以根节点距离相似归类和以高度归类。生态学研究表明：树木在水分养分的传输中趋向使用最短路径来达到资源配置的最优化（FAN *et al.*，2020；DONG *et al.*，2020），而以根节点距离相似归类的思想正是量化各点与根节点的最短路径的长度按距离“相似”的划分成各个区域，以此作为 bins。其中，根节点是指经过校正后的枝干点云中 z 轴高度最低的点，它代表在地面之上树根的位置；相似指的是按照距离的大小进行分组。而以高度归类是直接以根节点和树冠最高点的高度差直接分段。记 N 为 bins 分成的数量，分别取 N=20,50,100 比较两种规则分段后的结果以及 N=100 时得到的骨架，如图 2 所示。

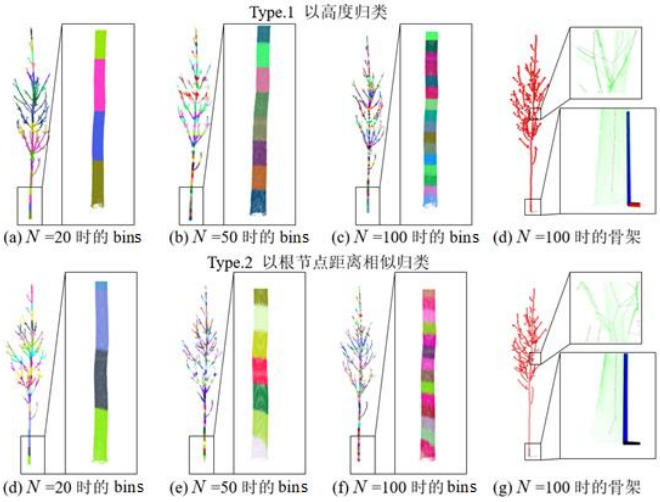

图 2 两种规则的 bins 效果及骨架建立效果

Fig.2 Bins effect and skeleton building effect of two rules

由图 2 可以看出，以根节点距离相似归类区别于以高度归类之处在于以根节点距离相似归类得到的 bins 以根节点向外呈现辐射状；当 N 增大时，根处的 bins 的形状呈现单侧塌陷，并在 N 充分大时直观上不闭合，这将会导致根节点明显偏移树木根部中心位置，产生这一问题原因是由于点云获得的是树木表面的信息，因此根节点在树木表面上，并非树根处的中心位置，现已有研究对这一问题进行了改进（王斌，2015），然而目前很多使用了以根节点距离相似归类方法的文献未注意到这一问题，未改进时在根节点处存在明显的劣势，但以 N=100 时的两种规则建立出来的骨架进行对比，可见其在枝条分叉处的处理更加合理，分叉处得到的骨架中心点更靠近枝条真实的中心位置，有利于后续骨架的优化。

现有的大多数文章虽然介绍了以根节点距离相似归类并提及了 Dijkstra 算法求解最短路径，但通过前面介绍的两种规则得到的 bin 的差异，可以判断出其中由很多文献仍然使用的是以高度归类的规则来建立骨架，一方面可能是由于采用了不合适的数据结构，另一方面是 Dijkstra 算法本身计算复杂度高，因而在处理海量点云时执行速度较慢。

本研究主要针对基于几何特征的树木枝干点云骨架提取时，考虑到枝干点云数据量大，讨论了图的表达方式，对经典的最短路径算法及其改进的适用性进行了讨论，并对适用的最短路径的算法各自作一定的改进以适用于枝干点云且满足内存节省与快速处理的需求。实验使用 C++编程实现，以真实树木点云比较若干算法的实际效果。

1 相关研究

图 $G=(V,E)$ 的典型表示方法包括邻接矩阵与邻接表两种方法（ROBERT，2011；CORMEN et al.，2009），性能复杂度如表 1 所示。

表 1 性能复杂度对比

Table.1 Performance complexity comparison

数据结构	所需空间	新增边 (v,w)	遍历 v 的所有相邻顶点
邻接矩阵	$T(\|V\|^2)$	$O(1)$	$O(\|V\|)$
邻接表	$T(\|E\|+\|V\|)$	$O(1)$	$O(\deg\|V\|)$

对于枝干点云，若点数非常大，采用邻接矩阵会对内存的需求非常大，以本研究将使用的杨树的点云为例，枝叶分离后得到的枝干点云点数为 $|V|=19850$（后文使用 $|P|$），邻接矩阵所需的理论内存为

$198500^2 \times sizeof(double) \approx 293.57GB$，这远远超过了一般设备的承受范围，而且由于并不是任意两点均可达，事实上每一个点只和本身附近很小范围内的点可达，因而该矩阵中很多元素都是∞，占用了过多不必要的空间。

求解图的最短路径问题的常用算法包括 Floyd 算法、Dijkstra 算法和 Bellman-Ford 算法。由于 Floyd 算法中需要存储最短距离矩阵所需空间为 $T(|V|^2)$，因而不适用于海量点云，不再进一步讨论。另外两种算法适用于海量点云，但仍有一定的改进空间，对于 Dijkstra 算法，用堆进行优化；对于 Bellman-Ford 算法，用队列进行优化得到 SPFA 算法，而 SPFA 算法可进一步优化，可采用三种策略，包括 Small Label First（SLF）、Large Label Last（LLL）、SLF 与 LLL 结合（SLF+LLL）（KARBOWSKI et al.，2007），若干算法的时间复杂度如表 2 所示。

表 2 最短路径算法时间复杂度对比

Table.2 Comparison of time complexity of shortest path algorithms

	Dijkstra 算法	堆优化的 Dijkstra 算法	Bellman-Ford 算法	SPFA 算法	SPFA 算法的 SLF 优化	SPFA 算法的 LLL 优化	SPFA 算法的 SLF+LLL 优化
时间复杂度	$O(\|V\|^2)$	$O(\|E\|\log\|V\|)$	$O(\|V\|\|E\|)$	$O(k\|E\|)$，$k<<V$，最坏为 $O(\|V\|\|E\|)$	与 SPFA 相同	与 SPFA 相同	与 SPFA 相同

由于从 LiDAR 设备获取的枝干点云是以无序点云的形式保存的，点云中的点的序号并不能反映出一定的规律。由枝干点云建立的边是每一个点与其附近一定范围内的点的直线距离，并不是负数，因而不存在负权边。这些算法现有的实现中所使用的邻接表并不全都满足可处理无向图，无需考虑负权边或尽可能节省内存使用，即仍然使用邻接矩阵实现，或应用于点云时算法有不必要的判断增加了时间消耗，或采用的邻接表存放的数据结构不够精炼，因而在获取枝干点云的最短路径方面，若干算法需要进一步地改进。

2 方法

2.1 初始化阶段

首先，对获得的树木点云枝叶分离，利用 k-d tree 以半径 R 进行最近邻检索，得到枝干点云，半径的选取应当以保证枝干上的每个点都应当至少有一个邻近点且足够小为宜。再次使用 kd-tree 以半径为 r 进行最邻近检索以获取图，其中，务必保证 r≥R，且考虑到枝干点云中每个点只能和附近一定范围内的点可达，因而 r 不宜太大。表 3 给出使用的符号及其含义，获取边和参数初始化的算法如算法 1 所示。

表 3 符号及其含义

Table.3 Symbols and their meanings

符号	含义
P_i	枝干点云 P 中序号为 i 的点
$S_i(r)$	点云中序号为 i 的点以半径 r 进行最邻近检索得到的近邻点在点云中的序号构成的序列，该序列以距离由近到远依次顺序存储，且 $i \notin S_i(r)$
E	邻接表，其中 E_i 为起始序号为 i 的边的序列，$E_i = \bigcup_{j \in S_i(r_2)} \left\{\left(j, \left\|\overrightarrow{p_i p_j}\right\|_2\right)\right\}$。对某一 $e=\left(j, \left\|\overrightarrow{P_i P_j}\right\|_2\right) \in E_i$ 用 $e[0]$、$e[1]$ 分别表示终

	止序号 j 与距离 $\left\|\overrightarrow{P_iP_j}\right\|_2$ $\|E\|$ 表示邻接表中边总数
x	设置的根节点的序号（该序号作为源节点）
D	存储当前最短路径距离的序列，其中，D_i 为 P_i 与 P_x 间当前最短路径距离
F	存储当前最新路径的序列，其中，F_i 为 P_i 与 P_x 间当前最短路径中与 P_i 相连的点（若只需求最短路径距离，则可以将与其相关的从代码中去掉）

算法 1 邻接表的建立及参数初始化

Algorithm.1 Establishment of adjacency table and parameter initialization

E.resize($|P|$)
D.resize($|P|$)
F.resize($|P|$)
kdtree.init()
kdtree.setInputCloud(P);
for $i := 1 \to |P|$ do
$S_i(r) :=$ kdtree.radiusSearch(P_i, r)
foreach 序号 $j \in S_i(r)$ do
    E_i.push_back($(j, \|\overrightarrow{P_iP_j}\|_2)$)
end for
if $i \neq x$ then
    $D_i := +\inf$
    $F_i :=$ NULL
else
    $D_i := 0$
    $F_i := i$
end if
end for

2.2 适用于枝干点云的最短路径算法

2.2.1 Dijkstra 算法 Dijkstra 算法基于广度优先搜索思想，从选定的源节点开始，跟踪每个节点到源节点当前已知的最短路径距离，如果发现比当前更短的路径，则更新最短路径距离值。一旦找到源节点和某一节点间的最短路径，就标记该节点为“已访问”，并在路径表中添加到该节点的最新路径。持续该过程直至所有节点都在路径表中有路径，此时的路径表中每一条路径遵循了源节点可能到达的每个节点的最短路径。

实际对路径的存储时，只需存放与每一节点相连接的那一段路即可，一方面，由前述的 $r \geqslant R$ 保证，图势必是连通的，通过逆向找查就可以回溯得到完整路径，另一方面，能够节省空间并提升算法的执行速度。

算法 2 适用于枝干点云的 Dijkstra 算法

Algorithm.2 Dijkstra’s algorithm for branch point cloud

初始化标记序列 B，其中 $B_i :=$ false，$i = 1, 2, \cdots, |P|$
for $i := 1 \to |P|$ do
$v :=$ NULL
for $j := 1 \to |P|$ do
    if $B_j =$ false and ($v =$ NULL or $D_v > D_j$) then
        $v \leftarrow j$

end if
end for
if $v = \text{NULL}$ then
break
end if
$B_v \leftarrow \text{true}$
foreach 元组 $e \in E_v$ do
$w := e[0]$
$d := e[1]$
if $B_w = \text{false}$ and $D_w > D_v + d$ then
$D_w \leftarrow D_v + d$
$F_w \leftarrow v$
end if
end for
end for

2.2.2 堆优化的 Dijkstra 算法 Dijkstra 算法的一个缺点是遍历计算的节点很多，因而效率低，尤其在面对节点多或者边多的大图的处理时。堆（Heap）优化是利用一个每次弹出其中最小元素的优先队列（Priority queue）来代替最近距离的找查，堆优化的 Dijkstra 可以大幅节约时间开销。

算法 3 适用于枝干点云的堆优化的 Dijkstra 算法

Algorithm.3 Dijkstra’s Algorithm implemented with a Min-Heap for branch point cloud

初始化优先队列 Q，比较方式为升序
Q.push ($(x, 0)$)
while Q 不为空 do
$q := Q$.top()
Q.pop()
$v := q[0]$
if $D_v < q[1]$ then
continue
end if
foreach 元组 $e \in E_v$ do
$w := e[0]$
$d := e[1]$
if $D_w > D_v + d$ then
$D_w \leftarrow D_v + d$
$F_w \leftarrow v$
Q.push((w, D_w))
end if
end for
end while

2.2.3 Bellman-Ford 算法 Bellman-Ford 算法（或称 Bellman-Ford-Moore 算法、距离向量算法）通过简单地对所有边松弛，以自下而上的方式搜寻最短路径。首先寻找路径中只有一条边的距离，之后增加路径长度以找到所有可能的解决方案。

算法 4 适用于枝干点云的 Bellman-Ford 算法

Algorithm.4 Bellman-Ford algorithm for branch point cloud

while true do
  $b := \text{false}$
for $i := 1 \to |P|$ do
  foreach 元组 $e \in E_i$ do
    $w := e[0]$
    $d := e[1]$
    if $D_i \neq +\inf$ and $D_w > D_i + d$ then
      $D_w \leftarrow D_i + d$
      $F_w \leftarrow i$
    end if
  end for
  end for
end while

2.2.4 SPFA 算法 SPFA（Shortest Path Faster Algorithm）算法采用对一个队列进行维护以使在一个结点当前的最短路径更新后无需立即更新其他结点，大大减少了重复的操作次数。但是在最坏情况下，每一个节点都入队 $|V|-1$ 次，此时也就退化成了 Bellman-Ford 算法（赵卫绩等，2018；郑海虹，2013）。

算法 5 适用于枝干点云的 SPFA 算法

Algorithm.5 SPFA for branch point cloud

初始化标记序列 B，其中 $B_i := \text{false}$，$i = 1, 2, \cdots, |P|$
$B_x \leftarrow \text{true}$
初始化队列 Q'
Q'.push(x)
while Q' 不为空 do
$v := Q'$.front()
Q'.pop()
foreach 元组 $e \in E_v$ do
  $w := e[0]$
  $d := e[1]$
  if $D_w > D_v + d$ then
    $D_w \leftarrow D_v + d$
    $F_w \leftarrow v$
    if B_w =false then
      Q'.push(w)
      $B_w \leftarrow \text{true}$
    end if
  end if
end for
end while

2.2.5 SPFA 算法的三种优化 SLF 策略即把较小的元素提前，利用双端队列，如果新入队的元素

比队首的元素更小，则加入到队首，否则排到队尾。

LLL 策略则通过比较队列中元素的最短路径距离的平均值和待压入队列的元素的当前最短路径长度，如果大于平均值，则压入队尾。

SLF 与 LLL 策略并不冲突，因而又可以同时被采用。在某些图求解最短路径的问题上，SPFA 算法的 SLF 优化比 SPFA 算法快 10%~20%，而 SPFA 算法的 SLF+LLL 优化甚至能比 SPFA 算法快近 50%。

算法 6　适用于枝干点云的 SPFA 算法的 SLF 优化

Algorithm.6　The SLF algorithm for branch point cloud

初始化标记序列 B，其中 $B_i :=$ false，$i=1,2,\cdots,|P|$
$B_x \leftarrow$ true
初始化双端队列 Q''
Q''.push_back(x)
while Q'' 不为空 do
$v := Q''$.front()
Q''.pop_front()
$B_v \leftarrow$ false
foreach 元组 $e \in E_v$ do
　$w := e[0]$
　$d := e[1]$
　if $D_w > D_v + d$ then
　　$D_w \leftarrow D_v + d$
　　$F_w \leftarrow v$
　　if B_w =false then
　　　$B_w \leftarrow$ true
　　　if Q'' 不为空 and $D_w < D_{Q''.\text{front}()}$ then
　　　　Q''.push_front(w)
　　　else
　　　　Q''.push_back(w)
　　　end if
　　end if
　end if
end for
end while

算法 7　适用于枝干点云的 SPFA 算法的 LLL 优化

Algorithm.7　The LLL algorithm for branch point cloud

初始化标记序列 B，其中 $B_i :=$ false，$i=1,2,\cdots,|P|$
$B_x :=$ true
初始化队列 Q'
Q'.push(x)
$d_{Q'} := 0$　　%队列中所有序号的最短路径距离总和
$n_{Q'} := 1$　　%队列中序号个数
while Q' 不为空 do

$v := Q'$.front()
while $D_v \cdot n_{Q'} > d_{Q'}$ do
Q'.pop()
Q'.push(v)
$v \leftarrow Q'$.front()
end while
Q'.pop()
$n_{Q'} \leftarrow n_{Q'} - 1$
$d_{Q'} \leftarrow d_{Q'} - D_v$
$B_v \leftarrow$ true
foreach 元组 $e \in E_v$ do
$w := e[0]$
$d := e[1]$
if $D_w > D_v + d$ then
$D_w \leftarrow D_v + d$
$F_w \leftarrow v$
if B_w =false then
Q'.push(w)
$n_{Q'} \leftarrow n_{Q'} + 1$
$d_{Q'} \leftarrow d_{Q'} + D_w$
$B_w \leftarrow$ true
end if
end if
end for
end while

算法 8 适用于枝干点云的 SPFA 算法的 SLF+LLL 优化

Algorithm.8 The (combined) SLF/LLL algorithm for branch point cloud

初始化标记序列 B，其中 $B_i :=$ false，$i = 1, 2, \cdots, |P|$
$B_x :=$ true
初始化双端队列 Q''
Q''.push_back(x)
$d_{Q'} := 0$ %队列中所有序号的最短路径距离总和
$n_{Q'} := 1$ %队列中序号个数
while Q'' 不为空 do
$v := Q''$.front()
Q''.pop_front()
if $D_v \cdot n_{Q'} > d_{Q'}$ do
Q''.push(v)
continue
end if
$n_{Q'} \leftarrow n_{Q'} - 1$
$d_{Q'} \leftarrow d_{Q'} - D_v$
$B_v \leftarrow$ true

foreach 元组 $e \in E_v$ do
  $w := e[0]$
  $d := e[1]$
  if $D_w > D_v + d$ then
    $D_w \leftarrow D_v + d$
    $F_w \leftarrow v$
    if B_w =false then
      if Q'' 不为空 and $D_w < D_{Q''.\text{front}()}$ then
        Q''.push_front(w)
      else
        Q''.push_back(w)
      end if
      $n_{Q'} \leftarrow n_{Q'} + 1$
      $d_{Q'} \leftarrow d_{Q'} + D_w$
      $B_w \leftarrow$ true
    end if
  end if
end for
end while

3 实验验证

3.1 数据来源与预处理

本次实验使用的树木位于江苏省宿迁市泗洪县陈圩林场马浪湖分场（北纬 33°15’，东经 118°18’），树种均为美洲黑杨（*Populus deltoides Marshal*）中南林 797 杨，使用 RIEGL VZ-400i 地基激光雷达扫描仪对树木所在样地进行多站扫描（蒋昳萱等，2021），扫描模式设置为 Panorama40，详细参数如表 4 所示。

表 4 RIEGL VZ-400i 扫描模式 Panorama40

Table.4 RIEGL VZ-400i scanning mode panorama40

扫描分辨率	扫描时间	测量	分辨率@20m	分辨率@50m	扫描站数
40 mdeg	45 sec	22.5Mio	14 mm	34mm	400

将项目文件从设备导出，使用 RiSCAN Pro 2.7 多站拼接后导出为 las 文件格式，然后在 LiDAR360 软件中去噪、去除地面点、根据地面点归一化后导出 las 文件，导入到 Cloud Compare 中先切成单排点云后再进行单木分割并将每木点云导出成 pcd 文件格式得到了本次实验所用的树木点云，最后，在 Visual Studio 2019+PCL 1.11.0 内去除重复点，将单木点云进行适当变换使树干点云的根部切面与平面 *xOy* 尽可能重合，所有点均在平面 xOy 上或上方，并使根部中心尽可能靠近原点(0,0,0)处，以此作为实验数据，如图 3 所示。

图 3 实验数据

Fig.3 Experimental Data

3.2 实验环境

实验环境如表 5 所示。

表 5 实验环境

Table 5 Experimental environment

项目	配置
操作系统	Windows 11 专业版
处理器	11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz
内存	16.0GB

3.3 实验结果与分析

对实验所用的树木点云以参数 $R=0.05$ 枝叶分离得到枝干点云，此时的枝干点云点数 $|P|=198{,}500$，按照算法 1 以参数 $x=0$ 构建邻接表和初始化需要的参数。比较枝干点云获取最短路径长度进行优化的若干算法的实际运行时长。以 $r=R, r=2R, r=3R$ 并各自进行 10 次试验测试，结果如表 6 所示。

表 6 若干算法的执行时间（单位：秒）

Table.6 Execution time of algorithms (Unit: second(s))

		Dijkstra	堆优化的 Dijkstra	Bellman-Ford	SPFA	SPFA 的 SLF 优化	SPFA 的 LLL 优化	SPFA 的 SLF+LLL 优化
r=R=0.05 $\|E\|$=14,139,900 邻接表内存使用 290MB	1	34.953	0.047	0.093	0.046	0.047	0.031	0.032
	2	33.688	0.063	0.078	0.031	0.047	0.047	0.032
	3	33.828	0.062	0.11	0.031	0.031	0.031	0.031
	4	33.985	0.062	0.094	0.031	0.047	0.031	0.047
	5	33.734	0.047	0.094	0.047	0.031	0.031	0.031
	6	34.329	0.063	0.078	0.031	0.031	0.031	0.031
	7	33.969	0.063	0.109	0.047	0.046	0.031	0.032

	8	33.797	0.063	0.078	0.031	0.031	0.032
	9	34.109	0.063	0.094	0.047	0.031	0.031
	10	33.875	0.063	0.078	0.047	0.031	0.031
	Avg.	34.0267	0.0596	0.0906	0.0405	0.0326	0.033
	1	33.985	0.141	1.375	0.093	0.11	0.11
	2	33.953	0.14	1.359	0.094	0.109	0.093
	3	34.171	0.141	1.391	0.094	0.094	0.11
	4	33.828	0.141	1.375	0.11	0.125	0.125
	5	33.86	0.14	1.359	0.093	0.093	0.11
$r = 2R = 0.1$ $\|E\| = 57{,}079{,}350$ 邻接表内存使用 1.0GB	6	34.156	0.141	1.36	0.093	0.094	0.11
	7	34.063	0.141	1.375	0.125	0.109	0.109
	8	33.875	0.141	1.359	0.093	0.093	0.125
	9	33.89	0.156	1.375	0.094	0.094	0.11
	10	34.141	0.141	1.359	0.109	0.094	0.094
	Avg.	33.9922	0.1423	1.3687	0.0998	0.1015	0.1096
	1	34.406	0.25	2.047	0.188	0.187	0.187
	2	34.219	0.282	2.063	0.203	0.188	0.172
$r = 3R = 0.15$ $\|E\| = 128{,}273{,}662$ 邻接表内存使用 2.3GB	3	34.563	0.266	2.062	0.203	0.188	0.219
	4	34.422	0.25	2.062	0.188	0.187	0.203
	5	34.64	0.266	2.047	0.187	0.187	0.219
	6	34.359	0.265	2.062	0.172	0.203	0.218

7	34.422	0.25	2.079	0.188	0.203	0.187	0.219
8	34.547	0.25	2.063	0.219	0.203	0.218	0.188
9	34.344	0.265	2.078	0.203	0.204	0.187	0.188
10	35.078	0.281	2.062	0.203	0.172	0.187	0.203
Avg.	34.5	0.2625	2.0625	0.1985	0.1923	0.1919	0.2016

由表可知，采用邻接表所使用的内存远远小于邻接矩阵所需要的。同样是邻接表来表示图，传统使用的 Dijkstra 算法速度远远慢于其他所有算法，而 SPFA 及其优化相比之下更快。在求解枝干点云的最短路径问题上，SPFA 的三种优化并不如在某些图的求解上非常显著且稳定地比 SPFA 更快，因此，在枝干点云获取最短路径长度方面，选用适配后的 SPFA 算法是理想方案，对 SPFA 的三种改进则不一定需要。图 4 所示的是建立的最短路径的在根部附近的展示，为了便于展示，调整了枝干点云的观察角度。

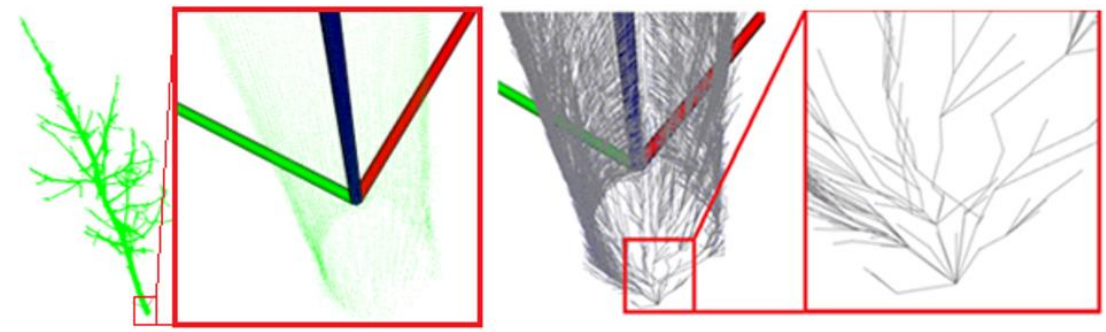

图 4 根部附近的最短路径图

Fig.4 Shortest path graph near the root

4 结论

本研究简要介绍了树木建模中利用点云以及基于几何特征的枝干点云的骨架提取的意义，对比了以根节点距离相似归类和以高度归类两种规则的优劣，前者在树干分叉处鲁棒性较好，但现有研究中大多采用的 Dijkstra 算法处理枝干点云的速度较慢，亦或采用了不合适的表达图的数据结构因而实际使用的较少。针对这一现象，本研究首先采用了邻接表的形式作为图的数据结构的表达，并将现有的若干求解最短路径的算法进行改进以应用于枝干点云，实验结果表明采用枝干点云的 SPFA 算法是理想方案，大大减少了执行时间。本研究能够为精细化点云树木建模提供一定帮助。

参考文献

张玥焜, 余文杰, 赵习之, 等. 基于机载激光雷达点云的交互式树木分割与建模方法研究[J]. 图学学报, 2021,42(04): 599-607.

Pyataev A. S. 3d tree modeling algorithm[J]. Siberian Journal of Science and Technology, 2018,4(19): 598-604.

曹伟, 陈动, 史玉峰, 等. 激光雷达点云树木建模研究进展与展望[J]. 武汉大学学报(信息科学版), 2021, 46(2): 203-220.

HE T, LIU Y F, SHEN C H, et al. Instance-aware embedding for point cloud instance segmentation[C]// ECCV, New York: Springer, 2020.

夏明鹏, 官凤英, 范少辉, 等. TLS 技术在森林资源调查中的应用现状与展望[J]. 西北林学院学报, 2018,33(03): 238-244.

石银涛. 基于点云的真实树木三维仿真理论与技术[M]. 上海: 同济大学出版社, 2018.

XU H, GOSSETT N, CHEN B. Knowledge and heuristic-based modeling of Laser-Scanned trees[J]. ACM Transactions on Graphics, 2007,26(4): 11-19.

FAN G, NAN L, DONG Y, et al. AdQSM: A new method for estimating Above-Ground Biomass from TLS point clouds[J]. Remote Sensing, 2020,12(18).

DONG Y, FAN G, ZHOU Z, et al. Low cost automatic reconstruction of tree structure by AdQSM with Terrestrial Close-Range Photogrammetry[J]. Forests, 2021,12(8).

王斌. 基于地基激光雷达点云的树木三维结构自动重建技术研究[D]. 电子科技大学, 2015.

ROBERT S K W. Algorithms Fourth Edition[M]. Hoboken: Addison-Wesley Professional, 2011.

CORMEN T H, LEISERSON C E, RIVEST R L, et al. Introduction to Algorithms Third Edition[M]. Cambridge: MIT Press, 2009.

KARBOWSKI A. Parallel algorithms for multicore routers in large computer networks – a review[J]. IFAC Proceedings, 2007,40(9): 298-303.

赵卫绩, 巩占宇, 王雯, 等. 几种经典的最短路径算法比较分析[J]. 赤峰学院学报(自然科学版), 2018,34(12): 47-49.

郑海虹. 常用最短路径算法分析与比较[J]. 安徽电子信息职业技术学院学报, 2013,12(04): 31-33.

蒋昳萱,温小荣,蒋佳文,等. 基于 2 期地基激光数据的杨树干形变化监测[J]. 西北林学院学报, 2021,36(5): 140-145.