doi: 10.7690/bgzdh.2013.06.024

一种改进的点云数据栅格算法

张春成
(海军装备部驻重庆地区军事代表局，重庆 404312)

摘要：点云数据的空间划分是点云数据拓扑关系建立的关键一步，而栅格算法是点云数据空间划分主要方法之一。简要分析传统算法的优缺点，在此基础上提出了一种不计算点云数据包围盒的实时栅格化点云数据的栅格算法。主要阐述算法的栅格划分、数据结构设计，并给出算法的流程，同时还通过实验验证了算法的正确性。改进算法简化了数据空间划分的步骤和程序，在效率上有一定提高，并且为实时点云数据重绘奠定了一定的基础。

关键词：点云数据；空间划分；栅格；数据结构；算法流程
中图分类号：TP301.6　　**文献标志码**：A

An Improved Grid Algorithm of Point Cloud Data

Zhang Chuncheng
(*Military Representative Bureau of Navy Equipment Department in Chongqing District*, *Chongqing* 404312, *China*)

Abstract: Space division of point cloud data is an important step in the reconstruction of the surface, and the grid algorithm is one of the main methods dividing the space. The paper provides a brief analysis of the advantages and disadvantages of the traditional grid algorithm, based on this provides a modified grid algorithm that doesn't calculate point cloud data bounding box, and that real-time grid point cloud data. Elaborate the grid division and data structure design and gave the algorithm steps process. At the same time, verify the correctness of the algorithm by experiments. The improved algorithm simplifies the steps and procedures for the point cloud data space division, improves some efficiency, and laid a foundation for real-time redrawing point cloud data.

Key words: point cloud data; space division; grid; data structure; algorithm flow

0　引言

逆向工程[1]在现代化战争和国防建设中意义重大，如数字地球、军事设备逆向分析和制造。数字地球广泛应用于战略、战术和战役的各种军事地理信息系统，并运用虚拟现实技术建立数字化战场。军事设备逆向分析和制造广泛应用于对别国先进装备的学习和仿制，以及自己装备的定量分析和改进。

在逆向工程中，探测得到的空间数据（大量的空间数据通常称为点云数据）量很庞大，对这些数据的表面重建，需要进行顶点的邻域搜索。点云数据[1-5]的空间划分能大大地提高邻域搜索的速度，是物体逆向还原中的重要一步。栅格算法是点云数据空间划分的主要方法之一。笔者提出了一种改进的栅格算法来对点云数据进行空间划分，它不计算点云数据的包围盒，对探测到的点云数据进行实时的栅格化处理。

1　栅格算法

栅格划分是一种常用且有效的划分策略。对于空间散乱点云数据，首先求出所有数据点坐标的最大值与最小值，再根据这些最大最小坐标值建立一个与坐标轴平行的最小长方体包围盒。根据给定的划分边长将长方体空间划分成一系列小立方体（称为栅格），把所有数据点归入到其对应的栅格中。

对于一个给定点云数据$\{P_i(i=1,2,\cdots,n)\}$，其数据点在坐标轴方向上的极值分别为 $x_{\max}$，$x_{\min}$，$y_{\max}$，$y_{\min}$，$z_{\max}$，$z_{\min}$。按照给定边长 L 划分为 m 个栅格，其中：

$$m=\left\lceil\frac{(x_{\max}-x_{\min})}{L}\right\rceil\bullet\left\lceil\frac{(y_{\max}-y_{\min})}{L}\right\rceil\bullet\left\lceil\frac{(z_{\max}-z_{\min})}{L}\right\rceil \quad (1)$$

对点云数据进行划分后，栅格信息是由二维数组存储的，每个数据点存入其对应栅格所对应的索引数组中。进行点的邻域搜索时，可先在点所在的栅格里搜索，如果失败，可在与本栅格相邻的栅格中搜索。这样，避免了对整个点云数据的搜索，大大减少了对非近邻点的计算，提高了搜索的效率。

但是该算法要计算点云数据的包围盒，要求将被探测对象所有的顶点数据扫描完了以后才能进行空间数据的划分，这不利于点云数据的快速栅格化，不能满足点云数据的实时重绘的要求。

收稿日期：2013-01-28；修回日期：2013-03-19
作者简介：张春成（1982—），男，重庆人，硕士，工程师，从事通信与信号处理、机电一体化研究。

2　改进的栅格算法

在点云数据的探测中，一旦探测设备确定，那么探测得到的点云数据的密度也就相对确定，而栅格算法中栅格中的顶点个数有理想值，所以用这个理想值除以点云数据密度就可以得出栅格边长。这样就省去了传统栅格算法中计算点云数据包围盒的过程。笔者就是基于这样的思想，直接对点云数据进行实时的栅格空间划分，同时不计算、不保存没有顶点的栅格。这样不光简化了传统的栅格算法，还为点云数据的实时重绘奠定了基础。

2.1　点云数据的栅格划分

待处理的点云数据保存在一个文本文件(original point data.txt)中，以数组格式保存，名称为 point，其格式为 point[…, x_i y_i z_i, …]n×3，i 为顶点{x_i y_i z_i}的索引，0≤i≤n，i∈Z(Z 是指整数)。例如：

point [0 24.41 0, 0 23.94 −4.762, −0.929 23.94 −4.67, −1.822 23.94 −4.399, −2.645 23.94 −3.959, …, x_i y_i z_i, …]

笔者提出的栅格划分过程是：依照顶点在 point 数组中的序号，逐点计算其所属栅格号，并保存数据，这样就把顶点划分到其所属的栅格中。

对 point 数组中序号为 i 的顶点 p_i=(x_i y_i z_i)，其栅格号计算公式为：

$$g=\begin{pmatrix} g_x & g_y & g_z \end{pmatrix}=\begin{cases} g_x=\lfloor (x_i-x_0)/L \rfloor \\ g_y=\lfloor (y_i-y_0)/L \rfloor \\ g_z=\lfloor (z_i-z_0)/L \rfloor \end{cases} \tag{2}$$

式中：浮点数据 L 为栅格边长；(x_0 y_0 z_0)是 point 数组的第零号顶点，作为栅格号计算的参考点。$\lfloor x \rfloor$的含义为小于等于 x 的最大整数，例如$\lfloor 0 \rfloor=0$，$\lfloor 0.1 \rfloor=0$，$\lfloor 1 \rfloor=1$。g_x表示栅格号在 x 轴上的分号；g_y表示栅格号在 y 轴上的分号；g_z表示栅格号在 z 轴上的分号。

栅格划分的第 0 号栅格如图 1 所示。图 1 中的坐标圆点为 p_0，根据式 (2) 计算出它的栅格号为(0 0 0)。图 1 中，立方体的 1 号顶点的栅格号为(1 0 0)，2 号顶点的栅格号为(1 1 0)，3 号顶点的栅格号为(0 1 0)，4 号顶点的栅格号为(0 0 1)，5 号顶点的栅格号为(1 0 1)，6 号顶点的栅格号为(1 1 1)，7 号顶点的栅格号为(0 1 1)。这个立方体表面内的点的栅格号都是(0 0 0)。

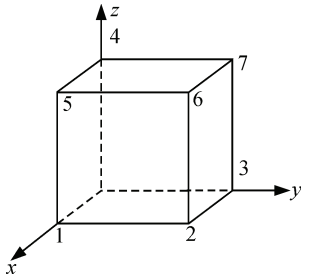

图 1　第 0 号栅格

2.2　算法的数据结构

栅格算法过程中涉及到的数据有：点云数据、栅格号数据、栅格顶点索引数据。栅格顶点索引指这个顶点在点云数据中的排号，这个排号是从 0 开始，从小到大顺序排列。栅格算法涉及到 3 个数据，所以就需要对应的设计 3 个数据结构，来分别保存它们。

2.2.1　点云数据结构的确定

根据点云数据的特征和算法需要，应满足以下条件：1) 点云数据中顶点的顺序是在超声探测时确定的，探测到某个顶点的时间在前，那么它的序号也在前。这就表示它们在点云数据结构中的前后顺序不会改变，同时还意味着在已经存在的 2 个顶点之间不会插入新的顶点。2) 本算法不删除点云数据结构中的顶点。3) 本算法需要经常快速地读取点云数据结构中的顶点坐标值。

根据上述特征和需要，确定用 C++中的 vector 模板类作为点云数据结构，便于快速访问点云数据。首先将文本中的点云数据推入 vector 中，然后才用访问数组元素的方法访问 vector 中的顶点元素。

2.2.2　栅格号数据结构的确定

栅格号数据是通过栅格号公式计算点云数据得到的。对于某一栅格号(g_x, g_y, g_z)在数据结构存储格式为“g_x g_y g_z”。如果现在有 2 个栅格号(g_{1x}, g_{1y}, g_{1z})和(g_{2x}, g_{2y}, g_{2z})，这就需要 2 个栅格号的大小比较方法和存储规则。

规定 2 个栅格号的大小比较方法，可分为 3 步：1) 进行 2 个 g_x 的大小比较，如果比较的结果是不相等，则比较结束，不再进行第 2、3 步，将这个结果作为 2 个栅格号的大小比较的结果；2) 进行 2 个 g_y 的大小比较，如果比较的结果不相等，则比较结束，不再进行第 3 步，将这个结果作为 2 个栅格号的大小比较的结果；3) 进行 2 个 g_z 的大小比较，

将比较结果作为 2 个栅格号的大小比较的结果，比较结束。

根据上述比较大小的结果来存储栅格号。存储规则为：小的栅格号存储在数据结构的前面，大的栅格号存储在数据结构的后面，如果 2 个栅格号相等，则它们是同一个栅格号。

在点云数据栅格化过程中可能出现这样的情况：对于某一顶点的栅格号 g_i，假如它是第一次出现，它比存在于栅格号结构中的某一栅格号 g_j 小，按照栅格号存储规则，要把 g_i 存储在 g_j 前面，也就是要把 g_i 插入到 g_j 前面。针对这个情况，在 C++中，该选择那一种数据结构呢？

list 模板类和 vector 模板类都支持数据的插入和删除。它们之间的关键区别在于，list 在链表中任一位置进行插入和删除的时间都是固定的；vector 模板提供了除结尾处外的线性时间的插入和删除，在结尾处，它提供了固定时间的插入和删除。保存栅格号数据时，需要经常做插入保存，并且插入的位置不定，所以选择 list 模板类作为栅格号数据结构，这将能减少栅格号数据的保存时间。

2.2.3 栅格化顶点索引数据结构的确定

设计栅格顶点索引数据结构的特征：在栅格顶点索引数据结构中，每个栅格的最后一个元素是“-1”，作为栅格标识符，用来区分不同的栅格。当顶点的栅格号存储在栅格号数组结构中后，就要存储顶点的序号于栅格顶点索引数据结构中。在栅格顶点索引数据结构中，当一个顶点的栅格已经存在时，只要栅格顶点索引值插入到这个栅格的标识符之前；当这个顶点的栅格还没有创建时，就要在栅格顶点索引数据结构中找出唯一的元素，将这个新的栅格插入到它的前面。

通过上述特征，把选择 list 模板类作为栅格化顶点索引数据结构，便于插入元素。现举例说明栅格号数据结构和栅格化顶点索引数据结构的关系：

栅格号数据结构：[-1 -2 -1 -1 -2 0 …]

栅格顶点索引数据结构：[258 259 260 261 … 397 398 399 400 425 426 427 428 429 430 431 432 457 458 459 460 461 462 463 464 -1 …]

栅格号数据结构和栅格化顶点索引数据结构中，相同序号的栅格是同一栅格，只是它们存储栅格的不同数据，栅格号数据结构存储的是顶点的栅格号，栅格化顶点索引数据结构存储的是顶点的索引值。利用这个关系可以方便地找到顶点索引值要插入的位置。

2.3 算法中用到的有关 C++知识

1) 创建 vector 变量。

在程序中要使用 vector 模板类，必须在程序引用头文件的位置包含 vector 模板，语句为“#include <vector>”。

要创建 vector 模板对象，可使用通常的<type>表示法来指出使用的类型。另外，vector 模板使用动态内存分配。

2) 创建 list 变量。

在程序中要使用 list 模板类，必须在程序引用头文件的位置包含 list 模板，语句为“#include <list>”。

要创建 vector 模板对象，可使用通常的<type>表示法来指出使用的类型。

3) 对 vector、list 容器中数据的操作方法简介。

vector 与 list 容器都具有下面相同的数据操作方法。size()——返回容器中元素数目；swap()——交换 2 个容器的内容；begin()——返回一个指向容器中第一个元素的迭代器；end()——返回一个表示超过容器尾的迭代器(指向容器最后一个元素后面的那个元素)。

push_back()将元素添加到矢量末尾。它负责内存管理，增加矢量的长度。

erase()方法删除矢量中给点区间的元素。它接收 2 个迭代器参数，这些参数定义了要删除的区间。第 1 个迭代器指向区间的起始处，第 2 个迭代器位于区间终止处的后一个位置。

insert()方法的功能与 erase()相反。它接收 3 个迭代器参数；第一个参数指定了新元素的插入位置，第 2 个和第 3 个迭代器参数定义了被插入区间，该区间通常是另一个容器对象的一部分。

list 的用法与 vector 相似。与 vector 不同的是，list 不支持数组表示法和随机访问；从容器中插入或删除元素之后，链表迭代器指向元素将不变。

2.4 算法流程

本栅格算法是用 C++编程来实现的，用到的数据结构有：数组、vector 模板类、链表和迭代器等。程序中包含的头文件有：<iostream>、<fstream>、<cstdlib>、<vector>、<list>和<iterator>。算法的大致流程是，将 original point data.txt 中的点云数据读取到一个一维的 vector 中，然后逐点计算栅格号，

并保存栅格号于栅格号链表中，保存顶点的索引值于栅格顶点索引链表中。栅格算法的流程如图 2。

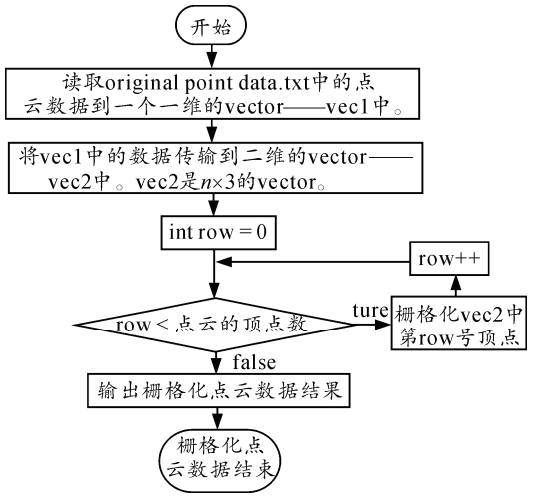

图 2　栅格算法流程

栅格化 vec2 中第 row 号顶点的流程如图 3。

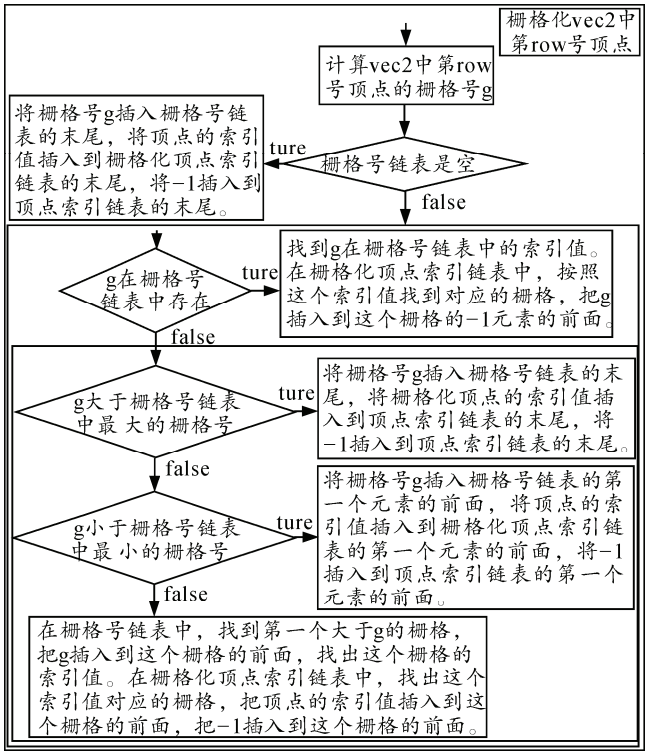

图 3　栅格化 vec2 中第 row 号顶点

3　实验数据

point 数据是点云数据，是程序的输入数据。栅格号链表——gridset，栅格化顶点索引链表——coordig，这 2 个链表里面的数据是程序的输出数据。当栅格边长 L=25 时，本栅格算法的实验结果为：

number of points is 964.　　//顶点总个数

point:

0, 24.41, 0,　0, 23.94, −4.762,　−0.929, 23.94, −4.67, … 103.959, 76.06, 97.355,　103.367, 76.06, 96.633,　102.645, 76.06, 96.041,　101.822, 76.06, 95.601,　100.929, 76.06, 95.33,　100, 75.59, 100,

gridset.size()/3 = 18　　//栅格号总个数

gridset:

−1 −2 −1　−1 −2 0　−1 −1 −1　−1 −1 0　0 −2 −1
0 −2 0　0 −1 −1　0 −1 0　0 0 0　3 2 3　3 2 4
3 3 3　3 3 4　4 2 3　4 2 4　4 3 3　4 3 4　4 4 4

coordig:

258 259 260 261 262 263 264 290 291 292 293 294 295 296 322 323 324 325 326 327 328 354 355 356 357 358 359 360 386 387 388 389 390 391 392 418 419 420 421 422 423 424 450 451 452 453 454 455 456 −1　… 482 −1

其中，point 里的数据是点云数据，存储的是每个顶点的三维坐标值(x,y,z)。gridset 里的数据是点云数据栅格化后所有的栅格号(g_x,g_y,g_z)。coordig 里的数据是栅格化后的顶点的索引值，一个“−1”就对应着 gridset 里的一个栅格号。

4　结论

笔者详细介绍了栅格算法的原理、数据结构和处理流程[6]，并通过运行 C++程序验证了算法的正确性。该栅格算法不计算点云数据的包围盒，所以节省了相应程序的运行时间和运行内存；同时，该算法是逐点栅格化点云数据，为实时的点云数据重绘奠定了基础。

参考文献：

[1] 赵伟玲. 三维点云的数据预处理和圆提取算法研究[D/OL]. 万方数据, [20100-08-14]. http://d.g.wanfangdata.com.cn/ Thesis_Y1436701.aspx.

[2] 刘立强. 散乱点云数据处理相关算法的研究[D/OL]. 万方数据, [20100-10-8]. http://d.g.wanfangdata.com.cn/Thesis_Y1678646.aspx.

[3] 张春成. 超声探伤点云数据拓扑关系的建立[J]. 兵工自动化, 2011, 30(4): 64−66.

[4] 石海潜, 陈荦, 刘义. 一种网络空间数据集成访问方法[J]. 兵工自动化, 2011, 30(10): 51−54.

[5] 张西山, 闫鹏程, 孙江生, 等. 维修保障信息系统的数据处理方法[J]. 兵工自动化, 2011, 30(7): 93−96.

[6] 宋春霞, 张磊, 关锦生. 一种新的多模型联合概率数据关联方法[J]. 四川兵工学报, 2010, 31(5): 123.