一种基于边指针搜索及区域划分的三角剖分算法

张 俊 [1] 田慧敏 [2]

摘 要 针对大规模数据处理时 Delaunay 三角剖分过于耗时的问题, 本文提出了一种基于边指针搜索及区域划分的三角剖分算法. 基于边指针设计了一种能够反映三角形之间位置关系的数据结构, 并优化了目标三角形的搜索路径. 基于该数据结构, 利用区域划分进一步降低目标三角形的搜索深度. 超级三角形所在的正方形被划分成具有相同尺寸的区域, 目标三角形的搜索从插入点所在的区域的入口三角形开始, 这大大缩小了目标三角形的搜索范围. 实验证明, 与传统的 Delaunay 三角剖分算法相比, 该算法的效率显著提升.

关键词 Delaunay 三角剖分, 三维重建, 边指针, 区域划分

引用格式 张俊, 田慧敏. 一种基于边指针搜索及区域划分的三角剖分算法. 自动化学报, 2021, **47**(1): 100−107

DOI 10.16383/j.aas.c190155

A Triangulation Algorithm Based on Edge-pointer Search and Region-division

ZHANG Jun[1] TIAN Hui-Min[2]

Abstract To solve the problem of taking too much time when dealing with large-scale data, a triangulation algorithm based on edge-pointer search and region-division is proposed. A data structure based on the edge-pointer is designed to reflect the positional relationship between triangles, and the search path of the target triangle is also optimized. Moreover, region-division is utilized to reduce the search depth. The square which contains the super triangle is divided into some regions of the equal size. The search for the target triangle begins with the entry triangle of the region where the insertion point is located. As a result, the search scope of the target triangle is narrowed. Experiment results show that the algorithm is much more efficient than the traditional Delaunay triangulation algorithm.

Key words Delaunay triangulation, 3D reconstruction, edge-pointer, region-division

Citation Zhang Jun, Tian Hui-Min. A triangulation algorithm based on edge-pointer search and region-division. *Acta Automatica Sinica*, 2021, **47**(1): 100−107

三维重建技术是根据物体的二维图像信息恢复其三维模型, 多年来一直是计算机图形学领域的研究热点[1−2]. 三角剖分是三维重建过程中最重要的基础之一, 通过将散点连接形成许多互不相交的三角形或四面体, 从而使模型面化或者体化[3]. 经过多年探索, 研究者们对于二维三角剖分的研究已经取得了很多成果, 提出了多种三角网格的优化准则和构造方法. 其中, Delaunay 三角剖分由于具有良好的几何特性 – 可以最大化最小角[4], 即能使得到的三角网格最平均, 使用最广泛[5]. 然而, 随着计算机图形学技术的发展, 扫描设备的精度越来越高, 用于三维重建的点集对象的规模也越来越大. 因此, 在保证三角网格的整体质量的提前下, 有必要寻找一种更高效的 Delaunay 三角剖分, 从而满足三维重建的实时性.

Delaunay 三角剖分算法有很多种类, 主要分为三大类: 逐点插入算法[6]、分治算法[7]、三角网生长算法. 其中, 逐点插入法由于实现相对简单且占用空间相对较小[8], 被大量研究者所采用; 分治算法在执行时间方面可以达到最优, 但占用内存较大, 不适用于一般的计算机平台; 三角网生长算法相对于前两种算法效率较低, 不适用于大规模点集, 在实际应用中很少被采用. 逐点插入法的主要流程如下: 1) 构造一个包含所有插入点的超级三角形; 2) 将存储在链表中的点按序插入, 遍历三角形链表找出包含插入点的三角形 (即目标三角形), 利用插入点将目标三角形拆分生成新的三角形, 在三角形链表中删除目标三角形, 完成一个点的插入; 3) 利用空圆特性对新生成的三角形进行 Delaunay 规则判断; 4) 循环执行 2) 和 3), 直至完成对所有插入点的处理. 经典的逐点插入法有 Bowyer 算法[9]、Watson 算法[10] 以及 Lawson 算法[11] 等.

虽然逐点插入法易于实现, 但在处理大规模的

收稿日期 2019-03-13 录用日期 2019-05-23
Manuscript received March 13, 2019; accepted May 23, 2019
国家自然科学基金 (61571466) 资助
Supported by National Natural Science Foundation of China (61571466)
本文责任编委 刘艳军
Recommended by Associate Editor LIU Yan-Jun
1. 中南大学自动化学院 长沙 410083 2. 中南大学计算机学院 长沙 410083
1. School of Automation, Central South University, Changsha 410083 2. School of Computer Science and Engineering, Central South University, Changsha 410083

散点集数据时, 实时性不是很好, 为了提高逐点插入法的效率, 研究者们进行了大量的研究. 一部分研究者认为可以对插入点进行预处理, 通过改进插入点的序列来提高三角剖分的效率. Liu 等[12] 提出了一种基于广度优先搜索的确定性插入序列, 使用 k-d 树来构造 Delaunay 三角关系, 但是构造 k-d 较为复杂, 需耗费大量时间. 在多重网格插入法[13] 的基础上, Su 等[14] 提出了使用 Hilbert 曲线遍历插入点, 减少了三角剖分过程中狭长三角形出现的数量, 从而降低局部优化时间. Zalik 等[15] 提出一种两级均匀细分加速技术, 将目标三角形问题转化为最近点问题, 在他们的方案中, 从最近的 Delaunay 点开始, 然后从第二个最近的 Delaunay 点开始, 以这种递归的方式直到找到目标三角形. 在此基础上, Zadravec 等[16] 提出结合哈希表与跳跃表来寻找插入点的最近点, 从而快速定位目标三角形. 一些研究人员提出基于重心方向来定位目标三角形, 但是, 当出现分界点时, 这种方法的搜索路径可能不唯一. 针对此问题, Xi 等[17] 提出可以通过移动重心来避免截点, 从而使目标三角形可以连续搜索. 随着计算机技术的快速发展, 许多研究者们提出了并行 Delaunay 三角剖分, 将点集划分成许多独立的分区, 这些分区可以同时进行 Delaunay 三角剖分, Kohout 等[18]、Rong 等[19]、Cuong 等[20]、Lo[21] 均对此进行了研究及改进. 此外, 杨昊禹等[22] 提出了并行动态 Delaunay 三角剖分算法, 可以解决新增点位于原来三角网之外的情况. 另外李国庆等[23] 提出了基于凸多边形的 Delaunay 三角剖分算法, 使用生成的凸多边形代替传统算法中的超级三角形. 常见的生成凸包算法有 Graham 扫描法[24]、分区算法[25] 等. 刘斌等[26] 提出将主成分分析法 (PCA) 与二分法结合, 通过快速确定凸包边缘点来计算平面点集的凸包.

通过对 Delaunay 三角剖分逐点插入法的研究, 可知在构建三角网过程中最耗时的部分为寻找插入点的目标三角形. 本文在传统算法的基础上, 将边指针与区域划分相结合, 很大程度上提高了构建 Delaunay 三角网的效率.

1　基于边指针的搜索

在传统的逐点插入法中, 对每个插入点的目标三角形的搜索总是从三角形链表的表头开始. 这意味着, 目标三角形若位于链表的表头, 则可以很快被找到; 若位于链表的尾部, 则需要遍历整个链表才能被找到. 随着点的数量大幅度增加, 三角形链表的长度也必然会大幅度增加, 从而导致寻找目标三角形会越来越耗时, 降低三角剖分的效率. 针对此问题, 本文对三角形的数据结构进行优化, 提出了基于边指针的数据结构. 如图 1 所示, 在 $\triangle ABC$ 中, A 为起始顶点且 A、B、C 逆时针分布, AC 为边 1, AB 为边 2, BC 为边 3. 在三条边上分别存在边指针 $P1$、$P2$、$P3$, 分别指向边指针所在的公共边的相邻三角形, 如 $P1$ 指向与边 AC 相邻的三角形, 对于 $P2$、$P3$ 情况类似.

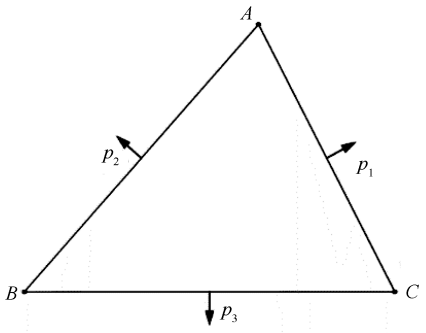


图 1　三角形数据结构

Fig. 1　Data structure of triangle

1.1　点与三角形的位置判断

点与三角形之间的位置关系可以根据其坐标的相对关系来判断. 对于任意三个点 $A(x_A, y_A)$, $B(x_B, y_B)$, $C(x_C, y_C)$, 其中

$$Flag(A,B,C) = (x_B - x_A)(y_c - y_B) - (x_c - x_B)(y_B - y_A)^{[27]} \quad (1)$$

若 $Flag(A,B,C) > 0$, 则 A、B、C 按逆时针方向分布; 若 $Flag(A,B,C) < 0$, 则 A、B、C 按顺时针方向分布; 若 $Flag(A,B,C) = 0$, 则 A、B、C 三点共线. 如图 2 (a) 所示, 若 $Flag(C,A,M)$、$Flag(A,B,M)$、$Flag(B,C,M)$ 三者的值都大于零, 则点 M 位于 $\triangle ABC$ 的内部. 若三者中一个值为零且其余二者皆为正值, 则位于某条边上, 如图 2 (b) 中 $Flag(A,B,M)$ 为零, $Flag(C,A,M)$、$Flag(B,C,M)$ 为正值, 故点 M 位于边 AB 上. 若三者中至少有一个为负值, 则点位于三角形的外部. 如图 2 (c) 所示, $Flag(A,B,M)$ 为负值, 故点 M 位于边 AC 的外侧. 当然, 点位于三角形外侧还会出现图 2 (d) 与图 2 (e) 的情况. 在图 2 (d) 中, $Flag(A,B,M)$、$Flag(B,C,M)$ 皆为负值, 在此情况下, 由于 AB 为边 2, 而 BC 为边 3, 本文中优先取编号在前的边, 故认为 M 位于边 AB 的外侧. 在图 2 (e) 中, 点 M 虽位于三角形外部且与边 AC 共线, 但是 $Flag(B,C,M)$ 为负值, 故本文认为点 M 位于边 BC 的外侧.

1.2　基于边指针搜索的路径确定

在搜索目标三角形的过程中, 搜索路径通过判断插入点与当前被搜索三角形的位置关系来确定的.

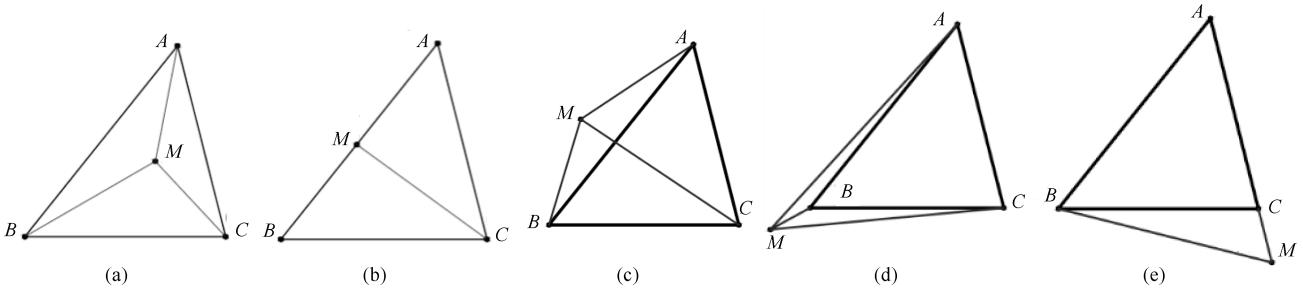


图 2 点与三角形位置关系判断

Fig. 2 Judgment of positional relationship between point and triangle

如图 3 所示, 点 V 为插入点, 假设 $\triangle AIJ$ 存储在三角形链表的表头, 搜索过程如下所述. 1) 检测点 V 与 $\triangle AIJ$ 的位置关系, 首先确定了点 V 位于 $\triangle AIJ$ 的外部, 然后确定点 V 位于 $\triangle AIJ$ 的哪条边的外侧, 根据式 (1) 可知点 V 位于边 IJ 外侧, 故下一个被搜索的三角形为 $\triangle AIJ$ 的边 IJ 上的边指针 p_1 所指的 $\triangle IEJ$. 2) 用同样的方法确定接下来的被搜索三角形分别是 $\triangle EFJ$、$\triangle EHF$, 然后根据式 (1) 可知点 V 位于 $\triangle EHF$ 的内部, 故 $\triangle EHF$ 为点 V 的目标三角形. 在整个搜索过程中, 共经过 4 次搜索找到点 V 的目标三角形, 搜索路径为: $\triangle AIJ(p_1) \rightarrow \triangle IEJ(p_2) \rightarrow \triangle EFJ(p_3) \rightarrow \triangle EHF$. 边指针搜索的方式, 由于在搜索目标三角形的过程中通过边指针的方向性可以快速地定位目标三角形, 使搜索路径大大缩短, 从而大大降低了三角剖分的耗时.

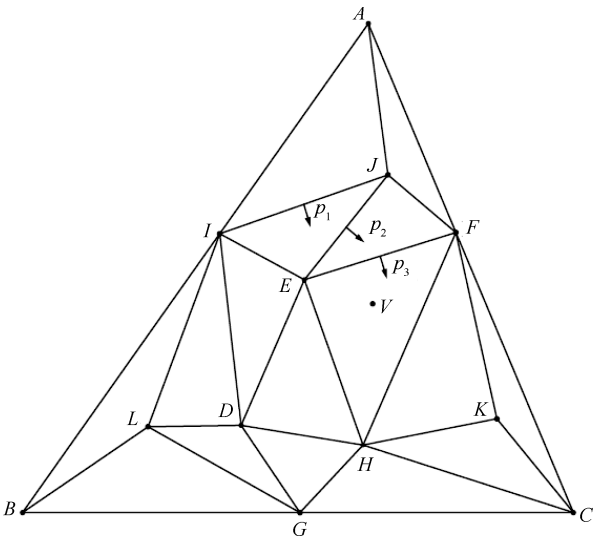


图 3 点 V 的目标三角形的搜索路径

Fig. 3 Search path for target triangle of point V

2 基于边指针搜索及区域划分的三角剖分

2.1 区域分块

基于边指针搜索目标三角形的方法, 虽然可以利用插入点与三角形的位置关系来快速定位目标三角形, 但是在点数量很大的情况下, 目标三角形的搜索路径可能仍然会比较长, 此时依旧较为耗时. 因此, 在边指针搜索的基础上, 提出了区域划分的方案, 以缩小目标三角形的搜索范围.

设区域分块为 $N \times N$, 构造超级三角形的时候, 首先遍历所有散点找到最大的纵横坐标值 $X_{\max}$、$Y_{\max}$, 取两者中较大者得到 max, 然后根据 max 构造超级三角形, 其各点坐标为:

$$\begin{cases} V_1 = (0, 4 \times max) \\ V_2 = (-4 \times max, -4 \times max) \\ V_3 = (4 \times max, 0) \end{cases} \tag{2}$$

区域划分是指将包含超级三角形的正方形划分成 $N \times N$ 个大小相同的区域, 并将这些区域存储在二维数组中. 每个区域的位置坐标值 X 和 Y 对应于其在二维数组中的存储位置, 如位置坐标 (1, 2) 的区域块存储在数组 area[1][2] 中. 对于插入点 $P(x, y)$, 它的区域坐标由其本身的横纵坐标值确定:

$$\begin{cases} X = floor\left(\dfrac{x}{L \times N}\right) \\ Y = floor\left(\dfrac{y}{L \times N}\right) \end{cases} \tag{3}$$

在式 (3) 中, L 指包含超级三角形的正方形的边长.

区域可以划分成不同的大小, 对应的区域数量也会不同. 对于某个确定的点集, 如果区域太大, 搜索范围会变大. 但是, 如果区域太小, 则过多的区域很难被分配到入口三角形, 从而导致入口三角形的使用率下降, 进一步导致搜索目标三角形的时间增加. 所以对于处理不同规模的点集, 最优的分块也会不同.

对于点集 $V = \{V_1, V_2, \cdots, V_n\}$, 两个点组成一条边, 三条边组成一个三角形, 三角形通过公共边上的边指针相互连接组成三角形网格. 基于边指针搜

索及区域划分的 Delaunay 三角剖分的数据结构如表 1 所示.

表 1　基于边指针及区域划分的算法数据结构

Table 1　Data structure of algorithm based on edge-pointer and region-division

Structure				
POINT	x	y	entry_flag	
EDGE	V_1	V_2		
TRIANGLE	V_1, V_2, V_3	$*p_1, *p_2, *p_3$	entry	X, Y
REGION	valid	*entrytri		

数据结构包括 4 部分: 点、边、三角形和区域. 每个顶点包含 x 和 y 坐标以及区域入口标志 $entry\_flag$; 连接顶点的边包含顶点 V_i 和 V_j 的编号; 三角形采用空间网状结构, 包含顶点 V_i、V_j、V_k 的编号, 三角形的区域入口标志 $entry$ 以及该区域的坐标 X、Y, 将位置相邻的三角形连接起来的边指针 p_1、p_2、p_3, 区域坐标只有在 $entry = 1$ 时才意义, 如果 $entry = 1$, 意味着三角形是该区域的入口三角形; 区域块包含区域是否分配入口三角形的标志 valid, 以及区域的入口三角形指针 $entry\_tri$. 对于一个入口三角形, 在其三个顶点中存在一个入口顶点, 其 entry_flag 值被置为 1, 并且入口顶点位于于该三角形所记录的区域中. 边指针搜索及区域划分数据结构中不同区域之间的关系如图 4 所示, 不同局域会分配一个入口三角形作为该区域的搜索入口, 该入口三角形再通过边指针指向存在该区域内或其它区域内的相邻三角形.

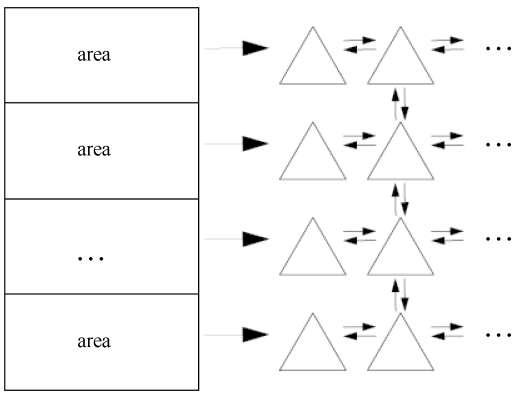


图 4　结构体关系图

Fig. 4　Structure diagram

2.2　搜索目标三角形

实现 Delaunay 三角剖分最重要也最耗时的部分就是搜索目标三角形, 本文对目标三角形的搜索过程如算法 1 所示. 在基于边指针搜索的基础上, 区域划分的搜索方式为每个区域分配了入口三角形. 这使得对散点的目标三角形的搜索不是从三角形链表的表头开始, 而是从入口三角形开始, 故搜索的起始三角形更接近目标三角形, 目标三角形的搜索路径进一步被简化.

算法 1. 搜索插入点 V_i 的目标三角形

S1: 确定 V_i 所在区域;

S2: **if** 该区域已经分配入口三角形, then

S3:　　从入口三角形开始搜索;

　　else

S4:　　从最新生成的三角形开始搜索;

S5: **if** V_i 位于当前搜索三角形内部, then

S6:　　找到并返回 V_i 的目标三角形;

S7: **else if** V_i 位于当前三角形的边上, then

S8:　　根据边指针搜索共边三角形并返回这两个三角形;

　　else

S9:　　根据边指针确定下一个要搜索的三角形, 转到 S5.

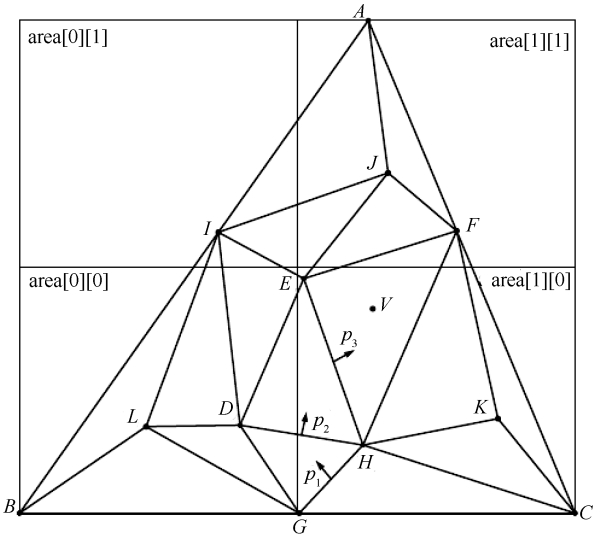
图 5 展示了基于边指针搜索及区域划分算法中搜索目标三角形的过程. 如图 5 (a) 所示, 包含超级三角形的正方形被划分成 2×2 个区域. 插入点 V 时, 首先根据点 V 的坐标确定其所在区域为 area[1][0]. 由于在点 V 之前, 点 H 与点 K 已经被插入该区域, 故该区域一定存在入口三角形, 假设 $\triangle GCH$ 为入口三角形, 对点 V 的目标三角形的搜索从 $\triangle GCH$ 开始. 接下来, 按照第 1.2 节中所述边指针搜索的方式, 确定目标三角形的搜索路径为: $\triangle GCH(p_1) \rightarrow \triangle DGH(p_2) \rightarrow \triangle EDH(p_3) \rightarrow \triangle EHF$, 目标三角形为 $\triangle EHF$.

如果检测到点 V 位于三角形的某条边上, 则需要根据边指针搜索另一个包含该点的一个三角形. 如图 5 (b) 所示, 点 V 位于边 HF 上, 当找到 $\triangle EHF$ 之后, 根据边 HF 上的边指针 p_4 找到共边三角形 $\triangle FHK$. 故 $\triangle EHF$ 与 $\triangle FHK$ 为点 V 的目标三角形, 并且点 V 将会被插入这两个三角形所构成的四边形内.

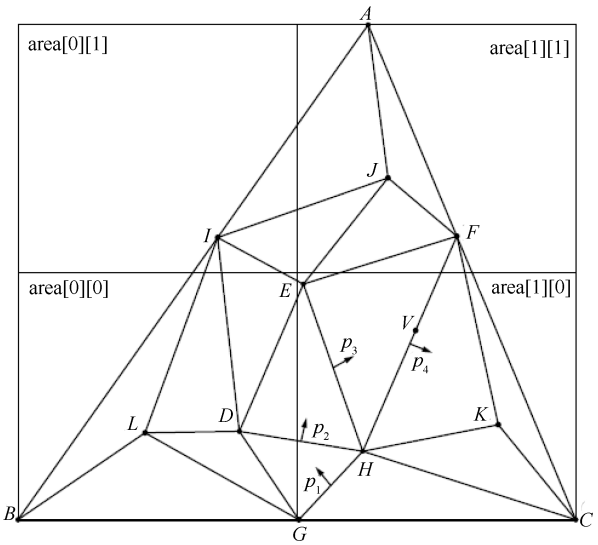
如果在点 V 所在区域中没有入口三角形, 则从最新插入的点生成的三角形开始搜索目标三角形, 搜索路径由边指针确定. 由于该算法主要用于三维重建, 而三维重建中扫描得到的连续点在位置上可能是相邻的, 因此, 由最新插入点生成的三角形可能更接近当前插入点的目标三角形.

在传统算法中, 假设当前所有三角形按一下顺序存储在三角形链表中: $\triangle AIJ$、$\triangle AJF$、$\triangle IEJ$、$\triangle EFJ$、$\triangle IBL$、$\triangle ILD$、$\triangle IDE$、$\triangle EDH$、$\triangle EHF$、$\triangle FHK$、$\triangle CFK$、$\triangle BGL$、$\triangle DLG$、$\triangle DGH$、$\triangle GCH$. 插入散点 V 时, 则需要依

次判断链表中在 $\triangle EHF$ 之前的三角形是否包含点 V, 可以发现一共需要经过 9 次搜索才能找到目标三角形, 而本文的算法只需要经过 4 次搜索就能找到目标三角形, 搜索深度为传统算法的 44.4 %. 随着点集的规模不断变大, 传统算法的执行时间将会呈指数增长, 本文算法的优越性会越来越明显.



(a) $\triangle EHF$ 内部
(a) Inside $\triangle EHF$



(b) 边 HF 上
(b) On edge HF

图 5 搜索点 V 的目标三角形

Fig. 5 Searching for target triangle of point V

目标三角形的定位是根据插入点与当前搜索三角形的位置关系来确定的, 通过判断插入点位于三角形的哪条边的外侧来确定下一个被搜索的三角形. 每一次被搜索的三角形都比上一次的三角形更接近插入点, 因此通过这种方式一定能找到目标三角形, 故该算法是收敛的.

2.3 区域入口三角形及边指针的更新

随着点地不断插入, 区域的入口三角形也需要不断更新以确保其不会随着某些三角形的拆分而消失. 同时, 与新三角形有关的边指针也需要被更新以确保三角形之间能够互相找到相邻的三角形. 找到目标三角形之后, 需要将点插入到目标三角形中, 并使用空圆特性对新生成的三角形进行 Delaunay 规则判断及调整违背规则的三角形. 在此过程中, 需要更新入口三角形及边指针, 入口三角形的更新分为本区域的更新以及他区域的更新.

如图 6 所示, 找到目标三角形 $\triangle EHF$ 之后, 插入点 V, 生成了三个新的三角形 ($\triangle EHV$, $\triangle HFV$ 与 $\triangle FEV$), 并将原目标三角形 $\triangle EHF$ 从三角形链表中删掉. 首先, 对本区域的入口三角形进行更新. 如果在此之前, 区域 area[1][0] 没有分配到入口三角形, 则选择 $\triangle EHV$ 为该区域的入口; 如果 area[1][0] 之前已经分配了入口三角形如第 2.2 节中所提到的 $\triangle GCH$, 则 $\triangle EHV$ 代替 $\triangle GCH$ 作为该区域的新入口, 并将点 V 指定为入口顶点. 然后, 根据被删除三角形的标志位及区域坐标信息检测其是否为其他区域的入口, 若是, 则需要指定一个新的三角形来更新此区域的入口. 例如, 如果三角形 $\triangle EHF$ 是区域 area[1][1] 的入口, 判断 $\triangle EHF$ 的哪个顶点是入口顶点, 根据点的入口标志位得出点 F 是入口顶点, 因此, 包含点 F 的 $\triangle FEV$ 成为了 area[1][1] 的新入口.

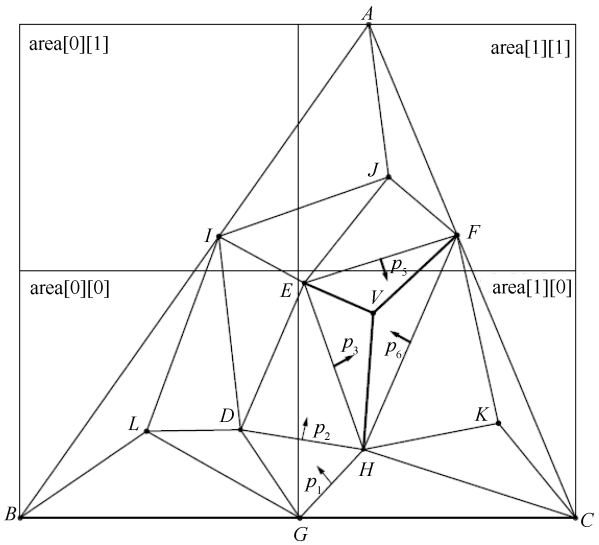


图 6 入口三角形及边指针的更新

Fig. 6 Update of entry triangle and edge-pointer

此外, 还需要更新和原三角形 $\triangle EHF$ 有关的边指针. 在图 6 中, 随着 $\triangle EHF$ 的删除, 原本指向该三角形的边指针应该指向新的三角形, $\triangle EDH$ 的边 EH 上的边指针 p_3 现在应该指向 $\triangle EHV$, 同

理, $\triangle EFJ$ 的边 EF 上的边指针 p_5、$\triangle FHK$ 的边 FH 上的边指针 p_6 现在分别指向其相邻三角形 $\triangle FEV$、$\triangle HFV$.

在进行 Delaunay 规则判断的过程中, 区域入口及边指针会以同样的方式更新, 此处不再赘述.

3　实验结果及分析

为了验证基于边指针搜索及区域划分的三角剖分算法的效率, 本文对随机生成的不同规模大小的点集进行三角剖分, 并比较了传统的 Delaunay 三角剖分 (TD)、使用 CGAL 库的三角剖分、基于边指针搜索的三角剖分 (EPD)、基于边指针搜索和区域划分相结合的三角剖分 (EPRDD) 几种算法之间的执行时间, 此处的执行时间不包括读取插入点数据的时间.

在进行区域划分的时候, 包含超级三角形的正方形被划分成了 15×15 个区域. 本实验的实验环境为 Intel(R) Core(TM) CPU i5-8500 @ 3.00 GHz, 16 GB 内存, Windows 10, VS2013, 64 位操作系统, 得到的结果如表 2 所示:

表 2　算法的执行时间 (s)

Table 2　Running time of algorithms (s)

散点数量	2 万	3.5 万	5 万	7 万	10 万
TD	113.515	298.687	756.656	1 943.39	3 972.81
CGAL	1.719	3.074	4.281	6.265	8.594
EPD	0.563	1.547	2.562	3.218	6.172
EPRDD	0.117	0.234	0.325	0.531	0.813

从表 2 可以看出, 当插入点的数量为 2 万时, 基于边指针搜索的三角剖分耗时为传统算法的 1/201, 加入区域划分之后, 算法的执行时间为传统算法 1/970. 当插入点的数量为 10 万时, 基于边指针的搜索算法执行时间分别为传统算法的 1/643, 而基于边指针及区域划分算法的执行时间为传统算法的 1/4 886. 此外, 与 CGAL 算法相比, 只使用边指针的算法优势并不是很明显; 但是, 边指针与区域划分相结合之后, 算法的耗时明显低于 CGAL 算法. 当对 10 万个点进行三角剖分的时候, CGAL 算法需要 8.594 s, 而基于边指针及区域划分的算法仅仅需要 0.813 s 就能完成.

为了比较几种算法的执行时间与散点数量的关系, 绘制了图 7 所示折线图. 由于传统算法的执行时间远远高于其余几种算法, 故不与其他算法作比较. 从图 7 可以看出, 随着散点数量的增多, CGAL 算法, 基于边指针搜索的算法与基于边指针搜索及区域划分算法的执行时间基本上都是呈线性增长的, CGAL 算法与基于边指针搜索的算法的执行时间随散点数量变化的增长速度还是较快, 但是基于边指针及区域划分算法的执行时间的增长速度较为缓慢, 斜率远远小于前两种算法. 可以得知, 随着插入点数量的增长, 边指针搜索与区域划分结合的方法在时间上的优势将会越来越大.

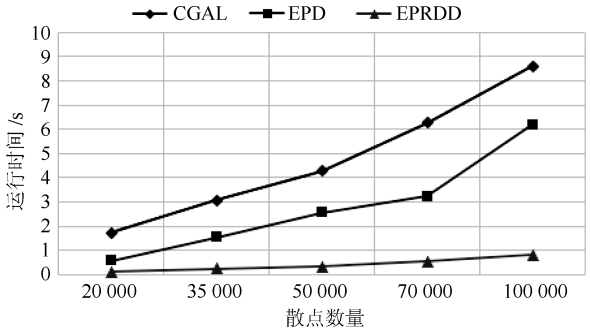


图 7　执行时间比较

Fig. 7　Comparison of run time

进一步对几种算法的平均搜索深度进行比较, 结果如表 3 所示. 搜索深度是指在将散点插入到 Delaunay 网格中的过程中搜索三角形的次数, 搜索深度越小, 三角剖分的实时性越好. 由于 CGAL 的源代码没有找到, 在此处未对 CGAL 的搜索深度进行比较. 从表中可以看出, 在传统算法中平均搜索深度与散点数目也是线性增长的, 而基于边指针搜索及区域划分的算法的平均搜索深度很小. 即使散点数量成倍增加, 平均搜索深度也只是稍微增加, 证明边指针与区域划分结合的算法实时性要高于传统 Delaunay 三角剖分算法. 其原因在于, 区域划分缩小了目标三角形的搜索范围, 边指针优化了搜索方向, 最终使得搜索深度大大减小, 执行时间得到降低.

表 3　算法的平均搜索深度

Table 3　Average search depth of algorithms

散点数量	2 万	3.5 万	5 万	7 万	10 万
TD	10 003	17 543	24 989	34 893	49 854
EPD	108	229	233	247	278
EPRDD	8	11	13	15	18

4　结论

为了满足三维重建的实时性, 针对其基础部分三角剖分处理时间过长的问题, 本文提出了一种基于边指针搜索及区域划分的 Delaunay 三角剖分算法. 在对传统 Delaunay 三角剖分算法进行研究及

实现的基础上, 进一步优化了三角形存储的数据结构及搜索过程, 设计了一种通过边指针反映三角形空间相邻关系的三角形数据结构来取代传统算法中的一维三角形链表. 由于边指针具有方向性, 可以用于确定距离插入点最近的相邻三角形, 从而快速确定目标三角形的搜索方向. 在此基础上将整个超级三角形进行区域划分, 且随着散点的插入, 每个区域将会分配一个区域入口三角形, 作为搜索目标三角形的起始三角形, 且入口三角形在构建 Delaunay 三角网的过程中会不断更新. 这使得目标三角形的搜索范围由以前的整个三角形链表缩小到插入点所在区域, 从而进一步缩短搜索的起始三角形与目标三角形之间的距离, 大大缩小了搜索范围, 降低了搜索目标三角形的耗时, 以满足三维重建的实时性需求. 实验结果表明, 该算法的执行时间随散点数量的增长较为缓慢, 且散点数量为 10 万时, 所耗费的执行时间仅为 0.813 s.

本文中的区域划分为固定划分, 当散点分布密度变化大时, 对于密度大的区域搜索深度可能会相应增加. 为了进一步扩大算法的适用范围, 后续研究中将引入自适应算法, 寻求一种能够根据散点分布密度或者搜索深度的变化对区域划分方式进行调整的优化算法.

References

1 Chen Jia, Zhang Yu-Qi, Song Peng, Wei Yan-Tao, Wang Yu. Application of deep learning to 3D object reconstruction from a single image. *Acta Automatica Sinica*, 2019, **45**(4): 657−668
(陈加, 张玉麒, 宋鹏, 魏艳涛, 王煜. 深度学习在基于单幅图像的物体三维重建中的应用. 自动化学报, 2019, **45**(4): 657−668)

2 Xue Jun-Shi, Yi Hui, Wu Zhi-Huan, Chen Xiang-Ning. A hybrid multi-View 3D reconstruction method based on scene graph partition. *Acta Automatica Sinica*, 2020, **46**(4): 782−795
(薛俊诗, 易辉, 吴止锾, 陈向宁. 一种基于场景图分割的混合式多视图三维重建方法. 自动化学报, 2020, **46**(4): 782−795)

3 Liu Jian, Bai Di. 3D point cloud registration algorithm based on feature matching. *Acta Optica Sinica*, 2018, **38**(12): 240−247
(刘剑, 白迪. 基于特征匹配的三维点云配准算法. 光学学报, 2018, **38**(12): 240−247)

4 Liu Hai-Qiang , Yu Jian-Bo. A bidimensional local mean decomposition algorithm. *Journal of Computer-Aided Design & Computer Graphics*, 2018, **30**(10): 1859−1869
(刘海强, 余建波. 二维局部均值分解算法. 计算机辅助设计与图形学学报, 2018, **30**(10): 1859−1869)

5 Yan Zi-Geng, Jiang Jian-Guo, Guo Dan. Image matching based on SURF feature and Delaunay triangular meshes. *Acta Automatica Sinica*, 2014, **40**(6): 1216−1222
(闫自庚, 蒋建国, 郭丹. 基于 SURF 特征和 Delaunay 三角网格的图像匹配. 自动化学报, 2014, **40**(6): 1216−1222)

6 Liu Qin-Qin. Overview of point location algorithm of Delaunay triangulation on plane domain. *Electronic Design Engineering*, 2017, **25**(1): 47−51
(刘琴琴. 平面域 Delaunay 三角网点定位算法研究综述. 电子设计工程, 2017, **25**(1): 47−51)

7 Lin J, Chen R, Yang C, Shu Z, Wang C, Lin Y, et al. Distributed and parallel Delaunay triangulation construction with balanced binary-tree model in cloud. In: Proceedings of 15th International Symposium on Parallel and Distributed Computing. Fuzhou, China: IEEE, 2017. 107−113

8 Lin J, Chen R, Wu L, Shu Z, Yang C. Adaptive parallel Delaunay triangulation construction with dynamic pruned binary tree model in Cloud. *Concurrency & Computation Practice & Experience*, 2017, **29**(8): e4175

9 Bowyer A. Computing Dirichlet tessellations. *The Computer Journal*, 1981, **24**(2): 162−166

10 Watson D F. Computing the N-Dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal*, 1981, **24**(2): 167−172

11 Lawson C L. Software for C^1 surface interpolation. *Mathematical Software*, 1977: 161−194

12 Liu J F, Yan J H, Lo S H. A new insertion sequence for incremental Delaunay triangulation. *Acta Mechanica Sinica*, 2013, **29**(1): 99−109

13 Lo S H. Delaunay triangulation of non-uniform point distributions by means of multi-grid insertion. *Finite Elements in Analysis & Design*, 2013, **63**(4): 8−22

14 Su T, Wang W, Lv Z, Wu W, Li X. Rapid Delaunay triangulation for randomly distributed point cloud data using adaptive Hilbert curve. *Computers & Graphics*, 2016, **54**: 65−74

15 Zalik B, Kolingerova I. An incremental construction algorithm for Delaunay triangulation using the nearest-point paradigm. *International Journal of Geographical Information Science*, 2003, **17**(2): 119−138

16 Zadravec M, Zalik B. An almost distribution-independent incremental Delaunay triangulation algorithm. *Visual Computer*, 2005, **21**(6): 384−396

17 Xi J H, Zuo S C. An improved algorithm based on incremental insertion in delaunay triangulation. *Applied Mechanics and Materials*, 2013, **397−400**: 1691−1694

18 Kohout J, Kolingerova I. Parallel Delaunay triangulation based on circum-circle criterion. In: Proceedings of the 19th Spring Conference on Computer Graphics. Slovakia: ACM, 2003. 73−81

19 Rong G, Tan T S, Cao T T, Stephanus. Computing two-dimensional Delaunay triangulation using graphics hardware. In: Proceedings of the 2008 Symposium on Interactive 3D Graphics and Games. California, USA: ACM, 2008. 89−97

20 Cuong N, Philip J R. TIPP: Parallel delaunay triangulation for large-scale datasets. In: Proceedings of the 30th International Conference on Scientific and Statistical Database Management. Bolzano, Italy: ACM, 2018. 1−12

21 Lo S H. Parallel Delaunay triangulation — application to two dimensions. *Finite Elements in Analysis & Design*, 2012, **62**: 37−48

22 Yang Haoyu, Liu Li, Zhang Cheng, Yu Hao. Parallelism-oriented dynamic incremental Delaunay triangulation algorithm. *Journal of Frontiers of Computer Science and Technology*, to be published
(杨昊禹, 刘利, 张诚, 于灏. 面向并行的动态增量式 Delaunay 三角剖分算法. 计算机科学与探索, 即将出版)

23 Li Guo-Qing, Ma Feng-Shan, Deng Qing-Hai. Delaunay triangulation based on convex polygons. *Engineering Geology Computer Application*, 2007(2): 8−10
(李国庆, 马凤山, 邓清海. 基于凸多边形的 Delaunay 三角剖分. 工程地质计算机应用, 2007(2): 8−10)

24 Graham R L. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1972, **1**(4): 132−133

25 Cui Guo-Hua, Hong Fan, Yu Xiang-Xuan. A class of optimal algorithms for determining the convex hull of a set of nodes in a plane. *Chinese Journal of Computers*, 1997, **20**(4): 330−334
(崔国华, 洪帆, 余祥宣. 确定平面点集凸包的一类最优算法. 计算机学报, 1997, **20**(4): 330−334)

26 Liu Bin, Wang Tao. An efficient convex hull algorithm for pPlanar point set based on recursive method. *Acta Automatica Sinica*, 2012, **38**(8): 1375−1379
(刘斌, 王涛. 一种高效的平面点集凸包递归算法. 自动化学报, 2012, **38**(8): 1375−1379)

27 Jia Xiao-Lin, Wu Li-Xin, Wang Yan-Bing. Two dimensional local updating for Delaunay TIN: point insertion and point deletion. *Geography and Geo-Information Science*, 2004, **20**(5): 28−31
(贾晓林, 吴立新, 王彦兵. 二维 Delaunay 三角网局部更新: 点插入与点删除. 地理与地理信息科学, 2004, **20**(5): 28−31)

张 俊 中南大学自动化学院教授. 2004 年获得日本东北大学机械智能工学博士学位. 主要研究方向为三维图形芯片即处理器软件, 图形图像处理加速 FPGA 设计, 电机驱动控制器 FPGA 设计, SOC 系统集成芯片.
E-mail: zhangjun1000@hotmail.com
(**ZHANG Jun** Professor at the School of Automation, Central South University. He received his Ph. D. degree in Mechanical Intelligence Engineering from Tohoku University, Japan in 2004. His research interest covers 3D graphics chip and processor software, graphics image processing accelerated FPGA design, motor drive controller FPGA design, and SOC system integrated chip.)

田慧敏 中南大学计算机学院硕士研究生. 主要研究方向为三维点云数据处理, 计算机视觉. 本文通信作者.
E-mail: thuimin2018@126.com
(**TIAN Hui-Min** Master student at the School of Computer Science and Engineering, Central South University. Her research interest covers 3D point cloud data processing, and computer vision. Corresponding author of this paper.)