软件学报 ISSN 1000-9825, CODEN RUXUEW
2025,36(9):4241−4249 [doi: 10.13328/j.cnki.jos.007307] [CSTR: 32375.14.jos.007307]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于网格行创建条带结构的点在多边形内判断方法*

李佳玮[1,2,3], 王盛春[1,2,4], 王文成[1,2,3]

[1](基础软件与系统重点实验室 (中国科学院 软件研究所), 北京 100190)

[2](计算机科学国家重点实验室 (中国科学院 软件研究所), 北京 100190)

[3](中国科学院大学, 北京 100049)

[4](中船智海创新研究院, 北京 100036)

通信作者: 王文成, E-mail: whn@ios.ac.cn

摘　要: 对于点在多边形内的检测处理, 近期提出的一种网格法具有很高的计算效率. 该方法对于每个网格单元内的多边形片段进行条带结构的组织, 使得每个条带中的边均与该条带的左右边界相交. 如此, 该方法加强了局部化计算, 并能方便使用 GPU 进行并行计算, 使得检测效率优于以往的各种方法. 但该方法基于网格单元创建条带结构, 会产生冗余的条带, 并且创建时的空间需求较大而不便在 GPU 上创建条带结构. 对此, 提出基于网格行创建条带结构, 由此可消除冗余的条带, 减少创建计算的空间需求, 因而能在 GPU 上进行条带结构的创建, 提高工作效率. 实验表明, 相比原有方法, 新方法大幅加快了条带结构的创建, 甚至可加速 40 余倍, 并且有更快的检测速度, 能更高效地处理动态多边形.

关键词: 点在多边形中判断; 网格法; 条带

中图法分类号: TP393

中文引用格式: 李佳玮, 王盛春, 王文成. 基于网格行创建条带结构的点在多边形内判断方法. 软件学报, 2025, 36(9): 4241–4249. http://www.jos.org.cn/1000-9825/7307.htm

英文引用格式: Li JW, Wang SC, Wang WC. Row-based Stripe Construction in Grids for Point-in-polygon Tests. Ruan Jian Xue Bao/Journal of Software, 2025, 36(9): 4241–4249 (in Chinese). http://www.jos.org.cn/1000-9825/7307.htm

Row-based Stripe Construction in Grids for Point-in-polygon Tests

LI Jia-Wei[1,2,3], WANG Sheng-Chun[1,2,4], WANG Wen-Cheng[1,2,3]

[1](Key Laboratory of System Software (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[2](State Key Laboratory of Computer Science (Institute of Software, Chinese Academy of Sciences), Beijing 100190, China)

[3](University of Chinese Academy of Sciences, Beijing 100049, China)

[4](CSSC Intelligent Innovation Research Institute, Beijing 100036, China)

Abstract: In terms of point-in-polygon tests, a grid method proposed recently exhibits high computational efficiency. This method organizes the polygon fragments within each grid cell into stripe structures, ensuring that edges in each stripe intersect with both the left and right boundaries of the stripe. In this way, localization computation is enhanced, and GPUs are used for convenient parallel computation, resulting in a detection efficiency superior to that of various previous methods. However, stripe structures constructed based on grid cells generate redundant stripes. Besides, the method has a high space requirement for stripe construction, making it inconvenient to construct stripe structures on GPUs. In response to this, this study proposes to construct stripe structures via grid rows. Thus, redundant strips can be eliminated, and the space requirement for the creation of computation is reduced, due to which stripe structures can be constructed on GPUs, and work efficiency is improved. Experimental results show that, compared with the original method, the new method significantly accelerates the construction of stripe structures, even by over 40 times. Moreover, it has a faster detection speed and can handle dynamic polygons more efficiently.

* 基金项目: 国家自然科学基金 (62072446)
收稿时间: 2024-04-17; 修改时间: 2024-08-19; 采用时间: 2024-10-09; jos 在线出版时间: 2025-02-19
CNKI 网络首发时间: 2025-02-19

Key words: point-in-polygon test; grid method; stripe

判断一个点是否位于一个多边形内, 是计算几何和计算机图形学中的一个基本问题, 有广泛的应用需求, 如地理信息系统中的位置确定[1]、遥感数据的地域分类[2]、计算机辅助设计[3]等. 针对该问题, 已提出很多方法, 其中网格法[4,5]便于实现, 有很高的计算效率, 是当前主流的方法之一. 该方法对多边形的包围盒进行均匀网格划分, 并在各个网格单元 (简称网元) 中记录所包含的多边形的边; 这样, 检测一个点是否位于多边形内时, 可基于均匀网格的规整性快速定位其所在的网元, 然后只处理该网元所含的多边形的边, 即可判断该点是否位于多边形内. 迄今, 已提出了多种网格法, 如预计算网元中心点位于多边形内/外属性的方法[6]、网格分辨率优化的方法[7]等. 近期提出的一种网格法[8], 对网元中所含的多边形边片段进行条带结构的组织管理, 简称网元法, 大幅提升了网格法的检测效率, 优于已有的各种方法. 该方法建立的条带结构, 确保条带中的多边形边片段均与该条带的左右两边相交 (不失一般性, 假设条带按平行 y 轴的方向建立, 见图 1), 并预先计算各个条带的上端位于多边形内/外的属性. 这样, 测试点可通过定位其所在的条带结构, 只处理该条带中的多边形的边, 即可判断其是否位于多边形内. 这样, 加强了局部化处理, 降低了计算复杂性, 同时有利于使用 GPU 进行并行计算, 能很好地提高点在多边形内检测的效率. 如图 1 所示, 网元以浅色网格线划分而得, x_i (i=1,…,4) 表示位于一条网格横线上的带箭头虚线与多边形的边的交点, s_j (j=1,…,13) 表示一个网格行中生成的 13 个条带结构 (以它们之间平行 y 轴的断线和双实线分隔). 在此, 有冗余的条带 (它们所含的多边形边的情况一样), 如 s_4 与 s_5 同性, s_8 与 s_9 同性, s_{10} 与 s_{11} 同性. 该方法将在第 2 节具体介绍.

网元法很好地提高了测试效率, 但其基于网元创建条带结构, 有一些不足: 1) 一个网格行中相邻的 2 个网元, 其网格分割线左右两侧的 2 个条带很可能包含相同的一组边, 且它们上端位于多边形内/外的属性也一样, 因而会产生冗余的条带结构; 2) 创建时, 各个网元独自处理, 会有一些冗余计算; 3) 该方法在创建条带结构时, 每个网元预期的空间需求要能处理多边形的所有边, 空间需求大, 妨碍了其在 GPU 上的实现. 为此, 本文提出基于网格行来创建条带结构, 而网元则仅作为定位测试点所在条带的一种加速辅助结构. 这样, 一个网格行中的相邻条带不会一样, 可大幅减少条带数量, 也因而减少大量冗余计算, 见图 2. 特别是, 这样处理, 只需估计每个网格行的预期空间, 使其能处理多边形的所有边即可, 从而大幅降低了空间需求, 因此可在 GPU 上方便实现条带结构的创建. 在第 3 节, 我们将对此进行具体讨论. 新方法基于网格行创建条带结构, 我们称之为行元法.

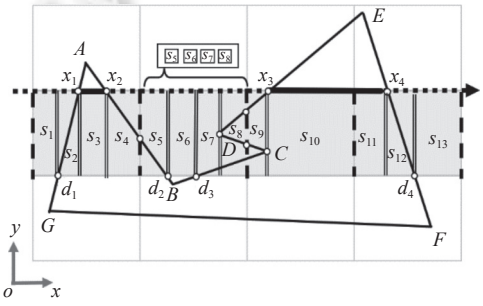


图 1 网元法创建的条带结构

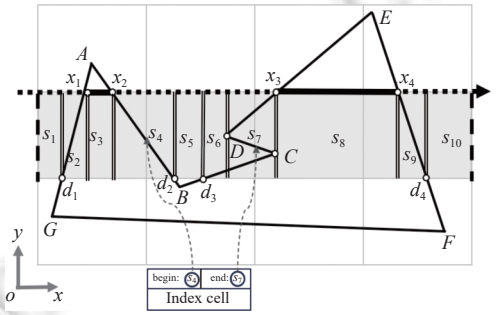


图 2 行元法创建条带的条带结构

实验表明, 相比网元法, 行元法能大幅提升条带结构的创建速度, 甚至高达 40 余倍. 特别是, 行元法能在 GPU 上创建条带结构, 有力提高了处理动态多边形的效率, 在我们实验中可提高帧速达 35.0 倍.

1 相关工作

点在多边形内判断的方法已有很多, 一般可分为两大类: 逐边处理的方法, 及对多边形的边进行一定组织的方法. 逐边处理的方法, 如射线法[9]、绕角法[10,11]等, 它们无需预处理, 但每次检测时要处理多边形的所有边, 如射线法从测试点发出射线, 要检测其与各条边的相交情况, 统计相交次数, 如相交次数为偶数, 则测试点在多边形外; 反

之则在多边形内. 这类方法的计算复杂度较高, 为 O(N), N 为多边形的边数. 为降低测试计算复杂度, 许多方法提出特定数据结构管理多边形的边, 使得每次测试时无需处理多边形的所有边, 如将多边形分解成较简单形状的梯形法[12]、凸剖分法[13]、分层法[14,15], 以及对多边形的包围盒进行均匀划分的网格法[4,5]、层次化迭代划分的四叉树、KD 树[16]、三叉树[17]等. 这些方法能很好地降低检测的计算复杂度, 比如梯形法[12]的计算复杂度为 O(log$N$). 在这些方法中, 网格法是应用最广泛的方法之一, 因为其易于实现, 且计算效率较高. 为提高网格法的效率, 已有大量相关研究[4−7,18], 比如中心点法[6]预计算每个网元中心点位于多边形内/外的属性, 在检测时, 只需计算点与网元中心点之间的连线与其所在网元中的多边形边片段的相交次数. 这样处理可很好地减少需要求交测试的边数量, 因为该连线较短, 可简便剔除许多与连线不相交的边.

网格法的一个困境是网格分辨率的处理. 如果网格分辨率低, 则网元较大, 每个网元所包含的多边形的边就多, 对测试点进行测试时要处理的边就多, 检测效率会降低. 反之, 如果网格分辨率高, 每个网元所包含的多边形的边就少, 有利于测试效率的提高, 但此时网元数量较多, 使得网格结构的创建开销较大, 不便处理动态多边形或边数很多的大规模多边形. 对此, 近期提出的网元法[8]对每个网元中的边进行条带结构的组织, 能较好地化解检测效率对于网格分辨率的敏感性, 大幅提升点在多边形内的检测效率. 这是因为网格分辨率较少, 使得网元较大, 所含多边形的边多, 则网元中的条带就较多; 反之, 网格分辨率较大, 则网元较小, 所含多边形的边少, 因而网元中的条带较少. 但该方法在条带结构的创建时, 会产生过多的条带、有许多冗余计算, 且其空间需求较大, 妨碍了其在 GPU 上的实现, 见引言中相关讨论. 对此, 本文提出行元法, 以改进网元法的不足, 将更好地促进相关应用的发展.

2 网元法简介

网元法[8]是一个基于网格单元均匀划分的点在多边形内检测算法. 该方法将多边形的包围盒向外扩张少许, 使得包围盒的边均位于多边形外; 然后, 对扩张的包围盒进行均匀的网格划分, 并根据每一个网元内的多边形片段创建条带结构, 见图 1.

不失一般性, 根据图 1 对网元法进行简要介绍. 在此, 由浅色实线划分而得的均匀网格单元, 即是网元; 网元中由断线和双细实线分割而成的每个矩形框就是一个条带, 每个条带中的所有多边形边, 均与该条带的左右两边相交. 每个条带记录以下信息: 该条带所包含的多边形的边, 该条带上端位于多边形内/外的属性.

基于这样的组织结构, 点在多边形内的检测, 按以下步骤进行.

1) 根据测试点的坐标, 以 O(1) 的时间获知其所在的网元.

2) 根据该测试点的 x 坐标, 对比该网元中各个条带的 x 坐标范围, 可知测试点所在的条带.

3) 从测试点发出的一条垂直于 x 轴的向上射线, 检测该射线与条带中的边相交情况, 统计相交的边的数量. 根据射线法的原理, 如果有偶数条边相交, 则测试点位于多边形内/外的属性与该条带上端位于多边形内/外的属性相同, 否则相反.

基于网元法进行点在多边形内的判断具有很高的执行效率, 其原因如下: 1) 只需处理一个条带中的多边形的边, 且一个条带所含的多边形的边一般较少; 2) 测试点发出的射线与 x 轴垂直, 使得该射线与一条边的相交测试可以较少的计算来完成, 其处理为: 根据该测试点的 x 坐标, 计算该边的相应点的 y 坐标, 并比较该 y 坐标是否大于测试点的 y 坐标, 即可知晓射线是否与该边相交; 3) 一个条带中的边均与该条带的左右两边相交, 因此, 检测测试点发出的射线与这些边的相交情况时, 便于在 GPU 上进行高效的并行化实现.

下面, 介绍网元法的条带结构的生成步骤如下.

1) 对多边形的每条边, 计算其与各个网格划分横线 (平行 x 轴的网格线) 的交点 x 坐标 (交点的 y 坐标即横线的 y 坐标), 由此知晓各个网元中所包含的多边形边的片段情况.

2) 根据每个网元中所含的多边形边的片段, 创建该网元中的条带结构. 在此, 依据网元内的多边形顶点 (如图 1 中的 D 点), 以及多边形的边与该网元的上、下网格横线的交点 (如图 1 中的 d_i、x_j (i, j = 1,…,4)), 由它们的 x 坐标生成平行 y 轴的直线段 (如图 1 中的双实线), 即得到了该网元内的条带划分结构. 当然, 相邻网元之间的分隔线也

是相关条带的分隔线 (如图 1 中的断线). 然后各个条带记录其所包含的多边形边.

3) 判断各个条带上端位于多边形内/外属性的方法如下: 根据网格划分横线与多边形的交点情况, 该横线可划分为多个横线片段, 然后依据射线法[9]即可知晓各个片段位于多边形内/外属性. 如图 1 中位于网格横线上的带箭头虚线所示, 其与多边形的边相交于 4 个点, 记为 x_i (i=1,…,4); 由此网格横线被划分成 5 个横线片段. 该虚线最左端起始点位于包围盒外, 必定在多边形外; 然后, 其由左向右行进, 每遇到一个交点, 其位于多边形内/外的属性就交替地改变 1 次. 因此, 该射线的 x_1x_2 横线片段、x_3x_4 横线片段是位于多边形内, 其他横线片段位于多边形外. 每个条带的矩形框的上端只能位于其所在网格横线上的一个横线片段内[5], 因此其位于多边形内/外的属性即可知.

如图 1 中所示, 网元法会产生冗余的条带, 如条带 s_8 和 s_9 所包含的多边形边相同, 且它们上端位于多边形内/外的属性也一样. 但它们分属于不同的网元, 因而被重复创建了.

3 行元法

本文提出的行元法, 基于网格行进行条带结构的创建, 可减少创建的条带数量及计算开销, 大幅减少空间需求, 便于在 GPU 上创建条带结构, 克服网元法难以在 GPU 上进行条带结构创建的不足, 很好地提高计算效率.

下面, 我们先介绍行元法创建条带结构的处理步骤, 然后讨论其相比于网元法的改进.

3.1 基于网格行的条带结构创建

不失一般性, 我们依据图 2 介绍行元法创建条带结构的步骤如下.

1) 对多边形的每条边, 计算其与各个网格划分横线 (平行 x 轴的网格线) 的交点 x 坐标 (其 y 坐标即横线的 y 坐标).

2) 对于每一网格行, 其上、下两条网格横线上与多边形的交点以及位于该网格行中的多边形的顶点, 就决定了该网格行中的条带结构. 具体地, 根据这些交点及顶点的 x 坐标, 形成平行 y 轴的直线段, 就得到了该网格行的条带分隔线, 见图 2 中的双实线; 然后, 根据该网格行中各个多边形片段的起止端点, 即可知各个条带所包含的边的情况. 如图 2 中所示, 行元法所得的条带结构, 不会出现相邻条带同性的情况 (所包含的边情况、其上端位于多边形内/外属性均一样). 相比图 1 中网元法创建的条带结构, 图 2 中创建的条带结构就减少了 3 个. 一个网格行中创建的条带结构, 按照条带分隔线的 x 坐标依序排列, 用一个数组管理, 如图 2 中的条带数组依序管理 s_1–s_{10}.

3) 对于各个条带结构的上端位于多边形内/外的属性, 按照第 3 节中介绍的相关处理即可.

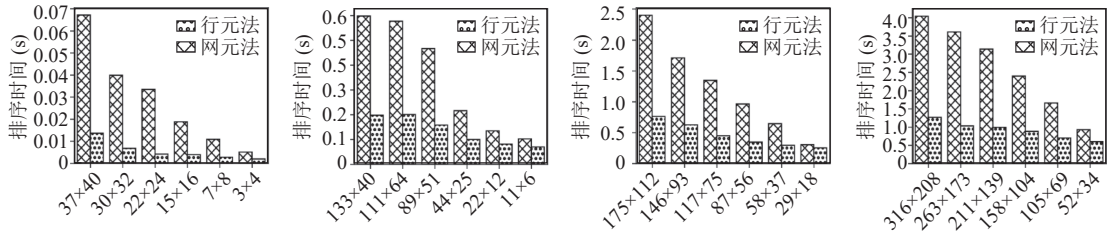
4) 一个网格行中可能包含较多的条带. 为高效找到包含测试点的条带, 我们在各个网格行进行均匀的网元划分, 每个网元只需记录其所包含的条带在条带数组中的首、末下标即可. 如图 2 所示, 包含顶点 D 网元, 只需记录其起始于条带 s_4 并结束于条带 s_7. 这样, 检测时, 以 O(1) 的时间可定位测试点所在的网元, 然后只需检测该网元中的少量条带即可知测试点所在的条带结构, 这与网元法在每个网元中要检测的条带结构的数量是一样的.

3.2 行元法的性能改进

行元法相比于网元法, 有以下一些改进.

1) 减少了条带数量. 基于网元法, 相邻 2 个网元的网格划分线的左右 2 个条带, 往往所包含的多边形边的情况是一样的. 而对于行元法, 这样的 2 个条带只需生成一个. 当网格分辨率为 $r_h \times r_v$ 时, 假设左右相邻的 2 个网元均有 1 个冗余的条带构建, 则行元法最多可减少的条带数量为 $(r_h-1)\times r_v$. 这样就减少了空间需求, 相应地也减少了创建的计算开销, 比如, 创建条带结构时, 行元法可大幅减少调用多边形边的内存访问次数.

2) 行元法对一个网格划分横线上的交点进行统一的快速排序, 以进行各个横线片段位于多边形内/外属性的判断, 具有更高的排序效率. 对此, 网元法的排序处理等价于按网元划分桶后再在每个桶内部进行排序. 根据相关文献[19], 桶排序算法往往只在各桶内要排序的单元数量相近时才会取得比快速排序更高的效率. 显然, 各个网元中的交点数量一般难以一致, 使得网元法对于一条网格划分横线上的交点进行排序的效率, 不及行元法. 如图 3 所示, 对于测试的 4 个多边形 (在实验部分具体介绍), 行元法在创建条带结构时用于排序的时间开销, 明显少于网元法的相关开销.



(a) pol62 (b) pol1248 (c) pol2171 (d) pol7288

图 3 行元法与网元法创建条带结构时用于排序的时间比较

3) 对于一个具有 N 条边的多边形, 假设网格划分分辨率为 $r_h \times r_v$, 则行元法进行条带结构创建所预期的空间复杂度为 $O(r_v N)$, 以便在一个网格行中处理所有的边. 对此, 网元法的预期空间复杂度为 $O(r_h r_v N)$, 以便在一个网元中能处理所有的边. 因此, 处理规模较大的多边形时, 网元法难以在 GPU 上进行条带结构的创建, 只能在 CPU 上创建后再迁移到 GPU 中以进行检测处理. 显然, 这不便处理动态多边形. 由于行元法很好地降低了空间需求, 因而可在 GPU 上创建条带结构, 也节省了从 CPU 往 GPU 迁移条带结构的时间, 能大幅提高动态多边形的处理效率.

4 实验分析

本文提出的行元法, 相比于网元法, 可减少需生成的条带数量, 并降低预处理时的空间预期, 由此可在 GPU 上进行条带结构的创建. 进行点在多边形内的检测时, 行元法与网元法一样, 先依据测试点的坐标, 定位其所在的网元, 并进一步在该网元中定位其所在的条带, 然后在该条带结构内局部地执行射线法, 利用该条带中所包含的多边形的边及条带上端位于多边形内/外属性, 即可得知测试点在多边形内/外的属性. 行元法与网元方在检测质量上是一样的, 因此, 我们在实验中主要比较两者的预处理和检测测试点的效率. 鉴于基于预处理的方法有非层次化组织方法 (如网格法) 和层次化组织方法, 我们也与层次化组织的四叉树、KD 树方法[16]进行了对比实验.

我们的实验安排按照文献 [8] 进行相关处理. 所用的 4 个多边形见图 4. 对各个多边形测试时, 所用的网格分辨率, 及所用的随机而均匀采样的 1×10^8 个测试点, 均依文献 [5] 的处理. 实验所用的设备是一台服务器, 具有一个 Intel Xeon Gold 6154 CPU, 以及一个 NVIDIA Tesla V100 GPU.

(a) pol62 (b) pol1249 (b) pol2171 (d) pol7288

图 4 实验中使用的多边形, 名称中的数据代表多边形边数

4.1 预处理

图 5 中给出了行元法与网元法创建条带结构的时间耗费及加速比 (方法时间耗费的比值), 其中网元法需要将条带迁移到 GPU 中. 行元法在 CPU 或 GPU 上创建条带结构的时间开销, 相比网元法 (网元法标记为 R-GP[8]) 有大幅的降低, 且网格分辨率越高、多边形的边数越多, 则加速效果更好, 甚至可加速 40 余倍. 网元法将 CPU 上创建的条带结构迁移到 GPU 上的时间开销很大, 甚至超过了网元法在 CPU 上创建条带结构的时间. 因此, 行元发直接在 GPU 上进行条带结构的创建, 更有利于动态情况的检测效率提升.

行元法在 CPU 上创建条带结构的时间, 相比其在 GPU 上的处理, 一般需要较多的时间. 但是, 当多边形的边较少且网格分辨率较低时, 行元法在 CPU 上创建条带结构的时间可能少于其在 GPU 上的相关处理, 如较低网格分辨率下处理 pol62 多边形的情况. 相关原因如下: 1) GPU 上对条件判断的分支处理比较麻烦, 相关时间开销较大; 2) 较低网格分辨率时, 网元中含有较多的边, 需要较多的比较运算以进行分支处理; 3) 多边形的边较少时, 求交计算的需求不大, 难以发挥 GPU 的并行性能.

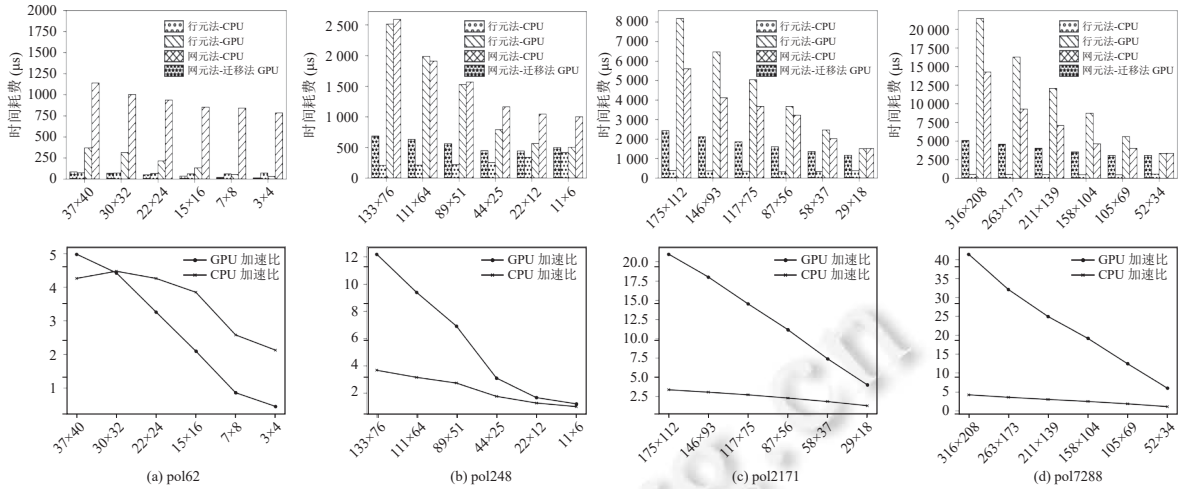


图 5 行元法与网元法创建条带结构及网元法迁移条带结构到 GPU 的时间开销及加速比

后文表 1 中列出了行元法与网元法创建条带结构的对比情况. 相比网元法, 行元法很好地减少了所生成的条带数量, 且网格分辨率越高, 行元法因消除冗余而减少的条带数量就越多, 从而有更高的创建效率. 同时, 表 1 中的统计数据表明, 行元法相比网元法大幅减少了相关的加法次数、比较次数. 由于乘法操作在条带结构创建中只用于多边形的边与网格横线的求交计算, 两者在乘法运算次数方面是一样的.

表 1 行元法与网元法创建条带结构的比较

测试多边形	分辨率	加法次数 (add)		比较次数 (cmp)		生成条带数量	
		网元法	行元法	网元法	行元法	网元法	行元法
pol62	3×4	5 473	2 582	6 193	1 920	118	110
	7×8	4 111	1 574	4 371	2 784	222	174
	15×16	7 285	3 070	7 790	4 536	514	290
	22×24	11 881	5 452	12 730	7 918	922	418
	30×32	17 697	8 582	18 996	12 336	1 466	538
	37×40	24 442	12 354	26 257	17 810	2 106	666
pol1249	11×6	392 676	101 299	392 676	101 299	1 412	1 352
	22×12	302 932	49 870	302 932	49 870	1 853	1 601
	44×25	283 390	45 028	283 390	45 028	3 133	2 063
	89×51	398 688	70 521	398 688	70 521	7 359	2 925
	111×64	486 456	94 927	486 456	94 927	10 317	3 370
	133×76	583 801	123 773	583 801	123 773	13 645	3 770
pol2171	29×18	866 488	52 042	871 248	119 482	4 601	4 105
	58×37	1 010 281	57 462	1 018 505	103 990	8 175	6 156
	87×56	1 315 108	81 313	1 328 014	132 955	12 845	8 239
	117×75	1 697 540	112 683	1 716 104	174 170	18 506	10 262
	146×93	2 128 220	149 443	2 153 151	224 473	24 889	12 176
	175×112	2 667 132	195 713	2 699 863	290 459	32 694	14 307
pol7288	52×34	3 036 383	91 083	3 048 254	178 443	11 532	9 838
	105×69	2 905 475	119 714	2 924 972	191 153	19 309	12 381
	158×104	3 515 891	186 048	3 547 049	283 062	30 925	15 024
	211×139	4 396 810	274 164	4 443 128	407 753	46 052	17 635
	263×173	5 435 737	383 063	5 500 094	567 122	64 145	20 109
	316×208	6 685 722	517 510	6 772 285	765 726	86 350	22 708

4.2 测试效率

按照文献 [8] 中的方式, 我们检测了行元法与网元法测试 1×10^8 个测试点的效率. 相关时间开销的统计数据见图 6. 从这些统计数据可知, 行元法相比网元法在 CPU 和 GPU 上均有相当的检测效率, 且大多数情况下略快. 相关分析见下.

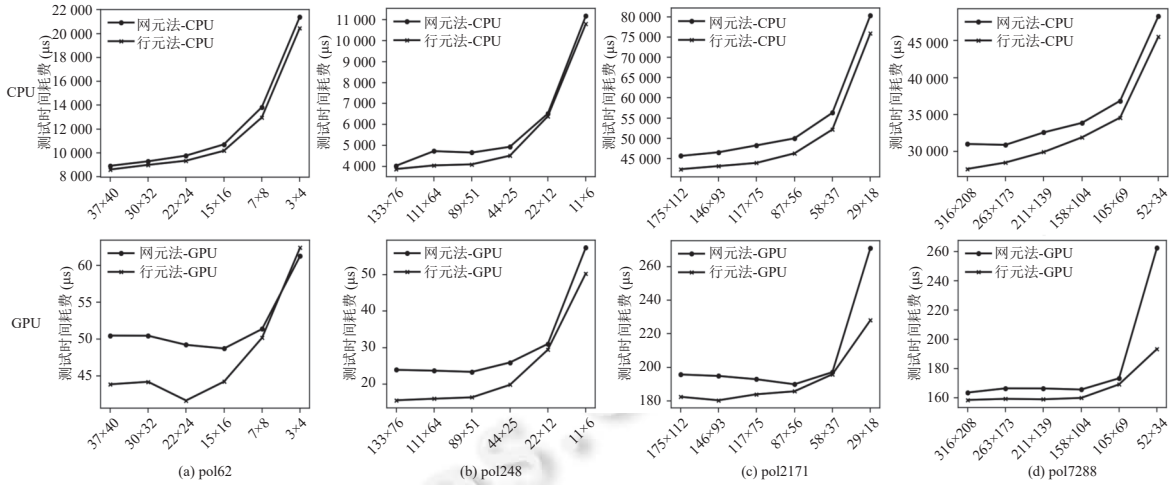


图 6 行元法与网元法在 CPU 和 GPU 上的测试效率比对

行元法与网元法在判断一个测试点位于多边形内/外属性的计算步骤是一致的, 相关计算开销相当. 但是, 行元法对于条带结构的组织管理更紧凑, 使得编译器能够优化出效率更高的寻址操作来找到对应条带. 在我们的实验中, 寻找测试点所在的网元地址, 行元法仅需 1 条指令, 而网元法需要 2 条指令. 因此, 行元法相比网元法可提升一定的检测速度.

4.3 动态多边形的处理

按照文献 [8] 的例子, 我们测试行元法处理动态多边形的效率. 实验模拟海水上涨逐渐淹没一个小岛的过程, 该岛位置为北纬 39°15.17′, 东经 122°59.46′. 该过程按时间序列分成 575 帧处理, 每一帧用一组多边形来描述小岛未被淹没部分. 实验中, 网元法和行元法均需对每一帧的多边形进行条带结构的创建, 然后检测随机挑选的 1×10^4 个位置是否被淹没. 图 7 中给出了一帧的对比情况.

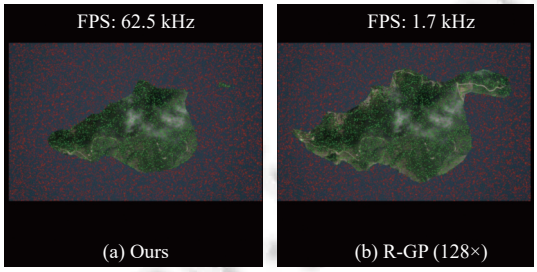


图 7 行元法和网元法处理动态多边形的帧速对比示例

行元法将不同帧的多边形情况顺序送入 GPU 进行预处理和测试, 而网元法需对每一帧的多边形情况在 CPU 上完成条带结构的创建再送入 GPU 进行检测. 实验表明, 网元法的平均帧率为 1.36 kHz, 而行元法的平均帧率为 47.61 kHz, 帧速提高约 35.0 倍.

为测试本文提出的行元法处理各种多边形的情况, 我们设计了一个多边形动态变化的例子以进行相关测试. 此例中, 多边形会剧烈地动态变化, 甚至出现边相交、边重叠等非流形情况. 网元法与本文提出的行元法均可处理这些

多边形, 因为它们都是在各个条带结构内局部地实施射线法, 而射线法只需考虑从测试点发出的射线与多边形边的相交次数, 因而可有效处理各种形态的多边形, 甚至是非流形情况. 图 8 中给出了该动态例子中几个时刻的多边形情况, 位于内部的测试点, 着色为蓝色; 反之, 位于外部的测试点着色为红色. 实验表明, 行元法均能得到正确的结果.

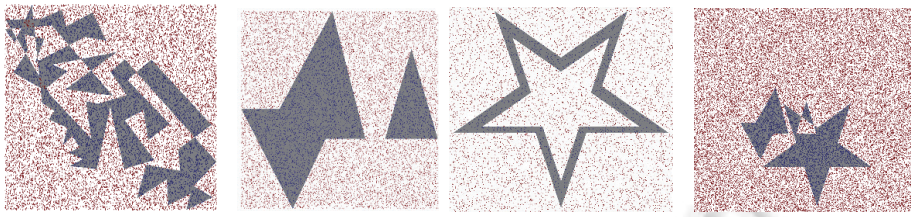


图 8 行元法处理动态多边形几个时刻的情况及多相关测试点检测的情况

4.4 与层次化检测方法的比较

我们在此检测了行元法与四叉树、KD 树方法[16]的对比情况. 四叉树、KD 树方法是在叶节点中存储少量的多边形边, 然后在测试时, 先找到测试点所在的叶子节点, 再在该叶子节点中找到距离测试点最近的点, 以进行相关检测. 参考文献 [8], 我们进行了相关实验设置, 见表 2. 从图 9 的实验情况可知, 行元法的预处理时间介于四叉树与 KD 树之间, 这是因为四叉树自适应生成的层级较浅; 由于行元法要存储较复杂的条带结构, 空间需求较大; 但行元法的检测效率明显优于四叉树、KD 树方法.

表 2 对比算法的实验设置

方法	设置项目	pol62	pol1248	pol2171	pol7288
四叉树	树结构层级	5	7	7	8
KD树		7	10	11	12
行元法	网格分辨率	3×4	11×6	29×18	52×34

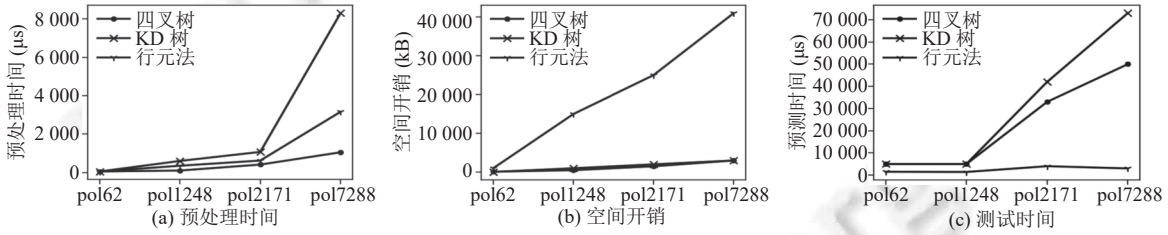


图 9 行元法与四叉树、KD 树的对比实验情况

5 总 结

网元法是近年提出的一种高效检测点是否位于多边形内的方法, 其对网元内的多边形的边片段进行条带结构的组织, 很好地加强了局部化计算, 提高了检测效率. 但网元法基于网元创建条带结构, 会产生冗余条带, 计算开销较大, 且相关空间需求大, 不便在 GPU 上进行条带结构的创建处理. 本文提出行元法, 基于网格行创建条带结构, 可消除大量冗余条带, 大幅减少计算开销和空间需求, 由此便于在 GPU 上创建条带结构, 能更好地提高点在多边形内的检测效率, 特别是有利于处理动态多边形.

对于动态多边形的处理, 我们目前是对每个时刻的多边形进行各自条带结构的创建, 以普适地处理各种动态情况, 如多边形形态的剧烈变化. 当多边形的形态变化不大时, 有些已创建的条带结构可重用以节省创建开销, 但检测可重用的条带结构需要一些开销. 如何优化处理, 需要进一步研究. 另外, 与许多点在多边形内判断的方法一样, 本文提出的行元法也可推广以处理点在多面体内的判断, 比如对多面体的包围盒进行均匀网格划分, 再对网格内的多面体面片构建与条带结构类似的规整组织, 以局部实施射线法进行检测. 我们将在未来对这些作进一步的探讨.

References:

[1] Song WS, Lee JW, Schulzrinne H. Polygon simplification for location-based services using population density. In: Proc. of the 2011 IEEE Int'l Conf. on Communications. Kyoto: IEEE, 2011. 1–6. [doi: 10.1109/icc.2011.5963369]

[2] Ketzner R, Ravindra V, Bramble M. A robust, fast, and accurate algorithm for point in spherical polygon classification with applications in geoscience and remote sensing. Computers & Geosciences, 2022, 167: 105185. [doi: 10.1016/j.cageo.2022.105185]

[3] Tran TA, Lobov A, Kaasa TH, Bjelland M, Midling OT. CAD integrated automatic recognition of weld paths. The Int'l Journal of Advanced Manufacturing Technology, 2021, 115(7): 2145–2159. [doi: 10.1007/s00170-021-07186-0]

[4] Žalik B, Kolingerova I. A cell-based point-in-polygon algorithm suitable for large sets of points. Computers & Geosciences, 2001, 27(10): 1135–1145. [doi: 10.1016/S0098-3004(01)00037-1]

[5] Yang S, Yong JH, Sun JG, Gu HJ, Paul JC. A point-in-polygon method based on a quasi-closest point. Computers & Geosciences, 2010, 36(2): 205–213. [doi: 10.1016/j.cageo.2009.06.008]

[6] Li J, Wang WC. Point-in-polygon test method based on center points of grid. Ruan Jian Xue Bao/Journal of Software, 2012, 23(9): 2481–2488 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4087.htm [doi: 10.3724/SP.J.1001.2012.04087]

[7] Li J. Point-in-polygon tests by applying the ray crossing method locally via grid center points. Int'l Journal of Electrical Engineering: Trans. of the Chinese Institute of Engineers, 2014, 21(3): 85–92. [doi: 10.6329/CIEE.2014.3.02]

[8] Wang WC, Wang SC. Efficient point-in-polygon tests by grids without the trouble of tuning the grid resolutions. IEEE Trans. on Visualization and Computer Graphics, 2022, 28(12): 4073–4084. [doi: 10.1109/TVCG.2021.3073919]

[9] Hughes J F, van Dam A, McGuire M, Sklar D F, Foley J D, Feiner S K, Akeley K. Computer Graphics: Principles and Practice. 3rd ed., Boston: Pearson Educ, 2014.

[10] Preparata F P, Shamos M I. Computational Geometry: An Introduction. New York: Springer, 2012. [doi: 10.1007/978-1-4612-1098-6]

[11] Hormann K, Agathos A. The point in polygon problem for arbitrary polygons. Computational Geometry, 2001, 20(3): 131–144. [doi: 10.1016/S0925-7721(01)00012-8]

[12] Berg M, Cheong O, Kreveld M, Overmars M. Computational Geometry: Algorithms and Applications. Berlin: Springer, 2008. [doi: 10.1007/978-3-540-77974-2]

[13] Li J, Wang WC, Wu EH. Point-in-polygon tests by convex decomposition. Computers & Graphics, 2007, 31(4): 636–648. [doi: 10.1016/j.cag.2007.03.002]

[14] Rueda AJ, Feito FR. El-REP: A new 2D geometric decomposition scheme and its applications. IEEE Trans. on Visualization and Computer Graphics, 2011, 17(9): 1325–1336. [doi: 10.1109/TVCG.2010.246]

[15] Wang WC, Li J, Wu EH. 2D point-in-polygon test by classifying edges into layers. Computers & Graphics, 2005, 29(3): 427–439. [doi: 10.1016/j.cag.2005.03.001]

[16] Samet H. Foundations of Multidimensional and Metric Data Structures. Burlington: Morgan Kaufmann, 2006.

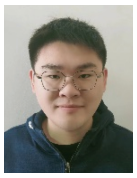
[17] Jiménez JJ, Feito FR, Segura RJ. A new hierarchical triangle-based point-in-polygon data structure. Computers & Geosciences, 2009, 35(9): 1843–1853. [doi: 10.1016/j.cageo.2008.09.013]

[18] Haines E. Point in polygon strategies. In: Heckbert PS. Graphics Gems IV. San Diego: Academic Press Professional, Inc., 1994. 24–26.

[19] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to Algorithms. 4th ed., Cambridge: The MIT Press, 2022.

附中文参考文献:

[6] 李静, 王文成. 基于网格中心点的点在多边形内的高效判定. 软件学报, 2012, 23(9): 2481–2488. http://www.jos.org.cn/1000-9825/4087.htm [doi: 10.3724/SP.J.1001.2012.04087]



李佳玮(2000－), 男, 硕士生, 主要研究领域为可视化, 虚拟现实, 计算机图形学.

王文成(1967－), 男, 博士, 研究员, 博士生导师, CCF 高级会员, 主要研究领域为可视化, 虚拟现实, 计算机图形学, 图像编辑.

王盛春(1991－), 男, 博士, 工程师, 主要研究领域为计算机图形学, 图像处理.