计算机研究与发展
Journal of Computer Research and Development
DOI:10.7544/issn1000-1239.20200689
59(3):706-719,2022

用于索引视域的凸多边形树

苗 雪[1] 郭 茜[1,2] 王昭顺[1] 谢永红[1,2]
[1](北京科技大学计算机与通信工程学院 北京 100083)
[2](材料领域知识工程北京市重点实验室(北京科技大学) 北京 100083)
(miaoxue@xs.ustb.edu.cn)

Convex Polygon Tree for Indexing Field-of-Views

Miao Xue[1], Guo Xi[1,2], Wang Zhaoshun[1], and Xie Yonghong[1,2]
[1](*School of Computer & Communication Engineering, University of Science and Technology Beijing, Beijing* 100083)
[2](*Beijing Key Laboratory of Knowledge Engineering for Materials Science (University of Science and Technology Beijing), Beijing* 100083)

Abstract Smart device, such as smart phones, can record their geographical positions and optical parameters into image files when capturing photos and videos. We can extract and use the information to restore the sector-shaped geographical area, which is known as the field-of-view (FOV). We store the images together with their FOVs in order to support the spatial queries on images. Typically, a user circles a region on the map as a query, and the computer returns the images which capture the region. In fact, the query is to find the FOVs that intersect with the query region. To make FOV queries fast, an index should be built. However, the existing indexes exploit the sector-shaped feature inadequately. We design a convex polygon tree to index pentagons which are the approximations of sectors. Each tree node is a k^*-convex polygon, which encloses a set of polygons. The number of sides of the polygon must be no more than k, and the difference between the polygon and its elements should be minimized. We propose a submerging algorithm to compute such k^*-convex polygons. To build a convex polygon tree, we insert FOVs iteratively. Each time we select a best leaf node for insertion. The selection criteria are to make the dead space and the increasing space as small as possible, and to make the overlap area between the old node and the FOV as large as possible. We also propose an FOV query algorithm based on the proposed convex polygon tree. The experimental results show that the convex polygon tree performs better than the existing indexes.

Key words spatial query; field-of-view (FOV); convex polygon; tree structure; sector-shaped

摘 要 智能手机等设备在拍摄照片和录制视频时会将拍摄位置和光学参数记录到影像文件中,可以提取并利用这些信息,在二维平面空间中还原出图片所对应的扇形视域(field-of-view, FOV).将影像文件及其对应的FOV存储在计算机中,用来支持用户对影像文件的空间查询.一种典型的空间查询是

收稿日期:2020-09-07;**修回日期**:2021-01-07
基金项目:国家自然科学基金项目(61602031);中央高校基本科研业务费专项资金(FRF-BD-19-012A,FRF-IDRY-19-023);国家重点研发计划项目(2017YFB0202303)
This work was supported by the National Natural Science Foundation of China (61602031), the Fundamental Research Funds for the Central Universities (FRF-BD-19-012A, FRF-IDRY-19-023), and the National Key Research and Development Program of China (2017YFB0202303).
通信作者:王昭顺(zhswang@sohu.com)

用户在地图上指定查询区域,计算机找出拍摄到这个区域的影像返回给用户,其实质是找出与查询区域存在交集的 FOV.为了提升查询效率,需要设计合理的数据结构来索引 FOV.然而,现有的索引结构没有充分利用 FOV 的形状特点.使用五边形近似描述 FOV,并设计凸多边形树来索引五边形.树的节点是 k^* 凸多边形.k^* 凸多边形是包围一组多边形的最佳多边形,它的边数不超过 k 并且无效区域最小,即它本身与其内部元素的差集最小.提出了淹没算法来找出这样的包围多边形.在构建凸多边形树时,将逐一插入 FOV,为每个待插入 FOV 选择最优叶子节点的标准是让 FOV 插入后新节点的无效区较小,新节点的增加区较小,并且旧节点与 FOV 的重合区较大.同时,提出了基于凸多边形树的 FOV 查询算法.实验结果表明凸多边形树与现有索引相比可以提升查询效率.

关键词　空间查询;视域;凸多边形;树形索引;扇形

中图法分类号　TP311.13

智能手机在拍摄照片和录制视频时会在图片文件中嵌入设备的地理位置和参数信息.Ay 等人[1]提出使用视域(field-of-view, FOV)描述图片所反映的地理区域,视域是由拍摄地点 loc、相机朝向 θ、可视角度 α 和可视距离 r 决定的扇形区域,如图 1 所示.除了根据内容和关键字来搜索图片[2-4]外,人们也可以根据视域来搜索图片.图 1 中的 5 个扇形 f_1, f_2,…, f_5 分别代表 5 张照片 img_1, img_2,…, img_5 的视域,假设用户指定了查询范围 Q(即图 1 中矩形框),系统返回的结果为照片 img_1, img_2, img_3,因为它们的视域 f_1, f_2, f_3 均与 Q 有交集.

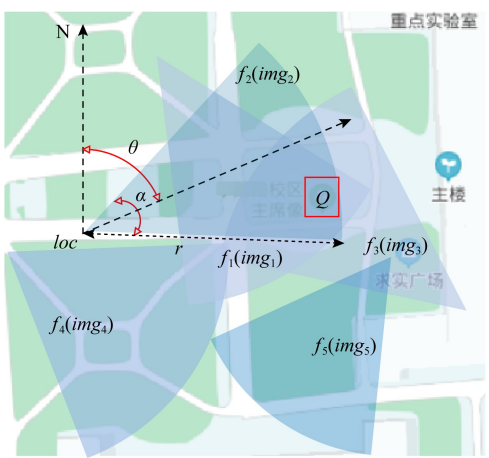
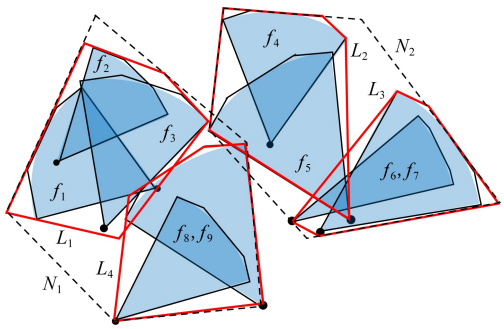


Fig. 1　An example of the FOV query

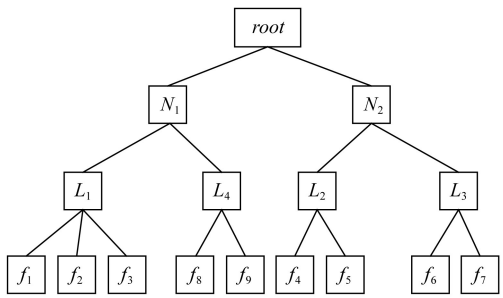
图 1　FOV 查询示例

图 1 示例中的查询对象为视域扇形(FOV),查询范围是一个矩形区域,我们称其为面向 FOV 的窗口查询(简称为 FOV 查询).现有查询算法有 2 种:基于 R* 树[5]及其变体的查询算法[6-8]和基于网格索引的查询算法[9].R* 树索引的对象是 FOV 的最小包围矩形,由于矩形和扇形形状差异较大,导致节点内的无效区(dead space)较大;OR 树[8]是 R* 树的变体,它索引的对象是 FOV 的圆心,所以有时会出现节点间重合较大的情况;Ma 等人[9]提出了三级网格索引,当 FOV 的可视距离差异较大时,查询会出现大量冗余的候选网格.

本文设计了凸多边形树来提升 FOV 查询的性能.凸多边形树索引的对象是包围 FOV 的最小五边形,如图 2(a)中的 f_1～f_9.距离较近的五边形聚合在一起构成上层节点,如图 2(a)中的粗实线多边形 L_1 由 f_1, f_2, f_3 聚合而成,依此类推其余的五边形可聚合成更大的五边形 L_2, L_3, L_4.与 R* 树类似,凸多边形树逐层向上将较近的多边形聚集在一起直到根节点,并且每个节点对应的凸多边形边数小于等于 k,如图 2(a)中的限定 $k=5$.



(a) 二维平面空间中的节点示意图



(b) 树的结构示意图

Fig. 2　Convex polygon tree

图 2　凸多边形树

为了找出可以包围一组多边形的大多边形,我们提出了淹没算法,它一方面控制大多边形的边数,当最小包围多边形的边数大于 k 时,算法将边数减少到 k;另一方面它保证大多边形中无效区最少.我们构建凸多边形树的过程是逐个插入新的 FOV f_{new},算法保证插入 f_{new} 之后新叶子节点中引入无效区的比重小于 $\varepsilon_{无效}$,同时我们也考虑 f_{new} 与旧叶子重合区的比重 $V_{重合}$,以及新叶子增加区的比重 $V_{增加}$.如果 f_{new} 不适合插入任何一个叶子,则将其暂存入等待队列中.在查询时,除了对凸多边形树进行剪枝操作以减小搜索空间之外,我们还提出角度分区筛选算法找出候选结果集合.实验结果表明,虽然创建凸多边形树比创建 R* 树花费的时间多,但基于凸多边形树的 FOV 查询算法比已有查询算法的查询效率更高.

本文工作的主要贡献有 3 个方面:

1) 使用五边形近似 FOV,提出了索引此种五边形的凸多边形树.由于树的节点为多边形,为了保证孩子节点数量稳定,提出了淹没算法来控制多边形的边数.

2) 设计了构建凸多变形树的算法,包括节点插入算法和节点分裂算法.为了优化树的结构,提出了使用等待队列的方法.

3) 制作了仿真数据集并在仿真数据集上对算法进行了性能测试.

1 相关工作

在空间数据库领域中,对于可定位影像数据的研究工作主要包括影像数据视域的建模方法和影像数据的空间查询技术.

1.1 可定位影像数据的视域建模方法

为了建立影像和地理区域之间的映射关系,我们利用影像文件中内嵌的地理坐标和光学参数来重构影像所反映的地理区域.但是,拍摄设备是由多个透镜组成的复杂光学系统,我们很难准确还原光路并推算出影像对应的地理区域.一种常见的做法是忽略拍摄设备的高度并且把拍摄过程简化为小孔成像,使用扇形视域来近似地描述影像对应的地理区域[1,10].随着无人机技术的广泛应用,也有研究者使用二维平面中的四边形来近似地描述无人机在地面上的视域[11-12].这些建模方法将视域简化为二维平面图形,使得我们可以从空间数据查询的角度来搜索图片.比如 Kim 等人[13]设计并开发了 TVDP 平台,并利用空间数据处理技术对可定位城市影像进行查询和分析;Toyama 等人[14]开发了一个用于图片搜索的数据库,它通过拍摄位置的经纬坐标和时间戳索引大规模图像数据;Li 等人[15]提出了一种估计照片视角方向的实用方法,它可以找出非位置感知设备拍摄的具有错误姿势的照片.

1.2 可定位影像数据的空间查询

在空间数据库中,可定位影像数据查询问题的实质是找出与用户输入的查询区域相交的视域.根据应用场景的不同,视域的形状可以抽象为扇形[8]或四边形[11].这类基本查询找出所有与查询区域相交的视域,并将它们全部作为结果返回给用户.为了提升结果的质量,Alfarrarjeh 等人[16]提出根据视域对查询区域的覆盖程度以及视域的朝向对查询结果进行排序,Ay 等人[17]提出了衡量连续视域与查询区域相关性的方法.

为了提高查询效率,针对不同的查询问题,研究者们提出了不同的索引技术.根据索引对地理信息的利用程度,我们将其分为两大类:1)建立索引时仅考虑拍摄设备的地理位置,如 Alfarrarjeh 等人[18]同时考虑设备的位置和影像的语义为图片建立了索引;2)根据设备的视域来构建索引,如 Ay 等人[1]使用 R* 树来索引扇形视域(即 FOV).然而,直接使用 R* 树索引 FOV 会产生较大的节点内无效区和节点间重合区.

为了进一步提升查询效率,Lu 等人[8]提出使用 OR 树来索引 FOV.OR 树是 R* 树的变体,它的节点中不仅存储了包围若干拍摄位置(即 FOV 圆心)的最小边界矩形,而且也存储了这些 FOV 的方向范围和可视距离范围.构建 OR 树的策略是尽量将拍摄位置靠近、可视距离相似和可视角度范围相似的 FOV 放入同一个节点中.这种构建策略的缺点是节点间的重合较大,并且树的结构取决于 FOV 的插入顺序.

本文在 Lu 等人[8]工作的基础上,提出了一种新的 FOV 索引,即凸多边形树,用于支持 FOV 查询.虽然凸多边形比矩形更适合包围扇形等不规则形状,但多边形边数不确定以及形状不规则等因素为索引的构建带来了困难和挑战.本文将在第 3 节详细介绍构建凸多边形树的方法.

1.3 特定场景下的影像数据空间查询

为了更好地索引视频帧在二维平面空间中的视域,Kim 等人[6]提出了 GeoTree.GeoTree 的叶子节点中存储的是用于包围连续视域的最小倾斜矩形.

GeoTree 适用于索引视域方向变化不剧烈的视频.为了更准确地拟合拍摄设备的运动轨迹,Lee 等人[7]提出 GeoVideoIndex,它根据设备位置的变化趋势构造包围矩形,使其可以包围住更多的 FOV,从而减小了索引占用的存储空间和索引的构建时间.Ma 等人[9]提出一种基于可视场景、设备位置和视域方向的三级网格索引,这种网格结构适用于索引可视距离较接近的视域.针对无人机的高空拍摄场景,Lu 等人[11]提出了用于索引四边形视域的 TetraR 树.

2　问题定义与解决方法概述

本节给出了 FOV 查询的正式定义并给出了 FOV 查询的解决方法概述.该过程主要包括 2 个阶段,即索引构建阶段和查询处理阶段.

2.1　问题定义

在二维欧氏空间中,FOV 由 4 个参数决定:设备位置 loc、镜头朝向 θ、最大可视角度范围 α 和最大可视距离 r.前 2 个参数由设备自带的 GPS 和方向传感器自动获取,后 2 个参数可由镜头本身参数推算得到.本文使用四元组(loc,α_b,α_e,r)描述 FOV,其中 α_b 和 α_e 是以顺时针为序的 FOV 可视角度范围的起始角度和终止角度.

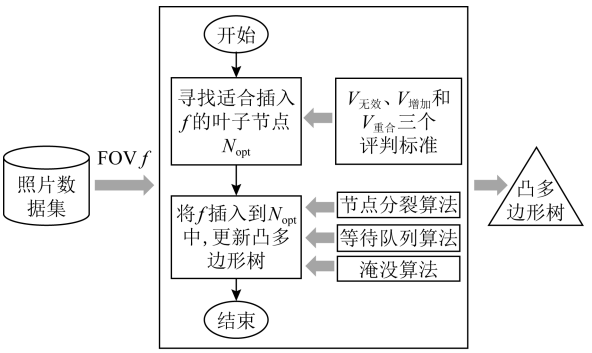
定义 1. FOV 查询.在二维欧氏空间中,用户指定矩形查询区域 Q,系统从已有的 FOV 集合 F 中找出所有与 Q 相交的 FOV.

如图 1 所示,用户指定了可以恰好包围住雕像的矩形查询框 Q,系统判断只有照片 img_1,img_2,img_3 的 FOV 与 Q 相交,所以系统会将这 3 张照片返回给用户.

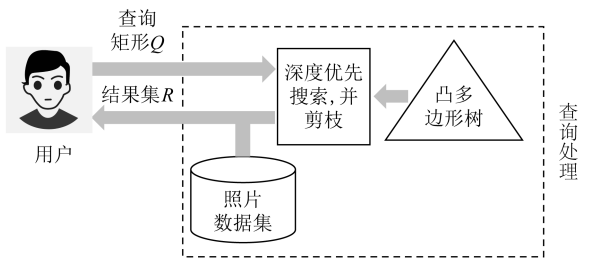
2.2　解决方法概述

解决 FOV 查询的基本方法是检查每个 FOV 是否与查询区域 Q 相交,并且返回与 Q 有交集的 FOV 作为结果.这种方法需要遍历整个 FOV 集合,查询效率较低.为了提高查询效率,我们提出使用凸多边形树索引 FOV 的方法来减小搜索空间.在预处理阶段,我们为整个 FOV 集合构建凸多边形树.在查询处理阶段,我们使用深度优先搜索的方式遍历凸多边形树,在遍历的过程中剪掉不可能包含结果的节点.图 3 展示了索引构建和查询处理的主要流程.

在索引构建阶段,我们使用逐一插入 FOV 的方式来构建凸多边形树.凸多边形树的结构与 R* 树类似,与 R* 树不同的是它的节点对应的区域不是



(a) 索引构建概述图



(b) 查询处理概述图

Fig. 3　The overview of our solution

图 3　解决方法概述图

包围矩形而是包围凸多边形.为了保证孩子节点数量稳定,我们提出淹没算法使包围凸多边形的边数不超过 k(3.1 节将详细介绍淹没算法).在插入某个 FOV f 时,我们根据 $V_{无效}$,$V_{增加}$ 和 $V_{重合}$ 这 3 个评判标准找到适合插入 f 的叶子节点 N_{opt},然后将 f 插入到 N_{opt} 中,并自底向上更新 N_{opt},N_{opt} 的父亲节点,直至根节点.如果我们没有找到适合插入 f 的叶子节点,则将 f 放入等待队列中.如果在插入的过程中出现孩子节点的数量超过 M_{max} 的情况,则执行节点分裂操作(3.2 节将详细介绍节点的插入和分裂策略).

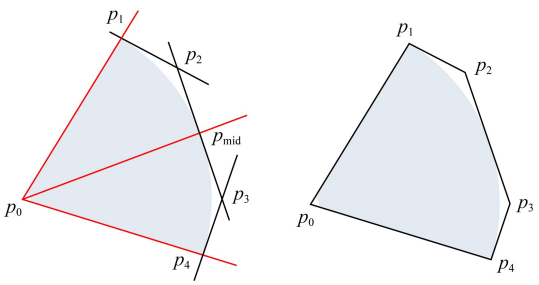
在查询处理阶段,用户输入矩形查询范围 $Q(x_1,y_1,x_2,y_2)$,其中 (x_1,y_1) 和 (x_2,y_2) 分别为 Q 的左下顶点和右上顶点坐标.我们从根节点开始自上而下遍历凸多边形树.当下降至某个节点 N 时,判断其是否与 Q 相交,如果相交,就下降到 N 的孩子节点.最后,将与 Q 有交集的所有 FOV 对应的照片作为结果返回给用户.

以图 1 为例,在索引构建阶段,我们构建一个凸多边形树索引所有的 FOV.在查询处理阶段,当用户输入查询范围 Q(矩形框)之后,我们用深度优先搜索的方式遍历凸多边形树,在遍历的过程中剪掉与 Q 不相交的节点,并将与 Q 相交的 FOV 存入到结果集中.最后,我们将结果集中 FOV 对应的照片展

示给用户,即 img_1,img_2,img_3.当用户输入其他查询范围时,我们可以重复利用凸多边形树进行搜索.

3 凸多边形树

为了方便计算,我们将 FOV 简化为包围五边形,如图 4(a)所示,取弧的边缘点 p_1 和 p_4,以及弧的中点 p_{mid},做此 3 个点的切线,切线的交点 p_2,p_3 与 p_1,p_4,p_0 组合在一起构成 FOV 的包围五边形,如图 4(b)所示.



(a) 计算时用的切线 (b) 包围五边形

Fig. 4 Five-side bounding polygon of an FOV

图 4 FOV 的包围五边形

我们使用 k^* 多边形来包围一个多边形集合,用以构成树的节点.

定义 2. k^* 包围多边形.给定一个多边形集合 $F=\{f_1,f_2,\cdots,f_n\}$,如果一个边数不超过 k 的多边形可以包围住 F 中所有的多边形,并且使无效区最小,则此多边形被称为集合 F 的 k^* 包围多边形,用 $BP_k(F)$表示.

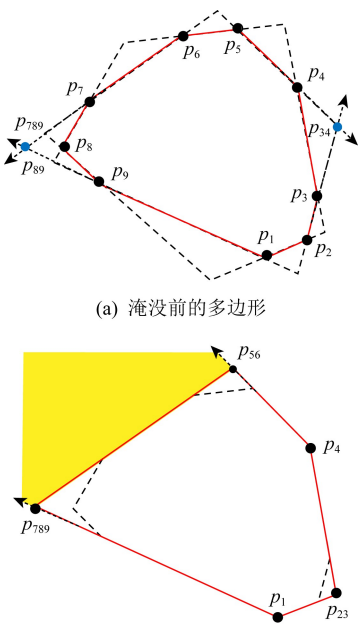
这里无效区是指属于 $BP_k(F)$但不属于 F 的区域,即 $BP_k(F)-F$.如图 2(a)所示,多边形集合 $\{f_1,f_2,f_3\}$的 5^* 包围多边形为 L_1,即 $BP_5(\{f_1,f_2,f_3\})=L_1$.同理,多边形集合$\{L_1,L_4\}$的 5^* 包围多边形为 N_1,即 $BP_5(\{L_1,L_4\})=N_1$.

凸多边形树的索引对象是 FOV 的包围五边形集合 $F=\{f_1,f_2,\cdots,f_n\}$,每个叶子节点负责 m 个 f_i,它存储包围这些 f_i 的 k^* 多边形以及指向这些 f_i 的指针.每个叶子节点的上一层节点负责 m 个叶子节点,它存储包围这些叶子节点的 k^* 多边形以及指向这些叶子节点的指针,依次往上直到根节点.跟 R^* 树类似,凸多边形树中每个叶子节点(或中间节点)包含的 FOV(或孩子节点)数目 m 在 $M_{\min}$ 和 $M_{\max}$之间且根节点至少有 2 个孩子节点.

图 2 的例子中有 9 个 FOV 的包围五边形$\{f_1,f_2,\cdots,f_9\}$,其中$\{f_1,f_2,f_3\}$由叶子节点 L_1 负责,即 L_1 用更大的 k^* 多边形($k=5$)包围了$\{f_1,f_2,f_3\}$.同理,$\{f_4,f_5\}$由叶子节点 L_2 负责,$\{f_6,f_7\}$由叶子节点 L_3 负责,$\{f_8,f_9\}$由叶子节点 L_4 负责.叶子节点$\{L_1,L_2,\cdots,L_4\}$又进一步聚合成 2 个更大的 k^* 多边形 N_1 和 N_2,而中间节点$\{N_1,N_2\}$又进一步向上聚合成根节点 $root$,最终形成如图 2(b)所示的树形结构.

3.1 淹没算法

本节提出淹没算法来找出包围一组多边形集合 $F=\{f_1,f_2,\cdots,f_m\}$的 k^* 多边形.淹没算法的起点是包围 F 的最小凸多边形 $BP(F)$,如图 5(a)中的多边形为包围某多边形集合 F 的最小多边形 $BP(F)$,为了让图更简洁,我们没有画出集合 F 中的多边形.如果 $BP(F)$的边数$\leqslant k$,则 $BP(F)$为 F 的 k^* 多边形,算法直接将此多边形作为结果返回;如果 $BP(F)$的边数$>k$,则算法每次淹没一条 $BP(F)$的边,直到边数减少到 k 为止.



(a) 淹没前的多边形

(b) 淹没后的多边形

Fig. 5 Submerging convex polygon sides when $k=5$

图 5 $k=5$ 时凸多边形的淹没操作

淹没操作选择 2 个相邻顶点 p_i 和 p_{i+1},将边 $p_{i-1}p_i$ 延长并且将边 $p_{i+2}p_{i+1}$延长,用这 2 条延长线的交点取代 p_i 和 p_{i+1}将淹没掉边 p_ip_{i+1}.如图 5(a),如果想淹没边 p_3p_4,则延长 p_2p_3 和 p_5p_4,用延长线的交点 p_{34}作为多边形的新顶点,这个操作将淹没掉边 p_3p_4.通过淹没操作得到的新多边形仍然是凸多边形.

多边形的新增区域为新顶点与2个旧顶点组成的三角形,如图5(a)中的 $\Delta p_3p_{34}p_4$.此三角形为无效区,因为此三角形与 F 中的所有多边形都没有交集.为了尽量减小节点的无效区,我们每次选择淹没当前多边形的一条最优边,这里最优边的含义是如果淹没它那么引入的三角形最小.如图5(b)所示,我们首先淹没边 p_2p_3,因为淹没它引入的三角形最小.在得到新的多边形之后,我们选择淹没最优边 p_8p_9,依此类推,我们淹没 p_5p_6,然后淹没 p_7p_{89},其中 p_{89} 为淹没边 p_8p_9 时生成的新顶点.最后,算法得到 $k=5$ 的 k^* 多边形.

算法1. 淹没算法.

输入:BP,k;

输出:BP_k.

① $L\leftarrow\{\langle p_1p_2,\Delta p_1p_2\rangle,\cdots,\langle p_np_1,\Delta p_np_1\rangle\}$;

② while BP 边数大于 k do

③　遍历 L 找出最优边 p_ip_{i+1};

④　求边 $p_{i-1}p_i$ 与边 $p_{i+2}p_{i+1}$ 的延长线交点 p_{new};

⑤　$BP\leftarrow BP-\{p_i,p_{i+1}\}\cup\{p_{\text{new}}\}$;

⑥　$L\leftarrow L-\langle p_ip_{i+1},\Delta p_ip_{i+1}\rangle$;

⑦　$\langle p_{i-1}p_{\text{new}},\Delta p_{i-1}p_{\text{new}}\rangle\leftarrow\langle p_{i-1}p_i,\Delta p_{i-1}p_i\rangle$;

⑧　$\langle p_{\text{new}}p_{i+2},\Delta p_{\text{new}}p_{i+2}\rangle\leftarrow\langle p_{i+1}p_{i+2},\Delta p_{i+1}p_{i+2}\rangle$;

⑨ end while

⑩ $BP_k\leftarrow BP$.

淹没算法如算法1所示.算法的输入是凸多边形 BP[①] 和 k,输出是 k^* 多边形 BP_k.首先,我们使用链表 L 存储凸多边形每条边 p_ip_{i+1} 及其对应三角形 Δp_ip_{i+1}[②] 的面积(行①).然后,算法每次淹没一条最优边(行②~⑨),直到多边形边数是 k 为止.每次循环时,算法遍历整个 L 找出最优边(行③),求出新顶点 p_{new}(行④)并更新多边形的顶点集合 BP(行⑤).此外,算法还要将被淹没的边及其三角形从 L 中去掉(行⑥),并更新两条旧边(行⑦⑧).最后,算法得到 k^* 多边形 BP_k(行⑩).

在使用算法1时,需要注意多边形可能存在无法被淹没的边.如图5(b)所示,假设我们想淹没边 $p_{56}p_{789}$,而 p_1p_{789} 的延长线和 p_4p_{56} 的延长线无交点.遇到此种情况时,我们将对应的三角形面积设置为无穷大,让此种边不会被选中.

3.2　节点插入策略

我们采用逐个插入FOV[③] 的方式构建凸多边形树.插入一个FOV f 分为2步:1)自顶向下找出插入 f 的最优叶子节点 N_{opt},这步被称为寻找父节点;2)将 f 插入到 N_{opt} 中并自底向上更新 N_{opt},N_{opt} 的父节点,直至根节点,这步被称为更新父节点.在寻找父节点时,根据 $V_{\text{无效}}$,$V_{\text{增加}}$ 和 $V_{\text{重合}}$ 三个评判标准找出最优叶子节点.$V_{\text{无效}}$ 用于衡量如果将 f 插入节点 N,那么新节点 N_{new} 会出现多少无效区,即:

$$V_{\text{无效}}=(Area_{N_{\text{new}}}-Area_{N\cup f})/Area_f,\quad(1)$$

其中,$Area_{N_{\text{new}}}$ 表示包围 f 和 N 的 k^* 多边形的面积,$Area_{N\cup f}$ 表示 $N\cup f$ 本身的面积,$Area_f$ 表示 f 本身的面积.如图6所示,旧节点 N 为实线填充的区域,f 为无任何填充的区域,新节点 N_{new} 为既包含 N 也包含 f 的多边形,而虚线填充的区域为无效区.$V_{\text{增加}}$ 用于衡量新节点 N_{new} 相比于原节点 N 增加了多少面积,即:

$$V_{\text{增加}}=Area_{N_{\text{new}}}-Area_N,\quad(2)$$

其中,$Area_N$ 表示 N 本身的面积.$V_{\text{重合}}$ 用于衡量 f 与 N 的重合区域的大小,即:

$$V_{\text{重合}}=Area_{N\cap f}/Area_f.\quad(3)$$

其中,$Area_{N\cap f}$ 表示 f 与 N 重合区域的面积,如图6所示的粗五边形区域.

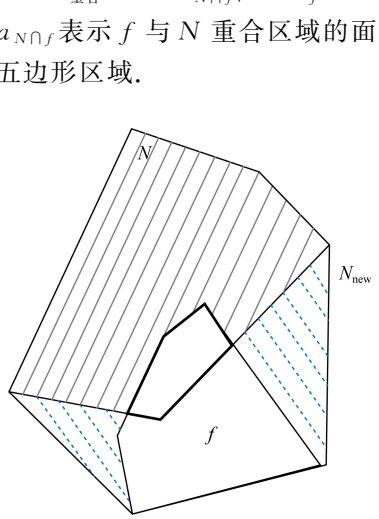


Fig. 6　Display of the old node N, FOV f, and new node N_{new}

图6　旧节点 N,FOV f 和新节点 N_{new} 的展示

我们寻找最优叶子 N_{opt} 的策略是,优先筛选出满足 $V_{\text{无效}}\leqslant\varepsilon_{\text{无效}}$(条件 A)的叶子 L_A.此种叶子 $L_i\in L_A$ 的优点是 f 插入 L_i 后引入的无效区较小.然后,我们分3种情况处理:

① 包围集合 F 的最小凸多边形 $BP(F)$ 简写为 BP,k^* 多边形 $BP_k(F)$ 简写为 BP_k.

② 三角形 $\Delta p_ip_{i,i+1}p_{i+1}$ 简写为 Δp_ip_{i+1},如 $\Delta p_3p_{3,4}p_4$ 简写为 Δp_3p_4.

③ 在不会发生混淆的情况下,FOV除了指视域扇形本身之外,也指包围此扇形的凸五边形.

1) 如果此种叶子只有 1 个,即$|L_A|$[①]$=1$,则此叶子即为最优叶子 N_{opt},我们将 f 插入 N_{opt}.

2) 如果没有此种叶子,即$|L_A|=0$,则说明 f 远离当前的所有叶子,我们创建一个仅包含 f 的新叶子 N_{opt},并将其插入到合适的上层节点中.此处上层节点是指叶子节点的上一层节点.

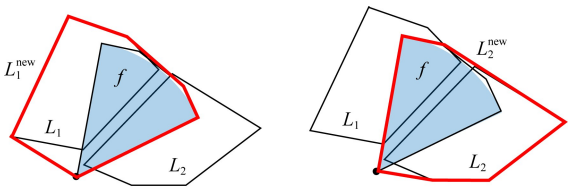
3) 如果此种叶子有多个,即$|L_A|\geqslant 2$,我们根据 $V_{重合}$进一步从 L_A 中筛选出 $V_{重合}\geqslant\varepsilon_{重合}$(条件 B)的叶子 L_B.筛选此种叶子的动机是:f 插入与其重合较大的叶子后,会使得新叶子与其兄弟叶子的重合较大.

根据集合 L_B 中元素的数量,我们分 3 种情况讨论:

① 如果此种叶子只有 1 个,即$|L_B|=1$,则此叶子即为最优叶子 N_{opt},我们将 f 插入 N_{opt}.

② 如果没有此种叶子,即 $L_B=\varnothing$,我们从 L_A 中找出 $V_{增加}$最小的叶子作为最优叶子 N_{opt},原因是 f 插入此叶子后引入的新增面积较少,也会在一定程度上减小新叶子与其兄弟叶子之间的重合.

③ 如果此种叶子为多个,即$|L_B|\geqslant 2$,我们将 f 放入等待队列 W 中,原因如图 7 所示,2 个叶子节点 L_1 和 L_2 既满足条件 A 也满足条件 B,即$L_B=\{L_1,L_2\}$.如果将 f 插入 L_1 中,则生成的新叶子节点 L_1^{new} 与 L_2 重合较大,如图 7(a)所示.如果将 f 插入 L_2 中,则生成的新叶子节点 L_2^{new} 与 L_1 重合较大,如图 7(b)所示.总之,在插入 f 时,选择 L_1 和 L_2 中任何一个都会导致兄弟叶子之间出现较大的重合,所以,我们暂时不插入 f,而是将其放入等待队列中.



(a) 将f插入到L_1中 (b) 将f插入到L_2中

Fig. 7 Waiting list

图 7 等待队列

算法 2 描述了将一个 FOV f 插入到凸多边形树的过程.首先,算法根据条件 A 筛选出叶子集合 L_A(行①).筛选的过程是从根节点开始向下搜索,将符合条件 A 的孩子节点展开,直到找出所有符合条件 A 的叶子为止.此处可以逐层展开中间节点的依据是如果父节点不满足条件 A,则其孩子节点必然不满足条件 A.然后,算法根据 L_A 中元素的数量分 3 种情况处理:1)行②~⑤对应情况 2,其中算法找出 $V_{无效}$最小的节点作为合适的上层节点,也就是说将新叶子 N_{opt}插入到此节点后引入的无效区最小(行④);2)行⑥~⑧对应情况 1,其中行⑧将 f 插入到 N_{opt}之后,要逐层向上更新父节点,直到根节点为止;3)行⑨~⑳对应情况 3,算法首先根据条件 B 进一步筛选 L_A 中的叶子得到集合 L_B(行⑩),根据 L_B 集合中元素的个数又细分为 3 种情况.行⑪~⑬对应情况 3 中①;行⑭~⑯对应情况 3 中②;行⑰⑱对应情况 3 中③,其中 W 表示等待队列.

算法 2. 节点插入算法.

输入:凸多边形树的根节点 *root*、准备插入树中的 FOV f;

输出:插入 f 之后的树.

① $L_A\leftarrow$满足 $V_{无效}\leqslant\varepsilon_{无效}$ 的叶子节点;
② if $|L_A|=0$ then
③ $N_{opt}\leftarrow$仅包含 f 的新叶子节点;
④ 将 N_{opt}插入到合适的上层节点中;
⑤ 向上更新父节点一直到 *root*;
⑥ else if $|L_A|=1$ then
⑦ $N_{opt}\leftarrow L_A$ 中仅有的叶子节点;
⑧ $N_{opt}\leftarrow N_{opt}\cup\{f\}$,向上更新父节点一直到 *root*;
⑨ else /* 即$|L_A|\geqslant 2$ */
⑩ $L_B\leftarrow$从 L_A 中找出满足 $V_{重合}\geqslant\varepsilon_{重合}$ 的叶子;
⑪ if $|L_B|=1$ then
⑫ $N_{opt}\leftarrow L_B$ 中仅有的叶子节点;
⑬ $N_{opt}\leftarrow N_{opt}\cup\{f\}$,向上更新父节点一直到 *root*;
⑭ else if $|L_B|=0$ then
⑮ $N_{opt}\leftarrow L_A$ 中 $V_{增加}$最小的叶子节点;
⑯ $N_{opt}\leftarrow N_{opt}\cup\{f\}$,向上更新父节点一直到 *root*;
⑰ else /* 即$|L_B|\geqslant 2$ */
⑱ 将 f 放入等待队列 W 中;
⑲ end if
⑳ end if

节点分裂方法:在使用算法 2 时需要注意,当新 FOV(或新孩子节点)插入到叶子节点(或中间节点)

① $|\cdot|$表示集合中元素的数量,如$|L_A|$表示集合 L_A 中元素的数量.

N 时,N 可能会溢出.我们处理溢出的方法是将 N 分裂为 2 个新节点 N_1 和 N_2.分裂的方法为:首先,我们从 N 的孩子中选取 2 个关键孩子 c_1 和 c_2,它们能够成为关键孩子的原因是包围它们的 k^* 多边形面积最大.关键孩子 c_1 和 c_2 分别成为 N_1 和 N_2 的第 1 个孩子.然后,我们将剩余的孩子分配给 N_1 和 N_2,分配规则是每次选择 1 个 $V_{增加}$ 最小的孩子放入 N_1(或 N_2)中.在此过程中,我们需要确保 N_1 和 N_2 的孩子数目达到 $M_{\min}$.算法 3 展示了将 N 分裂为 N_1 和 N_2 的过程.

算法 3. 节点分裂算法.

输入:溢出节点 N;

输出:分裂后的节点 N_1 和 N_2.

① $c_1,c_2 \leftarrow N$ 的 2 个关键孩子;
② $N_1 \leftarrow N_1 \cup \{c_1\}$;
③ $N_2 \leftarrow N_2 \cup \{c_2\}$;
④ while $|N_1|<M_{\min}$或$|N_2|<M_{\min}$ do
　/* $|N_1|$和$|N_2|$表示 N_1 和 N_2 的孩子数目 */
⑤　if $|N_1|<M_{\min}$ then
⑥　　$c_i \leftarrow N-N_1-N_2$ 中让 N_1 的 $V_{增加}$ 最小的孩子;
⑦　　$N_1 \leftarrow N_1 \cup \{c_i\}$;
⑧　end if
⑨　if $|N_2|<M_{\min}$ then
⑩　　$c_j \leftarrow N-N_1-N_2$ 中让 N_2 的 $V_{增加}$ 最小的孩子;
⑪　　$N_2 \leftarrow N_2 \cup \{c_j\}$;
⑫　end if
⑬ end while
⑭ for c_t in $N-N_1-N_2$ do
⑮　if $V_{增加}(N_1,c_t)>V_{增加}(N_2,c_t)$ then
⑯　　$N_2 \leftarrow N_2 \cup \{c_t\}$;
⑰　else
⑱　　$N_1 \leftarrow N_1 \cup \{c_t\}$;
⑲　end if
⑳ end for

等待队列的生成:如果 f 目前不适插入到任何一个叶子中,算法 2 将其放入等待队列 W 中(行⑱).等待队列 W 中的元素为 FOV 的集合,等到时机成熟时,它们会作为叶子插入到凸多边形树中.当我们要将 f 放入 W 时,如果 W 非空,则遍历 W 找出最佳元素 E_{opt} 并将 f 并入 E_{opt} 中,即 $E_{opt} \leftarrow E_{opt} \cup \{f\}$;如果 W 为空或没有找到最佳元素,则直接将 $\{f\}$ 放入 W 中.衡量最佳元素的标准是在 f 并入到此元素后,使得 $V_{无效} \leqslant \varepsilon_{无效}$.如果符合条件的元素超过 1 个,则从中选取 $V_{增加}$ 最小的元素并将 f 并入该元素.算法 4 展示了生成等待队列 W 的过程.

算法 4. 等待队列生成算法.

输入:FOV f、等待队列 W;

输出:更新之后的 W.

① if $|W|\neq 0$ then
②　$L_C \leftarrow W$ 中满足 $V_{无效} \leqslant \varepsilon_{无效}$ 的元素;
③　if $|L_C|=0$ then /* 没有找到最佳元素 */
④　　$W \leftarrow W \cup \{f\}$;
⑤　else if $|L_C|=1$ then
　　/* 找到 1 个最佳元素 */
⑥　　$E_{opt} \leftarrow L_C$ 中仅有的元素;
⑦　　$E_{opt} \leftarrow E_{opt} \cup \{f\}$;
⑧　else /* 找到多个最佳元素 */
⑨　　$E_{opt} \leftarrow L_C$ 中 $V_{增加}$ 最小的元素;
⑩　　$E_{opt} \leftarrow E_{opt} \cup \{f\}$;
⑪　end if
⑫ else /* W 为空 */
⑬　$W \leftarrow W \cup \{f\}$;
⑭ end if

当 W 中的元素数目达到 $M_{\max}$ 时,我们将 W 中存储的所有元素作为叶子重新插入到树中.如果元素中仅有 1 个 FOV f_i,则遍历凸多边形树找出最优叶子节点 N_{opt} 并将 f_i 插入 N_{opt};如果元素中包含多个 FOV,则生成对应的叶子节点 L_i,并将其插入到合适的上层节点中.

4　查询处理

本节介绍凸多边形树上的 FOV 查询处理算法.从根节点开始向下搜索可能存在结果的子树,即只有当孩子节点与查询范围 Q 有交集时才下降到此节点继续搜索.当下降到某个叶子节点 L 时,检查 L 包含的每个 FOV(扇形)是否与 Q 存在交集.

函数 1. FOV 查询函数 $FovQuery(N,Q)$.

输入:凸多边形树的节点 N、查询框 Q;

输出:与 Q 有交集的 FOV.

① if N 不是叶子节点 then
②　for N 的每个孩子节点 c_i do
③　　if $c_i \cap Q$ 不为空 then
④　　　$FovQuery(c_i,Q)$; /* 递归调用 */
⑤　　end if

```
⑥    end for
⑦ else /* N 是叶子节点 */
⑧    for N 的每个 FOV f_i do
⑨      if f_i ∩ Q 不为空 then
⑩        输出 f_i; /* 输出结果 */
⑪      end if
⑫    end for
⑬ end if
```

函数 1 描述了 FOV 查询的具体过程，即函数 *FovQuery*().我们采用递归的方式对凸多边形树进行深度优先遍历，在主程序中传入函数 *FovQuery*() 的参数是 *root* 和 Q，其中 *root* 是凸多边形树的根节点.执行函数 *FovQuery*()时，首先判断节点 N 是否为叶子节点(行①).如果不是叶子节点，则将与 Q 有交集的孩子节点作为参数递归地调用函数(行②～⑥).如果是叶子节点，则判断节点中的每个 FOV 是否与 Q 有交集(行⑨)，并输出有交集的 FOV(行⑩).

我们提出角度分区筛选算法以便快速地找出与 Q 有交集的 FOV(行⑨).算法根据 Q 的位置将空间划分为 9 个区域，如图 8 所示.如果 FOV 的圆心位于 Q 之内，则其肯定与 Q 有交集.如果 FOV 圆心位于其他分区内，则判断两式是否同时成立：

$$MinDist(loc,Q)<r. \tag{4}$$

$$[\alpha_b,\alpha_e]\cap[\alpha_{qb},\alpha_{qe}]\neq\varnothing. \tag{5}$$

式(4)中 $MinDist(loc,Q)$ 是圆心 loc 到 Q 的最小距离，r 是 FOV 的半径.如果式(4)不成立，则 FOV 离 Q 过远，不可能覆盖到 Q.式(5)中，$[\alpha_b,\alpha_e]$ 是 FOV 的方向范围，$[\alpha_{qb},\alpha_{qe}]$ 是 Q 相对于 loc 的方向范围(如图 8 所示).如果式(5)不成立，则 FOV 的方向范围偏离了 Q 相对于 loc 的方向，FOV 无法覆盖到 Q.如果式(4)和式(5)同时成立，则继续判断

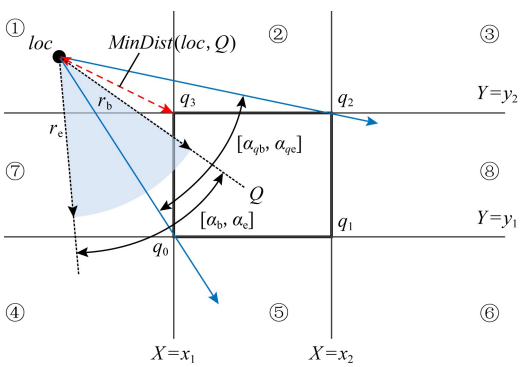


Fig. 8 Angle partition filtering method

图 8 角度分区筛选方法

FOV 的半径边 r_b 或 r_e 是否与 Q 距离 loc 较近的边有交点，如果有交点，则判定此 FOV 与 Q 有交集.

证明. 1)若 $MinDist(loc,Q)\geqslant r$，无论 $[\alpha_b,\alpha_e]\cap[\alpha_{qb},\alpha_{qe}]$ 是否为 $\varnothing$，FOV 一定不会与 Q 相交，如图 9 中 f_1 所示；2)若 $[\alpha_b,\alpha_e]\cap[\alpha_{qb},\alpha_{qe}]=\varnothing$，无论 $MinDist(loc,Q)$ 是否小于 r，FOV 一定不会与 Q 相交，如图 9 中 f_2 所示；3)若 $MinDist(loc,Q)<r$，$[\alpha_b,\alpha_e]\cap[\alpha_{qb},\alpha_{qe}]\neq\varnothing$，则 FOV 可能与 Q 相交，需要进一步判断，如图 9 中的 f_3 所示.所以，当式(4)和式(5)不能同时满足时，FOV 与 Q 一定不相交.

证毕.

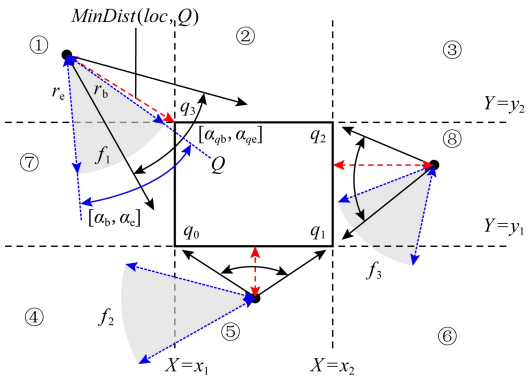


Fig. 9 Proof of the angle partition filtering method

图 9 角度分区筛选方法的证明

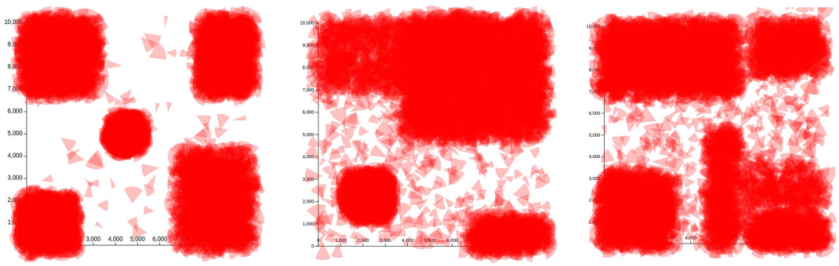
5 实 验

本节提出了生成仿真数据集的方法，并利用此数据集进行了算法性能验证实验.在实验中对比了凸多边形树、R* 树和 OR 树的构建效率，并且对比了这 3 种索引对 FOV 查询性能的影响.此外，我们展示和分析了在真实数据集上的查询结果.

5.1 实验环境设置和仿真数据集生成

实验平台是 Intel® Core™ i5-8300H CPU (2.30 GHz)，8 GB 内存，Win10 专业版操作系统.实验中的所有算法均使用 Java 语言实现.实验中测试了节点扇出值等因素对索引构建和查询性能的影响.实验中使用了 3 种数据集：1)均匀分布的数据集，即 FOV 的圆心在空间中随机分布，FOV 的数量为 10^3，10^4，10^5，依次令 S_1，S_2，S_3 表示这 3 个均匀分布数据集.2)仿真数据集，为了模拟真实生活中的拍照行为，我们根据常用拍照设备的光学参数生成了大量符合实际情况的 FOV.在二维空间中摆放这些 FOV 时，我们让其圆心(即拍摄位置)聚集在某些热门区域并且零散分布在冷门区域，以此来模拟现实生活中热门景点的照片较多而冷门区域的照片

较少的现象.图 10 展示了实验中使用的 3 个仿真数据集 D_1,D_2,D_3,每个仿真数据集中 FOV 数量都是 10^4.3)真实数据集,我们使用 iPhone 6s Plus 实地拍摄了 100 张照片用于展示查询效果.均匀分布数据集和仿真数据集用于测试索引构建和 FOV 查询的性能.由于互联网上没有大量的带地理信息的标准影像数据集,所以我们手工拍摄的少量照片仅用于分析和验证 FOV 查询的结果.

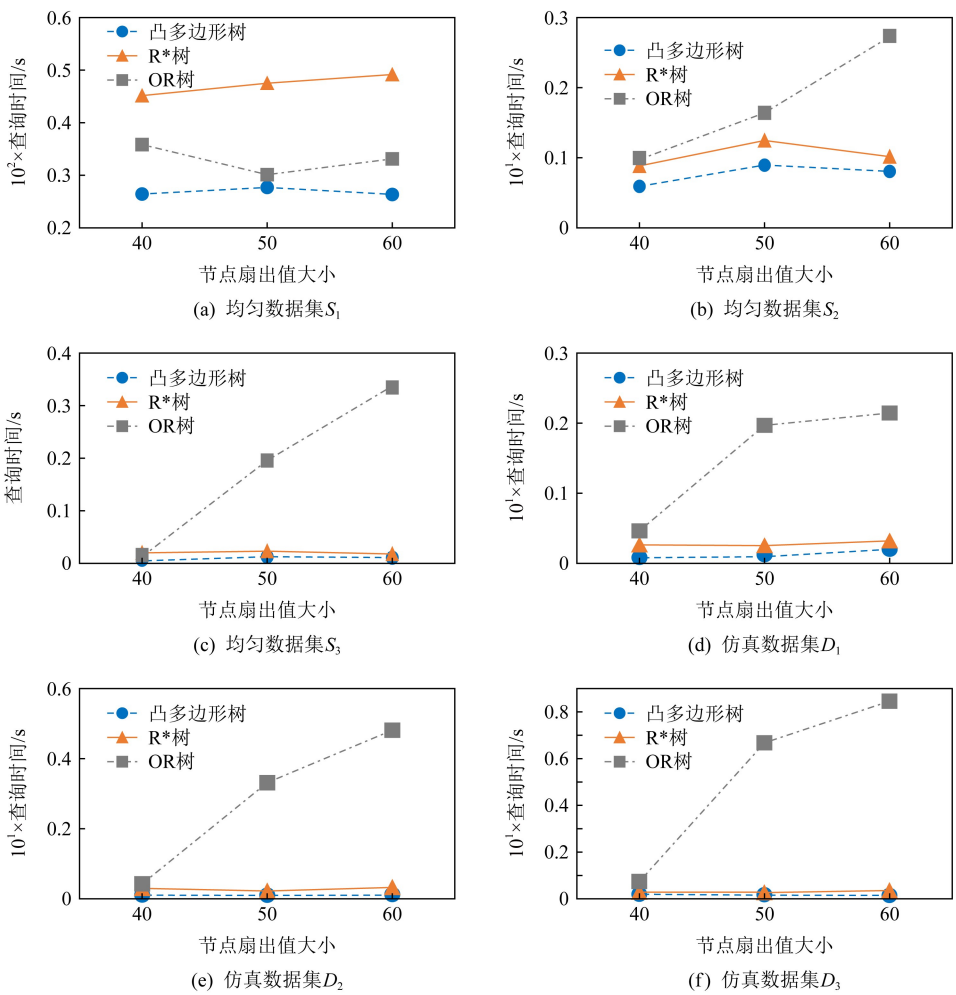


(a) D_1　(b) D_2　(c) D_3

Fig. 10　Three simulation datasets generated by mobile phone photography

图 10　通过手机拍照生成的 3 个仿真数据集

在制作仿真数据集时,我们调查了目前国内主流智能手机的光学参数,使用 Ay 等人[1]提出的方法计算出智能手机镜头的最大可视角度 α 范围为[20°,80°],最大可视距离 r 为[200,400](单位:m).在生成仿真 FOV 时,我们从这 2 个区间中随机选取 α 和 r 的值,镜头朝向 θ 从[0°,360°)中随机选取.



(a) 均匀数据集S_1　(b) 均匀数据集S_2

(c) 均匀数据集S_3　(d) 仿真数据集D_1

(e) 仿真数据集D_2　(f) 仿真数据集D_3

Fig. 11　Time cost of FOV queries

图 11　FOV 查询所花费的时间

我们在空间中随机生成了若干不相交的矩形作为热门区域的边界,让 $h\%$ 的 FOV 圆心落入这些热门区域中,让 $(1-h\%)$ 的 FOV 圆心落在热门区域之外,其中仿真数据集 D_1 中 $h\%$ 约为 99%,D_2 和 D_3 中 $h\%$ 约为 92%.

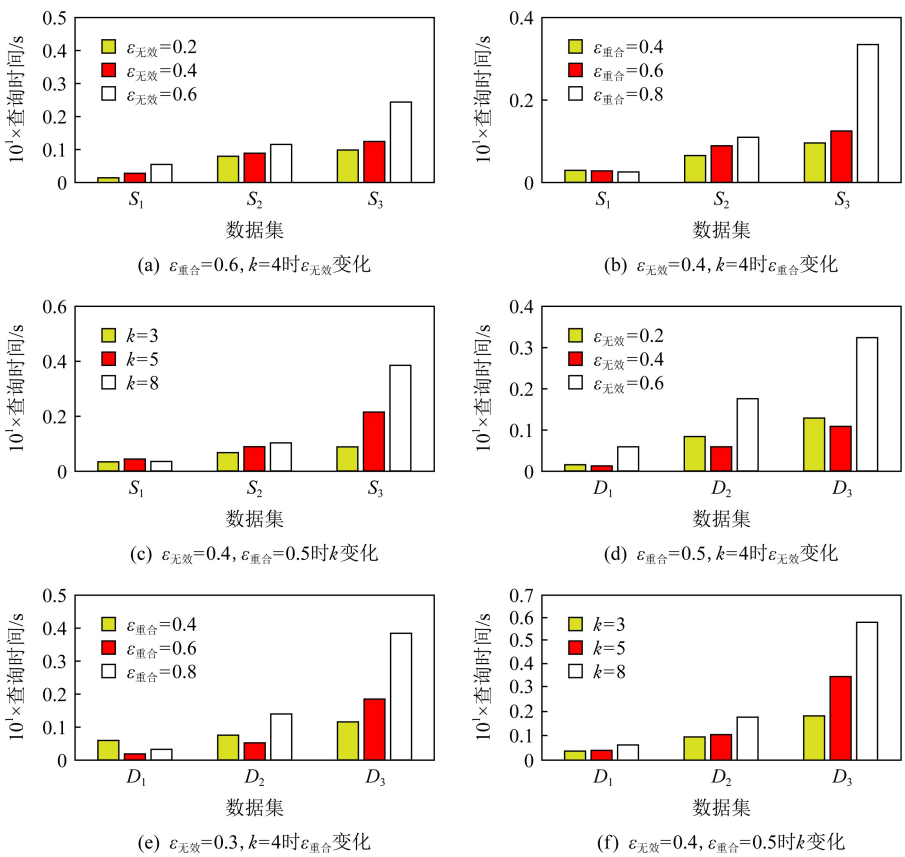
5.2 算法性能实验分析

图 11 展示了在均匀数据集和仿真数据集上改变节点扇出值 M_{max} 时,基于 3 种索引的 FOV 查询效率对比.由图 11 可见,当节点扇出值为 40 时,基于凸多边形树的 FOV 查询效率较高;无论在真实还是在仿真数据集上,当节点扇出值变化时,在凸多边形树上进行 FOV 查询比在另外 2 种树上更节省时间.

图 12 展示的是在均匀数据集和仿真数据集上,当参数 $\varepsilon_{无效}$,$\varepsilon_{重合}$ 和 k 变化时,在凸多边形树上进行 FOV 查询所花费的时间.由图 12 可见,当 $\varepsilon_{无效}$ 设置的较小时查询效率较高,因为较小的 $\varepsilon_{无效}$ 可以有效控制无效区的面积;在较大数据集上 $\varepsilon_{重合}$ 对性能的影响较明显,因为数据集越大节点重合越严重;边数 k 设置得较小时查询效率较高,因为当凸多边形边数增加时,判定凸多边形是否与 Q 相交需要花费更多的时间.

图 13 展示的是在均匀数据集和仿真数据集上,改变查询范围 Q 的大小时,利用 3 种索引进行 FOV 查询所花费的时间.在实验中,我们将 Q 的宽度设置为 500 m,Q 的长度设置为 50 m,500 m,5 000 m,从而使 Q 的大小呈 10 倍增长.由图 13 可见,对于不同大小的 Q,凸多边形均表现出较好的查询性能.这是因为凸多边形树中节点内无效区较小且节点间的重合区较小,这样可以避免访问大量的错误节点(错误节点为不包含结果的节点).

图 14 展示了当数据集大小或节点扇出值变化时,构建 R* 树、OR 树、凸多边形树所花费的时间.如图 14(a)所示,在 3 个数据集上构建 R* 树花费的时间少于构建凸多边形树和 OR 树所花费的时间;当数据集较小时,构建 3 种索引花费的时间差不多;但随着数据集增大,构建凸多边形树和 OR 树所花费的时间显著增加,构建 OR 树的时间增幅比构建凸多边形树的时间增幅要大很多.这是因为在构建 R* 树和凸多边形树时,节点的插入和分裂不需要复杂的计算,而构建 OR 树时节点插入和分裂需要大量的复杂计算并且在节点插入和分裂过程中需要动态更新节点中附加的可视距离和方向信息.如图 14



(a) $\varepsilon_{重合}$=0.6, k=4时$\varepsilon_{无效}$变化

(b) $\varepsilon_{无效}$=0.4, k=4时$\varepsilon_{重合}$变化

(c) $\varepsilon_{无效}$=0.4, $\varepsilon_{重合}$=0.5时k变化

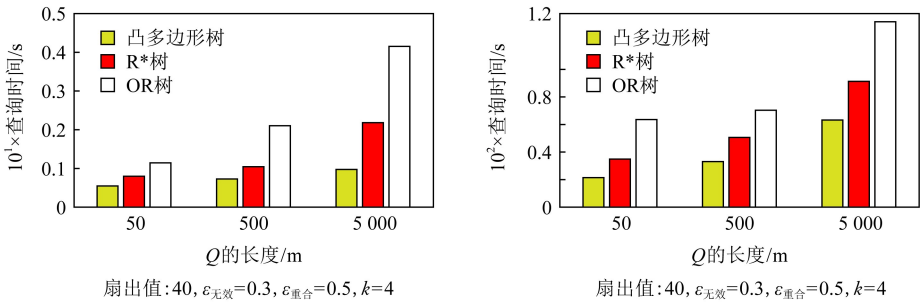
(d) $\varepsilon_{重合}$=0.5, k=4时$\varepsilon_{无效}$变化

(e) $\varepsilon_{无效}$=0.3, k=4时$\varepsilon_{重合}$变化

(f) $\varepsilon_{无效}$=0.4, $\varepsilon_{重合}$=0.5时k变化

Fig. 12 The influences of different parameters on FOV query performance

图 12 不同参数对 FOV 查询性能的影响

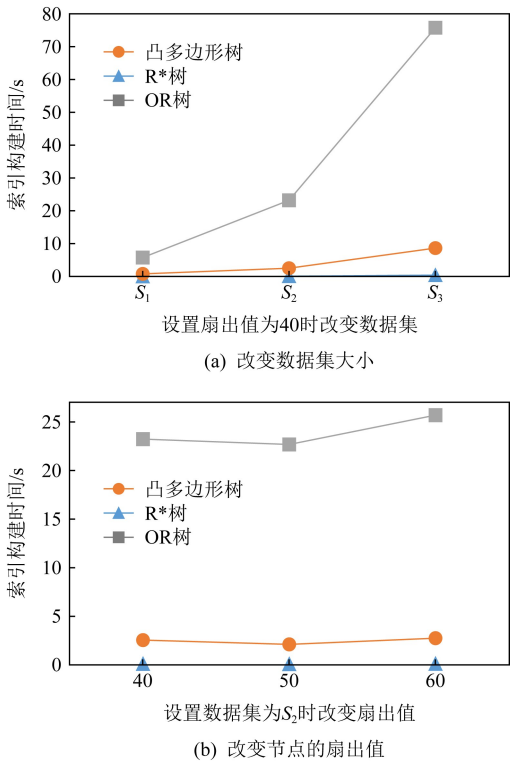


(a) 均匀数据集S_2　　(b) 仿真数据集D_2

Fig. 13　The effect of the size of Q on query performance

图 13　Q 的大小对查询性能的影响

(b)所示,当节点扇出值变化时,凸多边形树的构建时间与 R* 树的构建时间相近,R* 树的构建时间最少,OR 树的构建时间最多.



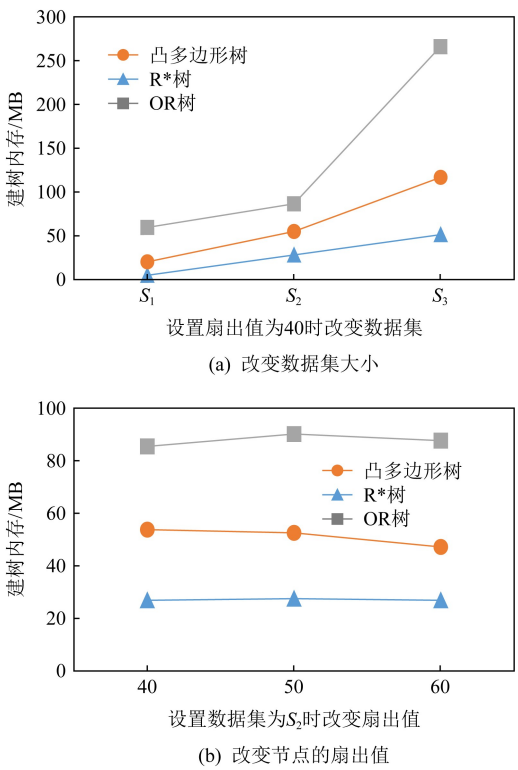
(a) 改变数据集大小

(b) 改变节点的扇出值

Fig. 14　Compare of the time cost of building indexes

图 14　对比构建索引的时间

图 15 展示了当数据集大小和节点扇出值变化时,构建 3 种索引的内存消耗情况.如图 15(a)所示,当数据集大小变化时,构建 OR 树消耗的内存最多,构建 R* 树消耗的内存最少,而构建凸多边形树消耗的内存位于两者中间.当数据集较小时,构建 3 种索引的内存消耗相似;当数据集增大时,构建 R* 树和凸多边形树所需内存的变化不明显,而构建 OR 树所需的内存明显增大.这是因为 R* 树节点中仅存储包围矩形和孩子节点指针信息,由于存储的信息较简单,所以内存消耗较小;OR 树中除了存储相机位置的包围矩形之外,还存储了节点所包含的 FOV 的可视距离和方向信息,所以内存消耗较大;凸多边形树中仅存储了节点的包围凸多边形,并且凸多边形的边数不超过 k,因此内存消耗相对较小.如图 15(b)所示,当节点扇出值变化时,构建 R* 树的内存消耗最少,构建 OR 树的内存消耗最多.



(a) 改变数据集大小

(b) 改变节点的扇出值

Fig. 15　Memory consumption when building indexes

图 15　构建索引的内存消耗情况

5.3　实验结果展示与分析

实验中用到的真实数据集是使用智能手机采集的,包含 100 张照片.由于实验中使用的智能手机仅

能将拍摄位置和光学参数嵌入到照片文件中,而不能获取拍摄角度,所以拍摄角度是使用手机中自带的指南针记录的.图 16 中的气球标出了部分拍摄位置.

如图 16 所示,用户在地图上圈定了查询范围 Q_1,Q_2 和 Q_3,系统要找出拍摄到这 3 个区域照片.如图 17(a)~(d)为拍摄到区域 Q_1 的照片,图 17(e)(f)为拍到区域 Q_2 的照片,图 17(g)~(j)为拍到区域 Q_3 的照片.图 16 中的被圆圈圈出的气球标出了查询结果的拍摄位置.以 Q_3 为例,虽然照片 No.15 的拍摄位置离 Q_3 较远,但它的 FOV 覆盖到了 Q_3,使其成为查询结果.虽然照片 No.42 的拍摄位置比 No.15 的拍摄位置更近,但由于拍摄时没有朝向 Q_3,它的 FOV 未能覆盖到 Q_3,所以它没有成为查询结果.

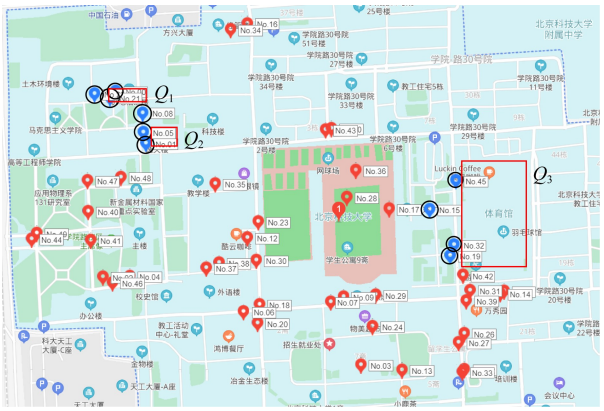


Fig. 16 Query regions on real datasets

图 16 真实数据集上的查询区域

(a) No.00

(b) No.08

(c) No.21

(d) No.25

(e) No.01

(f) No.05

(g) No.15

(h) No.19

(i) No.32

(j) No.45

Fig. 17 Query results on real datasets

图 17 真实数据集上的查询结果

6 结束语

本文提出了索引 FOV(扇形)的凸多边形树,它的节点形状为 k^* 多边形.为了构建 k^* 多边形,我们提出了淹没算法.构建凸多边形树时,我们使用将 FOV 逐个插入的方法.为了选择最优叶子节点将 FOV 插入,我们提出了根据无效区域大小($V_{\text{无效}}$)、重合区域大小($V_{\text{重合}}$)、增加区域大小($V_{\text{增加}}$)三个因素选择叶子节点的方法,并提出了将不适合插入的 FOV 暂存入等待队列中的方法.同时,我们也提出了在凸多边形树上进行 FOV 查询的算法,并在均匀数据集和仿真数据集上对算法进行了实验和性能分析.

作者贡献声明:苗雪、郭茜提出研究问题并设计研究方案,还进一步负责起草论文,苗雪同时负责收集、处理数据并进行实验验证;王昭顺、谢永红负责最终版本的修订.

参　考　文　献

[1] Ay S A, Zimmermann R, Kim S H. Viewable scene modeling for geospatial video search [C] //Proc of the 16th ACM Int Conf on Multimedia. New York: ACM, 2008: 309-318

[2] Zhou X, Huang T. Unifying keywords and visual contents in image retrieval [J]. Multimedia IEEE, 2002, 9(2): 23-33

[3] Jin Hai, He Ruhan, Tao Wenbing, et al. Vast: Automatically combining keywords and visual features for Web image retrieval [C] //Proc of the Int Conf on Advanced Communication Technology. Piscataway, NJ: IEEE, 2008: 2188-2193

[4] Liu Shenglan, Feng Lin, Sun Muxin, et al. Group and rank fusion for accurate image retrieval [J]. Journal of Computer Research and Development, 2017, 54(5): 1067-1076 (in Chinese)
(刘胜蓝, 冯林, 孙木鑫, 等. 分组排序多特征融合的图像检索方法[J]. 计算机研究与发展, 2017, 54(5): 1067-1076)

[5] Guttman A. R-trees: A dynamic index structure for spatial searching [C] //Proc of the ACM SIGMOD Int Conf on Data Management. New York: ACM, 1984: 47-57

[6] Kim Y W, Kim J, Yu H. GeoTree: Using spatial information for georeferenced video search [J]. Knowledge-Based Systems, 2014, 61: 1-12

[7] Lee Donghua, Oh J, Loh W K, et al. Geovideoindex: Indexing for georeferenced videos [J]. Information Sciences, 2016, 374: 210-223

[8] Lu Ying, Shahabi C, Kim S H. Efficient indexing and retrieval of large-scale geo-tagged video databases [J]. Geoinformatica, 2016, 20(4): 829-857

[9] Ma He, Ay S A, Zimmermann R, et al. Large-scale geo-tagged video indexing and queries [J]. Geoinformatica, 2014, 18(4): 671-697

[10] Fritze H, Degbelo A, Brüggentisch T, et al. Feature-centric ranking algorithms for georeferenced video search [C] //Proc of the 25th ACM SIGSPATIAL Int Conf on Advances in Geographic Information Systems. New York: ACM, 2017: 340-349

[11] Lu Ying, Shahabi C. Efficientindexing and querying of geo-tagged aerial videos [C] //Proc of the 25th ACM SIGSPATIAL Int Conf on Advances in Geographic Information Systems. New York: ACM, 2017: 141-150

[12] Liu Shuo, Li Hongguang, Yuan Yongxian, et al. A method for UAV real-time image simulation based on hierarchical degradation model [J]. Foundations of Intelligent Systems, 2014, 277: 221-232

[13] Kim S H, Alfarrarjeh A, Constantinou G, et al. TVDP: Translational visual data platform for smart cities [C] //Proc of the 35th IEEE Int Conf on Data Engineering Workshops (ICDEW). Piscataway, NJ: IEEE, 2019: 45-52

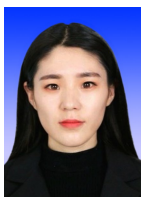
[14] Toyama K, Logan R, Roseway A. Geographic location tags on digital images [C] //Proc of the 11th ACM Int Conf on Multimedia. New York: ACM, 2003: 156-166

[15] Li Haojie, Tang Jinhui, Wang Yi, et al. Looking into the world on Google maps with view direction estimated photos [J]. Neurocomputing, 2012, 95: 72-77

[16] Alfarrarjeh A, Kim S H, Deshmukh A, et al. Spatial coverage measurement of geo-tagged visual data: A database approach [C] //Proc of the 4th IEEE Int Conf on Multimedia Big Data (BigMM). Piscataway, NJ: IEEE, 2018: 1-8

[17] Ay S A, Zimmermann R, Kim S H. Relevance ranking in georeferenced video search [J]. Multimedia Systems, 2010, 16(2): 105-125

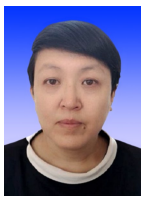
[18] Alfarrarjeh A, Shahabi C. Hybrid indexes to expedite spatial-visual search [C] //Proc of the Thematic Workshops of ACM Multimedia. New York: ACM, 2017: 75-83



Miao Xue, born in 1992. PhD candidate. Student member of CCF. Her main research interest is spatial databases.
苗　雪,1992 年生.博士研究生,CCF 学生会员.主要研究方向为空间数据库.

Guo Xi, born in 1983. PhD, associate professor. Member of CCF. Her main research interest is spatial databases.
郭　茜,1983 年生.博士,副教授,CCF 会员.主要研究方向为空间数据库.

Wang Zhaoshun, born in 1969. PhD, professor. His main research interests include software engineering, software testing, information security, and ASIC chip design.
王昭顺,1969 年生.博士,教授.主要研究方向为软件工程、软件测试、信息安全、ASIC 芯片设计.



Xie Yonghong, born in 1970. PhD, associate professor. Her main research interests include knowledge discovery, intelligent system, and database theory.
谢永红,1970 年生.博士,副教授.主要研究方向为知识发现、智能系统和数据库理论.