Available online at www.sciencedirect.com

ScienceDirect

Data & Knowledge Engineering 64 (2008) 365–380

DATA & KNOWLEDGE ENGINEERING

www.elsevier.com/locate/datak

Efficient and compact indexing structure for processing of spatial queries in line-based databases [☆]

Hung-Yi Lin [*]

Department of Logistics Engineering and Management, National Taichung Institute of Technology, 129, Sanmin Rd., Sec. 3, Taichung, Taiwan, ROC

Received 8 August 2006; received in revised form 3 August 2007; accepted 4 September 2007
Available online 22 September 2007

Abstract

Points, lines and regions are the three basic entities for constituting vector-based objects in spatial databases. Many indexing schemes have been widely discussed for handling point or region data. These traditional schemes can efficiently organize point or region objects in a space into a hashing or hierarchical directory, and they provide efficient access methods for accurate retrievals. However, two difficulties arise when applying such methods to line segments: (1) the spatial information of line segments may not be precisely expressed in terms of that of points and/or regions, and (2) traditional methods for handling line segments can generate a large amount of dead space and overlapping areas in internal and external nodes in the hierarchical directory. The first problem impedes high-quality spatial conservation of line segments in a line-based database, while the second degrades the system performance over time. This study develops a novel indexing structure of line segments based on compressed B^+ trees. The proposed method significantly improves the time and space efficiencies over that of the *R*-tree indexing scheme.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Spatial database; Line segments; Indexing structure; GIS; Compressed B^+-tree; *R*-tree

1. Introduction

Large spatial databases have been extensively adopted in the recent decade, and various methods [3,8,19,21] have been presented to store, browse, search and retrieve spatial objects. A good spatial database can preserve and arrange explicit and implicit information of spatial objects. Explicit information of an object includes its location, extent, orientation, size and circumference. Implicit information includes the spatial relationship between distinct objects, the distribution and density of objects in a specific area and the coverage for some objects. Gaede and Gunther [4] classified spatial objects in *d*-dimensional Euclidean space (E^d) into $d+1$ types. For each k $(0 \leqslant k \leqslant d)$, the set of *k*-dimensional polyhedra forms a data type. For instance, a

[☆] This research was supported by Nation Science Council of ROC under Grant 96-2221-E-025-013.
[*] Tel.: +886 4 2219 6769; fax: +886 4 2219 6161.
E-mail address: linhy@ntit.edu.tw

0169-023X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2007.09.009

two-dimensional graph may contain a zero-dimensional polyhedron (points), a one-dimensional polyhedron (lines, polylines) and a two-dimensional polyhedron (regions, polygons). An efficient spatial database has a well-organized indexing structure, and enables the easy retrieval interested objects for user queries.

The construction of spatial databases differs from that of traditional databases in many respects. First, the data model representing spatial objects must be determined in advance of processing an index scheme. A spatial object may be a single point, a line segment, a curve, a polygonal segment, a 2D polygon or a multidimensional polygon, its spatial information needs to be preserved precisely. Second, insertions and deletions are interleaved with updates, since spatial objects are often dynamic. The data structures adopted in this context need to support this dynamic behavior without deteriorating over time. Third, spatial databases tend to be large, making the integration of secondary and tertiary memory essential for efficient processing.

Two-dimensional objects can be categorized according to their space occupancies. Points with zero spatial occupancy are generally depicted by explicit coordinates, which are handled and queried by many traditional methods. Polygons, circles, ellipses and rectangles are regional data with nonzero spatial occupancy, and are generally depicted by rectangular objects, which are also indexed and queried accordingly by many traditional methods. A line segment does not enclose any area, so cannot be categorized as a nonzero-size object. However, in many previous studies [9,16,22], line segments are enclosed and represented by grids, cells, rectangles or *MBR*s (minimum bounding rectangles). Such methods represent line segments using nonzero-size objects and the corresponding index entries include much redundant information in their representations. Such redundancy at leaf level in the hierarchical directory propagates upward to the root and aggravates this redundant condition at higher levels. The resulting indexing structure suffers from many problems, such as the heavy building overhead cost of index structures, poor system execution performance, low retrieval accuracy and the incapability of processing certain types of query.

The rest of this paper is organized as follows. Section 2 reviews pertinent literature on point and region indexing methods. The inefficiency of the traditional methods for handling line objects is then addressed. Section 3 then presents a variant of B^+-tree to facilitate line indexing. Section 4 describes the construction of an indexing structure for line segments, and demonstrates the process by an example. Section 5 presents the algorithms for insertion and deletion of a line segment, and three query processing methods. Section 5 also analyzes the time complexity for all query processes. Section 6 discusses the performance of the proposed indexing structures. Section 7 summarizes the experimental results of a GIS application, in which the storage requirement and retrieval performance of the proposed system are compared with those of the *R*-tree indexing scheme. Conclusions are finally drawn in Section 8.

2. Overview of previous methods

Many methods for handling point and region data have been proposed during the last two decades. The grid file [13] and its variants [2,20] adopt the point access method based on hashing. The *KD*-tree [1] is a binary search tree that stores points in k-dimensional space, and at each intermediate node, the *KD*-tree divides the k-dimensional space in two parts by a $(k-1)$-dimensional hyperplane. The *K–D–B* tree [17] and the *G*-tree [10] are the typical structures for indexing point data. Grid files proposed in [13] can handle spatial objects with points or regions. The *R*-tree proposed by Guttman [5] is probably the most popular structure for indexing nonzero-size objects. An *R*-tree adopts *MBR*s to enclose nonzero-size objects, and then represent them as indexed entities. *R*-trees have been widely employed to index the spatial objects in a large pictorial database such as the GIS application [15].

Although the above-mentioned structures can efficiently process a large number of point objects, uniform grid data and non-uniform *MBR* data, they do not index line segments well. Applying these structures to line segments causes three major problems. First, using only the endpoints or some specific points of a line does not preserve the full spatial information, since a line comprises infinitely many points, and can involve a wide space. Second, adopting *MBR*s for slender or winding objects can easily introduce dead space. Dead space is the redundant space outside an object and inside its *MBR*. Such redundant space easily leads to serious overlaps between *MBR*s at each level in an indexing tree. Fig. 1 shows MBR_1 and MBR_2 enclosing l_1 and l_2, respectively. Lines l_1 and l_2 do not intersect, yet MBR_1 and MBR_2 overlap each other. Third, various spatial relationships exist among line segments. For instance, a line may have no joint with others; a line may be a