K-Order Neighbor: the Efficient Implementation Strategy for Restricting Cascaded Update in Realm[1]

Yong Zhang[1], Lizhu Zhou[1], Jun Chen[2], RenLiang Zhao[2]

[1] Department of Computer Science and Technology, Tsinghua University
Beijing, P.R.China, 100084
zhangy97@mails.tsinghua.edu.cn
[2] National Geomatics Center of China
Beijing, P.R.China, 100044
chenjun@nsdi.gov.cn

Abstract. A realm is a planar graph over a finite resolution grid that has been proposed as a means of overcoming problems of numerical robustness and topological correctness in spatial database. One of the main problems of realm is cascaded update. Furthermore, cascaded update causes heavy storage and complex management of transaction. Virtual realm partially resolves the problem of space overhead by computing the portion of realm dynamically. K-order neighbor is a concept commonly used in Delaunary triangulation network. We use K-order neighbor in the Voronoi diagram of realm objects to restrict cascaded update. Two main update operations – point insertion and segment insertion are discussed. In point insertion, the distortion caused by cascaded update is restricted to 1-order neighbor of the point. In segment insertion, two end points of the segment are treated specially. This strategy can be used in both stored realm and virtual realm.

1 Introduction

A realm [3] [4] is a planar graph over a finite resolution grid that has been proposed as a means of overcoming problems of numerical robustness and topological correctness in spatial database. These problems arise from the finite accuracy of number in computer. In realm based spatial database, the intersections between spatial objects are explicitly represented in insertion/update, can be modified slightly if necessary.

One of the main problems of realm is cascaded update [6] [7], that is, the update of some spatial object can modify the value of spatial objects previously stored. Furthermore, cascaded update causes heavy storage and complex transactions.

ROSE algebra [4] [5] is a collection of spatial data types (such as points, lines and regions) and operations. It supports a wide range of operations on spatial data type,

[1] This paper is supported by Natural Science Foundation of China (NSFC) under the grant number 69833010.

P.M.A. Sloot et al. (Eds.): ICCS 2002, LNCS 2331, pp. 994–1003, 2002.

and has been designed in the way that all these operations are closed. In this paper, we call the value of spatial data type as spatial object.

In the implementation in [3] [4], realm is stored explicitly (called as stored realm). Stored realm is organized as a separated layer and has spatial index. Virtual realm [8] stored realm objects within spatial objects. It partially resolves the heavy storage problem. However, both approaches do not resolve the problem of cascaded update.

K-order neighbor [9] is a concept commonly used in Delaunary triangulation network. We use K-order neighbor in Voronoi diagram of realm objects to restrict cascaded update. Two main update operations – point insertion and segment insertion are discussed. In point insertion, the distortion caused by cascaded update is restricted to 1-order neighbor of the point. In segment insertion, two end points of the segment are treated specially. This strategy can be used in both stored realm and virtual realm.

This paper is organized as follows: in section 2, we describe redrawing in data update, and concepts of stored realm and virtual realm; in section 3, K-order neighbor is described; in section 4, we apply K-order neighbor into realm; section 5 gives a comparison; and the last section is conclusion.

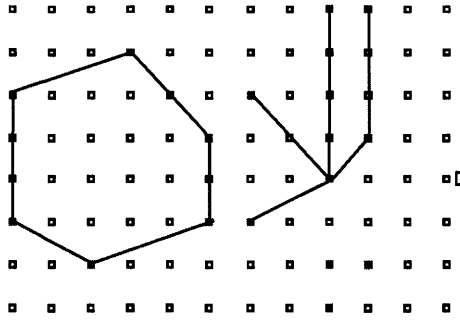

Fig. 1. The example of realm

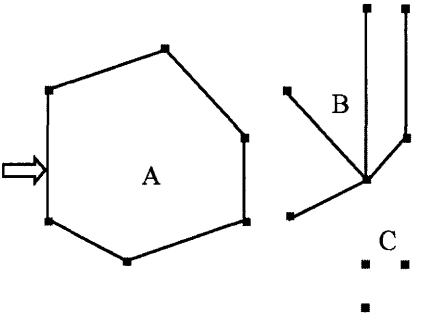

Fig. 2. Spatial objects built from realm

2 Basic Concepts in Realm

Fig. 1 shows an example of realm that is a set of points and non-intersecting segments in the finite resolution grid. (Here we call the point in realm as point, and the segment in realm as segment.) In applications, all spatial objects take these points and segments as elements. Fig. 2 shows a regions object A, a lines object B and a points object C, all these objects can be constructed from the points and segments in Fig. 1.

One of the main problems of realm is cascaded update. A point or segment inserted can modify the points and segments in database (to guarantee numerical robustness and topological correctness), that is, data update is cascaded. During this procedure, many segments have to be redrawn. So many segments are generated. At the same time, too many segments needed redrawing make the strategy of locking very complex; therefore it is difficult to manage the transactions.

In the following, we explain redrawing in data update, and the basic concepts of stored realm and virtual realm.

2.1 Redrawing in Data Update

Redrawing [1] of segment is the source of cascaded update. After redrawing, one segment is divided into several segments. The idea is to define for a segment s an envelope E(s) roughly as the collection of points that are immediately above, below, or on s. An intersection point between s and other segment may lead to a requirement that s should pass through some point P on its envelope. This requirement is then fulfilled by redrawing s by some polygonal line within the envelope rather than by simply connecting P with the start and end points of s. Fig. 3 shows that P lies on the envelope of AB; after redrawing segment AB is divided into AQ, QP and PB rather than AP and PB.

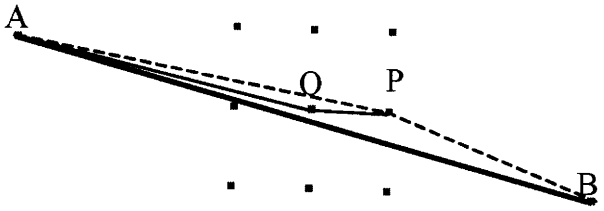

Fig. 3. Redrawing of segment AB passing through point P (Modified from [3])

This approach guarantees that the polygonal line describing a segment always remains within the envelope of the original segment. In realm it then means that a segment cannot move to the other side of a point. [3] extended envelope to "proper envelope" that is the subset of envelope points that are not end points of segments (denoted as $\overline{E}(s)$ for segment s). [3] also added an integrity rule for points and segments that are very close to each other:

No (R-)point lies on the proper envelope of any (R-)segment.

In the worst case, the number of redrawn segments of one segment is logarithmic in the size of the grid. At the same time, if a point or segment is inserted into an area with a high concentration of points and segments, there may be many segments needed redrawing. Therefore, we must decrease the segments needed redrawing as few as possible.

2.2 Stored Realm and Virtual Realm

In stored realm [3] [5], there exist bi-links between spatial objects and realm objects. The realm objects compose a single layer and have spatial index; the spatial objects compose another layer and also have spatial index. The bi-links between realm objects

and spatial objects are redundancy, because we can use the pointers in one direction to get the pointers in the contrary direction by traveling.

The links from realm objects to spatial objects are only needed in data update. In stored realm, we firstly operate on the realm layer, and then propagate the changed realm objects to the corresponding spatial objects. There is another implementation approach – virtual realm [8]. In this approach, the realm objects are stored within spatial objects. During data update, we firstly find the spatial objects influenced, and then operate on the set of the realm objects corresponding to these spatial objects. Both approaches can get the same result, but they do not resolve the problem of cascaded update.

3 K-Order Neighbor

K-order neighbor [9] is a concept commonly used in Delaunary triangulation network. Here we define K-order neighbor using Voronoi diagram. Voronoi diagram is the partition of the space based on the neighbor relation.

Spatial neighbor is the degree of the distance between two spatial objects; it is a fuzzy spatial relation. Voronoi diagram provide a clear definition of spatial neighbor [2]: If the Voronoi regions of two objects have the common boundary, then they are defined as spatial neighbor (Fig. 4(a)).

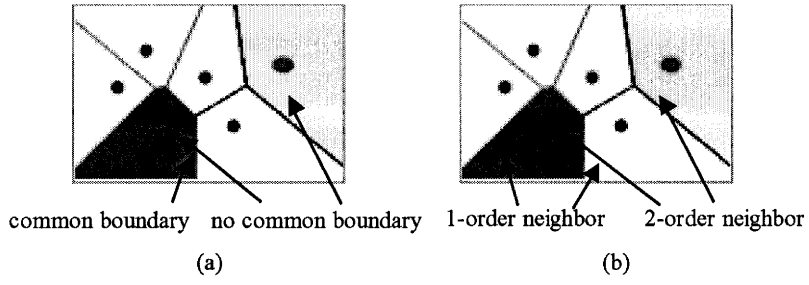

Fig. 4. Spatial neighbor and K-order neighbor

This definition only describes the relation between two objects whose Voronoi regions have common boundary, but does not consider two objects that do not have common Voronoi boundary. K-order neighbor is the extension of spatial neighbor. It is can be used to describe the relation between two objects whose Voronoi regions do not have common boundary. We give the definition using Voronoi diagram (Fig.4(b)):

(1) If the Voronoi regions of P and Q have common boundary, then P is 1-order neighbor of Q, and Q is 1-order neighbor of P.

(2) If the Voronoi region of P has the common boundary with the Voronoi region of one of the 1-order neighbors of Q, and P is not 1-order neighbor of Q, then P is 2-order neighbor of Q.

.....

(k) If the Voronoi region of P has the common boundary with the Voronoi region of one of the (k-1)-order neighbors of Q, and P is not (k-1)-order neighbor of Q, then P is k-order neighbor of Q.

4 Application of K-Order Neighbor in Realm

As Fig. 5 Shows, a point is inserted into a realm with a high concentration of segments; point p lies on the proper envelopes of s1, s2, s3 and s4. Using algorithms in [3] and [8], s1-s4 are all needed redrawing. After redrawing, four segments are divided into 10 segments. Moreover, all these segments intersect at the same point, which changes the original topological relations (separated) between them.

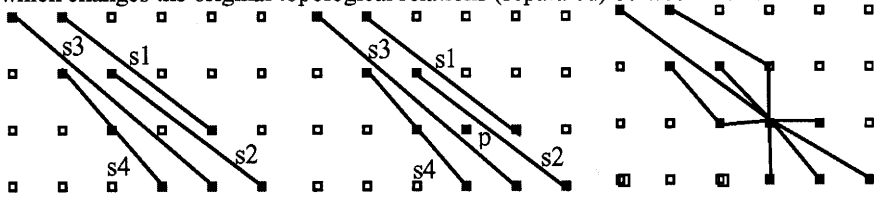

Fig. 5. Insert a point into a realm with a high concentration of segments

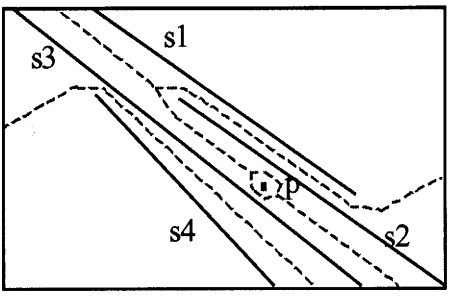

Fig. 6. The Voronoi diagram of the realm objects in Fig. 5

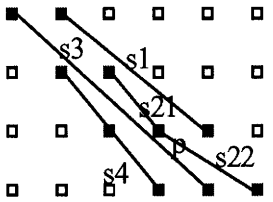

Fig. 7. Redrawing of s2 in Fig. 5

Fig. 6 shows the Voronoi diagram of p being inserted into realm. (Notice we use more precision here.) From Fig. 6 we can find that point p is 1-order neighbor of s2 and s3, and 2-order neighbor of s1 and s4. We want to restrict the influence of point p to the scope of its 1-order neighbor, that is, although s1 and s4 are also in its envelope, they are not the "nearest", so we do not need to redraw them.

Let us analyze the point insertion in Fig. 5 step by step. Because point p lies on the proper envelope of s2, we firstly redraw s2, and then s2 is divided into two segments s21 and s22 (As Fig.7 shows). Then we can find that point p has become the end point of s21 and s22, and the proper envelopes of s1, s3 and s4 have also been changed, so point p does not lie on the proper envelopes of them. Hence we do not need to redraw s1, s3 and s4. Therefore, the distortion is restricted to the scope of 1-order neighbor of point p. In practice, we do it as follows: firstly computing the local Voronoi diagram

containing point p, and then redrawing the 1-order neighbor segment of p firstly found.

There are three kinds of approaches to get the Voronoi diagram: (1) generate the Voronoi diagram dynamically; (2) store the Voronoi diagram in database; (3) select some not easily changed spatial objects, and store the Voronoi diagram of them in database, and then generate detail Voronoi diagram dynamically. The generating of Voronoi diagram is very slow, and the boundaries generated is in the size of the actual objects, so the first and second approaches are not practical, here we use the third approach.

In the above, what we considered is only the insertion of isolated point. In the algorithms of segment insertion in [3] [8], two end points are firstly inserted into realm as isolated points, and then the segment is inserted. In fact, the insertions of the end points of a segment are different from the insertion of isolated point greatly. As in Fig. 5, if point p is one of end point of a segment to be inserted, then it does not lie on any proper envelope of the segments. Here we must ensure that the insertions of two end points of a segment and the insertion itself are included in a transaction. If the end points of the segment are inserted, but the insertion of the segment fails, then we have to remove the end points from realm and recover realm to the state before insertion.

Overall, using our approach, there are two cases to decrease the segments needed redrawing:

(a) The inserted point lies on the proper envelopes of many envelops, but only one segment needs to be redrawn;

(b) The inserted point is one of the end points of a segment to be inserted, then no segment needs to be redrawn or only one segment needs to be redrawn.

Our approach can be used in both stored realm and virtual realm. In the following, we use stored realm to explain the algorithms in our approach.

4.1 Algorithm of Point Insertion

The algorithm presented in Fig. 8 for inserting a point into a realm is similar to that given in [3]. In point insertion, we have to deal with four cases:

(1) The point is in the realm (line 13).

(2) The point is new, and does not lie on any envelope (line 15).

(3) The point is in some segment; then we separate the segment into two new segments (line 17).

(4) The point lies on one or more proper envelopes (but not in any segment). Here are two conditions:

(a) The point is an end point of a segment to be inserted, and then we do nothing (line 19).

(b) The point is an isolated point, and then we select the segment firstly found in 1-order neighbor of the point to be redrawn (line 20-23).

00 Algorithm: InsertPoint(R, p, flag, R', r, SP)

```
Input: R: realm, and R = P ∪ S, P is the point set, and S is the segment set
       p: point
       flag: the type of point, 0: isolated point, 1: the end point of some segment
Output: R': modified realm
        r: realm object identifier corresponding to p
        SP: the set of identifiers of influenced spatial objects

Step 1: Initialization
SP:=∅;

Step2: Find the segment to be redrawn
If ∃q∈P: p=q; (one such point at most)
Then r:=roid(q); R' := R; return;
Else if ∀s∈S: p∉ Ē(s) (not exists, and not lie on any proper envelope)
     Then r:=roid(p); R'=R∪{(p,r,∅)}; return;
     Else if ∃ s∈S: p in S
          Then Insert a hook h=<p,p> on s;
          Else if flag = 1
               Then r:=roid(p); R'=R∪{(p,r,∅)}; return;
               Else r:=roid(p);
                    Generate the Voronoi diagram near p;
                    Search the 1-order neighbor of p in the Voronoi diagram, s
is first segment found;
                    Insert a hook h = <base(p,s),p> on s;

Step3: Redraw the segment with hook
Redraw segment s according to [1];
Let {s1, ..., sn} is the set of segments after redrawing, such that si=(qi-1,qi),
i∈{1,...,n};

Step 4: Update realm
R' := R\{s, roid(s), scids(s)};
(Insert the end points and segments in the set of segments of the redrawings, if
they do not already exist in the realm)
for each i in 0..n do
        if not ExistsPoint(qi) then R' := R' ∪ {( qi, roid(qi), ∅)};
for each i in 1..n do
```

35 if not ExistsSegment(s_i) then R':= R' ∪ {(s_i, roid(s_i), ∅)};

36 SP := {(sc, {(s_i, roid(s_i))|i∈ {1,...,n}}) | sc ∈ scids(s)};

37

38 End InsertPoint

Fig. 8. Algorithm of point insertion

4.2 Algorithm of Segment Insertion

Begin Transaction
InsertPoint(R, p, 1, R', r, SP);
InsertPoint(R, q, 1, R', r, SP);
InsertSegment(R, s, R', RD, SP) (see [3] for detail, we omit parameter "ok");
Commit Transaction

Fig. 9. The algorithm of segment insertion

Our algorithm of segment insertion is similar to those in stored realm and virtual realm. The process of inserting a segment s = (p, q) requires three steps: (1) insert p into realm; (2) insert q into realm; (3) insert s into realm. Because in point insertion, we distinguish if a point is an end point of a segment to be inserted, these three steps must be completed in one transaction (Fig. 9). Otherwise, the integrity rule of proper envelope will be violated.

5 Performance Analysis

Taking stored realm as an example, this section presents an informal comparison of the performance of point insertions using K-order neighbor and not using it.

If the distribution of spatial objects is dense, the advantage of our approach is very clear: the segments needed redrawing decrease greatly. Therefore, fewer segments are generated after redrawings. At the same time, the restriction of distortion provides a good foundation for the simplicity of transaction management.

Table 1 summarizes that tasks while inserting a point using K-order neighbor and not. The column of difference indicates whether K-order neighbor increase (+) or decrease (-) the cost of performing input-output (I/O) or processing (CPU) tasks.

Table 1. The influence of using K-order neighbor on point insertion in stored realm (modified from [8])

Stored realm	Stored realm using K-order neighbor	Difference
Retrieve required nodes of spatial index (many entries)	Retrieve required nodes of spatial index (many entries)	-I/O, -CPU

Retrieve segments and points from MBR of inserted segment (possible many)	Retrieve segments and points from MBR of inserted segment (possible many)	-I/O
	Retrieve the corresponding part of basic Voronoi diagram	+I/O
	Compute the local detail Voronoi diagram	+CPU
	Compute the modification of Voronoi diagram	+CPU
	Use Voronoi diagram to search the neighbors	+CPU
Compute changes in the realm	Compute changes in the realm	-CPU
Delete changed segments from disk	Delete changed segments from disk	- I/O
Write new segments into disk	Write new segments into disk	- I/O
Compute changes to spatial index	Compute changes to spatial index	-CPU
Write changed index to disk	Write changed index to disk	- I/O
	Write changed basic Voronoi diagram to disk (maybe not necessary)	+I/O
Retrieve the spatial objects related to the changed segments	Retrieve the spatial objects related to the changed segments	- I/O
Replace the changed segments in spatial objects	Replace the changed segments in spatial objects	- I/O
Write changed spatial objects to disk	Write changed spatial objects to disk	- I/O
	Delete the local detail Voronoi diagram	+CPU

The results presented in the table can be summarized with respect to the effect on I/O costs and CPU time:

I/O: (a) There are two factors to increase I/O activities, the reason is that we store a basic Voronoi diagram in database; (b) There are eight factors to decrease I/O activities, the reason is that the total number of points and segments in database is fewer.

CPU: (a) There are four factors to increase CPU time, the reason is that the processes related Voronoi diagram are added; (b) There are three factors to decrease CPU time, the reason is that the total number of points and segments in database is fewer.

Overall, there are two main reasons that influence I/O costs and CPU time: Voronoi diagram and the realm objects in database. It is hard to say that because of using K-order neighbor, I/O costs and CPU time are saved. The saving of I/O costs and CPU times depends on the distribution of spatial objects (i.e., the applications). However, we can conclude that using K-order neighbor decreases the realm objects and

simplifies the transaction management. Furthermore, the stored Voronoi diagram can speed many ROSE algebra operations such as *inside*, *intersection*, *closest* and so on.

6 Conclusion

This work is being carried out in the context of a project of 3D spatial database based on realm. We find that in some applications (especially the distribution of segments is dense), in spite of using stored realm or virtual realm [3] [8], point insertions and segment insertions bring out boring cascaded update. In these conditions, many segments have to be redrawn, which result many segments to occupy large storage space and make the management of transactions very complex. So we want to find an approach to restrict cascaded update.

K-order neighbor [9] is a concept commonly used in Delaunary triangulation network. We use Voronoi diagram to describe this concept. K-order neighbor restricts cascaded update efficiently. Presently we have implemented the algorithm of K-order neighbor and used it in the data update of realm.

References

1. Greene, D., Yao, F.: Finite-Resolution Computational Geometry. Proc. 27th IEEE Symp. on Foundations of Computer Science (1986) 143-152
2. Gold, C.M.: The Meaning of "Neighbour". In Frank, A. U., Campari, I. And formentini, U. (Eds) Theories of Spatial -Temporal Reasoning in Geographic Space. Lecture Notes in Computer Science 639, Berlin: Springer-Verlag (1992) 220-235
3. Guting, R.H., Schneider, M.: Realms: A Foundation for Spatial Data Type in Database Systems. In D. Abel and B.C. Ooi, editors, Proc. 3rd Int. Conf. on Large Spatial Databases (SSD), Lecture Notes in Computer Science, Springer Verlag, (1993) 14-35
4. Guting, R.H., Ridder, T., Schneider, M.: Implementation of the ROSE Algebra: Efficient Algorithms for Realm-Based Spatial Data Type. 4th Int. Symp. on Advances in Spatial Databases (SSD), LNCS951, Springer Verlag (1995) 216-239
5. Guting, R.H., Schneider, M.: Realm-Based Spatial Data Types: The Rose Algebra. VLDB Journal, Vol.4 (1995) 100-143
6. Cotelo Lema, J. A., Guting, R.H.: Dual Grid: A New Approach for Robust Spatial Algebra Implementation. FernUniversity Hagen, Informatik-Report 268 (2000)
7. Cotelo Lema, J. A.: An Analysis of Consistency Properties in Existing Spatial and Spatiotemporal Data Models. Advances in Databases and Information Systems, 5th East – European Conference ADBIS' 2001, Research Communications, A. Caplinskas, J. Eder (Eds.): Vol. 1 (2001)
8. Muller, V., Paton, N.W., Fernandesyy, A.A.A., Dinn, A., Williams, M.H.: Virtual Realms: An Efficient Implementation Strategy for Finite Resolution Spatial Data Types, In 7th International Symposium on Spatial Data Handling - SDH'96, Amsterdam (1996)
9. Zhang, C.P., Murayama, Y.J.: Testing local spatial autocorrelation using k-order neighbours. Int. J. Geographical Information Science, Vol.14, No.7, (2000) 681-692