INTEGRATION OF SPATIAL OBJECTS IN A GIS

Richard G. Newell, Mark Easterfield & David G. Theriault
Smallworld Systems Ltd,
8-9 Bridge Street,
Cambridge, England
CB2 1UA

ABSTRACT

A GIS is distinguished from other database systems in a number of respects, particularly in the requirement to handle spatial objects with extent. Whereas a common approach is to treat "geometry" and "attributes" separately, a more integrated approach is to treat the spatial aspect as but one part of an integrated data model which accommodates all objects and their attributes in a seamless manner. Spatial objects differ from other object attributes in that they usually have extent and therefore efficient retrievals cannot be achieved by the mechanisms implemented within database systems on their own. This paper addresses the problem of implementing efficient spatial retrieval methods within an integrated object data model. An improved quadtree mechanism for clustering objects on disk is also described.

INTRODUCTION

In recent years, the problem of organising large numbers of spatial objects in a database has become much better understood. It is now up to the system implementors to apply the known methods in real systems. However, although there are many algorithms described for indexing and storing spatial objects, there is little published information on how to apply the algorithms within the context of a complete integrated database system which would support a GIS. In particular, there is an almost total lack of clear descriptions of implementations within the realm of the relational model, although papers have been published which seem to get good results using this approach (Abel 1983 and 1984, Bundock 1987).

Older systems use a proprietary structure of sheets or tiles to organise their spatial data, but this leads to serious problems in large systems. A modern approach is to implement a spatial database which is logically seamless. Some systems separate out the geometric objects into a separate proprietary database which is specifically built for fast spatial retrievals, and then this is linked somehow to a database of attributes. It is our contention that this does not meet any criteria of integration. This does of course beg the question of what to do about the integration of spatially located data that is already committed to institutional databases and which needs to be

accommodated within a GIS. We have no elegant answer to this problem.

Currently available relational database systems come under a lot of criticism for use with spatial data, on account of their apparent poor performance. However, in our view, the jury is still out on this issue, as, if appropriate data models are used, then acceptable spatial performance seems to be achievable (Abel 1983 and 1984), but this depends on the use to be made of the GIS. For relational databases, there are more serious issues than this to overcome, especially the management of long transactions and versioning (Easterfield et al 1990).

It is significant however, that most commercial systems that use a commercial relational database system for holding spatial data employ a system of check out into a single user proprietary database before work starts, and they may even go one step further to employ a display file to gain adequate graphics performance.

We have been researching the implementation of fast spatial retrieval algorithms within the context of a version managed tabular database, which uses an interactive object oriented language for its development and customisation environment (Chance et al 1990, Easterfield et al 1990). Our approach has been to implement spatial retrieval methods in the system by using the normal indexing methods of the tabular database, without any ad hoc structures showing through to the system customiser and developer. We neither employ check out for reasons of handling multiple users nor do we need to employ a display file to achieve good performance.

HOW TO ACHIEVE FAST SPATIAL ACCESS

The nub of all spatial access algorithms (e.g. range trees, quadtrees, field trees, k-d trees etc) seems to be the same, in that if one can organise one's data somehow in the form of a tree structure, where each node of the tree corresponds to a subset of the area covered by its parent node, then candidate objects inside an area or surrounding a point can be found quickly. Such algorithms can retrieve an object in a time proportional to the logarithm of the number of objects in the database.

One approach is to provide an external spatial index into a database of objects which is not spatially organised. If one is retrieving many objects within say a rectangle, then the candidate objects can be identified very quickly, but retrieving the objects themselves is like using a pair of tweezers to extract each one from disk. The logarithmic behaviour still applies, but the constant term is very large because of disk seek time.

Thus, to gain the full benefit, the actual object data itself needs to be organised spatially on disk, by clustering the data, so that one can at least use a "shovel" to retrieve objects (bulk retrieval), instead of a pair of "tweezers" (random retrievals).

Certain methods of spatial indexing are structured so that each object is contained once only in the index. Other approaches

duplicate an object's entry in the index, based on sub-ranges of the total object. This has the advantage that candidate objects are more likely to be relevant, but the disadvantage that duplicates must be eliminated. (Abel 1983)

There does not seem to be a great performance difference between the many methods of indexing and clustering that have been described in the literature (Smith and Gao 1990, Kriegel et al 1990). The message is, just do anything in a tree structure and you will get most of the benefit, the rest is just tuning.

However, we are rather concerned with implementing such mechanisms within an overall integrated data model, where the peculiarities of particular methods are hidden, because if they are not, the complexity and cost of development is increased. In addition, as new spatial indexing mechanisms are discovered, these should be implementable independently of the overall data model.

SPATIAL KEYS IN TABULAR DATABASES

It is well known that it is possible to encode the size and position of an object in a unique spatial key, so that objects which are close to each other in space generate similar keys. Further, if objects with similar keys are stored at similar locations on disk, then the number of disk accesses required to retrieve objects can be greatly minimised.

Some methods lend themselves easily to the generation of a spatial key, such as a quadtree index and its many close variations (Samet 1989). However, mechanisms such as range trees preclude this approach, indeed the actual structure produced depends on the order in which the database is created.

We have not investigated methods such as range trees for clustering objects, because they are ad hoc and it seems to us difficult to hide the storage mechanisms behind an acceptable interface for system developers.

It is common in a tabular database that records with similar primary keys are close to each other on disk, especially if the fundamental storage mechanism is something like a B*tree. Thus, if the most significant part of the primary key of all spatial objects is a spatial key, then the desired effects of spatial ordering on disk can be achieved (Abel 1983 and 1984, Libera & Gosen 1986). Further, topological relationships between objects (e.g. represented by an association table) can also be arranged to have the same spatial keys as parts of their primary keys so these will become spatially clustered as well.

However, the approach has one potential drawback. Consider the problem of changing the geometry of such an object in a way that its spatial key changes. Thus, making some, possibly minor, geometric edit could result in a change to the primary key, i.e. identity, of the object, resulting in problems of maintaining the overall integrity of the database. Modification of a primary key effectively means deletion followed by re-insertion.

One might consider a storage mechanism where records are clustered according to a spatial key which is not a part of the primary key. While this would be satisfactory for extremely simple models (e.g. a single table of records with an auxiliary index for primary key retrievals) it would not cluster the records in the many association tables that exist in a richly connected data model.

A PRAGMATIC APPROACH

The idea of containing a spatial key within the identity of each object does not complicate other kinds of retrieval. As far as these procedures are concerned, a spatial key is no different from an ordinary key. Our pragmatic idea is that at the time an object is first created, we generate a spatial key as part of its unique identifier. The value of this identifier never changes from this point on, even if the location and geometric extent of the object is changed. The pragmatic part comes in that geometric edits are rare, and major changes in position or extent are even rarer. Thus the object rarely moves far in space, so why should it move far on disk? Thus an object's identity is a function of where it is born and we assume that it never moves far from its place of birth.

However, by doing this, spatial retrievals may become unreliable, because some objects which should be retrieved may be missed. Our solution to this is to have a single external spatial address table, with accurate spatial keys which are always maintained up-to-date, and it is this which is used to retrieve objects. The "sloppy" spatial key is no more than a clustering device, i.e. an accelerator to speed spatial search. In the worst case, if large parts of the geometry were modified significantly (e.g. following rectification), then the system may get slow, but it would still work correctly.

The method used to implement the index does not need to be the same as the method used to organise the clustering of the actual data so that, for example, a range tree index (Guttman 1984) could be used to index data which is clustered in a quadtree.

In our implementation, we use the same approach for both clustering the data on disk and for building the external index.

HOW TO GENERATE A SPATIAL KEY

As there seems to be general agreement that there is minimal difference in performance between the many tree-based approaches to spatial clustering, (Kriegel et al for example found differences of the order of 20% between the various methods that they investigated) then perhaps the next criterion could be to aim for simplicity. This therefore eliminates the range tree, because neither is it simple, nor is it easy to see how one generates a permanent, reproducible key from it. Smith and Gao found that methods based on B-trees were good on storage utilisation, insert speed and delete speed, but were inferior on search times. We suggest here a modification to the method of creating a key based on a linear quadtree which gives a worthwhile performance improvement

without degrading the other performances, nor adding any undue complexity.

Now it is well known that point data can be incorporated in a Morton Sequence, which is directly equivalent to encoding the quadtree cell in which the point exists. A quadtree index is very good for encoding point data, because all points exist in the leaves of the tree (See figure 1) (see the MX quadtree in Samet 1989).

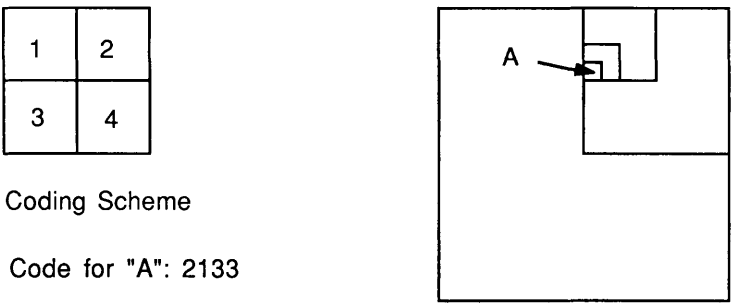
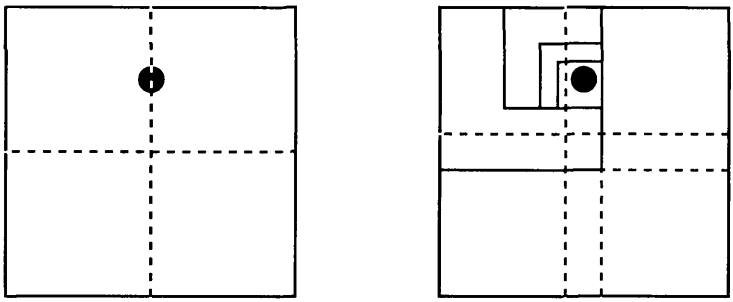


Figure 1: Quad-tree Encoding of Point Object

Encoding spatial objects with extent in a quadtree can also be done, but many objects will exist near the root (see MX-CIF Quadtree in Samet 1989 and Batty 1990), thereby leading to them being included in many retrievals when they are not in fact relevant. For large databases, this leads to a degradation in retrieval performance. Samet's book contains a number of schemes for getting around this problem by allowing an object to exist in more than one quadtree node. However, this is not suitable for clustering the object data itself.

In our earlier researches we had investigated solving this problem by using a key based on a nine-tree in which each square is divided into 9 equal sized overlapping squares each of which is a quarter of the size (half the dimension) of its parent. Although it had the desired effect of not populating the root, the tree is not well balanced and the retrieval strategy is more complex. This in fact was the basic approach behind Bundock's paper (Bundock 1987).

We include here a simpler solution to this problem because we have not seen it described elsewhere. The idea is based on the fact that most objects are very small compared to the world. So in order to avoid trapping objects near the root of the tree, the subdivision method is modified so that each quadtree cell is divided into 4 parts which overlap slightly, i.e each quadtree sub-square is slightly more than half the dimension of its parent square (See figure 2). The overlap never needs to be more than the size of the largest object in the database, and in practice can be less than this. The optimum overlap depends on some statistic of the object size, such as the mean object size times a factor.



Normal Quad-tree
Key : 0

Overlapping Quad-tree
Key : 1244

Figure 2: Quad-tree Encoding of Object with Extent

This slight modification to the simple quadtree key is no more complex to program, but does lead to worth while performance improvements for retrieving objects inside an area and in finding objects surrounding a point compared to the simple quadtree key mechanism.

A DATA MODEL FOR GIS

Figure 3 below illustrates graphically a simplified data model. It should be regarded as just a part of the complete model required for a GIS application. A large number of users' requirements for modelling their geometry and associated topology can be handled by a generic model of this form. Where users differ from one another is in the modelling of their own objects and interrelationships. The philosophy is not that geometric objects, such as polygons, have attributes, but that real world objects can be represented geometrically by such things as polygons. In this diagram, a line with an arrow signifies a one-to-many or many-to-one relationship and a line with an arrow at both ends signifies a many-to-many relationship. Of course, in a physical implementation, a many-to-many relationship is implemented by means of an intermediate table, which itself should also be spatially clustered.

The diagram should be read starting from the top. For example, a real world object, such as a wood is represented by an area, which may be made up of one or more polygons (these polygons may have resulted from intersections with other polygons in the same topological manifold). It is possible that each polygon may have one or more "islands" such as a lake (i.e in this case, a lake is an island). The lake area would of course share the same polygon as used by the wood area. Polygon boundaries are represented by a closed set of links, each one connecting exactly two nodes.

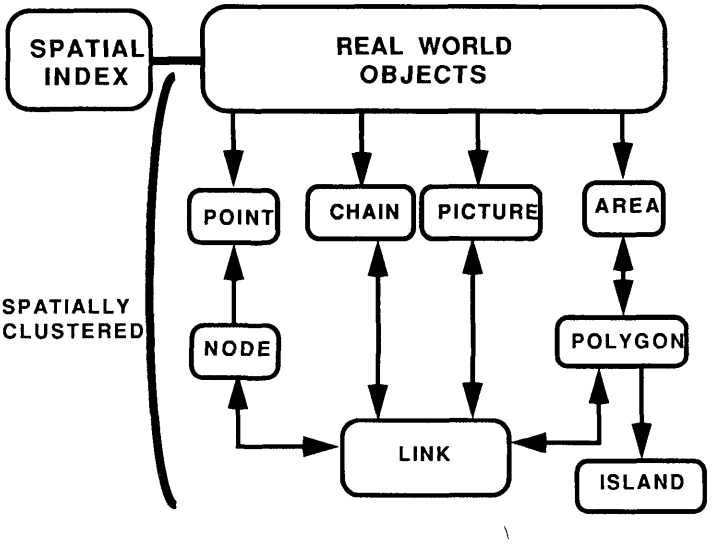


Figure 3: Spatially Indexed Topological Model

From the point of view of the system implementor, it is only the interrelationships of this model that he wishes to worry about, without himself being concerned with efficient spatial retrievals. However, if all entities in the model are identified by means of a key with spatial content, then the desired clustering of instances in each table will occur transparently. As it is common that objects which are topologically related are also spatially near to one another, disk accesses for topological queries should also be greatly reduced.

All that is needed in addition to this model is the external spatial index itself, which is merely a device for generating candidate keys for spatial retrieval. The point is that the spatial indexing method does not perturb the logical structure of the model.

In our implementation, the spatial index is a table, just like any other, which refers to real world objects, such as houses, lakes and utility pipe segments. A real world object may then have a number of different representations depending on such contexts as scale.

CONCLUSION

This paper has been concerned with the implementation of fast spatial indexing methods within the context of an integrated GIS data model. A design criterion has been to implement the spatial mechanisms without complicating other retrievals from the database. An approach is advocated based on a precise spatial index which can generate candidate keys within a tabular database whose primary keys contain a permanent, but "sloppy", spatial key. The external precise spatial index could be regarded as part of the data model, or could indeed be implemented as an entirely different mechanism. The method proposed for generating spatial keys is a minor

improvement to the simple quadtree, which we have described here, because we have not seen it published elsewhere.

ACKNOWLEDGEMENTS

We are grateful to Mike Bundock for his helpful comments on an earlier draft of this paper.

REFERENCES

Abel, D.J. & Smith, J.L. (1983): A Data Structure and Query Algorithm Based on a Linear Key for a Rectangle Retrieval Problem, *Computer Vision, Graphics, and Image Processing 24,1, October 1983.*

Abel,D.J. & Smith, J.L. (1984): A Data Structure and Query Algorithm for a Database of Areal Entities, *The Australian Computer Journal, Vol 16, No 4.*

Batty, P. (1990): Exploiting Relational Database Technology in GIS, *Mapping Awareness magazine, Volume 4 No 6, July/August 1990.*

Bundock, M. (1987): An Integrated DBMS Approach to Geographical Information Systems, *Autocarto 8 Conference Proceedings, Baltimore, March/April 1987.*

Chance, A., Newell, R.G. & Theriault, D.G. (1990). An Object-Oriented GIS - Issues and Solutions, *EGIS '90 Conference Proceedings, Amsterdam, April 1990.*

Easterfield, M.E., Newell, R.G. & Theriault, D.G. (1990): Version Management in GIS - Applications and Techniques, *EGIS '90 Conference Proceedings, Amsterdam, April 1990.*

Guttman, A. (1984): R-trees: A Dynamic Index Structure for Spatial Searching, *Proceedings of ACM SIGMOD Conference on Management of Data, Boston, June 1984.*

Kriegel, H., Schiwietz, M., Schneider, R., Seeger, B. (1990): Performance Comparison of Point and Spatial Access Methods, *in Design and Implementation of Large Spatial Databases: Proceedings of the First Symposium SSD '89, Santa Barbara, July 1989.*

Libera, F.D. & Gosen, F. (1986): Using B-trees to Solve Geographic Range Queries, *The Computer Journal, Vol 29, No 2.*

Samet, H. (1989): The Design and Analysis of Spatial Data Structures, *Addison Wesley, 1990, ISBN 0-201-50255-0.*

Seeger, B. & Kriegel,H. (1988): Techniques for Design and Implementation of Efficient Spatial Access Methods, *Proceedings of the 14th VLDB Conference, Los Angeles, California, 1988.*

Smith, T. R. and Gao, P. (1990): Experimental Performance Evaluations on Spatial Access Methods *Proceedings of the 4th Spatial Data Handling Symposium, Zürich 1990, Vol 2, p991.*