HOW SPATIAL DATA MODELS AND DBMS PLATFORMS AFFECT THE PERFORMANCE OF SPATIAL JOIN

by

Wei Zhou

B. Sc., Peking University, China, 1988

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
August 1993

APPROVAL

Name: Wei Zhou

Degree: Master of Science

Title of thesis: How Spatial Data Models and DBMS Platforms Affect the Performance of Spatial Join

Examining Committee: Dr. Robert Hadley
Chair

Dr. Wo-Shun Luk
Professor, Computing Science
Senior Supervisor

Dr. Jiawei Han
Associate Professor, Computing Science
Supervisor

Dr. Tiko Kameda
Professor, Computing Science
External Examiner

Date Approved: August 9, 1993

PARTIAL COPYRIGHT LICENSE

I hereby grant to Simon Fraser University the right to lend my thesis, project or extended essay (the title of which is shown below) to users of the Simon Fraser University Library, and to make partial or single copies only for such users or in response to a request from the library of any other university, or other educational institution, on its own behalf or for one of its users. I further agree that permission for multiple copying of this work for scholarly purposes may be granted by me or the Dean of Graduate Studies. It is understood that copying or publication of this work for financial gain shall not be allowed without my written permission.

Title of Thesis/Project/Extended Essay

How Spatial Data Models and DBMS Platforms Affect the Performance of Spatial Join.

Author: _____
(signature)

Wei Zhou
(name)

Aug. 11. 1993
(date)

Abstract

This thesis studies how the performance of a common spatial database operation, called spatial join, can be affected by different data models and DBMS platforms. We consider three spatial data models: Relational, BLOB (Binary Large Object Block) and Parent-Child Pointer model, which have different degrees of pointer involvement at the database schema level. ObjectStore and Sybase are chosen as representatives of object-oriented and relational DBMS. Our R-tree based spatial join algorithm, which is optimized in the step of polygon overlap checking, is presented. We measure the performance of this spatial join algorithm against five combinations of data models and DBMS platforms. Among other findings, the experimental results show that the Relational model does poorly on either DBMS platform, the running time being 3 to 4 orders of magnitude worse than the others. We introduce a technique called *application caching* that bridges the gap between the storage data structure (the data model) and the application data structure. By applying this technique to the algorithm runnning against the Relational and the BLOB data models, we show how the effect of data model/DBMS platform on performance of spatial join can be neutralized.

Acknowledgements

I wish to express my deepest gratitude and appreciation to Dr. Wo-Shun Luk, my senior supervisor, for his guidance, supervision, support and encouragement.

I am also grateful to Dr. Jiawei Han and Dr. Tiko Kameda for their careful readings and valuable comments on this thesis.

I wish to thank my frineds who provided me with their generous help. Especially, Chor Guan Teo, who spent much of her precious time correcting grammatical mistakes in my thesis, Hong Fan, who helped me understanding her research project during the preparation of my thesis. Roman Bacik, who gave me many suggestions in the implementation of the spatial join algorithm.

Finally, this thesis is dedicated to my parents and my sister for their love and care.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivations

Recently spatial database processing has become an active research area and many studies are concerned with the performance of spatial data structures. Typically, a number of data structures and/or algorithms are considered, and then their performance are evaluated either analytically (e.g., [11]) or experimentally (e.g., [4], [13], and [1]). In this thesis, we take a different approach to performance study of spatial database processing, which should complement the traditional ones. Assuming an algorithm has been selected for implementation, we consider how the performance of this algorithm may be affected by various implementation strategies. In most performance studies, analyses or experiments are mostly conducted in a system environment that is not representative of the system environments for many real-life applications. For example, many use the number of disk accesses as the performance metric, which implies the algorithm runs directly on the I/O system of the operating system. In an

application environment, however, many GIS (geographic information systems) use database systems (DBMS) to store their spatial and attribute data. Neither the end user nor GIS vendor/researcher has direct access to the I/O system. The I/O accessing is determined by the DBMS platform which controls how data are physically stored and the spatial data model which presents to the application a logical view of data. (In this thesis, spatial data models are meant to be only a slightly more abstract form of data schema at the DBMS level.) In comparing the performance of the same algorithm across different data models and DBMS platforms, we hope to gain insight into how an algorithm interacts with data models and DBMS platforms, which may lead to further optimization of the overall performance, for a given data model and a given DBMS platform.

With the advent of a next generation of database systems, generally known as Object-Oriented DBMS (OODBMS), it is natural to ask whether OODBMS is a more suitable platform than RDBMS (Relational DBMS) for implementation of GIS applications. In the research literature, GIS, along with CAD, CASE and others, is generally considered to be an application which these OODBMSs are designed for. Unlike CAD and CASE, however, most GISs continue to rely on relational DBMS as the data store. Even within the academic GIS community, there is an on-going debate on this issue. In the NCGIA (National Center for Geographic Information and Analysis) Core Curriculum on GIS [19], there is a comment that some OODBMSs may not be well-suited to GIS applications. It is not clear whether it is the object-oriented data modelling technique or the performance of OODBMS vis-a-vis RDBMS that is called into question. In any case, it is indicative of the need for more research in this direction. In this thesis we choose ObjectStore and Sybase, as the representatives of OODBMS and RDBMS, respectively.

There is a general perception that data models are intrinsically linked to specific DBMS platforms. For example, pointers may be included in a persistent (storage) data structure under OODBMS, while in a RDBMS, all persistent data are flat files in some normal forms. Here we argue that excluding physical pointer representation, the same data model (schema) can be defined on both OODBMS and RDBMS, and they are routinely done in many spatial database applications. For example, one can define flat files on an OODBMS, which is a common data model for (huge) attribute data associated with geographic objects. On the other hand, many RDBMS now provide a BLOB (Binary Large Object Block) data type, which is defined as part of a relation, while in implementation there is a pointer linkage between the contents of the BLOB attribute and those of other attributes. Spatial data models at the schema level have great impact on performance of the application since the data model determines how data may be accessed, e.g., by pointer (address) or by query (associative retrieval). Often a spatial application may not get to choose a data model which is most optimized for the application. This may be because the data model is pre-defined by the vendor who supplies the data (e.g., Maps used in Census [37]), or because the same data set being shared by other applications has a data model which is a compromised choice. In this thesis, we design three different spatial data models to study their impact on the performance of the application.

1.2 Thesis Objectives

This thesis uses a rather unusual method to study the performance issue in spatial database processing. We hope to get the perception in whether it is the spatial data modelling technique or the DBMS architecture that affects the performance of

spatial database processing. We design and implement some experiments to test the performance of one spatial application against different spatial data models and DBMS platforms. One goal of this thesis is to get some practical experience in answering the above question.

If it is true that for most non-traditional applications, OODBMS is better suited than RDBMS because of its pointer 'de-referencing' facility, one immediate question is whether it is possible to make RDBMS more efficient since most GIS still depend on RDBMS as the data store. From Ju Wu's research thesis [16], the answer is a positive one. In that project, in order to narrow the performance gap between OODBMS and RDBMS in one specific application – DSQL (Dynamic Spatial Query Language), they came up with a technique called *Preloading* which simulated ObjectStore's Memory Mapping and Pointer Swizzling method to "load the entire database into the client machine's virtual memory and handle the object by its virtual memory address instead of the relational key". In this thesis, we borrow and improve this *Preloading* idea in our performance study on spatial data modelling technique and DBMS platform. We call this strategy *application caching* in order to distinguish it from *client caching* in ObjectStore.

1.3 Methodology

The spatial application we choose for the performance study is called *spatial join*, which is a common spatial database operation, and has been studied in [23], [1], [11]. It is similar to the join operation in the relational database, but the two operands of the operation are sets of polygons instead of tuples from two relations. Like the

relational join operation, spatial join is a complex operation, and spatial indices are often needed as a filter to eliminate those pairs of polygons that are clearly non-overlapping. The algorithm we use for implementation is an improved version over many published algorithms which tend to focus exclusively on the choice of spatial indices.

We consider three data models for the spatial join operation : Relational, BLOB and Parent-Child Pointer model, which have different degrees of pointer involvement at the database schema level. That means data accessing is done by querying or by pointer 'de-referencing'.

There have been several studies on performance comparison between OODBMS and RDBMS. The most recent one is reported in [6]. For that study, a suite of queries (e.g., insert, lookup, and traversal) are performed against an engineering database (i.e., a network of parts wired together). This database is stored as a collection of flat files on an RDBMS and as pointer network on an OODBMS, respectively. It is shown that the performance of these operations on an OODBMS is, as a whole, much better than that on an RDBMS primarily due to the overhead of each SQL call to the RDBMS server and the ability of OODBMS to cache data on the client workstation. This research is not a benchmark study of DBMS platforms; its emphasis is on spatial database processing. Instead of simple operations, we study a complex operation, and how it interacts with various data models and DBMS platforms. Nonetheless the findings of the benchmark study have motivated us to conduct this research. The representatives we choose for the two different DBMS platforms are Sybase (RDBMS) and ObjectStore (OODBMS). We compare the performance of our R-tree based spatial join algorithm against five combinations of data models and DBMS platforms. The

five combinations are : Relational and BLOB model on Sybase, Relational, BLOB and Parent-Child pointer model on ObjectStore.

1.4 Thesis Overview

The rest of this thesis is organized as follows. The spatial join operation is discussed in Chapter 2. Spatial join definition and algorithm as well as some related works in both computational geometry and spatial database areas are presented there. The spatial data modelling is discussed in Chapter 3. We consider three different data models Relational, BLOB and Parent-Child Pointer model for our experiment. These models have different degrees of pointer involvement at the database schema level, so that data fetching methods from underlying database are different. We compare the two different DBMS platforms (object-oriented and relational) in Chapter 4. The representatives we used in our research are ObjectStore and Sybase. The *application caching* technique is also discussed there. The experimental setups which include implementation environment, polygonal data generation and the six data sets used in our experiments are described in Chapter 5. In Chapter 6 we analyze the experimental results. The conclusions of the thesis are presented in Chapter 7.

Chapter 2

Spatial Join

Spatial join is a basic geometric calculation operation. It can be found in many GIS applications. Basically, spatial join is a set operation that operates on two sets of spatial objects. The output of this operation consists of pairs of spatial objects, one from each input set. In GIS there are always many different maps, such as soils, crop productivity, land usage, time zones, administrative areas, etc., for each geographical area. Thus spatial join is a very useful spatial operation as it can synthesize information found in different maps of the same geographical area and can therefore answer complex spatial queries. For example, "find all the administrative areas in a certain time zone" can be answered by performing a spatial join operation on the time zone map and the administrative area map.

In this chapter, we will first discuss our R-tree based spatial join algorithm, followed by a short literature survey on spatial join.

2.1 Algorithm

In this thesis, the spatial objects for the spatial join operation are polygons. We represent a map as a set of non-overlapping polygons. The spatial join algorithm is an R-tree based two-step algorithm. Before presenting the algorithm, some basic definitions and assumptions are given. In addition, we will briefly describe the R-tree Spatial Indexing Method (SIM).

2.1.1 Definitions

In our spatial data models, polygons are simple polygons which have no intersecting edges and no holes. Other entities comprising each spatial data model include *vertex*, *edge*, *node* and *chain*. According to Hong Fan's thesis [14], we give a brief definition of these entities below: (See Chapter 3 for examples of these entities.)

- *vertex*: a *vertex* is a point in the 2-dimensional space.
- *edge*: an *edge* is a straight line between two *vertices*.
- *node*: a *node* is a *vertex* at which more than two *edges* terminate.
- *chain*: a *chain* consists of the sequence of *edges* between two adjacent *nodes*.
- *polygon*: a *polygon* is a closed sequence of *chains*.

The spatial join operation is about detecting overlapping polygon pairs from two polygon sets. There are two ways in which two polygons may overlap: partial overlap and total overlap. Two polygons partially overlap when (at least) one of the edges of

one polygon intersects or overlaps an edge of the other. Two polygons overlap totally when one is contained (enclosed) in another. Our spatial join algorithm is concerned with only partially overlapping polygons. The formal definition of the operation is given below:

Definition : Given two sets of polygons S and R, the spatial join operation returns all pairs of polygons (s, r), where s belongs to S, r belongs to R, and s and r overlap partially.

2.1.2 R-tree SIM

Since spatial operations usually deal with 2 or 3 dimentional data, traditional B-tree indexing technique used in many DBMSs is not suitable for multidimensional situation. A large number of spatial indexing methods (SIM) have been proposed to improve the performance of spatial operations. These include R-trees [12], K-D-B-trees [26], Quadtrees [29], [30], [31], Grid File [20] and so on. Our spatial join algorithm is based on one popular SIM – R-tree. The R-tree is a hierarchical SIM dealing with rectangular data. Each non-leaf node of the R-tree contains entries of the form *(MBR, child-pointer)*, where MBR is a Minimum Bounding Rectangle of all the MBRs in its child node, and child-pointer is the address of its child node. The leaf node has entries of *(MBR, object-id)*, where MBR is the object's MBR and object-id refers to a certain object. The node size M is defined as the maximum number of entries in each node. M is always chosen so that a node fits a page and I/O cache block. That is why R-tree is regarded as a page-oriented SIM. The number of entries in each node lies between m and M, where $m = \lfloor M/2 \rfloor$ is the minimum number of entries per node.

2.1.3 R-tree Based Algorithm

There are many similarities between the spatial join operation and the join operation in relational algebra, one of which is that the operation is quadratic in complexity if a brute-force nested loop algorithm is used. (The basic spatial join algorithm is shown below.) Thus, a spatial index is frequently used by a spatial join algorithm, much as a B-tree index is used by a join algorithm. Incidentally, the most frequently used spatial index, R-tree [12], or one of its variants, e.g. R^+-tree [9] and R^*-tree [1], is modelled after the B-tree.

INPUT : m polygons in *Map1*, n polygons in *Map2*;

OUTPUT : Set of pairs (P_i, Q_j), such that P_i belongs to *Map1* and Q_j belongs to *Map2*, and P_i and Q_j intersect. where $0 \leq i < m$, $0 \leq j < n$;

BEGIN

1. For each polygon P_i from *Map1*, $0 \leq i < m$

2. For each polygon Q_j from *Map2*, $0 \leq j < n$

    2.1. For each chain C of polygon P_i

    2.2. For each chain C' of polygon Q_j

        2.2.1. For each edge E of chain C

        2.2.2. For each edge E' of chain C'

            if E intersects with E', report intersection of P_i and Q_j.

END

Unlike the relational join algorithm, the purpose of a spatial index is to reduce the number of pairs of polygons that must be subjected to the polygon (partial) overlap test. This is because R-tree, or any of its variant, stores only rectangle approximations of the polygons. (See Fig. 2.2 for some examples of MBR). Given a polygon r in the polygon set R, using an R-tree index, one can locate only those polygons in S whose MBRs overlap with the MBR of r. Further computation is required to determine, precisely, which of these polygons actually overlap with r. Thus our spatial join algorithm contains two major steps:

- *R-Tree Join*: This preprocessing step acts as a filter to find all potentially overlapping polygon pairs. This is accomplished by checking for MBR overlap of each polygon in R against an R-tree of MBRs for polygons in the other set, S.

- *Polygon Overlap Computation*: For each pair of polygons returned by R-Tree Join, we check whether they really overlap. This is done by checking for intersection of all possible pairs of chains.

Since we do not assume existence of any auxiliary data structure, the construction of the R-tree is included in our algorithm. (As an option, we can also store the R-tree in the database.) Thus we can express our algorithm in the following sequence of major operations, which we call, the *execution plan.*

- 1. Open the database

- 2. Construct an R-tree

- 3. Compute R-tree join

- 4. Perform polygon overlap computation
- 5. Close the database

Up to now, our algorithm is very similar to other published two-step spatial join algorithms. After an initial implementation, it was discovered that the polygon overlap computation (step 4 in the plan) consumes most of the CPU time! This bias would reduce the spatial join to an unsuitable candidate for our experimental purposes. Fortunately, we have managed to introduce the following two techniques during the polygon overlap computation, which result in drastic reduction in the execution time.

- *Utilizing topological information*: Since the spatial data model can provide additional topological information for each chain, its left and right polygons, more information can be obtained when two chains intersect. Consider Fig. 2.1 which shows two polygons each from *Map1* and *Map2*. If during the checking for polygon overlaps of any pairs of polygons from *Map1* and *Map2*, chain AB is found to intersect chain CD, then one can conclude that all four pairs of polygons overlap. A 2-dimensional matrix is used to store this information, which is consulted each time before the pair-wise polygon checking is performed. It seemed that this technique would have reduced the execution time by about 75% because one pair of intersecting chains results in four pairs of intersecting polygons. But the experimental results show that this technique just reduces the execution time of step 4 by 15%. The reason for this is the result set from R-tree join step always has at least 1/3 polygon pairs that do not really overlap. We call them *false candidate pairs*. The polygon overlap computation for all these *false*

candidate pairs is most time-consuming since each chain in one polygon has to be checked with all the chains in the other one before we find out that they do not overlap. Time used on *false candidate pairs* cannot be reduced by utilizing topological information because no chain pair overlaps each other. Therefore, the performance of step 4 is improved by only 15% instead of 75% after using more topological information.

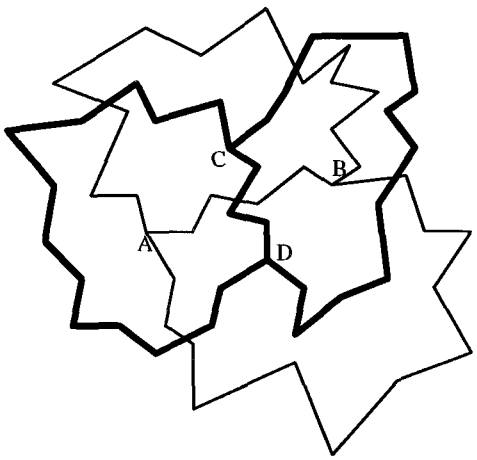

Figure 2.1: Intersecting chains AB and CD imply 4 pairs of intersecting polygons

- *Utilizing MBR intersection information (from the R-tree Join)*: Recall that even if the MBRs of two polygons overlap, the two polygons themselves may not overlap. Conversely, if these polygons do overlap, the overlapping region must be confined to the overlapping region of their respective MBRs, which for convenience is called *RECT*. We further infer that only those chains intersecting *RECT*(partially or totally) may intersect. In particular, if there are no chains from *either polygon* intersecting *RECT*, the two polygons cannot overlap. By applying a sort of line clipping algorithm in Graphics [27] to determine whether

the edges intersect RECT instead of checking whether two edges intersect each other, the performance can be significantly improved. Thus another filtering process is introduced in step 4 to reduce the polygon overlap computation. The idea is to calculate the intersection of two overlapped MBRs from R-tree join step instead of checking the two polygons directly. The result is another small rectangle, say *RECT*, such as the shaded region in Fig. 2.2. All chains of P_i and Q_j outside *RECT* are excluded from consideration. For the remaining ones, edge-by-edge intersection checking is done for all possible pairs of edges from P_i and Q_j. For example, in Fig. 2.2, only edges AB and BC in polygon P_i are checked with edges XY and YZ in polygon Q_j.

Thanks to this technique, the computation has been further reduced by 80-90%.

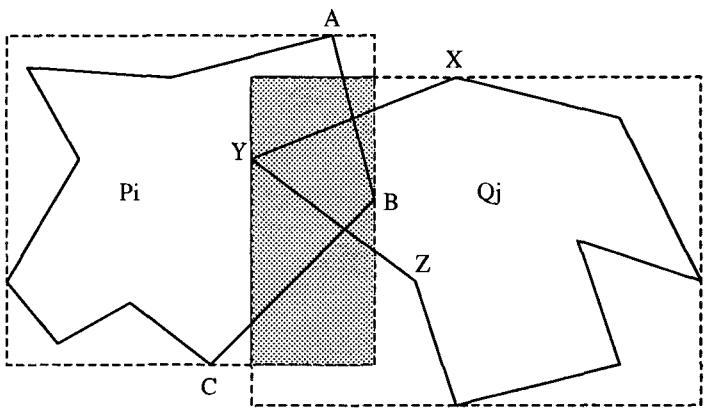

Figure 2.2: Only edges AB and BC are checked with edges XY and YZ

The detailed algorithm is presented below :

INPUT : m polygons in *Map1*, n polygons in *Map2*, where $m \leq n$;

OUTPUT : Set of pairs (P_i, Q_j), such that P_i belongs to *Map1* and Q_j belongs to *Map2*, and P_i and Q_j intersect, where $0 \leq i < m$, $0 \leq j < n$;

BEGIN

1. Initialize the result matrix $M_{n \times m}$ by setting all its elements to 0.

2. Create R-tree RT for the *Map2* which has the larger number of polygons. The MBR of each leaf node is the MBR of each polygon.

3. For each polygon P_i in *Map1*, where $0 \leq i < m$

   3.1. Read its MBR R_i

   3.2. Starting from N = the root of R-tree RT, for each element e of node N with MBR R'_e

   if R_i intersects with R'_e

   if N is a leaf node, record pair (P_i, Q_e);

   if N is a non-leaf node, for each child node of element e, let N be that child node and repeat 3.2;

4. For each pair (P_i, Q_j) recorded in the third step, where $0 \leq i < m$, $0 \leq j < n$

   4.1. if $M[P_i, Q_j] == 1$, goto 4;

   4.2. Initialize edge sets ES_1 and ES_2 to the empty set.

   4.3. Compute the intersection of MBRs of P_i and Q_j, call it R;

   4.4. For each chain C of polygon P_i

   4.4.1. For each edge E of chain C

if E is not outside R, insert E to edge set ES_1;

4.5. If edge set ES_1 is empty, goto 4;

4.6. For each chain C' of polygon Q_j

4.5.1. For each edge E' of chain C'

if E' is outside R, insert E' to edge set ES_2;

4.7. If edge set ES_2 is empty, goto 4;

4.8. For each E of edge set ES_1, suppose E also belongs to P_k

4.8.1. For each E' of edge set ES_2, suppose E' also belongs to Q_l

if E intersects with E'

Set $M[P_i, Q_j]$, $M[P_i, Q_l]$, $M[P_k, Q_j]$, $M[P_k, Q_l]$ to 1, goto 4

5. For each $M[i, j] == 1$

report intersection P_i and Q_j.

END

The experimental data showing the improvement of our spatial join algorithm will be presented in Chapter 6 after we explain all the important concepts, such as *spatial data modelling*, *Sybase and ObjectStore platforms*, *application caching*, and describe the experiment setups which include implementation environment and data sets.

2.2 Literature Review on Spatial Join

The polygon intersection problem is not a new topic in the field of computational geometry. There have been many studies on finding optimal intersection algorithms

for basic geometrical elements such as segment in the literature [34], [2] and [3]. Theoretically, these algorithms are very efficient. However, they always require complex auxiliary data structures and extensive preprocessing of raw data.

With the spatial database becoming an active research area, researchers in database area have been working on spatial data query processing which also needs to manipulate geometric objects. Many practical solutions for spatial operations have been developed and implemented for spatial database (e.g., [22], [24], [23], [4], [11], [36], [28]).

2.2.1 Spatial Overlap Query in PROBE

PROBE[22] is a research project for an object-oriented image database system. In PROBE, spatial objects are constructed by *point sets* entities. The *point set* of an object is a set of points in the space occupied by that object. *Point set* has a well-defined set of operations suitable for many applications. Spatial overlap operation of two spatial objects could be performed on their corresponding *point sets*. The problem with this approach is its large time and space requirements. For this reason, PROBE introduces a geometry filter : "a k-D point set indicating a spatial object is approximated by superimposing a k-D grid of cells on the space." This grid representation is a conservative approximation of the object. The spatial overlap query in PROBE is also processed in two steps. First, a geometry filter is involved. Next, ordinary geometrical computation is performed to refine the candidate set.

One of the interesting part of this research is the geometry filter. Spatial overlap operation for grid representations can be implemented simply – the same logical AND

operation is applied for each grid cell. However, in this case, the performance is still a big problem when grid resolution is high. PROBE overcomes this drawback by encoding the grid. "The encoding is obtained by recursively partitioning the space where the object is occupied until the boundary of the object is obtained or the maximum resolution is reached." One region obtained by a sequence of splitting has a unique corresponding bit string which is called z value of that region. By sorting z values lexicographically, z order is obtained. Z order is a kind of mapping from k-D space to 1-D space with the spatial proximity preserved. Clustering can be achieved for efficient disk access by preserving spatial proximity. Another practical benefit of using Z order is that traditional indexing methods (B-tree, B^+-tree) or other conventional file organizations can be used for spatial database. Under this encoding scheme, spatial overlap can be performed by checking whether a z value in one input is a prefix of that in the other input.

2.2.2 Spatial Join Based on Different Approximations

In [4], an *approximation-based query processing* mechanism for managing large sets of complex polygonal objects was introduced. Just like many other methods of spatial query processing, the approximation-based query processing is also performed in two steps, *filter step* and *refinement step*. But it has two important features in addressing the efficiency issue of managing geometric objects.

- To make the *filter step* as fast and accurate as possible, the approximation of object should be simple and should have a good approximation quality. This research designed and investigated several convex conservative approximations

which meet these requirements: Minimum Bounding Rectangle, Rotated Minimum Bounding Rectangle, Minimum Bounding Circle, Minimum Bounding Ellipse, Convex Hull and Minimum Bounding n-corner. From their testing results, in most cases (different complexity of objects and different type of queries), "the approximations 5-corner, ellipse and rotated bounding rectangle clearly outperform the bounding rectangle which is used in many spatial query processing mechanisms."

- Instead of using complicated Spatial Indexing Methods, such as sphere tree[25], cell tree[10], polyhedra-tree [15] or P-tree[33], R*-tree which is originally designed for bounding rectangles is adopted to organize the non-rectangular approximations. In this case, spatially adjacent approximations are grouped into one leaf-node. Since more complex approximations need more storage, the node size is determined by the complexity of the approximation and it will further influence the performance of the Spatial Indexing Method. Just like an ordinary R*-tree, each non-leaf node has elements of form (*MBR*, *child-pointer*) and thus can be organized by the R*-tree in the normal way.

Both our spatial join algorithm and this approximation-based query processing are aimed to reduce the execution time in *the refinement step* during which a complex and time-consuming computational geometry algorithm is performed. Our strategy is to design another filter processing to simplify the polygon overlap computation. This research focuses on investigating different object approximations with high approximation quality to reduce the *false hits* from the *filter step.*

Chapter 3

Spatial Data Models

3.1 Spatial Data Model Representations

It is widely known that the representations of spatial data models fall within two major classes : vector representation and raster representation. The vector data model represents geographical feature by explicitly defining its component geometric entities, such as points, lines and polygons, as well as the topological information among these entities. In the raster data model, the whole space is participated into small pieces or cells (like PROBE project). Each object is approximated by a set of cells overlapping the object. The representation is therefore conservative, and the precision is limited by the resolution of the cell. Compared to its raster equivalent, the vector data model has much higher precision while representing objects, but requires much more complicated geometrical computations.

We adopt the vector data model representation for all three spatial data models

designed for our spatial join algorithm.

3.2 Spatial Data Models

Three spatial data models are considered in this thesis. These are chosen as representatives from a spectrum of data models, according to the extent to which pointers are present in the data model. These models are: Relational, BLOB (Binary Large Object Block), and Parent-Child Pointer model (or simply Pointer model). Although Relational and BLOB models are actually designed for the relational DBMS platform, both of them can also be implemented in an object-oriented database. In this case, data accessing has to be done by query (associative retrieval) instead of 'de-referencing'. Obviously, there is a trade-off between performance and flexibility. With pointers, access from one object to other associated objects is very fast, and is more so for the current generation of OODBMS. Relational data model, on the other hand, uses key/foreign-key to associate related objects. A query is the only way to access data from the database. This results in a reduction in the performance. However, pointer linkage tends to 'hard-wire' association of objects which works for some applications but may not work well for other applications which require different access strategies.

A spatial data model for spatial join is not very different from that for other spatial applications since it must define all basic entities in a polygonal map, and its topology information of the polygons therein. The only application-specific information included in the spatial data models we use is about the Minimum Bounding Rectangle (MBR) of each polygon.

3.2.1 Relational Data Model

In this spatial data model, each polygon is broken into multiple entities which are stored in separate tables. The data model is constructed as a four-level hierarchical structure – polygon, chain, edge and vertex. In this model most of the topology information is provided by foreign keys. The schema definitions and some sample tuples of these four tables are given below. These tables are defined according to the common relational data model such as the one described in [7]. This data model can also be implemented on ObjectStore. Since in the schema definitions only two basic data types (int and float) are used, the only thing we need to do is to translate each table to a class.

This model, while containing all information required by our spatial join algorithm, is designed without considering the efficiency of this or other applications. Many geometric data models are defined in a similar way. It is clear that this model is not 'compatible' with the data structure requirement of our algorithm. For example, many database accesses and joins are needed to get all the information of one polygon required by our algorithm.

```
Table POLYGON
( PolygonID     int,     /* Polygon ID */
  ChainNum      int,     /* No. of chains in the polygon */
  StartChain    int,     /* The first chain in the polygon */
  MBR_llx       float,   /* The low-left coordinate of polygon's MBR */
  MBR_lly       float,
  MBR_urx       float,   /* The upper-right coordinate of polygon's MBR */
```

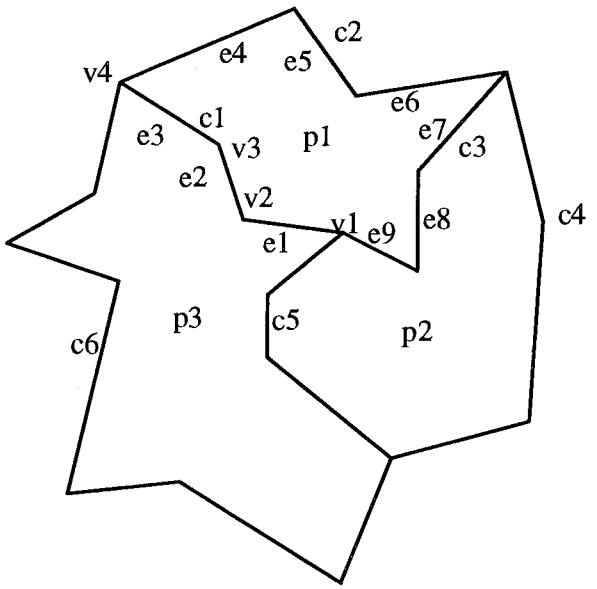

Figure 3.1: An Example of a Polygonal Map

```
  MBR_ury       float )

Table CHAIN
( ChainID                 int, /* Chain ID */
  EdgeNum                 int, /* No. of edges in the chain */
  StartEdge               int, /* The first edge in the chain */
  LeftPoly                int, /* The polygon on the left side of the chain */
  NextChainInLeftPoly     int, /* The next chain in the left polygon */
  RightPoly               int, /* The polygon on the right side of the chain */
  NextChainInRightPoly int) /* The next chain in the right polygon */


Table EDGE
```

PolygonID	ChainNum	StartChain	MBR_llx	MBR_lly	MBR_urx	MBR_ury
p1	3	c1				
p2	3	c3				

Table 3.1: Polygon Relation

ChainID	EdgeNum	StartEdge	LeftPoly	NextChainInLeftPoly	RightPoly	NextChainInRightPoly
c1	3	e1	p3	c5	p1	c2
c3	3	e7	p1	c1	p2	c4

Table 3.2: Chain Relation

```
( EdgeID      int,    /* Edge ID */
  Vertex1     int,    /* The first vertex in the edge */
  Vertex2     int,    /* The second vertex in the edge */
  NextEdge    int )   /* The next edge in the chain */

Table VERTEX
( VertexID    int,      /* Vertex ID */
  x           float,    /* x coordinate of vertex */
  y           float )   /* y coordinate of vertex */
```

EdgeID	StartVertex	EndVertex	NextEdge
e1	v1	v2	e2
e2	v2	v3	e3
e3	v3	v4	nill

Table 3.3: Edge Relation

VertexID	x	y
v1		
v2		

Table 3.4: Vertex Relation

3.2.2 BLOB Data Model

The major difference between this model and the Relational data model is the addition of the BLOB data type, which does not have a fixed length. BLOB data type is the answer of RDBMS vendors to the user's need for the DBMS to handle non-text data, such as images. It is also frequently used as a device to handle variable-sized text data. Many GIS applications represent their map data in either images (raster-based) or text (vector-based). In this thesis, although we choose to represent a map as a set of non-overlapping polygons, BLOB data type is still very useful because each polygon has a variable number of chains, each of which in turn has a variable number of vertices. BLOB data type has another implication that is important to this research. Sybase's equivalent of BLOB, called *image* data type, is capable of holding up to 2,147,483,647 bytes of binary data. Image data is stored, internally, on a linked list(s) of data pages separated from other data storage for the relation [35]. It is in this sense that the BLOB data model is a compromise between the relational model (strictly no pointers) and the parent-child pointer model (explicitly defined pointers).

Our BLOB data model consists of two relations: polygon and chain, each of which has an image attribute.

```
table POLYGON
( PolygonID     int,     /* Polygon ID */
```

PolygonID	ChainNum	MBR_llx	MBR_lly	MBR_urx	MBR_ury	ChainBuf
p1	3					c1 c2 c3

Table 3.5: Polygon Relation

ChainID	LeftPoly	RightPoly	VertexNum	VertexBuf
c1	p3	p1	4	x1 y1 x2 y2 x3 y3 x4 y4

Table 3.6: Chain Relation

```
  ChainNum      int,    /* No. of chains in the polygon */
  MBR_llx       float,  /* The low-left coordinate of polygon's MBR */
  MBR_lly       float,
  MBR_urx       float,  /* The upper-right coordinate of polygon's MBR */
  MBR_ury       float,
  ChainBuf      image ) /* Sequence of chains in the polygon */

table CHAIN
( ChainID       int,    /* Chain ID */
  LeftPoly      int,    /* The polygon on the left side of the chain */
  RightPoly     int,    /* The polygon on the right side of the chain */
  VertexNum     int,    /* No. of vertices in the chain */
  VertexBuf     image ) /* Sequence of vertices in the chain */
```

ObjectStore does not support BLOB, or image data type. Instead we use the *os_List* construct as the equivalent in order to implement the BLOB model on ObjectStore. An os_List is a parameterized class, which is used to define an arbitrary

collection of objects belonging to the same class. ObjectStore allows many ways to access objects in this collection, but for our purposes here, this collection is used in the same way as the attributes stored in an image attribute.

```
class Vertex {
    public:
        float   x, y;
};

class Poly {
    public:
        int             polyid        indexable;
        RECTANGLE       *MBR;
        int             chainnum;     // No. of chains
        os_List<int*>   chainbuf;     // Sequence of chain ids
};

class Chain {
    public:
        int               cid             indexable;
        int               lpolyid;        // left polygon id
        int               rpolyid;        // right polygon id
        int               verticenum;     // No. of vertices
        os_List<Vertice*>  verticebuf;     // Sequence of vertices
};
```

3.2.3 Parent-child Pointer Data Model

Even though the BLOB data model provides pointer access to variable-sized data, there are two major limitations of BLOB. First, there is no way to address directly a specific segment within the image data. Of course, once the data is in memory, any part of the image data can be addressed, but the *entire* image must be brought into the memory prior to that. Second, there cannot be any image field within the image field. This is because no explicit pointer is allowed in the model. The implication is that it is impossible to build multi-level hierarchies with BLOB data type.

The Parent-child pointer data model provides the flexibility that is required to model the storage data structure (i.e., the layout of the data in the storage) as closely to application data structure as possible. This is the case for the data model (schema) below which is optimized for the purpose of this application. For example, during the polygon overlap computation, information about a polygon's chains are frequently accessed. Likely, all edges (or their vertices) are frequently accessed from the chain they belong to. The parent-child pointer model supports fast accesses in these situations.

Clearly, this pointer data model can be implemented only on the OODBMS platform.

```
class Vertex {
    public:
        float   x, y;
};

class Chain {
```

```
    public:
        int     ChainID;     // Chain ID
        int     LeftPoly;    // Left polygon id
        int     RightPoly;   // Right polygon id
        int     VertexNum;   // No. of vertices
        Vertex  VertexBuf[NumofVertices];   // Sequence of vertices
};

class Polygon {
    public:
        int         PolygonID;  // Polygon ID
        RECTANGLE   *MBR;       // The MBR of the polygon
        int         ChainNum;   // No. of chains
        Chain       *ChainBuf[NumofChains];  // Pointers to chains
};
```

The main difference between BLOB and the Parent-child pointer model is the way in which chain information is stored in the polygon structure. In the BLOB model, a sequence of chain IDs that form the polygon are stored. In this case, database queries based on the chain IDs have to be performed in order to get chain information. On the other hand, in the Parent-child pointer model, the pointers to the chains are saved in the polygon. Thus, chain information can be obtained by 'de-referencing' the pointer. As we will see in the performance analysis in chapter 6, this difference results in substantially different performances.

Chapter 4

DBMS Platforms and Application Caching

Currently relational DBMS platform is still a major storage mechanism for large collections of data and is still widely used for various applications, even for some non-traditional ones such as GIS. One major reason for this trend is because relational DBMS is well-developed and technologically-mature. However, because of its limitations, such as uniformity, tuple orientation, small data items and atomicity of attribute[17], more and more non-traditional applications are leading the database model research in different directions.

One direct way to overcome the shortcomings of the relational database model is to extend this model. One example of an extended relational database model is the Nested Relation. The Nested Relational database model allows relations that are not in first normal form. That means the value of a tuple on an attribute may be an atomic value or a relation. Thus, in this model, a complex object with a hierarchical

structure can be directly represented by a single tuple of a nested relation.

Another alternative is to develop the object-oriented database model. Object-oriented databases are based on the object-oriented programming paradigm and augmented by persistency as well as other database features such as transaction management. In an object-oriented database model, users can use concepts, such as hierarchical structure, inheritance, etc., to describe their understanding of the data structure of an object in the real world. That means a complex object can be represented as an individual unit which is closer to the user's concept, while, relational database models require users to consider a complex object in multiple relations[18]. However, the object-oriented database model also sacrifices many of the advantages of the relational database model. For example, it provides few means of descriptive set-operations[32].

The two different DBMS platforms we adopted for our performance study are Sybase and ObjectStore. Although there are many differences between Sybase and ObjectStore, we will only examine here two major ones that would affect the performance of the application: persistence and local caching.

4.1 DBMS Platforms – Sybase and ObjectStore

ObjectStore is essentially a database programming language [5], i.e., C^{++} with database extensions. Persistent and transient data are treated in a uniform way. The only difference is that when the data is allocated, persistent data must be declared as *persistent.* Once data is declared as such, the application programmer need not be concerned with its longevity. There are two ways to access persistent data: *de-referencing* and

by query (or querying). To de-reference means to load the contents given the virtual memory address. ObjectStore's virtual memory database architecture stipulates that every piece of persistent data is given a virtual memory address. If the data is not in memory, a page fault will be generated and program control will be passed on to ObjectStore. Once the data is located, a proper virtual memory address is assigned and its contents is then loaded. ObjectStore also provides a query facility for associative retrieval, i.e., an object is located given the values of its attribute(s) [21]. The same query facility can also be used to explicitly write an object into ObjectStore (e.g., an insert command). Mostly, however, the persistent objects are created and modified in the same way as transient objects, without resorting to querying. In contrast, applications running on Sybase, like many other DBMSs, must use SQL to read or write persistent data, i.e., data stored in Sybase.

In a typical database processing situation, the application and the DBMS run as separate processes, or, on separate machines if it is a client/server architecture. In Sybase, or other RDBMS, the query is issued by the application and sent to the server, which, after processing, returns the result data set to the application. In contrast, an OODBMS sends to the application more data than requested, which remains in the cache memory of the local process/machine until it is swapped out. This is called *local caching* [6] and it is transparent to the application programmer. There are two types of local caching: object-level and page-level, depending on whether the object, or the physical page (or segment) that contains the requested data, is sent. In [8], a performance comparison is made between these two types of local caching. In this thesis we consider page-level local caching, since it is what ObjectStore has adopted.

Regardless of which type of local caching is adopted, its existence causes a major

system architectural difference between the RDBMS and OODBMS and contributes to the superior performance for the latter for certain types of application. By pre-loading the data into the local cache, the OODBMS creates a physical copy of the data. This data is identical in contents but physically distinct from the corresponding storage data in the sense that the OODBMS must maintain consistency between these two images of the same data. However, this cost arising from concurrency control is outweighed by the benefit provided by the local caching. Without that, de-referencing style of data accessing would be impossible. In addition, ObjectStore has a clever way of determining whether a data item, given its virtual memory address, is located in the cache without any overhead. This makes it possible for an ObjectStore application to access persistent data as efficiently as transient data. However, there is a cost associated with the unique database architecture. We discovered in our experiments that the database open time is much lengthier than that of Sybase. This is discussed in greater detail in Chapter 6.

4.2 Application Caching

Local caching is done by ObjectStore transparently to the application. However, there is no reason why the same approach cannot be taken by the application programmer, in order to achieve the same kind of superb performance improvement for the RDBMS platform. Since the local caching is done by the application, not by the DBMS, we call this *application caching* to differentiate between the two. The major advantage of application caching over local caching is that the former is controlled by the application, and consequently the layout of the data in the cache, i.e., the cache data structure, can be tailored to the needs of the application. In addition, by pre-loading

the bulk of data from the database before the objects are required by the application, the query cost in the application can be reduced. This is because fetching a set of relational tuples by a range query is much less expensive than shipping a single tuple from the database because of the high start up cost per data transmission.

4.2.1 Application Caching Mechanism

To include the step for application caching, we modify the execution plan of the spatial join algorithm as follows:

- 1. Open database
- 2. Perform application caching
- 3. Construct R-tree
- 4. Compute R-tree Join
- 5. Perform polygon overlap computation
- 6. Close database

Since the goal of this thesis is to study how the performance of a common spatial database operation – spatial join, can be affected by different data models and DBMS platforms, and since the cache data structure is not tailored to the requirements of the application during the cache pre-loading, the data conversion from the spatial data model to the application data structure is conducted during the spatial join operation. The main function of the application caching is to pre-load all or part of database

(depending on the database size) to the local cache in order to cut down the query cost in the application program.

The complexity of application caching is of course application dependent. It also depends on the size of the database. For large databases which may not be loaded entirely at the same time, caching management would be required. Basically our application caching mechanism is constructed in a fashion similar to ObjectStore. It consists of two major components : *address converter* and *application cache manager*. An overview of the application caching mechanism is shown in Fig. 4.1.

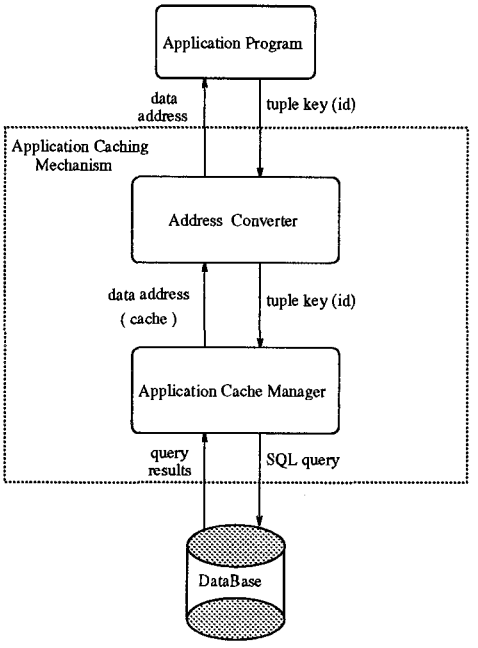

Figure 4.1: Application Caching Mechanism

The function of each part of the application caching mechanism is described below:

- *Address Converter* : The address converter should maintain a mapping table which converts a key of a relational tuple into its virtual address in the application cache. In our experiment, the application cache is composed of a number of arrays. Each array conforms to its correspondent relation in the spatial data model. The data type of each array is exactly the same as the definition of its correspondent relation. That means no data structure alignment is applied at the data pre-loading stage. Since each relation of our spatial data model has an integer primary key (e.g., Polygon ID, Chain ID, etc.), this key can be used as the index value for direct access to the required tuple in the application cache. If the primary key cannot be used as a value of the index, then a suitable hash function can be applied to map the primary key into an appropriate index value. This access will be slower than the access by 'de-referencing' in ObjectStore because it requires some computation to translate the index value into the virtual address of the tuple. However, it is considerably faster than query access.

- *Application Cache Manager*: The application cache manager is required because it may not be possible to pre-load the entire database into the application cache due to the swap space limitation of the client machine. Thus when the address converter fails to return the cache address of the required tuple, it will pass the primary key of the tuple to the application cache manager to retrieve the tuple from the database by issuing queries. Since in this case the application cache has already been full, an existing tuple in the application cache will be replaced by the incoming tuple according to a first-in-first-out (FIFO) policy. After adjusting the relevant entries in the mapping table, the virtual cache address of the required tuple will be passed to the application program by the address converter.

An example of the application caching algorithm for the relational model is given below. In the relational model, tuples which are always accessed together in one relation are given sequential IDs and stored in that sequence in the relational table in order to achieve data clustering. For example, all the edges in one chain are arranged in a sequence of edge IDs and stored one after another. When the chain information is required by the application program, all the edges of that chain can be fetched by a single range query.

The application caching algorithm comprises two sections : pre-loading algorithm and cache manager algorithm.

- Pre-loading Algorithm :

  BEGIN

  1. Loading *polygon* tuples.

     1.1. Set POLYNUM to the minimum value of the number of polygon tuples in the polygon relation and the number of polygons that can be loaded into the application cache.

     1.2. For each of the POLYNUM polygons

        1.2.1. Load the polygon tuple into the application cache.

        1.2.2. Invoke the address converter to maintain the mapping (from the polygon ID to its virtual address in the application cache).

  2. Loading *chain* tuples.

     1.1. Set CHAINNUM to the minimum value of the number of chain tuples in the chain relation and the number of chains that can be loaded into the application cache.

1.2. For each of the CHAINNUM chains

1.2.1. Load the chain tuple into the application cache.

1.2.2. Invoke the address converter to maintain the mapping (from the chain ID to its virtual address in the application cache).

3. Loading *edge* tuples.

3.1. Set EDGENUM to the minimum value of the number of edge tuples in the edge relation and the number of edges that can be loaded into the application cache.

3.2. For each of the EDGENUM edges

3.2.1. Load the edge tuple into the application cache.

3.2.2. Invoke the address converter to maintain the mapping (from the edge ID to its virtual address in the application cache).

4. Loading *vertex* tuples.

4.1. Set VERTEXNUM to the minimum value of the number of vertex tuples in the vertex relation and the number of vertices that can be loaded into the application cache.

4.2. For each of the VERTEXNUM vertices

4.2.1. Load the vertex tuple into the application cache.

4.2.2. Invoke the address converter to maintain the mapping (from the vertex ID to its virtual address in the application cache).

END.

- Cache Manager Algorithm :

BEGIN

1. Invoke the address converter to get the virtual address for the polygon ID from the mapping table.
2. If success, return its virtual address to the application program.
3. Upon failure, get a polygon tuple from database server (GetPolygonFromDB).
4. From the polygon tuple, get the first chain ID in the polygon.
5. For this chain :
    5.1. Invoke the address converter to get the virtual address for the chain ID from the mapping table.
    5.2. If success, return its virtual address to the application program.
    5.3. Upon failure, get a chain tuple from database server (GetChainFromDB).
    5.4. From the chain tuple, get the first edge ID in the chain.
    5.5. For this edge :
        5.5.1. Invoke the address converter to get the virtual address for the edge ID from the mapping table.
        5.5.2. If success
            5.5.2.1. Return its virtual address to the application program.
            5.5.2.2. From the edge tuple, get the two vertex IDs of the edge.
            5.5.2.3. For each vertex :
                5.5.2.3.1. Invoke the address converter to get the virtual address for the vertex ID from the mapping table.
                5.5.2.3.2. If success, return its virtual address to the application program.

5.5.2.3.3. Upon failure, get a vertex tuple from database server (GetVertexFromDB).

5.5.2.4. From the edge tuple, get the next edge ID in the chain, goto step 5.5.

5.5.3. Upon failure, get edge tuples from database server (GetEdgesFromDB).

5.6. From this chain tuple, get the next chain ID, go to step 5.

END.

GetPolygonFromDB

BEGIN

1. Create a query by the polygon ID and send it to the database server.
2. Wait for the query result.
3. According to the FIFO policy, choose an existing polygon tuple in the application cache to be replaced by the incoming polygon tuple.
4. Invoke the address converter to maintain the mapping (from the polygon ID to its virtual address in the application cache).
5. Return its virtual address to the application program.

END

GetChainFromDB

BEGIN

1. Create a query by the chain ID and send it to the database server.
2. Wait for the query result.

3. According to the FIFO policy, choose an existing chain tuple in the application cache to be replaced by the incoming chain tuple.
4. Invoke the address converter to maintain the mapping (from the chain ID to its virtual address in the application cache).
5. Return its virtual address to the application program.

END

GetEdgesFromDB

BEGIN

1. Create a range query by all the rest edge IDs in the chain and send it to the database server.
2. Wait for the query results.
3. According to the FIFO policy, the incoming edges will replace the same number of existing edges in the application cache.
4. Invoke the address converter to maintain the mapping (from the edge IDs to their virtual addresses in the application cache).
5. Return their virtual addresses to the application program.
6. For each edge tuple, get the two vertex IDs of the edge.
7. For each vertex :
   - 7.1. Invoke the address converter to get the virtual address for the vertex ID from the mapping table.
   - 7.2. If success, return its virtual address to the application program.
   - 7.3. Upon failure, get a vertex tuple from database server (GetVertexFromDB).

END

GetVertexFromDB

BEGIN

1. Create a query by the vertex ID and send it to the database server.
2. Wait for the query result.
3. According to the FIFO policy, choose an existing vertex tuple in the application cache to be replaced by the incoming vertex tuple.
4. Invoke the address converter to maintain the mapping (from the vertex ID to its virtual address in the application cache).
5. Return its virtual address to the application program.

END

4.2.2 Evaluation of Application Caching

The performance results of implementing application caching on top of the relational database show that we can get an amazing improvement of performance with a reasonable cost. Detailed performance analysis will be presented in Chapter 6. Interestingly, applications running on ObjectStore can benefit from the application caching as well. That means the application caching is applied on top of the local caching by ObjectStore. Why? In case of local caching by ObjectStore, the layout of data in the cache, i.e., the cache data structure, is beyond the control of the application programmer, in the sense that it is determined by the database schema. If, for some reason, the cache data structure is not 'compatible' with the application, expensive searching for

data requested by the application is required. In this case, local caching by ObjectStore may not be very effective. Thus, a case can be made for another level of local caching, by the application, whose main purpose is to convert the cache data structure determined by the database schema into one that is well-suited to the application.

Since no concurrency control mechanism is included into the implementation, currently the application caching works properly only in read-only situations. Fortunately, many spatial database operations, especially those complicated ones, do not modify spatial data, e.g., maps, which are usually compiled in separate processes.

Chapter 5

Experimental Setup

In Chapters 3 and 4, we presented three different data models Relational, BLOB and Parent-Child Pointer and discussed two different DBMS platforms ObjectStore and Sybase for our spatial join operation. In this chapter, a brief overview of the experimental setup, which includes the implementation environment and the polygon set generation, will be given.

5.1 Implementation Environment

We use Sun workstations for our experiments. For ObjectStore, both the client and server (ObjectStore) run on a Sun 4/280 with 32 MB memory, but the system and its files are stored in a remote disk. Sybase runs on a similar platform but the client runs on a separate machine. Despite the asymmetry between the setups of these two DBMS platforms, we do not think the differences are significant enough to alter our qualitative comparisons of the results in connection with these platforms. All the

experimental results are obtained during the mid-night or weekend when the computer workload and network traffic are low.

5.2 Generating Polygon Set

5.2.1 Randomly Generating Polygon Set

The spatial space that contains the polygon set is restricted to a square which is from lower-left (0, 0) to upper-right (1, 1). The polygon set generating algorithm has three steps :

- Generating Random Lines : A number of straight lines are generated in this step. The two end points of each line are located on two different edges of the space square. Thus, the line orientation has six possibilities: SW, SN, SE, WN, WE, NE. All the end points of straight lines are generated on one of the four edges according to the uniform probability distribution. Fig. 5.1 shows seven straight lines generated in the space.

- Tracing Polygon Set : Polygons which are composed of intersections of the straight lines are traced out from the line net generated in the first step. To simulate real area boundaries in the map, m number of additional points generated by a uniform bias toward the original straight line segment are added to each line segment. The original line segment will be replaced by a sequence of new line segments connecting all the m points. This sequence of new line segments actually is a chain which we have already defined in Chapter 2. The uniform bias can be controlled by *smoothness factor*, by which the chain can

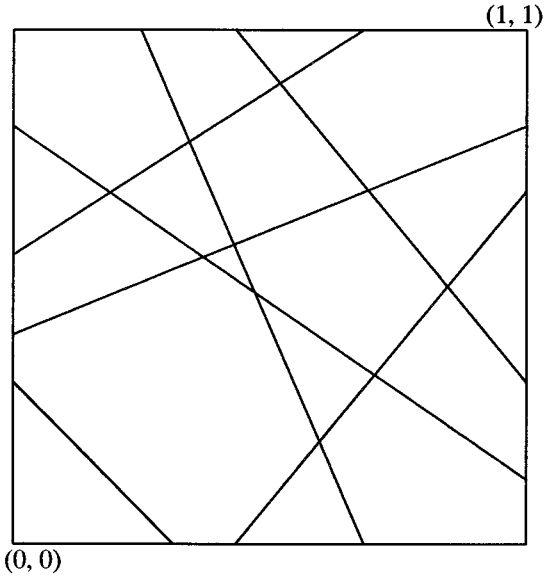

Figure 5.1: Line Net With Uniform Distribution of Ending Points

zigzag to different extent. The smoothness factor ranges from 0 to 100. The higher the smoothness factor, the more winding the chain will be. For example, if the *smoothness factor* is 100, then all the additional points are generated on the original line segment. With the *smoothness factor* is 0, an extremely jagged chain connecting the additional points will be generated. Therefore, the number of additional points m together with the *smoothness factor* determines how jagged the chain will be. By restricting the area in which each sequence of additional points are generated, we can guarantee that all the polygons in the space do not overlap. Fig. 5.2 presents the polygon set after the boundary modification of Fig. 5.1.

- Eliminating Small Polygons : The number of straight lines directly affects the total number of polygons in the polygon set. In this last step, a ratio can be set

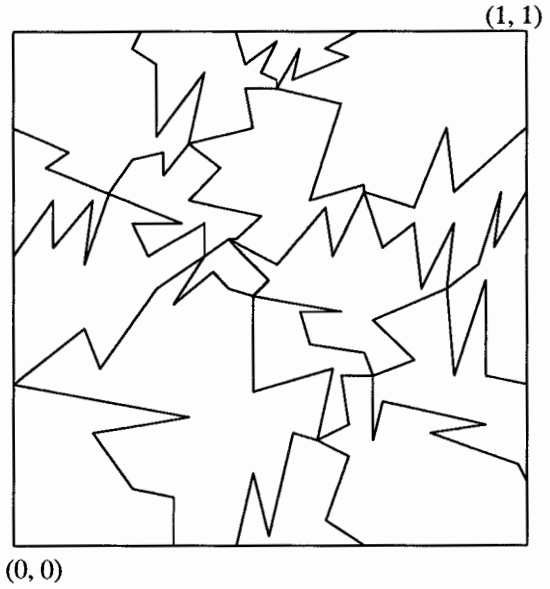

Figure 5.2: Polygon Set from Fig. 5. 1 with smoothness factor=20

to eliminate those polygons which are relatively small compared to the largest one. For example, if the ratio is set to 0.01, that means we only keep those polygons whose area sizes are at least 1% of that of the largest one.

While this method of generating random maps may not produce uniform distributed intersection points, there is no reason to doubt the randomly generated polygon sets will favor any specific algorithm. In one spatial join performance study done by Hong Fan [14], polygon sets with different distributions (in the random generation of lines) were used and no discernible difference in performance was shown in that study.

Data Set	# 1		# 2		# 3		# 4		# 5		# 6	
	Map1	Map2	Map1	Map2	Map1	Map2	Map1	Map2	Map1	Map2	Map1	Map2
# of Polygons	119	136	201	202	362	330	461	395	795	742	1178	1066
# of Chains	261	296	430	432	762	698	965	833	1648	1542	2424	2200
# of Edges	1766	2032	3048	3064	5564	5052	7118	6062	12372	11524	18440	16648
# of Vertices	1648	1897	2848	2863	5203	4723	6658	5668	11578	10783	17263	15583
# of polygon pairs from R-Tree join	833		1400		2604		3058		5791		9013	
# of intersecting polygon pairs	519		815		1507		1732		3382		5166	

Table 5.1: Data Set Statistics

5.2.2 Data Sets for the Experiment

Six data sets with varying sizes are generated for the experiments. Each set consists of two polygonal maps, each of which contains a set of non-overlapping polygons. Table 5.1 shows the statistical information of each of the six data sets generated.

5.3 Experimental Setup

There are three different data models/schemas, and two different DBMS platforms, each of which may run with or without application caching. Thus, theoretically, there should be altogether 12 combinations for our spatial join algorithm:

- 1. Relational Model on Sybase with Application Caching Off.
- 2. Relational Model on Sybase with Application Caching On.
- 3. Relational Model on ObjectStore with Application Caching Off.
- 4. Relational Model on ObjectStore with Application Caching On.
- 5. BLOB Model on Sybase with Application Caching Off.

- 6. BLOB Model on Sybase with Application Caching On.
- 7. BLOB Model on ObjectStore with Application Caching Off.
- 8. BLOB Model on ObjectStore with Application Caching On.
- 9. Parent-child Pointer Model on Sybase with Application Caching Off.
- 10. Parent-child Pointer Model on Sybase with Application Caching On.
- 11. Parent-child Pointer Model on ObjectStore with Application Caching Off.
- 12. Parent-child Pointer Model on ObjectStore with Application Caching On.

In practice, it is not possible to run the parent-child pointer model on Sybase, because Sybase does not support navigational access through pointer. Thus versions 9 and 10 do not exist. Version 12 does not exist either, since the cache data structure built by ObjectStore is already well-suited to the application and by 'de-referencing' pointer instead of query to access object the data accessing cost is already very small, nothing can be gained from application caching.

To abbreviate, we label each version as XXX/YYY/ZZZ, where XXX is the data model, YYY is the DBMS platform, and ZZZ (On, Off) indicates whether application caching is on or off. Following the Unix tradition, we use a '$*$' to present the collection of all models, or platforms.

We use the total execution time of the entire version, including the database open and close times consumed by the DBMS, to measure the performance of each version. The database load time is not included since data sets are already stored in the DBMS prior to the execution of the algorithm.

Chapter 6

Experimental Results and Analysis

In this chapter, we tabulate the experimental data from nine different versions of our spatial join operation on the six data sets given in Chapter 5 and present our analysis of these performance results. Before giving these experimental results obtained from different spatial data models and different DBMS platforms with or without application caching, we report some performance data to show the improvement of our spatial join algorithm after applying two technical strategies : utilizing topological information and utilizing MBR intersection information as mentioned in Chapter 2.

6.1 Comparison of Spatial Join Algorithms

In Chapter 2, we described our spatial join algorithm which was developed from a basic R-tree based algorithm and explained why the performance can be improved by

	Algorithm 1	Algorithm 2	Algorithm 3
R-tree Construction Time (Sec)	0.430	0.430	0.430
R-tree Join Time (Sec)	0.760	0.760	0.760
Polygon Overlap Time (Sec)	8.831	7.642	1.200

Table 6.1: Performance Results from Different Algorithms for Data Set No.1

utilizing topological information and MBR intersection information from the R-tree join. Here we show some performance data in Tables 6.1 - 6.6. In this section, we call the basic R-tree based spatial join algorithm Algorithm 1. The algorithm applied with topological information is called Algorithm 2. The final version of the algorithm which utilizes both topological information and MBR intersection information is defined as Algorithm 3. For each algorithm, the execution time consists of three timing results:

- *R-tree Construction Time*: time for creating the R-tree index
- *R-tree Join Time*: time duration for finding all potentially overlapping polygon pairs
- *Polygon Overlap Computation Time*: execution time for real polygon overlapping checking

The spatial data model and DBMS platform used for testing these three algorithms are Parent-Child Pointer Model and ObjectStore. According to our numbering of versions in Chapter 5, it is version 11.

In order to provide a good comparison, we summarize the polygon overlap computation time of all six data sets in one chart (Figure 6.1). To make the chart better looking, polygon overlap computation time is given in its logarithmic form (ln).

	Algorithm 1	Algorithm 2	Algorithm 3
R-Tree Construction Time (Sec)	0.660	0.660	0.660
R-tree Join Time (Sec)	1.420	1.420	1.420
Polygon Overlap Time (Sec)	15.074	13.051	1.880

Table 6.2: Performance Results from Different Algorithms for Data Set No.2

	Algorithm 1	Algorithm 2	Algorithm 3
R-Tree Construction Time (Sec)	1.360	1.360	1.360
R-tree Join Time (Sec)	3.100	3.100	3.100
Polygon Overlap Time (Sec)	33.736	29.198	4.080

Table 6.3: Performance Results from Different Algorithms for Data Set No.3

	Algorithm 1	Algorithm 2	Algorithm 3
R-tree Construction Time (Sec)	1.840	1.840	1.840
R-tree Join Time (Sec)	4.720	4.720	4.720
Polygon Overlap Time (Sec)	36.628	31.676	4.390

Table 6.4: Performance Results from Different Algorithms for Data Set No.4

	Algorithm 1	Algorithm 2	Algorithm 3
R-tree Construction Time (Sec)	3.590	3.590	3.590
R-tree Join Time (Sec)	9.480	9.480	9.480
Polygon Overlap Time (Sec)	67.910	58.519	8.420

Table 6.5: Performance Results from Different Algorithms for Data Set No.5

	Algorithm 1	Algorithm 2	Algorithm 3
R-tree Construction Time (Sec)	5.611	5.611	5.611
R-tree Join Time (Sec)	15.741	15.741	15.741
Polygon Overlap Time (Sec)	114.455	99.018	13.681

Table 6.6: Performance Results from Different Algorithms for Data Set No.6

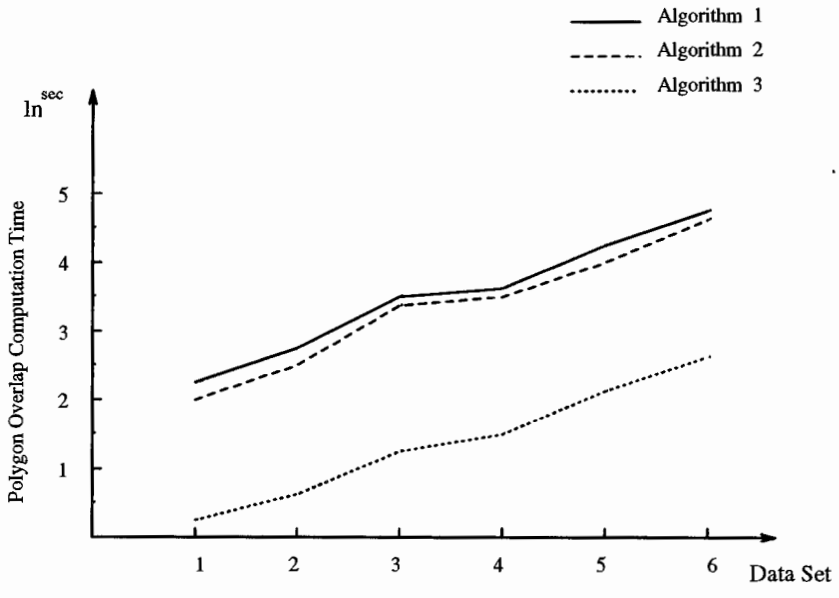

Figure 6.1: Comparison of Polygon Overlap Computation Times in Logarithmic Form

As the algorithm analysis has already been given in Chapter 2, it is not being repeated in this section. From Tables 6.1 - 6.6, it can be observed that polygon overlap computation time does not constitute a dominant part when Algorithm 3 is used. In the next section, we will report and analyze the main part of our experimental results, the spatial join performance of different spatial data models and DBMS platforms based on this algorithm – Algorithm 3.

6.2 Experimental Results

The experimental data are tabulated in Tables 6.7 - 6.12. For each version, three timing results are given:

- *DB Open Time*: time duration for the DBMS to execute the database open statement
- *Pre-load Time*: time for carrying out pre-loading by the application into the local application cache
- *Run Time*: execution time for the algorithm.

The only time that is not reported here is the DB close time, which is so small compared with the other three that it is negligible. The rest of this section is devoted to the analysis of the experimental data in these tables, supported by other relevant finer-grained data.

In this section, it is assumed that when the application caching is performed, the local application cache is large enough to accommodate a huge database. In the next

DBMS	Sybase					ObjectStore				
Application Caching	On			Off		On			Off	
	DB Open	Pre-load	Run Time	DB Open	Run Time	DB Open	Pre-load	Run Time	DB Open	Run Time
Relational Model	0.01	14.78	5.18	0.01	694.96	8.85	0.87	3.04	8.92	314.45
BLOB Model	0.01	7.24	5.10	0.01	135.69	8.96	0.76	3.04	9.11	4.75
Pointer Model	NA	NA	NA	NA	NA	NA	NA	NA	9.03	2.72

Table 6.7: Performance Result for Data Set No.1 (Sec)

DBMS	Sybase					ObjectStore				
Application Caching	On			Off		On			Off	
	DB Open	Pre-load	Run Time	DB Open	Run Time	DB Open	Pre-load	Run Time	DB Open	Run Time
Relational Model	0.01	15.70	7.00	0.01	1218.79	8.67	1.33	4.96	8.96	774.02
BLOB Model	0.01	9.69	6.81	0.01	249.84	9.06	1.10	4.89	9.01	8.49
Pointer Model	NA	NA	NA	NA	NA	NA	NA	NA	8.89	4.53

Table 6.8: Performance Result for Data Set No.2 (Sec)

section, we will give some testing results in which the database cannot be totally loaded into local application cache at once.

Do these experimental data justify our performance study? In other words, are there great differences in performance for different versions of the algorithm? The answer is of course a positive one, since there are 3 to 4 orders of magnitude differences between the best and the worst times. The data also show the effectiveness of the application caching technique. When it is on, the differences in timing are so much reduced that the timings appear similar to each other across data models and DBMS platforms.

DBMS	Sybase					ObjectStore				
Application Caching	On			Off		On			Off	
	DB Open	Pre-load	Run Time	DB Open	Run Time	DB Open	Pre-load	Run Time	DB Open	Run Time
Relational Model	0.01	19.41	11.03	0.01	2132.44	9.32	2.25	9.75	8.86	2349.88
BLOB Model	0.01	16.85	10.36	0.01	417.56	9.03	1.83	9.56	8.91	15.98
Pointer Model	NA	NA	NA	NA	NA	NA	NA	NA	8.92	9.65

Table 6.9: Performance Result for Data Set No.3 (Sec)

DBMS	Sybase					ObjectStore				
Application Caching	On			Off		On			Off	
	DB Open	Pre-load	Run Time	DB Open	Run Time	DB Open	Pre-load	Run Time	DB Open	Run Time
Relational Model	0.01	22.85	12.91	0.01	2678.95	9.03	2.81	12.29	8.95	3674.29
BLOB Model	0.01	19.62	12.39	0.01	843.92	9.14	2.48	12.22	8.87	19.16
Pointer Model	NA	NA	NA	NA	NA	NA	NA	NA	8.98	12.53

Table 6.10: Performance Result for Data Set No.4 (Sec)

DBMS	Sybase					ObjectStore				
Application Caching	On			Off		On			Off	
	DB Open	Pre-load	Run Time	DB Open	Run Time	DB Open	Pre-load	Run Time	DB Open	Run Time
Relational Model	0.01	30.89	24.56	0.01	5134.11	9.12	5.10	27.10	8.85	11246.36
BLOB Model	0.01	25.13	24.06	0.01	1707.85	8.94	4.00	26.54	8.99	38.83
Pointer Model	NA	NA	NA	NA	NA	NA	NA	NA	9.03	23.85

Table 6.11: Performance Result for Data Set No.5 (Sec)

DBMS	Sybase					ObjectStore				
Application Caching	On			Off		On			Off	
	DB Open	Pre-load	Run Time	DB Open	Run Time	DB Open	Pre-load	Run Time	DB Open	Run Time
Relational Model	0.01	57.70	37.58	0.01	7439.24	8.95	7.69	42.75	9.03	24770
BLOB Model	0.01	37.54	36.97	0.01	2619.38	8.99	7.29	42.44	8.97	75.20
Pointer Model	NA	NA	NA	NA	NA	NA	NA	NA	9.01	37.03

Table 6.12: Performance Result for Data Set No.6 (Sec)

6.2.1 Who is the Winner ?

We are reluctant to pick a winner for several reasons. Although we have been using these two database systems for many research projects, we are by no means experts and have not tried to tune the DBMS. Thus, it is likely that some performance numbers are smaller by some percentage because different techniques are applied, though we are confident that these changes would not significantly alter our findings here. Another reason is that there are more than one way to compute the total time. It would seem entirely reasonable that the Pre-load Time is included in the total time. But there are two issues about whether to include the DB open time of ObjectStore in the comparison. (Its counterpart for Sybase is so small that it is negligible.) First, we are not sure what to include, although we do have some representative numbers in the tables. This time, with each run, is almost constant across all data set sizes. Second, without the knowledge of the internal system, we have no way of knowing how much of this time is spent on tasks that are related to the execution of this algorithm, such as local caching, which serves similar purposes as the application caching. If the DB Open Time for ObjectStore is excluded from the total time, should the Pre-load Time be also excluded?

When we compare the performance by each timing category, many trends are easily identifiable. The DB Open Time of ObjectStore is quite large, in comparison with that of Sybase, and it is identical for data sets of all sizes for each set of experiments. The Pre-load Time of ObjectStore increases as the size of the data set increases, but not quite as much as that of Sybase. The sum of the DB Open Time and the Pre-load Time of Sybase is much larger than that of ObjecStore. (See Figure 6.2). The most interesting timing category is of course the Run Time, which will be used as the

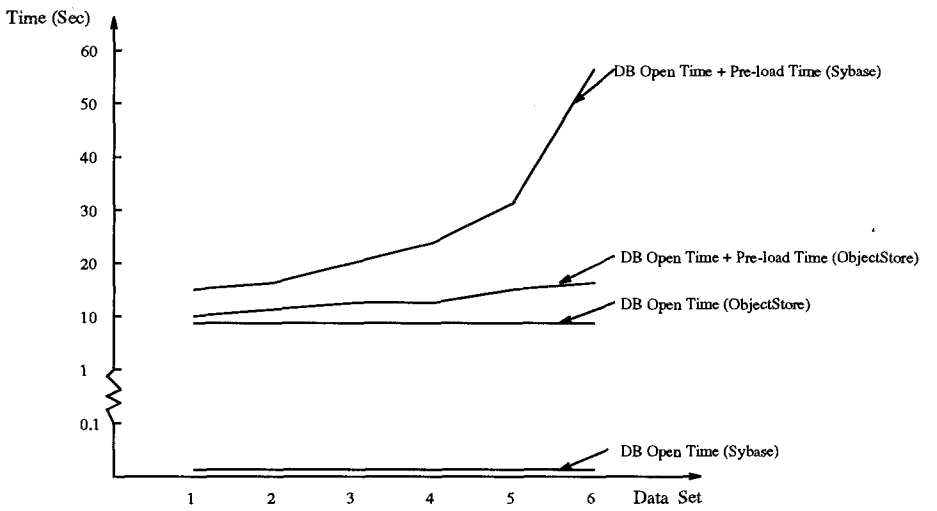

Figure 6.2: Comparison of DB Open Time and Pre-load Time on Sybase and ObjectStore

performance criteria for the rest of this section.

The combination of Pointer model and ObjectStore offers the best system environment. We call this version the Baseline Version. This is expected, since our spatial join algorithm has an application data structure that is most compatible with the Pointer model, and least compatible with the Relational model. What is unexpected is that despite the fact that ObjectStore is known for its efficiency in navigational accessing (de-referencing), the Relational/*/On and BLOB/*/On are very close to the Baseline Version in performance. In Figures 6.3 - 6.6, we compare the Run Times of all the versions to that of the Baseline Version. The Run Time is still given in its logarithmic form just like in Figure 6.1. From these four figures, the performance improvement by applying application caching is quite obvious. Without application caching, both data models (Relational and BLOB) have poor results on either RDBMS Sybase or OODBMS ObjectStore. When application caching is applied, all the four versions

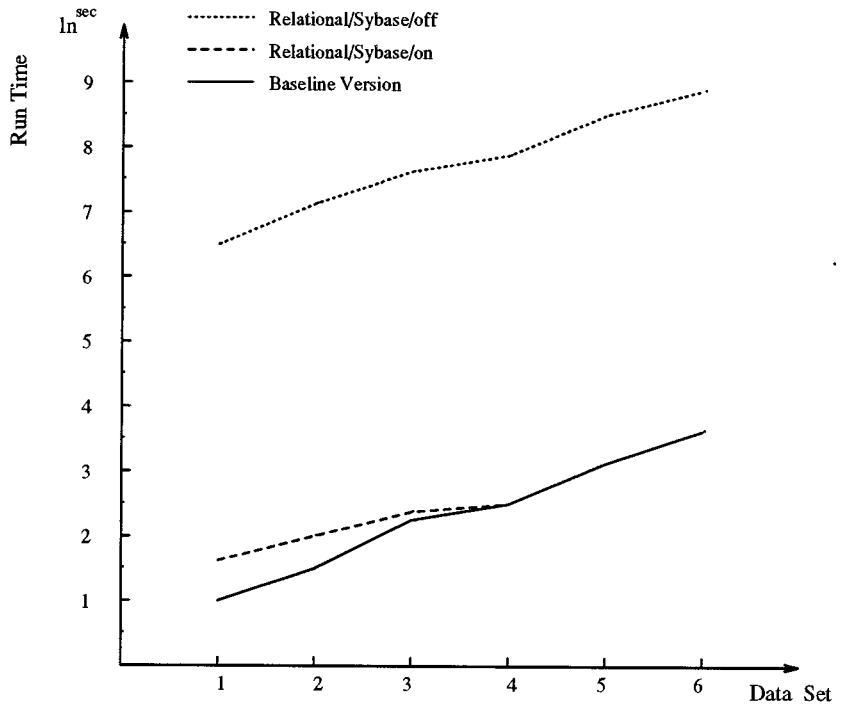

Figure 6.3: Comparison of Run Times: Relational in Sybase vs. Baseline Version

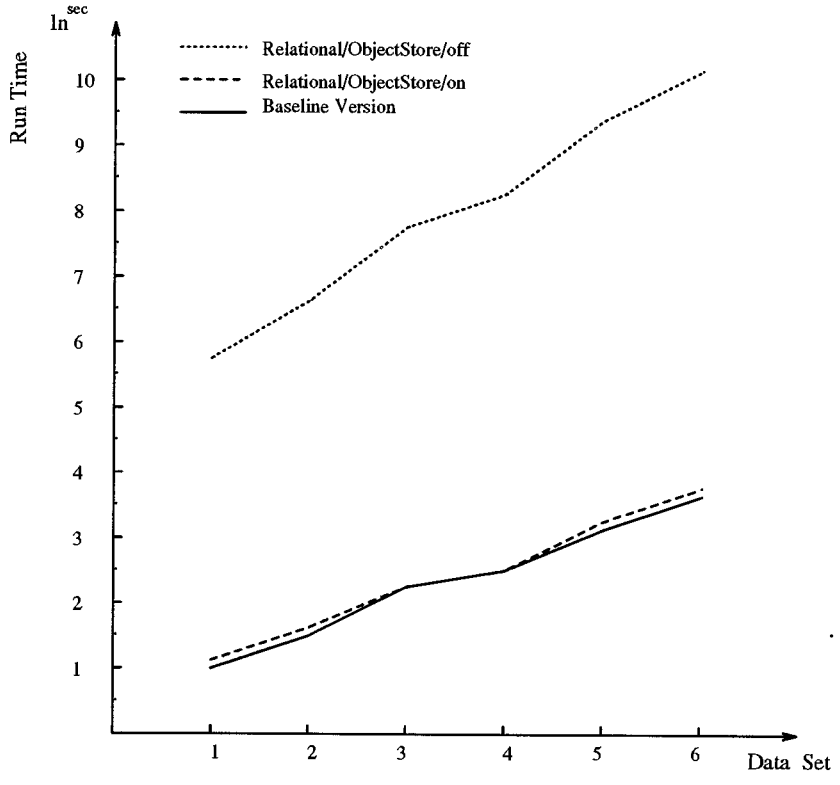

Figure 6.4: Comparison of Run Times: Relational in ObjectStore vs. Baseline Version

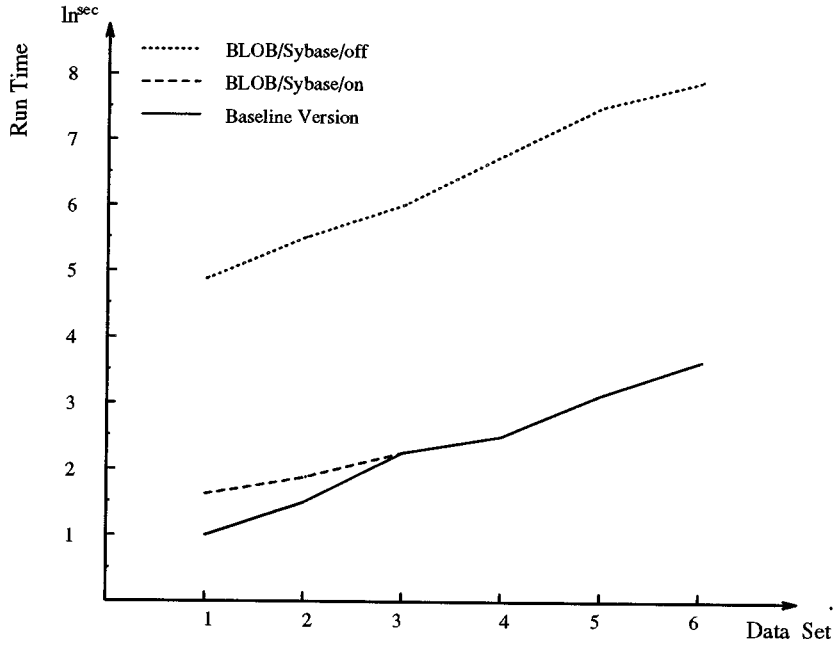

Figure 6.5: Comparison of Run Times: BLOB in Sybase vs. Baseline Version

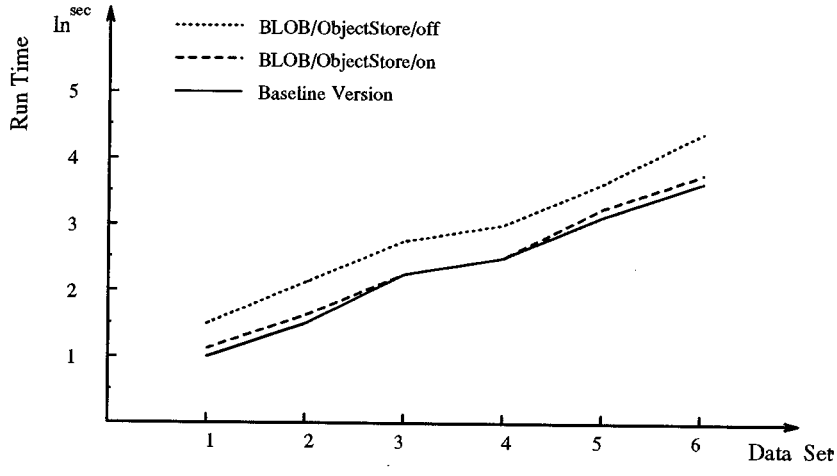

Figure 6.6: Comparison of Run Times: BLOB in ObjectStore vs. Baseline Version

have very comparable performance to the Baseline Version.

6.2.2 BLOB Model vs. Relational Model

Comparing versions of Relational model with those of BLOB model on Sybase (See Figures 6.3 and 6.5), we can make some interesting observations. In Relational model, information of each polygon is distributed in four tables (polygon, chain, edge and vertex) and each table requires normalization. In BLOB model, the polygon information is 'compressed' into two tables (polygon and chain) and each table has a BLOB attribute. Thus, the number of database accesses (the number of SQL queries) to fetch the objects from the database in Relational/Sybase/Off is about 6 times more than that in BLOB/Sybase/Off (See Table 6.13). Since SQL queries are the most time-consuming part in the total execution, the performance of Relation/Sybase/Off is obviously worse than that of BLOB/Sybase/Off. On the other hand, when we apply application caching to both of the models to cut down the SQL query time, the difference between these two models is almost negligible even though the mapping from data model to application data structure in Relation model is more complicated than that in BLOB model. The same conclusion can be drawn by comparing versions of Relational model with those of BLOB model on ObjectStore. (See Figures 6.4 and 6.6) When we make a comparison between BLOB/Sybase/Off and BLOB/ObjectStore/Off in Figures 6.5 and 6.6, we observe that the same data model results in some difference in performance. The latter is much faster than the former. ObjectStore's local caching seems to be the reason for this performance difference, since during the database open time, part of the database has already been pre-loaded into the local cache by ObjectStore. (See Chapter 4 for more information about ObjectStore's local caching.)

Data Set	1	2	3	4	5	6
Relational	17330	28090	49036	60199	112751	166760
BLOB	2910	4797	8147	10671	20502	30871

Table 6.13: Number of Queries in Relational Model and BLOB Model

However, here comes an unexpected result. From the above analysis of two data models (Relational and BLOB), it is easy to imagine that the performance of Relational/ObjectStore/Off will be worse than that of BLOB/ObjectStore/Off, but it shouldn't be that much different (see Figures 6.4 and 6.6). We will try to explain this mystery in Sec.6.2.3.

6.2.3 Performance of ObjectStore

ObjectStore in this research does not always offer good results. In fact, we are surprised that the combination of Relational model and ObjectStore with no application caching gives the worst performance. Our interpretation of this is that ObjectStore is optimized toward 'de-referencing' rather than querying as a persistent data access mechanism, while querying is the only means to access data in Sybase. Because of this reason, we can explain why the performance difference between Relational/ObjectStore/Off and BLOB/ObjectStore/Off is bigger than that between Relational/Sybase/Off and BLOB/Sybase/Off. As we see from Table 6.13, the number of queries in Relational model is 6 times more than the number in BLOB model. So many queries are really a burden to ObjectStore. Therefore, the performance of Relational/ObjectStore/Off is much worse than that of BLOB/ObjetStore/Off. Tables 6.14 and 6.15 show the estimated costs of querying for the Relational/Sybase/Off and

Date Set	1	2	3	4	5	6
No. of queries	17330	28090	49036	60199	112751	166760
Time difference (sec)	690	1212	2121	2666	5110	7402
Time per query (sec)	0.04	0.04	0.04	0.04	0.04	0.04

Table 6.14: Querying Cost: Relational/Sybase/Off

Date Set	1	2	3	4	5	6
No. of queries	16193	25569	45282	55348	101069	149415
Time difference (sec)	311	769	2341	3662	11219	24727
Time per query (sec)	0.02	0.03	0.05	0.07	0.11	0.16

Table 6.15: Querying Cost: Relational/ObjectStore/Off

Relational/ObjectStore/Off. To arrive at an estimated querying cost, we calculate the difference between the run times of Relational/*/On and Relational/*/Off to first estimate the cost of processing all queries, which is then divided by the total number of queries. On Sybase, the average query access cost is almost constant across data sets of different sizes. On ObjectStore, the average increases rapidly as the size of data set increases. In comparison with the Sybase version, it is half as much for the smallest data set, and increases to four times as much for the largest data set.[1] This result shows why ObjectStore's performance deteriorates much faster than Sybase as the data set size increases.

[1]To verify this unexpected result, we tried indexing on both Sybase and ObjectStore. But the results do not help us to answer the question.

6.2.4 Application Caching

Application caching as a bridge between the application data structure (or access requirements) and the incompatible storage data structure (or data model) is effective in drastically improving the performance of spatial join, at a very modest cost. When it is applied, the difference in the data models seems to almost disappear. For example, the performance of Relational/Sybase/On and Relational/ObjectStore/On are only marginally worse than that of Pointer/ObjectStore/Off, which is the most optimized version. The performance of the Relational model and the BLOB model, with application caching on, are practically indistinguishable. We will give more experimental data and more discussion on application caching in the following section.

6.3 More Discussion on Application Caching

In the above section, we showed the effectiveness of the application caching technique. When it is applied, no matter what data models and DBMS platforms we are using, the performance can be improved substantially. However, all the performance data we presented in the last section are based on the assumption that the local application cache is large enough to store the entire database. While, in many real applications, this condition may not be true as the database can be humongous. In this section, we abandon this assumption and perform the application caching in a more realistic situation.

We still use those 6 data sets as our testing data. For each data set we preload 100%, 90%, 75% and 50% of database respectively. In Table 6.16 we report the experimental data. The algorithm version used is the combination of the Relational

Data Set	1	2	3	4	5	6
Loading 100%	3.20	3.77	5.20	5.92	10.16	15.01
Loading 90%	12.61	31.96	29.39	38.47	98.05	105.45
Loading 75%	41.21	86.29	125.57	141.81	303.77	405.86
Loading 50%	100.91	176.80	327.02	478.70	659.88	955.97

Table 6.16: Run Times in (Sec) with Different Database Loading Percentage

Data Set	1	2	3	4	5	6
Loading 100%	0	0	0	0	0	0
Loading 90%	546	1635	1382	1516	4734	4715
Loading 75%	2239	4809	6507	6707	16083	21954
Loading 50%	5616	9917	16126	18789	33842	51117

Table 6.17: Number of Queries Executed Agaist Different Data Sets

data model and Sybase with application caching on. This time we have a different implementation environment. The Sybase server is running on a Sun 4/75(SS2) with 16 MB memory. The client runs on a separate machine Sun 4/50(IPX) with 16 MB memory. Since we are only interested in the Run Time differences among the 4 different loading sizes, the changed environment will not bias the results. We also show the number of queries that are required to access the data when they do not reside in the local cache. (See Table 6.17) This number is affected by three factors :

- The size of database
- The percentage of the pre-loaded section
- The distribution of polygons in each map

The first two factors are obvious. The last one can be explained by an example in

Table 6.17. In the second row of that table (loading 90%), the number of queries for data set 2, 3, 4 is 1635, 1382, and 1516, respectively, even though the size of data set increases from data sets 2 to 4. This fact indicates that the polygons in data set 3 and 4 have a better clustering than those in data set 2. Therefore, fewer queries are required.

Data in Table 6.16 show that there is a big gap between loading the whole database and loading 90% of the database. This proves that the query call is a major cost in the total execution. Comparing Table 6.16 with the performance of Relational/Sybase/Off in Tables 6.7 - 6.12, we can also observe that when only 50% of the database is loaded, we can still gain performance improvement by applying application caching. It is observed that the advantage of application caching is not fully utilized by the spatial join application. This is because in spatial join, all the polygons in both maps are required to be accessed to find out how many polygon pairs overlap. Thus, if only part of the database is loaded, data swap-in-and-swap-out during computation is unavoidable. We claim that in some other applications which do not require to scan entire database application caching is an effective strategy to achieve better performance.

Chapter 7

Conclusions

7.1 Thesis Summary

In order to study how the spatial data model together with its underlying database platform affect the performance of the complex spatial operation, we designed and implemented three spatial data models on two different DBMS platforms. The three data models have different degrees of pointer involvement which means the data accessing will be different (querying or pointer 'de-referencing').

The spatial join operation, a specific application we used for the performance study in this thesis, is very efficient after applying new techniques: utilizing topological information and utilizing MBR intersection information from the first filter step. Therefore, we guarantee that our spatial join operation is a suitable candidate for our research.

We have shown that there can be interesting observations to be made out of this

performance study, although *care must be taken to generalize these findings beyond the context of the application.* Some of them confirm the conventional wisdom. For example, the more the data model resembles the application data structure, the faster the algorithm is. For that reason, OODBMS can provide superior performance for spatial database operations due to its capability of including pointers in the schema level data model. By using BLOB data type in the data model, spatial information is 'compressed' into a smaller number of tables than in the relational model, so that better performance can be achieved. On the other hand, we have managed to show that by applying the application caching technique, the relational model and BLOB model on either OODBMS or RDBMS platform can provide acceptable performance. The advantage of conducting application caching is two folds :

- Since the application caching is managed by the application, the cache data structure can be tailored to the need of the application.
- By pre-loading bulk of the data into the application cache before the objects are required by the application, the high query cost can be reduced.

7.2 Contributions

We have taken a rather unusual approach to performance study, i.e., considering the impact of system environment (data models and DBMS platforms) on the performance of a specific application.

- Three spatial data models were designed : Relational Model, BLOB Model and Parent-Child Pointer Model.

- A typical spatial database operation - spatial join, was selected as our specific application. We carefully designed and implemented an efficient R-tree based spatial join algorithm to make it a suitable candidate for our experimental purposes.
- We carefully designed the map data generation, implementation environment and different testing versions of the same algorithm to make the results reasonable.
- We introduced the application caching technique to bridge the gap between the spatial data model and the application data structure. Our performance results showed that by applying this technique to the Relational and BLOB models the effect of data model/DBMS platform can be neutralized.

7.3 Future Work

Much more research is needed in order to perfect the application caching technique. Further investigation about it is required to determine how well it works with other applications. We believe the same conclusion can be obtained. Although the technique as described here is readily implementable, it works only in read-only situations, and more work is also required in the area of concurrency control, when the persistent data in a multi-user environment are modifiable.

At present, OODBMS is the only kind of DBMS that can provide acceptable performance for specialized applications that require fast graph traversals. With the perfection of application caching technique, it is possible that an RDBMS can be an acceptable alternative.

Bibliography

[1] N. Beckmann, H. Schneider, and B. Seeger. "The R*-Tree: An Efficient and Robust Access Method for Points and Rectangles", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Altantic City, N. J., pages 322-331, May 1990.

[2] J. L. Bentley and D. Wood. "An Optimal Worst Case Algorithm for Reporting Intersections of Rectangles", *IEEE Transactions on Computers*, Vol. C-29, No. 7, pages 571-577, July 1980.

[3] J. L. Bentley and T. Ottmann. "Algorithms for reporting and counting geometric intersections", *IEEE Transactions on Computers*, Vol. C-28, pages 643-647, Sept. 1979.

[4] T. Brinkhoff, H. P. Kriegel, and R. Schneider. "Comparison of Approximations of Complex Objects Used for Approximation-based Query Processing in Spatial Database Systems", *Proc. 9th Int. Conf. on Data Engineering*, pages 40-49, Vienna, 1993.

[5] R.G.G. Cattell. *Object Data Management*, Addison-Wesley, Reading, Mass., 1991.

[6] R.G.G. Cattell. *An Engineering Database Benchmark*, pages 247-282, Springer-Verlag, 1991.

[7] C.J. Date. *An Introduction to Database Systems*, Vol. 1, 5th edition, Addison-Welsey, 1990.

[8] D.J. DeWitt, D. Maier, P. Futtersack, and F. Velez. "A Study of Three Alternative Workstation-Server Architectures for Object Oriented Database Systems", *Proc. 16th Int. Conf. on VLDB*, Brisbane, Australia, pages 107-121, 1990.

[9] C. Faloutsos, T. Sellis, and N. Roussopoulos. "Analysis of Object Oriented Spatial Access Methods", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 426-439, San Francisco, 1987.

[10] O. Günther. "The Design of the Cell Tree: An Object-Oriented Index Structure for Geometric Databases", *Proc. 5th Int. Conf. on Data Engineering*, Los Angeles, CA., pages 508-605, 1989.

[11] O. Günther. "Efficient Computation of Spatial Join", *Proc. 9th Int. Conf. on Data Engineering*, Vienna, pages 50-59, 1993.

[12] A. Guttman. "R-trees: A Dynamic Index Structure for Spatial Searching", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Boston, pages 47-57, 1984.

[13] E. G. Hoel and H. Samet. "A Qualitative Comparison Study of Data Structures for Large Line Segment Databases", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, San Diego, CA., pages 205-214, 1992.

[14] H. Fan. "Spatial Join: A Study of Complex Spatial Operation and Its Underlying Spatial Indexing Methods", Master Thesis, School of Computing Science, Simon Fraser University, Burnaby, B. C., Canada, 1992.

[15] H. V. Jagadish. "Spatial Search with Polyhedra", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, San Diego, CA., pages 205-214, 1992.

[16] J. Wu. "Implementation and Evaluation of Dynamic Spatial Query Language", Master Thesis, School of Computing Science, Simon Fraser University, Burnaby, B. C., Canada, 1992.

[17] H. F. Korth and A. Silberschatz. *Database System Concepts*, Second Edtion. McGraw-Hill Inc., 1990.

[18] D. J. Maguire, M. F. Worboys and H. M. Hearnshaw. "An Introduction to Object-Oriented Geographical Information Systems", Mapping Awareness, 4(2), pages 36-39, 1990.

[19] M. Goodchild, K. Kemp, and T. Poiker. *Core Curriculum on GIS*, National Center for Geographic Information Analysis, University of California, Santa Barbara, 1990.

[20] J. Nievergelt, H. Hinterberger and K. C. Sevcik. "The Grid File: An Adaptable, Symmetric Multikey File Structure", *ACM Transactions on Database Systems*, Vol. 9, No. 1, pages 38-71, 1984.

[21] J. Orenstein, S. Haradhvala, B. Margulies, and D. Sakahara. "Query Processing in the Objectsotre Database System", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, San Diego, Calif., pages 403-412, 1992.

[22] J. A. Orenstein and F. A. Manola. "PROBE Spatial Data Modeling and Query Processing in an Image Database System", *IEEE Transactions on Software Engineering*, Vol. 14, No. 5, pages 611-629, May 1988.

[23] J.A. Orenstein. "Spatial Query Processing in an Object-oriented Database System", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Dallas, Texas, pages 326-336, 1986.

[24] J. A. Orenstein. "Redundancy in Spatial Databases", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Portland, Oregon, pages 294-305, June 1989.

[25] P. J. M. Oosterom. "Reactive Data Structures for Geographic Information Systems", Ph. D. thesis, Dept. of Computer Science at Leiden University, Netherlands, 1990.

[26] J. T. Robinson. "The K-D-tree: A Search Structure for Large Multidimensional Dynamic Indices", *Proc. ACM SIGMOD Int. Conf. on Management of Data*, Ann Arbor, MI, pages 10-18, 1981.

[27] D. F. Rogers, *Procedural Elements for Computer Graphics, McGraw-Hill Book Company*, 1985.

[28] N. Roussopoulos and C. Faloutsos. "An Efficient Pictorial Database System for PSQL", *IEEE Transactions on Software Engineering*, Vol. 14, No. 5, pages 639-650, May 1988.

[29] H. Samet. "The Quadtree and Related Hierarchical Data Structures", *ACM Computing Surveys*, 16, pages 187-260, June 1984.

[30] H. Samet. *The Design and Analysis of Spatial Data Structure*, Reading, Addison-Wesley, MA. 1990.

[31] H. Samet. *Applications of Spatial Data Structure: Computer Graphics, Image Processing, and GIS*, Reading, Addison-Wesley, MA. 1990.

[32] Hans-Jorg Schek and Marc H. Scholl. "From Relations and Nested Relations to Object Models", *Proc. 2nd Symp. on the Design of Spatial Database*, Zurich, Switzerland, pages 1-19, 1991.

[33] M. Schiwietz. "Organization and Query Processing of Spatial Objects", Ph. D. thesis, Institute for Computer Science, University of Munich, 1993.

[34] M. I. Shamos and D. Hoey. "Geometric Intersection Problems", *Proc. 17th Annu. IEEE Symp. Foundations of Computer Science*, pages 208-215, 1976.

[35] *Transact-SQL User's Guide*, Sybase Corporation, Emeryville, CA.; 1991.

[36] A. T. Teng, S. A. Joseph and A. R. Shojaee. "Polygon Overlay Processing: A Comparison of Pure Geometric Manipulation and Topological Overlay Processing", *Proc. 3rd Int. Symp. on Spatial Data Handling*, Vol. 1, pages 102-119, 1988.

[37] Bureau of the Census, Washington, DC. *TIGER/Line Precensus Files: 1990 Technical Documentation*, 1989.