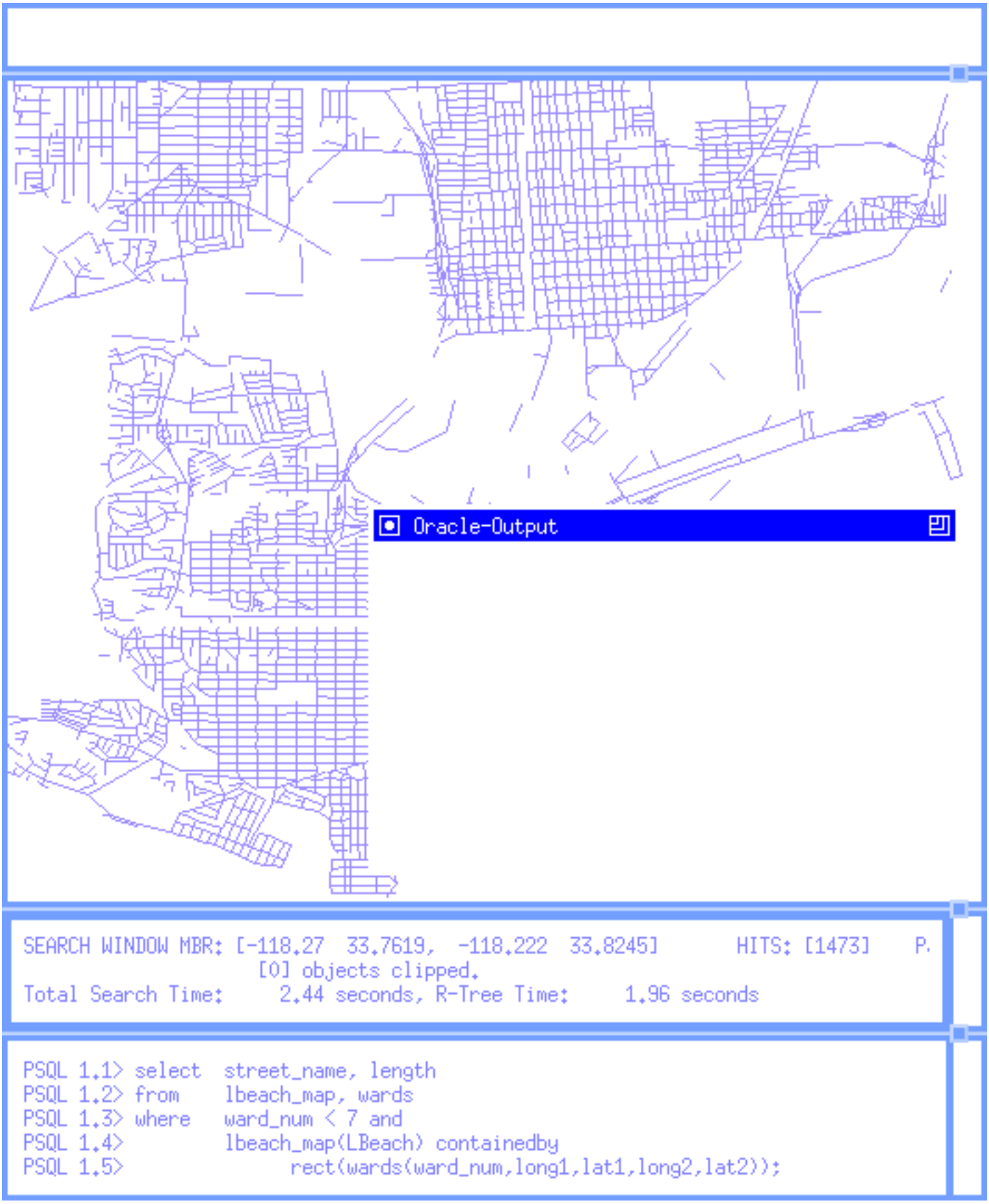

Figure 7: Spatial Join in PSQL

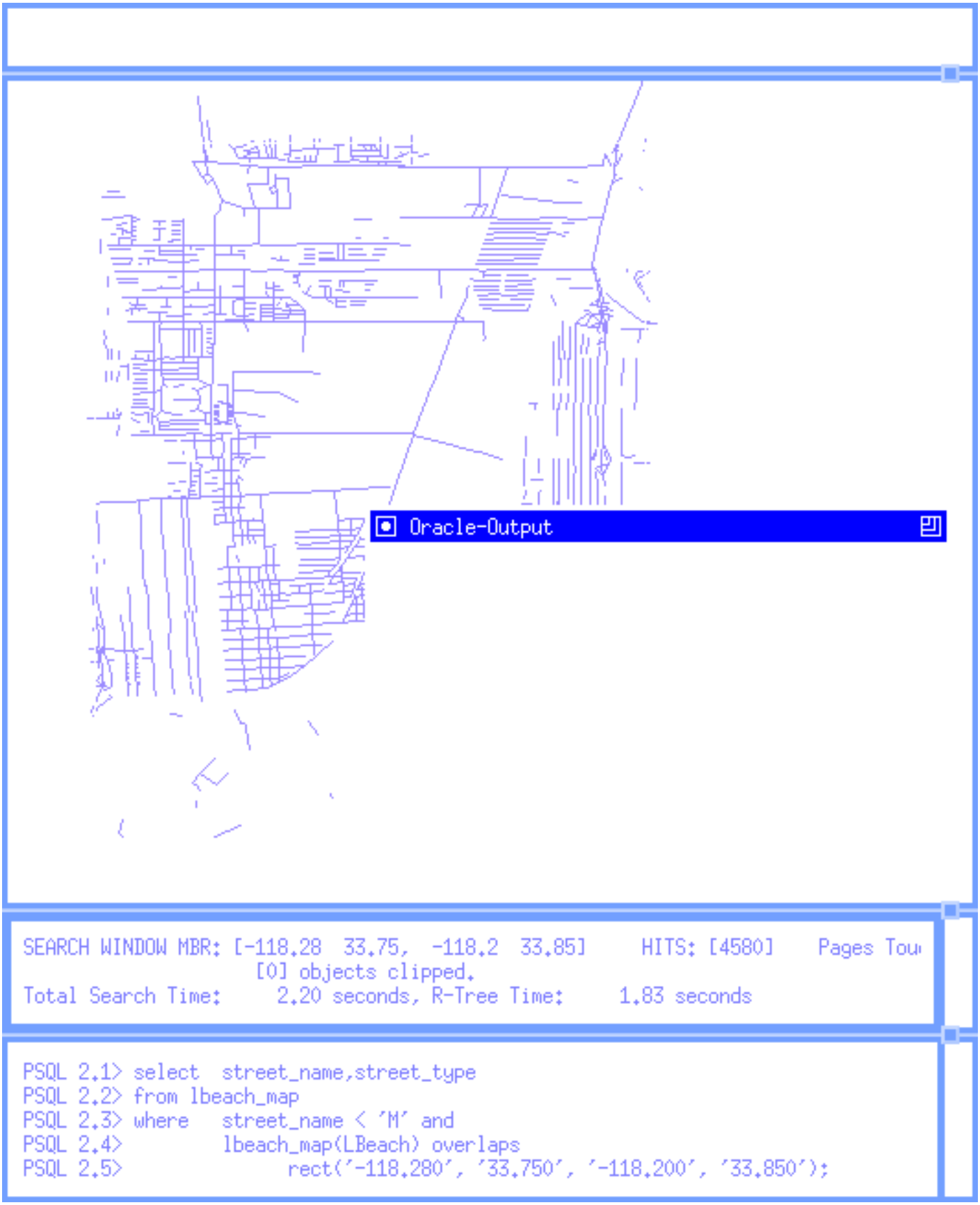

Figure 6: Spatial Select in PSQL

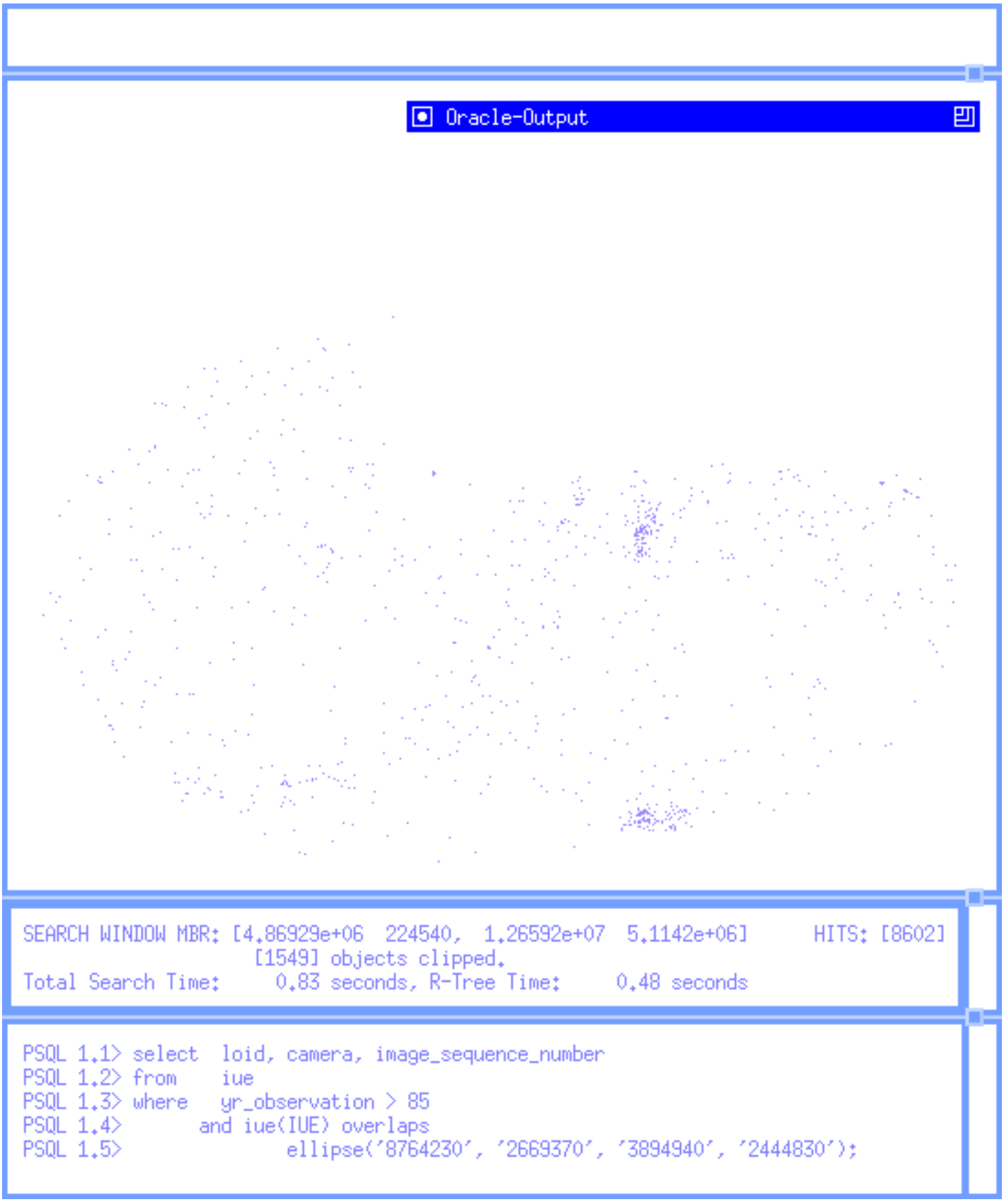

Figure 5: Spatial Select in PSQL

[SeFaRo87]

Sellis, T., Roussopoulos, N., Faloutsos, C., “The R+tree: A Dynamic Index for Multi-Dimensional Objects,” Proceedings of the 13-th *International Conference on Very Large Data Bases*, Brighton, England, September 1-4, 1987.

6.0 Conclusions

We believe that spatial queries supported by GIS and alphanumeric queries provided by RDBMSs should be integrated into a common language which provides a uniform interface to both, but their representation and processing must be clearly distinguished. In this paper we described PSQL, a modest GIS system for ORACLE. PSQL allows both spatial objects to be represented, stored and queried in their analog form, as well as offering the user to do direct query formulation and manipulation on the alphanumeric database.

.Spatial search and spatial joins are supported by variations of R-trees, which are excellent search devices for spatial objects and their relationships in multi-dimensional space. The main features of the PSQL software package are that it requires no modification to the ORACLE databases and that it is very easy to use.

7.0 References

[Gutt84]

Guttman, A.,“R-Trees: A Dynamic Index Structure for Spatial Searching,” Proc. ACM SIGMOD, Boston, Massachusetts, June 1984.

[FaSeRo87]

Faloutsos, C., Sellis, T., Roussopoulos, N.,“Analysis of Object Oriented Spatial Access Methods,” Proceedings of the *ACM-SIGMOD International Conference on Management of Data*, San Francisco, May 28-30, 1987.

[RoKeVi]

Roussopoulos, N., Kelley, S., Vincent, F., “Nearest Neighbor Queries”, *Proceedings of the ACM-SIGMOD International Conference on Management of Data,* San Jose, May 22-25, 1995

[RoLe85]

Roussopoulos, N., Leifker, D., “Direct Spatial Search on Pictorial Databases Using acked R-Trees,” Proceedings of the *ACM-SIGMOD International Conference on Management of Data*, May 28-31, 1985, Austin Texas.

[RoSeFa88]

Roussopoulos, N., Faloutsos, C., Sellis, T., “An Efficient Pictorial Database System for PSQL,” *IEEE Trans. on Software Engineering*, Vol. 14, No 5, May 1988, pp. 639-650.

The next query illustrates a qualified selection from the Long Beach database using a constant region specification:

```
select    street_name, street_type
from      lbeach_map
where     street_name < 'M'
           and lbeach_map(LBeach) overlaps
                    rect('-118.280', '33.750', '-118.200', '33.850');
```

The output of the above query is shown in Figure 6.

5.2.2 Spatial Join Queries

Spatial joins are an important and potentially *very* expensive class of queries. Records from the relations are joined if they all satisfy the underlying spatial operation conditions. The following query illustrates the spatial join of the Long Beach street relation with the one on the fictitious political wards:

```
select    street_name, length
from      lbeach_map, wards
where     ward_num < 7
           and lbeach_map(LBeach) containedby
                    rect(wards(ward_num,long1,lat1,long2,lat2));
```

The output of the above query is shown in Figure 7.

As can be seen, spatial joins allow *spatial overlays* or the synthesis of information stored in multiple tables, yet referring to the same spatial objects. We are currently implementing similar overlays to include bitmaps as well as spatial objects.

5.2.3 Nearest Neighbor Queries

A frequently encountered type of query in Geographic Information Systems is to find the k nearest neighbor objects to a given point in space. Processing such queries requires substantially different search algorithms than those for location or range queries [RoKeVi].

Our system implements an efficient branch-and-bound R-tree traversal algorithm to find the nearest neighbor object to a point, and then generalize it to finding the k nearest neighbors.It also uses metrics for an optimistic and a pessimistic search ordering strategy as well as for pruning.

```
select    street_name,street_type

from      lbeach_map

where     lbeach_map(LBeach) nearest point('-118.19', '33.78');
```

5.1.3 Dropping an R-tree Index from PSQL

To drop a spatial index from PSQL we added the following syntax to the standard SQL drop statement:

drop rindex <index-name>

where <index-name> is the name of the R-tree index to be dropped.

5.2 Data Manipulation in PSQL

PSQL supports both select and update statements of SQL. Update statements are captured and decomposed into a sequence of single tuple modifications which are then passed to ORACLE. If the ORACLE transactions complete normally, the equivalent R-tree modifications are passed to the R-tree Index System. The select statement has been extended to accept spatial expressions in the target list and the where clause qualification:

select <target-list>
from <relation-list>
where <qualification>

where

<qualification>=<spatial-expression> | <regular-SQL-expression>

<spatial-expression> = <spatial-operand> <spatial-operator> <spatial-operand>

<spatial-operand> = <table-name>(<index-name>) | <table-name>(loid, <spatial-attribute-list>) | <object-type>(<table-name>(loid, <spatial-attribute-list>)) | <object-type>(<spatial-value-list>) <spatial-operator> = overlaps | contains | containedby | overlay

A spatial expression is a predicate consisting of a spatial operator and two spatial operands and may be specified in a PSQL statement whenever a non-spatial predicate might be found in a standard SQL statement- e.g. in the where clause of a *select, insert, delete, or update* statement.

5.2.1 Spatial Search Queries

The following query illustrates a spatial selection. It selects observation information from the NASA IUE database which are spatially restricted to a fixed ellipitical region and restricted by date to more recent observations.

```
select   loid, camera, image_sequence_number
from     iue
where    yr_observation > 85 and
         iue(IUE) overlaps
                  ellipse('8764230', '2669370', '3894940', '2444830');
```

The output of the above query is shown in Figure 5, at the end of this paper.

R-trees use rectangles in 2-D space, parallelepipeds in 3-D space, and hyper-parallelepipeds in higher dimensions to index the underlying search spaces. Since some of the PSQL data types may partially overlap with the space of the above spatial objects, special purpose *clipping functions* are used by the spatial operators to discover overlap and containment. These clipping functions are applied after the search on all the returned by the R-tree objects.

For illustration purposes, we will use three spatial relations stored in an ORACLE database. The first relation is a street network from the US Census Bureau which stores all the street segments of the Long Beach California area. The data type is segment. The spatial attributes are the longitude and latitude of the end-points. The second relation stores Wards (fictitious political units) in the Long Beach area and it is of type rectangle. The third relation is from an Astrophysics database used for the NASA International Ultraviolet Explorer project. It stores observations of stars in the sky whose coordinates are given by Right Ascension (RA) and Declination (DEC). This is a point data type. Their schema are:

LBEACH_MAP (NODE1, NODE2, STREET_ID, LENGTH, FILE_NO, STREET_NAME, STREET_-TYPE, CLASS_CODE, L_LEFT, H_LEFT, L_RIGHT, H_RIGHT, PL_LEFT, PL_RIGHT, SOUNDX, DIR_CODE, LONGIT1, LATIT1, LONGIT2, LATIT2)

WARD (WARD_NUM, WARD_NAME, LONG1, LAT1, LONG2, LAT2)

IUE (LOID, OID, PID, RA,DEC, MAGNITUDE, EBV_TYPE, MAG_1000, SPECTRAL_TP, LUMINOC-ITY, OBJECT_CLASS, FES_MODE, ES_COUNTS, CAMERA, IMAGE_SEQUENCE_ DIS-PERSION, APERTURE, LARGE_APERTURE_STATUS, EXPOSURE_LENGTH, YR_OBSERVATION, DAY_OBSERVATION, HR_OBSERVATION, MIN_OBSERVATION, ACQUISITION_STATION, STATUS_NOTE, YR_PROCESSING, DAY_PROCESSING, COM-MENTS)

5.1.2 Creating an R-tree Index from PSQL

To enable the user to create an R-tree index based upon some set of spatial attributes of his or her data, we added the following syntax to the standard SQL create statement:

> create rindex [unique] <index-name> of type <object-type> on <table-name>(loid,<spatial-attribute-list>)
>
> <object-type>= point | hseg | segment | ect | circle | ellipse | polygon
>
> <spatial-attribute-list>= <spatial-attribute1>, <spatial-attribute2>[,...]

where <index-name> may be any legal identifier in SQL and <table-name> is the name of the relation in ORACLE on which a spatial index is to be built. If loids uniquely identify the table row, the unique keyword will make the access more efficient. The number of spatial attributes in the spatial attribute list depends upon the spatial object type - i.e. the point object type requires 2 attributes be specified in a 2-dimensional index whereas rect requires 4 attributes be specified. The following are examples of rindex creation:

> create rindex IUE of type point on iue(loid,ra,dec)
>
> create unique rindex LBeach of type segment on lbeach_map(street_id, longit1, latit1, longit2, latit2);

down the search from *h* to *hN*, where *h* is the height of the tree. Clearly, the dynamic splits of R-trees, may cause some degradation on the performance of the search.

It has been shown, that zero overlap and coverage is only achievable for data points that are known in advance and, that using a packing technique for R-trees, search is dramatically improved [RoLe85]. In the same paper it is shown that zero overlap is not attainable for region data objects. A variation of the basic R-trees are the *R+*trees [SeFaRo87] which avoid overlap at the leaf level at some small extra cost in the overall space requirements. This achieves less traversals of paths from the root to the leaf but slightly higher heights in the trees. Detailed comparison of these variations are beyond the scope of this paper but can be found in [SeFaRo87] and [FaSeRo87].

5.0 The Language of PSQL

PSQL was introduced to cope with the problem of having to learn a specialized GIS language to do spatial searches in the databases. Since no standards have been accepted for GIS, almost any implementation of such a system has its own query language intertwined with graphical interfaces. This makes it very difficult to learn. The biggest asset of PSQL is that an SQL-trained database user would be able to learn the spatial search extensions in a matter of minutes and use it right away. The other premise of it is that, since PSQL is a superset of SQL, generic SQL queries to the database would be recoginized as such and passed directly to the underlying RDBMS where they would be handled appropriately. The user can execute all his or her queries and updates within the domain of one application; both spatial and non-spatial types.

5.1 Data Definition in PSQL

5.1.1 Spatial Domains & Relations

The relational model requires that relations be defined over a set of domains each of which has one or another form of an alphanumeric data type. PSQL extends the definition of relations over spatial domains using an abstract data type approach.

Every domain in PSQL is an abstract data type. The spatial comparison operators and functions defined on each spatial domain hide from the user the low level implementation details which deal with the encoding of the low level representation of the domain. The advantage of this approach is that the representation of a domain can change (e.g. increase the resolution of images) *without affecting the relations defined over them.* In the sequel, domain and data type of the domain are used interchangeably.

Currently, PSQL supports seven basic spatial domains: points, line segments, horizontal line segments (useful for modelling duration of events on a time line), circles, ellipses, rectangles, and polygons. Segments and polygons are considered as objects whose internal representation and discretization is not explicitly modeled by the relational primitives of PSQL, but using special purpose external data structures appropriate for each domain. In addition to these basic spatial domains, PSQL also supports all the alphanumeric types supported by SQL.

(LOIDs) as it is for the case of ORACLE) or Tuple IDentifiers (TIDs) which are direct pointers to the internal storage of the RDBMS.

Non-leaf R-tree nodes contain entries of the form

(*RECT*, *p*)

where *p* is a pointer to a successor node in the next level of the -tree, and *RECT* is a minimal rectangle which bounds all the entries in the descendent node. The term *branching factor* (also called f*an-out*) can be used to specify the maximum number of entries that a node can have; each node of an R-tree with branching factor four, for example, points to a maximum of four descendents (among non-leaf nodes) or four objects (among the leaves). To illustrate the way an R-tree is defined on some space, Figure 3 shows a collection of rectangles and Figure 4 the corresponding R-tree built for a branching factor of 4.

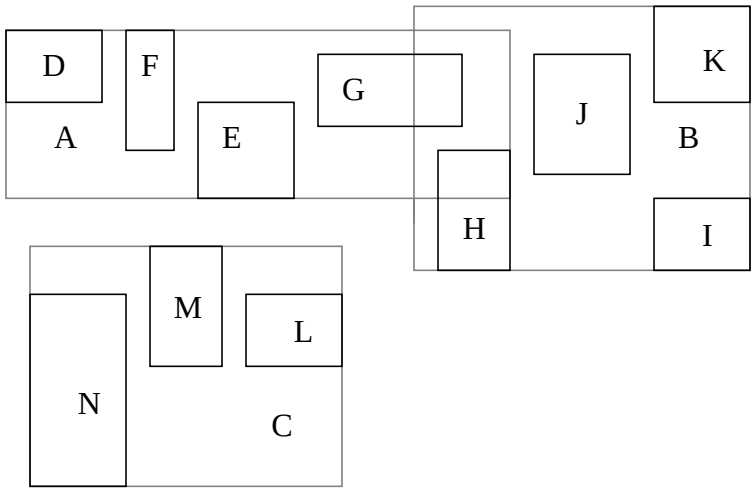

Figure 3: Some rectangles organized in an R-tree

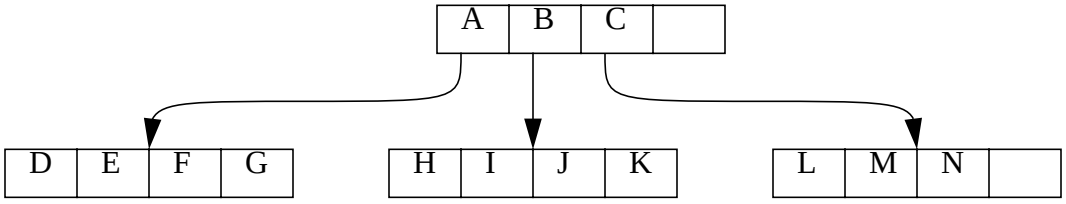

Figure 4: The Corresponding R-Tree

In considering the performance of R-tree searching, the concepts of *coverage* and *overlap* [RoLe85] are important. Coverage is defined as the total area of all the MBR's of all leaf R-tree nodes, and overlap is defined as the total area contained within two or more leaf MBR's. Obviously, efficient R-tree searching demands that both overlap and coverage be minimized. Minimal coverage reduces the amount of dead space covered by the leaves and eliminates at a very high level in the R-tree, fruitless searches in such space. Minimal overlap seems to be even more critical than minimal coverage. For a search window falling in the area of *N* overlapping leaves, in the worst case, *N* paths from the root to each of the overlapping leaves have to be followed slowing

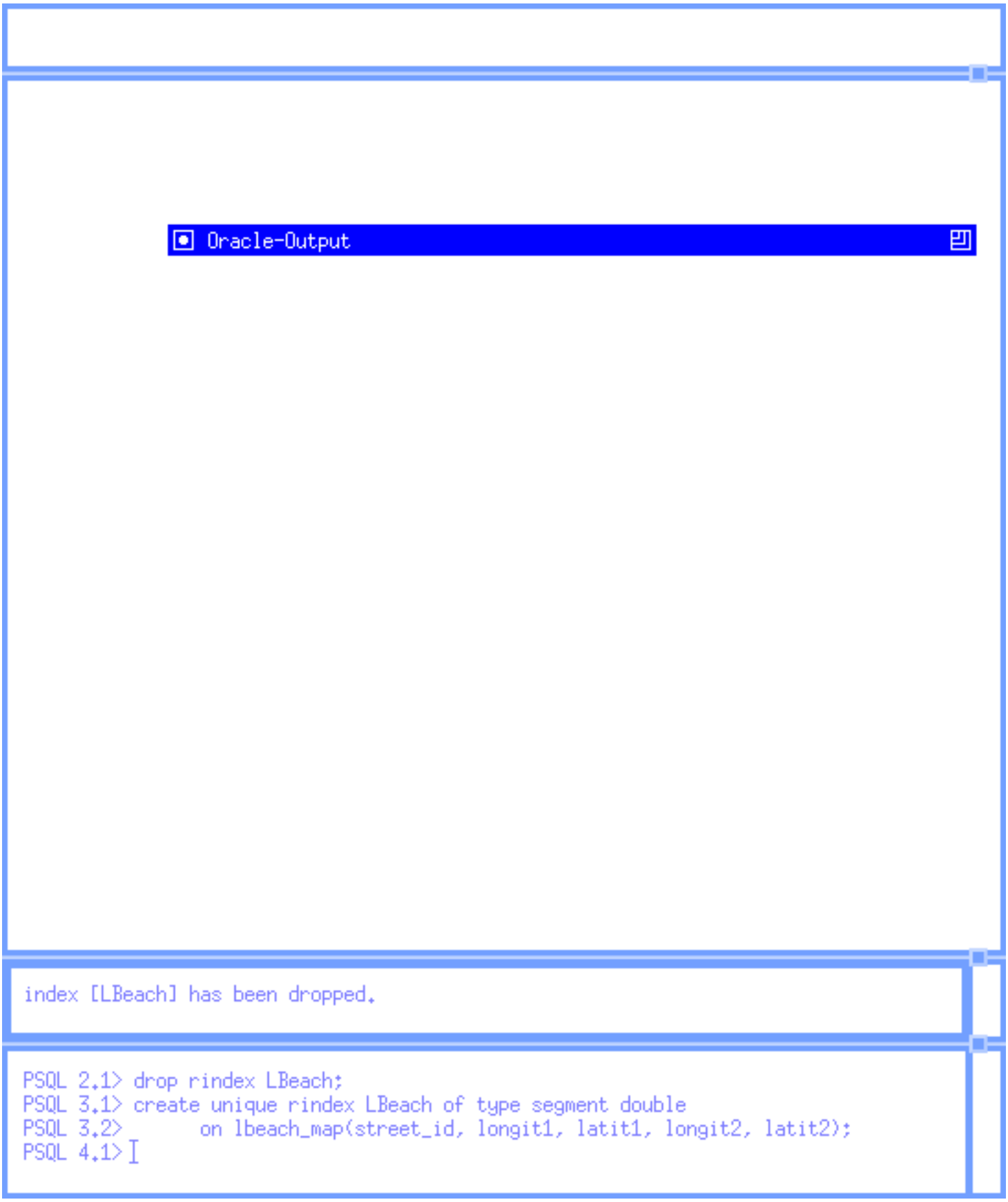

Figure 2: The X Interface of PSQL

between ORACLE and the R-tree Index System provided by the Interoperability Services maintains open a connection to ORACLE and uses buffering techniques for improving performance.

Overall, this connection performs quite well despite the indirection and the heavy overhead of exporting from ORACLE values for the spatial search and then importing to it LOIDs for the final processing. The reason is that the high search performance of the R-tree indexes offsets by far the cost of searching combined attribute B-tree indexes in ORACLE, especially in higher dimensions. Incorporation of PSQL (i.e. PSQL extensions to the SQL parser and R-tree Index System) inside a host environment, offers an order of magnitude higher performance as it has been demonstrated in a prototype DBMS in which tuple identifiers are used in place of LOIDs.

3.0 The X-Interface

The X -Interface provides a front-end to the PSQL software and adds GIS capabilities. The user interface is depicted in Figure 2. The interface provides a text window for specifying PSQL queries, (bottom -most window in Figure 2), a message window, placed just above the text query window, a graphics window for 2-dimensional query input/output, placed above the message window, and a window dedicated for ORACLE output (displayed overlayed). The spatial input/output window has its own menu driven language for object/query drawing, searching, zooming, etc. The searching capability on the underlying space can be used to assist the formulation of the query by doing spatial searches without having to submit a query to ORACLE.

4.0 R-trees: An Efficient Spatial Index

R-trees are index devices for multi-dimensional space. Their main advantages are: a) they maintain themselves balanced for fast and uniform access cost, b) they dynamically adjust the space in a way that avoids “dead-space” (i.e. space containing no objects) and excessive “overlap” of objects, and c) they provide a natural and high level *object oriented search* as opposed to low level grid oriented search. The storage organization of R-trees is similar to B-trees and provides disk buffering and main memory page caching for arbitrary size files.

The decomposition used in R-trees is dynamic, driven by the spatial data objects. Therefore, if a region of an n-dimensional space includes dead-space, no entry in the R-tree is introduced. Leaf nodes of the R-tree contain entries of the form

$$(RECT, oid)$$

where *oid* is an object-identifier and is used as a a pointer to a data object and *RECT* is an n-dimensional minimal rectangle (called Minimal Bounding Rectangle or MBR) which bounds the corresponding object. For example, in a 2-dimensional space, an entry *RECT* will be of the form

$$(x_{low}, x_{high}, y_{low}, y_{high})$$

which represents the coordinates of the lower-left and upper-right corner of the rectangle. The possibly non-atomic spatial objects stored at the leaf level are considered atomic, as far as the search is concerned, and, in the same R-tree, they are not further decomposed into their spatial primitives, i.e. into quadrants, line segments, or pixels. Note that *oids* can be either logical

passed to ORACLE for execution. Updates to ORACLE tables which have spatial index(s) built on them are converted into sequences of ORACLE and R-tree operations on single tuples. The ORACLE operation is performed first, followed conditionally by the equivalent R-tree operation.

The R-tree Index System includes functions for creating an R-tree, loading an unpacked R-tree, packing an R-tree, searching an R-tree with one or more search-windows, inserting one or more entries into an R-tree, deleting one or more entries from an R-tree, and some additional functions to gather and report statistics of the various operations.

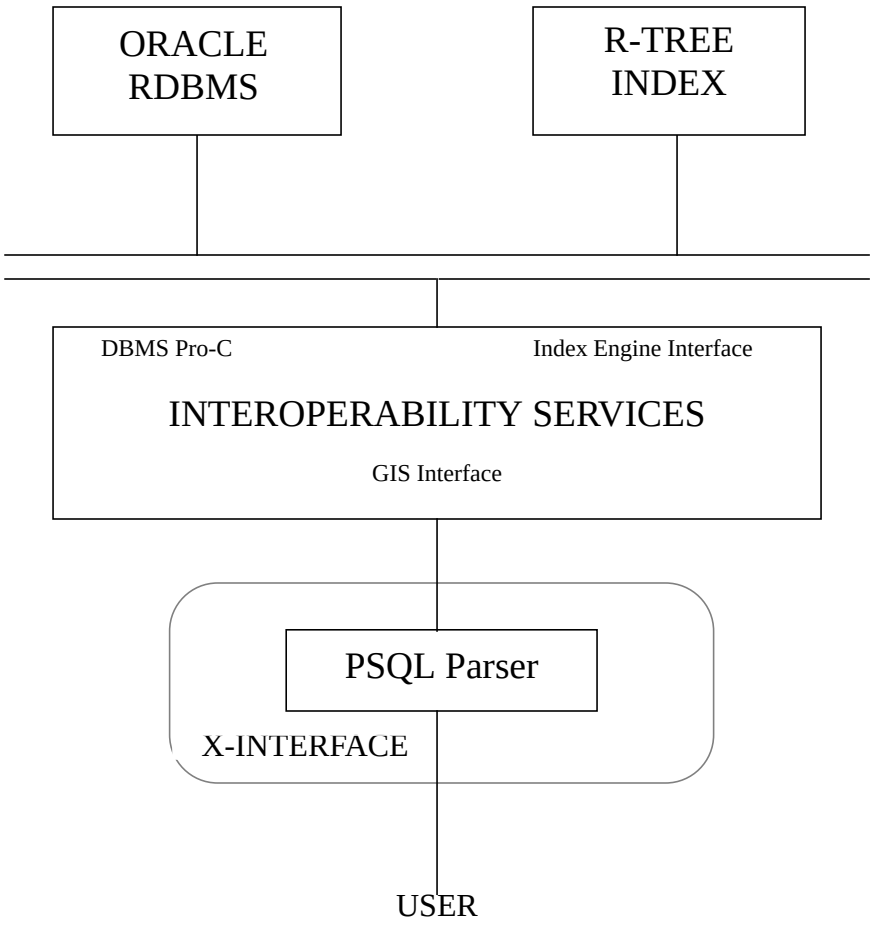

Figure 1 The PSQL Software Architecture

As was mentioned above, the interactions between the R-tree Index System and the host environment of ORACLE is achieved through Logical Object IDentifiers (LOIDs). Each spatial object is defined by its coordinates and a unique LOID. To create an R-tree, the *create rindex* function first obtains from ORACLE the LOIDs and their location (coordinates) in the space they occupy. The R-tree loading function then creates an R-tree which uses its own external and internal storage structures. To search for spatial objects within a search window, the window and the name of the R-tree is passed to the search function which returns the list of LOIDs of the objects located inside (or partially inside) the search-window. The returned LOIDs are imported to and used by ORACLE for retrieving any additional attributes of the qualified objects. This *direct connection*

sentation layer is needed so that spatial objects and their spatial relationships can be presented in their analog *object-oriented* form and not their internal RDBMS encoding.

This paper describes the implementation of PSQL [RoSeFa88] on ORACLE. PSQL is an extension of SQL which allows the creation and use of multi-dimensional spatial indexes, and direct specification of spatial queries that invoke spatial searches and spatial joins.

We have been using variations and extensions of R-trees for spatial indexes. *R-trees* were first introduced by Guttman in [Gutt84] and improved through better compaction techniques by Roussopoulos and Leifker in [RoLe85]. They are based on B-trees, but extended for multi-dimensional space. Their most frequent application is in 2-dimensional spaces where they provide better performance than ordinary B-trees. In fact, in some systems, B-tree indexes on 2-dimensional attributes confuse the query optimizer and make the search a lot slower than even a sequential scan of all the records. On the other hand, R-trees have been shown to be one of the fastest secondary index methods for 2-dimensional searches and very unique in maintaining performance characteristics in higher dimensions.

Although many spatial access methods exist, most of them are not very useful because they require that a) the data files get reorganized in the format that these structures can operate, and b) these methods cannot be incorporated with existing commercial systems which are closed or with in-house systems that are very hard to extend. Reorganizing existing data is totally impractical not solely because of the high conversion cost, but for other technical (consistency maintenance) and political reasons. Therefore, one of the strictest requirements is that the spatial data remain in the original storage and format structures.

The software architecture of PSQL is presented in section 2. The X-interface of PSQL is described in section 3. R-trees are described in section 4. The language of PSQL and examples in it are presented in section 5. Section 6 contains the conclusions.

2.0 The PSQL Software Architecture

The architecture of the PSQL Software is depicted in Figure 1. It consists of four major components all connected through the Interoperability Services Layer which provides the connectivity between the other components.

The X Interface provides a front-end to the PSQL software with modest GIS capabilities. The dotted line around it indicates that it is optional and that queries can be directed to the PSQL Parser either from an I/O channel, or from another GIS front-end able to converse PSQL.

The PSQL parser enables the user to perform queries using the full expressive capability of SQL and the spatial expression extensions of PSQL combined with the power of the ORACLE RDBMS. The parser intercepts all user input and diverts to ORACLE all standard ORACLE SQL statements which require no spatial search nor updates to tables which have spatial indexes. Statements requiring spatial search are preprocessed using the R-tree Index System. This preprocessing involves searching the R-trees, producing and importing to ORACLE the list of logical identifiers (LOIDs) satisfying the spatial search condition and, finally, modifying the initial query to an equivalent one based on the imported identifiers. The converted SQL statement is then

PSQL: AN INEXPENSIVE GIS SOLUTION FOR ORACLE

Nick Roussopoulos, Steve Kelley

Advanced Communication Technology Inc.
1209 Goth Lane
Silver Spring, MD 20905

ABSTRACT

This paper describes the ORACLE implementation of PSQL, a query language that allows spatial search and GIS capabilities on ORACLE databases. Spatial search and spatial joins are supported by variations of R-trees, which are excellent storage and retrieval devices for spatial objects and their relationships in multi-dimensional space. Other excellent features of the PSQL software package are that it requires no modification to the ORACLE databases and interoperates with ORACLE on a direct and interactive connection.

1.0 Introduction

Relational databases have mostly dealt with alphanumeric data types, (i.e. numerals and strings), and numeric or string comparison operators. Extensions to incorporate abstract types are emerging in the newer releases of commercial database systems. However, spatial operators and Geographic Information Systems (GIS) functionality are still not available.

There are several approaches for extending a Relational Data Base Management System (RDBMS) to include GIS functions. One is to have a GIS front-end as a presentation layer, and have a commercial RDBMS for storing and exporting the data to the GIS through a gateway. This is an expensive solution because the GIS requires the whole presentation data be imported to its own internal data structures before it can be used. Incremental access to an ORACLE database and efficient spatial search to select a portion of the database cannot be utilized by gateway software. The other approach is to expect the RDBMS vendors incorporate a GIS in future releases. This is an even less viable solution because of the diversity of GIS requirements and functionality.

A third, and middle of the road approach, is to build an intermediate layer of software which can interact with both the RDBMS and the front-end GIS package at a deeper than gateway level and provide both spatial search and efficient buffering on the underlying RDBMS.

Our approach is the middle of the road one. We believe that spatial relationships must be stored in an RDBMS and efficiently searched using multi-dimensional indexing in a manner similar to alphanumeric searches. The user should interact with both the RDBMS in its native language, SQL, and with the GIS in a similar and native language for manipulating spatial objects. The pre-