GEOGRAPHIC INFORMATION PROCESSING IN THE PROBE DATABASE SYSTEM*

Frank Manola, Jack Orenstein, Umeshwar Dayal
Computer Corporation of America
Four Cambridge Center
Cambridge, Massachusetts 02142

ABSTRACT

This paper describes the facilities of PROBE, an object-oriented DBMS being developed at CCA (Dayal 1985, Dayal and Smith 1986). and how these facilities apply to the requirements of geographic information processing

INTRODUCTION

The application of database technology to new applications. such as geographic information systems, CAD/CAM, software engineering, and office automation. is an extremely active area of database research. The characteristics of these applications impose a number of requirements on supporting database systems that are "unconventional" when compared to requirements associated with conventional commercial database applications. For example, the requirements of conventional applications can be adequately modeled by relatively simple and regular data structures, such as tables (relations), and a small, predefined set of operations on those structures On the other hand, GISs must deal with objects, such as features, that have complex and irregular structures Moreover, these objects will be created and operated on by a complex set of processes Such objects cannot be easily modeled by collections of simple attribute values, or by uniform database operations such as reads and writes Also, unlike commercially-oriented DBMSs which maintain only one up-to-date version of any record, a GIS may be required to maintain many different "versions" of information about the same object (e g , information at different scales, in both raster and vector forms, compiled at different times) Finally. commercially-oriented DBMSs are able to be fairly rigid in the class of data structures and processes on them that are supported, forcing users to adapt their requirements to these capabilities GISs have a greater requirement for adaptability, both to accomodate the diversity of specialized spatial data structures currently used in various applications, and to accommodate changes in technology and data requirements over the system life-cycle

In response to these requirements, a number of DBMS research organizations are pursuing development of "object-oriented" DBMSs (Dittrich and Dayal 1986, Lochovsky 1985, Manola and Orenstein 1986) These DBMSs allow meaningful application objects to be more-or-less directly modeled in the database Objects are accessed and manipulated only by invoking operations meaningful to the application, and specifically defined for the type of object involved Data structures and details of the implementation of the objects and operations can be hidden, or revealed only to specialized processes as needed Moreover, new object types, with their own specialized operations, may be freely defined by users for their own applications, rather than having to rely only on predefined, built-in data structures and operations PROBE is an object-oriented database system being developed by Computer Corporation of America This paper describes the basic features of the PROBE database system, and shows how they apply to GIS requirements The paper begins by describing the PROBE Data Model (PDM), and its spatial data capabilities This data

*This work was supported by the Defence Advanced Research Projects Agency and by the Space and Naval Warfare Systems Command under Contract No N00039-85-C-0263 The views and conclusions contained in this paper are those of the authors and do not necessarily represent the official policies of the Defense Advanced Research Projects Agency, the Space and Naval Warfare Systems Command, or the U S Government model is an object-oriented extension to a data model called *Daplex* previously developed and implemented at CCA (Shipman 1981). The utility of Daplex in spatial data modeling has been described in (Norris-Sherborn and Milne 1986). The paper then describes our approach to incorporating spatial data processing into database operations, and other aspects of the PROBE system. The paper concludes by describing the current status of the system.

PDM DATA OBJECTS

There are two basic types of data objects in PDM, *entities* and *functions* An *entity* is a data object that denotes some individual thing The basic characteristic of an entity that must be preserved in the model is its distinct identity. Entities with similar characteristics are grouped into collections called *entity types* For example, a GIS might have an entity type **FEATURE**, representing geographic features

Properties of entities, relationships between entities, and operations on entities are all uniformly represented in PDM by *functions* Thus, in order to access properties of an entity or other entities related to an entity, or to perform operations on an entity one must evaluate a function having the entity as an argument For example

- the single-argument function **POPULATION(CITY)** → **integer** allows access to the value of the population attribute of a **CITY** entity
- function **LOCATION(PT_FEATURE)** → **(LATITUDE,LONGITUDE)** allows access to the value of the location attribute of a point feature (note that a function can return a complex result)
- the multi-argument function **ALTITUDE(LATITUDE,LONGITUDE,MODEL)** → **HEIGHT** allows access to the altitude values contained in a digital terrain model
- function **COMPONENTS(FEATURE)** → **set of FEATURE** allows access to the component features of a group feature (such as a city).
- function **OVERLAY(LAYER,LAYER)** → **LAYER** provides access to an overlay operation defined for sets of polygons separated into different coverage layers

Functions may also be defined that have no input arguments, or that have only boolean (truth valued) results. For example.

- the zero-argument function **FEATURE()** → **set of ENTITY** is implicitly defined for entity type **FEATURE**, and returns all entities of that type (such a function is implicitly defined for each entity type in the database).
- the function **OVERLAPS(POLYGON,POLYGON)** → **boolean** defines a predicate that is true if two polygons geometrically overlap All predicates within PDM are defined as boolean-valued functions

In PDM, a function is generically defined as a relationship between collections of entities and scalar values. The types of an entity serve to define what functions may be applied with the entity as a parameter value There are two general classes of functions *intensionally-defined (ID) functions*, with output values computed by procedures, and *extensionally-defined (ED) functions*, with output values determined by conventional database search of a stored function *extent* (ID-functions may also involve the use of stored extents, in addition to computation.) References to all functions are treated syntactically as if they were references to ID-functions, even when a stored extent exists, rather than treating the various classes of functions differently However, particularly in the case of ED-functions, functions can often be evaluated "in reverse" i e , with "output" variables bound, to return "input" values (since both are available in a stored extent)

Entity types may be divided into *subtypes*, forming what are known as *generalization hierarchies* For example, one might define **POINT_FEATURE** as a subtype of **FEATURE**, and **RADIO_ANTENNA** as a subtype of **POINT_FEATURE**. As another example, the declarations

entity LAND_DIVISION
DESCRIPTION(LAND_DIVISION) → character
AREA(LAND_DIVISION) → POLYGON

entity OWNED_PARCEL isa LAND_DIVISION
OWNERSHIP(OWNED_PARCEL) → OWNER

define a **LAND_DIVISION** entity type having two functions, and a subtype **OWNED_PARCEL** having an additional function. Because **OWNED_PARCEL** is a subtype of **LAND_DIVISION**, any entity of type **OWNED_PARCEL** is also an entity of the **LAND_DIVISION** supertype, and automatically "inherits" the **DESCRIPTION** and **AREA** functions On the other hand it sometimes desirable that specialized versions of what appears to be the "same function" be available for different subtypes For example, one might wish to provide a general **SQ_MILES** function to compute the number of square miles in any 2-dimensional shape, but have different specialized implementations for various representations of those shapes

At the top of the generalization hierarchy, both entities and functions are members of the generic type **OBJECT**. In addition, the entity and function type definitions themselves are modeled as a collection of entities and functions, so that information in database definitions can be queried in the same way as database data.

Generic operations on objects (entities and functions), such as selection, function application, set operations, and formation of new derived function extents, have been defined in the form of an algebra (Manola and Dayal 1986) similar in some respects to the algebra defined for the relational data model Like the relational algebra, our PDM algebra provides a formal basis for the definition of general database operations. In particular, the algebra serves to define the semantics of expressions in our query language, *PDM Daplex*, involving functions and entities, such as:

```
for C in CITY, for M in MAP
  print(NAME(C)) where
    POPULATION(C) > 50000
  and SQ_MILES(AREA(C,M)) < 10
```

This object-oriented approach has a number of advantages for a GIS. First, geographic data can be dealt with at a level of abstraction appropriate for the processing involved, as suggested in (Claire and Guptill 1982) For example, some users may wish to operate only at the "feature" level, ignoring how a feature may be represented in terms of geometric entities such as polygons, or how these polygons may be represented in terms of lower level constructs, such as vectors nodes or chains or their physical encodings This is true even when the users wish to use selection predicates or invoke operations involving geometric properties of features, since the functions can conceal their access to geometric or lower level objects (a terrain model may, for example, be in either regular grid or TIN form without affecting the user's view of the **ALTITUDE** function). Second, the use of the functional syntax provides a smooth interface between stored data and computations in the model (for example, interpolation in a digital terrain model). These computations may include complex cartographic processes and knowledge-based techniques (which may be implemented by specialized hardware or software), even when these are not part of the DBMS per se, since the processes can be represented by functions in the model, and referenced within "database" requests Again the functional syntax allows the exact nature of required processing to be hidden when this is appropriate, and provides a uniform syntactic approach throughout the model

SPECIALIZATIONS FOR SPATIAL DATA

The general objects and operations provided by PDM do not obviate the need to define specialized spatial object types for specific applications, or the need to define the details of the implementation of spatial objects in terms of discrete representations in computer storage, and operations on them. They do, however, provide a framework within which these can be smoothly incorporated in an overall system architecture As suggested by the examples above, we model spatial properties of entities by functions (such as **AREA**) that map from the entities (such as a **LAND_DIVISION**) to entities of special types (such as **POLYGON**) that denote sets of points in space (such as lines, areas, or volumes), and embody various specific representations of such point sets. These special types are defined as subtypes of the generic spatial type **PTSET**, which represents a general set of points.

Only the most general point set semantics are defined for the **PTSET** type The detailed behavior and characteristics of spatial entities required for particular applications are defined in subtypes of **PTSET** defined to represent specialized classes of point sets (e g 2D and 3D point sets), and specific types of objects within those general classes (e g , 2D lines and curves, 3D solids) (Points and intervals can also be defined in the same way to represent temporal objects, thus allowing many different concepts of both space and time to be modeled) For each subtype, additional specialized functions are defined to represent the user-visible spatial or nonspatial properties, predicates, and operations appropriate to the type of object being represented. Moreover, "internal" functions, hidden from outside the object, are defined to represent aspects of the *implementation* of the object. Attributes, such as **ALTITUDE**, that vary in (2D) space can be represented by multiargument functions

For example, we might define subtype **POLYGON** as:

entity POLYGON
internal:
EXACT_POLYGON(POLYGON) $\rightarrow$ **set of EXACT_REP**
GF_POLYGON(POLYGON) $\rightarrow$ **set of ELEMENT**

Entities of type **POLYGON** store the actual representations of polygons. Subtypes such as **POLYGON** required for geographic applications would incorporate the mathematical constraints required to correctly represent a closed, bounded 2-dimensional shape, together with its topological relationships to other spatial entities The details of these representations are hidden from the general-purpose database routines within PROBE, but are used by specialized functions added specifically to manipulate polygons, and requiring access to the details of internal representations In this case, function **EXACT_POLYGON**, given a polygon, returns a set of entities defining the exact representation of the polygon, while function **GF_POLYGON** returns the PROBE geometry filter's representation of the polygon: a collection of elements resulting from the decomposition of the polygon (described in the next section)

Facilities for adding such specialized entity subtypes and functions are provided within PROBE These subypes would either inherit the definitions of operations from the **PTSET** type, or would provide specialized versions of such operations For example, if the intersection of two entities of the type **POLYGON** were expected to produce only common subareas (i.e , a result of the same type), a specialized version of the intersection operation would be required, since an ordinary point set intersection might also return common boundary segments (or points), or even shared components of the underlying representations. (Such dimensionally-constrained set operations are known as "regularized set operations" in the 3D solid modeling literature (Requicha 1980)).

In addition to dealing with **PTSET** entities as individual objects, there are many situations in which it is necessary to deal with PTSETs contained within other PTSETs For example, a map would be defined as containing a collection of component objects (roads land units, etc) The **POLYGON** defining the spatial representation of the map would have to contain all the **PTSET** entities (polygons and other representations) of those components (together with the complex topological relationships among them) Moreover, a component representing a complex feature may itself contain further subcomponents, and so on Thus, both the map and its component objects, and their associated spatial representations, naturally exhibit a hierarchical structure. While it is not possible to deal with this subject further in this paper, PROBE includes facilities both for modeling these containment relationships, and for efficiently processing queries involving them (Orenstein and Manola 1986, Rosenthal 1986).

By using different functions (or a single set-valued function), a database entity can be related to any number of different spatial representations For example, a bridge might be represented as a point in one map, as a line in a map showing greater detail, as a space frame in its design data, etc Also, as suggested above, the 2D point set representing the bridge in a particular map can be associated with the point set representing the area covered by the entire map, enabling the bridge to be associated (and located) with respect to the other features in the same map

QUERY PROCESSING

For a DBMS to be practically usable in a given application, it must provide adequate performance A key component of the DBMS in determining its performance is its query processor, since this is where efficient strategies for performing database operations are determined. A considerable literature exists on query optimization strategies for conventional database systems. However, because a conventional database system provides only a fixed set of data types and operations, support for those types and operations can be coded directly into the physical data structures and algorithms of the DBMS query processor This simplies the optimization problem considerably.

An extensible database system, such as PROBE, is more difficult to implement because it is no longer possible to build into the query processor's implementation the definition of each data type and operation that may be encountered. Instead, the extensibility of the system implies that some important details of the system will be provided in user-defined object types For example, for the query.

```
for S in STATE where NAME(S) = "Florida"
  for C in CITY where AREA(C) is in AREA(S)
    print(NAME(C));
```

the **AREA** functions might return values of a user-defined specialized data type, and the processing of the **is in** predicate might be implemented by a user-supplied procedure whose implementation details are hidden from the query processor

A reasonable division of labor would be for the query processor to handle sets of objects of the most generic types (e g , **ENTITY** and **PTSET**) and for the more specialized object types to provide for detailed manipulation of individual objects This allows DBMS implementors to be concerned primarily with generic database issues, while application specialists can concern themselves primarily with application-specific issues. For example, an application specialist could define a **POLYGON** object class whose operations work on individual polygons (e g. do two polygons overlap? does the polygon contain a point? what is the area of a polygon?) The DBMS can then implement operations that handle arbitrary numbers of polygons (e g find all the polygons overlapping a given polygon) in terms of these operations

Given this division of labor, how can the data model, especially its spatial components, be supported efficiently? Our approach to the problem has two components. First, we present an architecture compatible with the division of labor discussed above. Second, we describe how the efficient processing of spatial data can be accomplished under this architecture. This step is necessary to show that PROBE's approach to spatial data is feasible as well as general

Architecture for the Query Processor

Many queries (including spatial queries) can be expressed in terms of iteration over one or more collections of objects, and the application of one or more functions to each object or to a group of objects within the iteration For example, in order to find all pairs of objects in a set, S, of spatial objects, within a given distance, d, of one another, the following algorithm can be used:

```
for each x in S, for each y in S
 if distance(x,y) < d then output (x, y)
```

(It is a simple matter to eliminate symmetric results (x overlaps y iff y overlaps x) and reflexive results (x always overlaps x).) This kind of algorithm is very compatible with the division of labor discussed above The database system can handle the iterations, passing pairs of objects to a distance function (part of a spatial object class) whose boolean result indicates whether the pair should be part of the output This architecture is shown in figure 1

The problem with this approach is one of performance. Each loop over a set of spatial objects corresponds to an actual scan of the set Nesting these loops leads to polynomial time algorithms (whose degree is equal to the level of nesting). This will not be acceptable in practice since much more efficient, special-purpose algorithms often exist However, it is not possible to build in a collection of special-purpose algorithms and retain generality It is therefore necessary to consider another architecture

The PROBE approach is to provide a generally useful *geometry filter* that helps optimize such nested loops The output from the filter will be a set of *candidate* objects (or a set of groups of objects) that satisfy the query Any object or group that is not included in the candidate set is certain not to satisfy the query. An object or group in the candidate set *is likely to* satisfy the query. The set of candidates will then be refined to yield the precise answer by applying user-supplied predicates (such as distance). This architecture is shown in figure 2.

Note that this architecture is also compatible with the division of labor described above. The user only has to supply, as part of a spatial object class, a predicate that tests a group of objects. The geometry filter is part of the database system and relies on another object class (**ELEMENT** in figure 2) that is provided as part of PROBE The ideas behind the geometry filter and the **ELEMENT** object class are described below.

How the Geometry Filter Works

The geometry filter is based on a grid representation of spatial data. For a spatial object, s, a grid cell that contains any part of s is "black" while a grid cell that is completely outside s is "white" The collection of black cells forms a ***conservative approximation*** of s In order for the geometry filter to retain its filtering property (i e. not discard positive results), it is important for the approximation to be conservative - i e contain the exact representation This is because positive results are indicated by the overlap of objects A non-conservative approximation does not necessarily contain the exact representation and some overlap relationships involving the exact representation would not be detected by the approximation (Unless stated otherwise, all results in this section are from (Orenstein 1986, Orenstein and Manola 1986).)

Many spatial operations can be carried out in a single scan of a grid, replacing the nested loops algorithms described above The problem with this approach is that grids can be very large (at high resolution), and that, as a result, a scan of the grid will be very slow However, it is usually the case that spatial data contains much regularity There will be large black regions and large white regions The geometry filter exploits this regularity by using a compact encoding of these regions of uniformity. Each spatial object is represented by a collection of rectangular regions called "elements" Each element can, in turn, be represented by a range of integers. Typically, a 32-bit word is sufficient to represent an element

Elements are obtained by partitioning the grid in a highly constrained way As a result, elements have some simple and useful mathematical properties

- The size, shape and position of an element can be described very concisely by a "z value" - a short string of bits (Orenstein 1984) The same encoding has been discovered independently by several other researchers (Abel and Smith 1983, Gargantini 1982, Mark and Lauzon 1985)

- Any two elements either contain one another or precede one another (when ordered by z value). Overlap (except for containment) cannot occur.
- In a sequence of elements sorted by z value, proximity in the sequence is highly correlated with proximity in space.

These properties lead to simple and efficient algorithms for a wide variety of spatial problems The scan of the grid cells is replaced by a scan or merge of z-ordered sequences of elements. Algorithms of this kind are possible because of the absence of overlap relationships

The performance of these algorithms looks promising for two reasons. first, the property that proximity in z-value corresponds to proximity in space leads to good clustering Database implementers go to great lengths to ensure that records to be retrieved together are stored near each other, ideally on the same page or cylinder (e g . see (Lorie 1977)) In spatial applications, objects that are near each other are often retrieved together. and z order maps this proximity to proximity in real storage devices Second. with some simple reasoning about z values, it is possible to "skip" parts of a space that could not contribute to the result This reasoning has been incorporated into the geometry filter algorithms and there is analytical and experimental evidence that the savings are substantial The performance for range queries matches that of the best practical data structures (e g. the kd tree).

The representation used by the geometry filter, the collection of elements, can be seen as an abstraction of the quadtree and all its variants There is an exact correspondence between a leaf of a quadtree and an element of the geometry filter (The quadtree can be seen as a trie of order 4, keyed by elements Similarly, the octtree can be seen as a trie of order 8, keyed by elements) However, instead of requiring the use of a particular data structure, geometry filter algorithms can use *any* data structure or file organization that supports random and sequential accessing This is a very important consideration since it permits the use of efficient and widely available structures such as sorted arrays, binary trees (and variations), B-trees (and variations), ISAM, etc

The geometry filter supports a wide variety of spatial operations, including many operations for which the quadtree and related structures have been proposed Given two sets of spatial objects, R and S, **spatial-join**(R, S) locates pairs of objects (r, s) such that r belongs to R, s belongs to S, and r and s overlap spatially This operation can be used to evaluate range queries, partial match queries (important in database systems), containment queries and proximity queries. It has also been applied to interference detection and to polygon overlay.

EXAMPLE

(Zobrist and Nagy 1981) give several examples of multistep, geographic information processing tasks that demonstrate the need for manipulation, integration, and conversion of geographic data stored in different representations (without going into how the various steps might be implemented in a database system) To demonstrate the use of PROBE for geographic information processing, we now "translate" some of the processing from one of the examples into steps that could be carried out in PROBE

The example selected is a study of the California Desert Conservation Area. We will concentrate on the latter part of the example, after multiple sources of data (such as LANDSAT frames and digital terrain data) have been integrated to yield, for each point of the study area, a classification (Other steps in the example could be modeled in PROBE as well for example. geometric transformations stored as PROBE ID-functions can be applied to LANDSAT images stored as PROBE point set entities similarly vector/raster conversion routines stored as PROBE functions can be applied to boundary files stored as PROBE objects)

To model the data and the required computations, we use the entity types **LAND_DIVISION**. **OWNED_PARCEL**, and **POLYGON** defined earlier, and additional types and functions such as

entity IMAGE
PIXELS(IMAGE) → set of PIXEL
COVERAGE(IMAGE) → POLYGON
CLASS(IMAGE,PIXEL) → CLASS_VALUE
QUALITY(IMAGE) → RATING

RASTERIZE(POLYGON) → set of PIXEL
PIXEL_TO_ACRE(integer) → ACRE

An entity of type **LAND_DIVISION** represents the area of study In this case, its **NAME** function would return "California Desert Conservation Area" An entity of type **OWNED_PARCEL** identifies a parcel of land whose owner can be identified. Since both are entities of supertype **LAND_DIVISION**, they have an **AREA** function that returns a polygon (assumed to have absolute coordinates) describing the land area **IMAGE** defines an image object, with a **CLASS** function that gives a classification value (vegetation, urban, etc) for each pixel for each image and a **COVERAGE** function defining its area of coverage. By specifying constraints on the values of image attributes such as **COVERAGE** and **QUALITY** a set of images can be identified **RASTERIZE** is an ID-function that, given a polygon, returns the set of pixels that are completely or mostly within the polygon (Pixels on the boundary are included in the result only if they are mostly within the polygon) Function **PIXELS_TO_ACRES** uses the unit of area covered by a pixel to convert from a pixel count to acres

The goal of the last part of the example is to overlay the land classification information with "boundary files" (giving ownership information in this case) to obtain acreages of land classes per region. This processing can be done in PROBE in five steps.

Select an image: There are several criteria that might influence the selection of an image The most important is the area covered by the image After selecting images that cover the area of the study, further selection can be based on other attributes To locate images in the right area, the following steps can be used

1 Do a spatial join between the **COVERAGE** function of the set of images, and the **AREA** function of the **LAND-DIVISION** entity representing the study area, to locate the images overlapping the study area
2. Select one image based on other attributes and create a new ED-function storing the needed information, **STUDY_IMAGE(PIXEL) → CLASS_VALUE**.

Get the relevant ownership information: Obtain the owned-parcel polygons, and do a spatial join with the study area's polygon to find the regions covering the study area.

Convert to common representation: Convert the owned-parcel polygons to raster format using the **RASTERIZE** function and produce a new ED-function
PIXEL_OWNER(OWNER) → set of **PIXEL**

Do the overlay: Each pixel is related to a region through the **PIXEL_OWNER** function, and has a class, as indicated by the **STUDY_IMAGE** function. These functions can be composed, yielding **OVERLAY(PIXEL) → (OWNER,CLASS_VALUE)**

Compute acreage of class per district: PROBE provides general-purpose aggregation functions, such as sums, maxima, minima, and counts In this example, aggregation can be used to count the number of pixels of each class in each region **USAGE(OWNER.CLASS) → count** Finally, the **PIXEL_TO_ACRES** function can be applied to **USAGE** to convert the count of pixels to area, measured in acres

CURRENT WORK

Work related to PROBE is ongoing in a number of areas, some of them mentioned in previous sections A breadboard implementation of PDM and its algebra, and of some query processing algorithms, is under way. The breadboard will be tested against a number of example applications (one of them a geographic application). This involves the definition and implementation of a number of specific entity types incorporating spatial semantics (Manola and Orenstein 1986).

ACKNOWLEDGEMENTS

We express our thanks to Alex Buchmann, Upen Chakravarthy, Mark Dewitt, David Goldhirsch, Sandra Heiler, and Arnie Rosenthal for their contributions to the development of PROBE.

REFERENCES

Abel, D J and Smith, J L 1983, "A data structure and algorithm based on a linear key for a rectangle retrieval problem", *Computer Vision, Graphics and Image Processing* 27(1)

Claire, R W and Guptill, S.C 1982, "Spatial Operators for Selected Data Structures", *Proc. Fifth Intl Symp. on Computer-Assisted Cartography*, ACSM

Dayal, U. et al. 1985, "PROBE - A Research Project in Knowledge-Oriented Database Systems: Preliminary Analysis", Technical Report CCA-85-03, Computer Corporation of America.

Dayal, U. and Smith, J.M 1986, "PROBE A Knowledge-Oriented Database Management System", in M L. Brodie and J. Mylopoulos (eds), *On Knowledge Base Management Systems Integrating Artificial Intelligence and Database Technologies*, New York, Springer-Verlag.

Dittrich, K and Dayal, U 1986 (eds), *Proc Intl Workshop on Object-Oriented Database Systems*, Washington, IEEE Computer Society Press

Gargantini, I 1982, "An effective way to represent quadtrees", *Comm ACM*, 25(12)

Lochovsky, F. 1985 (ed), *Database Engineering*, Vol. 8, No 4, Special Issue on Object-Oriented Systems, IEEE.

Lorie, R A. 1977, "Physical Integrity in a large segmented database", *ACM Trans on Database Systems*, 2(1), 91-104

Manola, F A and Orenstein, J 1986, "Toward a General Spatial Data Model for an Object-Oriented DBMS", *Proc 12th Intl Conf Very Large Data Bases*, IEEE

Manola, F.A. and Dayal, U 1986, "PDM An Object-Oriented Data Model", in (Dittrich and Dayal 1986)

Mark, D M and Lauzon, J P. 1985, "Approaches for quadtree-based geographic information systems at continental or global scales", *Proc Seventh Intl Symp. on Computer-Assisted Cartography*, ACSM.

Morehouse, S 1985, "ARC/INFO A Geo-Relational Model for Spatial Information", *Proc Seventh Intl Symp on Computer-Assisted Cartography*, ACSM

Norris-Sherborn, A and Milne W J 1986. "A Practical Approach to Data Modelling in Spatial Applications", *Software—Practice and Experience*, Vol 16(10), 893-913, (October 1986)

Orenstein, J.A. 1984, "A Class of Data Structures for Associative Searching", *Proc. ACM SIGACT/SIGMOD Symp. on Principles of Database Systems*, New York, ACM.

Orenstein, Jack 1986, "Spatial Query Processing in an Object-Oriented Database System", *Proc 1986 ACM-SIGMOD Intl Conf on Management of Data*, New York, ACM

Orenstein, J and Manola, F. 1986, "Spatial Data Modeling and Query Processing in PROBE", Technical Report CCA-86-05, Computer Corporation of America

Requicha, A. 1980, "Representations for Rigid Solids Theory, Methods, and Systems", *Computing Surveys*, 12(2), 437-464 (December 1980)

Rosenthal, A. et al 1986, "A DBMS Approach to Recursion", *Proc 1986 ACM-SIGMOD Intl Conf on Management of Data* New York, ACM

Shipman, David 1981, "The Functional Data Model and the Data Language DAPLEX" *ACM Trans Database Systems* 6(1), 140-173

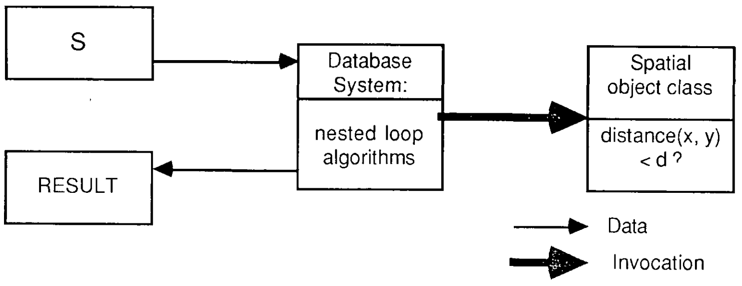

Figure 1.

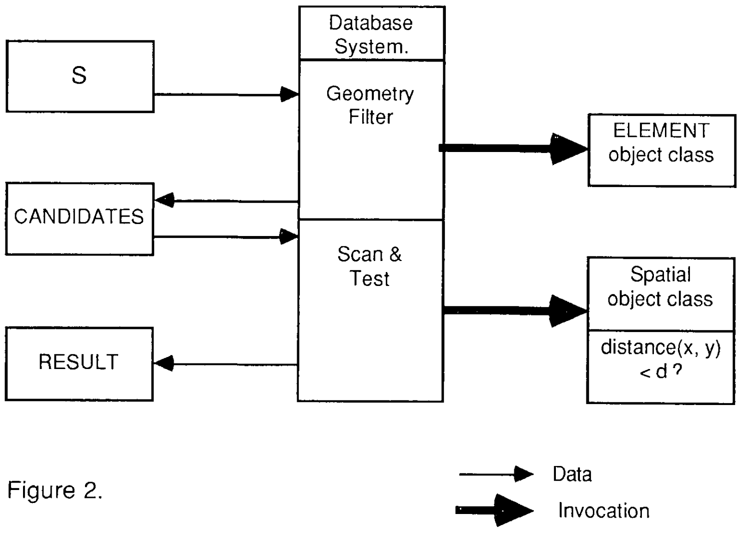

Figure 2.