Symbolic and Spatial Database for Structural Biology

Dan Benson, Greg Zick
Department of Electrical Engineering, FT-10
University of Washington / Seattle, WA 98195
benson@ee.washington.edu
zick@ee.washington.edu

ABSTRACT

This paper describes the development of a database to support three-dimensional image reconstruction of structural biology using object-oriented technology. The requirements of this system encompass many of the popular justifications for the application of object-oriented technology, such as non-standard data types and complex composite data, but we also find advantage in the increased functionality obtained for spatial relationship operations and access methods. We focus attention on the implementation of the spatial data, its representation, operations, indexing, and queries.

1 INTRODUCTION

Biomedical imaging is making a tremendous impact on medical knowledge, teaching, and practice due to the fact that images provide a great deal of information that is otherwise unobtainable. Improvements on the acquisition, dissemination, understanding, and use of this information comprise the bulk of imaging research activities. A brief summary of them includes the development of new image modalities, image acquisition techniques, image processing, feature extraction, object recognition, and applications which make use of the extracted information. As these technologies advance, and even become automated, the crucial missing component is the organization and management of the underlying data that is generated [DUER83]. Shown graphically in Figure 1, data generated by these biomedical imaging activities can be organized in a database that supports multiple data types and multi-level models.

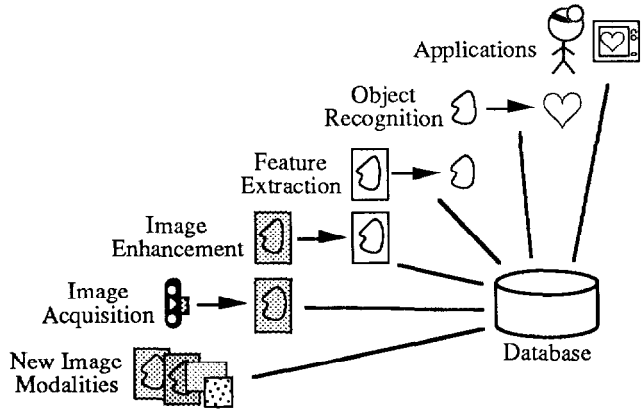

Figure 1. The role of a Biomedical Imaging Database

Images contain information about real world objects. Once acquired, each stage in the process carries this information along while "squeezing out" certain elements of the image contents forming an internal model or representation that can be used by applications. These stages are all related, in fact, they build off of each other.

OOPSLA'91, pp. 329-339

Likewise, there exists relationships among the data each stage generates.

There are two basic categories of data a biomedical imaging database must support, symbolic and spatial. Symbolic data is the sort of alphanumeric data commonly found in traditional databases, such as patient accounting information, or employee records. Spatial data, on the other hand, consists of geometric information, such as maps, images and their contents, or three-dimensional anatomical objects.

A traditional relational database stores information describing real world entities but is limited in the ways these entities can be represented and accessed. There exists a semantic gap between the user's representation of the world and the representation in the database. This gap is substantially narrowed through an object-oriented implementation that provides multiple layers of abstractions closely modeling the user's world view.

In this paper, we describe the development of a database to support three-dimensional image reconstruction of structural biology using object-oriented technology. This application involves all of the stages of biomedical imaging mentioned above. The requirements of this system encompass many of the popular justifications for the application of object-oriented technology, such as non-standard data types and complex composite data, but we also find advantage in the increased functionality obtained for spatial relationship operations and access methods. We place particular attention to the implementation of the spatial data, its representation, operations, indexing, and queries.

The current prototype consists of the GemStone [GEMS90] object-oriented database running as a server on an IBM RS/6000 and Objectworks for Smalltalk-80 v. 2.5 [PARC90] running on a Macintosh IIfx as the application interface. We based our selection of these tools on their prototyping capabilities, data impedance matching, and availability at the time the project was initiated.

The organization of this paper is as follows: Section 2 provides the background and context of the application domain. Section 3 describes the spatial data representation and operations of spatial relationship. Section 4 presents an object-oriented spatial index that augments the vendor-supplied access to object sets. Section 5 describes spatial queries and how the application of object-oriented technology improves accuracy and precision in spatial search. Section 6 concludes with a summary and discussion of future work.

2 BACKGROUND

For a number of years, researchers in the department of Biological Structures at the University of Washington have been developing and refining methods for 3-D image reconstruction [STIM88, PROT89, MCLE91] in which three-dimensional images of anatomical objects are reconstructed from sets of ordered 2-D cross-sectional slices, somewhat like a loaf of sliced bread. The images and animations produced by these techniques reveal anatomical structures in ways never seen before and allow interactive manipulation of accurate quantified data in anatomy.

The data acquisition process leading up to the 3-D reconstruction begins with various specimens prepared in ways that allow millimeter-thin slices to be removed as images are acquired of each new surface. The objects of interest are then traced manually by professional anatomists for each image taken, similar to contour lines on a map. These surface boundaries are labeled and digitized into computer-readable form for input to a 3-D graphics editor where the data is edited and displayed as three-dimensional surface reconstructions.

Currently underway is a project to acquire data encompassing the entire human body, providing the foundation for a distributed knowledge base of structural biology that will be used to solve problems in basic science, teaching, and clinical medicine [BRIN89]. Because anatomy is a fundamental framework upon which most of the basic medical sciences rest, a knowledge base of biological structure would have profound implications in many areas of medicine. A key aspect to the success of this system is the underlying management of the exceptionally large amount of data, the complexity and structure of the data, and its relationships. The organization of this data must also provide efficient symbolic as well as spatial access to the anatomical objects. Symbolic access is retrieval based on attribute values such as, "*Select all images of the liver taken after October 6, 1991,*" whereas spatial access is based on spatial properties such as, "*Select all objects within 10 mm of the heart.*"

Up to now, this data has been stored in flat files and organized in a file directory hierarchy. While this has been adequate for relatively small sets of data, knowledge of the relevancy and structure of objects, as well as relationships among various objects, exists in the minds and memories of the biologists rather than as an integrated part of the data itself and information retrieval does not go beyond simple filename lookup. This method of organization fails as the amount of data increases, the relationships become more complex, and access to the data more sophisticated.

Conventional relational database technology does not meet the needs of modern imaging applications, characterized by highly complex and structured data, multiple data types and relationships, and non-traditional database processing. It lacks adequate data models and poses a rigid table structure for the definition of the relationships between data records, thus preventing efficient representation and access of spatial or complex data structures.

We believe that an object-oriented database approach offers many advantages in supporting biomedical applications having spatial data. It shortens the semantic gap between real-world objects and their corresponding abstractions and thus offers a more flexible model for dealing with complex data. The data modeling capabilities provided by the object-oriented model make it possible to support not only multiple types of images but also highly structured data such as graphics. Furthermore, the extensibility of an object-oriented database allows us to implement user-defined index structures which are essential in achieving adequate performance in spatial data access.

3 SPATIAL DATA

Spatial data can be defined as anything having a location in a given global space with zero size (point) or non-zero size (occupies space). A spatial database, then, supports data structures for the representation of spatial data, efficient spatial access capabilities, and may also support a subset of geometric operators on the data [GÜNT88]. Spatial data is found in many application areas including anatomy, solid modeling, geography, computer aided design (CAD), robotics, and others.

Anatomical objects make up the primary data set for the structural biology database. Each three-dimensional object is described spatially in terms of its boundary. This information is obtained from a series of images taken of 2-D cross-sectional slices in which objects of interest are identified explicitly as ordered sets of points that trace their contours. Each set of ordered 2-D contours describe an object in three dimensions. The relationship between the original images, the 2-D contours, and the 3-D surface reconstruction is depicted in Figure 2.

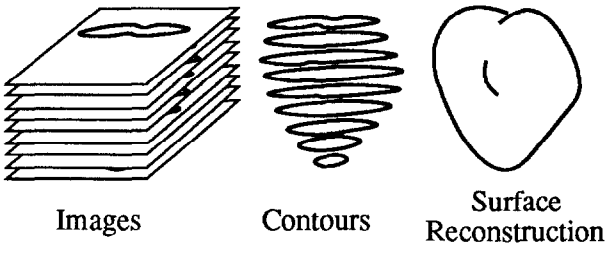

Figure 2. Spatial data abstractions

In an object-oriented implementation, the internal representation is hidden and can be modified with minimal effect on the overall system. No matter which representation is used internally, the object interface reflects generalizations about all spatial objects. For instance, each object is located at a point in space. All spatial objects, therefore, respond to the message requesting their location. Another generalization is that all spatial objects occupy a portion of space (a point having zero size). Because the description of the regions occupied by objects can be arbitrarily complex, a common approximation of an object's extent is a bounding box, defined by an *n*-dimensional rectangle describing intervals in *n* dimensions that completely enclose the bounds of the object.

We represent spatial data types through a class hierarchy defining 3-D objects, shown in Figure 3.

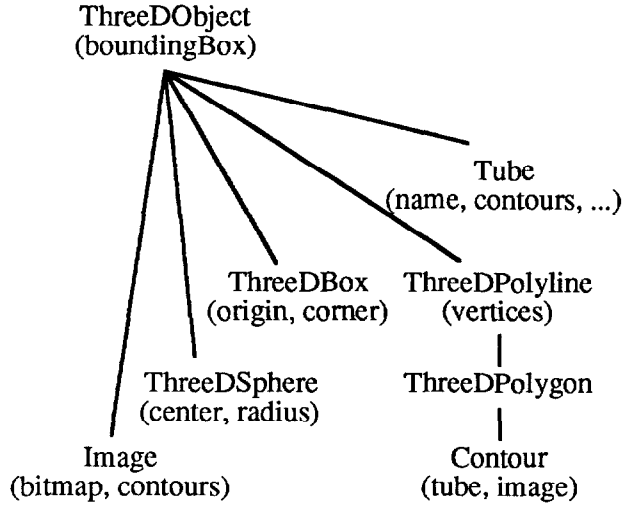

Figure 3. ThreeD class hierarchy.

The generalizations of all 3-D objects are captured in the common superclass called ThreeDObject. Some of the basic attributes ThreeDObjects can be asked for include:

boundary	returned in various forms such as points, lines, contours, etc.
bounding box	defined as the minimum ThreeDBox containing all of the object's extent
center	defined as the center point of the object's bounding box
location	defined as the closest point of the object's bounding box to the origin
surface area	area of object's surface
volume	volume of object's bounds

Each subclass of ThreeDObject may override these basic methods and may have additional specialized attributes. ThreeDPolygons, for instance, are confined to a plane and can therefore be asked for their area and perimeter; a ThreeDPolyline can answer its length and a ThreeDSphere can provide its radius.

The basic 3-D anatomical object is called a Tube. A Tube has instance variables describing symbolic information and one instance variable describing spatial data called *contours* that is an ordered collection of instances of the class Contour. As a subclass of ThreeDPolygon, each Contour is defined by an ordered collection of ThreeDPoints and is associated with its Tube and Image that contains the original bitmap data. Besides the general-purpose polyhedra objects, other specialized 3-D object classes are defined that are useful for spatial queries: ThreeDBox, ThreeDPolyline, and ThreeDSphere.

The various attributes that can be obtained from 3-D objects are used in determining spatial relationships between objects. The two basic spatial relationships are intersection and containment between two objects. For example, an object can determine whether it intersects or contains another object through the messages, *intersects: aSpatialObject* and *contains: aSpatialObject,*

respectively. The argument to these messages, *aSpatialObject*, can be any spatial object.

Each class of spatial object may rely on a specific set of tests when determining its relationship with other spatial objects. A brute-force implementation of the *intersects:* method would be to first determine the type of object passed and then invoke the appropriate algorithm for intersection test based on the argument type. For example, the *intersects:* method for the Tube class would look like:

```
intersects: aSpatialObject
  (aSpatialObject isKindOf: Tube)
    ifTrue: [ ... code to test for intersection
              with another Tube ...].
  (aSpatialObject isKindOf: ThreeDPolygon)
    ifTrue: [ ... code to test for intersection
              with a ThreeDPolygon ...].
  (aSpatialObject isKindOf: ThreeDSphere)
    ifTrue: [ ... code to test for intersection
              with a ThreeDSphere ...].
  ...
```

This, however, results in a lengthy case-like statement that is both inefficient and difficult to maintain for each type of spatial object in the system.

For spatial relationship operations, we prefer to implement a double-dispatching technique, similar to that used in the Smalltalk-80 kernel classes for handling arithmetic operations among Number subclasses [GOLD83]. In double-dispatching, the receiver object returns the result of sending the argument object a more specific message with itself as the argument. It is a useful technique for efficiently choosing an algorithm based on the class of the argument of a message and the class of the receiver. Using the previous example, the Tube *intersects:* method becomes:

```
intersects: aSpatialObject
  ^aSpatialObject intersectsTube: self
```

A complete implementation requires that all spatial object classes implement an *intersectsTube:* method containing the appropriate code to test specifically for intersection with a Tube object. The same would apply for other types of spatial objects. Double-dispatching provides significant speed at the expense of a large number of typed methods and makes it possible to send the generic *intersects:* message to all types of spatial objects. If the number of classes participating in double-dispatching becomes too large, other techniques can be incorporated, such as coercion [PARC90].

The intersection relationship is commutative so double-dispatching simply reverses the arguments. However, the containment relationship must be rephrased to an equivalent relationship when double-dispatching. For instance, the *contains:* method would return the result of sending *containedIn[selfClassName]: self* to the argument object.

4 SPATIAL ACCESS

An important aspect of a database containing spatial information is providing efficient spatial access to objects. For large data sets, indices can aid the search process in order to obtain adequate performance. Retrieval of spatial objects in the database is based on symbolic and/or spatial properties. Current database systems support conventional indices, such as B-tree, ISAM, and hashing on simple data types, but do not provide spatial data indexing [ULLM88]. Spatial access, then, is limited to linear iterative search across the entire collection of objects.

An object-oriented database, being extensible, allows us to construct a user-defined indexing structure using high-level objects. An ideal solution would implement the index at a low level inside the database kernel as close to the disk activity as possible. Although implemented at a higher level than conventional indices, an object-level index does offer several advantages. The

database administrator has direct control and design of the index and it can be tuned for specific applications. If desired, the index can be easily replaced if an improved index is found. Care must be taken, however, to see that the index maintains consistency and operates as a built-in index would.

A number of index structures for organizing spatial data have been proposed. The most common consist of variations of hierarchical and bucket methods such as quad-trees, oct-trees, k-d trees, k-d-B trees, grid files, RTrees, and cell trees [GÜNT88]. Most methods are designed primarily for point data. Of those that support objects of non-zero size, we chose the RTree as the most suitable structure for the anatomical data because it readily supports extended objects, such as lines, regions, and volumes, it does not sub-divide objects, and does not restrict occupancy to fixed-grid cells.

We have designed and implemented an object-oriented R*Tree [BECK90] spatial index. The R*Tree, an enhanced variant of the original RTree [GUTT84], is in the family of spatial access methods that are based on the approximation of complex spatial objects by their bounding boxes. This approximation makes the R*Tree efficient in terms of both space and time because the information stored at each node in the tree consumes a limited number of bytes and simple rectangular regions can be compared quickly.

The R*Tree organizes spatial objects in a height-balanced tree structure by essentially grouping objects into neighborhoods. Each tree has one root node; each node can have a maximum of *M* children. Leaf nodes contain references to the actual spatial objects and intermediate nodes contain references to children nodes and a parent node. Each node also stores its own bounding box representing the total region covered by its children.

Based on a high-level tree-structure, the R*Tree is inherently object-oriented. It lends itself well to data and behavior encapsulation and is designed to intermix spatial objects of multiple dimensions. The basic behavior of an R*Tree object is specified through the actions of insertion, deletion, and searching.

Database objects are stored in the database in container objects, such as Sets or Bags, similar to the function of a relation storing records in a relational database. For spatial objects the container class, IndexedSpatialSet, is defined that has two instance variables, *objectSet*, an instance of Set that acts as the holder for all spatial objects inserted into the container, and *spatialIndex*, an instance of R*Tree that provides spatial access to the set of objects.

When an object is inserted into an IndexedSpatialSet it is inserted into the *objectSet* and the *spatialIndex*, where the *objectSet* organizes objects based on a hashing method and the *spatialIndex* organizes them based on their spatial properties. Deletion of objects occurs in a similar fashion. An IndexedSpatialSet, with its instance variables, is depicted graphically in Figure 4.

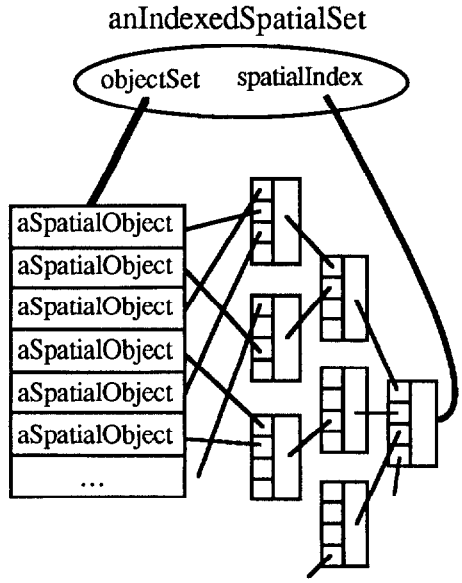

Figure 4. An IndexedSpatialSet

During insertion, the R*Tree object needs to know the bounding box of the object to be inserted, obtained through the *boundingBox* message. The R*Tree object need not be concerned with the type of spatial object it is inserting as long as the object responds appropriately to the *boundingBox* message. In fact, an *n*-dimensional R*Tree is able to accept a spatial object of *n* dimensions or lower since all comparisons between bounding boxes are done by the bounding box objects themselves which can handle differences in dimensionality.

5 SPATIAL QUERIES

Spatial queries on objects are formulated in terms of spatial properties such as location and regions of occupancy. The two most common queries are:

- *Object intersection query*: Given a spatial object S, find all objects, O, where $O \cap S \neq \varnothing$
- *Object containment query*: Two variations: Given a spatial object S, find all objects, O, where (1) $S \supseteq O$, and (2) $O \supseteq S$

Two problems encountered with spatial search, accuracy of results and precision of query region, are related to the index structure. Conventional implementations of non-point spatial indices cannot provide completely correct answers to spatial queries [OREN90]. Furthermore, each index structure imposes a restriction on the type of query region that can be specified. Most methods, for example, allow for only a rectangular region parallel to the global object space coordinate axes. While this may be adequate for some applications, there is often a need to specify other types and more precise search regions. For instance, given a set of three-dimensional anatomical objects, a possible query might be, "*Find all objects inside the skull.*" Restricting the search region to a rectangular cube makes it impossible to describe precisely the volume inside the skull.

We have shown that an object-oriented spatial index overcomes the problems of inaccuracy and precision of query region [BENS91]. Spatial index search operations are inaccurate because they are each based on an approximation of the data objects (e.g., bounding box) so the accuracy of the search is only as good as the approximation. The object-oriented R*Tree returns completely accurate answers because each candidate object, identified at the leaf node level, is interrogated as to whether or not it actually does fulfill the search criteria rather than only its approximation. Secondly, through polymorphism and late binding, any arbitrary spatial object may be specified as the search region so that the precision of the query depends only on the precision of the query object. The object-oriented R*Tree uses the query object's bounding box during the tree traversal but the final intersect or contain operation is performed with the actual query object, thereby guaranteeing accurate results.

The flexibility of the IndexedSpatialSet class is seen in its ability to perform symbolic queries, spatial queries, or combined symbolic and spatial queries. For purely symbolic queries, such as "*Find all objects with names between 'K' and 'M'*," the query request is redirected to the *objectSet* where the built-in accessing methods are utilized. For purely spatial queries, such as "*Find all objects intersecting object O*," the query request is redirected to the *spatialIndex* which is more efficient in finding objects spatially.

Queries that combine symbolic and spatial predicates are more complicated. The current system relies on the spatial access for any queries containing spatial predicates. However, at the leaf nodes, when the actual objects are interrogated, the symbolic predicates are checked before the final spatial requirements. This heuristic is based on experience gained from testing various combinations of queries and appears to provide the best performance so far. Additional

investigation is needed in query optimization for a more thorough solution to this problem.

We have constructed a very simple interface to experiment with spatial query concepts on a data set of randomly generated two-dimensional objects. Figure 5 shows the layout of the query interface with a set of 1000 spatial objects from classes we've named City, Crop, Lake, River, and Road.

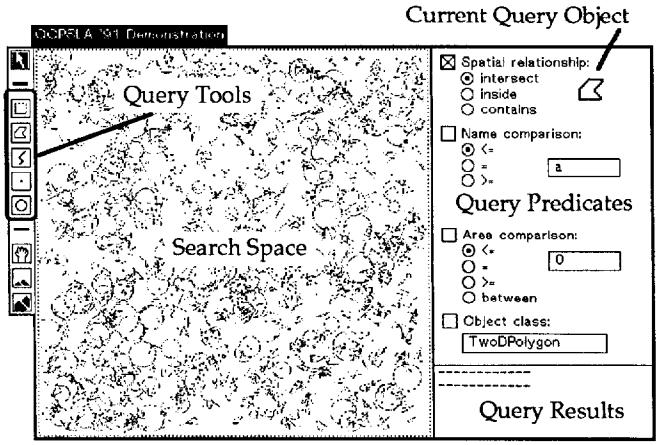

Figure 5. A simple spatial query interface

The set of possible spatial query objects are displayed as drawing tool icons on the left-hand side as: rectangle, polygon, polyline, point, and circle. The two-dimensional search space is shown as a bitmap image (Smalltalk Form) containing all the objects in the data set. This image can be scaled and scrolled in all directions within its window.

Spatial and symbolic query predicates are specified on the right-hand side of the interface. Check boxes indicate inclusion/exclusion of the predicate in the query, currently combined only by the AND operator. Specific operations for each type of predicate are selected through radio buttons. The set of objects returned by the query appear as a scrolling list in the lower right-hand corner of the interface.

Spatial queries are formulated graphically. The spatial query object is specified by selection of an appropriate drawing tool and is drawn directly in the search space, denoting its location and boundary. Figure 6 shows the result of executing a spatial intersect query with a polygon object. For purely spatial queries such as this example, the R*Tree index uses the bounding box of the polygon object during tree traversal but relies on the polygon query object itself to check the final spatial predicate. The objects returned are listed by name and are drawn in the search space.

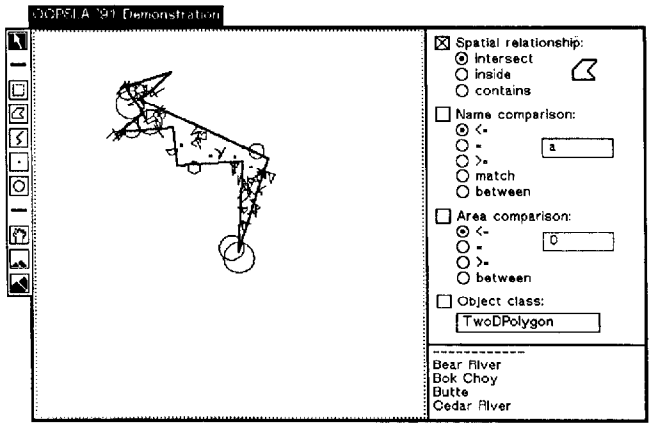

Figure 6. All objects intersecting a polygon

Combinations of spatial and symbolic queries are formulated by checking multiple check boxes in the query predicate area. Figure 7 shows the result of a query involving all four possible predicates. The query asks for instances of the class Lake that are inside the circle object having names <= 'West' and areas between 400 and 600. In this case, the R*Tree index uses the bounding box of the circle object during tree traversal. As each candidate object is found, the symbolic predicates are checked before the final spatial predicate using the circle query object.

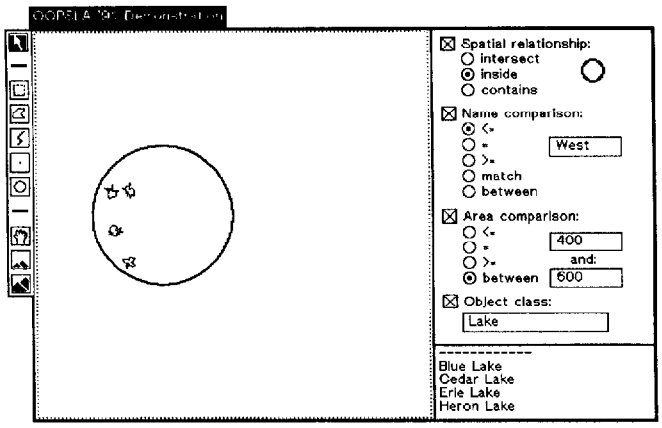

Figure 7. Combination spatial and symbolic query

Queries in an object-oriented database can be done in many ways. As yet there does not exist an equivalent SQL-like declarative language for specifying arbitrary queries. The GemStone database includes the OPAL language, which closely resembles Smalltalk, for data definition and data manipulation. In the current prototype we generate a GemStone Block object for symbolic predicates. This block is passed to the IndexedSpatialSet object in the database for evaluation. For queries involving spatial and symbolic predicates, the block object is passed to the R*Tree where it is evaluated at the leaf node level.

6 SUMMARY & FUTURE WORK

In this paper, we have described the development of a database to support three-dimensional image reconstruction of structural biology using object-oriented technology. The application domain is characterized by multiple data types and multi-levels of abstraction of data obtained from images. Much of this information is spatial data, so particular attention was placed on the implementation of the spatial data, its representation, operations, indexing, and queries.

We represent spatial data through the class hierarchy of three-dimensional objects. The ThreeDObject class contains generalizations of all 3-D objects and serves as the superclass for current and future specializations. We implement spatial relationship operations through double-dispatching techniques that provide an efficient method of choosing appropriate algorithms based on the class of the argument of a message and the class of the receiver.

A container class for spatial objects was defined that has two instance variables, a Set object that holds the spatial objects and provides access based on symbolic attributes, and an R*Tree index object that provides efficient spatial access to the object set. We have shown how the object-oriented R*Tree extends the functionality of spatial indices through accuracy of query results and precision of query region. This was possible because of the object-oriented features of polymorphism and late binding.

Our work presented here accomplishes the framework for a database schema that outlines the representation, operations, and access methods for spatial objects and a spatial/symbolic query interface. As a prototype, the current system is in its development stage and requires further refinement before it can be integrated with the data acquisition and production system in Biological Structures. Once completed, it will have an impact on and improve the 3-D reconstruction efforts in Biological Structures by providing advanced data management capabilities with increased functionality and support for spatial data.

The spatial query examples shown in this paper were done with a set of two-dimensional objects. Formulation of 3-D query objects will be accomplished through an enhancement to the 3-D editor application currently in use [PROT89] to include support for arbitrary 3-D query object construction and connection to the database. The R*Tree spatial index is designed to support objects of any dimension so it requires no modifications.

One path of research currently underway is the presentation of symbolic queries in a more generalized fashion. The interface should be dynamic such that the choices presented reflect the set of instance variables in the scope of target objects. For instance, queries over all ThreeDObjects can be formulated according to those instance variables common to all ThreeDObjects. However, if the query is narrowed to only Contour objects, then the predicate choices should include those variables common to all Contours. In this way, the presentation of query predicates can be generated as the query is formulated interactively.

One observation we have regarding spatial queries is the proportion of time spent on computation of spatial relationships done by the objects compared to the time spent in traversal of the index. Intersection seems to be the most computationally intensive operation and appears to be the bottleneck that overshadows the index search. As some spatial operations are hindered by the data representation, we are looking at alternative representations, the roles they play in aiding spatial operations, and the feasibility of object conversions in the database.

ACKNOWLEDGEMENTS

We wish to thank the 3D Reconstruction researchers in the Department of Biological Structures for their assistance on the work presented here. We also wish to thank IBM for support provided through a Graduate Student Fellowship awarded to the first author, and the W. M. Keck Foundation for initial support of this research.

REFERENCES

BECK90 N. Beckmann, H-P Kriegel, R. Schneider, B. Seeger. The R*Tree: An Efficient and Robust Access Method for Points and Rectangles. *Proceedings ACM-SIGMOD International Conference on the Management of Data*, 322-331, 1990.

BENS91 D. Benson, G. Zick. Obtaining Accuracy and Precision in Spatial Search. *Technical Report DEL-91-01*, Department of Electrical Engineering, University of Washington, 1991.

BRIN89 J. F. Brinkley, J. S. Prothero, J. W. Prothero, C. Rosse. A Framework for the Design of Knowledge-Based Systems in Structural Biology. *Proceedings Thirteenth Annual Symposium on Computer Applications in Medical Care*, IEEE Computer Society Press, 61-65, 1989.

DUER83 A. J. Duerinckx, S. J. Dwyer. Guest Editors' Introduction: Digital Picture Archiving and Communication Systems in Medicine. *Computer, Vol. 16, No. 8, Special Issue on Digital Image Archiving in Medicine*, 14-16, August 1983.

GEMS90 Gemstone Object-Oriented Database Management System, Version 2.0, Servio Corporation, 1990.

GOLD83 A. Goldberg, D. Robson. *Smalltalk-80 The Language and its Implementation.* Addison-Wesley, 714 pgs., 1983.

GUTT84 A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. *Proceedings ACM-SIGMOD International Conference on the Management of Data*, 47-57, 1984.

GÜNT88 O. Günther. *Efficient Structures for Geometric Data Management*, Lecture Notes in Computer Science 337, edited by G. Goos and J. Hartmanis, Springer Verlag, 1988.

MCLE91 M. McLean, J. Prothero. Three-dimensional reconstruction from serials sections. V. Calibration of dimensional changes incurred during tissue preparation and data processing. Analytical & Quantitative Cytology and Histology (in press), 1991.

OREN90 J. Orenstein. A Comparison of Spatial Query Processing Techniques for Native and Parameter Spaces. *Proceedings ACM-SIGMOD International Conference on the Management of Data*, 343-352, 1990.

PARC90 Objectworks for Smalltalk-80, Version 2.5, ParcPlace Systems, Inc. 1990.

PROT89 J. S. Prothero, J. W. Prothero. A software package in C for interactive 3-D reconstruction and display of anatomical objects from serial section data. *NCGA Conference Proceedings*. I:187-192, 1989.

STIM88 G. K. Stimac, J. W. Sundsten, J. S. Prothero, J. W. Prothero, R. Gerlach, R. Sorbonne. Three-dimensional Contour Surfacing of the Skull, Face, and Brain from CT and MR Images and from Anatomic Sections. *AJR* 151:807-810, 1988.

ULLM88 J. Ullman. *Principles of Database and Knowledge-Base Systems, Vol. 1.* Computer Science Press, 1988.