SPATIAL OPERATORS FOR OBJECT-ORIENTED GEOGRAPHIC INFORMATION PROCESSING

Mark J. Carlotto and Jennifer B. Fong
The Analytic Sciences Corporation
55 Walkers Brook Drive
Reading, MA 01867

ABSTRACT

The design and implementation of spatial operators that use multiple spatial representations within an object-oriented geographic information system are discussed. The data structures used are capable of representing point, line, and areal features. Selection of a representation is based on the nature of the computation involved. Conversions between representations occur transparently. Examples of spatial operators that use symbolic and iconic data structures are described. A case study illustrating the use of the object-oriented paradigm within a GIS application is included.

INTRODUCTION

Geographic information systems are often classified in terms of their underlying data structures, e.g., whether they are raster- or vector-based, or use quad-trees. On the other hand, it may be more meaningful to think of the world being composed of objects, (Gahegan and Roberts 1988), especially within the context of knowledge-based systems (Smith, et al 1987) (McKeown 1987) (Fong et al 1990). A result has been the emergence of object-oriented geographic information systems in recent years. While the advantages of one data structure over another are subject to debate, it is generally recognized that multiple representations are advantageous.

This paper presents a uniform approach to the design and implementation of spatial operators that use multiple spatial representations within an object-oriented geographic information system. First some basic concepts in object-oriented programming are reviewed. This is followed by a description of the underlying spatial representations that are used. After a brief description of the software architecture, representative spatial operators are described. A case study is presented last to illustrate a GIS application.

OBJECT-ORIENTED PROGRAMMING

Object-oriented programming allows the combination of data structures and procedures for accessing and manipulating data structures into *objects*. Once an object has been defined, copies of that object may be *instantiated*. In addition to the modularity inherent in object-oriented programming, interactions between an object instance and the outside world takes the form of uniform *messages* which are handled by the object without requiring the sender to be aware of the processing involved. Thus, object-oriented programming offers a powerful tool for creating systems that are highly modular, flexible, and maintainable.

The object-oriented spatial operators described in this paper were developed as part of a larger knowledge-based GIS described elsewhere in these proceedings (Fong et al, 1990). Although the most

recent version of this system uses a commercial object-oriented database management system, the initial version was developed almost entirely in Common Lisp and is described here. Data types are defined in terms of object class descriptions and operators for handling messages to instances of object classes. Objects are known as *tokens* and are implemented using Common Lisp *structures*. A structure is defined with the creation of a new token class using the form:

(deftoken token-class (mixins) (attributes))

where *mixins* are other token classes and *attributes* are the possible slots for *token-class*. Mixins allow tokens to be constructed hierarchically from other tokens, inheriting attributes, values, and operators. *Regions* are a basic token class used to represent connected pixel sets

(deftoken region () (label polygon x-raster y-raster boundary-points bounding-rectangle)).

The attributes provide local storage for a region's spatial representation(s). Other object classes that have spatial extent can then be built upon regions using *mixins* ; e.g.,

(deftoken surface-material (region) (class area)).

Messages to tokens are handled by operators attached to a structure attribute. An operator is defined with the form

(defop (message token-class) (arguments) body)

and would be called as

(send-message token-instance message (arguments)).

Examples of spatial operators are provided later in the paper.

SPATIAL REPRESENTATIONS

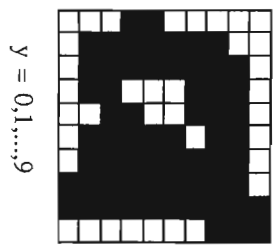
From a data structures point of view, it is natural to think of geographic objects as being divided into points, lines, and regions (Nagy and Wagle, 1979). However, to do so increases the number of kinds of spatial operations that must be performed by the GIS. For example, if different spatial representations are used, up to six different algorithms may be needed just to compute the distance between two objects. One design goal was to select multiple data structures, each capable of representing point, line, and areal features. Since computation is performed on a discrete grid, they can all be represented as connected pixel sets. Given that different representations are better suited to certain types of spatial computation, a second design goal was to develop a means for converting between representations in a transparent fashion.

Six kinds of spatial representations are currently used: label maps, x-rasters, y-rasters, polygons, boundary points, and bounding rectangles.

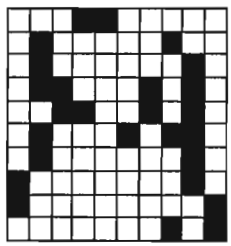
Label Maps are the most direct way of representing thematic data in iconic form. The label map partitions a database layer into disjoint pixel sets which may, or may not be spatially connected. Each pixel set is assigned a unique number (label) corresponding to the

slot value of the object that it represents.

X- and y-rasters are used to represent connected sets of pixels using row- and column-oriented run-length encoding schemes. They are based on the concept of a tightly closed boundary (Merrill 1973) defined below. (The use of the term raster in this context should not be confused with images.) Raster representations may be generated from label maps or from polygons. X-rasters are defined by a y offset (the y coordinate of the top row of the connected set) and an array of x coordinates. The elements of the array contain the lists of the x coordinates which define the beginning and end points of runs of pixels in each row. The y coordinate of the run is equal to the array index plus the y offset. The length of the array is equal to the number of rows in the connected set. In Fig. 1 the x-raster representation of the bitmap (a) is depicted in (b). A similar approach is used to represent y-rasters which are defined by an x offset (the x coordinate of the leftmost column of the pixel set) and an array of y coordinates which define the beginning and end points of runs of pixels in each column (c).

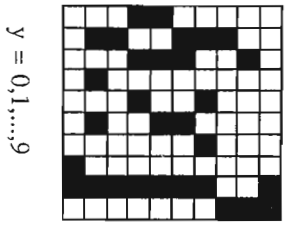


(a) Connected region



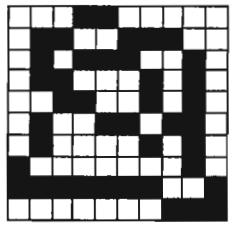
(3 4)
(1 7)
(1 8)
(1 2 6 8)
(2 3 6 8)
(1 5 7 8)
(1 8)
(0 8)
(0 9)
(7 9)

(b) X-raster representation



(7 8)
(1 3 5 8)
(1 8)
(0 2 4 8)
(0 2 5 8)
(1 2 5 8)
(1 4 6 8)
(1 9)
(2 9)
(8 9)

(c) Y-raster representation



(d) Boundary points

Fig. 1 Spatial representations based on the tightly closed boundary

Boundary points are the pixels which comprise the boundary of a polygon or a connected pixel set represented in a bitmap. The boundary points define a tightly closed boundary (Merrill, 1973) where adjacent points are no farther than a diagonal distance apart. The boundary points of an object are the union of its x- and y-rasters (Fig. 1d).

Polygons provide a means for representing information that is manually specified or has been generalized. They are stored as a list of vertices and may be converted to rasters, boundary points, or bounding rectangles. Bounding rectangles are a special type of polygon which are used for range searching by certain spatial operators.

For each representation, a looping macro is defined to facilitate access of pixels within that representation. In the spirit of object-oriented programming, conversions between representations take place transparently as they are needed. For example, if the boundary points of an object are needed by a spatial operator and the x- and y-rasters exist, then the boundary points are computed from the intersection of the x- and y-raster representations; if the polygon exists, it is computed by generating the intermediate points between the polygon's vertices; otherwise, the user is notified that the operation cannot be performed with the available data.

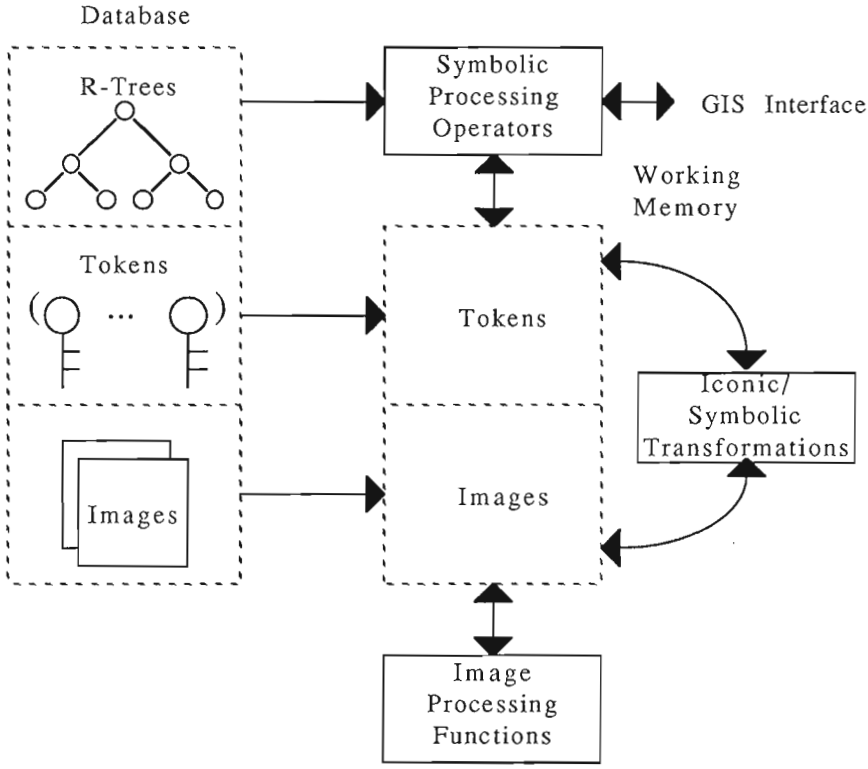


Fig. 2 Hybrid (image/symbol) architecture for spatial analysis

SOFTWARE ARCHITECTURE

Fig. 2 depicts the software architecture. The database pictured to the left is partitioned into areas for storing images, tokens and R-trees. A database management system, described in (Fong et al 1990) is responsible for moving the various data structures in and out of working memory. In the most recent implementation, R-trees are being used by some symbolic processing operators for range searching. Transformations between iconic (i.e., images) and symbolic domains are performed as part of certain spatial operations and include connected pixel labeling and functions for writing tokens into arrays. Image processing includes morphological operations on bitmaps, and thresholding and averaging on numerical data (e.g., slope).

SPATIAL OPERATORS

Spatial operators may be divided into two types: unary operators that compute information about a single object at a time, and n-ary operators that compute information about relationships between objects.

Unary operations include those for computing the area, perimeter, compactness, length, width, and average value of an object. As an example, the implementation of an operator that computes the average value of an object over an array is presented below:

```
(defop (:average token) (array)
 (let ((result 0))
      (loop-x-raster (send-message myself :x-raster) 1 1 array)
        (incf result (aref array x y)))
      (send-message myself :set-average
                                  (/ result
                                    (send-message myself :area)))))
```

When the form *(send-message token-a :average slope)* is executed, if the slot value of *average* in *token-a* is *nil*, the above operator is invoked. The macro *loop-x-raster* generates (x,y) addresses within the support of *token-a*. The average value is computed over the array *slope* and the result is stored in the slot *average* of *token-a*.

Another example of a unary operation is computing the length and width of an object with respect to a given direction ϕ. An object's bounding rectangle can be determined from its boundary points by finding the largest and smallest values of x and y. The length and width in the direction ϕ can reduced to the bounding rectangle calculation by rotating the boundary points

$$x' = x \cos \phi + y \sin \phi$$
$$y' = -x \sin \phi + y \cos \phi$$

and finding the largest and smallest values of x' and y'.

N-ary operations include those that do not require auxillary representations such as centroid-to-centroid distance and containment, and those that do, e.g., adjacency, minimum distance, and line of sight. The syntax for containment is typical of the n-ary operators:

(send-message token-a :contains other-tokens)

which returns all tokens from the list *other-tokens* that are contained within *token-a*. The dual operator *:is-contained-by* returns all tokens in the list *other-tokens* that contain *token-a*. The problem of determining whether or not a point is contained in a given region is well known. Merrill (1973) shows that region containment is easily solved using TCBs. However when regions are disjoint, the TCB containment algorithm must be modified.

Consider the problem of determining if the connected region represented by the six white interior pixels are contained in the black connected region in Fig. 1. Evidently the intervals (3 5), (4 5), and (6) lie outside the corresponding intervals (1 2 6 8), (2 3 6 8), and (1 5 7 8) which implies that the white region is not contained within the black region. A modified containment algorithm was thus developed that requires that the intervals of the x- and y-rasters of object-a lie within the corresponding "outer" intervals of the x- and y-rasters of object-b; i.e., that the x intervals (3 5), (4 5), and (6) lie within (1 8), (2 8), and (1 8); and that the y intervals (3), (3 4), (3 4), and (5) lie within (0 8), (0 8), (1 8), and (1 8).

Operations like containment may need to examine a large number of objects. As a result it is important that the list of candidate objects be pruned using an inexpensive operation, e.g., by eliminating all objects whose bounding rectangle is not contained by that of the object being queried. In the most recent implementation R-trees are used, providing much improvement over a linear search of bounding rectangles.

Auxillary representations are also used by certain spatial operators as mentioned earlier. Consider the adjacency operator:

(send-message token-a :adjacent-to other-tokens)

which returns all tokens from the list *other-tokens* that are adjacent to *token-a*. A direct way for computing adjacency involves creating a label map of *other-tokens* and looping around the boundary of *token-a* building a list of unique labels. If, when the label map is built, another data structure that indexes tokens by their label is also built, the list of labels collected along the boundary can be converted to a list of adjacent tokens.

A common query is to return all objects from one set that are a certain edge-to-edge distance from another set. The computational complexity can be prohibitive when there are a large number of objects in both sets. An alternative is to use a distance transformation (Rosenfeld and Pfaltz 1966) that involves marking one set of objects with zeros in a array and computing the distance from any pixel in that set to all pixels in the array. The distance transform algorithm requires two passes over the array:

$$d(m,n) = \begin{cases} 0 & \text{if } d(m,n) = 0 \\ \min\,[d(m-1,n) + 1,\ d(m,n-1) + 1)] & (m,n) \neq 1 \\ M + N & \text{otherwise} \end{cases}$$

in the first pass performed left to right and top to bottom, and

$$d(m,n) = \min\,[\,d(m,n),\ d(m+1,n) + 1,\ d(m,n+1) + 1\,]$$

in the second pass performed right to left and bottom to top. Once

the distance image has been computed, the edge-to-edge distance to each object is found by looping along the edge of the object.

It should be noted that for single object queries, use of auxillary label maps may be inefficient since the computation will be of the order of the size of the label map.

CASE STUDY: AREA PRIORITIZATION

A knowledge-based system for prioritizing large geographic areas was the primary motivation for developing the spatial operators described in this paper. The system identifies areas that satisfy a set of constraints expressed in the form of rules. The rules are grouped into frames which are further organized in a hierarchical structure. In the earliest implementation, constraints returned the objects that satisfied the constraint; e.g.,

(distance-between forests roads '< 100)

would evaluate the form

(send-message forest-n :distance-between roads)

for each token in the list *forests*. An object would be returned if it satisfied the specified predicate and value. The current version uses fuzzy predicates instead of hard constraints and returns the object along with a value between zero and one.

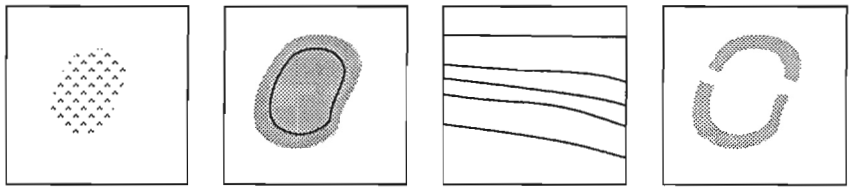


Fig. 3 Segmentation and the creation of new objects

The inference process can be viewed as a series of symbolic filters that are applied to the objects that represent the underlying terrain. Constraints can also lead to the creation of new objects by segmenting objects from one database layer with information from another. Fig. 3 depicts the extraction of open areas that are adjacent to a forest and illustrates the use of image processing techniques within the GIS. The forested region is marked with zeros in a working array d(x,y) and dilated using the distance transform discussed earlier. The distance image is thresholded by an amount proportional to the desired width of the adjacent areas. An auxillary constraint is that the slope within a newly created connected region must be less than some amount. The slope image is thresholded and intersected with the thresholded distance image. Tokens are then created and returned for subsequent processing by the GIS.

SUMMARY

The implementation of spatial operators using multiple representations within an object-oriented GIS was described. The added complexity in using multiple representations can be ameliorated, in part, by using an object-oriented paradigm. The advantages of using iconic representations may depend on the nature of the processing involved. In particular, the greatest benefit is achieved for certain n-ary operations involving large numbers of objects. Additional research into the advantages of mixed representations is needed.

REFERENCES

M. J. Carlotto and J. B. Fong, "Omega: An object-oriented image/symbol processing environment," *SPIE*, Vol. 1003, November 1988, Cambridge MA.

M. N. Gahegan and S. A. Roberts, "An intelligent object-oriented geographical information system," *Int. J. Geographical Information Systems*, Vol. 2, No. 2, 1988.

D. M. McKeown, "The role of artificial intelligence in the integration of remotely sensed data with geographic information systems," *IEEE Trans. Geoscience and Remote Sensing*, Vol. GE-25, No. 3, May 1987.

R. D. Merrill, "Representation of contours and regions for efficient computer search," *Communications of the ACM*, Vol. 16, No. 2, Feb. 1973.

G. Nagy and S. Wagle, "Geographic data processing," *Computing Surveys*, Vol. 11, No. 2, June, 1979.

A. Rosenfeld and J. L. Pfaltz, "Sequential operations in digital picture processing," *Journal of the ACM*, Vol. 13, No. 4, October, 1966.

T. Smith, D. Peuquet, S. Menon, and P. Agarwal, "KBGIS-II: A knowledge-based geographic information system," *Int. J. Geographical Information Systems*, Vol. 1, No. 2, 1987.