Recognition and Representation of a Hierarchy of Polygons with Holes

G. H. KIRBY, M. VISVALINGAM AND P. WADE
Department of Computer Science, University of Hull, Hull HU6 7RX

Polygons with holes must be handled in digital cartography, 2D and 3D modelling, image analysis and picture generation. The Dissassociative Area Model (DAM) described in this paper uses polygonal units of uncut space, termed primitive regions, in the representation of the geometric and topological properties of a set of areal entities. Algorithms and a data structure are presented which extract boundaries of these polygons from vector data and determine their topology. The algorithms recognise the containment hierarchy of polygons with holes without the need for information relating to the areal entities. The hierarchy is represented by a rooted tree of boundaries, rather than regions, and does not involve bridges between inner and outer boundaries.

Received September 1987, revised November 1987

1. INTRODUCTION

The derivation of an appropriate conceptual model for describing areal entities is of concern to many activities in computer graphics. Areal entities are represented on a computer by geometric information (which describes the course of their boundaries), topological information (which describes the spatial units to which boundaries belong) and object-related information (which describes the identity and/or attributes of areal entities to which the spatial units belong). When shading areas both user and computer view spatial units as a set of polygons, describing uncut planar surfaces. In contrast, when displaying the linework of areal boundaries, these outlines are perceived as a network of boundaries between neighbouring spatial units.

In this paper we are concerned with the modelling of the geometric and topological properties of a set of static areal entities and the associated algorithms and data structure. The unit of uncut space forms the basic spatial unit in this model and acts as the link between the geometric and object-related information. The topological information is represented in terms of these uncut spatial units by holding the adjacencies between them, the points where they meet (i.e. their connectivity) and their containment hierarchy. The containment hierarchy arises from the existence of 'holes' in areal entities.

There is a need to automatically extract the above topology in an elegant and efficient manner and represent it in a complete and coherent form within an internal working model. Given such a model, it would be easy to extract alternative data structures most attuned to existing algorithms or differing tasks, such as data capture, data analysis, display or transfer.

Spatial units, regions or polygonal faces with holes pose problems in the modelling of 3D objects, in image analysis, in picture generation and in digital cartography. Kilgour reports[8] that some polygonal representations, such as the CIF format used in VLSI, represent faces with holes as a single loop by introducing additional edges to join a point on the outer boundary to a point on the hole. The polygon resulting from the insertion of this 'bridge' is no longer simple but many graphics and geometric algorithms handle this type of polygon successfully. The Graphical Kernel System (GKS) uses this strategy for coping with holes, since the *fill area* primitive accepts only a single loop (ref. 8, p. 61). Ballard[2] similarly uses the contrivance of a bridge in the strip tree representation of a region with a hole.

Bridges are usually established by the user during digital input. In image analysis, though, scanned images and scenes may be automatically segmented into regions and represented as vector-encoded maps for spatial modelling.[12, 13] Consequently, the use of bridges requires either automatic or manual insertion of these connections, i.e. the nesting of holes within polygons must either be specified, or preferably, automatically extracted.

The input of all necessary topological data during vector digitising makes data capture a time-consuming, error-prone and costly exercise. Consequently, mapping agencies such as the Ordnance Survey (OS) of Great Britain adopt pragmatic data models and data structures for large-scale digitising. The link and node structure used by the OS does not directly describe areal objects, for example administrative areas. Instead, it encodes point and line features, which provide fragmented and scattered clues about areal objects. The extraction of a complete and coherent description of hierarchically organised administrative areas from feature-coded map details also requires the automatic deduction of topologic information.[16]

The Working Group on Terms and Definitions of the American National Committee for Digital Cartographic Data Standards held that 'holes in cartographic objects constitute a gap in our knowledge' (ref. 10, p. 24). A number of data models and data structures have been advanced for the representation of areal entities by researchers in digital cartography.[1, 3-5, 9, 11] The hierarchical polygonal data structure proposed by Edwards[4] for representing areal entities with holes, holes in holes and so on, relies on the input of object-related information, in the form of left/right codes for the areal objects on either side of the boundaries. Such left/right codes originated in the Dual Independent Map Encoding (DIME) system of the U.S. Bureau of the Census.[3] The GIRAS system, used by the U.S. Geological Survey[9] fixes the outer and inner boundaries of a spatial unit with holes with object-related information attached to an

Correspondence to: Dr G. H. Kirby at the above address.

areal seed, also called a polygon label. Containment relationships are explicitly recorded within GIRAS. Ref. 6 (p. 13) described one reason for this, namely that this would make it easy for small polygonal areas to be eliminated or combined with larger surrounding areas during cartographic or information generalisation.

ARC/INFO,[5] a current example of a comprehensive geographical information system (GIS), accepts the input of locational information without left/right codes or polygon labels. It is possible for the user to discover that holes exist in a spatial unit from the topological information automatically derived by the ARC component of the system. Within ARC, the spatial extent of each polygonal area is described by a list of the lines needed to create its boundaries; the lines of the outer boundary being the first in the list. However, neither this list nor the co-ordinates of the arcs are available through INFO and users only have access to INFO and not ARC. It is difficult therefore, and sometimes impossible, to answer questions about the containment relationships between areal objects. For example, a user would find it very difficult to establish whether A contains B or B contains A, or to determine the areas contained by A. Wade *et al.*[17] provide a more detailed review of data models for representing areal entities in cartography.

This paper describes a new model, the Disassociative Area Model (DAM), for representing the spatial relationships between polygons. It disassociates the geometric from the object-related descriptions of areas. A data structure and algorithms are presented for progressing automatically from a node-matched set of vectors, which describe the linework of boundaries, to a complete topological description of the uncut spatial units which can be extracted from the boundaries.

2. DISASSOCIATIVE AREA MODEL (DAM)

Conceptually, the vector encoding of polygonal areal entities can be broken down into three distinct stages:

(1) the geometry of the linework of the areal boundaries is encoded;

(2) the topology (or spatial relationships) is determined; and

(3) the geography of the areal entities is encoded.

There are several advantages in separating these stages. There can be flexibility in the order and the method by which data is supplied to the system if the geometry and the geography are considered separately. It is possible to derive the topology by an automatic process, thus minimising the information which has to be input by the user and reducing the possibility of error. The separation of the geometry from the geography enables a data structure to be used which can represent complex relationships between areas (such as overlaps, hierarchies, holes and islands), and also allows sets of areas to be held in the same database. If the topological description of the geometry is also held within the data structure, it is possible to use this knowledge to perform spatial operations on areas and on data associated with areas; for example, the disaggregation and reassignment of attribute data from one set of areas to another. The computed topology can be used to check for completeness and consistency in the data supplied, and to ensure that any rules about how areas are related (such as in a hierarchy) are obeyed in the data.

2.1 Geometry and topology

Using only the geometry of the bounding lines, the areal map can be dissected into a set of non-overlapping parts, which we call primitive regions. For example, the linework of Fig. 1*a* results in eight primitive regions (A–H), including one (A) which surrounds all the other primitive regions. The primitive region is similar to the least common geographic unit of GEOGRAF[11] and to the regions resulting from image decomposition.[13]

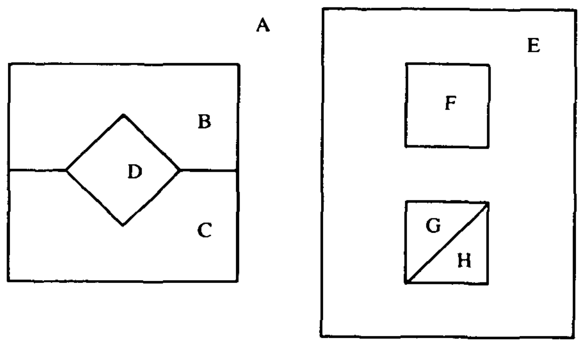
Primitive regions exist only in concept. The representation of the topology hinges on the boundary. Figure 1*b* shows the set of boundaries required to describe the primitive regions of Fig. 1*a*. Each boundary is a closed loop, with direction, which defines one extent of a primitive region. A boundary can be sub-classified as being either an enclosing boundary or a hole. The outer boundary of a primitive region is known as an enclosing boundary, and any inner boundaries are known as holes. Thus each boundary forms an extent of one, and only one, primitive region and each primitive region is bounded by one enclosing boundary and zero or more holes. There is one exception to this – the outermost primitive region has no enclosing boundary but just one or more holes (in Fig. 1*b* the outermost primitive region is described by holes 1 and 5). This primitive region forms the complement of the union of all the other primitive regions in the plane surface.

Each boundary has a separate existence within our data structure. Boundaries are equivalent to a closed loop in geometry, but they also have an associated direction to distinguish enclosing boundaries from holes. Where one primitive region completely surrounds another primitive region, the hole in the outer primitive region and the enclosing boundary of the inner primitive region are identical in shape. The two boundaries, however, remain unique since they have opposite direction (for example see boundaries 7 and 8 in Fig. 1*b*). The direction of a boundary thus relates it to one specific primitive region. However, in our data structure the geometry of two such boundaries is stored once only.

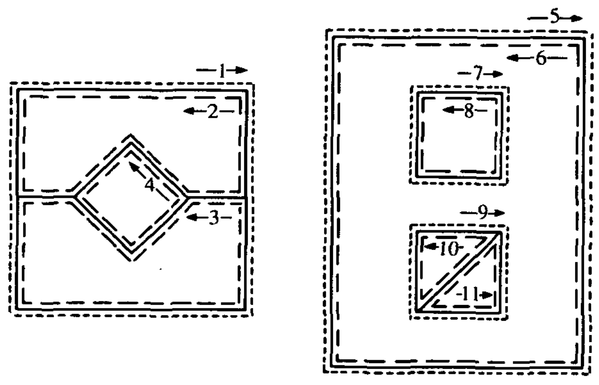
DAM assumes that lines do not cross each other except at junction points called nodes. Nodes form the end points of polylines, which are referred to as links or arcs in cartographic applications. DAM requires that links are node-matched in order to extract the boundaries by using the algorithms described later. It is, however, unconcerned as to whether the link geometry is represented by a list of co-ordinates or in parametric form. The latter would require minor changes in the data structure, replacing the co-ordinates by the parameters of each link.

When primitive regions all occur at the same level, that is, when there are no nested holes, there is a one-to-one correspondence between boundaries and primitive regions. For example in Fig. 1, if the outermost primitive region, A, is excluded from consideration, there is a one-to-one correspondence between B, C, D and 2, 3, 4 respectively. Thus each primitive region may also be assigned the identity of its enclosing boundary. In our data structure (Section 3.1), links provide the adjacency relationships between boundaries and their corresponding primitive regions.

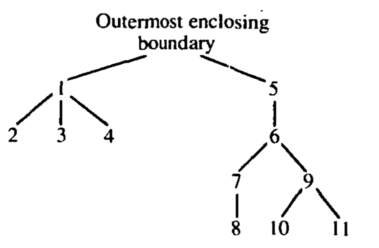
When primitive regions are nested and therefore contain holes, there is a many-to-one correspondence



(*a*) The primitive regions formed by the linework of a map. Key:———, links; A–H, primitive regions.



(*b*) The set of boundaries which describe the above primitive regions. Key: ———, links; --1→--, holes; —←2—, enclosing boundaries (with their internal system numbers and direction).



	Level in tree	Type of boundary
....	0	Enclosing
....	1	Hole
....	2	Enclosing
....	3	Hole
....	4	Enclosing

(*c*) The rooted tree representing the containment hierarchy of the boundaries

Figure 1. Representation of the geometry of areal objects. *(a)* The primitive regions formed by the linework of a map. *(b)* The set of boundaries which describe the above primitive regions. *(c)* The rooted tree representing the containment hierarchy of the boundaries.

between boundaries and such primitive regions. The links continue to provide the adjacency relationships between boundaries, and also between clusters of adjacent primitive regions. The problem is that the identity of primitive regions containing holes remains unknown. What exists, instead, is the distinct references to separate boundaries at the outer and inner extent of such primitive regions, and no single reference to the primitive regions themselves. For example, boundaries 6, 7 and 9 remain as separate references and there is no knowledge at this stage that they all belong to E.

The containment relationships between a complete set of boundaries can be viewed as forming a hierarchy which can be represented by a rooted tree. The root of the tree consists of a nominal reference to the enclosing boundary of the outermost primitive region, that is, the part of the plane surface which surrounds all the other boundaries. Each boundary is enclosed spatially by every boundary which precedes it in the tree but no other. When two boundaries are identical in shape, the one which is the hole is considered as surrounding the one which is the enclosing boundary. Thus the level of each boundary in the tree is equal to the number of other boundaries which surround it. Also, boundaries at even levels of the tree are enclosing boundaries and those at odd levels are holes. Fig. 1*c* illustrates the rooted tree. Note that the tree is not an ordered rooted tree as there is no set order to the edges leaving each vertex of the tree.

This hierarchical system can be applied to any set of boundaries irrespective of their complexity. For example, currently the area within hole number 7 is uncut and forms a single primitive region. It could feasibly be cut into a number of primitive regions, as is the area within hole number 9. Some of these primitive regions may in turn contain further holes. This would all be represented by suitable branches starting from boundary 7 in place of the one shown.

It is also possible that a map frame will be digitised around all the boundaries. This would have the effect of creating additional boundaries and primitive regions. If the map frame does not touch any of the other boundaries, then the hole and enclosing boundary created by the frame would be at levels 1 and 2 respectively in the tree, and all the other boundaries would be moved down by two levels.

The derivation of such a tree *fully* resolves the containment relationships between the boundaries – and thus also the spatial extent of the primitive regions – since the holes within each primitive region immediately follow the enclosing boundary for that primitive region in the tree. The set of holes at level 1 of the tree describe the holes in the outermost primitive region, whose enclosing boundary is undefined. The tree also makes explicit the nesting of primitive regions within holes in other primitive regions. The derivation of this tree for a set of boundaries is a one-off process for static data requiring an exhaustive spatial search. An algorithm which attempts to minimise the computation required is given in Section 3.2.

Once a primitive region assumes an identity, it becomes the basic building block for spatial modelling and forms the connection between the geometry and the geography.

Equally, the importance of the concept of the boundary should be realised. ARC/INFO and GIRAS do not maintain a separate identity for each boundary, but instead just give an ordered list of arcs (links) which describe the outer and inner extents of each polygon. Thus if these systems were to make explicit the containment hierarchy in a similar way to DAM, it would be a hierarchy of polygons (that is, primitive regions) rather than a hierarchy of boundaries. Whilst it would give the containment of polygons within other polygons, it would not indicate the clusters of polygons which lie within each hole of another polygon. The rooted tree shown in Fig. 2 shows the containment hierarchy of the primitive regions of Fig. 1*a*. Compare, for example, the clustering of primitive regions F, G and H in Fig. 2 with Fig. 1*c*. Some of the information concerning separation and nesting is lost. The same criticism applies to the tree representations obtained by using the relationship 'inside of' and applied to composite regions in image description (ref. 7, p. 374).

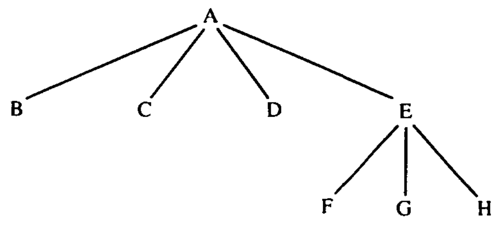


Figure 2. The rooted tree representing the containment hierarchy of the primitive regions in Figure 1*a*.

2.2 Geography

Areal objects are specific instances of different types of spatial phenomena. Some applications require only simple objects, such as a description of the administrative areas at a specified level or the faces of a polyhedron. Others require complex objects, such as a hierarchy of administrative areas. Each application may define its own set of types of simple and complex objects.

The spatial descriptions and other attributes of areal objects may be provided in different formats by various sources, and applications may require access to some or all of these descriptions. DAM provides an application independent model of geometric entities and a framework for integrating the geographic and geometric descriptions where necessary. At the bottom-most level, there may be a one-to-one mapping between objects and primitive regions in the simplest case (Fig. 3*a*). Where there are disjoint parts, there is a one-to-many relationship between objects and primitive regions (Fig. 3*b*). DAM also allows a many-to-many mapping between objects and primitive regions (Fig. 3*c*). Thus it copes with both hierarchic objects (e.g. administrative hierarchies) as well as objects which overlap at any one level (e.g. broadcasting areas).

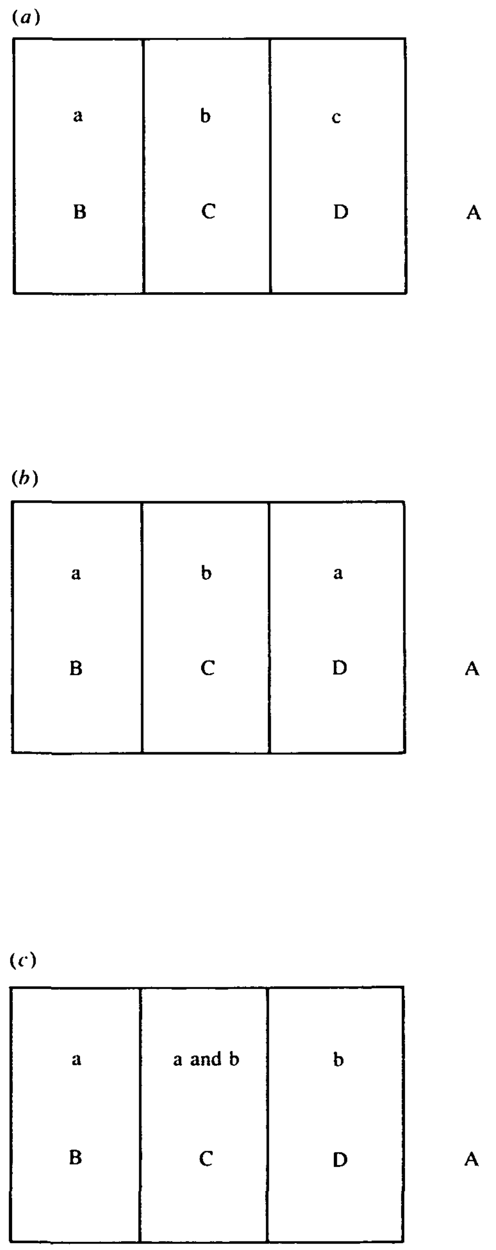


Figure 3. Relating areal objects to the primitive regions. *(a)* Simple areal objects have a one-to-one relationship to primitive regions. *(b)* Areal objects with disjoint parts, e.g. a, have a one-to-many relationship to primitive regions. *(c)* Overlapping areal objects, e.g. a and b, have a many-to-many relationship to primitive regions. Key: a–c, area objects; A–D, primitive regions.

DAM is not a universal model of topographic, let alone spatial, phenomena. It does, however, provide a framework for the flexible input of geographic areal information in a variety of formats. For example, attribute data about areal objects might be associated with the links, or with digitised points which lie within the objects, or both. Alternatively, the user could point to the primitive regions which belong to each areal object once the digitising of the links has been completed. In addition, information may be available from other sources, such as a table which defines how a single level of objects form a hierarchy. Since phenomena under consideration and the input format are both variable, further development of DAM requires an interface to a rule-processing facility. *Ad hoc* processes, based on a given rule-set, must otherwise be provided for integrating the geography with the geometry and for data validation and automatic editing.

DAM adopts a 'human' view of the geometry of areal entities in that, given the links, it is possible to extract a unique identity for each primitive region, to establish adjacency relationships and also to extract the information about the nesting of primitive regions. Since the geometric and geographic relationships are each processed separately and then related through the primitive region, DAM also provides a capability for cross-checking the geometric and geographic information.[17]

Once the object data has been assigned to the primitive regions, the boundaries of geographic areal objects can be computed. This is done by forming the union of the primitive regions belonging to each object and discarding any internal lines. Each geographic object comprises one or more disjoint parts, and each disjoint part can be described by one outer boundary and zero or more inner boundaries in a similar way to the geometric primitive regions. The boundaries are stored as an ordered list of links. Through the links, the adjacent primitive regions, and thus adjacent areas, to this area can be found. In addition, if a list of the constituent primitive regions is kept for each object and a list of objects is kept for each primitive region, spatial relationships between areas can easily be determined.

3. ALGORITHMS AND DATA STRUCTURE FOR DERIVING THE TOPOLOGY

This section describes algorithms for deriving the topology of a set of primitive regions from the geometry of a set of node-matched links. This operation has two

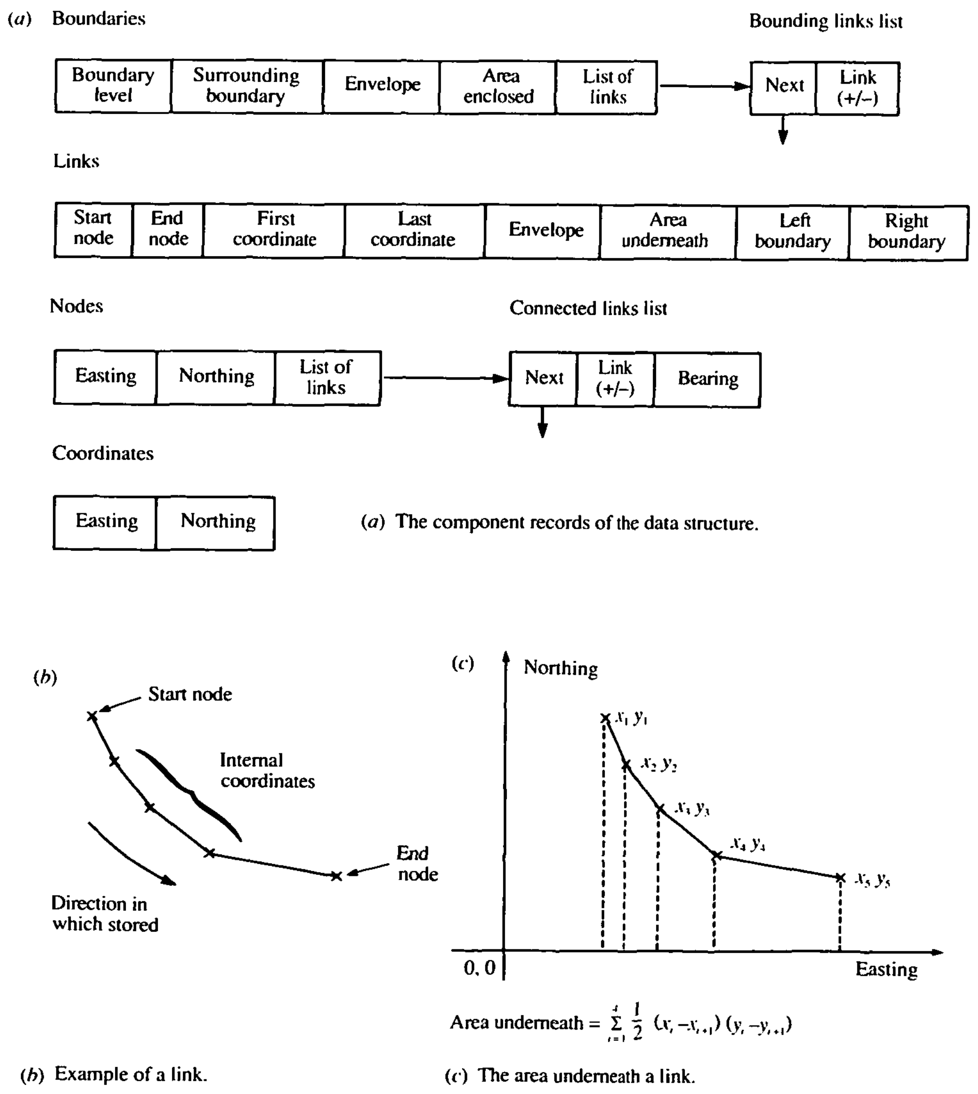


Figure 4. Data structure used for holding the geometry and topology of the linework of areal boundaries. *(a)* The component files. *(b)* Example of a link. *(c)* The area underneath a link.

distinct stages: first, a set of boundaries of primitive regions are extracted from the links; then the containment hierarchy between the boundaries is computed. The data structure used during these operations is also given.

3.1 Extracting the boundaries

This process assumes that the links have been topologically structured into a link and node structure; if necessary, the data must be pre-processed to convert it to this form.

The links are encoded using the link, node, and coordinate files represented in Fig. 4*a*. One link record is used for holding the details of each link (Fig. 4*b*). The coordinates of the two end points of the link are stored in node records, whose record numbers are kept in the link record. The internal points of the link (if any) are stored separately in a contiguous block of records in the coordinates file. The record numbers of the first and last coordinates are again kept in the links record. There need be no restriction on the number of internal points in a link, although it may be desirable to set an upper limit for implementation purposes, in which case long links can be split into two or more. The envelope (that is enclosing rectangle) and area underneath a link are calculated and held in the link record, as these values are each required several times later. The area is obtained by summing the area of the trapezium underneath each consecutive pair of points in the link using the standard formula – this yields a negative area for some links, which is used to advantage later. For example, this formula would yield a negative area for the link in Fig. 4*c*; however, if the link was stored in the opposite direction (that is $x_5 y_5$ before $x_4 y_4$ and so on) then the area obtained would have the same magnitude but be positive.

There is one node record for each location where one or more links start and end. With each node is kept a list of the record numbers of the links which meet there. The record number is signed to indicate whether the link starts (positive) or ends (negative) at the node. The bearing at which each link leaves or enters the node is also held, and the lists are sorted on this field before the boundaries are formed so that the links are referred to in a clockwise order around the node.

Once all the links have been added to a database, the process of boundary formation can take place. One boundary record is used to hold the details of each boundary. An ordered list of the links which describe each boundary is formed, and the envelope and area enclosed are calculated. The algorithm used to form each boundary works by initially selecting one link, and then taking a succession of leftmost links radiating from the previously selected link until the beginning of the first link is reached again. This is now the preferred method used by several systems, as it only uses the geometric information about the links. Many earlier systems used information about the areal objects on the left and right of the links when chaining the boundaries.

The method for forming the boundaries is given by the following algorithm. The names of variables are given in upper case, and references to the files have the form '*FILE_NAME [RECORD_NUMBER].FIELD_NAME*', for example '*LINKS [I].START_NODE*' refers to the value held in the 'start node' field of the *i*th 'links' record. Comments to explain the logic are enclosed by curly brackets.

Algorithm to form the set of boundaries of a set of links

```
let BOUND = 0 {a counter for the number of boundaries formed}
for each LINK
  if LINKS [LINK].LEFT_BOUNDARY is still unknown
  then {form the boundary on the left of this link}
    let BOUND = BOUND+1
    call FORM_BOUNDARY (BOUND, LINK)
  end if
  if LINKS [LINK].RIGHT_BOUNDARY is still unknown
  then {form the boundary on the right of this link}
    let BOUND = BOUND+1
    call FORM_BOUNDARY (BOUND, -LINK)
  end if
end for
{BOUND now holds the number of boundaries formed}
```

Procedure to form the boundary on one side of a specified link

```
procedure FORM_BOUNDARY (BOUND,
    START_LINK)
  let LINK = START_LINK
  let BOUNDARIES [BOUND].ENVELOPE
    = LINKS [abs (LINK)].ENVELOPE
  let BOUNDARIES
    [BOUND].AREA_ENCLOSED = 0.0
repeat {chain round the links which describe this boundary}
  if LINK > 0
  then {the boundary on the left of the link is being
        formed, so traverse the link in its stored direction,
        that is from its start node to its end node}
    let LINKS [LINK].LEFT_BOUNDARY = BOUND
    let BOUNDARIES [BOUND].AREA_ENCLOSED
      = BOUNDARIES[BOUND].AREA_ENCLOSED
        + LINK [LINK].AREA_UNDERNEATH
    let DEST_NODE = LINKS [LINK].END_NODE
  else {the boundary on the right of the link is being
        formed, so traverse the link in reverse direction, that
        is from its end node to its start node}
    let LINKS [-LINK].RIGHT_BOUNDARY =
      BOUND
    let BOUNDARIES
      [BOUND].AREA_ENCLOSED =
      BOUNDARIES [BOUND]
      AREA_ENCLOSED - LINKS
      [-LINK].AREA_UNDERNEATH
    let DEST_NODE =
      LINKS [-LINK].START_NODE
  end if
  let BOUNDARIES [BOUND].ENVELOPE =
    the maximum extents of the union of BOUNDARIES
    [BOUND].ENVELOPE and LINKS [abs (LINK)].
    ENVELOPE
  append LINK to the list of links of BOUND
  {find the entry in the list of links of DEST_NODE for
```

further immediate descendents of hole 1. Note that the algorithm identifies all immediate descendents of a hole, even if they do not share any common links with it (for example enclosing boundary 4 was correctly identified as following hole 1).

The same process would take place for holes 5 and 7, thus identifying the surrounding hole for enclosing boundaries 6 and 8 respectively. Hole 9 would also be found to surround enclosing boundaries 10 and 11.

The second part in the process is to find the immediate descendents of each enclosing boundary in the tree:

Algorithm to discover the holes which immediately follow each enclosing boundary

```
for each HOLE
  initialise an empty list for the possible surrounding
    boundaries of HOLE
  for each ENCLOSING_BOUNDary
    {perform an envelope and area test}
    if BOUNDARIES [HOLE]. ENVELOPE is properly
      contained in
        BOUNDARIES [ENCLOSING_BOUND].
          ENVELOPE and
      -(BOUNDARIES [HOLE].
        AREA_ENCLOSED) <
        BOUNDARIES
          [ENCLOSING_BOUND].
          AREA_ENCLOSED
    then
      add ENCLOSING_BOUND to the list of possible
        surrounding boundaries of HOLE
    end if
  end for
  let FOUND = false
  obtain the coordinates of an arbitrary point on the
    polygon of HOLE
  while the list is not empty and not FOUND
    remove the ENCLOSING_BOUNDary with the
    smallest area from the list
    if the point from the polygon of HOLE is enclosed
      by the polygon of
      ENCLOSING_BOUND {perform a point-in-
        polygon test}
    then {HOLE is an immediate descendant of
      ENCLOSING_BOUND in the tree}
      let FOUND = true
      let BOUNDARIES [HOLE].
      SURROUNDING_BOUNDARY =
      ENCLOSING_BOUND
    end if
  end while
  if not FOUND
  then {there is no boundary which surrounds HOLE, and
      thus the hole is at level 1 of the tree}
    let BOUNDARIES [HOLE].
      SURROUNDING_BOUNDARY = 0 {a rogue
      value}
    let BOUNDARIES [HOLE].
      BOUNDARY_LEVEL = 1
  end if
end for
```

Returning to Fig. 1*b*, no enclosing boundaries will be found which surround holes 1 and 5, so a surrounding boundary of 0 is recorded. In addition, their level will be recorded as 1. The envelope and area test will suggest that hole 7 immediately follows enclosing boundary 6 in the tree, and this will be confirmed by the point-in-polygon test. Similarly hole 9 will be connected to enclosing boundary 6.

The final part of the process is to determine the level of each boundary. This is straightforward – the holes at level 1 are known, and so the enclosing boundaries which are surrounded by these holes are found and recorded as being at level 2. The holes surrounded by these enclosing boundaries can then be found and recorded as being at level 3, and so on, and this continues down the hierarchy. This completes the formation of the tree.

4. AN APPLICATION

DAM has been applied to a subset covering four 100 km squares of the Ordnance Survey 1:625000 digital database of Great Britain.[15,16] The Ordnance Survey established a link and node structured database for the 1:625000 scale for experimentation with cartographic database design. The database does not provide an explicit description of the boundaries of areas, nor the hierarchical and spatial relationships between them. Instead it holds the essential topographic details as feature-coded links and areal seeds.

The OS database design considered the cost-effective capture of data as one of the most important requirements. Using DAM, it was established that this design was adequate in concept, structure and content for other purposes. For example, it proved possible to derive a complete and coherent description of a hierarchy of administrative districts, counties and countries from the fragmented and scattered features. The DAM model was then used to cross-check the consistency of geographic information, to establish whether data was missing and to infer these where possible. The capacity to detect a few isolated errors of various kinds was also demonstrated.

The subset of the 1:625000 database used in this application contained 558 links, 513 nodes and 9336 internal points of the links representing 68 districts and 8 counties in the two countries, England and Wales. Using the algorithms given in Section 3, coded in Fortran 77, it took 0.98 seconds to extract the 115 enclosing boundaries and holes and 2.88 seconds to form the tree of boundaries on an ICL 3980 computer running VME. These average times for executing the entire program include time for reading and writing as well as processing the data. The database occupies approximately 156K bytes in DAM format.

DAM is a conceptual model. The performance of various tasks may be optimised with suitable functional models. The application described above was more concerned with testing DAM than with performance.

5. CONCLUSIONS

DAM and the associated algorithms, described in this paper, were designed to automatically identify spatial units with holes and determine the topological relationships of connectivity, adjacency and nested containment of the set of primitive regions. The model and algorithms were developed to solve problems in digitial cartography,[14] but the decomposition of images and objects into

```
LINK entering the node (or leaving it if LINK is being
followed in reverse direction)}
  let POINTER = NODES [DEST_NODE].
    LIST_OF_LINKS
  while CONNECTED_LINKS [POINTER].
    LINK ≠ –LINK
    let POINTER = CONNECTED_LINKS
      [POINTER].NEXT
  end while
  {assign the next link in the list of links of DEST_NODE
  to LINK. This is the next link to leave or enter
  DEST_NODE when moving round the node in a
  clockwise direction, and will be the next link followed to
  continue the formation of the boundary}
  if CONNECTED_LINKS [POINTER].NEXT = nil
  then {the entry for –LINK is the last in the list of links
      of DEST_NODE, so loop round to the first entry
      in the list}
    let LINK = CONNECTED_LINKS
      [NODES [DEST_NODE].
      LIST_OF_LINKS].LINK
  else
      let LINK = CONNECTED_LINKS
      [CONNECTED_LINKS
      [POINTER].NEXT].LINK
    end if
  until LINK = START_LINK
  end procedure FORM_BOUNDARY
```

At the end of this process each link is part of two boundaries, where the boundary on its left has the same direction as the link, and the boundary on its right has the reverse direction. Note that the link is only stored once. As a result of the method used for chaining the boundaries, it is automatic that enclosing boundaries have an anti-clockwise direction and holes have a clockwise direction. A further consequence is that enclosing boundaries have a positive area and holes have a negative area. This is the area enclosed by the boundary, and not that of the corresponding primitive region. The boundary does not have a separate identity in the ARC/INFO and GIRAS data structures, and so they only record the area of the primitive region. In DAM, the area of a primitive region can be derived by summing the areas of all its boundaries, whereas the converse (obtaining the area enclosed by a boundary given the area of the primitive regions), it is not possible in ARC/INFO[5] and GIRAS.[9]

Each boundary has a separate existence within our data structure. The adjacencies between boundaries are given by the link records. However, where a primitive region contains holes, there is as yet no connection between the enclosing boundary of the primitive region and the boundaries which describe its holes. For example, there would be no connection between enclosing boundary 6 and holes 7 and 9 in Fig. 1*b*. The set of holes belonging to the outermost primitive region have also not been identified. These relationships need to be resolved, and an algorithm for computing this is described next.

3.2 Forming the containment hierarchy

As mentioned in Section 2.1, forming the containment hierarchy of a set of boundaries fully resolves the relationships between the boundaries, and thus the topology of the primitive regions. This hierarchy can be represented by a rooted tree, and the derivation of this tree for a set of boundaries is a one-off process. Edwards *et al.* (ref. 4, pp. 20–22) describe the algorithm used in their system; it utilises left/right references to the areal objects and lacks the efficiency of the method given below.

The following algorithm for forming the containment hierarchy between a set of boundaries does not require any geographical information, and attempts to minimise the computation required. During the process, the first two fields of the boundary records are completed (Fig. 4*a*); that is, the level of each boundary in the tree and also which boundary immediately precedes each boundary in the tree are ascertained. This information gives a complete description of the rooted tree.

The first part of this process is to find the immediate descendents of each hole in the tree:

Algorithm to discover the enclosing boundaries which immediately follow each hole

```
for each HOLE {that is, those boundaries with a negative
    area}
  push HOLE onto an empty stack
  repeat
    pop an item off the stack into BOUND
    {loop for each link in the list of links of BOUND}
    let POINTER = BOUNDARIES [BOUND].
      LIST_OF_LINKS
    while POINTER ≠ nil
      let LINK = BOUNDING_LINKS
      [POINTER].LINK
      if LINK > 0
      then {BOUND is on the left side of LINK}
        let ADJACENT_BOUND = LINKS
          [LINK].RIGHT_BOUNDARY
      else {BOUND is on the right side of LINK}
        let ADJACENT_BOUND = LINKS
          [–LINK].LEFT_BOUNDARY
      end if
      if ADJACENT_BOUND ≠ HOLE and
        BOUNDARIES [ADJACENT_BOUND].
          SURROUNDING_BOUNDARY is still
          unknown
      then
        push ADJACENT_BOUND onto the stack
        let BOUNDARIES [ADJACENT_BOUND].
          SURROUNDING_BOUNDARY = HOLE
      end if
      let POINTER = BOUNDING_LINKS
        [POINTER].NEXT
    end while
  until the stack is empty
end for
```

Applying this to Fig. 1*b*, when following the links of hole 1, enclosing boundaries 2 and 3 will be identified as being surrounded by hole 1, and they will be pushed into the stack. When following the links of the next boundary taken from the stack (either 2 or 3), enclosing boundary 4 will also be identified as being surrounded by hole 1, and it too will be pushed onto the stack. The stack will then be emptied by the next iterations as there are no regions or faces prior to modelling is a widely practiced technique.[8, 12] The ideas presented in this paper should be of relevance to a wide range of applications which need to automatically recognise and manipulate a hierarchy of polygons with holes.

Our approach does not require the introduction of contrivances, such as bridges, areal seeds and left/right codes, for recognising holes in polygons. However, subsequent representation of holes in polygons must be attuned to the requirements of existing algorithms and software. The data structures and formats used in different contexts may be viewed as pragmatic expressions of the abstract data model. The articulation of DAM contains explicit reference to boundaries to the left and right of each polyline. Once object-related information is connected to DAM through the primitive region, it is easy to extract either the polygonal outlines of hierarchic or overlapping areal entities or left/right object or attribute codes as required by some mapping packages. Also the sorted tree of boundaries, generated by the DAM-based approach, facilitates the automatic insertion of bridges should this be necessary for use with existing algorithms. The tree of boundaries also expedites the allocation of suitable segment priorities or the adoption of the painter's algorithm when using the *fill area* output primitive in GKS.

Acknowledgements

We are grateful to the SERC and the Market Analysis Division of CACI, London, the collaborating organisation, for a CASE studentship awarded to P. Wade. This work was also supported by Ordnance Survey and we are pleased to acknowledge the help provided by Mr P. Haywood and Mr N. S. Smith of the R&D Division. Our thanks are due to our colleagues Dr J. G. Griffiths and Mr D. J. Jordan for discussion on containment tests and terminology in areal topology, respectively, and to Mr M. S. Bundock of Cambridge Interactive Systems for his comments on an early draft of this paper.

REFERENCES

1. W. R. Allder and A. A. Elassal, USGS Digital Cartographic Data Standards: Digital Line Graphics from 1:24000 scale Maps. U.S. Geological Survey Circular 895-C (1984).
2. D. H. Ballard, Strip trees: a hierarchical representation for curves. *Communications of the ACM* **24**, 310–321 (1981).
3. D. F. Cooke and W. H. Maxfield, *The Development of a Geographic Base File and its Uses for Mapping*. Fifth Annual Conference of the Urban and Regional Information Systems Association, pp. 207–219 (1967).
4. R. G. Edwards, R. C. Durfee and P. R. Coleman, Definition of a hierarchical polygonal data structure and the associated conversion of a geographic base file from boundary segment format. Advanced Study Symposium on Topological Data Structures for Geographic Information Systems, Harvard University, Cambridge, Massachusetts (1977).
5. ESRI, *ARC/INFO Users Manual* – Version 3. Environmental Systems Research Institute, Redlands, California (1985).
6. R. G. Fegeas, R. W. Claire, S. C. Guptill, R. E. Anderson and C. A. Hallam, USGS Digital Cartographic Data Standards: Land Use and Land Cover Digital Data. U.S. Geological Survey Circular 895-E (1983).
7. R. C. Gonzalez and P. Wintz, *Digital Image Processing*. Addison-Wesley Publishing Company, Reading, Massachusetts (1977).
8. A. C. Kilgour, Techniques for modelling and displaying 3D scenes. In *Advances in Computer Graphics II*, edited R. J. Hubbold and D. A. Duce, pp. 55–113. Springer-Verlag, Berlin (1986).
9. W. B. Mitchell, S. C. Guptill, K. E. Anderson, R. C. Fegeas and C. A. Hallam, GIRAS: A Geographic Information Retrieval and Analysis System for Handling Land Use and Land Cover Data. U.S. Geological Survey Professional Paper 1059 (1977).
10. H. Moellering, A Working Bibliography for Digital Cartographic Data Standards, Issues in Digital Cartographic Data Standards, Report No. 4, National Committee for Digitial Cartographic Data Standards, The Ohio State University, Columbus, Ohio (1984).
11. T. K. Peucker and N. Chrisman, Cartographic data structures. *The American Cartographer* **2**, 55–69 (1975).
12. A. Rosenfeld, Image analysis: problems, progress and prospects. *Pattern Recognition* **17**, 3–12 (1984).
13. A. Rosenfeld and L. S. Davis, Image segmentation and image models. *Proceedings of the IEEE* **67**, 764–772 (1979).
14. M. Visvalingam, P. Wade and G. H. Kirby, Extraction of Area Topology from Line Geometry. *Proceedings Auto Carto London* **1**, 156–165 (1986).
15. M. Visvalingam, G. H. Kirby and P. Wade, Extraction of a complete description of hierarchically related area objects from feature-coded map details. SORSA '87 Durham. Ordnance Survey, Southampton (1987).
16. P. Wade, Derivation of hierarchic area objects from O.S. feature coded vectors. In *Research Based on Ordnance Survey Small-Scales Digital Data*, edited M. Visvalingam, pp. 61–70. Cartographic Information Systems Research Group, University of Hull (1986).
17. P. Wade, M. Visvalingam and G. H. Kirby, From Line Geometry to Area Topology, Discussion Paper 1. Cartographic Information Systems Research Group, University of Hull (1986).