AD-A071 124 ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE F/G 8/2
STRIP TREES: A HIERARCHICAL REPRESENTATION FOR MAP FEATURES.(U)
DEC 78 D H BALLARD N00014-78-C-0164
UNCLASSIFIED TR-32 NL

1 OF 1
AD A071124

END
DATE FILMED
8-79
DDC

(12)

AD A071124

DDC FILE COPY

LEVEL II

STRIP TREES:

A Hierarchical Representation for Map Features

Dana H. Ballard
Computer Science Department
University of Rochester

TR 32

December 1978

Department of Computer Science
University of Rochester
Rochester, New York 14627

STRIP TREES:

A Hierarchical Representation for Map Features

Dana H. Ballard
Computer Science Department
University of Rochester

TR 32

December 1978

ABSTRACT

There is increasing interest in map features such as points, lines and regions both as a pictoral data base for resource management and as an aid to identifying objects in aerial images. Owing to the very large amount of data involved, and the need to perform operations on this data efficiently, the representation of such features is a crucial issue. We describe a hierarchical representation of map features that consists of binary trees with a special datum at each node. This datum is called a strip and the tree that contains such data is called a strip tree. Lower levels in the tree corresponds to finer resolution representations of the map feature. The strip tree structure is a direct consequence of using the method for digitizing lines given by [Duda & Hart, 1973; Turner, 1974; Douglas & Peucker, 1973] and retaining all intermediate steps. This representation has several desirable properties. For features which are well-behaved, calculations such as point-membership and intersection can be resolved in O(logn) where n is the number of feature points. The map features can be efficiently encoded and displayed at various resolutions. All these properties depend on the hierarchical tree structure which allows primitive operations to be performed at the lowest possible resolution with great computational savings. The strip tree representation also can allow parts of the map feature to be accessed sequentially. This feature is usually desired when the map feature is used in analyzing images.

The price paid for the improved performance is an increased storage cost. This is approximately 4n, where n is the storage needed to represent the xy coordinates.

The research described in this report was supported partially by DARPA Grant #N00014-78-C-0164 and partially by NIH Grant #R23-HL-21253-01.

DISCLAIMER NOTICE

THIS DOCUMENT IS BEST QUALITY PRACTICABLE. THE COPY FURNISHED TO DDC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: TR32 (6)
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:
4. TITLE (and Subtitle): Strip Trees: A Hierarchical Representation for Map Features.
5. TYPE OF REPORT & PERIOD COVERED: (9) Technical Report.
6. PERFORMING ORG. REPORT NUMBER:
7. AUTHOR(s): (P) Dana H. Ballard
8. CONTRACT OR GRANT NUMBER(s): (15) N00014-78-C-0164
9. PERFORMING ORGANIZATION NAME AND ADDRESS:
   Computer Science Department
   University of Rochester
   Rochester, N. Y. 14627
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: (12) 44p.
11. CONTROLLING OFFICE NAME AND ADDRESS:
    Defense Advanced Research Projects Agency (11)
    1400 Wilson Boulevard
    Arlington, VA 22209
12. REPORT DATE: December 1978
13. NUMBER OF PAGES: 37
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office):
    Office of Naval Research
    Information Systems
    Arlington, Va. 22217 (14) TR-32
15. SECURITY CLASS. (of this report): unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:
16. DISTRIBUTION STATEMENT (of this Report):

This document has been approved for public release and sale; its distribution is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):
18. SUPPLEMENTARY NOTES:
19. KEY WORDS (Continue on reverse side if necessary and identify by block number):

data bases
trees
aerial images
hierarchy maps
computational complexity

20. ABSTRACT (Continue on reverse side if necessary and identify by block number):

ABSTRACT IS LISTED ON REVERSE SIDE.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

410 386 LB

1. Introduction

We present a general representation for polylines (connected line segments) and areas (closed polylines). Although this representation may have wide applications, its principal motivation arose from the problem of representing geographical data bases of map features.

A map has several interesting kinds of features such as contour lines, lakes, rivers, roads, etc. These can be roughly divided into four feature classes for representation in the computer [Sloan, 1978]:

feature	examples in map domain
points	towns (large scale maps) bridges (small scale maps)
lines	roads, coastlines
strips	wide roads, rivers
regions	lakes, counties

Our main interest is in representing lines and regions. A point is such a simple datum that it can be easily treated as a primitive in any representation. Collections of points from a single class can be efficiently represented as k-d trees [Bently, 1975; Barrow et.al., 1977] and so points are not the focus of our interest, although they do interact with our representation. A strip feature is essentially a line where a locally varying thickness is important, examples of which are rivers and roads. As we shall see, our representation for lines will also encompass this type of feature.

We regard collections of these map features as a data base that might be used to perform the following tasks:

.Find where a road intersects a river

.Display a subset of map features that appear in a given map sector

.Find out if a given point is in a region

.Search an aerial image near the edge of a dock for ships.

A very important aspect of all these tasks is that we may be satisfied if they are performed at resolution lower than the ultimate resolution represented.

Our representation for lines and regions consists of a binary tree structure where, in general, lower levels in the tree correspond to finer resolutions. The tree structure is a direct consequence of using the method for digitizing lines given by [Duda and Hart, 1973; Turner, 1974] and retaining all intermediate steps in the digitization process. As an example of the representation, Figure 1 shows some roads represented at various levels (resolutions) in the tree structure.

The idea of representing a line by sets of strips was recognized by [Peucker, 1976]. In particular he was able to find line intersection and point in polygon algorithms. However, the tree structure is a vast improvement over the set organization: the algorithms are more efficient, line-area intersection and area-area intersection and union can now be dealth with, and the tree structures are closed under these operations.

Figure 1. Map features displayed at various resolutions using the hierarchical structure.

2. The Strip Tree

2.1 Notation

We define a strip segment $\underline{L}$(delta) as the vector $\underline{L}$ and the scalar delta as shown by Figure 2. The vector $\underline{L}$ starts at (XBeg,YBeg) and ends at (XEnd,YEnd). We use S to denote the set of points inscribed by the rectangle defined by $\underline{L}$(delta). Also we denote the boundaries of the rectangle by the line segments l+, l-, e+, e- as shown.

Figure 2. Definition of a Strip Segment.

A polyline is an ordered list of discrete points y0,...,yn subsets of which may be colinear. For the moment we require these points to be considered as connected; later we will relax this condition. We say a polyline is <u>represented at resolution delta*</u> if there exists an ordered sequence of m strip segments

$$\underline{L}_k(\text{delta}),\ k=0,\ \ldots,\ m-1$$

such that

$$\text{delta}_k < \text{delta}^*\ k=0,\ \ldots,\ m$$

$$y_i \in \bigcup_{k=0}^{m} \underline{L}_k\ i=1,\ \ldots,\ n$$

If within a strip segment there is a point y that is a member of e+, another that is a member of e-, and there is a point v that

is a member of 1+ and another that is a member of 1-, then the strip segment is said to be compact. The compactness property is very important for some of the algorithms which follow. Figure 1 shows some examples for different deltas.

2.2. Digitization

Suppose we have a polyline P. such as shown by Figure 3a. For any resolution delta we can approximate this line with strip segments as follows [Duda & Hart, 1973; Turner, 1974]:

> Consider the polyline P defined by $\{y0, yn\}$ For each point $y \in P$ find the perpendicular distance d(y) from y to P. Denote the subset of y P such that $\underline{y}.\underline{I} \geq 0$ as P+. P-= P-P+. Now find $d+ = \max_{y \in P+} d(y)$ and $d- = \max_{y \in P-} d(y)$. If (d+) + (d-) <delta* then the polyline is compactly represented at resolution $\overset{delta*}{\wedge}$ by the strip tree consisting of a single root strip 1((d+)+(d-)). If not then the desired strip tree is obtained by recursively applying the algorithm to the Ps $\{y0,\ldots,y+\}$ and $\{(y+)+1,\ldots,yn\}$ and making the results the left son and right son respectively of the strip tree. In the case of ties for the maximum distance d, we will arbitrarily pick the point nearest the mid point (in arc length).

For the purposes of the union and intersection algorithms to follow it is helpful to think of the strip trees as completely expanded down to individual points, even though these points may be colinear. Figure 3 shows an example of two levels of

recursion of this algorithm.

Figure 3. Steps in the Digitization Process.

To see formally that the convergence is guaranteed, note that a P of k points can always be approximated by a single strip segment L(k) with length k assuming eight-connectedness. Thus for any delta there must be a strip tree with leaves consisting of no more than n/delta strip segments which approximate P. Since the digitization algorithm splits each P into two parts such that each part has finite length, the process must ultimately consider sets of P of delta points or less.

2.3 Strip Tree definitions

The binary tree resulting from the digitization process is called a strip tree, where the datum at each node is a strip, I. The nodes of the tree are initially ordered on arc length. (Later we will see that when intersection occurs in two areas which are represented in strip trees, this property is sometimes not preserved).

In the ensuing algorithms we will use the following definitions:

T ≡ symbol for a Strip Tree obtained by the digitization process.

S(T) ≡ the points associated with the strip at the root node of T; i.e. $\{x \mid x \in S(T)\}$

Area(T) ≡ the area associated with the strip at the root node of T. We measure area in pixels so that a strip $\underline{L}(0)$ still has finite area. The most primitive strip, a single point, has unit area.

LSon(T) = the left son of the node T

RSon(T) = the right son of the node T

A node of the strip tree is completely defined by the seven-tuple (LSon, RSon, Area, XBeg, XEnd, YBeg, YEnd). The measure Area(T) is better for some of the algorithms to follow. Area and delta are related by delta ≈ Area/||L||.

2.4. Why Binary Trees?

The polylines can also be represented as a tree with nodes of more than two siblings. In fact, nodes could have different numbers of siblings which would still be ordered. Figure 4 shows an example of the alternate encoding scheme. In certain cases this may be a more concise representation for the polyline and for all the algorithms that follow we can extend the operations from two sons to multiple sons. However, this change does not alter the complexity of the operations that we would like to perform and can be more inefficient than the binary tree representation.

Figure 4: A portion of an encoding using m-ary trees.

3. Operations on Polylines

Computational complexity of the various operations is difficult to characterize, as it depends on the particular geometry of polylines. If the polylines are "well-behaved", that is they are relatively smooth and do not self-intersect for more than a few points, then the algorithms are very efficient. What this means for a particular operation in terms of the strip tree is that if the number of strips that must be examined at any level is constant, then the complexity of the operation is O(log n).

3.1. Testing the Proximity of a Point

If we would like to find out if a point is near a polyline, this may be discovered early using the strip tree. We can make this more precise by exploiting the following property:

Property P1:

A. If a point z is inside a compact strip 1(delta) then it can be at most 2^{delta} units away from the P.

B. If a point z is outside a compact strip 1(delta) then the distance of the point from the P is bounded by

$$0 \leq z \leq d_S(z, L(\text{delta})) + \text{delta}$$

It is interesting to study these bounds as the depth in the resolution tree increases. Although the convergence is not monotonic, the bounds do converge to the actual set-theoretic distance $d_S(z,P)$. Now suppose we want to answer the question: is $d_S(z,P) < d_0$? If this can be answered affirmatively we will find this out at the point where <u>any</u> upper bound is less than d_0. If the answer is no, then this will be discovered when the tree has been explored to the point where all minimum bounds are greater than d_0. Similar arguments can be made for the qualitative level-of-effort required to answer: is $d_S(z,P) > d_0$? From this discussion we can see that the search will be inefficient only if $d_0 \approx d_S(z,S(T))$ and a large number of the strips are nearly d_0 from z. Figure 5a shows this case together with a more representative example.

Figure 5. Two of many Possible Geometrics When Testing the Distance of a Point from an P.

To summarize this discussion, we provide the algorithms to test for $d_S(z,P) < d_0$ and $d_S(z,P) > d_0$. These algorithms use the notion of the distance of a point to a set which is defined as follows. For any strip S, if a point is outside S i.e. $x \notin S$ then its distance to S is characterized by the set theoretic

distance $d_S(z,S) = \min_{x \in S} d(x,z)$ where d is the euclidean distance between the points x and z. For clarity, the algorithms are presented as procedures in a pseudo-Algol language. Rigor has been sacrificed mainly in the specification of data types, but these should be obvious from the earlier definitions.

Algorithm A1: Is a point within d0 of a polyline?

```
boolean procedure Within (z,d0,T)
    begin
    if d0≤ ds(z,S(T)) + 2.delta(T) then return (true);
    if z / S(T) and d0>ds(z,S(T)) then return (false);
    return (Within (z,d0,LSon(T)) or Within (z,d0,RSon(T)));
    end;
```

Algorithm A2: Is a point further than d0 from a polyline?

```
boolean procedure Further (z,d0,T)
    begin
    if d0≤ ds(z,S(T)) + 2.delta(T) then return (false);
    if z ∉ S(T) and d0>ds(z,S(T)) then return (true);
    return (Further (z,d),LSon(T)) and Further (z,d0,RSon(T)));
    end;
```

3.2 Displaying a Polyline at Different Resolutions

As previously demonstrated in Section 2, a polyline may be represented as a set of strip segments such that each strip segment L has a resolution delta less than some fixed delta0. The algorithm to display such a representation using the strip tree is as follows. This algorithm uses a device-dependent

subroutine DisplayRectangle which paints the rectangle on the particular display device.

Algorithm A3: Display a polyline at Resolution delta0

```
procedure PolyDisplay (T,delta0)
    begin
    if delta(T) ≤ delta0 then DisplayRectangle (I(T),delta(T))
       else (PolyDisplay (LSon(T),delta0) and PolyDisplay
    (RSon(T),delta0));
    end;
```

3.3 Intersecting Two Polylines

One of the important features of the representation is the ability to compute intersections between polylines. Strip trees provide the facility to not only compute intersection points, but, in the case where lower resolution is satisfactory, to compute small areas containing the intersection points at great computational savings. In order to develop the intersection methodology, we need the following definitions:

A. Two strip segments (L1 derived from P1) and (L2 derived from P2) do not intersect iff $L1 \cap L2 = \emptyset$

B. Two strip segments L1, L2 have a clear intersection iff: L1+ and L1- intersect L2+ and L2-.

C. Two strip segments L1 and L2 have a possible intersection if condition B is not satisfied yet $L1 \cap L2 \neq \emptyset$.

These cases are illustrated by Figure 6. A fairly obvious but very important lemma is:

Clear Intersection Lemma. [Doucker, 1976] If two strip segments have a clear intersection and the strips are both compact, then the corresponding Ps must also intersect.

To see this for condition B, consult Figure 6b. P1 divides the region R into two parts and P2 must cross from one to the other. The only way the P2 can do this is by intersecting P1.

Figure 6: Different Ways Strips can Intersect

The algorithms to check for intersections between two polylines are recursive, and assume the existence of an integer procedure StripIntersection which will return the type of intersection and, in the case of a clear type, will return a parallelogram Q containing the intersection points.

Algorithm A4: Finding out whether two polylines

```
intersect

Comment. If the two root strip segments do not
intersect then the Ps do not intersect. If the
root segments have a clear intersection then the
Ps intersect. Since the task is to just determine
whether or not an intersection exists, we are done
the moment we find a clear intersection.

boolean procedure Intersection (I1,I2, Primitive
Flag)
comment Primitive Flag allows the use of a single
   strip as the first argument
begin
Case StripIntersection (S(T1),S(T2),Q) into
   [Null] return (false),
   [Possible] if (Area(T1)>Area(T2)) or (Primitive
Flag) then
      return ((Intersection(LSon(T1),T2) or
(Intersection(RSon(T1),T2));
      else return
   (Intersection(I1,LSon(T2)) or
Intersection(T1,RSon(T2)));
   [Clear] return(true);
end;
```

This procedure is easily modified to return a set of parallelograms comprising intersection points. Further easy

modifications can be made to constrain these parallelograms to be of a certain size related to the delta(T1) and delta(T2); i.e., they can be made to be as small as we want.

Note, however, that smaller resolutions may be much more computationally expensive, as shown in the following example (Figure 7) where intersection at the coarsest resolution is simple, but multiple intersections occur at lower levels.

Figure 7: An intersection may be simple at one level and complicated at lower levels.

If the two Ps are not convoluted about each other the intersection will be computed in 0(mlog(n)) steps where m is the number of intersection points. If the Ps do not intersect but have a closest distance d =ds(P1,P2) then this will be discovered at a level in the tree no deeper than a point where $d/_2 <$ delta1 + delta2.

The worst case performance is intolerable as the algorithm's computation will grow exponentially as long as all the strip segments in one tree intersect all the strip segments in the other. In fact, the computation can be shown to be $0(2^K)$ where K is the sum of the depths in each tree where the comparisons are taking place! If this situation were encountered in a practical application, one way of handling it would be to report the possible intersection regions at the point where the limit of

some bound on allotted resources was exceeded.

3.4 The Union of Two Polylines

The union of two strip trees can be accomplished by defining a strip that covers both of the two root strips.

Algorithm A5: P-P Union.

> For two Ps defined by $\{y_0' \ldots y_n'\}$, $\{y_0'' \ldots y_m''\}$ treat these as two subsets and concatenate the subsets. That is, the resultant ordering is such that we have $y_0 = y_0'$, $y_{m+n+1} = y_m''$. Now define a strip segment that covers $\{y_0, \ldots, y_{m+n+1}\}$ such that c=0 and delta =d*. By construction, this satisfies all the properties of a strip segment. Make this the root node of a new P-tree. The two subtrees are the two Ps of the union.

This construction is shown in Figure 8. The variable c is defined below.

Figure 8: Construction for Union of Strip Trees Representing Two Polylines

Of course this construction introduces a problem in that the new strip is no longer compact and therefore the Clear Intersection Iemma no longer holds. To overcome this problem we must add one bit of information to each node to mark whether the underlying polyline is compact. Since later algorithms may result in underlying polylines that are disconnected, we include this in the following definition of C:

$$C(T) = \begin{cases} 1 & P \text{ represented by } S(T) \text{ is known to be compact and connected} \\ 0 & \text{otherwise} \end{cases}$$

With this strategy we can preserve the eloquence of the previous algorithms in the following manner. When bit C(T) is not one we apply the recursion regardless of the intersection type. In algorithm 14 this means that clear intersections are reported as possible if the bit C(T) is set.

This technique can also be used as a digitization method for m non-connected segments $\{(y_0,\cdots,y_i),(y_{i+1},\cdots,y_j),\cdots,(y_k,\cdots,y_n)\}$. These segments are given an ordering as shown. The previous digitization algorithm is applied to this set of points, and the perpendicular distance d* is computed from the set of disconnected vs and used to define the of the root strip as before. However now the set is divided into two subsets of connected segments (rather than using v*) and the digitization algorithm is applied recursively to the subsets. Once this process produces connected subsets, the earlier digitization

scheme is applied.

4. Areas Represented by Strip Trees

We take the boundary of an area to be a closed polyline. Interestingly enough, the digitization method described in Section 2 works for closed polylines and, incidentally, also for self-intersecting polylines. Furthermore, if an area is not simply connected it can still be represented as a strip tree, which at some level has connected primitives. The method for doing so was described in the previous section. If a region has holes it can be represented by a single boundary curve using a construction (Figure 9).

Figure 9: A Region with a Hole

If the holes are important, they themselves should be independently represented as strip trees.

The most remarkable fact is that by representing an area in this way many useful operations such as intersection between a polyline and an area, determining whether a point is inside an area, and intersecting two areas are carried out very

efficiently.

4.1 Determining Whether a Point is Inside an Area

The strip tree representation of an area by its boundary allows the determination of whether a point is inside the area in a straightforward manner. If any semi-infinite line terminating at the point intersects the boundary of the area an odd number of times, the point is inside. This result appears in [Minsky and Papert, 1969]. This result is computationally simplified for strip trees in the following manner:

> Point Membership Property
>
> To decide whether a point z is member of an area represented by a strip tree, we need only compute the number of <u>clear intersections</u> of the strip tree with any semi-infinite strip I which has delta = 0 and emanates from z. If this number is odd then the point is inside the area.

An extension to the clear intersection lemma which makes this property hold is that the underlying curves may intersect more than once but must intersect an odd number of times. The following algorithm is used to determine whether a point is inside an area:

Algorithm A6: Point Membership

```
boolean procedure Inside(z,T)
   begin
   CreateStrip(S0,z)
   comment CreateStrip creates a strip for the half line.
   if NoOfClearIntersections(S0,T) is odd then return (true)
      else return (false);
   end;
integer procedure NoOfClearIntersections(S,T)
   begin
   CaseStripIntersection(S,S(T)) into
      [Null] return (0);
      [Possible] return (NoOfClearIntersections(S,LSon(T))
            + NoOfClearIntersections (S,RSon(T)));
      [Clear] return (1);
   end;
```

A potential difficulty exists with the procedure NoOfClearIntersections when the strip S0 is tangent to the polyline. Since this problem will only occur at the lowest level of the tree, we can examine neighboring leaves of the tree to resolve it.

4.2 Intersecting a polyline with an Area

The strategy behind intersecting a strip tree representing a polyline with a strip tree representing an area is to create a new tree for the portion of the polyline which overlaps the area. This can be done by trimming the original polyline strip tree.

This is done efficiently by taking advantage of an obvious property of the intersection process:

> Pruning Property: Consider two strips Sp e Tp and Sa e Ta. If the Sp$\cap$Ta is null, then (a) if any point on Sp is inside Ta the entire tree whose root strip is Sp is inside or on Ta and (b) if any point on Sp is outside of Ta then the entire tree whose root strip is Sp is outside of Ta.

This leads to the recursive procedure \7 for polyline-area intersection using trees. Note that since strip nodes under a clear or possible strip intersection may be pruned, the bit c for the latter strip is set to 0 to denote that it no longer has the compactness property. Of course as repeated intersections are carried out with different areas more and more upper-level strips may have their bits set to 0; nevertheless, the intersected polyline is accurately represented at the leaves of the strip tree.

Note that if the polyline strip is "fatter," i.e., Area(T1) > Area(T2), we can copy the node and resolve the intersection at lower levels, whereas in the converse case we have to sequentially prune the tree by first intersecting the polyline strip with the left area strip and then intersecting the resultant pruned tree with the right area strip.

```
Algorithm A7:  Polyline-Area Intersection
reference procedure PolyAreaInt(T1,T2)
  begin
  A:=T2
  comment A is a global used by PAInt;
  return(PAInt(T1,T2);
  end;

reference procedure PAInt(T1,T2)
  begin
  Case StripInt(T1,T2) into
     [Null or Primitive]
          if Intersection (T1,A, TRUE) = null then
                if Inside(T1,A) then return (T1)
                else return (null);
          else return (T1);
     [Clear or Possible] if Area(T1)>Area(T2) then
        begin
        C(NT):=0
        comment non-compact strip
        XBeg(NT):= XBeg(T1);
        YBeg(NT):= YBeg(T1);
        XEnd(NT):= XEnd(T1);
        YEnd(NT):= YEnd(T1);
        Area(NT):= Area(T1);
        LSon(NT):= PAInt (LSon(T1),T2);
        RSon(NT):= PAInt (RSon(T1),T2);
        return(NT);
        end
```

```
    else comment Area(T1) ≤ Area(T2)
        Return (PAInt(PAInt(T1,LSon(T2)),RSon(T2)));
end;
```

4.3 Intersecting Two Areas

The problem of intersecting two areas can be efficiently carried out using their strip tree representations. The method is to decompose the problem into two polyline area intersection problems (refer to Figure 10).

Figure 10: Decomposition of Area-Area Intersections

If we treat the boundary of A1 as representing a polyline instead of representing an area and intersect its strip tree with the strip tree representing A2 the lowest level result is shown by the thick lines in Figure 10a. If we reverse the roles of the two strip trees the result is given by the thick lines in Figure 10b. The union of these two strip trees (see Section 3.4) is the answer we want! Thus we can write the area-area intersection procedure in terms of strips as follows:

Algorithm A2: Area-Area Intersection

```
reference procedure AreaAreaInt (T1,T2)
```

```
begin
return (Union (PolyAreaInt (T1,T2)),(PolyAreaInt
(T2,T1)));
end;
```

where Union is a procedure that accomplishes the construction described in Section 3.4.

Note that in the case of areas that intersect in a way that fragments their boundaries, the order of the segments will not be preserved by the intersection procedure. (Until this point we were guaranteed that strips in the tree would be ordered according to the arc length of their underlying polylines). However, all the other properties of the representation are preserved.

4.4 The Union Operation

The union operations are slightly simpler than the intersection operation. For the union of a P-tree and a P-tree we use a construction similar to the digitization methods for disconnected Ps. The result is a P-tree. Note that the union operation for strip trees is not commutative. Also, we do not define a union operation for a strip tree representing a polyline and a strip tree representing a region. The union of two region strip trees is defined and is a region strip tree. If these two strip trees do not intersect, then the union is straightforward and is identical to the method for polylines. However, if the contrary is true, then we must go to the trouble of defining a

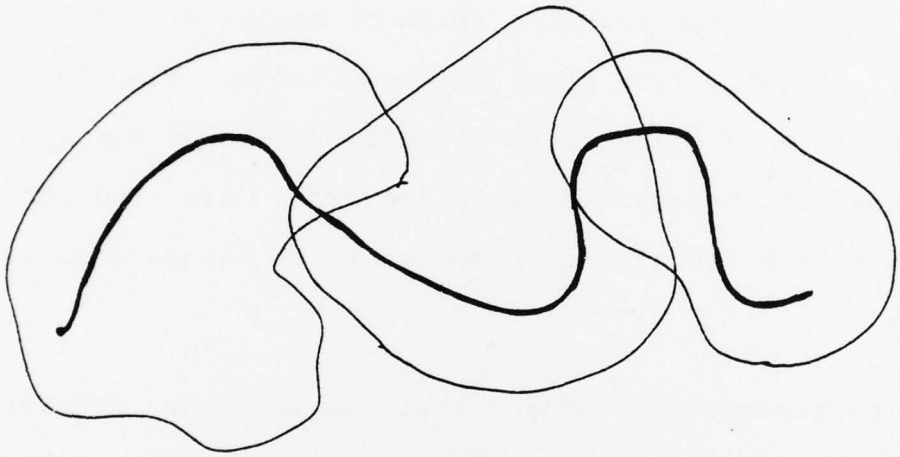
new strip tree that represents the union by finding the points of intersection in the same way as was done for region strip tree intersections.

5. Conclusions

Strip trees provide a powerful representation for polylines and areas. Current work is directed towards characterizing their computational complexity more precisely but it can already be shown that the representation is superior to its competitors. The main drawback is that there is a large overhead in terms of space. If n is the required space to represent a polyline then its strip tree will take about 4n space units. Also the creation of a strip tree is a laboricus process, requiring O(n log n) time units. However, neither of these drawbacks are thought to be important in the use of this representation for geographical data bases.

The representation defines strip segments as primitives to cover subsets of the line after [Peucker, 1976]. Our organization of these segments into a tree may be viewed as a particular case of a general strategy of dividing features up and covering them with arbitrary shapes such as depicted by Figure 10. Other attempts in this class have been tried by [Barrow et al.,1977; Burton, 1977; Tanimoto, 1975], but they do not capture the notions of orientation and resolution anywhere nearly as precisely as strip segments, and do not have the union and intersection properties.

Figure 11: The Notion of an Arbitrary Divide-And-Conquer Strategy



Acknowledgements

The author wishes to thank F. Peet and P. Meeker for their work in the preparation of this document. Thanks also go to D. Weaver for his work in implementing the algorithms in SAIL.

References

Barrow, H.G., "Interactive Aids for Cartography and Photo Interpretation" Semiannual Technical Report 12May 1977-11Nov.1977, ARPA contract DAAG 29-76-C-0057, SRI International.

Bently, J.L., "Multidimensional Search Trees Used for Associative Searching" CACM Vol. 18, No. 9, September, 1975.

Burton, W., "Representation of Many-Sided Polygons and Polygonal Lines for Rapid Processing" CACM Vol. 20, No. 3, March, 1977.

Douglas, D.H. and Peucker, T., "Algorithms for the Reduction of the Number of Points Required to Represent a Line or its Caricature," The Canadian Cartographer, Vol 10, No. 2, December, 1973.

Duda, R.O. and P.E. Hart, Pattern Classification and Scene Analysis, Wiley-Interscience 1973.

Minsky, M.L. and S. Papert, Perceptions:an introduction to computatonal geometry, MIT Press, Cambridge, Mass., 1969.

Peucker, T., "A Theory of the Cartographic Line,"

International Yearbook of Cartography, 16, 1976.

Sloan, K.R., "Maps and Map Data Structures," forthcoming Technical Report, Computer Science Department, University of Rochester.

Tanimoto, S., and Pavlidis, T., "A Hierarchical Data Structure for Picture Processing" Comp. Graphics and Image Processing, Vol. 4, No. 2, June, 1975.

Turner, K.J., "Computer perception of curved objects using a television camera", Ph.D. thesis, University of Edinburgh, 1974.

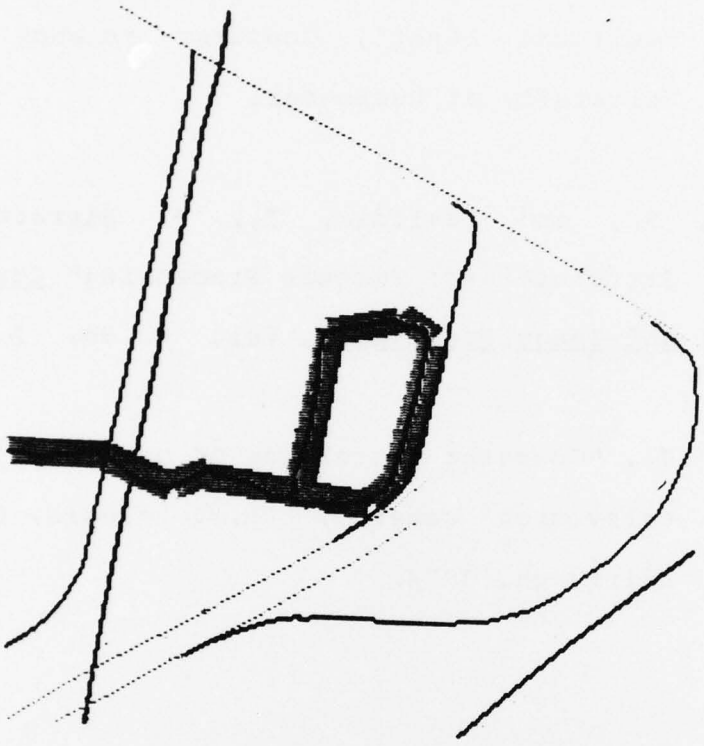


Figure 1. Map features displayed at various resolutions using the hierarchical structure.

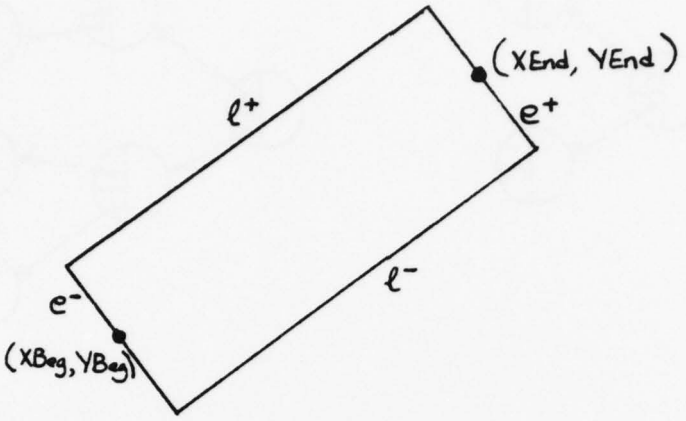


Figure 2: Definition of a Strip Segment.

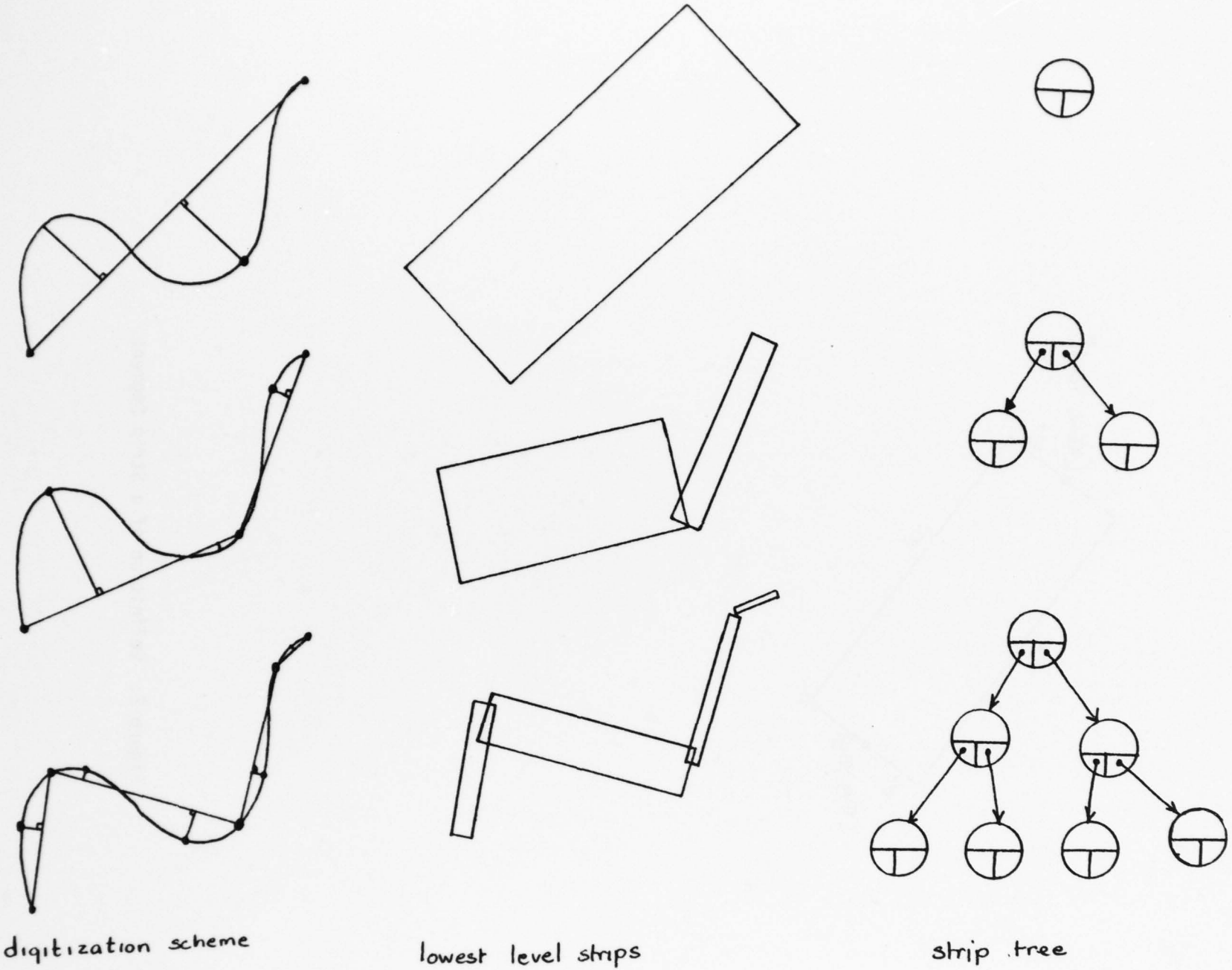


Figure 3: Steps in the Digitization Process.

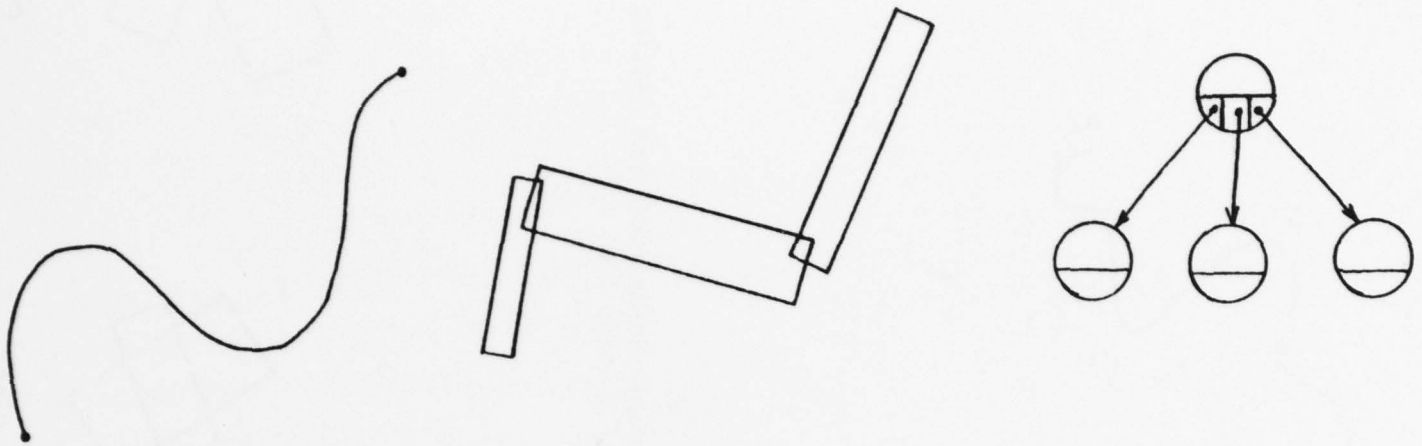


Figure 4. A portion of an encoding using m-ary trees.

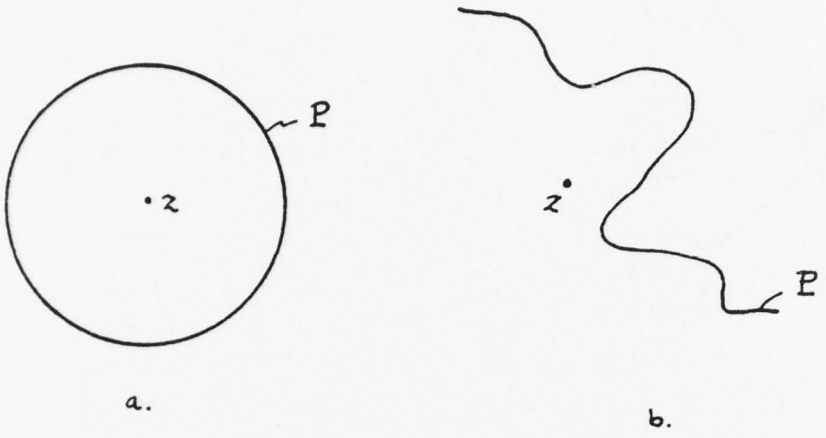


Figure 5: Two of Many Possible Geometrics When Testing the Distance of a Point from a P.

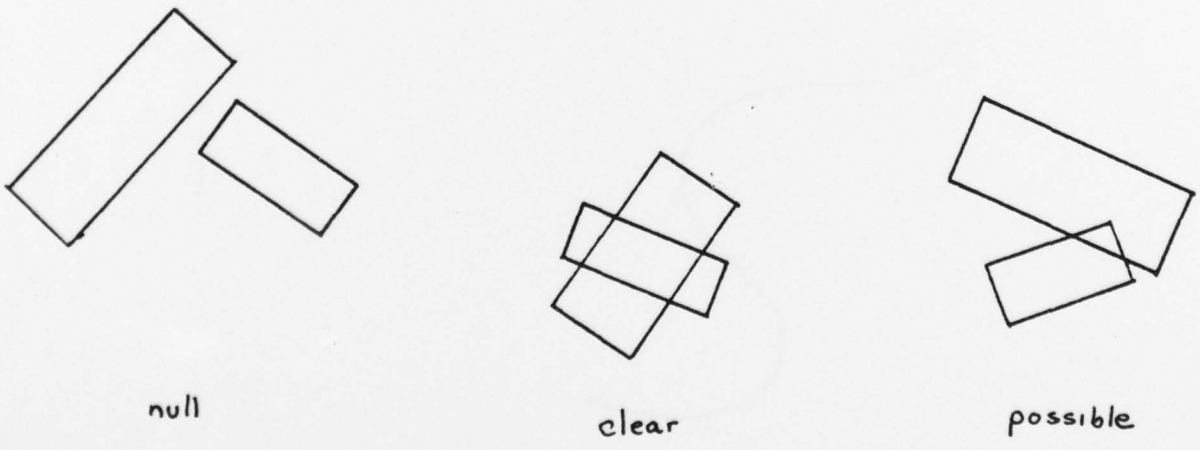


Figure 6: Different Ways Strips Can Intersect

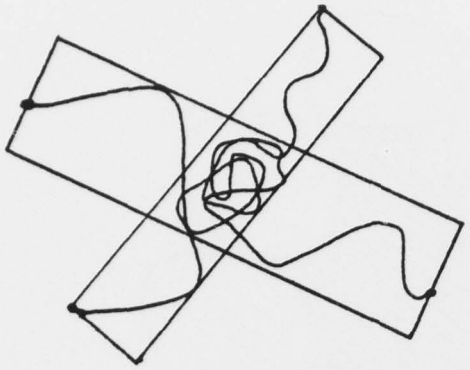


Figure 7. An intersection may be simple at one level and complicated at lower levels.

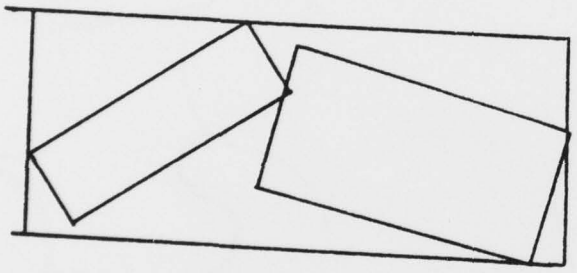


Figure 8: Construction for Union of Strip Trees Representing Two Polylines

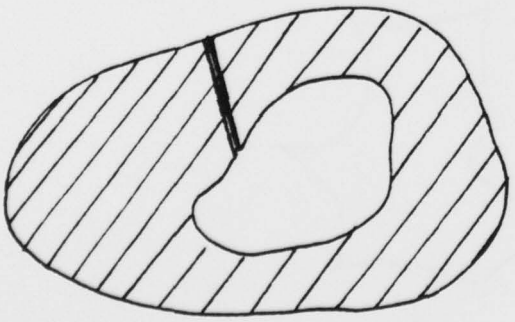
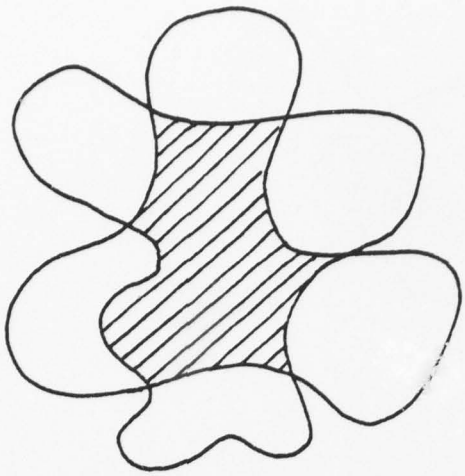
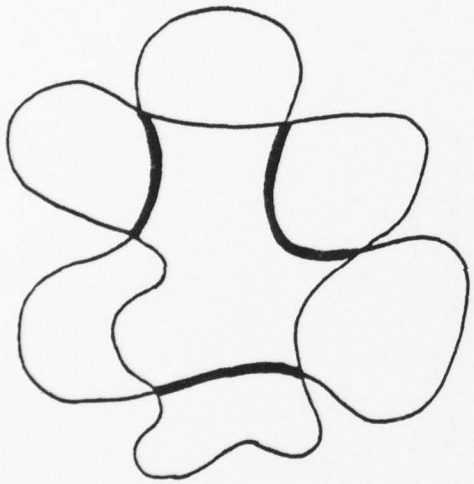


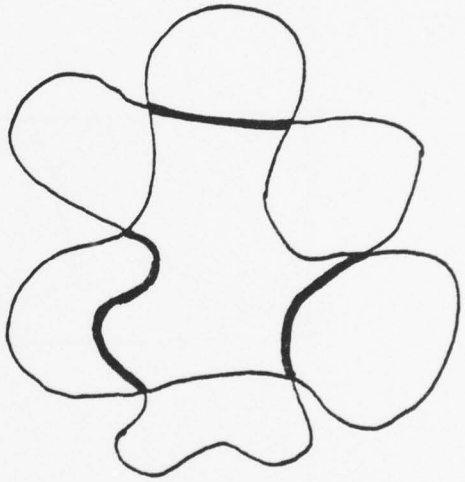
Figure 9: A Region with a Hole



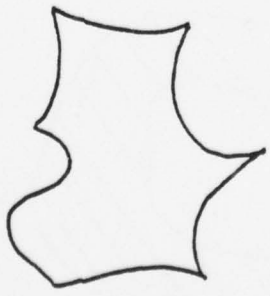
a.



b.



c.



d.

Figure 10: Decomposition of Area-Area Intersections

a. Desired Result $A_1 \cap A_2$

b. Result of $T_{L_1} \cap T_{A_2}$

c. Result of $T_{L_2} \cap T_{A_1}$

d. Union of Two Results: the polyline segments covered by the result tree.