STRIP TREES:
A Hierarchical Representation for Curves

Dana H. Ballard
Computer Science Department
University of Rochester

TR 32
December 1978
revised
November 1979

ABSTRACT

The use of curves as a representation of two dimensional structures is an important part of many scientific fields. For example, geographers make extensive use of curves as a representation for map features such as contour lines, roads and rivers; circuit layout designers use curves to specify the wiring between circuits. Owing to the very large amount of data involved, and the need to perform operations on this data efficiently, the representation of such curves is a crucial issue. We describe a hierarchical representation 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 correspond to finer resolution representations of the curve. 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 $O(\log n)$ where n is the number of feature points. The curves can be efficiently coded and displayed at various resolutions. The representation is closed under intersection and union and these operations can be carried out at different 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 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.

1. Introduction

We present a general representation for planar curves. This representation allows operations such as union and intersection to be performed efficiently and is thus of great interest to fields using data bases of such curves. Two such fields are geography and computer-aided circuit design.

Consider the application to geography. A map has several interesting kinds of features such as contour lines, lakes, rivers, and roads. These can be roughly divided into four feature classes for representation in the computer.

feature	examples in map domain
points	towns (large scale maps) bridges (small scale maps)
lines	roads, coastlines
strips	wide roads, rivers
areas	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 curves 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, Fig. 1 shows a curve represented at two levels (resolutions) in the tree structure.

An idea similar to that of representing a line by strips was recognized by [Peucker, 1976]. In particular he was able to find line intersection and point in polygon algorithms by using sets of bands. Another related idea is that of Burton [1977] who covers curves with tree hierarchies of rectangles of a single orientation. Strip trees is an improvement over both of these ideas because the notion of a strip is a more intuitive and

computationally cleaner way of covering curves. As a result, the algorithms are simpler and more efficient, line-area interesection and area-area intersection and union can now be dealth with, and the tree structures are closed under these operations.

Figure 1. A curve displayed at two resolutions using the hierarchical structure.

2. The Strip Tree

2.1 Notation

We define a strip S to be a six-tuple $(\underline{x}_b, \underline{x}_e, w_r, w_l)$ where[1] $\underline{x}_b = (x_b, y_b)$ denotes the beginning of the strip, $\underline{x}_e$ denotes the end, and w_r and w_l are right and left distances of the strip borders from the directed line segment $[\underline{x}_b, \underline{x}_e)$. These definitions are depicted in Fig. 2. We define w as $w_r + w_l$. We use six parameters even though only five are needed to define a strip. The usefulness of the redundant characterization will become clear after we look at strip tree operations. When the strip consists of a line segment, w = 1, it is important to be precise in defining the end points $\underline{x}_b$ and $\underline{x}_e$. Thus we will regard $\underline{x}_b$ as being included in the segment and $\underline{x}_e$ as not, i.e. the primitive strip is the half open segment $[\underline{x}_b, \underline{x}_c)$.

Figure 2. Definition of a Strip Segment.

A curve is approximated by an open polygonal line given by an ordered list of discrete points $\underline{x}_0, \ldots, \underline{x}_n$ 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</u> w* if there exists an ordered sequence of m strip segments

$$S_k, \; k=0, \; \ldots, \; m-1$$

---

[1] Throughout this paper we will use $\underline{x}$ to denote a point in the plane with coordinates (x, y).

such that

$$(w_l + w_r)_k < w^* \qquad k=0, \ldots, m$$

$$x_i \in \bigcup_{k=0}^{m} S_k \quad i=1, \ldots, n$$

If within a strip segment there is a point y that touches all four sides, then the strip segment is said to be compact. The compactness property is a very important part of most of the algorithms which follow.

2.2. Digitization

Suppose we have a curve C denoted by $[x_0, \ldots x_n)$ like that such as shown by Figure 3a. For any resolution w* we can approximate this line with strip segments as follows:

> Consider the line L defined by $[x_0, x_n)$. For each point x in C find the perpendicular distance d(x) from $\underline{x}$ to this line. Denote the subset of $\underline{x}$ in C such that $\underline{x}.\underline{L}>0$ as C+. C-= C-C+. Now find $w_r = \max_{\underline{x} \in C+} d(\underline{x})$ and $w_l = \max_{\underline{x} \in C-} d(\underline{x})$. If $w < w^*$ then the polyline is compactly represented at resolution w by the strip tree consisting of a single root strip $S(\underline{x}_0, x_n, w_r, w_l)$. If not then the desired strip tree is obtained by recursively applying the algorithm to the curves $[x_0, \ldots, x_k)$ and $[x_k, \ldots, x_n)$ where x_k is the larger of the points which define w_l and w_r and making the results the left son and right son respectively of the strip tree. In the case of ties for x_k at maximum distance d, we will arbitrarily pick the point nearest

the mid point (in arc length).

For the purposes of simplifying the algorithms to follow, we regard the strip trees as completely expanded down to unit line segments, even though they may be colinear. Figure 3 shows an example of two levels of recursion of this algorithm.

The digitization scheme can be visualized as finding the smallest rectangle which has a side parallel to the line segment. This scheme works for closed curves, where $\underline{x}_0 = \underline{x}_n$, if distances are measured with respect to the tangent to the point $\underline{x}_0$. To define the tree unambiguously, the point $\underline{x}_0$ can be picked as the end of the largest diameter of the closed curve.

Figure 3. Steps in the Digitization Process.

To see formally that the convergence is guaranteed, note that a curve C of k points can always be approximated by a single strip segment with length k assuming eight-connectedness. Thus for any w there must be a strip tree with leaves consisting of no more than n/w strip segments which approximate C. Since the digitization algorithm splits each segment into two parts such that each part has finite length, the process must ultimately consider segments of w 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, S. The nodes of the tree are initially ordered on arc length. (Later we will see that when intersecting two areas which are represented in strip trees, this property is sometimes not preserved).

We formally define a strip tree T as either null or a node consisting of the eight-tuple (S, Lson, Rson) where Lson and Rson are strip trees which are either null or have strips S_l and S_r which are related to S by the digitization scheme.

2.4. Why Binary Trees?

The curves 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 curve and for all the algorithms tht 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. Basic Operations on Strip Trees

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 at any node we only have to look at one of the two sons, 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 curve, 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 S then it can be at most $w^{(S)}$ units away from the curve C.

B. If a point z is outside a compact strip S then the distance of the point from the P is bounded by

$$0 \leq z \leq d(z, S) + w(S)$$

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(z,C). Now suppose we want to answer the question: is $d(z,C) < d_0$? If this can be answered affirmatively we will find this out at the point where <u>any</u> upper bound is less than d . 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 $d(z,C) > d_0$? From this discussion we can see that the search will be inefficient only if a large number of the strips are nearly d from z. Figure 5a shows this case together with a more representative example.

Figure 5. Two of many Possible Geometries When Testing the Distance of a Point from a Curve.

To summarize this discussion, we provide the algorithms to test for $d(z,C) < d_0$ and $d(z,C) > d_0$. 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 then its distance to S is characterized by the set theoretic distance $d(z,S) = \min_{x \in S} d(z,x)$ 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≤ d(z,S(T)) + 2*w(T) then return (true);
    if z ∉ S(T) and d0 > d(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 d_0 from a polyline?

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

3.2 Displaying a Curve 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 S has a resolution delta less than some fixed value. 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 Curve at Resolution w

```
procedure CurveDisplay (T,w)
    begin
    if w1(T) + wr(T) < w then DisplayRectangle (S(T))
       else (CurveDisplay (LSon(T),w)) and (CurveDisplay
    (RSon(T),w));
    end;
```

3.3 Intersecting Two Trees

One of the important features of the representation is the ability to compute intersections between curves. 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 (S_1 derived from C_1) and (S_2 derived from C_2) do not intersect iff $S_1 \cap S_2 = \emptyset$

B. Two strip segments S_1, S_2 have a <u>clear</u> intersection iff all the sides parallel to the segments given by $[\underline{x}_b, \underline{x}_e)_1$ and $[\underline{x}_b, \underline{x}_e)_2$ intersect.

C. Two strip segments S_1 and S_2 have a <u>possible</u> intersection if condition B is not satisfied yet $S_1 \cap S_2 = \emptyset$.

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

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

(Peucker showed this was true for the similar case of bands [1976]. To see this for condition B, consult Fig. 6b. C1 divides the region R into two parts and C2 must cross from one to the other. The only way the C2 can do this is by intersecting C1. A simple refinement of the clear intersection lemma is that is strips have a clear intersection, then the underlying curves must intersect an odd number of times.

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.

Algorithm A4: Finding out whether two polylines intersect

Comment. If the two root strip segments do not intersect then the curves do not intersect. If the root segments have a clear intersection then the curves intersect. Since the task is to just determine whether or not an intersection exists, the algorithm terminates upon finding a clear intersection.

```
boolean procedure Intersection (T1,T2,)
```

```
begin
Case StripIntersection (S(T1),S(T2)) into
   [Null] return (false),
   [Possible] if (w(T1)>w(T2)) or (Primitive Flag)
   then
      return ((Intersection(LSon(T1),T2) or
   (Intersection(RSon(T1),T2));
      else return
         (Intersection(T1,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 w(T1) and w(T2); i.e., they can be made to be as small as we want. Fig. 7 shows an example of intersecting two curves at a given resolution.

Figure 7: Intersecting Two Curves at Low Resolution

Note, however, that smaller resolutions may be much more computationally expensive, as shown in the following example (Fig. 8) where intersection at the coarsest resolution is simple, but multiple intersections occur at lower levels.

Figure 8: An intersection may be simple at one level and complicated at lower levels.

If the two curves are not convoluted about each other the intersection should be computed in O(mlog(n)) steps where m is the number of intersection points. If the curves do not intersect but have a closest distance d =ds(C_1,C_2) then this will be discovered at levels in the tree no deeper than a point where $d/2 < w(T_1) + w(T_2)$.

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 $O(2^K)$ 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 Strip Trees

The union of two strip trees can be accomplished by defining a strip that covers both of the two root strips. The two curves defined by $[y_0', \ldots y_n')$, $[y_0'', \ldots, y_m'')$ are treated as two concatenated lists. That is, the resultant ordering is such that we have $y_0 = y_0'$, $y_{m+n+1} = y_m''$.

Algorithm A5: Union of Two Strip Trees T_1 and T_2

Define a strip segment that covers the two root strips of T_1 and T_2. By construction, this strip can be made to satisfy all the properties of a strip segment (except compactness). Make this the root node of a new strip tree.

This construction is shown in Fig. 9.

Figure 9: Construction for Union of Strip Trees Representing Two Polylines

This construction introduces a problem in that the new strip is no longer compact and therefore the Clear Intersection Lemma 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 S:

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

Now a strip is the seven-tuple ($\underline{x}_b$, $\underline{x}_e$, w_l, w_r, C). 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 A4 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 $[\underline{x}_0, \ldots, \underline{x}_i)$, $[\underline{x}_{i+1}, \ldots, \underline{x}_i)$, ..., $[\underline{x}_k, \ldots, \underline{x}_n)$. These segments are given an ordering as shown. The previous digitization algorithm is applied to this set of points, and the perpendicular distance is computed from the set of disconnected curves 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 $\underline{x}^*$) and the digitization algorithm is applied recursively to the subsets. Once this process produces connected subsets, the earlier digitization scheme is applied.

4. Closed Curves Represented by Strip Trees

We represent an area by its boundary which is a closed curve. As we mentioned before, the digitization method described in Section 2 works for closed curves 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 as shown in Fig. 10.

Figure 10: 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 can be carried within the strip trees formalism. Examples of such are intersection between a curve and an area, determining whether a point is inside an area, and intersecting two areas.

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</u>

intersections of the strip tree with any semi-infinite strip L which has delta = 0 and emanates from z. If this number is odd then the point is inside the area.

It is the extension to the clear intersection lemma which makes this property hold: the underlying curves may intersect more than once but must intersect an odd number of times. The following algorithm uses this property 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;
```

Here StripIntersection will report possible instead of clear if either of the strips has C(S) = 0. A potential difficulty exists with the procedure NoOfClearIntersections when the strip SO is tangent to the curve. Many tangent cases will not cause a problem as they will be under clear intersections in an arrangement similar to that of Fig. 7. However if the strip SO passes through an end point at the lowest level then there is no way of determining the parity of the intersection. Fig. 11 shows this ambiguity. To overcome this difficulty in practice, a different SO is used but for the examples tried so far this problem rarely arises.

Figure 11: Indeterminancy of Endpoint Intersections for Inside vs. Outside

4.2 Intersecting a Curve with an Area

The strategy behind intersecting a strip tree representing a curve 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 curve strip tree. This is done efficiently by taking advantage of an obvious property of the intersection process:

> Pruning Property: Consider two strips S_C from T_C and S_a from T_a. If the intersection

of S_C with T_a is null, then (a) if any point on S_C is inside T_a the entire tree whose root strip is S_C is inside or on T_a and (b) if any point on S_C is outside of T_a then the entire tree whose root strip is S_C is outside of T_a.

This leads to the recursive procedure A7 for curve-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. This procedure can be trivially modified to return the part of the curve that is outside of the area, by changing "Inside" to "Not Inside". Similarly, the boundary of T_a can represent the area "outside" of the curve. This tree is denoted T_a and has an extra flag at its root to denote the change of parity to be used by "Inside".

Note that if the polyline strip is "fatter," i.e., w(T1) > w(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: Curve-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 w(T1)>w(T2) then
        begin
        C(NT):=0
        comment non-compact strip
        S(NT) := S(T1) ;
        w(NT):= w(T1);
        LSon(NT):= PAInt (LSon(T1),T2);
        RSon(NT):= PAInt (RSon(T1),T2);
        return(NT);
        end
     else comment w(T1)≤ w(T2)
        Return (PAInt(PAInt(T1,LSon(T2)),RSOn(T2)));
  end;
```

4.3 Intersecting Two Areas

The problem of intersecting two areas is simple using their strip tree representations. Surprisingly, this problem can be decomposed into two curve area intersection problems (refer to Fig. 12).

Figure 12: 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 11a. If we reverse the roles of the two strip trees the result is given by the thick lines in Figure 11b. The union of these two strip trees (see Section 3.4) is the answer we want! Thus we would like to write the area-area intersection procedure in terms of strips as follows:

Algorithm A8: Area-Area Intersection

```
reference procedure AreaAreaInt (T1,T2)

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

where Union is a procedure that accomplishes the construction described in Section 3.4. The actual procedure is almost this simple but contains a modified version of CurveAreaInt to handle special cases at the primitive level. Fig. 13 shows these cases.

Figure 13: Special Cases for Area-Area Intersection

Fig. 14 shows the result of an area-area intersection using strip trees.

Figure 14: Area-Area Intersection

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 curves). However, the integrity of the tree representation is preserved; the new tree can be composed with other trees using any of the tree operations.

4.4 The Union Operation

The union operation is simpler than the intersection operation. For the union of an area-area strip with another, we use a construction similar to the digitization methods for

disconnected curves. The result is an area strip tree. If these two strip trees do not intersect, then the union is straightforward and is identical to the method for curves. 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 area strip tree intersections. That is, the new tree T is defined as:

$$T = \begin{cases} \text{Union } (T1,T2) \text{ if } T1 \cap T2 = \varnothing \\ \text{Union } (\text{CurveAreaInt}(T1,T2),\ \text{CurveAreaIntIT2},T1)) \text{ otherwise} \end{cases}$$

5. Discussions and Conclusions

For the purposes of simplifying the algorithms, the primitives in the tree were regarded as unit segments. In fact, they can be segments of arbitrary lengths, but the corresponding algorithms are more complicated. This is because it is often necessary to divide long segments and build new parts of the tree. The details of these algorithms will be discussed in Tanaka and Ballard [1979]. The advantage of using arbitrary length segments is that any level in the tree can be regarded as a primitive level by using its line segment $[\underline{x}_b, \underline{x}_e)$. Thus all the operations can be carried out at any specified resolution.

Strip trees thus provide a powerful representation for curves. 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 laborious 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 and computer-aided design.

The representation defines strip segments as primitives to cover subsets of the line. 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 Fig. 15. 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 15: The Notion of an Arbitrary Divide-And-Conquer Strategy

Acknowledgements

The author wishes to thank R. Peet and P. Meeker for their work in the preparation of this document. Thanks also go to H. Tanaka and D. Weaver for their 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.

Tanaka, H. and D.H.Ballard, "Extensions of Strip Tree Operations," Computer Science Dept., University of Rochester, forthcoming Technical Report.

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.

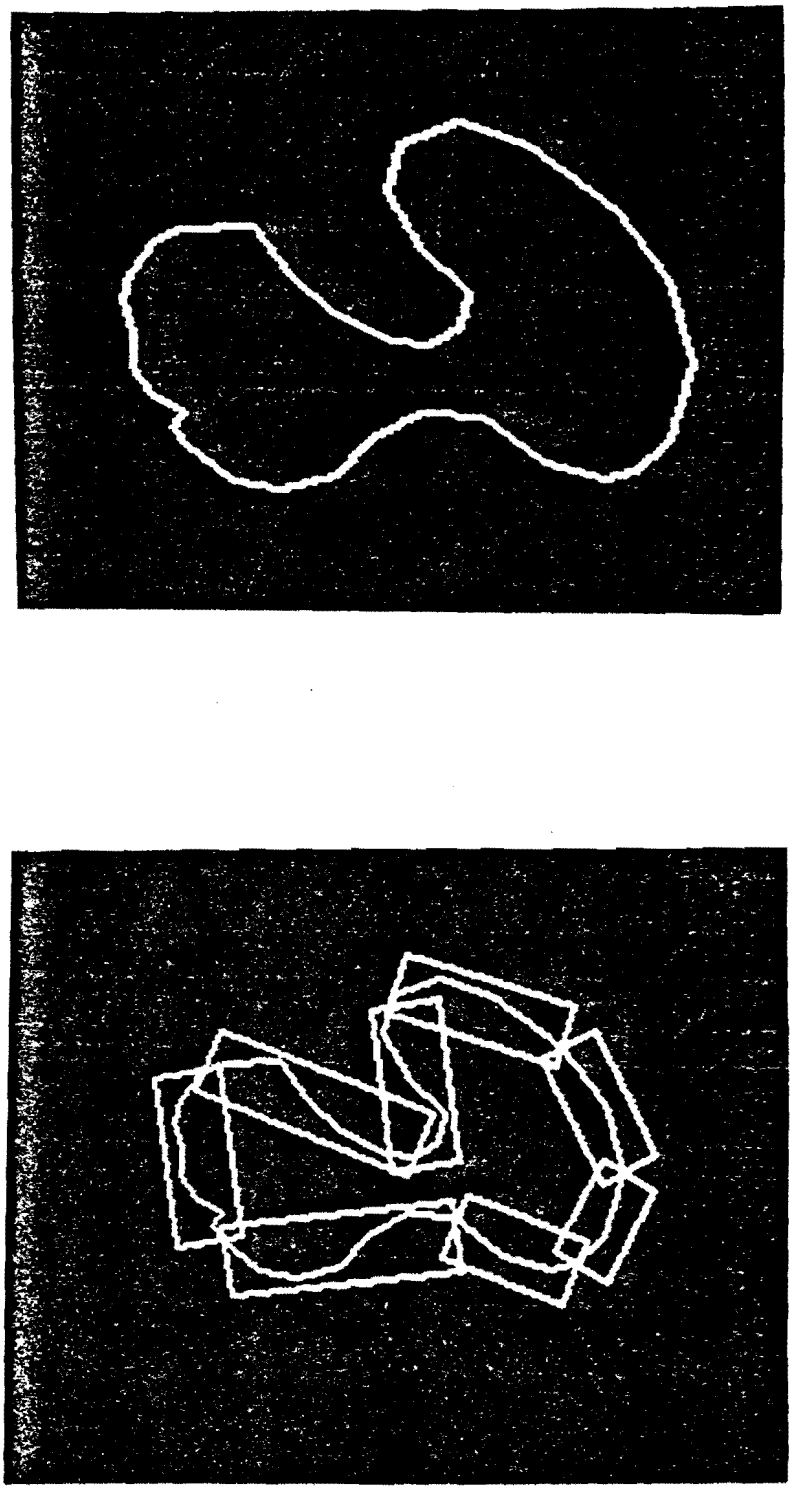


Figure 1: A curve displayed at two resolutions using the hierarchical structure.

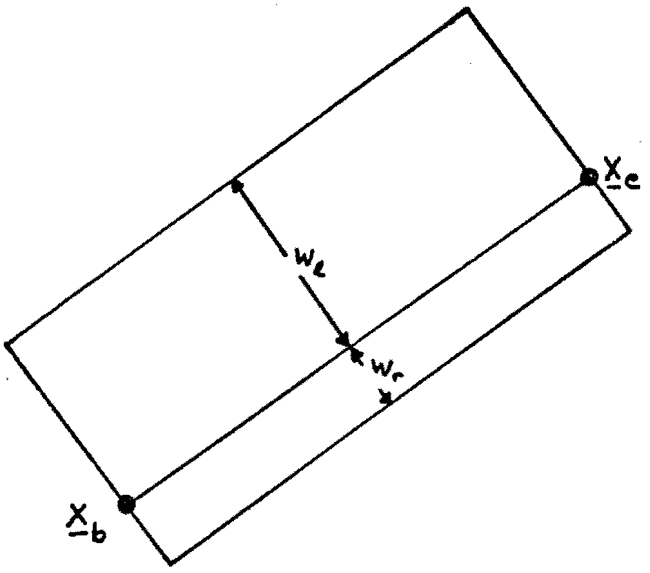


Figure 2: Definition of a Strip Segment.

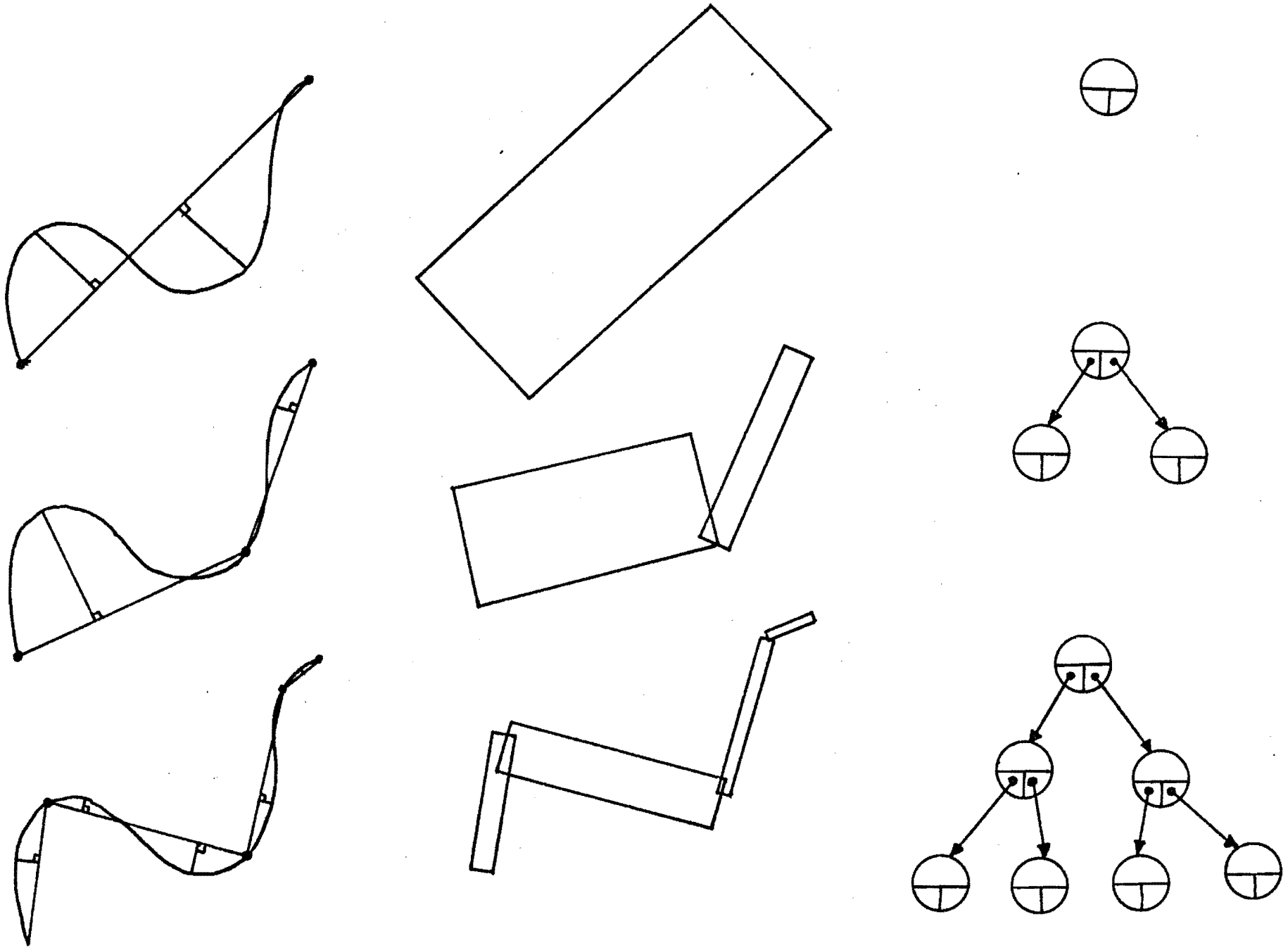


Figure 3: Steps in the Digitization Process.

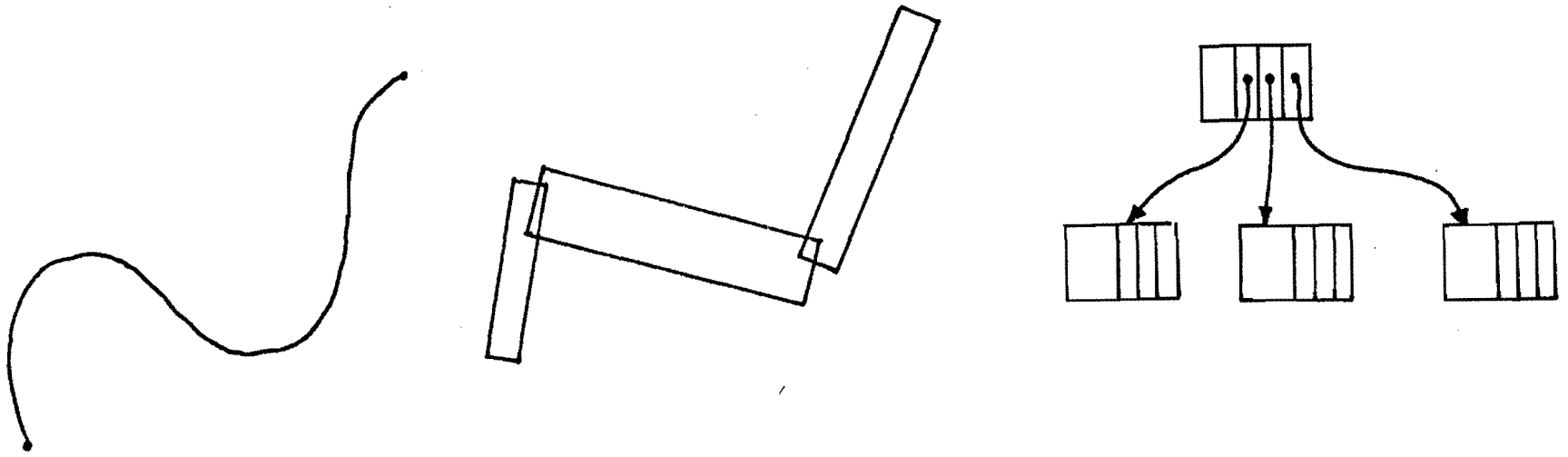


Figure 4: A portion of an encoding using m-ary trees.

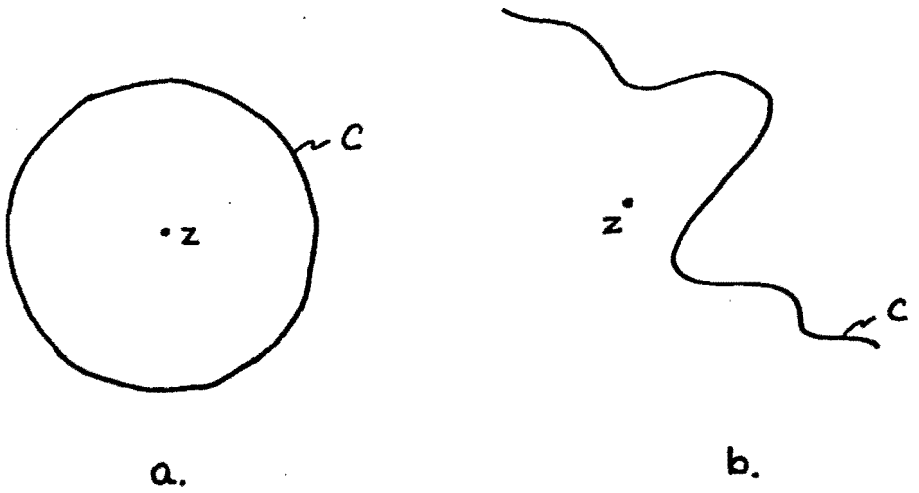


Figure 5: Two of Many Possible Geometries When Testing the Distance of a Point from a Curve.

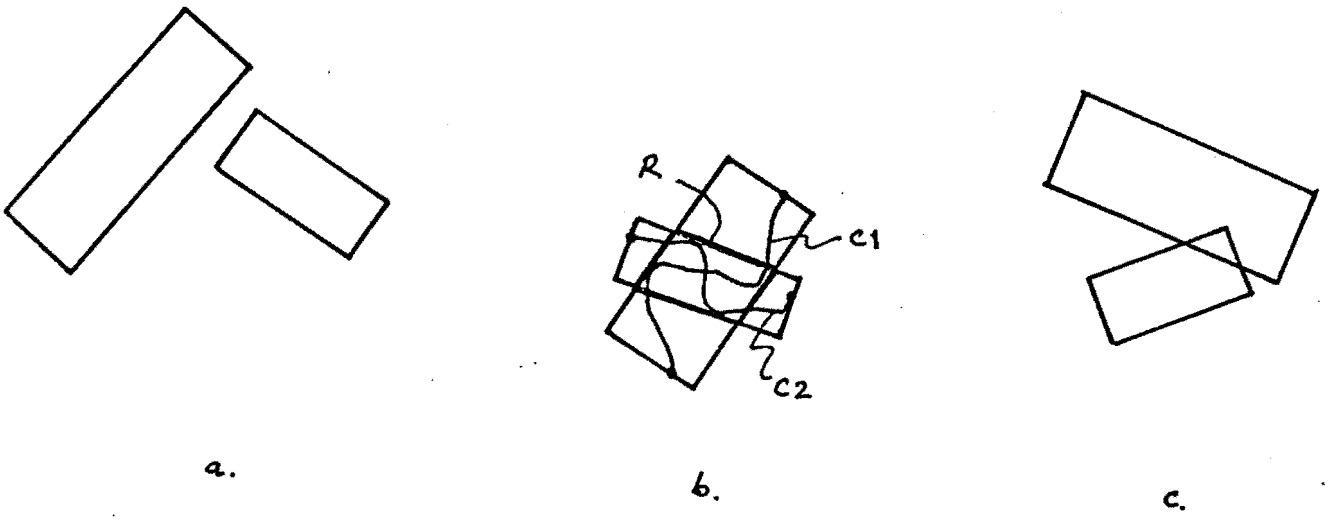
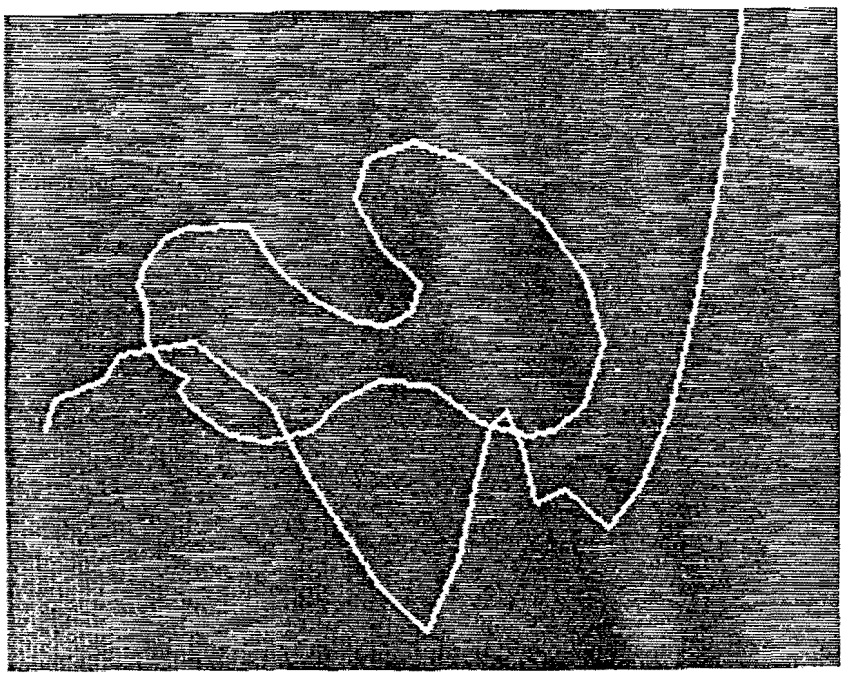


Figure 6: Different Ways Strips Can Intersect.



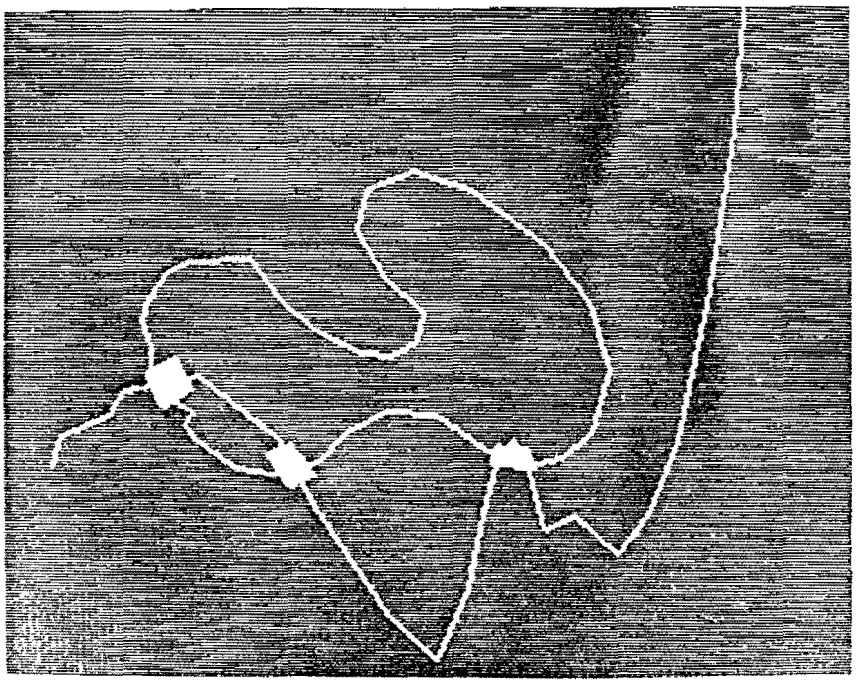


Figure 7: Intersections at Low Resolution.

a. Two curves

b. Results of Strip Tree Intersection

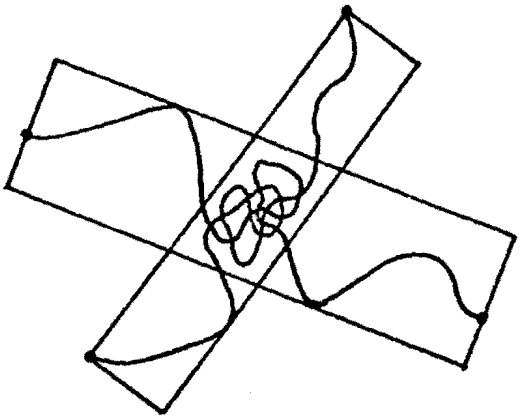


Figure 8: An Intersection may be simple at one level and complicated at lower levels.

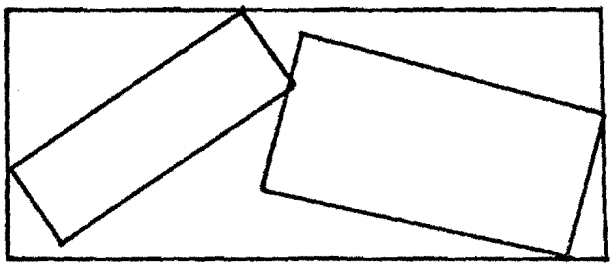


Figure 9: Construction for Union of Strip Trees Representing Two Polylines

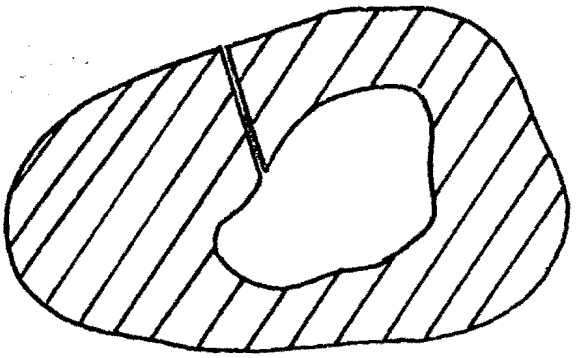


Figure 10: A Region with a Hole

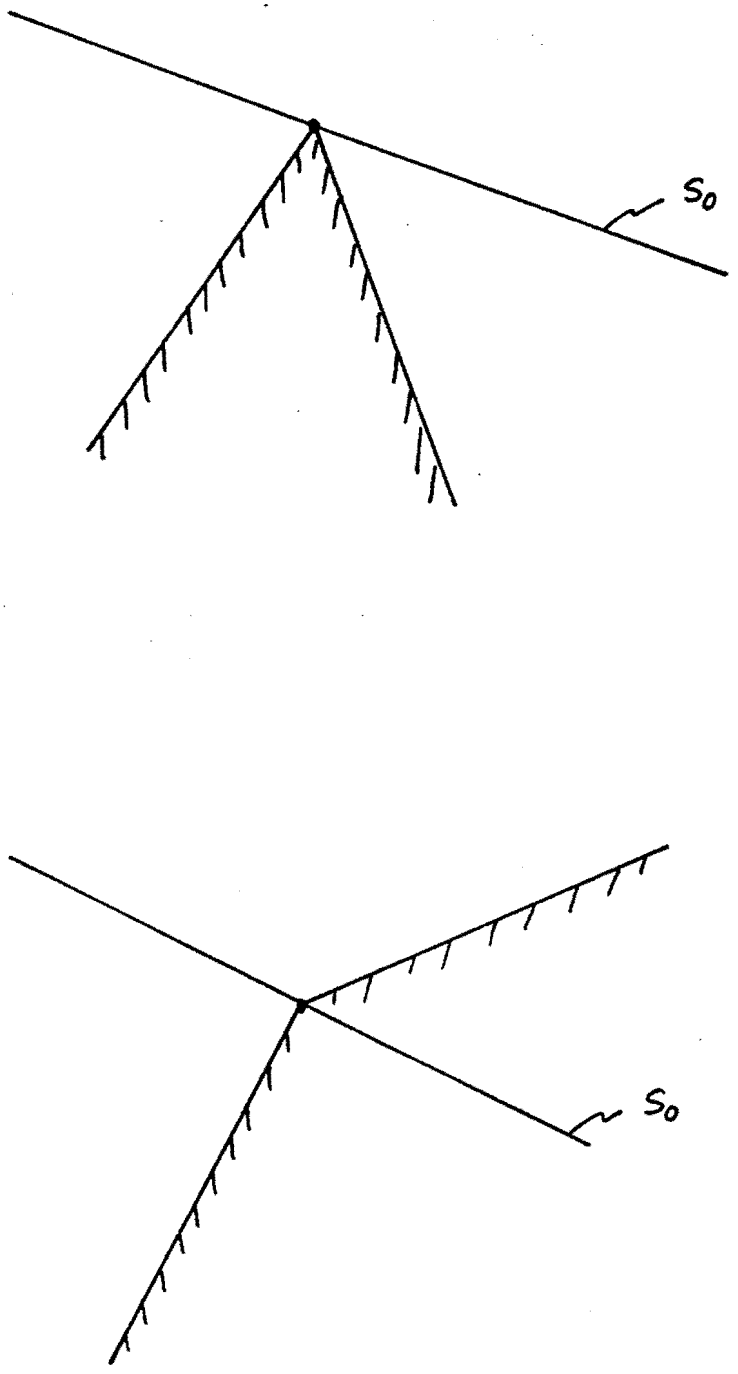
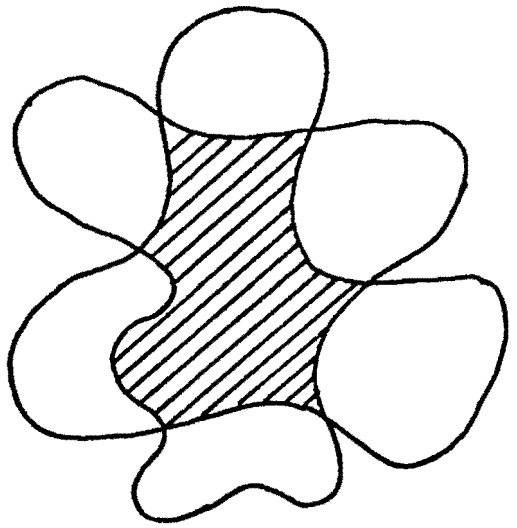
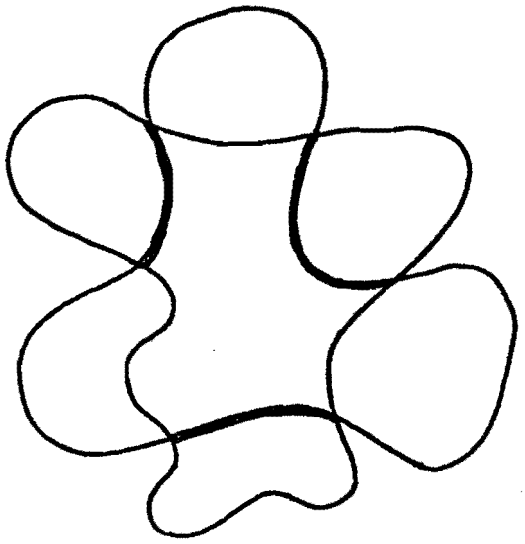


Figure 11: Indeterminancy of Endpoint Intersections for Inside vs. Outside

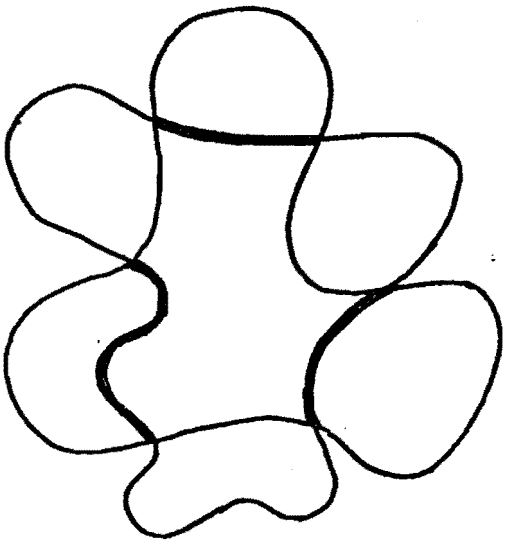
Fig 11



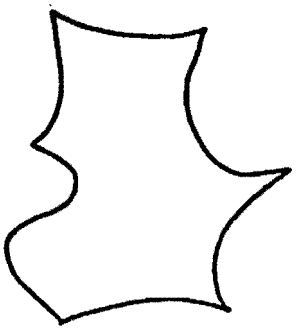
a.



b.



c.



d.

Figure 12: 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.

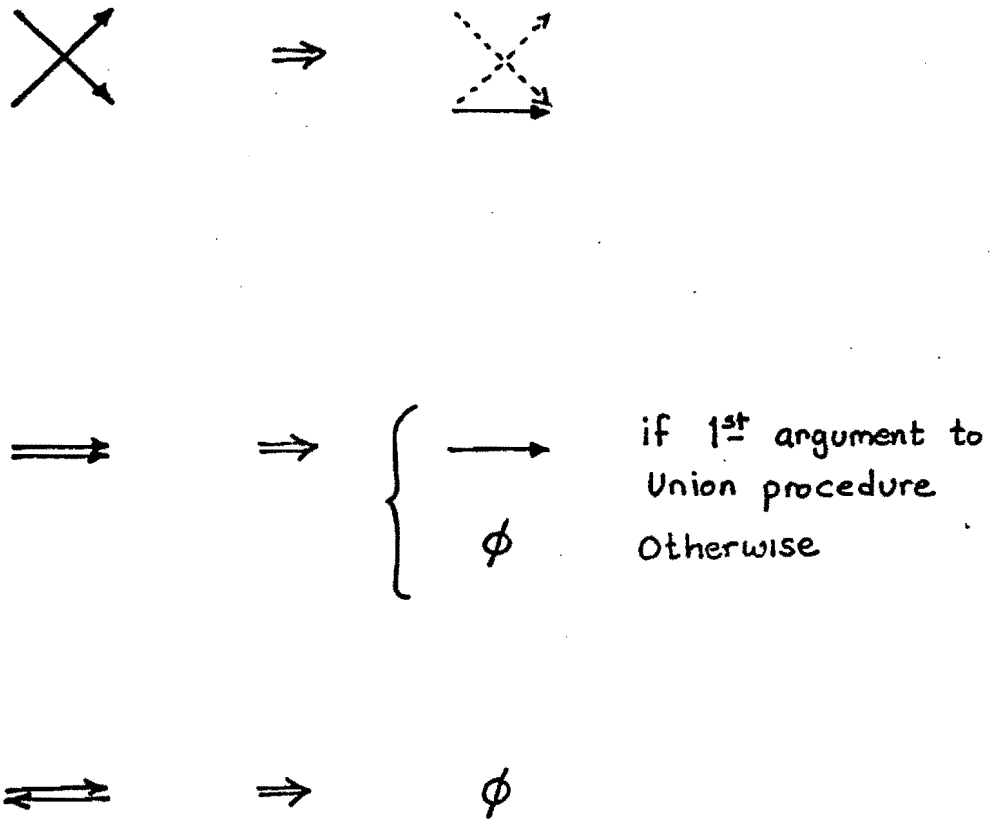
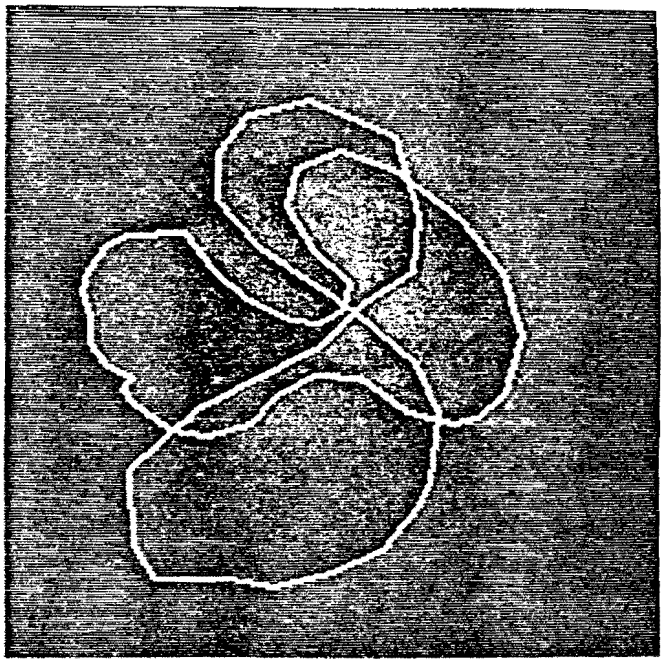


Figure 13: Special Cases for Area-Area Intersection



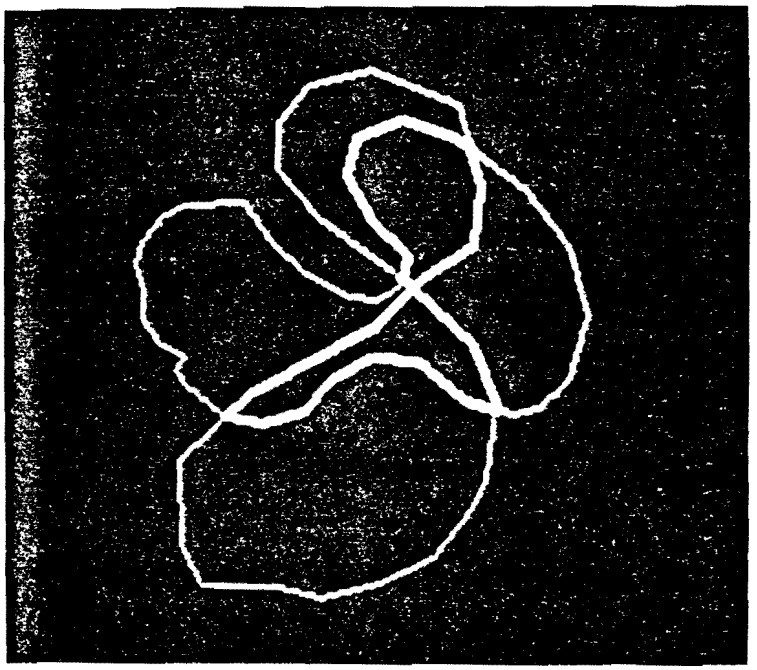


Figure 14: Area-Area Intersection

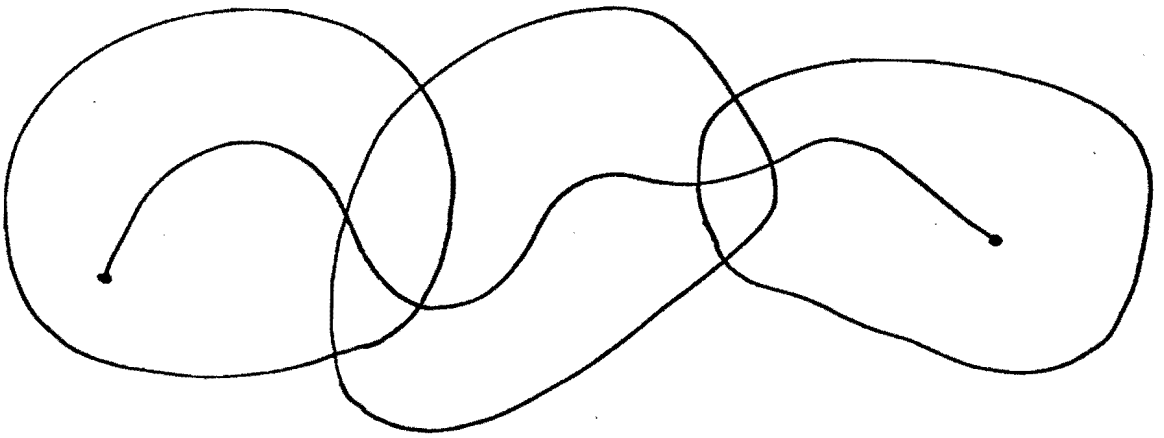


Figure 15: The Notion of an Arbitrary Divide-And-Conquer Strategy