Graphics and Image Processing

J. Foley Editor

Strip Trees: A Hierarchical Representation for Curves

Dana H. Ballard
University of Rochester

The use of curves to represent two-dimensional structures is an important part of many scientific investigations. For example, geographers use curves extensively to represent map features such as contour lines, roads, and rivers. Circuit layout designers use curves to specify the wiring between circuits. Because of 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. A hierarchical representation consisting of binary trees with a special datum at each node is described. 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 a special method for digitizing lines and retaining all intermediate steps. This gives several desirable properties. For curves that are well-behaved, intersection and point-membership (for closed curves) calculations can be resolved in 0(log n) where n is the number of points describing the curve. The curves can be efficiently encoded 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 time savings.

Strip trees is a linear interpolation scheme which realizes an important space savings by not representing all the points explicitly. This means that even when the overhead of the tree indexing is added, the storage requirement is comparable to raster representations which do represent most of the points explicitly.

Key Words and Phrases: boundary line representation, cartography, computer graphics, computer-searchable structures, contour representation, geographic information processing, graphic data retrieval, intersection of curves, line-drawing processing, points in polygons, polygons, regional boundary representation, spatial information.

CR Categories: 3.14, 3.23, 3.30, 3.79, 8.2

I. Introduction

A general representation for planar curves is presented. This representation allows operations such as union and intersection to be performed efficiently and is thus of greast interest in fields such as geography and computer-aided circuit design, which use data bases of planar curves.

Consider the application to geography. A map has several interesting kinds of features: contour lines, lakes, rivers, and roads. These can be roughly divided into four feature classes [13].

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 curves and areas. A point is such a simple datum that it can easily be treated as a primitive in any representation. Collections of points from a single class can be efficiently represented as k–d trees [1, 2]. A strip feature is essentially a line where a locally varying thickness is important, e.g., rivers and roads. Our representation of curves will encompass these types of features.

Collections of these map features are regarded 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.

The research described in this report was supported partially by the Defense Advanced Research Projects Agency Grant N00014-78-C-0164 and partially by the National Institutes of Health Grant R23-HL-21253-01.

Author's present address: Dana H. Ballard, Department of Computer Science, University of Rochester, Mathematical Sciences Building, Rochester, NY 14627.

Fig. 1. A Curve Displayed at Two Resolutions Using the Hierarchical Structure.

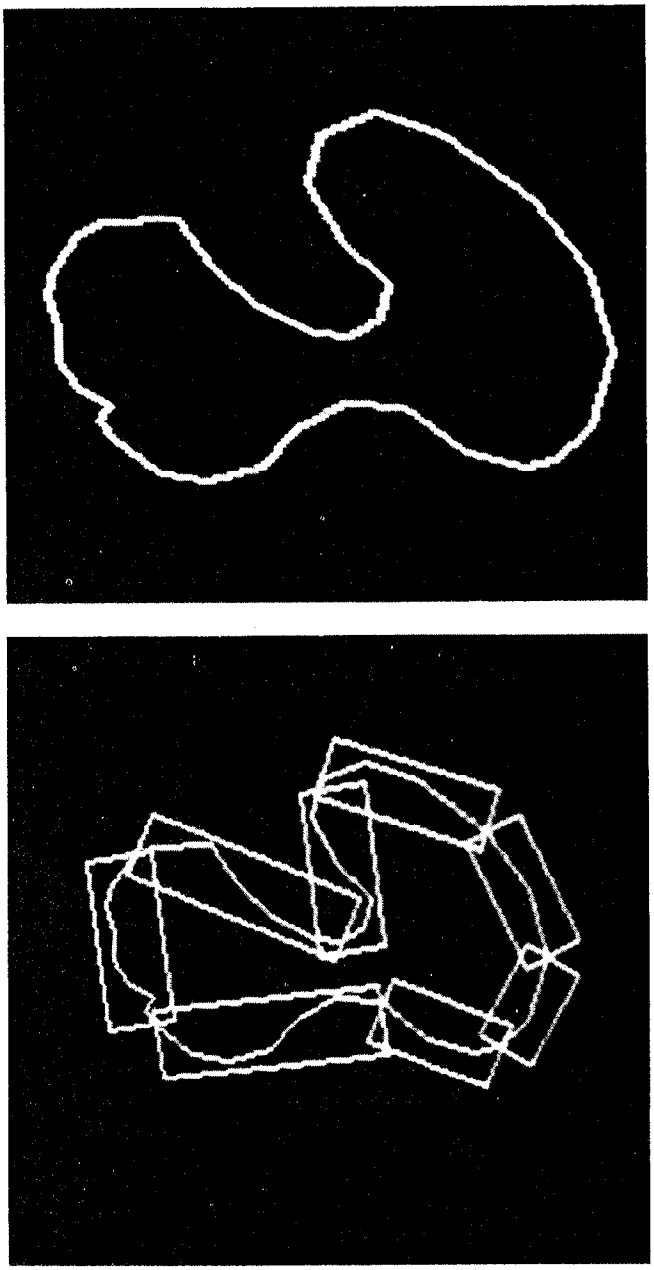
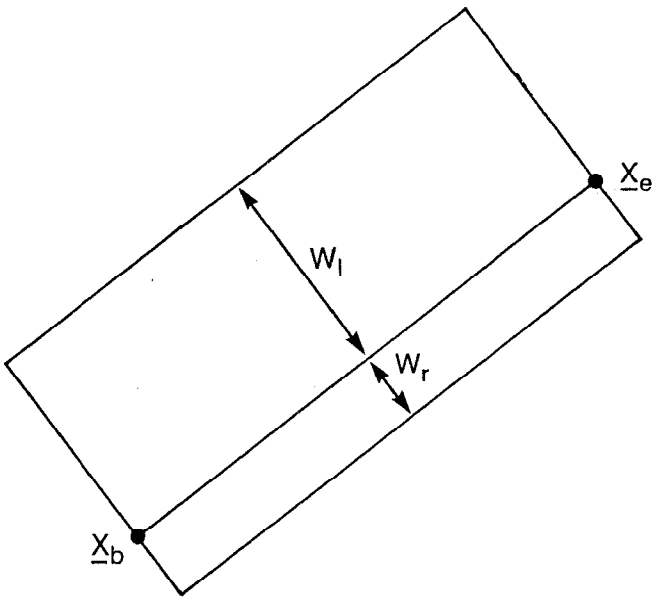


Fig. 2. Definition of a Strip Segment.



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

Our representation of 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 Duda and Hart's [5] method for digitizing lines (see also [16]) and retaining all intermediate steps in the digitization process. Figure 1 is an example of a curve represented at two levels (resolutions) in the tree structure.

Peucker [10] recognized an idea similar to that of representing a curve by strips. In particular, he was able to find line intersection and point-in-polygon algorithms by using sets of bands. Burton [3], on the other hand, covered curves with tree hierarchies of rectangles of a single orientation. Strip trees are an improvement over both of these ideas because the notion of a strip is a more intuitive and computationally cleaner way of covering curves. Consequently, the algorithms are simpler and more efficient, the curve–area intersection and area–area intersection and union can now be dealt with, and the tree structures are closed under these operations.

II. The Strip Tree

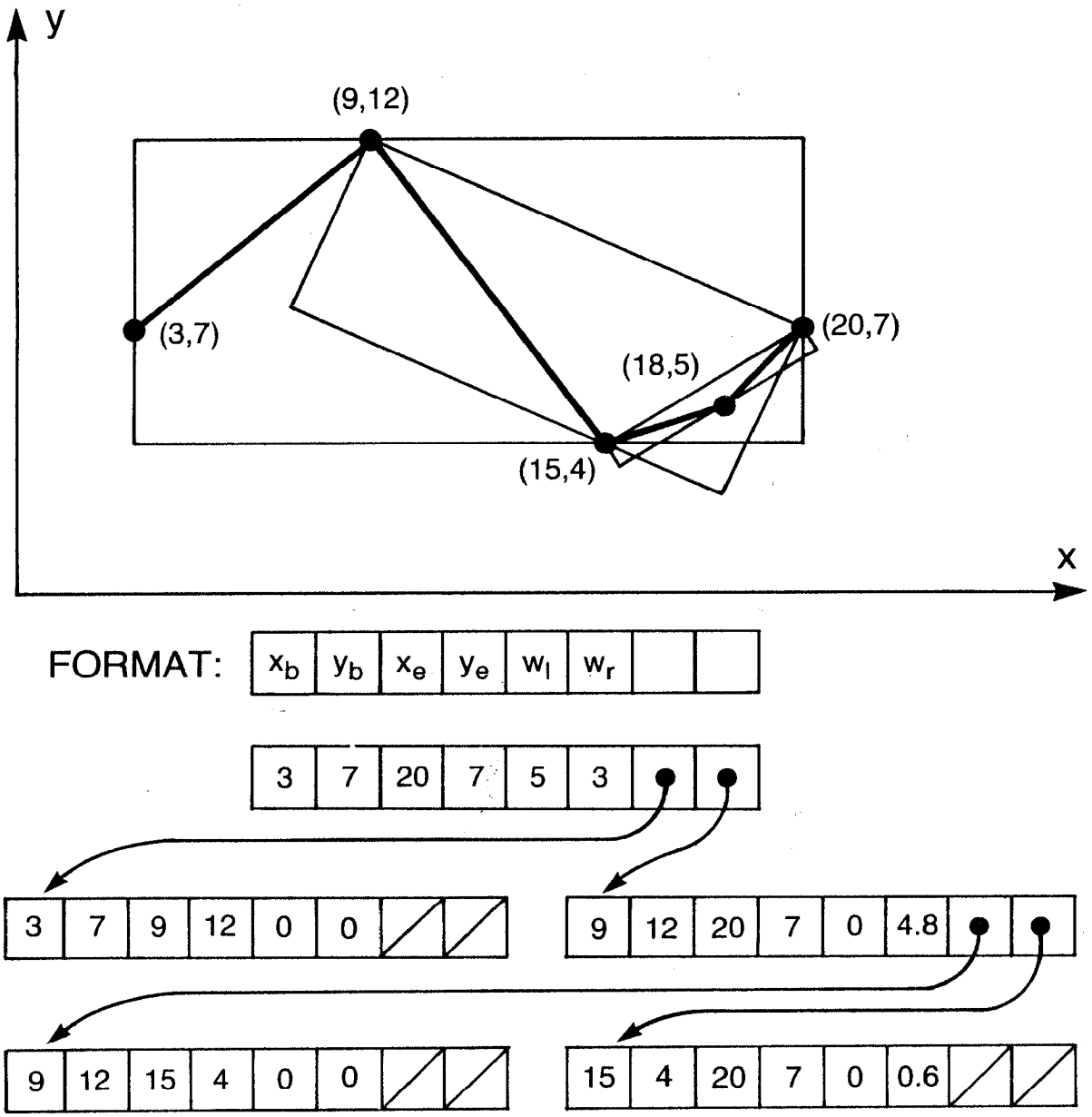
II. A. Notation

A strip S is defined to be a six-tuple (x_b, x_e, w_r, w_l) where [1] $x_b = (x_b, y_b)$ denotes the beginning of the strip, 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 $[x_b, x_e)$. These definitions are depicted in Figure 2. w is defined as $w_r + w_l$. We retain both w_r and w_l, i.e., all six parameters even though only five are needed to define a strip. The usefulness of the redundant characterization will become clear after strip tree operations are looked at. When the strip consists of a line segment, precision is important in defining the end points x_b and x_e. Thus, x_b is regarded as included in the segment and x_e is not, i.e., the primitive strip is the half-open segment $[x_b, x_c)$. Occasionally, discussion about the area of a strip, which is simply $w\|x_b - x_c\|$, will be necessary. This area is denoted by A.

A curve is approximated by an open polygonal line given by an ordered list of discrete points $\mathbf{x}_0, \ldots, \mathbf{x}_n$, subsets of which may be collinear. For the moment these

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

Fig. 3. An Example of the Digitization Algorithm ($w^* = 3$).



points are considered as connected; later this condition will be relaxed. A curve is *represented at resolution* w^* if there exists an ordered sequence of m strips

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

such that

$(w_l + w_r)_k < w^* \quad k = 0, \ldots, m-1,$

$\mathbf{x}_i \in \bigcup_{k=0}^{m-1} S_k \; i = 0, \ldots, n$

and the S_k are defined by the following digitization algorithm.

II. B. Digitization

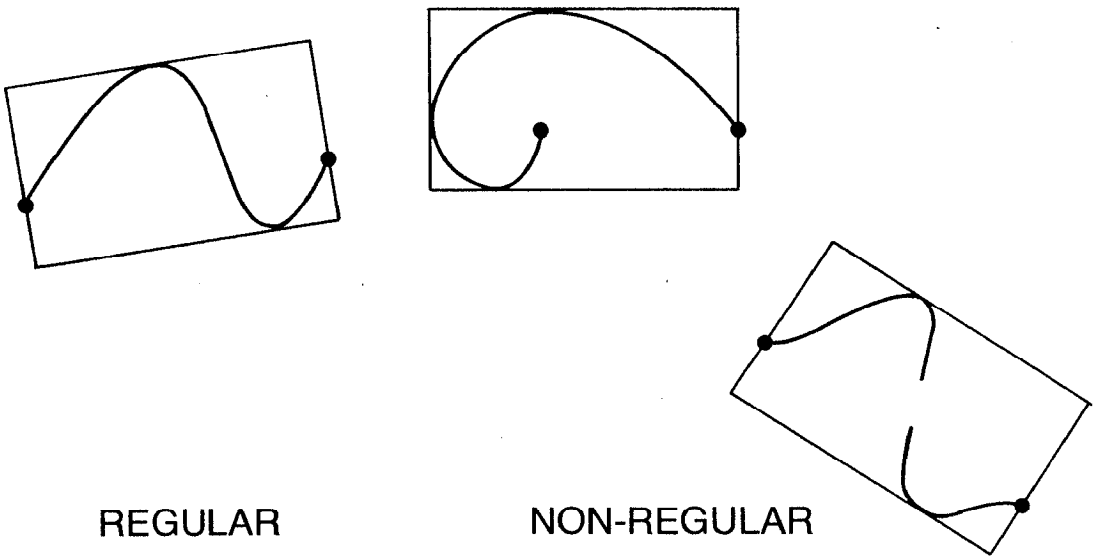
Suppose a curve C is denoted by $[\mathbf{x}_0, \ldots \mathbf{x}_n)$. For any resolution $w^* \geq 0$, this line can be approximated with strips as follows:

Algorithm A0 Digitization
Find the smallest rectangle with a side parallel to the line L through $\mathbf{x}_0$ and $\mathbf{x}_n$ which just covers all the points. This rectangle is the strip of the root node of the strip tree. Now pick a point $\mathbf{x}_k$ which touches one of the two sides of the rectangle that are parallel to L. Repeat the process for each of the two sublists $[\mathbf{x}_0, \ldots, \mathbf{x}_k)$ and $[\mathbf{x}_k, \ldots, \mathbf{x}_n)$. This results in two subtrees that are sons of the root node. The process terminates when all strips have width $w \leq w^*$.

Figure 3 shows an example of two levels of recursion of this algorithm.

To see formerly that the convergence is guaranteed, note that a curve C of length P can always be approximated by a single strip segment with width P. Thus, for any w there must be a strip tree with leaves consisting of no more than P/w strip segments which approximate C.

Fig. 4. Regular and Nonregular Strips.



Since the digitization algorithm splits each segment into two parts such that each part has finite length, the process must ultimately consider strips of width w or less.

In the digitization process, special cases arise when the curve is closed or extends beyond its endpoints. The scheme works for closed curves where $\mathbf{x}_0 = \mathbf{x}_n$, if distances are measured with respect to the tangent to the point $\mathbf{x}_0$. To define the tree unambiguously, the point $\mathbf{x}_0$ can be picked as the end nearest the origin of the largest diameter of the closed curve. In the second case, either of the ends of the strip, $\mathbf{x}_b$ or $\mathbf{x}_e$, may not be on the curve, but are still chosen to be on the line L so that the curve is entirely contained within the strip.

Algorithm A0 does not take advantage of the fact that the points are ordered by arc length. This can be used to improve the digitization algorithm in the following way.

Algorithm A0′ Improved Digitization:
Take pairs of points $(\mathbf{x}_0, \mathbf{x}_1), (\mathbf{x}_1, \mathbf{x}_2) \ldots (\mathbf{x}_{n-1}, \mathbf{x}_n)$ and cover them with strips $S_0^1 \cdots S_{n-1}^1$. Now take pairs of these strips $(S_0^1, S_1^1), (S_2^1, S_3^1) \cdots (S_{n-2}^1, S_{n-1}^1)$ and cover them with larger strips $S_0^2, S_3^2, \ldots$, etc. If there are an odd number of strips, use the extra strip at the next level. Ultimately, there will be a single root strip.

This algorithm is of $0(n)$ as opposed to the previous algorithm, $0(n \log n)$. The strips in the A0′ digitization do not cover the curve as tightly as those from Algorithm A0, but the complexity of the algorithms that operate on these trees is the same. For our purposes, it is assumed that the trees have been created by the A0 Algorithm.

Solely for the purpose of simplifying the algorithms to follow, the strip trees are regarded as completely expanded down to a *primitive* level of unit line segments on a discrete grid, even when the underlying curves are collinear, i.e., the algorithms assume that at the leaf level, $\mathbf{x}_0$ and $\mathbf{x}_e$ are grid neighbors on some discrete grid. For example, a primitive level for the example of Figure 3 would be ((3, 7), (4,8), (5,9), (6,10), (7,11) (8,11), (9,12), (10,11), (11,10), (11,9), (12,8), (13,7), (14,6), (14,5), (15,4), (16,4), (17,5), (18,5), (19,6), (20,7)). The reasons for this will become apparent when the algorithms are described. It will also become apparent that this simplification is not essential. Finally, the modifications for curves of arbitrary segments are described.

II. C. Regular Strips

It is important to know whether a strip is *regular*. A strip is defined to be regular if its underlying curve:

(1) is connected;

(2) has its endpoints touching the ends of the strip.

Examples of regular and nonregular strips are shown in Figure 4. To keep track of this property, strips have an associated bit $C(S)$ such that

$$C(S) = \begin{cases} 1 & \textit{if strip } S \text{ is regular;} \\ 0 & \text{otherwise.} \end{cases}$$

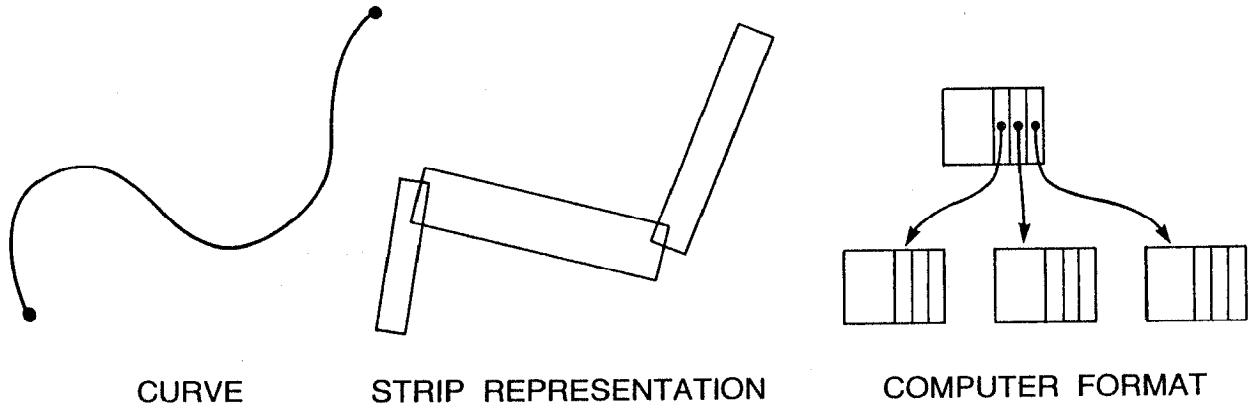
The introduction of the regularity bit $C(S)$ also allows us to digitize curves that consist of disconnected parts. To do this, the segments are ordered and Algorithm A0 is used. The only modification required is to set $C(S) = 1$ if the underlying curve turns out not to be connected.

II. D. 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 leaf nodes of the tree are initially ordered on arc length. (It will be seen later that when intersecting two areas that are represented in strip trees, this property is sometimes not preserved.)

A strip tree T is formally defined as either null or a node consisting of the tuple (S, *LSon*, *RSon*), i.e., ($\mathbf{x}_b$, $\mathbf{x}_e$, w_l, w_r, *LSon*, *RSon*) where *LSon* and *RSon* are strip trees that are either null or have root strips that are related to S by the digitization scheme.

Fig. 5. A Portion of an Encoding Using m-ary Trees.



II. E. 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 ordered siblings. Figure 5 shows an example of the alternate encoding scheme. In certain cases this may be a more concise representation of the curve and for all the algorithms that follow, the operations can be extended from two sons to multiple sons. However, this change does not alter the complexity of the operations that we would like to perform. In fact, the nonbinary tree of the operations can be more inefficient than the binary tree representation.

III. Basic Operations on Strip Trees

Computational complexity of the various operations is difficult to characterize because it depends on the particular geometry of the curves. If the curves are "well-behaved," i.e., if 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 is that in terms of the strip tree, if we only look at one of the two sons at any node, then the complexity of the operation is 0(log n).

III. A. The Length of the Curve

The strip tree provides a simple way of calculating the length of a curve at a given resolution. To compute the length of a curve at resolution w^* the following procedure is used. (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 this should be obvious from the earlier definitions.

```
Algorithm A1 Length of a Curve
Real procedure Length (w*,T);
  begin
  if w(T) < w* then
    return (SQRT ((x_b(T) − x_e(T))^2 + (y_b(T) − y_e(T))^2)
  else
    return (Length(w*,RSon(T)) + Length (w*,LSon(T)));
  end;
```

Of course, Length (w^*,T) is only an estimate of the true length which in the case of Peano and space-filling curves may be infinite. Such curves have been termed fractals by Mandelbrot [7], who has suggested schemes similar to strips for approximating them.

III. B. Testing the Proximity of a Point

To test the proximity of a point to a curve, the notion is used of the distance of a point to a set that 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 $d(z,S) = min_{x \in S}\, d(\mathbf{z},\mathbf{x})$ where d is the Euclidean distance between the points x and z.

The strip tree may be used to find out quickly if a point is near a curve by exploiting the following property:

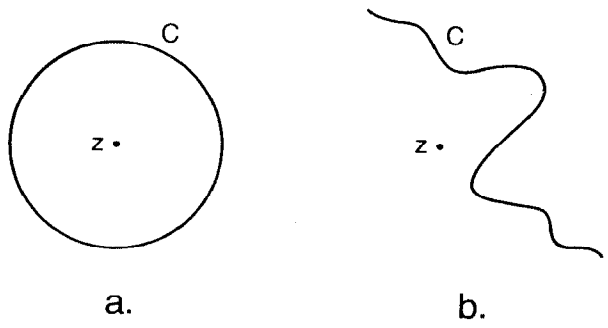
Upper Bounds Property

(a) If a point z is inside a regular strip S, then it is, at most, w(S) units away from the curve C.

(b) If a point z is outside a regular strip S, then the distance of the point from the curve is bounded by $d(\mathbf{z},S) + w(S)$.

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 $d(\mathbf{z},C)$ Now suppose we want to answer the question, for some distance d_0, is $d(\mathbf{z},C) < d_0$? If this question can be answered affirmatively, we will find this out at the point where *any* 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 $d(\mathbf{z},C) > d_0$? It can be seen from this discussion that the search will be inefficient only if a large number of the strips are nearly d_0 from z. Figure 6(a) shows this case together with a more representative example in Figure 6(b). In the former case, if d_0 is equal to the radius of the circle, all of the strips will be within the upper and lower bounds. In the latter, more representative case, many points on the curve can be elimi-

Fig. 6. Two of many Possible Geometrics when Testing the Distance of a Point from a Curve.



nated since their strips will fail the bound checks. To summarize this discussion, the algorithms to test for $d(\mathbf{z},C) < d_0$ and $d(\mathbf{z},C) > d_0$ are provided.

First define LowerBound $(\mathbf{z},S)$ for a point $\mathbf{z}$ and strip S as:

```
if z is inside S then
  LowerBound(z,S) = 0
else
  LowerBound(z, S) = d(z, S);
```

and define Upperbound $(\mathbf{z},S)$ as:

```
if z is inside S then
  UpperBound(z,S) = w(S)
else
  UpperBound(z,S) = d(z,S) + w(S);
```

Then the algorithms are as follows:

Algorithm A2 Is a point within d_0 of a curve?

```
Boolean procedure Within (z,d0,T);
  begin
  S := T;
  if d0 ≤ LowerBound(z,S) then return (true);
  if d0 ≥ UpperBound(z,S) then return (false);
  return (Within (z,d0,LSon(T)) or
    Within (z,d0,RSon(T)));
  end;
```

Algorithm A3 Is a point further than d_0 from a curve?

```
Boolean procedure Further (z,d0,T);
  begin
  S := T;
  if d0 ≤ LowerBound (z,S) then return (false);
  if d0 ≥ UpperBound(z,S) then return (true);
  return (Further (z,d0,LSon(T)) and
    Further (z,d0,RSon(T)));
  end;
```

III. C. Displaying a Curve at Different Resolutions

As demonstrated in Sec. II, a curve may be represented as a set of strips such that each strip has a resolution width less than some fixed value. The strip tree algorithm to display such a representation is as follows. This algorithm uses a device-dependent subroutine DisplayRectangle which paints the rectangle on the particular display device.

Algorithm A4 Display a curve at resolution w^*

```
procedure CurveDisplay (T,w*);
  begin
  if w(T) ≤ w* then Display Rectangle (T)
  else (CurveDisplay (LSon(T),w*)) and
      (CurveDisplay (RSon(T),w*));
  end;
```

III. D. Intersecting Two Trees

One of the important features of the representation is its' ability to compute intersections between curves. Strip trees provide the facility not only to 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, the following definitions are necessary:

(a) Two strip segments (S_1 derived from C_1 and S_2 derived from C_2) have a *null* intersection *iff* $S_1 \cap S_2 = \emptyset$.

(b) Two strip segments S_1, S_2 have a *clear* intersection *iff* all the sides of the strips parallel to the line segments given by $[\mathbf{x}_b, \mathbf{x}_e)_1$ and $[\mathbf{x}_b, \mathbf{x}_e)_2$ intersect and both strips are regular.

(c) Two strip segments S_1 and S_2 have a *possible* intersection if condition (b) is not satisfied, yet $S_1 \cap S_2 \neq \emptyset$.

These cases are illustrated by Figure 7. A fairly obvious but very important lemma is

CLEAR INTERSECTION LEMMA. *If two strip segments have a clear intersection and the strips are both regular, then the corresponding curves must also intersect.*

(Peucker [10] showed this was true for the similar case of bands.) To see this for condition (b), consult Fig. 7(b). With a clear intersection there will be a region R where the strips overlap. C_1 will divide into two parts and C_2 must cross from one to the other. The only way C_2 can do this is by intersecting C_1.

The algorithms to check for intersections between two curves are recursive and assume the existence of an integer procedure StripIntersection that will identify the type of intersection (null, clear, or possible).

Algorithm A5 Finding out whether two curves 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 just to determine whether or not an intersection exists, the algorithm terminates upon finding a clear intersection. A heuristic in the recursion is to divide the largest strip in the recursion;

```
Boolean procedure Intersection (T1, T2);
  begin
  Comment strips at the leaf levels must be either Null or Clear. A(T)
        is the area of the strip at node T.
        Case StripIntersection (T1, T2) of
    [Null]
      return (false);
    [Possible]
      if A(T1) > A(T2) 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;
```

StripIntersection is a procedure which performs the geometrical computations necessary to determine the strip intersection type according to the earlier definitions. This procedure is easily modified to return a set of parallelograms comprising intersection points. Other easy modifications can be made to constrain these parallelograms to be of a certain size related to the widths of the strips in T_1 and T_2; i.e., they can be made to be as small as we want. Figure 8 shows an example of two intersecting curves at a given resolution. Note, however, that smaller resolutions may be much more computationally expensive, as shown in Figure 9 where intersection at the coarsest resolution is simple, but multiple intersections occur at lower levels.

If the two curves are not too badly intertwined about each other, the intersection should be computed in $O(m \log(n))$ steps where m is the number of intersection points. It is assumed that for each intersection point, one branch of each tree will usually have to be examined.

The worst case performance is intolerable because the algorithm's computation grows 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! This assumes that all intersections in the tree above the primitive level are possible. In a practical application, one way of handling this case would be to report the possible intersection regions at the point where the limit of some bound on allotted resources was exceeded.

III. E. 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 $[\mathbf{x}'_0, \ldots, \mathbf{x}'_n)$, $[\mathbf{x}''_0, \ldots, \mathbf{x}'_m)$ are treated as two concatenated lists. That is, the resultant ordering is such that $\mathbf{x}_0 = \mathbf{x}'_0$, $\mathbf{x}_{m+n+1} = \mathbf{x}''_m$.

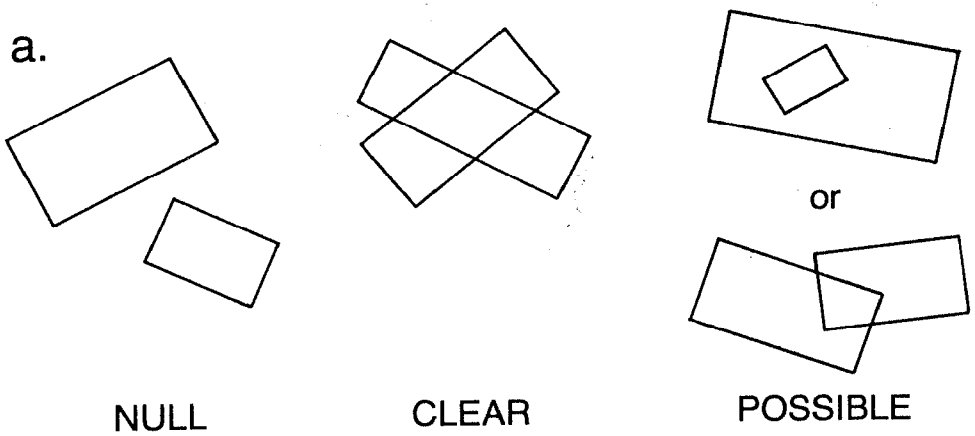


Fig. 7. Different Ways Strips Can Intersect.

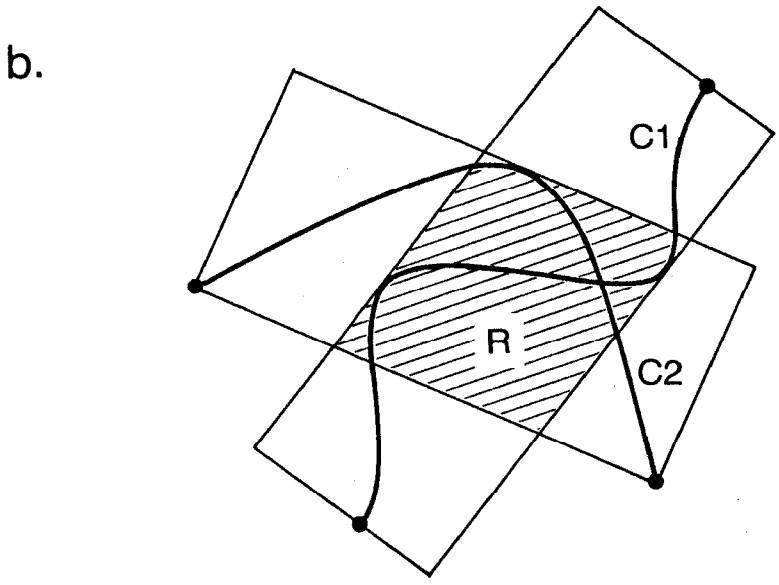
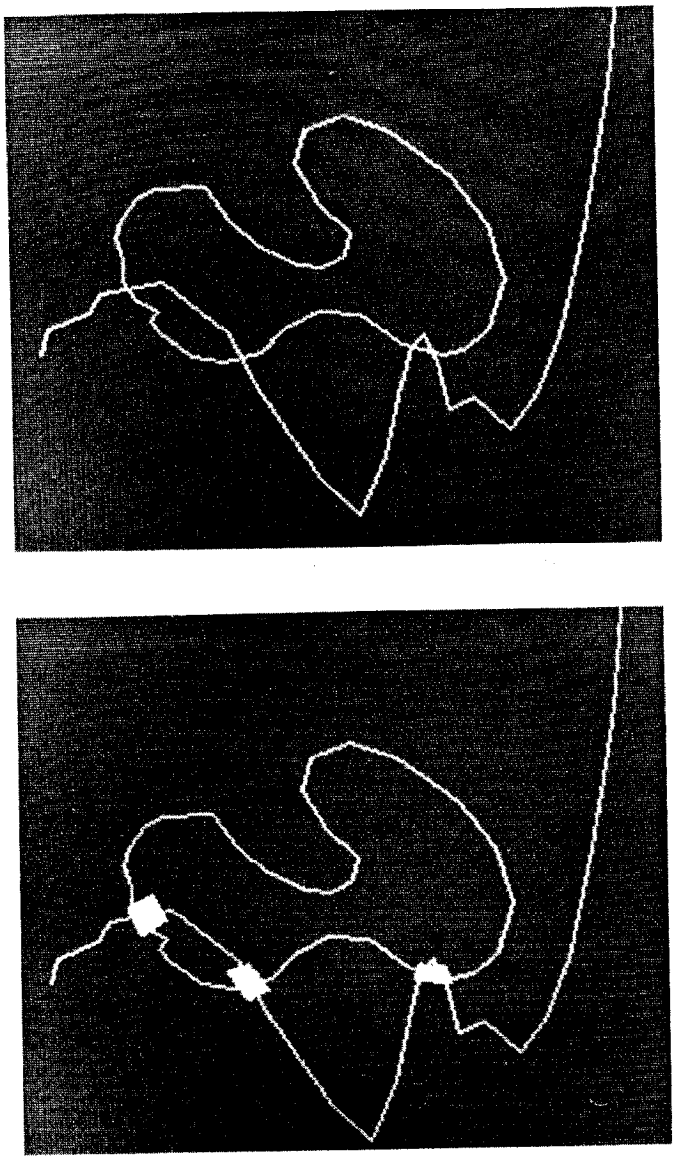


Fig. 8. Intersecting Two Curves at Low Resolution.

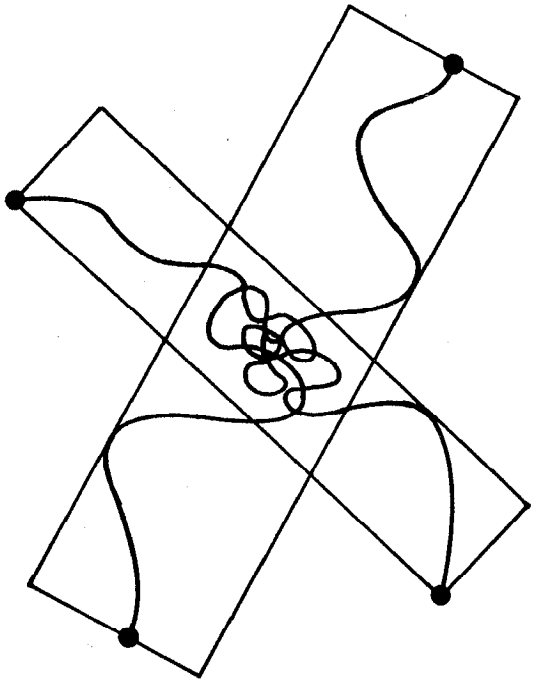


Algorithm A6 (Union of two strip trees T_1 and T_2)
Define a strip segment that covers the two root strips of T_1 and T_2. By construction, this strip can be made to satisfy all the properties of a strip segment (but it need not be regular). Make this the root node of a new strip tree.

This construction is shown in Figure 10. This construction introduces a problem in that the new strip is no longer regular and therefore the Clear Intersection Lemma no longer holds. It was partly to overcome this problem that one bit of information was added to each node to mark whether the underlying curve is regular. This is also important, since later algorithms may produce strips whose underlying curves are not regular.

The regularity bit allows preservation of the elegance of the previous algorithms in the following manner: when bit $C(S)$ is not one, the recursion is always applied. In Algorithm A5 this means that clear intersections are reported as possible if the bit $C(S)$ is set.

Fig. 9. An Intersection may be Simple at one Level and Complicated at Lower Levels.



IV. Closed Curves Represented by Strip Trees

An area is represented by its boundary which is a closed curve. As we travel around the boundary it is

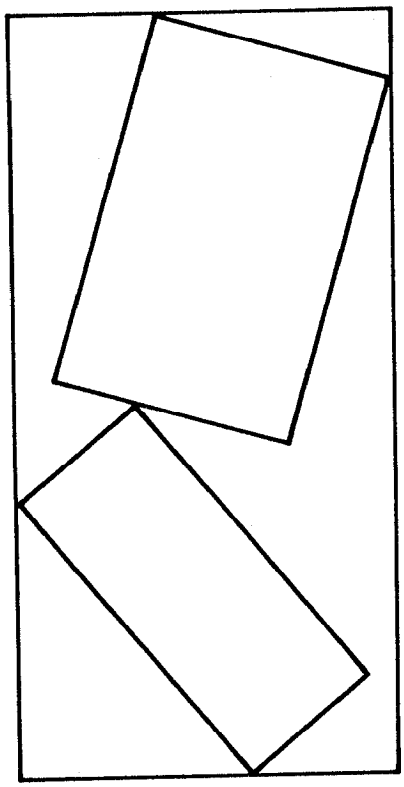


Fig. 10. Construction for Union of Strip Trees Representing Two Curves.

assumed that the interior is to the right. As mentioned before, the digitization method described in Sec. II works for closed curves and, incidentally, also for self-intersecting curves. 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 Sec. III. If a region has holes, it can be represented by a single boundary curve using a construction such as the one shown in Figure 11. If the holes are important, they should be independently represented as strip trees.

By representing an area in this way many useful operations can be carried out within the strip tree formalism, i.e., area computation, intersection between a curve and an area, whether a point is inside an area and intersection between two areas.

IV. A. Area from an Area Strip Tree

Using a method similar to the method for computing the length of a curve from the strip tree, the area at different resolutions can be computed. The standard formula for a polygonal area is used:

$$\text{Area} = \tfrac{1}{2} \sum_{i=0}^{n} (x_i\, y_{i+1} - x_{i+1}\, y_i).$$

where $i+1$ is computed modulo n.

This can be translated into strip notation in a straightforward manner. Suppose we have a set of strips S_k, $k = 0, \ldots, m-1$ for some resolution w^* as defined in Sec. II. A. Then the area contribution of each strip is given by $\frac{1}{2}(x_b y_e - x_e y_b)$. Thus total area at resolution w^* is given by

$$\text{Area}(w^*) = \sum_{k=0}^{m-1} A_k,$$

where A_k is the contribution of the corresponding portion of the area defined by S_k. The recursive procedure for area calculation is as follows:

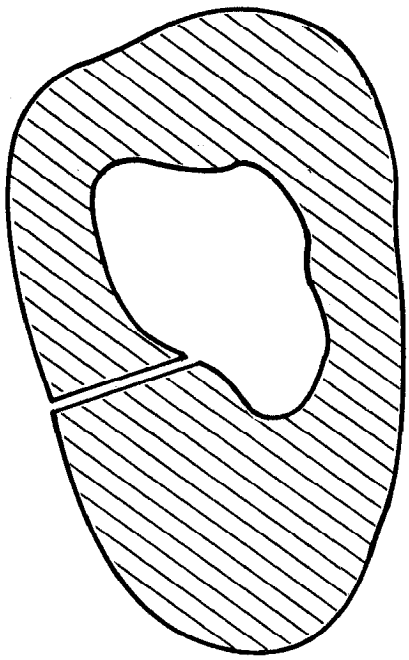
```
Algorithm A7 Area Calculation
real procedure Area(w*,T);
  begin
  if w(T) < w* then
    return (½(xb(T)ye(T) − xe(T)yb(T)))
  else
    return (Area(w*, LSon(T)) + Area (w*, RSon(T)));
  end;
```

Although error bounds could not be placed on the curve length, we can bound the area. The error contribution of each strip is bounded by $w(\|\mathbf{x}_b - \mathbf{x}_e\|)$, so that the total error is bounded by the sum of these individual bounds.

IV. B. Determining Whether a Point is Inside an Area

The strip tree representation of an area by its boundary allows the determination in a straightforward manner of whether a point is inside the area. 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 is a familiar result [9] which is computationally simplified for strip trees in the following manner:

Fig. 11. A Region with a Hole.



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 *clear intersections* of the strip tree with any semi-infinite strip S_0 which has width 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 A8 (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
  Case StripIntersection(S, T) of
    [Null] return (0);
    [Possible] return (NoOfClearIntersections(S, LSon(T))
    + NoOfClearIntersections (S, RSon(T)));
    [Clear] return (1);
  end;
```

Again StripIntersection will report Possible instead of Clear if either of the strips has $C(S) = 0$. A potential difficulty exists with the procedure NoOfClear-Intersections when the strip S_0 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 Figure 8. However, if the strip S_0 passes through

Fig. 12. Indeterminancy of Endpoint Intersections for Inside vs. Outside.

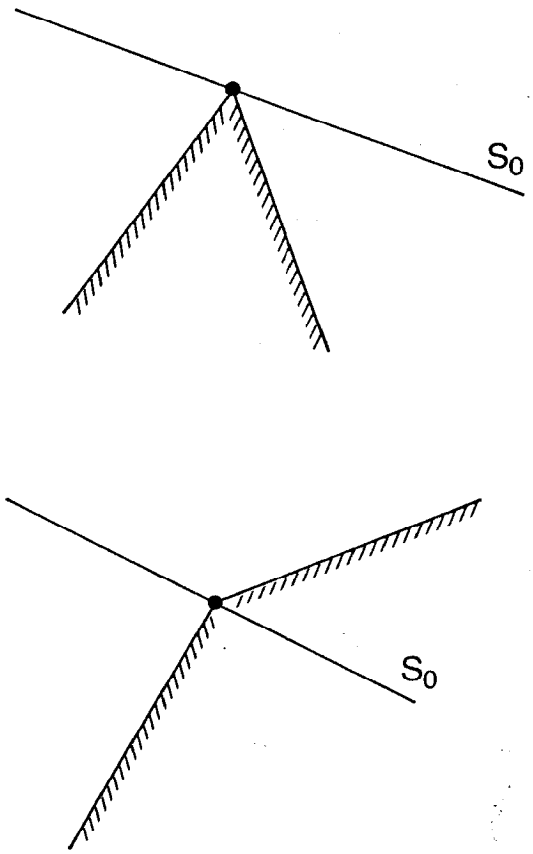
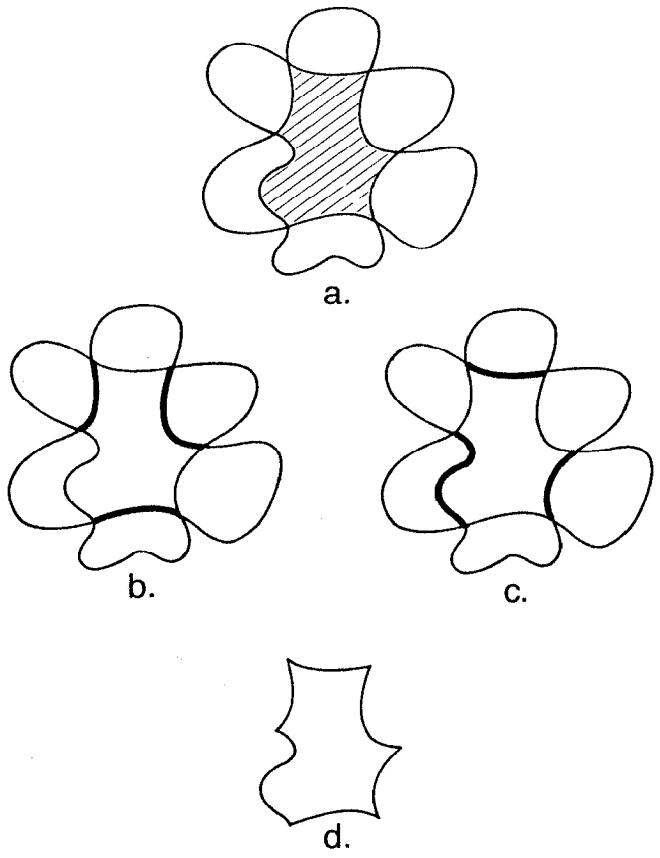


Fig. 13. Decomposition of Area-Area Intersections.



an end point at the lowest level, then there is no easy way of determining the parity of the intersection. Figure 12 shows this amgbiguity. To overcome this difficulty, in practice a different S_0 is used, but for the examples tried so far this problem rarely arises.

IV. C. 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 curve 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:

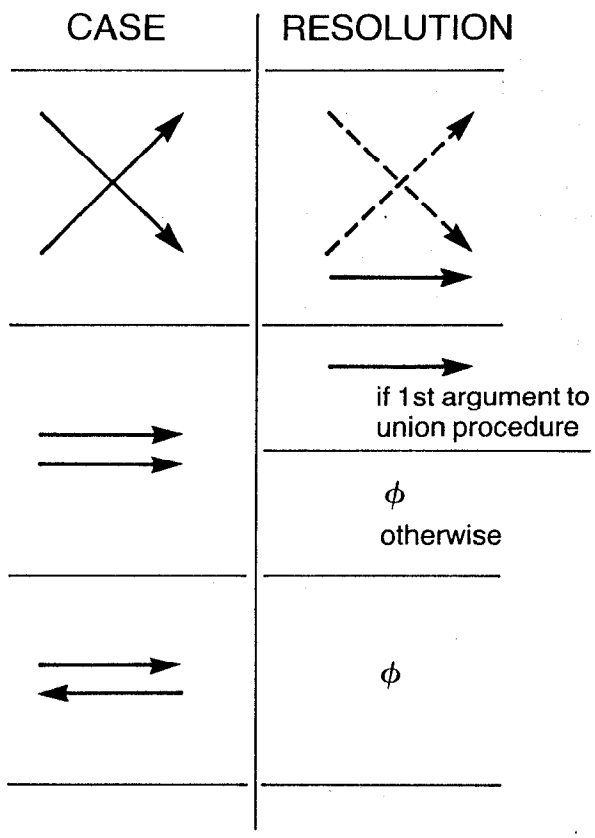
Pruning Property
Consider two strips S_c from T_c and S_a from T_a. If the intersection of S_c with T_a is null, then (1) if any point on S_c is inside T_a, the entire tree whose root strip is S_c is inside or on T_a and (2) if any point on S_c is outside of T_a, then the entire tree whose root strip is S_c is outside of T_a.

This leads to the recursive procedure A9 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 zero to denote that it no longer has the regularity property. Of course, as repeated intersections are carried out with different areas, more and more upper-level strips may have their regularity bits set to zero; nevertheless, the intersected curve 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 strip from the curve is "fatter," i.e., $A(T_1) > A(T_2)$, the node can be copied and the intersection at lower levels resolved, whereas in the converse case the tree must be sequentially pruned by first intersecting the curve strip with the left area strip and then intersecting the resultant pruned tree with the right area strip. The "primitive" case is where both strips are leaves of the tree.

```
Algorithm A9 (Curve-area intersection)
reference procedure CurveAreaInt(T1, T2);
  begin
  T := T2;
  comment T is a global used by CInt;
  return(CInt(T1, T2));
  end;
reference procedure CInt(T1, T2);
```

Fig. 14. Special Cases for Area–Area Intersection.

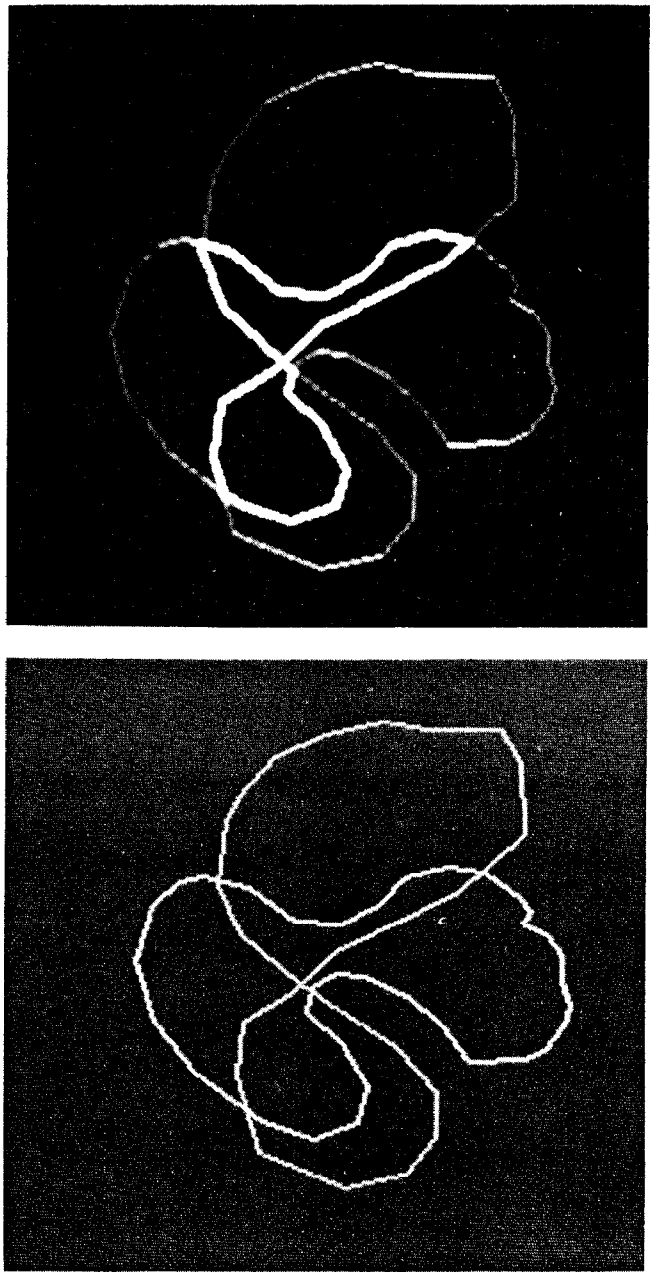



```
begin
Case StripInt(T1, T2) into
  [Null or Primitive]
    if Intersection (T1, T) = null then
      if Inside(T1, T) then return (T1)
      else return (null)
    else return (T1);
  [Clear or Possible] if A(T1) > A(T2) then
    begin
    NT := GetNode;
    C(NT) := 0;
    comment nonregular strip;
    xb(NT) := xb(T1);
    xe(NT) := xe(T1);
    w1(NT) := w1(T1);
    wr(NT) := wr(T1);
    LSon(NT) := CInt (LSon(T1), T2);
    RSon(NT) := CInt (RSon(T1), T2);
    return(NT);
    end
  else comment A(T1) ≤ A(T2)
    return (CInt(CInt(T1, LSon(T2)), RSon(T2)));
end;
```

IV. D. Intersecting Two Areas

The problem of intersecting two areas is simple if we use their strip tree representations. Suprisingly, this problem can be decomposed into two curve–area intersection problems (Figure 13). If the boundary of A_1 is treated as representing a curve instead of representing an area, and its strip tree intersects with the strip tree representing A_2, the lowest level, the result is shown by the thick lines in Fig. 13(b). If the roles of the two strip trees are reversed, the result is given by the thick lines in Fig. 13(c). The union of these two strip trees which represents the curve in Fig. 13(d), is the answer we want! Thus we would like to write the area–area intersection procedure in terms of strips as follows:

Fig. 15. Area–Area Intersection.



Algorithm A10 (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 Sec. III. D. The actual procedure is almost as simple but contains a modified version of CurveAreaInt to handle special cases at the primitive level. Figure 14 shows these cases. The first case is a way

of rounding off at the primitive level. The next two cases cover situations where the boundaries are coincident. Figure 15 shows the result of an area–area intersection using strip trees.

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 it was 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.

IV. E. The Union Operation

The union operation is similar to the intersection operation. For the union of one area strip tree with another, use is made of Algorithm A6. 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 a new strip tree must be defined 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:

$$T = \begin{cases} \text{Union } (T_1, T_2) \text{ if } T_1 \cap T_2 = \varnothing \\ \text{Union (CurveAreaInt}(T_1, T_2), \\ \text{CurveAreaInt}(T_2, T_1)) \text{ otherwise} \end{cases}$$

V. Discussion and Conclusions

For the purposes of simplifying the algorithms, the primitives in the tree were regarded as unit segments on some lowest-resolution grid. In fact, they can be line segments of arbitrary lengths defined with real numbers. The corresponding algorithms are essentially the same but more complicated. This is because in this case, it is often necessary to divide long leaf segments and build new parts of the tree at the lowest levels. The details of these algorithms are discussed in [14]. The advantage of being able to use arbitrary length segments is that any level in the tree can be regarded as a primitive by using its line segment $[\mathbf{x}_b, \mathbf{x}_e)$. Thus all the operations can be carried out at any specified resolution.

Strip trees provide a powerful representation of curves. The representation defines strip segments as primitives to cover subsets of the line. 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. Other attempts in this class have been tried [1, 3, 4, 15], but they do not capture the notions of orientation and resolution nearly so precisely as strips segments, and do not have the union and intersection properties.

A relatively new hierarchical way of representing areas (and curves) is that of quad trees [6, 12]. Quad trees also exploit a hierarchical structure, but must be defined with respect to a digital grid of pixels. This means that, for example, in order to intersect two quad trees, both trees must be defined on the same grid. In contrast, strip trees can be defined in terms of points which are grid-independent, and therefore can be arbitrarily translated and scaled.

Strip trees might seem to require a large overhead in terms of space. If B is the required space to represent a curve, then its strip tree will take about $4B$ space units. However, it must be remembered that since the underlying segments are a linear interpolation of the curve, not all the points on the curve are represented. This savings effectively cancels the overhead of the tree indexing scheme and makes the storage required comparable to raster-oriented representations [8, 11].

Current work is directed towards characterizing the computational complexity of strip tree operations more precisely.

Acknowledgments. The author wishes to thank R. Peet and P. Meeker for their work in the preparation of this document. Thanks also goes to H. Tanaka and D. Weaver for their work in implementing the algorithms in SAIL. Last, but not least, the author wishes to thank the referees, who suffered through early versions of this manuscript and made numerous helpful suggestions.

Received 11/79; revised and accepted 12/80

References

1. Barrow, H.G. Interactive aids for cartography and photo interpretation. Semiannual, Tech. Rep, DARPA Contract DAAG 29-76-C-0057, SRI Int., Nov 1977.
2. Bentley, J.L. Multidimensional search trees used for associative searching. *Comm. ACM 18*, 9 (Sept. 1975), 509–517.
3. Burton, W. Representation of many-sided polygons and polygonal lines for rapid processing. *Comm. ACM 20*, 3 (March 1977), 166–171.
4. Douglas, D.H. and Peucker, T. Algorithms for the reduction of the number of points required to represent a line or its caricature. *Can. Cartogr. 10*, 2 (Dec. 1973).
5. Duda, R.O. and Hart, P.E. *Pattern Classification and Scene Analysis.* Wiley-Interscience, New York, 1973.
6. Dyer, C.R., Rosenfeld, A., and Samet, H. Region representation: Boundary codes from quadtrees. *Comm. ACM 23*, 3 (March 1980), 171–178.
7. Mandelbrot, B.B. *Fractals: Form, Chance and Dimension.* W.H. Freeman, San Francisco, 1977.
8. Merrill, R.D. Representation of contours and regions for efficient computer search. *Comm. ACM 16*, 2 (Feb. 1973), 69–82.
9. Minsky, M.L. and Papert, S. *Perceptrons: An introduction to computational geometry.* MIT Press, Cambridge, MA, 1969.
10. Peucker, T. A theory of the cartographic line. *International Yearbook of Cartography 16*, 1976.
11. Peuquet, D.J. Raster processing: An alternative approach to automated cartographic data handling. *Am. Cartogr. 6*, 2 (April, 1979), 129–139.
12. Samet, H. Region representation: Quadtrees from boundary codes. *Comm. ACM 23*, 3 (March 1980), 163–170.
13. Sloan, K.R., Jr. Maps and map data structures. Forthcoming Tech. Rept., Comptr. Sci Dept, Univ. of Rochester, Rochester, New York, 1981.
14. Tanaka, H. and Ballard, D.H. Extension of strip tree operations. Comptr Sci Dept, Univ. of Rochester, Rochester, New York, 1980.
15. Tanimoto, S. and Pavlidis, T. A hierarchical data structure for picture processing. *Comptr Graphics and Image Processing 4*, 2 (June 1975), 104–119.
16. Turner, K.J. Computer perception of curved objects using a television camera. Ph.D. thesis, Univ. of Edinburgh, Scotland, 1974.