STANFORD RESEARCH INSTITUTE
Menlo Park, California 94025 · U.S.A.

January 1970

AROS

ALGORITHMS FOR PARTITIONING A PICTURE

by

Claude R. Brice

Claude L. Fennema

Stephen A. Weyl

Artificial Intelligence Group

Technical Note 18

SRI Projects 7494 and 8259

I INTRODUCTION

In the region analysis work done by Brice and Fennema (1969), a one-pass algorithm was used to initially partition the picture into named, homogeneous, connected components and to set up the region structure (AROS). This one-pass algorithm is rather complex in nature, and several alternatives have been suggested, one of which (a two-pass algorithm) we have chosen to use for comparison purposes because of its simplicity and apparent efficiency.*

Both algorithms use an L-shaped window to compare the elements of the picture array. If two points differ in properties (only horizontal and vertical comparisons are made), a pair of elementary vectors are placed to separate them (see Figure 1).

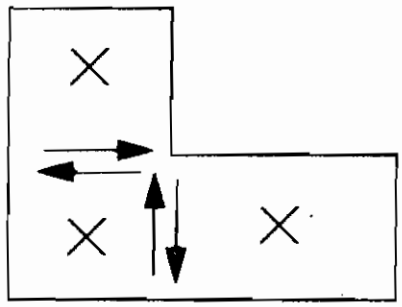

Figure 1

The curves made up of these elementary vectors partition the picture into regions and serve as the boundaries of these regions. The two algorithms discussed here differ in the way they label the picture elements and the boundaries. Section II is a description of the one-pass algorithm, Section III explains the two-pass algorithm, and Section IV is a comparison of their performance.

*This work was sponsored by the Advanced Research Projects Agency and the Rome Air Development Center under Contract F30602-69-C-0056 and by the Advanced Research Projects Agency and the National Aeronautics and Space Administration under Contract NAS12-2221.

II THE ONE-PASS ALGORITHM

A. Background

The structure on which both of these algorithms operate is made up of two arrays: a picture array T1 and a component array T2. Each word of the picture array is two picture elements, each of which is divided into the bit fields of Figure 2. Here S is a special bit

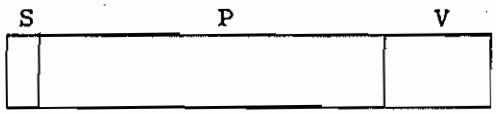

Figure 2

to denote curve headers; P is a field that contains the grayscale of (i,j) when S = 0 or a pointer to a T2 word when S = 1. V contains the vector code of the elementary boundary vectors that exist near the point (i,j) (see Figure 3).

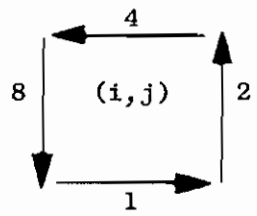

Figure 3

The component array T2 is an array of words (called component words) having the format of Figure 4. One of these words is assigned to each

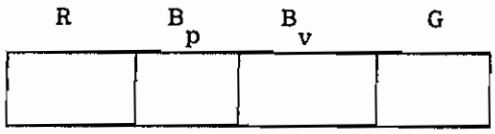

Figure 4

curve header (picture element with S = 1). The R field contains the name (in code form) of the region to which the point belongs; the B_p and B_v contain the point and vector that begin the boundary component tagged by this word. (B_p is, of course, the pointer to the picture element that points to this word.) Lastly, G is the grayscale of the point B_p.

The boundary of the region R1 is then extracted by scanning T2 for all the words with Code R1 in their R fields, going to the beginning vector (point) and following the boundary using the SUCCESSOR function, which goes from a vector VI to the next leftmost vector V2, as in Figure 5.

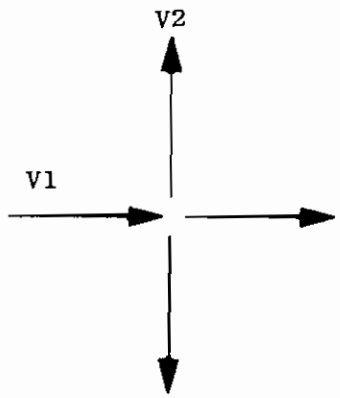

Figure 5

B. Some Notions

To simplify the presentation of the algorithm, we shall make some conventions in terminology.

First we will always by convention denote the X field of a word (or half word) Y by X(Y). For example, V(A) is the V field of the point A. G(T2(P(PT))) is the G field of the element of T2 pointed at by the P field of PT, and thus it contains the grayscale of PT. Next, suppose PT = (i,j) is a picture point (half word) when we define the grayscale function

$$GS(PT) = \begin{cases} P(PT) \text{ if } S(PT) = 0 \\ G(T2(P(PT))) \text{ if } S(PT) = 1 \end{cases} .$$

Because an elementary vector is defined by a point PT and a vector code, v, we will denote this vector (PT,v). Lastly, remembering that SUCCESSOR (SUCC) is the name of the function used in following the boundary and PREDECESSOR (PRED) is its inverse, we define 3 functions:

$$R_p(PT,X) = \begin{cases} R(T2(P(PT))) \text{ if } S(PT) = 1 \\ R_p((PRED(PT,X))) \text{ if } S(PT) = 0 \end{cases}$$

$$R_s(PT,X) = \begin{cases} R(T2(P(PT))) \text{ if } S(PT) = 1 \\ R_s((SUCC\ (PT,X))) \text{ if } S(PT) = 0 \end{cases}$$

$$RG(PT) = \begin{cases} R_p(PT,8) \text{ if } (AND\ PT,8) = 1 \\ RG\ (LFT(PT)) \text{ otherwise} \end{cases}$$

where LFT(PT) is the point to the left of PT.

C. The Algorithm

The algorithm is best explained by the flowchart of Figure 6 with the following notes: Box A (and V(B),8), is a check for the presence of bit 8 in the V field of B, which means, since $GS(A) \neq GS(B)$, that the configuration in Figure 7 has been encountered.

This means (as far as we know at this point) that B is a point of a new region.

Box B tests if the predecessor of (A,1) is (D,2). If so, we have the configuration of Figure 8, meaning that we have encountered a new curve for the boundary of the region containing A.

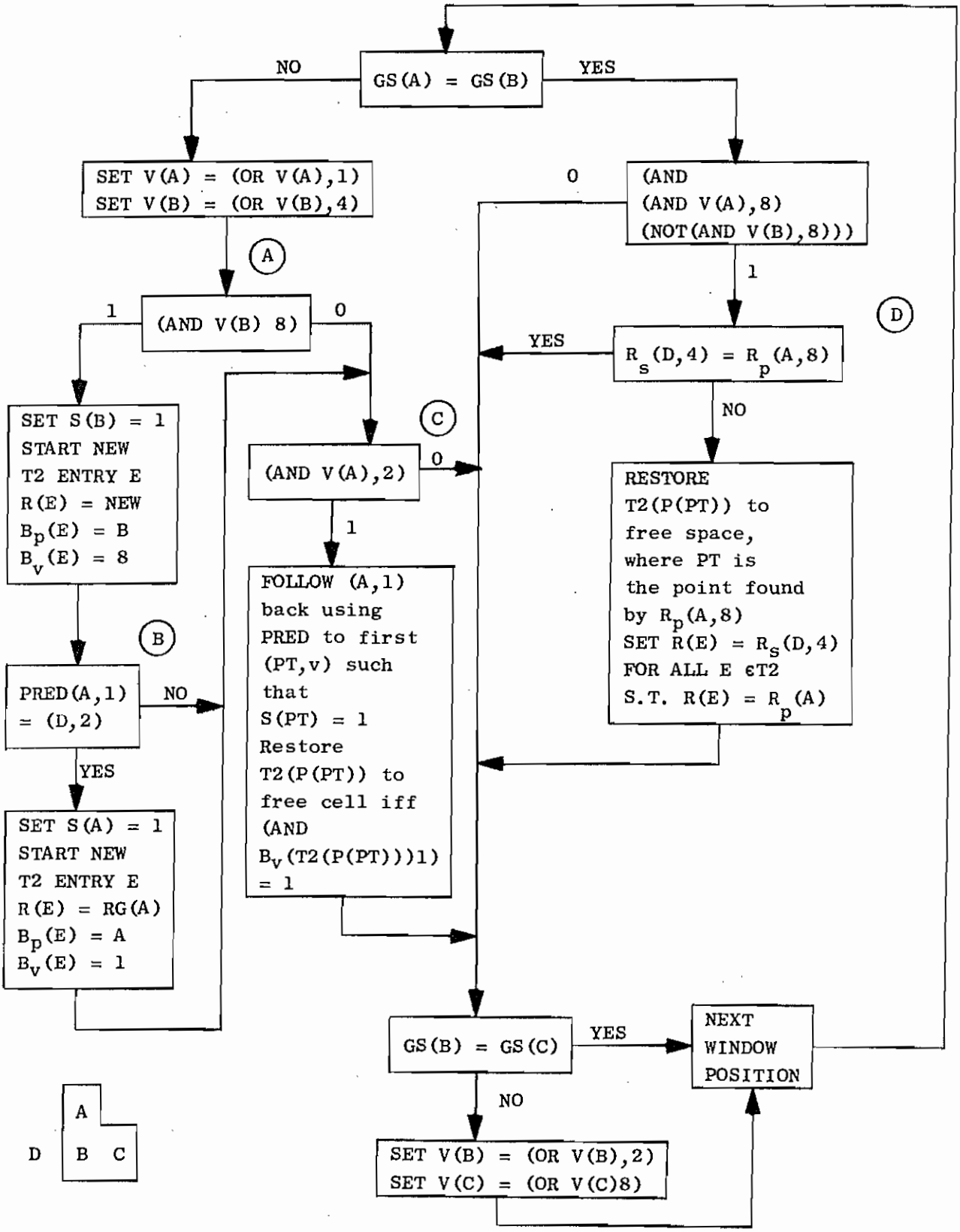
GS(A) = GS(B)
NO
YES
SET V(A) = (OR V(A),1)
SET V(B) = (OR V(B),4)
A
(AND V(B) 8)
1
0
SET S(B) = 1
START NEW
T2 ENTRY E
R(E) = NEW
B_p(E) = B
B_v(E) = 8
B
PRED(A,1) = (D,2)
NO
YES
SET S(A) = 1
START NEW
T2 ENTRY E
R(E) = RG(A)
B_p(E) = A
B_v(E) = 1
C
(AND V(A),2)
0
1
FOLLOW (A,1)
back using
PRED to first
(PT,v) such
that
S(PT) = 1
Restore
T2(P(PT)) to
free cell iff
(AND
B_v(T2(P(PT)))1)
= 1
0
(AND
(AND V(A),8)
(NOT(AND V(B),8)))
1
D
YES
R_s(D,4) = R_p(A,8)
NO
RESTORE
T2(P(PT)) to
free space,
where PT is
the point found
by R_p(A,8)
SET R(E) = R_s(D,4)
FOR ALL E ϵT2
S.T. R(E) = R_p(A)
GS(B) = GS(C)
YES
NEXT
WINDOW
POSITION
NO
SET V(B) = (OR V(B),2)
SET V(C) = (OR V(C)8)
D
A
B
C

Figure 6

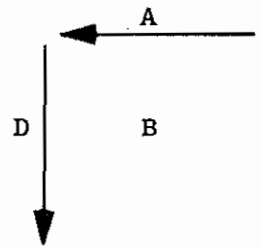

Figure 7

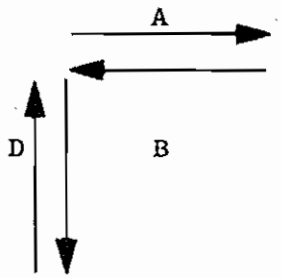

Figure 8

The other tests are of a similar nature and deal with the connection of two partial curves (boxes C and D) and the merging of two partial, expanded regions (box D). This procedure is executed in a raster-like fashion until the entire picture has been processed.

III THE TWO-PASS ALGORITHM

The basic data structure remains the same as for the one-pass algorithm, but a bit table of four bits per picture point is added for marking purposes, one for each possible vector. During the first pass the separating vectors are created, and during the second pass the regions and components are named.

A. First Pass

The first pass is merely to create and store in the array, in the proper field, the elementary vectors of each boundary, using the

same L-shaped window as the one-pass algorithm, but without naming regions and components. During this pass each picture point is visited twice.

B. Second Pass

The algorithm proceeds from left to right, following a line until it finds an unmarked vector. (The bit table is used to mark each possible separating vector.) When an unmarked vector is found, the algorithm marks it, creates a special pointer that indicates the beginning of a new component, and makes a new entry in T2 similar to that used in the one-pass algorithm. Then, starting from this vector, the boundary is followed and every vector is marked until a marked vector is found (obviously the starting vector). When following the boundary, the algorithm sets up a counter to which -1 is added when a left turn occurs and +1 is added when a right turn occurs. At the end of the following procedure, if the result of the counter is negative, the boundary is counterclockwise-oriented; it is clockwise-oriented if the result is positive. A clockwise boundary means that there is a hole in one region, and therefore there is a region with more than one component (see Figure 9).

If the result of the counter at this end of the following procedure is positive, the name of the region to which this component belongs is found by stepping to the left along one line until a marked vector is found and by following this boundary to its beginning. Otherwise, a new region name is created and stored in T2, and pointers are set up in the same manner as for the one-pass algorithm. During this second pass every point is visited at least once.

IV COMPARISON OF THE TWO ALGORITHMS

The two-pass algorithm has the obvious advantage of simplicity, and the one-pass one may be faster in execution time, but it seems

pretty difficult to estimate what would be the running time of the two algorithms without coding and executing them. In order to get a rough approximation of these times, we decided to simulate the action of both algorithms on an illustrative picture that includes most of the problems met in a typical partition (see Figure 9).

A. Common Operations

The basic operations involved in both algorithms are: visiting each picture element, comparing its value with the value of the adjacent element in the L-shaped window, and putting in the elementary separating vector if needed. These operations are the same for both algorithms and therefore have no influence on the comparison.

B. Differences

Besides these common operations, the one-pass algorithm involves mostly the following operations:

Follow: Using the SUCCESSOR or PREDECESSOR function, the program, during this procedure, follows the boundary until a special pointer is encountered. The cost of this operation is measured in the number of elementary vectors (arrows). The basic operation, consisting of passing from one vector to its successor, has been estimated as 30 μs. (A sample program has been written.)

Arrow test: During the course of the execution, the program has to test for a certain arrow configuration. The basic instruction is to skip on mask bits. The cost has been evaluated to 5 μs.

In the same condition, the two-pass algorithm involves, besides the operations already mentioned, the following operations:

Follow (with orientation): The boundary is followed, and a counter is altered by -1 or +1, depending on whether the boundary turns to the right or to the left. The cost of this operation is a little larger than the usual follow; it has been estimated to be 36 μs.

Marking: While the above operation is performed, each unmarked vector is marked. The basic operation includes a test and an access to the bit table; its cost has been estimated to be 10 μs.

Scanning: During the second pass, the program scans the picture array, looking for an unmarked vector. The cost of the basic scan from one picture element to another has been estimated to be 10 μs.

With these definitions and estimations, the performance of the two algorithms for the same picture (Figure 9) is compared on Table I.

V CONCLUSION

As far as our estimations are correct, it seems that the one-pass algorithm works twice as fast as the two-pass algorithm. However, given the same picture, the two-pass algorithm depends only on the number of curves, but the one-pass algorithm depends on the nature of the curves. The worst case for the one-pass algorithm would be the case where there are a lot of 45° lines (with respect to the horizontal). This, however, is not a problem in a real environment.

A decision has been made to implement the one-pass algorithm for two reasons: It seems faster, and it consumes less space since it needs no bit table.

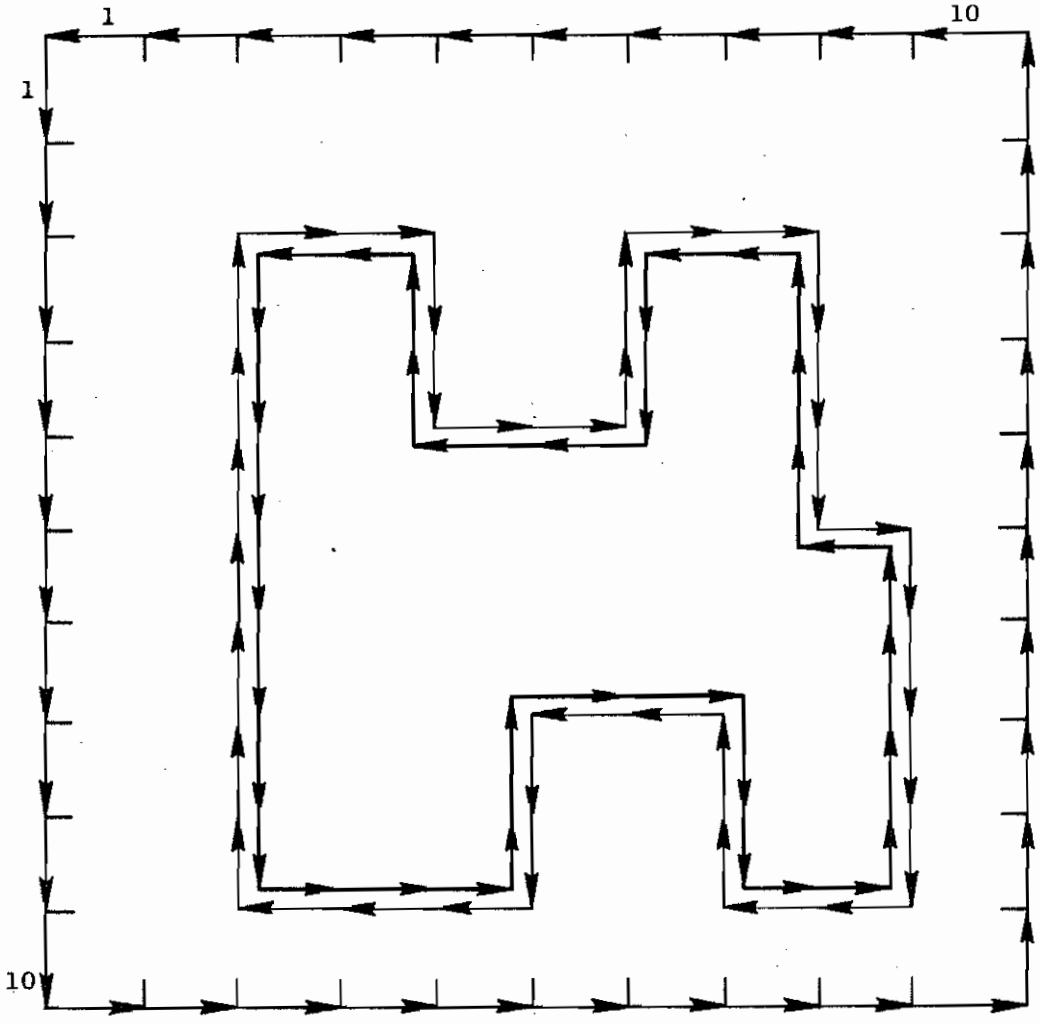

Figure 9

Table I

Operations	Time (μs)	A One-Pass	B Two-Pass	Difference A-B
Follow	30	117 X 30 = 3510	38 X 30 = 1140	2370
Arrow test	5	200 X 5 = 1000	0	1000
Follow (with orientation)	36	0	112 X 36 = 4032	-4032
Marking	10	0	112 X 10 = 1120	-1120
Scanning	10	0	200 X 10 = 2000	-2000
TOTAL (μs)		4510	8292	-3782