258

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence Project
Memo Number Vision [redacted] 111

Memorandum MAC-M-332
October 1966

Summer Vision Programs

Leslie Lamport

We assume that we are given a square array that describes a scene. The name of the array will be "array." The number of points representing the side length of the array will be called "pts." (I.e., $(pts)^2$ is the total number of entries in the array.)

For nefarious purposes, the array will be considered to be split into square boxes. The number of such boxes on a side of the array will be called "boxes." (Thus, there are $(boxes)^2$ number of boxes whose union contains all the points of the array.)

The following programs are designed to use color pictures of the scene, with color predicates, to produce output for use with Guzman's program DT.

The programs are listed in some sort of semi-logical order. The function name is given, then the arguments, output, other functions called by it, and a brief description of what it goes through to produce the output.

<u>PREDT</u>

<u>Arguments</u>: array, pts, boxes

<u>Output</u>: (atom1, atom2, atom3, ... atom n)

where atom i is a generated atom which has on its property list entries under the indicators SHAPE and NEIGHBOR. n is the total number of regions found by EXEC2. To find out the significance of the entries on the property list of the atoms, see the programs SHAPEALL (for the entries with indicator SHAPE) and NEIGHBORS (for the entries with indicator NEIGHBOR).

Functions Called: EXEC2, BOUNDIT, NEARREGIONS, ATOMGEN, SHAPEALL, NEIGHBORS.

Method: The PREDT function is a trivial top-level function to tie together the operation of the functions which it calls.

EXEC2

Arguments: array, pts, boxes

Output: (boxlist, regionsno, regionlist)

where boxlist = (box0.0, box1.0, box2.0, ... , boxn.n)

with n = boxes

box i.j = (regdesc1, regdesc2, ..., regdesc k)

regdesc α = (region a, ptsinbox, FRONTIER
or
INTERIOR)

where regions a is a pointer to an element of the list regionlist (see below),

ptsinbox = number of points of the region which are in box i.j.

FRONTIER means that the region does not completely fill box i.j.

INTERIOR means that it does.

k = total number of regions which have points lying inside box i.j.

regionsno = total number of regions found by the program.

regionlist = (region1, region2, ..., region k)

where k = regionsno

region i = (name i, COLORPRED, inpts, bdrypts, bdry)

name i = (REGION, i)

inpts = total number of points in region i.

bdrypts = number of points on the boundary of region.

bdry = output of Sussman's program SORTBND1.

Functions Called: INITLIST, GETNAME, CHOOSEP, FINDREGION.

Method: The purpose of this EXEC2 is to control REGIONS1. It uses REGIONS1 to find the regions using the predicated COLORPRED. It uses CHOOSEP to choose the next box in which to set REGIONS1 to work. It then calls FINDREGION to do all the real work. The function prints the total number of regions found. It produces as output a list of regions, together with a list of boxes (of length $(\text{boxes})^2$). The list of boxes has, for each element, at list of the regions lying at least partially within that box together with some miscellaneous information about the region. The entry for each region is a list containing various stuff, most important being the sorted boundary of the region.

For the rest of this description, we give the name "rlist" to either the output of EXEC2, or to the list of the same form which is kept by EXEC2 and built up as it goes along, eventually being returned as its value.

INITLIST

Arguments: boxes

Output: rlist

Functions Called: EXPT

Method: A trivial routine for initializing rlist.

EXPT

Arguments: number exponent

Output: $(\text{number})^{\text{exponent}}$

Functions Called: None

Method: A trivial exponentiation hack that is moby slow and should be done away with, since it is never called with an exponent of other than 2.

GETNAME

Arguments: rlist

Output: (REGION, n)

where n is one plus the value of regionsno (regionsno = CADR (rlist))

Functions Called: None

Method: Triviality. The name of the region (CAR (region)) is used only by REGDISP, and is expendable if that function (one of the display functions) is changed.

CHOOSEP

Arguments: predlist, rlist, boxes

Output: ((i, j), COLORPRED)

where (i, j) is the next box in which to try to find a region.

Method: CAR (predlist) = NIL initially, the first time called, in which case it returns ((0,0), COLORPRED).

At other times, if it is give ((i, j), COLORPRED) as its input, then it returns ((H, k), COLORPRED) where (h, k) is the next box after (i, j) which does not have any region lying within it (as determined by rlist). It is a vestigal routine from EXEC1, where it was separated to allow for some clever choosing of predicates and boxes wherein to try them.

FINDREGION

Arguments: name, box, COLORPRED, array, pts, boxes, rlist

where name = output of GETNAME

box = (i, j)

= car (output of CHOOSEP)

Output: region = an element of CADDR (rlist) or

NIL if no region or too small a region was found, or

TOOBIG if a region which was too big was found.

Functions Called: MIDPOINT, REJECT, ADDREGLIST, plus the following functions for which Sussman is responsible: INITCOL, REGIONS1, SORTBND1, ARESET.

<u>Method</u>: It calls MIDPOINT to get the midpoint of box (note that it only tries REGIONS1 with that single point, and doesn't elsewhere in the box if no region is found there), calls INITCOL with that point to initialize COLORPRED and then calls REGIONS1 with that point as its starting point, COLORPRED as its predicate, array as the array it is to work with. (<u>Thus, REGIONS1 is given the entire array to handle</u>.) It then calls REJECT to decide if the region returned by REGIONS1 is to be accepted. (Note: REGIONS1 leaves the array munged [sic] and the array must be ARESET after calling it. REJECT does this if it rejects that region.) If so, it calls SORTBND1 to sort the boundary. (Due to a Sussman idiocy, it SETQ's XMAX and YMAX to the value of pts, because SORTBND1 uses them. This probably has to be remembered if the business is to be compiled.)

It then calls ADDREGLIST to put a pointer to the region on each of the appropriate elements of boxlist (= CAR (rlist)). Finally, it puts the region onto the end of regionlist (= CADDR (rlist)).

Obviously, if FINDREGION is given a predicate other than COLORPRED, and the call to INITCOL is removed, it behaves as it does in EXEC1, calling REGIONS1 with that predicate and putting the name of the predicate (rather than COLORPRED) as the second item of region.

If a region is rejected by REJECT, FINDREGION prints out (REJECT (i, j) why) where (i, j) is the box in which the region found was rejected, and why = NIL if it was too small (or non-existent) and = TOOBIG if it was too large. This can be used to adjust the tolerance of the color predicate. Presumable if it finds something too large, it means that the tolerance must be lessened, if it rejects too many regions, it means the tolerance is too small. (Since every point belongs to some region, there should be no rejections with a reasonable scene containing reasonably large items. However, noise may cause some micro-small regions to appear. This shouldn't harm things because the regions should be large compared to the size of a box, and should contain the midpoints of a reasonable number of boxes—in fact should fully contain a reasonable number of boxes (like 3) ???)

<u>MIDPOINT</u>

<u>Arguments</u>: box increment

where box = (i, j)

and increment = QUOTIENT (pts, boxes)

<u>Output</u>: (x, y) where x and y are the coordinates of the midpoints of box (i, j)

<u>Functions Called</u>: None

<u>Method</u>: Trivial

<u>REJECT</u>

(N.B.: Routines may require tuning to work on real data.)

<u>Arguments</u>: regl, array, increment, pts

where regl is the output from REGIONS1, and increment is the same as for MIDPOINT

<u>Output</u>: (NIL) if the region is accepted

NIL if the region is rejected for being too small

TOOBIG if the region is rejected for being too big

<u>Functions Called</u>: ARESET

<u>Method</u>: The theory is that the region is accepted if its boundary is big enough. If the boundary is small, that indicates that either the region itself is small, or that the complement of the region is small. The numbers that are in there at the present time to decide if the boundary is too small are just hacks, and probably need modification. Similarly the numbers to decide if the rejected region is too big, although that isn't a difficult decision. After deciding to reject, the array must be ARESET. ARESET then returns the total number of points in the region to allow the decision to be made.

<u>ADDREGLIST</u>

<u>Arguments</u>: i, j, boxes, rlist, regdesc

where (i, j) are the coordinates of a box, and

regdesc is an element of the form of a member of an element of CAR (rlist).

<u>Output</u>: some randomness, not interesting because the function is called to effect the construction of rlist.

<u>Functions Called</u>: None

Method: By various RPLACAs, RPLACDs and NCONCs, it puts a pointer to regdesc into the entry for box (i, j) in CAR (rlist) (= boxlist).

The functions INITDISPLAY and REGDISP are display functions to put the output of EXEC2 into comprehensible form on the scope. To use them, first call INITDISPLAY to initialize the necessary scope arrays. Then call REGDISP to display any desired region.

INITDISPLAY

Arguments: pts, boxes

Output: A message indicating the number of words of binary program space used by the arrays.

Method: Trivial. Note that the two DISINIs are superfluous and can be removed without changing matters. The function need only be called once (unless pts or boxes are increased).

REGDISP

Arguments: number, rlist, pts, boxes

Output: (REGION, number) if there is a region numbered number

(NO, SUCH, REGION) if there is no region numbered number in rlist

Functions Called: DISBOXES, DISBDRYS

Method: The scope display that it sets up contains the boundary points of the region, plus the boundaries of all the boxes in which the region is found. Those boxes which are completely filled up by the region are marked by a diagonal slash through the box.

DISBOXES and DISBDRYS do the work in actually setting up the display, Note that this function kills any LISP display that may have been going on when it was called.

DISBOXES

Arguments: pts, boxes, rlist, regionlist

where regionlist is a list which is of the same form as CADDR (rlist) (i.e., is a list of regions).

Output: number of entries stored in the array BOXARRAY.

Functions Called: DISPLAYBOX

Method: It is given a list of regions, and it displays the boxes wherein those regions are found.

DISPLAYBOX

Arguments: i, j, boxes, pts, mode, count

where (i, j) = the box to be drawn

mode = INTERIOR or something else

count = the location in array BOXARRAY into which display information is to be stored.

Output: newcount = one plus the last location in BOXARRAY into which display information was stored.

Functions Called: None

Method: It stores display information to display box (i, j) on the scope—with a diagonal slash if mode = INTERIOR. It uses count to tell it where to start storing in the array, and newcount should be the argument count when the function is next called.

DISBDRYS

Arguments: regionlist, pts, arraysize

where regionlist is the same as for DISBOXES and

arraysize = size of the one-dimensional array BDRYARRAY.

Output: Some randomness containing the number of entries in array BDRYARRAY if something is displayed, NIL if there is too much to display.

Functions Called: None

Method: The size of BDRYARRAY is set by INITDISPLAY to some reasonable size, but not large enough for any pathological case. Hence, if the display called for is too big, it prints (BOUNDARY TOO BIG) and does nothing. Otherwise, it displays the boundarys of each of the regions in regionlist.

BOUNDIT

(Note that BOUNDIT mungs [sic] rlist and converts it to brlist—it doesn't copy it.)

Arguments: rlist, pts, boxes

<u>Output</u>: brlist

where brlist is the same as rlist except that the entry region i now equals

(name i, whitebdry, COLORPRED, inpts; bdrypts, bdry)

where whitebedry = list of elements each of which is the output of WHITE

(c.f. output of EXEC2 and output of WHITE)

<u>Functions Called</u>: LSORT, WHITE

<u>Method</u>: This function is used to call WHITE in order to produce a line segemented approximation to the boundary, and enter into the moby rlist. It does this for each region, putting the list whose elements are the output of WHITE as the second element of the list for the region by appropriate RPLACD and RPLACAing. Segments of the sorted boundary which are too small (by some mystic calculation that needs tuning) are not sent to WHITE, but are forgotten. (SOERBND1 produces a list of list of points, each of which is one connected segment of the boundary of the region --presumably one outside boundary and n-1 inside boundaries. Noise could easily generate some very small inside boundaries which should be ignored.)

<u>LSORT</u>

<u>Arguments</u>: list

<u>Output</u>: a list whose elements are the same as those of list, except sorted according to their length, longest first.

Functions Called: DELI

Method: Trivial bit of recursive cleverness, finding the longest element of the list, calling DELI to remove that element from the list and CONSing that element to LSORT of the DELI list.

It is used to sort the boundary of the region so that the boundary segments are in decreasing order of length. In reasonable objects, this means that the outside boundary will be the first on the list. (????????)

WHITE

Arguments: a list of points, where a point is a list of two numbers.

Value: (segment 1, segment 2, ... segment n)

where segment i = (0, pt 1, pt 2)

or (b ± 1, pt 1, pt 2, ... pt m)

where pt = (x,y,α,d) Note: not d^2.

Functions Called: ????

Method: This function has not yet been written. However, all the pieces of it have been. The segments represent a segmentation of the boundary into straight and curved line segments.

CAR (segment) = 0 indicates a straight line segment

= ± 1 indicates a segment curved in or out.

The boundary must be ordered in a clockwise direction. If the segment is a curved segment, the points are a polygonal approximation to the curve. α and d have the meanings assigned to them by J. White in his various documents.

<u>**NEARREGIONS:**</u>

<u>Arguments</u>: brlist (c.f. output of BOUNDIT)

<u>Value</u>: nrlist where nrlist is of the same from as rlist except that the entry for region i now looks like this:

(name i, nbrlist, whitebdry, COLORPRED, intputs, bdrypts, bdry) where nbrlist = list of regions which are near to region i in the sense described under method.

<u>Functions Called</u>: none

<u>Method</u>: Two regions are defined to be near if they both contain points in some box. (I.e., if they are both members of some element of CAR (rlist).) The property of nearness is defined to be non-commutative in the sense that if region a is near region b (i.e., region a is a member of the nbrlist entry in the region b entry) then region b is <u>not</u> near to region a. This is useful in function NEIGHBORS, and has the nice property that because of the way the function goes about things, no circular list structure is created (a serendipitous effect). Note that the input is munged, and not copied over.

<u>**ATOMGEN:**</u>

<u>Arguments</u>: nrlist

<u>Value</u>: (anrlist, atomlist)

where anrlist is the same as rlist except that the entry for region i now looks like:

(genatom i, name i, nbrlist, whitebdry, COLORPRED, intpts, bdrypts, bdry) where genatom i = a generated atom

and atomlist = (genatom n, genatom n-1, ..., genatom 1) where n = number of regions (CADR nrlist).

Functions Called: none

Method: This function merely generates atoms, and puts a generated atom onto the front of the region list for each region on the rlist. It is done because Guzman's DT wants the regions as atoms with various things on their property lists. Again, it mungs the input list rather than copying it.

SHAPEALL

Arguments: anrlist (c.f. value of ATOMGEN)

Value: NIL

Functions Called: SHAPE

Method: This function calls SHAPE for each region, and puts the value of SHAPE on the property list of the generated atom of that region with the indicator SHAPE.

SHAPE

Arguments: whitebdry (c.f. value of BOUNDIT)

Output: At present, same as its argument.

Functions Called: ???????

Method: This function has not been really written, just a dummy whose output is the same as its input. Something must be written which will look at the boundry and decide what the shape of the region is. The input, whitebdry, is a list of elements each of which is the output of WHITE. (I.e., each of which is a polygonal approximation to either the outside, or one of the inside boundaries of the region.) There is no way of being sure that this list is in the right order, though the use of LSORT in BOUNDIT should help. It's value must be something that will be of use by Guzman's DT.

NEIGHBORS:

Arguments: anrlist

Value: NIL

Functions Called: NBRRGN

Method: This function is used to put a list of atoms on the property list of one of the generated atoms with the indicator NEIGHBOR. This list contains the generated atoms of all regions which are neighbors to the region of which the aforementioned generated atom is. (This doesn't make much sense, but it's fairly obvious what it all means.) The decision about whether or not two regions are neighboring is made by NBRRGN.

NBRRGN

Arguments: whitebdry 1, whitebdry 2

where whitebdry i = list of elements, each of which is of the form of the output of WHITE

(c.f. BOUNDIT)

Value: T or NIL, depending upon whether or not the two boundries are neighboring.

Functions Called: STRNBR, CURVNBR

Method: whitebdry i is a list of lists of segments. Each list of segments is the boudry of some closed curve, reduced to straight line approximations. Call such a list by the name "bdry". Each element of bdry is a list of pts (c.f. WHITE) with the first element of the list 0 or ±1. 0 indicates that it is a straight line segment. ±1 indicates that it is a curved segment. For each segment of whitebdry 1, either STRNBR or CURVNBR is called with each bdry of whitebdry 2, depending upon whether the segment is straight or curved. If that function returns T, then the value of NBRRGN is T. Otherwise, the checking continues. Thus for two regions to be neighbors, one of the segments of the first must be a neighbor of one of the boundrys of the second.

STRNBR

Arguments: pair, bdry

where pair = (pt 1, pt 2) and

pt i = (x, y, α, d) c.f. WHITE

and bdry = (segment 1, segment 2, ... segment n)

where segment i = CONS (0, pair)

or (±1, pt 1, pt 2, ..., pt i)

Value: T or NIL depending upon whether or not the line segment represented by pair is close to being the same line represented by one of the straight line segments of bdry.

Functions Called: LINE

Method: The equation of the line determined by the points of pair is computed in the form $aX + bY - c = 0$, normalized so that $a^2 + b^2 = 1$. Then, plugging the values of x and y for a point into the left-hand side of the equation gives as its value the distance of that point from the line. A straight line segment of bdry is considered to be close to the straight line determined by pair if the distance of each of its two points from that line is less than XYERROR.

Thus, XYERROR is a parameter to be adjusted to determine how things should work. Also, if necessary, some test could be made to assure that the two line segments were actually close to one another, and not just two random line segments that happened to belong to almost the same line. This didn't seem necessary in a first effort.

LINE

Arguments: x,y,a,b,c

Value: $ax + by + c$

Functions Called: none

Method: trivial

CURVNBR

Arguments: segment, bdry

where segment = (±1, pt 1, pt 2, ... pt n) and pt i, and bdry same as for STRNBR.

__Value__: T or NIL depending upon whether or not segment is close to one of the segments of bdry.

__Functions Called__: INBDRY

__Method__: Since the sign of the CAR of a segment indicates whether that segment curves in or out, for two segments of different boundries to be healthy type neighbors, the sum of their CARs should be 0. Hence, for each segment of bdry, CURVNBR checks whether nor not their CARs sum to zero. If so, it calls INBDRY to determine if they are actually close. If so, it returns T, if not it keeps trying and returns NIL if it fails. Note that by not having INBDRY as part of itself, it causes some wasted effort (notably, calling CONVSEG). However, since bdry should usually be a list of length 1, it doesn't matter too much.

__INBDRY__:

__Arguments__: segment 1, segment 2

where segment i = CDR (segment i) (c.f. CURVNBR)

__Value__: T or NIL, depending upon whether or not the two segments are close to each other.

__Functions Called__: CONVSEG

---

* Note: This makes use of the fact that all the boundries are oriented in the same way (e.g., clockwise).

<u>Method</u>: The method is a not too satisfactory hack. It looks to see if any straight line segment (that joining two successive points) of egment 1 is close to any one of egment 2.

The criterion for being close is that (1) the angle of the slopes should be the same to within ANGERR radians, and that the distance between the first point of the straight segment of egment 2 and the second point of the straight segment of egment 1 are such that the sum of the absolute value of the differences of their coordiates (i.e., $|\Delta x| + |\Delta y|$) is less than PTERROR.

(For the first and second point jazz, draw two neighboring polygons both oriented clockwise to see why).

The coordinate sum, rather than distance function is a meaningless hack to simplify things. CONVSEG is used to compute the angles of slope of the line segments of egment 1. Those of egment 2 are computed as they are tested.

<u>CONVSEG</u>

<u>Arguments</u>: (pt1, pt2, ...pt n) (i.e., one single argument)

where pt i, = (x, y, α_i, d)

and α_i = the interior angle of the polygon determined by the n points at point i if $i \neq 1$.

α_1 = angle between the line determined by pt 1 and pt 2 and the x-axis.

Value: same as argument, except that α_i is converted to the angle between the x-axis and the line determined by points i and i +1.

Functions Called: none

Method: Obvious recursive hack, using some plane geometry. It is called by INDBDRY for the obvious reason.

SEXPRINT

Arguments: atom, number

Value: Randomness

Functions Called: GRINDEF

Method: Uses GRINDEF to write out a file named "SEXEC atom" on tape numbered number, containing all the functions mentioned on pages 1 - 15 of this document. It does the necessary UWRITEing and UFILEing.

EXEC1

Arguments: array, plist, pts, boxes

where array, pts, boxes are as in EXEC1, and plist is a list of predicate names, a predicate being defined as required as input for Sussman's REGIONS1.

Value: Same as for EXEC2, except that "bdry" is the unsorted boundry as produced by REGIONS1: i.e., simply a list of points.

Functions Called: Same as for EXEC2

Method: Identical to EXEC2 except that instead of working with the single predicate COLORPRED, it uses the list of predicates given to it as its second argument. The only difference in the workings of EXEC1 as compared to EXEC2 is in the function CHOOSEP. CHOOSEP of EXEC1 is the same as for EXEC2, except that in the latter case, it does only a trivial operation.

Also, FINDREGION does not have the function of calling INITCOL. Otherwise, the two functions are identical. Note: EXEC1 does not print rejected regions, and if it finds a region which is too big, it stops looking with that predicate.

<u>CHOOSEP</u>

<u>Arguments</u>: predlist, rlist, boxes

where predlist = (NIL or (i,j), pred 1, pred 2, ... pred n)

where pred i is a predicate.

<u>Value</u>: ((i.j), pred, pred, ..., pred) or NIL

(i,j) not necessarily same as in argument

and "preds" are not the same

<u>Functions Called</u>: None

<u>Method</u>: If CAR (predlist) = NIL, value is

CONS((0,0) CDR (predlist)).

If CAR (predlist) =(i,j), it searches all the boxes after (i,j) looking for the first one in which there is no region found using predicate CADR (predlist). If that box is (n,m), it returns CHOOSEP (CONS (NIL, CDDR(predlist)))

If predlist = (NIL), value = NIL.

As mentioned before, this is a trivial method, just stepping through the various predicates in order, making sure that no predicate is tried in a box that already has a region found with that predicate. (Otherwise the same region could be found twice with the same predicate. It can still be found twice with different predicates).

There is no reason why it couldn't be more clever if given some additional outside information.

EXEC1 Display Functions

The EXEC1 display functions work very much like the EXEC2 ones, except with the output of EXEC1 rather than that of EXEC2 (The difference in the boundry representation of a region being the major one).

INITDISPLAY must be called first, with the same arguments. REGDISP works pretty much the same with one major difference: It has an additional argument, its arguments being

number, rlist, array, pts, boxes

It uses the argument array to display all the points of the array which satisfy the predicate under which the region was found The display of the points of the array is done with the function DISPRED. The boundry point in order to make the boundry stand out. It does slightly but you have to look close. Also, instead of the region number, it types the predicate.

There is another display function: PREDISP with the following arguments.

pred, rlist, array, pts, boxes

It produces a display which is the logical or of the displays produced by calling REGDISP for each region found with the predicate pred. It types "n REGIONS", where n is the number of regions found with that predicate.

Note that DISPRED will not work with a predicate that does some remembering (e.g., colorpred), as most interesting predicates will, so that some random points may be displayed which allege to be points of a region found with the predicate, but are actually randomness.

C'est la guerre.

CS-TR Scanning Project
Document Control Form

Date : 12 / 14 / 95

Report # AIM - 111

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☐ Technical Report (TR) ☒ Technical Memo (TM)
☐ Other: ____________

Document Information

Number of pages: 25 (29 - IMAGES)
Not to include DOD forms, printer intstructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☒ Single-sided or
☐ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☐ Unknown ☒ Other: MEMEOGRAPH

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☐ Cover Page
☐ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: ____________

Page Data:

Blank Pages (by page number): ____________

Photographs/Tonal Material (by page number): ____________

Other (note description/page number):

Description : Page Number:

IMAGE MAP! (1 - 25) UN#'ED TITLE PAGE, 2-25
(26-29) SCANCONTROL, TRGT'S (3)

Scanning Agent Signoff:
Date Received: 12 / 14 / 95 Date Scanned: 1 / 23 / 96 Date Returned: 1 / 25 / 96

Scanning Agent Signature: Michael W. Cook