145

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence Project
Vision Memo. No. 101.

Memorandum MAC-M-320
August 1, 1966

SIDES 21

Richard Greenblatt and Jack Holloway

Described by Donald A. Sordillo

SIDES 21 produces a graph consisting of the locations of lines which comprise the sides of either a geometric solid or a plane figure. The representation is in floating point mode, suitable for subsequent processing. The input is a picture intensity-function.

Introduction

Let us define a table as a black area, delimited by a white border. Consider a white opaque object (in this instance a cube) resting on the table. The whole of the table is considered to be in the field-of-view of the vidisector. Sides 21 will "find" the cube, in that a representation of the location of each of the edges visible to the vidisector will be stored in the computer.

The following terms will be used only as defined below:

Edge	A physical entity: that part of a cube formed by the meeting of two and only two sides.
Vertex	That part of a cube formed by the intersection of two or more edges. To the program, the number of edges at a vertex is dependent upon the angle at which the cube is placed with respect to the vidisector. [It is assumed that this angle is kept constant during the execution of the program.]
Line	That area demarcated by successive positions of the acceptance box (q.v.). This has no physical existence and thus differs from an edge. A line will have a representation which is a function of the program's past history in looking at an edge.

Box A plane figure comprised of three rectangles with ratios as indicated in Figure 1.

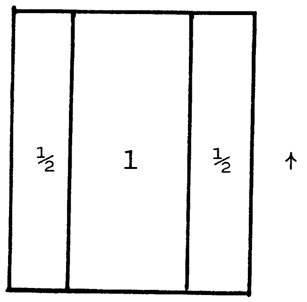

Figure 1.

The innermost rectangle is termed the acceptance box; the outer two rectangles are collectively termed the looking box. The dimension indicated by ↑ is the major axis. When tracking, the noisier (i.e. less sharply defined) the edge is, the wider the box must be to successfully track it. The length and width of the box are functions of the current length of the line and the iteration number of the box.

Operation

The program has, as initial conditions, the coordinates of a starting point. For now, assume that this point is sufficiently near the cube so that an edge will be within the area swept over.

The program forms boxes on the periphery of an imaginary circle about the starting point by using the algorithm

$$X_i = X_{i-1} + \frac{Y_{i-1}}{K}, \quad Y_i = Y_{i-1} - \frac{X_{i-1}}{K}.$$

The parameter, K, (whose current value is 5) determines the number of boxes about a point. If necessary, slightly more than 360° will be traversed before the program stops trying to find an edge.

The box tracks and searches by constructing perpendiculars to the length of the box as in Figure 2. As the line extends across the width, it searches for the maximum gradient. This is determined by taking successive differences of the form: $\log V_i - \log V_{i-1}$.

[Where V_i and V_{i-1} are the inputs from the vidisector - integers between 0 and 256, which are inversely proportional to the intensity of the light seen. The program uses only the logs of the V_i 's for computation.]

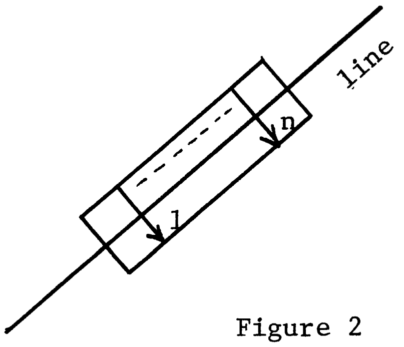

Figure 2

For each sweep, the greatest difference observed is stored; and, at the end of the sweep, this number is compared with a parameter (the current value of which is equivalent to 3 vidisector units.) If the number is less than this parameter, the program thinks that no gradient was found. The program does not differentiate between this condition and one in which the gradient was outside of the acceptance box. The relative location of the maximum gradient with respect to the box is also known to the program; and this information is subsequently used to steer the box. The parameter is set low so that the box does not go berserk when servoeing under noisy conditions. Yet, since the program decides which points to accept and to reject, the function that tries to extend the line is not affected by a point with no gradient (or one outside the box) for these points are never even considered.

The "looking" section steers the box as a function of where the general trend of the maximum gradient appears to go. If the line is within the acceptance box as well, the program considers that the correct location of the line has been found, and thus will track further.

In the case of a noisy edge, the box is widened as a function of how far the maximal gradient goes astray. If, on a given try a certain percentage of points on the edge fall within the acceptance box, the program computes any steering necessary, servoes onto the intended line and extends itself a certain percentage of the box's length. 90% of the points must be inside the acceptance box before the box will extend. To compensate for the case where the box is near the end of the line and the first 89% of the point may be in and the remaining 11% out, a running total is kept so that the box will advance, but not as much as usual. As the length of the line increases, the points further out count more for steering. The box is, in theory, free to rotate and is not constrained by the axis; however, the servoeing mechanism is overdamped and the program will not let the box rotate too far without looking for a corner.

Within the program is a function that has available to it the current length of the line of interest. If the line cannot be extended, this function decides what to do. E.g. it may increase the dimensions of the box and try again. [When this is done, the percentage of increase of length is larger than the percentage of increase of width. This has the effects of 1) counteracting vidisector noise; and 2) speeding up the program.] When just starting to look for a line, and loose tolerances are desired, a negative line length is given to this function. This results in a larger box.

Thus, when tracking a line, it uses a small box (close tolerances) whcih requires few vidisector points and, consequently, runs fast. When noise is encountered a wider box is called into use, which allows for more latitude in tracking, at a sacrifice of speed.

All parameters can be set as a function of how long the line is at the present time. Thus when the line is long, the direction that the line is going is known fairly accurately and not much angular deviation is tolerated. If the apparent line goes off in another direction, it is most likely a different line. If the line is short and there is a considerable amount of deviation, it tracks the deviating line on the assumption that it did not know the real direction. If an attempt is made to extend a line and it fails, other attempts are made up to a certain maximum, designated by a parameter.

When the box cannot be extended any further along the line, that point is recorded as a vertex and the radiation from that vertex of another sharp gradient change is sought.

To find a new edge, trial boxes are drawn in a circle about the vertex. Hopefully one of these boxes finds an edge. This done, the program goes through two iterations of using very large scans and wide acceptance factors to attempt to get onto the edge. (This prevents the occurrence of a large error in the assumed direction.) Lines that emanate from the same vertex are stored as a circular list structure. As the vertex is traversed, the ring is built up so that it eventually has entries for all of the end points that belong to the vertex. There is an implicit link between the end points, so vertices are connected to one another by this data structure.

The program continues in this fashion until the terminating condition is reached, viz. it finds a line which it had previously found. This condition is detected by a routine which makes periodic tests on the line being tracked, after it exceeds a certain length. The cross-product of the line being tracked and a line previously found is formed and normalized. This gives a number, proportional to sin Θ, which is compared with a threshold. If they are not parallel, the lines are ordered so that the two ends with the closest x value are together. Then the cross-products of vectors 1 and 2 are computed. [See Figure 3.] If 1 and 2 are parallel, they are different lines.

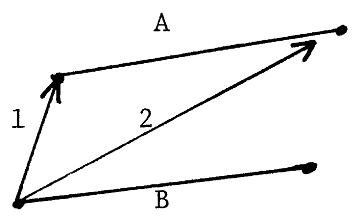

Figure 3.

If the lines appear to be different, checking is continued until the end of the line. Before the line is accepted, the entire line is retested. [The vidisector is so slow it is worth spending the time on the chance that the line will be deleted.]

When the terminating condition is met, the program jumps back one vertex and starts checking for additional lines emanating from it. If, after a sweep of slightly more than 360° about the vertex, no lines are found, it considers that vertex completely checked out. If the initial scan runs into an edge at a place between vertices, there is no problem of finding a half-line since the box searches with wide tolerances until it finds a corner and the line it first tracked is ignored--to be found later on.

The current operation of the program is such that there are tolerances in %LEAV which say that the line must be extended at least _n_ tries or it is rejected.

The routine will accept as different two abutting lines (within a certain tolerance) but will reject two lines that overlap. If a line is not found after sweeping around the initial point, it gives up. An inspection of LTBL will reveal this situation.

The lines are represented (in floating point) by two ordered pairs of numbers, and are stored in LTBL--four register per line. Each pair represents the x and y coordinate of an end of the line.

After the program has "found" a cube, further processing is available on the lines stored in LTBL, as noted below.

%FLUSH	This routine has as its arguments _n_ lines stored in LTBL. If it finds three lines that are parallel it eliminates the middle line. Thus, in the case of a cube, one would expect the three inner lines of the cube to be deleted and the perimeter of the projection to remain. The lines are also ordered by this routine as a function of their coordinates--the effect being the storage of the perimeter of the figure by consecutive sides.

Summary

In Figure 4, assume the program is given point X, and begins a circular scan, picking up the edge at position b. It will then track, with very large acceptance factors, until it reaches either vertex B or vertex B'. When it is at one of these vertices, it will sweep out in a circle, until it picks up a line. It will then trace this line to a new vertex. When this is done, it pushes down the information one level and starts out from the new vertex. When it is satisfied that it has tracked around once, it advances one level on its list and searches again. Finally all nine lines are found and stored.

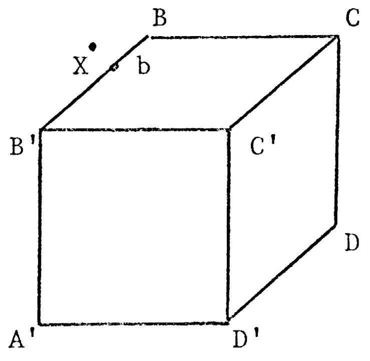

Figure 4

Normal Usage

The following is a description of how the entire package would operate. Those persons interested in using special subsections for particular applications are urged to consult either of the authors for details since the first derivative of the program is not yet zero.

The program is called by: PUSHJ P, NSIDE. It will first make a rought vertical scan with the vidisector, taking gradients and finding an average value for the scene. It then makes horizontal scans until it finds something that exceeds the gradient threshold. When a vertex is found, %LEAV is called. This tries to trace all lines leaving the point. To do this, %LEAV calls LINEX. This routine will try to advance the box one position each time it is called. [The actual displacement of the box is a function of the past history of the line.] If the box is able to be advanced, the next instruction is skipped. LINEX calls the length and width functions which determine the dimensions of the box.

When all the lines are found, %FLUSH is called to order them and delete inner lines. Finally CØRNS[1] is called to find the vertices.

If fewer than four sides are found, the program tries again by resuming the horizontal scan from where it left off.

This scheme, attempting to input an arbitrary graph instead of a simple periphery, has the advantage that if a corner is badly defined, by either poor lighting or physical wear, and the program fails to see the other edges leaving it, there is a high probability of getting the corner since it will be approached from the other sides.

A weakness is that the program does not consider the probability of other edges radiating from a point unless the edge it was on has terminated.

---

[1]For a description of CØRNS, see Artificial Intelligence Memo. [redacted] by Robert South (M.I.T.: October 1965)