MASSACHUSETTS INSTITUTE OF TECHNOLOGY

ARTIFICIAL INTELLIGENCE LABORATORY

Working Paper 113 March 29, 1973

ON SOLVING THE FINDSPACE PROBLEM, or

How to Find Out Where Things Aren't ....

Gregory F. Pfister

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-70-A-0362-0006.

The FINDSPACE problem is that of finding a volume, in a bounded space containing impenetrable objects of fixed position and orientation, where another object of specified dimensions will fit. In two dimensions it corresponds to deciding where to put something down on a cluttered table. The genesis of the problem name is a function in the block manipulation programs of Winograd's natural language understanding system [1]. The applicability of this problem to robotics is clear.

Sussman noted [2] that the FINDSPACE problem appears to involve two major subproblems: a proposer, to suggest a position for an object; and a verifier, to check if the suggested position is feasible. The task of the verifier is more straightforward. It tests whether an object with a given position and orientation intersects with any of a set of fixed objects. The proposer, on the other hand, appears difficult due to the presumed difficulty of finding "the places where objects are't" -- in two dimensions, those areas on a bounded plane not occupied by other objects.

In fact, finding "where objects aren't" -- the FINDEMPTYSPACE (or FES) problem -- can be done very efficiently in two dimensions. This working paper presents an efficient solution which generalizes well to three dimensions. The solution presented is a simple modification of the well-known Warnock algorithm [3] for removing hidden lines from three-dimensional pictures consisting of planar polygons. The critical inner loop test of the FES algorithm presented is basically Sproull and Sutherland's algorithm [4] for "clipping" line segments where they cross a rectangular area. The exact inner loop test needed has a hardware implementation in

the Evans and Sutherland LDS-1 display system [5]. The test also may be implemented efficiently in software.

DATA:

This solution uses data in the following form:

(1) the left, bottom, right, and top coordinates of the area to be considered.

(2) a list of polygons. Each polygon is represented as an ordered list of (X,Y) vertex coordinates, obtained by a walk around the polygon.

The area under consideration is assumed to be rectangular, and its sides are assumed to be parallel to the coordinate axes.

The polygons may be interpreted as the outer boundaries of the figures obtained when objects "resting on the plane" are projected isometrically onto the plane.

Which vertex starts each polygon list is irrelevant.

The polygons may be arbitrarily complex: they may be concave, they may intersect, and they need not be realizable physically (lines connecting vertices may cross).

TOP-LEVEL ALGORITHM

The solution proceeds first by examining each polygon and putting it into one of the following classes:

I. The polygon totally surrounds the area

II. The polygon lies totally outside the area

III. The polygon intersects or lies inside of the area.

(Clearly, every polygon falls into one of those classes. The classification algorithm is crucial, and is described below).

If any polygon is in class I, FES gives up, since clearly there is no free space available.

If all of the polygons are in class II, FES tells the verifier that the entire area is free, and terminates. (The verifier should be a co-routine of FES).

Otherwise, FES recurses: It divides the "total area" in half, using, in alternate recursions, a vertical or horizontal line. It passes each "half area", and all the polygons in class III, to each of two separate recursions. This recursion terminates ultimately when an initially chosen resolution limit is reached.

The top-level algorithm is a log search for free areas; it has the good quality that it tends to find large free areas first. Warnock's original algorithm was, of course, interested in areas of class I; these were passed to a display file. The originial algorithm also recursed fourfold at each

step, dividing the total area into four quadrants. This is, in fact, probably a good recursion/iteration tradeoff if the polygons are uniformly distributed rather than bunched. Since the algorithm converges to areas immediately surrounding polygons very quickly, thereby "bunching" polygons, fourfold recursion may always be better for this algorithm; it was not specified because it makes the verifier work harder; by unnecessarily splitting free areas.

CLASSIFYING POLYGONS

First, consider intersection. If no line of the polygon intersects the total area, the polygon does not intersect the total area. Each line, defined by successive pairings of vertices, can be tested as follows:

(1) if either endpoint lies inside the area, the line intersects the area or lies totally within it.

(2) if both endpoints simultaneously lie above, below, to the left, or to the right of the area, the line does not intersect the area.

(3) if the line satisfies neither of the above conditions, find its midpoint and recurse on each half.

The "recursion" in step (3) can be done by an iteration, since one-half of the line is always trivally in or out according to step (1) or (2). Also, note that if the endpoints are translated to coordinate systems based on the edges of the area, the tests in (1) and (2) are sign tests.

The ultimate termination is again done by a resolution limit (possibly different from that of the top-level algorithm).

This is another log search. Sproull and Sutherland's original algorithm continued until it found the point of intersection between the line and an edge of the area.

If no line of a polygon intersects the area, the polygon could either surround the area or lie outside it. This can be tested by counting how many times any ray from within the area crosses lines of the polygon. If the direction of the crossing is counted -- +1 for counterclockwise, -1 for clockwise -- a result of Ø means the polygon lies outside. A directed intersection check is conveniently done as part of the intersection test if the ray chosen is colinear with a side of the area.

The classification algorithm presented here is implemented as the "minimum effort mode" of the Evans and Sutherland LDS-1 display.

GUIDING THE SEARCH:

The search for free areas can be made somewhat faster -- by making the areas found larger -- if "interesting" positions for splitting the area are used instead of a simple half-and-half split. Such "interesting" positions are:

(1) the locations of vertices within, or on the edge of, the area

(2) the intersections of lines with the edge of the area.

In fact, use of vertex locations above produces good results for little effort, since it produces areas where diagonals are lines (the optimum) and can be computed as a side effect of classifying polygons. Intersections are probably not worth the effort.

Which of a group of vertices is "most interesting" is a harder question to answer, and the speed with which the top-level algorithm converges implies that not much time should be spent answering it. One reasonable ordering is by "Manhattan distance" ($\Delta X + \Delta Y$) from the area center.

On the other hand, the verifier could possibly help by indicating where it would like to find more free area, since using and passing such information down in recursion would not require much work. The most important help this could give would be an indication of which side to recurse down first.

The latter considerations lead to thoughts of a program to do a guided breadth-first search. The recursion depth could be controlled by manipulating the top-level (not classifier) resolution limit, and the branch followed next could be determined by the verifier. Excessive complexity should be guarded against, however, since situations where such complexity is "needed" will probably find the verifier being the real weak link: how do you translate and rotate an object to fit in the free space you already have? (This "verifier" itself contains a proposer and a verifier; the inner verifier is clearly a full hidden-surface test.)

THREE DIMENSIONS

The top-level algorithm obviously generalizes to three dimensions, using a rectangular volume and planar polyhedra. The classifier, however, must now worry about intersections of three-dimensional planar polygons with a rectangular volume. This is a special case of the hidden surface problem. If arbitrary polygons are allowed, even this special case is difficult to solve. However, if all polygons are triangularized, the following extension of the two-dimensional classifier works:

(1) if any of the three vertices are inside the volume, the triangle intersects the volume.

(2) if all three vertices are simultaneously above, below, left of, right of, in front of, or in back of the volume, the triangle does not intersect the volume.

(3) otherwise, recurse using the centroid of the triangle (vector sum of vertices divided by 3) to define three triangles within the given one.

This has the unfortunate characteristic that only one of the three triangles in the recursion will necessarily terminate trivially on the next step.

I suspect that there is a non-recursive and/or faster way to do this. Line intersections with the volume can be computed by a trivial extension of the two-dimensional classifier, and this will lead to faster "trivial"

detection of intersection in many cases. However, it is unclear whether the extra computational overhead is worth it; experiments are in order. If line intersection information can easily be combined with above/below etc. vertex information to also increase the number of non-intersections "trivially" found, I believe that the line intersection computations would stand a very good chance of being justified.

It should be noted that the type of algorithm presented here is not complete in a practical sense. Simply finding an appropriate free volume is not enough, since one also needs to know the path an object must take to reach the volume. This significantly harder problem does not appear in the two-dimensional problem, since for practical cases the latter is always embedded in three-dimensions (a cluttered table).

CONCLUSION

This memo has presented a method for finding the unoccupied space in a bounded plane containing arbitrary polygons, and has indicated how the method can be extended to the three-dimensional case.

The full FINDSPACE problem, however, has not been solved. In particular, little has been said about either (1) consolidating the separate rectangular free areas produced into larger, irregular areas; or (2) deciding whether a given object actually fits into such an irregular free area. This has been assumed to be the domain of the verifier.

However, the solution described has the good property that if it is done

breadth-first, the largest free areas are found first. This means that in an uncluttered plane, i.e., one with large free areas, an extremely "stupid" verifier can be successfully used. Moreover, if the polygons are restricted to being rectangles oriented along the coordinate axes, as in the original problem context of Winograd's blocks manipulator, an optimal verifier is not very complex. In addition, the algorithm presented will be extremely efficient in this context.

For the case where the bounded plane is very cluttered with general polygons, it was noted that the verifier can guide the algorithm presented so as to aid the verification process. An optimal verifier for this case will be very complex, since it must incorporate a general pattern matcher (and possibly a jigsaw puzzle solver). Of course, people don't do too well in the bad cases either.

BIBLIOGRAPHY

Winograd, T., A Computer Program for Understanding Natural Language, M.I.T. Ph.D. dissertation, February, 1971.

Sussman, G., The FINDSPACE Problem, M.I.T. A.I. Laboratory Memo No. 286, March, 1973.

Warnock, J., A Hidden Surface Algorithm for Computer Generated Halftone Pictures, University of Utah Ph.D. dissertation.

Sproull, R., and Sutherland, I., A Clipping Divider, Proc. AFIPS 1968 FJCC, Vol. 33, Pt. 1, AFIPS Press, Montvale, N.J., pp. 765-775.

-------, Line Drawing System Model 1, System Reference Manual, Evans and Sutherland Computer Corporation, 3 Research Road, Salt Lake City, Utah, November, 1970.