MASSACHUSETTS INSTITUTE OF TECHNOLOGY
A. I. LABORATORY

Artificial Intelligence
Memo No. 287 March 1973

FINDING THE SKELETON OF A BRICK*

Tim Finin

ABSTRACT

TC-SKELETON's duty is to help find the dimensions of brick shaped objects by searching for sets of three complete edges, one for each dimension. The program was originally written by Patrick Winston, and then was refined and improved by Tim Finin.

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense, and monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0005.

---

*This memo was first issued in August 1971 as A.I. Vision Flash 19.

FINDING THE SKELETON OF A BRICK

I. INTRODUCTION

TC-SKELETON is a program which is given the task of finding 3 lines of a brick which will determine its dimensions, i.e. a brick's "skeleton". This "skeleton" is used by other programs in the robot system which eventually determine the brick's location in real space. TC-SKELETON was originally written to cope with line finders which would return very bad data-many missing lines-and was adapted to its present function. TC-skeleton has again been rewritten to better perform its task.

THE OLD TC-SKELETON

The general procedure for finding a brick's skeleton was as follows: The lines of a brick (which assumes to have one face parallel with the table) are classified as:

TYPE-ZERO: vertical lines

TYPE-ONE: lines with slope ≥ 0

TYPE-TWO: lines with slope ≤ 0

(see figure 1)

TC-skeleton first attempted to find a type-zero (vertical) line. It then examined the endpoints of that line to find an intersecting line of type one

or two. When one was found it looked at the endpoints of the two lines so far for an intersecting line of the other type. Thus, TC-skeleton returned a connected set of three lines, some examples of which are shown in figure 2.

THE NEW TC-SKELETON

A new TC-skeleton has been written with the following major changes:

1. It will avoid mistakes which the old TC-skeleton could make.
2. It will handle scenes which the old TC-skeleton would not.
3. It must no longer return a connected set of three lines as a skeleton.
4. It will return a "partial skeleton" (one or two lines) if it is unable to return a complete one because of obscuring objects.
5. The code has been generally cleaned up. Minor bugs and inconsistencies have been removed.

LINE TYPES

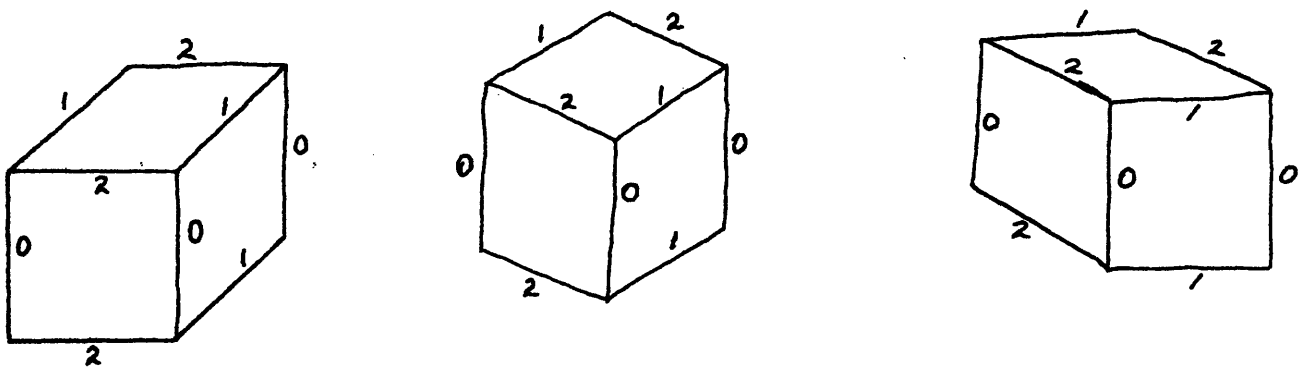

FIGURE 1

SKELETONS

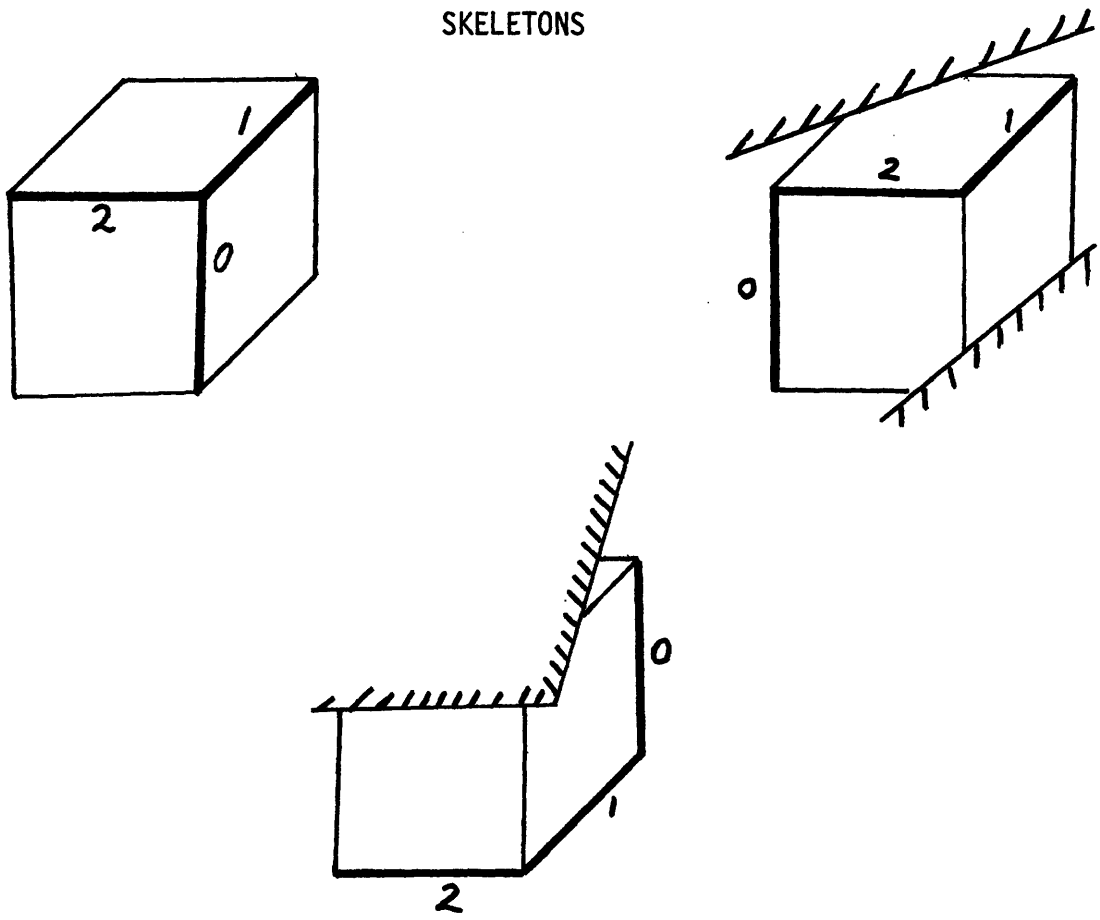

FIGURE 2

II. CENTRAL PROBLEMS

There are two central problems in finding a brick's skeleton. Any candidate skeleton line must be:

1. A true edge of that brick, vs. an edge of some obscuring object.

2. A complete edge, rather than an edge of which we see only a part.

In general, a line which divides regions of two bodies could be a true physical edge of both bodies or just one. Although TC-skeleton has no real procedures for finding the true edges of a brick, there are several embedded

heuristics which will discover many of them. For example, interior lines, lines which lie between two regions of the same brick, are recognized as true edges of that brick. Also if the shaft of an arrow vertex is an edge of a brick, then the lines forming the barbs of that arrow will most likely be edges of that brick.

The information about lines is actually information about line segments. For example, in figure 3 the system does not know of line AD, only of the line segments AB, BC and CD. We must be sure to extend a line such as AB to its true length AD. We must also be able to determine when we have an entire line, as opposed to a part of the line where the rest is obscured. The lines AE and FG must be rejected as components of the brick's skeleton.

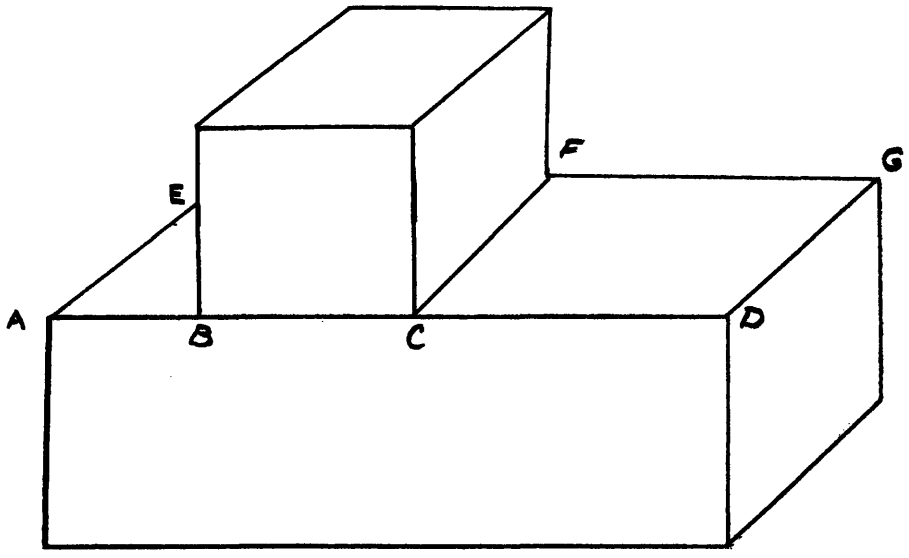

FIGURE 3

III. THE NEW TC-SKELETON

THE STRUCTURE OF TC-SKELETON

The structure of TC-SKELETON is shown in figure 4. The type of a line is determined by its slope. Instead of directly examining the slope of a line, TC-SKELETON looks at the relative slope of lines at vertices, in particular, at fork-arrow- and L-vertices thru calls to TC-USE-FORK, TC-USE-ARROW, and TC-USE-L. TC-FIND-VERT is used to find vertical lines when looking for a type-zero line. TC-FIND-END is used to find the actual termination (if possible) of a candidate skeleton line.

TC-USE-FORK

TC-USE-FORK is the first theorem tried in looking for a line of any type. The program attempts to find a fork-vertex for which all three regions surrounding it belong to the brick in question. This guarantees that all thru lines forming the "interior-fork" vertex will be edges of the brick. We can pick out the type of line we seek as follows:

> The vertical line will be a type-zero.
>
> The next line, going counter-clockwise around the vertex from the vertical line, will be a type-one.
>
> The next line, going clockwise around the vertex from the vertical line, will be a type-two.
>
> (see figure 5)

Once we have a candidate skeleton line, TC-FIND-END is applied to it. TC-FIND-END attempts to find the other end of the line. If it fails, i.e. the other end of the line is obscured, the candidate line is rejected and TC-USE-FORK fails.

TC-USE-ARROW

TC-USE-ARROW is tried if TC-USE-FORK was unsuccessful in finding a line of the type sought. TC-USE-ARROW looks for arrow vertices where the two narrow angle regions are faces of the brick in question, suggesting that the arrow barbs are true edges of the brick. The lines are identified as follows:

The vertical line is a type-zero.

The next line going counter-clockwise from the vertical is a type-two.

The next line going clockwise from the vertical is a type-one.

(see figure 6)

Again, TC-FIND-END is applied to any candidate lines found.

If TC-USE-ARROW also fails to find a line of the type sought, either TC-USE-L or TC-FIND-VERT is called, depending on the type of line sought.

TC-FIND-VERT

TC-FIND-VERT is used only when looking for a type-zero line. It simply looks for vertical lines which bound a region of the brick and for which both ends can be found (via TC-FIND-END).

TC-USE-L

TC-USE-L is applied when looking for a type-one or type-two line. It looks for L-vertices at the ends of skeleton lines already found. The implies heuristic is that if one leg of an L-vertex is an edge of the brick, then the other is also an edge of the brick. If such an L-vertex is found at the end of a vertical line, then the counter-clockwise angle between the vertical line and the other line determines the type of the other line. As figure 7 shows, if this angle is less than 180° it is a

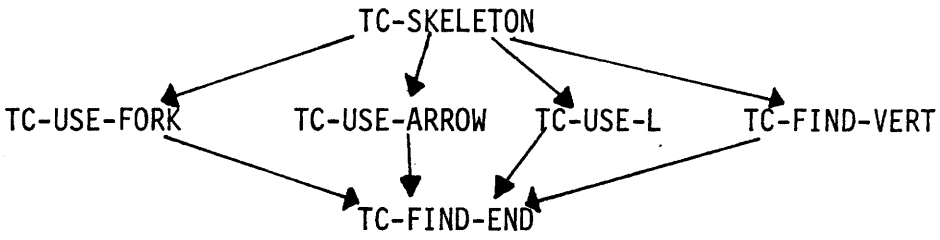

FIGURE 4

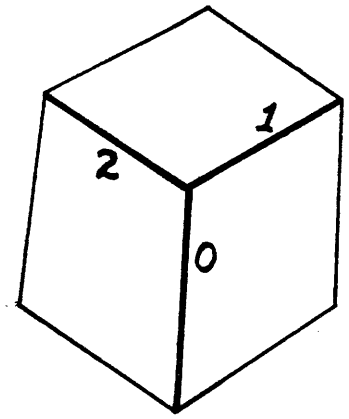

FIGURE 5

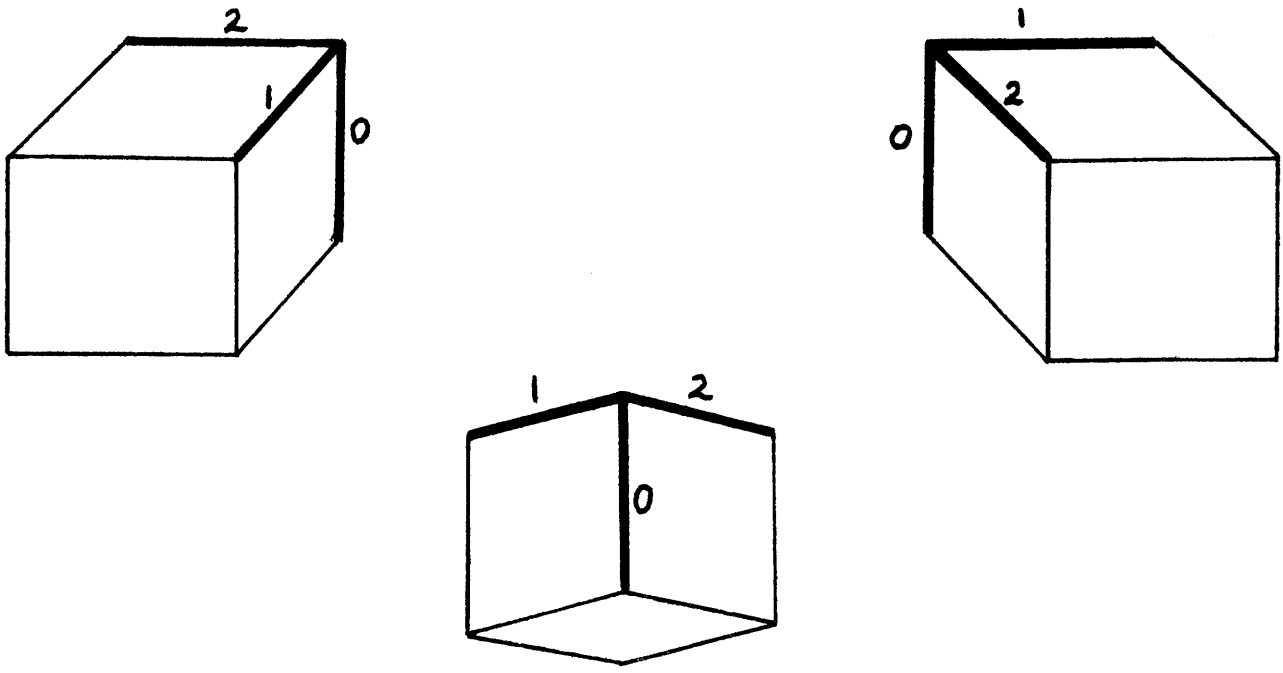

FIGURE 6

type-one, and if greater than 180° it is a type-two.

If neither of the L-vertex legs is a vertical, then one must be a type-one and the other a type-two. Since we are looking at L vertices for which we know the type of one of the legs, the other leg will be of the other type.

Examining L-vertices has the potential for leading us astray, as, if some of the lines are missing, arrow vertices may appear as L-vertices. For example, looking at vertex V1 in figure 8, TC-USE-L would erroneously conclude that the line V1 - V2 is a type-two instead of a type-one. This problem is avoided by ignoring any L-vertex where one of its legs ends in a fork vertex.

TC-FIND-END

TC-FIND-END is called when a candidate skeleton line has been found. It attempts to find the actual end of the line as opposed to an apparent line caused by some obscuring object. TC-FIND-END extends edges thru collinear lines of T- and K-vertices. In figure 9 the AB are extended to their true length AC.

THREE TYPES OF L-VERTICES

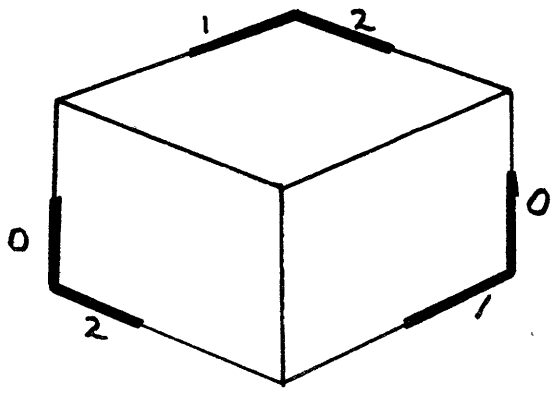

FIGURE 7

A BAD L-VERTEX

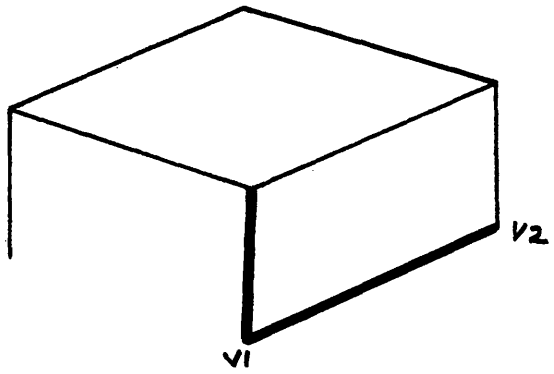

FIGURE 8

After extending a line as far as possible, TC-FIND-END must determine if the far end is the actual termination of the edge. The set of vertices accepted as actual ends of an edge depends on the vertex at the other end of that edge (i.e. which theorem is applying TC-FIND-END). If TC-USE-FORK is the calling theorem, X-, K-, L- and ARROW- vertices are accepted as edge termini. Otherwise, X-, K-, and L- vertices are accepted.

A line is also taken as complete if it ends in a:

1. T vertex with the table (background) on the wide angle side of T vertex. (figure 10A)

2. A vertex with the TABLE as one of the regions around it, but not on either side of the line being considered (figure 10B).

3. A T vertex where one of the narrow angled regions is a face of the brick in question, and both other regions belong to another body (figure 10C).

If the vertex at the end of a candidate skeleton line is other than any of these, TC-FIND-END fails, and the candidate is rejected.

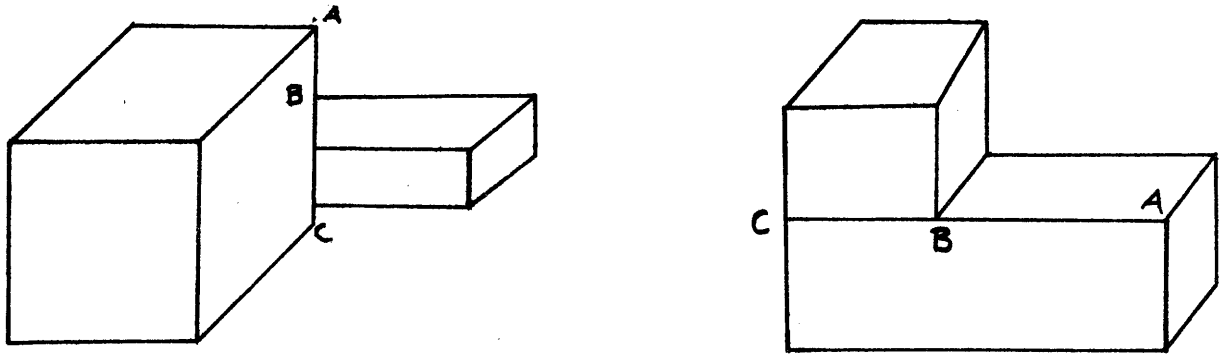

FIGURE 9

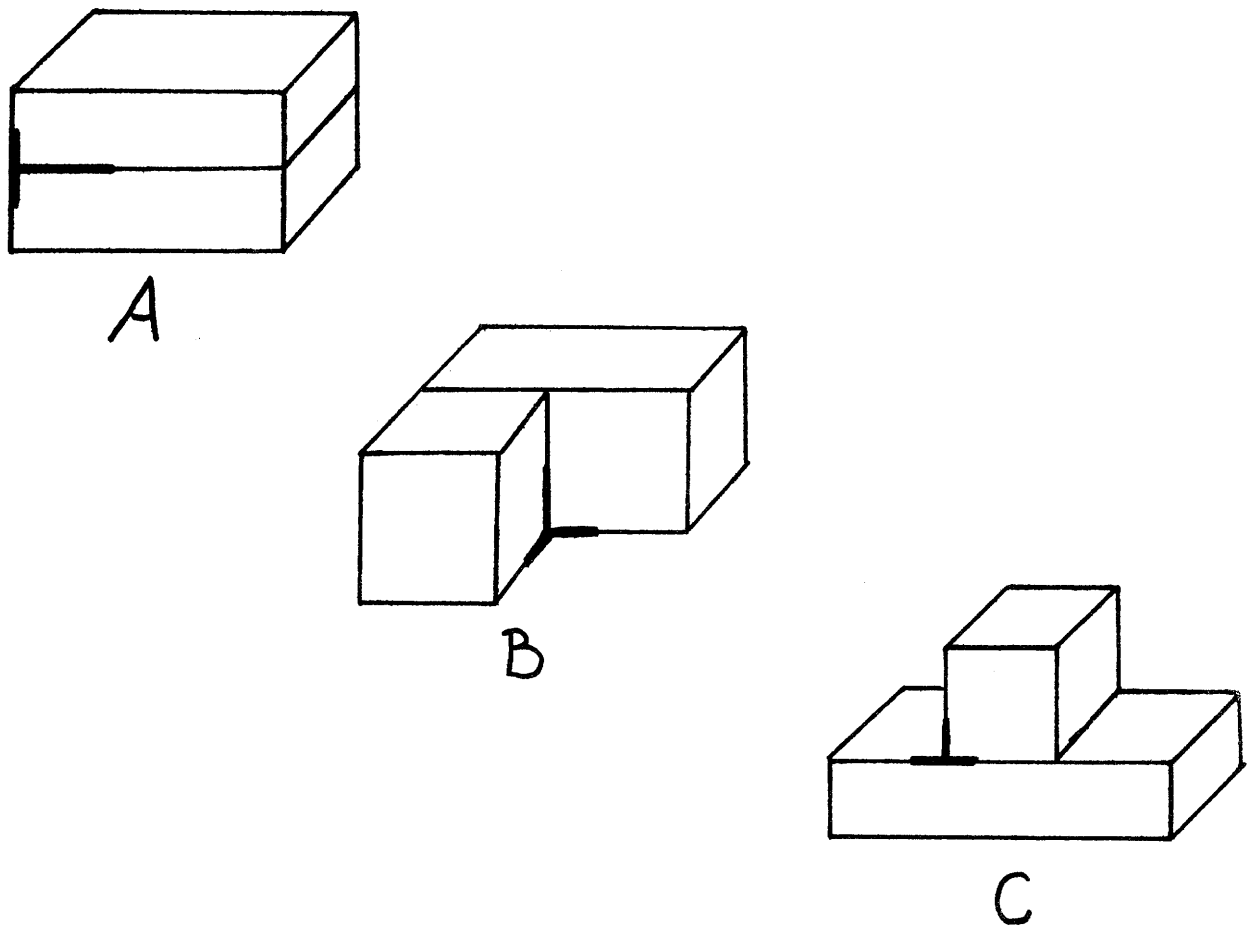

FIGURE 10

IV. SUMMARY

The assumption that the DATA might be very bad has been relaxed slightly. This reflects improvements in the line finder, and has allowed some changes which cause TC-SKELETON to win more often while still retaining its ability to handle scenes with a few missing lines.

TC-SKELETON has very simple "edge-assigning" heuristics, since it handles only FORK-, ARROW- and L-vertices. In particular, scenes of aligned bricks with X- and K- vertices will easily foil TC-SKELETON. In figure 11, TC-SKELETON will only find a type-zero line for B2's skeleton.

This problem could be solved with better edge assignment. A set of heuristics already exists which does a good job of assigning lines to bodies in even complex scenes. Extended Huffman-like labeling of scenes seems to be a most promising approach. In such a scheme, an arrow label determines that the line is a true edge of the region to the right of the arrow. A + or CRACK ('FLAT') labeling implies that the line is a true edge of both regions. A - labeling must be decomposed if it results in the junction of two bodies. In such cases it becomes an arrow label.

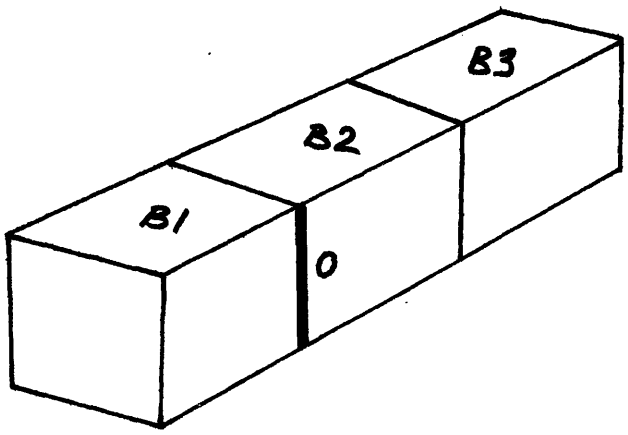

FIGURE 11