MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence
Memo. No. 163
Revised

August 1968
April 1970

HOLES

Patrick H. Winston

This memo originally had two parts. The first dealt with certain deficiencies in an early version of Guzman's program, SEE. The problems have been fixed, and the corresponding discussion has been dropped from this memo. The part remaining deals with line drawings of objects with holes.

Part II: Holes in Objects

Guzman's program does not work very well on holes. The scene of figure 10 is identified as two objects. Humans do a little better but do not know how. If one selects a typical human and asks how he is sure something is a hole rather than a second object, he usually complains that it just cannot be any object.

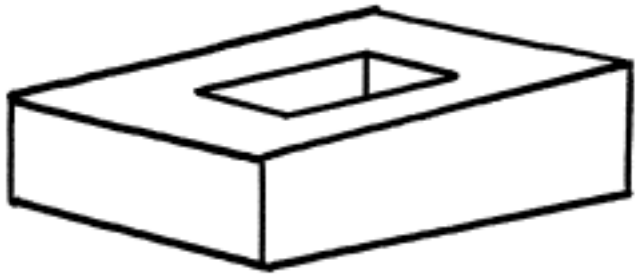


Figure 10. A hole.

I do not think we identify holes through elimination of other candidates. But then the question is What is it in a conglomeration of lines that we call a hole such that we do indeed call it a hole. To answer this question, I examine a number of obscured holes to determine what is sufficient for the identification of holeyness.

In figure 11, one sees that the arrow alone does not help much. The arrow might represent a hole, but an upended brick or wedge are also possible.

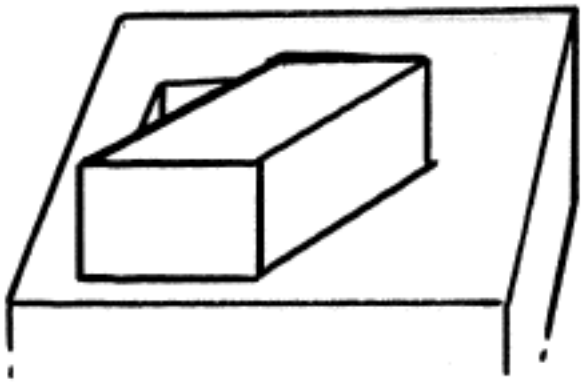


Figure 11, The arrow of a hole.

The arrow together with an L are better. Examine figure 12. On the right one has a hole,* while on the left one has a brick resting on a wedge. The only difference is the direction of the arrowed line. Figure 13 shows intermediate cases. Most people agree that on the right one has an oddly shaped hole, while on the left, a lopsided brick. The emergent rule seems to be: If the direction of the further line of the L is parallel to the center line of the fork, then no hole; if the direction is parallel to the other line of the fork, then a hole is quite likely; intermediate directions indicate situations between these bounds.

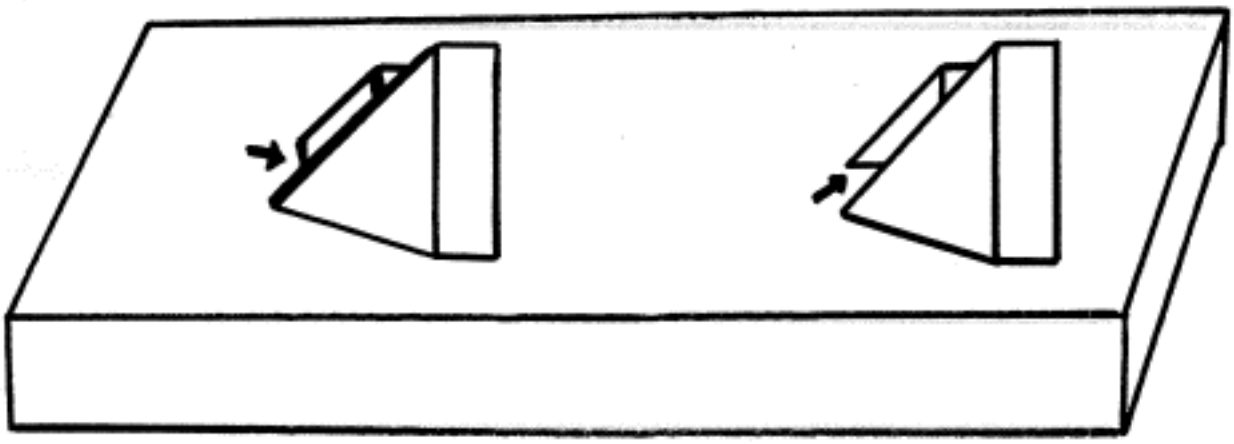


Figure 12. A hole and a brick.

*The pyramid and wedge are also possible, but only if viewed from a degenerate angle; notice that the lines observed to be parallel lose that feature under perturbation except in the case of a hole.

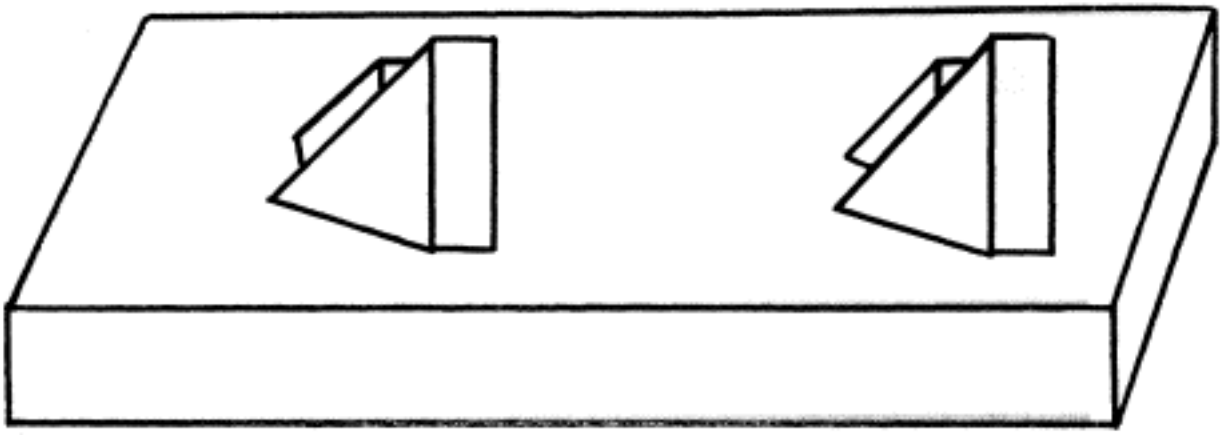


Figure 13. A poor hole and a poor brick.

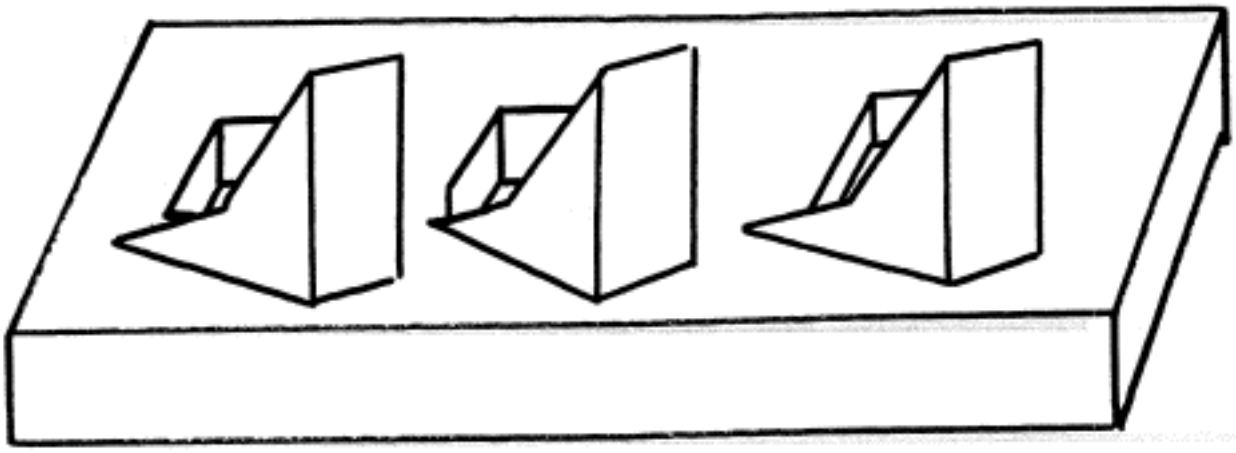


Figure 14. Adding a fork.

Adding the bottom of the hole does not add much, as one can see in figure 14. In one case we have a hole; in another, an upended brick; and without the L, complete ambiguity just as before. Naturally in the case of holes with bottoms it is useful to do a simple check to see if the hole goes all the way through.

Turning to the solo T joint as seen in figure 15, one notes first of all that Guzman's program fails to bind. Humans have a hard time too. Identification in these cases seems to be global and vague, perhaps since so little evidence is present.

In any case, the explanation of the through line of the T decides the case. If it makes sense as an edge of some known object, there is probably no hole; otherwise, hole. How to derive the necessary explanation is the next question, and I have no good answer to that yet. Attaching an L to the T does not change the test. A scene similar to that of figure 15 can be drawn to support this conclusion. Figure 16 shows that a fork also makes no change. Again the decision between hole and obscured object lies in the explanation of the through line of the T.

Finally, with unobscured holes (figure 10), one has all of these factors working together. Moreover, the connection of the T to the arrow via the L is local evidence that the through line of the T is not an edge of some obscuring body.

* * * * * *

I submit that correct identification of the parts of the scene in figure 17 would be a good final exam for a hole program. Guzman's program would suggest 1-2-3 and 4-5-6-7-8-9-10 as objects using bottom of the barrel heuristics. I think the ideas of Part I and Part II allied with a lot of sweat and cursing could handle the job.

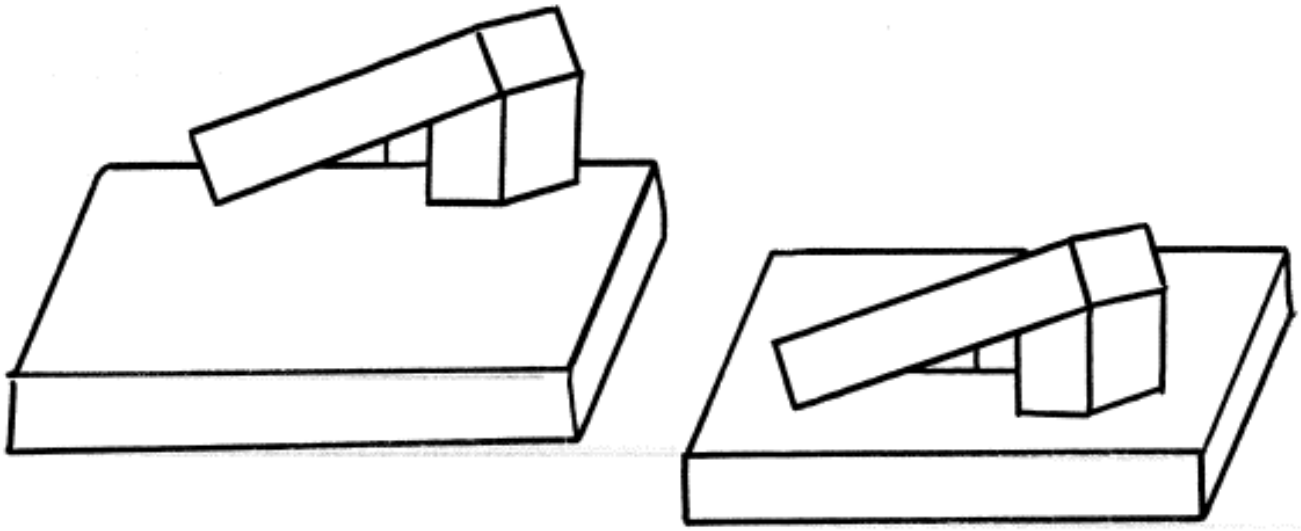


Figure 15. The T joint.

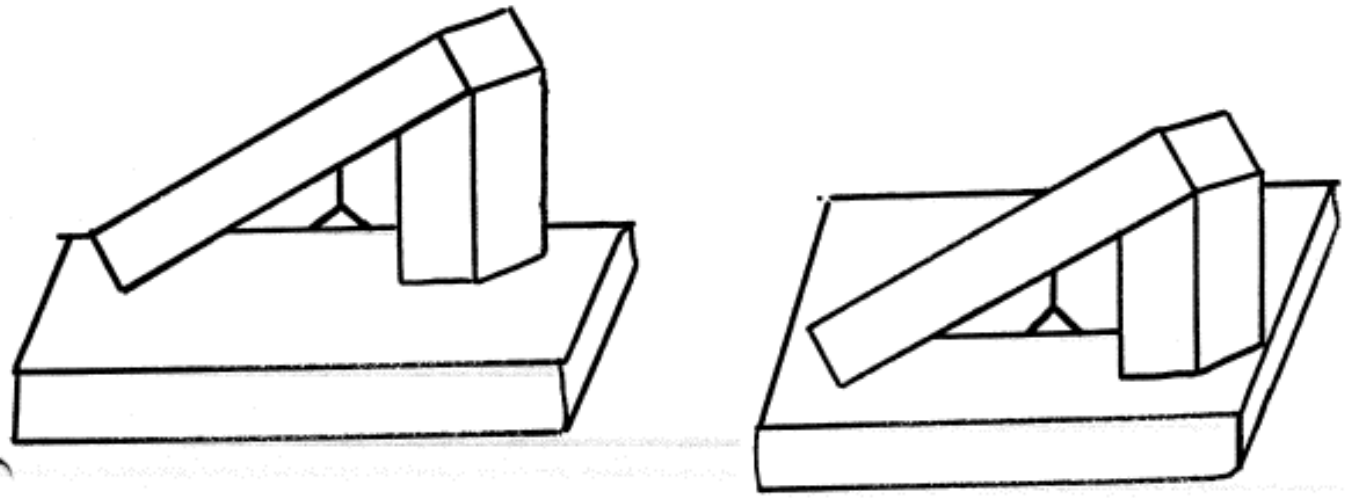


Figure 16. Adding a fork.

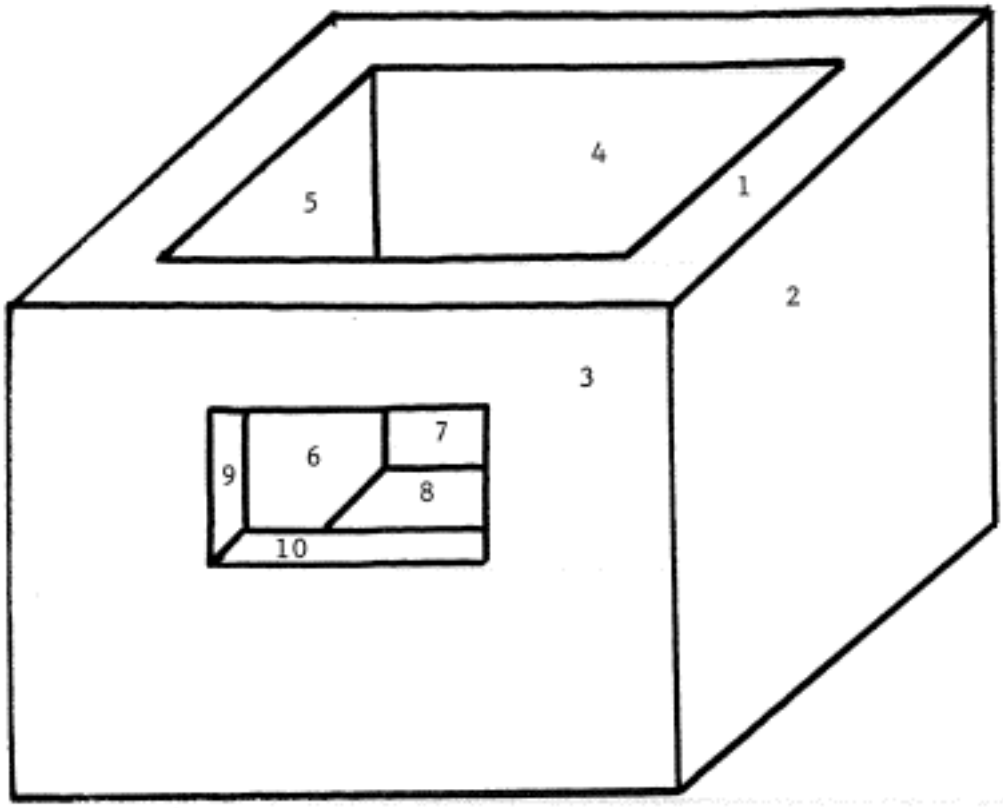


Figure 17. The final exam.

Obsolete after April 1970. Page 8 becomes

Part I: Holes in Guzman's Program

Guzman's program* amalgamates regions into objects adroitly and, for the most part, the resultant partitionings are in harmony with human tastes. This is particularly impressive since the knowledge of the world built into the program is so slight.

But while the program is a remarkable achievement indeed, it can be fooled by scenes with which humans have no trouble. Guzman himself reports some situations in which the program is too conservative and fails to unite all parts of certain objects.** Beyond these, it is also possible to find situations in which the program is too liberal. Consider figure 1. Surprisingly, the three blocks are reported as a single object! The fault lies with the two fork joints indicated by the arrows. These joints create spurious links which are sufficient to bind everything together.

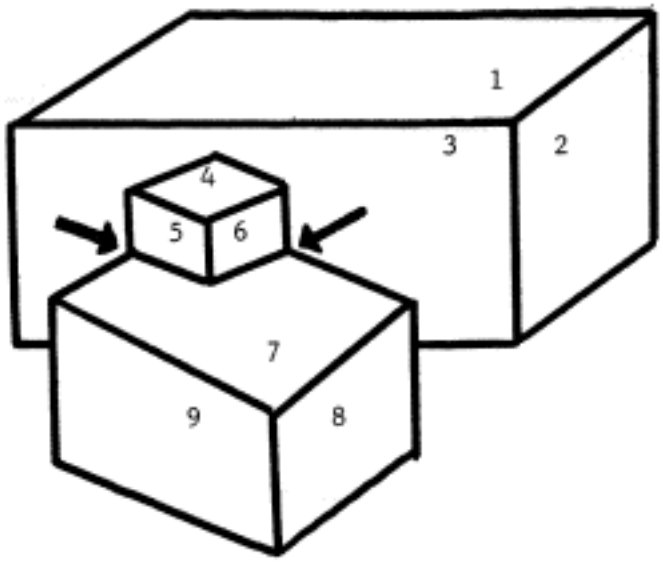


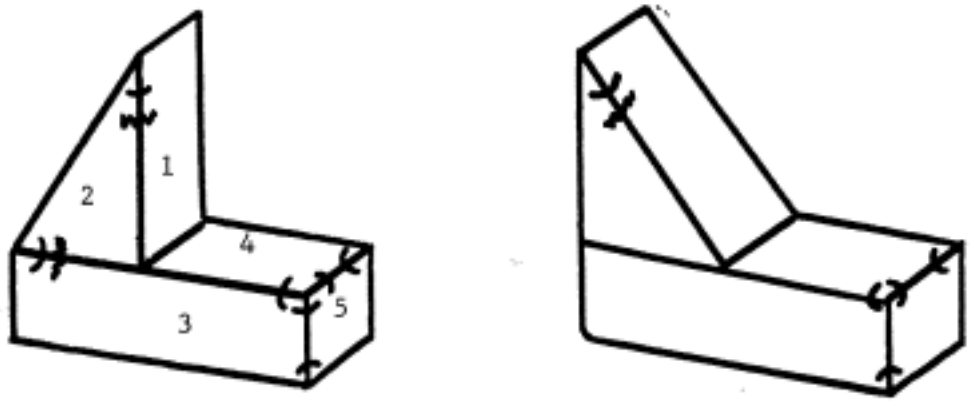
Figure 1. A bad scene for Guzman's program.

---

*MAC-M-357. A.I. No. 139.

**See Guzman, pages 42, 45, and 47.

The scene in figure 2a also causes a blunder. Here the same linkage combination that binds regions 1 and 2 of the wedge also binds region 2 to region 3. Both objects are reported as a single object. Curiously, turning the wedge around, as in figure 2b, eliminates the problem.



a b

strong links:
weak links:

Figure 2. Another bad scene.

There are at least three possible approaches to improvement: tuning the program, providing the program with more information, and integrating the program with an object recognizer. Tuning refers to adding to or slightly modifying the basic heuristics of the program. This seems fruitless. Most ideas that help the program over these examples foul it up somewhere else.

Giving the program more information has promise in situations where a scene happens to be recorded from a bad angle. By this I mean that the program would be cautioned against pseudo-joints of various kinds whose hoax can be exposed through infintesimal perturbation of the viewer's angle. Hence the T joints in figure 3 would be identified as degenerate since perturbation of the viewer's angle changes them into an arrow and a fork. On the other hand, the Ts of figure 4 are persistant since they remain Ts under any infintesimal perturbation of the viewer's angle.

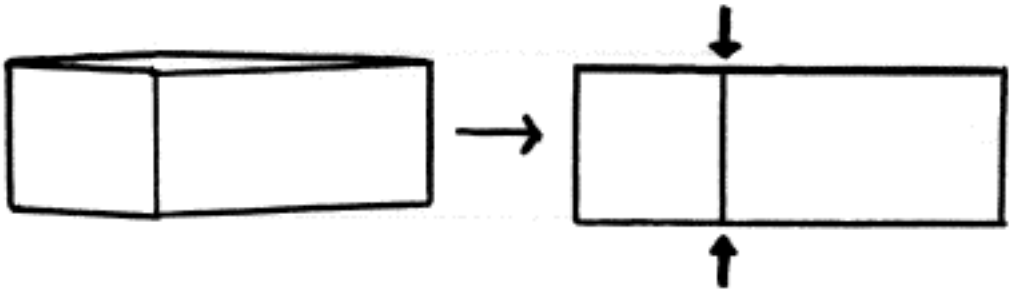


Figure 3. Degenerate Ts

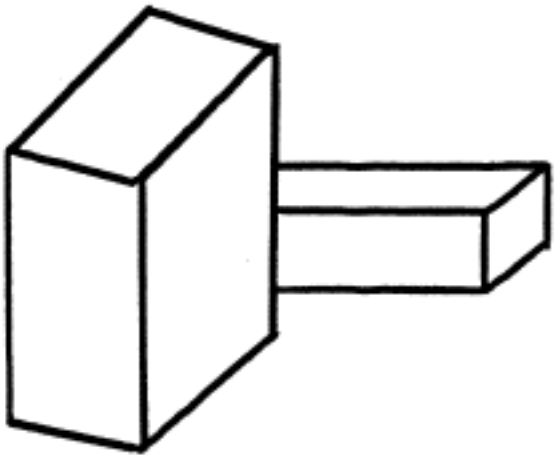


Figure 4. Persistant Ts.

In cases where perturbation of the degenerate T results in an arrow or fork, the appropriate binding could, of course, be effected with the usual confidence. In other cases such Ts would yield misleading linkage were they not identified as degenerate. Joints indicated by arrows in figure 5 illustrate this point. Checking the validity of the five other joints that cause binding should be similarly helpful.

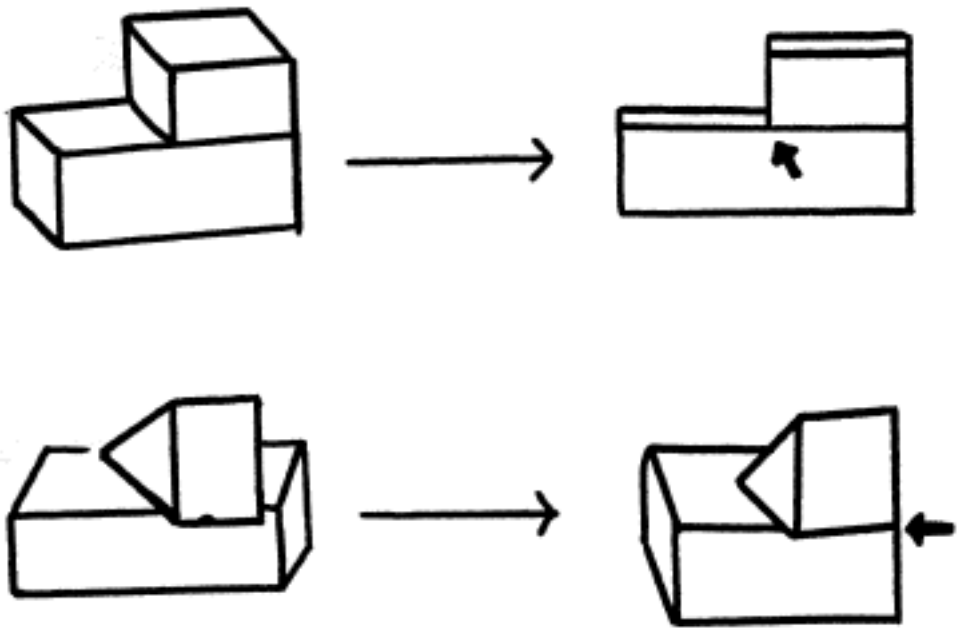


Figure 5. More degenerate Ts.

The third way to improve performance is more radical, difficult, and potentially powerful. Rather than organize the system in the vertical chain of figure 6, one would marry region partitioner and object identifier When the object identifier finds a proposed group of regions unseemly, control would pass to a routine that triesto split the group into more palatable subgroups which are again checked by the object identifier. In the case of figure 1, such a program would have a good hint. Notice that the linkage pairs that bind up 1-2-3, 4-5-6, and 7-8-9 all occur across a single line as follows:

But the linkage pair that binds 3-7 and causes the trouble involves two separated lines. This seems weaker. Noticing this, the regrouping program would correctly propose 1-2-3, 4-5-6, and 7-8-9 as objects, which in turn would be confirmed by the object identifier.

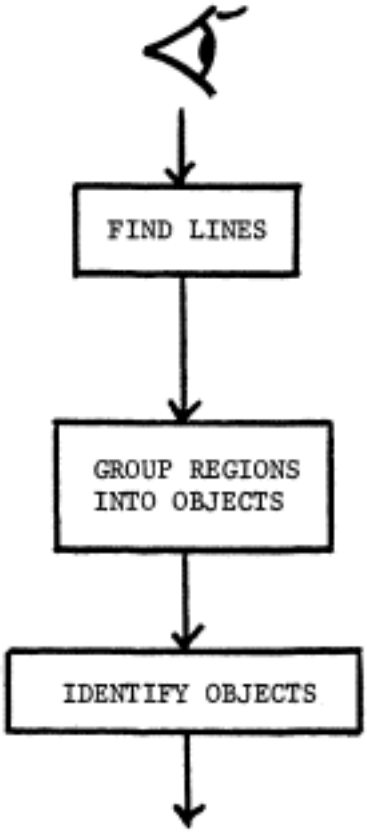


Figure 6. Elementary vision system organization.

This particular heuristic is not useful in the case of figure 2a since the same kind of joint both correctly binds together the wedge and incorrectly binds the wedge to the block. The K joint might save the day here (Guzman's program does not use the K joint at all). I think the K offers evidence that region 3 should be bound to either 2 or 4 but not both. Given this, the regrouping program would try two alternatives. One would be to add a link between 2 and 3 while breaking the link between 3 and 4. This brings no improvement. The other possibility would be to add a link between 3 and 4 and break those between 2 and 3. Victory. The region identifier discovers a wedge and a brick.

The regrouper should also be equipped to try binding regions on the strength of a common edge, particularly when this involves a region that is otherwise unbound. This would allow the object of figure 7 to be identified as a wedge rather than two single-region objects.

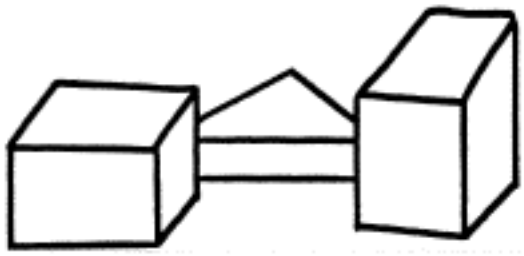


Figure 7. A troublesome wedge.

Remarks on the nature of an object identifier must be even more mystical at this point. One or two not very well substansiated views may be worth mentioning however.

First, I do not think extension of occluded lines is a very good way to identify obscured objects. For one thing, there are cases in which the line extender would have to propose more than one extension possibility. Figure 13, to be seen in Part II, displays an example of such a scene. Secondly, reasonably extended lines often do not help much. Figure 8 looks like two blocks, but it is difficult to imagine how line extension could contribute to that conclusion.

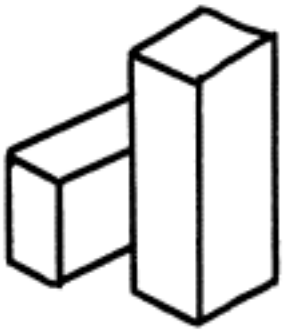


Figure 8. Two blocks.

Instead, I think recognition should be done by a description matching routine capable of matching observed features against models while excusing absent features that may logically be obscured.

Finally, these models might be arrayed in a preference list. Higher positions on the list would be reserved for several object types: simple objects; objects the program is told to look for; and objects already found in the surround. An unknown would be compared to these in turn. Thus the obscured object in figure 9 could be identified as a wedge or cube depending on which the program is looking for or which predominates in the rest of the scene.

If no match is found on the list, a regrouping of regions could be tried or the object could be described as a near match with some flaw.

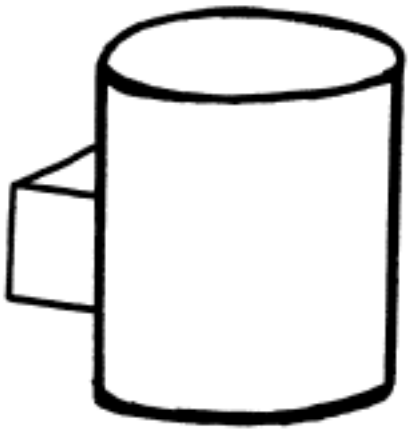


Figure 9. Use of the preference list.