414

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

PROJECT MAC

Artificial Intelligence
Memo No. 96

Memorandum MAC-M-308
May, 1966

POLYBRICK: ADVENTURES IN THE DOMAIN OF PARALLELEPIPEDS

A world without perspective

Adolfo Guzmán

A collection of programs tries to recognize, each one more successfully than its predecessor, 3-dimensional parallelepipeds (solids limited by 6 planes, parallel two-by-two), using as data 2-dimensional idealized projections.

Special attention is given to the last of those programs; the method used is discussed in some detail and, in the light of its success and failures, a more general one is proposed.

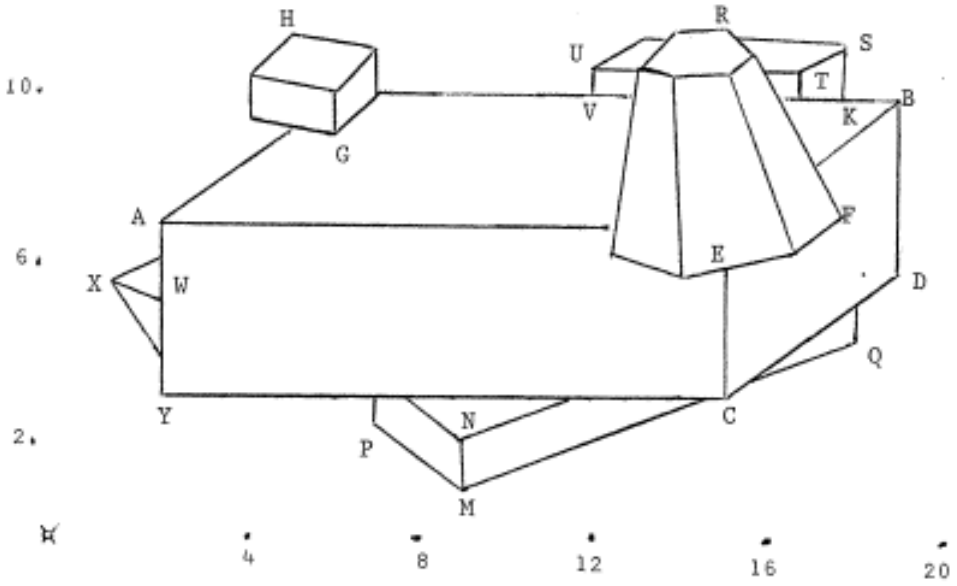


Fig. 1. The machine has to determine how many cubes there are and where are they.

Introduction. We would like the computer to solve the following problem:

> Here is a figure -a picture-.
> What do you see? I am in particular interested in tables, books, and boxes which have in one side written the sign "DANGER".
> Tell me where they are.

The programs described here solve the following problem:

> A picture has been taken in a factory of parallelepipeds (hereafter called sloppily "cubes"). All of them are black, and the only way to differentiate them is because their edges are white and bright. They are also small, so they do not show perspective. The picture looks like fig. 1.
> Noise is not present --all the edges are sharp--, but an extraneous object, which human beings recognize as a 'pyramid', is present, and makes the problem harder [or more interesting].
> Problem: tell me how many cubes we have here and which they are (fig. 1).

A good answer would be something like:

CUBE 1 IS G H

CUBE 2 IS U V T S K

CUBE 3 IS A C B E D

CUBE 4 IS (MAYBE) X Y W

CUBE 5 IS N M Q P.

A wrong answer would be

CUBE 1 IS G H

CUBE 2 IS A W Y X

CUBE 3 IS E R F

CUBE 4 IS U V T S Q

CUBE 5 IS P K B N

An answer is considered bad when it misses some cube, or if it confuses them. On the other hand, ambiguous cubes or partially-identified

ones should be reported as such. The program should also give the position of such cubes, to the extent such information is available.

Input to the program

Eventually the program will read its data directly from the screen. Right now, the picture is transformed (by hand) to a list of corners and points of intersection (real or virtual), and their coordinates in the picture*, together with its nearest adjacent points. William H. Henneman ** and Bill Mann are working on a program which will produce this list, reading the figure from the PDP-6 scope or vidisector, so as to eliminate this manual (and tedious) step.

For example, the input associated with fig. 2 is

```
(A (B F)  B (A G C)  C (B D)  D (G E C)  E (F D)  F (A E G)  G (G D B)  )
 [1 2]    [2, 1]     [4, 1]   [4, 2]     [3, 3]   [1, 3]     [2, 2]
```

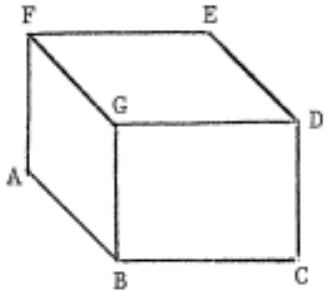


Fig. 2.
A cube showing its vertices.

GORDO (name given to fig. 3) is described by the following list:

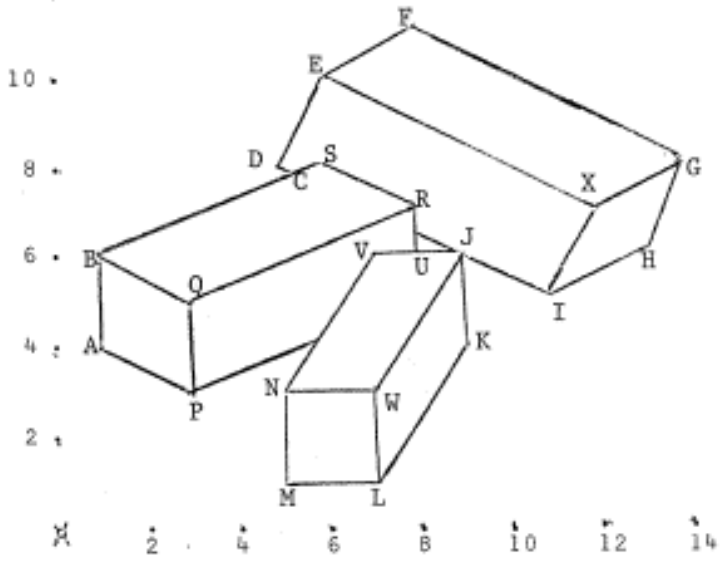
* that is, two-dimensional
** and Elaine Gord

(A 1.0 4.0 (B P) B 1.0 6.0 (A Q C) C 5.4444444 7.77777778

(B D S)
D 5.0 8.0 (E C) E 6.0 10.0 (D X F) F 8.0 11.0 (E G)
G 14.0 8.0 (X H F) H 13.0 6.0 (I G) I 11.0 5.0 (X H J)
J 9.0 6.0 (I T U W K) K 9.0 4.0 (J L) L 7.0 1.0 (K W M)
M 5.0 1.0 (N L) N 5.0 3.0 (O W M) O 5.7272727 4.0909091 (V N P)
P 3.0 3.0 (A Q O) Q 3.0 5.0 (B P R) R 8.0 7.0 (Q S T)
S 6.0 8.0 (C R) T 8.0 6.5 (R U J) U 8.0 6.0 (T V J) V 7.0 6.0 (U O)
W 7.0 3.0 (N J L) X 12.0 7.0 (I G E)

The numbers are stored under the property list of each vertex.

GORDO



A	1	4	B	P					
B	1	6	A	Q	C				
C	$5^{4}/9$	$7^{7}/9$	B	D	S				
D	5	8	E	C					
E	6	10	D	X	F				
F	8	11	E	G					
G	14	8	X	H	F				
H	13	6	I	G					
I	11	5	X	H	J				
J	9	6	I	T	U	W	K		
K	9	4	J	L					
L	7	1	K	W	M				
M	5	1	N	L					
N	5	3	O	W	M				
O	5.7213	4.0909	N	P	V				
P	3	3	A	Q	O				
Q	3	5	B	P	R				
R	8	7	Q	S	T				
S	6	8	C	R					
T	8	6.5	R	U	J				
U	8	6	T	V	J				
V	7	6	U	D					
W	7	3	N	J	L				
X	12	7	I	G	E				

Fig. 3. G O R D O

To its right is its description list, the input to the program.

Format of the answer. We use the CONVERT processor and apply the function cubs (in the file 'CUBE LISP') to the picture GORDO (in file 'gordo'). Here is the operation in CTSS.

```
load ((a cube gordo))
  (CERO UNDECLARED)
  (CERO UNDECLARED)
 NIL
                              e (cubs gordo)
 (THERE ARE AT LEAST 3 OR 3 CUBES)

 (CUBE 1 IS (N (W O M) W (N J L) L (M K W))   )

 (CUBE 2 IS (I (J H X) G (F X H) X (E G I) E (X F D))   )

 (CUBE 3 IS (P (A O Q) R (S Q T) Q (B R P) B (Q C A))   )
```

THE PROGRAMS.

They are written in CONVERT, a pattern-driven symbolic transformation language [1], and we will discuss here the following:

CUBES2	LISP	000	Original, uses continuity.
CUBS	LISP	000	Partitions the set into disjoint classes.
CUBA	LISP	000	Final version; uses the unit distance notion
CUBE	LISP	000	Breaks into (not connected).

The last one is the one currently in use, but it is interesting to talk about all of them.

CUBES2 LISP. Use of neighborhood.

The idea is to find a corner; if found, to find a "Square" (parallelogram); if found, to find the cube.

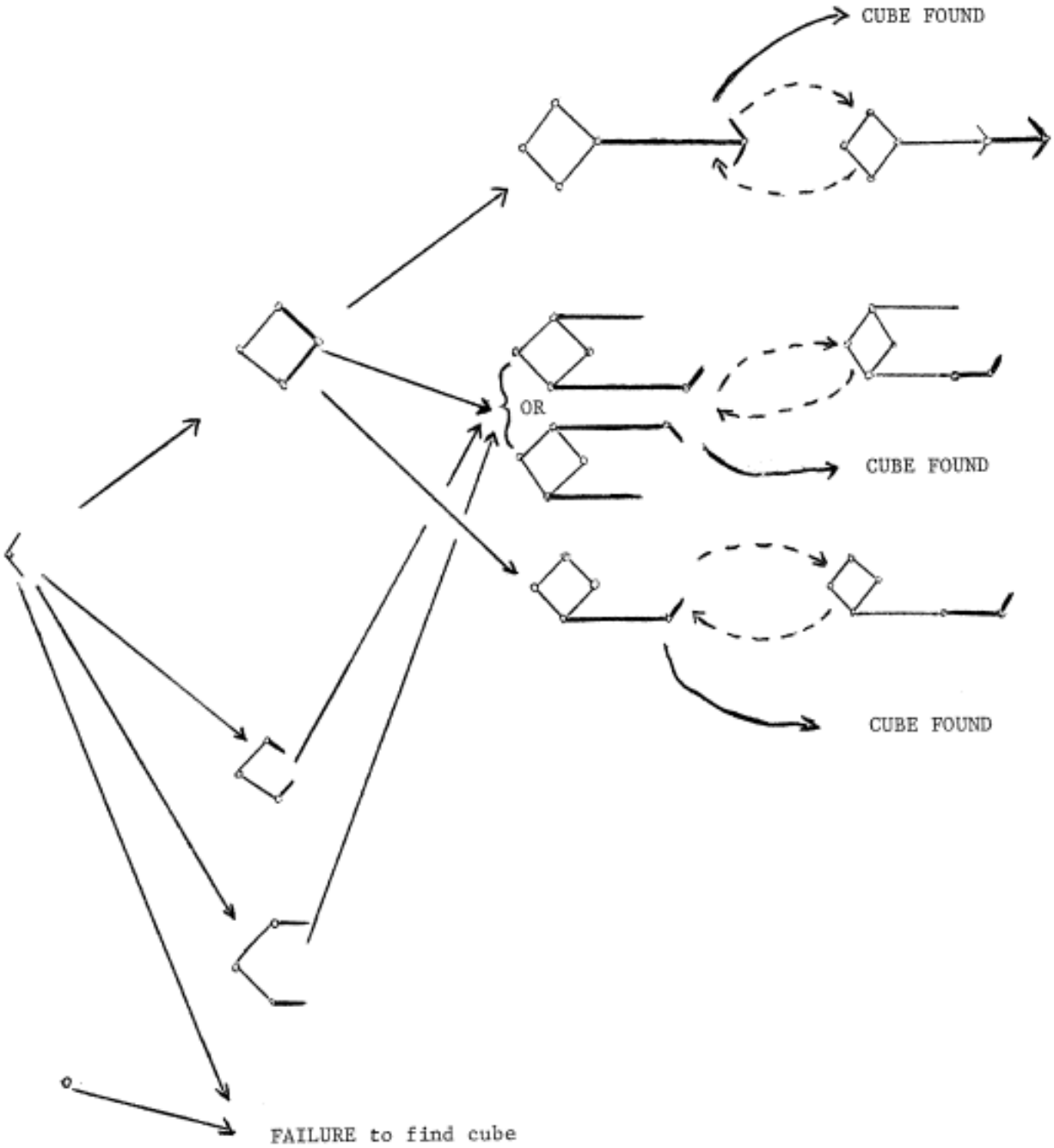
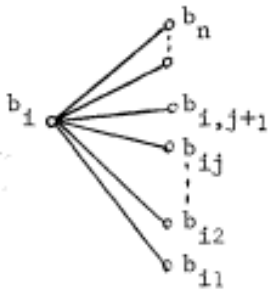


Fig. 4. Flow of control of CUBES2.

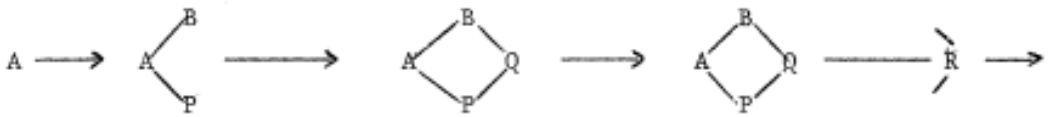
The program is best explained in the chart of Fig. 4. If a corner ( ) is found, we look for a parallelogram ( ) which has that corner (we use here the information about which points are joined to which); as usual, solid arrows in the flow chart indicate the direction of success; broken ones, the direction of failure.

I am using the following (opposite to the normal) convention in the tree structure: if I fail completely in $b_{i,j+1}$, I go down one branch and try to match $b_{i,j}$. For example, if I fail to recognize a , I go down and try to find a .



When a cube is found, it is completed; that is, the program now tries to "fill it" and to recognize all the vertices (points) belonging to it. Then, it reports this cube, and erases it from the picture (it deletes all its lines, but only vertices that it is sure are not shared by other cubes) and tries again with the remaining part of the picture.

For example, in GORDO (fig. 3), CUBES2 proceeds in this way:



Now it tries again; it finds all the 3 cubes.

C O M M O N

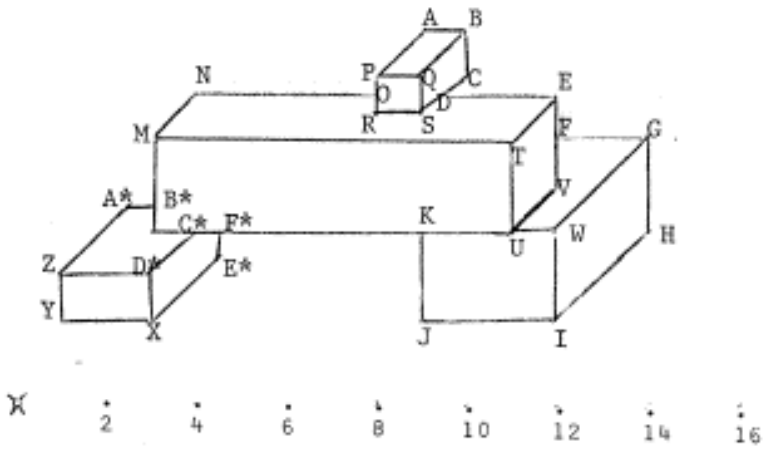


Fig. 5. C O M M O N

What to erase and what to leave

Once all the points of a cube are found, we have to delete it from the picture, in order to process the remainder. Or, if you do not want to delete points, you still have to mark them as "already processed". This process is explained with COMMON, the example in fig. 5.

Once the cube K J I W U V F G H is found, we delete these points from the graph. The point G, for example, is safely deleted, since its neighbors, F, H and W also belong to the encountered cube. But F, for example, is still not deleted, since it has as neighbors points outside (not belonging to) the actual cube. Therefore, one pass through the graph eliminates all the lines arriving at points in the cube; for example, F* (E* C* K) is transformed to F* (E* C*), since K was in the cube. In this way we delete the line F* - K, if we also make the transformation from K (U J F*) to K (U J).

Another pass looks for points of the form W ( ), that is, points "isolated" (not connected to anything else), and deletes them.

The first pass is done with the CONVERT rule

[(XXX (YYY U ZZZ) WWW) (XXX (*REPT* ((YYY ZZZ) WWW)))]

where we define U as "member of CUBEJUSTFOUND".

The second pass --deletion of isolated points-- is done with

[(XXX X ( ) YYY) (XXX (*REPT* (YYY)))].

In this way points shared by several cubes (like K) are preserved. But not the lines; for example, the line K - U is erased (fig. 6), because it belonged to the cube K J I W U V F G, even if it also belongs to the

cube M N P D E F V T.

In general, there is no way to predict such an event, since the second cube has not yet been found, and therefore there is no way to tell what its parts are. We will discuss this point later.

In general, this is not a serious defect, but see the example TRICKY, fig. 9.

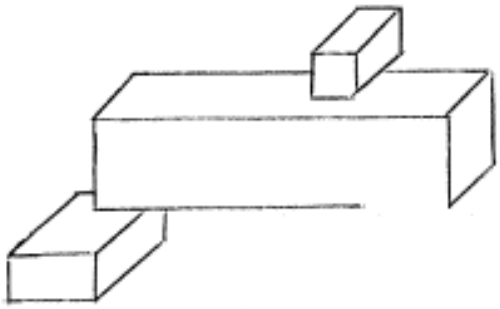
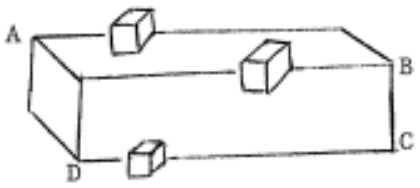


Fig. 6. COMMON after erasing cube K J I W U V F G.

Shortcomings of CUBES2.

The scheme just presented gives an idea of the power or weakness of CUBES2. It is able to find connected cubes; for example, it solves GORDO (Fig. 3) and COMMON (Fig. 6), but it fails to find A B C D in the figure at the right because it is formed of two disconnected parts (disconnected in the sense that, in order to go from one part A D to the other B C, we have to cross other cubes).



CUBS LISP. Classification of the corners.

We want to be able to recognize "disconnected" cubes, and the way CUBES2 (fig. 4) works does not allow us to do that. Roughly speaking, the problem is this: in some way I manage to know that A Q C looks like it is going to be a cube (see also fig. 7), so I would like to look for a corner of the form in the direction Q C. That corner happens to be U W T V at the bottom, but in order to find it I have to continue the line Q C for a while, and stop after finding W T, which is the contintuation.

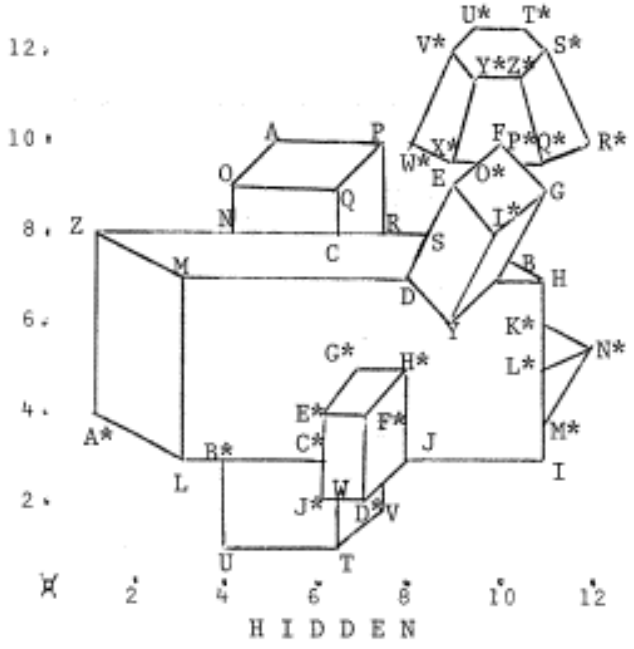


Fig. 7. The cubes A Q V U T and A* H are disconnected.

We could use the scheme of trying to extend all lines that seem to be stopped --like QC, TW--, making the picture somewhat transparent. Also,

when looking for corner T, we could extend slowly the line Q C, and every 2 millimeters or so ask: Have I hit a point yet?

Instead of that, we use the opposite approach: look for the points (corners) which exist, and see which of them may be continuations of Q C. But it would be better not to look at all of them, but just to the most promising ones. That is what CUBS does.

The vertices may be CORNERS, Y's, T's or ANY's.

The program classifies the vertices of the picture into several categories:

CORNERS: With this name we denote vertices at which two lines arrive, for example U, A, I, R*, etc. in HIDDEN (fig. 7).

Y's: Three lines meeting in a point, no two of which are parallel (colinear). Z, T, N*, Q.

T's: Three lines meeting at a point, two of them collinear; B*, W, K*, L*, M*.

ANY's: Vertices having more than 3 lines.

What the program CUBS does is divides the vertices into CORNERS, Y's, T's and ANY's. The Y's are also classified into classes, according to the slope of its sides.

After this, all the Y's of a particular cube can be found in a given class; if it happens that there are no parallel cubes, like in

STICKS (fig. 8), then you simply print the classes, because each class contains exactly one cube.

That is not the complete solution. There is more to be said, of course. When a single class contains just one vertex, such as G or M* (STICKS, fig. 8), it may or may not be part of a cube. CUBS makes further analysis and depending upon the kind of vertices attached to the lines forming that Y, an acceptance or rejection is made. For information purposes, a message "FALSE CUBE FOUND" is issued.

For example, analyzing the points attached to H, XM and F, the "Y" G is accepted as a cube; analyzing the points N*, T* and Z*, the point M* is rejected, that is to say,

is not part of a cube.

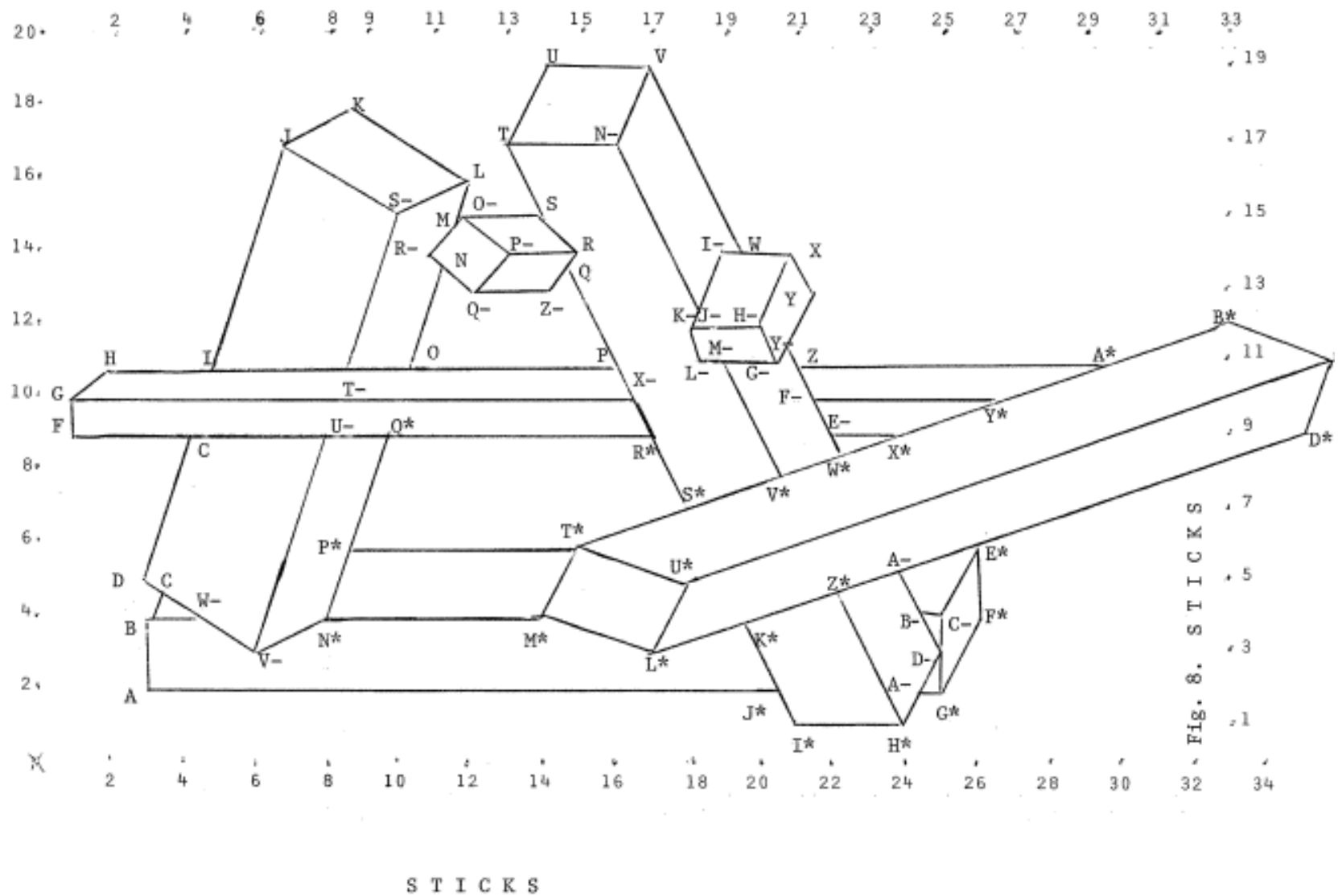


Fig. 8. STICKS

This is the solution for STICKS, as the program CUBS does it:

```
(CORNERS == ZM (QM Q) RM (M N) LM (KM MM) IM (W JM) I* (H*
J*) F* (E* G*) D* (C* E*) B* (C* A*) Y (X YM) U (T V) K (J
L) H (G I) F (G E) D (C E) A (B J*))

(TES = A/ (H* DM G*) YM (GM Z Y) XM (P R* G) WM (C B VM) UM
(Q* E VM) TM (SM I O) MM (LM GM V*) JM (KM IM NM) FM (Z EM
Y*) EM (W* FM X*) BM (AM CM DM) AM (Z* BM E*) Z* (K* AM H*)
Y* (FM X* A*) X* (EM W* Y*) W* (EM X* V*) V* (S* MM W*) S*
(R* T* V*) R* (Q* S* XM) Q* (UM P* R*) P* (N* T* Q*) K* (L*
Z* J*) J* (K* I* A) A* (Z Y* B*) Z (YM A* FM) W (IM V X) Q
(ZM P R) P (Q XM O) O (N TM P) N (RM QM O) M (RM OM L) I (H
J TM) E (F D UM) C (D B WM))

(FALSE CUBE ( 0.30000002E1  0.5E0 -0.0 N* (VM P* M*)))

(FALSE CUBE ( 0.2E1  0.0 -0.33333333 M* (T* L* N*)))

(CUBE 1 IS (U* (T* L* C*) L* (U* M* K*) C* (B* U* D*)))

(CUBE 2 IS (KM (JM HM LM) HM (KM X GM) GM (MM HM YM) X (W Y
HM)))

(CUBE 3 IS (NM (T V JM) H* (A/ Z* I*) V (U NM W) T (S NM U)))

(FALSE CUBE ( 0.0 -0.1E1 -0.2E1 S (T OM R)))

(CUBE 4 IS (QM (N PM ZM) PM (OM QM R) OM (M PM S) R (S PM Q)))

(CUBE 5 IS (VM (WM N* UM) SM (J L TM) L (SM K M) J (I SM K)))

(CUBE 6 IS (G F H XM))

(CUBE 7 IS (CM (BM DM E*) G* (DM F* A/) B *A WM C)))

(ANYS = DM (BM CM A/ G*) T* (S* U* M* N*) E* (AM D* CM F*))
```

We print, as additional information, the CORNERS and the T's. Note that only a small part of each cube is printed; for example, of the long horizontal cube, only vertices G, F, H and XM are printed. It is not

difficult to "fill" the cube, as CUBES2 does, but CUBS does not do that, (if for no other reason, because we already know how to do it, so it is just a matter of adding that part of the program).

Also, CUBS does not use any information about CORNERS; again, in more complicated cases we need it, and a complete program should have it.

Shortcomings of CUBS.

I think the most serious one is that it is unable to make recognition among parallel cubes, for example cubes A Q U T and G* F* J* H* in "HIDDEN" (fig. 7) are confused and reported as just one, since they lie in the same class. A better (or worse) example is COMMON (fig. 5), where all the four cubes are parallel, and the program thinks there is just one. Also, the program does not check for length of edges.

Let us not get angry at CUBS. It is obvious that the program is incomplete, and it is also obvious what should be done.

The main good idea in CUBS is that, by dividing the cubes into classes, we transform the problem of finding all the cubes, into the problem of finding the cubes in a given class, in which all of them are parallel. This approach also solves the disconnectivity problem.

Discussion of the program CUBS

I want at this point to discuss the program in considerable detail, and to see how it achieves its goals. If the reader does not want to

follow me, he is welcome to skip to page 24, where we talk about CUBA and CUBE, the actual (and more complicated) programs in use.

The input. In this page we present a list corresponding to figure STICKS in page 15. This list, without the numbers, is used as argument E of the function CUBS. [as a detail, in the figure appear points such as A-, B-, etc., which are named AM, BM, etc., in the list]. Actually it is convenient to have this list in a separate file because often we want to make additions or modifications.

```
CSET (STICKS (A 3. 2. (B J*) B 3. 4.(A WM C)
C 3.3749999 4.75 (D B WM) D 3. 5. (C E) E 4.3333333 9.(F D UM)
F 1. 9. (G E) G 1. 10. (F H XM) H 2. 11. (G I)
I 5. 11. (H J TM) J 7. 17. (I SM K) K 0. 18. (J L)
L 12. 16. (SM K M) M 11.5 14.5 (RM OM L)N 11.25 13.75 (RM QM O)
O 13.333333 11. (N TM P) P 16. 11. (Q XM O) Q 14.666666
13.6666666 (ZM P R) R 15. 14. (S PM Q) S 14. 15. (T OM R)
T 13. 17. (S NM U) U 14. 19. (T V) V 17. 19. (U NM W)
W 19.5 14. (IM V X) X 21. 14. (W Y HM) Y 21.2 13. (X YM)
Z 21. 11. (YM A* FM) A* 30. 11. (Z Y* B*) B* 33. 12. (C* A*)
C* 36. 11. (B* U* D*) D* 35. 9. (C* E*) E* 26. 6. (AM D* CM F*)
F* 26. 4. (E* G*) G* 25. 2. (DM F* A/) H* 24. 1. (A/ Z* I*)
I* 21. 1. (H* J*) J* 20.5 2. (K* I* A) K* 19.571428 3.8571429
(L* Z* J*) L* 17. 3. (U* M* K*) M* 14. 4. (T* L* N*)
N* 8. 4. (VM P* M*) P* 8.666666666 6. (N* T* Q*) Q* 9.666666666
  9. (UM P* R*) R* 17. 9. (Q* S* XM) S* 18. 7. (R* T* V*)
T* 15. 6. (S* U* M* N*) U* 18. 5. (T* L* C*) V* 20.571428
7.9571429 (S* MM W*) W* 22.285714 8.4285717 (EM X* V*)
X* 24. 0. (EM W* Y*)  Y* 27. 10. (FM X* A*) Z* 22.142857
4.7142858 (K* AM H*) AM 23.857143 5.2857141 (Z* BM E*) BM 24.5 4.
(AM CM DM) CM 25. 4. (BM DM E*) DM 25. 3. (BM CM A/ G*) EM 22. 9.
(W* FM X*) FM 21.5 10. (Z EM Y*) GM 20.2 11. (MM HM YM)
HM 20. 12. (KM X GM) IM 19. 14. (W JM ) JM 18.25 12.5 (KM IM NM)
KM 18. 12. (JM HM LM) LM 18.2 11. (KM MM) MM 19. 11. (LM GM V*)
NM Lç. L&. (T V JM) OM 12. 15. (M PM S) PM 13. 14. (OM QM R)
QM 12. 13. (N PM ZM) RM 11. 14. (M N) SM 10. 15. (J L TM) TM 8.666666666
 11. (SM I O) UM 8. 0. (Q* E VM) VM 6. 3. (WM N* UM)
WM 4.5 4. (C B VM) XM 16.5 10. (P R* G) YM 20.6 11.8 (GM Z Y) ZM 14. 13.
                    (QM Q)
A/ 24.5 2. (H* DM G*)
```

```
A/ 24.5 2. (H* DM G*)
(LAMBDA (X) (CSET X (LLENA STICKS))) (STICKS)
STOP ))) ) ) )) ) )  )) ) )  ) ) ) )  )) ) )))  ) ) ))
```

This is how the figure STICKS looks like, in list format. At the end we call to LLENA, a LISP function which simply takes the coordinates out of the list and puts them in the property list of the corresponding atom, under XCOR and YCOR, and then gives this expression to the CONVERT program.

The program. The CUBS program is rather short. We start by defining CUBS, a LISP function which calls CONVERT. It has one argument, the (list representing the) picture we want to analyze. The program appears in the next page.

The first argument of CONVERT is the dictionary of declared variables. Let us analyze it:

GR	PAV	GRR	GR will match with anything that GRR matches, and will retain that value.
GRR	STG	X	GRR will match with a number if it is strictly greater than (the number matched by) X.

```
(CUBS (LAMBDA (E) (CONVERT (QUOTE(
GR   PAV   GRR       GRR  STG  X          (CCC)  CNT  1
ORD   REPT ( ((XXX X YYY GR WWW) (=REPT= (XXX GR YYY X WWW)) ))
                                  X*   PAT   (=XEC= EQUAL1 == X)
                                  Y*   PAT   (=XEC= EQUAL1 == Y)
                                  Z*   PAT   (=XEC= EQUAL1 == Z)
Z1   SKEL  (=PRNT= =BLNK=)        NUM  SKEL  (=SETQ= X1 (=INCR= X1))
TG   REPT  ( ((X) (=EXEC= TAN X X1) ))
YZW  PAT  (=OR= Y Z W)
)) (QUOTE(  Y  X  Z (XXX) (YYY) (WWW) W  ))   E  (QUOTE( C1(

( == (=PROG= (X1 SAME (ANY)(YS)(CRS)(TS)Y1)    (=SETQ= SAME =SAME=)
1(=WHEN= SAME(X Y XXX)((=SETQ= X1 X)(=SETQ= Y1 Y)(=SETQ= SAME(XXX)))
                                                  (=GOTO= 2))
(=REPT= Y1 C2 (
     ( (== ==) (=SETQ= (CRS) (X1 Y1 CRS)) )
     ( (X Y Z) (=REPT= ( (YS) ((TG X)(TG Y)(TG Z)) ) C3 (
          ( =PRI= (C3 X1  =BLNK=))
          ( (== (=== X === X* ===)) (=SETQ= (TS) (X1 Y1 TS)) )
          ( (X (Y Z W)) (=NEXT= ((ORD Y Z W) X)) )
          ( ((X Y Z)()) (=SETQ- (YS) ((X Y Z X1 Y1))) )
          ( ((X Y Z) (XXX(X* Y* Z* YYY)WWW)) (=SETQ=(YS)(XXX(X Y Z X1 Y1
                                                    YYY) WWW)) )
          ( ((XXX)==) (=SETQ= (YS) ((XXX X1 Y1) YS)))  )))
     ( == (=SETQ= (ANY) (X1 Y1 ANY)))  ))
(=GOTO= 1)     2 Z1  (=SETQ= X1 0)
(=PRNT= (CORNERS == CRS))  Z1 (=PRNT=(TES = TS))
3 Z1
(=COND= (YS) (X XXX) (=GOTO= (=QUOT= 3) (=SETQ= (YS) (XXX))
     (=PRNT= (=REPT= (X (CRS TS)) C4 (
          ( ((== == == Y Z W XXX)==)  (CUBE NUM IS (Y Z W XXX)) )
          ( ((== == == X (Y Z W)) (=== YZW === YZW === YZW ===))
               (CUBE NUM IS (X Y Z W)) )
          ( (X ==)  (FALSE CUBE X)) ) ))   ))
(=RETN= (ANYS = ANY)) ))  ) ))    )))
```

The CUBS program.

(CCC) CNT 1 CCC will match any fragment with length 1 (or longer), and it will remember this number; for example, (A CCC B CCC D) requires that between A and B and between B and D there must be two fragments of equal length ($\geq$ 1).

ORD REPT (((XXX X YYY GR WWW) (=REPT= (XXX GR YYY X WWW))))
The value of (ORD ZZZ) is a list which contains the same elements of ZZZ, but in increasing order --we suppose ZZZ formed by numbers--. The unique rule used says: if you see X and in some place to its right some other number bigger than X, (XXX X YYY GR WWW), interchange and then repeat the process: (=REPT= (XXX GR YYY X WWW)).

X* PAT (=XEC= EQUAL1 == X) X* will match a number N for which (EQUAL1 N X) is true. X* is approximately equal to X. Note that =XEC= allows us to call LISP predicates, as in this case; == is replaced by the expression being matched. Y* and Z* are approx. equal to Y and Z, respectively.

Z1 SKEL (=PRNT= =BLNK=) Its value is a BLANK, which is also printed. This is a way to print a line of blanks, in the console of the time sharing system.

TG REPT (((X) (=EXEC= TAN X X1))) TG applies the LISP function TAN to its argument and X1.

YZW PAT (=OR= Y Z W) YZW will match with any expression matching either the pattern Y or the pattern Z or the pattern W.

The second argument declares X, Y, Z and W as free variables, and XXX, YYY and WWW as free fragment variables, that is to say, in the mode UAR.

The third argument is E, the picture we are going to examine.

The last argument of CONVERT is a list of collections of rules; in this case there is just one, called C1.

C1 has also one rule, which says [== (=PROG= ( ) ...)]

"If you see an expression (E) matching ==, replace the skeleton

(=PROG= ...)". Since == matches anything, no matter what is the expression E, we enter the =PROG=. The =PROG=ram feature for CONVERT is described in A.I. Memo 95.

We see that (X1 SAME (ANY)(YS)(CRS)(TS)Y1) declares to X1, Y1 and SAME as being =PROG= variables, and to ANY, YS, CTS and TS as being =PROG= fragment variables.

(=SETQ= SAME =SAME=) We assign to SAME the value of =SAME=, that is, the expression which we are trying to analyze.

The statement No. 1 of the =PROG= says: when SAME is decomposable in 1st, 2nd and remainder, or (X Y XXX), store the first expression in X1, the second in Y1 and the remainder in SAME. So, X1 will contain a vertex, and Y1 is the list of the points connected to it. The only case when SAME does not match (X Y XXX) is when it is empty, in which case we go to 2.

Now we apply to Y1 the rule-set C2, and they go to 1, to do the same with other vertices.

[(== ==) (=SETQ= (CRS) (X1 Y1 CRS))] Corners are simply added to CRS.

[(X Y Z) (=REPT= ((YS) ((TG X)(TG Y)(TG Z))) C3 ...)]
If we see a triplet, we compute the slope of its sides, and we form a list of (YS) and them, and apply to such an entity the rule-set C3.

The first rule of C3 prints C3, X1 and a blank.

The second rule of C3 looks to see if two of the slopes are equal; this being the case, we are dealing with a T, so it is added to (TS).

[(X (Y Z W)) (=NEXT= ((ORD Y Z W) X))] Otherwise, we order our slopes, interchange (YS) to the right and go on to the next rule.

```
[ ((X Y Z) ())  (=SETQ= (YS)  ((X Y Z X1 Y1))) ]  If (YS) is empty,
```
we initialize it in the way described.

```
( ((X Y Z) (XXX(X* Y* Z* YYY)WWW)) (=SETQ=(YS)(XXX(X Y Z X1 Y1
                                                    YYY) WWW)) )
```
says: if there exists in (YS) a class with the same slopes as those matched in (X Y Z), we put the present vertex, X1 Y1, into that class.

```
( ((XXX)==) (=SETQ= (YS) ((XXX X1 Y1) YS)))
```
Otherwise, we create a new class. That finishes C3.

The last rule of C2 says [== (=SETQ= (ANY) (X1 Y1 ANY))]: otherwise --that is, if Y1 is not a triplet-- append it to ANY. Then, go to 1.

When we finish this preprocessing of the picture, CRS contains all the corners, TS all the T's, etc. We print them. (See example of output, page 16). In particular, (YS) looks like

((2 0 -2 H*(I* A/ Z*) T(U NM S) ...) (2 1/3 - 1/3 U*(L* C* T*) ...) . . .),

divided into classes with the three numbers --slopes-- at the front.

Now we execute statement 3, other loop, it starts by printing a blank, Z_1.

```
3 Z1
(=COND= (YS)  (X XXX)  (=GOTO= (=QUOT= 3)  (=SETQ= (YS) (XXX))
       (=PRNT= (=REPT= (X (CRS TS)) C4 (
            [ ((== == == Y Z W XXX)==)  (CUBE NUM IS (Y Z W XXX)) ]
            [ ((== == == X (Y Z W)) (=== YZW === YZW === YZW ===))
                 (CUBE NUM IS (X Y Z W)) ]
            [ (X ==)  (FALSE CUBE X) ] ) ))    ))
```

This conditional says: if (YS) has one element X and a remainder XXX, go to 3, but first store (XXX) in (YS), and print some other skeletons. When (YS) is ( ), this (=COND= ...) fails, in which case we execute the next statement: (=RETN= (ANYS = ANY)), finishing the computations.

The set C4 applies to (X (CRS TS)), that is, to a list formed by the class X of vertices and a list consisting of the corners plus the T's, the following rules:

```
[ ((== == == Y Z W XXX)==)  (CUBE NUM IS (Y Z W XXX)) ]
```

If the class consists of at least 3 elements, besides its 3 numbers (matched by == == ==), that is, if the class contains more than 1 vertex, we print that class as a cube. No attempt is made to differentiate between parallel cubes, which fall into the same class.

```
[ ((== == == X (Y Z W)) (=== YZW === YZW === YZW ===))
     (CUBE NUM IS (X Y Z W)) ]
```

If the class consists of only one vertex, X (Y Z W), we examine more closely the points Y, Z and W, to see if they are either corners or T's; if this is the case (as it is G (F H XM) in STICKS, fig. 13) we accept X as a cube, otherwise we reject it:

```
[ (X ==)  (FALSE CUBE X) ]
```

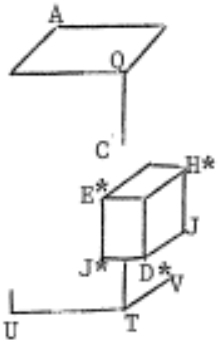
The program uses some simple LISP functions:

```
(TAN (LAMBDA (A B) (COND
((EQUAL (GET B (QUOTE XCOR)) (GET A (QUOTE XCOR))) 0.1E7)
(T (QUOTIENT (DIFFERENCE (GET B (QUOTE YCOR)) (GET A (QUOTE YCOR)))
             (DIFFERENCE (GET B (QUOTE XCOR)) (GET A (QUOTE XCOR)))
  ))) ))

(EQUAL1 (LAMBDA (A B)(LESSP (ABSVAL (DIFFERENCE A B)) CERO) ))

(ABSVAL (LAMBDA (A) (COND ((MINUSP A) (MINUS A)) (T A) )))  .
     CSET (CERO 0.003)
```

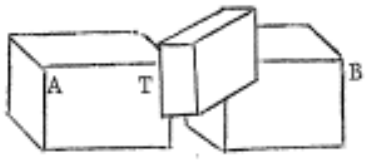
CUBA LISP. Differentiating among parallel cubes.

The program just discussed takes a figure and separates the cubes into classes, each one of them containing parallel cubes. For example, in HIDDEN (fig. 7, page 12) the cubes A Q V U T and E* J* D* J H* are parallel. We would like to differentiate among them. Here

we use the collinearity among two vertices; for example, Q and T are collineal --figure at the right--, but Q and D* are not, so Q and D* can not form a cube.



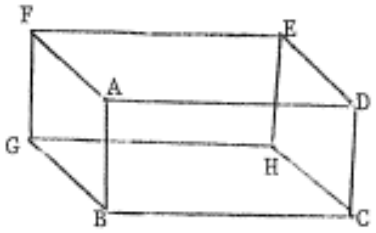
Also, we do not want to compare Q with all the vertices of its same class in order to select the possible ones; it seems that a further classification of vertices of the same class is desirable. Collinearity is not sufficient. For example, vertices A and B --see figure below-- are collineal, and still do not form a cube; therefore, we will select all the vertices collineal to A in the direction A T and (if there are some) select the appropriate one.



As the last figure shows, we have to take into account more information about the lines, in order to know that cube A T ends before [to the left of] vertex B. As an alternative, we could say: "A T does not give enough information about the remainder of this cube, so we may as well forget about A T and try another line, with the expectation that we will be luckier this time".

Numbering the Y's. Unit distance vertices.

Take a cube, pick any vertex and establish the three directions of its lines, as done in the figure. Now, examine for each vertex, the lines which depart from it. For vertex A, all its lines depart in the positive directions ↖ , → and ↓ ;

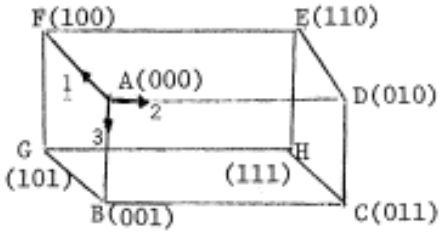


these 3 directions were arbitrarily chosen to be (+), (+), (+).

therefore, is (+ + +) or (0 0 0) or). For vertex B,

line B G is ↖ (+)
line B C is → (+)
line B A is ↑ (-) ;

therefore, vertex B is (+ + -) or (0 0 1) or 1. When we finish this process, our cube now looks like this:



This numbering scheme is independent of the starting vertex (0 0 0) and of the directions which are considered positive.

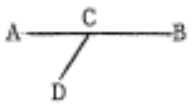
Connected vertices are unit-distant, that is, their binary words differ in exactly one bit. Vertices which are 2 units apart lie on the diagonal of the faces (AE, AG, BH, etc.) and vertices lying in opposite extremes of the diagonals of the cube are 3 units apart, for example F (1 0 0) and C (0 1 1).

Pre-processing. The pre-processing done in CUBA is more complicated than the one done in CUBS.

Vertices are divided into CORNERS, T's, Y's and ANY's (as before);

1. CORNERS are divided according to the slope of the sides.

2. T's are divided according to the slope of the top (A B) and the slope of the tail (C D).



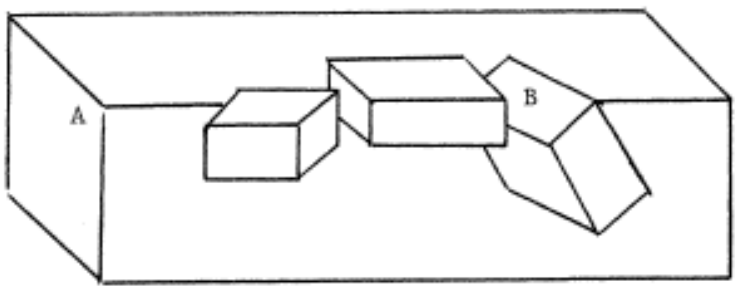
3. Y's are divided into classes, according to slope.

   In each class, vertices are divided according to the unit distance concept. If certain vertex happens to be the first of a given class, the number (0 0 0) is assigned to it.

Localization of the Cubes. A second part of CUBA applies to each class of Y's the following process:

1. A vertex is selected and the program tries to attach to it a cube, if possible; therefore, its unit-distance vertices are looked for [if the vertex in question has number $(x_1\ x_2\ x_3)$, only sub-classes $(\overline{x}_1\ x_2\ x_3)$, $(x_1\ \overline{x}_2\ x_3)$ and $(x_1\ x_2\ \overline{x}_3)$ are searched]; a vertex has to pass the test for collinearity and, if several are found, the closest is chosen. It turned out that these 3 tests are still not sufficient; for example, B is
   (1) unit-distant from A
   (2) colineal
   (3) the closest

   and still A - B is not (part of) a cube. In relation with this, see also TOWER (fig. 11).



2. We apply to the vertices found in (1) the same process (1), up to a certain depth.

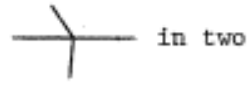
3. The cube formed in this way is accepted if it has 2 or more vertices; if it has only one, as N* (K* L* M*) in HIDDEN, fig. 7, page 11, we check the extreme points [K*, L* and M* in the example], as explained in CUBS.

A fancier program should say, after finding a cube such as N*: "I am not sure it is really a cube, but it looks like one". This comment can be inserted in this part of the program.

4. Accepted cubes are reported and their vertices erased from the subclasses where they were found, and the whole process is applied again to the next vertex of the subclass.

5. When a subclass (or a class) is empty, the next one is searched.

CUBE LISP.

Is the program currently in use; in addition to what CUBA does, it also breaks vertices of the type  in two Y's: and .

Use of CORNERS.

We still do not use information about the corners; the program reports just the Y's of a cube, and does not try to complete it, as CUBES2 does. To complete a cube once is found is not difficult, if we avoid ambiguous or undetermined cases, and for that matter we could use those rules and skeletons which CUBES2 uses for this purpose.

Use of LINES.

No information is used actually about individual lines. A more general program should also classify lines according to its slopes; this

information would allow us to complete the cube G F H X M (STICKS, fig. 8) with its otherwise totally disconnected part Z A* EM X*.

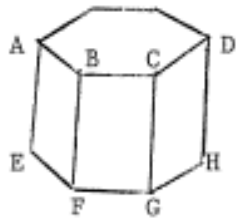
Recognition of Cubes in a Picture which also contains other objects.

In the presence of non-cubic objects, an effort is made by the program to see cubes in them; if none is found, these objects are simply ignored. A good example is HIDDEN (fig. 7, page 11), where the truncated pyramid is ignored, but only after several "false cubes" found in it. The output is the following:

SOLUTION TO H I D D E N

```
(FALSE CUBE (Z* (Q* Y* S*)))
(FALSE CUBE (Y* (V* Z* X*)))
(FALSE CUBE (X* (W* O* Y*)))
(FALSE CUBE (S* (T* Z* S*)))
(FALSE CUBE (Q* (Z* P* R*)))
(CUBE 1 IS (N* K* L* M*))
(FALSE CUBE (X (H Y B)))
(FALSE CUBE (J (I K H*)))
(CUBE 2 IS (H* (G* F* J) E* (F* G* C*) F* (E* H* D*) D* (W K F*))  )
(CUBE 3 IS (P (A Q R) O (Q A N) Q (O P C) T (U V W))  )
(CUBE 4 IS (L (A* B* M) Z (M N A*) M (Z D L) H (B X K*))  )
(FALSE CUBE (V* (Y* U* W*)))
(CUBE 5 IS (Y (D X I*) G (P* I* B) I* (E G Y) E ( I* O* S))  )
(FALSE CUBE (D (Y M S)))
```

If instead of a pyramid we put a hexagonal prism, it will recognize in it the "cubes" A B C E F G and B C D F G H!

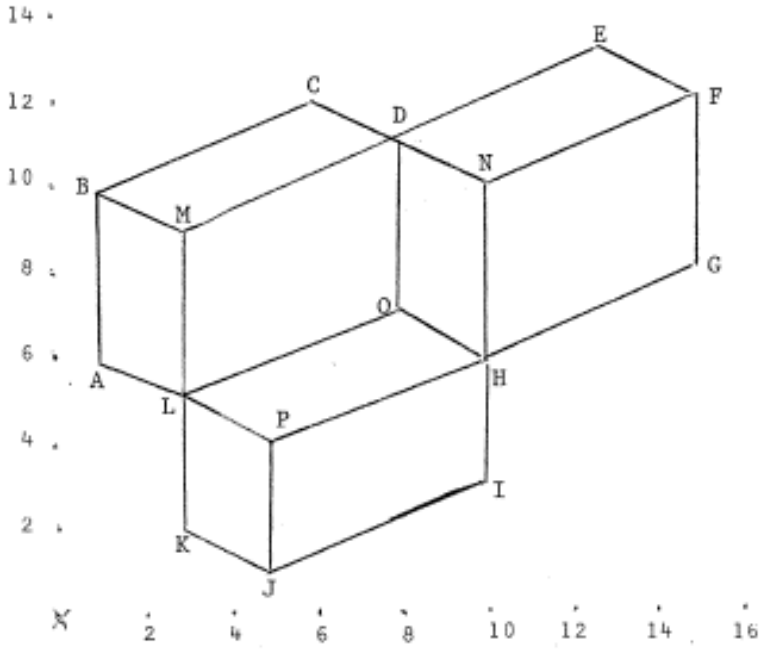


As you see, CUBE is not very successful in a foreign environment. A more general program should be more careful about accepting candidates which look good.

Preliminary guess. After the preprocessing of the figure, the number of CORNERS is computed and divided by 3 (a cube can not have more than 3 CORNERS). The classes of Y's are also counted. Both numbers are inserted in a message "there are N1 or N2 cubes".

Some Examples. We have already shown several figures which the program analyzes correctly; they are COMMON (fig. 5), GORDO (fig. 6), HIDDEN (fig. 7), STICKS (fig. 8). Some of them, like HIDDEN (pg. 12) are somewhat complicated, since they involve parallel cubes, disconnected cubes, 1-corner cubes, extraneous objects, etc.

I would like to present now a couple of examples, TRICKY (fig. 9) and WHAT? (fig. 10), where the answer is ambiguous (non- unique). The program does its best, and its answers are acceptable but, in general, CUBE is not designed to solve optical illusions.



A	1	6	(B L)
B	1	10	(A M C)
C	6	12	(D B)
D	8	11	(O M C N E)
E	13	13	(D F)
F	15	12	(E N G)
G	15	8	(F H)
H	10	6	(O N P I G)
I	10	3	(J H)
J	5	1	(K P I)
K	3	2	(L J)
L	3	5	(A M O P K)
M	3	9	(B L D)
N	10	10	(D F H)
O	8	7	(D L H)
P	5	4	(L H J)

Fig. 9 T E I C K Y

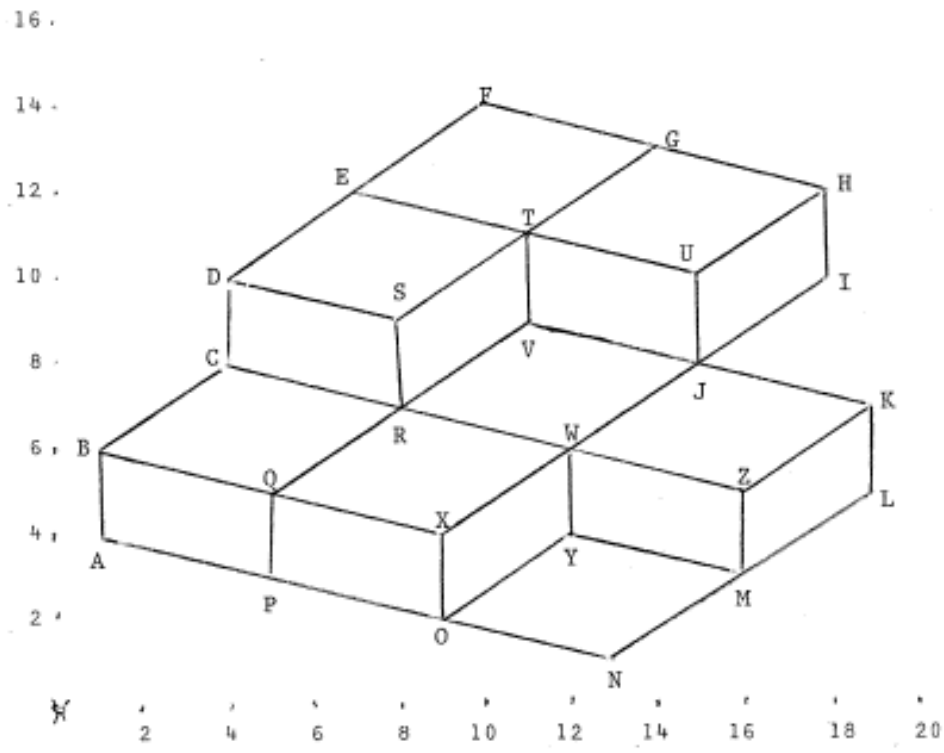


Fig. 10 W H A T?

```
load ((a cube tricky))
 ( CERO UNDECLARED)
 (CERO UNDECLARED)
NIL
              e (cubs tricky)

B
F
J
M
N
O
P
(THERE ARE 2 OR 1 CUBES)

(CORNERS = ( 0.4E0  0.1E7 I (J H) G (H F)) (-0.5E0  0.4E0 E
(F D) C (D B)) (-0.5E0  0.1E7 K (J L) A (L B)))

(TES =)
(YSS = (-0.5E0  0.4E0  0.1E7 ((0 0 0) B (M C A)) ((0 1 1) O
(H L D)) ((1 0 0) P (L H J) N (D F H) M (B D L)) ((1 0 1) J
(K I P)) ((1 1 0) F (E N G))))

(LOOK (0 1 0) B C)
(LOOK (1 0 0) B M)
(LOOK (1 0 1) M L)
(LOOK (1 1 0) M D)
(LOOK (0 0 1) B A)

(CUBE 1 IS (M (B D L) B (M C A))  )
(LOOK (0 0 1) O L)
(LOOK (1 1 1) O H)
(LOOK (0 1 0) O D)

(FALSE CUBE (O (H L D))  )
(LOOK (1 1 0) P H)
(LOOK (0 0 0) P L)
(LOOK (1 0 1) P J)
(LOOK (1 1 1) J I)
(LOOK (0 0 1) J K)

(CUBE 2 IS (J (K I P) P (L H J))  )
(LOOK (1 1 0) N F)
(LOOK (0 1 0) F E)
(LOOK (1 1 1) F G)
(LOOK (0 0 0) N D)
(LOOK (1 0 1) N H)

(CUBE 3 IS (F (E N G) N (D F H))  )

(ANYS = L (A M O P K) H (O N P I G) D (O M C N E))
```

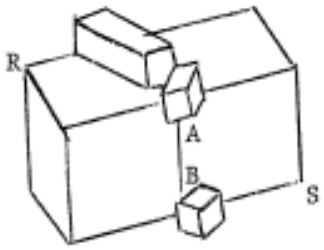
Results for TRICKY (fig. 9). CUBE accepts the 3 exterior cubes and rejects O (H L D). The program was run in a semi-traced mode, and additional information is displayed.

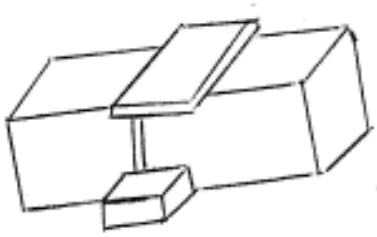
```
1 oad ((a cube what))
  (CERO UNDECLARED)
  (CERO UNDECLARED)
  NIL
                        e (cubs what)

(THERE ARE AT LEAST  2 OR 1 CUBES)

(CUBE 1 IS (O (P Y X) X (Q W O) Q (X R P))  )

(CUBE 2 IS (S (D T R) D (S E C))  )

(CUBE 3 IS (Q (B R P) B (Q C A))  )

(CUBE 4 IS (M (Y L Z) K (J Z L) Z (W K M))  )

(CUBE 5 IS (H (G U I) U (T H J))  )

(CUBE 6 IS (M (Y N Z) Y (M O W) O (N Y X))  )
(FALSE CUBE (V (J R T)))
(FALSE CUBE (C (R B D)))
```

These are the results for WHAT?. 6 cubes are found; M Y O W is accepted, but J V R T is not. This is (fig. 10) certainly a possibility; otherwise, how does one explain with cubes the presence of lines O N and N M ?

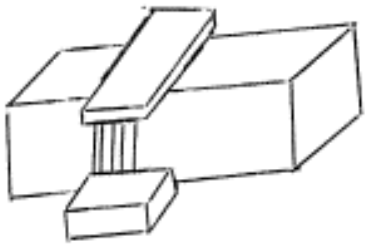
The next two pages show the example TOWER (fig. 11). All the cubes but one are correctly identified: cubes C* T and P I are [con]fused and they appear in the answer as only one, namely C* N* A* I. This is because we do not use information about lines; if this were the case, line A B --see figure below-- would tell us not to think of R S as being just one cube, but two instead.

It is not clear, on the other hand, how many cubes we see in figures (A) and (B).





(A)



(B)

e (cubs tower)

(THERE ARE AT LEAST 3 OR 2 CUBES)

(CUBE 1 IS (A* (X I X*) X* (Z O A*)))

(CUBE 2 IS (X* (Z O U*) U* (H S X*)))

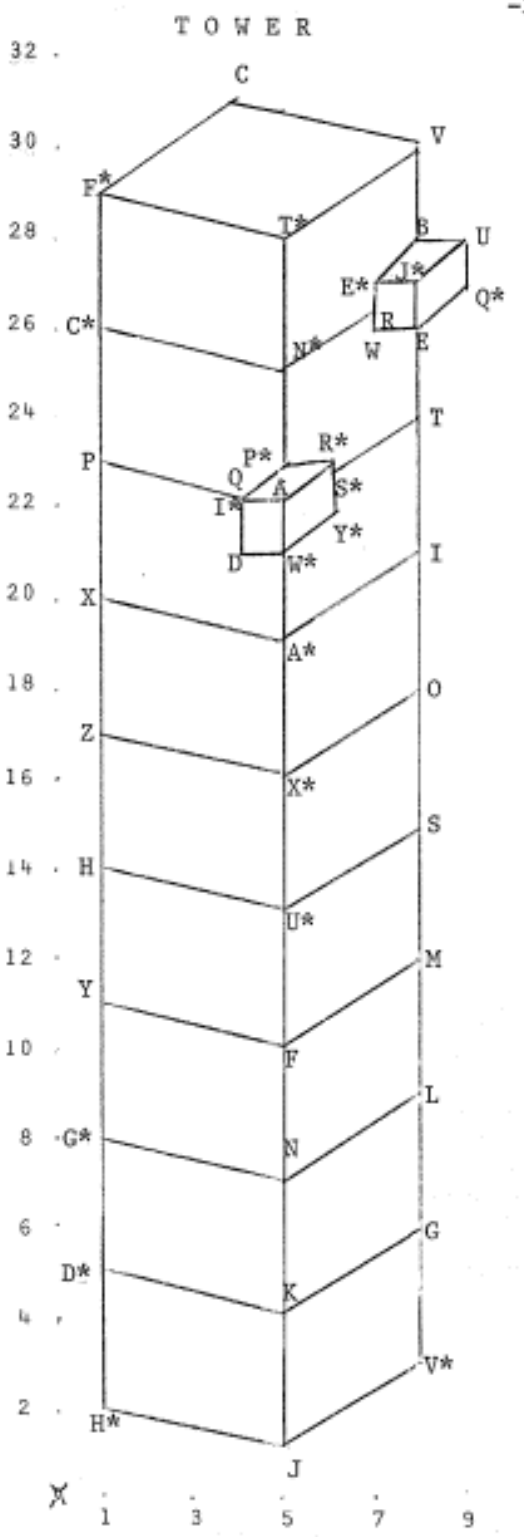
(CUBE 3 IS (V (C T* B) F* (T* C C*) T* (F* V N*) N* (C* R T*))
)

(CUBE 4 IS (N* (C* R P*) A* (X I W*)))

(CUBE 5 IS (F (Y M N) N (G* L F)))

(CUBE 6 IS (N (G* L K) K (D* G N)))

(CUBE 7 IS (K (D* G J) J (H* V* K)))



A	5	22	(I* R* W*)
B	8	28	(V U E*)
C	4	31	(F* V)
D	4	21	(I* W*)
E	8	26	(W J* Q* T)
F	5	10	(Y U* M N)
G	8	6	(K L V*)
H	1	14	(Y Z U*)
I	8	21	(T O A*)
J	5	1	(H* K V*)
K	5	4	(D* N G J)
L	8	9	(N M G)
M	8	12	(F S L)
N	5	7	(G* F L K)
O	8	18	(X* I S)
P	1	23	(C* X Q)
Q	4.2	22.2	(I* P P*)
R	7	26⅓	(E* W N*)
S	8	15	(O M U*)
T	8	24	(E S* I)
U	9	28	(B Q* J*)
V	8	30	(C T* B)
X	1	20	(P A* Z)
Y	1	11	(H F G*)
Z	1	17	(X X* H)
A*	5	19	(X W* I X*)
C*	1	26	(F* N* P)
D*	1	5	(G* K H*)
E*	8	27	(B J* R)
F*	1	20	(C J* T*)
G*	1	8	(Y N D*)
H*	1	2	(D* J)
I*	4	22	(Q A D)
J*	8	27	(E* U E)
N*	5	25	(R P* C* T*)
P*	5	23	(N* Q R*)
Q*	9	27	(U E)
R*	6	23	(P* S* A)
S*	6	22⅔	(R* T Y*)
T*	5	28	(F* V N*)
U*	5	21	(D A Y*)
V*	8	3	(G J)
X*	5	16	(U* O A* Z)
Y*	6	22	(S* W*)
W	7	26	(R E)

Fig. 11 T O W E R

```
(CUBE 8 IS (U* (H S F) F (Y M U*))  )

(FALSE CUBE (W* (D Y* A*)))

(CUBE 9 IS (E (W Q* J*) E* (J* B R) U (B J* Q*) J* (E* U E))
 )
(FALSE CUBE (E (W Q* T)))

(CUBE 10 IS (W* (D Y* A) I* (A Q D) R* (P* A S*) A (I* R* W*))
 )
(FALSE CUBE (P* (R* Q N*)))
(FALSE CUBE (B (U E* V)))
```

Solution to TOWER (fig. 11).

Vertices such as K (D* N G J) having 4 connected points, two of which (J and N) are collineal, get decomposed in two y's: K (D* N G) and K (D* J G).

The next pages contain a listing of the program. The CONVERT function which recognizes cubes is (CUBS E), and occupies less than 2 pages.

```
(CUBS (LAMBDA (E) (CONVERT (QUOTE (
GR    PAV  GRR        GRR  STL  X        (CCC)  CNT  1
ORD   REPT ( ((XXX (X Y) YYY (GR Z) WWW) (=REPT= (XXX (GR Z)YYY (X Y)
                                                          WWW)) )
             ( (PA1) (=PROG= () (=SETQ= Y1 S) F))  )
X*    PAT  (=XEC= EQUAL1 == X)
Y*    PAT  (=XEC= EQUAL1 == Y)
Z*    PAT  (=XEC= EQUAL1 == Z)
Z1    SKEL (=PRNT= =BLNK=)
(TG)  SKEL (=EXEC= TAN X1)
YZW   PAT  (=OR= Y Z W)
NUM   SKEL (=SETQ= X2 (=INCR= X2))
NU     SKEL (=REPT= (Y1 U V) C8 (
                ( ((X XXX) U (Y YYY)) ((=WHEN=(=EXEC= PARALELL X1 X U Y
                     *T* 0 1) (*REPT* ((XXX) U (YYY)) )) )
                ( == () ) ))
CO    BUV  (=AND= =ATO= (=XEC= COLINEAL V W ==))
(COL) PAT  ((*OR* () ( (=OR= CO ==) == COL )))
LOOK REPT (
             ( (U V W (=== (U COL) ===))
                    (=WHEN= ((ORD (*ITER* J CO ((=EXEC= LENGH
                     J V) J) )) Y1      ((X ===) (Y ===))
                (=WHEN= SAME (XXX (U YYY Y Z UUU) ZZZ)
                     (Y Z (*ANUL*(=SETQ= SAME (XXX(U YYY UUU)ZZZ)))) ())
                            ()                                  ))
          ( == ()) )
```

```
COM    REPT ( ((1) 0) (== 1) )
(PA1)  PAT  ((*OR* () ((F S) PA1) ))     F BUV ==    S BUV  ==
(WATCH)REPT ( ( (T Y ===(== T === Y Z ===)===) (Z))
              ( == ()) )
(SEE)  REPT ( ( (T V Y Z ===(T V === Y (=OR= ((=AND= (=XEC= PARALELL
                    == Y X1 Y) U) W) (W U)) ===) ===)
               (=WHEN= (=EXEC= PARALELL Y W X1 Z) *T* ((U W)) () ))
              ( == ()) )
POINTS SKEL (=ITER= J =SAME= ((TG J) J) )
)) (QUOTE (Y Z W X U V (XXX)(YYY)(ZZZ)(WWW)(VVV)(UUU)T )) E (QUOTE( C1(
( == (=PROG= (X1 SAME (ANY)X2(YS) (CRS) (TS) Y1)  (=SETQ= SAME =SAME=)
1(=WHEN= SAME(X Y XXX)((=SETQ= X1 X)(=SETQ= Y1 Y)(=SETQ=SAME(XXX)))
                                                    (=GOTO= 2))
(=REPT= Y1 C2 (
     ( (== ==) (=REPT= ( (ORD (*FRAG* POINTS)) CRS ) C6 (
          ( ((X Y) XXX(X* Y* YYY)WWW)
                              (=SETQ= (CRS) (XXX (X Y X1 Y1 YYY)WWW)) )
          ( ((X Y)===)  (=SETQ= (CRS)  ((X Y X1 Y1) CRS)))   )))
     ( (== == ==) (=REPT= ( (YS) POINTS ) C3 (
          ( (== (XXX (X U) YYY (X* V) ZZZ))
               (=CONT= (XXX YYY ZZZ TS) C5 (
                    ( ((Y W) VVV (X Y WWW) UUU) (=SETQ= (TS)
                         (VVV (X Y X1 (U V W) WWW) UUU)) )
                    ( ((Y W) UUU) (=SETQ= (TS) ((X Y X1 (U V W)) UUU)))
                                                                  )))
          ( (X (Y Z W)) (=NEXT= ((ORD Y Z W)  X)) )
          ( ((X Y Z) (XXX (X* Y* Z* ((=AND=(0 0 0)W) U V ZZZ) VVV)WWW))
              (=SETQ=(YS) (XXX(X Y Z(*REPT*(NU(W U V ZZZ)VVV)C7))WWW)))
          ( ((XXX)==) (=SETQ= (YS) ((XXX((=QUOT=(0 0 0))X1 Y1))YS))) )))
     ( (== == == ==)  (=REPT= POINTS C10 (
          ( (XXX(X Y)YYY(X* Z)ZZZ)  (=CONT= (XXX YYY ZZZ) C11 (
               ( ((== U)(== V))  (=SETQ= SAME
                    (X1 (Y U V) X1 (Z U V) (*FRAG* SAME))) ) )))  )))
     ( == (=SETQ=  (ANY) (X1 Y1 ANY))) ))
(=GOTO= 1)
2 (=PRNT= (THERE ARE AT LEAST (=WHEN= (CRS)(CCC)(=INCR=(=DIVD= CCC 3)))
                         OR (*COND* (YS)(CCC)(CCC)) CUBES))          Z1
(=SETQ= X2 0)
3 Z1
(=COND= (YS) (X XXX) (=GOTO= (=QUOT= 3)  (=SETQ= (YS) (XXX))
     (=PROG= (N1 N2 N3     )  (=SETQ= SAME X)
     4 (=REPT= SAME C4 (
          ( (X Y Z) (=RETN= END) )
          ( (X Y Z(U)VVV) (=GOTO=(=QUOT= 4) (=SETQ= SAME(X Y Z VVV))))
          ( (XXX ((X Y Z) V (T U W) UUU) VVV)
               (=NEXT= (=PROG= ((N4))
                    (=SETQ= SAME (XXX ( ((=SETQ= N1 X)(=SETQ= N2 Y)
                                        (=SETQ= N3 Z)) UUU) VVV))
                    (=SETQ= (N4) (V (T U W)))
                    (=COND= (LOOK (N1 (COM N2) N3)V U SAME)(X(U V W))
                         ( (=SETQ= (N4) (X (U V W) N4))
                           (=COND=(LOOK((COM N1)(COM N2)N3) X U SAME)
                                  (X (U V W))
                                  (=SETQ= (N4) ((*FRAG*(LOOK
```

```
((COM N1)(COM N2)(COM N3))X W SAME)) X (U V W) N4)))
                              (=COND= (LOOK (N1 (COM N2)(COM N3)) X W
SAME) (X(U V W))                        (=SETQ= (N4) ((*CONC*
(LOOK ((COM N1) (COM N2)(COM N3)) X U SAME)
(LOOK (N1 N2 (COM N3)) X V SAME)) X (U V W) N4))      )))
                         (=COND= (LOOK ((COM N1)N2 N3) V T SAME) (X (U V W)
  )                           (=SETQ=(N4)  ((*CONC*
(LOOK ((COM N1)N2(COM N3)) X W SAME)
(LOOK ((COM N1)(COM N2) N3) X V SAME)) X (U V W) N4))  )
                         (=COND= (LOOK (N1 N2 (COM N3)) V W SAME)
(X (U V W))                  (=SETQ= (N4) ( (*CONC*
(LOOK (N1 (COM N2) (COM N3)) X V SAME)
(LOOK ((COM N1) N2 (COM N3)) X U SAME)) X (U V W) N4))  (N4) )  )) )
             ( (X(Y Z W)) (=GOTO= (=QUOT= 4)  (=PRNT=
                  (=REPT= ((*ANUL* ((=SETQ= X1 X) (=SETQ= Y1 (Y Z W))))
                            ((TG Y)Y) ((TG Z)Z) ((TG W)W) )  C12 (
                         ( ((T Y) (U Z) (V W)) (=NEXT= ( (SEE T V Y W CRS)
                            (WATCH T Y TS)              (SEE T U Y Z CRS)
                            (SEE T U Z Y CRS)           (SEE U V Z W CRS)
                            (WATCH U Z TS)              (SEE U V W Z CRS)
                            (WATCH V W TS)              (SEE T V W Y CRS)
     )))                 ( (== == ==) (CUBE NUM IS (X1 (*FRAG* Y1)) ))
                         ( == (FALSE CUBE (X1 Y1)))  ))  )))
             ( =COM= THE NEXT RULE SHOULD BE MODIFIED, BECAUSE WE
                  DO WANT TO MAKE SOME CHECKING )
             ( == (=GOTO=(=QUOT= 4)(=PRNT=(CUBE NUM IS =SAME= Z1)) )) )))
                                                  ))
(=RETN= =BLNK=) ))      )
C7(
( (U XXX (U YYY) ZZZ)  (XXX (U X1 Y1 YYY) ZZZ) )
( (U V VVV) (V (U X1 Y1) VVV)) )     ))     )))
```

LISP FUNCTIONS USED

```
                                                                  DEFINE((
(PARALELL (LAMBDA (A B C D) ((LAMBDA (R S)
(AND (NOT (MINUSP (DOTT R S)))
     (LESSP (ABSVAL (CROSS R S)) CERO)))
(LIST (DIFFERENCE (GET B (QUOTE XCOR)) (GET A (QUOTE XCOR)))
      (DIFFERENCE (GET B (QUOTE YCOR)) (GET A (QUOTE YCOR)) ))
(LIST (DIFFERENCE (GET D (QUOTE XCOR)) (GET C (QUOTE XCOR)))
      (DIFFERENCE (GET D (QUOTE YCOR)) (GET C (QUOTE YCOR)) )) )))

(CROSS (LAMBDA (R S) (DIFFERENCE (TIMES (CAR R) (CADR S))
                                 (TIMES (CAR S) (CADR R)) )))

(DOTT (LAMBDA (R S) (PLUS (TIMES (CAR S) (CAR R))
                          (TIMES (CADR R) (CADR S)) )))
(TAN (LAMBDA (A B ) (COND
((EQUAL (GET B (QUOTE XCOR)) (GET A (QUOTE XCOR)) )0.1E7)
(T (QUOTIENT (DIFFERENCE (GET B (QUOTE YCOR)) (GET A (QUOTE YCOR)))
             (DIFFERENCE (GET B (QUOTE XCOR)) (GET A (QUOTE XCOR)))
  ))) ))

(EQUAL1 (LAMBDA (A B) (LESSP (ABSVAL (DIFFERENCE A B))CERO ) ))
```

```
                                                        )) DEFINE ((
(LENGH (LAMBDA (A B)
(EXPT(PLUS(EXPT(DIFFERENCE(GET A (QUOTE XCOR))(GET B(QUOTE XCOR)))2)
(EXPT(DIFFERENCE (GET A (QUOTE YCOR)) (GET B (QUOTE YCOR)))2))0.5)))

(COLINEAL (LAMBDA (V W C) (EQUAL1 (LENGH V C)
     (PLUS (LENGH V W) (LENGH W C)) )))

(ABSVAL (LAMBDA (A) (COND ((MINUSP A) (MINUS A)) (T A) )))
))
 CSET (CERO 0.003) COMPILE ((PARALELL CROSS DOTT TAN EQUAL1 ABSVAL))
 COMPILE ((COLINEAL LENGH     ))
 EXCISE (T)
STOP)))))))) ) ) ) ))   ) ))   ) ) ) ) )  ) ) ) )  )))) )))
```

REFERENCES

1. Guzmán, A., and McIntosh, H. V. "CONVERT", Communications of the ACM 9, 8 (August 1966), pp. 604-615. Also available as a Project MAC Memorandum MAC-M-316 (AI Memo 99), June 1966.

2. Hodes, L. "Machine Processing of Line Drawings", Report 54G-0028 [U], Lincoln Laboratory, M.I.T. (March 1961).

3. Guzmán, A., and McIntosh, H. V. A Program Feature for CONVERT. Memorandum MAC-M-305 (AI Memo 95); Project MAC, M.I.T. April 1966.

Addendum

POLYBRICK works on a scene or picture, and finds parallelepipeds in it; thus, (CUBS (QUOTE GORDO)) [cf. p. 6] finds three cubes in figure 'GORDO'.

We would like to be able to specify in some suitable notation, a model of the classes of objects we are interested in (models will be 'cube', 'triangular pyramid', 'chair'), and have a program look for all instances of that model in a given scene or figure. Two arguments would have to be supplied now to our program: the model of the object we are interested in, and the scene that we want to analyze. Programs to do this are described in:

4. Guzmán, A. Scene Analysis Using the Concept of Model. Report AFCRL-67-0133; Computer Corporation of America, Cambridge, Mass. January 1967.

5. Guzmán, A. A Primitive Recognizer of Figures in a Scene. Memorandum MAC-M-342 (AI Memo 119); Project MAC, M. I. T. January 1967.

An important restriction here is that partially occluded bodies get incorrectly identified.

A Master's Thesis discusses many ways to identify objects of known forms:

6. Guzmán, A. Some Aspects of Pattern Recognition by Computer. M. S. Thesis. Electrical Engineering Department, Massachusetts Institute of Technology. February 1967. Also available as a Project MAC Technical Report MAC-TR-37.

It will be advantageous that we could find the bodies that form a scene, without knowing their exact description (that is, without having a model of them). SEE is a program that works on a scene presumably composed of three-dimensional rectilinear objects [that is, formed by plane faces], and analyzes the scene into a composition of 3-dim objects. Partially occluded objects are properly handled. This program is discussed in:

7. Guzmán, A. Decomposition of a Visual Scene into Bodies. Memorandum MAC-M-357 (AI Memo 139); Project MAC, M.I. T. September, 1967.

8. Guzmán, A. Decomposition of a Visual Scene into Three-Dimensional Bodies. AFIPS Proceedings of the 1968 Fall Joint Computer Conference
Thompson Book Co. Washington, D.C.

Handling of stereo information (two views, left and right, of the same scene), improvements to deal with noisy (imperfect) input, figure-background discrimination, etc., will be found in a doctoral thesis:

9. Guzmán, A. Computer Recognition of Three-Dimensional Objects in a Visual Scene. Ph. D. Thesis, Electrical Eng. Dept., M. I. T. (end of 1968 or beginning 1969) Will probably appear, too, as a Project MAC Technical Report