Comput. & Graphics Vol. 13, No. 4, pp. 505–521, 1989
Printed in Great Britain.

0097-8493/89 $3.00 + .00

Technical Section

CONSTRUCTION OF 3D SOLID OBJECTS FROM ORTHOGRAPHIC VIEWS

UDAY G. GUJAR and I. V. NAGENDRA
School of Computer Science, University of New Brunswick, Fredericton, N.B., Canada E3B 5A3

Abstract—An algorithm to generate three-dimensional solid objects, made up of planar surfaces, from the given three conventional engineering orthographic views is presented in this paper. Consisting of six major steps, the algorithm has been programmed in C on IRIS 1400 graphics workstation. The algorithm generates all possible solutions. The infinite space has been divided into finite subspaces by making use of the surface normals and the direction of travel of the edges that connect the faces. Classification of the probable 3D subobjects into the certain and uncertain ones has proved to be very useful in reducing the time taken by the algorithm. Several illustrative examples, simple as well as complex giving single and multiple solutions, are included.

1. INTRODUCTION

Wire frame and solid geometric models have attracted a lot of attention from researchers. Ambiguous in nature, wire frame models do not carry surface information and as such they are not useful in calculating volumetric properties[1]. Solid models do not suffer from this drawback. Solid geometric models can be constructed in a number of ways such as constructive solid geometry[2], interpolation between the specified space contours[3, 4], surfaces of revolution[5–7], sweeping of the specified contours[3, 8], and generation of solids from the orthographic views[9].

It is a common engineering practice to generate three two-dimensional orthographic views (namely, plan, front view and side view) to convey the form of the three-dimensional objects. To date, researchers have come up with an array of algorithms which deal with the generation of solids from orthographic views. After Idesawa's paper appeared in 1973[10], several algorithms have been developed[11–22], each having its own advantages and limitations. Some of these algorithms deal with planar surfaces[10, 11, 13, 14, 21, 22], while others deal with both planar as well as curved surfaces with some restrictions. Nagendra and Gujar give a comprehensive survey of these algorithms outlining their capabilities and limitations and a categorization tree structure[9].

Although these algorithms are based on a variety of approaches, one approach known as the bottom-up approach has been widely used. The major steps involved with this approach are the following:

1. Transformation of 2D vertices to 3D vertices;
2. Generation of 3D line segments from 3D vertices;
3. Construction of faces from 3D line segments;
4. Formation of 3D objects from faces.

The algorithm presented in this paper combines the different steps proposed by Markosky *et al.*[14, 23] and Sakurai *et al.*[17] and introduces some new concepts. Surface normals and the direction of travel of the edges which connect faces have been used effectively. One of the prime concepts introduced in this algorithm is that it categorizes the different probable 3D subobjects into the certain and uncertain ones by applying different criteria. This has a significant effect on reducing the amount of time required to assemble the probable 3D subobjects.

2. ALGORITHM

The overview of the various phases of the system is given in Fig. 1. Phase 1 deals with the format of the input data which is in the form of two-dimensional coordinate values along with the tags indicating the type of connection between the current 2D vertex and the previous one. Phase 2 is the main subject of this paper; it is broken down into six major steps (see Fig. 2). Phase 3 deals with the manner in which the 3D object is represented in the computer. The 3D objects are expressed in terms of faces which in turn are related to edges which are expressed as straight lines joining 3D vertices. Thus, the generated 3D objects are in the form of a linked list data structure. Phase 4 consists of a program to display the given input views along with the generated object(s) in the form of a line drawing; an interface with a ray-tracing program[3] generates realistic solid objects. The details of all the phases appear in [21].

We now deal with the main phase, namely, Phase 2 which consists of "Algorithm working on the orthographic views." The major steps along with the substeps have been outlined in Fig. 2. All these steps are described in the following subsections. While constructing a 3D object, one has to consider each quantity as a probable quantity since that quantity may not be present in the final analysis. For convenience, probable 3D vertices, probable 3D edges, probable faces, probable 3D subobjects are written as p-vertices, p-edges, p-faces, p-subobjects, respectively.

2.1. *Step* 1: *2D vertices and 2D lines*

In this step we process the input data and remove certain redundancies. All the points and lines that are input by the user are 2D vertices and 2D lines. First, all the duplicate specifications of these vertices, if any, are removed. Next, since the user is not required to input the intersection points of two straight lines, it is

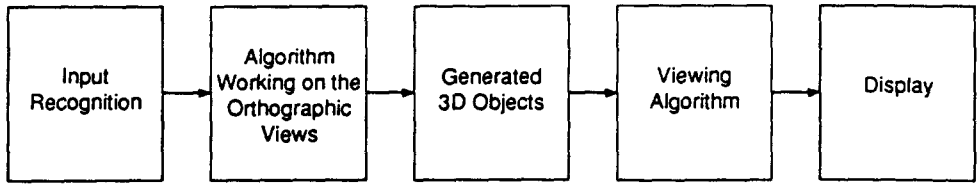


Fig. 1. Overview of the various phases.

determined if an intersection point exists and if it does, it is added to the list of vertices. Finally, the collinear lines of the same type, namely, solid or dashed, are combined to form one line.

The intersection of two lines is determined using the following algorithm [24]. Let the two lines under consideration be KL and MN. The parametric equations for these lines can be written as:

$$x = x_K + (x_L - x_K)s$$
$$y = y_K + (y_L - y_K)s, \quad 0 \le s \le 1 \qquad (1)$$

for line KL and

$$x = x_M + (x_N - x_M)t$$
$$y = y_M + (y_N - y_M)t, \quad 0 \le t \le 1 \qquad (2)$$

for line MN, where s and t are the parameters. The point of intersection, J, satisfies both Eqns. (1) and (2). One can obtain the values of the parameters s and t at J as:

$$s_J = \frac{(x_N - x_M)(y_M - y_K) - (y_N - y_M)(x_M - x_K)}{(x_N - x_M)(y_L - y_K) - (y_N - y_M)(x_L - x_K)}, \qquad (3)$$

$$t_J = \frac{(x_L - x_K)(y_M - y_K) - (y_L - y_K)(x_M - x_K)}{(x_N - x_M)(y_L - y_K) - (y_N - y_M)(x_L - x_K)}. \qquad (4)$$

If both s_J and t_J are in the range of 0 to 1, then the intersection occurs within the two line segments and is given by

$$x_J = x_K + (x_L - x_K)s_J = x_M + (x_N - x_M)t_J,$$
$$y_J = y_K + (y_L - y_K)s_J = y_M + (y_N - y_M)t_J. \qquad (5)$$

This point is then added as a 2D vertex. If $s_J < 0$ or s_J

1. 2D Vertices and 2D Lines
2. Probable 3D Vertices
3. Probable 3D Edges (repeat until there is no false p-vertex)
   - Generation of p-edges
   - Checking of p-vertices; (test for false p-vertext)
4. Probable Faces (repeat until there is no false p-edge)
   - Determination of planar surfaces
   - Generation of p-edge closed loops; (test for false p-edge)
   - p-Edge loops relationship
     - contained in-loops
     - contained out-loops
   - Formation of p-faces
   - Testing of 2D dashed lines; (test for false p-edge)
   - Checking of p-edges; (testing for false p-edge)
5. Probable 3D Subobjects (repeat until there is no false p-face)
   - connection of p-faces; (test for false p-face)
   - classification of p-subobjects
     - certainty test 1
     - if required, perform certainty test 2
6. Assembling and Testing

Fig. 2. Major steps of the algorithm.

> 1 or $t_J < 0$ or $t_J > 1$, then the line segments intersect outside their span and the intersection is not a valid 2D vertex.

2.2. *Step* 2: *Probable* 3*D vertices*

A list of probable 3D vertices is constructed in this step. If any two 2D vertices, belonging to different views, have the same coordinate value for the shared coordinate axis, then the third view is searched for the 2D vertex which has the same values as the remaining two coordinates from the original two 2D vertices under consideration. If the search is successful then a 3D p-vertex containing the common x, y, z coordinates from three 2D vertices is found. This procedure is carried out for all the 2D vertices.

2.3. *Step* 3: *Probable* 3*D edges*

Recursive in nature, this step involves both the generation of p-edges and checking the validity of p-vertices. If any p-vertex is found to be invalid, that p-vertex is deleted and the procedure goes back to the beginning of this step.

2.3.1. *Generation of p-edges.* A straight line which connects any two p-vertices is a p-edge provided the projections of this 3D edge can be found in all the three input views. Since this algorithm deals with planar objects, these projections can appear as 2D lines or as a 2D vertex. Whenever this test is satisfied the p-edge formed under consideration is compared with the previously generated p-edges. If the most recently generated p-edge contains any one of the previously generated p-edges, then the most recently generated p-edge is not included in the p-edges table. On the other hand, if any of the previously generated p-edges contain the most recently generated p-edge then all those previously generated p-edges are deleted. The criterion behind this test is not to store overlapping p-edges.

2.3.2. *Checking of p-vertices.* The p-edges are stored in terms of their end points. From this, a table that gives the p-edge numbers to which each p-vertex belongs is created. Since the algorithm deals with solid objects, each p-vertex should belong to at least three p-edges. If a p-vertex belongs to less than three p-edges, then it is assumed to be a false p-vertex and is deleted. Whenever a p-vertex is deleted, the validity of the p-edges is no longer guaranteed; hence the process returns to the beginning of this step (*i.e.*, step 3). If none of the p-vertices is deleted then the process advances to the next step.

2.4. *Step* 4: *Probable faces*

In this step, a list of probable planar faces is constructed. This step is composed of the following substeps:

1. Determination of planar surfaces;
2. Generation of p-edge closed loops;
3. P-edge loop relationships;
4. Formation of p-faces;
5. Testing of 2D dashed lines;
6. Checking of p-edges.

2.4.1. *Determination of planar surfaces.* The list of p-edges is searched to find out a pair of p-edges with a common p-vertex. A planar surface passing through these p-edges can be represented mathematically as:

$$ax + by + cz + d = 0.$$

Given KJ and JL as the p-edges with (x_J, y_J, z_J), (x_K, y_K, z_K) and (x_L, y_L, z_L) as the coordinates of the p-vertices J, K and L, respectively, the equation of the plane through these points is given as[24]:

$$\begin{bmatrix} x - x_J & y - y_J & z - z_J \\ x_K - x_J & y_K - y_J & z_K - z_J \\ x_L - x_J & y_L - y_J & z_L - z_J \end{bmatrix} = 0.$$

Then the coefficients a, b, c, d can be determined as

$$a = (y_K - y_J)(z_J - z_J) - (z_K - z_J)(y_L - y_J),$$

$$b = (z_K - z_J)(x_L - x_J) - (x_K - x_J)(z_L - z_J),$$

$$c = (x_K - x_J)(y_L - y_J) - (y_K - y_J)(x_L - x_J),$$

$$d = -(ax_J + by_J + cz_J).$$

A list of all such planar surfaces, in terms of their coefficients, passing through all the pairs of connecting p-edges is formed. It is possible that some duplicate surfaces may be formed; these duplicates are deleted. This is done by calculating the perpendicular distances between the two p-vertices of a p-edge and a surface. If these perpendicular distances are in the vicinity of zero (typically 0.0001), then the p-edge lies on the surface under consideration. After collecting all the p-edges which lie on each surface, duplicate surfaces are deleted. (Note that simply searching for the duplicate coefficients a, b, c, d is not sufficient since these are not normalized. It is possible to normalize these coef-

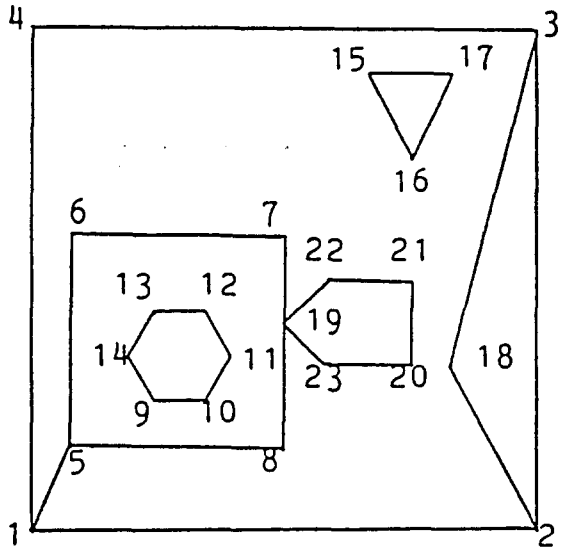


Fig. 3. Coplanar p-edges.

ficients to look for duplicates instead of using the method described.)

2.4.2. *Generation of p-edge closed loops.* The p-edge closed loops are generated using surface normals (see Appendix I).

The p-edge closed loops are determined by examining all the coplanar p-edges by following the turn-to-the-left-most rule. For this purpose, any p-edge is chosen as the starting line and a traversal is carried out in the same, arbitrarily chosen, direction from one p-edge to another. If more than one p-edge meets the p-edge being traversed, the p-edge that makes the smallest angle, in the clockwise direction, with the current p-edge is chosen as the next p-edge. The smallest angle is chosen to determine the closed loops. It is possible to use the largest angle, in which case the direction of the closed loops will be reversed.

A loop is complete when the starting p-edge is to be traversed in the same direction again. If any p-edge is traversed in the opposite direction while finding a loop, that p-edge is assumed to be a false p-edge and is deleted from the list. Whenever this happens, the process goes back to the beginning of step 4 since the validity of the already determined closed loops can no longer be guaranteed. All the closed p-edge loops, lying on a planar surface, are said to have been found when all the p-edges on that surface have been traversed in both directions.

It is important to find the region that is bounded by each p-edge loop, *i.e.*, whether each p-edge loop bounds its inside or outside region. The total turning angle is used for this purpose [17]. A turning angle is the angle by which a p-edge has to be rotated at the common p-vertex while going from the present p-edge to the connecting p-edge. Clockwise rotations are assumed negative. Thus the total turning angle for a p-edge loop is

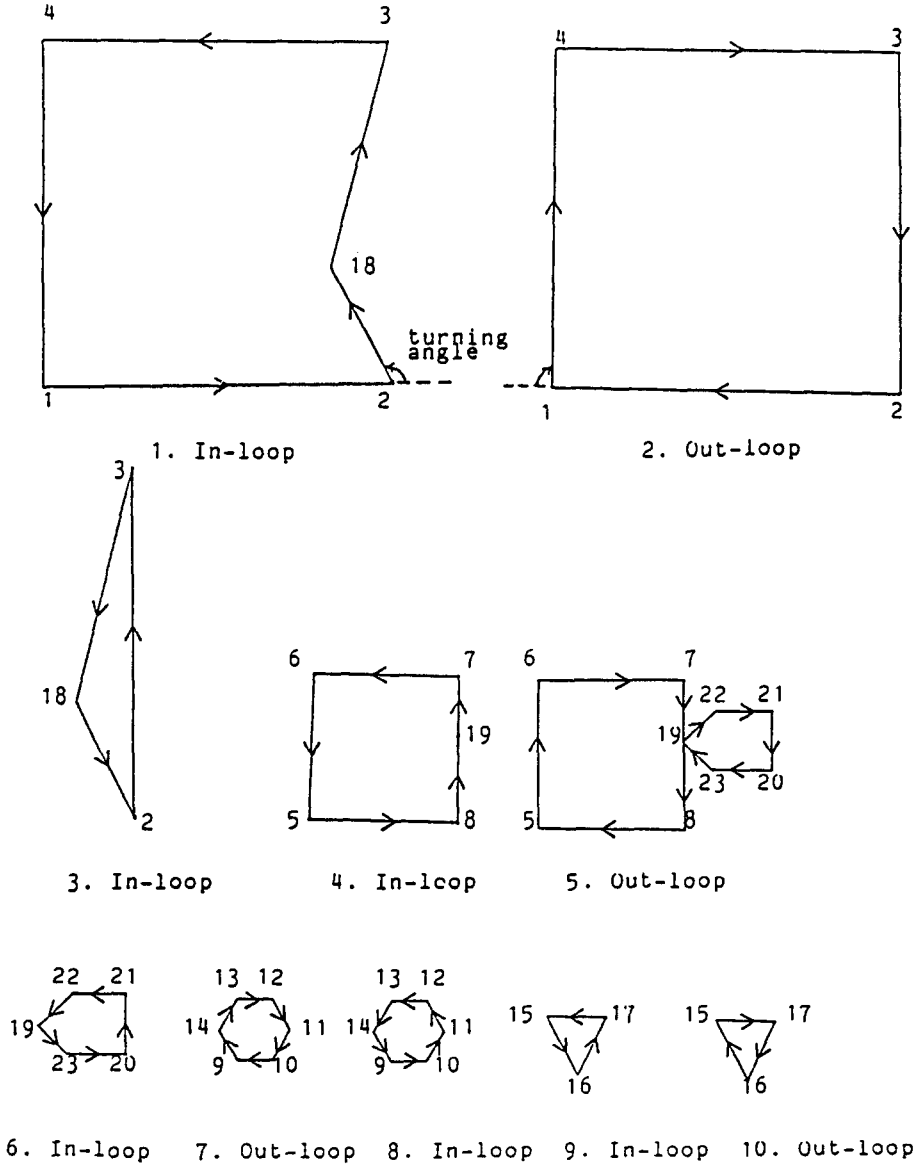


Fig. 4. p-Edge loops.

either +360°, in which case that p-edge loop bounds the inside region and is called an in-loop, or −360°, in which case the p-edge loop bounds the outside region and is called an out-loop.

As an example, an illustration of the coplanar p-edges is given in Fig. 3 where the p-vertices have been given numbers. If one follows the p-edge 1-2, in that direction, one goes through the process of collecting the series 1-2-18-3-4-1-5-6-7-19-22-21-20-23-19-8-5 at which point the next p-edge is 5-1 which has already been selected in the opposite direction (as 1-5). Hence, this is detected as a false p-edge and is deleted. All the p-edge loops generated for Fig. 3 are given in Fig. 4 where each one is also identified as an in-loop or an out-loop.

2.4.3. *p-edge loops relationship.* Now we determine the relationships between different p-edge loops in the sense if one loop is contained in the other. Two types of tests are performed to determine the contained in-loops and contained out-loops.

2.4.3.1. *Contained in-loops.* An in-loop is said to be contained in an out-loop if they share at least one p-edge. Other in-loops which share p-edges with the already tested in-loop are also contained in the out-loop under consideration even if those in-loops do not share any p-edge with the out-loop.

With respect to Fig. 4, loop 2 contains loops 1 and 3, loop 5 contains loops 4 and 6, loop 7 contains loop 8 while loop 10 contains loop 9. Note that in all the cases, it is an out-loop which contains in-loop(s).

2.4.3.2. *Contained out-loops.* In this test we determine if an out-loop is contained in an in-loop. Obviously, if it is already determined that an in-loop is contained in an out-loop, this test need not be applied for that pair.

The inside test which determines if a point is inside or outside a polygon is used to decide if an out-loop is contained in an in-loop. For this purpose, the number of intersections between edges of the polygon and a semi-infinite line starting from the test point are counted. If the count is odd, the point is inside the polygon, otherwise the point is outside the polygon[25]. In our case, any p-vertex belonging to the out-loop under consideration is used as a test point agiants the in-loop under consideration as the polygon. A number of singularities occur during this test, for example, the test point may be on an edge of the polygon, or the semi-infinite line may pass through one or more of the vertices. If the test point lies on any of the p-edges of the in-loop under consideration, then the test point is classified as being inside.

Fig. 5 shows the relationships between the p-edge loops of Fig. 4. The in-loop 1 contains three out-loops, namely, 5, 7 and 10. A face can be made up of any number of p-edge loops such that one of them describes the external periphery and others being internal to the external one but not internals to each other. In this connection, loop 1 defines the external periphery and loops 5, 7 and 10 must be internal to loop 1 but not to each other. However, in this case, loop 7 is also inside loop 5. This violates the definition of a face. Whenever an in-loop is found to contain more than one out-loop, the inside test is applied to the contained out-loops to determine the containment of the out-loops in one another. In this case, out-loop 5 is found to contain out-loop 7; hence the link from loop 1 to loop 7 is dropped.

2.4.4. *Formation of p-faces.* p-Faces are now constructed from the p-edge loop relationship table. Thus, the p-faces that are generated from Fig. 5 (keeping in mind that the link between loop 1 and loop 7 does not exist anymore) are shown in Fig. 6.

2.4.5. *Testing of 2D dashed lines.* The dashed lines in the input views are lines obscured by some p-face(s). At this stage a p-edge which represents a dashed 2D line is identified. Then this p-edge is tested against the different p-faces to determine if at least one p-face hides the p-edge when it is projected onto that particular view plane. If this condition is not met then that p-edge is deleted and the process goes back to the beginning of step 4.

2.4.6. *Checking of p-edges.* Since we are dealing with solid objects, a p-edge should belong to at least two noncoplanar p-faces. To determine this condition, the

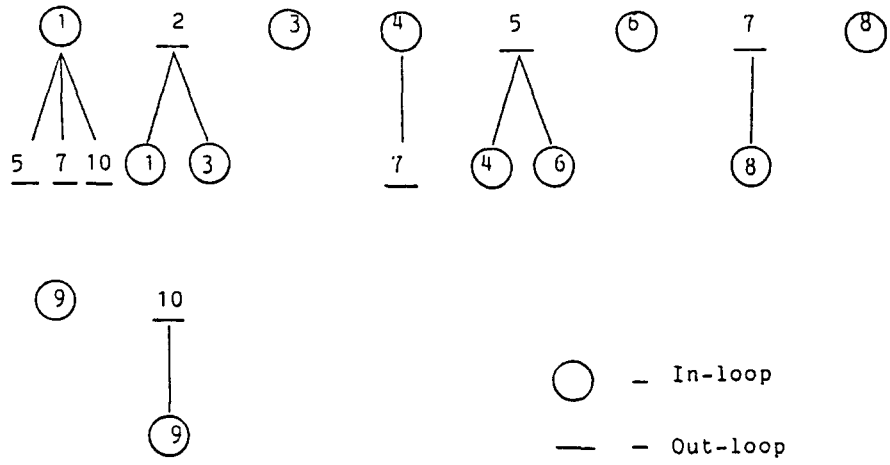


Fig. 5. Relationships of p-edge loops.

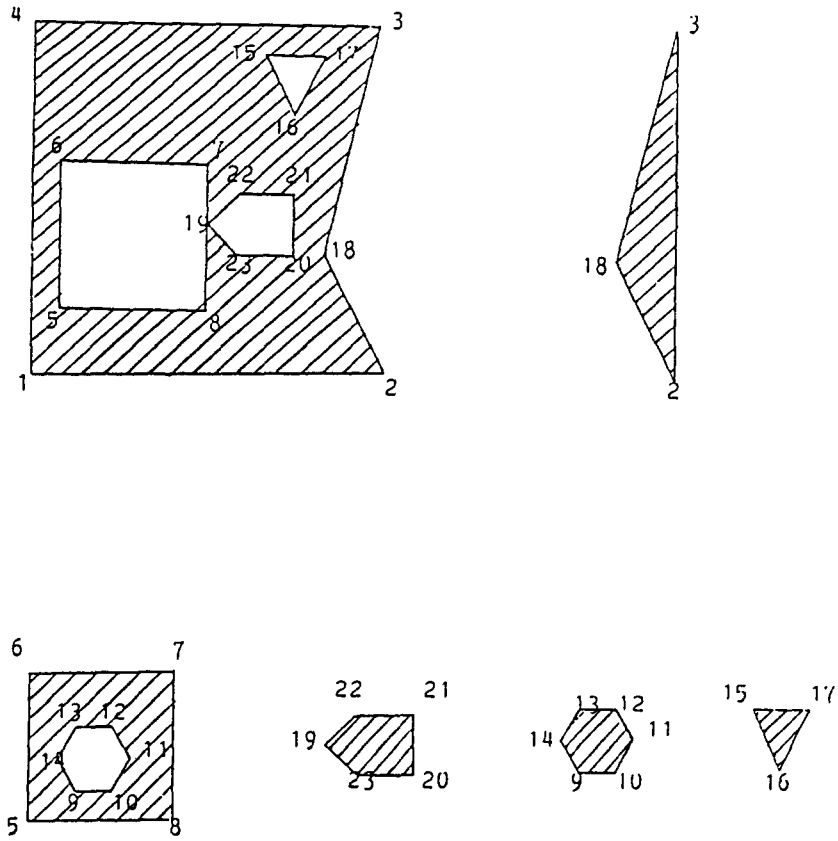


Fig. 6. p-Faces.

number of noncoplanar p-faces to which each p-edge belongs is computed. If this count is less than two, then that p-edge is deleted and the process goes back to the beginning of step 4. If none of the p-edges is deleted, then the algorithm advances to the next step.

2.5. *Step* 5: *Probable* 3*D subobjects*

In this step the p-faces are connected to form p-subobjects which are then classified as certan and uncertain p-subobjects.

2.5.1. *Connection of p-faces.* This phase is analogous to the generation of p-edge loops. Each p-face has two sides. The p-edge loops are generated on that side of the surface where the surface normal is pointing away from the origin; we will denote this side as the negative p-face and the other side as the positive p-face. Two p-faces are connected to each other at the common p-edge. When more than two p-faces share a p-edge, the p-face which makes the minimum face connection angle with the reference p-face is selected. This rule is similar to the smallest angle rule described in Section 2.4.2. To connect p-faces, the cross-product and dot-product between the surface normals are used.

Any p-face is chosen as the starting p-face. With this p-face as the reference p-face, the connecting p-faces are found by making use of Table 1. Next, with any

Table 1. Connection of p-faces.

Reference p-face	Direction of vector of cross product between surface normals with respect to the reference p-edge	Direction of traversal of the shared p-edge	Face connecting angle (with θ as minimum angle between surface normals)	Side to be chosen
o	opposite	same	$360 - \theta$	s
o	opposite	opposite	$180 - \theta$	o
o	same	same	θ	s
o	same	opposite	$180 + \theta$	o
s	opposite	same	θ	o
s	opposite	opposite	$180 + \theta$	s
s	same	same	$360 - \theta$	o
s	same	opposite	$180 - \theta$	s

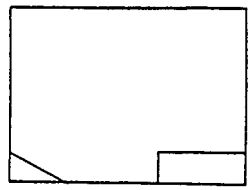
Note: o means the face that bounds the opposite side of the surface normal. s means the face that bounds the same side of the surface normal.

one of the just found p-faces as the reference p-face, its connecting p-faces are found. This procedure continues until all the connecting p-faces are found. At this stage, a p-subobject is said to have been found. In order to find the next p-subobject, any p-face that has not been used so far is chosen as the reference p-face and the procedure repeated. When both sides of all the p-faces have been used, the search for all the p-subobjects is finished. While forming a p-subobject, if both sides of a p-face are required, then that p-face is assumed to be a false one and is deleted; the process in that case goes back to the beginning of step 5.

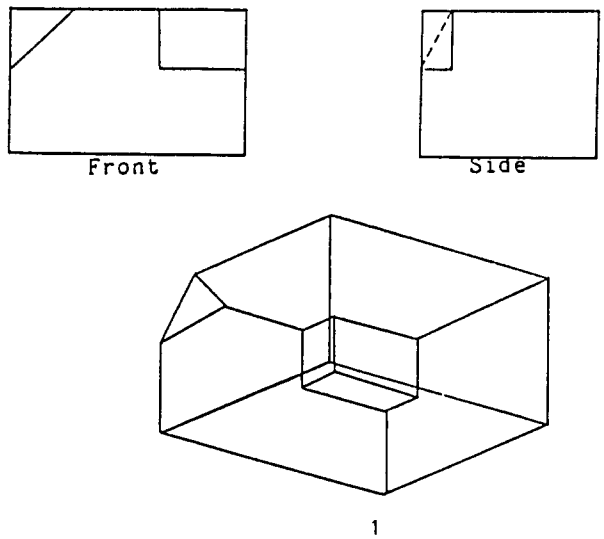
One of the p-subobjects generated is the infinite subspace representing antimatter, *i.e.*, a hole with matter all around it. This is represented by an average face connecting angle of greater than 180°. Such p-subject is discarded. An example of p-subobjects is shown in Fig. 7.

2.5.2. *Classification of p-subobjects.* In order to reduce the time taken by the algorithm by an order of magnitude, the p-subobjects are classified into certain and uncertain p-subobjects. Those classified as "certain p-subobjects" must be contained in the final object. The importance of this classification can be seen from the following analysis. If there are n p-subobjects, then the possible combinations, N, is given by

$$N = \binom{n}{1} + \binom{n}{2} + \binom{n}{3} + \cdots + \binom{n}{n}$$
$$= 2^n - 1.$$



(a) Input Views



(b) p-Subobjects

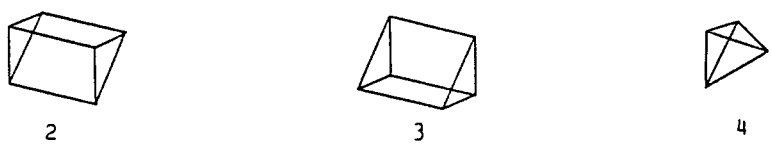


Fig. 7. Input views and generated p-subobjects.

Thus, if $n = 10$, then $N = 1023$. Even if we classify only one p-subobject as a certain p-subobject, the number of combinations reduces to $N = 511$ and if two p-subobjects can be classified as certain p-subobjects, then N reduces to 255.

2.5.2.1. *Certainty test* 1. A table containing the p-subobject numbers to which each input 2D edge belongs is created by taking the three projections of each p-subobject and then comparing the projected lines with the input 2D lines. Those p-edges of a p-subobject which, sometimes, become partly hidden and partly visible when projected onto a plane, are mapped to the appropriate 2D lines of the input views. If a 2D edge is related to only one p-subobject, then that p-subobject is essential in the final object generation step; hence that p-subobject is classified as a certain p-subobject.

2.5.2.2. *Requirement for certainty test* 2. All the certain p-subobjects are first assembled together (see Section 2.6) and tested against the input views; if this produces a perfect match, then we have found the solution or one of the solutions and there is no need to go through the certainty test 2.

2.5.2.3. *Certainty test* 2. This test further examines the remaining uncertain p-subobjects. Here we examine the dashed lines from the input views.

The p-edges that correspond to dashed lines of the input views are tested to find out the p-subobject(s) that obscure them. If there is more than one p-subobject, then that p-subobject which is closest to the projection plane (*i.e.*, front or top or side) is classified as a certain p-subobject.

We demonstrate this further with the help of an example. As shown in Fig. 8, the three identical input views give rise to 27 identical p-subobjects. The first test determines that none of the p-subobjects is a certain p-subobject because there is no 2D edge in a view that uniquely corresponds to any p-subobject. Therefore, the possible combinations are $2^{27} - 1 = 134,217,727$.

Now referring to the dashed lines, it is necessary to

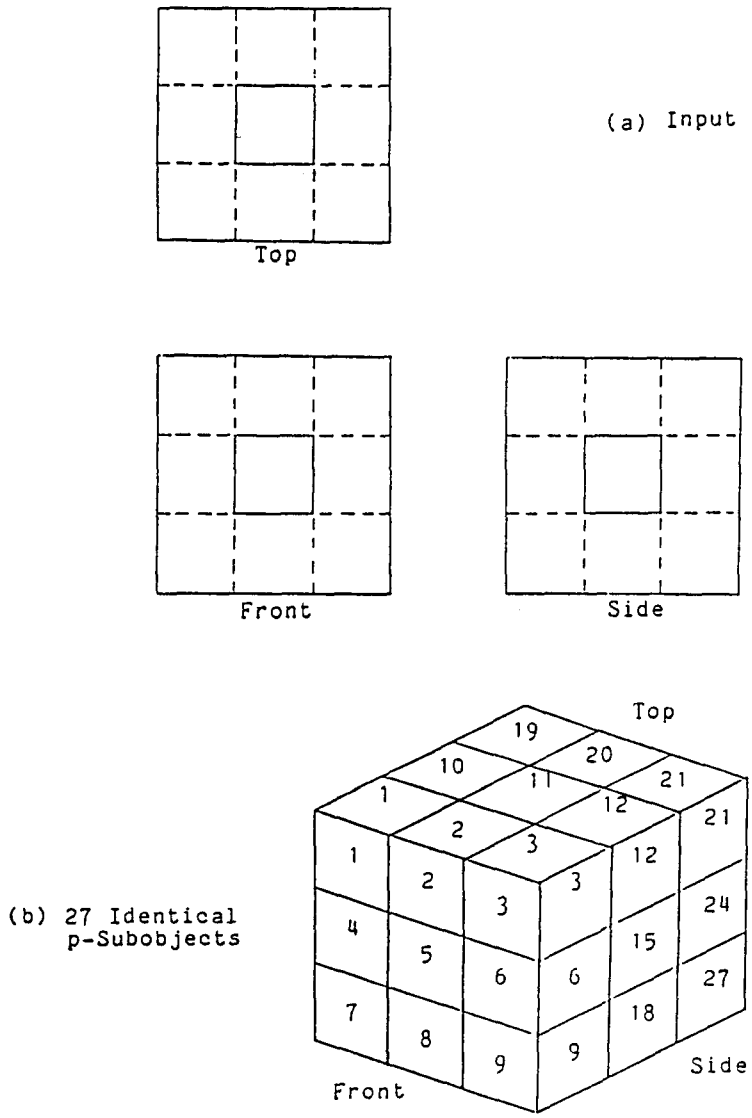


Fig. 8. Input views and 27 identical cubes (p-subobjects).

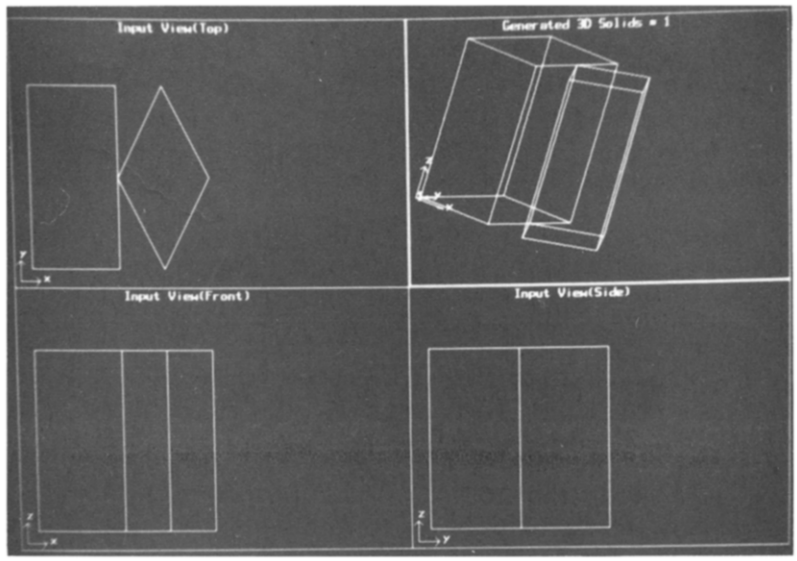


Fig. 9. Example 1.

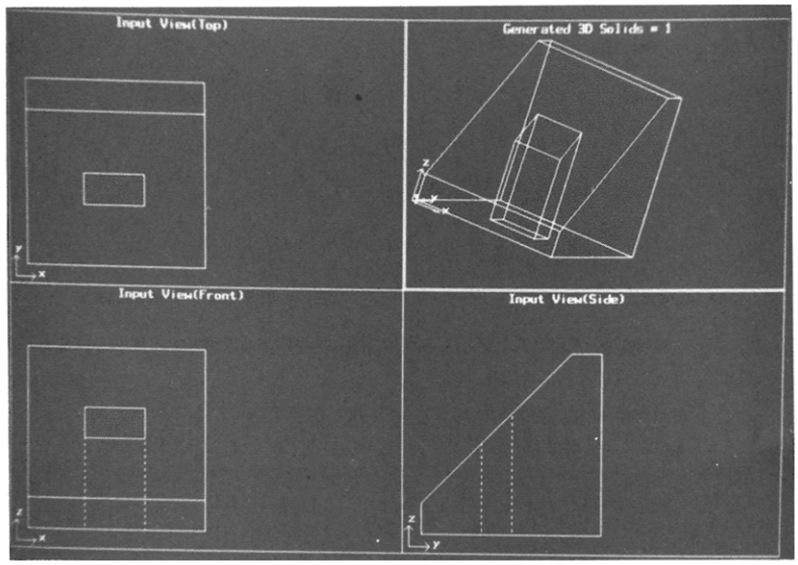


Fig. 10. Example 2.

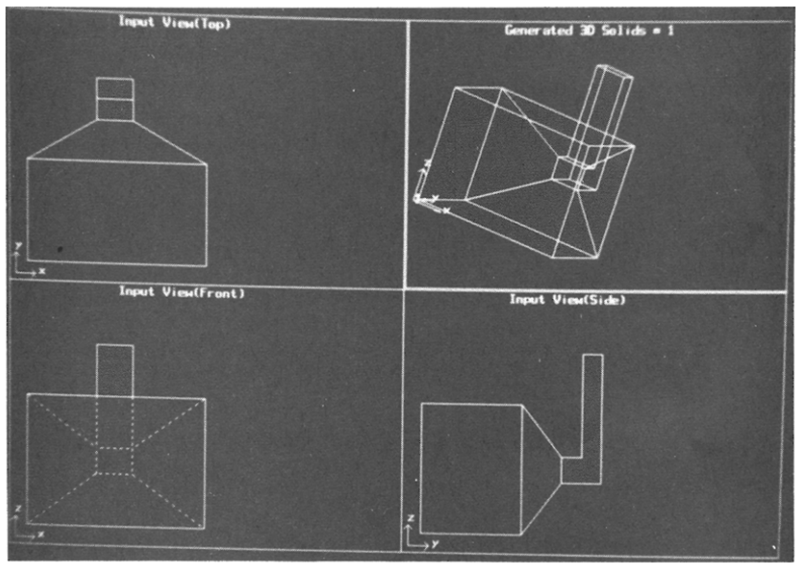


Fig. 11. Example 3.

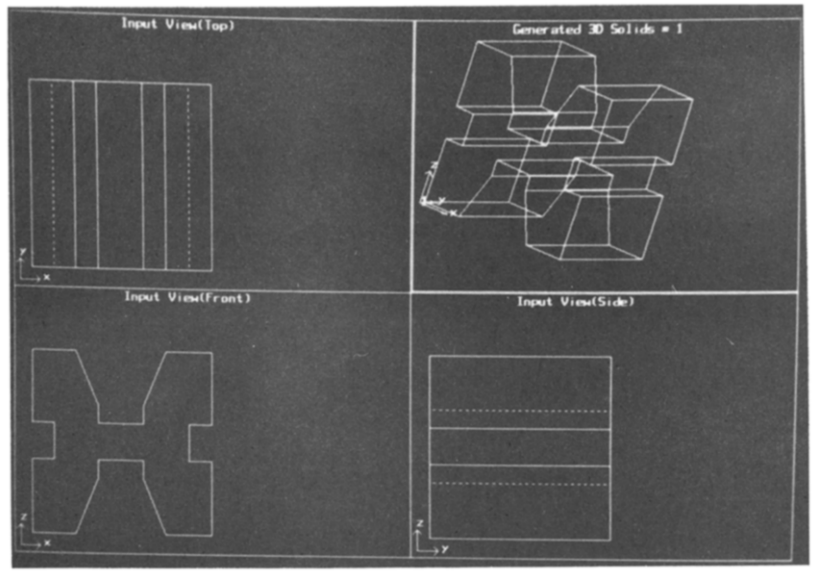


Fig. 12. Example 4.

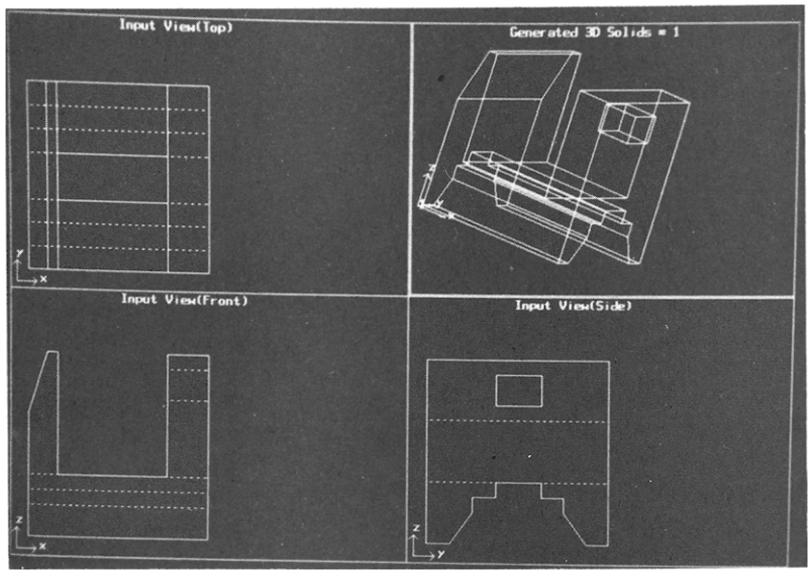


Fig. 13. Example 5.

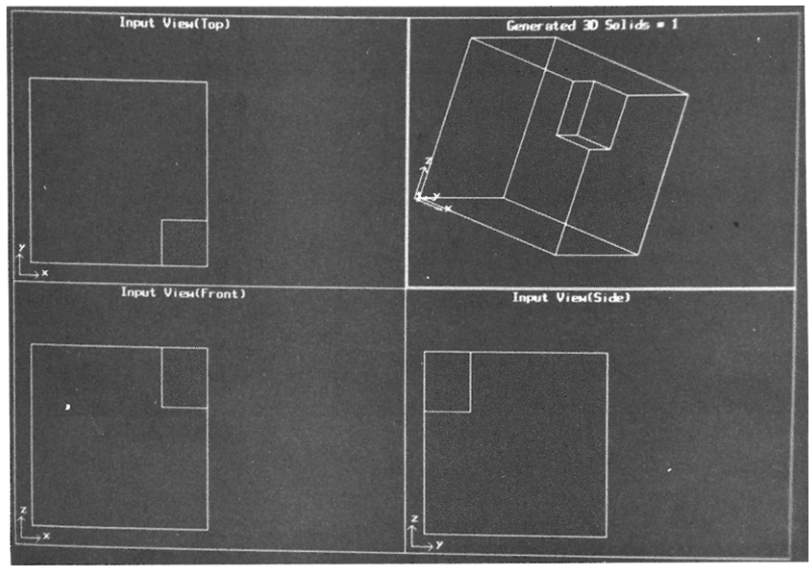


Fig. 14. Example 6.

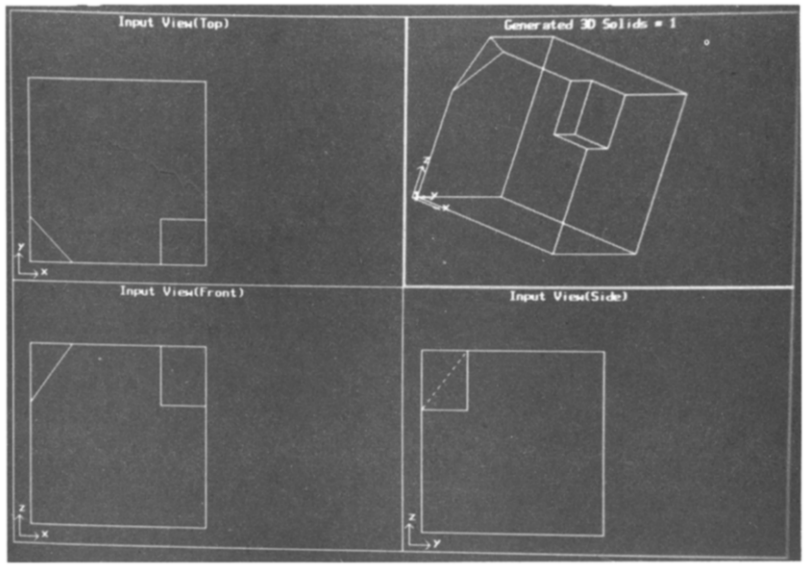


Fig. 15. Example 7.

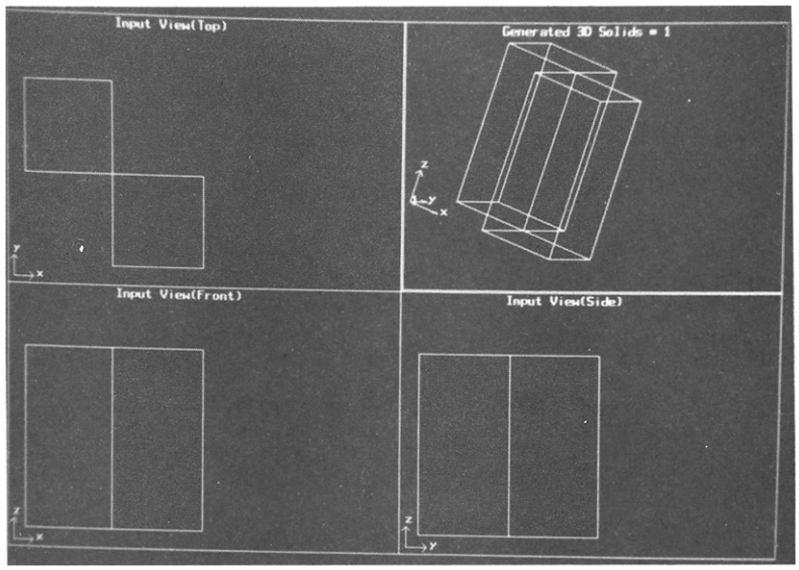


Fig. 16. Example 8.

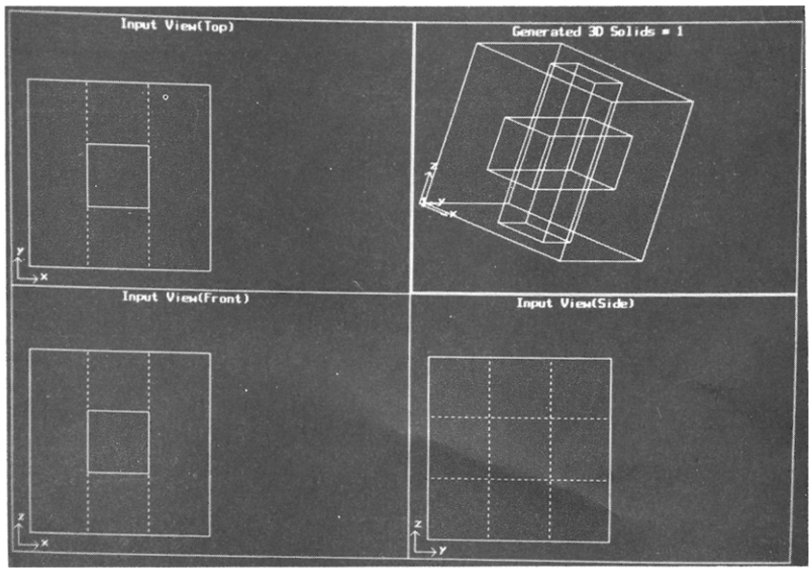


Fig. 17. Example 9.

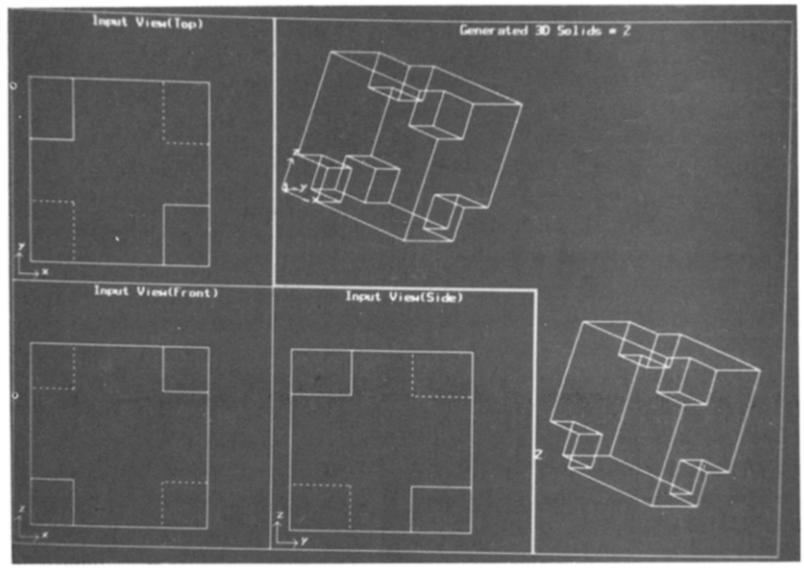


Fig. 18. Example 10.

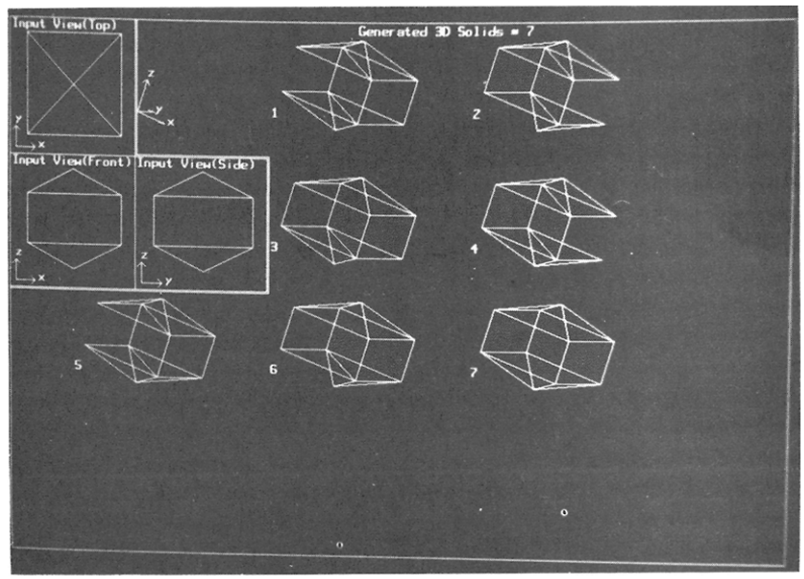


Fig. 19. Example 11.

have the cubes 1, 2, 3, 4, 6, 7, 8 and 9 to obtain the required dashed lines in the front view. Likewise, cubes 1, 2, 3, 10, 12, 19, 20 and 21 are always necessary so that the dashed lines in the top view make sense. Similarly, looking at the side view, cubes 3, 6, 9, 12, 18, 21, 24 and 27 are always required. Thus, the second test has classified 16 p-subobjects as certain p-subobjects. This reduces the possible combination to $2^{11} - 1 = 2047$ which represents a huge order of difference in magnitude.

2.6. *Step 6: Assembling and testing*

First, all the certain p-subobjects are assembled to form a p-object and projections of this object are tested against the input views. If the assembled final p-object gives the same input views, then this p-object is one of the final objects or the only final object. Then each of the different possible combinations of the uncertain p-subobjects is added to the assembly of the certain p-subobjects. The projections of this p-object are then tested against the input views to determine if it represents a solution. This procedure continues until all possible solutions are found by considering all possible combinations.

While assembling the p-subobjects, some p-faces and p-edges may disappear. If a positive p-face of a p-subobject is assembled with a negative p-face of another p-subobject, then that p-face disappears. If two coplanar p-faces which share a p-edge are assembled, then the shared p-edge disappears because that p-edge is traversed in the opposite directions.

While taking the projections, a p-edge should appear

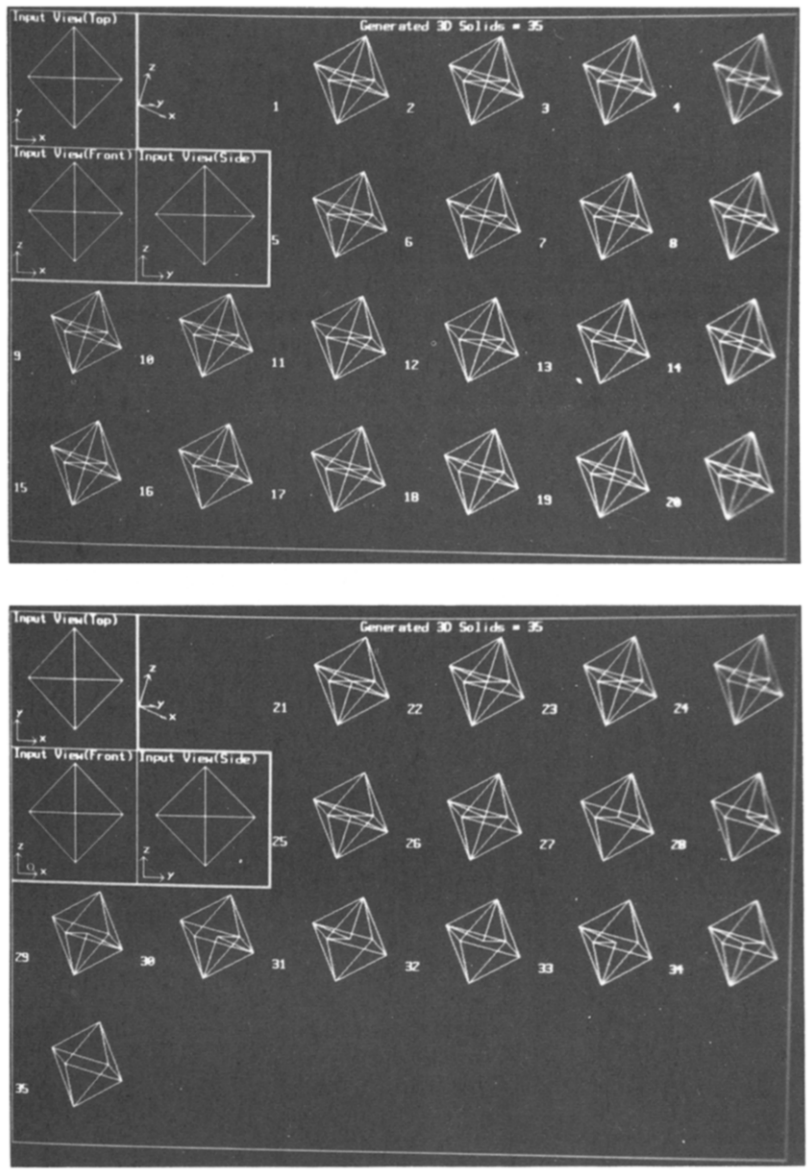


Fig. 20. Example 12.

as a 2D vertex or as a 2D line. If it is a 2D line, then it should appear as a solid 2D line if it is not obscured by any p-face or as a dotted 2D line if it is obscured by a p-face. If a solid and a dotted 2D line are overlayed, then the resultant line should be a solid line. Some p-edges may be partly obscured by p-face(s); in that case the 2D line is broken into solid and dotted 2D lines. After getting all the 2D lines, collinear 2D lines which are of the same type are joined together and stored as a single 2D line. The regenerated 2D lines are compared with the original 2D lines; if they match exactly, then the assembled p-object is one of the solutions or the only solution.

3. RESULTS AND DISCUSSION

The entire algorithm has been implemented in C on the IRIS 1400 Silicon Graphics Workstation. The program consists of about 4700 lines of code including about 700 lines of comments. This program accepts the data for orthographic views as input and generates the data for p-subobject(s), final object(s) and that required for the ray tracing program [3, 21, 22]. The p-subobject(s) and the final object(s) are displayed using display programs which make use of the graphics capabilities of the host computer.

Figs. 9 to 20 show the input orthographic views and the corresponding generated object(s) (which are actually generated as solid models in terms of their bounding faces). Table 2 lists the various input and output parameters as well as the time taken to generate the solutions. The times may be brought down by further refinements.

Figs. 9 to 17 contain a unique solution each while the inputs in Figs. 18 to 20 have resulted in multiple solutions in each case.

For Figs. 9 to 15, all the three orthographic views

Table 2. Analysis of the results.

Fig. no.	No. of 2D non-collinear lines in input views: Front (solid + dashed)	Top (solid + dashed)	Side (solid + dashed)	No. of dissimilar orthographic views	No. of p-subobjects	No. of solutions	No. of certain p-subobjects: After 1st categorization	After 2nd categorization	No. of uncertain p-subobjects (n)	No. of combinations ($2^n - 1$)	Time taken (m:s)
9	6 + 0	8 + 0	5 + 0	3	2	1	2	2	0	0	2:54
10	9 + 2	9 + 0	5 + 2	3	1	1	1	1	0	0	3:32
11	7 + 8	11 + 0	12 + 0	3	3	1	3	3	0	0	5:16
12	24 + 0	8 + 2	6 + 2	3	1	1	1	1	0	0	20:39
13	9 + 6	9 + 8	18 + 3	3	2	1	2	2	0	0	26:51
14	6 + 0	6 + 0	6 + 0	3	2	1	1	1	1	1	3:35
15	7 + 0	7 + 0	6 + 1	3	4	1	1	1	3	7	6:24
16	5 + 0	6 + 0	5 + 0	1	2	1	2	2	0	0	2:56
17	8 + 4	8 + 4	4 + 4	1	11	1	2	5	6	63	105:08
18	8 + 4	8 + 4	8 + 4	3	9	2	1	4	5	31	81:03
19	8 + 0	6 + 0	8 + 0	1	6	7	0	0	6	63	19:48
20	6 + 0	6 + 0	6 + 0	0	8	35	0	0	8	255	40:12

are dissimilar. For Figs. 9 to 13, all the p-subobjects are classified as certain p-subobjects, thus quickly giving rise to a unique solution. Note that the timings for Figs. 12 and 13 are relatively large since the objects are fairly complex.

Two input views are similar for Figs. 16 and 17. The number of uncertain p-subobjects for Fig. 17 is large resulting in a large number of combinations.

For multiple solution cases, *i.e.*, for Figs. 18 to 20, the number of dissimilar views are 3, 2 and 0 respectively. The times taken to generate the solutions are 81:03, 19:48 and 40:12 (min:sec.), respectively. The number of combinations of p-subobjects are 31, 63 and 255, respectively. It is interesting to notice that the time taken to generate the 2 solutions of Fig. 18 is twice that required for generating 35 solutions of Fig. 20 although the number of combinations for Fig. 18 is 8 times lesser than that of Fig. 20. The main reason is the complexity of each of the p-subobjects.

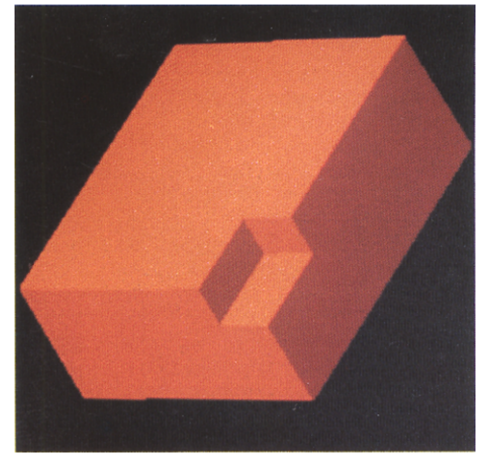
The multiple solutions of Figs. 18 and 19 are relatively easy to understand since each one is different from the others. Many of the solutions in Fig. 20 look identical. However, each one is generated by different combinations of the p-subobjects; the complete analysis of this case is very interesting and is given in [21].

Fig. 21 shows three objects which are obtained from the orthographic input views using this algorithm and then ray traced with the help of a ray tracing algorithm [3]. Fig. 21(a) corresponds to the object shown in Fig. 12, whereas Fig. 21(b) corresponds to the object shown in Fig. 14. The orthographics views given in Fig. 21(c) give rise to the object represented in Fig. 21(d).

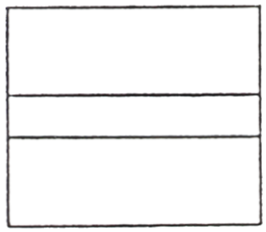
Based on the above examples, the following observations can be made about the algorithm:

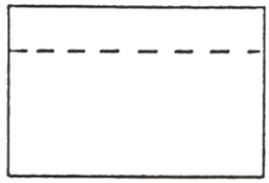
1. The time required for generating a 3D object is directly proportional to the complexities of the input views when all the input views are dissimilar.
2. The number of solutions is reduced considerably as the number of dissimilar input views increases. In most cases, when all three views are dissimilar, a unique solution is obtained; only in one case, two solutions have been found. Three cases, where two input views in each case are similar have been considered; two give rise to a unique solution each, while the third generates seven solutions. One case, where all the input views are similar (*i.e.*, no dissimilar views), generates 35 solutions. However, one obvious exception is a cube for which three views are identical; here, only one solution exists and is found quickly. Thus, in general, no direct correlation exists between the number of dissimilar views and the time taken by the algorithm.
3. It is conjectured that the more complex the input views, the more p-subobjects are classified as certain p-subobjects, thus reducing the number of combinations and therefore possibly the number of solutions.
4. The time taken for generating the solution(s) increases as the number of uncertain p-subobjects increases.

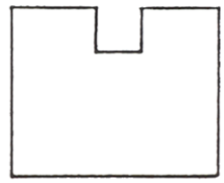
(a)



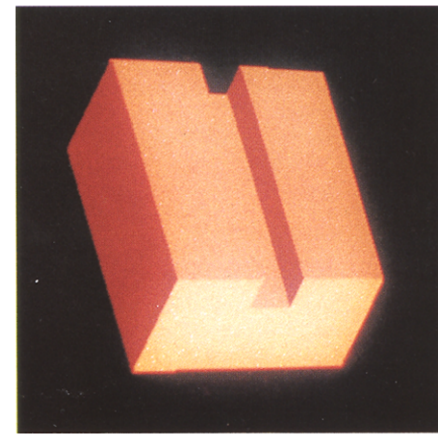
(b)







(c)



(d)

Fig. 21. Ray traced 3D solids.

5. The time taken by the algorithm depends upon the complexities of the p-subobjects.

4. CONCLUSIONS

An algorithm to generate solid objects from the conventional three engineering orthographic views has been designed and implemented. The complete details of the algorithm are included. The algorithm works for objects made up of planar surfaces; *i.e.*, for those orthographic views which contain only straight lines (solid and dotted).

The method in which the probable faces are connected to form the probable subobjects is based on the surface normals and the direction of travel of the connecting probable edges. This method has proved to be very useful. Although surface normals have been mentioned in the literature[14, 17], no details are available.

The probable subobjects are classified as certain and uncertain p-subobjects based on two methods of categorizations. These methods are novel, interesting and useful in the sense they reduce the number of uncertain p-subobjects drastically thus reducing the time taken by the algorithm by an order of magnitude.

Several illustrative examples have been included to demonstrate the variety of cases that can be handled by the algorithm. The algorithm successfully generates all possible multiple solutions. The time taken by the algorithm depends upon the complexities of the input views, the number of p-subobjects, the number of p-subobjects that can be categorized as certain and uncertain p-subobjects and the complexities of the p-subobjects.

Further work to accomodate curved surfaces, sectional orthographic views, auxiliary views, etc., is underway.

REFERENCES

1. Y. T. Lee and A. G. Requicha, Algorithms for computing the volume and other integral properties of solids. *Comm. of the ACM* **25**(9), 635–650 (1982).
2. A. G. Requicha, Representations for rigid solids: Theory, methods and systems. *ACM Comp. Surveys* **12**(4), 437–464 (1980).
3. N. N. Datar, Some methods for generating three-dimensional objects. M.Sc. (C.S.) Thesis, School of Computer Science, University of New Brunswick, Fredericton, N.B., Canada (April 1986).
4. U. G. Gujar, V. C. Bhavsar and N. N. Datar, Interpolation techniques for 3-D object generation. *Comp. & Graphics* **12**(3/4), 541–555 (1988).
5. U. G. Gujar, 3-D graphics in APL: User perspective. TR84-025, School of Computer Science, University of New Brunswick, Fredericton, N.B., Canada (July 1984).
6. U. G. Gujar, Generation and manipulation of three-dimensional objects. *Proceedings of the Annual APICS Computer Science Conference,* University of New Brunswick, Fredericton, N.B., Canada, 64–72 (November 1984).
7. U. G. Gujar, A three dimension wireframe graphics system. *APL-87 Conference Proceedings*, Dallas, Texas, 1–16 (May 10–14, 1987).
8. N. N. Datar, U. G. Gujar and V. C. Bhavsar, Some methods for generating three-dimensional objects. *Proceedings of the Annual APICS Computer Science Conference,* Dalhousie University, Halifax, NS., Canada, 14–32 (November 1985).
9. I. V. Nagendra and U. G. Gujar, 3-D Objects from 2-D orthographic views—A survey. *Comp. & Graphics* **12**(1), 111–114 (1988).
10. M. Idesawa, A system to generate a solid figure from a three view. *Bull. JSME* **16**, 216–225 (February 1973).
11. G. Lafue, Recognition of three-dimensional objects from orthographic views. *ACM/SIGGRAPH*, 103–108 (July 1976).
12. T. C. Woo and J. M. Hammer, Reconstruction of three-dimensional designs from orthographic projections. *Proc. 9th CIRP Conference,* Cranfield Institute of Technology, Cranfield, England, 247–255 (1977).
13. K. Preiss, Algorithms for automatic conversion of a 3-view drawing of a plane faced part to the 3-D representation. *Comp. in Industry* **2**(2), 133–139 (1981).
14. G. Markowsky and M. A. Wesley, Fleshing out projections. *IBM J. Res. & Develop.* **25**, 934–954 (November 1981).
15. R. M. Haralick and D. Queeney, Understanding engineering drawings. *Comp. Graphics and Image Processing* **20**(3), 244–258 (1982).
16. B. Aldefeld, On automatic recognition of 3D structures from 2D representations. *Computer-Aided Design* **15** (2), 59–64 (1983).
17. H. Sakurai and D. C. Gossard, Solid model input through orthographic views. *ACM/SIGGRAPH* **17**(3), 243–252 (1983).
18. B. Aldefeld and H. Richter, Semiautomatic three-dimensional interpretation of line drawings. *Comp. & Graphics* **8**(4), 371–380 (1984).
19. K. Preiss, Constructing the solid representation from engineering projections. *Comp. & Graphics* **8**(4), 381–389 (1984).
20. H. Yoshiura *et al.*, Top-down construction of 3-D mechanical object shapes from engineering drawings. *IEEE Comp. Magazine*, 32–40 (December 1984).
21. I. V. Nagendra, Three dimensional solids from orthographic views. M.Sc.(C.S.) Thesis, School of Computer Science, University of New Brunswick, Fredericton, N.B., Canada (May 1986).
22. I. V. Nagendra and U. G. Gujar, An algorithm for generating three dimensional solids from orthographic views. *APICS Computer Science Conference,* Memorial University of Newfoundland, 26–40 November 7–8, 1986).
23. G. Markowsky and M. A. Wesley, Fleshing out wire frames. *IBM J. Res. & Develop.* **25**, 934–954 (November 1981).
24. A. Bowyer and J. Woodwark, *A Programmer's Geometry,* Butterworths, London, England (1985).
25. W. K. Giloi, *Interactive Computer Graphics*, Prentice-Hall, Inc., Englewood Cliffs, NJ, U.S.A. (1978).

APPENDIX I
COMPUTATION OF SURFACE NORMALS

The p-edge closed loops are generated in a manner which is similar to the one proposed by Markowsky and Wesley [14, 23]. In order to generate such loops, surface normals are used. Since every surface has two sides, it is assumed that the generation of p-edge loops is carried out on that side of the surface where the surface normal points away from the origin. The equation of the surface normal which points away from the origin is derived as follows (see Fig. I1).

The equation of a line in space which passes through a point (x_o, x_o, z_o) can be written as

$$\frac{x - x_o}{f} = \frac{y - y_o}{g} = \frac{z - z_o}{h} = r \tag{I1}$$

where, a unit vector, V, along the line is given by

$$V = f\underline{i} + g\underline{j} + h\underline{k} \tag{I2}$$

where, $\underline{i}$, $\underline{j}$ and $\underline{k}$ are the unit vectors along the principal axes x, y and z, respectively. Note that

$$\frac{x - x_o}{sf} = \frac{y - y_o}{sg} = \frac{z - z_o}{sh} = t \tag{I3}$$

also represents the same line as above for $s \neq 0$.

Since the equation of the plane is represented as $ax + by + cz + d = 0$, the equation of the vector, N, which is normal to the plane can be written as

$$N = a\underline{i} + b\underline{j} + c\underline{k}. \tag{I4}$$

Hence, a unit vector, M, along the normal N is given by

$$M = (a\underline{i} + b\underline{j} + c\underline{k})/\text{sqrt}(a^2 + b^2 + c^2), \tag{I5}$$

that is,

$$M = q(a\underline{i} + b\underline{j} + c\underline{k}) \tag{I6}$$

where

$$q = 1/\text{sqrt}(a^2 + b^2 + c^2). \tag{I7}$$

Now, the equation of the line passing through the origin (*i.e.*, $x_o = 0$, $y_o = 0$, and $z_o = 0$) and along the direction of vector N can be written as

$$\frac{x}{qa} = \frac{y}{qb} = \frac{z}{qc} = r \tag{I8}$$

or

$$\frac{x}{a} = \frac{y}{b} = \frac{z}{c} = qr = t. \tag{I9}$$

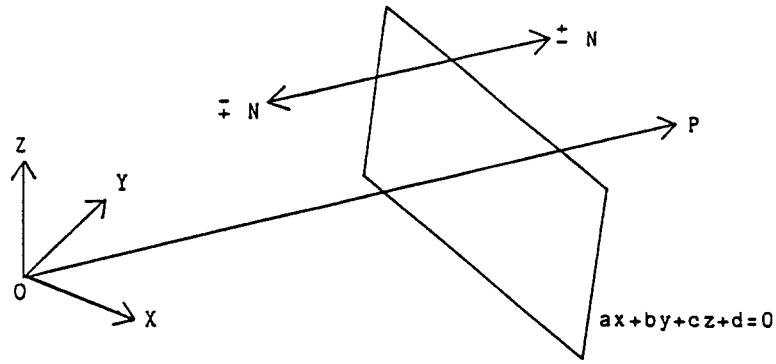


Fig. I1. Surface normal pointing away from the origin.

Therefore,

$$x = at, \quad y = bt, \quad z = ct. \tag{I10}$$

Substituting the values of x, y and z in the equation of the plane, one obtains

$$a^2t + b^2t + c^2t = -d \tag{I11}$$

or

$$t = -d/(a^2 + b^2 + c^2). \tag{I12}$$

By substituting the value t in Eq. (I10), the equations for x, y and z can be expressed in terms of the constants a, b, c and d.

Therefore, the representation of the vector OP which is normal to the plane and which also points away from the origin is

$$OP: \quad (x - 0)\underline{i} + (y - 0)\underline{j} + (z - 0)\underline{k} \tag{I13}$$

or, substituting for x, y, z from Eq. (I10),

$$OP: \quad -d(a\underline{i} + b\underline{j} + c\underline{k})/(a^2 + b^2 + c^2). \tag{I14}$$

If $d = 0$, or is in the vicinity of 0 for computational purposes, that is, if the plane passes through the origin, then the equation of the normal, as given by Eq. (I4) points away from the origin and one does not have to consider Eq. (I14).