Reconstruction of curved solids from two polygonal orthographic views

D Dutta and Y L Srinivas

It has long been known that the 2-view orthographic representation of a mechanical part is ambiguous. Existing literature addresses the recognition/reconstruction of solids (mostly polyhedral solids) from either 1-view (perspective) drawings, or from three orthographic views. The paper presents algorithms for reconstructing curved solids from only two polygonal orthographic views. First, all the polyhedral solids that correspond to the pair of prespecified orthogonal views are reconstructed, and then the associated third views are developed. Then, for each solid, the three views are analyzed simultaneously, and the authors reason about the possibilities of including circular arcs in the newly generated third view. This corresponds to the inclusion of solids that are bounded by quadrics. Possible application areas include computer-aided design, machine vision and automated inspection systems.

orthographic views, solid reconstruction, line-drawings automatic conversion

INTRODUCTION

2-view orthographic representation

The use of parallel projections for unambiguous representations of solid objects was developed by the French mathematician Gaspard Monge in the middle of the 18th century. It has since formed the basis of what is now referred to as *engineering drawing*, *multiview drawing* and *orthographic drawing*. In this paradigm, every 3D object can be completely described by a set of three orthographic views (projections) on mutually perpendicular planes. Of the six possible orthographic views (obtained when the object is thought of as being enclosed in a transparent box), the views most commonly used in engineering are the *front*, *top* and *right-side* views. While simple objects such as cylinders, bushings and bolts can typically be described by only a pair of orthographic views, the precise description of complex machine elements often requires sectional and/or auxiliary views in addition to the three views.

This paper is interested in the inverse problem, that of developing solids whose orthographic views are given. In particular, the problem in which only two of the three views are given is considered. There are two aspects of this problem that contribute to its complexity. First, it is well known that a pair of orthogonal views (e.g. the front and top views) correspond, in general, to more than one solid, each giving rise to a unique third view (i.e. the right-side view). Thus, the 2-view representation of most objects is ambiguous. Second, straight lines in one of the two views can correspond to curves in the missing third view, giving rise to solids bounded by nonplanar surfaces.

Fundamental in engineering graphics and descriptive geometry, and usually encountered by students in their first engineering drawing/graphics course, is the missing-view problem: given two views, generate the third view(s). This problem can be surprisingly complicated. It is so because every correct solution to this problem first requires the visualization of a solid that corresponds to the two given views. The third view can then be determined. Therefore, visualization in 3D is required, and this, in itself, is a cause of frustration. Further, as a 2-view representation is nonunique, the completeness of the solution can be difficult to verify.

If an object has several features that project to a single line in one or more views, the reconstruction of the same object starting from the three views can be very complicated. Lines and points on the given views must be systematically analyzed in the process of developing a correct solid. When only two views are given, the problem is clearly more difficult. For a simple object, it is best to visualize it partially from the two given views, and then complete the process by the addition of

Design Laboratory, Department of Mechanical Engineering, University of Michigan, Ann Arbor, MI 48109, USA
Paper received: 31 July 1991. Revised: 14 October 1991

 0010-4485/92/030149-11

consistent right-side views, one by one. However, if the given views are complex, the powers of spatial visualization become inadequate in the guidance of the analysis, which is inherently combinatorial. That there has been no reliable way to determine exactly how many solids correspond to a given pair of orthogonal views is therefore not surprising.

This paper presents an algorithm for the reconstruction of all the solids that are bounded by planes and quadrics when only two orthographic views are given. First, the algorithm generates all the polygonal third views (i.e. the views that are composed of straight lines only) that correspond to a given pair of orthogonal views. It does so by reconstructing all the polyhedral solids that correspond to the orthogonal views. The three views are then analyzed to identify the line segments in the right view that can be replaced by circular arcs (other conics can be included easily). Finally, the nonpolyhedral solids are generated from the modified three views.

The reconstruction procedure, which is akin to the gift-wrapping method of generating convex hulls, begins from a vertex, and faces are added one by one, until a solid is generated. Considering every valid triple of vertices, the algorithm evaluates all candidate faces for inclusion. It considers all the valid configurations of faces, and determines the complete set of valid solids that correspond to the given views. An empty solution set proves that the given views are inconsistent.

This section of the paper is concluded by a review of prior work on the generation of solids from line drawings, and potential applications are mentioned. The second section of the paper analyzes the problem and discusses the algorithm. The third and fourth sections contain the algorithms for polyhedral- and curved-solids generation. Examples are given in the fifth section.

Previous work

As a result of the use of digital computers for image processing, various aspects of the automatic recognition and reconstruction of line drawings have been researched since the 1960s. In particular, two variants of the problem discussed in this paper, i.e. reconstruction from *single-view* and *3-view* drawings, have received attention thus far.

An excellent survey of the research on the machine interpretation of line drawings can be found in the first few pages of a text by Sugihara[1]. In this book, Sugihara addressed, in particular, the recognition of polyhedral objects and scenes described by single-view line drawings (which are also referred to as perspective drawings). An important subclass of this problem is the interpretation of multiview line drawings.

Idesawa presented a method for the reconstruction of solid models from three orthographic projections[2]. This method, however, was capable of producing invalid objects. Lafue developed heuristics to detect such invalid objects by the imposition of a prespecified input format[3]. In 1980, Wesley and Markowsky[4] developed an algorithm for the production of orthographic views from wireframe models of mechanical parts. They further extended the method to construct polyhedral solids from the three orthographic projections[5]. Methods for the extraction of 3D information from orthographic views can also be found in References 6 and 7. In Reference 8, the Wesley and Markowsky approach was extended to include circular arcs in the orthographic views. The resulting solids, therefore, could include quadrics and toroidal surfaces.

The problem of generating all solids from only two views was, to the best of the authors' knowledge, first examined by Wilde[9]. Wilde examined the difficulties involved in spatial visualization, and, in particular, in the reconstruction of solids from two orthogonal views. This paper presents algorithms for the reconstruction of polyhedral and curved solids from two views.

Applications

An automatic procedure to generate solids from line drawings has many applications in engineering. For example, one can then verify the consistency of complicated engineering drawings automatically. Such an algorithm can also be part of an interactive system to assist a designer during the development of new product designs. Moreover, drawings continue to be the most natural way of conveying geometric information for designers. Therefore, a system to convert automatically engineering drawings into solid models can be very useful. The conversion of engineering drawings into solid models is a task that many industries have to address in the process of adopting solid-modelling systems. The recognition of orthographic projects also has applications in machine vision and automated inspection.

ANALYSIS OF RECONSTRUCTION PROBLEM

Canonical orientation

It is a common practice in engineering drawing to orient the object such that it yields the simplest orthographic views. It is easy to see that a simple object can produce complicated orthographic views if it is not oriented in the best possible way. For example, a cube standing on its vertex would produce a rather complicated set of orthographic views, with all its edges foreshortened. However, when three sides of the cube are parallel to the three principal planes of projection, the orthographic views are three squares. In the latter orientation, there is no foreshortening of the edges, and they appear in their *true lengths*. That orientation of an object that maximizes the number of edges that appear in true lengths in all three views is referred to as the object's *canonical* orientation. However, not all objects can be oriented such that the edges comprising the three views appear in their true lengths (e.g. an equilateral tetrahedron).

Edge and vertex multiplicities

When three orthographic views are provided, there is enough information in the drawings for it to be deduced which lines appear in true lengths. The reconstruction of

the solid is not difficult thereafter. However, when only two views are provided, the reconstruction procedure requires a systematic analysis of the views. Central to this analysis is the detection of line multiplicity, which is achieved by reasoning about vertex multiplicities. It is easy to see that more than one edge of a solid can project to the same line in one orthographic view. Such lines are considered as multiple lines. The multiplicity m of a line l in any view refers to the *maximum* number of edges that can project onto l while being consistent with the other view.

Multiple edges are able to be detected by reasoning about multiple vertices, as each edge must correspond to a pair of vertices. Therefore, the multiplicity m of vertex v refers to the *maximum* number of distinct vertices that it can conceal, in consistency with the views. The multiplicity of vertex v in the top/front can be obtained by the vertices in the front/top view that lie on the vertical line through v being counted. The following is intuitive.

Proposition 1: If vertices with multiplicities larger than 1 exist in two orthographic views, the pair corresponds to more than one valid solid.

Combinatorial analysis

An important difference between the problem discussed in this paper and its two well known variations (i.e. the generation of solids from a *perspective view* and from *three* orthographic views) is that the problem has multiple solutions, whereas the other two have exactly one solution each. In other words, it is necessary for both the correctness and the completeness of the solution to be determined for this problem, whereas the other two problems are solved by a correct solution. It is this difference that leads to the combinatorial nature of this problem.

A bound on the number of right views (and, hence, the distinct polyhedral solids) that correspond to a pair of orthographic views can be derived by analysis of the standard technique used in engineering drawing to develop third views from two given views. The procedure is illustrated in Figure 1, but discussions are not included. An explanation can be found in any standard text on engineering drawing, e.g. Reference 10.

In Figure 1, assume that the top and front views, shown in the top quadrant TQ and the front quadrant FQ, respectively, were generated by the analysis of a solid S with m vertices and n edges. By construction, all the candidate vertices in the right quadrant RQ are the intersections of the horizontal and the vertical lines drawn from the vertices in FQ and TQ, respectively. There are $O(m)$ vertices in TQ and in FQ. Therefore, RQ has $O(m^2)$ candidate vertices. However, the right view cannot have more vertices than its progenitor S. Thus, each correct right view R_i developed from the candidate vertices in RQ is a graph $G_i(V, E)$, and $V = O(m)$. (Note that firm lines in the right view cannot intersect, except at the candidate vertices. However, intersections between firm and hidden lines can create new vertices.) If $M = m^2$, there are C_m^M possibilities for the vertex set V.

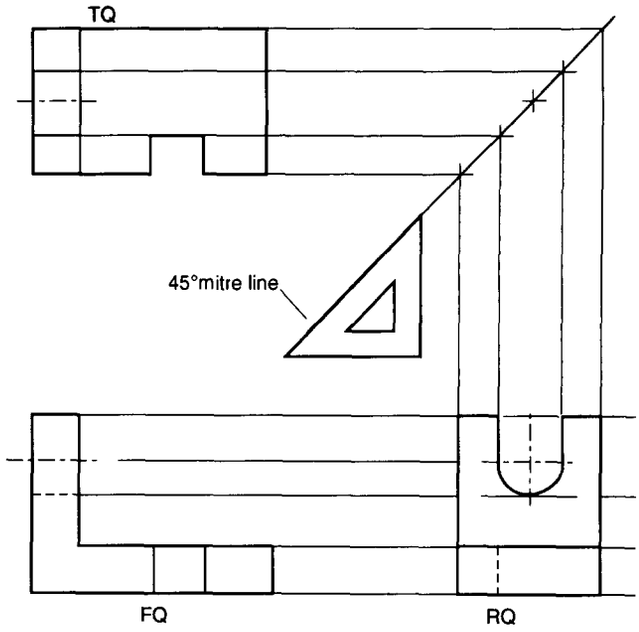


Figure 1. *Construction of right view from top and front views*

Consider a vertex set V_k of cardinality m. With m vertices, there can be $O(m!)$ trees*. However, the orthographic views cannot contain any trees, as trees correspond to dangling faces and edges (i.e. unbounded solids). Therefore, the set of valid right views P, which corresponds to vertex set V_k, consists of all the graphs, but no trees or dangling edges. For example, every triangulation of V_k is a topologically valid right view, as it contains cycles. It is easy to see that the cardinality of the set P is, in general, exponential in m. As there are C_m^M such vertex sets, the possibilities for right views are exponential in number. Clearly, this is a loose bound on the number of right views, as consistency with the top and front views is the final check for a valid right view. Moreover, the inclusion of nonpolyhedral solids in the above analysis results in an infinite number of valid right views.

Correctness of procedure

It is asserted that a 2-view representation can be unique under special conditions.

Lemma 1: If at least one of the two given orthographic views has no multiple vertices, the 2-view representation is unambiguous.

Proof: Let the top and front views be given, the former without any multiple vertices. The top view then determines the total number of vertices in the object and their unique connectivities (i.e. the edges). Clearly, the top view determines the unique topology of the solid. By definition, the front view contains the elevation of each vertex that appears in the top view. Therefore, the given representation is unambiguous. □

* For example, with three vertices A, B and C, there can be trees such as A-B-C, A-C-B, B-A-C and so on.

Note that a pair of orthographic views, as per Lemma 1, does not, in general, facilitate visualization. However, for the purposes of the reconstruction algorithms, they are valid inputs. If the multiplicity of a vertex is n, it can conceal up to $n-1$ vertices in the view. Therefore, such a vertex is accounted for in the construction procedure by consideration of the view in which the vertex appears n times, once with each value of the vertex multiplicity (i.e. $0, 1, \ldots, n-1$).

Theorem 1: In the reconstruction of all the valid solids from a pair of orthographic views, it is necessary and sufficient to analyze only one view, and to account for all the vertices in that view that have a multiplicity larger than 1.

Proof: The necessity follows from Proposition 1; this proof covers sufficiency. Accounting for a multiple vertex v in a view requires consideration of n instances of the view, once with each value of the multiplicity of v. The companion view is used for validity checks. Clearly such an accounting procedure is exhaustive. Sufficiency is then an immediate consequence of Lemma 1. □

Any procedure that does not account for all the vertices with multiplicities larger than 1 is, by Theorem 1, incomplete. The multiple vertices are accounted for in one view by the direct construction of the solid while the other view is used to verify consistency. The algorithm is based on an enumeration scheme. It considers all the possible configurations that can arise from the two given views. From these candidate configurations, only those that correspond to valid representations of solids are identified and stored. As all the candidate configurations are enumerated, the algorithm ensures that the solution set is complete, i.e. that there are no solids other than those in the solution set that generate the given pair of orthographic views.

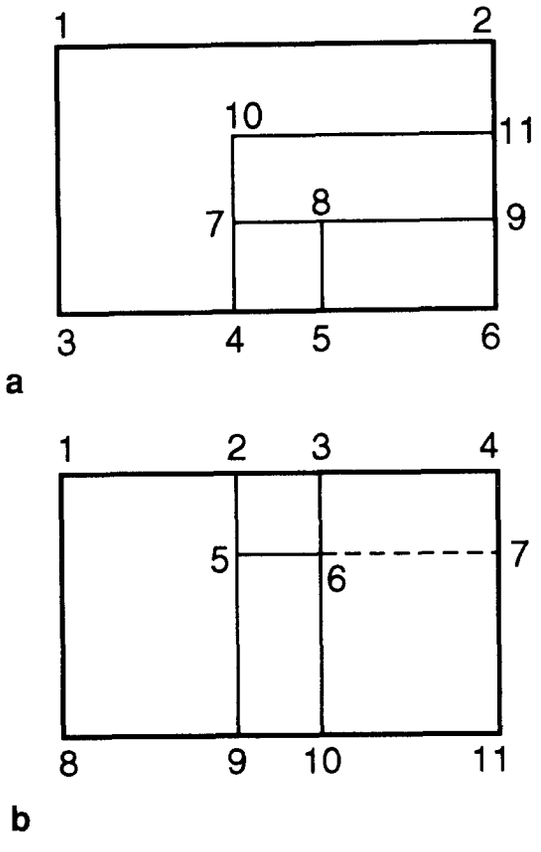


Figure 2. *Orthogonal views; (a) top view, (b) front view*

OVERVIEW OF ALGORITHM

Algorithm input

The inputs to the algorithm are two vertex connectivity matrices that correspond to the given views. The coordinates of each vertex are also supplied to the algorithm. Each connectivity matrix is an $n \times n$ square matrix, where n is the number of distinct vertices in the corresponding view. Figure 2 shows a pair of orthographic views. The associated connectivity matrices are as follows (the first matrix corresponds to the top view, and the second matrix corresponds to the front view):

$$\begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\
0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1
\end{bmatrix}$$

$$\begin{bmatrix}
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 & 1 & 2 & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 1 & 2 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 2 & 2 & 1 & 0 & 0 & 0 & 1 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}$$

An element $C(i, j)$ of the matrix has a value of 0 if node i is not connected to node j, a value of 1 if node i is connected to node j by a complete solid line, and a value of 2 if node i is connected to node j by a complete or partially hidden line. For example, in Figure 2b, vertex 5 is connected to vertex 7 by a partially hidden line, and so $C(5, 7) = 2$. The connectivity matrices are symmetric.

Description of algorithm

The algorithm works as follows. As a vertex in any view can actually correspond to the orthogonal projections of one or more lines, a vertex analysis is first performed to detect such multiple vertices, and their multiplicities are recorded. For example, vertex 11 in Figure 2b has a multiplicity of 4, as there are, at most, four vertices in the top view, i.e. 6, 9, 11 and 2, whose projections can coincide with it.

Next, the vertex with the lowest multiplicity, v_1 say, is selected. From the connectivity matrices, all the neighbors of v_1 are identified and stored in V. Every noncollinear triple in V containing v_1 defines a plane that passes through v_1. These planes are generated and stored in P. Finally, a set of faces F is created by the systematic detection of all the vertices, except v_1 and its neighbors, that lie on each element of P. Therefore, each element of F is a face that contains the vertex v_1.

Next, the faces in F are analyzed. In particular, it is necessary to determine how many faces of F can lead to a valid solid. A solid is considered to be valid only if it corresponds to the given views. The analysis involves the expansion of faces of F, and it proceeds as follows. A face f_1 is selected from F. If f_1 is to be the face of a valid solid, each edge of f_1 must be shared by exactly one more face*. The neighbors of f_1 are identified and stored in G_1. This process is repeated for every face in F. The set G_i includes only those faces encountered in the analysis that do not intersect any face in $G_{i-1}, G_{i-2}, \ldots, G_1$ (i.e. it excludes nonmanifolds). Further, consistency with the given orthographic views is verified prior to the addition of a face in G_i.

A depth-first strategy is used, and the faces of G_1 are expanded next. Concurrently with the face-expansion procedure for G_1, the constructed solids are checked for completeness (i.e. that they are bounded solids) and validity (i.e. consistency with the given orthographic projections). Any solid that passes this check is a valid solid, and it is stored in the solution set. The connectivity matrix that corresponds to the right view is then generated. The face-expansion process is carried out for all sets G_i until all the faces are expanded. The algorithm then terminates, as all the solids that correspond to the given pair of orthographic views have been identified, and their right-view connectivity matrices have been stored.

Now the three views for each solid have been obtained. The last step is to identify from these views which lines in the right view can be replaced by circular arcs. (Note that the concern is only to introduce curves in the right view, and not in the top and front views, which were prespecified.) It is assumed that the solid being analyzed is in a canonical orientation. Once again, one view is analyzed, and the other is used to check consistency. Clearly, all the vertical lines in the top view are candidates for curved edges when they are viewed from the right. Further, a vertical line in the top view can only correspond to an inclined line, or a horizontal line, in the right view. If it corresponds to an inclined line $l_1 = (u, v)$, it can be replaced directly by a circular arc C. C passes through u and v, and it has l_2 and l_3 as tangents, where l_2 and l_3 intersect l_1 at u and v, respectively. If the vertical line in the top view corresponds to a horizontal line in the right view, the corresponding solid has a sharp corner (and an edge) that is a candidate for 'rounding' or 'filleting'. Further checks are carried out to ensure consistency with the front view, and the corner/edge in the solid is rounded, or filleted, by the replacement of the appropriate vertex in the right view with a circular arc. The blend radius can be chosen by the user, and it determines the point of tangencies.

* Nonmanifold objects are not considered.

ALGORITHM

The main algorithm invokes three substeps: the form-face, expand-face and curves steps. The first two steps are recursive.

MAIN algorithm

MAIN algorithm

```
Input: connectivity matrices for orthographic view R
and T
Output: solids corresponding to R and T

  begin

  for each vertex in R, compute multiplicity

  for minimum-multiplicity vertex v1, while multiplicity
  > 0
    F ⇐ form-face (vi)
    while F not empty, expand-face(F)
    P ⇐ polyhedral solids; Q ⇐ right views
    while Q not empty, curves(Q)
      C ⇐ curved solids

  output P and C

  end
```

The MAIN algorithm requires that, in the orthographic views, there be a path from each vertex to every other vertex (i.e. no edge loop is properly contained in another). This condition disqualifies solids that have holes or pockets that open on planes. The algorithm can be modified to overcome this limitation by the addition of dummy edges between the appropriate vertices of the disconnected loops.

A brief description of the procedures used by the MAIN program is given below:

- *Input-connectivity-matrices (top-view, front-view):* This reads the data for both views from the input file, and returns the data in matrices *top-view* and *front-view*.
- *Generate-multiple-nodes (top-view, front-view, multiple-node-list):* This generates all the possible multiple nodes in the front view from the information given in

the *top-view* and *front-view* matrices, and returns the list of multiple nodes, properly labelled, in *multiple-node-list*.

- *Get-min-mnode (multiple-node-list, min-loc):* This determines the node in the front view that has the least number of multiple nodes in the front view, and returns the identification of the node in *min-loc*.
- *Get-mnodes (min-loc, multiple-node-list, mnode-list):* This extracts all the *mnodes* at node *min-loc* in the front view from the *multiple-node-list*, and returns the resulting list in *mnode-list*.
- *Get-connected nodes (mnode, connected-list):* This determines all the *mnodes* in the *multiple-node-list* that are connected to *mnode*, and returns the resulting set in *connected-list*.
- *Get-next-plane (mnode, connected-list, face):* This determines the next candidate plane in the *connected-list* that passes through *mnode*. The previous plane, if any, is passed via *face*. The resulting plane is returned in *face*.
- *Get-nodes-in-plane (face, vertices):* This determines all *mnodes* that lie in the plane defined by *face*, and returns this mnodes list in *vertices*.
- *Initial-solid (solid, face):* This sets up the initial *solid* with a single face that is defined by *face*.
- *Output-solution( ):* This outputs the solution set of solids.

Maintenance of candidate list

In a process that is inherently combinatorial, the prompt rejection of invalid elements can substantially reduce a candidate list. Throughout the reconstruction process, topological and geometric inconsistencies in each configuration of candidate faces, edges and vertices are detected, and the invalid elements are rejected outright.

In the MAIN algorithm, vertex triples define an initial set of candidate planes through the chosen vertex v_1. These planes are carriers of faces that pass through v_1. However, not all vertex triples are candidates. All collinear triples and dependent triples of vertices are rejected (see, for example, Figure 3). In Figure 3, vertex triples (1, 2, 3) and (5, 2, 4) define the same plane, and so (5, 2, 4) is considered as a dependent triple, and it is rejected.

When MAIN invokes the form-face algorithm, each valid vertex triple (v_i, v_j, v_k) (i.e. a carrier plane for some face) is passed along with lists of additional vertices, if any, that also lie on the same carrier plane. Every such carrier plane is a candidate that is evaluated by the form-face algorithm.

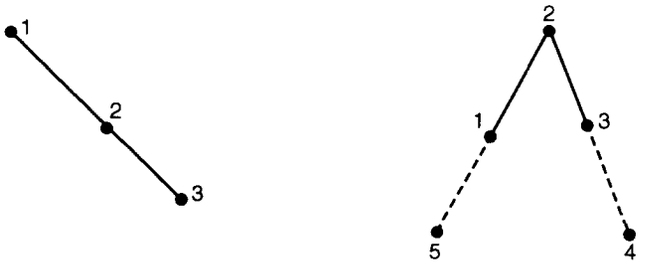


Figure 3. *Invalid vertex triples in* MAIN *procedure*

Form-face algorithm

The form-face procedure recursively generates all the possible faces that can be produced from the vertex set.

Form-face algorithm

(1) Identify all the neighbors of the last vertex of the face
(2) For each of the vertices in this list, repeat
(2.1) Add the vertex to face at the current last vertex
(2.2) If the face is complete and valid add to the list of possible faces
(2.3) Mark this vertex as USED in the list of vertices that lie in the plane
(2.4) Invoke form-face( ) to generate the possible faces for the updated face
(2.5) Remove this vertex (the last vertex) from the face
(2.6) Mark the vertex as FREE in the list of vertices that lie in the plane
(3) End of procedure; Return

The procedures used by the form-face algorithm are briefly described below:

- *Get-next-vertex-list (face, vertices, next-list):* This procedure returns all the FREE members of vertex list *vertices* that can be connected to the last vertex of *face*. The list of these vertices is returned in *next-list*.
- *Add-vertex-to-face (face, vertex):* This procedure expands the partially constructed *face* by adding *vertex* to the last vertex of the face and updating *vertex* as the last vertex of *face*.
- *Valid-face (face):* This procedure verifies whether a given *face* is a complete and consistent face, i.e. the edges form a closed loop, and do not cross each other.
- *Add-face (face):* This procedure stores *face* in the list of candidate valid faces.
- *Mark-vertex-list (vertices, vertex, value):* This procedure marks *vertex* in the list *vertices* with the value *value*. IF *value* is USED, then *vertex* will not be used as a candidate in subsequent searches. If *value* is FREE, *vertex* will be used.
- *Remove-vertex-from-face (face, vertex):* This procedure removes *vertex* from the end of the partially constructed *face*. The previous vertex in *face* now becomes the last vertex.

Maintenance of candidate list

The form-face algorithm seeks to construct valid faces for the solid from the set of carrier planes supplied by MAIN and a set of vertices obtained from the connectivity matrices. The size of the candidate list of faces is kept at a minimum by the performance of checks to preserve the

topological and geometric validity of the partially constructed solid.

The topological check consists of the retrieval, for each carrier plane P_i, of the set S_i of all the associated vertices that can possibly be connected to the last vertex v_k of P_i. The set S_i can be constructed from the connectivity matrices. A face is considered to be topologically valid if it consists of a single, closed edge loop. Therefore, only those vertices from the set S_i that satisfy the above condition are added to the initial triple of vertices; all others are rejected. This avoids the consideration of dangling edges (see Figure 4a) and 'faces within faces' as candidates (see Figure 4b). Next, the geometric validity is considered. For a face to be geometrically valid, it is required that edges intersect only at the common vertices. Thus, if the addition of a vertex from the set S_i produces intersecting edges (see Figure 4c), the vertex is rejected. Every face that passes the above-mentioned checks is a candidate for evaluation by the expand-face algorithm.

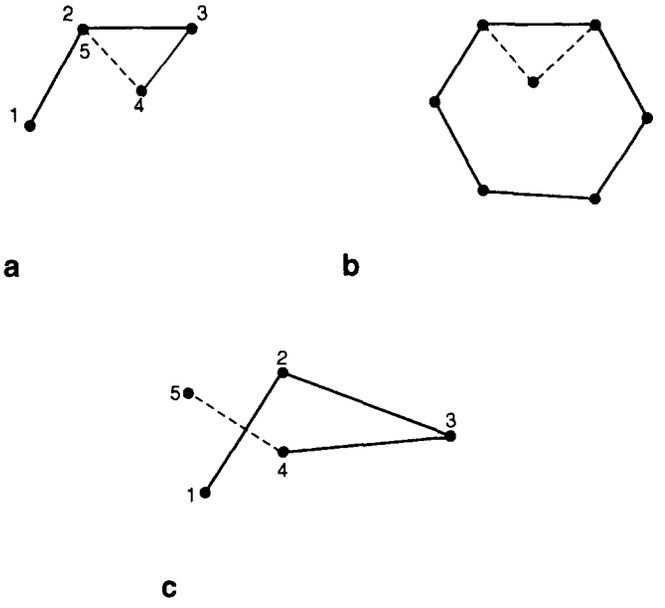


Figure 4. *Rejected candidate faces in form-face procedure; (a) dangling edges, (b) faces within faces, (c) intersecting edges*

Expand-face algorithm

Expand-face( ) recursively expands all the edges of the face. It also checks for the consistency and validity of the partially constructed solid, and stores any valid solids that may be encountered during the process. This algorithm is similar to MAIN in that it generates triples that correspond to the carrier planes that contain possible faces.

Expand-face algorithm

(1)	For each edge of the face repeat
(1.1)	Extract vertices in face that correspond to the edge
(1.2)	If edge is already shared by two faces go to Step 1.5
(1.3)	Generate list of faces that can be incident to edge
(1.4)	For each face in this list repeat
(1.4.1)	Add face to the partially constructed solid
(1.4.2)	If solid is complete and valid, add to solution
(1.4.3)	Invoke expand-face( ) to expand this face
(1.4.4)	Remove face from solid
(1.5)	Continue until the list is completely processed
(2)	End of procedure; Return

The procedures used by the expand-face algorithm are briefly described below:

- *Number-of-edges (face):* This determines the number of edges in *face*.
- *Extract-edge (face, i, edge):* This extracts the edge *i* from *face*. This edge is returned in the variable *edge*.
- *Already-shared (edge, solid):* This checks whether *edge* is already shared by two of the faces of the partially constructed *solid*. If so, if returns TRUE; else returns FALSE.
- *Get-next-face-list (edge, face, solid, face-list):* This gets the list of all the other possible faces for the *edge* that lies on *face*. This procedure checks and returns only those faces that, when added, preserve the consistency of the partially constructed *solid*. The result is returned in *face-list*.
- *Add-face-to-solid (solid, face):* This grows the partially constructed *solid* by adding *face* as one more face of *solid*. It assumes that *face* qualifies to be a valid face of *solid*.
- *Valid-solid (solid):* This checks whether *solid* represents a valid solid that can be a member of the solution set. It checks whether (a) all the edges are shared by exactly two nodes, and (b) the solid produces all the vertices and lines in the given view information. If *solid* passes these checks, it returns TRUE, else returns FALSE.
- *Store-solid (solid):* This adds *solid* as a member of the solution set.
- *Remove-face-from-solid (solid, face):* This removes the face *face* from a partially constructed solid *solid*. It assumes that the resulting partial solid is still valid (i.e. that it does not contain discontinuities).

Maintenance of candidate list

The generation of vertex triples in the first part of the expand-face procedure is similar to what happens with MAIN, but the triples satisfy additional constraints. First, the triple must share the two vertices of an existing edge of the face. Second, the addition of the new third vertex must not result in edges that pierce through existing faces of the solid (see Figure 5a). Finally, the face normal defined by the new triple must be distinct, thereby disallowing more than one face on a single carrier plane (see Figure 5b).

The vertex generation in the second part of the expand-face procedure is similar to that in the form-face

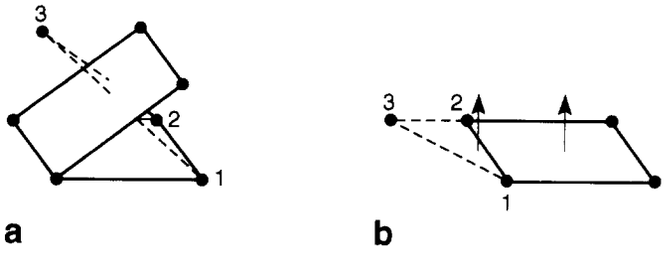


Figure 5. *Invalid cases in the expand-face procedure; (a) edges that pierce through existing faces, (b) more than one face on single carrier plane*

algorithm. The additional restrictions that need to be adhered to in the choosing of new vertices include the restrictions stated above (i.e. for Figures 5a and b), and the restriction that the resultant edge can be shared by exactly one more face. When loop closure is achieved, the resultant face is added to the solid.

As invalid elements are detected and eliminated throughout the generation process, a pruned candidate list is maintained, and all the elements introduced into the solution are valid. Thus, for example, the possibility of an edge carrying one or more invalid faces does not arise. The solid is considered to be complete (i.e. to be a solution to the missing-view problem) when each edge has two incident faces and is consistent with the given orthographic views.

Curves algorithm

Curves( ) detects the edges in the newly generated right views that can be replaced by circular arcs, while remaining consistent with the specified top and front views.

Curves algorithm

(1)	$E \Leftarrow$ all vertical lines in top view.
(2)	For each element e_i of E repeat
(2.1)	Find corresponding edge r_i in right-side view
(2.2)	If r_i is not horizontal, replace r_i by a circular arc
(2.3)	If r_i is horizontal, repeat
(2.3.1)	If edges intersecting r_i are vertical, replace r_i by circular arc
(2.3.2)	Else, for each vertex u_i of r_i repeat
(2.3.3)	If u_i has vertical and horizontal neighbors, blend u_i
(3)	Output valid solids; STOP

This algorithm identifies the surfaces of the solid that can be curved. At present, it replaces with circular arcs only. However, extension to include all conics is straightforward; for example by the use of Liming's method[11]. The inputs to the algorithm are the three connectivity matrices that correspond to the three orthogonal views of the solid. It is required that these views consist entirely of straight-line segments only (i.e. it is assumed that projections of the curved surfaces are straight lines in the top and front views). The algorithm then determines the edges in the right view that can be curved. The limits on the parameter values of the circular arcs can be adjusted such that no two curves overlap or intersect, and the solid remains valid.

The procedures used by the curves algorithm are briefly described below:

- *Get-vertical edges (edge-list)*: This generates a list of edges that correspond to the vertical edges in the top and front views. These edges project as either straight lines or points in both these views. The list is returned in *edge-list*.
- *Horizontal (edge-list, i)*: This determines whether the *i*th edge in the *edge-list* is horizontal. It returns TRUE if the edge is horizontal. Otherwise, it returns FALSE.
- *Mark-parametric-curve (edge-list, i)*: This records that the *i*th edge in the *edge-list* can be replaced by a family of circular arcs that can be represented in the parametric form.
- *Get-discrete-points (right-view, point-list)*: This identifies all the points in the *right-view* that have their projections as horizontal lines in both the front and top views. The list of resulting points is returned in *point-list*.
- *Valid-discrete-curve (edge-list, i, point-list, j)*: This determines whether edge *i* can be replaced by a circular arc that spans up to the point *j*. It returns TRUE if the edge can be replaced.
- *Mark-discrete-curve (edge-list, i, point-list, j)*: This records that edge *i* in the *edge-list* can be replaced by a circular arc that spans up to point *j* in the *point-list*.
- *Get-tangent-nodes (right-view, edge-list)*: This identifies all the nodes in the *right-view* that can be removed and replaced by a set of parametric arcs that are tangent to the edges that join at that node. The result is returned in *node-list*.
- *Mark-tangent-node (node-list, i)*: This records that node *i* in the *node-list* can be removed and replaced by a set of parametric tangent curves.

EXAMPLES

The workings of the algorithm are illustrated by three examples. The algorithm has been implemented in C (and run on an Apollo DN 4000 workstation). The first example solves for polyhedral solids only, while the second and third generate solids that are bounded by quadrics.

Polyhedral solids

The polyhedral-solids example has been borrowed from Reference 9, in which ten solids were generated that corresponded to the pair of orthographic views shown in Figure 2. With the algorithm, the complete set of solids (16) and the corresponding right-side views were generated. They are shown in Figure 6. On the Apollo DN 4000, it took 2.6 s to generate all the right-view connectivity matrices (from which the right views were drawn).

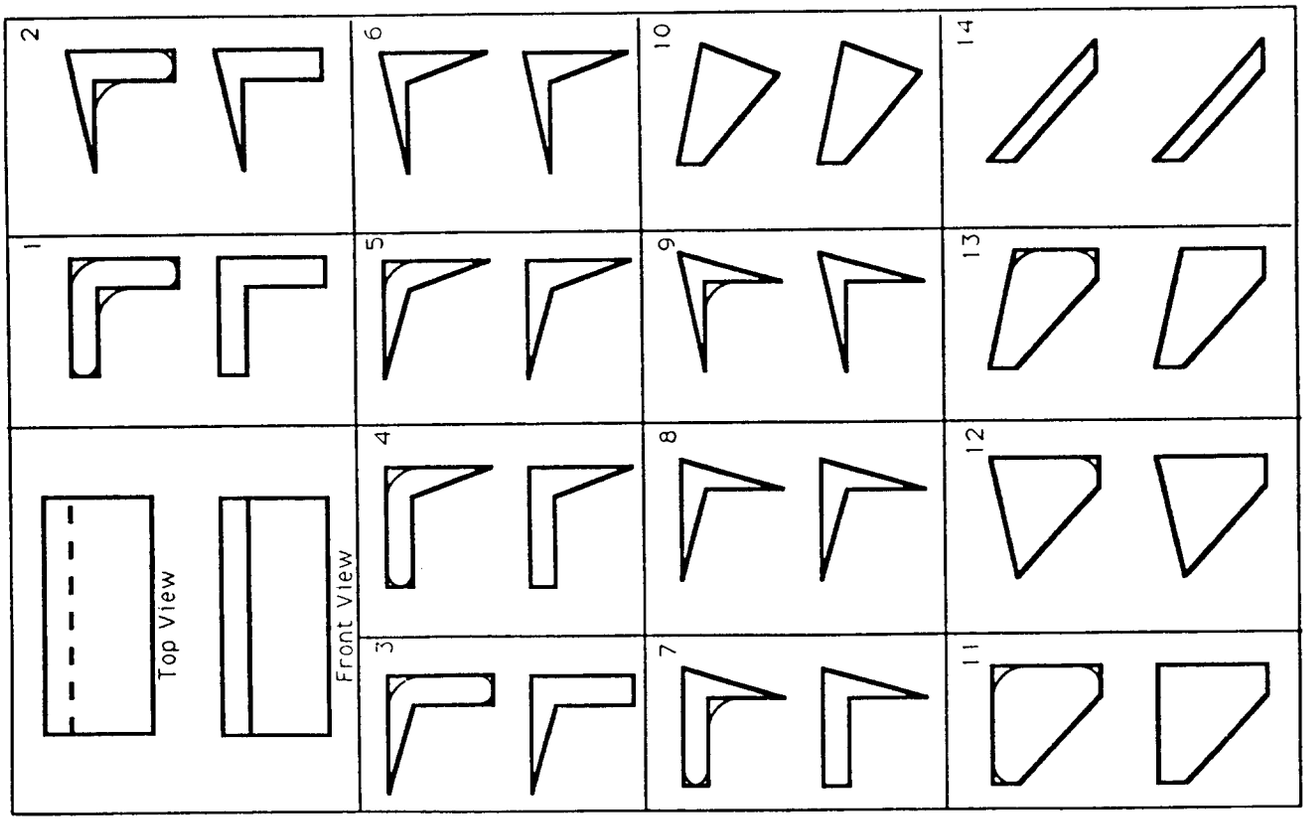


Figure 8. *Second set of curved solids*

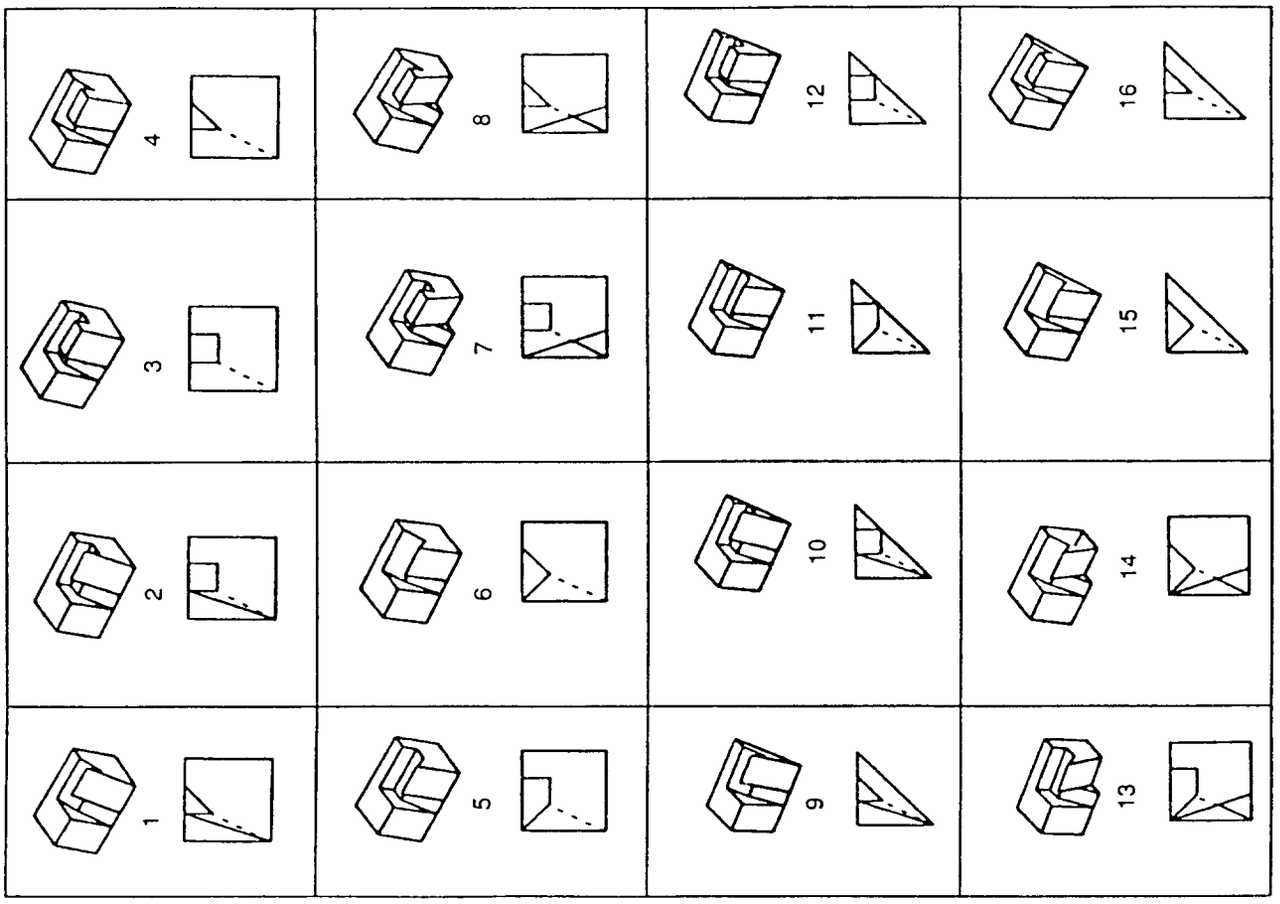


Figure 6. *Polyhedral solids*

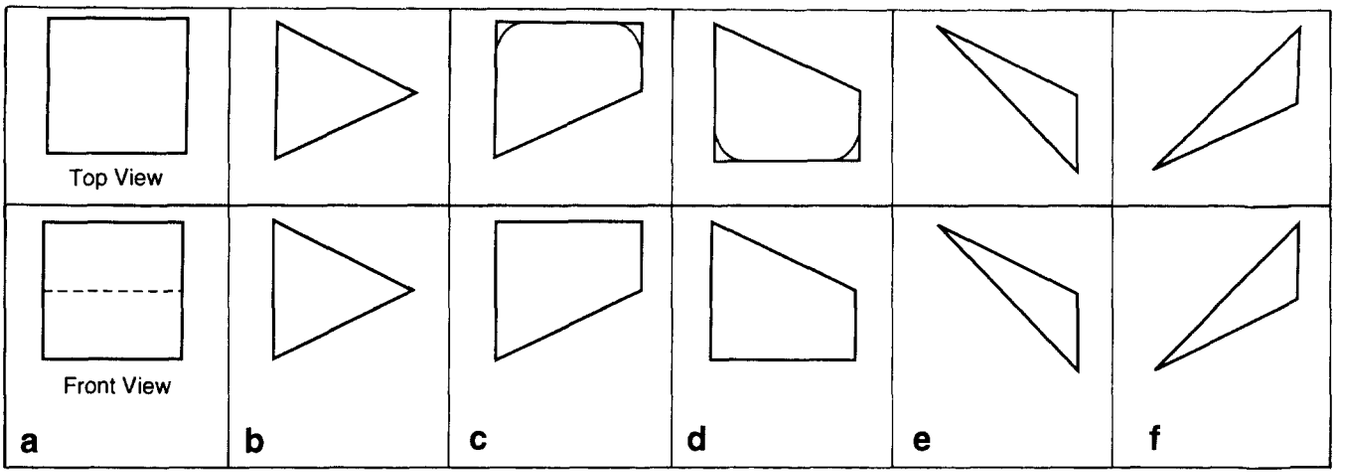


Figure 7. *First set of curved solids*

For the solution of the above problem, the assumption was made that every solid has a planar (rectangular) base, the boundary of which corresponds to the boundary (i.e. the outer edge loop) in Figure 2a. This assumption disallows any prismatic protrusions below the base. When this restriction is removed (i.e. prismatic protrusions below the base that do not affect other views are allowed), the number of solutions increases rapidly. The reader can visualize the new solids when, for example, triangular prisms are attached to the bottom face of solutions 9, 10, 11, 12, 15 and 16 such that the top and front views are unaffected.

Solids bounded by curved surfaces

The reconstruction of curved solids from a pair of orthographic views (straight lines only) is illustrated. The top and front views have intentionally been chosen to be simple. They permit the demonstration of the working of the algorithm, and, most importantly, they show the large number of possible solutions. In Figures 7 and 8, instead of the complex curved solids being drawn, the two right views for each case, i.e. a polygonal right view and a right view with circular arcs, are illustrated. Note that many straight lines in the right views can further be replaced by curves. For example, the inclined lines l_1 in Figure 7b can be replaced by any one curve from the family of conics that are tangent to l_2 and l_3 at the vertices a and b, as long as their projection does not create an additional hidden line in the front view. Such valid modifications can lead to a large number of geometrically distinct solids. Such geometric variations are not shown in Figures 7 and 8.

In Figure 7, the top and front views are given. The corresponding five right views are also shown. This simple example took less than 1 s for each polygonal right view and curved right view on the Apollo DN 4000. Note that not all views permit curves. For example, the replacement of the sharp corner in Figure 7b by any circular curve is not permissible if consistency with the dimensions in the top and front views is to be maintained. Figures 7e and f are similar.

In Figure 8, the top and front views are given. For this set, the algorithm identified 14 polygonal right views in 3.2 s, and the corresponding curved right views in 2.1 s, on the Apollo DN 4000. As before, not all the right views permit curves. It is interesting to note that, for the given top and front views, one typically visualizes an L section or some variation thereof; views 10–14 do not come to mind easily. However, large crosssectional areas are contained in views 10, 11, 12 and 13, and the smallest in view 14. The algorithm can readily be adapted for the automatic detection (or generation) of solids with such geometric properties.

User interaction, as in the case of polyhedral solids (Figure 6), is helpful in the reconstruction process. The search process can then be guided to include, or exclude, certain features on the solid. Also, in the curved solids (see Figures 7 and 8), the radius for rounding and filleting can be chosen by the user. Further, with the use of Liming's method, the circular arcs can easily be replaced by other conics such as elliptical, parabolic or hyperbolic arcs (see, for example, Reference 11).

SUMMARY

An algorithm has been presented that reconstructs all solids bounded by planes and quadrics that correspond to a pair of prespecified orthogonal views. As a byproduct, the missing third views are also obtained. The algorithm ensures completeness by considering all the candidate faces for inclusion in the solid. Consistency with the given views guarantees the validity of the generated solids and the correctness of the algorithm.

Visualization in three dimensions is a difficult task. For complex solids, even when all three views are provided, a manual reconstruction process can be very complicated. It requires the ascertaining of correspondence between the given views, and then the construction of the solid based on the views. The reconstruction problem considered in this paper is more difficult, because of the combinatorial nature of the solution brought about by the incomplete specification. The solution procedure can benefit extensively from user interaction (as in Figure 4). As a 2-view representation typically corresponds to multiple solids, the search process can be guided to exclude classes of solids that the user deems unnecessary.

ACKNOWLEDGEMENTS

The authors thank Professor Doug Wilde of Stanford University, USA, for suggesting this problem for further study. Professor Dutta was supported in part by US National Science Foundation Grant DDM 90-10411. Y L Srinivas was supported by the University of Michigan, USA, Rackham Award, and by NSF Grant DDM 90-10411.

REFERENCES

1 **Sugihara, K** *Machine Interpretation of Line Drawings* MIT Press (1985)

2 **Idesawa, M, Soma, T, Goto, E and Shibata, S** 'Automatic input of line drawings and generation of solid figure from three-view data' *Proc. Int. Computer Symp. – Vol II* (1975) pp 304–311

3 **Lafue, G** 'Recognition of three-dimensional objects from orthographic views' *Comput. Graph.* Vol 10 No 2 (1976)

4 **Markowsky, G and Wesley, M A** 'Fleshing out wire frames' *IBM J. Res. & Develop.* Vol 24 No 5 (1980) pp 582–597

5 **Markowsky, G and Wesley, M A** 'Fleshing out projections' *IBM J. Res. & Develop.* Vol 24 No 6 (1981) pp 934–954

6 **Preiss, K** 'Algorithms for automatic conversion of a 3-view drawing of a plane-faced part of the 3-D representation' *Comput. Industry* Vol 2 (1981) pp 133–139

7 **Aldefeld, B** 'Automatic 3-D reconstruction from 2-D geometric part descriptions' *Proc. IEEE Conf. Computer Vision & Pattern Recognition* (1983) pp 66–72

8 **Sakurai, H and Gossard, D C** 'Solid model input through orthographic views' *Comput. Graph.* Vol 17 No 3 (1983) 243–247

9 **Wilde, D J** 'The geometry of spatial visualization: two problems' *Proc. 8th IFTOM World Congr.* Prague, Czechoslovakia (Aug 1991)

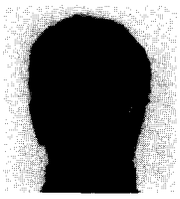
10 **Luzadder, W J** *Fundamentals of Engineering Drawing* (9th Ed.) Prentice–Hall (1986)

11 **Faux, I D and Pratt, M J** *Computational Geometry for Design and Manufacture* Ellis Horwood, UK (1979) p 31

BIBLIOGRAPHY

Harlick, R M and Shapiro, L G 'Understanding engineering drawings' *Comput. Graph. & Image Proc.* Vol 20 (1982) pp 244–258

Debasish Dutta is an assistant professor in the Design Laboratory, Department of Mechanical Engineering and Applied Mechanics, University of Michigan, USA. He obtained a PhD from Purdue University, USA, in 1989. His areas of interest include geometric modeling, computational geometry in design and manufacturing automation, and descriptive geometry.



Lakshmi Srinivas obtained his bachelor's degree in mechanical engineering from the Indian Institute of Technology, Madras, India, in 1989, and his master's degree in mechanical engineering from the University of Toledo in 1990. He is currently a doctoral student in the Design Laboratory, Dept. of Mechanical Engineering and Applied Mechanics, University of Michigan, USA. His research interests include geometric/solid modeling, motion planning and the blending of surfaces.