Technical Report # UM-MEAM-91-10

Reconstructing Curved Solids From Two Orthographic Views

Debasish Dutta*
and
Y. L. Srinivas†

Design Laboratory
Department of Mechanical Engineering
The University of Michigan
Ann Arbor, MI 48109

e1qn
UMR0905

Reconstructing Curved Solids From Two Orthographic Views

Debasish Dutta*

and

Y. L. Srinivas†

Design Laboratory
Department of Mechanical Engineering
The University of Michigan
Ann Arbor, MI 48109

July 24, 1991

Abstract

It has long been known that the two-view orthographic representation of a mechanical part is ambiguous. Existing literature addresses recognition/reconstruction of solids, mostly polyhedral, from either one-view (perspective) drawings, or from three orthographic views. In this paper, we present algorithms for reconstructing curved solids from only two orthographic views. First, we reconstruct all polyhedral solids that correspond to the pair of prespecified orthogonal views and then develop the associated third views. For each solid, we analyze the three views simultaneously and 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. Application areas include computer-aided design, machine vision and automated inspection systems.

*Supported in part by NSF Grant DDM 90-10411

†Supported by the University of Michigan Rackham Award and by NSF Grant DDM 90-10411

1 Introduction

1.1 Two-view Orthographic Representation

The use of parallel projections for unambiguous representations of solid objects was developed by the French mathematician Gaspard Monge, in mid-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 three dimensional 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 enclosed in a transparent box) the ones most commonly used in engineering are the *front, top* and the *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 require sectional and/or auxiliary views in addition to the three views.

In this paper we are interested in the inverse problem, that of developing solids whose orthographic views are given. In particular, we consider the problem where only two of the three views are given. 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) will, in general, correspond to more than one solid, each giving rise to a unique third view (i.e., the right-side view). Thus, the two-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 non-planar 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 corresponding to the two given views. The third view can then be determined. Therefore, vizualization in 3-dimensions is requisite, and that in itself is cause for frustration. Furthermore, since a two-view representation is non-unique 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 adding consistent right side views, one by one. However, if the given views

are complex, powers of spatial visualization become inadequate in guiding 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.

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

Akin to the gift wrapping method of generating convex hulls, our reconstruction procedure 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 valid configurations of faces and determines the complete set of valid solids corresponding to the given views. An empty solution set proves that the given views are inconsistent.

We conclude this section by reviewing prior work in generating solids from line drawings, and mentioning potential applications. In section 2, we analyze the problem and discuss the algorithm. Sections 3 and 4 contain the algorithms for polyherdal and curved solid generation. Examples are shown in Section 5.

1.2 Previous Work

Motivated by the use of digital computer for image processing, various aspects of automatic recognition and reconstruction of line drawings have been researched since the 1960s. In particular, two variants of our problem, viz., reconstruction from *single-view* and *three-view* drawings, have received attention thus far.

An excellent survey of the research on machine interpretation of line drawings can be found in the first few pages of the text by Sugihara [Sugihara 1986]. In his book, Sugihara addresses, in particular, the recognition of polyhedral objects and scenes described by single-view line drawings (also referred to as perspective drawings). An important subclass of this problem is the interpretation of multi-view line drawings.

Idesawa presented a method to reconstruct solid models from three orthographic projections [Idesawa 1973]. The method, however, was capable of producing invalid objects. Lafue developed heuristics to detect such invalid objects by imposing a prespecified input format [Lafue 1976]. In 1980, Wesley and Markowsky developed an algorithm for producing orthographic views from wire-

frame models of mechanical parts [Wesley and Markowski 1980]. They further extended the method to construct polyhedral solids from the three orthographic projections [Wesley and Markowsky, 1981]. Methods for extracting 3-D information from orthographic views can also be found in [Preiss 1981] and [Aldefeld 1983]. In [Sakurai and Gossard 1983], the Wesley and Markowski 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, to the best of our knowledge, was first examined in [Wilde 1991]. Wilde examined the difficulties involved in spatial visualization, and in particular, in the reconstruction of solids from two orthogonal views. In this paper, we present algorithms for the reconstruction of polyhedral and curved solids from two views.

1.3 Applications

An automatic procedure to generate solids from line drawings has many applications in engineeering. 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 automatically convert engineering drawings into solid models can be very useful. The conversion of engineering drawings into solid models is a task many industries have to address in the process of adopting solid modelling systems. Recognizing orthographic projections will also have applications in machine vision and automated inspection.

2 Analysis of the Reconstruction Problem

2.1 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 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 edges and they appear in their *true lengths*. We refer to that orientation of an object which maximizes the number of egdes appearing in true

lengths in all three views, 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 tetrahedra).

2.2 Edge and Vertex Multiplicities

When three orthographic views are provided, there is enough information in the drawings to deduce which lines appear in true lengths. Reconstructing 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. We consider such lines 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.

Detection of multiple edges is enabled by reasoning about multiple vertices, since each edge must correspond to a pair of vertices. Therefore, multiplicity m of vertex v refers to the *maximum* number of distinct vertices it can conceal, consistent with the views. The multiplicity of vertex v in the top/front can be obtained by counting the vertices in the front/top view that lie on the vertical line through v. The following is intuitive.

Proposition 2.1
If vertices with multiplicities > 1 exist in two orthographic views, the pair corresponds to more than one valid solid.

2.3 A Combinatorial Analysis

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

A bound on the number of right views (and hence distinct polyhedral solids) corresponding to a pair of orthographic views can be derived by analyzing the standard technique employed in engineering drawing to develop third views from two given views. We illustrate the procedure in Figure 1 but omit discussions. An explanation can be found in any standard text on engineering drawing, e.g.

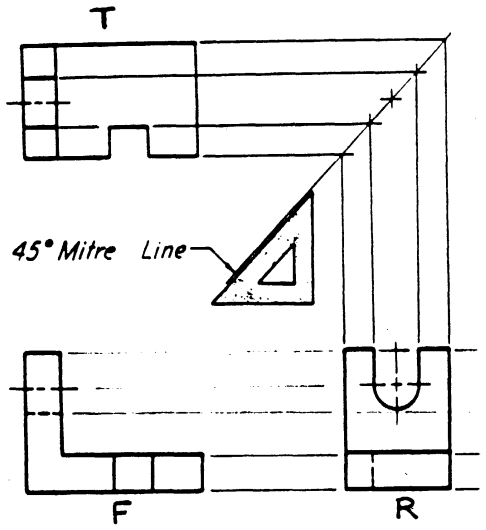

Figure 1: Constructing a right view from top and front views

[Luzadder 1986].

In Figure 1, let us assume that the top and front views, shown in the top quadrant (TQ) and front quadrant (FQ) respectively, were generated by analyzing a solid S with m vertices and n edges. By construction, all candidate vertices in the right quadrant (RQ) are the intersections of the horizontal and vertical lines drawn from the vertices in FQ and TQ, respectively. There are $O(m)$ vertices in TQ and in FQ. So RQ has $O(m^2)$ candidate vertices. But 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: Firm lines in the right view cannot intersect but at the candidate vertices. However, intersections between firm and hidden lines can create new vertices.) Letting $M = m^2$, there are C_m^M possibilities for the vertex set V.

Let us consider a vertex set V_k of cardinality m. With m vertices, we can have $O(m!)$ trees. [1] But, the orthographic views cannot contain any trees since trees correspond to dangling faces and edges (i.e., unbounded solids). Therefore, the set of valid right views P, corresponding to vertex set V_k, consists of all graphs but no trees or dangling edges. For example, every triangulation of V_k is a topologically valid right view since it contains cycles. It is easy to see that the cardinality of the set P will, in general, be exponential in m. Since 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 since consistency with the top and front views is the final check for a valid right view. Moreover, inclusion of non-polyhedral solids in the above analysis will result in an infinite number of valid right views.

[1] For example, with three vertices A, B and C we can have trees like A-B-C, A-C-B, B-A-C and so on.

2.4 Correctness of Procedure

We begin by asserting that a two-view representation can be unique under special conditions.

Lemma 2.2
If at least one of the two given orthographic views has no multiple vertices, then the two-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. □

Note, that a pair of orthographic views, as per Lemma 2.2 will, in general, not facilitate visualization. However, for the purposes of our reconstruction algorithms they are valid inputs. If the multiplicity of a vertex is n, it could conceal upto $n-1$ vertices in the view. Therefore, we account for such a vertex in the construction procedure by considering the view in which the vertex appears n times, once with each value of the vertex multiplicity (i.e., $0, 1, .., n-1$).

Theorem 2.3
In the reconstruction of all valid solids from a pair of orthographic views, it is necessary and sufficient to analyze only one view and account for all vertices in that view that have multiplicity > 1.

Proof
That it is necessary follows from Proposition 2.1: we argue for 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 and accounting procedure is exhaustive. Sufficiency is now an immediate consequence of Lemma 2.2. □

Any procedure that does not account for all vertices with mutiplicity > 1 is, by Theorem 2.3 incomplete. We account for multiple vertices in one view by directly constructing the solid while using the other view for verifying consistency. The algorithm is based on an enumeration scheme. It considers all possible configuration that can arise from the two given views. From these candidate configurations, only those which correspond to valid representations of solids are identified and stored. Since all candidate configurations are enumerated, the algorithm insures that the solution set is complete, i.e., there are no solids other than those in the solution set that will generate the given pair of orthographic views.

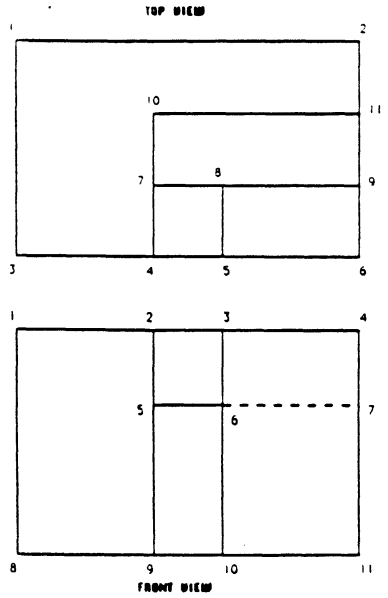

Figure 2: Orthogonal views

3 Overview of Algorithm

3.1 Algorithm Input

The input to the algorithm are two vertex connectivity matrices that correspond to the given views. Coordinates of each vertex are also supplied to the algorithm. Each connectivity matrix is a $(n \times n)$ square matrix, where n is the number of distinct vertices in the corresponding view. Figure 2 shows a pair of orthographic views, and the associated connectivity matrices are shown in Figure 3.

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 2 (Front View), vertex 5 is connected to vertex 7

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

Top View

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

Front View

Figure 3: Connectivity Matrices

by a partially hidden line and so $C(5,7) = 2$. The connectivity matrices are symmetric.

3.2 Description of Algorithm

The algorithm works as follows. Since 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 2 (front view) has a multiplicity of 4 since, there are at most four vertices in the top view, viz., 6, 9, 11 and 2, whose projections can coincide with it.

Next, the vertex with least multiplicity, say v_1, is selected. From the connectivity matrices, all neighbors of v_1 are identified and stored in V. Every non-collinear triple in V containing v_1 defines a plane passing through v_1. These planes are generated and stored in P. Finally, a set of faces F is created by systematically detecting all vertices, except v_1 and its neighbors, that lie on each element of P. Therefore, each element of F is a face containing the vertex v_1.

Next, we analyze the faces in F. In particular, we need to determine how many faces of F can lead to a valid solid. A solid is considered valid only if it corresponds to the given views. The analysis involves expanding faces of F and 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. [2] 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} ..... G_1 (i.e., excludes non-manifolds). Furthermore, consistency with the given orthographic views is verified prior to adding a face in G_i.

A depth first strategy is employed 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., bounded solid) and validity (i.e., consistency with the given orthographic projections). Any solid that passes this check is a vaild solid and is stored in the solution set. The connectivity matrix corresponding to the right view is then generated. The face expansion process is carried out for all sets G_i, until all faces are expanded. The algorithm then terminates, since all solids corresponding to the given pair of orthographic views have been identified and their right view connectivity matrix stored.

Now we have the three views for each solid. The last step is to identify from these views, which lines in the right view can be replaced by circular arcs. (Note, we are only concerned with introducing curves in the right view and not in the

[2] We do not consider non-manifold objects

top and front views which were prespecified). We assume that the solid being analyzed is in a canonical orientation. Once again, we analyze one view and use the other for checking consistency. Clearly, all vertical lines in the top view are candidates for curved edges,when viewed from the right. Furthermore, 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 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 done to ensure consistency with the front view and the corner/edge in the solid is rounded, or filleted, by replacing the appropriate vertex in the right view with a circular arc. The blend radius can be chosen by the user and determines the point of tangencies.

4 Algorithm

The main algorithm invokes three substeps, viz., Form-Face, Expand-Face and Curves. The first two are recursive.

4.1 Algorithm MAIN

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 v_1, **while** *multiplicity* > 0
    F $\Leftarrow$ Form-Face (v_i)
    while F not empty, Expand-Face(F)
    P $\Leftarrow$ polyhedral solids; Q $\Leftarrow$ right views
    while Q not empty, Curves(Q)
        C $\Leftarrow$ Curved solids

output P and C

end

The algorithm MAIN requires that in the orthographic views there be a path from each vertex to every other vertex (i.e., no edge loop in is properly contained in another). This condition disqualifies solids that have holes and pockets that

open on planes. The algorithm can be modified to overcome this limitation by adding dummy edges between the appropriate vertices of the disconnected loops.

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

1. **Input-connectivity-matrices** *(top-view, front-view)* :
Reads the data for both views from the input file and retruns the data in matrices *top-view* and *front-view.*

2. **Generate-multiple-nodes** *(top-view, front-view, multiple-node-list)* :
Generates all the possible multiple nodes in the front view from the information given in *top-view* and *front-view* matrices and returns the list of multiple nodes, properly labelled, in *multiple-node-list.*

3. **Get-min-mnode** *(multiple-node-list, min-loc)* :
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.*

4. **Get-mnodes** *(min-loc, multiple-node-list, mnode-list):*
Extracts all the mnodes at node min-loc in the front view from the multiple-node-list and returns the resultant list in mnode-list.

5. **Get-connected nodes** *(mnode, connected-list):*
Determines all the mnodes in the *multiple-node-list* that are connected to *mnode* and returns the resultant set in *connected-list.*

6. **Get-next-plane** *(mnode, connected-list, face):*
Determines the next candidate plane in the *connected-list* that passes through *mnode.* The previous plane, if any, is passed via *face.* The resultant plane is returned in *face.*

7. **Get-nodes-in-plane** *(face, vertices):*
Determines all *mnodes* that lie in the plane defined by *face* and returns this mnodes list in *vertices.*

8. **Initial-solid** *(solid, face):*
Sets up the initial *solid* with a single face defined by *face.*

9. **Output-solution()**:
Outputs the solution set of solids.

4.2 Algorithm Form-Face:

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

Algorithm:

1. Identify all neighbors of 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 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 plane
   - 2.4 Invoke form-face( ) to generate the possible faces for updated face
   - 2.5 Remove this vertex (the last vertex) from face
   - 2.6 Mark vertex as FREE in the list of vertices that lie in the plane
3. End of preocedure: Return

The procedures used by Form-Face are briefly described below.

1. **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*.

2. **Add-vertex-to-face** *(face, vertex):*
This procedure expands the partially constructed *face* by adding *vertex* to the last vertex of face and updating *vertex* as the last vertex of *face*.

3. **Valid-face** *(face):*
This procedure verifies if a given *face* is a complete and consistent face, i.e. edges form a closed loop and do not cross each other.

4. **Add-face** *(face):*
This procedure stores *face* in the list of candidate valid faces.

5. **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.

6. **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.

4.3 Algorithm Expand-Face

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.

Algorithm:

1. For each edges 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 Expand-Face are briefly described below.

1. **Number-of-edges** *(face):*
Determines the number of edges in *face*.

2. **Extract-edge** *(face, i, edge):*
Extracts the edge *i* from *face*. This edge is returned in the variable *edge*.

3. **Already-shared** *(edge, solid):*
Checks if *edge* is already shared by two of the faces of the partially constructed *solid*. If so, if returns TRUE; else returns FALSE.

4. **Get-next-face-list** *(edge, face, solid, face-list):*
Gets the list of all other possible faces for the *edge* which 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*.

5. **Add-face-to-solid** *(solid, face):*
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*.

6. **Valid-solid** *(solid):*
Checks if *solid* represents a valid solid that can be a member of solution set. It checks (i) if all the edges are shared by exactly two nodes and (ii) the solid produces all the vertices and lines in the given view information. If *solid* passes

these checks, it returns TRUE, else returns FALSE.

7. **Store-solid** *(solid):*
Adds *solid* as a member of the solution set.

8. **Remove-face-from-solid** *(solid, face):*
Removes the face *face* from a partially constructed solid *solid*. It assumes that the resulting partial solid is still valid (i.e., does not contain discontinuities)

4.4 Algorithm Curves

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

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 neighbours, blend u_i
3. Output valid solids; STOP

This algorithm identifies the surfaces of the solid which can be curved. At present it replaces with circular arcs only. However, extension to include all conics is straightforward; e.g., using Liming's method [Faux and Pratt 1987, page 31]. The input to the algorithm are the three connectivity matrices corresponding to the three orthogonal views of the solid. It is required that these views consist entirely of straight line segments only (i.e., we assume that projections of the curved surfaces are stright 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 Curves are briefly described below.

1. **Get-vertical-edges** *(edge-list):*
Generates a list of edges that correspond to the vertical edges in top and front views. These edge project as either straight lines of points in both these views.

The list is returned in *edge-list.*

2. **Horizontal** *(edge-list, i):*
Determines if *i th* edge in the *edge-list* is horizontal. It returns TRUE if the edge is horizontal. Otherwise, it returns FALSE.

3. **Mark-parametric-curve** *(edge-list, i):*
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.

4. **Get-descrete-points** *(right-view, point-list):*
Identifies all the points in the *right-view* that have their projections as horizontal lines in both front and top views. The list of resultant points is returned in *point-list.*

5. **Valid-descrete-curve** *(edge-list, i, point-list, j):*
Determines if edge *i* can be replaced by a circular arc that spans upto the point *j*. Returns TRUE if the edge can be replaced.

6. **Mark-descrete-curve** *(edge-list, i, point-list, j):*
Records that edge *i* in the *edge-list* can be replaced by a circular arc that spans upto point *j* in the *point-list.*

7. **Get-tangent-nodes** *(right-view, edge-list):*
Identifies all the nodes in the *right-view* that can be removed and replaced by a set of parametric arc that are tangent to the edges that joint at that node. The result is returned in *node-list.*

8. **Mark-tangent-node** *(node-list, i):*
Records that node *i* in the *node-list* can be removed and replaced by a set of parametric tangent curves.

5 Examples

We illustrate the working of our algorithm by three examples. The algorithm was implemented in C and run on an Apollo DN 4000 workstation. The first example solves for polyhedral solids only, while in the second and third we generate solids bounded by quadrics.

5.1 Polyhedral Solids

The first example has been borrowed from [Wilde 1991] where ten solids were generated corresponding to the pair of orthographic views shown in Figure 2. Using our algorithm we generated the complete set of solids (sixteen) and the corresponding right-side views. They are shown in Figure 4. On the Apollo DN

4000, it took 2.6 seconds to generate all right view connectivity matrices (from which the right views were drawn).

While solving the above problem we made the assumption that every solid has a planar (rectangular) base, the boundary of which corresponds to the boundary (i.e., the outer edge loop) on the top view in Figure 2. 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 increase rapidly. The reader can visualize the new solids when, for example, triangular prisms are attached to the bottom face of the solutions 9, 10, 11, 12, 15 and 16 such that the top and front views are unaffected.

5.2 Solids bounded by Curved Surfaces

Here we illustrate the reconstruction of curved solids from a pair of orthographic views (straight lines only). The top and front views were intentionally chosen to be simple. They permit us to demonstrate the working of our algorithm, and most importantly show the large number of possible solutions. In Figure 5 and Figure 6, instead of drawing the complex curved solids, we have shown two right views for each case, viz., a polygonal right view and a right view with circular arcs.

In Figure 5, the top and front views are given, as shown. The corresponding five right views are shown in the numbered boxes alongside. This simple example took less than a second each, for polygonal right views and curved right view on the Apollo DN 4000. Note, all views do not permit curves. For example, the replacing sharp corner a by any circular curve is impermissible if one is to remain consistent with the dimensions in the top and front views. Views 4 and 5 are similar.

In Figure 6 the top and front views are given, as shown. For this set, the algorithm identifies 14 polygonal right views in 3.2 seconds, and the corresponding curved right views in 2.1 seconds, on the Apollo DN 4000. As before, not all right views permit curves.

User interaction, such as in Example 1 (Figure 4) is helpful in the reconstruction of the solids. The search process can then be guided to include, or exclude, certain features on the solid. Also, in Examples 2 and 3, the radius for rounding and filleting can be chosen by the user. Furthermore, using Liming's method the circular arcs can easily be replaced by other conics such as elliptical, parabolic or hyperbolic arcs; e.g., see [Faux and Pratt 1979, page 31].

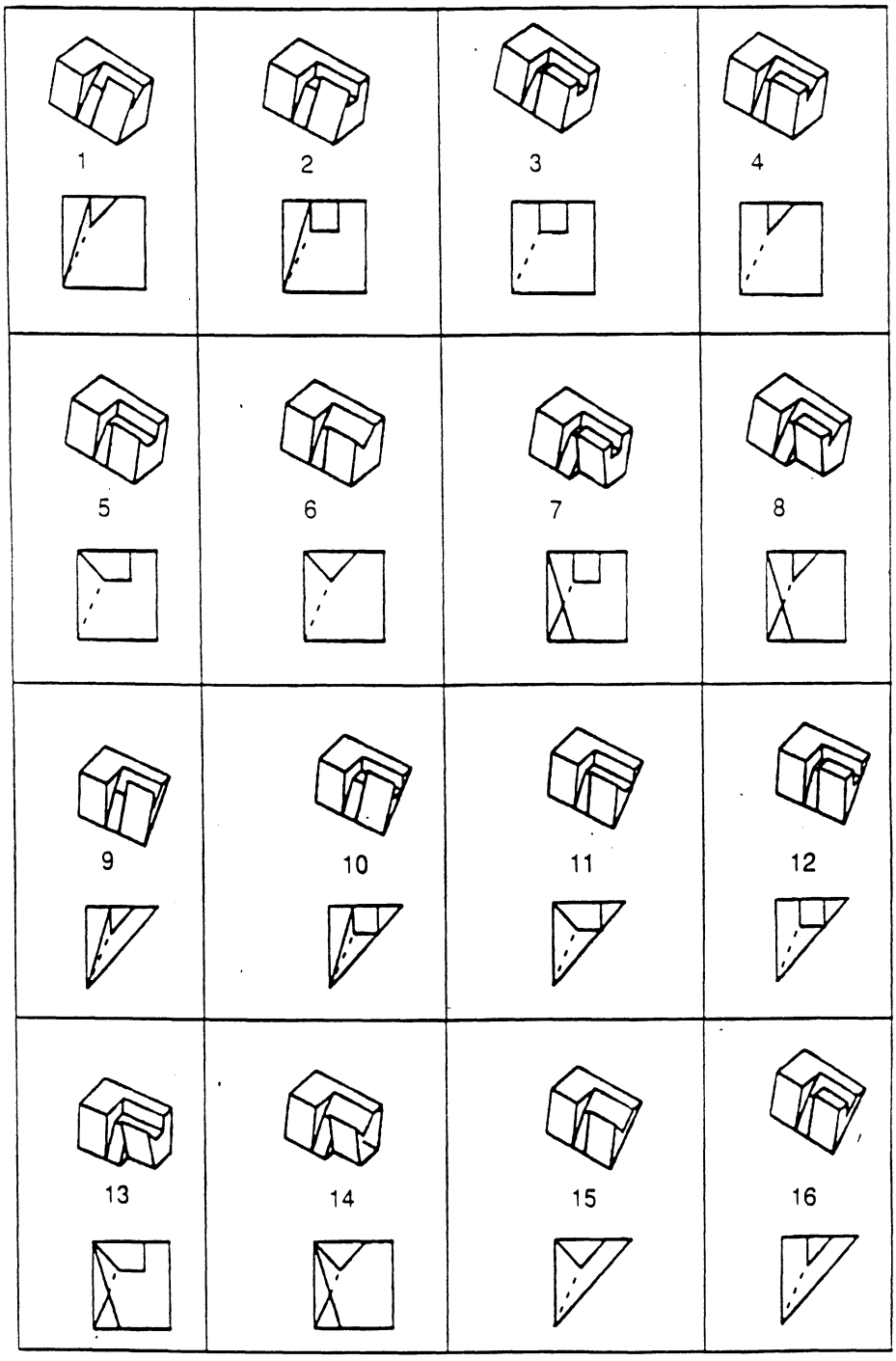

Figure 4: Example 1: Polyhedral solids

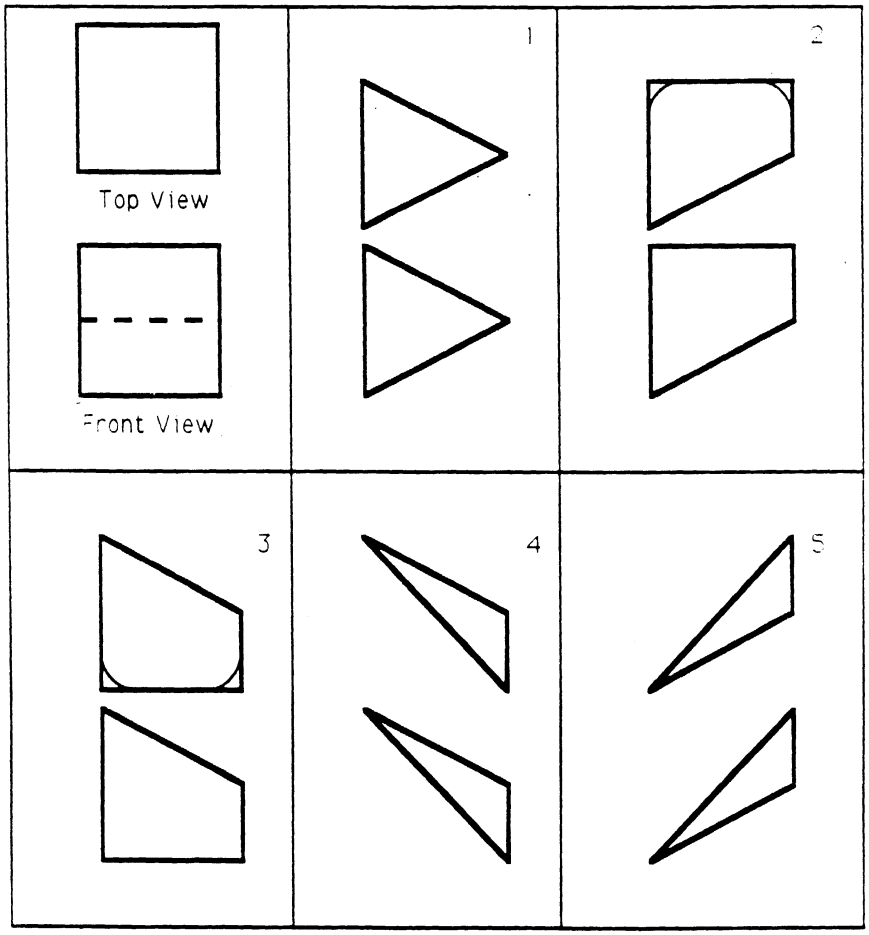

Figure 5: Example 2: Curved Solids

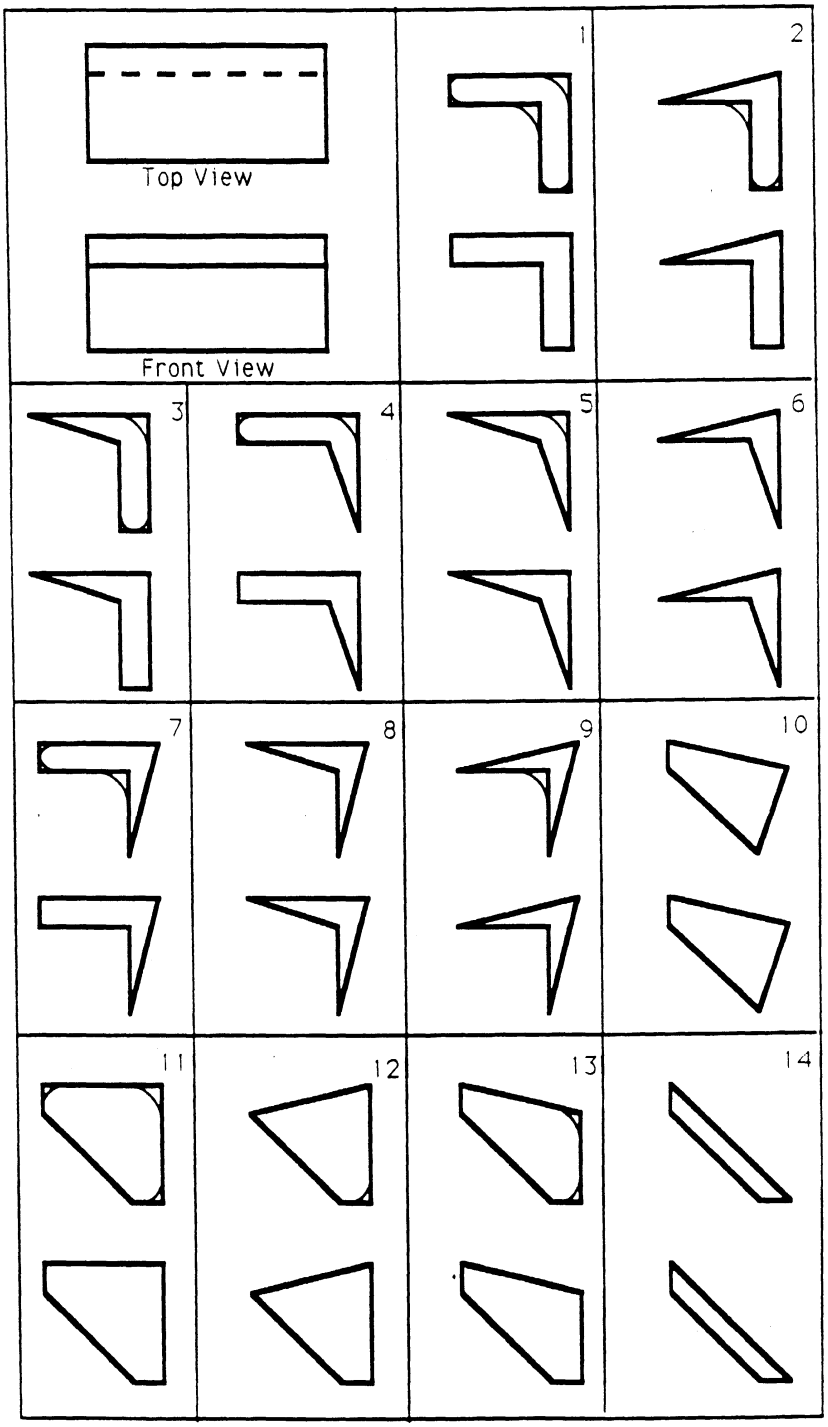

Figure 6: Example 3: Curved Solids

6 Summary

We have presented an algorithm 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 insures completeness by considering all candidate faces for inclusion in the solid. Consistency with the given views guarantees validity of the generated solids and 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 ascertaining correspondence between the given views and then 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 Example 1). Since, a two-view representation typically corresponds to multiple solids, the search process can be guided to exclude classes of solids that the user deems unnecessary.

Acknowledgements

We thank Professor Doug Wilde of Stanford University for suggesting this problem for further study.

7 References

1. Aldefield, B., (1983) "Automatic 3-D reconstruction from 2-D geometric part descriptions," in *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition* 1983, pp.66-72.

2. Faux, I.D. and Pratt, M.J., (1979) *Computational Geometry for Design and Manufacture*, Ellis Horwood Publishers.

3. Harlick, R. M., and Shapiro, L. G., (1982) "Understanding Engineering Drawings" *Computer Graphics and Image Processing* Vol. 20, pp. 244-258

4. Idesawa, M., Soma, T., Goto, E., and Shibata, S., (1975) "Automatic Input of Line Drawings and Generation of Solid Figure from Three-View Data" *Proceedings of International Computer Symposium*, Vol II, pp. 304-311

5. Lafue, G., (1976) "Recognition of Three-Dimensional Objects from Orthographic Views" *Computer Graphics* Vol. 10, No. 2

6. Luzadder, W.J., (1986) *Fundamentals of Engineering Drawing*, Ninth Edition, Prentice Hall

7. Markowsky G., and Wesley, M. A., (1980) "Fleshing out Wire Frames," *IBM Journal of Research and Development* Vol. 24, No. 5, pp. 582-597.

8. Markowsky G., and Wesley, M. A., (1981) "Fleshing out Projections," *IBM Journal of Research and Development* Vol. 24, No. 6, pp. 934-954.

9. Preiss, K., (1981) "Algorithms for automatic conversion of a 3-view drawing of a plane-faced part to the 3-D respresentation," *Computers in Industry* vol. 2, pp. 133-139.

10. Sakurai, H., and Gossard D.C., (1983) "Solid Model Input through Orthographic Views" *Computer Graphics* vol. 17, No. 3, pp. 243-247.

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

12. Wilde, D. J., (1991) "The Geometry of Spatial Visualization: Two Problems," *manuscript*, To be presented at *8th IFTOM World Congress* Prague, August 1991.