Efficient algorithm for the reconstruction of 3D objects from orthographic projections

Qing-Wen Yan, C L Philip Chen* and Zesheng Tang†

An efficient algorithm for reconstructing all polyhedral 3D solid models from 2D orthographic projections is addressed. The algorithm can handle pathological cases and identify all possible correct solutions efficiently. The algorithm has efficient techniques for constructing candidate wireframes, handling broken lines in the input data, generating face loops, determining cutting edges and vertices, building body loops, and assembling body loops into candidate objects. Several symmetric and complex orthographic projections are given to show the completeness of the algorithm.

Keywords: reconstruction, solid models, orthographic projections

In the process of computer-aided mechanical-engineering design and manufacturing, the automatic representation and construction of solid models with computers is a vital step. Three approaches are commonly used to create solid models: (a) primitive modelling, (b) translational and rotational sweeping, and (c) planar projection reconstruction. Primitive modelling directly uses simple 3D objects, including blocks, cylinders and cones. A set of regularized Boolean operations, such as intersection, union and difference, are applied to these to define complex structures. Sweeping modelling is a technique that uses baselines of objects to model a solid object. Sweeping modelling constructs geometric models through either translating the contour of an object along a trajectory, or rotating the cross-section around its centreline. The first two methods have been applied in many commercial software packages. In contrast, projection reconstruction is a 2D-oriented approach that is based on an analogy with conventional engineering drawings. It generates 3D models from a set of 2D planar projections, such as the front view, the top view, the side view, and local views. It is a procedure for recovering information from a low dimension to a high dimension. Although obtaining 2D projections from a given 3D object is straighforward, the inverse operation becomes rather implicit and difficult. The difficulties of this approach arise from the loss of semantics information when a 3D object is represented with 2D projections.

Although the literature on reconstructing 3D information is extensive, few authors have given a complete algorithm and chosen to represent objects formally[1–7]. Usually, the algorithms are unable to handle the full range of pathological cases and ambiguities that occur in practice. Most of the previous methods primarily consist of the following steps:

- Generate candidate 3D vertices from 2D nodes.
- Construct 3D edges from 3D vertices.
- Form the face loops of objects from 3D edges.
- Build components of objects from face loops.
- Combine components to find solutions.

Idesawa's algorithm[1] used the above bottom-up approach, and gave a set of rules for judging pathological edges and vertices in the process of reconstruction. The work dealt with simple objects without pathological cases or multiple-solution cases. Aldefeld[2,3] tried to interpret 3D objects on the basis of pattern recognition, using heuristic searching, instead of the bottom-up method. A 2D projection is considered as a set of 2D elementary graphical elements, such as straight lines, rectangles, arcs and circles. These elements are classified into different clusters according to distinct pattern characteristics. A heuristic search is then used to search for the possible solutions in the space. The algorithm is restricted to uniform-thickness objects, and it may generate one possible solution which may not always recover the entire 3D semantics of objects. Markowsky and Wesley's algorithm[4] also uses the bottom-up approach. The method can handle polyhedral objects, and it allows the

Scanning Management Systems, Canoga Park, CA 91304, USA
*Department of Computer Science and Engineering, Wright State University, Dayton, Ohio 45435, USA
†Department of Computer Science and Technology, Tsinghua University, Beijing, China

Paper received: 22 June 1992. Revised: 2 July 1993

existence of nonmanifold objects, but it cannot handle broken-line (i.e. 'dashed'-line) projections. The approach can deal with multisolution or pathological cases. Sakurai and Gossard[5] extended the work from that[4] of Markowsky and Wesley. They considered objects with cylindrical, conical, and spherical surfaces, and required that the axes of curve surfaces should be parallel to one of the coordinate axes. Gujar[6] proposed an algorithm that can only process polyhedral objects. However, the algorithm cannot reject pathological cases, and it only handles solid-line projections. Gu *et al.*[7] combined the bottom-up and pattern-recognition approaches, and allowed the cylindrical axis to be parallel to one of the coordinate planes. However, no broken-line projections are discussed.

We devise a reconstruction algorithm to find all the polyhedral objects with a given set of 2D orthographic projections. The algorithm uses depth information such as broken (i.e. 'dashed') and solid lines in the projections to reduce the potentially substantial amount of computing of face loops and body loops. It also effectively reduces the possible number of ambiguous objects. The ambiguity is caused by the fact that many objects may have the same set of projections, and the algorithm can recognize all the correct solids from a given set of projections. Our algorithm can deal with the questions of how to identify input data, how to generate the candidate wireframes more effectively, how to take the advantage of the solid/broken-line attributes to accelerate the process of finding the correct solutions, and how to guarantee that all the possible and correct solutions for any given valid input are found.

We solve these questions in the following sections. The next section describes our algorithm. Various examples performed by our algorithm are then given and analysed. Finally, we summarize the algorithm, and suggest future work. In the appendix, we define the geometric concepts used in the paper, such as object, face, loop, face loop, body loop, wireframe, and cutting edge/vertex.

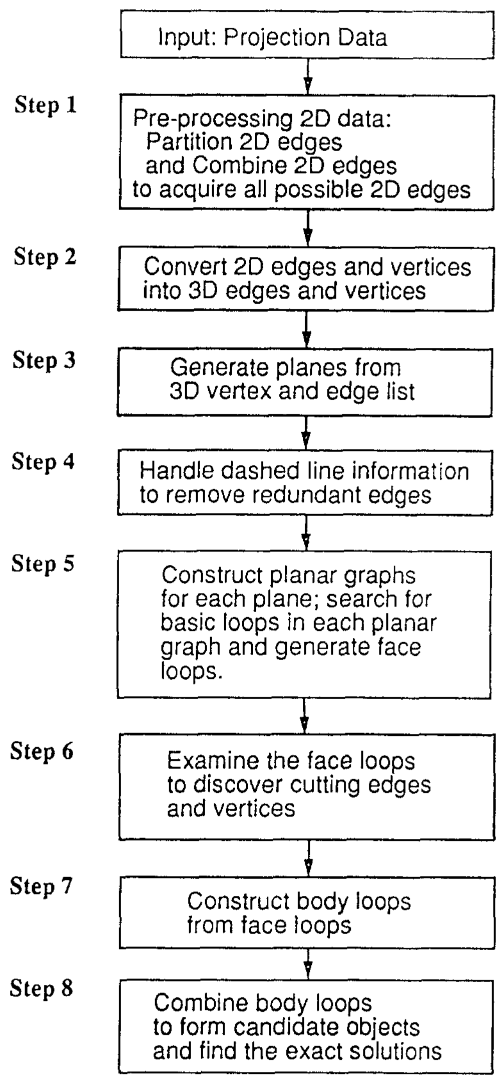

Input: Projection Data
Step 1
Pre-processing 2D data: Partition 2D edges and Combine 2D edges to acquire all possible 2D edges
Step 2
Convert 2D edges and vertices into 3D edges and vertices
Step 3
Generate planes from 3D vertex and edge list
Step 4
Handle dashed line information to remove redundant edges
Step 5
Construct planar graphs for each plane; search for basic loops in each planar graph and generate face loops.
Step 6
Examine the face loops to discover cutting edges and vertices
Step 7
Construct body loops from face loops
Step 8
Combine body loops to form candidate objects and find the exact solutions

Figure 1 Algorithm

RECONSTRUCTION ALGORITHM

The block diagram of our algorithm is shown in *Figure 1*. At each stage, the algorithm generates all the possible information, and eliminates illegal vertices, edges and faces using a set of criteria to accelerate the formation of a correct object. Before an object is constructed, let *p_vertex* and *p_edge* be, respectively, a potential 3D vertex and a potential 3D edge for the construction of the object. The main steps of the algorithm are briefly described below.

(1) Translate the structured 2D vertices and edges into a 2D vertex list and an edge list. The projections are represented by 2D graphical elements such as lines, rectangles, polylines, polygons, and piecewise circles and arcs (see *Figure 2a*).

(2) Construct all the possible 3D vertices and corresponding edges from the 2D vertex list and edge list, and check for their legality (see *Figure 2b*).

(3) Construct all the possible planar faces to generate the face loops, and check for their uniqueness and legality (see *Figure 2c*).

(4) Handle broken-line information to increase the efficiency of computation.

(5) Form face loops to generate the body loops as shown in *Figure 2d*.

(5.1) Find all the planar graphs.
(5.2) Search the loop generators.
(5.3) Determine the relationship between the basic loops, and set up an inclusion-relationship table for each planar graph.
(5.4) Generate the face loops.

(6) Examine the relationship between the face loops if some cutting edges and vertices exist (for a detailed definition of cutting edges and cutting vertices, see the appendix). For example, *Figure 2e* shows the cutting edges e_1 and e_2; *Figure 2f* shows the new face loops formed by cutting face loops F_1–F_4 in *Figure 2d*.

(7) Construct all the body loops, i.e. blocks, to generate objects as shown in *Figure 2g*.

   (7.1) Find the successive face loops of a given face loop.
   (7.2) Establish elementary body loops.
   (7.3) Check the closeness of each body loop.
   (7.4) Classify the body loops.

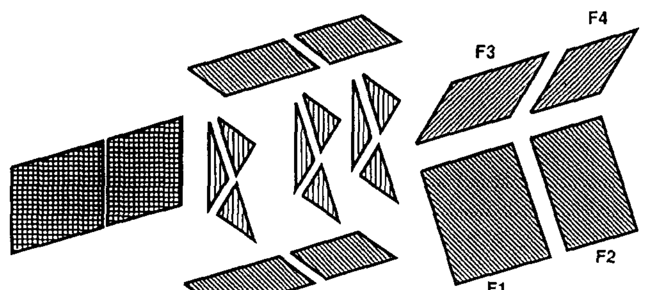
(8) Combine the blocks into objects, and make final decisions as shown in *Figure 2h*.

   (8.1) Rearrange the face loops to form new face loops.
   (8.2) Check the legality of a hypothetical object.
   (8.3) Check the consistency between the generated object and the input planar views of the object.

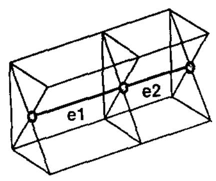
The algorithm is given in detail below.

Construct wireframe

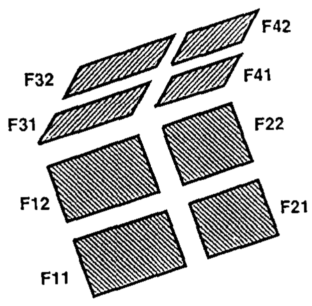
During the reconstruction process, redundant or pathological edges can be generated by the mapping from 2D edges and vertices into 3D edges and vertices. Let *v_list*(●) be a 2D vertex list, in which each vertex consists of a coordinate value (x, y), where (●) can be f, t or s,



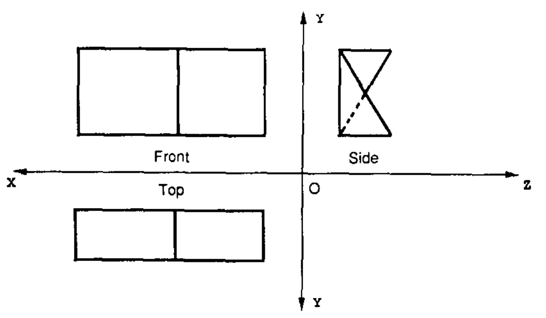
(d)



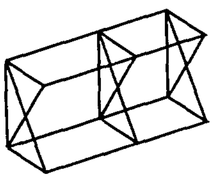
(e)



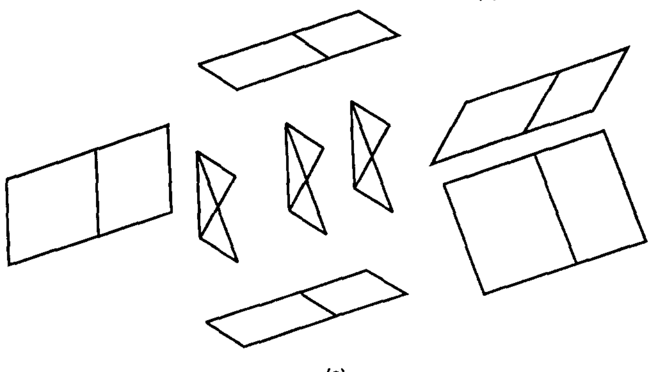
(f)



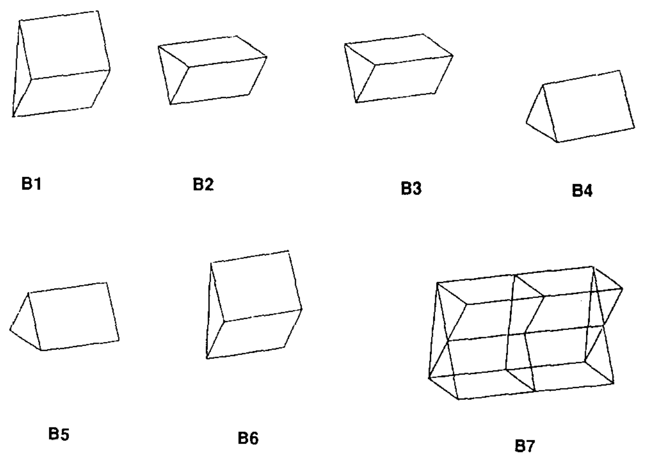
(a)



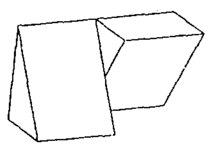
(b)



(c)



(g)



(h)

Figure 2 Example; (a) three views of 2-wedge object, (b) wireframe of object in *Figure 2a*, (c) planar graphs of wireframe in *Figure 2b*, (d) face loops formed from each planar graph, (e) cutting edges and cutting vertices discovered, (f) new face loops created by cutting face loops F_1–F_4 in *Figure 2d*, (g) inner body loops B_1–B_6, and outer body loop B_7, (h) correct object

representing the front, top and side projections, respectively. Let *e_list*(●) be a 2D edge list, in which each edge records two endpoint indices in *v_list*(●), the depth information (broken or solid) of each edge, and the edge type (simple edge or complex edge). In this section, we devise a wireframe-construction algorithm which can remove redundant and pathological edges. The algorithm combines the frontal-projection-based decision-tree search algorithm, the redundant-edge-removal algorithm, and a set of rules to remove redundant and pathological edges.

Previous work[1,3-6] has shown that 3D vertices and edges can be generated in two steps. First, generate 3D vertices by choosing a vertex from each projection's *v_list*(●), and compare the coordinates of the three vertices on their common coordinate axes. The comparison stops if no more 3D vertices are created. Second, generate 3D edges by selecting a pair of vertices (assume that there is an edge between the two vertices). If the three projections of this hypothetical edge are all in their corresponding *e_lists*(●), a *p_edge* is created; otherwise, select another pair of vertices, and repeat this hypothesis-and-verification process until no more 3D edges can be created.

The method described above requires a large amount of computation of edges and vertices comparisons, and it lacks efficiency. We design an efficient approach to the generation of 3D edges and vertices directly from the *v_lists*(●) and *e_lists*(●) in *only one step*, without the separate generation of 3D vertices and edges. The 2D

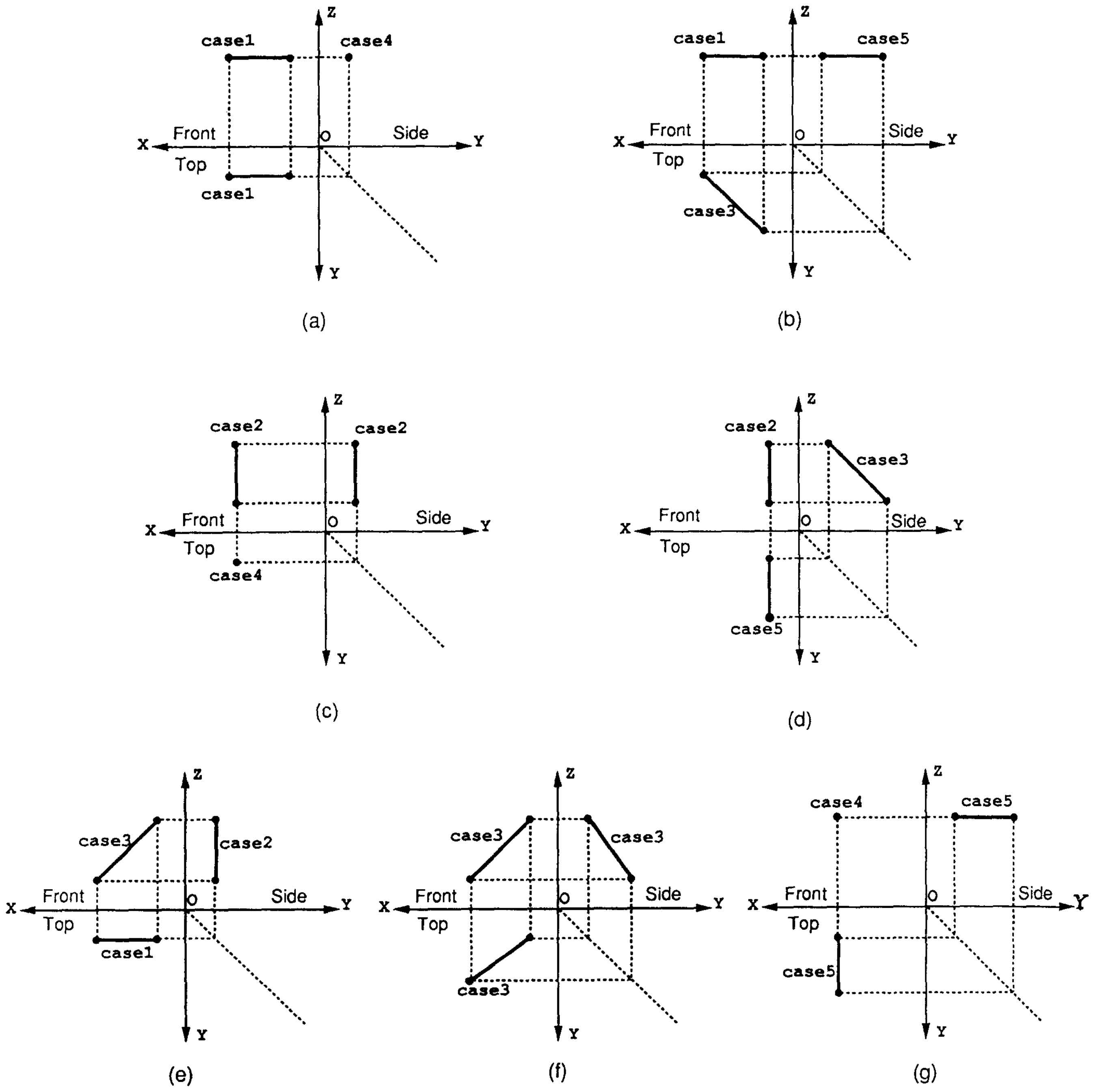


Figure 3 Edge classification based on frontal view

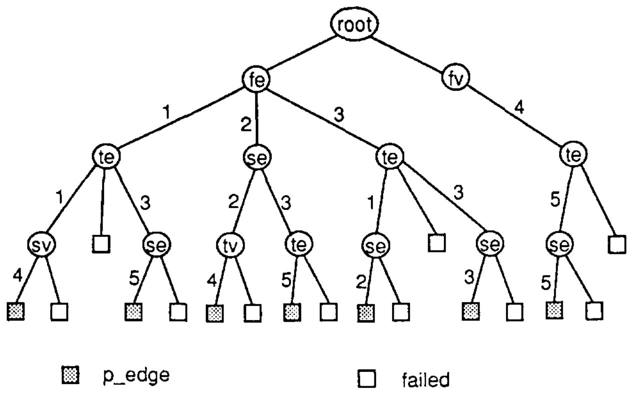


Figure 4 Front-view-based decision tree of objects [*fv*: front-view vertex set, *tv*: top-view vertex set, *sv*: side-view vertex set, *fe*: front-view edge set, *te*: top-view edge set, *se*: side-view edge set. 1, 2, 3, 4 and 5 correspond to the cases with the same numbers.]

edges in *e_list*(●) can be classified into the following five clusters by features:

- *case 1:* parallel to x axis,
- *case 2:* parallel to z axis,
- *case 3:* parallel to none of the coordinate axes,
- *case 4:* a focus point formed by accumulation,
- *case 5:* parallel to y axis.

Figure 3 shows each case for different positions and orientations of an edge. In *Figure 3a,* the frontal and top projections of a *p_edge* belong to case 1, while, in *Figure 3b*, its top projection falls into case 3. The side projections in *Figures 3a* and *b* belong to case 4 and case 5, respectively. All the other projections are shown in *Figures 3c–g.* On the basis of the frontal projection, a decision tree is constructed as shown in *Figure 4* to reveal 3D information about an edge given in *e_list.* The 3D edges and vertices can be generated by traversing this decision tree with the wireframe-construction algorithm, which is described below. The symbols required for this algorithm are as follows:

fe = selected edge from *e_list*(*f*)
te = selected edge from *e_list*(*t*)
se = selected edge from *e_list*(*s*)
fv = selected vertex from *v_list*(*f*)
tv = selected vertex from *v_list*(*t*)
sv = selected vertex from *v_list*(*s*)

In Figure 4, 1, 2, 3, 4 and 5 correspond to the cases discussed above.

Algorithm WC (wireframe construction): Given *v_list*(●) and *e_list*(●), the algorithm generates the wireframe of a solid object.

(W1) [*Traverse the decision tree (as shown in Figure 4) to generate 3D edges and vertices.*] First, select an edge *fe* and trace the decision tree to decide into which case (case 1, case 2 or case 3) the edge falls. Next, select *te* for case 1 or case 3, or select *se* for case 2. Trace down the tree to reach the leaves of the decision tree to see if a *p_edge* can be generated. Finally, select an *fv* and trace the tree to reach node *te*. Then, select a *te* of case 5, if successful, to reach node *se*. At node *se*, we are trying to select an *se* of case 5 to obtain a *p_edge.* If this fails, the traverse backtracks to select a new *te*. When we reach the leaf nodes, we create a *p_vlist* that consists of nonduplicated vertices from each *p_edge*. Similarly, we can create a *p_elist* that consists of nonduplicated *p_edges*. The *p_edge* is represented by a pair of *p_vertices*. The *p_vlist* and *p_elist* contain all the possible 3D vertices and edges. Some of these edges may partially overlap and need to be dealt with in the next step.

(W2) [*Resolve overlapping edges in p_ elist.*] The overlapping edges are redundant. They not only increase the complexity of the computation, but also introduce ambiguities in the basic-loops and face-loops generation process. *Figure 5* shows three overlapping cases. For the wireframe shown in *Figure 5a*, vertices A, B, C and D are in the *p_vlist*. In the *p_elist*, edge AB overlaps with AD, as shown in *Figure 5b*, or edge BC overlaps with edge AD, as shown in *Figure 5c*, or edge AC overlaps with edge BD, as shown in *Figure 5d.* The overlapping parts of edges are considered to be 'redundant edges', and they are removed by the redundant-edges removal (RER) procedure as discussed below.

Procedure RER: Given a *p_elist* generated from Step W1, the procedure eliminates redundant edges in the *p_elist*.

(W2.1) Initially, mark all the edges in *p_elist* 'unexamined'.
(W2.2) Select an 'unexamined' edge e_i from *p_elist*, and set $E \leftarrow \{e_i\}$. If all the edges have been examined, procedure RER stops.
(W2.3) For every edge $e_j \in$ *p_elist*, $e_j \neq e_i$, if e_j and e_i are collinear and e_j overlaps with one edge in E, then set $E \leftarrow E \cup \{e_j\}$.
(W2.4) If there is only one edge in E, then no other edges overlap with the edge in E. Mark this edge 'examined', and go to Step W2.2; otherwise, continue.
(W2.5) If there is more than one edge in E, remove all the edges in E from *p_elist*, and sort all the vertices of the edges in E according to their coordinate values. Let $v_1 v_2 \dots v_k$ be the sorted vertex sequence. Add edges (v_j, v_{j+1}), $j = 1, \dots, k-1$, into *p_elist*, and mark them 'examined'. Go to Step W2.2.

Using this procedure, the edges in *p_elist* are all unique and nonoverlapped between any two edges. After procedure RER, for example, only edges AB and BD are left in *p_elist* for *Figure 5b,* and only edges AB, BC and CD are left for *Figures 5c* and *d.* Although we have removed redundant overlapped edges, some pathological edges/vertices, such as dangling edges, isolated edges and vertices, may exist and need to be eliminated. On the basis of wireframe-projection properties, the pathological-edges/vertices removal (PEVR) procedure is designed to remove these pathological edges/vertices. The procedure is given in Step W3.

(W3) [*Remove pathological vertices and edges.*] Let $\rho(p\_vertex)$ represent the number of *p_edges* shared at *p_vertex*. Pathological edges or vertices are deleted by the PEVR procedure described below.

Procedure PEVR: Given *p_vlist* and *p_elist*, the procedure removes pathological edges and vertices.

(W3.1) Delete an isolated *p_vertex* if $\rho(p\_vertex)=0$. See vertex v in *Figure 6a*.

(W3.2) If $\rho(p\_vertex)=1$, remove the dangling edge from *p_elist* at the *p_vertex*, and remove the *p_vertex* from *p_vlist*. As shown in *Figure 6b*, edge e and vertex v are removed.

(W3.3) If $\rho(p\_vertex)=2$, and two edges are collinear, delete the *p_vertex* and merge two adjacent edges at the *p_vertex*. As shown in *Figure 6c*, v is removed, and e_1, e_2 are merged. If $\rho(p\_vertex)=2$, and two edges are not collinear, delete both *p_vertex* and two adjacent edges at this *p_vertex*. In *Figure 6d*, v, e_1 and e_2 are all removed.

(W3.4) If $\rho(p\_vertex)=3$, and two of three edges are collinear, delete the *p_vertex* and the third edge. Merge these two collinear edges. In *Figure 6e*, v and e_2 are removed; e_1 and e_3 are merged.

(W3.5) If $\rho(p\_vertex)=3$, and three edges are coplanar, but any two edges are not collinear, delete *p_vertex* and these three edges. In *Figure 6f*, v, e_1, e_2 and e_3 are all removed.

(W3.6) If $\rho(p\_vertex)\geqslant 4$, the four edges are coplanar, and only two edges are collinear, then merge two collinear edges, and delete *p_vertex* and two noncollinear edges. In *Figure 6g*, v, e_2 and e_3 are removed, and e_1 and e_4 are merged. If two pairs of edges are collinear or all the edges are noncollinear, then delete *p_vertex* and all the edges. In *Figure 6h* and *i*, v, e_1, e_2, e_3 and e_4 are all removed.

End of WC algorithm

We have established the wireframe of an object with the *p_vlist* and *p_elist* containing the information about 3D vertices and edges. After we have obtained the wireframe, we need to construct planar graphs to acquire face-loops information. It is essential to mention that the PEVR procedure eliminates current pathological vertices and edges generated from Step W2. However, some potential pathological edges can be discovered from the face-loop-generating procedures described below. The PEVR procedure is applied again if pathological vertices and edges are found until no more pathological vertices and edges can be removed. The PEVR procedure finally reaches a stable state in *p_elist* and *p_vlist*.

Constructing planar graphs

The WC algorithm constructs a wireframe for the input 2D edges and vertices. The wireframe does not explicitly give any information about the surfaces of objects. We need to construct the planar graphs from this wireframe to generate the surface information about objects. A planar graph is a collection of coplanar edges. The algorithm for generating all distinct planar graphs is discussed in this section. The input of this algorithm is *p_elist* and *p_vlist* from the WC algorithm discussed in the previous section.

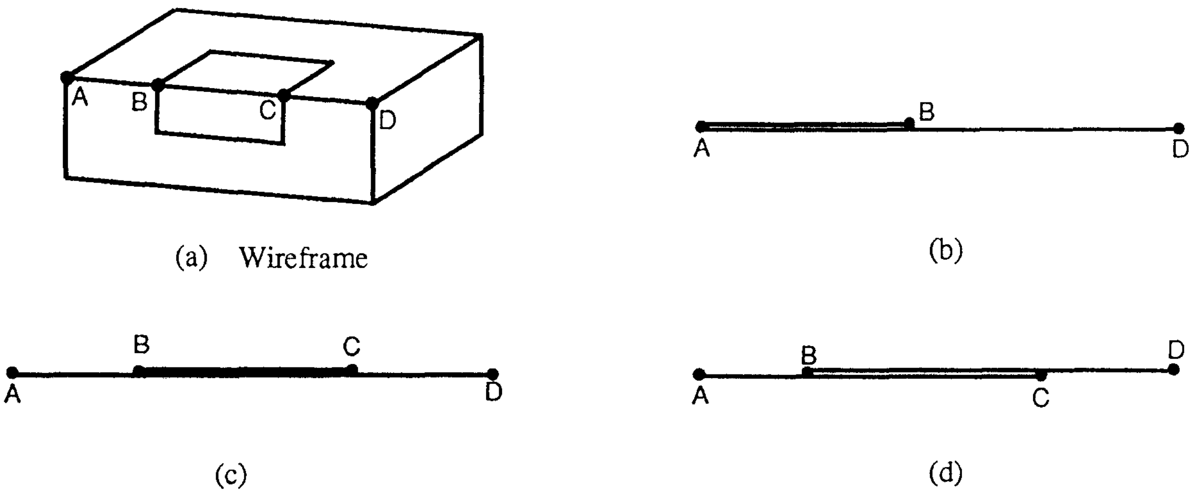


Figure 5 Handling of overlapped edges

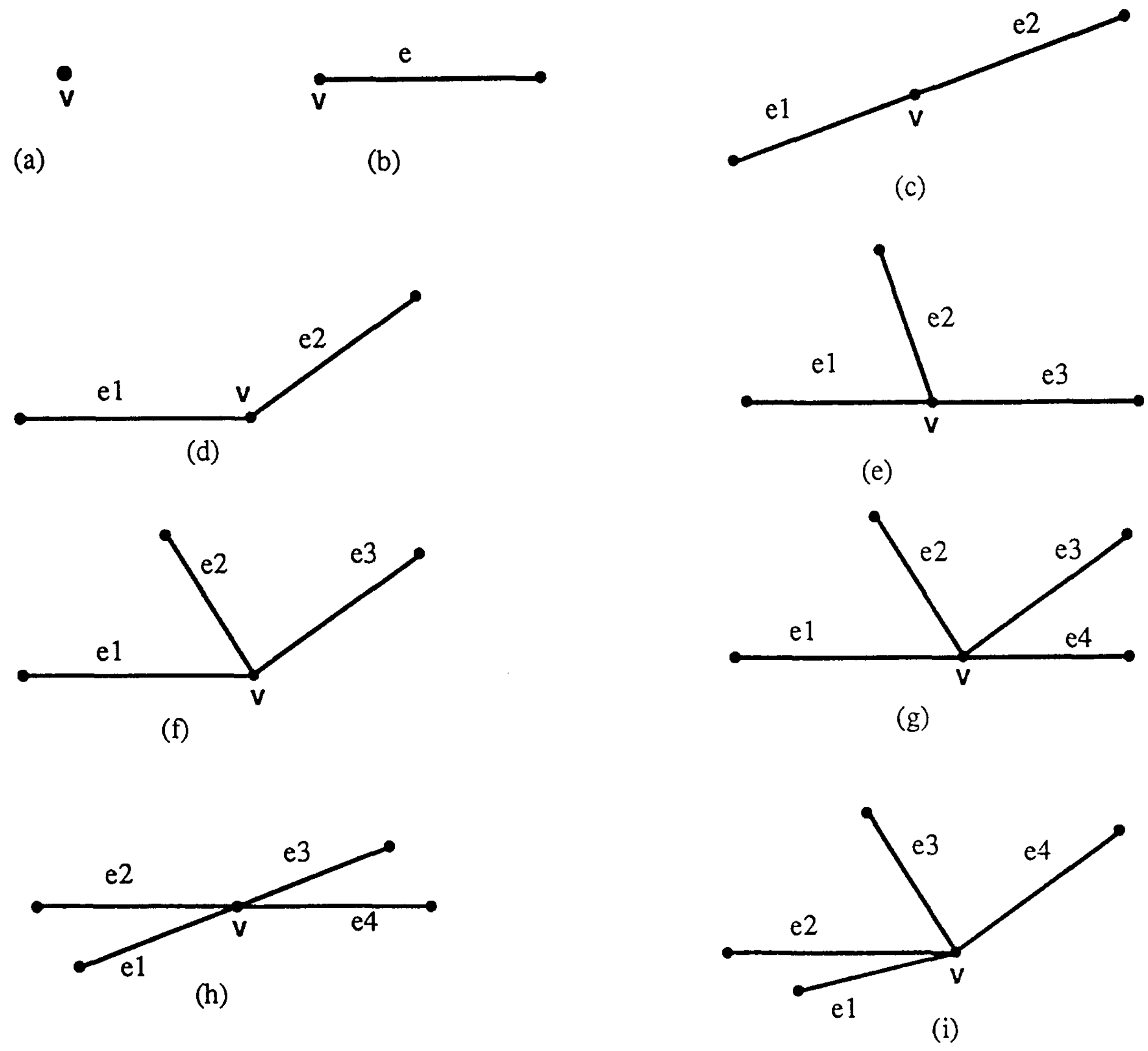


Figure 6 Removal of redundant edges; (a) deg=0, (b) deg=1, (c) deg=2, (d) deg=2, (e) deg=3, (f) deg=3, (g) deg=4, (h) deg=4, (i) deg=4

Algorithm PGG (planar-graph generation):

(P1) Record the adjacent edges for each vertex in *p_vlist*.

(P2) [*Construct planes.*] A plane can be determined by any two noncollinear adjacent edges at this vertex. For each v_i in *p_vlist*, construct a plane determined by e_1 and e_2, where $e_1=(v_i, v_j)$, and $e_2=(v_i, v_k)$, $i \neq k$.

(P3) [*Hypothesize the outer normal vector of a plane.*] Since we do not know the outer side of the plane that was constructed above, the outer normal vector of each plane cannot be determined. The hypothetical normal vector of a plane is a vector starting from the origin, and perpendicularly penetrating the plane. As shown in *Figure 7*. **OP** is a hypothetical outer normal of that plane. The side from which the hypothetical normal vector leaves the plane is the positive side of the plane, and the other side is the negative side. In *Figure 7*, $+\mathbf{N}$ is the hypothetical outer normal vector of the plane on the positive side. The hypothetical outer normal vector of the plane determined in Step P2 can be obtained as follows:

$$N=\begin{cases} -(a\mathbf{i}+b\mathbf{j}+c\mathbf{k}) & d<0 \\ a\mathbf{i}+b\mathbf{j}+c\mathbf{k} & \text{otherwise} \end{cases}$$

where i, j, k are unit vectors in the x, y, z directions, respectively.

(P4) [*Eliminate a duplicated plane.*] Delete plane j if there exists a plane i, $i \neq j$, such that the distance $D_j \leqslant \xi$ (ξ is a tolerance number, say 10^{-5}), where D_j is approximated by $((a_i-a_j)^2+(b_i-b_j)^2+(c_i-c_j)^2+(d_i-d_j)^2)^{1/2}$, (a_i, b_i, c_i, d_i) and (a_j, b_j, c_j, d_j) are the coefficients of the corresponding plane equations.

(P5) [*Search for all the edges on the plane.*] For every *p_edge* in the *p_elist*, compute the distances of its two endpoints D_1 and D_2 to the plane. If $|D_1| \leqslant \xi$ and $|D_2| \leqslant \xi$, *p_edge* is on the plane. The *p_edge* on the plane is used to generate face loops.

(P6) [*Check the legality of each planar graph.*] Although all the planar graphs are generated, some of them are pathological, and need to be adjusted or removed.

(P6.1) If a planar graph consists of two or fewer edges, discard this planar graph.

(P6.2) If a *p_edge* in *p_elist* belongs to only one planar graph, delete this *p_edge*. Apply the PEVR procedure to remove any new pathological edges and vertices that appear.

(P6.3) If the boundary of a planar graph is not closed, i.e. there exist isolated or dangling edges, as shown in *Figure 8*, remove the isolated or dangling edges from this planar graph, and check again for its closeness. Remove the unclosed planar graphs, if any.

End of PGG algorithm

At this point, we have successfully generated the planar graphs and the hypothetical normal vectors of the planes. With these planar graphs, 2D broken- or solid-line attributes are examined to verify the consistency of the constructed wireframe.

Handling broken edges in planar views

Without planes and their outer normal vectors, we are unable to determine the orientation relationship between planes and edges. When plane equations and hypothetical normal vectors are available, we can investigate the legality of the *p_edges* being generated from 2D edges of broken- or solid-line type in the planar views. The following procedure removes the invalid 3D edges from the broken-line information given by the input views.

Procedure: invalid edge removal from broken-line information

For any *p_edge* in *p_elist*, the following steps are applied.

(I1) [*Check its frontal-view projection.*] From the input data, we know the type of projection line of each 2D edge. If its frontal projection is a broken line, there should be at least one planar graph that blocks this *p_edge* when viewing from the direction of $+Oy$. If no such planar graph exists, this *p_edge* must be removed; go to Step I4. If this *p_edge* is valid, then continue.

(I2) [*Check its top-view projection.*] If its top projection is a broken line, there should be at least one planar graph that blocks this *p_edge* when viewing from the direction of $+Oz$. If no such planar graph exists, this *p_edge* must be removed; go to Step I4. If this *p_edge* is valid, then continue.

(I3) [*Check its side-view projection.*] If its side projection is a broken line, there should be at least one planar graph that blocks this *p_edge* when viewing from the direction of $+Ox$. If no such planar graph exists, this *p_edge* is removed; go to Step I4.

(I4) If every *p_edge* has been examined, then go to Step I5; otherwise, select another *p_edge*, and go to Step I1.

(I5) The removal of invalid edges may create new pathological edges and vertices. Therefore, the PEVR procedure is applied to purge further pathological edges and vertices.

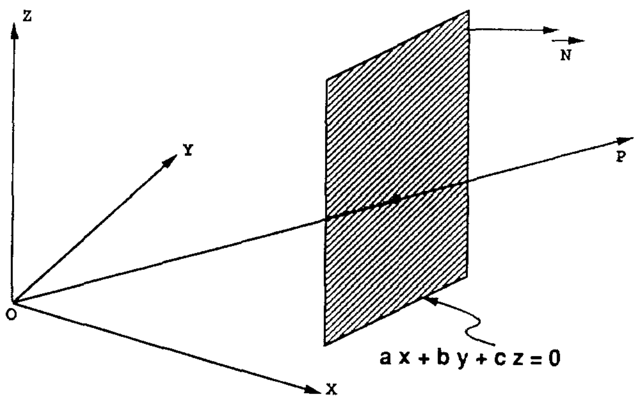


Figure 7 Definition of hypothetical outer normal vector of plane

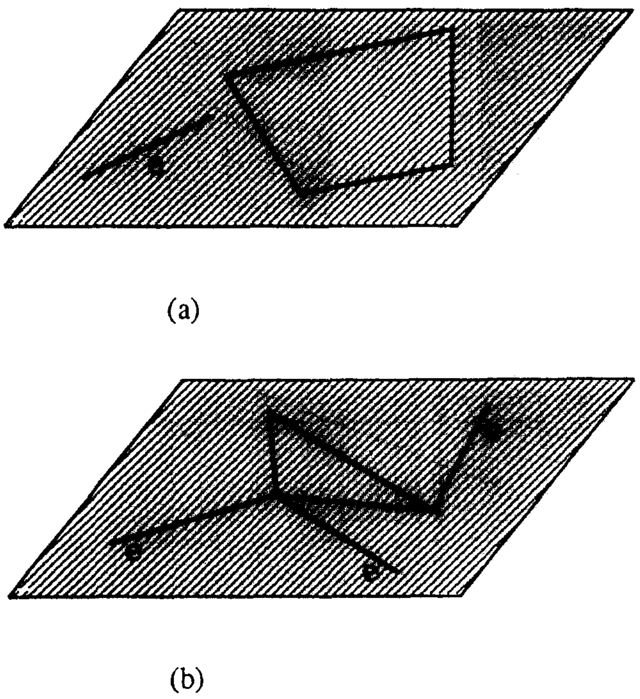


Figure 8 Irregular planar graphs; (a) isolating edge, (b) dangling edges

The processing of broken-line information is beneficial before the generation of face loops in the following steps because the elimination of irrelevant edges results in the faster computation of basic loops, face loops, and body loops, because of a reduction of the ambiguity caused by irrelevant edges. With broken-line information processing, many redundant edges are removed. This avoids much further redundancy when face loops and body loops are generated, and thus the computation is efficient.

Generation of face loops

In previous sections, we have collected information about vertices, edges and planar graphs. To determine the face loops of an object, we need to generate all the basic loops in each planar graph. On the basis of the definitions of loops and face loops give in the appendix, we present the face-loop generation (FLG) algorithm, which determines all the basic loops that are used to construct all the face loops for each planar graph. Let $\psi = \{L_1, L_2, \dots, L_k\}$ be a set of *basic loops* in which $L_i \cap L_j = \{edges, vertices\}$, and $L_i \not\subseteq L_j$ or $L_j \not\subseteq L_i$, for all i and j. Any loop $L_k \notin \psi$ can be obtained from

$$L_k = \bigcup_{s=1}^{m} L_{k_s}$$

for some $m \geqslant 1$, where $L_{k_s} \in \psi$. In addition to generating face loops, the FLG algorithm also determines the relative position between two loops (i.e. two loops are separated or overlapped), and it classifies inner/outer loops and generates face loops from the basic loops. An example is given at the end of this algorithm.

Algorithm FLG (face-loop generation): Given a planar graph generated by the PGG algorithm, this algorithm generates all the face loops on the planar graph.

(F1) [*Construct vertex–edge adjacent table for the input planar graph.*] For each vertex in the planar graph, construct a vertex–edge list (v, ne, SE), where $v \in p\_vlist$, ne is the number of adjacent edges at v, and $SE = \{e_1, e_2, \dots, e_{ne}\}$ is a list of all the adjacent edges at v on the planar graph. The vertex–edge table *adj_tab* is a collection of all the vertex–edge lists.

(F2) [*Find the ordered adjacent edge of a vertex.*]

- (F2.1) Select an entry (v, ne, SE) from *adj_tab*.
- (F2.2) Let $OE(v)$ be a set of ordered adjacent edges at v. The edges in $OE(v)$ are arranged in ascending order on the basis of the *clockwise* angle with respect to a selected edge e_k. The clockwise angle is calculated around the hypothetical outer normal vector of the plane.
- (F2.3) If all the entries have been processed, go to Step F3; otherwise, go to Step F2.1.

(F3) [*Find all the basic loops within a planar graph.*] Each edge (v_i, v_j) has two alternative directions in which to travel: from v_i to v_j, and *vice versa*. Let E_c denote a selected current edge, and let V_{start} and V_{end} denote its starting vertex and ending vertex of a basic loop, respectively. Initially, let the basic-loop set $\psi \leftarrow \varnothing$, and a basic-loop edge set $L \leftarrow \varnothing$.

- (F3.1) Select an edge $e = (v_i, v_j)$ in the planar graph, and let $E_c \leftarrow e$. If all the edges have been selected in this planar graph, then go to Step F4; otherwise, $L \leftarrow L \cup \{E_c\}$, and continue.
- (F3.2) Select the adjacent edge of E_c at V_{start} from $OE(V_{start})$. Suppose that $OE(V_{start}) = \langle e_1, \dots, E_c, e_{c'}, \dots, e_{ne}, e_1 \rangle$. Then, $e_{c'} = (v_k, v_m)$ is the *first adjacent edge* of E_c at V_{start}. Set $L = L \cup e_{c'}$.
- (F3.3) If E_c has visited in both directions in the current basic loop L, then E_c is a *bridge* in the planar graph, and is removed. Let

  $$L = \langle e_1, \dots, e_j, E_c, e_k, \dots, e_l, E_c \rangle$$

  A basic loop is formed by the edges from e_k to e_l in L. Set $\psi \leftarrow \psi \cup \{L\}$.
- (F3.4) If $V_{start} = V_{end}$, a basic loop is discovered; set $\psi = \psi \cup L$, and let $L = \varnothing$, and go to Step F3.1; otherwise, go to Step F3.2.

(F4) [*Identify inclusion relationship between the basic loops on a planar graph.*] For every pair of loops L_i and L_j, if L_i includes L_j, then *include* $(i,j) = 1$, else *include* $(i,j) = 0$.

(F5) [*Form face loops.*] Given the inclusion relationship between basic loops,

> If L_i does not include any loops, then L_i forms a face loop;
> Else if L_i includes L_j, then L_i is an outer loop, L_j is an inner loop, and L_i includes L_j directly;
> Else if loop L_i includes more than one loop, i.e. L_{j_1}, L_{j_2}, $\dots$, L_{j_n}, check whether the L_js are included by other basic loops.

A face loop is formed from L_i and those basic loops included by L_i *directly*. In this face loop, L_i is the outer basic loop, and the others are inner basic loops.

End of FLG algorithm

For each planar graph, we use the FLG algorithm to generate the face loops. Because dangling edges or isolated edges are removed when the planar graphs are found, each edge should belong to two or more different face loops after all the face loops have been constructed. If an edge connects fewer than two face loops, this edge is pathological, and it should be removed from the *p_elist*. In this case, the PEVR procedure is applied again to purge this redundant edge.

Example: As an example, in *Figure 9a*, we trace the FLG algorithm. The eight basic loops are shown in *Figures 9b–i*. The directions of the loops are shown by the arrows. L_1, L_7 and L_8 are outer loops. They do not participate in the generation of face loops, and they are removed. The inner basic loops are L_2, L_3, L_4, L_5 and L_6. With L_2–L_6, all the face loops on this planar graph are generated by Step F4. The face loops generated are shown in *Figure 10*.

Cutting edges and cutting vertices

In the previous section, all the face loops were generated by traversing the planar graphs on each plane. The face loops on different planes may intersect each other, and create edges and vertices; these are called cutting edges and cutting vertices, respectively. With cutting edges and cutting vertices, more face loops in different planes can be discovered. Given any two noncoplanar face loops, the

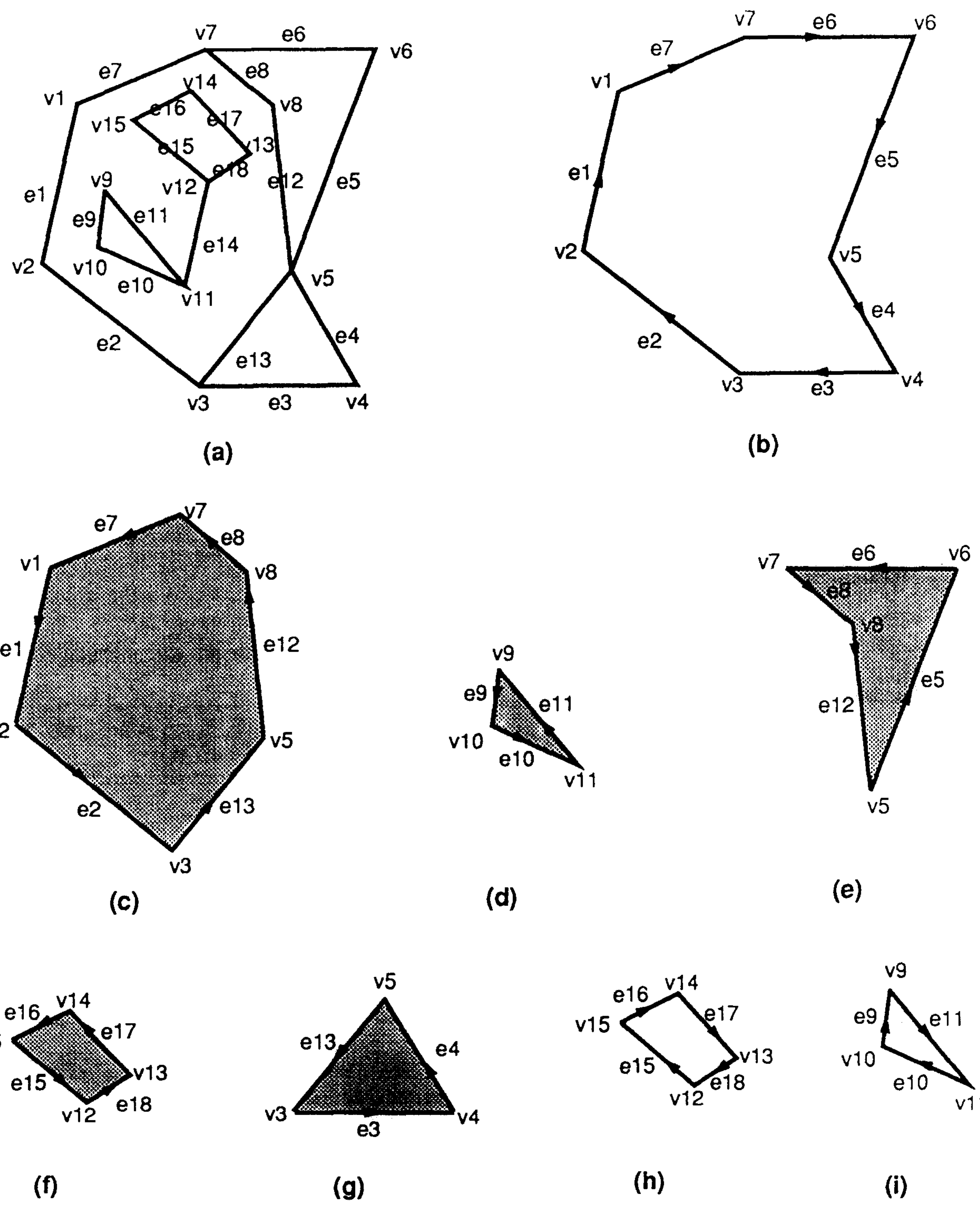


Figure 9 Typical planar graph and its basic loops; (a) graph, (b) L_1 (c) L_2, (d) L_3, (e) L_4, (f) L_5, (g) L_6, (h) L_7, (i) L_8 [L_1, L_7 and L_8 are outer basic loops; L_2–L_6 are inner basic loops.]

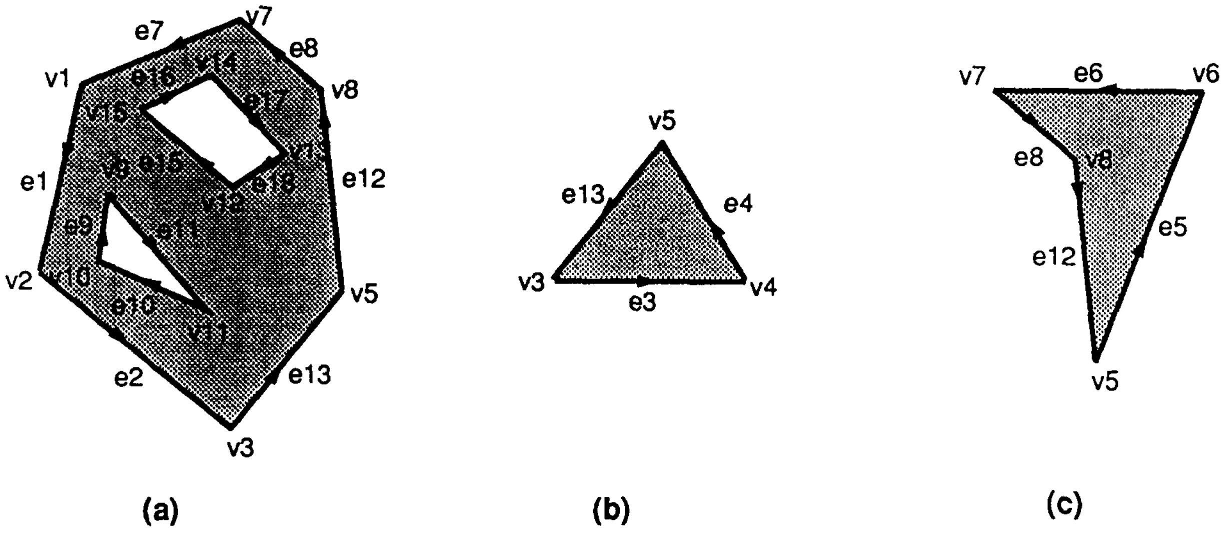


Figure 10 Face loops generated from *Figure 9a* planar graph; (a) F_1, (b) F_2, (c) F_3

intersecting line between them can be classified into the following three cases.

- *Case 1:* The intersecting line is an edge of the face loop, as shown in *Figure 11a*.
- *Case 2:* The intersecting line falls into both face loops, and its two endpoints are on the boundaries of both face loops (see *Figure 11b*).
- *Case 3:* The intersecting line falls in both of the face loops, but not as in case 2 (see *Figure 11c*).

The intersection of face loops in case 1 is regular. In case 2, cutting edges/vertices are necessary, while case 3 is pathological. As a result, in case 3, the face loop that includes both the endpoints of the intersecting line should be removed.

After the cutting edges and vertices have been found, the relationship of any two face loops is that of either intersecting at an edge or separated. It is necessary to mention that these cutting edges and vertices do not actually exist in the object. They are introduced only to assist in the generation of correct objects. Therefore, the cutting edges and vertices are removed when the final object is constructed.

Generation of body loops

Body loops can be viewed as a group of subobjects which do not have common interior points. They can share some vertices, edges or face loops. On the basis of three orthographic views, every body loop has a unique location in space.

Algorithm BLG (body-loop generation): Given the face loops $\mathfrak{F}_1, \mathfrak{F}_2, \ldots, \mathfrak{F}_m$, the BLG algorithm discovers all the body loops.

(B1) [*Initialization.*] A face loop $\mathfrak{F}_i$ has two sides. The side from which the hypothetical normal vector comes is called the positive side, denoted as $+\mathfrak{F}_i$, while the other side is called the negative side, denoted as $-\mathfrak{F}_i$. Set each side of $\mathfrak{F}_i$ as 'unused' by setting $\chi(i,+)\leftarrow 0$, $\chi(i,-)\leftarrow 0$, $1 \leqslant i \leqslant m$. Let the set of face loops $S(\mathfrak{F})\leftarrow\varnothing$.

(B2) [*Select a starting face loop and its side selection.*] Select an $\mathfrak{F}_j$ as a starting face loop to initiate a body loop search. If $\chi(j,+)=0$, then set $S(\mathfrak{F})\leftarrow S(\mathfrak{F})\cup\{+\mathfrak{F}_j\}$ and $\chi(j,+)\leftarrow 1$, and mark $+\mathfrak{F}_j$ in $S(\mathfrak{F})$ as 'unexpanded'; go to Step B3. If $\chi(j,-)=0$, then set $S(\mathfrak{F})\leftarrow S(\mathfrak{F})\cup\{-\mathfrak{F}_j\}$ and $\chi(j,-)\leftarrow 1$, and mark $-\mathfrak{F}_j$ in $S(\mathfrak{F})$ as 'unexpanded'.

(B3) [*Select an 'unexpanded' face loop $\oplus\mathfrak{F}_k$ ($\oplus$ could be $+$ or $-$) from $S(\mathfrak{F})$, and compute the successive face loops of each edge on $\oplus\mathfrak{F}_k$'s boundary.*] Let the adjacent face loops at edge e on $\mathfrak{F}_k$ be $\mathfrak{F}_1, \mathfrak{F}_2, \ldots, \mathfrak{F}_n$, $n \geqslant 2$. The *successive face loop* of $\oplus\mathfrak{F}_k$ at edge e is $\oplus\mathfrak{F}_s$, $1 \leqslant s \leqslant n$ and $s \neq k$, and $\mathfrak{F}_s$ needs the smallest rotating angle to coincide with $\oplus\mathfrak{F}_k$. Let α be the rotating angle between $\oplus\mathfrak{F}_k$ and $\oplus\mathfrak{F}_s$. Let $\mathbf{n}_k$ and $\mathbf{n}_s$ be the hypothetical normal vectors of $\mathfrak{F}_k$ and $\mathfrak{F}_s$, respectively. Let θ be the angle between $\mathbf{n}_k$ and $\mathbf{n}_s$. Then, we have

$$\theta = \cos^{-1}\left[\frac{\mathbf{n}_s \cdot \mathbf{n}_k}{|\mathbf{n}_s||\mathbf{n}_k|}\right]$$

Let a unit vector $\mathbf{e}$ be a vector along edge e, and satisfy the right-hand rule with $\mathbf{n}_k$. If $\oplus\mathfrak{F}_k$ is $+\mathfrak{F}_k$, its successive face loop at edge e can be computed with the following criteria. We find out the values of α and the side selection $\oplus$ for each adjacent face loop at edge e so that we choose the face loop with the smallest α value.

- If $\mathbf{n}_s \times \mathbf{n}_k$ is in the same direction as $\mathbf{e}$, and $+\mathfrak{F}_s$ is to the right of $\mathbf{e}$, then $\alpha = \pi - \theta$; assign $+\mathfrak{F}_s$ to $\oplus\mathfrak{F}_s$ (see *Figure 12a*).

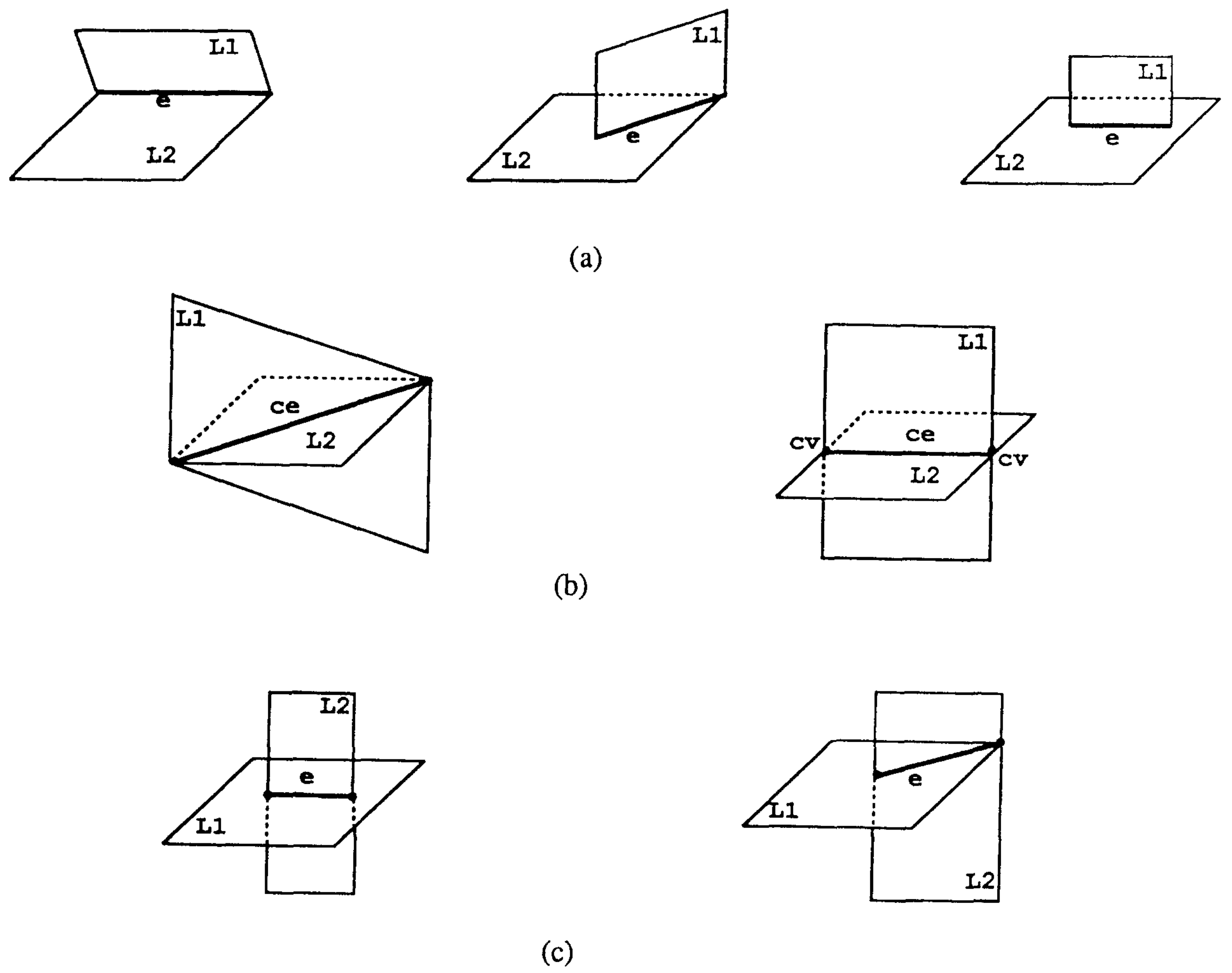


Figure 11 Classification of intersection line between two face loops

- If $\mathbf{n}_s \times \mathbf{n}_k$ is in the opposite direction from $\mathbf{e}$, and $+\mathfrak{F}_s$ is to the left of $\mathbf{e}$, then $\alpha = \theta$; assign $-\mathfrak{F}_s$ to $\oplus\mathfrak{F}_s$ (see *Figure 12b*).
- If $\mathbf{n}_s \times \mathbf{n}_k$ is in the same direction as $\mathbf{e}$, and $+\mathfrak{F}_s$ is to the left of $\mathbf{e}$, then $\alpha = 2\pi - \theta$; assign $-\mathfrak{F}_s$ to $\oplus\mathfrak{F}_s$ (see *Figure 12c*).
- If $\mathbf{n}_s \times \mathbf{n}_k$ is in the opposite direction to $\mathbf{e}$, and $+\mathfrak{F}_s$ is to the right of $\mathbf{e}$, then $\alpha = \pi + \theta$; assign $+\mathfrak{F}_s$ to $\oplus\mathfrak{F}_s$ (see *Figure 12d*).

If $+\mathfrak{F}_s \notin S(\mathfrak{F})$, then $S(\mathfrak{F}) \leftarrow S(\mathfrak{F}) \cup \{+\mathfrak{F}_s\}$ and $\chi(s, +) \leftarrow 1$. Mark $+\mathfrak{F}_s$ 'unexpanded' in $S(\mathfrak{F})$. This side-selection procedure guarantees that either all the face loops in $S(\mathfrak{F})$ are facing to the interior of the body loop being formed, or they are facing to the exterior of the body loop being formed. When the successive face loop of $\mathfrak{F}_k$ at each edge on $\mathfrak{F}_k$ has been determined using the above criteria, mark $+\mathfrak{F}_k$ 'expanded'. Go to Step B4. Similarly, if $\oplus\mathfrak{F}_k$ is $-\mathfrak{F}_k$, its successive face loop at edge e can be determined accordingly; go to Step B4. As an example, the hypothetical normal vectors of the adjacent face loops $\mathfrak{F}_1$–$\mathfrak{F}_5$ in *Figure 13a* are shown in *Figure 13b*. The successive face loops' counterparts are $+\mathfrak{F}_1$ and $-\mathfrak{F}_2$, $-\mathfrak{F}_1$ and $-\mathfrak{F}_5$, $+\mathfrak{F}_4$ and $-\mathfrak{F}_3$, and $-\mathfrak{F}_4$ and $+\mathfrak{F}_5$.

(B4) Repeat Step B3 until there are no new face loops to be expanded in $S(\mathfrak{F})$. The face loops in $S(\mathfrak{F})$ may form a body loop. Go to Step B5.

(B5) [*Check body-loop legality.*] A body loop must be closed by face loops, without dangling face loops. As for the current body loop in $S(\mathfrak{F})$, if every edge involved in this body loop is shared by two and only two face loops, then the face loops in the current $S(\mathfrak{F})$ form a valid body loop; store this closed body loop. Otherwise, face loops in $S(\mathfrak{F})$ cannot form a closed body loop; go to Step B6 to initiate another search.

(B6) Set $S(\mathfrak{F}) \leftarrow \varnothing$, and repeat Steps B1–B5 until every face loop $\mathfrak{F}_i$, $1 \leqslant i \leqslant m$, had been visited on both sides.

End of BLG algorithm

The BLG algorithm generates all the body loops from the face loops. Some of the body loops generated above are unbounded, and are called outer body loops. The others are inner body loops, because they are bounded. The outer body loops are useless for constructing objects, and thus they are discarded.

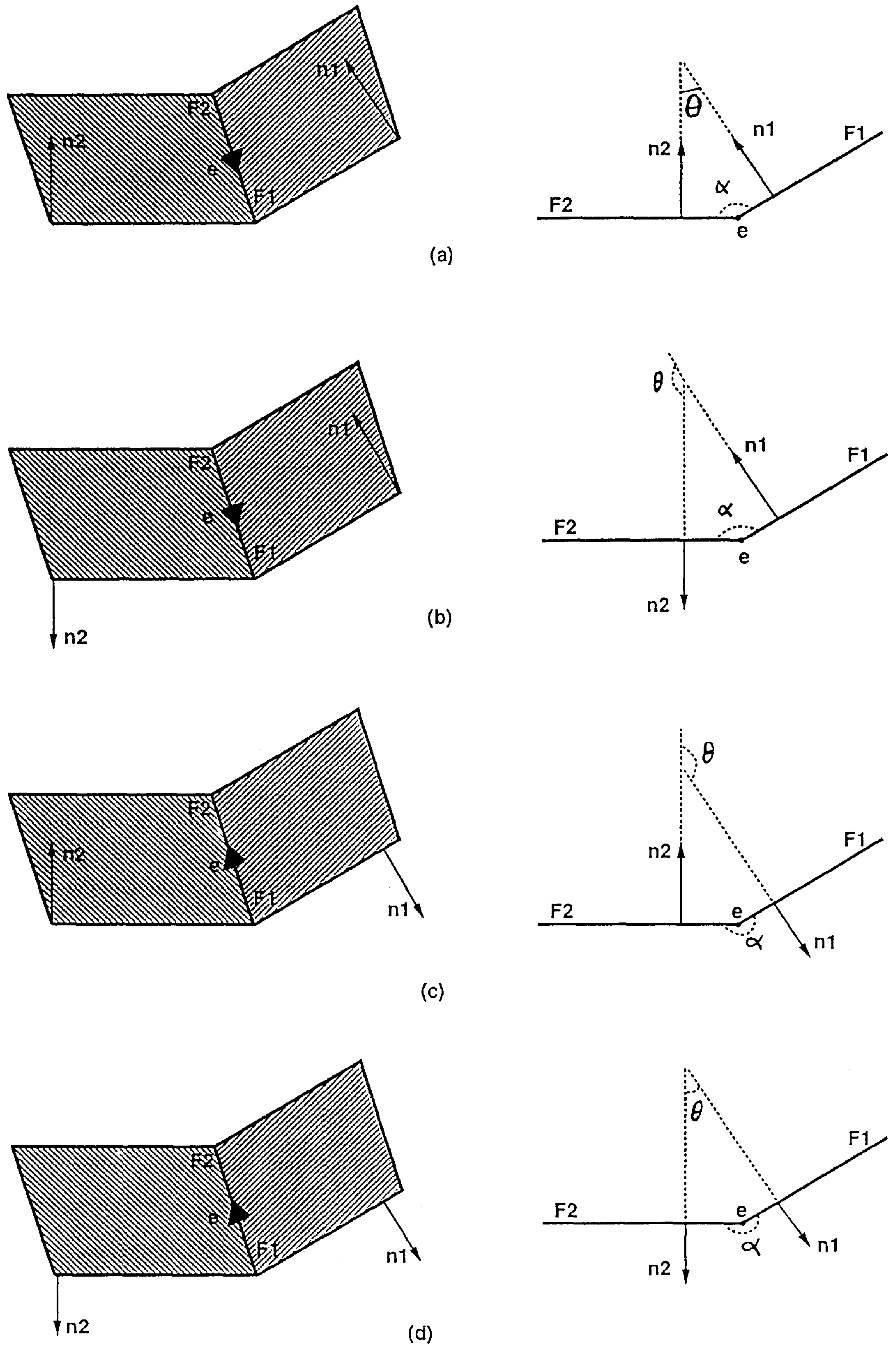


Figure 12 Discoveries of successive face loops

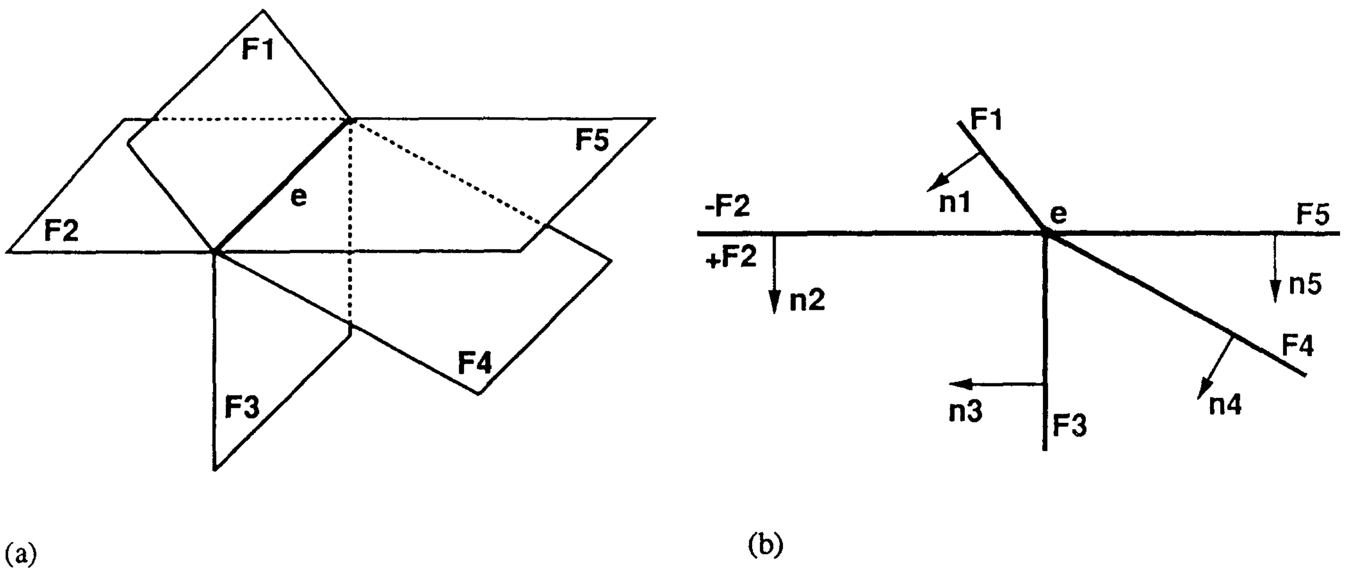


Figure 13 Example of successive face loops; (a) adjacent face loop of edge *e* in space, (b) projection along direction of edge *e*

To classify inner and outer body loops, the side-selection information (i.e. the value of $\oplus$) of the face loops in generating a body loop is important. In a body loop, the sides of all the face loops selected using the above algorithm are facing the interior of the body loop. As a result, the interior of an outer body loop is unbounded, and the exterior is a bounded empty 'solid'. To classify body loops into inner or outer loops, we use the following steps:

- For a body loop, select any two adjacent face loops $\oplus\mathfrak{F}_i$ and $\oplus\mathfrak{F}_j$ at their shared edge *e*, and construct a ray *l* starting at the midpoint of *e* along the direction of

$$\mathbf{l}=\frac{\oplus\mathbf{n}_1}{|\mathbf{n}_1|}+\frac{\oplus\mathbf{n}_2}{|\mathbf{n}_2|}$$

  where $\oplus\mathbf{n}_i$ and $\oplus\mathbf{n}_j$ are the hypothetical normal vectors of $\oplus\mathfrak{F}_i$ and $\oplus\mathfrak{F}_j$, respectively, and $\oplus$ corresponds to their side selection.
- Analyse the situations in which the line *l* intersects with the face loops of the body loop. If the number of intersecting points made by *l* and all the face loops on the body loop (excluding $\mathfrak{F}_i$ and $\mathfrak{F}_j$) is one or more, then the body loop is inner; otherwise, it is outer.

We use the hypothetical normal vectors of the face loops. For any face loop $\mathfrak{F}$ on a body loop, if $\mathfrak{F}$ is selected with the negative side, i.e. $-\mathfrak{F}$, then its hypothetical outer normal vector $+\mathbf{n}$ points to the exterior of the body loop, while $-\mathbf{n}$ points to the interior of the body loop. Similarly, if $\mathfrak{F}$ is selected with the positive side, i.e. $+\mathfrak{F}$, then its hypothetical outer normal vector $+\mathbf{n}$ points to the interior of the body loop, while $-\mathbf{n}$ points to the exterior of the body loop. This implies that the authentic outer normal vector of face loop $-\mathfrak{F}$ on the body loop is $+\mathbf{n}$, or the authentic outer normal vector of face loop $+\mathfrak{F}$ on the body loop is $-\mathbf{n}$. Therefore, we can obtain the authentic outer normal vectors for every face loop on every body loop. The authentic outer normal vector information can be used to construct complete *boundary representations* (R-rep) of objects.

Combination of body loops and making of decisions

For the body loops generated above, we test all the possible combinations of these body loops to guarantee that we can discover all the objects that match the original-input three views. If the projections of a candidate object are consistent with the corresponding three input views, the candidate object is a solution of the input views. Given *N* body loops, the total number of candidates among them is 2^N-1. We check each candidate object for legality, and compare its projections with the input views for consistency.

Legality of candidate object

Since a body loop has its unique orientation in space, the relationship between any two body loops can be classified into only four cases:

- *Case 1:* Two body loops are separated, as shown in *Figure 14a.*
- *Case 2:* Two body loops share some vertices, as shown in *Figure 14b.*
- *Case 3:* Two body loops share some edges, as shown in *Figure 14c.*
- *Case 4:* Two body loops share some face loops, as shown in *Figure 14d.*

For each candidate object, we check its legality in terms of the rules listed below, as shown below.

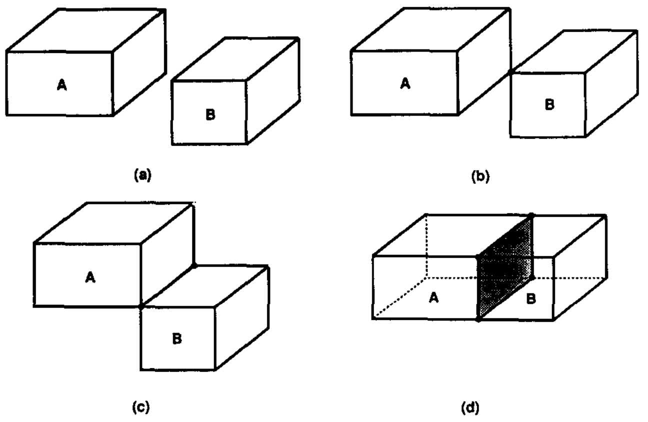


Figure 14 Contact relationship between two objects in space

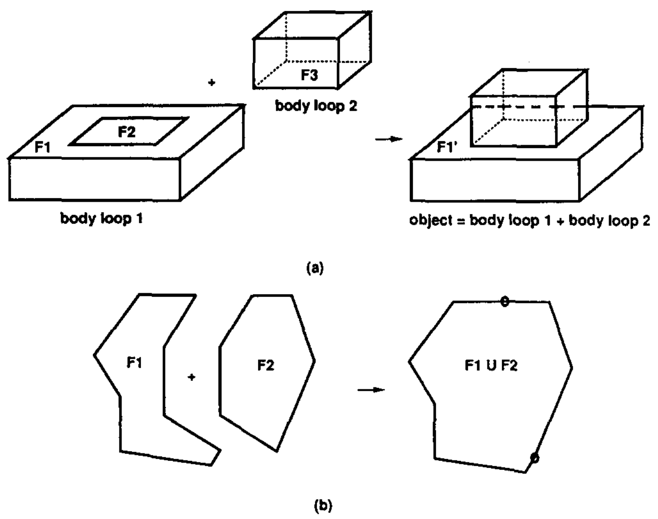


Figure 15 Some special situations for constructed objects; (a) removal of duplicate face loops, (b) removal of duplicate edges

(1) [*Eliminate redundant face loops.*] Eliminate every face loop for which both sides were selected in the same solid, since we know that these face loops are the interior faces of the solid, but not the surface of the solid. Face loop $\mathfrak{F}_2$ in *Figure 15a*, for example, is redundant.

(2) [*Eliminate redundant edges.*] Eliminate those edges which are shared by two and only two coplanar face loops. After these edges have been deleted, merge the two coplanar face loops into one to form a new face loop of the solid (see *Figure 15b*).

(3) [*Discard illegal candidate using vertices information.*] If two face loops only share some vertices, this candidate object is not regular, and is illegal.

(4) [*Discard illegal candidate using edges information.*] Since each edge belongs to two and only two face loops in a solid object, if an edge has more than two adjacent face loops, and the loop connection of an edge is more than two, this candidate object is also not regular.

Each legal candidate is examined further for the consistency of its projections with the input views.

Consistency of candidate object with input three views

First, a candidate object is projected onto the corresponding three projection planes (xOz, xOy, and yOz), and three new views are formed. Second, compare the three new orthographic views with the original input views. If they match each other completely, this object is a solution of the input views.

During the projection, a 3D edge may become a 2D vertex or a 2D line segment. We need to consider the following cases. If a 3D edge is not blocked by any planes, the projection of this 3D edge is a solid segment; otherwise, it is a broken-line segment. If a broken-line segment overlaps with a solid-line segment, the overlapping sections should be solid. Therefore, 3D edges can be projected into a combination of several solid-line segments and several broken-line segments.

With the 2D line segments from the projection, we merge the adjacent line segments of the same type (solid or broken). Compare these 2D line segments with those in the input data files. If the corresponding line segments are fully overlapped and matched, the constructed object is a correct solution. Otherwise, it is not a solution of the input views. In the next section, we give a number of examples to demonstrate the correctness and performance of the algorithms.

EXAMPLES AND ANALYSIS

The algorithm has been implemented in C on a Sun 3/60 workstation. Each step of the procedure for constructing solid models is displayed in realtime. Each example performed by the algorithm consists of three orthographic views of the object and the corresponding 3D wireframe.

Figure 16 shows an example which demonstrates the processing of broken-line information. *Figure 16a* shows the wireframe before broken-line processing, while *Figure 16b* shows the wireframe after broken-line processing. *Figures 16c–f* show the body loops of the wireframe. The correct object is shown in *Figure 16g*. The efficiency of computation is greatly increased by the reduction of redundant edges.

Figure 17 shows another example in which it is necessary to discover cutting edges/vertices. With the discovery of cutting edges/vertices, the planar views can be correctly interpreted to guarantee that the correct objects are reconstructed. *Figure 17a* shows the input three views and the wireframe generated by the WC algorithm. *Figure 17b* shows that cutting edges are discovered, and a cutting vertex is introduced. The body loops for this wireframe are listed in *Figures 17d–m*. On the basis of different interpretations of broken lines, it has three solutions, as shown in *Figures 17n–p*. Without consideration of broken-line information, these input planar views should have four corresponding solid models.

From the above examples, we determined the following

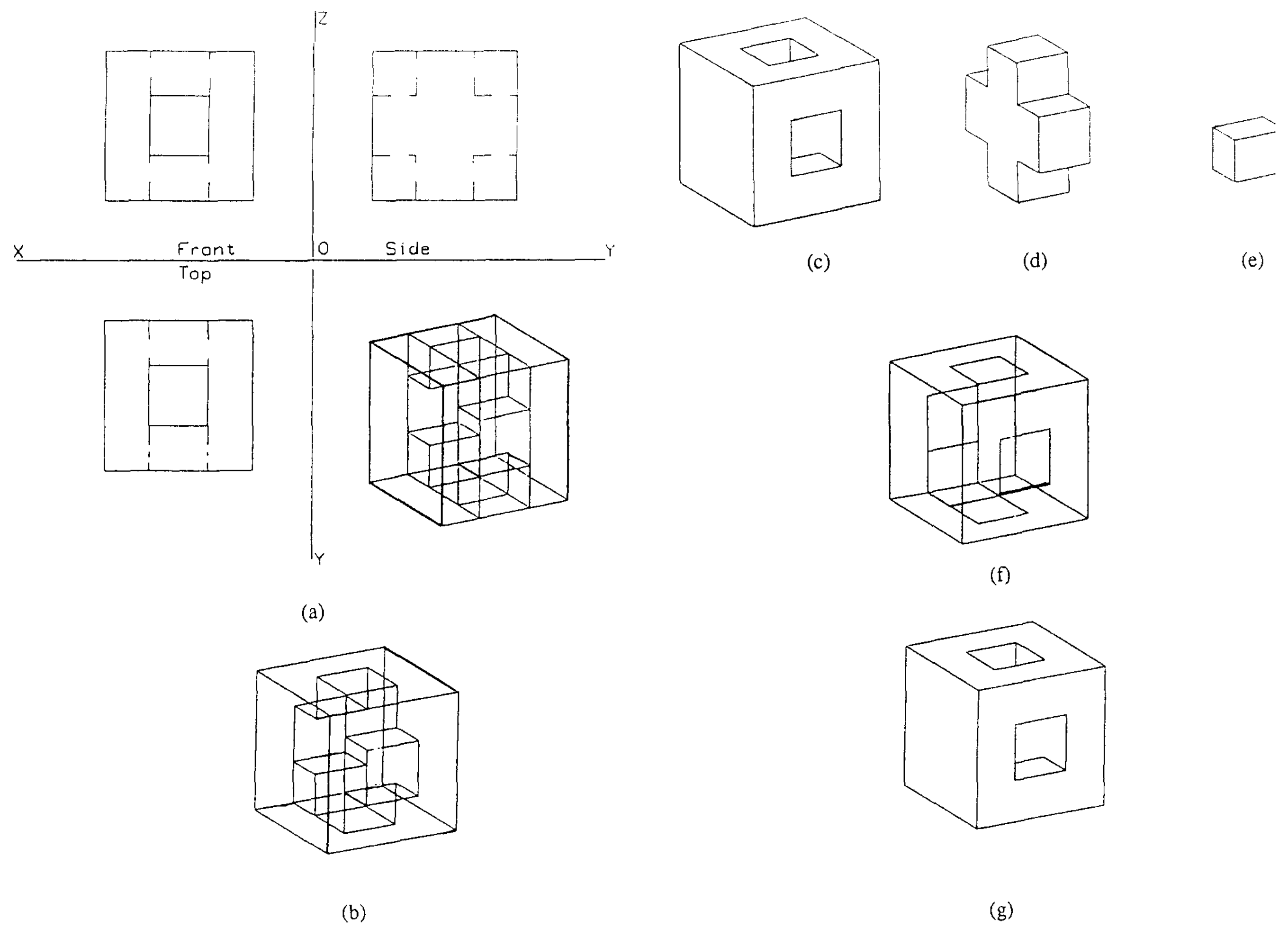


Figure 16 Example of broken-line information processing; (a) three views and wireframe, (b) wireframe after broken-line information processing, (c)–(e) inner body loops, (f) outer body loop, (g) correct object

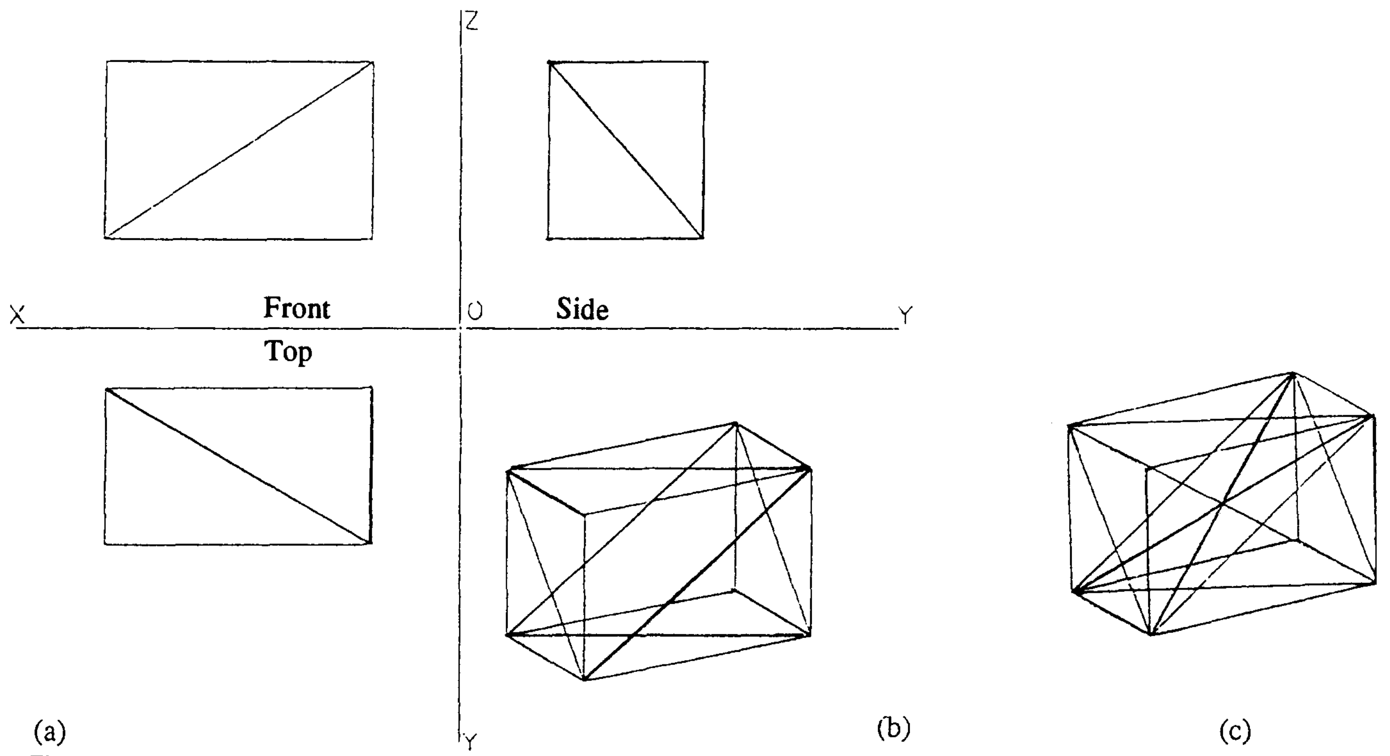


Figure 17 Example with multiple correct objects; (a) three views, (b) wireframe, (c) cutting edges and cutting vertices discovered (*figure continued on next page*)

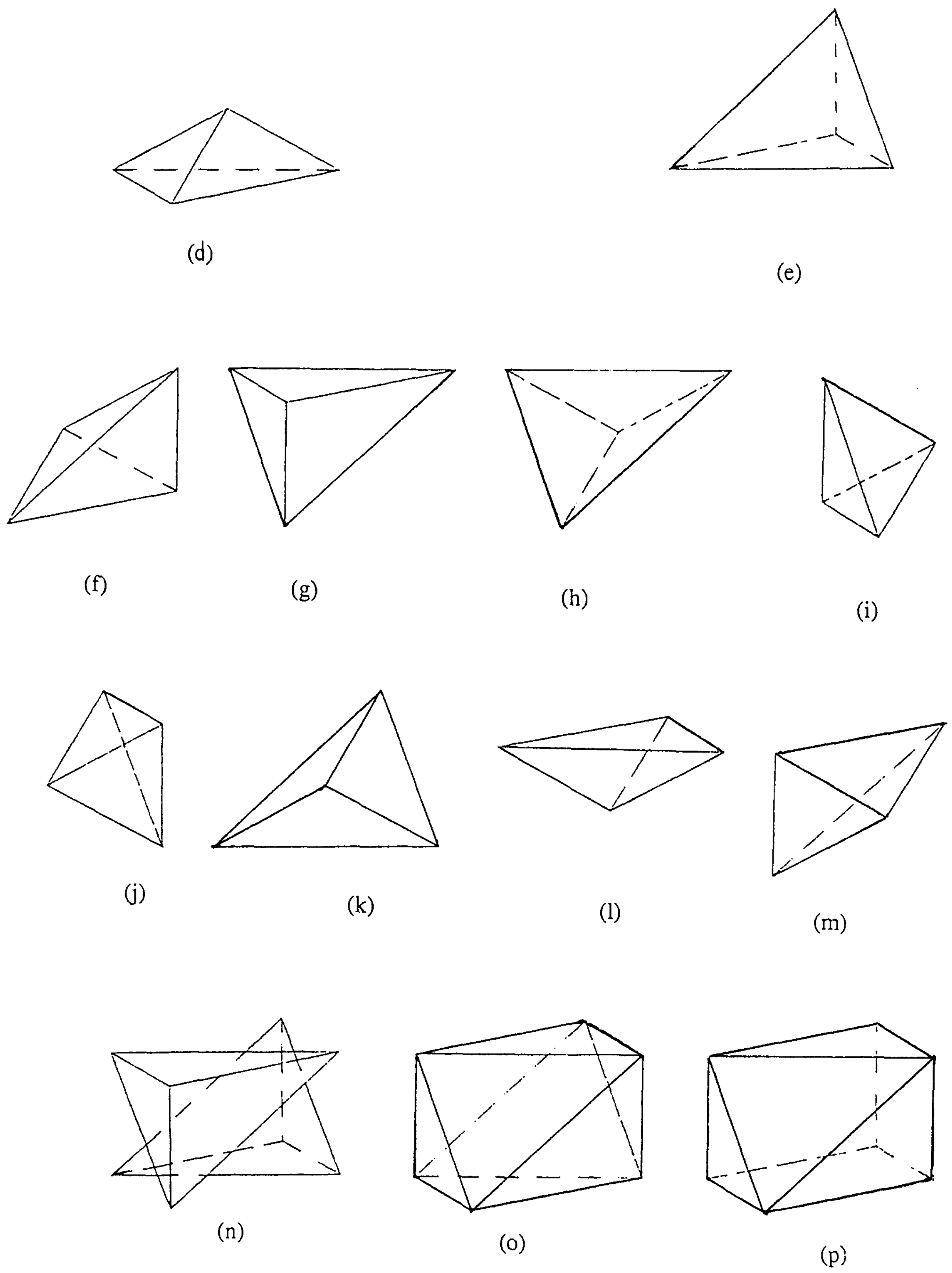


Figure 17 (*continued*) (d) outer body loop, (e)–(m) inner body loops, (n)–(p) three correct solutions

facts:

- The more input data there is, i.e. the more complicated the solid models are, the greater the runtime that is needed to construct the solid.
- There may not always be multiple solutions for highly symmetrical planar views. *Figure 17* has multiple solutions, while *Figure 16* does not.
- The processing efficiency is increased when we consider broken-line information in the algorithm. In *Figure 16*, if broken-line information is ignored, there are six possible body loops. After we process the broken-line information, there are two body loops. The number of candidate objects decreases from $2^6 - 1 = 63$ to $2^2 - 1 = 3$. Therefore, the computation efficiency of identifying the correct objects is improved exponentially.

CONCLUSIONS

The number of investigations of 3D information reconstruction from 2D information has been increasing rapidly. Continuing achievements have been realized in the recovery of solid models from their planar views, but the methods in the algorithms described in the literature are, to a greater or lesser degree, not complete. In this paper, an algorithm is proposed that handles the reconstruction of 3D polyhedral solid models from their planar views. The algorithm can not only find the correct solution or multisolution, but also remove pathological cases, if there are any. The algorithm is guaranteed to find the correct solution from the given inputs, provided that the input views are correct. The algorithm takes full advantage of broken-line information to remove the pathological cases. Our algorithm uses the hypothetical outer normal vectors to determine the classifications of face loops and body loops, and to assist in the construction of body loops. Eventually, by adjustment of the hypothetical outer normal vectors of each face loop on the object, the authentic normal vectors are obtained for the construction of complete boundary representations of objects.

REFERENCES

1 Idesawa, M 'A system to generate a solid figure from a three view' *Bull. Jap. Soc. Mech. Eng.* Vol 16 (Feb 1973) pp 216–225

2 Aldefeld, B 'On automatic recognition of 3D structures from 2D representations' *Comput.-Aided Des.* Vol 15 No 2 (1983) pp 59–64

3 Aldefeld, B and Ricjter, H 'Semiautomatic three dimensional interpretation of line drawings' *Comput. & Graph.* Vol 8 No 4 (1984) pp 371–380

4 Markowsky, G and Wesley, M 'Fleshing out projections' *IBM J. Res. & Develop.* Vol 25 No 6 (1981) pp 934–954

5 Sakurai, H and Gossard, D 'Solid model input through orthographic views' *Comput. Graph.* Vol 17 No 3 (1983) pp 243–252 (*Proc. ACM/SIGGRAPH Conf.*)

6 Gujar, U and Nagendra, I 'Construction of 3D solid objects from orthographic views' *Comput. & Graph.* Vol 13 No 4 (1989) pp 505–521

7 Gu, K, Tang, Z and Sun, J 'Reconstruction of 3D solid objects from orthographic projections' *Comput. Graph. Forum* Vol 10 No 5 (1986) pp 317–324

APPENDIX

Concepts in solid geometry

This appendix gives the definitions of a geometric object, face, vertex, edge etc. In this paper, all objects refer to polyhedrons unless they are explicitly specified otherwise.

Object

The *object* O is a nonempty, bounded region in $\mathbb{R}^3$ space denoted as $O = \{\partial O, iO\}$, where ∂O represents the boundary or surface of the O, and iO is the interior of O. The complement space cO of O is called the exterior of O. The following properties hold for ∂O:

- iO and cO are disjoint subspaces separated by ∂O.
- Removing any point from ∂O makes iO and cO become a connected subspace.
- For any point $p \in \partial O$, if there is a tangent plane at p, then the normal vector of O at p points to the direction of the cO subspace.

Face

The *face F* of an object is a nonempty and bounded plane denoted as $\mathfrak{F} = \{\partial\mathfrak{F}, i\mathfrak{F}\}$, where $\partial\mathfrak{F}$ is the circumference of $\mathfrak{F}$, and $i\mathfrak{F}$ is the interior of $\mathfrak{F}$. In our domain, $\partial\mathfrak{F}$ is composed of straight-line segments and ∂O is made up of a limited number of faces.

Vertex set

The *vertex set* $V(\mathfrak{F})$ of face $\mathfrak{F}$ is the collection of all the intersection points between any two nonlinear line segments. The *vertex set* $V(O)$ of object O is the collection of all the intersection points of any three noncoplanar faces.

Edge set

The *edge set* $E(\mathfrak{F})$ of face $\mathfrak{F}$ is the circumference of face $\mathfrak{F}$, or $\partial\mathfrak{F}$. Each endpoint belongs to $V(\mathfrak{F})$, and any interior point on an edge does not belong to $V(\mathfrak{F})$. The *edge set* $E(O)$ of object O is the collection of the circumference of all the faces on ∂O. Each edge in $E(O)$ satisfies the following:

- Both endpoints are in $V(O)$.
- Any interior point of an edge is not in $V(O)$.
- For any point p in an edge, there always exist two faces $\mathfrak{F}_1 \in bO$ and $\mathfrak{F}_2 \in \partial O$ such that $p \in \partial\mathfrak{F}_1 \cup \partial\mathfrak{F}_2$.

Wireframe

The *wireframe* $WF(O)$ of an object O is defined as a set of all the ordered pairs $[V(O), E(O)]$.

Loop

The *loop* L is a collection of coplanar edges $L = \{e_1, e_2, \ldots, e_k\}$, where e_i is an edge. Each edge in L satisfies the following:

- The intersecting point of any two edges e_i and e_j must be in $V(\mathfrak{F})$.
- An endpoint of each edge can be connected by two and only two edges.
- The loops are all disjoint.

Loops are classified into inner loops and outer loops, in terms of the area that they encompass. If the area is bounded, the loop is called an inner loop; otherwise, it is an outer loop.

Face loop

A *face* can be represented in terms of *loops*. Let L_0, $L_1, \ldots, L_k$ be the loops on face $\mathfrak{F}$, where L_0 is an inner loop that includes $L_1, L_2, \ldots, L_k$ directly, and there is no overlap between these loops, i.e. $L_i \cap L_j = \varnothing$. The face loop $\mathfrak{F}$ is defined as follows:

- The boundary of $\mathfrak{F}$ is determined by
$$\partial\mathfrak{F} = \bigcup_{i=0}^{k} L_i$$
- $\mathfrak{F}$ consists of all the points inside L_0 and outside
$$\bigcup_{i=1}^{k} L_i$$
and the points on
$$\bigcup_{j=0}^{k} L_j$$

Body loop

A *body loop* is the collection of face loops. Let body loop $B = \{\mathfrak{F}_1, \mathfrak{F}_2, \ldots, \mathfrak{F}_m\}$, where $\mathfrak{F}_i$ is a face loop, such that

- for any $\mathfrak{F}_i$, $\mathfrak{F}_j$, $i \neq j$, their intersecting line lies in $\partial F_i \cap \partial F_j$, if they intersect,
- for any edge $e \in E(\mathfrak{F})$, there exist two and only two face loops $\mathfrak{F}_i$ and $\mathfrak{F}_j \in B$, $i \neq j$, such that $e \in \partial\mathfrak{F}_i \cap \partial\mathfrak{F}_j$, which ensures the closure of a body loop.

Similarly, there are inner and outer body loops in terms of the volume that the body loops contain. Bounded body loops are called inner body loops, and unbounded loops are outer body loops. Using the definition of body loop B, an *object* can be defined, alternatively, as follows.

Object

An *object* is a collection of a limited number of inner body loops. Let $O = \{B_1, B_2, \ldots, B_m\}$, where the B_i, $i = 1, 2, \ldots, m$, are inner body loops, such that the following hold:

- $$\partial O = \bigcup_{i=1}^{m} \partial B_i$$

where $\bigcup$ is a regularized set operator.

- The interior of O includes all the points in
$$\bigcup_{i=1}^{m} B_i$$
- There are no overlaps between any two distinct body loops B_i and B_j, $i \neq j$.

Cutting vertices/edges

After the construction from 2D projection data to 3D vertices and edges, these vertices and edges may not form a legal 3D wireframe, and the following pathological cases may exist:

- *Case 1:* The intersecting point of two 3D edges is not the endpoint of the edges, but the interior point of edges.
- *Case 2:* The intersecting line of two planes which are generated from the vertices and edges of the wireframe is on the boundary of the planes, but the intersecting line is not an edge of the wireframe.

To remedy pathological case 1, we add the intersecting point into the vertex set of the wireframe, and call such a vertex a *cutting vertex*. To remedy pathological case 2, we append the intersecting line into the edge set of the wireframe, and call such an edge a *cutting edge*. With cutting vertices/edges, the wireframe is legal in the sense of geometric concepts.

Qin-Wen Yan received a BS and MS in computer science and technology from Tsinghua University, China, in 1988 and 1990, respectively. He gained a second MS from Wright State University, USA, in 1992. He is currently a software scientist with Dataserv, CA, USA. His research interests are in the areas of computer graphics, user interfaces, and computer-aided design.

C L Philip Chen is an assistant professor. He gained a first degree, an MS and a PhD in electrical engineering from NTIT, Taiwan, the University of Michigan, USA, and Purdue University, USA, in 1979, 1985 and 1988, respectively. He was a research assistant at Purdue University and the University of Michigan on projects funded by the US National Science Foundation Engineering Research Center for Intelligent Manufacturing Systems and AFOSR, respectively. His principal research areas are learning-based memory-driven assembly and process planning systems, feature-based design, and computer graphics.

Zeshing Tang is a professor and the director of the Computer Aided Design Technology Research Group at Tsinghua University, China. His current research interests include geometric modelling and its application, volume visualization, and computational geometry. He is the vice chairman of the CAD and Computer Graphics Society of China Computer Federation, and the vice chairman of the Computer Engineering and Application Society of China Electronics Institute.