ELSEVIER

Computer-Aided Design 34 (2002) 321–336

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

Embedding visual cognition in 3D reconstruction from multi-view engineering drawings

Weidong Geng[a,*], Jingbin Wang[b], Yiyang Zhang[b]

[a]*Media Arts Research and Study, Institute for Media Communication, GMD, D-53754, St Augustin, Germany*
[b]*Artificial Intelligence Institute, Department of Computer Science and Engineering, Zhejiang University, Hangzhou, 310027, P.R. China*

Received 6 June 2000; revised 15 December 2000; accepted 29 December 2000

Abstract

In this paper we discuss how to reconstruct 3D models from multi-view engineering draws by employing human engineers' approaches. Human's 'divide and conquer' interpretation strategy in visual cognition is simulated, and successfully carried out on the basis of spatial division of 3D object space. At first, a volume-oriented method is utilized to decompose the 3D object space in a set of 3D 'cell-boxes' whose three-view bounding rectangles will isolate its related sub-projections from input projections views, and in every cell-box a cell primitive is implied. Then, a 3D model of each cell primitive is 'locally' generated from its 'sub-projection' views by wire-frame oriented algorithms. The final interpretation result of the overall projection-views is a 'Union' of these cell primitives. To deal with the ambiguity, a visual reasoning engine is implemented on the basis of principles from Gestalt psychology. It will be activated to pick out the most reasonable interpretation, when ambiguities are generated during the reconstruction process. The section views are also incorporated in getting rid of the ambiguity. Moreover, we design a natural and convenient interaction way to encourage the user to be involved in the process while interpreting complex projections. The key steps of this human-like reconstruction approach are presented in detail.

Keywords: Interpretation of engineering drawings; 3D reconstruction from multi-projections; Visual cognition

1. Introduction

Multi-view engineering drawings are used widely and play an essential role in traditional engineering. 2D orthographic views are not only a powerful representation of 3D objects, but also standard media between design and manufacture. Most existing products are represented in orthographic projections. Since most engineering tasks involve modification of existing designs presented on paper, an automatic system for constructing a 3D model of a multiple-view engineering drawing would make full use of old designs, and have a number of practical applications in addition to being of theoretical interest.

The interpretation of engineering drawings by computer is a challenging task. Its main objective is to enable the computer to have human engineers' ability to understand engineering drawings. Up to now, researchers have come up with a number of algorithms to generate 3D objects from user-specified orthographic projections (or views). Some excellent surveys are given by Wang and Grinstein [2] and Naendra and Gujar [3]. These algorithms can be categorized into two main types [2–4].

1. Wire-frame-oriented approach or B-rep methods.
2. Volume-oriented approach or CSG methods.

Other recent noteworthy advances are included in Refs. [5–14,24], which mostly focused on the efficiency and practicality of the algorithms. However, while comparable with the ability of human understanding of multiple projections, these algorithms are still in the lab phase, and could not fully satisfy the anticipated practical requirements. These disadvantages mainly embody the following aspects:

1. While multiple interpretations exist, the engineers can make use of its intrinsic shape features and context-dependent knowledge to give a suitable 'conjecture' by reasoning, and obtain the expected interpretation. Most existing algorithms lack this ability. Sometimes there is only one interpretation in an engineer's eyes, but ambiguity will arise in a machine's reconstruction.

* Corresponding author. Tel: +49-224-114-1510; fax: +49-224-114-2449.
E-mail address: weidong.geng@gmd.de (W. Geng).

0010-4485/02/$ - see front matter
PII: S0010-4485(01)00092-6

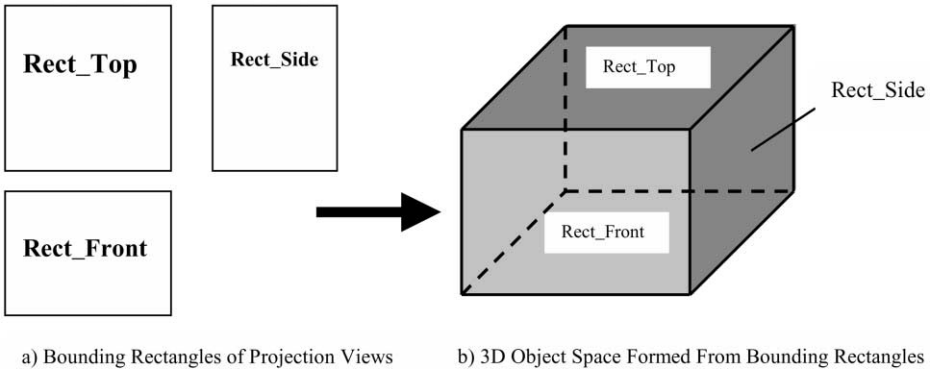


Fig. 1. Definition of 3D object space—S_Box.

2. 3D geometry and topology are usually generated simultaneously when human engineers interpret 2D engineering drawings, but this 3D recovering mechanism is seldom taken into account either in current B-rep or CSG approaches. 'Ghost' elements (some pathological or surplus edges), which make sense in projection but fail to satisfy topological properties of 3D objects, are generated in the wire-frame-oriented/B-rep approach [2,3]. It is not easy to eliminate all of them in complicated cases. The volume-oriented approach assumes that each 3D solid object can be hierarchically built from certain primitives. However, it is difficult to recognize special patterns of primitives from complicated engineering drawings.
3. Section views play an important role in getting rid of the ambiguity of hidden parts in 3D models. However, it is often ignored in existing approaches. In fact, section views are a useful complement of the three-view projections representation, being very efficient in conveying the interior structure of 3D models.

As an attempt to solve these problems, here we propose a 'human-like' multiple-view reconstruction approach, which is a hybrid of wire-frame and volume-oriented scheme for automatic interpretation of engineering drawings. A prototype system, AUTOBUILD, is also presented.

2. Algorithm framework for 3D reconstruction

2.1. Algorithmic description of human engineer's approach

Human engineers usually adopt a 'divide and conquer' strategy to do the interpretation of engineering drawings [15]. This tactic in three-view reconstruction could be summarized as three major steps [16,17]: 'partitioning the object space'; 'extracting the sub-projections'; and 'figuring-out the solid shape'. In the first step, an engineer decomposes the 3D object space (Fig. 1b) in his or her mind based on experience, preference, and background knowledge, and 'mark' their division in multi-view projections using 2D rectangles. That is, the 3D object space is at first divided into a set of 'cell'-boxes that are described by its three-view 2D rectangles (a regular decomposition sample is shown in Fig. 2b). Then, 'extracting the sub-projections' will try to isolate the sub-projection views in related 2D rectangles, purge unrelated drawings, check the consistency, resolve their conflicts, and accordingly get cell-box's sub-projections in each view. In each cell-box, 'figuring-out the solid shape' is to recover 3D geometry and topology from the sub-projection drawings included in the corresponding rectangles, and create its 3D model inside the cell-box. If it is still too complicated to be understood, it could be divided recursively into sub-cell-boxes. Finally, all these 3D models will be combined together into the resulting solid.

From the point of view of 3D modeling, the 3D object space can be defined as a minimal 3D bounding box of the resulting solid. It could be directly formed from bounding rectangles of top, front and side views. In a normal three-view case, let Rect_Top, Rect_Side, Rect_Front, respectively, be the bounding rectangle of top view, side view and front view. Then the 3D object space (abbreviated as S_Box hereafter) is the 3D box formed from the triple rectangles (Rect_Top, Rect_Side, Rect_Front) (Fig. 1).

An example of regular decomposition of 3D object space is shown in Fig. 2b. We can see that each cell-box could be re-projected to the 2D input views and

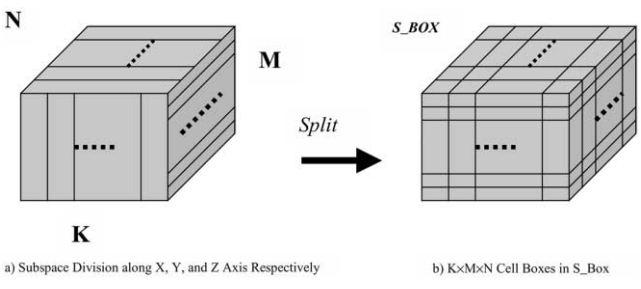


Fig. 2. Subspace division and cell-box partition in S_Box.

generate corresponding 2D rectangles in projection views. This can explain why the human engineer can use 2D rectangles to directly construct cell-boxes and partition the 3D object space.

In 3D modeling, a lot of available methods can be employed to decompose the 3D object space, S_Box, into regular cell-boxes. However, it is observed that partition from human engineers is intentional and smart. The mechanical experts [16,17] say that normally they will try to avoid unnecessary partitions in 3D object space. That is to say, one general principle behind a human engineer's 3D object space division is to generate as few cell-boxes as possible. We can assume that the 3D object space could be respectively divided into K, M and N sub-space along the X, Y and Z axes (Fig. 2a), and $K \times M \times N$ cell-boxes (Fig. 2b) are further generated from them by split operations. Let us see how this minimizes the number of cell-boxes in the 3D object space in mathematics.

Theorem 1. If K, M, N are integers, and $+\infty > K, M, N \geq 1$, then

$$K \times M \times N \geq \text{maximum of } (K, M, N)$$

where $K \times M \times N \geq$ maximum of (K, M, N) if and only if at least two of (K, M, N) are equal to 1.

What does this mathematical conclusion tell us? It obviously says that, if human engineers intend to 'minimize' the number of cell-boxes in S_Box, one reasonable approach is that they would like prefer to partition the 3D object space based on tactics of 'storey by storey' (the 'storey' here is the subspace division along one specific projection direction, shown in Fig. 2a), rather than 'room by room' (the room here is the cell-box in 3D object space, shown in Fig. 2b). That is to say, if human engineers can understand and reconstruct the 3D object in 'storeys', they will not further divide it into 'rooms'. This adaptive tactic is technically sound and consistent with phenomena observed in reality.

In order to show how it works at algorithm level, we give a reasonable non-recursive algorithmic description of the human approach using C-like pseudo code in Appendix A. Understand_Engineering_Draws() describes the main reconstruction pipeline of human engineers. The try-until block clauses mimic human engineers' 'trying to do' activities. Cell_reconstruction()simulates human engineers' basic understanding ability of engineering drawings. The core activities of Partition&Reconstruction() and Further_Partition&Reconstruction() are both to partition the 3D object space/subspace into cell-boxes and reconstruct the 3D objects in them. They are given separately here because the non-recursive description is expected. However, it should be addressed here that the pseudo code in Appendix A is just to explain the main idea of the human approach. It does not mean that human engineers always fulfill reconstruction task exactly that way.

2.2. Algorithm idea

According to the pseudo code in Appendix A, the key issues in mimicking human interpretation approaches lie in the following aspects,

1. How to 'divide' S_Box. The 3D object space partition should make sense by simplifying the 3D object in cell-boxes, as one obvious purpose of division is to make sure that the 3D solid in the cell-box can be reconstructed more easily than that in S_Box. Human engineers can accomplish the object space division easily, as they have sophisticated intelligence including commonsense, experience, and learning skills. Is it possible to find a general and domain-independent way to partition the 3D object space, and can the computer do this automatically?
2. How to 'conquer' the interpretation in cell-boxes. Human engineers are very smart in interpreting engineering drawings, as they can take non-geometric constraints (commonsense, experience, background knowledge, etc.) into consideration. Can we find such an effective algorithmic mechanism to do Cell_Reconstruction() similarly? How to design a better algorithmic mechanism that could well simulate the interpretation ability of human engineers? Fortunately a lot of work [1–14,18–24] about 3D reconstruction from multi-views has been done so far. They will help us find reasonable solutions for these crucial points in simulation of the human interpretation approach.

The major principle behind the object space partition is that the 3D object in the cell-box should be simpler than that in the overall S_Box, and can be more easily reconstructed. According to the pseudo code in Appendix A, the key point of S_Box partition is how to automatically figure out X_Partition, Y_Partition, and Z_Partition by computer. Comparing Fig. 2a with Fig. 1b, we find that the subspace division in 3D object space, S_Box, can be converted into partition of the bounding rectangles, Rect_Top, Rect_Front and Rect_Side, of the partition.

Considering a typical bottom-up wire-frame/B-rep pipeline shown Fig. 3, it starts from the 2D points in projections, and ends with the 3D object. It obviously shows that the complexity of the resulting 3D object is mostly controlled by 2D points

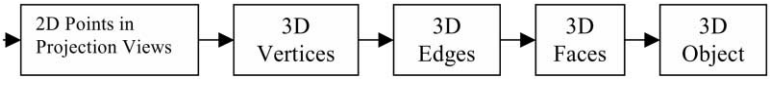


Fig. 3. A typical pipeline of wire-frame/B-rep approach.

in projection views. That is to say, the 3D object in the cell-box will probably be simplified if we try to decompose the object space by 2D points in projection views. We manage to design such an automatic sub-space division method according to all 2D points in multi-view projections. Its major steps of how to generate X_Partition, Y_Partition, and Z_Partition are as follows.

Algorithm for generating X_Partition, Y_Partition and Z_Partition.

1. Collect all 2D points in projection views, and form a set of these points, Point_Set.
2. For each element in Point_Set: extract the X component (if it exists) of its coordinates, and insert it into one specific set, X_Set = {a|a is X component of coordinates among all 2D points in projection views}. Therefore, we obtain X_Set in which the X components of all the2D points in projection views are included. We can respectively generate the Y_Set = {b|b is Y component of coordinates among all 2D points in projection views} and Z_Set = {c|c is Z component of coordinates among all 2D points in projection views} in a similar way.
3. Sort and purge the elements in X_Set (no duplications are allowed), then create a new set of halfspace descriptions (plane equations), X_Split = {X = k|k ∈ X_Set}, whose elements accordingly correspond to each element in X_Set. It is obvious that X_Split will form a subspace partition of the 3D Object space along the X axis.
4. Similarly, we can sort and purge Y_Set and Z_Set, and accordingly create the halfspace description set, Y_Split = {Y = m|m ∈ Y_Set}, and Z_Split = {Y = m|m ∈ Z_Set}.
5. Create the subspace description along the *X*, *Y*, *Z* axes, X_Partition, Y_Partition, Z_Partition, on the basis of X_Split, Y_Split and Z_Split.

About how to 'conquer' the interpretation in cell-box, one of its key points is how to effectively make use of the specialized features of cell primitives, and take more constraints into account in Cell_Reconstruction(). One significant specialty of cell primitives comes from the assumption that the 3D object in the cell-box is simpler than that in the overall S_Box. It means that we can give some general prerequisite restrictions to describe 'how simple is simple' for 3D objects in cell-boxes, and incorporate some commonsense constraints into reconstructing them. We learn from the CSG approach that a usual way to include additional constraints into interpreting engineering drawings is to assign primitive types such as box and cylinder to the 3D objects in cell-boxes. In our case, the S_Box has already been divided into cell-boxes, if we can specify types for cell primitives, which can describe the 3D object in almost any cell-box, our reconstruction algorithm will be able to take 'extra' non-geometric constraints into consideration.

At the algorithm level, the sub-task of understanding engineering views in a cell-box is to restore geometric and topologic information lost during the process of projection from 3D object to 2D. Human engineers are often reported to simultaneously recover 3D geometry and topology from projection views, instead of B-rep's using geometric information to decide topologic information in a 3D object, which might be the main cause of generating 'ghost' elements in the B-rep approach. Can we find a new human-like mechanism to create the 3D model like this?

Considering modeling operations in 3D solid modeling [22], we find that the translational sweeping operations have the advantage of 'simultaneously' generating 3D geometry and topology based on its 2D baseface. So, in Fig. 4 we propose such a reconstruction pipeline for cell primitives. 'Ghost' elements might be greatly reduced, as its 3D model creation makes full use of 2D topological information (loops) in input views by employing general translational sweeping.

Therefore, we combine the aforementioned requirements together, and fortunately 'discover' such a primitive type, **cell primitive**, which is defined as its 3D object that could be generated by general translational sweeping operation. Its formal definition is given in Section 2.3.

After the core issues of how to make the algorithm work have been resolved, the rest is how to improve the algorithm performance including efficiency, robustness and reliability, etc. On the basis of the pseudo code in Appendix A, the following interaction options are further added to optimize algorithm performance, although the algorithm itself can work automatically.

1. Explicit order/sequence specification of **try**-clauses. The user can explicitly change the default sequence of **try**-clauses by interactively specifying main/secondary

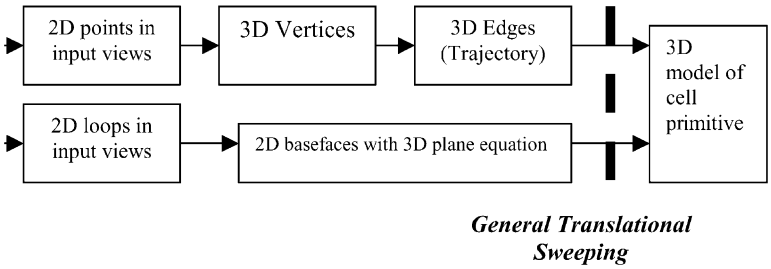


Fig. 4. Cell primitive reconstruction.

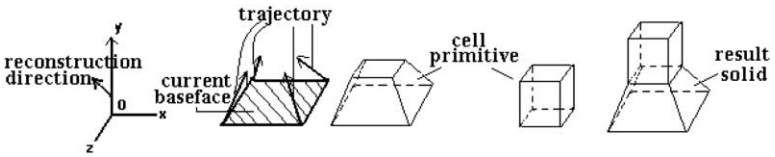


Fig. 5. Definitions of some terminologies

reconstruction direction. From the pseudo code in Appendix A we see that on average three **try**-clauses will be executed for a fixed order/sequence of **try**-clauses. However, if the user can re-define the order of **try**-clauses for each individual task, one trial will usually lead to the resulting interpretation. This will significantly improve algorithm efficiency.

2. 'Smart' partition. The user is allowed to interactively partition the3D object space by drawing 2D rectangles in input views, and related cell-boxes will be constructed directly from these 2D rectangles. It is very helpful to deal with some special cases in the partition of the 3D object space (see discussion in Section 7).
3. Result verification when ambiguity exists. When ambiguity arises during interpreting cell primitives, the user can confirm or rectify the most possible interpretation by answering 'Yes/No' questions, or directly picking up the desired result from a list of candidates. This makes the result be more reliable, and ensure that the resulting solid will be more consistent with a human engineer's interpretations.

In addition to the aforementioned interactive options, section-views processing and visual reasoning, which are especially designed for dealing with multiple interpretations, are also introduced in our algorithm. The reconstruction pipeline and major algorithm steps are described in the next section.

2.3. Algorithm pipeline and major steps

The terminology in 3D reconstruction from multi-view task varies among different papers. Some terminologies related to our approach are first defined here (shown in Fig. 5).

- *Cell primitive*—Cell primitive refers to the 3D object in the cell-box. Here, it is further specified as the 3D solid that could be generated by a general translational sweeping modeling operation.
- *Baseface*—A baseface is the bottom face of the cell primitive that could be swept translationally to construct a 3D model of the cell primitive. It is a 2D face that lies in a 3D plane.
- *Trajectory*—The trajectory is a 3D edge vector that defines the sweeping movement of every vertex in the baseface.
- *Reconstruction direction*—It is a normalized vector that indicates the translational sweeping direction. The potential reconstruction directions are input views' projection direction and its opposite. It can be interactively specified by the user.
- *Horizontal plane* (HP)—A horizontal plane is a projection whose view direction is parallel to the reconstruction direction.
- *Vertical plane* (VP)—Vertical planes are the projections whose view directions are perpendicular to the reconstruction definition.
- *Reconstruction context*—It is a composite data structure that describes all the related information related to the current interpretation task, including reconstruction direction, HP, VPs, section views, half-space partition, current baseface and the 3D model of the cell primitives.

An overall interpretation process to carry out the algorithm idea in Section 2.2 is shown in Fig. 6. It is based on our previous work on multi-view reconstruction [23].

Considering the characteristics of a general translational sweeping operation, the major steps of 3D reconstruction in AUTOBUILD are given as follows.

Step 1 Reconstruction direction is optionally picked up from positive or negative projection directions of all input views. A secondary reconstruction direction can also be specified if necessary. A default one is the negative projection direction of the top view. The

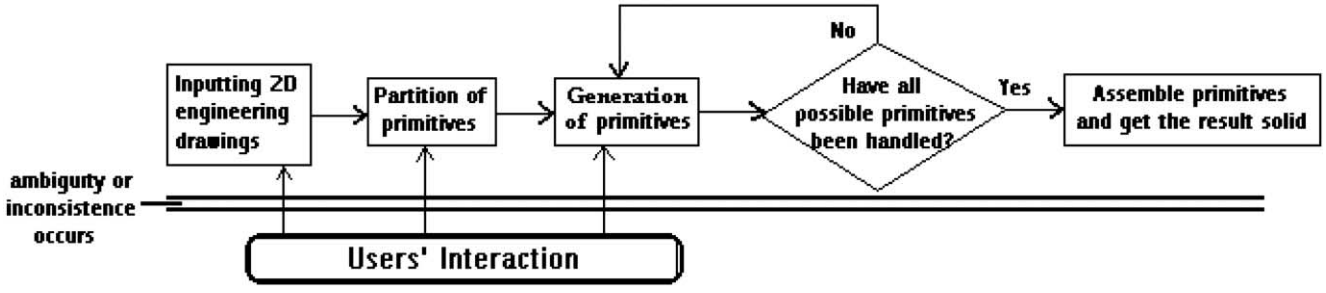


Fig. 6. An overall pipeline of reconstruction in AUTOBUILD.

system will try to do its interpretation task first from this reconstruction direction. In the extreme case, all six potential reconstruction directions will be put into trial by the system.

Step 2 According to the reconstruction direction and the 2D points in the relevant views, the 3D object space is divided into cell-boxes, and in each cell-box its sub-projections are extracted from the source input views. The cell-boxes can also be directly created from 2D rectangles drawn in engineering views by the user.

Step 3 In each cell-box, the pipeline in Fig. 4 is employed to recover geometric and topologic information of cell primitives. Its corresponding baseface and trajectories of each vertex in the baseface are reconstructed from multiple subviews, and then the general translational sweeping operation is performed to generate its 3D model. If there are ambiguities or inconsistencies, section view processing and visual reasoning will be activated to help select a reasonable result. Users are also permitted to interactively eliminate unsuitable interpretations if the system could not solve these ambiguities automatically.

Step 4 After 3D models of all possible primitives are generated, the assembly operations, i.e. face-glue, Union, etc., are executed to compose the resulting solid step by step. The resulting solid will be checked to see whether it could interpret the input views consistently and completely. If true, the resulting model is successfully returned as both CSG tree and boundary representation based on a half-edge data structure [22], otherwise, the user is required to intervene in the interpretation process, and re-interpret it again in an interactive manner.

In the rest of the paper, the partition of the 3D object space will be discussed in Section 3, and Section 4 covers the creation of a 3D model for cell primitives, including generation of baseface and calculation of sweeping trajectory. The interpretation of section views and visual reasoning engine are discussed in Sections 5 and 6, respectively, and finally some discussions and future work are presented.

3. The partition of 3D object space

According to the pseudo code in Appendix A, the major task of 3D object space partition is to figure out X_Partition, Y_Partition and Z_Partition, and isolate cell primitives' sub-projections by cell-boxes. The assumption behind our space division is that the complexity of the3D resulting object is mainly dependent on the 2D points in projection views. Of course, we can directly utilize the partition algorithmic steps in Section 2.2 to decompose the 3D object space. Unfortunately, in most cases it is redundancy to generate all subspace partitions in advance. In AUTOBUILD, the 3D object space decomposition is refined by reconstruction direction, and classified as two main cases: single-direction partition and multi-direction partition.

The single direction partition works as follows.

P1: All points in VPs are sorted incrementally according to their vertical (height) component. This is similar to Y-Bucket sorting, and height layers along the reconstruction direction are generated, as shown in Fig. 7.

P2: At each height in sorted bucket, plane equations are created by setting its normal as the reconstruction direction. These planar surfaces will split the 3D object space into subspaces along the reconstruction direction, and the loops contained in the VPs will be further checked to merge some adjoining subspaces, and accordingly form a more suitable half-space description.

P3: Isolate the sub-projections, purge unrelated drawings and check the consistency. Finally, a reconstruction context is created to store the 3D object space partition including split-plane equations, subviews of HP and VPs, etc.

This partition process can automatically be accomplished in AUTOBUILD. The cell primitives of Fig. 7 are shown in Fig. 8. Its reconstruction direction is $+y$. HP is the x–z

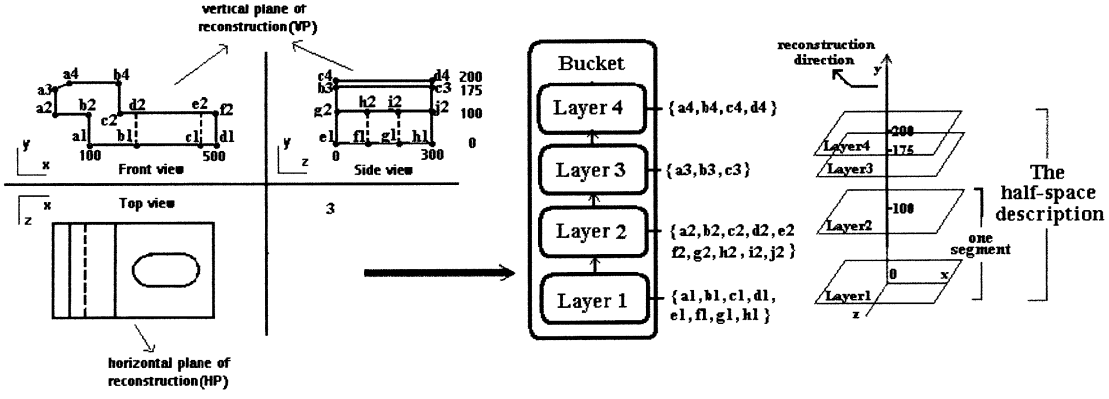


Fig. 7. Generation of half-space description.

plane. Its partition consists of three subspaces, i.e. [0, 100], [100, 175], [175, 200], along the $+y$ direction.

If one direction is not sufficient to clearly and completely partition the resulting solid into cell primitives, multiple reconstruction directions will be specified to split the 3D object space, and the system will try to figure out their corresponding subspaces based on the loops information in projections. Along each reconstruction direction, the subspaces are further partitioned in the same way as that in single direction partition. The system will isolate the subdivided projection drawings by different reconstruction directions and create a root context for multi-direction partition. The related reconstruction sub-contexts are created for each reconstruction direction. The interpretation processes will be recursively performed in reconstruction sub-contexts, and the resulting models are a 'union' of 3D objects reconstructed from reconstruction sub-contexts. Fig. 9 is an example of multi-direction partition.

If the user interactively specifies the 3D object space partition by 2D rectangles, then cell-boxes will be directly built from these 2D rectangles, and accordingly generate the reconstruction context.

4. 3D model creation for cell primitives

According to the definition of cell primitives and the pipeline in Fig. 4's, if the baseface has been formed and the trajectory of each vertex in the baseface has been determined, the 3D model of the cell primitive could be directly created by a general translating sweeping operation. Its major steps are as follows.

C1: Create the baseface of the cell primitive from the loops in the HP and bounding information of VPs.

C2: Calculate the trajectory of each vertex in the baseface on the basis of HP and VPs.

C3: Generate the 3D model of the cell primitive by general translational sweeping operation.

Steps 1–3 will be repeated in each cell-box until all 3D models of primitives are created.

4.1. Generation of baseface

In general, the baseface of cell primitives consists of one out-loop and a group of inner-loops. The out-loop describes the boundary of the baseface, and the inner-loops form its inner holes. We must identify these related loops, and accordingly form the baseface from them. In AUTOBUILD, its algorithm steps are as follows.

1. Based on the boundary vertices of VPs in the cell-box, a bounding rectangle for candidate outer-loops of current baseface is created in the HP.
2. The algorithm in Ref. [11] is employed here to look for all loops in HP. The loops whose bounding rectangles are the same as the current bounding rectangle are picked out as the candidate out-loops. These candidate out-loops are further checked with the constraint of VPs. All unsuitable loops will be discarded.
3. For each candidate out-loop, the loops inside it are all selected as candidates for inner-loops. The candidate inner-loops will also be similarly checked by constraints from VPs.
4. The baseface is initially constructed from the out-loop by Euler operation [22], and the holes in the baseface are further created according to the inner-loops.

These steps are illustrated in Fig. 10 by four phases.

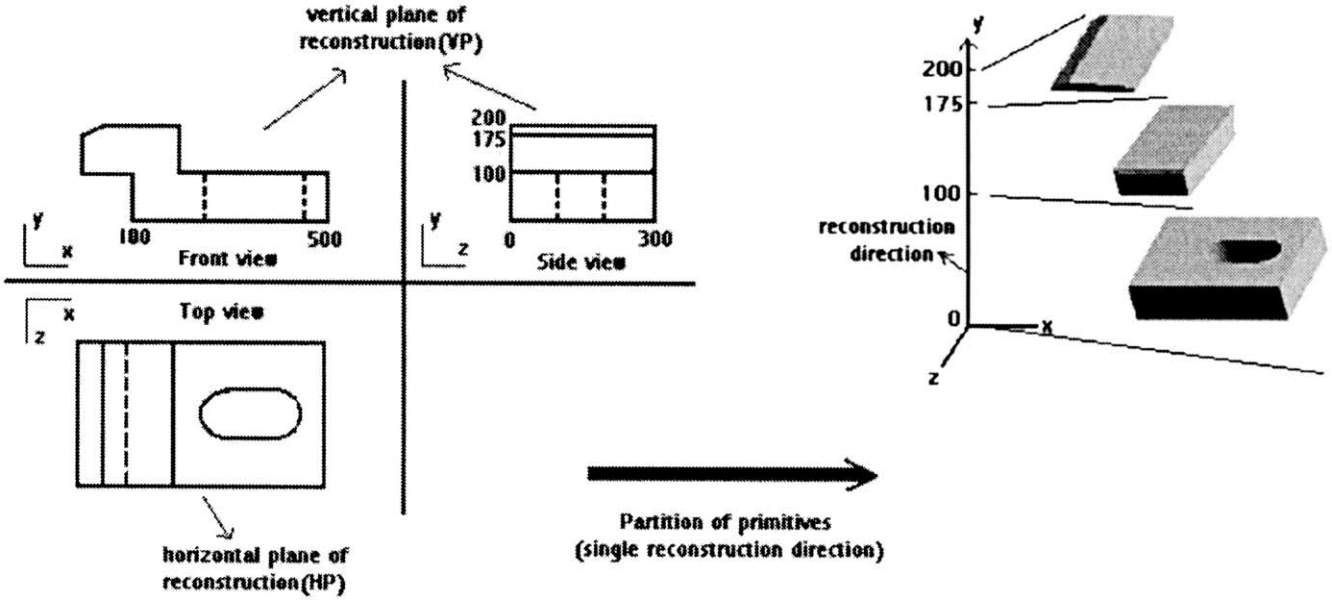


Fig. 8. Partition by single direction.

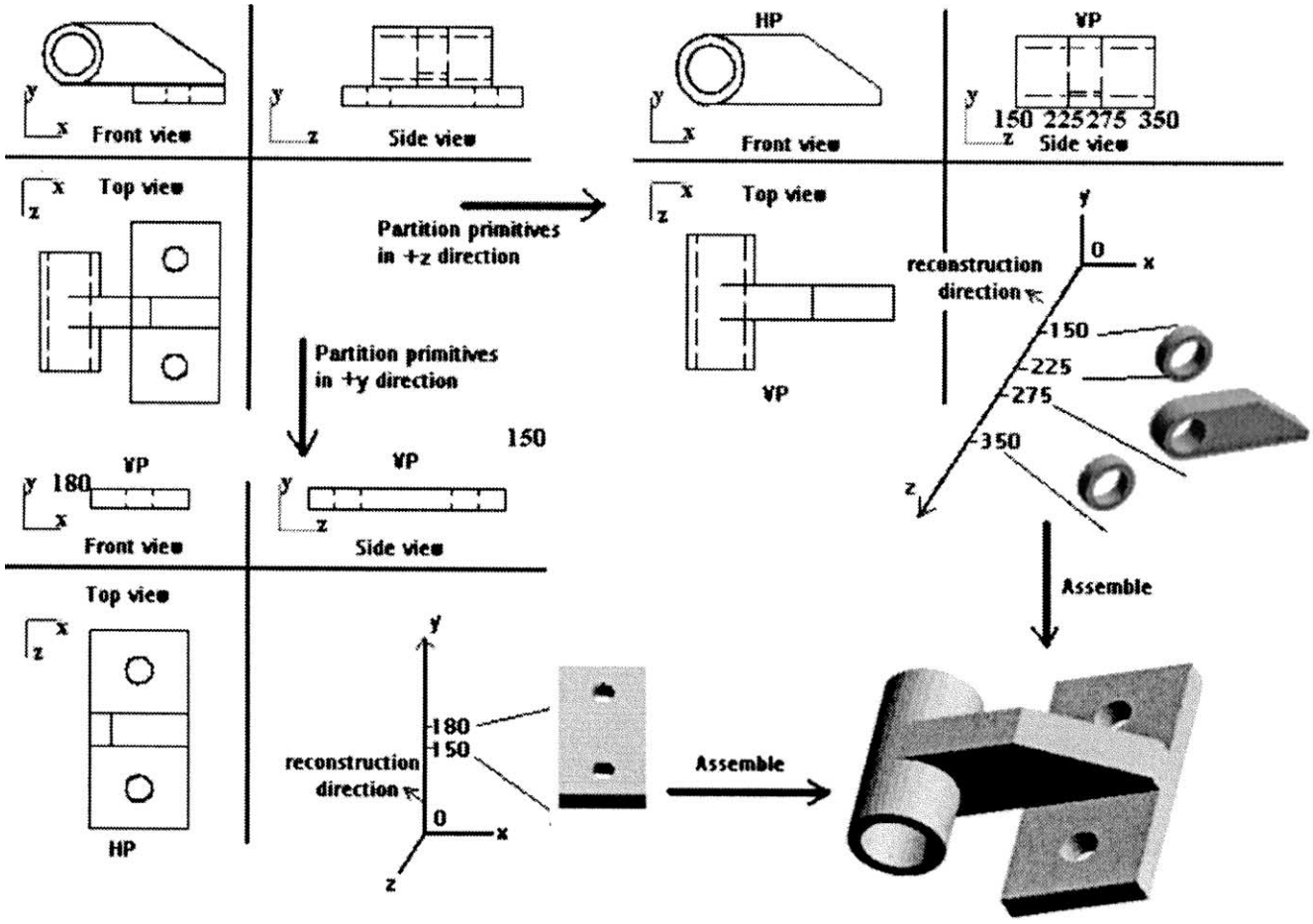


Fig. 9. Partition by multiple directions.

Normally, the baseface will be acquired uniquely after these steps. However, ambiguity sometimes appears during this process, and then section views processing and visual reasoning (Sections 5 and 6) will be fired to deal with them. In some complicated steps, users are also required to verify the resulting loops for the baseface.

In general, this algorithm works effectively, and the baseface could be directly obtained from the HP. However, the 3D object space partition algorithm does not guarantee that the loops of the baseface could always be directly found in HP. Fortunately, it could be reasonably assumed that at least one cell primitive's baseface can be formed directly from the loops in the HP at the beginning of interpretation. After the first primitive is reconstructed, its top face will be re-projected to the HP, and create 'virtual' loops in the HP to ensure that the baseface of the next primitive could be successfully generated from the HP. In AUTOBUILD, three procedures are respectively developed to deal with different cases in constructing the baseface. 'FindStartFace' is for the case that the loops matched with the baseface of the current cell primitive are all contained in the HP. 'FindNextFace' is to find the baseface of the next cell primitive after the top faces of the previously reconstructed cell primitives have been re-projected to the HP. 'FindMidFace' handles the other special cases in forming the baseface.

4.2. Calculation of sweeping trajectory for baseface

From Fig. 4, we see that most B-rep algorithms could be employed here to calculate the sweeping trajectory for vertices in the baseface. The relevant algorithms can be found in Refs. [1–3]. In AUTOBUILD, all possible 3D edges that could interpret vertices in the baseface are specified as candidate sweeping trajectories (the curves need to be handled specially here). Then, unsuitable trajectories will be eliminated step by step based on the qualitative analysis of the attributes (visibility, etc.) of the vertex in the baseface and the modeling principles of manifold solid [22]. The trajectory calculation steps in AUTOBUILD are as follows.

1. All 2D edges of VPs in the current cell-box are checked to find all 2D candidate edges corresponding to vertices in the baseface.
2. The 2D candidate edges are purged according to some basic rules. For example, if a vertex in the baseface is visible in a VP, then 2D edges that interpret it in this VP

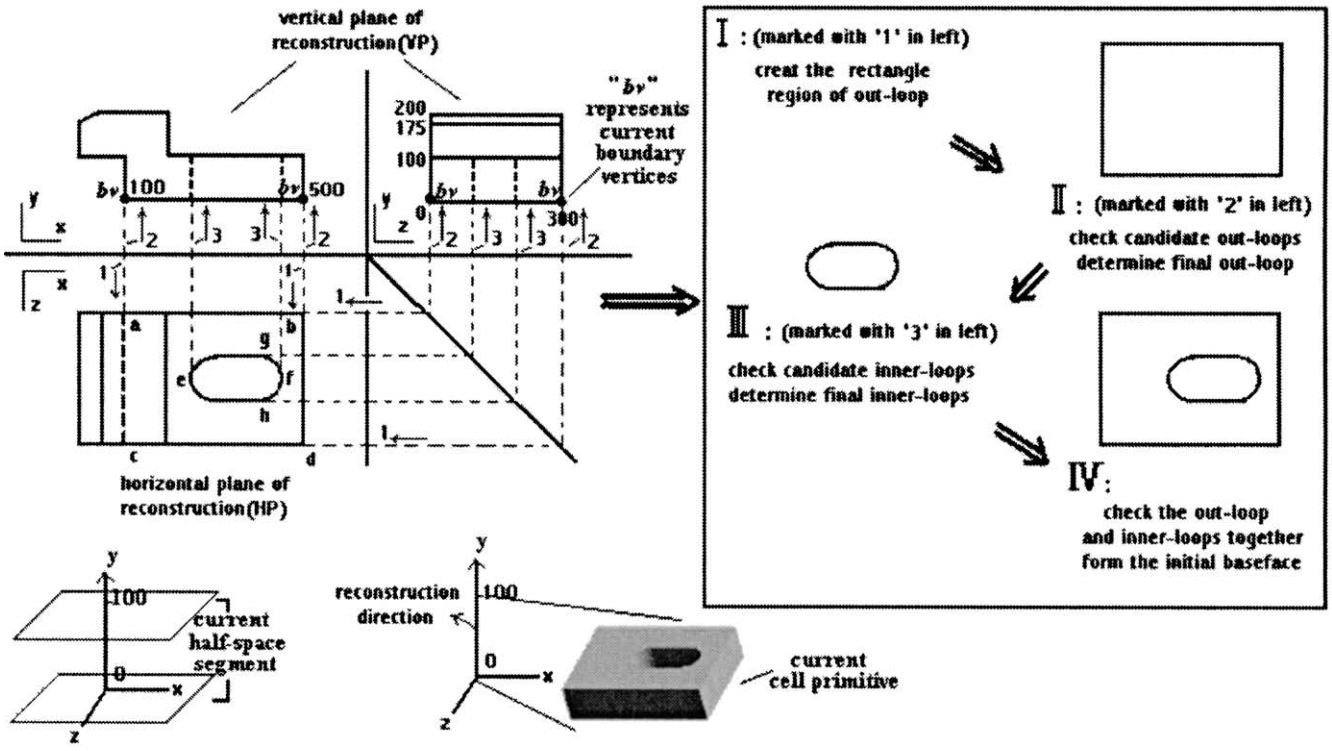


Fig. 10. Generation of the baseface.

should also be visible. So, the dashed candidate edges in this VP should be deleted from the 2D candidate edges list.

3. 3D vertices are calculated from triplets of end points of 2D candidate edges. If the coordinates of 2D points in the front, top and side views are $P_f(x_f, y_f)$, $P_t(x_t, z_t)$, $P_s(y_s, z_s)$, respectively, and they satisfy $x_f = x_t$, $y_f = y_s$, $z_t = z_s$, then the 3D vertex is (x_f, y_f, z_t). 3D edges are further generated by examining the connectivity of two 3D vertices.
4. The curves in the baseface are tackled specially. At present AUTOBUILD allows circles and arcs to appear in input views. These curves should be kept as a whole in computing the sweeping trajectory. However, AUTOBUILD's translational sweeping operation is performed on the half-edge data structure [22], in which circles and arcs have been converted into discrete edge segments. So, some necessary assemble operation is needed to revert them to their original type during trajectory calculation. If other types of curves exist in the baseface, they should be handled in a similar way.
5. Post-processing. Normally, only one sweeping trajectory is permitted to be assigned to one specific vertex in the baseface. However, sometimes more than one trajectory can be potentially assigned to the same vertex in the baseface, or there is no corresponding trajectory to it at all. In these cases, we should split the vertex in the baseface by inserting 'null' vertices, or 'merge' it to its neighboring vertices during sweeping. The user could also interactively decide the corresponding trajectory if the automatic splitting or merging fails.

An example of trajectory calculation is shown in Fig. 11. Fig. 11a is the input three-view drawing. Its baseface is shown in Fig. 11b. In front view, both L1 and L2 could interpret the vertex e or f in the baseface, but according to the rules in step 2, it is obvious that only L2 can interpret vertex f. C1 and g–h in the baseface are treated as circles and arcs. Post-processing is also needed here. The resulting solid is shown in Fig. 11c.

4.3. A typical reconstruction example

As a brief summary of this section, a typical interpretation process is shown in Fig. 12.

5. The interpretation of section view

From a pragmatic viewpoint, an effective mechanism of coping with multiple interpretation is necessary, as multi-view projections are imperfect in conveying 3D models. Information is somewhat lost when 3D objects are projected into engineering drawings. Although the invisible features of a simple object usually may be described on a three-view drawing by the use of hidden lines, it is unwise to depend on such a perplexing representation to describe the interior of a

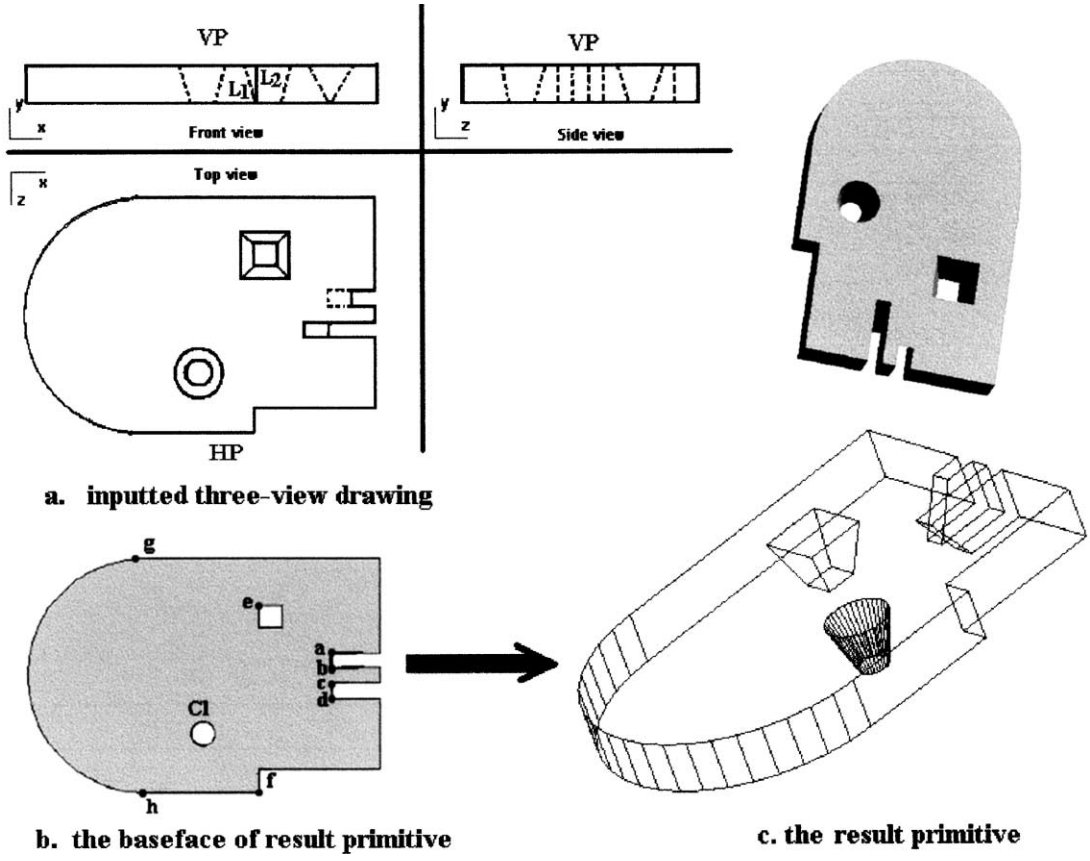


Fig. 11. Sweeping trajectory calculation for the baseface.

complicated object. At this point, the section view plays an important role in representing the internal structure and eliminating the ambiguities. So, it is reasonable to introduce the section view into the interpretation process and enhance the capability of handling ambiguities in 3D reconstruction.

From the standpoint of interpreting engineering drawings by machine, the section view is different from the orthographic projection view. It indicates the critical cues for interior shape structure, and the shape patterns presented in a section view are somewhat more important than pure geometric constraints. Normally, their interpretation is based on the partial 3D model. That is to say, the related 3D models are needed while interpreting the section view. In AUTOBUILD, section views are handled as an additional virtual partition of 3D object space. This makes section views be 'naturally' included in our human-like approach.

For the time being, AUTOBUILD only has the ability of processing full sections (one main type of section view), which includes two main types: longitudinal section and crosswise section. Longitudinal section can help check candidate loops in baseface generation, or confirm the trajectory selection. As shown in Fig. 13a, the shape patterns are employed to rectify the inner loops (holes in the solid). Its processing steps are as follows.

1. The section views are stored together with orthographic views in the reconstruction context, and registered as a virtual partition with the cell-boxes that it passes through.
2. If ambiguity arises during the construction of the baseface, shape patterns extracted from the section view will help select candidate loops. The boundary shapes are contributory to picking up out-loops, and non-boundary patterns are helpful to match the holes in the baseface.
3. If multiple interpretations exist in the trajectory calculation of the baseface, all possible 3D models from the multi-interpretation are generated first, then a virtual plane (cut-face) based on the section view will split these candidate 3D models in the current cell-box, and compare the shape patterns in the cut-face with that in the section views. 3D candidate models which do not match the shape patterns in section views will be thrown away.

In crosswise section, the cut-face in the section view usually indicates the contour of the baseface. Shape patterns such as loops can be directly used to determine candidate loops of the baseface (the candidate loops of the baseface shown in Fig. 13b are shown in Fig. 14). In AUTOBUILD, it works as follows: at first, the relationship of crosswise section views and the current cell-box is inspected. If the section view and the baseface are in the same 3D plane, then the loops in the section views will be utilized to identify the candidate loops for the

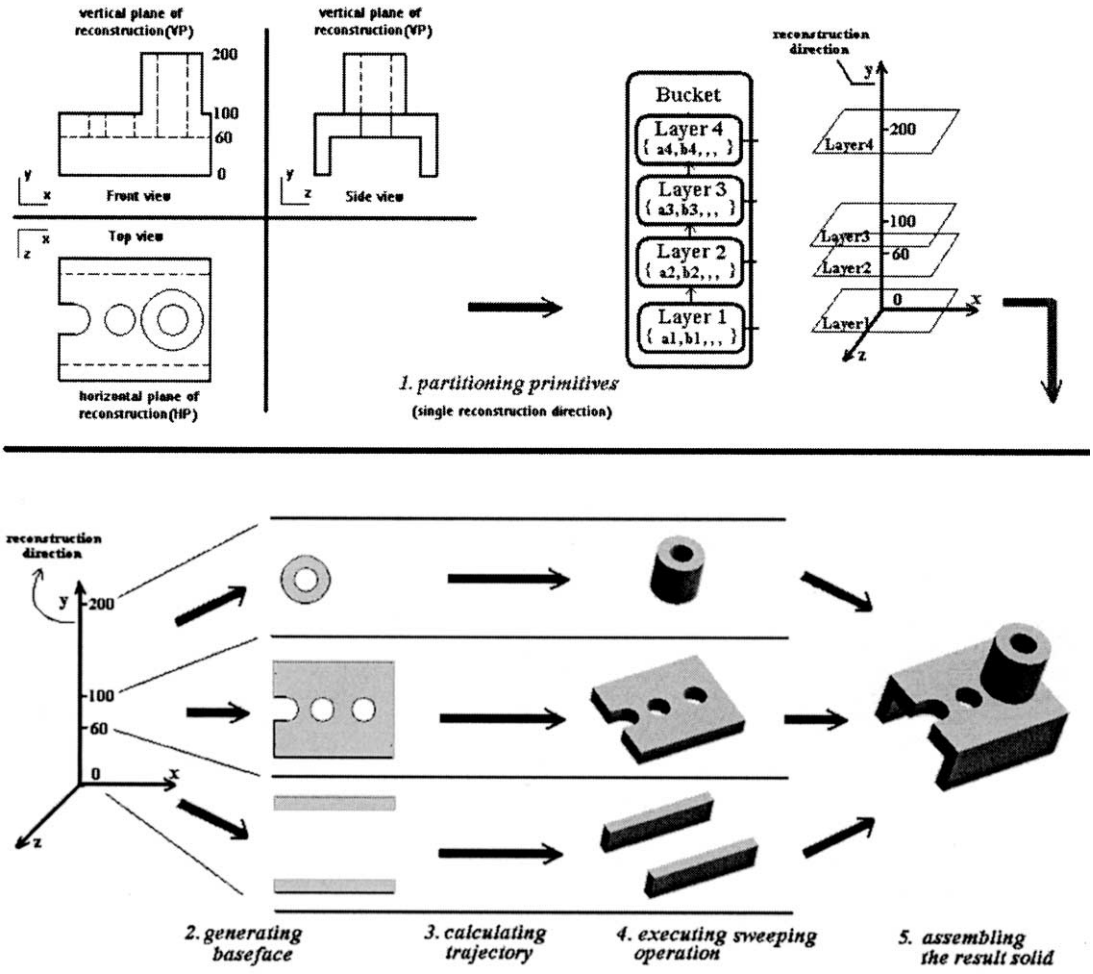


Fig. 12. A typical reconstruction process.

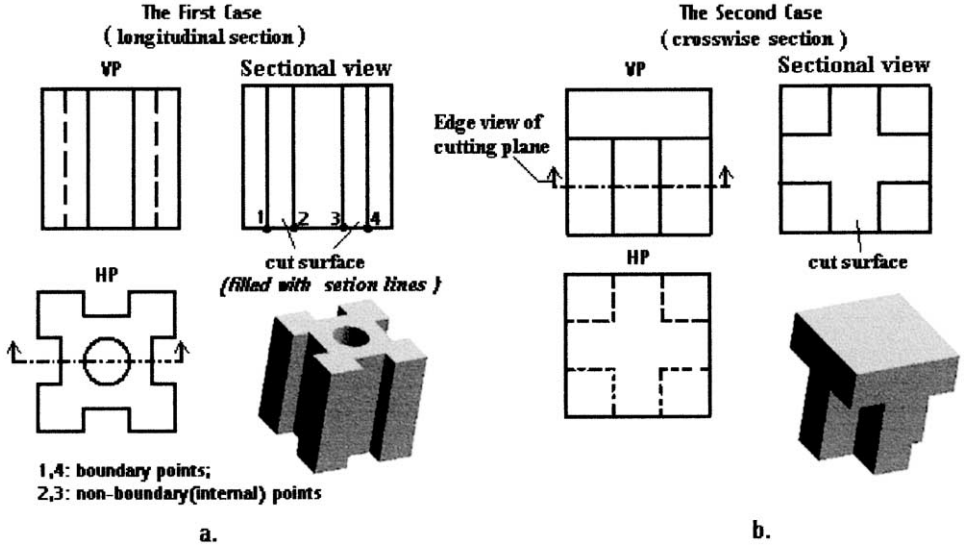


Fig. 13. Section view processing.

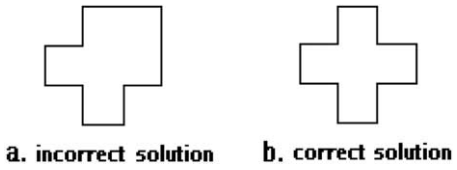


Fig. 14. Candidate basefaces.

out-loop and inner-loops of the baseface. If they are not in the same plane, the intersection points between section planes and the candidate trajectories are computed. Those candidate trajectories whose intersection points collide with that in section views will be considered as 'unsuitable' ones. It is also useful to decide the mapping between candidate trajectories and vertices in the baseface.

6. The visual reasoning engine

Visual reasoning is designed to help the user deal with multiple interpretations. It makes the interpretation generated by computer mostly be the same as that from human engineers. It is based on the following facts: experienced engineers sometimes can understand engineering drawings although there are ambiguities; they can speculate the most probable results based on their experience including the principles of visual cognition, commonsense, and background knowledge. The visual reasoning engine is proposed here to simulate this capability of experienced engineers, and the desire to enable the computer have similar ability to that of human engineers.

The visual cognition principles are an abstract description of the psychological mechanism by which humans fulfill visual cognitive tasks such as understanding and recognizing visual shapes. In AUTOBUILD, three well-known cognitive principles derived from the Gestalt Psychology are employed to determine which one has the maximum possibility when multiple interpretations are generated.

1. *The principle of symmetry* (SYP): the candidate shape which includes most symmetry will have more possibility of becoming the anticipated interpretation.
2. *The principle of tendency* (TP): the candidate shape which could continue the movement tendency indicated by most sweeping trajectories will become the most possible interpretation.
3. *The principle of simplicity* (SIP): the shape which is the simplest among the candidates will have the maximum possibility as expected interpretation.

In AUTOBUILD, SYP and SIP principles are mainly applied to evaluate the candidate loops of the baseface and directly pick out the most probable interpretation. The 'symmetry' and 'simplicity' of candidate loops are calculated and sorted in descending order. For each candidate loop, let I_{sym} and I_{sim} be the sorted index of 'symmetry' and 'simplicity', respectively, W_{sym} and W_{sim} the weights of 'symmetry' and 'simplicity', are specified by the users. Then its synthesized possibility I_{ss} is $(W_{sym}/I_{sym} + W_{sim}/I_{sim})$ The candidate loop, which has the largest I_{ss}, will be chosen as the most probable anticipated loop. The pseudo code to calculate the 'symmetry' and 'simplicity' of loops are given in Appendix B.

The TP principle is helpful to get rid of multiple mappings between candidate trajectories and vertices in the baseface, and select the one that best keeps the movement tendency of the trajectory. The intersection angle between the vector of the previous trajectory and the current candidate are calculated as the weight of keeping movement tendency. The candidate trajectory, whose intersection angle is smallest, will be selected as the most probable trajectory. An example of applying visual reasoning is shown in Fig. 15.

7. Discussion and conclusion

We have implemented the prototype system, AUTOBUILD, using VisualC/C++ 6.0 in MS-Windows 98/NT environment, in which we successfully simulate human engineers' 'divide and conquer' strategy of 3D reconstruction from multiple views. Its resulting solids are represented as both CSG tree and boundary representation on half-edge data structure [22]. As box, cylinder, cone, sphere could be generated by general translational sweeping operations, currently, AUTOBUILD has the ability of recovering a 3D object which is composed of planar (polyhedral), cylindrical, conical, or spherical surfaces.

From the point of view of algorithm, we successfully bring B-rep and CSG approaches together in context of 3D reconstruction from multiple views. In such an integrated solution, our major contributions are as follows.

1. We present a general way to partition the resulting solid into primitives, instead of current CSG approaches, recognizing primitives from projections. It is obvious that primitive partition is easier to be realized than primitive recognition.
2. We propose an innovative sweep-based 3D model creation method that makes full use of 2D topology (loops) explicitly represented in input views, and directly 'generate' 3D topology in the 3D object, instead of the existing B-rep approach, using geometry to decide topology. It significantly reduces 'ghost' elements. Even if the 'ghost' elements do exist, the principles from existing B-rep algorithms are more effective to eliminate them, as cell primitives here are relatively simple.
3. One significant 'by-product' of our approach is that processing of section views are easily and naturally incorporated into the reconstruction process by treating

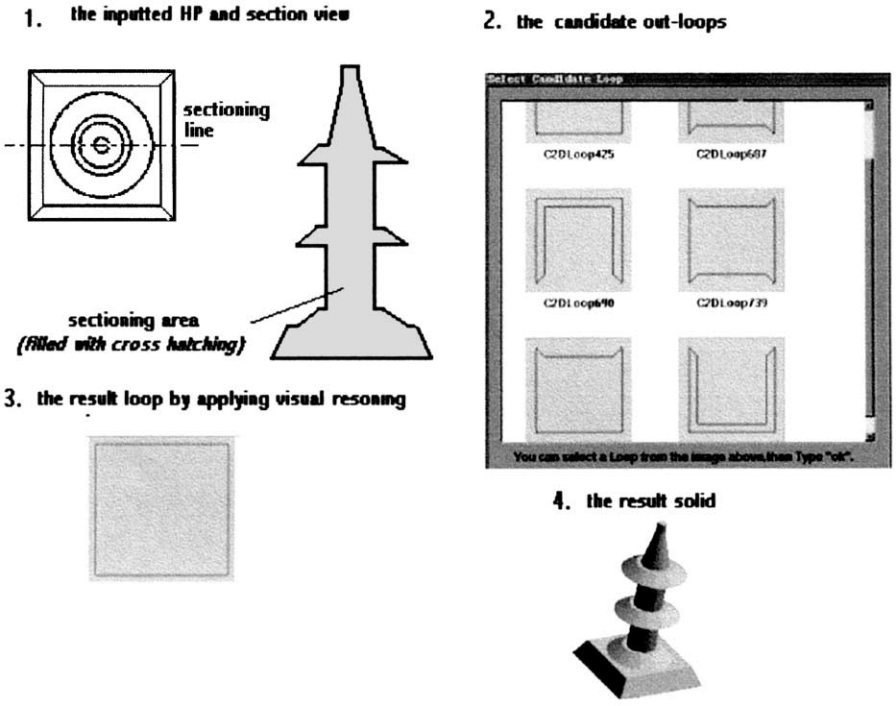


Fig. 15. Reasoning with visual cognition.

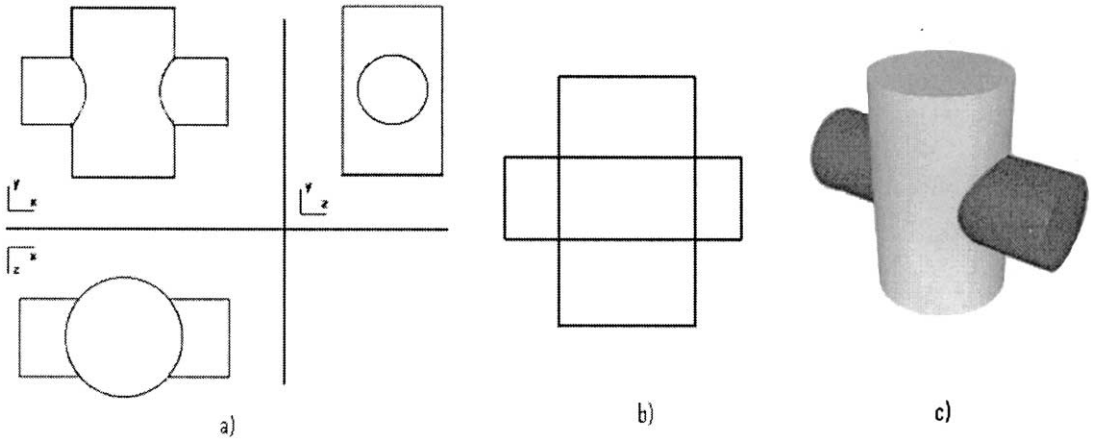


Fig. 16. Two cross-intersected cylinders.

section view as an additional virtual partition of the 3D object space.

4. Visual reasoning engine is introduced to cope with the problem of multiple interpretations in multi-view 3D reconstruction.
5. At the algorithm implementation level we designed a human–machine cooperation mechanism in which a natural and convenient interaction is provided. The standpoint to let the reconstruction process be open to human engineers is that, with some potential assistance from the user, the system can to some extent individually incorporate domain knowledge and commonsense into its current reconstruction task. It can also compulsorily correct any inconsistency that appears in the projection views. It is very useful to help achieve the main objective of enabling the computer to have human engineers' ability to understand engineering drawings.

The major limitations of our approach include the following.

1. The 3D object space partition is based on half-space description, and normally no superposition is allowed between cell-boxes. Unfortunately, such partition means does not always 'reduce' the complexity of the 3D object in a cell-box for a few curvilinear objects such as the cross-intersection of two cylinders (shown in Fig. 16a). Our

current solution is to let the user interactively draw two intersecting 2D rectangles (Fig. 16b) in projection views, and directly create two intersecting cell-boxes for them. The resulting solid is still the union of 3D models in cell-boxes (Fig.16c).

2. For the time being, only one primitive type is allowed in a cell-box, and it sometimes is not sufficient to describe all 3D objects in cell-boxes. For example, the torus, which is difficult to generate by general translational sweeping operations, is excluded for now. A potential solution is to include more primitive types. That is, Cell_Reconstruction() can try to re-interpret the 3D object in a cell-box according to another primitive type if the interpretation based on the current primitive type fails to get a result. However, such a re-interpretation will sacrifice algorithm efficiency. In our current implementation, we do little for this, as we have not yet found an acceptable domain-independent tradeoff between algorithm efficiency and reconstruction ability.

The future work we are planning to do is to 'customize'/'specialize' this approach according to a specific application domain. The continuing extension work includes the following.

1. 'Intelligent' partition and specification of reconstruction direction. The first step is to try to automatically generate intersecting cell-boxes from some predefined subviews by similar extrusion techniques [24]. The other is to re-represent cell-box partitions of the 3D object space by binary tree data structure, and try to heuristically search the most economical path of S_Box division by mature AI techniques such as backtracking and branch pruning.
2. Enhancing the interpretation ability of Cell_Reconstruction(). The overall reconstruction ability is heavily dependent on the interpretation ability of Cell_Reconstruction(). More curvilinear primitive types such as torus, which do make sense to one specific domain, are desired to be included in Cell_Reconstruction(). The system will automatically try to re-interpret the 3D object in cell-boxes according to these new primitive types after its first try of interpretation is failed. Some related work [6,10] can also be integrated into the reconstruction of curvilinear objects.
3. Powerful mechanism of handling ambiguity. One way is to improve its ability of processing section views by introducing more types of section views. The other is to let the reasoning engine generate a more reasonable result by adding more background knowledge (rules) and conventions, which are acquired from the specific application domain, to its knowledge base.

Acknowledgements

Our work is supported by the Chinese National Science Foundation. Project ID is 69973044. Thanks for kind assistance from GMD.IMK.MARS colleagues during paper revising. Thanks to the editor-in-chief and reviewers for their sophisticated comments and contributions to the revised version.

Appendix A. Algorithmic pseudo codes of human engineer's approach

```
// Global variables for the task of understanding engineering drawings
BoundingBox S_Box; //the 3D object space
ViewList InputViews; //a list of projection views and section views
//at least two projection views are required
Subspace_Partition X_Partition; //subspace partitions along the X axis
Subspace_Partition Y_Partition; //subspace partitions along the Y axis
Subspace_Partition Z_Partition; //subspace partitions along the Z axis
SolidList ResultSolids; //a list of solids recovered from cells or subspace
Boolean bUnderstood; //indicates whether the task is completed successfully or not
Boolean bFurther; //indicates whether the further try of subtask is successful or not
//the main subroutine of human's reconstruction pipeline
Task Understand_Engineering_Draws()
{
bUnderstood = FALSE; //initialize the global Boolean values
Find the bounding rectangles of projection views in the InputViews, and accordingly create the 3D object space, S_Box;
If (Cell_Reconstruction(S_Box) = = TRUE) {
  bUnderstood = TRUE; //the human engineer has understood the input views
  }
else {
  Respectively figure out X_Partition, Y_Partition and Z_Partition, in which include the half-space description of subspace division along each axis etc;
  Try {//the actual sequence (order) of following try-clauses are dependent on human
  //engineers' preferences, experience, skills etc.
    Partition&Reconstruction(S_Box, X_Partition, Y_Partition, Z_Partition);
    Partition&Reconstruction(S_Box, X_Partition, Y_Partition, Z_Partition);
    Partition&Reconstruction(S_Box, Y_Partition, X_Partition, Z_Partition);
    Partition&Reconstruction(S_Box, Y_Partition, Z_Partition, X_Partition);
    Partition&Reconstruction(S_Box, Z_Partition, X_Partition, Y_Partition);
    Partition&Reconstruction(S_Box, Z_Partition, Y_Partition, X_Partition);
```

```
    } Until (bUnderstood = = TRUE} or (all clauses
    have been tried);
  }
If (bUnderstood = = TRUE) {
  Create the result solid based on the list of ResultSolids,
  or just represent it as CSG tree;
  Return 'I have understood it';
  {
else {
  Return 'I need help' or 'something goes wrong';
  }
}//end of Understand_Engineering_Draws()
//this subroutine is to mimic humans' basic understanding
ability
Boolean Cell_Reconstruction(Bounding_Box,
Cell_Box) {
Extract the sub-projection views from the InputViews
according to Cell_Box;
If engineers can 'easily' recover the 3D model based on
the sub-projection views {
  Create its 3D model and insert it into the list of Result-
  Solids;
  Return TRUE;
}
else Return FALSE;
}
//Partition the 3D object space and recover the solid based
on Cell_Reconstruction()
Subtask Partition&Reconstruction(BoundingBox
Sub_Box, //the current subspace
Subspace_Partition firstPartition,
Subspace_Partition secondPartition,
Subspace_Partition thirdPartition)
{
Integer n, i;
n = the number of subspaces in firstPartition;
Divide the Sub_Box into n cell-boxes: cell_0, cell_1, …,
cell_{n-1};
For i = 0 to n - 1 do {
  If (Cell_Reconstruction(cell_i) = = FALSE) {
    bFurther = = FALSE;
    Try {//the sequence (order) of following try-clauses
    are dependent on human
    //engineers' preferences, experience, skills etc.
    Further_Partition&Reconstruction(cell_i, secondPar-
    tition, thirdPartition);
    Further_Partition&Reconstruction(cell_i, thirdParti-
    tion, secondPartition);
    } Until (bFurther = = TRUE) or all clauses have
    been tried;
    If (bFurther = FALSE)) {
      bUnderstood = FALSE;
      Delete all 3D solids inside Sub_Box from the list
      of ResultSolids;
      Return;
      } //end of bFurther = = FALSE
    } //end of if Cell_Reconstruction(cell_i) = = FALSE
  } //end of For…do
bUnderstood = TRUE;
Return;
}
//The core activity of this subroutine is the same as that in
Partition&Reconstruction(). It is
//given separately here just because the non-recursive
requirement
Subtask Further_Partition&Reconstruction(Boun-
dingBox Sub_Box,
Subspace_Partition firstPartition,
Subspace_Partition secondPartition)
{
Integer m, n, i, j;
n = the number of subspaces in firstPartition;
Divide the Sub_Box into n cell-boxes: cell_0, cell_1, …,
cell_{n-1};
For i = 0 to n - 1 do {
  If (Cell_Reconstruction(cell_i) = = FALSE) {
    m = the number of subspaces in secondPartition;
    Divide the cell_i into m sub-cell-boxes: subcell_0,
    subcell_1, …, subcell_{m-1};
    For j = 0 to m - 1 do {
      If (Cell_Reconstruction(subcell_j) = = FALSE) {
        bFurther = FALSE;
        Delete all 3D solids inside Sub_Box from the
        list of ResultSolids;
        Return;
        } //end of If
        (Cell_Reconstruction(subcell_j) = = FALSE)
      } //end of For j = 0 to m - 1 do
    } //end of If (Cell_ReconstructionIcell_i) = FALSE)
    FALSE)
  } //end of For i = 0 to n - 1 do
bFurther = TRUE;
Return;
}
```

Appendix B. Pseudo codes for calculation of 'symmetry' and 'simplicity'

```
Float Calculate_Symmetry_of_Loop(CL)
Loop *CL;
{
Loop *Temp_Loop;
Temp_Loop = MaximizedLoop(CL);
// Temp_Loop is the same as CL except that the neighbor-
ing vertices
// and edges are merged if they share the same position or
line equation
If the lengths of all Temp_Loop's edges are the same
  Return Symmetry of CL as 10
Else if (Temp_Loop is symmetric in vertical or horizontal
direction)
  Return Symmetry of CL as 5
```

```
    Else if (Temp_Loop is similar to parallelogram)
      Return Symmetry of CL is assigned as a value
      between 2 and 3 based on their similarity
      Else Return Symmetry of CL as 1
}
Float Calculate_Simplicity_of_Loop(CL)
Loop *CL;
{
Loop *Temp_Loop;
Int Edge_Number;
Temp_Loop = MaximizedLoop(CL);
// Temp_Loop is the same as CL except that the neighbor-
ing vertices
// and edges are merged if they share the same position or
line equation
Edge_Number = Count_Edges_In_Loop(Temp_Loop);
// the number of edges in Temp_Loop is assigned to
Edge_Number
Return Simplicity of CL as (1/Edge_Number);
}
```

References

[1] Idesawa Masanori. A system to generate a solid figure from three views. Bull JSME 1973;16:216–25.

[2] Wang Weidong, Grinstein GG. A survey of 3D solid reconstruction from 2D projection line drawings. Computer Graphics Forum 1993;12(2):137–58.

[3] Nagendra IV, Gujar VG. 3D objects from 2D orthographics views—a survey. Computers & Graphics 1988;12(1):111–4.

[4] Weidong Geng, Yunhe Pan. A survey of 3D reconstruction from multiple views. Proceedings of CAD/CG'94 Conference of the Chinese Automation Association. Xi'an, PR China, 1994; 87–91.

[5] Dori D, Tombre K. From engineering drawings to 3D CAD models: are we ready now? CAD 1995;27(4):243–54.

[6] Kuo MH. Reconstruciton of quadric surface solids from three-view engineering drawings. CAD 1998;30(7):517–27.

[7] Han Jung Hyan, Requida AAG. Feature recognition from CAD models. IEEE CG&A 1998;March/April:80–103.

[8] Masuda H, Numao N. A cell-based approach for generating solid objects from orthographic projections. CAD 1997;29(3):177–87.

[9] Watanabe T, Tashiro A, Fuji S. Estimation of three-dimensional objects from orthographic views with inconsistencies. Computers & Graphics 1995;19(1):111–4.

[10] You Chun-Fong, Shing Shih. Reconstruction of curvilinear manifold from orthographic views. Computers & Graphics 1996;20(2):275–93.

[11] Yan QW, Chen CLP, Tang Z. Efficient algorithm for the reconstruction of 3D objects from orthographic projections. CAD 1994;26:699–717.

[12] Lysak DB, Decaux PM, Kastari R. View labelling for automated interpretation of engineering drawing. Pattern Recognition 1995; 28(3):393–407.

[13] Liu Chia-Hwa, Perng Der-Beau, Cheng Zen. Automatic form feature recognition and 3D part reconstruction from 2D CAD data. Computer Ind Engng 1994;26(4):698–707.

[14] Eggli L. Hsu Ching-Yao. Bruderlin BD, Elbert G. Inferring 3D models from freehand sketches and constraints. CAD 1997;29(2):101–12.

[15] Zhang Ming. Pyschology of visual cognition. Shanghai, PR China Publishing House of East China Normal University, 1989.

[16] Wang Qichang. The principles of understanding engineering drawings—basic training for mechanical drawings. Beijing, PR China: The Press of Mechanical Industry, 1989.

[17] Mechanical Engineering Drawings Lab of Polytechnic College of North-East. How to draw and understand engineering drawings. Beijing, PR China: The Press of Beijing People, 1976.

[18] Pan Yunhe. The methods and models of intelligent CAD. Beijing, PR China: The Press of Science, 1997.

[19] Joseph SH, Pridmore TP. Knowledge-directed interpretation of mechanical engineering. IEEE Trans Pattern Anal & Mach Intell 1992;14(9):928–40.

[20] Gujar UG, Nagendra IV Construction. of 3D solid objects from orthographic views. Computer & Graphics 1989;13(4):505–21.

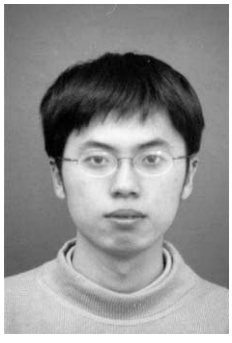
[21] Bin Ho. Inputting constructive solid geometry representations directly from 2D orthographic engineering drawings. CAD 1986;18(3):147–54.

[22] Mantyla M. An introduction to solid modeling. Maryland: Computer Science Press, 1988.

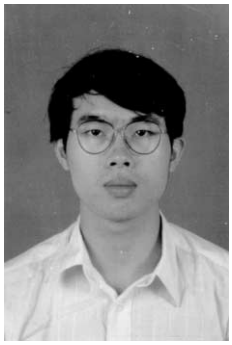
[23] Weidong Geng, Jingbin Wang, Yiyang Zhang, Yunhe Pan. Multiview reconstruction based on cell primitives. The 2000 International Conference on Imaging Science, Systems, and Technology (CISST'2000), June 26–29, 2000, Monte Carlo Resort, Las Vegas, USA. p. 459–66.

[24] Shum SSP, Lau WS, Yuen MMF, Yu Km. Solid reconstruction from orthographic opaque views using incremental extrusion. Computers & Graphics 1997;21(6):787–800.

Weidong Geng currently is a postdoctoral fellow of Institute of Media Communication, GMD-German National Research Center for Information Technology, Germany. He received a BSc degree from the Computer Science Department of Nanjing University, PR China in 1989, and an MSc degree from the Computer Science Department of National University of Defense Technology in 1992. In 1995 he received a PhD degree from the Computer Science and Engineering Department of Zhejiang University, PR China, and worked there until March 2000. His research areas span artificial intelligence, computer-aided design, multimedia, multimodal interface and image-based rendering. He recently authored/co-authored more than 30 papers on these topics.



Jingbin Wang is a master student in Computer Science and Engineering Department at Zhejiang University. He received a BSc degree in Computer Science and Engineering Department of Zhejiang University in 1998. His current research interests include computer graphics, solid modeling and artificial intelligence.



Yiyang Zhang is a master student in the Computer Science and Engineering Department of Zhejiang University. He received a BSc degree in Computer Science and Engineering Department from Zhejiang University in 1998. His current research interests include computer graphics and artificial intelligence.