

Geometric abstractions to support disassembly analysis

N. SHYAMSUNDAR and RAJIT GADH

I-CARVE Laboratory, Department of Mechanical Engineering, University of Wisconsin – Madison, Madison, WI 53705, USA
E-mail: gadh@engr.wisc.edu

Received March 1997 and accepted October 1997

Determining whether an assembly can be constructed from its components at the design stage potentially reduces downstream assembly problems. This determination can be accomplished by performing a disassembly analysis of the assembly's geometric model. This paper presents two abstractions derived from the assembly's geometric model that can determine the validity of the assembly: (1) the Assembly Topology Graph (ATG); and (2) the set of boundary components. The first abstraction, the ATG, is a graph whose nodes represent the components in the assembly and whose edges represent a non-null intersection of the convex hulls of component pairs. The second abstraction, the set of boundary components, represents components that intersect the boundary (or convex hull) of the assembly. These boundary components are typically the ones most accessible with respect to disassembly. This paper also discusses an algorithm, which utilizes the ATG to determine pair-wise interlocking components. If such component pairs are absent, then the disassembly sequence for the removal of components in the assembly is determined by analyzing the set of boundary components for disassembly. This procedure is repeated until all the components in the assembly are disassembled (for a valid assembly).

1. Introduction

The mechanical design process typically begins with the generation of the component design, followed by the creation of an assembled model. However, this design process does not take into account the existence of either an assembly or disassembly sequence for constructing an assembly, A (where $A = \{C_1, C_2, C_3, \dots, C_n\}$). In contrast, the current research performs assembly analysis by obtaining a disassembly sequence of the assembly configuration and reversing the disassembly sequence. Performing a disassembly analysis on a geometric model of the assembly before manufacture allows designers to redesign in case of assembly/disassembly problems.

This paper presents two abstractions to identify one possible disassembly sequence: (1) the Assembly Topology Graph; and (2) the Set of Boundary Components. A is defined as valid if no two components of A occupy the same location in space and if at least one disassembly sequence, \mathcal{S}^A , exists for removing each component ($C_i, i = 1 \dots n$) in the assembly. A requires not only the geometric definition and spatial location of all $C_i \in A$ but also the specific relationship that represents the interactions between $\{C_i\} \in A$. The relationships between all $\{C_i\} \in A$ is captured by the Assembly Topology Graph (ATG).

The ATG is a graph, $G^A(V, E)$ whose V represents $\{C_i\} \in A$ and whose E between two nodes, C_i and C_j in G^A , indicates a non-null intersection of the convex hulls (the minimum convex polyhedron containing the set of

vertices of C_i comprising of planar faces) of C_i and C_j . Using the ATG, pairs of candidate C_i s that could interlock are identified as a “potentially interlocking set”. From this potentially interlocking set, the pairs that interlock are determined by utilizing contact constraints. From these constraints, the disassembly directions are determined, and if none exists, then the pair of C_i s interlock and A is invalid. However, there may exactly be two or more interlocking components. Each subassembly ($SA \subset A$) containing m interlocking C_i s is defined as an Interlocking Sub-Assembly (ISA^m). From the previous definition of a valid A , it is clear that if an ISA^m is present then no \mathcal{S}^A exists, resulting in an invalid A . If \mathcal{S}^A exists, it is guaranteed that $ISA^m (m \geq 2)$ does not exist in A , which will be discussed in Section 4.5. To determine \mathcal{S}^A , the second abstraction, which represents $\{C_i\}$ on the boundary of A , is utilized to determine the candidate C_i s for disassembly.

The following assumptions are made in the current research:

- (1) $\{C_i\}$ are assumed to have planar faces. Most solid modelers contain in-built functions for faceting non-linear faces of a C_i .
- (2) Analysis is restricted to disassembly in the context of single translational motion.
- (3) Simultaneously separable assemblies and assemblies that cannot be taken apart with two hands are not considered.

- (4) For all C_i and $C_j, i \neq j, C_i \cap C_j = \phi$.
- (5) The input is assumed to contain the solid models of C_i s and the location of each C_i in a global coordinate system. There could be additional assembly information available but the existence of such information is not guaranteed and is therefore not utilized.

2. Related research

Determining a valid assembly is closely related to the problems of: (i) assembly sequence determination; (ii) disassembly planning; and (iii) disassembly for design applications. An analysis of the current literature is presented in this section.

2.1. Assembly sequence generation

DeMello and Sanderson [1] have proposed several representation schemes based on directed graphs, AND/OR graphs, establishment conditions and precedence relationships for representing assembly sequences. Assembly planning and generation of the assembly sequence has been analyzed by Wilson and Latombe [2], Mattikalli *et al.* [3], Agarwal *et al.* [4] and Liu and Popplestone [5]. The computational complexity of generating an assembly sequence has been analyzed by Goldwasser *et al.* [6]. Methods to determine the constraints on the translational and rotational motion of objects from their contact geometry have been investigated by Mattikalli and Khosla [7]. Halperin [8] and Chakrabarty and Wolter [9] have addressed the problem of identifying sub-assemblies.

2.2. Disassembly planning

Several approaches have been developed for planning the disassembly of assemblies. Woo and Dutta [10] have developed an algorithm to determine \mathcal{S}^A . In a later study of disassembly, Dutta and Woo [11] examine parallel assemblies. Beasley and Martin [12] consider the generation of disassembly motion for objects represented as a set of cubes. Shin and Cho [13] determine the disassemblability of a part using knowledge-based rules. Xu *et al.* [14] address the problem of geometric path determination to remove a portion of the assembly contained in a cavity within the parent assembly.

2.3. Disassembly for design applications

Kroll *et al.* [15] and Lowe and Niku [16] address methods to quantify the ease of disassembly. Lee and Gadh [17] study the problem of destructive disassembly. They provide an algorithm for determining the optimum cut locations and directions by minimizing the number of cuts for destructive disassembly.

In contrast to the existing research, this paper specifically addresses the problem of determining a valid assembly for which a \mathcal{S}^A exists. Finding an optimal \mathcal{S}^A has been the focus area of several researchers and it is computationally more expensive than determining if at least one \mathcal{S}^A exists. This paper presents an algorithm that determines if a \mathcal{S}^A for a given A exists.

3. Definitions and notation

In this section, the criteria for a valid assembly are stated. According to Requicha and Whalen [18], analysis of the validity of A requires the following two steps:

- (1) Checking for Non-Interference: No two $C_i \in A$ should occupy the same region of space. This requires that the volumetric intersection of any pair of C_i s be null. Turner [19] has discussed an efficient approach to this problem.
- (2) Checking for Assemblability/Disassemblability: C_i is assemblable/disassemblable if a geometric path exists in E^3 to remove C_i , and \mathcal{S}^A exists to assemble/disassemble it. In a valid A , all C_i s are disassemblable.

This paper assumes that for all C_i and $C_j, i \neq j$, Volumetric $(C_i \cap C_j) = \phi$. Therefore, only a valid A must satisfy the second condition above. This paper presents an algorithm to determine \mathcal{S}^A . A component C_i is defined to be *disassembled* from C_j if the intersection of \mathcal{H}_i (where \mathcal{H}_i is the convex hull of C_i) and \mathcal{H}_j is null. This criterion is based on Stoer and Witzgall's [20] theorem that "two non-empty convex polyhedra are linearly separable if and only if their convex hulls do not intersect". Having defined the terminology, the procedure to construct the ATG is discussed in the next section.

4. Assembly topology graph

An A is represented utilizing G^A which consists of: (i) C_i s, defined as solid models; and (ii) spatial relationship between all C_i and $C_j, i \neq j$, in E^3 . The spatial relationship between all pairs of C_i and $C_j, i \neq j$ consists of the relative positioning of C_i and C_j . G^A contains two types of edges: (i) i -edges, and (ii) t -edges.

An i -edge (t_{ij} or t_{ji}) in G^A between two nodes, (C_i and C_j), represents a non-null volumetric intersection of \mathcal{H}_i and \mathcal{H}_j , i.e. $\mathcal{H}_i \cap \mathcal{H}_j = \{\text{Volume}\}$. Therefore, C_i and C_j restrict each other's disassembly. However, a non-null volumetric intersection of \mathcal{H}_i and \mathcal{H}_j does not necessarily result in a ISA^2 .

A t -edge (t_{ij} or t_{ji}) in G^A between two nodes (C_i and C_j) indicates that \mathcal{H}_i and \mathcal{H}_j are physically in contact along a planar face, an edge, or a point i.e., $\mathcal{H}_i \cap \mathcal{H}_j = \{\text{Vertex, Edge or Face}\}$.

When this is the case, it is observed that C_i can be disassembled from C_j by translating along a straight line. C_i and C_j are therefore said to be translationally separable. It follows that if t_{ij} exists then C_i and C_j are translationally separable, which guarantees that C_i and C_j do not interlock.

Figure 1 shows A , G^A , t -edges and i -edges. \mathcal{H}_2 & \mathcal{H}_3 have a non-null volumetric intersection resulting in i_{23} . \mathcal{H}_1 & \mathcal{H}_2 intersect along an edge, resulting in t_{12} between C_1 and C_2 .

G^A is determined from the algorithm **Construct_Assembly_Topology_Graph** below. First the assembly data structure is defined to contain the following: (Assembly A : (i) a list of its components, C_i , $i = 1 \dots n$; (ii) the number of components in the assembly, n ; (iii) a list of convex hulls of C_i , \mathcal{H}_i , $i = 1 \dots n$; (iv) the ATG, G^A ; (v) the set of boundary components, \mathcal{B}_A ; (vi) the convex hull of A , \mathcal{H}_A .)

Construct_Assembly_Topology_Graph (Assembly A)

```

{
  Graph  $A \times G^A = \text{NULL}$ ;
  For ( $i = 1$  to  $A \times n$ )  $A \times \mathcal{H}_i = \text{ConvexHull}(A \times C_i)$ ;
  //for the algorithm refer to Chazelle [21]
  For ( $j = 1$  to  $A \times n$ ) {
    For ( $i = 1$  to  $A \times n$ ) {
      if ( $i \neq j$ ) {
        result = Intersect( $A \times \mathcal{H}_i$ ,  $A \times \mathcal{H}_j$ );
        if (result is non-null and contains one or more polyhedra)
          insert  $i$ -edge in  $A \times G^A$ ;
        else if (result is a face or edge or vertex)
          insert  $t$ -edge in  $A \times G^A$ ;
      }
    }
  }
}
    
```

4.1. Identification of pair-wise interlocking components

Once G^A is determined utilizing the above method, Pair-wise Interlocking Components (PIC) can be determined. If PIC (or ISA^2) is present, A is invalid.

Determination of PIC is explained in the following section. The four ways in which C_i and C_j are positioned

in E^3 result in four categories based on two variables: (i) the contacting nature; and (ii) the interlocking nature. The first variable, the contacting nature, is determined by the interfering volume (IV_{ij}), which is defined as $IV_{ij} = C_i \cap \mathcal{H}_j$. IV_{ij} is represented as boundary representations and may contain one or more disjoint solids. It is to be noted that $IV_{ij} \neq IV_{ji}$. IV contains three types of faces: (i) faces that are in contact with C_j (contacting faces in 3D); (ii) faces that are not in contact with C_j (non-contacting faces); and (iii) faces that are created as a result of the previous intersection operation (created faces). These are shown in Fig. 2. Based on these three types of faces, the two types of IV_{ij} are: (i) those that do not contain any non-contacting faces; and (ii) those that contain non-contacting faces.

The other variable, the interlocking nature, is TRUE if C_i and C_j interlock or FALSE if C_i and C_j do not interlock. Therefore the total number of combinations of these two variables is four as illustrated in Table 1.

The definitions of the four categories are:

(I) *Contact non-interlocking components*: This occurs when there are no non-contacting faces in IV_{ij} and C_i and C_j can be disassembled from each other.

(II) *Contact interlocking components*: This occurs when no face of IV_{ij} is a non-contacting face and C_i and C_j cannot be moved with respect to each other.

(III) *Non-contact non-interlocking components*: This occurs when at least one face of IV_{ij} is a non-contacting face and C_i and C_j can be separated from each other.

(IV) *Non-contact interlocking components*: This occurs when at least one face of IV_{ij} is a non-contacting face and C_i and C_j can be separated from each other. A necessary condition for this to occur is $\mathcal{H}_i \cap \mathcal{H}_j \neq \phi$. No combination of relative linear motion between C_i and C_j will result in $\mathcal{H}_i \cap \mathcal{H}_j = \phi$.

Examples of the above categories are shown in Fig. 3(a–d). Category I is shown in Fig. 3 (a). The contact

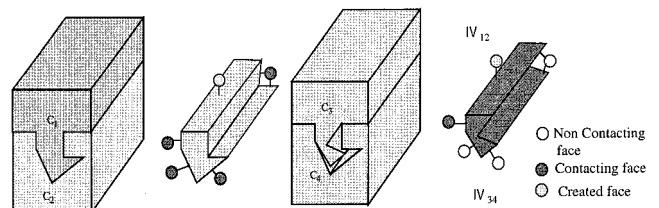


Fig. 2. Interfering volumes.

Table 1. Four categories resulting from the positioning of C_i and C_j in E^3

	IV_{ij} not containing non-contacting faces	IV_{ij} containing non-contacting faces
Non-interlocking	Category I	Category III
Interlocking	Category II	Category IV

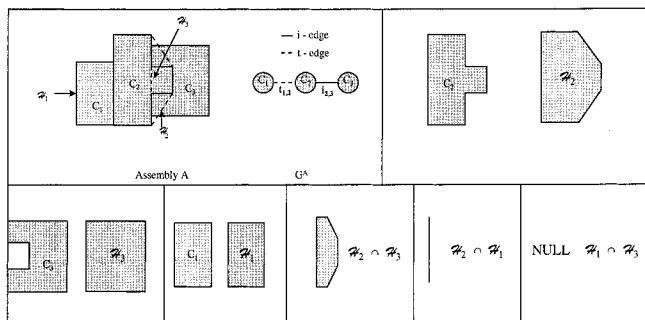


Fig. 1. Assembly topology graph.

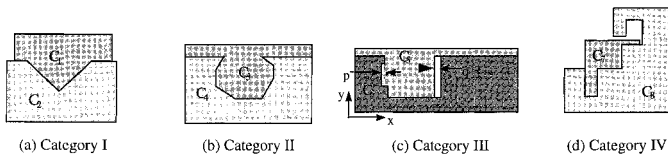


Fig. 3. Types of configurations that interlock.

constraints can be utilized to determine relative directions of disassembly of C_1 from C_2 . Category II, where contact causes the two components to interlock, is shown in Fig. 3(b) (assuming 2D assemblies). An example of Category III is shown in Fig. 3(c). If $p \leq q$, C_6 can be translated along the positive x direction a distance ' p ', after which another translation along the positive y direction disassembles C_5 from C_6 . However, if $p > q$ C_5 and C_6 are PI, the two components will belong to category IV. Figure 3(d) shows another assembly that interlocks due to non-contact constraints.

The difference between the assemblies shown in Fig. 3(a and b) and Fig. 3(c and d) is that their interlocking nature can be identified in the first group by contact geometry, while in the second group contact geometry is insufficient. Only by moving one component with respect to the other can the interlocking nature be determined.

4.2. Categories I and II

Identification of Categories I and II from a pair of components, C_i and C_j , is discussed. The disassembly directions for C_i depend on the contacting faces of C_i with C_j . When a given face of a component C_i mates with another face of C_j , the mating intersection is a *mating face* (there are $r_{i,j}$ mating faces). The separability (defined as a collection of directions in space as represented by vectors pointing from the center of the sphere to its surface) of C_i from the k^{th} mating face M_k between C_i and C_j and is denoted by $S_{i,j}^k$ (where k ranges from 1 to $r_{i,j}$). An example is shown in Fig. 4(a and b), where C_1 and C_2 are in contact along mating faces M_1 and M_2 and $S_{2,1}^1$ and $S_{2,1}^2$ are the separability of C_2 from mating faces M_1 and M_2 . Note that $S_{i,j}^k = -S_{j,i}^k$.

A component can be disassembled along a given direction only if all the mating faces of that component can be disassembled along that direction. If any one mating face cannot be disassembled along a given direction, then that direction is an invalid direction for disassembly. In

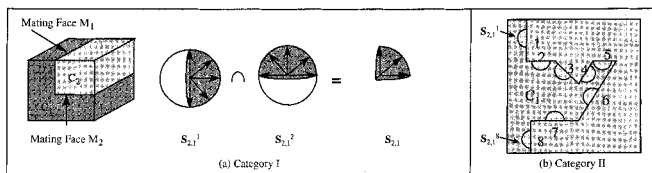


Fig. 4. Finding the separability directions (Category I and Category II).

general, the separability of C_i from C_j (which is the complete set of directions along which C_i can be separated from C_j) is calculated as:

$$S_{i,j} = S_{i,j}^1 \cap S_{i,j}^2 \cap S_{i,j}^3 \cdots S_{i,j}^{r_{i,j}-1} \cap S_{i,j}^{r_{i,j}}, \quad (1)$$

where \cap represents intersection. If $S_{i,j}$ is not NULL then the components belong to Category I, and if $S_{i,j}$ is NULL then the components belong to Category II (PIC).

Two example assemblies are presented for illustration of the calculation of $S_{i,j}$ (Fig. 4(a and b)). In Fig. 4(a), $S_{2,1}$ is not NULL and therefore C_i and C_j are not PIC. In Fig. 4(b), $S_{1,2} = S_{1,2}^1 \cap S_{1,2}^2 \cap S_{1,2}^3 \cdots S_{1,2}^8$, is found to be NULL, indicating that the components belong to Category II (or PIC).

The following algorithm returns a range of disassembly directions for two components of Category I, while it returns NULL for those of Category II.

Separability (C_i, C_j) {
 $S_{i,j}$ = all possible directions;
 For ($k = 1$ to $r_{i,j}$) $S_{i,j} = S_{i,j} \cap S_{i,j}^k$;
 return $S_{i,j}$;}

4.3. Categories III and IV

Categories III and IV cannot be identified by using contact geometry because the interlocking nature is not due to contact, but due to the components' size and position. To determine if C_i from C_j belong to these categories requires efficient motion planning. If a path exists for disassembling C_i from C_j then they belong to Category III, otherwise C_i and C_j belong to Category IV. Examples of different motion planning techniques are discussed in Latombe [22]. However, the current paper does not present a method for identifying Categories III and IV and it is a topic for future research.

In the next section, the procedure for determining the disassembly directions C_i from the rest of A is discussed.

4.4. Disassembly directions of a component in an assembly

Thus far, the analysis presented is for an A consisting of only C_i s. The analysis is now extended to an assembly containing more than two components. The separability DD_i of C_i in contact with C_a, C_b, C_c, \dots in A is given by:

$$DD_i = S_{i,a} \cap S_{i,b} \cap S_{i,c} \cdots \quad (2)$$

From G^A , the subset of A consisting of C_a, C_b, C_c , etc. sharing an edge with C_i is determined. Subsequently, DD_i is determined using Equation (2). If $DD_i = \text{NULL}$, C_i is not disassemblable from the rest of A .

For example, consider A with four C_i s, ($C_1 - C_4$), as shown in Fig. 5. For any C_i and $C_j, i \neq j$, $\mathcal{H}_i \cap \mathcal{H}_j$ is either a single face or NULL, and therefore only the edges of G^A are t -edges. In Fig. 5, DD_2 is a semi-circle as shown.

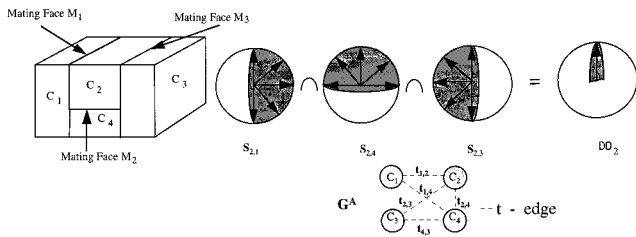


Fig. 5. Finding the range of disassembly directions.

The algorithm for computing DD_i can be summarized by the following procedure.

```

Component_Separability ( $C_i, A$ ) {
     $DD_i =$  all possible directions;
    For (every  $C_k$  sharing an edge with  $C_i$ ,  $i \neq k$ ,
        in  $A \times G^A$ ) {
         $S_{i,k} =$  Separability ( $C_i, C_k$ );
         $DD_i = DD_i \cap S_{i,k}$ ; }
    return  $DD_i$ 
}
    
```

4.5. Using ATG to identify PICs

In the previous sections, the construction of G^A and the procedure for identifying PIC (or category II) was presented. The computational efficiency of using G^A with Separability, as opposed to directly utilizing Separability for determining the presence of PIC in A is discussed in this section.

For a valid A , \mathcal{S}^A exists. An alternative definition for a valid A , in terms of ISA, is that a valid A does not consist of any ISA. These two definitions provide two methods for determining the validity of A . The first method determines if \mathcal{S}^A exists. If \mathcal{S}^A exists, then A is valid, otherwise, it is invalid. The second method utilizes the alternative definition and determines if A contains ISA^2 (PIC). If A contains ISA^2 , then A is invalid, else \mathcal{S}^A is determined. If ISA^m ($m \geq 3$) are present in A then A is invalid. If \mathcal{S}^A exists then ISA^m ($m \geq 3$) are absent in A .

The first and second methods differ in the procedure for determining \mathcal{S}^A . In the first method, the procedure utilized for determining \mathcal{S}^A is as follows.

The disassemblability of all $C_i \in A$ are determined using the algorithm **Component_Separability** and all removable C_i s are removed. This procedure is repeated until $C_i \in A$ are disassembled, in which case A is valid, or no $C_i \in A$ is disassemblable and then A is invalid. In the worst case, if an assembly consists of n C_i s, **Component_Separability** is invoked n^2 times.

In the second method, G^A is first constructed. Then by using G^A , all potential PIC are determined. C_i and C_j are potential PIC if i_{ij} exists. Utilizing G^A , only potential PIC (not PIC) can be determined because G^A indicates the relative spatial positioning of the C_i s, not the contact

constraints. However, it is to be noted that if C_i and C_j are PIC then there must exist an i_{ij} in G^A . To determine if a potential PIC is PIC, the **Separability** algorithm is required. If **Separability** returns NULL then the C_i s are PIC, else C_i s are not PIC.

A Venn diagram representing both potential PIC and PIC is shown in Fig. 6. If A contains ISA^2 then the assembly is invalid. Otherwise, \mathcal{S}^A is determined. Utilizing \mathcal{S}^A , if $\{C_i\} \in A$ can be disassembled then A is valid, otherwise A is invalid. In \mathcal{S}^A , instead of determining the disassemblability of all $\{C_i\} \in A$, the disassemblability of the boundary C_i s is first determined using **Component_Separability**. A disassemblable boundary component is disassembled and the set of boundary components is determined once again, and the disassemblability of the boundary components is computed. This procedure is repeated until all $C_i \in A$ are disassembled, in which case A is valid. If no $C_i \in A$ is disassemblable, A is invalid. The definition of the boundary components and the procedure to determine them is discussed in the next section.

In the second method, if there are PICs, the number of times **Separability** is invoked equals the number of i -edges in the ATG, which in the worst case is n^2 . Using the second method, A containing PICs are eliminated before \mathcal{S}^A is determined. In the first method, however, even if A contains PICs, \mathcal{S}^A is determined and, at some stage of the computation, the components forming the PIC are not disassemblable. Therefore, the assembly is determined to be invalid. If the PICs are present in the interior of A , the number of computations required to determine the validity of A is greater in the first method than in the second method. However, the second method requires fewer steps if A contains PIC. The second method is utilized in this paper, and the algorithm for this method is presented in Section 6. Moreover, in the general case, the computational complexity of the second method is less than the first method. A detailed discussion of the computational complexity of the second method is provided in Section 7.

The two methods explained above are illustrated using A , as shown in Fig. 7(a–c). Here C_6 and C_7 are PIC. Determining if A is valid using the first approach requires the following steps:

- (1) Determine the disassembly direction for removing all $C_i \in A$ by determining the constraints imposed

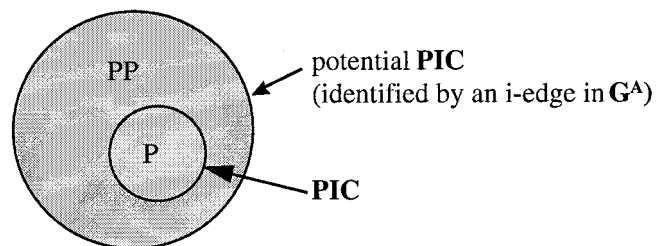


Fig. 6. Venn diagram of the space containing pairs of components in the assembly.

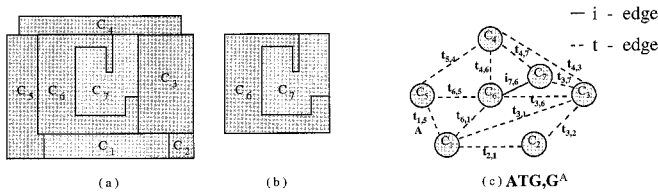


Fig. 7. Determining the invalidity of A.

by the contacting components. The components C_1, C_2, C_3, C_4, C_5 are disassemblable, therefore these components are removed.

- (2) The directions for removal of the remaining components in the assembly are determined (in this case C_6, C_7). Since there are no possible directions for disassembly, C_6 and C_7 are interlocking. Hence A is invalid.

Conversely, using the second approach (i.e., application of the weak condition) for determining the validity of the assembly given the ATG involves the following steps.

- (1) All component pairs sharing an i -edge in the ATG are determined. For the given A , there is only one pair (C_6 and C_7) containing an i -edge.
- (2) The algorithm **Separability** is invoked for every pair of components sharing an i -edge – in this case C_6 and C_7 . Since the separability of C_6 and C_7 is null they are interlocking. Thus A is invalid.

The second method is more computationally efficient than the first method if ISA^2 (or PIC) is present in A . In the second method, if ISA^2 are absent in A , the next step is to determine \mathcal{S}^A . During the determination of \mathcal{S}^A , the proposed algorithm attempts to disassemble the boundary components, which have greater accessibility than interior components. In the next section, the procedure for determining the boundary components is discussed.

5. Boundary components

Identification of the boundary components involves the construction of the convex hull of A (boundary of A) and the determining of all $\{C_i\} \in A$ that share a vertex, edge or face with the boundary of A . First, the following terms need to be defined:

- (a) *Assembly boundary*: It is defined as the boundary surface of the convex hull \mathcal{H}_A of the assembly, A , and is denoted as $\delta\mathcal{H}_A$.
- (b) *Boundary surface*: The boundary surface of C_i is defined as the set of all faces, edges and vertices of C_i denoted by δC_i .
- (c) *Boundary component*: A component is a boundary component if it has a non-null $\mathcal{H}_{i,A}$, where $\mathcal{H}_{i,A}$ is defined as the intersection of the boundary surface of $C_i, \delta C_i$ and the assembly boundary, $\delta\mathcal{H}_A$:

$$\mathcal{H}_{i,A} = \{\delta C_i \cap \delta\mathcal{H}_A\}. \tag{3}$$

For A shown in Fig. 8a(i), $\delta\mathcal{H}_A$ is such that $\mathcal{H}_{1,A}, \mathcal{H}_{2,A}$ and $\mathcal{H}_{3,A}$ are NON-NULL and are shown in Fig. 8a(iii), (iv) and (v). Hence C_1, C_2 and C_3 are boundary components, whereas C_4 is not, since $\mathcal{H}_{4,A}$ is null.

All $\{C_i\} \in A$ with non-null $\mathcal{H}_{i,A}$ form the set of boundary components, and this set is represented by \mathcal{B}_A . Based on our observation of industrial design, we assume that the greater the number of entities C_i shares with \mathcal{H}_A , the more accessible is C_i for removal from A . Another assumption is if two C_i s share both a face and an edge, then the C_i sharing a face with \mathcal{H}_A is considered to be more accessible than the one sharing an edge with \mathcal{H}_A . The justification of this assumption lies in the fact that it is typically easier to hold or grasp a C_i whose entire face is exposed rather than one without an exposed edge. From A shown in Fig. 8(b), $\mathcal{B}_A = \{C_1, C_2, C_3, C_4, C_5, C_6, C_9, C_{10}, C_{13}, C_{14}, C_{15}, C_{16}, C_{17}, C_{18}\}$. C_3 shares an edge with \mathcal{H}_A , while C_1 shares two faces with \mathcal{H}_A . As can be seen from Fig. 8(b), the C_i s sharing a face are easier to remove than one sharing an edge (in this case C_3). Also, C_2 shares a face with \mathcal{H}_A , whereas C_1 shares two faces with \mathcal{H}_A . It can be observed that C_1 has greater accessibility than C_2 .

For the purposes of disassembly, the components in \mathcal{B}_A are listed in the order of decreasing accessibility. However, the ordering is based on a heuristic and does not guarantee that the first element in the list will always be easier to access in practice than a subsequent element in the list. Yet the ordering provides a systematic way to search for disassemblable components in the assembly. To determine the order in \mathcal{B}_A , the boundary components are grouped into three sets: (i) set \mathcal{B}_A^F : these are $\{C_i\}, i = 1$ to n_F such that each $\mathcal{H}_{i,A}$ contains at least one face. Therefore, each δC_i shares at least one face with $\delta\mathcal{H}_A$; (ii) set \mathcal{B}_A^{EE} : these are $\{C_i\}, i = 1$ to n_{EE} for which each $\mathcal{H}_{i,A}$ contains at least one edge. Therefore, each δC_i shares at least one edge with $\delta\mathcal{H}_A$; (iii) set \mathcal{B}_A^{VV} : these are $\{C_i\}, i = 1$ to n_{VV} for which each $\mathcal{H}_{i,A}$ contains at least one vertex. Therefore, each δC_i shares at least one vertex with $\delta\mathcal{H}_A$.

$\{C_i\}$ sharing at least one face are grouped in the set \mathcal{B}_A^F . All $C_i \in A, C_i \notin \mathcal{B}_A^F$ (sharing at least one edge with \mathcal{H}_A) are grouped to form set \mathcal{B}_A^E , i.e., $\mathcal{B}_A^E = \mathcal{B}_A^{EE} - \mathcal{B}_A^F$. All $C_i \in A, C_i \notin \mathcal{B}_A^E, C_i \notin \mathcal{B}_A^F$ (sharing at least one vertex with \mathcal{H}_A) are grouped to form \mathcal{B}_A^V , i.e. $\mathcal{B}_A^V = \mathcal{B}_A^{VV} - (\mathcal{B}_A^E \cup \mathcal{B}_A^F)$. $C_i \in \mathcal{B}_A^F (= C_i^F, i = 1$ to $n_f)$ are ordered based on the number of faces each component shares with $\delta\mathcal{H}_A$,

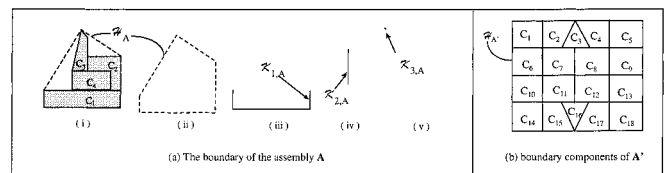


Fig. 8. The boundary of the assembly and boundary components.

in order of decreasing number of faces shared. $C_i \in \mathcal{B}_A^E (= C_i^E, i = 1 \text{ to } n_e)$ and $C_i \in \mathcal{B}_A^V (= C_i^V, i = 1 \text{ to } n_v)$ are also ordered in the same manner as \mathcal{B}_A^F except that \mathcal{B}_A^E is based on the number of edges and \mathcal{B}_A^V on the number of vertices. Subsequently, a set \mathcal{B}_A is defined as the ordered union (where $\cup^\#$ is the ordered union) of the sets $\mathcal{B}_A^F, \mathcal{B}_A^E$ and \mathcal{B}_A^V as follows:

$$\begin{aligned} \mathcal{B}_A &= \mathcal{B}_A^F \cup^\# \mathcal{B}_A^E \cup^\# \mathcal{B}_A^V \\ &= \{C_1^F, \dots, C_{n_f}^F, C_1^E, \dots, C_{n_e}^E, C_1^V, \dots, C_{n_v}^V\}, \\ &= \{\ell_A^1, \ell_A^2, b_A^3 \dots \ell_A^{n_f+n_e+n_v}\}. \end{aligned} \quad (4)$$

The classification of \mathcal{B}_A is summarized in the Venn diagram of Fig. 9. The following algorithm summarizes the determination of \mathcal{B}_A .

Identify_Boundary_Components (Assembly A)

```
{
Set_of_Boundary_Components  $A \times \mathcal{B}_A = \text{NULL}$ ;
// set of boundary components is initialized to NULL.
 $\mathcal{B}_A^F = \text{NULL}$ ,  $\mathcal{B}_A^{EE} = \text{NULL}$ ,  $\mathcal{B}_A^{VV} = \text{NULL}$ ;
 $\mathcal{H}_A = \text{ConvexHull}(A)$ ;
For ( $i = 1$  to  $A \times n$ ) {
 $\mathcal{H}_{i,A} = \text{NULL}$ ;
 $\mathcal{H}_{i,A} = \{\delta(A \times C_i) \cap \delta(A \times \mathcal{H}_A)\}$ ;
if ( $\mathcal{H}_{i,A} \neq \text{NULL}$ ) {
if ( $\mathcal{H}_{i,A}$  contains face(s))  $\mathcal{B}_A^F = \mathcal{B}_A^F \cup A \times C_i$ ;
if ( $\mathcal{H}_{i,A}$  contains edges)  $\mathcal{B}_A^{EE} = \mathcal{B}_A^{EE} \cup A \times C_i$ ;
 $\mathcal{B}_A^{VV} = \mathcal{B}_A^{VV} \cup A \times C_i$ ; //  $\mathcal{H}_{i,A}$  contains vertices
}
}
 $\mathcal{B}_A^E = \mathcal{B}_A^{EE} - \mathcal{B}_A^F$ ;
 $\mathcal{B}_A^V = \mathcal{B}_A^{VV} - (\mathcal{B}_A^E \cup \mathcal{B}_A^F)$ ;
Order  $\mathcal{B}_A^F$  in decreasing number of faces each component
shares with  $\delta(A \times \mathcal{H}_A)$ ;
Order  $\mathcal{B}_A^E$  in decreasing number of edges each component
shares with  $\delta(A \times \mathcal{H}_A)$ ;
Order  $\mathcal{B}_A^V$  in decreasing number of vertices each component
shares with  $\delta(A \times \mathcal{H}_A)$ ;
 $A \times \mathcal{B}_A = \mathcal{B}_A^F \cup^\# \mathcal{B}_A^E \cup^\# \mathcal{B}_A^V$ ; // where  $\cup^\#$  is the ordered
union operation
}
```

5.1. Determination of $\mathcal{H}_{i,A}$

In the above algorithm, $\mathcal{H}_{i,A}$, which equals $\delta C_i \cap \delta \mathcal{H}_A$, consists only of faces, edges and vertices but not volumes.

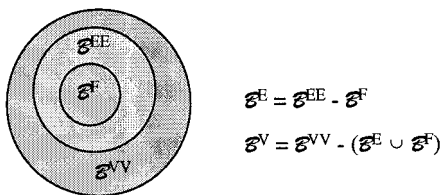


Fig. 9. Classification of boundary components.

Since the resulting entities are not volumes, a data structure that allows efficient intersection is discussed below. First, the procedure for determining $\delta C_i \cap \delta \mathcal{H}_i$ is discussed. Later, the generalization of this procedure to determine $\delta C_i \cap \delta \mathcal{H}_A$ is presented.

Consider C_1 , shown in Fig. 10(a), and \mathcal{H}_1 , shown in Fig. 10(b). The faces and edges in \mathcal{H}_1 and C_1 are not identical. However, the set of vertices of \mathcal{H}_1 is necessarily a sub-set of the vertices of C_1 (assuming planar boundary solid models). The vertex V_1 on C_1 is the corresponding vertex of V_1' on \mathcal{H}_1 . The faces of C_1 that are present in \mathcal{H}_1 (for example F_1 on C_1 and F_1' on \mathcal{H}_1) are called self-faces. Faces that belong to \mathcal{H}_1 and have no corresponding face on C_1 are called generated-faces. A similar terminology is used for the edges. E_1' is a self-edge (the corresponding edge on C_1 is E_1) and E_3' is a generated edge. A natural corollary of the gift wrapping algorithm for convex hull determination of Preparata and Shamos, [23] is that $\delta C_i \cap \delta \mathcal{H}_A$ includes the vertices of \mathcal{H}_A , since the vertices of C_i are required by the gift wrapping algorithm and no new vertices are created. Furthermore, some edges (self-edges) and faces (self-faces) are also common to both $\delta \mathcal{H}_1$ and δC_1 . The data structure for storing the convex hull is as follows:

```
Convex Hull  $\mathcal{H}_i$  {
List of vertices,  $V_i, i = 1 \dots p$ .
List of loops of vertices  $L_i, i = 1 \dots s$ , that form a
face; }
Vertex  $V_i$  {
Coordinates of the point representing the vertex;
Equations of the edges incident on the vertex;
Equations of the faces incident on the vertex;
Pointer to the corresponding vertex on the
component;
}
```

It is observed that $\delta C_i \cap \delta \mathcal{H}_i$ consists of all the vertices of \mathcal{H}_i , the self-edges \mathcal{H}_i and self-faces of \mathcal{H}_i . Therefore, if

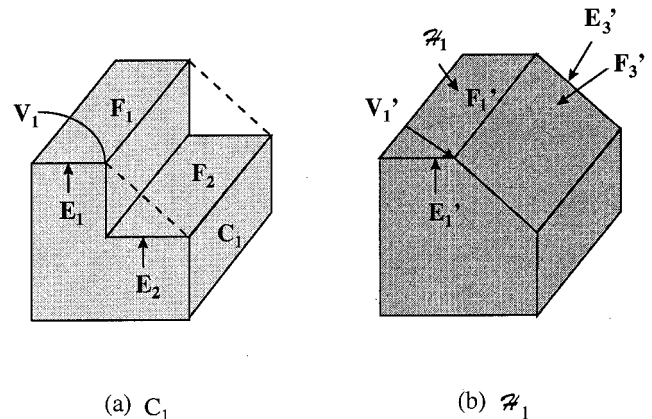


Fig. 10. Self and generated faces.

the self-edges and self-faces of \mathcal{H}_i are identified, $\delta C_i \cap \delta \mathcal{H}_i$ is determined. The procedure to identify the self-edges and self-faces is discussed next. Given C_i , its boundary representation consists of the geometry and topology of the vertices, edges and faces. A similar data structure is utilized for storing \mathcal{H}_i . Standard convex hull algorithms require a collection of points to be provided as input. The data structure that stores the resulting convex hull stores the vertices and the vertex topology. The vertex topology includes the geometry of the incident edges and incident faces. The topology of vertices of \mathcal{H}_i is utilized to determine the generated edges and faces. All the edges and faces of \mathcal{H}_i that are not generated edges or generated faces are self-edges and self-faces. $\delta C_i \cap \delta \mathcal{H}_i$ is equal to the vertices of \mathcal{H}_i , the self-edges of \mathcal{H}_i and the self-faces of \mathcal{H}_i . To determine the self-edges and self-faces of the convex hull, the following algorithm is utilized:

Self-Edges_and_Faces (\mathcal{H}_i)

```

For (each  $V_k$  of  $\mathcal{H}_i$ )
  For (every edge (E) incident on  $V_k$ )
    If (equation of incident E is = the equation of E
    incident on the corresponding vertex on  $C_i$ ) then
      E is tagged as self-edge.
    else E is a generated-edge.
  For (every face (F) incident on  $V_k$ )
    If (equation of incident F is = the equation of F
    incident on the corresponding vertex on  $C_i$ )
      then F is tagged as self-face.
    else F is a generated-face.
}
    
```

If there are v vertices in the convex hull and the degree of each vertex (where degree is the number of edges incident on the vertex) is d , the complexity of the above algorithm is $O(vd^2)$. If d is a constant then the complexity of the algorithm is $O(v)$.

Having explained the procedure for $\delta C_i \cap \delta \mathcal{H}_i$, the procedure for determining the $\mathcal{H}_{i,A} (= \delta C_i \cap \delta \mathcal{H}_A)$ is now discussed. To determine \mathcal{H}_A , the vertices of $\{C_i\}$ are required by the algorithm **ConvexHull**. The data structure storing \mathcal{H}_A contains a list of V_k s. Each V_k on \mathcal{H}_i has a pointer to V_i on a C_i (corresponding vertex) having the same co-ordinates as V_k . Through this pointer, the geometric information of the edges and faces incident on the corresponding vertex can be determined. Once the \mathcal{H}_A is constructed using **ConvexHull**, the **Self-Edges_and_Faces** is invoked with \mathcal{H}_A as the argument. The following algorithm is then utilized to compute $\mathcal{H}_{i,A}$:

```

Compute  $\mathcal{H}_{i,A}(C_i, \mathcal{H}_A)\{$ 
  Array  $\mathcal{H}_{i,A}$ ;
  For ( $i = 1$  to  $n$ )  $\mathcal{H}_{i,A} = \text{NULL}$ ;
  For (every vertex  $V_j$  of  $\mathcal{H}_A$ )\{
    
```

```

    Let  $C_i$  be the component having the
      corresponding vertex of  $V_j$ .
    //  $C_i$  can be determined using the pointer stored
    // in the data
    // structure of the vertex  $V_j$ .
    Add  $V_j$  to  $\mathcal{H}_{i,A}$ .
    Add all self-edges incident on  $V_j$  to  $\mathcal{H}_{i,A}$ 
    Add all self-faces incident on  $V_j$  to  $\mathcal{H}_{i,A}$ .
  }
}
    
```

The complexity of the above algorithm is linear ($O(v)$). Thus finding $\mathcal{H}_{i,A}$ in **Identify_Boundary_Components** involves accessing in the array of $\mathcal{H}_{i,A}$ generated by invoking the above algorithm after computing \mathcal{H}_A .

The algorithm **Identify_Boundary_Components** is explained using the assembly shown in Fig. 11(a-c). The assembly contains nine components. \mathcal{H}_A is shown in Fig. 11(b), and $\mathcal{H}_{i,A} (i = 1, 3, 6, 7, 8)$ are shown in Fig. 11(c). $\mathcal{H}_{1,A}, \mathcal{H}_{3,A}, \mathcal{H}_{6,A}, \mathcal{H}_{7,A}, \mathcal{H}_{8,A}$ are non-null. The other $\mathcal{H}_{i,A} (i = 1, 3, 6, 7, 8)$ are null. Hence the assembly has five boundary components (C_1, C_8, C_7, C_6, C_3). The boundary components are as follows:

$$\mathcal{B}_A^F = \phi \text{ (since the assembly is 2D),}$$

$$\mathcal{B}_A^E = \mathcal{B}_A^{EE} - \mathcal{B}_A^F = \{C_1, C_7, C_6, C_3\},$$

$$\mathcal{B}_A^{VV} = \{C_1, C_8, C_7, C_6, C_3\};$$

$$\mathcal{B}_A^V = \mathcal{B}_A^{VV} - \mathcal{B}_A^E = \{C_8\}.$$

After ordering the sets:

$$\mathcal{B}_A^F = \phi, \mathcal{B}_A^E = \{C_1, C_3, C_7, C_6\}; \mathcal{B}_A^V = \{C_8\},$$

$$\mathcal{B}_A = \mathcal{B}_A^F \cup^\# \mathcal{B}_A^E \cup^\# \mathcal{B}_A^V = \{C_1, C_3, C_7, C_6, C_8\}.$$

In the previous sections, the procedures for constructing the ATG and determining PIC given the ATG were discussed. The procedure for determining \mathcal{B}_A was discussed in this section. The next section presents the complete algorithm for determining the validity of an assembly using the algorithms developed thus far.

6. Identifying valid assemblies

In the algorithm for finding a valid A, G^A is first constructed. Then any PIC are identified from the potential PIC (all $C_i \in A$ that share an i -edge in the ATG). If any

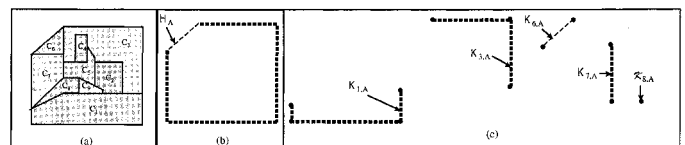


Fig. 11. Boundary components.

PIC is present, A is invalid. Otherwise, \mathcal{B}_A is determined. If one of $C_i \in \mathcal{B}_A$ is disassemblable then that C_i is removed from $\{C_i\} \in A$. The ATG is updated and \mathcal{B}_A is re-computed. The removal of a $C_i \in \mathcal{B}_A$ and the updating of G^A and \mathcal{B}_A is repeated until $C_i \in A$ are disassembled. If $\{C_i\} \in A$ are disassemblable then the algorithm reports that A is valid. However, if $C_i \in \mathcal{B}_A$ that is disassemblable does not exist then A is invalid. The detailed algorithm is presented below:

```

Identify_Valid_Assembly (Assembly  $A$ )
{
  1. Construct_Assembly_Topology_Graph( $A$ );
  2. For (each  $i$ -edge in  $A \times G^A$  between  $C_i$  and  $C_j$ ) {
  3.   return_value = Separability( $C_i, C_j$ );
  4.   if (return_value = NULL) {
  5.     return "Invalid Assembly";
  6.   }
  7. }
  8. Identify_Boundary_Components( $A$ );
  9. size = Cardinal_Number( $A \times \mathcal{B}_A$ );
  10.  $k = 1$ ;
  11. While ( $k \leq$  size) {
  12.   result = Component_Separability ( $\mathcal{C}_A^k, A$ );
  //  $\mathcal{C}_A^k$  :  $k$ th component in  $\mathcal{B}_A$ 
  13.   if (result  $\neq$  NULL) { // not interlocking
  14.     Decrement  $A \times n$  by 1;
  15.     Delete node containing  $C_i$  and all the edges
  incident on it in  $A \times G^A$ ;
  16.     if ( $A \times n = 0$ )
  17.       return "Valid Assembly";
  18.     Identify_Boundary_Components ( $A$ );
  19.      $k = 1$ ;
  20.     size = Cardinal_Number( $A \times \mathcal{B}_A$ );
  21.   }
  22.   else
  23.      $k = k + 1$ ;
  24. }
  25. return "Invalid Assembly";
}

```

To determine if A is valid the algorithm requires the assembly A as input. First, the G^A is constructed (Line 1). If A has one or more ISA^2 s (or PICs), A is invalid (Line 5) and the program exits. If no PICs are present, \mathcal{B}_A is determined (Line 8). The separability of \mathcal{C}_A^k , in \mathcal{B}_A is determined (Line 12). If $DD^k \neq$ NULL can be found, then the current component is removed. Subsequently, G^A is updated and \mathcal{B}_A is re-computed (Lines 15–20). By this algorithm, if $C_i \in A$ can be disassembled, then A is valid (Line 17). If, however, during any iteration, all the boundary components are interlocking then A is invalid (Line 25).

The working of the algorithm is illustrated by the following examples. An example of a valid $A(A1)$ with four

components is shown in Fig. 12(a). G^{A1} is shown in Fig. 12(a). As can be seen from G^{A1} , $i_{1,4}$ and $i_{2,3}$ are present. Since the two pairs ($\{C_1, C_4\}$ and $\{C_2, C_3\}$) share an i -edge they could be potential PIC. Therefore, the procedure to compute the separability is invoked (Line 3). The **return_value** is NON NULL, hence they are not PIC. Subsequently, \mathcal{B}_{A1} is constructed (Line 9), \mathcal{C}_{A1}^1 (C_1) is removed (Lines 12–24), the assembly remaining after the removal of $C_1, A1'$, is determined. Finally, lines 12–24 are repeated, $G^{A1'}$ and \mathcal{B}_A are updated, C_4 is removed (as it is $\mathcal{C}_{A1'}^1$), and lines 12–24 are repeated for $A1''$, which is the assembly after the removal of components C_1 and C_4 . Therefore, for this assembly, there exists a \mathcal{S}^{A1} . Hence the algorithm returns "Valid Assembly".

The second example of an $A(A2)$, one that is invalid, is shown in Fig. 12(d). G^{A2} is first constructed (Line 1). Since C_1 and C_3 share an i -edge, they could be PIC, and therefore line 5 is executed and the result obtained is null. Therefore, $A2$ is invalid.

7. Computational complexity

The current section presents the computational requirement of the algorithm presented in the previous section. The algorithm is based on the construction of two geometric abstractions: (i) the ATG G^A (the computational requirement for computing G^A is presented in Section 7.1); and (ii) \mathcal{B}_A (the computational requirement for computing \mathcal{B}_A is presented in Section 7.2). The computational requirement for the algorithm is presented in Section 7.3.

7.1. ATG

There are two steps in the construction of G^A : (i) the construction of \mathcal{H}_i ; and (ii) the determination of $\mathcal{H}_i \cap \mathcal{H}_j$.

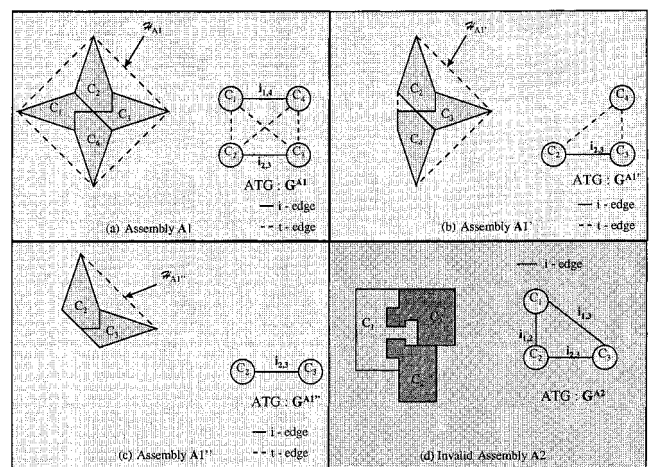


Fig. 12. A valid assembly and an interlocking assembly.

- (i) For constructing the convex hull of a set of points in E^3 , $O(k_i \log(k_i))$ (where k_i is the number of vertices in C_i) computations are required [24,21], assuming the geometry is modeled in terms of planar faces.
- (ii) There are several algorithms to find the intersection of convex polyhedra. Muller and Preparata [25] have proposed an algorithm. Stefan *et al.* [26] have proposed another algorithm. Both the algorithms require $O(p_{ij} \log(p_{ij}))$ computations to find the intersection, where $p_{ij}(= k_i + k_j)$ is the total number of vertices in the two polyhedra, C_i and C_j . A linear complexity algorithm, $O(p_{ij})$, which detects whether two polyhedra intersect (but does not calculate their intersection) is presented by Dobkin and Kirkpatrick [27]. Another algorithm, an extension of the previous one by Chazelle [28] performs this polyhedral intersection with linear complexity $O(p_{ij})$.

To construct G^A , the following is the complexity analysis: (i) n calls are made to the convex hull determination algorithm, and each call requires $O(k_i)$ computations, resulting in a complexity of this step to be $O(nk)$, where “ k ” is the max, k_i ; and (ii) n^2 calls to the polyhedral intersection detection algorithm are required, with each call having $O(p_{ij})$ complexity. Thus, the complexity for this step is $O(n^2p)$, where p (maximum (k_i, k_j) , $i = 1 \dots n$ and $j = 1 \dots n, i \neq j$) is the maximum of total number of vertices of all pairs of components in the assembly. Hence the complexity of the ATG determination algorithm is $O(n^2p + nk)$. Since $p > k_i$, the complexity becomes $O(n^2p)$. The next section explains the computational requirement for computing \mathcal{B}_A .

7.2. Boundary components

Determining \mathcal{B}_A involves the determination of $\mathcal{H}_{i,A}$ for all $\{C_i\} \in A$. This requires constructing \mathcal{H}_A and invoking the algorithms **Self-Edges_and_Faces** and **Compute_** $\mathcal{H}_{i,A}$. The procedure for finding $\mathcal{H}_{i,A}$ was discussed in Section 4.1. It may be inferred from the discussion that the computation required $O(v)$, where $v (\leq \sum k_i, \text{ where } i = 1 \dots n)$ is the number of vertices of \mathcal{H}_A . Having determined the computational requirement of two important steps in the algorithm, the total complexity of the algorithm **Identify_Valid_Assembly** is presented in the next section.

7.3. Identifying valid assemblies

The algorithm for identifying valid assemblies (**Identify_Valid_Assembly**) invokes the algorithm for constructing the ATG and also makes several calls to the algorithm for computing the boundary components.

The algorithm for computing the ATG (**Construct_Assembly_Topology_Graph**) is called at the begin-

ning of **Identify_Valid_Assembly**. Next, the \mathcal{S}^A is determined, which requires the determination of \mathcal{B}_A . For all $C_i \in \mathcal{B}_A$ if any C_i is disassemblable, it is removed from A . Subsequently, G^A is updated and \mathcal{B}_A is recomputed. This is repeated until all $C_i \in A$ are disassembled or until no $C_i \in \mathcal{B}_A$ is disassemblable.

Since there are nC_i s $\in A$, the algorithm for determining \mathcal{B}_A is invoked a maximum of n times. The complexity for determining \mathcal{B}_A is $O(v)$, the complexity for determining \mathcal{S}^A is $O(nv)$. The total complexity of the algorithm is the sum of the: (i) complexity for determining G^A ; and (ii) that for determining \mathcal{S}^A , which is $O(n^2p + nv)$. Since the algorithm has polynomial complexity, the determination of the validity of A can be performed in a reasonable amount of computational time.

8. Visible boundary components

There is a special case of A in which every $C_i \in \mathcal{B}_A$ is not disassemblable, yet A is valid. For A shown in Fig. 13 C_5 can be disassembled, and subsequently, the other components are removed sequentially. It can be inferred from Fig. 13 that $\mathcal{B}_A = \{C_1, C_2\}$ and none of the C_i are disassemblable. However, $C_5 \notin \mathcal{B}_A$ ($\mathcal{H}_{5,A}$ is NULL), is disassemblable. C_5 is disassemblable since it is accessible from \mathcal{H}_A . To state this formally, C_i is defined to be visible from $\delta\mathcal{H}_A$ if a straight line exists that joins a point on $\delta\mathcal{H}_A$ and a point on δC_i . Furthermore, the straight line should not intersect with any $C_i \in A$. A visible component C_i is first determined then the algorithm **Component_Separability** is invoked, resulting in a DD_i . From DD_i it is necessary to identify a direction along which the C_i can be translated infinitely without interfering with any other $C_i \in A$. To determine if this condition is satisfied, the following algorithm is utilized:

```
Sweep ( $A, d, C_i$ ) {
    Sweep  $C_i$  in the direction  $d$ ;
    If the swept volume does not intersect the rest of the
    assembly then  $C_i$  can be disassembled in direction  $d$ .}
```

If a visible disassemblable C_i can be found, then it is disassembled from A , the ATG and \mathcal{B}_A are updated and the validity of the rest of the assembly is determined using

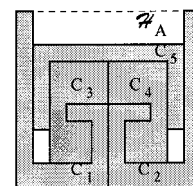


Fig. 13. Assemblies with interlocking boundary components.

the algorithm **Identify_Valid_Assembly**. If none of the visible $C_i \in A$ are disassemblable, A is invalid.

9. Summary

The current research presents an approach that analyzes the disassemblability of an assembly utilizing: (i) the assembly topology graph; and (ii) the set of boundary components. Using the geometric model of the assembly, the assembly topology graph is constructed. This graph is used to identify potentially interlocking pairs of components which, if present, result in an invalid assembly. If no such components are present, the second abstraction, which represents the set of boundary components, is determined. If an element of the set of boundary components is disassemblable, it is removed, and then the assembly topology graph, along with the set of boundary components, is updated. The process is repeated until all components are disassembled, in which case the assembly is valid. If none of the boundary components are disassemblable, then one or more visible boundary components can be disassembled or the assembly is invalid.

This paper discussed the two abstractions, the assembly topology graph and the set of boundary components as well as the algorithms for identifying valid assemblies using these abstractions.

References

- [1] Homem DeMello, L.S. and Sanderson, A.C. (1991) Representations of mechanical assembly sequences. *IEEE Transactions on Robotics and Automation*, **7**, 211–277.
- [2] Wilson, R.H. and Latombe, J.C. (1994) Geometric reasoning about mechanical assembly. *Artificial Intelligence*, **71**(2), 1–31.
- [3] Mattikalli, R.S., Khosla, P.K. and Xu, Y. (1990) Sub-assembly identification and motion generation for assembly: a geometric approach. *Proceedings of the 1990 IEEE International Conference on Systems Engineering*, Pittsburgh, PA, IEEE, IEEE Service Center, Piscataway, NJ, 399–403.
- [4] Agarwal, P.K., Berg, M.D., Halperin, D. and Sharir, M. (1996) Efficient generation of k -directional assembly sequences, in *Proceedings of the 7th ACM-SIAM Symposium on Discrete Algorithms*, Atlanta, USA, 122–131.
- [5] Liu, Y. and Poplestone, R.J. (1989) Planning for assembly from solid models, in *Proceedings of the IEEE International Conference on Robotics and Automation*, Scottsdale, AZ, IEEE Service Center, Piscataway, NJ, Vol 1, pp. 222–227.
- [6] Goldwasser, M., Latombe, J.C. and Motwani, R. (1996) Complexity measures for assembly sequences, in *Proceedings of the 13th IEEE International Conference on Robotics and Automation*, Part 2, Minneapolis, MN, IEEE Service Center, Piscataway, NJ, pp. 1581–1587.
- [7] Mattikalli, R.S. and Khosla, P.K. (1992) Motion constraints from contact geometry: representation and analysis, in *Proceedings of the IEEE International Conference on Robotics and Automation*, Nice, France, IEEE Service Center, Piscataway, NJ, pp. 2178–2185.
- [8] Halperin, D. (1994) Assembly partitioning with a constant number of translations. *Technical Report Sandia National Labs*, SAND94-1819, 1–17.
- [9] Chakrabarty, S. and Wolter, J. (1994) A hierarchical approach to assembly planning, in *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, IEEE Service Center, Piscataway, NJ, pp. 258–263.
- [10] Woo, T.C. and Dutta, D. (1991) Automatic disassembly and total ordering in three dimensions. *ASME Transactions Journal of Engineering for Industry*, **13**(2), May, 207–213.
- [11] Dutta, D. and Woo, A.C. (1992) Algorithms for multiple disassembly and parallel assemblies. *Concurrent Engineering*, **59**, 257–266.
- [12] Beasley, D. and Martin, R.R. (1993) Disassembly sequences for objects built from unit cubes. *CAD*, **25**, 751–761.
- [13] Shin, K. and Cho, H.S. (1994) On the generation of robotic assembly sequences based on separability and assembly motion stability. *Robotica*, **12**, 7–15.
- [14] Xu, Y., Mattikalli, R. and Khosla, P. (1995) Generation of partial medial axis for disassembly motion planning. *Journal of Design and Manufacturing*, **5**, 89–100.
- [15] Kroll, E., Beasley, B., Parulin, A. and Berners, D. (1994) Evaluating ease-of-disassembly for product recycling. *ASME Materials and Design Technology*, **62**, 165–172.
- [16] Lowe, A.S. and Niku, S.B. (1995) A methodology for design for disassembly. *ASME Design for Manufacturability*, **81**, 47–53.
- [17] Lee, K. and Gadh, R. (1996) Destructive disassembly to support virtual prototyping. *IIE Transactions*, **30**(10), 959–972.
- [18] Requicha, A.A.G. and Whalen, T.W. (1991) Representations for assemblies. *Computer-Aided Mechanical Assembly Planning*, (L.S. Homem de Mello, (ed)), Kluwer Academic Publishers, Boston, MA, 15–39.
- [19] Turner, J.U. (1991) Relative positioning of parts in assemblies using mathematical programming. *Computer Aided Mechanical Assembly Planning*, (L.S. Homem de Mello, (ed)), Kluwer Academic Publishers, Boston, MA, 111–126.
- [20] Stoer, J. and Witzgall, C. (1970) *Convexity and Optimization in Finite Dimensions I*, Springer-Verlag, Berlin, Germany.
- [21] Chazelle, B. (1993) An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, **10**, 377–409.
- [22] Latombe, J.C. (1991) *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA.
- [23] Preparata, F.P. and Shamos, M.I. (1985) *Computational Geometry*, Springer-Verlag, New York, USA.
- [24] Kepler, A. (1987) Lower bound on the complexity of the convex hull problem for simple polyhedra. *Information Processing Letters*, **25**, 159–161.
- [25] Muller, D.E. and Preparata, F.P. (1978) Finding the intersection of two convex polyhedra. *SIAM Journal of Computing*, **21**, 671–696.
- [26] Stefan, H., Mantyla, M., Kurt, M. and Jurg, N. (1984) Space sweep solves intersection of convex polyhedra. *Acta Informatica*, **21**, 501–519.
- [27] Dobkin, D. and Kirkpatrick, D.G. (1985) A linear algorithm for determining the separation of convex polyhedra. *Journal of Algorithms*, **6**, 381–392.
- [28] Chazelle, B. (1992) An optimal algorithm for intersecting three-dimensional convex polyhedra. *SIAM Journal of computing*, **21**, 671–696.

Biographies

N. Shyamsundar is a Ph.D. candidate in the Department of Mechanical Engineering at the University of Wisconsin-Madison. He has an M.S. from the Indian Institute of Science, India, and a B.S. from the College

of Engineering, Madras, India. His research interests include design for disassembly, assembly representation, geometric modeling, computational geometry, computer graphics and WWW-based technologies.

Rajit Gadh is Associate Professor of Mechanical Engineering at the University of Wisconsin-Madison and Director of the CAD-IT Consortium. Professor Gadh obtained his Ph.D. from Carnegie Mellon University (1991), his M.S. from Cornell University (1986), and his B.S. from Indian Institute of Technology, Kanpur (1984). He was a visiting researcher at The University of California, Berkeley for a year. His research interests are in virtual reality-based shape design, shape

abstractions for feature extraction in automatic meshing and virtual prototyping, and geometry-based disassembly abstractions. His research is supported by government agencies such as NSF, DOE and EPA, and several fortune 500 companies such as Ford, ALCOA, Caterpillar, Texas Instruments and Lucent Technologies. He is the recipient of the NSF CAREER Award (1995), the NSF/Lucent Technologies Industrial Ecology Fellowship (1997), and the Society of Automotive Engineers' Ralph R. Teetor Research and Educational award (1993).

Contributed by the Engineering Design Department.