

TECHNICAL RESEARCH REPORT

A Feature Based Approach to Automated Design of Multi-Piece Sacrificial Molds

*by Savinder Dhaliwal, Satyandra K. Gupta, Jun Huang,
Malay Kumar*

T.R. 2000-23



ISR develops, applies and teaches advanced methodologies of design and analysis to solve complex, hierarchical, heterogeneous and dynamic problems of engineering technology and systems for industry and government.

ISR is a permanent institute of the University of Maryland, within the Glenn L. Martin Institute of Technology/A. James Clark School of Engineering. It is a National Science Foundation Engineering Research Center.

Web site <http://www.isr.umd.edu>

A Feature Based Approach to Automated Design of Multi-Piece Sacrificial Molds

Savinder Dhaliwal, Satyandra K. Gupta¹, Jun Huang, Malay Kumar

Mechanical Engineering Department and Institute for Systems Research

University of Maryland
College Park, MD 20742

Abstract

This paper describes a feature-based approach to automated design of multi-piece sacrificial molds. We use multi-piece sacrificial molds to create complex 3D polymer/ceramic parts. Multi-piece molds refer to molds that contain more than two mold components or subassemblies. Our methodology has the following three benefits over the state-of-the-art. First, by using multi-piece molds we can create complex 3D objects that are impossible to create using traditional two piece molds. Second, we make use of sacrificial molds. Therefore, using multi-piece sacrificial molds, we can create parts that pose disassembly problems for permanent molds. Third, mold design steps are significantly automated in our methodology. Therefore, we can create the functional part from the CAD model of the part in a matter of hours and so our approach can be used in small batch manufacturing environments.

The basic idea behind our mold design algorithm is as following. We first form the desired gross mold shape based on the feature based description of the part geometry. If the desired gross mold shape is not manufacturable as a single piece, we decompose the gross mold shape into simpler shapes to make sure that each component is manufacturable using CNC machining. During the decomposition step, we account for tool accessibility to make sure that (1) each component is manufacturable, and (2) components can be assembled together to form the gross mold shape. Finally, we add assembly features to mold component shapes to facilitate easy assembly of mold components and eliminate unnecessary degree of freedoms from the final mold assembly.

1 Introduction

Molds are required in a large number of manufacturing operations such as metal casting, die making, injection molding, ceramic and polymer processing etc. Molded and cast parts are used extensively because they produce net-shape parts that require minimal secondary operations. On the basis of the number of pieces in a mold, molds can be divided into the following two categories:

1. *Two Piece Molds*: These molds consist of two pieces, a mold top and bottom. The two mold halves are separated at the surface known as the parting surface. Two piece molds are the most commonly used molds because they are easy to design and manufacture. However, they cannot be used to produce complex parts. Since there is only one parting surface in a two-piece mold, it imposes restrictions on the kind of parts that can be cast in such molds. Two piece molds often require cores to handle undercuts in parts. Undercuts are projections or recesses in the component which are unfavorably inclined with respect to the draw vector, and which hinder the removal of the part from the mold. The draw vector is the direction in which the mold top is removed. **Figure 1** shows a part having an undercut. The shaded part is the undercut and it hinders the removal of the part from the mold.
2. *Multi-Piece Molds*: These molds refer to molds having more than two pieces. These molds can produce complex parts that cannot be made using two piece molds. They enable the use of molding for making parts that were previously manufactured using other processes. Since they have more than two pieces, multi-piece molds have more than one parting surface. This enables the mold to be decomposed along different directions and thus can be used to make complex parts. **Figure 2(a)** shows an example of a part that requires a multi piece mold. This part consists of three sections, a rotor, a cylindrical connector, and a top piece.

¹ Author to whom all correspondence should be addressed.

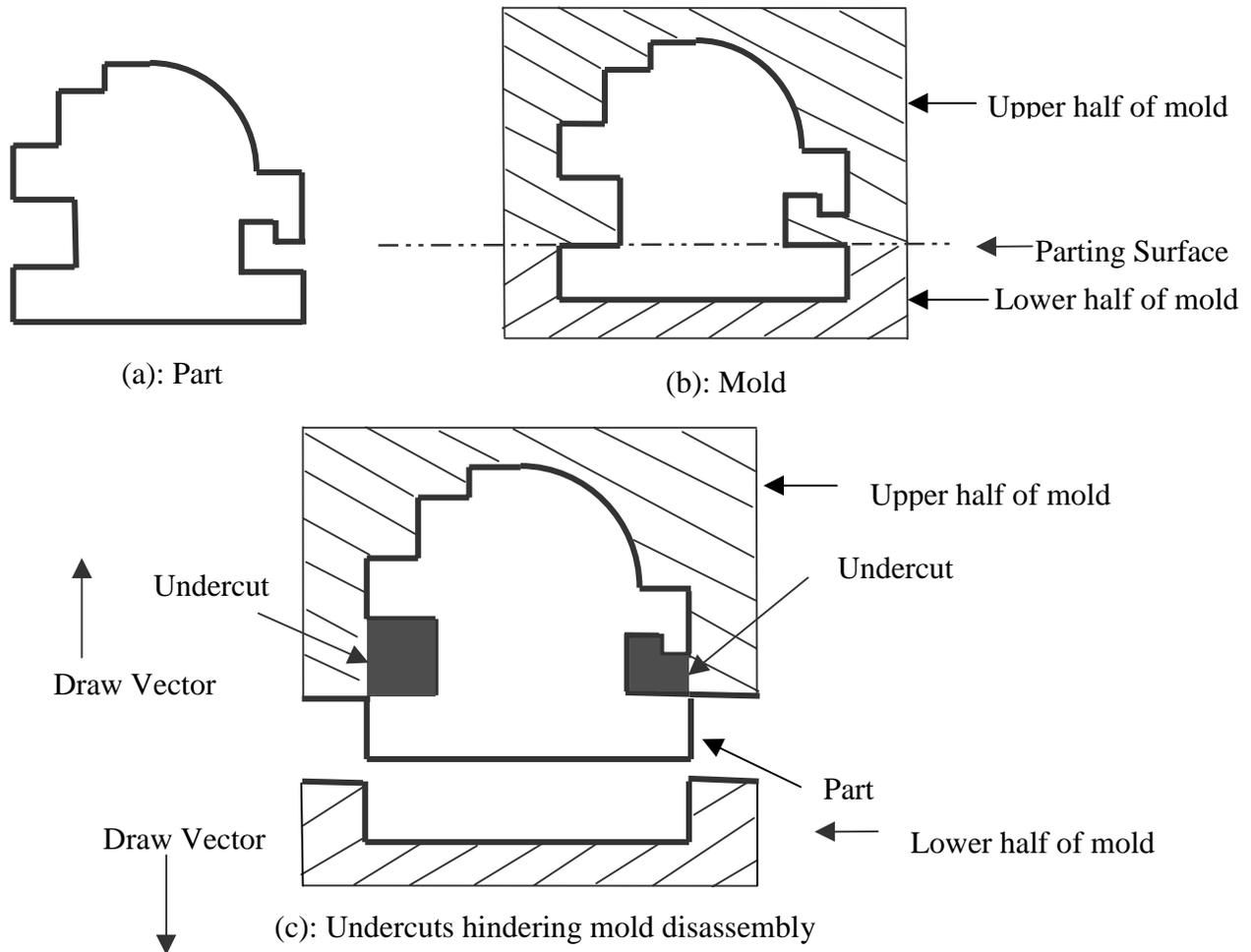


Figure 1: A part having an undercut

The mold component for the rotor can be removed only in the $-Z$ direction since the blades restrict the motion of the mold in any other direction. However the mold for the part top would need to be partitioned in two parts since there are two cylinders that need to be removed in opposite directions (X and $-X$ directions). Thus this part cannot be produced using a two piece mold. A possible multi piece mold for the part is shown in **Figure 2(b)**. This consists of a three piece mold, a mold bottom for the rotor that is removed in the $-Z$ direction, and two identical mold parts for the remaining part, one that is removed in $+X$ direction and the other that is removed in $-X$ direction.

Even though multi-piece molds enable the production of complex parts, they still present the problem of mold disassembly. After the part has been cast, the mold has to be separated from the part. Though it is generally not a problem for temporary mold castings such as sand casting, it does pose problems for permanent mold castings. **Figure 3** shows a part that cannot be cast using a permanent mold since mold for the cavity cannot be disassembled from the part once it is cast. This is because the mold components cannot be removed from the part. Sacrificial molds can be used to circumvent this problem. Sacrificial molds refer to molds that can be destroyed after the part has been produced. They are generally made of low melting point materials such as wax or ABS and are typically destroyed by heating the mold-part assembly. Moreover, the wax molds can be easily machined making them very easy to manufacture at high production rates. Sacrificial multi-piece molds find use in a number of manufacturing domains. Examples include:

1. *Manufacture of polymer parts:* They are used for making polymer parts. Polymer parts made up of materials such as polyurethanes solidify at room temperatures. They have typically a higher melting point than wax used for making the mold. Since the quality of the parts is not dependent on the material properties of the mold (e.g., porosity of mold), sacrificial molds made of wax provide an excellent alternative to traditional molds.
2. *Gelcasting of ceramic parts:* Another application domain for sacrificial multi-piece molds is gelcasting. Gelcasting is emerging as a popular method for making high performance ceramic parts for a wide variety of

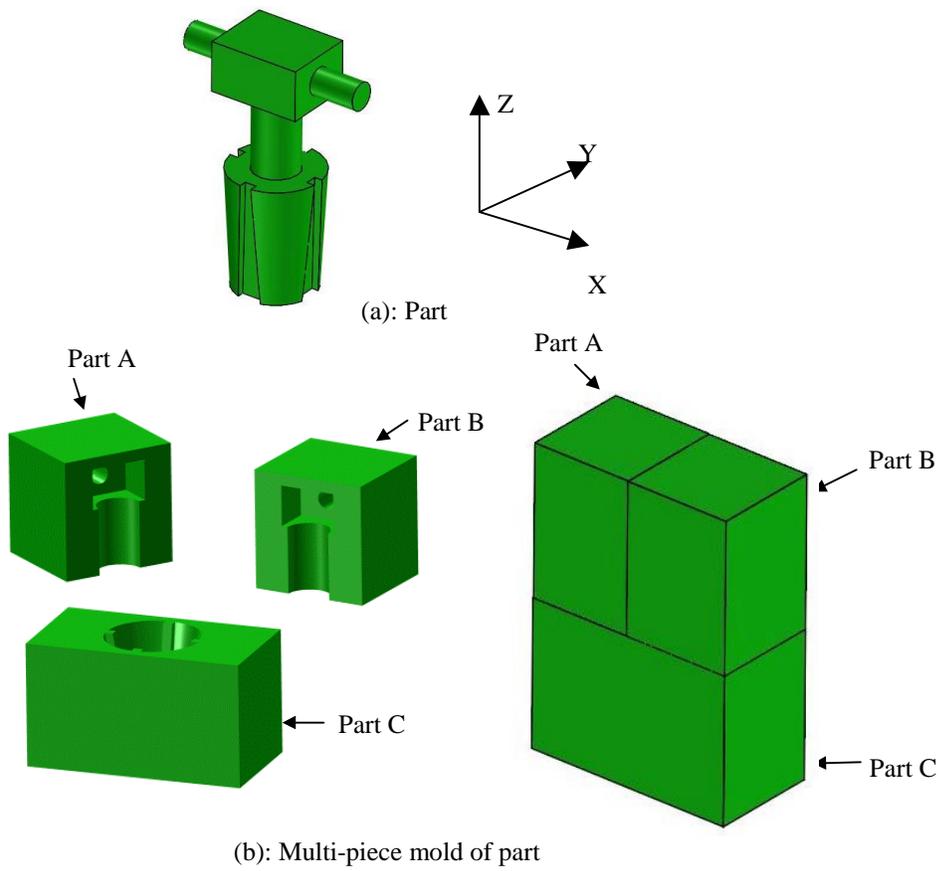


Figure 2: Part with multi-piece mold

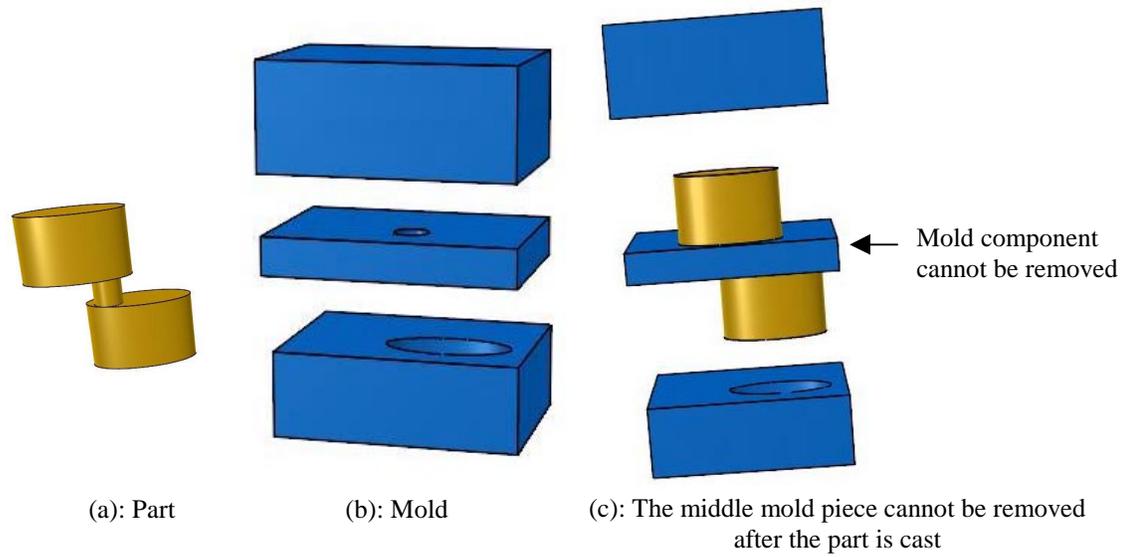


Figure 3: A part and its multi piece mold that cannot be disassembled after the part has been cast

aerospace, automotive, and industrial applications. Gelcasting produces large, complex shaped parts that are strong enough to be machined if necessary. The process is simple, economical and uses conventional equipment. Low melting temperatures in gelcasting enable use of sacrificial molds.

This paper describes a feature based approach to automated design of multi-piece sacrificial molds. The basic idea behind our mold decomposition algorithm is as following. We first form the desired gross mold shape based on the

feature based description of the part geometry. If the desired gross mold shape is not manufacturable as a single piece, we decompose the gross mold shape into simpler shapes to make sure that each component is manufacturable using CNC machining. During the decomposition step, we account for tool accessibility to make sure that (1) each component is manufacturable, and (2) components can be assembled together to form the gross mold shape. Finally, we add assembly features to mold component shapes to facilitate easy assembly of mold components and eliminate unnecessary degree of freedoms from the final mold assembly.

The remainder of this paper has been organized in the following manner. Section 2 reviews the related work in the area of automated mold decomposition. Section 3 presents the background and overview of approach. Section 4 describes the feature based decomposition algorithm. Section 5 describes the accessibility based decomposition algorithm. Section 6 describes the algorithm for reducing the mold components by eliminating over decomposition. Section 7 describes the algorithm for adding assembly features to the mold assembly. Section 8 discusses the system implementation and describes few example parts. Finally, Section 9 presents the conclusions of our research.

2 Review of Related Work

In the past, mold development has been a cumbersome, labor intensive process requiring a significant amount of subjective guesswork [Bern97]. Mold quality was normally determined by the skills of the designer. The designer used to analyze the part and then determine how the liquid would flow in the mold, depending on how the mold is gated and where the runners are placed. Nowadays, software tools are helping designers in the mold design process [Chen93a]. Software tools have been developed which can do the complete mold design for simple injection molded parts. Given a CAD model of the part to be cast, the output is a 3D model of the mold. In addition to this, these software can be used to calculate mold design parameters such as cooling time of the part, the draw area of the part, the ejection force, and the part shrinkage. Part shrinkage calculations are especially useful because they eliminate the need to produce shrink corrected drawings. Most of the work related to the integration of CAD with mold design has been for solid modeling, addition of shrink factors, mold base design and flow analysis for gating design. The simulation of the plastic flow in the molding process gives indications as to the pressure and temperature distributions and the cycle time, which is required for the determination of the gating, runner and cooling system to be used. The use of a standard solid modeler for the definition of libraries of runners, gates and mold plates has also been reported. It is also possible to generate the tool paths for NC manufacture of the mold. Other features of mold design software include the ability to create mirror designs for balanced molds or to create multiple copies of the mold geometry for multi-cavity applications. They also incorporate structural, thermal and hydraulic considerations into the design process.

For the sake of brevity, we only present the status of current research in the field of mold decomposition. In Section 2.1, we describe the parting surface selection for two piece molds. Parting surfaces determine the locations where the mold is decomposed. In Section 2.2, we discuss about automated decomposition of multi-piece molds.

2.1 Parting Surface Selection for Two-Piece Molds

One of the most significant design aspects that greatly influence mold manufacturing costs is the choice of the surface separating the two halves of the mold or die, referred to as the parting surface. The selection of the parting surface and the parting directions is important since it affects all subsequent steps in the design of the mold.

Most of the work done in the past is concentrated on parting line selection for two piece permanent molds. Due to this, the presence of undercuts plays a significant role in determining the parting line. Some of the researchers have discussed the cases in which cores are incorporated into the molds to take care of undercuts. Since we are interested in developing multi piece sacrificial molds, undercuts do not pose a problem. The primary concern is the accessibility of each of the individual mold components for machining. However, parting surface design approach provides valuable insight into the mold design process. In the remaining part of this section, we discuss different approaches for mold parting surface selection.

Ravi and Srinivasan [Ravi90] have developed a set of nine optimization functions that determine the best location for a specified draw direction. Some of the principal optimization functions are maximizing the projected area on the parting plane, selecting a parting plane having the highest degree of flatness, minimizing the draw, minimizing the draft, minimizing the volume of undercut, maximizing dimensional stability and minimizing flash. The parting directions considered by them consist of only the principal axis.

Hui and Tan [Hui92] define an optimum parting surface as one that minimizes the number of side cores and also has the minimum area in shear contact between the mold plate and the molded part. They have considered the normals of planar surfaces and the axes of cylindrical surfaces as possible parting directions.

Chen *et al.* [Chen93b] aim to find a pair of parting directions, which minimizes the number of required cores. They divide a part into convex and concave regions and have used the notion of visibility maps to determine the spherical polygons that represent the set of directions from which each region is accessible. A common intersection of all these spherically convex polygons is used to determine whether a side core is needed or not. If the intersection is non-null, this implies that a direction exists which results in a two-piece mold. If the intersection is null, a direction is selected that lies in the maximum number of polygons.

Weinstein and Manoochehri [Wein96, Wein97] build upon the work of Chen *et al.* They also use visibility maps to determine if a feasible direction exists that results in a two-piece mold. If no such direction exists, then a parting direction is selected which minimizes the objective function. The objective function is defined as a function of the flatness of the parting line, draw depth, number of undercuts, number of side cores required to form the undercuts, and the machining complexity. They have defined a constraint set which consists of the following constraints: a minimum draft angle for each surface, preventing flash from designated surfaces and maintaining critical dimensions. Their method can also be used to incrementally calculate the draw direction range and parting line location for a part, as features are added during the design process. This allows the designer to modify the features during the design process and obtain an acceptable draw direction and parting line location.

2.2 Automated Decomposition of Multi-Piece Molds

Krishnan [Kris97] describes automated two-piece and multi-piece mold design for injection molding. The part is constructed by stacking 2.5D primitives called C-entities along the Z direction through either a Constructive Solid Geometry (CSG) or Destructive Solid Geometry (DSG) operation. A C-entity is manufacturable if there are no thin walls created by the shape of the island and cavity profiles and there are no thin walls created by the position of the cavity profile with respect to the island profile. Each entity also has an accessibility attribute that is calculated with respect to its parent entity. The accessibility attribute is used to determine whether a two piece mold can be used. If a two piece mold cannot be used to make the injection mold part, it is checked whether a multi-piece mold can be used or not. A multi-piece mold is defined as one that has two or more pieces, and the direction of separation of the mold components is orthogonal to the Z direction, i.e. the direction in which the part was created. The mold separation is restricted to the X and Y direction.

Krishnan's research is in automating two-piece and multi-piece mold design for injection molding. Since the injection molds are permanent molds, disassemblability of mold plays an important factor in determining the parting surface and parting direction. Moreover, since the primitives considered are only 2.5D solids that are stacked along the Z direction, the complexity of the part is limited. The parting surface directions are also constrained to be along the X axis direction or the Y axis direction.

3 Background and Overview

3.1 Feature Based Part Representation

In a feature based representation of a part, the part is constructed from a set of primitives. The primitives can be selected from either a pre-defined library of shapes or they may be user-defined primitives subject to the shape constraints described in Section 3.2. The pre-defined library consists of basic primitives such as a rectangular block, cylinder etc. These primitives are created at the origin of the reference frame. A transformation matrix is provided to transform the primitives to their respective positions. The transformations supported are rotation and translation transformations.

The part can be represented by a tree structure in which the nodes represent the primitives while the edges denote the relationship between the primitives. The relationship can be a regularized Boolean union or subtraction operation. The relationship is represented by a directed line segment whose head points to a child primitive (i.e. a primitive that is added to or subtracted from another primitive) and the tail points to the parent primitive (i.e. a primitive to which a child primitive is added or subtracted). **Figure 4** shows an example part and **Figure 5** shows the feature tree for the part.

3.2 Problem Formulation

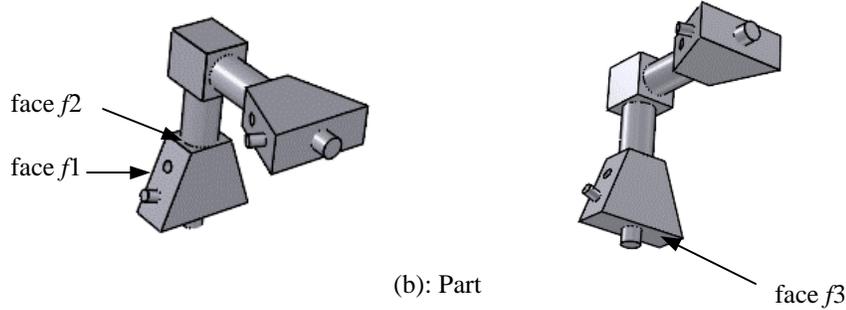
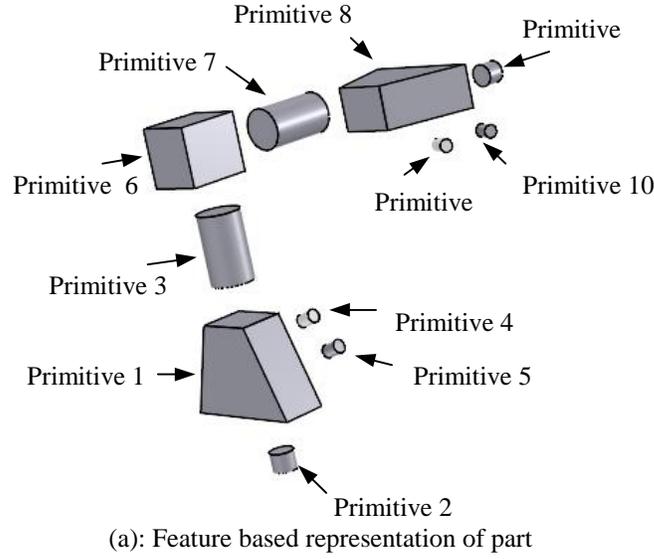


Figure 4: Feature based representation of part

This paper describes an algorithm for doing multi-piece sacrificial mold design. The input is a feature based representation of the part. The part is constructed by sequentially combining the primitives in the order specified in the input file. The primitives can either be added to or subtracted from their parent primitive. The primitives consist of either a solid from a pre-defined library of shapes or any solid represented in the boundary representation that satisfies the restrictions imposed in Section 3.3.

The output is an N piece assembly $M = \{m_1, m_2, \dots, m_N\}$ of solid models of mold components where each mold component is represented by its boundary representation. M needs to satisfy the following conditions:

1. Each $m_i \in M$ is a connected solid.
2. $M_S = \bigcup_{i=1}^N m_i$ such that the internal shell of M_S corresponds exactly to the boundary of the part.
3. Boundary of each $m_i \in M$ is completely accessible to a cutting tool.

3.3 Restrictions/Assumptions

The following restrictions are imposed on the feature based representation of the part:

1. The primitives should consist of at least one planar face along which they are added to or subtracted from the parent primitive.
2. The primitives do not share more than one planar face with their parent.
3. After the part has been constructed from its feature based representation, it does not have any internal shells and has only one lump. A shell is a set of connected faces while a lump represents a connected solid.

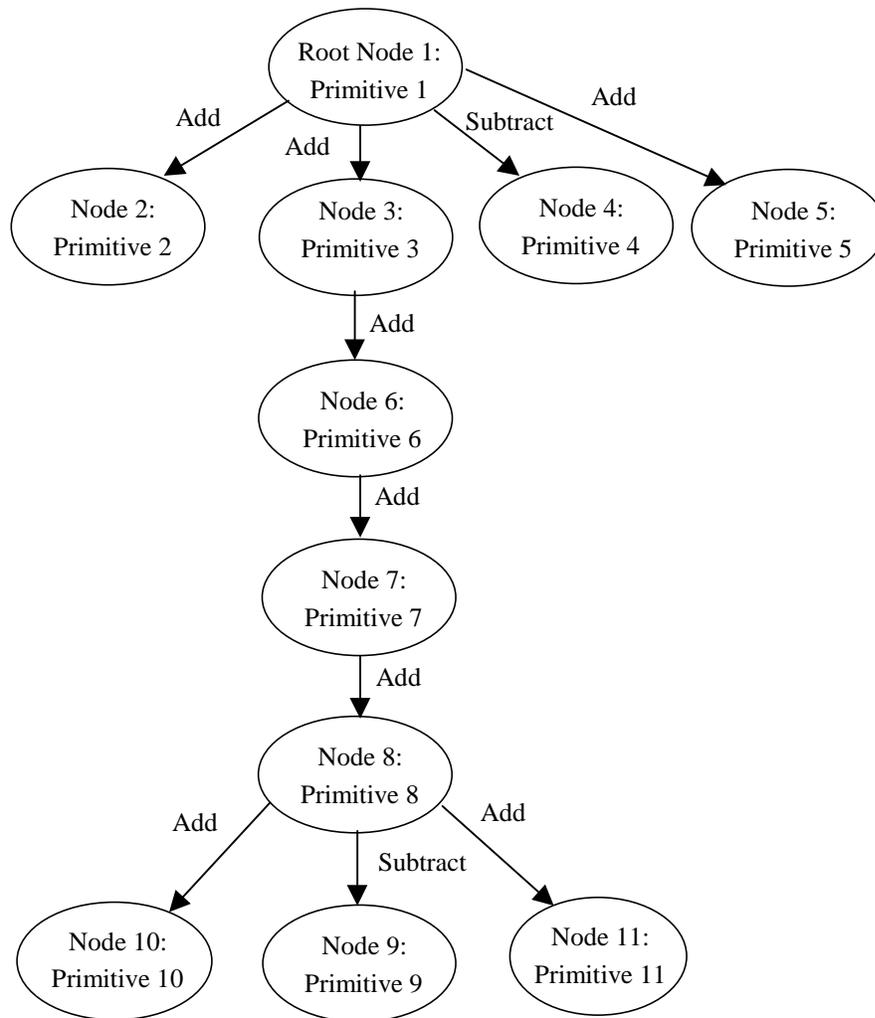


Figure 5: Feature tree for part

4. All the subtractive features can only be leaf nodes in the feature based representation.

3.4 Overview of Mold Decomposition Algorithm

This section discusses the approach followed in doing mold decomposition. It can be divided into six steps:

1. *Preprocessing*: As described earlier, each part consists of a feature based representation of the part. As a first step a validation check is performed on all the primitives to ensure that they satisfy the restrictions discussed in Section 3.3. If the feature based representation is valid, the part is scaled uniformly so that the cavity size of the mold accounts for the shrinkage in the part. Then the gross mold is created by subtracting the scaled part from a large rectangular block that completely encloses the part.
2. *Feature Based Decomposition*: A feature based decomposition of the mold is done to obtain individual mold components for each of the primitives constituting the part. All decompositions are performed along planar faces. This is a problem simplification process since it is easier to do geometric reasoning on individual mold components for primitives as compared to the gross mold. Section 4 discusses the algorithm for performing feature-based decomposition.
3. *Accessibility Based Decomposition*: Once the feature based decomposition is completed, some of the individual mold components may need to be further decomposed due to accessibility constraints. These mold components

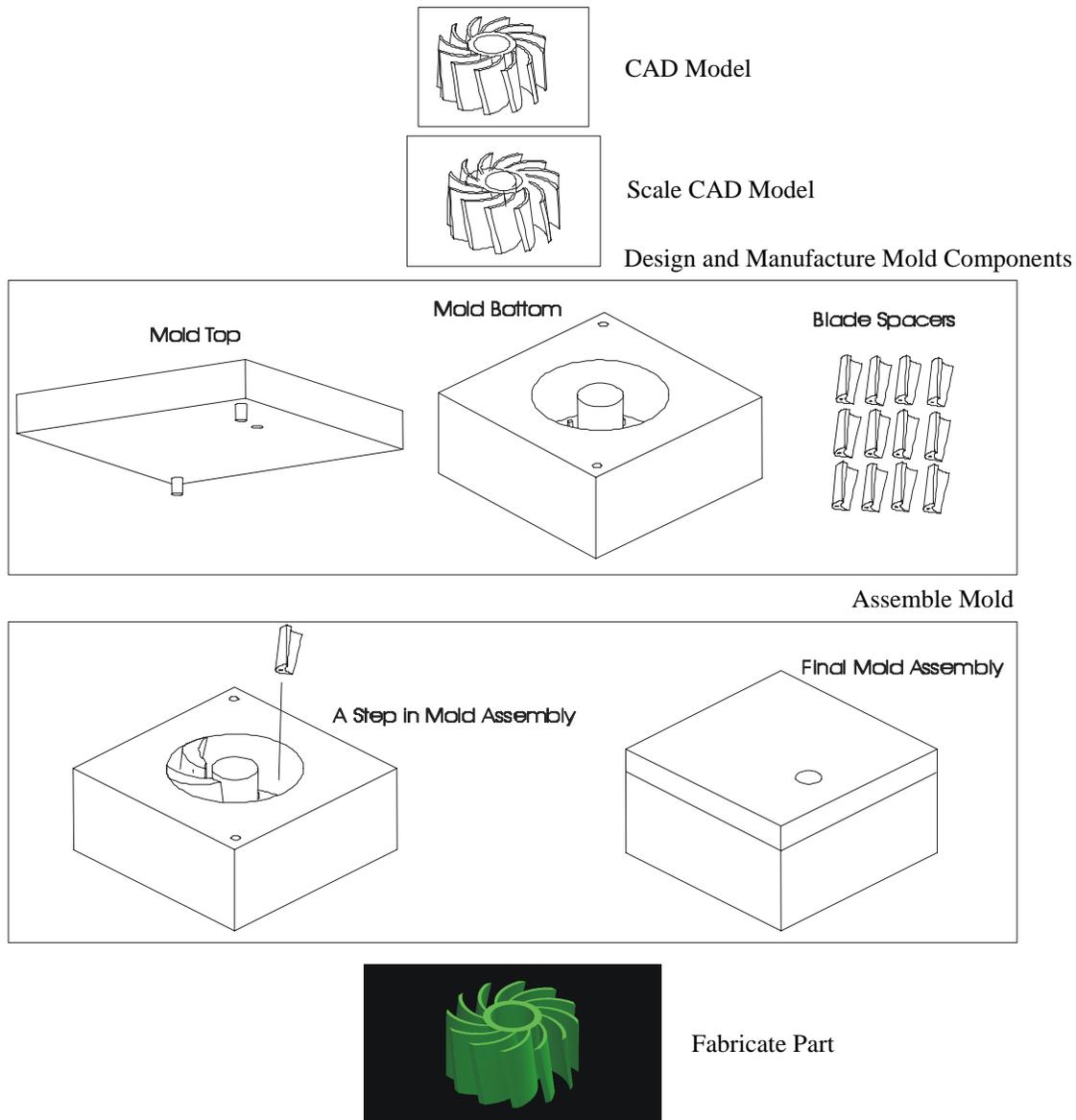


Figure 6: Steps in mold design and part fabrication

may not be completely accessible to the cutting tool in the specified set of directions. An accessibility check is performed on all the mold components and further decomposition is done for mold components that are not completely accessible. Section 5 discusses the algorithm for performing accessibility-based decomposition.

4. *Recombination to Reduce Mold Components:* Once the decomposition has been completed, some of the individual mold components may be combined if the resulting mold component is completely accessible. This is achieved by doing the accessibility analysis for each possible combination. The list of possible combinations consists of all those mold components that share a common face. Section 6 discusses the algorithm for combining mold components.
5. *Addition of Assembly Features:* Once the mold recombination is completed, assembly features are added to the mold components in the mold assembly.
6. *Postprocessing:* After the mold assembly for the part has been designed, the user has to select a mold component for creating the sprue. A sprue is a passage through which the liquid material is poured into the mold enclosure.

Figure 6 shows an overview of the mold design and fabrication process.

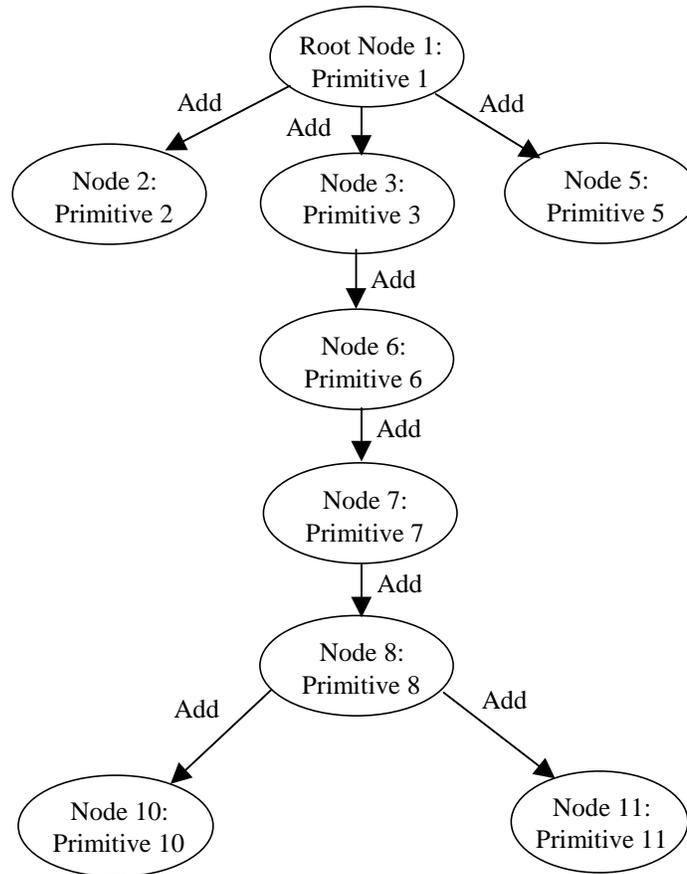


Figure 7: Feature tree for part with subtractive nodes removed

4 Feature Based Decomposition

A preprocessing step is performed before the feature based decomposition of the mold is initiated. It consists of the following steps:

1. *Verification:* The feature based representation of the part is checked to determine if it is a valid representation or not. This step ensures that the feature based representation does not violate the restrictions imposed on the input that are discussed in Section 3.3.
2. *Initialization:* The boundary representation of the part is constructed using the feature based representation of the part.
3. *Scaling of Part:* After the boundary representation of the part is created from its feature based representation, the part is scaled so that the cavity size of the mold accounts for the shrinkage in the part. We use isotropic scaling factors to scale the part. If P represents the point set corresponding to the solid model of the part and α is the scaling factor, the new part is given by αP .

The input to the algorithm is a feature based representation of the part. This feature based representation is used to build the feature tree for the part. This feature tree can be represented by a double (N, E) where N is a set of nodes in the feature tree and E represents the set of edges. The output is a set of mold components M such that each mold component represents the solid model of a primitive.

The feature based decomposition process can be divided into the following steps:

1. *Removing Subtractive Features from the Feature Tree:* All the nodes representing subtractive primitives are removed from the feature tree. This is done because all the subtractive features form inserts in the mold and thus can be added after the mold decomposition is completed. **Figure 7** shows the feature tree for a part in which all the nodes that correspond to subtractive primitives are removed.

2. *Gross Mold Formation:* The scaled part is used to form its gross mold shape. We create the desired gross mold shape by subtracting the solid model of the scaled part from the solid model of the mold enclosure. The mold enclosure considered in our implementation is a rectangular solid model that completely encloses the scaled part. **Figure 8** shows the gross mold shape of the part shown in **Figure 4**. During this step we also initialize the set M that contains mold components after decomposition. M is initialized to contain one solid model that corresponds to gross mold shape of the part.

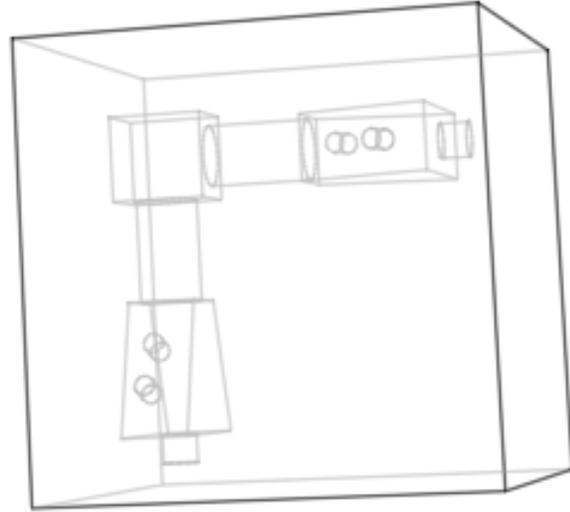


Figure 8: Gross mold of part

3. *Decomposition:* We need to construct a set T that contains all the feature trees in the data structure. Initially it contains only the feature tree of the part. While the set T contains a tree t that is not a primary tree (i.e. has more than one node), perform the following steps:
- Select the mold component m for the tree t from M .
 - Identify all candidate parting planes and sort them based on a priority criteria (Section 4.1 describes the procedure for identifying and sorting the parting planes).
 - Select the best candidate parting plane and check if it is a valid cutting plane (Section 4.2 presents the procedure for validating a candidate parting plane). If no valid parting plane can be found, the tree cannot be decomposed. The user is notified and the program terminates.
 - If a valid parting plane p is found, the mold is cut along the plane (The procedure for performing this step is described in Section 4.3).
 - If the valid parting plane is shared by more than two nodes, the mold components may need to be further decomposed (Section 4.4 describes the procedure for performing this step).
4. All the subtractive primitives are added to the set M after being scaled by the scaling factor α .

4.1 Identifying the Candidate Parting Planes

The candidate planes are identified and sorted in the following manner:

- The highest priority is given to the parting planes that are shared by exactly one leaf node and one other node. **Figure 9** shows an example of such a plane.
- If the candidate parting plane is shared by exactly two nodes and none of the nodes is a leaf node, then this parting plane is given the middle priority. This is shown in **Figure 10**.
- The lowest priority is given to a candidate parting plane that is shared by more than two nodes. **Figure 11** shows an example of such a plane.

If any of above described categories contain more than one plane, then the planes are arbitrarily sorted in that category.

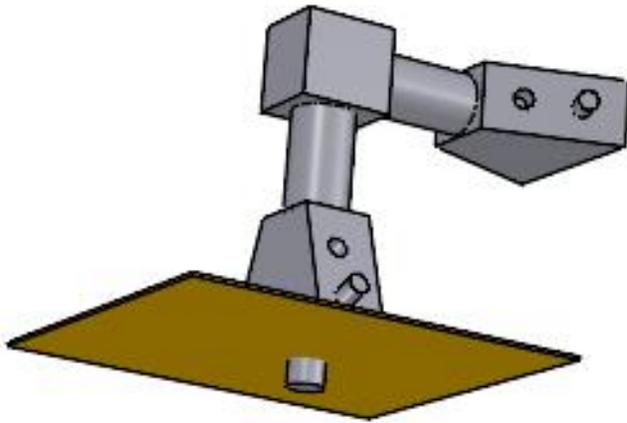


Figure 9: A cutting plane along face f_3 shared by two subparts, one of which is a primitive

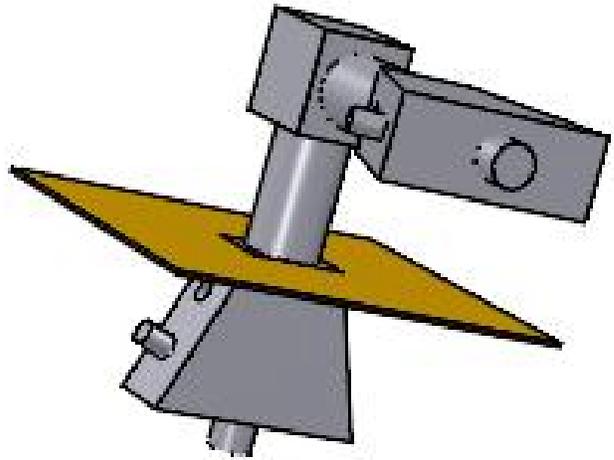
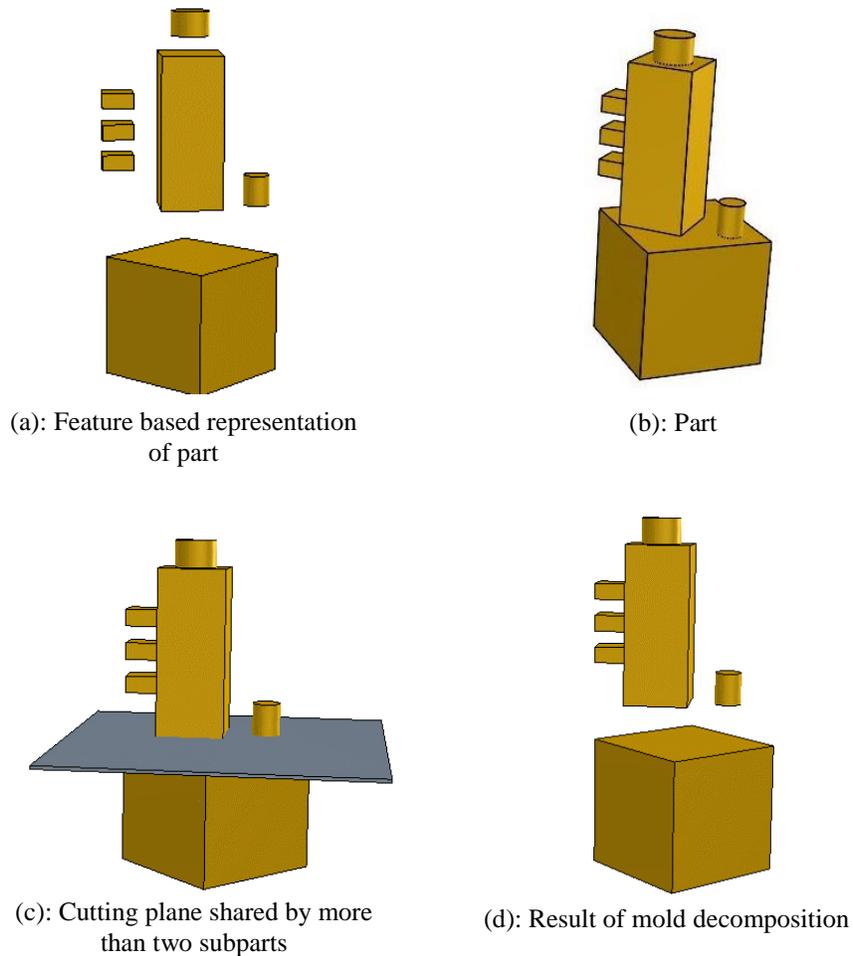


Figure 10: A cutting plane along face f_2 shared by two subparts, none of which is a primitive

4.2 Validating a Candidate Parting Plane

A candidate parting plane is a valid parting plane if its non-regularized intersection with all the primitives that do not share this plane is a null set. **Figure 12** shows an example of a valid and an invalid cutting plane. An invalid cutting



(a): Feature based representation of part

(b): Part

(c): Cutting plane shared by more than two subparts

(d): Result of mold decomposition

Figure 11: A cutting plane shared by more than two subparts

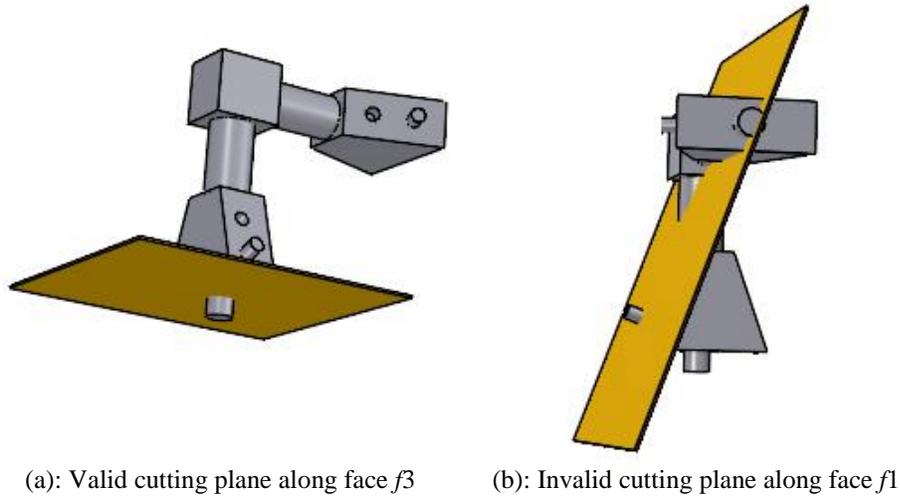


Figure 12: A valid and invalid cutting plane

plane may become a valid cutting plane at some later stage of mold decomposition. For example, the candidate plane in **Figure 12(b)** becomes a valid cutting plane after the part has been decomposed along the plane given in **Figure 13**.

4.3 Updating the Mold Component Set and Feature Trees

Once a valid cutting plane p cuts the mold component m , the current tree t that is being decomposed needs to be updated. This decomposition will also result in new trees that are added to the set of trees T . The following steps need to be performed:

1. m is cut into two mold components that are added to M while the mold m is removed from M .

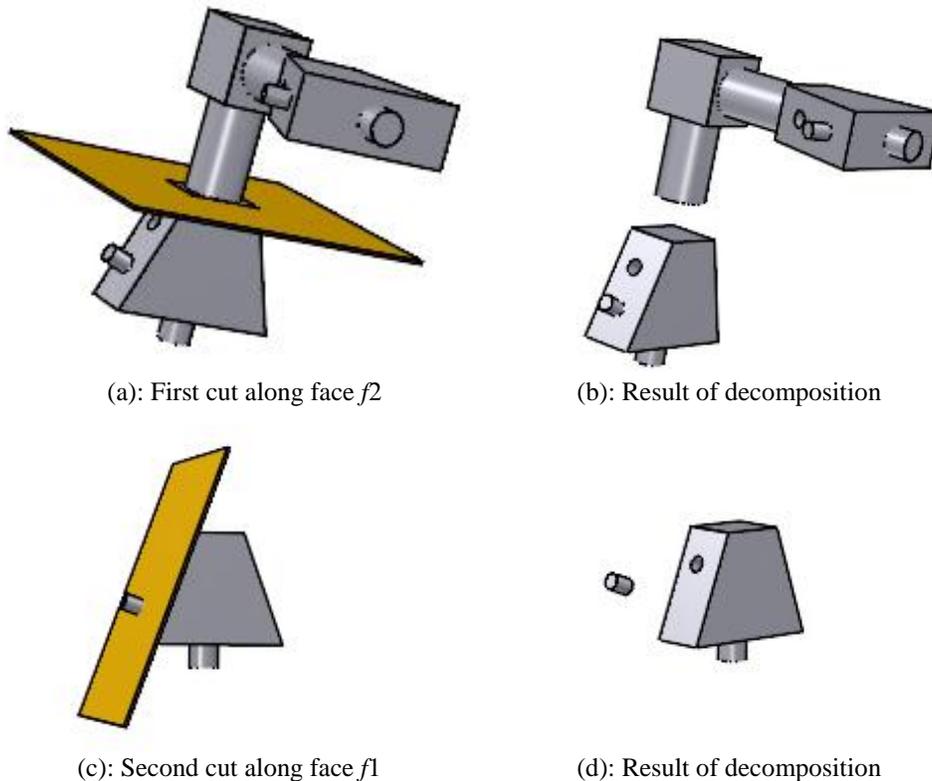


Figure 13: An invalid cutting plane becomes valid after cutting along another plane

2. If the parting plane is shared by exactly two nodes, identify the parent and child nodes. The child node is removed from the list of children nodes for the parent node. Create a new tree with the child node as its root node and add it to T .
3. If the parting plane is shared by more than two nodes, construct a set G that contains all these nodes. Identify the parent node that will still belong to the original tree t after the decomposition. Remove from the list of children nodes for this node all those nodes that lie in G . For all the other nodes in G do the following:
 - Remove from the list of children nodes for this node all those nodes that lie in G .
 - Create a new tree with this node as its root node and add it to T .

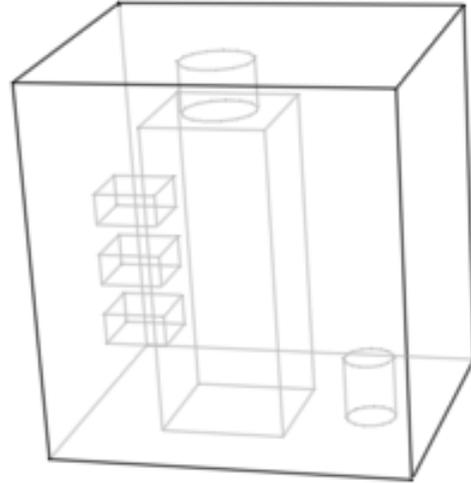


Figure 14: A mold component containing two trees

The mold component(s) that contains multiple trees may need to be decomposed further. This procedure is described in the next section.

4.4 Decomposition of Mold Components Containing Multiple Trees

If the cutting plane were shared by more than two nodes then decomposition along this plane would split the mold m into two components, both of which may have multiple trees. **Figure 14** shows such a mold component. These mold components need to be decomposed further so that each individual mold component has only one tree. If this parting plane were used to decompose the part, it would separate the part into more than two disconnected solids called lumps. This is shown in **Figure 11**. The lumps are divided into two sets: one set corresponds to one of the mold components and the other set corresponds to the other mold component. The remaining steps are performed for both the sets consecutively.

- a. Construct a set C_h that contains the convex hull of all the lumps belonging to one set of lumps. For every pair of convex hulls (c, c') where $c, c' \in C_h$ and $c \neq c'$, compute $I = c \cap c'$. If $I \neq \emptyset$, the mold component cannot be decomposed. In such a case terminate the program.
- b. Find a parting plane such that a pair of convex hulls (c, c') lie on opposite sides of this plane. Megiddo [Megi83] describes a linear time algorithm for computing such a plane using the approach of linear programming.
- c. It may be possible that a parting plane computed in the previous step might intersect another convex hull. Therefore a greedy approach is followed in which the first parting plane that does not intersect any of the convex hulls is used to decompose the mold components into sub-components. This decomposition would again divide the mold component into two sub-components each of that may be having more than one cavity. This plane divides the set of convex hulls in to two subsets lying on opposite sides of the plane. Steps *a* to *c* are repeated for each of these subsets till each mold component has only one cavity.
- d. All the mold components are added to M and mold m is removed from M .

In our implementation, we have made the following simplifications in the above described procedure:

- For faster computing, the convex hull of the lump is approximated by its spatial bounding box. Each bounding box results in a rectangular block with its sides along the principal axis.
- Since the bounding boxes have their faces along the principal axis, the candidate parting planes are assumed to be parallel to either to the $X = 0$ plane or $Y = 0$ plane. Thus the linear programming approach is not required to find the candidate planes.

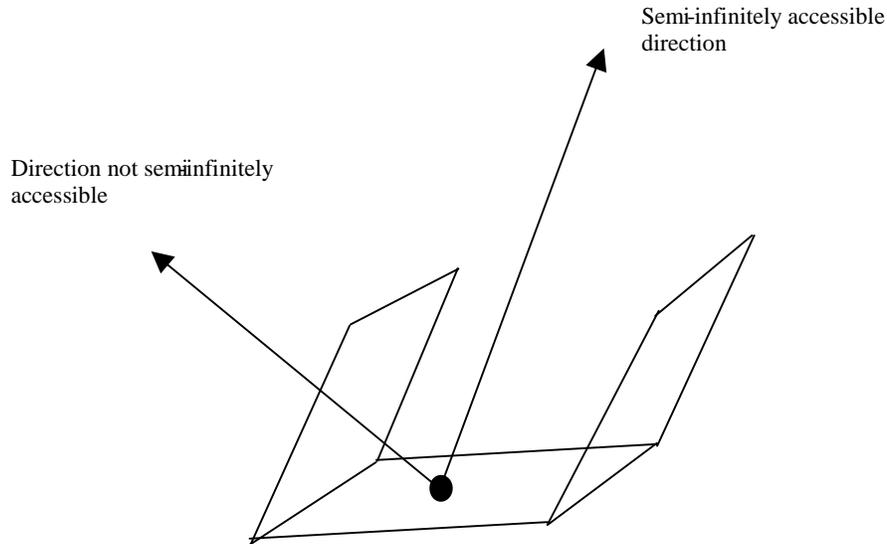


Figure 15: Semi-infinite accessibility of a point

5 Accessibility Based Decomposition

In the previous section we presented an algorithm for doing feature based decomposition of the mold. The result of the feature based mold decomposition is a set of mold components where each mold component contains the cavity for a particular primitive of a part. An accessibility analysis has to be performed for individual mold components to ensure that they can be machined on a CNC machine. If some mold components are not completely accessible, further decomposition based on accessibility will be required. We perform two stages of accessibility analysis. In the first stage, a global accessibility analysis of the mold components is done to ensure that each mold component is completely machinable by a cutting tool of semi-infinite length and zero diameter. Section 5.1 describes the procedure for detecting accessibility problems based on the semi-infinite accessibility criteria and Section 5.2 presents the accessibility based decomposition algorithm for alleviating semi-infinite accessibility problems. The second stage of accessibility based decomposition involves detecting accessibility problems due to non-zero cutting tool diameter. Section 5.3 describes the procedure for detecting accessibility problems due to non-zero cutting tool diameter while Section 5.4 presents the algorithm for accessibility based decomposition for alleviating accessibility problems due to a non-zero tool diameter.

5.1 Detecting Accessibility Problems Based on Semi-Infinite Accessibility Criteria

5.1.1 Definitions of Semi-Infinite Accessibility

- *Semi-infinite accessibility of a point in a given direction:* A point belonging to a geometric entity is said to have semi-infinite accessibility in a given direction if a ray of semi-infinite length can be drawn from it in the given direction without intersecting any other part of the geometric entity. **Figure 15** illustrates the concept of semi-infinite accessibility graphically. If the cutting tool is assumed to be of semi-infinite length and zero radius, then the direction from which a point is accessible represents a direction from which it is machinable.
- *Semi-infinite Accessibility of a face in a Given Direction:* A face is said to have semi-infinite accessibility in a given direction if all the points constituting the face have semi-infinite accessibility in the given direction.
- *Global accessibility cone of a point:* Global accessibility cone of a point represents the set of unit vectors along which the point has semi-infinite accessibility.
- *Global accessibility cone of a surface:* Global accessibility cone of a surface represents the set of unit vectors along which the entire surface has semi-infinite accessibility.

5.1.2 A Procedure for Computing Exact Inaccessibility

Given two planar triangular facets f and f' , the set of directions from which f is inaccessible (and accessible) can be computed using the method given by Dhaliwal [Dhal00]. Once again, let the set of directions from which f is inaccessible due to f' be given by I and the set of directions from which f is accessible be given by V . The following procedure computes the set of directions from which f is inaccessible due to f' .

procedure INACCESSIBLE(f, f')

1. Construct unit spheres at the three vertices of the facet f . Project f' on these spheres and let the projections be denoted by f'_{P1}, f'_{P2} and f'_{P3} . If any vertex of f' lies at the center of sphere, then do not include that vertex in the projection.
2. f'_{P1}, f'_{P2} and f'_{P3} are spherical triangles (or arc if the two facets share vertices) on the sphere and would result in maximum of nine vertices on the unit sphere.
3. Find the convex hull C_V of the vertices computed in the previous step.
4. $I = C_V$.

(I is a convex spherical polygon representing the set of directions from which f is inaccessible due to f' .)

5.1.3 Overview of Generation of Global Accessibility Cones for Various Facets in an Object

In order to generate the global accessibility cones for various facets in an object, a facetised representation of the object is first created. For a polyhedral object, the facetised representation is its exact geometric representation. For a body having curved surfaces, the number of facets in a given facetised representation is dependent on the accuracy requirements specified and also the maximum facet size specified by the user. The higher the accuracy requirement, the greater the number of facets in the facetised representation. The object is now represented in terms of facets rather than faces. An object consisting of triangular facets has two types of facets: convex-hull facets and non-convex hull-facets. Convex-hull facets are defined in the following manner. We take the convex hull of the part. *Convex-hull facets* are those facets on the part that are also on the convex hull. All the facets on the part other than convex-hull facets are called *non-convex-hull facets*.

A convex-hull facet is always exactly accessible from the hemisphere created by using its direction normal as the pole. However, the shape of the global accessibility cone of a non-convex-hull facet is generally much more complex due to the blocking by other facets. In Section 5.1.2, we described a procedure for finding the exact inaccessibility region for a facet due to another facet. In this section we describe how that procedure can be used to identify the global accessibility cones for all the non-convex-hull facets in the object.

5.1.4 Finding the Global Accessibility Cones for Non-Convex-Hull Facets

We initially assume a non-convex-hull facet f is accessible from all the orientations described by the hemisphere H created by the facet's direction normal as its pole. Procedure INITIALIZE performs this initialization for all non-convex-hull facets. We use a data structure called *Access_Status* to represent global accessibility cones. *Access_Status* represents a matrix. Rows of this matrix represent spherical triangles. Columns of this matrix represent various facets. Various entries in the matrix describe whether a facet is accessible from a spherical triangle or not. Each entry in the matrix stores a set of facets due to which the given facet is inaccessible from the set of directions represented by the spherical triangle. For example if j^{th} facet is accessible from i^{th} spherical triangle, entry that corresponds to i^{th} row and j^{th} column contains a null set. On the other hand, if j^{th} facet is not accessible from i^{th} spherical triangle, the entry that corresponds to i^{th} row and j^{th} column contains the set of facets due to which this facet is inaccessible. We use procedure UPDATE to find a facet's inaccessibility region due to the influence of other facets in the set. Procedure UPDATE calls procedure INACCESSIBLE to find the inaccessibility region for various facets.

INITIALIZE takes the set of non-convex-hull facets F_n of the object as an argument. It decomposes the surface of a unit sphere into a finite number of spherical triangles. In the decomposition, each hemisphere is equally divided into τ strips, and each strip i is further decomposed into δ_i spherical triangles. The values of τ and δ_i are determined such that the areas of the spherical triangles are kept uniform, within solution accuracy.

Table 1: An example of initial matrix *Access_Status*

Facets Spherical triangles	1	2	3	4	5	6	7	8	...	N
1	{ ϕ }	{ ϕ }	{v}	{ ϕ }		{v}				
2	{v}	{ ϕ }	{ ϕ }	{ ϕ }	{ ϕ }	{v}	{v}	{ ϕ }		{ ϕ }
3	{ ϕ }	{ ϕ }	{ ϕ }	{v}	{v}	{v}	{ ϕ }	{ ϕ }		{v}
4	{ ϕ }	{v}	{ ϕ }	{v}	{ ϕ }	{v}	{v}	{ ϕ }		{v}
.										
.										
.										
M	{v}	{ ϕ }	{ ϕ }	{v}	{v}	{ ϕ }	{ ϕ }	{v}		{v}

INITIALIZE uniquely classifies the spherical triangles according to the initial accessibility status of each facet. This initial accessibility information is stored in matrix *Access_Status* and will be later updated by the iterative procedure UPDATE.

Inaccessibility region of a non-convex-hull facet can be determined by only considering the influence of other non-convex-hull facets in the object. This property exists because of the following reason. A semi-infinite ray emanating from a point within a non-convex-hull facet either does not intersect any facet in the object or intersects a non-convex-hull facet [Chen93b]. Therefore to determine the complete inaccessibility region of a non-convex-hull facet, it is sufficient to only examine the influence of other non-convex-hull facets in the object.

procedure INITIALIZE (F_n)

1. Form the set of spherical triangles S by dividing each hemisphere of a unit sphere into τ strips, and each strip i into δ_i spherical triangles. τ is selected by the user.
2. Create an $M \times N$ matrix *Access_Status*, where N is the number of the facets in F_n , M is the number of spherical triangles in S . Each entry of the matrix will indicate the accessibility status (a set of facets or a null set) of a facet f from a set of directions defined by a spherical triangle s . Initially, for a facet f , if a triangle s is contained by hemisphere $H(f)$, the corresponding entry in the matrix is initialized to null (ϕ) set; otherwise, the corresponding entry is initialized to a set containing a value v. v implies that the spherical triangle s is not contained by hemisphere $H(f)$ and thus the set of directions represented by s can never be used as an accessible direction. An example of initial matrix is shown in **Table 1**.

UPDATE takes two arguments, S and F_n . S is the set of spherical triangles on the surface of the unit sphere generated by INITIALIZE. F_n is the set of non-convex-hull facets in the object. For each pair of facets (f, f') in F_n , UPDATE calls procedure INACCESSIBLE to find the inaccessibility region of f on the spherical map due to f' . **Figure 16** illustrates an inaccessible region I overlaid on the set of spherical triangles.

Then, for every spherical triangle s whose corresponding entry for facet f in *Access_Status* does not contain the value

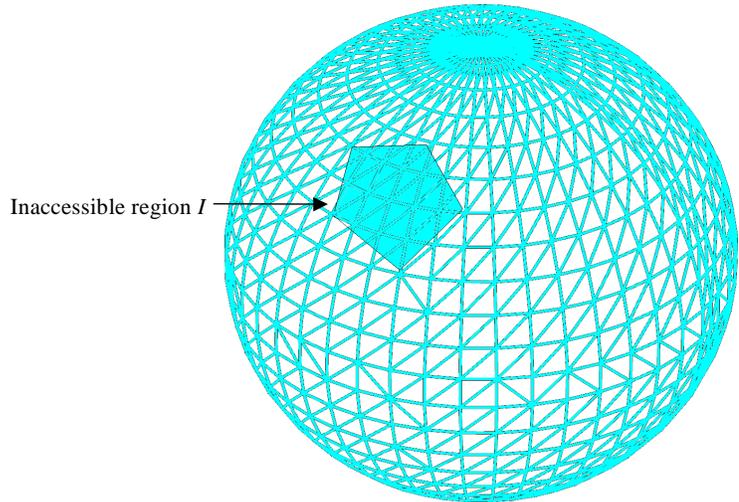


Figure 16: An inaccessible region I overlaid on a triangulated

Table 2: An example of updating matrix $Access_Status$

Facets Spherical Triangles	1	2	3	4	5	r...	N
1	{ ϕ }	{f3,f4,f5}	{ ϕ }	{ ϕ }	{f2,f3}		{v}
2	{f4,f2}	{f5,f3}	{f2,f7}	{v}	{f2,f3}		{fr,fr+1}
3	{f5,f2}	{f3,f4}	{v}	{f6,f7}	{fr,fr+1}		{fn-1,f7}
4	{ ϕ }	{v}	{ ϕ }	{v}	{v}		{v}
.							
.							
M	{ ϕ }	{ ϕ }	{v}	{v}	{ ϕ }		{fn-2,fn}

v (i.e. s is contained by $H(f)$), if the Boolean intersection of s with I is not a null set, update $Access_Status$ by adding f' in list of facets that make f inaccessible in the set of directions represented by s . Boolean operations on spherical triangles are performed by first converting spherical polygons to planar polygons using central projection [Lent49]. After this step we apply standard planar polygon intersection algorithms. **Table 2** is an example of an intermediate result while updating $Access_Status$ shown in **Table 1**. It shows the addition of facet f_3 in all those columns for facet f_5 where the Boolean intersection of I (obtained by calling $INACCESSIBLE(f_5, f_3)$) with s (where the corresponding entry does not include v) results in a not null set.

procedure UPDATE(S, F_n)

For every pair of facets (f, f') in F_n , do the following:

- Call $INACCESSIBLE(f, f')$, which returns I , the convex spherical polygon representing the inaccessibility region of f due to f' .
- For every spherical triangle s in S , if the corresponding entry for facet f in $Access_Status$ does not contain v , compute $C = I \cap s$. If $C \neq \phi$, update the entry for facet f in $Access_Status$ for the row s by adding f' to set of facets that make f inaccessible in set of directions represented by s .

5.2 Decomposition Algorithm to Alleviate Accessibility Problems

In the Section 5.1.4 we developed a procedure for computing the global accessibility cones for all the non-convex hull facets of an object. These cones can be used to analyze if an accessibility based decomposition of a mold component is required or not. The first step in this process would be to obtain the global accessibility cones for each mold component. The next step is to determine if all the facets for a mold component are accessible in some direction. In Section 5.2.1 procedure CHECKACCESSIBILITY describes the procedure for doing this test.

If there exist some facets in a mold component that are not accessible in any direction, the mold component has to be decomposed. A set of directions D is constructed that contains the unit normals of the candidate parting planes. At present, the set of directions for a mold component comprise of the six principal directions (i.e. $\pm X, \pm Y, \pm Z$), face normals for all the planar faces of the primitive for the mold component (each mold component represents the mold cavity for a particular primitive) and a set of user defined directions. This set of directions can be easily augmented.

Once the set of directions for a mold component have been determined, the next step involves creating a matrix $Direction_Status$ similar to $Access_Status$ for storing the set of facets that block a facet f in a direction d in D . Procedure CREATEMATRIX described in Section 5.2.2 creates this matrix. Each entry in this matrix represents whether a facet is accessible in the given direction or not.

Since we have a set parting planes represented by their unit normals (set D), we need to develop a procedure for selecting the best parting plane for doing the decomposition. The procedure described in Section 5.2.3 sorts the candidate parting planes and decomposes the mold component along the best candidate plane. It first calls CHECKACCESSIBILITY to determine if the mold component is completely accessible or not. If the mold component is not completely accessible it calls CREATEMATRIX to create the matrix *Direction_Status*. It then selects the best parting plane to decompose the mold component.

5.2.1 Checking if the Mold Component is Completely Accessible

The procedure CHECKACCESSIBILITY checks if the mold component is completely accessible. It uses the matrix *Access_Status* to perform this test. For each column in this matrix, if there exists at least one entry that contains a null set, it implies that the entire mold component is completely accessible. However, if there are columns that do not contain a single null set, the facet represented by the column is completely inaccessible. The mold component needs to be decomposed to make it accessible. This procedure returns a status (*True/False*) that specifies whether the mold component is completely accessible or not.

procedure CHECKACCESSIBILITY

1. For every column in *Access_Status* do the following:
 - Search for the first entry in the matrix that is a null set.
 - If an entry is found, then continue.
 - Else break the loop and return *False*.
2. Return *True*.

5.2.2 Creating Matrix *Direction_Status*

CREATEMATRIX takes the set of non-convex-hull facets F_n of the mold component and the set of directions D as arguments. It creates the matrix *Direction_Status* for classifying the accessibility of a facet in a given direction. It is initially assumed that all the facets are accessible in the set of directions D . v implies that the direction d is not contained by hemisphere $H(f)$ and thus can never be used as an accessible direction.

procedure CREATEMATRIX(F_n, D)

1. Create a $M_d \times N$ matrix *Direction_Status*, where N is the number of the facets in F_n , M_d is the number of directions in D . Each entry of the matrix will indicate the accessibility status (a set of facets or a null set) of a facet f from a direction d in D . Initially, for a facet f , if a direction d is contained by hemisphere $H(f)$, the corresponding entry in the matrix is initialized to null set; otherwise, the corresponding entry is initialized to a set containing a value v .
2. For every pair of facets (f, f') in F_n , do the following:
3. Call INACCESSIBLE(f, f'), which returns I , the convex spherical polygon representing the inaccessibility region of f due to f' .
 - For every direction d in D , if the corresponding entry for facet f in *Directions_Status* does not contain v , if $d \cap I \neq \emptyset$, update the entry for facet f in *Direction_Status* for the row d by adding f' to set of facets that make f inaccessible in the direction d .

5.2.3 Accessibility Based Decomposition

A mold component needs to be decomposed if the procedure CHECKACCESSIBILITY returns *False*. In such a case, the user specifies the maximum allowable cuts that may be performed on the mold component. The procedure CREATEMATRIX is then called to create a matrix *Directions_Status*. The next step is to identify all the columns in matrix *Direction_Status* that do not contain a single null set. These columns represent the facets that are completely inaccessible.

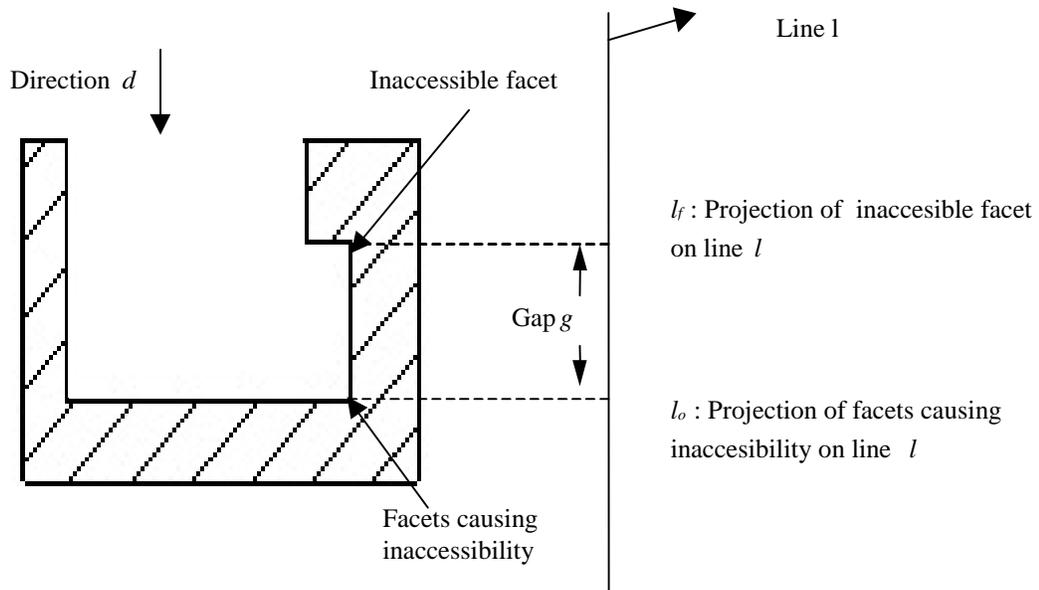


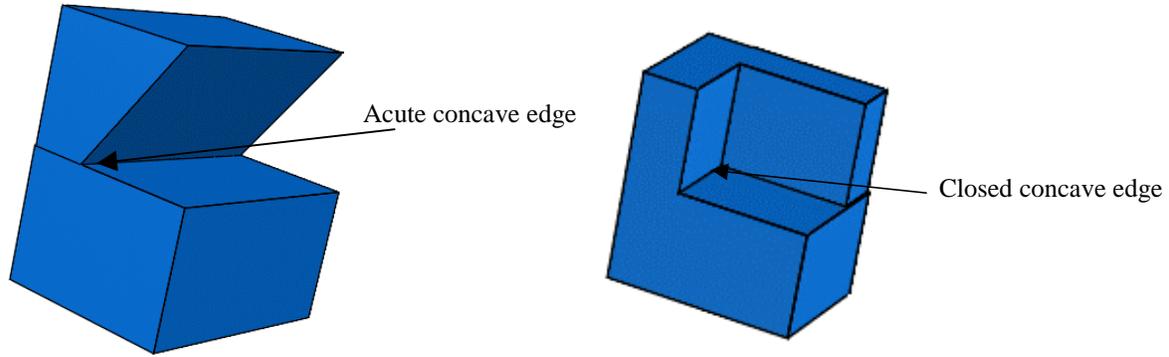
Figure 17: Projection of inaccessible facet and facets causing its inaccessibility on a line l representing a direction d from set D .

From the set D , those directions that represent the face normals of the primitive corresponding to the mold component are given highest priority. This is because this decomposition would be along the natural edges of the part. Therefore no visible parting line will be present in the part.

The following strategy is applied to find a suitable parting plane for partitioning the mold component:

1. A facet f is selected from the list of completely inaccessible facets that is being blocked by the least number of facets.
2. A direction d is selected from D that does not contain v in the entry for f in *Direction_Status*. If there are more than one such direction, a direction is selected that represents a face normal of the primitive corresponding to the mold component. If there are more than one such direction that represent the face normals, a direction is arbitrarily selected from among them. If no such direction exists, then any direction is arbitrarily selected.
3. If no direction exists that does not contain v , then any direction is arbitrarily selected.
4. The facet f and all the facets that are making f inaccessible are projected onto a line l whose direction is along d . This is shown in **Figure 17**. Let l_f be the line segment on l that represents the projection of f and l_o be the line segment on l that represents the projection of all the other facets. If l_f and l_o overlap, this direction is not a suitable direction. Select the next direction.
5. If l_f and l_o do not overlap then:
 - If the direction represents a face normal of the primitive corresponding to the mold component, project the plane representing the face of the primitive on the line. If the projection of the plane lies in the gap (including the end points), select the plane as the parting plane.
 - Else the mid point of the gap g between l_f and l_o is computed. A cutting plane represented by normal d and root point g is used to decompose the mold component.

The approach described above may not be the most optimum partition of the mold component. Since the user specifies the maximum allowable cuts for a mold component, it does not guarantee that the mold component can be decomposed such that it is completely accessible. Since we only look at facet-facet interaction for determining the cutting plane, we are not looking for a one-cut solution that will make the mold component accessible. The implementation of this algorithm also provides a provision for the user to input a partition plane for decomposing a mold component.



(a): A part having an acute concave edge

(b): A part having a closed concave edge

Figure 18: Parts that are not completely accessible by a finite sized cutting tool

5.3 Detecting Accessibility Problems due to Non-Zero Cutting Tool Diameter

Mold components are typically machined using cylindrical milling cutters. Therefore, even if the entire boundary of the mold component is accessible using the accessibility criteria described in the previous sections, the mold component may not be machinable using any finite diameter cutter. **Figure 18** shows an example of such cases.

We use the following approach to detect locally unmachinable regions:

1. We label edges as convex or concave by performing the following procedure for all edges:
 - Suppose face f_1 and face f_2 intersect at edge e . Let \vec{n}_1 and \vec{n}_2 be the outer normal vector of the two faces respectively, and \vec{e} be the direction vector for edge e in the edge loop of face f_1 . The edge e is concave if \vec{e} is opposite to $(\vec{n}_1 \times \vec{n}_2)$. Otherwise the edge is convex.
2. For every concave edge we measure the angle between the faces that form the edge.
 - If the angle is less than 90 degrees then the edge is labeled as an acute concave edge and considered unmachinable.
 - If the angle is 90° or more, we extend the edge on both ends by arcs whose radius is equal to the minimum tool diameter and the arc angle is $\frac{\pi}{2}$ radians. The tangent to the arc at the point where it touches the edge is collinear with the edge. We consider two arcs at each end. These two arcs are in the same plane as the two faces that form the edge. One end of the arc touches the end of the edge and the other end is placed on away from the face that is not coplanar with the arc. If the extended edge intersects with the part, then the edge is labeled as concave edge with a closed end.

5.4 Decomposition for Concave Edges

If the mold component contains any acute concave edge or a closed concave edge, we need to decompose the mold component. We know that an acute concave edge (i.e. an edge with an angle between its adjacent faces less than 90°) is always unmachinable due to the finite diameter of the cutting tool. In order to eliminate this manufacturability problem, the only solution is to cut the object along either of the two faces so that the acute concave edge is eliminated.

For a closed concave obtuse edge (i.e. an edge with angle between its adjacent faces greater than or equal to 90°, and at least one of whose ends is closed), there are also multiple solutions for making it machinable with one cut on the object. Similarly to the acute edge case, we can cut the object along either of the two adjacent faces to eliminate the concave edge. For an obtuse edge with only one closed end, there exists another solution – cutting along the third face, which makes that end closed. As shown in **Figure 19**, edge AB is a closed obtuse edge with a closed end at B . We have three one-cut decomposition solutions to make it machinable: (1) cutting along face f_1 , (2) cutting along face f_4 , and (3) cutting along face f_2 . The first two solutions eliminate the concave edge, and the third

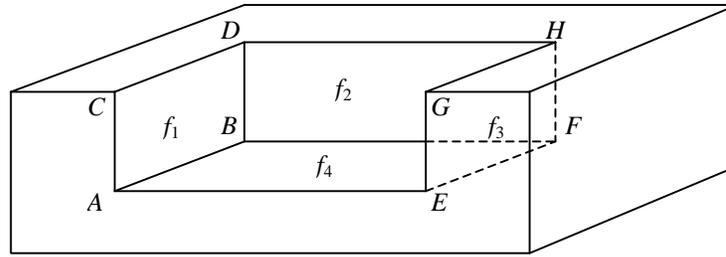


Figure 19: Decomposition for concave edges

solution simply makes the end at B open. For an obtuse edge with both ends closed, as edge BF shown in **Figure 19**, there are only two one-cut solutions, namely, cutting along face f_2 or cutting along face f_4 , both of which eliminate the concave edge.

Based on the above observations, we have developed a greedy algorithm to perform decomposition due to unmachinable concave edges.

1. For each of the unmachinable concave edges in the mold component, list all the candidate faces along which a one-cut solution can make it machinable.
2. Among the list, pick the face that appears the maximum number of times and cut the object along it.

The status of an edge may change after each cut. For example, an obtuse edge with both ends closed may have one of the ends opened after a cut. Thus, we need to re-check the status of the all the edges. This decomposition procedure runs recursively until all the edges become machinable.

Consider the example in **Figure 19**, we list all the candidate cutting planes for each of the unmachinable concave edges:

$$AB - f_1, f_2, f_4$$

$$BF - f_2, f_4$$

$$BD - f_1, f_2, f_4$$

$$FH - f_2, f_3, f_4$$

$$EF - f_3, f_4, f_2$$

Therefore, we have the following face list $[f_1, f_2, f_4, f_2, f_4, f_1, f_2, f_4, f_2, f_3, f_4, f_3, f_4, f_2]$. Both f_2 and f_4 appear five times, which is the maximum in this list. Therefore, cutting the object along either f_2 or f_4 will make the maximum number of unmachinable edges machinable. In this particular example, all the edges become machinable after a cut along either f_2 or f_4 . If there is still at least one concave edge remaining unmachinable after the cut, the above described procedure is applied recursively until all edges become machinable.

6 Reducing Mold Components by Eliminating Over Decomposition

The feature based decomposition might result in mold components that can be

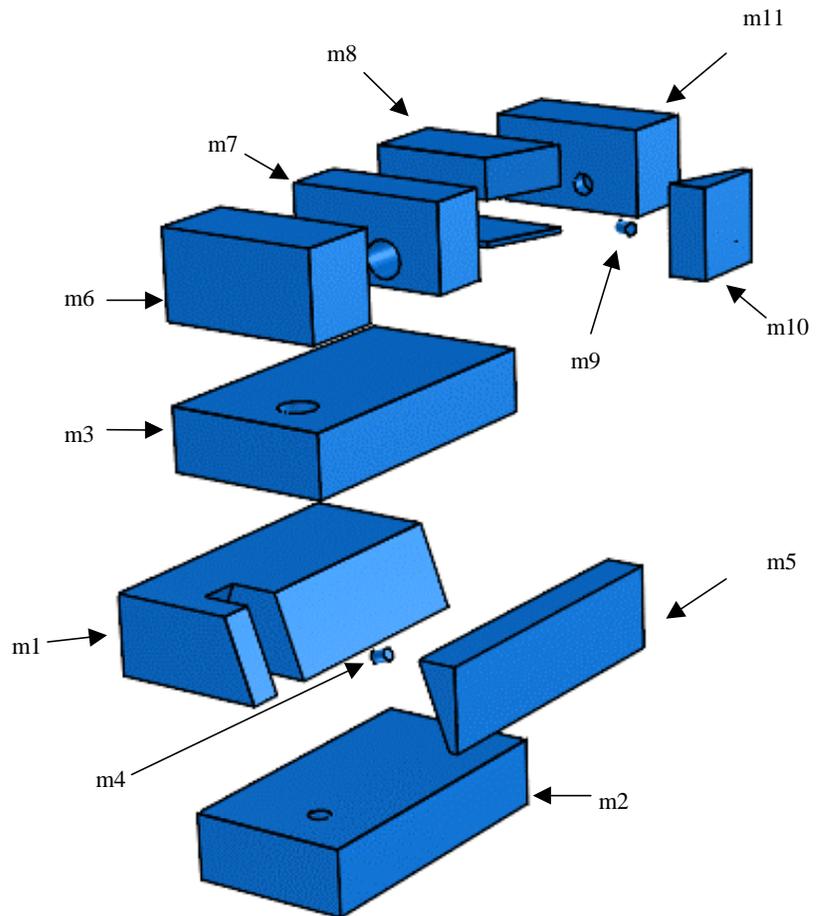


Figure 20: Mold components of part shown in Figure 4

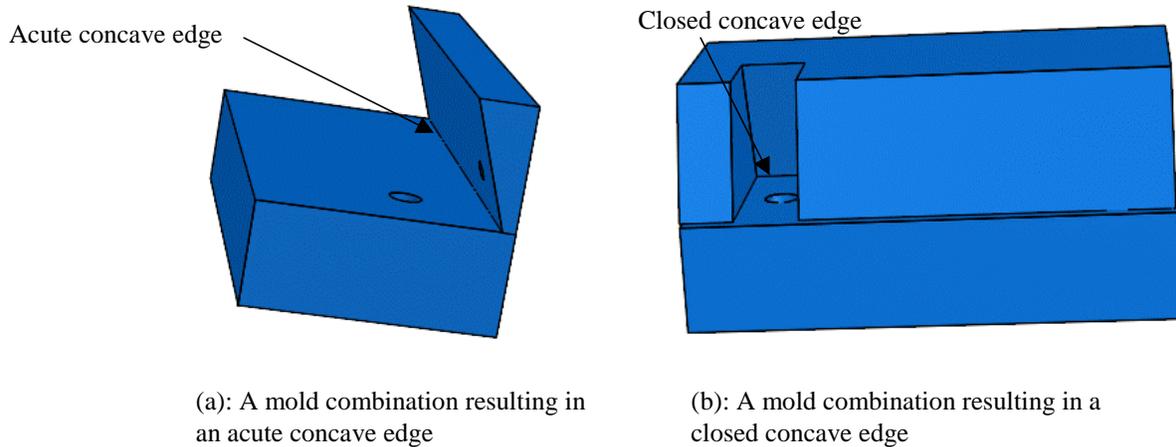


Figure 21: Invalid mold combinations due to acute or closed concave edges

combined without affecting the overall accessibility of the combination. **Figure 20** shows the mold components of the part shown in **Figure 4**. Since the feature based representation of the part consists of eleven primitives, the feature based decomposition of the mold results in eleven mold components. As there is no accessibility based decomposition in this example, the total number of mold components is eleven. However, some of the mold components can be easily combined without affecting the overall accessibility of the combined mold component. This section describes the conditions and provides an algorithm for doing recombination of mold components.

6.1 Conditions for Feasibility of Recombination

A pair of mold components that share a common planar face is a candidate pair for recombination if it satisfies the following conditions:

1. The recombination of the mold components should not result in acute concave edges or closed concave edges. In Section 5.3 we have described the procedure for identifying acute concave edges and closed concave edges. This condition invalidates the recombination of mold components shown in **Figure 21**.
2. The recombination of the mold components should not affect the global accessibility of the combined mold component. In Section 5.2.1 we have described the procedure for checking if a mold component is completely accessible. This procedure is used to check the accessibility of the combined mold component. If the combined mold component is not completely accessible, this combination is not valid. **Figure 22** shows a pair of mold components that cannot be combined because they do not satisfy this condition.

6.2 Overview of Recombination Algorithm

We follow a feasibility based approach to recombine the mold components. It tries to eliminate over decomposition as it identifies it till no further recombination is possible. However, this approach does not necessarily minimize the total number of mold components. There may be other ways of combining the mold components that result in an optimal number of mold components. However, the approach may be easily extended to produce the optimal result by backtracking during the search process. The different steps involved in the mold

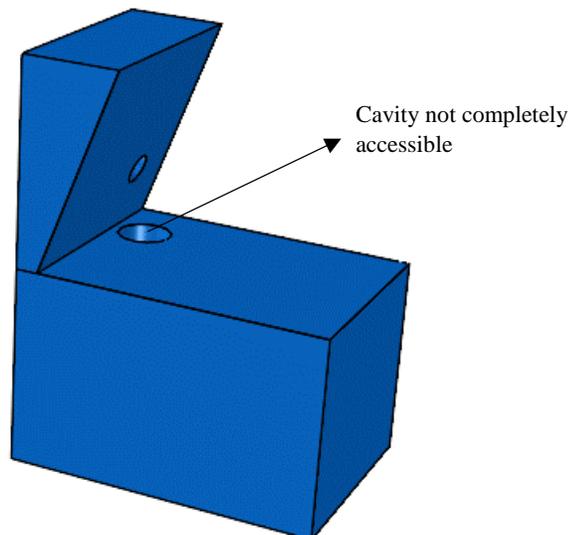


Figure 22: A mold combination resulting in inaccessible regions

recombination are as follows:

1. Determine which mold components share a common planar face. A contact graph is created that represents the connectivity of the mold components. The contact for the mold components in Figure 20 is shown in **Figure 23**.
2. Construct a set C initialized to a null set whose each element c is an unordered pair of mold components (c_{m1}, c_{m2}) that can be combined. For every pair of mold components that are connected in the contact graph, do the following:
 - Perform the validation checks described in Section 6.1 to determine if the pair of mold components can be combined. If the pair can be combined add it to C .

For the contact graph shown in Figure 23, C is given by

$$C = \{\{m2, m5\}, \{m4, m5\}, \{m9, m10\}, \{m7, m10\}\}$$

3. While C is not empty, do the following:
 - Select an element c from C and unite the pair of mold components (c_{m1}, c_{m2}) contained in c to produce a new mold component m_{new} . Remove c from C .
 - For every element d in C that contains either c_{m1} or c_{m2} do the following:
 - Replace c_{m1} or c_{m2} by m_{new} in d and perform the validation checks described in Section 6.1 to determine if the pair of mold components contained in d can be combined. If the pair of mold components cannot be combined, remove d from C .

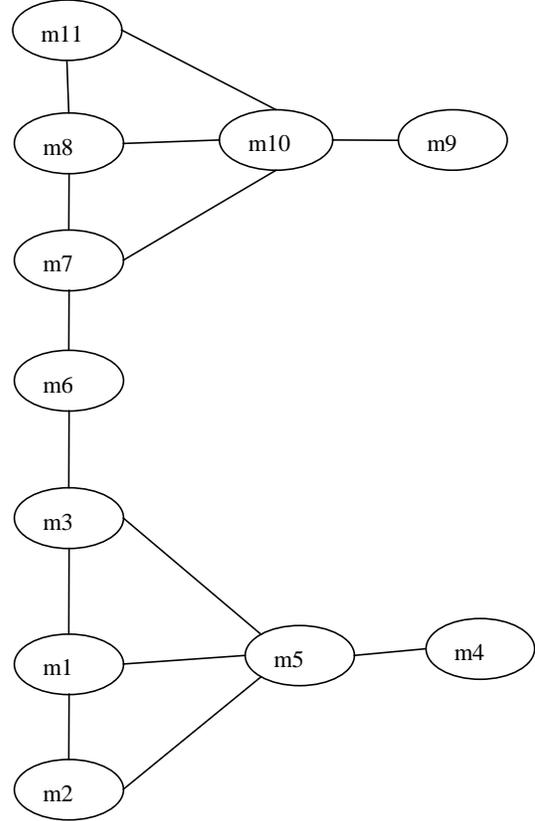


Figure 23: Contact graph of the mold components

6.3 Checking Recombination Feasibility for a Mold Component Pair

The algorithm described in the previous section can be used to recombine all possible pairs of mold components. However, the procedure described in Section 6.2 that is required for validating a combination of mold components is computationally expensive since the generation of the accessibility matrix requires performing pair-wise intersections of all the facets in the facetised representation of the mold component. It also requires performing an intersection of an inaccessible region for a facet due to another facet with all the spherical triangles on the unit sphere. Therefore, a set of heuristics have been developed that can be used to combine two mold components without generating a new accessibility matrix for every candidate combination of mold components.

We have developed two heuristics for identifying the pair of mold components that can be combined without creating a new accessibility matrix for the combined mold component. These heuristics are described in this section.

1. *Heuristic 1:* For any pair of mold components (m_1, m_2) that are connected in the contact graph, let \vec{n}_1 and \vec{n}_2 ($\vec{n}_2 = -\vec{n}_1$) be the two opposite normal vectors of the common planar face f along which we want to combine the mold components. A 2D view of the representative example is shown in **Figure 24**. The unit sphere that represents all the directions in space is divided by f into two hemispheres, h_1 and h_2 . A sufficient condition for validating the combination of these mold components is given below:
 - If an accessibility direction \vec{v}_1 exists for all the facets in the facetised representation of m_1 such that $\vec{v}_1 \cdot \vec{n}_1 \geq 0$ and an accessibility direction \vec{v}_2 exists for all the facets in the facetised representation of m_2

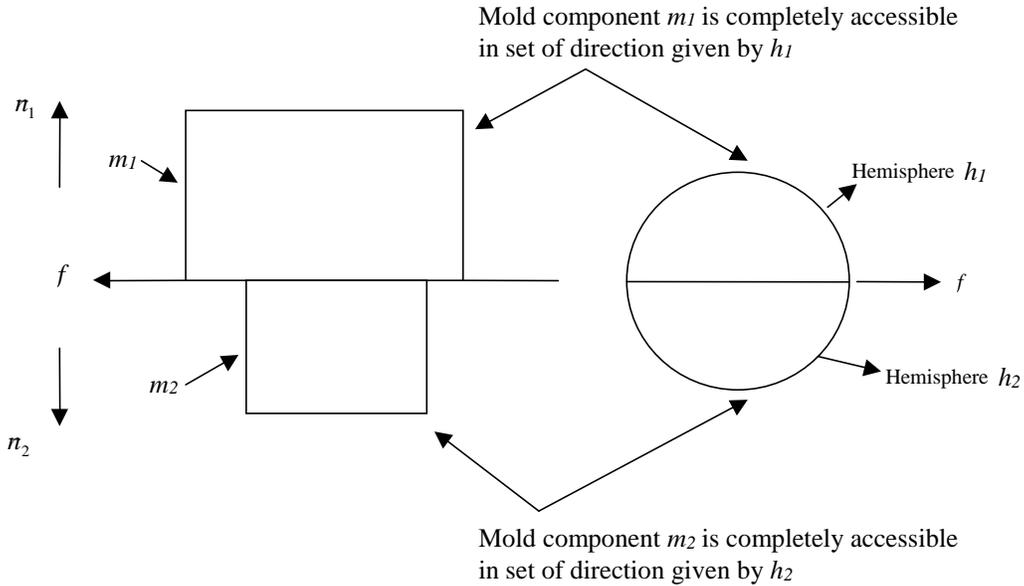


Figure 24: Mold components completely accessible in set of directions lying in different hemispheres

such that $\vec{v}_2 \cdot \vec{n}_2 \geq 0$, then the mold components can be combined. In such a case, every facet in m_1 will be accessible in at least one direction that lies on hemisphere h_1 while every facet in m_2 will be accessible in at least one direction that lies on hemisphere h_2 . Therefore, m_1 and m_2 can be combined without creating any accessibility problems.

2. *Heuristic 2*: Let m and m' be two mold components that correspond to 2.5D primitives such that $m \cup p$ is convex (where p is the primitive that corresponds to m) and $m' \cup p'$ is convex (where p' is the primitive that corresponds to m'). m and m' can be combined if the following condition is satisfied.

- The infinite sweep of the profile of p does not intersect with m' and the infinite sweep of the profile of p' does not intersect with m .

This condition ensures that the concave regions of m and m' do not have any accessibility problems due to combination. The convex regions of m and m' do not have any accessibility problems due to the argument presented in *Heuristic 1*. This is shown in **Figure 25**.

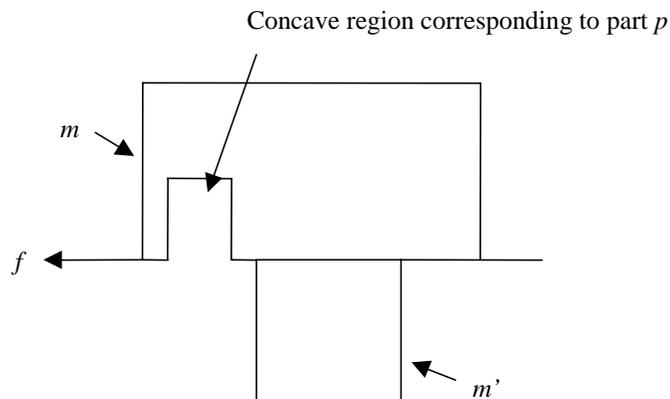


Figure 25: Mold components m and m' combining along face f

7 Adding Assembly Features into Mold Components

The addition of assembly features to the mold assembly consists of two steps: (1) selecting the appropriate pairs of mold components that need assembly features, and (2) adding assembly features at appropriate positions on these mold components. These steps are described in Section 7.1 and Section 7.2 respectively.

7.1 Selecting the Component Pairs to Add Assembly Features

The primary reason for adding assembly features to mold components is to constrain the degrees of freedom of each mold component such that all the mold components in the assembly have either no relative motion or translational motion only in the positive Z direction. To achieve this, an assembly feature in the form of a hole-pin combination is added to a pair of mold components along their contact face. Addition of assembly features to mold components increases the mold component manufacturing time. We have considered the set-up time for machining as the main factor that contributes to the total manufacturing time. An algorithm following a greedy heuristic has been developed that reduces the total setup time by orienting features such that multiple assembly features can be produced in the same setup.

Various contacts in an assembly of mold components can be represented by a graph $G(V, E)$. Each mold component is represented by a vertex v in set V . If any two mold components, say v_1 and v_2 , share a common face, an edge $e(v_1, v_2)$ is added to the set E . Since multiple assembly features on the same planar face of a mold component can be manufactured in a single setup, our algorithm tries to put as many assembly features as possible on a single contact face. We use a variation of the spanning tree algorithm that results in a solution in which mold components (represented by vertices in the graph), get kinematically constrained with respect to each other in a pattern of a spanning tree of the graph. Each edge in the spanning tree kinematically constraints two mold components by adding a pair of hole-pin combination on the contact face that corresponds to the edge.

We construct the spanning tree $T(S, X)$ using the procedure ASSEMBLY-TREE described below. After constructing the spanning tree, we add a pair of hole-pin combination on the contact faces corresponding to each edge in the tree. Technique for determining the locations of hole-pin features on the contact faces is described in Section 7.2.

procedure ASSEMBLY-TREE ($G(V, E), T(S, X)$)

1. Initialize $S = \Phi$ and $X = \Phi$.
2. Color all $e \in E$ such that all edges lying in the same contact plane have the same color.
3. Find a vertex $r \in V$ that has the highest number of edges of the same color. Add r into set S and make this the root node of tree T .
4. Let V' be the set of nodes in V that are common with the set S . Find v in V' that has the highest number of the crossing edges of the same color for partition $(V', V - V')$. We say that an edge $e \in E$ is a crossing edge for the partition $(V', V - V')$ if one of its vertices is in V' and the other is in $V - V'$. Let E_v be the set of highest cardinality of crossing edges of the same color for node v .
5. Add all edges in E_v into X , and all vertices in $(V - V')$ into S that are connected by E_v to v .
6. If $S \neq V$, go to 4. Otherwise, stop.

7.2 Adding Assembly Features

In order to add assembly features (i.e., two pin-hole pairs) to a pair of mold components that need to be assembled, we initially compute the shape of their contact region to determine which mold component has the pins and which mold component has the holes. This is performed comparing the areas of faces that contain the contact region. The component that has larger face area is assigned the hole assembly feature and the other component is assigned the pin assembly feature. Then we use procedure DETERMINE_PINS (described later in this section) to determine the positions of the pins. Finally we create the pin-hole pairs at the positions found in the previous step and attach them to the corresponding mold components.

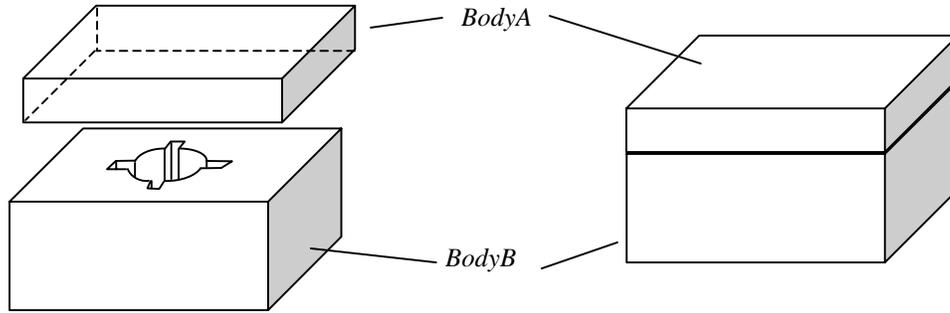


Figure 26: Example of *BodyA* and *BodyB* Assembly

Figure 26 shows an example of a pair of mold components that need to be assembled. *BodyA* and *BodyB* share a common planar face. The initialization procedure computes their contact region through a non-regularized intersection. The component that has the holes is returned as *HoleBody* while the component that has the pins is returned as *PinBody*. The procedure also transforms the contact region onto the X-Y ($Z=0$) plane and returns it as *BodyI* (a body that has only one planar face). *BodyI* will be analyzed further in procedure DETERMINE_PINS as a 2-D entity to determine the positions of the pin-hole pairs. **Figure 27** shows a contact region transformed onto the X-Y plane.

We use procedure DETERMINE_PINS to find the positions of the two pins. DETERMINE_PINS takes *BodyI* as an argument, which is the body resulting from transforming the contact region onto the X-Y plane. DETERMINE_PINS finds the edges and loops of the contact region. It then calls MESHING to generate a set of grid points on the plane where each point represents a candidate position to put the pin. The sub-procedure LOCATE_PINS then traverses all the candidate positions, checking their validity and finding two valid positions for pins of a desired radius so that the two pins are placed at the maximum possible distance between them. DETERMINE_PINS returns these positions as *PinPos1* and *PinPos2*.

procedure DETERMINE_PINS (*BodyI*)

1. Retrieve the information of all the loops and edges of the assembly region, and store it in a data structure *Profl*.
2. Call MESHING (*Profl*), which returns a set of grid points *Sp*.
3. Call LOCATE_PINS (*Sp*, *Profl*), which returns *PinPos1* and *PinPos2*.

MESHING first computes a rectangle bounding the contact region and then generates an orthogonal grid on the rectangular area at a reasonable resolution. All the crossing points on the grids represent the candidate positions to place a pin. This procedure takes *Profl* as an argument. **Figure 28** shows the grid on the example contact region.

procedure MESHING (*Profl*)

1. Calculate the bounding rectangle of *Profl*, the profile of the contact region.
2. Mesh the rectangle with an orthogonal grid. The space width of the grid should be small as compared to the diameter of the pin. The set of all the candidate positions is returned as *Sp*.

LOCATE_PINS takes *Sp*, the set of candidate positions, and *Profl*, the profile of the contact region, as two arguments. The procedure filters out all the invalid positions that lie outside *Profl*. It then calls procedure PIN_SIZE for each of the remaining positions to determine the maximum possible radius of the pin at that position. If the maximum

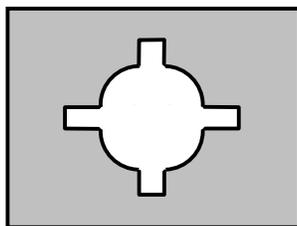


Figure 27: Assembly Region Transformed onto X-Y Plane

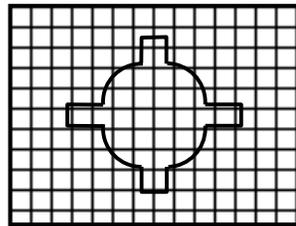


Figure 28: Meshing the Assembly Region

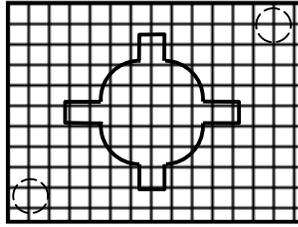


Figure 29: Locating the two pins

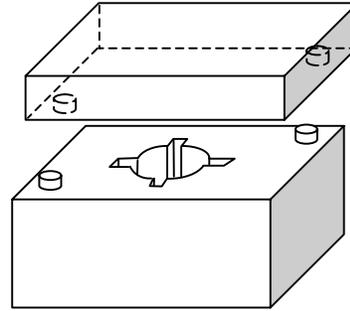


Figure 30: Creating Assembly Features

possible radius is smaller than the desired radius of the pin, then the position is also invalid. The pair of position that is farthest apart is returned as *PinPos1* and *PinPos2*.

procedure LOCATE_PINS (*Sp*, *Profl*)

1. For every candidate position p in set Sp , do the following:
 - If p is outside profile *Profl*, then delete this position.
 - Otherwise, call PIN_SIZE (P , *Profl*). It returns the maximum possible radius of a pin at position P . If the returned radius is smaller than the desired radius of the pin times a specified ratio (between 0 and 1), delete this position.
2. For all valid pin positions, calculate the pair-wise distance. Return the pair with the greatest distance as *PinPos1* and *PinPos2*.

PIN_SIZE takes p , a valid candidate position, and *Profl*, the profile of the assembly region, as two arguments. The procedure calculates the distance from p to each edge of the profile. The *distance* from position p to an edge e is defined as the length of the shortest line segment that can be drawn from p to a point on e . Since the radius of the pin at position p should be no greater than the smallest of the distances obtained, this smallest distance is returned as the maximum radius of the pin.

procedure PIN_SIZE (p , *Profl*)

1. For every edge e in profile *Profl*, calculate the distance from p to e .
2. Return *Pinr*, the maximum radius of the pin at p , which is the smallest distance from p to the various edges.

Figure 29 shows an example of the pin locations. **Figure 30** shows the pair of mold components after adding assembly features.

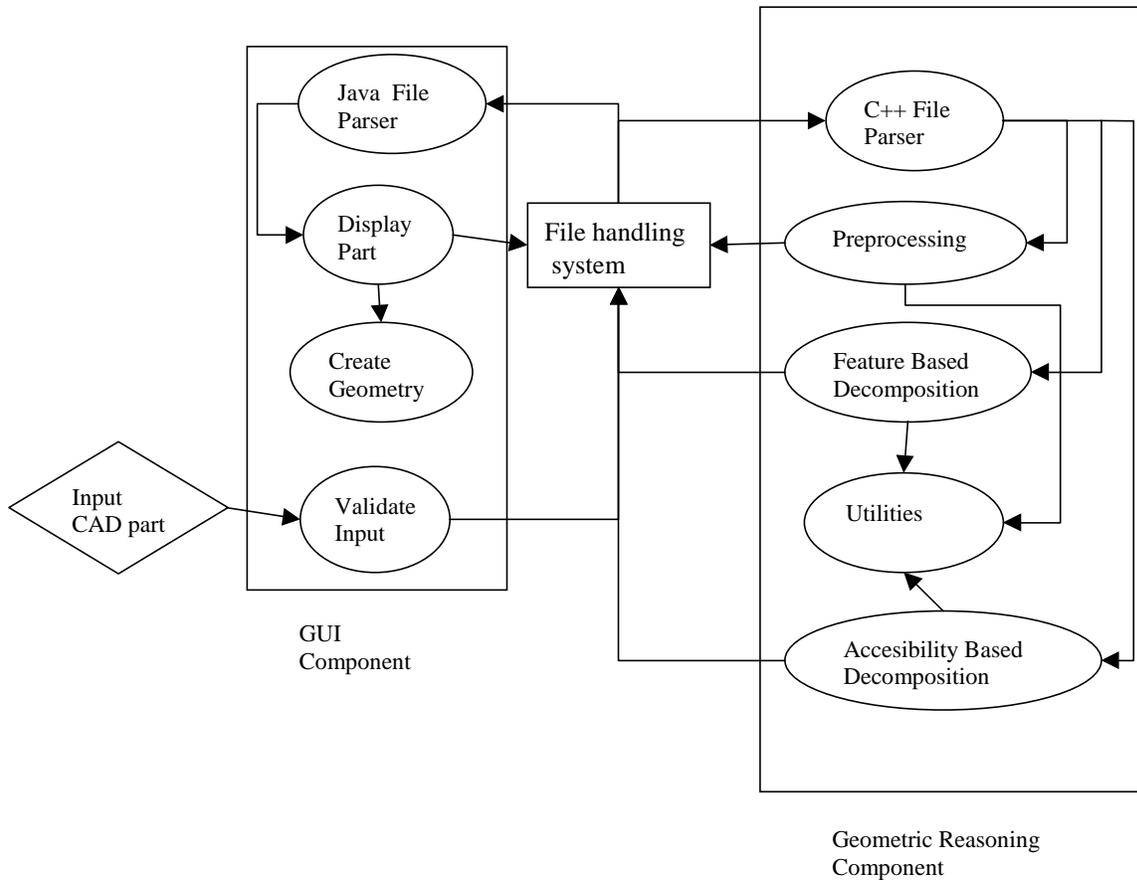


Figure 31: System architecture

8 System Implementation and Examples

8.1 System Architecture

The system architecture of our implementation is shown in **Figure 31**. The system consists of two main components each of which have multiple modules for performing various tasks. These two components are:

1. *Graphical User Interface (GUI):* The graphical user interface is implemented in Java. Java 3D, a package in Java 2, is used to render solid models of parts.
2. *Geometric Reasoning Component:* The geometric reasoning is done in C++/ACIS. ACIS is a 3D geometric modeler provided by Spatial Technologies.

The communication between the components is through a file based system. A set of keywords have been defined that enable the communication between these components. Each component contains a module that monitors a status file, which is used for reading and writing the messages between the components. A log file is created that maintains a record of all the communication between the GUI and the geometric reasoning component.

8.2 Example Run of the System

In this section we present an example run of the system. The different steps in the implementation are as follows:

1. Using the GUI, the user selects a file that contains the feature based representation of the part.



Figure 32: Solid model of part

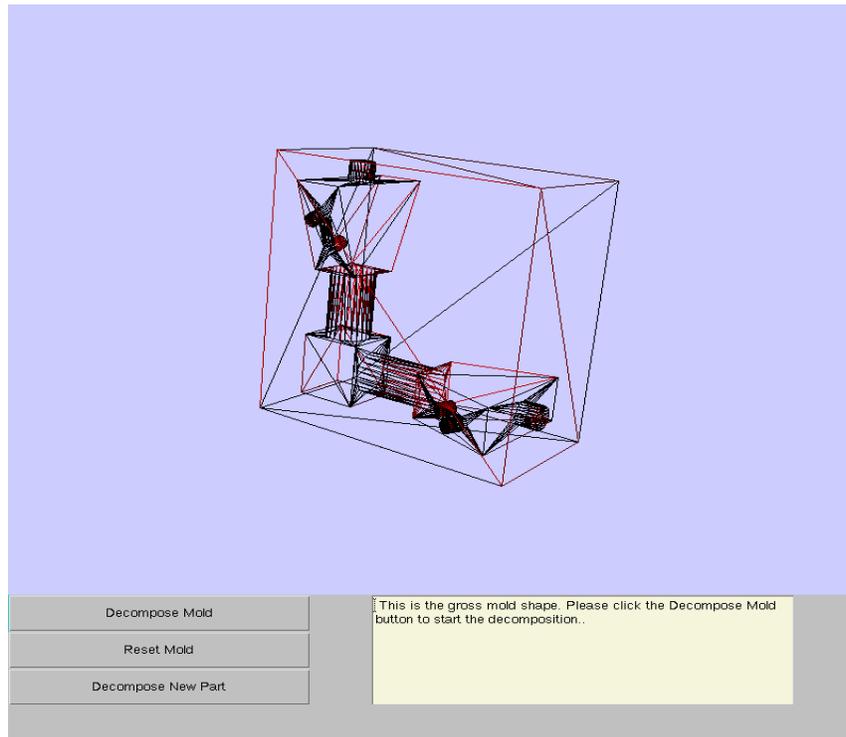
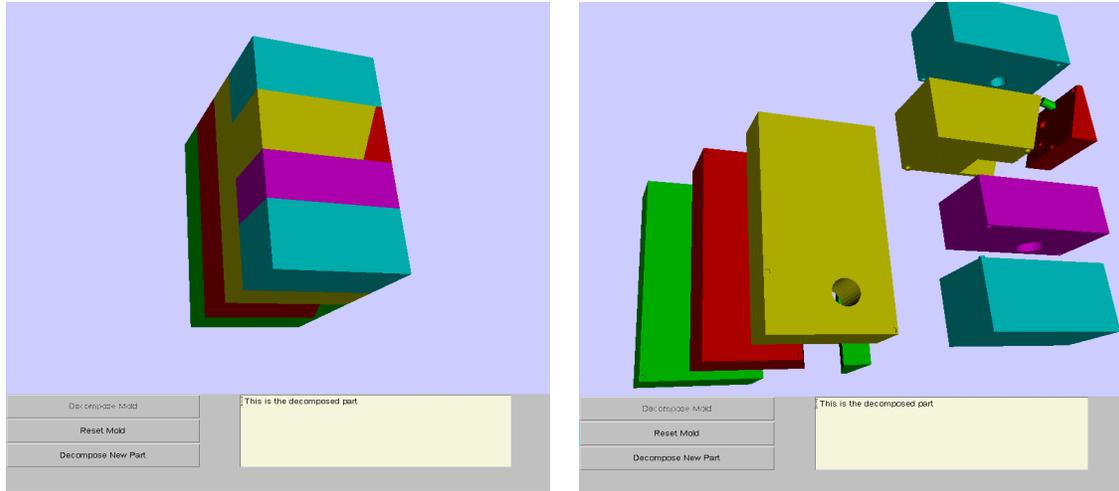


Figure 33: Wireframe model of gross mold

2. This file is then used by the Geometric Reasoning component. If the feature based representation of the part is valid, the part will be created and shown to the user. This is shown in **Figure 32**. The user will be prompted to scale the part if she/he desires.



(a): Assembled mold

(b): Disassembled mold

Figure 34: Decomposed mold

3. After the user has input the scaling factor for the part, a wireframe model of the mold is displayed. This is shown in **Figure 33**.
4. By clicking on the *Decompose Mold* button, the user initiates the decomposition of the mold. A decomposed mold for the part is shown in **Figure 34**.

8.3 Example Parts

We have used our implementation to design and manufacture parts requiring multi-piece molds. In this section we present a few example parts whose molds were designed using our system. The mold components for these molds were manufactured on a 3-Axis CNC milling machine. After the mold components were made, the mold was assembled manually using the assembly features incorporated on each mold component. The parts were made of polyurethane. Section 8.4 describes the polymer part fabrication process.

1. *Example Part 1- Indexer*: **Figure 35** shows the solid model of an indexer. The multi-piece mold for the indexer was made using our system. The mold consisted of four mold components. The assembly features were added to the mold components after the decomposition was done by the system. This was done to ensure an easy assembly of the mold components. **Figure 36** shows the mold components of the mold. **Figure 37** shows the polyurethane part that was made in this mold.
2. *Example Part 2-Multi-Stage Rotor*: **Figure 38** shows the solid model of a multi-stage rotor. The mold for the rotor consisted of seventeen mold components. **Figure 39** shows the mold components for this mold. Assembly features were incorporated to the mold components to ensure an easy assembly. **Figure 40** shows the polyurethane part that was made in this mold.

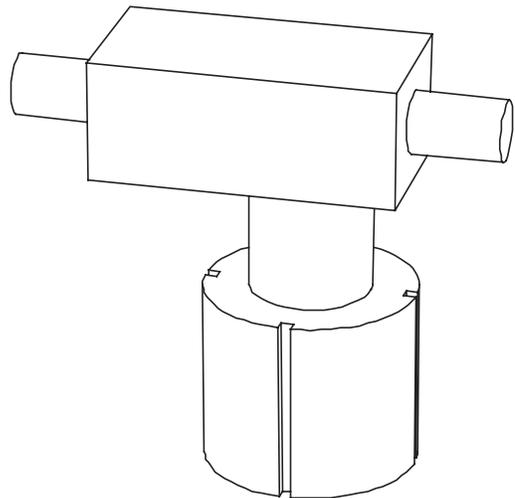


Figure 35: Solid model of indexer

8.4 Polymer Part Fabrication

Polyurethanes can be used to fabricate complex parts that require multi-piece molds. Polyurethanes (PURs) are a group of polymers with highly versatile properties and a broad range of commercial applications. PURs can be manufactured in an extremely wide range of grades, in polymer stiffness from very flexible elastomers to rigid hard plastics. The manufacturer generally sells the PUR chemical components, usually in the form of liquids, and may also provide information to enable the user to make the PUR. The user mixes together the PUR

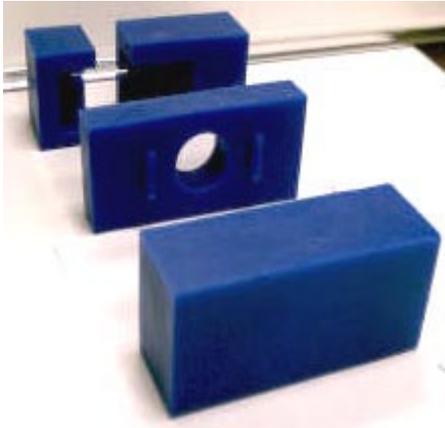


Figure 36: Mold for the part



Figure 37: Part

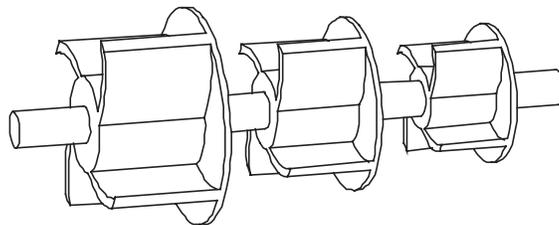


Figure 38: Solid model of multi-stage rotor

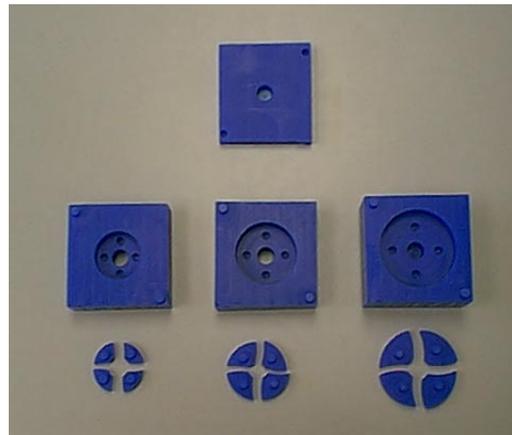
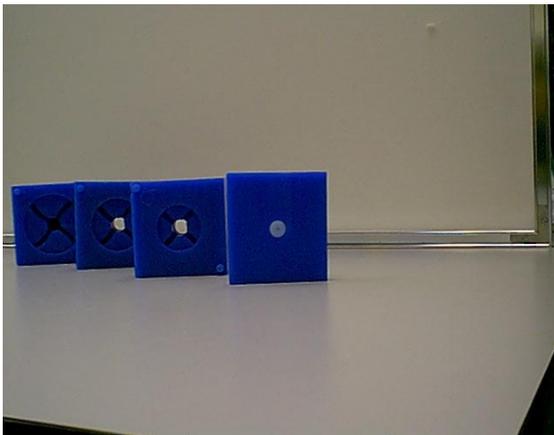


Figure 39: Mold for the part

chemicals that then react to make the required polymer product. Thus they can be tailored with remarkable accuracy to meet the needs of a particular application. Most of the PURs cure at room temp so no special curing facility is needed for these parts.

Complex polymer parts are easy to manufacture by casting PUR into multi-piece wax molds. The multi-piece molds are first fabricated from machinable wax on a CNC machine. After the mold components are machined at the machining facility, they are cleaned using acetone to remove dust particles that may get trapped in the mold cavities. Cleaning of dust particles is required to get a good surface finish. The mold components are assembled together using the designed assembly features. Then PUR components are mixed in the specified ratio by weight or volume and poured into the mold. The cast part is then left to set for the required demolding time. The demolding time typically varies from 2 hours to 10 hours depending on the grade of PUR used. The cast part is demolded by removing the wax mold components and the fabricated part is then cured for the ultimate cure time. The ultimate cure time varies from 2 days to 10 days.



Figure 40: Part

PURs are very easy to cast as most of the widely used PURs are in the liquid state at room temperature. However, care must be taken to ensure the fabrication of good quality polymer parts. The PUR components should be mixed and poured very quickly otherwise PUR may get cured partially before it could be poured into the molds. In such a case, it will not flow into the molds properly and that may result in under-filling in some portions of the mold. When mold components form a horizontal partitioning, these pieces need to be clamped together properly to avoid leakage of material under gravity through the gap between mold components.

9 Conclusions

This paper presents a systematic approach for doing automated design of multi-piece sacrificial molds. These molds can be used to make complex shaped parts that cannot be made using two-piece molds. Manufacturing techniques such as gelcasting and polyurethane manufacturing can be used to make parts using these molds. A novel approach has been developed for doing decomposition of a mold. The mold decomposition process has been divided into four steps:

1. The first step involves a feature-based decomposition of the part. The input to the algorithm is a feature based representation of the part. This decomposition results in different mold components for each primitive. The primary reason for doing feature based decomposition is to simplify the problem of mold decomposition since it is easy to do geometric reasoning with a mold components that represent individual primitives.
2. The next step is an accessibility-based decomposition of those mold components that are not completely accessible. Accessibility analysis is performed on all mold components and those mold components that are not completely accessible are further decomposed.
3. The feature-based decomposition might result in over decomposition of the mold. This step performs a pair-wise analysis of all mold components that share a common planar face and combines those mold components that can be combined without compromising their accessibility.
4. Finally assembly features are added into various mold components to facilitate assembly of the mold.

A system has been developed based on the above approach for designing multi-piece sacrificial molds. This system has been used to design multi-piece molds for several parts. The mold components of these molds have been machined on a 3-Axis CNC milling machine and the parts has been fabricated using polyurethanes. The successful fabrication of these parts shows that such a system can be used to design multi-piece sacrificial molds for complex shaped parts that cannot be produced using two-piece molds.

There are a number of potential benefits of automating the design of multi-piece molds. The principal benefits are enumerated below.

1. Mold design is a laborious process that requires significant time from the mold designer. This is aggravated in the case of multi-piece molds. Automated mold design significantly reduces the mold design time.

2. This approach allows us to manufacture parts that could not be produced earlier using two-piece molds. Thus it expands the design space for parts that can be produced using casting processes such as gelcasting and polyurethane manufacturing.
3. Since this approach automatically produces solid models of mold components, it can be integrated with CAM systems to generate the cutter path plans for manufacturing the individual mold components. Thus an integrated system can be developed that can simultaneously design and generate the cutter path plans for manufacturing the individual mold components in a mold assembly.

10 References

- [Chen93a] L.-L. Chen, S.-Y. Chou and T. C. Woo, "Separating and intersecting spherical polygons: computing machinability on three-, four-, and five-axis numerically controlled machines", *ACM Transactions on Graphics*, 1993, Vol. 12, No. 4, 305-326
- [Chen93b] L. L. Chen, S.-Y. Chou and T. C. Woo, "Parting directions for mould and die design", *Computer-Aided Design*, Vol. 25, pp. 763-767, 1993.
- [Dhal00] S. Dhaliwal, S. K. Gupta and J. Huang, "Computing exact global accessibility cones for polyhedral objects", Accepted for publication for the 2000 ASME DETC/DFM Conference, September 10-13, Baltimore, USA.
- [Hui92] K. C. Hui and S.T. Tan, "Mould Design with Sweep Operations – a Heuristic Search Approach". *Computer Aided Design*, Vol. 24(2), 1992.
- [Kris97] S. Krishnan, "Design for Manufacture: An Integrated System for Injection Molding and Milling", *PhD Paper, Mechanical Engineering, University of Maryland, College Park*, 1997.
- [Lent49] J. G. Lenthem, "Spherical trigonometry, for the use of colleges and schools", *Macmillan Pub.*, London, 1949
- [Megi83] N. Megiddo, "Linear-Time Algorithm for Linear Programming in R^3 and Related Problems", *Society for Industrial and Applied Mathematics*, Vol. 12, No. 4, November 1983.
- [Ravi90] B. Ravi and M. N. Srinivasan, Decision Criteria for Computer-Aided Parting Surface Design. *Computer Aided Design*, Vol. 22(1), 1990.
- [Wein96] M. Weinstein and S. Manoochehri, "Geometric Influence of a Molded Part on the Draw Direction Range and Parting Line Locations", *Journal of Mechanical Design* Vol. 118, March 1996.
- [Wein97] M. Weinstein and S. Manoochehri, "Optimum Parting Line Design of Molded and Cast Parts for Manufacturability", *Journal of Manufacturing Systems*, Vol. 16(1), 1997.