This document contains the draft version of the following paper:

A.K. Priyadarshi and S.K. Gupta. Algorithms for generating multi-stage molding plans for articulated assemblies. *Robotics and Computer Integrated Manufacturing*, 32(3/4):350-365, 2009.

Readers are encouraged to get the official version from the journal's web site or by contacting Dr. S.K. Gupta (skgupta@umd.edu).

ALGORITHMS FOR GENERATING MULTI-STAGE MOLDING PLANS FOR ARTICULATED ASSEMBLIES

Alok K. Priyadarshi * Satyandra K. Gupta †

August 31, 2007

Abstract

Multi-stage molding is capable of producing better-quality articulated products at a lower cost. During the multi-stage molding process, assembly operations are performed along with the molding operations. Hence, it gives rise to a new type of planning problem. It is difficult to perform the planning manually because it involves evaluating large number of combinations and solving complex geometric reasoning problems. This paper investigates the problem of generating multi-stage molding plans for articulated assemblies. We present a planning framework that allows us to utlize constraints from experimentally proven molding plans. As a part of the planning problem, we determine the molding sequence and intermediate assembly configurations. We present algorithms for all the steps in the planning problem and characterize their computational complexities. Finally we illustrate our approach with representative examples.

1 Introduction

Plastic products are usually produced by first molding individual components separately, and then assembling them together. A recent alternative, referred to as in-mold assembly process, performs molding and assembly steps concurrently inside the mold itself. This means that an entire assembly consisting of multiple components can be produced by a single set of molds, thereby eliminating the need for secondary assembly operations and the use of bolts, welds, glue, or other fasteners.

In-mold assembly has several advantages over traditional techniques that involve molding the components separately and then assembling them. It allows integration of functional elements, thereby reducing the number of components and additional assembly steps. Several studies have indicated that assembly costs make up 40% to 50% of the manufacturing costs to produce a product. Reduction in number of components reduces the associated assembly labor, purchasing, inspecting, warehousing, capital requirements and piece part costs of a product. In-mold assembled products also have better component-alignment and overall structural integrity than their traditional counterparts.

In-mold assembly opens up the design space and presents new possibilities. One of its applications has been in producing multi-material rigid and compliant structures where material interfaces are adhered to each other completely constraining the motion between them [Gouk06]. One of the most recent successful applications of in-mold assembly has been in producing multi-material rigid-body articulated devices. Unlike compliant mechanisms, rigid-body articulated devices have non-binding interfaces with selective degrees of freedom between components. Multi-material articulated devices are widely used in toys, medical instruments, consumer products, and household appliances. Figure 1b shows the swash plate made traditionally. The number of components in the traditionally manufactured swash-plate is eleven, while the in-mold version has only five.

The most common and economically feasible way of performing in-mold assembly is through multi-stage molding, a form of specialized injection molding technique [Pirk98, Plan02, Good02, Li04]. Various polymers composing the different material sections are heated to their melting temperatures, then injected in sequence

*Department of Mechanical Engineering and Institute for Systems Research, University of Maryland
†Department of Mechanical Engineering and Institute for Systems Research, University of Maryland

into a mold or set of molds. The mold cavity shape changes after each molding stage to accommodate the material to be injected in the next stage. The liquefied polymers solidify into their desired shapes by taking on the form of the mold cavities in which they reside.

Multi-stage molding process gives rise to a new type of planning problem. Solving this planning problem requires finding a sequence in which components can be molded. For each step in the sequence we also need to find the assembly configuration in which the new component will be molded onto the previously completed assembly. This planning problem is different from classical assembly planning problems due to following reasons. First, sequence constraints are fundamentally different and they mainly arise due to molding considerations (e.g., formation of undercuts, melting points, shrinkage). Second, multiple components can be added to the assembly at the same time. Third, joints can be manipulated to change assembly configuration at each step. Finally, the cost function is different from the assembly cost function. It needs to account for not only the operation cost but also the tooling and defect costs. The mold design software systems (MoldWizard, ProMold, IMold, etc.) available in market today do not generate a molding plan, but provide tools for analyzing moldability and creating mold pieces. A molding plan needs to be developed manually which is difficult. Developing a plan which is both feasible and optimal involves examining many sequences and solving complex geometric reasoning problems. The desired articulation and multiple molding stages introduce geometric constraints, which if violated, results in poor part quality, longer molding cycles, and high tooling cost.

This paper investigates the problem of generating multi-stage molding plans for articulated assemblies. We present a planning framework that allows us to utilize constraints from experimentally proven molding plans. As a part of the planning problem, the algorithm presented in this paper determines the molding sequence and intermediate assembly configurations. We present algorithms for all the steps in the planning problem and characterize their computational complexities. We illustrate our approach with three representative examples.

2 Related Work

During the multi-stage molding process, assembly operations are performed along with the molding operations. Our work is hence related to assembly sequence planning and multi-stage mold design.

2.1 Assembly Sequence Planning

Automatic assembly sequence planning has been an important research topic in geometric modeling, robotics and artificial intelligence. The process of assembling and disassembling are two different but similar processes in a real world. The assembly process starts from a configuration where all the components are completely disassembled and the last assembly operation leads to the final product. An assembly sequence plan specifies the order in which parts and subassemblies are to be inserted into each subassembly. Research in this domain is intended to generate a good and feasible assembly sequence plan automatically.

The early assembly sequence planners were mainly interactive in nature. Geometric constraints were supplied by a human, interactively, by answering a series of questions asked by the computer. One such system was by Fazio and Whitney [Fazi87] which asked questions like: a) Is it true that a component C_i cannot be inserted after components C_j and C_k are assembled or b) Is it true that C_i cannot be inserted if components C_j and C_k are yet to be assembled. The user of these systems had to answer these questions and system used some logical reasoning to produce a feasible assembly plan. Automated geometric reasoning were later used to answer these questions automatically. These approaches generated several candidate assembly sequences and tested their feasibility by applying geometric reasoning. But these approaches tend to generate a large number of candidate assembly sequences and were repeating the same geometric reasoning many times. Then attempts were made to store and reuse previous computations and in some cases new assembly representations were used that implicitly reduced the number of geometric computations.

The work by Wilson and Latombe [Wils94] used Non-Directional Blocking Graph (NDBG) representation which implicitly contained the geometric constraints. NDBG was automatically generated from the input geometry of the product by applying geometric reasoning. Woo and Dutta [Woo91] proposed construction of an disassembly tree and generated disassembly sequences by modeling disassembly as an "onion

peeling" procedure where one start from the boundary components and works inwards using the disassembly tree. An algorithm for component disassembly was developed and used to perform disassemblability analysis of a component from an assembly or sub-assembly. This algorithm only considers disassembly of 1-Disassemblable subassemblies. Beasley and Martin [Beas93] went one step ahead and developed an algorithm for 2-disassemblable subassemblies, but they only considered the objects built from integral number of cubes.

2.2 Multi-Stage Mold Design

Kumar and Gupta [Kuma02] developed an algorithm to design multi-stage molds for producing multi-material objects. In order to find a feasible mold-stage sequence, the algorithm decomposes the multi-material object into a number of homogeneous components to find a feasible sequence of homogeneous components that can be added in a sequence to produce the desired multi-material object. The algorithm starts with the final object assembly and considers removing components either completely or partially from the object one at a time such that it results in the previous state of the object assembly. If a component can be removed from the target object leaving the previous state of the object assembly a connected solid, they consider such decomposition a valid step in the stage sequence. This step is recursively repeated on new states of the object assembly until the object assembly reaches a state where it only consists of one component. When an object-decomposition has been found that leads to a feasible stage sequence, the gross mold for each stage is computed and decomposed into two or more pieces to facilitate the molding operation. The limitations of their algorithm are as follows. The contact surface between homogeneous components is assumed to be planar. This limits the types of material interfaces in the multi-material object that can be handled by the algorithm. The object decomposition algorithm does not always find a feasible object partitioning sequence because it only decomposes components along the material interfaces.

Li and Gupta [Li04] extended the work presented in [Kuma02]. They presented a geometric algorithm for automated design of multi-stage mold designs for rotary-platen process. The algorithm is limited to two-material two-lump objects. It consists of two steps – determination of molding strategy and creation of mold pieces. In the first step, a molding strategy is determined depending on the geometry of the two lumps. The molding strategy consists of the number of mold stages and fabrication sequence that will be required to mold the object. They show that a two-material two-lump object can be molded in either two or three stages. In the second step, the algorithm automatically generates the mold pieces for different mold stages. The limitations of their algorithm are as follows. First, only cylindrical undercut features are handled by this algorithm. Second, the input object cannot have more that two materials and two lumps. Third, material-based precedence constraints are not incorporated. Finally, configuration of articulated assemblies cannot be changed during the molding process. Yin et al. [Yin06] recently proposed an algorithm to automatically generate a sequence of mold stages to produce a multi-material object. But their algorithm also does not consider articulated assemblies.

3 Problem Formulation

In multi-stage molding, a multi-material articulated assembly $A = \{a_1, \ldots, a_m\}$ is produced using a sequence of molding stages $S = (s_1, \ldots, s_n)$. The number of molding stages n may not be equal to the number of components m. In each molding stage s_i, a set of components C_i is added to the already molded sub-assembly A_{i-1} to produce A_i, such that

- $A_i = A_{i-1} \cup C_i = \bigcup_{j=1}^{j=i} C_j$
- $A_0 = \emptyset$
- $A_n = A$

A molding plan consists of a sequence of molding stages required to mold an articulated assembly. A molding stage s_i is represented as:

$$s_i = (C_i, T_i) \tag{1}$$

where

- C_i is the set of components to be molded in the i^{th} molding stage, and
- T_i represents the configuration of subassembly A_i, given by.

$$T_i = (\Theta, \bar{T}) \tag{2}$$

where

- $\Theta = \{\theta_1, \ldots, \theta_m\}$ are the joint coordinates, and
- $\bar{T}$ is the homogeneous transformation applied to the whole assembly.

Figure 2 shows the molding plan for a swash plate. Figure 2a shows the swash plate in the given initial configuration. Figure 2b shows the first molding stage s_1. The component set C_1 consists of two components c_1 and c_3. The molding configuration T_1 is obtained by rotating c_1 about the joint axis $\vec{t_1}$ by $-30°$. Figure 2c shows the second molding stage in which c_2 is molded.

3.1 Feasibility of a Molding Plan

A molding plan $S = (s_1, \ldots, s_n)$ is considered feasible for a given multi-material articulated assembly $A = \{a_1, \ldots, a_m\}$ if:

1. Each molding stage s_i is valid. A molding stage $s_i = (C_i, T_i)$ is considered valid if it satisfies the following constraints:
   (a) *Concurrency constraints.* Each component in C_i has the same material attribute m_i and is connected to a common base component a_i
   (b) For the given configuration T_i, the components of subassembly $A_i = \bigcup_{j=1}^{j=i} C_j = \{a_1, \ldots, a_k\}$ do not
      i. *Intersection constraints.* Intersect with each other, i.e., $a_p \cap^* a_q = \emptyset, \forall p, q \in \{1, \ldots, k\}$ and $p \neq q$
      ii. *Shadow constraints.* Cast shadows on each other, i.e., $\bar{a_p} \cap^* \bar{a_q} = \emptyset, \forall p, q \in \{1, \ldots, k\}$ and $p \neq q$, where $\bar{a_i}$ is projection of a_i on the x-y plane (because the parting direction is along the z-direction). Figure 3 shows an invalid configuration where component A casts shadow on component B.
2. Each component a_i is assigned to exactly one molding stage

3.2 Relative Cost of a Molding Plan

The main difference between two solution paths is the number of molding stages and molding configuration for each molding stage. Hence we only consider following relative costs.

1. *Relative molding cost*: This represents the cost of molding a stage. It represents setup time, cooling time, and ejection time. The setup time is the time taken to reconfigure the mold before a molding stage. It usually requires a constant time and hence incurs a constant cost. The cooling time is the time taken to cool the injected part before it can be ejected out of the mold. The cooling time is directly proportional to the wall thickness of the part. The ejection time is the time taken to eject the molded components. If there are undercuts on the components, side actions are required to form them. Depending on the production volume, the side actions can be actuated by a cam mechanism or a human being. Operating a side action complicates ejection and takes time. Hence the ejection time is directly proportional to the number of undercuts on the molded components.

$$C_m = k_1 + (k_2 T_h^2) + (k_3 N_u) \tag{3}$$

where,

- k_1 is the constant stage setup cost which consists of mold loading time and stage transfer time. Based on an analysis of a typical injection molding shop operation, we will use $k_1 = \$1.60$.
- T_h is the maximum wall thickness (in mm) of the injected components [Boot01]. Based on an analysis of a typical injection molding shop operation and a typical thermoplastic material, we will use $k_2 = \$0.8/mm^2$.
- N_u is the total number of undercuts on the components for which side actions are required. Based on an analysis of a typical injection molding shop operation, we will use $k_3 = \$0.14$.

2. *Relative defect cost*: This represents the cost of producing defective components. Constructing parting lines as 'flat' as possible is one of the best mold design practices followed in the molding community. The parting line defines the profile of the contact surface (shutoff surface) between the core and cavity. A flat parting line results in an accurate and high precision shutoff surface [Ravi90]. It also increases the sealing pressure between the core and cavity, which in turn reduces the material flash. In other words, a flat parting line reduces the defect cost. Hence the defect cost can be described in terms of the flatness of parting line as follows:
$$C_d = 1/\rho \tag{4}$$
where ρ is a measure of flatness of the parting line. Increasing the flatness decreases the defect cost.

3. *Relative tooling cost*: This represents the cost of manufacturing the mold for a molding stage. The cost mainly depends on the time taken to machine the shutoff surface. The cost of machining a surface patch s is given by:
$$C(s) = k_4 A_s + k_5 N_s \tag{5}$$
where,
   - A_s is the surface area (in sq. inch) of s. Based on an analysis of a typical mold making shop operation, we will use $k_4 = \$100/inch^2$.
   - N_s is the number of surface normals on surface patch s (s is represented as a connected set of facets). Based on an analysis of a typical mold making shop operation, we will use $k_5 = \$8.5$.

   Hence the relative tooling cost can be written as:
$$C_t = k_4 A_s + k_5 N_s \tag{6}$$
   where A_s is the area of the shutoff surface and N_s is the number of normals on the shutoff surface.

The molding cost is usually very large compared to the defect cost and the tooling cost. The tooling cost is a fixed cost while the molding cost is a running cost. The molding cost becomes more significant as the production volume increases. This paper performs a hierarchical optimization to minimize the cost of a molding stage. We first optimize the molding cost, then defect cost, and finally tooling cost. When comparing the cost of two candidate molding stages, we only compare the molding cost of the two. The defect cost and tooling cost are used only in case of a tie.

3.3 Problem Statement

Input: Multi-material articulated assembly A.

- Each component is a polyhedron, that is, a solid bounded by a piecewise linear surface. The boundary of the polyhedron (union of vertices, edges, and facets on the surface) is a connected 2-manifold. Each facet of the polyhedron is a triangle.
- The components do not have any internal shell. A hollow part (having internal shells) is not moldable.

Output: A molding plan, which is a sequence of molding stages $S = (s_1, \ldots, s_n)$. Each mold stage s_i is represented as a tuple (C_i, T_i) as defined in Equation 1.

- The molding plan is feasible as defined in Section 3.1
- The molding plan is optimal, i.e., the cost of the molding plan as defined in Section 3.2 is minimum.

4 Overview of Approach

We formulate the molding planning problems as a state-space search problem. The search space is represented as a tree $T = (N, E, S, G)$, where

- N is the set of nodes in the tree. Each node in the search tree represents a search state, i.e., an intermediate assembly.
- E is the set of edges between the nodes. Each edge in the search tree represents a molding operation or stage. Each molding stage is described by the set C_i of components to be molded, called stage components and the configuration T_i of the subassembly in which the molding will take place. It is mathematically represented as a two-tuple (C_i, T_i) as in Equation 1. Each edge or mold stage has an associated manufacturing cost.
- S is the root node of the search tree, which is an empty assembly.
- G is the set of leaf nodes in the tree. The leaf nodes of the search tree are either nodes corresponding to the final assembly, or nodes corresponding to the intermediate subassemblies that have infeasible molding configuration. The leaf nodes that do not correspond to the final part are called *blocked nodes*. New search nodes cannot be generated from the blocked nodes.

Figure 4 shows a portion of the state space for the swash plate shown in Figure 2a. A solution is a path through this graph from the start node S to a goal node in G. The path with the lowest manufacturing cost is the optimal solution.

The state-space search problems are known to be combinatorial optimization problems. For an assembly with m components, a brute-force approach that evaluates all possible states will take $O(m!)$ time. The state-space search problems are usually solved using branch and bound algorithm with a lower time complexity. The main objective in a branch and bound algorithm is to perform an enumeration of the alternatives without evaluating each search node. We use a heuristic variant of the branch and bound algorithm, which is a combination of Depth-First Search (DFS) as the overall principle and Best-First Search (BeFS) when choice is to be made between nodes at the same level of the search tree.

In order to use branch and bound search effectively, we need to overcome two challenges – large number of search nodes and high node-generation time. Due to the exponential nature of the problem, even a moderate size assembly can lead to a large state space. In a typical case, very few feasible plans exist, and the algorithm wastes a lot of time evaluating infeasible solution paths. Generating a new search node involves extensive geometric reasoning which takes a lot of time. We use the following two techniques to solve the state-space search problem efficiently:

1. *Reduce the search space.* We use proven molding plans for individual joints in the assembly to reduce search space. Several experimentally-validated plans have been presented in [Priy06b, Priy07] for prismatic, revolute, and spherical joints. In addition to satisfying the mold-stage feasibility constraints described in Section 3.1, we also satisfy the feasibility constraints imposed by the molding plans for joints. This reduces the search space and also ensures that our method will only generate feasible plans. The joint molding plans includes (a) precedence constraints, (b) joint-axes constraints, and (c) joint-parameter constraints. The precedence constraints specify a sequence in which the connected components should be molded to obtain proper joint clearance. The joint axes and parameter constraints specify a feasible configuration space of the joint axes such that the parting line does not lie on the joint interfaces. If the parting line lies on the joint interface, the flash between the connected components will cause jerky motion.

2. *Reuse the results of a search node.* Each node in the search tree contains an intermediate subassembly as shown in Figure 4. A particular subassembly can be reached via different paths in the search tree. Any two molding stage sequences that cover the same set of components will lead to the same subassembly. For example the molding sequences [Mold $C_3 \rightarrow$ Mold C_2] and [Mold $C_2 \rightarrow$ Mold C_3] in Figure 4 will result in the same intermediate subassembly containing C_2 and C_3. So the subtree below both the nodes will be identical. This observation is used to reuse the results of previously visited search nodes and avoid repetitive computations. When we need to branch a node, we first check if the

subassembly in the node has already been reached from a different path. If such a node is found, we reuse the solution path from that node to the goal node.

5 The Search Algorithm

We use a heuristic variant of the branch and bound algorithm, which is a combination of Depth-First Search (DFS) as the overall principle and Best-First Search (BeFS) when choice is to be made between nodes at the same level of the search tree.

Our planning problem involves short sequences. Currently, it is not practical to do more than four shots on injection molding machines. Moreover, complex geometric computations need to be performed to determine the best assembly configuration during a molding step. Finally, this planning problem has an objective function which enables use of hierarchical optimization. Based on these considerations, we have selected DFS as our planning algorithm. In DFS, a live node with the largest level in the search tree is chosen for exploration. An advantage of this strategy is that the memory requirements in terms of number of nodes to store at the at the same time is bounded above by the number of levels in the search tree multiplied by the maximum number of children of any node. It allows the use of recursion to traverse the tree, which enables one to store the information about the current subproblem in an incremental way, so that only the constraints added in connection with the creation of each subproblem need to be stored. It also quickly produces a feasible solution. However, if the incumbent is far from the optimal solution, large amounts of unnecessary computations take place.

Combining BeFS to choose between the nodes at the same level of the search tree avoids this problem. BeFS proceeds preferentially through nodes that problem-specific heuristic indicates might be on the best path to a goal. A heuristic evaluation function is used to help decide which node is the best one to explore next. Given a node n in the search tree, the heuristic evaluation function $f(n)$ estimates the total path cost of going from a start node to a goal node via n. The value returned by $f(n)$ is an underestimate and hence can also be used as a bounding value for discarding unpromising solution paths. BeFS selects the node for which $f(n)$ is minimum. The idea is that exploring the node with minimum estimated cost first hopefully leads to a *good* feasible solution.

Algorithm GenerateMoldingPlan
Input:

1. Multi-material articulated assembly $A = \{a_1, \ldots, a_m\}$.
2. Each component a_i is a lump with an associated material attribute m_i.
3. Each joint j_k in the assembly has an associated molding plan p_k.

Output: A sequence of molding stages $S = (s_1, \ldots, s_n)$
Steps:

1. Initialize solution:
   - IncumbentSolution := $\emptyset$
   - IncumbentCost := ∞
2. Initialize search:
   - $A_0 := \emptyset$
   - $P_0 := \{A_0\}$
3. Start search: ProcessNode(P_0)
4. Return IncumbentSolution

The algorithm GenerateMoldingPlan recursively traverses a search tree to return the molding plan with minimum cost. The algorithm initializes the search with a node containing an empty assembly. It then calls the algorithm ProcessNode that recursively builds the assembly by inserting components. The variables IncumbentSolution and IncumbentCost are global that are updated by the algorithm ProcessNode whenever a solution better than the incumbent solution is found.

Algorithm ProcessNode
Input: Search node P containing the current subassembly A_P
Output: Updates the global variables IncumbentSolution and IncumbentCost defined in algorithm GenerateMoldingPlan *Steps*:

1. $S :=$ Path from P_0 to P
2. If $A_P = A$, then
   (a) If $\text{Cost}(S) <$ IncumbentCost, then
      - IncumbentSolution $:= S$
      - IncumbentCost $:= \text{Cost}(S)$
   (b) Return.
3. If A_P has already been processed, then
   (a) Retrieve the shortest path S_P from A_P to A
   (b) $S := S \cup S_P$
   (c) If $\text{Cost}(S) <$ IncumbentCost, then
      - IncumbentSolution $:= S$
      - IncumbentCost $:= \text{Cost}(S)$
   (d) Return.
4. Find a lower-bound cost $f(P) := \text{Cost}(S) + h(P)$ as described in Section 5.1
5. If $f(P) \geq$ IncumbentCost then return.
6. Branch on P generating children search nodes $P' = \{P_1, \ldots, P_q\}$ using algorithm GenerateMoldingStages(A_P) described in Section 6
7. If $|P'| = 0$, i.e., no feasible molding stage is possible for the current subassembly then return.
8. LivePriorityQueue $:= \bigcup\{(P_i, f(P_i))\}$
9. Repeat until LivePriorityQueue $= \emptyset$
   (a) Extract a node P_i with minimum lower-bound cost $f(P_i)$ from LivePriorityQueue to be processed
   (b) ProcessNode(P_i)

The algorithm ProcessNode processes a node in the search tree. Each node P in the search tree stores the current subassembly A_P. The search node is either fathomed or branched into multiple search nodes. A search node is fathomed in three scenarios:

1. The search node is a leaf node, i.e., the current subassembly A_P is the final assembly A.
2. The subassembly A_P contained in the search node has already been processed before, and the result can be reused.
3. If the lower bound cost $f(P)$ is no better than the incumbent. The current solution path is not promising because no feasible solution of the subproblem can be better than the incumbent solution.

Whenever a new solution is found, it is compared with the incumbent solution. If it is better, the incumbent solution is updated with the new solution. If a search node cannot be fathomed, the possibility of a better solution cannot be ruled out. The node is branched into multiple search nodes $\{P_1, \ldots, P_q\}$. The new nodes P_i to be processed are inserted into a priority queue indexed on the lower-bound cost $f(P_i)$. Extracting a node from the priority queue always returns a node with the lowest lower-bound cost. Therefore we always process the best node available, which is why this is called the best-first search.

5.1 Bounding Function

The bounding function is the key component of any branch-and-bound algorithm. Given a node P, the bounding function $f(P)$ estimates the total path cost of going from a start node to a goal node via P. Ideally the value of a bounding function for a given subproblem should be equal to the value of the best feasible solution to the problem, but obtaining this value itself is NP-hard. So a trade off is struck between quality and time when dealing with bounding function. The algorithm described above uses the following bounding function for a node P:

$$f(P) = \textsc{Cost}(S) + h(P)$$

where,

- $\textsc{Cost}(S)$ is cost of the path S from P_0 to P
- $h(P)$ estimates the cost of the path from P to a goal node containing the final assembly A

The path S is essentially the sequence of molding stages used to reach P from P_0. The function $\textsc{Cost}(S)$ returns the sum of the cost of all molding stages in the sequence.

The bounding function $h(P)$ must be an underestimate so that a solution path can be safely pruned. Each node in the search tree stores the current subassembly A_P. We need to find a lower bound cost for molding the remaining set of components $\{A - A_P\}$. We calculate the lower bound by *relaxing* the precedence constraints coming from the joint molding plans. We still follow the feasibility constraints given in Section 3.1. We assume that all components of the same material and connected to a common base component can be molded in a single stage. We also assume that all the components in a particular stage can be oriented in a configuration that minimizes the number of undercuts. These assumptions make the cost parameters (cooling time and number of undercuts) of a stage independent of molding sequence. Hence the cost of molding each component can be calculated offline and reused to quickly compute a lower bound cost for a subproblem.

5.2 Branching Rule

In algorithm ProcessNode, a branching rule is applied to a search node if it cannot be discarded. The node is branched into multiple nodes to be investigated in subsequent iterations. We refer to the node from which new nodes are generated as the starting node and the nodes that are being generated as the target nodes. All branching rules can be seen as subdivision of a problem (starting node) into two or more subproblems (target nodes). The search is considered converging if the size of each generated subproblem is smaller than the original subproblem.

Each node P in the search tree stores the current subassembly A_P. The problem represented by this search node is generating a feasible molding plan for the subassembly $\{A - A_P\}$. For example, the problem represented by node N_1 in Figure 4 is generating a feasible molding plan for a subassembly consisting of components C_1 and C_2.

We branch a node by generating molding stages. A molding stage adds a set of components to the current subassembly (starting node) to create a new subassembly (target node). The molding stage is represented by the directed edge between the starting node and the target node. For a given subassembly, we can generate multiple molding stages by selecting different sets of stage and base components. The number of subproblems that can be generated from a problem is equal to the number of feasible molding stages for a subassembly. For example, the node N_1 in Figure 4 can be subdivided into two nodes N_2 and N_3 by molding C_1 and C_2 respectively. The subdivided nodes N_2 and N_3 represent smaller problems of generating a molding plan for

C_2 and C_1 respectively. The algorithm for generating molding stages for a given subassembly is described in Section 6.

6 Generating Molding Stages

Figure 5 shows the steps involved in generating a molding stage. The middle column in the figure shows the steps that need to be executed in the specified sequence. The right column is the set of constraints extracted from the molding plans of the joints. The left column is the set of problem constraints. Section 6.1 describes an algorithm to find the stage components C_i. Section 6.2 describes a method to determine $\bar{T}$. Section 6.3 and Section 6.4 determine the joint coordinates θ_i for stage components. We use an incremental approach to determine θ_i for pre-stage components. We incrementally change the joint parameter θ_i for each pre-stage component a_k and find all the feasible configurations. The algorithm for generating all possible molding stages for a subassembly is described below.

Algorithm GenerateMoldingStages
Input: Current subassembly A_P
Output: All possible molding stages $S_P = (s_P^1, \ldots, s_P^q)$ for A_P
Steps:

1. Find the sets of stage components $C = \{C_1, \ldots C_q\}$ using algorithm FindStageComponents described in Section 6.1
2. $S_P = \emptyset$
3. For each $C_i \in C$ do
   (a) Find a component $b \in A_P$ that is connected to all components in C_i. In case of multiple such components, any one can be arbitrarily chosen
   (b) Orient the base component b to determine $\bar{T}$ as described in Section 6.2
   (c) $\Theta = \emptyset$
   (d) For each stage component $a_j \in C_i$ do
      i. Find a feasible and optimal configuration θ_j as described in Section 6.3 and Section 6.4
      ii. $\Theta = \Theta \cup \theta_j$
   (e) Pre-stage components $A_Q = A_P - b$
   (f) For each pre-stage component $a_k \in A_Q$ do
      i. Iteratively find a feasible configuration θ_k
      ii. $\Theta = \Theta \cup \theta_k$
   (g) $S_P = S_P \cup \{\Theta, \bar{T}\}$
4. Return S_P

6.1 Finding Stage Components and the Base Component

This is the first step of generating a molding stage for the current subassembly. Before we can generate a molding stage, we need to find the components that can be added to the current subassembly in a molding stage.

Algorithm FindStageComponents
Input:

1. Input assembly A
2. Current subassembly A_P

3. Precedence constraints G: The precedence constraints are derived from the joint molding plans. Each joint molding plan specifies a sequence in which the connected components need to be molded. This defines a partial ordering on the assembly components, which is represented as Directed Acyclic Graph (DAG).

4. Concurrency constraints: The concurrency constraints represent the requirement that all components molded in a single stage must be of the same material and connected to a common base component. The parameters for this constraint are available in the input assembly model. Each component model a_i has a material attribute m_i and mating data in the assembly provides the connectivity information.

Output: Sets of stage components $C = \{C_1, \ldots C_q\}$. Each C_i is a set of components that can be molded in the next stage.
Steps:

1. For each component a_i in A_P remove a_i and associated edges from G

2. $U :=$ set of all nodes in G for which in-degree count is zero

3. Partition U into groups of components $C = \{C_1, \ldots C_q\}$ such that $C_i \subset U$ and $\cup C_i = U$.

4. Return C.

6.2 Orienting the Base Component

The configuration T_i of an assembly is defined in Equation 2 as a tuple $(\Theta, \bar{T})$ where $\Theta = \{\theta_1, \ldots, \theta_m\}$ are the joint coordinates and $\bar{T}$ is the homogeneous transformation applied to the whole assembly. In this step, we determine $\bar{T}$ to orient the base component such that joint-axes constraints are satisfied. The molding plan for a joint specifies a feasible configuration space of the joint axes. For example, the joint axis for a revolute joint must be perpendicular to the parting direction. A molding stage forming a component of a joint must be configured such that the joint axis is within the feasible configuration space.

Orienting the base component to satisfy the joint-axis constraints amounts to building a transformation matrix $\bar{T}$. Here we are interested in orienting an axis or a vector along a particular direction. Since a vector always passes through the origin, this transformation is equivalent to one or more rotations about the three principle coordinate axes.

6.3 Finding the Feasible Configuration Space for a Stage Component

This step describes a method to determine feasible configuration space for a stage component. In order for a molding stage to be feasible, the parameter θ of the joint between the stage component a and the base component b must be defined such that following three constraints are satisfied:

1. *Joint parameter constraints.* θ must be within the range specified by the joint molding plan, i.e., $\theta^l \leq \theta \leq \theta^u$

2. *Intersection constraints.* The stage component does not intersect with the base component, i.e., $a \cap^* b = \emptyset$.

3. *Shadow constraints.* The stage component and the base component do not cast shadow on each other, i.e., $\bar{a} \cap^* \bar{b} = \emptyset$, where $\bar{a}$ and $\bar{b}$ are projections of a and b on the x-y plane (because the parting direction is along the z-direction).

We sweep the stage component over the initial range $[\theta^l, \theta^u]$ specified by the joint molding plan. The sweep is a translation or rotation depending on on whether the stage component is connected to a prismatic or revolute joint. We then find the actual ranges for which the stage component does not intersect with or cast shadow over the base component. This partitions the initial range specified by the molding plan into sets of feasible and infeasible ranges.

If shadow constraint is satisfied for a particular configuration, the intersection constraint is automatically satisfied. Hence, we can conclude that if we sweep the stage component a over the initial range $[\theta^l, \theta^u]$

specified by the joint molding plan, the set of joint parameters θ for which the projection of the stage component a and the base component b do not intersect, constitutes the feasible configuration space of a.

The projection of a triangulated polyhedron p onto a plane consists of a set of triangles. A projection $\bar{p_1}$ intersects with another projection $\bar{p_2}$ if any triangle in $\bar{p_1}$ intersects with any triangle in $\bar{p_2}$. A triangle t_1 intersects with another coplanar triangle t_2 if any edge in t_1 intersects with any edge in t_2 or it is completely enclosed inside t_2. Hence, the intersection of projection of two polyhedrons can be tested by just considering projected edges. We can further reduce the number of projected edges to be tested by just intersecting the silhouette the two polyhedrons. The projection of a polyhedron is a simple polygon with holes. The boundary of this polygon is called silhouette of the polyhedron.

Let us first consider prismatic joints. A proven molding plan for prismatic joint specifies that the parting direction be perpendicular to the joint axis, i.e., joint axis is in the x-y plane. If a stage component is connected to the base component via a prismatic joint, the sweep of a vertex v of the stage component is a line segment. We project this line segment onto the x-y plane to get another line segment l. It can be seen in Figure 6a that the overlap status of an edge e connected to this vertex v can only change at the intersections of l and the silhouette h of the base component. Figure 6b shows that for each edge on the stage component, the initial feasible configuration space θ is partitioned into feasible and infeasible ranges. The feasible ranges for each edge e_i can be represented as $\{[\theta_j^1, \theta_j^2], \ldots, [\theta_j^{k-1}, \theta_j^k]\}$ such that $\theta_j^1 \geq \theta^l$ and $\theta_j^k \leq \theta^u$. The final feasible range for the stage component is the intersection of the feasible ranges for each edge on the stage component.

$$\theta = \cap_{j=0}^{m} \{[\theta_j^1, \theta_j^2], \ldots, [\theta_j^{k-1}, \theta_j^k]\}$$

As the stage component is translated along the joint axis, the silhouette of the stage component does not change. Hence we can further optimize the implementation by only considering the silhouette edges of the stage component.

If the stage component is connected to the base component via a revolute joint, the sweep of a vertex on the stage component is a circular arc. The projection of this arc on the x-y plane is again a line segment. Therefore we can use the same scheme used for prismatic joint. However, as the stage component is rotated, its silhouette continuously changes. Hence we need to consider all edges of the stage component.

Theorem 1. *Let P_a be a stage component with n_a vertices connected to a base component P_b with n_b vertices. The feasible configuration space for P_a can be calculated in $O(n_a n_b \log n_b)$ time if the type of joint between P_a and P_b is prismatic or revolute.*

Proof. The first step is to find the silhouette h of the base component. This is accomplished by projecting the facets of the base component onto the x-y plane and finding the union of the projected facets. The union can be found by plane-sweep algorithm which takes $O(n_b \log n_b)$ time.

Now consider each vertex of the stage component. The projection of the trajectory of each vertex of the stage component as it is translated or rotated, depending on whether it is connected to a prismatic or revolute joint, is a line segment l on the x-y plane. The number of potential intersections between l and the silhouette h is n_b. We need to determine the status of l at each intersection point whether it is entering or exiting h. This can be done by sorting the intersection points and finding the status of an endpoint of l. The status of l alternates at each successive intersection point. Sorting the n_b intersection points takes $O(n_b \log n_b)$ time. Finding whether a point lies inside or outside h takes n_b time. Hence it takes $O(n_b \log n_b)$ to process each vertex of the stage component. Processing n_a vertices will therefore take $O(n_a n_b \log n_b)$ time. □

The sweep of a stage component connected via a spherical joint is too complicated for the scheme presented above. We use an iterative scheme to handle spherical joints.

6.4 Finding an Optimal Configuration for a Stage Component

The previous section finds a feasible configuration space (range of joint parameter θ) for a stage component. This section chooses a configuration from the feasible range that minimizes the molding stage cost. From Equation **??**, we need to optimize the molding cost (C_m), defect cost (C_d), and tooling cost (C_t). As explained in Section 3.2, we follow a hierarchical approach in optimizing the parameters of a molding stage. We first optimize the molding cost, then defect cost, and finally tooling cost.

In the first step, we further subdivide the feasible range of joint parameter such that the number of undercuts in each subrange is constant. In other words, moving the stage component within the subdivided range does not change the number of undercuts. We select the range with the minimum number of undercuts. In the next step, we find the configuration within the selected range for which the parting line is flattest. It should be noted that this method is only applicable for revolute joints. The motion of a prismatic joint in a straight line does not change the number of undercuts or flatness of the parting line. In the final step, we construct a shutoff surface for which the machining cost is minimum. The final step is described in [Priy06b]. The first two steps are described in the following sections.

6.4.1 Minimizing the number of undercuts

Our first algorithm for minimizing the number of undercuts is inspired by the theoretical algorithm of Ahn et al. [Ahn02], who prove that all combinatorially distinct parting directions correspond to 0-, 1-, or 2-cells in an arrangement of *great circles* G_c on a Gaussian sphere, which is a unit sphere centered at the origin such that every point on it defines a direction in Euclidean 3-space. A great circle is the intersection of a sphere with a plane going through its center. Every facet normal and normal of the triangle formed by every edge-vertex pair of the part generates a great circle in their arrangement. These great circles correspond to the directions where a part face changes from front-facing to back-facing (directions contained in the plane of the face), and directions where a projection of one part face potentially changes from occluding to not occluding (or vice versa) another part face (directions contained in the planes through an edge-vertex pair from separate triangles).

The parting direction and the orientation of a stage component is equivalent. Rotating the parting direction clockwise is equivalent to rotating the stage component anticlockwise. For the sake of simplicity let us keep the parting direction fixed. The feasible ranges of revolute joint parameters can be represented as great arcs G_a on the Gaussian sphere. The great circles G_c intersect and subdivide the great arcs G_a. By construction of the great circles G_c, the status of a face (front-facing, back-facing, occluding, non-occluding) does not change within a subdivision. Hence the number of undercuts in a subdivision of G_a cannot change.

Theorem 2. *Let P be a stage component with n vertices connected to a base component with a revolute joint. The configuration with the minimum number of undercuts can be found in $O(n^4)$ time.*

Proof. A great circle is formed for every edge-vertex pair on the part. Hence the number of great circles G_c that can be drawn for a polyhedron with n vertices is $O(n^2)$. Since a great circle intersects with every other great circle on the sphere at two diametrically opposite points, the number of potential subdivisions of the great arcs G_a representing the feasible ranges of revolute joint parameters is $O(n^2)$. Hence we need to test $O(n^2)$ orientations. It takes $O(n^2)$ time to test each orientation because each facet needs to tested against every other facet. The overall complexity hence becomes $O(n^4)$. □

It should however be mentioned that although the theoretical worst-case complexity of this algorithm is $O(n^4)$, it behaves almost as $O(n^3)$. The proof of Theorem 2 states that it takes $O(n^2)$ time to test each orientation because each facet needs to tested against every other facet. However Priyadarshi and Gupta [Priy06a] present an algorithm for detecting undercuts that runs on GPU. Using the GPU-based algorithm, testing $O(n^2)$ directions is not that expensive.

6.4.2 Determining the flattest parting line

The previous section chooses a feasible range of joint parameters for which the number of undercuts is minimum. This section chooses a joint parameter within the feasible range for which the parting line is flattest. Once a feasible range is found, we calculate the mold-piece regions of the stage component for any orientation within the feasible range. The algorithm for calculating the mold-piece regions is described in [Priy06a]. The parting line is the set of boundary edges between the core region and the cavity region. If we rotate the stage component within the feasible range, the status of the mold-piece regions does not change, and hence the parting line also does not change. However, the flatness of the parting line with respect to the parting direction changes. Figure 7 shows a simple 2D case where the parting direction is along the z-direction and the joint axis is along the x-axis.

For simplicity, we assume that the lower bound of the feasible range is zero and the upper bound is θ, i.e., the selected feasible range is $[0, \theta]$. We need to find an angle α within this range for which the parting line is flattest. Before we can develop a method, we must formally define the notion of flatness of a parting line. Let us consider the scheme shown in Figure 7c. The parting direction is along the z-direction and the joint axis is along the x-axis. Let AB be the projection of a line segment in the parting line onto the yz-plane. Our measure of flatness of AB is:

$$\rho(AB) = (z_A - z_B)^2 = l_i^2 sin^2(\alpha_i)$$

where l_i be the length of AB and α_i be the angle between the y-axis and AB. Note that $\rho(AB) \geq 0$, with equality holding if and only if AB is perpendicular to the z-axis (parting direction). In general, the smaller the value of $\rho(AB)$, the flatter is AB. If AB is rotated to CD along the x-axis (joint axis) by an angle α as shown in Figure 7c, the new measure of flatness would be:

$$\rho(CD) = l_i^2 sin^2(\alpha_i + \alpha)$$

The optimization problem can now be formally stated as follows. Create the parting line $L = \{e_1, \ldots, e_k\}$ for the stage component oriented with $\alpha = 0$. Project each parting line edge e_i onto the yz-plane to create a line segment of length l_i that makes an angle α_i with the y-axis.

$$\begin{array}{cc} Minimize & f(\alpha) = \sum_{i=0}^{k} l_i^2 sin^2(\alpha_i + \alpha) \\ s.t. & 0 \leq \alpha_i \leq \theta \end{array} \tag{7}$$

Differentiating the minimization function gives:

$$\sum_{i=0}^{k} 2l_i^2[(2\sin\alpha_i \cos\alpha_i)\sin^2\alpha - (1 - 2\sin^2\alpha_i)\sin\alpha\cos\alpha + (\sin\alpha_i\cos\alpha_i)] = 0 \tag{8}$$

Solving the above equation gives α for which the parting line is flattest.

Theorem 3. *Let P be a stage component connected to a base component with a revolute joint. Let $[0, \theta]$ be a feasible range of configurations inside which the status (front-facing, back-facing, occluding, non-occluding) of any facet on P is invariant. Let $L = \{e_1, \ldots, e_k\}$ be the parting line of P for the configuration $\alpha = 0$. The configuration $0 \leq \alpha \leq \theta$ of the polyhedron for which the parting line is flattest can be found in $O(k)$ time.*

Proof. In the first step, each parting line edge is projected onto the yz-plane which takes $O(k)$ time. In the next step Equation 8 is formed and solved. Summing up the k terms takes $O(k)$ time and solving it takes constant time. Hence the overall complexity of the algorithm is $O(k)$. □

7 Results

We will illustrate the steps for generating molding plan with the help of three examples – swash plate, vent assembly, and universal joint.

7.1 Swash Plate

A swash plate is a mechanical device used in helicopters to control the motion of the main rotor blades. An example swash plate is shown in Figure 2a. The swash plate consists of three rings connected together by revolute joints. The inner and outer rings, C_1 and C_3 respectively are made of the same material ABS. The middle ring C_2 is made of polyethylene. The diameter of the hole and pin in the revolute joint is 1/4 in. In addition to the assembly model, the designer also provides a molding plan for each joint in the assembly. The swash plate example has two revolute joints. The designer compares the type, size, geometry, and material of the joints to find a molding plan for the joints. The assembly model of the swash plate and the selected molding plan is fed to the planner.

The planner first creates a Directed Acyclic Graph of components using the precedence constraints in the joint molding plan. The plan specifies that pin must be molded after hole, i.e., component C_2 must be molded after component C_1 and C_3. Figure 8 shows the precedence constraints for the assembly.

The planner next calculates the lower bound cost $h(P)$ of molding each component separately. These values are used to compute the bounding values $f(P)$ for the search nodes. As explained in Section 5.1, we only consider the relative cost between two solution paths. Figure 4 shows a partial state space for the swash plate example. The node numbers $(N_1, \ldots, N_8)$ correspond to the sequence in which each node is processed by the algorithm ProcessNode. Our techniques for solving the state-space search efficiently works very well on this example. The algorithm fully processes only two nodes – N_1 and N_2. It also calculates the bounding values for nodes N_3 and N_6, which is simply the sum of molding costs calculated offline. It can also be seen that nodes N_4 and N_7 are similar to each other and to the node N_1. Hence even if the algorithm had to process these nodes, the cached results from node N_1 could be simply used.

In the first stage, we choose node N_1 that has the minimum bounding value. In the second stage (node N_2), we just need to find a configuration for the assembly. We have one stage component C_2 and choose C_1 as the base component. The joint-axis constraints from the joint molding plan are used to calculate $\bar{T}$ and orient C_1. We find a feasible configuration space for component C_2 such that C_1 and C_2 do not occlude each other along the parting direction as shown in Figure 9. We then find a configuration within the feasible configuration space such that the number of undercuts is minimum and the parting line is flattest as shown in Figure 10. Figure 2 shows the plan generated for the swash plate.

7.2 Vent Assembly

A vent assembly is used at the end of intake or exhaust to regulate air flow. It is most commonly found on automobile dashboards. Figure 11 shows an example vent assembly. It consists of six components - one main body (C_1) and five vanes ($C_2, \ldots, C_6$). The main body is made of ABS, while the vanes are made of polyethylene. The vanes are connected to the main body via revolute joints of size 1/4 in.

To obtain joint clearance, pin must be molded after hole, i.e., the vanes must be molded after component the main body. Figure 12 shows the DAG representing the precedence constraints for the assembly. As per the precedence constraints, the main body C_1 needs to be molded before the vanes. So in the first stage, only C_1 can be molded. As per the joint axis constraint, it is oriented such that the joint axis is perpendicular to the parting direction. The configuration for the first stage eliminates any undercut and makes the parting line flattest. It is shown in Figure 13a.

After the main body is molded, we have the choice of molding the vanes in any order. But since all the vanes are made of the same material, they can be molded in a single stage. Figure 13b shows the configuration of the assembly for the second molding stage. It must be noticed that in the initial configuration shown in Figure 11, the vanes cast shadow on each other. They need to oriented vertically as shown in Figure 13b to avoid this problem.

7.3 Universal Joint

A universal joint in a rigid rod allows the rod to bend in any direction. It consists of a pair of ordinary hinges located close together, but oriented at 90^o relative to each other. Universal joints are common wherever a drive shaft needs to turn a corner; a drive shaft with a universal joint can freely rotate through the universal joint, and no gears are required to couple the two ends. The most obvious example of this application of a universal joint is in the drive shafts of automobiles.

Figure 14 shows an example of universal joint. It consists of three components - two shafts (C_1 and C_3) and a link (C_2). The shafts are made of ABS, while the link is made of polyethylene. The shafts and links are connected together via revolute joints of size 1/4 in.

To obtain joint clearance, pin must be molded after hole, i.e., the link must be molded after the shafts. Figure 15 shows the DAG representing the precedence constraints for the assembly. As per the precedence constraints, the shafts C_1 and C_3 need to be molded before the link C_2. So in the first stage, we can mold C_1, C_2, or both. The shafts can be molded sequentially using the same mold or using a multi-cavity mold depending on the volume of production. As per the joint axis constraint, it is oriented such that the joint axis is perpendicular to the parting direction. The configuration for the first stage eliminates any undercut and makes the parting line flattest. It is shown in Figure 16a.

The link can now be molded in the second stage. In this stage, the link (stage component) is connected to two shafts. Hence any one of the two shafts can serve the role of a base component. We arbitrarily choose

C_1 as the base component and use the joint-axis constraints from the molding plan to orient C_1 such that the joint axis is perpendicular to the parting direction (z-direction). The other shaft C_3 is oriented using the iterative scheme used for pre-stage components. Figure 16b shows the configuration of the assembly for the second molding stage. It must be noticed that this is only valid configuration. Any other configuration suffers from the shadow problem.

8 Conclusions

This paper describes an algorithm for generating a molding plan for an articulated assembly. This algorithm produces a molding plan, which is feasible as well as optimal with respect to the manufacturing cost. The molding planning problem is a combinatorial optimization problem. We formulate it as a state-space search problem and use branch and bound to search for an optimal solution. Our state space has large number of search nodes and processing each node takes a lot of time. We handle these problems by pruning infeasible solution paths and reusing the results of a search node. This paper also presents geometric reasoning algorithms for the subproblems that need to be solved as part of the overall planning problem. These subproblems include finding stage components and assembly configuration for each molding stage. The assembly configuration found by the algorithm is such that the number of undercuts on the stage components is minimum and the parting line is flattest. The algorithms have been tested with several complex assemblies for which multiple molding plans are possible.

Current limitations and direction for future work are as following. First, the algorithm for generating the molding sequence presented in this paper does not consider the flow and thermal limitations of molding machines. It is assumed that each component is feasible to mold from the mold-flow point of view in all possible sequences and the thermal management system is capable of providing appropriate cooling and heating. This assumption may however break down for some molding stages. Hence the constraints imposed by the flow and thermal considerations should also be incorporated into the future version of the algorithm. Second, for every molding stage, we need to find a feasible configuration for pre-stage components. Currently, we use an algorithm that incrementally evaluates only discrete configurations. This algorithm suffers from aliasing issues like any other discrete sampling algorithm. It may not be able to find a feasible configuration when it actually exists. More work is required to develop an algorithm that searches the continuous configuration space. Finally, the current algorithm only deals with serial mechanisms. The parallel mechanisms add another level of complexity to the problem where we may have to handle cyclic feasibility constraints. Further work is need to extend this work to deal with parallel mechanisms.

9 Acknowledgements

This research has been supported in part by NSF grants DMI0093142 and DMI0457058, and Army Research Office through MAV MURI Program (Grant No. ARMYW911NF0410176). Opinions expressed in this paper are those of authors and do not necessarily reflect opinion of the sponsors.

References

[Ahn02] Ahn, H.K., De Berg, M., Bose, P., Cheng, S.W., Halperin, D., Matousek, J., and Schwarzkopf, O. Separating an object from its cast. Computer-Aided Design, 34, 547-59, 2002.

[Beas93] D. Beasley and R.R. Martin, Disassembly sequences for objects built from Unit Cubes, *Computer Aided Design*, 25(12): 751-761, 1993

[Boot01] G. Boothroyd, P. Dewhurst, and W.A. Knight, Product Design for Manufacture and Assembly Revised and Expanded, CRC Press 2001.

[Fazi87] T. De Fazio and D. Whitney. Simplified generation of all mechanical assembly sequences. *IEEE Journal of Robotics and Automation*, 3(6), 640-658, December 1987

[Good02] V. Goodship and J.C. Love. Multi-Material Injection Moulding. Rapra Technology LTD.: Shawbury, UK, 2002.

[Gouk06] R.M. Gouker, S.K. Gupta, H.A. Bruck, and T. Holzschuh. Manufacturing Of Multi-Material Compliant Mechanisms Using Multi-Material Molding. International Journal of Advanced Manufacturing Technology, 30(11-12):1049-1075, 2006.

[Kuma02] M. Kumar and S.K. Gupta. Automated design of multi-stage molds for manufacturing multi-material objects. Journal of Mechanical Design, Vol. 124, No. 3, pp. 399-407, 2002.

[Li04] X. Li and S.K. Gupta. Geometric algorithms for automated design of rotary-platen multi-shot molds. Computer Aided Design, 36(12):1171–1187, 2004.

[Pirk98] J. D. Pirkl. Automating the Multi-Component Molding Process. In Proceedings of Technologies for Multi-Material Injection Molding (CM98-206). Troy, Michigan, May 1998.

[Plan02] H. Plank. Overmolding-Stack-Mold Technology: An Innovative Concept in Multi-Component Injection Molding. SME Technical Papers (CM02-225), 2002.

[Priy06a] A.K. Priyadarshi and S.K. Gupta. Finding mold-piece regions using computer graphics hardware. In *Geometric Modeling and Processing Conference*, Pittsburgh, PA, July 2006

[Priy06b] A.K. Priyadarshi. Algorithms for Generating Multi-Stage Molding Plans for Articulated Assemblies. *PhD Thesis,* Department of Mechanical Engineering, University of Maryland, College Park, 2006.

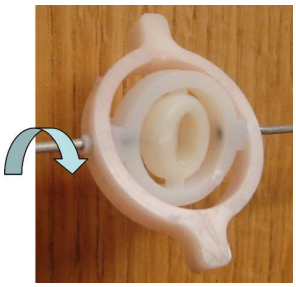
[Priy07] A.K. Priyadarshi, S.K. Gupta, R. Gouker, F. Krebs, M. Shroeder, and S. Warth. Manufacturing Multi-Material Articulated Plastic Products using In-Mold Assembly. International Journal of Advanced Manufacturing Technology, 32(3-4):350-365, March 2007.

[Ravi90] B. Ravi, and M.N. Srinivasan. Decision Criteria for Computer-Aided Parting Surface Design. *Computer-Aided Design,* 22(1), 1990.

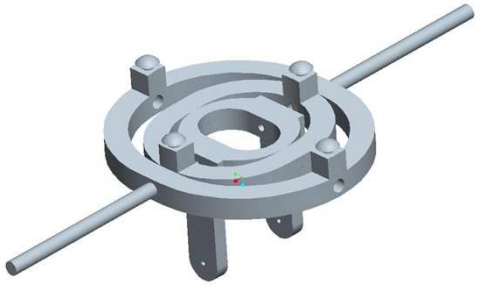
[Yin06] Z.P. Yin, H. Ding, and Y.L. Xiong. Geometric Reasoning on Molding Planning for Multishot Mold Design. Journal of Computing and Information Science in Engineering, 6(3):241-251, September 2006.

[Wils94] R.H. Wilson and J.C. Latombe, Geometric reasoning about mechanical assembly. *Artificial Intelligence*, 71: 371-396, 1994

[Woo91] T.C. Woo and D. Dutta, Automatic disassembly and total ordering in three dimensions. *ASME Journal of Engineering for Industry*, 113(1): 207-213, 1991

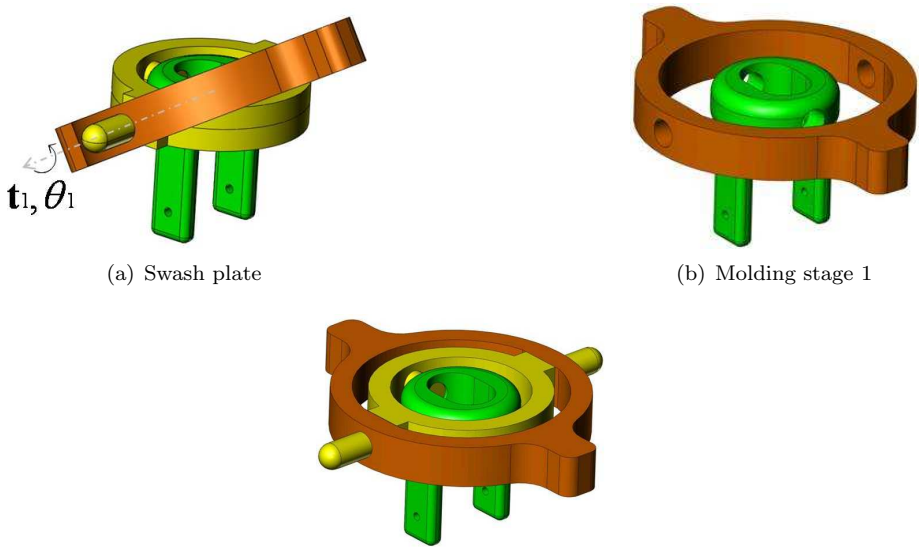


(a) Swash plate with in-mold assembled components



(b) Swash plate with traditionally assembled components

Figure 1: Examples of in-mold assembled articulated devices



(a) Swash plate

(b) Molding stage 1

(c) Molding stage 2

Figure 2: Molding plan example.

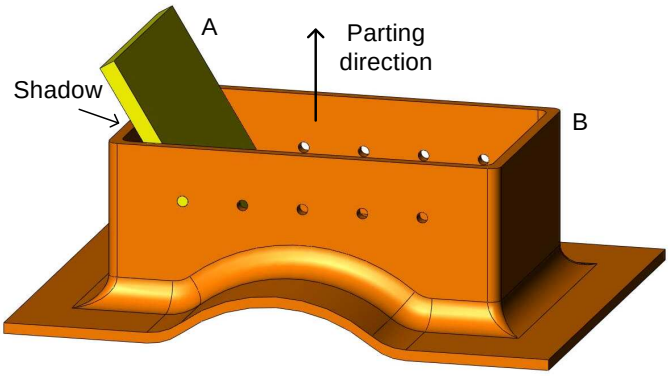


Figure 3: Component A casts shadow on component B

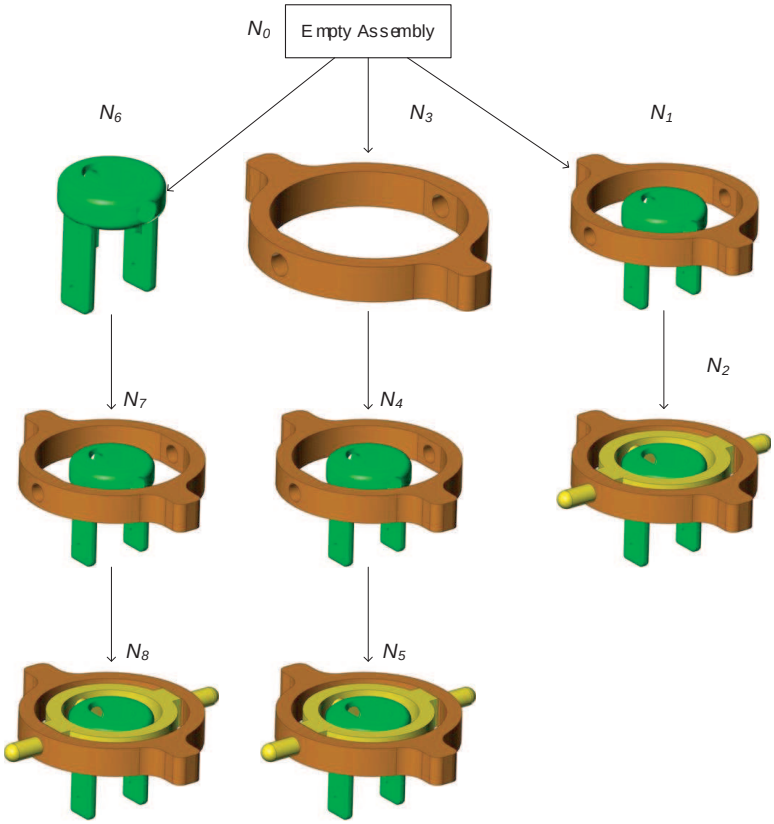


Figure 4: Partial state space for swash plate

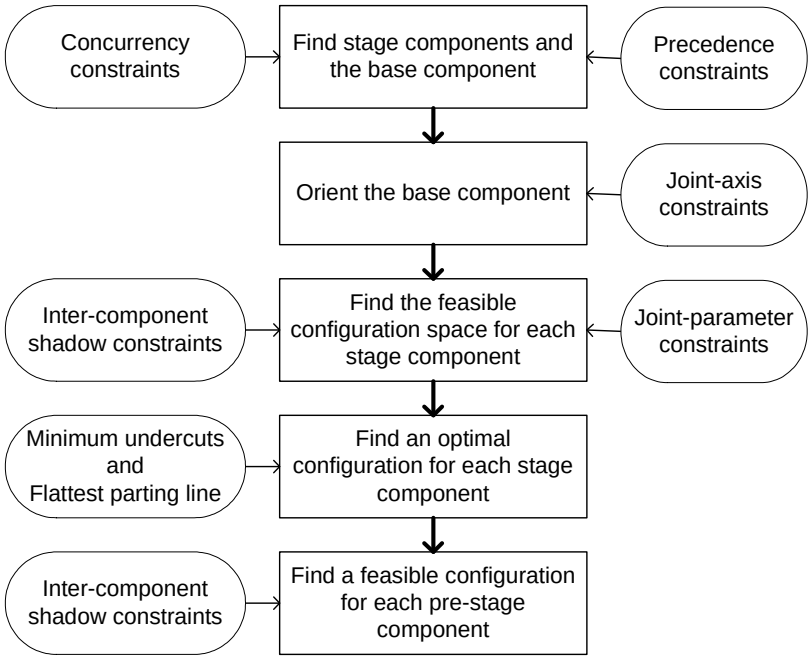
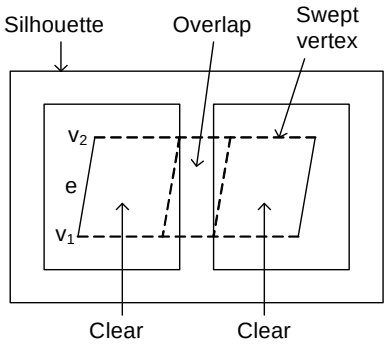
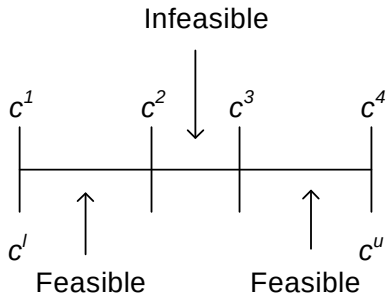


Figure 5: Method to generate a molding stage

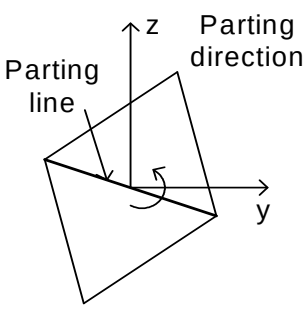


(a) Changing overlap status

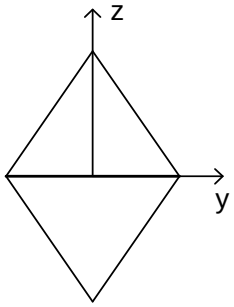


(b) Partition of the initial feasible space

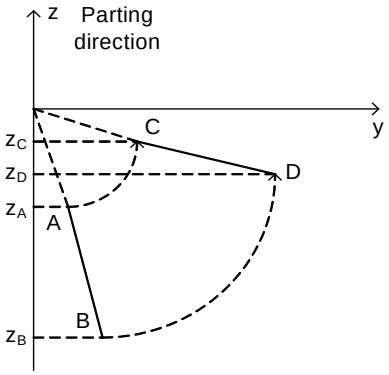
Figure 6: Determining the feasible configuration space for a stage component



(a) Initial orientation



(b) Optimal orientation



(c) Scheme for finding the optimal orientation

Figure 7: Finding a configuration for which the parting line is flattest

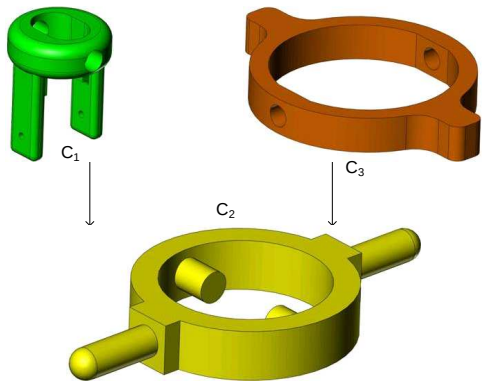


Figure 8: Joint precedence constraints for swash plate

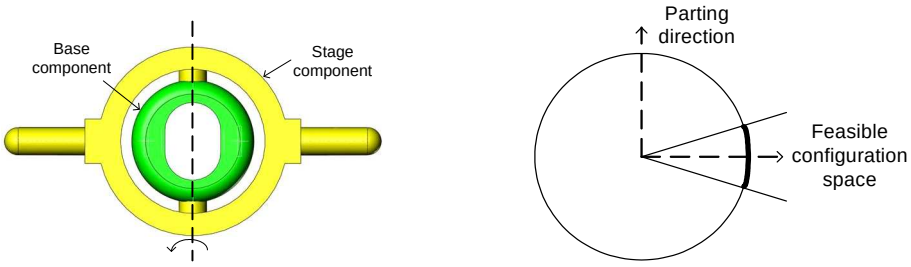


Figure 9: Feasible configuration space for C_2.

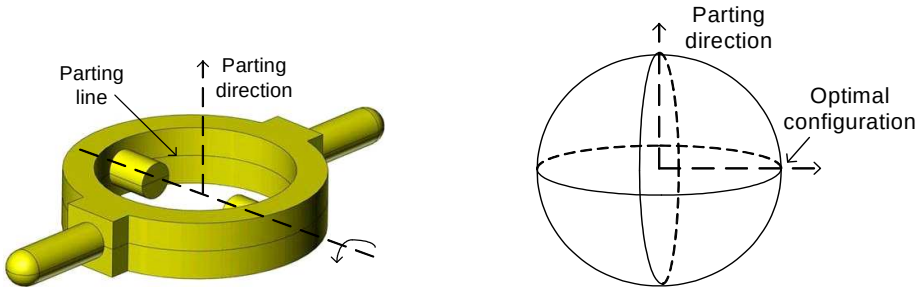


Figure 10: Optimal configuration for C_2.

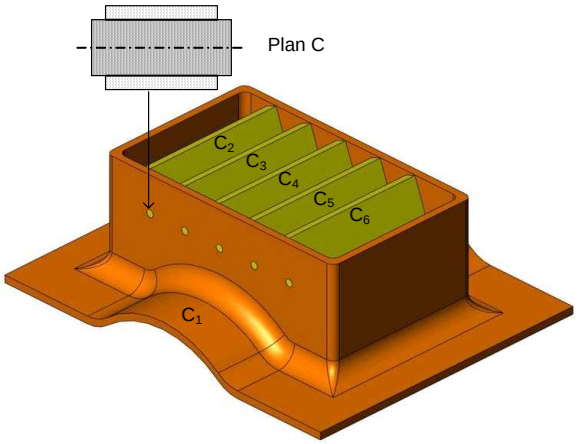


Figure 11: Vent assembly.

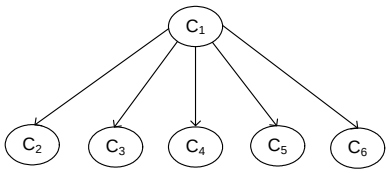
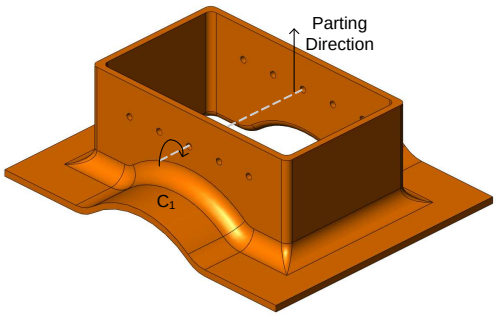
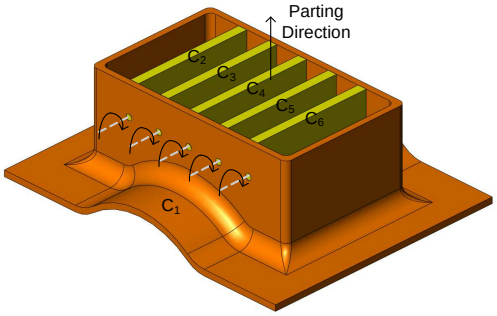


Figure 12: Joint precedence constraints for vent assembly.



(a) First molding stage



(b) Second molding stage

Figure 13: Molding stages for the vent assembly.

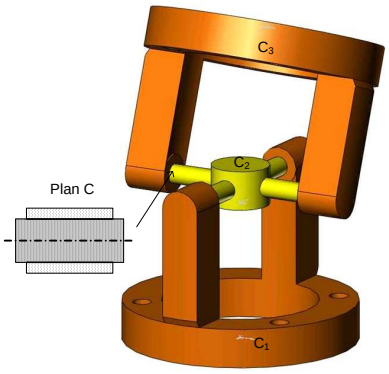


Figure 14: Universal joint.

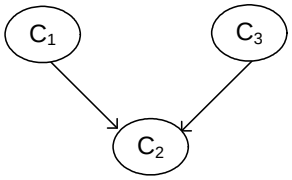
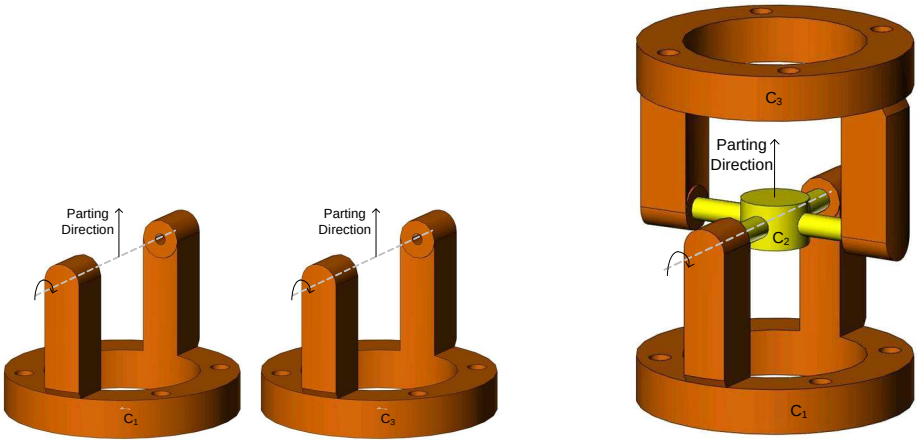


Figure 15: Joint precedence constraints for universal joint.



(a) First molding stage (b) Second molding stage

Figure 16: Molding stages for the universal joint.