ABSTRACT

Title of Dissertation: ALGORITHMS FOR GENERATING MULTI-STAGE MOLDING PLANS FOR ARTICULATED ASSEMBLIES

Alok K. Priyadarshi, Doctor of Philosophy, 2006

Dissertation directed by: Associate Professor Satyandra K. Gupta
Department of Mechanical Engineering

Plastic products such as toys with articulated arms, legs, and heads are traditionally 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. The most common technique of performing in-mold assembly is through multi-stage molding, in which the various components of an assembly are injected in a sequence of molding stages to produce the final assembly. Multi-stage molding produces better-quality articulated products at a lower cost. It however, gives rise to new mold design challenges that are absent from traditional molding. We need to develop a molding plan that determines the mold design parameters and sequence of molding stages. There are currently no software tools available to generate molding plans. It is difficult to perform the planning manually because it involves evaluating large number of combinations and solving complex geometric reasoning problems.

This dissertation investigates the problem of generating multi-stage molding plans for articulated assemblies. The multi-stage molding process is studied and the underlying governing principles and constraints are identified. A hybrid planning framework that combines elements from generative and variant techniques is developed. A molding plan representation is developed to build a library of feasible molding plans for basic joints. These molding plans for individual joints are reused to generate plans for new assemblies. As part of this overall planning framework, we need to solve the following geometric subproblems – finding assembly configuration that is both feasible and optimal, finding mold-piece regions, and constructing an optimal shutoff surface. Algorithms to solve these subproblems are developed and characterized.

This dissertation makes the following contributions. The representation for molding plans provides a common platform for sharing feasible and efficient molding plans for joints. It investigates the multi-stage mold design problem from the planning perspective. The new hybrid planning framework and geometric reasoning algorithms will increase the level of automation and reduce chances of design mistakes. This will in turn reduce the cost and lead-time associated with the deployment of multi-stage molding process.

ALGORITHMS FOR GENERATING MULTI-STAGE MOLDING PLANS FOR ARTICULATED ASSEMBLIES

by

Alok K. Priyadarshi

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2006

DEDICATION

To my parents

ACKNOWLEDGEMENTS

I would like to show my appreciation and express my gratitude to Dr. Satyandra K. Gupta for giving me an opportunity to conduct research under his supervision. I strongly believe that the skills I have acquired while working with him will prove to be extremely useful in my professional career. I would also like to thank my dissertation review committee members for their invaluable suggestions.

This work was supported by NSF Grant DMI-0093142. I am thankful to the University of Maryland, the Mechanical Engineering Department and the Institute for Systems Research for the opportunity and support. Regina Gouker, Florian Krebs, Martin Shroeder, and Stefan Warth helped me in prototyping the example parts. Many of the fundamental and technical ideas underlying this work were initiated from our discussions. Ralph and Mike from Space Ltd. patiently answered all my questions about molding. Cholly Nachman, my colleague at Solidworks taught me finer details of mold design.

My fellow researchers and friends at CIM lab – Abhijit, Arvind, Ashis, Bellam, Brent, Changxin, Chen, Deepak, Greg, Ira, Jun, Li, Malay, Mandar, Mukul, Rohit, Sunil, Tao, and Yao have provided good company and advice. Arindam, Siddharth, Dutta, Akash, and Janhavi hosted me while I was visiting Maryland.

And last but not the least, a special thank you to my wife Susmita and my family for all their support through the years.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

This chapter is arranged in the following manner. Section 1.1 describes the in-mold assembly process that is used to manufacture molded articulated assemblies. Section 1.2 describes the motivation behind the research undertaken in this dissertation. Section 1.3 describes the objectives and associated challenges of this work. Section 1.4 finally describes the outline of this dissertation.

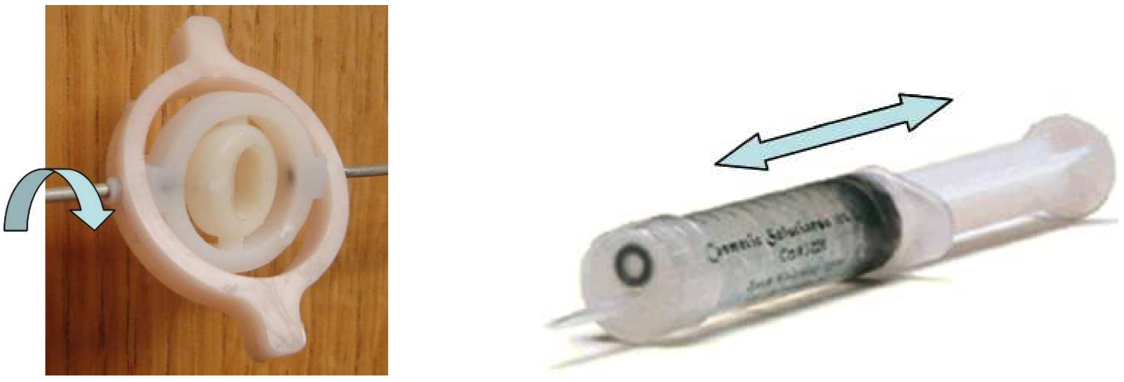
1.1 In-Mold Assembly

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 [Anan95]. Reduction in number of components reduces the associated assembly labor, purchasing, inspecting, warehousing, capital requirements and piece part costs of a product [Roth04]. 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, Bruc04]. For example, it can be used to mold gaskets directly onto parts that need to form tight seals, such as lids, connectors and the like. 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. Two examples are shown in Figure 1.1. Figure 1.2 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 (MSM). MSM is usually accomplished through some 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 into a mold or set of molds.



(a) Swash plate with in-mold assembled components

(b) Syringe with in-mold assembled seal, plunger, and closeable lid

Figure 1.1: Examples of in-mold assembled articulated devices

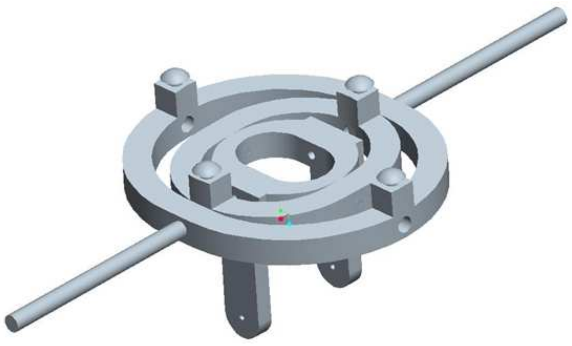


Figure 1.2: Swash plate with traditionally assembled components

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. MSM is described in detail in Section 2.2.

1.2 Research Motivation

Multi-stage molding has emerged as an important manufacturing process. It can be used to make better-quality articulated products at a lower cost. But at the same time, it gives rise to new mold design challenges that are absent from traditional molding. As opposed to traditional molding, multi-stage molding combines two processes – fabrication and assembly. This combination of processes introduces a new component of planning into multi-stage mold design.

In multi-stage molding, an articulated assembly A is produced using a sequence of molding stages $\{s_1, \ldots, s_n\}$. 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. The first stage s_1, starts with an empty assembly ($A_0 = \emptyset$), and the last stage s_n produces the final assembly ($A_n = A$). Intermediate stages require reconfiguring the mold as well as intermediate subassemblies A_i. The first and an integral step in designing a multi-stage mold for an articulated assembly is generating a molding plan, which essentially consists of a sequence of molding stages $\{s_1, \ldots, s_n\}$ required to produce the assembly. Once a molding plan is generated, available software tools for traditional mold design can be used to design a mold for each molding stage s_i.

There are currently no software tools to generate molding plans. The mold design software systems (MoldWizard, ProMold, IMold, etc.) available in market today only handle traditional molding. They provide a variety of tools to speed up the mold design process. They can examine part geometry, simulate analysis, and forecast potential problems. Most of the systems can perform draft analysis, undercut detection, parting line recognition, and core-cavity split. A high-end tool such as MoldFlow can help analyze the flow, cooling, shrinkage, warpage and stress during the thermoplastic injection process. Some others also help in designing ejection and cooling systems. To summarize, the available commercial software packages do not generate a molding plan, but provide low-level tools for analysis and creating mold pieces. A molding plan needs to be manually developed to make use of available tools.

It is difficult to perform the planning manually. Like any manufacturing process planning task, generating molding plan is a challenging problem. The components of an assembly can be molded in any order. But out of all possible permutations and combinations, there are usually very few feasible sequences that lead to a product with desired characteristics. Identifying a feasible sequence that also minimizes the manufacturing cost is even harder. It involves examining a large number of combinations 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.

In the absence of software tools, it usually takes a long time – about three to four weeks on an average to develop a molding plan. There are also concerns about the correctness of a molding plan because many decisions are based on subjective guesswork. In many cases, designers are not able to discover errors until very late in the design process. Discovering problems after investing so much time and money results in expensive product and missed market opportunity. The cost to fix a potential problem is multiplied by several times the further it is discovered down the product development cycle. A problem discovered while designing the part or mold is far more economical to fix than if it is discovered after the molds are built, the parts are molded, and the products are assembled, packaged and delivered to the customer.

The injection molding industry is one of the largest and most competitive industries. To maintain competitive edge, companies need to continuously strive for better, faster, and cheaper products. Shorter design and manufacturing lead times, good dimensional and overall quality, and frequent design improvements are keys to success. Multi-stage molding aims to reduce the manufacturing cost by eliminating secondary assembly operations. But those cost benefits cannot be realized unless we produce multi-stage molds in much less time and cost. It is observed that software lowers the overall cost by reducing design/manufacturing lead times. It reduces chances of error through robust geometric calculations and also explores design alternatives that are otherwise difficult due to human constraints. Hence it would be useful to have software tools to generate multi-stage molding plan.

1.3 Research Objectives and Issues

The purpose of this dissertation is to investigate the problem of generating multi-stage molding plan for articulated assemblies. This dissertation formally defines the planning problem and develops algorithms to solve them. These algorithms can be used to develop software that can automatically generate multi-stage molding for articulated assemblies.

This dissertation makes the following assumptions:

- *The input assembly is a serial mechanism.* This dissertation does not handle parallel mechanisms.

- *The input assembly consists of only rigid-body joints.* Articulation can be achieved by both – compliant joints [Gouk06, Bruc04] and rigid-body joints. The compliant joints are created using a soft (compliant) material between two rigid materials. The material interfaces are adhered to each other completely constraining the motion between them. In contrast, there is some clearance and hence one or more degrees of freedom between material interfaces in rigid-body joints. This dissertation only deals with rigid body joints.

- *Sequencing is not affected by flow considerations.* It is assumed that each component is feasible to mold from the mold-flow point of view in all possible sequences.

- *Sequencing is not affected by thermal considerations.* It is assumed that the thermal management system is capable of providing appropriate cooling and heating.

We need to develop a framework for solving the planning problem. Performing assembly with fabrication leads to new problems that are absent from the traditional mold design problem. These new problems are geometric in nature and need to be solved in order to do planning. So we need to develop geometric reasoning algorithms for the same. The rest of this section discusses the associated research issues.

1.3.1 Planning Framework

Generating molding plan is in some ways similar to other manufacturing operation-planning problems such as machining and sheet-metal bending. There are two traditional approaches to process planning. The first approach is called *generative* process planning. In this approach a plan is synthesized from the first principles by trying various alternatives in generate-and-test paradigm. The second approach is called *variant* process planning. In this approach, a plan is generated by modifying an existing plan. Sometimes a hybrid approach is also used that combines elements from generative and variant techniques. We need to identify the approach that is most suitable for our planning problem.

Incorporating Experimentally-Verified Molding Plans for Joints

When a successful plan is developed and experimentally validated for molding a joint, a lot of useful knowledge is generated. Developing a successful molding plan is a complex and time-consuming process. Hence, when a joint similar to a previously molded joint is found in a new assembly, it would be more efficient to reuse the previously generated plan than reinventing it from scratch. The various types of joints used are very few. So it is very common to find similar joints in new assemblies. We need to develop an approach for incorporating the experimentally-verified molding plans for individual joints into the overall planning framework for assemblies.

Formulation of Molding Planning Problem

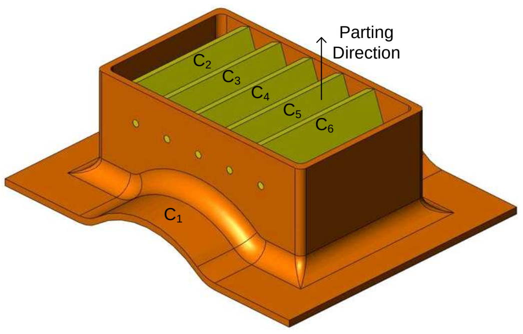
The molding planning problem has not yet been formally defined. We first need to develop mathematical definitions to represent entities such as articulated assemblies, configuration space, and molding stage. A molding plan is considered feasible if it leads to a product with desired characteristics. We need to study the multi-stage molding process and identify the underlying governing principles and constraints that a plan needs to satisfy in order to be feasible. Multiple feasible molding plans may exist for a given assembly. In that case, it is desired that we select the molding plan with minimum manufacturing cost. The cost of a molding plan depends on a variety of factors. We need to identify all those factors and develop a computationally-tractable cost function that needs to be minimized.

Development of Search Technique

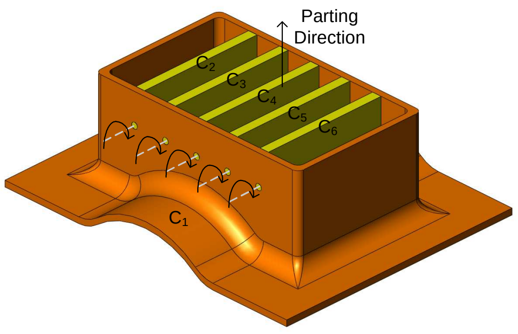
Manufacturing operation-planning problems that are usually formulated as state-space search and uses branch and bound (B&B) search to obtain an optimal solution. However the effectiveness of B&B depends on problem-specific heuristics that guides the search to an optimal solution quickly. Any problem employing B&B needs to provide a bounding cost function. In our case we need to calculate a lower bound cost for molding a subassembly. Another problem that we need to consider is reducing the time taken to generate a search node. Generating a search node involves extensive geometric reasoning and making queries to a geometric kernel. Such computation is time-consuming and leads to very high node-generation time. Slow node generation also makes it difficult to explore large portions of the search space. We need to develop an approach to avoid generating redundant nodes.

1.3.2 Geometric Algorithms to Support Planning

In each molding stage, we need to find a configuration of the intermediate subassembly A_i in which molding will take place. The configuration should be such that it is feasible as well as optimal. In a feasible configuration, there is no obstruction between two components along the parting direction. The characteristics of an optimal configuration is that the cost of the molding stage is minimum. This dissertation defines the cost of a molding stage as sum of molding cost, defect cost, and tooling cost. The number of undercuts on the components to be molded (stage components) is a contributing factor to the molding cost. Similarly, complexity of the parting line and machining cost of the shutoff surface are the contributing



(a) Infeasible configuration



(b) Feasible configuration

Figure 1.3: Examples of infeasible and feasible configuration

factors to the defect cost and the tooling cost respectively.

Hence for each molding stage, we need to find a configuration that is obstruction-free, and for which the number of undercuts on the stage components is minimum and the parting line is flattest. For each molding stage, we also need to create a shutoff surface for which the machining cost is minimum.

Finding Obstruction-Free Configuration

In a valid molding stage s_i, the components of subassembly A_i do not obstruct the accessibility of each other. Figure 1.3 shows examples of infeasible and feasible configuration for a vent assembly. The configuration shown in Figure 1.3a is infeasible because components $C_2 - C_6$ obstruct each other along the parting direction. Orienting the components vertical as shown in Figure 1.3b makes the configuration obstruction-free and hence feasible.

The algorithm for finding an obstruction-free configuration space for A_i requires determining global ray-accessibility of facets. The algorithm for finding the mold-piece regions also needs to determine the accessibility of facets. It is required that the accessibility analysis algorithm be both *efficient* and *robust.* In computational geometry, efficiency and robustness are usually conflicting attributes for an algorithm. It should be efficient because it will be very heavily used. It should be robust because we are dealing with polyhedral objects. The curved surfaces on the part boundary are faceted and approximated by smaller triangles. Due to the surface tolerances introduced by faceting, a robust method is required to determine the accessibility of near-vertical facets.

The available *visibility* algorithms cannot be used because there is a small, but important difference between ray-accessibility and visibility. A facet perpendicular to a direction $\vec{d}$ (vertical facet) is not visible but accessible in $\vec{d}$. Mathematically, suppose $\vec{n}$ is the facet normal, then the facet is visible if $\vec{d}.\vec{n} > 0$, but accessible even if $\vec{d}.\vec{n} = 0$. Therefore, a new approach to determine global ray-accessibility of a facet needs to be developed.

Finding Assembly Configuration with Minimum Undercuts

The tooling cost for a component is directly proportional to the number of side actions that are required to form the undercuts on the component. Here we are interested in finding a molding configuration for a component for which the number of undercuts is minimum. Finding a parting direction and finding a configuration are equivalent problems. There are primarily two approaches used for finding parting direction: approaches based on accessibility analysis and approaches based on feature recognition. Unfortunately none of them provide complete solution for the problem being considered in this dissertation. Hence we need to find a new approach to find a configuration with minimum undercuts.

Finding Assembly Configuration with Flattest Parting Line

Mathematically, parting line of a part is equivalent the silhouette of the part. However, the silhouette is not always a good parting line. 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. 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 it is proposed that the flattest possible parting line be found [Ravi90, Majh99, Chen03]. The available approaches to finding the flattest parting line is limited to simple convex parts. Here we need to solve a related, but different problem. We need to find a configuration for which the parting line is flattest.

Creating Shutoff Surface with Minimum Machining cost

One of the most popular approaches to create shutoff surface is by extending parting lines toward side-walls of the mold enclosure. This approach works fine on simple planar parting lines. But simply extending complex non-planar parting lines may produce shutoff surface that intersects with the part or with itself. It also may not follow the standard techniques used by the mold designers to reduce machining cost and flash. Therefore, we need to develop a new approach to create shutoff surface.

1.4 Dissertation Outline

The ensuing chapters of this dissertation discuss how the above-described objectives and challenges are addressed by this dissertation.

Chapter 2 provides the technical background required to understand this dissertation. Chapter 3 formally defines the problem being investigated in this dissertation. Chapter 4 presents a survey of related work. Chapter 5 presents a framework for representing reusable molding plans for articulated joints. This chapter also presents molding plans for three basic joints – prismatic, revolute, and spherical. Chapter 6 is the main chapter of this dissertation. It describes an algorithm for generating multi-stage molding plan for articulated assemblies. Chapter 7 describes an algorithm to robustly and efficiently find the mold-piece regions for a given object. Chapter 8 presents an algorithm to create a provably correct and optimal shutoff surface for a given parting line. Chapter 9 summarizes the conclusions reached from this research and provides suggestions for future extensions.

Chapter 2

BACKGROUND

This chapter provides the background required to understand the material presented in this dissertation. Section 2.1 describes a brief background on the injection molding process. It also describes the basic components of a mold system. This dissertation presents an algorithm for generating molding plan for articulated assemblies created using multi-stage molding. Section 2.2 describes the multi-sage molding technique. Section 2.3 provides a brief review of a technique used for rendering shadows using computer graphics hardware. The algorithm for finding mold-piece regions described in Chapter 7 is based on this hardware shadow mapping technique. Some basic geometric modeling concepts are described in Section 2.4.

Generating molding plan is similar to other manufacturing operation-planning problems such as machining and sheet-metal bending. Many operation-planning problems are formulated as combinatorial optimization problems. They are generally solved using state-space search. Section 2.5 briefly reviews the state-space search algorithms.

2.1 Injection Molding

Injection molding is one of the most common plastic manufacturing process used today. Products produced with this process permeate virtually every aspect of our lives. From the coffee maker and toothbrush we use in the morning, to the car we drive to work, to the computer and telephone we use during the day, so many products we use are injection molded. Injection molding is used in almost every market and represents the mainstay of the designer's toolbox of processes. One of the main reasons for its dominance is its versatility. The forms that designers can create are almost unlimited. The wide range of plastic materials we can choose to mold is so broad, it can address most of our needs. Typical examples of products made by injection molding are appliance casings (for example, computer monitors, CPUs), aircraft and automotive parts, and utensils to name a few.

Injection molding is a near-net-shape manufacturing process that can produce parts with no or very few secondary manufacturing processes. The parts produced have good surface quality and accuracy. It is particularly well suited to high volume production using economies of scale and short cycle times to drive costs down. While much of the consumer industry involves the design and fabrication of injection-molded thermoplastic parts, metals and ceramic parts can also be produced.

2.1.1 Molding Process

In the injection molding process, we take raw plastic material in the form of small pellets (also referred to as resin), heat it gently to the point where it will flow under moderate pressure, and inject it (push it with a plunger) into a mold. The mold is usually made up of two separable halves. After allowing enough time for the

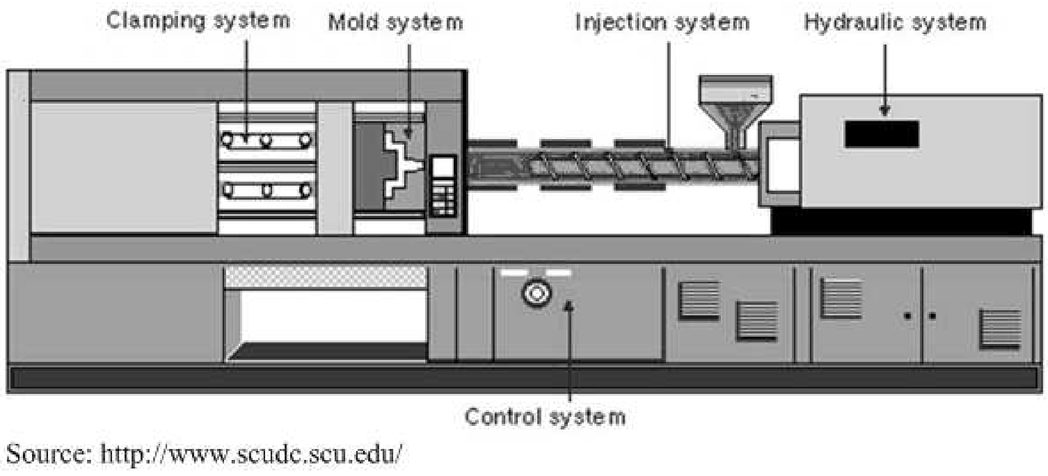


Figure 2.1: A single screw injection molding machine for thermoplastics

plastic to cool off and solidify, the mold opens (separates), and the molded part is removed. The process utilizes specialized equipment called injection-molding machines. These machines can be quite large, usually much larger than one would expect relative to the sizes of the parts they make. Figure 2.1 shows a typical injection-molding machine for thermoplastics.

Injection System is responsible for melting and injecting molding material. It confines and transports the material as it progresses through the feeding, compressing, degassing, melting and injection stages. It contains a reciprocating screw, which while turning, compresses, melts and feeds the material being molded.

Mold System is an important and costly part of the injection-molding machine. It contains the cavity into which molten material is injected. It also contains cooling channels that regulate temperature on the mold surface. Section 2.1.2 describes the mold system in detail.

Clamping System supports and carries the constituent parts of the mold system. It is responsible for opening and closing of mold. When the molten material is being injected, it provides sufficient force to prevent the mold from opening. The size range of machines is usually stated in tons, which refers to the clamping pressure that they can apply to the mold halves. Typical machine capacities range from 40 to 2000 tons, suitable for plastic parts up to 80oz.

Hydraulic System provides power to the mold system (for energizing ejector pins and slides), the clamping system (for opening and closing the mold), and the injection system (for turning and driving reciprocating screw).

Control System is like the CPU of the whole system. It monitors and controls the processing parameters (temperature, pressure, injection speed, screw speed and position, and hydraulic position) and hence provides consistency and repeatability in machine operation.

2.1.2 Mold System

An injection mold consists of two main pieces – *core* and *cavity*. The core and cavity form the impression into which molten material is injected. The cavity determines the external shape of a molded part and the core forms the internal shape. *Parting Direction* is the direction along which the core and cavity are separated. *Undercuts* are “sideways” recesses or projections on the molded part that prevent the removal of the molded part from the mold along a parting direction. The undercuts are formed using *side actions*. When the mold opens, the side actions are moved out of the way, thereby allowing separation of the core and cavity. The motion of the side action usually consists of one or two translations away form the undercut. The core side of the mold consists of an *Ejection System*, which is responsible for ejecting the molded part after the mold opens. The most common type of ejection system consists of ejector pins. The stroke on these ejector pins clear the molded

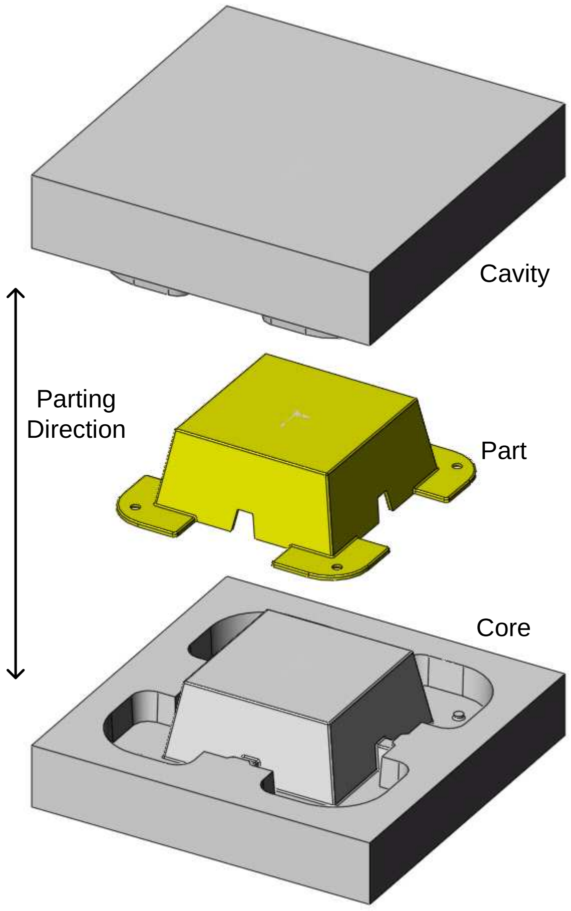


Figure 2.2: A basic mold for an example part

part out of the mold system. Figure 2.2 shows a basic mold for an example part. Figure 2.3 shows how the molded part is ejected from the mold.

The mold system consists of some additional features. *Sprue* provides the entry point of molten material into the mold. Sprue bush provides the seating for the injection-cylinder nozzle and conducts the hot molding material from the injection cylinder to the mold cavity. *Leader Pins* assist in assembly and fitting of the mold pieces. They maintain alignment during mold setup and when the mold is running. The mold system may also consist of *Cooling Channels*, that are

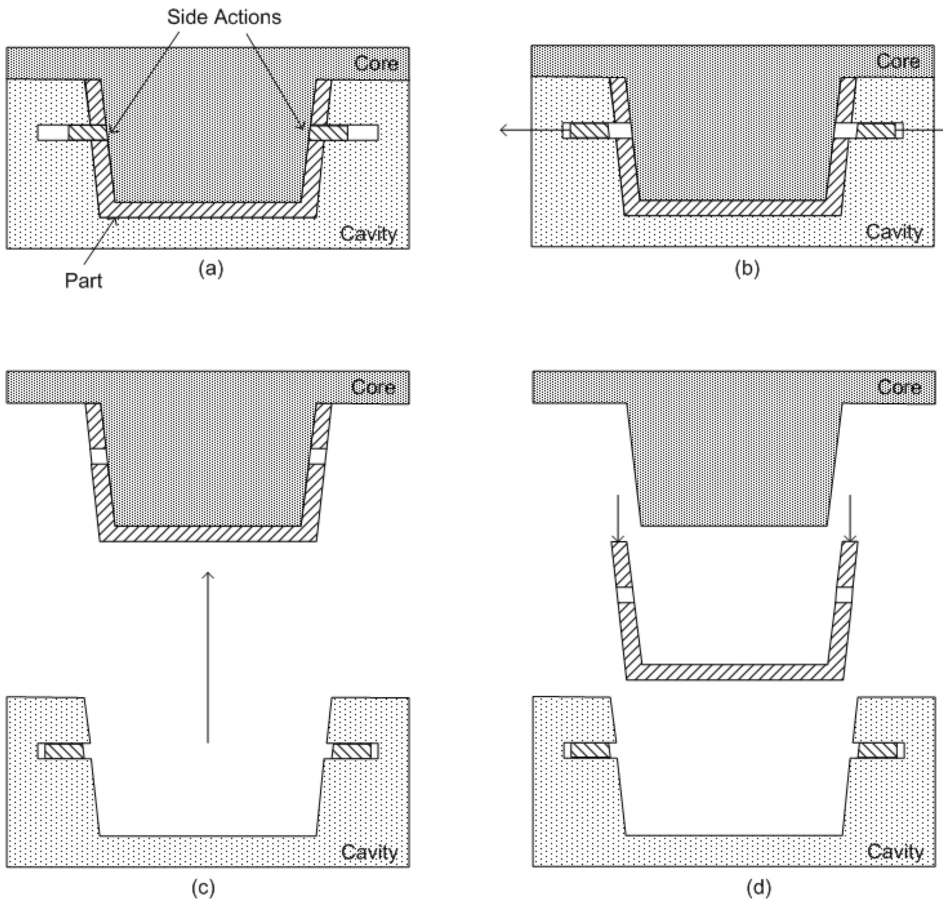


Figure 2.3: Mold System operation; (a) molten material has been injected; (b) after the material solidifies, the side actions retract; (c) the core separates taking the part along; (d) the molded part is pushed out of the core

passageways located within the body of a mold, through which a cooling medium (typically water, steam, or oil) circulates. It helps to regulate temperature on the mold surface.

2.1.3 Rapid Tooling

As a product develops, it requires tooling with different production capacity at each development stage. Pre-production prototyping is a common industrial practice. The purpose of prototyping is to help the designers visualize the object that is being designed and hence eliminate any design errors. With a prototype it can be actually seen in real life whether or not the two surfaces of a widget interfere, or if the screw threads on a fastener are of the wrong size. The majority of product development cost occurs in the concept and design validation phases. During those times, much time is spent designing and re-designing the product. Prototypes are made and evaluated. Then changes are made and the whole process starts over. Since this is an iterative process, prototyping needs to be quick and economical.

Rapid tooling is a molding technique used for producing limited number of prototypes. It is sometimes also used for low-volume pilot production and market testing. In the prototype production phase, it is required that molds be manufactured quickly and economically. Therefore, layered fabrication techniques (SLA, SLS, 3D printing) or high-speed milling of tooling blocks are used to manufacture epoxy or aluminum molds. The molds also do not contain expensive components such as actuated side actions and cooling channels.

2.1.4 Production Tooling

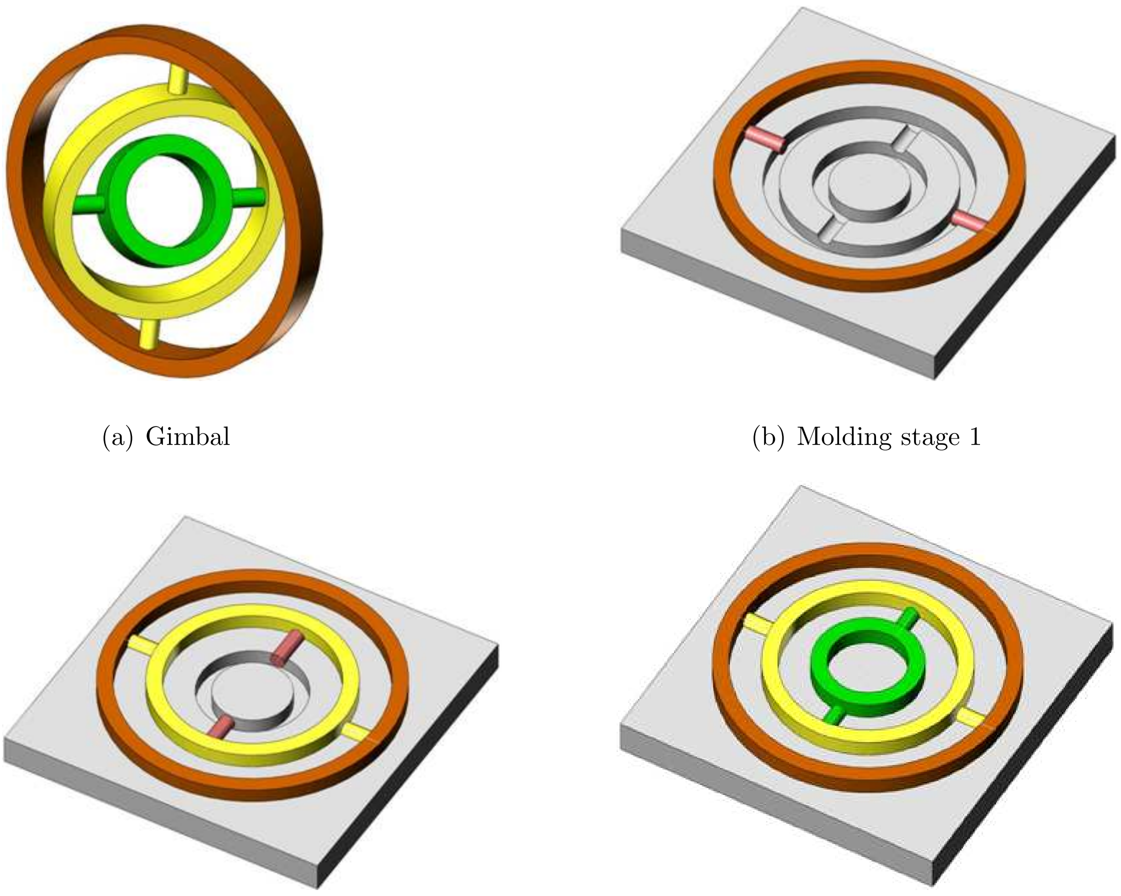
Production tooling is capable of high volume production. It is employed only after the product design is finalized and the product has stabilized in the market. Here the focus is on the durability of the mold and reducing the molding cycle time. Hence the production tooling are usually made by milling alloy steel blocks and measures are taken to squeeze every second out of the cycle time. The side actions are fully automated and there are extensive cooling channels to quickly solidify the injected material.

2.2 Multi-Stage Molding

Traditionally injection molding involves a single material and consists of only one molding stage. Multi-stage molding refers to a molding process in which multiple materials are added in a sequence to produce multi-material objects. For each component in the desired product, usually a separate molding stage is required. A molding stage entails injecting a single material in a specific mold configuration. Successive stages require reconfiguring the mold as well as the previously injected components.

Figure 2.4a shows an example of a three-material articulated gimbal that can be manufactured using multi-stage molding technique. The gimbal consists of three rings mounted on axes at right angles. It can be molded in three molding stages. The outer ring is molded in the first mold stage as shown in Figure 2.4b. Two inserts are used as placeholders for the middle ring. In the second mold stage, the middle ring is molded and the same inserts are used as placeholders for the inner ring as shown in Figure 2.4c. The inner ring is molded in the final mold stage after removing the inserts as shown in Figure 2.4d. The figure only shows the cavity mold piece for clarity. The core mold piece is similar to the cavity piece.

An overview of different multi-material molding techniques can be found in [Gouk06]. Because multi-stage molding techniques can be significantly different than traditional single-material molding, some new terminology has been adopted to better explain these techniques and processes. Because most molding involves injecting or shooting the polymer into the mold cavity, the word ‘shot’ is sometimes



(a) Gimbal

(b) Molding stage 1

(c) Molding stage 2

(d) Molding stage 3

Figure 2.4: Multi-stage molding.

used instead of 'stage'. Two more terms that arise frequently in the context of multi-stage molding are 'substrate' and 'overmold'. The substrate is the material that is injected in the first stage, usually forming the base or majority of the final component. The overmold is the subsequent shot which tends to form at least partially over top of the substrate.

2.3 Hardware Shadow Mapping

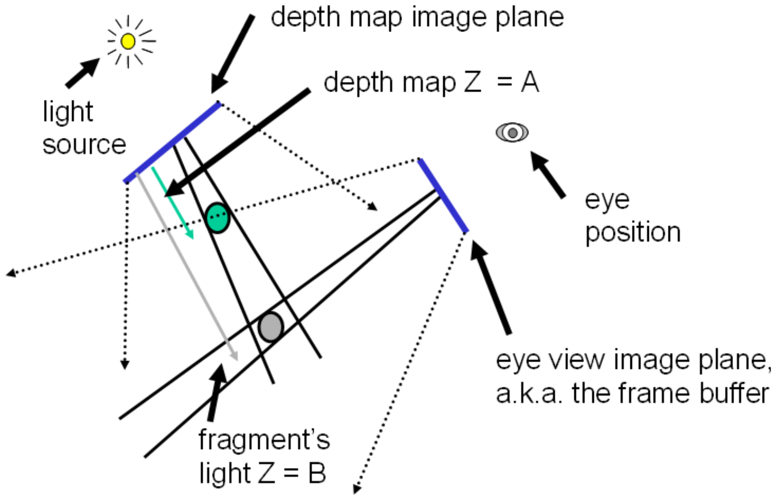
Hardware shadow mapping [Will78] is a hardware-accelerated image-based shadowing technique. It utilizes existing hardware functionality, texturing and depth buffering, to efficiently calculate complex high-quality shadows for a lighted 3D scene. A comprehensive explanation on hardware shadow mapping can be found in [Ever01]. We will briefly describe the technique here for the sake of completeness.

Consider a scene consisting of a couple of objects and a single point light. When rendering the scene, for each point (rasterized fragment), we need to find whether it is lit, or in shadow. Clearly, it is lit only if the straight path from the light to the point is not occluded by any other object present in the scene. This is the basic idea behind shadow mapping. The lit regions in the scene are exactly those that are visible to a viewer placed at the light source. Other regions are in shadow. The lit and shadowed regions are calculated by performing a visibility test for each light source at each rasterized fragment using the depth buffer technique. Following are the typical steps to do shadow mapping using graphics hardware:

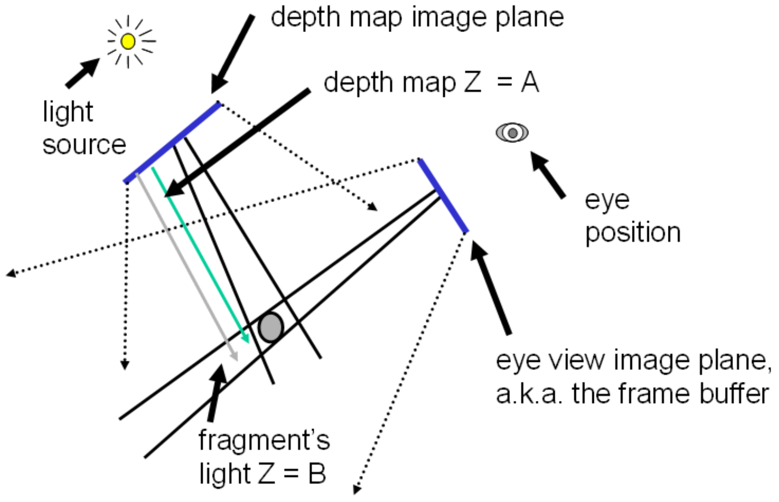
1. Render the scene from the light's point of view and save the depth buffer values into a shadow map (depth texture)

2. Render the scene from the camera's point of view. To determine whether a point is shadowed or not, compare the distance B between the point and the light with the corresponding depth A stored in the shadow map.

    - if $A < B$ (Figure 2.5a), there must have been an object in front of this point when looking from the light's position, so this point is in shadow.
    - if $A = B$ (Figure 2.5b), there was nothing occluding this point when drawing from the light source, so this point is lit.



(a) The $A < B$ shadowed fragment case



(b) The $A = B$ unshadowed fragment case

Figure 2.5: Depth comparison scheme used in shadow mapping [Ever01]

2.3.1 Self Shadowing

Shadow mapping is prone to self-shadowing artifact in which the equality test for unshadowed fragments yields incorrect results. This happens due to following two reasons:

1. *Lack of precision.* The depth values in the shadow map are stored as finite-precision floating point numbers. Using an equality to test for an unshadowed point may produce incorrect results due to the lack of precision. This is the same reason as that behind comparing two finite-precision floating point numbers.

2. *Lack of resolution and variable sampling location.* Due to fixed resolution of the shadow map and depth buffer, when the geometry is rasterized from the eye's point of view, it will be sampled in different locations than when it was rasterized from the light's point of view. It is also unlikely that a fragment will be exactly mapped onto a texel (texture element) in the shadow map. In that case, any interpolation of depths may produce incorrect results.

The most widely used solution to the self-shadowing artifact is called polygon offset, where a small bias is added to the depths stored in the shadow map. Due to variable sampling location, the amount of necessary bias depends on the slope of the rasterized polygon in light space. The reason for variable bias is explained in [Ever01].

Another approach to preventing self-shadowing artifact called second-depth shadow mapping is presented by Wang and Molner [Wang94] for the special case of a scene consisting of solid objects only. Their method eliminates the need for a bias by rendering only the back facets into the shadow map. This method is based on the observation that in case of solid objects there is always a back facet on top of a shadowed front facet. This generally works better than the polygon offset method because there is adequate separation between the front and back facets, but of course is limited to solid objects.

2.3.2 Percentage Closer Filtering (PCF)

Shadow mapping algorithm suffers from aliasing problems like any other sampling method. Usually, texture maps are accessed by filtering the texture values over some region of the texture map. Accessing depth values from shadow maps in similar manner is inappropriate. The basic problem is that filtering depth values bears no relation to the geometry of the scene and leads to undersampling artifacts. This problem is illustrated in Figure 2.6b. In the figure, we need to determine if a pixel at depth 0.55 is shadowed. The pixel is mapped to four pixels in the shadow map that are at depths 0.25 and 0.63. Filtering (averaging) these depth values we get 0.44. Hence, the pixel is marked as shadowed since it's depth (0.55) is greater than the filtered depth (0.44). This is quite wrong because there is nothing at 0.44, but at 0.25 and 0.63.

Reeves [Reev87] proposed a filtering technique called *Percentage Closer Filtering* (PCF) to produce anti-aliased shadows. PCF works by reversing the order of filtering and comparing. The given z value of the surface is first compared with the shadow map depths over a region. This comparison converts the shadow map under the region into a binary image, which is then filtered to give the proportion of the region in shadow. The resulting shadows have soft, antialiased edges. PCF technique is illustrated in Figure 2.6c. Here the pixel is reasonably marked as 50% shadowed.

2.4 Geometric Modeling Basics

A geometric model is a mathematical description of the shape of a physical object. Geometric modeling is the study of construction or processing of geometric models.

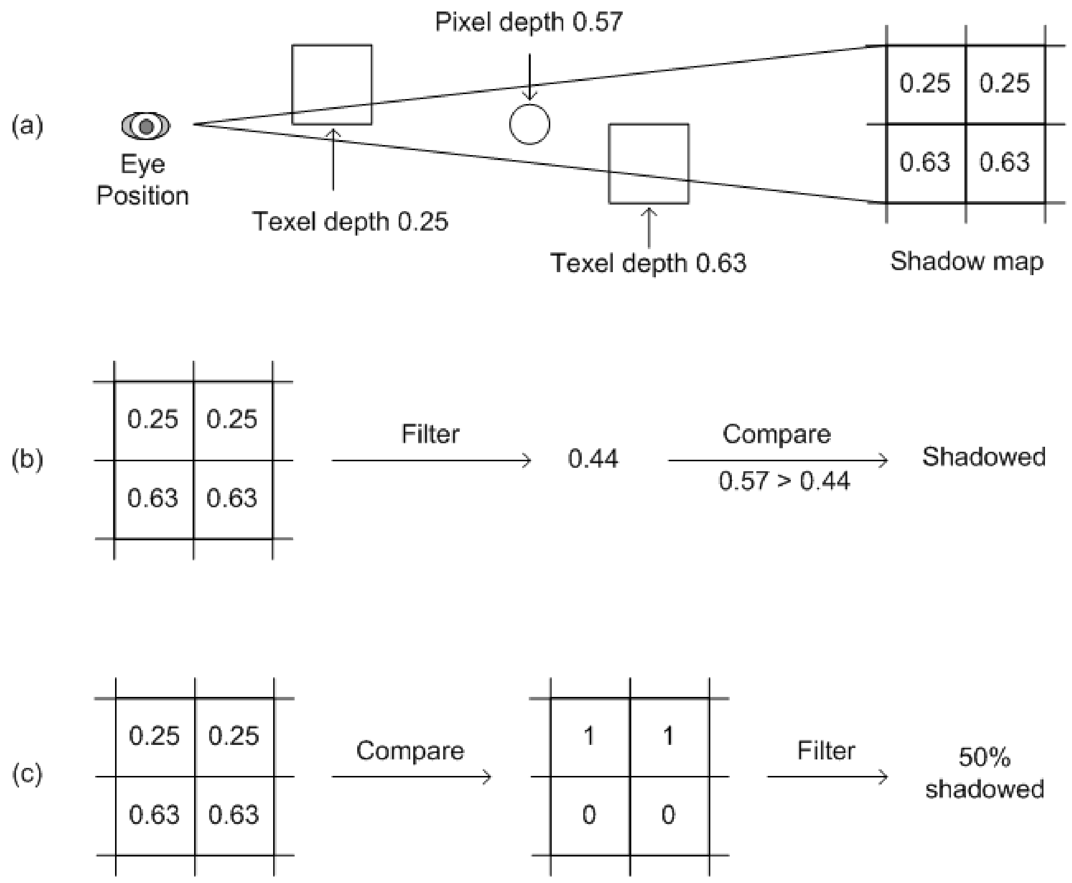


Figure 2.6: These figures illustrate percentage closer filtering technique; (a) Determining whether the pixel is in shadow; (b) Ordinary texture map filtering that does not work for shadow maps; (c) Percentage closer filtering

Geometric models are extensively used in computer-aided design and manufacturing and computer graphics.

There are mainly three types of models used to describe a geometry – wireframe, surface, solid. Wireframe models represent a shape by its characteristic lines and end points. Wireframe modeling systems were popular when geometric modeling was first introduced. In surface models, the mathematical description corresponding to a geometry includes surface information in addition to the information about the characteristic lines and their end points contained in the wireframe description. The mathematical description may include the information about surface connectivity (i.e., information on how surfaces are joined and which surfaces are adjacent to each other at which curves, and so on). Surface modeling systems are popular systems in sheet metal industry. Currently, Solid models are most widely used representation because it provides a richer set of information than wireframe and surface models.

2.4.1 Solid Models

Solid models are used to model a shape having a closed volume, called a solid. Unlike wireframe modeling systems or surface modeling systems, a simple set of surfaces or a simple set of characteristic lines is not allowed if it cannot form a closed volume. In addition to the information provided in a surface modeling system, the mathematical description of a shape created by a solid modeling system contains information that determines whether any location is inside, outside, or on the closed volume. Therefore any information related to the volume of the solid can be derived, and thus application programs can be written to do operations at the level of volume instead of at the level of surface. For example, an application program can be written to generate automatically the finite elements of a solid type from a solid model. Furthermore, an NC tool path generation program can be written to generate automatically all the tool paths to machine the volume to be removed from the workpiece. It can do so without generating the tool paths surface by surface that would require user input for each surface. These capabilities are realized when the model is created as a complete solid.

Developers of solid modeling systems try to provide simple and natural modeling functions so that users can manipulate the shape of a solid as they do for a physical model without having to consider the details of the mathematical description. Modeling functions such as primitive creation, Boolean operations, lifting, sweeping, swinging, and rounding typically require only a simple input from the user. They then take care of all the bookkeeping tasks needed to update the mathematical description. The modeling functions supported by most solid modeling systems can generally be classified into many groups. The first group includes the modeling functions that are used to create a simple shape by retrieving a solid, which is one of the primitive solids stored in the program in advance and by adjusting its size. Hence they are called primitive creation functions. The next group includes functions of adding to or subtracting from a solid. These functions are called Boolean operations. Another way to create a solid is by moving a surface. Thus the sweeping and skinning functions belong to this group. The sweeping function creates a solid by translating or revolving a predefined planar closed domain. The modeling function using the revolution of a planar domain is also called sweeping. Another possible approach is called parametric modeling; because various solids are generated by changing the parameters. The parameters may be some constants involved in the geometric constraints and/or dimension values. The skinning function generates a solid by creating the skin surface to enclose a volume when the cross sections of the desired solid are given. The rounding and blending functions create the solid by performing local modifications to the solid.

2.4.2 Assembly Models

Geometric modeling systems, whether they are wireframe, surface, or solid modeling systems, have been used mainly to design or model an individual part rather than for the assembly of parts. Until recently, engineers designed parts individually and then assembled them later in the development cycle to determine whether they fit properly and the product functioned as intended. Such an approach was fine for small design teams working on simple products. However. this approach is unworkable when the design is performed by several teams spread around the world and the assembly to be designed is complex. A designer may change a component configuration and fail to let others know or forget to make corresponding changes in other affected components. Considerable time is spent manually tracking part designs, part-to-part interfaces, engineering changes, product specifications, test results, and other essential information to be sure that individual part designs fit with one another. In the early 1990s, the growing need for collaborative engineering in industries was a primary driving force for the development of assembly design capabilities. These capabilities accurately keep track of parts and their relationships to one another so that designers can create part geometry in the context of other parts. Probably the greatest use of assembly design capabilities is in the automotive and aerospace industries. There the design of highly complex products must be coordinated not only for engineers throughout the world but also with second- and third-tier suppliers.

Assembly modelers provide a logical structure for grouping and organizing parts into assemblies and subassemblies. The structure enables a designer to identify individual parts, keep track of associated part data, and maintain relationships among parts and subassemblies. Relationship data maintained by an assembly modeling system include a wide range of information about a part and its association with others in the assembly. Mating conditions between parts in the assembly are among the most important pieces of relationship data. Mating conditions identify how the part is connected to others (e.g., two planar faces of a pair of parts are in contact or two cylindrical faces are coaxial). Instancing information identifies other places in the assembly where the same part is used; instancing is a useful concept, especially for standard parts, such as fasteners, because the part data can be stored only once even when the part is used many places in the assembly. Data on fit, position, and orientation specify exactly how parts are joined in the assembly and often include allowable tolerances. The position and orientation data of parts are derived from the mating conditions in many systems. Assembly modeling systems also provide the capability to create parametric constraint relationships between parts and to measure size and dimension information from one part and apply it to another, thus freeing the user from having to reenter geometric data where parts interface. Inter-part constraint relationships are helpful when many dimensions in an assembly depend on some key dimensions. Once such relationships have been input, the designer needs change only the key dimensions; the system takes care of other related dimensions automatically. This powerful capability also provides a mechanism for propagating a complete change (e.g., if the diameter of a shaft changes, the size of a hole that fits into it is updated as well). Thus designers' time is saved because the entire assembly doesn't have to be painstakingly modified whenever part designs are modified.

2.4.3 Geometric Transformations

Geometric transformations are quite often used during construction of geometric objects and for performing gometric reasoning. A geometric object can be considered as a set of points in $\mathbf{E}^2$ or $\mathbf{E}^3$. A geometric object is constructed and manipulated in a coordinate system. For any point p in space, a transformation maps it into a new point q. Theoretically, we can transform a geometric object by transforming each of its points.

Vector Representation of a Point

A point in $\mathbf{E}^3$ can be represented by its position vector as follows:

$$\vec{p} = x\hat{\imath} + y\hat{\jmath} + z\hat{k} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

where $\hat{\imath}$, $\hat{\jmath}$, and $\hat{k}$ represent unit vectors along three axis of current coordination respectively. x, y, z are the coordinates of point p.

Linear Transformation

A transformation maps a point in space into another point. A possible mapping can be accomplished using the following formula:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$x' = a_{11}x + a_{12}y + a_{13}z$$

$$y' = a_{21}x + a_{22}y + a_{23}z$$

$$z' = a_{31}x + a_{32}y + a_{33}z$$

Above equations are linear in nature. Therefore, we call this kind of transformation *linear transformation.*

Most transformations used in Geometric Reasoning are linear in nature. *Rigid Body Transformation* (e.g., Rotation and Translation) is a type of linear transformation. We often describe these transformations as rigid-body motions because they resemble physical movements. Rigid body transformations preserve the metric properties (i.e., the distance, angle, are, volume, etc. of the geometric objects are invariant).

Translation

A translation is a mapping given by Cartesian equations of the following forms:

$$x' = x + t_x$$

$$y' = y + t_y$$

$$z' = z + t_z$$

In matrix form, the above equations are represented as:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \tag{2.1}$$

The above form however is difficult to mix with other types of transformations such as rotation. Therefore, the computer graphics and computational geometry community uses *homogeneous* representation. The homogeneous coordinates of a point in n-dimensional space consist of $n + 1$ numbers. The homogeneous coordinates of a point p in $\mathbf{E}^3$ space is written as:

$$p = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

In all three-dimensional transformations, we will use a 4×4 matrix T. This is called the *homogeneous transformation matrix.* For this translation, T is as following:

$$T = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

When using homogeneous representation, all types of linear transformations can be calculated as multiplication of matrices. So Equation 2.1 can be rewritten as:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{2.2}$$

Rotation

The simplest rotation in three dimensions are rotations about one or more of the three principle coordinate axes. We denote the angle of rotation as ψ, θ, and ϕ about the x-axis, y-axis, and z-axis, respectively. The rotation matrices for the three angles are as following:

$$R_\phi = \begin{bmatrix} \cos\phi & -\sin\phi & 0 & 0 \\ \sin\phi & \cos\phi & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$R_\theta = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

$$R_\psi = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi & 0 \\ 0 & \sin\psi & \cos\psi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

We may specify the rotation of a point or other geometric object in space as the product of successive rotations about each of the three principle axes. It is important to establish some kind of convention describing how we might do this; here is one way (remember, order is important):

1. Rotation about z-axis (R_ϕ)

2. Rotation about y-axis (R_θ)

3. Rotation about x-axis (R_ψ)

The above convention will produce the transformed point p in the following manner:

$$p' = R_\psi R_\theta R_\phi p$$

Let $R = R_\psi R_\theta R_\phi$, the rotational component of R is called $R_{\psi\theta\phi}$.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & | & 0 \\ r_{21} & r_{22} & r_{23} & | & 0 \\ r_{31} & r_{32} & r_{33} & | & 0 \\ - & - & - & | & - \\ 0 & 0 & 0 & | & 1 \end{bmatrix} \tag{2.6}$$

2.4.4 Faceting

Faceting is an operation that generates approximate polygonal representations called *facets* for the faces of a solid body. While facets can be any polygon, this dissertation deals with triangular facets only. In general, faceted representations are used in rendering, in clearance analysis, and in operations where an approximation is acceptable in order to simplify calculations.

Faceting is performed in four phases: grid spacing determination, edge discretization, face subdivision, and triangulation. Each edge in the object is first subdivided by placing a list of points on the edge. Each face then is subdivided by laying a grid on the face in parameter space. The nodes created by subdividing the edges and faces are triangulated to produce the facets. Figure 2.7(b) shows the faceted version of the object shown in Figure 2.7(a).

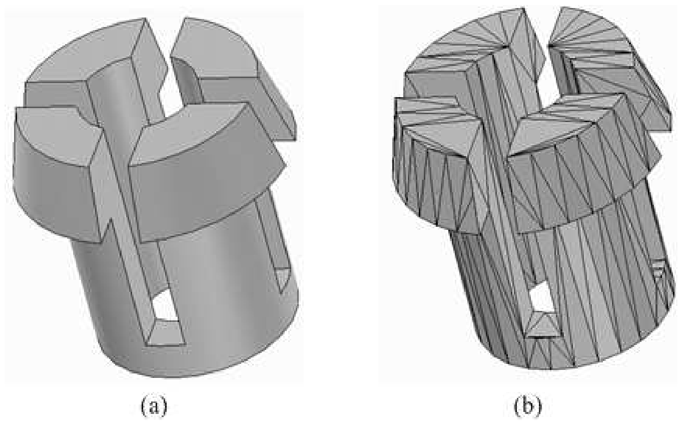


Figure 2.7: Faceting; (a) Original part; (b) Faceted part

By setting maximum surface tolerance, the inaccuracy introduced by faceting can be controlled. The surface tolerance is the distance between the facet and the part of the surface it is representing. Hence by specifying a maximum surface tolerance ϵ, we are ensuring that nowhere the distance between a facet and the true surface is greater than ϵ. The proper value of maximum surface tolerance is dependent on the model size. It is calculated by using the dimensions of the bounding box of the object.

The faceted version of a solid body is a polyhedron. Many schemes are used to represent a polyhedron. We will use the most common representation called the Boundary Representation (BRep). It stores the boundary information of the polyhedron, i.e., topological entities (vertices, edges, and facets) together with the information on how they are connected. The facets are triangles and the edges are line segments with vertices at the endpoints. A polyhedron is considered a valid manifold if the following conditions are satisfied:

1. Each facet must have exactly 3 edges, otherwise it will not be a triangle.

2. Each edge must have exactly two vertices, otherwise it will not be a line segment.

3. The edges associated with a facet must form a loop or closed circuit, to ensure that they enclose a 2-D area. This condition is satisfied if and only if each vertex in a facet belongs exactly to two of the facet's edges.

4. The facets must form one or more closed surfaces or shells, to ensure that they enclose a 3-D volume. This condition is satisfied if and only if each edge belongs to exactly two facets.

5. Each vertex, represented by a 3-tuple of coordinates must correspond to a distinct point in 3-space.

6. Edges must either be disjoint or intersect at a common vertex, otherwise there would be missing vertices in the representation.

7. Similarly, facets must either be disjoint or intersect at a common edge or vertex.

These conditions are easy to establish intuitively, and can be derived mathematically. Conditions 1-4 are combinatorial. They are easy to check algorithmically by counting nodes or links in the boundary graph. In contrast, conditions 5-7 are metric, i.e., they involve coordinates of vertices and equations of lines and planes. They are computationally expensive to check, because they require intersection tests. We conclude that validity checking for BReps is not computationally attractive, and should be avoided. Most geometric modeling systems attempt to embed the required validity conditions in the algorithms used to construct the representations, instead of testing representational validity after the BReps are built. This dissertation only handles manifold polyhedrons.

2.4.5 Convex Hull

Convex hull is one of the most basic concepts in computational geometry. Convex hull serves as a first preprocessing step to many geometric algorithms. Convex hulls are used extensively in the area of collision detection and shape analysis.

The hull quickly captures a rough idea of the shape or extent of a data set. Intuitively, the convex hull of a set of points in a plane is the shape taken by a rubber band stretched around nails punched into the plane at each point. The boundary of the convex hull of points in three dimensions is the shape taken by a plastic wrap stretched tightly around the points. Convex hull of a polyhedron P, denoted as CH(P) is defined as the smallest convex polyhedron that has within it or on its boundary all the defining vertices of the original geometry. Figure 2.8 shows a 3D polyhedral part and its convex hull. The convex hull of a set of points S in n dimensions is the intersection of all convex sets containing S. For N points $P = \{p_1, \ldots, p_N\}$, the convex hull $CH(P)$ is then given by the expression:

$$CH(P) = \{\sum_{i=1}^{N} \lambda_i p_i : \lambda_i \geq 0 \forall i, \sum_{i=1}^{N} \lambda_i = 1\}$$

The above expression also represents the intersection of all *halfspaces* that contain S. A halfspace is the set of points on or to one side of a plane (line in 2D). Note that the convex hull is a 'closed' set. Computing the convex hull for n points takes $O(n \log n)$ time.

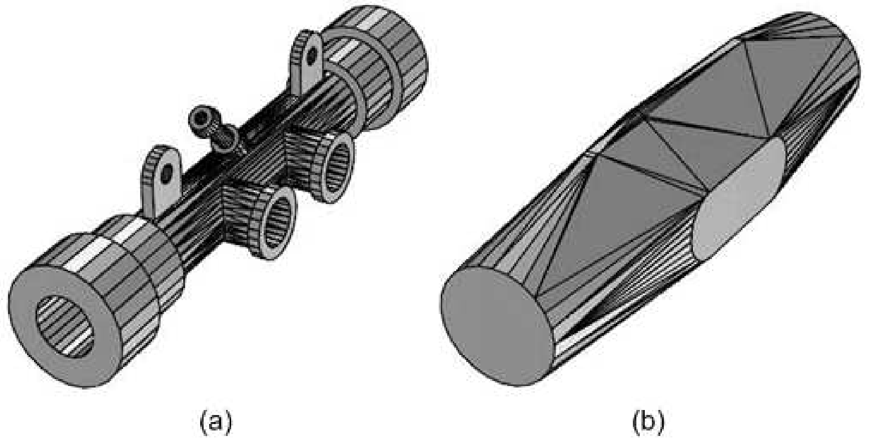


Figure 2.8: 3D Convex Hull; (a) Polyhedral Object; (b) Convex Hull of the Object

2.5 State-Space Search

Many engineering and design problems require finding a feasible or an optimal solution by searching through the solution space. One possible method to solve these problems is to enumerate all the candidate solutions and examine each of them. A feasible or optimal solution is found when all of the candidate solutions have been explored [Sahn98]. In many search problems, during the search process, solutions are generated incrementally by adding additional steps to previously generated partial solutions. In such cases when a complete solution has been found which is as good as all the partial solutions examined so far, then the search can be terminated. When solution spaces are very large, quite often heuristics are used to avoid enumeration of those solutions that are not expected to be the answer to the problem.

In order for algorithms to systematically explore the candidate solutions, the solution space needs to be well organized. State-space graphs are the data structures that are very commonly used for the organization purposes. If each candidate solution consists of an alternative combination and sequence of all the known possible steps, then a most efficient structure for keeping track of the effects of these alternative combinations and sequences of steps is a directed graph, called a state-space graph. Each node in the graph represents a distinguishable state of the world model of the problem. Each edge in the graph represents a step that transforms the world model of the problem from one state to another. One node in the graph representing the initial state is called the start node. Some other node in the graph representing an externally specified goal state is called a goal node [Nils98]. A feasible solution of the problem is a path from the start node to a goal node. If each edge in the graph is assigned some cost values, then an optimal solution of the problem is a path from the start node to some goal node while the total cost of the path is minimal. State-space search algorithms refer to algorithms that solve problems by systematically exploring the state-space graphs of the problems and find feasible or optimal paths in the graph.

There are two broad classes of state-space search processes – uninformed search and heuristic search. Uninformed search is the type of search in which, insofar as finding a path to a goal node is concerned, there are no problem-specific reasons to prefer one part of the state-space graph to another. On the other hand, heuristic search is the type of search in which there is problem-specific information to help focus the search [Nils98].

Breadth-first search and depth-first search are the two fundamental types of algorithms used for uninformed search [Nils98]. Breadth-first search examines all the direct successors of the start node first, then again the direct successors of these examined successors, and so on. The action of finding the successors of a node is called expanding the node. Once the procedure finds the node to be expanded next is a goal node for the first time, it guarantees to have found the path from the start node to this goal node as a feasible solution that includes the least number of steps. The procedure terminates if the problem only requires a feasible solution, or continues until a feasible solution has been found and proved to be an optimal solution to the problem. A disadvantage of breadth-first search, however, is that it requires the generation and storage of a tree whose size is exponential in the depth of the node at which the procedure terminates.

Depth-first search generates the successors of the start node just one at a time. As soon as a successor is generated, it is examined and one of its successors is generated, and so on. But no successor is generated whose depth is greater than a depth bound. The depth bound is a presumed bound that has the following property: not all goal nodes lie beyond this bound in terms of the distances from the start node. Using depth bound, searching algorithms ignore those parts of the graph that do not contain a sufficiently close goal node. A trace is left at each node to indicate that other successors of it can be generated and examined later through backtracking if needed. Only the path currently being explored, as well as the traces at nodes that are not yet fully expanded, is needed to be stored for depth-first search. Since the length of a path is no greater than the depth bound, depth-first search only requires memory storage that is linear in the depth bound. A disadvantage of depth-first search, however, is that when a goal node is found, it does not guarantee to have found a solution that includes the least number of steps. Depth-first search suffers from another problem. If a shallow goal is the only goal node and a successor of a node expanded late in the process, then the depth-first search may have to explore a large part of the search space to find this goal node.

Heuristic search is also called best-first search. Heuristic search, unlike uninformed search, makes use of problem-specified information for guiding the search procedure. It proceeds preferentially through nodes that problem-specific heuristic indicates might be on the best path to a goal. Heuristic search is especially useful in solving real engineering problems. The basic idea of heuristic search is the following [Nils98]:

1. A heuristic evaluation function is used to help decide which node is the best one to expand next. This evaluation function is based on information specific to the problem domain.

2. Always expand next the node that has the minimum value from the evaluation function.

3. Terminate when the node to be expanded next is a goal node or when an already examined goal node has lead to an optimal solution.

Given a node n in the state-space graph, the heuristic evaluation function $f(n)$ is defined in the following manner:

$$f(n) = g(n) + h(n) \tag{2.7}$$

where,

- $h(n)$ is some heuristic estimate of the cost of the minimal cost path between node n and a goal node (over all possible goal nodes and over all possible paths from n to them), and

- $g(n)$ is the cost of the lowest-cost path found so far from the start node to node n.

In implementation, all successors can be stored in a priority queue [Sahn98] in the non-decreasing order of evaluation function values, and each time the first node in the queue is to be expanded next.

Chapter 3

PROBLEM FORMULATION

This chapter defines the problem being investigated in this dissertation. The first two sections define the concepts required to describe the problem. Section 3.1 describes a mathematical model for representing an articulated assembly, while Section 3.2 defines the basic mold design terminology. Section 3.3 and Section 3.4 describe the problem and associated constraints. Section 3.5 finally presents the problem statement and Section 3.6 presents an overview of the technical approach that has been adopted to solve the problem.

3.1 A Model for Articulated Assemblies

In order to develop molding plans for articulated assemblies, we first need to develop a mathematical model to describe them. We are specifically interested in describing the structure and configuration of articulated assemblies. An articulated assembly can be informally defined as an object with movable parts, or a set of rigid bodies connected together so as to allow motion between them.

We first define a model for rigid-body articulated joints in Section 3.1.1. We also describe the motion equations for three basic types of joints – prismatic, revolute, and spherical. Section 3.1.2 defines a model for describing the configuration space of rigid-body articulated assemblies. We will use these mathematical models for describing the molding plans for articulated joints and assemblies.

3.1.1 Articulated Joints

An articulated joint can be defined as a *connection* between two rigid bodies having relative motion. Each articulated joint allows (1) translational motion, (2) rotational motion, or (3) a combination of translational and rotational motion. An articulated joint represents one or more mechanical degrees of freedom (DoF) between the two bodies it connects.

One of the connected bodies by convention plays the role of *base* and the other body plays the role of *follower*. The base is regarded as fixed, while the follower moves with respect to the base. For ease of computing the kinematic equations for the connected bodies, the motion of the follower is described with respect to a coordinate frame attached to the base. The position and orientation of the follower is defined in terms of *joint variables.* A joint variable represents a DoF of the joint. So the number of joint variables for a joint is equal to the number of DoF of that joint. A joint variable j is defined as:

$$j = (\hat{t}, \theta) \tag{3.1}$$

where $\hat{t}$ is the axis and θ is the coordinate of relative motion between the connected bodies. For translational motion, the joint variable is a linear coordinate along a direction. For rotational motion, the joint variable is an angular coordinate about an axis. For ease of calculation, coordinate frame attached to the base is aligned with the joint axes.

A *primitive* joint expresses a single DoF or coordinate of motion. There are two types of primitive joints – prismatic joint for translation and revolute joint for rotation. Other more complicated composite joints can be modeled in terms of these two primitive joints. The spherical joint, which can be modeled as a combination of three revolute joints is sometimes also treated as a primitive joint.

Prismatic Joint.

A prismatic joint (Figure 3.1) allows one translational degree of freedom along a direction. Prismatic joints are seen in a wide variety of assembled objects. Applications range from the well-known slider-crank mechanisms, used in the cylinder of an internal combustion engine, to more recent mechanisms found in CD and DVD drives.

The motion of a point p on the follower with respect to the base can be described in terms of joint variables as follows. Suppose the point p has an initial position $\vec{p_0} = (x_0, y_0, z_0)$. The joint axis, or the direction of motion of the follower is $\hat{t}$. For a given joint parameter, or amount of translation θ, the new position of p is given by:

$$\vec{p'} = \vec{p_0} + \theta \cdot \hat{t} \tag{3.2}$$

Revolute Joint.

A revolute joint (Figure 3.2) allows one rotational degree of freedom about a specified axis. Pin connections are an excellent example of a simple revolute joint. Revolute joints are extensively used in making robots and bar linkages.

The motion of a point p on the follower with respect to the base can be described

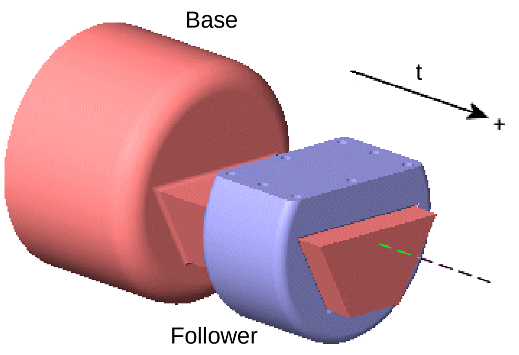


Figure 3.1: Prismatic joint [Simm06]

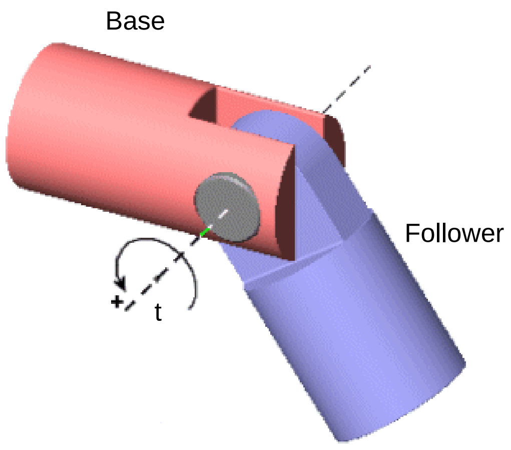


Figure 3.2: Revolute joint [Simm06]

in terms of the joint variable $j = (\hat{t}, \theta)$ as follows. Suppose the point p has an initial position $\vec{p_0} = (x_0, y_0, z_0)$. The new rotated position of p is given by [Weis99]:

$$\vec{p'} = \vec{p_0} \cos\theta + \hat{t}(\hat{t} \cdot \vec{p_0})(1 - \cos\theta) + (\vec{p_0} \times \hat{t}) \sin\theta \tag{3.3}$$

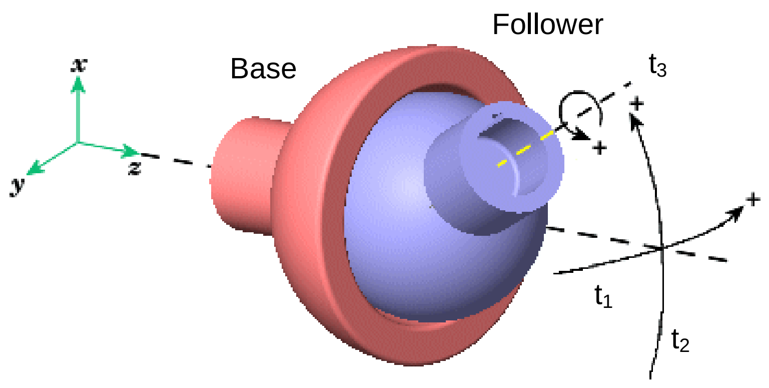


Figure 3.3: Spherical joint [Simm06]

Spherical Joint.

A spherical joint (Figure 3.3) allows three rotational degrees of freedom at a single pivot point. A common example of this type of joint is known as the ball and socket joint. This type of joint is well known for its uses in the human body. As shown in Figure 3.3, the motion of follower connected by a spherical joint is described in terms of three joint variables – one joint variable for each degree of freedom.

The spherical joint can be modeled as a combination of three revolute joints. Two rotational DoFs specify a directional axis, and the third rotational DoF specifies rotation about that directional axis [Simm06]. The three rotational axes are perpendicular to each other. The equation of motion for the spherical joint is a simple concatenation of three rotational motions.

3.1.2 Articulated Assemblies

An articulated assembly is defined as a structure A which is composed of:

- Two or more rigid lumps $L = \{l_1, \ldots, l_n\}$, and
- Articulated joint variables $J = \{j_1, \ldots, j_m\}$ between pair of lumps as defined in Equation 3.1

Every articulated assembly has an implicitly defined configuration space (CS). The CS has dimension equal to the total number of degrees of freedom of the assembly. A point in the CS specifies a particular configuration of the assembly. We define a configuration of the assembly as:

$$T = (\Theta, \bar{T}) \tag{3.4}$$

where

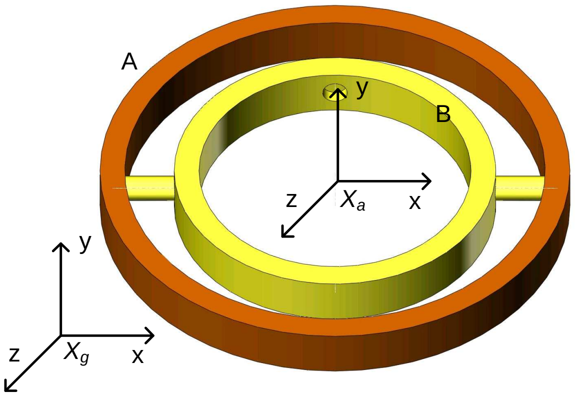
- $\Theta = \{\theta_1, \ldots, \theta_m\}$ are the joint coordinates defined in Equation 3.1, and
- $\bar{T}$ is the homogeneous transformation applied to the whole assembly.

By selecting different Θ and $\bar{T}$, we can generate different assembly configurations. In the initial configuration, all the joint coordinates θ_i^0 are equal to zero and $\bar{T}^0$ is an identity matrix.

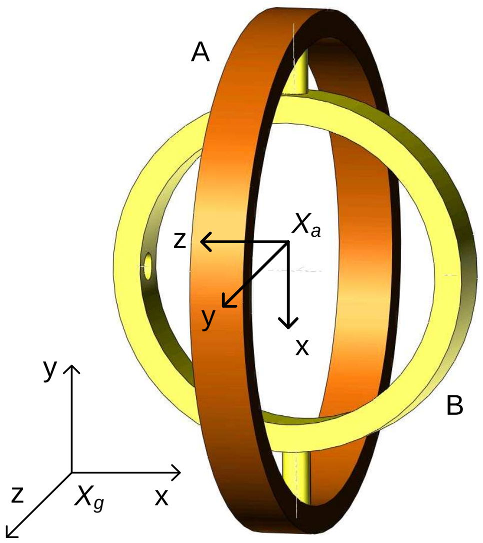
$$\theta_i^0 = 0, \forall i \in \{1, \ldots, m\} \tag{3.5}$$

$$\bar{T}^0 = I = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.6}$$

The configuration scheme is illustrated in Figure 3.4. The shown assembly consists of two components A and B connected by revolute joints. There are two revolute joints in the assembly, but the joint axes for both the joints are collinear. This means that for configuration purposes, they are equivalent and one



(a) Initial configuration



(b) A new configuration

Figure 3.4: Assembly configuration scheme.

of them can be neglected. Let the selected joint be $j_1 = (\hat{t_1}, \theta_1)$. We will consider component A as base and component B as follower. Hence, motion of B will be measured with respect to the coordinate system X_a attached to A. With respect to X_a, the joint axis for the joint j_1 is along the x-axis. The coordinate system X_g is attached to the ground, i.e., fixed.

Figure 3.4a shows the initial configuration, where $\bar{T}^0 = I$ and $\theta_1^0 = 0$. Figure 3.4b shows another configuration obtained by rotating the whole assembly about the z-axis of X_g by 90°, and rotating B about the x-axis of X_a by 90°. The parameters for this configuration are given by:

$$\bar{T} = [0, 0, 1, 90^\circ] = \begin{bmatrix} \cos 90^\circ & -\sin 90^\circ & 0 & 0 \\ \sin 90^\circ & \cos 90^\circ & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

$$\theta_1 = 90^\circ \tag{3.8}$$

We need to calculate the transformation matrix for components A and B to describe their position and orientation with respect to the fixed coordinate system X_g. The transformation matrix T_A for A is simply equal to $\bar{T}$. B is subjected to two successive rotations. Matrices for such transformations are calculated by chaining the matrices of all transformations together.

$$T_A = \bar{T} \tag{3.9}$$

$$T_B = T_A \cdot T_{B/A} \tag{3.10}$$

$$T_{B/A} = [1, 0, 0, 90^\circ] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos 90^\circ & -\sin 90^\circ & 0 \\ 0 & \sin 90^\circ & \cos 90^\circ & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

$T_{B/A}$ is the transformation of B with respect to A. This is equivalent to the transformation of B with respect to the coordinate system X_a attached to A. As can be seen aligning the coordinate frame with the joint axes simplifies the calculations.

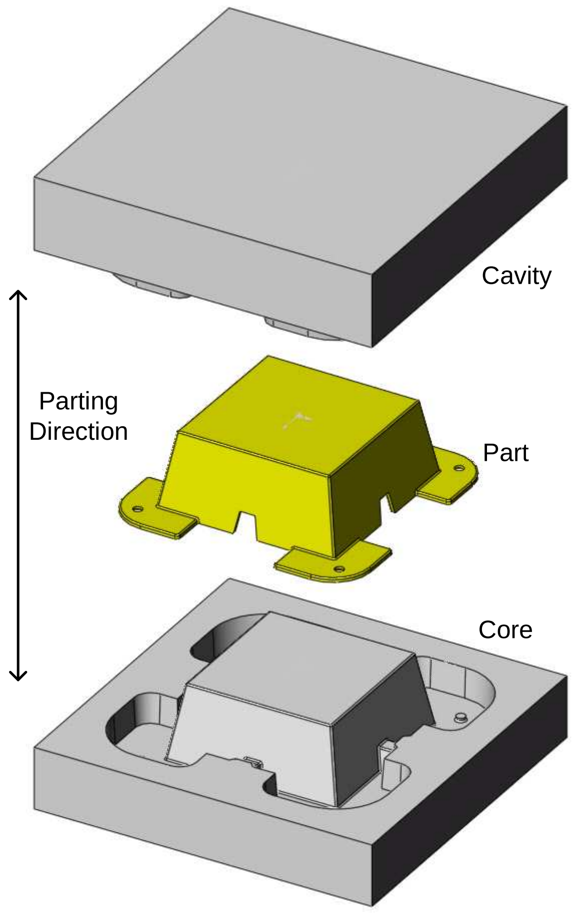


Figure 3.5: Mold pieces mold for an example part

3.2 Definition of Mold Design Terms

An injection mold consists of two main pieces – *core* and *cavity*. The core and cavity form the impression into which molten material is injected. The cavity determines the external shape of a molded part and the core forms the internal shape. Figure 3.5 shows the mold pieces for an example part.

Parting Direction is the direction along which core and cavity are separated from the molded part. Figure 3.5 shows the parting direction for the mold pieces.

A *Mold-Piece Region* of a part is a set of part facets that can be formed by a single mold piece. There can be four types of mold-piece regions – core region (C_o), cavity region (C_a), both region (B_o), and undercut region (U_c). Figure 3.6 shows various mold-piece regions for a part. Given a polyhedral object P and a parting direction $\vec{d}$, each set of part facets has the following property:

1. C_o, which is formed by core, is accessible from $+\vec{d}$, but not $-\vec{d}$.
2. C_a, which is formed by cavity, is accessible from $-\vec{d}$, but not $+\vec{d}$.
3. B_o, which can be formed by either of them, is accessible from both, $+\vec{d}$ and $-\vec{d}$.
4. U_c, which cannot be formed by either of them, is not accessible from either $+\vec{d}$ or $-\vec{d}$. The undercut region is formed by side actions that are usually separated in a direction perpendicular to the parting direction.

A facet f on a polyhedral object P is accessible in a direction $\vec{d}$, if for every point p on f, the ray starting from p to infinity in the direction $\vec{d}$ does not intersect the interior of P.

Parting Line of a part is a continuous closed curve on the surface of the part that defines faces to be split into different mold pieces. Hence, a parting line is actually the boundary of a mold piece region as shown in Figure 3.6.

ShutOff Surface is the contact surface of two mold pieces. Another property of the shutoff surface is that it meets the part at the parting line. Figure 3.7 shows the parting line and the parting surface of the part shown in Figure 3.5.

Mold enclosure is an oriented rectangular block enclosing the object.

Gross Mold of an object is a connected solid obtained by subtracting the object from its mold enclosure. The gross mold is split into core and cavity using shutoff surface.

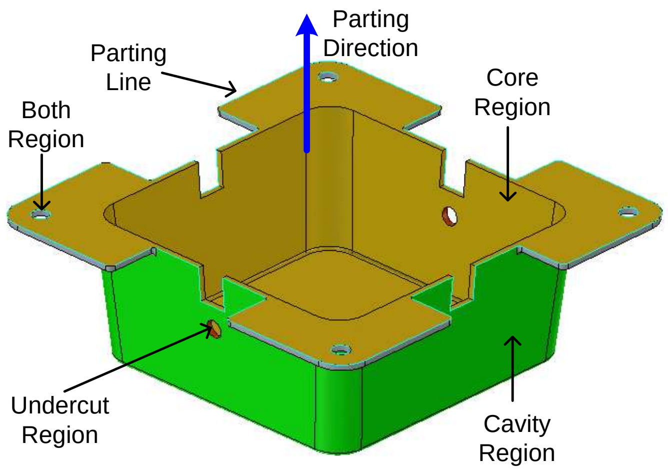


Figure 3.6: Mold-Piece Regions

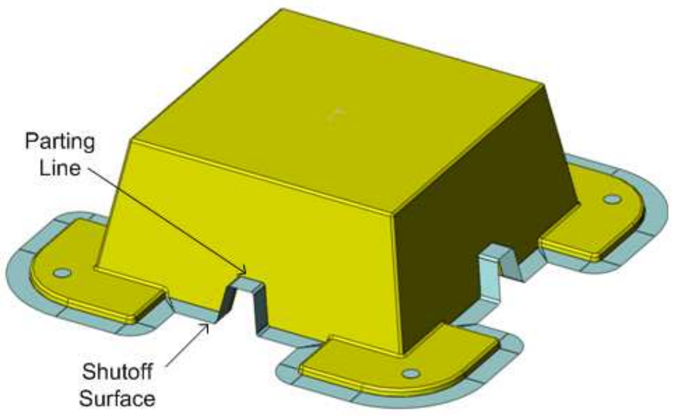


Figure 3.7: Shutoff surface for an example part.

3.3 Generating Molding Plan

Generating molding plan is similar to other manufacturing process planning activities that translate design information into process steps and instructions to efficiently manufacture products. A molding plan consists of a sequence of molding stages required to mold an articulated assembly.

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 m may not be equal to the number of components n. In each molding stage s_i, a set of components C_i is added to the already molded subassembly 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 stage s_i is described by:

1. The set of components C_i to be molded in stage s_i
2. The parting direction d_i for the stage s_i
3. The configuration of the subassembly A_i in which molding will take place in stage s_i

On closer observation, we find that instead of specifying both, the parting direction and the subassembly configuration, we can fix the parting direction to be the same for all molding stages, and the subassembly configuration will adjust accordingly. Without the loss of generality, we can assume that the parting direction is always along the z-direction.

So each molding stage s_i can be represented as:

$$s_i = (C_i, T_i) \tag{3.12}$$

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 as defined in Equation 3.4

Figure 3.8 shows the molding plan for a three-material articulated gimbal. Figure 3.8a shows the gimbal in given initial configuration. Figure 3.8b shows the first molding stage s_1. The component set C_1 consists of a single component A. The molding configuration T_1 is obtained by rotating the subassembly C_1 about the x-axis by 90°. Figure 3.8b and Figure 3.8c show second and third molding stage respectively. The parameters for each molding stage are given below.

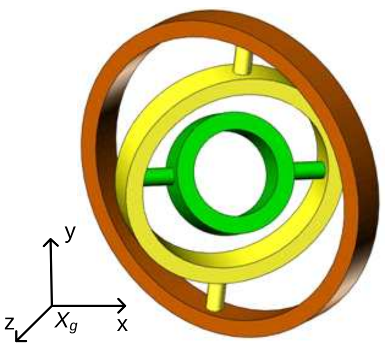
$\mathbf{s_i}$	$\mathbf{C_i}$	$\mathbf{T_i}$
s_1	$\{A\}$	$\bar{T} = [1, 0, 0, 90°]$
s_2	$\{B\}$	$\bar{T} = [1, 0, 0, 90°], \theta_1 = 90°$
s_3	$\{C\}$	$\bar{T} = [1, 0, 0, 90°], \theta_1 = \theta_2 = 90°$

3.3.1 Feasiblity of a Molding Plan

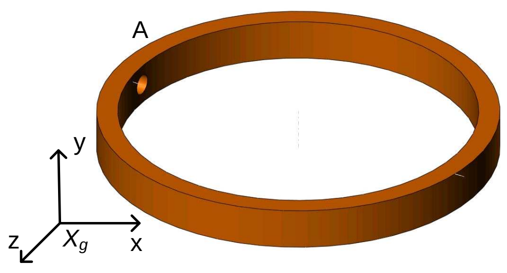
A molding plan is considered feasible if the sequence of molding stages leads to the desired articulated assembly. The first requirement for a molding plan to be feasible is that all the molding stages are valid. A molding stage s_i is considered valid if:

1. Each component in C_i has the same material attribute m_i. In multi-stage molding, the various materials are injected sequentially, i.e., only one type of material is injected in a molding stage.

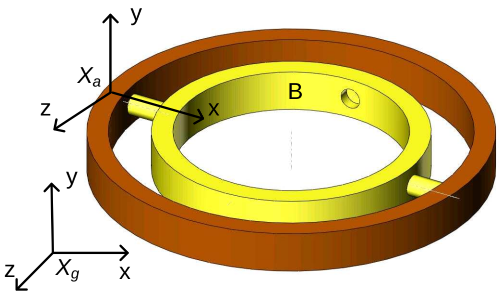
2. Each component in C_i is connected to a common base component a_i. If the components to be injected are distributed throughout the assembly, designing a runner system to feed the material becomes very difficult. So it is required



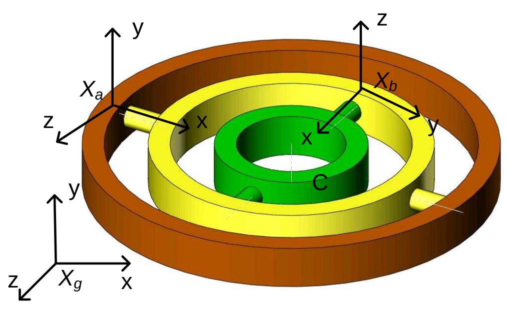
(a) Gimbal



(b) Molding stage 1



(c) Molding stage 2



(d) Molding stage 3

Figure 3.8: Molding plan example.

that all the components to be molded in a stage are conneceted to a common base component.

3. 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

   (a) Intersect with each other, i.e., $a_p \cap^* a_q = \emptyset, \forall p, q \in \{1, \ldots, k\}$ and $p \neq q$

   (b) 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 3.9 shows an invalid configuration where component A casts shadow on component B.

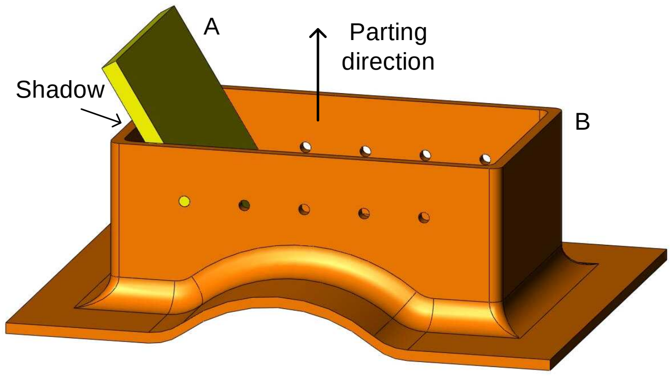


Figure 3.9: Component A casts shadow on component B

The feasibility requirement for a molding plan can be summarized as follows. 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
2. Each component a_i is assigned to exactly one molding stage, i.e., $C_i \cap C_j = \emptyset, \forall i, j \in \{1, \ldots, n\}$ and $i \neq j$
3. The molding sequence is consistent with one of the known feasible molding plans.

3.3.2 Cost of a Molding Plan

This section describes a cost equation for a molding plan. This cost equation is used to compare two molding plans. Hence we only consider the cost that varies between two molding plans. We neglect the costs that are the same in all possible molding plans for an assembly. Such constant costs consist of the following:

1. Material cost

2. Undercut cost required for joint features

3. Machining cost for mold surface corresponding to the part surface

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. *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.

   $$C_m = k_1 + (k_2 T_h^{1.4}) + (k_3 N_u) \tag{3.13}$$

   where,

   - k_1 is the constant stage setup cost which consists of mold loading time and stage transfer time that take 2 hours and 2 minutes respectively. Using injection molding rate at \$50 per hour, we get $k_1 = \$203.30$.
   - T_h is the maximum wall thickness (in mm) of the injected components and $k_2 = 1.5$.

- N_u is the total number of undercuts on the components for which side actions are required. It takes about 5 seconds to operate a side action, hence we will consider $k_3 = 0.7$.

2. *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. 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:

$$C_d = 1/\rho \tag{3.14}$$

   where ρ is a measure of flatness of the parting line. Increasing the flatness decreases the defect cost.

3. *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:

$$C(s) = k_4 A(s) + k_5 N(s) \tag{3.15}$$

   where,

   - $A(s)$ is the surface area (in sq. inch) of s. It takes about 2 hours to machine each sq. inch. using machining rate at \$50 per hour, we get $k_4 = 100$.

- $N(s)$ is the number of surface normals on s. It takes 10 minutes per surface normal, which gives $k_5 = 8.5$.

Hence the tooling cost can be written as:

$$C_t = k_4 A_s + k_5 N_s \tag{3.16}$$

where A_s is the area of the shutoff surface and N_s is the number of normals on the shutoff surface.

From the above, the cost of a molding stage can be written as:

$$c_i = C_m + C_d + C_t \tag{3.17}$$

However, 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 dissertation 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. Hence for all practical purposes, the cost of a molding stage can be safely approximated as following:

$$c_i \approx C_m \tag{3.18}$$

The total cost of a molding plan $S = \{s_1, \ldots, s_n\}$ is the sum of the cost of each molding stage s_i.

$$C = \sum_{1}^{n} c_i \tag{3.19}$$

3.4 Multi-Stage Mold Design for Articulated Assemblies

For a given articulated assembly, a multi-stage mold can be designed in two steps. A molding plan is first generated in the first step. The molding plan consists of a sequence of molding stages as described in Section 3.3. The second step generates a mold design for each molding stage. For each molding stage $s_i = (C_i, T_i)$, it adds the stage components (C_i) to the already molded subassembly (A_{i-1}) and orients the resulting subassembly (A_i) in the specified configuration T_i. The resulting subassembly (A_i), is considered as a single homogeneous object for which a mold needs to be designed. Designing a mold for a single homogeneous object has been widely studied. It mainly consists of four steps – find mold-piece regions, create parting line, create shutoff surface, and create mold pieces. These mold design terms are defined in Section 3.2. The third molding stage of gimbal shown in Figure 3.8d will be used as an example to illustrate the mold design process for a molding stage.

1. *Finding Mold-Piece Regions*: A *Mold-Piece Region* of a part is a set of part facets that can be formed by a single mold piece. A facet can belong to one of the four mold-piece regions – core region (C_o), cavity region (C_a), both region (B_o), and undercut region (U_c) depending on its accessibility along the parting direction. Hence, the problem of finding mold-piece regions reduces to performing accessibility analysis of P along $+\vec{d}$ and $-\vec{d}$ and decomposing the part facets F into four sets C_o, C_a, B_o, and U_c. Figure 3.10 shows the various mold-piece regions of the third molding stage of gimbal. The cavity region is not visible in the figure, but it is similar to the core region.

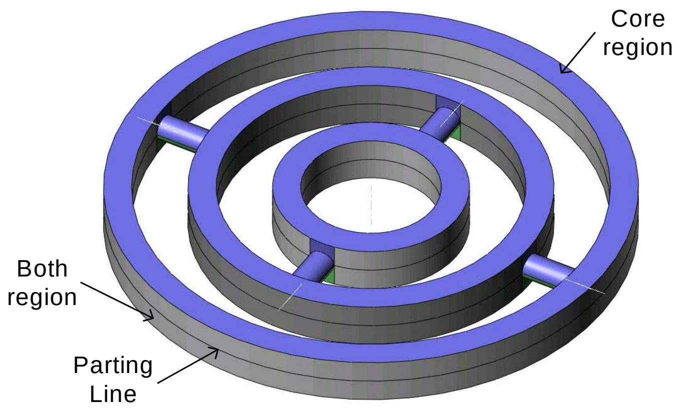


Figure 3.10: Mold-piece regions and parting line for the third molding stage of gimbal shown in Figure 3.8d.

2. *Creating Parting Line*: A parting line of a part is a continuous closed curve on the surface of the part that defines faces to be split into different mold pieces. Hence, a parting line can be the boundary of a mold piece region. However, flat parting lines are cheaper to manufacture. So it is required that the parting line be as flat as possible. Figure 3.10 shows the parting line for the third molding stage of gimbal shown in Figure 3.8d. It can be seen in the figure that the parting line does not follow the boundary of the core region. It lies on a plane that goes through the middle of the both region. Since both region can be molded by any mold piece, core or cavity, the parting line can be placed inside both region.

3. *Creating Shutoff Surface*: A shutoff surface is the contact surface of two mold pieces. The parting line of a part consists of one outer loop and may have multiple inner loops. The gimbal example has three inner loops. The shutoff surface is created by covering the inner loops by a surface, and extending the outer loop. Figure 3.11 shows the shutoff surface for the third molding stage

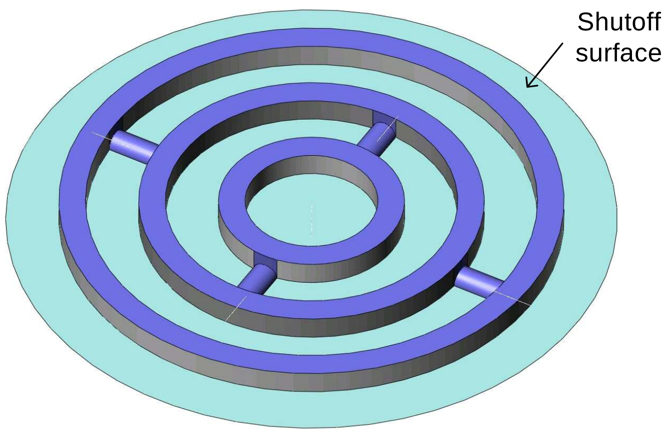


Figure 3.11: Shutoff surface for the third molding stage of gimbal shown in Figure 3.8d.

of gimbal.

4. *Creating Mold Pieces*: A mold enclosure is first created for the given part. The part is then subtracted from the mold enclosure to form the gross mold. The shutoff surface is used to split that gross mold into core and cavity mold pieces. Figure 2.4d shows the cavity mold piece for the third molding stage of gimbal. The core mold piece, which is similar to the cavity mold piece is not shown for clarity.

3.5 Problem Statement

Problem GENERATEMOLDINGPLAN

Input: Multi-material articulated assembly A as defined in Section 3.1.2.

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 3.12.

Output Requirements:

- The molding plan is feasible as defined in Section 3.3.1
- The molding plan is optimal, i.e., the cost C of the molding plan as defined in Section 3.3.2 is minimum.

Input Restrictions:

- 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.

3.6 Overview of Approach

Generating molding plans for articulated assemblies is a challenging planning problem. This requires determining the sequence and configuration in which the assembly components will be molded. In a typical case, very few feasible plans exist. Unfortunately, many plan feasibility constraints are order-dependent (e.g., a molding pose that is feasible in one sequence may not be feasible in another sequence). Hence, a large number of planning constraints cannot be generated a priori. Moreover, plan parameters must be selected such that the resulting molding plan is optimal. Many of the subproblems (determining mold-piece regions, parting line, and shutoff surface) that need to be solved as part of this overall planning problem require geometric reasoning. This eliminates the use of symbolical reasoning type of planning techniques. These factors combined together make this problem a challenging process planning problem.

There are two traditional approaches to process planning. The first approach is called *generative* process planning. In this approach a plan is synthesized from the first principles by trying various alternatives in generate-and-test paradigm. The second approach is called *variant* process planning. In this approach, a plan is generated by modifying an existing plan. Both approaches have their relative strengths and weaknesses. Generative approaches can generate a plan for any planning problem instance from the domain. However, generative approaches are computationally more expensive. Moreover, the required domain knowledge must be explicitly represented and used by the planning system. Variant approaches can generate plans for only those planning problems for which a close enough plan already exist. But they are relatively faster. Moreover, a previously generated plan may contain useful proper knowledge in an implicit form (e.g., process parameter settings used in the plan might have been selected because they reduce chatter). This implicit knowledge can be incorporated in the new plan.

Purely generative approaches are unlikely to work in the molding planning domain. A lot of knowledge that is needed to successfully mold joints does not exist in an explicit geometric form. For example, in order for a revolute joint of a certain size to work, one may have to experimentally determine the molding sequence and a set of molding poses. Currently, effects of molding parameters on joint clearances and flash generation are not well understood. Joint clearance and flash affect how well a given joint will work. To successfully create a joint one also needs to design a runner and a cooling system. Currently, an explicit model for solving these problems also does not exist. Many of these problems are currently solved by trial and error. In the absence of a framework for explicitly modeling the required planning constraints, it will be impossible to guarantee the feasibility of a generated plan without performing experimental validation.

Purely variant approaches are also unlikely to work in the molding planning problem. Every new assembly is significantly different from the previously generated assemblies. Hence minor modification of a previous molding plan is unlikely to work. In most practical cases, major changes in molding sequences and poses will be needed.

Variant approaches, can however be applied to developing molding plans for individual joints. The various types of joints used are very few. So it is very common to find similar joints in new assemblies. When a successful plan is developed and experimentally validated for molding a joint, a lot of useful knowledge is generated. When a joint similar to a previously molded joint is found in a new assembly, the previously generated knowledge can be applied to the new joint. Hence, it appears that plans that exist for molding individual joints can be reused in new assemblies. Rather than reusing the entire molding plans from the mold assemblies, we can synthesize new plans by reusing plans for individual joints.

In this dissertation, we use a hybrid approach that combines elements from generative and variant techniques. We reuse molding plans for individual joints to generate plans for new assemblies. This allows us to reuse existing molding knowledge and yet ensure that we can handle a wide variety of molding planning problems. For each joint type, we store *reusable molding plans.* When a new assembly is encountered we first check if the joints used in the assembly are sufficiently similar to the joints for which plans exist. This is performed by comparing the type, size, geometry, and material of the joints. If a joint in the assembly is sufficiently similar to a joint with the known molding plan, then the molding plan for the joint is used as feasibility constraints in the planning process. This

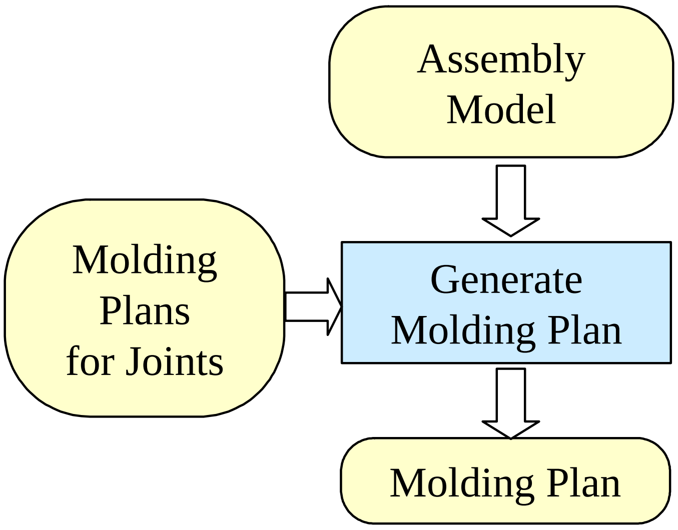


Figure 3.12: Overview of approach

ensures that our method will only generate feasible plans. The rest of the planning proceeds in a generative manner. The overall hybrid approach is illustrated in Figure 3.12.

Chapter 5 presents a framework for representing reusable molding plans for articulated joints. This representation is used to build a database of reusable molding plans for common types of joints. The proposed representation is comprehensive in the sense it captures all important information and makes it easy to classify and catalog the molding plans. This chapter also presents molding plans for three basic joints – prismatic, revolute, and spherical.

Chapter 6 is the main chapter of this dissertation. It describes an algorithm for generating multi-stage molding plan for articulated assemblies. The algorithm formulates the molding plan problem as a state-space search and adopts a combination of generative and variant approach to reach an optimal solution. It solves the problem in two steps. It first uses the molding plan database described in Chapter 5 to reduce the search space, and then the geometric reasoning algorithms presented in Chapter 7 and Chapter 8 to generate an optimal molding plan.

Chapter 7 describes an algorithm to robustly and efficiently find the mold-piece regions for a given object. We have developed two versions of the algorithm – object-space and image-space. The image-space version runs on current-generation computer graphics hardware (GPU). The image-space version exploits the computational power offered by the GPUs to find a solution in real time. The object-space version runs on CPU and can be used where special graphics hardware is not available. This algorithm is used by the molding plan algorithm (Chapter 6) to evaluate the feasibility of a molding stage configuration.

Chapter 8 presents an algorithm to create a provably correct and optimal shut-off surface for a given parting line. This algorithm is used by the molding plan algorithm (Chapter 6) to evaluate the optimality of a molding stage configuration. The algorithms for finding mold-piece regions and creating shutoff surface can also be used by the designers as a software tool to design mold for a molding stage.

Chapter 4

RELATED WORK

This chapter provides a review of the related work and the state of the art in geometric algorithms for mold design. Automation of two-piece mold design has been studied very widely. Multi-piece and Multi-stage mold design are relatively new concepts and hence very few papers have been published in these domains. This chapter has been organized in the following manner. Section 4.1 presents the work done in accessibility analysis that forms an integral component of many mold design algorithms. Sections 4.2, 4.3, and 4.4 review the representative approaches in the two-piece, multi-piece, and multi-stage mold design respectively. Section 4.5 reviews the assembly planning literature. Section 4.6 reviews the representative work in hybrid generative/variant process planning.

4.1 Accessibility Analysis

Accessibility analysis of a surface seeks to determine the directions along which the surface is *accessible* in presence of an obstacle. Accessibility analysis is used to perform process planning in a number of different manufacturing applications. In machining, accessibility analysis is used to find the set of directions from which the part may be approached by the cutting tool. This helps in determining the work-piece orientations that would minimize the number of set-ups required for machining the part [Suh95, Kang97] and helps in cutter path planning [Bala00]. Accessibility analysis is also used in disassembly-based decomposition of the gross mold to ensure part ejection. It allows the selection of parting surface that minimizes or eliminates the undercuts. This helps in reducing the number of side cores required in the mold design [Chen93, Wein97, Dhal01, Priy04]. In assembly operations, it is used to find the directions from which the assembly and disassembly operations can be carried out. Accessibility analysis is also used in automated planning and programming tasks with a Coordinate Measuring Machine (CMM). It helps in determining part orientation on the CMM and identifying the directions from which a probe can approach the part to perform measurements [Spit99]. We use accessibility analysis to find mold-piece regions in Chapter 7.

Many different papers have been published in the area of accessibility analysis for a variety of manufacturing domains. They can be broadly classified into two categories – approaches that perform local analysis and those that perform global analysis.

4.1.1 Local Accessibility analysis

A Gaussian map of a surface is the set of end points of the unit normal vectors of the surface. Gaussian maps can be represented as a spherical region (i.e., a subset of the boundary of a unit sphere). By extending the basic idea behind Gaussian maps, Chen et al. [Chen93] developed the concept of Visibility Map to represent *local* accessibility.

Visibility Map of a surface is an spherically convex region. Any point in the

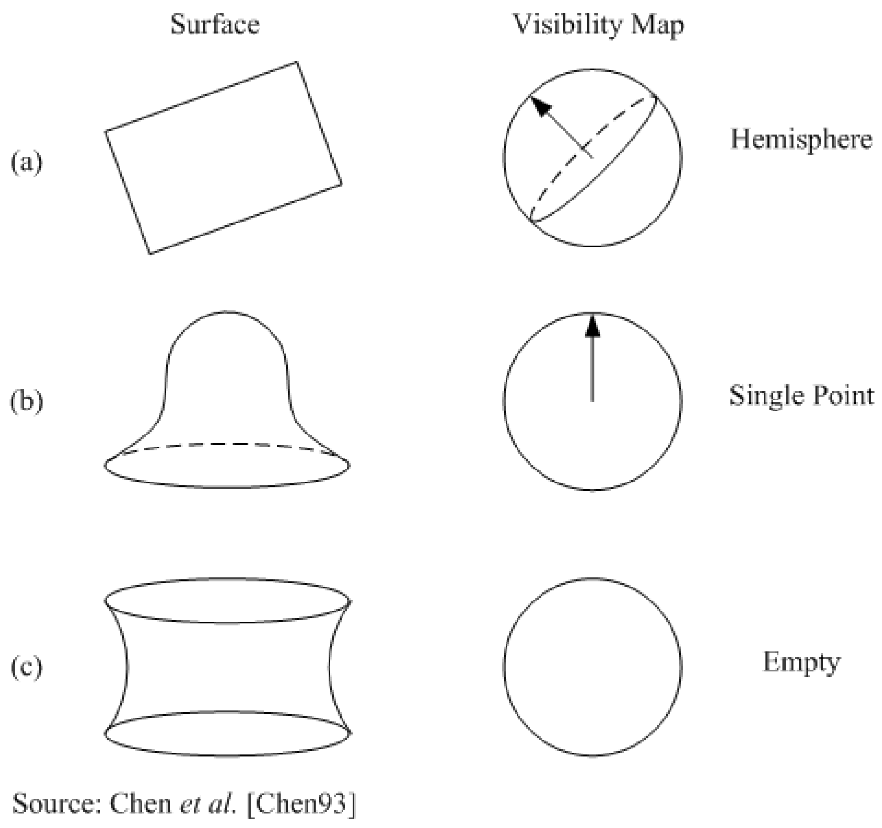


Figure 4.1: Visibility Maps of some simple surfaces

visibility map of a surface corresponds to a direction from which the entire surface is locally accessible. Local accessibility of a point on a surface is defined by the hemispherical region constructed by using the surface normal at the point as the pole. Therefore, the visibility map of a point is a hemispherical region on a unit sphere. The visibility map of a surface is the intersection of visibility maps of all the points on the surface. The visibility map of a region is the intersection of visibility maps of all the surfaces in the region. Figure 4.1 shows the visibility maps of some simple surfaces.

The accessibility of a surface can be interfered by both, parts of the same surface (*local interference*) and, by other surfaces of the object (*global interference*). Since the faces of polyhedral parts are planar, there is no local interference, i.e., any point on a face does not block any other point on the same face. The notion of *pockets* is introduced for detecting global interference. The boundary of the part is divided into spatially independent convex and concave regions (pockets) by taking the regularized difference between the convex hull of the object and the object itself. A ray emanating from a point in a pocket will either intersect a surface in the same pocket or go to infinity. Hence, the visibility map of a pocket represents a set of directions from which all the faces in the pocket are globally accessible.

But as soon as a pocket is decomposed, and visibility maps are calculated for decomposed parts separately, these visibility maps no longer represent global accessibility. This is because the visibility map of a region is constructed using the local visibility of region surfaces without considering other surfaces on the object. Figure 4.2 shows an example that illustrates this observation. Faces A, B, and C form a concave region R. The visibility map of R (Figure 4.2(b)) contains only one direction $\vec{d_1}$. This means that the faces A, B, and C are completely and globally visible from only one direction $\vec{d_1}$. The visibility map of face A, calculated separately without considering B and C, contains all directions in a hemisphere (Figure 4.2(c)). It can be seen in Figure 4.2(a) that though direction $\vec{d_2}$ is present in the visibility map of A, a point p on A is not visible from $\vec{d_2}$. The above argument can be summarized as follows. The visibility map represents global visibility of a face present in a concave region R only if it has been calculated considering all the faces present in R. Otherwise it represents local visibility only.

The concept of visibility maps has been extended by Kim et al. to cover bezier surfaces [Kim95]. They have defined and provided algorithms for computing tangent, normal and visibility maps for regular bezier surfaces. Elber and

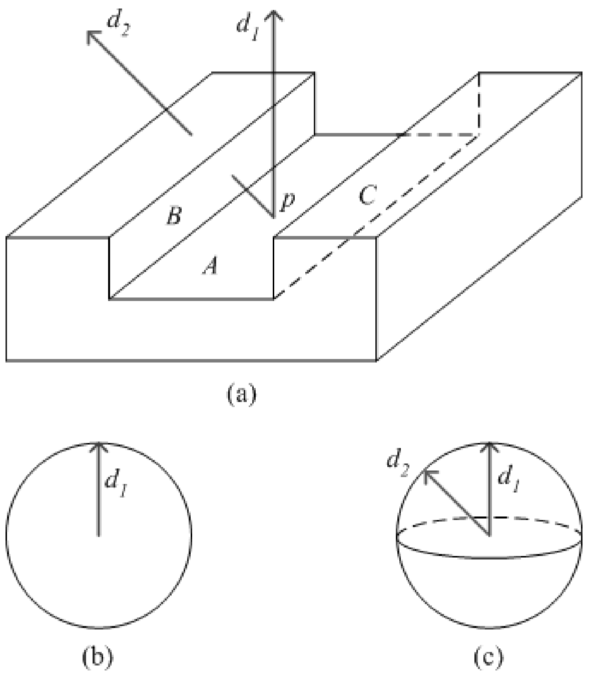


Figure 4.2: Visibility Map represents local visibility; (a) Faces A, B, and C form a concave region (b) Visibility Map of the concave region; (c) Visibility Map of face A

Coher [Elbe95] presented an approach to compute "visibility set" (a similar concept to visibility map) for freeform surfaces.

4.1.2 Global Accessibility Analysis

Suh and Kang developed an approach for performing accessibility analysis for NC machining [Suh95]. They compute accessibility by constructing the binary spherical maps. The part surface is faceted into triangular patches. The unit hemisphere is also faceted using spherical triangles. Accessibility is computed by projecting centroids of various facets on the unit sphere and identifying the spherical triangles that contain them. Due to approximations, this approach is prone to the following two types of errors. First, it might report that an entire facet is accessible in a certain spherical triangle while actually only a portion of the facet is accessible. Second, it might report that a facet is not accessible from an entire spherical triangle while actually the facet is accessible from a portion of the spherical triangle.

Recently, methods have been developed to perform accessibility analysis by taking advantage of computer graphics hardware [Bala00, Spit99]. Graphics cards make use of the depth-buffer implemented using hardware to perform fast hidden surface removal and render the object in a given scene. If all the individual faces on the object have been assigned different colors, then the accessibility of each face in a given direction can be detected by rendering the object using the given direction as the viewing direction, and querying the colors that appear on the pixel map after rendering. Since each rendering actually corresponds to a particular viewing direction, the point accessibility can be approximated by sampling a finite number of directions on the Gaussian Sphere. This approach involves two types of approximations. First, it uses finite sampling of viewing directions on Gaussian sphere. Second, it assumes that the face is so small that presence of a single pixel on the rendered scene can identify its accessibility. Therefore, the results produced by this approach are only an approximation of the exact solution.

Stage and Roberts described a framework for representing and computing tool accessibility from manufacturability evaluation point-of-view [Stag97]. This is primarily a feature-based approach, focusing on the shape/size compatibility between a tool and an entity (a face or a set of faces) to be machined. The advantage of this approach is that it works for an object with curved surfaces without any need for faceting. However, the notion of accessibility is closely tied with a particular tool.

Dhaliwal et al. [Dhal03] presented an algorithm for computing global accessibility cones for various facets of a polyhedral object. The paper describes exact mathematical conditions and the associated procedure for determining the set of directions from which a facet is inaccessible due to another facet on the object. By utilizing the procedure to compute the exact inaccessibility region for a facet, the paper presents an algorithm for computing global accessibility cones for various facets on the object. A unit sphere that represents all possible set of directions is divided into small spherical triangles. Global accessibility cone for a facet is the set of all spherical triangles from which the facet is accessible.

4.2 Two-Piece Mold Design

In the area of traditional two-piece molds, mold design problem has been studied mainly form two perspectives – determination of parting direction and determination of parting line and parting surface. Section 4.2.1 reviews the approaches for determining parting direction. Section 4.2.2 reviews the approaches for determining parting line and parting surface.

4.2.1 Determination of Parting Direction

In determining parting direction, most works consider demoldability as the primary factor in the determination. There are primarily two approaches used for determining parting direction: approaches based on accessibility analysis and approaches based on feature recognition.

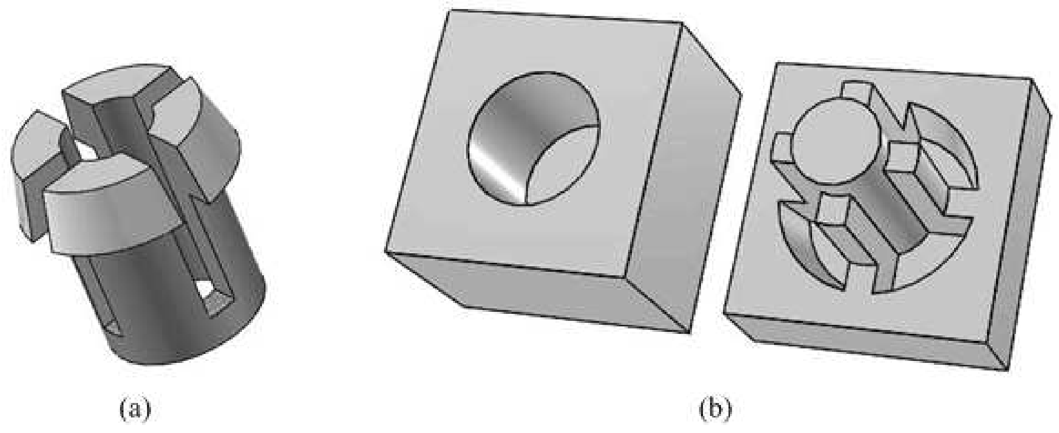


Figure 4.3: Cores can be avoided even if the visibility map is empty; (a) a part with one pocket; (b) two-piece mold for the part

Approaches based on Accessibility Analysis

Chen *et al.* [Chen93] formulated demoldability as a visibility problem. By computing the intersection of visibility maps of “pockets” on the part, the problem of finding the parting direction that minimizes the number of cores is transformed to finding a pair of antipodal points p and $-p$ that maximize the number of visibility maps which contain either p or $-p$. “Pockets” are non-convex regions on an object. Pockets on mold shape form basic elements of potential mold components. Based on this formulation, several other approaches have been presented for computing optimal parting directions of mold [Wein96, Vija98]. Pockets on an object can be generated by subtracting the object from its convex hull [Chen93], or by testing adjoining surfaces [Wein96].

This approach has the following limitations:

- It is assumed that a side core is inevitable for a pocket that has an empty visibility map. But sometimes, the side cores can be completely eliminated by subdividing the pocket. The part shown in Figure 4.3(a) has one pocket whose visibility map is empty. Figure 4.3(b) shows a valid mold design that is found by subdividing the pocket such that the two subdivisions are formed by different mold halves.

- It is also assumed that if the intersection of visibility maps of all the pockets is empty, side cores cannot be avoided. Subdividing the pocket and correctly attaching the subdivisions to different mold halves can again avoid the cores. Figure 4.4 shows one such example. The part shown in Figure 4.4(a) has five pockets, each of which has a valid draw range (non-empty visibility map). But the intersection of the visibility maps is empty i.e., there is no pair of antipodal directions along which all the pockets are completely visible. Figure 4.4(b) shows that a draw direction is possible for the decomposed pockets.

The above examples show that the attempt to form a whole pocket by a single mold piece fails. The pockets had to be decomposed irrespective of whether the pockets, had empty or non-empty visibility maps. And, as soon as a concave region is decomposed, the global nature of visibility maps is destroyed. Therefore, the approaches based on visibility maps cannot be applied to pocket subdivisions as it will not guarantee global accessibility and hence demoldability.

Hui and Tan [Hui92] heuristically generated a set of candidate parting directions that consisted of planar face normals and axis of cylindrical faces. Based on the observation that the face normals of the openings of the cavity solid (pocket in [Chen93]) determine a zone of possible directions for clearing the corresponding undercut, Hui [Hui97] added some more directions to the set of candidate parting directions by using normals to the cavity opening faces (lid faces in [Chen93]). He

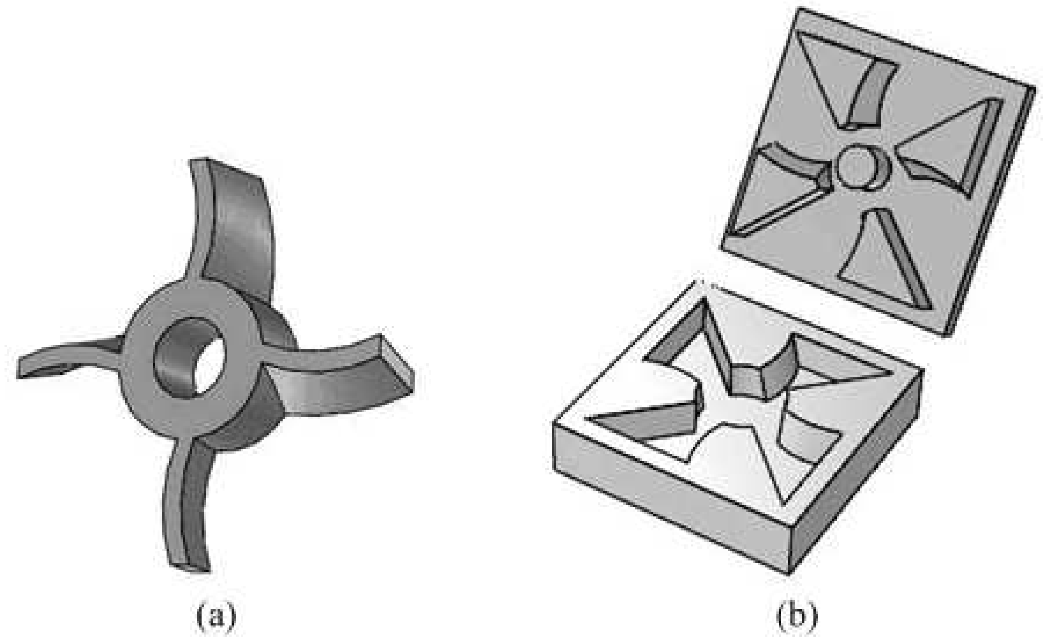


Figure 4.4: Cores can be avoided even if the intersection of visibility maps is empty; (a) a part with five pockets; (b) two-piece mold for the part

also developed a partitioning scheme to subdivide the pockets without destroying their global nature with respect to visibility. For a candidate parting direction $\vec{d}$, a pocket is partitioned by a series of planes each containing an edge of the pocket and vector parallel to $\vec{d}$. The elements obtained after partitioning are convex and are either completely blocked or cleared in $\vec{d}$. Figure 4.5 shows the proposed partitioning scheme. Each candidate parting direction is evaluated, and the direction requiring minimum number of side cores is chosen as the main parting direction. Though this approach provides a valuable method for partitioning the pockets without destroying the global nature of the visibility map, the heuristically found set of candidate parting directions may not be complete. For some very complex parts, it may not be able to find the sufficient set of directions required for the

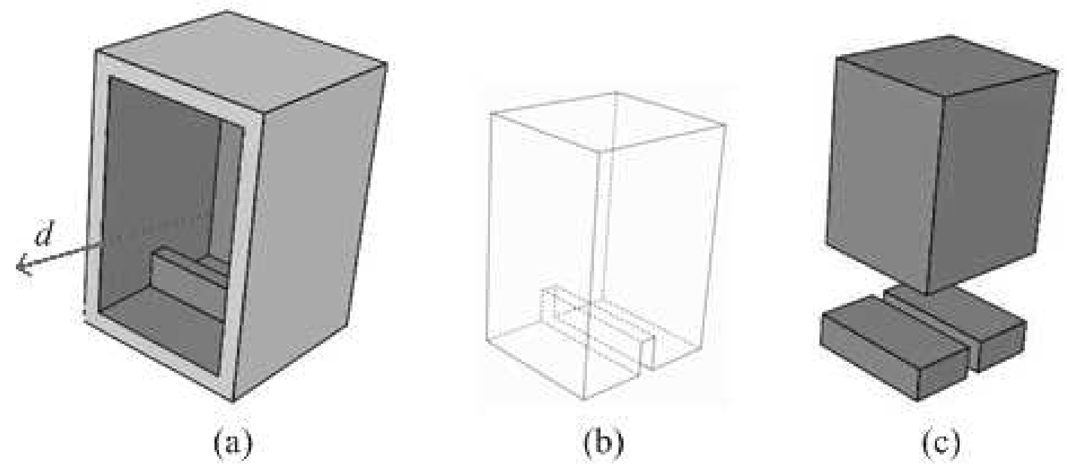


Figure 4.5: Partitioning of a pocket; (a) a part with one pocket; (b) pocket; (c) partitioning of the pocket

part.

Ahn et al. [Ahn02] presented an $O(n \log n)$ algorithm to test whether a polyhedral object is castable from a given direction. They also presented an algorithm to find all combinatorially distinct directions in which the object is castable. They represent every possible direction by a point on a unit sphere centered at the origin. They build an arrangement on vertices, edges, and cells of the object on the sphere. They prove that the vertices of this arrangement found by intersecting the curves on the unit sphere represent the complete set of distinct directions in which the object may be castable. There are $\Omega(n^4)$ distinct directions that can be computed in $O(n^4)$ time. So the total running time for finding a feasible parting direction takes $O(n^5 \log n)$ time. Building on this, Elber et al. [Elbe05] have developed an algorithm based on aspect graphs to solve the two-piece mold separability problem for general free-form shapes, represented by NURBS surfaces. McMains and Chen [Mcma04] have determined moldability and parting directions for polygons with curved (2D spline) edges.

The algorithm presented by Ahn et al. [Ahn02] has recently been implemented by Khardekar et al. [Khar05] using programmable GPUs. They describe a two-pass algorithm to determine moldability in a given direction. The camera is placed above the part along the given direction. In the first pass, the front faces of the part are rendered and the depth buffer is recorded. In the second pass, depth test is adjusted so that only the front faces invisible in the first pass are rendered. If there is any pixel rendered in the second pass, it means there are undercuts on the part. They further describe a method similar to Shadow Mapping to highlight the undercuts using shader programs.

Approaches based on Feature Recognition

These approaches make use of various feature recognition techniques to detect undercuts. A feature may be explicitly an undercut, or just a combination of certain types of surfaces. Each of the recognized features has its own set of possible parting directions due to accessibility considerations. These possible parting directions form a set of candidate parting directions and are evaluated using evaluation functions that measure goodness of a parting direction. It is a well-known fact that feature recognition is a difficult problem. This is especially so when the features interact with each other. Hence all the algorithms based on feature recognition techniques cannot handle arbitrarily complex parts.

Urabe and Wright [Urab97] select three principal coordinate directions as the candidate parting directions. Undercuts are recognized along each of these candidate directions. Number of undercuts, projected area, and number of cone surfaces for each of the candidate directions are calculated to determine the main parting direction. The complexity of the part shapes that this approach can handle is limited as stated in their paper.

Gu *et al.* [Gu99] developed a universal hint-based feature recognition algorithm to recognize features (holes, steps, pockets, protrusions, etc) on a part to be molded. Each type of feature has its own set of candidate parting directions. The optimal parting direction is the one with maximum value from an evaluation function.

Fu *et al.* [Fu99] developed an algorithm to recognize undercut features and classify them as Inside Internal Undercut, Outside Internal Undercut, Inside External Undercut, and Outside External Undercut. More recently, Yin *et al.* [Yin01] presented an approach of determining optimal parting direction by minimizing the number of undercuts among candidate parting directions. The mold components are also constructed based on the undercut features.

Lu and Lee [Lu00] presented an approach for analyzing interference elements and release direction for die casting and injection molding. A three-dimensional ray detection method was developed to recognize and extract the interference elements. Again, each recognized interference element has its own candidate release directions. The optimal release direction is computed to minimize the number of side cores.

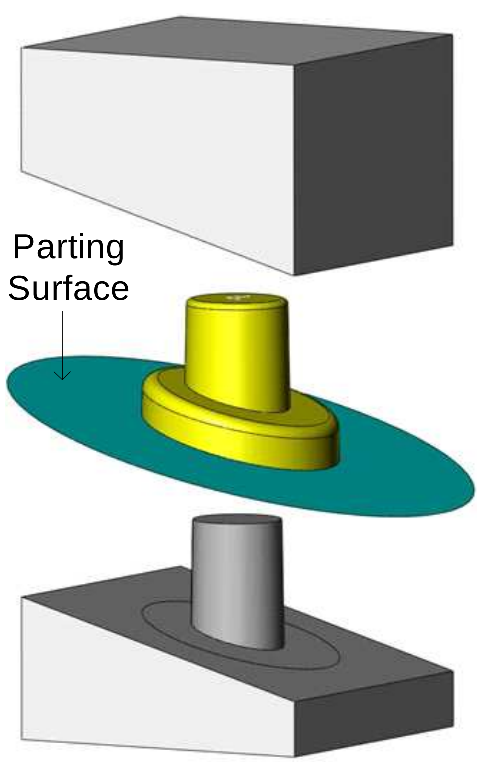
4.2.2 Determination of Parting Line and Parting Surface

In the approaches to determining parting line, either the parting direction is already set, or the parting direction does not cause any undercuts due to the convex shape of the object.

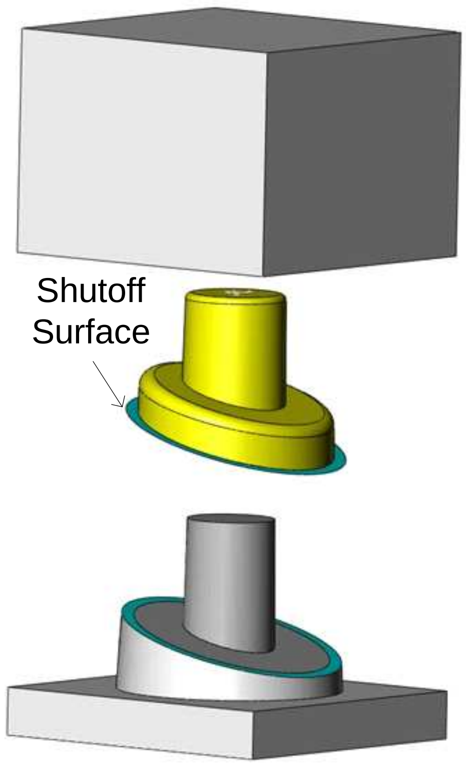
Ravi and Srinivasan [Ravi90] presented sectioning and silhouette methods for parting line generation. Wong *et al.* [Wong96] presented a slicing strategy for generating the parting line. Through a recursive uneven slicing method, several parting surfaces are generated for further evaluation. Weinsten and Manoochehri [Wein96, Wein97] formulated the parting line determination problem as an optimization problem. Their objective function is defined as a function of the flatness of the parting line, draw depth, number of side cores required to form the undercuts, machining complexity, etc. Majhi *et al.* [Majh99] presented an algorithm for computing an undercut-free parting line that is as flat as possible for a convex polyhedral object.

For non-planar parting line, Tan *et al.* [Tan88] presented a method to create the parting surface by extending the parting lines. The parting line has an outer loop and may have multiple inner loops. The outer loop is projected on a plane perpendicular to the parting direction and convex edges are identified. Each convex hull edge is projected to an adjacent side face of the mold enclosure. The projection direction is perpendicular to the parting direction and parallel to the surface normal of the mold face on which the edge is being projected. The gaps in the corners and inner loops are filled by triangulation. Nee *et al.* [Nee98] use a similar approach of extruding the parting lines to create the parting surface.

The limitations of the previous approaches are the following. They may not handle complex parts. The adjacent surface patches may intersect with each other or create undercuts. The produced surface may also not be optimal to machine. But most importantly, the previous approaches are trying to solve the wrong problem. The parting surface is not appropriate for complex 3D parting lines. The mold designers actually create what is called a *shutoff surface.* Figure 4.6 shows the difference between the parting surface and shutoff surface for an example part.



(a) Parting surface



(b) Shutoff surface

Figure 4.6: Difference between the parting surface and the shutoff surface

The parting surface or the shutoff surface is the contact surface between the mold pieces. They are machined with very high precision to minimize flash. The surface area of the parting surface is larger than that of the shutoff surface as shown in the figure. Machining such a large surface with very high precision is very expensive.

4.3 Multi-Piece Mold Design

Although multi-piece molds can fabricate much more complex parts than two-piece molds, there are very few papers in this area. There are two types of multi-piece molds – sacrificial and permanent. The sacrificial mold pieces are manufacturable but may not be disassemblable. They need to be destroyed before ejecting the molded part. The permanent mold pieces are both manufacturable and disassemblable.

4.3.1 Sacrificial Mold Design

Dhaliwal et al. [Dhal01] presented a feature-based approach to solving the problem of automated design of multi-piece sacrificial molds. For those objects whose geometry can be represented by their feature-based representation, the approach provides a 3D spatial partitioning scheme to computationally efficiently solve the mold design problem. However, this approach cannot be used to design molds for arbitrarily complex parts.

Huang and Gupta [Huan02] describe an algorithm based on accessibility-driven partitioning approach to design multi-piece sacrificial molds. Gross shape of the mold is constructed by subtracting the part model from the mold enclosure. Accessibility analysis of the gross mold is performed. The gross mold is partitioned using the accessibility information. Each partitioning improves accessibility and a set of mold components is produced. Each mold component is accessible and therefore can be produced using milling and drilling operations.

4.3.2 Permanent Mold Design

Krishnan and Magrab [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-molded 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 direction is restricted to the X and Y direction.

This approach has several limitations. Since the primitives considered are only 2.5D solids that are stacked along the Z direction, the complexity of the part is limited in this approach. Moreover, since the mold separation directions are constrained to be along either the X-axis direction or the Y-axis direction, it is not always guaranteed to find a solution.

Chen and Rosen [Chen01a, Chen01b] subdivide multi-piece mold design process into two subsequent processes: Mold Configuration Design Process and Mold Construction Process.

1. *Mold Configuration Design Process*: In this step, object boundary is subdivided into smaller regions that will be formed by different mold pieces. Parting direction is found for each mold piece region by solving a linear optimization problem. If a combined region (pockets) does not have a parting direction, it is split into concave regions (all internal edges concave) and convex faces (all face edges convex). The plane of a face containing an internal convex edge is used to split the region. If the region does not have any internal convex edge, it is not split. It is assumed that the part is non-moldable.

2. *Mold Construction Process* An approach based on Reverse Glue Operation is developed to produce a two-piece mold. Glue faces and planar parting surfaces are used to split the core and cavity. Selecting different glue faces and parting planes produce different mold pieces. This algorithm for two-piece molds is used recursively to produce a multi-piece mold.

This being the first multi-piece mold design approach seen in literature that allows three-dimensional mold decomposition, provides an excellent groundwork to improve upon. One of the areas that requires immediate attention is in the determination of parting directions. Since a local approach has been followed to find the parting direction of a region, disassembly of mold pieces is not guaranteed. A separate interference test simulation module is required to verify the mold design generated by this approach. If the mold design is found to be incorrect, the whole process has to be repeated again and again. There may even be cases when this approach fails to produce a feasible solution. Figure 4.7 shows one such example. Since the concave region R does not have a parting direction, it needs to be split into concave regions and convex faces. But R does not have any convex edge along which it can be split. It can be seen that the proposed region splitting approach fails to produce a feasible solution in this case. A robust implementation of region splitting algorithm is also challenging because splitting a face in some cases, may produce slivers. Slivers have two vertices so close to each other that it becomes difficult to do correct vertex classification.

Priyadarshi and Gupta [Gupt03, Priy04] developed geometric algorithms for completely automated design of multi-piece molds. Given the CAD model of a part and the mold enclosure dimensions, they presented a novel five-step approach called Multi-Piece Mold Design Algorithm (MPMDA) to generate a multi-piece

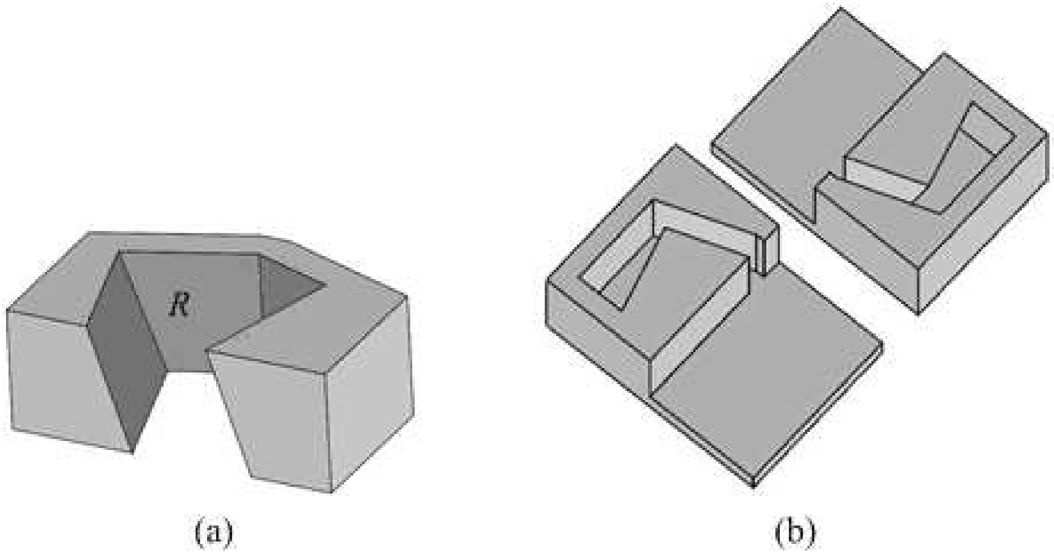


Figure 4.7: Concave regions having no parting direction and no internal convex edge are also moldable; (a) a part with a concave region that has no parting direction and all concave internal edges; (b) two-piece mold for the part

mold design for the part.

1. A heuristic set of candidate parting directions D is first generated based on the geometry of the part.

2. For each direction $\vec{d}$ in D, ray-accessibility of every facet on the part is checked. For the part to be moldable, every facet on the part needs to be ray-accessible from at least one direction. If a facet is accessible from none of the directions in D, accessibility analysis of the facet is performed to compute accessibility cone of the facet. If the accessibility cone is non-empty a direction is chosen from the accessibility cone and included in D. Otherwise, the part is discarded as non-moldable. For each candidate parting direction, accessible-facet sets are found.

3. Using the set of accessible-facet sets, the part boundary is then divided into different mold-piece regions.

4. Out of all mold-piece regions, minimum number of mold-piece regions is selected such that the entire part boundary is covered. This is equivalent to solving set-cover problem.

5. After minimum number of mold-piece regions has been identified, mold pieces are finally constructed.

4.4 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 novel features of their algorithm are as follows.

- It finds multiple partitioning planes to perform partitioning of the mold-pieces.
- It performs object and mold decomposition needed to ensure the assembly and disassembly of mold-pieces during mold-stage assembly.
- It generates the complete molding sequence of the multi-stage molds. The algorithm specifies the mold-pieces that should be added and removed from the previous stage to produce the mold assembly at each stage.

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, Gupt02] 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 novel features of their algorithm are as follows.

- The algorithm can handle curved contact surface. The types of curved surfaces is however limited to spherical and conical.
- The algorithm partitions the gross mold by curved analytical surfaces and combine the resulting solids to form the final mold pieces. This ensures that the contact surfaces between two mold pieces is perfect and does not leak.
- The disassemblability of the generated mold pieces is guaranteed.

The limitations of their algorithm are as follows:

- This algorithm uses a simple extension of Tans algorithm [Tan88] for finding parting lines. Hence it cannot handle complex parting lines.
- Only cylindrical undercut features are handled by this algorithm.
- The input object cannot have more that two materials and two lumps.
- The mold pieces generated by the algorithm may not have optimal geometric shape.
- Material-based precendece constraints are not incorporated.
- Configuration of articulated assemblies is not changed after every molding stage.

4.5 Assembly Sequence Planning

Automatic assembly sequence planning has been an important research topic for researchers in geometric modeling, robotics and artificial intelligence. The process of assembling component parts to make a final product and disassembling a final part into its components are two different but similar processes in a real world. The process of assembly can be considered as traversing in an ordered list of assembly operations. An assembly operation can be defined as going from one spatial configuration of components to another spatial configuration of components by moving one or more components. 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 is a high-level plan for constructing a product from its component parts. It specifies which sets of parts form subassemblies, 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. A variety of automated systems have been designed to generate such assembly sequence plans. While excellent progress has been made in developing methods to quickly find a geometrically feasible plan for a product, there have been both definitional and computational problems with finding good assembly plans.

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.

4.6 Hybrid Process Planning

Process planning translates design information into process steps and instructions to efficiently manufacture products. Automated computer-sided process planning has evolved as an important element in integrating design and manufacturing [Alti89]. There are two traditional approaches to process planning. The first approach is called *generative* process planning. In this approach a plan is synthesized from the first principles by trying various alternatives in generate-and-test paradigm. The second approach is called *variant* process planning. In this approach, a plan is generated by modifying an existing plan.

Elison et al. [Elis97] proposed a hybrid approach that combined the characteristics of both variant and generative process planning. They built a variant database of designs and process plans classified using *design signatures.* When a process plan is needed for a new design, *slices* of plans are retrieved from the database built earlier. These plan slices are combined in a generative manner to produce a new plan. Balasubramanian et al. [Bala98] proposed another hybrid approach that used a generative approach for process selection and then a variant procedure to select fixtures.

4.7 Summary

The previously published algorithms for mold design are not adequate for designing multi-stage molds for articulated assemblies. The algorithms to find mold-piece regions are most importantly not robust. They produce inconsistent results in the presence of near-vertical facets. There is also a need to increase the efficiency to be able to handle high resolution parts. The algorithms for creating parting line and parting surface cannot handle very complex shapes that are quite common in case of multi-stage molds. With the increase in complexity of the parting lines and parting surfaces, there is an increase in the need to automatically optimize them. This is missing from the current literature.

Chapter 5

REPRESENTING REUSABLE MOLDING PLANS FOR ARTICULATED JOINTS

This chapter presents a framework for representing reusable molding plans for articulated joints. This representation is used to build a library containing reusable molding plans for common types of joints. This library is used by the algorithm described in Chapter 6 to generate molding plans for articulated assemblies. The material presented in this chapter is expanded version of the material published in [Priy06a].

This chapter is organized in the following manner. Section 5.1 describes where the joint molding plans fit into the overall approach. Section 5.2 describes a set of basic assembly design principles that if followed may lead to feasible and efficient molding plans. Section 5.3 describes the framework for representing reusable molding plans. Sections 5.4 – 5.6 present molding plans for three basic joints – prismatic, revolute, and spherical. Section 5.7 finally summarizes this chapter.

5.1 Introduction

Generating molding plan is similar to other manufacturing operation-planning problems such as machining and sheet-metal bending. There are two traditional approaches to process planning. The first approach is called *generative* process planning. In this approach a plan is synthesized from the first principles by trying various alternatives in generate-and-test paradigm. The second approach is called *variant* process planning. In this approach, a plan is generated by modifying an existing plan. Purely generative approaches are unlikely to work in the molding planning domain. A lot of knowledge that is needed to successfully mold joints does not exist in an explicit geometric form. Purely variant approaches are also unlikely to work because every new assembly is significantly different from the previously generated assemblies.

Variant approaches, can however be applied to developing molding plans for individual joints. The various types of joints used are very few. So it is very common to find similar joints in new assemblies. We use a hybrid approach that combines elements from generative and variant techniques. We reuse the molding plans for individual joints to generate plans for new assemblies. When a new assembly is encountered we first check if the joints used in the assembly are sufficiently similar to the joints for which plans exist. If such plans are available, they are reused in a generative manner to develop the molding plan for the assembly.

Reusing molding plans for joints allows us to reuse existing molding knowledge and yet ensure that we can handle a wide variety of molding planning problems. When a successful plan is developed and experimentally validated for molding a joint, a lot of useful knowledge is generated. There is a great value of design knowledge in engineering. Experienced designers make good designs. Meanwhile novices are overwhelmed by the design requirements. Experienced designers evidently know something that inexperienced ones do not. One thing experienced designers know not to do is solve every problem from first principles. Rather, they reuse solutions that have worked for them in the past. When they find a good solution, they use it again and again. Such experience is part of what makes them experts.

Knowledge and experience is especially important in mold design. Developing a molding plan is a hard problem. It requires significant amount of time, effort, and expertise. There are often concerns about the feasibility of a molding plan because many decisions are based on subjective guesswork. The desired articulation and multiple molding stages introduce geometric constraints, which if violated, results in poor quality, longer molding cycles, and high tooling cost.

5.2 Basic Assembly Design Principles for Achieving Feasible and Efficient Plans

Currently a systematic approach to designing articulated in-mold assembled products does not exist. It is an iterative process and successful implementation mainly depends on the designer's experience. There is also very little published literature available in the public domain. In this section we identify a set of basic assembly design principles that lead to feasible and efficient molding plans, which in turn result in high performance and reduced overall cost. These design principles are mainly for selecting the right geometry and material for assembly components. They are based on interactions with experienced mold designers and experiments in our laboratory. The designers populating the library of reusable molding plans

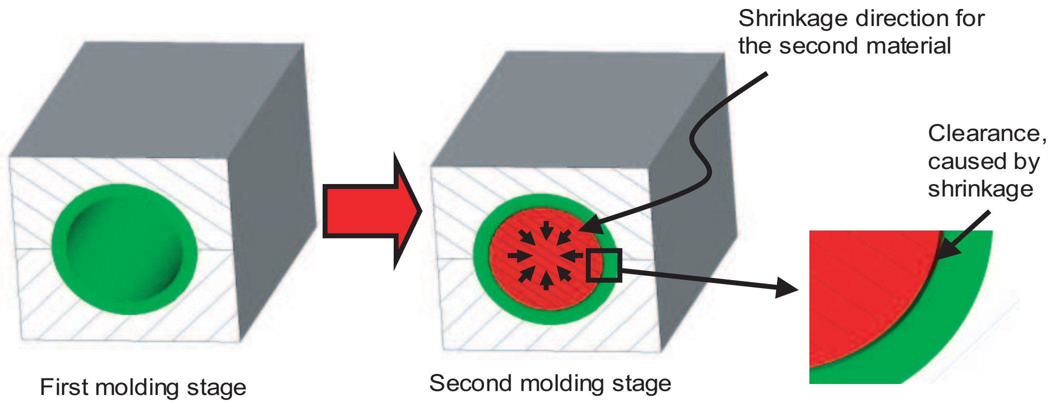

Shrinkage direction for the second material
Clearance, caused by shrinkage
First molding stage
Second molding stage

Figure 5.1: Effect of Shrinkage on Joint Clearance

must follow these principles when designing joints. This will ensure that only feasible and efficient plans are added to the library.

5.2.1 Achieving Proper Joint Clearances

The molded parts shrink as they solidify. This shrinkage directly affects the amount of clearance in a molded joint as illustrated in Figure 5.1. Estimating shrinkage for molded parts is a challenging problem in the molding community. This problem is extremely important in articulated assemblies because all the components have to fit together and work. Uncompensated shrinkage in even one component can cause the assembly to have excessive or insufficient clearances. Excessive clearance causes poor kinematic performance, while insufficient clearance causes geometric locking. This section describes a model for designing articulated joints and molding process so that the final product can meet the performance goals. We outline a systematic approach that will help a designer to determine component dimensions, material properties, and molding parameters.

Shrinkage is defined as the difference in dimensions of a molded part and the

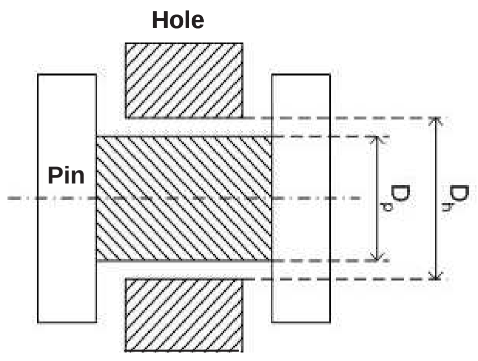


Figure 5.2: Shrinkage Analysis for Revolute Joint

mold cavity at room temperature. Uncompensated shrinkage leads to either sink marks or voids in the molding interior. Shrinkage of individual components can stackup or combine together to affect a final assembly dimension. We describe shrinkage analysis, similar to the assembly tolerance analysis to help predict the final assembly dimensions and hence the success of an in-mold assembly. For the success of an assembly, there are some critical dimensions that need to be controlled. A critical assembly dimension Y_s is a function of the component feature dimensions x_i, and is given by:

$$Y_s = f(x_1, \ldots, x_n) \tag{5.1}$$

This function between the assembly and component dimensions is known as assembly function or stackup function and must be derived for each assembly. Shrinkage analysis for assemblies is concerned with determining the critical assembly dimension Y_s. This analysis will be illustrated with the help of a simple revolute joint shown in Figure 5.2. The clearance between the pin and the hole is a critical dimension and is given by:

$$C = D_h - D_p \tag{5.2}$$

where D_h and D_p are the hole and pin diameters respectively. The component dimensions D_h and D_p are obtained after shrinkage. Because of the dynamic nature of the molding process, estimating shrinkage is not straightforward. However hard we try, some uncertainty will always be present. So we represent the component dimensions as interval numbers. For example, a dimension D will be represented as:

$$D = [D_{max}, D_{min}] \tag{5.3}$$

The corresponding uncertainty in the clearance C can be calculated as follows:

$$\begin{aligned} C_{max} &= D_h^{max} - D_p^{min} && (5.4) \\ C_{min} &= D_h^{min} - D_p^{max} && (5.5) \\ \Delta C &= C_{max} - C_{min} \\ &= \Delta D_h + \Delta D_p && (5.6) \end{aligned}$$

The next step in shrinkage analysis is to understand the shrinkage of individual dimensions. Shrinkage of a dimension can be mathematically expressed as:

$$S = \alpha.D \tag{5.7}$$

where D is the original dimension and α is the shrinkage coefficient. The shrinkage coefficient α is dependent on material properties and other process parameters such as melt temperature, mold temperature, injection pressure and hold time. This parameter is usually experimentally determined for a given material and process parameters by creating a test specimen with the volume and section thickness comparable to the desired part. [Jans98] presented a systematic study on the effect of processing conditions on mold shrinkage. They concluded that the shrinkage of injection molded products is mostly influenced by the holding pressure and the melt temperature. [Post05] recently published experimental results on assessment of the effects of process parameters on shrinkage. These studies can also be utilized to estimate the shrinkage coefficient for a given material and process parameters. After shrinkage, the dimension of a molded part D_p is given by:

$$D_p = (1 - \alpha) D_p^m \tag{5.8}$$

where D_p^m is the mold dimension. The uncertainty in the molded part dimension can be estimated as:

$$\Delta D_p = \Delta\alpha . D_p^m \tag{5.9}$$

where $\Delta\alpha$ is the uncertainty in α, which can be controlled by changing molding parameters such as injection pressure, pack-hold time or cooling time, and holding pressure. The uncertainty in the mold dimension due to machining is negligible. Therefore, these terms are omitted from further consideration. From the above equation we get the following governing equations:

$$\begin{aligned} C &= (1 - \alpha_h) D_h^m - (1 - \alpha_p) D_p^m && (5.10) \\ \Delta C &= \Delta\alpha_h D_h^m + \Delta\alpha_p D_p^m && (5.11) \end{aligned}$$

where D_h^m and D_p^m are the mold dimensions for the hole and pin respectively. Usually C and ΔC are given based on design goals. It is the job of the designer to determine the values for α_h, α_p, $\Delta\alpha_h$, $\Delta\alpha_p$, D_h^m, and D_p^m that meet the given values of C and ΔC. As is obvious from Equations 4.10 and 4.11, the number of variables is more than the number of equations. Therefore this problem is under-constrained. These equations can be solved by assigning fixed values to some variables and solving for the remaining variables. The above equations have been derived for a revolute joint shown in Figure 5.2. Similar types of equations can be derived for other types of joints as well. The above model is intended as a useful guideline for designing articulated joints and molding process. It usually provides a first order approximation. In many cases, shrinkage is not linear and constant in different directions. It is measured along three directions – direction of flow, cross-flow direction, direction of part thickness. These often may have different magnitudes leading to uneven volumetric shrinkage along the three directions. Moreover, in case of complex joints, it is very difficult to derive the assembly function. In such cases, shrinkage effects must be determined via empirical experiments.

Our experiments indicate that the following guidelines would be useful in controlling shrinkage in multi-stage molding process:

1. The outer components, which envelop the other components, must be cast before the inner components are molded. This will cause the last stage to shrink away from its preceding stage and establish clearance between the components. If the stage order is reversed, the final mold stage will tend to shrink onto the interior parts resulting in a friction increase at the interface, in turn restricting the relative motion.

2. Since shrinkage is directly proportional to component dimension, the above criterion may be ignored for small parts.

3. Pack-hold time or cooling time should be increased to reduce shrinkage.

4. Shrinkage can also be reduced by controlling injection pressure and melt temperature.

5.2.2 Preventing Adhesion at Joint Interfaces

During the multi-stage molding process, when the second material is injected on top of the already molded material, the two materials tend to adhere to each other.

To created articulated devices, we need to prevent the adhesion at the interfaces so that we can create free moving articulated devices. Based on the published research and our own tests, the following process and material parameters affect adhesion at the multi-material interfaces:

1. *Cross-linking of polymers*: This dictates the strength of bonding on molecular scale.
2. *Mold pressure*: Higher pressure enhances intimacy of microscale contact at interface.
3. *Curing temperature*: This parameter controls crosslinking at the interface.
4. *Anti-stiction agents*: These agents inhibit adhesion at the interface.
5. *Surface roughness of substrate*: Higher surface roughness enhances mechanical interlocking at microscale.
6. *Shrinkage stress*: Shrinkage induced stresses generate forces to separate interface.

Based on our experiments the best way to prevent adhesion is to select materials that are chemically incompatible with each other and hence do not promote cross-linking of polymers. The number of available materials for molding is quite large, resulting in countless material combinations possible for multi-material molding. Unfortunately, the adhesion quality for all combinations of materials is not known. In many cases, this has to be determined experimentally for the given component configuration and processing conditions. Some research has been conducted on the compatibility of various materials and it is observed that polymers of similar nature adhere to each other well. The material combinations that do not adhere well and are good for articulated joints include Acrylonitrile-Butadiene-Styrene (ABS) and Polyvinyl Carbonate (PVC), acrylic-styrene-acrylonitrile (ASA) and Polystyrene (PS).

An additional rule to material selection is that the melting temperature of the first molded material must be much greater than the melting temperature of the material being used in the second molding stage. If this rule is not followed, the second material will melt the first one and the two will stick with each other. The same general rule holds true for all subsequent molding stages. The designer must match up material properties with molding stages as well as product specifications.

Apart from selecting the right materials, increasing the curing time between subsequent mold stages also helps prevent adhesion problems. There are also mold-release sprays (e.g., Silicone Mold Release from Huron Technologies, Inc.) available in the market that when applied to material interfaces, prevent adhesion.

5.2.3 Minimizing the Number of Molding Stages

Usually a separate molding stage is required for each component. But in some cases, the mold sequence can be specified in such a way that the assembly can be molded in lesser number of stages. For example, a product with three components has six different possible sequences. One way to minimize manufacturing time is by using a mold-staging strategy that involves injecting as many components in one sequence as possible. For example, consider the two-material object illustrated in Figure 5.3. Because components A and C never touch and are of the same material, component B can be injected in the first stage and then A and C can both be injected in the second stage. It is important to specify an appropriate sequence of stages so that the product can be manufactured in the least number

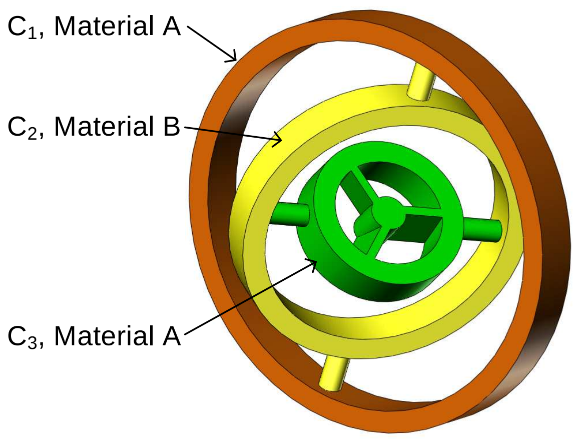


Figure 5.3: Three components can be injected in two stages

of stages.

5.2.4 Simplifying the Method for Changing Cavity Shape

To carry out multi-stage molding, the cavity shape needs to change after every molding stage. The first stage starts with the first stage material being injected into an empty cavity. The material fills the cavity completely and solidifies. Before starting the second stage molding, the cavity shape needs to be altered to create room for injecting the second stage material. This step requires changing the shape of the original cavity. The cavity shape should be changed while satisfying the assembly and disassembly constraints imposed on the mold pieces. Therefore, different types of geometries require different cavity shape change methods. Following are the different ways cavity shape can be changed in the increasing order of complexity:

1. One or more mold pieces can simply be moved away from the first stage material in the cavity and hence can expand the cavity (Figure 5.4(a)).

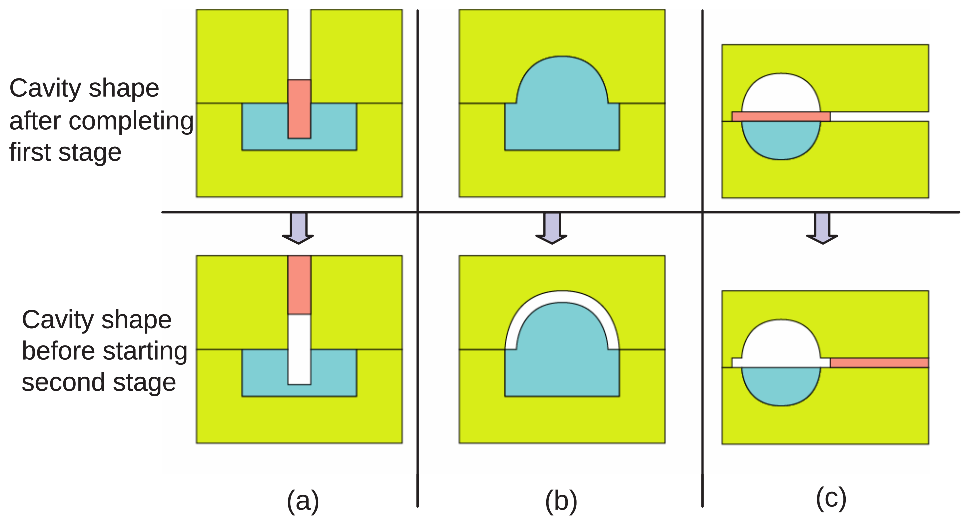

Cavity shape after completing first stage
Cavity shape before starting second stage
(a)
(b)
(c)

Figure 5.4: Changing mold cavity shape between stages

2. One or more mold pieces in the initial cavity can be swapped with a mold piece with a different shape (Figure 5.4(b)).

3. Partitions can be added in the initial cavity and then removed during subsequent stages (Figure 5.4(c)).

4. Molded part can be completely transferred to another mold with a different cavity shape. This method is called cavity transfer.

The first three techniques alter the cavity shape without moving the already molded shape. Only a few mold pieces around the molded shape are moved. The first technique is desirable than the second and third technique because it is easier and faster to just move the mold pieces. The last cavity transfer technique is one of the easiest to design a mold for, but is also quite challenging to implement into mass production process. This is due to the fact that molded parts must be manually or robotically manipulated between mold stages. Orientation of the components between stages is also difficult when transferring it. It can be seen that shape-change method mainly depends on the complexity of component interfaces. If it is a simple hole, the first technique can be employed. But in the case of complex interfaces, the cavity transfer technique needs to be employed, or the mold design becomes too complicated. Hence a simple interface should be designed between two components.

5.3 Framework for Representing Joint Molding Plans

The purpose of developing a representation for molding plans is to record them in a reusable form. To effectively reuse a molding plan for a new assembly, the first thing that we need to determine is whether that molding plan is applicable. The molding plan for the new assembly will be developed using the molding plans for each joint in the assembly. The molding plan for the whole assembly needs to satisfy the constraints of each joint molding plan to be feasible. So the next thing that we need to know is the set of feasibility constraints that needs be to added to the overall molding problem for the new assembly. It is important to compile this information in a consistent format so that the molding plans are easy to classify and search.

Section 5.3.1 describes the necessary conditions that a joint in a new assembly needs to satisfy before a molding plan can be applied. Section 5.3.2 enumerates the relevant feasibility constraints imposed by a joint molding plan. Section 5.3.3 describes a comprehensive format for representing a molding plan.

5.3.1 Applicability Conditions

We can reuse a molding plan only if it is applicable. So a representation for molding plans should contain information about the situations in which a particular molding plan can be applied. The applicability conditions for a molding plan should contain all critical parameters that affect the molding plan. But at the same time, it should not contain irrelevant parameters that does not directly the molding plan. These irrelevant parameters unnecessarily reduce the applicability of the molding plan.

From Section 5.2, we know that the geometry and material of a joint are important factors in developing a molding plan for the joint. They directly affect the obtained joint clearance and play a significant role in determining the feasibility and efficiency of the generated molding plan. Hence it is reasonable to assume that if the geometry and material of two joints match, the molding plan for one joint can be used for the other. The molding plan must contain all combinations of materials that can be used and the following parameters that define the geometry of a joint:

1. Type of joint: There are several standard joints (prismatic, revolute, spherical, universal, etc.) used in articulated assemblies. The standard joint are represented by their name, while the non-standard ones are represented graphically or by a CAD model.

2. Size of joint: The plan must provide the range of joint sizes on which it can be used.

3. Where the joint components can be extended: The molding plan provides a graphical representation of the joint. The regions where the joint components can be extended is marked.

5.3.2 Feasibility Constraints

Based on the design principles described in Section 5.2, we can identify certain constraints that need to be satisfied for a molding plan to be feasible.

1. Precedence constraint: As described in Section 5.2.1, the outer components should be cast before the inner components. This requirement is however not important for small dimensions. But in case of large dimensions, the molding plan must contain this constraint. The precedence constraint is represented by a partially ordered set. A partially ordered set (or poset) is a set taken together with a partial order on it. Formally, a partially ordered set is defined as an ordered pair $P = (X, \leq)$, where X is called the ground set of P and $\leq$ is the partial order of P [Insa04]. In our case, $\leq$ represents the sequence in which components need to be molded. As example, consider $X = \{a, b, c\}$. The relation $a \leq b$ and $a \leq c$ states that a must be molded before b and c.

2. Joint-Axes Constraints: The joints should be molded in a configuration such that parting line does lie on the joint interfaces. Flash is usually formed along the parting line. If the parting line lies on the joint interface, the flash between the connected components will cause jerky motion. In the worst case, it can even interlock the components and prevent any relative motion. Another factor that influences the molding configuration of a joint is the direction of removing side actions. It is usually economical to remove the side actions in a direction perpendicular to the parting direction.

   Due to these performance and cost reasons, the molding plan for a joint must specify a configuration space of the joint axes. The configuration space for each axis of the joint is represented as a *spherical polygon* on a *Gaussian*

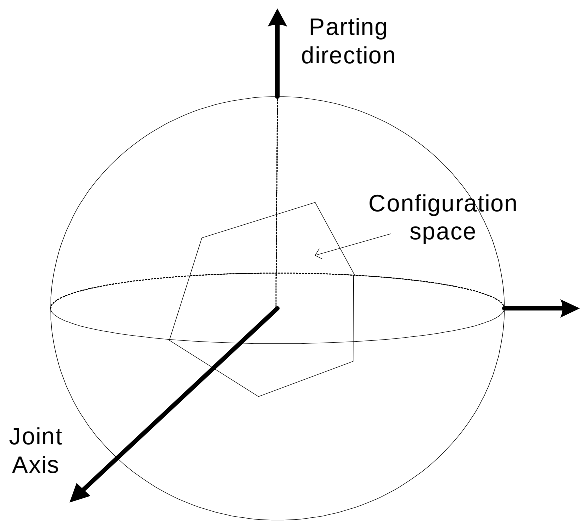


Figure 5.5: Feasible configuration space of a joint axis

sphere, which is a unit sphere centered at the origin such that every point on it defines a direction in Euclidean 3-space. A spherical polygon is a portion of the surface of a unit sphere that is bounded by the arcs of great circles. A great circle is the intersection of a sphere with a plane going through its center. We define the configuration of the joint axes with respect to the parting direction. Without the loss of generality, we assume that the parting direction is along the z-direction (north pole on the Gaussian sphere). Figure 5.5 shows an example configuration space for a joint axis.

3. Joint-Parameter Constraints: The geometry of a joint sometimes imposes a constraint on joint parameters. For a joint, there may be certain values of a parameter in which molding cannot take place. Figure 5.6 shows an example where the joint is valid only for a finite range of joint parameter θ. In such cases, the molding plan must specify a a feasible range of joint parameters. A range is represented by a tuple $[\theta_l, \theta_u]$ where θ_l is the lower bound and θ_u

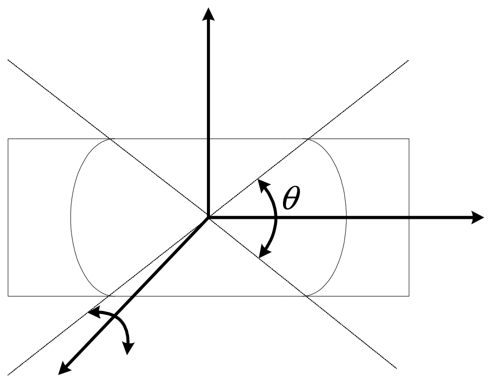


Figure 5.6: Feasible range of a joint parameter

is the upper bound of the range.

5.3.3 Representation Format

A common vocabulary or language is fundamental to expressing the concepts of any engineering discipline. We too need a consistent format for representing the molding plans in order to provide a communication platform for sharing molding plans for commonly encountered articulated joints. Forming a common description format for conveying the problems and proposed solutions allows us to capture the body of knowledge and intelligibly reason about them. The requirements for a representation are the following:

1. It should be flexible enough to be easily adapted. The molding plans for joints are not intended to be directly used. For a given assembly, they need to be merged with molding plans for other joints in the assembly and modified according to the geometry of assembly components. So the representation should only present the core of the plan in such a way that it can be adapted to different scenarios.

2. It should contain all information important for developing a feasible plan for a joint.

3. It should not contain irrelevant information that reduces applicability and over-constrains the design space.

4. It should be easy to classify so that the molding plans can be stored in a library and retrieved.

This dissertation proposes the following format to describe a molding plan which satisfies the requirements outlined above. A molding plan for joints contains four pieces of information:

1. *Applicability* to easily identify the plan applicable to a certain problem
   - (a) Type of joint
   - (b) Size of joint
   - (c) Where the joint components can be extended
   - (d) Material
2. *Solution*
   - (a) Number of molding stages
   - (b) Molding sequence (includes explicit precedence constraints)
   - (c) Molding configuration (includes explicit joint axes and parameter constraints)
   - (d) Method for changing mold-cavity shape change
3. *Example* to better understand the solution

4. *Consequences* to describe the positive and negative aspects of the solution

It must be noted that the 'Solution' section of the representation includes explicit feasibility constraints of a molding plan. The molding sequence is the precedence constraint. The molding configuration includes the joint axes and parameter constraints. It gives the orientation of the joint axes with respect to the parting direction and the range of joint parameters.

5.4 Molding Plans for Prismatic Joint

Prismatic joints are seen in a wide variety of assembled objects. Applications range from the well-known slider-crank mechanisms, used in the cylinder of an internal combustion engine, to more recent mechanisms found in CD and DVD drives. The basic criterion to define a prismatic mechanism is that it restricts all rotational motion and allows the object to translate in one direction. This section presents two design plans for realizing prismatic joints. Both plans employ planar contact surface to constrain rotation and facilitate translation along one direction. The difference is in the geometry of the parts that form the prismatic joint.

5.4.1 Plan A

Applicability

This molding plan is for prismatic joints where the handle completely envelops the slide. Figure 5.7 shows regions where the joint components can be extended.

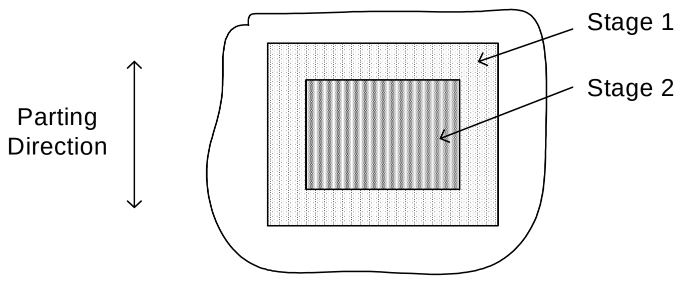


Figure 5.7: Prismatic joint plan A

Solution

This plan requires two mold stages. The handle is molded in the first stage. The slide is molded in the second stage. The parting direction is perpendicular to the translation axis. The cavity shape is changed using a sliding core. The core acts as a placeholder for the slide in the first stage. It is pushed out of the mold in the second stage creating a cavity into which the material for the slide is injected.

Example

Figure 5.8 shows an example for this plan. It requires two mold pieces and a side core. Figure 5.8(b) and 5.8(c) show the two mold stages for molding the prismatic joint. In the first mold stage, Mold piece A and B are assembled and the core is inserted to the position shown in Figure 5.8(b). The first material is then injected into the assembled mold. Figure 5.8(c) shows the molded part after the first stage. For the second mold stage, the core pulled out to the position shown in Figure 5.8(c). The second material is then injected to produce the final part.

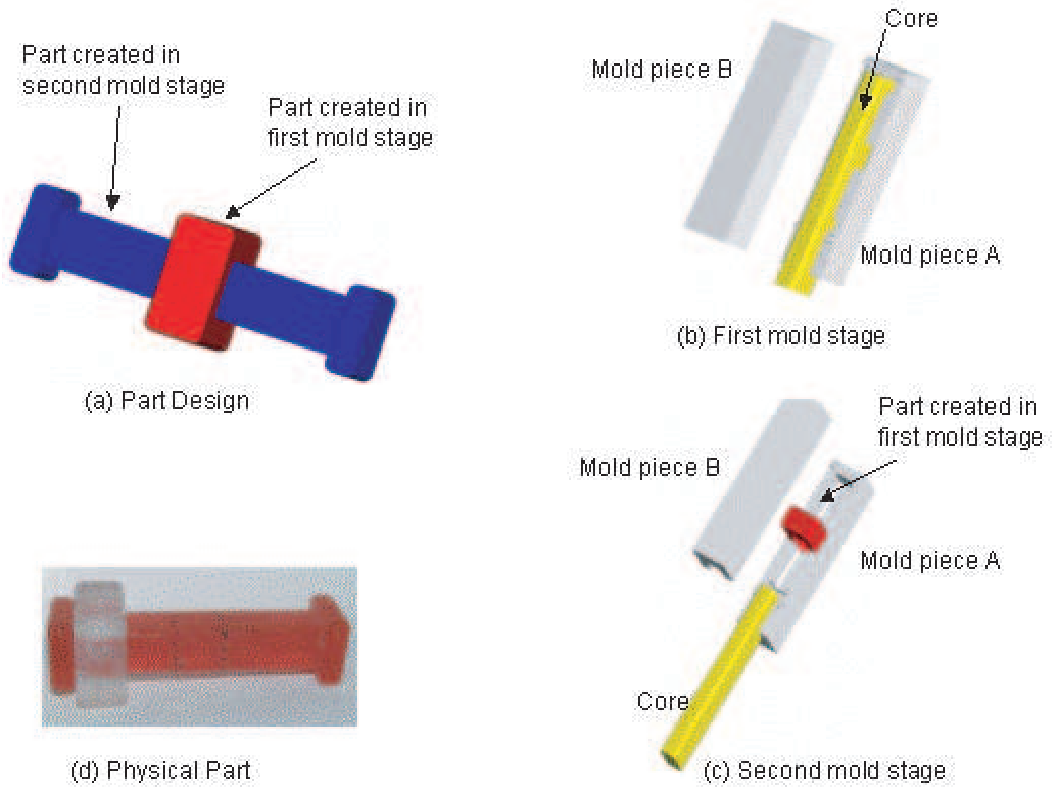


Figure 5.8: Example for prismatic joint plan A

Consequences

This plan follow the clearance principles outlined in Section 5.2.1 of molding the inner component after the outer component. This will cause the inner component (slide) to shrink away from the outer component (handle) establishing clearance between the contact surfaces. It is a very simple plan but restrictive. The geometry of the part containing the slide can only be modified at the ends.

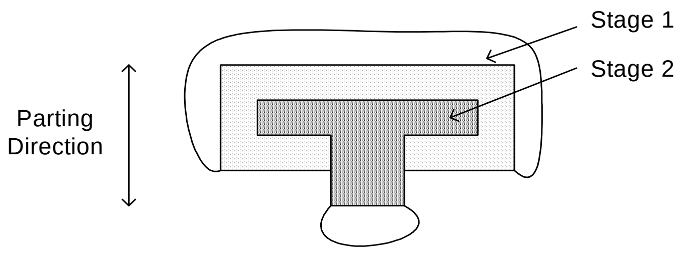


Figure 5.9: Prismatic joint plan B

5.4.2 Plan B

Applicability

This molding plan is for prismatic joints where the handle partially envelops the slide. Figure 5.9 shows regions where the joint components can be extended.

Solution

This plan requires two mold stages. The handle is molded in the first stage. The parting direction for this stage is along the translation axis. The slide is molded in the second stage. The parting direction for this stage is perpendicular to the translation axis. This plan employs over-molding, i.e., the handle molded in the first stage is transferred one mold to another.

Example

Figure 5.10 shows an example for this plan. This design requires four mold pieces. Figure 5.10(b) and 5.10(c) show the two mold stages for the prismatic joint. In the first mold stage, the first material is injected into the assembly of mold piece A and B. Figure 5.10(c) shows the molded part after the first stage. For the second mold stage, the molded part produced in the first mold stage is transferred to the

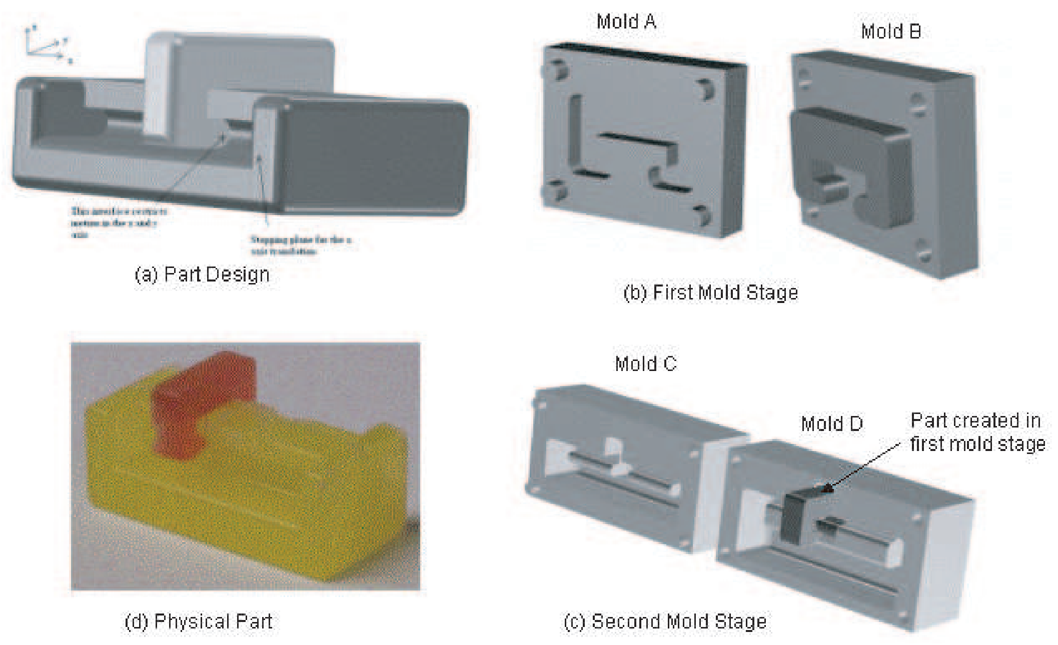


Figure 5.10: Example for prismatic joint plan B

assembly of mold piece C and D as shown in Figure 5.10(c). The second material is then injected to produce the final part.

Consequences

This plan follow the clearance principles outlined in Section 5.2.1 of molding the inner component after the outer component. This will cause the inner component (slide) to shrink away from the outer component (handle) establishing clearance between the contact surfaces. Since the handle only partially envelops the slide, this plan is more general than plan A, but the mold design and operation is much more complex. It requires four mold pieces, while plan A only requires two mold pieces and a simple core. More importantly, plan A employs a simple cavity manipulation technique where the core is simply pulled out of the previous mold

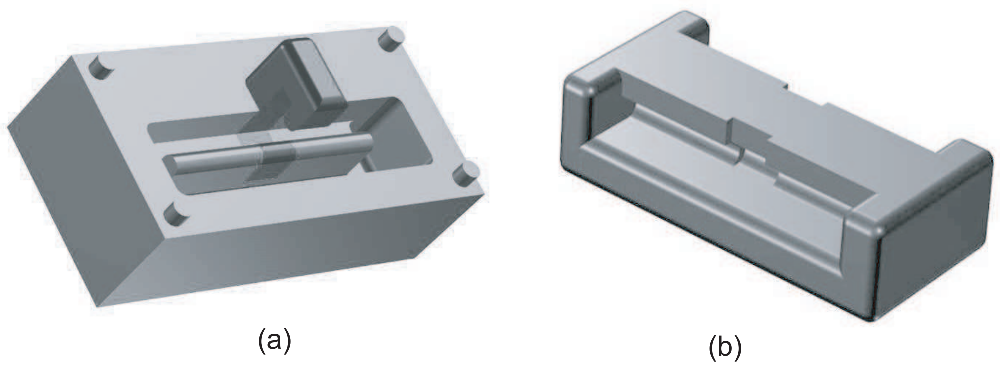


Figure 5.11: Effect of excessive shrinkage on part quality; (a) critical surface for shrinkage; (b) deformation caused by excessive shrinkage

stage, while plan B employs the complex cavity transfer mechanism where the part is transferred form one mold stage to the next. This can have very serious implications on part quality. Careful analysis of shrinkage in the handle mechanism is required in order to ensure correct part alignment in the second mold stage. As seen in Figure 5.11(a), the locking mechanism for the handle must perfectly align with the molded curvature for the slide. Figure 5.11(b) demonstrates how features of a part can become deformed if shrinkage is not accounted for. The slide will take on a deformed shape when molded. Despite the fact that the slide channel will shrink away from the handle after demolding, the part may not fully recover from this deformity. Consequently, the slide will suffer from tremendous friction loses and in extreme cases may be geometrically locked in place.

5.5 Molding Plans for Revolute Joint

While combinations of translational or prismatic joints make up 3 DOF possible for joint connections, revolute joints encompass the other 3 DOF of purely rotational

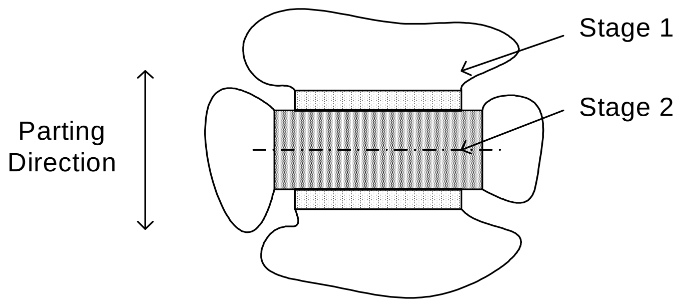


Figure 5.12: Revolute joint plan C

motion. Depending on the desired function, an articulated part may require 1 to 3 DOF for a revolute connection. Examples of these connections range from simple 1 DOF dimmer light switches to 3 DOF robotic arms. Pin connections are an excellent example of a simple revolute joint. These connections restrict all translational movements and only allow the part to rotate along one axis. When designing, one can combine several pin connections to increase the functionality of the part. This section presents two molding plans for realizing revolute joint. Both molding plans employ cylindrical contact surface to facilitate rotation along one axis but different features to constrain translation. One molding plan uses a cap-end connection while the other uses a groove connection.

5.5.1 Plan C

Applicability

This molding plan is for revolute joint with cap-end connection. The hole is made of ABS and the pin is made of Polyethylene. The diameter and length of the joint lies between [1/8, 1/2] inch. Figure 5.12 shows regions where the joint components can be extended.

Solution

This plan requires two mold stages. The outer cylinder is molded in the first stage. The inner capped cylinder is molded in the second stage. The parting direction is perpendicular to the rotation axis. The cavity shape is changed using a sliding core that is translated along the rotation axis. The core acts as a placeholder for the inner cylinder in the first stage. It is pushed out of the mold in the second stage creating a cavity into which the material for the inner cylinder is injected.

Example

Figure 5.13 shows an example for this plan. This design requires two mold pieces and a core. Figure 5.13(b) and 5.13(c) show the two mold stages. In the first mold stage, Mold pieces A and B are assembled and the core is inserted to the position shown in Figure 5.13(b). The first material is then injected into the assembled mold. Figure 5.13(c) shows the molded part after the first stage. For the second mold stage, the core is pulled out to the position shown in Figure 5.13(c). The second material is then injected to produce the final part.

Consequences

This plan follows the guidelines outlined in Section 5.2.1 of molding the inner component after the outer component. This will cause the inner component to shrink away from the outer component establishing clearance between the cylindrical surfaces. However, it must be noted that the inner component also shrinks along the cylindrical axis. This might cause the cap to stick to the outer component and jam the motion. Length of the inner component must be accordingly specified to compensate for this shrinkage.

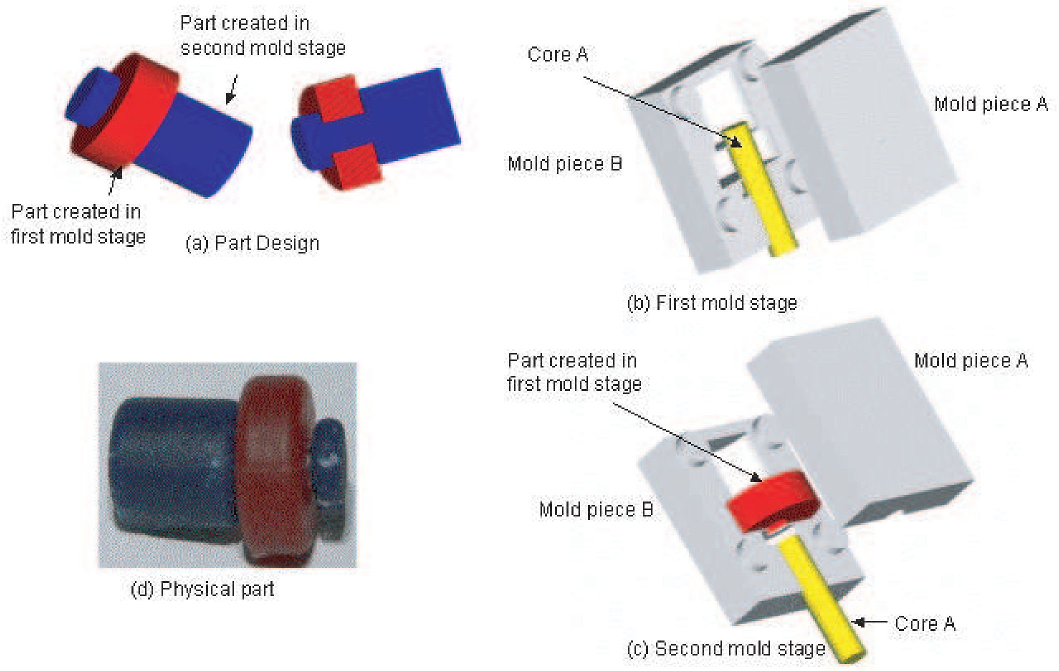


Figure 5.13: Example for revolute joint plan C

5.5.2 Plan D

Applicability

This molding plan is for revolute joint with a groove connection. Figure 5.14 shows regions where the joint components can be extended.

Solution

This plan requires two mold stages. The outer cylinder is molded in the first stage. The inner grooved cylinder is molded in the second stage. The parting direction is perpendicular to the rotation axis. The cavity shape is changed using two sliding cores that are translated along the rotation axis. The cores acts as placeholders for the inner cylinder in the first stage. They are pushed out of the mold in the second

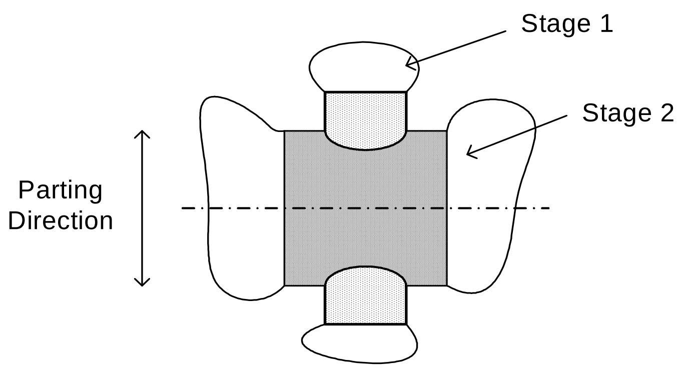


Figure 5.14: Revolute joint plan D

stage creating a cavity into which the material for the inner cylinder is injected.

Example

Figure 5.15(a) shows an example for this plan. This design requires three mold pieces and four cores. Figure 5.15(b) and 5.15(c) show the two mold stages. For the first mold stage core A1 is inserted into mold piece A and core A2 is inserted into mold piece C. Mold pieces A and C are assembled with mold piece B as shown in Figure 5.15(b). First stage material is then injected. Figure 5.15(c) shows the molded part after the first stage. For the second mold stage, cores A1 and A2 are replaced with cores B1 and B2 respectively as shown in Figure 5.15(c). The second material is then injected to produce the final part. Please note that Cores A1 and A2 have the same shape. They have been assigned different names to clearly illustrate the different assembly steps. Same is true with Cores B1 and B2.

Consequences

This plan follows the guidelines outlined in Section 5.2.1 of molding the inner component after the outer component. This will cause the inner component to shrink

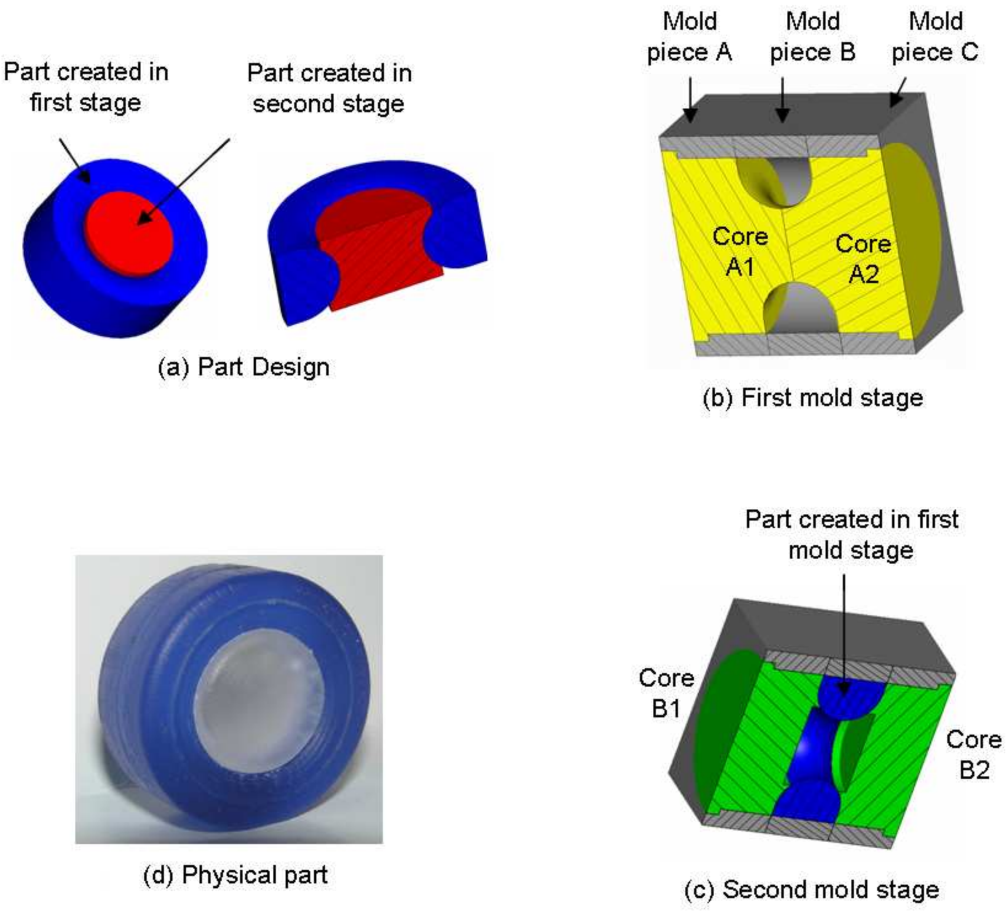


Figure 5.15: Example for revolute joint plan D

away from the outer component establishing clearance between the cylindrical surfaces. Although this plan provides similar functionality as plan C, but the mold design is much more complicated. It requires four cores compared to only one in plan C. Also the cavity shape change method is very complicated.

5.6 Molding Plans for Spherical Joint

Spherical joint allows for rotation in any direction simultaneously. A common example of this type of joint is known as the ball and socket joint. This type of joint is well known for its uses in the human body. There are many uses for the ball and socket joint in molded and assembled parts as well. Spherical joints are not widely used in molded plastic products because the joint needs to be realized by very precisely assembling many individual pieces. Therefore, instead of implementing a spherical joint, two or three pin connections are used in series to produce the same function.

In-mold assembly facilitates molding of spherical joints. A spherical socket requires a spherical core, which obviously cannot be disassembled from the molded socket. There are two options to disassembling the spherical core - using sacrificial core or splitting the core into three or more pieces. Using sacrificial core is costly and may not be suitable for many situations. Split cores introduce flash on the spherical surface that causes jerky motion. This section again presents two molding plans for realizing spherical joints.

5.6.1 Plan E

Applicability

This molding plan is for spherical joint where the socket surrounds about three-fourth of the spherical ball. Figure 5.16 shows regions where the joint components can be extended.

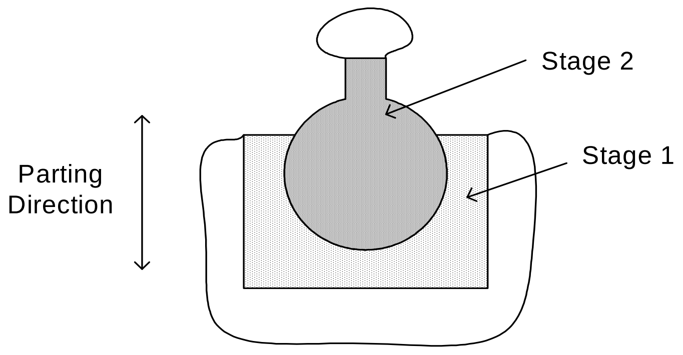


Figure 5.16: Spherical joint plan E

Solution

This plan requires two mold stages. The outer component (socket) is molded in the first stage. The inner component (ball) is molded in the second stage. The parting direction is along the opening of the socket. The cavity shape can be changed either using a sacrificial core or split cores that are translated along the parting direction itself. The cores acts as placeholders for the ball in the first stage. They are pushed out of the mold in the second stage creating a cavity into which the material for the ball is injected.

Example

Figure 5.17 shows an example for this plan. This design requires two mold pieces and five cores. Figure 5.17(b) and 5.17(c) show the different steps for molding the spherical joint. In the first mold stage, Mold pieces A and B are assembled with cores A, C1 and C2 as shown in Figure 5.17(b). The first material is then injected into the assembled mold. Figure 5.17(c) shows the molded part after the first stage. For the second mold stage, the cores A, C1 and C2 are replaced by cores B1 and B2 as shown in Figure 5.17(c). The second material is then injected

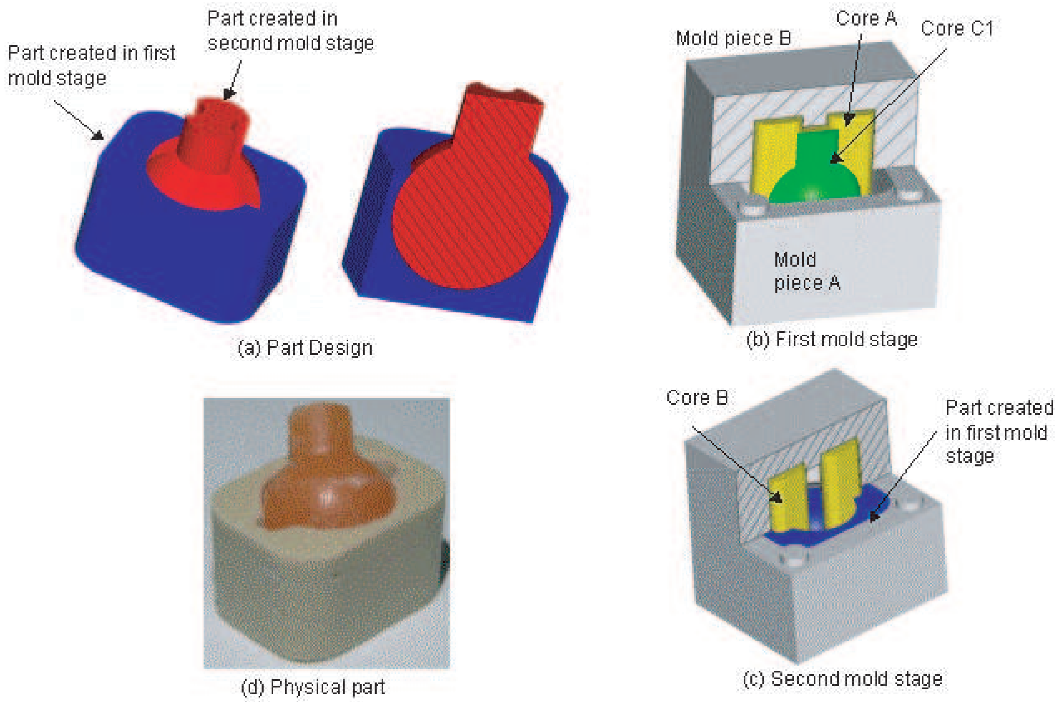


Figure 5.17: Example for spherical joint plan E

to produce the final part.

Consequences

This plan follows the guidelines outlined in Section 5.2.1 of molding the inner component after the outer component. This will cause the inner component (ball) to shrink away from the outer component (socket) establishing clearance between the contact surfaces. This plan uses split cores, but still produces smooth motion. The trick to getting a smooth motion in spite of the flash produced by split cores is not molding the entire ball and socket. Both, ball and socket are split and there are two empty spaces on both of them. As the joint is rotated, the flash on the surface will be pushed into those empty spaces causing a smooth motion.

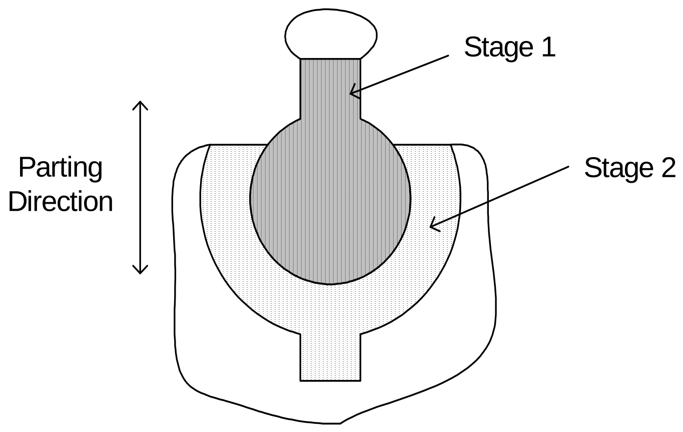


Figure 5.18: Spherical joint plan F

5.6.2 Plan F

Applicability

This molding plan is for spherical joint where the socket surrounds the spherical ball only around the equator. This enables the joint to have maximum rotational capabilities, but lesser mechanical strength. Figure 5.18 shows regions where the joint components can be extended.

Solution

This plan requires two mold stages. The inner component (ball) is molded in the first stage. The outer component (socket) is molded in the second stage. The parting direction is perpendicular to the equator where the socket surrounds the ball. The cavity shape is changed using two sliding cores that are translated perpendicular to the parting direction. The cores act as placeholder for the socket in the first stage. They are pushed out of the mold in the second stage creating a cavity into which the material for the socket is injected.

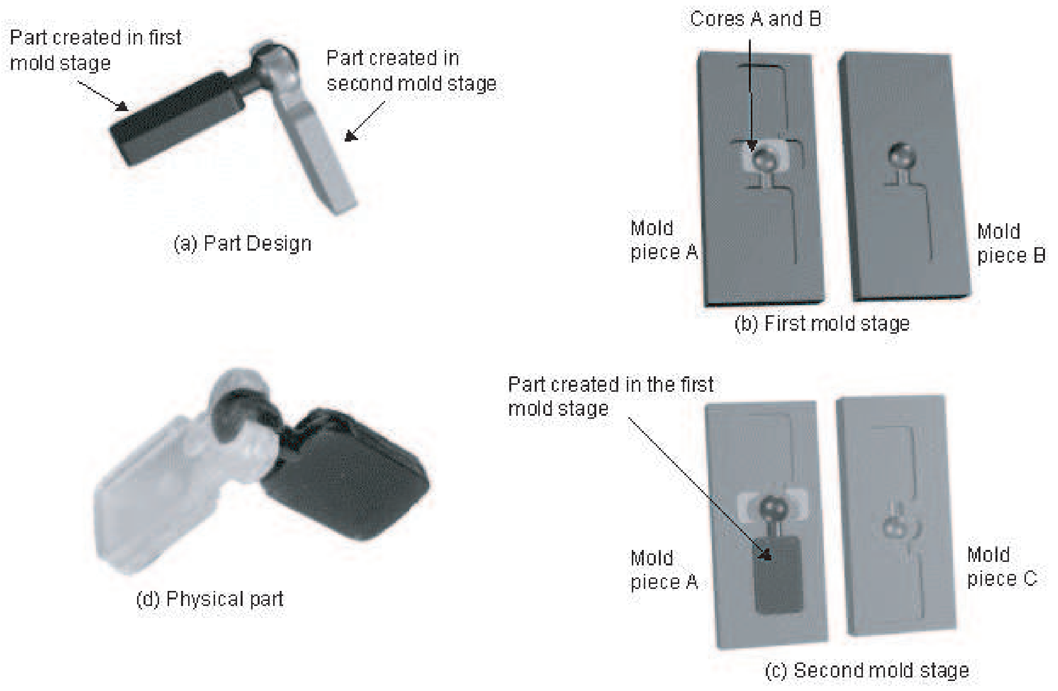


Figure 5.19: Example for spherical joint plan F

Example

Figure 5.19 shows an example for this plan. This design requires two mold pieces and two cores. Figure 5.19(b) and 5.19(c) show the different steps for molding the spherical joint. In the first mold stage, Mold pieces A and B are assembled with cores A and B as shown in Figure 5.19(b). The first material is then injected into the assembled mold. Figure 5.19(c) shows the molded part after the first stage. For the second mold stage, the mold piece B is replaced by mold piece C, and the cores A and B are simply pushed out sideways as shown in Figure 5.19(c). The second material is then injected to produce the final part.

Consequences

This plan does *not* follows the guidelines outlined in Section 5.2.1 of molding the inner component after the outer component. This might cause the socket to shrink onto the ball leading to a tighter clearance. As described in Section 5.2.1, this plan is not appropriate for components of large dimensions. It also requires uses of anti-stiction agents and materials with negligible shrinkage. However the mold design for this plan is much simpler as compared to plan E.

5.7 Summary

Generating feasible molding plans for joints is a time-consuming process. The designers need to experiment with size, geometry, material and molding sequence to generate a feasible and efficient molding plan. When a successful plan is developed and experimentally validated, a lot of useful knowledge is generated. In the absence of a formal framework to capture this information, most of it is lost and needs to be painstakingly regenerated for every new assembly.

This chapter presents a framework for representing molding plans for articulated joints. This framework allows us to record molding plans for joints in a reusable form. When a joint similar to a previously molded joint is found in a new assembly, the previously generated knowledge can be applied to the new joint. This chapter identified four assembly design principles that lead to feasible and efficient molding plans. If these design principles are followed, it is possible to reduce the molding cost and ensure that the molded assembly is of desired quality. This chapter identified the complete set of applicability conditions that needs to be satisfied in order to use a molding plan for a particular joint. This chapter also identified all feasibility constraints that can be transferred from the molding plan of the joint to the overall molding planning problem for an assembly. The molding plan for the assembly needs to satisfy these constraints of the joint molding plan in order to be feasible. This chapter finally presents six reusable molding plans for three basic joints – prismatic, revolute, and spherical.

Chapter 6

GENERATING MOLDING PLANS FOR ARTICULATED ASSEMBLIES

This chapter investigates the problem of generating molding plans for articulated assemblies. This chapter is arranged in the following manner. Section 6.1 formulates the molding plan problem as a state-space search problem. Section 6.2 presents a high-level overview of approach, while Section 6.3 presents a detailed description of the overall algorithm. Section 6.4 describes the geometric algorithms that are used to generate a molding stage. Section 6.5 finally illustrates the algorithm with the help of some examples.

6.1 State Space Formulation

Generating molding plan is similar to other manufacturing operation-planning problems such as machining and sheet-metal bending. There are two traditional approaches to process planning. The first approach is called *generative* process planning. In this approach a plan is synthesized from the first principles by trying various alternatives in generate-and-test paradigm. The second approach is called *variant* process planning. In this approach, a plan is generated by modifying an existing plan. Purely generative approaches are unlikely to work in the molding planning domain. A lot of knowledge that is needed to successfully mold joints does not exist in an explicit geometric form. Purely variant approaches are also unlikely to work because every new assembly is significantly different from the previously generated assemblies.

We use a hybrid approach that combines elements from generative and variant techniques. We reuse the molding plans for individual joints described in Chapter 5 to generate plans for new assemblies. This allows us to reuse existing molding knowledge and yet ensure that we can handle a wide variety of molding planning problems. When a new assembly is encountered we first check if the joints used in the assembly are sufficiently similar to the joints for which plans exist. This is performed by comparing the:

- Type of joint,
- Size of joint,
- Geometry, and
- Material of connected components

If a joint in the assembly is sufficiently similar to a joint with the known molding plan, then the molding plan for the joint is used as feasibility constraints in the planning process. This ensures that our method will only generate feasible plans. The rest of the planning proceeds in a generative manner.

The problem statement for generating molding plan (GenerateMolding-Plan) is given in Section 3.5. The input to GenerateMoldingPlan is the articulated assembly for which molding plan is to be generated. To solve the

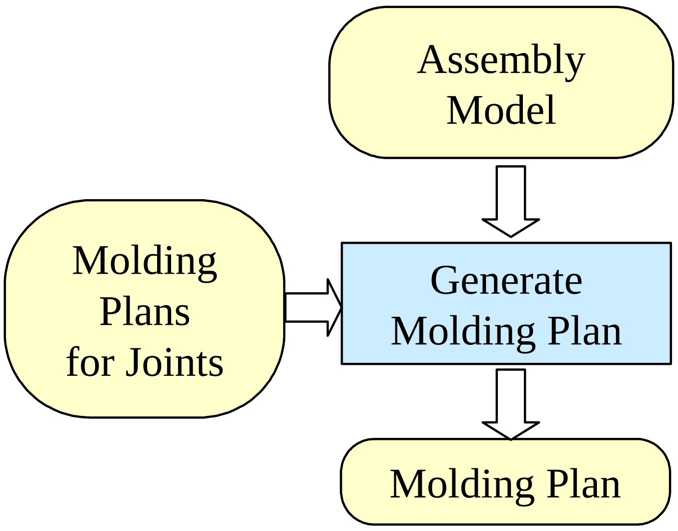


Figure 6.1: Molding plan problem

problem using hybrid approach, we also need a molding plan for each joint in the assembly. A molding plan for a joint can be obtained automatically as follows. The joint parameters (type, size, geometry, and material) are first identified by analyzing the CAD model of the input assembly. The molding plan database described in Chapter 5 is then queried with the joint parameters to obtain the molding plan of the closest-match joint. This dissertation does not provide low-level details for accomplishing the same. The algorithm assumes that a molding plan is provided for each joint in the assembly. This molding plan may either be obtained automatically or explicitly provided by the designer. The molding plan problem is represented graphically in Figure 3.12.

For a molding plan to be feasible, it must satisfy the feasibility constraints of the joints in addition to the molding stage feasibility constraints given in Section 3.3.1. The new stage constraints can be informally summarized as follows. The mathematical definitions for each constraint is given in Section 3.3.1 and Section 5.3.2.

1. *Joint precedence constraints.* The molding plan for a joint specifies the order in which the connected components need to be molded. For example, hole must be molded before pin in a revolute joint. The molding plan for the assembly must follow precedence constraints for all the joints in the assembly.

2. *Joint axis constraints.* 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.

3. *Joint parameter constraints.* The molding plan for a joint specifies a feasible range of joint parameters. A molding stage forming a component of a joint must be configured such that the joint parameter is within the feasible range.

4. *Concurrency constraints.* All components molded in a single stage must be of the same material and connected to a common base component.

5. *Intersection constraints.* The stage subassembly must be configured such that components do not intersect with each other.

6. *Shadow constraints.* The stage subassembly must be configured such that components do not cast shadow upon each other.

This dissertation formulates the molding planning problems as a state-space search problem. Section 2.5 briefly reviews the state-space search algorithms. For molding plan problem, one can develop many different types of state-space formulations. We will use the forward chaining formulation for explaining the scheme described in this thesis. 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 3.12. 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 6.3 shows a portion of the state space for the gimbal assembly shown in Figure 6.2. 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.

6.2 Overview of Approach

The state-space search problems are known to be combinatorial optimization problems. For an assembly with m components, a brute-force approach that evaluates

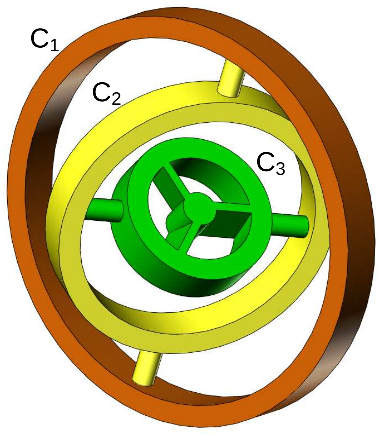


Figure 6.2: Gimbal

all possible states in Figure 6.3 will take $O(m!)$ time. The state-space search problems are usually solved using branch and bound algorithm with a lower time complexity. This technique generates one path at a time, keeping track of the best solution so far. It uses that best solution as a bound on future branches of the search. A bounding function assigns a bound to each node in the search tree. For leaves the bound equals the value of the corresponding solution, whereas for internal nodes the value is a lower bound for the value of any solution in the subspace corresponding to the node. 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. The branch and bound technique is briefly reviewed in Section 2.5.

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

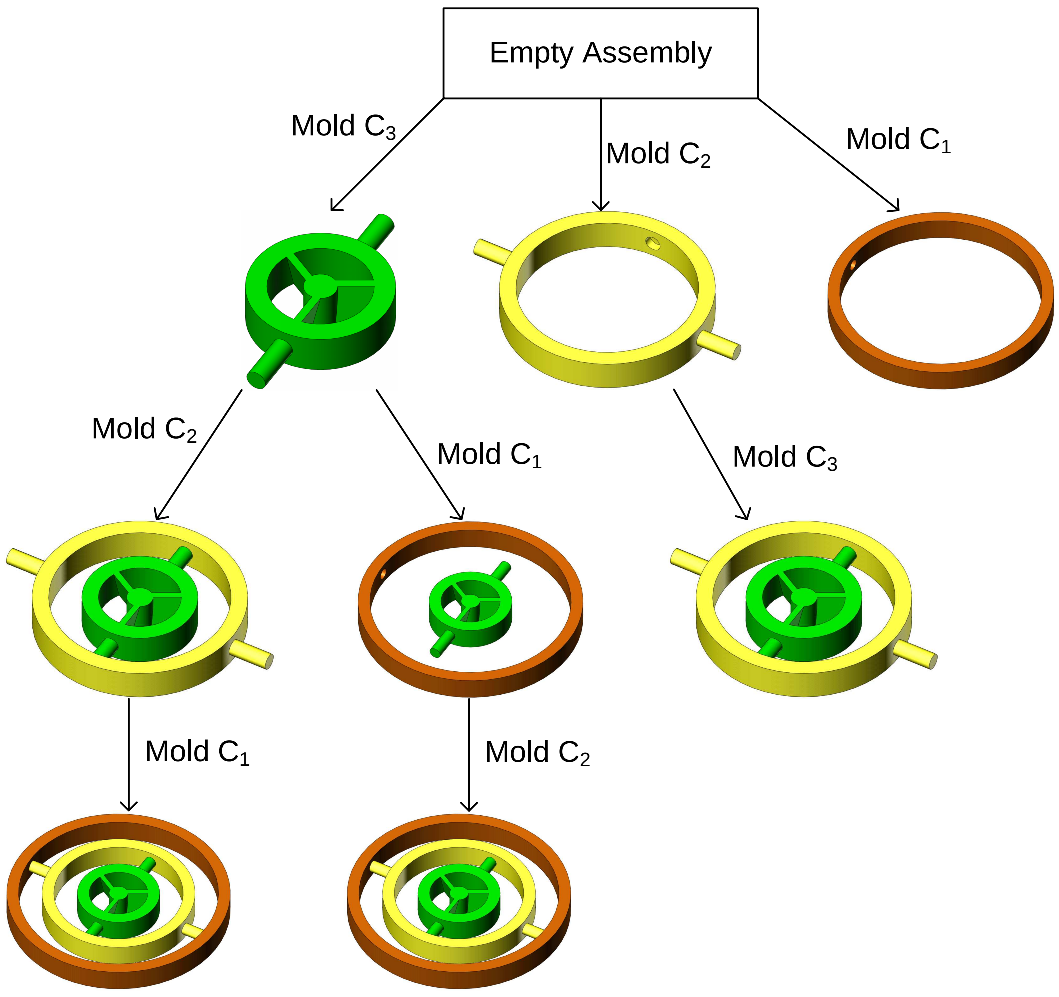


Figure 6.3: Partial state space

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 also takes a lot of time. This step usually involves extensive geometric reasoning and making queries to a geometric kernel. Such computation is time-consuming and leads to very high node-generation time. Slow node generation also makes it difficult to explore large portions of the search space. We use the following two techniques to solve the state-space search problem efficiently:

1. *Reduce the search space using feasibility constraints.* We eliminate infeasible solutions as early as possible to save time. As we will see later, the first step of generating a node is to find stage components. This can be done by simply enumerating all alternatives, but it will lead to redundant computation. We observe that the precedence constraints from the joints and the concurrency constraints from the problem requirements can be applied at this stage itself to quickly eliminate the infeasible sequences. We in fact create a Directed Acyclic Graph (DAG) from the precedence constraints of the joints and traverse it to generate stage components. We also arrange the steps of the node-generation algorithm such that least time is wasted on infeasible nodes.

2. *Reuse the results of a search node.* Each node in the search tree contains an intermediate subassembly as shown in Figure 6.3. 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 6.3 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.

6.3 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.

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. This number is quite manageable in most practical cases. 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 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 Cost($S$) $<$ IncumbentCost, then
        • IncumbentSolution $:= S$
        • IncumbentCost $:=$ 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 Cost($S$) $<$ IncumbentCost, then
        • IncumbentSolution $:= S$
        • IncumbentCost $:=$ Cost($S$)
```

   (d) Return.

4. Find a lower-bound cost $f(P) := \textsc{Cost}(S) + h(P)$ as described in Section 6.3.1

5. If $f(P) \geq$ IncumbentCost then return.

6. Branch on P generating children search nodes $P' = \{P_1, \ldots, P_q\}$ using algorithm
   $\textsc{GenerateMoldingStages}(A_P)$ described in Section 6.4

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) $\textsc{ProcessNode}(P_i)$

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 (IncumbentSolution). 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 (LivePriorityQueue) 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.

From the above, our algorithm consists of three main components:

1. A *bounding function* (Section 6.3.1) providing a lower bound for the best solution obtainable in a subspace.

2. A *strategy for reusing* (Section 6.3.2) the results of previously processed nodes.

3. A *branching rule* (Section 6.3.3) to be applied if a search node after investigation cannot be discarded, hereby branching the current node into two or more nodes to be investigated in subsequent iterations.

6.3.1 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. Section 3.3.2 describes a method to calculate the relative cost of a molding stage. It consists of molding cost, defect cost, and tooling cost. However, as described in Section 3.3.2 the molding cost is usually very large compared to the defect cost and the tooling cost. Therefore, when comparing two molding plans, it is sufficient to only compare the molding cost. The molding cost for a molding stage is given by Equation 3.13, which consists of:

1. Setup time (constant cost)
2. Cooling time (directly proportional to the wall thickness of the part)
3. Ejection time (directly proportional to the number of undercuts required for non-joint features)

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 3.3.1. 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.

6.3.2 Reusing the Results of a Search Node

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 6.3 will result in the same intermediate subassembly. So the subtree below both the nodes will be identical. This observation is used to reuse the results of previously solved subproblems.

6.3.3 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 6.3 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 6.3 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 6.4.

6.4 Generating Molding Stages

This section describes an algorithm for generating a molding stage for an intermediate assembly. This algorithm is used by algorithm ProcessNode to branch an interior node in the search tree. A molding stage s_i is represented as a tuple (C_i, T_i) as defined in Equation 3.12, where C_i is the set of components to be molded in molding stage s_i and T_i represents the configuration of the subassembly A_i. The configuration T_i of an assembly is defined in Equation 3.4 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.

Figure 6.4 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 6.4.1 describes an algorithm to find the stage components C_i. Section 6.4.2 describes a method to determine $\bar{T}$. Section 6.4.3 and Section 6.4.4 determine the joint coordinates θ_i for stage components, while Section 6.4.5 determines the θ_i for pre-stage components. 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 FIND-STAGECOMPONENTS described in Section 6.4.1

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 6.4.2
   (c) $\Theta = \emptyset$
   (d) For each stage component $a_j \in C_i$ do
       i. Find a feasible and optimal configuration $\theta_j$ as described in Sec-
          tion 6.4.3 and Section 6.4.4
       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. Find a feasible configuration $\theta_k$ as described in Section 6.4.5
       ii. $\Theta = \Theta \cup \theta_k$
   (g) $S_P = S_P \cup \{\Theta, \bar{T}\}$
4. Return $S_P$
```

6.4.1 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

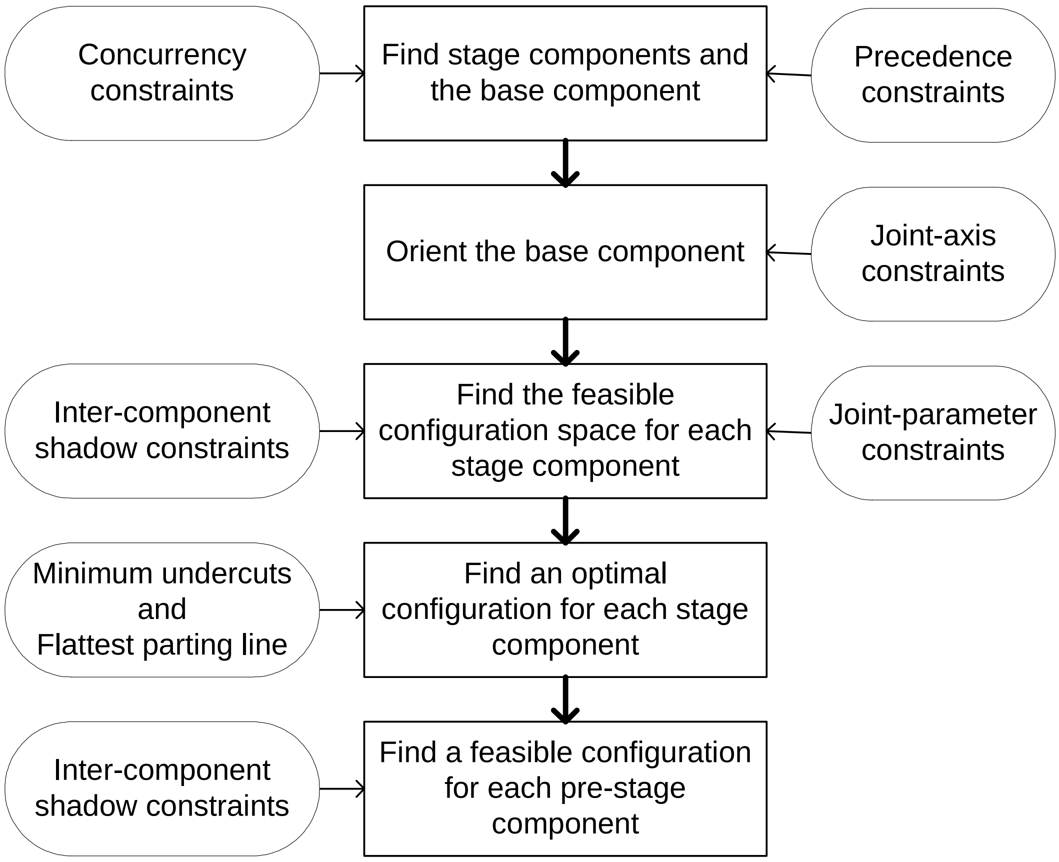

Concurrency constraints
Find stage components and the base component
Precedence constraints
Orient the base component
Joint-axis constraints
Inter-component shadow constraints
Find the feasible configuration space for each stage component
Joint-parameter constraints
Minimum undercuts and Flattest parting line
Find an optimal configuration for each stage component
Inter-component shadow constraints
Find a feasible configuration for each pre-stage component

Figure 6.4: Method to generate a molding stage

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. This partial ordering can be represented as a Directed Acyclic Graph (DAG). Figure 6.5 shows the precedence constraints DAG for the gimbal shown in Figure 6.2. The DAG shown in the figure implies that component c_3 must be molded after c_2, which must be molded after c_1.

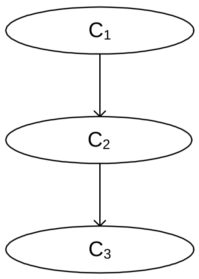


Figure 6.5: Joint precedence constraints for gimbal

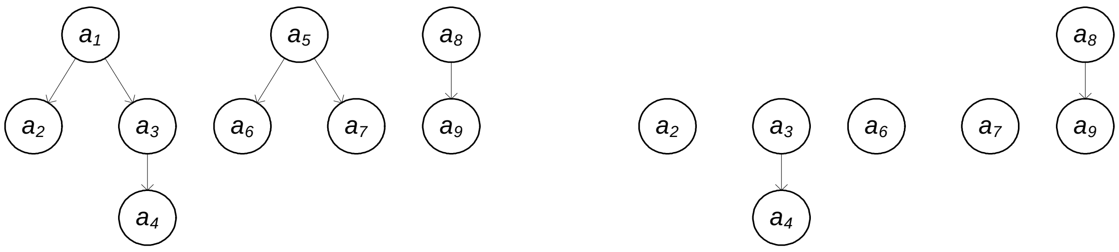
4. Concurrency constraints: The concurrency constraints represent the requirement that all components molded in a single stage must be of the same material and connected to a common base component. The parameters for this constraint are available in the input assembly model. Each component model a_i 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 indegree count is zero
3. Partition U into groups of components $C = \{C_1, \ldots C_q\}$ such that $C_i \subset U$ and $\cup C_i = U$. This can be achieved by union-find algorithm [Corm90] that partitions a set of elements into equivalent groups.
4. Return C.

Suppose that the input assembly is $A = \{a_1, \ldots, a_9\}$ and the current subassembly is $A_P = \{a_1, a_5\}$. The precedence constraints DAG is shown in Figure 6.6a.



(a) Initial DAG (b) After molding a_1 and a_5

Figure 6.6: Precedence constraints for an assembly.

Figure 6.6b shows the DAG after removing the nodes a_1 and a_5 corresponding to the current subassembly. The components for which indegree count is zero are $U = \{a_2, a_3, a_6, a_7, a_8\}$. Suppose that a_2 and a_3 are made of same material and connected to a_1. Similarly a_6 and a_7 are made of same material and connected to a_5. U can thus be partitioned into three groups of stage components $C = \{\{a_2, a_3\}, \{a_6, a_7\}, a_8\}$. The base component for the group $\{a_2, a_3\}$ is a_1 and that for $\{a_6, a_7\}$ is a_5. Hence, the node P corresponding to the subassembly A_P can be branched into three new nodes.

6.4.2 Orienting the Base Component

The configuration T_i of an assembly is defined in Equation 3.4 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-axis 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 $\bar{T}$. Section 2.4.3 provides some background on transformations applied to geometric entities. 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 as described in Section 2.4.3.

6.4.3 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.

Lemma 6.1. *If two components do not cast shadow on each other along a viewing direction, they also do not intersect.*

Proof. From the separating axis theorem (described in Section 7.2.2), if there exists a plane for which the projection of two objects do not intersect, then the objects do not intersect. Suppose that the viewing direction is along the z-direction. By definition, if two components a and b do not cast shadow on each other along the z-direction, the projections of a and b on the x-y plane do not intersect. Hence a and b do not intersect. □

The above lemma implies that the shadow constraint is stricter than the intersection constraint. 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.

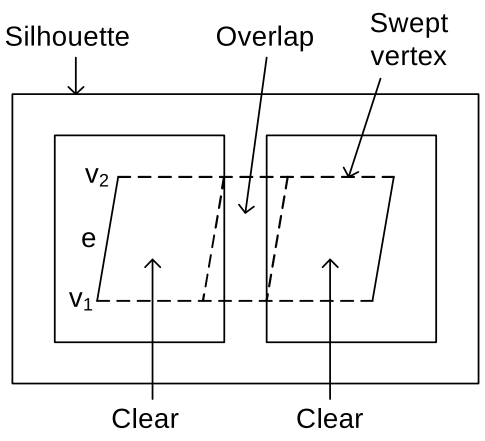
The projection of a triangulated polyhedron p onto a plane consists of a set of triangles. A projection $\bar{p_1}$ intersects with another projection $\bar{p_2}$ if any triangle in $\bar{p_1}$ intersects with any triangle in $\bar{p_2}$. A triangle t_1 intersects with another coplanar triangle t_2 if 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. The molding plans for prismatic joints presented in Chapter 5 specify 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 6.7a 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 6.7b 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.

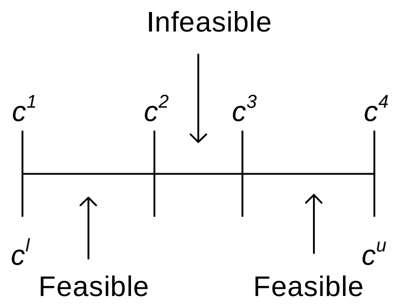
$$\theta = \cap_{j=0}^{m} \{[\theta_j^1, \theta_j^2], \ldots, [\theta_j^{k-1}, \theta_j^k]\}$$

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



(a) Changing overlap status



(b) Partition of the initial feasible space

Figure 6.7: Determining the feasible configuration space for a stage component

silhouette continuously changes. Hence we need to consider all edges of the stage component.

Theorem 6.1. *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 presented in Section 6.4.5 to handle spherical joints.

6.4.4 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. As described in Section 3.3.2, we need to optimize the molding cost (C_m), defect cost (C_d), and tooling cost (C_t). The molding cost is directly proportional to the number of undercuts, defect cost is directly proportional to the complexity of the parting line, while the tooling cost is directly proportional to the time taken to machine the shutoff surface. Usually,

$$C_m \gg C_d \gg C_t \tag{6.1}$$

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. Hence, we follow a *hierarchical* approach in optimizing the parameters of a molding stage. This hierarchical optimization scheme will not work in cases where Equation 6.1 does not hold.

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 Chapter 8. The first two steps are described in the following sections.

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 (defined in Section 5.3.2). 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

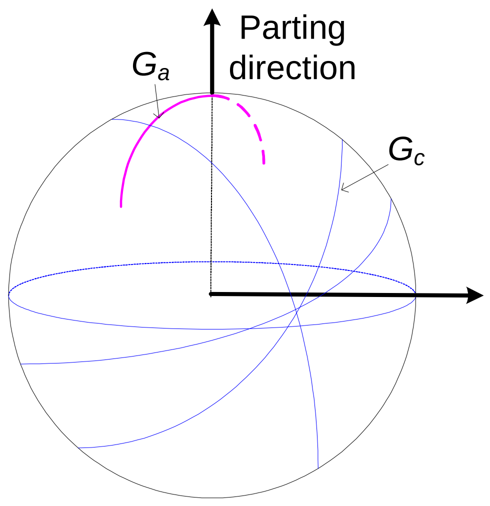


Figure 6.8: Scheme for finding the configuration with minimum undercuts

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 these great arcs G_a as shown in Figure 6.8. 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. The algorithm for detecting the undercuts on a component is described in Chapter 7.

Definition 6.1. *The curve of intersection of a plane and a sphere is a circle. It* *is called a great circle if the plane passes through the center of the sphere. Two non-coincident great circles intersect at exactly two points that are diametrically opposite.*

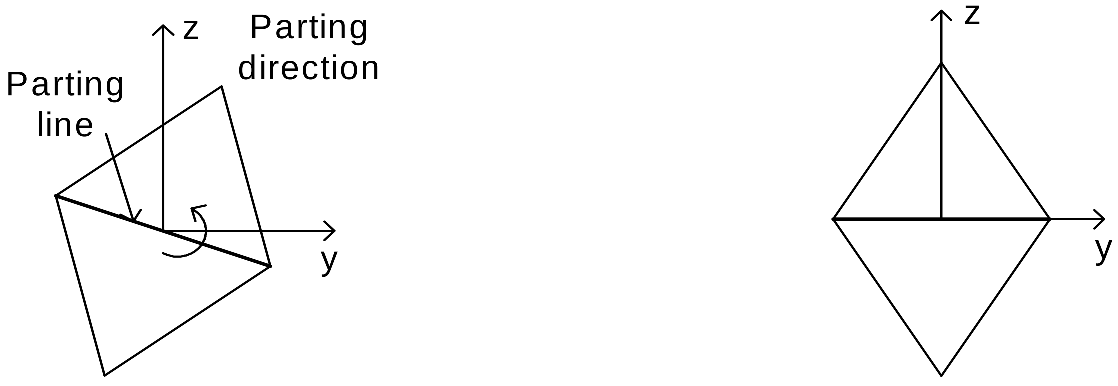
Theorem 6.2. *Let P be a stage component with n vertices connected to a base component with a revolute joint. The configuration with the minimum number of undercuts can be found in $O(n^4)$ time.*

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 6.2 states that it takes $O(n^2)$ time to test each orientation because each facet needs to tested against every other facet. However Section 7.2 presents an algorithm for detecting undercuts which is almost linear. Section 7.3 presents another fast algorithm that runs on GPU. Using the GPU-based algorithm, testing $O(n^2)$ directions is not that expensive.

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



(a) Initial orientation (b) Optimal orientation

Figure 6.9: Finding a configuration for which the parting line is flattest

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 Chapter 7. 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 6.9 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 6.10. The parting direction is along the z-direction and the joint axis is along the x-axis. Let AB be the projection of

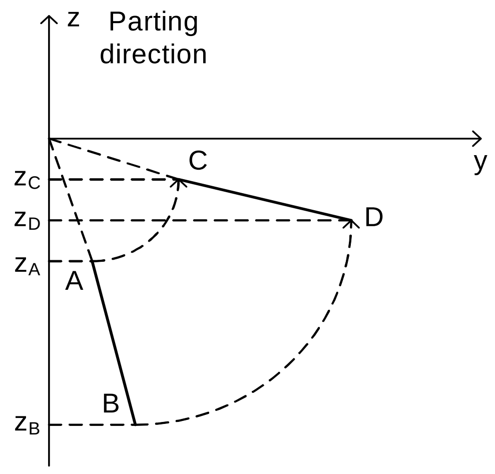


Figure 6.10: Scheme for finding the orientation for flattest parting line

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 6.10, 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.

$$\begin{aligned} Minimize \quad & f(\alpha) = \textstyle\sum_{i=0}^{k} l_i^2 sin^2(\alpha_i + \alpha) \\ s.t. \quad & 0 \leq \alpha_i \leq \theta \end{aligned} \tag{6.2}$$

Differentiating the minimization function gives:

$$\sum_{i=0}^{k} 2l_i^2[(2\sin\alpha_i\cos\alpha_i)\sin^2\alpha - (1 - 2\sin^2\alpha_i)\sin\alpha\cos\alpha + (\sin\alpha_i\cos\alpha_i)] = 0 \tag{6.3}$$

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

Theorem 6.3. *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 6.3 is formed and solved. Summing up the k terms takes $O(k)$ time and solving it takes $O(1)$ time. Hence the overall complexity of the algorithm is $O(k)$. □

6.4.5 Finding Feasible Configurations for Pre-Stage Components

A molding stage s_i consists of three types of components for which we need to find a feasible configuration (joint parameter θ):

1. Base component b: Section 6.4.2 describes an algorithm to orient b.

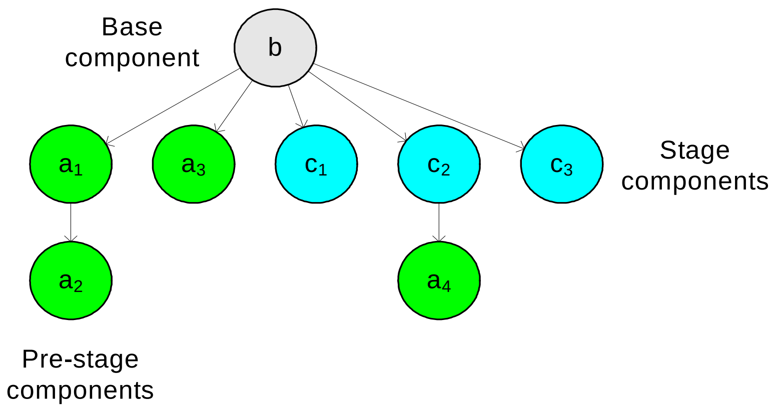


Figure 6.11: Tree representation of the subassembly produced by a molding stage

2. Stage components C_i: Section 6.4.3 and Section 6.4.4 describe algorithms to orient C_i.

3. Pre-stage components A_Q: This stage finally orients the pre-stage components such that there is no shadow between the A_Q and $b \cup C_i$.

The subassembly that will be produced by stage s_i can be represented as a tree with the base component b at the root. Figure 6.11 shows such a representation. The nodes in the figure are components while the edges are the joints between the components. Each branch of the tree is a chain of links (components). We need to find the joint parameters for the pre-stage components with the links corresponding to the base component and the stage components fixed.

We describe an incremental approach for finding feasible configurations for pre-stage components. We incrementally change the joint parameter θ_i each pre-stage component a_k and mark all the feasible configurations. If we consider the subassembly shown in Figure 6.11, the set of feasible configurations for the pre-stage components $\{a_1, a_2, a_3, a_4\}$ can be represented as another tree shown in Figure 6.12. Each node in the tree shows a feasible configuration for a pre-stage component.

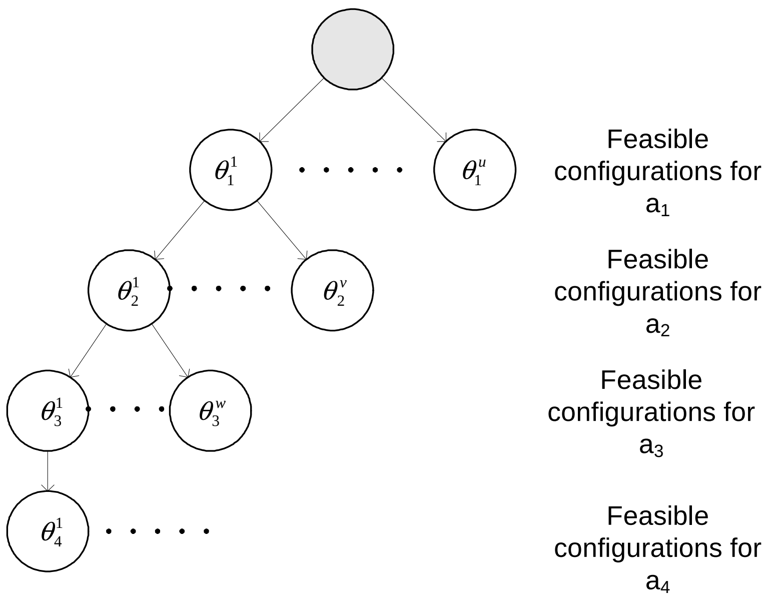

θ_1^1
θ_1^u
Feasible configurations for a_1
θ_2^1
θ_2^v
Feasible configurations for a_2
θ_3^1
θ_3^w
Feasible configurations for a_3
θ_4^1
Feasible configurations for a_4

Figure 6.12: Feasible configurations for pre-stage components

The nodes on the first level of the tree show the feasible configuration for component a_1. Similarly nodes on levels two to four show the feasible configurations for components a_2 to a_4. Any path from the root to a leaf node gives feasible configurations for the pre-stage components. We use the algorithm for finding mold-piece regions described in Chapter 7 to determine the feasibility of a configuration. The transformation of the components in a chain is calculated by concatenating the transformation of components from the root of the tree to that component as described in Section 3.1.2. For example, the transformation of component a_2 in Figure 6.11 is:

$$T_{a_2} = T_b \cdot T_{a_1/b} \cdot T_{a_2/a_1}$$

This algorithm incrementally evaluates only discrete configurations and hence suffers from aliasing issues like any other discrete sampling algorithm. It may not be able to find a feasible configuration when it actually exists.

6.5 Results

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

6.5.1 Swashplate

A swashplate is a mechanical device used in helicopters to control the motion of the main rotor blades. An example swashplate is shown in Figure 6.13. The swashplate 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 swashplate example has two revolute joints. The designer compares the type, size, geometry, and material of the joints to find the molding plan C described in Section 5.5.1 for the joints. The assembly model of the swashplate and molding plan C is fed to the planner.

The planner first creates a Directed Acyclic Graph of components using the precedence constraints in the joint molding plan. Plan C specifies that pin must be molded after hole, i.e., component C_2 must be molded after component C_1 and C_3. Figure 6.14 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. Section 3.3.2 gives the equation for calculating the total cost of a molding stage. As explained in Section 6.3.1, we only consider the relative cost between two solution paths. The relative cost consists of setup cost, cooling cost, and undercut cost for non-joint features. Using the cost equation and pa-

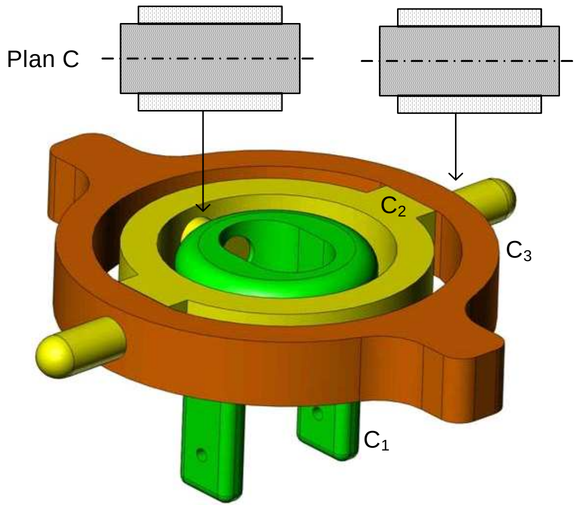


Figure 6.13: Swashplate

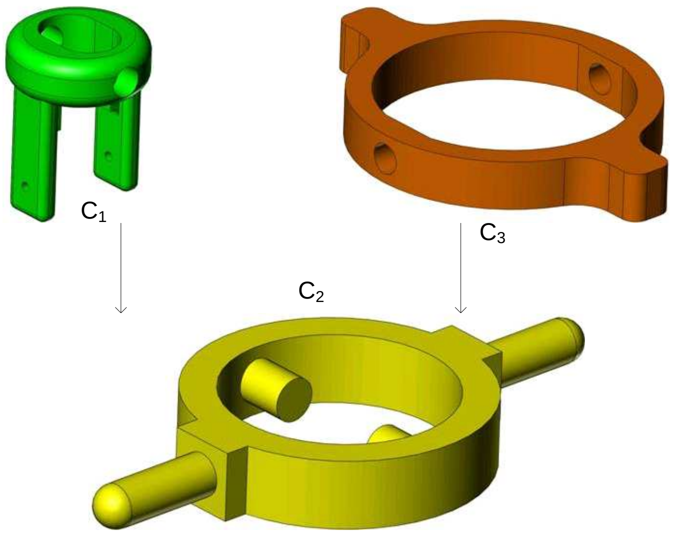


Figure 6.14: Joint precedence constraints for swashplate

rameters given in Equation 3.13, we get the following lower bound cost values for each component:

- $h(C_1) = 203.30 + (1.5 \times 0.5) + (0.7 \times 2) = 205.45$

- $h(C_2) = 203.30 + (1.5 \times 6.5) = 213.05$
- $h(C_3) = 203.30 + (1.5 \times 19.6) = 232.70$

Figure 6.15 shows the complete state space for the swashplate example. The node numbers $(N_1, \ldots, N_8)$ correspond to the sequence in which each node is processed by the algorithm ProcessNode. The bounding value f is also shown against first-level nodes.

Processing N_0

In the first stage, we can either mold C_1 (node N_6) or C_3 (node N_3) because they must be molded before C_2. We can in fact mold C_1 and C_3 together (node N_1) in a single stage because they are made of the same material. We use the best-first search strategy when choice is to be made between nodes at the same level of the search tree. Figure 6.15 shows the bounding values f for the nodes N_1, N_3, and N_6. We choose N_1, which has the minimum bounding value.

Processing N_1

At this stage, only one component is left to be molded. Hence we do not need to branch the node. We just need to find the molding configuration for the assembly and create a shutoff surface. Here we have one stage component C_2. It is connected to two components C_1 and C_3. Hence any one of the two can serve the role of a base component. We arbitrarily choose C_1 as the base component and use the joint-axis constraints from plan C to orient C_1 such that the joint axis is perpendicular to the parting direction (z-direction). The component C_3 is oriented using the inverse kinematics scheme described in Section 6.4.5.

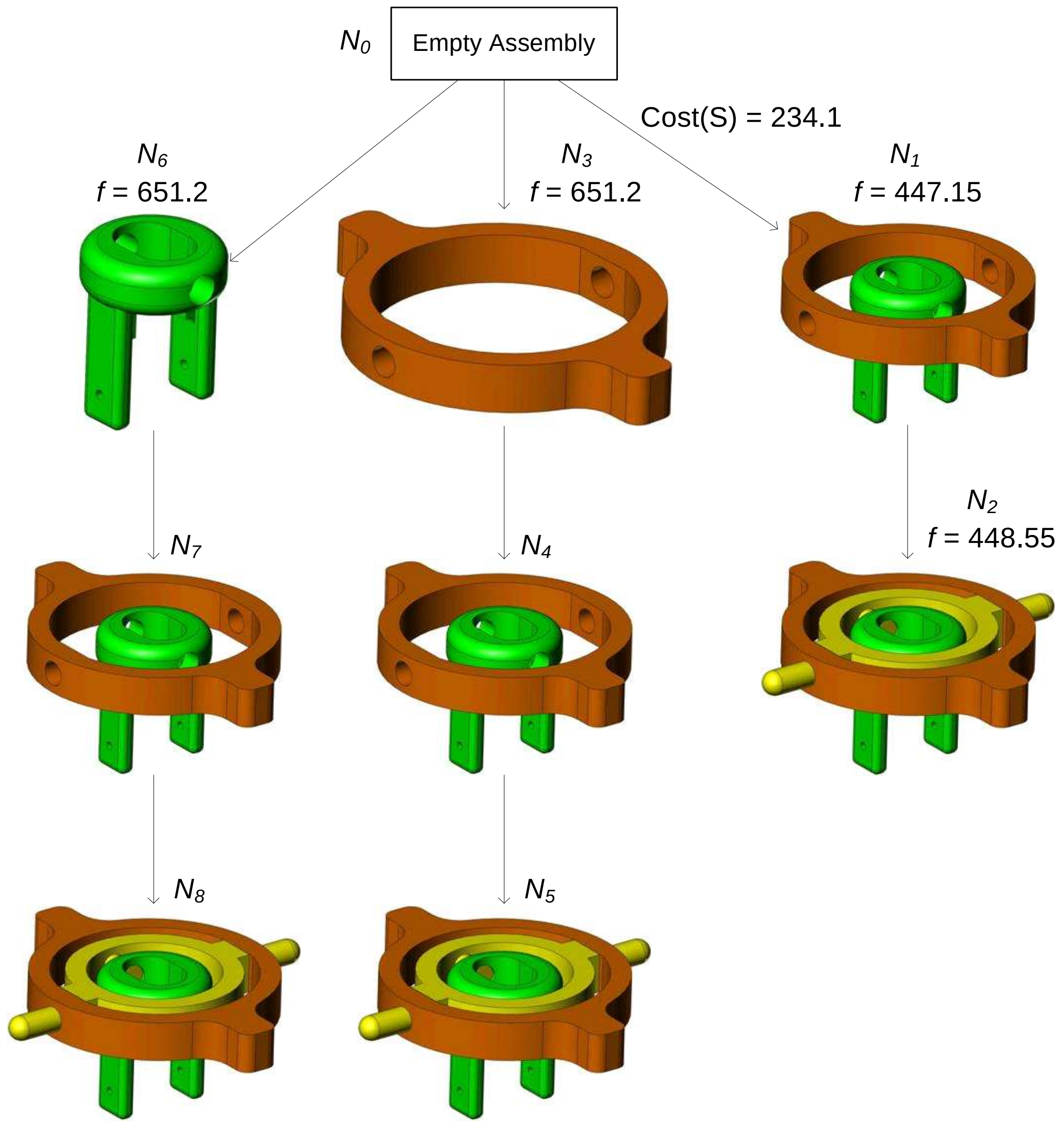


Figure 6.15: Complete state space for swashplate

Next, we need to find a feasible configuration space for component C_2 such that C_1 and C_2 do not occlude each other in z-direction. Figure 6.16a shows the top view (camera pointed toward $-z$-direction) of the subassembly. It also shows the axis along which C_2 can be rotated. Figure 6.16b shows the range of joint angle θ for which C_1 and C_2 do not occlude each other.

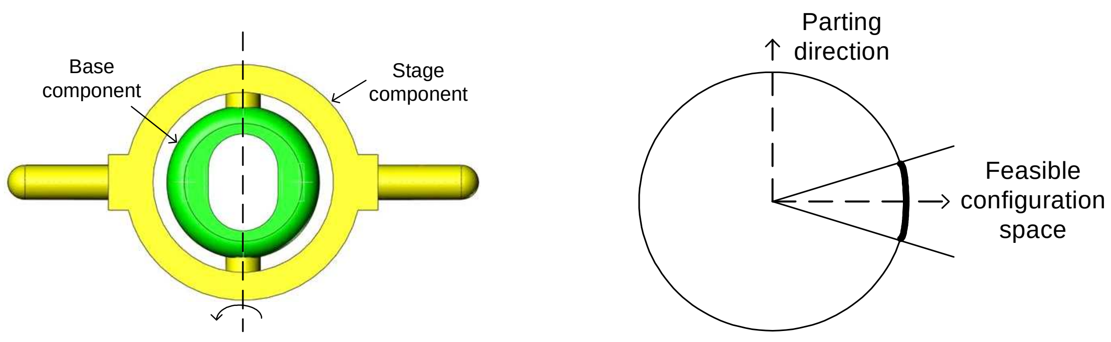


Figure 6.16: Feasible configuration space for C_2.

The next step is to find a configuration for C_2 such that the number of undercuts is minimum and the parting line is flattest. Section 6.4.4 describes an algorithm that uses a gaussian sphere to find an optimal configuration for a component. The feasible configuration space for component C_2 is represented as a great arc S_c on the sphere. Each facet plane can also be represented as great circles on the sphere that intersect S_c. The intersection points represent visibility events where visibility of a facet changes. It is sufficient to evaluate the number of undercuts and flatness of parting line at those visibility events. Figure 6.17a shows the component C_2 and the joint axis along which it is rotated. Figure 6.17b shows the gaussian sphere and a couple of visibility event points. It also shows the point at which optimal configuration is achieved.

Processing N_3 and N_6

These nodes can be safely fathomed as the bounding value is greater than the incumbent solution. If we had to traverse the subtree below N_3 and N_6, we could have reused the results from N_1 – the subassemblies in nodes N_4, N_7, and N_1 are equivalent. It must be noted that in most of the cases, the molding plan with

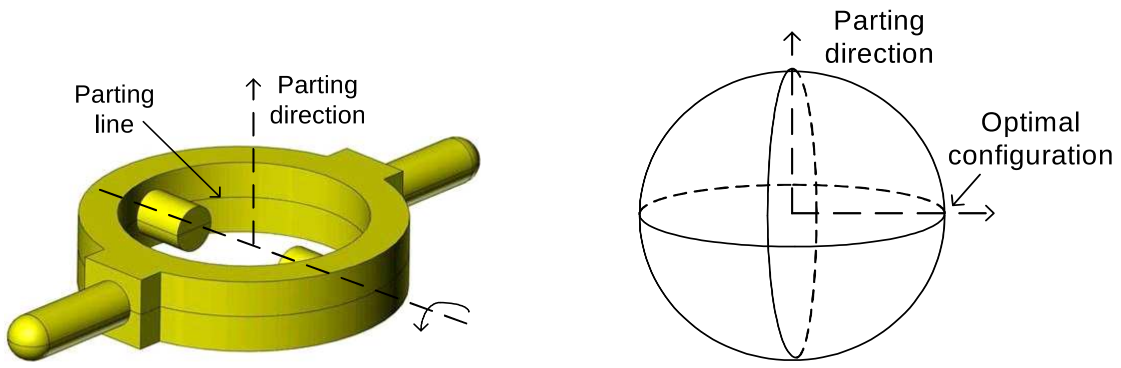


Figure 6.17: Optimal configuration for C_2.

minimum number of stages is the winner.

6.5.2 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 6.18 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.

The revolute joints in the vent assembly are similar to that in plan C which will be used for this example. Plan C specifies that pin must be molded after hole, i.e., the vanes must be molded after component the main body. Figure 6.19 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

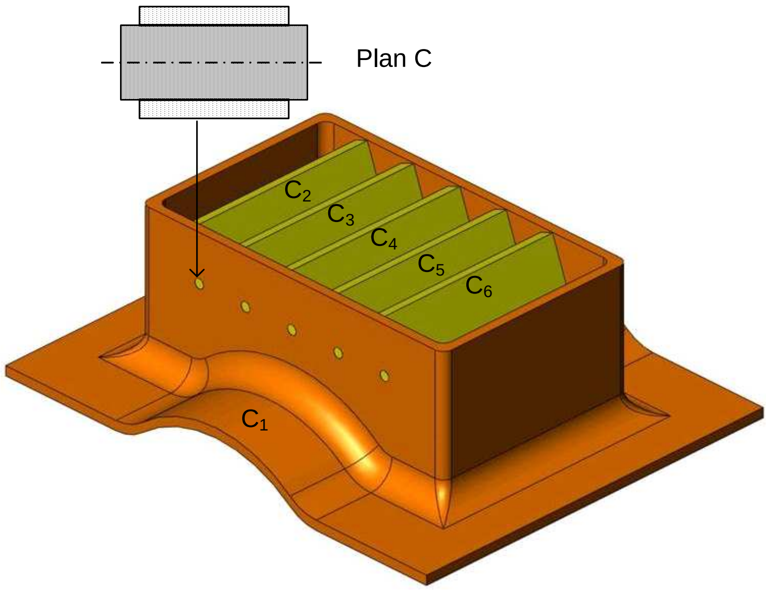


Figure 6.18: Vent assembly.

$$\begin{tikzcd}
 & & C_1 \arrow[dll] \arrow[dl] \arrow[d] \arrow[dr] \arrow[drr] & & \\
C_2 & C_3 & C_4 & C_5 & C_6
\end{tikzcd}$$

Figure 6.19: Joint precedence constraints for vent assembly.

direction. The configuration for the first stage eliminates any undercut and makes the parting line flattest. It is shown in Figure 6.20 and is given by:

$$\bar{T}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

After the main body is molded, we have the choice of molding the vanes in

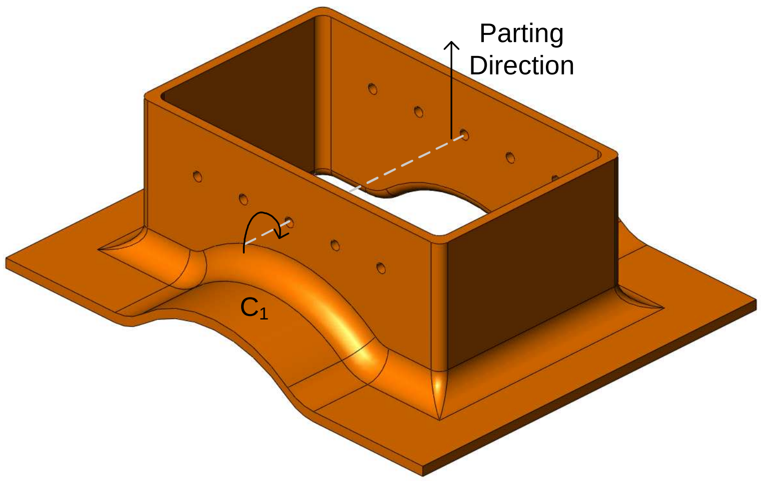


Figure 6.20: First molding stage for vent assembly.

any order. But since all the vanes are made of the same material, they can be molded in a single stage. Figure 6.21 shows the configuration of the assembly for the second molding stage. It must be noticed that in the initial configuration shown in Figure 6.18, the vanes cast shadow on each other. They need to oriented vertically as shown in Figure 6.21 to avoid this problem. The configuration for the second stage is given by:

$$\Theta_2 = [30°, 30°, 30°, 30°, 30°]$$

$$\bar{T}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6.5.3 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

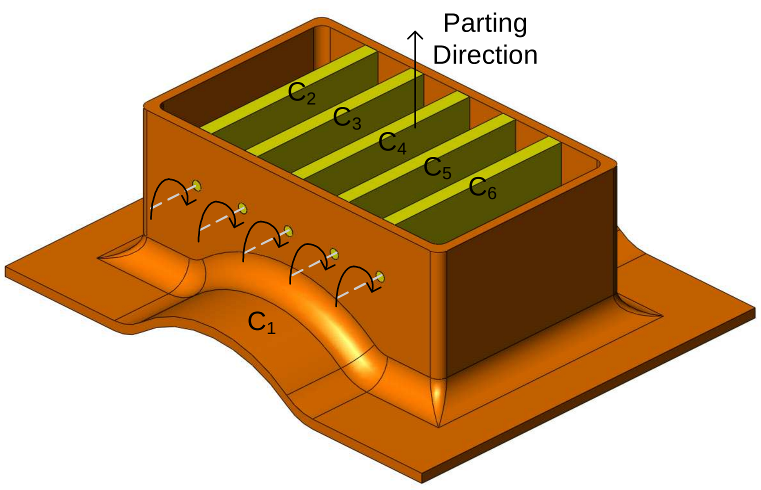


Figure 6.21: Second molding stage for vent assembly.

other. Universal joints are common wherever a driveshaft needs to turn a corner; a driveshaft 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 driveshafts of automobiles [Wiki05b].

Figure 6.22 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.

The revolute joints in the universal joint are similar to that in plan C which will be used for this example. Plan C specifies that pin must be molded after hole, i.e., the link must be molded after the shafts. Figure 6.23 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

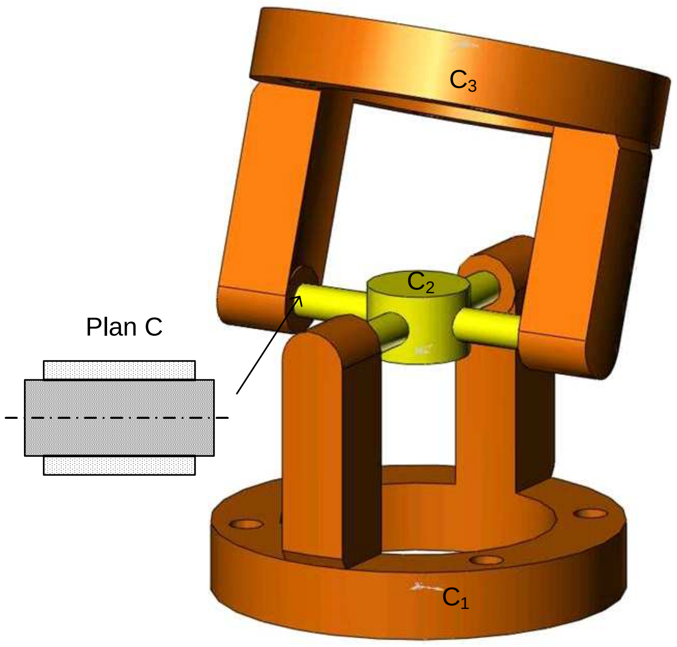


Figure 6.22: Universal joint.

$$\begin{tikzcd}
C_1 \arrow[dr] & & C_3 \arrow[dl] \\
& C_2 &
\end{tikzcd}$$

Figure 6.23: Joint precedence constraints for universal joint.

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

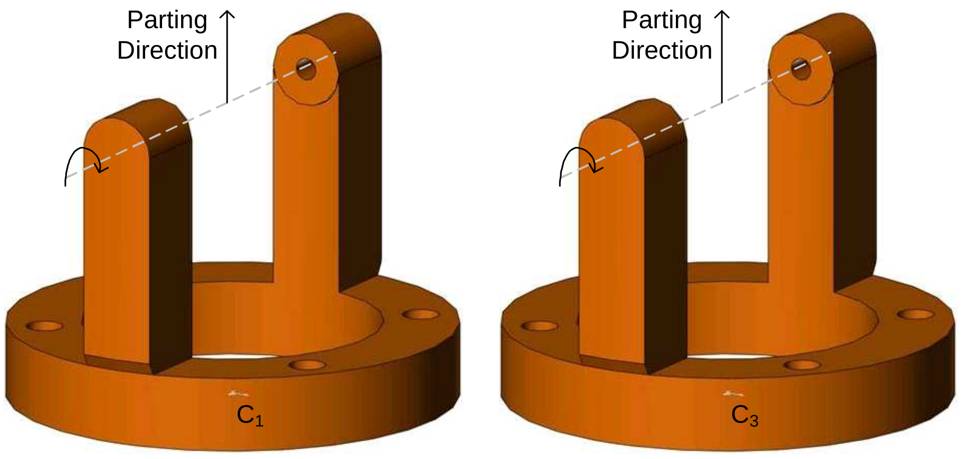


Figure 6.24: First molding stage for universal joint.

line flattest. It is shown in Figure 6.24 and is given by:

$$\bar{T}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

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 plan C 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 inverse kinematics scheme described in Section 6.4.5. Figure 6.25 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. The configuration parameters for the second stage are:

$$\Theta_2 = [0°, 90°]$$

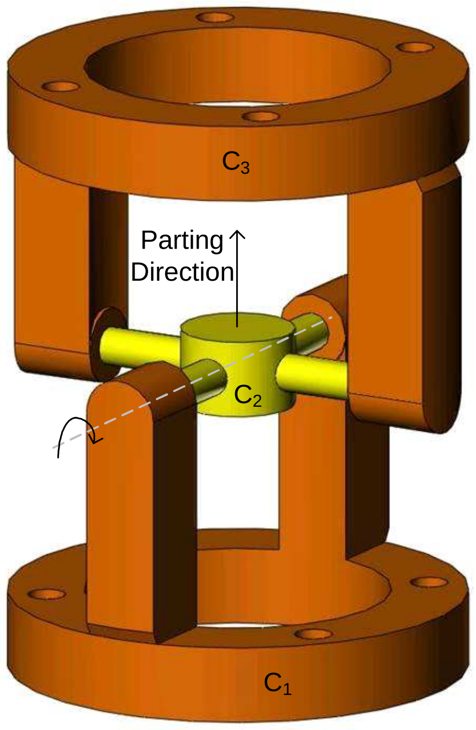


Figure 6.25: Second molding stage for universal joint.

$$\bar{T}_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6.6 Summary

Using multi-stage molding for articulated devices is a relatively new technology, so software tools for generating molding plans do not exist. The available software tools only handle traditional molding. The molding plan is generated manually by the mold designer based on prior experience. This is difficult because it involves examining a large number of combinations and solving complex geometric reasoning problems.

This chapter 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 chapter 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. This algorithm can be adapted for assembly planning where monotonocity assumptions do not hold.

Chapter 7

FINDING MOLD-PIECE REGIONS

This chapter describes the algorithm for finding the mold-piece regions on a part. The material presented in this chapter is expanded version of the material published in [Dhal03] and [Priy06b].

Section 7.1 defines the problem of finding mold-piece regions. Section 7.2 presents an object-space algorithm, which can be executed on the central processing unit (CPU) of a computer, while Section 7.3 presents an image-space algorithm, which can be executed on the graphics processing unit (GPU) of a computer. The image-space algorithm seeks to exploit the computational power of the current generation computer graphics hardware.

7.1 Problem Definition

Kwong [Kwon92] and Chen et al. [Chen93] first formulated the condition for demoldability of a mold surface in terms of ray-accessibility of the part surface it is forming. If a surface is accessible along a direction, it is also demoldable along that direction. This geometric property is illustrated in Figure 7.1 and can be formally stated as follows.

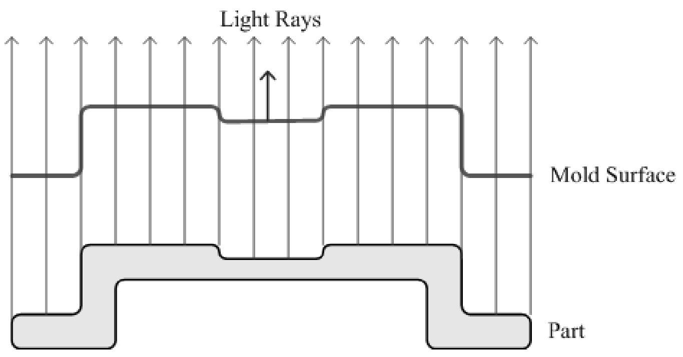


Figure 7.1: Accessibility of a surface means demoldability

Theorem 7.1. *If a surface S on an object O is completely and globally ray-accessible from a direction $\vec{d}$, the mold surface forming S can be translated to infinity in the direction $\vec{d}$, without intersecting the interior of O.*

However, some surfaces for which, the above property does not hold, may also be moldable. Those surfaces that are not ray-accessible from any direction, may be formed by special cores, called split cores. Split cores are disassembled in two steps. Figure 7.2 illustrates the functionality of a split core. This dissertation does not handle such class of objects.

A *Mold-Piece Region* of a part is a set of part facets that can be formed by a single mold piece. There can be four types of mold-piece regions – core region (C_o), cavity region (C_a), both region (B_o), and undercut region (U_c). Figure 7.3 shows various mold-piece regions for a part. Given a polyhedral object P and a parting direction $\vec{d}$, each set of part facets has the following property:

1. C_o, which is formed by core, is accessible from $+\vec{d}$, but not $-\vec{d}$

2. C_a, which is formed by cavity, is accessible from $-\vec{d}$, but not $+\vec{d}$

3. B_o, which can be formed by either of them, is accessible from both, $+\vec{d}$ and

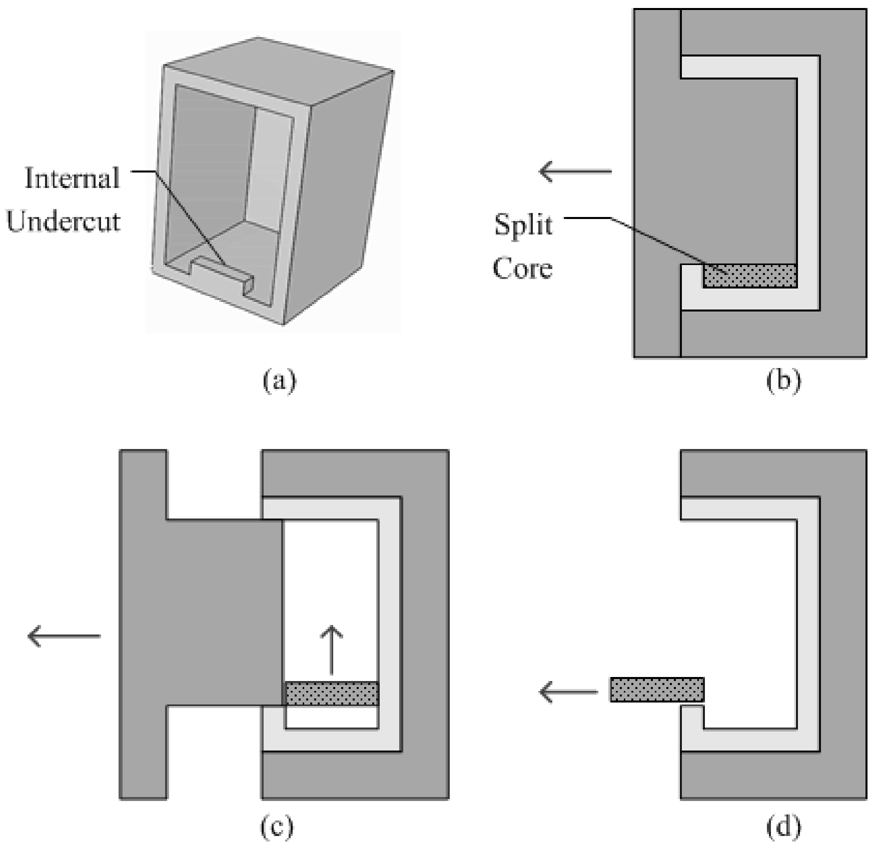


Figure 7.2: Split core; (a) a part with inaccessible surface; (b) the inaccessible surface is formed by the split core; (c) split core moves away from the undercut; (d) split core moves along the main parting direction

$-\vec{d}$

4. U_c, which cannot be formed by either of them, is not accessible from either $+\vec{d}$ or $-\vec{d}$

Hence, the problem of finding mold-piece regions reduces to performing accessibility analysis of P along $+\vec{d}$ and $-\vec{d}$ and decomposing the part facets F into four sets C_o, C_a, B_o, and U_c.

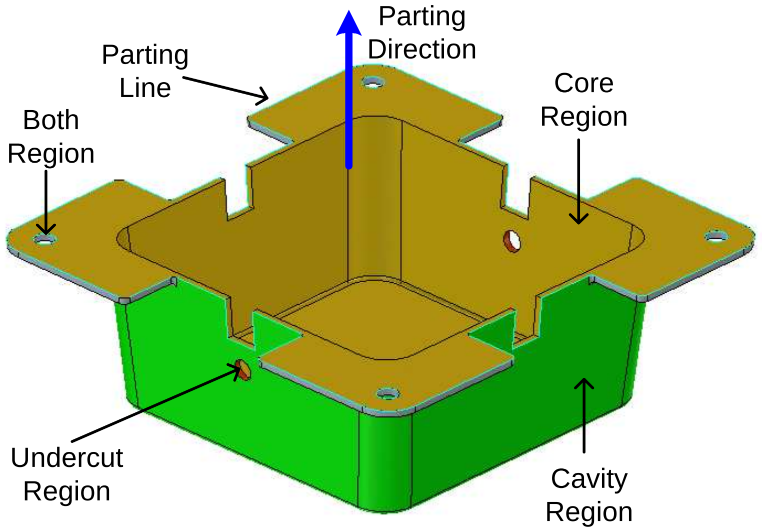


Figure 7.3: Mold-Piece Regions

7.2 Object-Space Algorithm

This section presents an object-space algorithm, which can be executed on the central processing unit (CPU) of a computer. As explained above, finding the mold-piece regions of a part is equivalent to performing the accessibility analysis of the part along the given parting direction. Suppose F is the set of facets on the part and $\vec{d}$ is the given parting direction. We determine the accessibility of each facet $f \in F$ along both $+\vec{d}$ and $-\vec{d}$ directions. Depending on its accessibility, it is assigned to C_o, C_a, B_o, or U_c. Section 7.2.1 describes an algorithm to test whether a facet is accessible from a particular direction. Section 7.2.3 describes an approach to robustly determine the accessibility of near-vertical facets (whose normals are near-perpendicular to the parting direction). Section 7.2.4 presents ideas based on geometric properties of polyhedral objects for efficient implementation of the algorithm.

7.2.1 Determining the Accessibility of a Facet

This section describes an algorithm to test whether a facet is accessible from a particular direction.

Definition 7.1. *A facet f on a polyhedral object O is accessible in a direction $\vec{d}$, if for every point p on f, the ray starting from p to infinity in the direction $\vec{d}$ does not intersect the interior of O.*

For a facet to be accessible from a direction, it needs to pass two tests – Orientation test and Obstruction test. The orientation test looks at the orientation of the facet normal with respect to the test direction. Since facets form the outer surfaces of a solid object, they have an orientation convention: *inside* and *outside.* All facets have material on one side (the "inside"), and air on the other (the "outside"). The normal to a facet always points away from the inside region. With respect to a direction, a facet is *back-facing* or *front-facing.* If the dot product of a facet's normal $\vec{n}$ and a viewing direction $\vec{d}$ is negative, the facet is inaccessible, and is called back-facing. If the dot product is non-negative, the facet is potentially accessible, and is called front-facing.

The obstruction test further checks whether a potentially accessible front-facing facet f is obstructed by another facet f' on the object. If none of the facets on the object obstructs f, it is accessible. The basic idea behind this algorithm is based on the following definition.

Definition 7.2. *A facet f_2 obstructs a front-facing facet f_1 in a direction $\vec{d}$, if for any point p on f_1, the ray starting from p to infinity in the direction d pierces f_2.*

Though this is a mathematically rigorous definition, it is computationally impractical since there would be infinite points on f_1 from which a ray needs to be shot. A mathematically equivalent but computationally practical procedure is described below.

We are interested in checking whether a facet f_2 obstructs a front-facing facet f_1 in a direction $\vec{d}$. Assume that the entire scene is rotated so that the direction of access $\vec{d}$ is aligned with +z-direction and the facets f_1 and f_2 are orthogonally projected along the z-axis onto the plane z=0. f_1 is projected onto S_1 and f_2 is projected onto S_2. The intersection of S_1 and S_2 is called S. Figure 7.4 shows the above arrangement. Following can be deduced from the intersection region S:

- If S is empty, then the projections do not overlap and there is no obstruction.
- Any point $p(x, y, 0)$, lying in S, corresponds to the point $p_1(x, y, z_1)$ in f_1 and the point $p_2(x, y, z_2)$ in f_2. The point p_1 is obstructed by the point p_2 if and only if $z_2 > z_1$. If this true is for:
  - all points in S: "f_2 obstructs f_1". This thesis does not differentiate between partial and complete obstruction. A facet is marked as obstructed even if it is partially obstructed.
  - some points in S and false for others: "f_1 and f_2 intersect each other". However, this cannot be true for any two facets belonging to the boundary of a valid solid. The boundary of a valid solid is a regular surface that does not self-intersect. Therefore, if $z_2 > z_1$ for any point in S, it must be true for all points in S.
  - no point in S: "f_2 does not obstruct f_1".

Based on the above observation, following algorithm is developed to check if a front-facing facet f_1 is obstructed by another facet f_2.

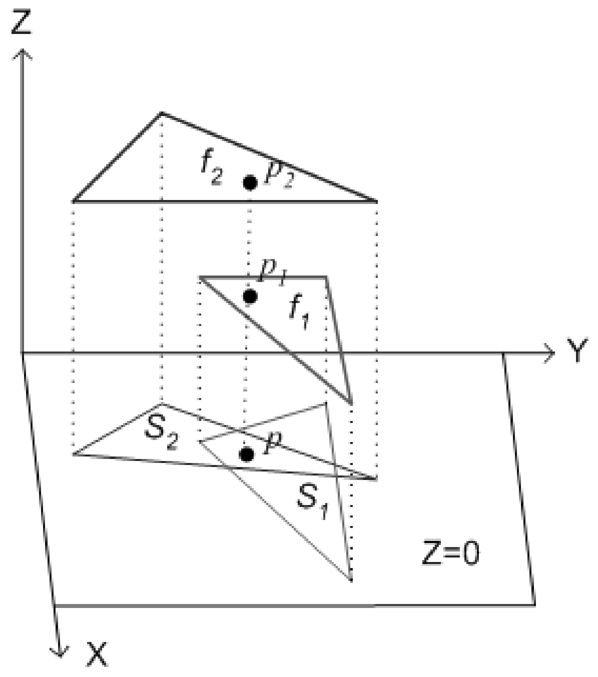


Figure 7.4: Projecting facets on the viewing plane

Algorithm IsObstructing

Input:

- Facets f_1 and f_2
- Direction of access $\vec{d}$

Output: True, if f_2 obstructs f_1 in $\vec{d}$, else False.

Comments: This algorithm assumes that f_1 is front-facing, i.e. the dot product of $\vec{d}$ and f_1's normal $\vec{n}$ is non-negative. It returns True even for partial ostruction.

Steps:

1. Transform f_1 and f_2 such that $\vec{d}$ is aligned with +z-direction.
2. If f_2 is completely below f_1, return False.

3. Project f_1 and f_2 orthogonally along the z-axis onto the plane z=0. f_1 and f_2 are projected onto S_1 and S_2 respectively.

4. Check overlap of S_1 and S_2 using separating axis method (described in Section 7.2.2).

5. If S_1 and S_2 do not overlap, return False.

6. Else do

    (a) If f_2 is completely above f_1, return True.

    (b) Else do

        i. Find an intersection point p of S_1 and S_2.

        ii. Map $p(x, y, 0)$ to $p_1(x, y, z_1)$ on f_1, and p_2 on $f_2(x, y, z_2)$.

        iii. If $z_2 > z_1$, return True else return False.

7.2.2 Separating Axis Method for Convex Polygons

Algorithm IsObstructing uses the *method of separating axis* to check whether or not the two projected triangles overlap. The main focus of this method is on the *test intersection* geometric query, a query that just indicates whether or not an intersection exists. The problem of computing the set of intersection is denoted as *find intersections* geometric query and is generally more difficult to implement than the test intersection query. Information from the test query can help determine the contact set that the find query must construct.

A test for non-intersection of two convex objects is simply stated: If there exists a line for which the intervals of projection of the two objects onto that line do not

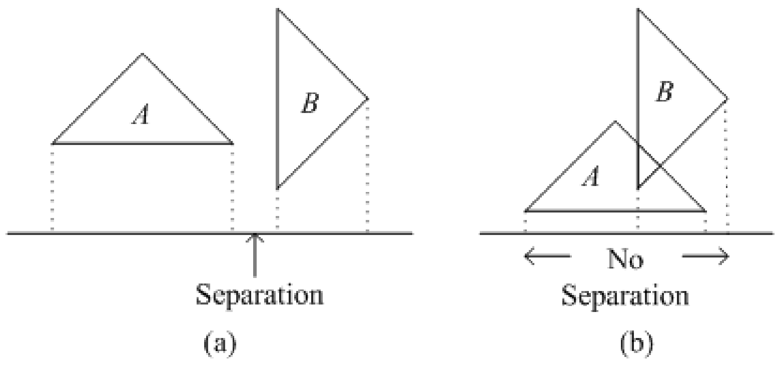


Figure 7.5: Separating Axis Method for convex polygons

intersect, then the objects do not intersect [Gott96]. Such a line is called a separating line or, more commonly, a separating axis. For a pair of convex polygons in 2D, only a finite set of direction vectors needs to be considered for separation tests. That set includes the normal vectors to the edges of the polygons. Figure 7.5(a) shows two nonintersecting polygons that are separated along a direction determined by the normal to an edge of one polygon. Figure 7.5(b) shows two polygons that intersect (there are no separating directions). If the separating axis method indicates an intersection, the actual intersection points can be found along the edges of the polygons.

7.2.3 Handling Near-Vertical Facets

We need to robustly handle facets whose normals are very close to being perpendicular to the parting direction. These near-vertical facets are usually produced as a result of the approximation introduced by faceting vertical curved surfaces. Figure 7.6 shows a cylindrical surface that has been faceted. It can be seen that for a direction of access along the cylinder axis, some of the facets are back-facing and hence inaccessible. From the user's point of view, this is obviously not the

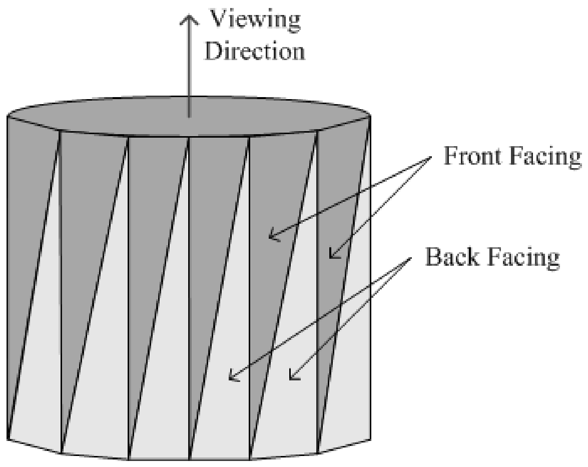


Figure 7.6: Surface tolerance problems with near-vertical facets

desired solution.

To find facets accessible from a direction $\vec{d}$, the facets on the part boundary are divided into three categories:

1. Front-Facing: $\vec{d}.\vec{n} \geq \tau$
2. Back-Facing: $\vec{d}.\vec{n} \leq -\tau$
3. Near-Vertical: $|\vec{d}.\vec{n}| < \tau$

Where $\vec{n}$ is the facet normal and τ is normal tolerance whose value is dependent on the surface tolerance introduced by faceting or that of the part. It is normally set to 2-3 degrees.

Front-facing and near-vertical facets are potentially accessible facets. Determining the accessibility of front-facing facets has already been discussed in the previous sections. Algorithm IsObstructing is used to check if a facet obstructs a front-facing facet. The accessibility of a near-vertical facet is determined in two steps:

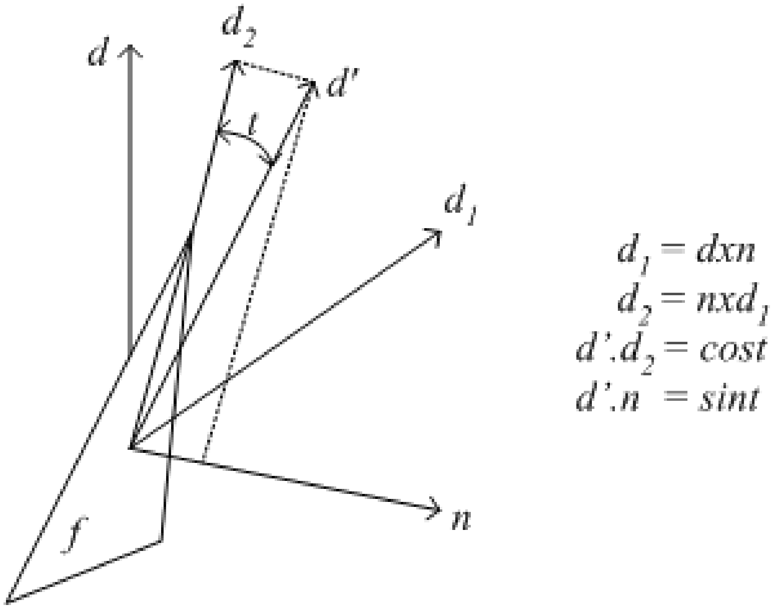


Figure 7.7: Compensating surface tolerance by rotating viewing direction

1. The direction of access $\vec{d}$ is slightly rotated to $\vec{d'}$ such that the near-vertical facet f becomes front-facing for $\vec{d'}$. The procedure to rotate the direction of access is illustrated in Figure 7.7.

2. Algorithm IsObstructing is used to check if any facet obstructs f in direction $\vec{d'}$. If f is accessible in $\vec{d'}$, it is assumed to be accessible in $\vec{d}$ also.

If the above procedure is not applied to near-vertical facets, then many of those facets may be wrongly rejected as inaccessible. A near-vertical facet is not rejected just because it is back-facing by a small amount. They are rejected only if a facet obstructs it in a direction very close to the original direction of access.

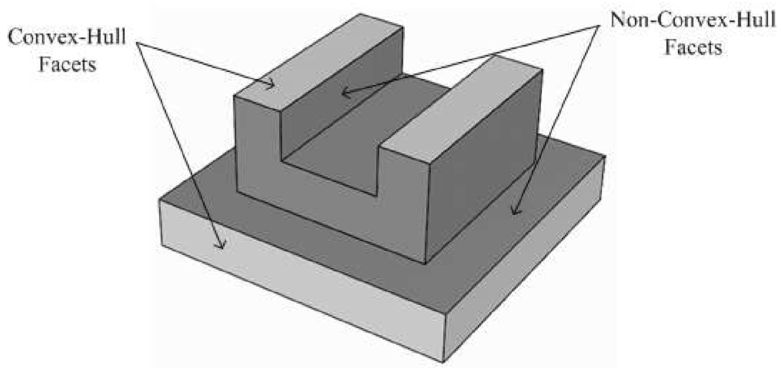


Figure 7.8: Convex-Hull facets and Non-Convex-Hull facets

7.2.4 Pruning Unnecessary Obstruction Tests

If there are n facets on the object, to determine the accessibility of a facet from a certain direction, the algorithm IsObstructing has to be called $O(n)$ times making the running time of the algorithm for finding mold-piece regions $O(n^2)$. However, this time complexity is again a loose-bound complexity. It can be made efficient by pruning out unnecessary obstruction tests.

A faceted object boundary consists of two types of facets: convex-hull facets and non-convex-hull facets. *Convex-hull facets* are those facets on the part that are also on the convex hull of the part. All the facets on the part other than convex-hull facets are called *non-convex-hull facets*. Figure 7.8 shows examples of convex-hull and non-convex-hull facets. Connected sets of non-convex-hull facets form concave regions (pockets in [Chen93]).

Theorem 7.2. *The global accessibility cone of a convex-hull facet is a hemisphere generated using the direction normal of the facet as its pole [Chen93].*

Theorem 7.3. *A non-convex-hull facet can be blocked only by a non-convex-hull facet present in the same concave region. A ray emanating from a point in a*

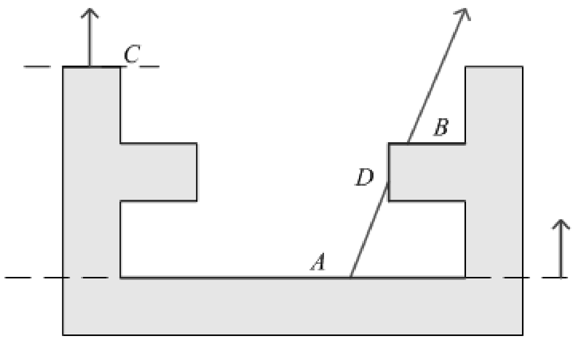


Figure 7.9: Polyhedral-object accessibility properties

concave region will either intersect a facet in the same concave region or go to infinity [Chen93].

Theorem 7.4. *The accessibility of a facet f_1 can be obstructed by another facet f_2, only if f_1 and f_2 see each other. Since the boundary of the object is continuous, if the facets cannot see each other, there will always be at least one facet in between the two facets. A facet f_1 can see another facet f_2, if any vertex of f_2 lies in the outer half-space of f_1 [Dhal03].*

Figure 7.9 illustrates the properties stated above. A facet can see all facets present in its outer half-space. Facet A can see facet B, but B cannot see A. Therefore any ray starting from A cannot reach B before piercing another facet on the object. Facet C being a convex-hull facet cannot see any other facet present on the object.

The following steps are taken for an efficient implementation of the algorithm for finding mold-piece regions:

1. Since the accessibility cone of a convex-hull facet is always a hemisphere (Theorem 7.2), a front-facing convex-hull facet is always accessible. Hence, we only need to perform obstruction tests for non-convex-hull facets.

2. Due to Theorem 7.3, obstruction tests need to be performed only for those facet pairs that are present in the same concave region. Therefore, the object boundary is subdivided into different concave regions, and accessible facets are found in each concave region separately.

3. A ray emanating from a point in a concave region in the viewing direction will either pierce a back-facing facet present in the same concave region, or go to infinity. If the point is obstructed, the ray pierces a back-facing facet before entering the interior of the object. Therefore, if a front-facing facet f in a concave region is not obstructed by any of the back-facing facets present in the same concave region, f is accessible. This further implies that if a concave region has no back-facing facets for a direction of access, all facets present in the concave region are accessible from the direction.

4. If a back-facet f_2 is completely below a front-facing facet f_1, then f_2 cannot obstruct f_1.

5. Due to Theorem 7.4 obstruction tests need to be done only for those facet pairs that can see each other.

The above steps can be summarized as follows. We only need to perform obstruction tests for front-facing non-convex-hull facets. To determine the accessibility of a non-convex-hull front-facing facet f, it is sufficient to perform the obstruction tests only with those back-facing facets that:

1. are present in the same concave region,

2. are not below f, and

3. can see f.

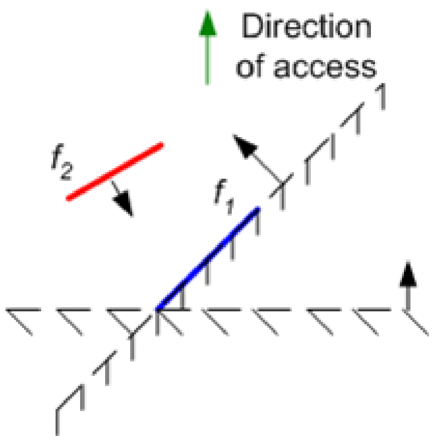


Figure 7.10: Obstruction test need not be performed for all facet pairs

This pruning scheme is illustrated in Figure 7.10. The facet f_1 can only be obstructed by facets present in the region formed by the intersection of two half-spaces A and B. Moreover, a facet f_2 present in that half-space can obstruct f_1 only if it is present in the same concave region as f_1, and can see f_1.

In addition to using these properties, we have used a hierarchical axis-aligned bounding box tree to store the facets. This data structure allows us to quickly prune many redundant obstruction tests.

7.2.5 Implementation and Results

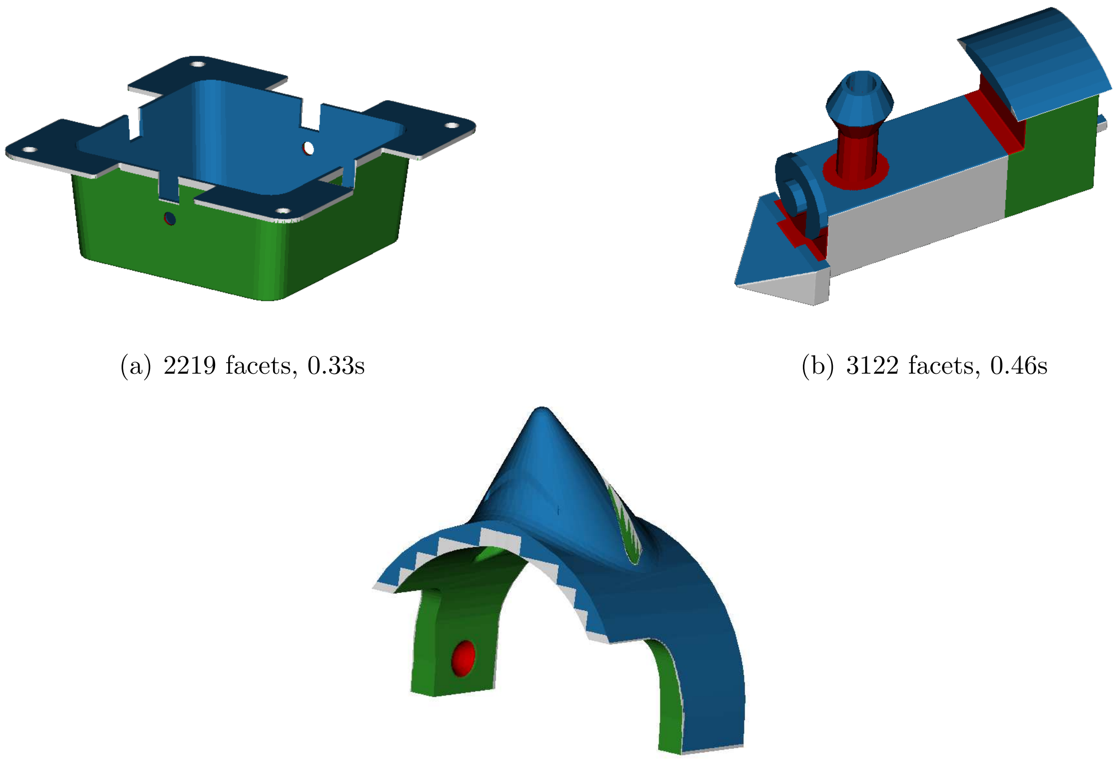
We have implemented the algorithm for finding mold-piece regions described above in C++. Other libraries used in the system are GNU Triangulated Surface Library [GTS], OpenGL, and Microsoft Foundation Classes [MFC]. GTS is an open-source software library intended to provide a set of data structures and functions to deal with three-dimensional surfaces meshed with interconnected triangles. OpenGL provides application programming interface (API) for rendering 2D and

3D graphics primitives. MFC is a windowing toolkit for Microsoft Windows platform. Our software takes the input for three-dimensional polyhedral objects in Stereolithography (STL) file format. The output can be rendered is both graphical and file-based. It has been successfully tested on more than 50 industrial parts.

Figure 7.11 shows the screenshot of three example parts. The reported running times were achieved by running the software on a Intel(R) Pentium(R) M Processor 1.80 GHz with 2 GB RAM. It can be seen that although the number of facets on part C is more than that on part A and Part B, it takes lesser time to calculate the mold-piece regions. This suggests that the running time is not only dependent on the number of facets. Due to our pruning techniques, it is also dependent on the convexity of the part and overall distribution of facets. To test the average case running time, we need to test it with parts having similar geometry. We took a 3D scanned model of a face and progressively simplified it to desired number of facets. We used the freely available QSlim software [QSlim] to do the simplification. Figure 7.12 shows the performance result obtained for the progressively simplified face. It can be seen that the running time grows almost linearly.

7.3 Image-Space Algorithm

The rapid increase in the performance of graphics hardware, coupled with recent improvements in its programmability, have made graphics hardware a compelling platform for computationally demanding tasks in a wide variety of application domains [Owen05]. Software that have traditionally been running on CPU are being migrated to GPU in increasing numbers. These software belong to a diverse set of applications ranging from audio and signal processing [Whal05] to data


(a) 2219 facets, 0.33s

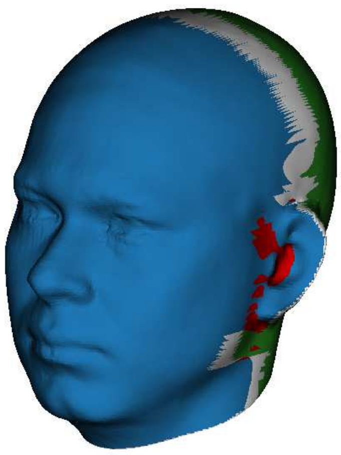
(b) 3122 facets, 0.46s

(c) 5716 facets, 0.24s

Figure 7.11: Screenshots of three example parts. The color scheme for highlighting is following. Core region is blue, cavity region is green, both region is gray, and the undercuts are red. Number of facets and obtained running time is reported against each subfigure.

mining [Govi05].

This section describes our algorithm for finding and highlighting the mold-piece regions. Section 7.3.1 gives an overview of our approach. Section 7.3.2 and Section 7.3.3 address the robustness issues of the algorithm. Section 7.3.4 presents an algorithm for transferring the results from the GPU to CPU.



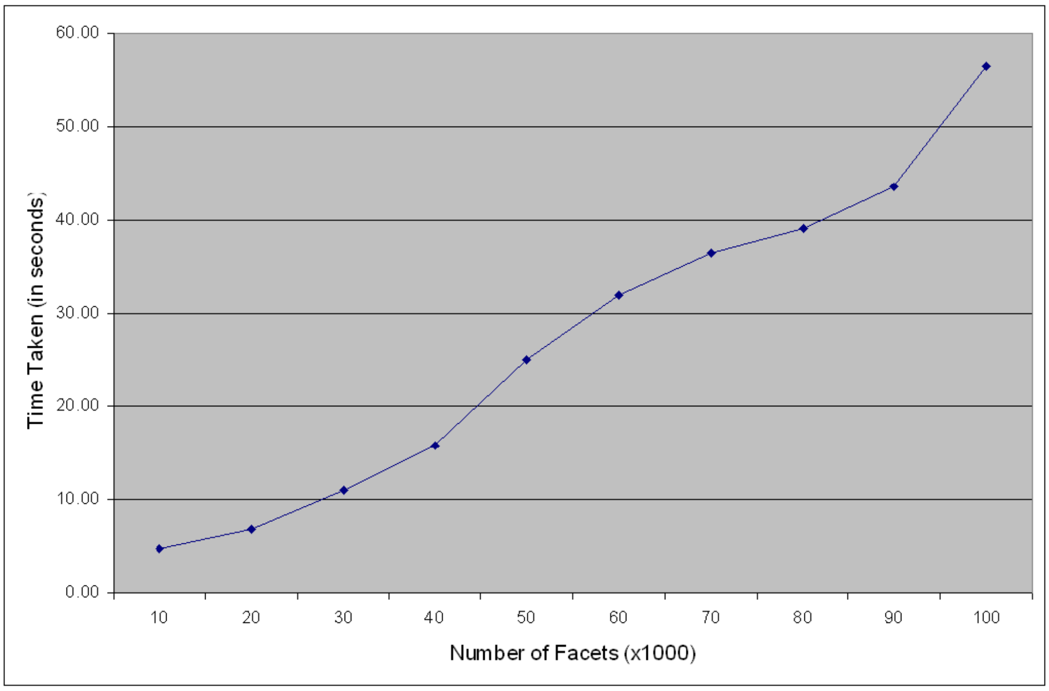


Figure 7.12: Performance result for a progressively simplified part.

7.3.1 Overview of Approach

We use programmable GPUs to highlight the mold-piece regions on a part. The basic idea is very similar to shadow mapping described in Section 2.3. The given part is illuminated by two directional light sources located at infinity in the positive and negative parting directions. The regions that are lit by the upper and lower lights are marked as 'core' and 'cavity' respectively. The regions lit by both the lights are marked as 'both', while the regions in shadow are marked as 'undercuts'.

For a given parting direction, our approach highlights the mold-piece regions on a part in two steps:

1. *Preprocessing*: We create two shadow maps by performing the following procedure. First the part is rendered with the camera placed above the part and view direction along the negative parting direction. The resulting z-buffer is transferred to a depth texture (shadow map). The current orthogonal view matrix is also stored for the next step. The same procedure is repeated with the view direction along positive parting direction.

2. *Highlighting*: The user can then rotate the camera and examine the mold-piece regions of the part from all directions. A vertex program transforms the incoming vertices using the two model-view matrices stored in the preprocessing stage. The fragment program determines the visibility of each incoming fragment by comparing its depth with the depth texture values stored in the preprocessing stage and colors it accordingly. Fragments visible along positive parting direction (core region) are colored blue while those accessible along negative parting direction (cavity region) are colored green. Fragments visible along both directions (both region) are colored gray and invisible fragments (undercuts) are colored red.

If the algorithm is implemented as described, all the vertical facets will be reported as undercuts. Also since our method is based on shadow mapping, it is prone to self-shadowing. Section 7.3.2 and Section 7.3.3 describe techniques to handle these issues.

7.3.2 Handling Near-Vertical Facets

There is a slight difference between the notion of visibility in computer graphics and accessibility. The mathematical conditions for visibility and accessibility of a facet with normal $\vec{n}$ in direction $\vec{d}$ are the following:

Visible if: $\vec{d} \cdot \vec{n} > 0$

Accessible if: $\vec{d} \cdot \vec{n} \geq 0$

In other words, a facet perpendicular to a direction (vertical facet) is not visible, but accessible which means all the vertical facets will be reported as undercuts.

In addition to vertical facets, we also need to robustly handle facets whose normals are very close to being perpendicular to the parting direction. These near-vertical facets are usually produced as a result of the approximation introduced by faceting vertical curved surfaces. Figure 7.6 shows a cylindrical surface that has been faceted. It can be seen that for a direction of access along the cylinder axis, some of the facets are back-facing and hence inaccessible. From the user's point of view, this is obviously not the desired solution.

The robustness problems in geometric computations are usually handled by slightly perturbing the input. But we cannot adopt this approach here as perturbing the vertices of the part will change it's appearance on the computer screen. We solve this problem by visibility sampling. To determine the accessibility of a rasterized fragment, the neighborhood of the corresponding texel in the shadow map is sampled in the image space. If any sample passes the visibility test, the fragment is marked as accessible. Incidentally, percentage closer filtering (PCF) used to produce anti-aliased shadows does just that. PCF is briefly described in Section 2.3.2.

For a given parting direction $\vec{d}$, we divide the part facets into three categories:

1. Up facets: $\vec{d}.\vec{n} \geq \tau$
2. Down facets: $\vec{d}.\vec{n} \leq -\tau$
3. Near-vertical facets: $|\vec{d}.\vec{n}| < \tau$

where $\vec{n}$ is the facet normal and τ is normal tolerance whose value is dependent on the surface tolerance introduced by faceting the part. It is usually set between 1-2 degrees.

Up and down facets are tested for accessibility along $-\vec{d}$ and $+\vec{d}$ respectively. The near-vertical facets are tested in both the directions with PCF enabled. In our implementation, we used the OpenGL extension ARB_shadow that samples the neighborhood of a fragment and returns the average of all the depth comparisons. If the returned value is greater than zero, we mark the fragment as accessible. The PCF kernel that determines the size of the sampling neighborhood should be adjusted according to the surface tolerance of the given part. We found that 3x3 kernel (9 samples) worked fine for most of the parts.

7.3.3 Preventing Self-Shadowing

Our algorithm is based on shadow mapping. So as explained in Section 2.3.1, it is also prone to self-shadowing due to precision and sampling issues. The focus of the currently available algorithms is mainly on producing aesthetically pleasing results. They may not be physically correct. We decided not to use the most popular polygon offset technique after a thorough experimentation. We found that it is indeed very difficult to specify an appropriate bias for a part automatically. If the bias is too little, everything begins to shadow. And if it is too much, shadow starts too far back i.e., some of the fragments that should be in shadow are incorrectly lit.

We found that this problem is exaggerated in case of mechanical parts with regions of high slope. Wang and Molner [Wang94] report that even for some simple test scenes, it is impossible to find an acceptable bias. We developed an adaptation of the second depth [Wang94] technique that prevents self-shadowing and robustly handles the near-vertical facets.

Second depth technique [Wang94] is based on the observation that in case of solid objects there is always a back facet on top of a shadowed front facet. It renders only the back facets into the shadow map and avoids many aliasing problems because there is adequate separation between the front and back facets. But it may show incorrect results when used with PCF for near-vertical facets. As explained in Section 7.3.2, we use PCF to sample the neighborhood of a point on a near-vertical facet. If any sample passes the visibility test, we mark the point as accessible. Because the shadow map only partially overlaps the PCF kernels for both points A and B, they will be reported as only 50% shadowed and hence accessible. This is the intended result for point B, but incorrect for point A.

To solve this problem, we use a visibility theorem for three-dimensional polyhedral surfaces based on the results presented by Lutz Kettner [Kett99].

Definition 7.3. *An edge is a contour edge if it is incident to a front-facing facet and a back-facing facet for a given viewing direction.*

Lemma 7.1. *For a given polyhedron and a viewing direction, if the edges and facets of the polyhedron are projected into the viewing plane, the visibility of the projected facets can only change at the intersection with contour edges [Kett99].*

Definition 7.4. *Suppose e is an edge between two facets f_1 and f_2 on a polyhedron P. The edge e is convex if the dihedral angle between the two facets f_1 and f_2 in the interior of P is less than π, otherwise it is concave.*

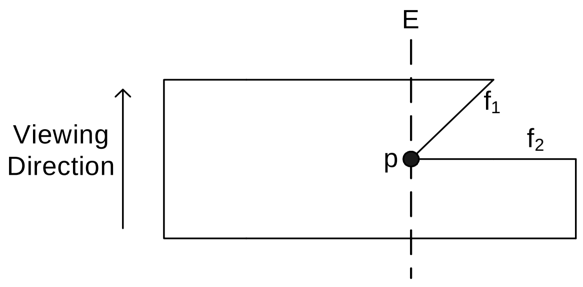


Figure 7.13: Visibility of projected facets cannot change at intersection with concave contour edges

Theorem 7.5. *For a given polyhedron P and a viewing direction $\vec{d}$, if the edges and facets of the polyhedron are projected into the viewing plane, the visibility of the projected facets does not change at the intersection with concave contour edges.*

Proof. Suppose e is a concave contour edge between a back-facing f_1 and a front-facing facet f_2 as shown in Figure 7.13. Consider a plane E containing e and perpendicular to the viewing direction $\vec{d}$. Consider a point p on the edge e. Plane E separates the neighborhood of point p into two regions – left and right of E. The region to the left of E lies in the interior of P and hence inaccessible. The region on the right is blocked by f_1. Hence the visibility of projected facets cannot change at the intersection with concave contour edges. □

We exploit the above theorem that the visibility of projected facets cannot change at the intersection with *concave* contour edges. When creating the shadow map, we also render thick concave contour edges along with the back facets. As can be seen in Figure 7.14(b), now that the shadow map fully overlaps the PCF kernel for point A, it will be correctly reported as fully shadowed and hence marked as inaccessible. It can also be seen that thickening the concave contour edges does not affect the accessibility of point B.

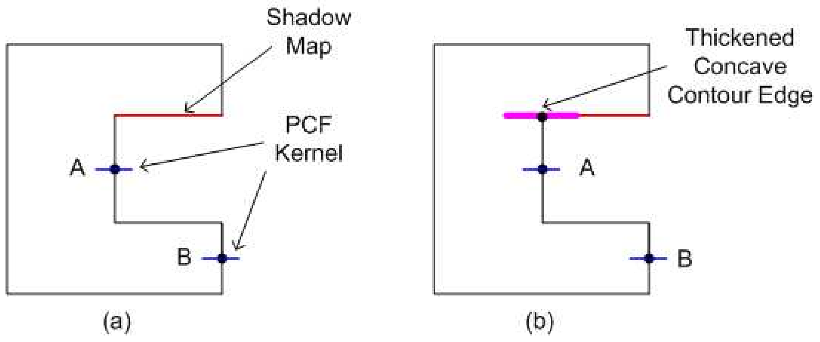


Figure 7.14: The problem with the second-depth technique when used with PCF (the PCF kernel and the contour edge has been exaggerated for illustration purposes); (a) Both point A and point B are reported as only 50% shadowed and hence accessible; (b) The problem is solved by rendering thick concave contour edges into the shadow map

7.3.4 Transferring Results from the GPU to CPU

The previous sections describe how to find and highlight the mold-piece regions using GPUs. This section describes how the information on mold-piece regions can be transferred back to the CPU for other purposes such as designing molds. We describe a simpler two-pass algorithm to accomplish the same.

We first assign a unique ID (color) to each facet of the given part. Almost all the currently available graphics cards support at least 24-bit color palette that can generate over 16 million unique colors. Then we follow the following procedure to obtain the results on the CPU. The part is first rendered with the camera placed above the part and view direction along the negative parting direction. The resulting frame buffer (image) is read back to the CPU. The facets whose IDs are present in the resulting image constitute the 'core' region. The same procedure is followed with the view direction along the positive parting direction to obtain the ‘cavity’ region. The facets missing from both the images are undercuts.

The problem with the above approach is that it cannot find the ‘both’ region. None of the facets will be present in both the frame buffers and all of the vertical facets will be reported as undercuts because being perpendicular to the viewing direction, they cannot be rendered. But now since the part is not rendered for visualization purposes, we can perturb the vertices of the part. For both the viewing directions (negative and positive parting direction), we slightly perturb the vertices of the near-vertical facets such that it becomes a front-facing facet for that viewing direction and hence an eligible candidate for being rendered. This perturbation can be done by either the CPU or by a vertex program loaded on the GPU. The perturbation scheme is illustrated in Figure 7.15. It is similar to adding a draft to the near-vertical facets. A reference plane is first located at the topmost vertex with respect to the viewing direction and then each vertex on the near-vertical facets is slightly moved along the surface normal at that point. The perturbation amount is in proportion to the distance of the vertex from the reference plane and is given by $d = z.\tan(\tau)$, where τ is a small user-defined angle, which depends on the average length of facets and resolution of the frame buffer. We found that for a 512x512 buffer, $\tau = 0.5°$ was appropriate for most of the parts.

The algorithm for transferring the results from the GPU to CPU is based on the assumption that the each facet belongs to only one mold-piece region. Sometimes a front facet needs to be split into a core and an undercut facet, or a vertical facet needs to be split into all the four mold-piece regions. A brute-force approach to overcome this limitation could be splitting each facet into very small facets. Another approach could be projecting each facet into the viewing plane and splitting them at the intersection with convex contour edges [Kett99], and

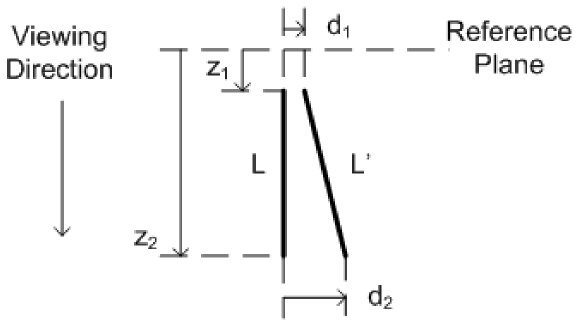


Figure 7.15: Perturbation scheme for near-vertical facets

performing trapezoidal decomposition of vertical facets [Ahn02].

7.3.5 Implementation and Results

GPUs have traditionally been used to efficiently render high-quality graphics on computer screens. Figure 7.16 shows a typical rendering pipeline. The latest GPUs allow users to load their own programs (shaders) to replace some stages of the fixed rendering pipeline. A shader can be a vertex program that replaces the vertex transformation stage, or a fragment program that replaces the fragment texturing and coloring stage.

We have implemented our algorithm for highlighting the mold-piece regions as shader programs that can be executed on programmable GPUs. The overall algorithm has been described below in the form of a pseudo code.

Pseudo Code: HighlightMoldPieceRegions
Input: Input part O and parting direction $\vec{d}$
Output: Rendered image of the part O with mold-piece regions highlighted with different colors *Steps*:

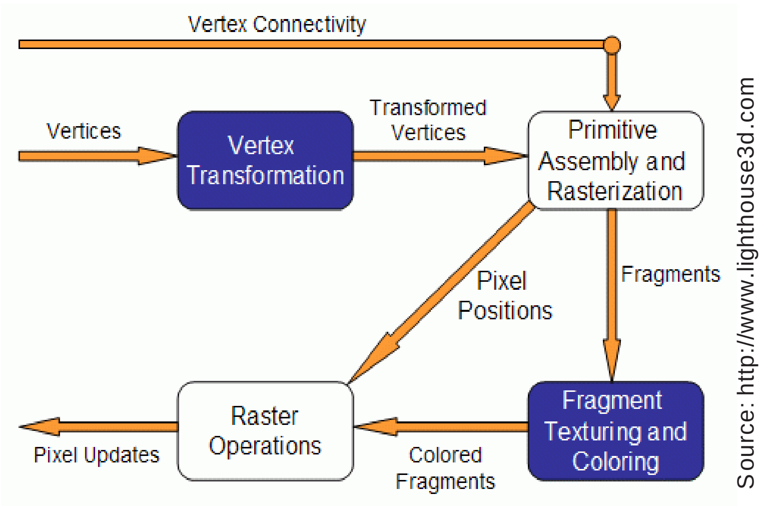


Figure 7.16: Graphics rendering pipeline

1. Build transformation matrices M^- and M^+ that place the camera above the part and view direction along $-\vec{d}$ and $+\vec{d}$ respectively.

2. Set up M^- as the model-view matrix. Render the part and thick concave contour edges. Transfer the resulting z-buffer into shadow map S^-. Similarly create the shadow map S^+ with M^+ as the model-view matrix. While creating the shadow maps, the part is rendered in a mode that allows only back faces to be rendered.

3. Load and initialize the shader programs with the transformation matrices (M^-, M^+) and shadow maps (S^-, S^+).

4. The user can now be allowed to rotate the camera and visualize the mold-piece regions from all directions. The following two steps calculate and highlight the mold-piece regions in each frame.

   (a) The vertex program transforms each incoming vertex using the current transformation matrix as well as M^- and M^+.

(b) The fragment program queries S^- and/or S^+ to determine the accessibility of each incoming fragment and highlights with appropriate color.

   i. If the incoming fragment is near-vertical, both S^- and S^+ are queried with PCF enabled to determine if it is core, cavity, both or undercut.

   ii. Else if it faces $-\vec{d}$ (up facet), S^- is queried to determine if it is core or undercut.

   iii. Else if it faces $+\vec{d}$ (down facet), S^+ is queried to determine if it is cavity or undercut.

It can be seen that the pseudo code for the fragment program consists of many conditional statements (if-then-else). The fragment processors are based on Single Instruction Multiple data (SIMD) architecture that assumes that all data elements are processed identically. The conditional statements break this assumption. The fragment processors implement the conditionals by executing both portions of the conditional statement, which means that for each conditional statement, the execution time almost doubles. The structure of the conditional block indicates that it can be moved to the CPU. We divide the part facets into three sets (up, down, and near-vertical) on the CPU and create three specialized fragment programs for each type of facet. For each frame we sequentially load each fragment program and render the facet set it handles.

We have written the shader programs in OpenGL Shading Language (GLSL). It requires four OpenGL extensions: ARB_vertex_shader, ARB_fragment_shader, ARB_depth_texture, and ARB_shadow. Other libraries used in the system are OpenGL Utility Toolkit [GLUT] and OpenGL Extension Wrangler Library [GLEW].

GLUT is a basic bare-bones windowing toolkit that supports OpenGL. GLEW is an OpenGL extension loading library. It provides efficient run-time mechanisms for determining which OpenGL extensions are supported on the target platform.

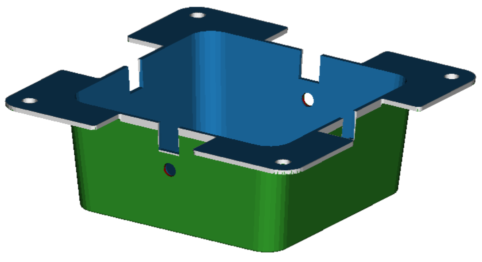
The implementation has been successfully tested on more than 50 industrial parts. It currently supports Stereolithography (STL) and Wavefront (OBJ) part files. Figure 7.17 shows the screenshot of four example parts.

Figure 7.18 shows the performance of our implementation on 128 MB NVIDIA Fx700Go card. It shows the obtained frame rates when simply rendering the part using fixed OpenGL pipeline (without highlighting) and with highlighting. It can be seen that the overhead imposed by the highlighting algorithm does not significantly affect the time taken by the GPU to render a frame. The observed drop in performance when highlight is at most one fps. In other words, the complexity of the algorithms solely depends on the time to render the given part.

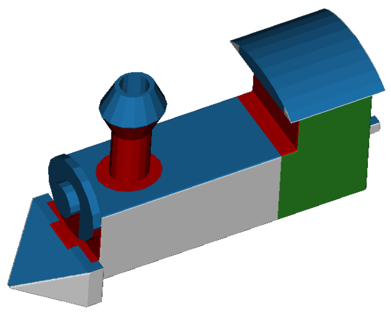
7.4 Summary

Finding mold-piece regions is a computationally intensive process. It takes a long time to robustly find mold-piece regions even for one parting direction. Efficiency is not an issue with traditional molding because parting direction is usually known and only one direction needs to be evaluated. But when generating molding plan, we need to evaluate thousands of directions for each molding stage to find a feasible molding configuration. Also, with the recent advances in three-dimensional scanning technology, the models can be very large. Therefore, we need an algorithm that is efficient as well as robust.

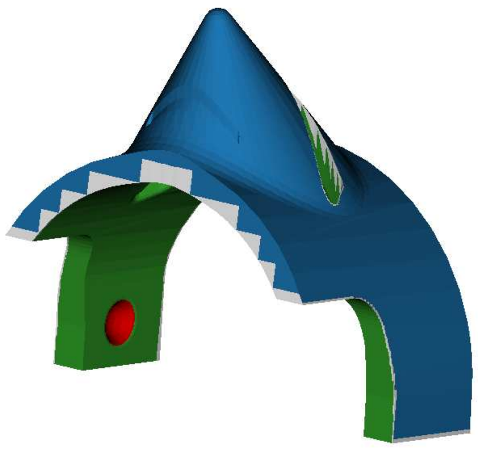
This chapter presented two geometric algorithms for finding mold-piece regions of components and assemblies. The first algorithm is an object-space algorithm,



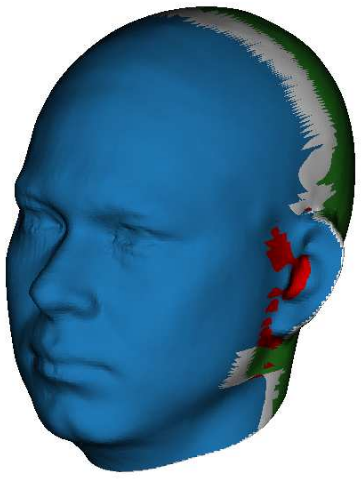
(a) 2219 facets, 60 fps



(b) 3122 facets, 58 fps



(c) 5716 facets, 47 fps



(d) 50000 facets, 5 fps

Figure 7.17: Screenshots of four example parts. The color scheme for highlighting is following. Core region is blue, cavity region is green, both region is gray, and the undercuts are red. Number of facets and obtained rendering speed is reported against each subfigure.

which runs on the central processing unit (CPU) of a computer. The algorithm handles the near-vertical facets robustly by perturbing the parting direction. The implementation utilizes geometric properties of polyhedral objects and hierarchical data structure for efficiency. The second algorithm is an image-space algorithm, which can be executed on the graphics processing unit (GPU) of a computer.

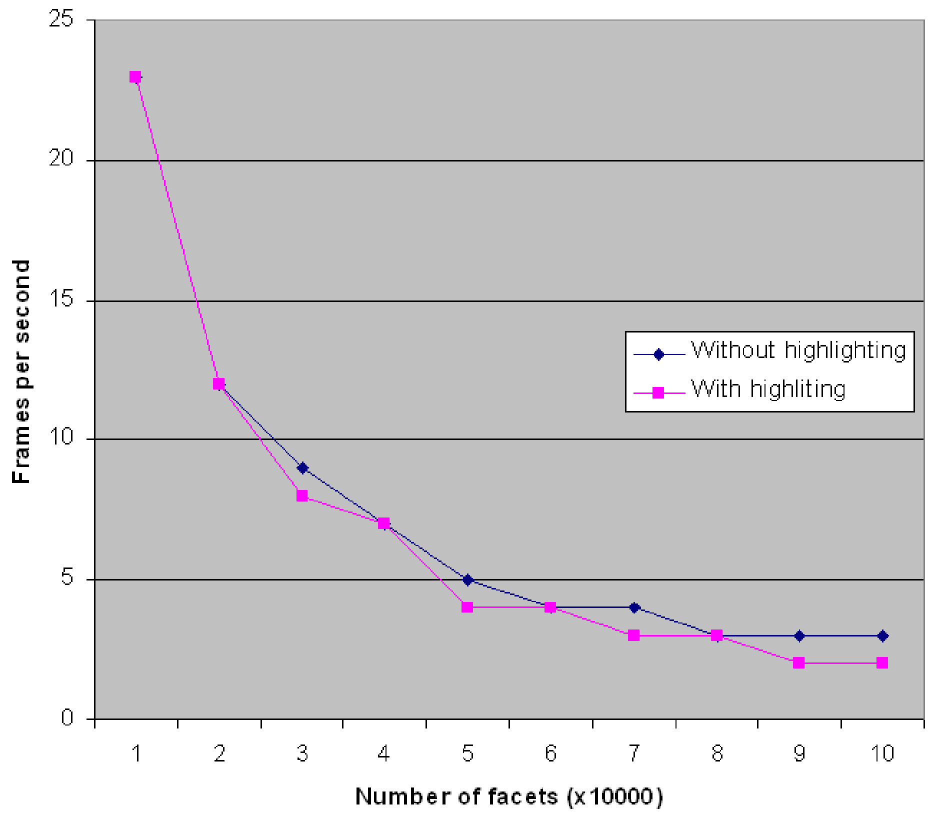


Figure 7.18: Performance of the image-space algorithm on 128 MB NVIDIA Fx700Go card. The plot shows the obtained frame rates when simply rendering the part (without highlighting) and those when also highlighting the mold-piece regions

It robustly handles the near-vertical facets by slightly perturbing the vertices on those facets and visibility sampling. It exploits the computational power offered by the GPUs. It was shown that an efficient implementation of this algorithm does not impose any significant overhead on the GPU. The mold-piece regions even for parts with more than 50,000 facets can be highlighted at interactive rates. Such a system that provides real-time information about mold-piece regions will be very useful to the part and mold designers alike. They can easily optimize the part and mold design and if needed make appropriate corrections upfront, streamlining the subsequent design steps. Both algorithms have been successfully tested with several complex components and assemblies. The results of the image-space algorithm was verified by comparing them with those obtained by the object-space algorithm. Both algorithms produce identical results. These algorithms can also be used for machining, die-casting, and sheet metal forming.

Chapter 8

CONSTRUCTING OPTIMAL SHUTOFF SURFACES

This chapter describes the algorithm for creating shutoff surface. Section 8.2 defines the problem of creating shutoff surface. Section 8.3 presents a high-level overview of the algorithm, while Section 8.4 provides a detailed description of the same.

8.1 Background

The previous chapter described the algorithm for finding mold-piece regions. Given a polyhedral object and a parting direction, the algorithm outputs four mold-piece regions:

1. Core region, which is formed by the core
2. Cavity region, which is formed by the cavity
3. Both region, which can be formed by either core or cavity
4. Undercut region, which is formed by side actions

The designer uses the mold-piece regions to create side actions and parting line. Creating side actions is mostly a manual task. There is no commercial software available to automate this task. There has been none to very few publications in this direction. Generating the shapes of the side actions requires solving a complex geometric optimization problem. Different objective functions are needed depending upon different molding scenarios (e.g., prototyping versus large production runs). Recently Banerjee and Gupta [Bane06] presented an algorithm to automatically design optimal side actions. But their algorithm is limited to a particular type of side action, commonly known as side core in molding terminology. More work is needed to handle other types of side actions, namely, split cores, lifters, etc.

After creating the side actions for undercuts, the designer is left with three mold-piece regions – core region, cavity region, and both region. The next step in the mold design process is creating parting line. A parting line of a part is a continuous closed curve on the surface of the part where core and cavity meet. Hence boundary of the core region or the cavity region is a valid parting line. However, it may not be an optimal parting line. Simple planar interfaces between mold pieces reduce mold fabrication cost and mold operation complexity. Hence it is proposed that the flattest possible parting line be found [Ravi90]. Since the 'both' region can be formed by either the core or cavity, it provides a feasible region where an optimal parting line can be located. Priyadarshi and Gupta [Priy04] presented an algorithm that uses the flatness criteria proposed by Majhi *et al.* [Majh99] to find an optimal parting line. The parting line is used to split the 'both' region into core region and cavity region. After assigning the split regions, the designer is left with only two mold-piece regions – core region and cavity region. Parting line is

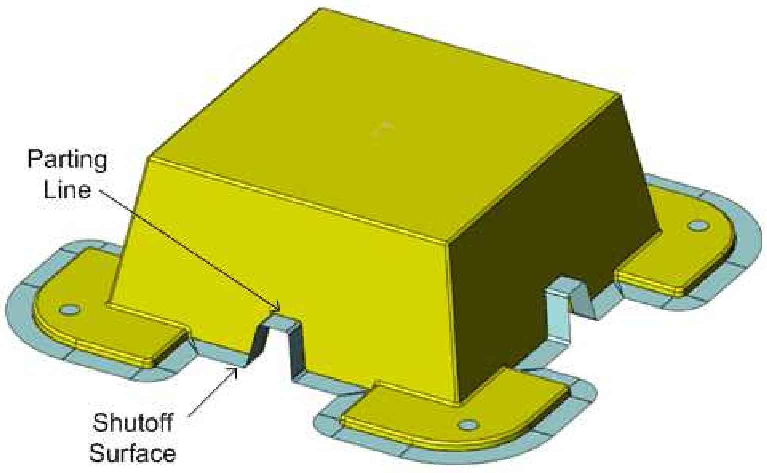


Figure 8.1: Shutoff surface for an example part.

the curve along which the two regions meet.

This chapter assumes that the undercuts have been resolved by the designer manually. The parting line has been created by the algorithm described in [Priy04], and the both region separated by the parting line have been merged into the core region and cavity region. So now the part has only two mold-piece regions – core region and cavity region.

8.1.1 Shutoff Surface

Shutoff Surface is the contact surface of the two mold pieces – core and cavity. It can be mathematically defined as the non-regularized intersection of core and cavity. It meets the part at the parting line. Figure 8.1 shows the shutoff surface for an example part.

The parting line has one outer loop and may have multiple inner loops. The inner loops correspond to the thru-holes present on the part. Figure 8.2 shows the outer and inner loops of a parting line. A shutoff surface is required for each

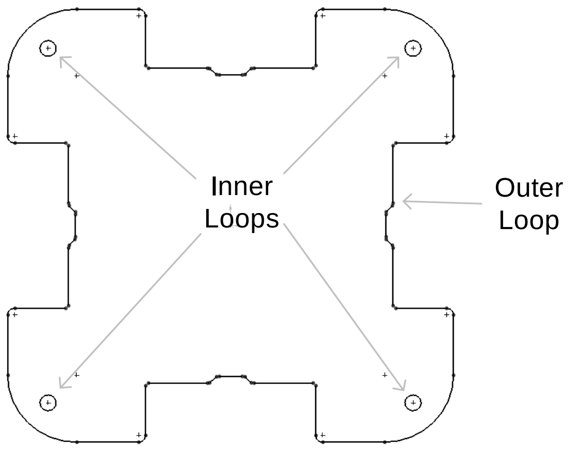


Figure 8.2: Outer and inner parting line loops

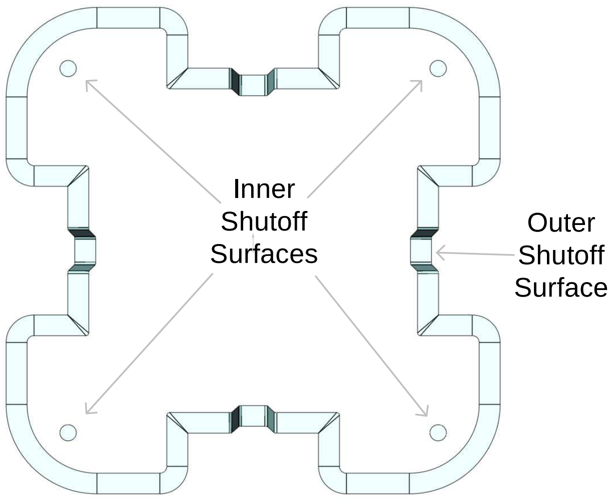


Figure 8.3: Outer and inner shutoff surfaces

parting line loop. Figure 8.3 shows the outer and inner shutoff surfaces for the parting line shown in Figure 8.2.

8.1.2 Methodology for Creating Shutoff Surface

Given an object, parting direction, mold-piece regions, and parting line, the methodology followed by the mold designers for creating shutoff surface is the following:

1. A shutoff patch is created for each inner parting loop by *filling* the loop by a surface. Filling is a surfacing method to fit a surface over the boundary defined by a closed and connected circuit of curves. The curvature of the filled surface is such that surface tangency is maintained at the loop edges.

2. The outer parting line loop L is split into multiple smooth segments $\{l_1, \ldots, l_n\}$.

3. A shutoff patch s_i is individually created for each parting line segment l_i using one of the following strategies:

    - Strategy 1 (T_1): Extruding l_i along a direction perpendicular to the parting direction (Figure 8.4a).
    - Strategy 2 (T_2): Extruding l_i along a direction tangential to the core surface (Figure 8.4b).
    - Strategy 3 (T_3): Extruding l_i along a direction tangential to the cavity surface (Figure 8.4c).

    The shutoff patch s_i, which is generated by extruding parting line segment l_i along direction d_i by distance w is defined as:

    $$s_i = \{q \in \mathbf{E}^3 : q = p + \lambda d_i, p \in l_i, 0 \leq \lambda \leq w\} \tag{8.1}$$

    In some cases, the extruded shutoff patches may overlap or intersect with each other or the part. Overlapping and intersecting shutoff surfaces create undercuts. The designer needs to manually eliminate these invalid cases.

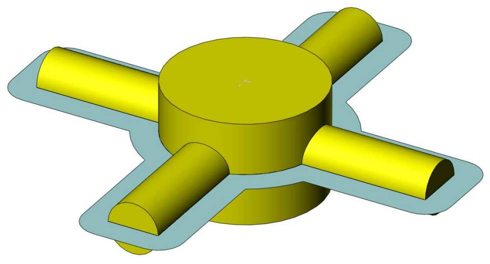
These cases are handled by trimming away the overlapping and intersecting portions of the shutoff patches.

4. The shutoff patches created for two adjacent parting segments using different strategies may not be joined together. Bridge patches are created to *stitch* such disjoint shutoff patches together (Figure 8.5).

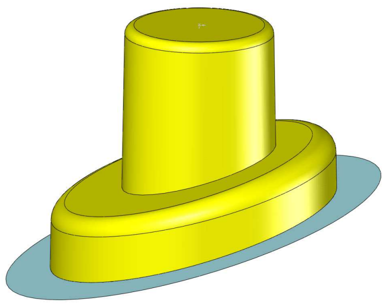
The strategies described above are standard in the molding community. These strategies tend to minimize flash and machining cost. Strategy 1 (extruding the parting segment along a direction perpendicular to the parting direction) is the most commonly employed strategy. It minimizes the surface area and hence the machining time. It also maximizes the clamp pressure to minimize flash. Strategy 2 and 3 (extruding the parting segment along a direction tangential to the core/cavity surface) is mainly used for planar surfaces. It results in a smooth surface and hence faster machining. It also allows sharing the machining setup with the core/cavity surface.

8.2 Problem Formulation

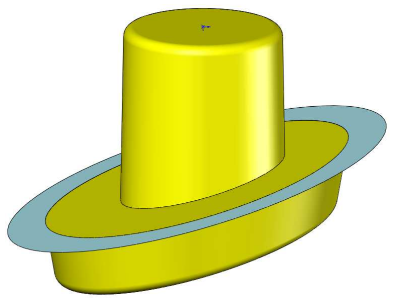
Most of the commercial CAD software, such as SolidWorks 2006$^{\text{TM}}$ [SWK06] provide tools for creating shutoff surface. Creating the inner shutoff surfaces is relatively easier. The CAD systems provide very sophisticated tools for creating filled surfaces that work fine for inner shutoff surfaces. The inner shutoff surfaces are created automatically without any problem. Creating the outer shutoff surface is difficult. The tools provided by the CAD systems are too low-level. The designer is expected to make all critical decisions. The designer decides which strategy to follow for each parting line segment. The designer also manually corrects the



(a) Strategy 1 (T_1): Extrude direction perpendicular to the parting direction



(b) Strategy 2 (T_2): Extrude direction tangential to the core surface



(c) Strategy 3 (T_3): Extrude direction tangential to the cavity surface

Figure 8.4: Strategies for creating shutoff surface.

invalid shutoff patches (overlapping and intersecting) and creates bridge patches. Since the main challenge lies in creating the outer shutoff surface, for the sake of clarity, we will assume that the given parting line does not have any inner parting loops. It only consists of one outer parting loop.

The aim of the mold designer is to create a shutoff surface, which is valid and requires minimum machining cost. The following sections define the validity and machining cost of a shutoff surface. Section 8.2.3 finally defines the problem in terms of these concepts.

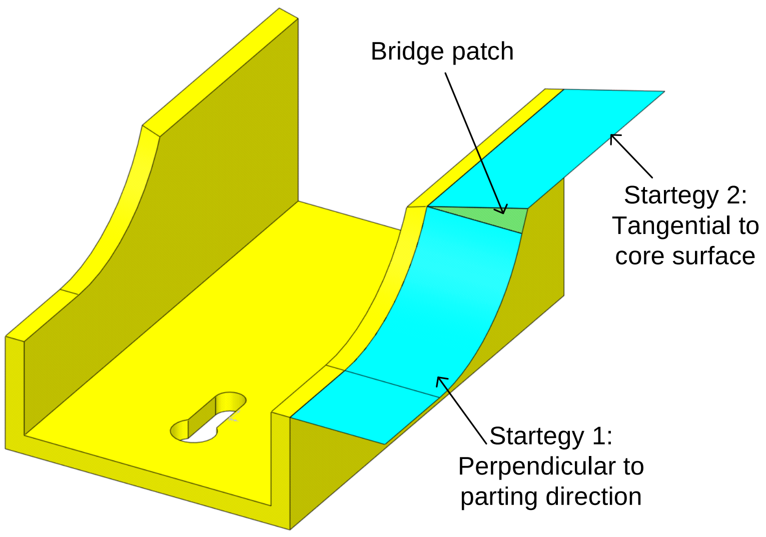


Figure 8.5: A bridge patch stitches two disjoint shutoff patches

8.2.1 Validity of a Shutoff Surface

For a given polyhedral object P and parting direction $\vec{d}$, the properties of a valid shutoff surface $S = \{s_1, \ldots, s_n\}$ are the following:

a) *Continuous*: The union of the shutoff surface and a mold-piece region (core or cavity) is a continuous surface without any holes, i.e., $S^* = S \cup C_o$ is a continuous surface with only one boundary loop.

b) *Accessible*: The shutoff surface is accessible along the parting direction, i.e., for every point p on S, the ray starting from p in the direction $\pm\vec{d}$ does not intersect the interior of P or S. This ensures that the shutoff surface does not form undercuts.

c) *Intersection-Free*: The shutoff surface does not intersect with itself ($s_i \cap s_j = \emptyset, \forall i, j \in \{1, \ldots, n\}$ and $i \neq j$) or the part ($S \cap P = \emptyset$).

8.2.2 Machining Cost of a Shutoff Surface

As explained in Section 8.1.2, for a given set of parting line segments $L = \{l_1, \ldots, l_n\}$, a shutoff surface S consists of two types of patches:

1. Segment patches, $S_s = \{s_1, \ldots, s_n\}$ created by extruding the parting line segments. Each segment patch s_i is created using a strategy $t_i \in \{T_1, T_2, T_3\}$ for the parting line segment l_i. Hence each segment patch s_i can be represented as a set of candidate segment patches.

$$s_i = \{s_i^1, s_i^2, s_i^3\} = \{s_i^j : j \in \{1, 2, 3\}\} \tag{8.2}$$

   where s_i^j is created using strategy T_j. The cardinality of the set s_i is three.

2. Bridge patches, $S_b = \{b_1, \ldots, b_n\}$ to stitch the adjacent segment patches. The bridge patch b_1 connects the segments patches s_1 and s_2, b_2 connects s_2 and s_3, and so on. At the end b_n connects s_n and s_1 to complete the loop. Each bridge patch b_i can be represented as a set of candidate bridge patches.

$$b_i = \{b_i^{j,k} : j, k \in \{1, 2, 3\}\} \tag{8.3}$$

   where $b_i^{j,k}$ connects the segment patches s_i^j and s_{i+1}^k. The cardinality of the set b_i is nine.

The machining cost of a segment patch s_i depends on its geometry, which in turn depends on the strategy used to create it. Hence the machining cost of the segment patch s_i can be written as:

$$C(s_i) = \sum_{j=1}^{3} \delta_i^j C(s_i^j) \tag{8.4}$$

where,

- $C(s_i^j)$ is the machining cost of the segment patch s_i^j and is given by Equation 3.15.

- $\delta_i^j \in \{0, 1\}$ is a binary variable. $\delta_i^j = 1$ if T_j is chosen to create s_i, else it is equal to zero. Also, only one strategy can be chosen for a segment patch, i.e., $\sum_{j=1}^{3} \delta_i^j = 1$.

Suppose a bridge patch b_i, connects two segment patches s_i and s_{i+1}. The machining cost of b_i depends on size of the gap between s_i and s_{i+1}, which in turn depends on the strategies used to create s_i and s_{i+1}. Hence the machining cost of a bridge patch b_i can be written as:

$$C(b_i) = \sum_{j=1}^{3} \sum_{k=1}^{3} \delta_i^j \delta_{i+1}^k C(b_i^{j,k}) \tag{8.5}$$

where $C(b_i^{j,k})$ is the machining cost of the bridge patch $b_i^{j,k}$ and is given by Equation 3.15. T_j and T_k are the strategies used to create the segment patches s_i and s_{i+1} respectively.

From Equation 8.4 and Equation 8.5, the total machining cost of the shutoff surface for a given set of parting segments $L = \{l_1, \ldots, l_n\}$ can be written as a function of the strategies used to create the segment patches.

$$C_t = \sum_{i=1}^{n} (C(s_i) + C(b_i)) \tag{8.6}$$

8.2.3 Problem Statement

Problem CreateShutoffSurface

Input:

1. A polyhedral object P

2. Parting direction $\vec{d}$

3. Mold-piece regions – core region (C_o), and cavity region (C_a)

4. Parting line segments $L = \{l_1, \ldots, l_n\}$. A parting line can be split into smooth segments by the designer manually or automatically by a software by identifying vertices where the angle of the parting line changes abruptly.

5. Required width w of the outer shutoff surface. The width is supplied by the designer and is usually in the range of $0.25 - 1.0$ inch. It is selected such that there is enough contact pressure between core and cavity to prevent the injected material from leaking. It depends on injection pressure and mold material.

Output:

1. Shutoff surface S

2. Machining cost C_t as defined in Equation 8.6.

Output Requirements:

1. Each segment patch s_i follows one of the standard strategies $t_i \in \{A, B, C\}$, i.e., the extrude direction is either perpendicular to the parting direction or tangential to C_o/C_a

2. The shutoff surface S is valid, i.e., it is (a) continuous, (b) accessible, and (c) intersection-free as described in Section 8.2.1

3. The machining cost C_t is minimum

8.3 Overview of Approach

The algorithm presented here works in two steps:

- Step 1: *Create an initial valid shutoff surface* (Section 8.4.1). An initial shutoff surface that satisfies all the requirements outlined in Section 8.2.1 is created. This portion of the algorithm is adapted from the algorithm presented by Ahn et al. [Ahn02]. The surface created in this step is mathematically valid (continuous, accessible, and intersection-free), but may not be optimal. It also does not follow any of the standard strategies.

- Step 2: *Optimize the initial shutoff surface* (Section 8.4.2). Strategy for each segment patch is selected such that the total machining cost of the shutoff surface is minimum. This problem of finding an optimal combination of strategies for creating the segment patches is reduced to the single-source shortest path problem and Dijkstra's algorithm is used to find the optimal solution. The strategies selected for each segment patch is applied to the shutoff surface created in the first step. The vertices of the shutoff surface are moved to make the shutoff surface follow the selected strategies while preserving its validity. Bridge patches are finally created to fill the gaps between adjacent segment patches.

8.4 The Algorithm

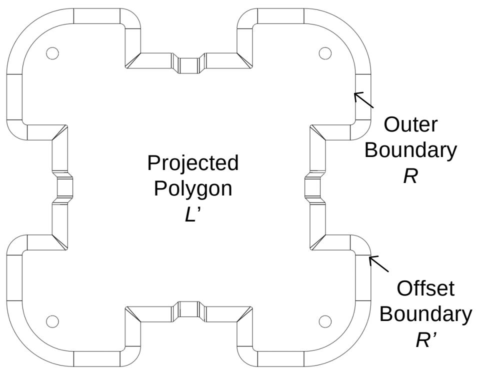
Without the loss of generality, we can assume that the parting direction is along the z-direction. Let $L = \{l_1, \ldots, l_n\}$ be the given parting line between the core region (C_o) and cavity region (C_a). We need to create a shutoff surface with minimum machining cost C. As described in Section 8.3, the algorithm for creating the optimal shutoff surface works in two steps. This section presents the detailed description of each step.

8.4.1 Creating an Initial Valid Shutoff Surface

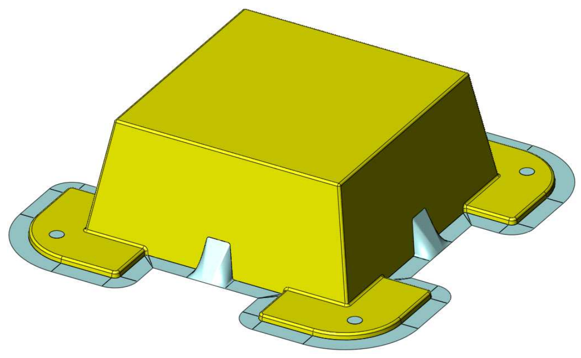
The method for creating a valid shutoff surface is adapted from the algorithm presented by Ahn et al. [Ahn02]. We project the parting line L onto the xy-plane to obtain a polygon L' with holes. Figure 8.2 shows the projected polygon for the part shown in Figure 8.1. It should be noted that the collinear edges are not merged in the projection. Every vertex v of the parting line L is projected to a vertex v' of the polygon L'. To compute the boundary of L', we need to determine the union of the projection of L. If n_v is the number of vertices in L, this can be done in $O(n_v \log n_v)$ time using a plane-sweep algorithm [Prep85].

The outer boundary R of the projected polygon L' is offset to a closed contour R' as shown in Figure 8.6a. The offset distance is equal to the required width w of the outer shutoff surface. The holes in the polygon $\bar{L}$ and the region formed between R and R' are triangulated in linear time [Chaz91]. This triangulation is lifted into three-dimensional space by replacing every vertex v' by its associated vertex v as shown in Figure 8.6b. Note that the contour R' remains on the xy-plane as it does not have corresponding three-dimensional vertices. The lifted triangulation S is a valid shutoff surface. It is continuous because after lifting, it meets the part at the parting line. It is also accessible along the z-direction and does not intersect with the part or itself. These properties are proved by Theorem 8.3 and Theorem 8.4.

The triangulated surface S is a valid shutoff surface, but may not be optimal. It also does not follow any standard strategy. The next section selects a strategy



(a) Projected contour is triangulated



(b) The triangulation is lifted into 3D

Figure 8.6: Creating a valid shutoff surface.

for each shutoff patch. But before that we need to split S into shutoff patches. The parting line consists of a set of smooth parting segments $\{l_1, \ldots, l_n\}$. Each parting segment l_i has a corresponding segment r_i on the offset contour R. All the triangles formed between l_i and r_i form a segment patch s_i. The triangulated surface S is hence split into a set of segment patches $S = \{s_1, \ldots, s_n\}$

8.4.2 Optimizing the Initial Shutoff Surface

The previous step (Section 8.4.1) creates a mathematically valid shutoff surface. But the machining cost of created surface may not be minimum. It may also not follow any of the standard strategies. From Equation 8.6, the total machining cost of a shutoff surface for a given set of parting segments depends on the strategies selected for creating the segment patches. This step selects a strategy for creating each segment patch such that the total machining cost of the shutoff surface is minimum. This optimization problem can be formally stated as follows:

$$\begin{aligned} &Minimize \quad && C_t = \textstyle\sum_{i=1}^{n}(\sum_{j=1}^{3}\delta_i^j C(s_i^j) + \sum_{j=1}^{3}\sum_{k=1}^{3}\delta_i^j\delta_{i+1}^k C(b_i^{j,k})) \\ &s.t. && \delta_i^j \in \{0,1\} \\ & && \textstyle\sum_{j=1}^{3}\delta_i^j = 1 \end{aligned} \tag{8.7}$$

This reduces the problem of minimizing the machining cost of a shutoff surface to that of selecting an optimal combination of strategies for creating the segment patches. The major difficulty in selecting the strategies is the combinatorial nature of the problem. We need to select one segment patch s_i^j from each set s_i given in Equation 8.2. This makes the total number of candidate shutoff surfaces equal to 3^n. It is clear that greedily selecting the minimum-cost strategy for each segment patch individually may not yield the minimum-cost shutoff surface. The cost of the bridge patches needed to connect the minimum-cost segment patches may raise the total cost of the shutoff surface higher than that for some other combination of strategies.

This problem of finding an optimal combination of strategies can be reduced to the single-source shortest path problem by representing it as a Directed Acyclic Graph (DAG) where the nodes represent the segment patches and the edges represent the bridge patches. Figure 8.7 shows such a graph where strategy T_1 is selected for the segment patch s_1. Any path from the node s_1^1 at the top to the same node at the bottom represents a candidate shutoff surface. All candidate shutoff surfaces are covered by constructing similar DAGs for segment patches s_1^2 and s_1^3. Let us now assign weights to the edges of this DAG. The weight assigned to an edge is equal to the sum of the corresponding bridge patch cost and the segment patch cost. For instance, the cost of the edge connecting s_1^1 and s_2^3 is equal to the sum of the bridge patch $b_1^{1,3}$ machining cost and the segment patch s_2^3

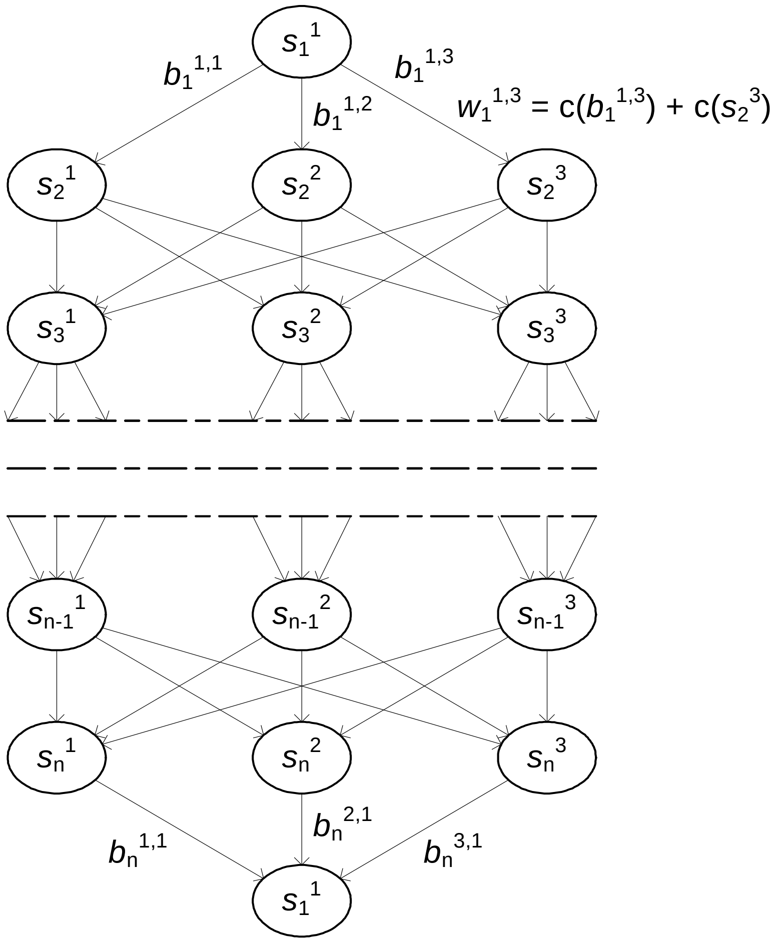


Figure 8.7: The problem of selecting the optimal combination of strategies

machining cost. The weight of the edge representing $b_i^{j,k}$ is given by:

$$w_i^{j,k} = c(b_i^{j,k}) + c(s_{i+1}^k) \tag{8.8}$$

The path for which the total edge cost is minimum represents the shutoff surface for which the machining cost is minimum. This is equivalent to solving the single-source shortest path problem. Moreover, since our graphs are directed and edge weights non-negative, we can apply Dijkstra's algorithm [Corm90] to find the minimum cost shutoff surface.

Based on the above observation, following algorithm is presented to find the optimal combination of strategies for the segment patches.

Algorithm SELECTOPTIMALSTRATEGY

Input:

- Set of segment patches $S = \{s_1, \ldots, s_n\}$ created in Section 8.4.1
- Cost function c(s) to calculate the machining cost of a surface patch. The cost function is usually supplied by the software tools such as MasterCAM and ProEngineer.

Output:

- Set of strategies $T = \{t_1, \ldots, t_n\}$ for each segment patch
- Machining cost C

Steps:

1. For each segment patch s_i, create the set of candidate segment patches given by Equation 8.2. Each candidate segment patch s_i^j is created by making s_i follow a strategy T_j. Section 8.4.3 describes a method for making a segment patch follow a strategy.
2. Create the set of candidate bridge patches $b_i^{j,k}$ given by Equation 8.3. Section 8.4.4 describes a method for creating a bridge patch for a given pair of segment patches.
3. Find the cost of segment patches s_i^j and bridge patches $b_i^{j,k}$ using the cost function c(s).

4. Construct weighted DAGs G_1, G_2, and G_3 for s_1^1, s_1^2, and s_1^3 respectively as shown in Figure 8.7.

5. Find the cost C_1, C_2, and C_3 of the shortest path in G_1, G_2, and G_3 respectively using the Dijkstra's algorithm.

6. Return the minimum of $\{C_1, C_2, C_3\}$ and the corresponding path T.

Figure 8.1 shows the optimized shutoff surface created by the above algorithm for the running example shown in Figure 8.6.

Theorem 8.1. *Let $L = \{l_1, \ldots, l_n\}$ be a set of parting line segments and $\{A, B, C\}$ be a set of standard strategies. The combination of strategies given by the algorithm* SelectOptimalStrategy *produces a valid shutoff surface with the minimum machining cost.*

Proof. By the structure of the DAGs $\{G_1, G_2, G_3\}$, any path includes all segment patches and bridge patches and only once. Hence the solution given by the algorithm is valid. Also, since we evaluate all possible combinations of strategies, every possible solution is embedded in the three DAGs. Suppose there is a solution T whose cost is less than that found by the algorithm OptimalStrategy. If T is a valid solution, it will be embedded in one of the three DAGs. And if it is embedded in the graph and has the lowest cost, the algorithm will find it. If T is not found, such a solution cannot exist. □

Theorem 8.2. *Let L be a parting line with n_s segments and n_v vertices. Let T be a set of n_{st} standard strategies. A valid shutoff surface with the minimum machining cost can be created in $O(n_v \log n_v)$ time.*

Proof. The complexity of the Dijkstra's shortest path algorithm is $O(V \log V + E)$, where V and E are the number of nodes and edges in the graph. This means that we can find the optimal combination of strategies in $O(n_s \log n_s + n_{st}^2)$ time, where n_s is the number of segments on the parting line and n_{st} is the number of strategies. Hence the overall complexity of the algorithm is $O(n_v \log n_v + n_s \log n_s + n_{st}^2)$. The number of strategies (n_{st}) is a fixed quantity, and the number of segments is always less than the number of vertices (n_v). So the overall complexity of the algorithm remains $O(n_v \log n_v)$. □

8.4.3 Making a Segment Patch Follow a Strategy

This section presents a method to modify a segment patch created in Section 8.4.1 such that each it follows a given strategy. This method is used by the algorithm OptimalStrategy described in Section 8.4.2 to build the DAGs $\{G_1, G_2, G_3\}$.

In Section 8.4.1, the shutoff surface was created by lifting a two-dimensional triangulation into three-dimensional space. The outer boundary R of the projected polygon L' was projected back to the parting line. However the offset contour R' remained on the xy-plane as it does not have corresponding three-dimensional vertices. We make each segment patch follow the assigned strategy by moving the vertices of R' in the $\pm z$-direction. We will first prove that moving the vertices of the shutoff surface in the $\pm z$-direction does not affect the validity of the shutoff surface, i.e., it remains continuous, accessible, and intersection-free. Since we do not move the vertices shared by the shutoff surface and the part, the shutoff surface remains continuous. The following theorems further prove that the shutoff surface also remains accessible and intersection-free.

Theorem 8.3. *Let S be a planar continuous surface on the xy-plane. Surface S',* *created by arbitrarily moving the vertices of S in the $\pm z$-direction, is accessible along the z-direction.*

Proof. To test if S' is accessible along the z-direction, we need to project the facets $\{f_1, \dots, f_n\}$ of S' on the xy-plane and check whether any pair intersects. Let $\bar{S}' = \{\bar{f}_1, \dots, \bar{f}_n\}$ be the projection of S' on the xy-plane. S' is accessible along the z-direction iff $\bar{f}_i \cap^* \bar{f}_j = \emptyset, \forall i, j \in \{1, \dots, n\}$.

Since S' is constructed by moving the vertices of S in the $\pm z$-direction, $\bar{S}'$ is the same as S and the facets $\{\bar{f}_1, \dots, \bar{f}_n\}$ are in fact the original facets of S that do not overlap. Hence, surface S' is accessible along the z-direction. □

Theorem 8.4. *Let S be a planar continuous surface on the xy-plane. Surface S', created by arbitrarily moving the vertices of S in the $\pm z$-direction, does not self-intersect.*

Proof. By separating axis theorem (Section 7.2.2), if there exists a plane on which the projection of the two facets do not overlap, then the facets do not intersect. From Theorem 8.3, the surface S' is accessible, i.e., the projection of the facets of S' on the xy-plane do not overlap. This implies that no two facets of S' intersect with each other and hence S' does not self-intersect. □

Each vertex v' on the contour R has a corresponding offset vertex v'' on the contour R' as shown in Figure 8.6a. The vertex v' also has a corresponding three-dimensional vertex v on a parting segment l_i. Hence each vertex v on a parting segment l_i has a corresponding vertex v'' on the offset contour R'. The vertex v is shared by two edges e_1 and e_2 on l_i. We lift the vertex v'' to the intersection line of the planes P_1 and P_2 containing e_1 and e_2 respectively. The orientation of the planes P_1 and P_2 depends on the strategy to follow.

If the selected strategy is T_1 (perpendicular to the parting direction), P_1 also contains the vector d_1 which is the cross-product of the edge e_1 and the z-direction, i.e., $\vec{d_1} = \vec{e_1} \times \vec{z}$. Similarly, P_2 also contains the vector d_2 which is the cross-product of the edge e_2 and the z-direction, i.e., $\vec{d_2} = \vec{e_2} \times \vec{z}$.

If the selected strategy is T_2 (tangential to core), P_1 is the plane of the core facet f_1 whose one of the edges is e_1. Note that f_1 is unique because the edge e_1 is a boundary edge of the core region. P_2 is the plane of the core facet f_2 whose one of the edges is e_2. Similarly, if the selected strategy is T_3 (tangential to cavity), P_1 and P_2 are the cavity facets whose one of the edges is e_1 and e_2 respectively.

8.4.4 Creating a Bridge Patch

This section presents a method to create a bridge patch for a given pair of segment patches. This method is used by the algorithm OptimalStrategy described in Section 8.4.2 to build the DAGs $\{G_1, G_2, G_3\}$.

As described in Section 8.4.3, a segment patch is made to follow a strategy by moving the vertices of the offset contour R' shown in Figure 8.6a in $\pm z$-direction. A vertex v'' on R', which is shared by two adjacent segment patches may be moved to different positions p_1 and p_2 with different z-coordinates. This creates a gap between the segment patches as shown in Figure 8.8.

Since the two segment patches are joined at the parting line, they always share a vertex on the parting line. This gap can be filled by single triangle b. The two positions p_1 and p_2 to which v'' is moved to have the same x and y coordinates, so b is vertical. By definition, a vertical facet is accessible along the z-direction and also does not occlude any other facet in the z-direction. But at the same time, a vertical shutoff patch is not desirable as it leads to potential mold problems (e.g.

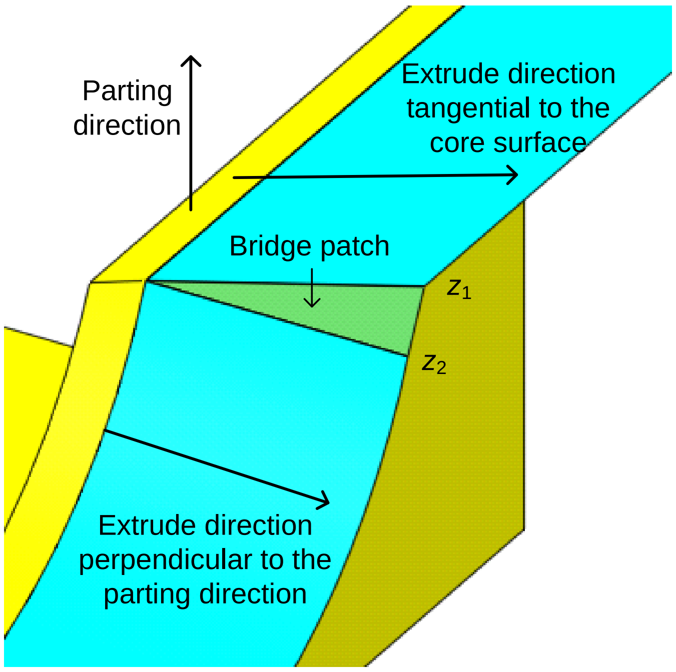

Parting direction
Extrude direction tangential to the core surface
Bridge patch
z_1
z_2
Extrude direction perpendicular to the parting direction

Figure 8.8: Creating a bridge patch

breakage) and part problems (e.g. flash). To avoid this, p_1 and p_2 are slightly perturbed in the $x - y$ plane so that the triangle b (bridge patch), does not remain vertical.

8.5 Results

Figure 8.9 shows the results of the algorithm for creating shutoff surface. The parts in Figure 8.9a, Figure 8.9b, and Figure 8.9c have a single shutoff patch. This does not require finding an optimal combination of strategies. Just finding the minimum-cost strategy does the job.

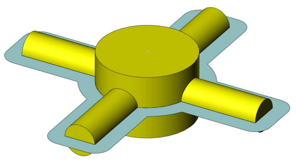
Strategy 3 (tangential to the cavity surface) is infeasible for the part shown in Figure 8.9d. This is because all the boundary facets of the cavity surface are parallel to the z-direction and hence cannot be extruded in the xy-plane. Another interesting feature of this part is that Strategy 1 (perpendicular to parting direction) and Strategy 2 (tangential to the core surface) result in the same surface everywhere. This is due to the fact that all the boundary facets of the cavity surface are perpendicular to the z-direction. Hence, although the part has multiple patches, we do not need to find an optimal combination of strategies.

The part shown in Figure 8.9e has multiple patches for which we need to solve the optimization problem as described in Section 8.4.2. The figure shows the different strategies used for different shutoff patches.

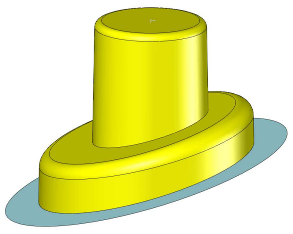
8.6 Summary

Mold design community currently uses a set of surface extension techniques to create shutoff surfaces. In case of complex parting lines, the parting line is manually split into parting line segments and different surface extension techniques are used on different parting line segments to minimize mold machining cost. Surface extension techniques are selected based on the prior experience of the mold designer.

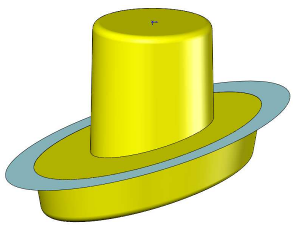
This chapter describes an algorithm for creating shutoff surface for a polyhedral object. The input to the algorithm includes the parting direction, mold-piece regions, and parting line of the object. The shutoff surface produced by the algorithm is created using commonly used surface extension techniques in the molding community. This algorithm produces provably optimal results for a given set of surface extension techniques without exhaustively enumerating all combinations of surface extension techniques. In fact the worst case time complexity for this algorithm is polynomial in terms of parting line segments. The current version



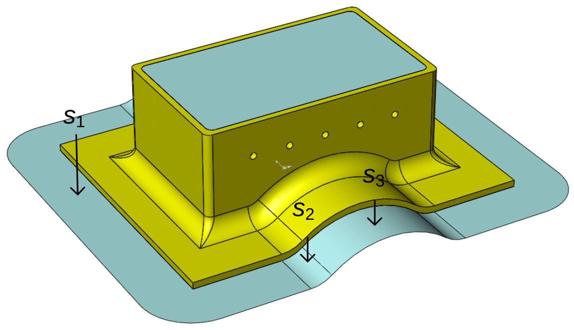
(a) Single patch: Perpendicular to the parting direction



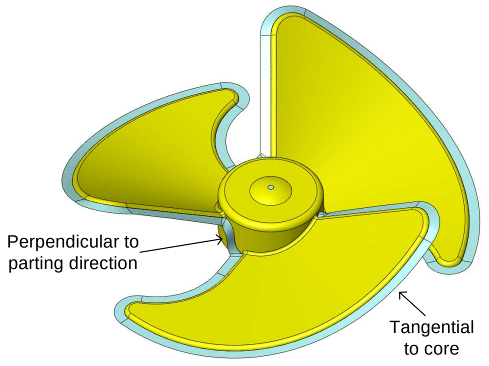
(b) Single patch: Tangential to the core surface



(c) Single patch: Tangential to the cavity surface



(d) Multiple patches: Perpendicular to pull and tangential to the core surface everywhere



(e) Multiple patches: Multiple strategies

Figure 8.9: Results of the algorithm for creating shutoff surface

of the algorithm works with three commonly known surface extension techniques and produces shutoff surfaces with guaranteed accessibility. The algorithm can be easily extended to include additional surface extension techniques as they become available. This algorithm has been tested with several complex objects which require use of multiple different surface extension techniques in creation of shutoff surfaces. This algorithm can also be used in die-casting.

Chapter 9

CONCLUSIONS

This chapter has been organized in the following manner. Section 9.1 describes the main intellectual contributions of this dissertation. Section 9.2 identifies the anticipated industrial benefits resulting from this research. Section 9.3 discusses the future research directions.

9.1 Intellectual Contributions

This dissertation makes intellectual contributions in the following areas:

1. *Framework for representing molding plans for articulated joints.* This dissertation presents a framework for representing molding plans for articulated joints. This framework allows us to record molding plans for joints in a reusable form. When a joint similar to a previously molded joint is found in a new assembly, the previously generated knowledge can be applied to the new joint. The complete set of applicability conditions are given to identify the applicable plan for a joint. Feasibility constraints that can be transferred from the molding plan of the joint to the overall molding planning problem for an assembly are identified. Four assembly design principles are given that

lead to feasible and efficient molding plans. If these design principles are followed, it is possible to reduce the molding cost and ensure that the molded assembly is of desired quality. This dissertation also presents six reusable molding plans for three basic joints – prismatic, revolute, and spherical.

2. *Algorithm for generating molding plans for articulated assemblies.* This dissertation describes an algorithm for generating a multi-stage molding plans for articulated assemblies. 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. This dissertation formulates it as a state-space search problem and uses branch and bound to search for an optimal solution. The state space may have large number of search nodes and processing each node takes a lot of time. These problems are handled by pruning infeasible solution paths and reusing the results of a search node. This dissertation 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. This algorithm can be adapted for assembly planning where monotonocity assumptions do not hold.

3. *Algorithm for finding mold-piece regions.* This dissertation presents two algorithms for finding mold-piece regions of components and assemblies. The first algorithm is an object-space algorithm, which runs on the central processing

unit (CPU) of a computer. The algorithm handles the near-vertical facets robustly by perturbing the parting direction. The implementation utilizes geometric properties of polyhedral objects and hierarchical data structure for efficiency. The second algorithm is an image-space algorithm, which can be executed on the graphics processing unit (GPU) of a computer. It robustly handles the near-vertical facets by slightly perturbing the vertices on those facets and visibility sampling. It exploits the computational power offered by the GPUs. It was shown that an efficient implementation of this algorithm does not impose any significant overhead on the GPU. The mold-piece regions even for parts with more than 50,000 facets can be highlighted at interactive rates. Such a system that provides real-time information about mold-piece regions will be very useful to the part and mold designers alike. They can easily optimize the part and mold design and if needed make appropriate corrections upfront, streamlining the subsequent design steps. Both algorithms have been successfully tested with several complex components and assemblies. The results of the image-space algorithm was verified by comparing them with those obtained by the object-space algorithm. Both algorithms produce identical results. These algorithms can also be used for machining, die-casting, and sheet metal forming.

4. *Algorithm for constructing optimal shutoff surfaces.* This dissertation describes an algorithm for creating shutoff surface for a polyhedral object. The input to the algorithm includes the parting direction, mold-piece regions, and parting line of the object. The shutoff surface produced by the algorithm is created using commonly used surface extension techniques in the molding community. This algorithm produces provably optimal results for a

given set of surface extension techniques without exhaustively enumerating all combinations of surface extension techniques. In fact the worst case time complexity for this algorithm is polynomial in terms of parting line segments. The current version of the algorithm works with three commonly known surface extension techniques and produces shutoff surfaces with guaranteed accessibility. The algorithm can be easily extended to include additional surface extension techniques as they become available. This algorithm has been tested with several complex objects which require use of multiple different surface extension techniques in creation of shutoff surfaces. This algorithm can also be used in die-casting.

9.2 Anticipated Industrial Benefits

Multi-stage molding has emerged as an important manufacturing process. It can be used to make better-quality articulated products at a lower cost. Unfortunately, there are currently no software tools to generate molding plans. It is difficult to perform the planning manually. It involves examining a large number of combinations and solving complex geometric reasoning problems. In the absence of software tools, it usually takes a long time – about three to four weeks on an average to develop a molding plan. There are also concerns about the feasibility and optimality of a molding plan because many decisions are based on subjective guesswork. 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.

The algorithm presented in this dissertation can be used to develop a software system that can automatically generate molding plans that are both feasible and economical. Currently a designer needs to manually develop a molding plan and feed it to the available mold design software to generate the mold pieces. The automation of planning will enable the complete automation of multi-stage mold design, which in turn will reduce the cost and lead-time associated with the deployment of multi-stage molds. Automation of mold design will also improve the part quality by exploring explores design alternatives that are otherwise difficult due to human constraints.

The economic deployment of in-mold assembly process will significantly impact the molding industry. In-mold assembly allows integration of functional elements, thereby reducing the number of components and additional assembly steps. Assembly is mostly a manual labor-intensive process and costs up to 50% of the total manufacturing cost of a product. The elimination of post-molding assembly operations will make manufacturing economically viable in places where labor costs are high. In-mold assembly also opens up the design space and present new possibilities. By eliminating the manual assembly operation, it allows the production of devices with extremely small components.

9.3 Future Work

Following the investigations taken up in this dissertation, a number of projects can be taken up:

1. *Incorporate the constraints imposed by flow and thermal considerations.* The algorithm for generating the molding sequence presented in this dissertation 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. If the components to be molded in a stage are too far, it may not be possible to deliver the molten plastic to all cavities with limited injection pressure. Also, if the volume of injected material in a molding stage is too much, the thermal management system may not have the capability to melt or cool down so much material. Hence the constraints imposed by the flow and thermal considerations should also be incorporated into the algorithm.

2. *Search the continuous configuration space of pre-stage components.* For every molding stage, we need to find a feasible configuration for pre-stage components. This dissertation presented 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.

3. *Develop plans for meso-scale joints.* The in-mold assembly process is especially useful when assembling the components manually is difficult such as when the components are very small in size. We believe that it will be very useful in meso-scale assemblies. This dissertation presented six molding plans for three basic joints. But the size of all those are joints are at the macro scale. Further investigation is required to develop plans for meso-scale joints.

4. *Extend the work to deal with parallel mechanisms.* This dissertation 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.

5. *Extend the methodology to incorporate compliant joints.* This dissertation only deals with rigid-body joints. Articulation can also be achieved by compliant joints. The compliant joints are created using a soft (compliant) material between two rigid materials. The assemblies with compliant joints have larger feasible configuration space than the assemblies with rigid joints. The geometry of the compliant joints can be 'stretched' away so that it no longer casts shadow over another component. The algorithm to find feasible configuration space for components need to be extended to handle compliant joints.

BIBLIOGRAPHY

[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.

[Alti89] L. Alting and H. Zhang. Computer aided process planning: The state of the art survey. *International Journal of Production Research*, 27(4):553–585, 1989

[Anan95] G.K. Ananthasuresh and S. Kota. Designing Compliant Mechanisms. Mechanical Engineering, 117(11): 93-96, 1995.

[Bala98] S. Balasubramanian, A. Elinson, J.W. Herrmann, and D. Nau. Fixture-based usefulness measures for hybrid process planning, *ASME Design for Manufacturing Conference*, Atlanta, Georgia, September 13-16, 1998

[Bala00] Balasubramaniam, M., Laxmiprasad, P., Sarma, S., and Shaikh, Z. Generating 5-axis NC roughing paths directly from a tessellated representation. Computer-Aided Design, 32: 261-277, 2000.

[Bane06] A.G. Banerjee, and S.K. Gupta, A step towards automated design of side actions in injection molding of complex parts. In Proceedings of Geometric Modeling and Processing, Pittsburgh, PA, 2006

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

[Berg00] M. de Berg, M. van Kreveld, M. Overmars, and Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 2000.

[Bruc04] H. Bruck, G. Fowler, S.K. Gupta, and T. Valentine. Towards bio-inspired interfaces: Using geometric complexity to enhance the interfacial strengths of heterogeneous structures fabricated in a multi-stage multi-piece molding process. Experimental Mechanics, 44(3):261–271, 2004.

[Bryc96] D. M. Bryce. Plastic Injection Molding, Vol. I: Manufacturing Process Fundamentals. Society of Manufacturing Engineers: Dearborn, Michigan, 1996.

[Chaz91] B. Chazelle. Triangulating a simple polygon in linear time. Discrete Comput Geom 1991;6:485-524.

[Chen93] L.L. Chen, S.Y. Chou, and T.C. Woo. Parting directions for mould and die design. *Computer-Aided Design,* 25: 762-768, 1993

[Chen01a] Y. Chen and D. Rosen. Problem formulation and basic elements for automated multi-piece mold design. *ASME Design Engineering Technical Conference,* Pittsburgh, PA, September 2001

[Chen01b] Y. Chen and D. Rosen. A region based approach to automated design of multi-piece molds with application to rapid tooling. *ASME Design Engineering Technical Conference,* Pittsburgh, PA, September 2001

[Chen03] Y. Chen and D. Rosen. A Reverse Glue Approach to Automated Construction of Multi-Piece Molds. *Journal of Computing and Information Science in Engineering,* 3(3): 219-230, September 2003

[Corm90] T. H. Cormen, C. E. Leiseron, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, 1990.

[Dhal01] S. Dhaliwal, S. K. Gupta, J. Huang, and M. Kumar. A feature-based approach to automated design of multi-piece sacrificial molds. *ASME Journal of Computing and Information Science in Engineering,* 1(3): 225-234, 2001.

[Dhal03] S. Dhaliwal, S.K. Gupta, J. Huang, and A. Priyadarshi. Algorithms for computing global accessibility cones. *Journal of Computing and Information Science in Engineering*, 3(3):200–209, September 2003

[Elis97] A. Elinson, J.W. Herrmann, I. Minis, D. Nau, and G. Singh. Toward hybrid variant/generative process planning. Design for Manufacturing Symposium, ASME Design Engineering Technical Conference, Sacramento, California, September 14-17, 1997

[Ever01] C. Everitt, A. Rege, and C. Cebenoyan. Hardware shadow mapping. Whitepaper, `http://developer.nvidia.com/object/hwshadowmap_paper.html`.

[Elbe95] G. Elber and E. Cohen. Arbitrarily precise computation of Gauss maps and visibility sets for freeform surfaces. In Proceedings of 3th Symposium on Solid Modeling and Applications, May 1995, Salt Lake City, Utah.

[Elbe05] G. Elber, X. Chen, and E. Cohen, Mold Accessibility via Gauss Map Analysis. *Journal of Computing and Information Science in Engineering*, Vol. 5, No. 2, (2005), 79-85

[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

[Fu99] M. W. Fu, J. Y. H. Fuh, and A. Y. C. Nee. Undercut feature recognition in an injection mould design system. *Computer-Aided Design,* 31: 777-790, 1999.

[GLEW] The OpenGL Extension Wrangler Library, Release 1.3.4,
Available `http://glew.sourceforge.net`.

[GLUT] The OpenGL Utility Toolkit, Version 3.7.6,
Available `http://www.xmission.com/~nate/glut.html`.

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

[Gott96] S. Gottschalk. Separating axis theorem. Technical Report TR96-024, Department of Computer Science, UNC Chapel Hill, 1996.

[Gouk06] R.M. Gouker, S.K. Gupta, H.A. Bruck, and T. Holzschuh. Manufacturing Of Multi-Material Compliant Mechanisms Using Multi-Material Molding. Accepted for Publication in International Journal of Advanced Manufacturing Technology, 2006.

[Govi05] N.K. Govindaraju, N. Raghuvanshi, M. Henson, and D. Manocha. A Cache-Efficient Sorting Algorithm for Database and Data Mining Computations using Graphics Processors. *UNC Tech. Report,* 2005

[GTS] GNU Triangulated Surface Library: Version 0.7.6,
Available `http://gts.sourceforge.net`.

[Gu99] Z. Gu, Z. Zhou, S. Gao, and J. Shi. Determination of mold parting direction based on automatic molding feature recognition. *ASME Computers and Information in Engineering Conference,* Las Vegas, Navada, 1999.

[Gupt02] S.K. Gupta, X. Li, and A. Priyadarshi. An algorithm for design of multi-stage molds for multi-material objects with complex interfaces. In *ASME International Mechanical Engineering Congress and Exposition*, New Orleans, Louisiana, November, 2002

[Gupt03] S.K. Gupta and A.K. Priyadarshi. Towards automated design of multi-piece molds. In *ASME Design Automation Conference*, Chicago, IL, September 2003.

[Huan01] J. Huang. Accessibility-driven spatial partitioning: A step towards automated design of multi-piece molds. *PhD Thesis,* Department of Mechanical Engineering, University of Maryland, College Park, 2001.

[Huan02] J. Huang and S.K. Gupta. Accessibility driven spatial partitioning for generating sacrificial multi-piece molds. *ASME Design for Manufacturing Conference,* Montreal, Canada, 2002.

[Hui92] K.C. Hui and S.T. Tan. Mould design with sweep operations - a heuristic search approach. *Computer-Aided Design,* 24(2), 1992.

[Hui97] K.C. Hui. Geometric aspects of the mouldability of parts. *Computer-Aided Design,* 29(3): 197-208, 1997.

[Insa04] M. Insall and E.W. Weisstein. Partially Ordered Set. From *MathWorld*–A Wolfram Web Resource. `http://mathworld.wolfram.com/PartiallyOrderedSet.html`.

[Jans98] K.M.B. Jansen, D.J. Van Dijk, and M.H. Husselman. Effect of Processing Conditions on Shrinkage in Injection Molding. Polymer Engineering and Science, vol. 38, no. 5, pp. 838-846, May 1998.

[Kang97] J.K. Kang and S.H. Suh. Machinability and set-up orientation for five-axis numerically controlled machining of free surfaces. International Journal of Advanced Manufacturing Technology, 1997, 13: 311-325.

[Kett99] Lutz Kettner. Software Design in Computational Geometry and Contour-Edge Based Polyhedron Visualization. PhD Thesis, ETH Zrich, Institute of Theoretical Computer Science, September 1999

[Khar05] Khardekar, Burton, and McMains, Finding Feasible Mold Parting Directions Using Graphics Hardware, Proceedings of the 2005 ACM symposium on Solid and Physical Modeling, Cambridge, MA, June 2005, pp. 233-243.

[Kim95] D.S. Kim, P.Y. Papalambros, and T.C. Woo. Tangent, normal, and visibility cones on Bzier surfaces. Computer Aided Geometric Design, 1995, Vol. 12, 305-320.

[Kris97] S. Krishnan and E.B. Magrab. A new approach to mold design using manufacturable entities. *Proceedings of the Design for Manufacturability*

Symposium: ASME Winter Annual Meeting, Dallas, TX, November 1997.

[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.

[Kwon92] K.K. Kwong. Computer-aided parting line and parting surface generation in mould design. PhD Thesis, The University of Hong Kong, Hong Kong, 1992

[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.

[Lu00] H. Lu and W.B. Lee. Detection of interference elements and release direction in die-cast and injection-moulded components. *Journal of Engineering Manufacture,* 214(B6): 431-441, 2000.

[Majh99] J. Majhi, P. Gupta, and R. Janardan. Computing a flattest, undercut-free parting line for a convex polyhedron, with application to mold design. *Computational Geometry Theory and Applications,* 13: 229-252, 1999.

[Mall94] R. A. Malloy. Plastic Part Design for Injection Molding: And Introduction. Hanser Gardner Publications, Inc.: Cincinnati, Ohio, 1994.

[Mcma04] S. McMains and X. Chen, Determining Moldability and Parting Directions for Polygons with Curved Edges. In International Mechanical

Engineering Congress and Exposition, ASME, IMECE, Anaheim, CA, (2004)

[MFC] Microsoft Foundation Classes, Available `http://msdn.microsoft.com/visualc`.

[Nee98] A.Y.C. Nee, M.W. Fu, J.Y.H. Fuh, K.S. Lee, and Y.F. Zhang. Automatic determination of 3-D parting lines and surfaces in plastic injection mould. *Annals of CIRP,* 47(1): 95-98, 1998

[Nils98] Nils J. Nilsson. *Artificial Intelligence: A New Synthesis*, Morgan Kaufmann Publishers, Inc., 1998

[OpenGL] OpenGL rendering library, Release 2.0, Available `http://www.opengl.org`.

[Owen05] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A.E. Lefohn, and T.J. Purcell. A Survey of General-Purpose Computation on Graphics Hardware. In Eurographics 2005, State of the Art Reports, August 2005, pp. 21-51.

[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.

[Post05] P. Postawa. Shrinkage of Moldings and Injection Molding Conditions. Polimery 50(3): 201-207, 2005.

[Prep85] F.P. Preparata and M.I. Shamos. Computational geometry: an introduction. New York: Springer-Verlag, 1985

[Priy04] A.K. Priyadarshi and S.K. Gupta. Geometric algorithms for automated design of multi-piece permanent molds. Computer Aided Design, 36(3): 241-260, 2004.

[Priy06a] 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. Accepted for publication in *The International Journal of Advanced Manufacturing Technology.*

[Priy06b] 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

[QSlim] QSlim simplification library, Version 2.1, Available `http://graphics.cs.uiuc.edu/~garland/software/qslim.html`.

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

[Reev87] W.T. Reeves, D.H. Salesin, and R.L. Cook. Rendering antialiased shadows with depth maps. In *Computer Graphics (SIGGRAPH 87 Proceedings)*, pages 283-291, July 1987

[Roth04] J. Rotheiser. Joining of Plastics, Handbook for Designers and Engineers. Hanser Gardner Publications, Inc.: Cincinnati, Ohio, 2004.

[Sahn98] S. Sahni. Data Structures, Algorithms, and Applications in C++, McGraw-Hill, 1998

[Simm06] SimMechanics documentation, Release 14, Available `http://www.mathworks.nl/access/helpdesk/help/toolbox/physmod/mech`

[Spit99] Spitz, S.N., Spyridi, A.J., and Requicha, A.A.G. Accessibility analysis for planning of dimensional inspection with coordinate measuring machines. IEEE Transactions on Robotics and Automation, 15(4): 714-727, Aug 1999.

[Stag97] R. Stage and C. Roberts. A framework for representing and computing tool accessibility. Proceedings of DETC'97, September 14-17, 1997, Sacramento, California.

[Suh95] S.H. Suh and J.K. Kang. Process planning for multi-axis NC machining of free surfaces. Internal Journal of Production Research, 1995, Vol. 33, No. 10, 2723-2738.

[SWK06] Solidworks 3D mechanical design and 3D CAD Software Version 2006, Available `http://www.solidworks.com/`.

[Tan88] S.T. Tan, M.F. Yuen, W.S. Sze, and K.W. Kwong. A method for generation of parting surface for injection moulds. *Conference on Computer Aided Production Engineering,* Edinburgh, UK, 1988

[Urab97] K. Urabe and P. K. Wright. Parting direction and parting planes for the CAD/CAM of plastic injection molds. *ASME Design for Manufacturing Conference,* Sacramento, CA, 1997

[Vija98] J.V.K. Vijay, U. Shrinivasa, and B. Gurumoorthy. Automatic draw direction generation for die design. *ASME Computers and Information in Engineering Conference,* Atlanta, GA, 1998

[Wang94] Y. Wang and S. Molnar. Second-Depth Shadow Mapping. *UNC-CS Technical Report TR94-019,* 1994

[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,* 118(3): 29-39, 1996

[Wein97] M. Weinstein and S. Manoochehri. Optimum parting line design of molded and cast parts for manufacturability. *Journal of Manufacturing Systems,* 16(1): 1-11, 1997

[Weis99] E.W. Weisstein. Rotation Formula. From *MathWorld*–A Wolfram Web Resource. `http://mathworld.wolfram.com/RotationFormula.html`

[Whal05] S. Whalen. Audio and the Graphics Processing Unit. *IEEE Visualization Tutorial,* 2004

[Wiki05a] Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Gimbal`, April 2005.

[Wiki05b] Wikipedia, the free encyclopedia. `http://en.wikipedia.org/wiki/Universal_joint`, July 2005.

[Will78] L. Williams. Casting curved shadows on curved surfaces. In *Proceedings of SIGGRAPH '78*, pages 270-274, 1978

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

[Wong96] T. Wong, S.T. Tan, and W.S. Sze. Parting line formation by slicing a trimmed surface model. *ASME-DETC,* Irvine, CA, 1996

[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

[Yin01] Z. Yin, H. Ding, and Y. Xiong. Virtual Prototyping of mold design: geometric mouldability analysis for near-net-shape manufactured parts by feature recognition and geometric reasoning. *Computer-Aided Design,* 33: 137-154, 2001