Abstract

Title of Dissertation: Geometric Algorithms for Recognition of Features from Solid Models

William C. Regli, III, Doctor of Philosophy, 1995

Dissertation directed by: Professor Dana S. Nau
Department of Computer Science

Collaborative engineering has expanded the scope of traditional engineering design to include the identification and elimination of problems in the manufacturing process. Manufacturing features and feature-based representations have become an integral part of research on manufacturing systems, due to their ability to model the correspondence between design information and manufacturing activities. One necessary component of an integrated Computer Aided Design/Computer Aided Manufacturing (CAD/CAM) environment is a tool to automatically recognize manufacturing features from a CAD or solid model.

In this thesis we present a methodology for recognizing a class of machining features and for addressing the computational issues involved in building tractable and scalable solutions for automated feature recognition. This approach is described for a class of volumetric features based on material removal volumes produced by operations on 3-axis vertical machining centers.

A computational framework is developed for representing different types of common machining features and specifying the recognition problem. Based on this framework, novel serial and multi-processor recognition algorithms are described and analyzed with respect to their completeness and complexity. The goal of this dissertation is to advance the understanding of the basic computational issues that arise in feature recognition from solid models of mechanical artifacts and to facilitate development of effective and efficient systems that can scale to address industrial problems.

Geometric Algorithms for Recognition of Features from Solid Models

by

William C. Regli, III

Dissertation submitted to the Faculty of the Graduate School
of The University of Maryland in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
1995

Advisory Committee:

Professor Dana S. Nau, Advisor
Professor Laveen Kanal
Associate Professor James Hendler
Associate Professor Ioannis Minis
Associate Professor David Mount
Associate Professor V. S. Subrahmanian

Dedication

To Susan...

Acknowledgements

I would like to express my deep gratitude to my graduate advisor, Professor Dana Nau. As I write this I find myself ill-equipped to describe the tremendous positive impact my six-plus years working under his direction has had on my intellectual and professional development. An Occam of words and mathematics, Dana consistently challenged me to re-think and re-formulate my presumed knowledge and adapt it to the exciting terrain of interdisciplinary research in which we were working. I can honestly say that Dana showed me all aspects of the academic process: from our writing of proposals and papers; working with government and industry; 3 AM staff meetings; fashion consulting; and total quality management and continuous self-improvement—often involving rewrites the day before (or day of) various deadlines. I valued his mentorship and will always take with me the experience and training I obtained in his company. A special thanks to one of my mentors at my alma mater Saint Joseph's, Professor Ranan Banerji, for pointing me toward Dana when I first got admitted to Maryland.

My gratitude to the members of my thesis committee: Professors Laveen Kanal, V.S. Subrahmanian and Dave Mount. In particular, thanks to committee members Professors Jim Hendler and Ioannis Minis with whom I have worked particularly closely during my time at the University of Maryland. I learned much from my interaction with them and their students. Thanks also to Professor Bill Gasarch, with whom I share an affinity for Star Trek and theoretical computer science. This thesis work shows some influence that can be directly attributed to the theory courses I had with Bill.

It would be impossible to overstate the influence of my colleagues and mentors at NIST. I owe great thanks to the Manufacturing Engineering Lab and the

Manufacturing Systems Integration Division, including Steve Ray, Mike Pratt, Kevin Lyons, and Tom Kramer. In particular, a special thanks to Pete Brown, my "Professor du Monde," for whom I began working at NIST in 1992. I can only hope that I have acquired but a small portion of Pete's special ability to navigate between the real world and that of academia.

To Scott T., with whom I shared the majority of my graduate school experience as well as many afternoons (and several early mornings) of playing hookey on the links. His friendship, batter and cakes, love of my euro-style coffee, and "musical" talents always managed to bring a certain perspective to things.

To Chris L., with whom I started my graduate study and with whom I am finishing it—albeit on different continents. Chris' innate ability to completely separate work from eating, skiing, traveling, salsa making, coffee drinking, and eating provided many defining moments to our friendship and taught me valuable lessons—one in particular was to always check flight numbers and not gate numbers when traveling by air and looking for your plane. My gratitude also to Kristen for putting up with (and often encouraging) our hijinks (as well as for getting me lots of cool stuff during her time in the retail industry) and to my new godchild, Allison, who I have just begun to get to know.

To SK Gupta, from whom I began to learn the complexities of the world of engineering . The close working relationship we developed over the past 5 years showed me the rewards of interdisciplinary study and open for us a constant stream of challenges and opportunities. May we continue in our mutual discovery of such opportunities.

To Stephen S., my officemate and friend for exactly one month longer than we have been at UMD together, for his intellectual insight and contagiously positive attitude. With Stephen I made it through the comprehensive exams—his uncanny abilities with the comps' arcane questions whipped me into a studying frenzy. He also consistently reminded me that school can still be fun—even graduate school.

A special thanks to Matt and Sue E.: to Matt for the valuable lessons he has showed me, including how to mix fundamental research with a real life and how to mix fine gin with vermouth; and to Sue for her fantastic butter-packed cooking and for bringing Suzy to Maryland in 1992 and back into my life (so that she and I could fall in love and get married etc.).

To Mark, Ellen and Harris R. for many things, including numerous "TV nites" and for introducing me to Duda's, Ostroski's, The Home Depot, and Seinfeld. Martin F. for starting me down the wise way of Emacs. Daniel M. for providing me with an early role model during my graduate study. The denizens of AVW 3228 and 3270 during my years of study (in particular Bill A., Brian K., Bob K., Bob B., Leonard D., Ed O., Kut E., Dan "The Dancer" E., Jerry "Greed" W., Lee

S., Diganta D., Alex E., Kirin H., Dave R., Nirupama C., Subrata G., and Sean L.) who kept me sane and entertained with their often weird political theories, lunchtime excursions, analyses of the latest in Macintosh/Internet "software," and numerous discussions of American television culture (TP, MST3K, ST:TNG, and The Simpsons).

Others who highlighted this trip: Gary F. the LaTeX guru, C hacker, and coffee lover; Lynne, frisbee guru, fellow classmate, and Gary's wife; Gary's officemate Dave B. for pointing at NIHCL and for his insightful job-hunting advice. Thanks to Dr. Nancy Eagle Lindley, CS Department's very own equivalent of Harry Tuttle, who always knew where to point me through the U of Maryland bureaucratic maze and who always did it with a smile. Raghu K. for recruiting me to work with Dana and introducing me to solid modeling using **ProtoSolid**, the world's first (and only) talking solid modeling and CAD package. Thanks to George V., coding wizard and author of ProtoSolid, for recruiting me to come to UMD and for the many things I learned from him during the "Internet summer" of 1993 when we had the chance to work together at NIST under Pete Brown's guidance. Special thanks also to Pete Cottrell and his amazingly efficient staff that services all computing needs of the CS department at UMD (including `stevek@cs`, `dabe@cs`, `chang@cs`, `harry@cs`, and the entire crew at StaffWorld `staff@cs`) . I learned a great deal from their knowledgeable answers to my many questions and consistently appreciated their willingness (often late at night) to "go the extra mile," "untar one more release of ACIS," and "keep that ski map web server up-and-running."

To my cousin Michael Gaubert (J.D., Georgetown University, 1991) and my brother Brian (B.A. Georgetown University, 1990) for making my first two years in graduate school interesting (to say the least). Todd K. and his feline companion, Brandy (the cat of peace). Thanks also to Fr. Tom Kalita, director of the Catholic Student Center at U of MD, for his heartfelt and often touching ministerial style.

My deep gratitude to Sensai Ali Tabassi and his brother, Sensai Michael Tabassi, under whom I undertook the study of Shotokan Karate in January 1990. I am very fortunate to have experienced their intense dedication to their Art, for without the physical, spiritual, and mental discipline acquired from my years of study with them this thesis would not have been possible. Thanks also to my many fellow Karateka with whom I trained during this time: William, Gramps/Will, Hootan, Monsour, Charles, Mike, Brian, Cheri, Margaret, (and everyone else) ... my time in the Dojo with them has taught me a great deal about myself, how to learn, and how to teach.

Food for this dissertation was provided by the Marathon Deli ("Home of the World's Best Gyros"), Jerry's Taco Fiesta ("Home of Taco Tuesday and $1 Tacos"), and Alario's Subs and Pizza (a slice and a salad for a mere $\$\pi$). I

should have had my direct deposit slips rerouted to them—it would have been more efficient.

A special thanks to my family: Mom, Dad, Brian, Theresa, and Tanya for their support. Amusingly, since heading off to graduate school in computer science, everyone in the Regli household has "gotten wired"—well, everyone except Dad. Maybe the recent increase in online "Web coupons" will add some motivation for him to move from his accountant's books to Lotus 1-2-3.

And finally, to Susan, my beloved wife and to whom this thesis and my life are dedicated. No matter what the geographic location, her love makes all things possible.

Table of Contents

List of Figures

Geometric Algorithms for Recognition
of Features from Solid Models

William C. Regli, III

December 4, 1995

This comment page is not part of the dissertation.

Typeset by LaTeX using the `dissertation` style by Pablo A. Straub, University of Maryland.

Chapter 1

Introduction

1.1 Background and Motivation

In the modern marketplace, computing is essential in all aspects of manufacturing activity. Computers have brought to life to terms like **collaborative engineering** and **agile manufacturing**, and have played a critical role in the re-invention of manufacturing in the United States [23]. In a continuing quest to decrease the time interval between the conceptualization of a product and first production, information technology has been fused with manufacturing practice: from design (graphics, visualization), to analysis (numerical methods, simulation), to synthesis (robotics, manufacturing planning).

It has become increasingly clear that manufacturing is an information intensive undertaking. The new world of highly Inter-networked collaborative engineering overflows with information—information of which until very recently human beings had been the sole custodians. This is changing as low-cost computational power and increasingly sophisticated software technologies are enabling development of **intelligent** systems for design and manufacturing [23, 205]. Computers are essential tools for modern engineers, enhancing human abilities to bring better and more cost-effective products to market quickly and efficiently.

Automation of design and manufacturing activities poses many difficult computational problems. As manufacturing activities are computerized, more fundamental computational issues emerge. Critical among them is how to enable computers to effectively and intelligently understand and reason about engineering information.

Engineering information is a broad category containing all varieties of data that are relevant to the product realization process. Information can pertain to manufacturing resources, production schedules, or business constraints. This thesis is concerned with understanding of computer-aided design (CAD) information. A CAD system is the modern designer's drafting table, allowing her or him to develop engineering drawings and diagrams defining artifacts that fulfill the designer's functional intentions. Individual artifacts can then be placed into assemblies; some assemblies have mechanical properties—many of which can be represented by annotating the computerized representation of the design and placing constraints on its configurations. While CAD has become a ubiquitous tool among designers, integration of CAD with manufacturing activity has remained difficult.

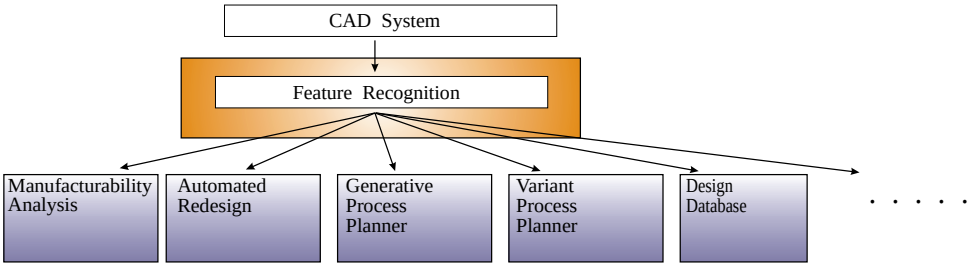


Figure 1.1: Feature recognition is the interface between CAD and downstream applications.

Integration of CAD with Manufacturing Applications. The vast majority of CAD activity involves the specification of the geometry and topology of artifacts. While CAD provides an excellent means of generating precise drawing and solid models of artifacts, much of the information crucial to the manufacturing process is not geometric. In integrating CAD with manufacturing applications, one major problem is how to automatically interpret low-level CAD information in a manner that is useful for automated manufacturing.

The commercial CAD/CAM marketplace is still very much in its infancy, with many vendors fulfilling limited niches, each with their own particular suite of integrated software modules and data structures. A **de facto** CAD/CAM integration scheme has yet to emerge.

Manufacturing **features** and feature-based representations have become a basic part of research in manufacturing systems integration. In the most general sense, **features** are higher level entities that model the correspondence between design information and manufacturing activities. For example, while basic CAD data consists of geometric and topological information, features can be used to represent how an artifact might be manufactured or assembled.

For the past decade, considerable research effort has studied the use of features as a way of abstracting higher level manufacturing data from lower level (geometric) CAD data. What has become evident is that a necessary component of any integrated Computer Aided Design/Computer Aided Manufacturing (CAD/CAM) environment is a tool for automatically recognizing manufacturing features directly from a CAD or solid model.

Feature recognition provides a communication medium between CAD and manufacturing applications, as illustrated in Figure 1.1. Feature recognition is a basic component of tools for design analysis and feedback, as well as for systems that automatically generate process plans and drive manufacturing processes. In all of these cases, recognizing features from a design is the means of providing a level of design understanding to manufacturing software systems.

1.2 Thesis Scope and Problem Statement

At a fundamental computational level, feature recognition is the problem of how a machine can be programmed to interpret the real world. In order to develop sound solutions to the problem one must, in a manner similar to problems in computer vision, artificial intelligence, and algorithms, develop a means to specify the problem, define algorithms for solving it, and measure its computational difficulty. The existing work has made significant advances in the development of various algorithms and approaches; however, little consensus has been reached after nearly two decades of effort.

In this thesis we develop a general approach for recognizing machining features from solid models of mechanical parts and for addressing some of the computational issues involved in building tractable and scalable solutions for automated feature recognition. The goal of this thesis is to further the development of systematic methodologies for the recognition of features from solid models of mechanical designs.

This thesis represents a step toward addressing these issues and making them more broadly applicable across many domains in which feature recognition is a critical technology.

1.2.1 Research Issues and Approach

This research is an effort toward enhancing the understanding of the computational issues in feature recognition. In this effort, we have had to address a number of basic research issues. The following subsections introduce these research issues and briefly outline how they are addressed by the work in the thesis.

1.2.1.1 Which Features to Recognize?

What general properties can be used to define a class of features that are useful for downstream planning and analysis? Often in existing systems, the approach pre-determines the types of features that will be within the scope of a system. Even within a fixed manufacturing domain, the notion of "which features to recognize" is application and implementation independent. If a better **a priori** definition for the features of interest can be put forward, the research and development community can then focus on developing new and improved algorithms for finding these features.

Chapters 3 and 4 of this thesis present definitions for a class of volumetric features that describe material removal volumes produced by machining operations on 3-axis vertical machining centers, such as the one pictured in Figure 1.2. Based on these definitions, we develop a specification for the recognition problem and trace-based algorithms for constructing feature instances from CAD data. Chapter 5 analyzes the complexity of these algorithms. Through this approach, we hope to improve the theoretical understanding of the feature recognition problem and its computational costs.

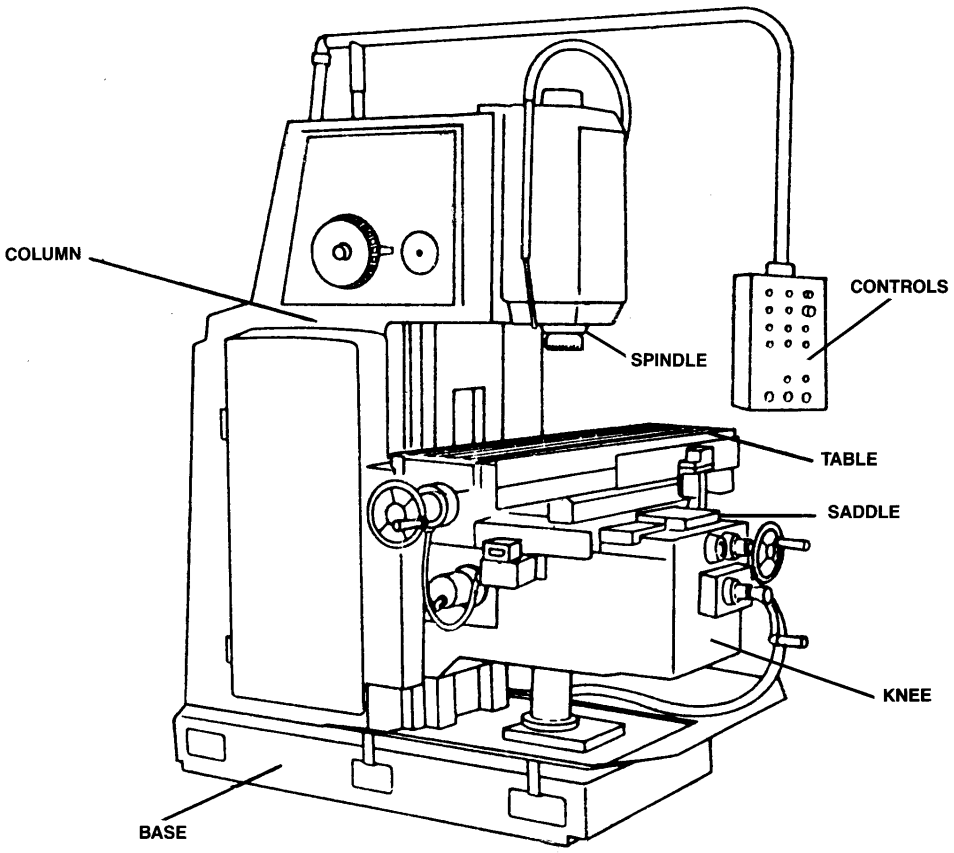
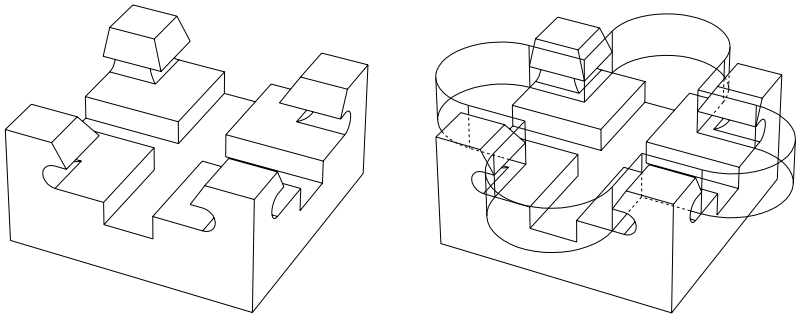


Figure 1.2: A typical machine tool: a vertical machining center [194]. Illustration used courtesy of Teledyne Cutting Tools, Inc.

1.2.1.2 Recognition of Interacting Features

When two or more features interact the individual feature instances can become distorted and information vital for reconstructing them can be removed. Figure 1.3 (a) shows an example of a solid model of a mechanical design containing a number of feature interactions. In particular, Figure 1.3 (b) shows the profile of one feature instance that one might like to find in order to perform effective manufacturing planning—the information remaining does not seem to provide sufficient information from which to generate this volume.

Devising a general means for handling feature interactions has proven very difficult for a variety of reasons—including the fact that there are differing notions of what it means for features to interact. The term "feature interaction" has become an area of intense research activity despite the fact there is little consensus on a definition. This has made it difficult to evaluate individual approaches. In most cases, interactions are handled with "rules of thumb" or a proliferation of heuristics that attempt to capture each special configuration



(a): part (b): profile of a target feature

Figure 1.3: A difficult interaction among features. In this case there are no faces left in the part from which to infer how to create the walls of the x-shaped pocket.

that might arise in a given application.

This thesis defines a categorization of feature interactions in Chapter 6 and discusses how the approach presented handles feature interactions from the various categories.

1.2.1.3 Recognition of Alternative Features

Just as there may be many different possible ways to achieve a goal state in AI planning, in manufacturing there usually exist several different ways in which a design can be realized. If features are in one-to-one correspondence with manufacturing operations, then there may be many different alternative sets of features that transform an initial workpiece into a final part. Early work on feature recognition focused on finding a single best feature decomposition for a given part. However, as the problem of feature alternatives has become better understood there is general agreement that consideration of alternative feature decompositions is critical.

Chapter 3 introduces notions of **well-behaved feature sets** and **feature-based models**. For example, in the machining domain it is often possible to machine a given design in several different ways, each corresponding to a different collection of machining features. Each such collection of features is an example of a feature model. In earlier literature, the term "feature model" meant different things to different researchers [53, 110]. This thesis synthesizes these concepts from earlier work in a definition of a **feature-based model** as a single, domain-specific feature-based representation of how to manufacture a given design.

The **well-behaved feature set** is a finite set of feature instances, each of which belongs to at least one (of the possibly many) feature-based models for the design. For example, Figure 1.4 illustrates the concepts of feature set and feature-based model in the machining domain. Consideration of alternative feature models is crucial to facilitate downstream activities such as design analysis or manufacturing planning where one wishes to optimize manufacturing cost or time.

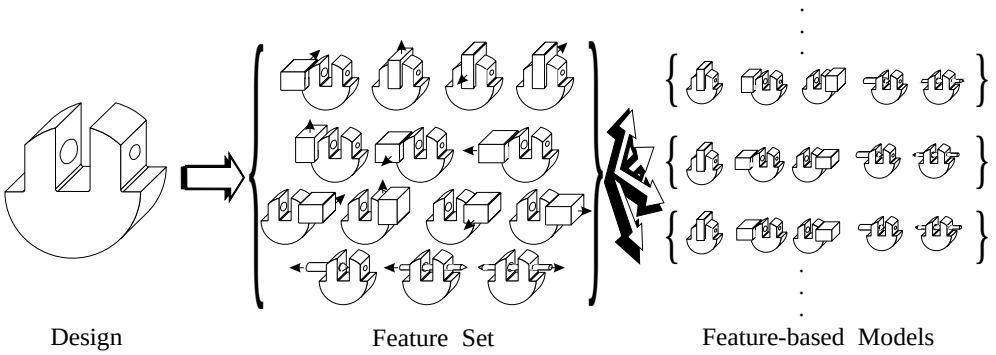


Figure 1.4: A design with alternate interpretations.

1.2.1.4 Building Feature-Based Models and the Interface to Downstream Applications

In some previous efforts, feature recognition was tightly coupled with a particular downstream application—usually manufacturing process planning. The emerging consensus is that feature recognition is a critical interface with a variety of downstream applications, as shown in Figure 1.1.

In those systems built for producing process plans or machining operation sequences, much of the process planning activity is considered during feature recognition: features are recognized and processes are selected "on the fly." Features that appear to be unlikely candidates for promising plans are discarded.

In this work we advocate separating feature recognition from planning. There are two primary motivations for this: first, feature recognition and its output might be used by a variety of downstream applications (as indicated above); therefore one should not force the heuristics specific to a single downstream application (such as process planning) into feature recognition. In the framework of an integrated environment for concurrent design and design analysis shown in Figure 1.5, feature recognition plays an important role in interpreting CAD data for each of the critiquing systems. Each individual critiquing system must analyze the design with respect to different manufacturing considerations. To this end, Chapter 7 presents techniques for post-processing of recognized features in order to produce more realistic features for downstream applications requiring machining features.

Second, when building a plan "on the fly," algorithms are making decisions about which features to discard without complete information. By analogy, in order to find the maximal element of a list one must check each element in the list. Similarly with recognizing features, one cannot reliably and efficiently perform manufacturing planning without complete knowledge of the available manufacturing options (i.e., one needs to have all of the features to determine which will be in the optimal plan). This is discussed more fully in Chapters 4 and 6.

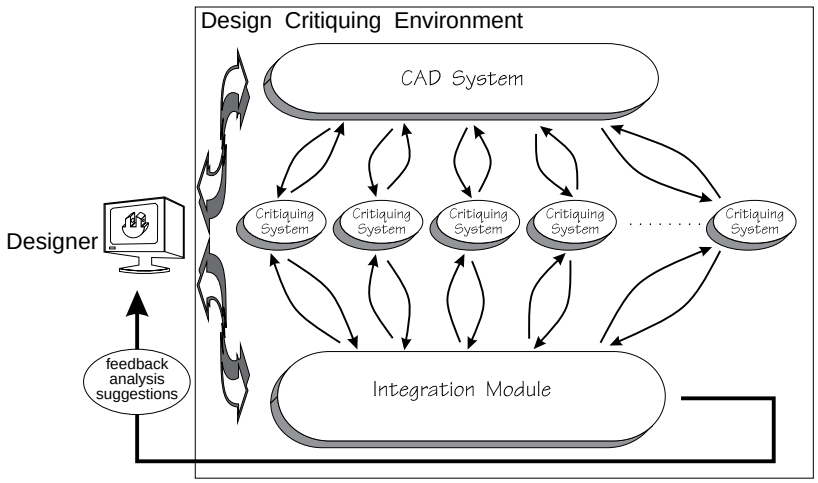


Figure 1.5: Integrated environment where computer-aided critiquing systems analyze the constraints posed on design by downstream manufacturing planning and production constraints.

1.2.1.5 Completeness

Given that a recognition system might be interfacing with a number of different CAM applications, better mathematical specifications are needed for the feature recognition "black box." In many respects, this issue shares much in common with current problems in software engineering. One needs to be able to pose questions about the input and the output of a software component for feature recognition and determine if there any limitations and restrictions. Without improved mathematical specifications, system integration is difficult and assessing the correctness of algorithms is impossible.

Chapter 4 defines and contrasts several specifications for the feature recognition problem based on the machining features introduced in Chapter 3. Chapter 6 defines a notion of **completeness** for feature-based modeling and feature recognition. In this context, **completeness** refers to the ability of an algorithm to unambiguously convey all of the information required to answer arbitrary questions about the manufacturing possibilities represented by the features.

1.2.1.6 Real-World Artifacts

Developments in feature recognition have been closely tied to advances in solid and geometric modeling. Many early feature recognition efforts focused on domains of polyhedral parts,

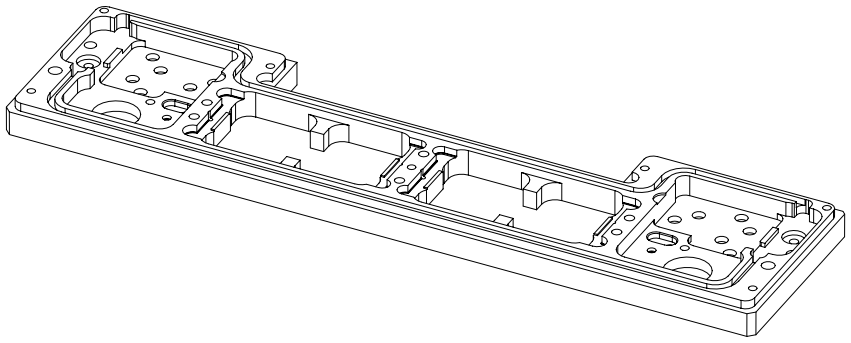


Figure 1.6: A solid model of a machined part from Allied Signal Corporation. This solid model contains 419 planar, cylindrical and conical surfaces and illustrates some of the complexities inherent in real world artifacts.

convex feature primitives, and algorithms to find patterns in graphical boundary representation data structures. As solid modeling technology has advanced, increasingly realistic and complex real world parts can be modeled. The advance in solid modeling representations, however, has not produced a corresponding advance in feature recognition technology to handle the more realistic parts. Feature recognition techniques are often based on assumptions about polyhedral objects; some approaches employ intrinsically costly (often NP-hard) algorithms which may prove difficult to scale to real-world parts.

In practice, mechanical parts are not polyhedra. For example, the machined part shown in Figure 1.6 is represented by 419 planar, conical, and cylindrical faces. Algorithms that work well on designs with dozens of surfaces may prove inappropriate for more complex parts with thousands of geometric and topological entities. This thesis addresses these issues in several ways. First, the definitions and algorithms of Chapters 3 and 4 are based on a realistic class of machining features [108] that describe realistic manufacturing operations. The definition of the domain of parts and the recognition algorithm accommodates most realistic configurations of machined artifacts that can be created with these features. Chapter 5 presents an analysis of the complexity of the algorithms of Chapter 4. Lastly, Chapter 7 presents techniques for incorporating information about the dimensions of real machine and cutting tools, as well as analysis of the accessibility of features during the machining processes.

1.2.1.7 Speed and Scalability

Engineering design is an interactive process and speed is a critical factor in systems that enable designers to explore and experiment with alternative ideas during the design phase. Even as hardware technology moves forward at an exponential pace, the complexity and size of software consumes every additional clock cycle and instruction. This growing complex-

ity, coupled with the intrinsic difficulties of handling the complicated artifacts of the real world noted above, requires the development of scalable solutions to the feature recognition problem.

Speed has not been an issue addressed in the context of feature recognition. Currently, for simple parts (polyhedral parts with a few dozen surfaces) recognition times can run minutes. If systems consider manufacturing issues such as tool accessibility (requiring some form of interference testing be performed), time can become a computational bottleneck in the integration framework of Figure 1.5.

Chapter 9 presents efforts toward developing a methodology for applying distributed, multiprocessor computing techniques to the feature recognition problem. Feature recognition has been approached using a variety of techniques, some of which are easier to parallelize than others. This thesis demonstrates that **trace-based** approaches to feature recognition (such as the one developed in Chapter 4) are particularly well suited for parallelization. In addition, the divide-and-conquer parallelism of Chapter 9 enabled the development of techniques for making geometric and topological modifications to the design that reduce the costs of solid modeling operations.

1.2.2 Research Objectives

This thesis presents a systematic approach for developing feature recognition systems for integration of CAD with downstream manufacturing activities. It is expected that this research will be enhance our formal understanding of the basic computational issues behind the feature recognition problem. In this way, improved and more rigorous feature recognition systems can be used as components in the next generation of CAD/CAM systems. In addition, this research reveals a gap at the application level between the development of theory and the development of CAD/CAM systems. We believe that further research on the specification, completeness, and complexity of geometric problems in design and manufacturing can enhance our understanding of the basic problem and lead to the production of better CAD/CAM software tools.

A proof-of-concept implementation of a feature recognizer based on the algorithms presented in this thesis, dubbed **F-Rex**, has been developed and integrated into **IMACS**. IMACS is a system for interactive manufacturability analysis and design critiquing developed at the University of Maryland at College Park's Institute for Systems Research. IMACS and the implementation of F-Rex are discussed in more detail in Chapter 8.

1.3 Thesis Organization

This thesis comprises 11 chapters. The contents of each are summarized as follows: Chapter 2 presents a summary of previous research work in the areas related to this thesis—in particular, research in the areas of solid modeling, features and feature-based manufacturing. Chapter 3 introduces a class of machining features and presents definitions for feature-based models and for primary and well-behaved features for the machining domain.

Chapter 4 presents a problem specification and develops a trace-based approach to recognizing instances of the machining features defined in Chapter 3. Chapter 5 addresses issues of computational complexity and analyzes the complexity of the algorithms presented in Chapter 4. Chapter 6 builds a notion of **completeness** for feature-based modeling and feature recognition and presents demonstrates the completeness of the algorithms presented in Chapter 4 over the class of well-behaved features defined in Chapter 3.

Chapter 7 uses information about the application domain of machining to improve and enhance the feature recognition process. In particular, Chapter 7 presents techniques to modify the set of features returned by the algorithms of Chapter 4 to improve their correspondence to actual machining operations and exclude those that are likely to be unrealistic for downstream applications. Chapter 8 gives an overview of the IMACS design critiquing system and of the implementation of a prototype feature recognition system **F-Rex** developed with the methodology described in Chapters 3 through 7.

Chapter 9 introduces a parallelization of the feature recognition techniques from Chapter 4 using distributed algorithms. In addition, this chapter outlines several methods that can exploit the parallelism to simplify individual feature recognition subproblems. Chapter 10 present a discussion of how the terminology and techniques presented in this thesis can be applied to other manufacturing domains. Lastly, Chapter 11 summarizes the research contributions of this thesis and outlines a number of research issues for the future.

Chapter 2

Survey of Related Research Work

This chapter presents a summary of previous research work in the areas related to this thesis. In particular, the chapter gives a brief history and summary of the field of solid modeling. It overviews the concept of features and the work on feature-based modeling and applications, including the fundamental literature on feature recognition from solid models. The chapter also covers some of the subject areas closely related to features and feature-based modeling, including process planning, GT code generation, manufacturability analysis and redesign. Lastly, it summarizes the current research problems and some of the open technical issues in automated feature recognition.

2.1 Solid Modeling

The field of **solid modeling** covers a wide area of activity directed toward the representation and manipulation of 3-dimensional surfaces and volumes. Solid modeling is a critical element in a growing number of application areas, including robotics, simulation, analysis, rendering, and (most relevant to this thesis) CAD/CAM. A distinction exists in the literature between the field of solid modeling and that of **geometric modeling.** Geometric modeling is concerned with the representation of surfaces (analytics, splines, etc.) and their manipulations. This is intimately related to solid modeling; however solid modeling is more concerned with those objects bounded by such surfaces.

Conceptually, work in solid modeling has proceeded in three levels of abstraction [81]:

1. **Symbolic and arithmetic foundations** represent the lowest level of abstraction and are concerned with the computer hardware support of integer and floating point arithmetic as well as with the abilities of a programming language to express computations and manipulate memory. Examples include the development of data structures and handling of issues of mathematical robustness.

2. **Mathematical and algorithmic infrastructure** describes the fundamental operations as implemented in the foundation above. Examples include operations for creating solids, for performing interference tests among solids, and for developing efficient

algorithms for basic manipulations (such as solid-solid intersection or point-solid classification).

3. **Application and user interface** is the highest level of abstraction, focusing on the development of applications in terms of the infrastructure created by the above. Traditionally, research in the lower levels of abstraction has been pursued because of the perception that solid modeling can lead to great enhancements in our ability to create more effective designs and easily perform complex analysis on them.

   As levels (1) and (2) have become better understood, increasingly sophisticated and robust solid modeling systems have emerged. This has led to increased research activity at the application level. In particular, given that one has adequate solid modeling tools at one's disposal, one begins to ask questions such as: How should functionality be represented? or How does one represent manufacturing operations and their effect on a model of a design?

This thesis is primarily concerned with problems that lie in the third category. There are numerous important concepts from the core solid modeling literature that are directly relevant to the work described in this thesis. The remainder of this section very briefly reviews some solid modeling concepts and terminology which will be used throughout this thesis—in particular, concepts concerned with manipulation and representation of solid models.

For more in-depth coverage of the of the field of solid modeling, interested readers are referred to the texts by Hoffmann [81], Mäntylä [116], Mortenson [128], Faux and Pratt [54] and Woodwark [211, 18]; as well as the proceedings from the recent conferences on solid modeling and applications [153, 154, 155]. For an overview of the architecture of solid modeling systems, readers are referred to [124]. The classic text on computer graphics of Foley and Van Dam [56] also covers solid modeling and its relationship to graphics and rendering. For information regarding commercial solid modeling systems, readers are referred to the product and reference information for Spatial Technologies ACIS modeler [177, 183, 184, 185, 178, 180, 181, 120, 182, 179] as well as the EDS/UNIGRAPHICS Parasolid solid modeling kernel [176, 175] and the PADL-2 project from the University of Rochester [21].

2.1.1 Representation of 3D Surfaces and Solids

There are three broad classes of schemes for representation of solid models [116, 149]:

1. Decomposition approaches that model a solid as collections of primitive objects connected in some way. Examples of these data structures include quad-trees and oct-trees [163], which represent space as collections of primitive cells (usually cuboidal).

2. Constructive approaches that model a solid as a combination of primitive solid templates. For example, a common approach is **constructive solid geometry** (CSG), which represents a solid as a boolean expression on some set of primitive solids. Figure 2.1 gives an illustration of a CSG tree for a simple part in terms of primitive blocks and cylindrical solids. In a CSG tree, the leaves of the tree contain primitive solids

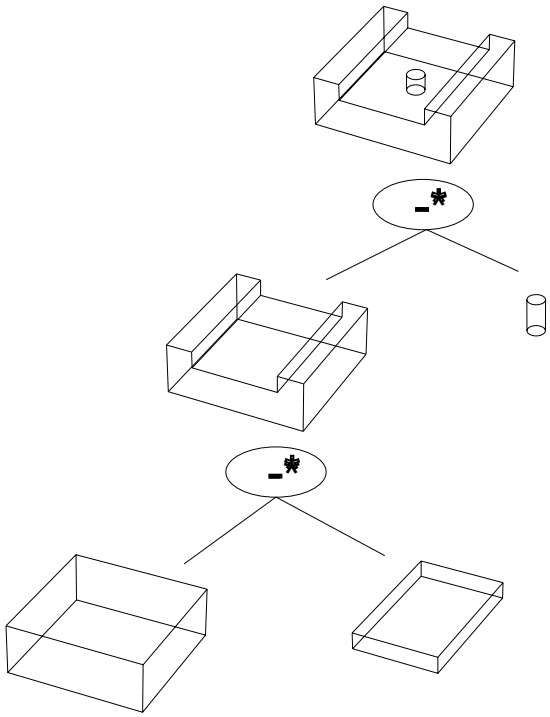


Figure 2.1: A CSG Tree.

(usually blocks, prisms, cylinders, etc.) and the tree's nodes contain operators on the primitives (in this example boolean subtraction).

3. Boundary-based approaches that model a solid using a data structure that represents the geometry and topology of its bounding faces. In recent years the boundary-representation (**b-rep**) approach has emerged as the dominant representation scheme in solid modeling and CAD systems. This is due in large part to the representational power and flexibility of boundary models, as well as to recent advances in numeric computation that overcame earlier problems with models becoming unstable and inconsistent.

The work described in this thesis assumes a boundary-representation solid modeler, and the notation and algorithms will show a bias toward the b-rep format. In particular, a b-rep usually consists of a graphical structure that models an entity's topology. The connections between the nodes in the b-rep graph represent the connections between the topological components of the entity's boundary. These topology nodes then contain pointers to their underlying geometric entities; for example, a **face** of a solid is a topological entity (represented as a collection of bounding edges) and it has associated with it a **surface** (represented as an equation). An illustration of the distinction between geometric and topological infor-

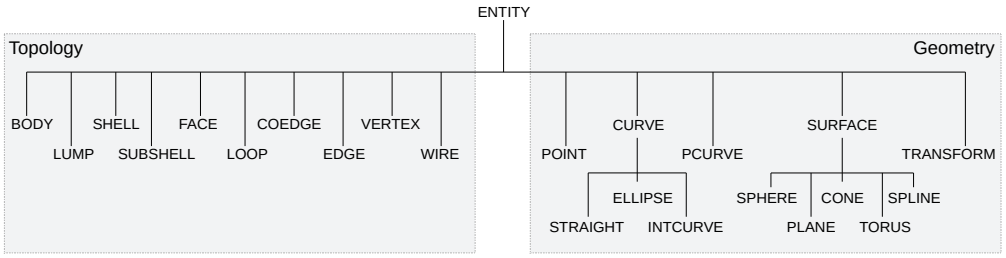


Figure 2.2: The distinction between geometric and topological information within the ACIS Solid Modeler [184].

mation from the ACIS Solid Modeling Kernel is given in Figure 2.2.

One of the most popular of these b-rep structures is the **winged-edge** representation [81] and its variations. For more information on boundary representation data structures, interested readers are referred to [81, 116, 207, 209].

Manifold and non-manifold solids. Operation on these data structures can give rise to many different configurations of solids. Intuitively, a **manifold** solid is one whose boundary can be uniformly unfolded onto a 2-dimensional plane. The manifold condition excludes, for example, solids whose bounding surfaces are self-intersecting. Non-manifold objects can be of mixed dimension. For example, the object shown in Figure 2.3 (b) does not have a closed boundary; the object shown in Figure 2.4 (c) has a boundary containing a dangling 2D surface.

In this thesis we will be primarily concerned with manifold objects. Some of the algorithms presented in subsequent chapters, however, will deal with objects of differing dimensions (i.e., boolean operations on 2D faces, building 3D solids from 2D cross-sections, etc.).

2.1.2 Manipulation of Solids

This section briefly describes some of the common operators used to manipulate geometric and solid models. In addition to operators common for handling 2D shapes in computer graphics [56] such as coordinate transformations, rotations, and scaling, there are additional functions specific to the nature of geometric modeling. Computer graphics is usually concerned with the production of images for display on a display device—the representation used to generate the picture is largely unimportant. Geometric and solid modeling are concerned with these underlying representations and how to maintain their consistency under mathematical operations.

The remainder of this section briefly introduces three common classes of operations: Euler

operations, boolean operations, and sweeps. We shall make extensive use of some of this terminology throughout the remainder of this thesis.

Euler Operators. Euler operators act on the topology of a boundary representation data structure. Starting from an idealized primitive "solid" consisting of one face and one vertex, one can create a solid through a series of local and global manipulations. Local manipulations can **create** (or **delete**) vertices, edges, and loops; global manipulations can be used to create (or delete) holes or to divide a body into multiple bodies.

Euler operators have a number of elegant mathematical and algorithmic properties, including the ability to describe all valid boundary data structures through a finite sequence of Euler operators.

A number of solid modeling systems have been based on Euler operators, most notably Mäntylä's Geometric WorkBench [116]. For a more detailed description of Euler operators, readers are referred to [116].

Boolean Operations. Solid models can be considered as bounded point sets. Boolean operations union ($\bigcup$), intersection ($\bigcap$) and difference ($-$) can be defined on solid models based on their action on point sets. For example, Figures 2.3 and 2.4 illustrate boolean difference and intersection, respectively.

The problem arises that solids may lose their "solidness" under boolean operations. For example, in Figure 2.3 (c) the intersection of A and B has created a 2D entity. In Figure 2.4 (c), the intersection of A and B has left a solid with a dangling 2D surface. In both of these cases, the result of the operation is not entirely 3D.

Regularized boolean operations correct these ambiguities. If A is a solid, $i(A)$ is the **interior** of A (point set A minus its boundary) and $c(A)$ is the **closure** of A (point set A plus its boundary); the regularized boolean operations are defined as follows [116]:

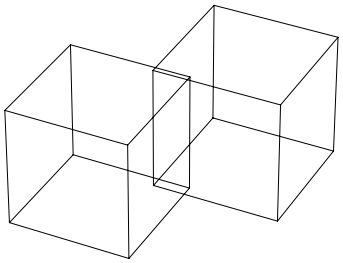
$\bigcup^*$, Union: $A \bigcup^* B = c(i(A \bigcup^* B))$;

$\bigcap^*$, Intersection: $A \bigcap^* B = c(i(A \bigcap^* B))$;

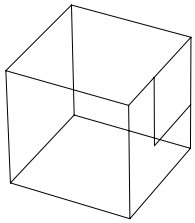
$-^*$, Difference: $A -^* B = c(i(A -^* B))$.

Examples of the effects of regularized boolean operations are shown in Figure 2.5.

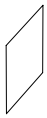
Sweeps. Sweeps are another common representation and tool for manipulation of solids. Sweeps can be used as a basis for the generation of primitives; for example, the rectangular volume shown in Figure 2.6 (b) can be modeled as the sweep of the closed edge profile (shown in Figure 2.6 (a)) along a vector for some fixed distance. Maintaining the topological consistency of models under sweeps has emerged as a problem of some delicacy. Much in the same way as with boolean operations, a sweep may not always result in a uniformly 3D solid model—one example of this can occur when the volume swept out by an edge profile along some curve is self-intersecting. Sweeping of 3D solids, as shown in Figures 2.6 (c) and (d)



(a): Two blocks, A and B



(b): Subtract, $A - B$



(c): Intersect, $A \cap B$

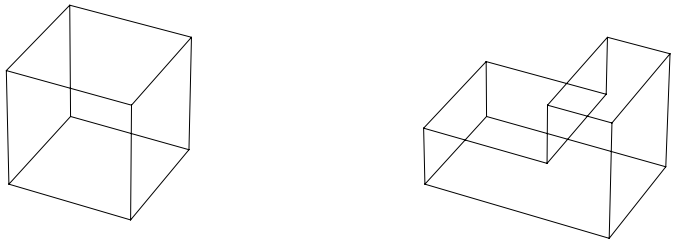
Figure 2.3: Non-regularized boolean operations.

for a polyhedra, introduces additional mathematical difficulties. In recent years, techniques have been developed for handling many of these situations [121].

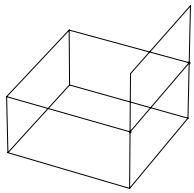
Sweeping of solids has important uses in manufacturing applications. One such application is that the representation of the effects of operations (such as material removal during machining) can be approximated as the path taken by a rotating 3D cutting tool when moved along a tool path.

2.2 Features and Feature-Based Modeling

The fundamental representation for mechanical designs within a CAD system is the solid model. For the most part, this exclusively geometric and topological representation is rather limiting. While solid models provide increasingly good ways to model shape and form, they do not lend themselves well to direct reasoning about manufacturing activities. The concept of a **feature** attempts to bridge this gap by modeling the relationship between the local



(a): Solid A (b): Solid B



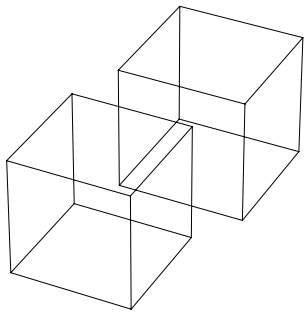
(c): Intersect, $A \cap B$

Figure 2.4: Another example of non-regularized intersection from Mäntylä [116].

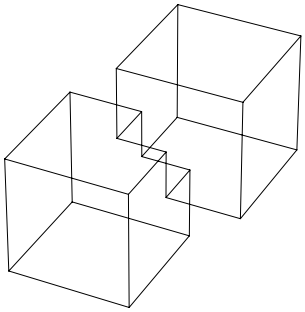
geometric and topological configurations of a design and the higher-level abstractions. In this way, semantic information can be conveyed along with the shape.

Features and feature-based approaches have proven popular in a variety of CAD/CAM application domains. The reason for this increasing popularity is that for most CAD/CAM problems the design needs to be interpreted in terms of the needs of the particular application. Significant work has been directed toward defining sets of features to serve as a means of communication between design and manufacturing [172, 122]. At present, however, most researchers are convinced that no single set of features can satisfy the requirements of every possible design and manufacturing domain.

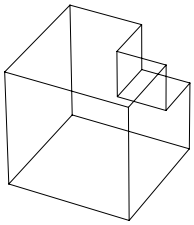
Feature-based characterization of design information has long been viewed as vital for design analysis and data exchange. In the research community, features have come to mean different things for different research projects—in the absence of a universal feature class, each researcher adopts her or his own definitions and types to suit the goals of the particular application. This has led to the need to formalize representation schemes for features and data exchange standards. In the absence of an agreed-upon standard, many research groups



(a): Two blocks, A and B



(b): Unite, $A \cup^* B$



(c): Subtract, $A -^* B$

(d): Intersect, $A \cap^* B$

Figure 2.5: Regularized boolean operations.

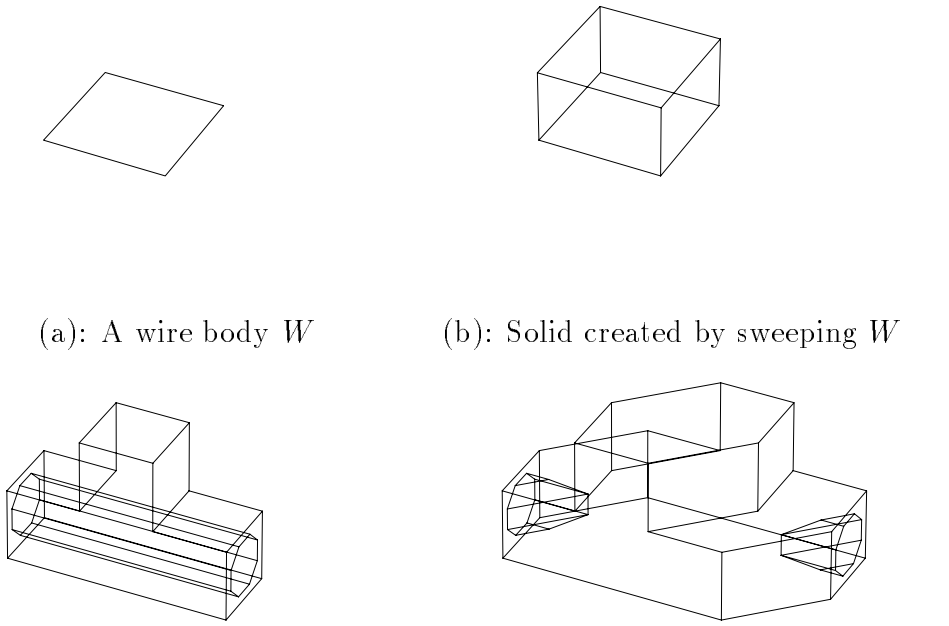
have created their own feature classes, languages, and hierarchies.

This feature technology relies heavily on the geometric and topological capabilities of solid modeling and CAD systems. Existing features research deals predominantly with machining applications—this has been facilitated by easy manipulation of the volume and surface information important to machining by existing solid modeling systems.

The remainder of this section reviews some of the basic concepts in features and feature-based modeling, with a special emphasis on work in automated feature recognition. A comprehensive overview of feature-based manufacturing can be found in [170].

2.2.1 Typical Feature Types

Independent of a manufacturing domain or a particular CAD/CAM application, there are several generic types of features that are common in the literature. While the following enumeration is by no means complete, most of the notions for and definitions of features that exist in the literature today are covered by these categories.



(a): A wire body W (b): Solid created by sweeping W

(c): A solid body A (d): A new solid created by sweeping A

Figure 2.6: Two examples of sweeping.

Form Features. At the most general level, a **form feature** can be defined as a local geometric, topological, or volumetric configuration on a part with engineering significance. Form features are meant to describe generic shapes with regular geometric and topological descriptions. A form feature is usually defined by a set of parameters and a set of conditions. For example, the notch shown in Figure 2.7 (a) can be represented with parameters defining the dimensions of the rectangular volume of the notch. Its relationship to the part is captured by a condition states that the three faces of the notch match with faces of the part.

A form feature might be related to a manufacturing operation, such as in Figure 2.8 where the definition for holes can be related to drilling operations. Other common examples of form features include slots, pockets, and shoulders, as illustrated in Figure 2.7. The faces corresponding to a "hole" might be related to specific downstream manufacturing activities such as machining or fixturing.

Surface Features. **Surface features** are usually defined as collections of 2D patches on the boundary of a solid. In most cases these 2D patches are simply collections of faces

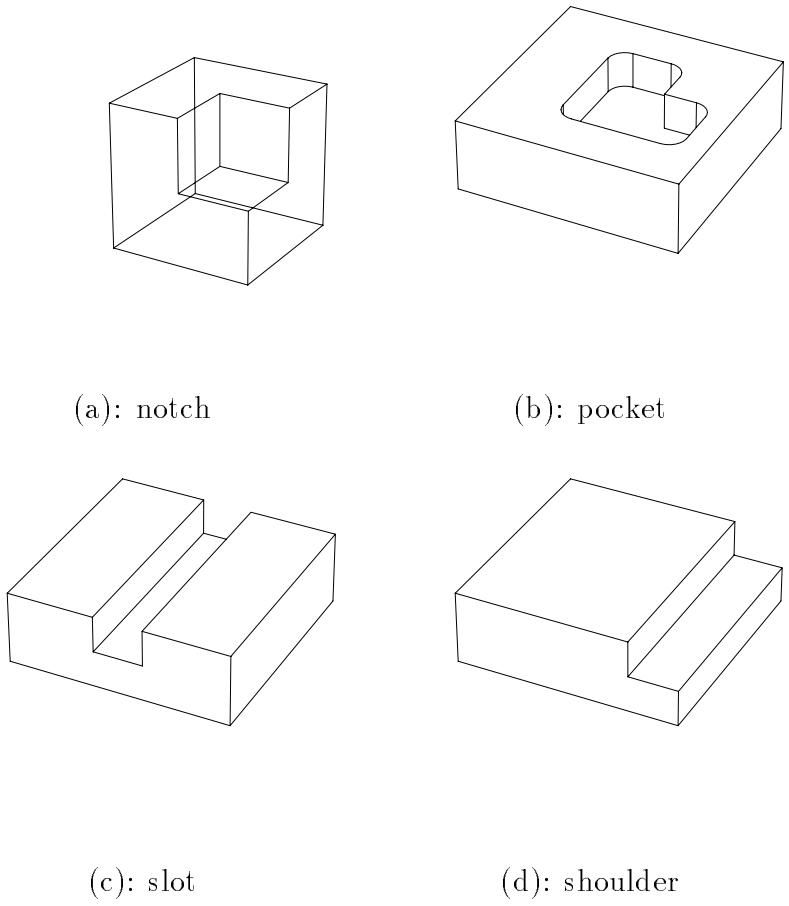


Figure 2.7: Examples of typical form features.

of the solid. Surface features provide a natural way to represent a variety of important manufacturing information such as:

1. **Tolerances.** Certain kinds of design and manufacturing tolerances can be modeled as relationships between surface features.

2. **Fixture design.** Surface features can describe the fixturable part surfaces.

3. **Manufacturing attributes.** Features can relate surface finish and roughness to manufacturing operations.

4. **Assembly planning.** Surface features can be used in robotic assembly planning to reason about possible placements for grippers.

5. **Sheet metal manufacturing.** The effects of sheet metal manufacturing operations (such as bending) lend themselves to description using surface features.

Radius:	R
Axis Direction:	$\bar{\mathbf{A}} = (a_x\ a_y\ a_z)$
Axis Origin:	$\bar{\mathbf{P}} = (p_x\ p_y\ p_z)$
Local Z-Axis:	$\bar{\mathbf{Z}} = \bar{\mathbf{A}}$
Local Y-Axis:	$\bar{\mathbf{Y}} = \frac{\bar{\mathbf{P}}}{\|\bar{\mathbf{P}}\|}$
Local X-Axis:	$\bar{\mathbf{X}} = \bar{\mathbf{Y}} \times \bar{\mathbf{Z}}$
Local & Semi-Infinite Accessibility:	$-\bar{\mathbf{A}}$ or $+\bar{\mathbf{A}}$ or both

(a)

1. A **Hole** begins with an entrance face.
2. All subsequent faces of the hole share a common axis.
3. All faces of the hole are sequentially adjacent.
4. The hole terminates with a valid hole bottom.

(b)

Figure 2.8: Definitions for a cylindrical hole from Vandenbrande [201] (a) and Henderson [77] (b).

Volumetric Features. A **volumetric feature** is a 3D shape bounded by a collection of surfaces. In recent years there has been an increased use of volumetric features, in particular to describe the material removal volumes created by machining operations. Volumetric features have a variety of advantages over surface features [144]; primary among them is that manufacturing process information (in particular in the machining domain) is more easily described volumetrically. By way of an example, a machine tool moves a cutting tool through 3D space. The volumetric intersection of this sweep with the current workpiece corresponds to the effect of the machining operation.

An observation worth noting about form features is that they can describe either surface or volumetric features. A second point is that the distinction between types of form features is usually solely based on geometric and topological considerations. From the point of view of manufacturing processes, these distinctions are often arbitrary—form features need not have any direct relationship to a specific manufacturing process. For example, while the features in Figure 2.7 represent different form features, they all can be machined using a end-mill on a vertical machining center. Hence, from a machining point of view, they can be considered to be instances of the same type of **machining** feature.

Recently there has been interest in features for other manufacturing life-cycle considerations, such as features with associated functional engineering significance. El Maraghy et al. [51] proposed and implemented a design tool employing functional features. Functional features were also a key part of the work of Schulte et al. [168]. Henderson and

Taylor [78, 79, 193] developed a system for conceptual modeling in an effort to represent features, functionality, dimensions, and tolerances within a solid modeling system.

2.2.2 Manufacturing Features

The feature concept had its beginning with the process planning of machined parts [122]. Historically, process planning systems for machining have employed features to represent machining operations. Manufacturing features have grown to fill important roles in a variety of manufacturing application domains, such as assembly, inspection, etc.

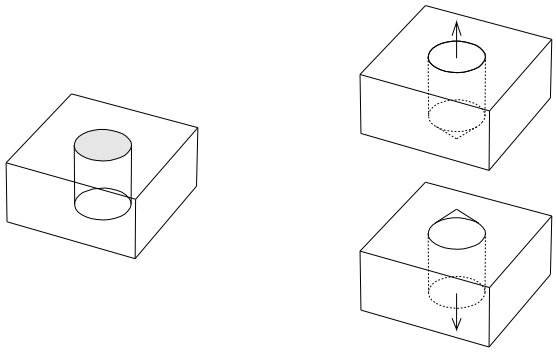
Each different manufacturing application domain has different requirements for its specific feature definitions. Requirements and specifications for satisfactory manufacturing feature definitions in various domains have been addressed numerous times in previous work. Pratt [122] states that features should be capable of representing, among other things, component primitives, connectivity between entities, and geometric, topological, and size constraints. As pointed out in [52], some of the properties that features should be capable of encoding are:

1. Functional: those relating to application-specific functions and operations.
2. Assembly: those modeling the relations between design attributes and datum hierarchies.
3. Precision: those representing tolerances, datums, and surface finishes.
4. Form: those for geometric entities, primitives, and boolean operations on primitives.
5. Manufacturing: those embodying domain specific-information about manufacturing processes, process capabilities, and tooling requirements.

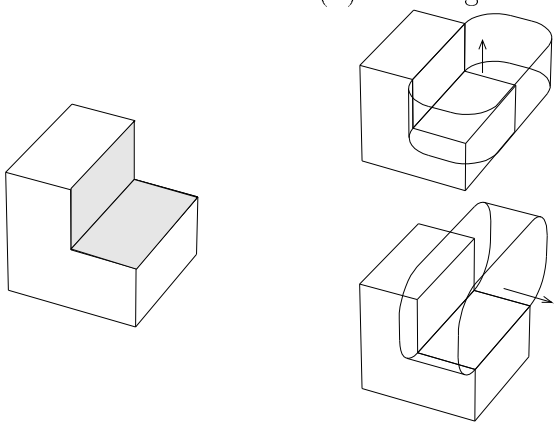
A number of attempts have been made to define and classify **manufacturing features** [22, 62, 200, 24]. Although there are differences among these approaches, many of them share important similarities. For example, a machining feature usually corresponds to the volume of material that can be removed by a machining operation. In general, manufacturing features usually have associated geometry and tolerance information that can be matched with the design attributes of the part and can be used to parameterize the manufacturing operations.

For manufacturing domains that involve discrete manufacturing operations (such as machining, sheet metal bending, forging, etc.), a feature can be thought of as a parameterized object. This notion of features is becoming increasingly common and is gaining widespread acceptance [152, 24].

Note that **manufacturing features** are not necessarily equivalent to the usual notion of **form features.** A manufacturing feature may be defined as a form feature; however, form features need not have any direct correspondence to manufacturing operations. An illustration of this distinction is given in Figure 2.9.



(a): form feature: hole (b): drilling features



(c): form feature: shoulder (d): milling features

Figure 2.9: Form features and their equivalent machining features.

2.2.2.1 Machining Features

The manufacturing domain that has received much of the attention of researchers in academia and industry has been machining. A **machining feature** attempts to capture the effect of a cutting tool (such as a drill or a mill) used in a machining center to operate on a workpiece. In general, machining features model material removal operations. For example, a machining feature might be defined as the volume swept by a cutting tool during machining and can be represented as a parameterized solid [24, 131].

Machining features have been defined both as surface features and as volumetric features [144]. When defined as surfaces, machining features are collections of faces that are to be created by a machining operation. Historically, the dominant approach was to define machining features as collections of surfaces. In recent years, however, an increasing number of researchers are adopting the position that machining features should be defined as parameterized solids that model the volume removed by the machining operation. Volumetric machining features provide a more comprehensive representation of the actual machining operation [144] and are becoming the norm in the current generation of feature-based CAD/CAM systems.

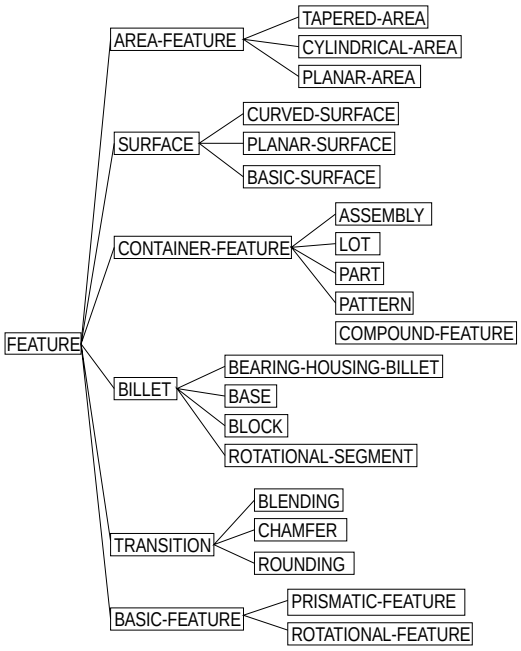


Figure 2.10: Example feature taxonomy from Laakko and Mäntylä [110].

Classifying machining features. There have been numerous attempts to classify machining features and devise feature hierarchies. In addition to the MRSEV Library for 3-axis machining features of Kramer [108, 107] mentioned in the next section, Nau et al. [130] developed a machining feature library for process planning. Similarly, Gindy [62] presented a feature hierarchy describing the geometric information required for a class of form features for process planning. Figure 2.10 presents a machining feature taxonomy developed by Laakko and Mäntylä [110].

In addition to academic efforts, there have been numerous feature categorization attempts made in industry. One of the early efforts was that of the Computer-Aided Manufacturing International consortium (CAM-I) [171]. CAM-I produced numerous reports and studies of the technological issues relating to features in the context of automated manufacturing.

Working with CAM-I, Pratt and Wilson [143] produced a study of the use of form features within a geometric modeling system. In this work, they provided a methodology for defining features and performing design in either boundary-representation or CSG-based solid modeling systems.

Another CAM-I study conducted by the John Deere Company [22], attempted to exhaustively enumerate all possible features that might be used for process planning of machined parts. This study attempted to produce a comprehensive list of all manufacturing features. This list was then to be used to generate higher-level descriptions of the part and deduce feature relationships for every component in the company. This work was survey-like in nature

and did not provide any mathematical structure or definitions for the feature types; nor did it attempt to determine the relationship between features and manufacturing operations.

In another industrial effort, Allied-Signal, as part of Brooks et al.'s [20] XCUT project, presented a classification of machining features for process planning.

The two most mathematically comprehensive approaches for defining machining features are those of Chang [24] and Vandenbrande and Requicha [200]. Chang's feature definitions are based on the shape of the cutting tool and the cutting trajectory. Vandenbrande and Requicha [200] adopt a similar method and classify tool-swept volumes as different types of volumetric machining features.

2.2.2.2 Other Types Of Features

Although much of the research work on features has dealt with machining, other recent work has employed features for inspection planning [119], part coding [109], and other manufacturing processes, such as injection molding [141]. Stanford University's NEXT-Cut [35] used features for concurrent design of both the product and the manufacturing process while considering constraints imposed by a variety of manufacturing domains. In other domains where designing for properties other than manufacturability (such as disassembly, reuse, etc.), the feature classes of interest may be quite different than those currently employed.

2.2.2.3 Feature Definition Systems and Feature Definition Languages

Because feature needs vary greatly by application domain, there have been a variety of efforts directed toward the development of general **feature definition languages**. The basic premise of a feature definition language is to give the user flexibility in defining their own feature class tailored to suit the needs of their particular application.

Examples of advanced feature-based modeling systems capable of incorporating user-definable functional features include: The ASU Feature Testbed [173, 172], which includes a generic feature mapping shell that allows mapping features from one application to other; and the systems developed by Laakko and Mäntylä [110] and El Maraghy et al. [51].

Some feature recognition systems have also incorporated the ability to have user-definable feature classes for the purposes of customization and extendibility [161, 110]. Recent work has applied Object-Oriented Design (OOD) methodologies to CAD tools to incorporate feature classes, customization, and recognition in a unified system [110]. In such systems, users can define classes of features relevant for individual applications. Another approach to customization is feature languages [41, 198]. In this case, the user defines a grammar and primitives for a dialect of features and their means of recognition.

2.2.2.4 Standards, Feature Classes, and Feature Hierarchies

One of the major objectives of features research has been the integration of CAD with CAM; this, in turn, has generated a great deal of attention and interest from the standards community about defining standard feature classes. The objective of standardization is to

get everyone to agree on classes of features, or at least to agree on how to define classes of manufacturing features, to enable integration of applications and data sharing among manufacturing applications at the feature level.

STEP. STEP is the International **ST**andard for the **E**xchange of **P**roduct Model Data being developed by the International Organization for Standardization (ISO). PDES stands for **P**roduct **D**ata **E**xchange using **S**TEP and it represents the activity of several organizations in the United States in support of STEP. The organizations involved with PDES comprise many corporate, government, and standards development entities.[1]

Description data in STEP is handled by defining an information model in the EXPRESS data modeling language [166, 186] for each type of data required. Once an information model is defined, data for representing a specific product can be represented using the STEP rules for mapping EXPRESS to a physical file [8, 7, 199]. The EXPRESS model defines the data entities that describe the class of objects in the domain.

Defining Features in STEP. A means for describing generic classes of features for the purposes of data exchange has been evolving within the standardization community. STEP Part 48 [46] deals with the characterization and representation of **form features** to cover a wide variety of shapes of industrial interest. The basic idea behind this document is that feature data occurs at three levels of abstraction:

1. **Application features** have domain-specific connotations that are not directly related to their shape. For example, an application feature named "assembly hole" conveys both functional (fastening in an assembly) and process (the hole might be drilled) information.

2. **Form features** are generic shape properties of a product (as mentioned above) with **no application connotation** and **no presumption about their representation.** For example, "circular cross-section" might be the name of a form feature that describes the assembly hole.

3. **Form feature representations** are employed to represent shape properties. For example, a "sweep" of a "circular profile" above might be used to model the circular cross-section form feature.

STEP Part 48 is concerned with feature representation at levels 2 and 3. Currently the standard is in a state of flux and what exists of it does not contain all of the information one

[1]The organizations include: IGES/PDES Organization, a voluntary standards organization that participates in developing the STEP standards; many IPO/ISO committees including the Form Features Committee and the Manufacturing Technology Committee; PDES, Inc., a consortium of major companies whose goal is advancing the implementation of PDES; and the National Institute of Standards and Technology, which has established a National PDES Testbed under the funding of the Computer-aided Acquisition and Logistic Support program of the Department of Defense.

would need to associate with feature definitions. For example, the ability to handle information about tolerances, user-defined form features, functionality, manufacturing process, and size is non-existent within Part 48. These issues may be addressed in later revisions to Part 48 or in other standards (as yet undefined).

STEP Application Protocol 224 (AP 224) is for defining mechanical engineering products for process planning using form features. STEP AP 224 builds on the generic representation mechanisms of Part 48 and defines specific form features relating to the shape of a single part that can be used in manufacturing. Specifically, the document provides a definition for **machining features** to facilitate the identification of manufacturing shapes that are human and machine-interpretable. Within AP 224, machining features are a class of shapes which represent volumes to be removed from a part by machining. In this way, the information represented using AP 224 to drive decisions made by computer-aided process planning systems.

While STEP does provide some mechanisms for representing geometric and topological form features, there exists no definitive structure for representing and exchanging all the relevant information associated with a manufacturing feature. Shah [174] describes his investigation of the STEP form features model.

Example of a STEP-based Feature Library. Kramer [108, 107] developed a library of Material Removal Shape Element Volumes (MRSEVs) as a means of categorizing the shapes of volumes to be removed by machining operations on a 3-axis machining center. MRSEVs can be defined using the EXPRESS modeling language and STEP form features. Kramer has written such definitions for a subset of the MRSEV library, and has defined the rest of the MRSEV library using an EXPRESS-like language.

MRSEVs features are volumetric form features and the MRSEV hierarchy provides a framework for describing a large class of entities of interest to machining. Each entity type has a number of required and optional attributes. MRSEV instances have been used for applications such as process planning and NC-program generation [106]. Kramer's main MRSEV types include linear swept features, edge-cut features, ramps, and rotational pockets. Figure 2.11 provide an illustration of the feature subclasses in the MRSEV hierarchy.

2.2.3 Feature-Based Design

While creating a design of a part, the designer interacts extensively with her or his CAD tool. A **design feature** is a shape or form that has significance to the design engineer. In many cases the design process on a CAD system consists of a sequence of operations and manipulations performed with design features.

Design features can have functional significance or aesthetic attributes. In this way a designer working with design features can more readily construct a detailed design of the artifact that she or he has in mind. In some systems, design features correspond directly to manufacturing operations. The basic idea is that as the design process progresses, feature information is recorded and becomes part of the CAD model. For example, if the designer

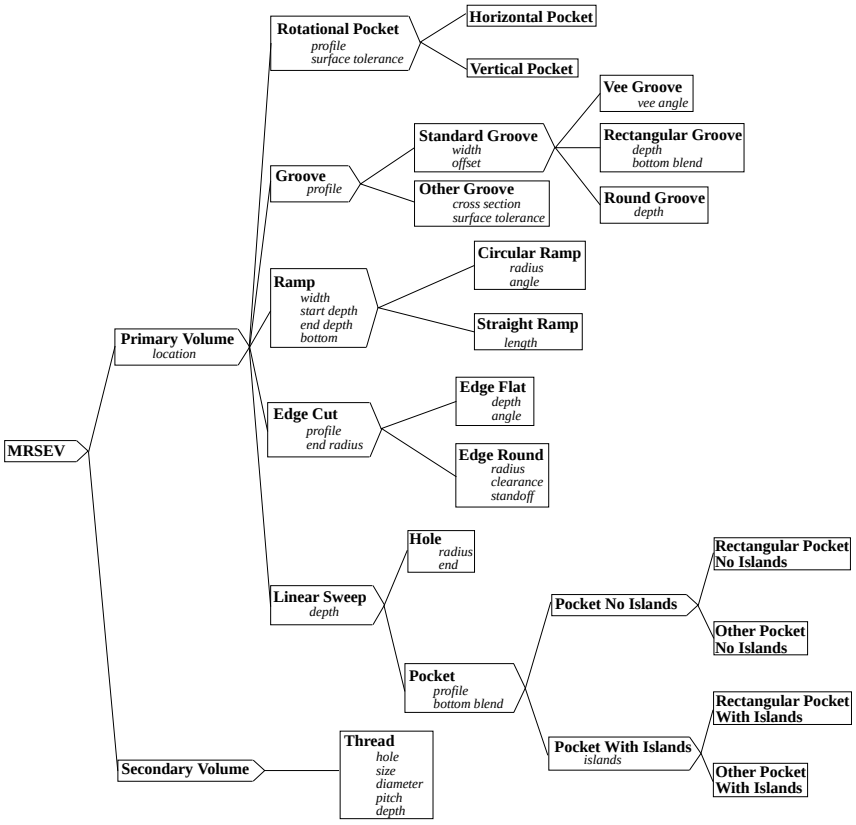


Figure 2.11: The MRSEV feature taxonomy from Kramer [108].

is designing a machinable component, the features might correspond to removal volumes created by moving cutting tools. When the design is completed, the CAD model contains all the information about how the designer arrived at the final product. Figure 2.12 provides an example of a typical design feature: a counter-sunk hole.

There are several prototype systems based on this approach, including NEXT-Cut [35] and the Quick Turnaround Cell (QTC) [24]. They provided a feature-based design interface and model the part directly in terms of machining features. In the commercial world, Parametric Technologies' Pro/ENGINEER and Bentley Systems' MicroStation (Figure 2.12) CAD packages are heavily dependent on the idea of designing with parametric features.

One of the most successful efforts at unifying geometric representation, visualization, and manufacturing has been the α_1 Project at the University of Utah [44, 192]. In α_1 ("alpha one"), geometry is represented and visualized using NURBS. Mechanical designs can be created in α_1 using a library of machining features—features that correspond directly to operations available in the α_1 manufacturing cell. For more complex shapes, the NURBS models can be used by α_1 to generate tool paths for machining the surfaces on a 6-axis

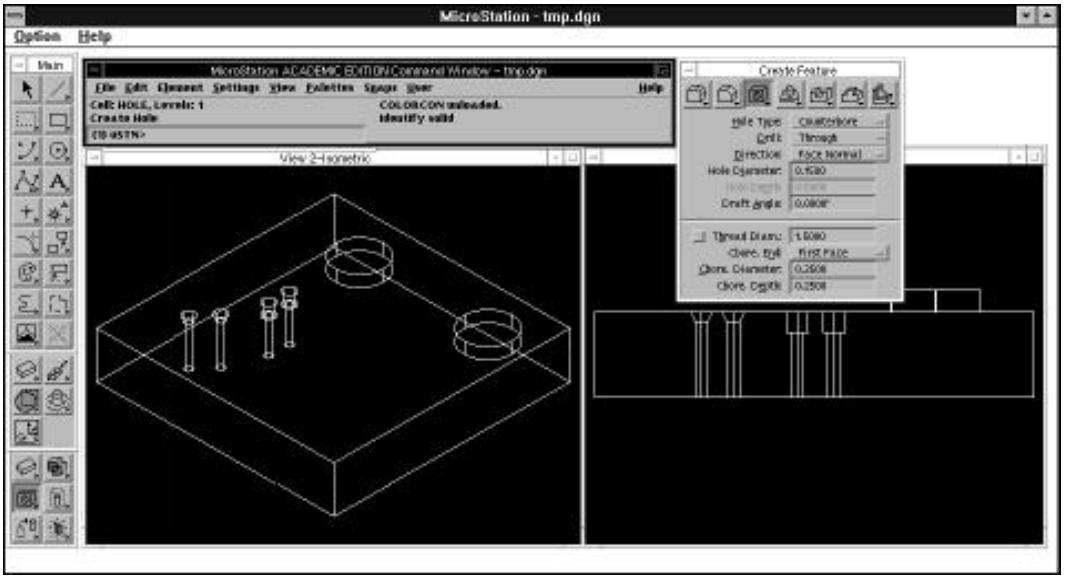


Figure 2.12: An example of a commercial feature-based CAD system: The MicroStation from Bentley Systems Incorporated. Note the feature palette at the upper right lists a number of parameters for holes. The part pictured has several design features: counter-sunk and counter-bore holes as well as two bosses.

machining center.

Traditionally, **feature-based design** was supposed to alleviate the need for feature recognition. However, feature-based design has not been able to overcome several significant inherent problems which it has alone.

1. With feature-based design, a designer models the design directly in terms of manufacturing features; hence in this scheme, design features are equivalent to manufacturing features. This eliminates the need to perform feature recognition because the design is, in some sense, defined in terms of a set of operations that may be used to manufacture it. But the features that the designer uses to define the final product may not represent the best way of creating the product.

2. In the majority of cases, the features that are most natural for use during the design phase are not manufacturing features. A designer working through a conceptual design and attempting to create detailed geometry does not think in terms of machining operations. Rather, design features are often better defined in terms of function, shape, and form. An additional problem, one that emerges strongly in the case of machined parts, is that design features might be additive. For example, a designer might add a rib to strengthen a part. The problem emerges that, while some design features might have some degree of correspondence with manufacturing operations, many will not. At some level, feature recognition will be required to translate the part information

(geometry, topology, and design features) into a description in terms of machining features.

It has also been pointed out that some of the most useful features to designers are those directly useful for performing analyses. Therefore, translation from a CAD model and/or design features to the features for analysis must occur at some point. For designs created using a feature-based design approach and represented with design features, there must exist some facility for translating these design features into feature viewpoints for use in the downstream applications [151].

3. For many manufacturing domains the feature-based description of a design is not unique [115, 96, 98, 95]. Selecting which interpretation is most suitable to drive manufacturing planning requires that the designer be an expert in the relevant domain.

4. As noted in the previous points, a design may need to be analyzed for a number of different manufacturing and product life-cycle considerations. Each of these different domains might require its own domain-specific set of features; hence the design must be interpreted differently for each life-cycle domain. This implies that, within an exclusively design-by-features system, the designer must assume the task of generating all of the relevant feature-based descriptions of the design—a task requiring substantial effort on the part of the designer and one that might introduce conflicts among various descriptions.

While feature-based design has proven quite useful as an effective means to allow the designer to work with functional features and create detailed designs more easily, the consensus appears to be that feature recognition will be required at some level. At the very minimum, feature recognition is perhaps the only reasonable way to generate multiple feature-based descriptions. It is also believed that feature-based design provides an opportunity to capture other important manufacturing information, such as the designer's intent and the design rationale [104, 172]. For a recent review of research in the area feature-based design, readers are referred to [162].

2.2.4 Feature Recognition

Creating a survey of the field of automated feature recognition presents a difficulty because attempts span a wide variety of applications. Research goals, application domain, and technique vary greatly over the works in the field and leave few bases for comparisons between methods. Motivations for feature recognition include classification of parts for group technology, generation of paths for numerically controlled machining, and creation of process plans for part manufacture.

We classify these approaches based on the computer science techniques they employ. This section presents an overview, by no means complete, of many existing approaches. For another summary of recent work readers are referred to [98].

2.2.4.1 Pattern Recognition

Extracting features from a 3D solid model can be viewed as a problem of pattern recognition. Syntactic pattern recognition [57] uses structural information to create a description or a classification of the artifact. This approach to pattern recognition has widespread use in computer vision (picture recognition, scene analysis, classification of pictorial patterns), speech recognition, natural language processing, and recognition of written characters.

A variation or extension of a context free grammar [82, 111] forms the core of a syntactic pattern recognition algorithm. In such a grammar, the **terminal symbols** usually represent a primitive element of the application domain. For automated feature recognition, a primitive can be an edge or a face—because these are among the fundamental building blocks of every feature. The grammar also contains **rules** that determine how these fundamental primitives may be combined. There may be a rule $S \longrightarrow \alpha$ that generates a specific type of feature or a rule $T \longrightarrow \gamma$ that generates a class of similar edge contours. The scene is then **parsed** much like a compiler parses a computer language [2]. During parsing the structural information embodied in the rules, such as the existence of a depression in the object, can be exploited. Utilizing these techniques, algorithms can recognize features and classify the shape of a solid. Most approaches based on pattern recognition techniques address only 2D shapes or 2D cross-sections of solids.

Ryszard Jakubowski [89, 90], using edges as pattern primitives, generated group technology part codes for mechanical parts based on their 2D cross-sections. This approach is concerned only with mechanical part classification and is described for a restricted set of $2\frac{1}{2}$-dimensional parts. Information is obtained by parsing the edge primitives in the silhouette of the part. This information and basic manufacturing data are used to categorize the part. In Jakubowski's work there is much grammatical formality and little mention of features. Among the practical limitations, for example, the approach addresses only $2\frac{1}{2}$-dimensional parts. Also, the primitives used in this study limit the complexity of a silhouette because there are only a finite number of 2D edges and rules in the grammar. These limitations and the fact that the method is presented as a formalism without demonstrated computational viability make this approach impractical for classification of complex parts and automated feature recognition.

In Srinivasan, Liu and Fu [187], an approach similar to Jakubowski's is investigated. The goals of the work are shape classification for group technology process selection and representation of the volume that must be machined to create the object. Grammars for generating shape families are presented. These grammars, like those in Jakubowski [89, 90], describe the outline of the part. This work built on Jakubowski's, but many of the same limitations still exist: the class of shapes is limited and it does not allow for the reasoning about geometry and topology necessary for automated feature recognition.

Henderson and Anderson [11] use a grammar to classify holes based on how they are manufactured. A 2D cross-section is obtained from a 3D CAD database. The grammar is then used to extract the holes from the cross-section, as illustrated in Figure 2.13. The grammar classifies the hole based on several hole families. The limitations in this approach are representative of the limits of all feature recognition systems based on syntactic pattern

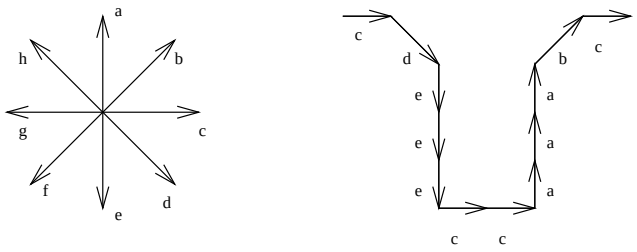


Figure 2.13: The pattern primitives and parse of a counter-sunk hole from Henderson and Anderson [11].

recognition techniques. The eight pattern primitives in the grammar are capable of representing only a small subset of the angles that could exist in a hole. In addition, there could be ambiguity in the 2D cross-section caused by features other than holes (this problem could be eliminated by assuming that a hole is the only type of feature). Ambiguity could also arise from interactions between holes—what if two perpendicular holes intersect? What if holes are not perpendicular to the surface of the object? These limitations do not diminish the significance of the work, which is its contribution to the ability to communicate with a geometric database on a feature level. However, this communication consists of a small set of basic questions about an extremely restricted set of objects.

In another related grammar-based effort, Kramer [105] presents a parser for converting a solid model representation of a design into a feature-based representation.

2.2.4.2 Graph-Based Methods

Graph-based algorithms have proven useful for extracting some classes of features. These methods fall into two categories: those based on graph search [39, 30] and those based on pattern matching [91, 141, 156]. A common difficulty for both categories of the graph-based approach is that the graph-based representations for solid models of parts are difficult to extend to the complex geometry and topology found in real industrial parts. Also methods based on pattern matching and finding subgraph isomorphisms (a problem known to be NP-hard) are prone to combinatorial difficulties.

Graph-based feature recognition systems represent the geometric and topological information about an object using graph structure—usually a structure obtained from (or embedded in) the data structure of the boundary representation of the object. Recognition of features can be defined as a search or parse of the graphical structure. An advantage of graph-based systems is the theoretical and algorithmic foundations of graph theory [75]. This affords the opportunity to exploit the mathematics of graph theory to define the problem and the opportunity to draw on the many algorithms that deal with searching, traversing, parsing, or matching graphs [3, 19, 145]. For a general reference on the combinatorial complexity of graph and grammatical approaches to feature recognition readers are referred to Peters [139].

Searching and Matching. In one of the most significant early works in feature recognition, Kyprianou [109] presented the first effort to automatically parse solid models of parts for group coding. His approach, as outlined in his doctoral dissertation [109], is often categorized as grammatical or syntactic pattern recognition; in actuality it was an early graph-based method to code parts for group technology classification. He developed an algorithm that identified basic types of protrusions and depressions in a CAD model and inferred from them a classification for the global shape of the object. The algorithm used a grammar based on adjacency relationships of faces, convexity and concavity of edges, and existence of edge loops to parse the object. Specific feature structures could be looked for during the parse and, based on the types of features found, a classification for the part could be generated.

Kyprianou's work did not include a specification for the set of objects it could classify and it is unclear what happens when the algorithm is presented with an object outside its scope. The method has difficulty parsing objects for which necessary structural information is lost due to interactions among feature types. Kyprianou's thesis was one of the first in the area of automated feature recognition and it has served as a fundamental reference for later works.

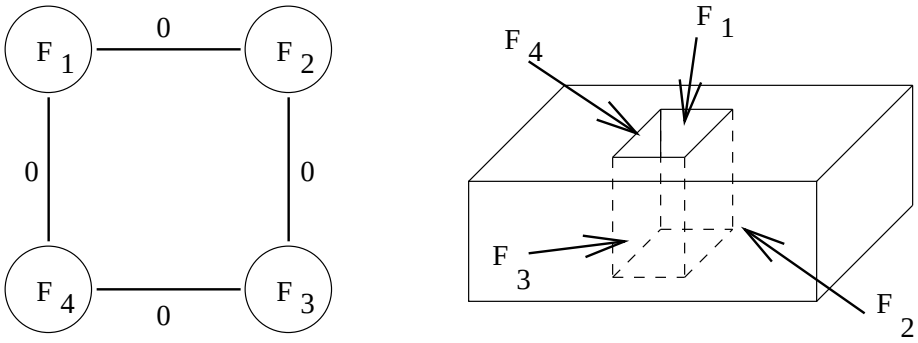


Figure 2.14: The attributed adjacency graph representation for a rectangular hole from Joshi and Chang [91].

Joshi and Chang [91] present a graphical structure called an **attributed adjacency graph** and, based on this data structure, the develop algorithms for feature recognition based on subgraph isomorphism algorithms to match feature patterns to patterns in the topology of polyhedral parts. Their goal is to improve machine understanding of design to facilitate the automation of process planning. The attributed adjacency graph is built from the information contained in the boundary representation of the solid model. Nodes in the graph represent faces of the object; arcs in the graph denote edges of the object. Each arc has an attribute indicating the concavity or convexity of the edge it represents. Figure 2.14 provides an example of the attributed adjacency graph representation of a rectangular hole.

An elegant aspect of this approach is that it clearly defines the types of 3D objects and features within its scope. This work has its limitations, however: it addressed only six

types of form feature and the object class is restricted to polyhedral objects. The feature recognition algorithm involves heuristic improvements to the NP-hard subgraph isomorphism problem. Attempts are made to handle feature interactions using rules for two specific types of interaction. These two types of interactions cover many, but not all, of the interactions possible for the six types of recognizable features. The drawbacks to this approach underscore the difficulty in developing an elegant and practical approach to automated feature recognition.

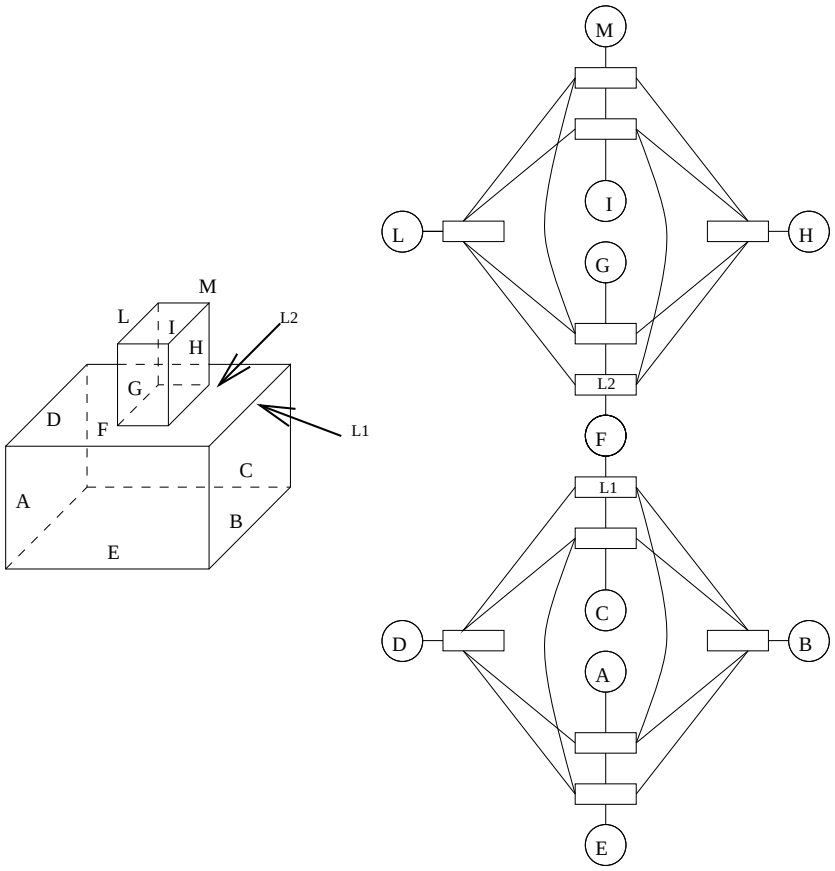


Figure 2.15: The generalized edge-face adjacency graph representation for an object from De Floriani [39].

De Floriani [39] introduces a graphical structure called a **generalized edge-face adjacency graph** (an example of which is shown in Figure 2.15) for recognition of features such as protrusions, depressions, through holes, and handles. In this scheme, the features and the object are represented using the generalized face-edge adjacency graph in such a way that the features form biconnected and triconnected graphs. Hence, feature extraction becomes a problem of searching a graph for the biconnected and triconnected components and classifying them as one of the feature types.

This approach uses graph theory and established, polynomial-time algorithms to produce a formal feature recognition system. The recognition algorithm does not have the complexity problems of the Joshi and Chang approach because the features are all defined as biconnected and triconnected components. Unfortunately, not all desired features can be described as biconnected or triconnected components in a generalized face-edge adjacency graph. The approach ignores most geometric information, placing little emphasis on shape and location of faces. In addition, it is unclear how this type of approach would handle shapes with nonplanar faces, nonperpendicular edges, and having complex feature interactions. The algorithm for recognizing the features defines the type of features that will be recognizable—those that form unique components in the generalized face-edge adjacency graph. Other work from the same research group [53, 40] uses an approach similar to this to generate a hierarchical feature structure to represent objects at higher levels of abstraction.

Sakurai and Gossard [161] developed a system for recognition of user-defined features based on a similar graph search strategy. An important contribution of this work is that it emphasizes the separation of the feature definitions from the feature recognition algorithm. Because features are often domain-specific, Sakurai and Gossard advocate a procedure to enable the user to specify those features that are important to the application. The features and the object are then represented with some variation of an attributed adjacency graph. The process of feature recognition is redefined to a graph search problem. This method is very similar to Joshi and Chang's [91] and shares many of its drawbacks. However, unlike Joshi and Chang, no attempt is made to characterize and deal with feature interactions. In fact, the approach characterizes features as collections of surfaces and then "fills in" the volumes when they are recognized. This act of filling in the feature volumes can make what remains of a valid object unrecognizable within the available feature set; since filling in one feature will inadvertently fill in part of any other feature that shares its volume. For example, if two features share a volume, "filling in" when finding the first of them may render the other feature unrecognizable.

Gavankar and Henderson [26, 60] explore graph-based pattern matching techniques to classify feature patterns based on geometric and topological information from the part. This approach operates solely on the topology of the solid model of the part and is limited in the kinds of features it can extract (i.e., only features with a single entrance face are considered) and in the fact that the general protrusion/depression information is not further classified into specific machining features.

Corney and Clark [30, 31, 34, 32, 33] have had success extending the capabilities of graph-based algorithms to more general $2\frac{1}{2}$-dimensional parts. In particular, they built boundary-representation-based recognition procedures to find DP-features (depressions and protrusions) from CAD models for machined parts (similar to the feature types of [109]). They point out that DP-features are basic elements in a number of feature hierarchies and taxonomies. They outline how procedures for DP-features can be used to find high-level shape information about a given part quickly and efficiently. They argue that this high-level DP-feature description can then be used for a variety of purposes, including designing more efficient algorithms for finding low-level features, such as features corresponding to

individual machining operations. Currently their approach only supports $2\frac{1}{2}$-dimensional polyhedral objects and does not evaluate whether or not the DP-features are accessible for machining.

Fields and Anderson [55] present an approach to feature recognition that overcomes some of the representation and efficiency problems common in previous work. Unlike pattern-based or decomposition-based recognition methodologies, they categorize sets of faces on the surface of the part into classes of general machining features: protrusions, depressions, and passages. The shapes within each class, while sharing many operational similarities, can vary in geometry and topology. For each of their feature classes, they present a linear-time algorithm.

Graph Grammars. Methods based on graph grammars have been used both to recognize features and to translate between different feature representations.

Chuang and Henderson [26] explored graph-based pattern matching techniques to classify feature patterns based on geometric and topological information from the part. In later work, Chuang and Henderson [27] were the first to note the need to address explicitly both computational complexity and decidability when defining the feature recognition problem. This paper formalized the problem of recognizing features (including compound features) by parsing a graph-based representation of a part using a web grammar.

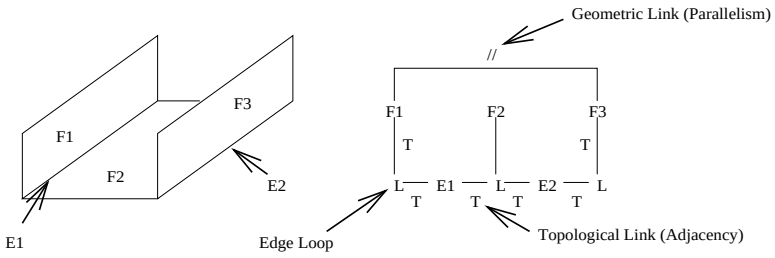


Figure 2.16: The representation of a slot using the augmented topology graph grammar from CMU [141, 156].

Efforts at Carnegie Mellon University by Finger et al. [141, 156] have employed graph grammars for finding features in models of injection molded parts. They define objects to be elements in a language generated by an **augmented topology graph grammar** and then use their graph grammar to parse a representation of an object. The advantage of their graphical structure is that it contains both geometric and topological information (unlike the structures of [39, 53, 91, 161]). This graphical structure, as shown in Figure 2.16, can also be used to define general features. A grammar can then be defined to describe the class of objects that can be generated with a specific set of features. To recognize the features, the grammar is used to parse the object. The grammar is used to produce the object in much the same way that a computer language compiler uses a grammar to determine if the program

it has been given is in a specific computer language [2]: as the object is parsed, information about the features can be produced. Extensions of these concepts to the problem translation between differing feature representations can be found in [151].

This approach has many attractive properties, among them the ability to define the needed features and the use of a formal graph grammar for describing a language of shapes. The use of a graph grammar to parse an object creates its own computational complexity, however, because of the pattern searching required for parsing. Feature interactions are not addressed and the scope of the work is limited to injection molding.

2.2.4.3 Volume Decomposition

Geometric algorithms for finding convex hulls have been employed to decompose polyhedral parts and identify form features. In an early effort along these lines, Woo [208] proposed a method for finding general depression and protrusion features on a part by decomposing the convex hull of the solid model. The approach had several problems, including the existence of pathological cases in which the procedure would not converge. The non-convergence of Woo's approach has been solved in recent work by Kim et al. [102, 103, 204, 136], whose system produces a decomposition of the convex hull of a part into general form features. Kim's approach uses convex volume decompositions to produce alternating sums of volumes and techniques for partitioning the solid to avoid non-convergence. Kim further improved the approach by performing additional mapping of the found volumes to machining features. Extension of this method from polyhedra to the more general surfaces required for realistic parts is currently under investigation [123].

In other work on volumetric approaches, Sakurai and Chin [159] propose an algorithm for recognizing general protrusions and cavities through "spatial decomposition and composition." The method generates alternative features and, although able to handle intersecting features, is computationally expensive and may generate very large numbers of alternative features. However, it is limited to a domain of polyhedral parts with orthogonal faces and it does not provide a means of grouping features into feature-based models.

More recently, Sakurai and his colleagues refined this cellular decomposition approach [158, 160, 159, 157, 38]. Such methods are computationally expensive, often producing a large number of cells with a large (often exponential) number of ways for them to be combined into features. They have several advantageous properties, however, including that they can be employed to produce alternative feature decompositions and, in the case of machining, that the cells can be used to generate many of the tool paths of interest in planning applications.

Efforts along similar lines have been undertaken at Arizona State University [169] and at the University of Texas at Austin [28]. The work at ASU uses a decomposition approach based on half-space partitioning. The primitive cells created by this process are then categorized using a degrees-of-freedom (DOF) analysis, wherein each cell is considered relative to the possible machining directions on a three-axis machining center. This DOF approach has several computational drawbacks, among them that it appears to be computationally intensive, as pointed out by Han and Requicha [74]. Further, is it unclear as to how these degrees of freedom, often presenting infinitely many selections for possible operation setup

directions, will be resolved into a finite number of tractable possibilities. For example, determining tool accessibility might require extensive boolean operations with the part (in order to determine interference) throughout many different possible degrees of freedom. The work at UT Austin formalizes a variation on the Sakurai approach to generate a decomposition in terms of maximal cells.

One limitation common to the decomposition approaches is that they appear to be exclusively applicable to material removal operations in the machining domain. It is unclear how such techniques can be adapted to deal with the features that occur in other manufacturing domains, such as assembly planning, sheet metal bending, casting, forging etc. A second limitation can be found in the complexity of the parts these approaches can handle: in all of the above cases, the domain is confined to polyhedra or planar and cylindrical surfaces. How these approaches can be modified to handle many real-world problems, where the parts might be manufactured with multiple processes (e.g., casting and then machining) and might be composed nearly entirely of curved surfaces is unclear. Thirdly, none of these approaches considers the constraints imposed on features by the cutting tools, in particular features may have unmachinable corners or might be inaccessible.

2.2.4.4 Knowledge-Based and Rule-Based

Expert systems are used to automate deductive reasoning tasks requiring expert knowledge. The goal is to encode the knowledge and experience of an expert into rules. Given a set of facts, the rules can be used to reason about the domain. A mechanical engineer must analyze a solid object to determine how to manufacture it; therefore, feature recognition can be viewed as the creation of an expert system to reason about the manufacturability of a solid object.

Henderson, in his 1984 Ph.D. thesis [77], created an expert system in Prolog to perform feature recognition. In this system, rules are used to define feature instances. For example, one possible definition for a **hole** is as "a cylindrical surface with an open top and open bottom." The feature rules become a part of an expert system that interacts with the CAD database. The process of feature recognition is performed by applying the rules to the database and letting Prolog's theorem prover determine feature instances. This approach was the first to employ an expert system to perform feature recognition.

An important aspect of Henderson's work is the attempt to create general rules. The idea is that the basic concept of "hole" remains unchanged despite the large, possibly infinite, variety of variations of specific instances. The system contains definitions for classes of $2\frac{1}{2}$-dimensional features including holes, slots, and pockets.

As the approach is detailed in the thesis, however, no attempt is made to deal with the feature interactions and the system is restricted to $2\frac{1}{2}$-dimensional parts. The Prolog rules defining the features rigidly specify each class of feature—hence each new feature type requires additional rules. Expert systems have proven effective tools for creating systems with large feature vocabularies. As these vocabularies grow, so does the time needed for Prolog to perform the search. For expert-system-based methods to deal with feature interactions, rules need to be given to specify **all** types of interactions—including interactions among feature

classes and interactions between specific instances. Expert systems perform well with limited sets of features and possible interactions but require large amounts of special-case information to be feasible for more realistic applications.

Dong [41, 42, 43] pursued the development of feature recognition by frame-based reasoning to facilitate computer-aided process planning from a CAD system. To address this problem, he presented a prototype frame-based knowledge representation scheme for features and parts, along with a rudimentary feature description language. Dong's approach included the ability to construct volumetric features from surface features and perform an analysis of tool accessibility. Using this knowledge representation, algorithms were developed to translate the solid model into the frame representation and recognize a set of eleven feature types from four feature classes. These eleven feature types are explicit manufacturing features that can be employed directly in the process planning system. The significance of this work lies not as much in its contributions as a feature recognition system, as it does in its recognizing of features adequate for interpretation with a process planner. The frame-based approach is novel; however, it remains unclear how feature interactions would affect the system's ability to find adequate feature information to facilitate process planning.

Marefat et al. [117] developed a formal feature recognition method based on Dempster-Schaffer theory and hypothesis-testing techniques. A key component of their work is an aggressive approach to handling feature interactions and intersections as well as proofs of the completeness of their feature generation hypotheses. Marefat's method builds on the representation scheme of [91] and uses a novel combination of expert system and hypothesis testing techniques to extract surface features from polyhedral objects. In arguing that the approach is **complete** over its class of polyhedral features, Marefat was the first to attempt to prove that an algorithm was capable of generating all features in a defined class that can be found from the geometry of a part.

In a recent follow-up effort to the work of Marefat, Trika et al. [196] extend and improve some of Marefat's original results and address completeness over a domain of iso-oriented polyhedral parts.

The most comprehensive approach to date for recognizing machining features and handling their interactions has been the OOFF system (Object-Oriented Feature Finder) of Vandenbrande and Requicha [200, 202, 201, 203]. Vandenbrande's thesis [201] advocates a sophisticated approach to feature recognition involving a variety of artificial intelligence and computational geometry techniques. Vandenbrande's work, using a knowledge-based approach like Dong's, provides a framework for recognizing a significant class of realistic machining features of interest for process planning via a frame-based reasoning system in combination with queries to a solid modeler. The goal of the work is to generate a feature description of an object that satisfies rigid machinability requirements.

Features are built by searching the boundary representation of the object for **hints**. A hint for a type of feature can be thought of as a byproduct of the feature being present in the object. For instance, a cylindrical surface can be a hint for the existence of a hole; a slot hint can be two parallel surfaces as shown in Figure 2.17. If the recognition algorithm finds a cylindrical surface, it may indicate the existence of a hole. The recognizer collects all the

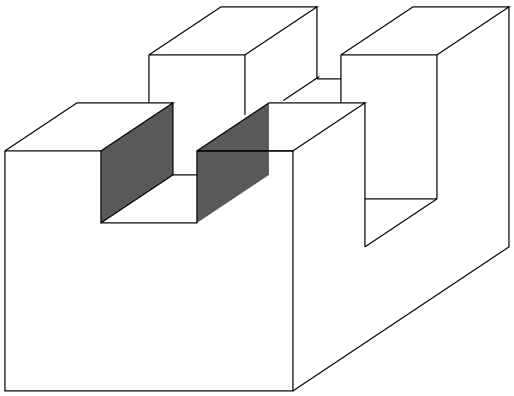


Figure 2.17: The highlighted parallel planar faces are a "hint" at the existence of a rectangular slot as described by Vandenbrande [201].

hints that exist in the object and categorizes them based on how promising they are.

The hints are extracted from the solid model and classified as to their potential for building a feature instance; unpromising hints are discarded. A frame-based reasoning system then acts on the hints and attempts to complete a feature frame including the information needed to construct geometrically maximal feature instances and build a set of features compatible with the hints. One of the fundamental contributions of Vandenbrande's work was a formal method for representing interactions among the features by calculating the "required" and "optional" volumes for each promising feature instance. Primitive features can be combined into composite features that have manufacturing significance.

This approach also allows the recognizer to deal with feature interactions in a more general manner because hints deal with the features on a more abstract level; thus interactions can be dealt with based on the hints. The hints, however, are actually special cases themselves: to add more features you need more hints and more rules for combining the hints. The approach based on a set of $2\frac{1}{2}$-dimensional features—no provision is made for the inclusion of other feature types. The approach also produces some alternate feature interpretations; however, the generation of alternatives is not well controlled nor is the class of alternatives produced by OOFF specified.

2.2.4.5 Hybrid Approaches

The recent work of Laakko and Mäntylä [110] couples feature-based design with feature recognition to provide for incremental feature recognition. This type of approach recognizes changes in the geometric model as new or modified features while preserving the existing feature information. The approach also provides for some forms of customizability by using of a feature-definition language to add new features into the system.

To address the combinatorial complexities of feature recognition for realistic artifacts,

recent work by Gadh and Prinz [58] describes techniques that abstract an approximation of the solid model and extract features from the approximation. They were the first to describe techniques for combating the combinatorial costs of handling complex and realistic industrial parts (i.e., those with thousands of topological entities). They point out that, in such cases, traditional knowledge-based, decomposition, and pattern-matching techniques are computationally impractical because the fundamental algorithms (i.e., forward chaining in a frame-based reasoning system or subgraph pattern matching) are inherently exponential.

Gadh and Prinz's method abstracts an approximation of the geometric and topological information in a solid model and finds shape features in the approximation. Their approach employs a differential depth filter to reduce the number of topological entities. A second pass maps the topological entities onto structures called "loops." In their work, features are defined using the high-level loops rather than as patterns in the boundary representation's geometry and topology. This approach significantly reduces the number of entities that need to be searched in order to build feature instances. While their of approach holds much promise for addressing combinatorial problems, they do not address how to extend the techniques to better handle interacting features and more non-linear (non-faceted) solid models.

2.2.4.6 Human Supervised/Assisted

When feature recognition has made it into commercial software tools it has been in the form of human supervised feature identification. The basic idea is to have the computer aid the human process planner (for example) in choosing the features. Once a suitable set of features has been selected by the user, the process planning tool can generate operation sequences. This approach is adopted by Van Houten in the ICEM-PART [198, 29] process planning system. PART supports more than 30 feature types, each classified as face sets or face patterns. PART first makes an attempt to automatically identify features in the CAD model using a graph-based method. The automatically generated feature set serves as a starting point for the user who then interacts with a feature editor to add, delete or modify features in this set prior to using it to generate a process plan.

2.2.4.7 Other Approaches

Other feature recognition work that does not fit directly into any of the above categories includes some recent work with neural networks. In particular, Peters [138] describes techniques for training neural networks to recognize feature classes that can be customized by the end user.

Prabhakar and Henderson [142] described the use of neural networks to recognize and classify features for a domain of polyhedral objects. A strength of this approach is that they exploit the trainability of a neural net to incorporate new feature types. Further, neural nets have been demonstrated to be effective in classifying patterns in domains where there is "noise." This noise is in the form of incomplete or missing feature data lost due to feature intersections and interactions.

The limitations of this connectionist approach include slow recognition speeds, difficulties in distinguishing between topologically similar features, and the fact that it has only been tested in a restricted domain.

2.3 Other Relevant Work

In this section we review several research areas in which feature recognition has significant impact and utility. Traditional feature recognition application areas such as part coding and process planning are no longer alone. Feature recognition and translation is becoming a crucial element for a variety of systems and applications—each with its own special set of requirements. We review some of these areas and their issues below.

2.3.1 Part Coding

As noted earlier in Section 2.2.4, a frequently addressed application for feature recognition has been the automated generation of group technology (GT) part codes. The basic idea group technology is to develop a systematic means of categorizing things that are "similar" from a manufacturing point of view. By analogy, group technology attempts to do for manufacturing what the biological classifications genus/species/phylum do for the categorization of living things. As noted in [25], group technology can be defined as the science of grouping similar problems with the objective that, by drawing on the knowledge of similarity, one can devise a single solution for sets of problems, thus saving time and effort.

Feature recognition has been applied to this problem with the idea that, if one can generate feature data about a part, one can classify it based on the manufacturing operations that will probably be used to produce the features. This was the objective of one of the first feature recognition methodologies, that of Kyprianou [109]. More recently Ames [10] describes a large-scale effort employing expert systems.

2.3.2 Process Planning

Automated process planning is one of the key elements required to seamlessly integrate CAD and manufacturing [9]. Numerous efforts have been made to automate process planning for machined components [35, 9, 20, 24, 130, 69, 206, 198]. Over the course of this research, two dominant approaches to computer-aided process planning (CAPP) have emerged: the **variant approach** and the **generative approach**.

In the variant approach, a process plan for a new part is obtained by retrieving an existing plan for a similar part from a corporate plan library. This existing plan is then modified for use with the new part. In the generative approach, new process plans are created from scratch using knowledge about the manufacturing processes and procedures that capture the planning task.

The process planning task involves a variety of related activities, including: selection of machining operations, cutting tools, tool assemblies, and machine tools; calculation of

setups and fixtures; selection of cutting parameters; sequencing of operations; part routing; and toolpath generation. As expected, all of these activities are intimately inter-related and, for the most part, cannot be executed independently. Several iterations are usually required to find an optimal process plan.

Identification of machining features from the CAD model of a part is one of the critical functions prior to planning. The set of recognized features completely determines the machining operations that get selected and, in turn, the set of plans that can be generated.

Although significant progress has been made in CAPP, there does not yet exist a completely automated planning system capable of handling moderately complex real-world parts. For more information on CAPP and a literature survey on plan generation, readers are referred to [9, 24, 206, 25].

2.3.3 Manufacturability Evaluation

Manufacturability evaluation and analysis involves the identification and diagnosis of manufacturing problems during the design phase. The fundamental goal of such systems is to interactively, while the designer creates a design, analyze its manufacturability with respect to the manufacturing domain at hand and to offer advice about potential ways of improving the design. A computer-aided manufacturability analysis tool can be used to assess the manufacturability of a design and calculate an estimate of the production cost and time. In doing so, problem design elements can be identified and suggestions can be made for changes to the design that improve its manufacturability.

In this application domain, feature recognition is used to construct feature instances from the data in the CAD model. This feature information is used to generate alternative interpretations of how the design might be manufactured; these are then used to estimate feasibility and machining cost and to generate feedback to the designer. In addition, the features constructed by a recognition system may be used to ensure that the available processes and facilities can satisfy the manufacturing constraints of the design. For example, there must be a set of features that can be used to generate a sequence of machining operations, meet the tolerance requirements of the design, and ensure that, at each step, the intermediate workpiece geometry is suitable for fixturing and setup.

Gupta et al. [67] describe a methodology for early evaluation of manufacturability for prismatic machining components. Their methodology identifies all machining operations that can be used to create a given design and, using those operations, generates different operation plans for machining the part. Each operation plan generated is examined for whether it can produce part's shape and tolerances. If the plan can create the part, a manufacturability rating (based on estimated machining time for the part) for the plan is calculated. If no operation plan can be found, then the given design is considered unmachinable; otherwise, the manufacturability rating for the design is the rating of the best operation plan.

For a survey of recent literature in the area of computer-aided manufacturability analysis for a wide variety of manufacturing domains, readers are referred to [66].

2.3.4 Automated Redesign

Automatic generation of suggestions for redesign goes hand in hand with manufacturability evaluation. For a manufacturability evaluation tool to be effective, it is not adequate for the tool to simply produce a rating component and a list of manufacturing problems. Many designers are not manufacturing specialists, and they might not be qualified to identify how to correct the problems noticed by a manufacturability evaluation system. This is particularly true in cases where the part is manufactured by multiple manufacturing methods or is produced by a supplier. To address such problems, it is emerging that manufacturability analysis systems will need the ability to generate redesign suggestions.

Traditionally, the redesign process has been manual task, however automated systems are beginning to appear. Most existing approaches for automated generation of redesign suggestions [167, 84, 86] propose only local geometric design changes (e.g., changes to the parameters of individual features), and some [76] present completely redesigned parts. Because of interactions among features, however, it is sometimes impossible to arrive at an improved design without carefully choosing a combination of modifications.

Based on the manufacturability analysis methods of Gupta et al. [67], Das et al. [37, 65] have created an approach for suggesting improvements to a given design to reduce the number of setups required to machine a part. This involves using different machining operations to satisfy the geometric constraints put on the part by the designer. These constraints are based on the functionality of the part. Different modifications are combined to arrive at redesign suggestions.

With automated redesign, feature recognition might be employed to generate a set of alternative features—features that might not occur in the best plan but which might provide some starting points for reasoning about how the design might be modified.

2.3.5 Design for "X"

The **Design for Manufacture and Assembly** (DFMA) methodology has been promoted as an effective way to identify and eliminate the manufacturing problems during the design stage. In DFMA, all of the design goals and manufacturing constraints are considered simultaneously and analyzed over the life cycle of a product. Manufacturing, however, consists of activities in many different domains, each having different characteristics. A design that is good for one domain may not be suitable in another. For example, a design that is easy to assemble may not be easy to machine. Consideration of these types of constraints starts at the conceptual design stage and continues through the embodiment and detailed design phases. Such analysis results can identify design elements that pose problems for manufacturing and suggest changes to address them. Figure 2.18 graphically presents the basic concepts behind DFMA.

There have been a variety of approaches to implementing the DFMA methodology, ranging from building interdepartmental design teams to equipping designers with manufacturability checklists. The pioneering work of Boothroyd and Dewhurst [15] in establishing design-for-assembly guidelines has led to the development of several automated advisory sys-

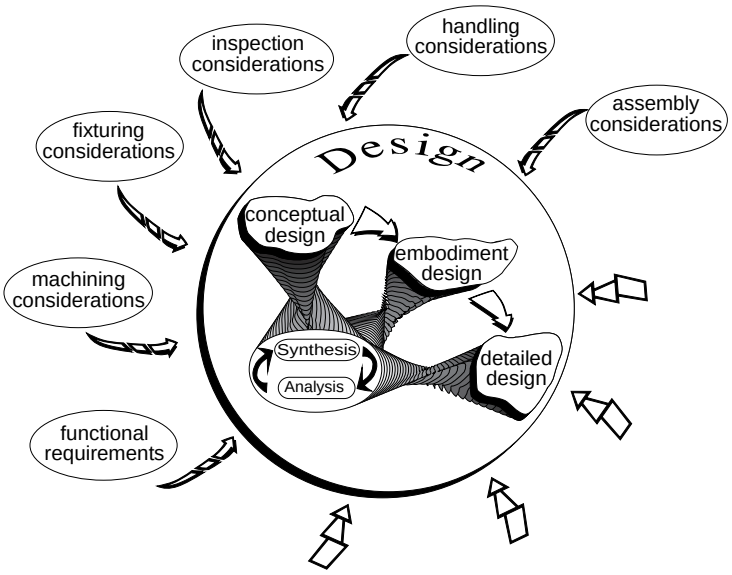


Figure 2.18: Design for Manufacture and Assembly: dealing with manufacturing life-cycle considerations during the design phase.

tems [88, 83]. Boothroyd [16] presents a review of DFMA methodologies in use at different companies.

With the advent and popularity of various CAD tools, there has been increasing interest in supporting DFMA through intelligent CAD systems: as a designer creates a design, a number of critiquing systems analyze its manufacturability with respect to different manufacturing domains (example manufacturing domains include machining, fixturing, assembly, and inspection) and offer advice about potential ways of improving the design.

Recently there has been a proliferation of tools for critiquing various aspects of a design (performance, manufacturability, assembly, maintenance, etc.). In many respects, DFMA is leading the way toward **design for "X,"** (DFX) where "X" might represent any potential product life-cycle consideration (i.e., any "-bility"). By taking into account other life-cycle considerations, more comprehensive analysis of a product can be performed [87, 85]. One major obstacle to DFMA/DFX is the immense complexity and difficulty involved in building a single system that can handle all manufacturing domains and all life-cycle considerations.

In this general DFX scenario, the role of feature recognition is to facilitate communication between the design and each manufacturing and life-cycle "view" of the design. For example, one feature recognition module might map the design to machining features to analyze manufacturability and perform process planning; a second feature recognition module might find surface features for fixturing and assembly planning; a third might communicate with simulation and analysis software to identify inefficiencies in the product's performance and how they relate to the shape of the design. In DFX, features come from a variety of manufacturing domains and from different stages in the manufacturing life cycle—this presents

great challenges to current CAD-based feature recognition technology.

2.4 Discussion: Research Issues for Feature Recognition

This section critically analyzes the previous research work in automated feature recognition. In particular, we focus on some of the key research issues that have not been adequately addressed in the current literature.

What is evident is that feature recognition has emerged as a critical research area in CAD/CAM integration and many different approaches have been developed over the last decade. Feature recognition is a problem of how to reason about shape. Based on the information in the existing literature, several crucial research issues are evident. The items on this list are highly inter-related; in addition this list is by no means exhaustive. It does, however, contain the points most relevant to the work in this thesis:

Developing a problem specification. One may argue that "producing a feature description of an object for manufacturing" serves as an adequate definition of the problem; however, as noted there is currently no clear definition of "manufacturing feature" [171]. Theoretically, there are infinitely many and in the absence of some formalization for the feature recognition problem it is difficult to characterize what systems actually do. Without a precise statement of the problem it is impossible to judge the effectiveness of any algorithm. This absence has tended to create ad hoc solutions with algorithms that are collections of special cases designed to deal with each feature and each interaction in isolated application domains.

Feature interactions. When features intersect with each other, this changes their topology and geometry in ways that can be difficult to take into account. Hence, it is often unclear what specific classes of parts and feature interactions can be handled by various existing approaches. The ability to handle interacting features has become an informal benchmark for feature recognition systems and has been the focus of numerous research efforts, among them [41, 58, 91, 117, 200].

What has emerged are several distinct types of feature interaction problems. One type is interaction during feature recognition, where an instance of one feature removes information require for the recognition of another. A second type of feature interaction is dependent on the manufacturing process. For example, in machining quite often operations interact—i.e., the presence of a certain operation affects the manufacturability of some other operations. Exhaustively enumerating all possible interactions can result in a large number of rules and make rule-based approaches unattractive. The part in Figure 1.3 (a) presents an example containing interactions at both levels. Figure 1.3 (b) shows one feature instance that should be found to perform effective planning; however, it is unclear how to generate this volume from the information in the part.

It has proven difficult to categorize and control which interactions are handled [200] and how; most current approaches rely on simple case-by-case or implementation-specific heuristics. A simpler and more general approach is needed.

Finding alternative feature interpretations. For machined parts, quite often a part has more than one valid interpretation. In most downstream applications it is important to be able to consider all feasible feature interpretations. This requires that alternative interpretations be generated as different collections of features. Most earlier research in feature recognition and process planning focused on generation of a single interpretation. Most existing systems focus on generating the optimum plan for individual features and many have limited capability to identify and account for feature interactions.

Recently, generation of alternative interpretations has received a great deal of research attention:

- Mäntylä et al. [115] proposed the concept of **feature relaxation** to support the idea of design by least commitment. The feature relaxation groups are pairs of geometric features that are interchangeable.

- Karinthi [96, 98, 95] performed the first systematic work on the generation of alternative interpretations for an object as different collections of volumetric features. They presented a means of computing the alternate interpretations of parts using an algebra to operate on the features.

- The AMPS process planning system [24] uses **feature refinement** heuristics to combine features into more complex configurations, or to divide features into multiple primitive features. Since the techniques are based on heuristics, it is not entirely clear when (and which) alternative interpretations will be produced.

- In feature recognition work by Sakurai [160], the volume to be machined is decomposed into cells. Exhaustively, each combination of cells is then matched against user-defined feature templates. While the method is capable of generating all alternative feature interpretations composed of the primitive cells, it does so at a large combinatorial cost.

- Waco and Kim [204] have extended convex decomposition techniques to produce alternative decompositions of the removal volume through aggregating and growing form feature primitives.

- Vandenbrande and Requicha's OOFF System [200] produces alternative features in certain cases—however, there is no specific definition for this class of alternatives. OOFF's class of alternatives is dependent on the process planning heuristics that are used to make decisions about which features to keep and which to discard as unpromising.

What has emerged is that there are two distinct ways in which feature recognition can be used to generate alternative feature-based models (FBMs) from a single CAD model.

1. **Generating FBMs Directly.** In this approach, feature-based models are generated "on the fly," as the feature instances are recognized. These approaches typically produce a single feature-based model for the given part. In this approach, whenever alternatives are encountered, a decision is made "on the fly" using a greedy heuristic to select the most promising feature or to discard others. Such greedy heuristics consider only the current feature in relation to the part (and sometimes the stock) and those features found up to that point in the recognition process. In this way, features are discarded based only on partial information and a potentially useful feature-based model could be eliminated from consideration.

   This approach has several drawbacks. First, until we have information about all of the other features that might be in the feature-based model, applying a greedy heuristic to build the model on a "best-fit" basis may require extensive backtracking to produce optimal results. Second, including domain-specific evaluation criteria as part of feature recognition introduces many difficulties. Hence, this approach is not appropriate for complex parts with a large number of alternative feature-based models. This is addressed in more detail in Chapter 4.1.

2. **Generating feature-based models from a Feature Set.** In this approach, two steps are used to generate feature-based models:

   (a) **Recognize a set of alternative features.** From the given part, first recognize a set of alternative features. Note that, at this level, all the features that appear promising are retained in this set of features.

   (b) **Generating and evaluating alternative feature-based models.** Once we have recognized a set of alternative features, we can generate feature-based models from this set. Intuitively, one can think of the set of alternative features found through feature recognition as vectors forming a basis for the "feature space" of the given part. Knowing a good set of spanning features allows us to better define upper and lower bounds so that the evaluation functions can navigate efficiently through the space of feature-based representations.

For parts with many different feature-based models this latter approach appears more promising. Previous research, however, has nearly exclusively addressed the former approach. In addition, it is clear that feature recognition shares an intimate relationship with process planning even though most systems continue to work with purely geometric information. Whenever alternative interpretations (alternative potential plans) are present, in the absence of full information about the complete set of alternative features, arbitrary choices are made among various possible alternatives.

CAD/CAM integration. As was illustrated in Figure 1.1, feature recognition is the interface between design and downstream applications. In order to perform reliable planning, manufacturability analysis, redesign, or DFX, feature recognition must be capable of generating all promising interpretations for the design. In certain cases, the set of valid feature

instances could be infinite or, at the very least, few of these valid feature instances will make practical manufacturing sense. In either case, there might be a very large number of interpretations, making it infeasible to generate all possible plans.

For the approach based on a feature set to work effectively, we believe there needs to be an improved definition for which features and feature-based alternatives should be recognized—we must decide which instances to recognize. Whether or not a system produces correct results will depend on the set of features used as a target in Step 1 and on knowledge that the system is complete (as described below) over that set of features.

Further development of trace-based feature recognition. The work of Marefat and Kashyap [117] presented an early trace-based technique where information from the solid model is used to generate hypotheses about the existence of features. These hypotheses are tested to see if they give rise to valid feature instances.

Vandenbrande and Requicha [200] were the first to formalize trace-based (or hint-based) techniques for constructing features from information in a solid model. In the work of Vandenbrande, the traces are used to fill "feature frames" in a frame-based reasoning system. After filling frames with the trace information in the part, the system classifies the partial frames and attempts to complete the frame information for those that appear promising, using a variety of geometric reasoning and computational geometry techniques.

At USC, recent work by Han and Requicha [73] has enhanced Vandenbrande's work and integrated it with feature-based design through an incremental approach. At Purdue, Trika has extended some of the results of Marefat. Aside from these two efforts, however, there has been no additional activity in the area of trace-based feature recognition to improve its underlying mathematical basis and enhance its ability to scale to real world problems.

Complexity. Computational complexity is a problematic issue for many approaches to feature recognition. As with problems in computational intelligence, the computer time grows tremendously when dealing with realistic situations [58]. The expense required to maintain the rules or perform searches and geometric computations can be difficult to manage. Because these will eventually be desirable to represent, it is important to attempt to measure computational complexity.

Existing methods make mention of complexity but rarely attempt to classify it. As pointed out by Peters [139], both grammatical methods and some graph-based approaches are prone to combinatorial difficulties. Some of the approaches that have incorporated complexity measures [39, 55] are often representationally limited, covering a restricted domain of parts (e.g., polyhedra) and/or feature types (e.g., not manufacturing process features).

For those approaches which employ expert systems and knowledge-based reasoning [41, 91, 117, 200], the inherent exponential nature of automated reasoning algorithms might impede their ability to scale to more complex problems. Further complicating matters is the fact that some approaches to feature recognition perform process planning "on the fly" as features are recognized. In the presence of multiple alternatives, performing elaborate plan optimization is computationally expensive and may result in time consuming analysis. Since

design analysis is best performed interactively, response time is a significant concern. Thus, for automated feature recognition, approaches should focus on quickly and efficiently producing a useful feature set, rather than on performing elaborate feature-by-feature optimization.

Handling real-world artifacts. Many approaches have proven difficult to extend to the more complex surfaces and features found in realistic manufacturing problems. For example, graph-based approaches [39, 91] began appearing in great number nearly a decade ago, yet they continue to prove difficult to extend more complex feature types. Similarly, the feature instances encoded with rule-based and knowledge-based systems are often only those for simple isolated feature instances or those with easily described interactions. Many approaches are still struggling to come to grips with parts that contain non-planar faces.

In reality, commercial CAD and solid modeling systems have made huge advances in representational ability. Existing feature recognition work is still mainly concentrated on mid-1980's solid modeling representations (polyhedra) and only on the most rudimentary test parts (e.g., brackets with only a handful of isolated features).

In the future, fast and effective feature recognition will be required for parts comprised of thousands of geometric and topological entities, hundreds of feature instances, and numerous feature types. Current technologies based on form features may not prove to be highly useful when faced with industry's need to have information generated about features that directly describe manufacturing processes. As more real-world shapes are incorporated, computational complexity (as described above) will become increasingly problematic.

Completeness. Informally, when one speaks of the **completeness** of a feature recognition algorithm, one is referring to the algorithm's ability to produce all features instances from some well-defined class of features and parts. Discussion of completeness was initiated with the work of Marefat [117]

From a devil's advocate point of view, one might claim that all feature recognition systems are complete over something; i.e., they are complete over the set of features they can handle. Unfortunately, such recursive definitions yield little information. What is evident is that a main purpose of feature recognition (perhaps **the** main purpose) is to generate alternative feature-based descriptions with which to reason about a design. In this context, completeness precisely categorizes which features and feature-based models will be generated—and hence which types of analyses using the features can be answered. Without some notion of completeness, feature-based models are produced on an ad hoc basis.

What is needed is a means of illustrating that completeness can be attained over a class of features that is useful for reasoning about manufacturing. Further, we would like to see that completeness can be defined independent of the algorithm that recognizes the features; i.e., first define the set of features of interest and then build the algorithm to recognize them. This type of approach would stand in contrast to much existing work, which often defines the feature types as by-products of what the algorithm is capable of generating (e.g., in using a connected component algorithm, one is limited to features expressible as connected components).

We anticipate that an analysis of completeness can help to better define which aspects of the feature recognition problem can be done automatically and which are best left to human supervision.

2.5 Summary

While automated feature recognition has emerged as a critical technology, many approaches that have been developed lack a consistent formalization of the problem—thus their overall utility has been difficult to evaluate. Further, many approaches employ techniques that are inherently limiting; either representationally or in terms of the computational complexity of the reasoning and recognition algorithms. Few approaches have demonstrated the ability to scale to real world artifacts and even fewer have presented concise mathematical means with which to specify the domain of features, alternatives, and parts that they are capable of handling.

Chapter 3

Developing Feature Definitions

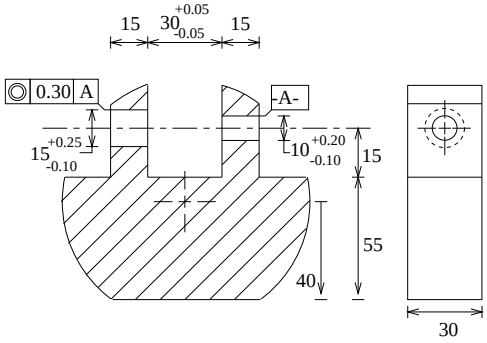
This chapter introduces a class of machining features and presents definitions for feature-based models, primary features and well-behaved features.

3.1 Machined Parts and Machining Features

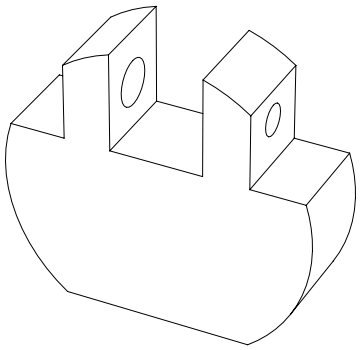
A common class of solids are those described by r-sets with manifold boundaries [116, 149]. In the context of this thesis, a **solid** is an r-set whose boundary representation is a manifold consisting of analytic surfaces (e.g., planar, elliptical, toroidal, conical and spherical surfaces). The **boundary** of a solid S, denoted $b(S)$, is the set of 2D surfaces contained in the boundary representation of S. A **design** is specified in terms of a solid model and associated design attributes (such as tolerances, design features, and geometric and topological specifications) that are to be realized through machining operations. In this thesis, we will focus on the geometric and topological design attributes.

A machined part, P, is a solid object represented by a solid model of the part design to be produced by a finite set of machining operations. For example, Figure 3.1(a) shows a design for a simple bracket and Figure 3.1(b) shows a CAD model of the bracket design. Machining operations remove material from a **workpiece** to create the design attributes of the part. The **initial workpiece** or **stock**, S, is the solid object of raw material to be acted upon by a set of machining operations generating features, (i.e., $P \subseteq S$ as shown in Figure 3.1(c)). As illustrated in Figure 3.1(d), the total removal volume is referred to as the **delta volume** (Δ), and it is the regularized difference [81] of the initial workpiece and the design: $\Delta = S -^* P$. Another design of a machined component is given in a later chapter (see Figure 8.7).

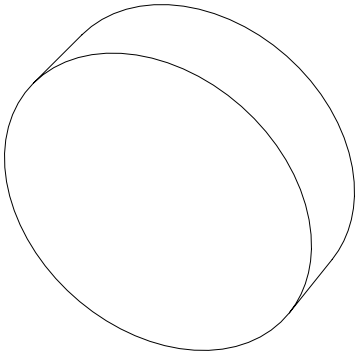
In this thesis a **machining feature**, M, is a parameterized volumetric template that represents the solid volume swept by a rotating **cutting tool**, such as those shown in Figure 3.2, during a machining operation. An instance of a machining feature, f, is created by a specific machining operation with a single cutting tool in one tool setup. To perform a machining operation, one sweeps the tool along some trajectory using a machining center (such as the one shown in Figure 1.2). Only a portion of this swept volume corresponds to the volume of material that can be removed by the machining feature—the portion swept out



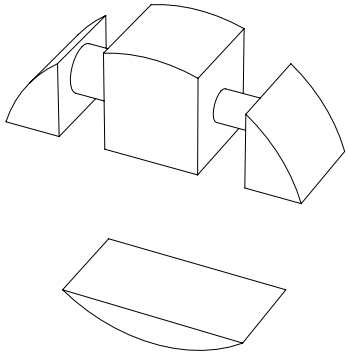
(a): design of a simple bracket



(b): part (after machining)

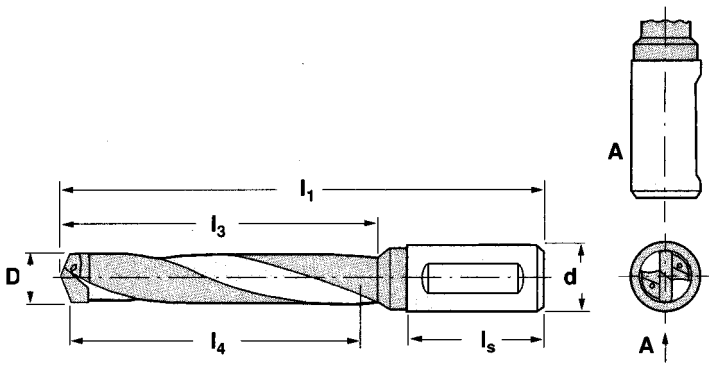


(c): stock (before machining)

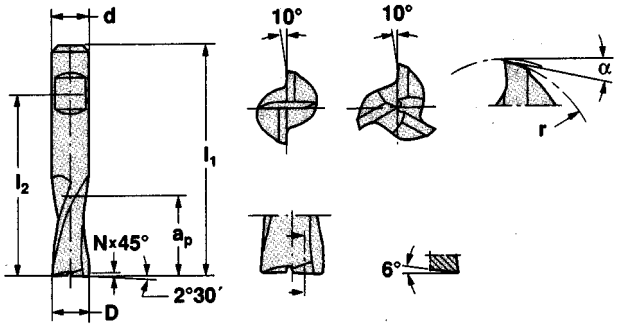


(d): the delta volume

Figure 3.1: An example part and stock.



(a): drilling tool



(b): end-milling tool

Figure 3.2: An illustration of typical cutting tools as found in a tool catalog [164, 165]. The variables refer to parameters in the specification for the tool which appears elsewhere in the catalog.

by the cutting surfaces of the rotating cutting tool. This volume is called **removal volume** of feature f and is denoted $\text{rem}(f)$. The volume swept out by the non-cutting portion of the tool is referred to as the **accessibility volume**, $\text{acc}(f)$. The volumetric intersection of the feature's removal volume with that of the workpiece is the **effective removal volume**, $\text{eff}(f) = S \cap^* \text{rem}(f)$. Feature accessibility and inaccessibility is discussed in greater detail in Chapter 7.

Feature Validity. In machining there are many conditions under which a feature is invalid. Many of these conditions can only be tested during the generation of a machining operation plan. However, there are some basic criteria that invalidate features regardless of their role in a particular operation plan. An instance f of a machining feature is **valid** if there exists at least one correct machining operation plan that is realizable with available manufacturing resources and that includes f; otherwise f is **invalid**. In particular, for feature recognition, we will say that a machining feature is **valid** for a given part P if:

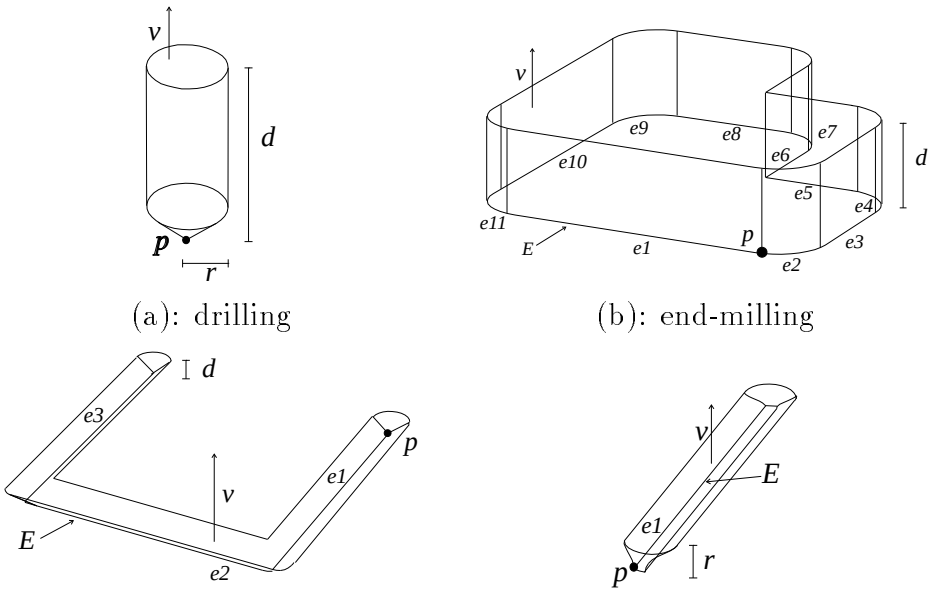


Figure 3.3: Parameterized feature instances.

1. f creates some portion of the boundary of P (i.e., $(b(\text{rem}(f))) \cap^* b(P) \neq \emptyset$)[1];

2. f does not interfere with P (i.e., $(\text{rem}(f) \bigcup \text{acc}(f)) \cap^* P = \emptyset$);

The set of all valid feature instances is called the **valid feature set**, $\mathcal{V}$. Testing of these feature validity conditions is addressed in more detail in Chapter 7.

Machining features are referred to in terms of the operations used to create them. For each feature f in $\mathcal{V}$, type(f) is f's feature type. For example, we say that the hole h in Figure 3.3(a) is an instance of a **drilling feature**. The pocket p in Figure 3.3(b) is an an instance of an **end-milling feature** and is characterized by the edge profile bounding the area swept by the milling tool.

In illustrations and figures of features in this thesis we adopt the conventions used by Gupta [70]. Features are drawn as their total removal volumes, i.e., as they are swept by the cutting portion of the cutting tool.

3.1.1 Definition of a Machining Feature Class

In a machining operation, a cutting tool is swept along a trajectory and material is removed by the motion of the tool relative to the current workpiece. The volume resulting from a machining operation is called a machining feature. A machining feature can be manufactured

[1]Note that $b(\text{rem}(f))$ and $b(P)$ are both 2D entities; hence their regularized intersection will also be a uniformly 2D entity.

with a single machining operation made on one machine setup. Each has has a single approach direction (or orientation) for the tool, represented by a unit vector, $\vec{v}$.

Features are parameterized solids that correspond to various types of machining operations on a 3-axis vertical machining center. The features described in this thesis are based on Kramer's MRSEV Feature Library, in particular hole, pocket, and edge-cut feature types. For more information, including the full mathematical and EXPRESS language definitions for the MRSEV features interested readers are referred to [108, 107].

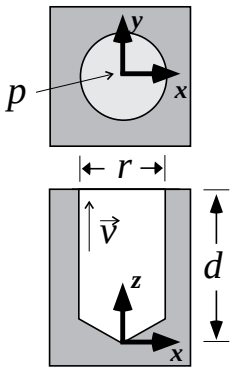
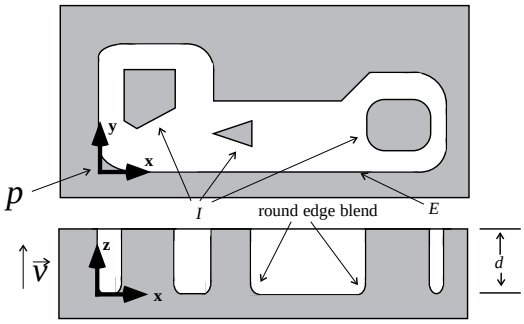
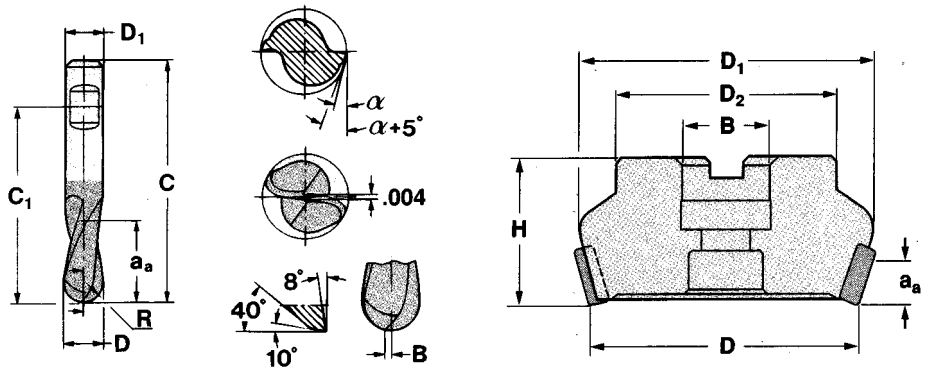
Drilling Feature		**Milling Feature**	
		location	p
location	p	**orientation**	$\vec{v}$
radius	r	**depth**	d
orientation	$\vec{v}$	**edge profile**	E
depth	d	**bottom blend**	b_{type}, b_d
		island edge profiles	I
			

Figure 3.4: Classes of machining features: drilling and milling.

1. In order to create a **drilling feature**, we will sweep a drilling tool (such as the one shown in Figure 3.2(a)) of radius r for a distance d along the trajectory represented by its **orientation vector** $\vec{v}$, ending at a **datum point** p, as shown in Figure 3.4. Thus, the volume describing the drilling feature can be modeled as a parameterized volume, as shown in Figure 3.3(a), formed by the regularized union of a cylinder of radius r with a cone that represents the conical tip of the drilling tool. The conical tip has a tip angle that describes the shape of the surface of the drilling tool end. We assume there exists a finite number of possible tip angles based on the available drilling tools; for simplicity all of the examples of drilling features shown in this thesis have tip angles of 120, 90, or 60 degrees.



(a): ball end-mill (b): face mill

Figure 3.5: Two other kinds of cutting tools for milling from tool catalogs: (a) a ball end-mill that could be used to create milling features with round bottom blends [48]; and (b) a face-milling tool [164].

2. To create a **milling feature**, we will sweep a milling tool (such as those shown in Figures 3.2(b), 3.5(a) and (b)) whose orientation vector is $\vec{v}$, starting at a datum point p, and moving through a swept volume of depth d whose cross-sectional area is bounded by an **edge profile** [108] $E = \{e_1, e_2, \ldots e_n\}$, as shown in Figure 3.4. An **edge profile** is an ordered collection of co-planar edges that, when joined end-to-end, form a closed, non-self-intersecting, continuous curve. Thus, the volume describing the milling feature can be modeled as a parameterized volume, as shown in Figure 3.3(b).

   In practice, it is impossible for a rotating cutting tool to sweep out a volume with convex corners. The parameters above describe the profile of the effective removal volume (eff(f)) for a milling feature; hence it is not required that all of the corners be non-convex or round. As a result, the effective removal volume on its own does not always correspond **directly** to the removal volume created by a milling operation. **Profile offsetting** (as described in Section 7.4) is preformed to deduce the shape of the removal volume rem(f) as swept by the tool from eff(f). This modifies the effective removal volume so that it more directly corresponds to the milling operation.

   The parameters for milling features also include:

   - Within the area bounded by the edge profile E there may be a finite set I of zero or more **islands**. Each island i in I is bounded by its own edge profile E_i.
   - Depending on the shape of the milling tool, there may be a transition surface between the bottom face of the milling feature and its side and island faces. There are three possibilities for the shape of this transition surface, as shown in Figure 3.7. To represent which of these shapes a milling feature has, we will associate an optional bottom blend type b_{type} and dimension b_d with the feature.

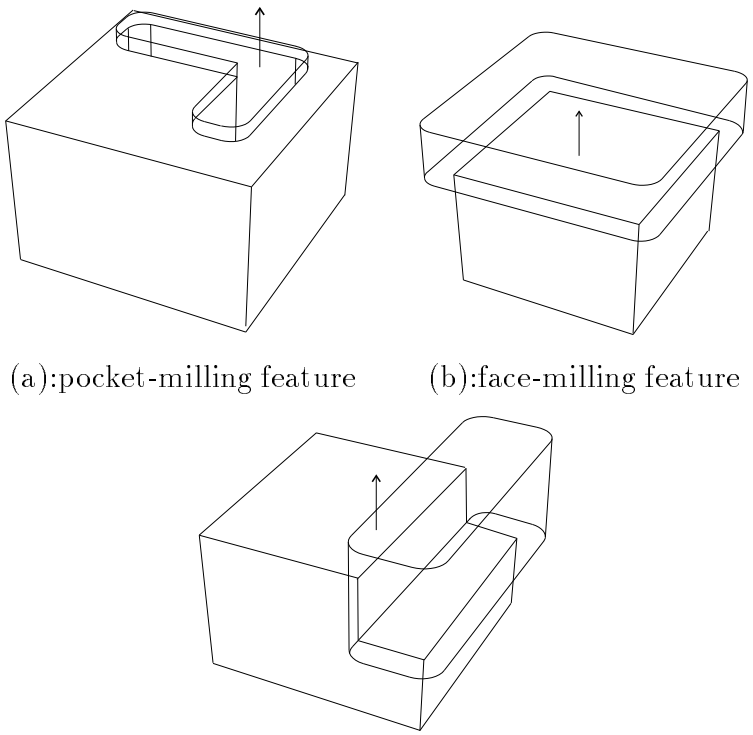
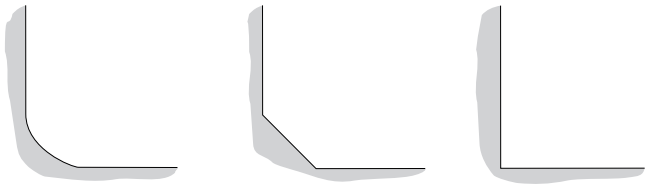


Figure 3.6: Subclasses of milling features based on the type of edges in their edge profiles. The corners of the features are curved to take into account the radius of the cutting tool used to machine them. Arrows indicate feature orientation.



(a): round blend (b): flat blend (c): no blend

Figure 3.7: Types of bottom blends: transition surfaces between side and bottom faces of milling features.

The shape and dimensions of blend surfaces are highly dependent on the shape and dimensions of individual cutting tools. This topic is discussed in more detail in Chapter 7.

The edges in the profile of a milling feature can be classified as **open** or **closed**. Suppose f is a milling feature with an edge profile E and e is an edge in the profile. Then each edge e of E is a subset of the boundary of some side face s of milling feature f, where s is the face formed by sweeping e along the orientation vector of the profile. We classify e is as **closed** edge if the interior of s intersects with the part boundary or if e is a concave edge on the part P. Otherwise, e is an **open** edge. Figure 3.8, from Gupta [70], shows three examples of milling profiles along with the edge classifications of each profile.

Intuitively, open and closed edges attempt to categorize the nature of machining a milling profile, during which the cutting tool cannot cross any closed edge. Crossing a closed edge implies that the tool is gouging into the final part shape. Crossing an open edge, on the other hand, is usually permissible and amounts to tool motion outside the workpiece.

In this thesis we adopt the convention used by Gupta [70] with regard to listing edges in an edge-loop: edges in an outer edge-loop are listed clockwise and the edges in any inner edge-loops are listed counter-clockwise. Hence, if traveling along the edges of any edge-loops in a milling profile, the material removed during the milling operation will be on the right side of the edges.

Milling features can be partitioned into three subclasses, depending on which edges of E are open and which edges of E are closed:

(a) Figure 3.6(a) shows an example of a **pocket-milling feature**. For this subclass of milling feature each edge in the edge profile E is closed.

(b) Figure 3.6(b) shows an example of a **face-milling feature**. Face-milling features are often machined with special face-milling cutting tools, such as the one pictured in Figure 3.5. For this subclass of milling feature there are no islands or bottom blends and all of the edges in the edge profile E are open.

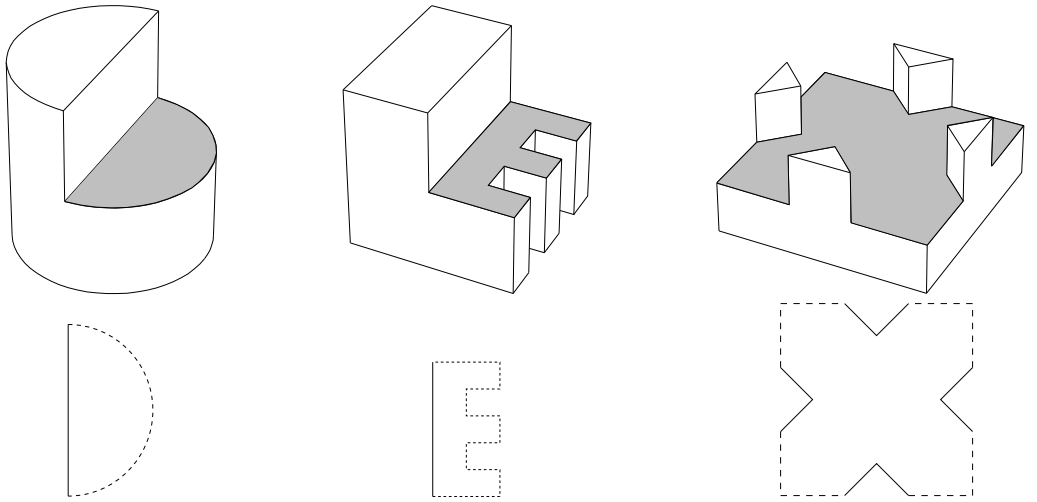


Figure 3.8: Examples of edge classification from [70]. Closed edges are drawn with solid lines; open edges are drawn with dashed lines.

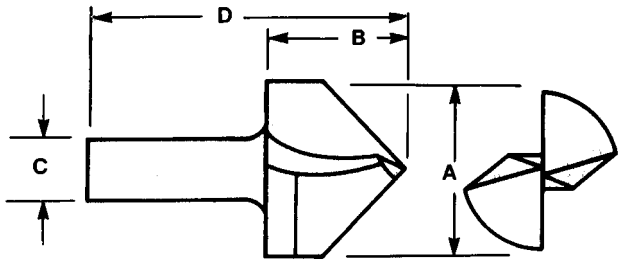
(c) Figure 3.6(c) shows an example of a **step-milling feature**. For this subclass of milling feature at least one, but not all, of the edges in E are open.

3. To create a **chamfering feature** we will sweep a chamfering tool such as the one shown in Figure 3.9(a) with orientation vector $\vec{v}$. The sweep will start at a datum point p and move at a depth d along a trajectory described by an edge profile $E = \{e_1, e_2, \ldots e_n\}$ of straight and elliptical edges, as depicted in Figure 3.10. Thus, the volume describing the chamfering feature can be modeled as a parameterized volume, as shown in Figure 3.3(c). The parameters for this volume also include the cutting tool's tip angle a[2] and an optional end radius r_e, to denote the conical surface that might be left at the start and end of a chamfer.

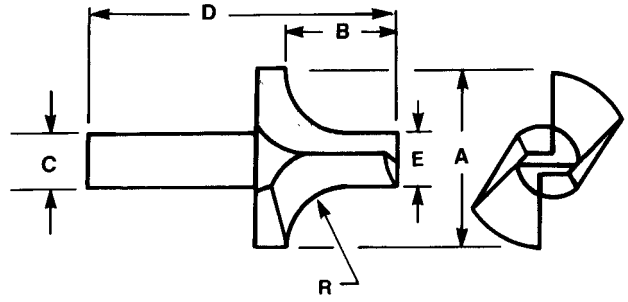
4. To create a **filleting feature** we will sweep a filleting tool such as the one shown in Figure 3.9(b) with orientation vector $\vec{v}$ and radius r. The sweep will start at a datum point p, and move along a trajectory described by an edge profile $E = \{e_1, e_2, \ldots e_n\}$ of straight and elliptical edges, as depicted in Figure 3.10. Thus, the volume describing the filleting feature can be modeled as a parameterized volume, as shown in Figure 3.3(d). Note that each edge e in the edge profile E bounds a curved surface of radius r that is tangent to the orientation vector $\vec{v}$ at the edge e.

   The parameters for this volume also include an optional end radius r_e, to denote the toroidal surface that might be left at the start and end of a fillet. We will assume that there is a finite set of available tools, each with fixed radii.

[2] We assume that there are one or more tools available with a 45 degree tip angle. The bounds on tooling parameters is discussed in Chapter 7.



(a): chamfering tool



(b): filleting tool

Figure 3.9: Other cutting tools for chamfering and filleting [48].

For drilling and milling features it is possible for a given feature instance to extend completely through the stock material. In this thesis, these feature instances will be called **through features.** An example of a through feature (a through hole) can be found in Figure 3.1(a). Features that do not extend through the part (e.g., those with bottom or ending surfaces) are called **blind features**.

In practice, chamfering and filleting features occur in the final stages of machining. One application of these types of features is to make minor modifications to the part surfaces to facilitate assembly operations. The edge profile for chamfering and filleting features is slightly different than that for end-milling features. Whereas for end-milling features the tool sweeps out the entire 2D cross-section inside the profile, for chamfering and filleting features the profile defines the path of the tool along the edges of the workpiece that are being modified. An example of a part with a chamfered surface is shown in Figure 3.11.

Feature Instances. The machining features defined above are parameterized solids. A specific instance is defined by assigning a specific attribute values. For example, suppose we choose the following attribute values for a drilling feature:

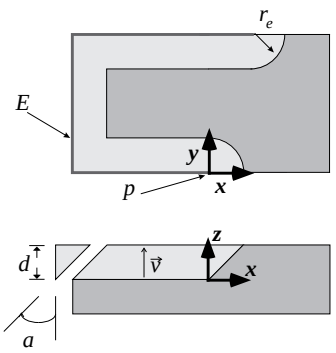
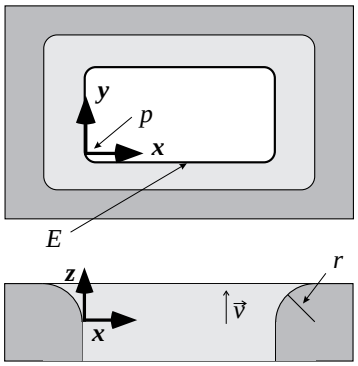
Chamfering Feature	Filleting Feature
location p **orientation** $\vec{v}$ **edge profile** E **depth** d **angle** a **end radius** r_e	**location** p **orientation** $\vec{v}$ **radius** r **edge profile** E **end radius** r_e
	

Figure 3.10: Classes of machining features: chamfering and filleting.

$$
\begin{array}{lcl}
\textbf{location} & = & (16, 10, 4); \\
\textbf{orientation} & = & (-1, 0, 0); \\
\textbf{depth} & = & 16; \\
\textbf{radius} & = & 4.
\end{array}
$$

This would define the conical-bottomed hole illustrated in Figure 3.12(a). Similarly, the following values would define a pocket with a single island as pictured in Figure 3.12(b):

$$
\begin{array}{lcl}
\textbf{location} & = & (0, 0, 1); \\
\textbf{orientation} & = & (0, 1, 0); \\
\textbf{depth} & = & 2; \\
\textbf{profile} & = & \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9, e_{10}, e_{11}\}; \\
\textbf{islands} & = & \{I_1\}; \\
\textbf{profile } I_1 & = & \{e_{12}, e_{13}, e_{14}, e_{15}\}; \\
\textbf{height } I_1 & = & 2.
\end{array}
$$

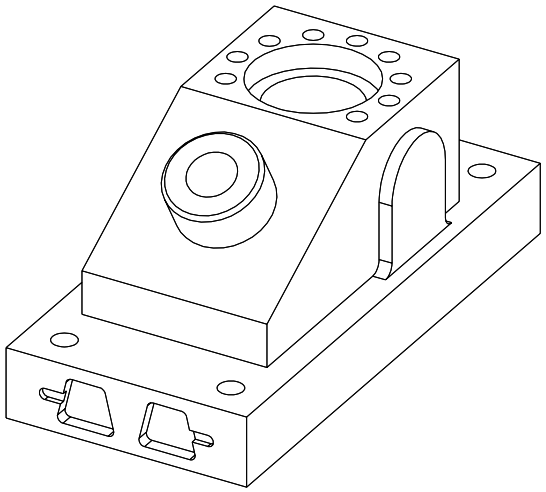


Figure 3.11: A part with a chamfer feature, the ANC101 test part from the CAM-I consortium (note the chamfered edges of the middle protrusion).

3.2 Feature-Based Models for Machining

A **feature-based model** (FBM) is a finite set of valid feature instances of machining features $F = \{f_1, f_2, f_3, \ldots, f_n\}$ that describes operations that can be used to create a part P from a piece of stock material S. The features in F describe the volume of material to be removed by the machining operations that create a part P from a piece of stock S. More specifically, an FBM is any finite set of machining features with these properties:

1. **Sufficiency**: the features in F describe one possible way to machine P from S:

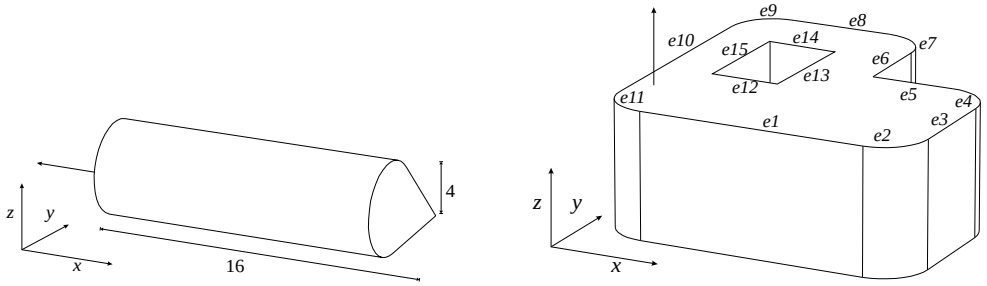
$$S -^* P \subseteq \bigcup_{f \in F} \text{rem}(f).$$

2. **Necessity**: no proper subset of F can be used to machine P from S, i.e., $\forall f_i \in F$:

$$S -^* P \not\subseteq \bigcup_{f \in (F - \{f_i\})} \text{rem}(f).$$

   In this way, an FBM does not contain redundant features and each feature of F contributes to the interpretation of the part.

3. **Validity**: as defined previously in Section 3.1. In general, validity would require that f meets machinability requirements such as those for accessibility, fixturability, and tolerances. Development of a general methodology that simultaneously considers all of these issues is beyond the scope of this thesis. Testing feature validity criteria is discussed in more detail in Chapter 7.



(a) An instance of a drilling feature. (b) An instance of a milling feature.

Figure 3.12: Example instances of drilling and milling features.

An FBM can be thought of as a high-level, unordered machining operation plan. Note that there may be many FBMs of P and S, each corresponding to a different interpretation of the part P as a set of machining features F. F need not model the optimal way of machining the design as there may exist many alternatives, each corresponding to a different collection of machining operations that could be used to produce the design from a given piece of stock material. By analogy, just as a solid model provides a unique description of the geometry and topology of an artifact, a feature-based model provides a machining features description of a part.

For a downstream manufacturing application, generation and evaluation of machining plans is performed by generating FBMs and determining possible orders in which the features in the FBM might be machined. Since each FBM is basically a volumetric set cover of the delta volume produced from the feature alternatives, models can be generated with variations on set covering techniques [147, 137], using pruning heuristics to discard unpromising FBMs. This was the approach taken to manufacturability by Gupta [70].

3.3 Primary and Well-Behaved Machining Features

For arbitrary manufacturing domains, the set of all valid features $\mathcal{V}$ can be unmanageably large (even infinite). Hence it is necessary to isolate a set of representative feature instances from the set of possible valid features. In particular, we define the set of **primary features** by imposing restrictions on the set of possible features. It has been shown that by manipulating primary features, one can generate and reason about the other feature instances of interest [70].

For an arbitrary part P there exists a set of valid features $\mathcal{V}$. One immediate observation is that given two features f and g in $\mathcal{V}$, if f's effect on the workpiece is subsumed by g's effect on the workpiece, then f can be ignored. We define a relation $\preceq$ among machining features as:

$$f \preceq g \Longrightarrow \mathrm{rem}(f) \cap^* S \subseteq \mathrm{rem}(g) \cap^* S$$

Specifically for the feature types in $\mathcal{M}$ this implies:

- Drilling: g has a depth and a radius greater than or equal to those of f;
- Milling: the 2D surface defined by the edge profile of g contains the 2D surface defined by the edge profile of f;
- Chamfering and Filleting: g's edge profile contains the edge profile of f.

Intuitively, these conditions imply that f can be replaced with g in any feature-based model that contains f. Given a set of valid feature instances, one can use the relation $\preceq$ to determine which features to keep and which to discard.

We can also use $\preceq$ to isolate which features from $\mathcal{V}$ are most suitable for generation and evaluation of feature-based models, as discussed in Section 3.2. In particular, we can use $\preceq$ to partition $\mathcal{V}$ into disjoint classes of **equivalent** features by building an equivalence relation on $\mathcal{V}$. Two features f and g are **equivalent** ($f \simeq g$) if there exists a valid feature h of the same type and setup orientation as both f and g such that $f \preceq h$ and $g \preceq h$.

Proof that $\simeq$ is an equivalence relation on $\mathcal{V}$:

- Reflexive: $f \simeq f$ follows from the definition of $\preceq$.
- Symmetric: Given that $f \simeq g$, we know $\exists h$ such that $f \preceq h$ and $g \preceq h$; using the same h, $g \simeq f$.
- Transitive: Given that $f \simeq g$ and $g \simeq h$, show $f \simeq h$.
  Proof of transitivity:
  $f \simeq g \Rightarrow \exists i$ such that $f \preceq i$ and $g \preceq i$;
  $g \simeq h \Rightarrow \exists i'$ such that $g \preceq i'$ and $h \preceq i'$
  We must show that there exists a feature j such that $f \preceq j$ and $h \preceq j$.
  There are three cases, if f and h are
  - Drilling features: then h is deeper and larger than g, therefore it is also deeper and larger than f;
  - Milling features: then h's edge profile contains g's edge profile which in turn contains f's edge profile;
  - Chamfering and Filleting features: Similar to the situation for milling features, h's edge profile contains g's edge profile which in turn contains f's edge profile.

  Hence $\simeq$ is an equivalence relation.

The equivalence classes induced on $\mathcal{V}$ by $\simeq$, $[\mathcal{V}]_{\simeq}$, can be used to define the set of primary machining features $\mathcal{P}$; i.e., each equivalence class contains one instance of a primary feature. We will be interested in features that correspond to the maximal realistic machinable volume made by a single machining operation in a single machining setup. Let C be a set of features (for example, those features in one of the equivalence classes $[\mathcal{V}]_{\simeq}$) and let f be any feature in C. Then f is C-**primary** if f satisfies these properties:

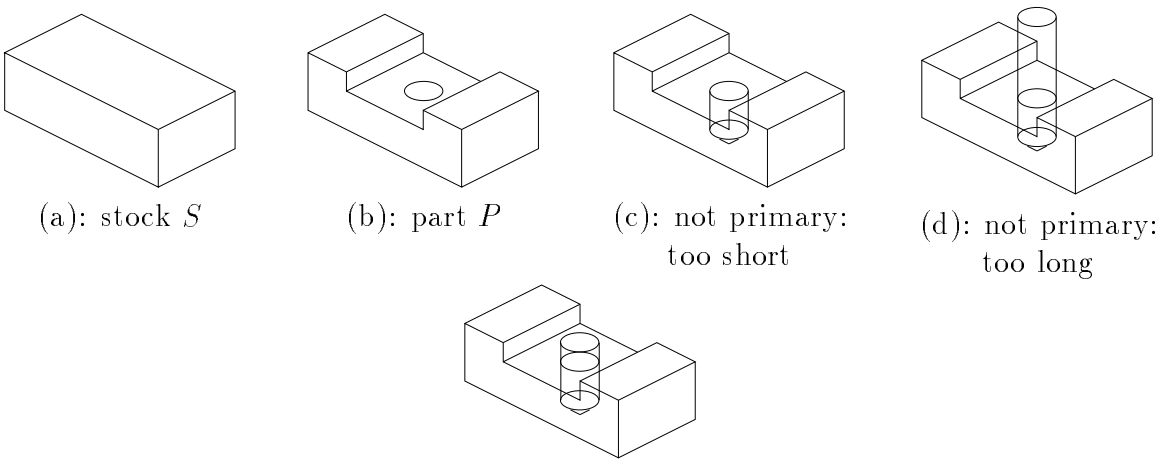


Figure 3.13: Example of primary and non-primary drilling features (from [70]).

- f removes as much stock material as possible without intersecting P. In other words, $\text{rem}(f)$ does not intersect P, and there is no feature instance g in C that has the same machining operation and orientation as f, does not intersect P, and removes more material than f. More formally, there is no g in C with the same orientation vector as f such that:

$$\text{rem}(f) \cap^* P = \emptyset$$

$$type(f) = type(g)$$

$$\text{rem}(g) \cap^* P = \emptyset$$

and

$$(\text{rem}(f) \cap^* S) \subset (\text{rem}(g) \cap^* S).$$

- f is the smallest feature in C that satisfies the above property. In other words, for every feature instance g in C that satisfies the above property,

$$\text{rem}(f) \subseteq \text{rem}(g).$$

A feature is **primary** if it is $[\mathcal{V}]_{\simeq}$-primary.

Figure 3.13 shows examples of primary versus non-primary drilling features. Figure 3.14 shows examples of primary and non-primary milling features. Figure 3.15 shows a similar example where the primary features take into account the radius of the milling tool used to machine them.

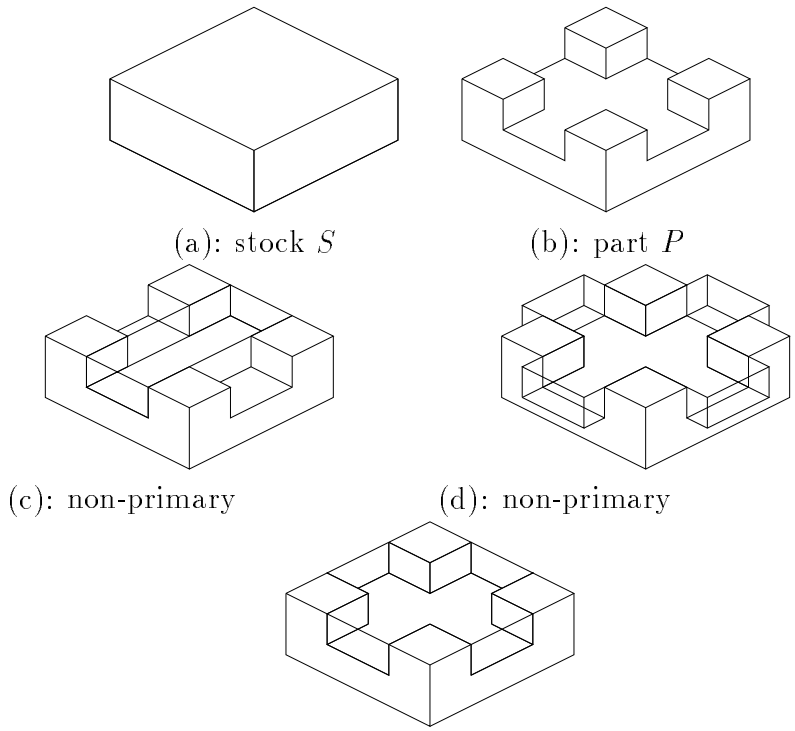


Figure 3.14: Examples of non-primary and primary instances of milling features (from [70]).

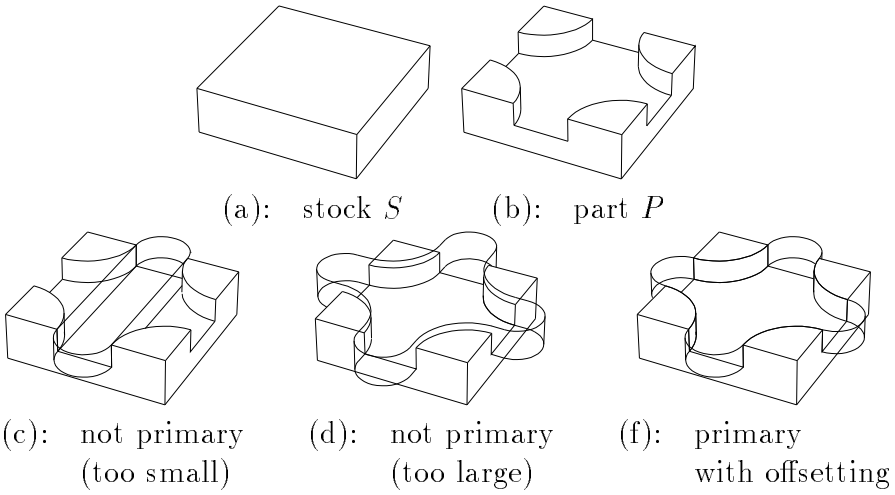


Figure 3.15: Examples of non-primary and primary instances of milling features when the radius of a milling tool is taken into account. See Chapter 7 for a full description of feature offsetting.

Well-Behaved Features. Primary features can occur in a number of configurations and not all of these yield features that describe reasonable machining operations. Note that the removal volumes for the features in $\mathcal{M}$ are bounded by different types of surfaces, each of which (planar, conical, etc.). When these surfaces occur on the boundary of a part P they can be considered a subset of the boundary of one or more instances of machining features in $\mathcal{M}$. As defined above, primary features are not related to their contributions to the shape of the part. Boundary information can be used to eliminate unrealistic primary features. In particular, we define a feature f to be **well-behaved** if it satisfies any of these properties:

1. f is a C-primary drilling feature, where C is the set of all drilling features f in $\mathcal{V}$ such that f's ending surface or a portion of its side surface is part of $b(\Delta)$, the boundary of Δ.

2. f is a primary milling feature, and f subsumes a C-primary milling feature, where C is the set of all milling features g in $\mathcal{V}$ such that the edges in $b(\Delta)$ contain portions of two or more non-collinear edges from g's edge profile. For an example, see Figure 4.6(c).

3. f is a C-primary through milling feature where C is the set of all through milling features f in $\mathcal{V}$ such that $b(\Delta)$ contains a portion of one or more of f's curved side surfaces or portions of two or more of f's planar side surfaces.

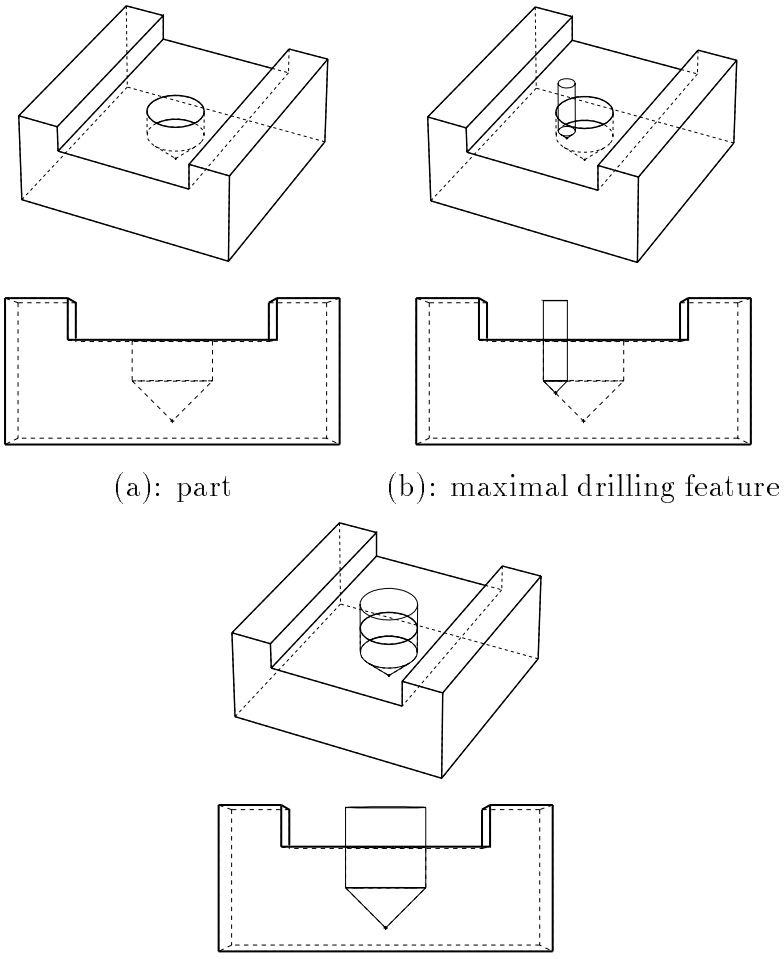
4. f is a C-primary filleting or chamfering feature, where C is the set of all filleting and chamfering features f in $\mathcal{V}$ such that $b(f) \subseteq b(\Delta)$ and every edge in $b(\text{rem}(f) \cap^* \Delta) \cap^* b(\Delta)$ is an edge in the set of edges of Δ.

Figure 3.16 shows an example of the distinction between primary and well-behaved drilling features; Figure 3.17 shows the same for milling features. For the example bracket in Figure 3.1, Figure 3.18 shows an example of a well-behaved feature set.

In previous work, it has been shown that well-behaved primary features can be manipulated to produce other feature instances in the same equivalence class as well as to generate the machining volumes used in a machining plan. Furthermore, a primary feature instance can provide good upper and lower bounds on parameters such as machining cost for the other features instances in the same equivalence class (the largest reasonable machining operation is described by the primary feature). Hence, the set of primary feature instances is suitable for performing many downstream applications (such as manufacturing planning or manufacturability analysis) [70]. The truncation operation will depend upon the other features used to generate the plan, and is discussed in greater detail in [67, 71, 70].

3.4 Summary

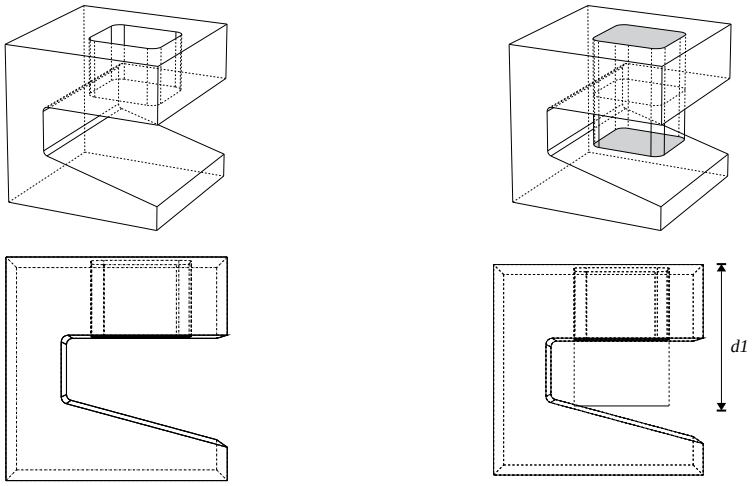
This chapter has introduced a class of features for machining and showed how to define feature-based models, as well as primary and well-behaved features in the context of machining features.



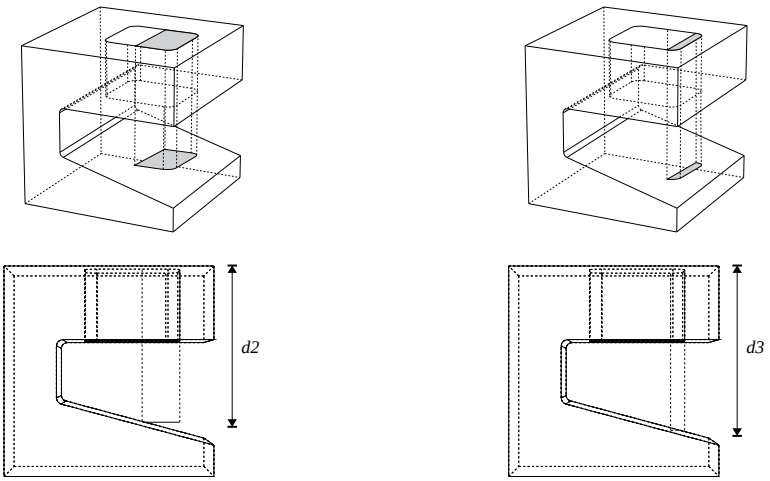
(a): part (b): maximal drilling feature

(c): well-behaved primary drilling feature

Figure 3.16: Maximal versus well-behaved drilling features (right isometric and side views). In Figure (b), $(b(\mathrm{rem}(f))) \cap^* b(P) = \emptyset$.



(a): part (b): well-behaved primary milling feature



(c): primary milling feature (d): primary milling feature

Figure 3.17: Primary versus well-behaved milling features (right isometric and side views).

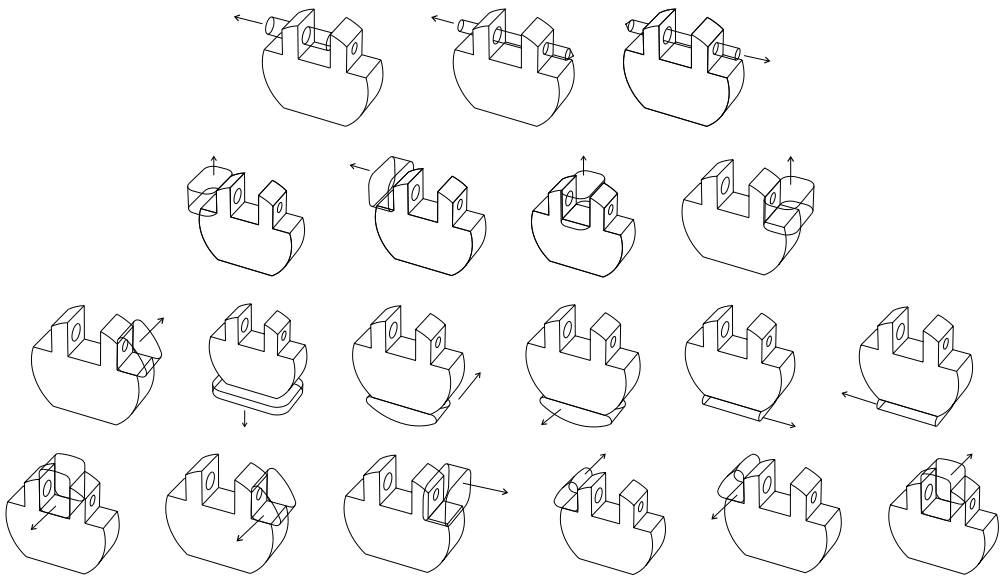


Figure 3.18: A well-behaved primary feature set for the bracket example of Figure 3.1. Arrows denote feature orientation. The curved edges on the milling features take into account the radius of a end-milling cutting tool (discussed in more detail in Chapter 7).

Chapter 4

Recognition of Features

This chapter presents our trace-based approach to recognizing instances of the machining features defined in Chapter 3.

4.1 Problem Specification

Prior to development of recognition algorithms a problem specification is required. This section explores several different problem definitions and develops a new specification for the feature recognition problem that is subsequently addressed in this thesis.

In particular, given solids representing the part P and the stock S as input, ideally one would like an algorithm that returns the set of all valid features that can be used to generate a machining plan for producing P from S. In this vein, the problem for this class of machining features might be stated as follows:

Definition 4.1.1 Feature Recognition

INPUT:	given a part P and a piece of stock S
OUTPUT:	return the set of all valid feature instances $\mathcal{F}$ found from P and S.

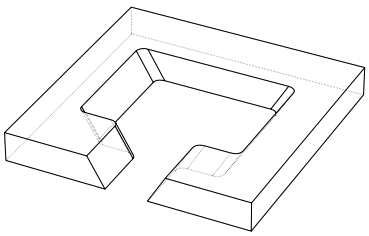
This definition corresponds to the task of finding all valid machining features that might be used to manufacture of a particular part. Computationally this definition produces the following problem:

Observation. There exist parts for which there are infinitely many valid feature instances.

Example 1. As an example, consider the part shown in Figure 4.1(a). This part has an angled pocket and needs to be machined from a rectangular block of stock using standard end-milling operations. We shall assume that our manufacturing resources include end-milling tools ranging in size from 3mm to 100mm (in 1mm increments). As shown in Figure 4.1(b), because of the angle on the walls of the pocket, two end-milling operations are needed to

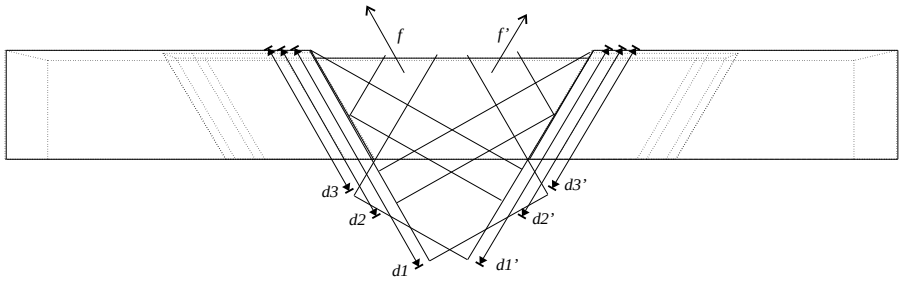
create it. Therefore, we need to represent this pocket as two end-milling features f and f'. Any value of d between d_1 and d_3 can be selected as the depth of the end-milling feature f; depending on d, there are many possible values for d', the depth of the end-milling feature f'. d' can have values between d'_1 and d'_3—this leads to infinitely many possible combinations of feature instances f and f'. Which pair of these feature instances are most appropriate depends on the available manufacturing resources and the optimization criteria. If this part had other features, they would also influence which of these possible parameterizations produces the most desirable feature instances f and f'.

In general, if a feature parameter (such as width or depth of cut) can be assigned values from a continuous scale (such as from a range of real numbers) and none of the values results in an invalid feature (i.e., makes invalid every plan that includes this feature), there will be an infinite set of feature instances for the part.



(a): upper right view

(b): front view

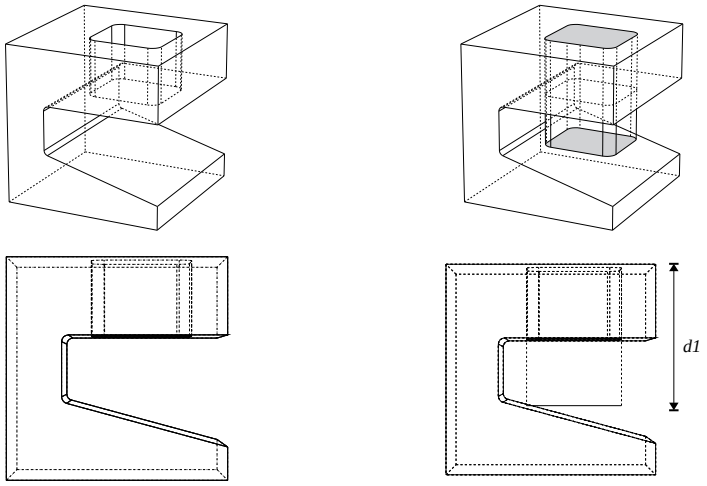


(c): possible depths for features f and f'

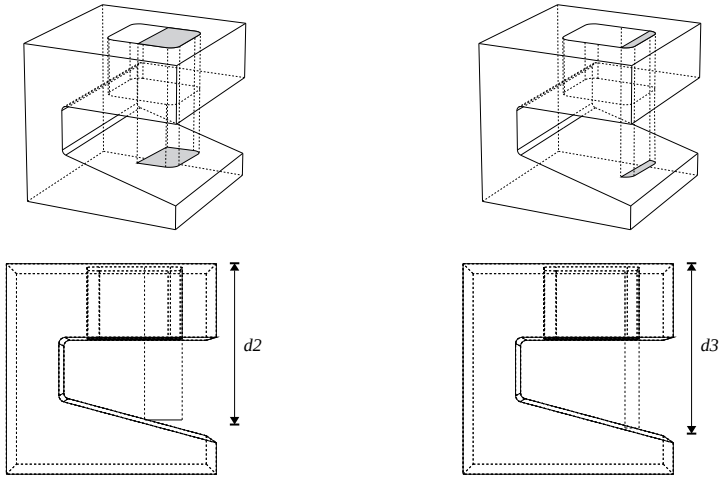
Figure 4.1: A part that gives rise to infinitely many unique feature instances of end-milling features f and f' (arrows denote feature orientation).

Example 2. One possible modification to this definition might be to consider a **maximal** feature instance. In the context of machining, a feature f is **maximal** if, given a part P and piece of stock S, it removes as much material as possible; i.e., for all other features g, $\text{rem}(g) \cap^* S \subseteq \text{rem}(f) \cap^* S$ and $\text{rem}(g) \cap^* P = \text{rem}(f) \cap^* P = \emptyset$.

Consider the example of a bottomless pocket pictured in Figure 4.2. In this case, depending on the depth d of the milling features there are infinitely many possible choices for



(a): part (b): milling feature of depth d_1



(c): milling feature of depth d_2 (d): milling feature of depth d_3

Figure 4.2: The part from Figure 3.17 which happens to give rise to infinitely many unique primary feature instances.

profiles of features to describe valid machining operations to manufacture portions of the pocket's volume. While only a few of the potential profiles are shown in the figure, there is a continuum of choices for where one might locate the bottom surface of a feature. Each of these choices gives rise to a different maximal feature profile (i.e., the profile cannot be enlarged any further in either the x or y directions without creating a feature that intersects with the part).

From these observations, it is evident that developing a general algorithm for producing the set of all valid feature instances (or maximal valid feature instances) is not possible. This stems from the fact that the above definition contains no notion of **which** of the many possible valid features one wants. In practice, one wants a set of features that optimizes some evaluation criteria—for example, an objective function measuring manufacturing cost. In this case, the goal of the feature recognition procedure is to directly recognize the set of features that map to a good (or even optimal) operation plan. This provides us with a second possible definition for the problem:

Definition 4.1.2 Feature Recognition

INPUT:	given a part P, a piece of stock S, and a function $\mathcal{C}(F)$ (e.g., $\mathcal{C}$ estimates the cost of the features in F).
OUTPUT:	return a feature-based model F for P and S with minimal cost, i.e., $\mathcal{C}(F) \leq \mathcal{C}(F')$ for all other feature-based models F' of P and S.

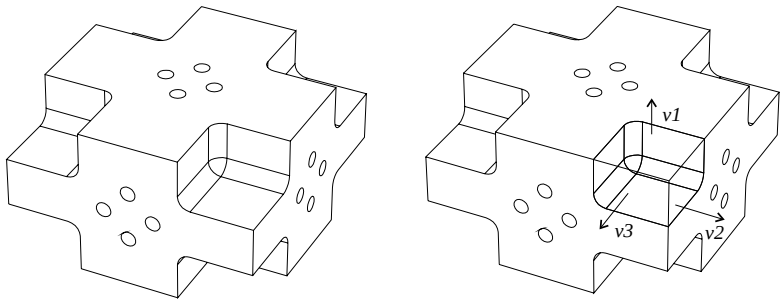
Under this definition, the feature recognition problem includes the task of finding the optimal feature-based model. In considering manufacturing constraints and knowledge from machining, feature recognition becomes an optimization problem. Many of the valid feature instances can be ignored because they are not going to prove most cost effective in a manufacturing plan; this reduces the infinite problem defined by Definition 4.1.1 to one of searching the space of alternative feature-based models.

Observation. In the worst case, for an arbitrary part P and piece of stock S, there exist exponentially many alternative feature-based models F.

Let $\mathcal{F}_r$ be a finite subset of the valid feature set (i.e., $\mathcal{F}_r \subset \mathcal{F}$) and let $\mathcal{I}$ be the size of $\mathcal{F}_r$ ($\mathcal{I} = |\mathcal{F}_r|$). Let $\mathcal{A}$ be the number of alternative feature-based models that can be defined using the feature instances in $\mathcal{F}_r$.

Consider the case in which a part can be expressed as m spatially disjoint regions to be manufactured and there are n_i choices of possible feature instances for the i^{th} region. Therefore $\mathcal{I} = n_1 + n_2 + \cdots + n_m$. The number of alternative feature-based models for this part is $\mathcal{A} = n_1 \times n_2 \times \cdots \times n_m$. The worst case for $\mathcal{A}$ will be when $n_1 = n_2 = \cdots = n_m$. If we let $n = n_1$, we get $\mathcal{I} = n \times m$, and $\mathcal{A} = n^m$. By substituting $m = \mathcal{I}/n$,

$$\mathcal{A} = (n^{1/n})^{\mathcal{I}}.$$



(a) (b)

Figure 4.3: A part that gives rise to exponentially many alternative feature-based models.

To determine the worst case, we differentiate

$$ln(\mathcal{A}) = \frac{\mathcal{I}}{n} ln(n)$$

$$\frac{dn}{d\mathcal{A}} = \frac{\mathcal{I}\mathcal{A}}{n^2}(1 - ln(n)).$$

Substituting for $\mathcal{A}$,

$$\frac{dn}{d\mathcal{A}} = \frac{\mathcal{I}(n^{1/n})^{\mathcal{I}}}{n^2}(1 - ln(n))$$

$\frac{dn}{d\mathcal{A}}$ equals to 0 only when

$$(1 - ln(n)) = 0.$$

Therefore the worst case occurs when $n = e$. This implies that, for discrete sets of features, the worst case is when $n = 3$.

Substituting 3 for n we get $\mathcal{A} = (\sqrt[3]{3})^{\mathcal{I}}$. Hence in the worst case the number of feature-based models for the part is exponential in number of feature instances

$$\mathcal{A} \in O(k^{\mathcal{I}}).$$

Consider the part shown in Figure 4.3(a). If machined from a rectangular piece of stock material, each of the 8 disjoint corner "notches" generates three possible feature instances, shown by the arrows in Figure 4.3 (b). Therefore, these 24 possible feature instances result in 24^3 different feature-based models that describe the notched regions of the part (when the through holes are taken into account the total number of FBMs is $24^3 \cdot 2^8$). An earlier version of this proof was presented in [72]; similar results have been reported by Han and Requicha [74].

In Definition 4.1.2, the feature recognition problem has been combined with the need to select an optimal set of features for some downstream application. An example of this type of definition can be found in many approaches to process planning, where a process

plan is developed as features are recognized. This type of approach has two immediate drawbacks, as mentioned in Chapter 2. First, without information about the other features in the feature-based model extensive backtracking through the exponential search space of FBMs might be required to produce optimal results. Second, with this definition, feature recognition discards features based on domain-specific evaluation criteria—thus restricting the overall utility of the output of the system. For example, the features best for process planning may not be best for automated redesign.

A third possibility is to recognize a subset of all valid feature instances that includes a representative set of features that are suitable for interfacing with downstream applications, such as cost estimation and operation planning. In the scope of this thesis, we shall define the feature recognition problem as the task of finding the set of **well-behaved** primary feature instances, as defined in Section 3.3:

Definition 4.1.3 Feature Recognition

INPUT:	given a part P, a piece of stock S.
OUTPUT:	return the set $\mathcal{F}$ of well-behaved primary features occuring in the feature-based models of P and S.

Observation. In the worst case, there are at most polynomially many well-behaved primary features.

Let f be a well-behaved primary feature from the feature class defined in Chapter 3 and assume that there are at most $O(n)$ geometric and topological entities in the boundary representation of the delta volume Δ.[1] There are four possibilities for what f is:

1. f is a drilling feature: Because f is well-behaved, we know that its removal volume $\text{rem}(f)$ must share a portion of its boundary with the boundary of the part Δ (i.e., $b(\text{rem}(f)) \bigcap^* b(\Delta) \neq \emptyset$). Further, the surface $b(\text{rem}(f)) \bigcap^* b(\Delta)$ is describable with at most two well-behaved primary features (one feature in the case where no through feature is possible).

   If there is another well-behaved primary feature capable of describing $b(\text{rem}(f)) \bigcap^* b(\Delta)$, then there would exist a drilling feature g, $g \neq f$, such that $b(\text{rem}(g)) \bigcap^* b(\Delta) \subseteq b(\text{rem}(f)) \bigcap^* b(\Delta)$. Looking at the feature parameters, if this is the case then we know that the radius of g cannot be less than that of f (otherwise g would not be primary); in addition, the radius of g cannot be greater than that of f (otherwise f would not be primary). Hence the radius of g is the same as that of f. Similar arguments can be made for the other feature parameters, therefore $f = g$. Therefore the number of well-behaved primary drilling features is $O(2 \cdot n)$.

2. f is an end-milling feature, in which case there are two possibilities:

[1] Please see Chapter 5 for a discussion of how to calculate a reasonable value for n.

(a) The edges of $b(\Delta)$ contain portions of two or more non-collinear edges from the edge profile of a primary end-milling feature g which is subsumed by f. In this case, given that there are $O(n)$ geometric and topological entities in $b(\Delta)$, there can be at most $O(n^2)$ possible fs.

(b) $b(\Delta)$ contains a portion of one or more of f's curved side surfaces or portions of two or more of f's planar side surfaces. Similarly, because the b-rep of $b(\Delta)$ is of size $O(n)$ there can be at most $O(n + n^2)$ possible fs.

For each of these cases, if an additional well-behaved primary feature is capable of covering $b(\text{rem}(f)) \bigcap b(\Delta)$, then there would exist a milling feature g, $g \neq f$, such that $b(\text{rem}(g)) \bigcap b(\Delta) \subseteq b(\text{rem}(f)) \bigcap b(\Delta)$. Considering the feature parameters, if this is the case then we know that the profile of g cannot be smaller than that of f (otherwise g would not be primary); in addition, the profile of g cannot be larger than that of f (otherwise f would not be primary). Hence the profile of g is the same as that of f and the depth of the features must be different. However, if both f and g have the same profile and different depths then one of them is not primary (i.e., the shorter feature can have its depth extended), therefore $f = g$ and the maximum number of well-behaved end-milling features is the total number of possibilities for the two cases above. Each of these cases can give rise to at most a constant number of end-milling features, therefore the total number is $O(n + 2 \cdot n^2)$.

3. f is a chamfering feature: Because f is a well-behaved chamfering feature we know that $b(\text{rem}(f)) \bigcap^* b(\Delta) \neq \emptyset$ contains the set of faces of the part that are effected by f. Further, we know that the edges in the edge profile for the feature are each bounding edges for these faces. If there existed another primary chamfering feature g that machined the same collection of faces, it would have an edge profile longer than the edge profile of f and contain the edge profile of f. Such a feature would contradict the assumption that f was primary. Hence the number of well-behaved chamfering features is in the order of the number of surfaces those features might have created. Because any given surface in the part P can belong to at most two chamfering features, the total number of well-behaved chamfering features is $O(2 \cdot n)$.

4. f is a filleting feature: This case is the same as that for chamfering features.

Based on the values for each feature type, the number of well-behaved primary feature instances is $O(n^2)$.

In this section we have shown that recognition of well-behaved primary features is a tractable goal. Techniques for recognizing this class of features are presented in the remainder of this chapter.

4.2 Trace-based Recognition of Features

In this section we present our methodology for recognizing the class of features $\mathcal{M}$ defined in Chapter 3. In particular, the algorithms developed below reconstruct the **effective removal volume** ($\text{eff}(f)$) for well-behaved machining features f from the geometry and topology of the part P and stock S. To accomplish this, we will employ a **trace-based** approach to feature recognition.

A **trace** represents the partial information remaining in the solid model of the part produced by an instance of a feature. A trace can comprises of geometry and topology, design features, tolerances, and other design attributes associated with the CAD model. While the traces addressed in this thesis cover only geometry and topology, this approach can be expanded upon with traces based on other forms of design information or enhanced with more domain-specific machining knowledge. A trace t_M for a feature type M represents:

- the information contributed to the part by an instance of a feature of type M (similar to the notion of **feature presence** [200]).
- sufficient information to calculate the parameters of one or more equivalent feature instances of type M that are also capable of creating the trace.

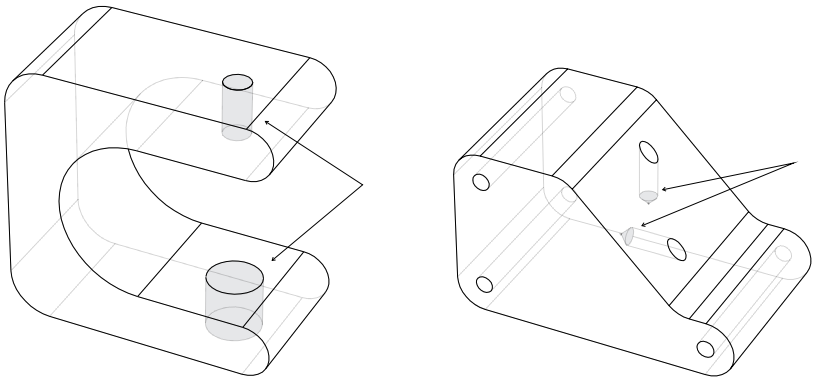
Trace-based approaches have several properties that are just beginning to be exploited by researchers, including:

- Feature traces can be derived from a variety of design information, such as tolerances, surface finish requirements, and functional information associated with surfaces.
- Feature classes can be customized by users. Recognition routines for new features can be built by introducing traces for the new features and methods for building instances of the new features from these traces.
- Trace-based techniques can be adapted to recognize features from a variety of manufacturing domains and processes. Existing feature recognition literature focuses primarily on machined parts, due in part to the fact that the functionality of solid modeling systems is well particularly suited for manipulating volumes that describe material to be machined and decomposing these volumes into features.

As noted in Section 2.4, trace-based techniques have been addressed previously by a number of researchers [117, 200, 73].

We are interested in recognizing the effective removal volumes for well-behaved primary features; hence the traces used to develop recognition algorithms are based on the properties of these features as presented in the previous section. For example, one trace for the drilling feature in Figure 3.13 is the conical ending surface of the hole. Similarly, a trace for the end-milling feature in Figure 3.15 might be its bottom surface. Techniques are presented in Chapter 7 to deduce the removal volume $\text{rem}(f)$ and the accessibility volume $\text{acc}(f)$ from the effective removal volume $\text{eff}(f)$.

The basic components of this approach to feature recognition are:



(a): Trace 1: cylindrical surface (b): Trace 2: conical surfaces

Figure 4.4: Parts illustrating drilling traces 1 and 2.

1. A finite set of feature types, $\mathcal{M}$. In the context of this thesis, $\mathcal{M}$ is made up of the feature types defined in Chapter 3.

2. Each feature type M in $\mathcal{M}$ has associated with it a finite set of traces $t_{M_1}, t_{M_2}, \ldots t_{M_k}$. Traces are developed below in Section 4.2.1.

3. For each trace, t_{M_i}, there is a procedure $\mathcal{P}_{t_{M_i}}()$ such that $\mathcal{P}_{t_{M_i}}()$ constructs, from the information in the solid model of the part and stock material, instances of features of type M capable of producing the trace t_{M_i}. Algorithms are presented at the end of this chapter (Section 4.3).

4.2.1 Defining a Set of Traces

This section introduces traces for the features defined in Chapter 3.

Drilling features. Instances of well-behaved primary drilling features can be found from any subface of their conical end surface or cylindrical side face. From these surfaces, one can determine the radius and orientation of a drilling feature. In the event that the surface was produced by a hole extending through the part, there are two possible primary machining feature instances: one in each direction along the axis of the cylindrical surface. These through holes are modeled as two unique drilling features as opposed to as a single form feature. For non-through features (those accessible in only one direction) the location for the primary feature instance can be found from the end surface, if one exists, or by calculating the deepest point at which the conical tip of the drilling tool (see Figure 3.2) can be placed without intersecting the part.

We present here two traces for drilling features:

1. Trace 1: any convex cylindrical surface s_c in the delta volume is a possible side surface created by a drilling operation.

   Rationale: This trace is used to build instances of drilling features when a portion of their side surface remains on the boundary of the delta volume. An example of this trace is illustrated in Figure 4.4(a).

2. Trace 2: a convex conical surface s_f in the delta volume as a conical ending surface describing the cutting tip of a drilling tool.

   Rationale: This trace is used to build an instance of a drilling feature when only a portion of its ending tip surface remains on the boundary of the delta volume. An example of this trace is illustrated in Figure 4.4(b).

End-milling features. Trace 1 for end-milling features is edge-based and captures a large number of configurations of milling features; traces 2 and 3 are used to build instances of end-milling features when only a portion of their side surfaces is present on the boundary of the delta volume. With traces 2 and 3, the end-milling features extend completely through the stock material. Examples of such features include through pockets and profiles.

1. Trace 1: an edge $e_1 = \langle v_1, v_2 \rangle$ in the delta volume.

   Rationale: This trace is used to determine the profile of end-milling features. Given an edge $e_1 = \langle v_1, v_2 \rangle$, orientations and locations for potential milling features can be obtained from other edges[2] $e_2 = \langle v_3, v_4 \rangle$ in the delta volume for which the vertices $v_1, v_2, , v_3, v_4$ are coplanar. Two examples of end-milling trace 1 are given in Figure 4.5.

   For trace 1, each edge pair e_1 and e_2 in the delta volume can belong to one of three different types of well-behaved primary feature instances:

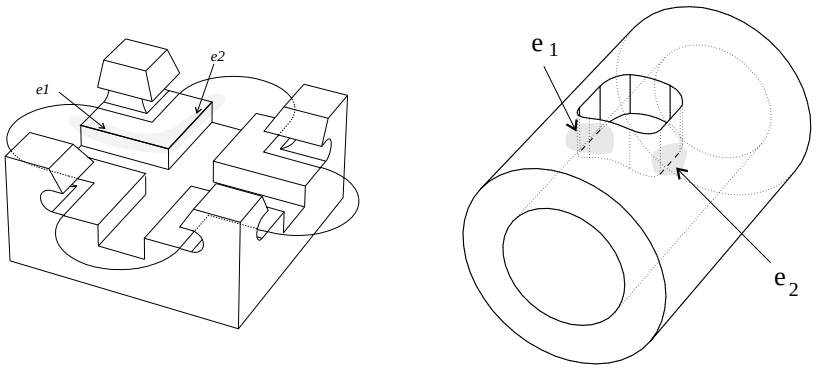
   (a) As pictured in Figure 4.6(a), edge e_1 is an edge of one of the bottom surfaces of a milling feature (i.e., e_1 could have been created as part of the planar bottom face or as a blend surface of the feature).

   (b) As pictured in Figure 4.6(b), edge e_1 could be an edge of a side surface of a milling feature having no bottom surface present in the part.

   (c) As pictured in Figure 4.6(c), edge e_1 could be a subset of an edge of a side surface of a milling feature that extends through the part (a through pocket).

   For example, one possibility is that e_1 and e_2 are coplanar with or on the boundary of the bottom surface of the end-milling feature, such as shown in Figure 4.5(b) and Figure 4.6(a). Another possibility is that the bottom surface of the end-milling feature has been eliminated through some interaction with other features, as illustrated in Figures 4.5(b) and 4.6(b) and (c).

[2]Note that in the solid model of the delta volume, the edges e_1 and e_2 might be non-linear curves, e.g., they could be elliptical.



(a): Part from Figure 1.3. (b): no bottom surface present.

Figure 4.5: Illustrations of milling trace 1. In (a) the wire denotes the profile of the desired end-milling feature.

2. Trace 2: a planar surface in the delta volume, considered as a face created by the side cutting surface of an end-mill during the same machining operation. Figure 4.7(a) shows an example of milling trace 2.

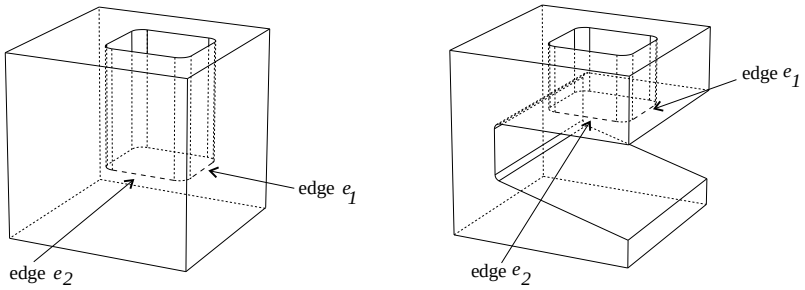
   Rationale: For some instances of through milling features, all that may remain are walls. This trace begins with a single planar wall and, by considering other planar surfaces in the delta volume, obtains orientations for potential through milling features from the normal vectors; i.e., two non-parallel planar surfaces can be used to determine the orientation of the through features that might have made them (if any) as the dot product of their normals.

3. Trace 3: a cylindrical surface in the delta volume as a surface created by the side cutting surface of an end-mill. An example of end-milling trace 3 is given in Figure 4.7(b).

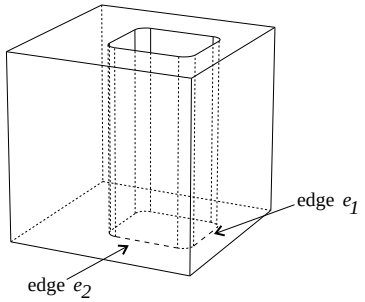
   Rationale: The profile of a milling feature might comprise curved edges, for example the corner radii created when a round tool machines a convex corner. This trace uses these curved surfaces to determine the orientation of potential through features.

Chamfering and Filleting features. In ordinary machining practice, the operations that create chamfer and fillet features usually occur after the major material removal operations. The traces for these features are based on a combination of faces and edges. The two fundamental parameters that must be determined to infer the existence of these feature types are the profile and the orientation. The profile consists of a sequence of edges and the orientation is computed using these edges along with the surfaces affected by the feature.

1. Chamfering Trace 1: a planar surface s bounded by four edges, two of which are equal in length to the cutting surface of an available chamfering tool and one of which e_1



(a): bottomed milling feature (b): bottomless milling feature



(c): through milling feature

Figure 4.6: Three possible cases of milling features.

is perpendicular to the orientation of the tool during the machining operation. An illustration of this trace is shown as edge profiles E_1 and E_2 in Figure 4.8.

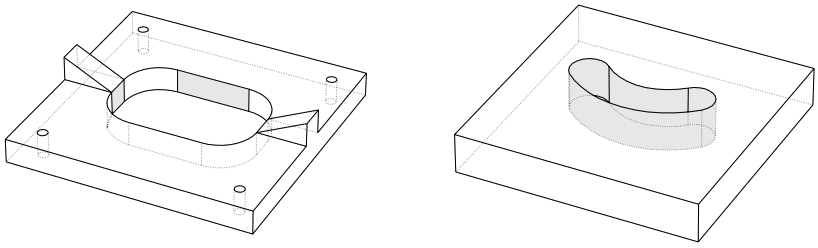
Rationale: Chamfering takes one or more convex edges of the workpiece and removes a triangular cross-section of material along the length of these edges. When the tool is moved along a straight edge profile (e.g., E_1) this process creates a new planar face on the part.

2. Chamfering Trace 2: a concave conical surface s with height and angle equal to that of the cutting surface of an available chamfering tool.

   Rationale: In this case the cutting tool is moved along a curved edge profile, creating a new conical face on the part.

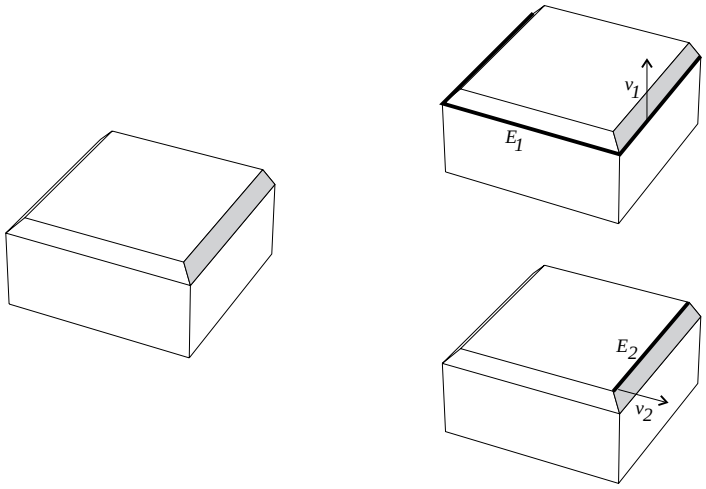
3. Filleting Trace 1: a concave cylindrical surface s with a radius equal to that of the cutting edge of an available filleting tool and with at least one bounding edge e perpendicular to the orientation of the tool during the machining operation. An illustration of this trace is shown as edge profiles E_1 and E_2 in Figure 4.9.

   Rationale: Filleting takes one or more convex edges of the workpiece and removes a quarter-circular cross-section of material along the length of these edges. When the tool



(a): Trace 2 (b): Trace 3

Figure 4.7: Parts illustrating end-milling traces 2 and 3.



(a): potential chamfered surface (b): traces for the features

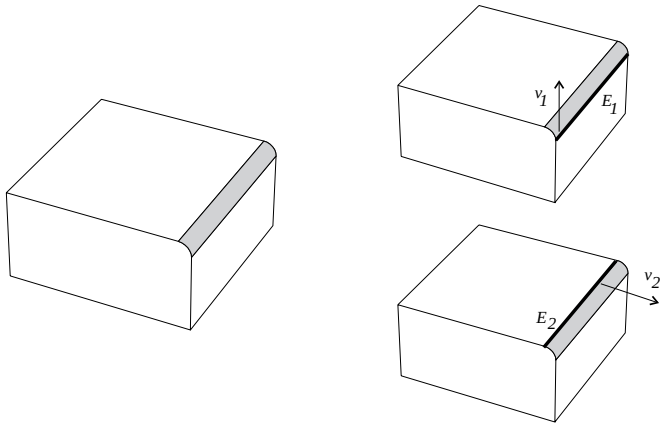
Figure 4.8: Traces for chamfering features.

is moved along a straight edge profile (e.g., E_1) this process creates a new cylindrical face on the part.

4. Filleting Trace 2: a concave toroidal surface s with a minor axis equal to that of an available filleting tool and with at least one bounding edge perpendicular to the orientation of the tool during the machining operation.

   Rationale: In this case the cutting tool is moved along a curved edge profile, creating a new toroidal face on the part.

A presentation of the details of the various procedures $\mathcal{P}_{t_{M_i}}()$ for constructing feature instances from these traces is given in the next section.



(a): potential filleted surface (b): traces for features

Figure 4.9: Traces for filleting features.

4.3 Feature Recognition Algorithms

Throughout this section we shall be using several example parts when describing the feature recognition algorithms. For most of the examples, we will use the part and stock pictured in Figure 4.10.

An algorithm for trace-based recognition of features can be given as follows:

RECOGNIZE_WELL-BEHAVED_FEATURES(P, S)
INPUT: solid models of a part P and stock S
OUTPUT: the set of well-behaved feature instances, $\mathcal{F}$.

1. Given a collection of feature types and their traces, $\mathcal{M}$, input a solid model for the part, P, and for the initial stock material, S.

2. Initialize $\mathcal{F} = \emptyset$.

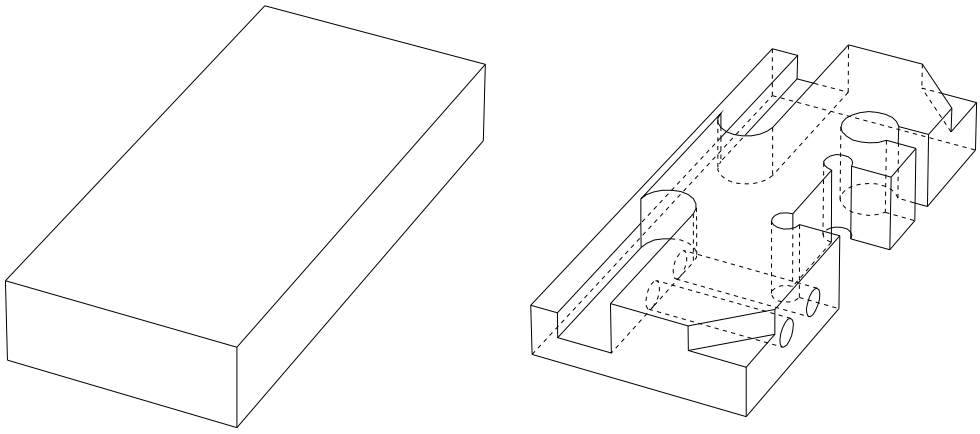
3. From P and S, identify the set of all potential traces present: $\mathcal{T}$. In this thesis $\mathcal{T}$ is a set of geometric and topological entities (such as edges, surfaces, and vertices).

4. For each potential trace t in $\mathcal{T}$ do

   (a) If t matches a t_{M_i}, call the procedure $\mathcal{P}_{t_{M_i}}(t)$ and construct (if possible) feature instances, $f_1, \ldots f_n$ of type M. Add these to the set of all feature instances, $\mathcal{F}$.

5. Return the feature set $\mathcal{F}$.

The following sections present the geometric algorithms for constructing individual well-behaved primary feature instances based on the traces in the previous section.



(a): stock (before machining) (b): part (after machining)

Figure 4.10: An example part and stock from [71].

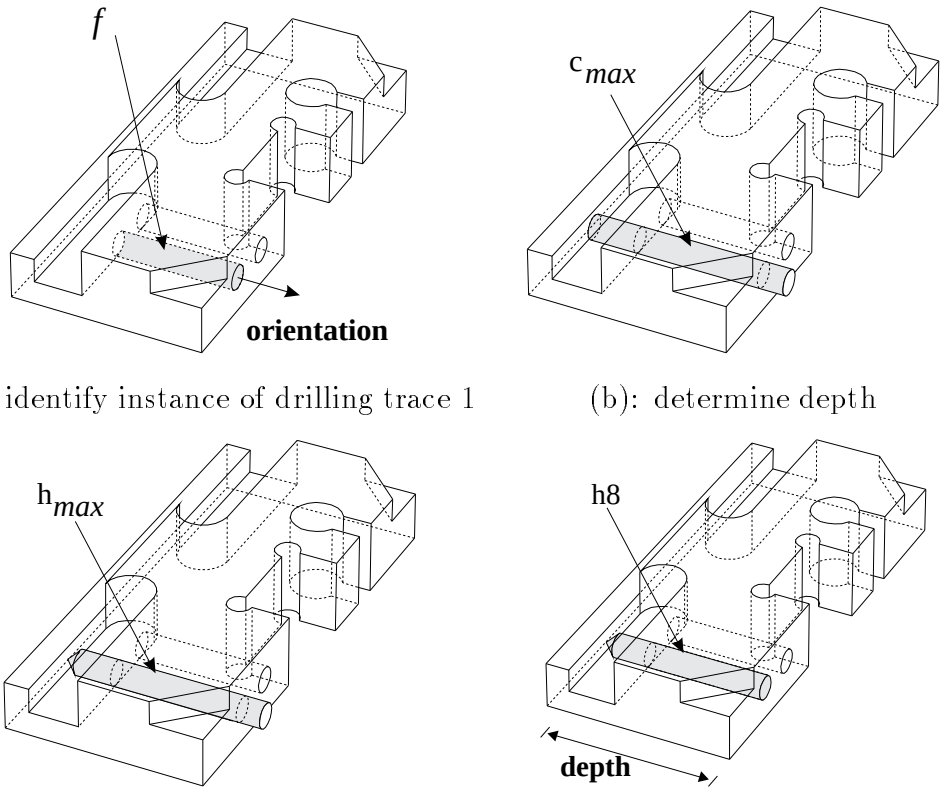
4.3.1 Recognizing Drilling Features

Well-behaved drilling features are perhaps the most straightforward to recognize. From a portion of the conical ending surface, one can determine both the location and the orientation of the drilling feature. The radius r for the primary feature is calculated based on two observations. First, it must be less than or equal to the maximum radius in the available set of drilling tools (this is discussed in more detail in Chapter 7). Second, it is the largest value such that the removal volume of the feature does not interfere with the part. The depth of the primary drilling feature is the minimal distance along its orientation vector from its location to the boundary of the workpiece.

For a given face f, there are several possibilities, as addressed below:

Algorithm: Drilling Trace 1.

1. Input face f, part P, and stock S.
2. Confirm that face f corresponds to drilling trace 1; i.e., f is a convex cylindrical face.
3. Determine values for radius and orientation parameters from f, as shown in Figure 4.11(a) where f is a cylindrical face that is part of the side of a drilling feature.
4. Find the maximal non-intrusive cylinder $c_{\max}$, as shown in Figure 4.11(b). If $c_{\max}$ cannot be extended beyond the stock, exit and return an empty list of features.
5. Determine a location for a maximal drilling feature $h_{\max}$, as shown in Figure 4.11(c). There are two cases:



(a): identify instance of drilling trace 1

(b): determine depth

(c): build feature

(d): truncate feature

Figure 4.11: Construction of a drilling feature from drilling trace 1.

(a) In the case of a through hole, a location outside the stock on the axis of the hole can be chosen for each of the two primary drilling features.

(b) In the case where f has not been made by a through hole, the end of the drilling feature is located on the planar side face of $c_{\max}$. Note that there exist situations where this yields an approximation of a primary feature; however, for purposes of machining [70], this approximation produces satisfactory results.[3]

6. Truncate [70] the features (in the example, truncate $h_{\max}$) to get the instances of the primary drilling features, as shown in Figure 4.11(d) for h_8. In the event that f is accessible bi-directionally, there will be two instances of primary drilling features (shown in Figure 4.15 for holes h_5 and h_6).

[3]The exact location point for these types of features can be calculated in several ways. We will not present an specific algorithm here as it would involve introduction of routines specific to a particular solid modeling system.

7. Exit and return the list of features that were found.

Algorithm: Drilling Trace 2.

1. Input face f, part P, and stock S.
2. Confirm that face f corresponds to drilling trace 2; i.e. f is a convex conical face.
3. Determine values for radius and orientation parameters from f where f is a conical face that is part of the tip of a drilling feature.
4. Find the maximal non-intrusive cylinder $c_{\max}$, as shown in Figure 4.11(b), and verify f's accessibility by checking that $c_{\max}$ extends beyond the stock along the orientation. If it does not extend beyond the stock, exit and return an empty list of features.
5. Determine a location for a maximal drilling feature $h_{\max}$, as shown in Figure 4.11(c), by calculating the apex of the conical surface f.
6. Truncate [70] the feature (in the example, truncate $h_{\max}$) to get the instance of the primary drilling feature, as shown in Figure 4.11(d) for h_8.
7. Exit and return a list containing the feature that was found.

4.3.2 Recognizing Face-Milling, Step-Milling and Pocket-Milling Features

The three major parameters that must be determined to recognize a milling feature are its orientation, location, and profile. These parameters are more interrelated than those of drilling features and hence there are different traces for each of the various possibilities. Determination of bottom blends are dependent on a number of tooling constraints and are handled as a part of post-processing activity, as described in Chapter 7.

Algorithm: Milling Trace 1. This is the most general trace for recognizing milling features. Starting from a single edge e_1, an orientation for a milling feature is determined from e_1 and a second coplanar edge e_2. For a given e_1, there are in the worst case $O(n)$ possible orientations for a primary milling feature, where n is the number of edges in the part P.

For each of the three cases shown in Figure 4.6, the profile of the milling feature or features are computed as a normal projection of the part faces that lie in the half-spaces above and below (with respect to the orientation) the plane containing the edges. The projection is computed onto the plane containing the edges e_1 and e_2 (shown in Figure 4.12(a)). In the first and second cases, this plane is the bottom surface of a milling feature that creates the edges and their adjacent surfaces. Note that this still applies when the bottom surface of the

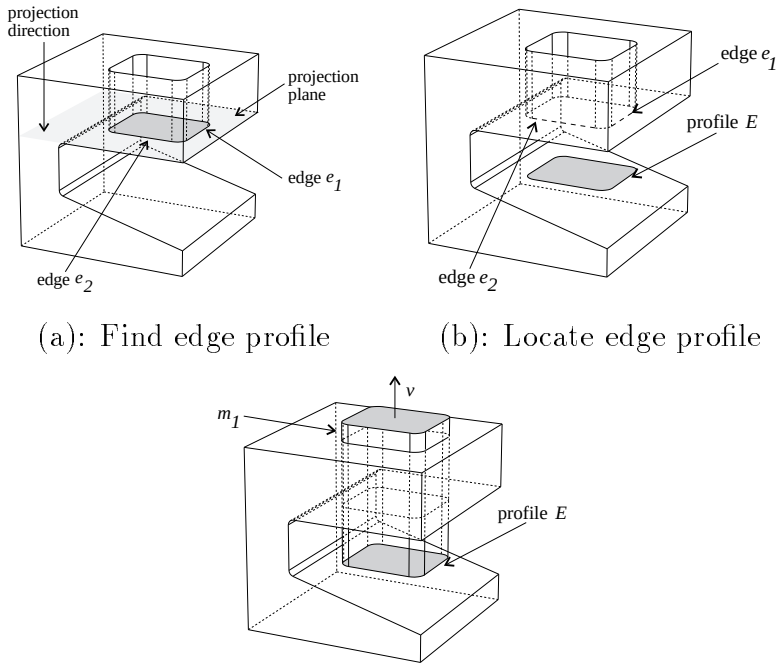
feature has been eliminated through interactions with other features and no longer exists in the model of the part P.

In the third case, the feature extends through the part and thus has no bottom surface present in the delta volume. For this situation, the plane containing e_1 and e_2 provides the orientation vectors $+/-\vec{v}$ for the through features. The part faces are mapped onto a projection plane perpendicular to $\vec{v}$, arriving at a cross-section of the through feature capable of creating these edges and surfaces.

1. Input edge $e_1 = \langle p_1, p_2 \rangle$, part P, and stock S.
2. For each $e_2 = \langle p_3, p_4 \rangle$ in the delta volume, $e_1 \neq e_2$ do:
   (a) Confirm that edges e_1 and e_2 are co-planar and non-colinear. If not, loop to (2).
   (b) Compute the orientation of the features, $(p_1 - p_3) \times (p_2 - p_4)$.
   (c) Let H be the plane through the points p_1, p_2, p_3 and p_4, and:
      i. let $H_{\mathbf{above}}$ be the projection of the faces of P lying above (+norm(H)) the plane H onto H.
      ii. let $H_{\mathbf{below}}$ be the projection of the faces of P lying below (−norm(H)) the plane H onto H.
   (d) If (a) the intersection $H_{\mathbf{above}} \cap^* H_{\mathbf{below}}$ contains an edge profile with portions of e_1 and e_2 or (b) $H_{\mathbf{above}} = H_{\mathbf{below}}$ and $H_{\mathbf{below}}$ contains an edge profile with portions of e_1 and e_2, then:
      i. Let $H' = H_{\mathbf{above}} \cap^* H_{\mathbf{below}}$ be the profile of the through feature.
      ii. Sweep H' in both directions (+norm(H) and −norm(H)) until it completely exits the stock material S.
      iii. Truncate the features to the size of the stock in order to make them primary.
      iv. Exit and return the list of features that were found.

      Else, build feature instances for each of $H_{\mathbf{above}}$ and $H_{\mathbf{below}}$:
      i. Sweep $H_{\mathbf{above}}$ along +norm($H_{\mathbf{above}}$) until it completely exits the stock material S (such as shown in Figure 4.12(a)).
      ii. Sweep $H_{\mathbf{above}}$ along −norm($H_{\mathbf{above}}$) until it hits the part P (such as shown in Figure 4.12(b)).
      iii. Select a vertex on the profile of the bottom face of this swept volume as a location for the feature.
      iv. Repeat steps 1-3 for $H_{\mathbf{below}}$.
      v. Truncate the two features to the size of the stock to make them primary.
      vi. Exit and return a list of features that were found.

Another examples of the steps in recognition of milling features from trace 1 is given in Figure 4.13.



(a): Find edge profile (b): Locate edge profile

(c): Build feature instance

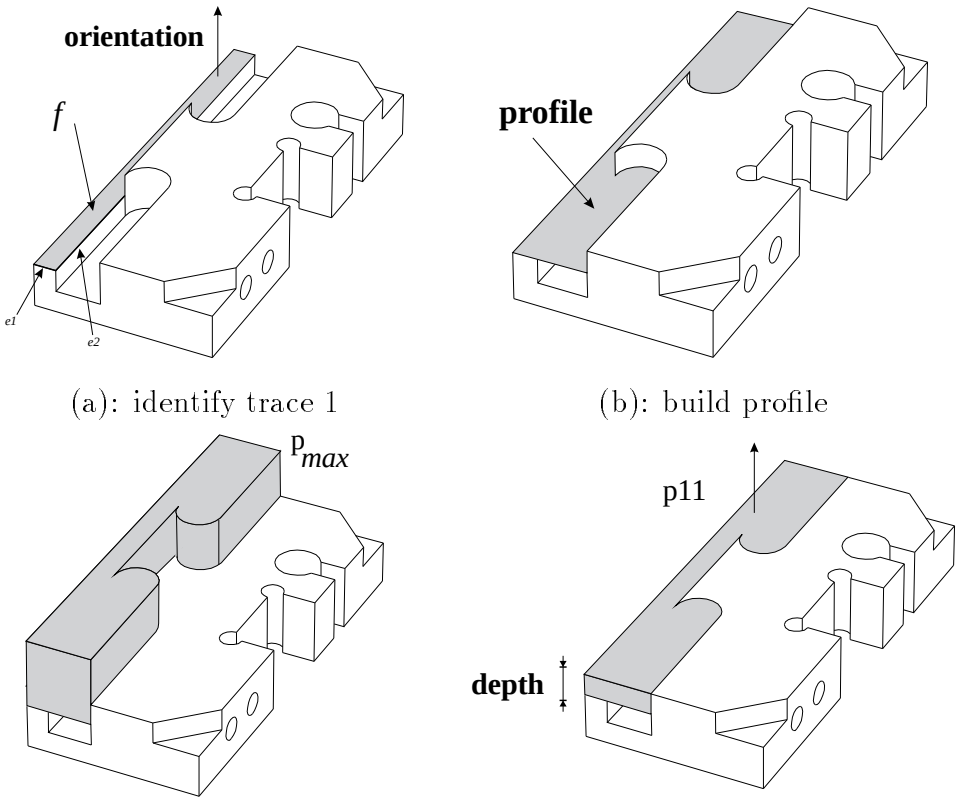
Figure 4.12: An example of the recognition of a bottomless pocket-milled feature from milling trace 1.

Algorithm: Milling Trace 2. Milling trace 2 covers a number of cases of through milling features not addressed by trace 1. With milling trace 2, construction of milling features begins with a pair of faces of the delta volume, f_1 and f_2. Milling trace 2 considers the case in which f_1 and f_2 are faces of the part P created as side surfaces of an instance of a through milling feature.

For these through features, the feature orientation is along one of the two directions $+(\text{norm}(f_1) \times \text{norm}(f_2))$ and $-(\text{norm}(f_1) \times \text{norm}(f_2))$. As these features extend through the part, there is no bottom surface present in the delta volume—hence an arbitrary location can be chosen for a projection plane p. Mapping all of the part faces onto p yields the largest milling feature profile capable of creating these surfaces. Given the profile, two primary milling feature instances are created. We truncate $p_{\max}$ to obtain the primary feature, as shown in Figure 4.13(d), with a depth sufficient to extend the feature instance outside the stock. An example of through milling features is given for features p_7 and p_8 in Figure 4.16.

1. Input face f_1, part P, and stock S.

2. For each face f_2 in Δ, $f_2 \neq f_1$ do:



(a): identify trace 1 (b): build profile

(c): build maximal feature (d): truncate feature to be primary

Figure 4.13: Construction of a step-milling feature from milling trace 1.

(a) Verify that f_1 and f_2 are non-parallel planar faces and determine the potential feature orientations: $+\text{norm}(f_1) \times \text{norm}(f_2)$ and $-\text{norm}(f_1) \times \text{norm}(f_2)$.

(b) Pick a vertex p_{f_1} of f_1 and create a plane H passing through p_{f_1} normal to $\text{norm}(f_1) \times \text{norm}(f_2)$.

(c) Project the surfaces of the part P onto H.

(d) If there is no profile created by the projection onto H incident with f_1 and f_2, exit returning an empty feature list.

(e) If there is a profile H' on H created by this projection incident with f_1 and f_2 then:

   i. Sweep H' along $+\text{norm}(H')$ and $-\text{norm}(H')$ until it completely exits the stock material S.

   ii. Select one vertex from each of the bounding edge profiles of the top and bottom faces of this swept volume to be locations for the features (one oriented $+\text{norm}(H')$, the other $-\text{norm}(H')$).

iii. Truncate the two features to the size of the stock to make them primary.
iv. Exit and return a list of the features that were found.

Algorithm: Milling Trace 3. Milling trace 3 covers a number of cases of through milling features not addressed by traces 1 and 2. In particular, it covers situations where the profile of the through milling feature contains curved edges. In this trace, construction of milling features begins with a convex cylindrical or elliptical face of the delta volume, f_1.

1. Input face f_1, part P, and stock S.
2. Verify that f_1 is a convex cylindrical or elliptical surface.
3. Determine feature orientations from the axis of f_1, $+\text{axis}(f_1)$ and $-\text{axis}(f_1)$.
4. Pick a point p_{f_1} on the axis of f_1 and create a plane H passing through p_{f_1} normal to $\text{axis}(f_1)$.
5. Project the surfaces of the part P onto H.
6. If there is no profile created by the projection onto H incident with f_1, exit returning an empty feature list.
7. If there is a profile H' created by this projection incident with f_1 and f_2 then:
   (a) Sweep H' along $+\text{norm}(H')$ and $-\text{norm}(H')$ until it completely exits the stock material S.
   (b) Select one vertex from each of the bounding edge profiles of the top and bottom faces of this swept volume as locations for the features (one oriented $+\text{norm}(H')$, the other $-\text{norm}(H')$).
   (c) Truncate the two features to the size of the stock to make them primary.
   (d) Exit and return a list of the features that were found.

Algorithm: Classification of Milling Features. Based on the edge profile and feature orientation, E and $\vec{v}$, the milling feature can be classified as pocket-milling, face-milling, or step-milling.

1. Input an edge profile E for a milling feature f, a set of island profiles I (possibly empty), and an orientation vector $\vec{v}$.
2. If I is empty and all of the edges in E are open, return "face-milling feature."
3. Else, if all of the edges in E are closed, return "pocket-milling feature."
4. Else, return "step-milling feature."

Profiles for any islands can be obtained by covering [183] the edge profile E with a planar face and calculating its 2D intersection with the part P. Any interior edge loops will correspond to islands.

4.3.3 Recognition of Chamfering and Filleting Features.

Given a surface s that is a portion of a face of a chamfering or filleting feature as shown in Figures 4.8(a) and 4.9(a), we construct a feature instance as follows:

1. Determine the set of possible orientations for feature instances that could have made the surface.

2. For each possible orientation, find the edge profile E for f and find the adjoining surfaces create by the operation. Figures 4.8(b) and 4.9(b) indicate the orientations for a chamfering tool and filleting tool.

3. Construct the feature instances, as shown in Figures 4.8(c) and 4.9(c).

The specifics of the algorithms for chamfering and filleting features depend on the parameters of the available cutting tools. In practice, such algorithms would have to interact with a manufacturing resource database—often a database specific to the machine shop at hand. Because these tool parameters can be passed into a commercial system when it is deployed, and in order to simplify the presentation of these algorithms, we will assume, referring to Figure 3.9(a), the availability of chamfering tools such that:

1. **Tip angle**: The tip angle is the angle between the cutting edge of the tool and the tool's axis. The algorithms below are based on a 45° tip angle.

2. **Cutting length**: The cutting length refers to the length of the cutting edge of the tool. In Figure 3.9(a), for tools with 45° tip angles, this length is $\frac{\sqrt{2}}{2} \cdot A$. In the algorithms below we will assume that there are minimum and maximum values for A, $A_{\min}, A_{\max}$.[4]

For filleting features the conditions are similar:

1. **Cutting radius**: The radius of the cutting surface of the tool, R as shown in Figure 3.9(b). In the algorithms below we will assume that there are minimum and maximum values for R, $R_{\min}, R_{\max}$. The value for parameter E in Figure 3.9(b) is typically a fraction of R. We will assume that $E = \frac{1}{2}R$.

2. **Cutting profile**: The cutting profile refers to the shape and length of the cutting edge of the tool. Referring to the parameters in Figure 3.9(b), the algorithms below will assume that the cutting profile is $\frac{1}{4}$ of the circular arc of radius R. Consideration of other cutting profiles (such as those with a combination of straight and curved edges) introduces additional complexity not addressed by the algorithms below.

If additional tools are available, these algorithms can be modified to retrieve tool specifications from a manufacturing resource database [93] and then determine the existence of a feature that might have been created to match those particular parameters. Other tooling-specific constraints are also addressed in Section 7.2.

[4]Specific values are discussed in Chapter 7.

Algorithm: Chamfering Trace 1.

1. Input a planar face f, part P, and stock S.

2. Verify that among the bounding edges for face f is a pair e_1, e_2 of equal length ($\text{len}(e_1) = \text{len}(e_2)$) and within the cutting length bounds for available chamfering tools ($A_{\min} < \text{len}(e_2) < A_{\max}$).

3. **Determine orientations:** As illustrated in Figure 4.8, let $e_3 = \langle p_1, p_2 \rangle$ and $e_4 = \langle p_3, p_4 \rangle$ be the other two bounding edges of the face f and calculate orientation vectors for potential features v_{o1} and v_{o2} such that: (a) v_{o1} is perpendicular to $(p_1 - p_2)$, (b) v_{o2} is perpendicular to $(p_3 - p_4)$, and (c) the angle between both v_{o1} and v_{o2} and $\text{norm}(f)$ is 45°. Start building two edge profiles, $E_1 = \{e_3\}$ and $E_2 = \{e_4\}$.

4. **Determine profile:** For the possible orientation v_{o1}, let $s_{v_{o1}}$ be the plane passing though e_3 and perpendicular to v_{o1}. Check edges e_i of Δ that lie in the plane s to see if they belong to the edge profile of the feature by asking:

   (a) Does e_i share a vertex with either of the edges at the ends of the profile E_1?

   (b) Is e_i adjacent to a face that can be made with the same chamfering operation? Test the condition in step 2 (above) and conditions in step 3 of the algorithm for chamfering trace 2 (below).

   If the answer to both of these is yes, then add e_i to the edge profile E_1. Repeat these steps until candidate edges are exhausted. This is basically a depth-first search on the edges connected to the edge e_3.

5. **Construct feature instances:** Build the profile of the tool and sweep it along the edges in the edge profile E_1. Unite the volumes created by sweeping along each of the segments of the profile and return the result—this is the chamfering feature, as shown in Figure 4.14(a).

6. Repeat steps 4-5 for e_4 and v_{o2}.

7. Return the set of the chamfering features that were found.

Algorithm: Chamfering Trace 2.

1. Input a conical face f, part P, and stock S.

2. **Determine orientations:** For a conical face f, the orientation of the feature is along the axis of the surface, $v_o = \text{axis}(f)$. Add the bottom edge e_1 of face f to the set of edges in the edge profile E.

3. Verify that the length of surface of f is within the cutting length bounds for available chamfering tools ($A_{\min} < \text{len}(f) < A_{\max}$). If yes, the f might have been made as a chamfer by a machining operation with orientation v_o.

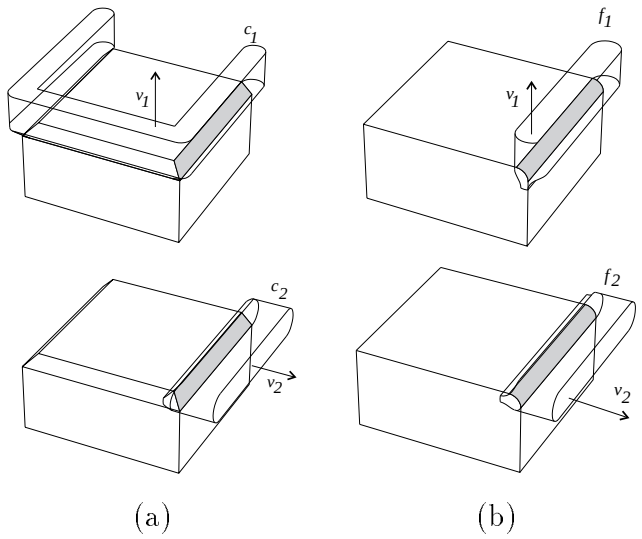


Figure 4.14: Instances of chamfering features for the part and traces in Figure 4.8 and instances of filleting features for the part and traces in Figure 4.9.

4. **Determine profile:** For the possible orientation v_o, let s_{v_o} be the plane passing though the edge e_1 and perpendicular to v_o. Check edges e_i of Δ that lie in the plane s to see if they belong to the edge profile of the feature by asking:

   (a) Does e_i share a vertex with either of the edges at the ends of the profile E?

   (b) Is e_i adjacent to a face that can be made with the same chamfering operation? Test the condition in step 2 of the algorithm for chamfering trace 1 and the conditions in step 3 (above).

   If the answer to both of these is yes, then add e_i to the edge profile E. Repeat these steps until candidate edges are exhausted.

5. **Construct feature instances:** Build the profile of the tool and sweep it along the edges in the edge profile E. Unite the volumes created by sweeping along each of the segments of the profile and return the result.

6. Return the set of the chamfering features that were found.

Algorithm: Filleting Trace 1.

1. Input a cylindrical face f, part P, and stock S.

2. Verify that the radius of the surface f is within the cutting radius bounds for available filleting tools ($R_{\min} <\text{len}(e_2) < R_{\max}$) and its profile is $\frac{1}{4}$ of a circular arc.

3. **Determine orientations:** As illustrated in Figure 4.9, let $e_3 = \langle p_1, p_2 \rangle$ and $e_4 = \langle p_3, p_4 \rangle$ be the two bounding edges of the face f and calculate orientation vectors for

potential features v_{o1} and v_{o2} such that: (a) v_{o1} is perpendicular to $(p_1 - p_2)$ and normal to the surface of f at e_3, (b) v_{o2} is perpendicular to $(p_3 - p_4)$ and normal to the surface of f at e_4.

4. **Determine profile:** For the possible orientation v_{o1}, let $s_{v_{o1}}$ be the plane passing though e_3 and perpendicular to v_{o1}. Check edges e_i of Δ that lie in the plane s to see if they belong to the edge profile of the feature by asking:

   (a) Does e_i share a vertex with either of the edges at the ends of the profile E_1?

   (b) Is e_i adjacent to a face that can be made with the same filleting operation? Test the condition in step 2 (above) and conditions in step 3 of the algorithm for filleting trace 2 (below).

   If the answer to both of these is yes, then add e_i to the edge profile E_1. Repeat these steps until candidate edges are exhausted.

5. **Construct feature instances:** Build the profile of the tool and sweep it along the edges in the edge profile E_1. Unite the volumes created by sweeping along each of the segments of the profile and return the result—this is the filleting feature, as shown in Figure 4.14(b).

6. Repeat steps 4-5 for e_4 and v_{o2}.

7. Return the set of the filleting features that were found.

Algorithm: Filleting Trace 2.

1. Input a toroidal face f, part P, and stock S.

2. **Determine orientations:** For a toroidal face f, the orientation of the feature is the axis of the torus, $v_o = \text{axis}(f)$. Add the bottom edge e_1 of face f to the set of edges in the edge profile E.

3. Verify that the radius of the surface f is within the cutting radius bounds for available filleting tools ($R_{\min} < \text{len}(e_2) < R_{\max}$) and its profile is $\frac{1}{4}$ of a circular arc.

4. **Determine profile:** For the possible orientation v_o, let s_{v_o} be the plane passing though the edge e_1 and perpendicular to v_o. Check edges e_i of Δ that lie in the plane s to see if they belong to the edge profile of the feature by asking:

   (a) Does e_i share a vertex with either of the edges at the ends of the profile E?

   (b) Is e_i adjacent to a face that can be made with the same filleting operation? Test the condition in step 2 of the algorithm for filleting trace 1 and the conditions in 3 (above).

   If the answer to both of these is yes, then add e_i to the edge profile E. Repeat these steps until candidate edges are exhausted.

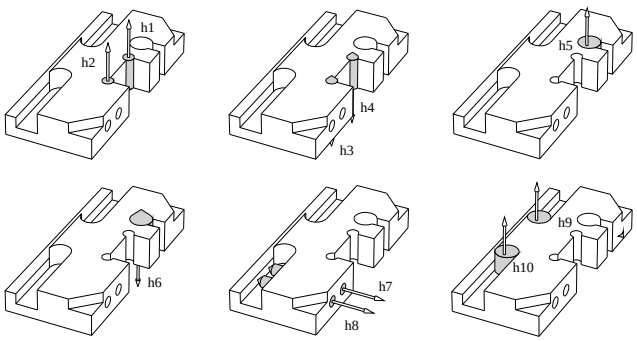


Figure 4.15: Drilling features for the part shown in Figure 4.10.

5. **Construct feature instances:** Build the profile of the tool and sweep it along the edges in the edge profile E. Unite the volumes created by sweeping along each of the segments of the profile and return the result.

6. Return the set of filleting features that were found.

4.4 Summary

In this chapter we developed a specification for the feature recognition problem and presented techniques for trace-based recognition of well-behaved primary machining features. This work consists of:

1. An enumeration of traces for the feature types in Chapter 3;

2. Presentation of the control structure for the general feature recognition algorithm;

3. Development of individual feature recognition algorithms based on the feature traces.

It is important to point out that these traces and functions are not presented as the optimal way of recognizing this particular class of features, but rather as one specific method. More importantly, this chapter presents an general algorithmic framework for the recognition problem—providing a structure that can be extended to include additional feature types and enhanced by adding new traces.

The effect of the algorithm RECOGNIZE_WELL-BEHAVED_FEATURES is shown in Figures 4.15 and 4.16. For the part shown in Figure 4.10, there exist traces to generate the features in the figures. In this case, the well-behaved primary feature set is

$$\mathcal{F} = \{p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14, p15, \\ h1, h2, h3, h4, h5, h6, h7, h8, h9, h10\}.$$

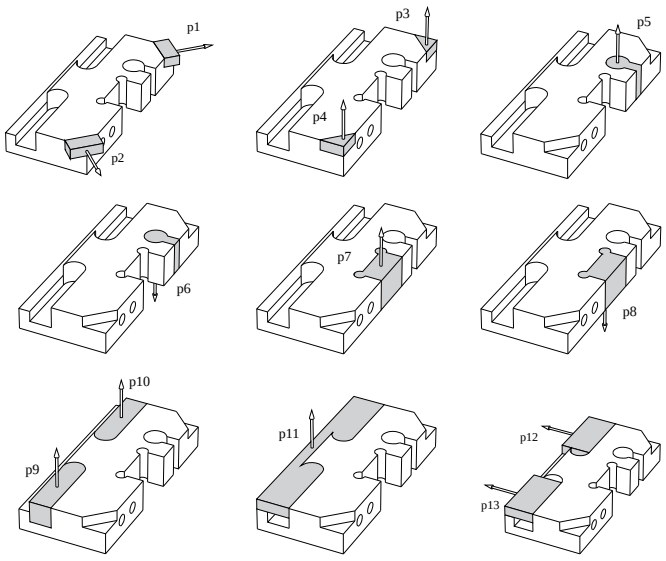


Figure 4.16: End-milling features identified for the part shown in Figure 4.10.

Chapter 5

Measuring Computational Complexity

This chapter addresses issues of computational complexity. Specifically, we will calculate the complexity of the algorithms presented in Chapter 4.

5.1 A Brief Background

The bibliography for complexity results that are specific to solid modeling is not extensive. Even when confined to boundary representation solid modeling, algorithms and data structures vary greatly among different systems. Existing work contains empirical and some theoretical analysis of the time and space costs for various data structures for representing solids.

Woo [209] analyzes several types of boundary representation data structures and compares their time costs for a set of primitive operations and space requirements. Weiler [207] presents data structures for curved surfaces and their time and storage complexities. Ala [4, 5] builds on this work and introduces variations on the boundary representation data structure with advantages for certain applications. Extensions to face-based representations are introduced in [40] and algorithms for their manipulation are analyzed.

Hoffmann [81] is an excellent source of worst-case complexity analysis for boolean operations on boundary data structures. In the feature recognition literature, De Floriani [39] presents an analysis of the complexity of her methodology. Peters [139] illustrates some of the combinatorial difficulties inherent in many graph-theoretic approaches to the feature recognition problem. Fields and Anderson [55] present a linear-time recognition algorithms for a variety of surface features.

Other attempts to measure performance include timing results, most notably in [10, 109, 38]. Results of this type are highly dependent on the hardware, the software implementation, the domain of interest, and the particular test cases chosen for the timing tests. Further complicating matters is the fact that in many cases the feature recognition problems addressed vary greatly. For example, most approaches for machining feature recognition do not perform an evaluation of feature accessibility for machining, Vandenbrande's [200] does and it incurs a cost to do so. Hence, plain timing results represent a weak basis for comparison between feature recognition methodologies.

5.2 Approach

The algorithms presented in the previous chapter are defined in terms of calls to a solid modeling system. Measuring the complexity of such algorithms poses several difficulties. Calculating the computational complexity in this case can be approached several ways:

1. **Data Structures Level:** The feature recognition problem can be defined in terms of a specific data structure and a set of algorithms on the data structure. For example, certain approaches define features as graph patterns to be found as subgraphs of a larger graph that represents the structure of the part.

2. **Application Level:** The feature recognition problem can be defined at an application level, where the functionality is specified in terms of specific types of data and operations on it without specifying in detail how the operations are going to be carried out—for example, sorting a list of solid models based on the volume of space that they occupy. An algorithm might be proposed which runs in $O(n^2)$ time, factoring out the cost of calls to a mass property procedure that calculates the volume of individual solids. In this case the details of how the mass property routine has been implemented will be specific to the solid modeler and not part of the specification of the sorting problem.

For approaches based on data structures abstracted from a solid model of a part, such as graph-based methods [39, 91, 30, 31, 34, 32, 33], the computational cost is most easily calculated using the first method: counting the number of operations on the data structures. Peters [139] uses a similar approach to compute abstract complexity bounds on instances of the feature recognition problem itself.

Other approaches to feature recognition, including the one described in this thesis, employ extensive queries to the solid modeling system to reason about geometry and to extract feature instances. In this case, the complexity of feature recognition algorithms depends on the cost incurred by executing boolean operations, sweeps, and queries to the solid modeler—and this in turn depends on many implementation-specific details such as data representation and overhead costs of the solid modeler.

What is evident is that application-level solid modeling algorithms are difficult to analyze using traditional algorithmic complexity measurement techniques. One fundamental reason for this is that the solid modeling data structures and algorithms operate at a much lower level than the problem itself. In order to address these issues and obtain reasonable metrics for measuring the complexity of these algorithms at the application level, we make the following observations:

- There is no authoritative reference on the general complexity of solid modeling operations (such as booleans, sweeps, and the like) for boundary-representation data structures. The complexity of these operations appear to lie between $O(n^2)$ and $O(n^4)$ or $O(n^5)$ time, depending on the particular configuration of geometric entities and many

implementation-specific details. Therefore, precise measurement of the complexity of these algorithms at the data structure level is not possible.

- The consensus is that solid modeling operations account for the vast majority of the computational cost during feature recognition [200]. Hence, measuring the number of solid modeling operations (and building algorithms that reduce the total number of solid modeling queries) is a reasonable and useful objective.

- One can choose to treat the solid modeling operations (in the case of the ACIS Solid Modeler [183], these operations are calls to the application protocol interface (API)) as taking $O(c)$ time, for some constant c. One can then measure the complexity of the feature recognition algorithms in terms of the number of solid modeling operations required; after which one can factor in their cost (somewhere between $O(n^2)$ and $O(n^5)$). Because functionality and APIs vary among existing systems, where one draws the line between the work of the recognition algorithms and the activity occurring within the modeler is somewhat arbitrary. In the analysis presented below, the modeler supports all the numeric activities as well as manipulation and interrogation: "Return the list of edges in an entity," "Does e bound face f?" "Does feature m_1 intersect feature m_2," etc.

The remainder of this chapter presents the theoretical complexity results for the algorithms presented in Chapter 4 with respect to the number of calls to solid modeling operations, using the above approach. It should be noted that, in measuring complexity in this way, we are developing rough upper bounds for the feature recognition problem. In most cases these complexities will apply only to involved operations; e.g., subtracting a simple cylinder from a rectangular block is unlikely to take $O(n^4)$ time.

5.3 Theoretical Results

Given that the part P and stock S are represented by some boundary data structure, there are several parameters we shall use in calculating the complexity of our trace-based feature recognition algorithms:

1. n, the complexity boundary models for the part and the stock.

   In the scope of this thesis, n is roughly equivalent to the size of the data structure representing the delta volume. In general for edge-based boundary representation data structures [207, 209], n will be $n = O(|E|)$ where E is the set of edges of the solid. To calculate a rough upper bound on the worst case, we can say the size is $O(n)$ where $n = E + V + F + 2G$ and E, V, and F are the number of edges, vertices, and faces of part, and G is its genus. Using the Euler-Poincaré formula, $0 = V - E + F - 2(1 - G)$, we can simplify this to $n = 2 + 2E$; hence $n = O(E)$. It should be noted that the Euler-Poincaré formula pertains to polyhedra, hence in this analysis $n = O(E)$ represents only a very rough upper bound on the class of solids discussed in this thesis.

Attaining a better upper bound would require more specific knowledge of the types of data structures used to represent analytic surfaces and of the algorithms used by the particular system to manipulate them.

2. $|\mathcal{M}|$, the number of different feature types to be recognized.

   Note that $|\mathcal{M}|$ is a constant when $\mathcal{M}$ is a fixed class of features (i.e., when there is no provision for the addition of user-defined feature types). As the problem is specified in Chapter 4, the class of features is not a parameter of the recognition procedure.

3. $n_{\text{traces}} = \max_{M \in \mathcal{M}}(|\{t_{M_1}, t_{M_2}, \ldots t_{M_k}\}|)$, the maximum number of traces for any one feature type.

   We will assume that each feature subclass has a fixed number of traces associated with it from which to reconstruct feature instances. Hence this value will be a constant.

4. $|T|$, the size of the set T of unique traces that can be generated from the part, stock, and feature type information.

   $|T|$ is a function of the number of the feature types, their traces, and the complexity of the part, stock, and delta volume.

5. $O(\tau(n))$ where $\tau(n) = \max_{\forall t_{M_i}}(O(t(n)))$, where $t(n)$ is the worst-case complexity of the procedure $\mathcal{P}_{t_{M_i}}()$. $O(\tau(n))$ is the maximal worst case complexity of the algorithms that build feature instances from traces.

   The value for $\tau(n)$ can be determined by calculating the complexity of each of the algorithms that construct feature instances from individual traces.

With the above parameters, one can calculate the complexity of a feature recognition system. Generally, on a single processor, the formula for complexity based on these parameters is:

$$O(|\mathcal{M}| \cdot n_{\text{traces}} \cdot |T| \cdot \tau(n)).$$

Because $|\mathcal{M}|$ and n_{traces} are constants, this can be simplified to

$$O(c \cdot |T| \cdot \tau(n))$$

for some constant c.

At question are the values for $|T|$ and the function $\tau(n)$. Specific values will depend on the traces used, the implementation of the algorithms to build feature instances out of traces, and the cost of executing solid modeling operations to do the necessary geometric reasoning. In the following section, we present a brief analysis of the complexity of the recognition algorithms given in Chapter 4.

5.3.1 Algorithm Analysis

There are two levels of algorithms presented in Chapter 4. The control algorithm RECOGNIZE_WELL-BEHAVED_FEATURES examines of each of the $|T|$ traces, and calls the feature recognition algorithms for each trace when appropriate. As this section presents, the analysis of the complexity of these algorithms with respect to the number of calls to a solid modeler is straightforward.

Analysis of RECOGNIZE_WELL-BEHAVED_FEATURES.
RECOGNIZE_WELL-BEHAVED_FEATURES traverses the information in the solid models of the part P and stock S in a search for traces. Hence, the complexity of the algorithm is $O(|T|)$. In the worst case each geometric and topological entity in P or S (or $S -^* P$) can belong to each trace for every feature. Based on the traces presented earlier we know the number of traces is roughly equal to the number of geometric and topological entities; hence $O(n)$ possible traces might arise from a specific P and S. Hence $|T| \in O(|\mathcal{M}| \cdot n_{\text{traces}} \cdot n)$. Factoring out the constants, $|T| \in O(n)$.

Analysis of Algorithm: Drilling Trace 1. Walking through the algorithm of Section 4.3.1, steps 1-3 consist of informational queries to the modeler. Step 4 involves building a primitive and performing an intersection test. Depending on the results of this test, step 5 determines a suitable location for the drilling feature using a finite set of queries to the modeler. Lastly, any features found (at most 2) are truncated to the size of stock—again through a fixed set of modeler operations. The algorithm for drilling trace 1 is purely sequential and contains no loops. Hence, it is $O(c)$ for a constant c, because the algorithm makes a fixed number of calls to the solid modeler.

Analysis of Algorithm: Drilling Trace 2. The complexity of this algorithm is also $O(c)$ for a constant c for the same reasons as the algorithm for drilling trace 1. The algorithm makes a fixed number of information gathering queries to the modeler (steps 1-3) then executes a fixed number of entity creation routines and boolean operations.

Analysis of Algorithm: Milling Trace 1. The algorithm for milling trace 1 of Section 4.3.2 is more intricate than the algorithms for either of the drilling traces. Steps 2 consists of a **for** loop considering all of the additional edges in the delta volume and of complexity $O(n)$. In the body of the loop, steps (a) and (b) consist of simple queries and entity creation routines. Step (c) performs projections to obtain candidate milling feature edge profiles. Step (d) takes the profiles and builds feature instances from them. In each of the steps a finite number of calls is made to routines for booleans, projections, and sweeps. While projections and sweep routines in particular can be intricate (of the order of $\Omega(n^2)$), the algorithm for milling trace 1 still makes only a fixed number of calls to a modeler each time it passes through the **for** loop—hence it is order of $O(c \cdot n)$ for a constant c.

Analysis of Algorithm: Milling Trace 2. The algorithm for milling trace 2 of Section 4.3.2 operates similar to that for milling trace 1.

In particular the algorithm consists of a **for** loop considering all of the additional planar faces in the delta-volume—this loop is of complexity $O(n)$. The body of the loop consists of simple queries and entity creation routines. The profile, as with the algorithm for milling trace 1, is calculated with projections. In each of the steps a finite number of calls is made to routines for booleans, projections, and sweeps. As in the case of the algorithm for milling trace 1, only a fixed number of calls is made to a modeler each time is passes through the **for** loop—hence it is order of $O(c \cdot n)$ for a constant c.

Analysis of Algorithm: Milling Trace 3. The algorithm for milling trace 3 presents a simpler situation in which the trace information does not require a traversal of the part and stock geometry and topology. In particular, proceeding from a single curved surface, a fixed number of operations is made to determine the orientation and profile of a through feature capable of creating the trace's curved surface. As with the algorithms for drilling features, this algorithm is in $O(c)$ for a constant c.

Analysis of Algorithm: Classification of Milling Features. The algorithm for classification of milling features, called once for each milling feature produced, examines each edge of the edge profile of the feature. At each edge it performs some fixed number of computations (face sweeps and booleans) to determine if a given edge is open or closed. This algorithm is then $O(n')$, where n' is the number of edges in the profile; because $n' < n$, this can be simplified as $O(n)$.

Analysis of Algorithms for Chamfering Traces 1 and 2. The algorithms for chamfering traces 1 and 2 (as well as those for filleting traces) are nearly the same in structure. Steps 1-3 perform queries to determine if the trace is within the bounds that can be considered as a chamfer and to calculate the orientation of the features that might have created it. The complexity in these algorithms occurs during the determinations of the edge profile. To build the edge profile, one needs to consider the edges formed by a slice of the part. As noted, this can be done as a depth-first search on the connected set of edges created by this slicing—at worst case $O(n)$ edges. For each of these edges, a fixed number of operations is called to see if it can be considered as part of the chamfer's profile and, if so, to construct the parameterized chamfer feature. This **for** loop is called at most twice; hence the overall complexity of the chamfer algorithms is $O(2 \cdot n)$ or just $O(n)$.

Analysis of Algorithms for Filleting Traces 1 and 2. The filleting algorithms operate are similar to those for chamfering traces 1 and 2. Their worst-case complexity is also $O(n)$.

Given the complexities for the trace algorithms, we know that $\tau(n) \in O(n)$. Thus, using the formula given above, the overall complexity of the algorithm RECOGNIZE_WELL-BEHAVED_FEATURES is $O(|T| \cdot n)$, or simply $O(n^2)$. Note that this is what we would

expect based on the argument in Section 4.1 that each individual geometric or topological element of the delta volume's boundary representation can belong to at most $O(n^2)$ well-behaved features. The algorithm RECOGNIZE_WELL-BEHAVED_FEATURES considers each of these $|T|$ traces and, for each trace, a constant number of well-behaved feature instances is constructed. To build each feature instance might itself involve the consideration of other geometry and topology data, at a cost of $O(n)$, for an overall complexity on the order of $O(n^2)$.

Factoring in the cost of the solid modeling operations, the overall complexity of RECOGNIZE_WELL-BEHAVED_FEATURES is between $O(n^4)$ and $O(n^6)$ or $O(n^7)$.

5.3.2 Additional Computational Issues

The above results are for the feature recognition algorithms only. Integration with downstream applications often implies the generation of alternative feature-based models and manufacturing plans. This search through the space of alternative feature-based models, as pointed out by [74], is inherently exponential. This is related to Definition 2 for the feature recognition problem as given in Chapter 4.1.

5.4 Summary

This chapter outlined a general formula for calculating the complexity of the feature recognition problem and applied it to the feature recognition algorithms in Chapter 4.

Chapter 6

Completeness

One of the fundamental problems in solid modeling is ensuring the accuracy and completeness of geometric representations [116]. Traditionally, geometric information was conveyed using line drawing and graphical models. Such representations prove quite adequate when human beings are the sole interpreters of the data but often fall very short of the requirements for machine interpretation. In this context, **completeness** refers to a representation's ability to unambiguously convey sufficient information about an object for calculating answers to arbitrary geometric questions.

This chapter builds an analogous notion of **completeness** for feature-based modeling and feature recognition. In this chapter we argue that the algorithms presented in Chapter 4 are complete over the class of well-behaved features (as defined in Chapter 3).

6.1 Motivations and Computational Issues

It has been pointed out by Marefat [117, 118] that existing feature recognition methodologies have had only limited success in identifying and describing alternative feature interpretations. There are a variety of reasons for this shortcoming. One reason is that since features can intersect with each other, the introduction of a new feature into a design can divide other features into spatially disjoint components—components which may be computationally expensive to identify and recombine. This poses difficulties in feature recognition: rule-based methods must capture all geometric situations that arise from the choice of feature hints and the ambiguities inherent in manipulating multiple interpretations with many separate rules; graph-based algorithms must syntactically or structurally capture these complexities.

A fundamental problem appears to be the lack of a systematic means to describe the appropriate set of feature instances to recognize. Without an a priori definition for **which feature instances** to recognize, the criteria for selecting which instances to generate are typically ad hoc heuristics based on local and incomplete information.

For example, in most approaches to feature recognition the particular set of features recognized is a byproduct of the implementation of the system. For some of the decomposition approaches, the features are primitive cells or combinations thereof. How specific cells are used will depend on the implementation and the geometry of the given part. For

knowledge-based approaches, the behavior of the system is embedded in the rules for completing features from the traces left in the CAD model. In this case, the feature set depends on the rules and their interactions in the reasoning system; rules must continually be added to handle each unforeseen special case.

6.2 Defining Completeness

Assessment of how well a feature recognition system addresses these problems can be accomplished by asking whether the system is **complete**. Intuitively, **completeness** refers to the ability of a system to produce all features appearing in a specific, well-defined class of feature instances. If a system produces all features in a given class $\mathcal{C}$, then we say that the system is **complete** over $\mathcal{C}$.

In the existing literature, there have been several efforts at guaranteeing completeness. The feature algebra of Karinthi [94], starting from a single initial feature interpretation, exhaustively generates alternative interpretations of the part by manipulating the features with algebraic operators, but does not include a methodology for recognizing the features. Sakurai [160] presents a system that decomposes the volume to be machined into disjoint cells and then recombines them to form compound feature instances. This method is complete over the class of features that can be built from compositions of these primitive cells. In another effort, Marefat [117] states that his hypothesis testing approach is complete over his class of hypothesis generators for features. All of these methods are prone to combinatorial obstacles and are limited to polyhedral models.

It should be noted that in existing systems completeness has not been addressed in terms of any factors that directly relate to manufacturing. In these cases, completeness is determined with respect to criteria that are artifacts of the computational approach chosen for recognizing the features.

6.2.1 Approach

In order to address this problem, we will define a feature recognition algorithm to be $\mathcal{C}$-**complete** over some class of features $\mathcal{C}$ if it produces, for any given part P and stock S, the set of all features in $\mathcal{C}$ that might be used in the manufacture of P.

A full formal proof of completeness of a feature recognition procedure would require:

1. A specification for the feature recognition problem, such as the one given in Chapter 4. This definition expresses the input set and output set of a feature recognition "black box."

2. Given a specification for the problem, one must determine whether or not the problem is **computable** [82]; i.e., one must show there exists an algorithm such that for every instance of the problem it halts and returns the correct answer. Proving computability, in a strictly formal sense, involves showing that the problem (feature recognition) can be coded into a Turing machine (in order to show the existence of such an algorithm).

In most domains, such a proof is a practical impossibility. It is widely accepted that if an the existence of an algorithm (stated in a programing language) can be shown, then the problem is computable.

3. To demonstrate an encoding into a Turing machine (or into a structured programming language at a reasonable level of detail), more mathematically rigorous definitions of features are required. In addition, one needs to map the problem from a continuous domain to a discrete domain. This means that a discrete version of the geometric problem must be developed in which solid models and feature instances are parameterized from a countable set.

With respect to (2) and (3), the feature recognition problem bears some resemblance to the problem of tiling the plane [100, 195, 14, 150]. Tiling describes the problem of asking, given templates for 2D tiles, whether the plane (or some subset of it) can be covered with instances of tiles. There are several computability results in the area of 2D tilings that have employed mappings of a geometric tiling problem to a Turing machine. A similar approach would have to be adopted to obtain a full and formal proof of completeness.

It is evident that for practical problem domains the mathematical tools for performing a full and formal proof of completeness do not exist. This has these immediate consequences:

- It is impossible for a feature recognition algorithm to be complete over the set of all valid machining features $\mathcal{V}$ because, as noted in Chapter 4, there are parts for which there are infinitely many possible instances of machining features.

- Even if we restrict ourselves to primary features, completeness is still impossible: there are simple machinable parts that can have infinitely many primary features (as illustrated earlier in Figure 4.2).

- There are only finitely (in fact, polynomially) many well-behaved primary features for any given part, as proven in Section 4.1. Thus completeness over the set of well-behaved features is an attainable goal.

Within the context of this thesis, an argument for the completeness of the algorithms follows from the fact that the feature traces are derived from the properties of the well-behaved features. In particular, if $\mathcal{W}$ is the class of well-behaved feature an argument that the procedure RECOGNIZE_WELL-BEHAVED_FEATURES from Section 4.3 is $\mathcal{W}$-complete over the class of well-behaved features can be made as follows: given a part P, a piece of stock S, and a well-behaved feature instance f, we know that f contributes unique geometric and topologic characteristics to the boundary of P. RECOGNIZE_WELL-BEHAVED_FEATURES searches for these contributions and reconstructs the feature f (or some equivalent feature). For each geometric and topological attribute of P, all well-behaved features capable of producing that entity are eventually produced.

More specifically, with regard to the well-behaved features from Chapter 3, there are three possible cases:

1. f **is a drilling feature.** The boundary of the delta volume, $b(\Delta)$, must contain a portion of f's cylindrical side surface or ending surface. An orientation for potential drilling features can be obtained from the axes of these surfaces; the surface information can also determine the feature profiles (in this case the radii of the surfaces). This orientation and the profile fix the other defining feature parameters (depth and location).

   The algorithms for drilling traces 1 and 2 described in Section 4.3 represent an implementation of this case.

2. f **is a milling feature.** Since f is well-behaved, $b(f) \cap b(\Delta)$ contains surface or edge information from f or from a feature f' that is subsumed by feature f. Hence, $b(f) \cap b(\Delta)$ contains surfaces and/or edges in a plane perpendicular to the orientation of the feature f. This plane contains the bottom face of feature f or of a feature f'. In the second case, the f' is used to determine the location of the bottom surface of f. Given the location of the bottom surface, one can calculate the edge profile, location, and depth for the feature f.

   The algorithms for milling traces described in Section 4.3 represent an implementation of this case.

3. f **is a chamfering or filleting feature.** The edges of f's edge profile are edges in $b(\Delta)$ and $b(f) \cap^* b(\Delta)$ is non-empty. An orientation for f is calculated by examining the edges and faces in $b(\Delta)$. Using edge profile E, the other surfaces made by the machining operation are found by examining the part topology—this information can be used to build one or more volumetric feature instances capable of creating $b(f) \cap^* b(\Delta)$.

   The algorithms for chamfering and filleting traces described in Section 4.3 represent an implementation of this case.

While the techniques used in this thesis to develop the algorithms of Chapter 4 are closely tied to the specification of the set of well-behaved features, this situation need not generalize: for example, one might pose the question of the completeness of an algorithm $\mathcal{A}$ across a number of different classes of features, $\mathcal{C}_1, \mathcal{C}_2, \ldots \mathcal{C}_k$. Alternately, one might consider a number of different algorithms, $\mathcal{A}_1, \mathcal{A}_2, \ldots \mathcal{A}_k$, and their completeness over a specific class of features $\mathcal{C}$. In this way, a formalization of completeness can separate the specification of "which features to recognize" from the particular recognition algorithm or method employed. The argument for the completeness of this thesis' approach serves to illustrate the utility of the general concept of completeness as developed in this chapter.

6.2.2 Feature Interactions

The ability to recognize interacting features has been a stated goal of a number research efforts, among them [41, 58, 117, 91, 200]. While it is agreed upon as a problem of critical importance [210], however, the concept of **feature interactions** lies largely undefined in

the general literature. In most cases, the definition for the term is implied, vague, or specific to one particular approach.

Feature interactions pose a major challenge to the development of robust and reliable feature recognition systems. Given the previous arguments for completeness of the approach presented in this thesis, it should follow that the approach presented in this thesis will find all well-behaved features regardless of how they "interact." The question arises as to what "interact" means. We observe that there are three different classes of feature interactions:

1. **Geometry/topology-level feature interactions** occur when the feature instances share common entities if considered as volumes or surfaces. A geometry/topology-level interaction occurs, for example, when two or more volumetric features have a non-empty intersection or when two or more surface features share boundaries.

2. **Trace-level feature interactions** occur when one or more features $f_1 \ldots f_n$ interfere with the traces of another feature, f_0. For example, $f_1 \ldots f_n$ might distort or destroy information critical to the algorithmic reconstruction of f_0. In such interaction, there is no tractable way to deduce the existence of f_0 from the information remaining in the design (geometry, topology, design attributes, etc.). An example of such an interaction would be a milling feature that leaves only a single side face in the delta volume. In such a situation there is no way to determine the orientation of the feature from this side face. Another example is that of a drilled hole made with a ball-end mill which contributes only a portion of its spherical tip surface on the part. In this case, there are infinitely many possible orientations for drilling features that left this trace.

3. **Manufacturing plan-level feature interactions** occur when the manufacturing operations represented by the features affect one another during the generation and execution of a manufacturing plan. For example, in machining, precedences among the machining operations are affected by the setups for each feature. Machining of a feature might destroy a precondition for fixturing the part during machining of another feature. As another example, given that two features f_1 and f_2 intersect volumetrically, a planner has to make a choice about how to machine the shared volume. This kind of feature interaction can only be handled when manufacturing attributes (such as tolerances, surfaces finish, and the like) are considered along with the other alternative features available to the planner.

Existing work has addressed the interactions problem in a number of ways, usually touching on one or more of the above levels. In most cases, however, it is usually not made clear which type of feature interaction is intended.

The approach to feature recognition presented in this thesis is complete for features with arbitrary geometric and topological interactions. With regard to trace-level interactions, this approach recognizes all features for which there exists at least one trace.

Lastly, the feature recognition algorithms presented in Chapter 4 do not address the problem of plan-level interactions. Planning, cost estimation, and other downstream applications are independent of feature recognition as defined in this thesis. In these types of

interactions, a planner must make a decision—a decision that is based on domain knowledge about the planning problem being addressed. Hence, these types of interactions have to be addressed by the downstream applications whose planning activity is based on the primary well-behaved feature set, $\mathcal{F}$.

6.3 Related Issues: Correctness and Soundness

Researchers have recently raised the question of proving correctness and soundness of feature recognition algorithms and systems. In this section we address some of the issues and obstacles to attaining mathematically rigorous proofs for these properties. Many of the ideas from the discussion of completeness in the previous section can be applied to shed additional light on how one might prove soundness and correctness.

Motivations for building mathematical tools with which to reason about these issues are numerous. Computational robustness is chief among these, due to the often mission-critical nature of engineering analysis software. As engineers increasingly rely on computer-aided analysis tools and automated reasoning software, the potential for grave errors being introduced by faults within computer software (or with the human user's interpretation of the results produced by software) increases. This has been extensively documented in the mainstream as well as science media. For example, Petroski [140] describes some of the hazards caused by analysis software for civil engineering and a recent cover story in Scientific American [61] describes some of the significant obstacles that still need to be overcome in the design and implementation of large software systems.[1]

In the following two sections, we propose definitions for correctness and soundness of feature recognition systems and discuss how these issues can be addressed.

6.3.1 Correctness

A program $\boxed{\Pi}$ described by a structured programming language (such as Lisp, C/C++, Pascal, etc.) is basically a function mapping a set of input data to a set of output data. To prove correctness, one must show that the program computes a function that is equivalent to its specification. The proof of correctness consists of showing that the effect on the data by the basic commands in the language as they are listed in the code add up to the specification.

Formally, **program correctness** is defined as showing that a given program $\boxed{\Pi}$ is correct with respect to some functional specification f_Π. For purely illustrative purposes, Figure 6.1 presents an example program and Figure 6.2 the formal proof of its correctness using Mills' **functional semantics** method [59, 125, 127].

To briefly describe some of the notation in the figure:

$\circ$	function composition;
$\boxed{\Pi}$	the function defined by code fragment Π;

[1] The article focuses on the highly publicized baggage system at the Denver International Airport.

```
factorial ( integer m)
   s = 1;
   while m > 0 do
    {
      s = s * m;
      m = m - 1;
    }
   return s;
```

Figure 6.1: A program $\boxed{\Pi}$ for computing $m!$.

domain($\boxed{\Pi}$)	the functional domain of $\boxed{\Pi}$;
$\neg, \vee, \mid$	standard meanings from logic: not, and, or.

Readers interested in reading more on formal methods for program correctness (or verifying the proof in Figure 6.2) are referred to the extensive literature in this area [114, 126, 45, 59, 80, 125, 127].

This example is solely intended to show the complexity of generating formal proofs of program correctness; hence, we shall not present any description of the technique. Note that this example proof is for a relatively trivial program—proofs for programs as easy as the straightforward case of sorting a list of numbers may run for many tedious pages. Matters are further complicated by the existence of complex data structures, nested loops, and function calls. One might speculate that a full proof of correctness for a feature recognition system might require many thousands of pages.

Given the practical impossibility of proving correctness for algorithms at the scale of a feature recognition system, what can be done to ensure the correctness of such a complex software system? This thesis proposes the following approach:

1. Develop implementation-independent and algorithm-independent specifications for the feature recognition problem, as was done in Section 4.1. This exercise involves developing several layers of definitions including those for a class of features and those for which members of the class are to be recognized.

2. Systematically divide the problem into structured components, as was done in Section 4.2, whose cumulative effect conforms to the problem specification. This involves developing algorithms to solve each of the subproblems.

We would informally claim that with respect to the specification given in Section 4.1, the trace-based algorithms in Section 4.2 are correct. However, it appears that a formal proof of correctness is beyond the capabilities of existing mathematical tools. For an essential reference on the hazards of building large software systems, readers are referred to [92].

Theorem 6.3.1 *The program* Π *in 6.1 is correct with respect to functional specification* $f_\Pi = (m \leq 1 \Rightarrow s = m; m = m - 1) \mid (m > 1 \Rightarrow s = m!; m = 0)$

Proof: using Mills' functional semantics.
Step I: Determine $f_{\textbf{while}}$, a functional specification for what is computed by the while loop ($\boxed{\textbf{while} \ldots}$).

Let B be the code body of the while loop.

We know that $f_{\textbf{while}} = \boxed{\textbf{while} \ldots}$ if and only if:

1. $f_{\textbf{while}\ldots} = \boxed{\text{if } b > 0 \text{ then } B} \circ f_{\textbf{while}\ldots}$;
2. domain($f_{\textbf{while}\ldots}$) = domain($\boxed{\textbf{while} \ldots}$);
3. if $b \leq$, $f_{\text{while}} = ()$ (i.e., nil).

Suppose

$$f_{\textbf{while}} = (m > 0 \Rightarrow s = s * m!; m = 0) \mid (m \leq 0)$$

Therefore

$$\boxed{\text{if } m > 0 \text{ then B}} = (m > 0 \Rightarrow s = s * m; m = m - 1) \mid (m \leq 0 \Rightarrow ())$$

Step I (a): show

$$f_{\textbf{while}\ldots} = \boxed{\text{if } b > 0 \text{ then B}} \circ f_{\textbf{while}\ldots}$$

using truth tables,

Table 1:

$(m > 0 \Rightarrow s = s * m; m = m - 1) \vee$ $(m > 0 \Rightarrow s = s * m!; m = 0)$	m	s
$m > 0$	$m - 1$	$s * m$
$m - 1 > 0$	0	$s * m * (m - 1)!$
$m > 1$	0	$s * m!$

Table 2:

$(m \leq 0) \vee$ $(m \leq 0 \Rightarrow ())$	m	s
$m \leq 0$	—	—
$m \leq 0$	—	—
	()	()

Figure 6.2: A formal proof of correctness for the factorial program Π from Figure 6.1 using the **functional semantics** techniques of Mills [59, 125, 127] (the proof is continued on the next page).

Truth Table 3:

$(m \leq 0) \vee$ $(m > 0 \Rightarrow s = s * m!; m = 0)$	m s
$m \leq 0$ $m > 0$	
	cannot happen

Truth Table 4:

$(m > 0 \Rightarrow s = s * m; m = m - 1) \vee$ $(m \leq 0)$	m s
$m > 0$ $m - 1 \leq 0$	$m - 1$ $s * m$
$m = 1$	0 $s * m!$

Therefore,

$$\boxed{\text{if } m > 0 \text{ then B}} \circ f_{\mathbf{while}} =$$
$$(m > 1 \Rightarrow s = s * m!; m = 0) \mid$$
$$(m = 1 \Rightarrow s = s * m!; m = 0) \mid =$$
$$(m \leq 0 \Rightarrow ())$$
$$(m > 0 \Rightarrow s = s * m!; m = 0) \mid (m \leq 0) = f_{\mathbf{while}}$$

Step I (b): Note that domain($f_{\mathbf{while}\ldots}$) = domain($\boxed{\textbf{while } \ldots}$).
Step I (c): Note that if $b \leq$ then $f_{\text{while}} = ()$.
Step II: determine the function $\boxed{\text{II}}$ computed by the program II and check that it conforms to the specification.

$$\begin{aligned}
\boxed{\text{s=1; } \mathbf{while} \ldots} &= f_{\mathbf{while}}(s = 1) \\
&= (m > 1 \Rightarrow s = 1 * m!; m = 0) \mid \\
&\quad (m \leq 1 \Rightarrow s = (1) * m; m = m - 1) \\
&= (m > 1 \Rightarrow s = m!; m = 0) \mid \\
&\quad (m \leq 1 \Rightarrow s = m; m = m - 1) \\
&= \boxed{\text{II}}
\end{aligned}$$

Figure 6.2 continued.

6.3.2 Soundness

In general, a program is said to be **sound** if it never produces incorrect results. By extension, a feature recognition procedure can be defined as **sound** if it always produces valid features. Presented at this level, soundness seems like a highly desirable quality for a feature recognition system to possess.

In the context of feature recognition, however, developing a mathematical definition for soundness presents several issues:

1. What is meant by an valid feature? In a domain of machined parts there are many conditions under which a feature could be invalid. For example, any volumetric feature that intersects with the final part geometry is considered invalid. Including any such feature in a plan would result in overmachining of the part. However, development a mathematical definition for what it means to have a valid feature is unclear and highly dependent on the particular application domain for which one is recognizing features.

2. Given that validity is a term relative to the downstream application at hand, even within the context of individual applications validity is defined on a largely ad hoc basis. In machining, as pointed out by Gupta et al. [72], a natural way of classifying the features is to partition them into those that we consider valid for manufacturing planning and those that we consider invalid—i.e., those unlikely to occur in any reasonable manufacturing plan. Hence, defining validity requires formalizing the relationship between features and the manufacturing plans they can be used to generate.

3. In defining validity in terms of manufacturing plans for various downstream applications, the soundness of features becomes dependent on a number of complex factors. In the realm of machining, this means real-world constraints such as the availability of certain individual machine tools on a shop floor or the manufacturing schedule currently planned for the shop at hand. Machine tool downtime or conflicts have a measurable effect on the soundness of individual plans and features. Without better computational characterizations of these effects, developing a general notion of soundness that has utility for the developers of feature recognition systems will prove difficult.

We observe that formal proofs for soundness depend on the context in which the features are being used. The soundness of a feature recognition system for cost estimation may be very different if used for process planning. Further, formally proving soundness requires a better mathematical understanding of how features map to manufacturing operations and plans.

6.4 Summary

This chapter developed the notion of completeness for feature recognition systems. In particular, the argument for the completeness of this thesis' approach serves to illustrate how to apply the general concept of completeness (as developed in this chapter) to a specific

instance of the feature recognition problem. In this way, formalization of completeness can be used to separate the specification of "which features to recognize" from the recognition algorithm or method employed. These results were extrapolated to illustrate the difficulties inherent in formalizing analogous notions for correctness and soundness.

Chapter 7

Post-Recognition Processing

The application domain of machining poses a variety of constraints on the types of features that can be considered valid and useful. We can use knowledge from a number of sources to improve and enhance the feature recognition process. This chapter presents techniques to modify the set of well-behaved primary features $\mathcal{F}$ returned by the algorithms of Chapter 4 to improve their correspondence to actual machining operations and exclude those which are likely to be unrealistic for downstream applications.

7.1 Approach

Although the feature types defined earlier are intended to correspond to machining operations, specific feature instances can sometimes present unrealistic machining requirements. The approach described in this thesis separates, to the greatest practical degree, feature recognition from domain-specific manufacturing considerations. Therefore, such considerations are performed as post-processing steps on the basic well-behaved primary feature set that is produced by the recognition algorithms.

Given the RECOGNIZE_WELL-BEHAVED_FEATURES procedure from Chapter 4, we introduce here an additional algorithm:
BUILD_FEATURES(P, S)
INPUT: solid models of a part P and stock S
OUTPUT: a set of feature instances, $\mathcal{F}$.

1. $\mathcal{F}$ = RECOGNIZE_WELL-BEHAVED_FEATURES(P, S).
2. For each feature f in $\mathcal{F}$ do:
   (a) Perform a check on the tooling constraints; if f is outside the bounds of reasonable tooling parameters then remove it from $\mathcal{F}$.
   (b) Perform an inaccessibility check of f; if f is inaccessible, remove it from $\mathcal{F}$.
3. For each milling feature f in $\mathcal{F}$ do:

(a) Examine the edge profile of f to identify subclassification for milling feature (i.e., pocket-milling, face-milling, or step-milling).

(b) Determine if the feature f has bottom blends.

(c) Offset the edge profile of f.

(d) Perform an inaccessibility check of f;
if f is inaccessible, remove it from $\mathcal{F}$.

4. Return($\mathcal{F}$).

This algorithm adjusts the set of well-behaved primary features, $\mathcal{F}$, to account for tooling, accessibility, and the machinability of profiles. The remainder of this chapter defines these terms and describes techniques and algorithms for dealing with their effects on features in $\mathcal{F}$.

7.2 Tooling Constraints

Some features may violate constraints on the physical dimensions of available tooling. For example, not every cylindrical surface may be machinable with a drilling operation because there are physical limits on the maximum radius available in the available set of drilling tools. Similarly, bounds exist on the dimensions of surfaces identified as blends, fillets, or chamfers.

While the specific selection of tooling is a task for process planning or analysis, we can devise some simple general checks to identify and discard any obviously unrealistic features at the time of recognition. Note that in creating these tests, our only objective is to discard or modify features that present fundamentally unmachinable requirements. In the context of other application domains, one would want to devise similar tests based on the requirements of that particular domain.

The following is a list of machining constraints and how they are addressed within the framework presented in Chapters 3 and 4. While the parameters presented below are by no means exhaustive, they can be use to eliminate what are, with high probability, unrealistic feature instances. The values are based on a survey of common cutting tools as listed in cutting tool catalogs [48, 99, 164, 165, 194].

- For drilling features there are three parameters we consider:

  Flute length: Flute length refers to the length of the cutting surface of the tool. For the drilling tool in Figure 7.1 the flute length is l_4. We truncate any drilling features requiring tools with flute length greater than 128mm (approximately 5 inches).

  Tool diameter: The D parameter on the drilling tool in Figure 7.1. We discard any drilling feature requiring a tool diameter greater than 40mm.

Length/Diameter (L/D) ratio: For drilling tools such as the one shown in Figure 7.1, the L/D is the ratio of the length of the tool to its diameter—i.e., the ratio of D to l_4. For most drilling tools the maximum drilling depth is 3 to 5 times the diameter of the tool. As a post processing step we truncate drilling features with a L/D ratio greater than 6.

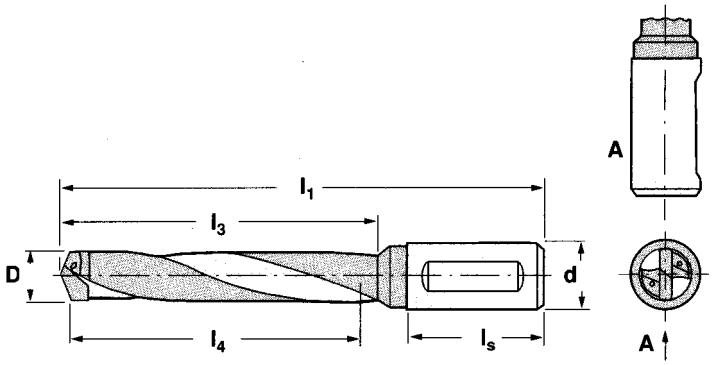


Figure 7.1: A drilling tool from [48].

- For milling features one parameter considered:

  Blend surfaces: Milling operations can leave a transition surface between the walls of the feature and the bottom surface of the feature. The size of these blend surfaces depends on the parameters of the available tool. We consider curved surfaces (cylinders, tori, and spheres) to be possible round blends if their diameter is between 3mm and 25mm. For flat blends, we consider only conical surfaces rectangular planar surfaces with width[1] between 2mm and 40mm. An algorithm for identifying bottom blends on milling features is given below.

- For chamfering and filleting features several parameters need to be considered, some of which were presented along with their recognition algorithms in Chapter 4. For chamfering features:

  Tip angle: The tip angle is the angle between the cutting edge of the tool and the tool's axis. In the context of this thesis, the tip angle is assumed to be 45°.

  Cutting length: The cutting length refers to the length of the cutting edge of the tool. In Figure 3.9(a), for tools with 45° tip angles, this length is $\frac{\sqrt{2}}{2} \cdot A$. In the algorithms below we will assume that there are minimum and maximum values for A, $A_{\min} = 3\text{mm}$ and, $A_{\max} = 35\text{mm}$.

[1] When speaking of a conical surface, width refers to the distance between the bounding edges at the top and base of the cone. For example, width would equal the distance between the vertex at the top of the cone and the cone's elliptical base, measured along the cone's surface.

For filleting features:

Cutting radius: The radius of the cutting surface of the tool, R as shown in Figure 3.9(b). We assume that there are minimum and maximum values for R, $R_{\min} = 3\text{mm}$, and $R_{\max} = 19\text{mm}$.

Cutting profile: The cutting profile refers to the shape and length of the cutting edge of the tool. Referring to the parameters in Figure 3.9(b), the algorithms below will assume that the cutting profile is $\frac{1}{4}$ of the circular arc of radius R, hence $2.35\text{mm} < R < 29.9\text{mm}$.

Both flute length and L/D ratio have a significant impact on how (and if) a feature can be machined. For example, milling features of excessive depth might not be machinable due to the lack of an available tool that is long enough and of the right diameter to machine the feature's contour and avoid excessive tool chatter.

Ideally, these parameters would be obtained from a manufacturing resource database that contains the cutting tools available in the context of the given application [93]. These parameters will vary greatly depending on the available set of cutting tools and machine tools, which in turn will vary greatly depending on the particular application. The interface of feature recognition with manufacturing resource information is beyond the scope of this thesis; hence we have opted to simplify the problem by selecting some common values. If necessary, the approach presented in this thesis could be extended to interface with different databases or could be fine-tuned to work with a specific collection of resources.

Determining Bottom Blends. Given bounds on the sizes of available tooling, the following algorithm takes a milling feature f and determines if there exists bottom blends. It does this by examining the surfaces of the part P adjacent to E for surfaces that fall within the parameter bounds to be considered as possible blends. In particular, if any blend surfaces are found, we adjust the profile E and modify the feature volume. Given a face b that is a bottom blend for a milling feature with profile E, an algorithm for adjusting E can be given as follows:

1. Input a part P, a milling feature f, and parameters $b_{\min}$ and $b_{\max}$ representing the minimum and maximum sizes for blend surfaces.

2. Let s be the planar face corresponding to the bottom surface of f, as defined by E's edges.

3. For each edge e in E and in the profiles of the islands of f do:

   (a) Let b be the face of P adjacent to e; if there is no such face, loop to (3).

   (b) Verify that b is a blend surface; if b is not a blend surface loop to (3). In the context of this thesis we consider only a fixed number of blends, therefore b is either a cylindrical, spherical, or toroidal surface with radius $b_{\min} < r < b_{\max}$; or b is a

planar or conical surface with width w (between bounding edges) $b_{\min} < w < b_{\max}$. For flat blends we will assume a 45° angled tool.

(c) Let p_b be the face formed by taking the projection of b onto the plane containing s.

(d) Let $s = p_b \cup^* s$ and let E' be the edges bounding s.

4. Return E'.

Note that in this thesis we consider only milling features with constant bottom blends; we do not consider milling features with either multiple (blends of mixed radii) or varying bottom blends. Further note that bottom blends are only possible for pocket-milling and step-milling features.

7.3 Offsetting

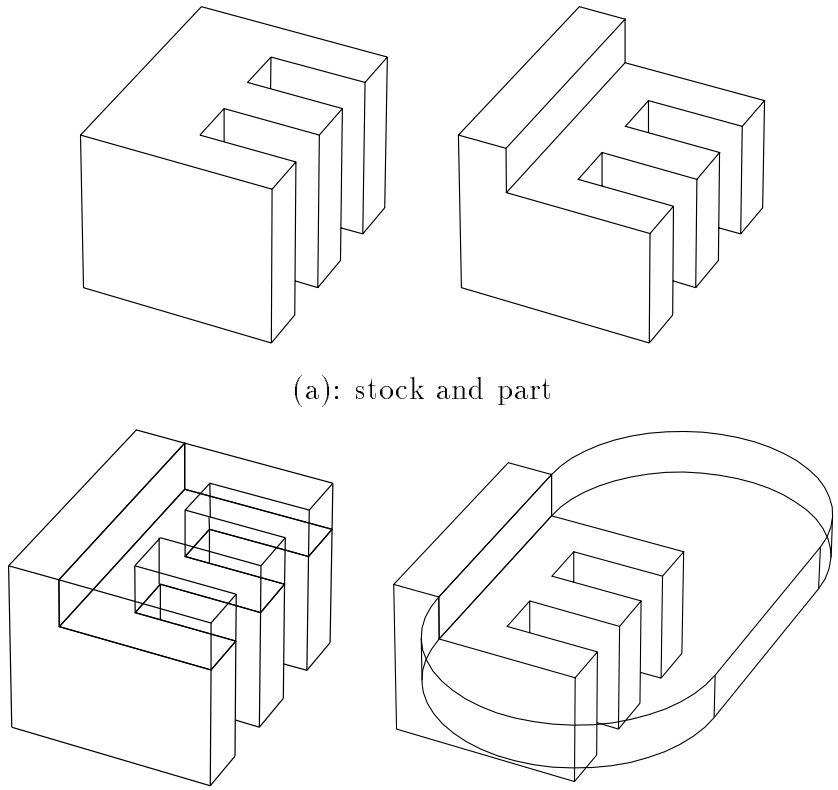
Offsetting is the process of deducing a milling feature's removal volume (rem(f)) from its effective removal volume (eff(f)). Determined solely from the part and stock, a milling feature's profile might contain sharp corners that cannot be machined by a milling tool; or it might be the case that the most cost-effective way to mill a volume is to perform the machining operation using the largest possible tool. Such situations might require the tool to move outside the boundary of the stock material.

After a possible milling profile has been identified, we will want to adjust its profile to provide an **offset feature,** such as that shown in Figure 7.2, that takes these machinability considerations into account. In the figure, the edges of profile E have been offset to take into account the radius of a cutting tool. An example of an offset step-milling feature is given in Figure 8.9(c).

Offsetting the edge profile of a potential milling feature involves the following steps:

1. **Estimation of an optimal tool size.** In a typical milling operation, a larger tool diameter implies a shorter cutting trajectory and less operation cost and time. However, a variety of constraints resulting from the geometric configuration of the profile might restrict the maximum tool size that can be employed. In this step, the geometry of the profile is used to calculate an upper bound on the tool size. For this, we employ the algorithms developed by Gupta [70].

2. **Alteration of the profile.** In some profiles, the estimation of tool size might reveal machinability problems. For example, two adjacent closed profile edges meeting at a convex corner result in a tool radius estimate of zero; a narrow distance between closed edges in the profile might return an estimate smaller than the smallest available tool. This step modifies profiles by offsetting convex corners inward to account for the corner radius left by a tool (shown for the closed edges in Figures 7.3(b) and (c)) or by dividing an otherwise unmachinable profile into a set of multiple profiles that can be

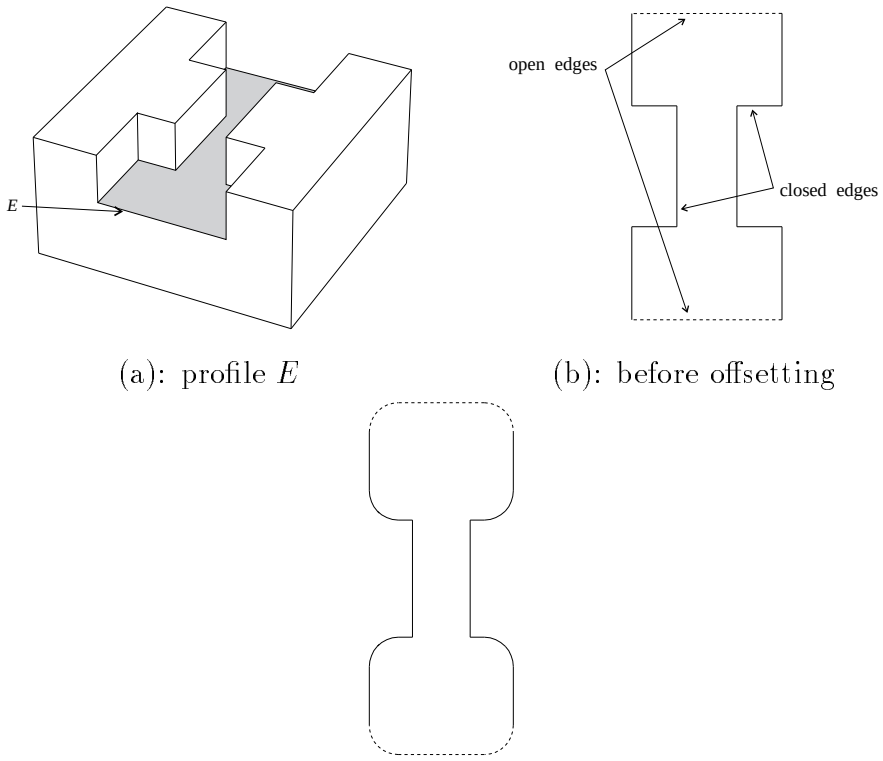


(a): stock and part

(b): feature

(c): offset feature

Figure 7.2: Offsetting to produce more realistic machining volumes.



(a): profile E (b): before offsetting

(c): after offsetting

Figure 7.3: An example of edge profile offsetting.

machined with the available milling tools. An algorithm for profile alteration is given below.

3. **Offsetting the profile.** After finding a bound on the tool size, the open edges of a milling feature are offset to account for the radius of the milling tool, as shown for the open edges in Figures 7.3(b) and (c). The tool can move on or outside these edges during machining. Again, we have used algorithms developed by Gupta [70].

7.3.1 Profile Alteration

In some profiles, the estimation of tool size might reveal machinability problems. For example, two adjacent closed profile edges meeting at a convex corner result in a tool radius estimate of zero; a narrow distance between closed edges in the profile might return an estimate smaller than the smallest available tool. This step modifies profiles by offsetting convex corners inward to account for the corner radius left by a tool (as shown for the closed

edges in Figures 7.3(b) and (c)) or by dividing an otherwise unmachinable profile into a set of multiple profiles that can be machined with the available milling tools.

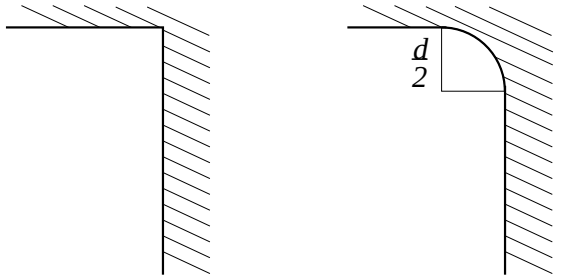
To address these situations we will want to modify the profile to conform to the machining constraints. This involves developing an algorithm that offsets convex corners to account for the corner radius and divides profiles into multiple profiles that can be machined with the given tooling. This algorithm assumes that the milling feature profile is composed of straight lines and circular arcs.

Algorithm: Modify Milling Profile.

1. Input a milling feature f with edge profile E, part P, and tool diameter d.
2. For every pair of consecutive closed edges in E meeting at a convex corner introduce a corner radius of $\frac{1}{2}d$, as shown in Figure 7.4.
3. For each closed edge e_i of the set of edges E do:
   (a) Let v be the vector pointing from edge e_i into the interior of the profile defined by E. Build the tool sweep area A as shown in Figure 7.6(a).
   (b) If A intersects with another closed edge e_j as shown in Figure 7.6(b) (for simplicity, we assume $i < j$) then
      i. Extend edges e_i and e_j to their intersection and introduce a corner radius (based on their vectors v_{e_i} and v_{e_j}) as in Step (2). This creates one new closed edge profile $e_i \ldots e_j$ and leaves one open edge profile $e_{j+1} \ldots e_{i-1}$.
      ii. With the remaining edges, join edges e_{j+1} and e_{i-1} with an open edge to close the profile.
4. Return the set of edge profiles created.

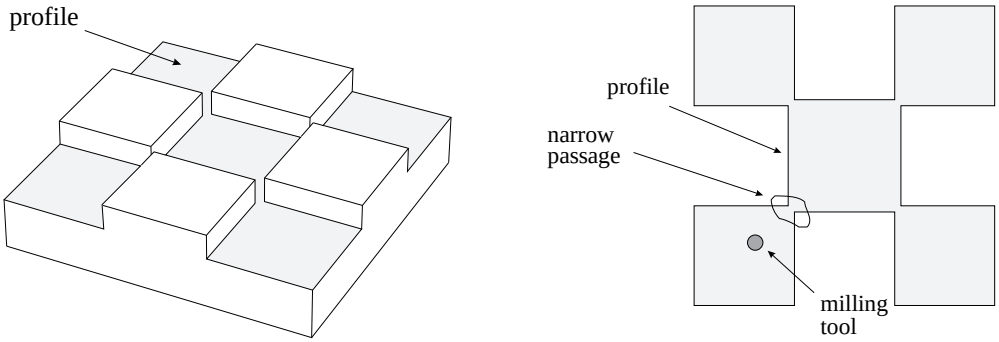
An example of the 5 new profiles that would be produced by this algorithm for the part pictured in Figure 7.5 is shown in Figure 7.7.

This algorithm is designed to produce improved profiles for milling where the original profile was undesirable or unmanufacturable. In practice an algorithm of this kind will have to include some form of optimization criteria. There are numerous trade-offs between selection of tool size, cutting speeds, wear, etc., not to mention the fact that the features represented by these profiles must be sequenced for machining during an operation plan. This algorithm does not address these latter concerns.



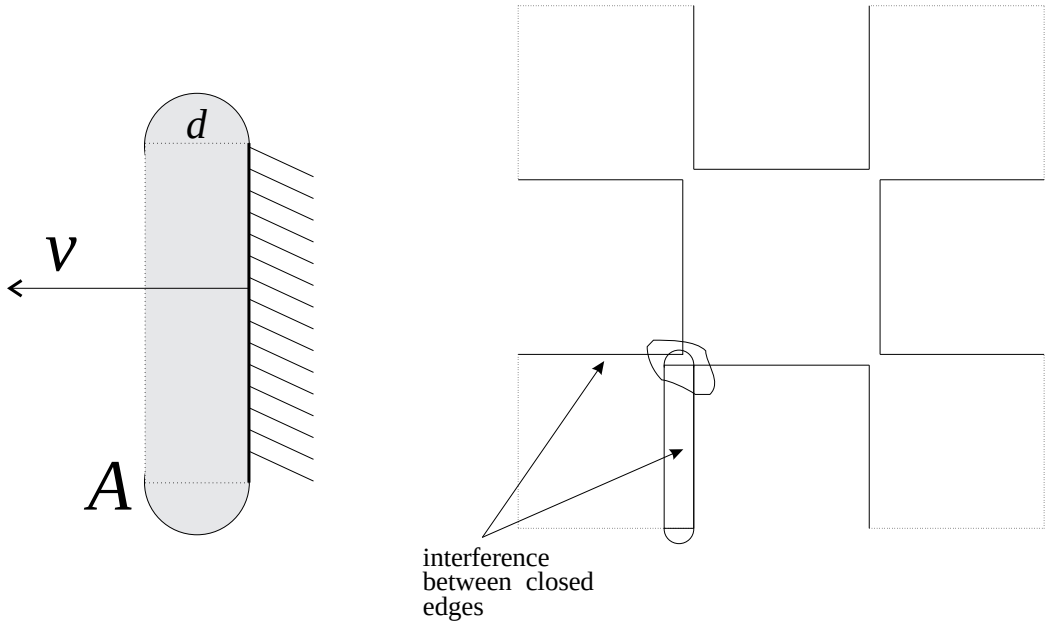
(a): Before offsetting (b): After offsetting

Figure 7.4: The effect of feature profile offsetting.



(a): Recognized profile (b): Tooling difficulty

Figure 7.5: An example of a problem profile for which the desired tool radius is too large to machine the entire feature.



(a): Build tool sweep area (b): Interference from Figure 7.5

Figure 7.6: Example of testing closed edges for interference.

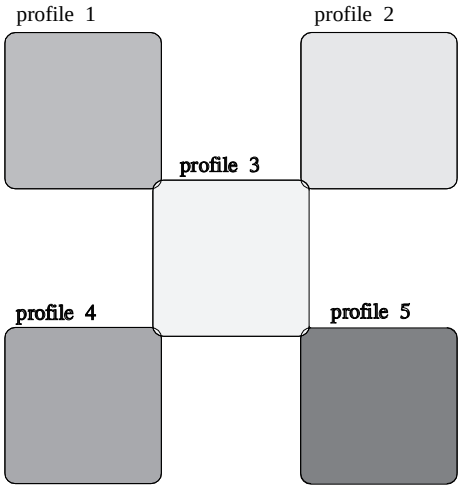


Figure 7.7: Altered profiles from example 7.5.

7.4 Accessibility

Accessibility is a very complicated property to verify. It depends on the shape and dimensions of the machine tool and the cutting tool, and the order in which the features are machined—all of which are not decided until a machining plan is generated. Development of a general methodology for determining whether a feature f is accessible would require generating all of the alternative operation sequences that employ the feature f, to see if f is accessible in any one of them. Such algorithms have been developed in the context of generating and evaluating alternative machining plans [68, 67, 70], but are beyond the scope of feature recognition per se.

In this section we develop criteria to eliminate obviously inaccessible features by calculating an approximation of a features accessibility volume $\mathrm{acc}(f)$ from its removal volume $\mathrm{rem}(f)$ and effective removal volume $\mathrm{eff}(f)$. In practice, a cutting tool is held by a tool assembly, such as those shown in Figure 7.8, and attached to a machine tool. For each feature f we calculate an accessibility volume, $\mathbf{acc}(f)$, that is an approximation of the volume swept out by the non-cutting portion of the tool and the tool assembly while the feature f is machined. If $\mathbf{acc}(f)$ has a non-empty intersection with the part P, then feature f is inaccessible in every machining plan for P, and thus can be discarded.

The accessibility volume for the features in $\mathcal{M}$ as defined in Chapter 3 is calculated using two parameters:

Maximum tool cutting length: This is the longest cutting length possible for a tool. For drilling and milling tools we use a value of 128mm. Hence the removal volume for any feature instance $\mathrm{rem}(f)$ can be no deeper than 128mm. For chamfering and filleting tools this value is much less. We will use an estimated value of 20mm for both chamfering and filleting feature types based on the bounds on tools presented in Section 7.2.

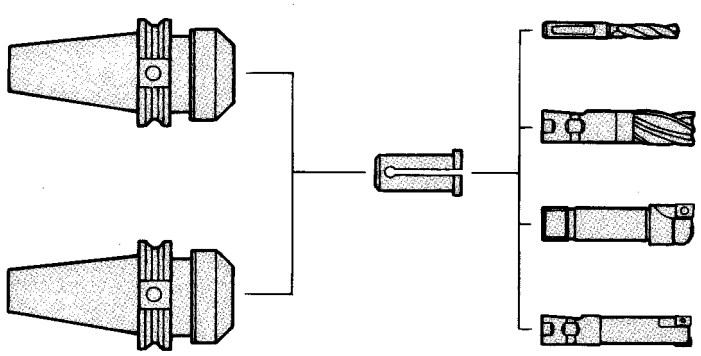
Maximum tool non-cutting length: This is the length of the non-cutting portion of the tool between the end of the cutting surface and the beginning of the tool assembly. For this parameter we use a value of 32mm.

Tool assembly radius: This is the radius of the tool assembly, r_a, such as those shown in Figures 7.8 and 7.9(a). In order to exclude only those obviously inaccessible features, we select a value of r_a =10mm.[2]

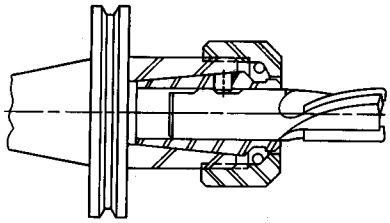
Given the above parameters, the accessibility volume for the features in $\mathcal{M}$ is calculated as follows:

Drilling feature: Given a drilling feature f of radius r with location p and a unit vector v denoting orientation, $\mathrm{acc}(f)$ is defined by four half-spaces, $\mathrm{acc}(f) = (H_1 \cap^* H_2) \cup^*$

[2]The smallest radius tool assembly available from a survey of a variety of catalogs [48, 99, 164, 165, 194]. This value represents an approximate lower bound on the size of the tool assembly needed to hold a tool of a given radius. Provisions for custom tooling is beyond the scope of this work.



(a): components in a typical tool assembly



(b): a milling tool assembly

Figure 7.8: Example tool assemblies from cutting tool manufacturer's catalogs [164, 99].

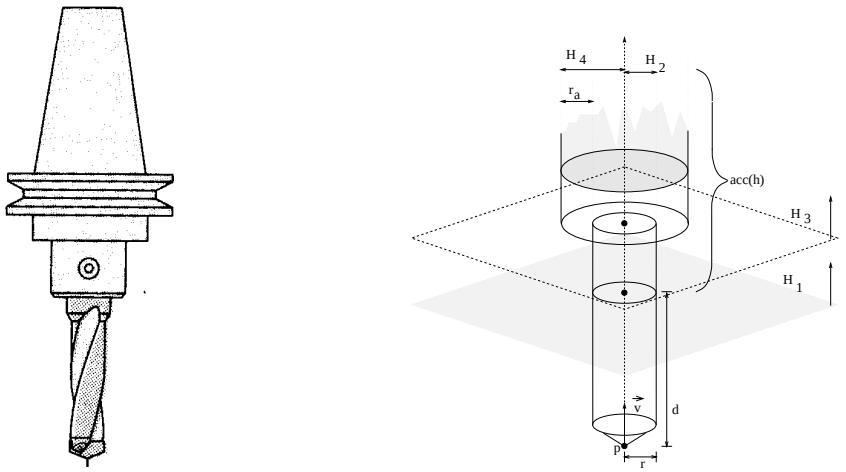
$(H_3 \cap^* H_4)$. H_1 is defined by the plane passing through point $(p + v \cdot 128)$ and normal to v; H_2 by the cylinder containing the points p' whose distance from the ray defined by v and p is less than r. H_3 is defined by the plane passing through point $(p + v \cdot (128 + 32))$ and normal to v and H_4 by the cylinder containing the points p' whose distance from the ray defined by v and p is less than $\max(20, r + 10)$mm. An example of this is shown in Figure 7.9(b).

Milling feature: Given a milling feature f with an edge profile E, location p and a unit vector v denoting orientation, $\text{acc}(f)$ is defined by four half spaces, $\text{acc}(f) = (H_1 \cap^* H_2) \cup^* (H_3 \cap^* H_4)$.

H_1 is defined by the plane passing through point $(p + v \cdot 128)$ and normal to v and H_2 by the volume containing the points p' which, when projected on $+/-v$ onto the plane containing E, lie within E. H_3 is defined by the plane passing through point $(p + v \cdot (128 + 32))$ and normal to v and H_4 by the volume containing all points p' which, when projected on $+/-v$ onto the plane containing E, lie within a distance of 10mm from E. An example of this is shown in Figure 7.10(a).

Chamfering and Filleting features: Given a chamfering or filleting feature f with an edge profile E, a unit vector v denoting orientation, and radius r of the cutting tool,[3]

[3]Referring to the parameters in Figure 3.9 the radius of the tool is calculated using parameter A for



(a): A drilling tool assembly, from [165] (b): Drilling feature h and acc(h)

Figure 7.9: Illustration of a tool assembly and accessibility volume for drilling features.

acc(f) is defined by four half spaces, acc(f) = $(H_1 \cap^* H_2) \cup^* (H_3 \cap^* H_4)$.

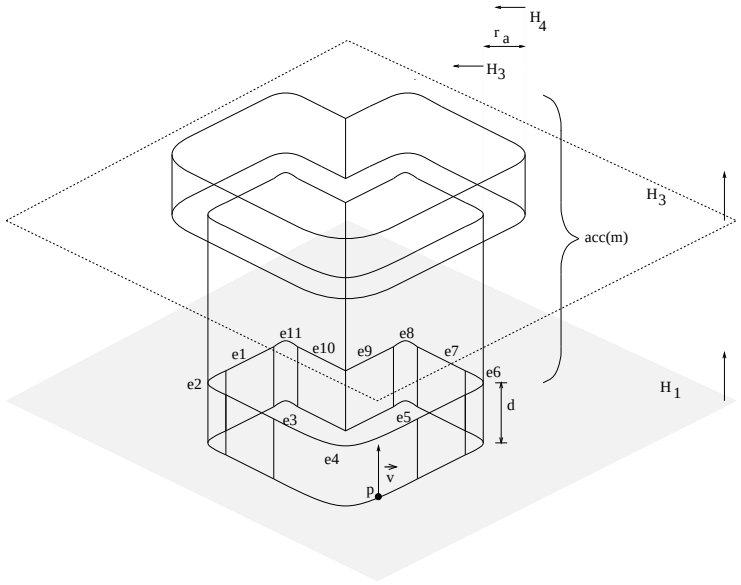
H_1 is defined by the plane passing through point $(p + v \cdot 20)$ and normal to v and H_2 by the volume containing the points p' which, when projected on $+/-v$ onto the plane containing E, lie within a distance of r from E. H_3 is defined by the plane passing through point $(p + v \cdot (20 + 32))$ and normal to v and H_4 by the volume containing all points p' which, when projected on $+/-v$ onto the plane containing E lie within a distance of $\max(20, r + 10)$mm from E.

While the volumes defined above represent very rough estimates of the accessibility volume for each feature, they do provide an effective means of excluding numerous instances of unreasonable features.

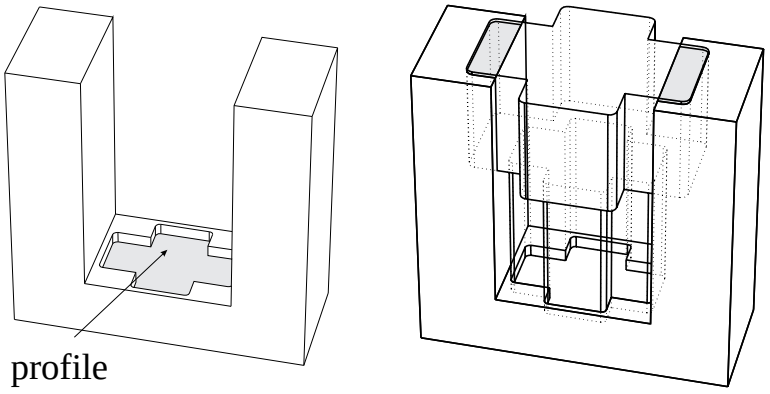
Figures 7.9(a) and (b) illustrate the accessibility and removal portions of feature volumes for drilling features.

An example of how these volumes can be used to calculate interference with the workpiece is given in Figure 7.10. In Figure 7.10 (b), the recognition algorithm for milling trace 1 returns a deep pocket. By calculating the accessibility volume (Figure 7.10 (a)) for this feature instance, one can determine the existence of tool interference with the workpiece (Figure 7.10(c)) and discard the feature because it is unlikely to be manufacturable with common tooling.

chamfering features and parameter R for filleting features.



(a): Accessibility volume for milling feature m, acc(m).



(b): A deep milling feature (c): Interference with acc(m)

Figure 7.10: Testing for feature accessibility.

7.5 Summary

This chapter presents three post-processing steps to operate on the set of well-behaved features produced by the algorithms in Chapter 4. In particular, features are discarded or truncated based on the bounds of common cutting tools. The profiles of milling features returned by the algorithms in Chapter 4 might not correspond directly to the volumes swept out by the rotating cutting tool, if they have characteristics such as sharp convex corners. The profiles of milling features are modified to enhance their correspondence to actual machining operations by taking into account the radius of a milling tool that might machine them. Lastly, features are tested for inaccessibility by approximating the volume occupied by the non-cutting portion of the tool and the tool assembly.

Chapter 8

Implementation and Examples

This chapter gives an overview of the IMACS design critiquing system and of the implementation of a prototype feature recognition system **F-Rex** developed with the methodology described in Chapters 3 through 7.

8.1 Overview of IMACS

The framework described in this thesis has been employed to develop a proof-of-concept implementation of a feature recognition system. This system, **F-Rex**, is part of the IMACS Project (Interactive Manufacturability Analysis and Critiquing System) under development at the University of Maryland College Park's Institute for Systems Research. Within IMACS, as illustrated in Figure 8.1, the role of the feature recognition module is to produce the set of well-behaved primary features from the part's geometry and topology. This feature set is used by the other subsystems for manufacturability analysis and redesign.

8.1.1 Software Tools Employed

During the implementation of **F-Rex** and IMACS we employed a number of development tools. The majority of the code is written in the C++ language [101, 189, 191, 190, 113, 50] using version 3.0.1 of the AT&T C++ compiler from SUN Microsystems. The current system runs on SPARCStations model IPX, 2, 5, and 10-30 workstations under SUN OS 4.1.3.

The remainder of this section gives a brief description of each of the other software tools:

ACIS and the ACIS 3D Toolkit. Spatial Technologies' ACIS© is a solid modeling kernel, a C++ library of of routines and functions with which to develop applications. There are two basic components of the ACIS Solid Modeler: the ACIS Kernel [183, 184, 185, 178] and the 3D Toolkit [180, 181, 120, 182, 179]. The Kernel provides the core C++ library and application protocol interface (API) for the modeler. The 3D Toolkit provides an extended set of higher-level API calls as well as Scheme [1] language interpreter based on the Elk Scheme [47] dialect. The Scheme language interface to ACIS includes both a separate development environment interpreter and an embeddable interpreter, so one can

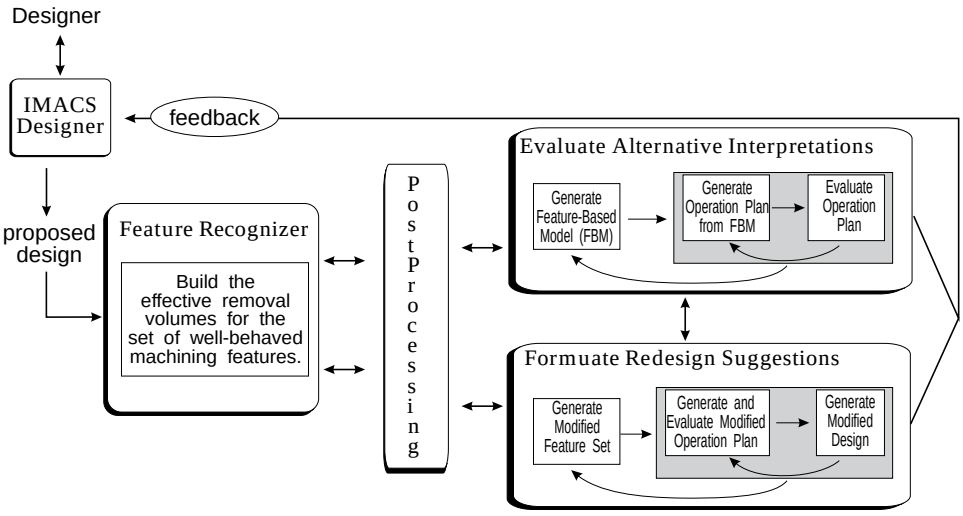


Figure 8.1: Feature recognition operating within the IMACS system.

incorporate 3D Toolkit Scheme code into a C++ application. The 3D Toolkit also provides tools for extending and customizing this Scheme interface.

ACIS is the geometry engine of IMACS and F-Rex. In much the same way as the use of Motif, Xlib and X Windows [146] can speed the development of interactive graphical programs and interfaces, ACIS provides a large library of routines for creation, manipulation, and interrogation of geometric and topological entities with which to build applications. F-Rex uses a mixture of calls to the ACIS kernel and Scheme functions.

HOOPS. Ithaca Software's HOOPS© [13] is a library of C-language routines for display and manipulation of 3D graphics. HOOPS is tightly integrated with ACIS to open display windows for ACIS solids and to control rendering.

NIHCL. The NIH C++ Class Library [64, 63] (previously known as the "OOPS" Class Library) is a portable, UNIX-system-compatible C++ class library from the National Institutes of Health. The current IMACS implementation is based on version 3.14 of the library, which provides method functions for common data structures such as lists, sets, stacks, and arrays. NIHCL was used to encapsulate many of the solid modeling functions by creating classes for solid models and features.

Tcl/Tk and Expect. Tcl is an embeddable tool command language developed by Ousterhout at the University of California at Berkeley [132, 134, 135]. Tk [133] is a graphical user interface toolkit based on Tcl.

Expect [112] is an extension to TCL that enables control of other interactive applications. Following a script, Expect knows what output is to be produced by a program and what the correct response should be. TCL provides structures for branching and control.

An Expect script can invoke other programs and run them simultaneously, passing information among their processes in a manner similar to reading and writing among multiple files. Expecttk is Expect plus the TK motif-like graphical user interface extensions to TCL. With Expect/Expectk, one can combine independent interactive applications into a package controlled by a single GUI.

Tcl, Tk and Expect/Expecttk were used to create the graphical user interface for the IMACS and F-Rex programs.

8.1.2 Description of other Modules

This section briefly describes the different components in the IMACS system and their relationship to the F-Rex feature recognition module.

IMACS Designer. IMACS Designer is a basic user interface for the ACIS© 3D Toolkit. The objective of IMACS Designer was to give users easy visual access to the functionality of the ACIS 3D Toolkit and enable them to invoke many of the basic Toolkit commands through a graphical user interface.

IMACS Designer is written in the language Expectk and runs on the UNIX platforms supported by the 3D Toolkit. The interface provides a matrix to many of the core commands of the 3D Toolkit. The basic idea is that we can parameterize the Toolkit's Scheme commands by selecting options with a mouse and entering the necessary parameters. Expect then constructs the appropriate scheme commands and sends them to the Toolkit. An illustration of the IMACS Designer interface is shown in Figure 8.2.

Tolerancing Module. This module is a prerequisite to the manufacturability analysis tool. The Tolerancing module allows the designer to associate a limited number of ANSI Y14.5 tolerances with the solid model of the design. This tool is further described in [70]. Currently, F-Rex does not make use of this tolerance information when generating feature instances.

Manufacturability Analysis. This is the core IMACS module—it uses the set of features produced by F-Rex to generate and evaluate alternative machining plans for the part by incorporating precedence constraints and information about machining parameters and tolerances. The cost and time for machining plans that satisfy the design requirements are used to estimate a rating of the part's manufacturability [67, 70].

Redesign. This module formulates redesign suggestions based on plan information and the set of well-behaved features. By making modifications to the operations in the generated

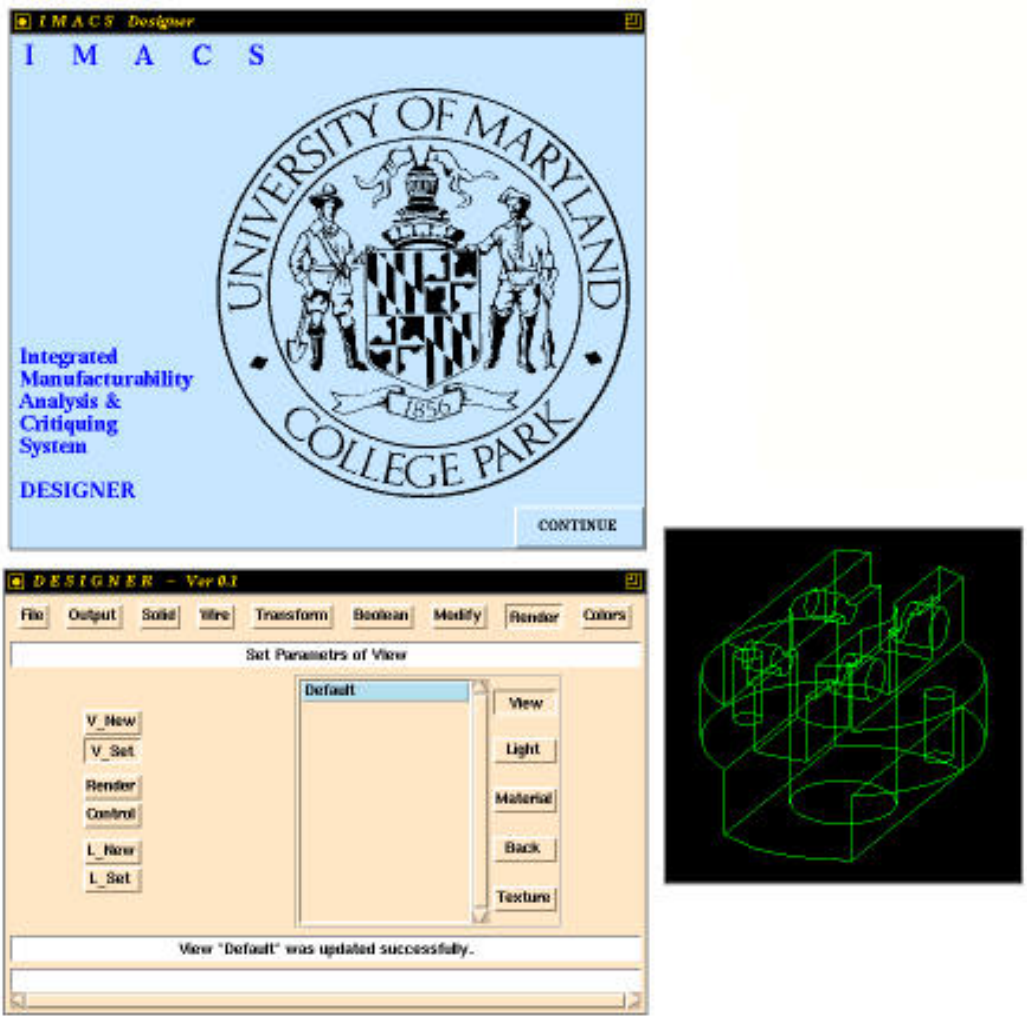


Figure 8.2: Screen capture of the IMACS Designer interface.

plans, it creates modified versions of the design that, in addition to satisfying design requirements, have improved manufacturability [65, 37]. The new designs are presented to the designer as alternative possibilities to be considered. At the current time, implementation of this module is in progress.

8.1.3 Integration of F-Rex with other Modules

F-Rex communicates with the other IMACS modules using the well-behaved primary feature set, $\mathcal{F}$. This information includes the ACIS solid models for each of the features that were recognized, along with their attributes (type, depth, edge profile, etc.). This information is transmitted in the form of a file containing the feature attribute values along with solid models (ACIS `.sat` files) for the instances of each feature.

8.2 The F-Rex Implementation: Limitations and Restrictions

F-Rex exists in both a serial and a distributed parallel version (the parallel version is discussed in Chapter 9). In both cases, the design and shape of the stock material are provided directly from a CAD system (IMACS Designer) as ACIS solid models. ACIS models representing the instances of features from the class of machining features $\mathcal{M}$ are produced as output, along with information about feature parameters. Both implementations of F-Rex are only proofs-of-concept based on the theory outlined in the preceeding chapters.

For the algorithms described in Chapter 4, the current implementation of F-Rex handles drilling traces 1 and 2 in their entirety. The implementation handles many common cases of milling traces 1-3. The algorithms of Section 4.3.2 rely on numerous calls to a projection routine—an operation not directly supported by the ACIS Solid Modeling Kernel.[1] For the purposes of the implementing of the framework in this thesis, we have written some limited extensions to ACIS to handle a number of the common cases where 3D projections are necessary. Development of robust 3D projection and sweeping routines for the full class of solids described by ACIS (i.e., both manifold and non-manifold objects with boundaries that might consist of both analytic and b-spline surfaces) has been the subject of intensive research activity [121] and development of robust sweeping algorithms is beyond the scope of this thesis. Identification of chamfering and filleting features has been omitted because they are required by the downstream IMACS modules.

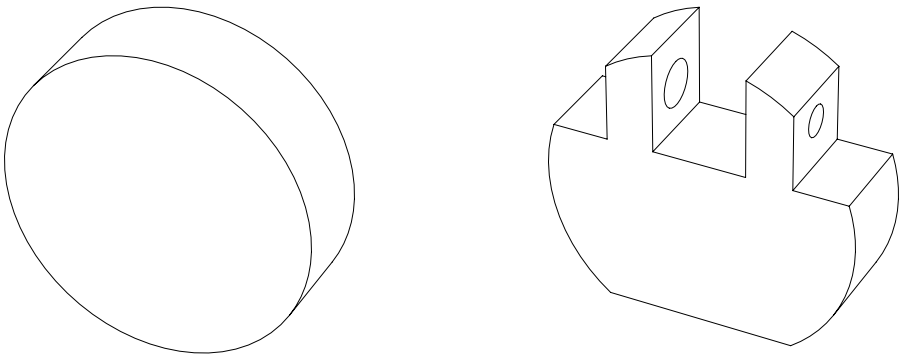
The post-processing algorithms and heuristics described in Chapter 7 are implemented partially in F-Rex and partially in the downstream modules. Currently F-Rex performs some accessibility/inaccessibility analysis. Offsetting is being addressed as part of the redesign system. Lastly, the heuristics for tooling constraints and bottom blends have been held for later implementation.

Even with these limitations, F-Rex is a functional feature recognition system and has been employed for a variety of real-world examples. It would require an expanded implementation effort to make it fully correspond to the theory outlined in this thesis—a software development task that would be most appropriately and effectively executed as part of a commercialization effort.

8.3 Examples

The examples below (unless otherwise noted) are the output of F-Rex. The orientation of milling features is noted by their extension beyond the stock material.

[1] Higher dimensional sweeping of 3D bodies and projections are supported by several third party ACIS "husks," including the STRATA tool path generation husk.



(a): stock (before machining) (b): part (after machining)

Figure 8.3: The simple bracket example from Figure 3.1.

8.3.1 Example: Simple Bracket

Figure 8.3 shows an example of a simple bracket with 16 faces that appeared earlier in Figure 3.1. F-Rex produces 3 drilling and 16 milling features for this part, as illustrated in Figure 8.4.

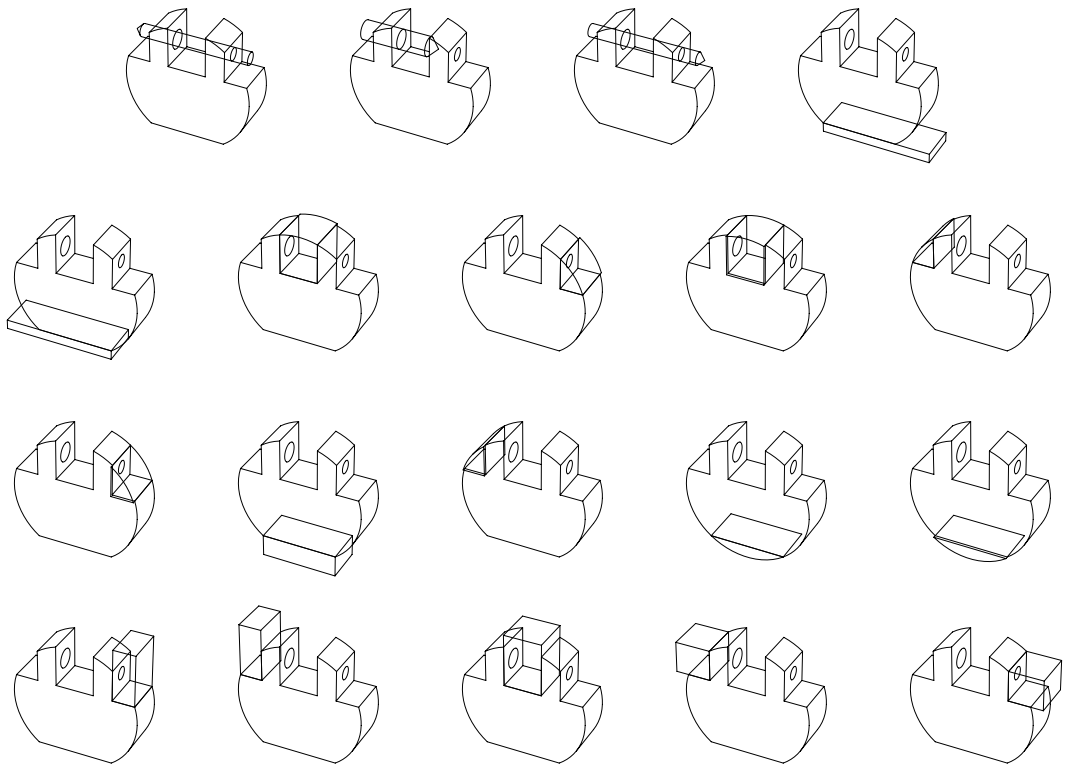


Figure 8.4: The features identified by the F-Rex system for the bracket from Figure 3.1. F-Rex has extended milling features slightly beyond the stock material to indicate their orientation. If milling features were truncated to the size of the stock, the above picture would include several apparent duplicates.

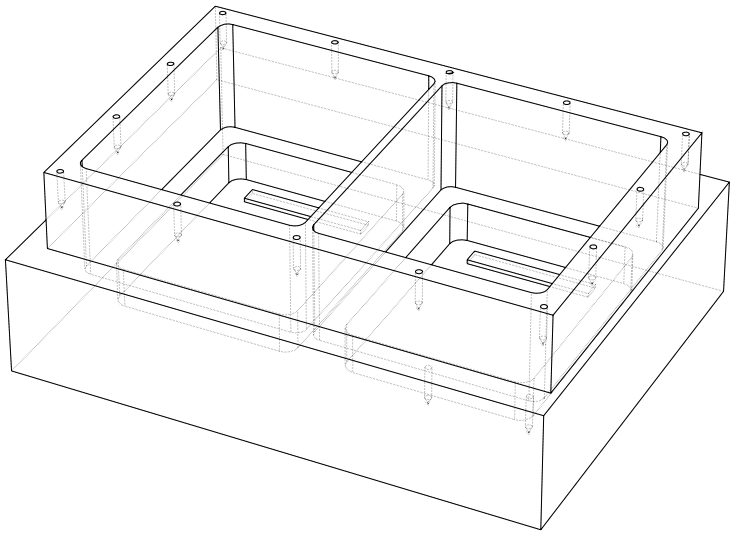


Figure 8.5: A part with a number of intersecting features.

8.3.2 Example: A Housing

Figure 8.5 presents a part with 89 planar, cylindrical, and conical faces to be machined out of a rectangular block of stock material. F-Rex identified 51 drilling and milling features; Figure 8.6 shows 35 of these features. Those not shown are additional drilling features that are entirely subsumed by one of the milling features.

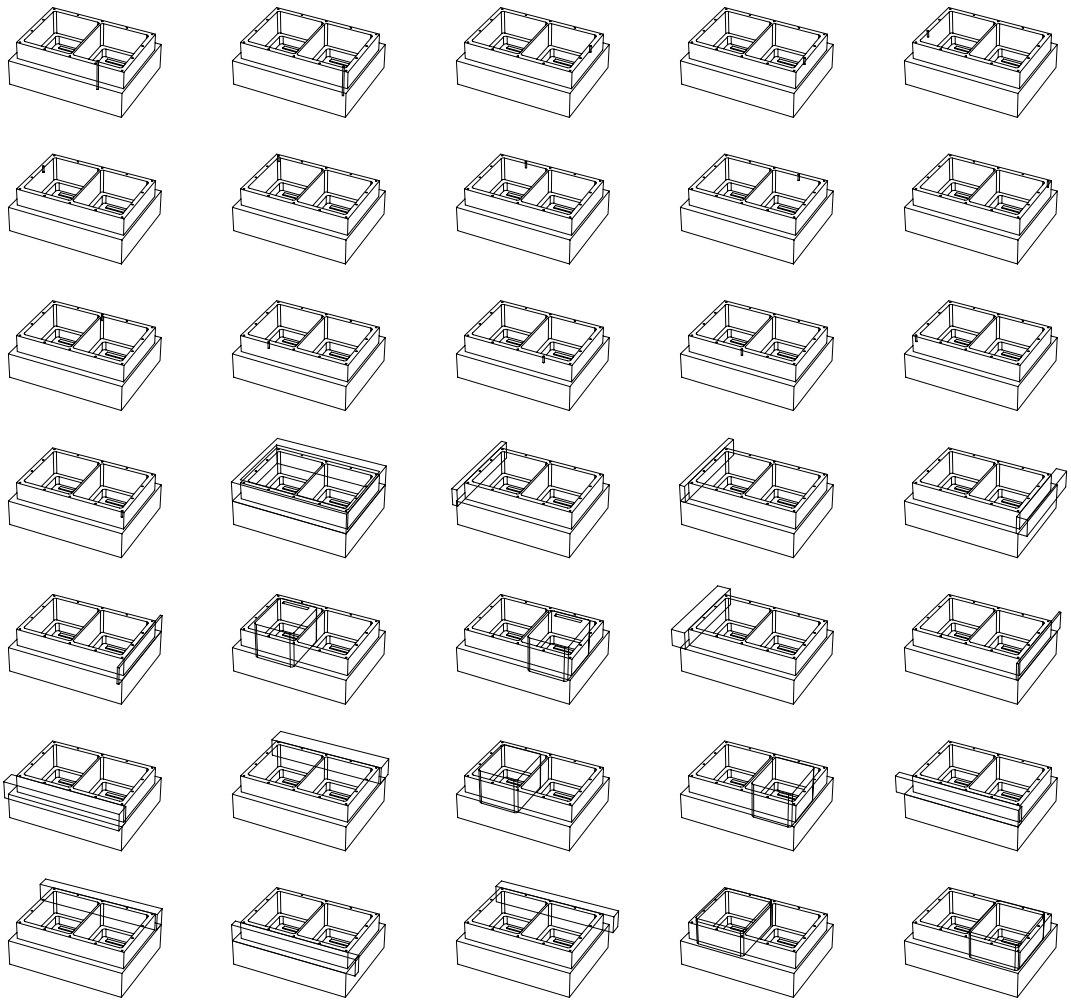


Figure 8.6: Some of the features found for the part in Figure 8.5.

8.3.3 Example: A Socket

The example part in Figure 8.7 is a design of a socket taken from [148]. Figure 8.7(a) shows the design for the socket and Figure 8.7(b) a CAD model of the design. The initial workpiece, S, is a cylindrical object of raw stock material to be acted upon by a set of machining operations that generate features (Figure 8.7(c)). Machining operations remove material from the initial workpiece to create the design attributes of the part (i.e., $P \subseteq S$). Figure 8.7(d) illustrates the delta volume for this particular part and stock.

This part, when machined from a cylindrical piece of stock material, has 37 faces in the delta volume. There are 12 drilling and 20 end-milling features in its feature-based models that can be produced. F-Rex identified 22 feature instances, as shown in Figure 8.8.

An example of post-processing. To illustrate the distinction between the theory and algorithms presented in Chapters 3 through 7 and the current implantation of F-Rex, Figure8.9 shows examples of non-primary, primary, and offset primary milling features for the socket. Figure 8.10 shows the 22 offset, accessible, well-behaved features produced by the BUILD_FEATURES algorithm for the part in Figure 8.7. Note that the features pictured in Figure 8.8 are described only as their effective removal volumes; the curved edges in the profiles of milling features shown in Figure 8.10 have been adjusted by profile offsetting.

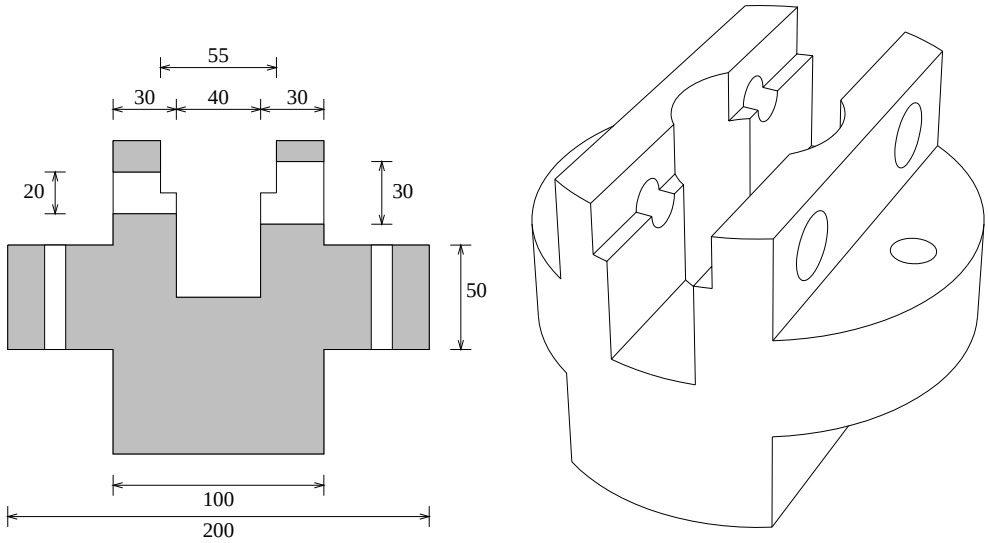
The features in Figures 8.10 can be used to generate 512 different feature-based models (FBMs) for the part; Figure 8.11 shows the two of the possible FBMs:

$$F_1 = \{h_1, h_2, h_3, h_4, h_5, h_6, h_7, m_1, m_2, m_3, m_4, m_5, m_6\}$$

and

$$F_2 = \{h_1, h_2, h_3, h_8, h_9, h_{10}, h_{11}, m_1, m_2, m_7, m_8, m_9, m_{10}\}.$$

Figure 8.12 illustrates a limitation of F-Rex's current implementation of the RECOGNIZE_WELL-BEHAVED_FEATURES algorithm. The figure shows several through features that can be recognized using the algorithm for milling trace 2. Our current implementation's limited form of projection does not allow it to find these features.



(a): design of a socket (b): part (after machining)

(c): stock (before machining) (d): the delta volume

Figure 8.7: An example from [148] of a design of a socket, along with solid models for the part, stock, and delta volume.

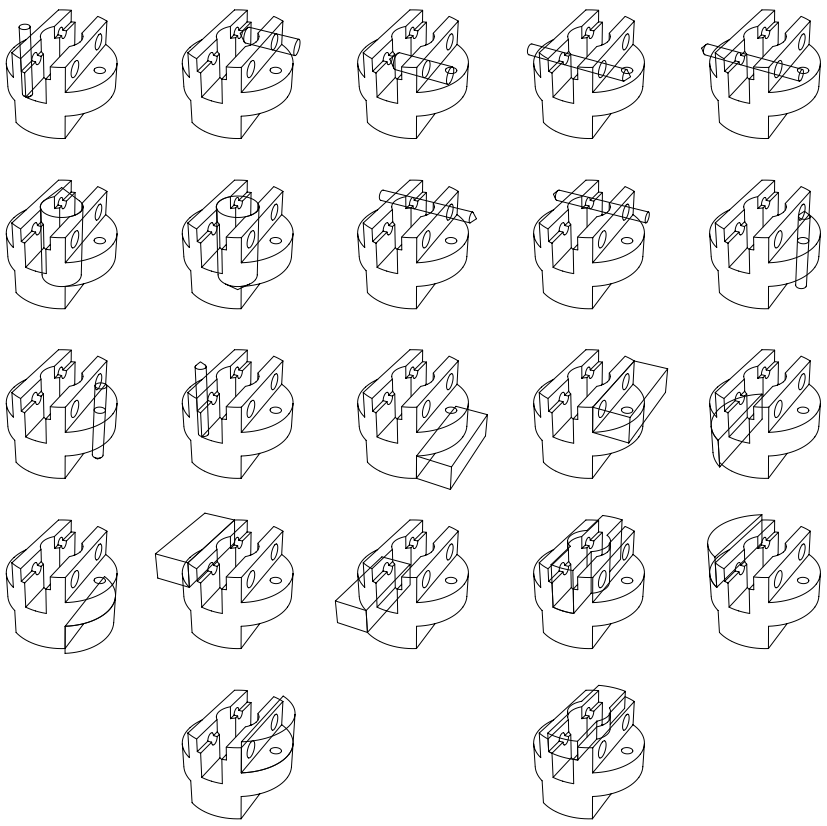
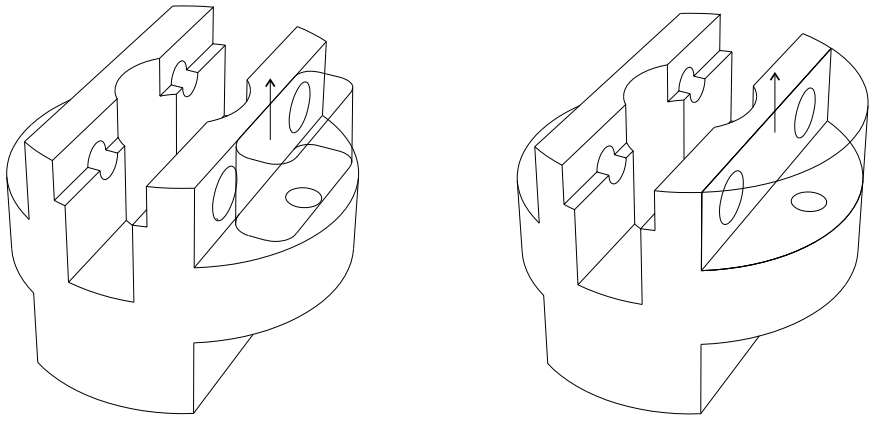
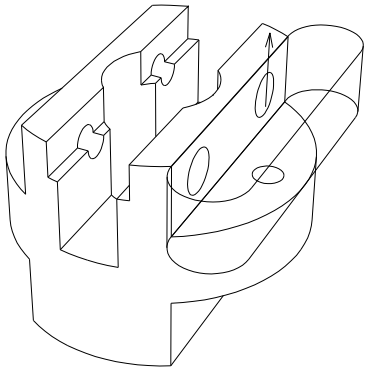


Figure 8.8: The features identified by F-Rex for the part shown in Figure 8.7(b).



(a): non-primary step-milling feature (b): primary step-milling feature



(c): offset primary step-milling feature

Figure 8.9: Examples of non-primary, primary, and offset primary feature instances.

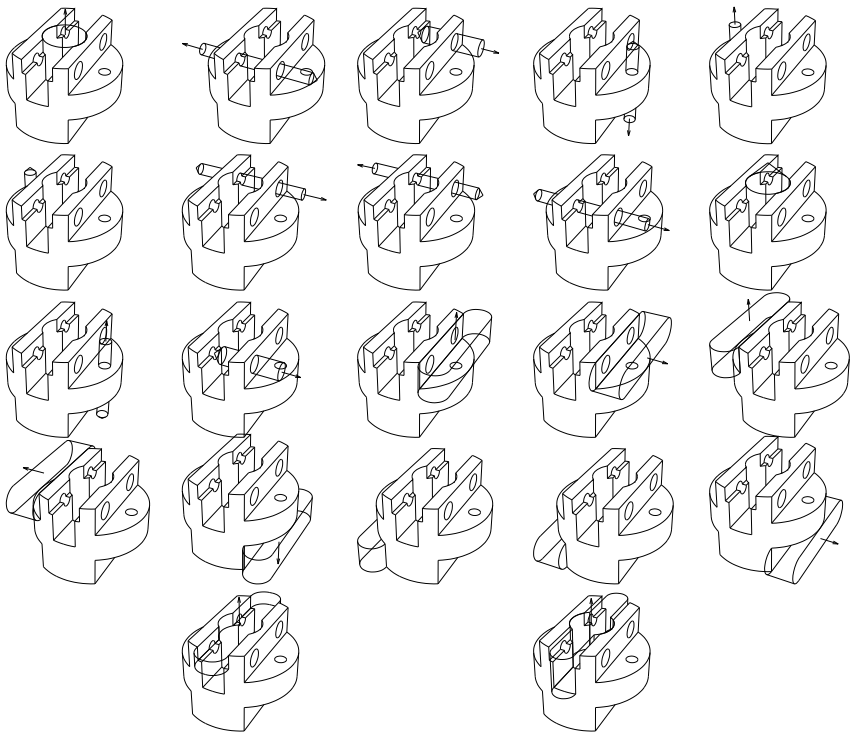


Figure 8.10: The offset well-behaved features as they would appear after post-processing by BUILD_FEATURES for the part shown in Figure 8.7(b).

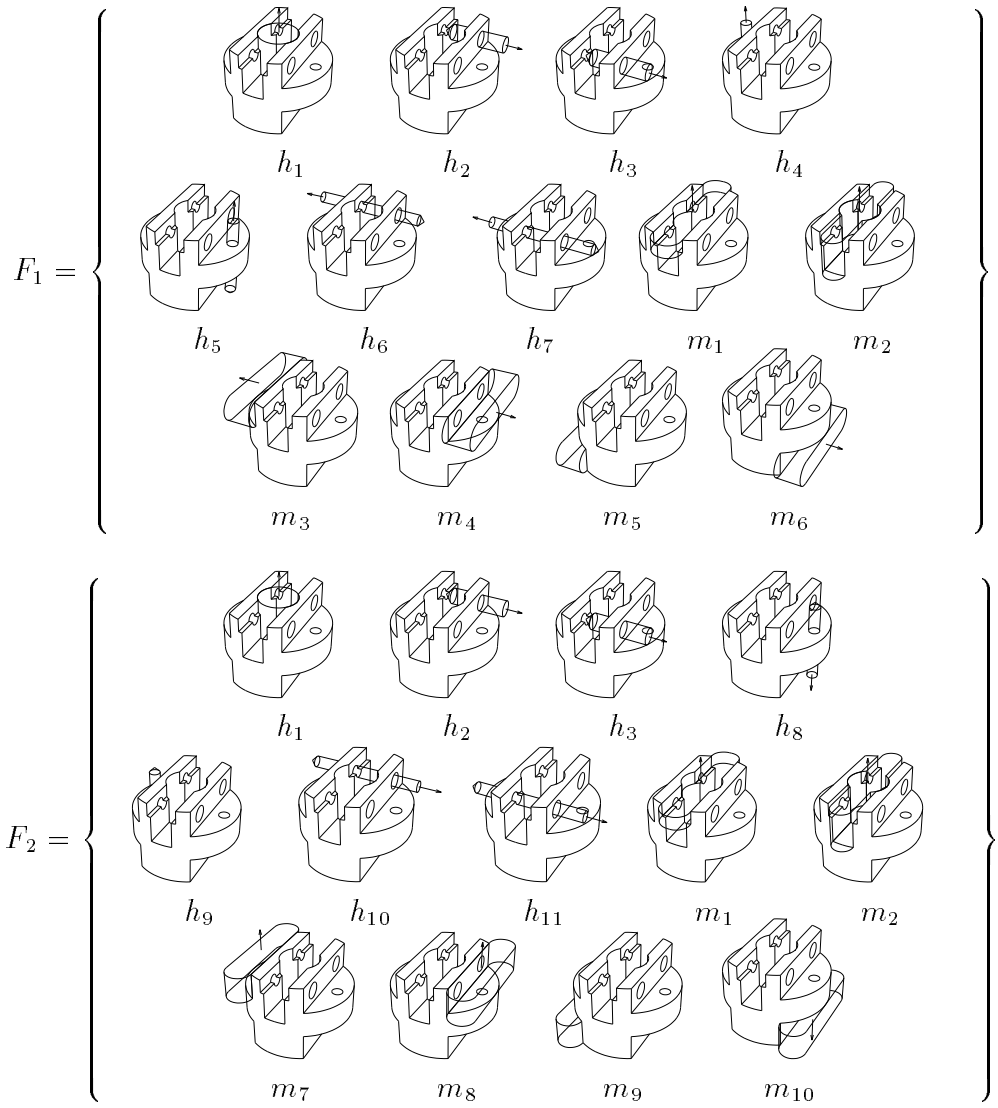


Figure 8.11: Two alternative feature-based models consisting of drilling and milling features for the part in Figure 8.7(b).

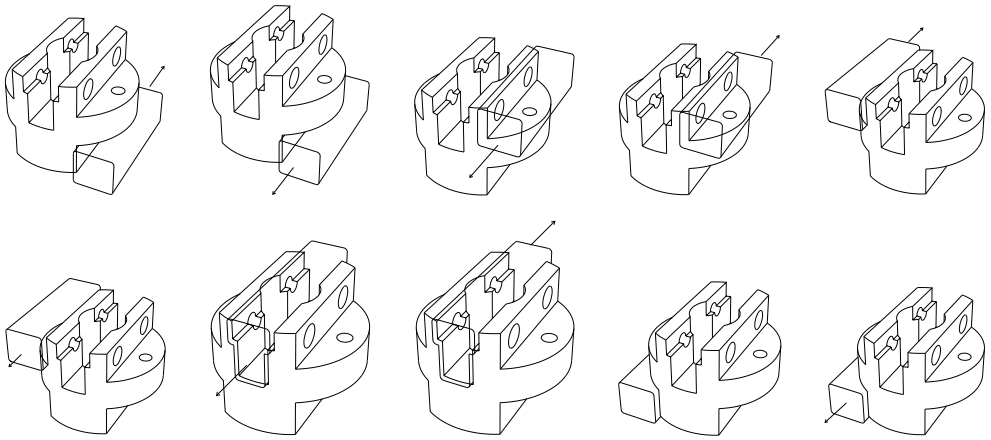


Figure 8.12: Features for the part shown in Figure 8.7(b) that are recognizable using milling trace 2 but not identified by the F-Rex implementation.

8.4 Summary

This chapter provided an overview of the implementation of the IMACS manufacturability analysis system and the F-Rex feature recognizer. We outline the distinctions between the algorithms and theoretical results developed in Chapters 3 through 7 and the implemented prototype. In addition, we present detailed examples of the output of the F-Rex feature recognition system.

Chapter 9

Parallelization

This chapter presents a parallelization of the feature recognition techniques from Chapter 4 using of distributed algorithms. In addition, this chapter outlines methods which can exploit this divide-and-conquer parallelism to further simplify individual subproblems.

9.1 Related Work: Multi-Processor Solid Modeling

The bibliography of work on multi-processor algorithms for solid modeling applications is limited but growing. Currently, most work has focused on parallel operations on CSG trees and other CSG representations of polygonal or polyhedral entities. Ellis et al. [49] have developed the RayCasting Engine: a hardware-implemented facility for sampling solids represented in CSG for a variety of purposes, including rendering and mass-property calculations. They outline how this special-case hardware makes possible brute-force solutions to difficult computational problems, such as spatial sweeping and offsetting.

Narayanaswami and Franklin [129] present a parallel multi-processor method for calculating the mass properties of polygonal CSG objects and outline some extensions for applying the techniques to 3-D polyhedra. Banerjee et al. [12] have developed parallelized algorithms for evaluating CSG trees that operate with a fixed number of processors with shared memory.

In the domain of boundary representation modeling, Karinthi et al. [97] have produced a parallel algorithm for performing boolean set operations on polygons and polygons with holes. In Almasi et al. [6], these techniques are extended to more general loops of edges.

Strip and Karasick [188] present techniques for performing solid modeling operations on a massively parallel SIMD (single instruction multiple data) computer. They provide a data structure for representation of solid models and a variety of parallel algorithms for implementing solid modeling operations. In addition, they present performance comparisons with serial implementations.

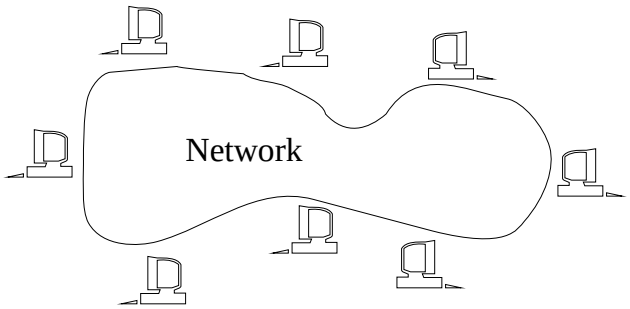


Figure 9.1: Inter-networked computational resources: the network as the computer.

9.2 Distributed Algorithms for Feature Recognition

Existing work on feature recognition has dealt with exclusively serial computer architectures. In the distributed computing paradigm, collections of autonomous computational resources are interconnected on a network, as illustrated in Figure 9.1 [197]. While these resources do not share main memory, they may share access to common devices such as peripherals, file systems, output devices, etc. Software systems can use the network and shared peripherals to exchange information between the autonomous resources.

A fundamental issue when building distributed software systems is how to enable independent computers to cooperate with each other in solving problems. In this section, we will apply distributed algorithms to the problem domain from Chapter 4 in order to build more efficient and scalable solutions to the feature recognition problem.

9.2.1 Observations

Trace-based techniques lend themselves well to parallelization, providing several levels at which the problem can be divided. What might be less evident is that, in parallelizing the problem, one can make additional geometric and topological simplifications to independent problem subtasks to reduce their computational difficulty.

The feature types (outlined in Chapter 3) and their traces (outlined in Chapter 4) each introduce natural partition lines along which the problem can be divided into independent subproblems to be solved by different processors.

As presented in Chapter 3, the final feature set $\mathcal{F}$ contains all those feature instances from $\mathcal{M}$ that are members of feature-based models of the part. $\mathcal{F}$ contains all instances of the feature types in $\mathcal{M}$ present in the given part. Note that for the features in $\mathcal{M}$, the act of recognizing a feature of type M_1 is independent of the recognition of a feature of type M_2—hence the feature instances of type M_1 can be calculated separately from those of type M_2. For instance, in the example domain presented in Chapter 4, a particular drilling feature f being a member of some feature-based model does not alter the potential existence of any end-milling features.

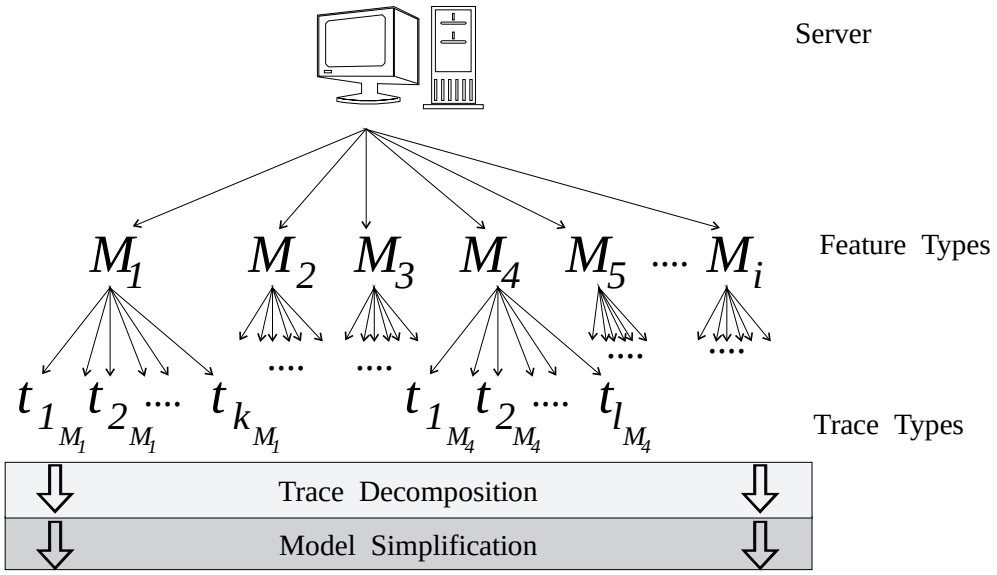


Figure 9.2: Overview of the divide-and-conquer distributed approach.

Secondly, the set of traces $\mathcal{T}$ (from the generic algorithm in Section 4.2) introduces an additional level for partitioning the problem. Recall that for each feature type M in $\mathcal{M}$, there is a collection of traces $t_{M1}, t_{M2}, \ldots t_{Mk}$ for building instances of features of type M. One can decompose the problem of finding all features of type M by trace and handle each trace t_{Mi} on a different processor.

One observation is that this may introduce some redundancy; i.e., it may be possible to find the same feature instance f in different ways using different traces. There are two possible approaches to handling this redundancy. One method is to delete duplicate features when building the final feature set $\mathcal{F}$. A second approach, and the one which we will employ, is to handle the traces capable of producing equivalent feature instances together on the same processor and remove duplicates as they are found. This introduces another level of parallelization by dividing the set of traces found into independent subsets. In this way redundancies are addressed at the level at which they occur, thus simplifying the task of building the final feature set $\mathcal{F}$.

Parallelizing feature recognition produces other indirect benefits—in particular, a large portion of the costs in a feature recognition system are due to the complexity of geometric computations and geometric reasoning. When isolating independent problem subtasks, one can make geometric and topological simplifications that identify the information in the original part needed to build and verify the feature instances. In this way, many of the subproblems may require only a fraction of the information present in the solid models of the original part and stock.

9.2.2 Distributed Algorithms

For the problem domain of Section 4.1, we will employ a divide-and-conquer approach: a central computing resource acts as a server to set up the problem and transmit subtasks to client machines distributed on the network, as illustrated in Figure 9.2. Each of the individual client processors is given an independent portion of the global feature recognition problem.

Recalling the serial trace-based algorithm of Section 4.2, we present an outline for a multi-processor trace-based feature recognition system. There are two main components to this system: a server algorithm and a client algorithm. The server algorithm is presented as follows:

Server algorithm.

1. Input a collection of feature types, $\mathcal{M}$, a solid model for the part P, and for the initial stock material S. Initialize $\mathcal{F} = \emptyset$.
2. For each feature type M in $\mathcal{M}$ do:
   (a) Identify a free resource and fork a new process on it.
   (b) For each trace type t_{Mi} for feature type M do:
      i. Find the set $T_{t_{Mi}}$ of instances of traces of type t_{Mi}.
      ii. Decompose the set $T_{t_{Mi}}$ into independent subtasks, $\tau_1, \tau_2, \ldots \tau_j$.
      iii. For each τ_i do:
         A. Decompose the part P using the τ_i. Result is P'.
         B. Fork a new process on a free resource to call the client recognition algorithm on P'.
      iv. Let $F_{t_{Mi}}$ be the set of features returned by the client.
3. $\mathcal{F} = \mathcal{F} \bigcup_{\forall t_{Mi}} F_{t_{Mi}}$.
4. Remove duplicate features from $\mathcal{F}$.
5. Return $\mathcal{F}$.

Client algorithm. The client algorithm is to be invoked by the server on each of the available computational resources:

1. Input a feature type, M, a trace type, t_{Mi}, a set of instances T of trace t_{Mi}, and solid models for the part P', and the stock material S.
2. Simplify the solid model of the part P'. Result is P''.
3. Call $\mathcal{P}(t_{Mi})$ to build feature set $F_{t_{Mi}}$.

4. Return $F_{t_{M_i}}$.

To implement this client-server algorithm, three technical areas must be addressed, as described in the subsequent subsections.

9.2.2.1 Task Initialization

There are four levels at which the recognition problem is initialized:

- **Types of features to be recognized:** different feature types (in this example drilling and end-milling) are considered by separate computing resources, as discussed in Section 9.2.1.
- **Types of feature traces:** different traces for each of the feature types are considered by separate computing resources, as discussed in Section 9.2.1.
- **Trace decomposition:** given a specific feature type and a trace for recognizing it, decompose the set of instances of this trace to independent subsets to subdivide the recognition task. This is discussed below in Section 9.3.1.
- **Part simplification:** given a specific feature type and a trace for recognizing it, alter the geometric and topological information in the solid model of the part to reduce its complexity. This is discussed in Section 9.3.1.

9.2.2.2 Task Distribution

Once tasks are initialized, the next phase is to distribute the individual tasks to the available computing resources. This is done by invoking a client feature recognition procedure for each separate task, with each client on its own processor.

In the example domain of Chapter 4, distributing tasks is straightforward: each task identified during initialization is executed on its own processor. This becomes more complex when bounds are placed on the number of available computing resources.

9.2.2.3 Synthesis of Results

Each separate client procedure, upon termination of its portion of the recognition task, transmits its results back to the server machine. The features returned are then integrated into an overall solution. In this domain, recombining results requires building the final feature set as the union of those returned by each client machine.

However, the fact that this example domain lends itself well to building an overall solution from the separate subtasks may not generalize to other manufacturing domains. For example, this phase might have to include additional computations such as modeling feature interactions, eliminating redundant features, or identifying compound features or feature groups.

9.3 Example

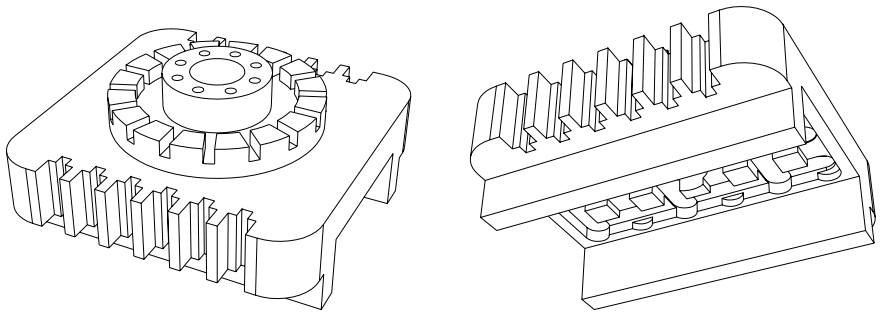
This section presents detailed examples of how the multi-processor algorithms described above are applied to the recognition of features from specific parts.

9.3.1 Task Initialization

The task decomposition stage groups feature information and isolates traces to be handled by separate computing resources. There are four levels of task decomposition.

For illustration purposes, we shall assume there is no limit on our computational resources. When there is a bound on the number of processors available, the task decomposition or the distribution of the task may need to vary to more efficiently partition the problem. In our implementation (discussed in Section 9.5), we distribute the tasks evenly over the available processors.

The decomposition by feature type and decomposition by trace, as noted before, are straightforward. In developing techniques for part decomposition and simplification, one is faced with a trade off between the sophistication of techniques and their computational costs. Using very sophisticated techniques to maximize the ability of each individual processor to produce useful feature instances in a minimal amount of time might increase the computational overhead to a degree that mitigates the benefits of parallelization. In choosing the following conditions, we have picked decompositions and simplifications that are computationally cheap. While it is certainly possible to present more complex decomposition criteria, an important consideration is that the conditions themselves cannot be more complex than the original recognition problem. If the decomposition conditions were themselves costly, the overhead considerations might eliminate any of the speedup benefits we hope to achieve by using a multi-processor approach.



(a): part (after machining) (b): part (underside view)

Figure 9.3: An example part to illustrate the multiprocessor techniques.

The remainder of this section discusses the decomposition of part geometry and topology and techniques for model simplification. For illustration purposes we shall refer to the example part in Figure 9.3 when describing the specifics of the parallel approach.

Trace Decomposition. A given feature instance might be created from any one of several traces it leaves in the part. The objective of this phase is to gather together all of the trace information capable of producing equivalent or identical feature instances. While we will only consider geometric and topological information in this thesis, this decomposition can be extended to include other data (i.e., tolerances, surface properties, etc.).

We present a four-part decomposition for the geometric and topological information in the part. The conditions are based on properties of the traces for constructing feature instances. There may be other conditions that provide an equivalent means of arriving at a task decomposition with the desired properties. Decomposition of the geometry and topology based on feature types and traces proceeds as follows:

1. Decomposition for drilling traces 1 and 2:
   Group together cylindrical and conical faces with equivalent axes that are convex with respect to the delta volume Δ.

   Rationale: This collects all possible drilling traces that might be machined in the same orientation. Drilling features with multiple traces (e.g., several separate cylindrical faces) can be isolated and identified.

2. Decomposition for end-milling trace 1:
   Group together all coplanar edges and faces. In the example illustrated in Figure 9.4, disjoint planar part faces and their bounding edges are grouped to be handled on the same processor. This grouping collects all faces sharing the same underlying surface.

   Rationale: This collects all possible end-milling traces that might be machined in the same orientation, possibly by the same operation. End-milling features with multiple traces (e.g., a bottom surface divided into multiple subfaces) can be isolated and identified.

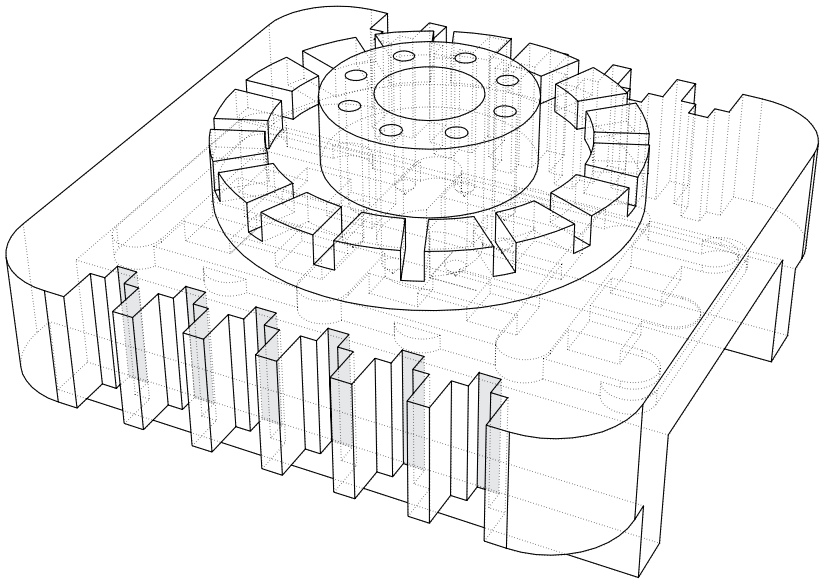
3. Decomposition for end-milling trace 2:
   Group cylindrical surfaces with equivalent axes.

   Rationale: This groups all potential corner radii and curved walls for end-milling features with the same machining orientation.

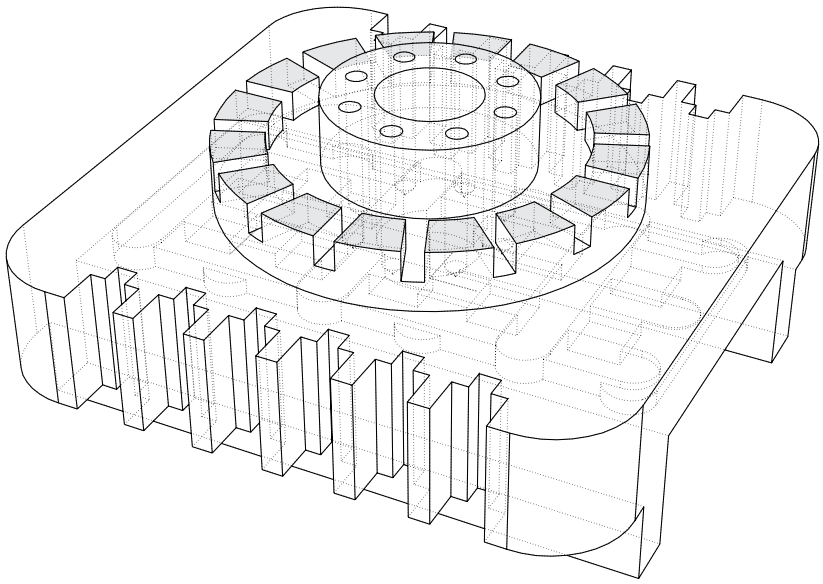
4. Decomposition for end-milling trace 3:
   Group planar surfaces with normals perpendicular to a common vector; i.e., for each grouping there is a vector v such that, for all surfaces s_i and s_j in the grouping, $\text{normal}(s_i) \cdot v = \text{normal}(s_j) \cdot v = 0$. Note that some surfaces may be present in more than one group.

   Rationale: This groups traces for end-milled features based on machining orientation; hence, through features that can be machined in the same orientation are placed in the same group.

The above decomposition groups together those traces from the part that might produce equivalent feature instances. In this way, redundancies can be eliminated at the subprocess level to facilitate later recombination of results.



(a)



(b)

Figure 9.4: Example trace decompositions based on milling trace 1. The shaded faces have been grouped based on their underlying surfaces and are to be handled on a single processor.

Part Simplification. The objective of this step is to reduce the amount of data that must be considered by each processor to a minimum sufficient to construct feature instances from the traces it has been given. In this way, one can reduce the cost incurred by solid modeling operations during feature recognition. For example, one can reduce the number of geometric and topological entities while still retaining the information required to construct feature instances from the particular trace. In this way the complex part geometry that does not affect the feature trace under consideration can be eliminated.

This approach to part simplification is similar to the bounding-box techniques used in current solid modelers. Many solid modeling systems, when performing complex operations (such as booleans or interference tests), compute bounding boxes around entities in the boundary model and preform simplified computations on the boxes. The results of these cheaper computations are used to determine whether more detailed numeric computations are necessary. An example of this is the basic boolean intersection algorithm; i.e., if the bounding boxes of two entities do not intersect then the two entities do not intersect.

We simplify the solid models of the part and stock in these ways:

1. Simplification based on drilling trace 1:
   Given a cylindrical surface c in the delta volume of radius r, P' contains all the portions of P that lie within r of the axis of c.

   Rationale: This simplification retains enough information to check for tool gouging and interference. To check for interference between the workpiece and the machine tool, this radius can be enlarged depending on the size of the tool assemblies available in the particular set of manufacturing resources.

2. Simplification based on drilling trace 2:
   Given a conical surface c in the delta volume with a maximum radius r and located at point d, P' contains all the portions of P that lie within r of the axis of c and in the half-space above d.

   Rationale: This simplification is the same as that for drilling trace 1.

3. Simplification based on end-milling trace 1:
   For a given set of co-planar edges and faces, let p be the plane containing this set of edges and faces. Let d be a root point in the plane p and let v be p's normal vector; P' contains all the portions of P that lie in the half-space defined by d and v.

   Rationale: This simplification retains all geometric and topological information that lies above the bottom surface of the milling feature and discards all information below it.

4. Simplification based on end-milling traces 2 and 3:
   No simplifications are made for these traces.

   Rationale: Finding these types of end-milling feature instances might require consideration of information from the entire part, and the processing required in this case might be costly.

Figure 9.5 illustrates part simplification for end-milling trace 1. In the figure, the edges of the shaded planar faces are being considered as traces indicating potential bottom surfaces of an end-milled features; vector v denotes the orientation of the potential feature. In each case, the trace information is used to eliminate the portion of the part lying below the trace—information that does not get considered when building a feature instance in direction v. Note that, in making this rudimentary simplification, the number of geometric and topological entities to be considered is greatly reduced.

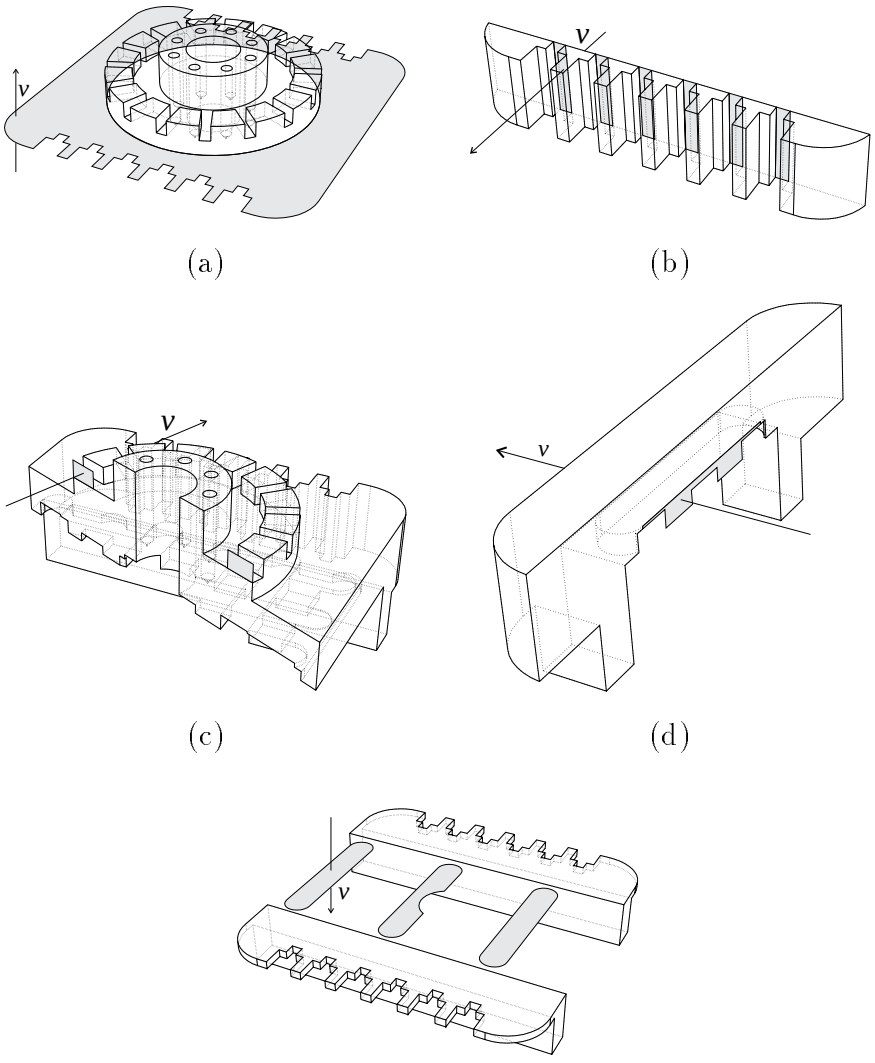


Figure 9.5: Example simplifications based on milling trace 1.

9.4 Computational Improvements

We can expect the speedup to be no more than a factor of K, where K is the number of processors available. In reality, the task decomposition to set up parallelization incurs some added cost, as does the recombination of results at the end. These additions are negligible, however, when compared with the costs incurred to perform the recognition process on each of the subproblems.

Within a trace-based methodology, as outlined in Section 5.3, the overall complexity of recognition depends on two factors: the difficulty in generating the set T of potential traces, and the complexity of the methods for generating feature instances from traces, $\tau(n)$.

Given K processors, we can theoretically expect the complexity of the parallelized version of the trace-based algorithm to be:

$$\Omega(\frac{|T| \cdot \tau(n)}{K})$$

In Section 5.3, a rough upper bound on the size of T was computed from the model of the part and the types of traces by counting the number of geometric and topological entities: $|T| \in O(n)$. The complexity of the feature construction routines is more difficult to assess and is where the majority of the computational costs occur. Much of this cost is due to geometric queries and reasoning operations used to find the parameters of feature instances. For the trace-based algorithms in Chapter 4, $\tau(n) \in O(n)$. By substitution, the lower bound on the performance of a parallelized algorithm is:

$$\Omega(\frac{n^2}{K})$$

In the worst case the complexity of the parallel algorithm will remain $O(n^2)$. While this situation can theoretically occur, most (if not all) of these pathological cases are for geometric configurations that are highly unlikely to correspond to a real-world machined part. In practice, therefore, the performance of these algorithms and their speedup from parallelization is likely to be better. For example, because basic solid modeling routines are of at least quadratic complexity in the size of the model, small reductions in the number of entities in the model (through simplification) translate into proportionally larger reductions in computational cost.

9.5 Distributed Implementation

In the distributed implementation, F-Rex runs on a cluster of SUN workstations with processes communicating over the Internet using UNIX-based and TCP/IP-protocol network software utilities and shared disk storage. The geometric computations required for task initialization are implemented with direct C++ calls to the ACIS kernel; distributed processes are invoked using UNIX remote shell commands (`rsh`); and the resulting feature set is generated by examining the features produced by each processor and eliminating redundancies.

The distributed implementation works as follows:

1. **Initialize the problem:** create, for each subproblem, (1) the set of traces, T, and (2) the simplified solid model of the part, P'. This information is written to shared disk storage.

2. **Distribute the subproblems:** use **rsh** (remote shell) to invoke serial F-Rex processes on each of the available machines with T and P' as input.

3. **Simplify:** for each trace $t \in T$ on each machine, further simplify P' using the complexity reduction techniques as described above to yield P''.

4. **Recognize:** execute the serial trace-based recognition procedure for trace t and part P''. Write recognized features (if any) to disk.

5. **Combine:** each separate process leaves a set of features F_i by writing to the disk. When all of the processes have terminated, combine all of the F_is into a single feature set and eliminate redundancies.

6. **Post-process:** perform inaccessibility check and offsetting on the recognized features.

7. **Output:** display the final feature set, $\mathcal{F}$.

The current distributed implementation is rather crude and could be made vastly more efficient by making use of inter-process and inter-machine communication tools and remote procedure calls (`rpc`'s). Currently most inter-process communication takes place using a shared disk. Ideally one would like the distributed implementation to be more transparent and adaptable—i.e., easily adjustable to variations in load on the different machines. In its current form, distributed F-Rex provides numerous computational advantages over its serial counterpart. With the incorporation of a number of basic network communication, synchronization, and data sharing algorithms, it is our belief that the performance would be vastly improved.

9.6 Examples

The data for the examples below has been collected using six processors, one SPARCStation model 10, one model 2, and 4 IPX models. In this version of the implementation, when the number of tasks is greater than 6, the tasks are distributed evenly over the available processors.

These timing results represent the elapsed clock and CPU times and are not absolute measures of the intrinsic difficulty of the feature recognition problem—this example domain is not directly comparable to those of other feature recognition efforts. Further, there are hidden costs in the implementation not directly related to the recognition of feature templates (such as feature accessibility analysis) and these algorithms and their implementation can certainly be optimized. The results are intended to provide a rough indication of the time-lag experienced by the user of the system. More significant than any precise calculation of elapsed time is the speedup factor between the serial and parallelized algorithms.

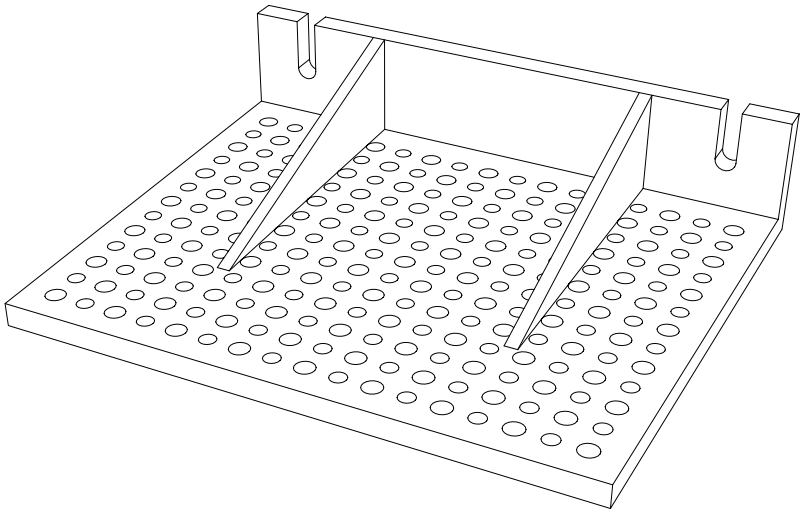


Figure 9.6: A fixture from ICEM's PART System.

Example. The example part in Figure 9.3 is a shuttle intended to move along a guideway, with many of the feature instances added to reduce weight. The solid model of this part contains 281 faces. In serial mode, F-Rex took over one hour to find the more than 100 feature instances. When running distributedly, F-Rex took 2 minutes to set up the task decomposition and approximately 32 minutes to find the features. In this case, simplification resulted in a 43% reduction in the number of geometric and topological entities that had to be considered.

Example. For the example part in Figure 8.7, when run distributedly on 6 processors, F-Rex took 10 seconds to set up the decomposition and approximately 12-16 seconds to identify the features. In this case, simplification resulted in a 35% reduction in the number of geometric and topological entities that had to be considered.

Example. The example part in Figure 9.6 is a fixture used in Control Data Corporation's ICEM PART Process Planning System. The solid model for this part contains 245 faces. When running in serial, F-Rex took over one hour to find the feature instances. In parallel, F-Rex took 1.3 minutes to set up the problem and approximately 12 minutes to recognize the features. In this case, simplification resulted in a 23% reduction in the number of geometric and topological entities that had to be considered.

9.7 Summary

This chapter developed a parallelization of the trace-based algorithms presented in Chapter 4. Basic distributed computing tools were used to implement a prototype of this parallel

approach and some experimental results were presented to illustrate the potential payoffs resulting from the use of multi-processor computing techniques.

Chapter 10

Application to Other Domains

As noted in Chapter 2, several attempts have been made to define and classify manufacturing features [22, 62, 200, 24]. Chapter 3 developed definitions for **machining features**, which in turn were used as the basis for developing the feature recognition algorithms presented in subsequent chapters. It is our belief that many of the concepts presented in this thesis in the context of machining can be applied to other manufacturing domains.

It is believed this will help to define more systematic approaches for a number of other problems in feature-based manufacturing. In this appendix, we present some brief examples.

10.1 Manufacturing Features

A **manufacturing feature** is a parameterized object describing a discrete manufacturing operation. The parameters of a feature either directly relate to or can be used to derive the parameters of the underlying manufacturing operation. The manufacturing operation associated with a particular feature is the feature's **type**. Various parameters of a feature can be assigned values from either a discrete or a continuous data set.

We will say that a feature instance f is **valid** if there exists at least one correct manufacturing plan that is realizable with available manufacturing resources and that includes f; otherwise f is **invalid**. In the domain of machined parts there are many conditions under which a feature is invalid; e.g., any volumetric feature that intersects with the final part geometry is considered invalid because including any such feature in a plan would result in cutting into the final part geometry. The set of all valid feature instances is called the **valid feature set**, $\mathcal{V}$.

Example: Machining.
As presented earlier in Chapter 3, machining features are volumes that correspond directly to the actions of individual cutting tools during a machining operation. The parameters of the feature volume can be used to calculate accessibility, tool size, tool path, and cutting speeds and feeds [70] for the machining operation that created it.

Example: Automated Assembly Planning.
One major aspect of the automated assembly planning problem is reasoning about how individual components can be configured using robotic manipulators. In the context of assembly planning, manufacturing features might correspond to gripping or mating surfaces on the individual components. For a given assembly configuration and part to be incorporated into it, there would exist a set of possible "gripping and mating features." An automated planning system can use these features to reason about whether the part can be gripped properly; if it is possible to place it into the assembly onto an appropriate set of mating surfaces; if it can be moved through space while being held on those features without interference (this is known as the motion planning problem); and if the robotic arm can extract itself from the assembly once the part is in place. In the context of assembly planning, a feature might have to contain information about geometry, topology, and spatial motion.

Similar feature types have been suggested for the domain of automated fixture planning to identify workpiece holding constraints [36].

Example: Sheet Metal Manufacturing.
Recent work by Bourne and Wang [17] proposes several categories of features for sheet metal manufacturing. Their feature types are based on the manufacturing operations required in their Intelligent Bending Workstation, including features describing bending operations and those that affect the positioning of robotic manipulators.

10.2 Feature-Based Models

We define a **feature-based model** (FBM) to be a finite set of valid feature instances $F = \{f_1, f_2, f_3, \ldots, f_n\}$ that describes a set of operations that can be used to create a part P from a piece of stock material S. More specifically, an FBM is any finite set of manufacturing features with the following properties:

1. **Sufficiency**: the features in F describe one possible way to create P from S.
2. **Necessity**: no proper subset of F creates P from S. In this way, an FBM does not contain redundant features and each feature of F contributes to the interpretation of the part.
3. **Validity**: validity would requires that each feature f in F meet manufacturability requirements. This will depend on the specific manufacturing domain.

An FBM can be thought of as a high-level, unordered operation plan. There may be many FBMs of P and S, each corresponding to a different interpretation of the part P as a set of manufacturing features F. A particular F need not model the optimal way of creating the design, as there might exist many alternatives, each corresponding to a different collection of operations that could be used to produce the design from a given piece of stock material. By analogy, just as a solid model provides a unique description of the geometry and topology of an artifact, a feature-based model provides a unique manufacturing description of an artifact.

10.3 Primary Features

Primary features represent a set of feature instances from the set of all possible valid features that are useful for reasoning and manufacturing planning. For arbitrary manufacturing domains, the set of all valid features $\mathcal{V}$ can be large (even infinite). In defining the set of **primary features**, one can impose restrictions on the set of possible features. The basic concept is that primary features, if defined properly for a given manufacturing domain, can be used and manipulated to generate and reason about all of the other feature instances of interest [70].

For an arbitrary part P there exists a set of valid features $\mathcal{V}$; we define the set $\mathcal{P}$ ($\mathcal{P} \subseteq \mathcal{V}$) of primary instances by building an equivalence relation:

1. Define the relation $\preceq$ between features as follows: given manufacturing features f and g from $\mathcal{V}$, $f \preceq g$ if f's effect on the workpiece is subsumed by g's effect on the workpiece. Intuitively this means that in the presence of g, the feature f is redundant; i.e., no feature-based model can contain both f and g.

2. The relation $\preceq$ can be used to form an equivalence relation on the set of all valid features, $\mathcal{V}$: f and g are **equivalent** ($f \simeq g$) if there exists a valid feature h of the same type as both f and g such that $f \preceq h$ and $g \preceq h$. Proof that $f \simeq g$ forms an equivalence relation will depend on the particular manufacturing domain.

3. The equivalence classes induced by $\simeq$ define the set of primary features $\mathcal{P}$; i.e., each equivalence class contains one instance of a primary feature. The specific characteristics of the primary features will depend on the particular manufacturing domain under consideration.

In the machining domain, one way to define primariness of features is by using a restricted form of volumetric maximality, as was done in this thesis. This is not the only way to define primary features for machining. The definition adopted in this thesis (and that of Gupta [70]) does have several intuitively appealing justifications of why these features are good for automated planning—in particular the fact that a primary feature volumetrically contains all features that might actually be used to produce reasonable machining plans. In this way primary features can be used to effectively prune the search space of alternative plans and to generate good upper bounds on cost.

In the context of other domains, such as assembly planning or sheet metal manufacturing, the characteristics of primariness may be less clear. Depending on the application, when performing automated planning the primary features in these domains can represent a range of possible operations—in contrast to machining, for which we adopted a definition in which the features correspond to single operations. For example, in the assembly domain a primary feature could represent the degrees of freedom and interference constraints in a motion path, in addition to the configuration of the gripper on a part.

10.4 Completeness

As presented in this thesis, the objective of **completeness** is to formally categorize when a particular approach or algorithm works well and when it does not work as well. To discuss completeness at all, one needs to develop a reasonably formal problem specification. While this may seem an obvious step, it is very difficult in practice. In particular, developing formal specifications for problems that are often arbitrarily defined in feature-based computer-integrated manufacturing is challenging.

In the context of other manufacturing applications, an analysis of completeness can be used to determine the limits of what is feasible for a fully automated system. Completeness can then be used to help refine the interface to a feature recognition system that incorporates human interaction by identifying situations that give rise to problems for automated approaches.

Chapter 11

Conclusions

This thesis has presented a systematic methodology for the development of algorithms for recognizing machining features from solid models of mechanical designs. This chapter summarizes the research contributions and outlines a number of research issues for the future.

11.1 Research Contributions

This thesis has attempted to advance the state-of-the-art in automated feature recognition on several fronts. In particular:

Developed an implementation-independent definition for which features to recognize. The approach in this thesis is to keep the definition of a feature class and the features to be recognized distinct from the choice of a feature recognition algorithm. It is my belief that the features defined in Chapter 3 can be used in conjunction with a number of different recognition methodologies. Chapter 4 of this thesis applies trace-based techniques to build recognition algorithms based on the specifications in Chapter 3.

Presented an approach for measuring the complexity of application-level CAD algorithms. Previous research on CAD/CAM and solid modeling applications lacks analyses of the complexity of the problems being solved. While there have been some notable efforts at measuring the complexity of individual approaches to feature recognition [40, 39, 55, 139, 38], none have presented a general way of measuring the complexity of algorithms built on top of the existing infrastructure provided by a solid modeling system.

I have proposed in this thesis that, given a means of specifying the problem to be solved, meaningful measures of the complexity of the real-world problem can be taken independent of the implementation. In particular, complexity can be measured in terms of the number of solid modeling operations. I believe that this technique can be easily applied to other domains where the level of algorithmic abstraction is several levels above the fundamental data structures.

Furthered the development of trace-based feature recognition. In the development of my traces and feature construction methods I have advanced the functionality and scope of trace-based recognition systems. In this area I have followed the lead of the research of Vandenbrande [200], in particular focusing on extending some of the traces he outlined to include cases not covered by his original work as well as incorporating traces for new feature types.

Presented feature-recognition as an interface to multiple downstream applications. Most previous work in feature recognition was developed with a single application goal in mind—most often automated process planning of machined parts. In this work I have illustrated that different downstream applications entail different requirements for a feature recognition system. The features most useful for process planning might not be the most useful for automated redesign.

My approach has been successfully used to develop methodologies for automated manufacturability analysis and redesign, problems similar in nature to the process planning problem yet with subtly different needs. I have shown that by separating feature recognition from specific applications I can build a more generic tool with a broader range of utility.

Introduced techniques for reasoning about completeness. Completeness and soundness have emerged as vital issues for system integration and for controlling computational complexity. In particular, I have introduced notation and a degree of formalism that can be used to specify the computational feature recognition problem in terms of its input and output. Previous work had not satisfactorily addressed this issue, resorting either to descriptions in terms of low level data structures in restricted toy domains or to vague descriptions based on the implementation.

Introduced parallelization and distributed algorithms to feature recognition. Existing work in feature recognition is exclusively serial in nature. I have shown that, using current technology and software tools, effective and useful parallelization can be achieved for computationally intensive problems in CAD/CAM such as feature recognition. I feel that this area has huge potential to radically change the way systems are designed and implemented. Whereas previously monolithic systems were constructed on a single desktop, I can now exploit the maximum resources the computer network can provide and design software to harness the power of the network rather than just an individual machine. In doing this I can achieve not only computational speedups but also reductions in the complexity of the individual feature recognition subproblems.

11.2 Anticipated Impact

It is expected that this research will enhance our formal understanding of the basic computational issues that lie behind the feature recognition problem. In this way, improved and more

rigorous feature recognition systems can be used as components in the next generation of CAD/CAM systems. In addition, this research reveals a gap at the application level between the development of theory and the development of CAD/CAM systems. It is believed that further research on the specification, completeness, and complexity of geometric problems in design and manufacturing can enhance our understanding about basic problems and lead to the production of better CAD/CAM software tools.

11.3 Recommendations for Future Work

To build a scalable and practical feature recognition system for real-world parts and integrate engineering design with downstream activities, a number of extensions to this research will be needed. In particular, these are some recommendations for future research:

1. **Integration of multiple feature recognition techniques into a single system.**

   Some approaches to feature recognition are good for finding high-level features, others for more detailed features; some are fast, some are more comprehensive. A great potential payoff exists if multiple feature recognition techniques can be effectively integrated into a single system. Such a "feature recognition cocktail" could operate at several levels of detail, depending on the situation and the application. It is my belief that F-Rex can be one component of such an integrated software solution and, instead of building all of these systems in isolation, a more holistic approach should be taken—one that incorporates the best aspects of each into one comprehensive feature recognition system.

2. **Shape and manufacturing similarity assessment.**

   The proliferation of CAD and solid modeling in industry has created an emerging problem: how is the CAD data stored in the corporate database? An obvious application might be for a designer to query the corporate database with a solid model of a preliminary design and ask "is there a design in the database that is similar to this one?" Traditional database indexing schemes do not lend themselves well to the indexing of solid models.

   I feel that feature recognition can be used to generate feature-based indexes for storage and retrieval of parts. Feature-based part indexing can have a number of advantages over existing approaches (such as GT coding), including the ability to index based on a space of different possible machining plans by indexing feature-based models.

3. **Move features beyond geometry.**

   Currently, features are tied to geometry and topology and, in most work, they are tied to machining as a manufacturing process. In the future, features will need to support multiple manufacturing domains and manufacturing processes. For example, planning for fixturing, machining, and assembly requires new types of feature-based

representations. Features might include information about the design's history and the designer's intent; as well as information about the conceptual design and the design's functional specification. Such higher level features will require feature definitions that are symbolic rather than merely numeric.

4. **Real-world parts.**

   As has been mentioned often in recent press, academic research needs to better understand the problems of industry. To this end, improved feature recognition research requires better and more realistic parts and fewer "simple bracket" examples, such as the one in Figure 3.1. Methods that work well on isolated simple features and parts for which there are only a few possible plans may prove difficult to scale to real-world parts. For example, the research community should begin to focus on a common collection of benchmark parts; parts with many curved surfaces and dozens of feature interactions, requiring multiple machining setups and incorporating interacting tolerance and precedence constraints.

5. **Improved complexity measurements.**

   Design is an interactive process and, for many potential applications, response time and computational complexity are vital issues for feature recognition systems. Existing work, while easily demonstrated on simple examples, might not provide acceptable results when faced with the complexity and ambiguity of the artifacts of the real world. Based on the work in this thesis, I believe that with careful assessment of the domain of interest, most feature recognition problems in design and manufacturing are not inherently intractable. The problem of optimal operation planning with features, however, is likely to be provably NP-hard. Better and more precise means of measuring the complexity of algorithms (such as those in Chapter 4) developed at the application level of a solid modeling system are needed.

6. **The role of human supervision.**

   The role of human supervision in the feature recognition process has not been rigorously addressed by the academic research community. What is likely to emerge is that there will be classes of features that are easy to identify through automated means and other classes of features (some of which will be vital for certain applications) that are difficult or impossible to recognize through automated means. In these situations an analysis of the completeness of automated methods can be used to find the dividing line between feasible and unidentifiable features. The issue then becomes one of how to (1) automatically identify what types of geometric configurations may contain problematic feature instances and (2) build a powerful yet pleasant user interface for a qualified human user to manually pick out the necessary feature instances. The role of the human being will be an integral component of the feature recognition system of the future.

7. **Development of a "science of features."**

The definitions presented in this thesis are an initial attempt to mathematically describe feature recognition as a computational problem in design and manufacturing. Many of the engineering concepts described in this thesis are difficult to define mathematically. While the definitions presented here proved an adequate basis on which to develop my approach, additional work will be necessary to improve and enhance the mathematical tools for describing engineering information.

It is my belief that research in this area must proceed along two lines:

(a) Additional work is needed to improve our understanding of the nature of engineering information. Features are still very much a vague "catch all" for describing ad hoc manufacturing knowledge. Continued study and categorization of engineering information and knowledge is required, coupled with the development of better representation schemes for modeling this data. Given more rigorous definitions, more formal proofs will be possible about the computability and complexity of manufacturing problems.

(b) Given more comprehensive representations of engineering information, the scope of the feature recognition problem will expand to include all types of mappings between different manufacturing viewpoints. This will require whole new classes of algorithms to be developed that can extend the features concept beyond simple geometric shapes to be the alphabet for an ontalingua that can describe engineering knowledge.

Bibliography

[1] Ableson and Sussmann. *Structure and Interpretation of Computer Programs.* MIT Press and McGraw-Hill, Cambridge, MA and New York, NY, 1985. ISBN 0-262-01077-1.

[2] Alfred Aho, Ravi Sethi, and Jeffery Ullman. *Compilers: Principles, Techniques, and Tools.* Addison-Wesley Publishing Company, MA, 1986.

[3] Alfred V. Aho, John E. Hopcroft, and Jeffery D. Ullman. *Data Structures and Algorithms.* Addison-Wesley Publishing Company, Reading, MA, 1983.

[4] Seshagiri Rao Ala. Design methodology of boundary data structures. In Jaroslaw Rossignac and Joshua Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 13–23, New York, NY 10036, USA, Austin, TX, June 1991. ACM SIGGRAPH, ACM Press.

[5] Seshagiri Rao Ala. Performance anomalies in boundary data structures. *IEEE Computer Graphics and Applications*, 12(2):49–58, March 1992.

[6] George Almasi, Raghu Karinthi, and Kankanahalli Srinivas. A parallel algorithm for computing set operations on loops. Technical Report TR 93-10, Department of Statistics and Computer Science, West Virginia University, August 1993.

[7] J. Altemueller. Mapping from EXPRESS to physical file structure. Technical Report ISO TC184/SC4 Document N280, International Organization for Standardization, 1988. September.

[8] J. Altemueller. The STEP file structure. Technical Report ISO TC184/SC4 Document N279, International Organization for Standardization, 1988. September.

[9] 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.

[10] Arlo L. Ames. Production ready feature recognition based automatic group technology part coding. In Jaroslaw Rossignac and Joshua Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 161–169, New York, NY 10036, USA, Austin, TX, June 1991. ACM SIGGRAPH, ACM Press.

[11] D. C. Anderson, M. R. Henderson, and S. M. Staley. Using syntactic pattern recognition to extract feature information from a solid geometric data base. *Computers in Mechanical Engineering*, September 1983.

[12] Raja P. K. Banerjee, Vineet Goel, and Amar Mukherjee. Efficient parallel evaluation of CSG trees using fixed number of processors. In Jaroslaw Rossignac, Joshua Turner, and George Allen, editors, *Second Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 313–322, New York, NY 10036, USA, May 1993. ACM SIGGRAPH, ACM Press. Montreal, Canada.

[13] Carl Bass, Milton Capsimalis, Robert Covey, Azhar Kahn, Jeffrey Kowalski, Ron Levine, Brian Mathews, James Merry, Eve Podet, Scott Sheppaard, and Garry Wiegand. *HOOPS Graphics System Reference Manual.* Ithaca Software, 1301 Marina Village Parkway, Alameda CA 94501, v4.0 edition, June 30 1994. Publication #05202-010000-5010.

[14] Robert Berger. The undecidability of the domino problem. *Memoriors of the American Mathematical Society*, (66), 1966. Providence, RI.

[15] G. Boothroyd and P. Dewhurst. *Design for Assembly -A Designer's Handbook.* Department of Mechanical Engineering, University of Massachusetts at Amherst, 1983.

[16] Geoffrey Boothroyd. Product design for manufacture and assembly. *Computer Aided Design*, 26(9):505–520, 1994.

[17] David A. Bourne and Cheng hua Wang. Design and manufacturing of sheet metal parts: Using features to resolve manufacturability problems. In A. A. Busnaina, editor, *ASME Computers in Engineering Conference*, pages 745–753, New York, NY 10017, September 17-20, Boston, MA 1995. ASME.

[18] Adrian Bowyer and John Woodwark. *A Programmer's Geometry.* Butterworths, London, 1983.

[19] Gilles Brassard and Paul Bratley. *Algorithms: Theory and Practice.* Prentice Hall Incorporated, NJ, 1988.

[20] S. L. Brooks and M. L. Wolf. Overview of Allied Signal's XCUT system. In Jami Shah, Martti Mäntylä, and Dana Nau, editors, *Advances in Feature Based Manufacturing.* Elsevier/North Holland, 1994.

[21] C. M. Brown. PADL-2: A technical summary. *IEEE Computer Graphics and Applications*, 2(2):69–84, March 1982.

[22] W. Butterfield, M. Green, D. Scott, and W. Stoker. Part features for process planning. Technical Report R-86-PPP-01, Computer Aided Manufacturing–International, Arlington, TX, USA, November 1986.

[23] Gene Bylinsky. The digital factory. *Fortune*, pages 92–110, November, 14 1994.

[24] Tien-Chien Chang. *Expert Process Planning for Manufacturing.* Addison-Wesley Publishing Co., Reading, Massachusetts, 1990.

[25] Tien-Chien Chang and Richard A. Wysk. *An Introduction to Automated Process Planning Systems.* Prentice-Hall, 1985.

[26] S. H. Chuang and M. R. Henderson. Three-dimensional shape pattern recognition using vertex classification and the vertex-edge graph. *Computer Aided Design*, 22(6):377–387, June 1990.

[27] S. H. Chuang and M. R. Henderson. Compound feature recognition by web grammar parsing. *Research in Engineering Design*, 2(3):147–158, 1991.

[28] J. K. Coles, R. H. Crawford, and K. L. Wood. Form feature recognition using base volume decomposition. In *ASME Advances in Design Automation Conference*, pages 281–297. ASME, September 1994.

[29] Control Data Corporation. *ICEM PART Reference Manual*, July 1994. Version 1.2.

[30] J. Corney and D. E. R. Clark. Method for finding holes and pockets that connect multiple faces in $2\frac{1}{2}$D objects. *Computer Aided Design*, 23(10):658–668, December 1991.

[31] J. Corney and D. E. R. Clark. Face based feature recognition: Generalizing special cases. *International Journal of Computer Integrated Manufacturing*, 6(1 & 2):39–50, 1993.

[32] J. Corney and D.E.R. Clark. A feature recognition algorithm for multiply connected depressions and protrusions in $2\frac{1}{2}$D objects. In Jaroslaw Rossignac and Joshua Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 161–169, New York, NY 10036, USA, June 1991. ACM SIGGRAPH, ACM Press. Austin, TX.

[33] J. Corney and D.E.R. Clark. Efficient face-based feature recognition. In Jaroslaw Rossignac, Joshua Turner, and George Allen, editors, *Second Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 313–322, New York, NY 10036, USA, May 1993. ACM SIGGRAPH, ACM Press. Montreal, Canada.

[34] Jonathan R. Corney. *Graph-Based Feature Recognition.* PhD thesis, Heriot-Watt University, Edinburgh, England, October 1993.

[35] M. R. Cutkosky and J. M. Tenenbaum. Toward a framework for concurrent design. *International Journal of Systems Automation: Research and Applications*, 1(3):239–261, 1992.

[36] Diganta Das, Satyandra K. Gupta, and Dana S. Nau. Estimation of setup time for machined parts: Accounting for work-holding constraints using a vise. In A. A. Busnaina, editor, *ASME Computers in Engineering Conference*, pages 619–631, New York, NY 10017, September 17-20, Boston, MA 1995. ASME.

[37] Diginta Das, Satyandra K. Gupta, and Dana S. Nau. Reducing setup cost by automated generation of redesign suggestions. In Kosuke Ishii, editor, *ASME Computers in Engineering Conference*, pages 159–170. ASME, September 1994.

[38] Parag Dave and Hiroshi Sakurai. Maximal volume decomposition and recognition of intersecting features. In A. A. Busnaina, editor, *ASME Computers in Engineering Conference*, pages 553–568, New York, NY 10017, September 17-20, Boston, MA 1995. ASME.

[39] Leila De Floriani. Feature extraction from boundary models of three-dimensional objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(8), August 1989.

[40] Leila De Floriani and Bianca Falcidieno. A hierarchical boundary model for solid object representation. *ACM Transactions on Graphics*, 7(1):42–60, January 1988.

[41] Xin Dong. *Geometric Feature Extraction for Computer-Aided Process Planning.* PhD thesis, Rensselaer Polytechnic Institute, Troy, NY, USA, 1988.

[42] Xin Dong and Michael Wozny. FRAFES, a frame-based feature extraction system. In *International Conference on Computer Integrated Manufacturing*, pages 296–305. IEEE, May 23-25, Troy NY 1988.

[43] Xin Dong and Michael Wozny. A method for generating volumetric features from surface features. In Jaroslaw Rossignac and Joshua Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 161–169, New York, NY 10036, USA, June 1991. ACM SIGGRAPH, ACM Press. Austin, TX.

[44] S. Drake and S. Sela. A foundation for features. *Mechanical Engineering*, 111(1):66–73, 1989.

[45] D. D. Dunlop and V. R. Basili. A comparitive analysis of functional correctness. *Computing Surveys*, 14(2):229–244, June 1982.

[46] Mark Dunn. Industrial automation systems — product data representation and exchange — part 48: Integrated generic resources: Form features. Technical Report ISO/WD 10303-48, International Organization for Standardization, January 2 1992. Working draft.

[47] R. Kent Dybvig. *The Scheme Programming Language.* Prentice Hall, 1987.

[48] Ekstrom, Carlson & Co., Rockford, IL 61110. Cutting tools, 1992. Catalog #ST-92.

[49] J. L. Ellis, G. Kedem, T. C. Lyerly, D. G. Thielman, R. J. Marisa, P. J. Menon, and H. B. Voelcker. The RayCasting Engine and ray representations. In Jaroslaw Rossignac and Joshua Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 255–267, New York, NY 10036, USA, Austin, TX, June 1991. ACM SIGGRAPH, ACM Press.

[50] Margaret A. Ellis and Bjarne Stroustrup. *The Annotated C++ Reference Manual.* Addison-Wesley Publishing Company, May 1991.

[51] H. A. ElMaraghy, K. F. Zhang, and H. Chu. A function-oriented modeler prototype. In P. J. Guichelaar, editor, *Design for Manufacturability, ASME Winter Annual Meeting*, pages 57–62. ASME, 1993.

[52] Hoda A. ElMaraghy and Waguih H. ElMaraghy. Computer-aided inspection planning. In Jami Shah, Martti Mäntylä, and Dana Nau, editors, *Advances in Feature Based Manufacturing.* Elsevier/North Holland, 1994.

[53] Bianca Falcidieno and Franca Giannini. Automatic recognition and representation of shape-based features in a geometric modeling system. *Computer Vision, Graphics, and Image Processing*, 48:93–123, 1989.

[54] I. D. Faux and M. J. Pratt. *Computational Geometry for Design and Manufacture.* Ellis Horwood, 1979.

[55] M. C. Fields and D. C. Anderson. Fast feature extraction for machining applications. *Computer Aided Design*, 26(11), November 1994.

[56] James D. Foley, Andries van Dam, Steven K. Feiner, and John F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley Publishing Company, Reading, MA, second edition, 1990.

[57] K. S. Fu. *Syntactic Pattern Recognition and Applications.* Prentice Hall Incorporated, NJ, 1982.

[58] R. Gadh and F. B. Prinz. Recognition of geometric forms using the differential depth filter. *Computer Aided Design*, 24(11):583–598, November 1992.

[59] J. D. Gannon, R. B. Hamlet, and H. D. Mills. Theory of modules. *Transactions on Software Engineering*, 13(7):820–829, July 1987.

[60] P. Gavankar and M. R. Henderson. Graph-based extraction of protrusions and depressions from boundary representations. *Computer Aided Design*, 22(7):442–450, September 1990.

[61] W. Wayt Gibbs. Software's chronic crisis. *Scientific American*, 271(3), September 1994.

[62] N. N. Z. Gindy. A hierarchical structure for form features. *International Journal of Production Research*, 27(12):2089–2103, 1989.

[63] Keith E. Gorlen. *The NIH Class Library Reference Manual.* The National Institutes of Health, Bethesda, Maryland, April 1990. Revision 3.10.

[64] Keith E. Gorlen, Sanford M. Orlow, and Perry S. Plexico. *Data Abstraction and Object Oriented Programming in C++.* John Wiley & Sons, 1990. ISBN 0471 92346 X.

[65] Diganta Das Satyandra K. Gupta and Dana S. Nau. Generating redesign suggestions to reduce setup cost: A step towards automated redesign. *Computer Aided Design*, 1995. Also available as University of Maryland ISR TR 95:39,CS-TR-3439, UMIACS-TR-95-36.

[66] S. K. Gupta, Diganta Das, William C. Regli, and Dana S. Nau. Current trends and future challenges in automated manufacturability analysis. In *Proceedings of ASME International Computers in Engineering Conference*, 1995. Also available as University of Maryland ISR TR 95:16.

[67] S. K. Gupta and D. S. Nau. A systematic approach for analyzing the manufacturability of machined parts. *Computer Aided Design*, 1995. To appear.

[68] S. K. Gupta, D. S. Nau, W. C. Regli, and G. Zhang. A methodology for systematic generation and evaluation of alternative operation plans. In Jami Shah, Martti Mäntylä, and Dana Nau, editors, *Advances in Feature Based Manufacturing.* Elsevier/North Holland, 1994.

[69] S. K. Gupta, P. N. Rao, and N. K. Tewari. Development of a CAPP system for prismatic parts using feature based design concepts. *The International Journal of Advanced Manufacturing Technology*, 7:306–313, 1992.

[70] Satyandra K. Gupta. *Automated Manufacturability Analysis of Machined Parts.* PhD thesis, The University of Maryland, College Park, MD, 1994.

[71] Satyandra K. Gupta, Thomas R. Kramer, Dana S. Nau, William C. Regli, and Guangming Zhang. Building MRSEV models for CAM applications. *Advances in Engineering Software*, 20(2/3):121–139, 1994.

[72] Satyandra K. Gupta, William C. Regli, and Dana S. Nau. Manufacturing feature instances: Which ones to recognize? In Jaroslaw Rossignac, Joshua Turner, and George Allen, editors, *Third Symposium on Solid Modeling Foundations and CAD/CAM Applications*, New York, NY, USA, May 17-19 1995. ACM SIGGRAPH and the IEEE Computer Society, ACM Press. Salt Lake City, Utah.

[73] JungHyun Han and Aristides A. G. Requicha. Incremental recognition of machining features. In Kosuke Ishii, editor, *ASME Computers in Engineering Conference*, pages 143–150. ASME, September 1994.

[74] JungHyun Han and Aristides A. G. Requicha. Integration of feature-based design and feature recognition. In A. A. Busnaina, editor, *ASME Computers in Engineering Conference*, pages 569–578, New York, NY 10017, September 17-20, Boston, MA 1995. ASME.

[75] Frank Harary. *Graph Theory.* Addison-Wesley, Reading, MA, 1969.

[76] C. C. Hayes, S. Desa, and P. K. Wright. Using process planning knowledge to make design suggestions concurrently. In N. H. Chao and S. C. Y. Lu, editors, *Concurrent Product and Process Design, ASME Winter Annual Meeting*, pages 87–92. ASME, 1989.

[77] Mark R. Henderson. *Extraction of Feature Information from Three-Dimensional CAD Data.* PhD thesis, Purdue University, West Lafayette, IN, USA, 1984.

[78] Mark R. Henderson. Representing functionality and design intent in product models. In Jaroslaw Rossignac, Joshua Turner, and George Allen, editors, *Proceedings of the Second Symposium on Solid Modeling and Applications*, pages 387–396. ACM SIGGRAPH and IEEE computer Society, 1993.

[79] Mark R. Henderson and LeRoy E. Taylor. A meta-model for mechanical products based upon the mechanical design process. *Research in Engineering Design*, 5(4):140–160, 1993.

[80] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informtica*, 1(4):271–281, 1972.

[81] Christoph M. Hoffman. *Geometric and Solid Modeling: An Introduction.* Morgan Kaufmann Publishers Incorporated, CA, 1989.

[82] John E. Hopcroft and Jeffery D. Ullman. *Intoduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, MA, 1979.

[83] Wynne Hsu, C. S. George Lee, and S. F. Su. Feedback approach to design for assembly by evaluation of assembly plan. *Computer Aided Design*, 25(7):395–410, July 1993.

[84] Yong-Jung Huh and Sang-Gook Kim. A knowledge-based CAD system for concurrent product design in injection molding. *International Journal of Computer Integrated Manufacturing*, 4(4):209–218, 1991.

[85] K. Ishii, C. F. Eubanks, and D. Houser. Evaluation methodology for post manufacturing issues in life-cycle design. *International Journal of Concurrent Engineering: Research and Applications*, 1(1):61–68, 1993.

[86] Kosuke Ishii. Modeling of concurrent engineering design. In Andrew Kusiak, editor, *Concurrent Engineering: Automation, Tools and Techniques, ASME Winter Annual Meeting*, pages 19–39. John Wiley & Sons, Inc., 1993.

[87] Kosuke Ishii. Life-cycle engineering design: Research overview. In *Proceedings of the 1994 NSF Design and Manufacturing Grantees Conference*, pages 41–42, Cambridge, MA, January 1994. NSF.

[88] M. Jakiela and P. Papalambros. Design and implementation of a prototype intelligent CAD system. *ASME Journal of Mechanisms, Transmission, and Automation in Design*, 111(2), June 1989.

[89] Ryszard Jakubowski. Syntactic characterization of machine parts shapes. *Cybernetics and Systems: An International Journal*, 13(1):1–24, 1982.

[90] Ryszard Jakubowski. Extraction of shape features for syntactic recognition of mechanical parts. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-15(5), September/October 1985.

[91] S. Joshi and T. C. Chang. Graph-based heuristics for recognition of machined features from a 3D solid model. *Computer-Aided Design*, 20(2):58–66, March 1988.

[92] Frederick P. Brooks Jr. *The Mythical Man-Month.* Addison-Wesley, 1975.

[93] Kevin K. Jurrens, James E. Fowler, and Mary Elizabeth A. Algeo. Modeling of manufacturing resource information: Requirements specification. Technical Report NISTIR 5707, National Institute of Standards and Technology, Gaithersburg, MD, 20899, July 1995.

[94] R. Karinthi and D. Nau. An algebraic approach to feature interactions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(4):469–484, April 1992.

[95] R. Karinthi, D. Nau, and Q. Yang. Handling feature interactions in process planning. *Applied Artificial Intelligence*, 1992. Special issue on AI for manufacturing.

[96] R. R. Karinthi and D. S. Nau. An algebraic approach to feature interactions. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 14(4):469–484, April 1992.

[97] Raghu Karinthi, Kankanahalli Srinivas, and George Almasi. A parallel algorithm for computing polygon set operations. Technical Report TR 93-4, Department of Statistics and Computer Science, West Virginia University, April 1993.

[98] Raghu R. Karinthi. *An Algebraic Approach to Feature Interactions.* PhD thesis, The University of Maryland, College Park, MD, USA, 1990.

[99] Kennametal Inc., P.O. Box 346, Latrobe, PA 15650-0346. Milling '87, 1986.

[100] Richard Kenyon. Tiling a polygon with parallelograms. *Algorithmica*, 9(4):382–397, April 1993.

[101] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, second edition, 1988.

[102] Y. S. Kim. Recognition of form features using convex decomposition. *Computer Aided Design*, 24(9):461–476, September 1992.

[103] Yong Se Kim and D. J. Wilde. A convergent convex decomposition of polyhedral objects. *Transactions of the ASME*, 114:468–476, September 1992.

[104] Mark Klein. Capturing design rationale in concurrent engineering teams. *Computer*, 26(1):39–47, 1993.

[105] Thomas R. Kramer. A parser that converts a boundary representation into a features representation. *International Journal of Computer Integrated Manufacturing*, 2(3):154–163, 1989.

[106] Thomas R. Kramer. An express schema for machining plugged into ALPS4. Technical report, The National Institute of Standards and Technology, Gaithersburg, MD 20899, January 1991.

[107] Thomas R. Kramer. Issues regarding material removal shape element volumes (MRSEVs). Technical Report NISTIR 4804, The National Institute of Standards and Technology, Gaithersburg, MD 20899, March 1992.

[108] Thomas R. Kramer. A library of material removal shape element volumes (MRSEVs). Technical Report NISTIR 4809, The National Institute of Standards and Technology, Gaithersburg, MD 20899, March 1992.

[109] Lycourgos K. Kyprianou. *Shape Classification in Computer Aided Design*. PhD thesis, Christ College, University of Cambridge, Cambridge, United Kingdom, 1980.

[110] Timo Laakko and Martti Mäntylä. Feature modelling by incremental feature recognition. *Computer Aided Design*, 25(8):479–492, August 1993.

[111] Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall Incorporated, NJ, 1981.

[112] Don Libes. *Exploring Expect*. O'Reilly & Associates, 1994.

[113] Stanley B. Lippman. *C++ Primer*. Addison-Wesley, second edition, 1991.

[114] Jacques Loeckx and Kurt Sieber. *The Foundations of Program Verification*. John Wiley and Sons, New York, NY, USA, second edition, 1987.

[115] M. Mäntylä, J. Opas, and J. Puhakka. Generative process planning of prismatic parts by feature relaxation. Technical report, Helsinki Institute of Technology, Laboratory of Information Processing Science, Finland, Feb 1989.

[116] Martti Mäntylä. *An Introduction to Solid Modeling.* Computer Science Press, College Park, MD, 1988.

[117] M. Marefat and R. L. Kashyap. Geometric reasoning for recognition of three-dimensional object features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):949–965, October 1990.

[118] Michael Marefat and R. L. Kashyap. Automatic construction of process plans from solid model representations. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(5):1097–1115, September/October 1992.

[119] Michael M. Marefat. Feature-based computer integrated inspection. In Fatih Kinoglu, editor, *ASME Computers in Engineering Conference*, pages 145–152. ASME, August 1993.

[120] Edward C. Martin. *Getting Started with Scheme Using the ACIS 3D Toolkit.* Spatial Technology Inc., 2425 55th Street, Building A, Boulder, CO 80301, v1.1 edition, Feburary 1994.

[121] R. R. Martin and P. C. Stephenson. Sweeping of three-dimensional objects. *Computer Aided Design*, 22(4):223–234, May 1990.

[122] S. Meeran and M. J. Pratt. Automated feature recognition from 2d drawings. *Computer Aided Design*, 25(1):7–17, January 1993.

[123] Sreekumar Menon and Yong Se Kim. Handling blending features in form feature recognition using convex decomposition. In Kosuke Ishii, editor, *ASME Computers in Engineering Conference*, pages 79–92. ASME, September 1994.

[124] James R. Miller. Architectural issues in solid modelers. *IEEE Computer Graphics and Applications*, 9(5):72–87, September 1989.

[125] H. D. Mills. The new math of computer programming. *Communications of the ACM*, 18(1):43–48, January 1975.

[126] H. D. Mills, V. R. Basili, J. D. Gannon, and R. G. Hamlet. *Principles of Computer Programmings: A Mathematical Approach.* Allyn and Bacon, 1987.

[127] Harlan D. Mills. *Software Productivity.* Dorset House Publishing, New York, NY, USA, 1988.

[128] Michael E. Mortenson. *Geometric Modeling.* John Wiley & Sons, New York, NY, 1985.

[129] Chandrasekhar Narayanaswami and William R. Franklin. Determination of mass properties of polygonal csg objects in parallel. In Jaroslaw Rossignac and Joshua Turner, editors, *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, pages 255–267, New York, NY 10036, USA, Austin, TX, June 1991. ACM SIGGRAPH, ACM Press.

[130] D. S. Nau. Automated process planning using hierarchical abstraction. *TI Technical Journal*, pages 39–46, Winter 1987.

[131] D. S. Nau, G. Zhang, S. K. Gupta, and R. R. Karinthi. Evaluating product machinability for concurrent engineering. In W. G. Sullivan and H. R. Parsaei, editors, *Concurrent Engineering: Contemporary Issues and Modern Design Tools*, pages 264–279. Chapman and Hall, 1993.

[132] John K. Ousterhout. Tcl: An embeddable command language. In *Winter USENIX Conference*, 1990.

[133] John K. Ousterhout. An X11 toolkit based on the Tcl language. In *Winter USENIX Conference*, 1991.

[134] John K. Ousterhout. Tcl/Tk tutorial. In *Winter USENIX Conference*, 1993.

[135] John K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, 1994.

[136] Frederic Pariente and Yong Se Kim. Incremental and localized update of convex decomposition for form feature decomposition. In A. A. Busnaina, editor, *ASME Computers in Engineering Conference*, pages 587–598, New York, NY 10017, September 17-20, Boston, MA 1995. ASME.

[137] Yun Peng and James A. Reggia. Diagnostic problem-solving with causal chaining. *International Journal of Intelligent Systems*, 2:265–302, 1987.

[138] Thomas J. Peters. Encoding mechanical design features for recognition via neural nets. *Research in Engineering Design*, 4(2):67–74, 1992.

[139] Thomas J. Peters. Mechanical design heuristics to reduce the combinatorial complexity for feature recognition. *Research In Engineering Design*, 4:195–201, 1993.

[140] Henry Petroski. Failed promises. *American Scientist*, 82(1):6–9, January-February 1994.

[141] J. Miguel Pinilla, Susan Finger, and Friedrich B. Prinz. Shape feature description using an augmented topology graph grammar. In *Proceedings NSF Engineering Design Research Conference*, pages 285–300. National Science Foundation, June 1989.

[142] S. Prabhakar and M. R. Henderson. Automatic form-feature recognition using neural-network-based techniques on boundary representations of solid models. *Computer Aided Design*, 24(7):381–393, July 1992.

[143] M. J. Pratt and P. R. Wilson. Requirements for support of form features in a solid modeling system. Technical Report R-85-ASPP-01, Computer Aided Manufacturing International, Arlington, TX, 1985.

[144] Michael J. Pratt. Tutorial paper on advanced topics in solid modeling: Form features and their application in solid modeling. In *SIGGRAPH 1987*. The Association for Computing Machinery, ACM Press, 1987.

[145] Paul W. Purdom and Cynthia A. Brown. *The Analysis of Algorithms*. Holt, Rinehart, and Winston, New York, NY, 1985.

[146] Valerie Quercia and Tim O'Reilly. *X Window System User's Guide*, volume 3. O'Reilly & Associates, Inc., x11 r3 and r4 edition, 1990.

[147] J.A. Reggia, D.S. Nau, and P.Y. Wang. A formal model of diagnostic inference. II. algorithmic solution and applications. *Information Sciences*, 37:257–285, 1985.

[148] William C. Regli, Satyandra K. Gupta, and Dana S. Nau. Extracting alternative machining features: An algorithmic approach. *Research in Engineering Design*, 7(3):173–192, 1995.

[149] Aristides A. G. Requicha. Representation for rigid solids: Theory, methods, and systems. *Computing Surveys*, 12(4):437–464, December 1980.

[150] Raphael M. Robinson. Undecidability and nonperiodicity for tilings of the plane. *Inventiones Mathematicae*, 12:177–209, 1971.

[151] David W. Rosen, John R. Dixon, and Susan Finger. Conversions of feature-based representations via graph grammar parsing. In *AMSE Design Theory Methodology Conference*, 1992.

[152] D.W. Rosen and J.R. Dixon. Languages for feature-based design and manufacturability evaluation. *Systems Automation Research and Applications*, 2(4):353–373, 1992.

[153] Jaroslaw Rossignac and Joshua Turner, editors. *Symposium on Solid Modeling Foundations and CAD/CAM Applications*, New York, NY, USA, June 5-7 1991. ACM SIGGRAPH, ACM Press. Austin, Texas.

[154] Jaroslaw Rossignac, Joshua Turner, and George Allen, editors. *Second Symposium on Solid Modeling and Applications*, New York, NY, USA, May 19-21 1993. ACM SIGGRAPH and the IEEE Computer Society, ACM Press. Montreal, Canada.

[155] Jaroslaw Rossignac, Joshua Turner, and George Allen, editors. *Thrid Symposium on Solid Modeling and Applications*, New York, NY, USA, May 17-19 1995. ACM SIGGRAPH and the IEEE Computer Society, ACM Press. Salt Lake City, Utah.

[156] Scott A. Safier and Susan Finger. Parsing features in solid geometric models. In *European Conference on Artificial Intelligence*, 1990.

[157] Hiroshi Sakurai. Generating volumes from faces of a solid model. In *NSF Design and Manufacturing Systems Conference*, January 1993.

[158] Hiroshi Sakurai. Decomposing a delta volume into maximal convex volumes and sequencing them for machining. In Kosuke Ishii, editor, *ASME Computers in Engineering Conference*, pages 135–142. ASME, September 1994.

[159] Hiroshi Sakurai and Chia-Wei Chin. Defining and recognizing cavity and protrusion by volumes. In Fatih Kinoglu, editor, *ASME Computers in Engineering Conference*, pages 59–65, August 1993.

[160] Hiroshi Sakurai and Chia-Wei Chin. Definition and recognition of volume features for process planning. In Jami Shah, Martti Mäntylä, and Dana Nau, editors, *Advances in Feature Based Manufacturing*, chapter 4, pages 65–80. Elsevier/North Holland, 1994.

[161] Hiroshi Sakurai and David C. Gossard. Recognizing shape features in solid models. *IEEE Computer Graphics & Applications*, September 1990.

[162] Otto W. Salomons, Fred J. A. M. van Houten, and H. J. J. Kals. Review of research in feature-based design. *Journal of Manufacturing Systems*, 12(2):113–132, 1993.

[163] H. Samet. The quadtree and related hierarchial data structures. *ACM Computing Surveys*, 16(3):287–260, 1984.

[164] Sandvik Coromant, 1993. Catalog CMP90-R93.3.

[165] Sandvik Coromant, 1994. Catalog CMP90-R94.2.

[166] D. Schenck. Exchange of product model data – part 11: The EXPRESS language. Technical Report ISO TC184/SC4 Document N64, International Organization for Standardization, 1990. July.

[167] J. M. Schmitz and S. Desa. The application of a design for producibility methodology to complex stamped products. In N. H. Chao and S. C. Y. Lu, editors, *Concurrent Product and Process Design, ASME Winter Annual Meeting*, pages 169–174. ASME, 1989.

[168] Michael Schulte, Christian Weber, and Rainer Stark. Functional features for design in mechanical engineering. *Computers in Industry*, 23(1):15–24, 1993.

[169] J. Shah, Y. Shen, and A. Shirur. Determination of machining volumes from extensible sets of design features. In Jami Shah, Martti Mäntylä, and Dana Nau, editors, *Advances in Feature Based Manufacturing*, pages 129–157. Elsevier/North Holland, 1994.

[170] Jami Shah, Martti Mäntylä, and Dana Nau, editors. *Advances in Feature Based Manufacturing*. Elsevier/North Holland, 1994.

[171] Jami Shah, Palat Sreevalsan, Mary Rogers, Rick Billo, and Abraham Mathew. Current status of features technology. Technical Report R-88-GM-04.1, Computer Aided Manufacturing–International, Arlington, TX, USA, November 1988.

[172] Jami J. Shah. Assessment of features technology. *Computer Aided Design*, 23(5):331–343, June 1991.

[173] Jami J. Shah. Conceptual development of form features and feature modelers. *Research in Engineering Design*, 3:93–108, 1991.

[174] Jami J. Shah. Experimental investigation of the STEP form-feature information model. *Computer Aided Design*, 23(4):282–296, May 1991.

[175] Shape Data Limited and Electronic Data Systems Corporation, Parker's House 46, Regent Street, Cambridge CB2 1DP England. *Parasolid v5.0 Functional Description*, 1992.

[176] Shape Data Limited and Electronic Data Systems Corporation, Parker's House 46, Regent Street, Cambridge CB2 1DP England. *Parasolid v5.0 Programming Reference Manual*, 1992.

[177] Spatial Technology Inc., Boulder, CO. *ACIS© Geometric Modeler*, 1993. Version 1.4.1.

[178] Spatial Technology Inc., 2425 55th Street, Building A, Boulder, CO 80301. *ACIS Geometric Modeler Test Harness User's Guide*, v1.6 edition, November 1994.

[179] Spatial Technology Inc., 2425 55th Street, Building A, Boulder, CO 80301. *ACIS 3D Toolkit Technical Overview*, 1995.

[180] Spatial Technology Inc. and 3D/EYE Inc., 2425 55th Street, Building A, Boulder, CO 80301. *ACIS 3D Toolkit API Reference*, v1.2 edition, November 1994.

[181] Spatial Technology Inc. and 3D/EYE Inc., 2425 55th Street, Building A, Boulder, CO 80301. *ACIS 3D Toolkit Application Guide*, v1.2 edition, November 1994.

[182] Spatial Technology Inc. and 3D/EYE Inc., 2425 55th Street, Building A, Boulder, CO 80301. *ACIS 3D Toolkit Programmer's Reference*, v1.2 edition, November 1994.

[183] Spatial Technology Inc., Three-Space Ltd., and Applied Geometry Corp., 2425 55th Street, Building A, Boulder, CO 80301. *ACIS Geometric Modeler API Reference*, v1.6 edition, November 1994.

[184] Spatial Technology Inc., Three-Space Ltd., and Applied Geometry Corp., 2425 55th Street, Building A, Boulder, CO 80301. *ACIS Geometric Modeler Application Guide*, v1.6 edition, November 1994.

[185] Spatial Technology Inc., Three-Space Ltd., and Applied Geometry Corp., 2425 55th Street, Building A, Boulder, CO 80301. *ACIS Geometric Modeler Programmers Reference*, v1.6 edition, November 1994.

[186] P. Spiby. Exchange of product model data – part 11: The EXPRESS language. Technical Report ISO TC184/SC4 Document N14, International Organization for Standardization, 1991. April.

[187] R. Srinivasan, R. C. Liu, and K. S. Fu. Extraction of manufacturing details from geometric models. *Computers & Industrial Engineering*, 9(2):125–133, 1985.

[188] D. Strip and M. Karasick. Solid modeling on a massively parallel processor. *International Journal of Supercomputing Applications*, 6(2):175–192, Summer 1992.

[189] Bjarne Stroustrup. What is object-oriented programming? *IEEE Software*, pages 10–20, May 1988.

[190] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, second edition, 1991.

[191] Bjarne Stroustrup. A history of C++: 1979-1991. Technical report, AT& T Bell Laboratories, 1991.

[192] Malgorzata Sturgill, Elaine Cohen, and Richard F. Riesenfeld. Feature-based 3-d sketching for early stage design. In A. A. Busnaina, editor, *ASME Computers in Engineering Conference*, pages 545–552, New York, NY 10017, September 17-20, Boston, MA 1995. ASME.

[193] LeRoy E. Taylor. *Meta-Physical Product Modelling*. PhD thesis, Arizona State University, 1993.

[194] Teledyne Firth Sterling, One Teledyne Place, La Vergne, TN 37086, 1991. MP-92, Catalog #224, Rev. 11/91.

[195] William P. Thurston. Conway's tiling groups. *American Mathematematics Monthly*, pages 757–773, October 1990.

[196] Sanjeev N. Trika and Rangasami L. Kashyap. Geometric reasoning for extraction of manufacturing features in iso-oriented polyhedrons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(11):1087–1100, November 1994.

[197] Amjad Umar. *Distributed Computing: A Practical Synthesis*. Prentice-Hall, Englewood Cliffs, NJ 07632, 1993.

[198] F. J. A. M. van Houten. *PART: A Computer Aided Process Planning System.* PhD thesis, University of Twente, 1991.

[199] J. Van Maanen. Product data representation and exchange – part 21: Clear text encoding of the exchange structure. Technical Report ISO TC184/SC4 Document N78, International Organization for Standardization, 1991. March.

[200] J. H. Vandenbrande and A. A. G. Requicha. Spatial reasoning for the automatic recognition of machinable features in solid models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(12):1269–1285, December 1993.

[201] Jan H. Vandenbrande. *Automatic Recognition of Machinable Features in Solid Models.* PhD thesis, University of Rochester, Rochester, NY, USA, 1990.

[202] Jan H. Vandenbrande and Aristides A. Requicha. Spatial reasoning for automatic recognition of interacting form features. In *ASME Computers in Engineering Conference*, August 1990.

[203] Jan H. Vandenbrande and Aristides A. Requicha. Geometric computations for the recognition of spatially interacting manufacturing features. In Jami Shah, Martti Mäntylä, and Dana Nau, editors, *Advances in Feature Based Manufacturing.* Elsevier/North Holland, 1994.

[204] Douglas L. Waco and Yong Se Kim. Geometric reasoning for machining features using convex decomposition. *Computer Aided Design*, 26(6):477–489, June 1994.

[205] Scott Wallace. Accelerating engineering design. *Byte*, July 1994.

[206] H. P. Wang and J. K. Li. *Computer Aided Process Planning.* Elsevier Science Publishers, 1991.

[207] Kevin Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, January 1985.

[208] Tony C. Woo. Feature extraction by volume decomposition. In *Conference on CAD/CAM Technology in Mechanical Engineering*, pages 76–94, March 1982.

[209] Tony C. Woo. A combinatorial analysis of boundary data structure schemata. *IEEE Computer Graphics and Applications*, pages 19–27, March 1985.

[210] J. R. Woodwark. Some speculations on feature recognition. *Computer Aided Design*, 20(4):189–196, May 1988.

[211] J. R. Woodwark, editor. *Geometric Reasoning.* Clarendon Press, Oxford, UK, 1989.